Resource-aware Deployment, Configuration and Adaptation for Fault-tolerance in Distributed Real-time Embedded Systems

Prof. Aniruddha Gokhale
a.gokhale@vanderbilt.edu
www.dre.vanderbilt.edu/~gokhale

With contributions from
Jaiganesh Balasubramanian, Sumant Tambe and Friedhelm Wolf

Department of Electrical Engineering & Computer Science
Vanderbilt University, Nashville, TN, USA

Work supported in part by DARPA PCES and ARMS programs, and NSF CAREER and NSF SHF/CNS Awards

Objectives for this Tutorial

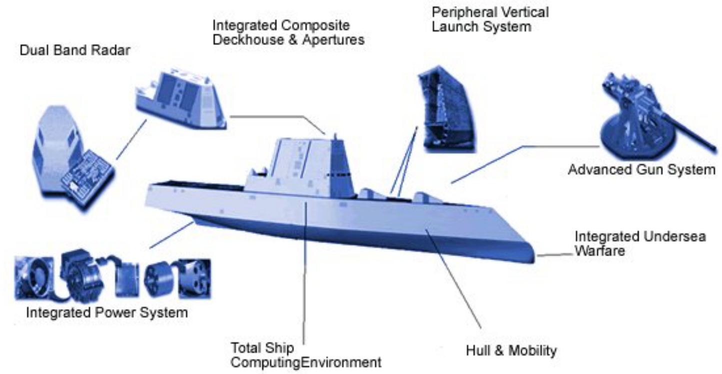
- To showcase research ideas from academia
- To demonstrate how these ideas can be realized using OMG standardized technologies
- To illustrate how the resulting artifacts can be integrated within existing industry development processes for large, service-oriented architectures
- To facilitate discussion on additional real-world use cases and further need for research on unresolved issues

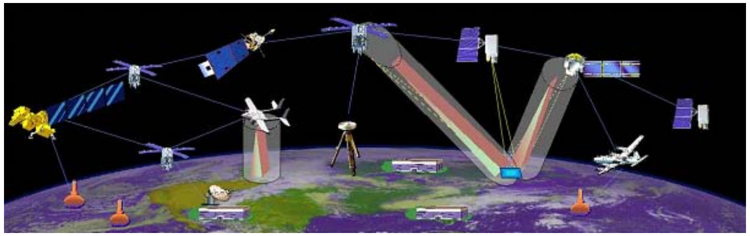
Presentation Road Map

- **Technology Context: DRE Systems**
- DRE System Lifecycle & FT-RT Challenges
- Design-time Solutions
- Deployment & Configuration-time Solutions
- Runtime Solutions
- Ongoing Work
- Concluding Remarks

Context: Distributed Real-time Embedded (DRE) Systems

- Heterogeneous soft real-time applications
- Stringent simultaneous QoS demands
  - High availability, Predictability (CPU & network) etc
  - Efficient resource utilization
- Operation in dynamic & resource-constrained environments
  - Process/processor failures
  - Changing system loads
- Examples
  - Total shipboard computing environment
  - NASA's Magnetospheric Multi-scale mission
  - Warehouse Inventory Tracking Systems
- Component-based application model used due to benefits stemming from:
  - Separation of concerns
  - Composability
  - Reuse of commodity-off-the-shelf (COTS) components

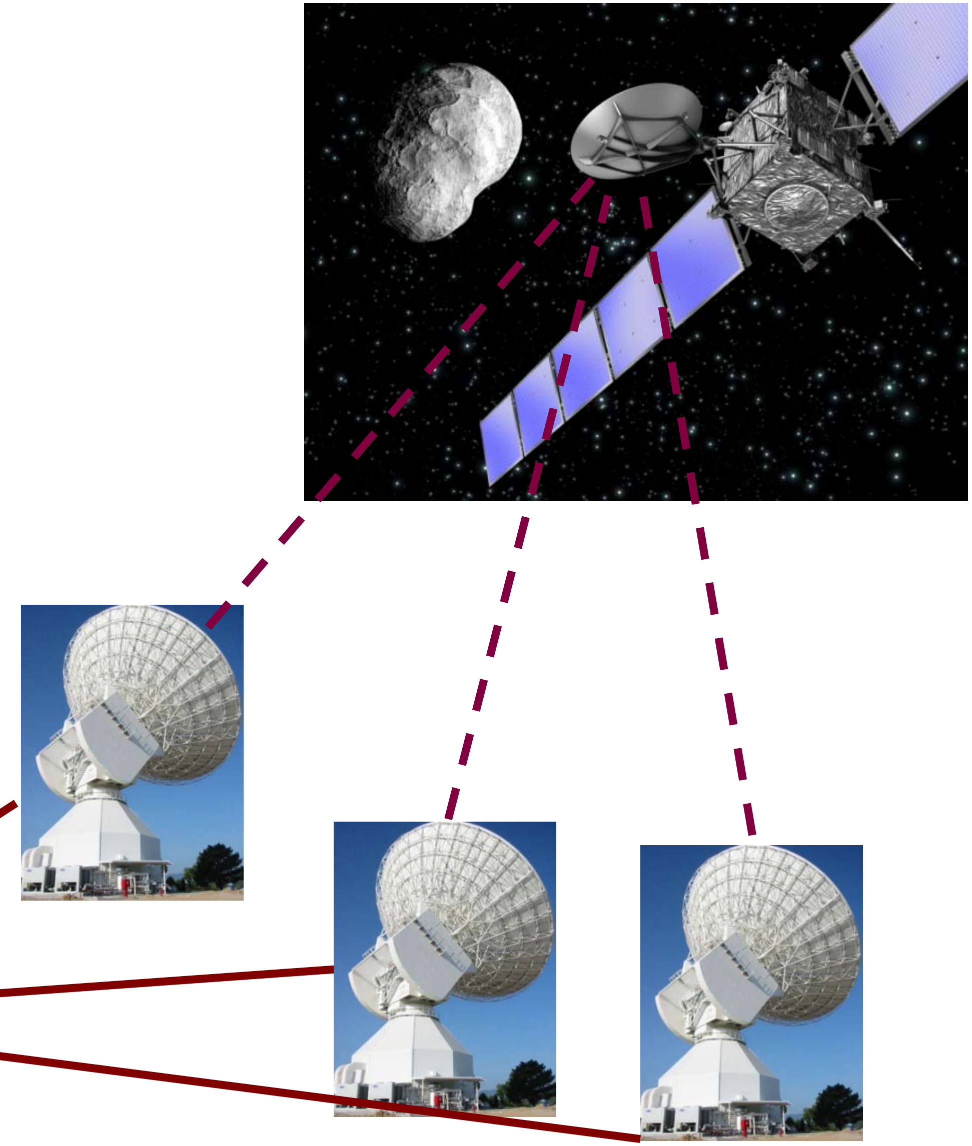




(Images courtesy Google)

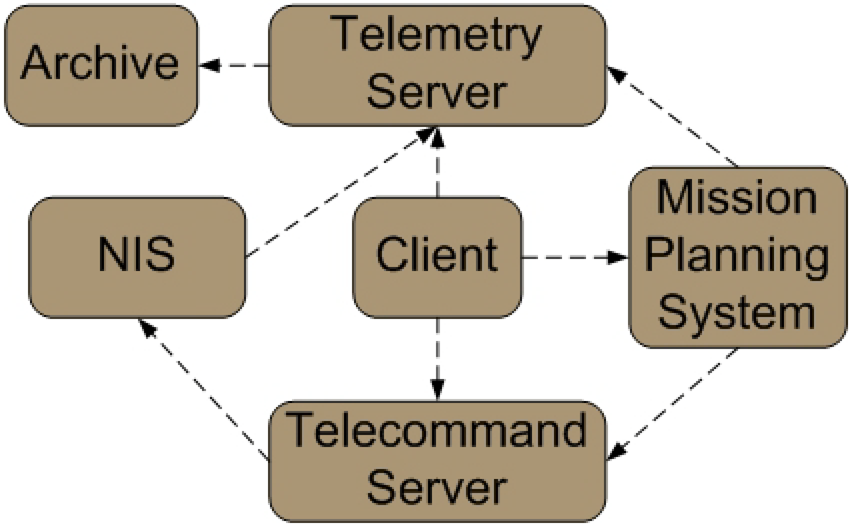
Motivating Case Study

- Mission Control System of the European Space Agency (ESA)
  - Short connection windows
  - No physical access to the satellites
  - Software must not crash
  - Very heterogeneous infrastructure
  - Must ensure correctness of data



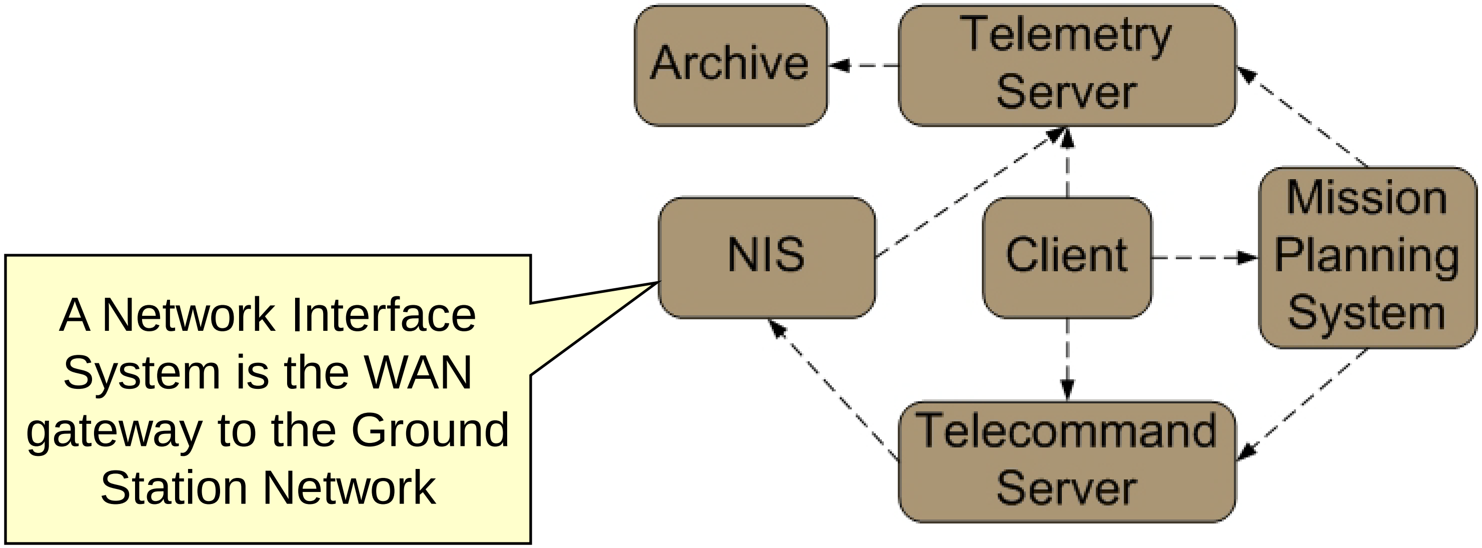
Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



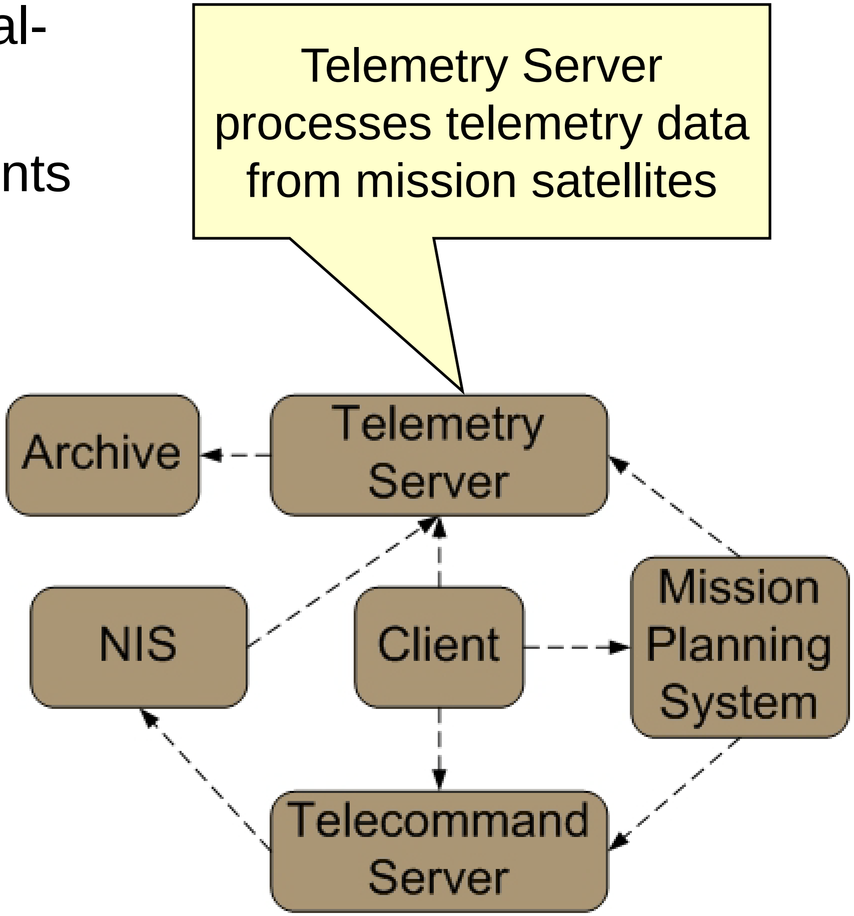
Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



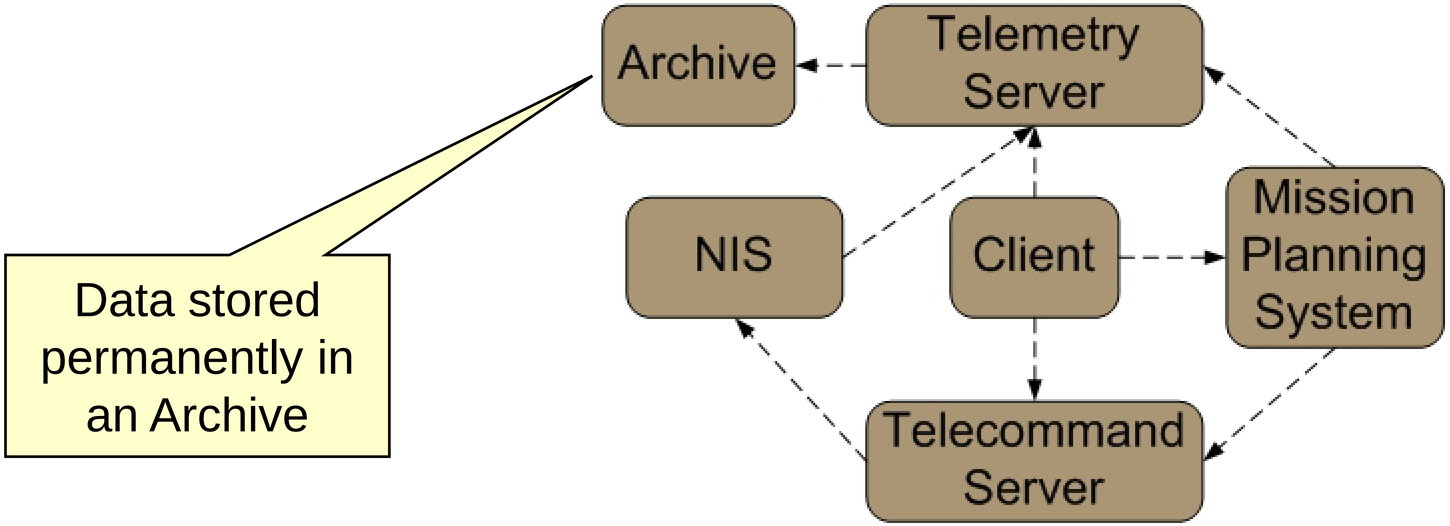
Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



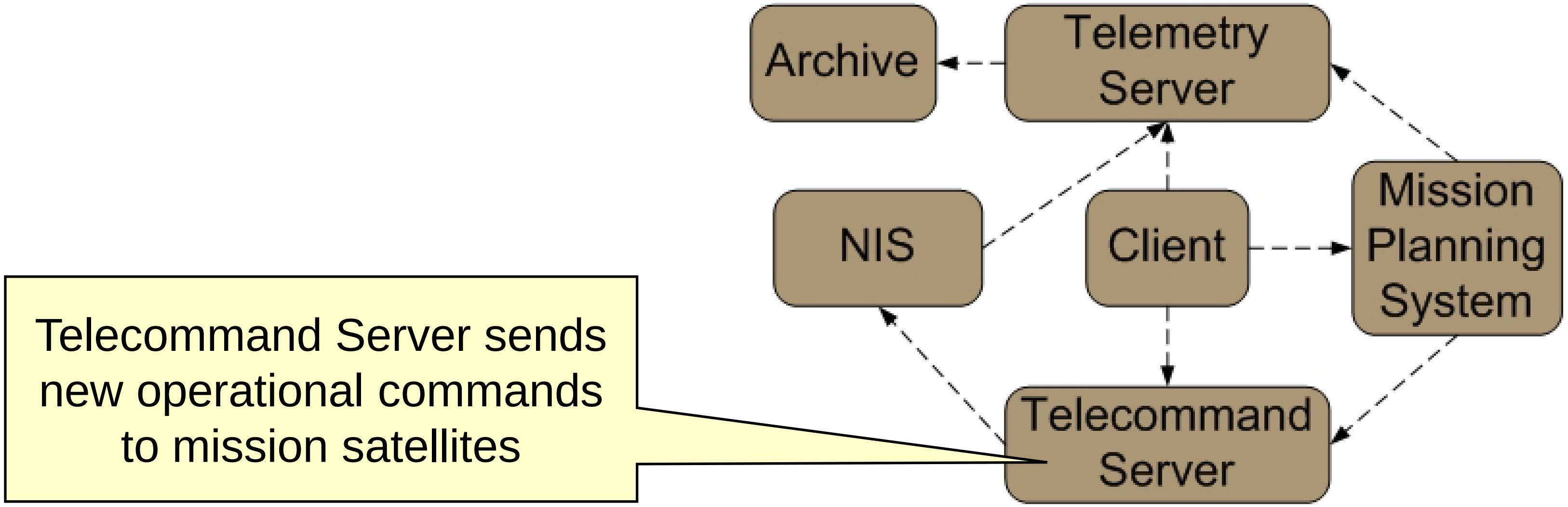
Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



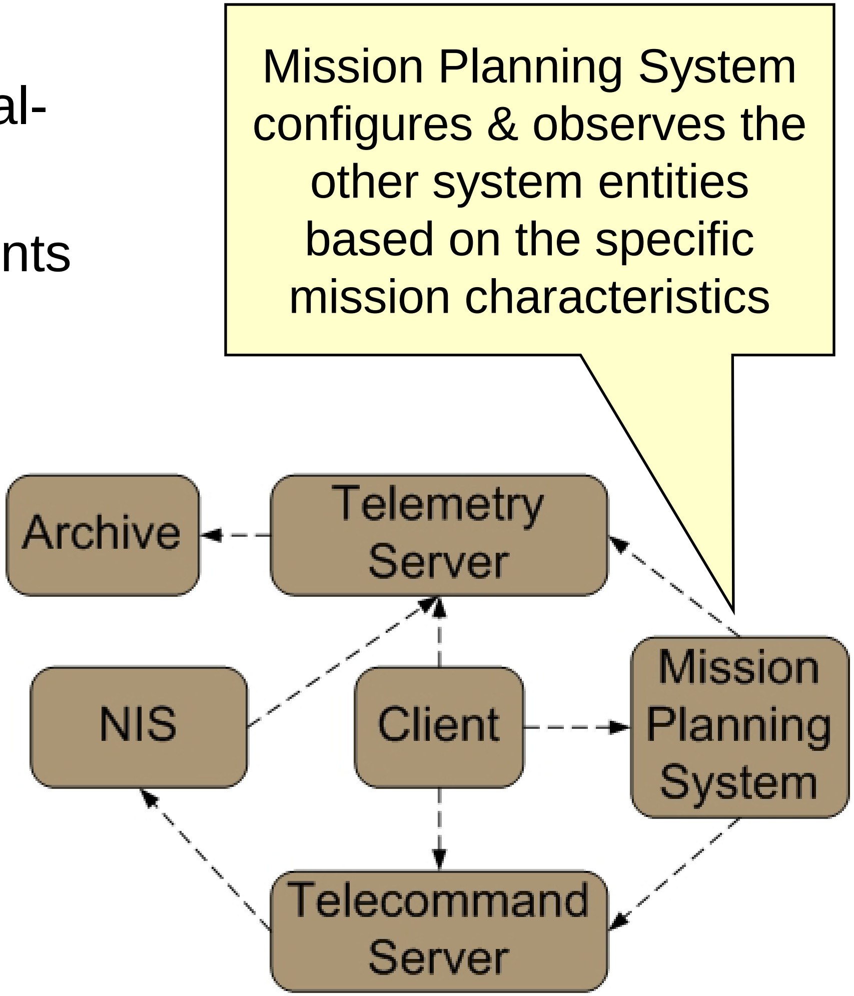
Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



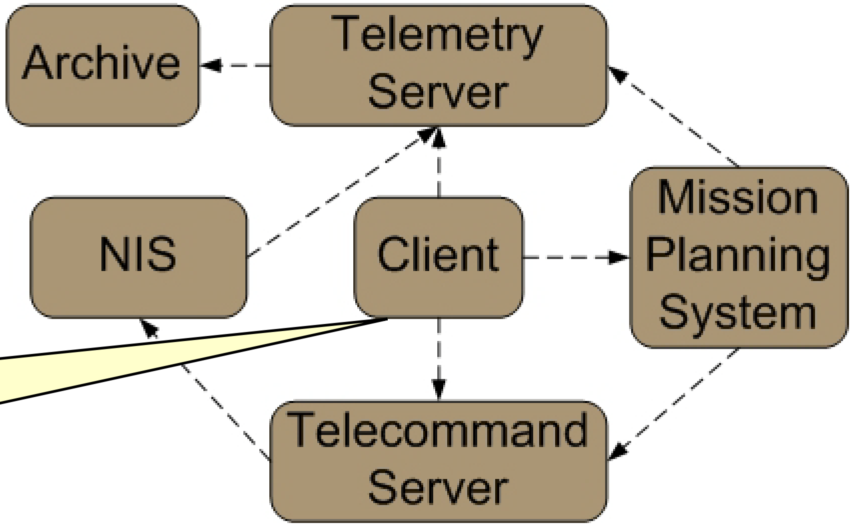
Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



Case Study: ESA Mission Control System

- Mission Control Systems are the central means for control & observations of space missions
- Simultaneous operations of multiple real-time applications
- Stringent simultaneous QoS requirements
  - e.g., high availability & satisfactory average response times



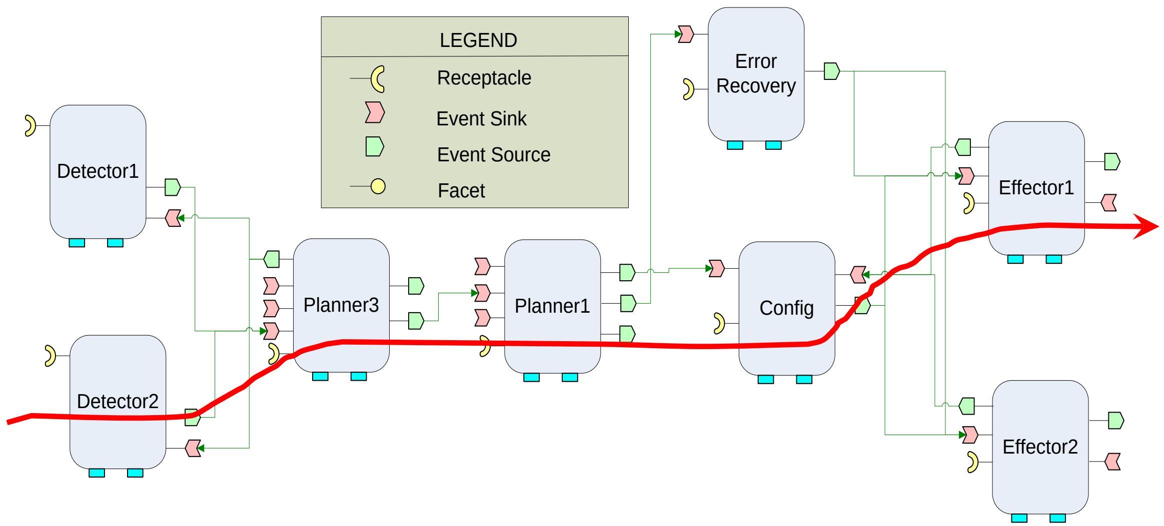
Client access, such as an operator GUI, needs to interact with several components

Presentation Road Map

- Technology Context: DRE Systems
- **DRE System Lifecycle & FT-RT Challenges**
- Design-time Solutions
- Deployment & Configuration-time Solutions
- Runtime Solutions
- Ongoing Work
- Concluding Remarks

Component-based Design of DRE Systems

- **Operational String model** of component-based DRE systems
  - A multi-tier processing model focused on the end-to-end QoS requirements
  - Functionality is a **chain of tasks** scheduled on a pool of computing nodes
  - Resources, QoS, & deployment are managed end-to-end
- **End-to-end QoS** requirements
  - **Critical Path:** The chain of tasks that is time-critical from source to destination
  - Need predictable scheduling of computing resources across components
  - Need network bandwidth reservations to ensure timely packet delivery
  - Failures may compromise end-to-end QoS



Must support highly available operational strings!

A Perspective of Component-based DRE System Lifecycle

QoS (e.g. FT) provisioning should be integrated within this lifecycle

Development Lifecycle

- Specification
- Composition
- Deployment
- Configuration
- Run-time

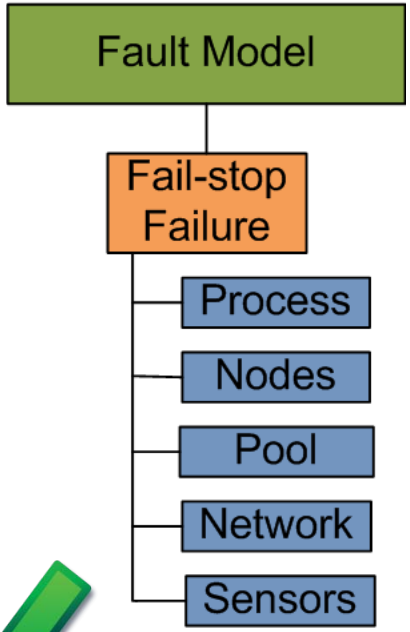
Specification:
- Gathering and specifying functional and non functional requirements of the system

Composition, Deployment, Configuration:
- Defining the operational strings through component composition
- Deploying components onto computing nodes
- Configuring the hosting infrastructure to support desired QoS properties

Run-time:
- Mechanisms to provide real-time fault recovery
- Mechanisms to deal with the side effects of replication & non-determinism at run-time

Specification: Fault Tolerance Criteria (1/4)

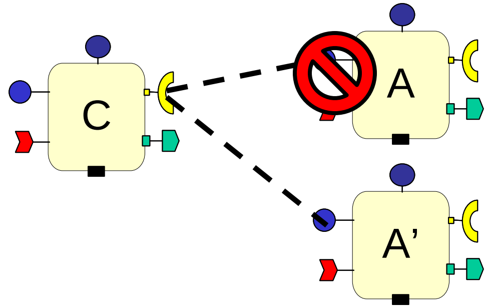
- The fault-model consists of fail-stop failures
  - Cause delays & requires software/hardware redundancy
  - Recovery must be quick to meet the deadline (soft real-time)
- What are reliability alternatives?
  - Roll-back recovery
    - Transactional
  - Roll-forward recovery: replication schemes
    - Active replication (multiple concurrent executions)
    - Passive replication (primary-backup approach)

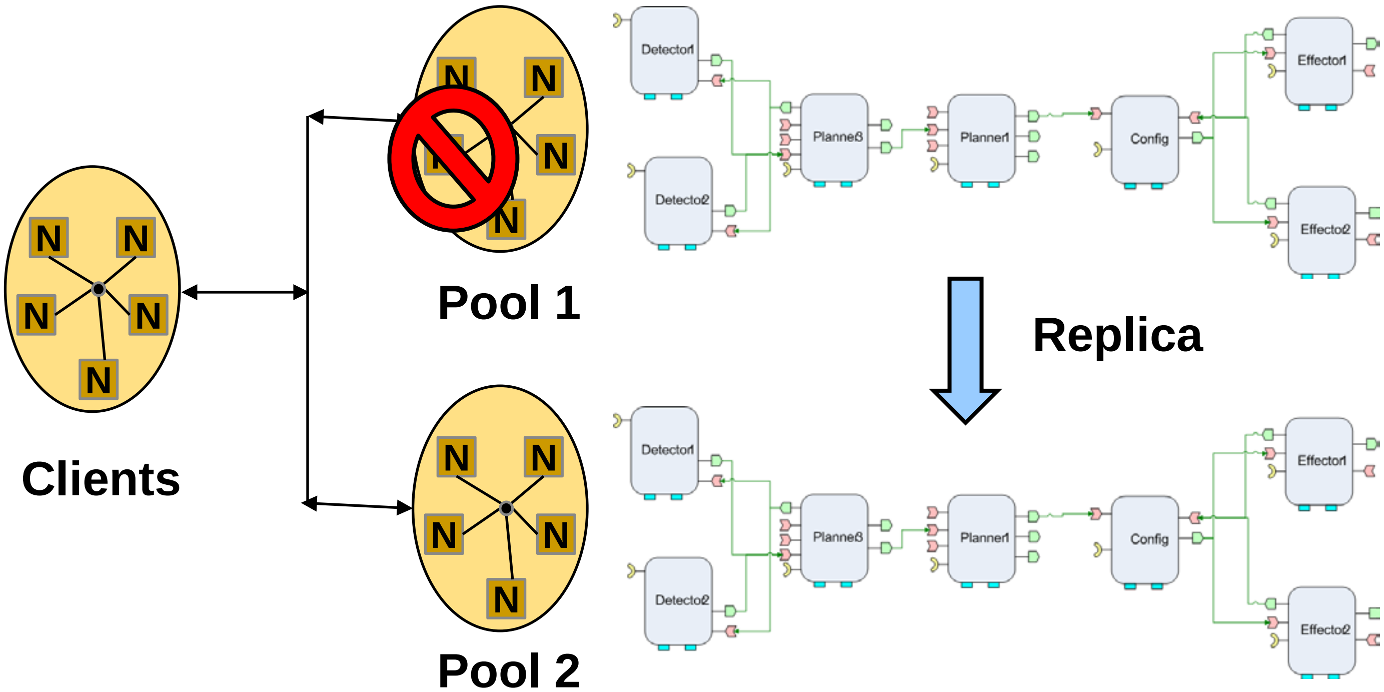


	Roll-back recovery	Active Replication	Passive Replication
Resources	Needs transaction support (heavy-weight)	Resource hungry (compute & network)	Less resource consuming than active (only network)
Non-determinism	Must compensate non-determinism	Must enforce determinism	Handles non-determinism better
Recovery time	Roll-back & re-execution (slowest recovery)	Fastest recovery	Re-execution (slower recovery)

Specification: Fault Tolerance Criteria (2/4)

- What is failover granularity for passive replication?
  - Single component failover only? or
  - Larger than a single component?
- **Scenario 1:** Must tolerate catastrophic faults
  - e.g., data center failure, network failure

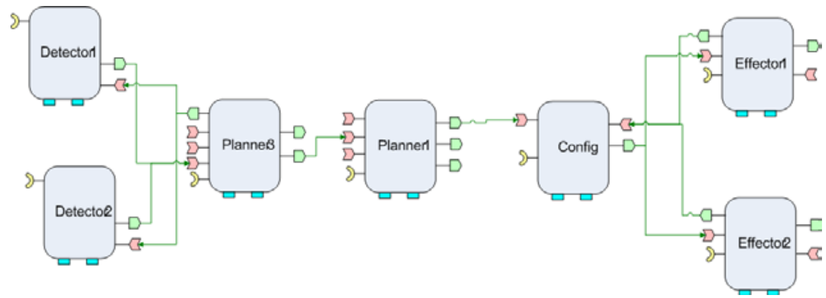




Whole operational string must failover

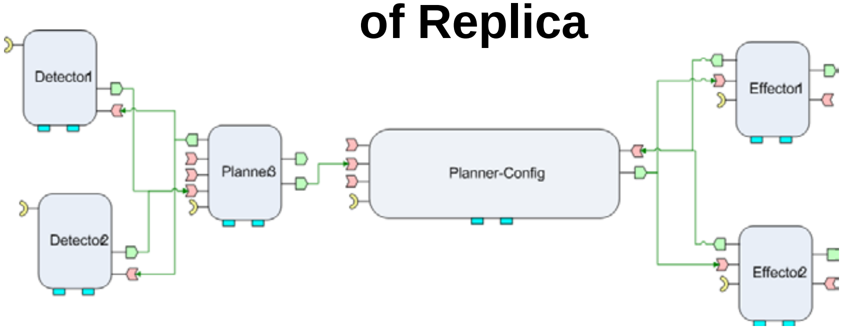
Specification: Fault Tolerance Criteria (3/4)

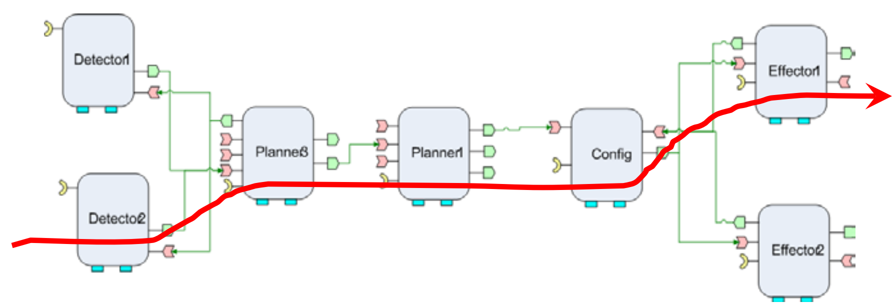
- **Scenario 2:** Must tolerate Bohrbugs
  - A Bohrbug repeats itself predictably when the same state reoccurs
- Preventing Bohrbugs by “reliability through diversity”
  - Diversity via non-isomorphic replication



Non-isomorphic work-flow and implementation of Replica

Different End-to-end QoS (thread pools, deadlines, priorities)





Whole operational string must failover

Specification: Fault Tolerance Criteria (4/4)

- **Scenario 3:** Must tolerate non-determinism
  - Sources of non-determinism in DRE systems
    - Local information (sensors, clocks), thread-scheduling, timers, timeouts, & more
  - Enforcing determinism is not always possible
- Must tolerate side-effects of replication + non-determinism
  - Problem: Orphan request & orphan state
  - Solution based on single component failover require costly roll-backks

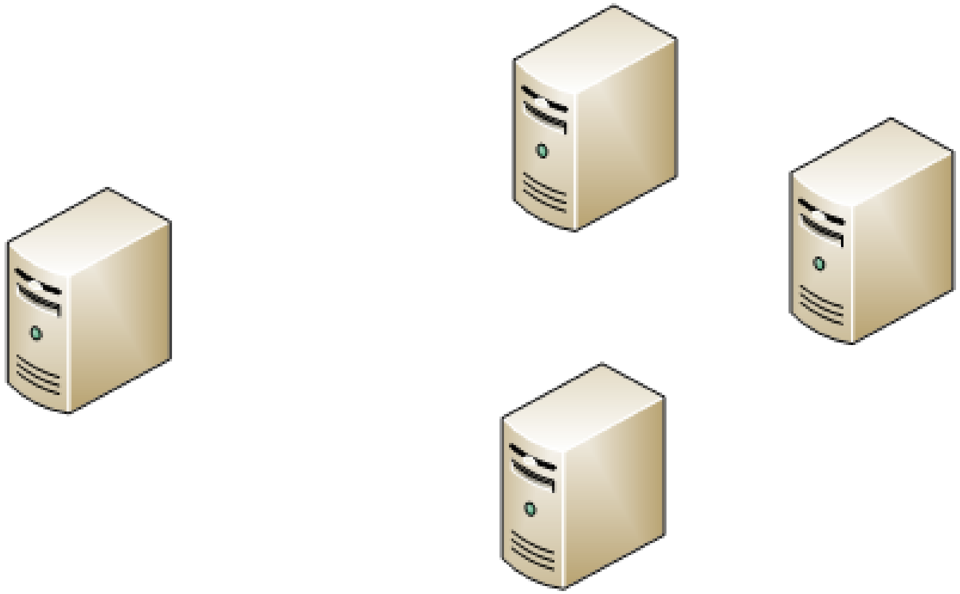
Non-determinism + Replication = Potential orphan state

- Fault-tolerance provisioning should be transparent
  - Separation of availability concerns from the business logic
  - Improves reusability, productivity, & perceived availability of the system

Need a methodology to capture these requirements and provision them for DRE systems

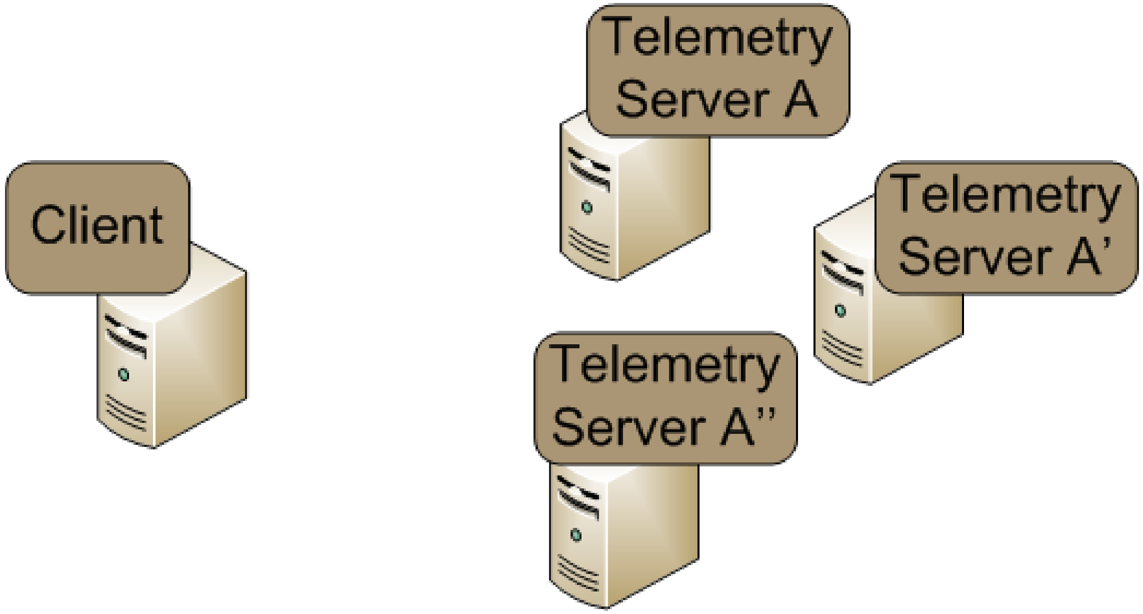
Deployment: Criteria for Fault-tolerance

- **Deployment of applications & replicas**



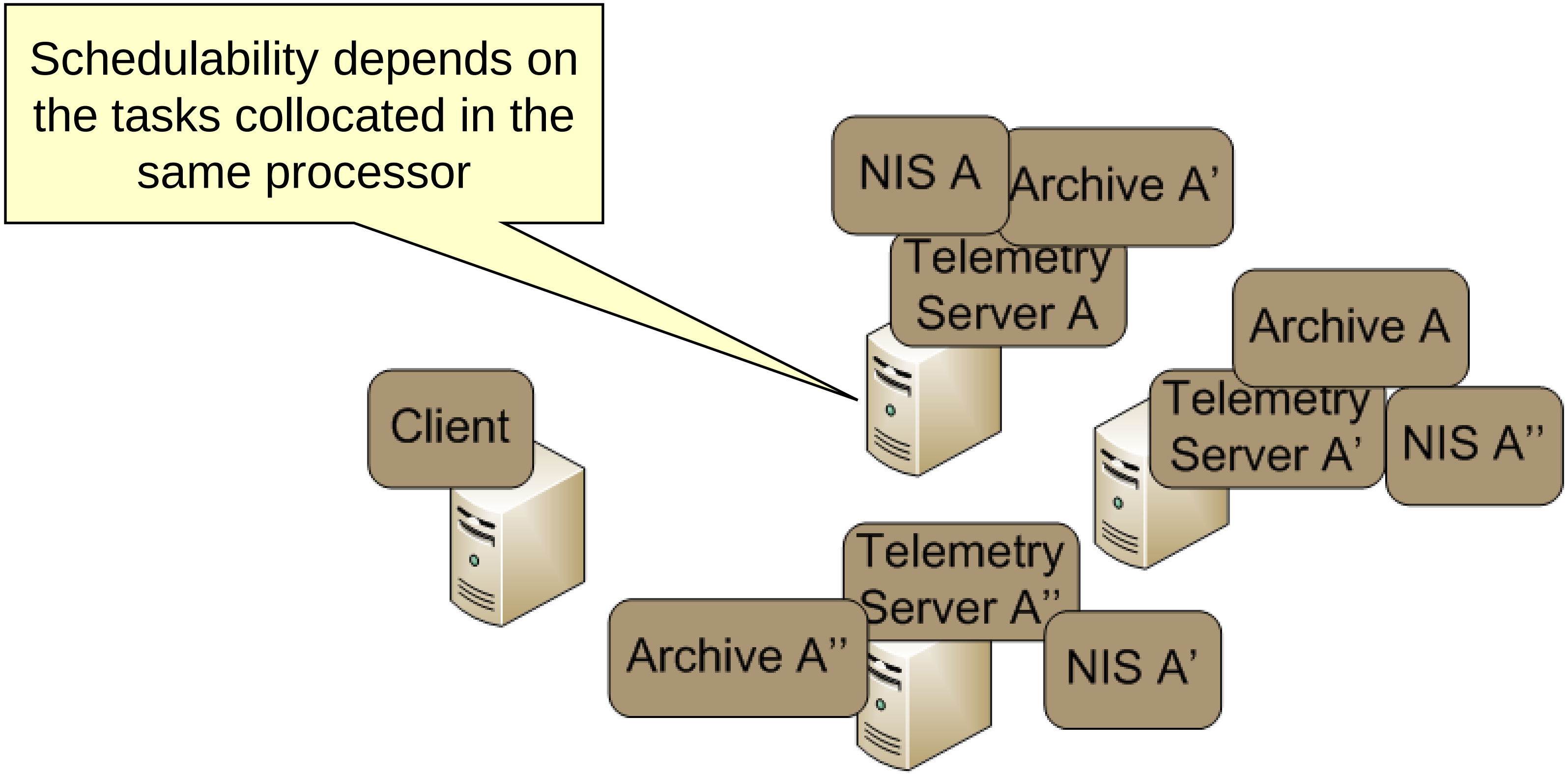
Deployment: Criteria for Fault-tolerance

- **Deployment of applications & replicas**
  - Identify different hosts for deploying applications & each of their replicas
    - no two replicas of the same application are hosted in the same processor
  - allocate resources for applications & replicas
    - deploy applications & replicas in the chosen hosts



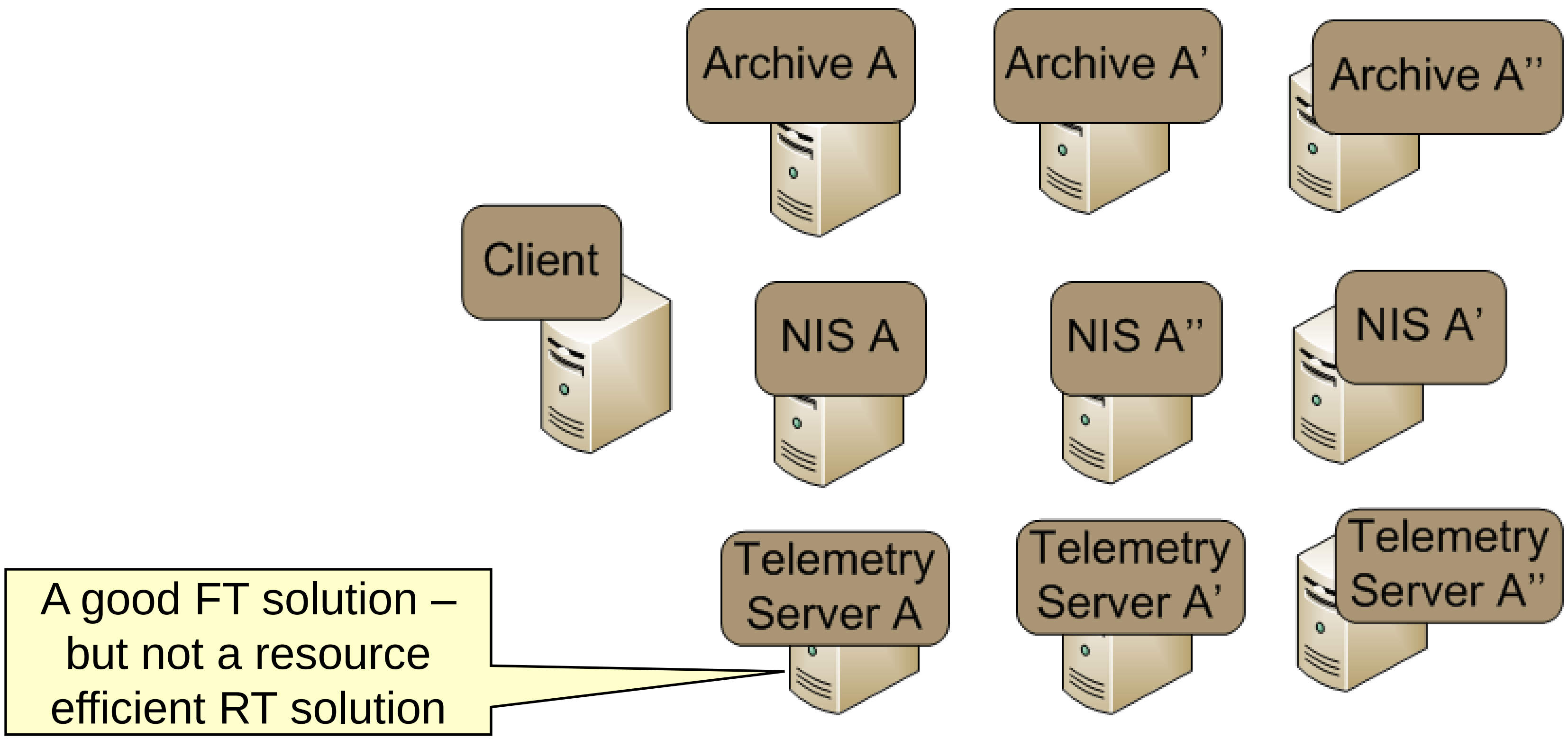
Challenges in Deployment of Fault-tolerant DRE Systems

- Ad-hoc allocation of applications & replicas could provide FT
  - could lead to resource minimization, however,
    - system might not be schedulable



Challenges in Deployment of Fault-tolerant DRE Systems

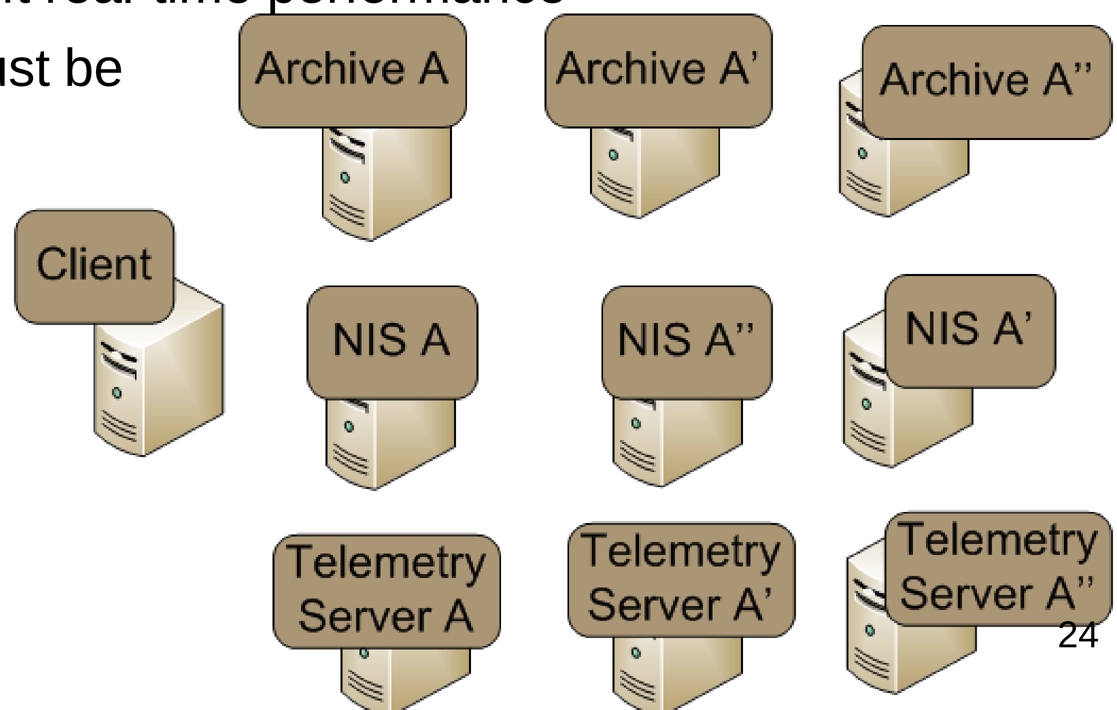
- Ad-hoc allocation of applications & replicas could provide FT
  - could lead to resource minimization, however,
    - system might not be schedulable
  - could lead to system schedulability & high availability, however,
    - could miss collocation opportunities => performance suffers
    - could cause inefficient resource utilization



Challenges in Deployment of Fault-tolerant DRE Systems

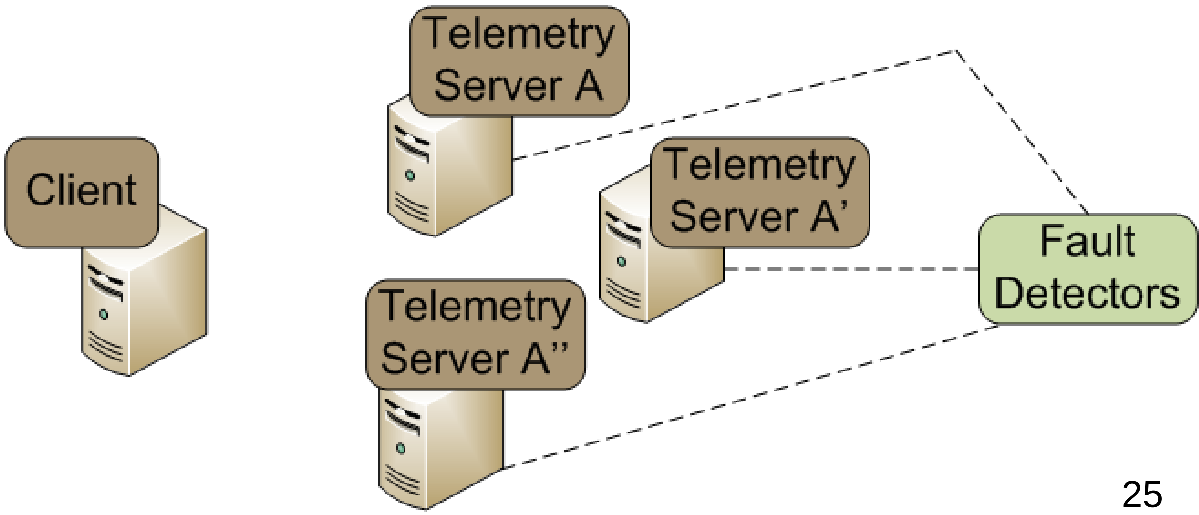
- Ad-hoc allocation of applications & replicas could provide FT
  - could lead to resource minimization, however,
    - system might not be schedulable
  - could lead to system schedulability & high availability, however,
    - could miss collocation opportunities => performance suffers
    - could cause inefficient resource utilization
- inefficient allocations – for both applications & replicas – could lead to resource imbalance & affect soft real-time performance
- applications & their replicas must be deployed in their <u>appropriate</u> physical hosts
  - ***need for resource-aware deployment techniques***

Need for Real-time, Fault-aware and Resource-aware Allocation Algorithms



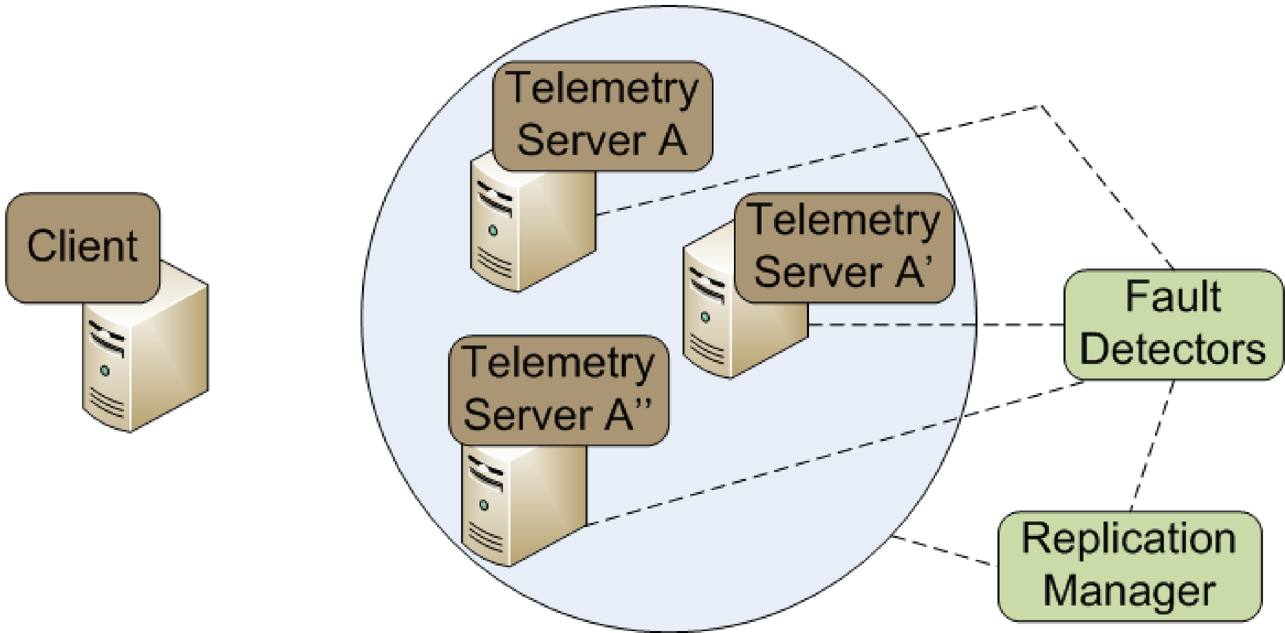
Configuration: Criteria for Fault-tolerance

- **Configuration of RT-FT Middleware**
  - Install & configure fault detectors that periodically monitor liveness on each processor



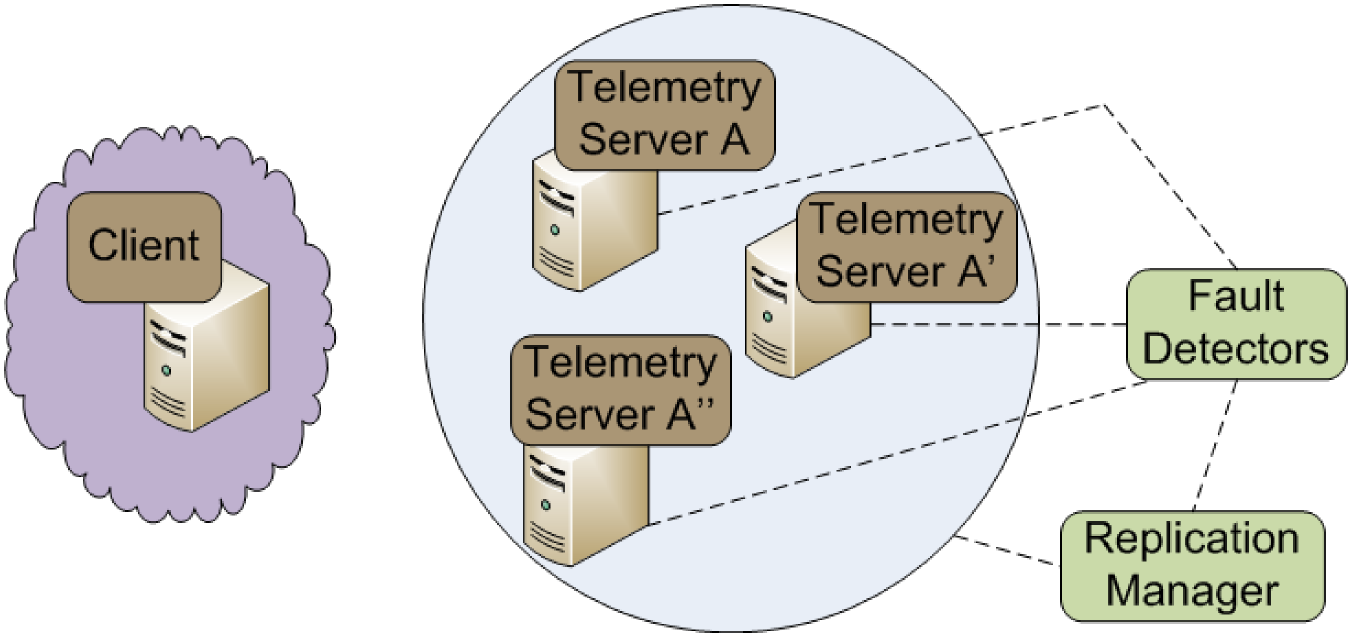
Configuration: Criteria for Fault-tolerance

- **Configuration of RT-FT Middleware**
  - Install & configure fault detectors that periodically monitor liveness on each processor
  - register all the applications, their replicas, & fault detectors with a replication manager to provide group membership management



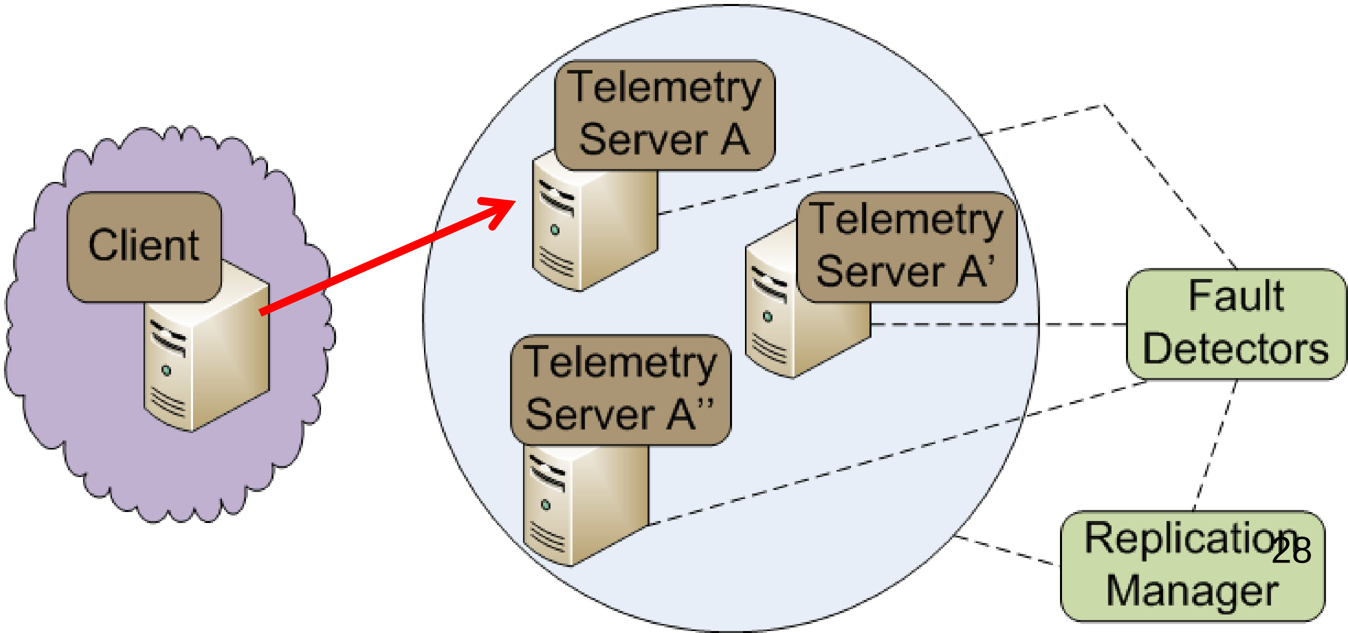
Configuration: Criteria for Fault-tolerance

- **Configuration of RT-FT Middleware**
  - Install & configure fault detectors that periodically monitor liveness on each processor
  - register all the applications, their replicas, & fault detectors with a replication manager to provide group membership management
  - configure client-side middleware to catch failure exceptions & with failure recovery actions



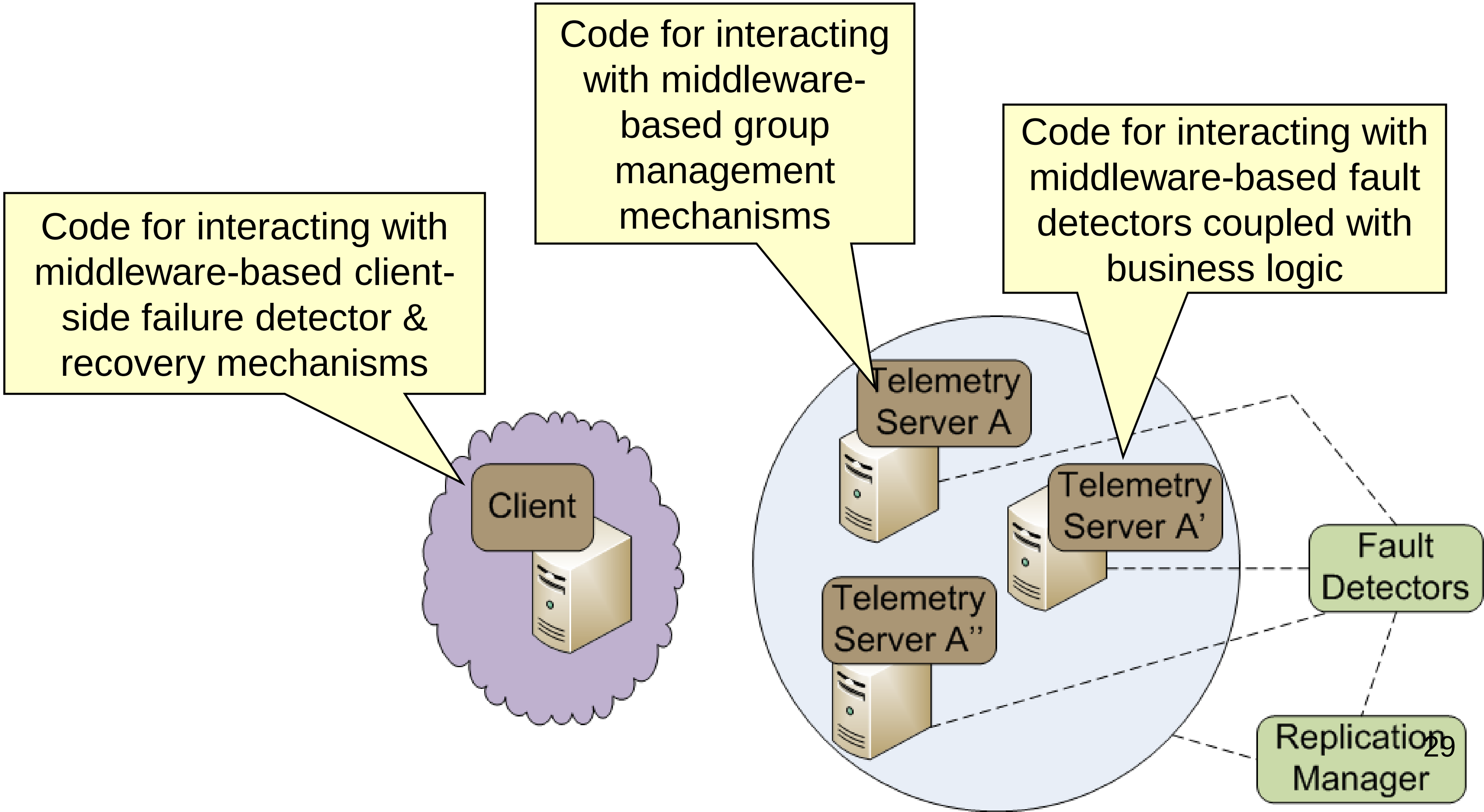
Configuration: Criteria for Fault-tolerance

- **Configuration of RT-FT Middleware**
  - Install & configure fault detectors that periodically monitor liveness on each processor
  - register all the applications, their replicas, & fault detectors with a replication manager to provide group membership management
  - configure client-side middleware to catch failure exceptions & with failure recovery actions
  - bootstrap applications



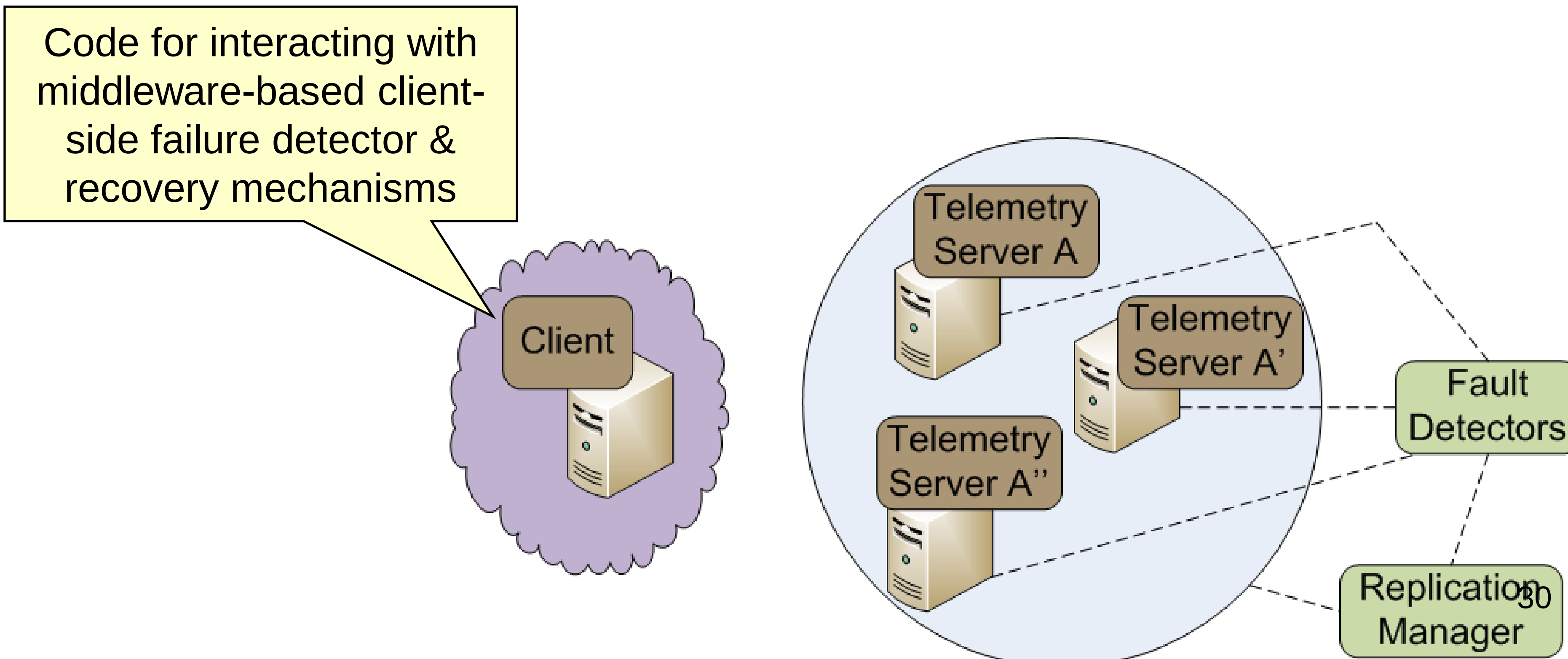
Challenges in Configuring Fault-tolerant DRE Systems

- Configuring RT-FT middleware is hard
  - developers often need to make tedious & error-prone invasive source code changes to manually configure middleware



Challenges in Configuring Fault-tolerant DRE Systems

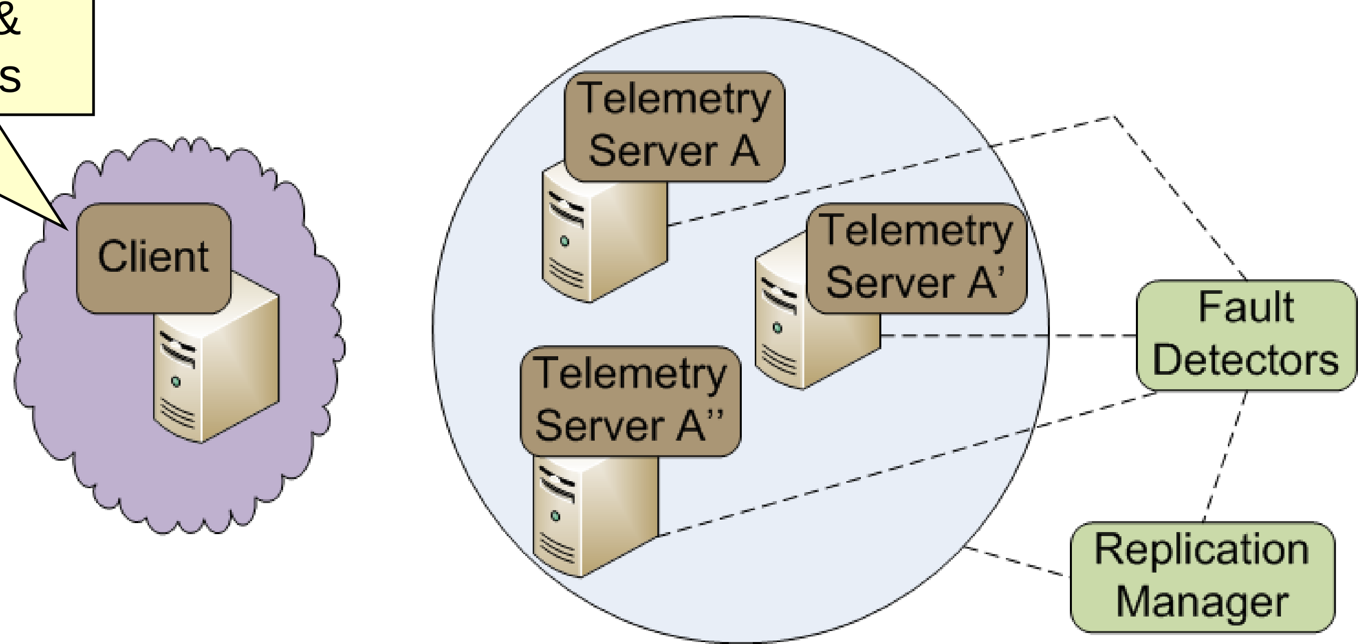
- Configuring RT-FT middleware is hard
  - developers often need to make tedious & error-prone invasive source code changes to manually configure middleware
  - manual source code modifications require knowledge of underlying middleware – which is <u>hard</u>



Challenges in Configuring Fault-tolerant DRE Systems

- Configuring RT-FT middleware is hard
  - developers often need to make tedious & error-prone invasive source code changes to manually configure middleware
  - manual source code modifications require knowledge of underlying middleware – which is <u>hard</u>
    - need to repeat configuration actions as underlying middleware changes

Code for interacting with middleware-based client-side failure detector & recovery mechanisms

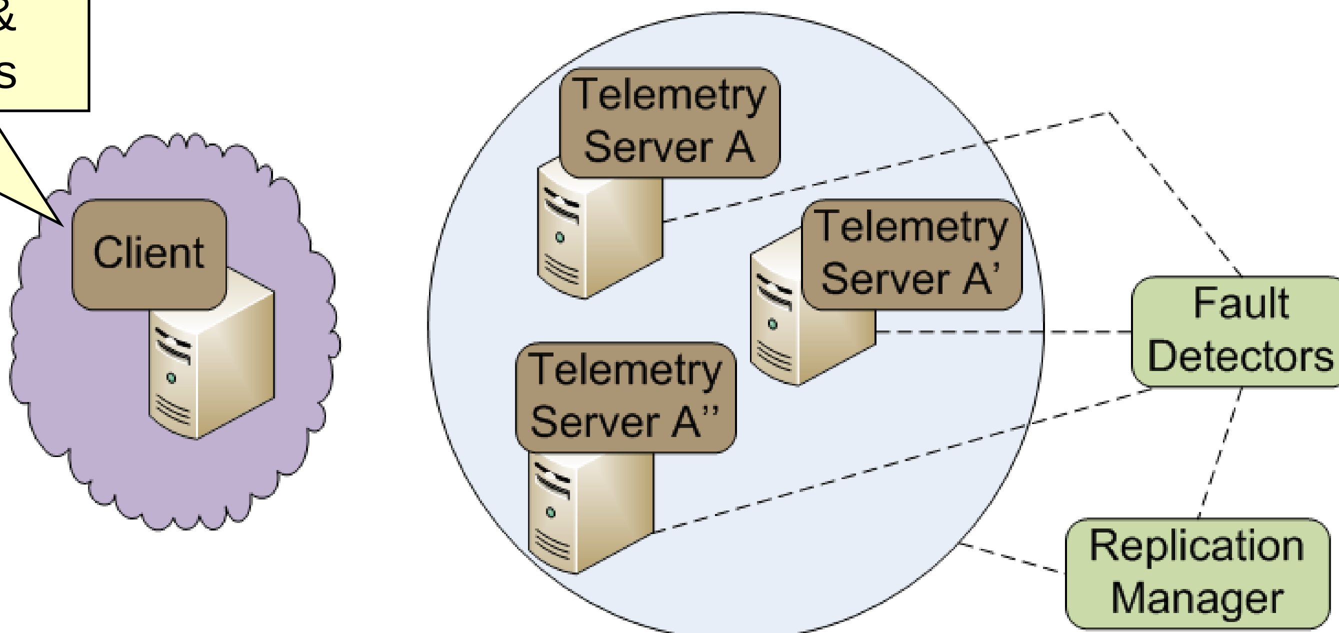


Challenges in Configuring Fault-tolerant DRE Systems

- Configuring RT-FT middleware is hard
  - developers often need to make tedious & error-prone invasive source code changes to manually configure middleware
  - manual source code modifications require knowledge of underlying middleware – which is hard
    - need to repeat configuration actions as underlying middleware changes

Scale & complexity of DRE systems make it infeasible to adopt manual techniques

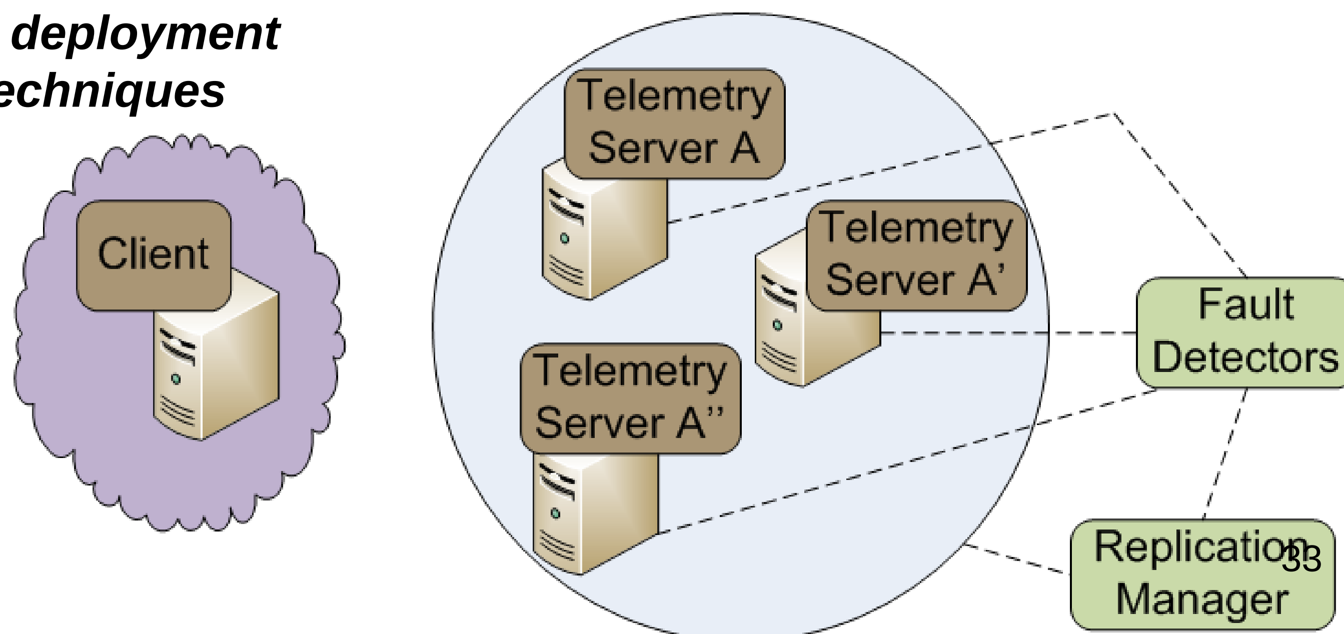
Code for interacting with middleware-based client-side failure detector & recovery mechanisms



Challenges in Configuring Fault-tolerant DRE Systems

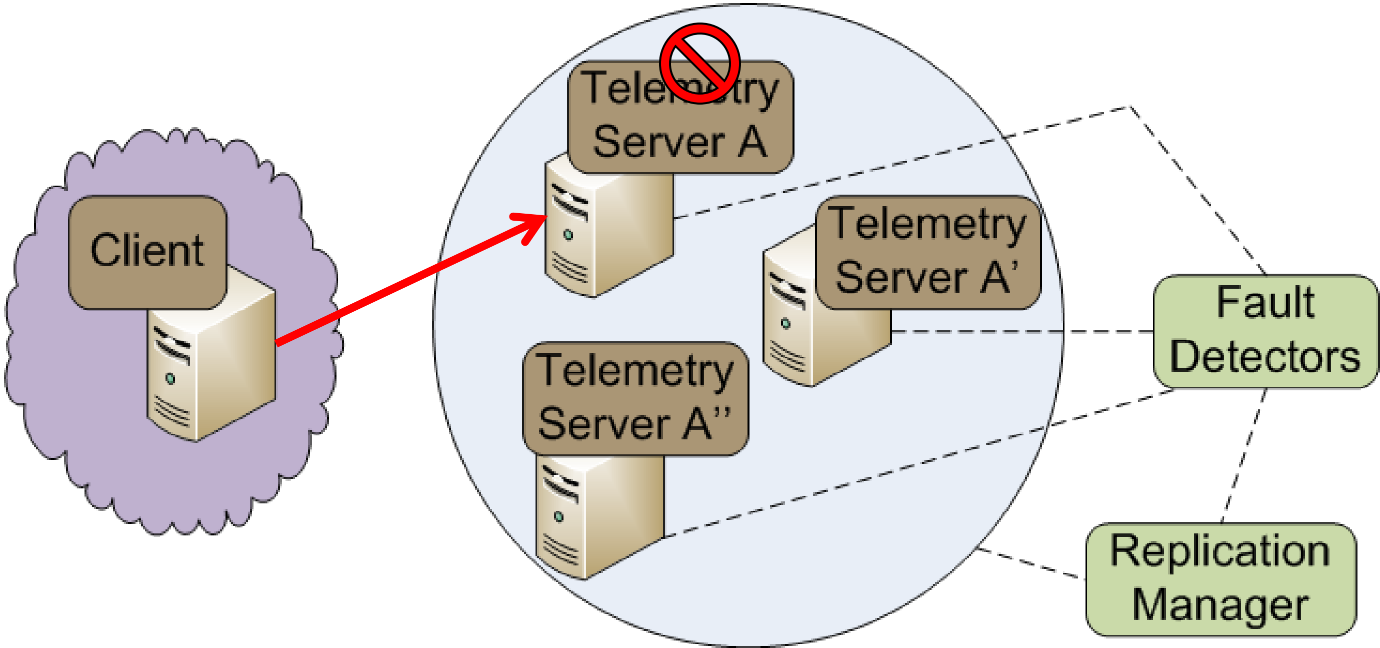
- Configuring RT-FT middleware is hard
  - developers often need to make tedious & error-prone invasive source code changes to manually configure middleware
  - manual source code modifications require knowledge of underlying middleware – which is <u>hard</u>
    - need to repeat configuration actions as underlying middleware changes
- Applications must seamlessly leverage advances in middleware mechanisms
  - QoS goals change, but business logic does not
  - ***need for scalable deployment & configuration techniques***

Need for Scalable Deployment & Configuration Middleware



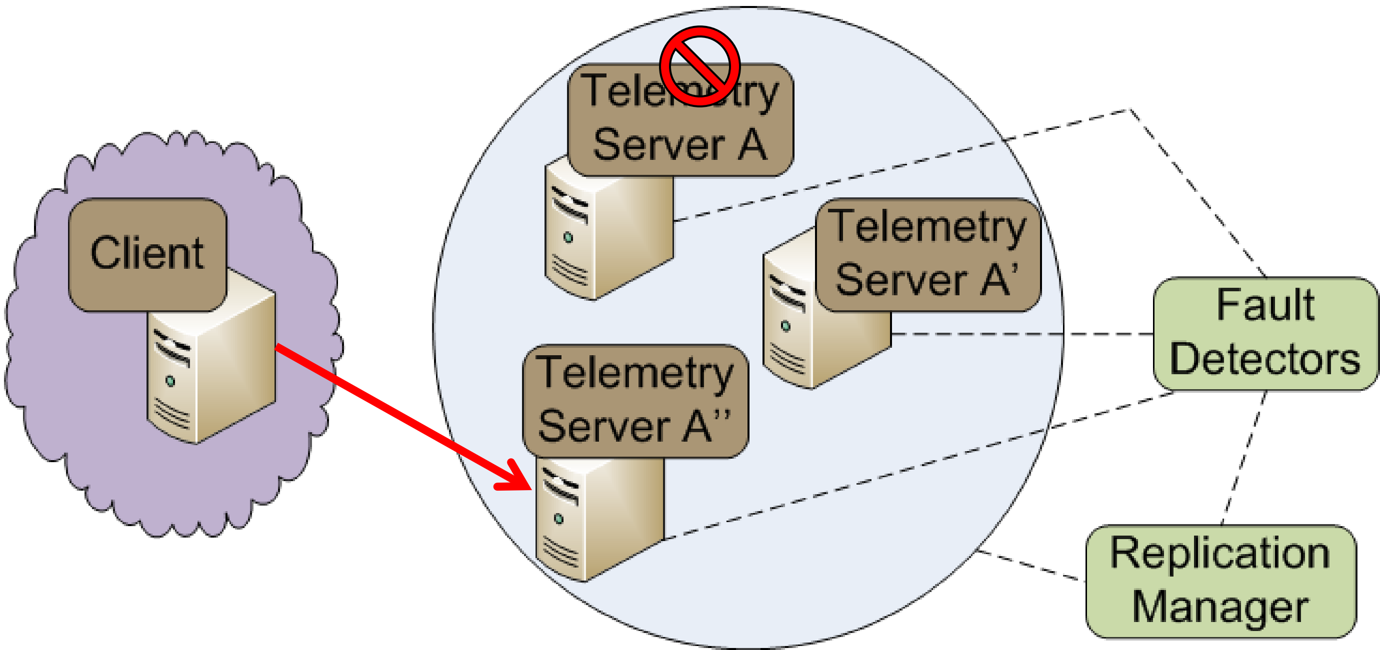
Runtime: Criteria for Fault-tolerant DRE Systems

- **Runtime management**
  - detect failures



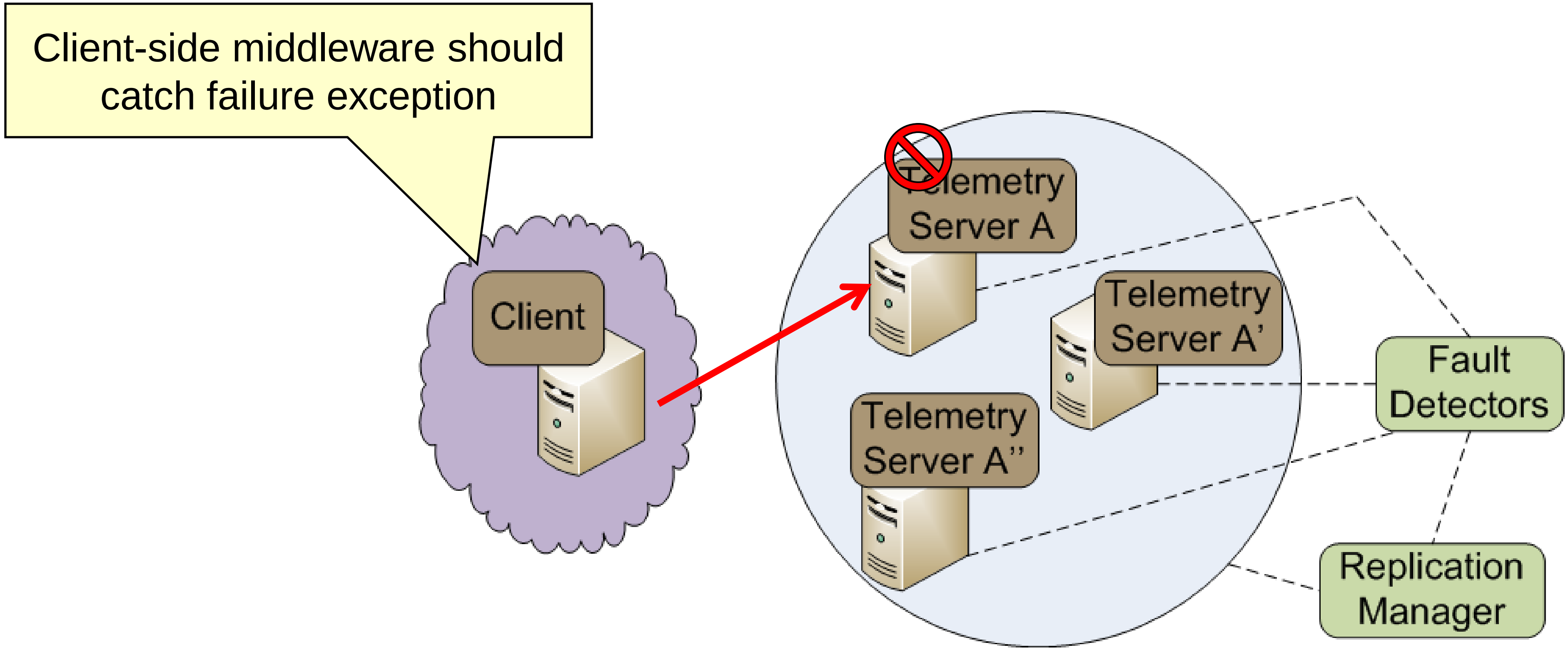
Runtime: Criteria for Fault-tolerant DRE Systems

- **Runtime management**
  - detect failures
  - transparently failover to alternate replicas & provide high availability to clients



