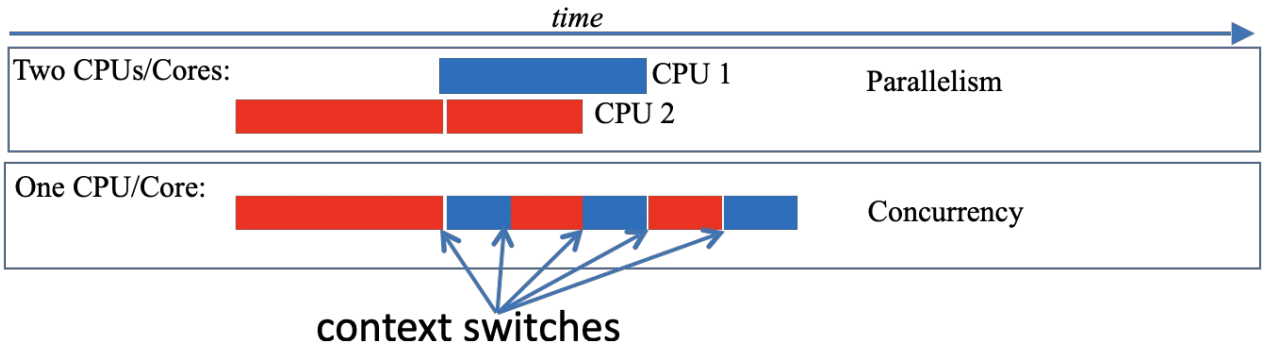
Concurrency: Threads

Tutorial 13 (19 May 2021)

