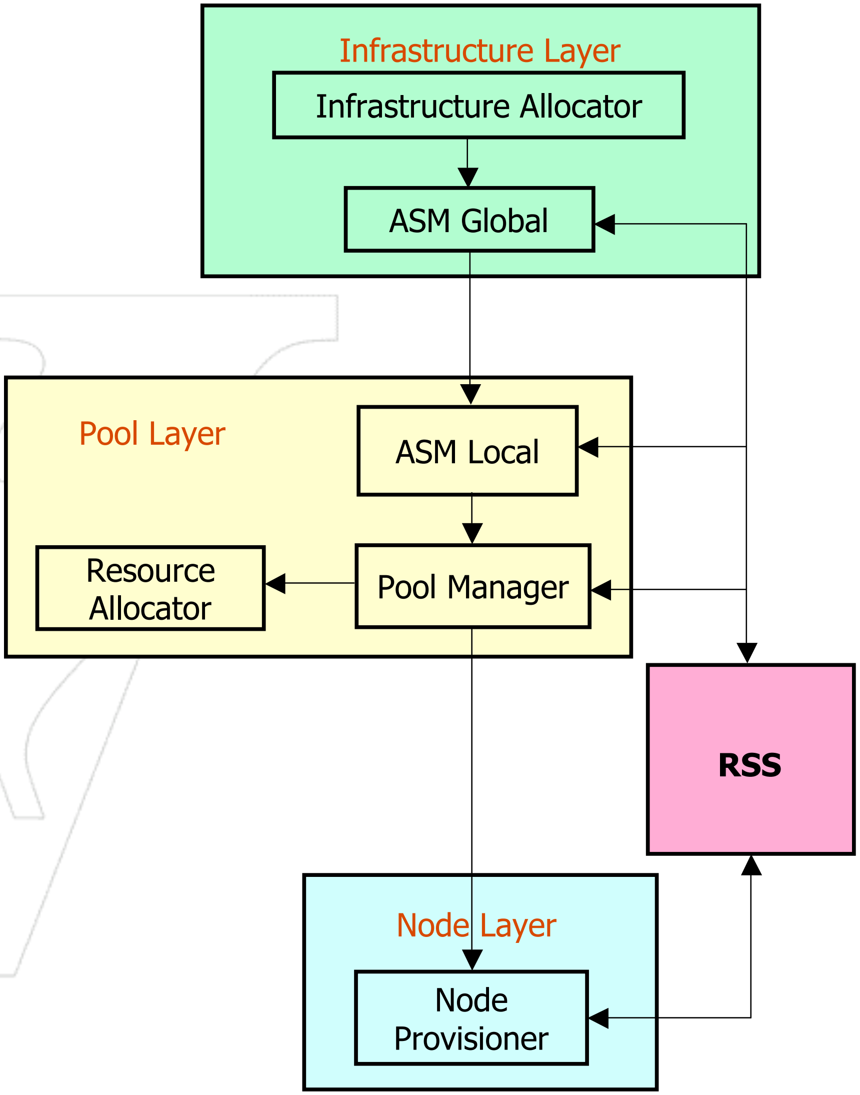
Resource Allocation & Control Engine (RACE)

William R. Otte
Jaiganesh Balasubramanian
Dr. Douglas C. Schmidt
Nishanth Shankaran

Thomas Damiano
Edward Mulholland
Gautam Thaker

Existing ARMS Phase 1 Architecture

- Non standard entities
  - Custom built for ARMS
- Monolithic implementation of each layer
  - Reduced flexibility
  - Increases complexity
- Number of layers in MLRM is fixed to three
  - Infrastructure Layer
  - Pool Layer
  - Node Layer
- Possibly limits scalability
- Roles performed by entities at Infrastructure Layer and Pool Layer are similar, but differ only in "scope"



Overview of Phase II MLRM Needs (1/2)

- ARMS' Primary Goal
  - Develop Adaptive Resource Management technology for DD(X)
- Research Goals
  - Develop *general* purpose Adaptive Resource Management into standardized software services
- Benefits
  - Life of technology is not limited to the lifetime of ARMS program
  - MLRM technology can be reused in other areas research other than ARMS
  - Increases ease of technology transfer to DD(X) if DRM capability is available via standardized service compared to custom application architectures
  - Leverages the latest development in CCM technology, and potentially enhances it
- As DARPA PM Joseph Cross says:
  - "We deliver Technology, and not Software!"
  - "If we can improve the design, lets improve it as now is the time to do so!"

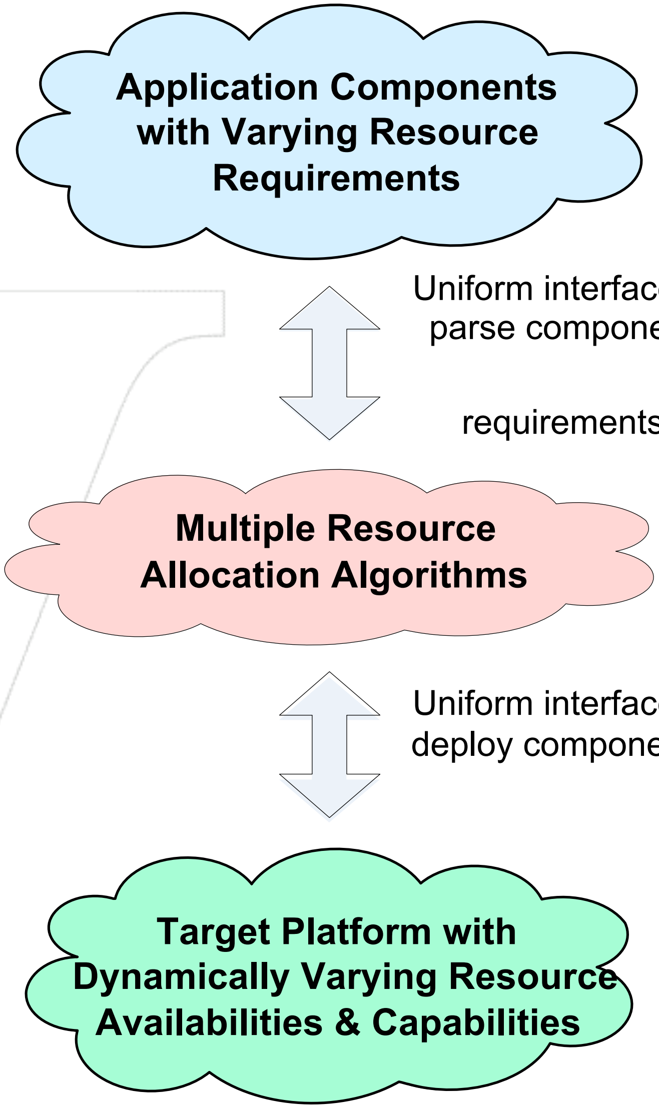
Overview of Phase II MLRM Needs (2/2)

