KINGLAND®

Concurrency and Multi-Threading in Java

2021 April

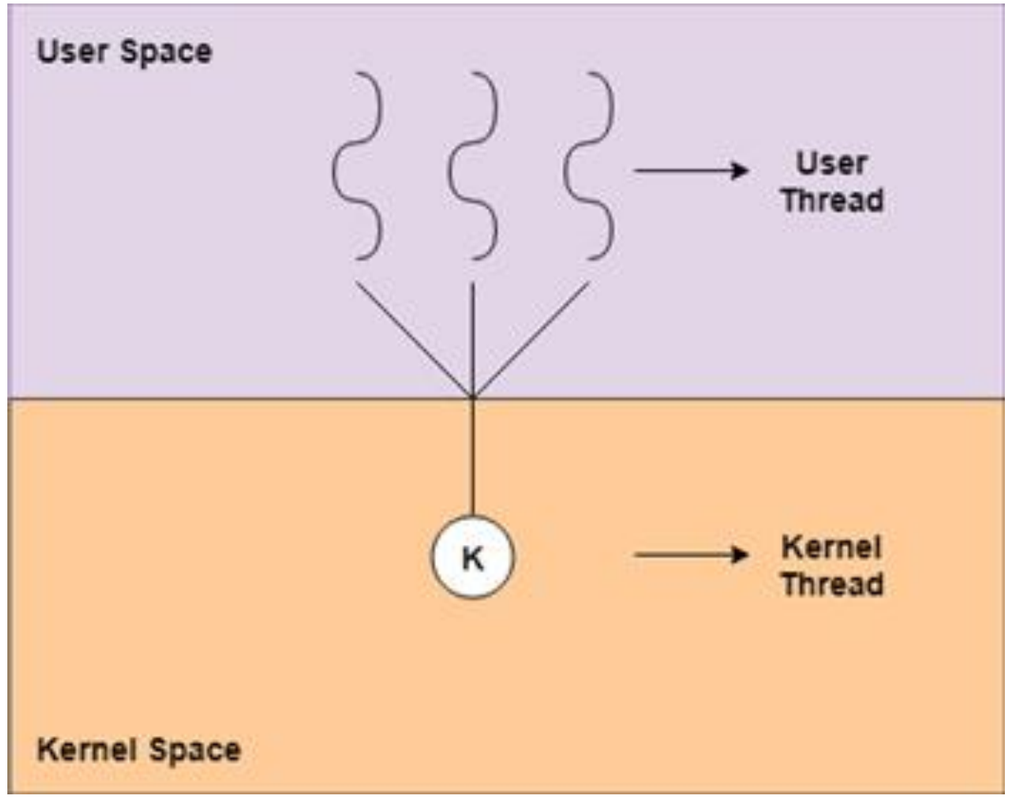
Agenda

- **What is Thread?**
- **Why Multi-Threading?**
- **Status of a Thread**
- **Threading in Java**
  - Methods
  - Implement a thread
  - FutureTask
  - ThreadPoolExecutor
- **synchronized**
- **volatile**

What is Thread?

- Defined at the OS level
- The actual unit of the execution of the OS process
- A set of instructions
- Can be executed “at the same time”
- Has various statuses

What is Thread?