Daniël Kuijsters

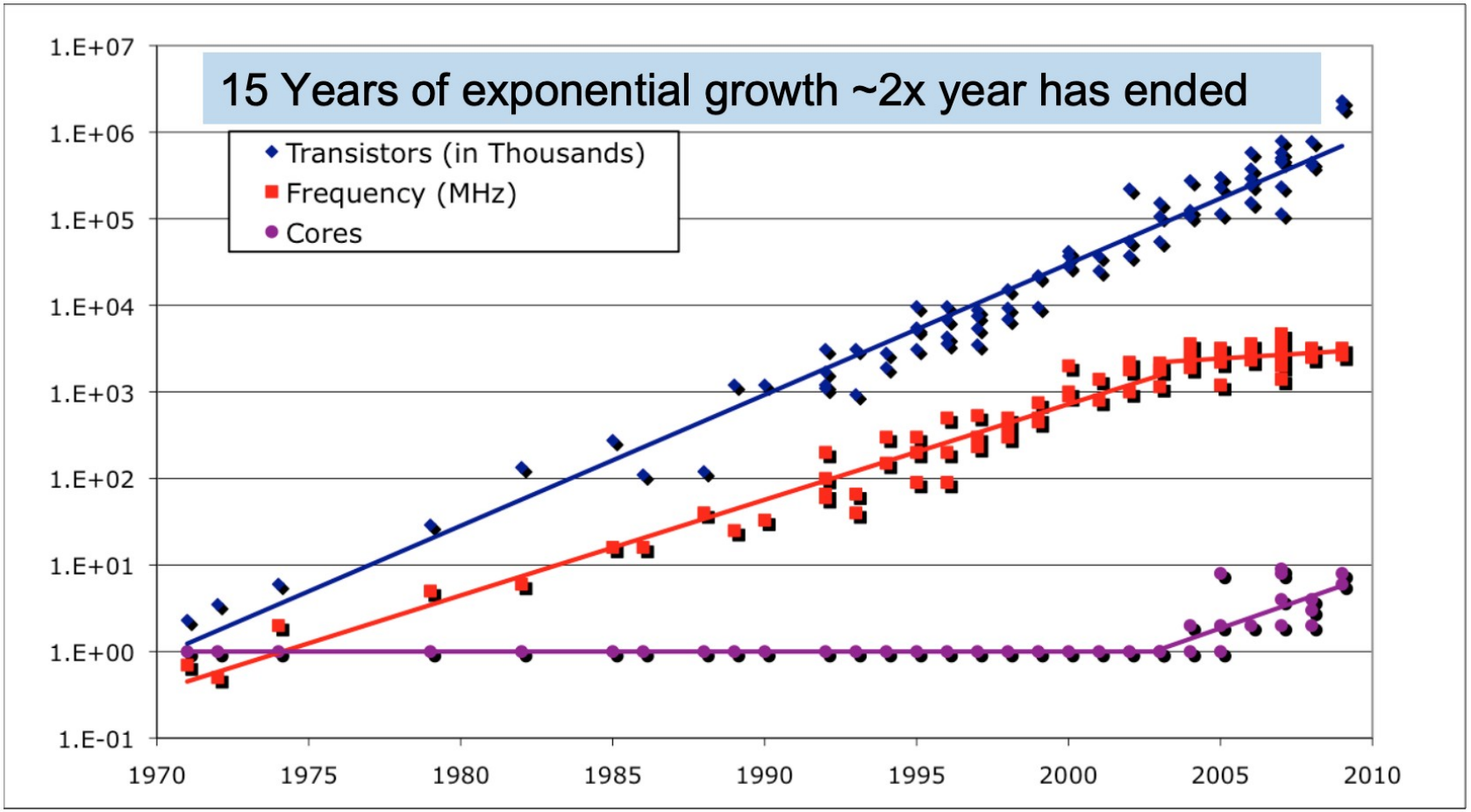
WE START AT 8:30

Parallelism Versus Concurrency

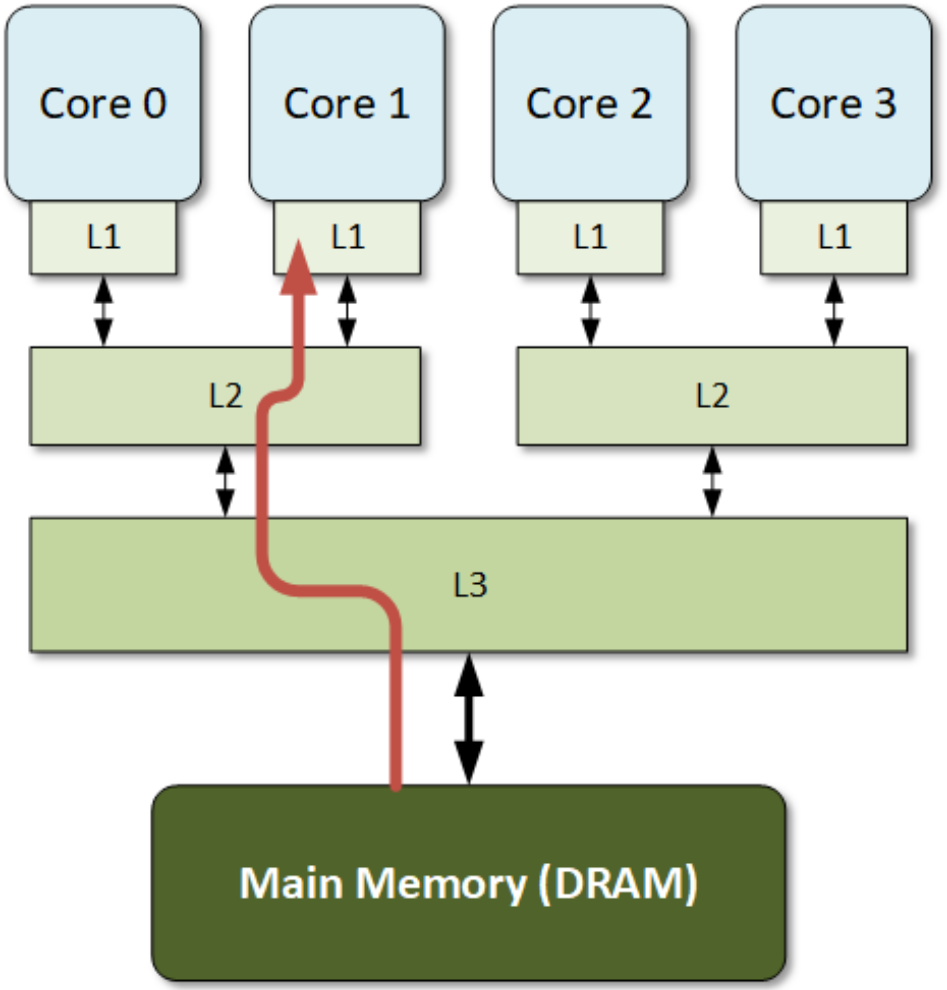


- Concurrency: At least two tasks are making progress at the same time. Used to hide latency.
- Parallelism: At least two tasks are being executed at the same time. Used to increase the speed or throughput.
- Parallelism implies concurrency.

