Key Methods in the Java ExecutorCompletionService

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

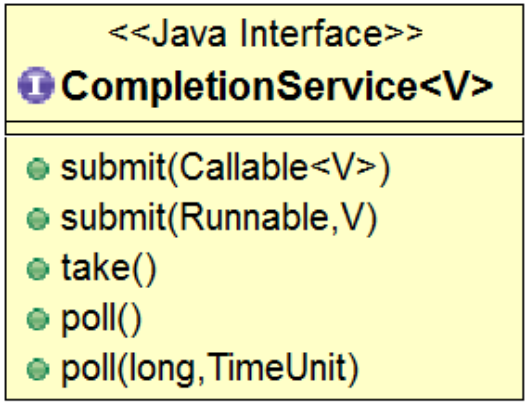
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

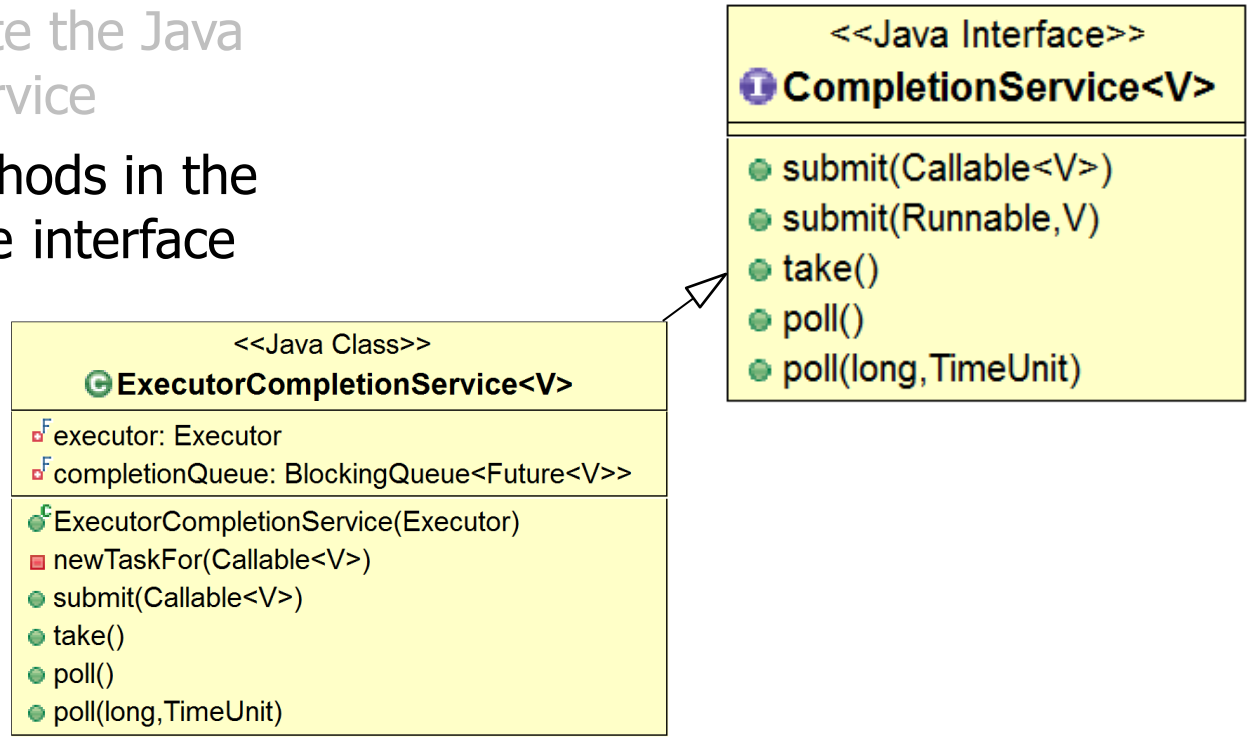
Learning Objectives in this Part of the Lesson

- Understand how the Java CompletionService interface defines a framework for handling the completion of asynchronous tasks
- Know how to instantiate the Java ExecutorCompletionService
- Recognize the key methods in the Java CompletionService interface



