Overview of Java Futures

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

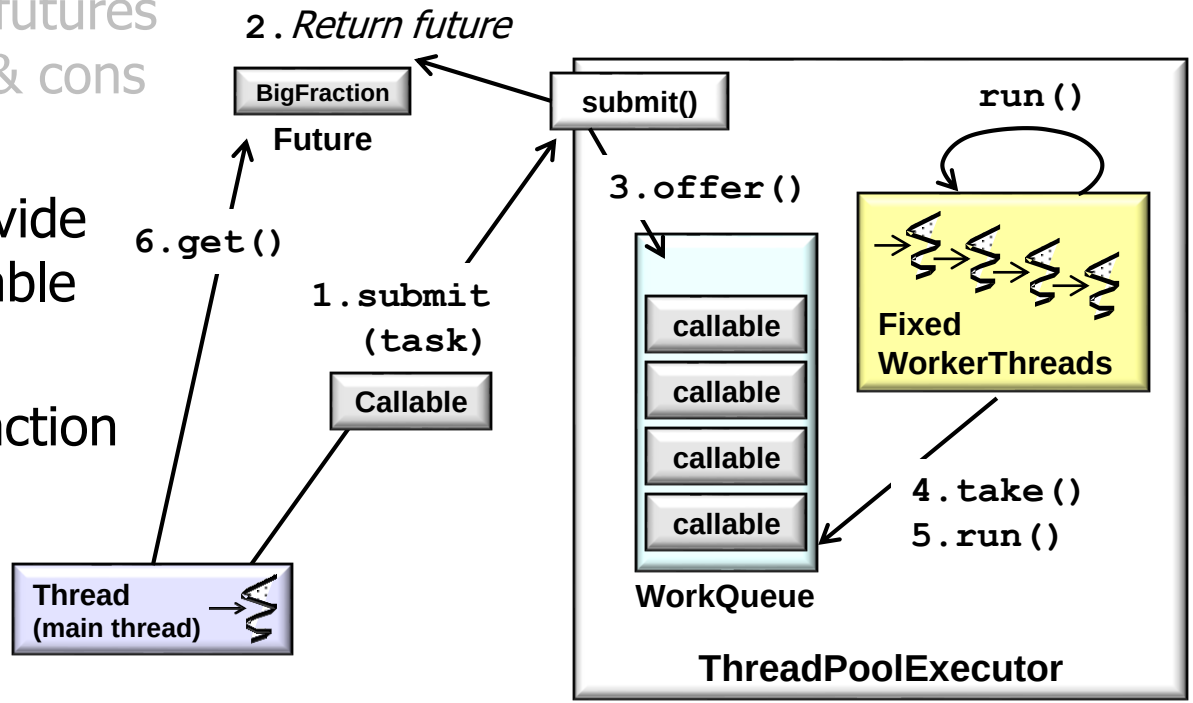
<<Java Class>>
CompletableFuture<T>

- CompletableFuture()
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long,TimeUnit)
- join()
- complete(T):boolean
- supplyAsync(Supplier<U>):CompletableFuture<U>
- supplyAsync(Supplier<U>,Executor):CompletableFuture<U>
- runAsync(Runnable):CompletableFuture<Void>
- runAsync(Runnable,Executor):CompletableFuture<Void>
- completedFuture(U):CompletableFuture<U>
- thenApply(Function<?>):CompletableFuture<U>
- thenAccept(Consumer<? super T>):CompletableFuture<Void>
- thenCombine(CompletionStage<? extends U>,BiFunction<?>):CompletableFuture<V>
- thenCompose(Function<?>):CompletableFuture<U>
- whenComplete(BiConsumer<?>):CompletableFuture<T>
- allOf(CompletableFuture[]<?>):CompletableFuture<Void>
- anyOf(CompletableFuture[]<?>):CompletableFuture<Object>

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

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
  - Visualize Java futures in action



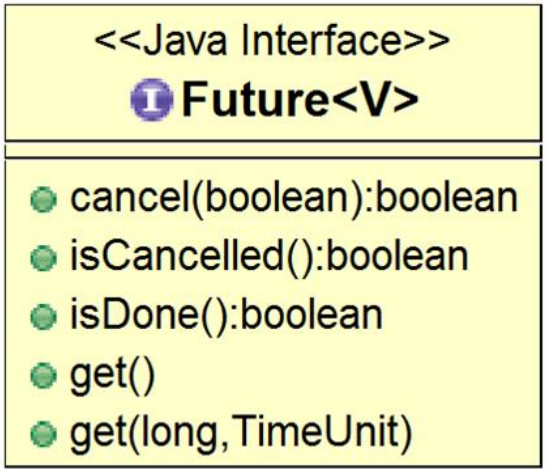
Learning Objectives in this Part of the Lesson

- 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
  - Visualize Java futures in action
  - Understand a human known use of Java futures

Overview of Java Futures

Overview of Java Futures

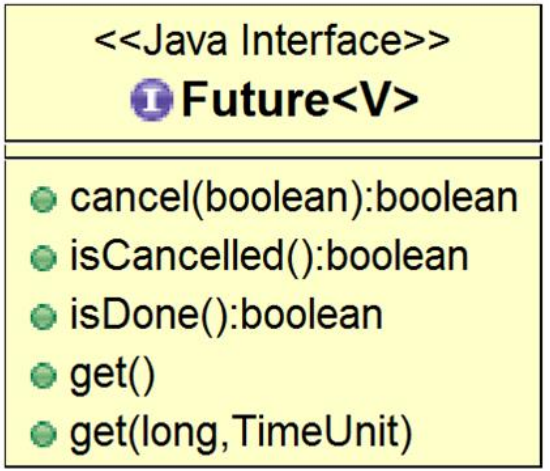
- Java 5 added async call support via the Java Future interface