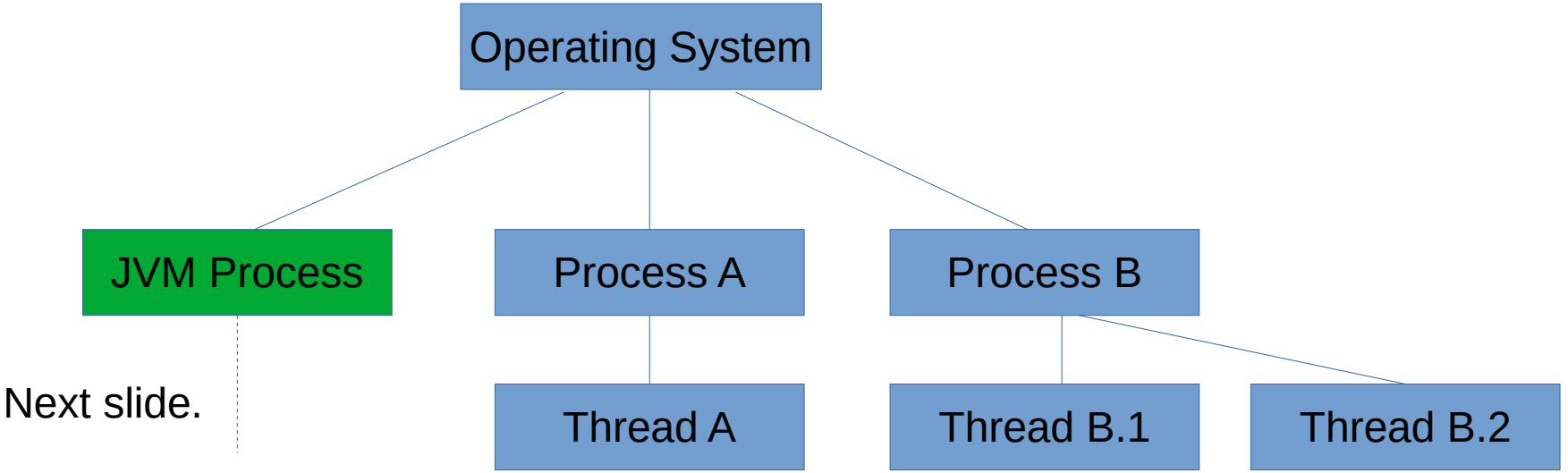
Moore's Law


15 Years of exponential growth ~2x year has ended
Transistors (in Thousands)
Frequency (MHz)
Cores
1.E+07
1.E+06
1.E+05
1.E+04
1.E+03
1.E+02
1.E+01
1.E+00
1.E-01
1970
1975
1980
1985
1990
1995
2000
2005
2010

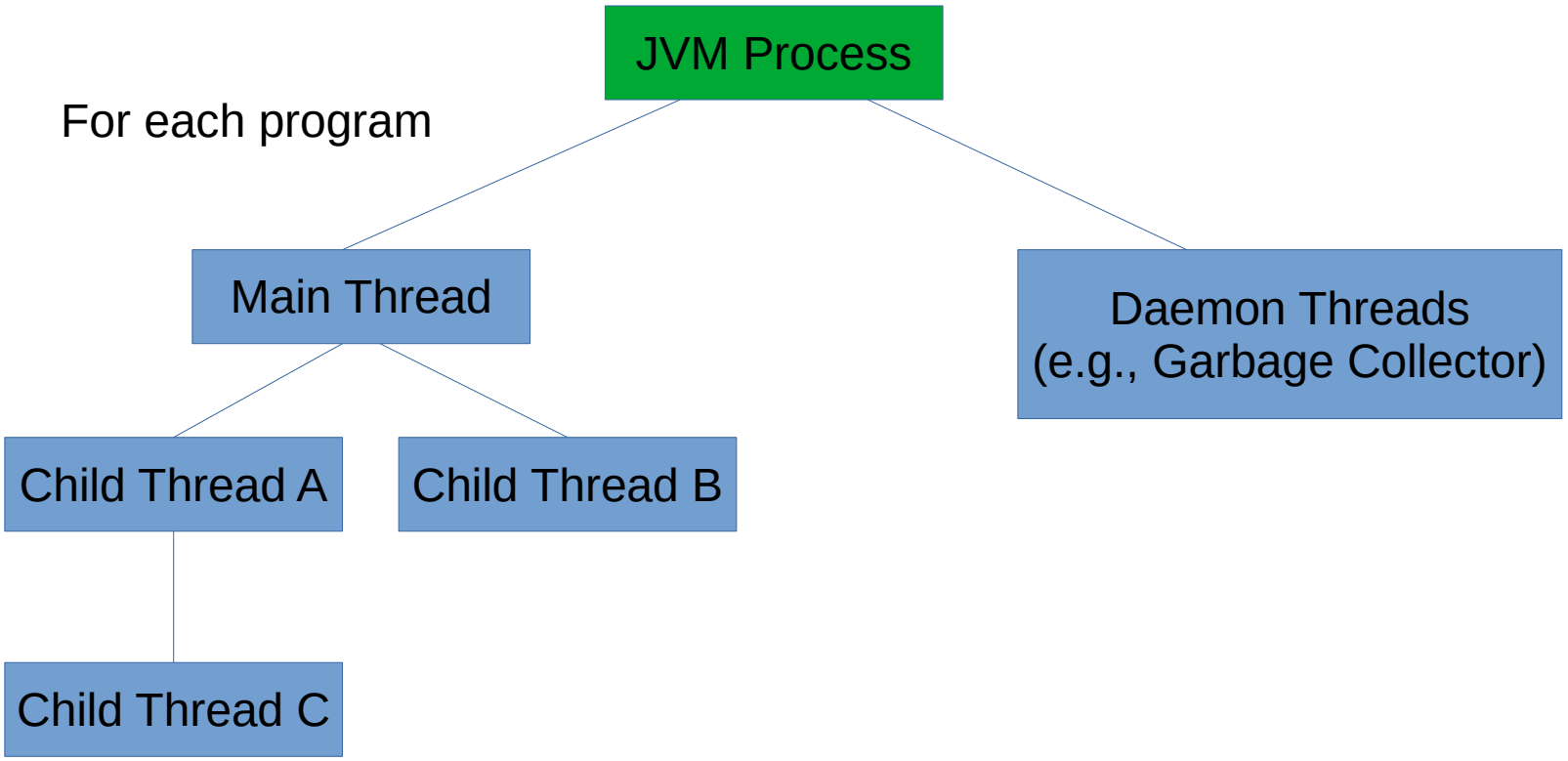
Possible Memory Architecture Multicore System



