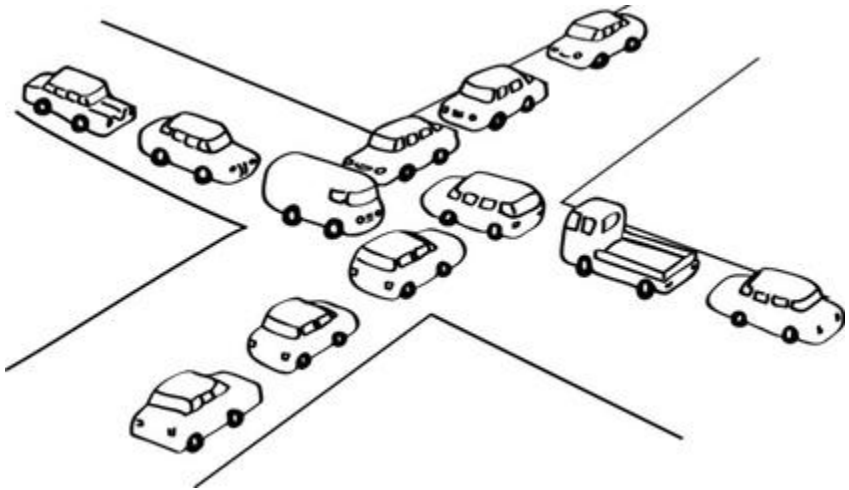
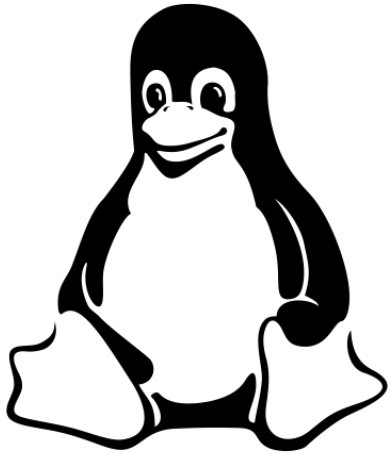
JAVA CONCURRENCY



