

Visualizing Java Futures in Action

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

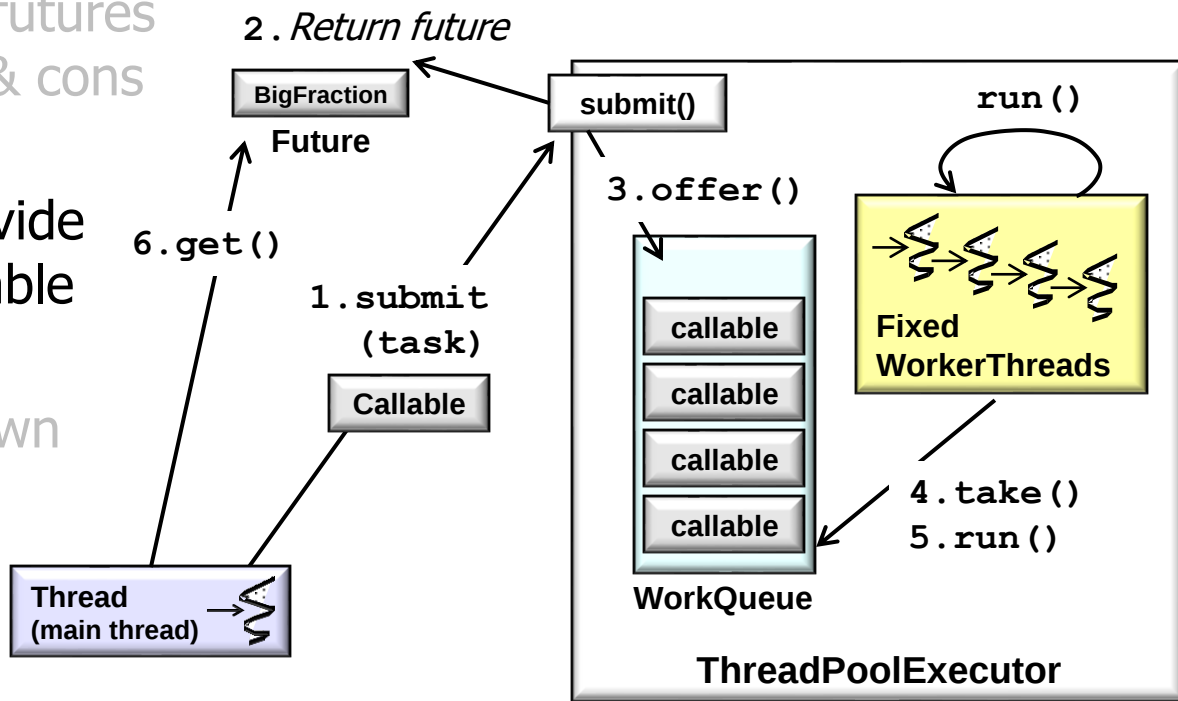
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

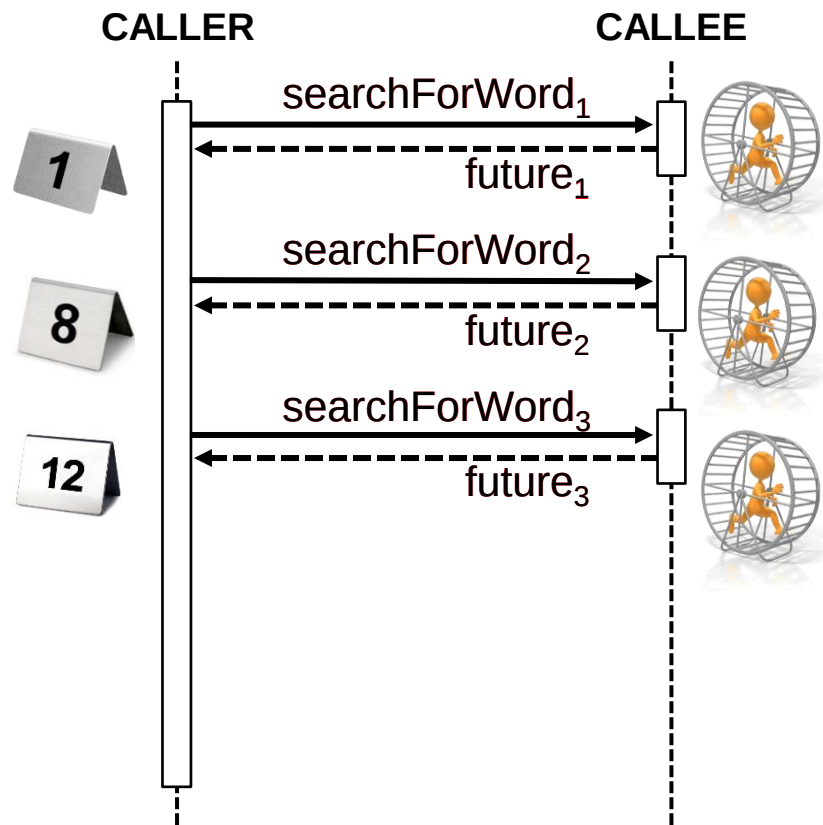
- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Know how Java futures provide the foundation for completable futures in Java
 - Understand a human known use of Java futures
 - Recognize the methods in the Future interface
 - Visualize Java futures in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