- Limitations with current MLRM infrastructure
  - Software Architecture for Phase I MLRM is brittle and difficult to reuse in contexts outside ARMS
  - Provide custom services for functions where emerging standardized services are available
    - e.g. Node Provisioner is a ARMS ⇔ Node Manager in DnC specification
  - Provides an all or nothing solution
    - Can not reuse individual components such as AMS, IA, or Pool Manager separately out slide the MLRM framework
- Proposed Approach
  - Package ARMS adaptive resource management capabilities into a set of a modular, general purpose, DnC spec compliant / coordinated services
  - Design framework for plugging algorithms specializing in resource allocation and adaptive resource management
  - Provides an abstraction to address the interdependencies between resource allocation and string management in a scalable and configurable manner
  - Is NOT
    - One monolithic software application
    - Algorithms or the core science

Meeting ARMS Phase II MLRM Needs (1/2)

Context

- Deploy application components with
  - Varying resource requirements
  - Varying resource availability/capability
  - Unique properties not shared by traditional CORBA objects
- Need for resource allocation
  - Initially place components
  - Generate configuration for OS level QoS mechanisms
- Need for adaptation
  - Changes in mission goals (modes) as execution progresses
  - Changes in mission goals because tasks cannot be completed on time
  - Degradation in system performance – loss of resources
  - Task execution times & resource requirements may vary dynamically



Application Components with Varying Resource Requirements

Uniform interface to parse component requirements

Multiple Resource Allocation Algorithms

Uniform interface to deploy components

Target Platform with Dynamically Varying Resource Availabilities & Capabilities

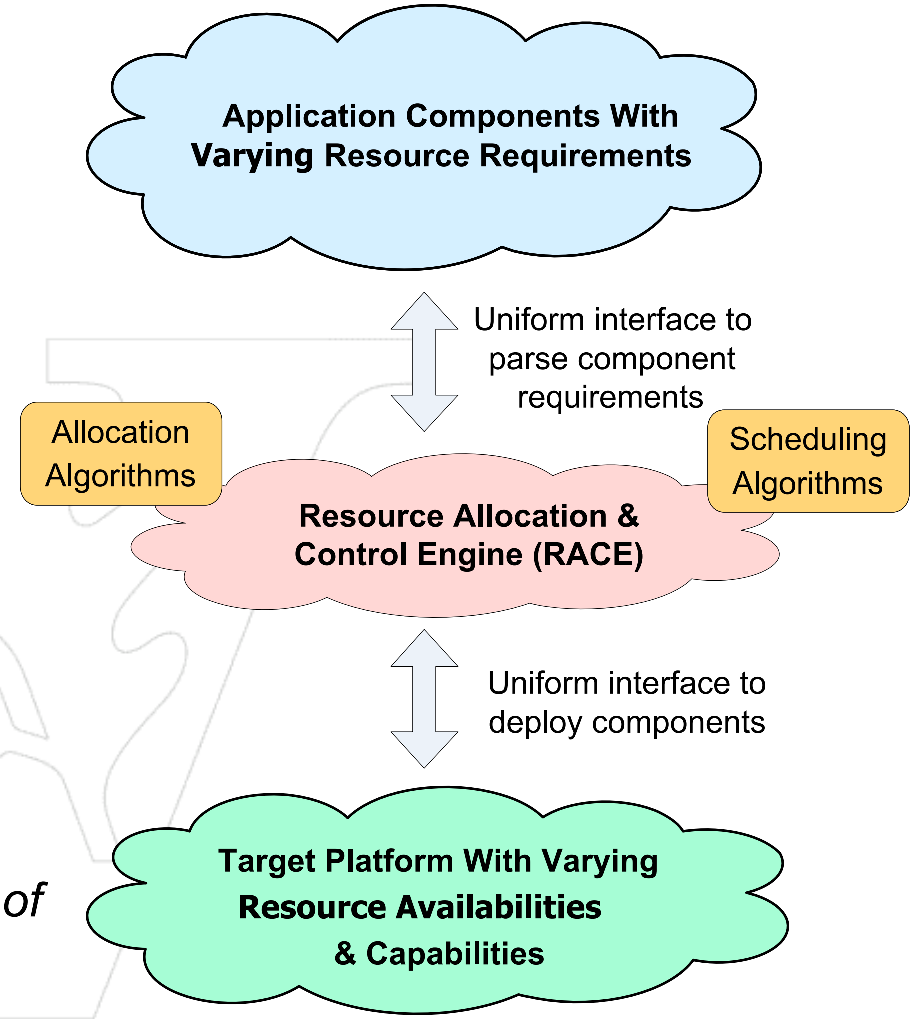
Meeting ARMS Phase II MLRM Needs (2/2)

Problem

- Develop effective allocation & RT scheduling algorithms (policies)
- Implement allocation & scheduling algorithms (mechanisms)
- Tight coupling between policies & mechanisms
  - Monolithic implementations increases memory footprint & complexity

Solution

- Separation of concerns
  - e.g., separate policy & mechanism
- Framework support for plugging in allocation/scheduling algorithms
- Leverage properties of existing adaptive middleware, such as QuO, *in the context of component middleware*

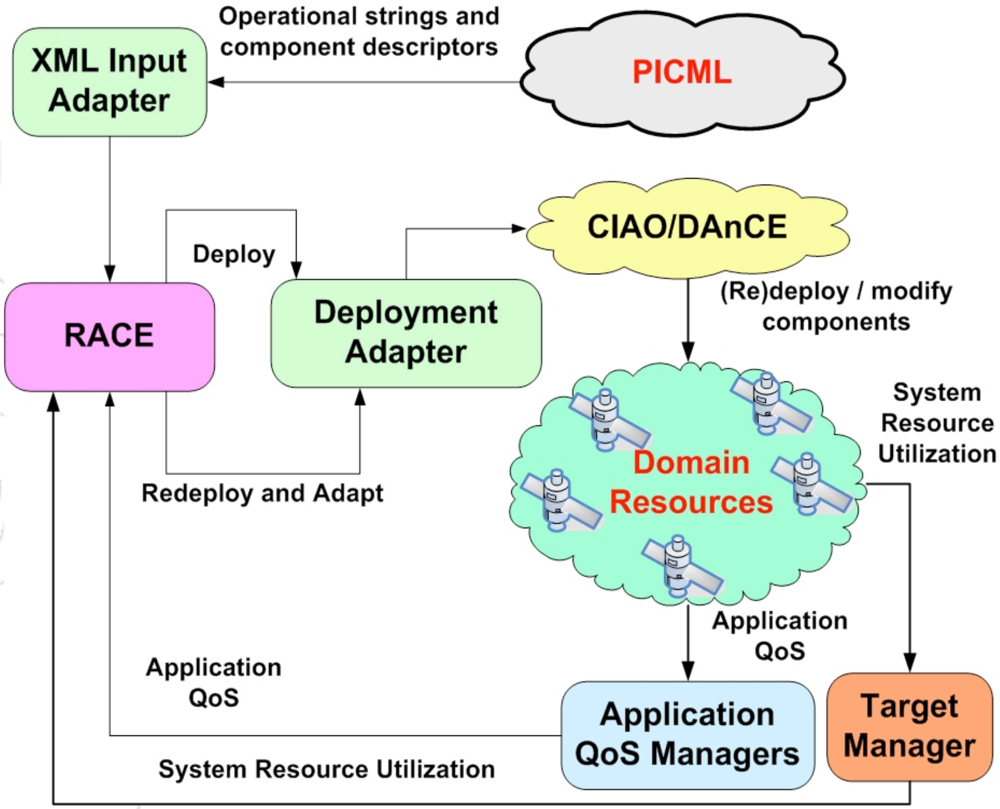

Application Components With Varying Resource Requirements
Uniform interface to parse component requirements
Allocation Algorithms
Scheduling Algorithms
Resource Allocation & Control Engine (RACE)
Uniform interface to deploy components
Target Platform With Varying Resource Availabilities & Capabilities