Java Virtual Machine (JVM) Process



JVM Threads


JVM Process
For each program
Main Thread
Daemon Threads
(e.g., Garbage Collector)
Child Thread A
Child Thread B
Child Thread C

How Do You Create Threads?

- Implement the Runnable interface

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

- Extend the Thread class

Extending Thread

```
public class Thread implements Runnable {
    ...
}

public class Task extends Thread {
    @Override
    public void run() {
        System.out.println("Run method executed by child thread.");
    }

    public static void main(String[] args) {
        Task t = new Task();
        t.start();
        System.out.println("Main method executed by main thread.");
    }
}
```

DON'T DO THIS!

Implement Runnable: Using Classes

```
public class Task implements Runnable {
    @Override
    public void run() {
        System.out.println("Run method executed by child thread.");
    }

    public static void main(String[] args) {
        Task t = new Task();
        Thread thread = new Thread(t);
        thread.start();
        System.out.println("Main method executed by main thread.");
    }
}
```

Implement Runnable: Using Classes

```
public class Task extends SomeSuperClass implements Runnable {
    @Override
    public void run() {
        System.out.println("Run method executed by child thread.");
    }

    public static void main(String[] args) {
        Task t = new Task();
        Thread thread = new Thread(t);
        thread.start();
        System.out.println("Main method executed by main thread.");
    }
}
```

Replacing start() with run(): What Happens?

```
public class Task implements Runnable {
    public Task() {
        Thread t = new Thread(this);
        t.run();
    }

    @Override
    public void run() {
        System.out.println("test");
    }

    public static void main(String[] args) {
        new Task();
    }
}
```

What Is Wrong In the Following Program?

```
public class Task implements Runnable {
    public Task() {
        Task t = new Task();
        new Thread(t).start();
    }

    @Override
    public void run() {
        System.out.println("test");
    }

    public static void main(String[] args) {
        new Task();
    }
}
```

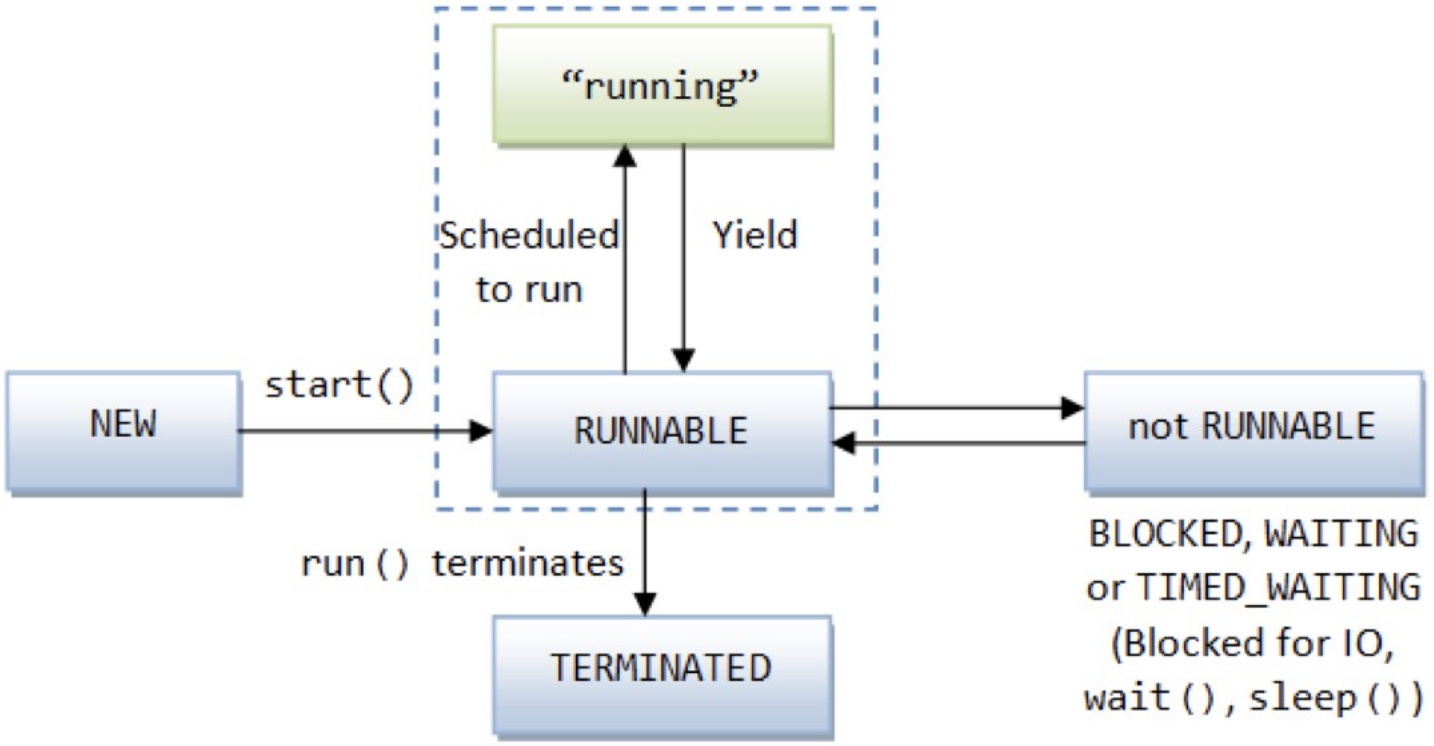
What Is Wrong In the Following Program (2)?

```
public class Task implements Runnable {
    public Task() {
        Thread t = new Thread(this);
        t.start();
        t.start();
    }

    @Override
    public void run() {
        System.out.println("test");
    }

    public static void main(String[] args) {
        new Task();
    }
}
```

