Visualizing Java Futures in Action

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

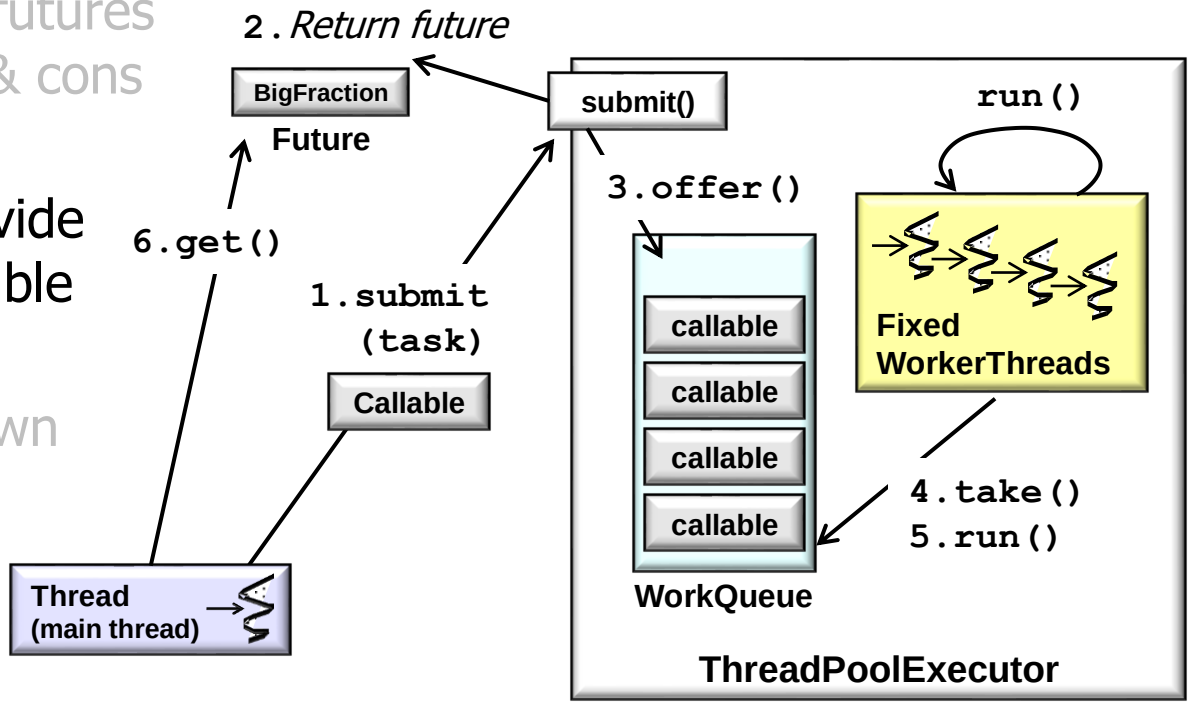
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

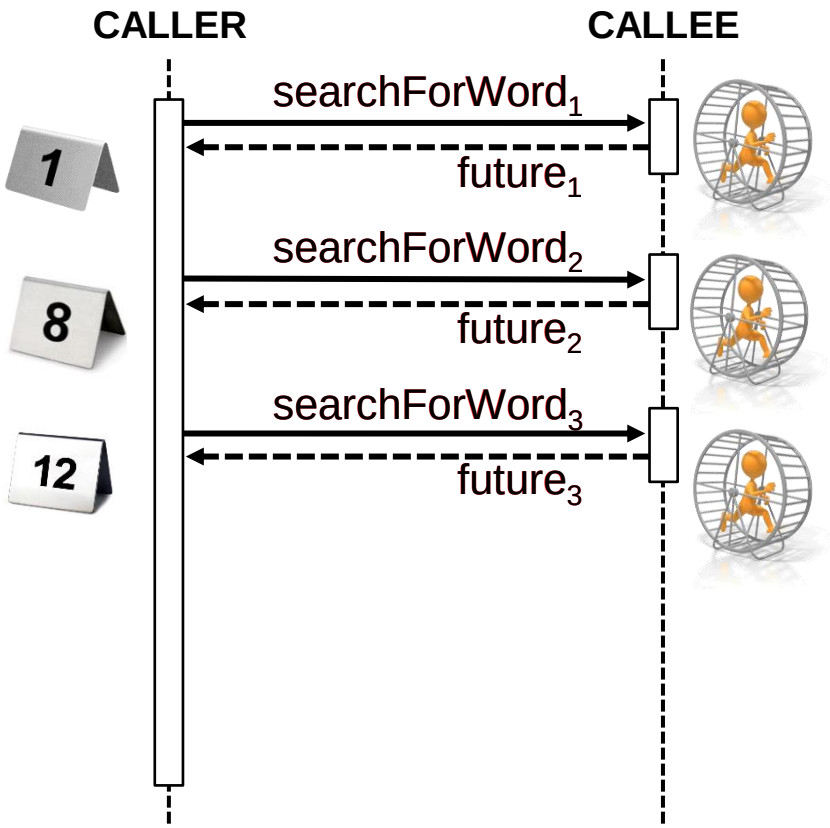
- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Know how Java futures provide the foundation for completable futures in Java
  - Understand a human known use of Java futures
  - Recognize the methods in the Future interface
  - Visualize Java futures in action