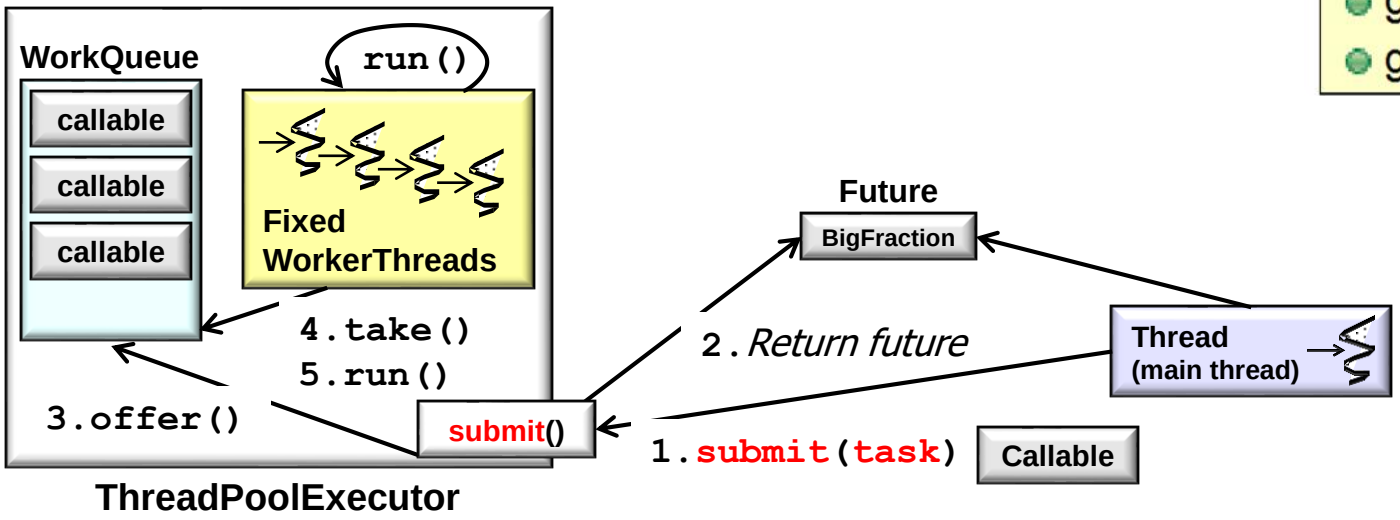


See en.wikipedia.org/wiki/Java_version_history

Overview of Java Futures

- Methods on Java Future can manage a task's lifecycle after it's submitted to run asynchronously

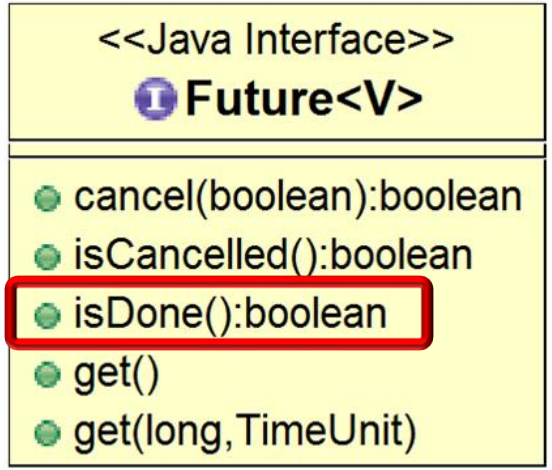


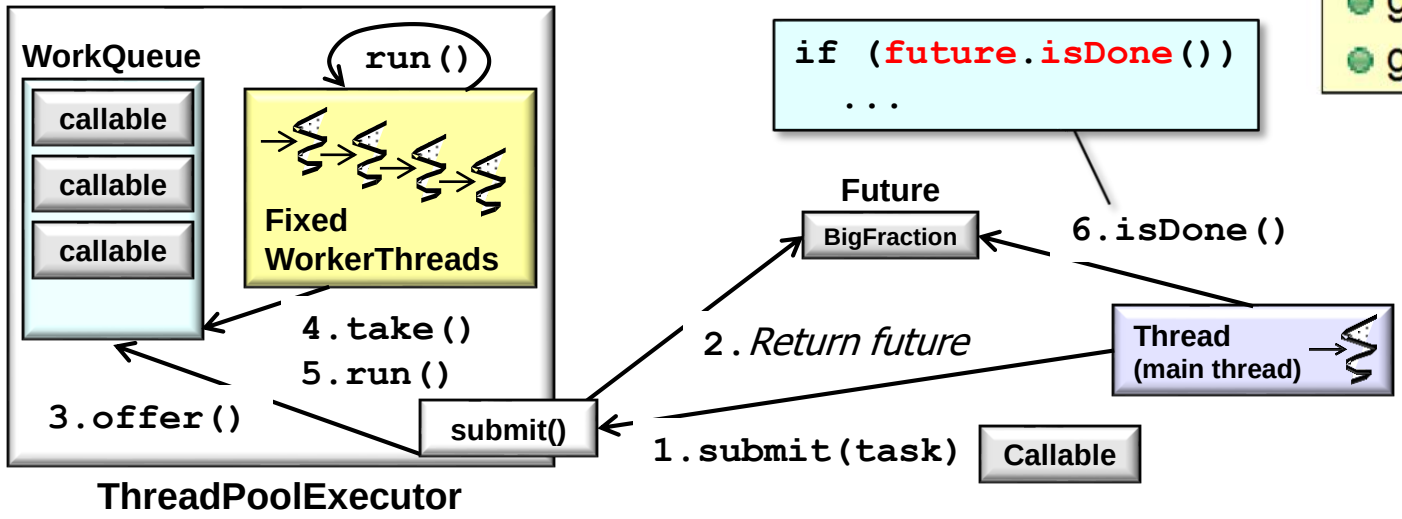


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html

Overview of Java Futures

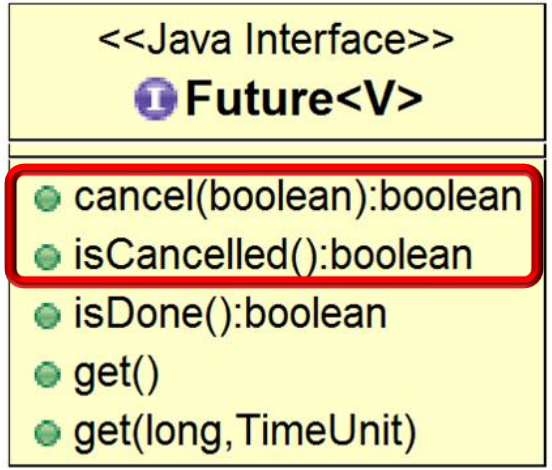
- Methods on Java Future can manage a task's lifecycle after it's submitted to run asynchronously, e.g.
  - A future can be tested for completion