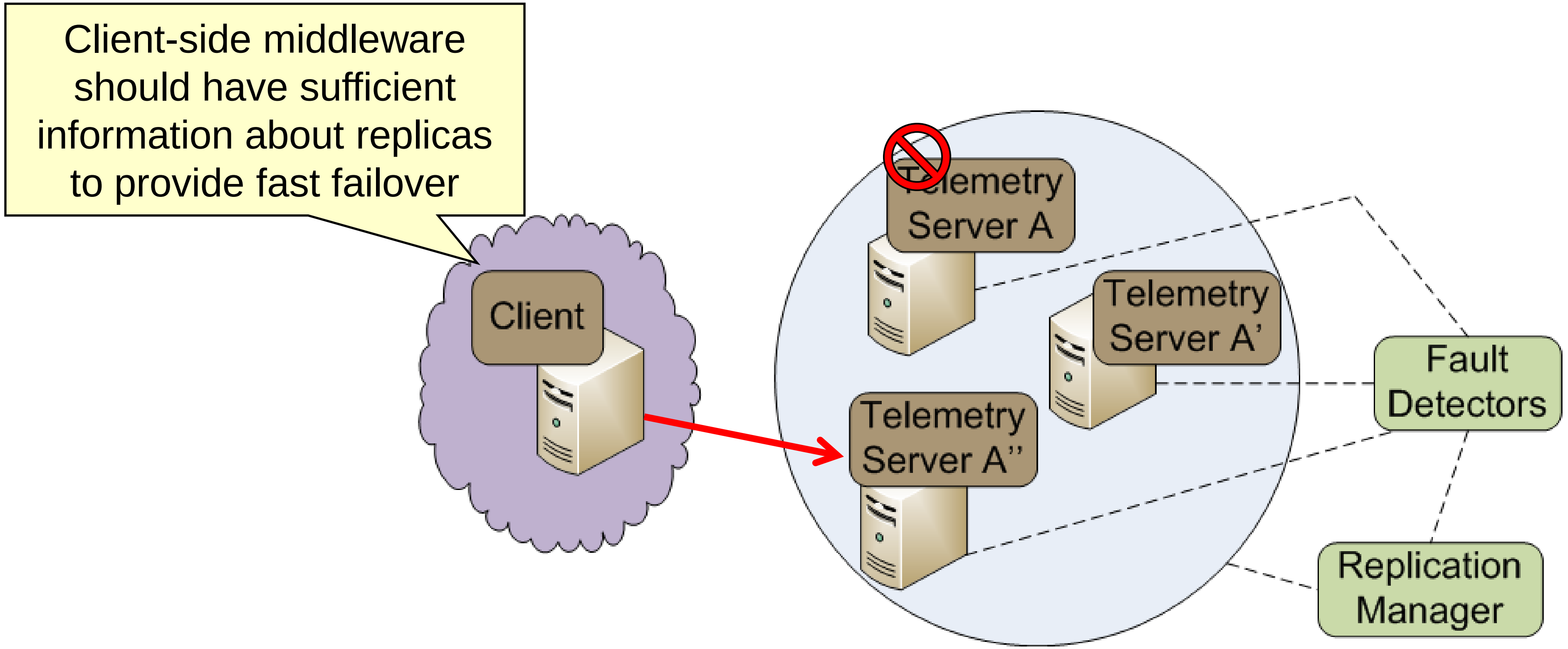
Challenges in Runtime Management of Fault-tolerant DRE Systems

- Providing high availability & soft real-time performance at runtime is hard
  - failures need to be detected quickly so that failure recovery actions can proceed



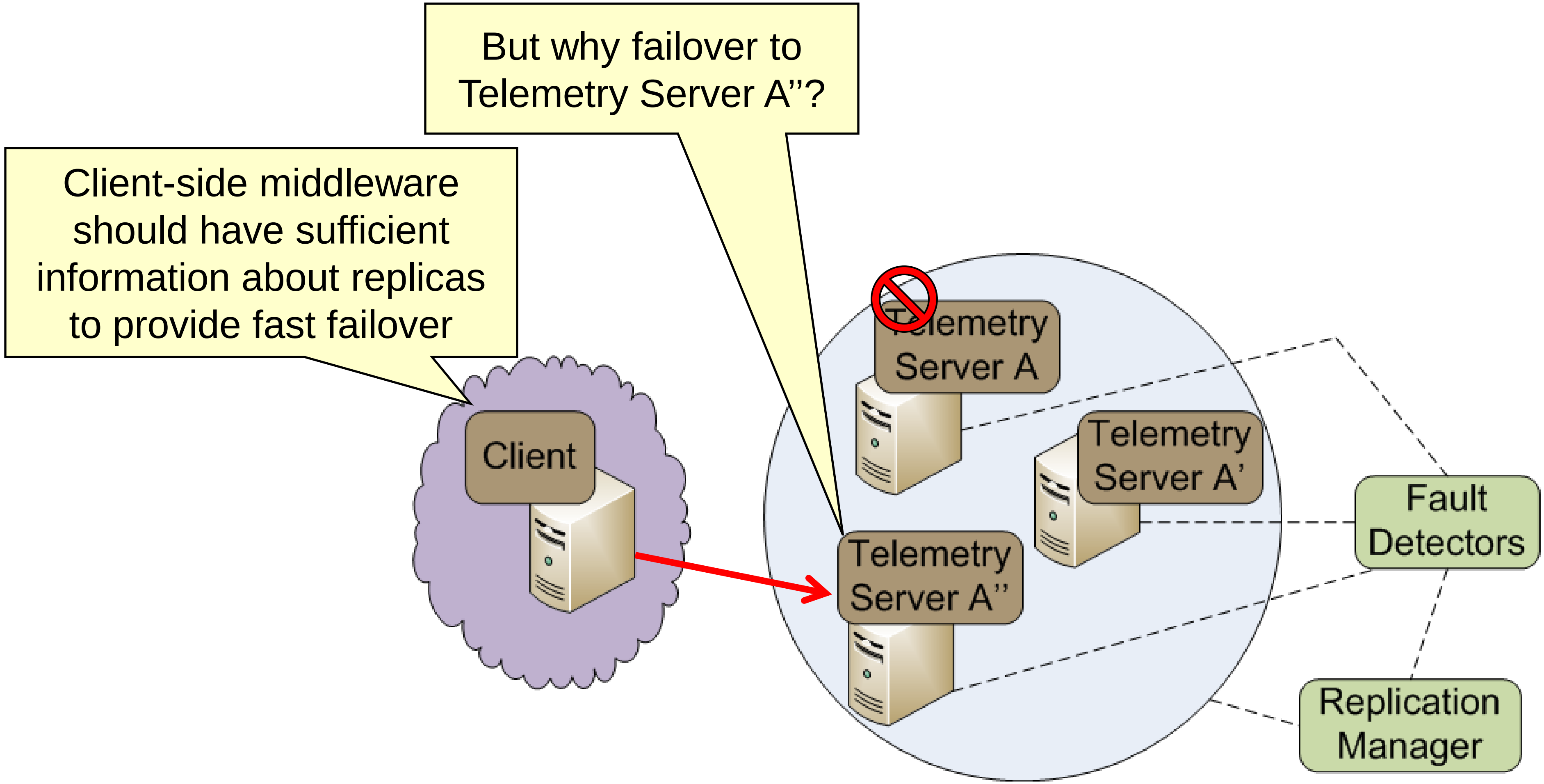
Challenges in Runtime Management of Fault-tolerant DRE Systems

- Providing high availability & soft real-time performance at runtime is hard
  - failures need to be detected quickly so that failure recovery actions can proceed
    - failure recovery should be fast



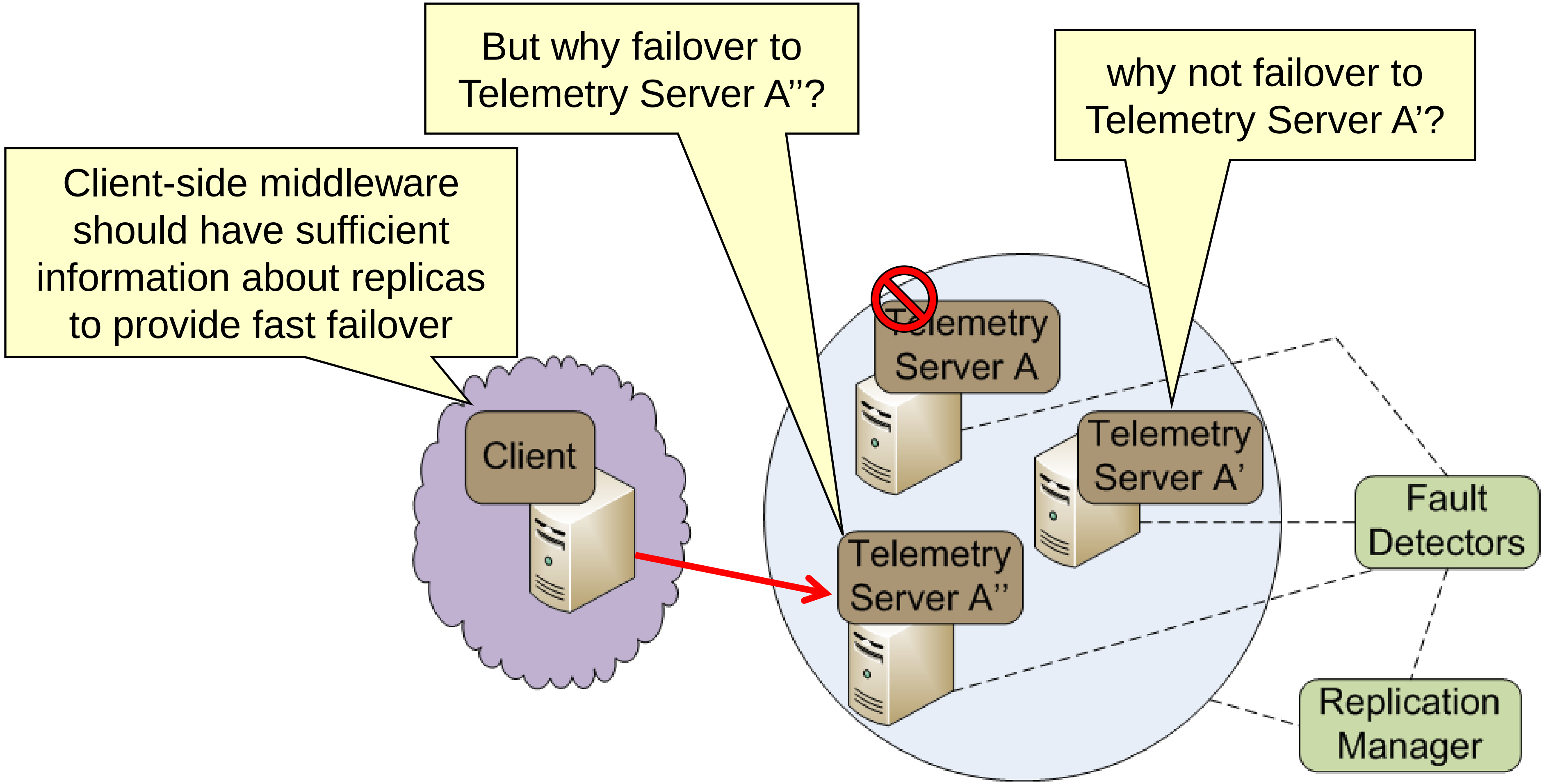
Challenges in Runtime Management of Fault-tolerant DRE Systems

- Providing high availability & soft real-time performance at runtime is hard
  - failures need to be detected quickly so that failure recovery actions can proceed
    - failure recovery should be fast



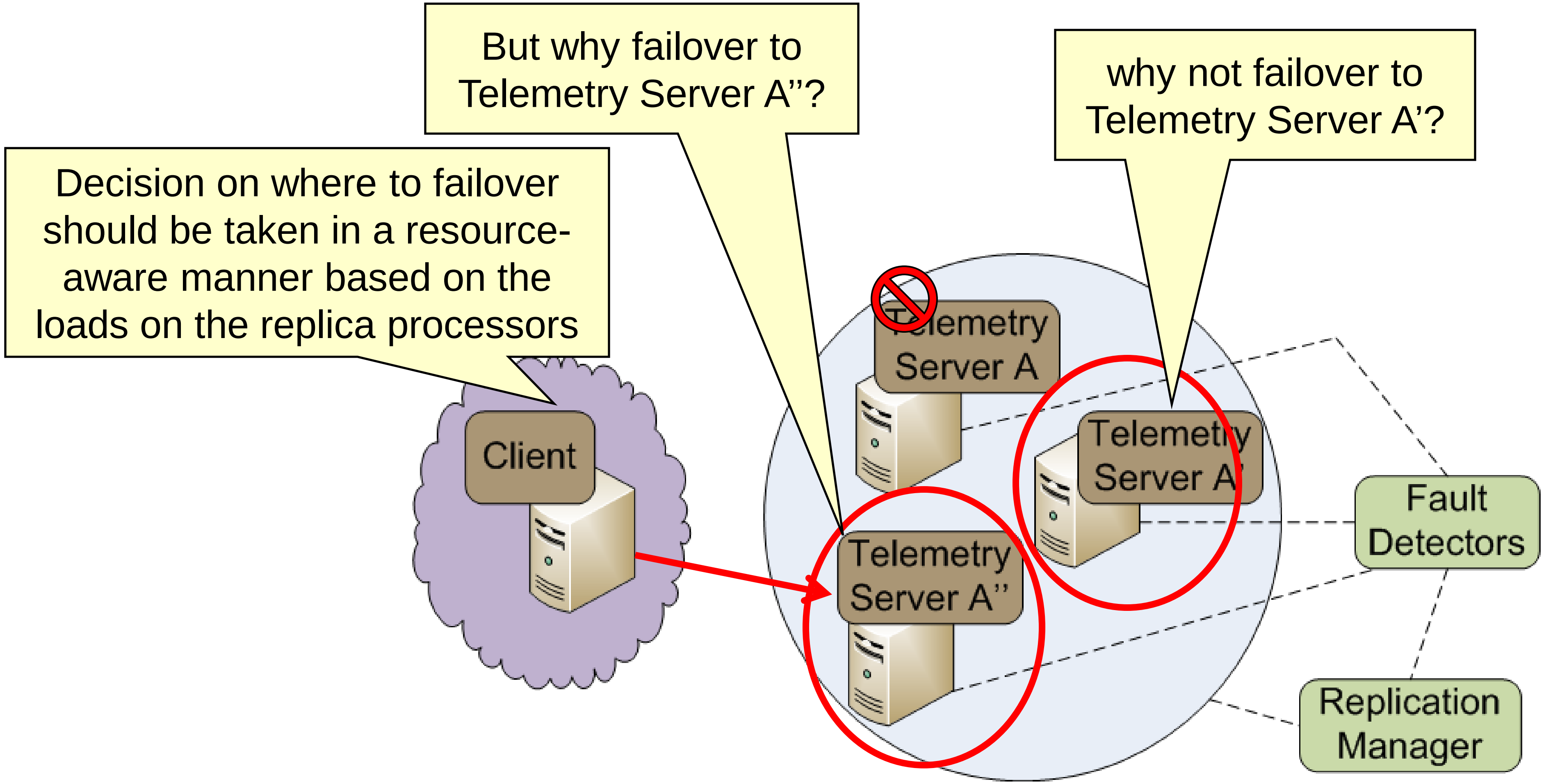
Challenges in Runtime Management of Fault-tolerant DRE Systems

- Providing high availability & soft real-time performance at runtime is hard
  - failures need to be detected quickly so that failure recovery actions can proceed
    - failure recovery should be fast



Challenges in Runtime Management of Fault-tolerant DRE Systems

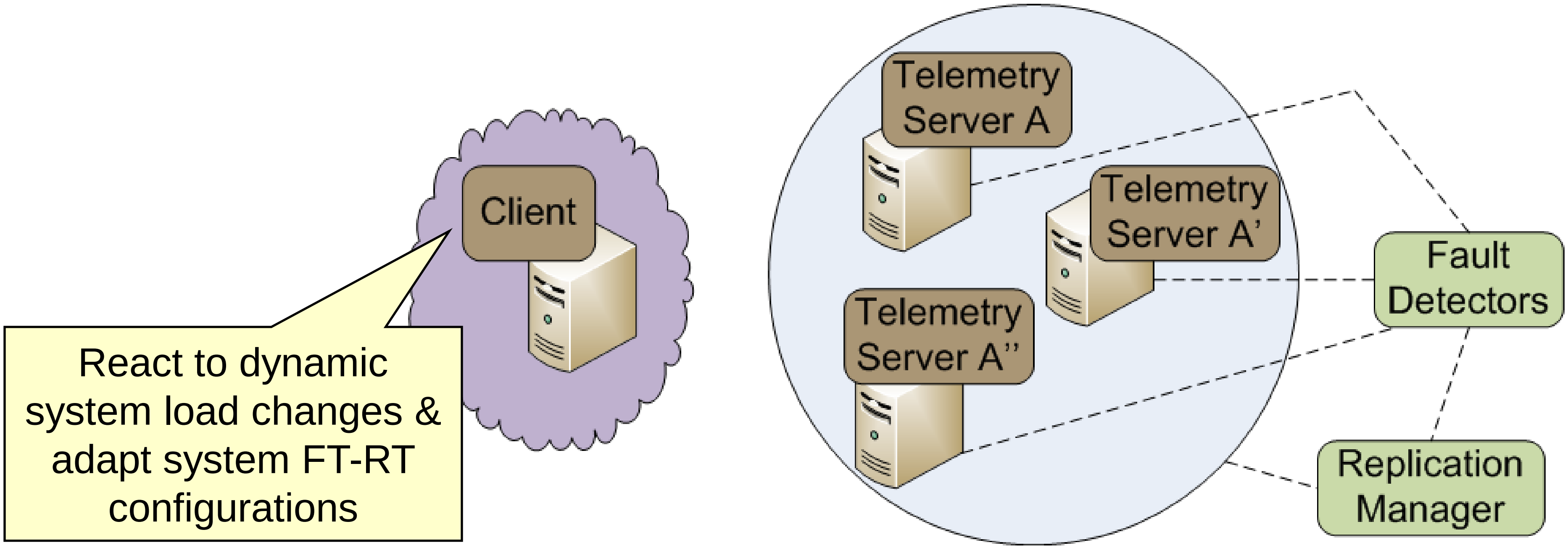
- Providing high availability & soft real-time performance at runtime is hard
  - failures need to be detected quickly so that failure recovery actions can proceed
    - failure recovery should be fast



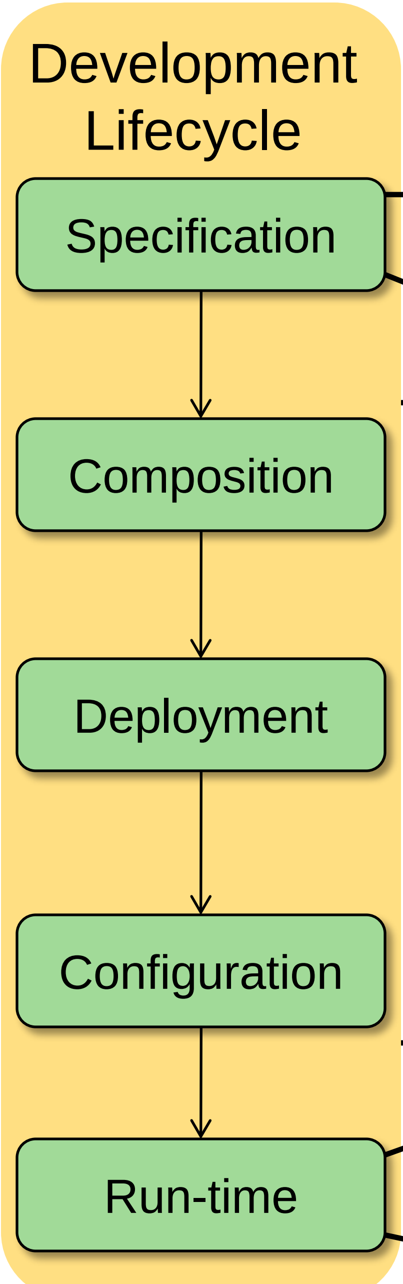
Challenges in Runtime Management of Fault-tolerant DRE Systems

- Providing high availability & soft real-time performance at runtime is hard
  - failures need to be detected quickly so that failure recovery actions can proceed
    - failure recovery should be fast
- Ad-hoc mechanisms to recover from failures & overloads could affect soft real-time performance of clients
  - ***need for adaptive fault-tolerance techniques***

Need for Adaptive Fault-tolerant Middleware



Summary of FT QoS Provisioning Challenges Across DRE Lifecycle



Our solutions integrate within the traditional DRE system lifecycle

- How to specify FT & other end-to-end QoS requirements?

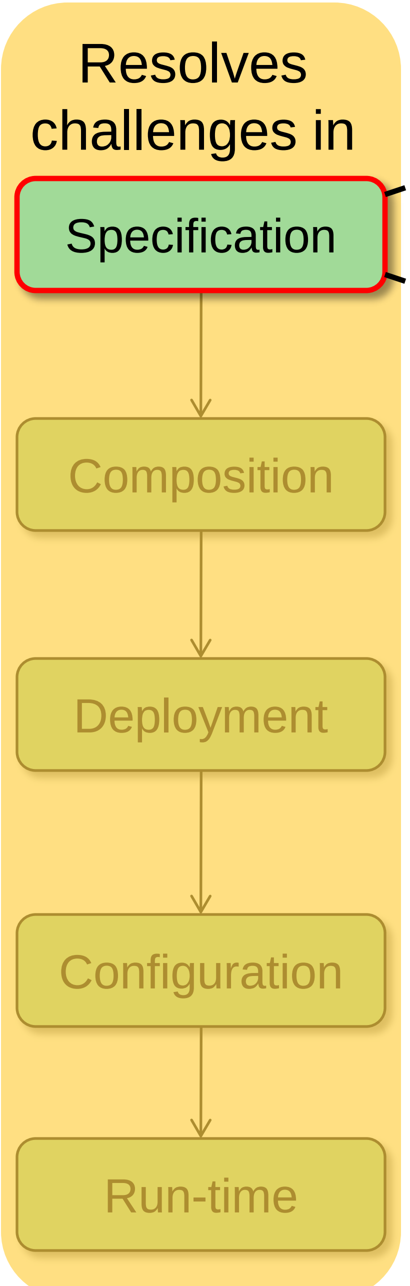
- How to compose & deploy application components & their replicas with concern for minimizing resources used yet satisfying FT-RT requirements?
- How to configure the underlying middleware to provision QoS?

- How to provide real-time fault recovery?
- How to deal with the side effects of replication & non-determinism at run-time?

Presentation Road Map

- Technology Context: DRE Systems
- DRE System Lifecycle & FT-RT Challenges
- **Design-time Solutions**
- Deployment & Configuration-time Solutions
- Runtime Solutions
- Ongoing Work
- Concluding Remarks

Specifying FT & Other QoS Properties



- Component QoS Modeling Language (CQML)
    - Aspect-oriented Modeling for Modularizing QoS Concerns

Focus on Model-driven Engineering and generative techniques to specify and provision QoS properties

Related Research: QoS Modeling

Category	Related Research (QoS & FT Modeling)
Using UML	1. *UML Profile for Schedulability, Performance, & Time (SPT)* 2. *UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics & Mechanisms (QoS&FT)* 3. *UML Profile for Modeling & Analysis of Real-Time & Embedded Systems (MARTE)* 4. Component Quality Modeling Language by J. ÃŸyvind Aagedal 5. *Modeling & Integrating Aspects into Component Architectures* by L. Michotte, R. France, & F. Fleurey 6. *A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture* by H. Wada, J. Suzuki, & K. Oba
Using domain-specific languages (DSL)	1. *Model-based Development of Embedded Systems: The SysWeaver Approach* by D. de Niz, G. Bhatia, & R. Rajkumar 2. *A Modeling Language & Its Supporting Tools for Avionics Systems* by G. Karsai, S. Neema, B. Abbott, & D. Sharp 3. *High Service Availability in MaTRICS for the OCS* by M. Bajohr & T. Margaria 4. *Modeling of Reliable Messaging in Service Oriented Architectures* by L. Gönczy & D. Varró 5. *Fault tolerance AOP approach* by J. Herrero, F. Sanchez, & M. Toro

Related Research: QoS Modeling

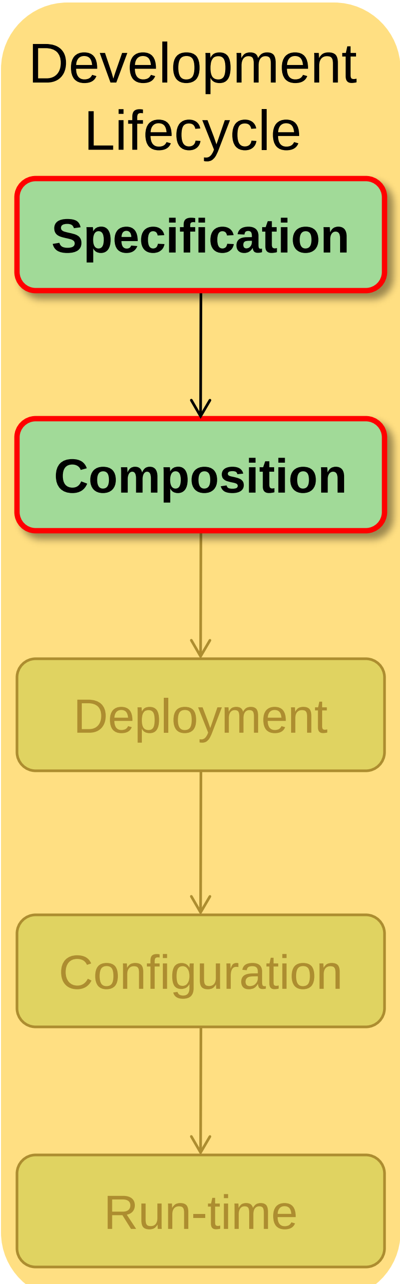
Category	Related Research (QoS & FT Modeling)
Using UML	1. *UML Profile for Schedulability, Performance, & Time (SPT)* 2. *UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics & Mechanisms (QoS&FT)* 3. *UML Profile for Modeling & Analysis of Real-Time & Embedded Systems (MARTE)* 4. Component Quality Modeling Language by J. ÃŸyvind Aagedal 5. *Modeling & Integrating Aspects into Component Architectures* by L. Michotte, R. France, & F. Fleurey 6. *A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture* by H. Wada, J. Suzuki, & K. Oba
Using domain-specific languages (DSL)	1. *Model-based Development of Embedded Systems: The SysWeaver Approach* by D. de Niz, G. Bhatia, & R. Rajkumar 2. *A Modeling Language & Its Supporting Tools for Avionics Systems* by G. Karsai, S. Neema, B. Abbott, & D. Sharp 3. *High Service Availability in MaTRICS for the OCS* by M. Bajohr & T. Margaria 4. *Modeling of Reliable Messaging in Service Oriented Architectures* by L. Gönczy & D. Varró 5. *Fault tolerance AOP approach* by J. Herrero, F. Sanchez, & M. Toro

Recovery block modeling and QoS for SOA

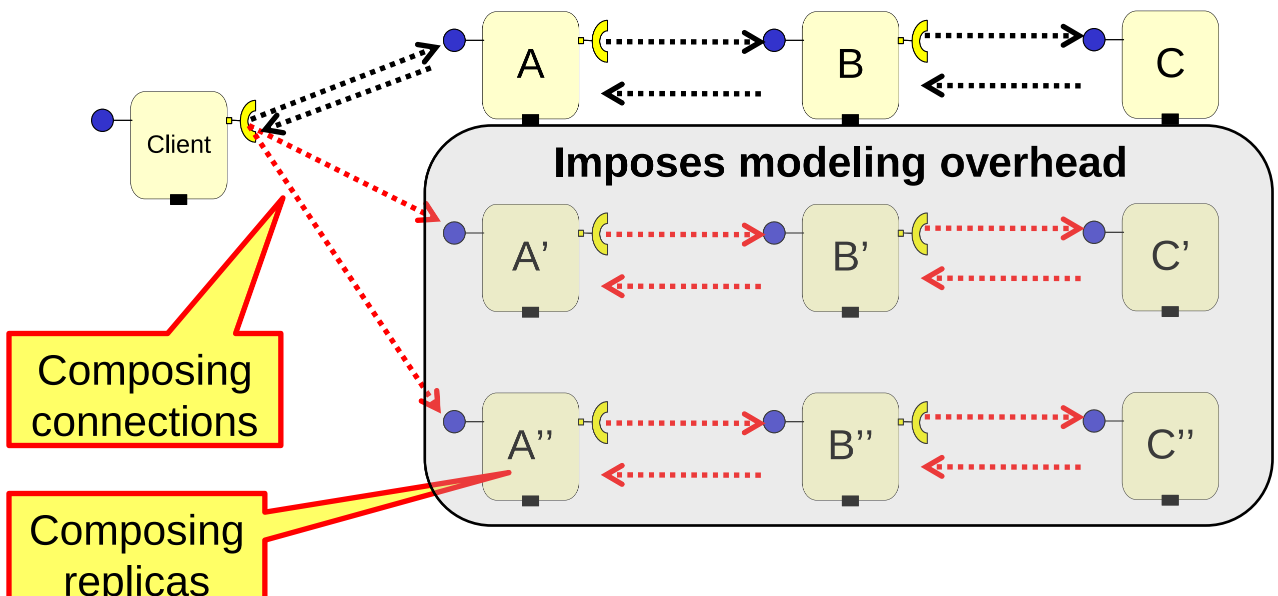
Lightweight & Heavyweight UML extensions

MoC = service logic graphs, state machine, Java extension

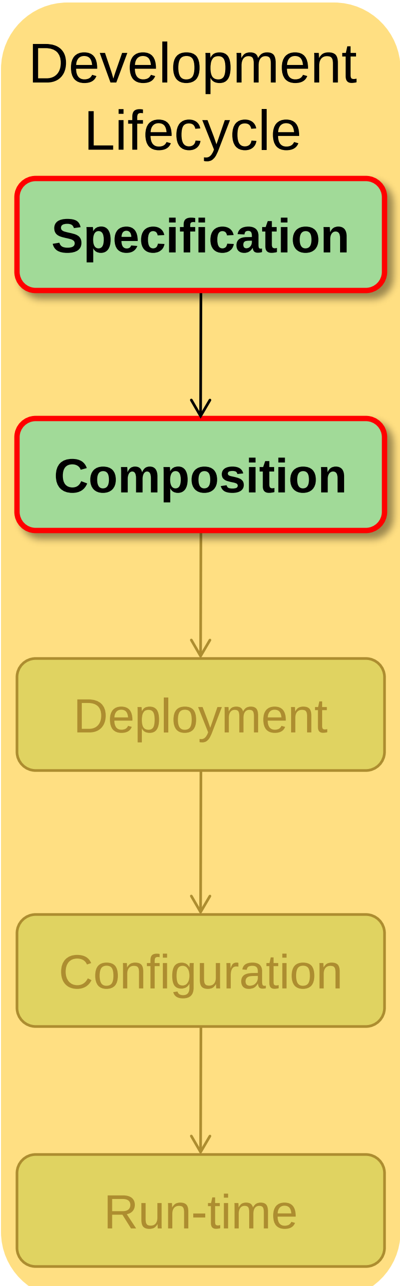
QoS Specification: What is Missing for DRE Systems?



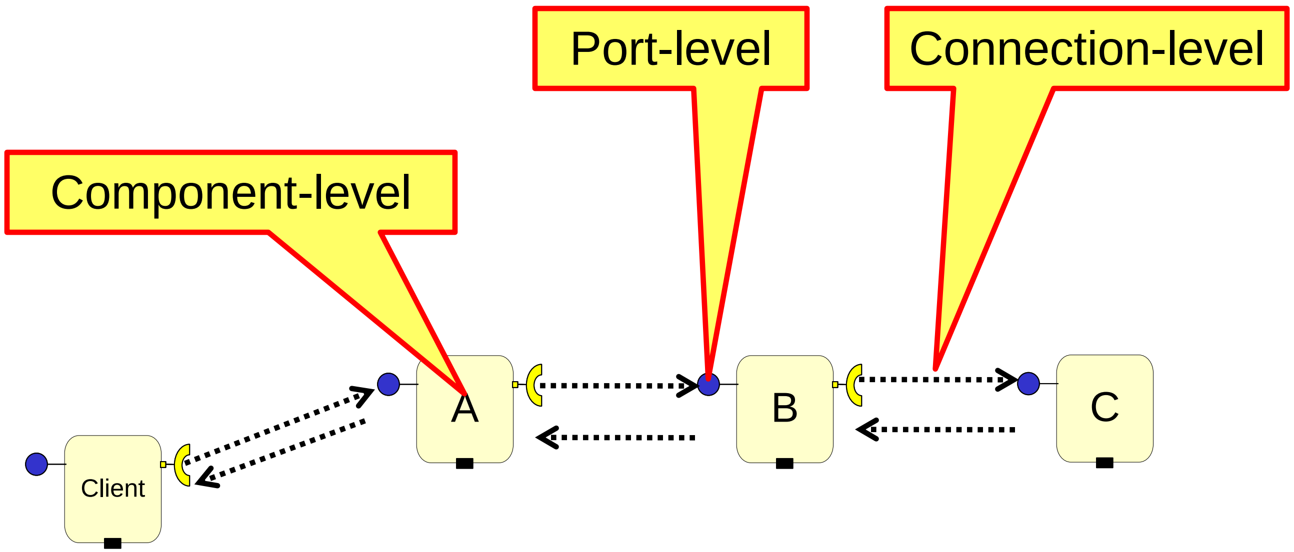
- Crosscutting availability requirements
  - Tangled with primary structural dimension
  - Tangled with secondary dimensions (deployment, QoS)
  - Composing replicated & non-replicated functionality
  - Example: Replicas must be modeled, composed, & deployed
- Imposes modeling overhead
- Supporting non-isomorphic replication
  - Reliability through diversity (structural & QoS)
  - Supporting graceful degradation through diversity



QoS Specification: What is Missing for DRE Systems?



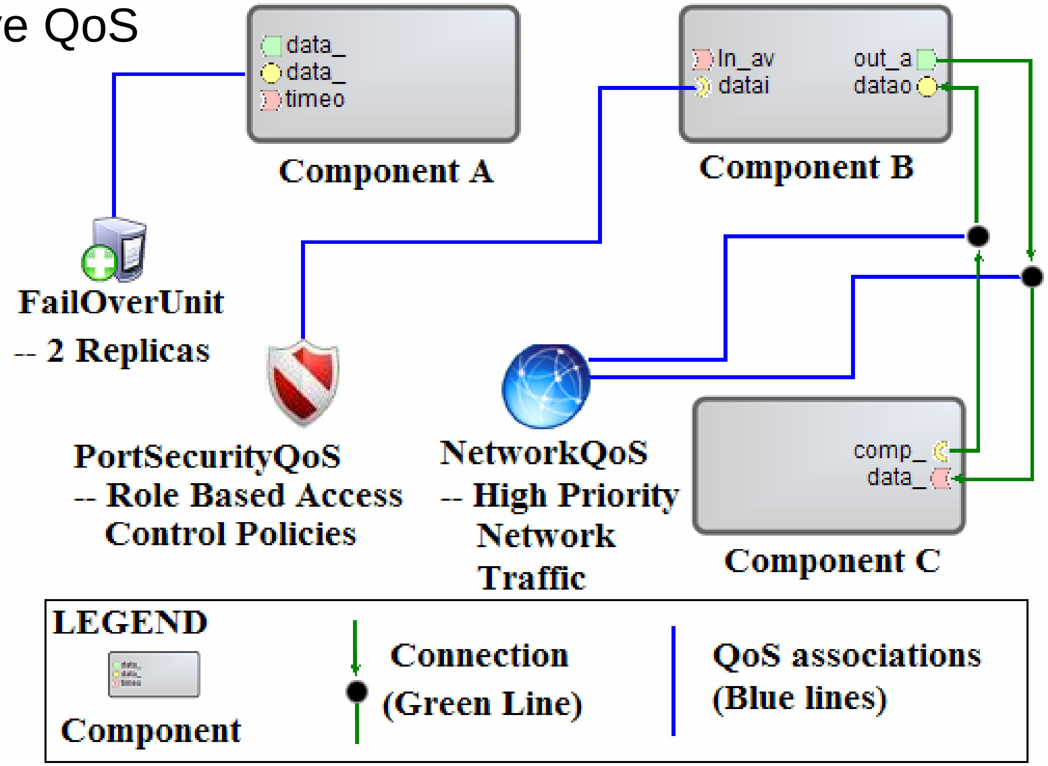
- Variable granularity of failover
  - Whole operational string, sub-string, or a component group
- Variable QoS association granularity



- Network-level QoS specification (connection level)
  - Differentiated service based on traffic class & flow
    - Example: High priority, high reliability, low latency
  - Bidirectional bandwidth requirements

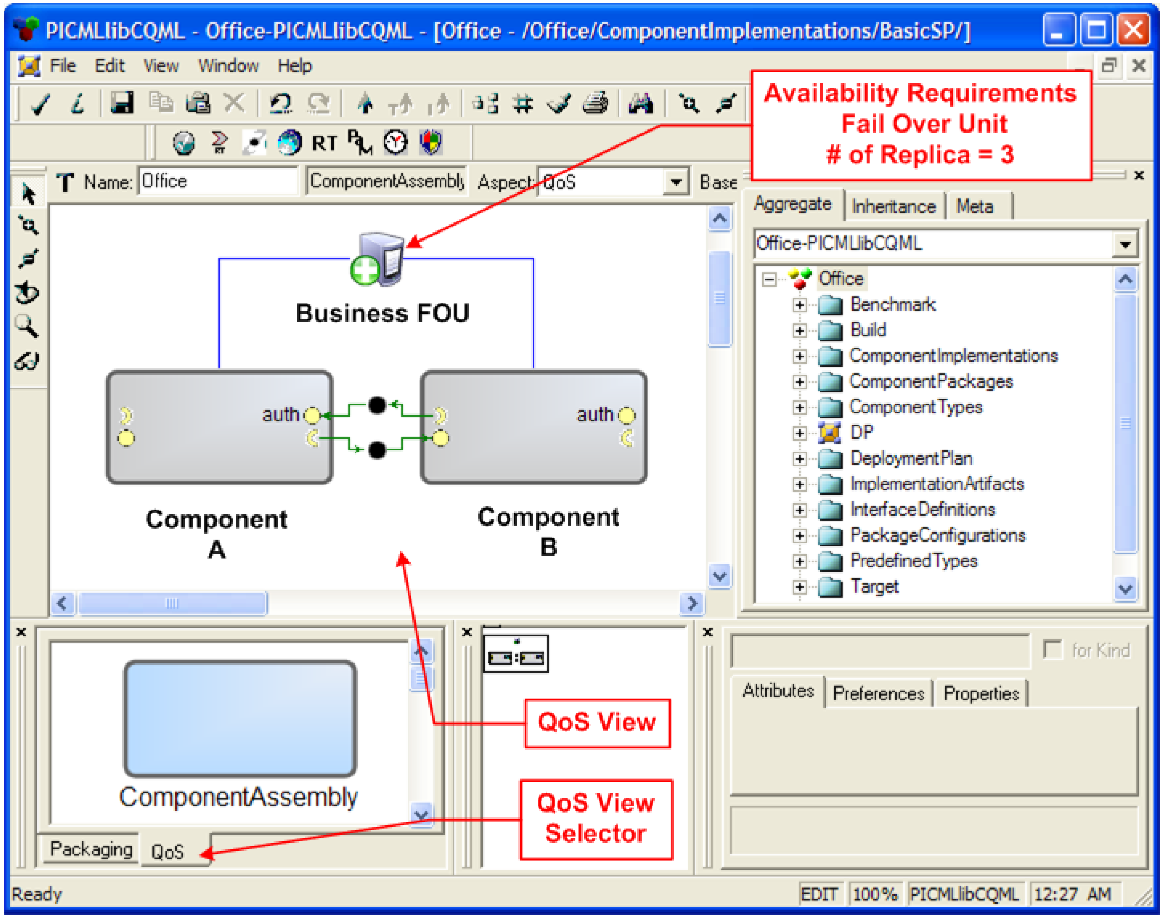
Our Solution: Domain Specific Modeling

- **Component QoS Modeling Language (CQML)**
  - A modeling framework for declarative QoS specification
  - Reusable for multiple composition modeling languages
- Failover unit for Fault-tolerance
  - Capture the granularity of failover
  - Specify # of replicas
- Network-level QoS
  - Annotate component connections
  - Specify priority of communication traffic
  - Bidirectional bandwidth requirements
- Security QoS
- Real-time CORBA configuration
- Event channel configuration



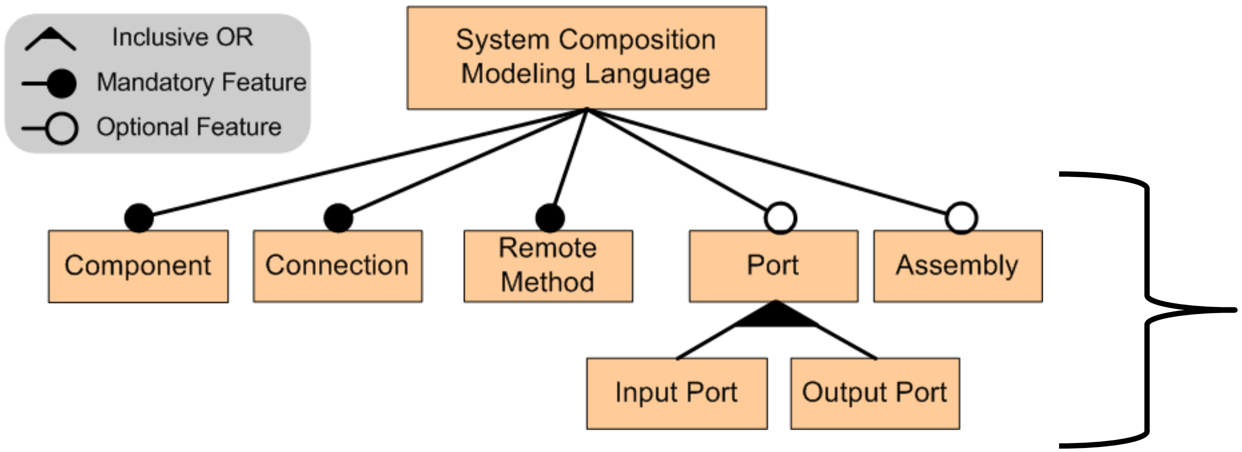
Separation of Concerns in CQML

- Resolving tangling of functional composition & QoS concerns
- Separate Structural view from the QoS view
- GRAFT transformations use aspect-oriented model weaving to coalesce both the views of the model



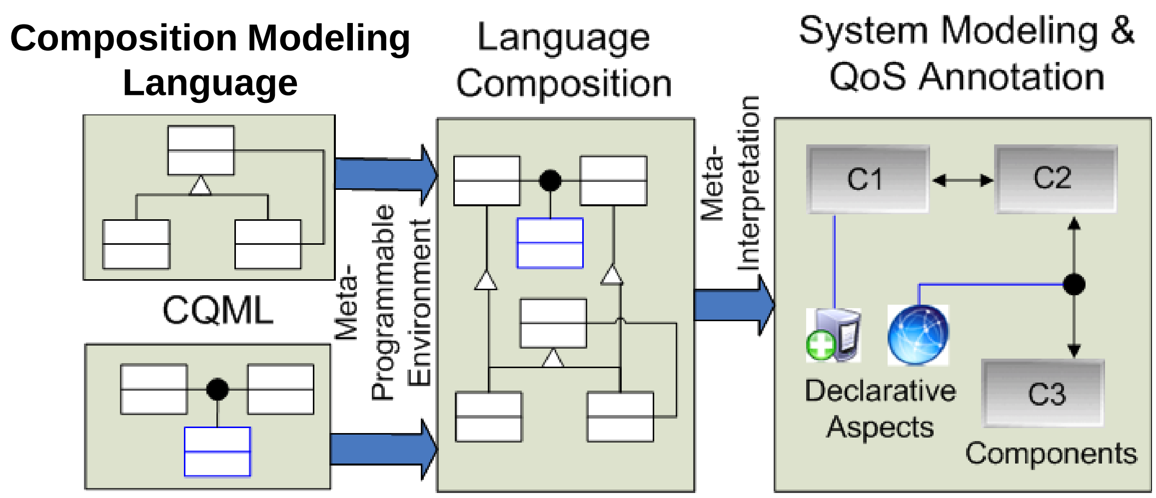
Granularity of QoS Associations in CQML

- Commonality/Variability analysis of composition modeling languages
  - e.g., PICML for CCM, J2EEML for J2EE, ESML for Boeing Bold-Stroke
- Feature model of composition modeling languages



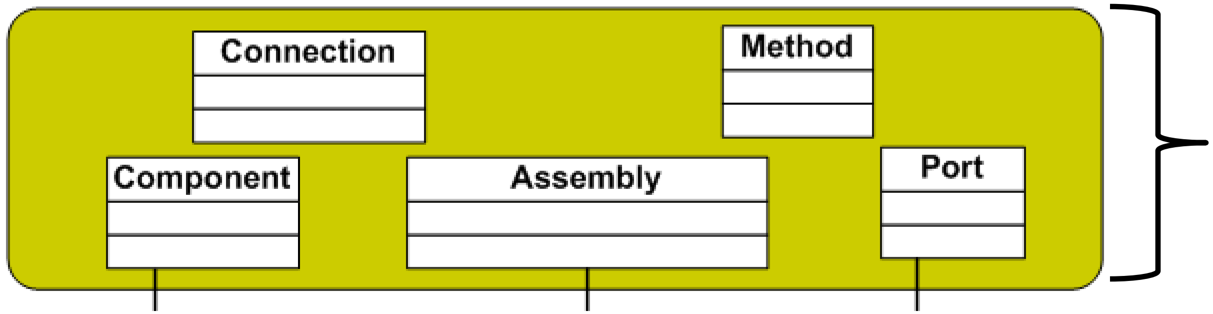
Dictates QoS association granularity

- Enhance composition language to model QoS
  - GME meta-model composition



Composing CQML (1/3)

Composition Modeling Language



PICML
or
J2EEML
or
ESML

Goal: Create reusable & loosely coupled associations

Concrete QoS Elements

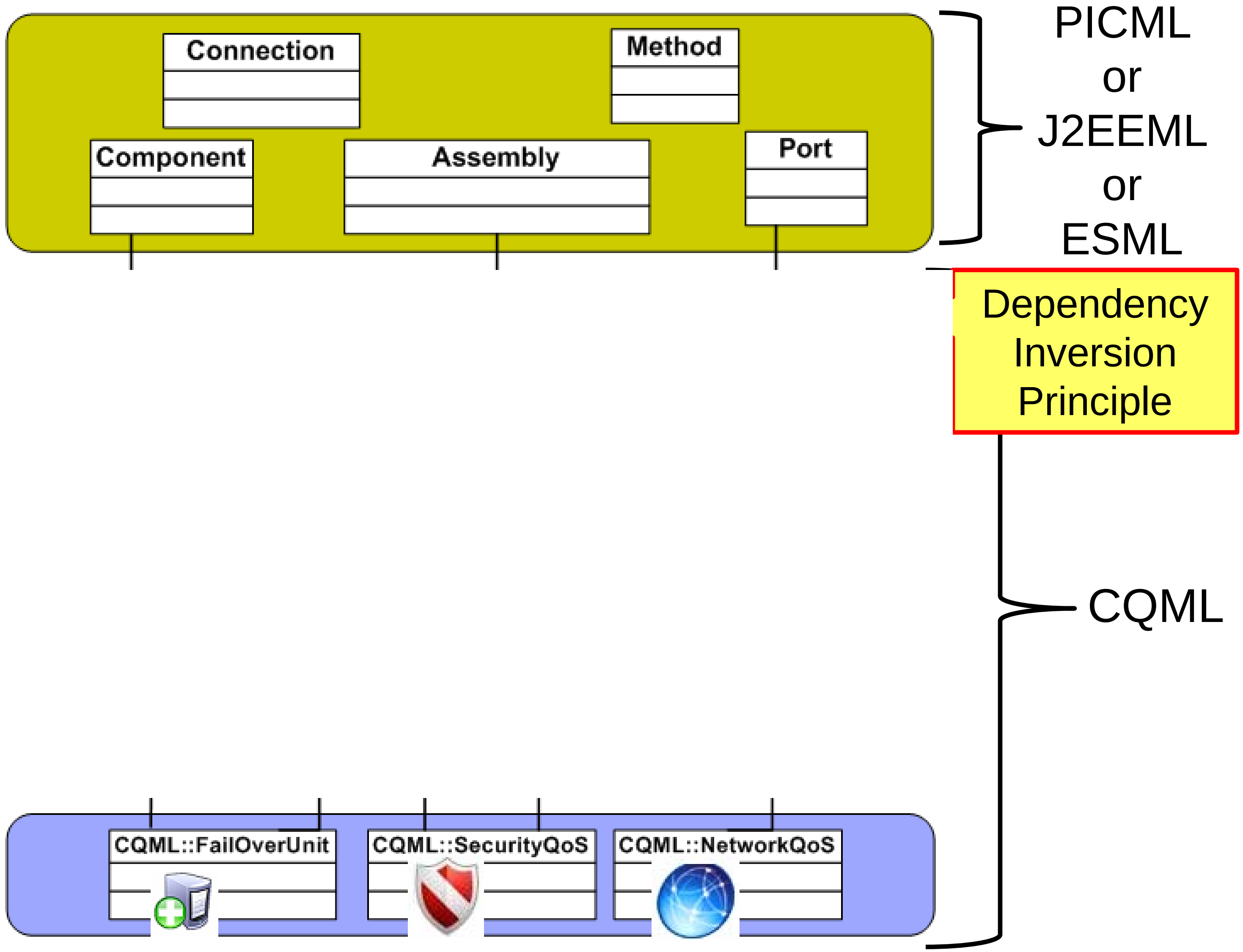
CQML

Composing CQML (2/3)

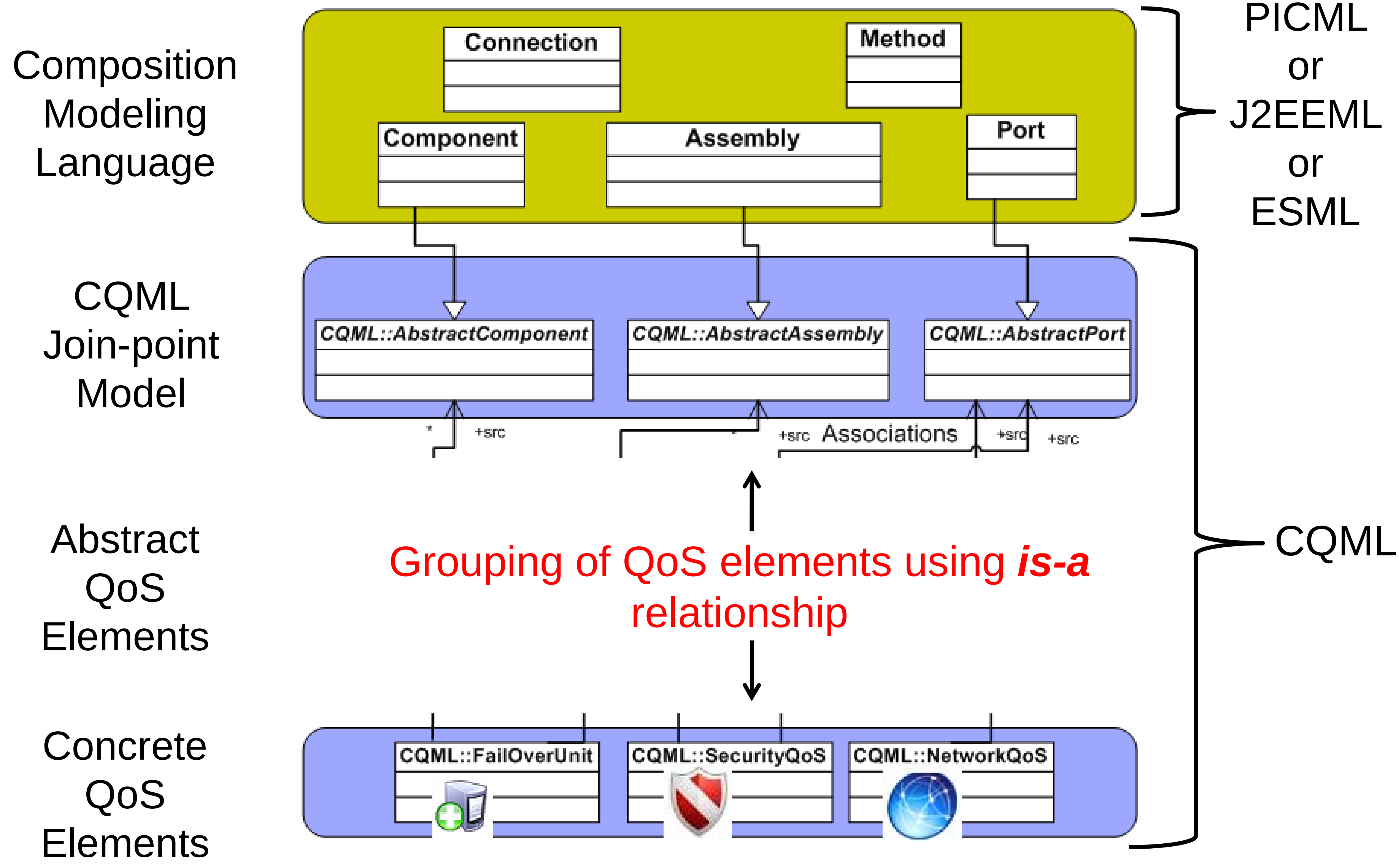
Composition Modeling Language

CQML Join-point Model

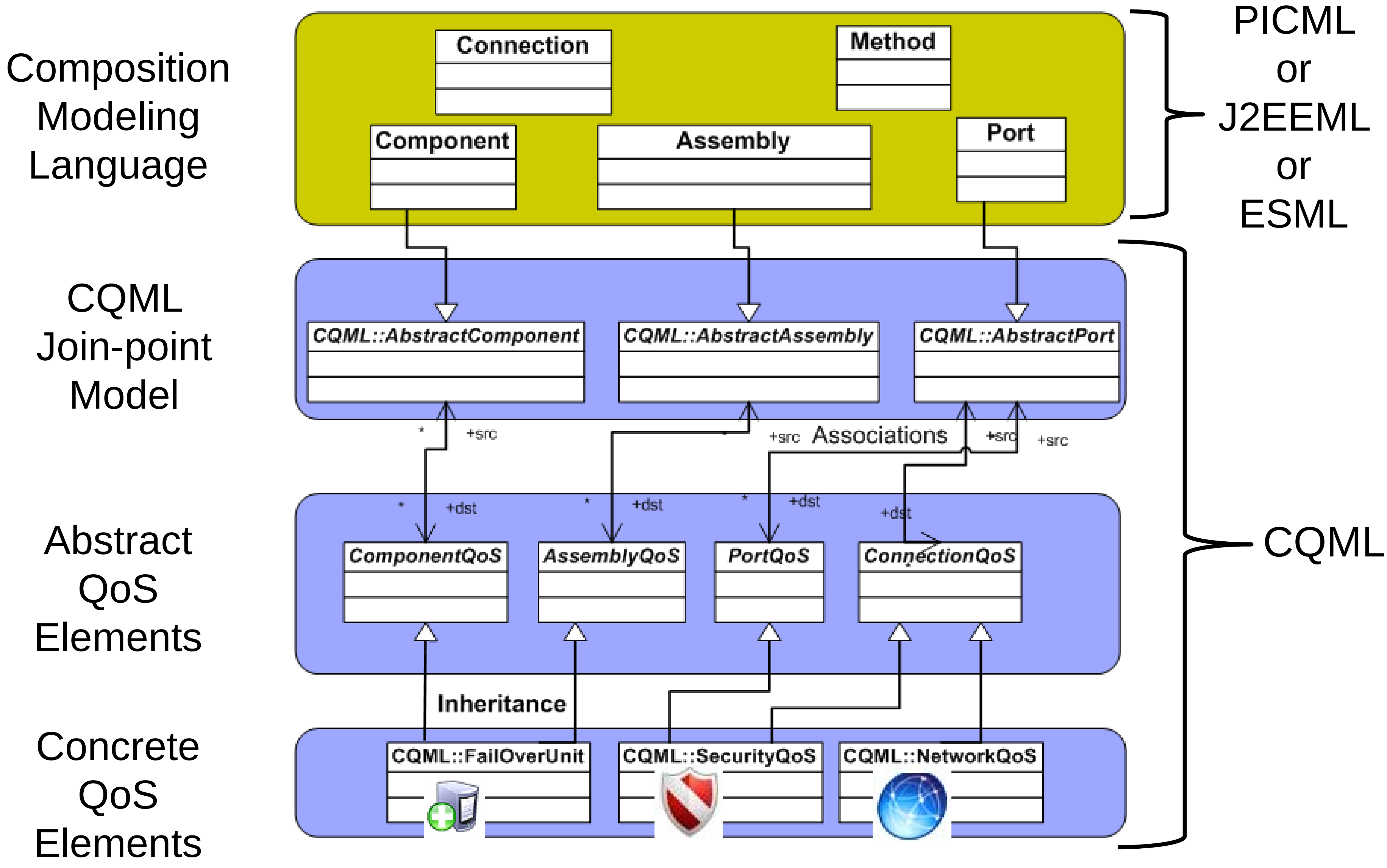
Concrete QoS Elements



Composing CQML (3/3)



Composing CQML (3/3)



Evaluating Composability of CQML

- Three composition modeling languages
  - PICML
  - J2EEML
  - ESML
- Available feature-set determines the extent of applicability of the join-point model
- Three composite languages with varying QoS modeling capabilities
  - PICML’
  - J2EEML’
  - ESML’

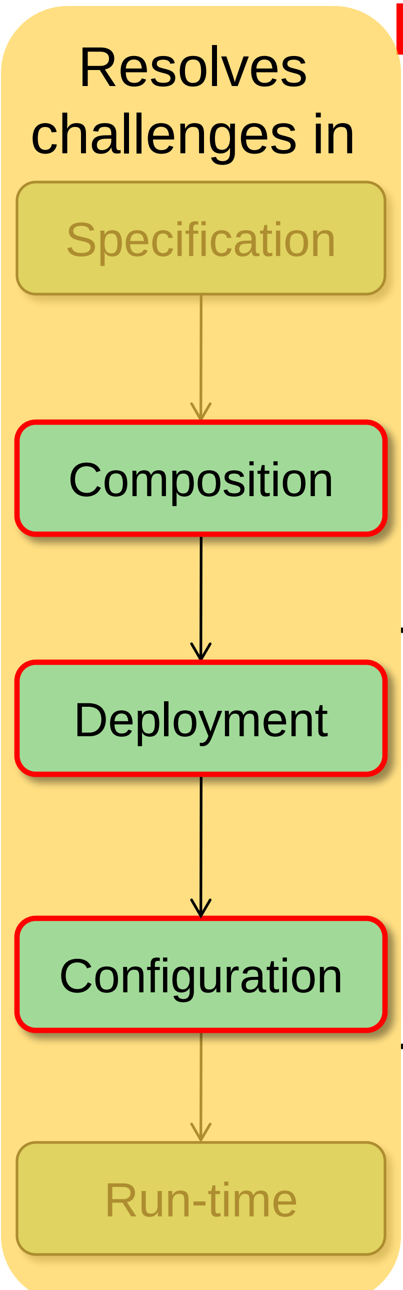
Supported Features	PICML	J2EEML	ESML
Component, Methods, and Connections	Yes	Yes	Yes
Provided Interface Ports	Yes	No	Yes
Required Interface Ports	Yes	No	Yes
Assemblies	Yes	Yes	No

Structural Elements	PICML′	J2EEML′	ESML′
Component	FailOverUnit	FailOverUnit	FailOverUnit
Assembly	FailOverUnit	FailOverUnit	N.A.
Connections	NetworkQoS	NetworkQoS	NetworkQoS
Provided Interface Ports	SecurityQoS	N.A.	SecurityQoS
Required Interface Ports	SecurityQoS	N.A.	SecurityQoS

Presentation Road Map

- Technology Context: DRE Systems
- DRE System Lifecycle & FT-RT Challenges
- Design-time Solutions
- **Deployment & Configuration-time Solutions**
- Runtime Solutions
- Ongoing Work
- Concluding Remarks

Post-Specification Phase: Resource Allocation, Deployment and Configuration



Focus on Resource Allocation Algorithms and Frameworks used in Deployment and Configuration Phases

- Deployment & Configuration Reasoning & Analysis via Modeling (DeCoRAM)
  - Provides a specific deployment algorithm
  - Algorithm-agnostic deployment engine
  - Middleware-agnostic configuration engine

Related Research

Category	Related Research
CORBA-based Fault-tolerant Middleware Systems	P. Felber et. al., *Experiences, Approaches, & Challenges in Building Fault-tolerant CORBA Systems*, in IEEE Transactions on Computers, May 2004 T. Bennani et. al., *Implementing Simple Replication Protocols Using CORBA Portable Interceptors & Java Serialization*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN 2004), Italy, 2004 P. Narasimhan et. al., *MEAD: Support for Real-time Fault-tolerant CORBA*, in Concurrency & Computation: Practice & Experience, 2005
Adaptive Passive Replication Systems	S. Pertet et. al., *Proactive Recovery in Distributed CORBA Applications*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN 2004), Italy, 2004 P. Katsaros et. al., *Optimal Object State Transfer – Recovery Policies for Fault-tolerant Distributed Systems*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN 2004), Italy, 2004 Z. Cai et. al., *Utility-driven Proactive Management of Availability in Enterprise-scale Information Flows*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2006), Melbourne, Australia, November 2006 L. Froihofer et. al., *Middleware Support for Adaptive Dependability*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2007), Newport Beach, CA, November 2007

Related Research

Category	Related Research
CORBA-based Fault-tolerant Middleware Systems	P. Felber et. al., *Experiences, Approaches, & Challenges in Building Fault-tolerant CORBA Systems*, in IEEE Transactions on Computers, May 2004 T. Bennani et. al., *Implementing Simple Replication Protocols Using CORBA Portable Interceptors & Java Serialization*, in Proceedings of the IEEE International Conference on D ... 4), Italy, 2004 P. Narasimhan et. al., *MEAD:* ... 4, in Concurrency & Computation: Practice & Experience, 2005
Adaptive Passive Replication Systems	S. Pertet et. al., *Proactive R* ... , in Proceedings of the IEEE Int ... ems & Networks (DSN 2004), Italy, 2004 P. Katsaros et. al., *Optimal Object State Transfer – Recovery Policies for Fault-tolerant Distributed Systems*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN 2004), Italy, 2004 Z. Cai et. al., *Utility-driven Proactive Management of Availability in Enterprise-scale Information Flows*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2006), Melbourne, Australia, November 2006 L. Froihofer et. al., *Middleware Support for Adaptive Dependability*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2007), Newport Beach, CA, November 2007

Runtime adaptations to reduce failure recovery times

Middleware building blocks for fault-tolerant systems

Related Research

Category	Related Research
Real-time Fault-tolerance for Transient Failures	H. Aydin, *Exact Fault-Sensitive Feasibility Analysis of Real-time Tasks*, In IEEE Transactions of Computers, 2007 G. Lima et. al., *An Optimal Fixed-Priority Assignment Algorithm For Supporting Fault-Tolerant Hard Real-Time Systems*, In IEEE Transactions on Computers, 2003 Y. Zhang et. al., *A Unified Approach For Fault Tolerance & Dynamic Power Management in Fixed-Priority Real-Time Systems*, in IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems, 2006
Real-time Fault Tolerance for Permanent Failures	J. Chen et. al., *Real-Time Task Replication For Fault-Tolerance in Identical Multiprocessor Systems*, In Proceedings of the IEEE Real-Time & Embedded Technology & Applications Symposium (IEEE RTAS), 2007 P. Emberson et. al., *Extending a Task Allocation Algorithm for Graceful Degradation of Real-time Distributed Embedded Systems*, In Proceedings of the IEEE Real-time Systems Symposium (IEEE RTSS), 2008 A. Girault et. al., *An Algorithm for Automatically Obtaining Distributed & Fault-Tolerant Static Schedules*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (IEEE DSN), 2003 S. Gopalakrishnan et. al., *Task Partitioning with Replication Upon Heterogeneous Multiprocessor Systems*, in Proceedings of the IEEE Real-Time & Embedded Technology & Applications Symposium (IEEE RTAS), 2006

Related Research

Category	Related Research
Real-time Fault-tolerance for Transient Fa	H. Aydin, *Exact Fault-Sensitive Feasibility Analysis of Real-time Tasks*, In IEEE Transactions of Computers, 2007 *Priority Assignment Algorithm For Supporting* *stems*, In IEEE Transactions on Computers, 2003 Y. Zhang et. al., *l Approach For Fault Tolerance & Dynamic Power Management in F riority Real-Time Systems*, in IEEE Transactions on Computer-Aided D of Integrated Circuits & Systems, 2006
Real-time Fault Tolerance for Permanent Failures	J. Chen et. al., *Real-Time Task Replic Fault-Tolerance in Identical Multiprocessor Systems*, In Pro d Technology & Applications Sym P. Emberson et. al., *Extending Degradation of Real-time Distributed Embedded Systems*, In Proceedings of the IEEE Real-time Systems Symposium (IEEE RTSS), 2008 A. Girault et. al., *An Algorithm for Automatically Obtaining Distributed & Fault-Tolerant Static Schedules*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (IEEE DSN), 2003 S. Gopalakrishnan et. al., *Task Partitioning with Replication Upon Heterogeneous Multiprocessor Systems*, in Proceedings of the IEEE Real-Time & Embedded Technology & Applications Symposium (IEEE RTAS), 2006

Used active replication schemes

Static allocation algorithms that deal with transient failures

Related Research

Category	Related Research
Passive Replication Based Real-time Fault-Tolerant Task Allocation Algorithms	R. Al-Omari et. al., *An Adaptive Scheme for Fault-Tolerant Scheduling of Soft Real-time Tasks in Multiprocessor Systems* , In Journal of Parallel & Distributed Computing, 2005 W. Sun et. al., *Hybrid Overloading & Stochastic Analysis for Redundant Real-time Multiprocessor Systems*, In Proceedings of the IEEE Symposium on Reliable Distributed Systems (IEEE SRDS), 2007 Q. Zheng et. al., *On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs*, in IEEE Transactions on Computers, 2009

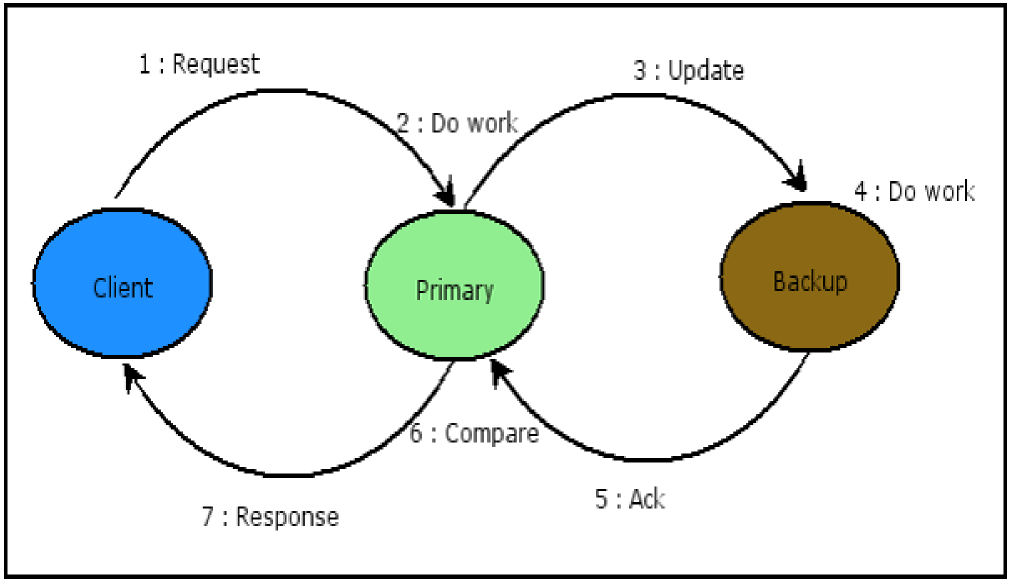
Related Research

Category	Related Research
Passive Replication Based Real-time Fault-Tolerant Task Allocation Algorithms	R. Al-Omari et. al., *An Adaptive Scheme for Fault-Tolerant Scheduling of Soft Real-time Tasks in Multiprocessor Systems* , In Journal of Parallel & Distributed Computing, 2005 W. Sun et. al., *Hybrid Overloading & Stochastic Analysis for Redundant Real-time Multiprocessor Systems*, In Proceedings of the IEEE Symposium on Reliable Distributed Systems (IEEE SRDS), 2007 Q. Zheng et. al., *On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs*, in IEEE Transactions on Computers, 2009

All these algorithms deal with dynamic scheduling

D&C: What is Missing for DRE Systems?

- Existing passive replication middleware solutions are not resource-aware
  - provide mechanisms – but no intuition on how to use them to obtain the required solution
  - timeliness assurances might get affected as failures occur

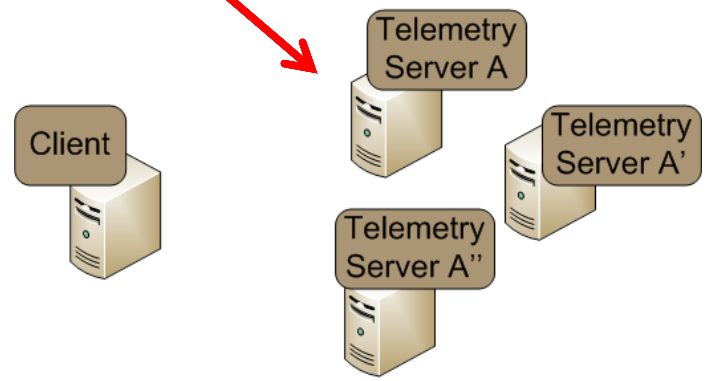


- Existing real-time fault-tolerant task allocation algorithms are not appropriate for closed DRE systems
  - they deal with active replication which is not ideal for resource-constrained systems
  - those that deal with passive replication
    - support only one processor failure
    - require dynamic scheduling – which adds extra unnecessary overhead

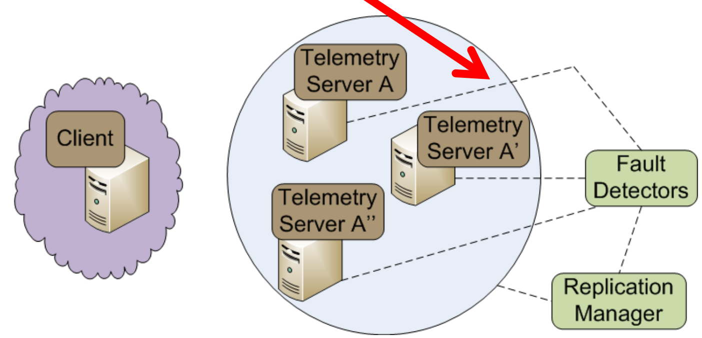
Our Solution: The DeCoRAM D&C Middleware

- **DeCoRAM** = “Deployment & Configuration Reasoning via Analysis & Modeling”
- DeCoRAM consists of
  - **Pluggable Allocation Engine** that determines appropriate node mappings for all applications & replicas using installed algorithm
  - **Deployment & Configuration Engine** that deploys & configures (D&C) applications and replicas on top of middleware in appropriate hosts
  - **A specific allocation algorithm** that is real time-, fault- and resource-aware

No coupling with allocation algorithm

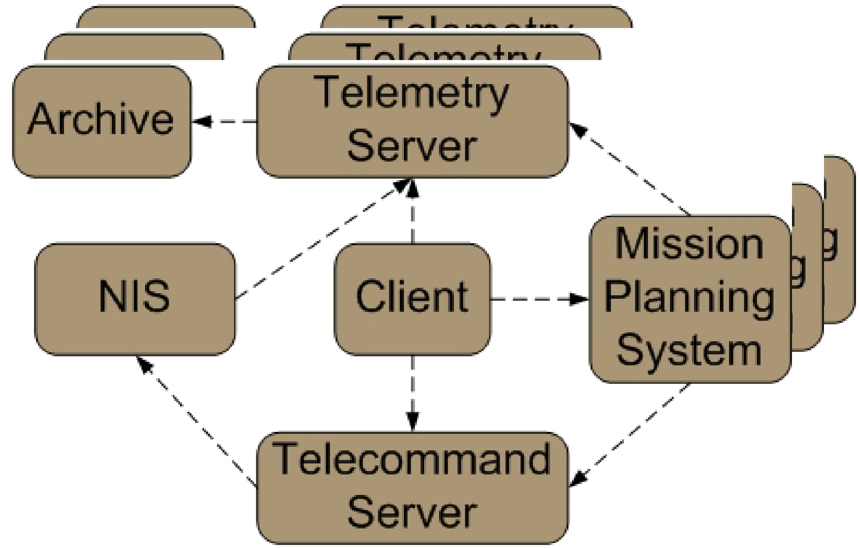


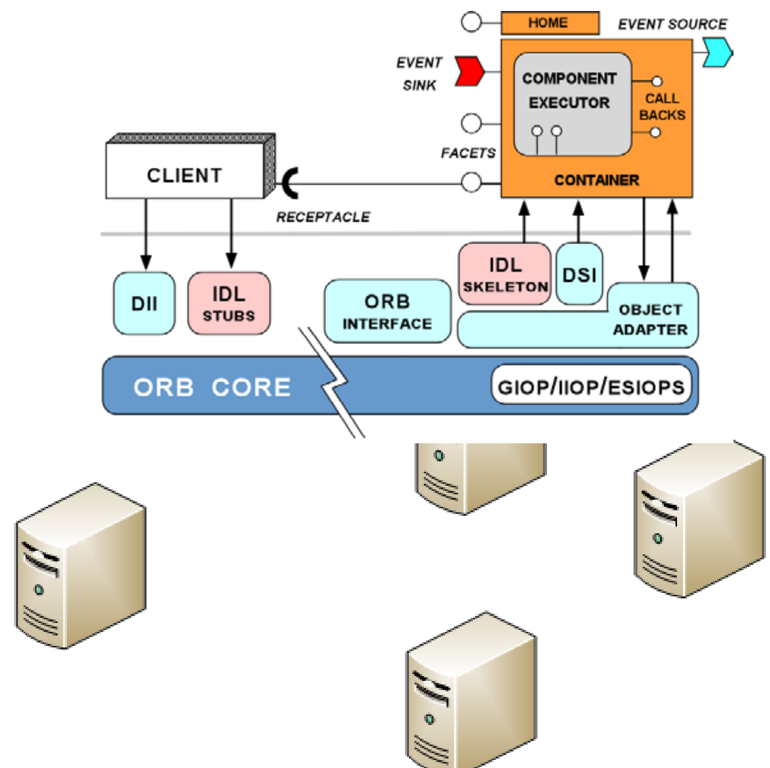
Middleware-agnostic D&C Engine



Overview of DeCoRAM Contributions

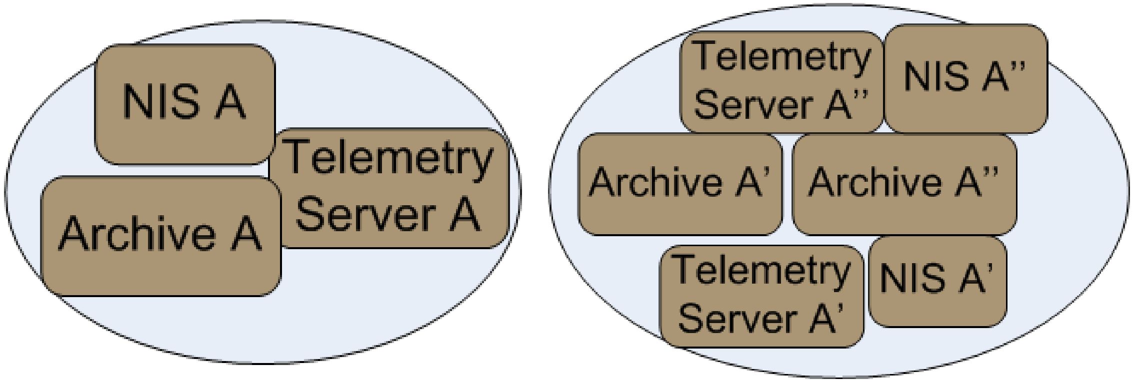
1. Provides a replica allocation algorithm that is
   - Real time-aware
   - Fault-aware
   - Resource-aware
2. Supports a large class of DRE systems => No tight coupling to any single allocation algorithm
3. Supports multiple middleware technologies => Automated middleware configuration that is not coupled to any middleware





DeCoRAM Allocation Algorithm

- **System model**
  - *N* periodic DRE system tasks
  - *RT requirements* – periodic tasks, worst-case execution time (WCET), worst-case state synchronization time (WCSST)
  - *FT requirements* – *K* number of processor failures to tolerate (number of replicas)
  - Fail-stop processors



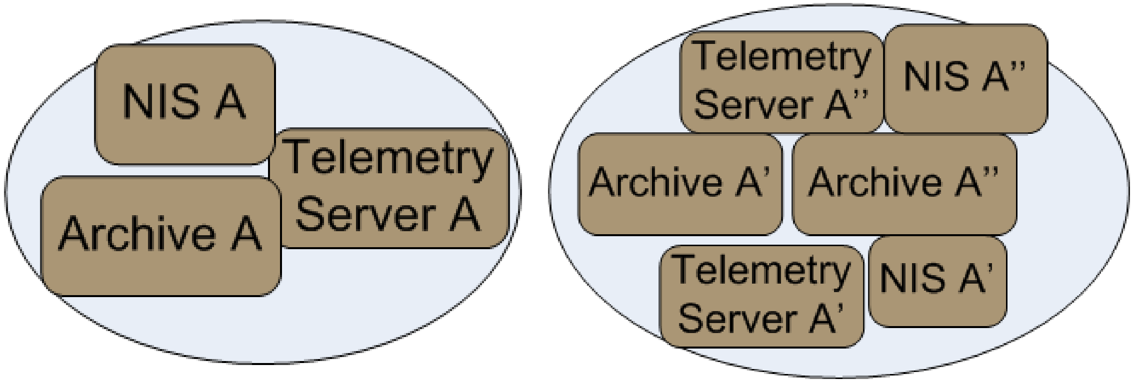
How many processors shall we need for a primary-backup scheme? – A basic intuition

Num proc in No-fault case <=

Num proc for passive replication <=
Num proc for active replication

DeCoRAM Allocation Algorithm (1/2)

- **System model**
  - *N* periodic DRE system tasks
  - *RT requirements* – periodic tasks, worst-case execution time (WCET), worst-case state synchronization time (WCSST)
  - *FT requirements* – *K* number of processor failures to tolerate (number of replicas)
  - Fail-stop processors



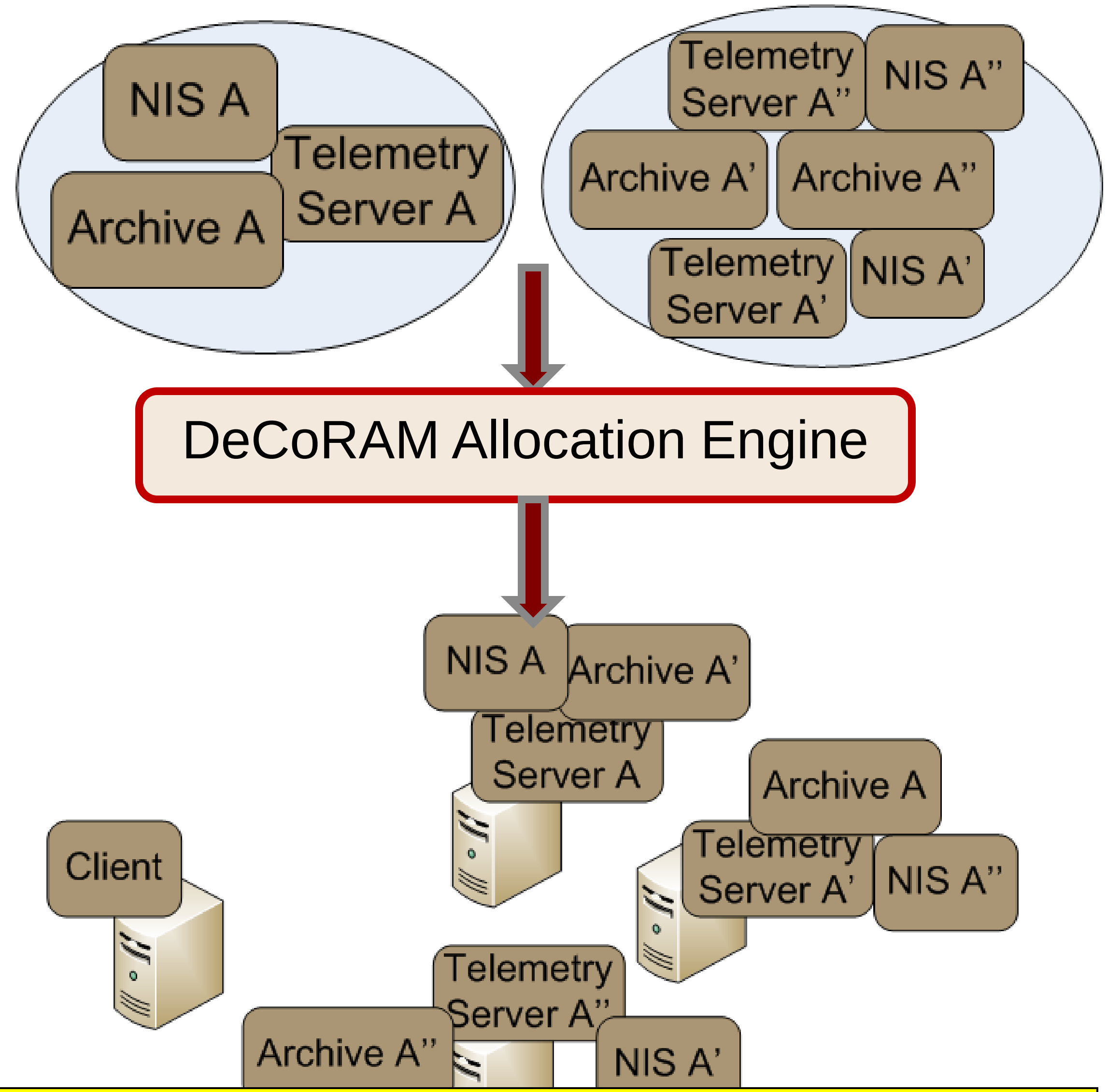
How many processors shall we need for a primary-backup scheme? – A basic intuition

Num proc in No-fault case <=

Num proc for passive replication <=
Num proc for active replication

DeCoRAM Allocation Algorithm (2/2)

- **System objective**
  - Find a mapping of *N* periodic DRE tasks & their *K* replicas so as to minimize the total number of processors utilized
    - no two replicas are in the same processor
    - All tasks are schedulable both in faulty as well as non-faulty scenarios



Similar to bin-packing, but harder due to combined FT & RT constraints

Designing the DeCoRAM Allocation Algorithm (1/5)

Basic Step 1: No fault tolerance

- Only primaries exist consuming WCET each
- Apply first-fit optimal bin-packing using the [Dhall:78]* algorithm
- Consider sample task set shown
- Tasks arranged according to rate monotonic priorities

Task	WCET	WCSST	Period	Util
A	20	0.2	50	40
B	40	0.4	100	40
C	50	0.5	200	25
D	200	2	500	40
E	250	2.5	1,000	25

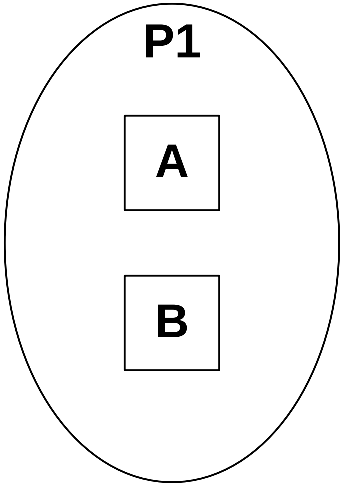
*[Dhall:78] S. K. Dhall & C. Liu, "On a Real-time Scheduling Problem", *Operations Research,* 1978

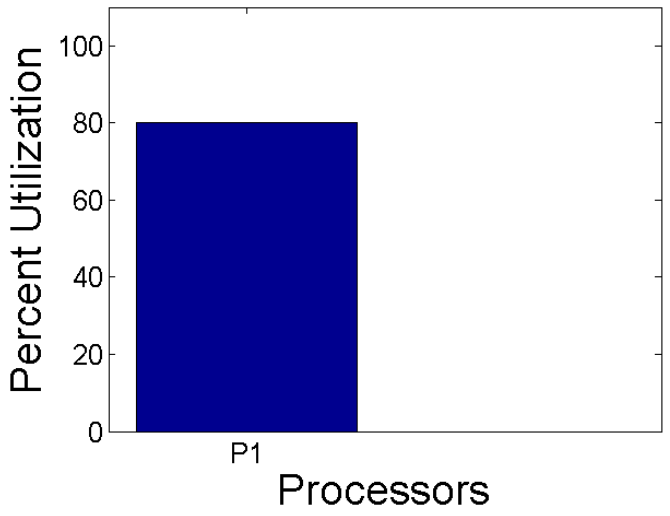
Designing the DeCoRAM Allocation Algorithm (1/5)

Basic Step 1: No fault tolerance

- Only primaries exist consuming WCET each
- Apply first-fit optimal bin-packing using the [Dhall:78] algorithm
- Consider sample task set shown
- Tasks arranged according to rate monotonic priorities

Task	WCET	WCSST	Period	Util
A	20	0.2	50	40
B	40	0.4	100	40
C	50	0.5	200	25
D	200	2	500	40
E	250	2.5	1,000	25



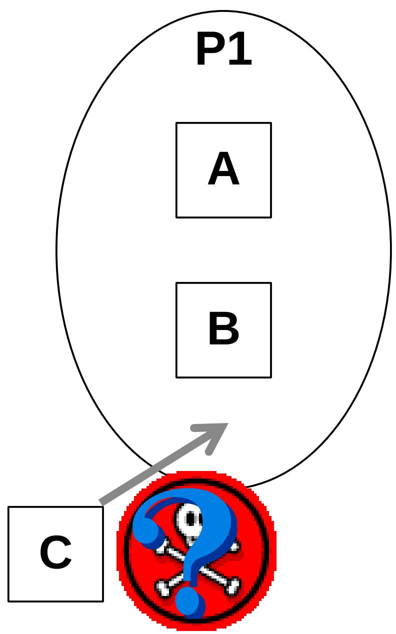


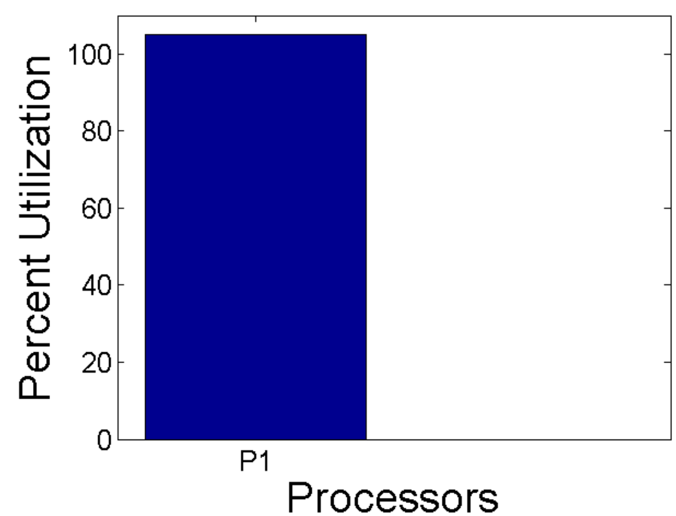
Designing the DeCoRAM Allocation Algorithm (1/5)

Basic Step 1: No fault tolerance

- Only primaries exist consuming WCET each
- Apply first-fit optimal bin-packing using the [Dhall:78] algorithm
- Consider sample task set shown
- Tasks arranged according to rate monotonic priorities

Task	WCET	WCSST	Period	Util
A	20	0.2	50	40
B	40	0.4	100	40
C	50	0.5	200	25
D	200	2	500	40
E	250	2.5	1,000	25