Visualizing Java Futures in Action

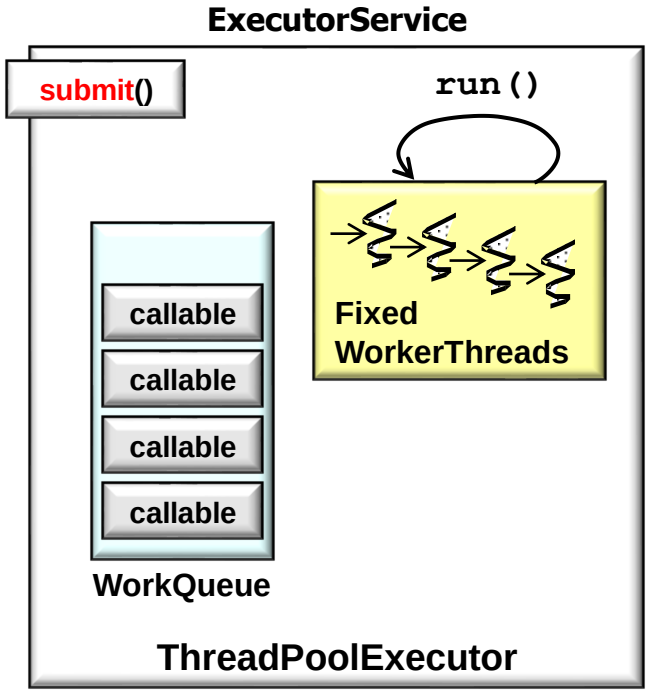
Visualizing Java Futures in Action

- An Java async call immediately returns a future & continues to run the computation in a background thead



