The Java Fork-Join Pool: Introduction

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

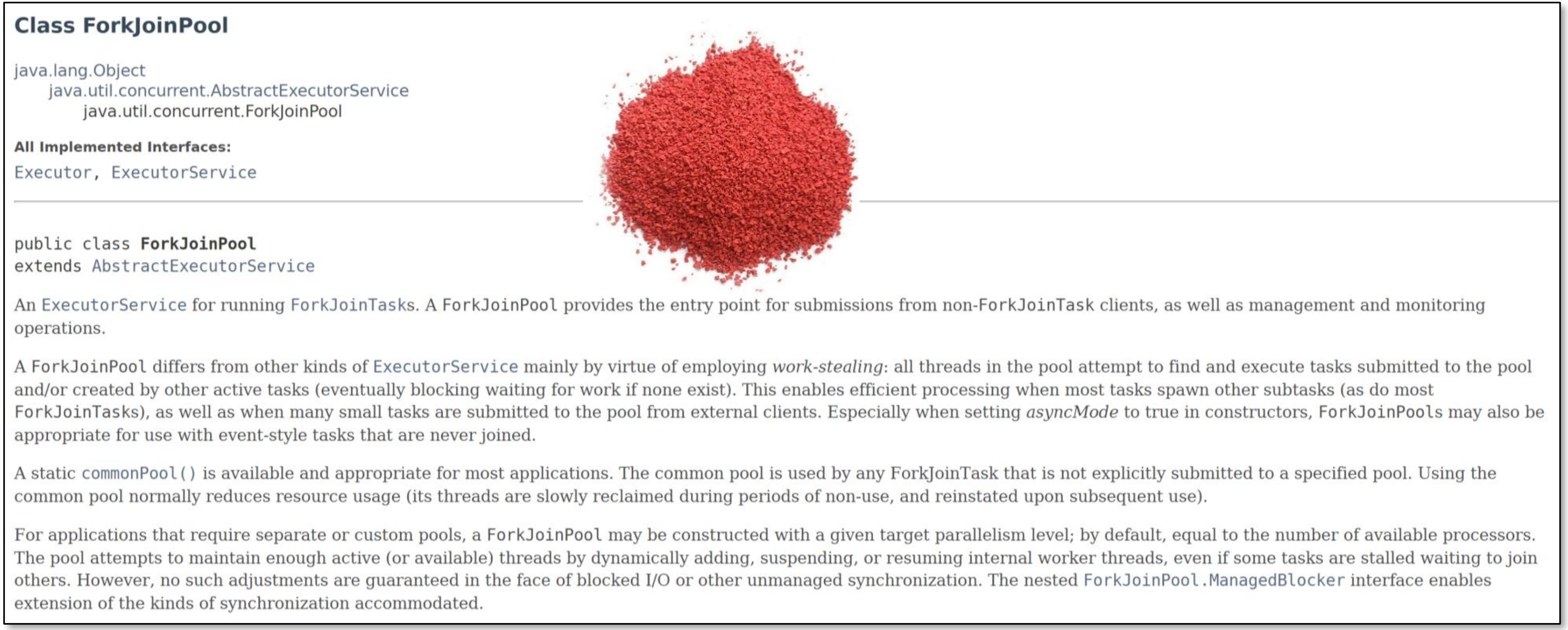
Learning Objectives in this Part of the Lesson

- Understand how the Java fork-join framework processes tasks in parallel

Overview of the Java Fork-Join Pool Computation Model

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism



Class ForkJoinPool

java.lang.Object
    java.util.concurrent.AbstractExecutorService
        java.util.concurrent.ForkJoinPool

All Implemented Interfaces:

Executor, ExecutorService

```
public class ForkJoinPool
extends AbstractExecutorService
```

An ExecutorService for running ForkJoinTasks. A ForkJoinPool provides the entry point for submissions from non-ForkJoinTask clients, as well as management and monitoring operations.

A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most ForkJoinTasks), as well as when many small tasks are submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, ForkJoinPools may also be appropriate for use with event-style tasks that are never joined.

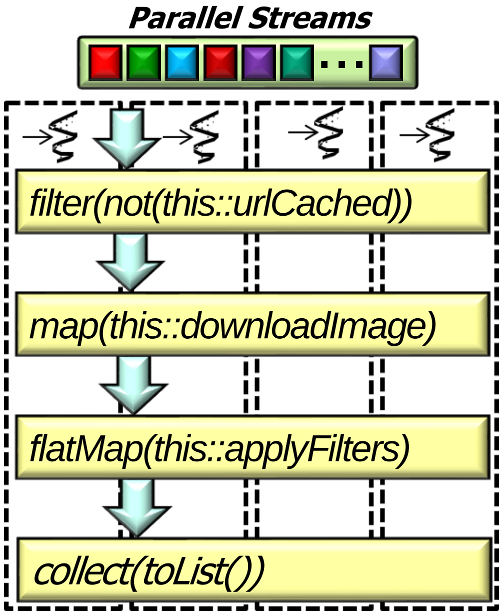
A static commonPool() is available and appropriate for most applications. The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool. Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).