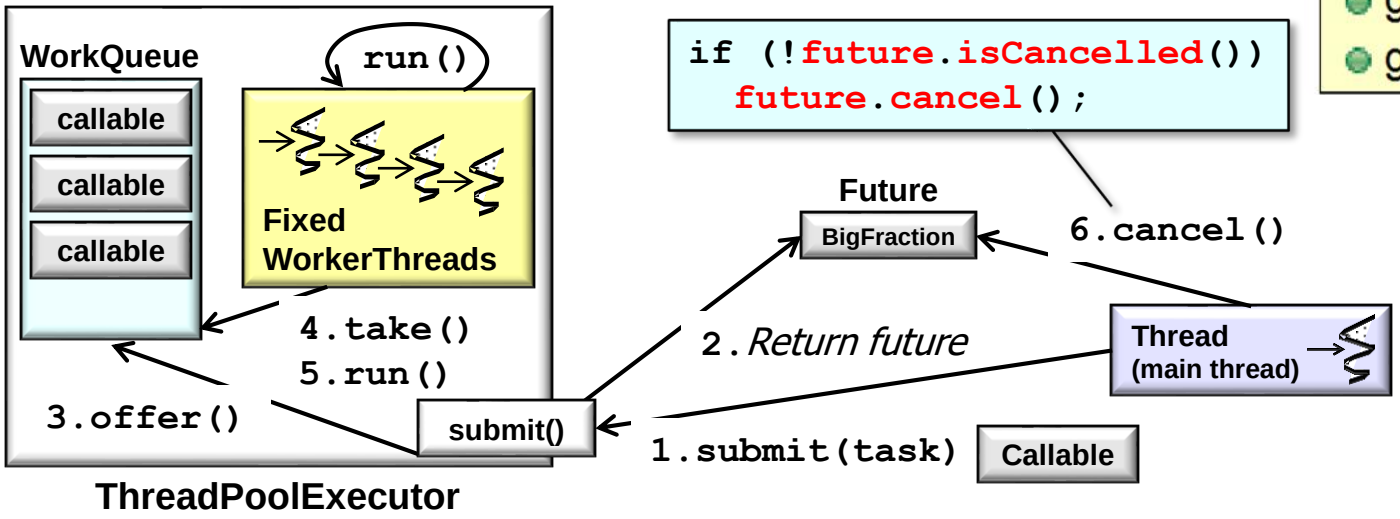




Overview of Java Futures

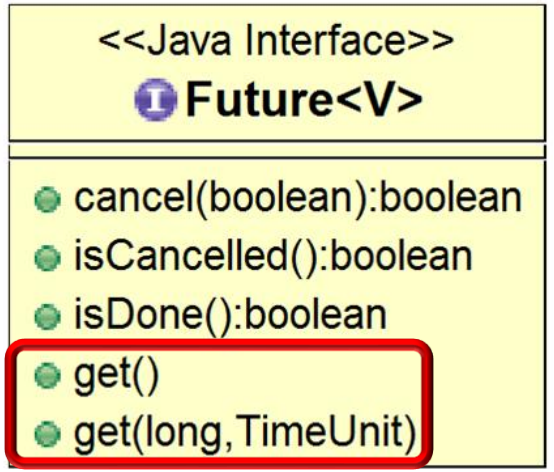
- Methods on Java Future can manage a task's lifecycle after it's submitted to run asynchronously, e.g.
  - A future can be tested for completion
  - A future be tested for cancellation & cancelled

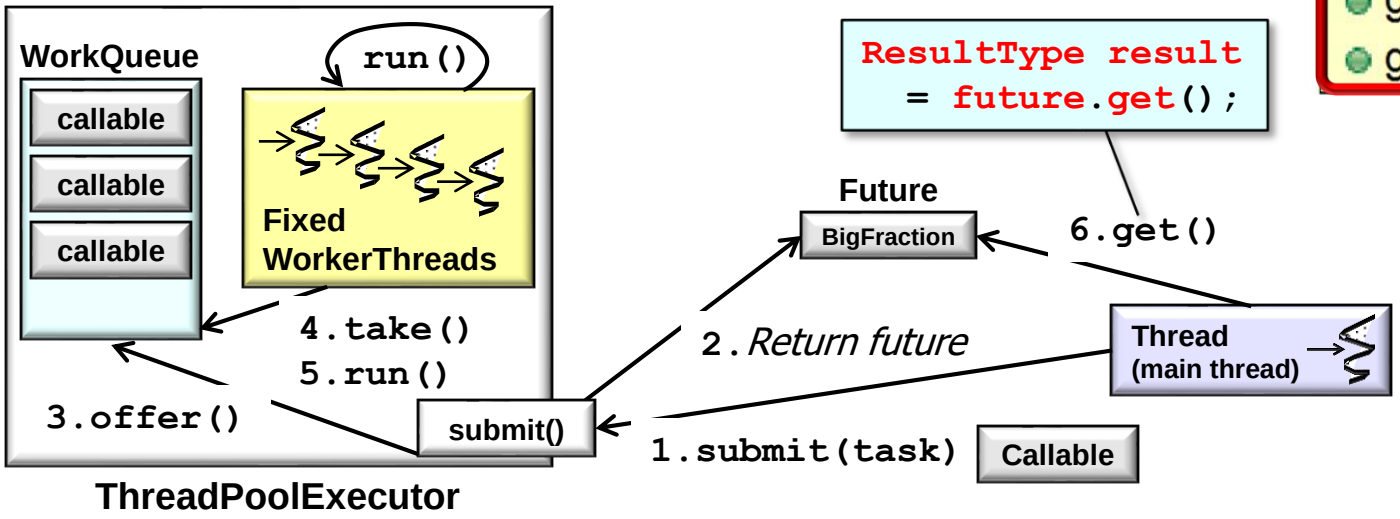




Overview of Java Futures

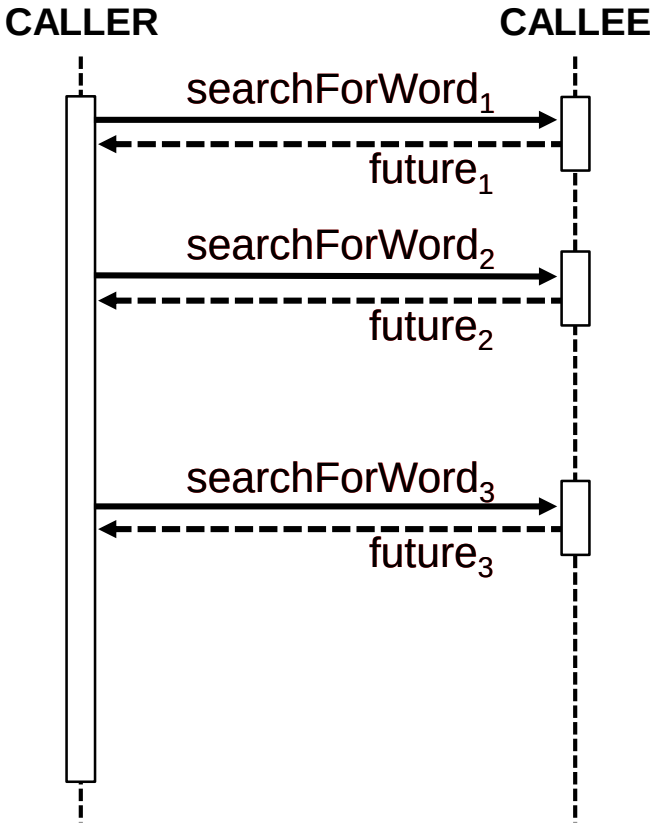
- Methods on Java Future can manage a task's lifecycle after it's submitted to run asynchronously, e.g.
  - A future can be tested for completion
  - A future be tested for cancellation & cancelled
  - A future can retrieve a two-way task's result