Visualizing Java Futures in Action

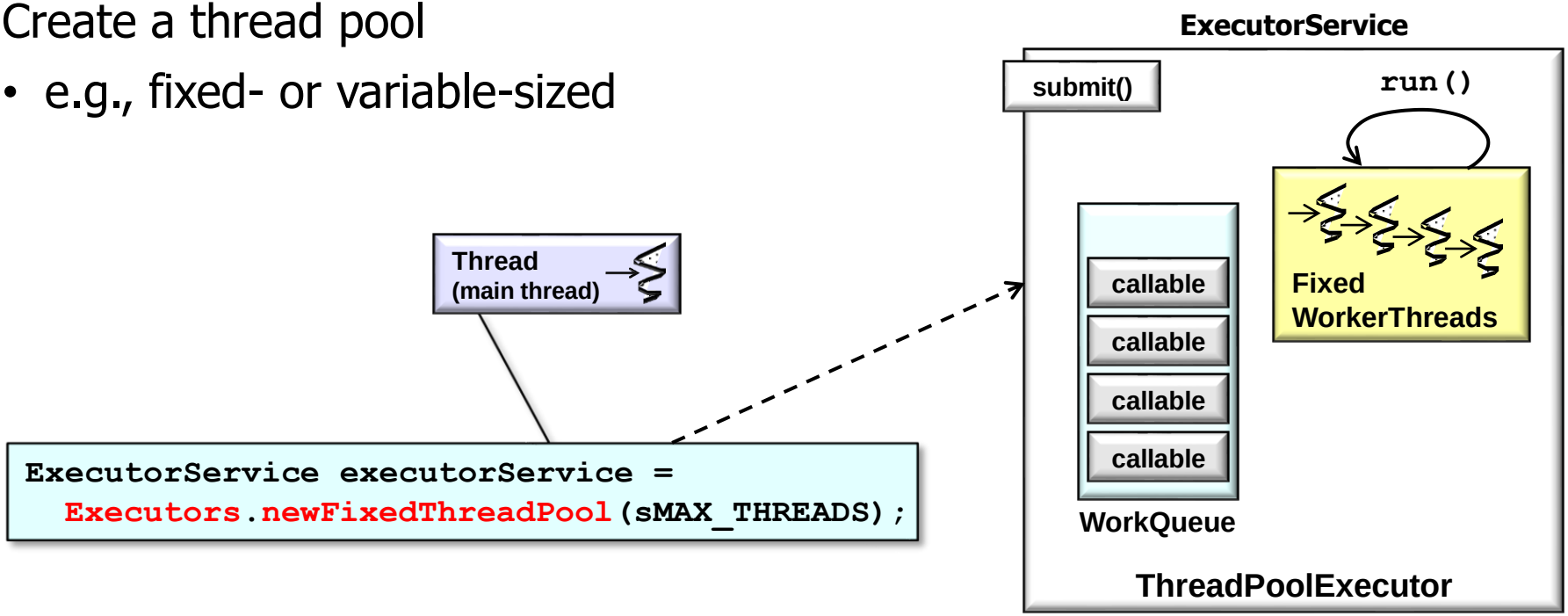
- ExecutorService.submit() can initiate an async call in Java



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit

Visualizing Java Futures in Action

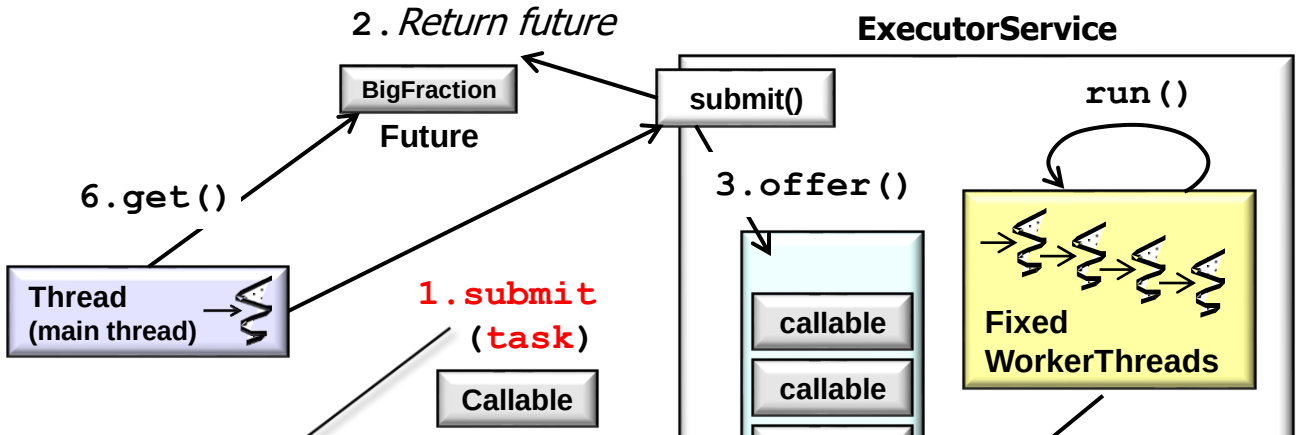
- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool
    - e.g., fixed- or variable-sized



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool

Visualizing Java Futures in Action

- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool
  - Submit a task
    - e.g., a callable



