Concurrency

What is concurrency?

Processes vs. Threads

What is a sequential program?

A single thread of control that executes one instruction and when it is finished execute the next logical instruction

What is a concurrent program?

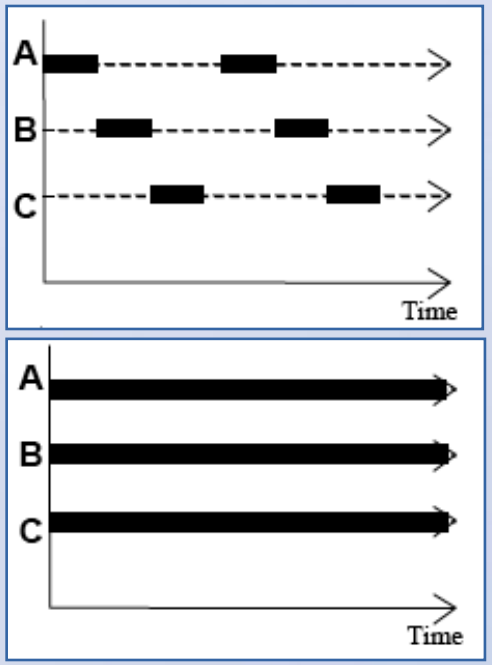
A collection of autonomous sequential threads, executing (logically) in parallel.

A multi-threaded program is one that can have multiple flows of control when executed.

At some time instance, there may exist multiple instructions or execution points) that are being executed in the program

Concurrency vs. Parallelism

- Interleaved Concurrency
  - Logically simultaneous processing
  - Interleaved execution on a single processor
- Parallelism
  - Physically simultaneous processing
  - Requires a multiprocessor or a multicore system



Why use Concurrent Programming?

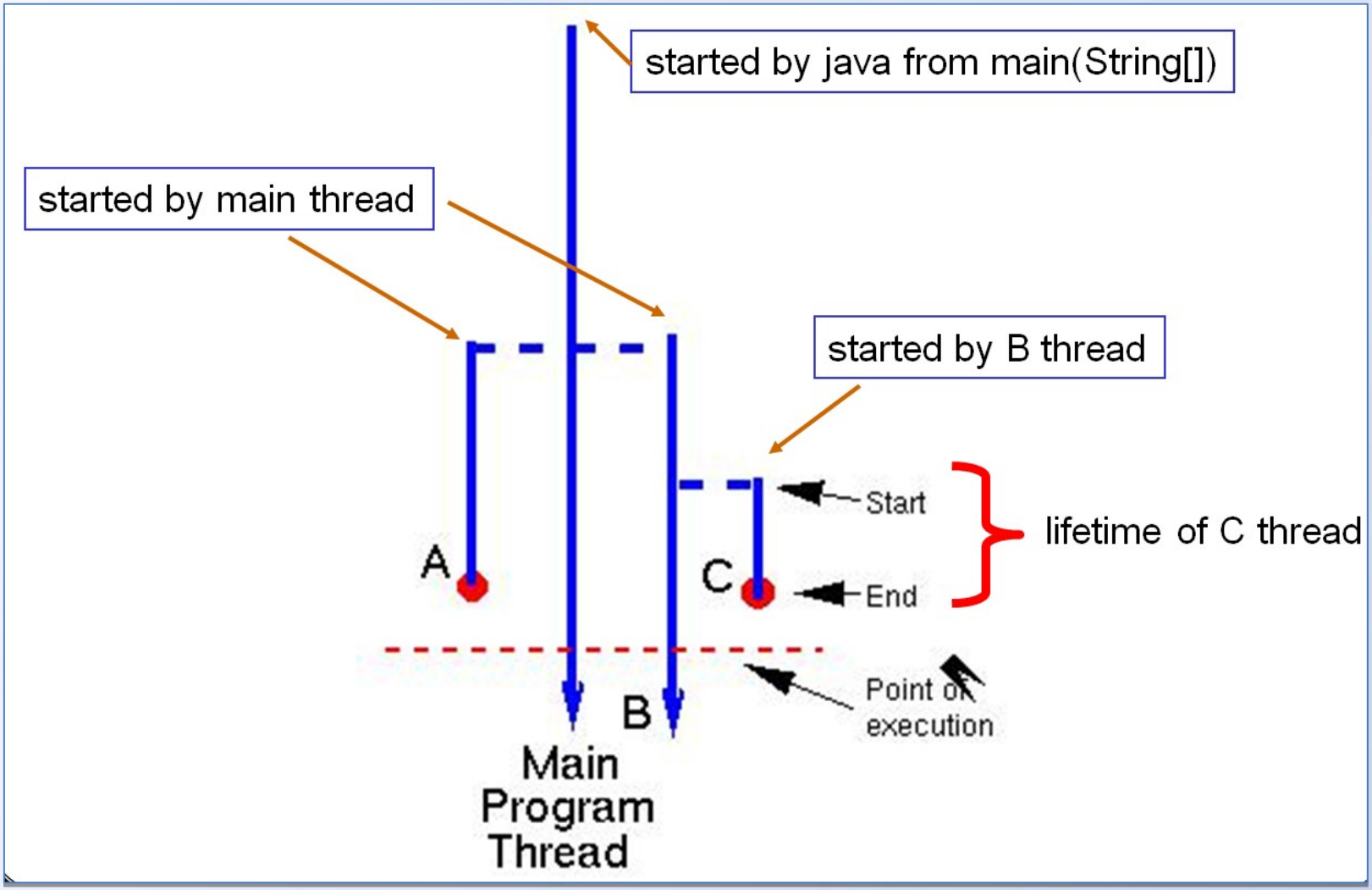
- Natural Application Structure
  - The world is not sequential! Easier to program multiple independent and concurrent activities.
- Increased application throughput and responsiveness
  - Not blocking the entire application due to blocking IO
- Performance from ultiprocessor/multicore
- Distributed systems
  - Single application on multiple machines