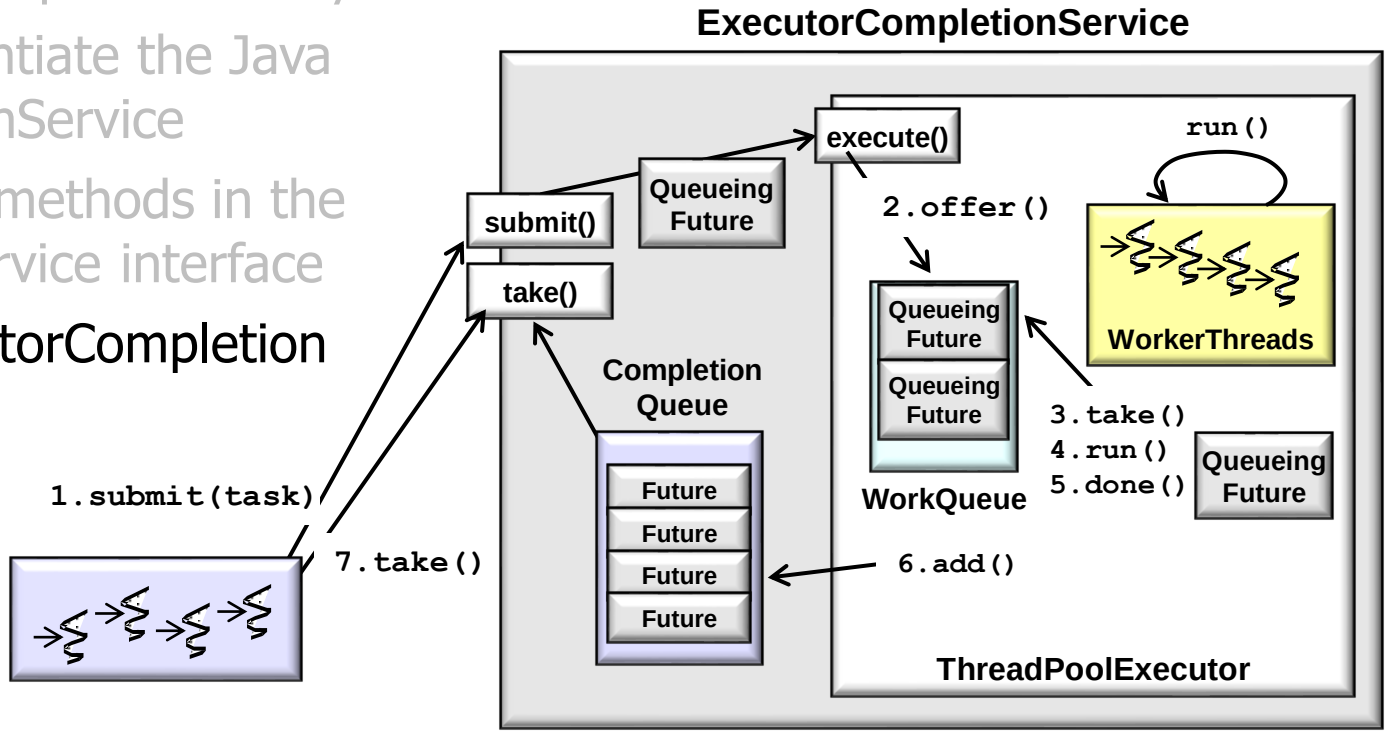
Learning Objectives in this Part of the Lesson

- Understand how the Java CompletionService interface defines a framework for handling the completion of asynchronous tasks
- Know how to instantiate the Java ExecutorCompletionService
- Recognize the key methods in the Java CompletionService interface
  - These methods are implemented by the ExecutorCompletion Service class



Learning Objectives in this Part of the Lesson

- Understand how the Java CompletionService interface defines a framework for handling the completion of asynchronous tasks
- Know how to instantiate the Java ExecutorCompletionService
- Recognize the key methods in the Java CompletionService interface
- Visualize the ExecutorCompletion Service in action



Key Methods in the CompletionService Interface

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods

Interface CompletionService<V>

All Known Implementing Classes:
`ExecutorCompletionService`

`public interface CompletionService<V>`

A service that decouples the production of new asynchronous tasks from the consumption of the results of completed tasks. Producers `submit` tasks for execution. Consumers `take` completed tasks and process their results in the order they complete. A `CompletionService` can for example be used to manage asynchronous I/O, in which tasks that perform reads are submitted in one part of a program or system, and then acted upon in a different part of the program when the reads complete, possibly in a different order than they were requested.

