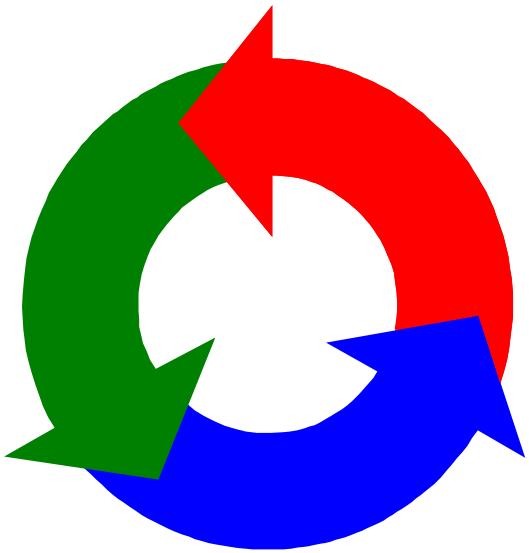
Chapter 2

Processes & Threads

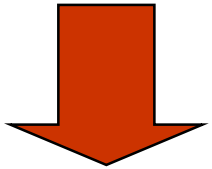


concurrent processes

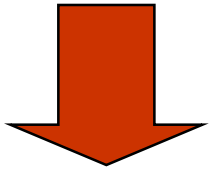
We structure complex systems as sets of simpler activities, each represented as a **sequential process**. Processes can overlap or be concurrent, so as to reflect the concurrency inherent in the physical world, or to offload time-consuming tasks, or to manage communications or other devices.

Designing concurrent software can be complex and error prone. A rigorous engineering approach is essential.

Concept of a process as a sequence of actions.



Model processes as finite state machines.



Program processes as threads in Java.