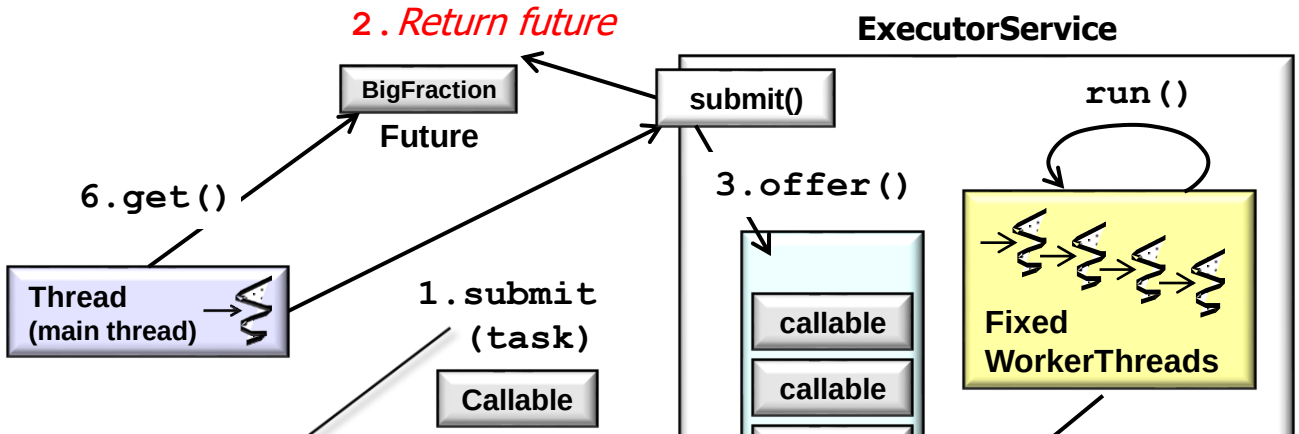

```
Callable<BigFraction> task = () -> {
  BigFraction bf1 = new BigFraction(f1);
  BigFraction bf2 = new BigFraction(f2);
  return bf1.multiply(bf2);
};

Future<BigFraction> future = executorService.submit(task);
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit

Visualizing Java Futures in Action

- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool
  - Submit a task
  - Return a future
    - e.g., implemented as a FutureTask



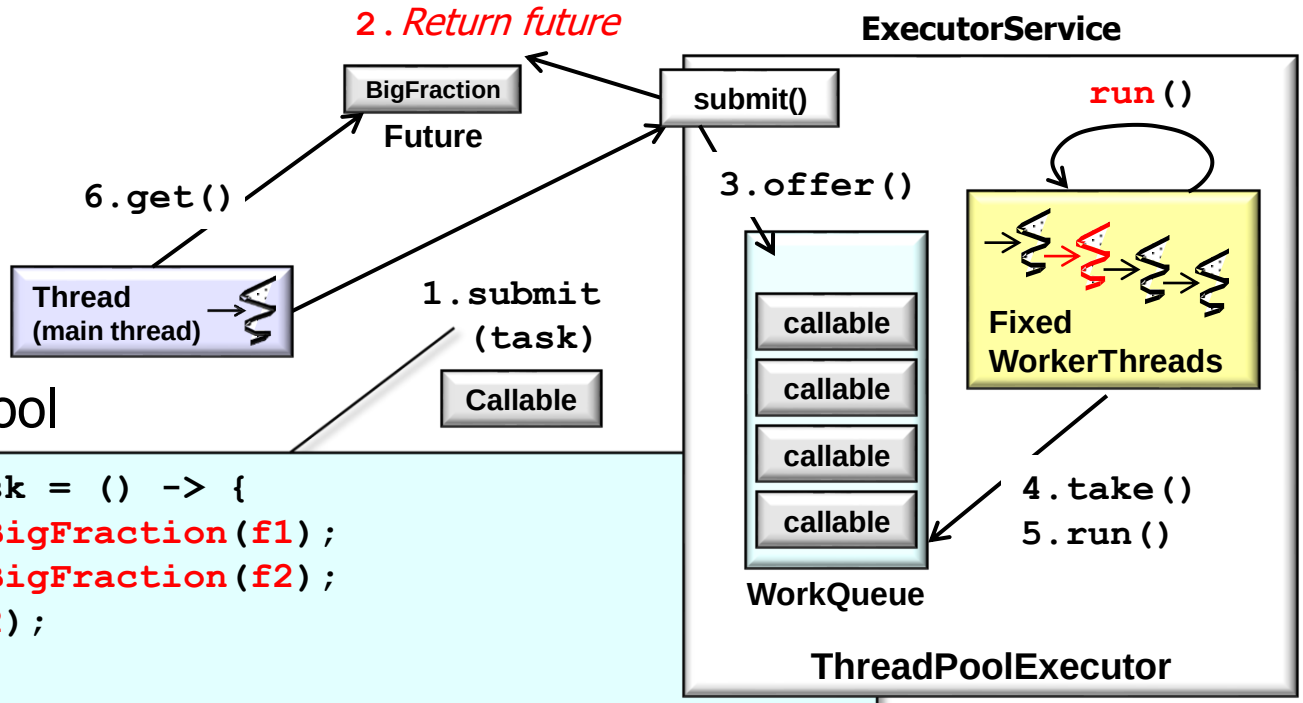