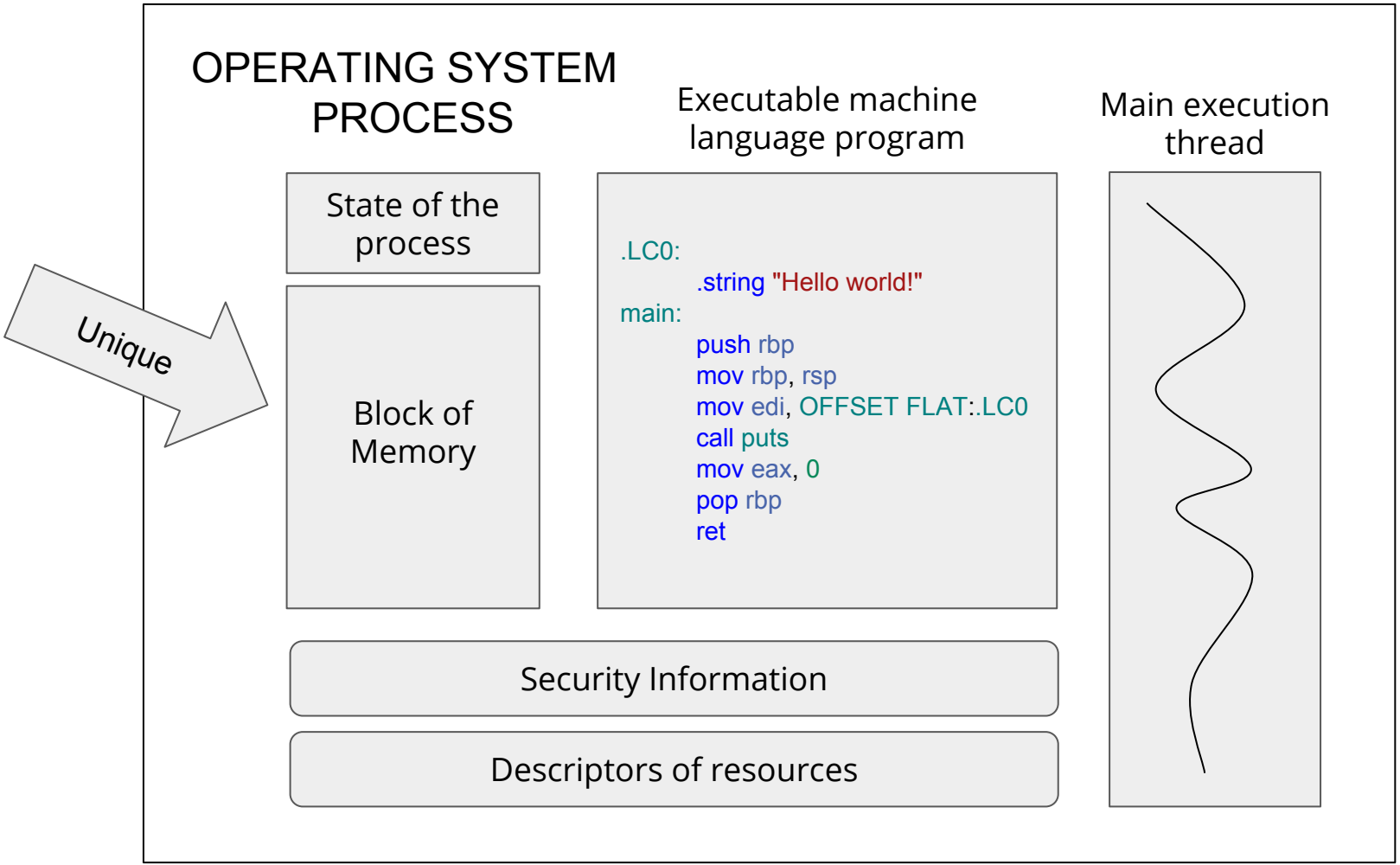
WHAT'S CONCURRENCY?