Visualizing Java Futures in Action

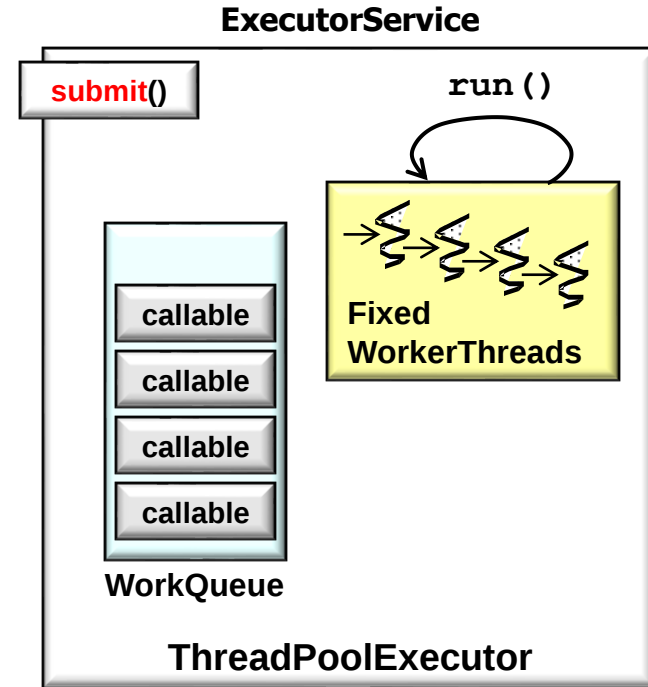
Visualizing Java Futures in Action

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



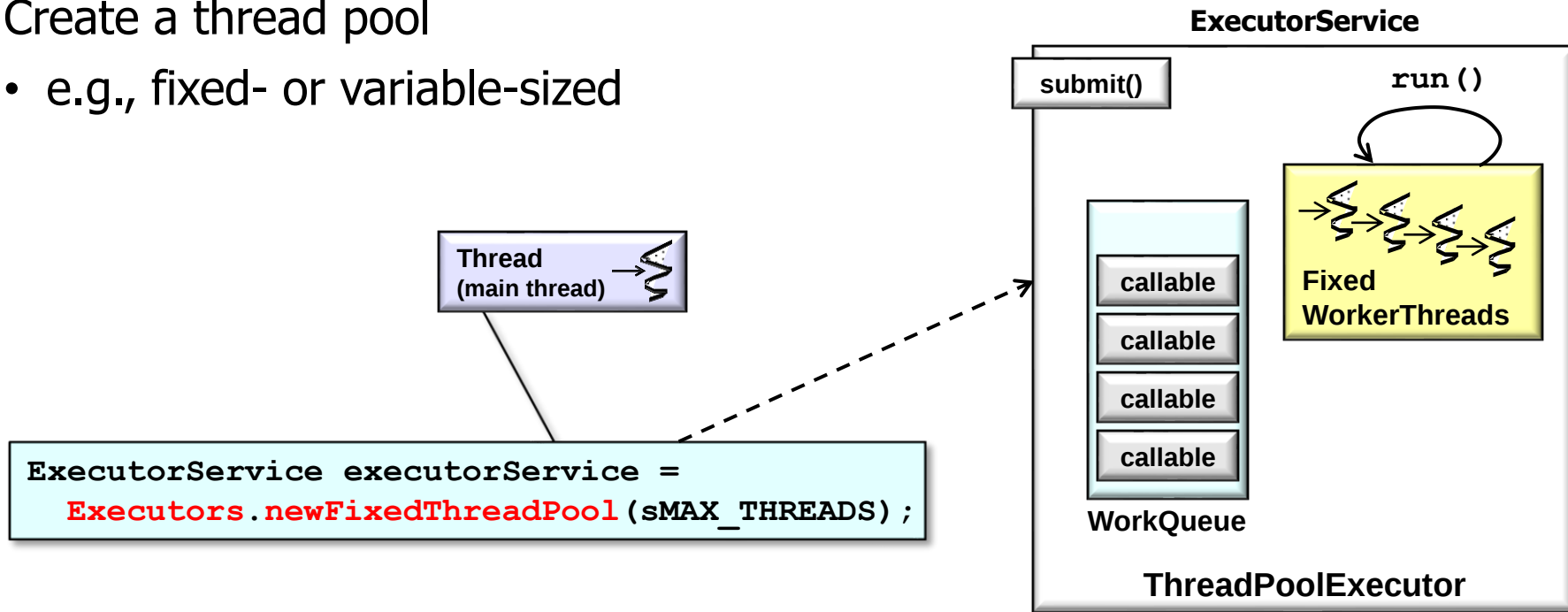
Visualizing Java Futures in Action