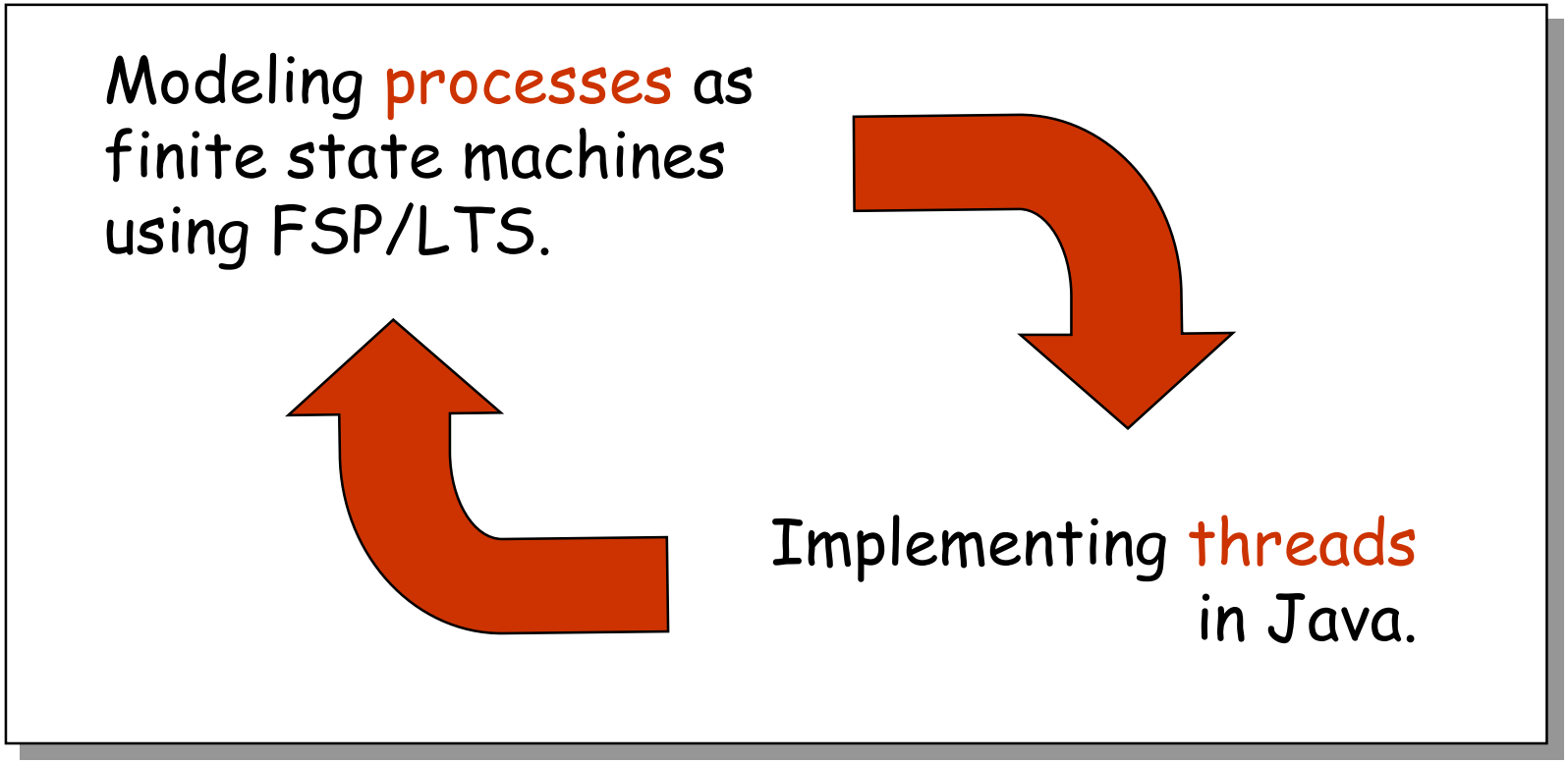
processes and threads

Concepts: processes - units of sequential execution.

Models: **finite state processes (FSP)**
to model processes as sequences of actions.
labelled transition systems (LTS)
to analyse, display and animate behavior.

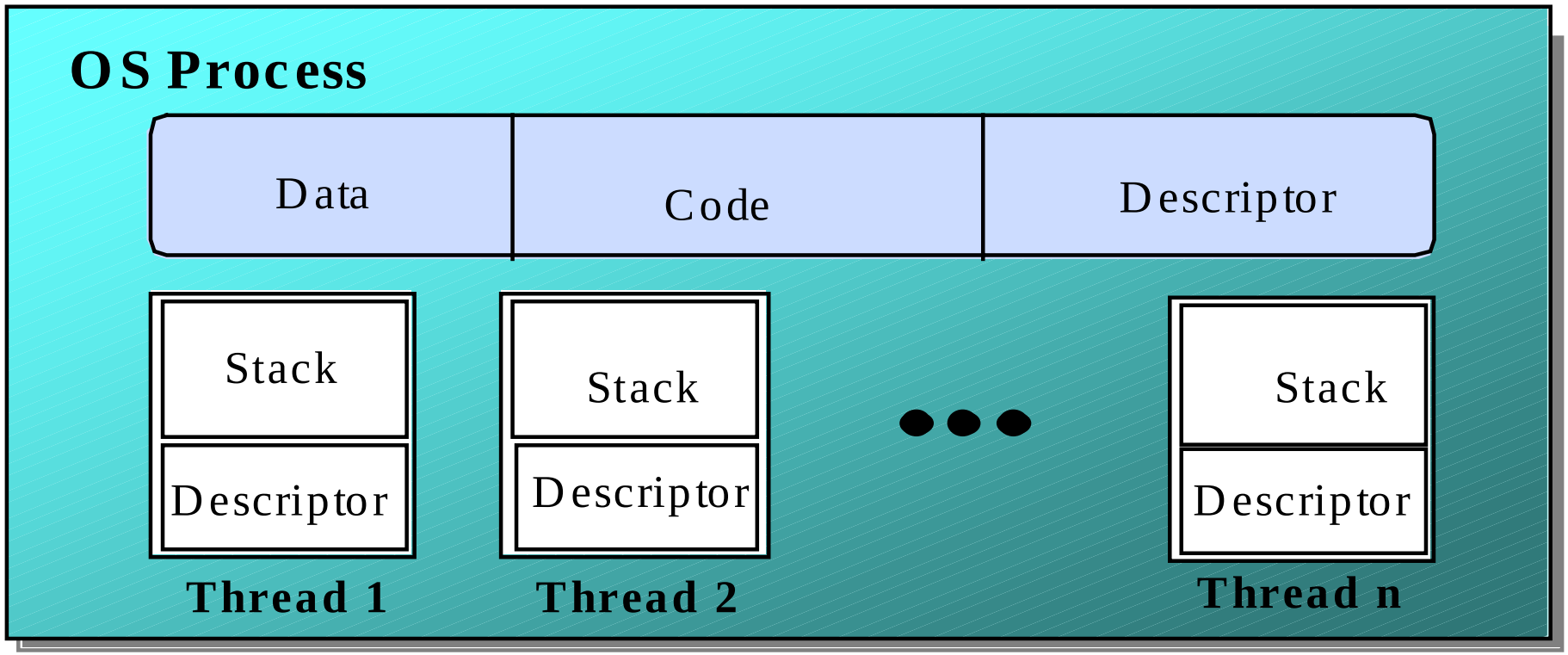
Practice: Java threads

2.2 Implementing processes



Note: to avoid confusion, we use the term ***process*** when referring to the models, and ***thread*** when referring to the implementation in Java.