Overview of Java Futures

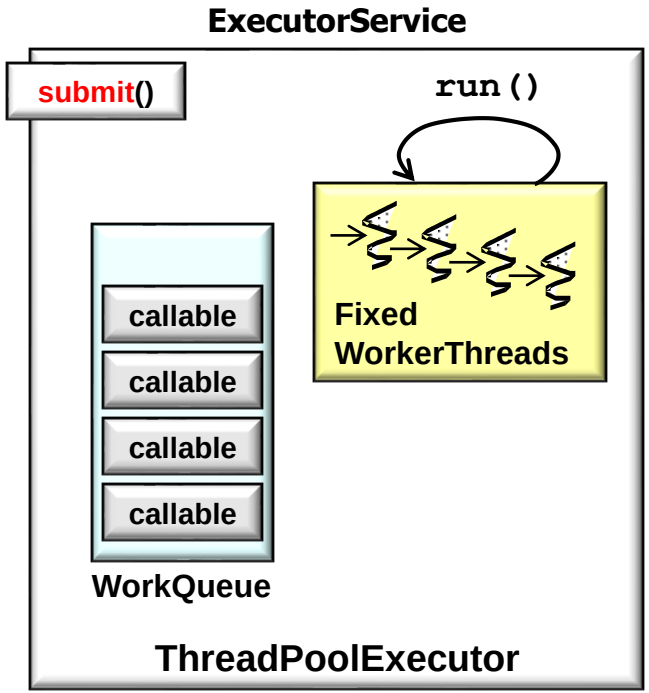
- An Java async call returns a future & continues running the computation in the background



Visualizing Java Futures

Visualizing Java Futures

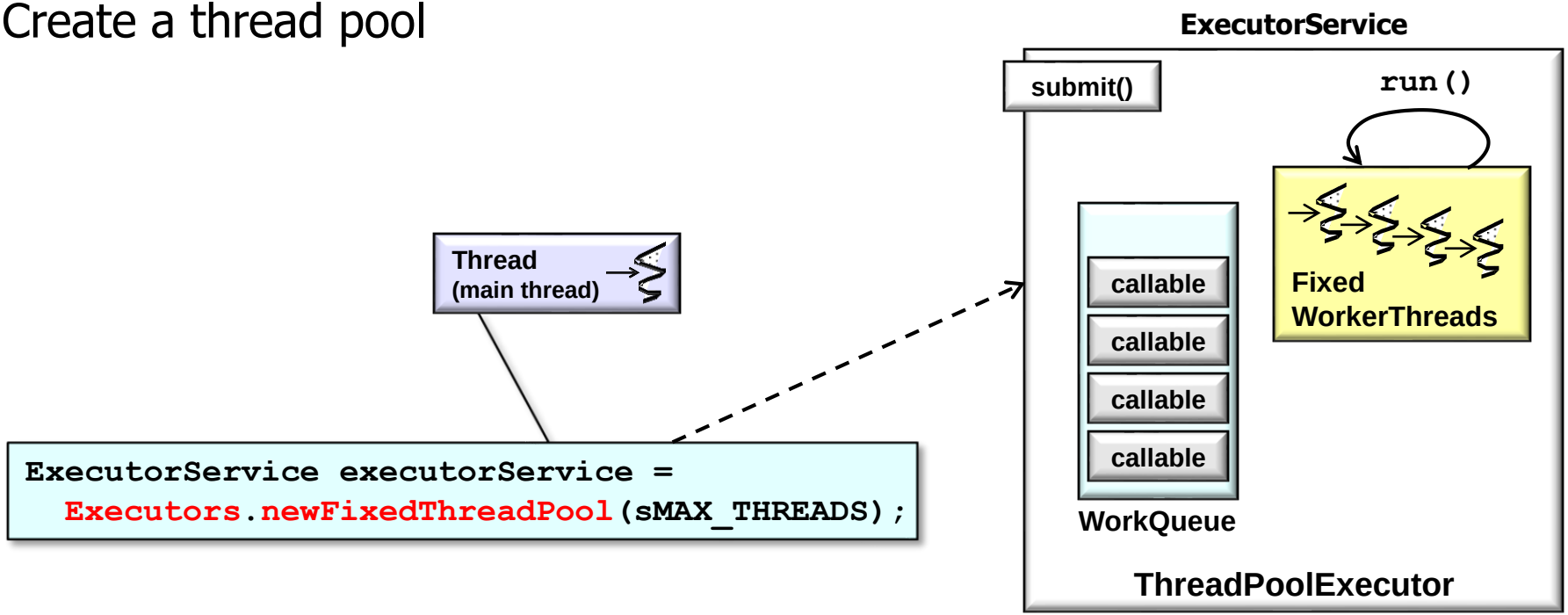
- ExecutorService.submit() can initiate an async call in Java