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



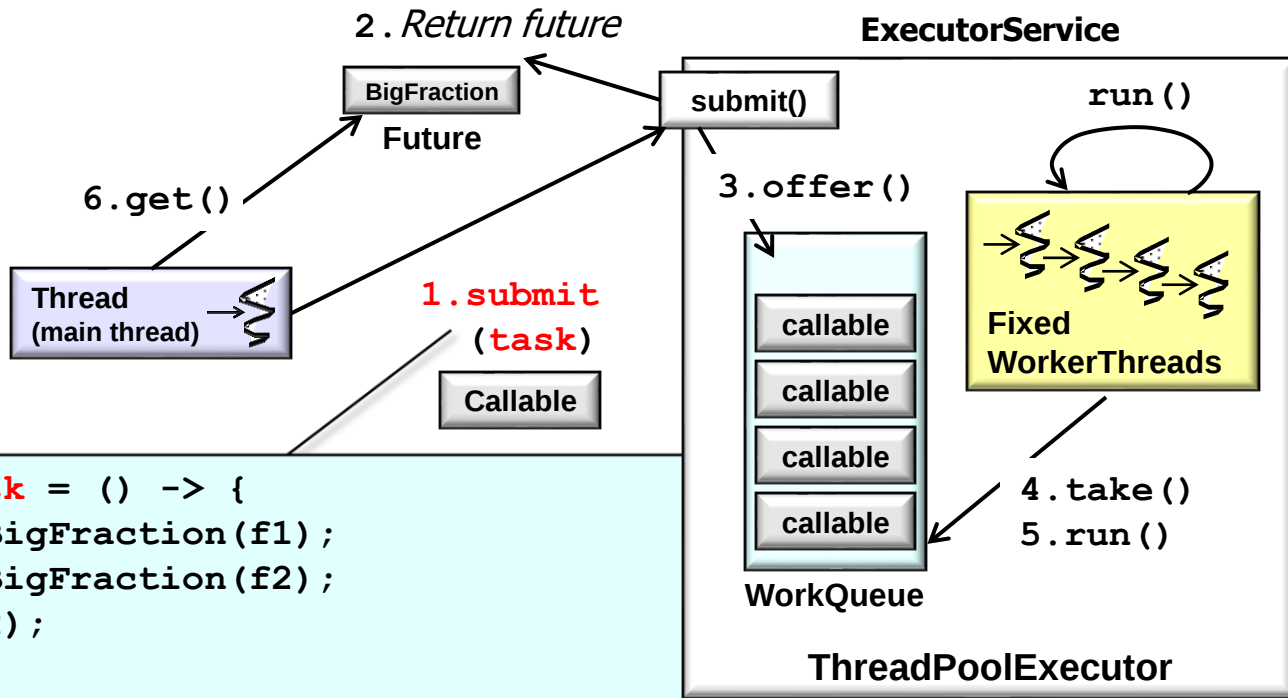
Visualizing Java Futures in Action

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



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