Typically, a `CompletionService` relies on a separate `Executor` to actually execute the tasks, in which case the `CompletionService` only manages an internal completion queue. The `ExecutorCompletionService` class provides an implementation of this approach.

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

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }


  public Future<V>
    submit(Runnable task,
           V result) {
    ...
  }
  ...
```

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
    - Submit a value-returning two-way task


```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }


  public Future<V>
    submit(Runnable task,
           V result) {
    ...
  }
  ...
```

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
    - Submit a value-returning two-way task

```
public interface Callable<V> {
  V call() throws Exception;
}
```

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }

  public Future<V>
    submit(Runnable task,
           V result) {
    ...
  }
  ...
```

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

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
    - Submit a value-returning two-way task
      - Provides an "asynchronous future" processing model

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }

  public Future<V>
    submit(Runnable task,
           V result) {
    ...
  }
  ...
```

Return values of submit() are typically ignored

i.e., no need to block on the future

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
    - Submit a value-returning two-way task
      - Provides an "asynchronous future" processing model
    - The main reason to access this future is to cancel the async computation


```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }


  public Future<V>
    submit(Runnable task,
           V result) {
    ...
  }
  ...
```

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
    - Submit a value-returning two-way task
  - Submit a one-way task that returns nothing


```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }


  public Future<V>
    submit(Runnable task,
           V result) {
    /* ... */
  }
  ...
```

Not as widely used as the two-way callable task

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
    - Submit a value-returning two-way task
  - Submit a one-way task that returns nothing

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  ...
  public Future<V>
    submit(Callable<V> task) {
    ...
  }


  public Future<V>
    submit(Runnable task,
           V result) {
    /* ... */
  }
  ...
```

```
public interface Runnable {
  void run();
}
```

See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
...
 public Future<V> take() ... {
    ...
 }

 public Future<V> poll() {
    ...
 }

 public Future<V> poll(long
    timeout, TimeUnit unit) ... {
    ...
 } ...
```

These methods access an internal blocking queue containing Queueing Futures whose tasks have completed

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

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results

get() never blocks on a future removed from the internal queue!

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
...
 public Future<V> take() ... {
    ...
 }

 public Future<V> poll() {
    ...
 }

 public Future<V> poll(long
    timeout, TimeUnit unit) ... {
    ...
 } ...
```

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results
    - Block until a future for next completed task is available & then retrieve/remove it

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
...
 public Future<V> take() ... {
    ...
 }

 public Future<V> poll() {
    ...
 }

 public Future<V> poll(long
    timeout, TimeUnit unit) ... {
    ...
 } ...
```

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results
    - Block until a future for next completed task is available & then retrieve/remove it
    - Retrieve/remove a future for the next completed task or null if none are available

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
...
 public Future<V> take() ... {
    ...
 }

 public Future<V> poll() {
    ...
 }

 public Future<V> poll(long
    timeout, TimeUnit unit) ... {
    ...
 } ...
```

