Overview of Parallel Programming Concepts

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

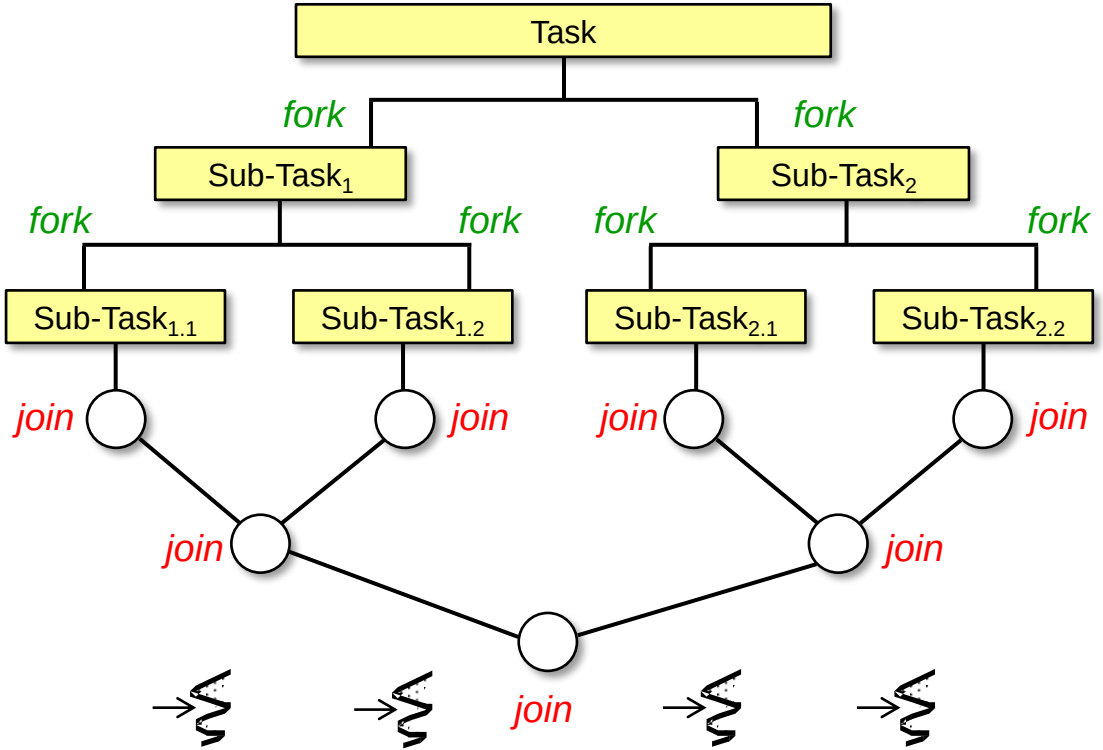
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the meaning of key parallel programming concepts



See en.wikipedia.org/wiki/Parallel_computing

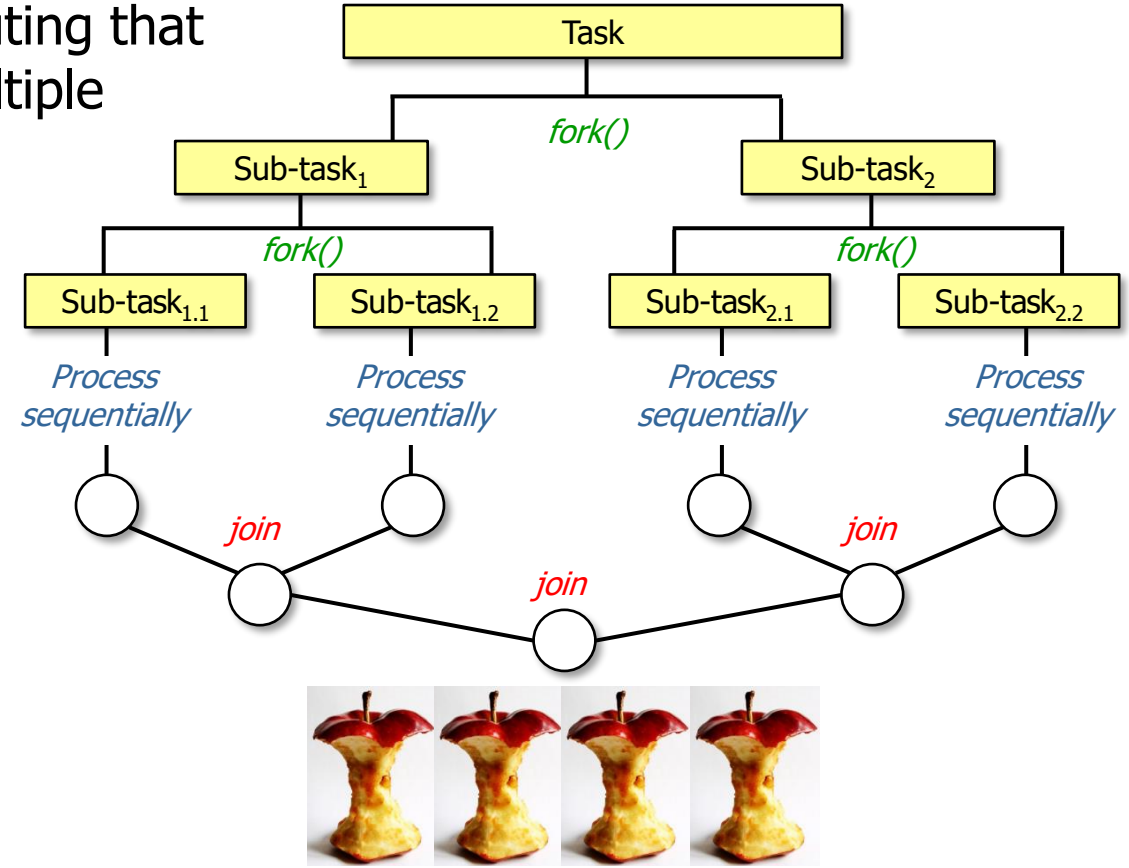
Learning Objectives in this Part of the Lesson