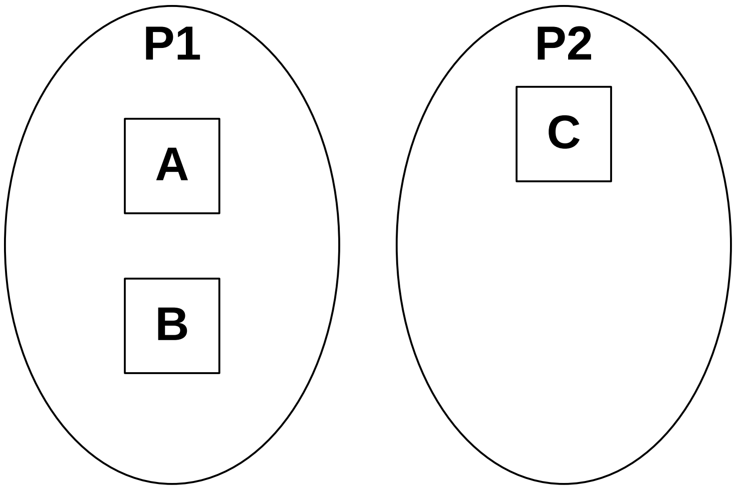


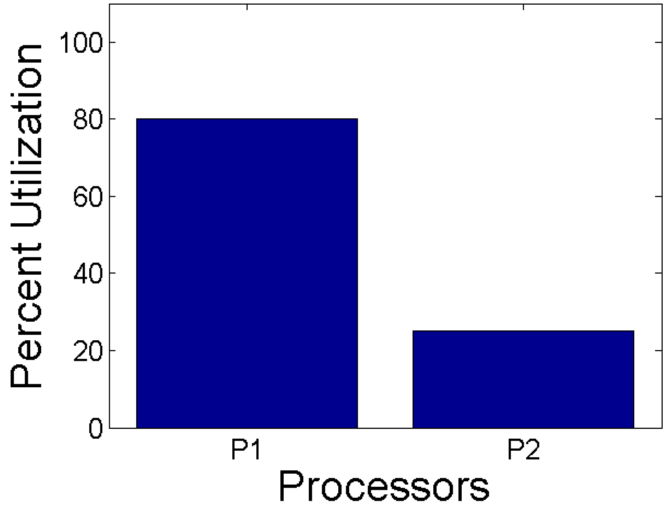
Designing the DeCoRAM Allocation Algorithm (1/5)

Basic Step 1: No fault tolerance

- Only primaries exist consuming WCET each
- Apply first-fit optimal bin-packing using the [Dhall:78] algorithm
- Consider sample task set shown
- Tasks arranged according to rate monotonic priorities

Task	WCET	WCSST	Period	Util
A	20	0.2	50	40
B	40	0.4	100	40
C	50	0.5	200	25
D	200	2	500	40
E	250	2.5	1,000	25



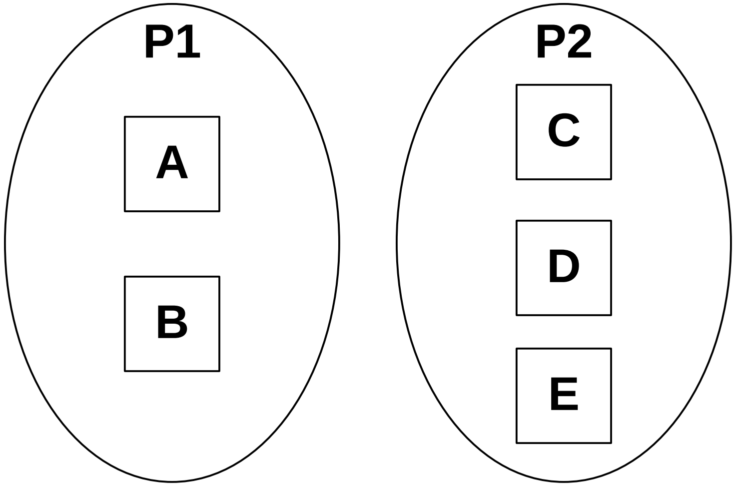


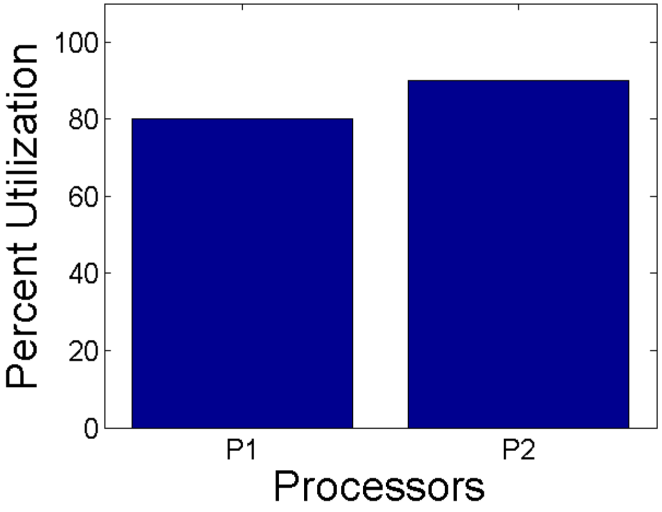
Designing the DeCoRAM Allocation Algorithm (1/5)

Basic Step 1: No fault tolerance

- Only primaries exist consuming WCET each
- Apply first-fit optimal bin-packing using the [Dhall:78] algorithm
- Consider sample task set shown
- Tasks arranged according to rate monotonic priorities

Task	WCET	WCSST	Period	Util
A	20	0.2	50	40
B	40	0.4	100	40
C	50	0.5	200	25
D	200	2	500	40
E	250	2.5	1,000	25





Designing the DeCoRAM Allocation Algorithm (1/5)

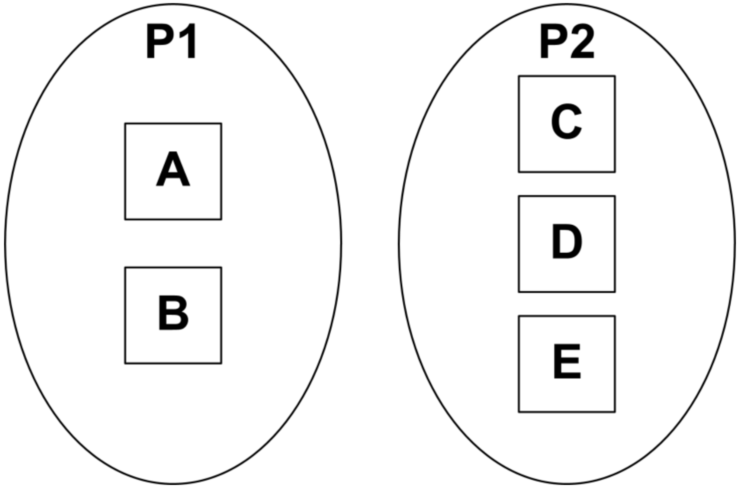
Basic Step 1: No fault tolerance

- Only primaries exist consuming WCET each
- Apply first-fit optimal bin-packing using the [Dhall:78] algorithm
- Consider sample task set shown
- Tasks arranged according to rate monotonic priorities

Task	WCET	WCSST	Period	Util
A	20	0.2	50	40
B	40	0.4	100	40
C	50	0.5	200	25
D	200	2	500	40
E	250	2.5	1,000	25

Outcome -> Lower bound established

- System is schedulable
- Uses minimum number of resources



RT & resource constraints satisfied; but no FT

Designing the DeCoRAM Allocation Algorithm (2/5)

Refinement 1: Introduce replica tasks

- Do not differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 replicas each
- Apply the [Dhall:78] algorithm

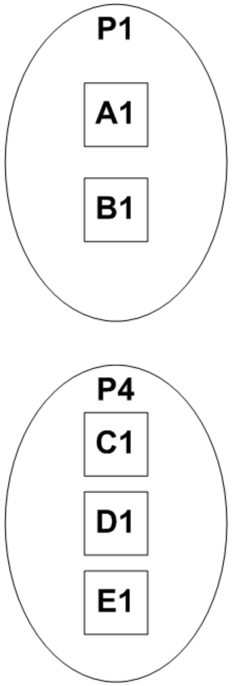
Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

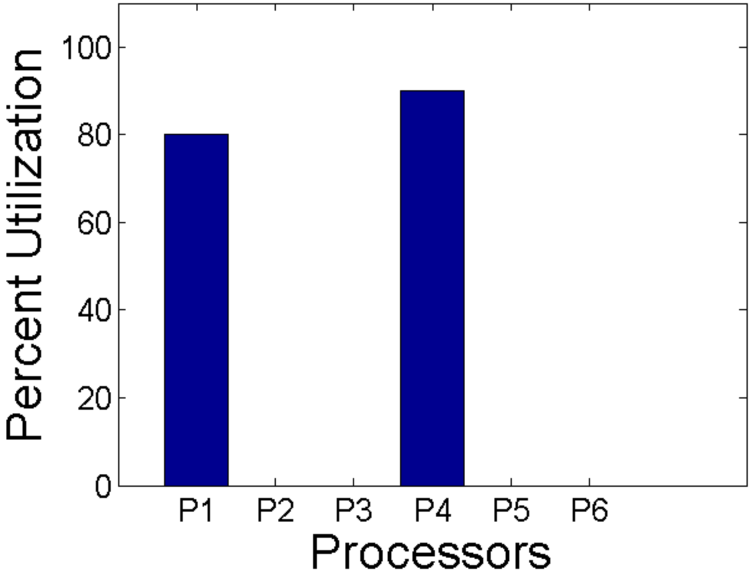
Designing the DeCoRAM Allocation Algorithm (2/5)

Refinement 1: Introduce replica tasks

- Do not differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000



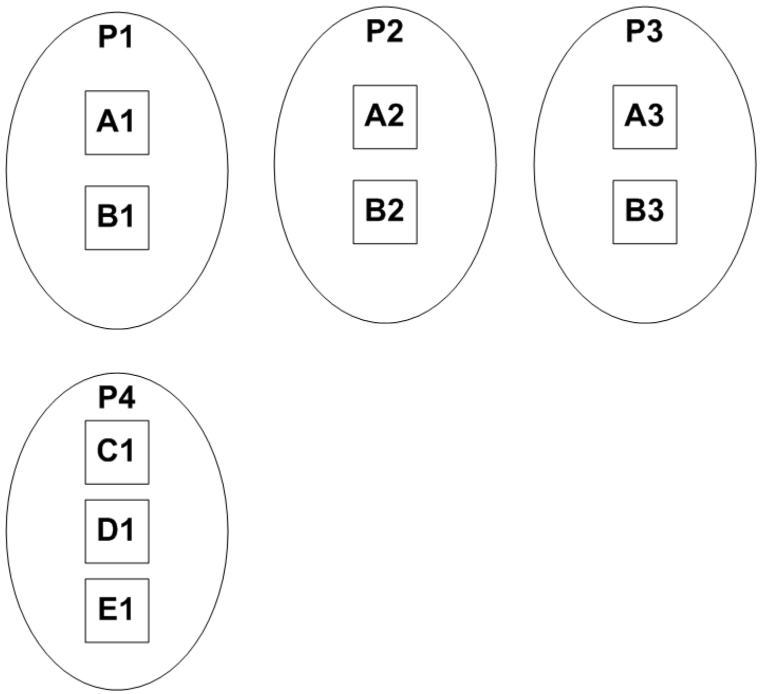


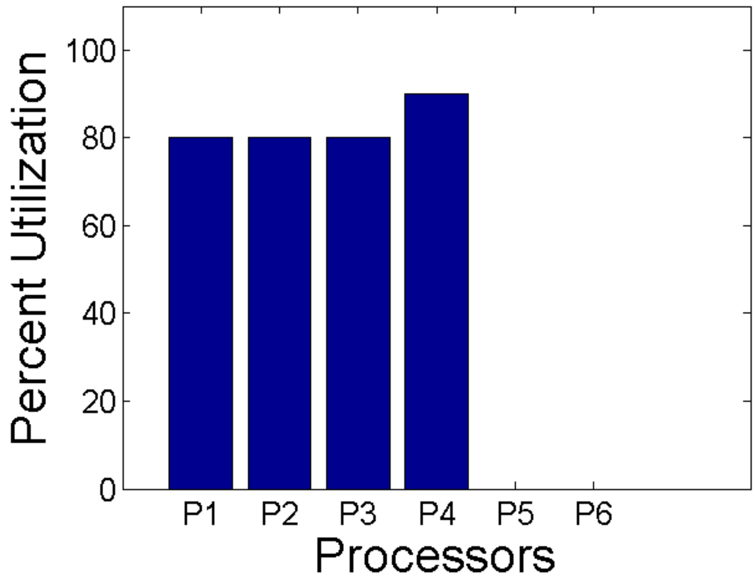
Designing the DeCoRAM Allocation Algorithm (2/5)

Refinement 1: Introduce replica tasks

- Do not differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000



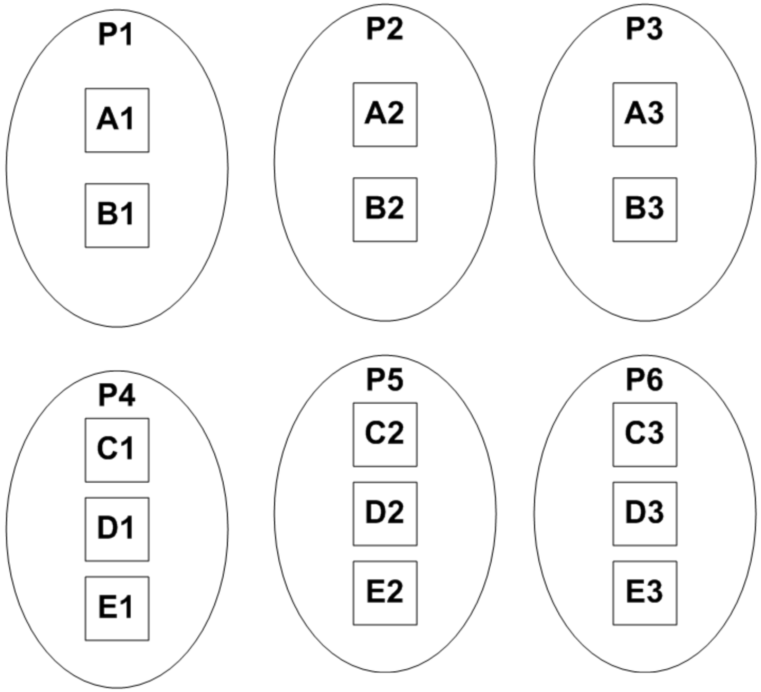


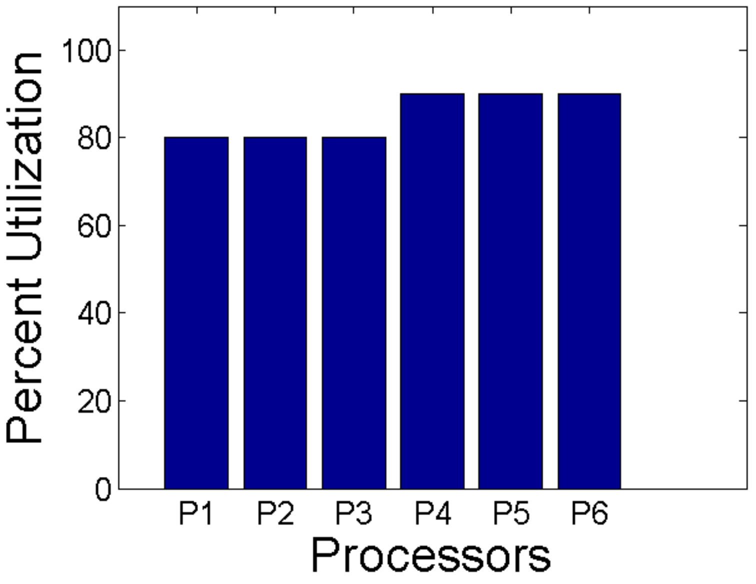
Designing the DeCoRAM Allocation Algorithm (2/5)

Refinement 1: Introduce replica tasks

- Do not differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000





Designing the DeCoRAM Allocation Algorithm (2/5)

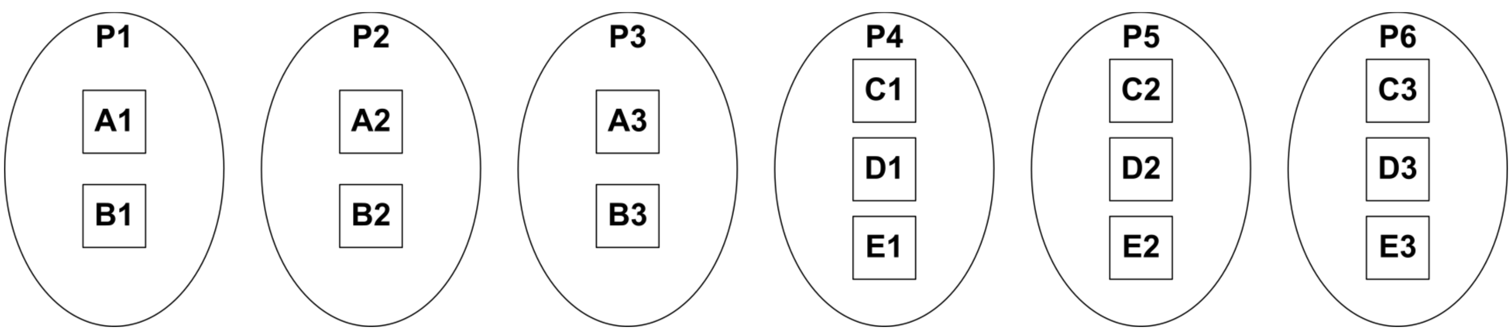
Refinement 1: Introduce replica tasks

- Do not differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

Outcome -> Upper bound is established

- A RT-FT solution is created – but with Active replication
- System is schedulable
- Demonstrates upper bound on number of resources needed



Minimize resource using passive replication

Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

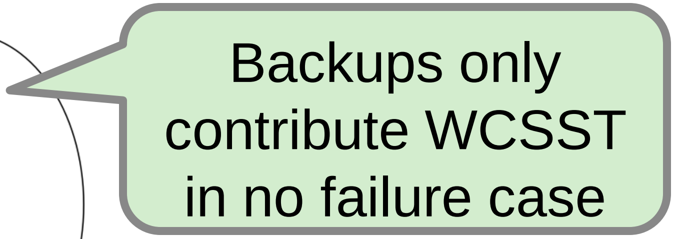
Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

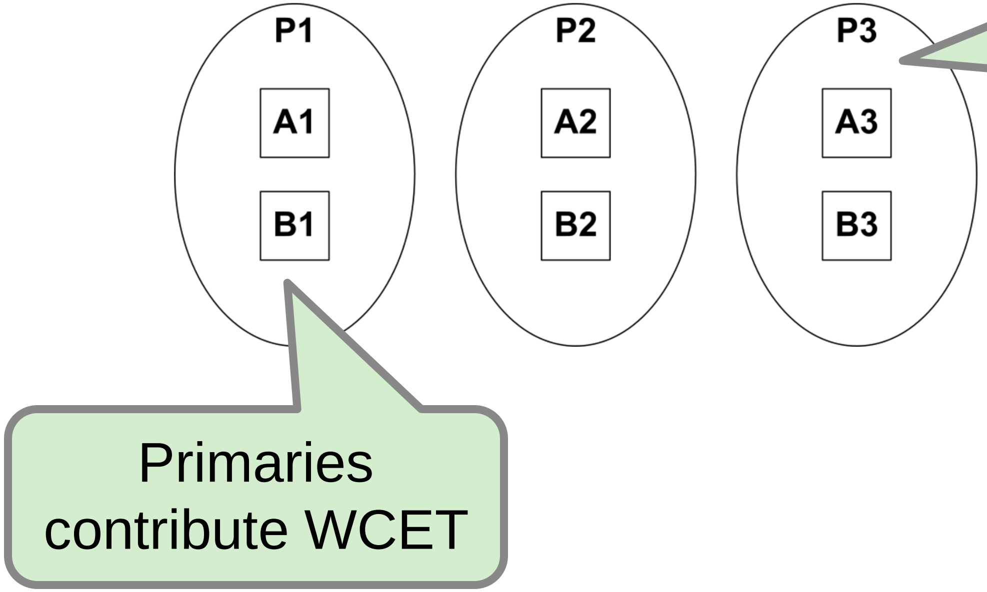
Designing the DeCoRAM Allocation Algorithm (3/5)

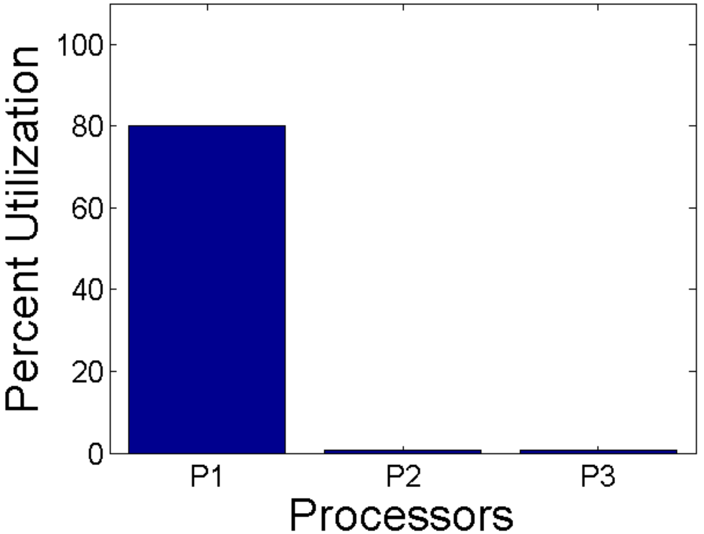
Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000





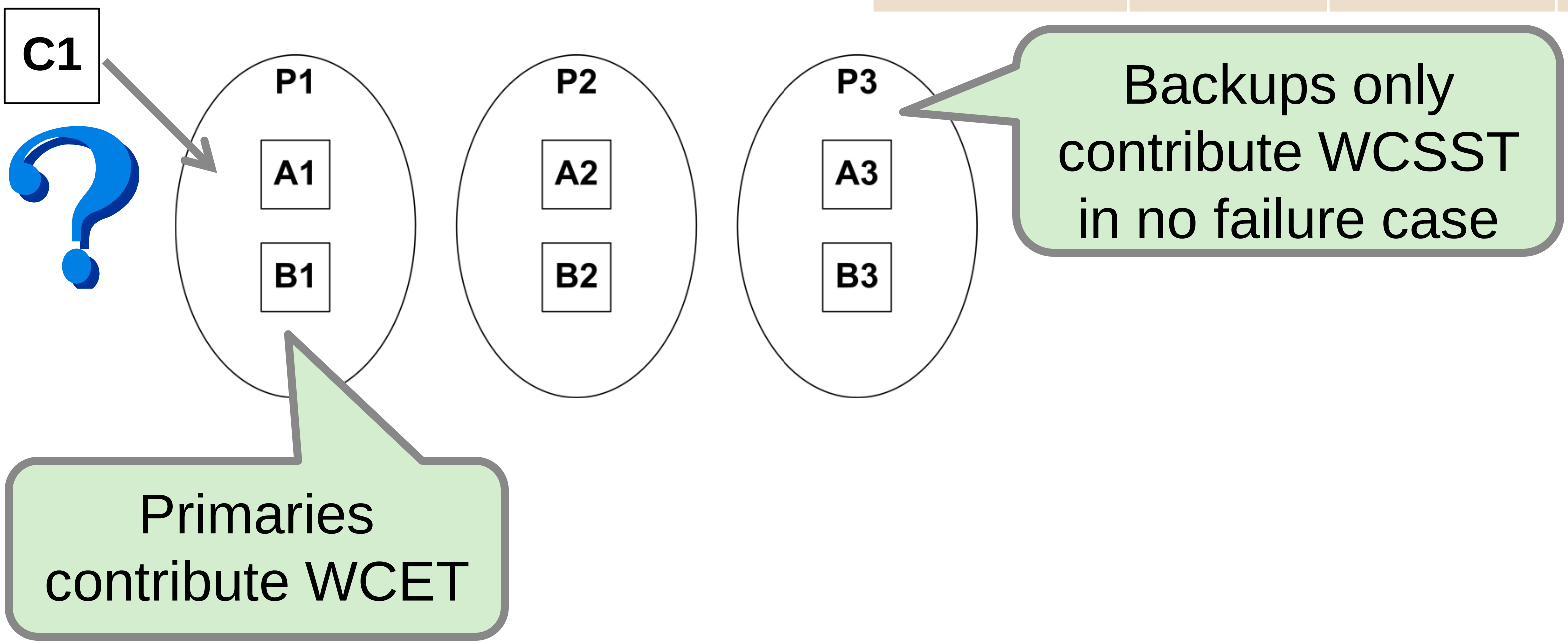


Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

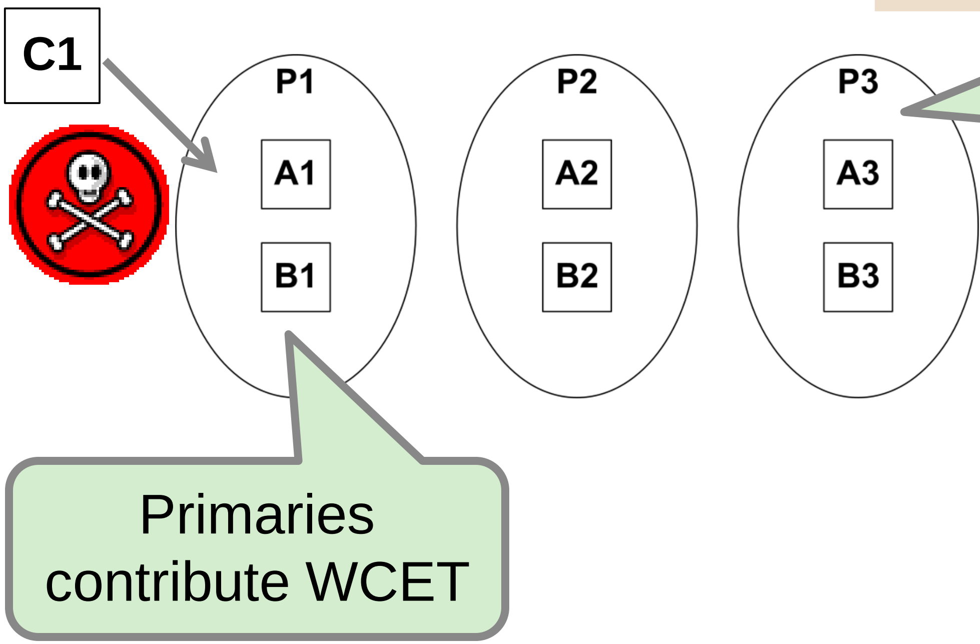


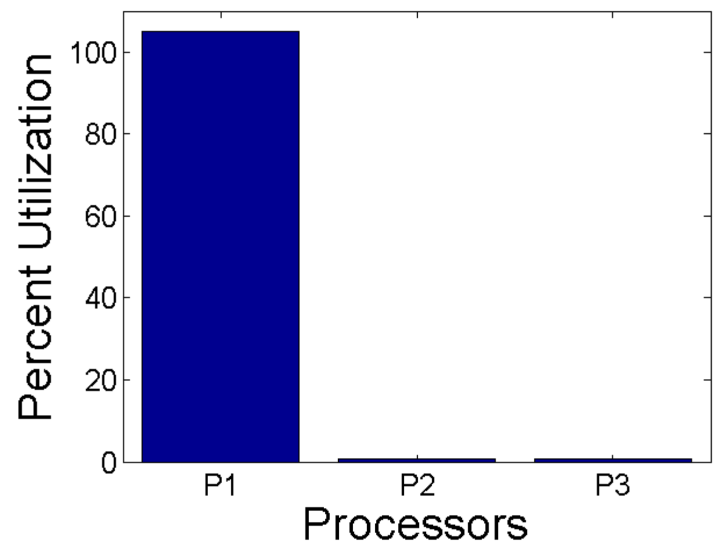
Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000



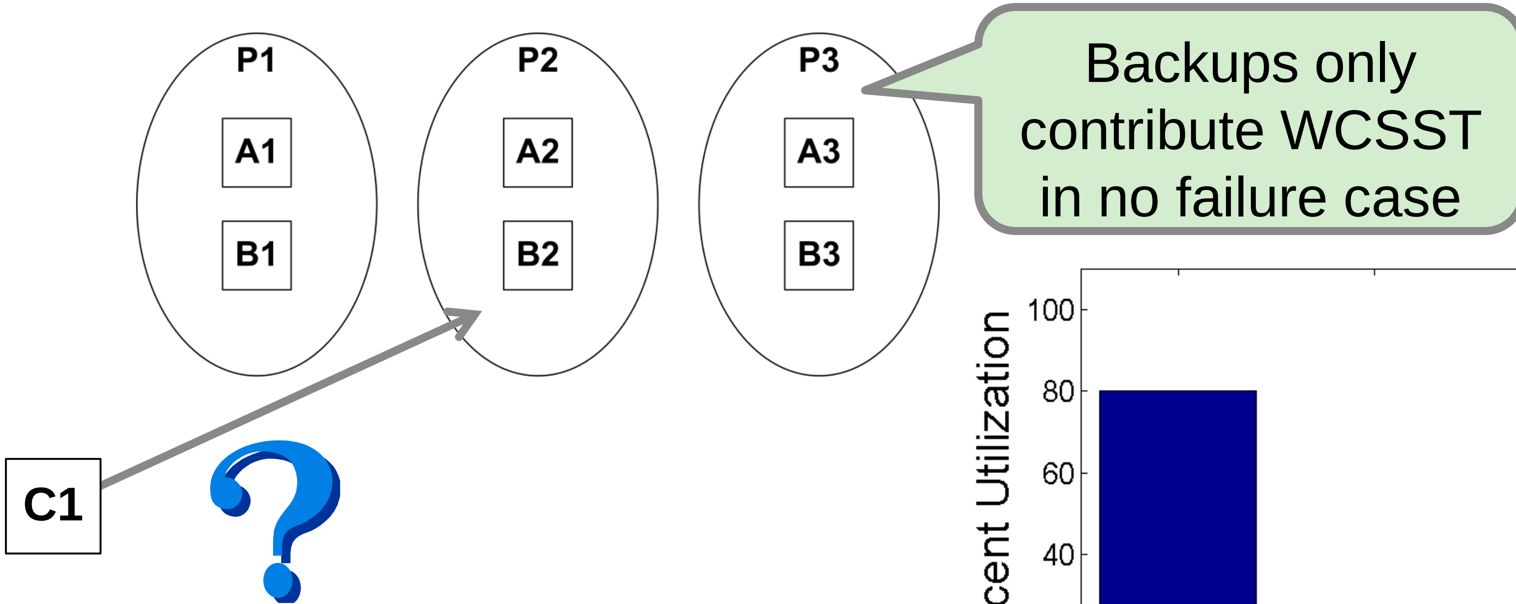


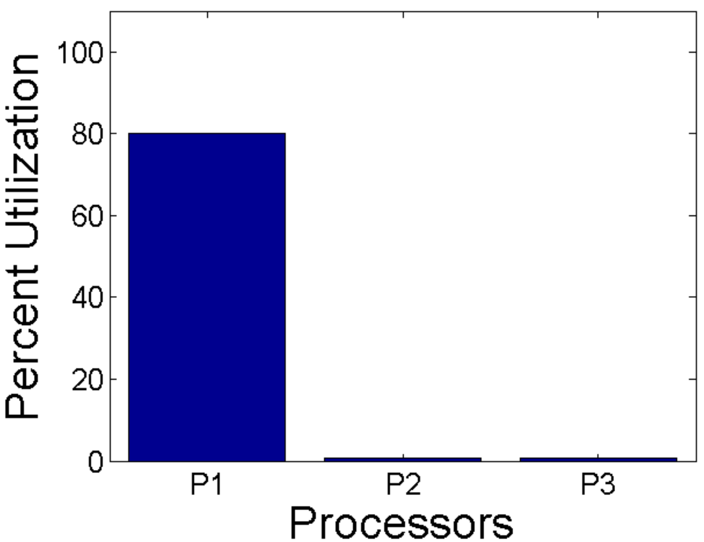
Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000



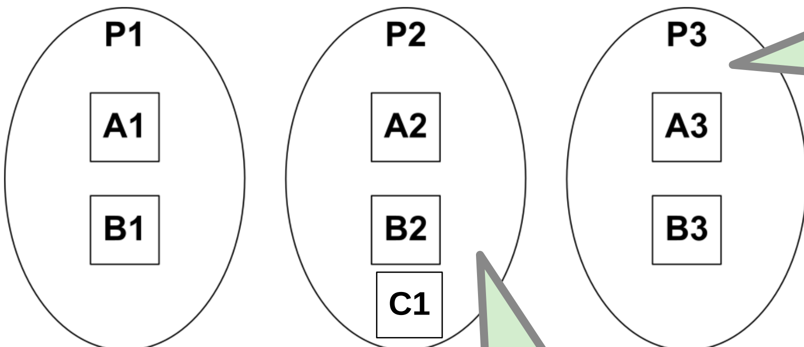


Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

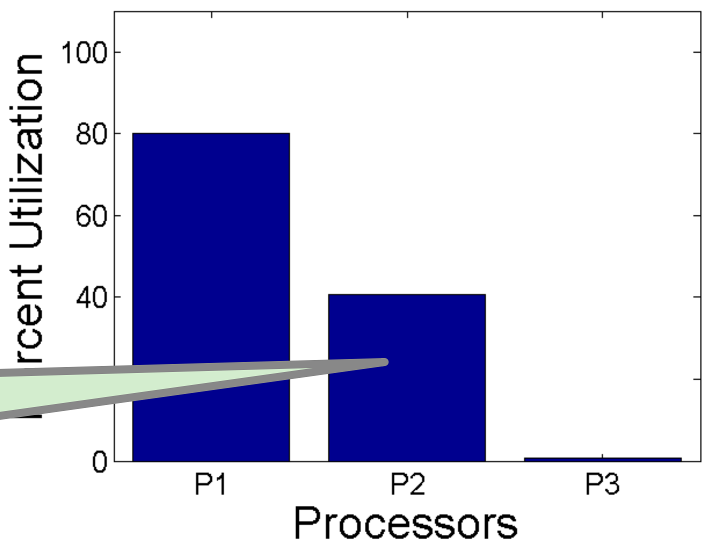
- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000



Backups only contribute WCSST in no failure case

Allocation is fine when A2/B2 are backups

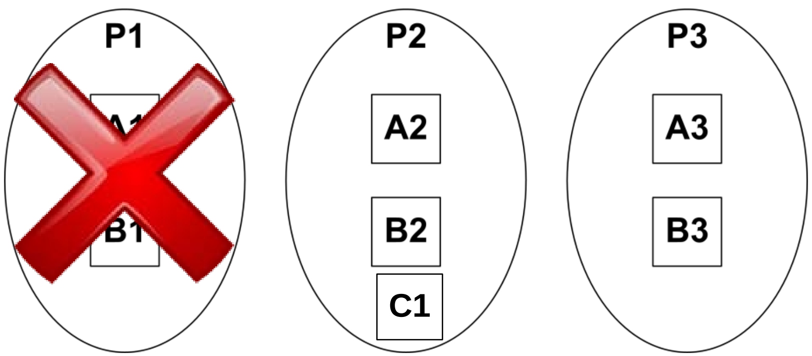


Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

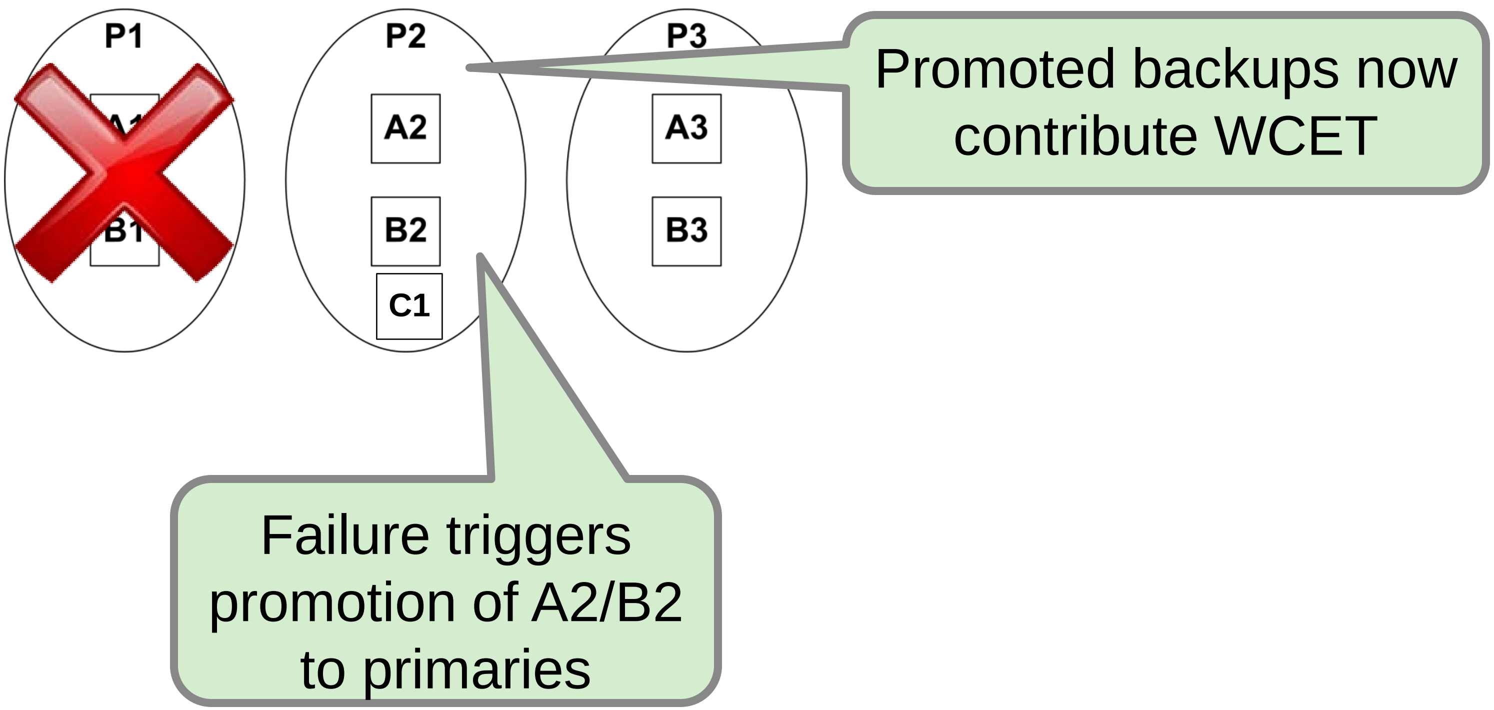


Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

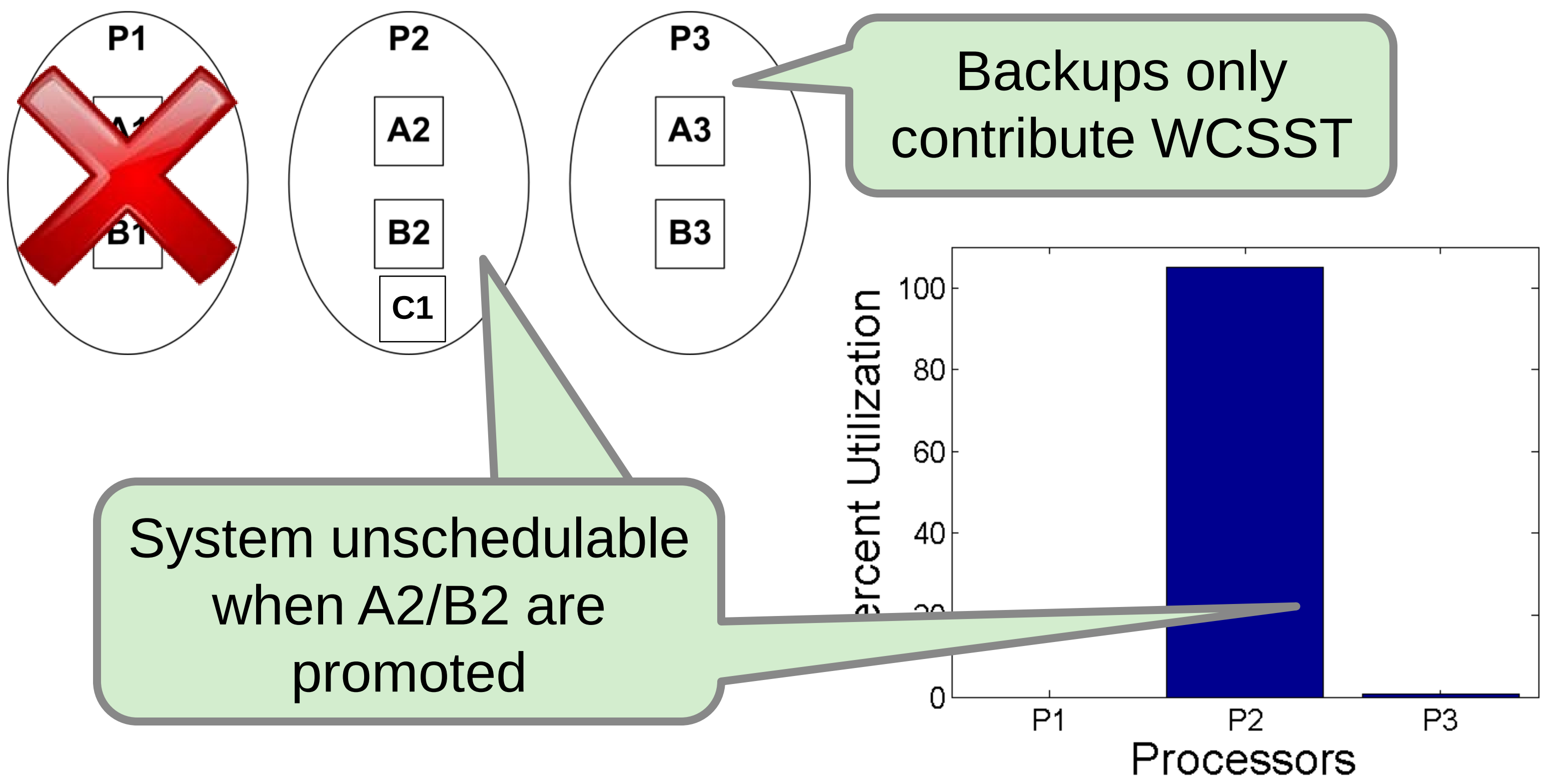


Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000



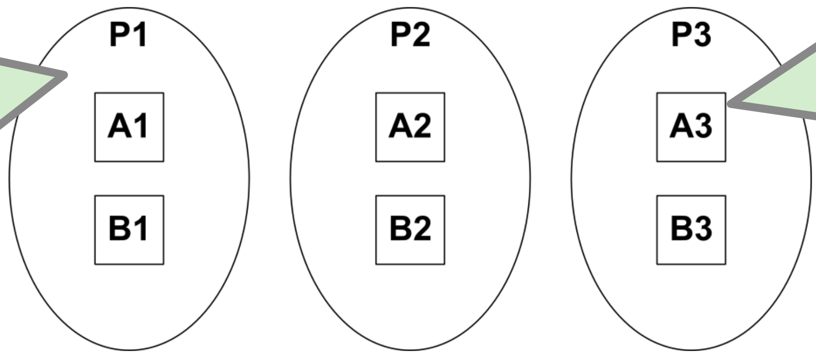
Designing the DeCoRAM Allocation Algorithm (3/5)

Refinement 2: Passive replication

- Differentiate between primary & replicas
- Assume tolerance to 2 failures => 2 additional backup replicas each
- Apply the [Dhall:78] algorithm

Task	WCET	WCSST	Period
A1,A2,A3	20	0.2	50
B1,B2,B3	40	0.4	100
C1,C2,C3	50	0.5	200
D1,D2,D3	200	2	500
E1,E2,E3	250	2.5	1,000

C1/D1/E1 cannot be placed here -- unschedulable



C1/D1/E1 may be placed on P2 or P3 as long as there are no failures

Outcome

- Resource minimization & system schedulability feasible in non faulty scenarios only -- because backup contributes only WCSST

- **Unrealistic not to expect failures**
- **Need a way to consider failures & find which backup will be promoted to primary (contributing WCET)?**

Designing the DeCoRAM Allocation Algorithm (4/5)

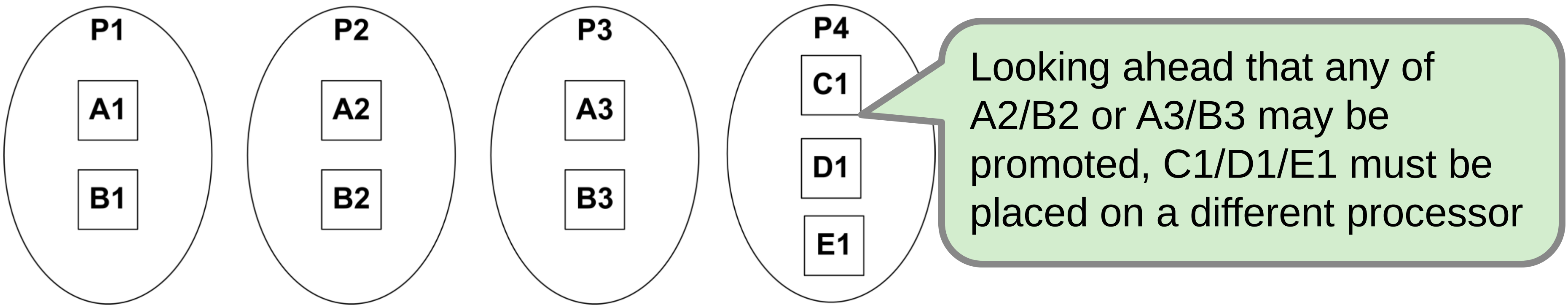
Refinement 3: Enable the offline algorithm to consider failures

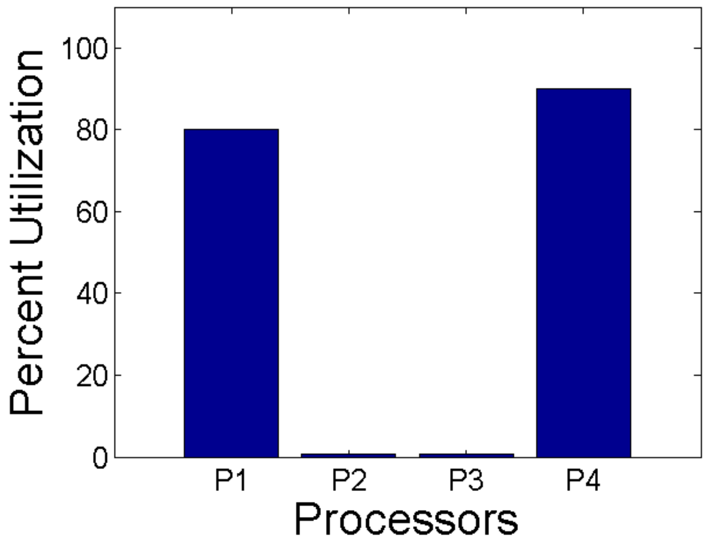
- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant

Designing the DeCoRAM Allocation Algorithm (4/5)

Refinement 3: Enable the offline algorithm to consider failures

- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant

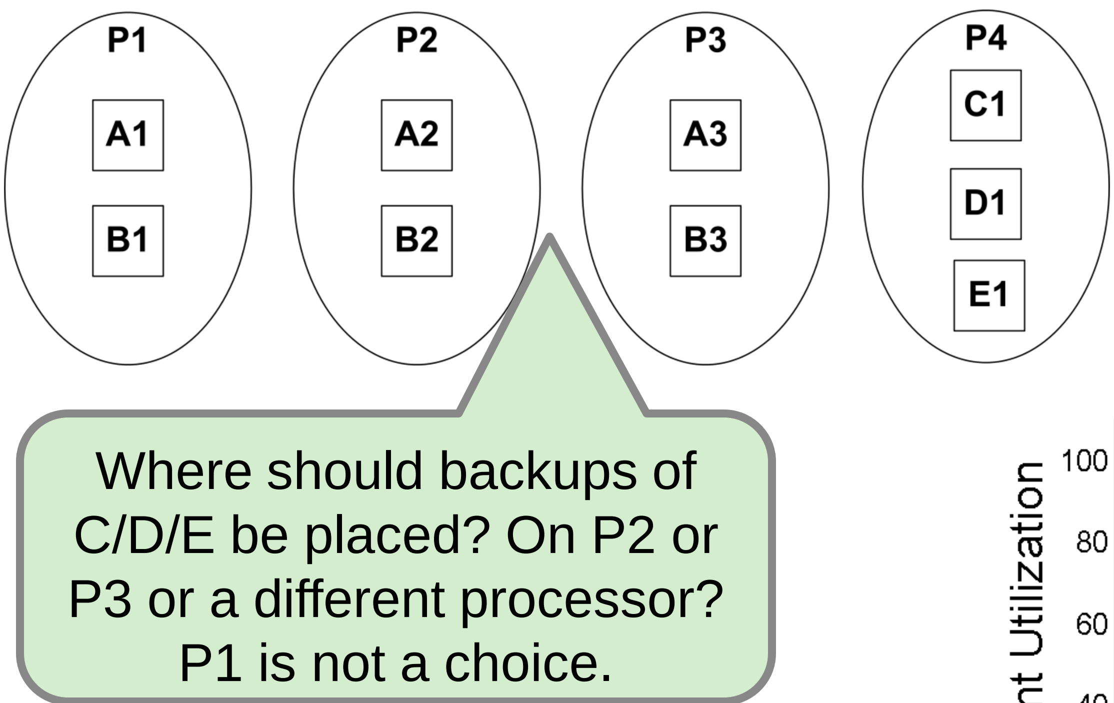


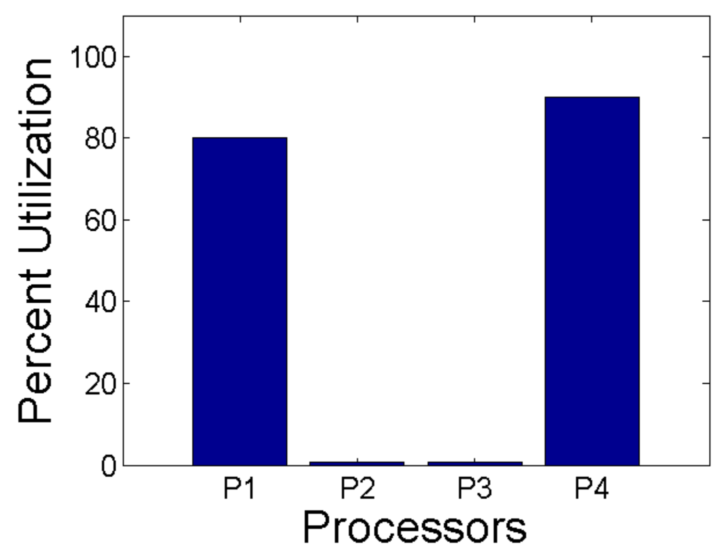


Designing the DeCoRAM Allocation Algorithm (4/5)

Refinement 3: Enable the offline algorithm to consider failures

- “Look ahead” at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant

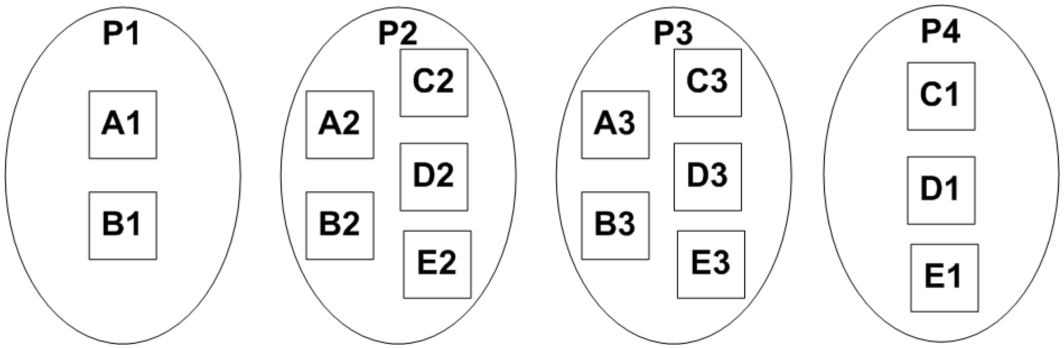




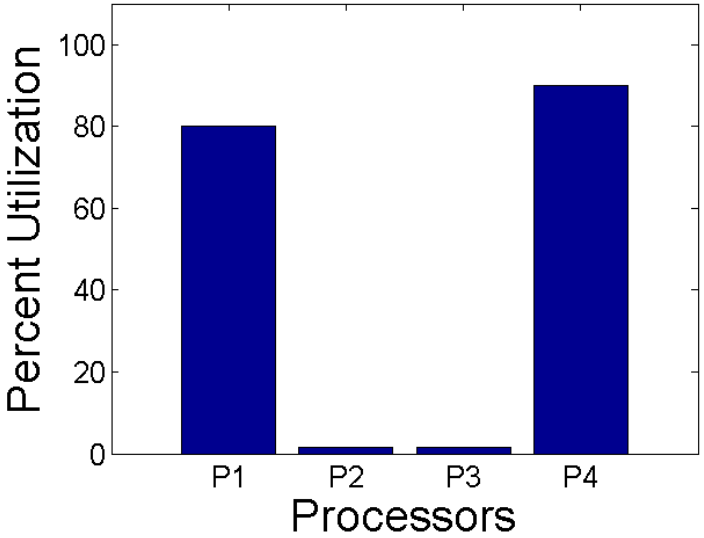
Designing the DeCoRAM Allocation Algorithm (4/5)

Refinement 3: Enable the offline algorithm to consider failures

- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant



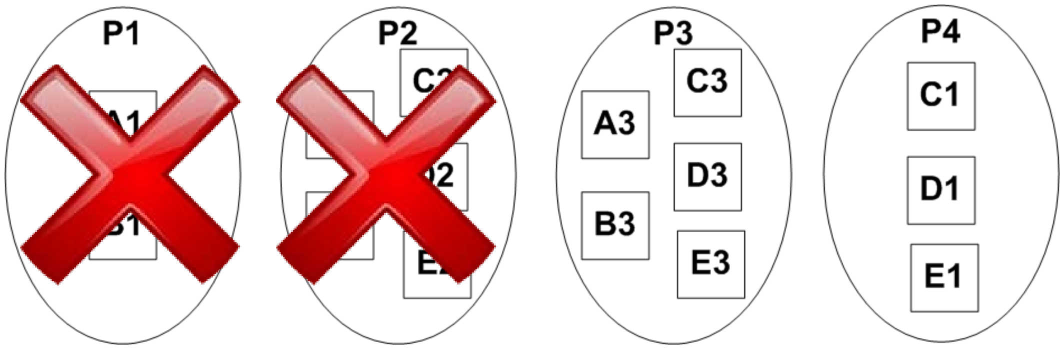
- Suppose the allocation of the backups of C/D/E are as shown
- We now look ahead for any 2 failure combinations



Designing the DeCoRAM Allocation Algorithm (4/5)

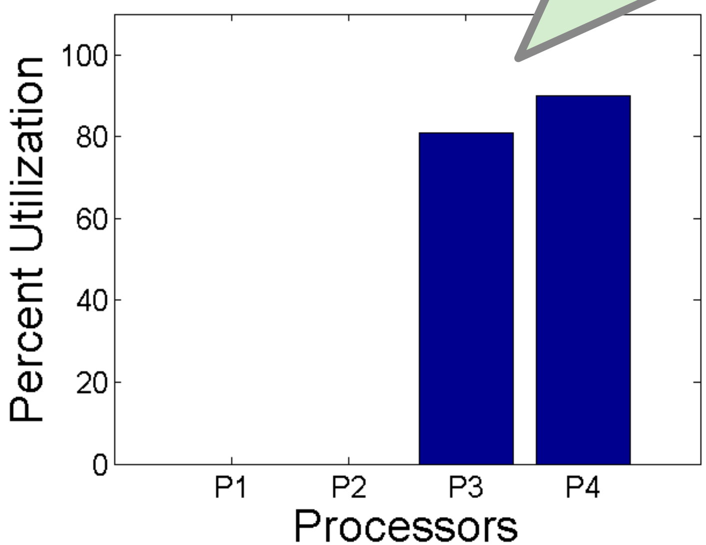
Refinement 3: Enable the offline algorithm to consider failures

- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant



- Suppose P1 & P2 were to fail
- A3 & B3 will be promoted

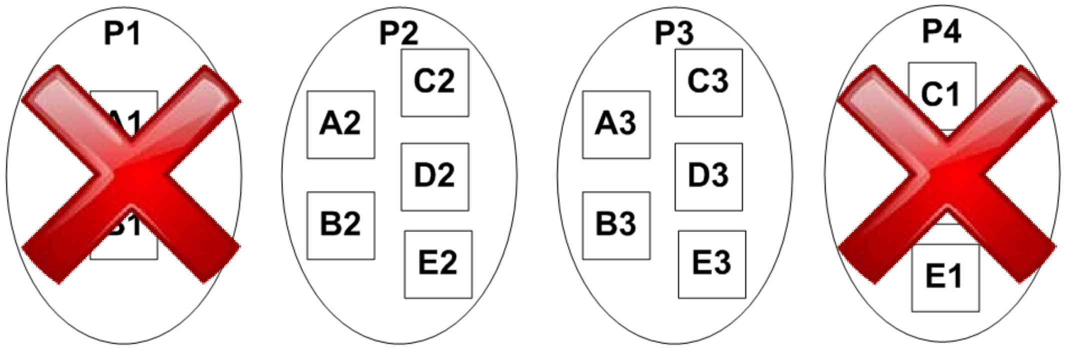
Schedule is feasible => original placement decision was OK



Designing the DeCoRAM Allocation Algorithm (4/5)

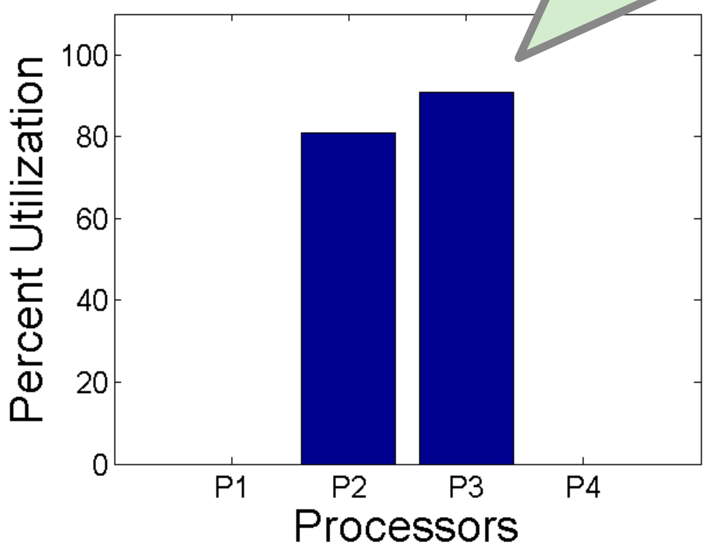
Refinement 3: Enable the offline algorithm to consider failures

- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant



- Suppose P1 & P4 were to fail
- Suppose A2 & B2 on P2 were to be promoted, while C3, D3 & E3 on P3 were to be promoted

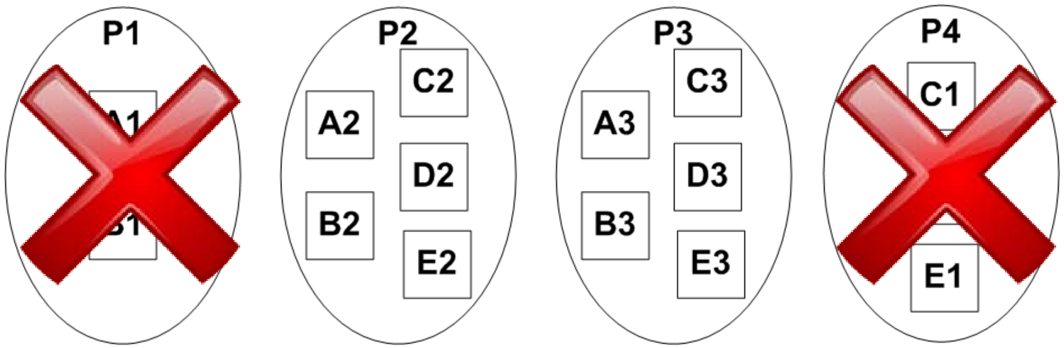
Schedule is feasible => original placement decision was OK



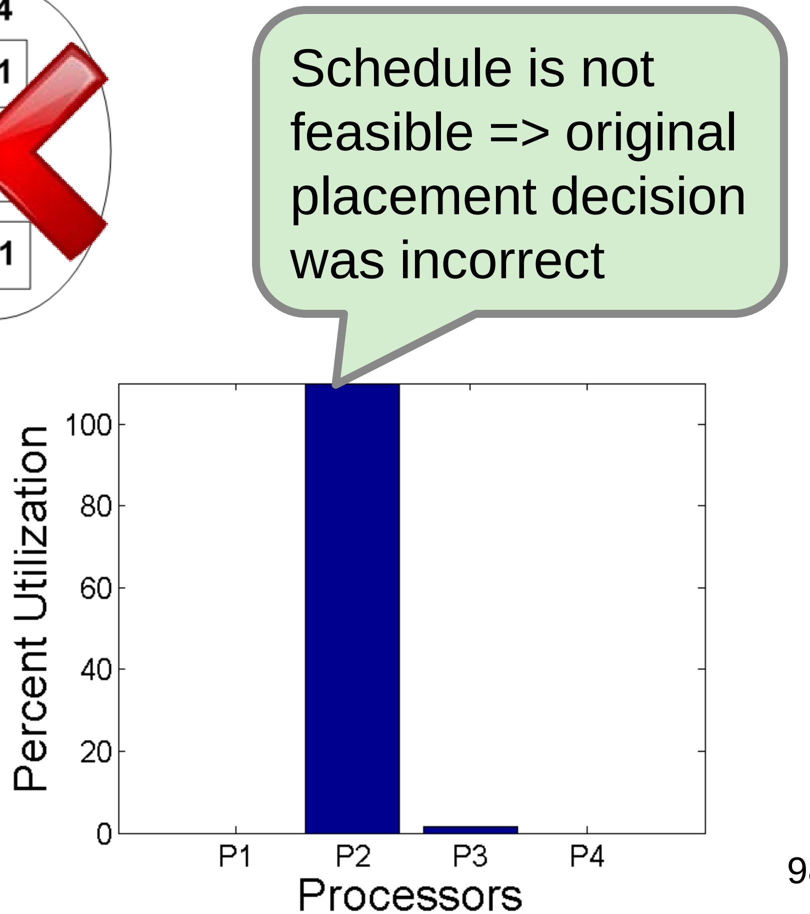
Designing the DeCoRAM Allocation Algorithm (4/5)

Refinement 3: Enable the offline algorithm to consider failures

- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant



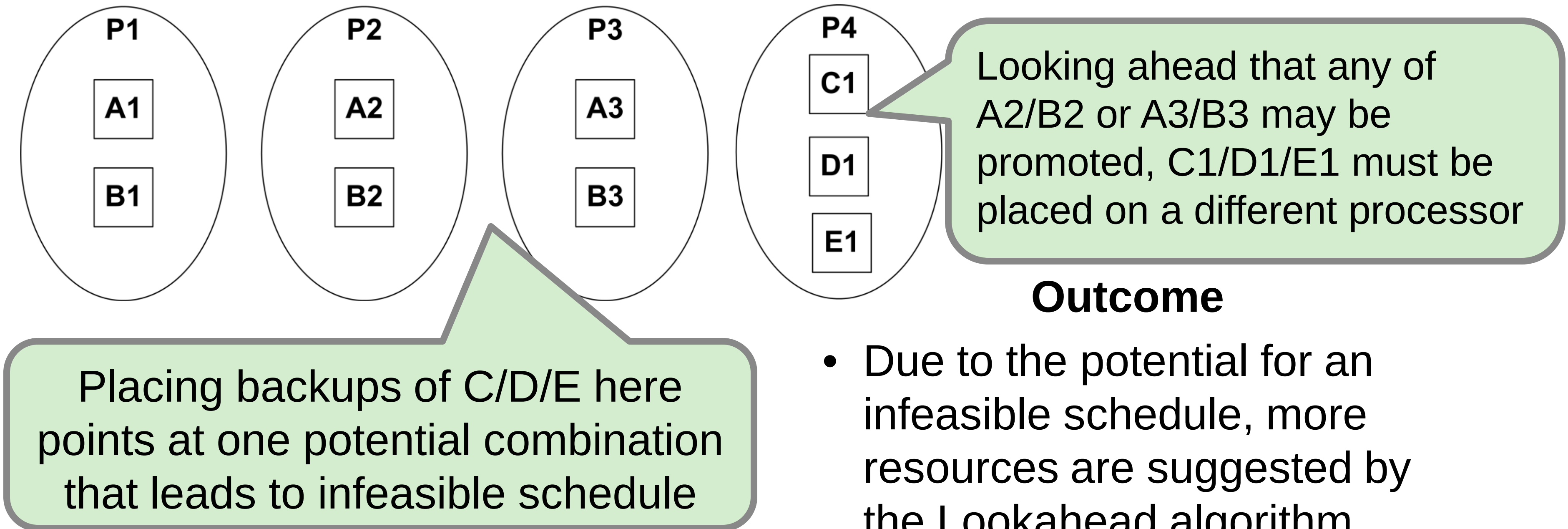
- Suppose P1 & P4 were to fail
- Suppose A2, B2, C2, D2 & E2 on P2 were to be promoted



Designing the DeCoRAM Allocation Algorithm (4/5)

Refinement 3: Enable the offline algorithm to consider failures

- "Look ahead" at failure scenarios of already allocated tasks & replicas determining worst case impact on a given processor
- Feasible to do this because system properties are invariant



Outcome

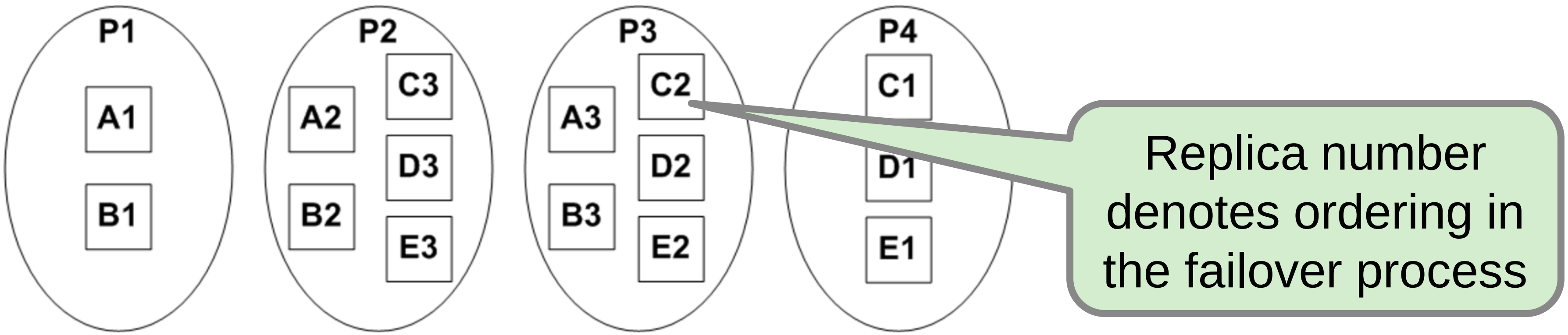
- Due to the potential for an infeasible schedule, more resources are suggested by the Lookahead algorithm

- **Look-ahead strategy cannot determine impact of multiple uncorrelated failures that may make system unschedulable**

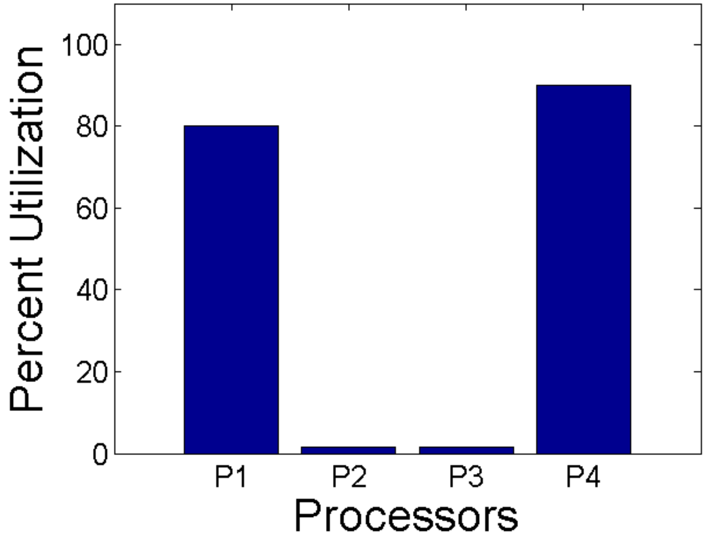
Designing the DeCoRAM Allocation Algorithm (5/5)

Refinement 4: Restrict the order in which failover targets are chosen

- Utilize a rank order of replicas to dictate how failover happens
- Enables the Lookahead algorithm to overbook resources due to guarantees that no two uncorrelated failures will make the system unschedulable



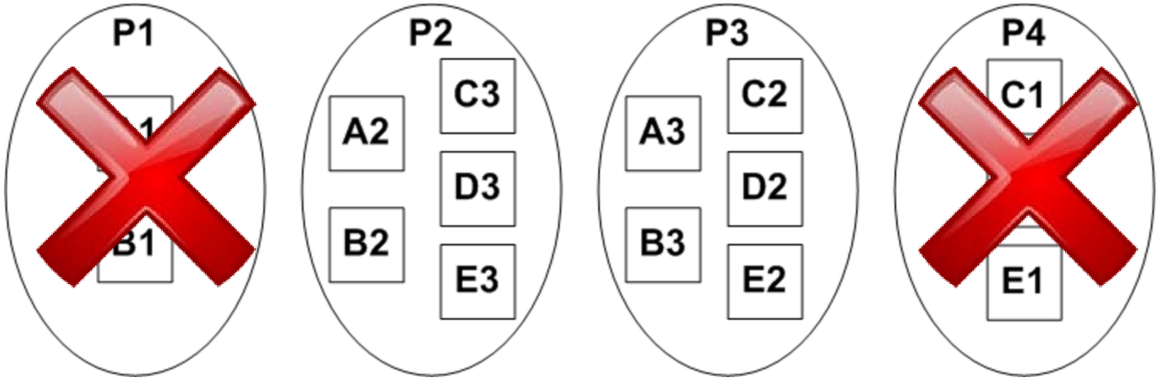
- Suppose the replica allocation is as shown (slightly diff from before)
- Replica numbers indicate order in the failover process



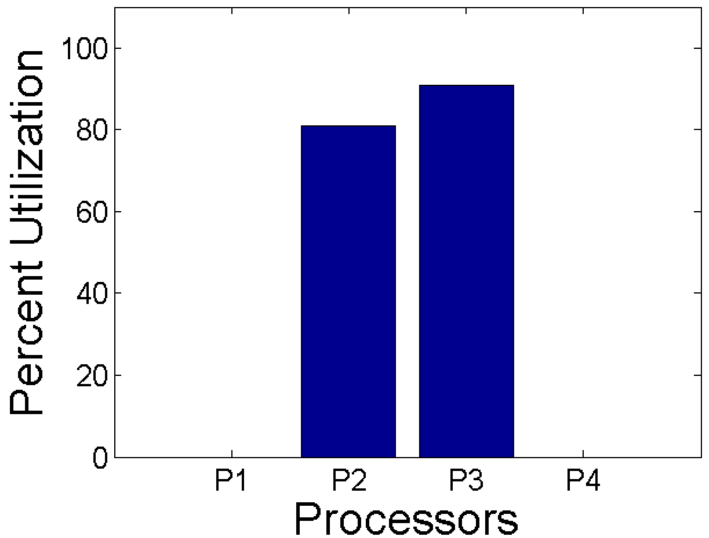
Designing the DeCoRAM Allocation Algorithm (5/5)

Refinement 4: Restrict the order in which failover targets are chosen

- Utilize a rank order of replicas to dictate how failover happens
- Enables the Lookahead algorithm to overbook resources due to guarantees that no two uncorrelated failures will make the system unschedulable



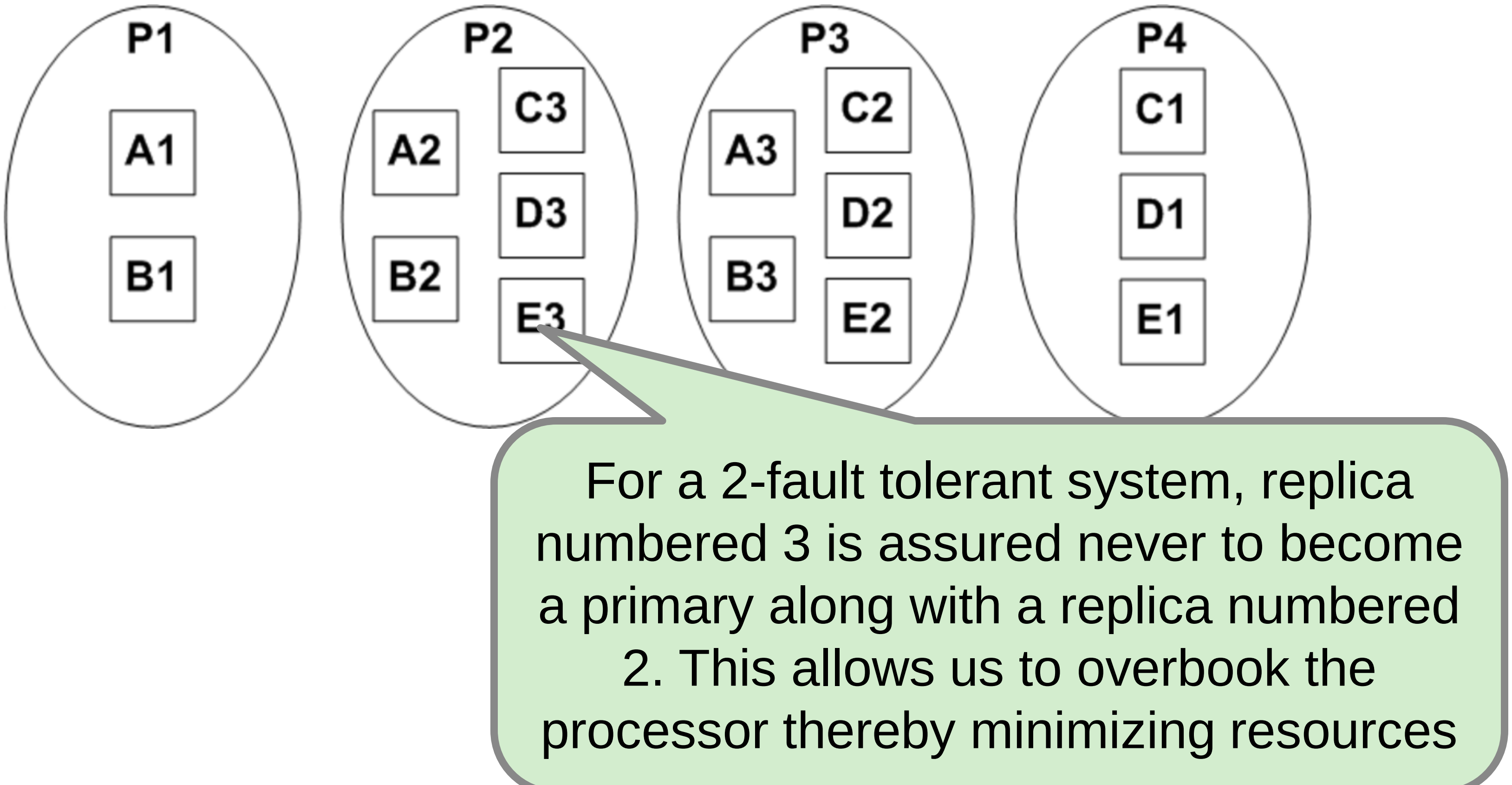
- Suppose P1 & P4 were to fail (**the interesting case**)
- A2 & B2 on P2, & C2, D2, E2 on P3 will be chosen as failover targets due to the restrictions imposed
- Never can C3, D3, E3 become primaries along with A2 & B2 unless more than two failures occur



Designing the DeCoRAM Allocation Algorithm (5/5)

Refinement 4: Restrict the order in which failover targets are chosen

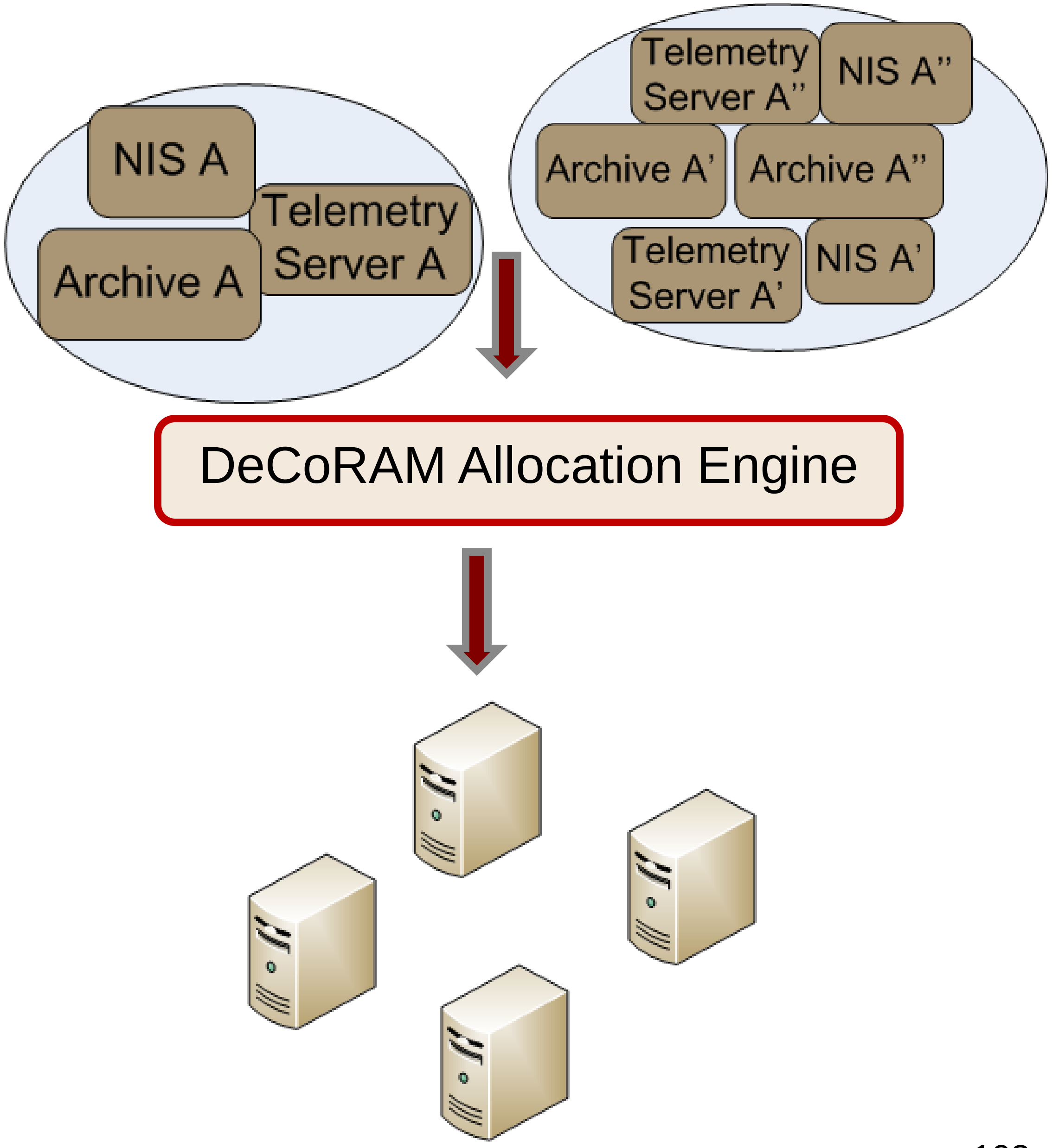
- Utilize a rank order of replicas to dictate how failover happens
- Enables the Lookahead algorithm to overbook resources due to guarantees that no two uncorrelated failures will make the system unschedulable



Resources minimized from 6 to 4 while assuring both RT & FT

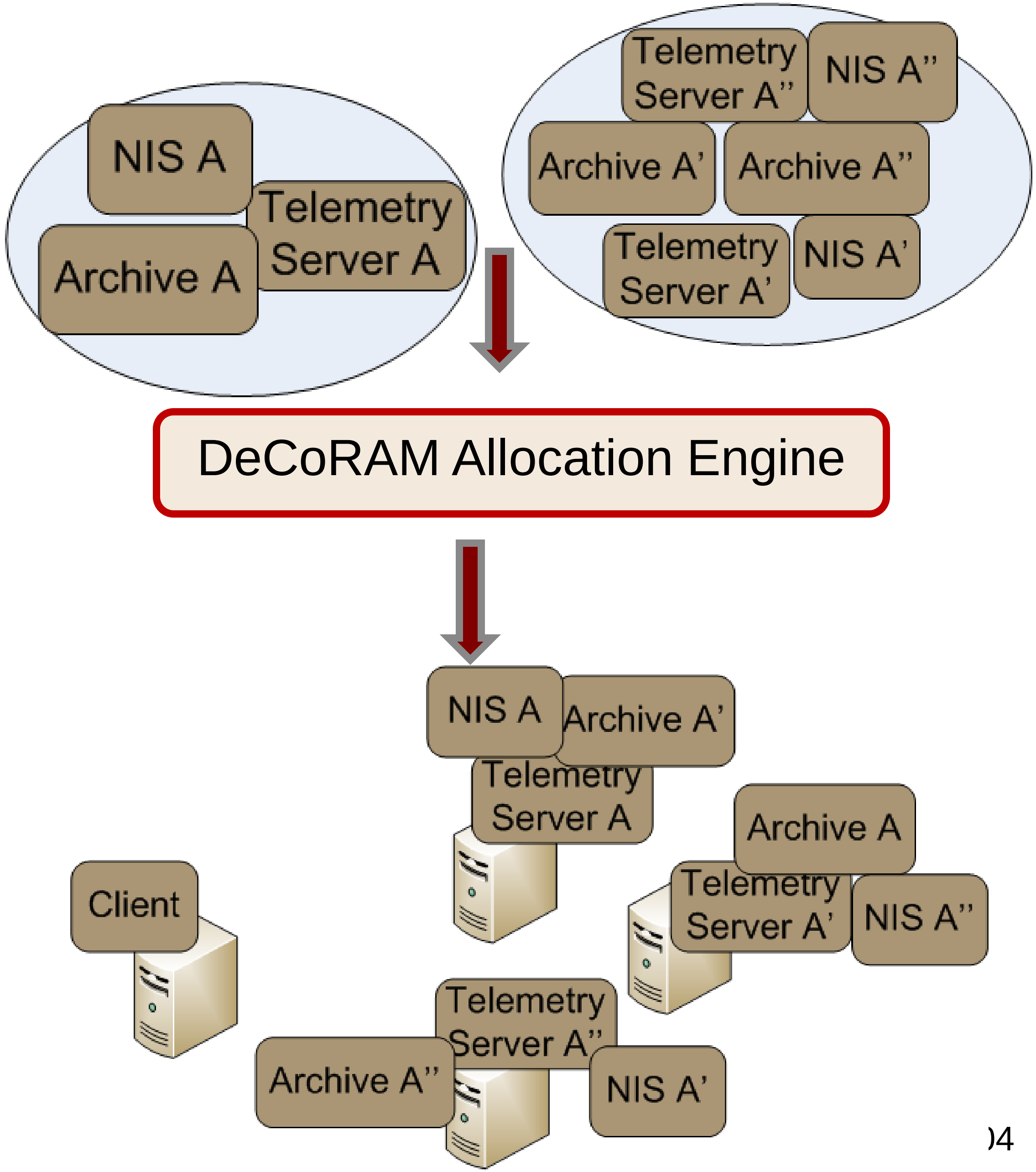
DeCoRAM Evaluation Criteria

- **Hypothesis** – DeCoRAM's Failure-aware Look-ahead Feasibility algorithm allocates applications & replicas to hosts while minimizing the number of processors utilized
  - number of processors utilized is lesser than the number of processors utilized using active replication



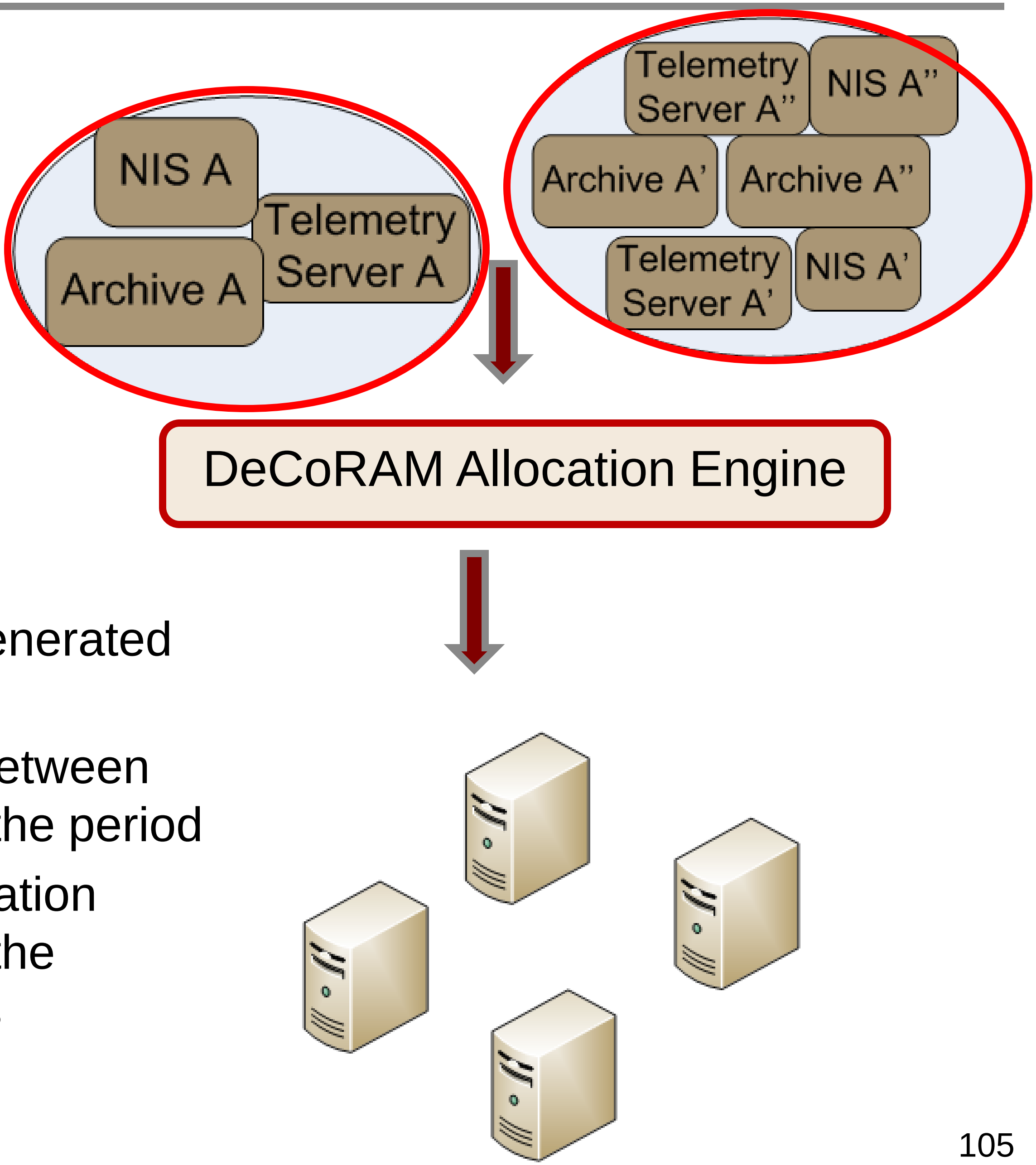
DeCoRAM Evaluation Hypothesis

- **Hypothesis** – DeCoRAM's Failure-aware Look-ahead Feasibility algorithm allocates applications & replicas to hosts while minimizing the number of processors utilized
  - number of processors utilized is lesser than the number of processors utilized using active replication
- Deployment-time configured real-time fault-tolerance solution works at runtime when failures occur
  - none of the applications lose high availability & timeliness assurances



