Java Concurrency

Allen

Agenda:

1:basic concepts of thread

2:basic synchronisation methods

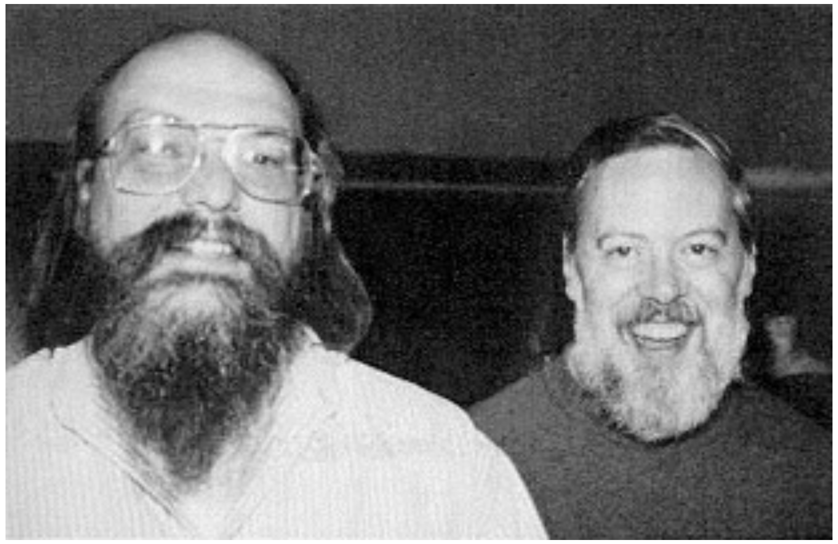
3:concurrency collections

4:thread pool framework

5:concurrency test

6:some classical problems

To me, process is a concept and thread is an implementation.
I would like to see the implementation get closer to the concept



Ken Thompson

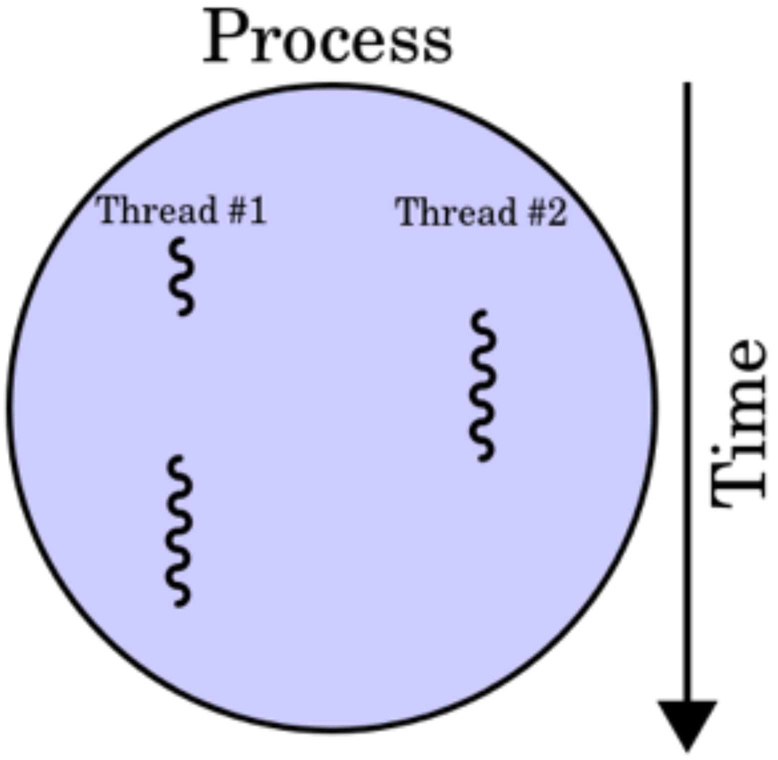
Unix system
The B programming language

basic concepts of thread

(1)the smallest sequence of programmed instructions

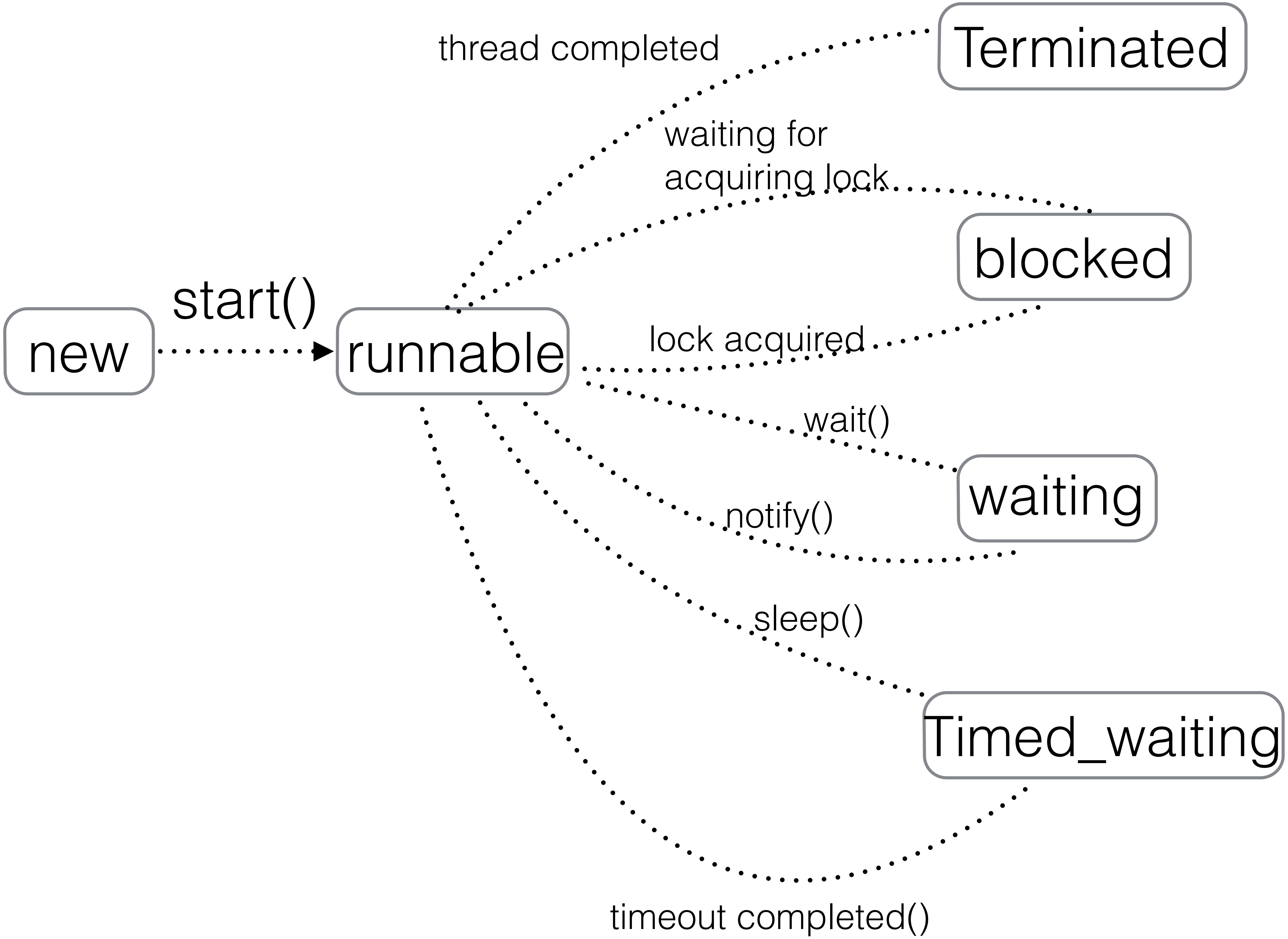
(2)sharing code,data and much lighter context switch