For applications that require separate or custom pools, a ForkJoinPool may be constructed with a given target parallelism level; by default, equal to the number of available processors. The pool attempts to maintain enough active (or available) threads by dynamically adding, suspending, or resuming internal worker threads, even if some tasks are stalled waiting to join others. However, no such adjustments are guaranteed in the face of blocked I/O or other unmanaged synchronization. The nested ForkJoinPool.ManagedBlocker interface enables extension of the kinds of synchronization accommodated.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool provides a high performance, fine-grained task execution framework for Java data parallelism
  - Its parallel computing engine is used by many higher-level frameworks



Parallel Streams

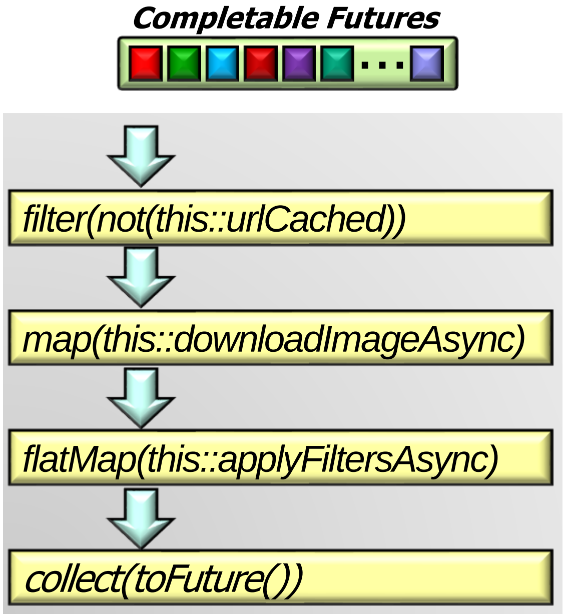
filter(not(this::urlCached))

map(this::downloadImage)

flatMap(this::applyFilters)

collect(toList())

ForkJoinPool



Completable Futures

filter(not(this::urlCached))

map(this::downloadImageAsync)

flatMap(this::applyFiltersAsync)

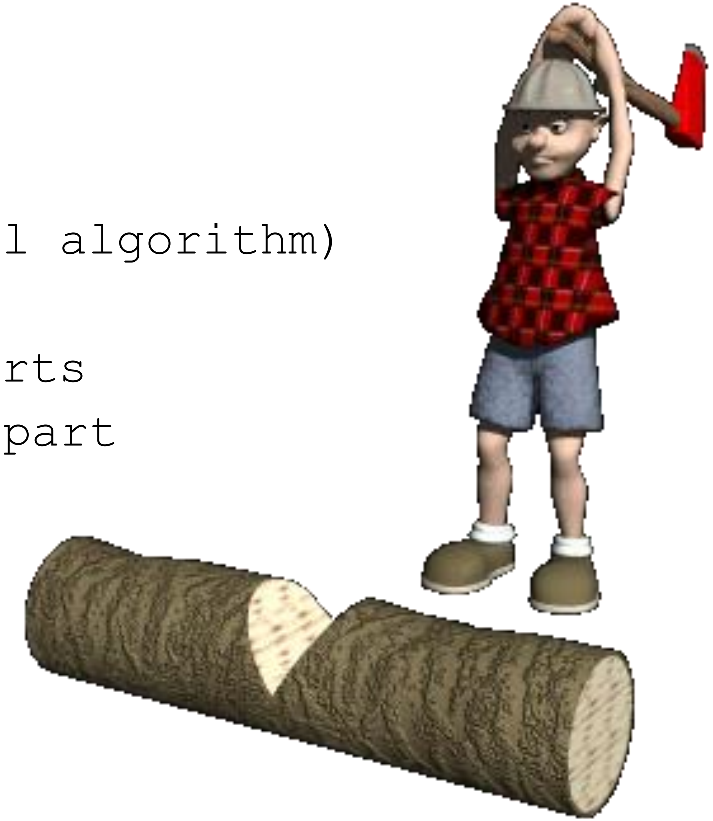
collect(toFuture())

See www.infoq.com/interviews/doug-lea-fork-join

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by “divide & conquer”