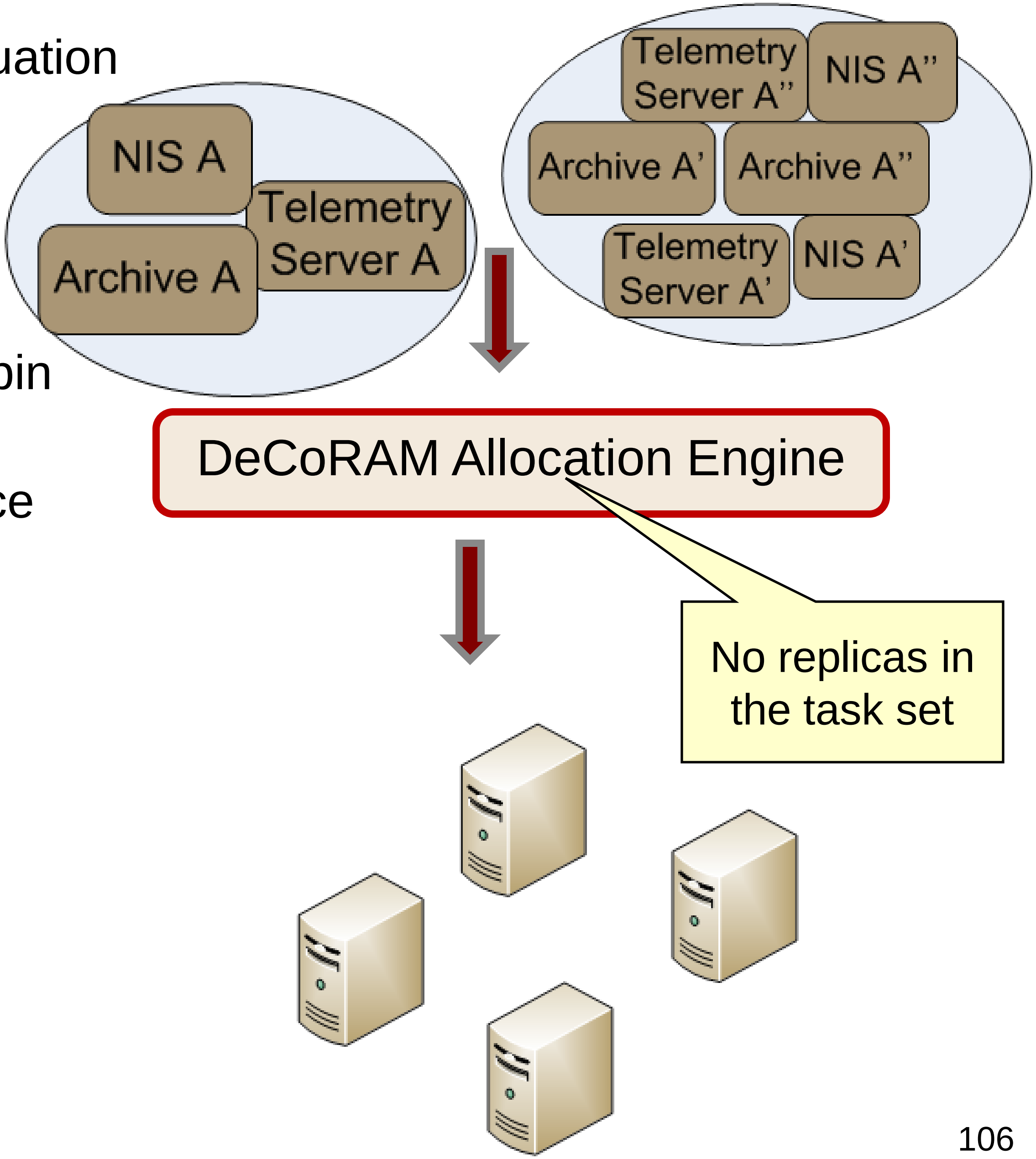
Experiment Configurations

- Determine # of processors utilized by
  - varying number of tasks dimension)
  - varying the number of replicas (FT dimension)
  - varying the maximum CPU utilization of any task in the task set
  - periods of tasks randomly generated between 1ms & 1000ms
    - each task execution time between 0% & maximum load % of the period
    - each task state synchronization time between 1% & 2% of the worst case execution times



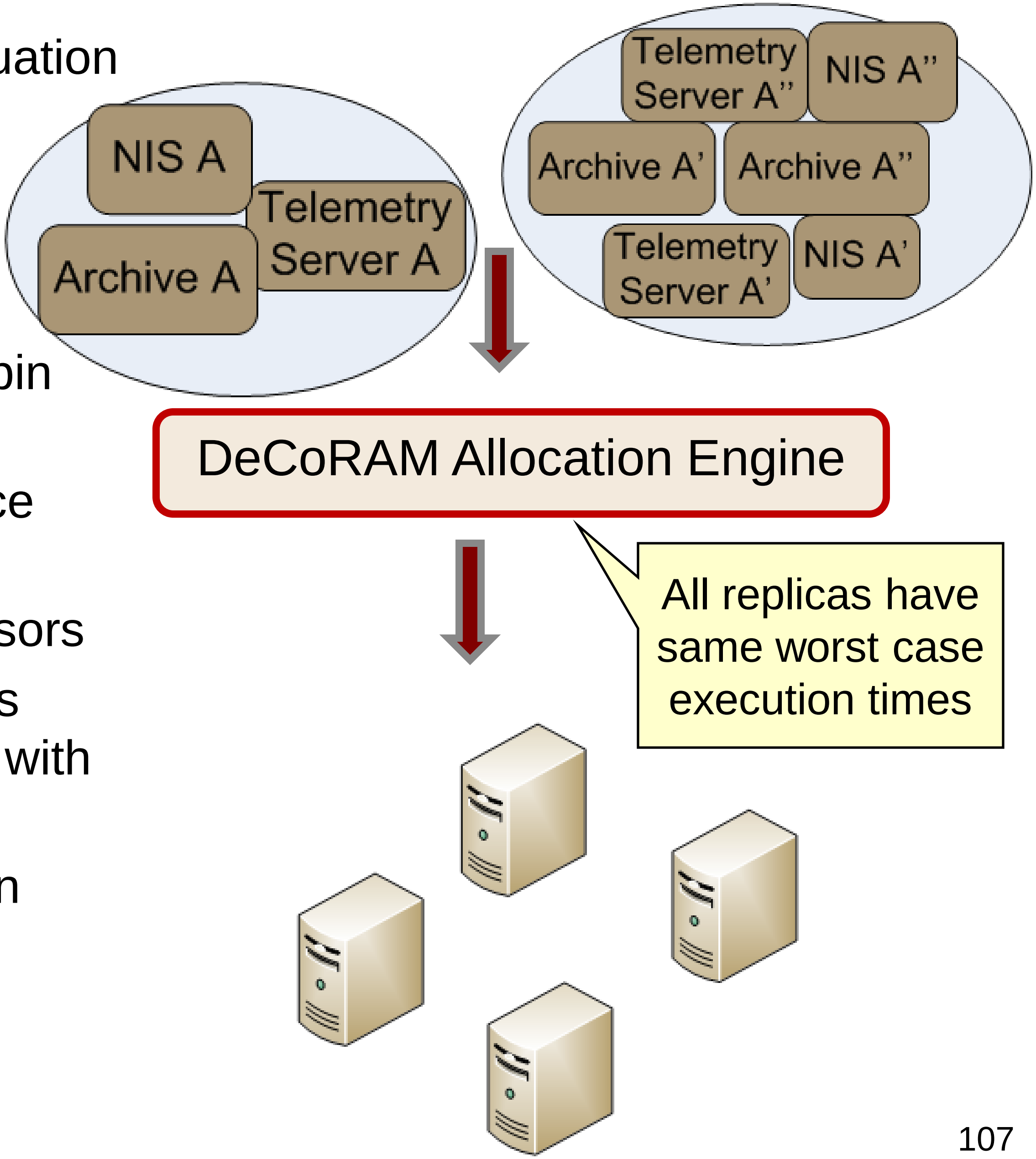
Comparison Schemes

- Comparison schemes for evaluation
  - lower bound on number of processors utilized
    - Implementing the optimal allocation algorithm in [Dhall:78] - uses First Fit bin packing scheme
      - Optimal no fault-tolerance scenario (No FT)



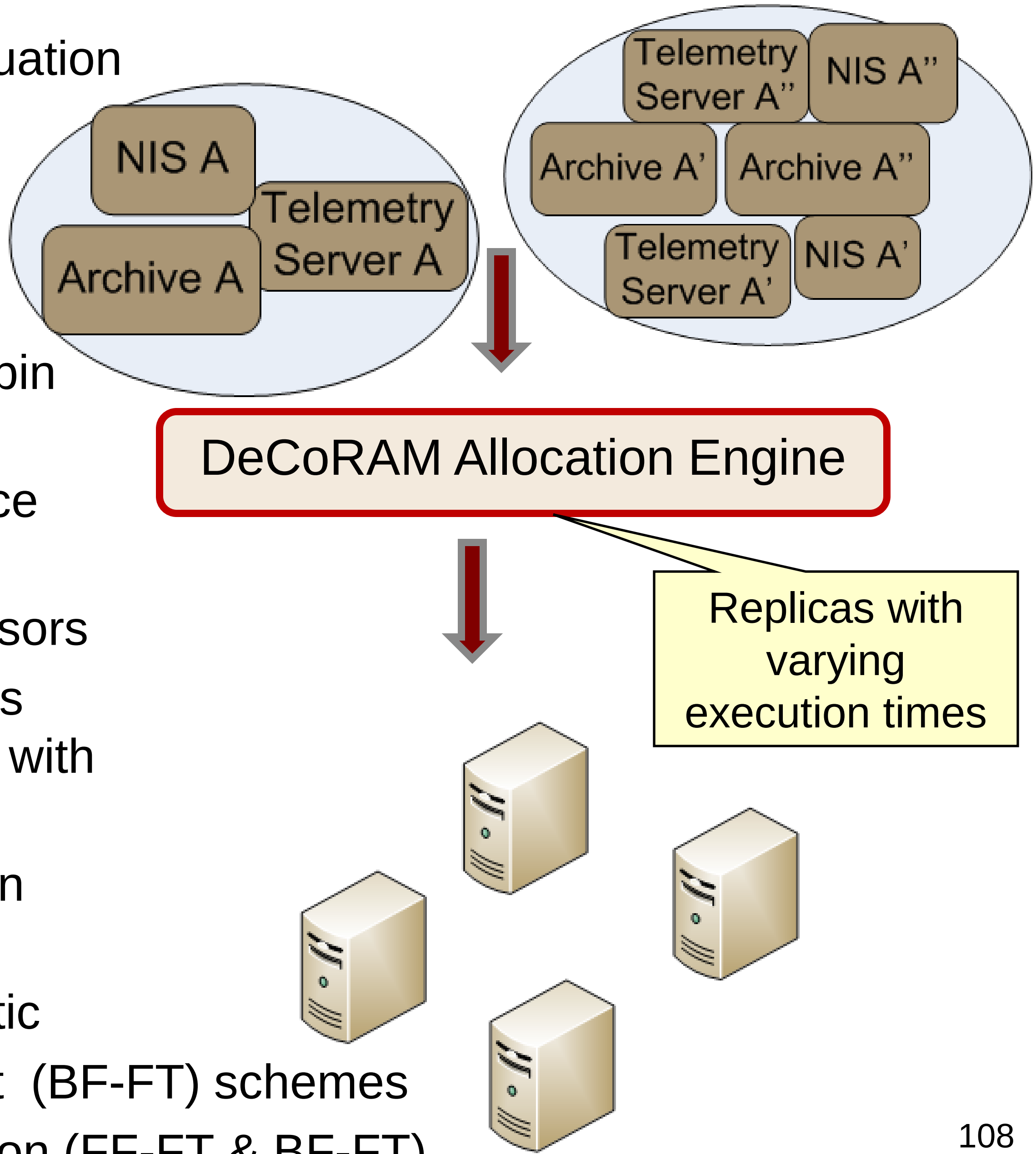
Comparison Schemes

- Comparison schemes for evaluation
  - lower bound on number of processors utilized
    - Implementing the optimal allocation algorithm in [Dhall:78] - uses First Fit bin packing scheme
      - Optimal no fault-tolerance scenario (No FT)
  - Upper bound on # of processors
    - Multiplying # of processors utilized in the No FT case with # of replicas
      - Optimal active replication scenario (AFT)

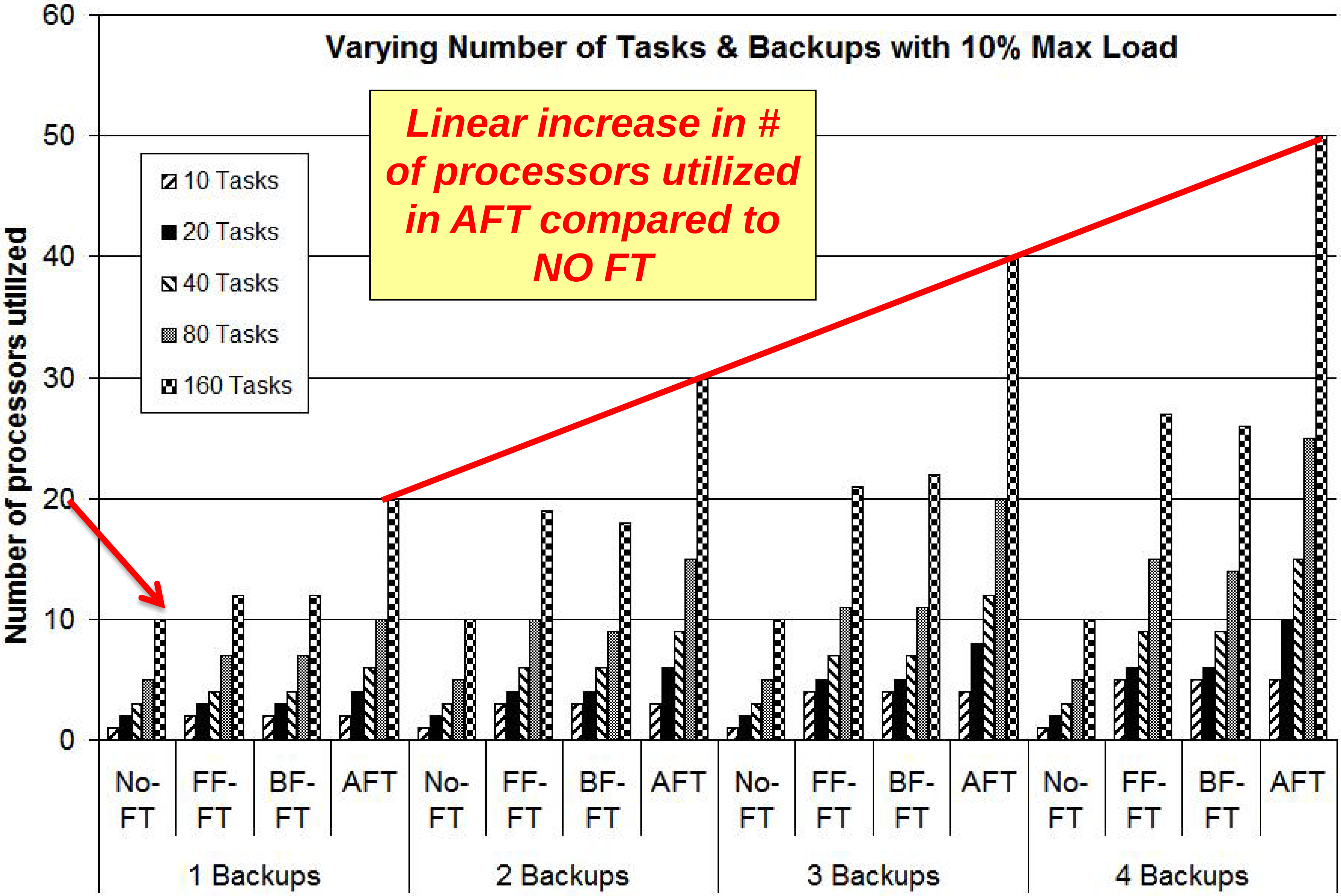


Comparison Schemes

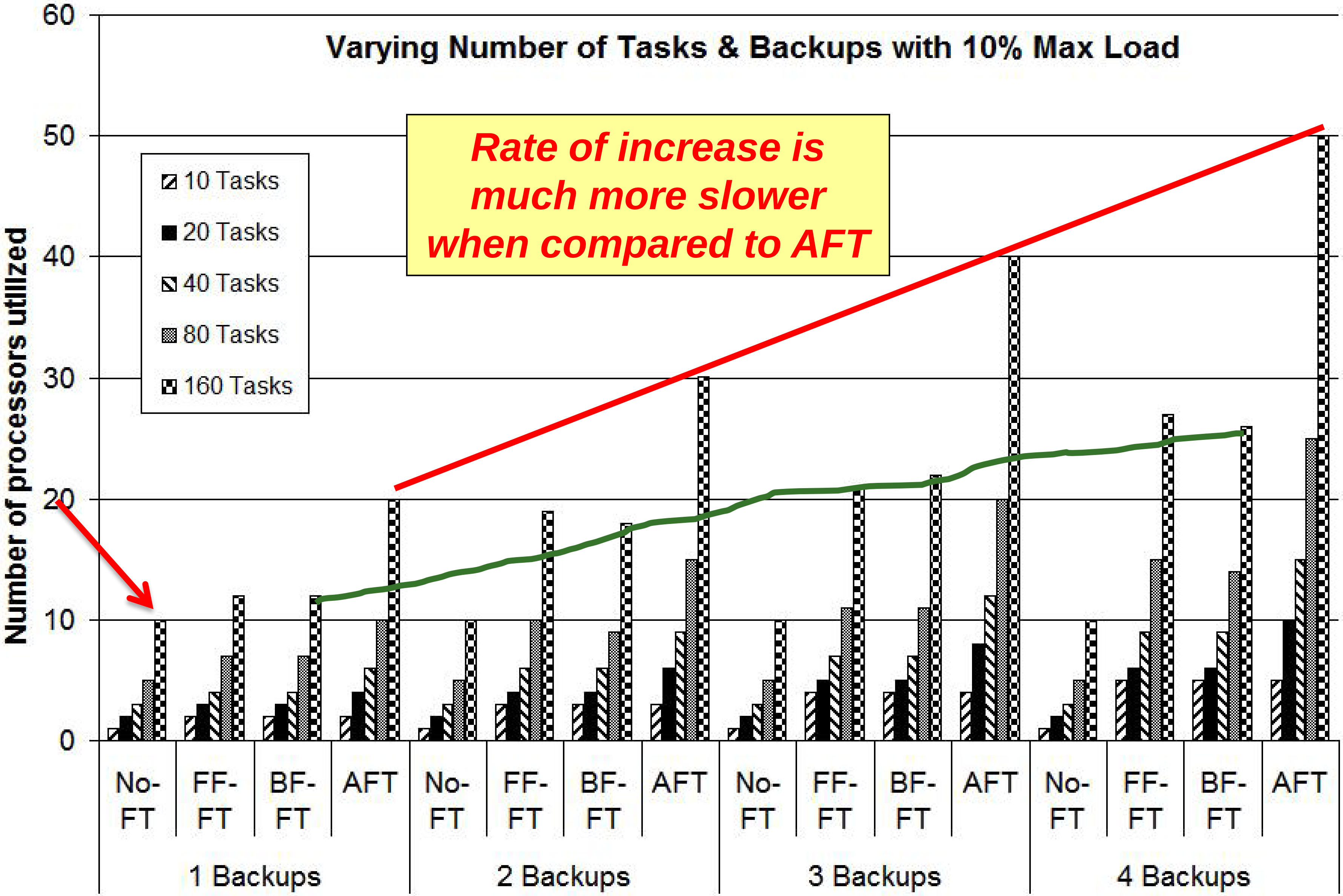
- Comparison schemes for evaluation
  - lower bound on number of processors utilized
    - Implementing the optimal allocation algorithm in [Dhall:78] - uses First Fit bin packing scheme
      - Optimal no fault-tolerance scenario (No FT)
  - Upper bound on # of processors
    - Multiplying # of processors utilized in the No FT case with # of replicas
      - Optimal active replication scenario (AFT)
  - DeCoRAM allocation heuristic
    - First Fit (FF-FT) & Best Fit (BF-FT) schemes
      - Optimal passive replication (FF-FT & BF-FT)



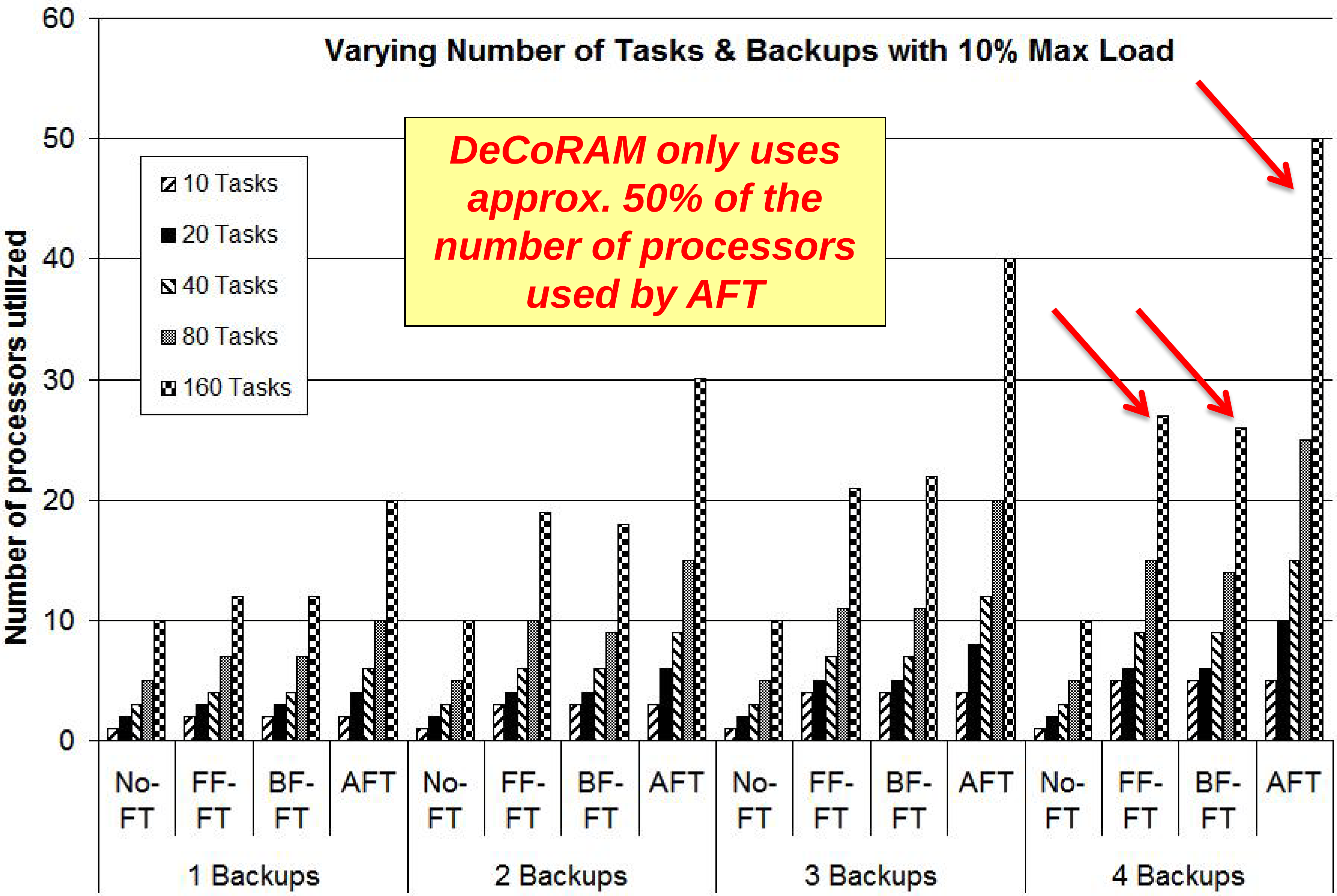
Experiment Results



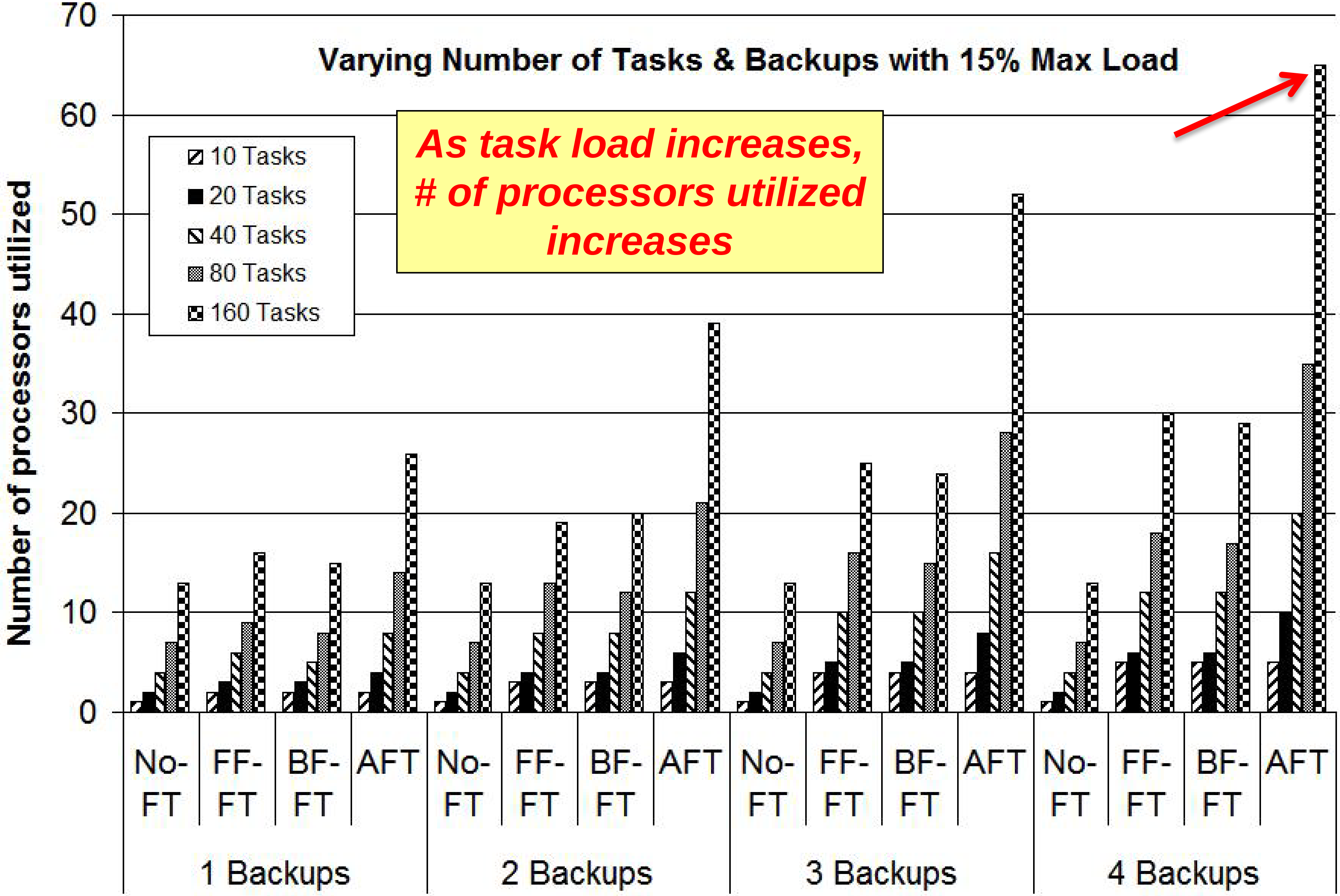
Experiment Results



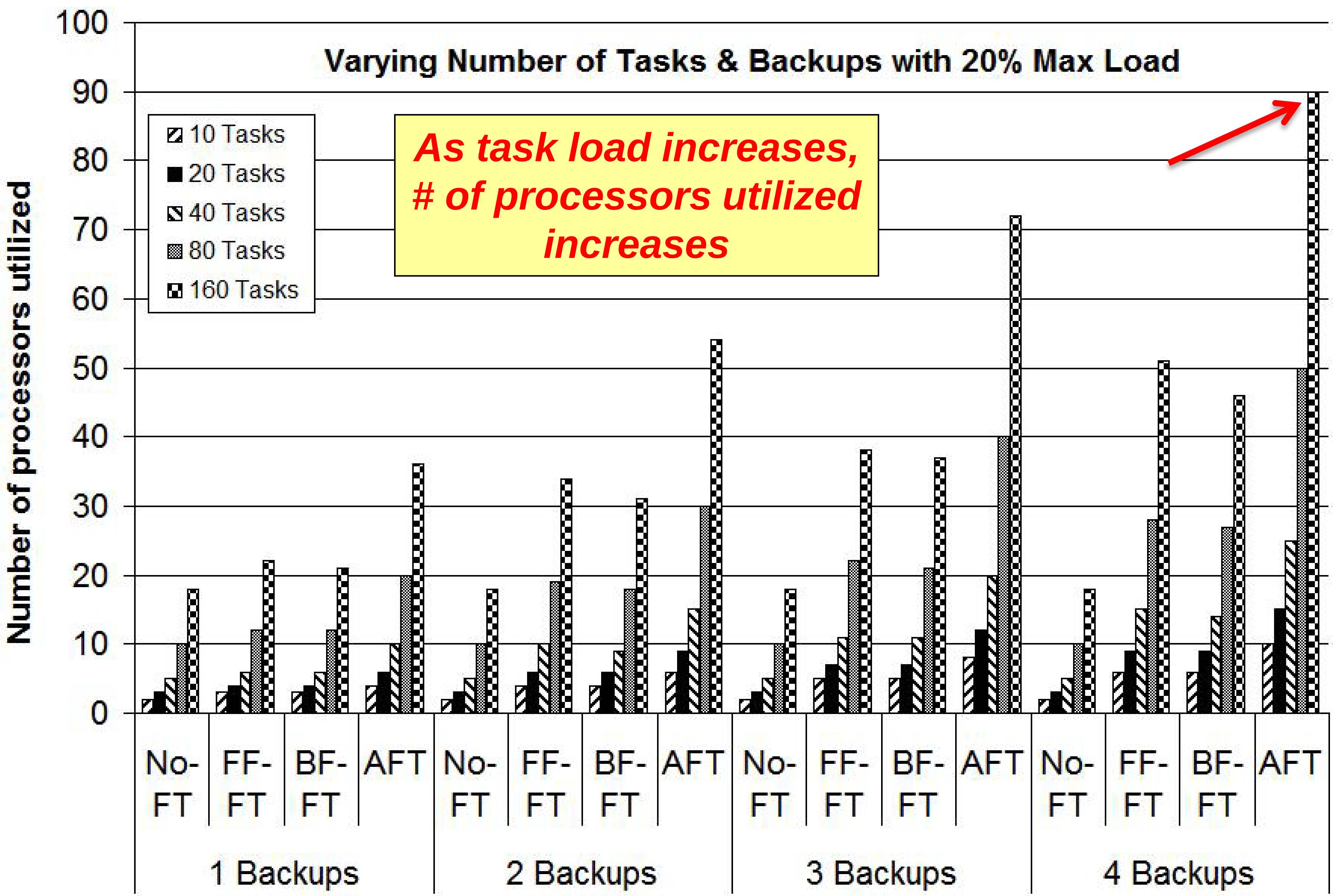
Experiment Results



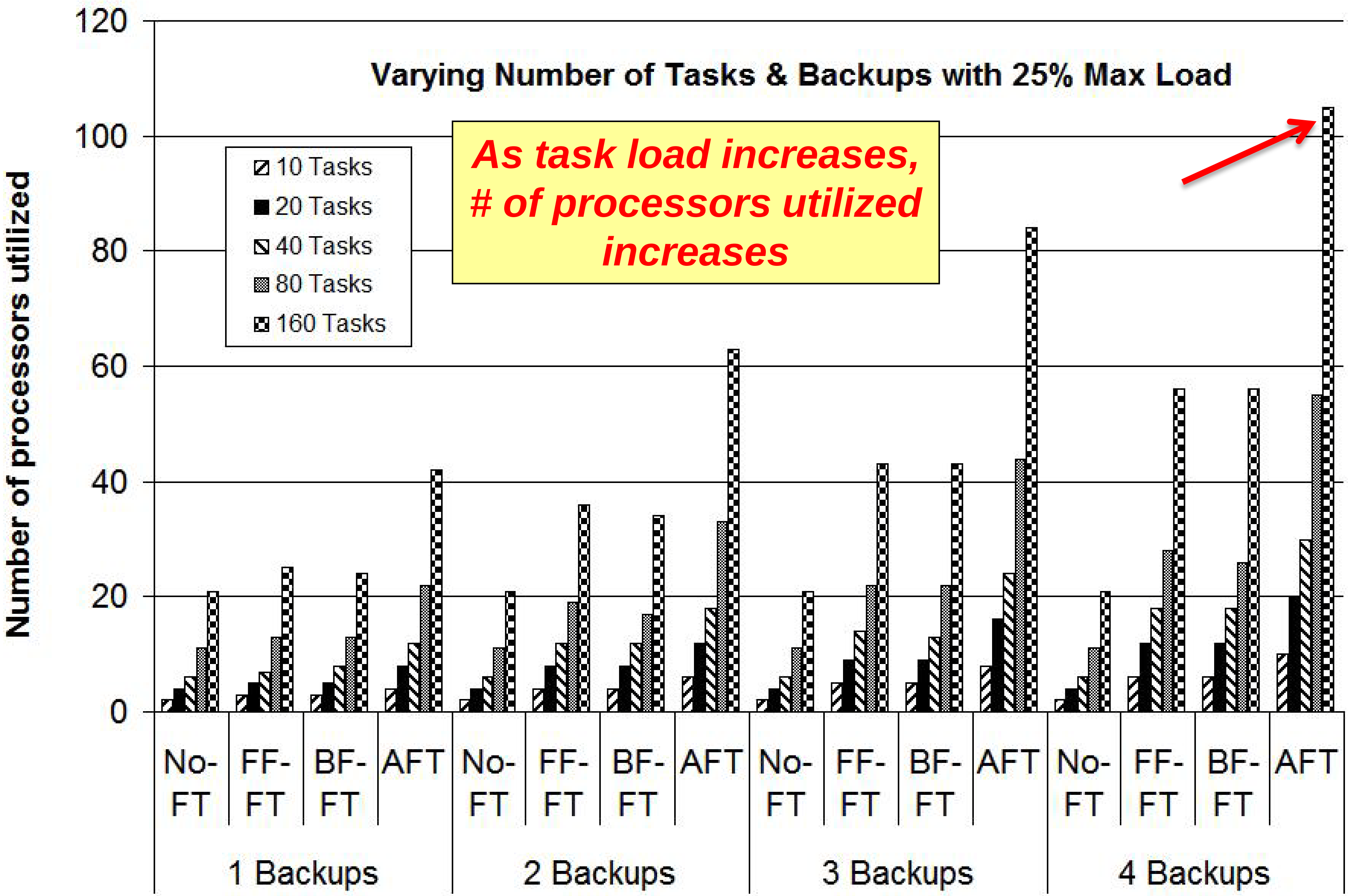
Experiment Results



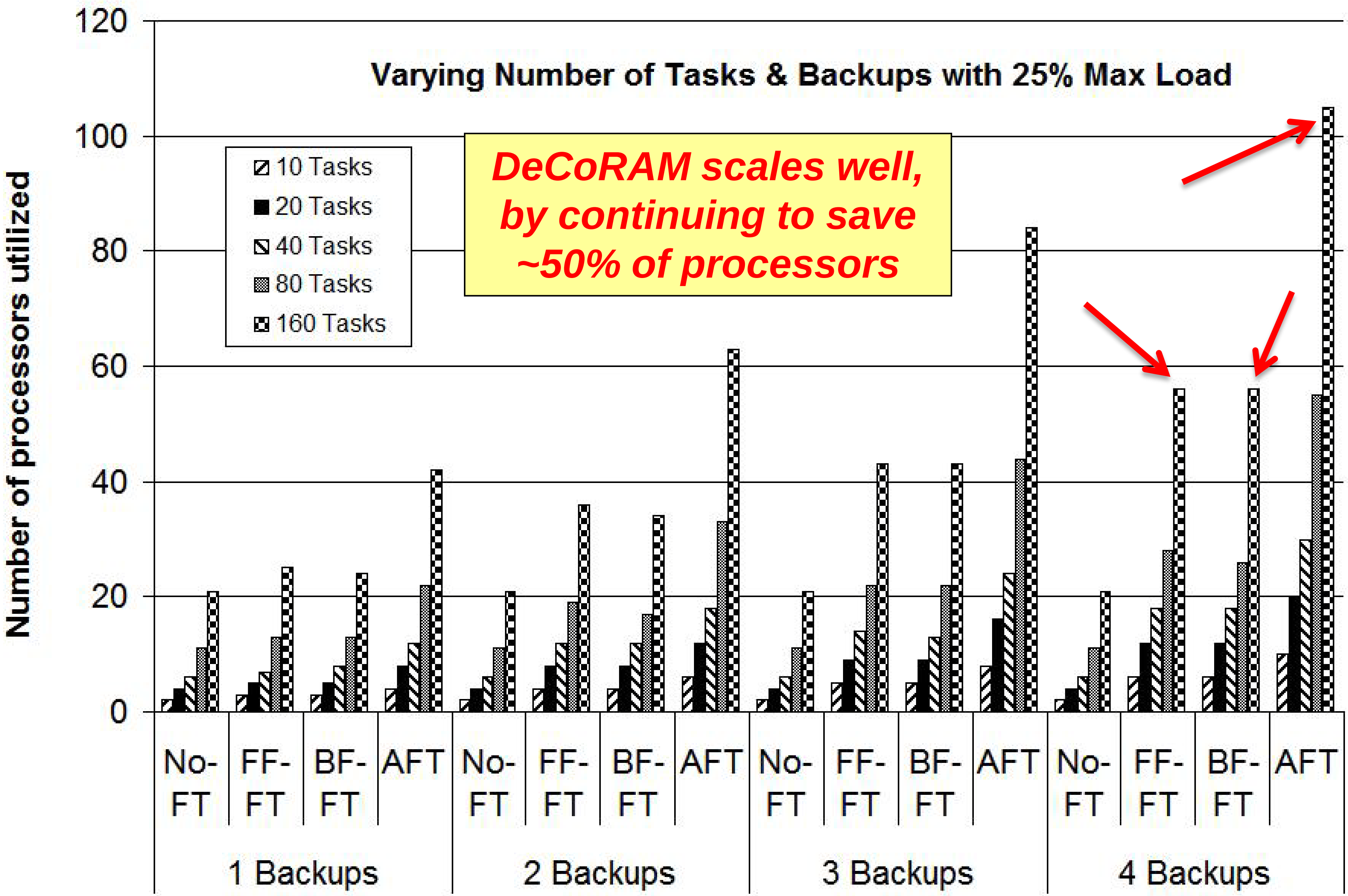
Experiment Results



Experiment Results

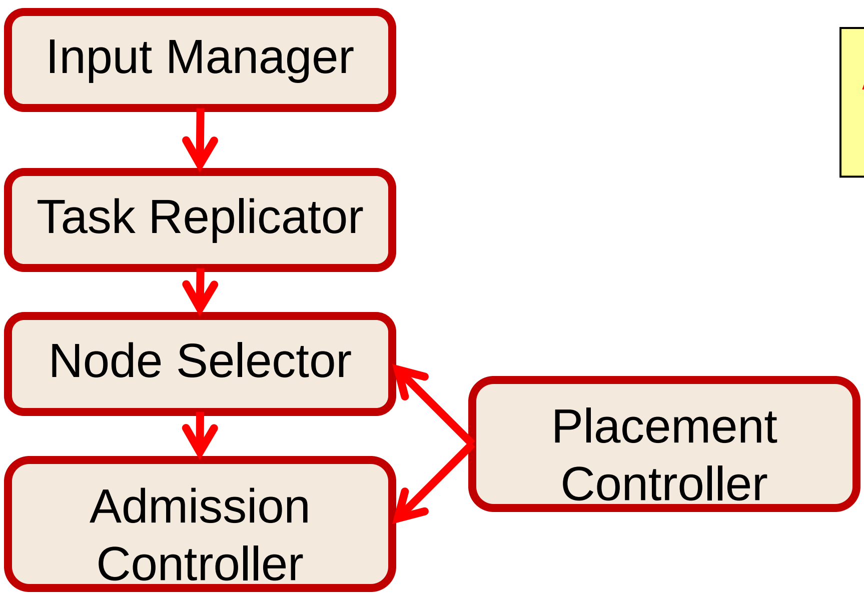


Experiment Results



DeCoRAM Pluggable Allocation Engine Architecture

- Design driven by separation of concerns
- Use of design patterns
- **Input Manager component** – collects per-task FT & RT requirements
- **Task Replicator component** – decides the order in which tasks are allocated
- **Node Selector component** – decides the node in which allocation will be checked
- **Admission Controller component** – applies DeCoRAM's novel algorithm
- **Placement Controller** component – calls the admission controller repeatedly to deploy all the applications & their replicas

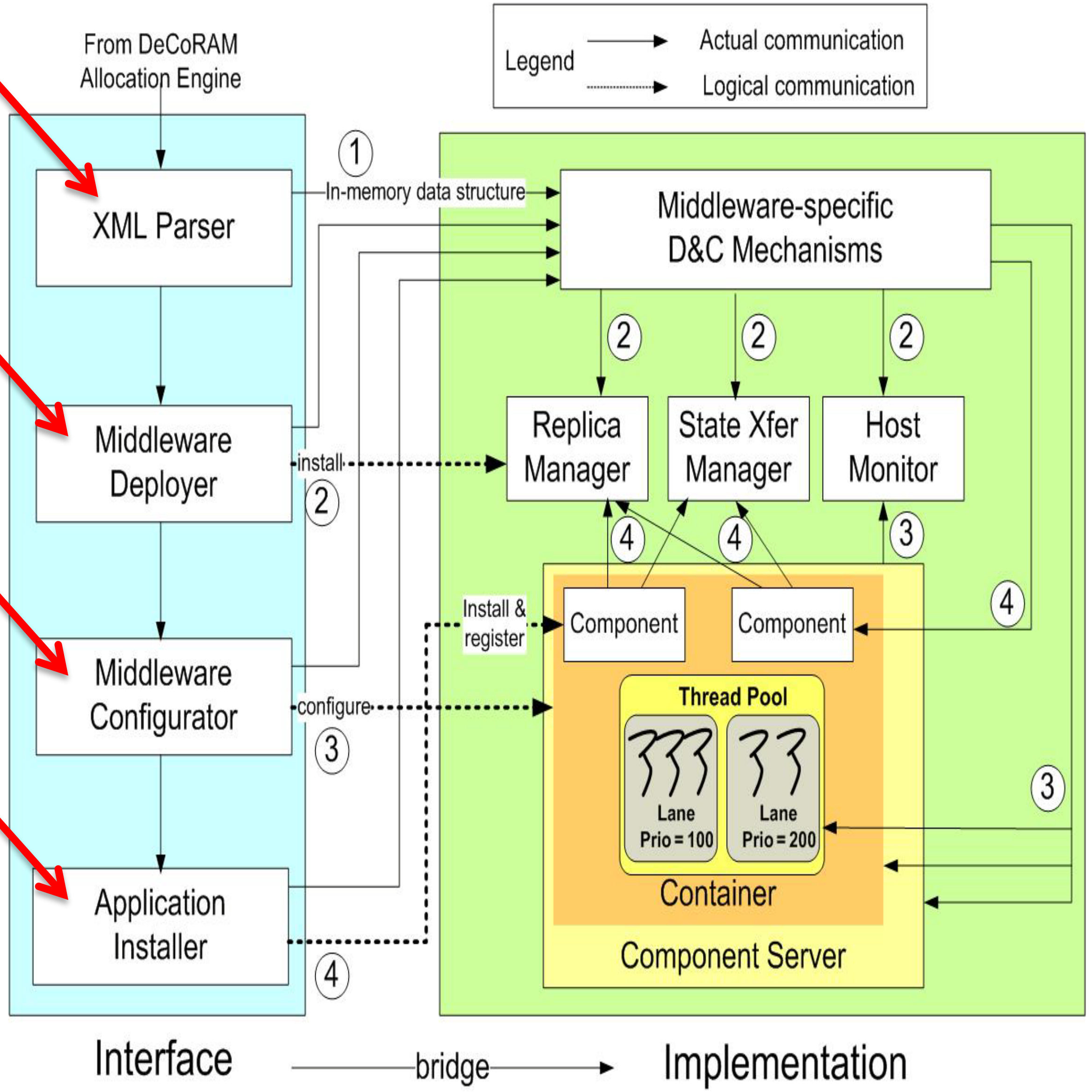


Allocation Engine implemented in ~7,000 lines of C++ code

Output decisions realized by DeCoRAM's D&C Engine

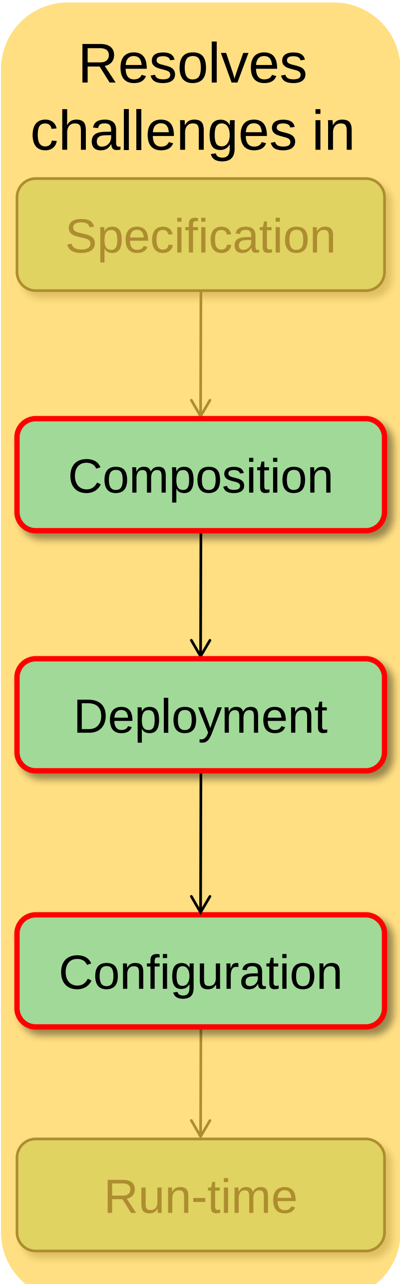
DeCoRAM Deployment & Configuration Engine

- Automated deployment & configuration support for fault-tolerant real-time systems
- **XML Parser**
  - uses middleware D&C mechanisms to decode allocation decisions
- **Middleware Deployer**
  - deploys FT middleware-specific entities
- **Middleware Configurator**
  - configures the underlying FT-RT middleware artifacts
- **Application Installer**
  - installs the application components & their replicas
- **Easily extendable**
  - Current implementation on top of CIAO, DAnCE, & FLARe middleware



DeCoRAM D&C Engine implemented in ~3,500 lines of C++ code

Post-Specification Phase: Generative Techniques to Support Missing Semantics



Focus on Generative Techniques for Introducing New Semantics into Middleware Implementations

- Generative Aspects for Fault-Tolerance (GRAFT)
  - Multi-stage model-driven development process
  - Weaving Dependability Concerns in System Artifacts
  - Provides model-to-model, model-to-text, model-to-code transformations

Related Research: Transparent FT Provisioning

Category	Related Research (Transparent FT Provisioning)
Model-driven	1. *Aspect-Oriented Programming Techniques to support Distribution, Fault Tolerance, & Load Balancing in the CORBA(LC) Component Model* by D. Sevilla, J. M. García, & A. Gómez 2. *CORRECT - Developing Fault-Tolerant Distributed Systems* by A. Capozucca, B. Gallina, N. Guelfi, P. Pelliccione, & A. Romanovsky 3. *Automatic Generation of Fault-Tolerant CORBA-Services* by A. Polze, J. Schwarz, & M. Malek 4. *Adding fault-tolerance to a hierarchical DRE system* by P. Rubel, J. Loyall, R. Schantz, & M. Gillen
Using AOP languages	1. *Implementing Fault Tolerance Using Aspect Oriented Programming* by R. Alexandersson & P. Öhman 2. *Aspects for improvement of performance in fault-tolerant software* by D. Szentiványi 3. *Aspect-Oriented Fault Tolerance for Real-Time Embedded Systems* by F. Afonso, C. Silva, N. Brito, S. Montenegro
Meta-Object Protocol (MOP)	1. *A Multi-Level Meta-Object Protocol for Fault-Tolerance in Complex Architectures* by F. Taiani & J.-C. Fabre 2. *Reflective fault-tolerant systems: From experience to challenges* by J. C. Ruiz, M.-O. Killijian, J.-C. Fabre, & P. Thévenod-Fosse

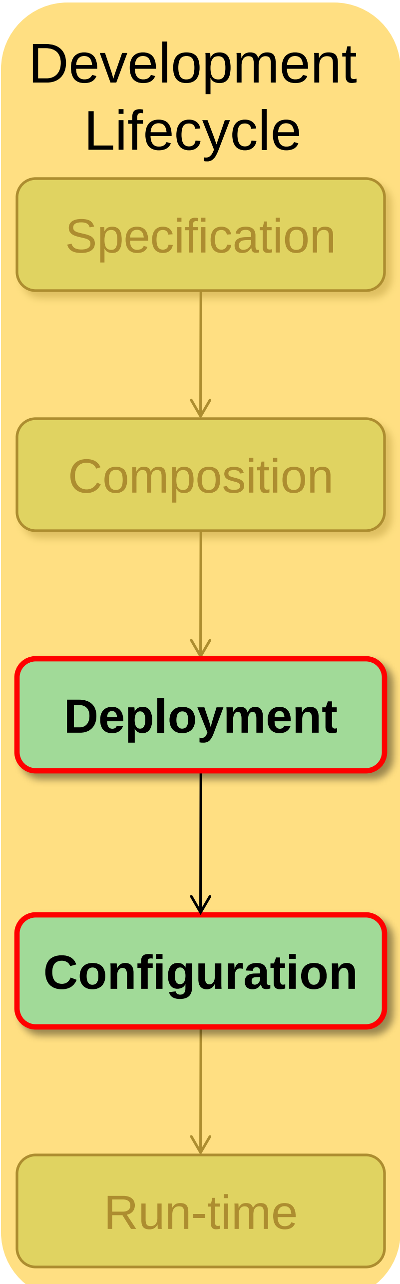
Related Research: Transparent FT Provisioning

Category	Related Research (Transparent FT Provisioning)
Model-driven	1. *Aspect-Oriented Programming Techniques to support Distribution, Fault Tolerance, & Load Balancing in the CORBA(LC) Component Model* by D. Sevilla, J. M. García, & A. Gómez 2. *CORRECT - Developing Fault-Tolerant Distributed Systems* by A. Capozucca, B. Gallina, N. Guelfi, P. Pelliccione, & A. Romanovsky 3. *Automatic Generation of Fault-Tolerant CORBA-Services* by A. Polze, J. Schwarz, & M. Malek 4. *Adding fault-tolerance to a hierarchical DRE system* by P. Rubel, J. Loyall, R. Schantz, & M. Gillen
Using AOP languages	1. *Implementing Fault Tolerance Using Aspect Oriented Programming* by R. Alexandersson & P. Öhman 2. *Aspects for improvement of performance in fault-tolerant software* by D. Szentiványi 3. *Aspect-Oriented Fault Tolerance for Real-Time Embedded Systems* by F. Afonso, C. Silva, N. Brito, S. Montenegro
Meta-Object Protocol (MOP)	1. *A Multi-Level Meta-Object Protocol for Fault-Tolerance in Complex Architectures* by F. Taiani & J.-C. Fabre 2. *Reflective fault-tolerant systems: From experience to challenges* by J. C. Ruiz, M.-O. Killijian, J.-C. Fabre, & P. Thévenod-Fosse

M2M transformation & code generation

Performance improvement for FT using AOP

What is Missing? Transparent FT Provisioning

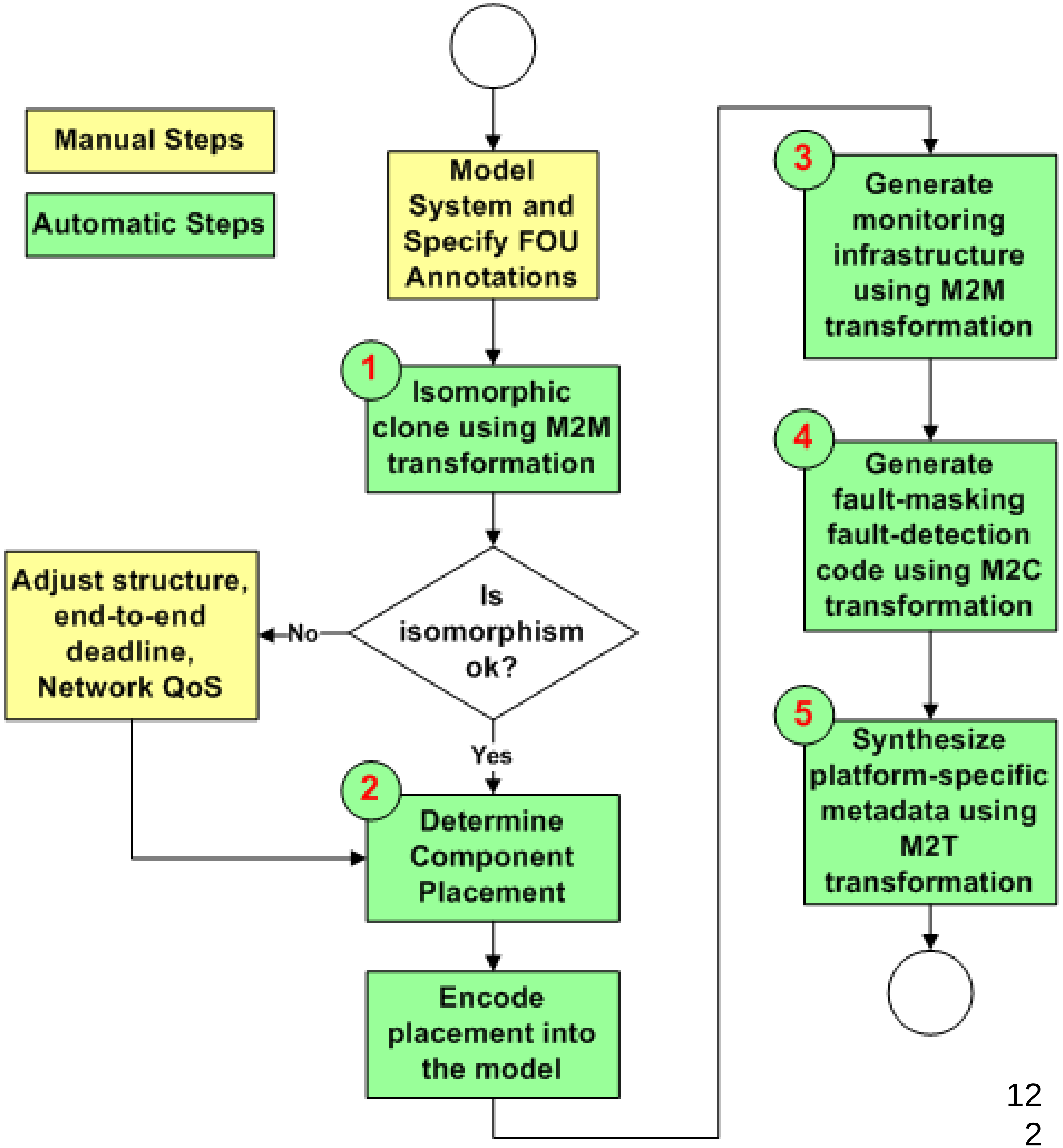


- Not all the necessary steps are supported coherently
  1. Automatic component instrumentation for fault-handling code
  2. Deciding placement of components & their replicas
  3. Deploying primaries, replicas, & monitoring infrastructure
  4. Platform-specific metadata synthesis (XML)
- Missing domain-specific recovery semantics (run-time middleware)
  - Group failover is DRE-specific & often neglected
  - Costly to modify the middleware
  - Application-level solutions lose transparency & reusability
- Missing transparent network QoS provisioning (D&C middleware)
  - Configuration of network resources (edge routers)
  - Configuration of containers for correct packet marking

1. How to add **domain-specific recovery semantics** in COTS middleware **retroactively?**
2. How to **automate** it to improve productivity & reduce cost?

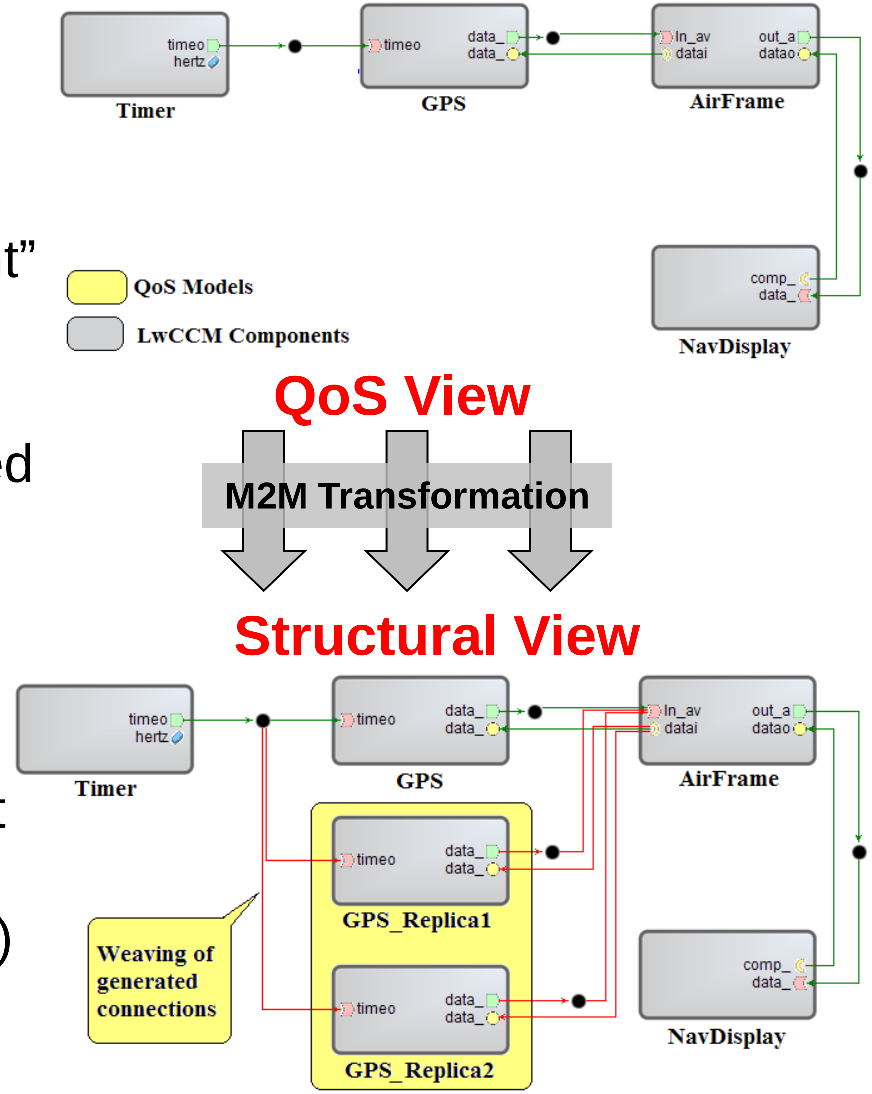
Soln: Generative Aspects for Fault Tolerance (GRAFT)

- Multi-stage model-driven generative process
- Incremental model-refinement using transformations
    - Model-to-model
    - Model-to-text
    - Model-to-code
- Weaves dependability concerns in system artifacts



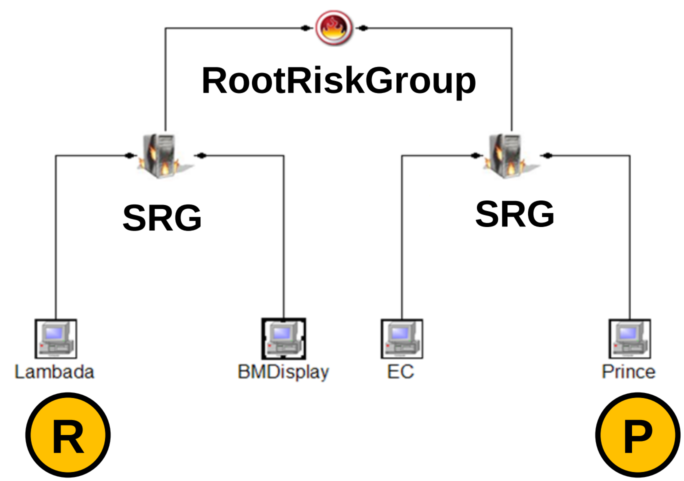
Stage 1: Isomorphic M2M Transformation

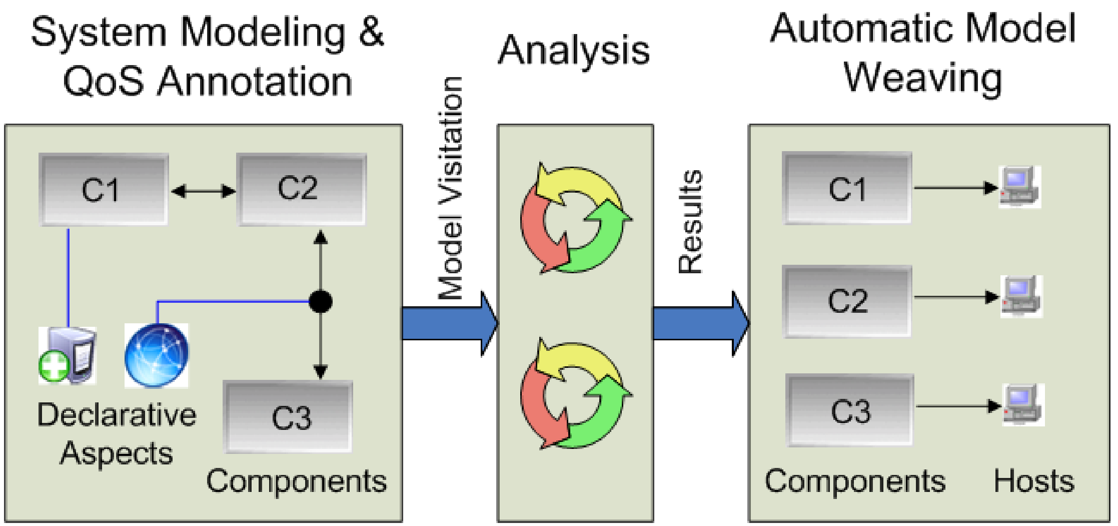
- Step1: Model structural composition of operational string
- Step2: Annotate components with failover unit(s) marking them “fault-tolerant” in the QoS view
- Step3: Use aspect-oriented M2M transformation developed using Embedded Constraint Language (ECL) of C-SAW
- Step4: Component replicas & interconnections are generated automatically
- Step 5: FOU annotations are removed but other QoS annotations are cloned (uses Dependency Inversion Principle of CQML)
- Step 6: Isomorphic clone can be modified manually (reliability through diversity)



Stage 2: Determine Component Placement

- Strategic placement of components, e.g. using DeCoRAM
  - Improves availability of the system
  - Several constraint satisfaction algorithms exist
- Placement comparison heuristic
  - Hop-count between replicas
  - Formulation based on the co-failure probabilities captured using Shared Risk Group (SRG)
    - E.g., shared power supply, A/C, fire zone
  - Reduces simultaneous failure probability
- GRAFT transformations weave the decisions back into the model





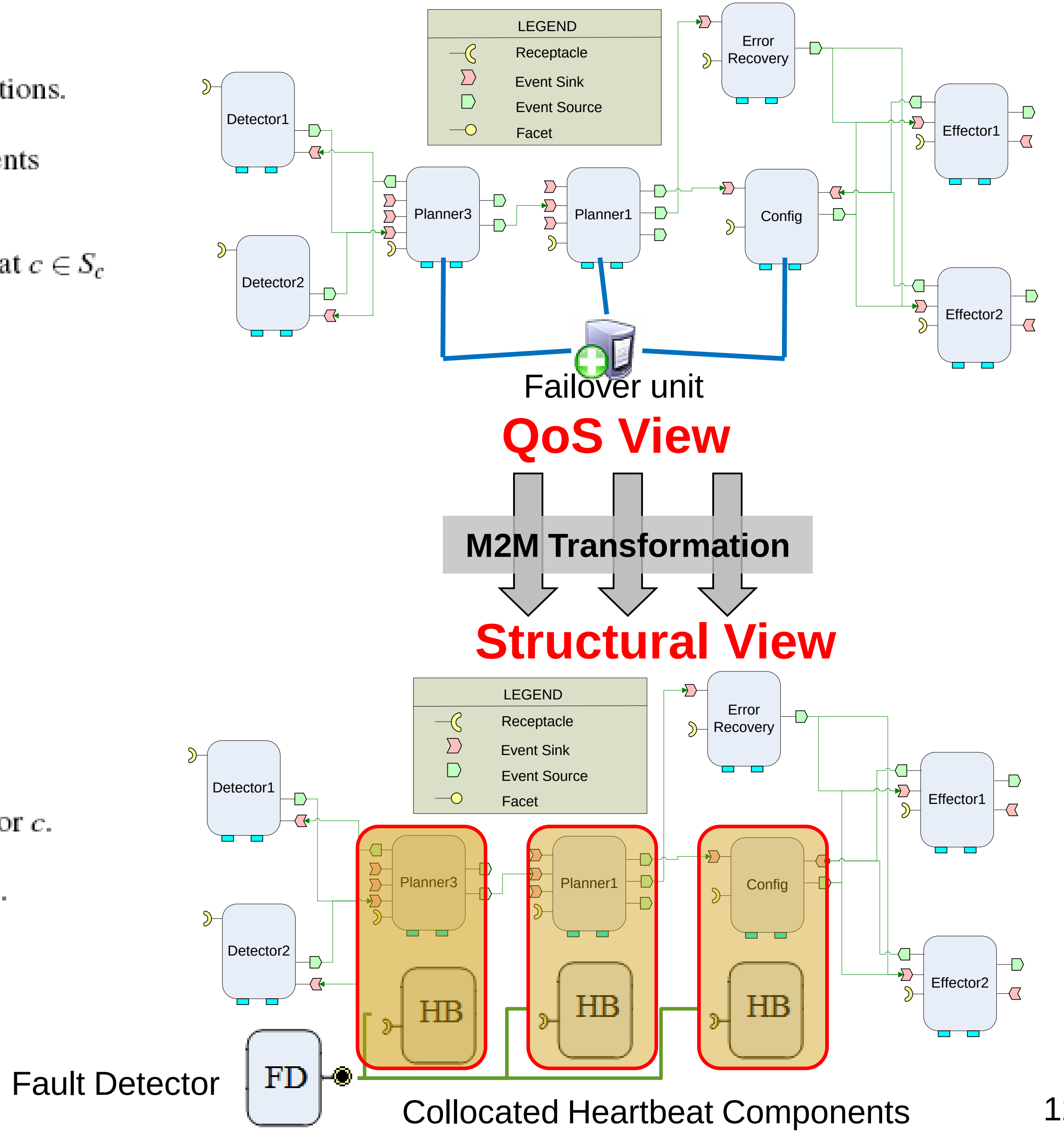
Stage 3: Synthesizing Fault Monitoring Infrastructure

Transformation Algorithm

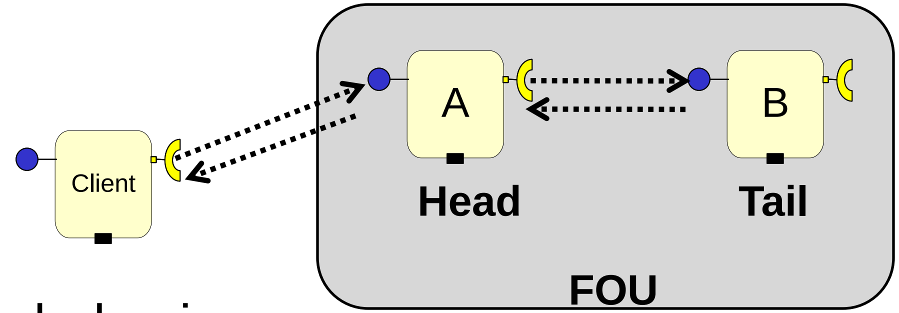
M : Systems's structural model with annotations.
D : Deployment model of the system
M_e : Extended M with monitoring components
D_e : Deployment model of M_e
c : A business component
S_c : A set of collocated components such that $c \in S_c$
HB_c : Heartbeat component monitoring c
F : Fault Detector component.

Input: M, D
Output: M_e, D_e (Initially empty)

begin
$M_e := M$
$D_e := D$
$S_F := \emptyset$
F := New fault detector component
$M_e := M_e \cup F$
$S_F := S_F \cup F$
$D_e := D_e \cup S_F$
for each component c in M **do**
  if a FailOverUnit is associated with c
    let HB_c := New heartbeat component for c.
    $M_e := M_e \cup HB_c$
    let i := New connection from F to HB_c.
    $M_e := M_e \cup i$
    let $c \in S_c$ and $S_c \in D$
    $S_c := S_c \cup HB_c$
    $D_e := D_e \cup S_c$
  endif
end for
end



Stage 4: Synthesizing Code for Group Failover (1/2)



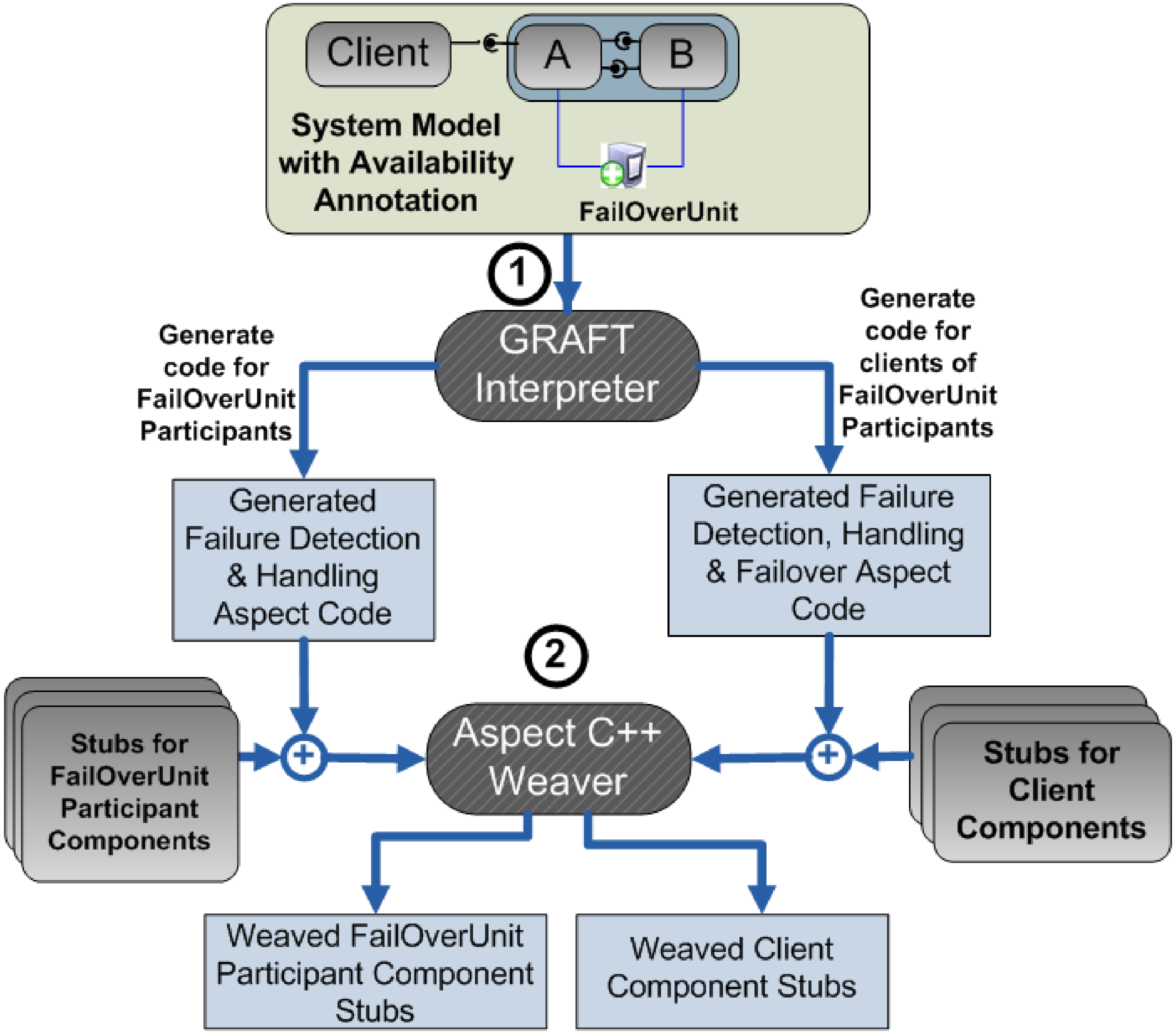
- Code generation for fault handling
  - Reliable fault detection
  - Transparent fault masking
  - Fast client failover
- Location of failure determines handling behavior

Head component failure	Tail component failure
Client-side code detects the failure	Only other FOU participants detect the failure. Client waits.
---	Trigger client-side exception by forcing FOU to shutdown
Client-side code does transparent failover	Client-side code detects passivation of the head component & does transparent failover

- FOU shutdown is achieved using seamless integration with D&C middleware APIs
  - e.g., Domain Application Manager (DAM) of CCM
- Shutdown method calls are generated in fault-handling code

Stage 4: Synthesizing Code for Group Failover (2/2)

- Two behaviors based on component position
- FOU participant's behavior
  - Detects the failure
  - Shuts down the FOU including itself
- FOU client's behavior
  - Detects the failure
  - Does an automatic failover to a replica FOU
  - Optionally shuts down the FOU to save resources
- Generated code: AspectC++



- AspectC++ compiler weaves in the generated code in the respective component stubs

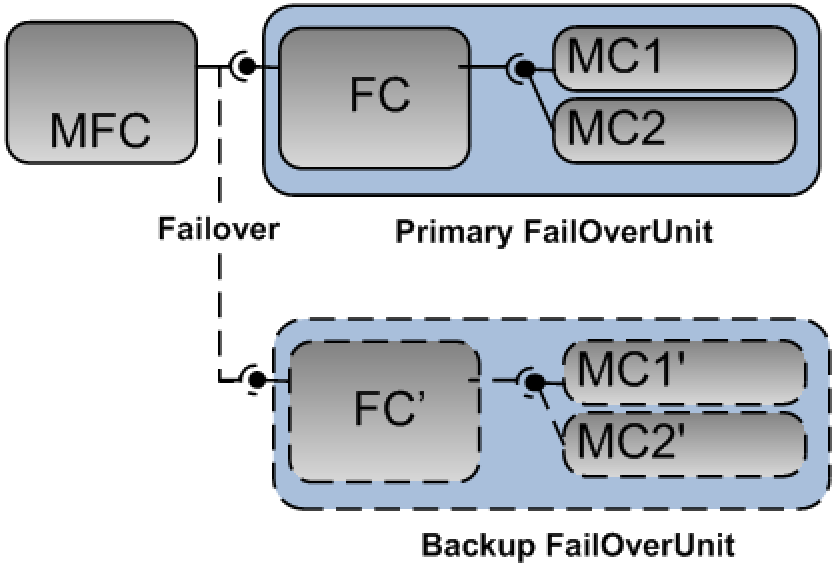
Stage 5: Synthesizing Platform-specific Metadata

- Component Technologies use XML metadata to configure middleware
- Existing model interpreters can be reused without any modifications
  - CQML's FT modeling is opaque to existing model interpreters
  - GRAFT model transformations are transparent to the model interpreters

GRAFT synthesizes the necessary artifacts for transparent FT provisioning for DRE operational strings

Evaluating Modeling Efforts Reduction Using GRAFT

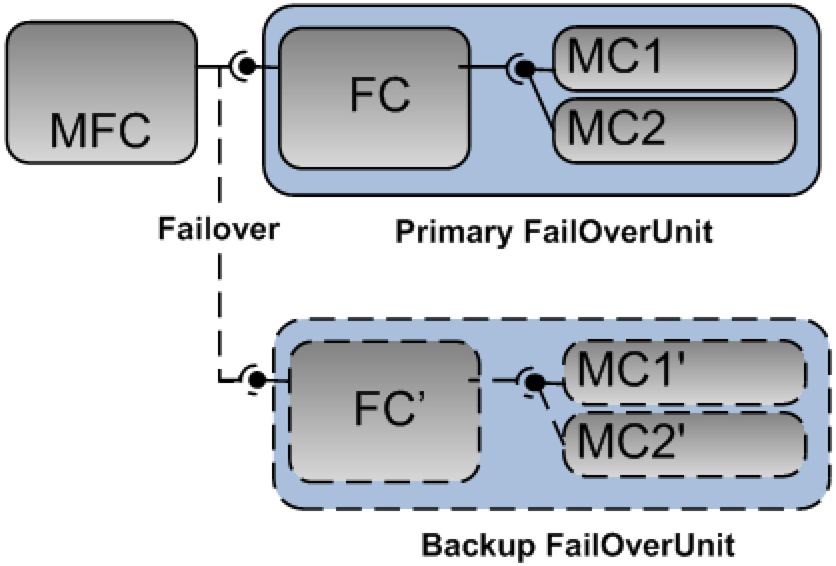
- Case-study - Warehouse Inventory Tracking System
- GRAFT's isomorphic M2M transformation eliminates human modeling efforts of replicas
  - Components
  - Connections
  - QoS requirements



	Fault-tolerance Modeling Efforts		
Component Name	# of original connections	# of replica components	# of replica connections
Material Flow Control	1 / 1	0 / 0	2 / 0
Flipper Controller	2 / 2	2 / 0	4 / 0
Motor Controller 1	1 / 1	2 / 0	2 / 0
Motor Controller 2	2 / 1	2 / 0	2 / 0

Evaluating Programming Efforts Reduction Using GRAFT

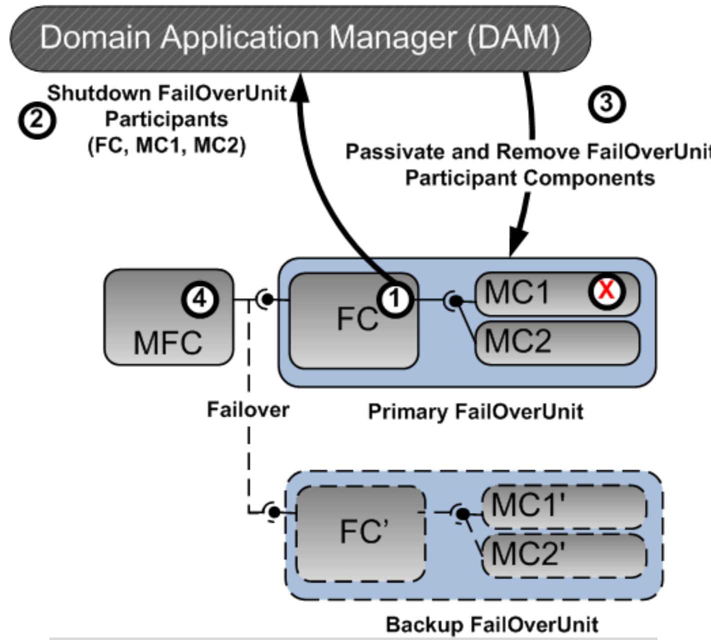
- GRAFT's code generator reduces human programming efforts
- Code for fault-detection, fault-masking, & failover
  - # of try blocks
  - # of catch blocks
  - Total # of lines

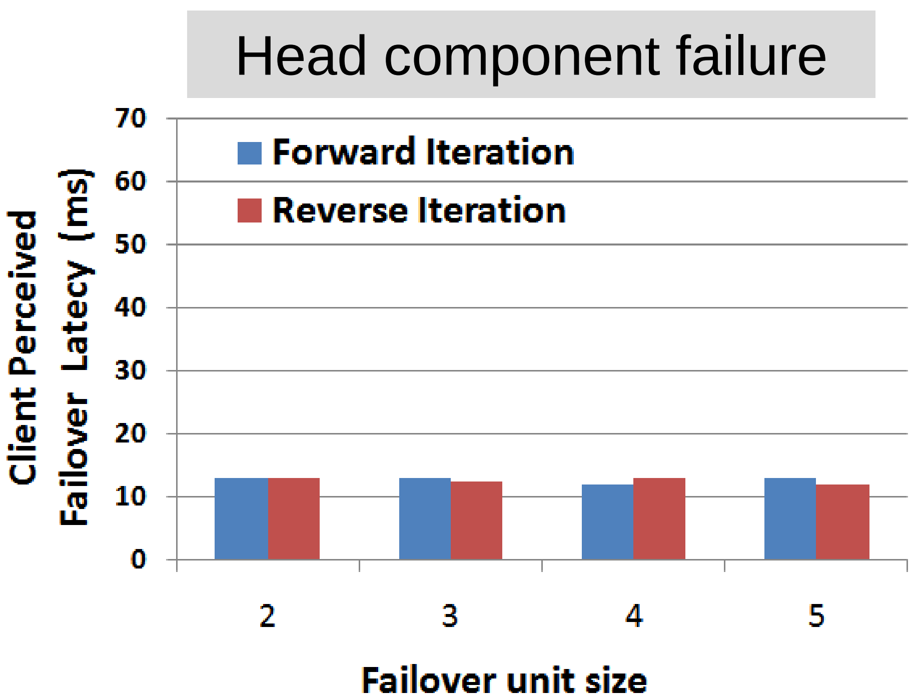


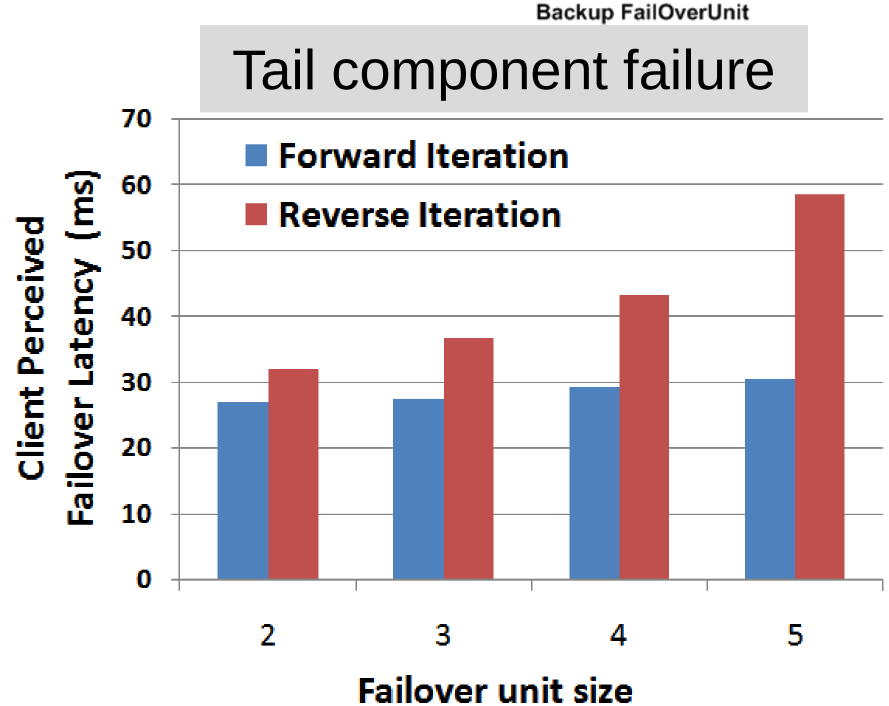
	Fault-tolerance Programming Efforts		
Component Name	# of try blocks	# of catch blocks	Total # of lines
Material Flow Control	1 / 0	3 / 0	45 / 0
Flipper Controller	2 / 0	6 / 0	90 / 0
Motor Controller 1	0 / 0	0 / 0	0 / 0
Motor Controller 2	0 / 0	0 / 0	0 / 0

Evaluating Client Perceived Failover Latency Using GRAFT

- Client perceived failover latency
  - Sensitive to the location of failure
  - Sensitive to the implementation of DAM
- Head component failure
  - Constant failover latency
- Tail component failover
  - Linear increase in failover latency







Presentation Road Map

- Technology Context: DRE Systems
- DRE System Lifecycle & FT-RT Challenges
- Design-time Solutions
- Deployment & Configuration-time Solutions
- **Runtime Solutions**
- Ongoing Work
- Concluding Remarks

Runtime Phase: Real-time Fault Detection & Recovery

Development Lifecycle

Specification

Composition

Deployment

Configuration

Run-time

- Fault Tolerant Lightweight Adaptive Middleware (FLARe)
  - Two algorithms (LAAF and ROME)

Related Research

Category	Related Research
CORBA-based Fault-tolerant Middleware Systems	P. Felber et. al., *Experiences, Approaches, & Challenges in Building Fault-tolerant CORBA Systems*, in IEEE Transactions on Computers, May 2004 T. Bennani et. al., *Implementing Simple Replication Protocols Using CORBA Portable Interceptors & Java S*… International Conference on D… 4), Italy, 2004 P. Narasimhan et. al., *MEAD:* … 4, in Concurrency & Computation: Practice & Experience, 2005
Adaptive Passive Replication Systems	S. Pertet et. al., *Proactive R*… , in Proceedings of the IEEE Int… ems & Networks (DSN 2004), Italy, 2004 P. Katsaros et. al., *Optimal Object State Transfer – Recovery Policies for Fault-tolerant Distributed Systems*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN 2004), Italy, 2004 Z. Cai et. al., *Utility-driven Proactive Management of Availability in Enterprise-scale Information Flows*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2006), Melbourne, Australia, November 2006 L. Froihofer et. al., *Middleware Support for Adaptive Dependability*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2007), Newport Beach, CA, November 2007

Runtime adaptations to reduce failure recovery times

Middleware building blocks for fault-tolerant systems

Related Research

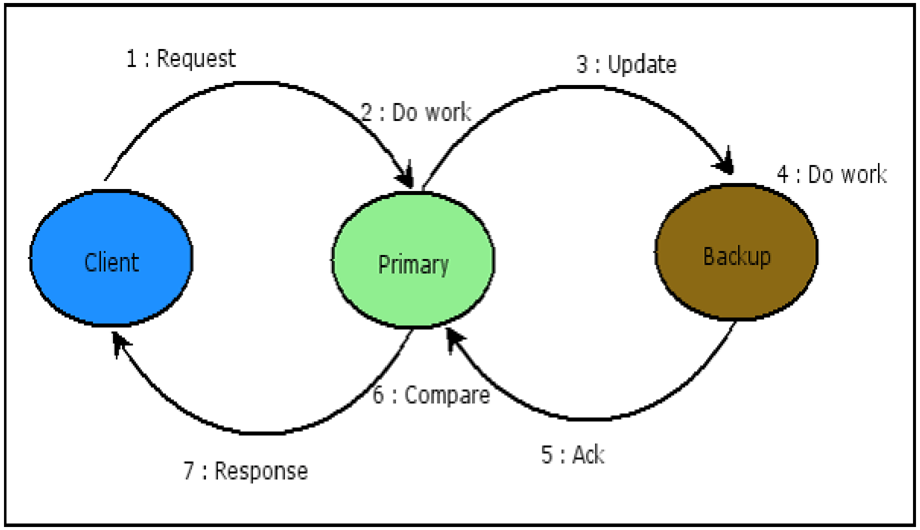
Category	Related Research
Load-Aware Adaptations of Fault-tolerance Configurations	T. Dumitras et. al., *Fault-tolerant Middleware & the Magical 1%*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2005), Grenoble, France, November 2005 O. Marin et. al., *DARX: A … ent Software*, In Proceeding… are Reliability Engineering (I… S. Krishnamurthy et. al., … *e for Replicated Services*, in IEEE Transactions on Parallel … Systems (IEEE TPDS), 2003
Real-time Fault-tolerant Systems	D. Powell et. al., *Distributed Fault … Lessons from Delta-4*, In IEEE MICRO, 1994 K. H. Kim et. al., *The PSTR… ia Active Object Replication &* … on Knowledge & Data Enginee… S. Krishnamurthy et. al., *Dy… ing Timing Faults*, in Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN 2001), 2001 H. Zou et. al., *A Real-time Primary Backup Replication Service*, in IEEE Transactions on Parallel & Distributed Systems (IEEE TPDS), 1999

Schedulability analysis to schedule backups in case primary replica fails, faster processing times

Load-aware adaptations – change of replication styles, reduced degree of active replication

Related Research: What is Missing?