KINGLAND

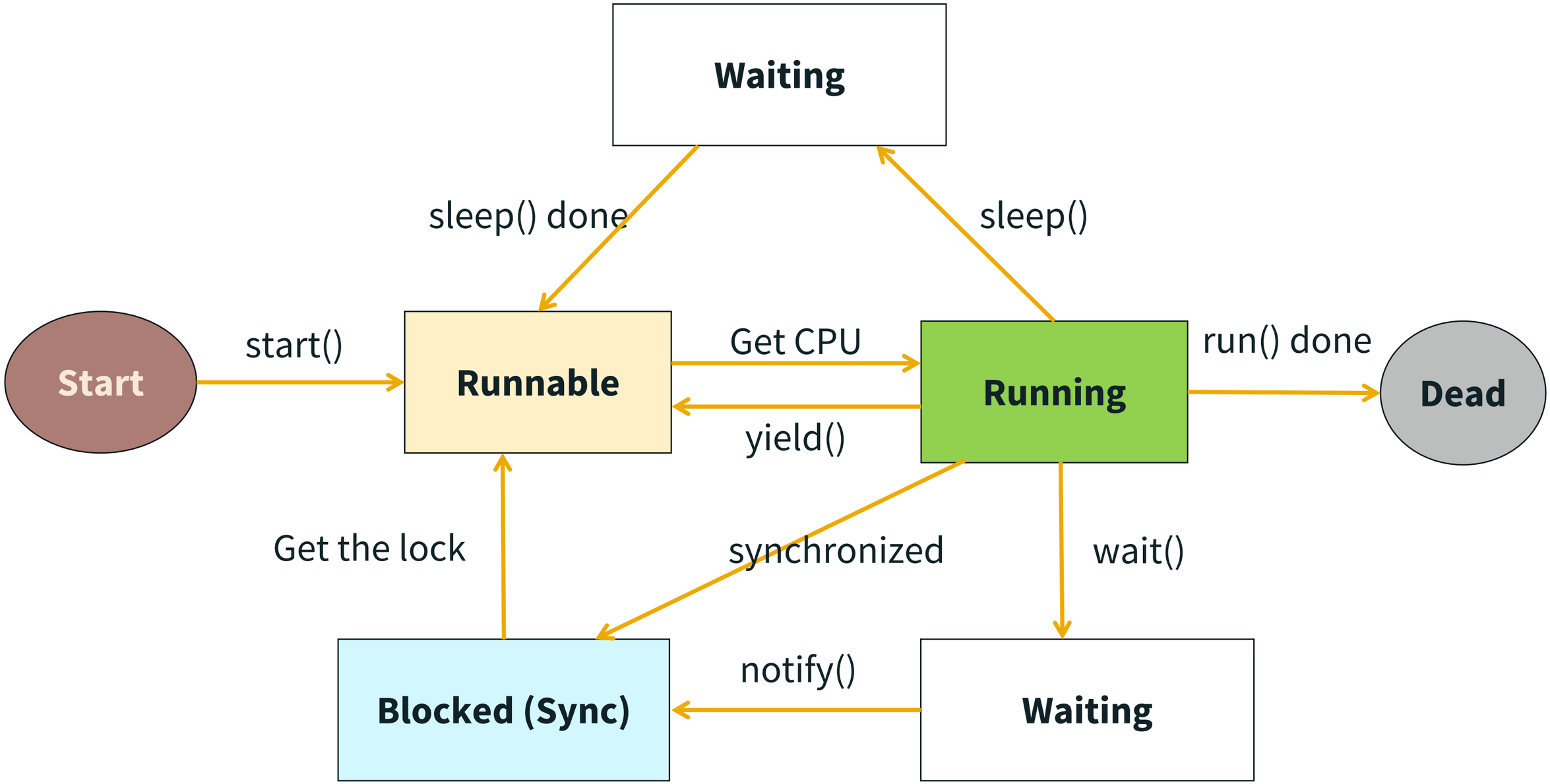
Why Multi-Threading?

Performance!

Status of a Thread

- New
- Runnable
- Running
- Blocked
- Dead

Status of a Thread



Threading in Java

Methods: Thread

- ***Thread.start():*** Move a thread to the runnable queue to race for the CPU time slice.
- ***Thread.yield():*** Releases the CPU time slice to the threads at same priority of the current thread. Does not release the lock.
- ***Thread.sleep(long millis):*** Releases the CPU time slice so that all threads have the opportunity. Supports the time setting. Do not release the lock.
- ***Thread.join():*** Forces the child thread to join the current parent thread. The current thread waits till the child thread completes.

Threading in Java

Methods: Object

- ***Object.wait():*** Releases the CPU time slide. Puts the current thread into the waiting pool. Releases the lock to the object which calls this method.
- ***Object.notify():*** Puts the current thread into the pool to race for the lock against the object calling this method.
- ***Object.notifyAll():*** Puts all the threads wanting the lock against the object calling this method into the pool for racing it.