- Understand the meaning of key parallel programming concepts
- Know when to apply parallelism

An Overview of Parallel Programming

An Overview of Parallel Programming

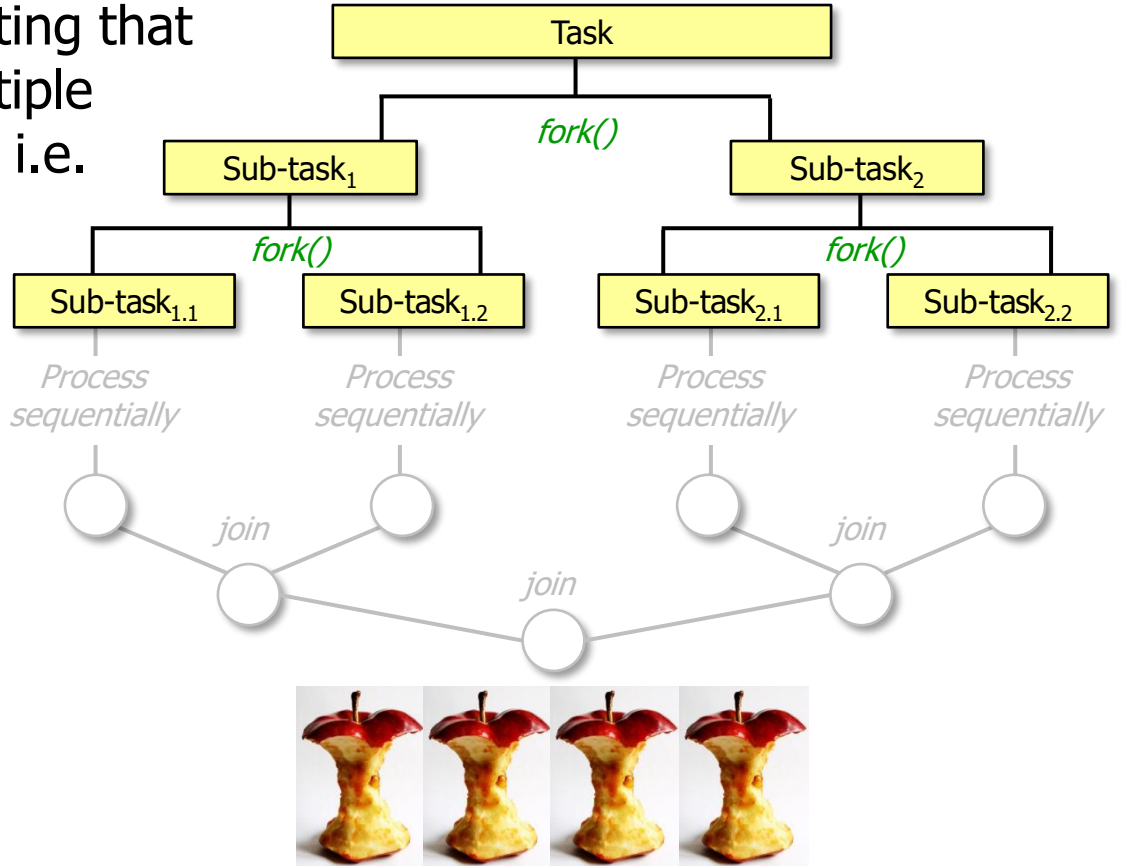
- Parallelism is a form of computing that performs several steps on multiple processors or processor cores