Resource Allocation & Control Engine (RACE)

RACE Overview

RACE is a component assembly implementing an on-line planner which:

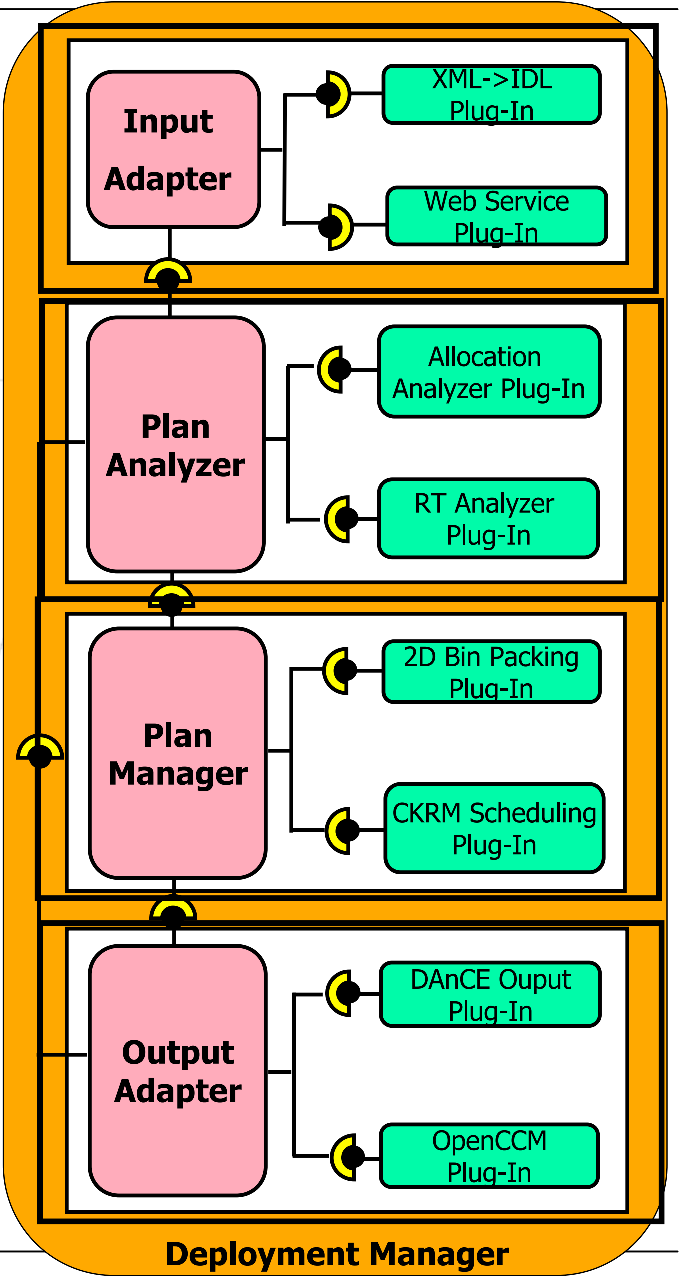
- Accepts input from an off-line planner (such as PICML)
- Examines plan meta-data and selects
  - Allocation and scheduling algorithms, which are encapsulated inside “Planners”
  - A deployment method (ie, DAnCE) for the modified plan
- Monitors current resource utilization and application performance
- Tune Application QoS and potentially redeploy components in response to changes in application performance or the environment.



Detailed RACE Allocation Architecture (1/3)

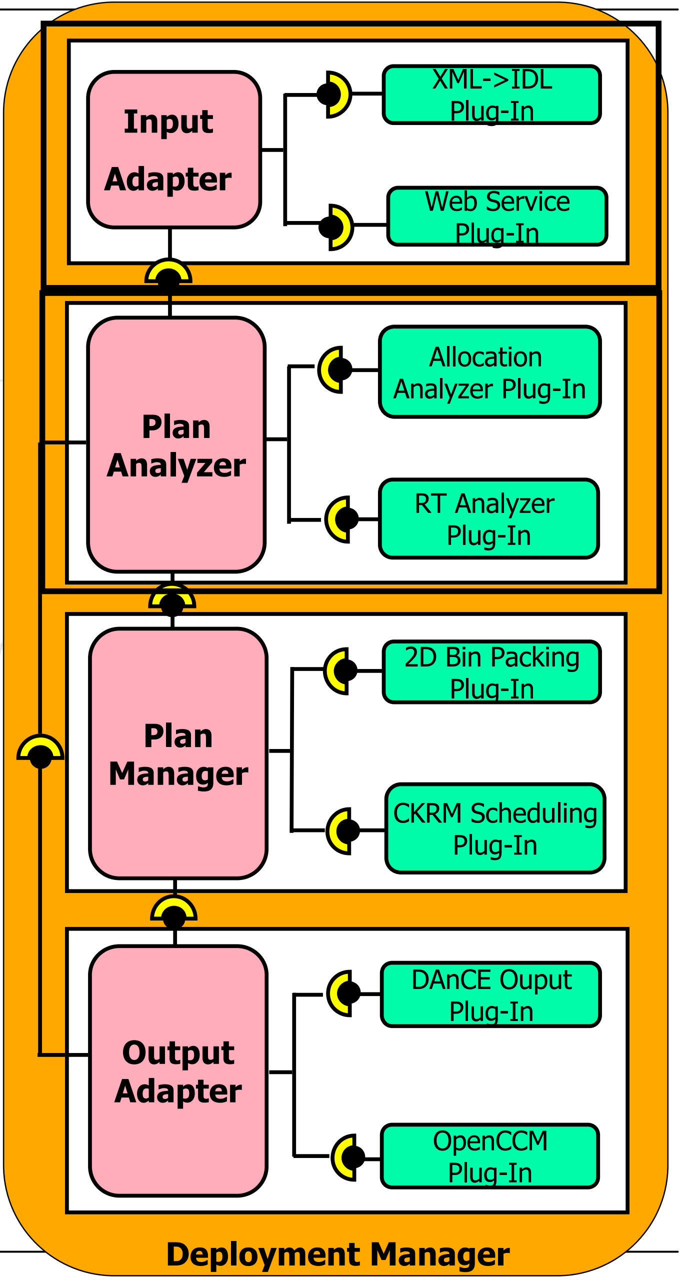
The Planning capability in RACE is implemented by four distinct components:

- The Input Adapters, which are responsible for translating input provided to RACE into IDL data structures
- The Plan Analyzer, which is responsible for examining metadata in the plan and selecting planners to be run on the plan.
- The Planner Manager, which executes the planners selected by the Plan Analyzer and maintains a registry of metadata about installed planners.
- The Output Adapters, which are responsible for translating the provisioned deployment plans into a native format for deployment.