Recall: Thread Finite State Machine



What Is Wrong In the Following Program (3)?

```
public class Main {
    private class Task implements Runnable {
        @Override
        public void run() {
            System.out.println("test");
        }
    }

    public static void main(String[] args) {
        Task task = new Task();
        Thread thread = new Thread(task);
        thread.start();
    }
}
```

Implement Runnable: Using Anonymous Inner Classes

```
public class Task {
    public static void main(String[] args) {
        Runnable r = new Runnable() {
            @Override
            public void run() {
                System.out.println("Run method executed by child thread.");
            }
        };
        Thread thread = new Thread(r);
        thread.start();
        System.out.println("Main method executed by main thread.");
    }
}
```

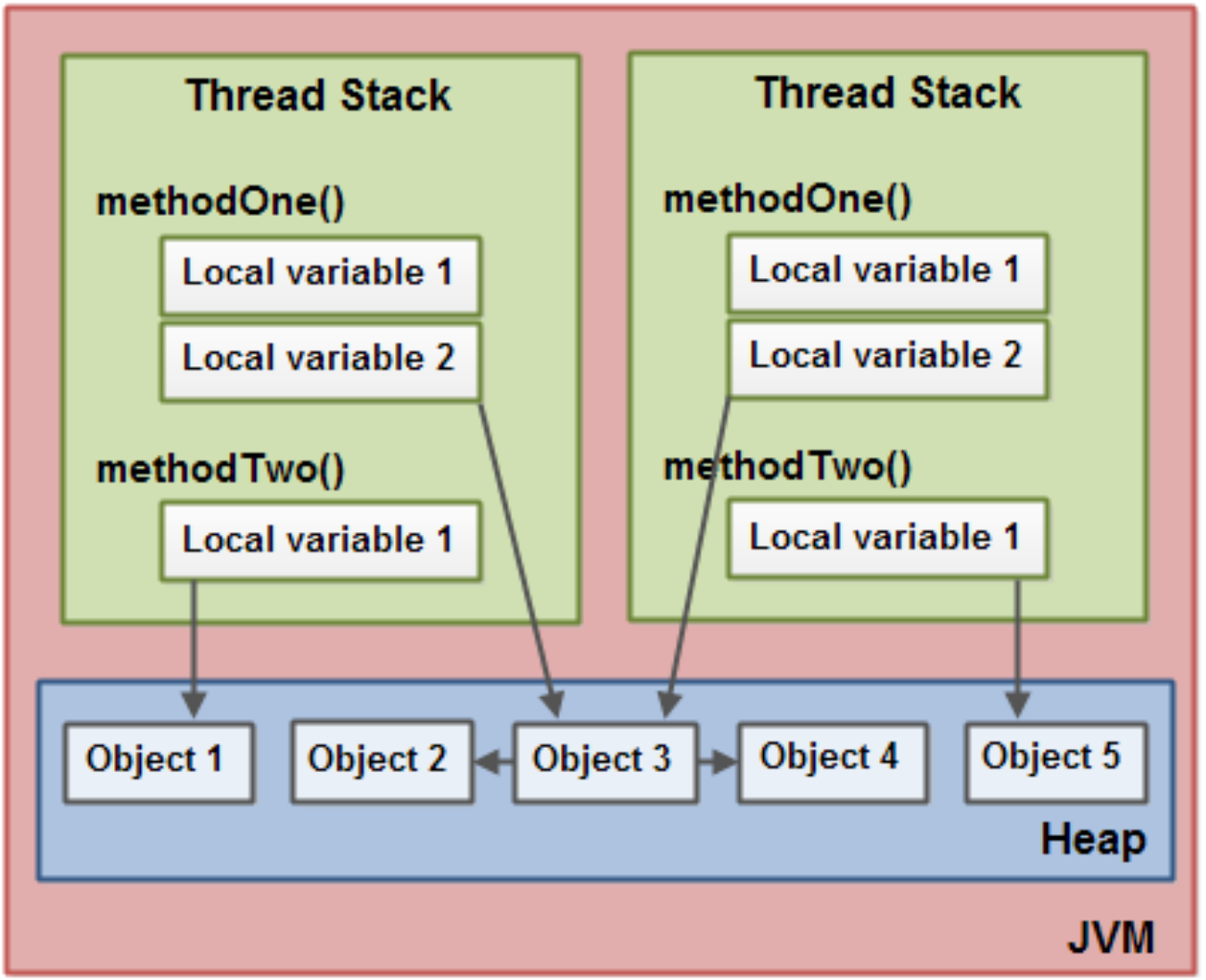
Implement Runnable: Using Lambda Expressions

```
public class Task {
    public static void main(String[] args) {
        Runnable r = () -> {
            System.out.println("Run method executed by child thread.");
        };
        Thread thread = new Thread(r);
        thread.start();
        System.out.println("Main method executed by main thread.");
    }
}
```