See www.jstatsoft.org/article/view/v040i01/v40i01.pdf

An Overview of Parallel Programming

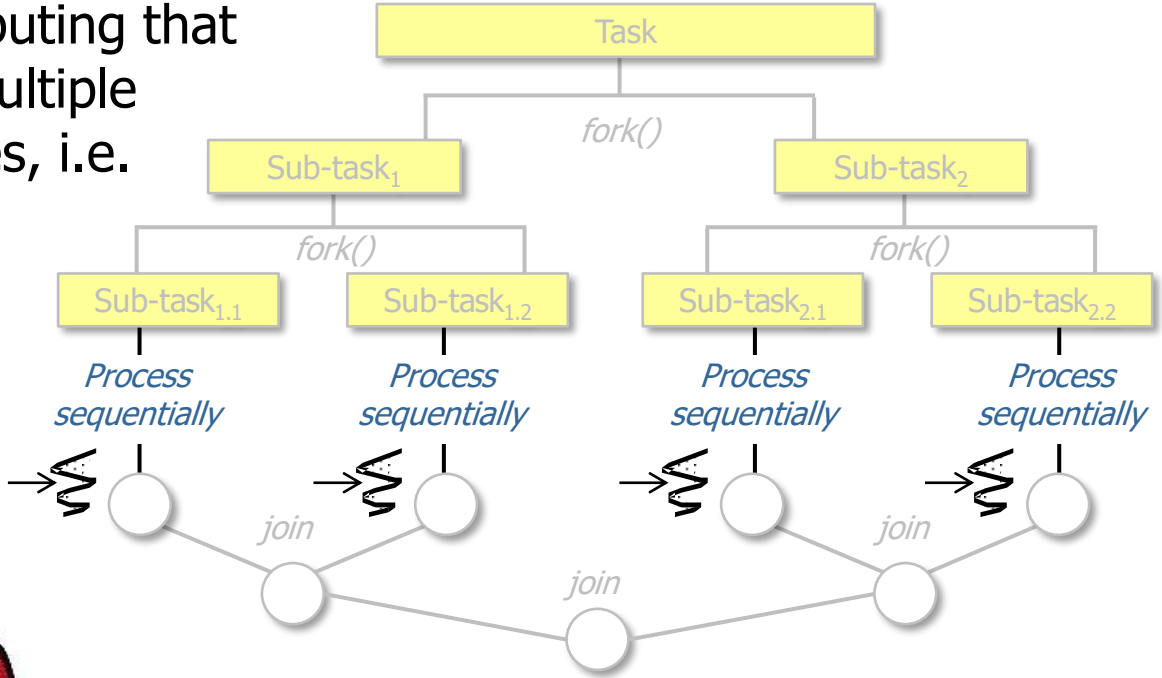
- Parallelism is a form of computing that performs several steps on multiple processors or processor cores, i.e.
  - Split – partition a task into sub-tasks

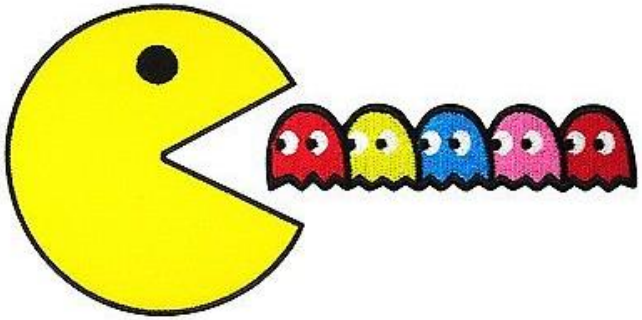


Ideally sub-tasks are split efficiently & evenly

An Overview of Parallel Programming

- Parallelism is a form of computing that performs several steps on multiple processors or processor cores, i.e.
  - Split – partition a task into sub-tasks
  - Apply – Run independent sub-tasks in parallel