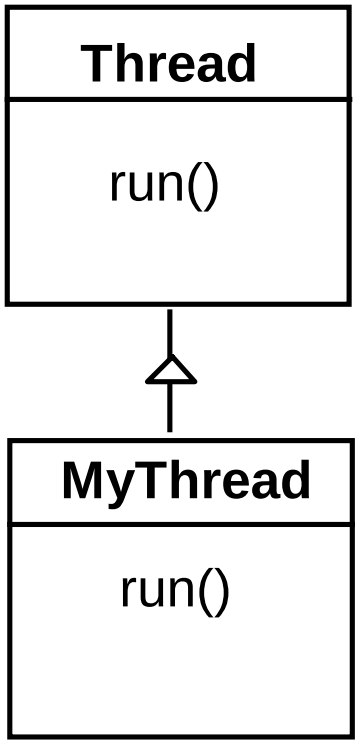
Implementing processes - the OS view



A (heavyweight) process in an operating system is represented by its code, data and the state of the machine registers, given in a descriptor. In order to support multiple (lightweight) **threads of control**, it has multiple stacks, one for each thread.

threads in Java

A Thread class manages a single sequential thread of control. Threads may be created and deleted dynamically.

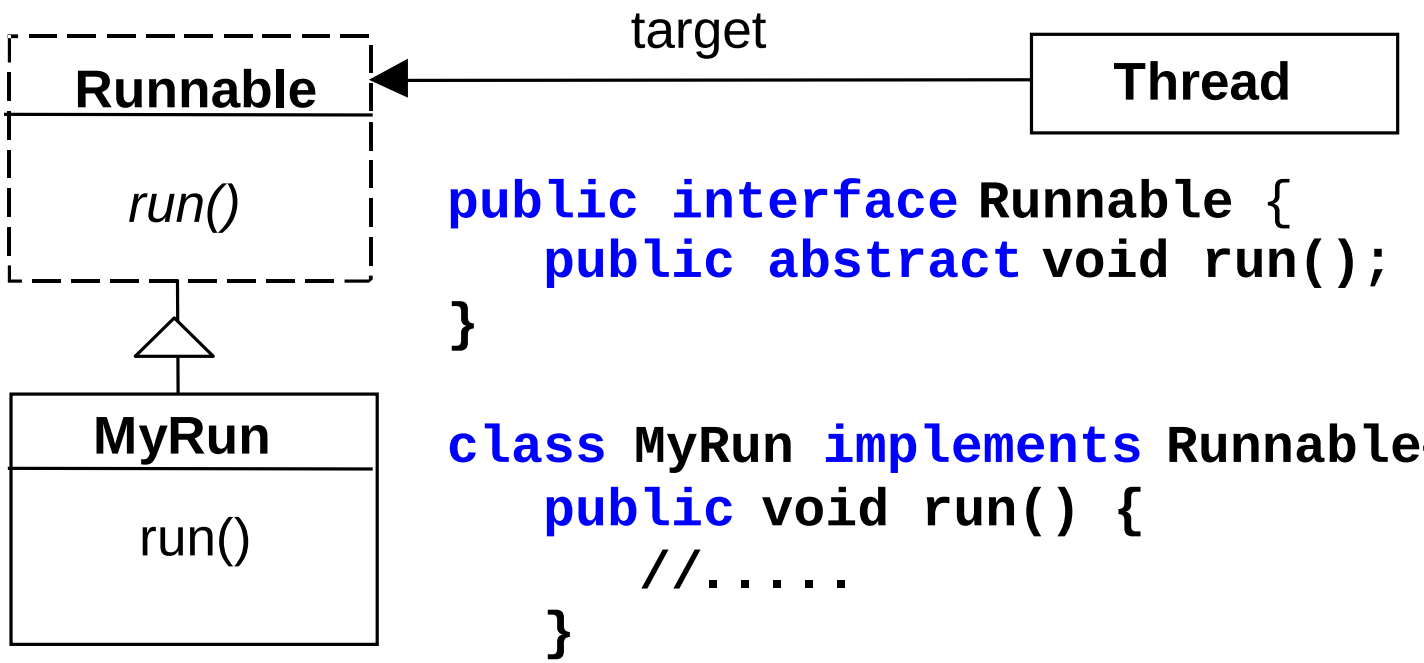