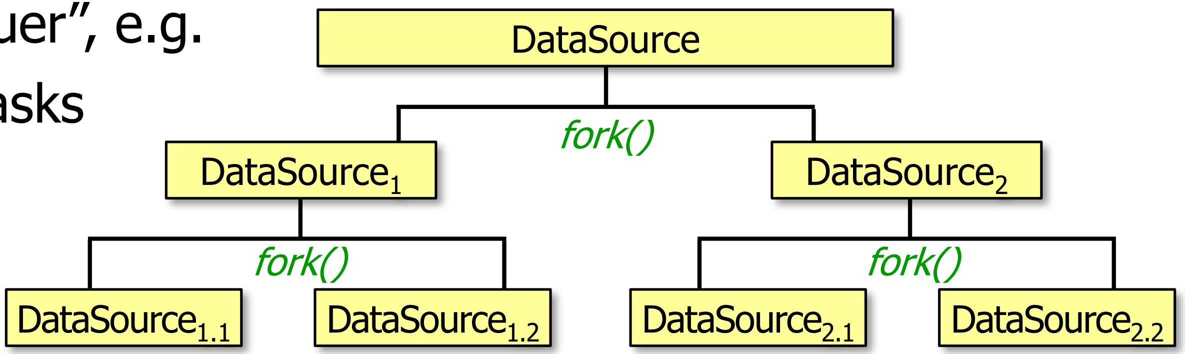
```
Solve(problem)
  if problem is small enough
    solve problem directly (sequential algorithm)
  else
    split problem into independent parts
    fork new sub-tasks to solve each part
    join all sub-tasks
    compose result from sub-results
```



See en.wikipedia.org/wiki/Divide_and_conquer_algorithm

Overview of the Java Fork-Join Pool Computation Model

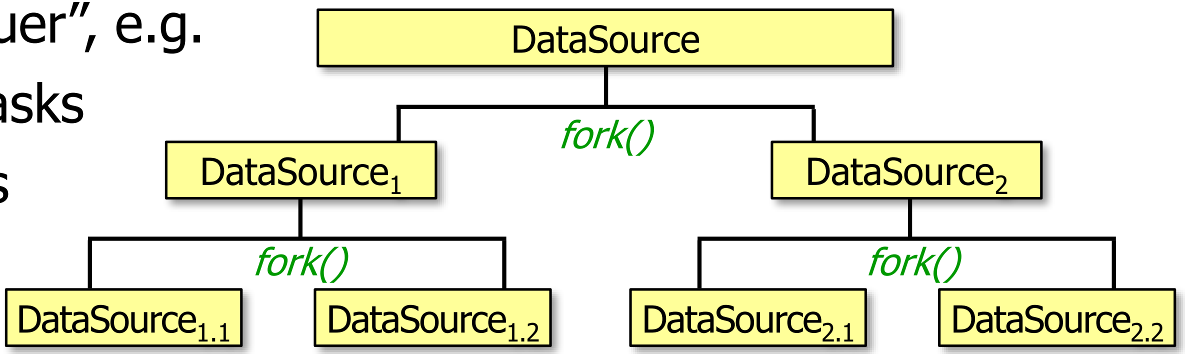
- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks



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

Overview of the Java Fork-Join Pool Computation Model

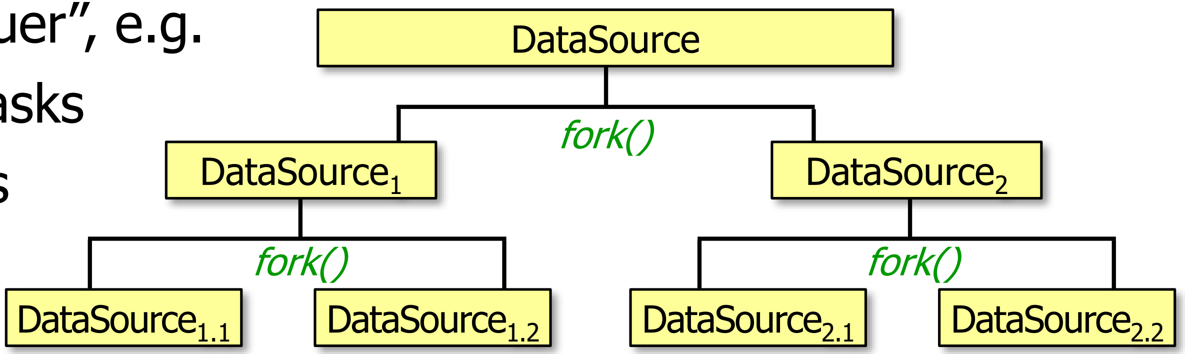
- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
    - A task creates sub-tasks by fork()'ing



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#fork

Overview of the Java Fork-Join Pool Computation Model

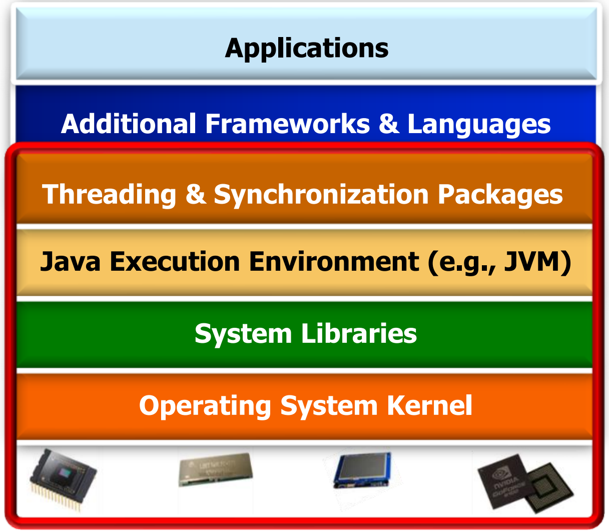
- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
    - A task creates sub-tasks by fork()'ing