Key Methods in the CompletionService Interface

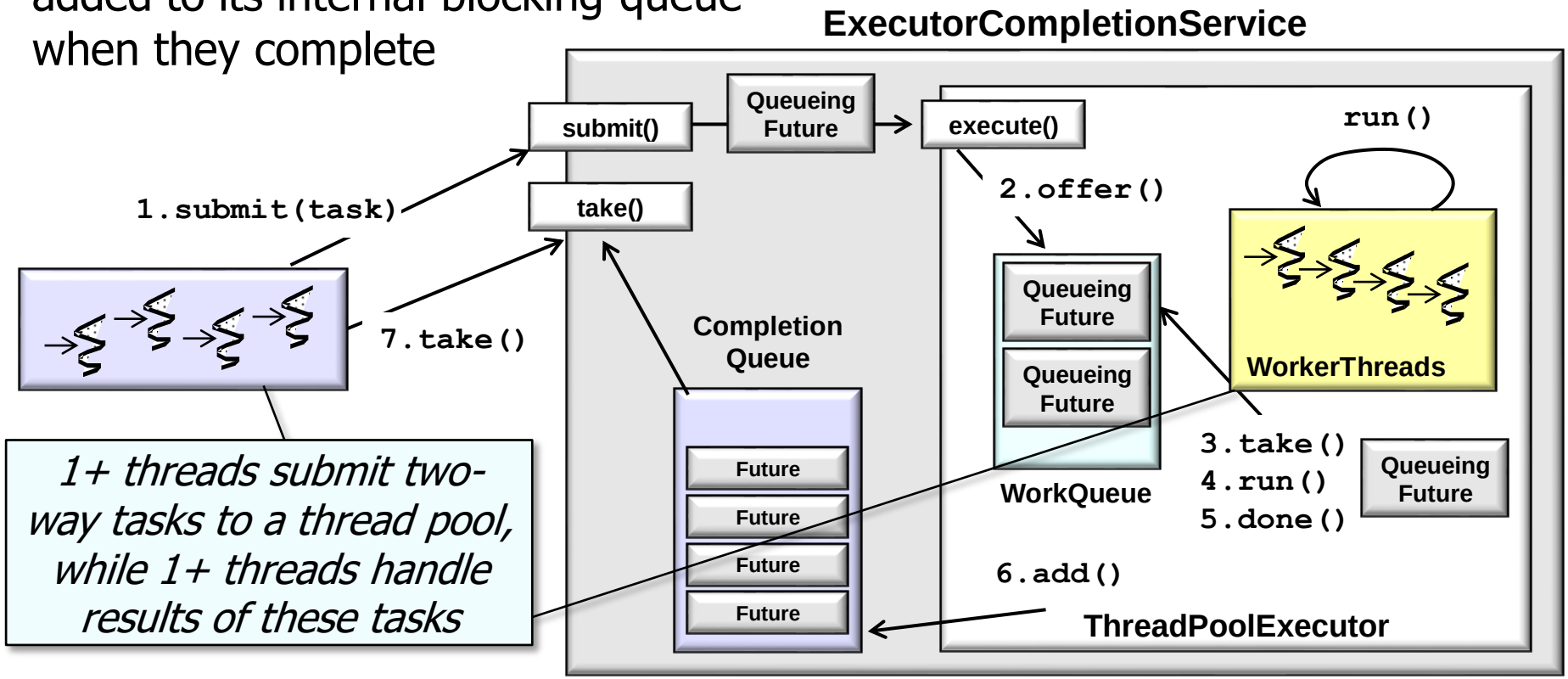
- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results
    - Block until a future for next completed task is available & then retrieve/remove it
    - Retrieve/remove a future for the next completed task or null if none are available
    - Block up to the specified wait time if future isn't available

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
...
 public Future<V> take() ... {
    ...
 }