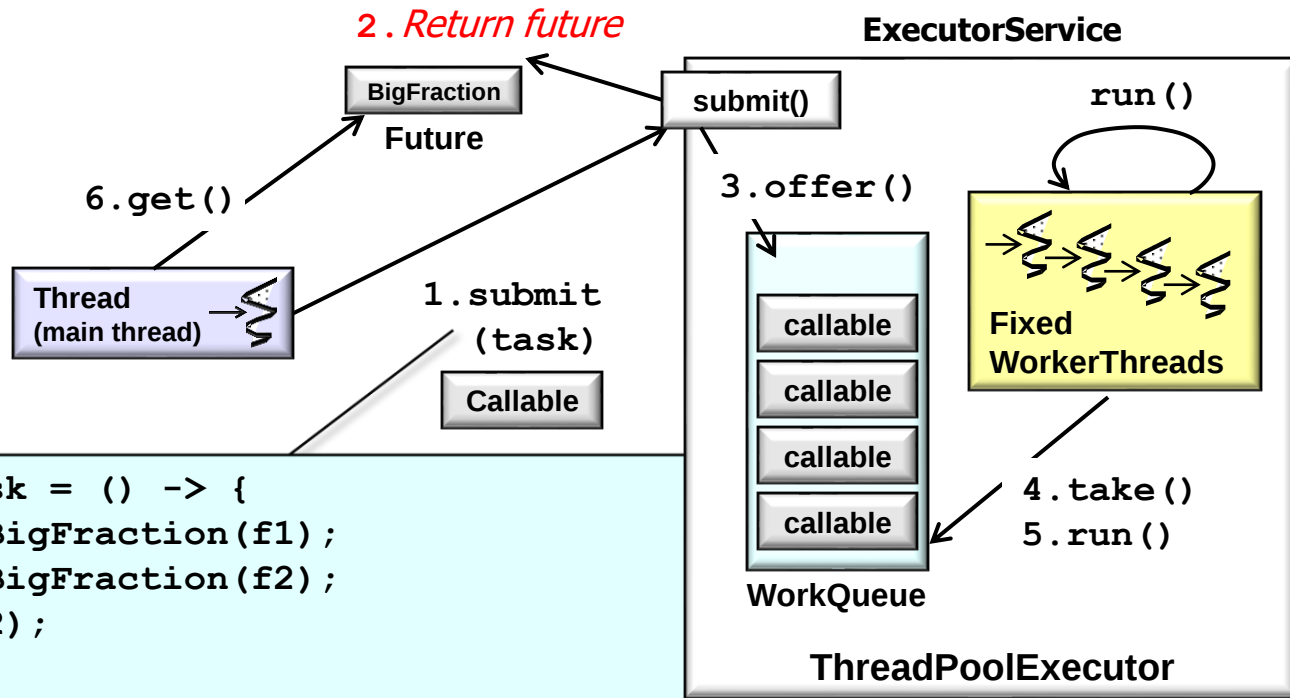
```
Callable<BigFraction> task = () -> {  
    BigFraction bf1 = new BigFraction(f1);  
    BigFraction bf2 = new BigFraction(f2);  
    return bf1.multiply(bf2);  
};
```