Built - in Java Threads

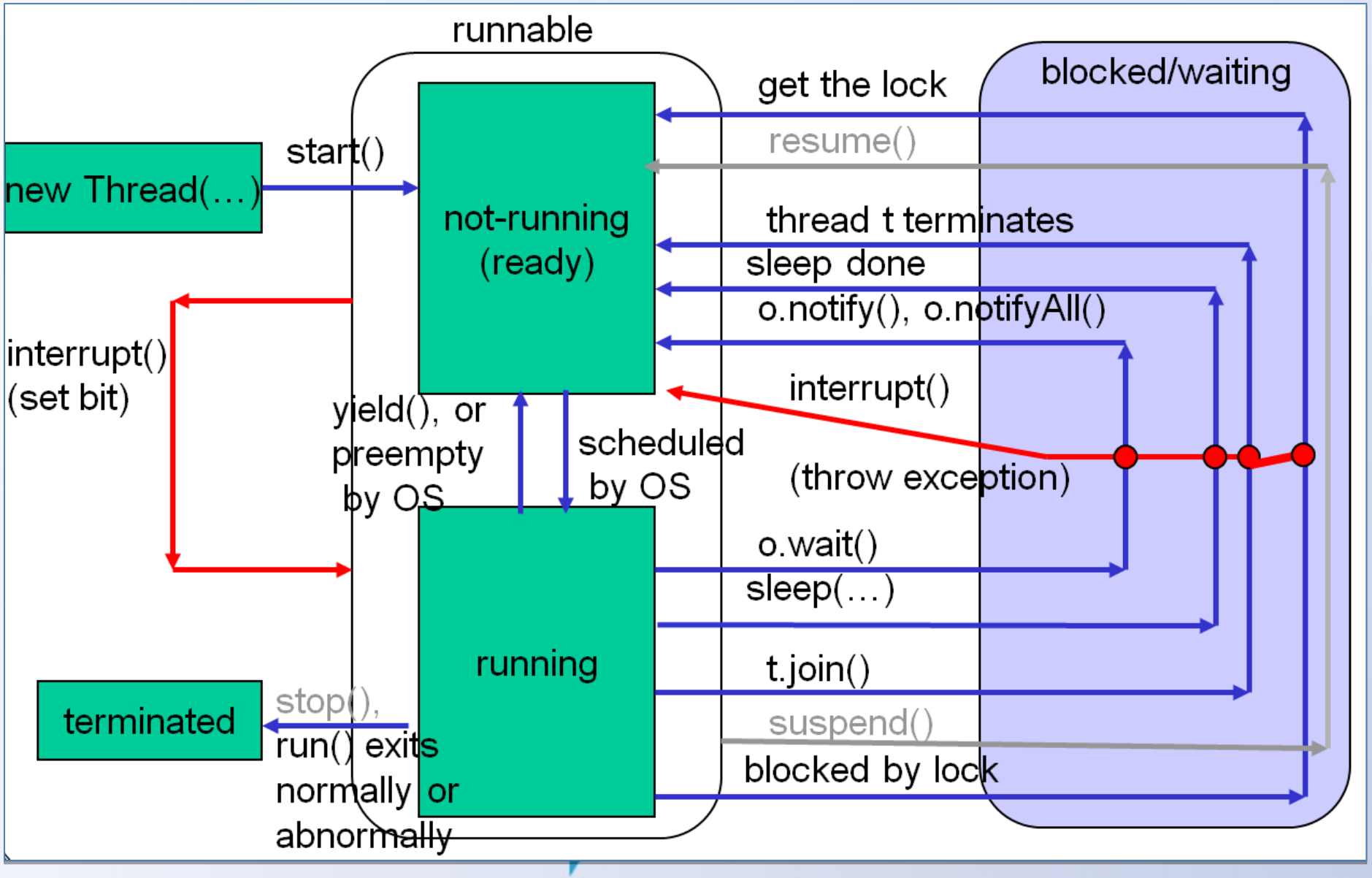
- java.lang.Thread
  - Thread t = new Thread();
  - t.start();
- java.lang.Runnable
  - Runnable r = new Runnable() {
  - public void run() {...}
  - }
- Thread actually extends Runnable
  - Thread t = new Thread(r);
  - t.start() -> actually calls r.run(), but r.run() DOES NOT start in parallel