```
Callable<BigFraction> task = () -> {
  BigFraction bf1 = new BigFraction(f1);
  BigFraction bf2 = new BigFraction(f2);
  return bf1.multiply(bf2);
};

Future<BigFraction> future = executorService.submit(task);
```

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

Visualizing Java Futures in Action

- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool
  - Submit a task
  - Return a future
  - Run computation asynchronously
    - e.g., in a thread pool

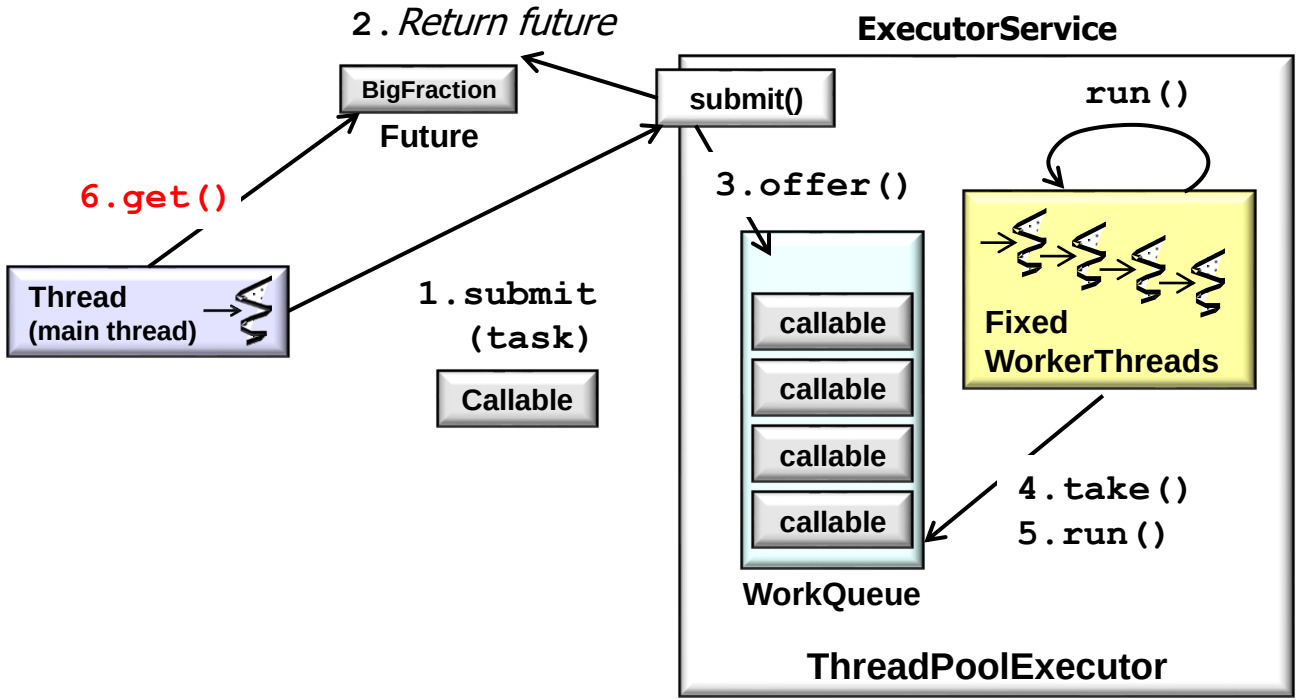



```
Callable<BigFraction> task = () -> {
  BigFraction bf1 = new BigFraction(f1);
  BigFraction bf2 = new BigFraction(f2);
  return bf1.multiply(bf2);
};