 public Future<V> poll() {
    ...
 }

 public Future<V> poll(long
    timeout, TimeUnit unit) ... {
    ...
 } ...
```

Visualizing the Java ExecutorCompletionService

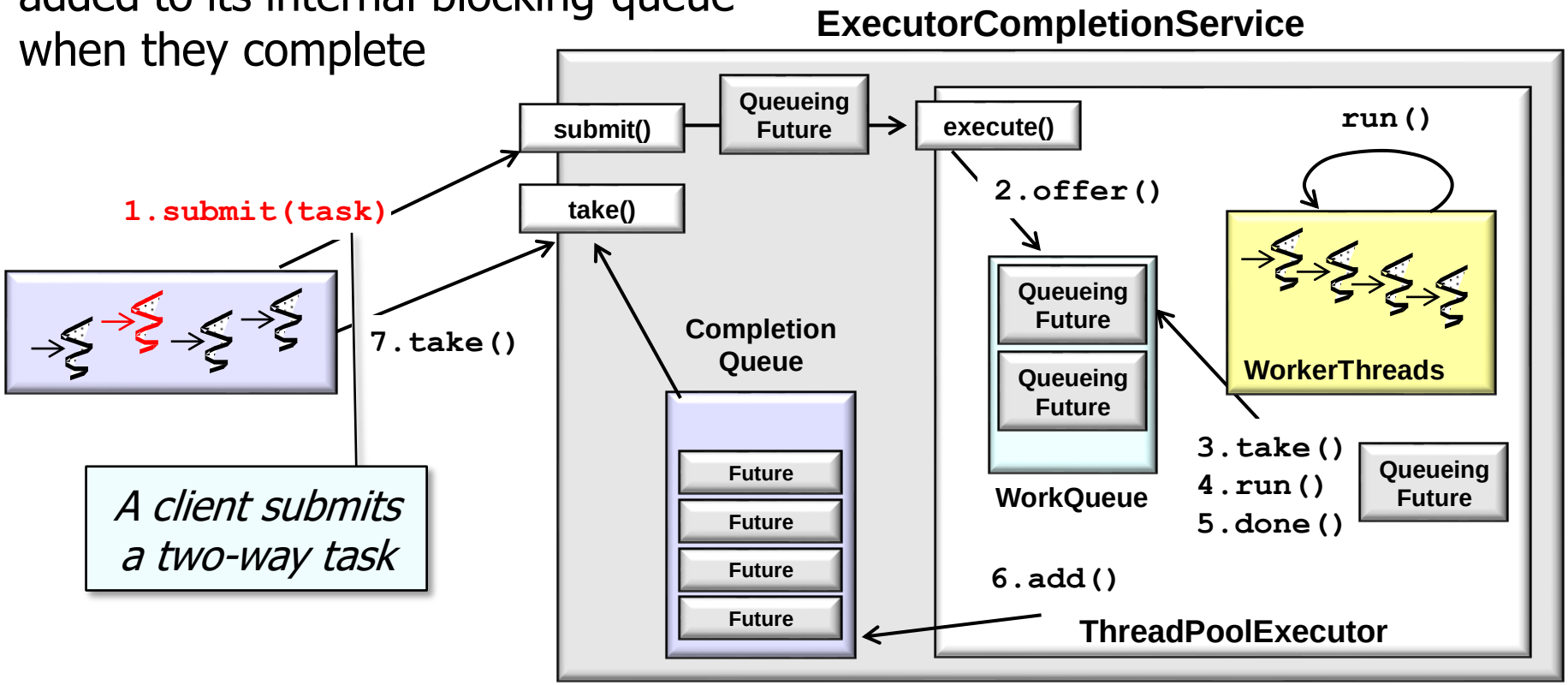