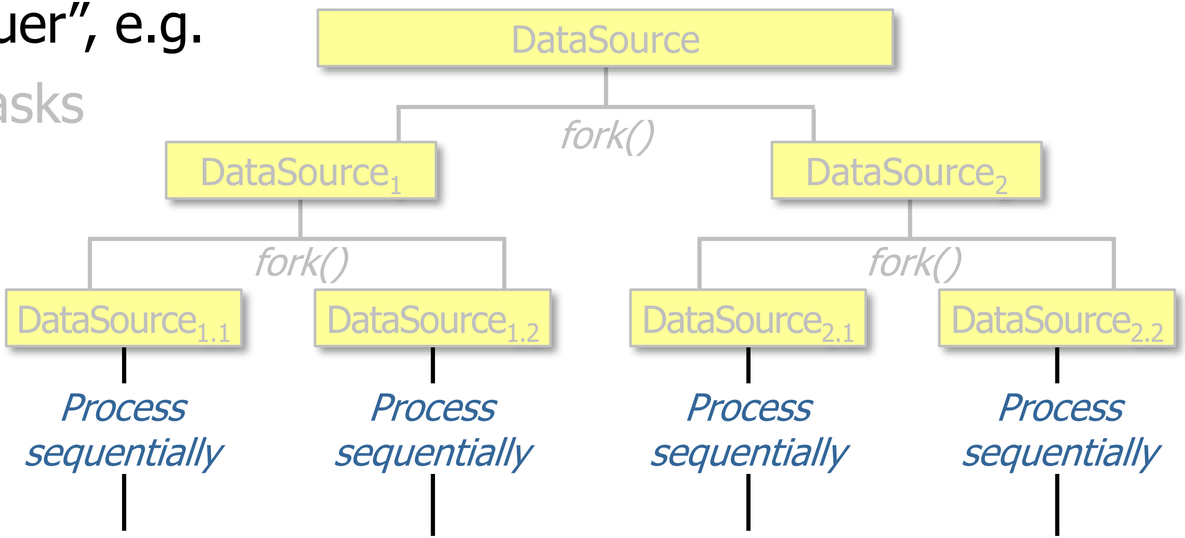


A (sub-)task only splits itself into (more) sub-tasks if the work is sufficiently big