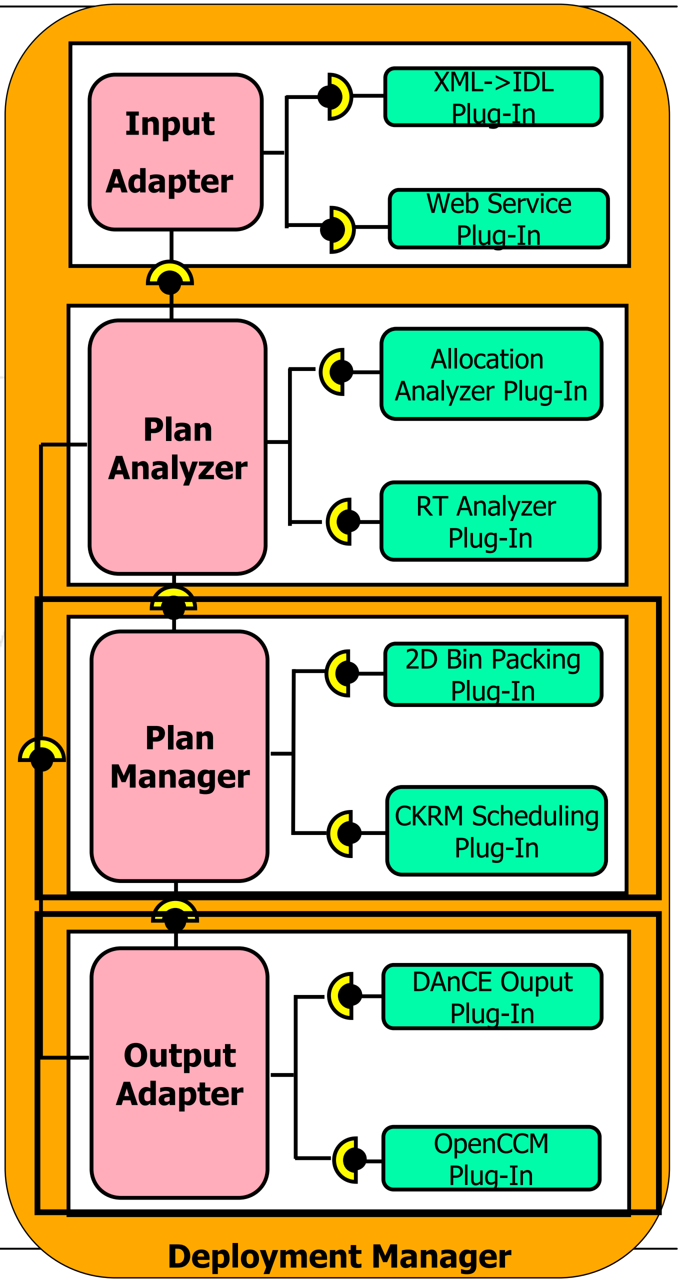
Detailed RACE Allocation Architecture(2/3)

- **Input Adapter**
  - Translates deployment information from some application or platform specific format into IDL data useful to RACE
  - Any number of adapters may plugged into the Input Adapter component.
- **Plan Analyzer**
  - Examines metadata in a plan to determine if a static or dynamic deployment is requested.
    - Dynamic plans are examined by Analyzer plug-ins
    - Static plans are passed directly to the Output Adapters
  - Analyzers are grouped into "classes." Only one analyzer will be chosen per class via a chain of responsibility.

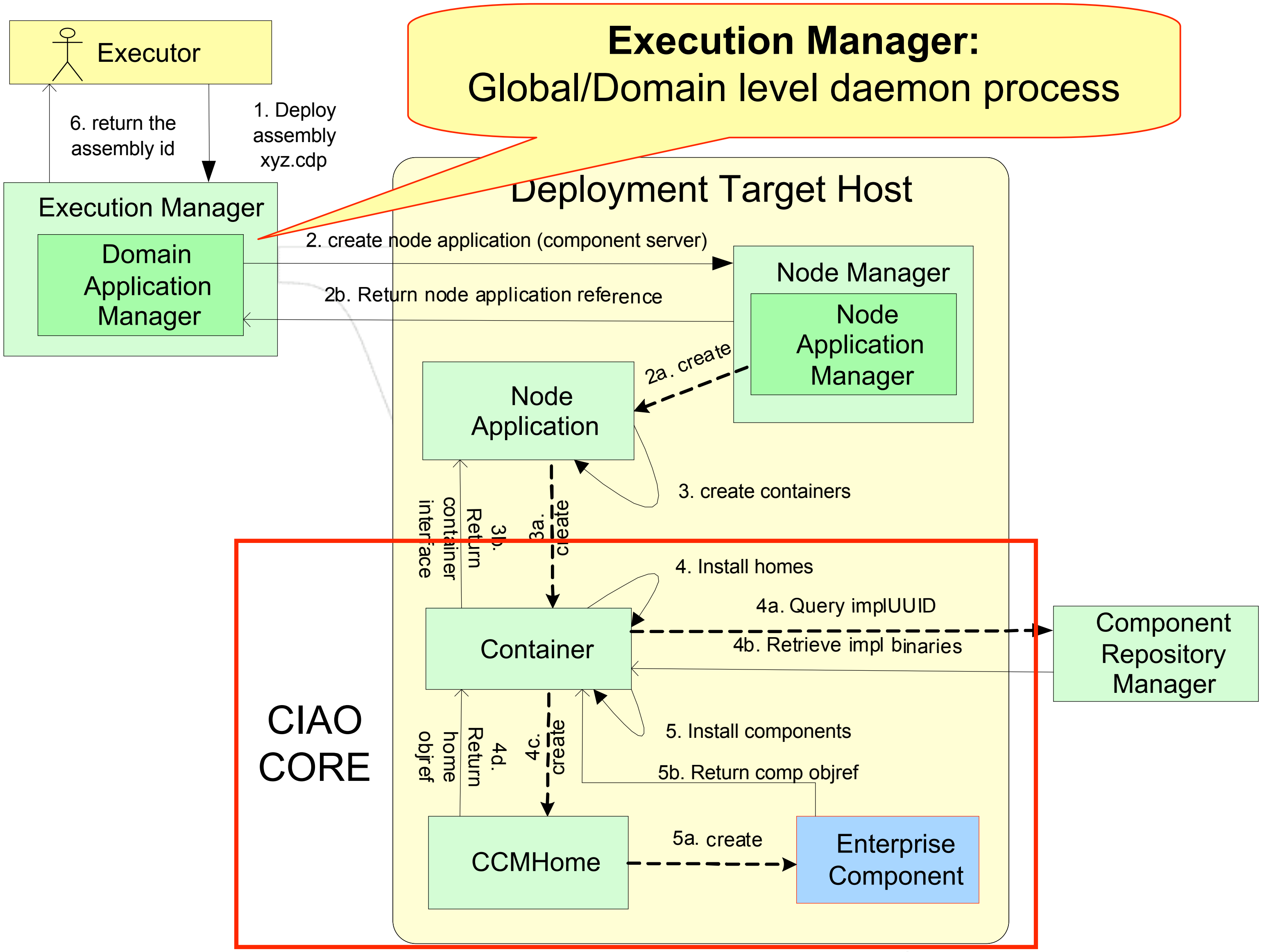


Detailed RACE Allocation Architecture (3/3)