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete



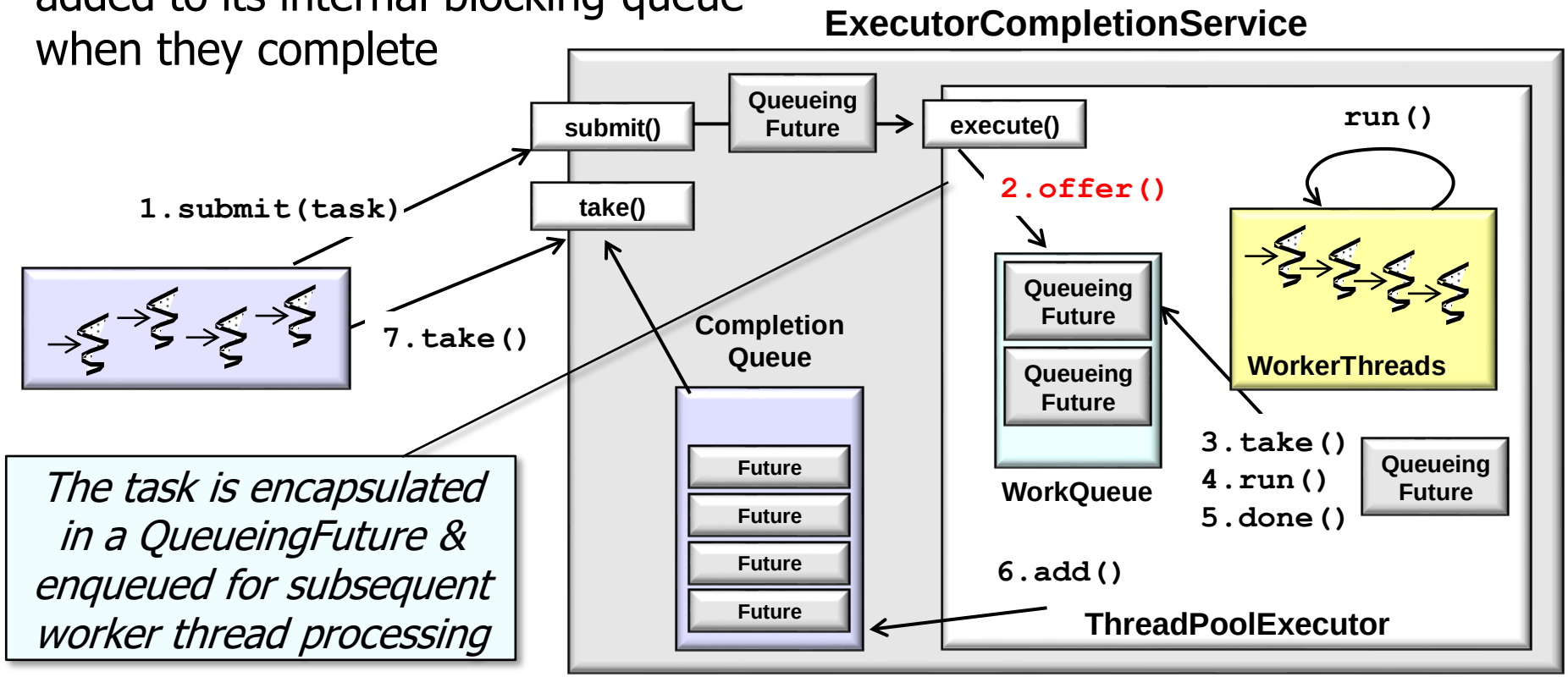
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete



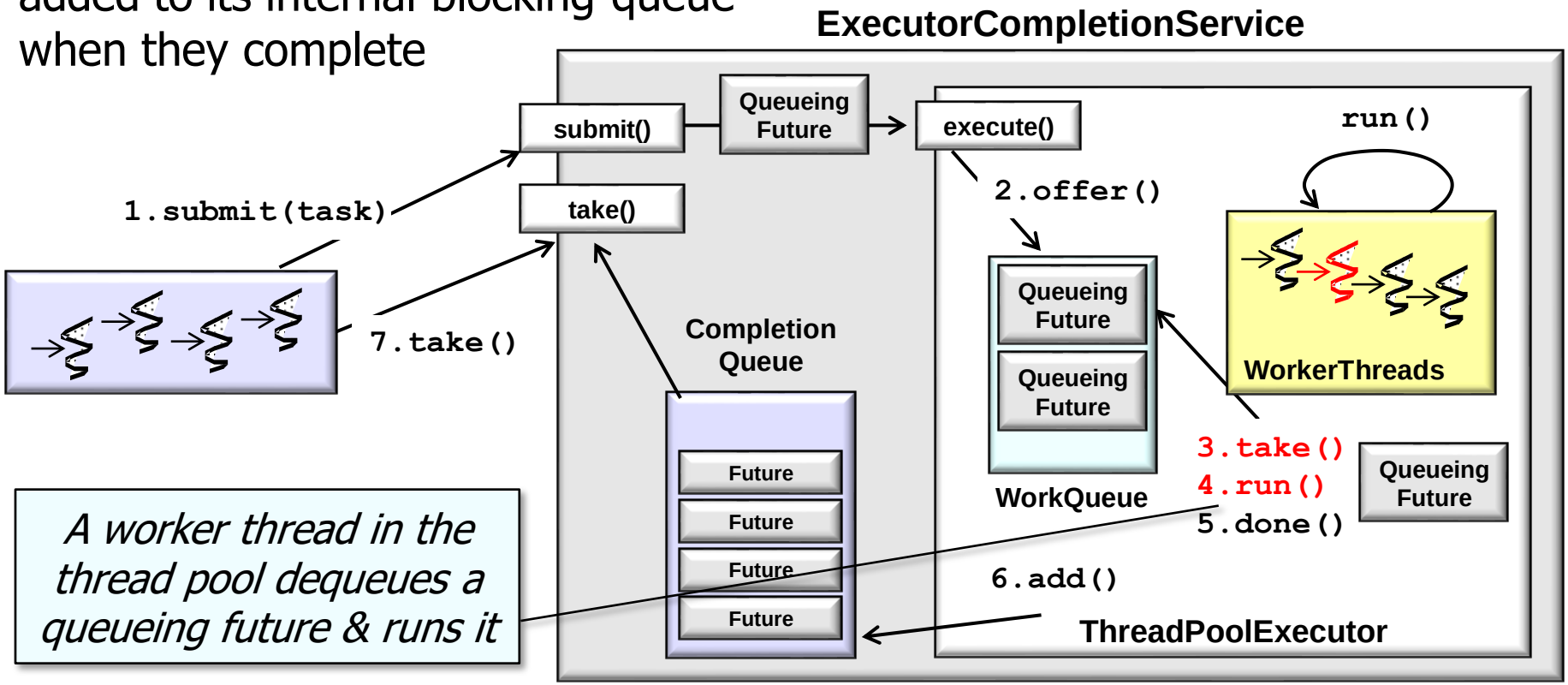