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel

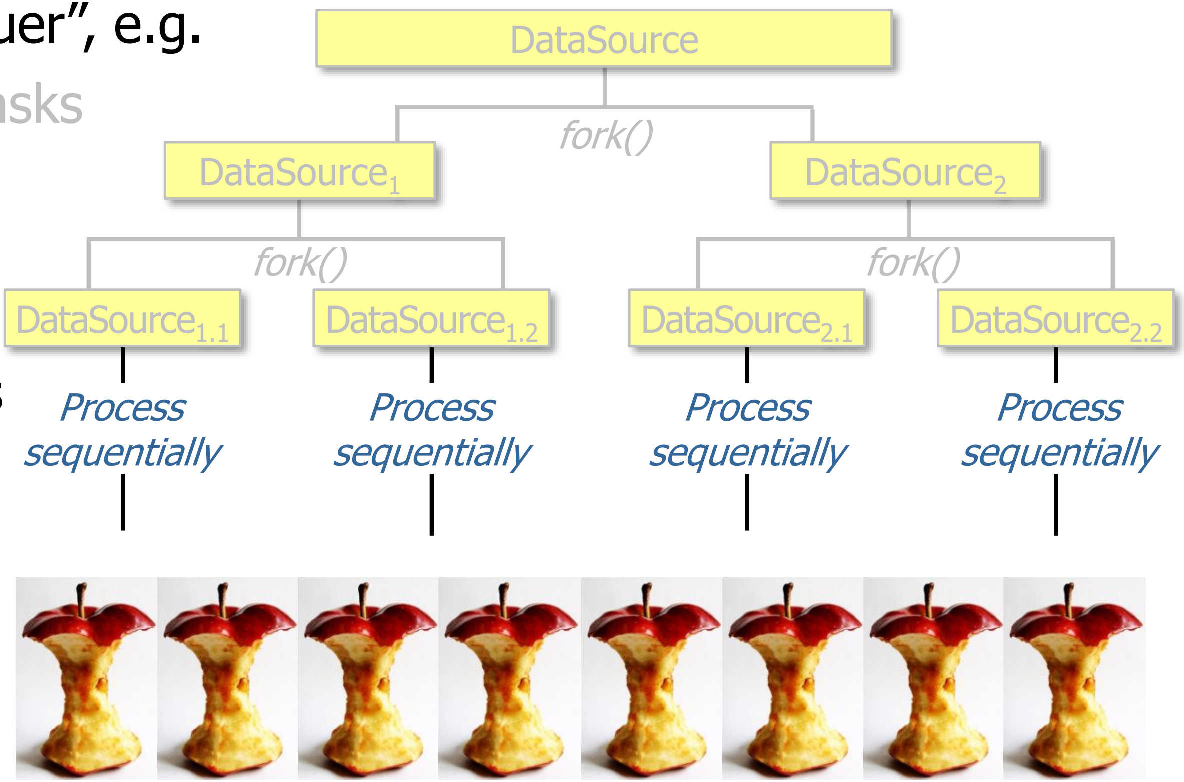

Applications
Additional Frameworks & Languages
Threading & Synchronization Packages
Java Execution Environment (e.g., JVM)
System Libraries
Operating System Kernel



Implemented by fork-join framework, Java execution environment, OS, & hardware

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
    - Sub-tasks can run in parallel on different cores



DataSource

fork()

DataSource$_1$

DataSource$_2$

fork()

fork()