(3)advantages and disadvantages of multithreading



multithread comparision

advantages	disadvantages
Responsiveness *Faster execution* *Lower resource consumption* *(Apache Http Server)* *Better system utilization* *Simplified sharing and communication* *Parallelisation*	*synchronisation*



new

start()

runnable

thread completed

Terminated

waiting for
acquiring lock

blocked

lock acquired

wait()

waiting

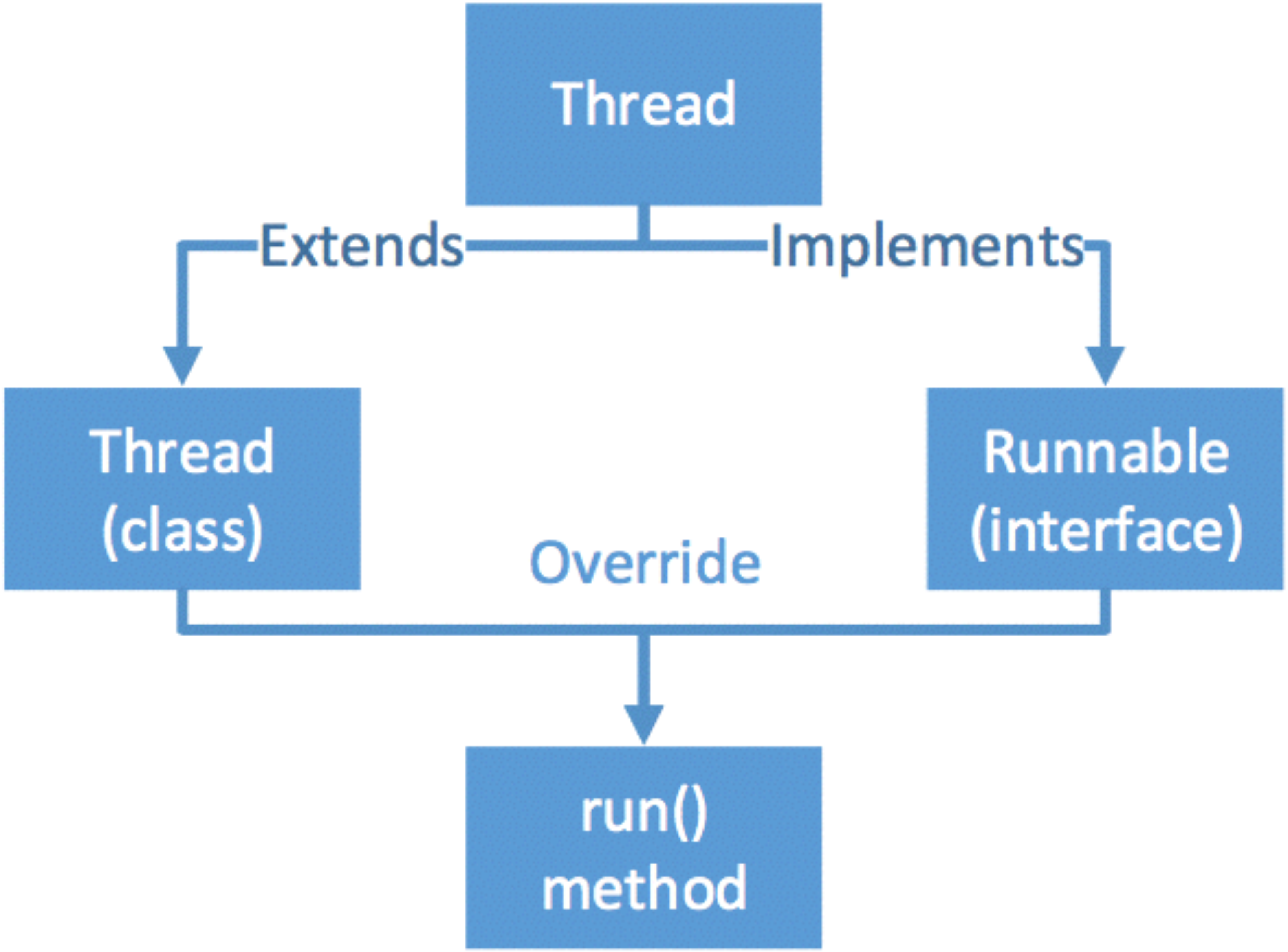
notify()

sleep()