Built - in Java Threads

Thread ecology in a java program



Life cycle of a thread



Thread methods

- Thread constructor ...
- sleep(millisec) – pauses execution
- interrupt()
- while (true) {
  - if (isInterrupted()) return;
- }
- t.join() - lets current thread wait for receiver thread (t)

to end for at most ms+ns time.

- yield() - Causes the currently executing thread object to temporarily pause and allow other threads to execute

Thread methods

```
1
2  public class SimpleThreads {// Display a message, preceded by
3      // the name of the current thread
4      static void threadMessage(String message)
5      {
6          String threadName = Thread.currentThread().getName();
7          System.out.format("%s: %s%n", threadName, message);
8      }
9
10     private static class MessageLoop implements Runnable
11     {
12         public void run()
13         {
14             String importantInfo[] =
15             {
16                 "Mares eat oats",
17                 "Does eat oats",
18                 "Little lambs eat ivy",
19                 "A kid will eat ivy too"
20             };
21             try
22             {
23                 for (int i = 0; i < importantInfo.length; i++)
24                 {
25                     // Pause for 4 seconds
26                     Thread.sleep(4000);
27                     // Print a message
28                     threadMessage(importantInfo[i]);
29                 }
30             }
31             catch (InterruptedException e)
32             {
33                 threadMessage("I wasn't done!");
34             }
35         }
36     }
37
```

Thread methods

```
38      public static void main(String args[]) throws InterruptedException
39      {
40              // Delay, in milliseconds before we interrupt MessageLoop thread (default one hour).
41              long patience = 10000;
42              // If command line argument present, gives patience in seconds.
43
44              threadMessage("Starting MessageLoop thread");
45              long startTime = System.currentTimeMillis();
46              Thread t = new Thread(new MessageLoop());
47              t.start();
48
49              threadMessage("Waiting for MessageLoop thread to finish");
50              // loop until MessageLoop
51              // thread exits
52              while (t.isAlive())
53              {
54                  threadMessage("Still waiting...");
55                  // Wait maximum of 1 second
56                  // for MessageLoop thread
57                  // to finish.
58                  t.join(2500);
59                  if (((System.currentTimeMillis() - startTime) > patience) && t.isAlive())
60                  {
61                      threadMessage("Tired of waiting!");
62                      t.interrupt();
63                      // Shouldn't be long now
64                      // -- wait indefinitely
65                  }
66              }
67              threadMessage("Finally!");
68          }
69  }
```