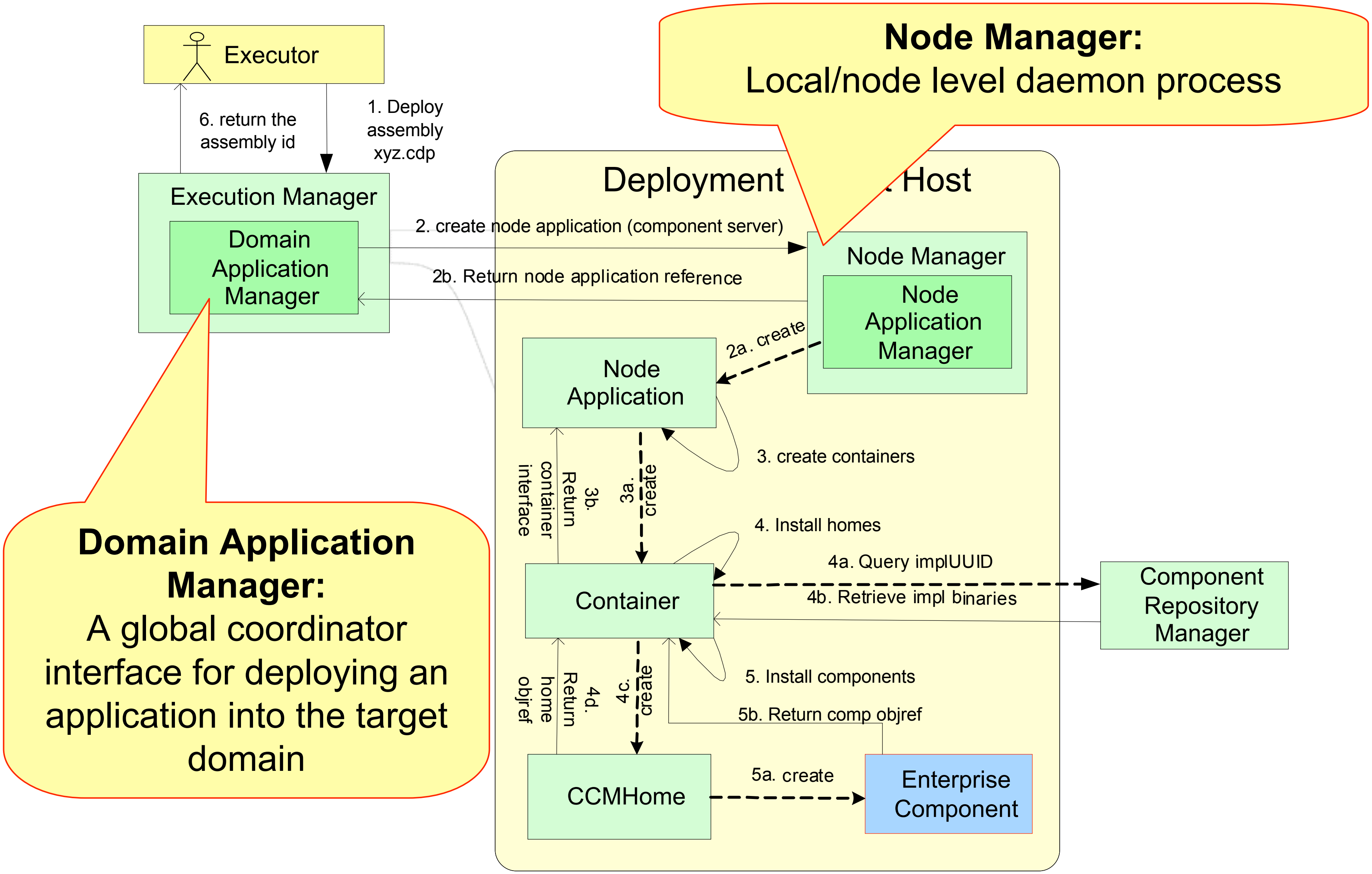
- **Planner Manager**
  - Responsible for executing planning strings provided by the Plan Analyzer
  - Plug-In architecture allows new planners to be updated and added in at run-time
  - The Planner Manager will maintain a catalog of all installed planners so Analyzers may take advantage of new algorithms as they are installed
- **Output Adapters**
  - The Output Adapter will examine metadata present in the plan and select an output adapter required by the plan
  - The Plug-In architecture allows RACE to support multiple CCM toolchains (DAnCE/OpenCCM) or even multiple middlware types (EJB, .NET)



DAnCE In Action (1/4)



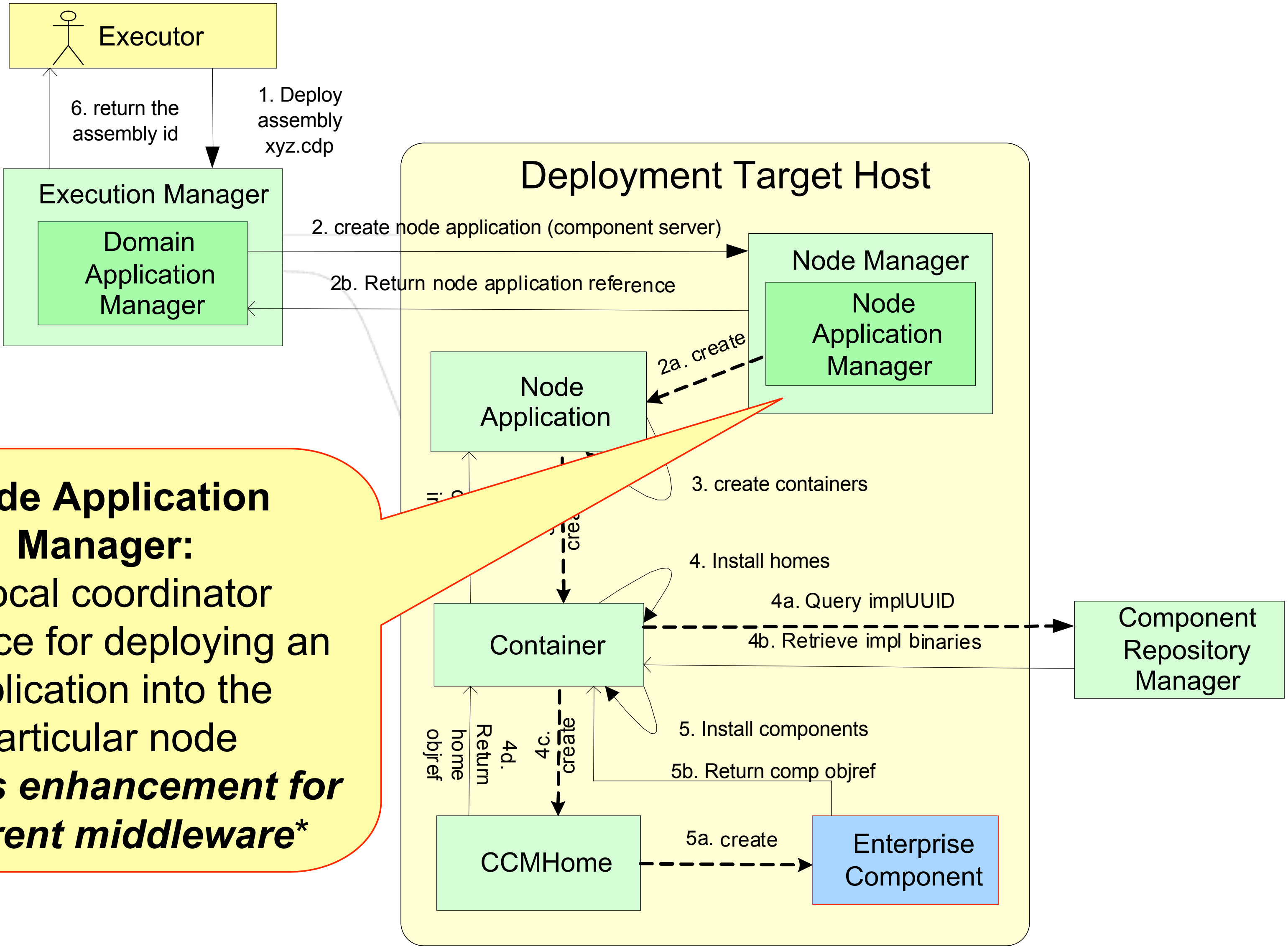
DAnCE In Action (2/4)