Timed_waiting

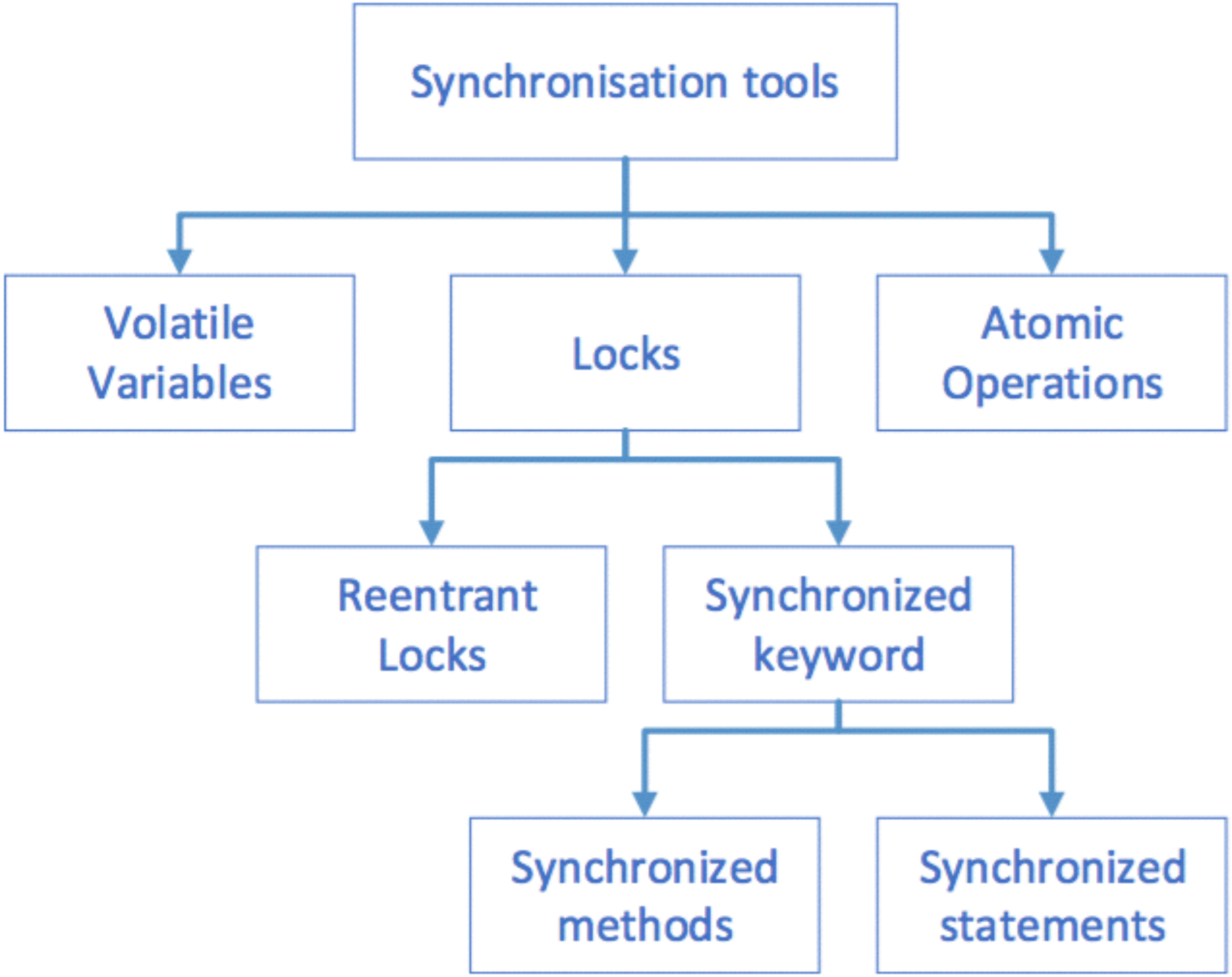
timeout completed()

Two mechanisms of creating threads



Which is better?

basic synchronisation methods



Synchronisation tools

Volatile Variables

Locks

Atomic Operations

Reentrant Locks

Synchronized keyword

Synchronized methods

Synchronized statements

(1):synchronized && volatile

- Atomicity
- Visibility

(2):Lock

- ReentrantLock
- ReentrantReadWriteLock

ReentrantLock vs synchronized

advantange	disadvantage
ability to handle interrupt	acquiring and releasing lock complicated code
a timeout on waiting for lock	
support fairness	
get List of all threads waiting for lock	

Synchronizer

(1):Semaphore
(2):CountDownLatch
(3):CyclicBarrier
(4):Exchanger
(5):Phaser(JDK 1.7)

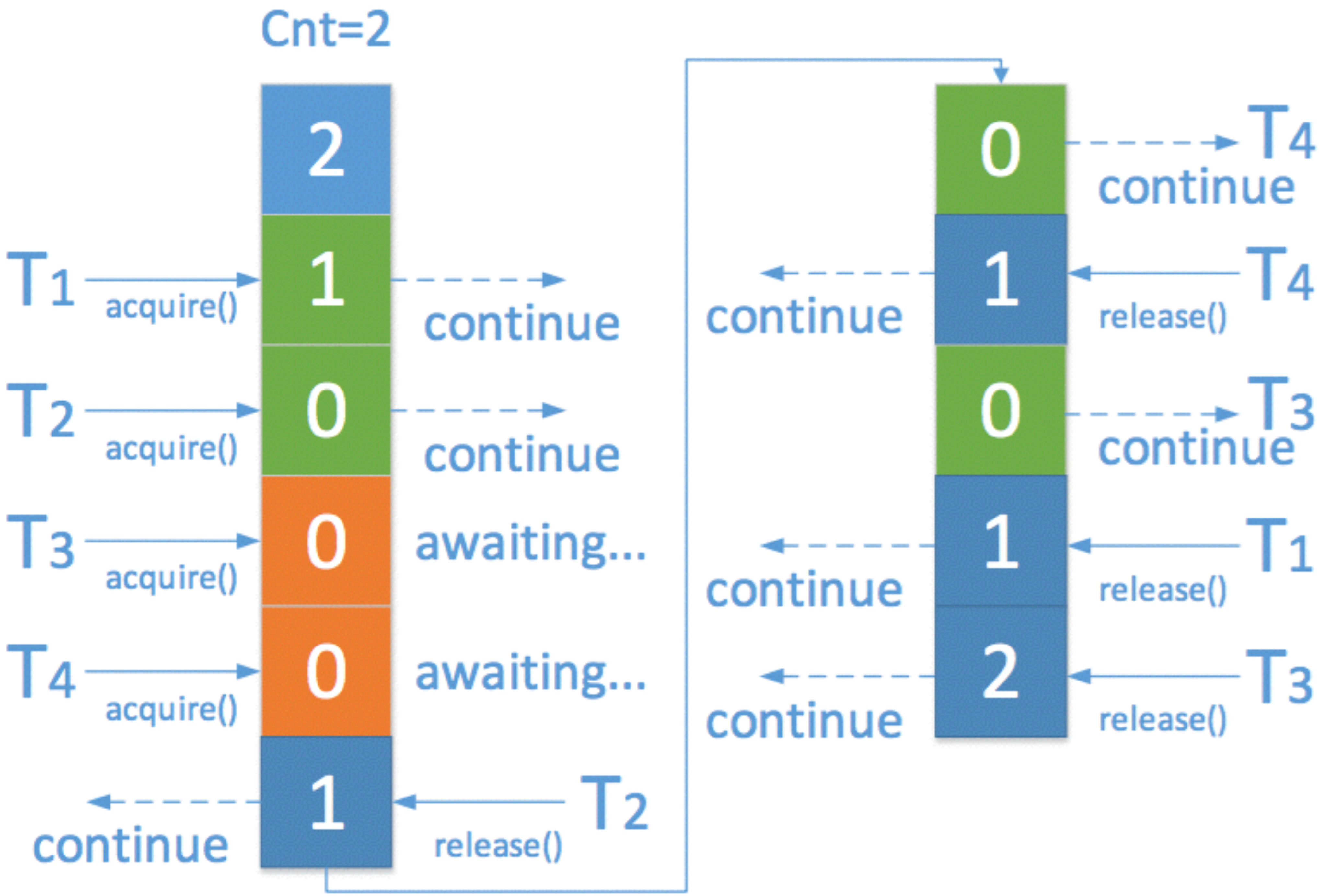
(1):Semaphore

Invented by the famous Dutch computer scientist Edsger Dijkstra in 1965

In Java, it is called counting semaphore, which maintains a set of permits (Semaphore value)

In Java, set the value to 1 and can be used as lock

How Semaphore works?