- Existing passive replication solutions do not deal with overloads
  - workload fluctuations & multiple failures could lead to overloads
  - response times affected – if overloads not handled

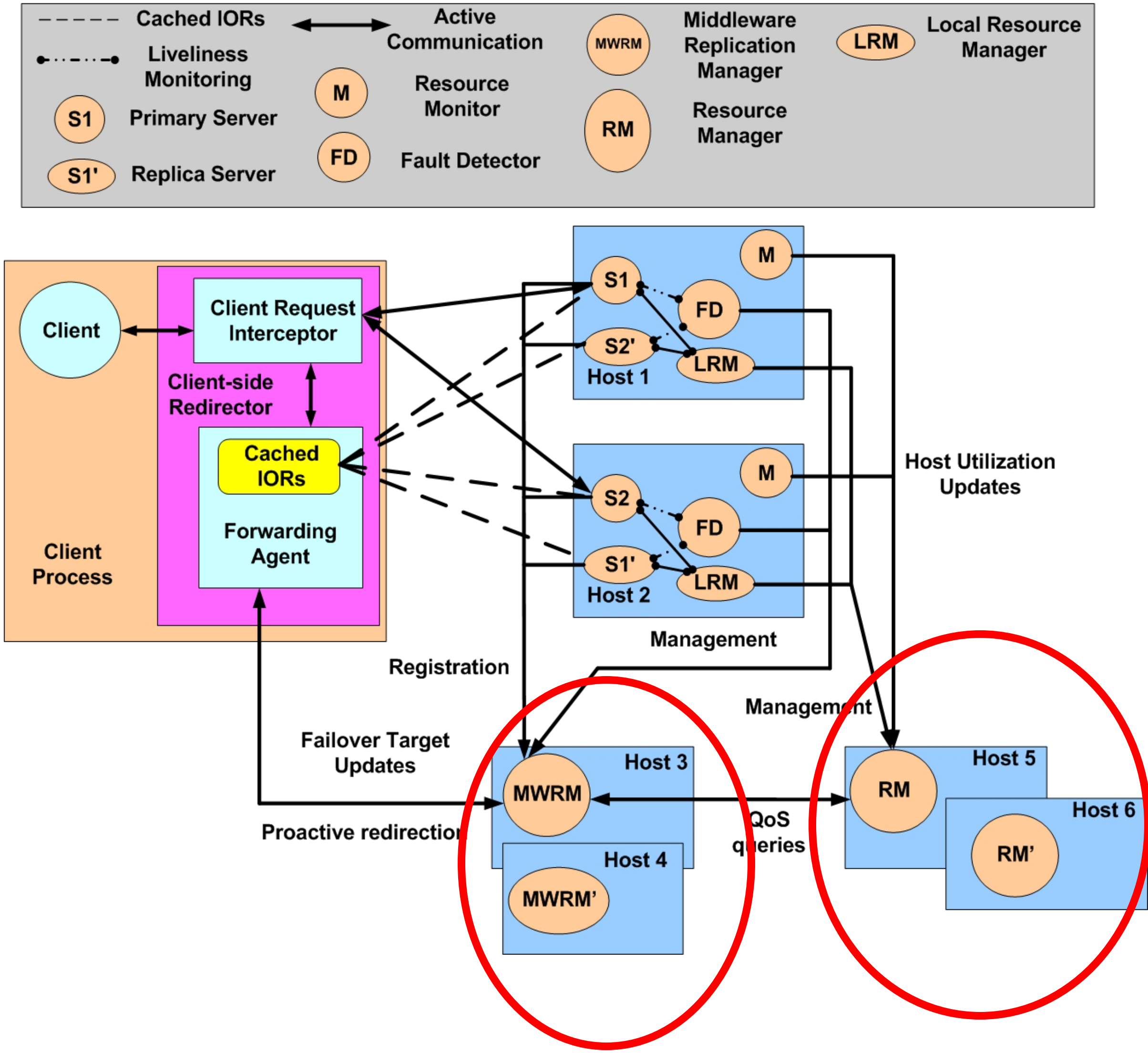


- Existing passive replication systems do not deal with resource-aware failovers
  - If clients are redirected to heavily loaded replicas upon failure, their response time requirements will not be satisfied
  - failover strategies are most often static, which means that clients get a failover behavior that is optimal at deployment-time & not at runtime

Solution Approach: FLARe : Fault-tolerant Middleware with adaptive failover target selection & overload management support

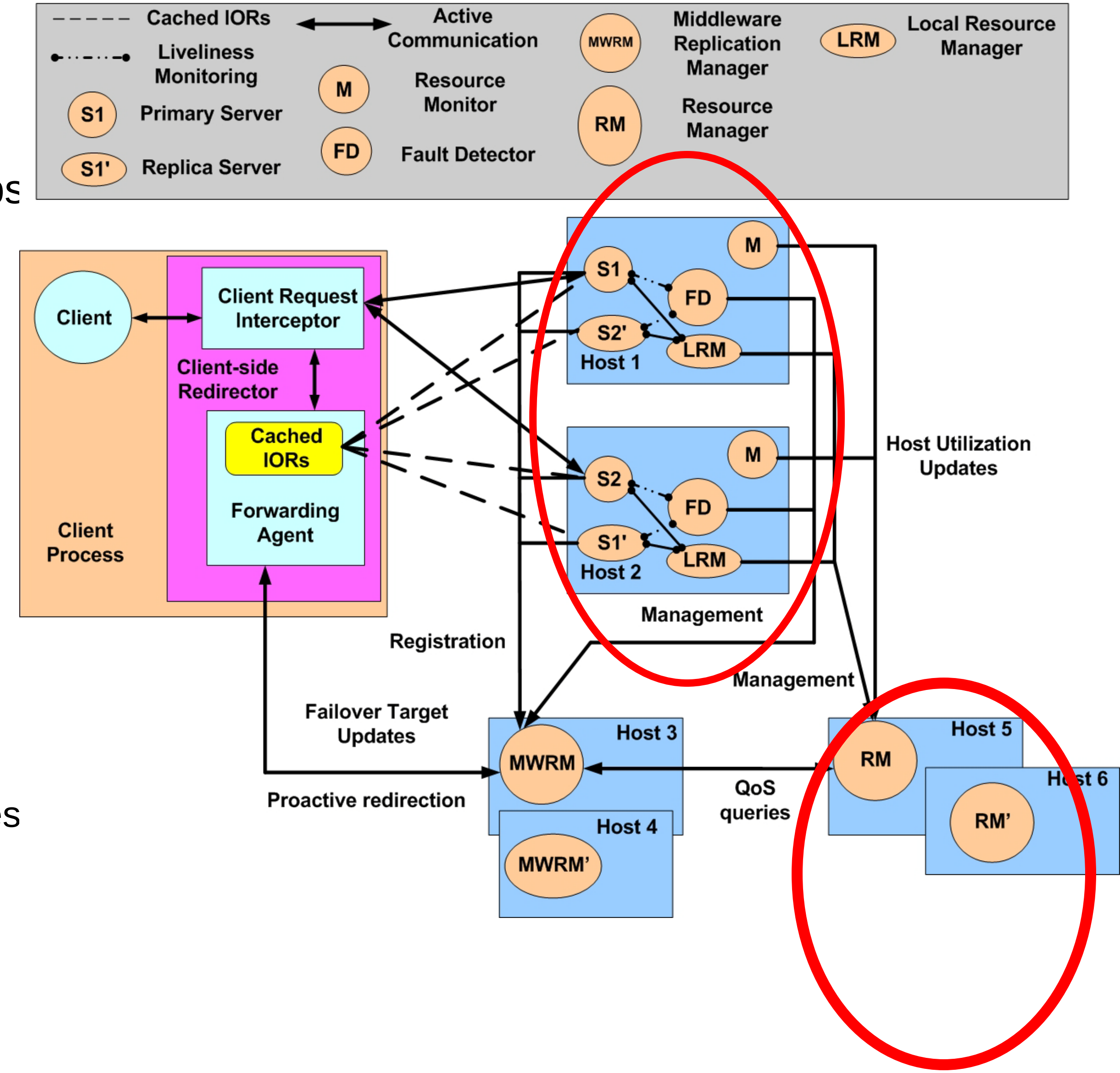
Our Approach: FLARe RT-FT Middleware

- **FLARe** = Fault-tolerant Lightweight Adaptive Real-time Middleware
  - RT-CORBA based lightweight FT
- **Resource-aware FT**
  - Resource manager – pluggable resource management algorithms
  - FT decisions made in conjunction with middleware replication manager
    - manages primary & backup replicas
    - provides registration interfaces
    - handles failure detection
    - starts new replicas



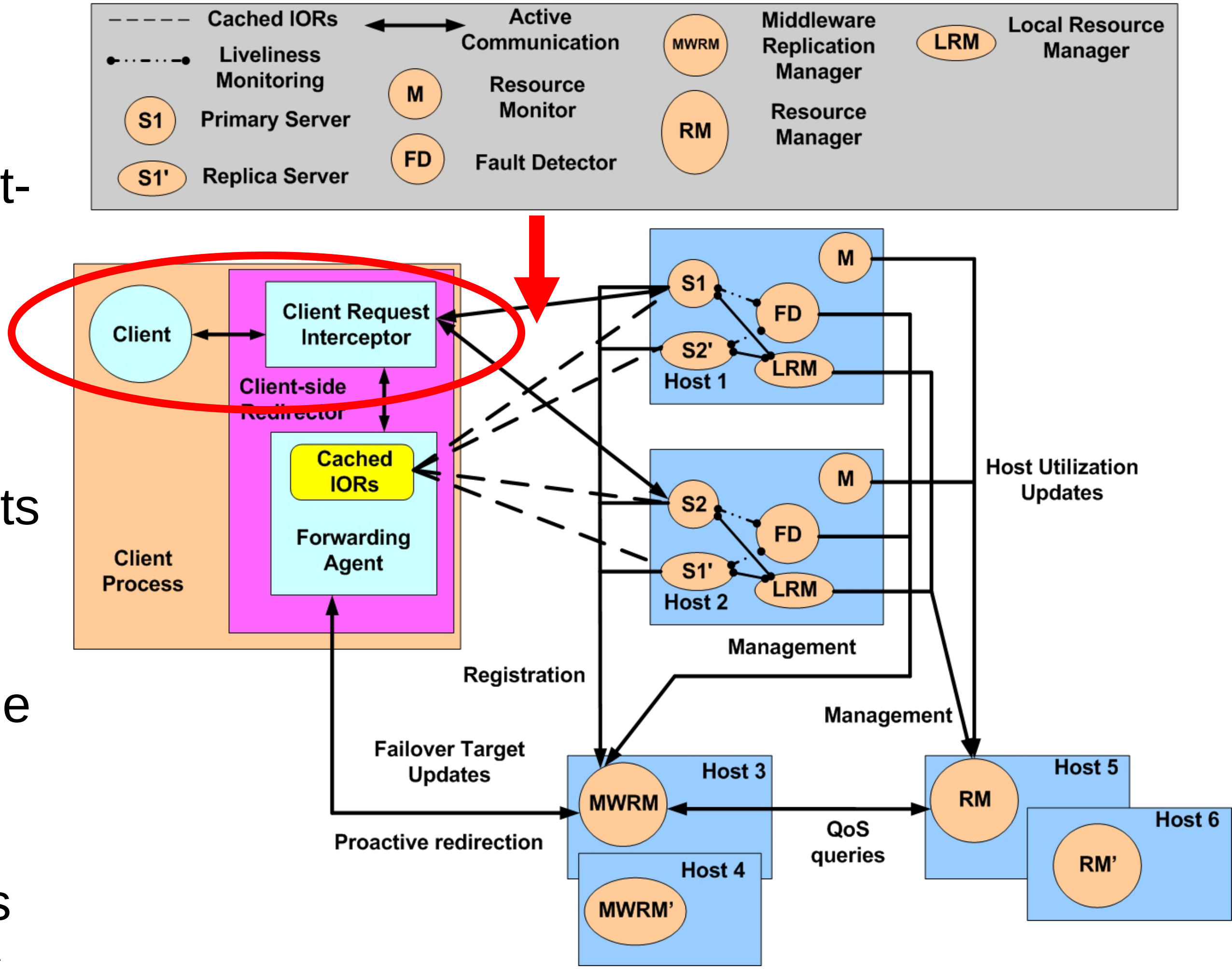
Our Approach: FLARe RT-FT Middleware

- **Real-time performance during failures & overloads**
  - monitor CPU utilizations at hosts where primary & backups are deployed
  - Load-Aware Adaptive Failover Strategy (LAAF)
    - failover targets chosen on the least loaded host hosting the backups
  - Resource Overload Management Redirector (ROME) strategy
    - clients are forcefully redirected to least loaded backups – overloads are treated as failures
  - LAAF & ROME adapt to changing system loads & resource availabilities



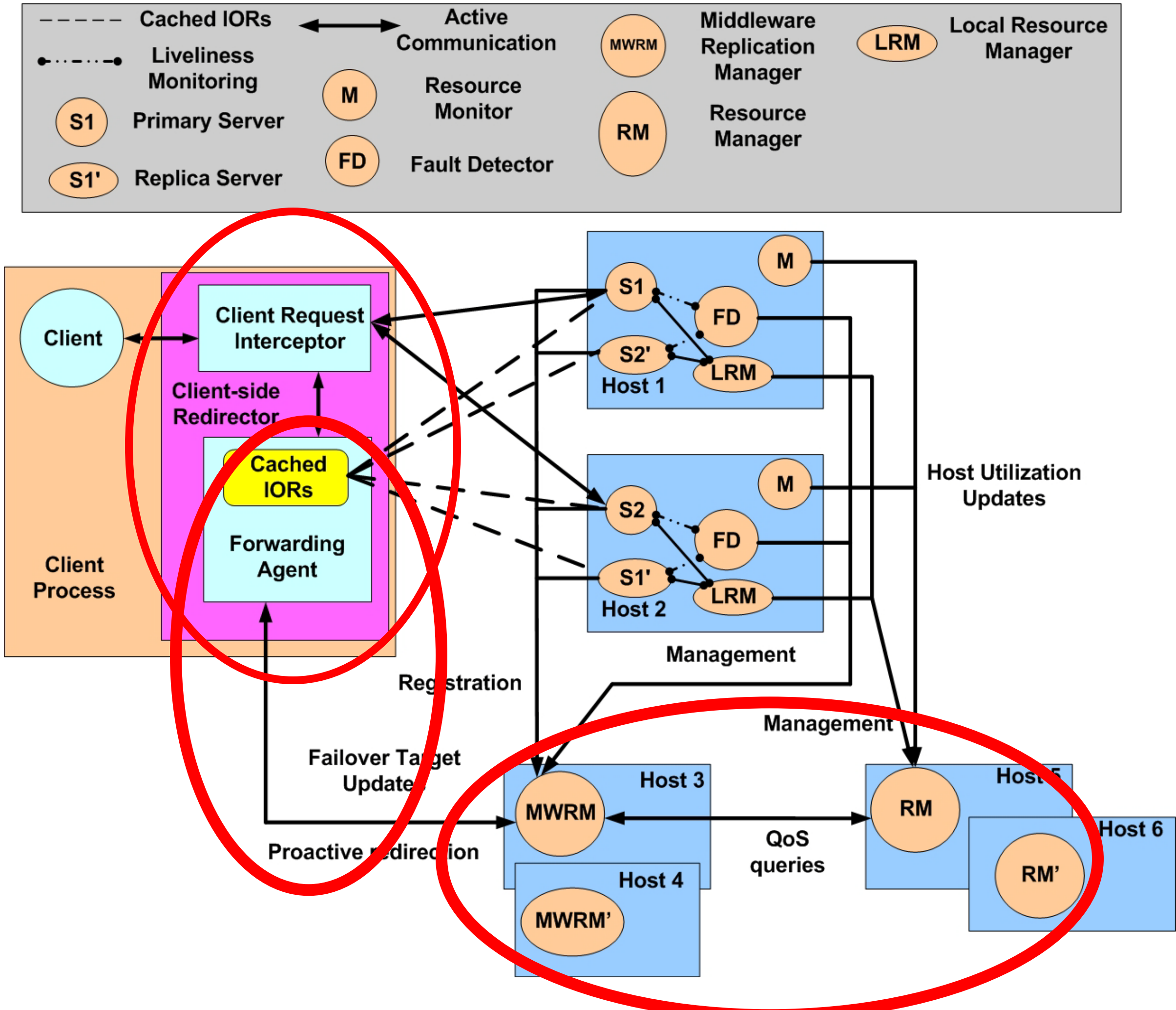
Our Approach: FLARe RT-FT Middleware

- **Transparent & Fast Failover**
  - Redirection using client-side portable interceptors
  - catches processor and process failure exceptions and redirects clients to alternate targets
  - Failure detection can be improved with better protocols – e.g., SCTP
    - middleware supports pluggable transports

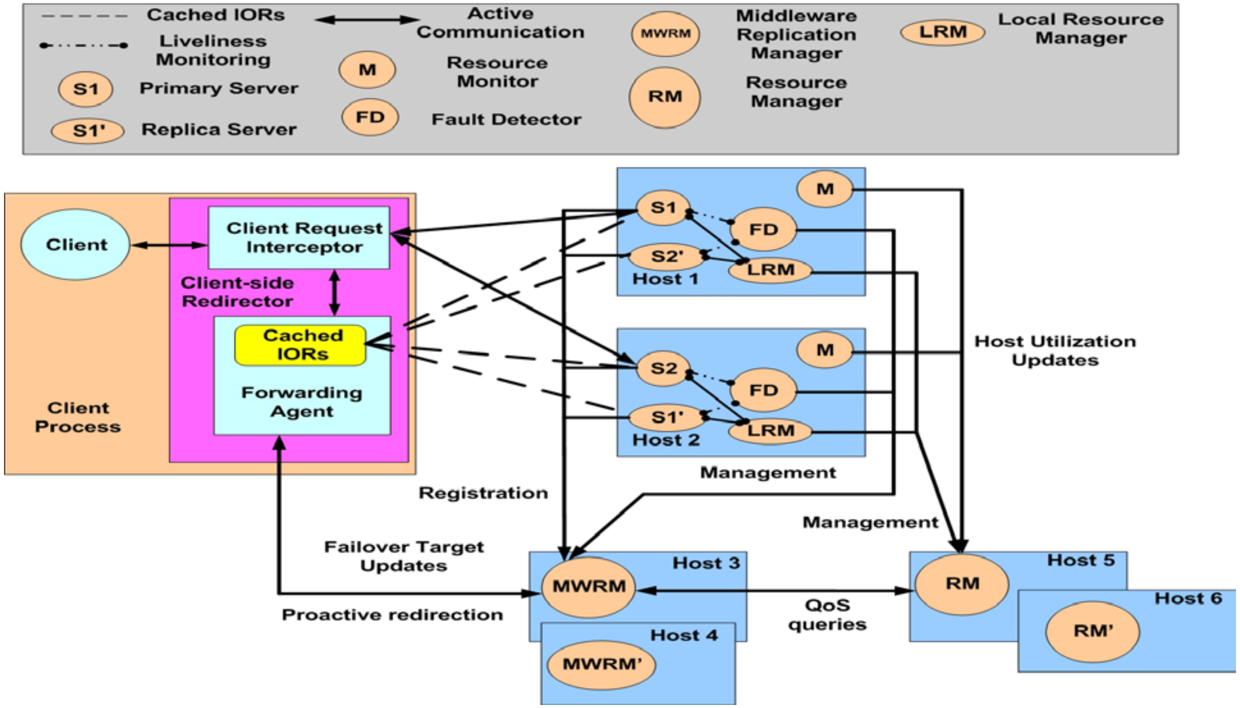


Our Approach: FLARe RT-FT Middleware

- **Predictable failover**
  - failover target decisions computed periodically by the resource manager
  - conveyed to client-side middleware agents – forwarding agents
  - agents work in tandem with portable interceptors
  - redirect clients quickly & predictably to appropriate targets
  - agents periodically/proactively updated when targets change

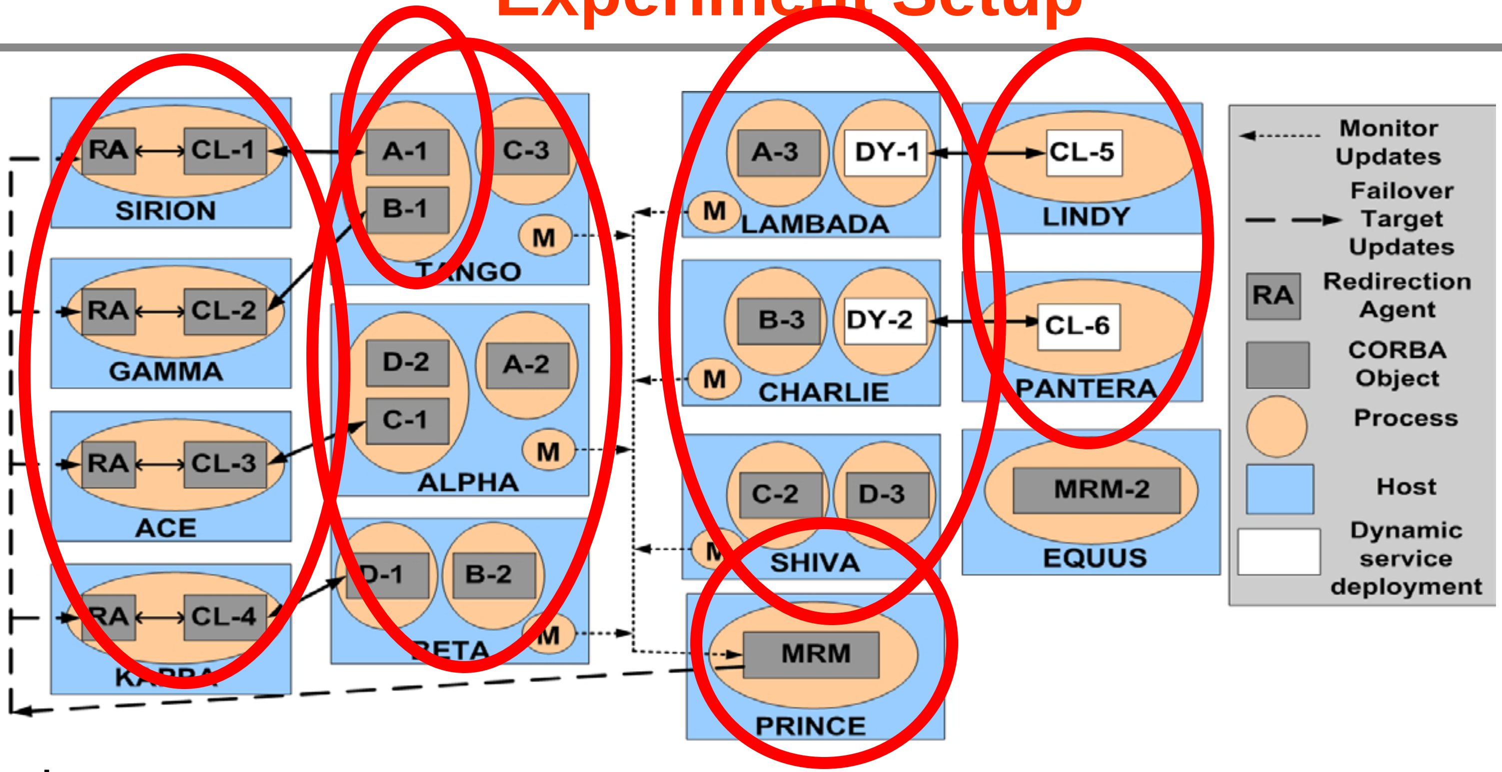


FLARe Evaluation Criteria



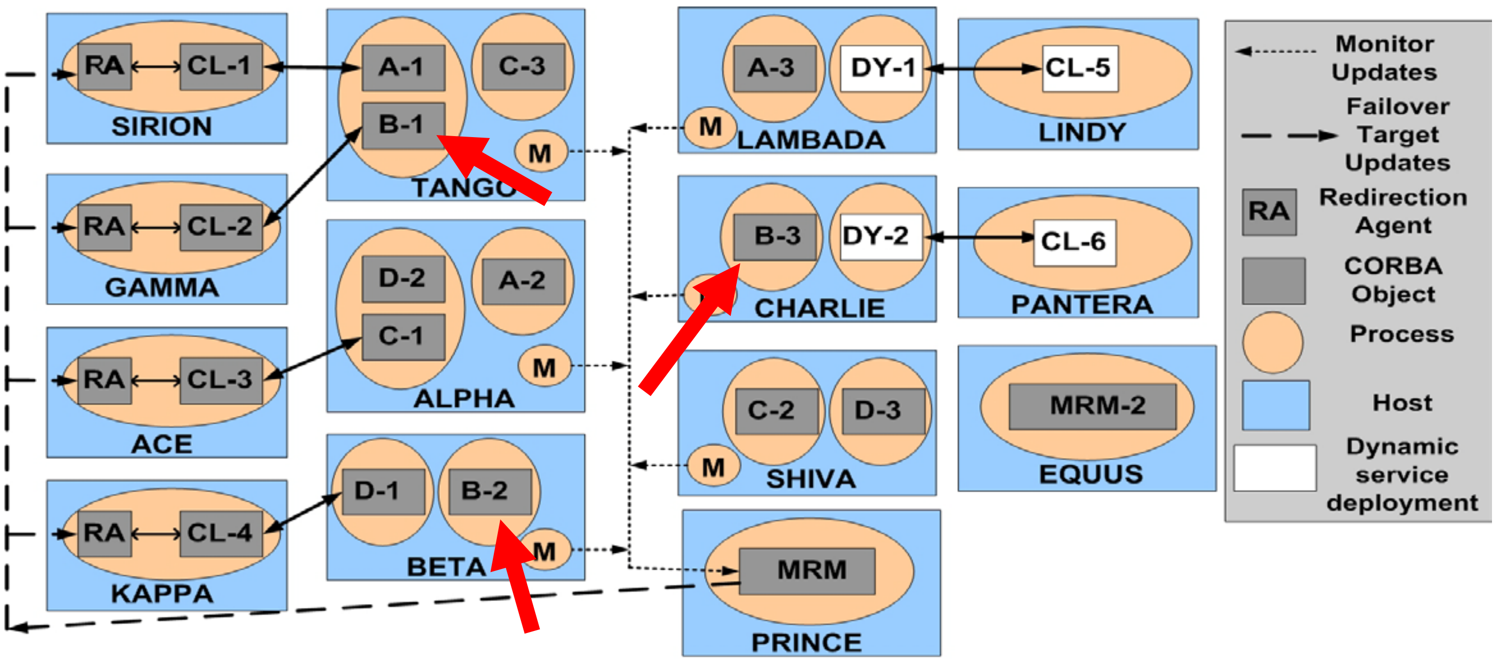
- Hypotheses: FLARe's
  - LAAF failover target selection strategy selects failover targets that maintain satisfactory response times for clients & alleviates processor overloads.
    - no processor's utilization is more than 70%
  - ROME overload management strategy reacts to overloads rapidly, selects appropriate targets to redirect clients, & maintains satisfactory response times for clients
    - no processor's utilization is more than 70%

Experiment Setup



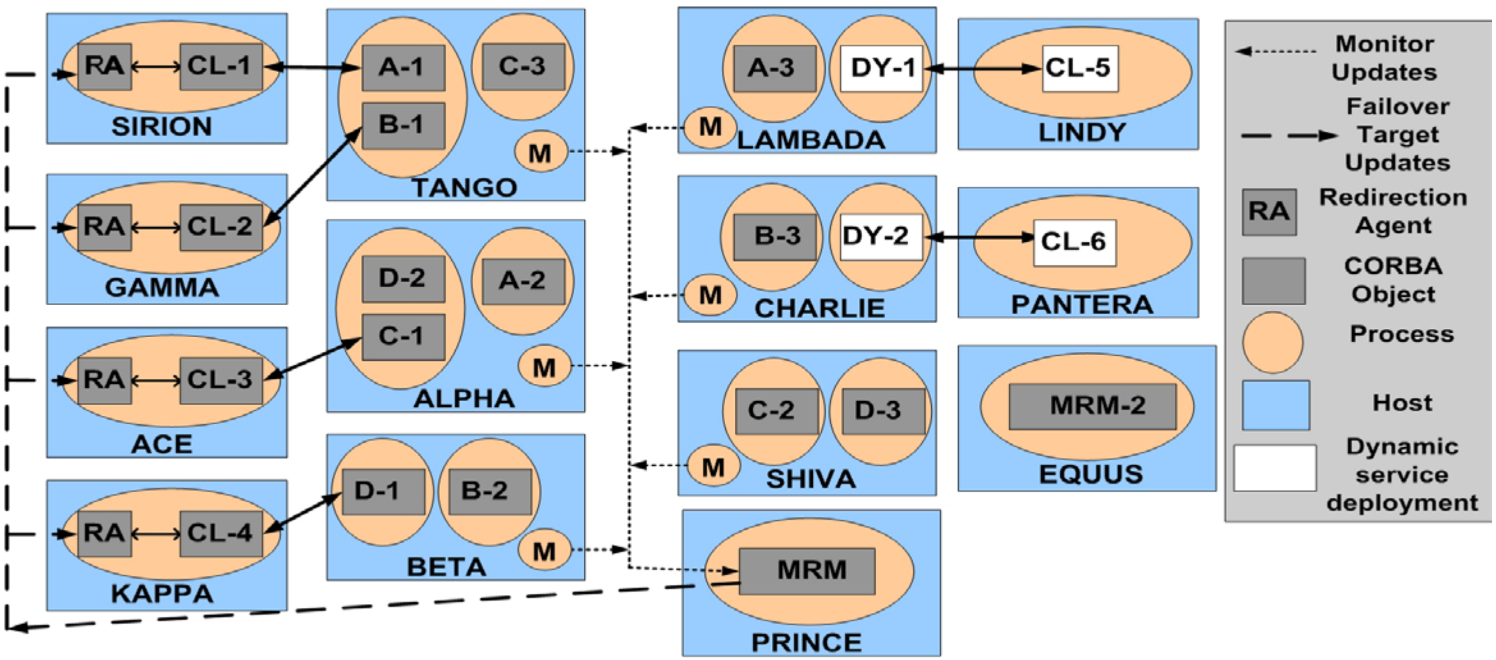
- Experiment setup
  - 6 different clients – 2 clients CL-5 & CL-6 are dynamic clients (start after 50 seconds)
  - 6 different servers – each have 2 replicas, 2 servers are dynamic as well
  - Each client has a forwarding agent deployed – they get the failover target information from the middleware replication manager
  - Experiment ran for 300 seconds – each server consumes some CPU load
    - some servers share processors – they follow rate-monotonic scheduling for prioritized access to CPU resources

Experiment Configurations



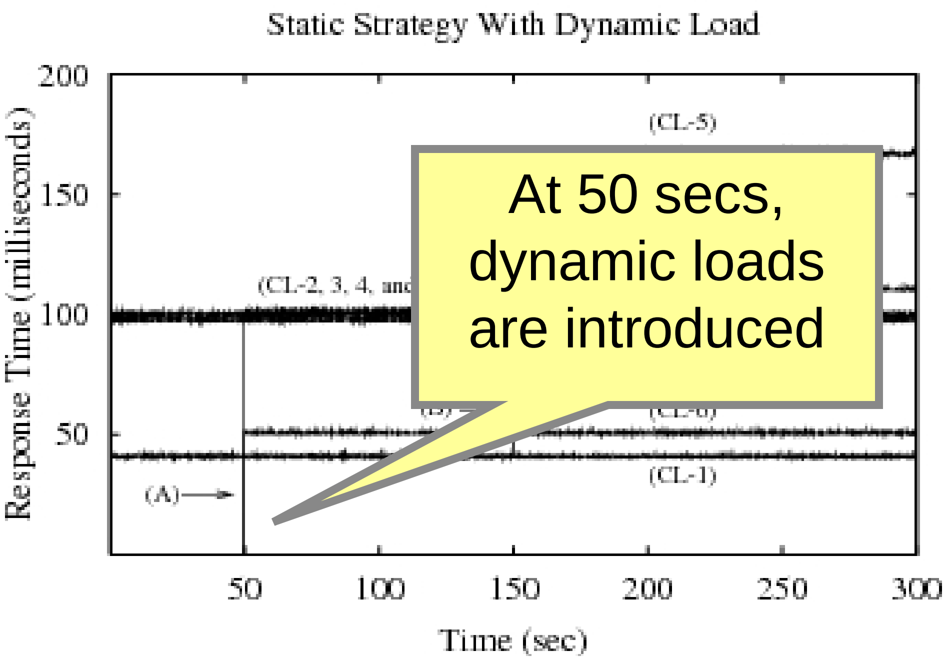
- Static Failover Strategy
  - each client knows the order in which they access the server replicas in the presence of failures – i.e., the failover targets are known in advance
  - for e.g., CL-2 makes remote invocations on B-1, on B-3 if B-1 fails, & on B-2 if B3-fails
  - this strategy is optimal at deployment-time (B-3 is on a processor lightly loaded than the processor hosting B-2)

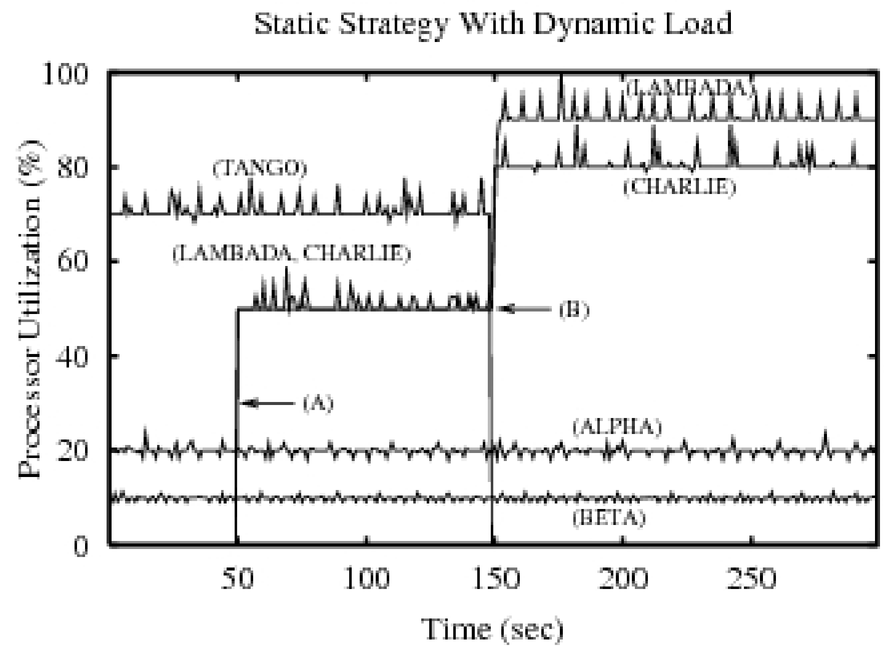
Experiment Configurations

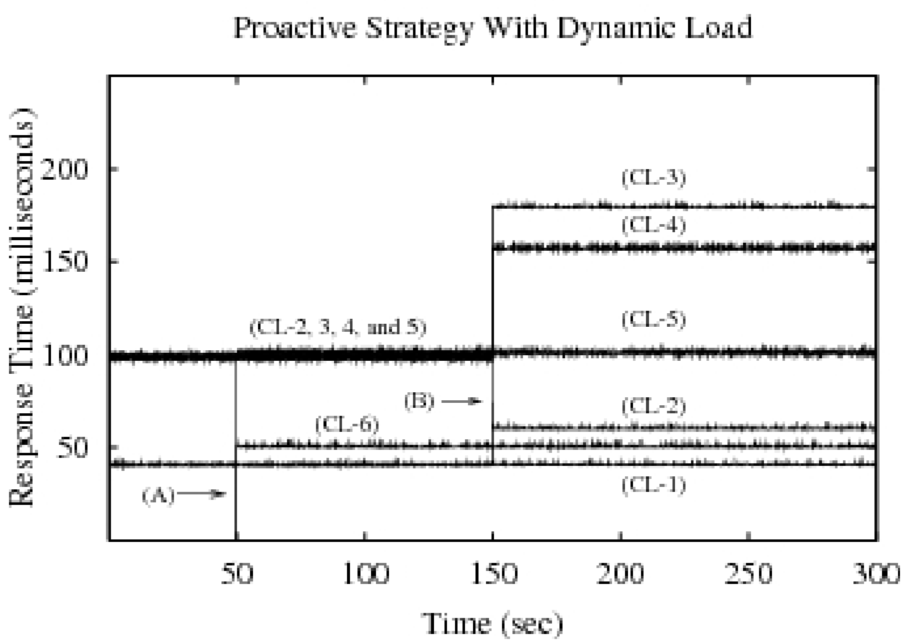


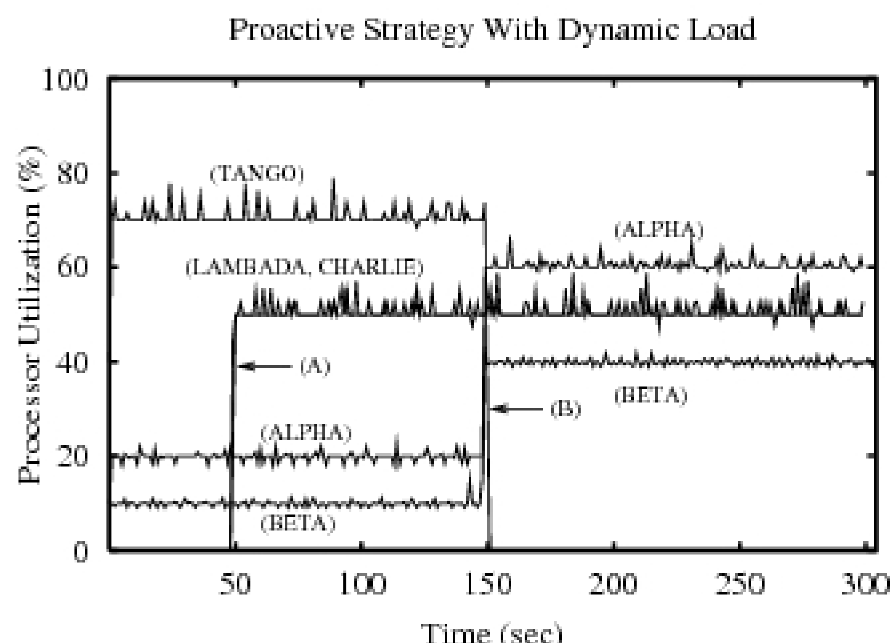
- LAAF Failover Strategy
  - each client knows only the reference of the primary replica
  - failover targets are determined at runtime while monitoring the CPU utilizations at all processors – that is why dynamic loads are added in the experiment

LAAF Algorithm Results

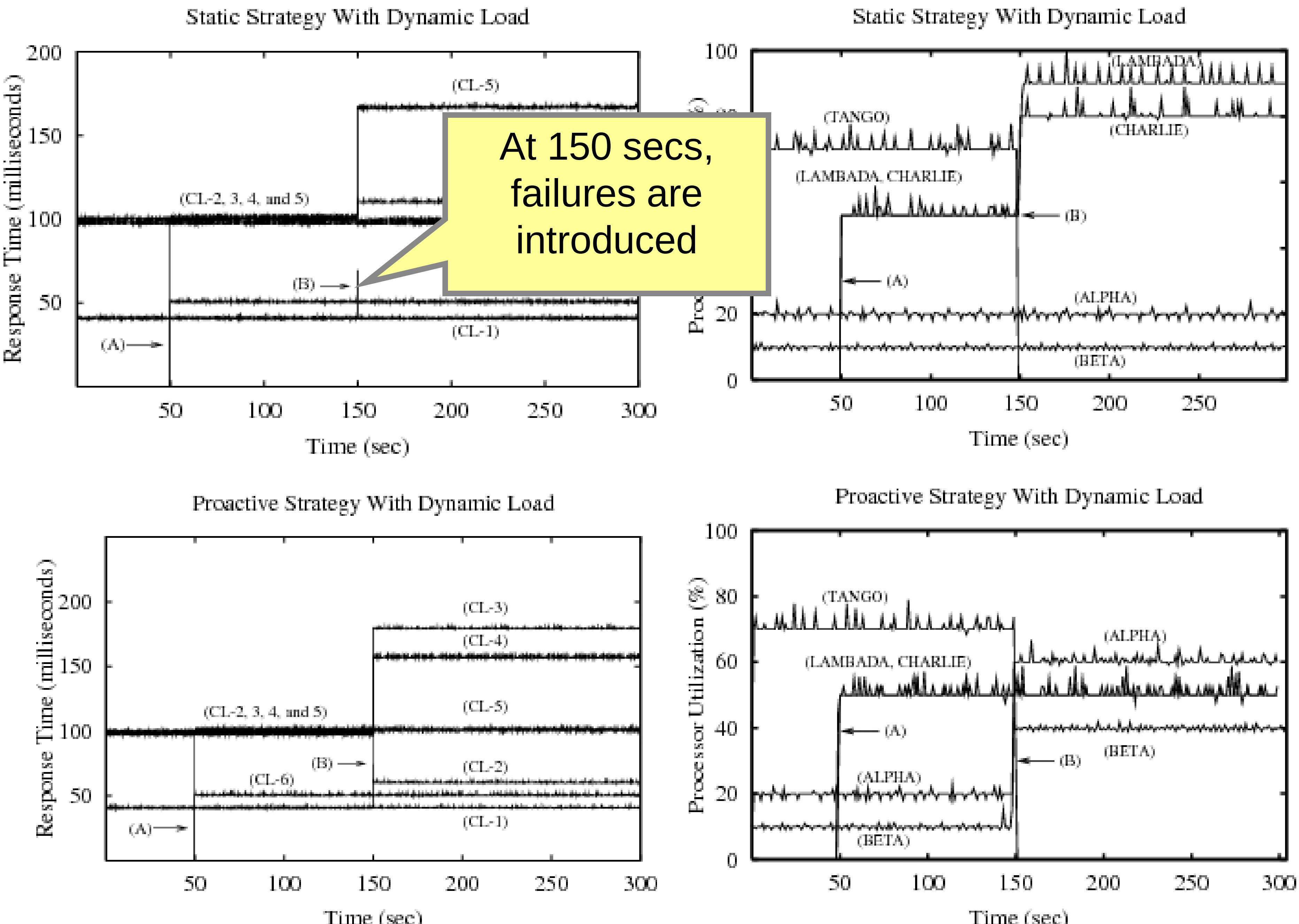




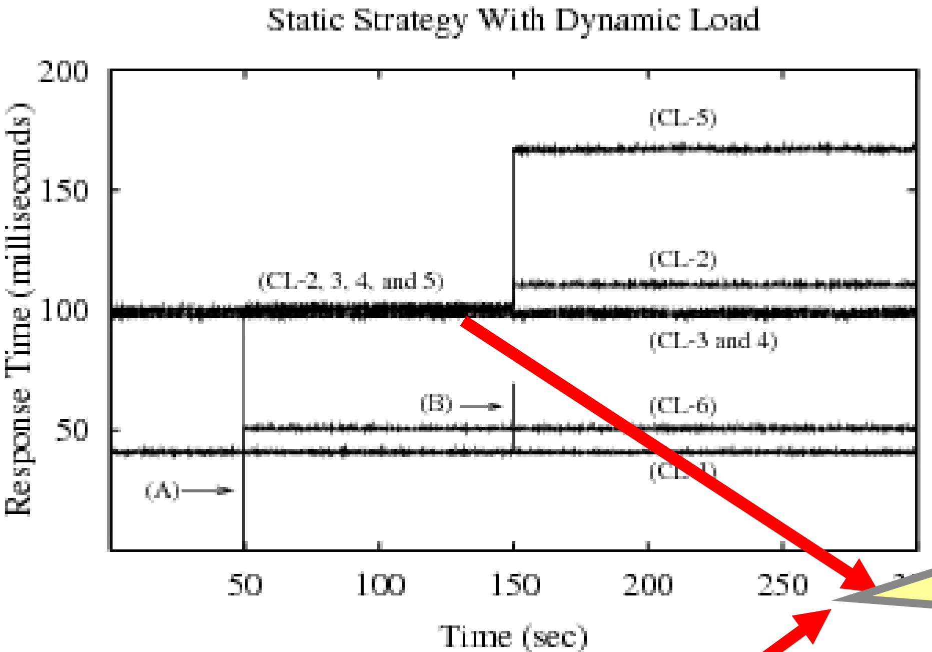


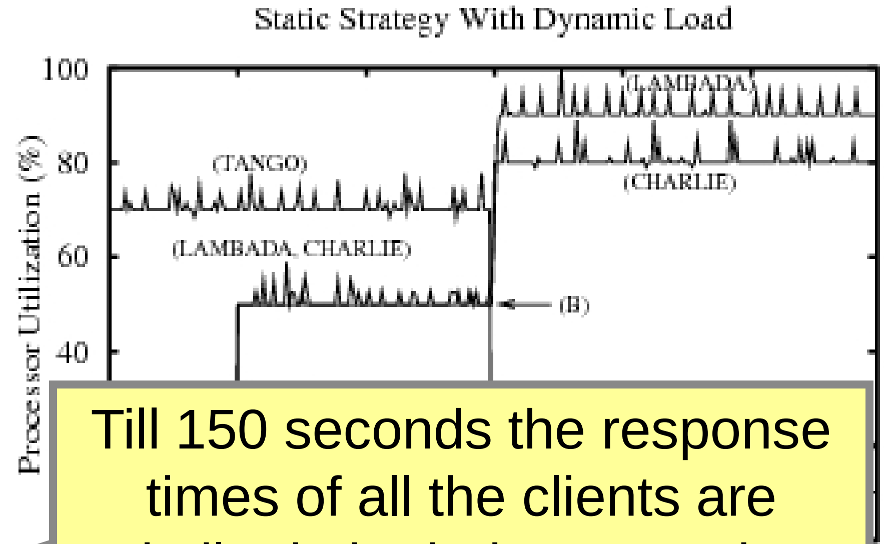


LAAF Algorithm Results

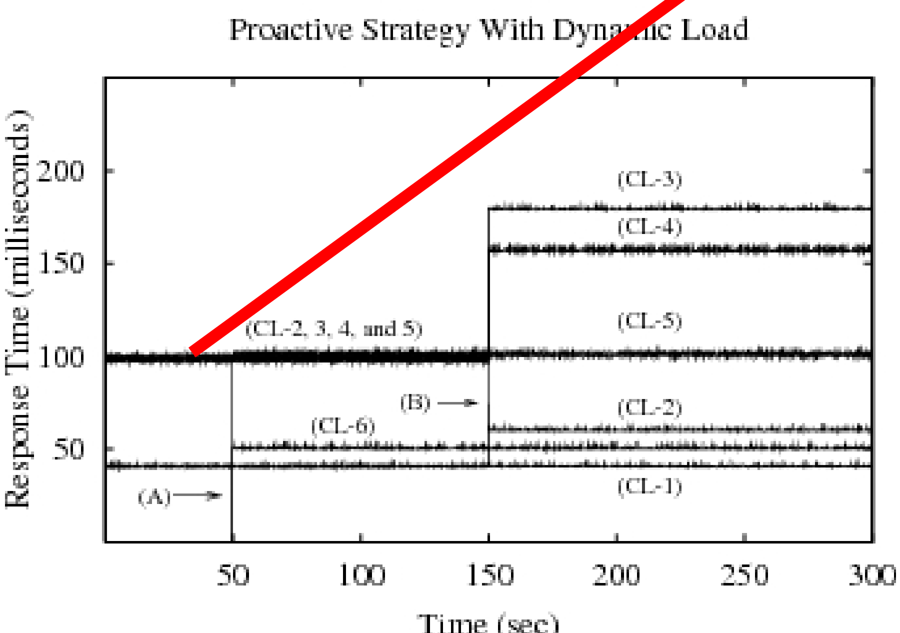


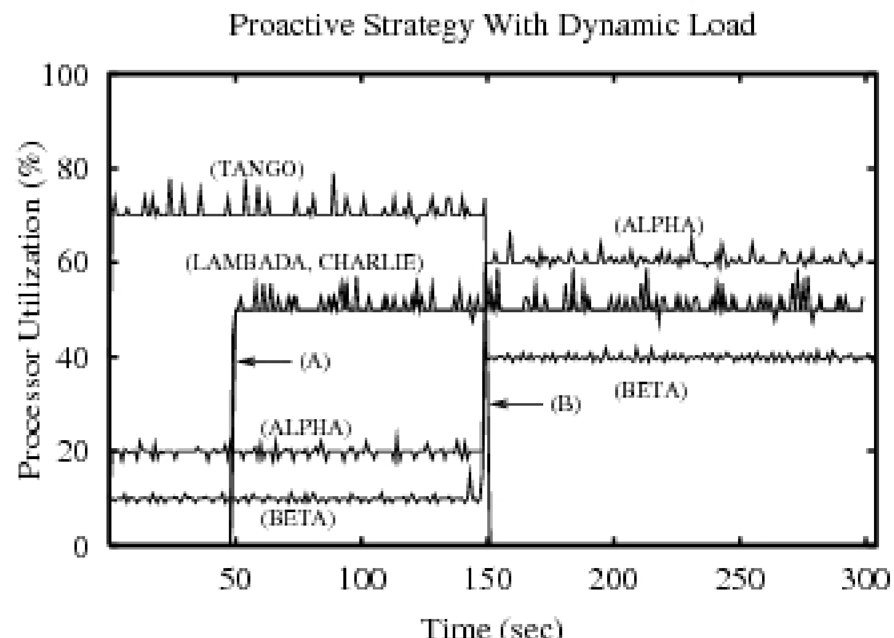
LAAF Algorithm Results



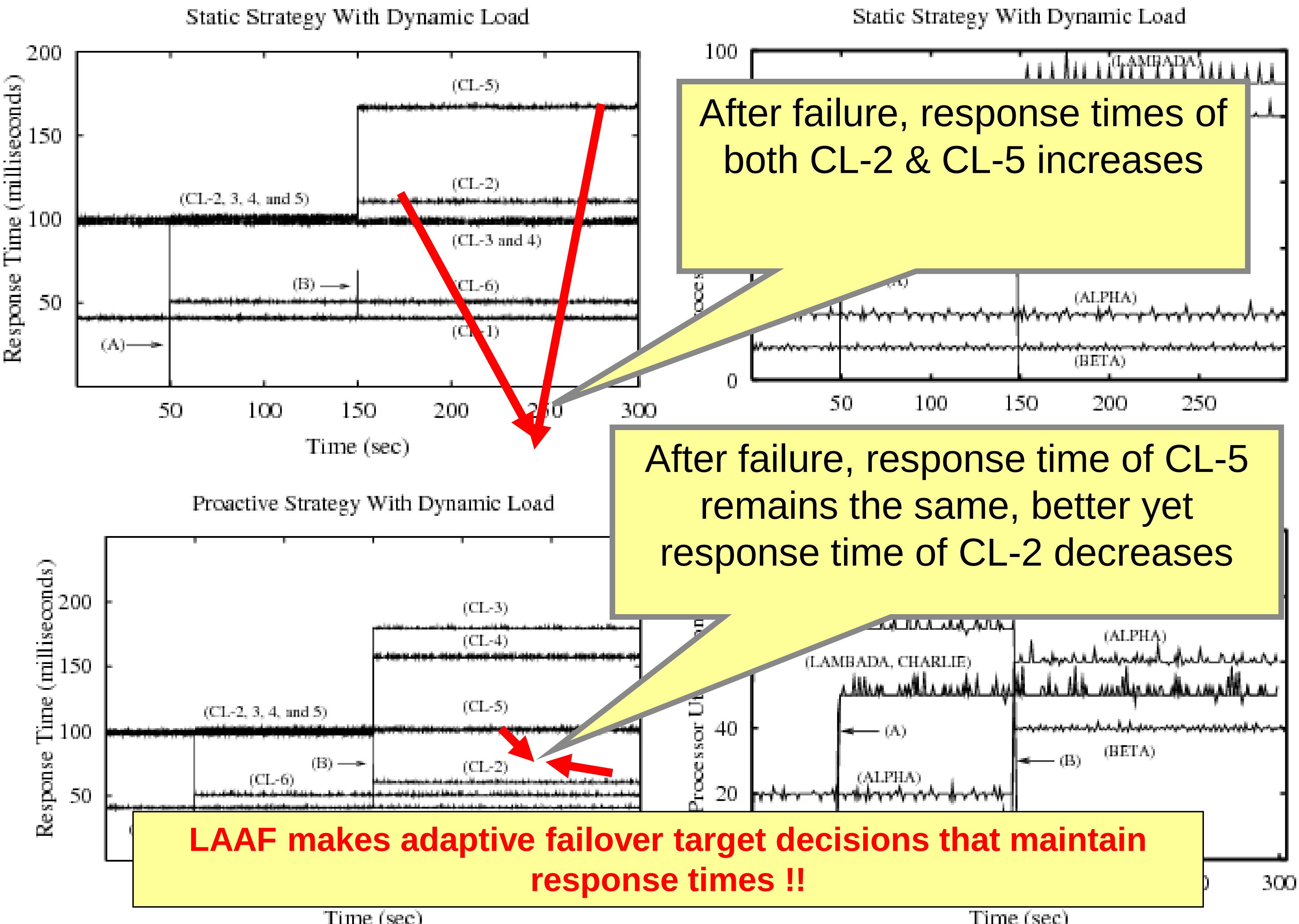


Till 150 seconds the response times of all the clients are similar in both the strategies





LAAF Algorithm Results

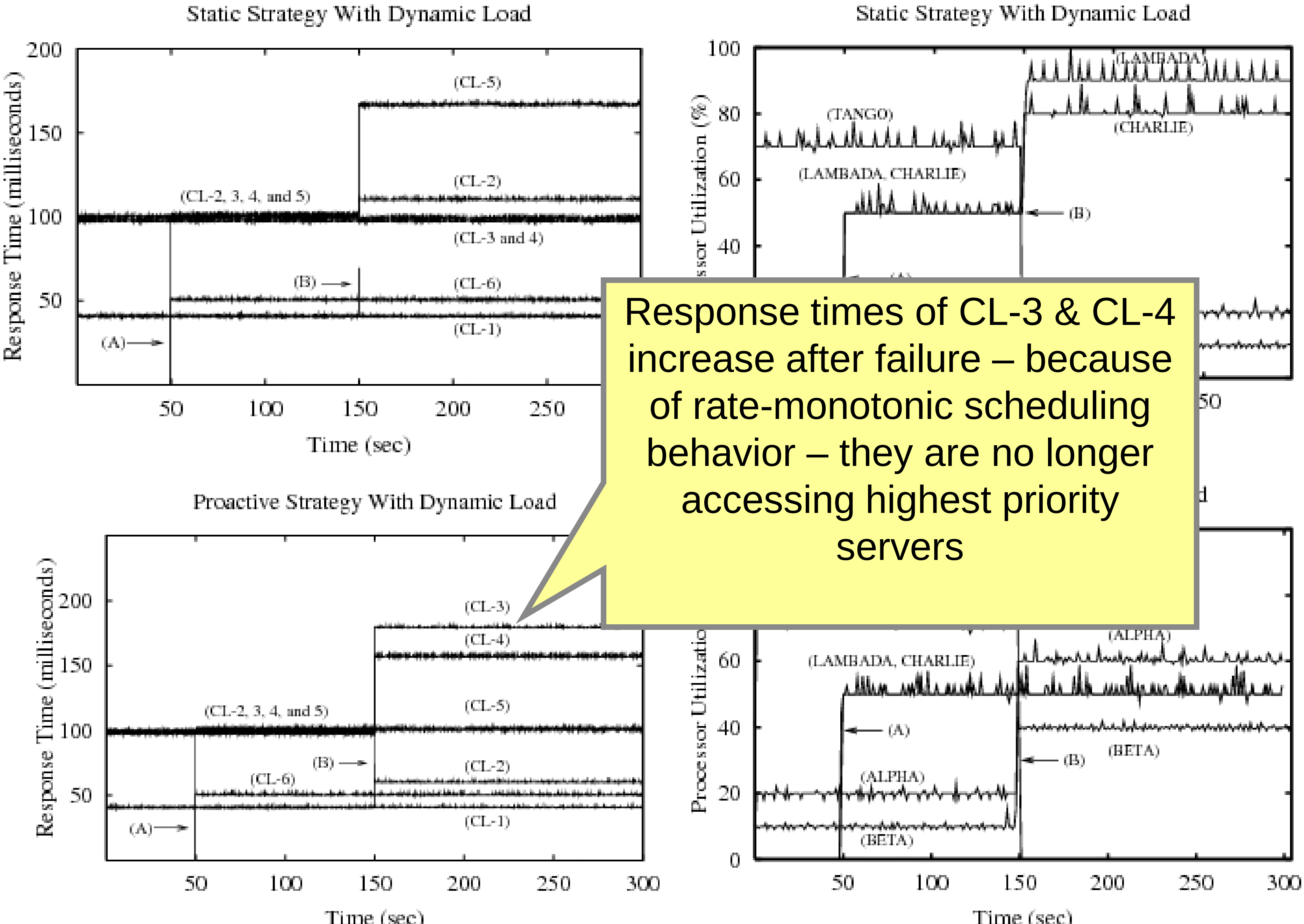


After failure, response times of both CL-2 & CL-5 increases

After failure, response time of CL-5 remains the same, better yet response time of CL-2 decreases

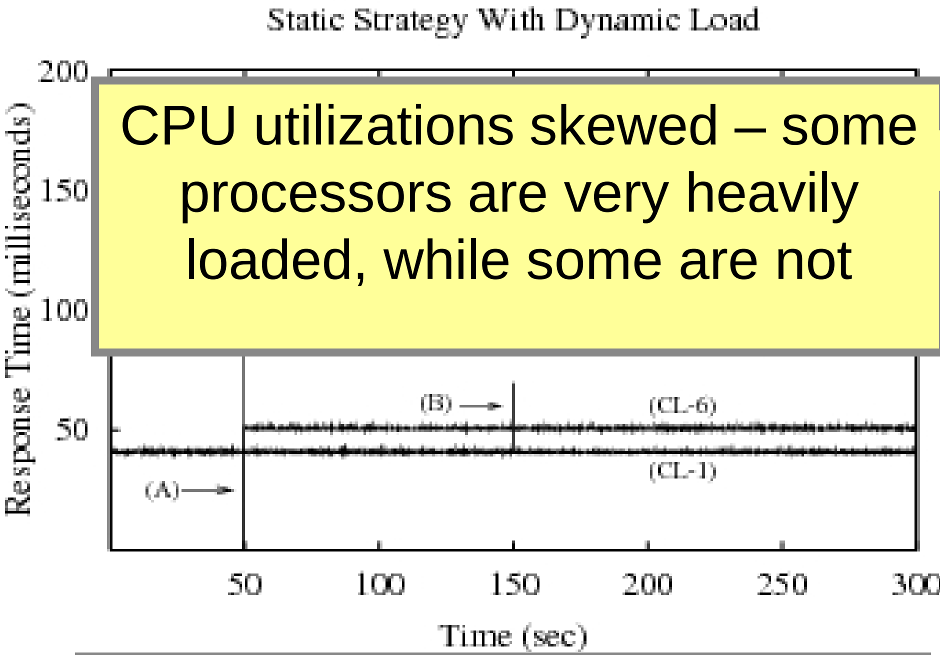
LAAF makes adaptive failover target decisions that maintain response times !!

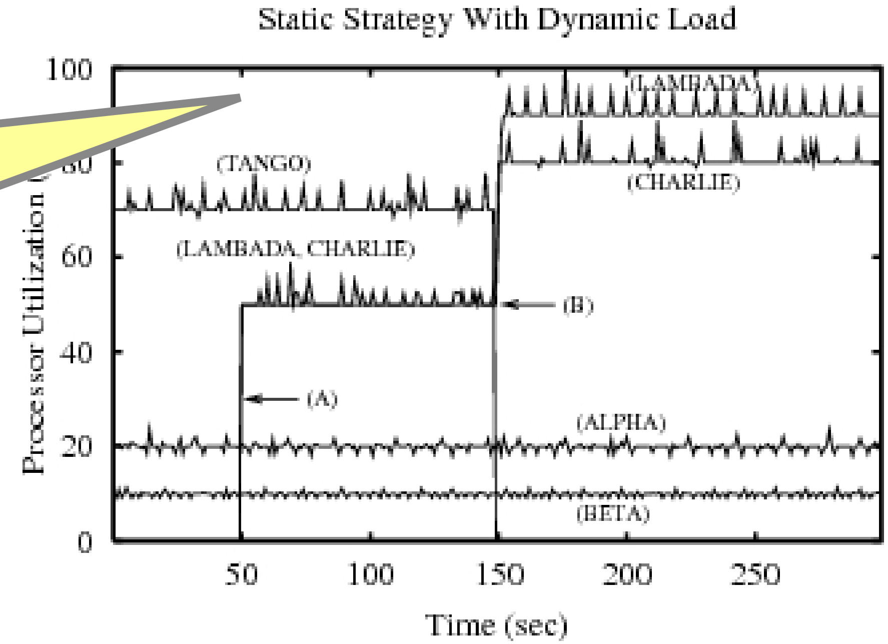
LAAF Algorithm Results



Response times of CL-3 & CL-4 increase after failure – because of rate-monotonic scheduling behavior – they are no longer accessing highest priority servers

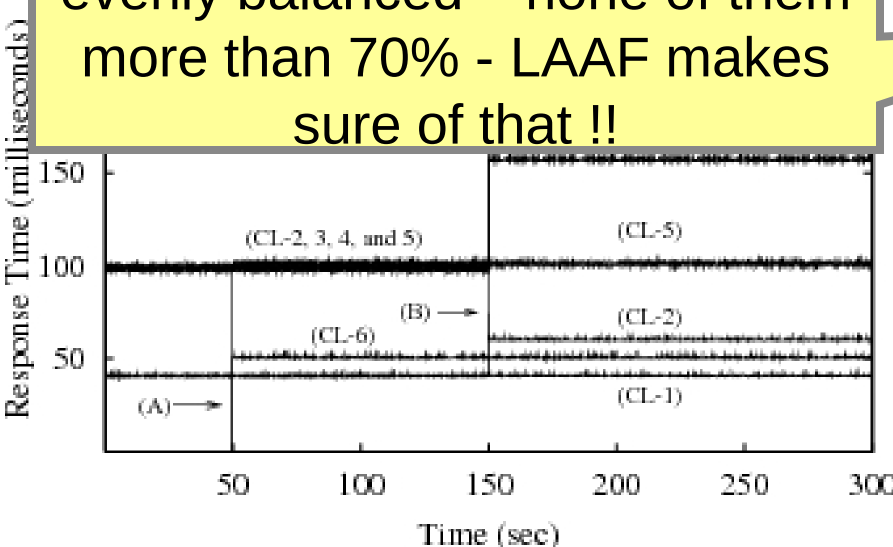
LAAF Algorithm Results

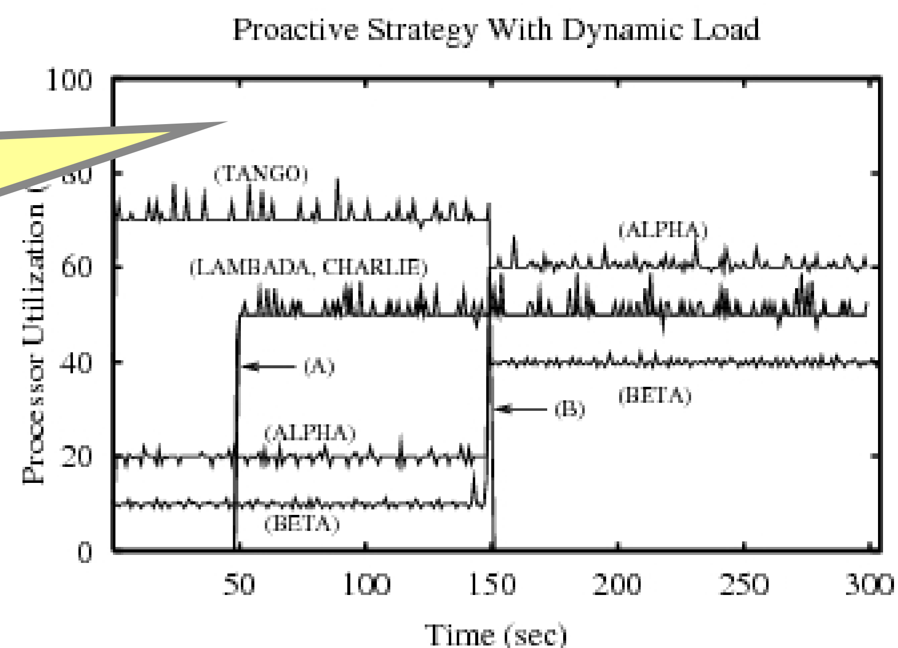




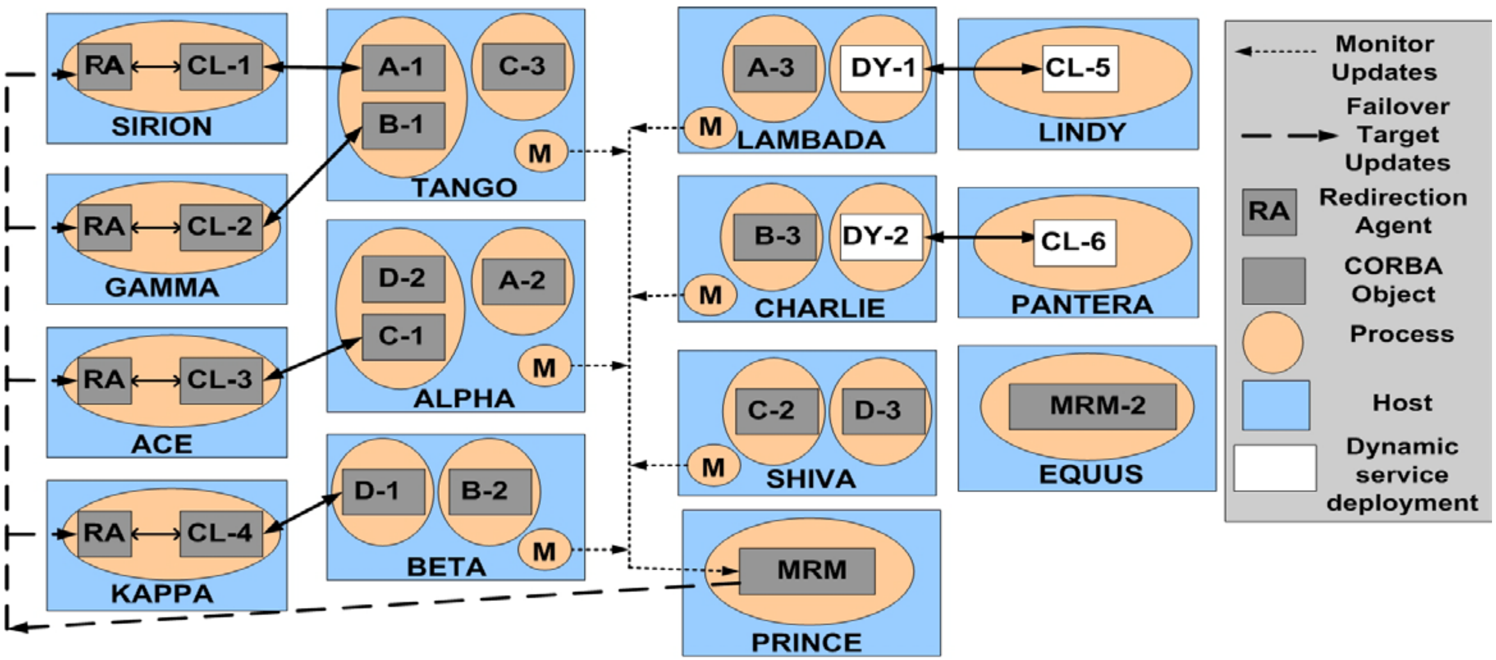
CPU utilizations skewed – some processors are very heavily loaded, while some are not

CPU utilizations are more evenly balanced – none of them more than 70% - LAAF makes sure of that !!



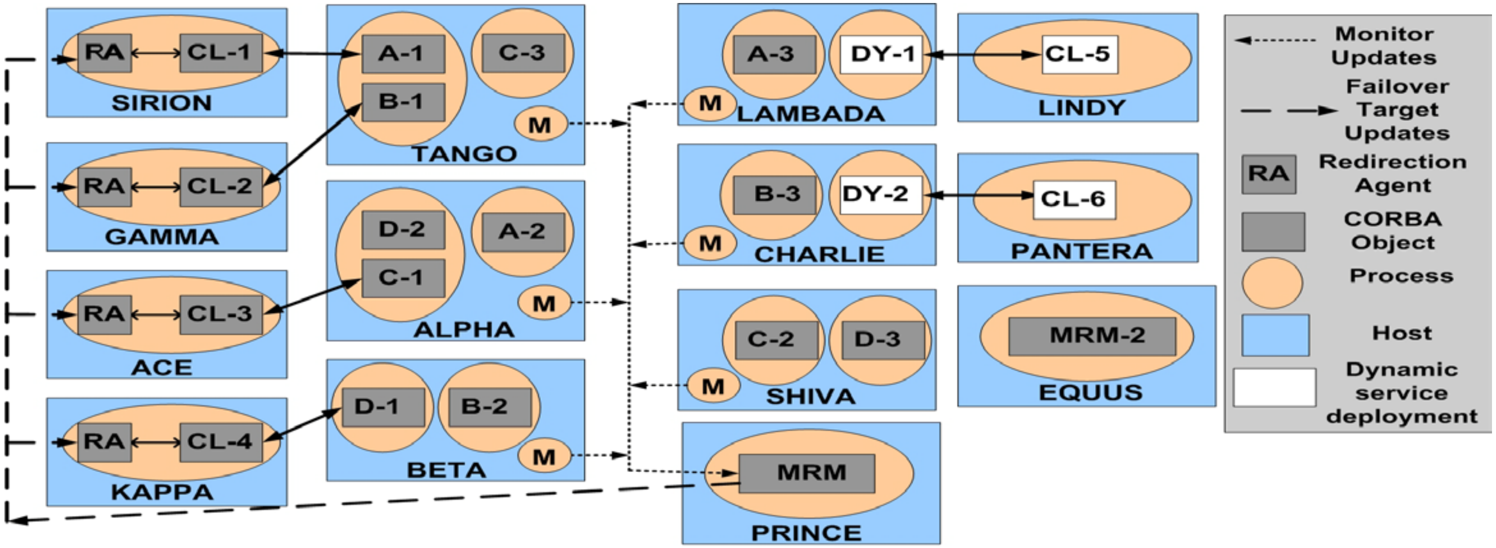


Summary of Results



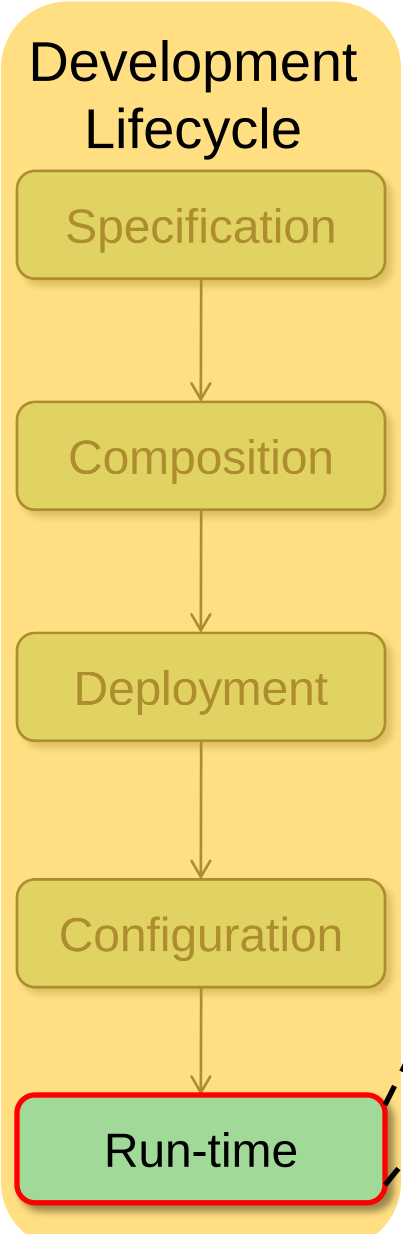
- FLARe's LAAF failover strategy maintains client response times & processor utilizations after failure recovery when compared to the static failover strategy (no processor is utilized more than 70%)
  - LAAF failover strategy always adapts the failover targets whenever system loads change – client failover to the least loaded backup
  - static failover strategy does not change the previously deployment-time optimal failover targets at runtime
    - client failover results in overload & hence higher response times

Summary of FLARe Results



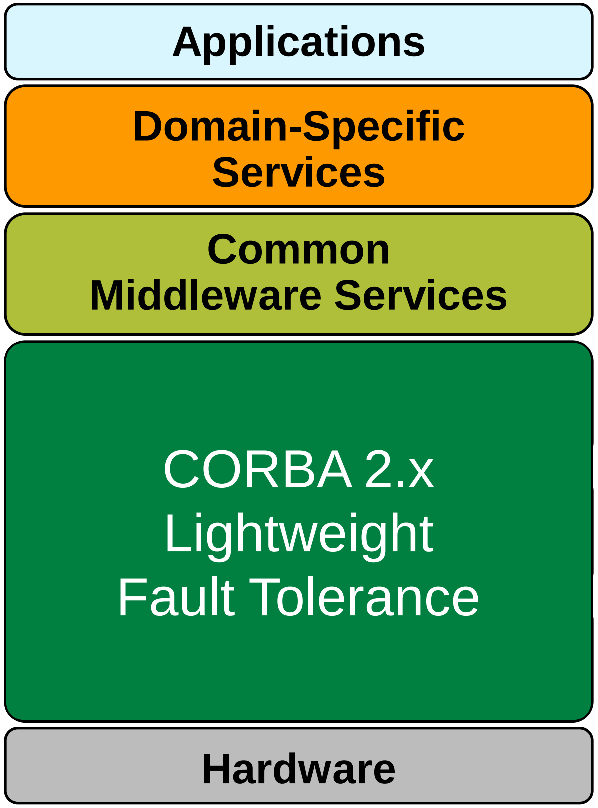
- ROME strategy reacts to overloads & maintains client response times – no processor is utilized more than 70%

Runtime Phase: Component-based Fault Tolerance



- Component Replication-based on Failover Units (CORFU)
  - Raise the level of fault tolerance to component level
  - Support group failover

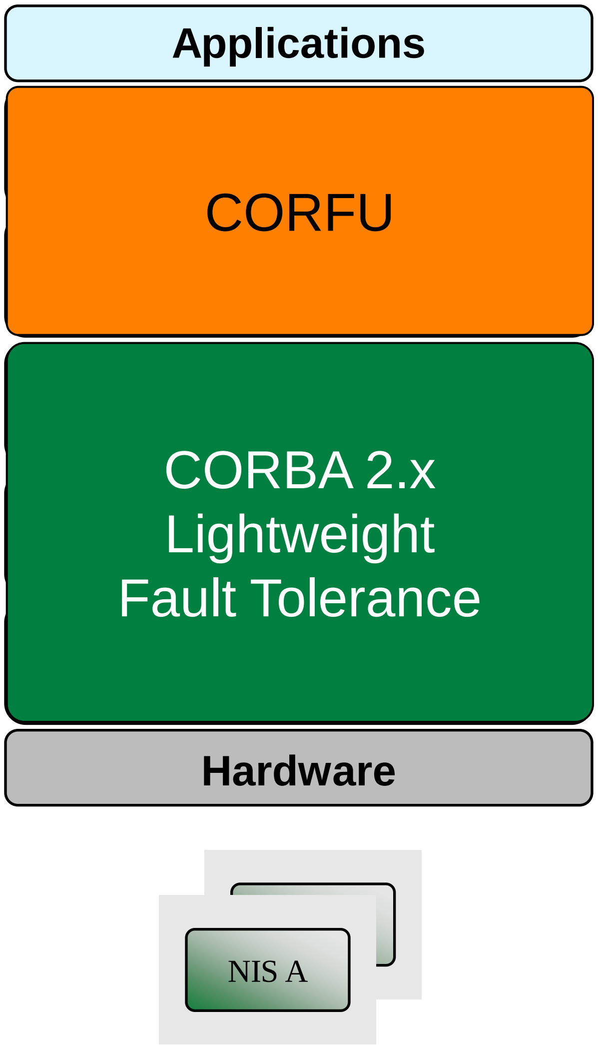
CORFU Contributions



Component Replication Based on Failover Units (CORFU)

- Raises the level of abstraction, from objects to

CORFU Contributions

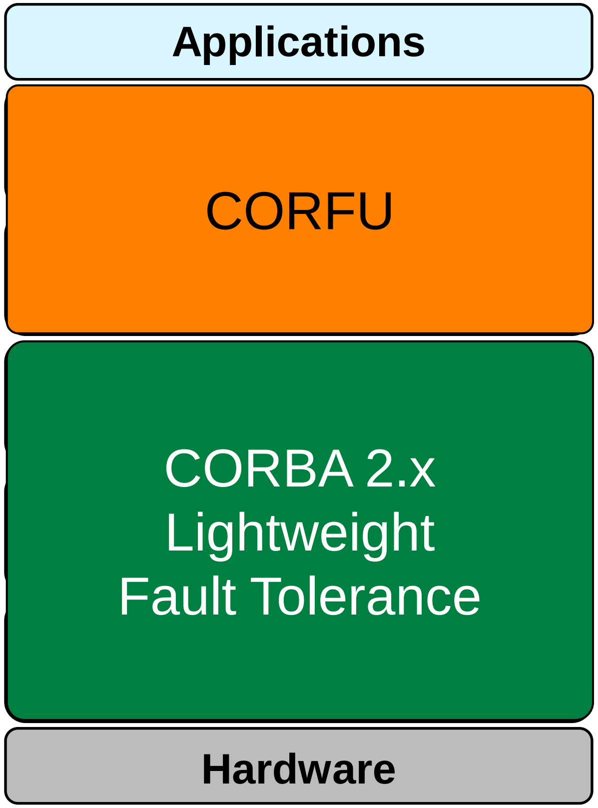


Component Replication Based on Failover Units (CORFU)

- Raises the level of abstraction, from objects to

  a) Fault-tolerance for single components

CORFU Contributions

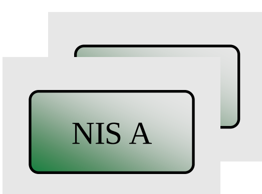


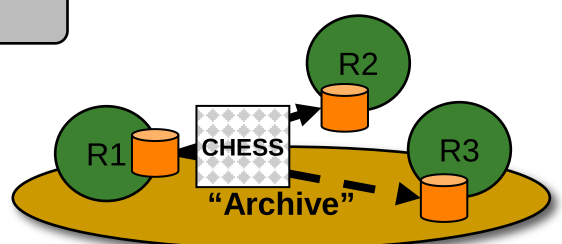
Component Replication Based on Failover Units (CORFU)

- Raises the level of abstraction, from objects to

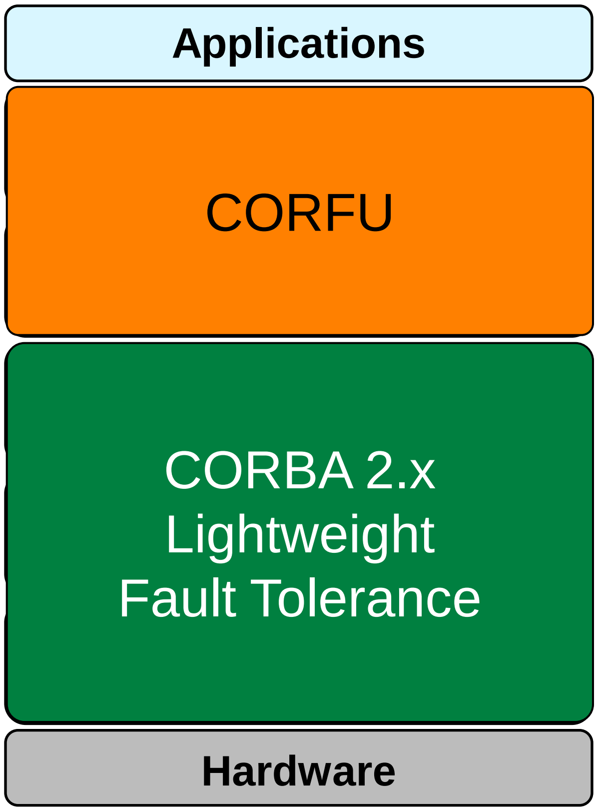
    a) Fault-tolerance for single components

    b) Components with Heterogenous State Synchronisation (CHESS)





CORFU Contributions



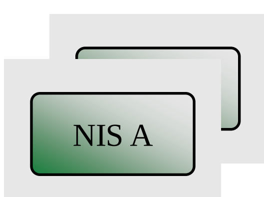
Component Replication Based on Failover Units (CORFU)

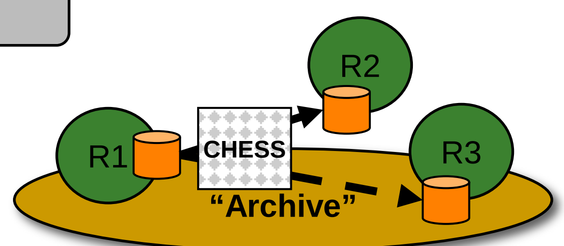
- Raises the level of abstraction, from objects to

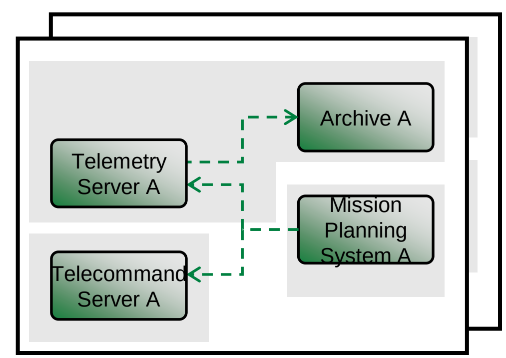
  a) Fault-tolerance for single components

  b) Components with Heterogenous State Synchronisation (CHESS)

  c) Fault-tolerance for groups of components



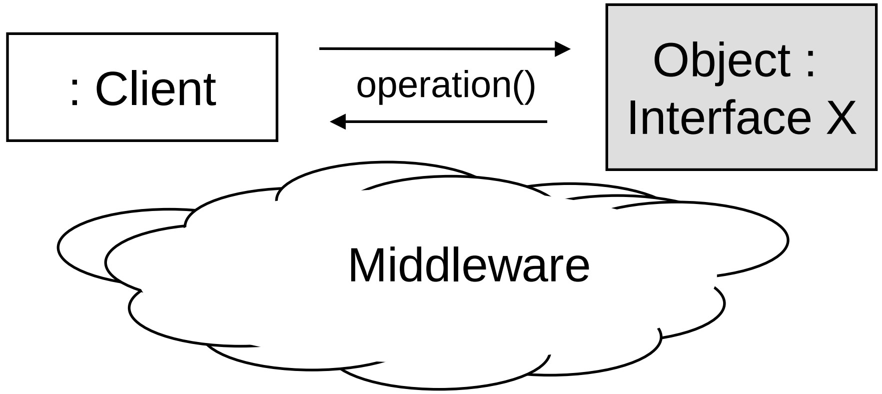




Bridges the abstraction gap for fault-tolerance

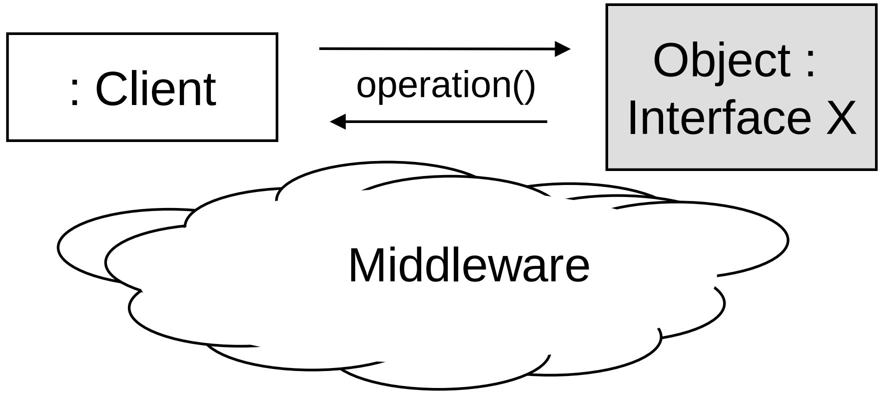
Prior Work: Object-based Fault Tolerance

- Conventional Fault-Tolerance solutions provide replication capabilities on the granularity of objects



Prior Work: Object-based Fault Tolerance

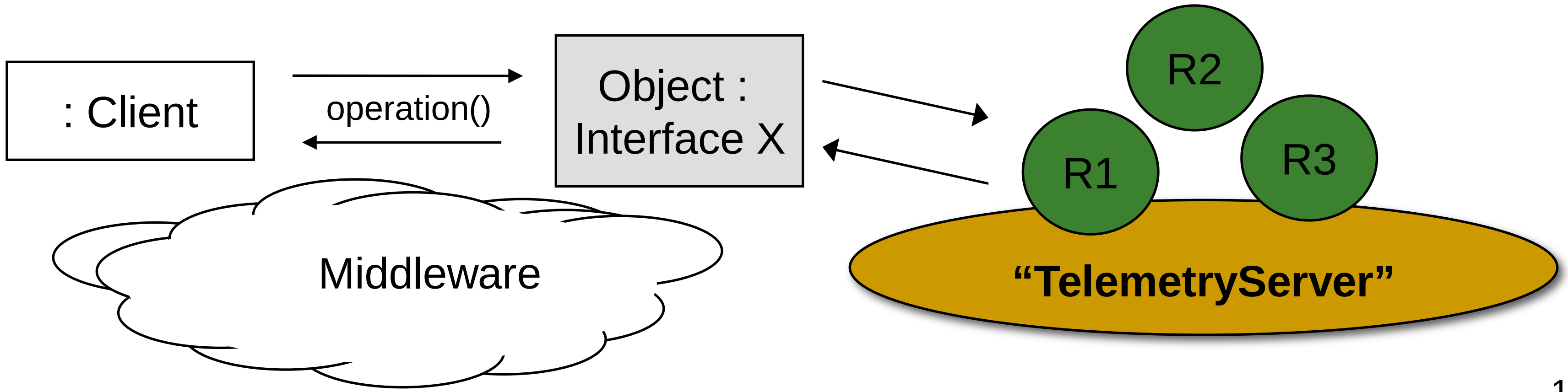
- Conventional Fault-Tolerance solutions provide replication capabilities on the granularity of objects
- FLARe takes a lightweight approach for DRE systems based on passive replication



Prior Work: Object-based Fault Tolerance

- Conventional Fault-Tolerance solutions provide replication capabilities on the granularity of objects
- FLARe takes a lightweight approach for DRE systems based on passive replication
- It provides mechanisms for