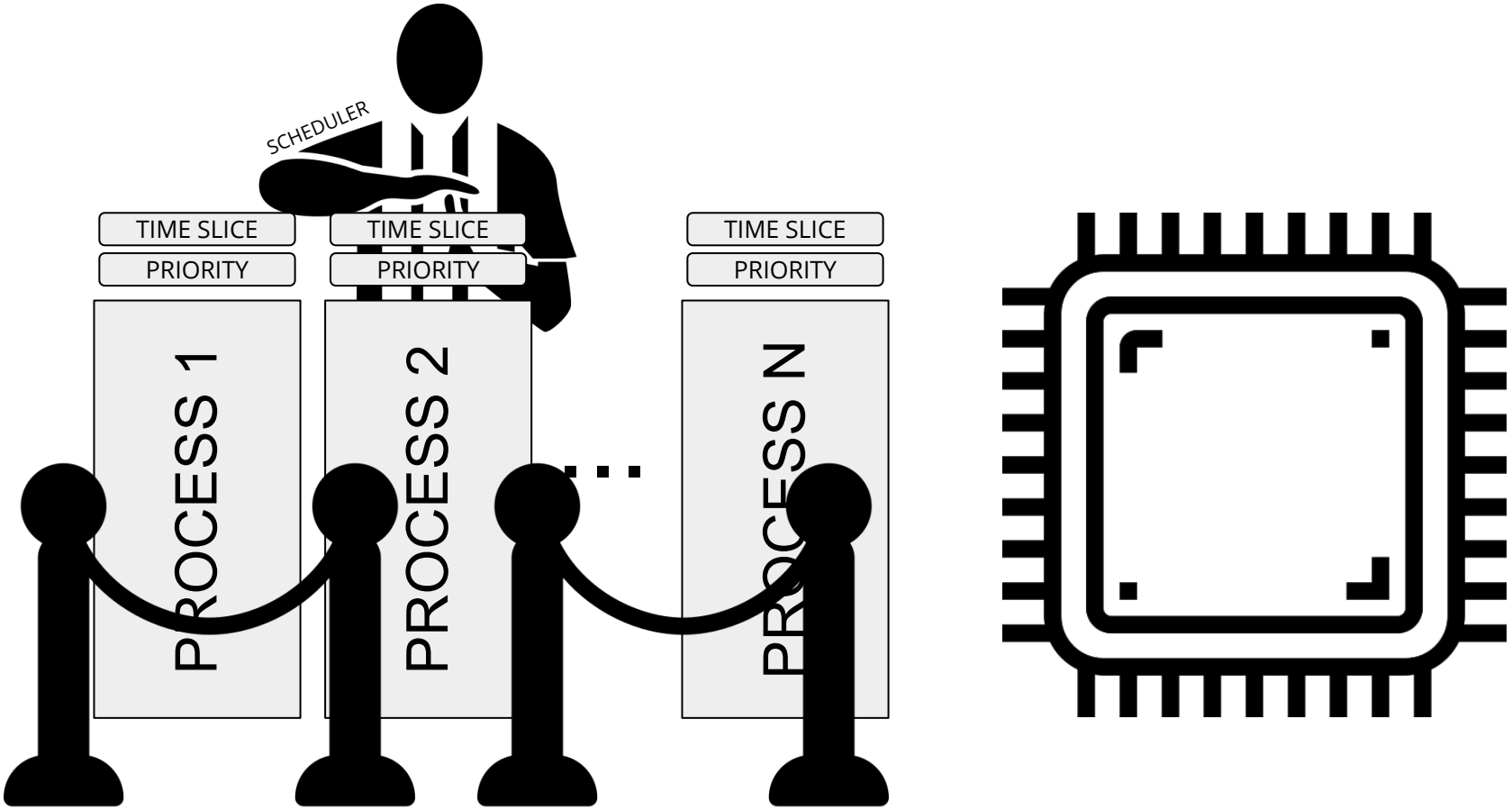
Freddie “xxTh3 U$3rxx”
(Which actually means “The user”)

A program



The Operating System

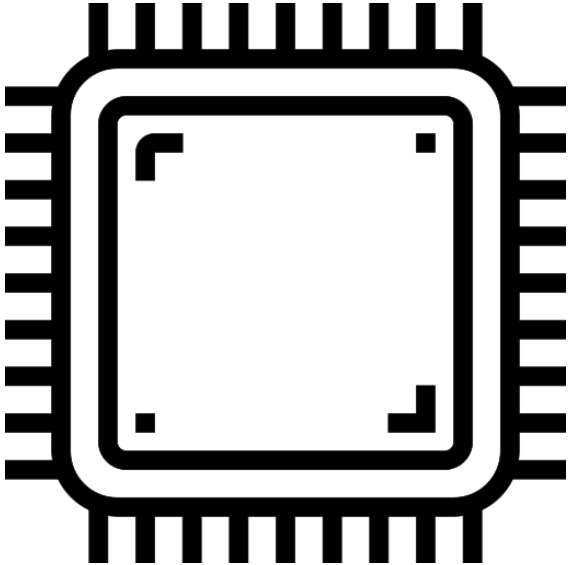

OPERATING SYSTEM PROCESS
Executable machine language program
Main execution thread
State of the process
Unique
Block of Memory
.LC0:
.string "Hello world!"
main:
push rbp
mov rbp, rsp
mov edi, OFFSET FLAT:.LC0
call puts
mov eax, 0
pop rbp
ret
Security Information
Descriptors of resources