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

Visualizing Java Futures

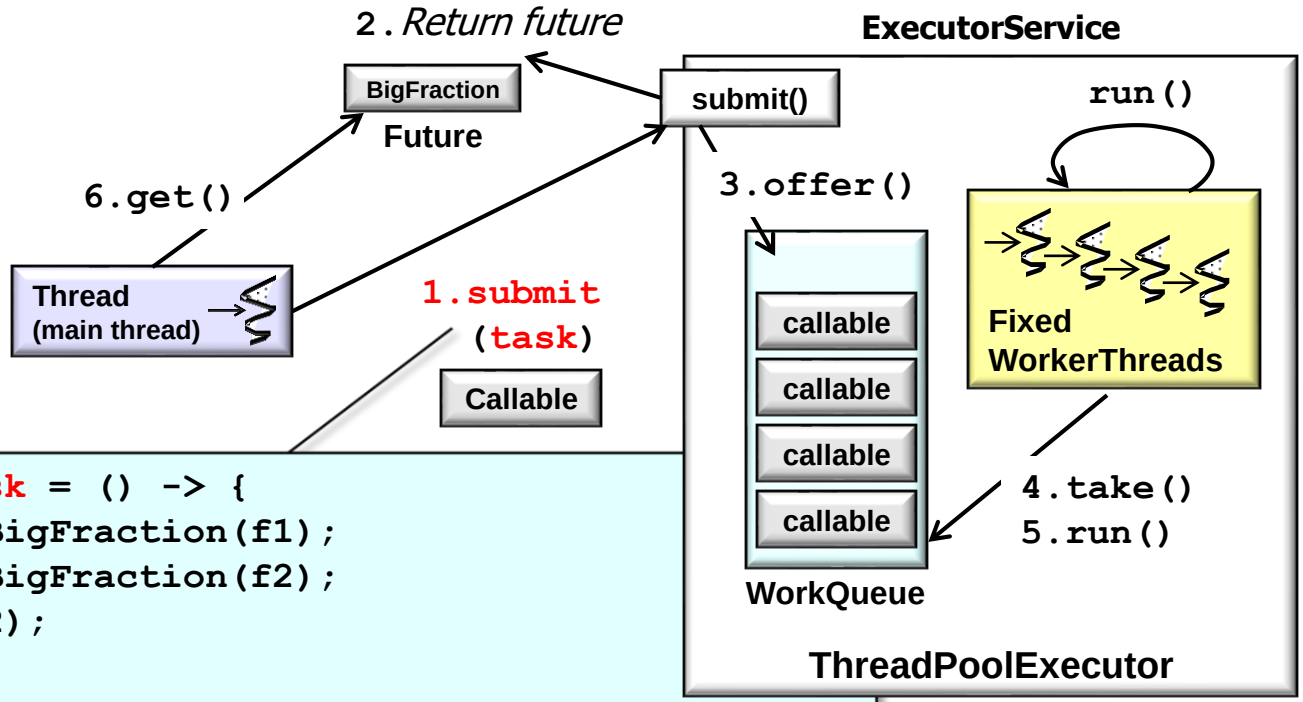
- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool



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

Visualizing Java Futures

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



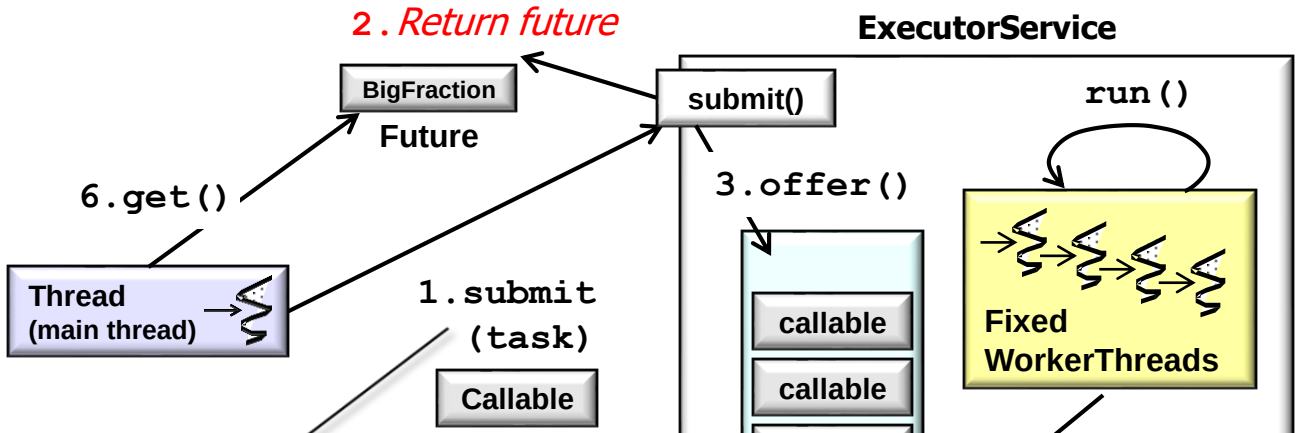

```
Callable<BigFraction> task = () -> {
  BigFraction bf1 = new BigFraction(f1);
  BigFraction bf2 = new BigFraction(f2);
  return bf1.multiply(bf2);
};

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

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

Visualizing Java Futures

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



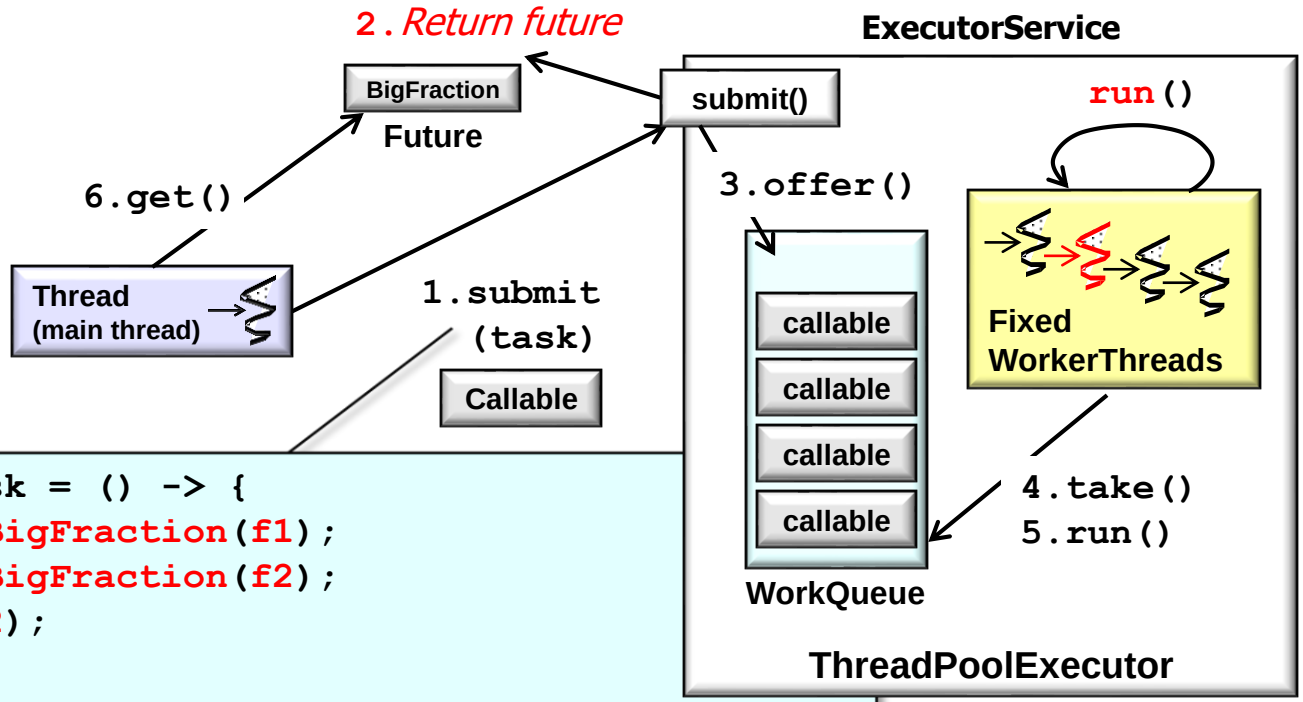