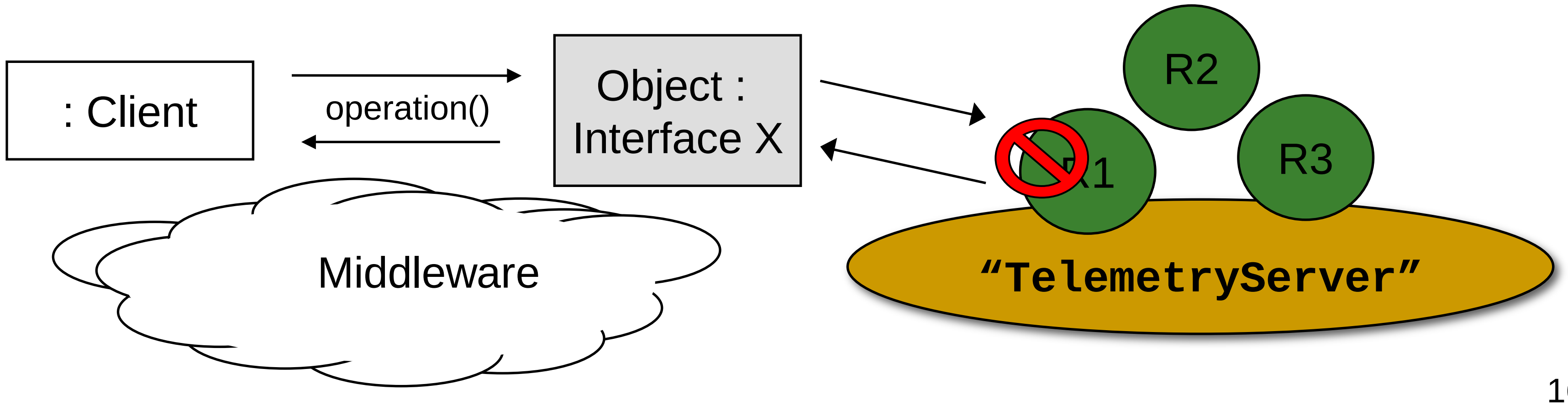
  1. Grouping of replica objects as one logical application



Prior Work: Object-based Fault Tolerance

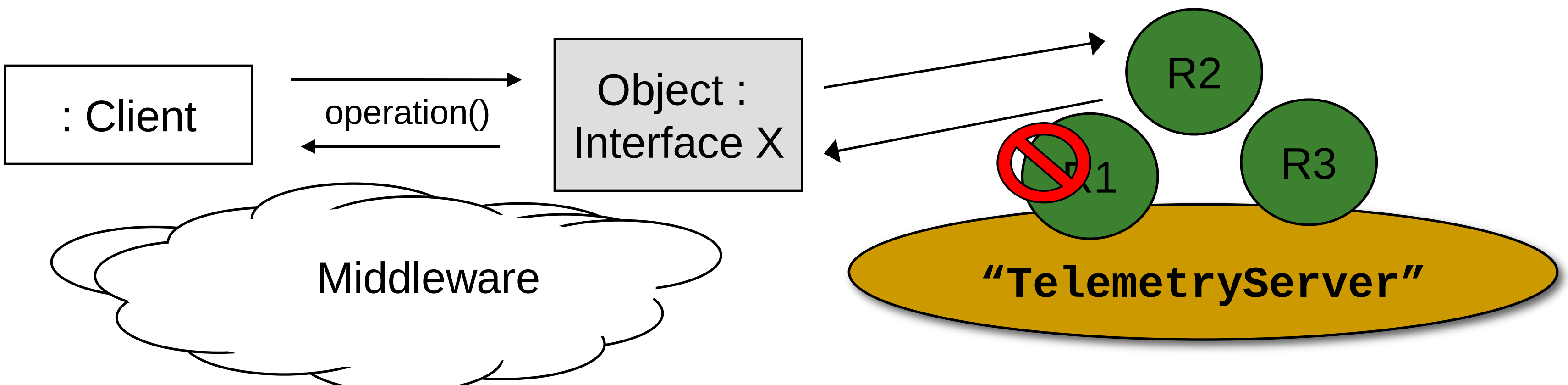
- Conventional Fault-Tolerance solutions provide replication capabilities on the granularity of objects
- FLARe takes a lightweight approach for DRE systems based on passive replication
- It provides mechanisms for

  1. Grouping of replica objects as one logical application
  2. Failure detection



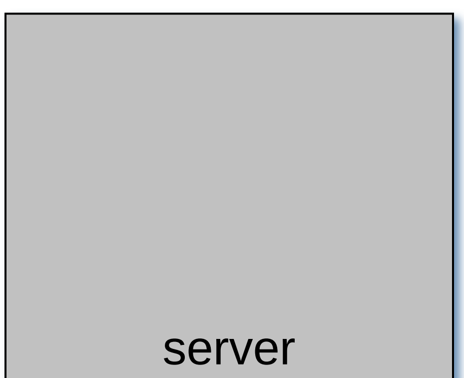
Prior Work: Object-based Fault Tolerance

- Conventional Fault-Tolerance solutions provide replication capabilities on the granularity of objects
- FLARe takes a lightweight approach for DRE systems based on passive replication
- It provides mechanisms for
  1. Grouping of replica objects as one logical application
  2. Failure detection
  3. Failover to backup replica



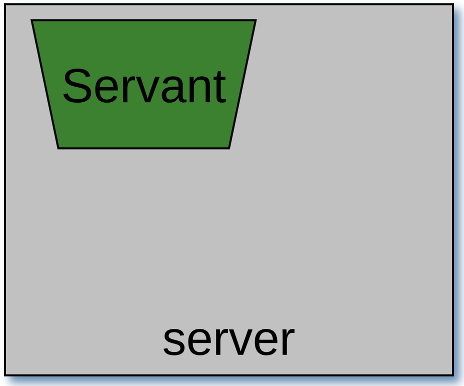
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		



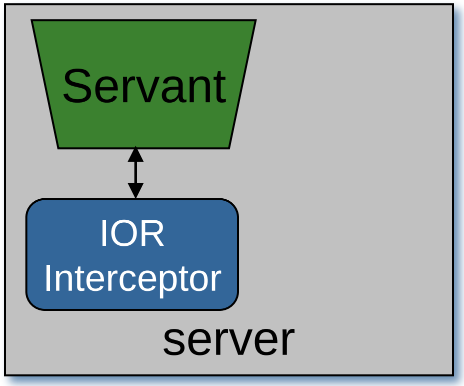
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		
Object Implementation		
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes		



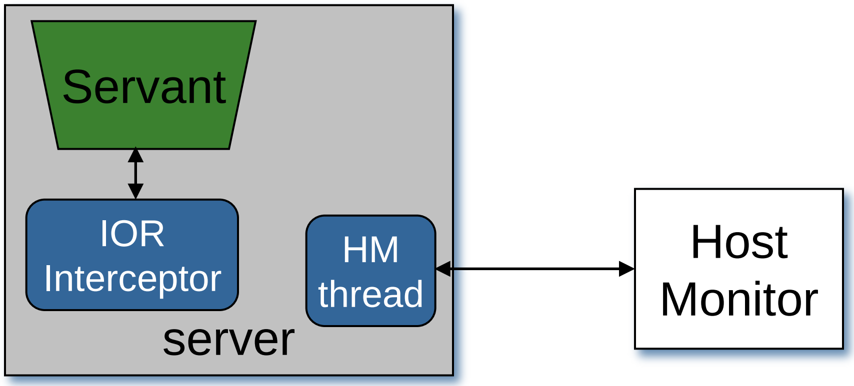
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		
Object Implementation	Initialization	
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes	1. Registration of IORInterceptor	



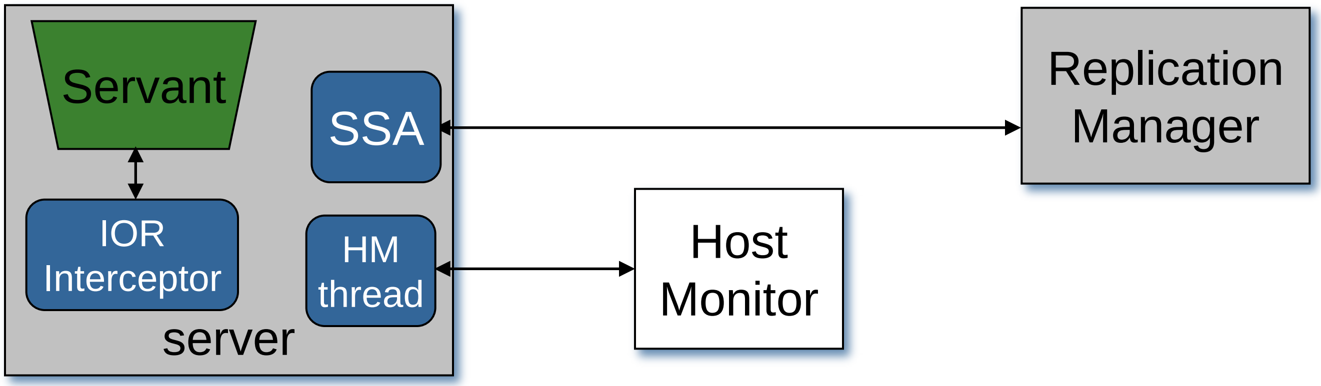
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		
Object Implementation	Initialization	
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes	1. Registration of IORInterceptor 2. HostMonitor thread instantiation 3. Registration of thread with HostMonitor	



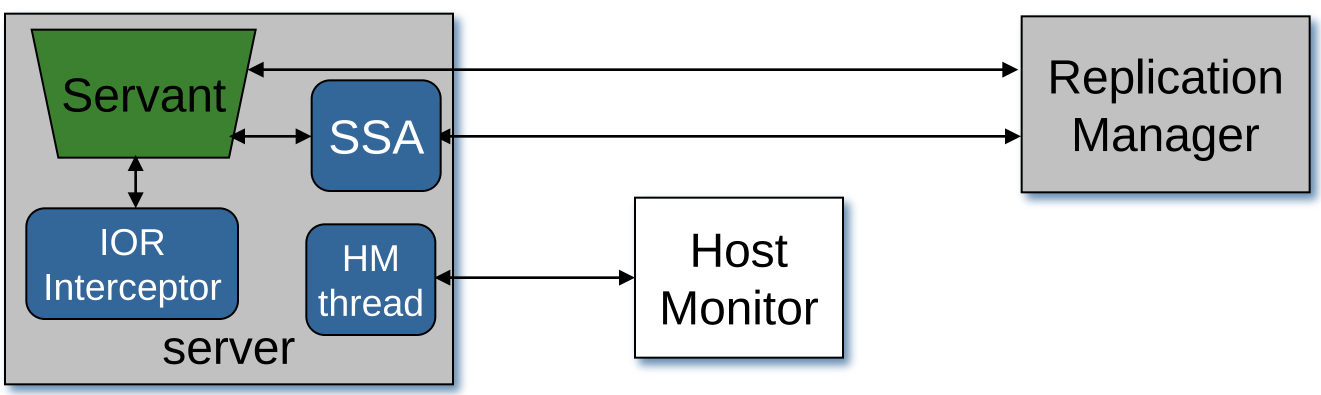
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		
Object Implementation	Initialization	
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes	1. Registration of IORInterceptor 2. HostMonitor thread instantiation 3. Registration of thread with HostMonitor 4. StateSynchronizationAgent instantiation 5. Registration of State Synchronization Agent with Replication Manager	



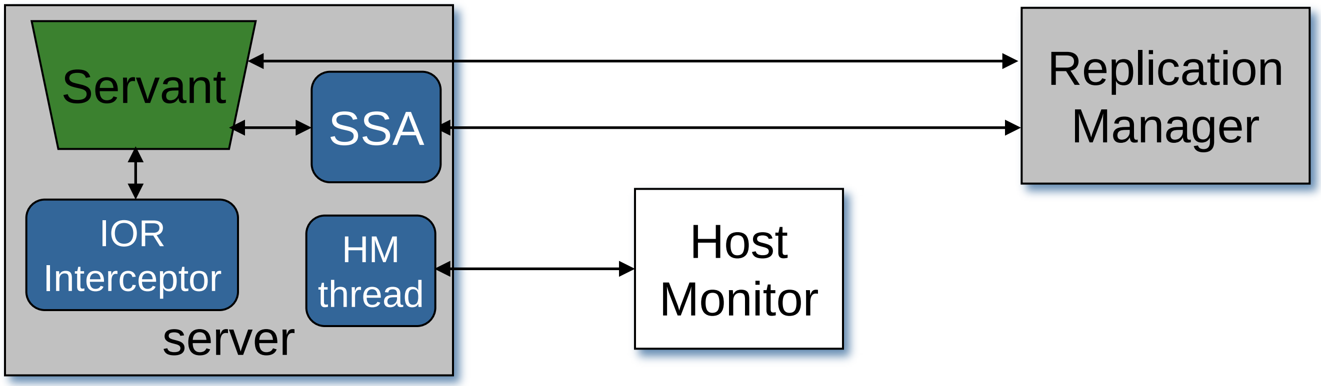
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		
Object Implementation	Initialization	
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes	1. Registration of IORInterceptor 2. HostMonitor thread instantiation 3. Registration of thread with HostMonitor 4. StateSynchronizationAgent instantiation 5. Registration of State Synchronization Agent with Replication Manager 6. Registration with State Synchronization Agent for each object 7. Registration with Replication Manager for each object	



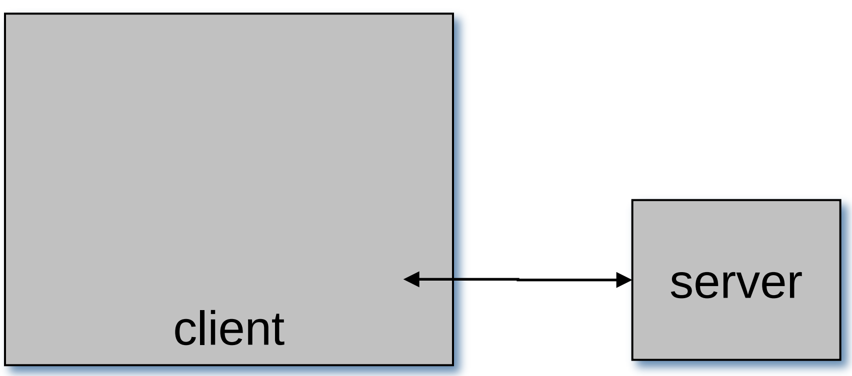
Object-based Server-side Fault Tolerance

CORBA 2.x Server Obligations		
Object Implementation	Initialization	Configuration
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes	1. Registration of IORInterceptor 2. HostMonitor thread instantiation 3. Registration of thread with HostMonitor 4. StateSynchronizationAgent instantiation 5. Registration of State Synchronization Agent with Replication Manager 6. Registration with State Synchronization Agent for each object 7. Registration with Replication Manager for each object	1. ReplicationManager reference 2. HostMonitor reference 3. Replication object id 4. Replica role (Primary/Backup)



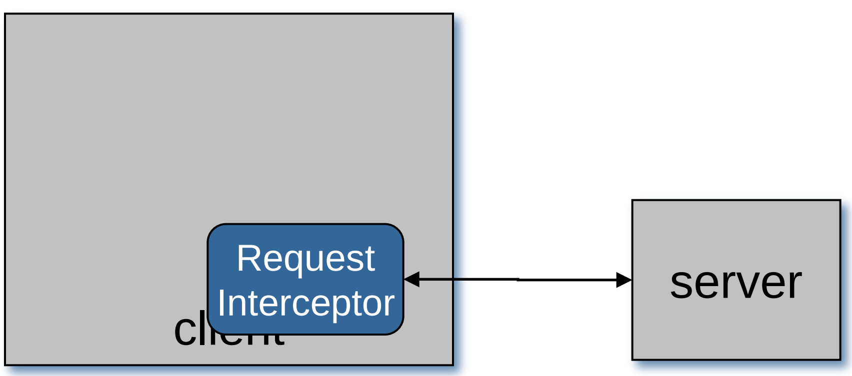
Object-based Client-side Fault Tolerance

CORBA 2.x Client Obligations	



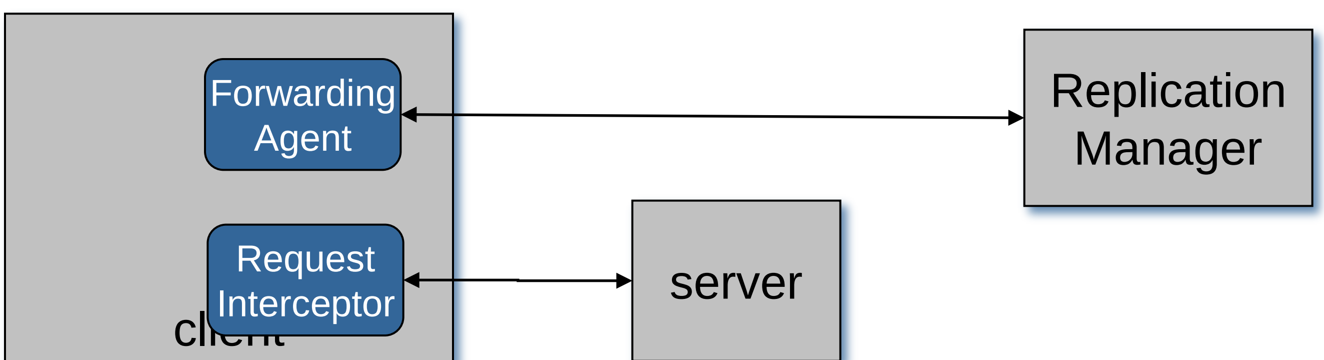
Object-based Client-side Fault Tolerance

CORBA 2.x Client Obligations	
Initialization	
1. Registration of Client Request Interceptor	



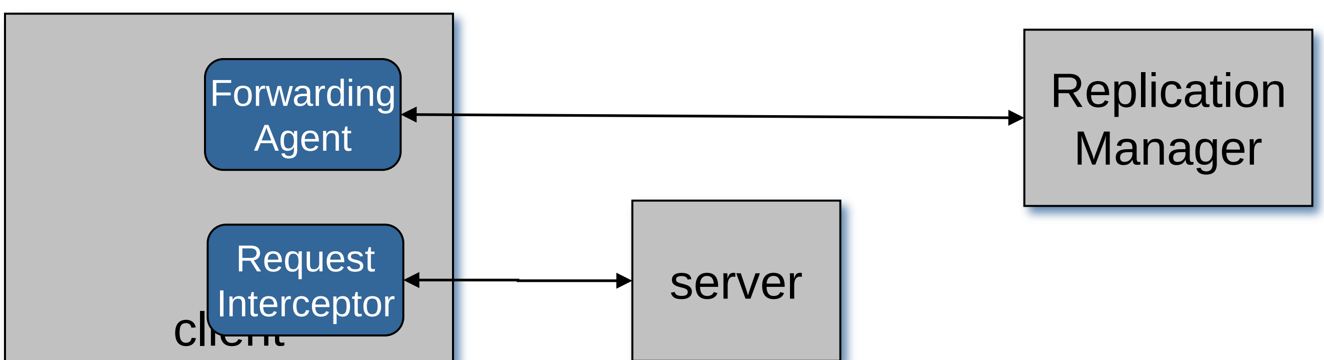
Object-based Client-side Fault Tolerance

CORBA 2.x Client Obligations	
Initialization	
1. Registration of Client Request Interceptor 2. ForwardingAgent instantiation 3. Registration of ForwardingAgent with ReplicationManager	



Object-based Client-side Fault Tolerance

CORBA 2.x Client Obligations	
Initialization	Configuration
1. Registration of Client Request Interceptor 2. ForwardingAgent instantiation 3. Registration of ForwardingAgent with ReplicationManager	1. ReplicationManager reference



Addressing Limitations with Object-based FT

Object-based fault-tolerance incurs additional development effort for

1. Object implementation
2. Initialization and setup of the fault-tolerance infrastructure
3. Configuration of fault-tolerance properties

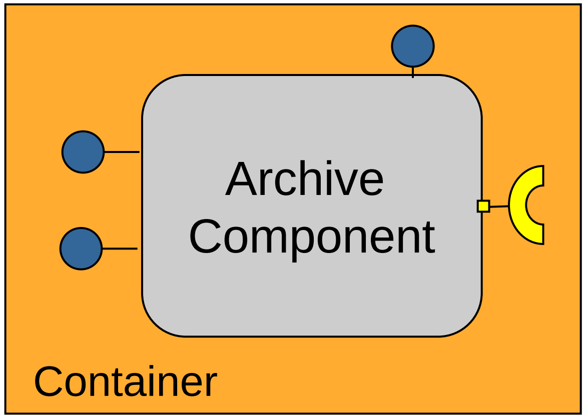
This adds additional sources for accidential errors such as missed intialization steps of wrong order of steps.

CORFU uses component-based infrastructure to reduce this effort

Single Component Replication Context

Component Middleware

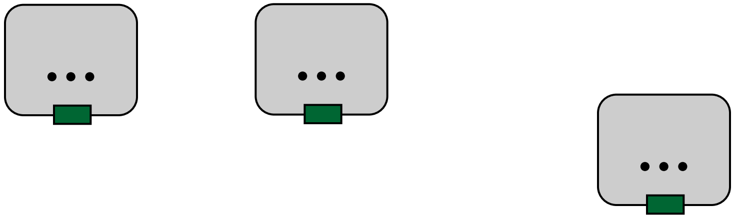
- Creates a standard "virtual boundary" around application **component** implementations that interact only via well-defined interfaces
- Defines standard **container** mechanisms needed to execute components in generic component servers
- Specifies the infrastructure needed to **configure & deploy** components throughout a distributed system



Single Component Replication Challenges

Components cause additional complexities for fault tolerance since they …

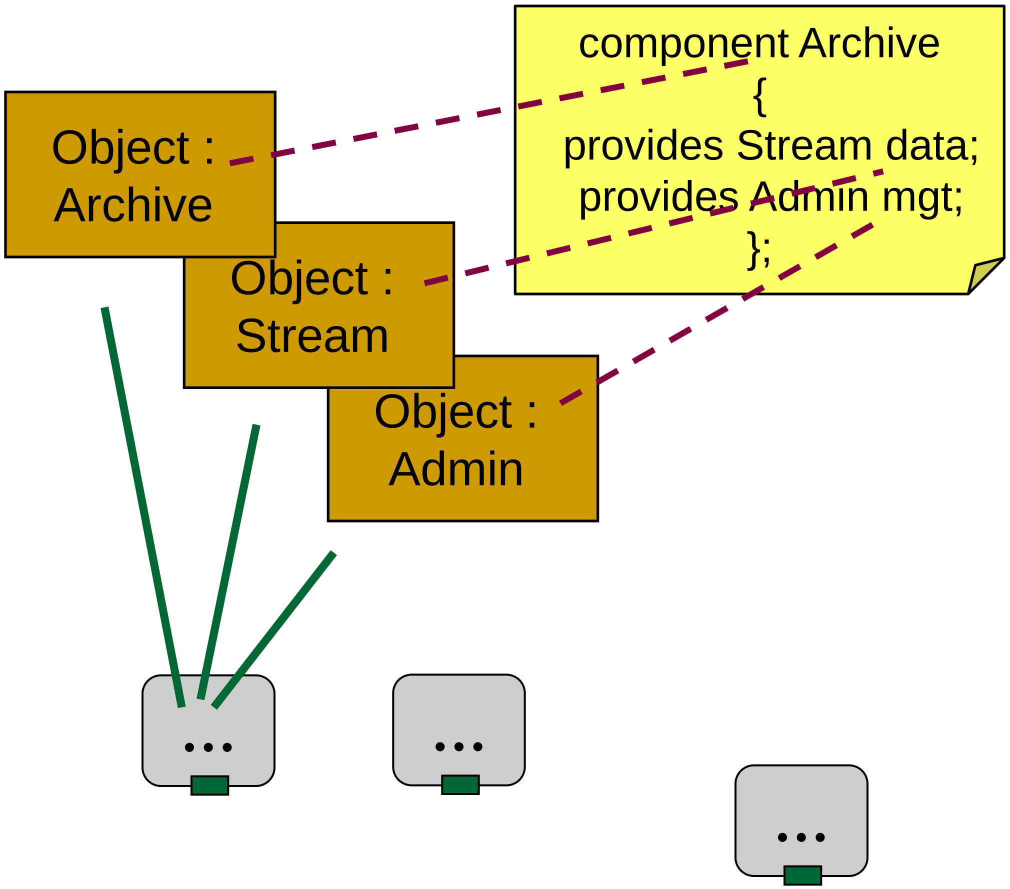
```
component Archive
{
provides Stream data;
provides Admin mgt;
};
```



Single Component Replication Challenges

Components cause additional complexities for fault tolerance since they …

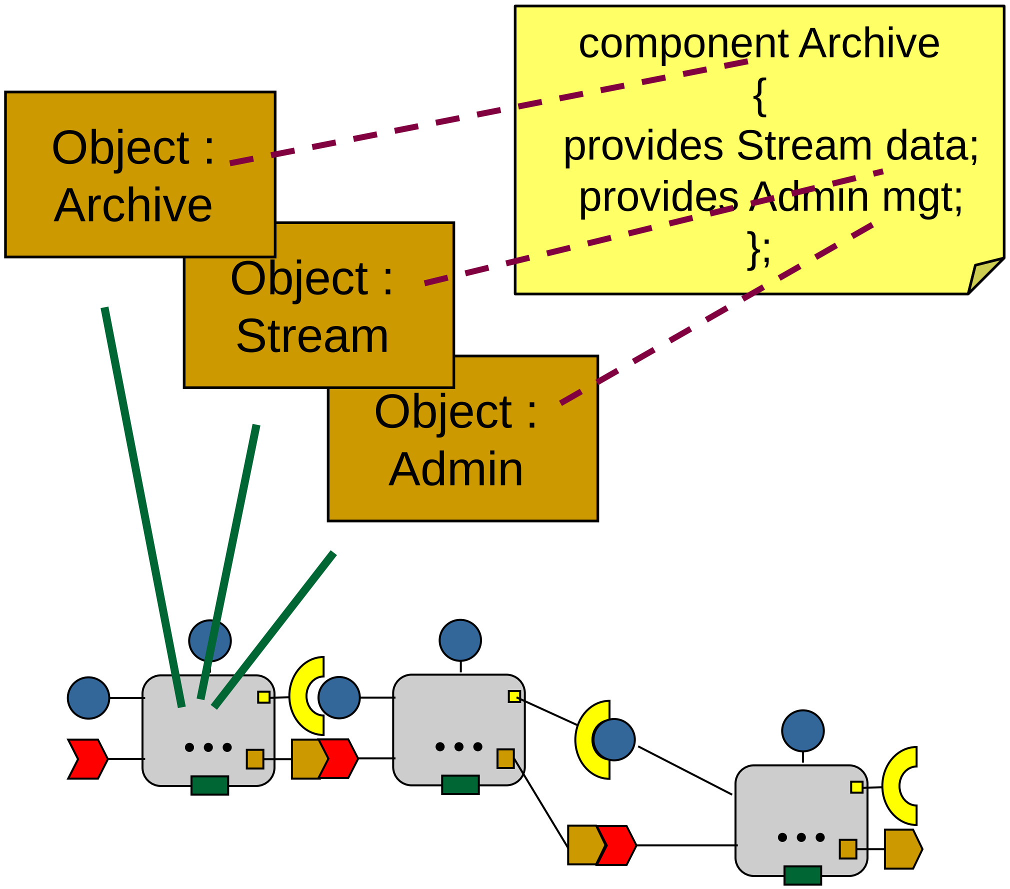
- can consist of several objects



Single Component Replication Challenges

Components cause additional complexities for fault tolerance since they …

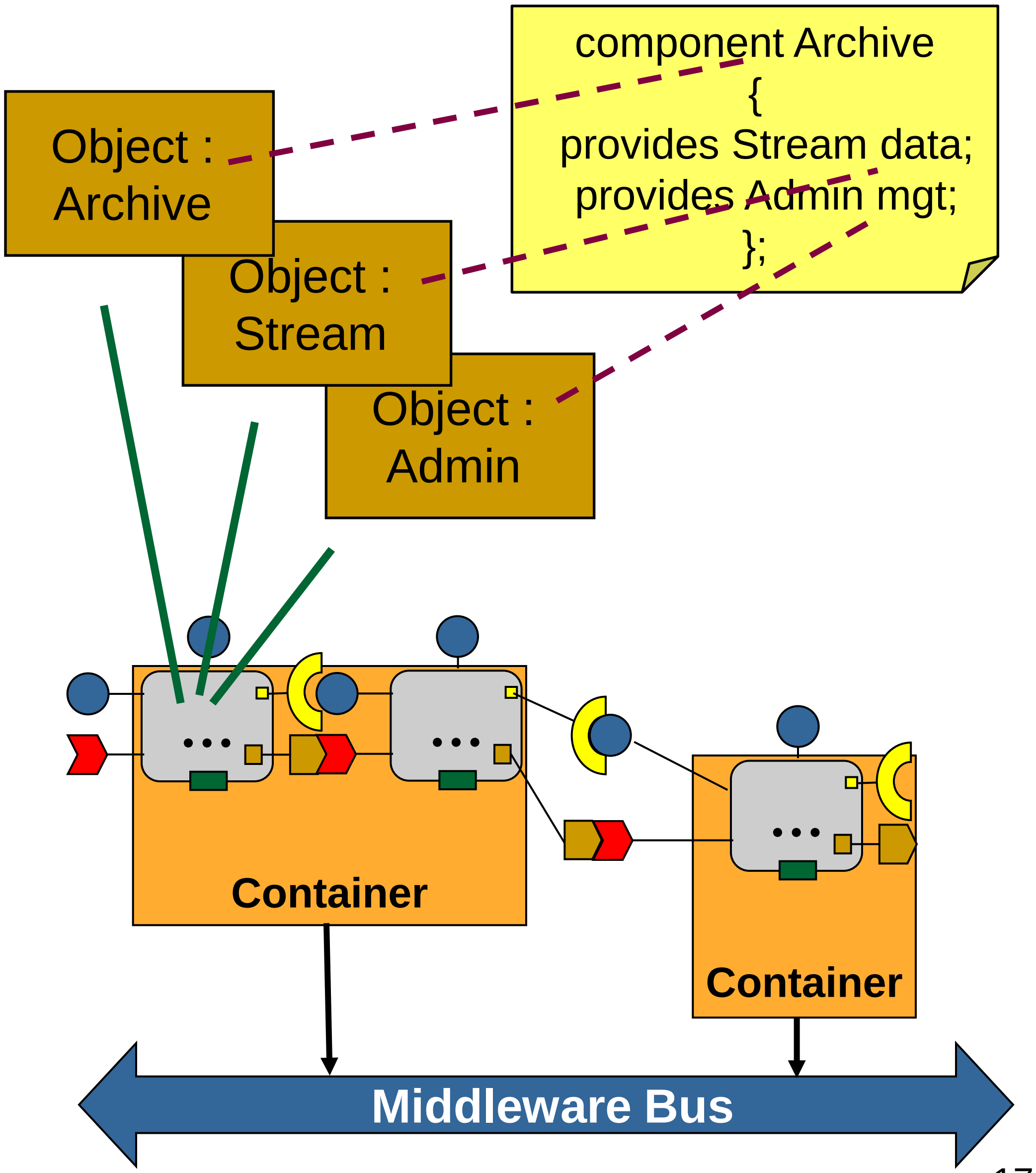
- can consist of several objects
- have connections that need to be maintained



Single Component Replication Challenges

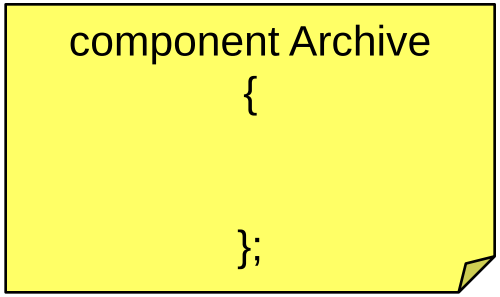
Components cause additional complexities for fault tolerance since they …

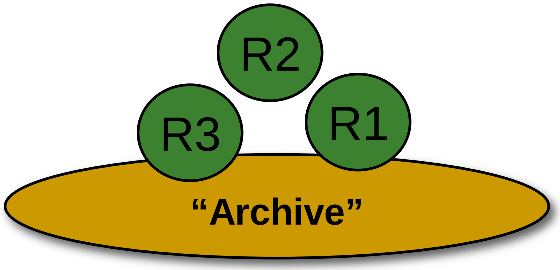
- can consist of several objects
- have connections that need to be maintained
- are shared objects & have no direct control over their run-time infrastructure



Single Component Replication Solutions

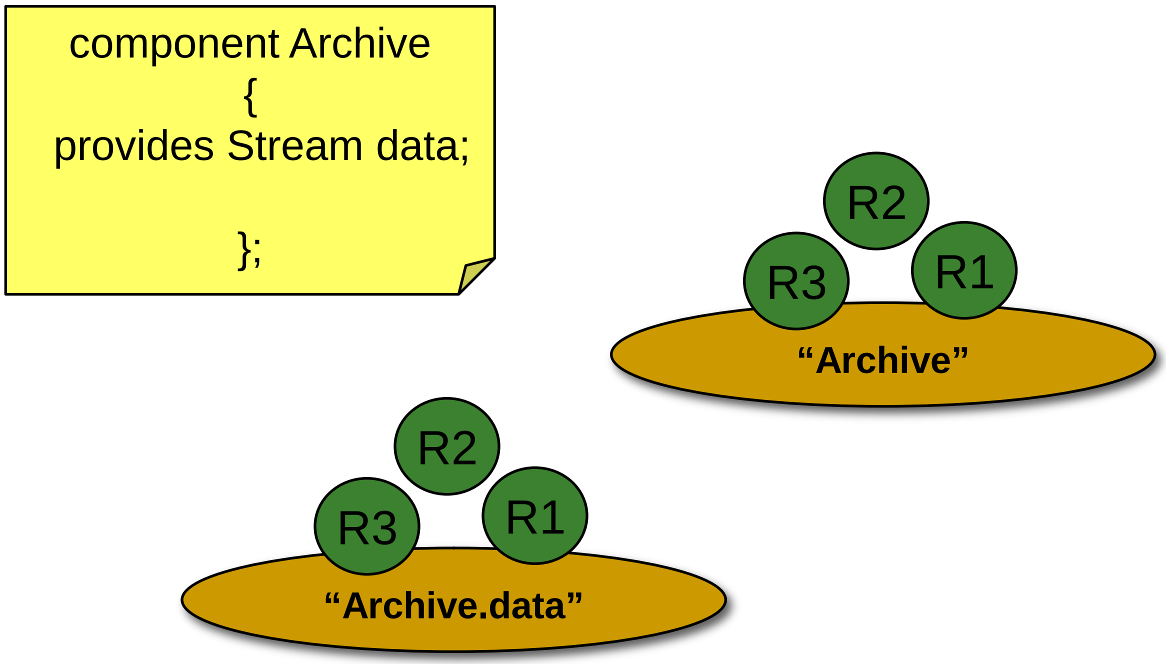
Solution Part 1: Hierarchical naming scheme for grouping objects implementing one component





Single Component Replication Solutions

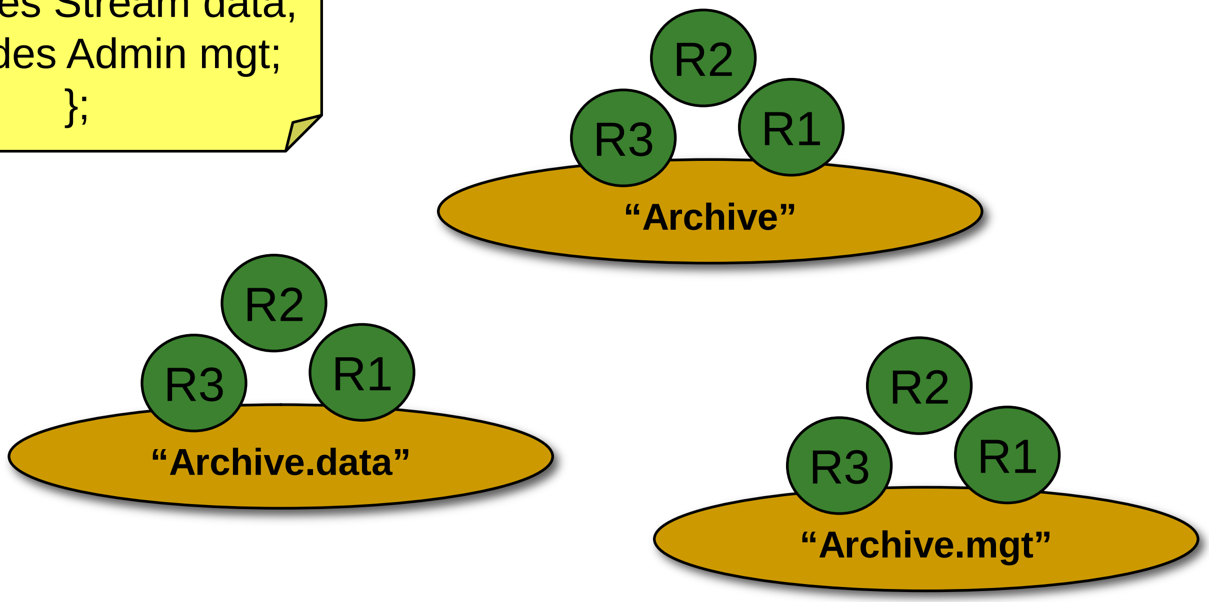
Solution Part 1: Hierarchical naming scheme for grouping objects implementing one component



Single Component Replication Solutions

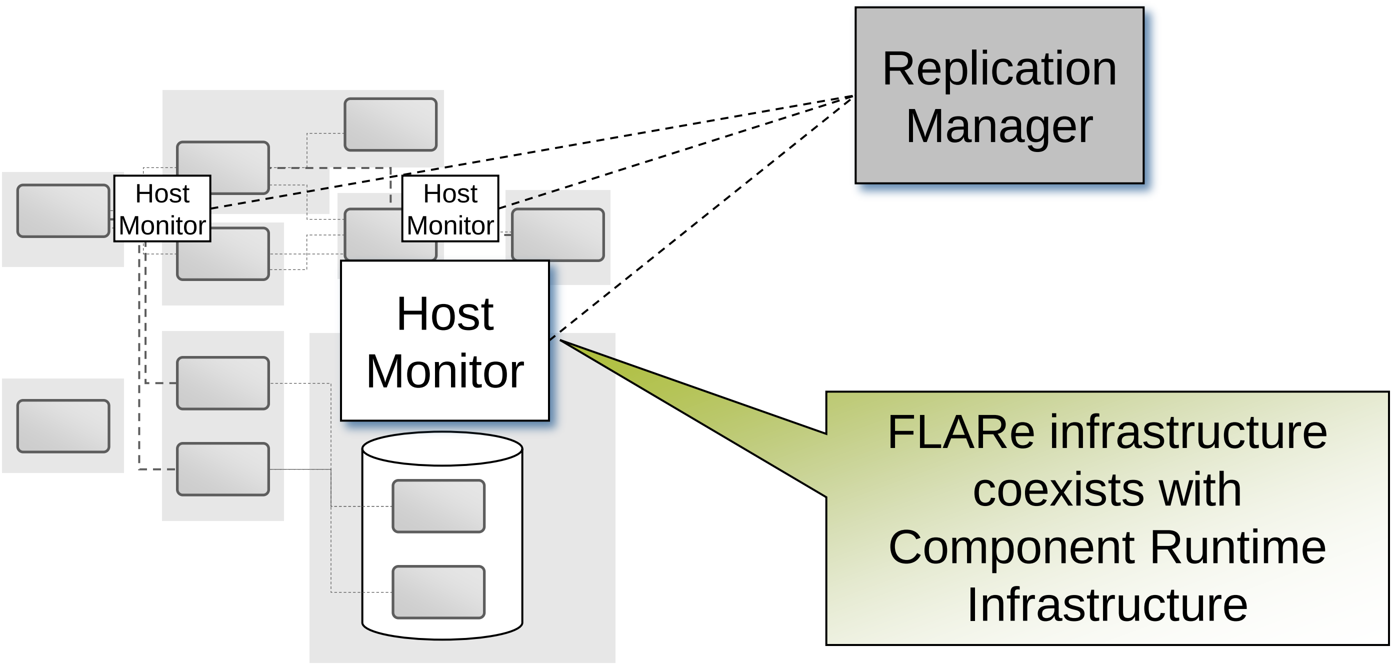
Solution Part 1: Hierarchical naming scheme for grouping objects implementing one component

```
component Archive
{
provides Stream data;
provides Admin mgt;
};
```



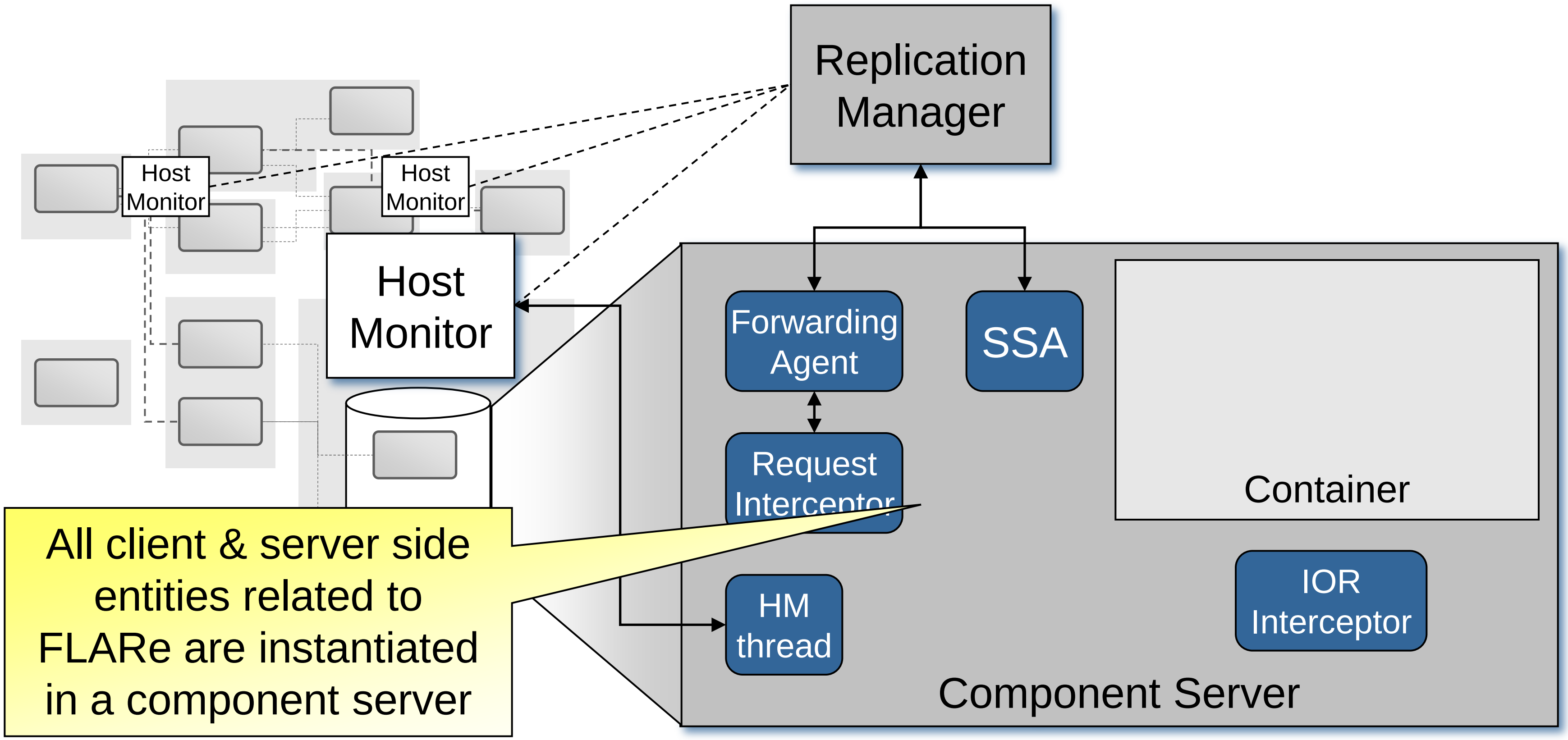
Single Component Replication Solutions

Solution Part 2: Integration of FLARE into a fault tolerant component server



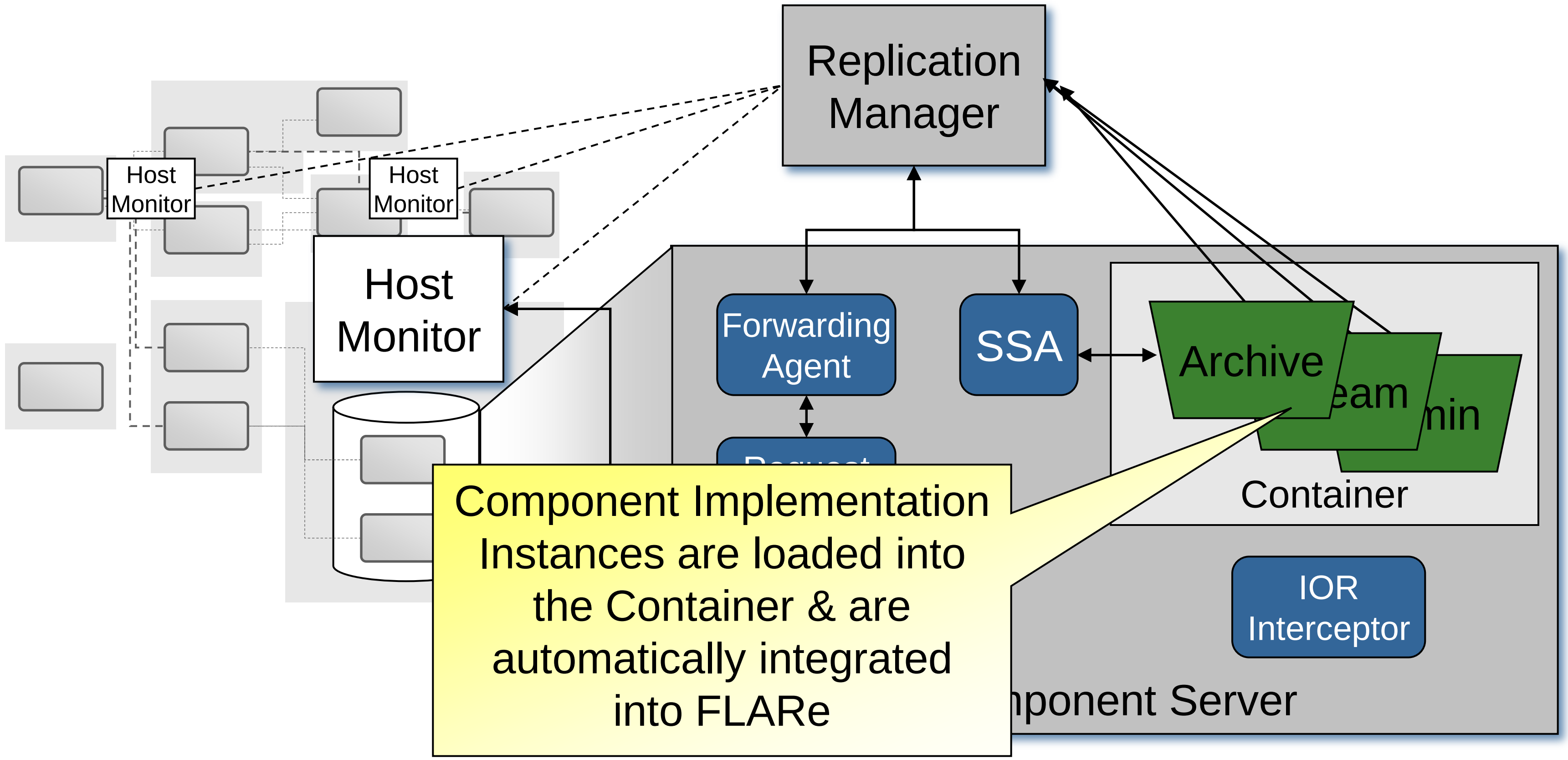
Single Component Replication Solutions

Solution Part 2: Integration of FLARE into a fault tolerant component server



Single Component Replication Solutions

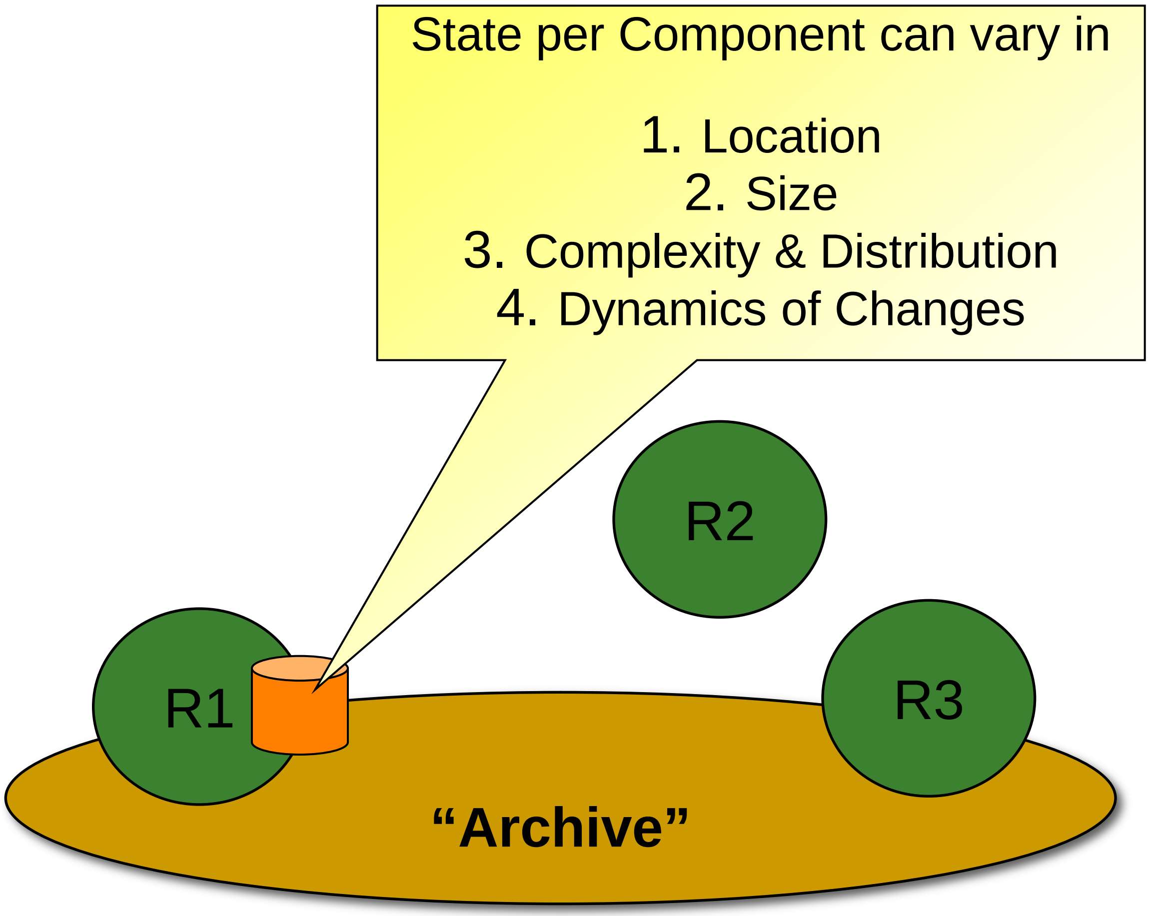
Solution Part 2: Integration of FLARE into a fault tolerant component server



Component State Synchronization w/CHESS

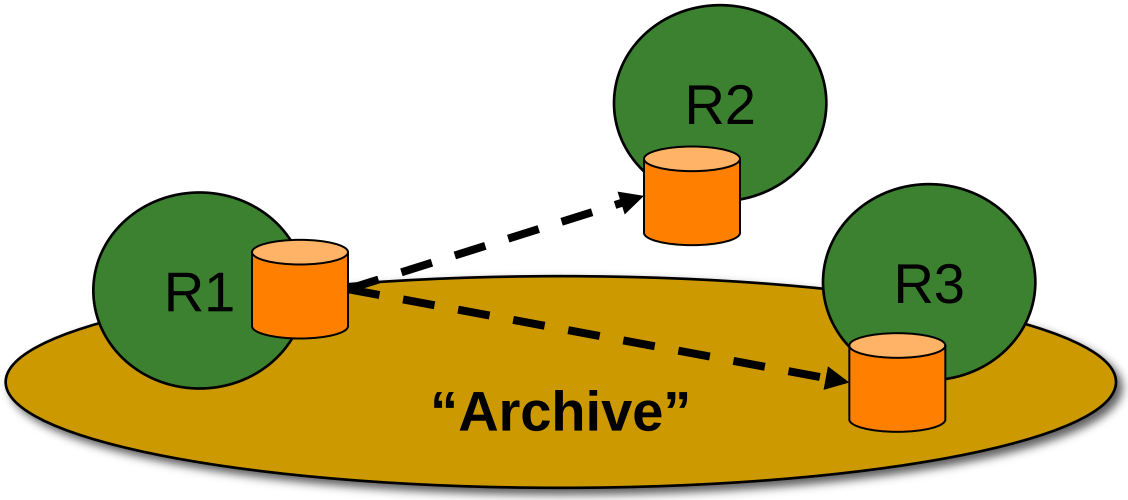
Components maintain internal state that needs to be propagated to backup replicas

CHESS = "Components with HEterogeneous State Synchronization"



Component State Synchronization w/CHESS

Components maintain internal state that needs to be propagated to backup replicas

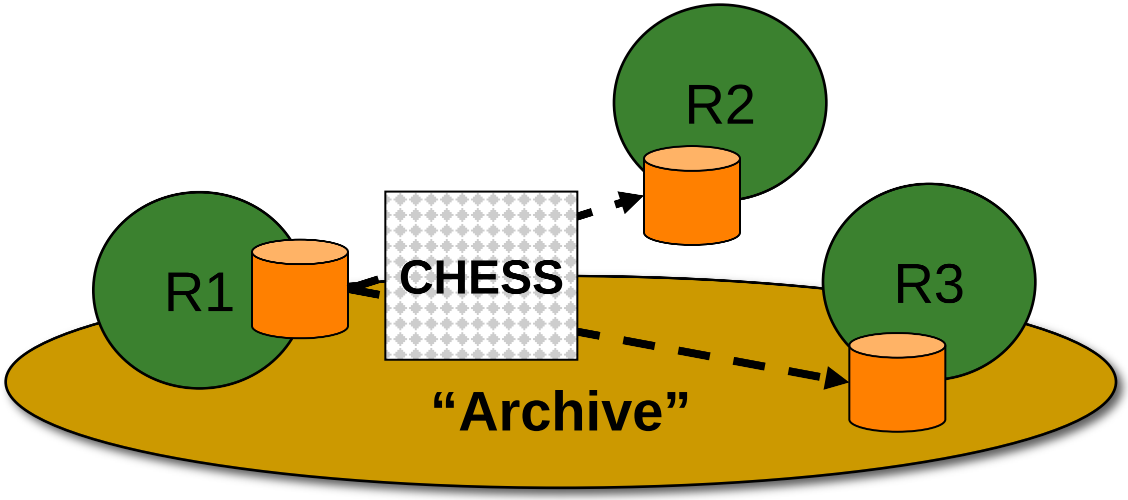


Component State Synchronization w/CHESS

Components maintain internal state that needs to be propagated to backup replicas

The CHESS Framework applies the Strategy pattern to allow

1. Registration of component instances in the local process space

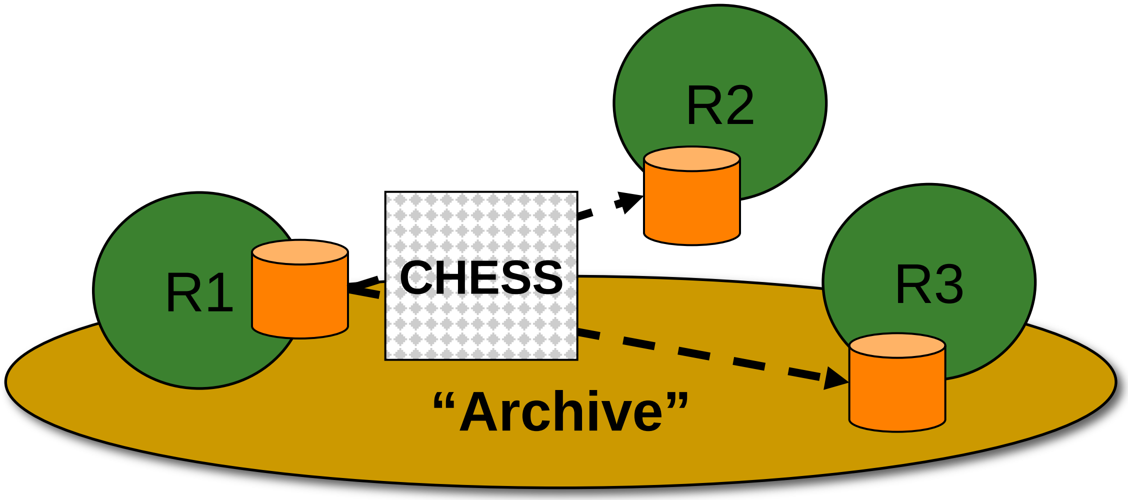


Component State Synchronization w/CHESS

Components maintain internal state that needs to be propagated to backup replicas

The CHESS Framework applies the Strategy pattern to allow

1. Registration of component instances in the local process space
2. Choice of the transport protocol for state dissemination (e.g. CORBA or DDS)

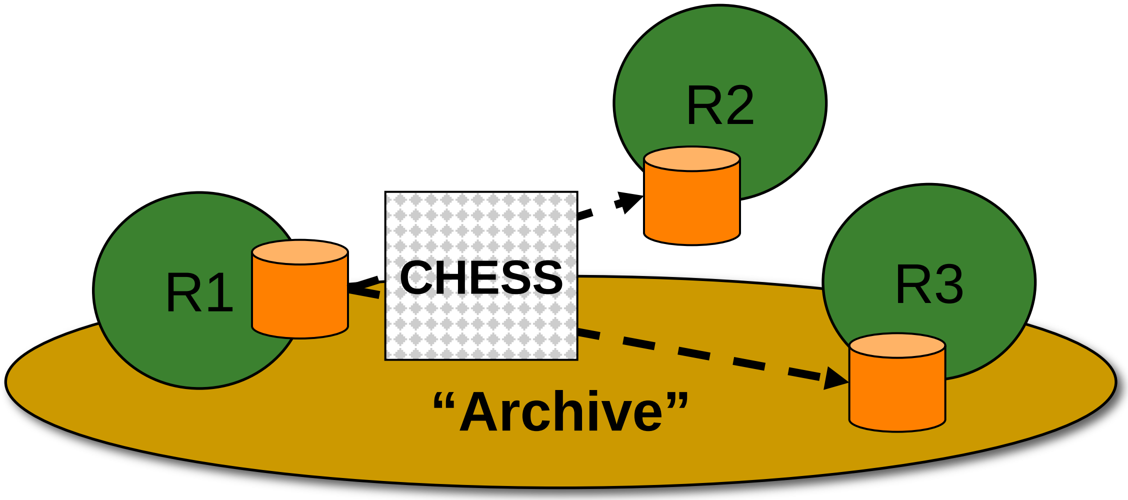


Component State Synchronization w/CHESS

Components maintain internal state that needs to be propagated to backup replicas

The CHESS Framework applies the Strategy pattern to allow

1. Registration of component instances in the local process space
2. Choice of the transport protocol for state dissemination (e.g. CORBA or DDS)
3. Connection management for communication with other components



Component State Synchronization w/CHESS

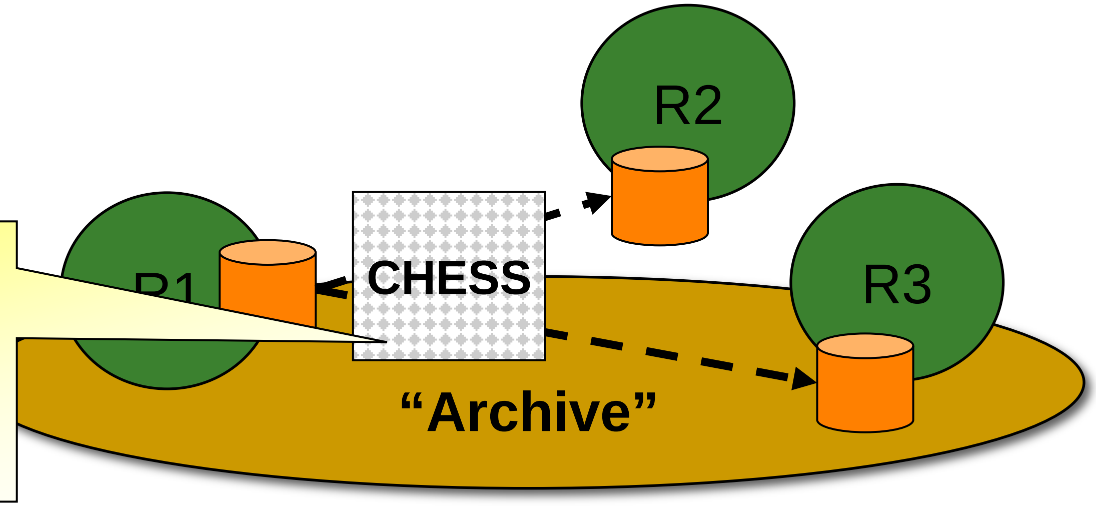
Components maintain internal state that needs to be propagated to backup replicas

The CHESS Framework applies the Strategy pattern to allow

1. Registration of component instances in the local process space
2. Choice of the transport protocol for state dissemination (e.g. CORBA or DDS)
3. Connection management for communication with other components
4. State Dissemination

CHESS gives flexibility in

1. Serialization of State
2. Timing Behavior
3. Protocol Choice



Benefits of CORFU FT vs. Object-based FT

CORFU integrates Fault Tolerance mechanisms into component-based systems

- Server & client side functionality is both integrated into one container

CCM Component Obligations		
Object Implementation	Initialization	Configuration
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls 3. Getter & setter methods for object id & state synchronization agent attributes	1. Registration of IORInterceptor 2. HostMonitor thread instantiation 3. Registration of thread with HostMonitor 4. StateSynchronizationAgent instantiation 5. Registration of State Synchronization Agent with Replication Manager 6. Registration with State Synchronization Agent for each object 7. Registration with Replication Manager for each object	1. ReplicationManager reference 2. HostMonitor reference 3. Replication object id 4. Replica role (Primary/Backup)

Benefits of CORFU FT vs. Object-based FT

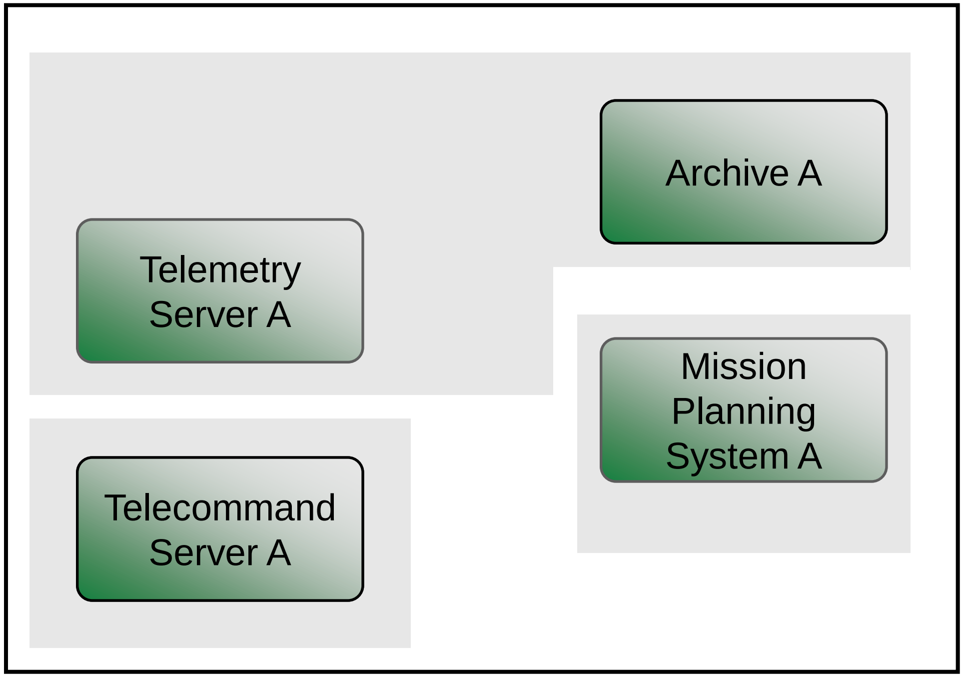
CORFU integrates Fault Tolerance mechanisms into component-based systems

- Server & client side functionality is both integrated into one container
- Fault tolerance related tasks are automated

CCM Component Obligations		
Object Implementation	Initialization	Configuration
1. Implementation of get_state/set_state methods 2. Triggering state synchronization through state_changed calls ✓ Partly automated through code generation	✓ Initialization is done automatically within the component server & container	✓ Configuration of components is done in the deployment plan through configProperties

Component Group Replication Context

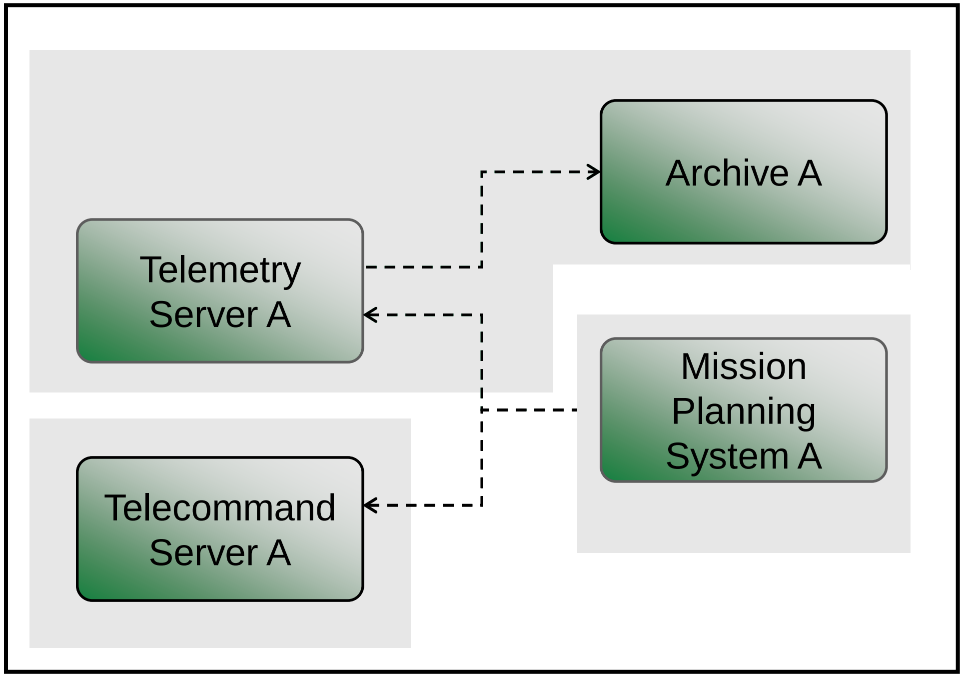
Assemblies of Components with Fault dependencies



Component Group Replication Context

Assemblies of Components with Fault dependencies

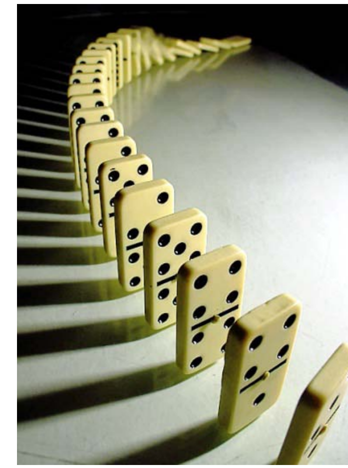
- Component Assemblies are characterized by a high degree of interactions



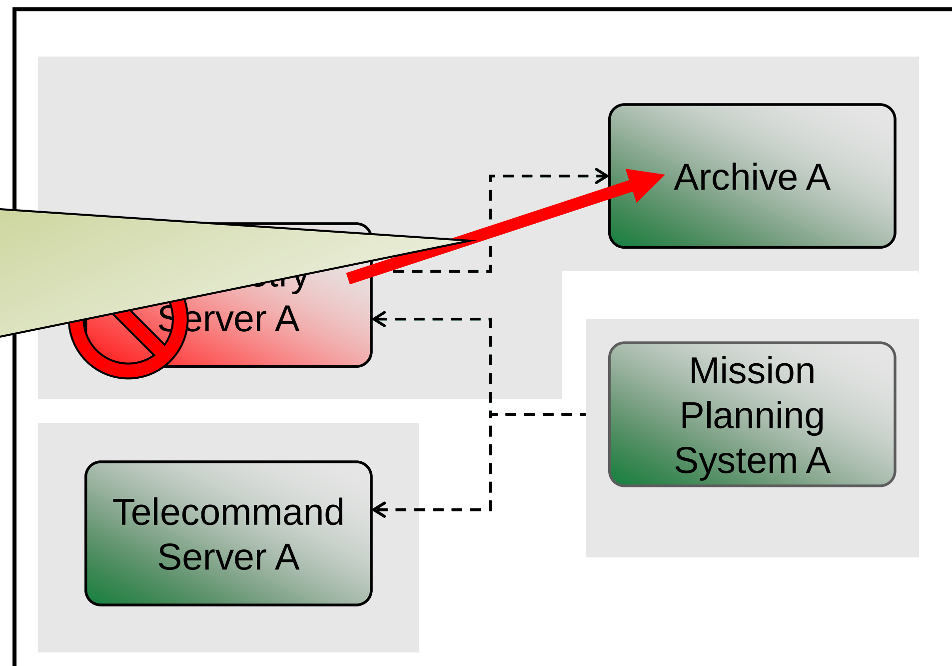
Component Group Replication Context

Assemblies of Components with Fault dependencies

- Component Assemblies are characterized by a high degree of interactions
- Failures of one component can affect other components



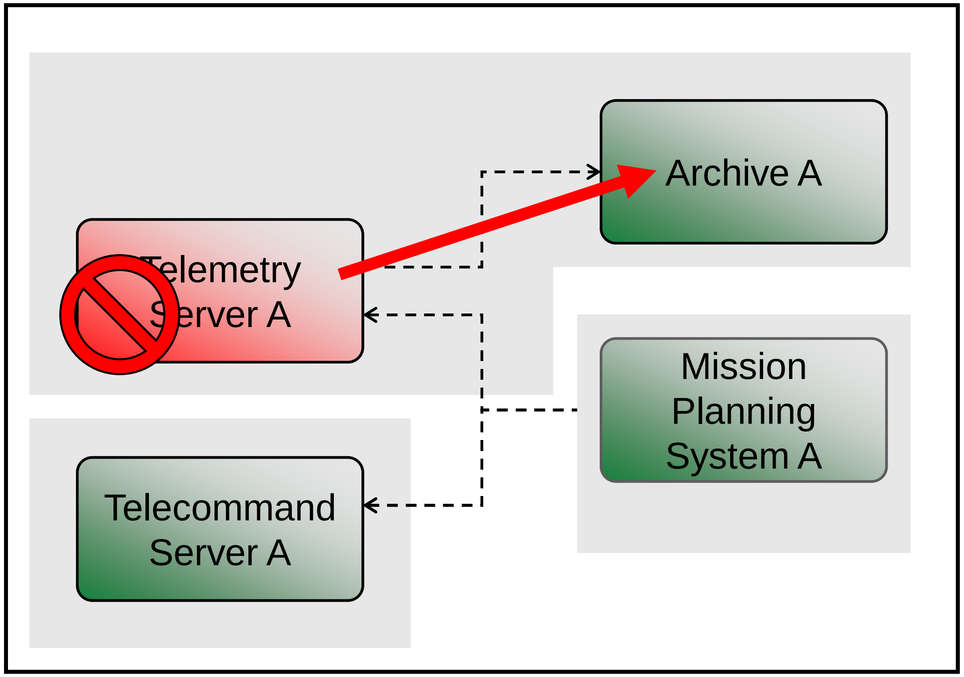
Faults can propagate across components through
1. Shared Hardware Infrastructure
2. Shared Networking Infrastructure
3. Shared Middleware Services
4. Component Port Connections



Component Group Replication Context

Assemblies of Components with Fault dependencies

- Component Assemblies are characterized by a high degree of interactions
- Failures of one component can affect other components
- Detecting errors early on allows to take correcting means & isolate the fault effects



Component Group Replication Related Work

Approach	Solution	Reference
Static Dependency Modeling	Cadena Dependency Model	John Hatcliff, Xinghua Deng, Matthew B. Dwyer, Georg Jung, & Venkatesh Prasad Ranganath. "Cadena: An integrated development, analysis, & verification environment for component-based systems." *International Conference on Software Engineering*, pages 0 - 160, 2003.
	Component Based Dependency Modeling (CBDM)	M. Vieira & D. Richardson. "Analyzing dependencies in large component-based systems." *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, pages 241–244, 2002.
	Event Correlation	Boris Gruschke. "A new approach for event correlation based on dependency graphs." In *In 5th Workshop of the OpenView University Association*, 1998.

White Box approach where dependencies are defined declaratively

Component Group Replication Related Work

Approach	Solution	Reference
Static Dependency Modeling	Cadena Dependency Model	John Hatcliff, Xinghua Deng, Matthew B. Dwyer, Georg Jung, & Venkatesh Prasad Ranganath. “Cadena: An integrated … & verification environment for component- … *tional Conference on Software* … 60, 2003.
	Compone… Depende… Modeling	…on. “Analyzing dependencies in large …ms.” *Automated Software Engineering, … 2002. 17th IEEE International …onference on*, pages 241–244, 2002.
	Event Correl…	Boris Gruschke. “A new approach for event correlation based on dependency graphs.” In *In 5th Workshop of the OpenView University Association*, 1998.
Observation based Dependency Modeling	Acti… Dependecy Di…overy (ADD)	A. Brown, G. Kar, A. Keller, “An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Application Environment,” IEEE/IFIP International Symposium on Integrated Network Management, pp. 377-390, 2001.
	Automatic Failure Path Inference (AFPI)	George Candea, Mauricio Delgado, Michael Chen, & Armando Fox. “Automatic failure-path inference: A generic introspection technique for internet applications.” In *WIAPP ’03: Proceedings of the The Third IEEE Workshop on Internet Applications*, page 132,Washington, DC, USA, 2003.

Black Box approach where dependencies are detected through fault injection & monitoring

CORFU Requirements

Fault Tolerance dependency information is used to group components according to their dependencies

CORFU Requirements

Fault Tolerance dependency information is used to group components according to their dependencies

CORFU is a middleware solution that provides fault tolerance capabilities based on such dependency groups

CORFU Requirements

Fault Tolerance dependency information is used to group components according to their dependencies

CORFU is a middleware solution that provides fault tolerance capabilities based on such dependency groups

Requirements that have to be met are:

1. Fault Isolation

CORFU Requirements

Fault Tolerance dependency information is used to group components according to their dependencies

CORFU is a middleware solution that provides fault tolerance capabilities based on such dependency groups

Requirements that have to be met are:

1. Fault Isolation
2. Fail-Stop Behavior

CORFU Requirements

Fault Tolerance dependency information is used to group components according to their dependencies

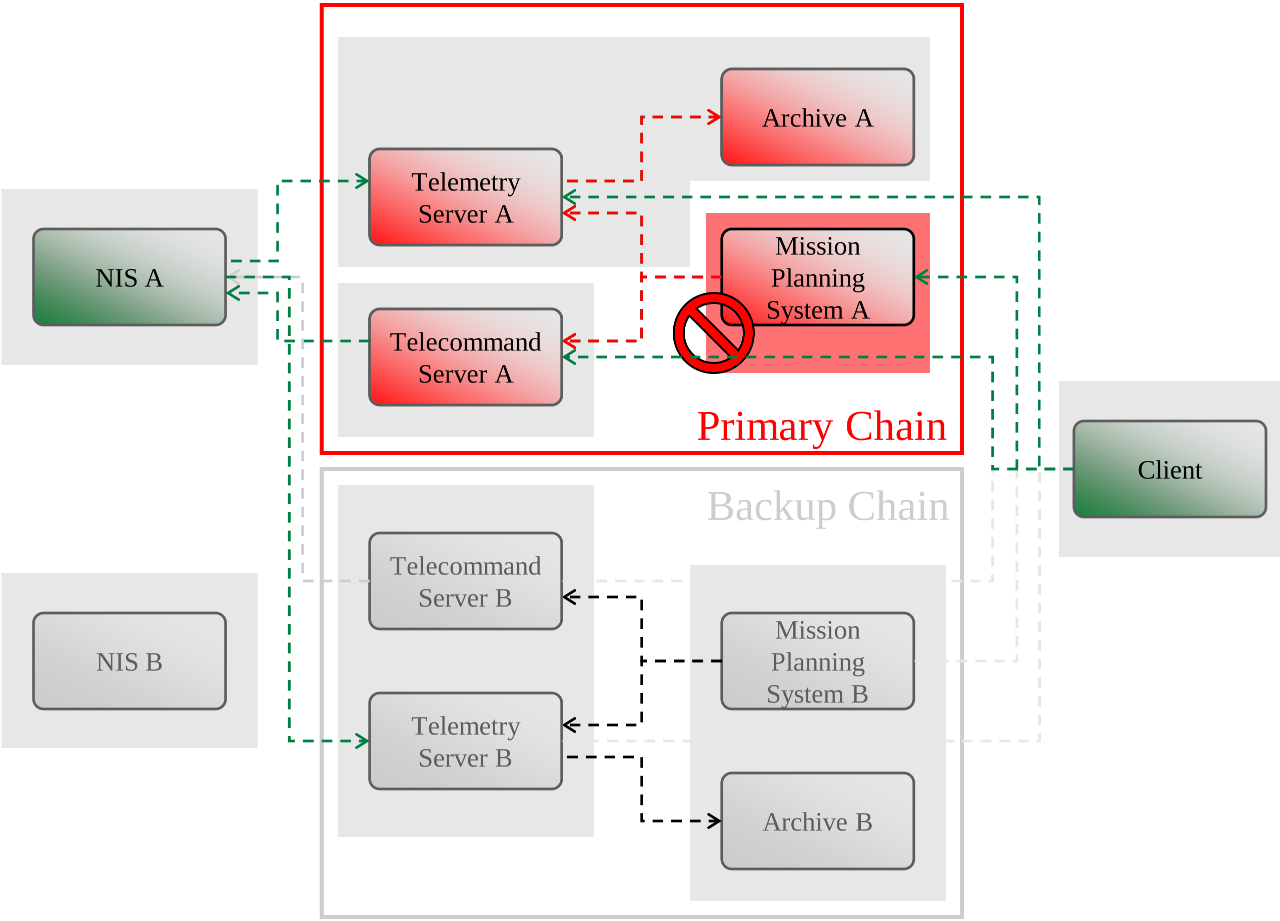
CORFU is a middleware solution that provides fault tolerance capabilities based on such dependency groups

Requirements that have to be met are:

1. Fault Isolation
2. Fail-Stop Behavior
3. Server Recovery

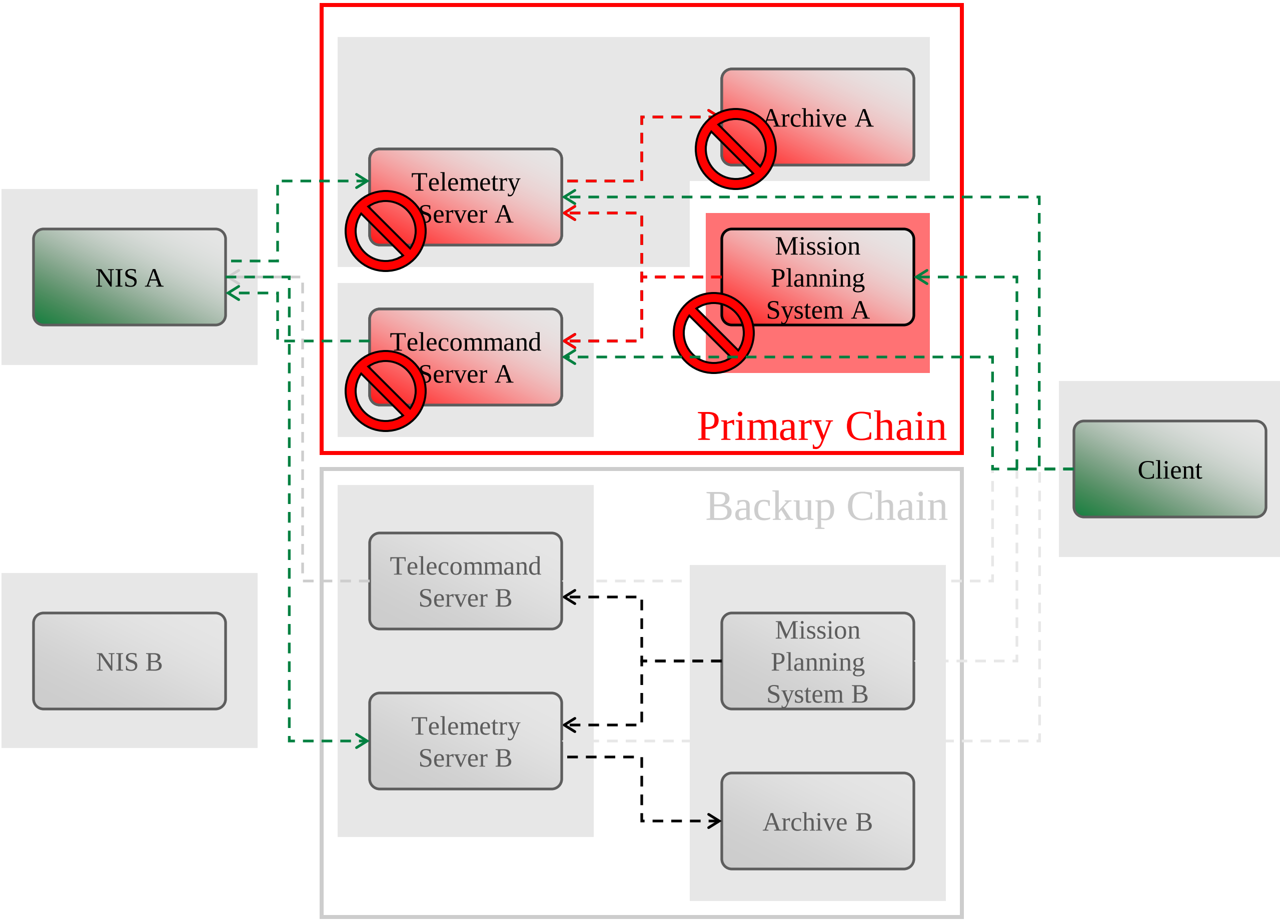
Requirement 1: Fault Isolation

- Occurrence of Server or Process faults
- Such faults need to be detected
- To isolate the fault all affected components need to be identified



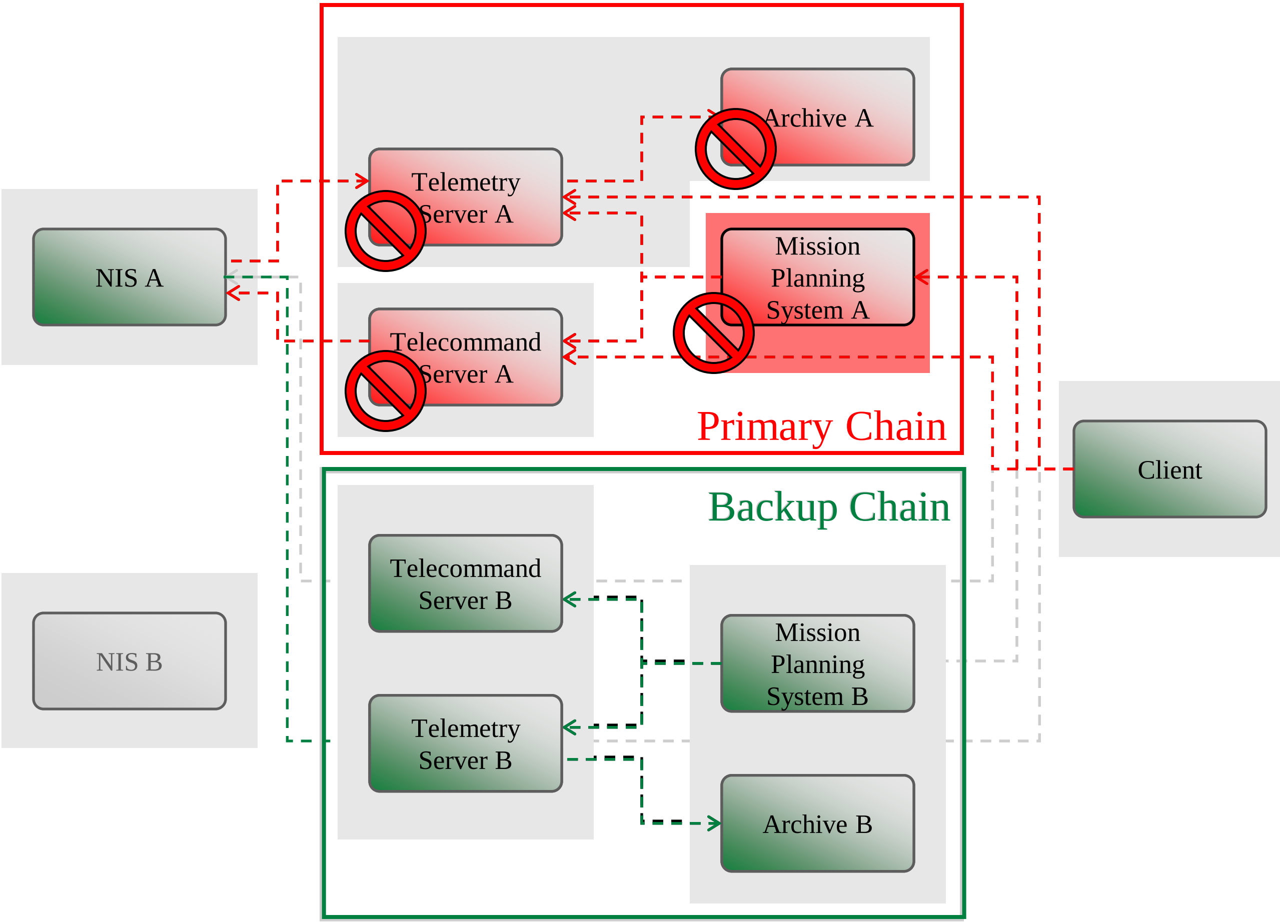
Requirement 2: Fail-Stop Behavior

- All affected components need to be stopped to prevent inconsistent system state
- This has to happen as synchronously as possible in a distributed system and
- As close to the detection of the failure as possible



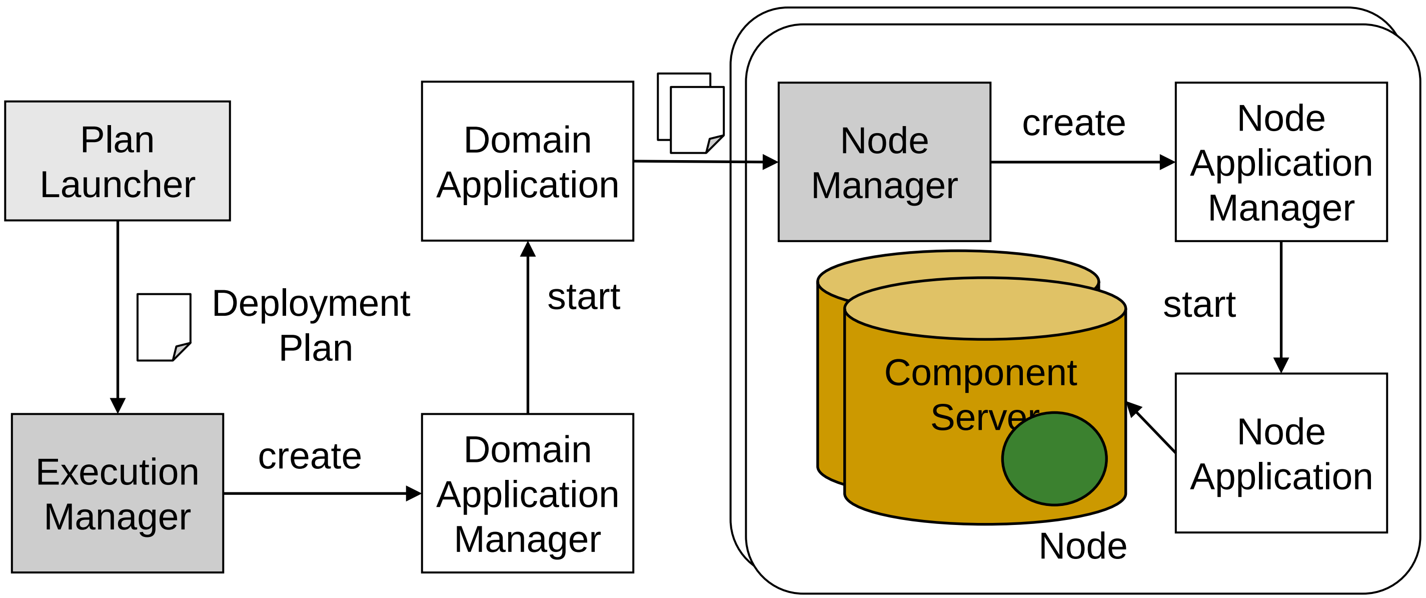
Requirement 3: Server Recovery

- Component failover mechanisms operate on a per component basis
- Failover needs to be coordinated for all failed components
- The right backup replica needs to be activated for each component to ensure consistent system state after failover