The Thread class executes instructions from its method run(). The actual code executed depends on the implementation provided for run() in a derived class.

```
class MyThread extends Thread {
      public void run() {
            //......
      }
}
```

threads in Java

Since Java does not permit multiple inheritance, we often implement the **run()** method in a class not derived from Thread but from the interface Runnable.

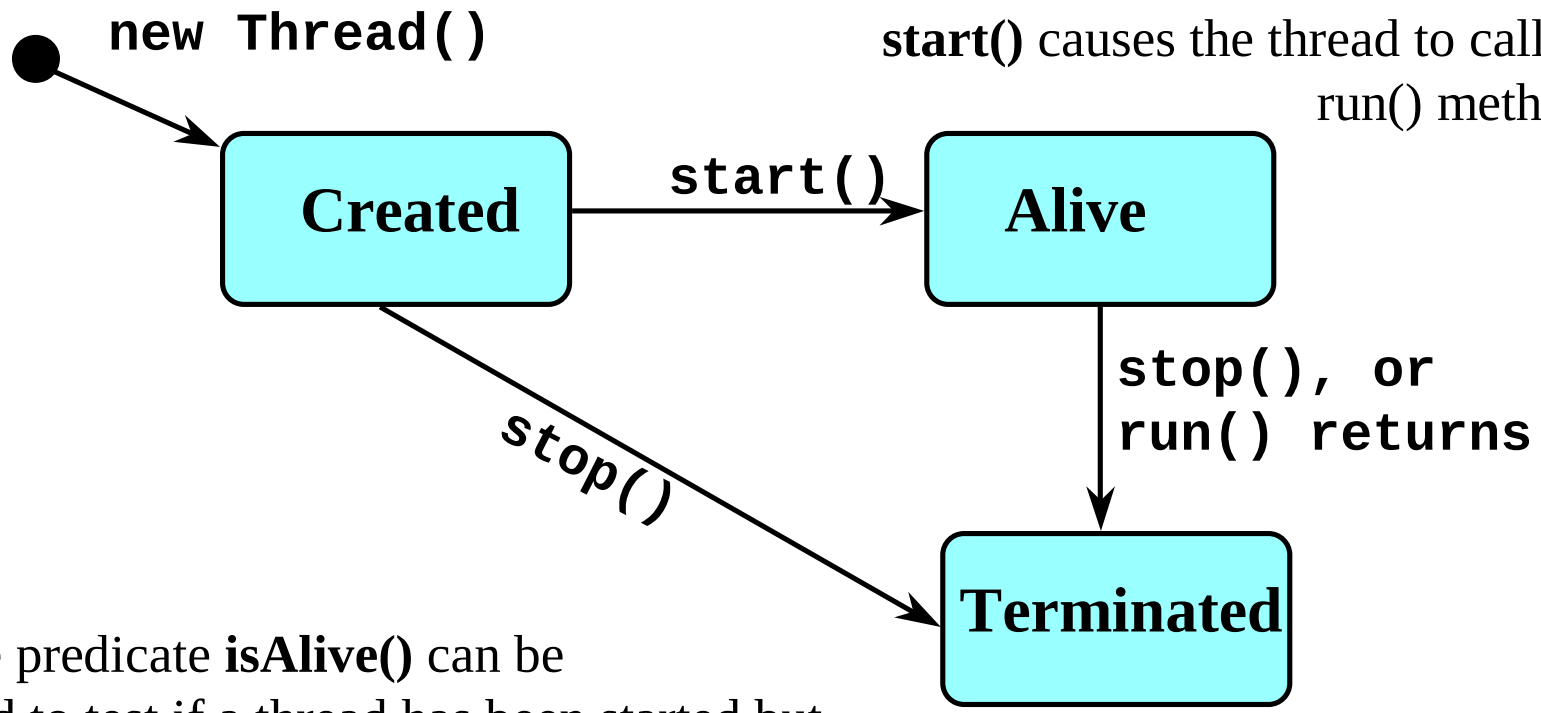



```
public interface Runnable {
    public abstract void run();
}

class MyRun implements Runnable{
    public void run() {
        //.....
    }
}
```