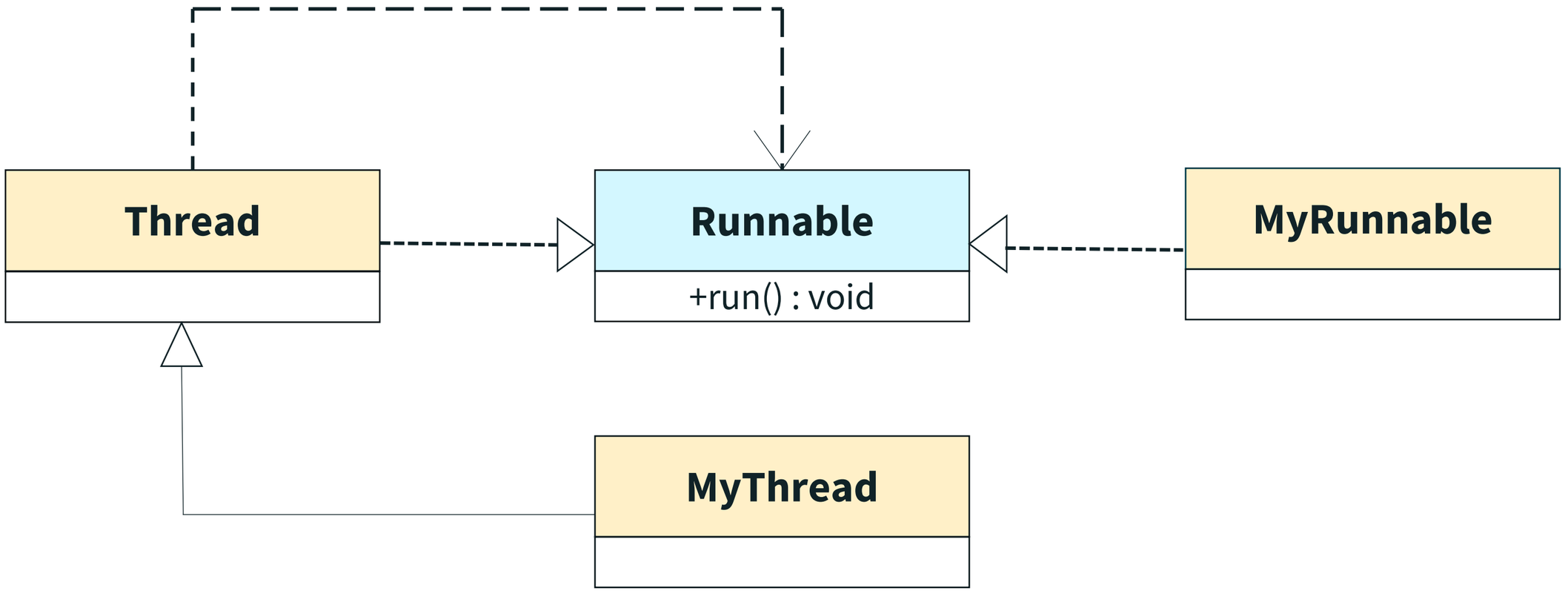
Threading in Java

Implement a Thread

- Extend *Thread* class
- Implement *Runnable* interface
- Callable<V> & Future

Threading in Java

Thread V.S. Runnable



Threading in Java

Thread V.S. Runnable

- *Thread* implements *Runnable*
- *Thread* mainly describes the threading functionalities
- *Runnable* is abstracted specifically focusing on the description of the resource
- *Runnable* implementation can be reused by different threads