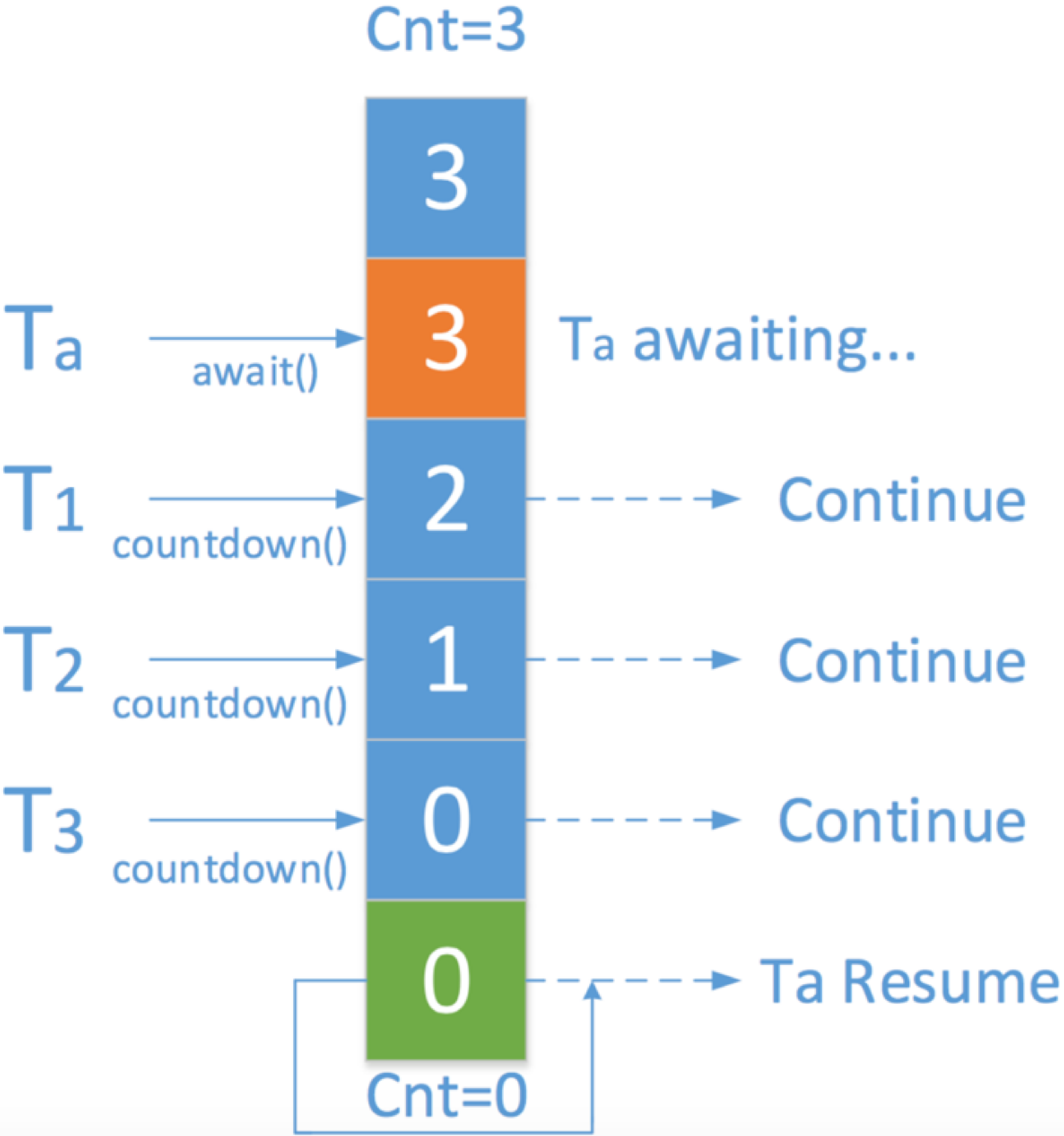


(2):CountDownLatch

A type of “switch” or “trigger” in concurrent programming

A thread or threads waits for the count value to reach zero before continuing to perform some process

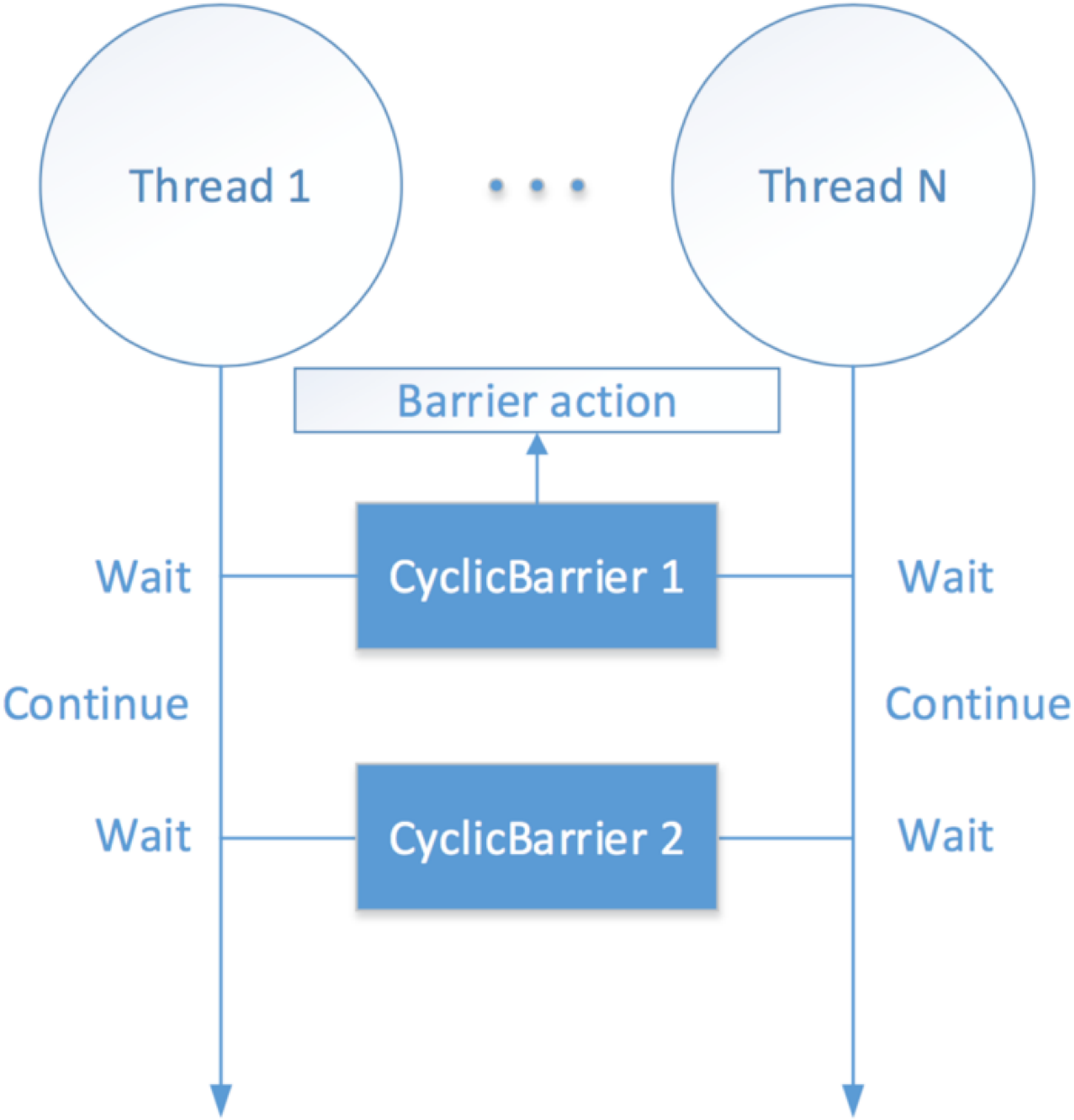
One-off process: Once the count value reaches 0, you cannot reset