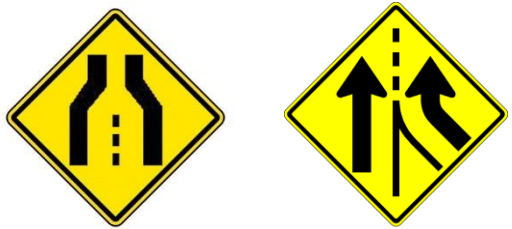


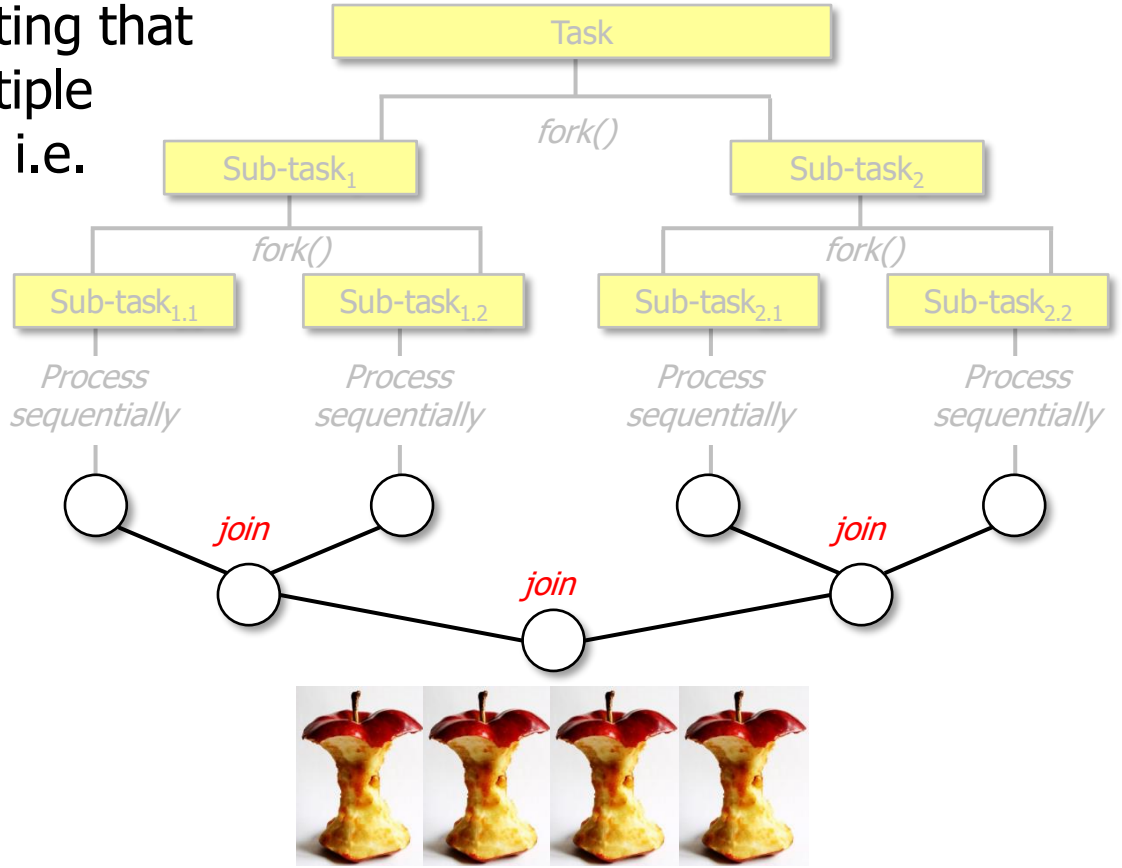


Each sub-task runs sequentially, but together they run in parallel

An Overview of Parallel Programming

- Parallelism is a form of computing that performs several steps on multiple processors or processor cores, i.e.
  - Split – partition a task into sub-tasks
  - Apply – Run independent sub-tasks in parallel
  - Combine – Merge the sub-results from sub-tasks into a single "reduced" result





An Overview of Parallel Programming

- A key goal of parallelism is to *efficiently* partition tasks into sub-tasks & combine results

An Overview of Parallel Programming

- A key goal of parallelism is to *efficiently* partition tasks into sub-tasks & combine results
  - Parallelism can thus be viewed as an optimization to improve performance