See the example

Threading in Java

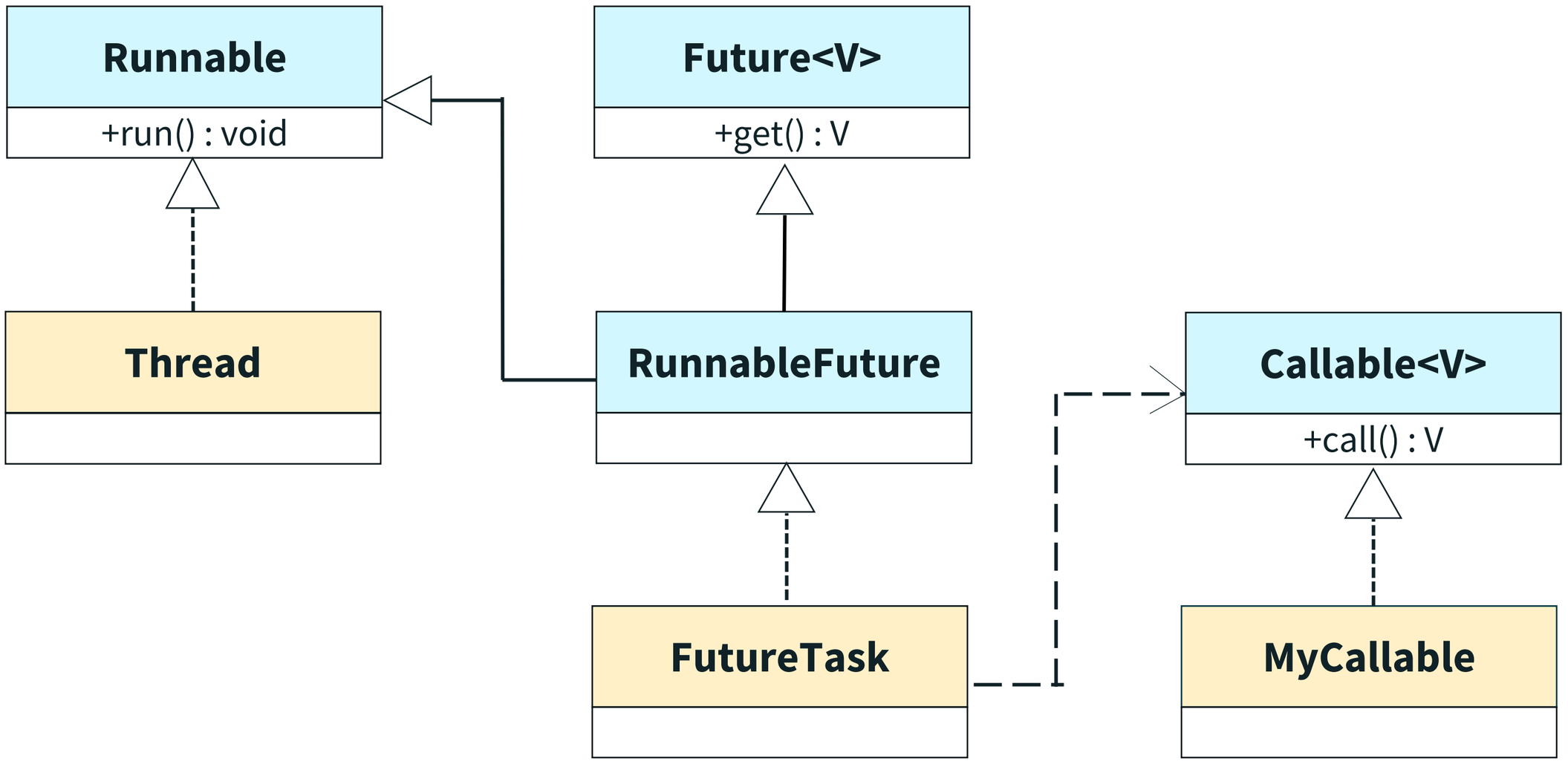
Runnable V.S. Callable

- They are both interfaces
- The *call()* method of *Callable throws exception*
- The *call()* method of *Callable* could return a result(works with FutureTask)

See the example

Threading in Java

Runnable V.S. Callable



KINGLAND

Threading in Java

FutureTask

- ***V get():*** Waits if necessary for the computation to complete, and then retrieves its result.
- ***boolean cancel(boolean mayInterruptIfRunning) :*** Attempts to cancel execution of this task.
- ***boolean isDone():*** Returns true if this task completed.
- ***boolean isCancelled():*** Returns true if this task was cancelled before it completed normally.

For more details:

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

Threading in Java

ThreadPoolExecutor

- Thread pools address two different problems: they usually provide improved performance when executing large numbers of asynchronous tasks, due to reduced per-task invocation overhead, and they provide a means of bounding and managing the resources, including threads, consumed when executing a collection of tasks.
- Each *ThreadPoolExecutor* also maintains some basic statistics, such as the number of completed tasks.

Threading in Java

ThreadPoolExecutor

- Implements *Executor* and *ExecutorService* (default implementation)
- Extends *AbstractExecutorService*
- 4 constructors:

ThreadPoolExecutor
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>)
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory)
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, RejectedExecutionHandler)
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory, RejectedExecutionHandler)

For more details:

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html

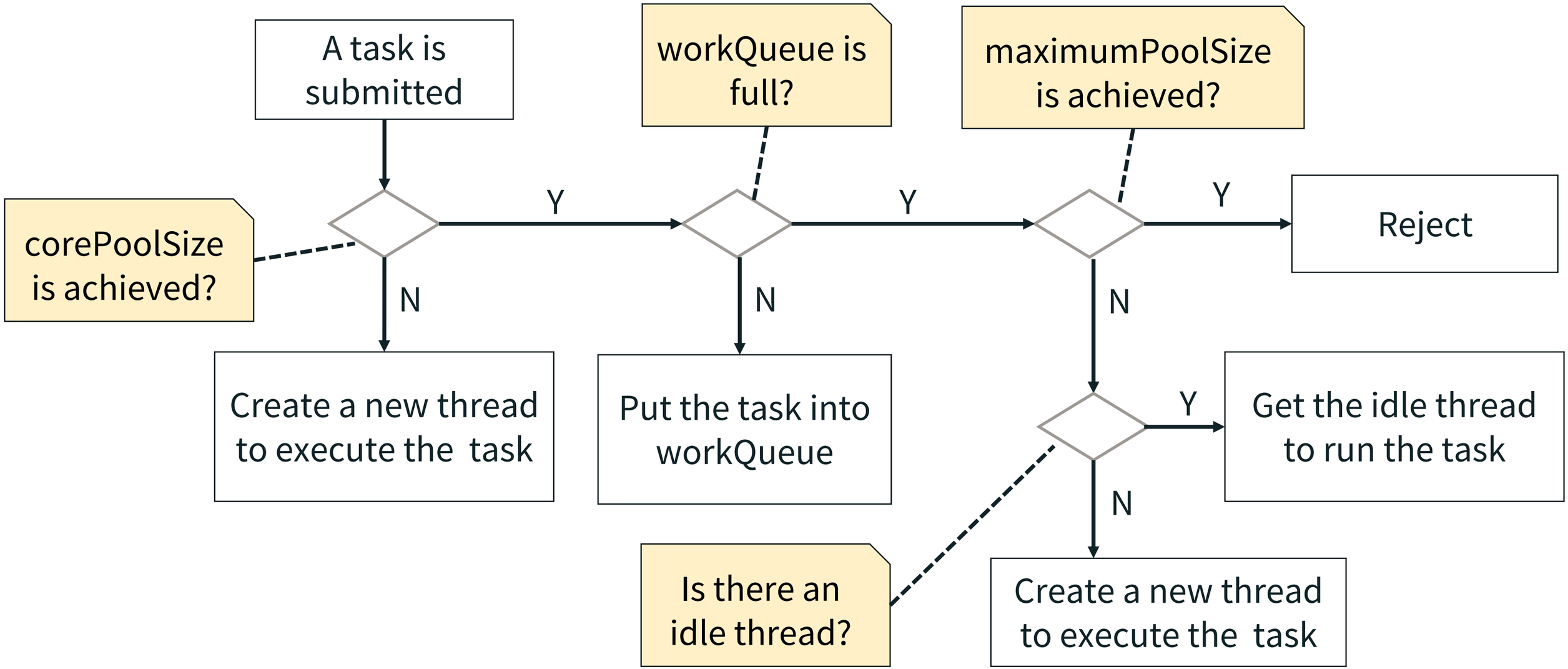
Threading in Java

ThreadPoolExecutor

1	corePoolSize	int
2	maximumPoolSize	int
3	keepAliveTime	long
4	unit	TimeUnit
5	workQueue	BlockingQueue<Runnable>
6	threadFactory	ThreadFactory
7	handler	RejectedExecutionHandler

Threading in Java

ThreadPoolExecutor



Threading in Java

ThreadPoolExecutor

3 strategies for queuing:

- **Direct handoffs**

  - hands off tasks to threads without holding
  - default choice for a work queue is a SynchronousQueue

- **Unbounded queues**

  - Never out-of-bound.
  - i.e. LinkedBlockingQueue

- **Bounded queues**

  - prevent resource exhaustion when used with finite maximumPoolSizes
  - i.e. ArrayBlockingQueue

Threading in Java

ThreadPoolExecutor

4 Default thread pool provided by ***Executor***:

- **FixedThreadPool**

  - core size = max size, keepAliveTime = 0

  - LinkedBlockingQueue

- **SingleThreadExecutor**

  - Wrappered by FinalizableDelegatedExecutorService

  - core size = max size = 1

  - LinkedBlockingQueue

Threading in Java

ThreadPoolExecutor

4 Default thread pool provided by ***Executor***:

- **CachedThreadPool**

  - corePoolSize = 0, maximumPoolSize = Integer.MAX_VALUE, keepAliveTime = 60

  - SynchronousQueue

- **ScheduledThreadPool**

  - Tasks are schedulable

  - core size is customizable, max size = Integer.MAX_VALUE

  - DelayedWorkQueue, unbounded

Threading in Java

synchronized

- Allows only one thread to entry, blocks the others
- Executable only when a thread gains the lock
- Can specify a block of codes, a method or a class

See the example

Threading in Java

volatile

- To specify a variable
- According to JMM, the threads usually copy the shared variables from the main memory to the work memory being used by the executors. But a volatile variable keeps being written back into the main memory so that no dirty read happens when reading it by any threads.

Kingland Systems. Discover Progress.

Our clients **know** that Kingland Systems delivers **faster, smarter,** more **reliable** solutions**.**

INDUSTRY SOLUTIONS

Kingland has been delivering Industry-specific solutions to leading global enterprises for more than 23 years.

SOLUTION PLATFORM

The Kingland Strategic Solution Platform means continuously smarter technology to deliver today and into the future.

EXPERT SERVICES

Kingland brings deep data and software expertise to every solution, helping you realize benefits swiftly — and with less risk.

KINGLAND

KINGLAND®

Thank You!

KINGLAND.COM