```
Callable<BigFraction> task = () -> {
  BigFraction bf1 = new BigFraction(f1);
  BigFraction bf2 = new BigFraction(f2);
  return bf1.multiply(bf2);
};

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

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

Visualizing Java Futures

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

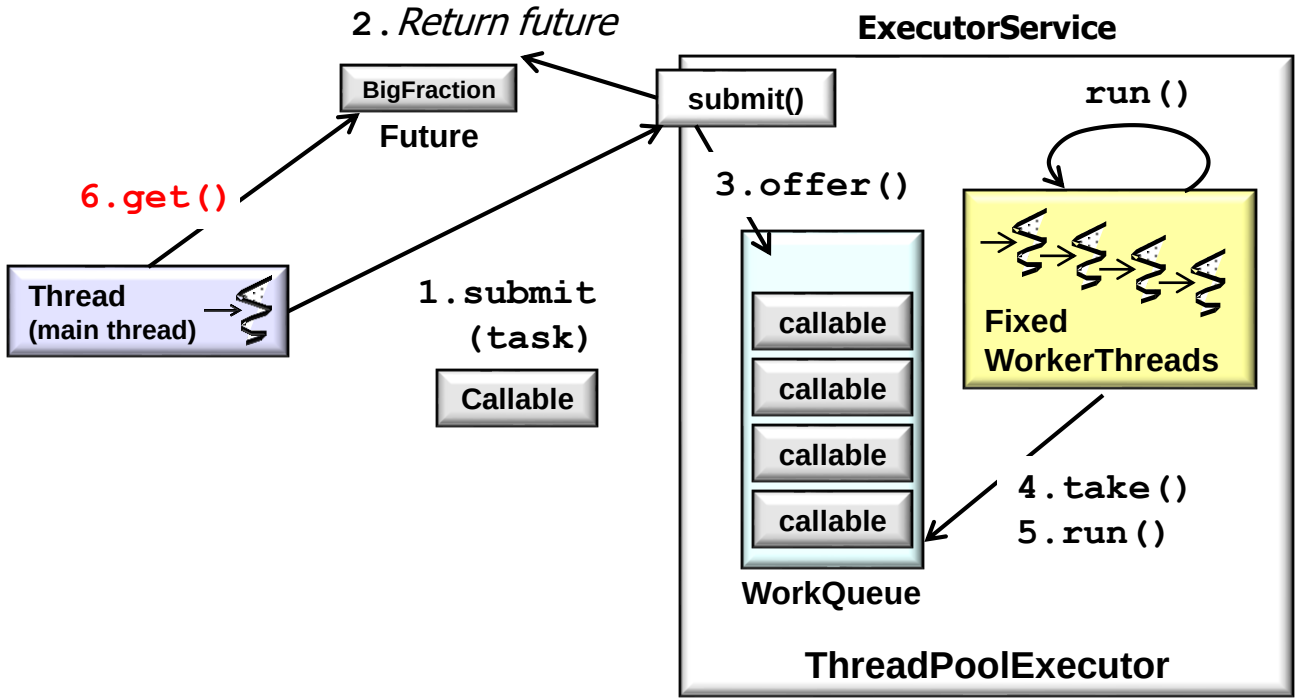



```
Callable<BigFraction> task = () -> {
  BigFraction bf1 = new BigFraction(f1);
  BigFraction bf2 = new BigFraction(f2);
  return bf1.multiply(bf2);
};

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

Visualizing Java Futures

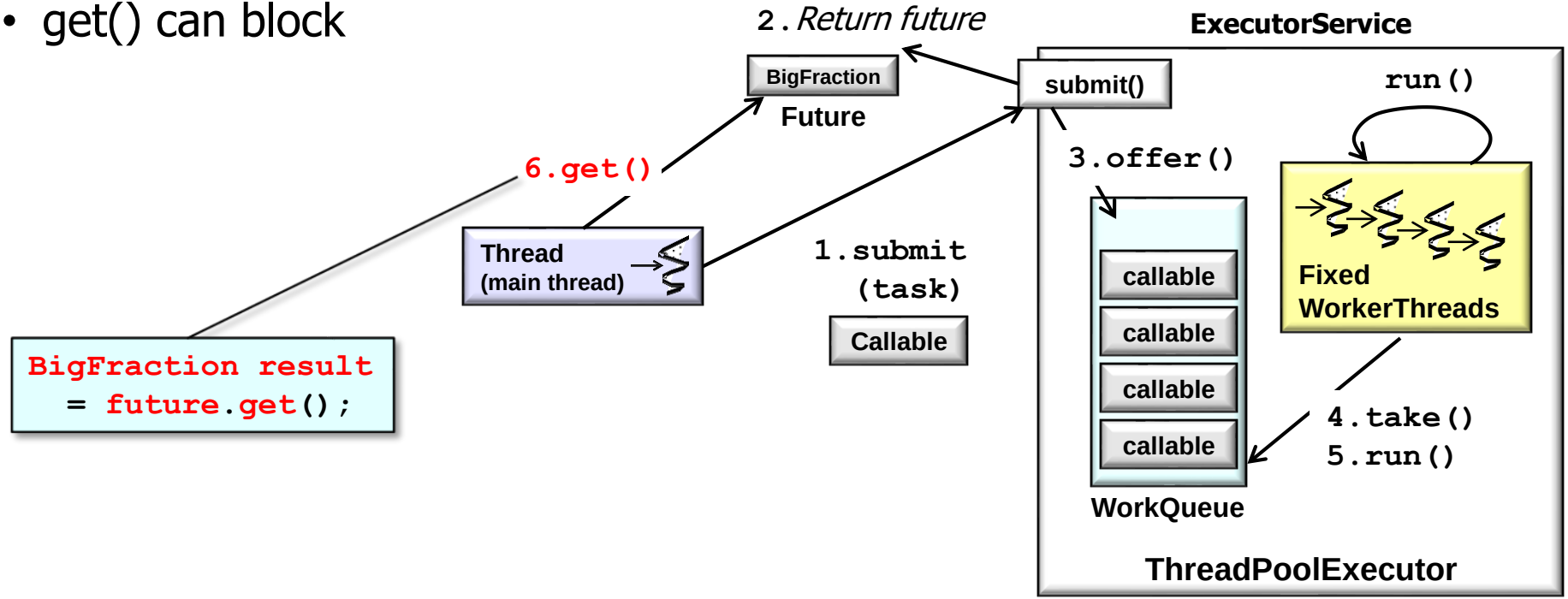
- When the async call completes the future is triggered & the result is available