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

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`

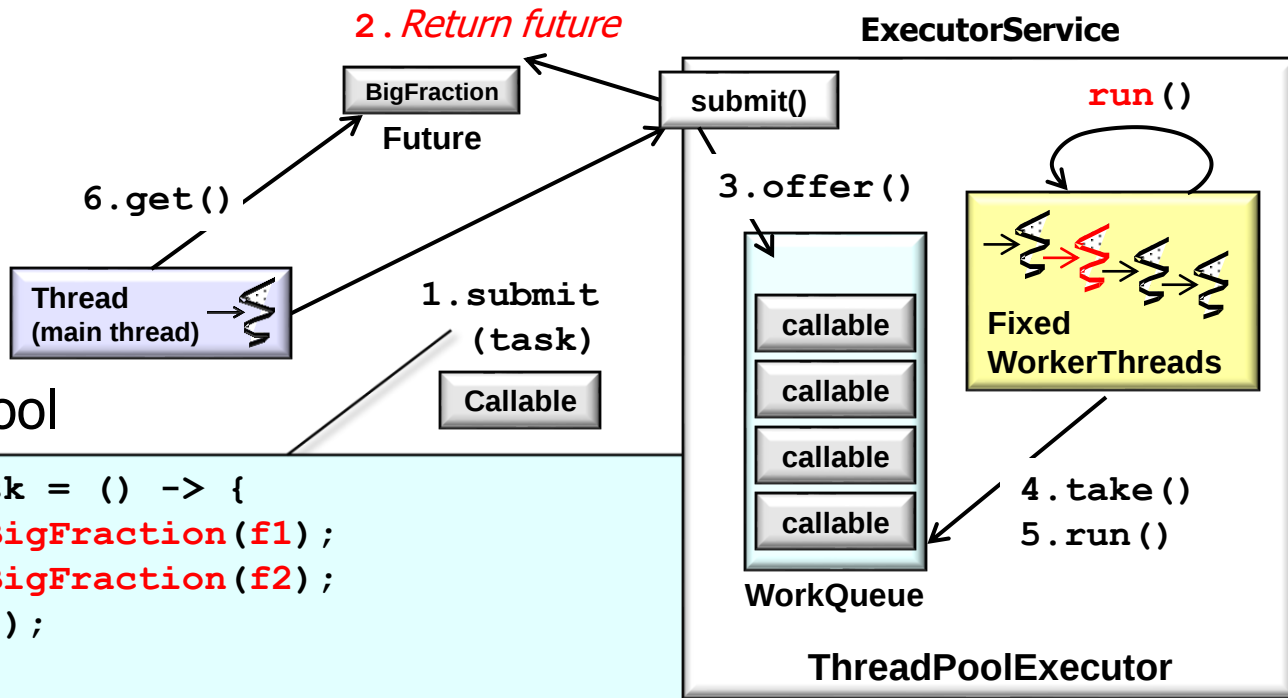


```
Callable<BigFraction> task = () -> {  
    BigFraction bf1 = new BigFraction(f1);  
    BigFraction bf2 = new BigFraction(f2);  
    return bf1.multiply(bf2);  
};
```

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

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

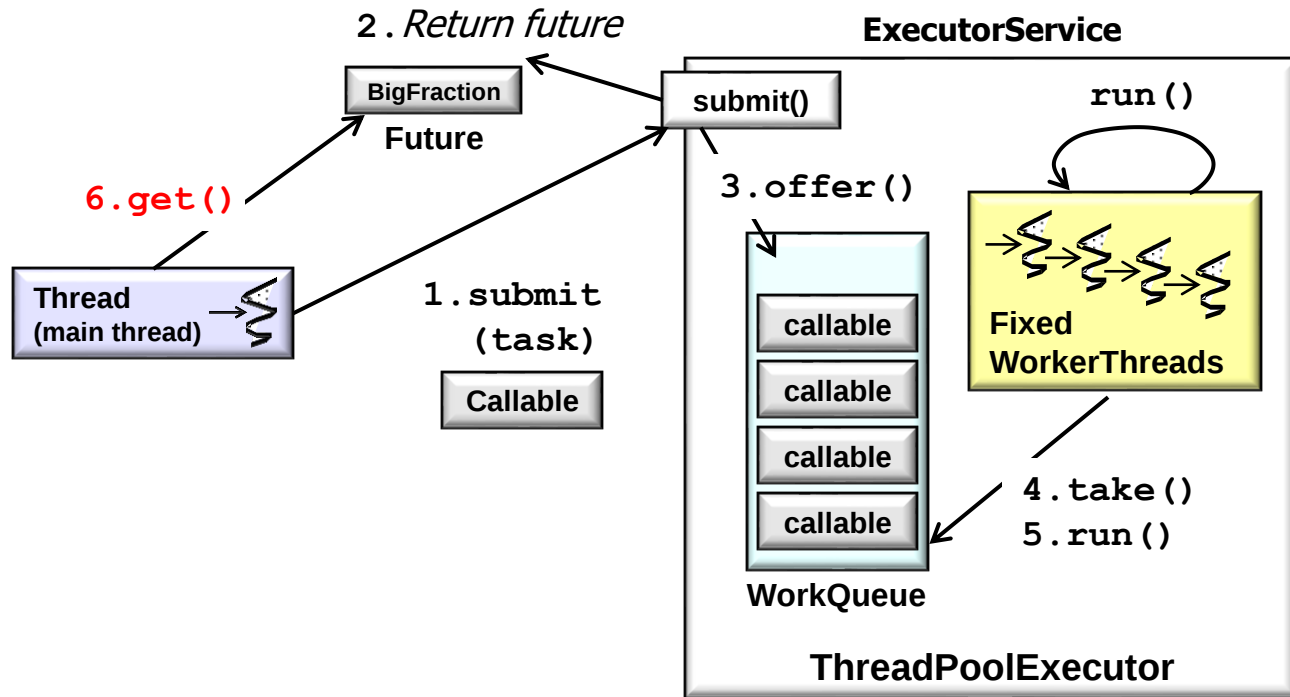


```
Callable<BigFraction> task = () -> {  
    BigFraction bf1 = new BigFraction(f1);  
    BigFraction bf2 = new BigFraction(f2);  
    return bf1.multiply(bf2);  
};
```