Implement Runnable: Using Lambda Expressions

```
public class Task {
    public static void main(String[] args) {
        Thread thread = new Thread(() -> {
            System.out.println("Run method executed by child thread.");
        });
        thread.start();
        System.out.println("Main method executed by main thread.");
    }
}
```

JVM Memory Model

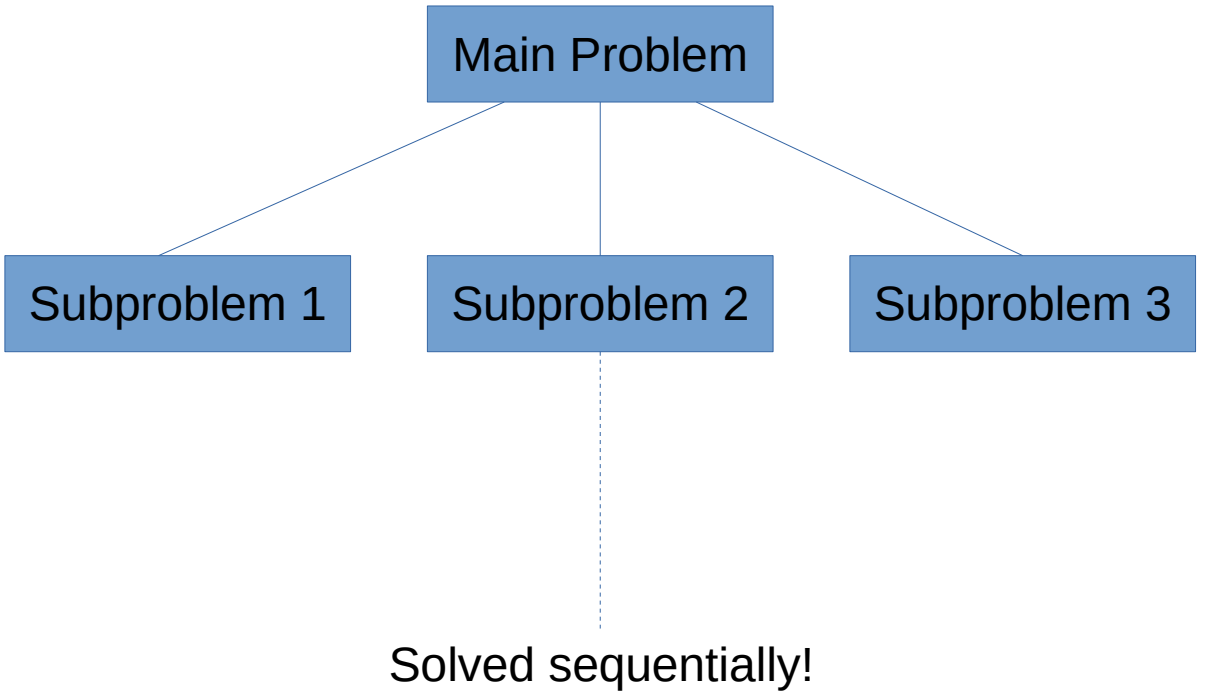


Sharing Data Between Threads Is Difficult!

How to solve this data sharing issue?

- Do NOT share data at all → This week's first assignment.
- Wait for their execution to finish and combine data → This week's second assignment.
- Make use of synchronized statements and/or locks → The next weeks.

Independent Subproblems WITHOUT Combiner



Also called “embarrassingly parallel” problems.

How Many Threads Should I Create?

- Not an exact science; avoid “too little” and “too many”.
- For compute-intensive tasks: #threads = #cores + 1.
- For tasks that include I/O or other blocking operations: #threads = #cores * (1 + wait time / compute time).
- In other words: experiment!

```
int N_CPUS = Runtime.getRuntime().availableProcessors();
```

How Many Threads Should I Create (2)?