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

Visualizing Java Futures

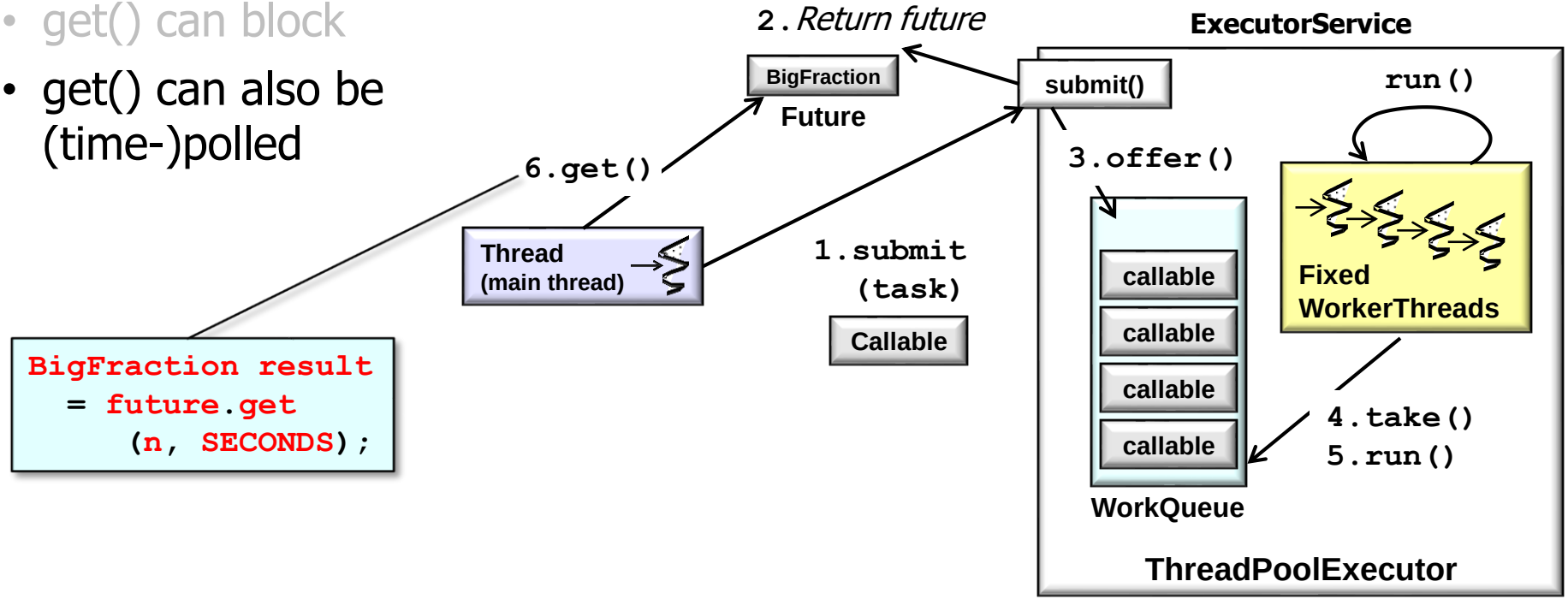
- When the async call completes the future is triggered & the result is available
  - get() can block



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

Visualizing Java Futures

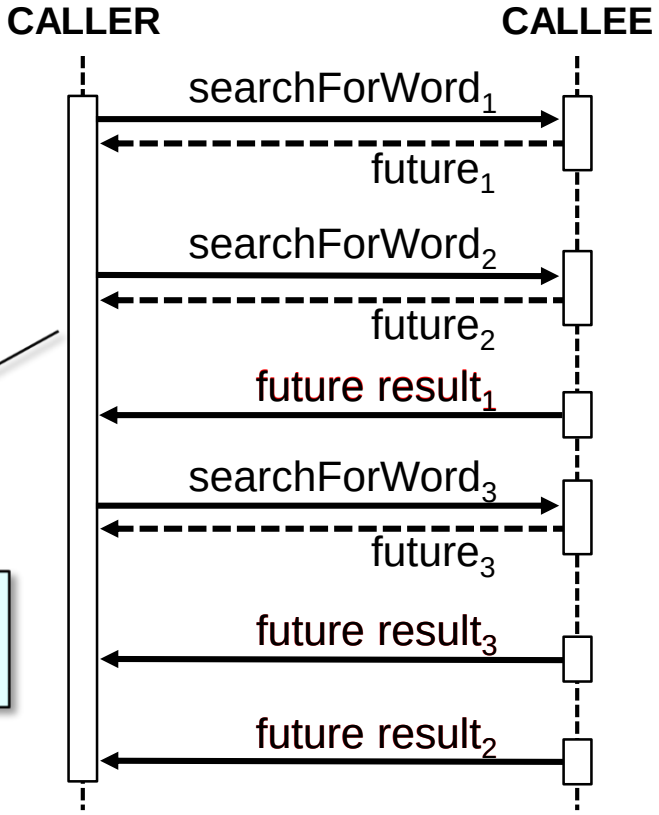
- When the async call completes the future is triggered & the result is available
  - get() can block
  - get() can also be (time-)polled



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

Visualizing Java Futures

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

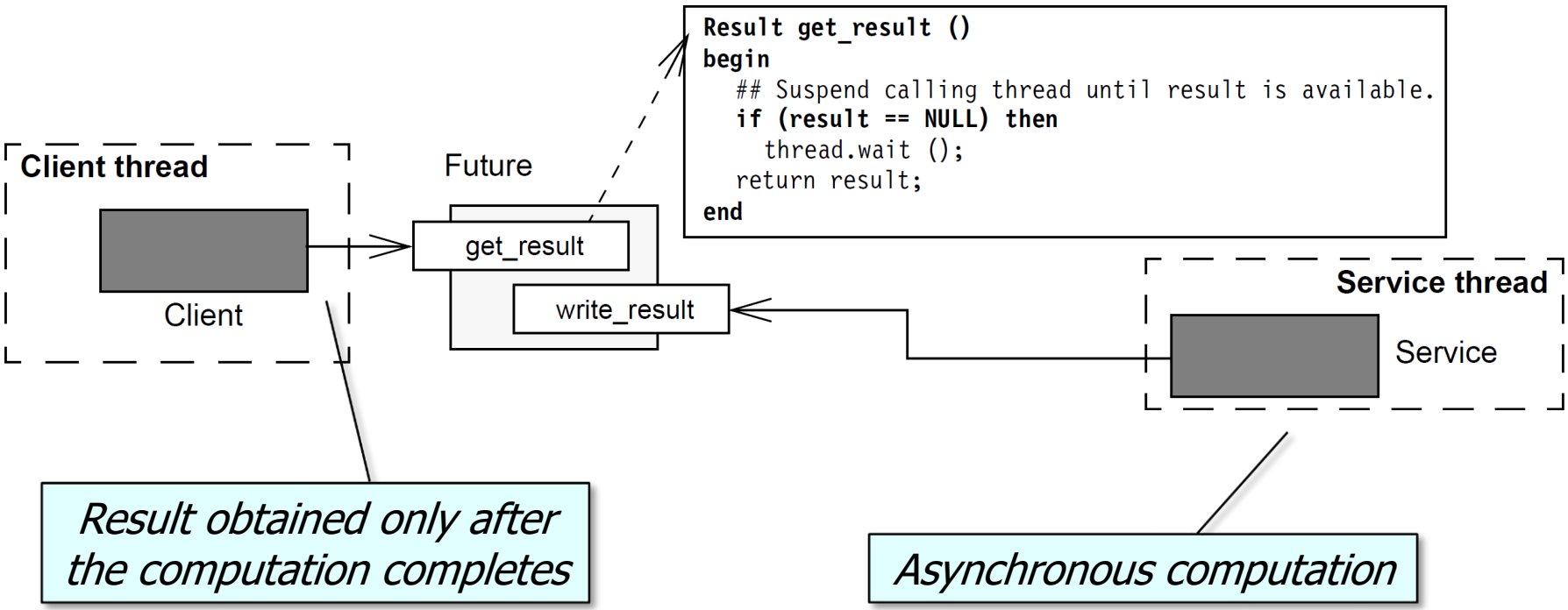


Computations can complete in a different order than the async calls were made