See developer.ibm.com/articles/j-java-streams-4-brian-goetz

An Overview of Parallel Programming

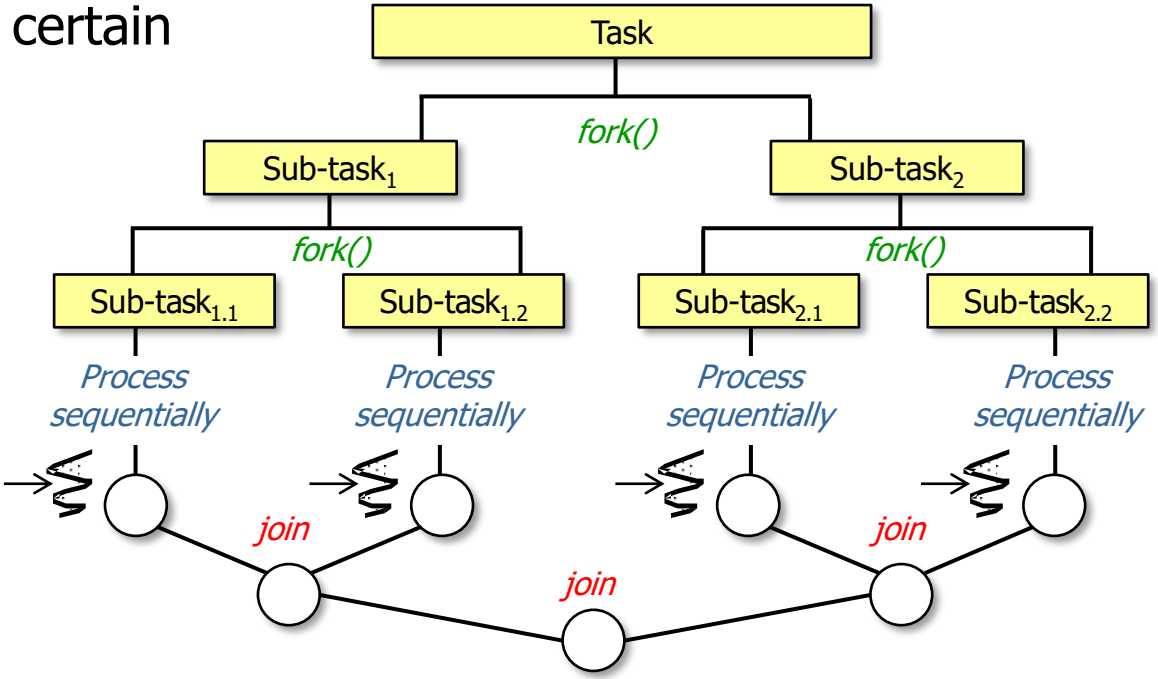
- A key goal of parallelism is to *efficiently* partition tasks into sub-tasks & combine results
  - Parallelism can thus be viewed as an optimization to improve performance
    - e.g., throughput, scalability, & latency