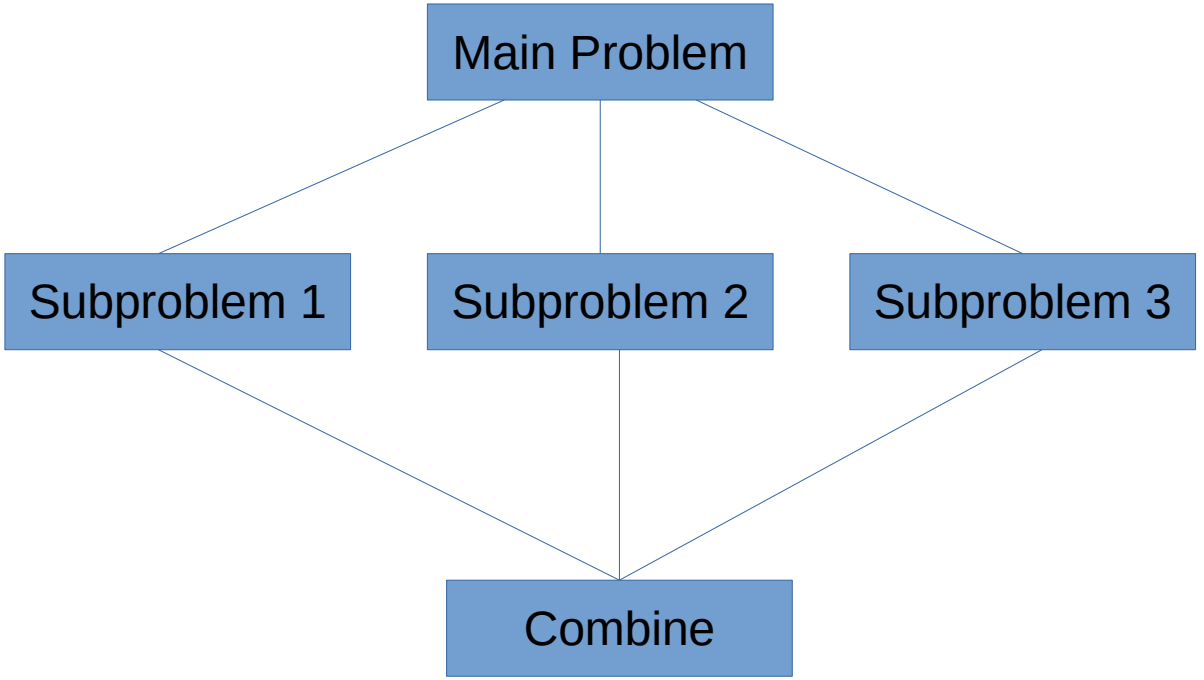
- Main thread is a thread itself, so let it do work too.
- Don’t forget: creating a thread is NOT for free (For this reason, we will use thread pools starting next week).
- Typically we set a threshold below which we use a sequential solution.

Demo: Parallel Find Element

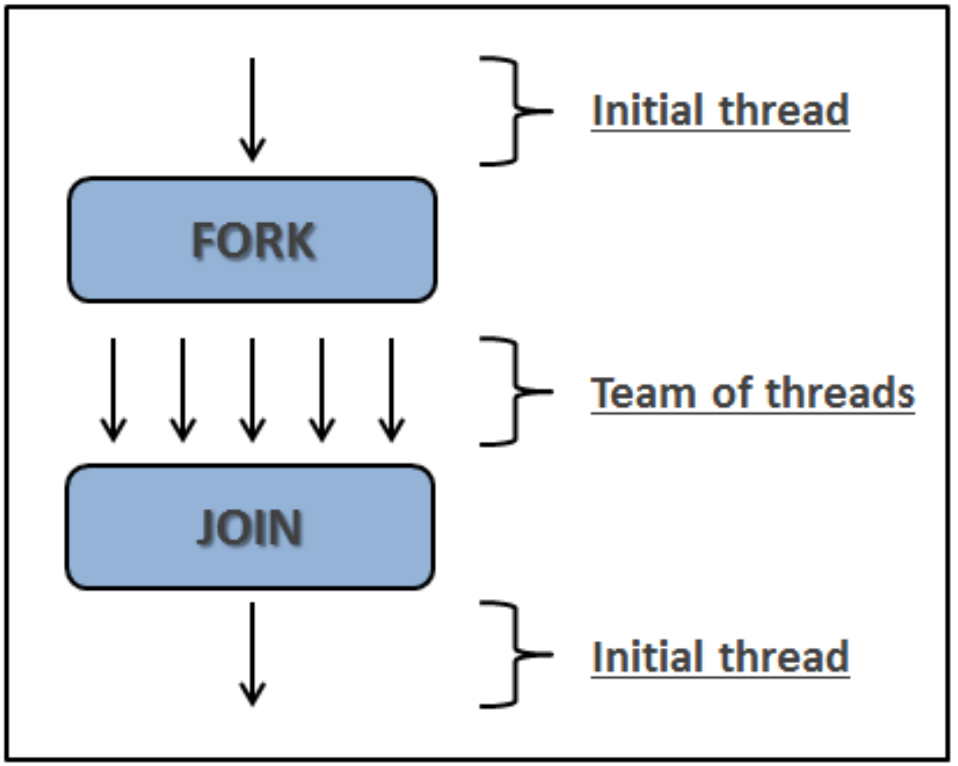
Assignment: FileFinder

- Sequential program that locates a given file in a directory.
- Since there can be many subdirectories and files, this procedure can be slow.
- Goal: speed this up using threads.
- Instantiation of “independent subproblems without combiner”.