Node Manager:
Local/node level daemon process

Domain Application Manager:
A global coordinator interface for deploying an application into the target domain

DAnCE In Action (3/4)

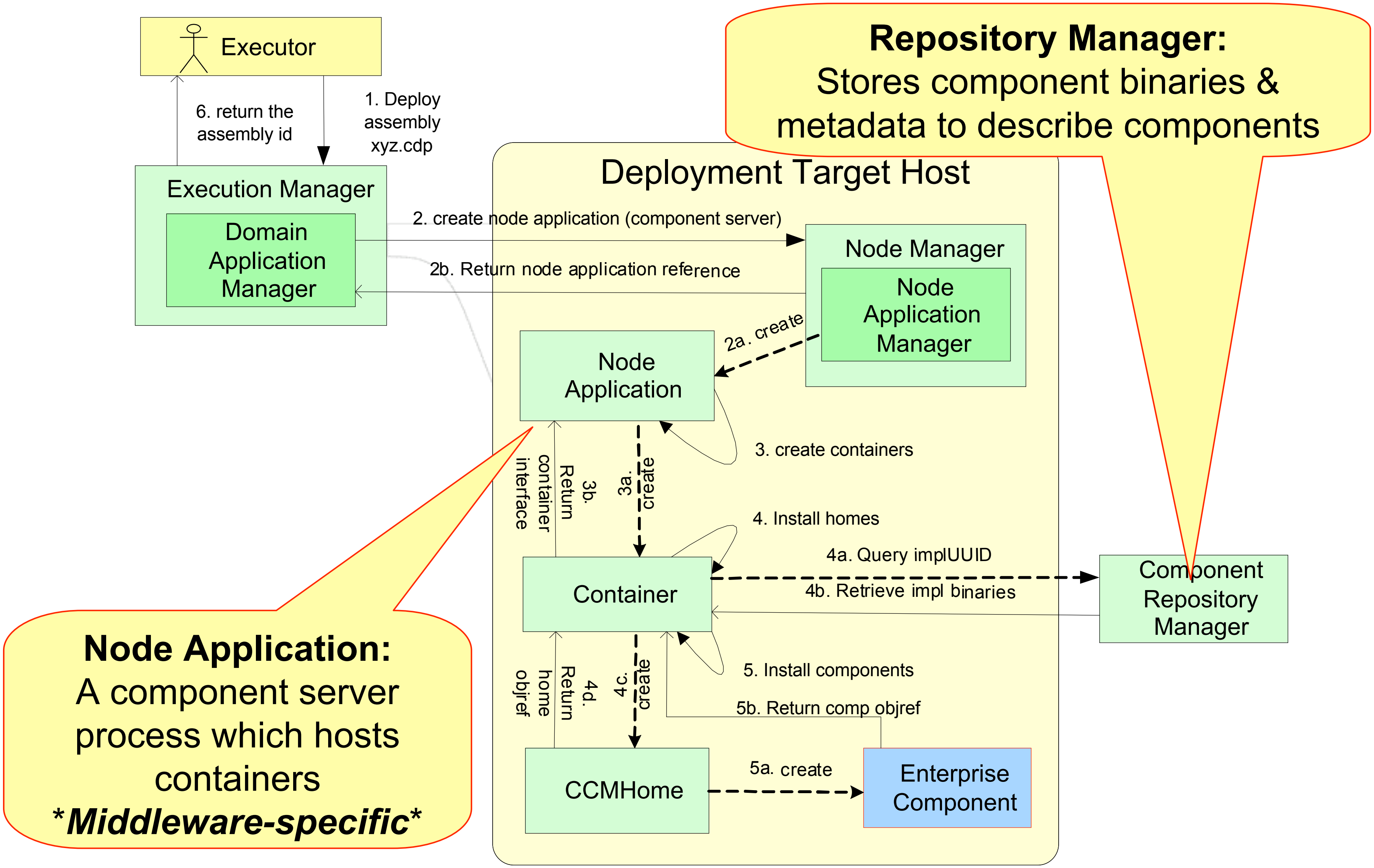


Node Application Manager:

A local coordinator interface for deploying an application into the particular node

**Needs enhancement for different middleware*

DAnCE In Action (4/4)



Repository Manager:
Stores component binaries & metadata to describe components

Node Application:
A component server process which hosts containers
****Middleware-specific****