Future<BigFraction> future = executorService.submit(task);
```

Visualizing Java Futures in Action

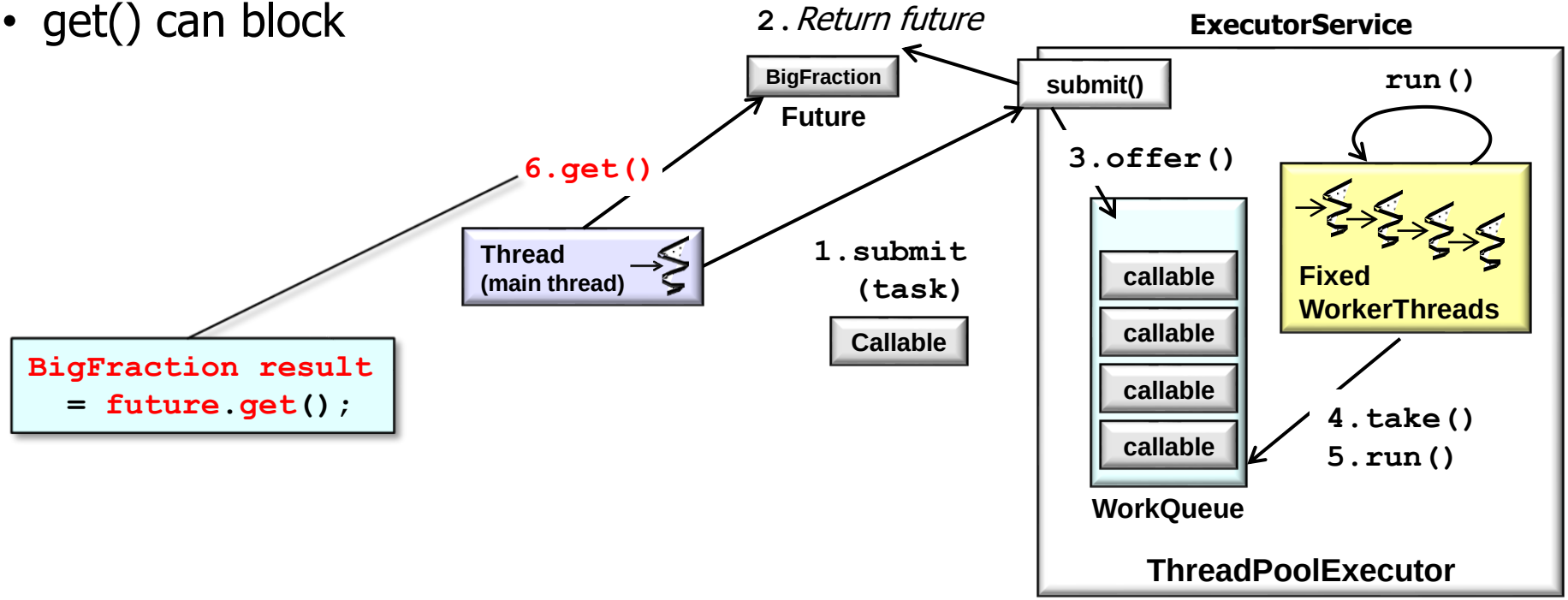
- When the async call completes the future is triggered & the result is available



See www.nurkiewicz.com/2013/02/javautilconcurrentfuture-basics.html

Visualizing Java Futures in Action

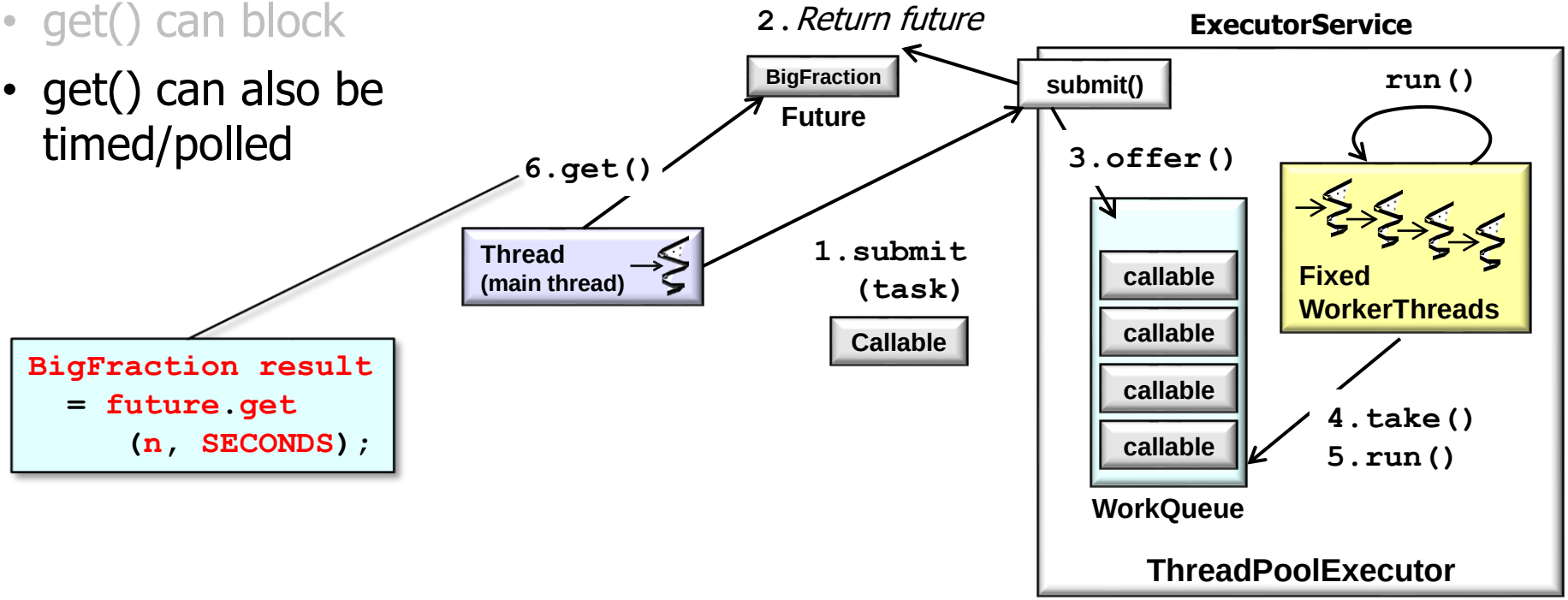
- When the async call completes the future is triggered & the result is available
  - get() can block



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get