DataSource$_{1.1}$

DataSource$_{1.2}$

DataSource$_{2.1}$

DataSource$_{2.2}$

Process sequentially

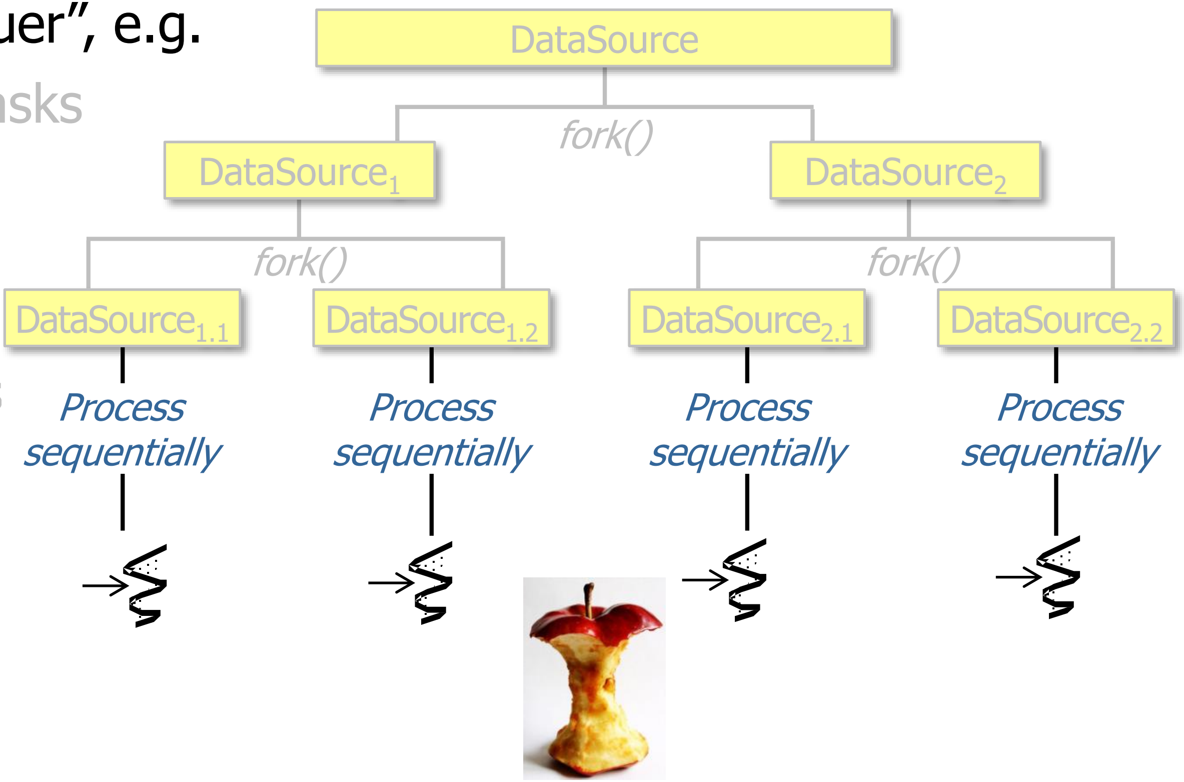
Process sequentially

Process sequentially

Process sequentially

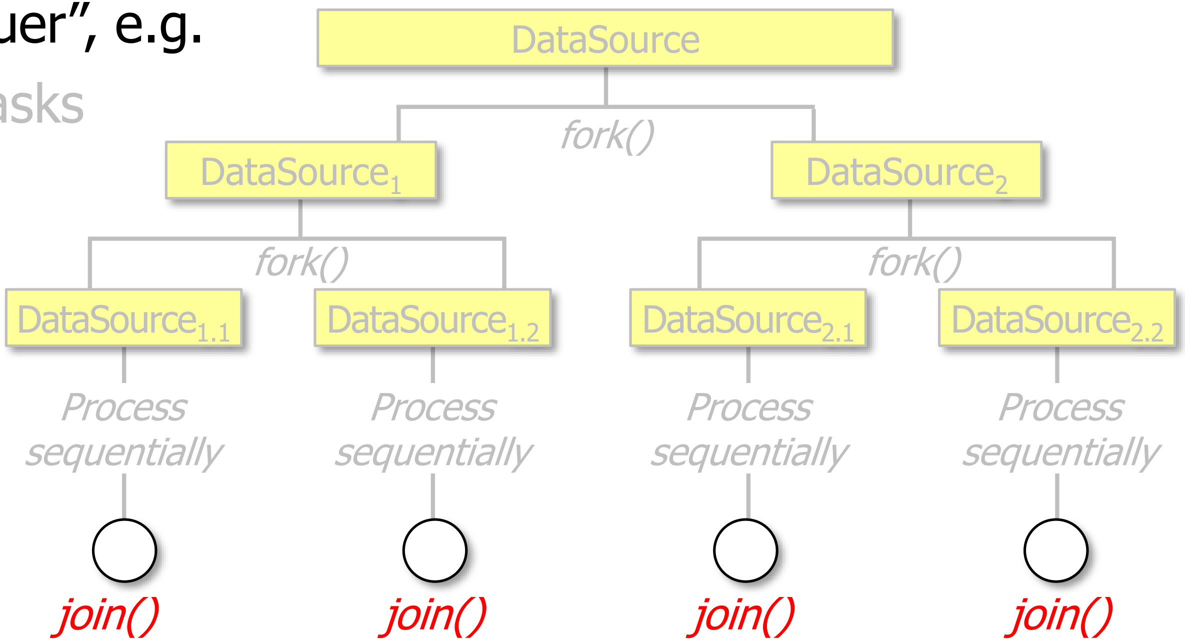
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
    - Sub-tasks can run in parallel on different cores
    - Sub-tasks can also run concurrently in different threads on a single core



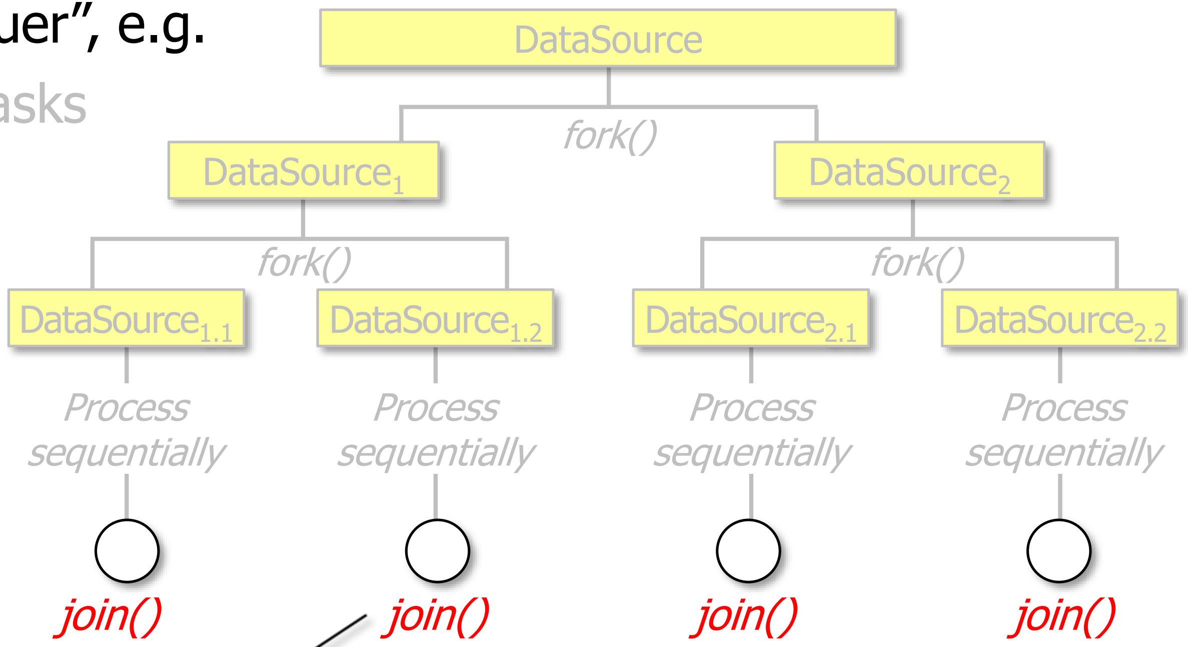
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
  - Waiting for them to complete



Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
  - Waiting for them to complete
    - join() waits for a sub-task to finish