Cnt=3
3
Ta
await()
3
Ta awaiting...
T1
countdown()
2
Continue
T2
countdown()
1
Continue
T3
countdown()
0
Continue
0
Ta Resume
Cnt=0

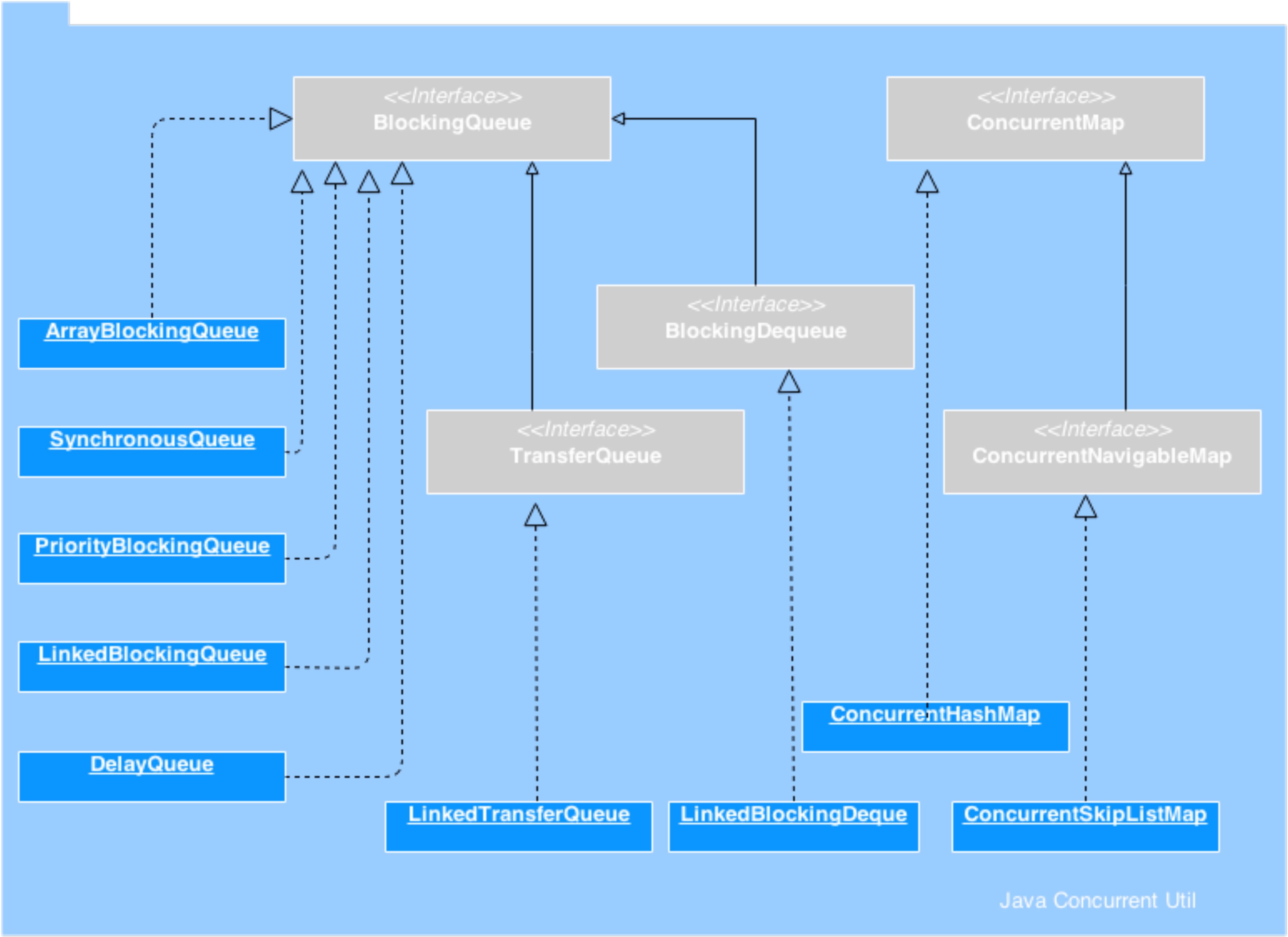
(3):CyclicBarrier

A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point

The barrier can be re-used after the waiting threads are released

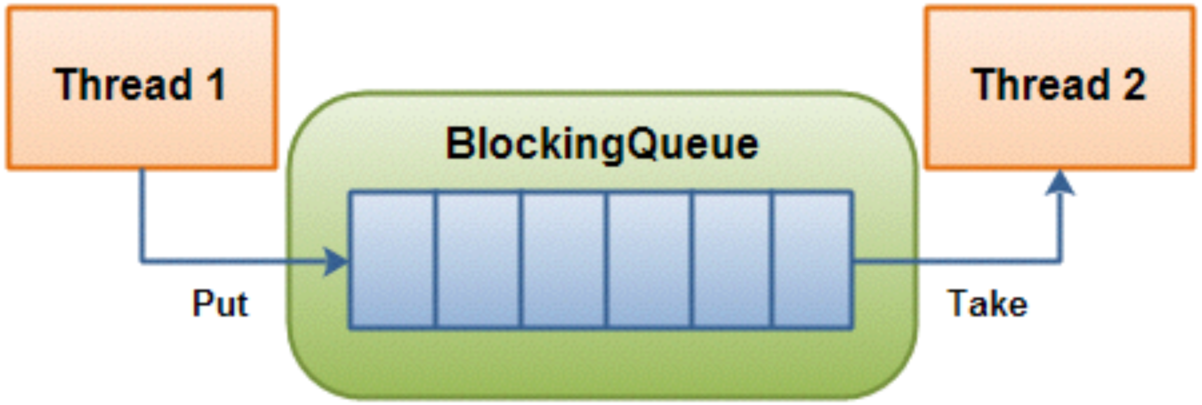

Thread 1
Thread N
Barrier action
Wait
CyclicBarrier 1
Wait
Continue
Continue
Wait
CyclicBarrier 2
Wait