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete



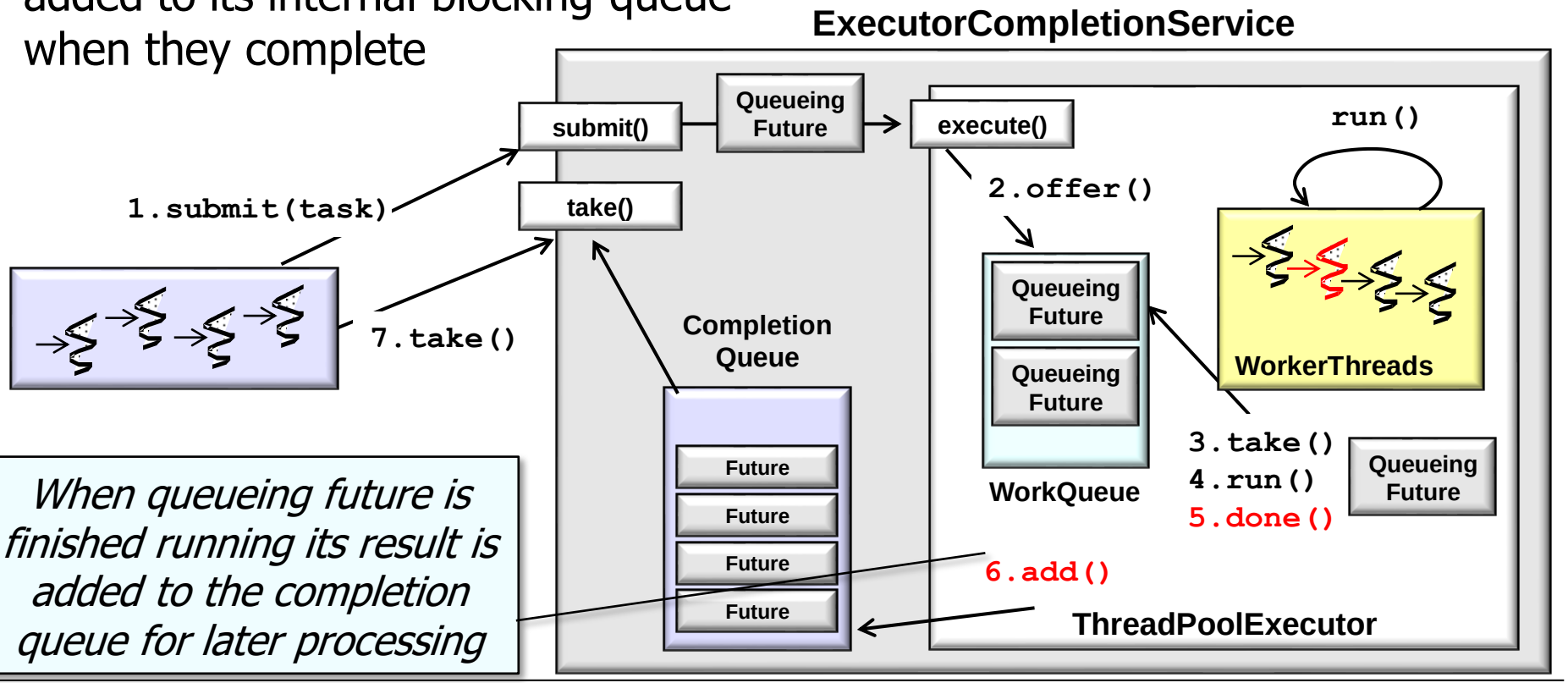
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete