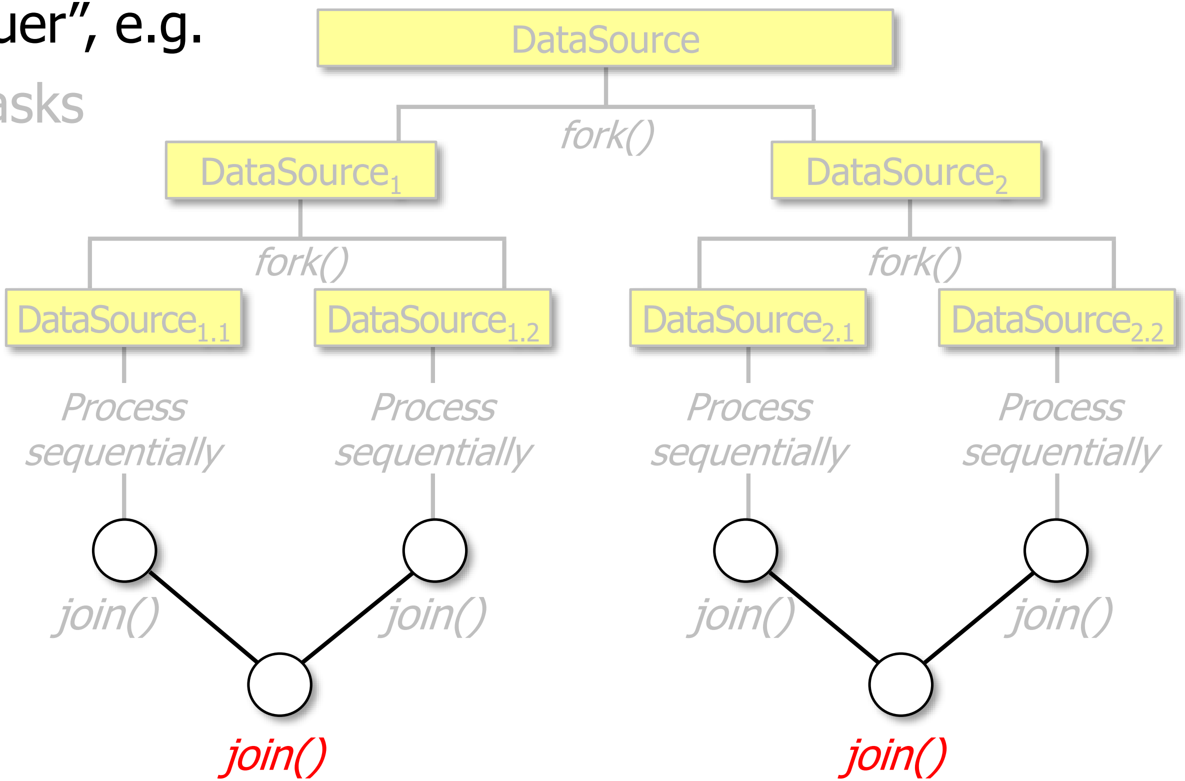


join() also plays a role in executing sub-tasks, as discussed shortly

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#join

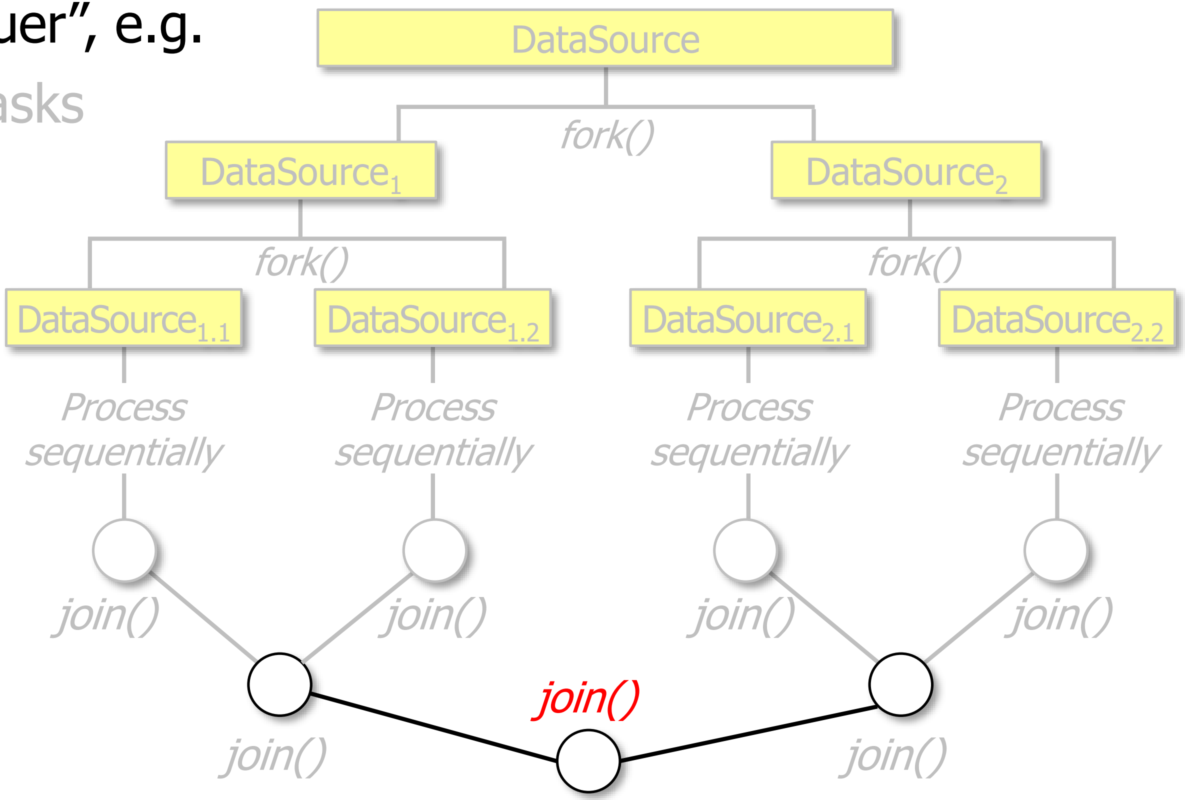
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
  - Waiting for them to complete
  - Merging the results