concurrency collections



BlockingQueue

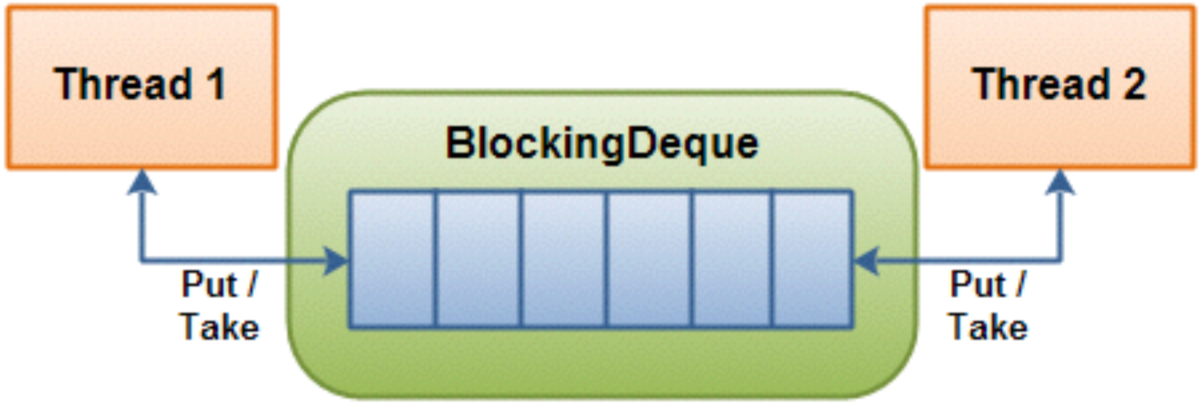
A queue that can be blocked when full or empty



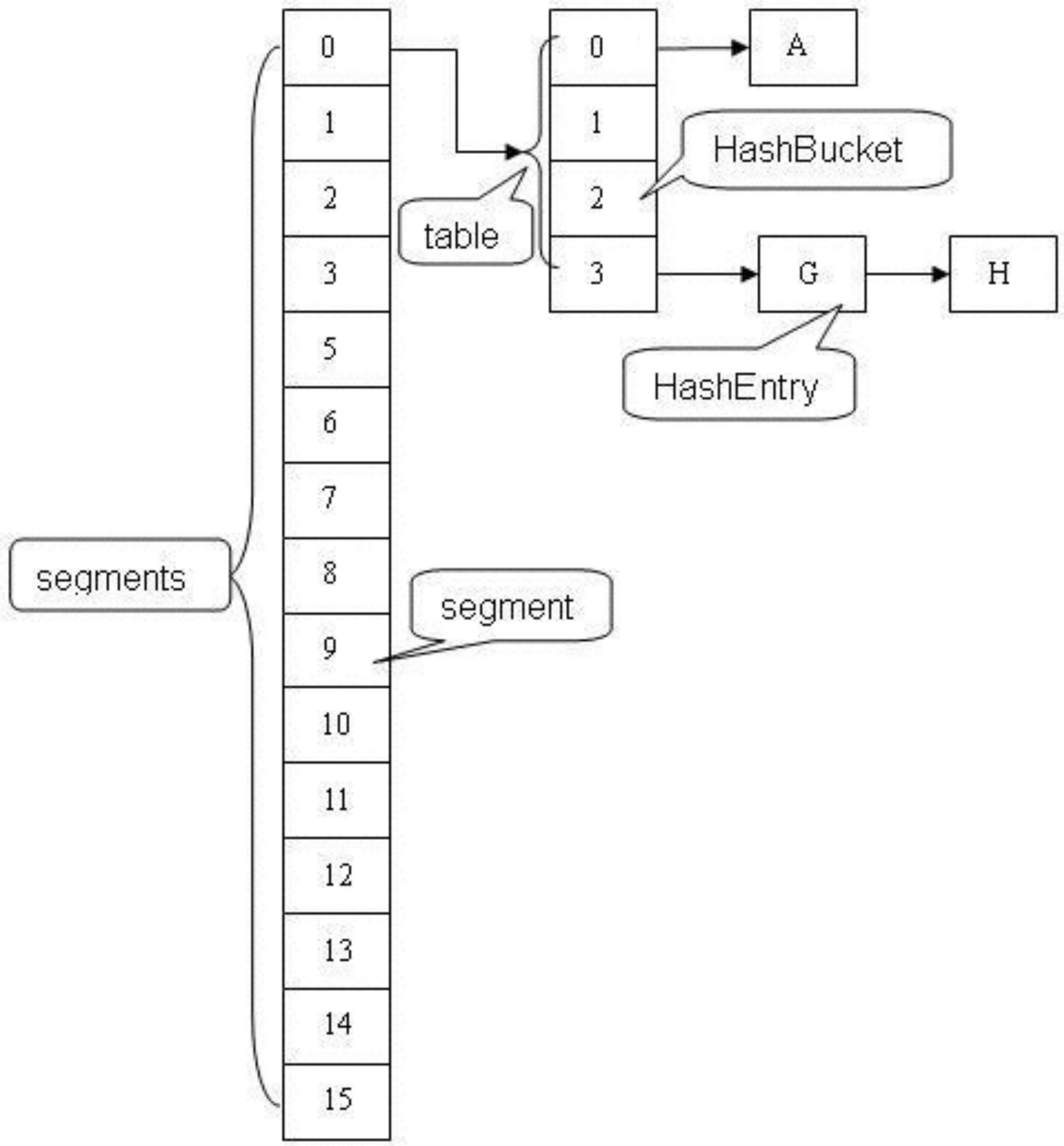
Queue Name	Usage
`ArrayBlockingQueue`	fixed bounded buffer&&elements FIFO
`LinkedBlockingQueue`	fixed bounded buffer&&elements FIFO
`PriorityBlockingQueue`	unbounded buffer&&with priority
`SynchronousQueue`	holding no data&&just channel
`DelayQueue`	used for Cache or close unused connections
`LinkedTransferQueue(JDK1.7)`	blockingqueue+waiting for consumer