A Human Known Use of Java Futures

A Human Known Use of Java Futures

- A future is essentially a proxy that represents the result(s) of an async call



See en.wikipedia.org/wiki/Futures_and_promises

A Human Known Use of Java Futures

- A future is essentially a proxy that represents the result(s) of an async call

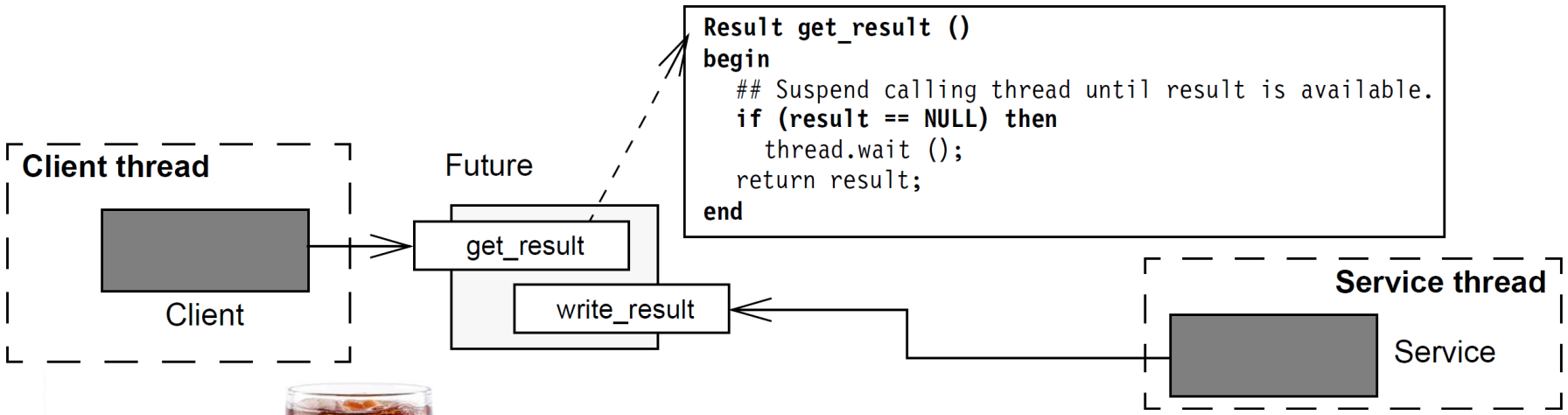


Table tent #'s are a human-known-use of futures!

See www.citygrafx.com/table-numbers-table-markers

A Human Known Use of Java Futures

- A future is essentially a proxy that represents the result(s) of an async call

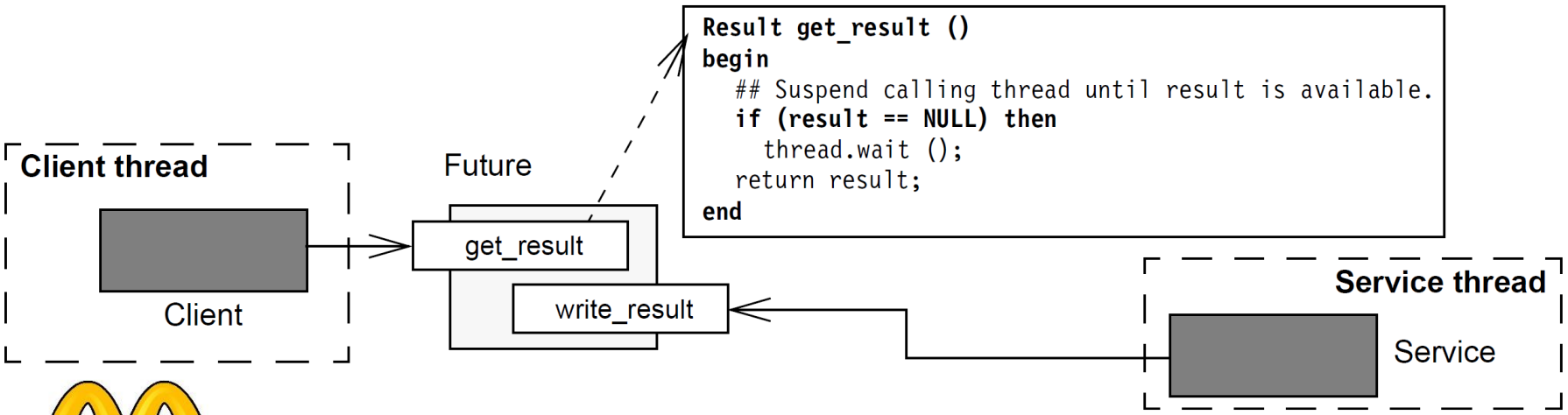


Table tent #'s are a human-known-use of futures!

e.g., McDonald's vs Wendy's model of preparing fast food

End of Overview of Java Futures