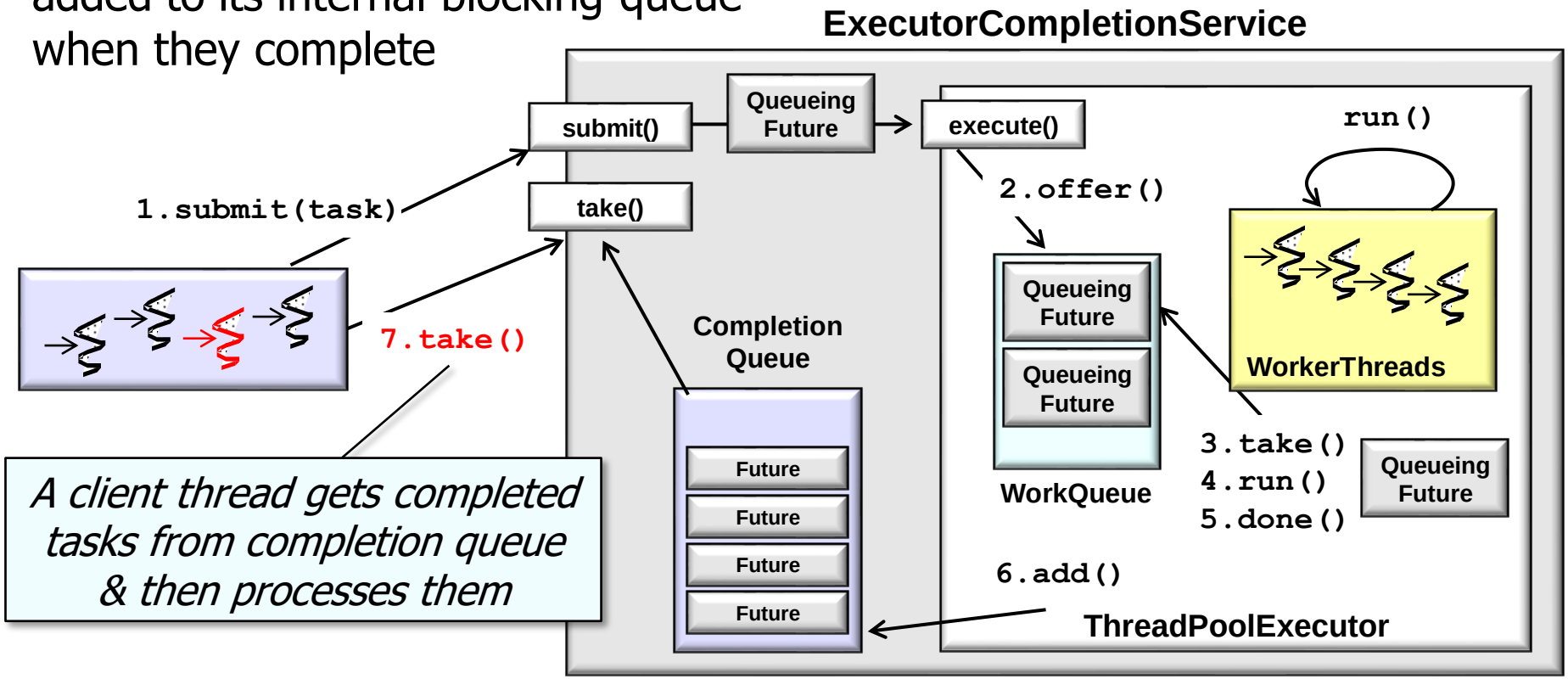
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete



Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete



End of Key Methods in the Java ExecutorCompletionService