BlockingDeque(JDK1.6)

Deque Name	usage
LinkedBlockingDeque	threads can put and take from both ends of the deque



ConcurrentHashMap



Segments
HashEntry

Atomic Variable

(1)Used to build lighter-weight high performance non-blocking synchronisation

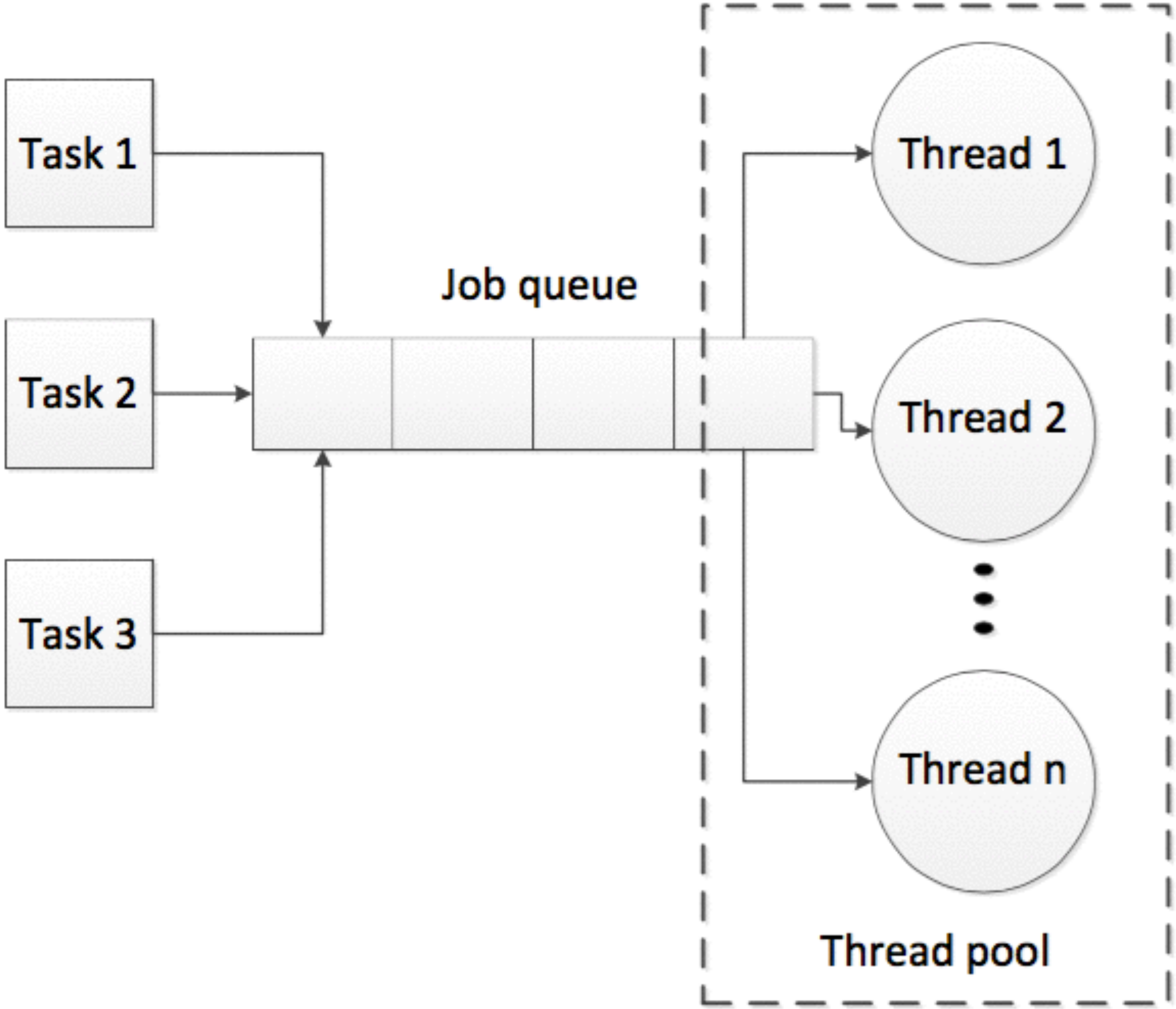
(2)Based on Compare-And-Swap operation

(3)Twelve atomic variable classes, two popular groups:

AtomicInteger, AtomicLong, AtomicBoolean,*********

AtomicIntegerFieldUpdater,AtomicLongFieldUpdater,*******

thread pool framework



benefits of Executor Framework:

(1)no need to write the code about the thread creation, ending and result get(Callable interface)

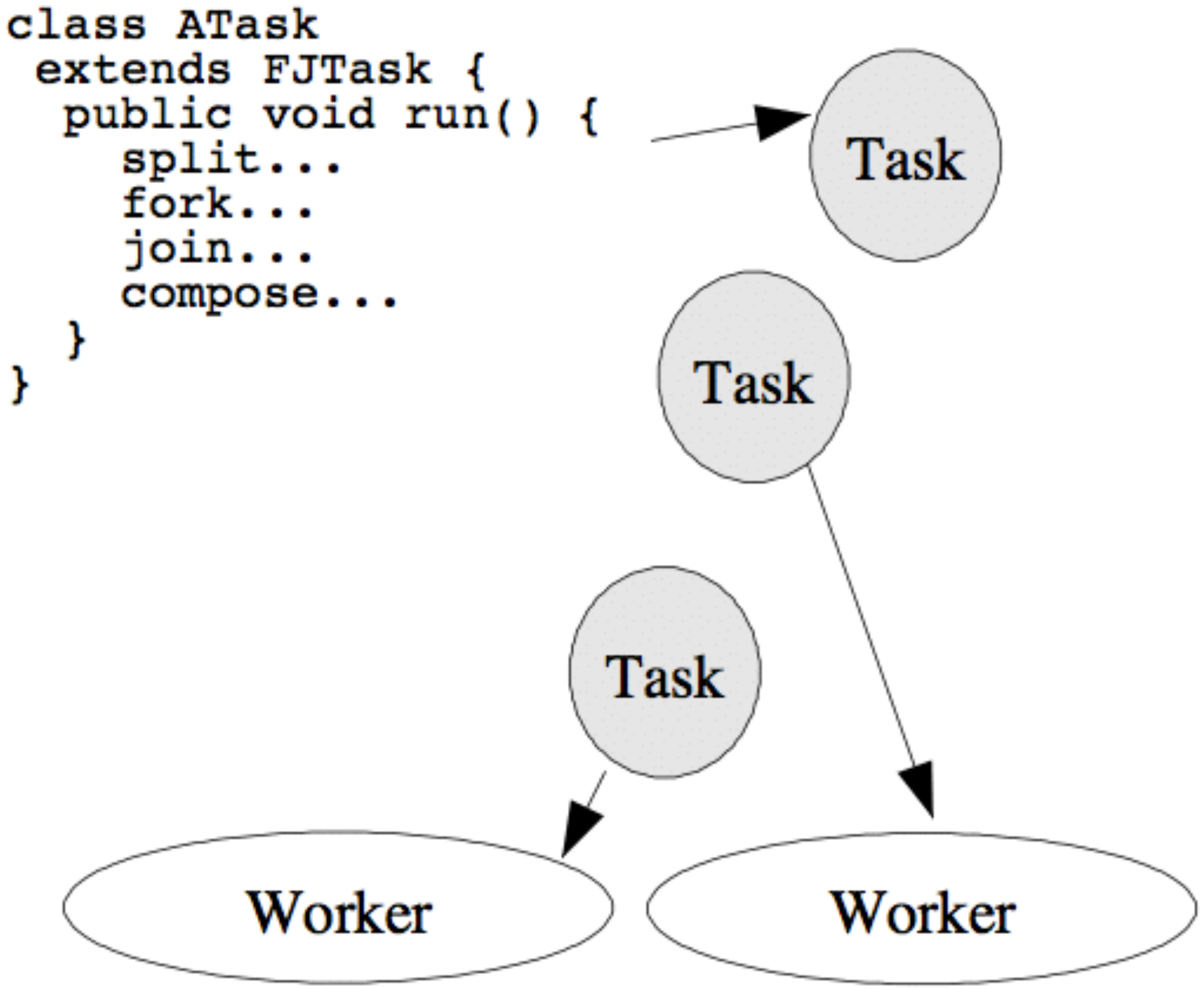
(2)no need to create the Thread Object manually