See en.wikipedia.org/wiki/Up_to_eleven

When to Apply Parallelism

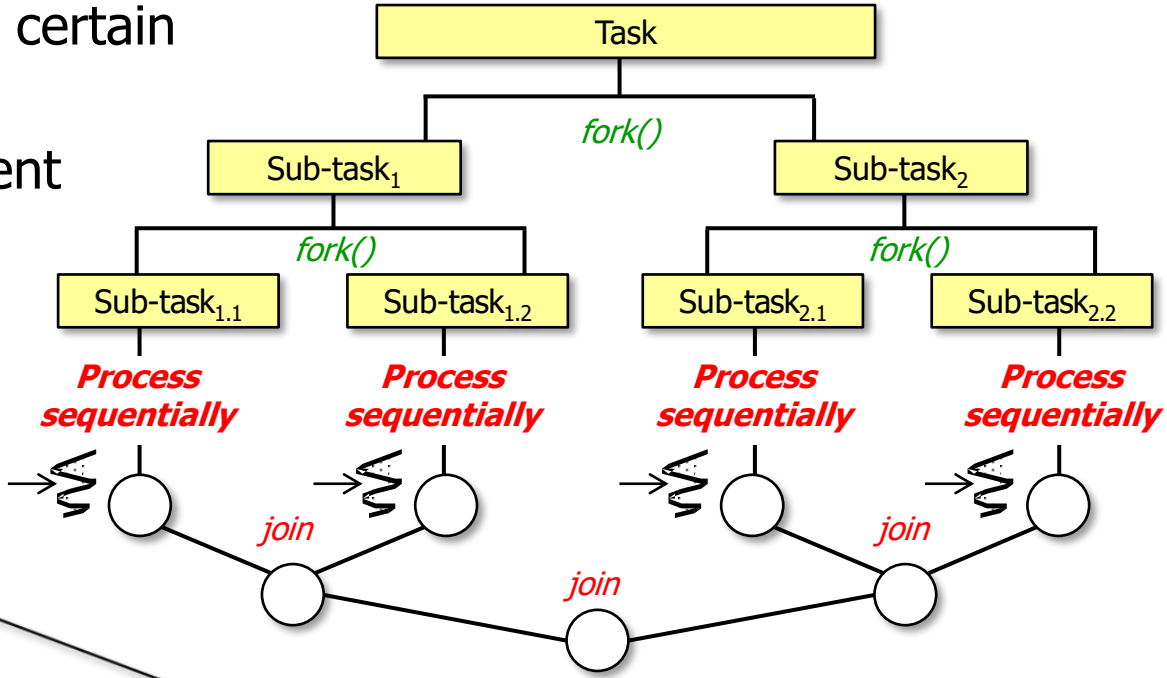
When to Apply Parallelism

- Parallelism works best under certain conditions



When to Apply Parallelism

- Parallelism works best under certain conditions, e.g.
  - When tasks are independent

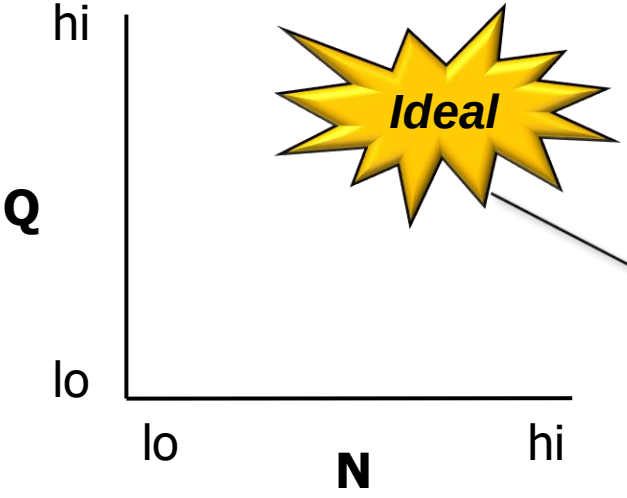


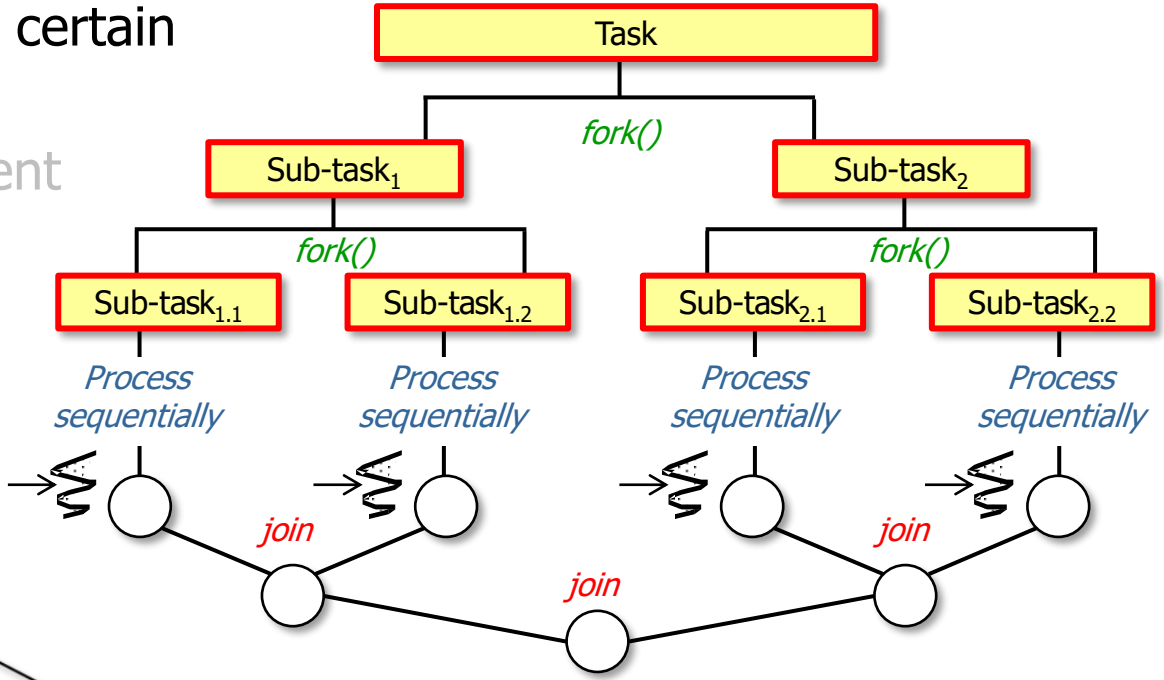
"Embarrassingly parallel" tasks have little/no dependency or need for communication between tasks or for sharing results between them