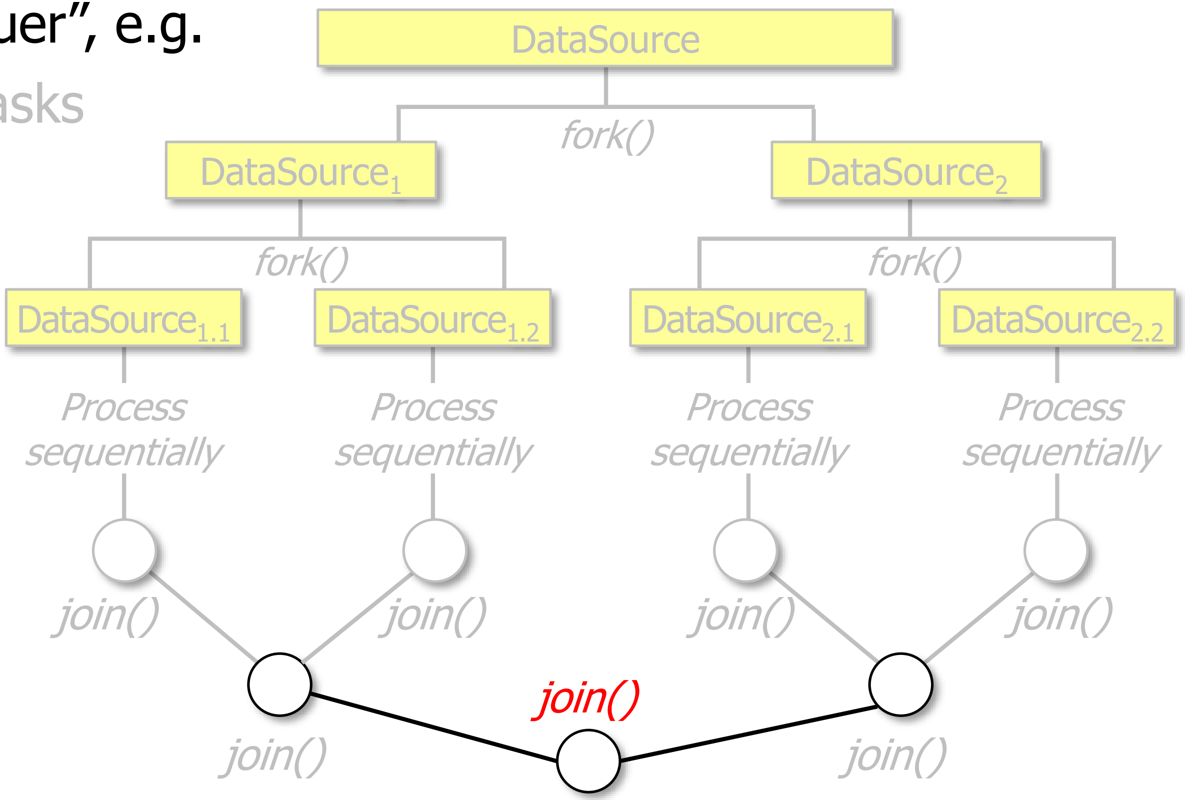
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
  - Waiting for them to complete
  - Merging the results
    - A task can use calls to join() to merge the sub-task results together



Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide & conquer", e.g.
  - Splitting a task into sub-tasks
  - Solving the sub-tasks in parallel
  - Waiting for them to complete
  - Merging the results
    - A task can use calls to join() to merge the sub-task results together



If a task does not return a result then it just waits for its sub-tasks to complete

End of the Java Fork-Join Pool: Introduction