thread life-cycle in Java

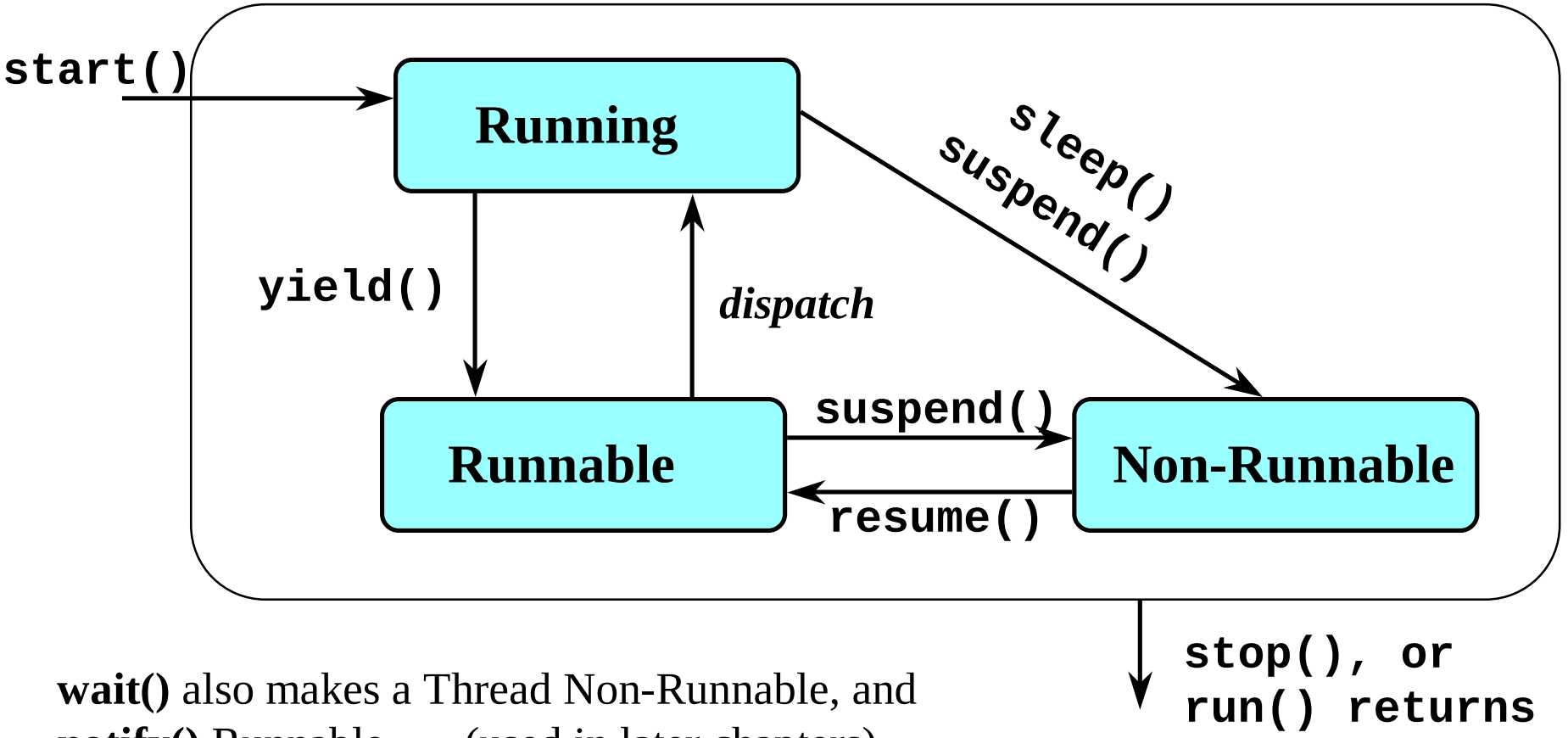
An overview of the life-cycle of a thread as state transitions:



The predicate **isAlive()** can be used to test if a thread has been started but not terminated. Once terminated, it cannot be restarted (cf. mortals).