(3)have better management of the computer resources

some most used thread pools

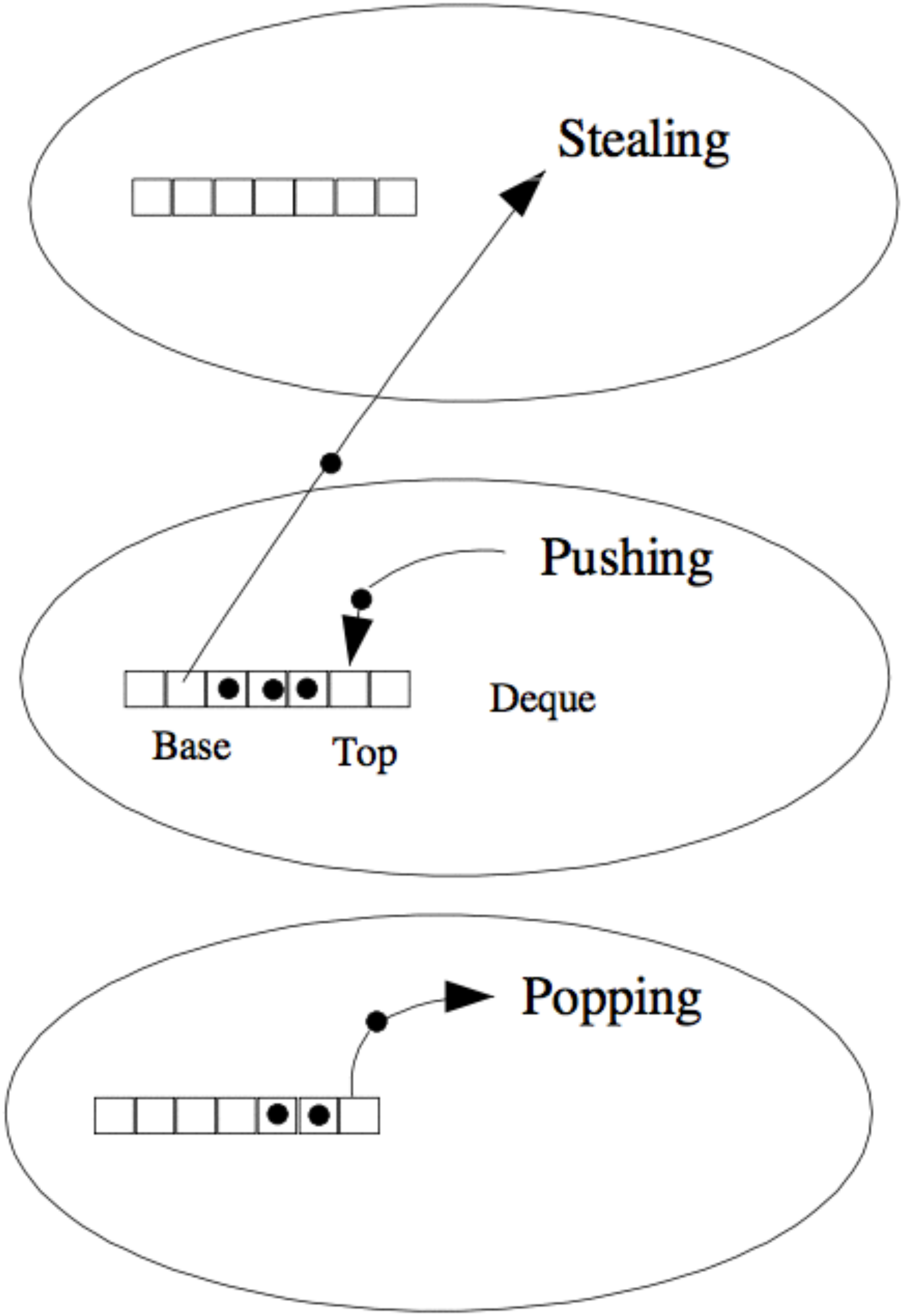
ThreadPool	usage
newFixedThreadPool	This executor is suitable for the web AppServer that deny the extra request to protect current user experience.
newSingleThreadExecutor	this executor is used only for one thread to start and can’t be reconfigurable
newCachedThreadPool	This executor is suitable for applications that launch many short-lived tasks.
newScheduledThreadPool	a fixed size thread pool that supports delayed and timed task execution.

Fork/Join Framework



```
class ATask
 extends FJTask {
  public void run() {
    split...
    fork...
    join...
    compose...
  }
}
```

Work-Stealing



concurrency test

1:test for correctness with JUnit

   (1)test bounded buffer

   (2)test the producer&&consumers

2:test for performance

   (1)concurrentHashMap&&Hashtable

Classical problems

(1)Producer&&Consumer
(2)Reader&&Writer
(3)Dining Philosophers Problem(Deadlock&&Solutions)

Thank you