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

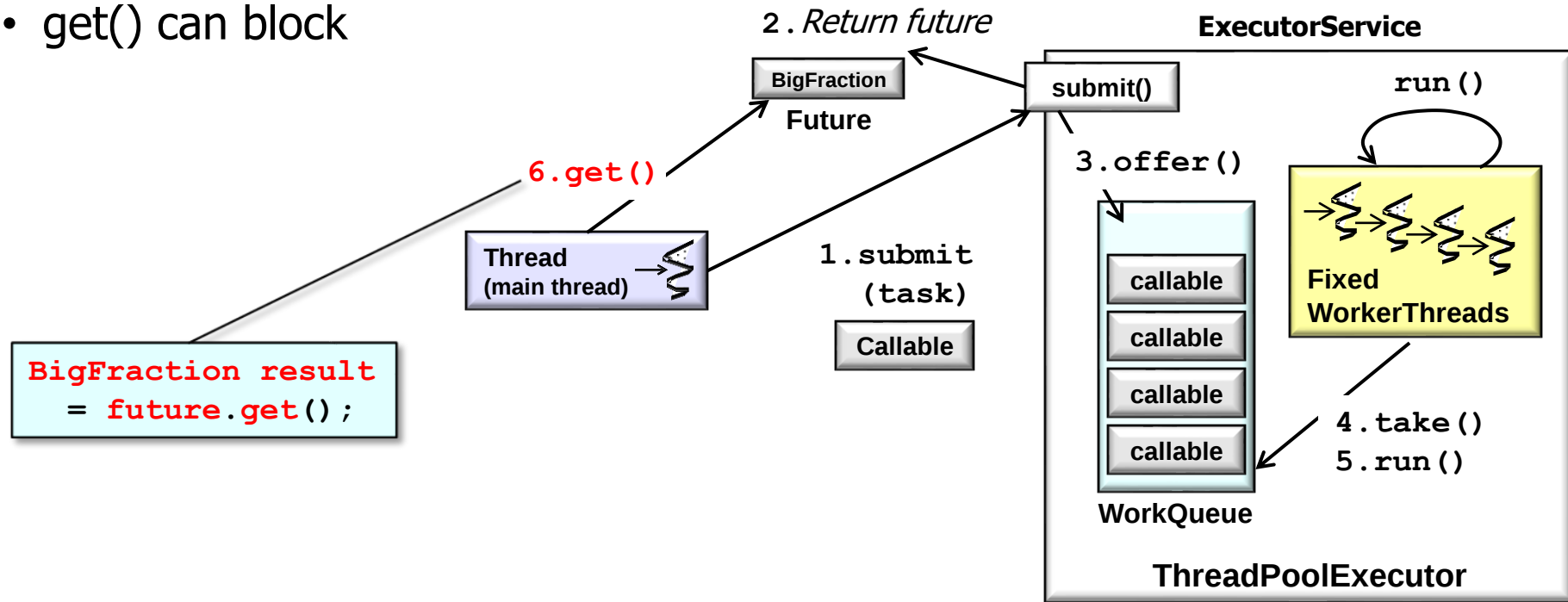
Visualizing Java Futures in Action

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



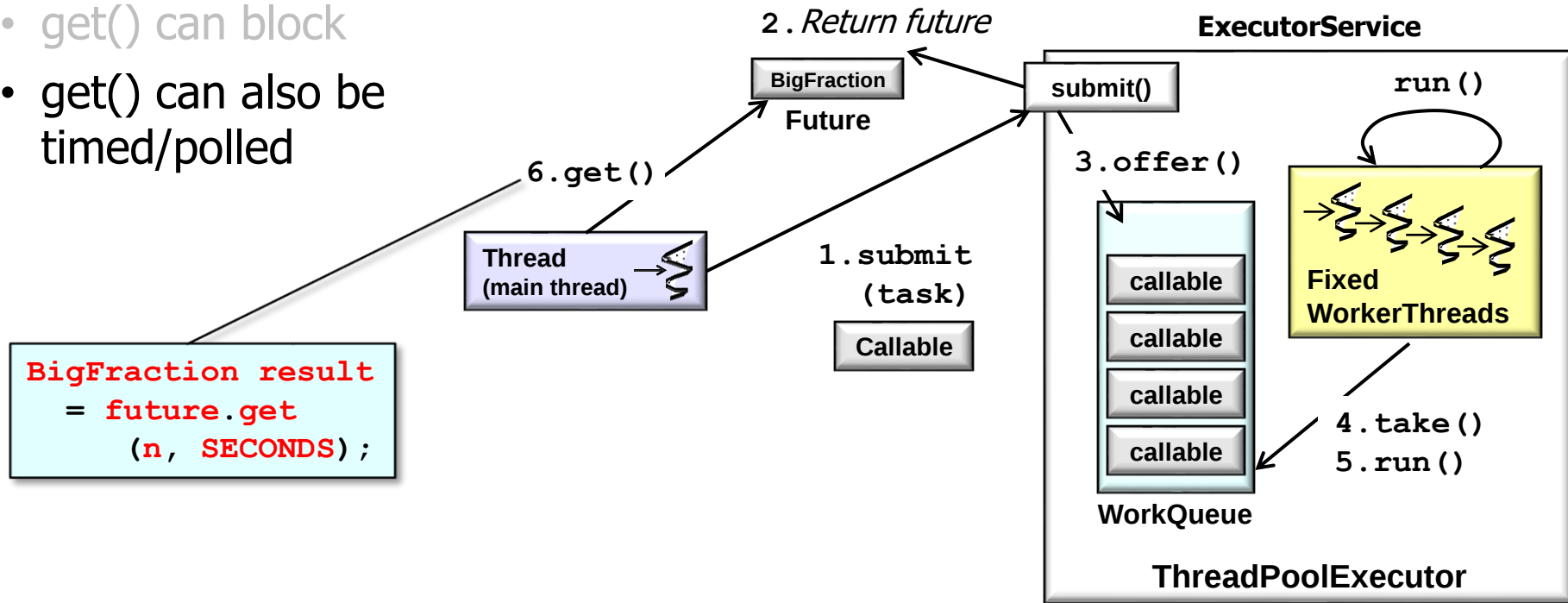
Visualizing Java Futures in Action

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