See en.wikipedia.org/wiki/Embarrassingly_parallel

When to Apply Parallelism

- Parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform



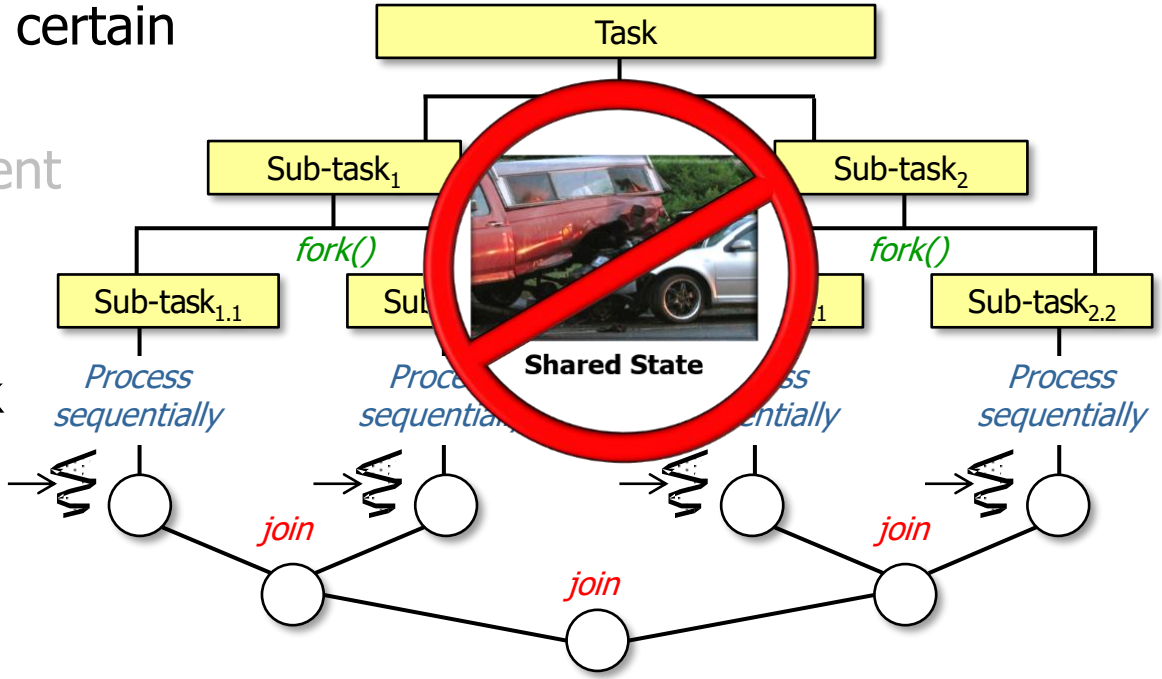


- *N is the # of data elements to process per thread*
- *Q quantifies how CPU-intensive the processing is*

See on-sw-integration.epischel.de/2016/08/05/parallel-stream-processing-with-java-8-stream-api