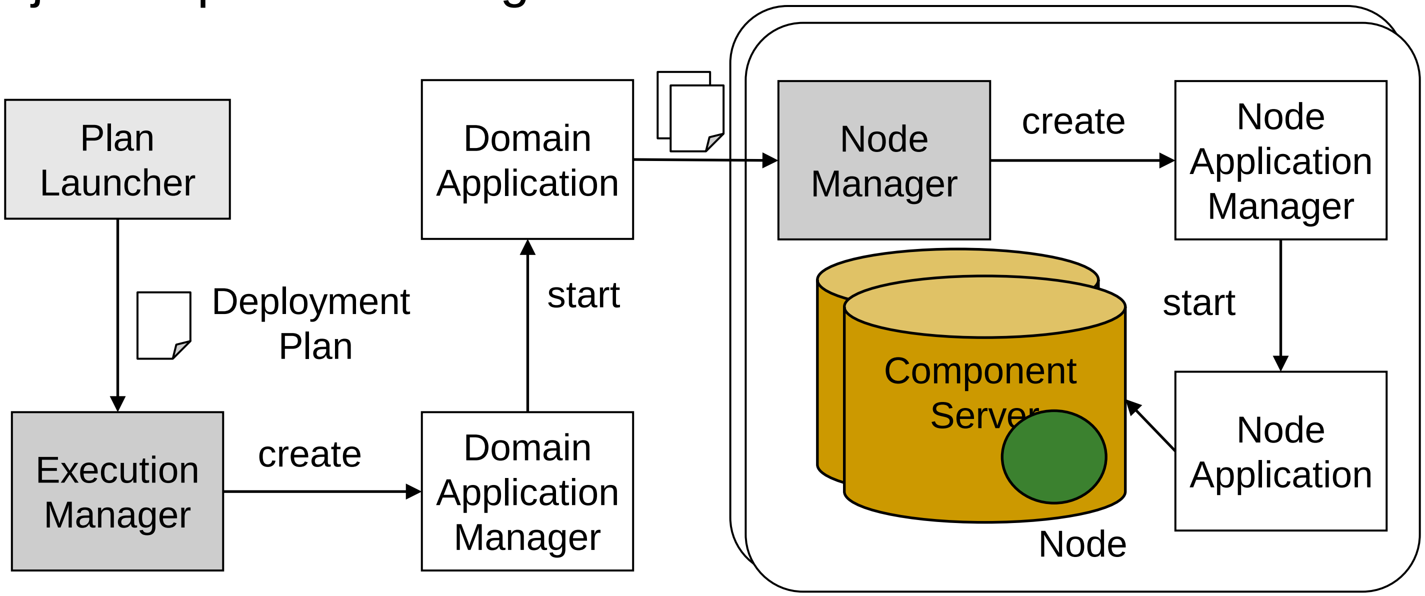
Component Group Fault Tolerance Challenges

- Standard Interfaces do not provide FT capabilities & cannot be altered
  - Additional Functionality needs to be standard compatible
- Interaction with DAnCE services is necessary to access system structure without reducing component performance significantly



Component Group Fault Tolerance Challenges

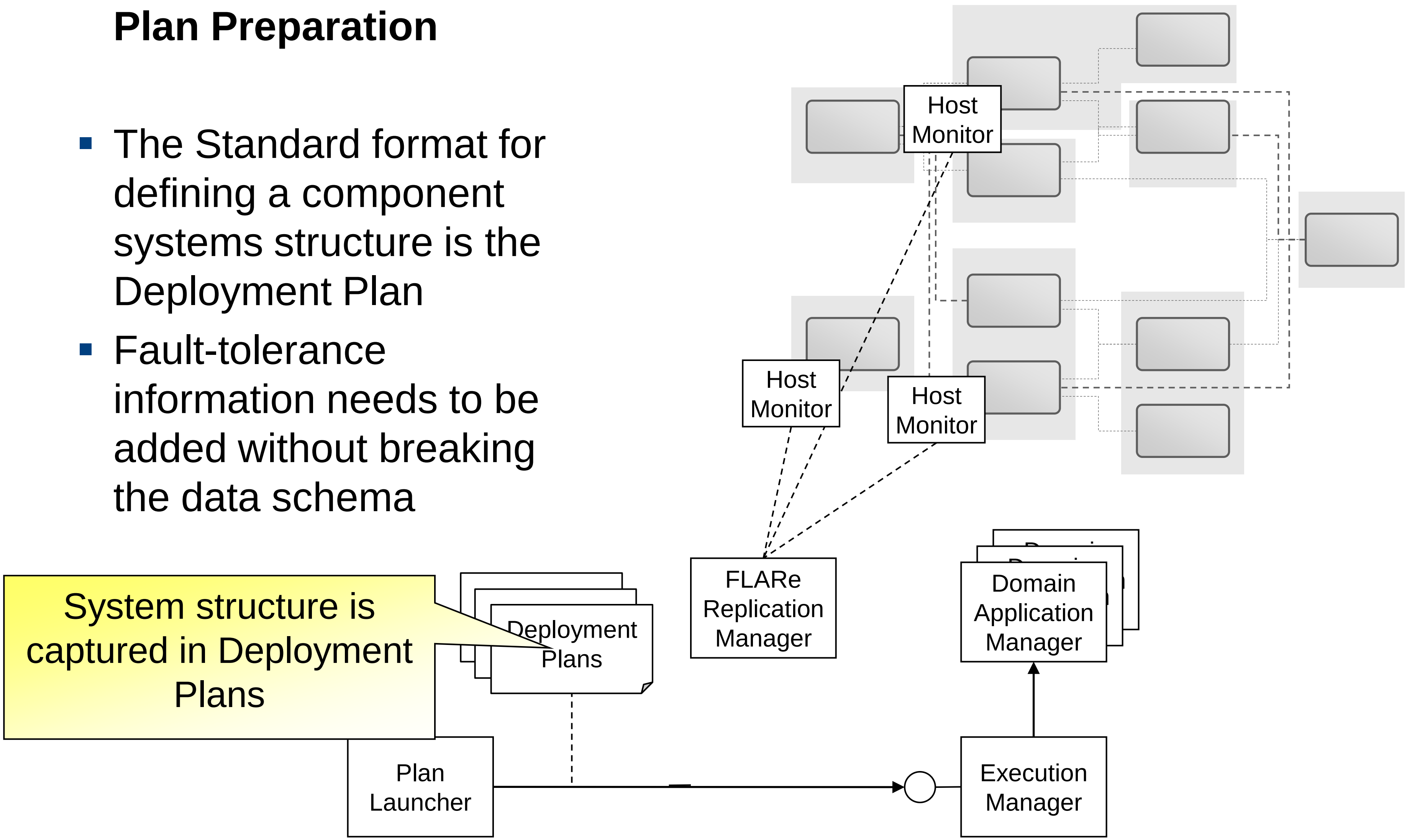
- Standard Interfaces do not provide FT capabilities & cannot be altered
- Additional Functionality needs to be standard compatible
- Interaction with DAnCE services is necessary to access system structure without reducing component performance significantly
- This includes
  1. Deployment Plan Preparation
  2. Integration of Failover Functionality
  3. Object Replica Ordering



Deployment Plan Preparation Solution

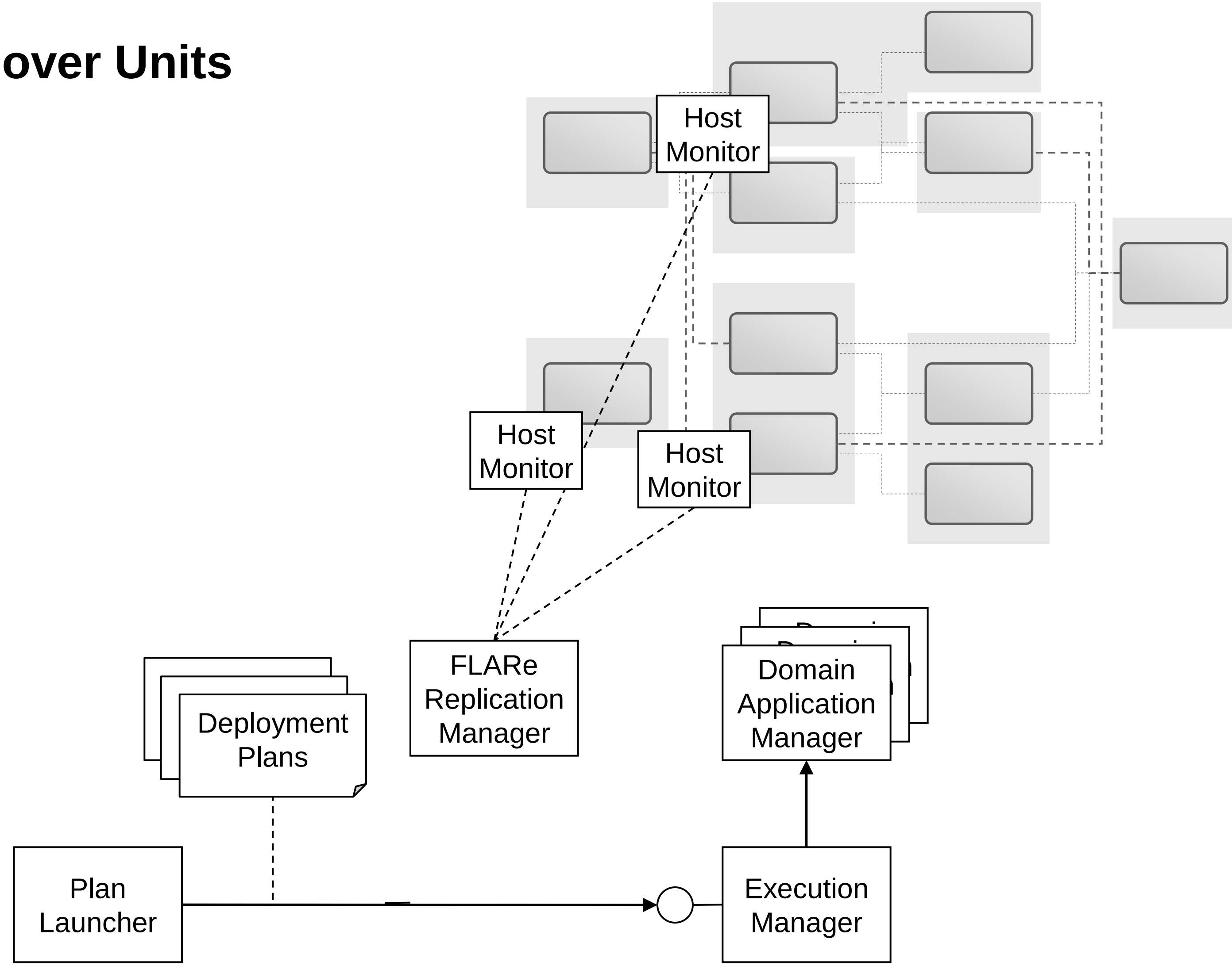
Challenge 1: Deployment Plan Preparation

- The Standard format for defining a component systems structure is the Deployment Plan
- Fault-tolerance information needs to be added without breaking the data schema



Deployment Plan Preparation Solution

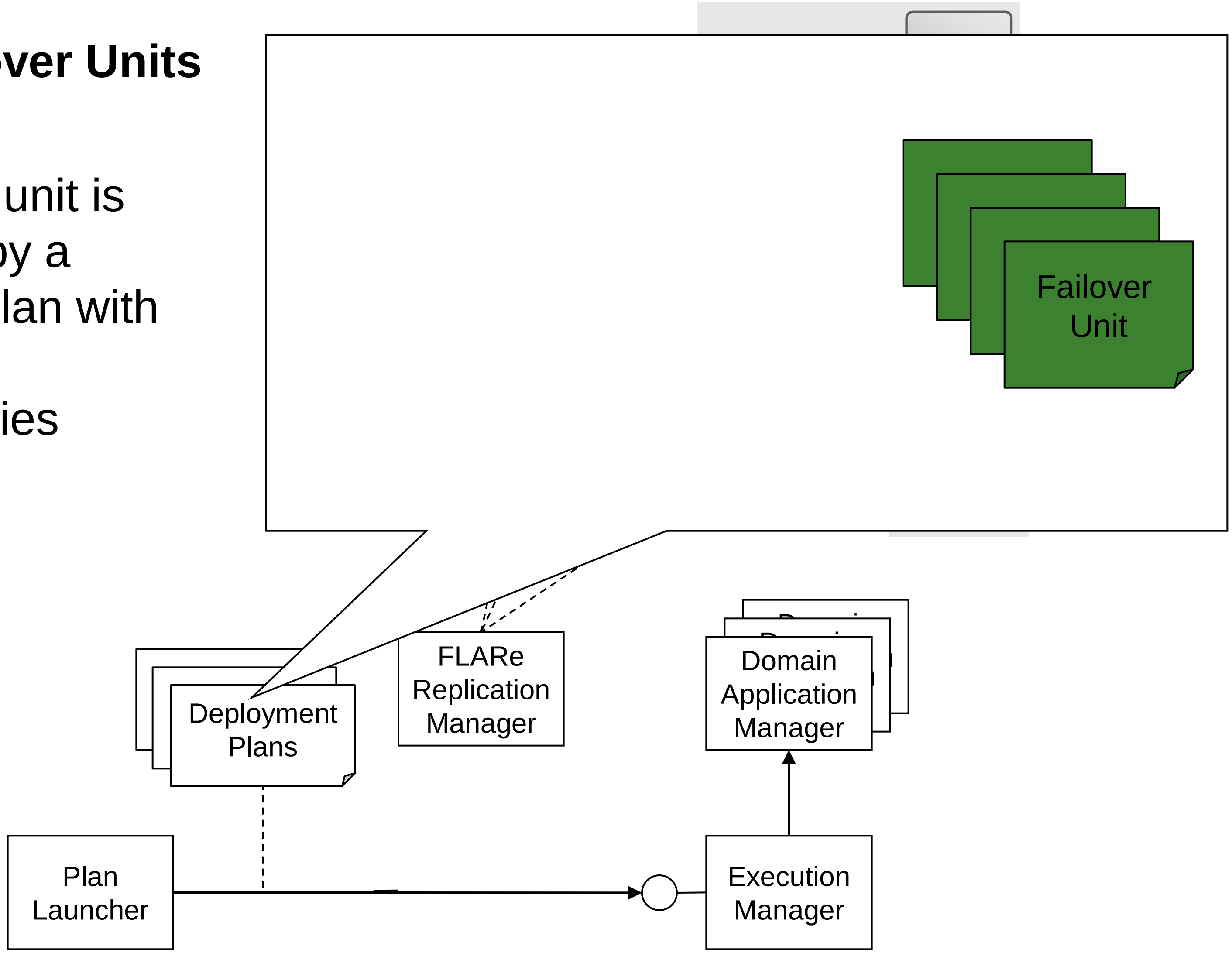
Solution: Failover Units



Deployment Plan Preparation Solution

Solution: Failover Units

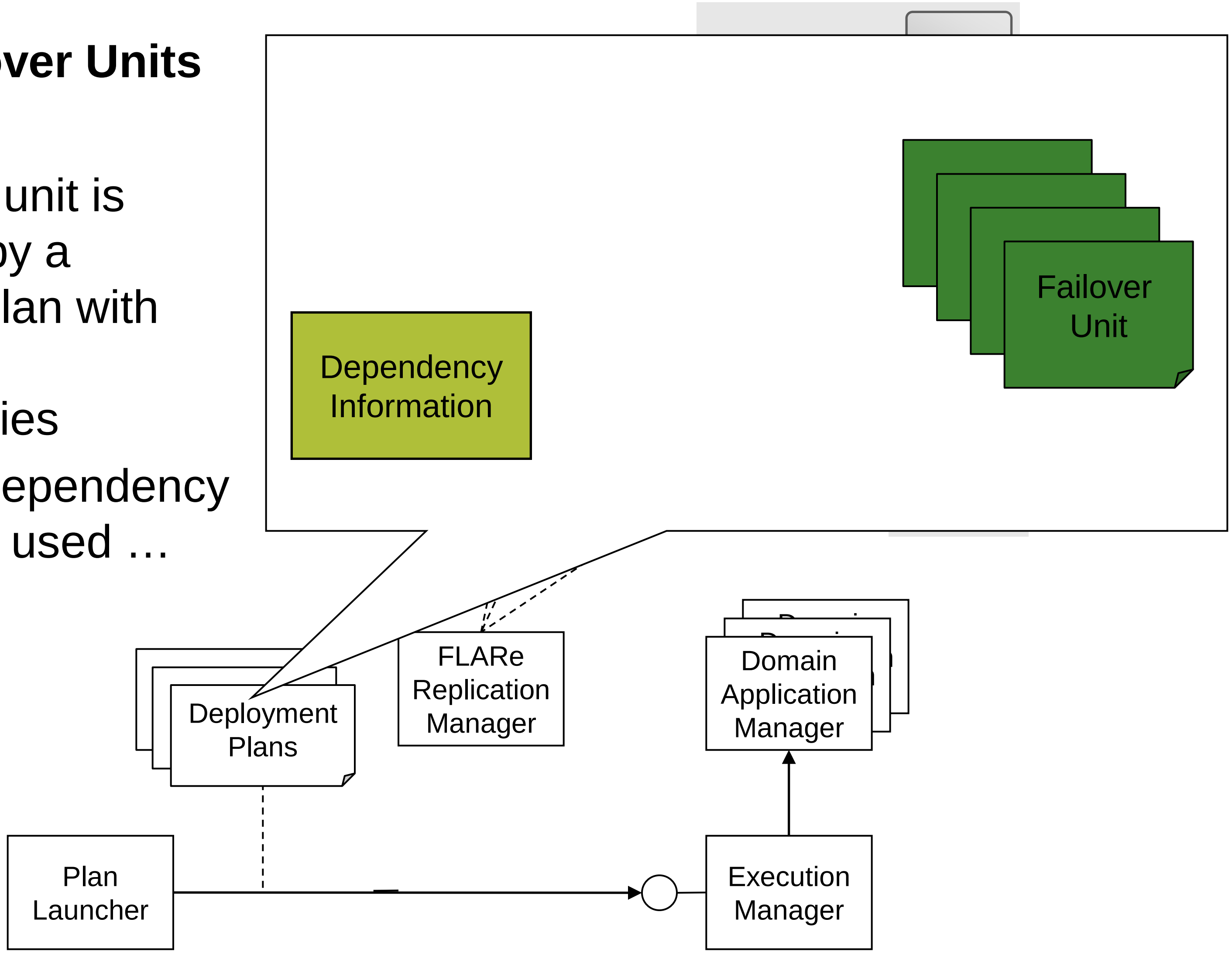
- Each failover unit is represented by a deployment plan with additional configProperties



Deployment Plan Preparation Solution

Solution: Failover Units

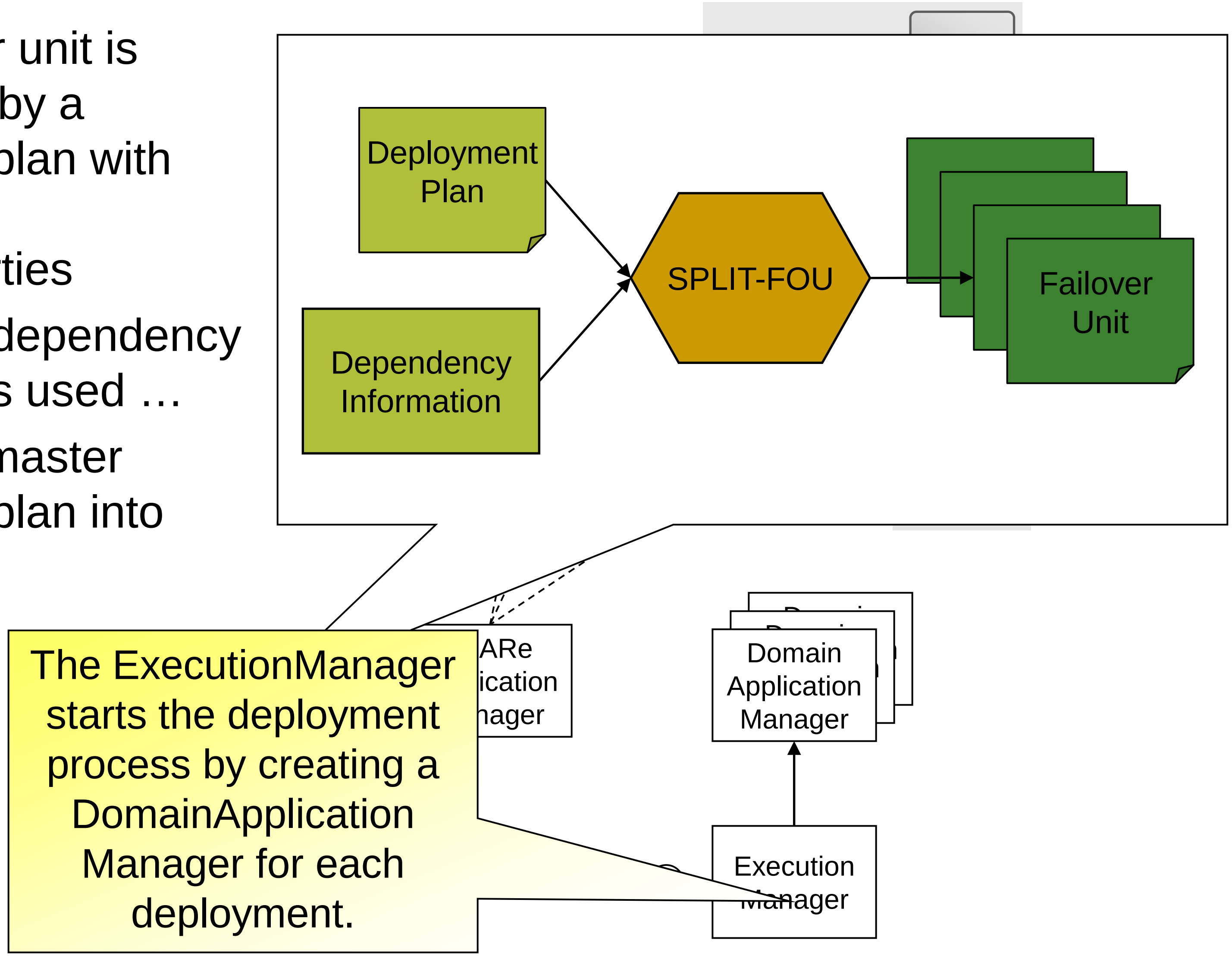
- Each failover unit is represented by a deployment plan with additional configProperties
- Component dependency information is used …



Deployment Plan Preparation Solution

Solution: Failover Units

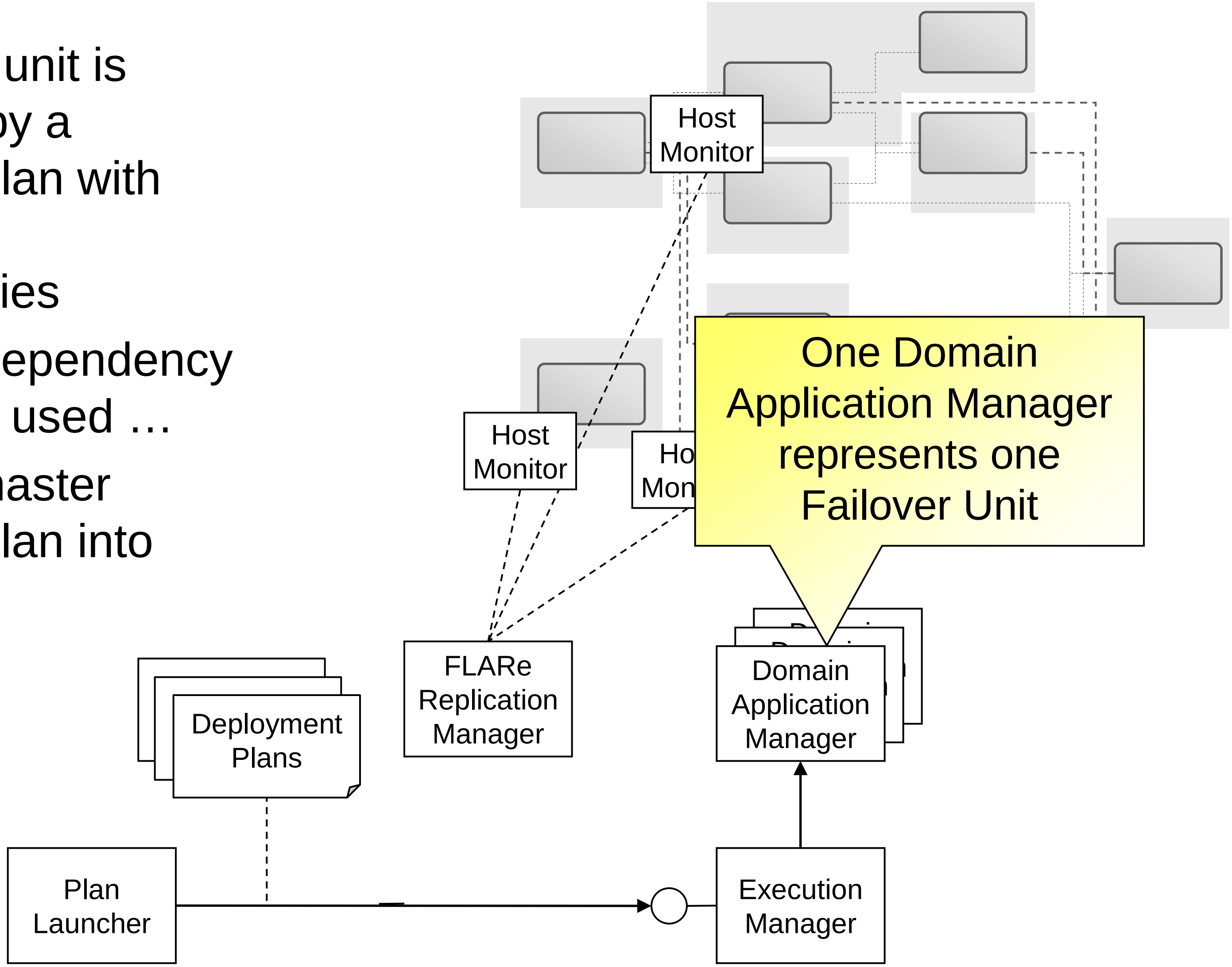
- Each failover unit is represented by a deployment plan with additional configProperties
- Component dependency information is used …
- … to split a master deployment plan into failover units



Deployment Plan Preparation Solution

Solution: Failover Units

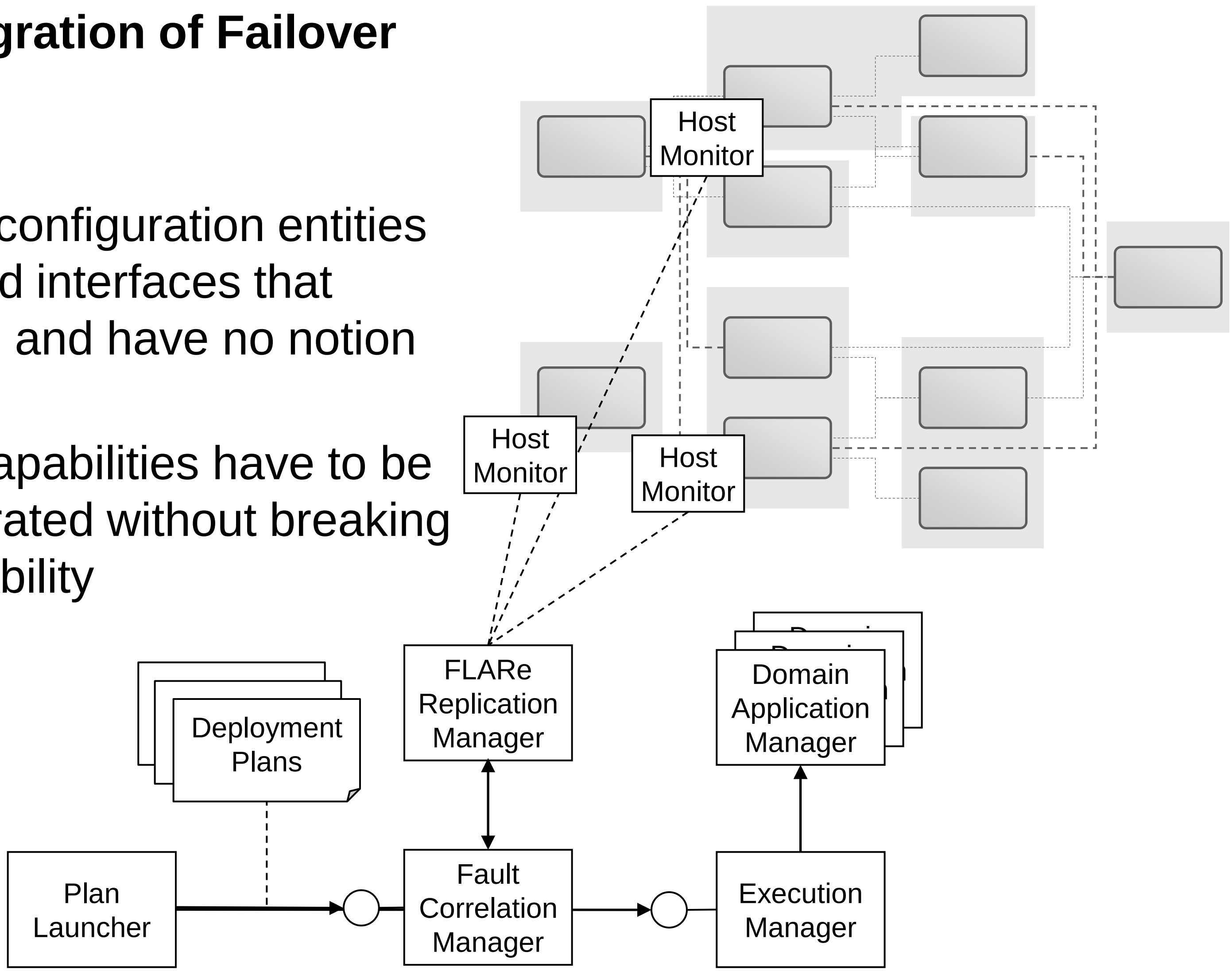
- Each failover unit is represented by a deployment plan with additional configProperties
- Component dependency information is used …
- … to split a master deployment plan into failover units



Integration of Failover Functionality Solution

Challenge 2 : Integration of Failover Functionality

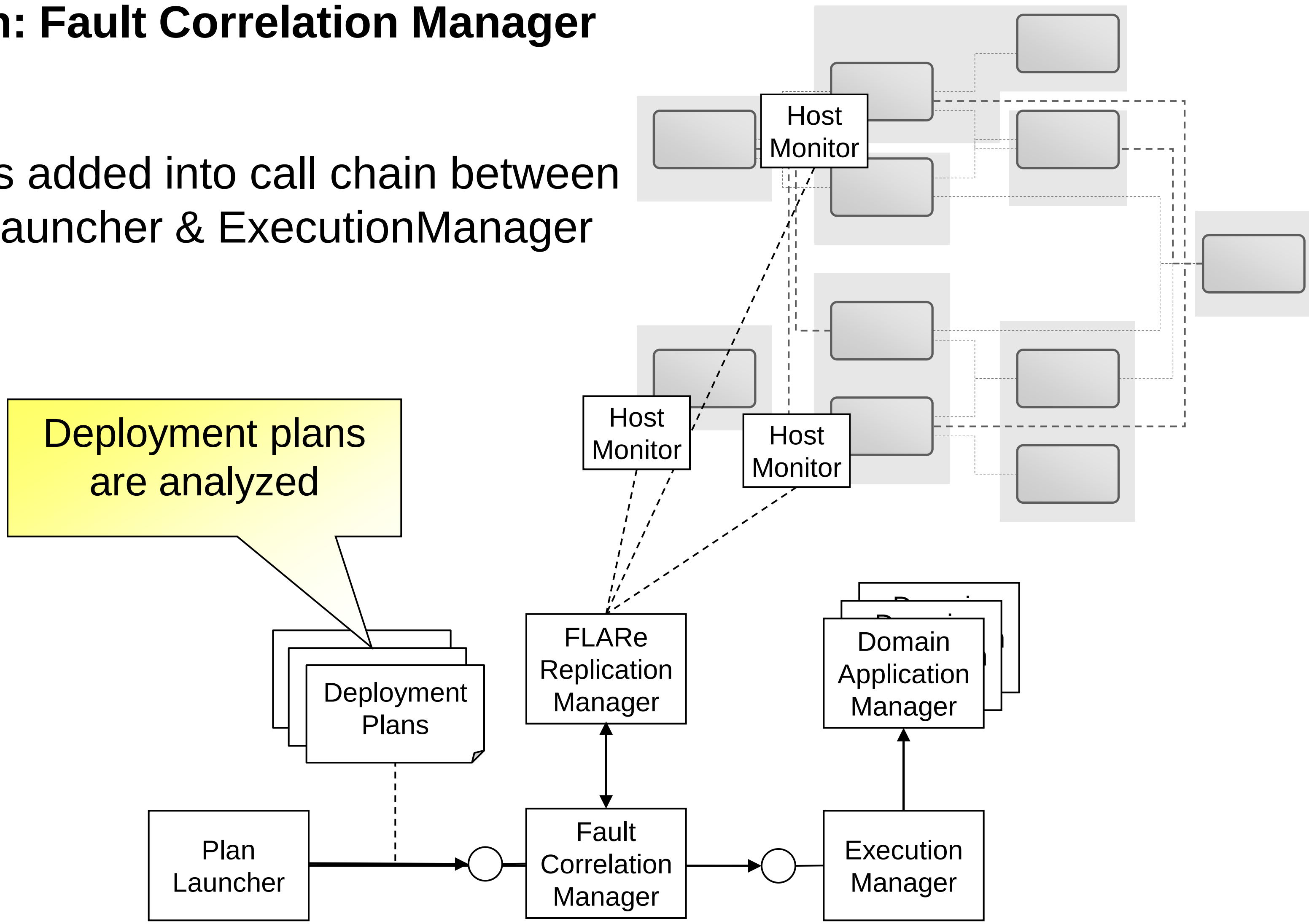
- Deployment and configuration entities have standardized interfaces that cannot be altered and have no notion of fault-tolerance
- Fault-tolerance capabilities have to be seamlessly integrated without breaking standard compatibility



Integration of Failover Functionality Solution

Solution: Fault Correlation Manager (FCM)

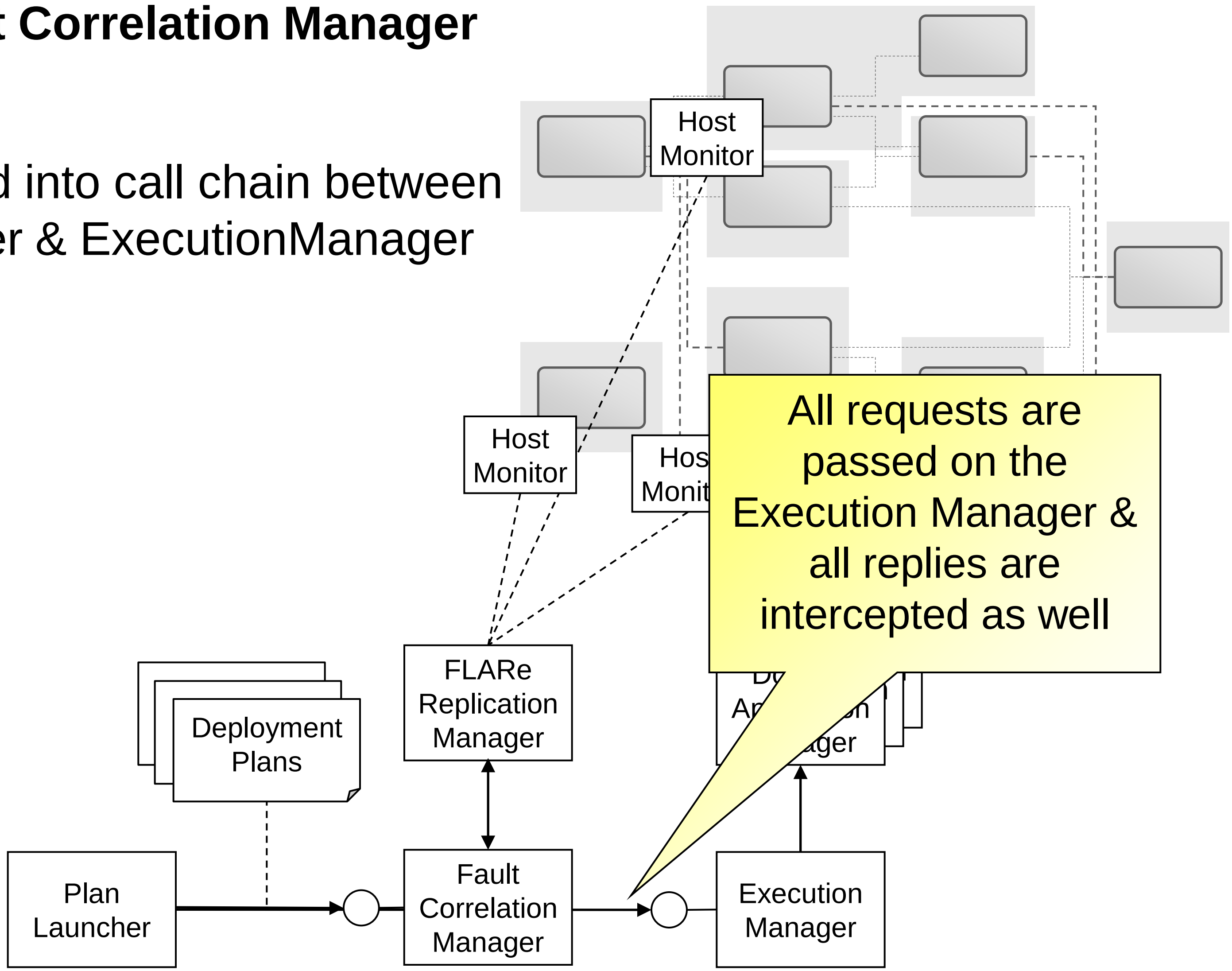
- FCM is added into call chain between Plan Launcher & ExecutionManager



Integration of Failover Functionality Solution

Solution: Fault Correlation Manager (FCM)

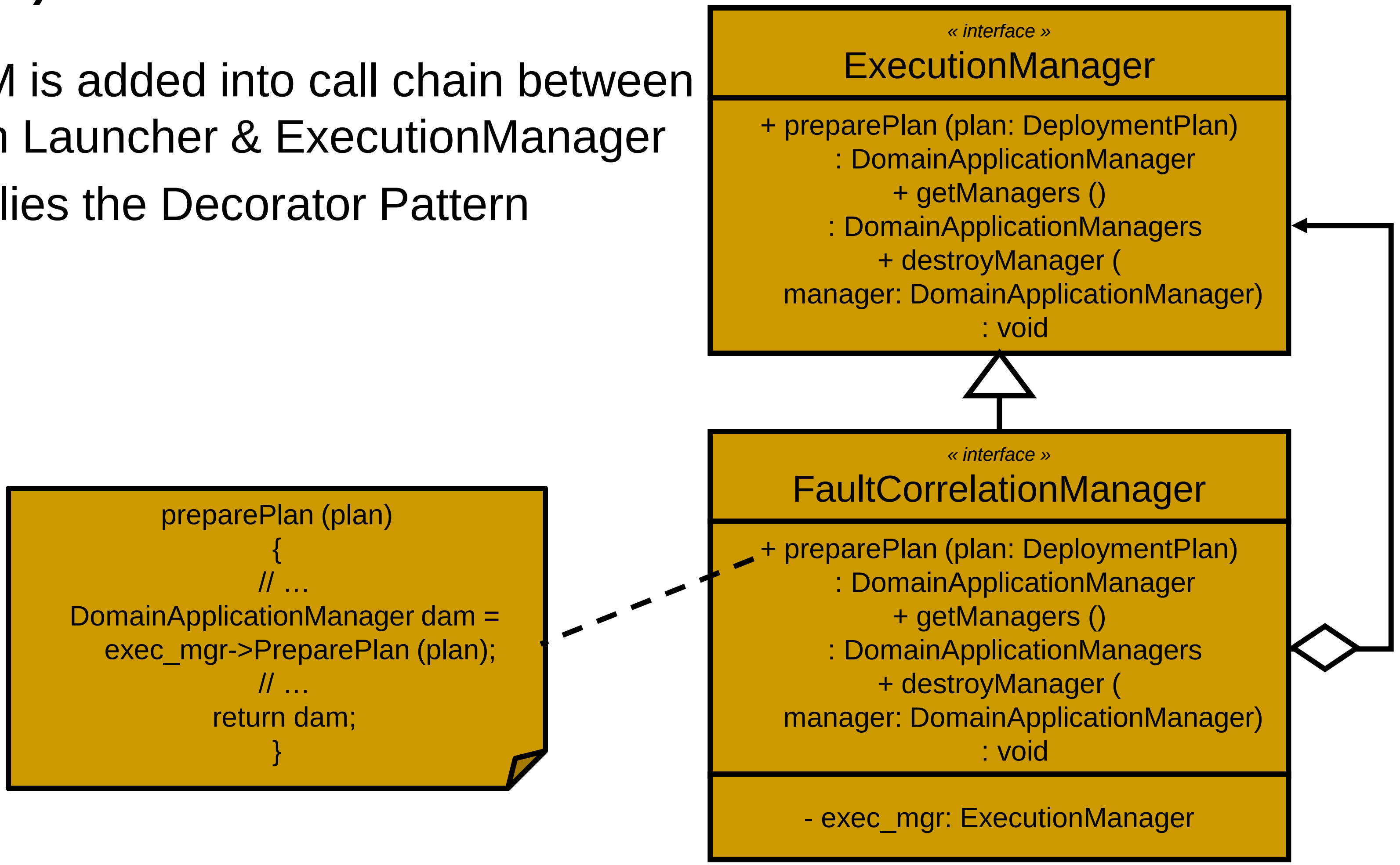
- FCM is added into call chain between Plan Launcher & ExecutionManager



Integration of Failover Functionality Solution

Solution: Fault Correlation Manager (FCM)

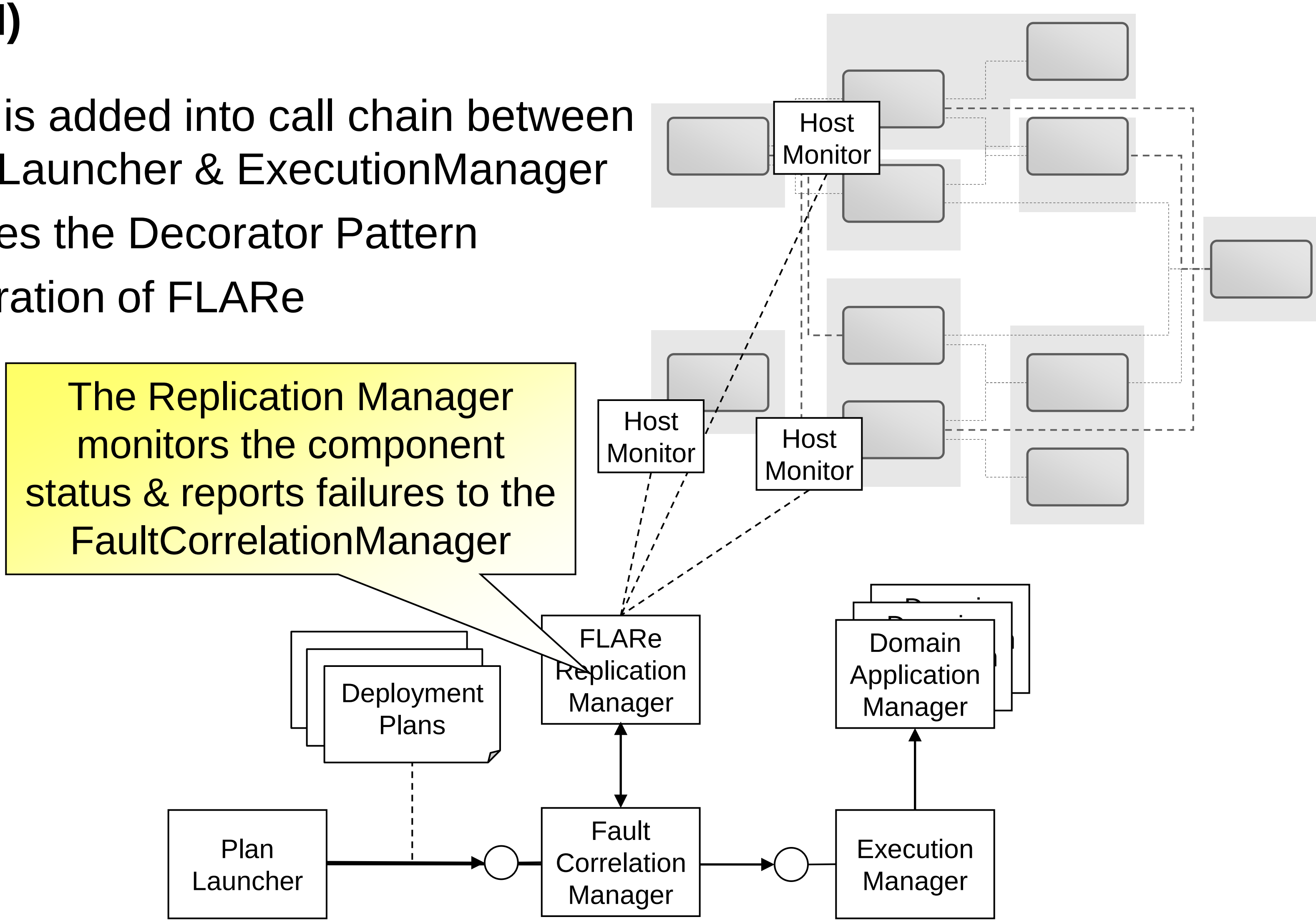
- FCM is added into call chain between Plan Launcher & ExecutionManager
- Applies the Decorator Pattern



Integration of Failover Functionality Solution

Solution: Fault Correlation Manager (FCM)

- FCM is added into call chain between Plan Launcher & ExecutionManager
- Applies the Decorator Pattern
- Integration of FLARe

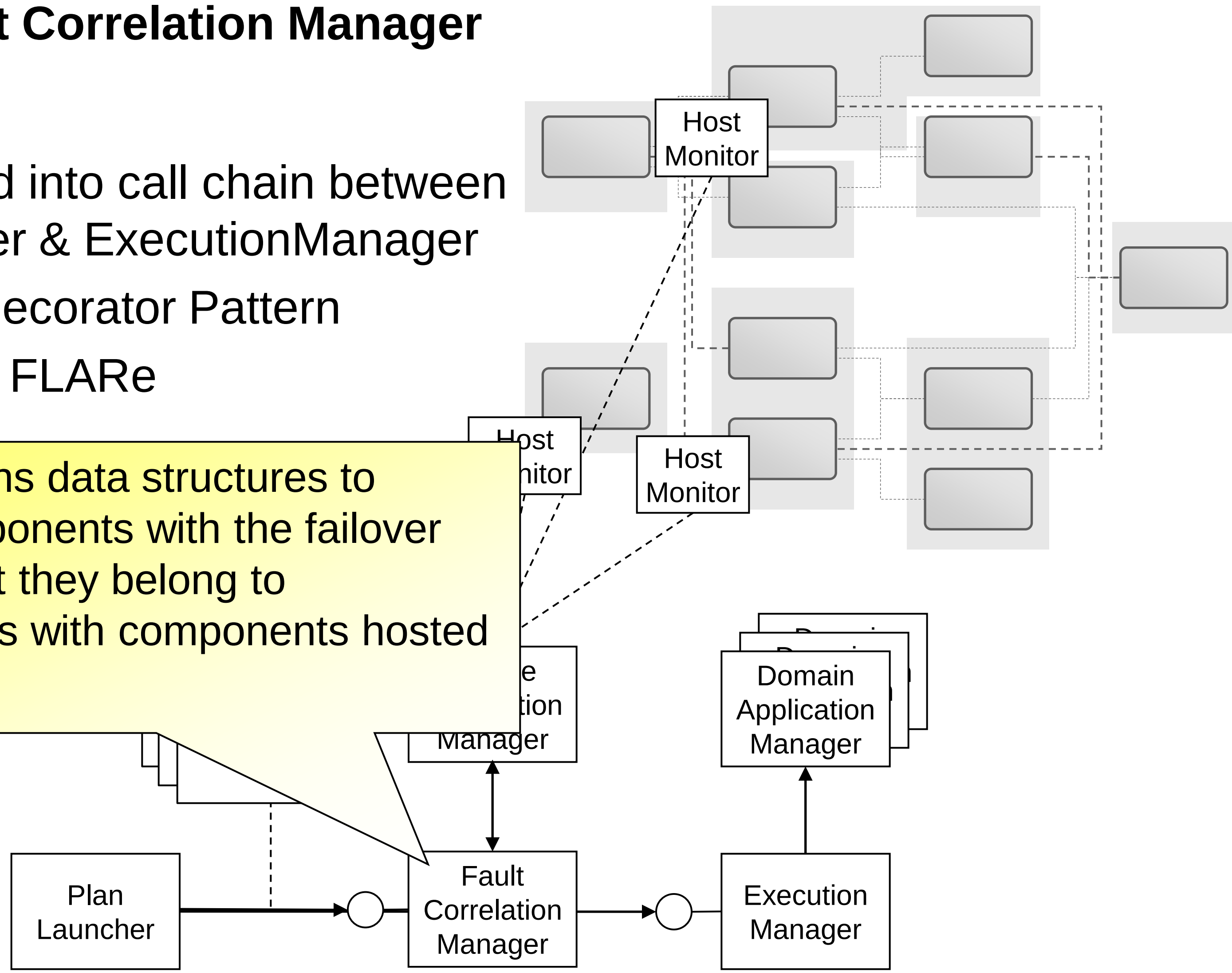


Integration of Failover Functionality Solution

Solution: Fault Correlation Manager (FCM)

- FCM is added into call chain between Plan Launcher & ExecutionManager
- Applies the Decorator Pattern
- Integration of FLARe

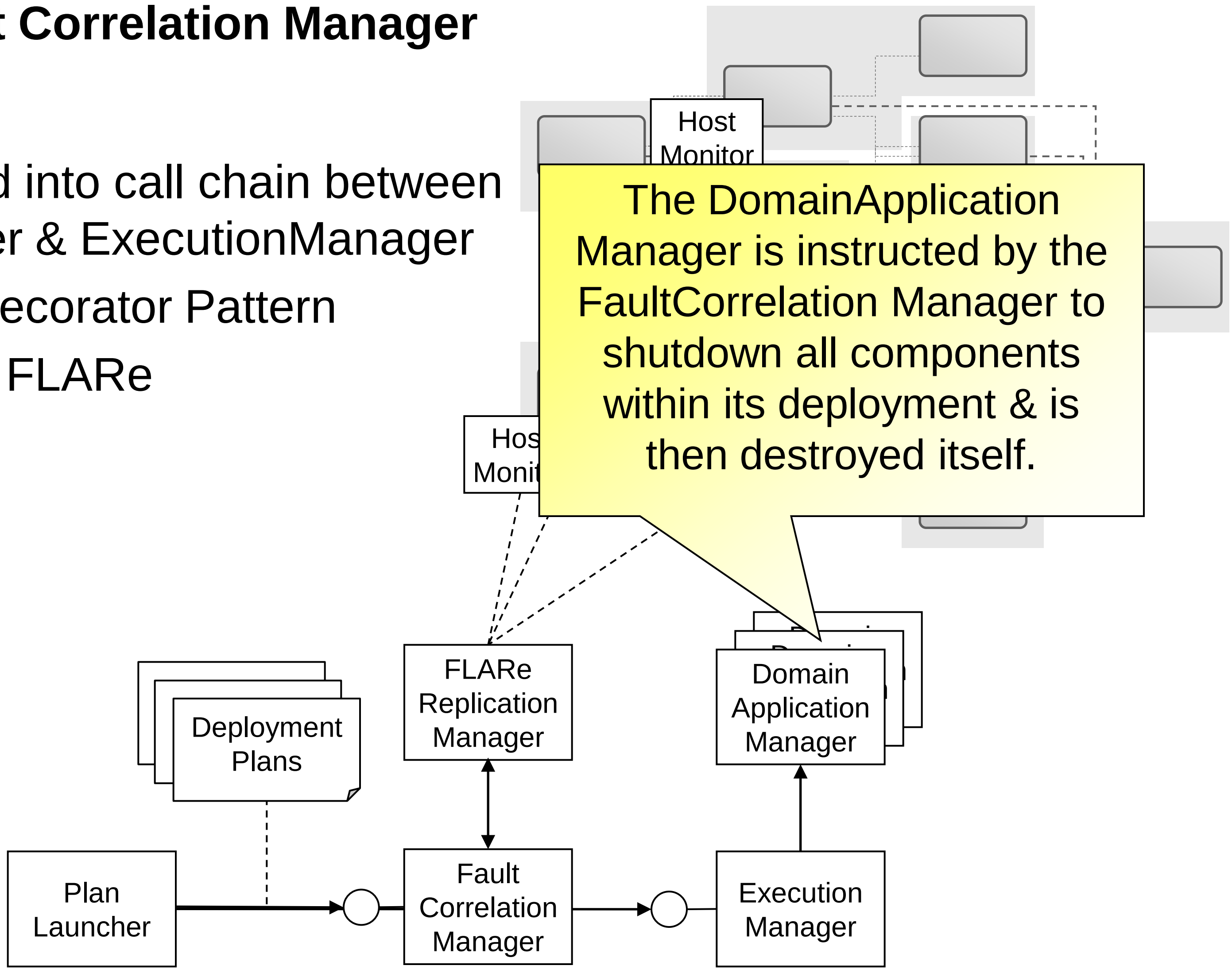
The FCM maintains data structures to
1. Associate components with the failover unit deployment they belong to
2. Associate nodes with components hosted on these nodes



Integration of Failover Functionality Solution

Solution: Fault Correlation Manager (FCM)

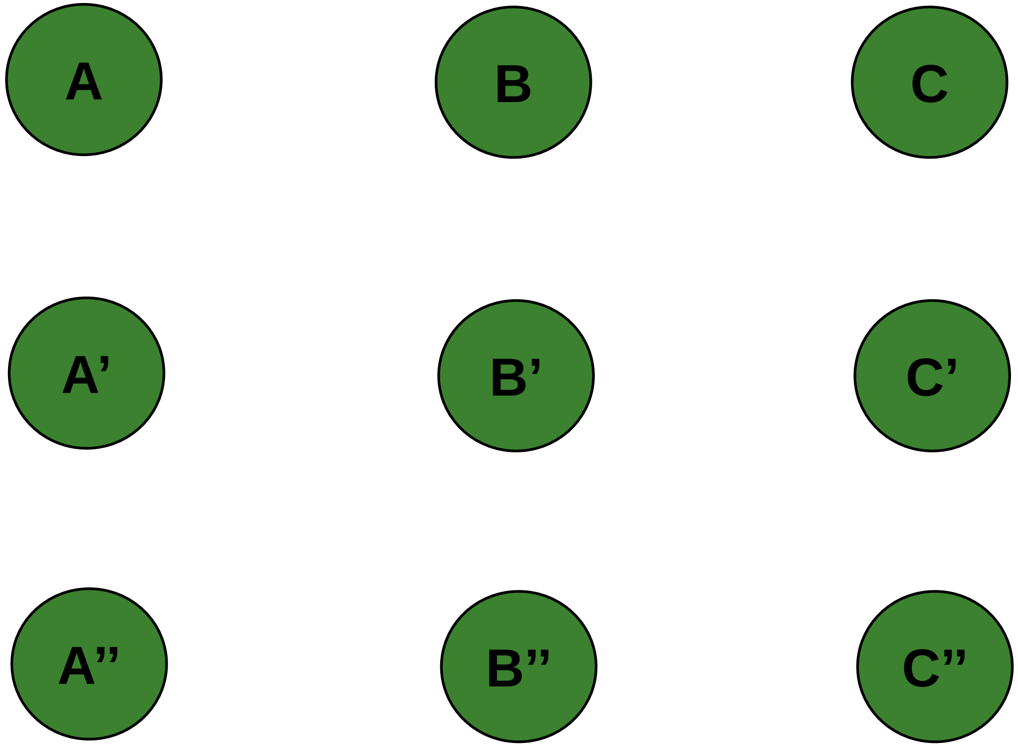
- FCM is added into call chain between Plan Launcher & ExecutionManager
- Applies the Decorator Pattern
- Integration of FLARe



Replica Failover Ordering Challenges

Challenge 3: Replica Failover Ordering

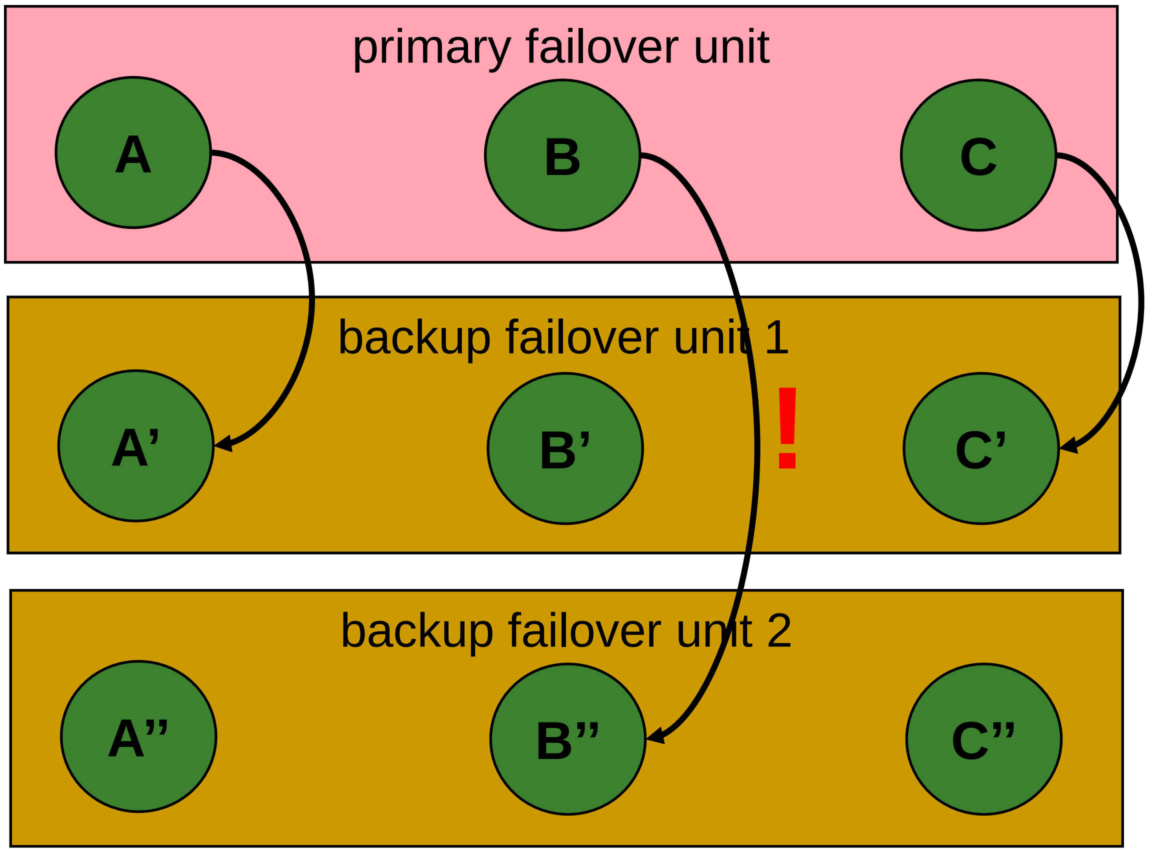
- Failovers happen on a per component /object basis



Replica Failover Ordering Challenges

Challenge 3: Replica Failover Ordering

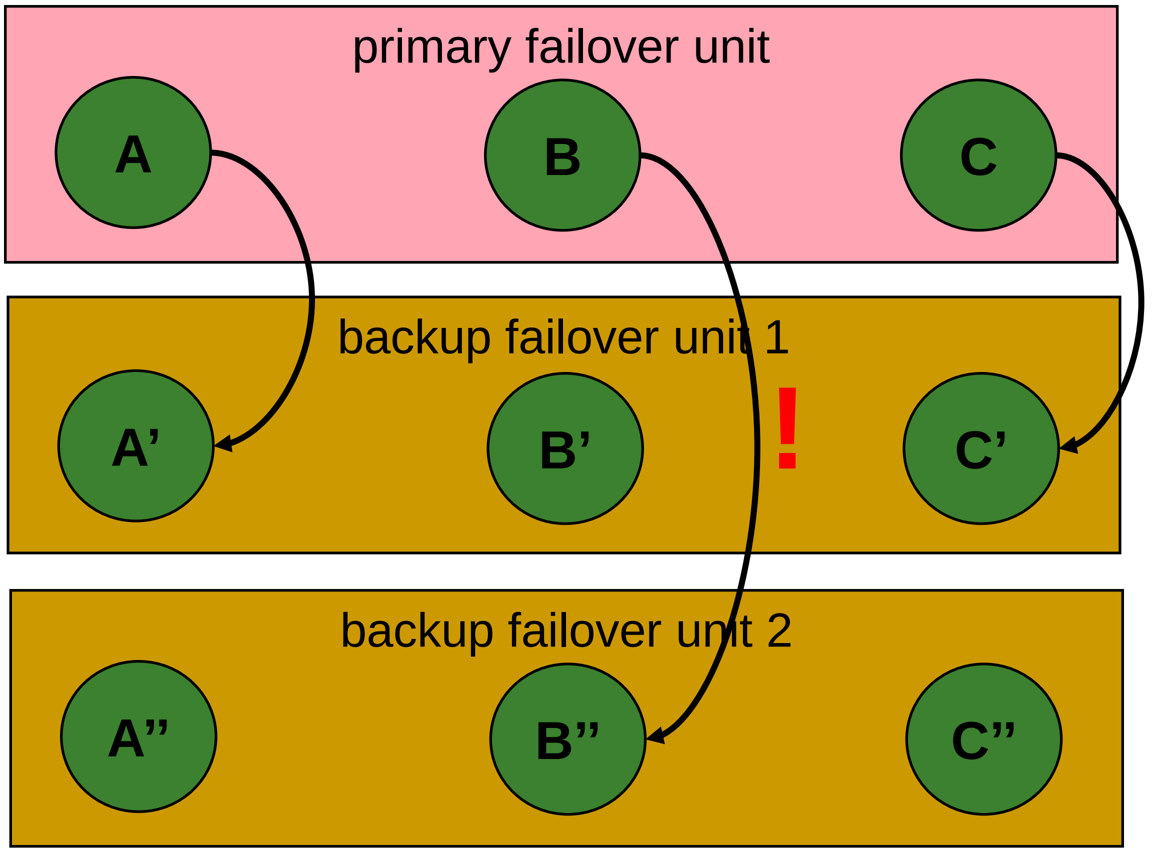
- Failovers happen on a per component /object basis
- FLARe uses a client side failover mechanism
  - An ordered list determines the failover order



Replica Failover Ordering Challenges

Challenge 3: Replica Failover Ordering

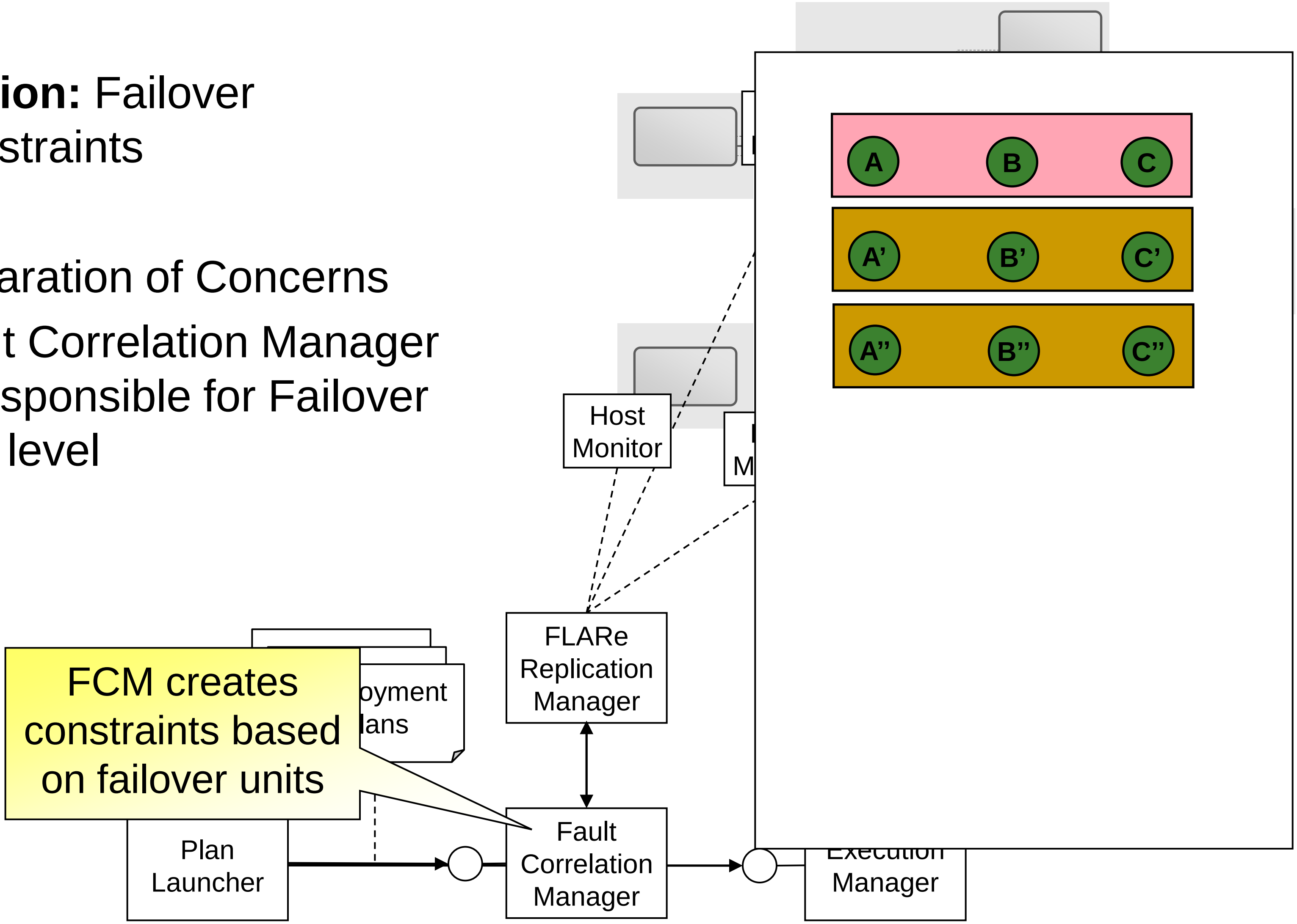
- Failovers happen on a per component /object basis
- FLARe uses a client side failover mechanism
  - An ordered list determines the failover order
- The ReplicationManager needs to provide correct ordering



Replica Failover Ordering Solution

Solution: Failover Constraints

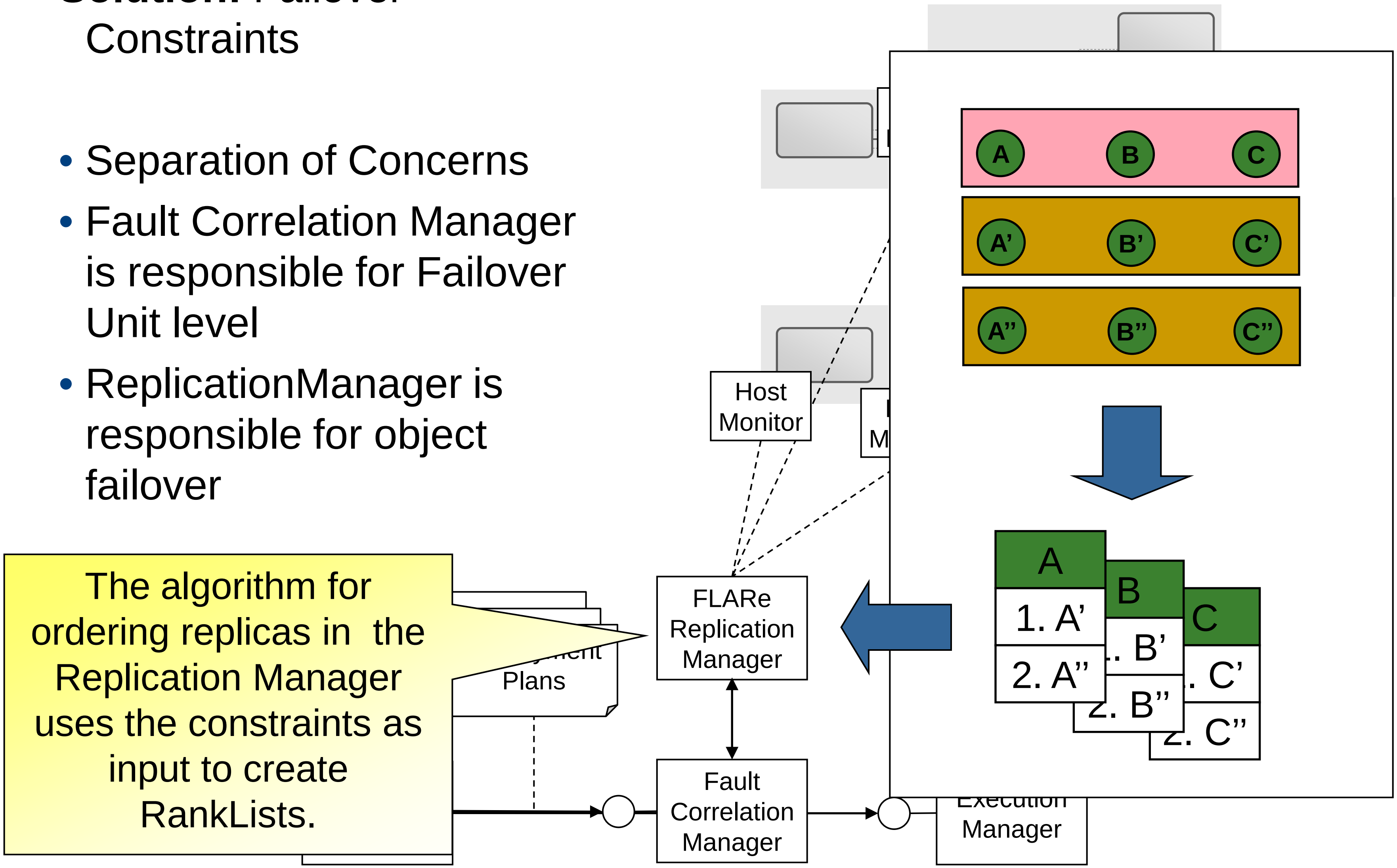
- Separation of Concerns
- Fault Correlation Manager is responsible for Failover Unit level



Replica Failover Ordering Solution

Solution: Failover Constraints

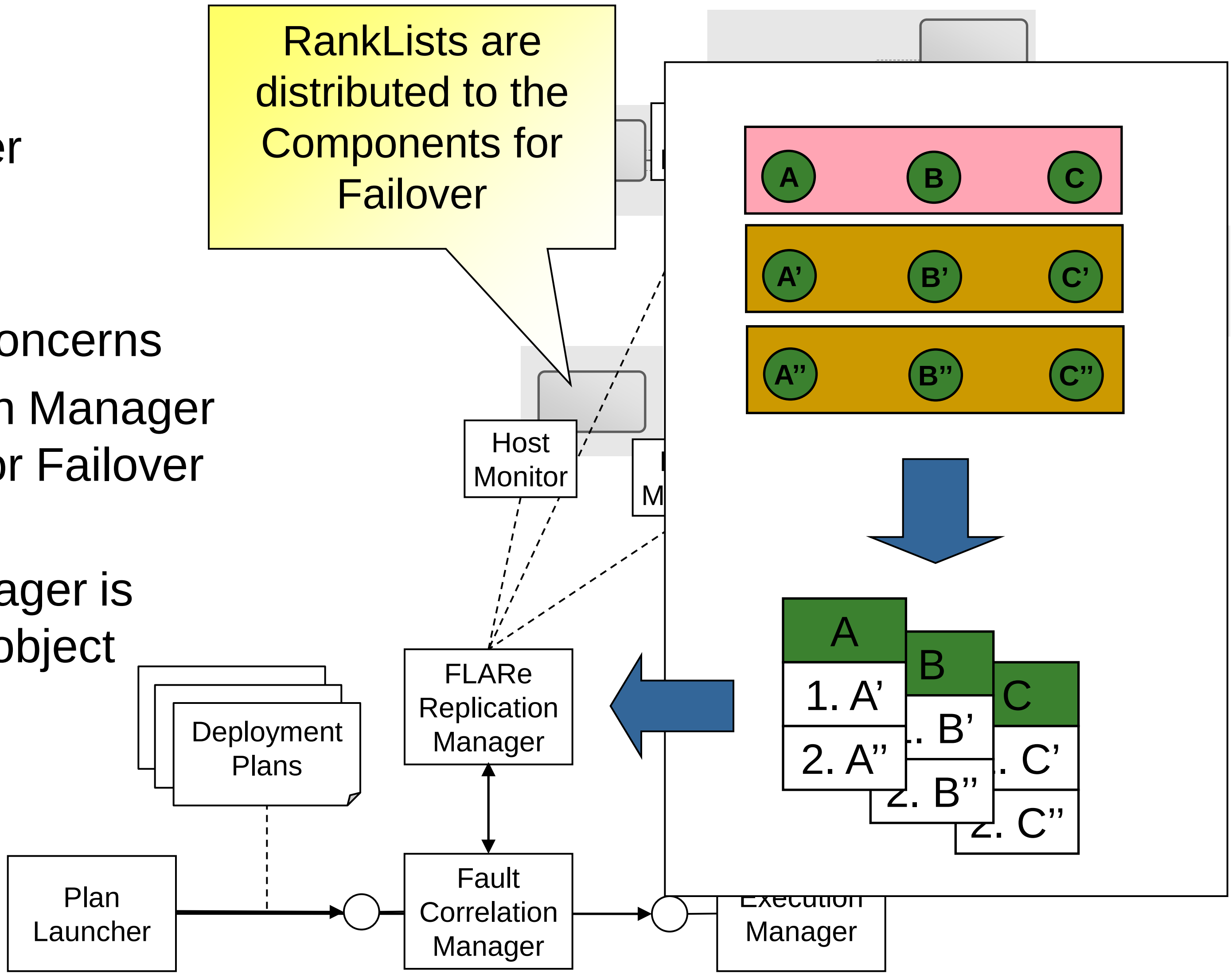
- Separation of Concerns
- Fault Correlation Manager is responsible for Failover Unit level
- ReplicationManager is responsible for object failover



Replica Failover Ordering Solution

Solution: Failover Constraints

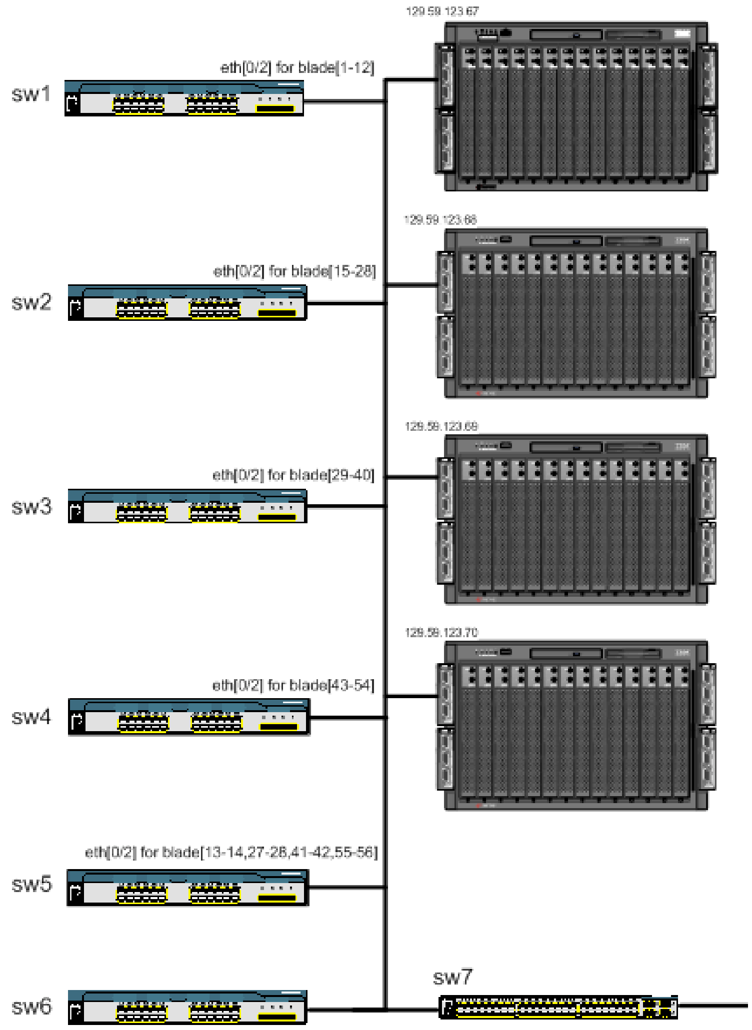
- Separation of Concerns
- Fault Correlation Manager is responsible for Failover Unit level
- ReplicationManager is responsible for object failover



Experimental Evaluation of CORFU

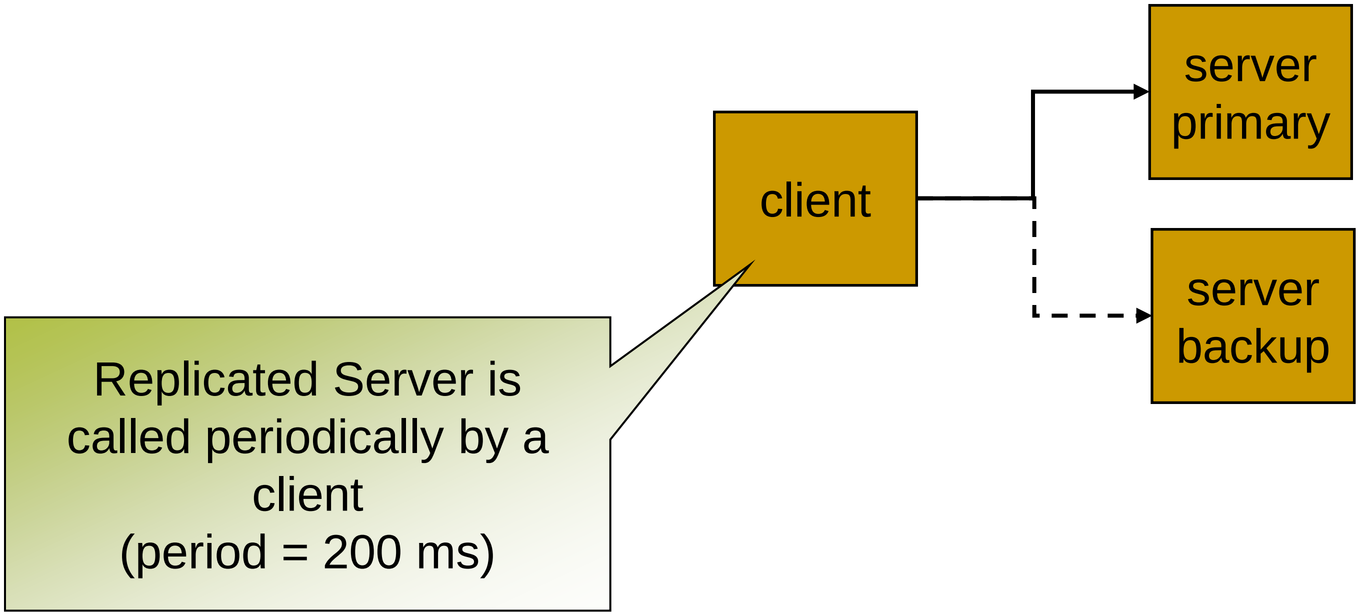
Testing Environment

- ISISLab LAN virtualization environment
- Identical blades with two 2.8GHz Xeon CPUs, 1 GB of RAM, 40 GB HDD, & 4 Gbps network interfaces (only one CPU used by kernel)
- Fedora Core 6 linux with rt11 real-time kernel patches
- Compiler gcc 3.4.6
- CORBA Implementation: TAO branch based on version 1.6.8 with FLARe
- CCM Implementation: CIAO branch based on version 0.6.8 with CORFU additions



Experimental Evaluation of CORFU

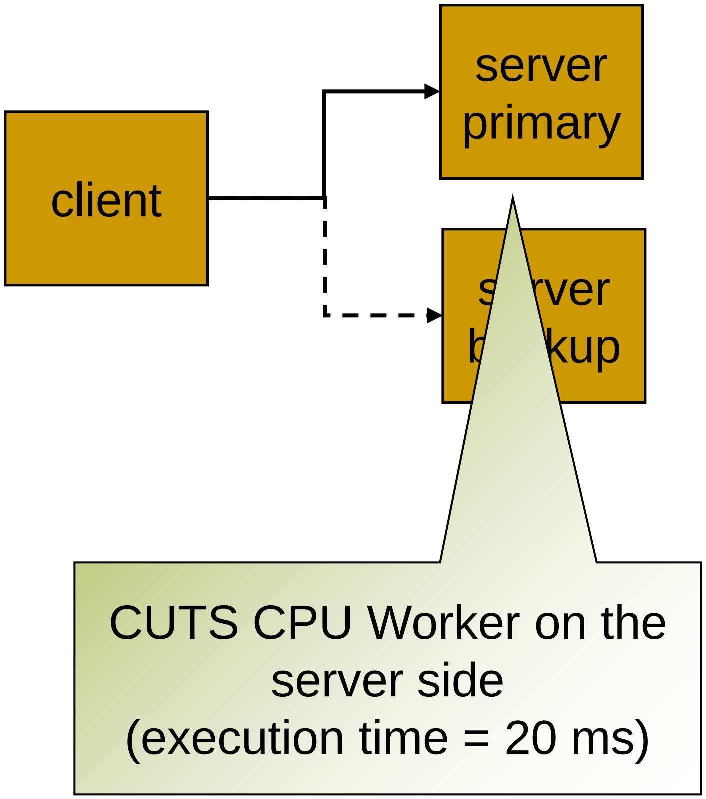
Experiment 1 - Overhead of Client Failover



Experimental Evaluation of CORFU

Experiment 1 - Overhead of Client Failover

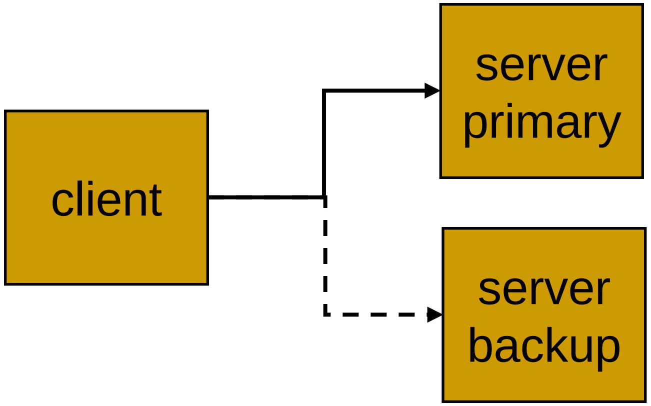
1. Two Setups: CORBA 2.x based executables & components



Experimental Evaluation of CORFU

Experiment 1 - Overhead of Client Failover

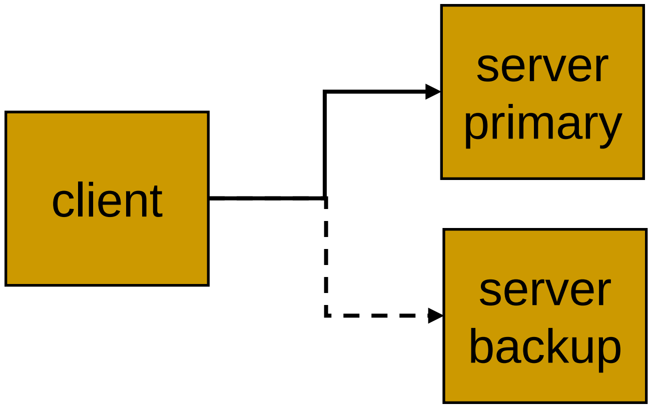
1. Two Setups: CORBA 2.x based executables & components
2. After a defined number of calls a fault is injected in the server that causes it to finish

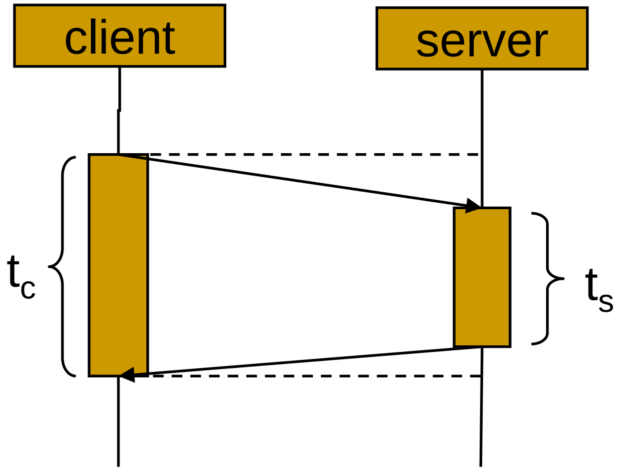


Experimental Evaluation of CORFU

Experiment 1 - Overhead of Client Failover

1. Two Setups: CORBA 2.x based executables & components
2. After a defined number of calls a fault is injected in the server that causes it to finish
3. Measure server response times in the client during failover



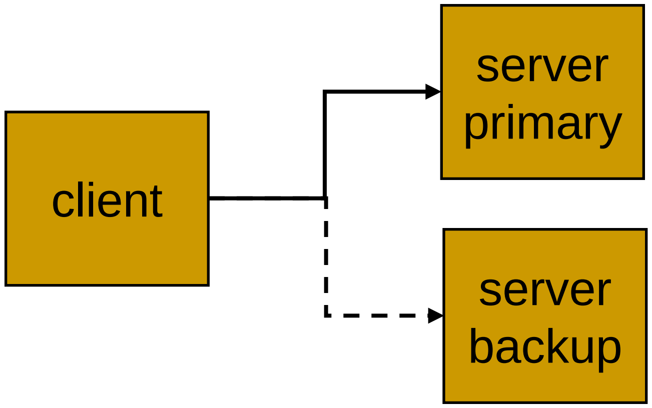


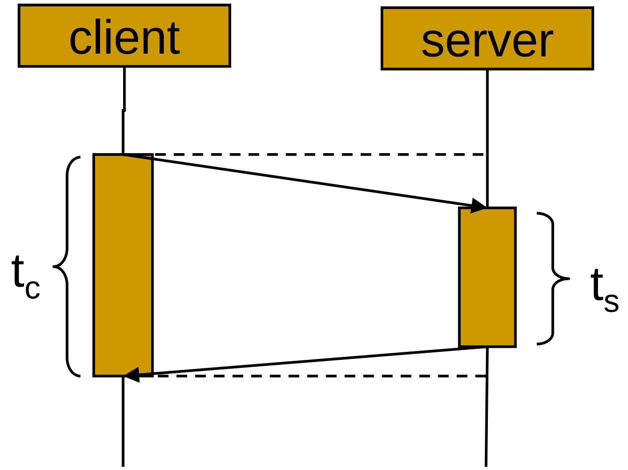
Communication Overhead $t_r = t_c - t_s$