Current Status & Implementation Plan

Completed

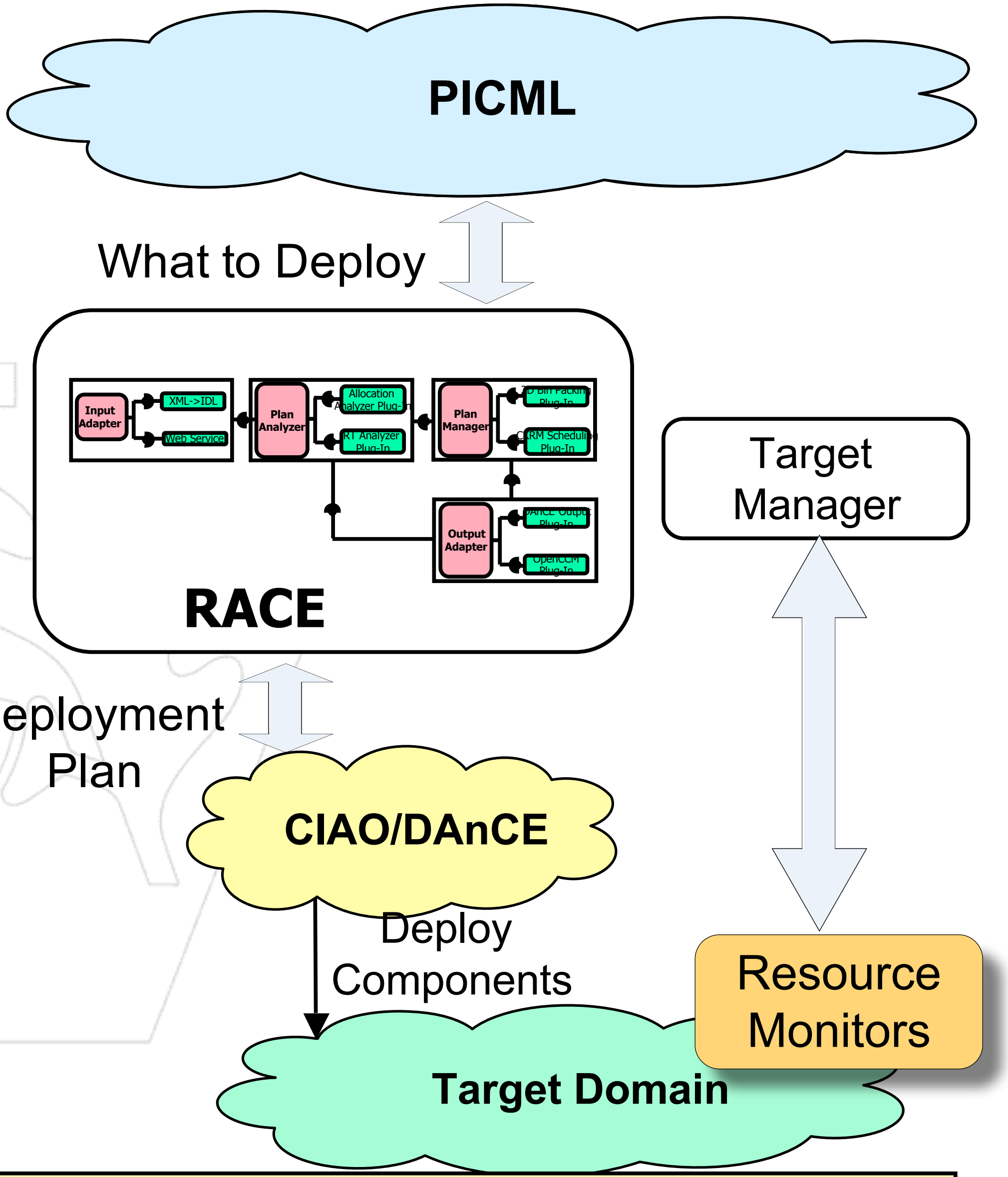
- CIAO/DAnCE infrastructure middleware
- Architecture of RACE
- CCM interfaces of RACE
- Simple bin-packing allocation planner
- Web application front end for selecting and deploying WLGs

In design phase

- Resource Monitors
- Target Manager
- Control aspects of RACE

Plan of action

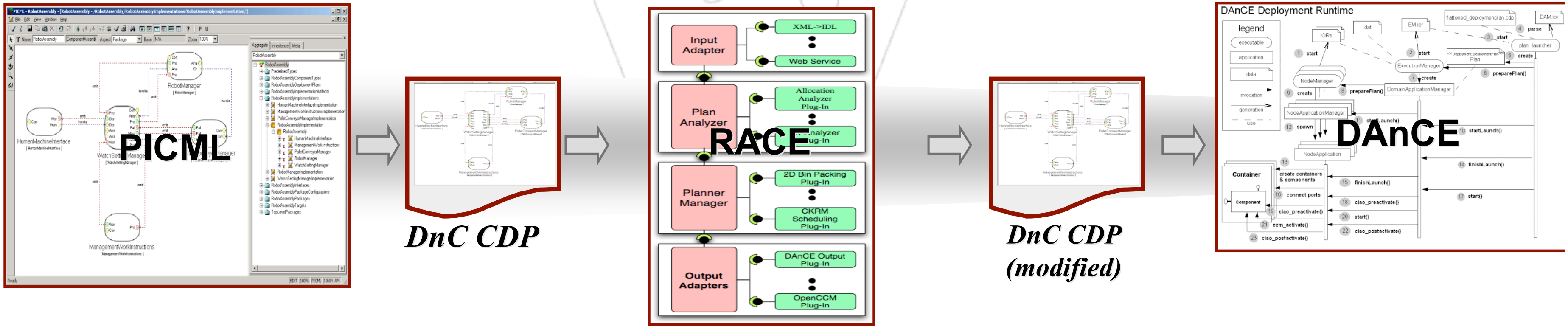
- Implement
  - Resource monitors
  - Target Manager



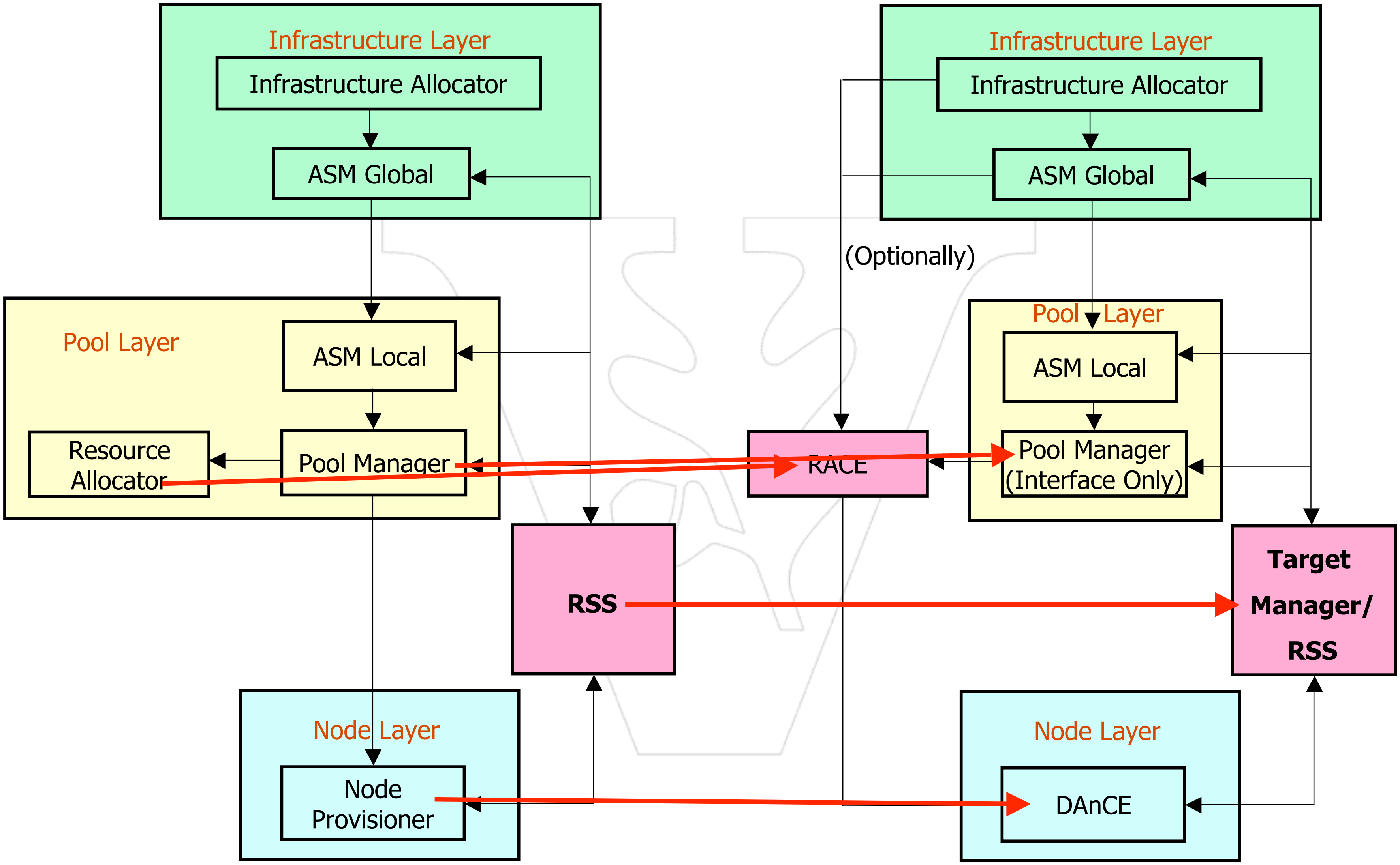
Current RACE source code is available on the ACE CVS repository under ACE_wrappers/TAO/CIAO/RACE