When to Apply Parallelism

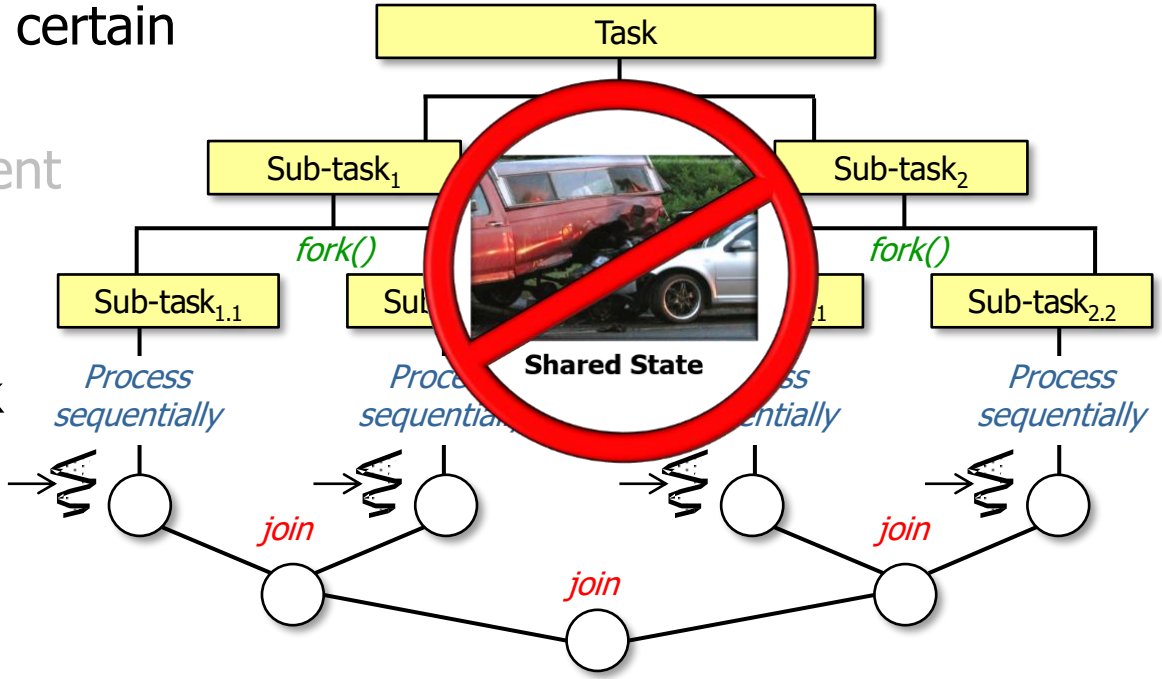
- Parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When threads neither block nor share mutable state



See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html

When to Apply Parallelism

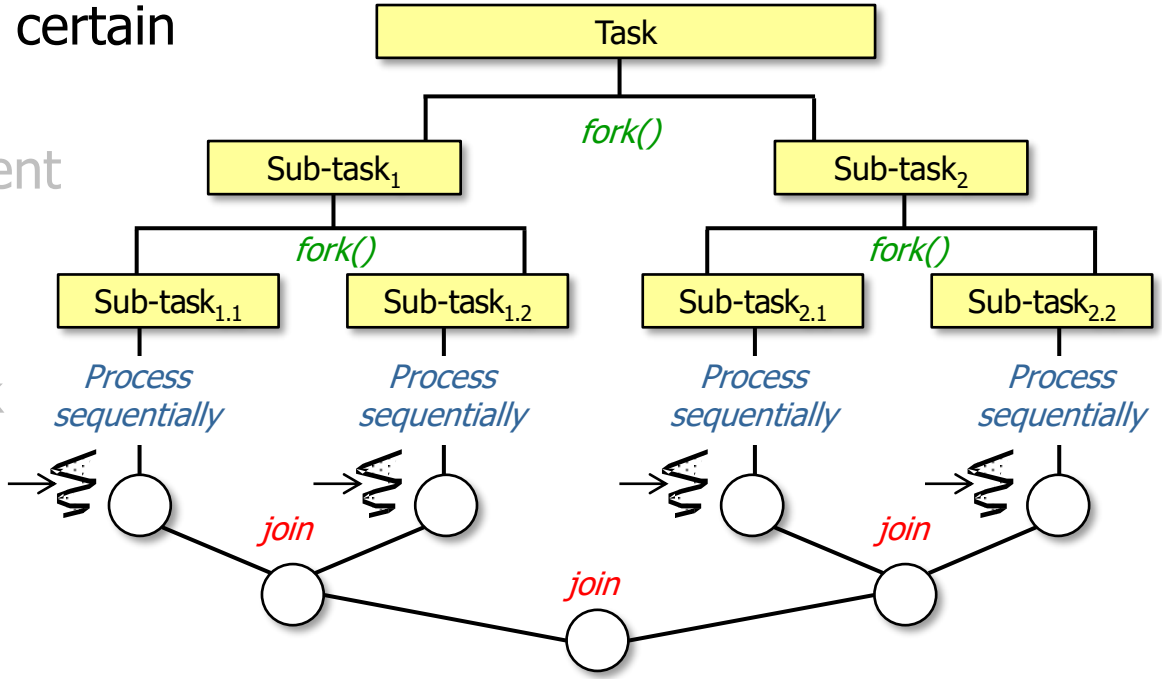
- Parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When threads neither block nor share mutable state
    - Hence Java's "fork-join" & "work-stealing" foci



See en.wikipedia.org/wiki/Fork-join_model & en.wikipedia.org/wiki/Work_stealing

When to Apply Parallelism

- Parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When threads neither block nor share mutable state
  - When there are many cores and/or processors



See en.wikipedia.org/wiki/Multi-core_processor & en.wikipedia.org/wiki/Multiprocessing

End of Overview of Parallel Programming Concepts