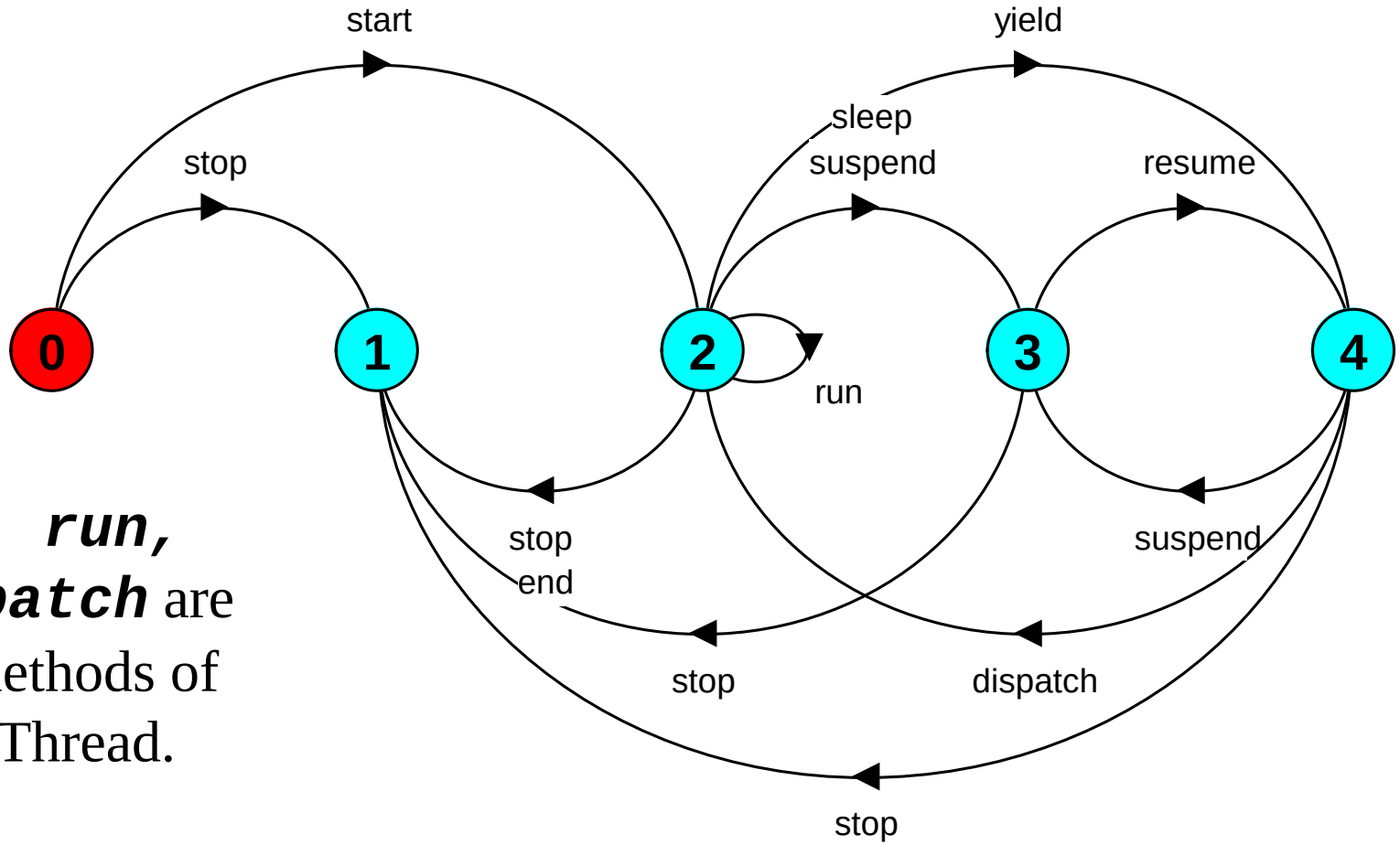
thread alive states in Java

Once started, an **alive** thread has a number of substates :



wait() also makes a Thread Non-Runnable, and
notify() Runnable (used in later chapters).

Java thread lifecycle - an FSP specification



end, run, dispatch are not methods of class Thread.

States 0 to 4 correspond to **`CREATED, TERMINATED, RUNNING, NON-RUNNABLE,`** and **`RUNNABLE`** respectively.

Summary

- Concepts
  - process - unit of concurrency, execution of a program
- Models
  - LTS to model processes as state machines - sequences of atomic actions
- Practice
  - Java threads to implement processes.
  - Thread lifecycle - created, running, runnable, non-runnable, terminated.