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/pollled

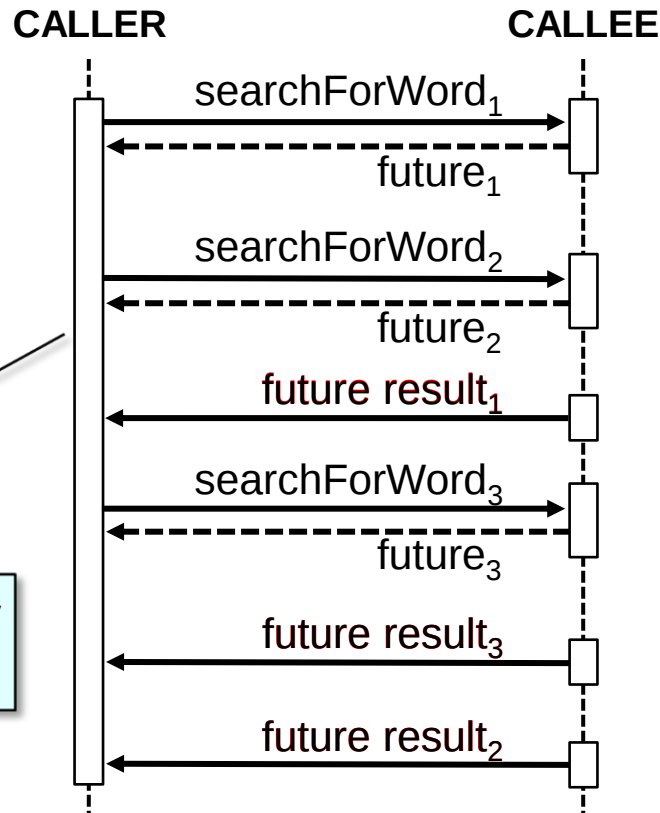


Visualizing Java Futures in Action

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

OUT OF ORDER

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



End of Visualizing Java Futures in Action