Experimental Evaluation of CORFU

Experiment 1 - Overhead of Client Failover

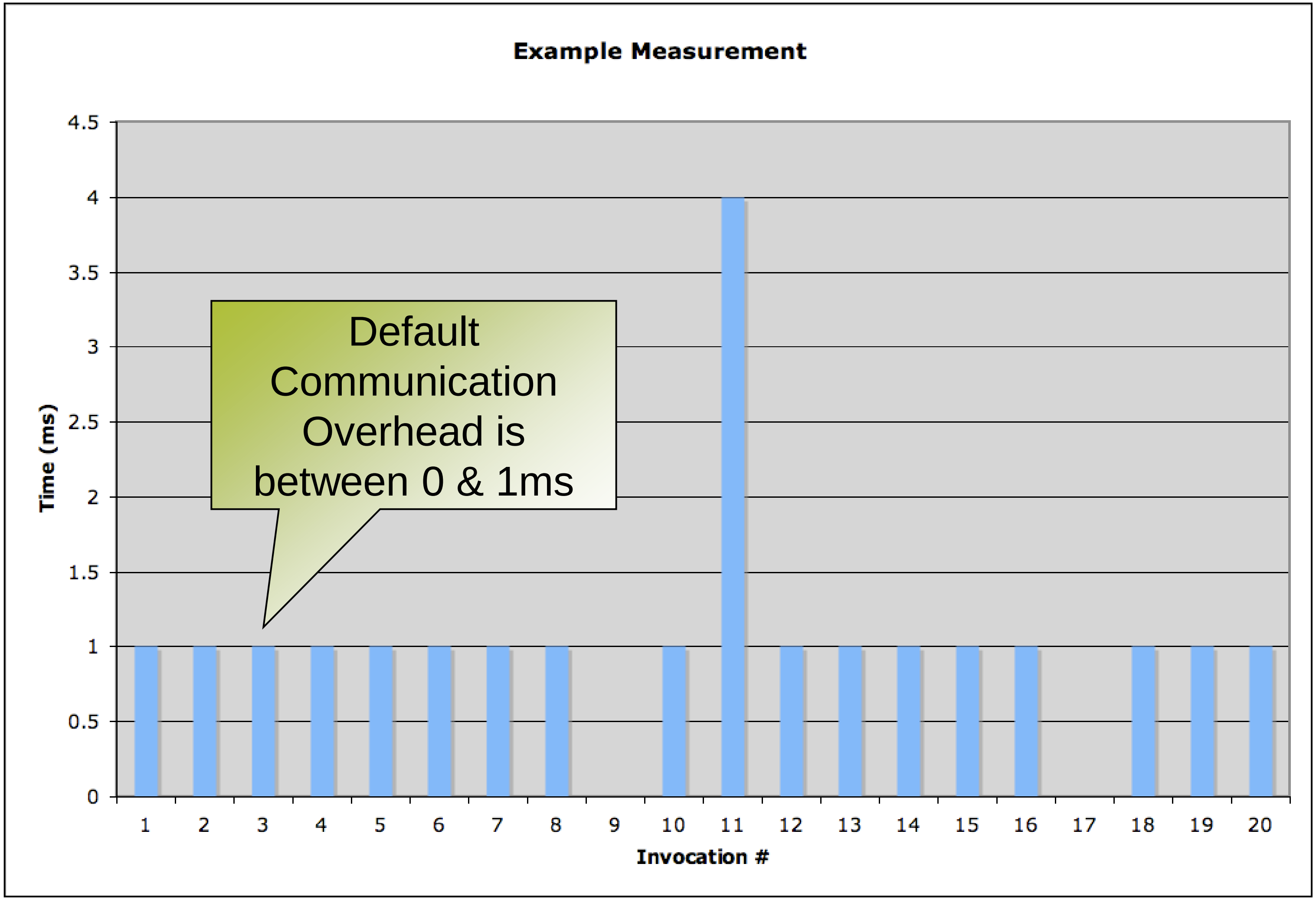
1. Two Setups: CORBA 2.x based executables & components
2. After a defined number of calls a fault is injected in the server that causes it to finish
3. Measure server response times in the client during failover
4. Compare response times between both versions
5. Three experiment configurations: 1 server application (10% load), 2 server applications (20%) & 4 server applications (40%)



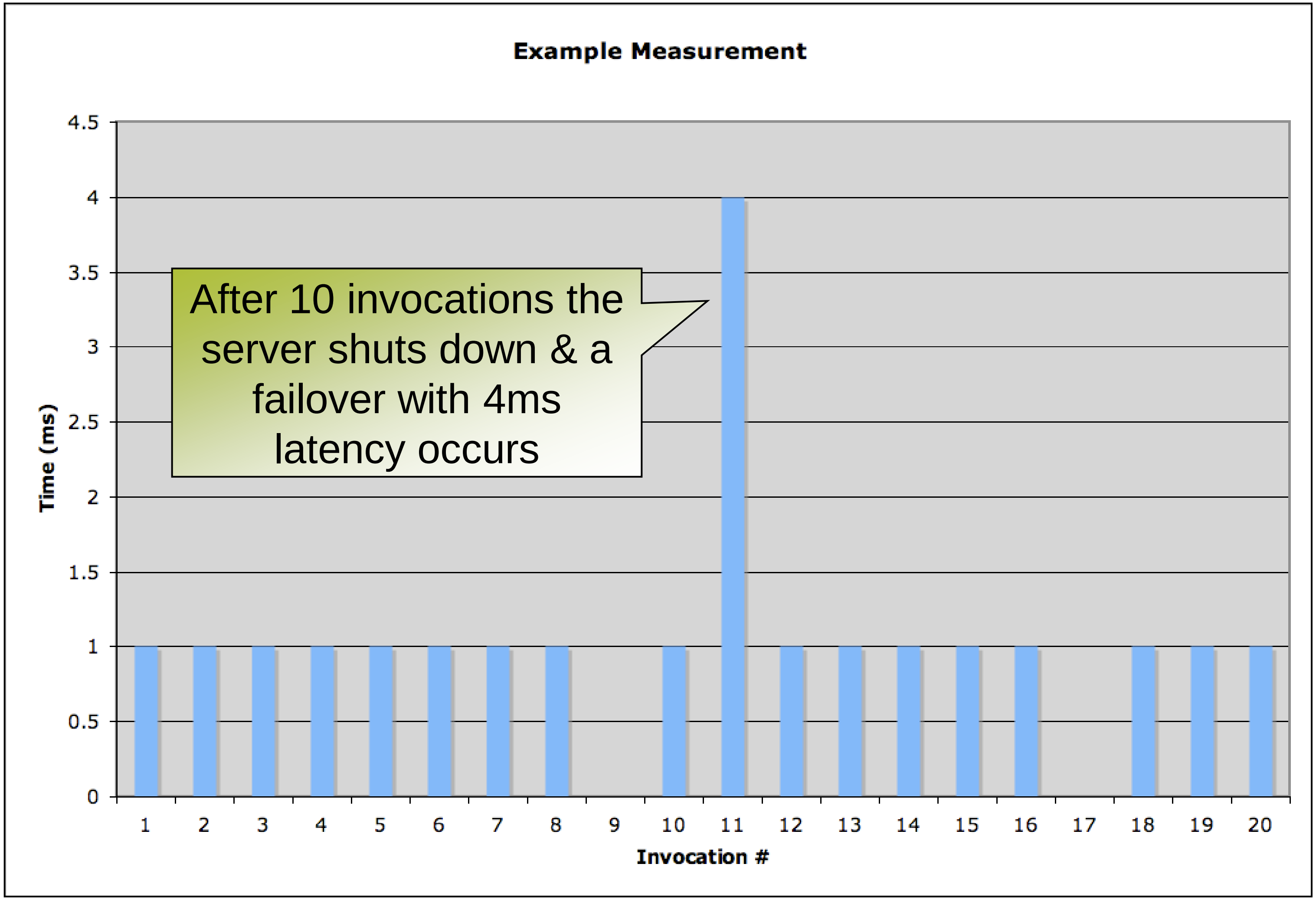


Communication Overhead $t_r = t_c - t_s$

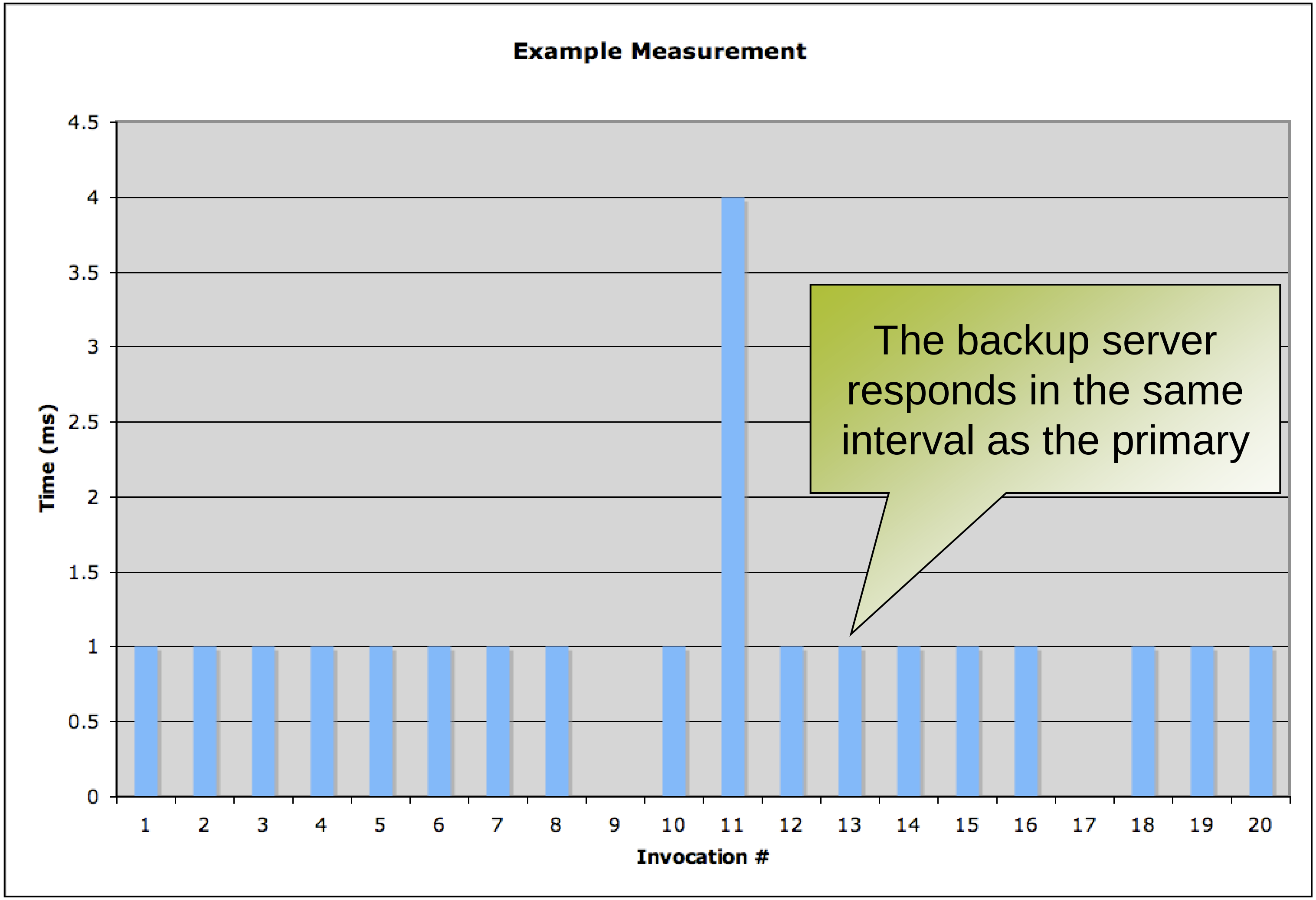
Experiment 1 - Results



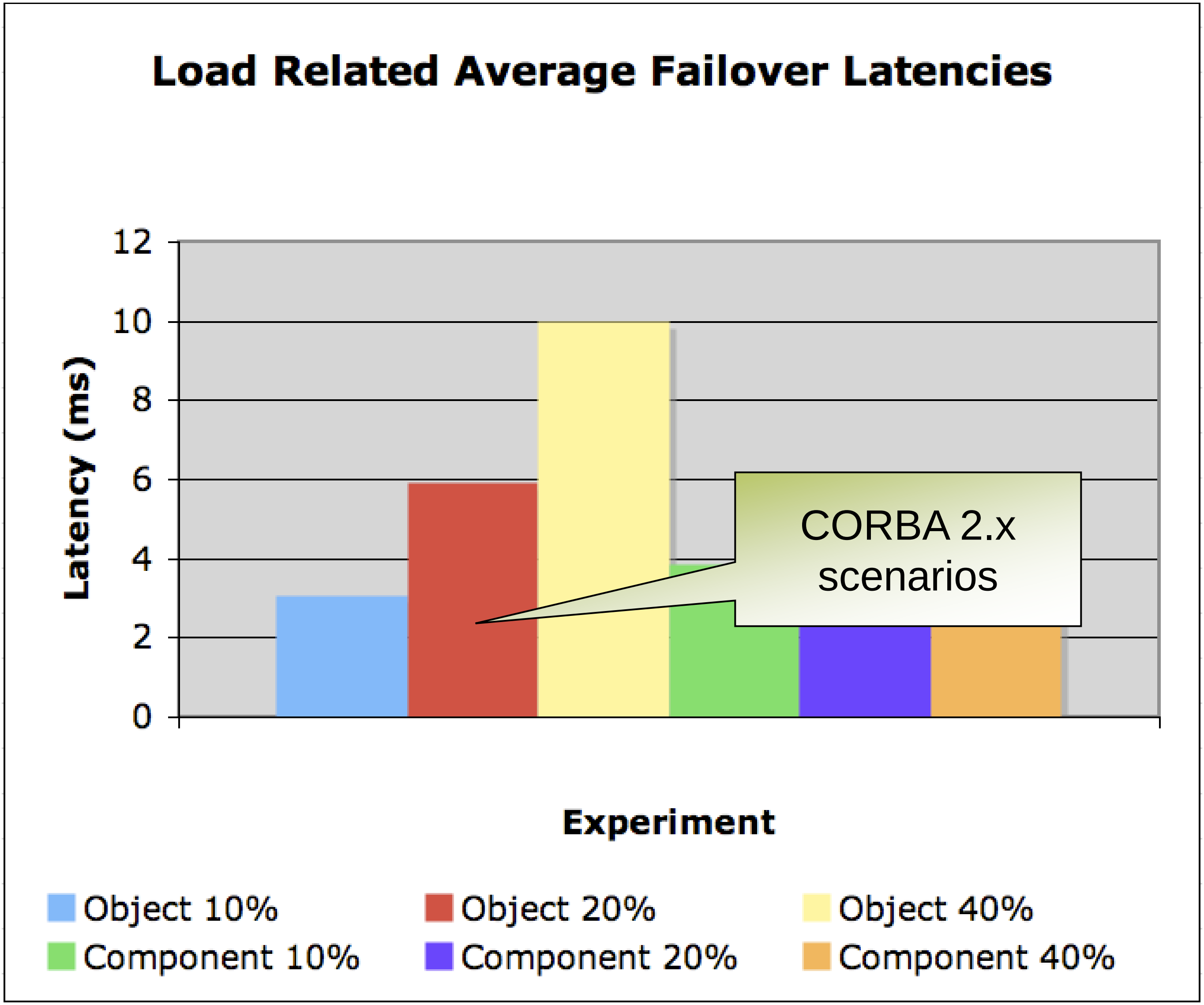
Experiment 1 - Results



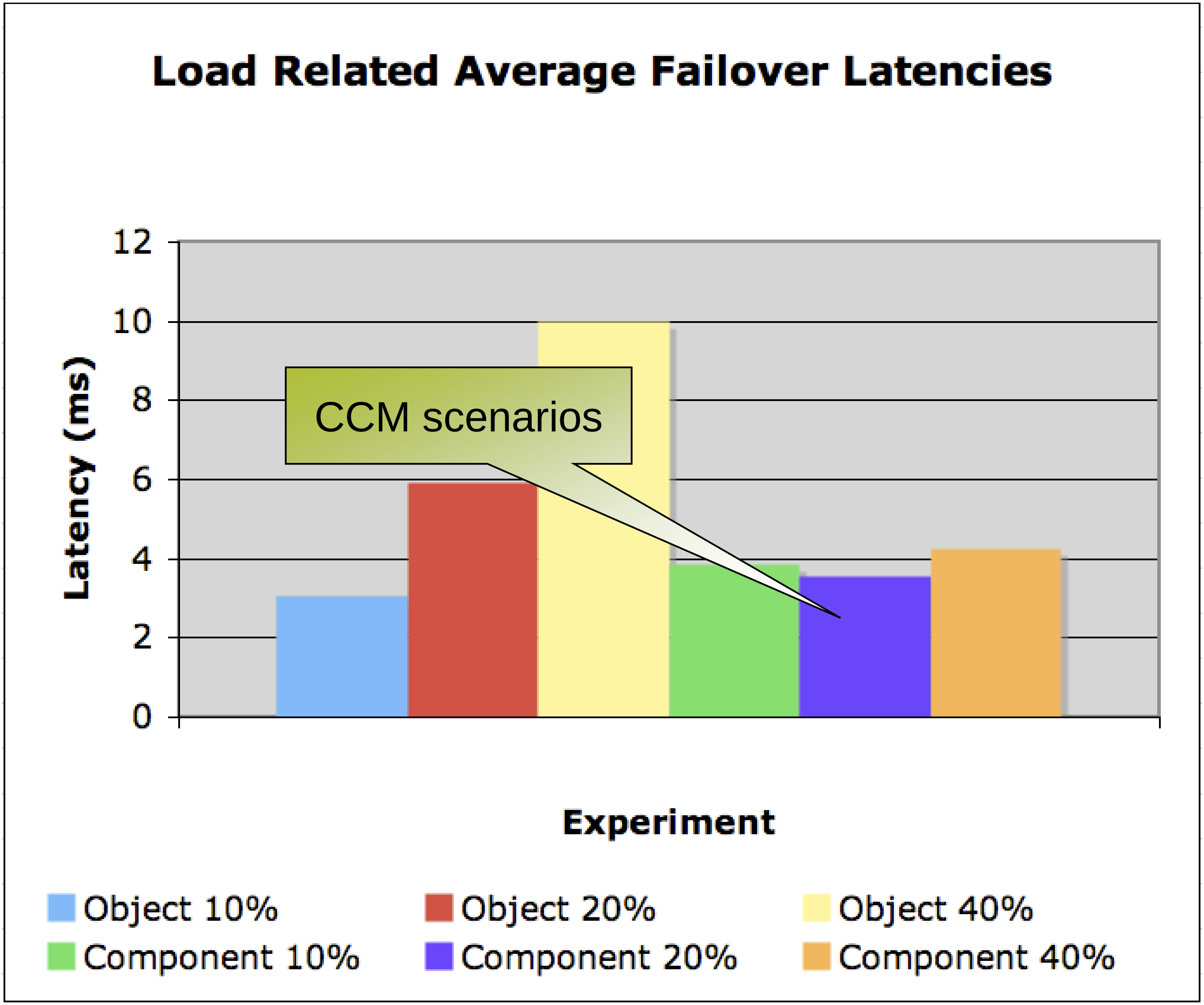
Experiment 1 - Results



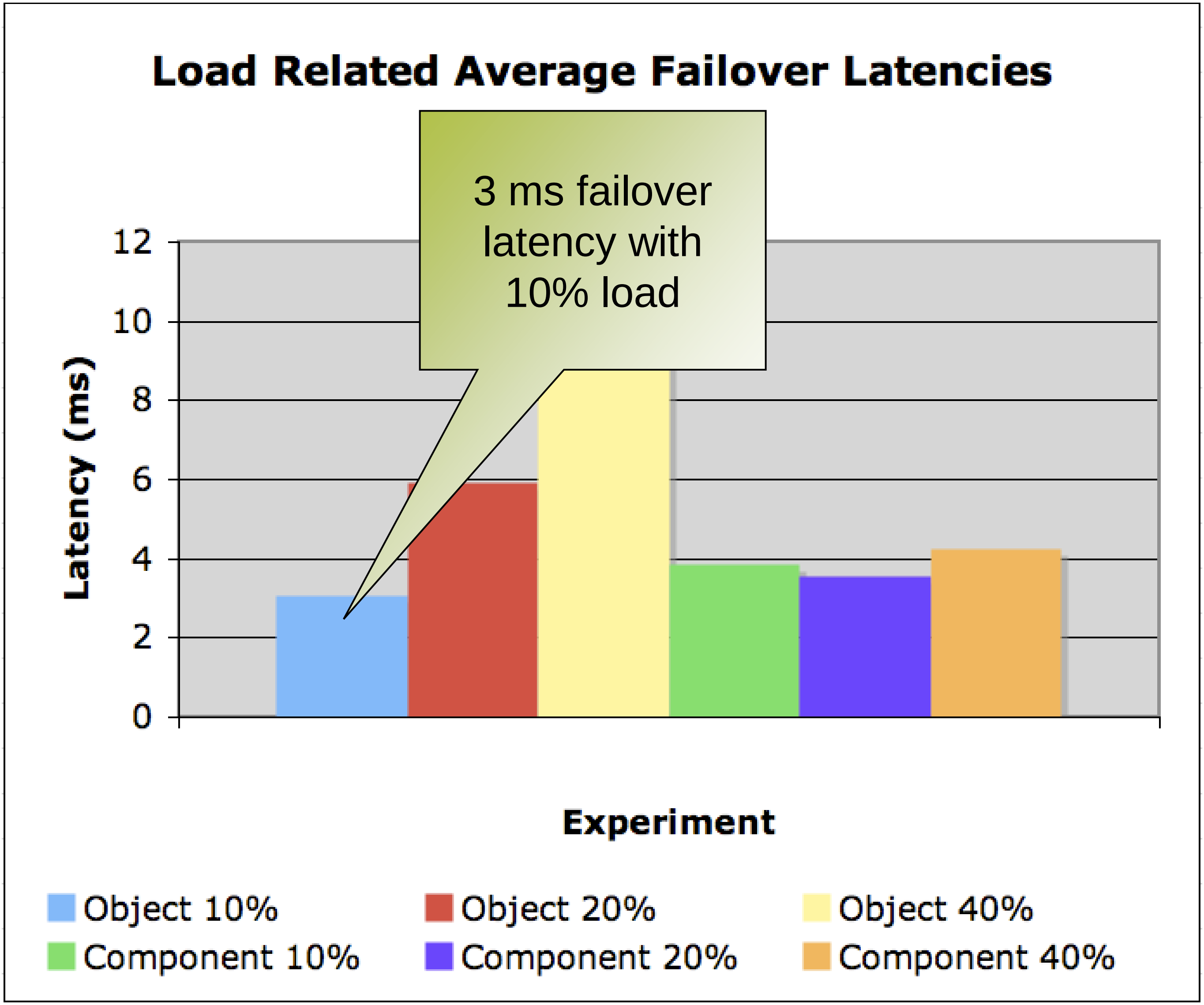
Experiment 1 - Results



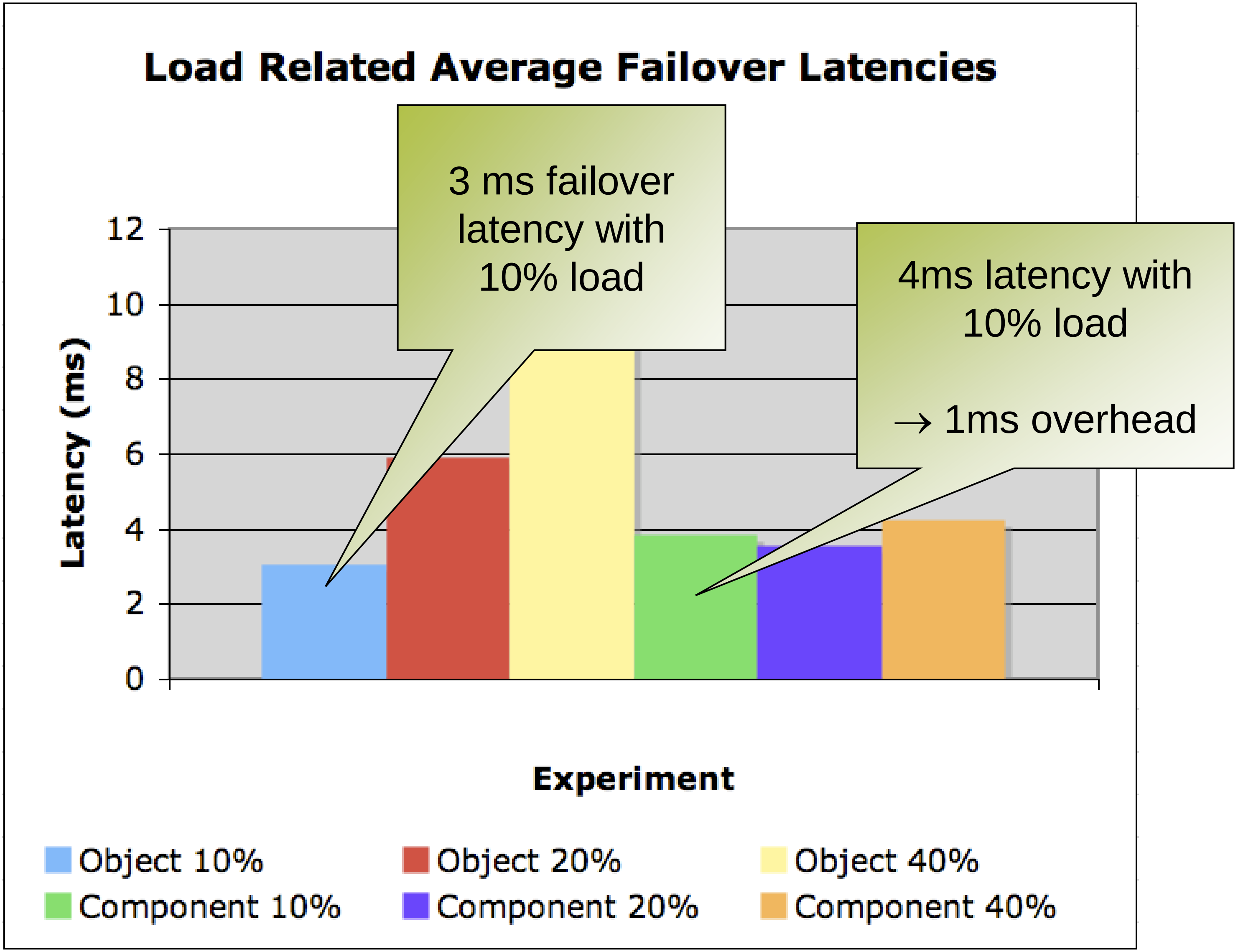
Experiment 1 - Results



Experiment 1 - Results



Experiment 1 - Results

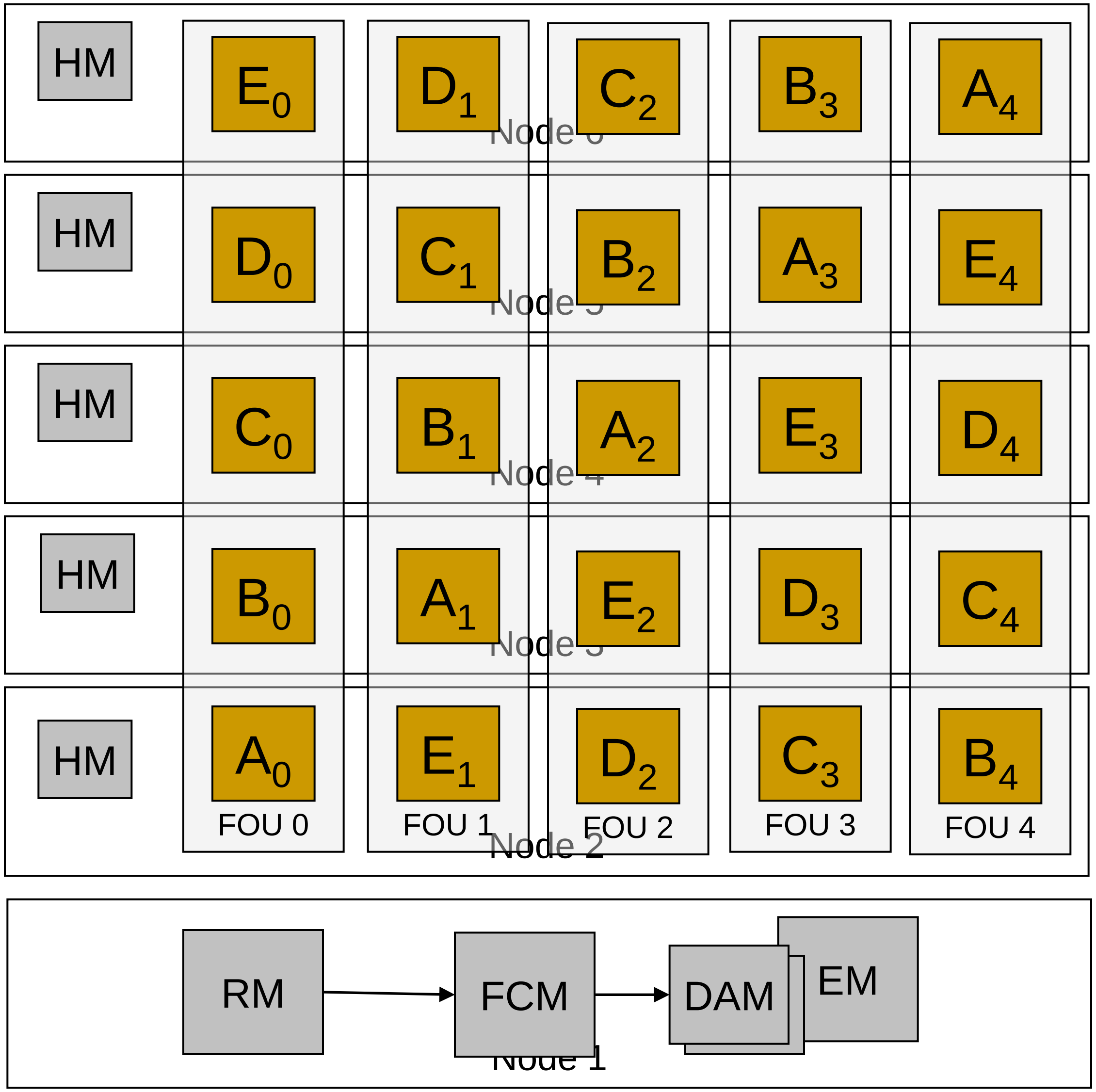


Experimental Evaluation

Experiment 2:

Fail-Stop shutdown latency

- Five Failover Units on Five Nodes

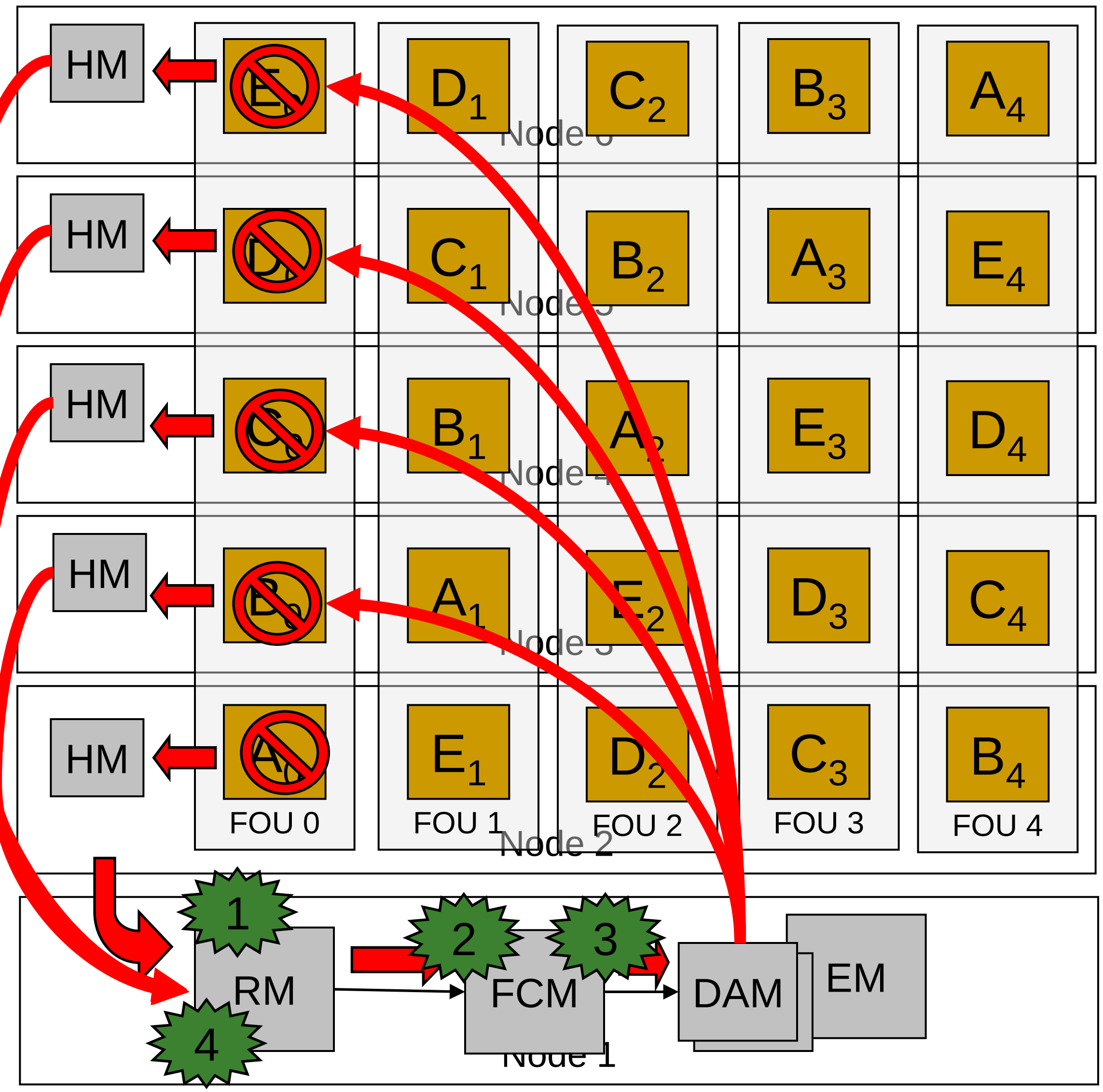


Experimental Evaluation

Experiment 2:

Fail-Stop shutdown latency

- Five Failover Units on Five Nodes
- Use ReplicationManager as point of measurement for 'failure roundtrip'
- Measure time between detection of initial failure & shutdown of components in the same failover unit.



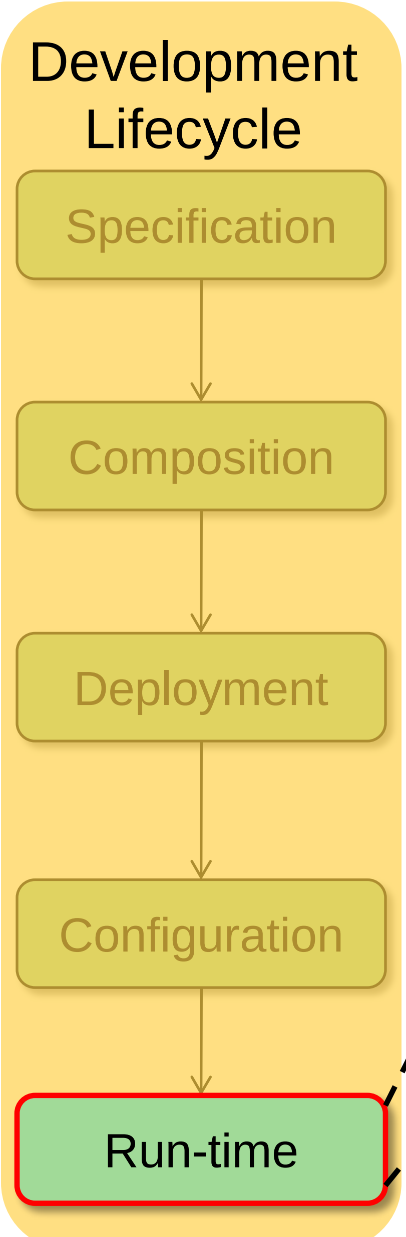
t_4-t_1 = $t_{roundtrip}$ ~70ms

t_3-t_2 = $t_{shutdown}$ ~56ms

Presentation Road Map

- Technology Context: DRE Systems
- DRE System Lifecycle & FT-RT Challenges
- Design-time Solutions
- Deployment & Configuration-time Solutions
- Runtime Solutions
- **Ongoing Work**
- Concluding Remarks

Ongoing Work (1): Tunable State Consistency



- TACOMA Adaptive State Consistency Middleware
  - Tune frequency of update and number of replicas with which state is made consistent

Related Research

Category	Related Research
Optimizations in Real-time Systems	H. Zou et. al., *A Real-time Primary Backup Replication Service*, in IEEE Transactions on Parallel & Distributed Systems (IEEE TPDS), 1999 S. Krishnamurthy et. al., *An Adaptive Quality of Service Aware Middleware for Replicated Services*, in IEEE Transactions on Parallel & Distributed Systems (IEEE TPDS), 2003 T. Dumitras et. al., *Architecting & Implementing Versatile Dependability*, in Architecting Dependable Systems Vol. III, 2005
Optimizations in Distributed Systems	T. Marian et. al., *A Scalable Services Architecture*, in Proceedings of the IEEE Symposium on Reliable Distributed Systems (SRDS 2006), Leeds, UK, 2006 Z. Cai et. al., *Utility-driven Proactive Management of Availability in Enterprise-scale Information Flows*, In Proceedings of the ACM/IFIP/USENIX Middleware Conference (Middleware 2006), Melbourne, Australia, November 2006 X. Zhang et. al., *Customizable Service State Durability for Service-Oriented Architectures*, In Proceedings of the 6^{th} European Dependable Computing Conference (EDCC 2006), Portugal, 2006
Optimizations in Real-time Databases	M. Xiong et. al., *A Deferrable Scheduling Algorithm for Real-time Transactions Maintaining Data Freshness*, in Proceedings of the IEEE International Real-time Systems Symposium (RTSS 2005), Lisbon, 2005 T. Gustafsson et. al., *Data Management in Real-time Systems: A Case of On-demand Updates in Vehicle Control Systems*, in Proceedings of the IEEE Real-time Embedded Technology & Applications Symposium (RTAS 2004), Toronto, 2004

Related Research

Category	Related Research
Optimizations in Real-time Systems	H. Zou et. al., *A Real-time Primary Backup Replication Service*, in IEEE Transactions on Parallel & Distributed Systems (IEEE TPDS), 1999 S [illegible] *Aware Middleware for F*[illegible] Distributed Systems ([illegible] T. Dumitras [illegible] *ecting & Implementing Versatile Dependability*, in Architecting De[illegible] Systems Vol. III, 2005
Optimizations in Distributed Systems	T. Marian et. al., *A S*[illegible]*ble Services Architectu*[illegible] Proceedings of the IEEE Symposium on Reliable Distributed Syste[illegible]S 2006), Leeds, UK, 2006 Z. Cai et. al., *Utility-drive*[illegible] *Enterprise-scal*[illegible] Middleware Con[illegible]06 X. Z[illegible] *vice Oriented Arch*[illegible] Computing Con[illegible]
Optimizations in Real-time Databases	M. Xiong et. al., *A Deferrable Scheduling Algorithm for Real-time Transactions Maintaining Data Freshness*, in Proceedings of the IEEE International Real-time Systems Symposium (RTSS 2005), Lisbon, 2005 T. Gustafsson et. al., *Data Management in Real-time Systems: A Case of On-demand Updates in Vehicle Control Systems*, in Proceedings of the IEEE Real-time Embedded Technology & Applications Symposium (RTAS 2004), Toronto, 2004

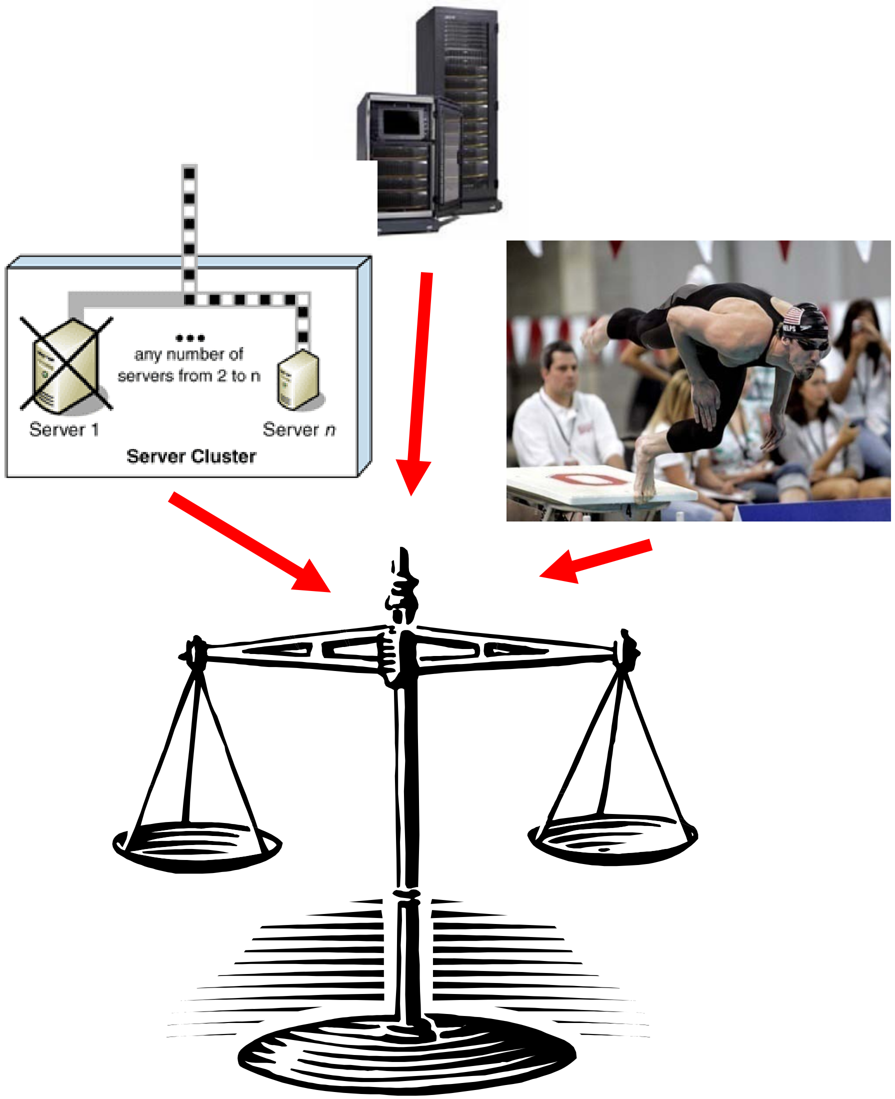
resource optimizations – lazy update propagation, where to store state? database or process?

resource opt[illegible] active replica[illegible] available resources to schedule updates, change of replication styles

schedule lazy updates based on data values

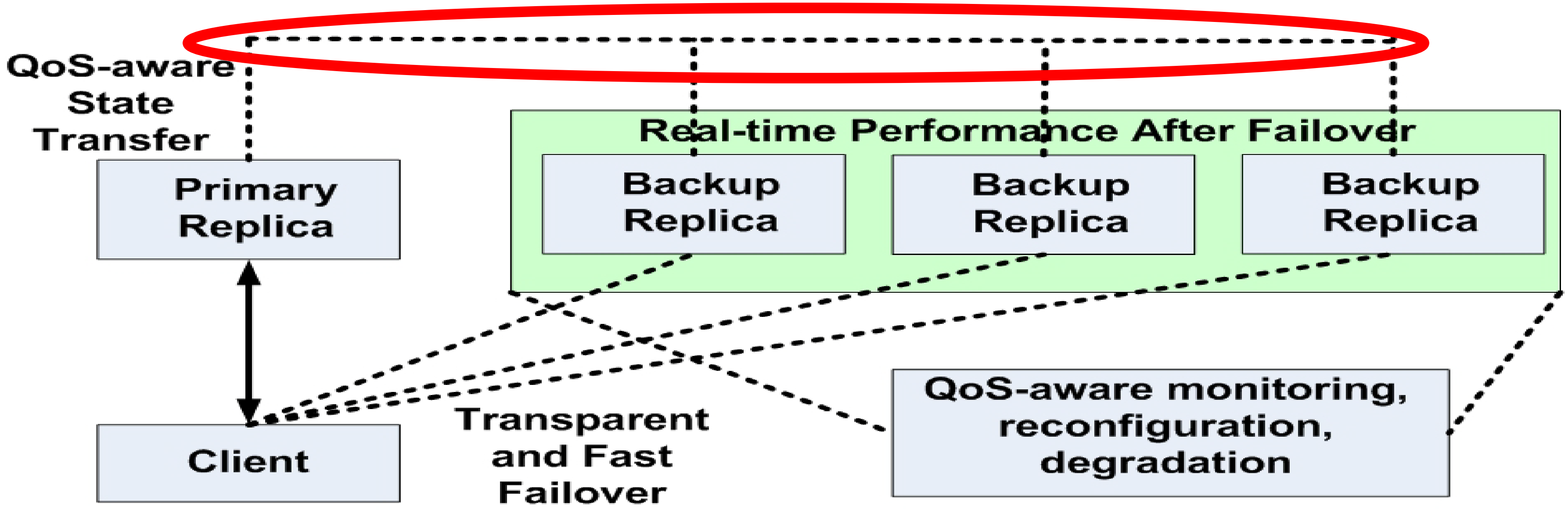
Related Research: What is Missing?

- Optimizations related to replication management restricted to tuning & optimizing frequency of checkpoints
  - lack of optimizations related to tuning & optimizing the depth of consistency
    - number of replicas that are made consistent with the primary replica - more time spent if more replicas are synchronized
  - lack of offline analysis of the operating region
    - e.g., if performance needs to be optimized, how much FT can be provided? (vice-versa for FT)
  - lack of adaptive and configurable middleware architectures to tune optimizations related to consistency depth



Need middleware architecture & optimization algorithms to optimize resource usage related to managing replica consistency

Missing Capabilities in Our Prior Work

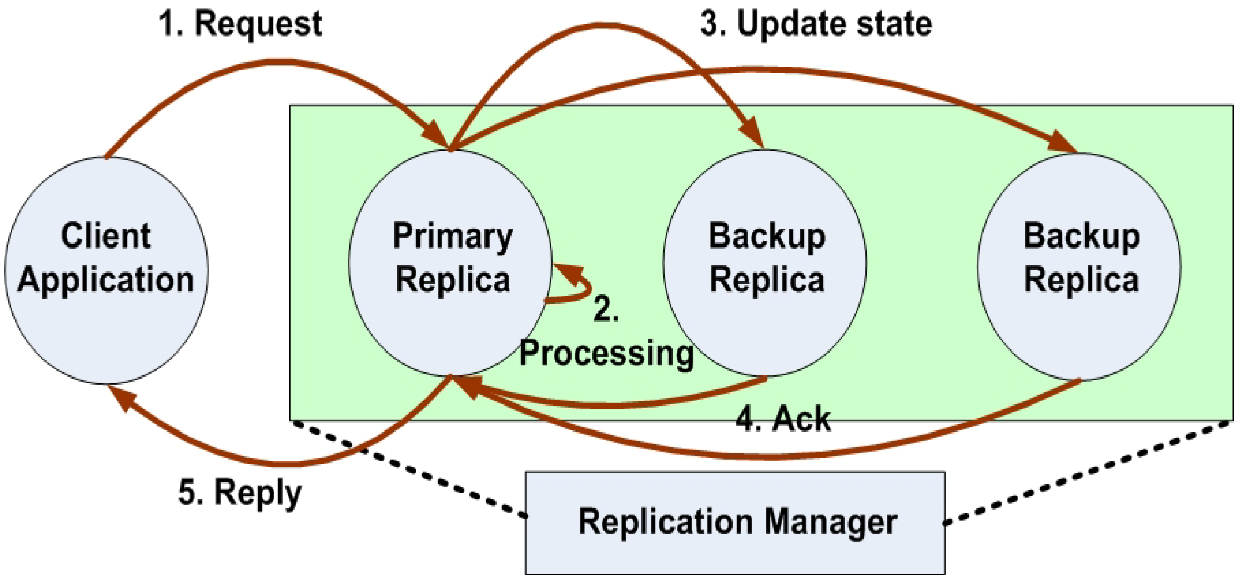


- Performance versus Fault-tolerance – optimize resource usage
  - Need for configurable application consistency management
    - support for range of consistency assurances – weak to strong
  - Need for analyzing & selecting trade-offs among FT & performance
    - resource usage for FT versus resource usage for performance
  - Need for multi-modal operations – degraded levels of FT & performance
    - dynamic adaptations to system loads & failures

Current Work: Resource-aware Replica Consistency Management

Replica & State Management in Passive Replication

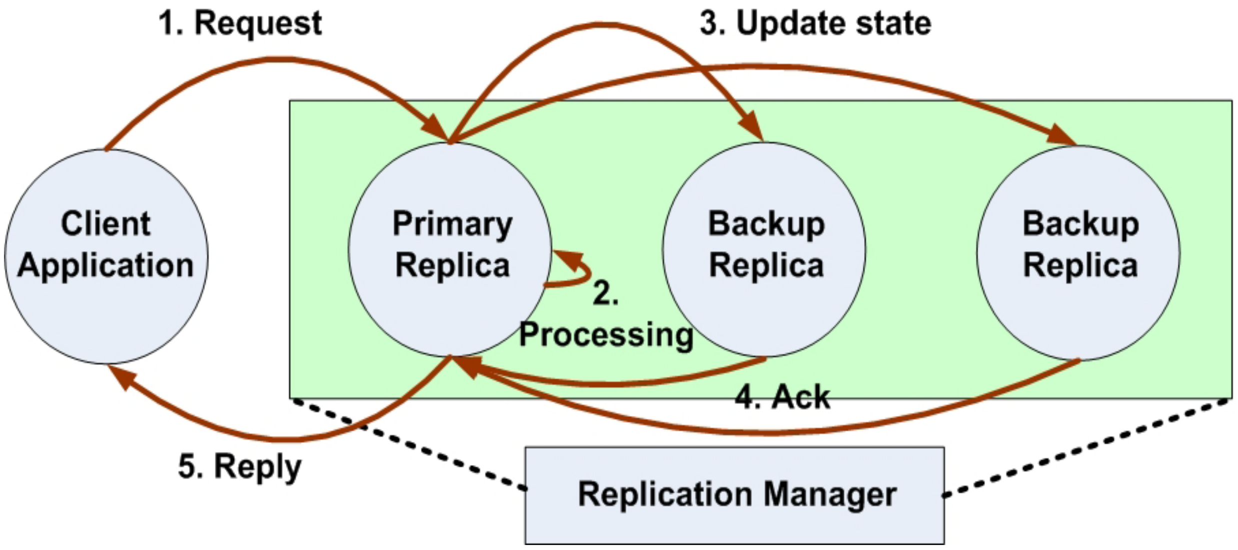
- Replica Management
  - synchronizing the state of the primary replicas with the state of the backup replicas



- Resource consumption trade-offs
  - performance (response times) versus fault-tolerance
  - e.g., if goal is better performance => lesser resources for state management => lesser levels of FT
  - e.g., if goal is better fault-tolerance => response time suffers until all replicas are made consistent

Resource consumption for FT affects performance assurances provided to applications & vice versa

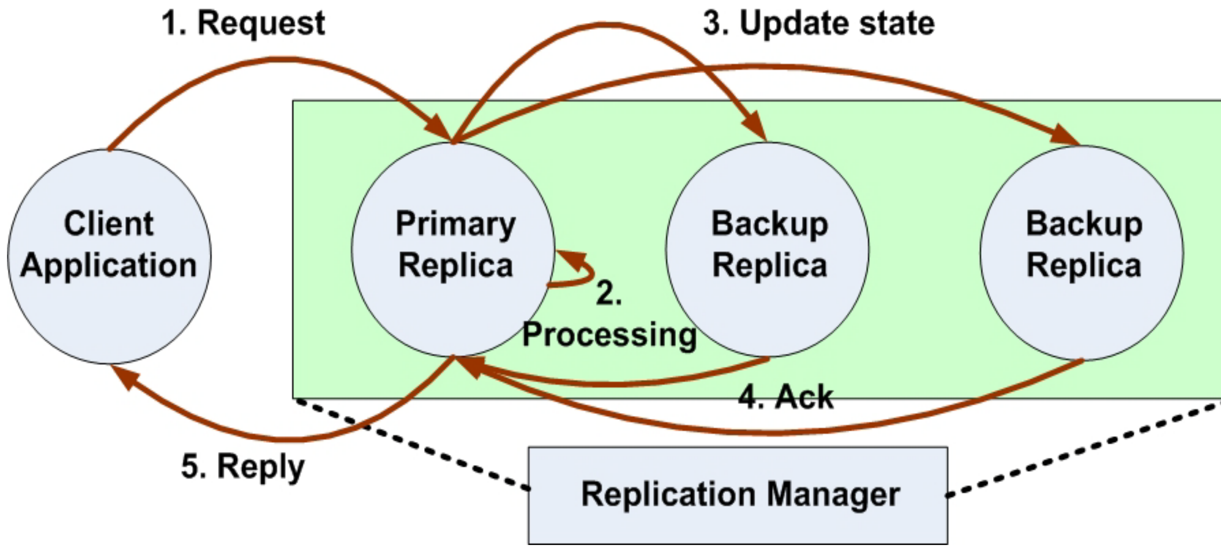
Replica & State Management in Passive Replication



- Diverse application QoS requirements
  - for some applications, FT important
  - for others, performance important
- Need tunable adaptive fault-tolerance
  - cater to the needs of variety of applications
    - no point solutions
  - configurable per-application fault-tolerance properties
    - optimized for desired performance
  - monitor available system resources
    - auto-configure fault-tolerance levels provided for applications

Focus on operating region for FT as opposed to an operating point

Replica & State Management in Passive Replication

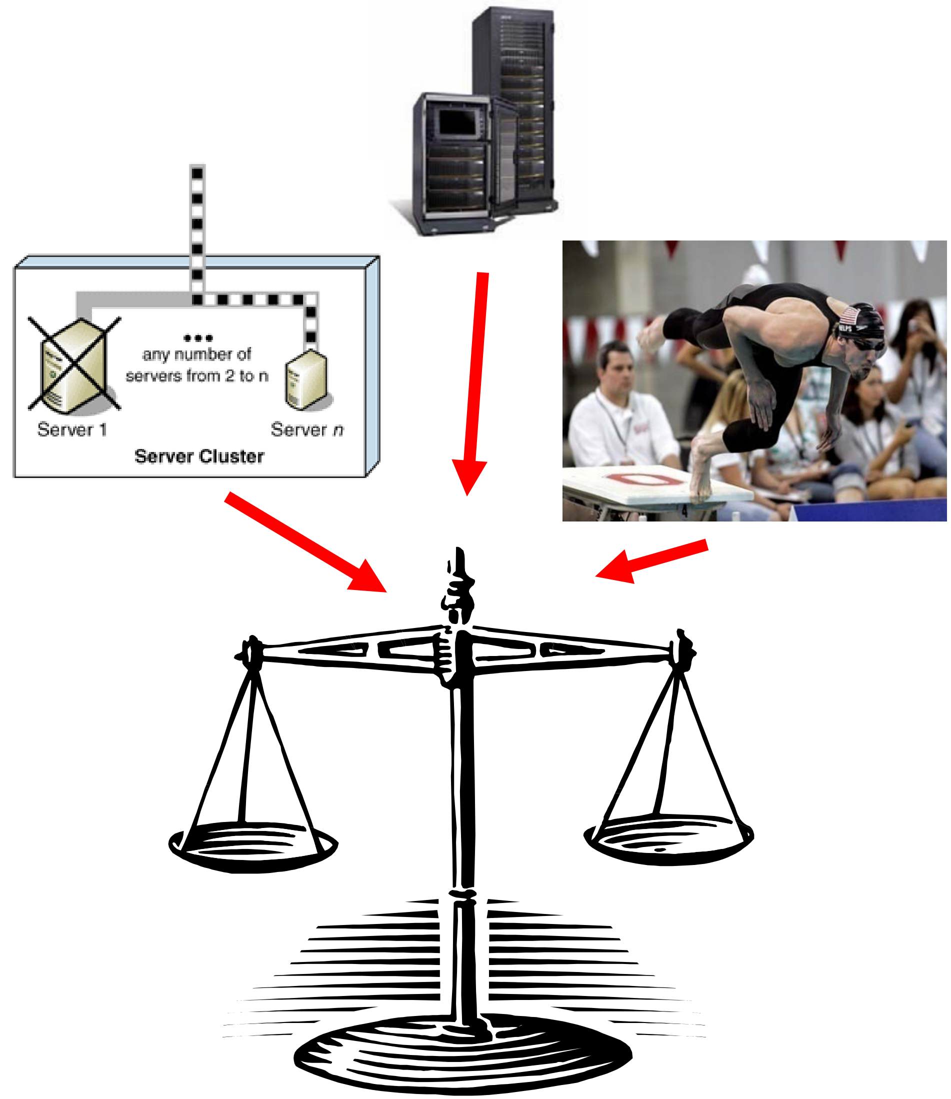


- Diverse application QoS requirements
  - for some applications, FT important
  - for others, performance important
- Need tunable adaptive fault-tolerance
  - input → available system resources
  - control → per-application fault-tolerance properties
  - output → desired application performance/reliability
  - fairness → optimize resource consumption to provide minimum QoS
  - trade-offs needed in resource-constrained environments
    - goal → maximize both performance and fault-tolerance
    - degrade QoS – either of FT or performance – as resource levels decrease

Focus on operating region as opposed to an operating point

Resource Optimizations in Fault-tolerant Systems

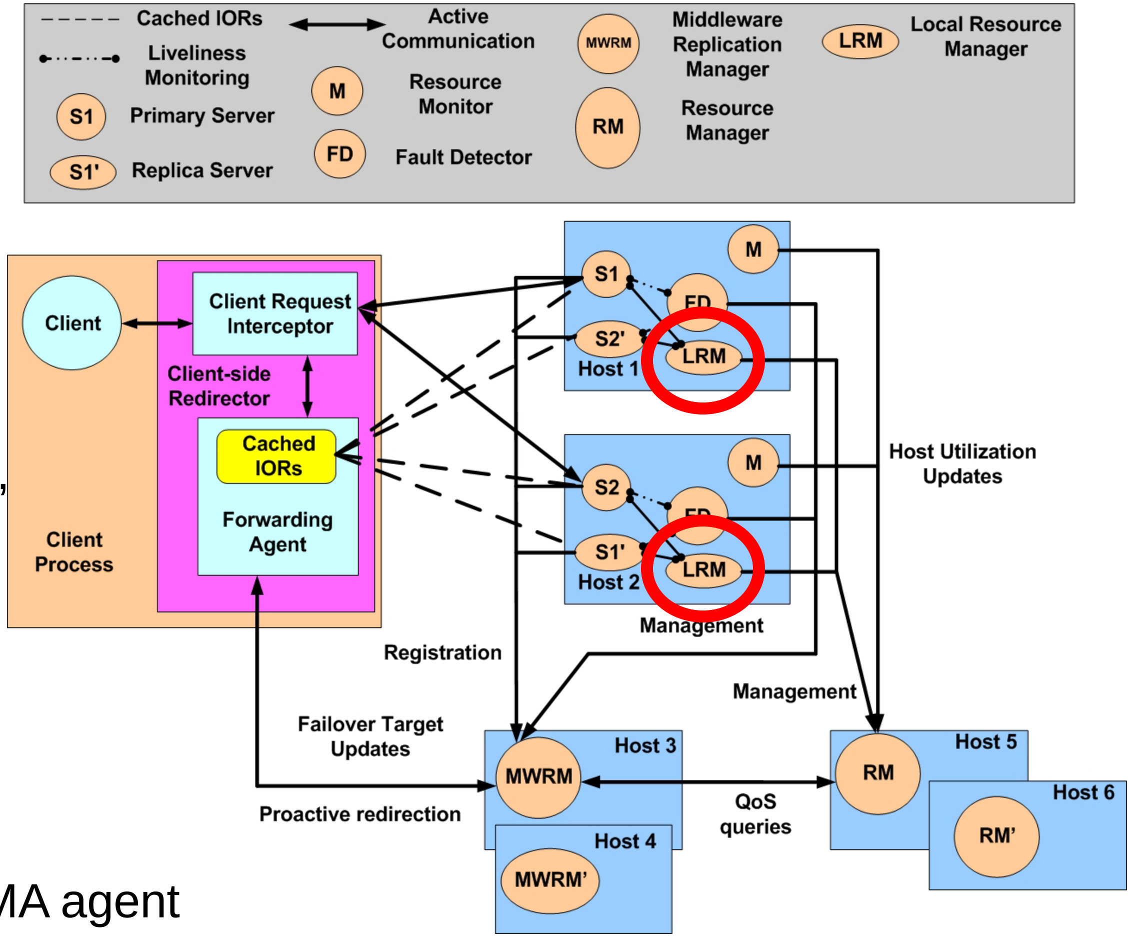
- Different applications have different requirements
  - e.g., FT more important than performance and vice-versa
- Configurable resource consumption needed on per-application basis
- Under resource constraints
  - trade-offs need to be made to balance the use of available resources for
    - fault-tolerance
    - response times



Need mechanisms that can focus on an operating region rather than an operating point to tune state management

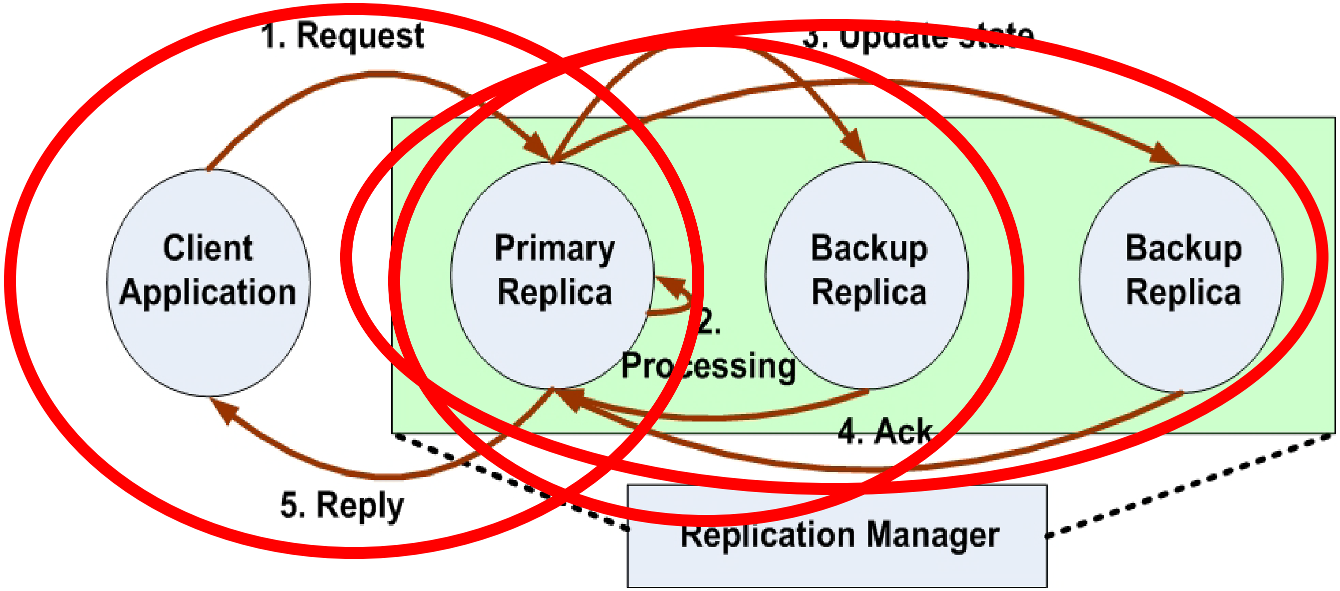
Solution Approach: TACOMA

- Tunable Adaptive COnsistency Management middlewAre (TACOMA)
  - built on top of the FLARe middleware
  - configurable consistency management middleware
    - resource-aware tuning of application consistency – i.e., number of replicas made consistent with the primary replica
    - use of different transports to manage consistency – e.g., CORBA AMI, DDS



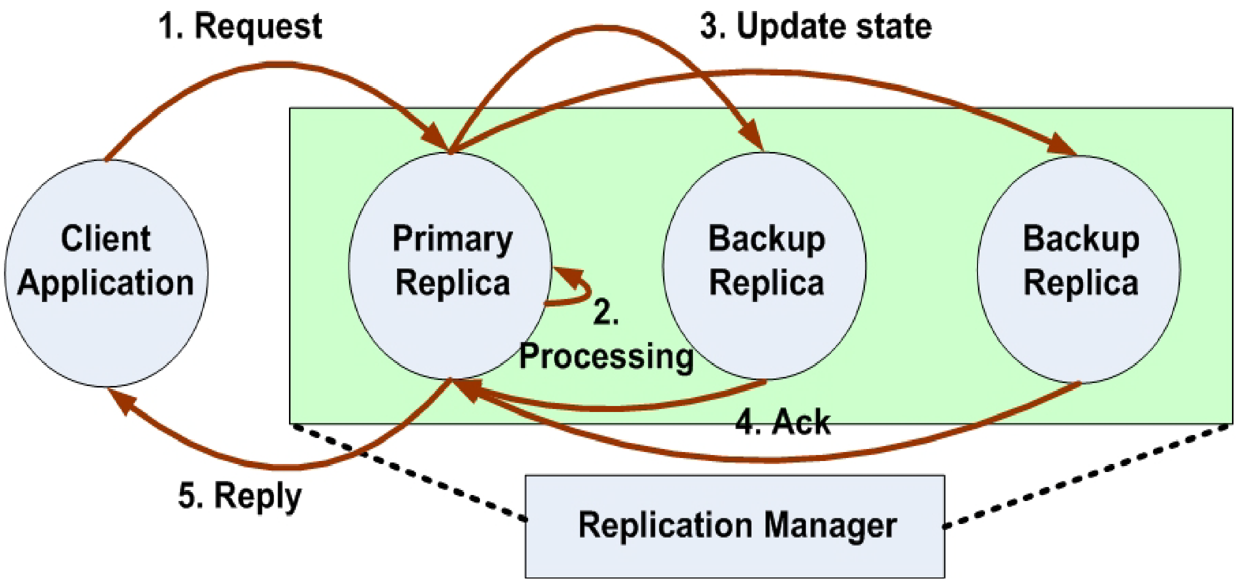
- Local Resource Manager – TACOMA agent
  - added on each processor hosting primary replicas
  - application informs the agent when state changes
  - agents synchronize the state of the backup replicas
    - works with FLARe replication manager to obtain object references

TACOMA: Configurable Consistency Management (1/2)



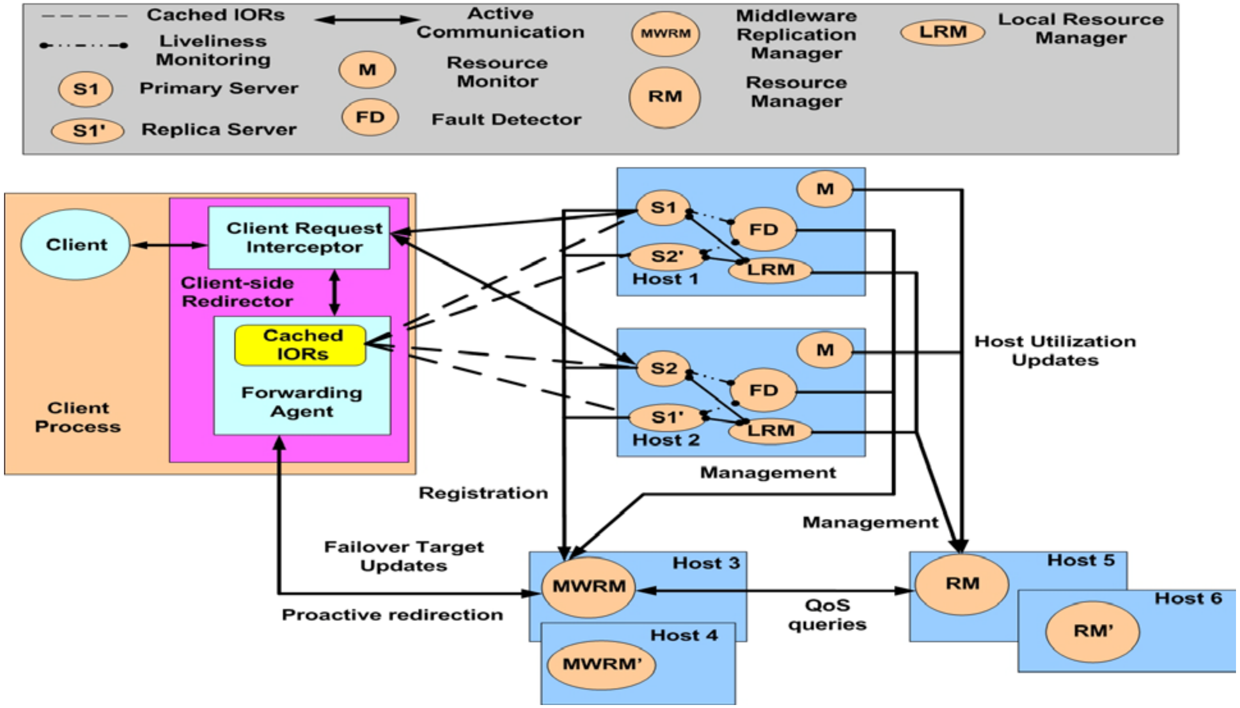
- Determine configurable consistency for each application
  - to respond to a client within a certain deadline, the state of how many backup replicas can be made consistent with the primary replica by the TACOMA agent?
  - Time taken to make one backup replica consistent equals
    - the worst case execution time of an update task initiated by the TACOMA agent in the primary replica
  - Sum of worst case execution times of update tasks at all backup replicas + processing time at primary replica = client response time

TACOMA: Configurable Consistency Management (2/2)



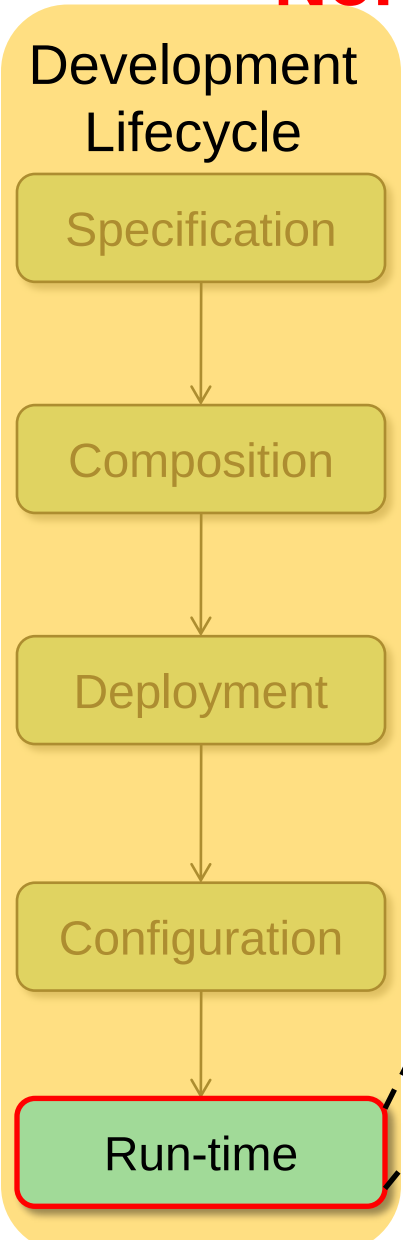
- Determine worst case execution times of update tasks
  - use time-demand analysis
- Tunable consistency management
  - input → available system resources
  - control → per-application consistency depth
  - output → desired application performance/reliability
  - fairness → provide minimum QoS assurances
- Configure TACOMA agents with the consistency depth determined

TACOMA Evaluation Criteria



- **Hypotheses:** TACOMA
  - is customizable & can be applied to a wide range of DRE systems
    - consistency depth range (1 to number of replicas)
  - utilizes available CPU & network resources in the system efficiently, & provides applications with the required QoS (performance or high availability)
    - response times are always met – no deadline misses
  - tunes application replication consistency depth at runtime, as resource availability fluctuates
    - consistency depth decreases from MAX (number of replicas) to MIN (1)

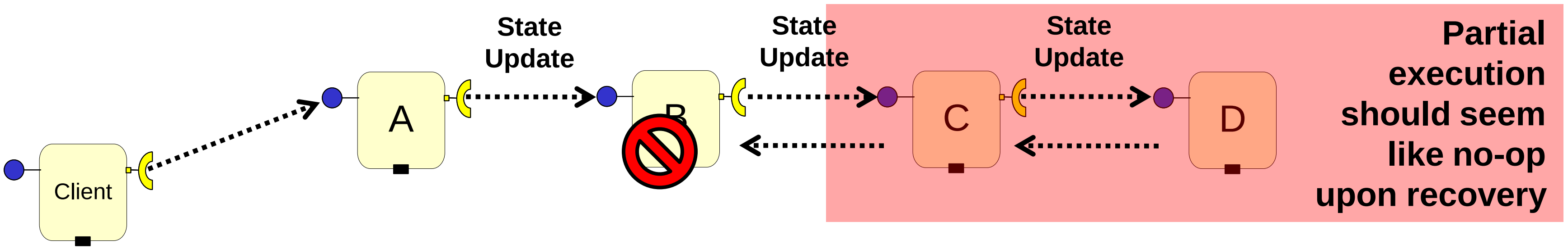
Ongoing Work (2): End-to-end Reliability of Non-deterministic Stateful Components



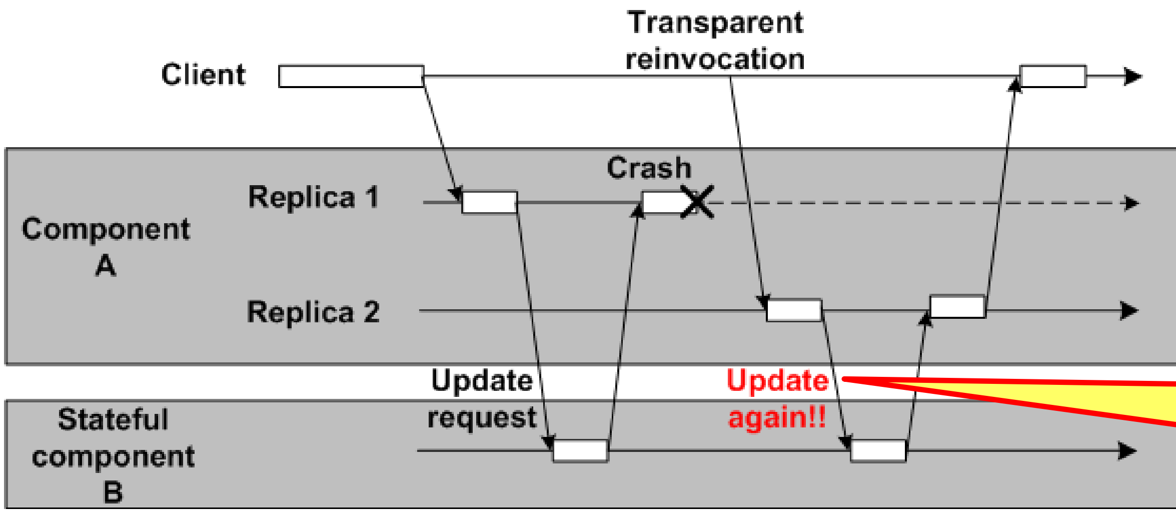
- End-to-end Reliability of Non-deterministic Stateful Components
  - Address the orphan state problem

Execution Semantics & High Availability

- Execution semantics in distributed systems
  - **May-be** – No more than once, not all subcomponents may execute
  - **At-most-once** – No more than once, **all-or-none** of the subcomponents will be executed (*e.g.,* Transactions)
    - Transaction abort decisions are not transparent
  - **At-least-once** – All or some subcomponents may execute more than once
    - Applicable to **idempotent** requests only
  - **Exactly-once** – All subcomponents execute once & once only
    - Enhances perceived availability of the system
- Exactly-once semantics should hold even upon failures
  - Equivalent to single fault-free execution
  - Roll-forward recovery (replication) may violate exactly-once semantics
    - Side-effects of replication must be rectified

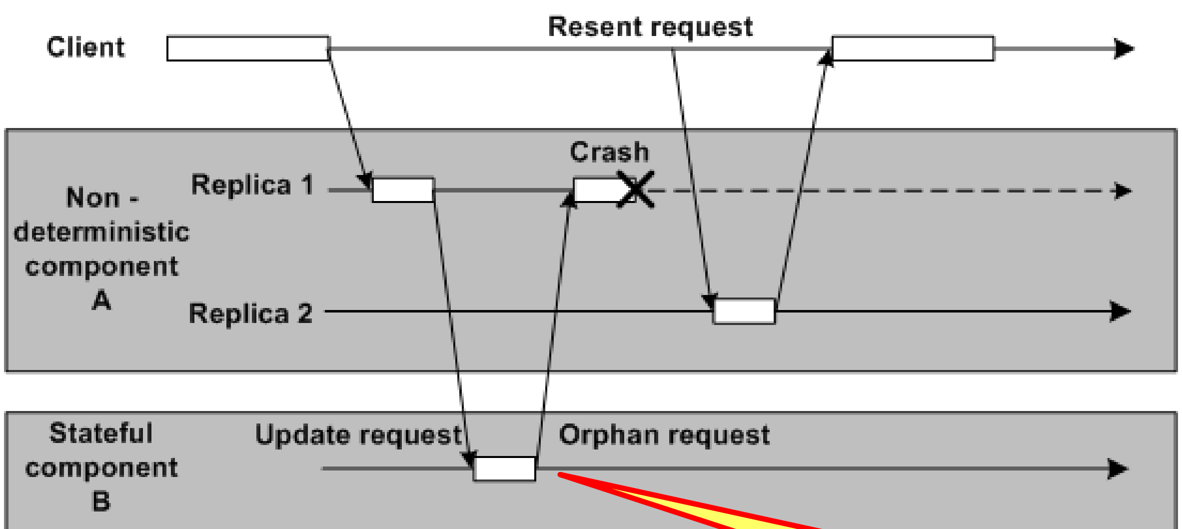


Exactly-once Semantics, Failures, & Determinism



- Deterministic component A
  - Caching of request/reply at component B is sufficient

Caching of request/reply rectifies the problem



Orphan request & orphan state

- Non-deterministic component A
- Two possibilities upon failover
  1. No invocation
  2. Different invocation
- Caching of request/reply does not help
  - Non-deterministic code must re-execute

Related Research: End-to-end Reliability

Category	Related Research (QoS & FT Modeling)
Integrated transaction & replication	1. *Reconciling Replication & Transactions for the End-to-End Reliability of CORBA Applications* by P. Felber & P. Narasimhan 2. *Transactional Exactly-Once* by S. Frølund & R. Guerraoui 3. *ITRA: Inter-Tier Relationship Architecture for End-to-end QoS* by E. Dekel & G. Goft 4. *Preventing orphan requests in the context of replicated invocation* by Stefan Pleisch & Arnas Kupsys & Andre Schiper 5. *Preventing orphan requests by integrating replication & transactions* by H. Kolltveit & S. olaf Hvasshovd
Enforcing determinism	1. *Using Program Analysis to Identify & Compensate for Nondeterminism in Fault-Tolerant, Replicated Systems* by J. Slember & P. Narasimhan *Living with nondeterminism in replicated middleware applications* by J. Slember & P. Narasimhan 3. *Deterministic Scheduling for Transactional Multithreaded Replicas* by R. Jimenez-peris, M. Patino-Martínez, S. Arevalo, & J. Carlos 4. *A Preemptive Deterministic Scheduling Algorithm for Multithreaded Replicas* by C. Basile, Z. Kalbarczyk, & R. Iyer 5. *Replica Determinism in Fault-Tolerant Real-Time Systems* by S. Poledna 6. *Protocols for End-to-End Reliability in Multi-Tier Systems* by P. Romano

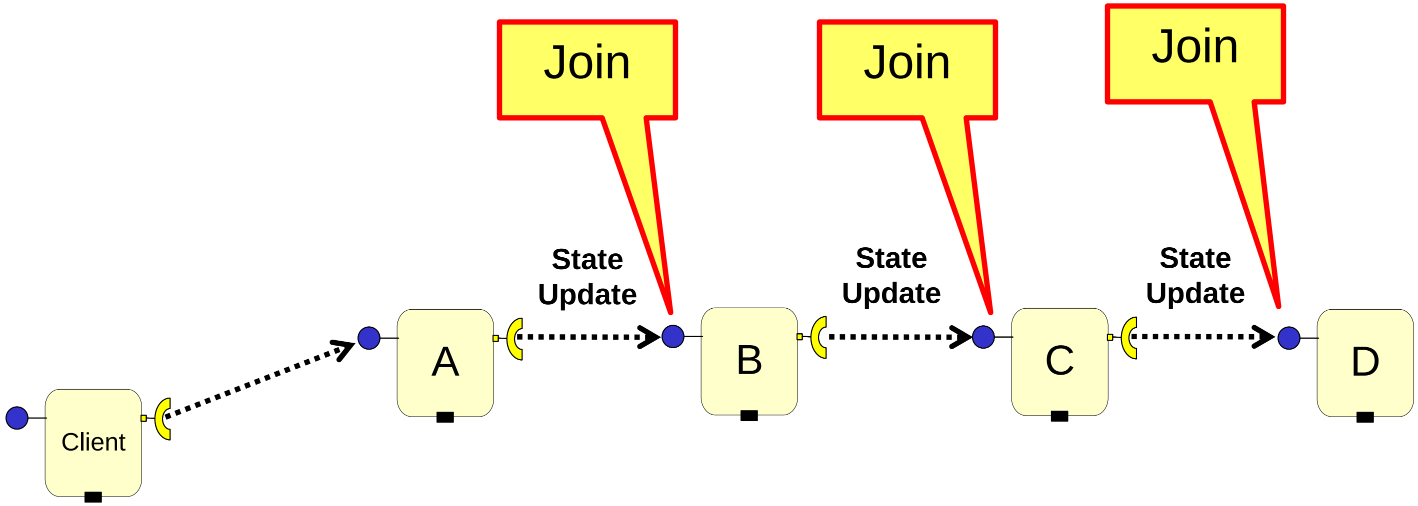
Database in the last tier

Deterministic scheduling

Program analysis to compensate nondeterminism

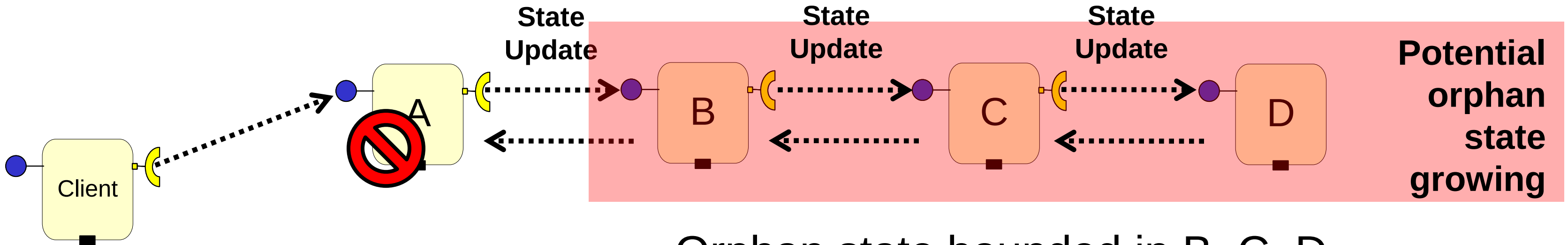
Unresolved Challenges: End-to-end Reliability of Non-deterministic Stateful Components

- Integration of replication & transactions
  - Applicable to multi-tier transactional web-based systems only
  - Overhead of transactions (fault-free situation)
    - *Join* operations in the critical path
    - 2 phase commit (2PC) protocol at the end of invocation



Unresolved Challenges: End-to-end Reliability of Non-deterministic Stateful Components

- Integration of replication & transactions
  - Applicable to multi-tier transactional web-based systems only
  - Overhead of transactions (fault-free situation)
    - *Join* operations in the critical path
    - 2 phase commit (2PC) protocol at the end of invocation
  - Overhead of transactions (faulty situation)
    - Must rollback to avoid orphan state
    - Re-execute & 2PC again upon recovery
  - Complex tangling of QoS: Schedulability & Reliability
    - Schedulability of *rollbacks* & *join* must be ensured
  - Transactional semantics are not transparent
    - Developers must implement: *prepare, commit*, *rollback (2PC phases)*



Orphan state bounded in B, C, D

Unresolved Challenges: End-to-end Reliability of Non-deterministic Stateful Components

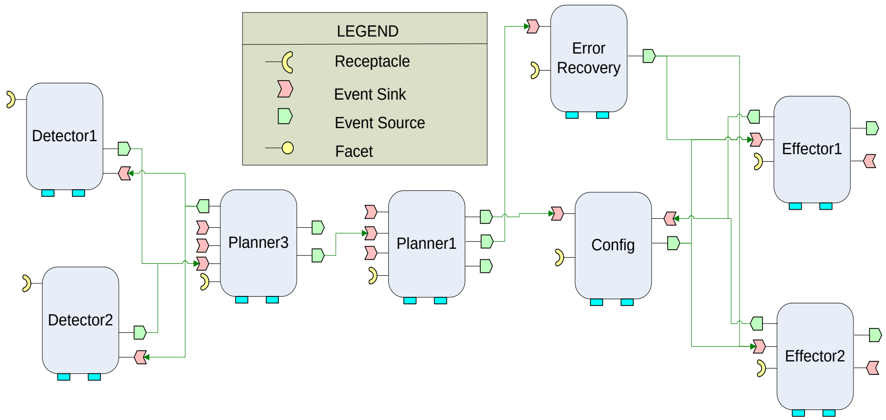
- Integration of replication & transactions
  - Applicable to multi-tier transactional web-based systems only
  - Overhead of transactions (fault-free situation)
    - *Join* operations in the critical path
    - 2 phase commit (2PC) protocol at the end of invocation
  - Overhead of transactions (faulty situation)
    - Must rollback to avoid orphan state
    - Re-execute & 2PC again upon recovery
  - Complex tangling of QoS: Schedulability & Reliability
    - Schedulability of *rollbacks* & *join* must be ensured
  - Transactional semantics are not transparent
    - Developers must implement all: *commit*, *rollback, 2PC phases*
- Enforcing determinism
  - Point solutions: Compensate specific sources of non-determinism
    - e.g., thread scheduling, mutual exclusion
  - Compensation using semi-automated program analysis
    - Humans must rectify non-automated compensation

Ongoing Research: Protocol for End-to-end Exactly-once Semantics with Rapid Failover

- Rethinking Transactions
  - Overhead is undesirable in DRE systems
  - Alternative mechanism needed to rectify the orphan state
- Proposed research: A distributed protocol that
  1. Supports exactly-once execution semantics in presence of
     - Nested invocations
     - Non-deterministic stateful components
     - Passive replication
  2. Ensures state consistency of replicas
  3. Does not require intrusive changes to the component implementation
     - No need to implement *prepare, commit,* & *rollback*
  4. Supports fast client failover that is insensitive to
     - Location of failure in the operational string
     - Size of the operational string
- Evaluation Criteria
  - Less communication overhead during fault-free & faulty situations
  - Nearly constant client-perceived failover delay irrespective of the location of the failure

Concluding Remarks

- Operational string is a component-based model of distributed computing focused on end-to-end deadline
- Operational strings need group failover
  - Not provided out-of-the-box in contemporary middleware
- Solution:
  - Component QoS Modeling Language (CQML) for end-to-end QoS specification
    - Failover unit modeling
  - Generative Aspects for Fault-Tolerance (GRAFT) for transparent FT provisioning
    - M2M, M2C, & M2T transformations
- Proposed research: End-to-end reliability of non-deterministic stateful components
  - Protocol to rectify orphan state problem allowing fast failover



Questions