Independent Subproblems WITH Combiner

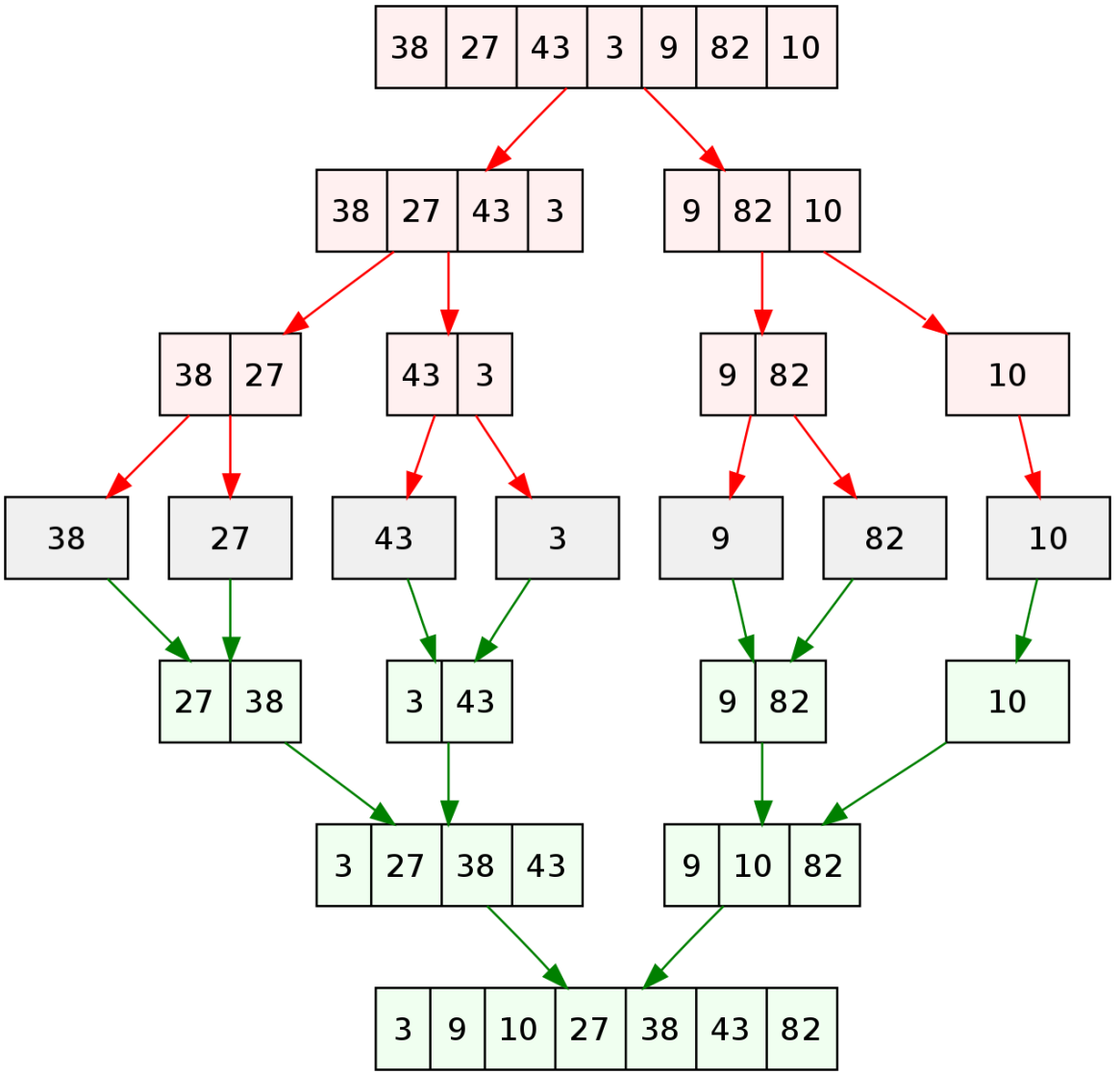


A.K.A. Fork-Join Model



Demo: Parallel Compute Max

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Assignment: Parallel MergeSort

- Merge sort is a sequential sorting algorithm.
- It is very suited for parallelization.
- Goal: speed this up using threads.
- Instantiation of “independent subproblems with combiner”.