RACE Demo

- Utilizes an instance of RACE configured with:
  - An Input Adapter that implements a web application using JAWS to:
    - Allow modification of WLG deployments
    - Select and deploy WLG both static and dynamic assemblies
  - A simple Plan Analyzer which selects the only planner available in the system
  - A Planner implementing a simple bin-packing algorithm
    - Examines a property describing CPU utilization of a WLG component
    - Does not take into account current resource utilization
  - An Output Adapter which deploys plans using DAnCE

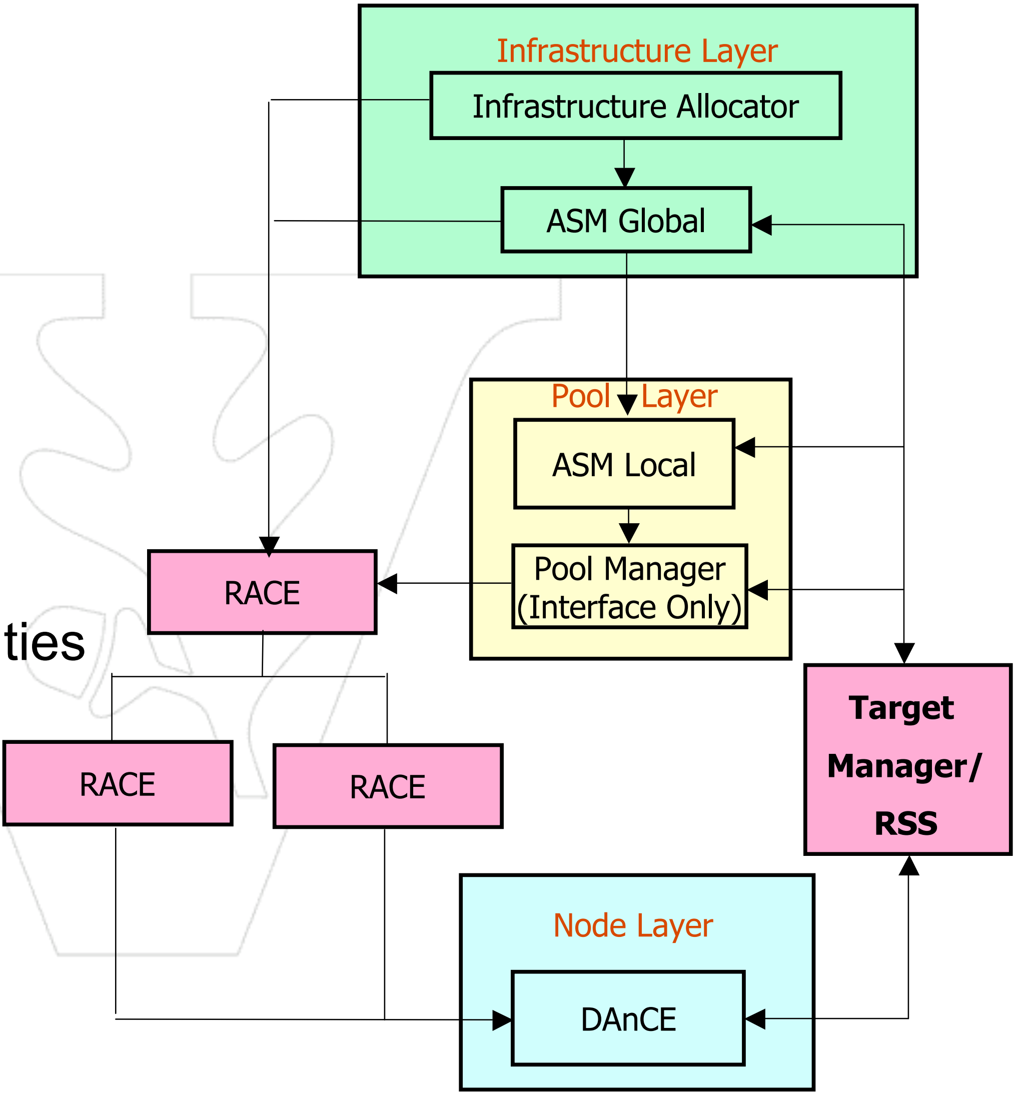


Integrating MLRM Architecture with RACE/DAnCE



Integrating MLRM Architecture with RACE/DAnCE

- DnC Spec compliant entities
  - Wide applicability
  - Can be reused outside ARMS
- Pluggable Framework
  - Supports multiple
    - Allocation Algorithms
    - Adaptation Algorithms
- Number of layers in MRLM architecture is flexible
  - Increases scalability
- Logical grouping of related entities
- Provides a ***template*** for each (higher) layers in the MLRM architecture
- MLRM is truly a *Multi*-Layer Resource Management Middleware, and not limited to three layers



RACE Implementation Milestones

RACE Element	Timeframe	POC
Integrate Demo code into CVS	1-2 weeks	Ed Mulholland
Plan Analyzer	1 Month	William Otte (wotte@dre.vanderbilt.edu)
Analyzer Plugins	1 Month	William Otte (wotte@dre.vanderbilt.edu)
Plan Manager	1 Month	Jai Balasubramanian (jai@dre.vanderbilt.edu)
Input/Output adapters	2 Weeks	William Otte (wotte@dre.vanderbilt.edu)
Implement Planner/Analyzer meta-data catalogue		Jai Balasubramanian (jai@dre.vanderbilt.edu)
Implement Target Manager with RSS	3 Weeks	Nilabja Roy (nilabja@dre.vanderbilt.edu)
Design RACE Monitoring Framework	2 Weeks	Nishanth Shankaran (nishanth@dre.vanderbilt.edu)
Design RACE Control Infrastructure	1 Month	Nishanth Shankaran (nshanth@dre.vanderbilt.edu)

ARMS RACE Integration Milestones

RACE Element	Timeframe	POC
Execute Phase 1 Gate Test 3 using RACE	1.5 Months	Ed Mulholland (emulholl@atl.lmco.com) Will Otte (wotte@dre.vanderbilt.edu)
Execute Phase 2 Gate Test 1 Using RACE	2 Months	Ed Mulholland (emulholl@atl.lmco.com) Will Otte (wotte@dre.vanderbilt.edu)
Develop DnC Deployment Plan representation of the AIM	3 Months	Josh Chattin (jchattin@atl.lmco.com)