Visualizing Java Futures in Action

- When the async call completes the future is triggered & the result is available
  - get() can block
  - get() can also be timed/polled

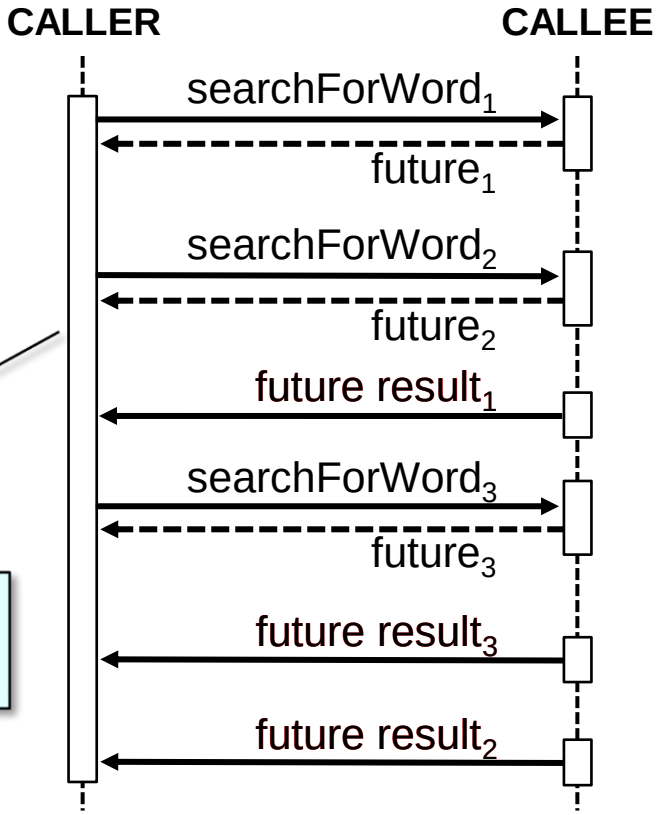


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get

Visualizing Java Futures in Action

- When the async call completes the future is triggered & the result is available
  - get() can block
  - get() can also be timed/polled

Computations can complete in a different order than the async calls were made



End of Visualizing Java Futures in Action