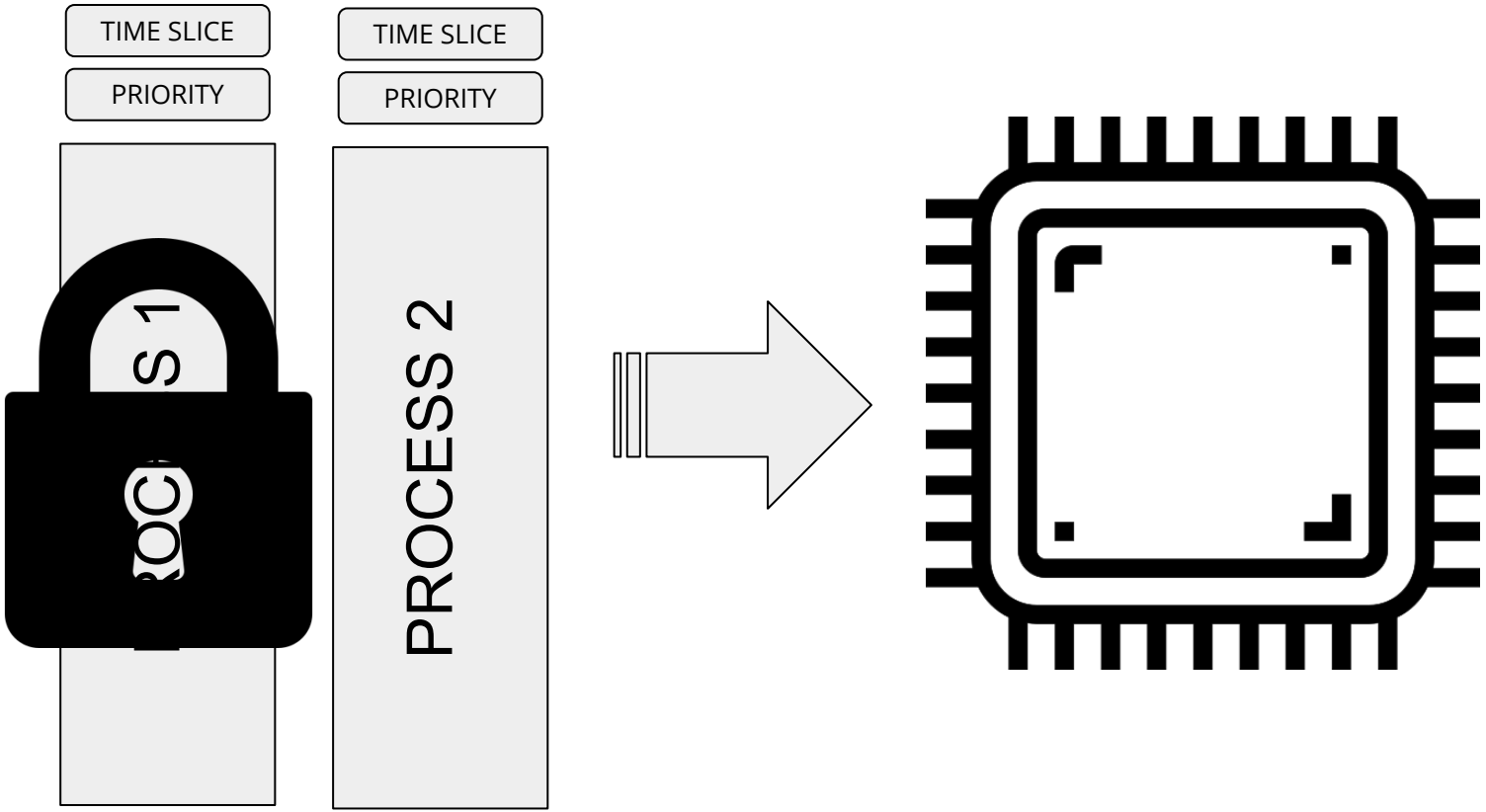

SCHEDULER
TIME SLICE
PRIORITY
TIME SLICE
PRIORITY
TIME SLICE
PRIORITY
PROCESS 1
PROCESS 2
PROCESS N

TIME SLICE

PRIORITY

PROCESS 1




TIME SLICE
PRIORITY
TIME SLICE
PRIORITY
PROCESS 2

TIME SLICE

PRIORITY

PROCESS 1

THREADING

THREAD 1

S 1

FILE READING

THREAD 2

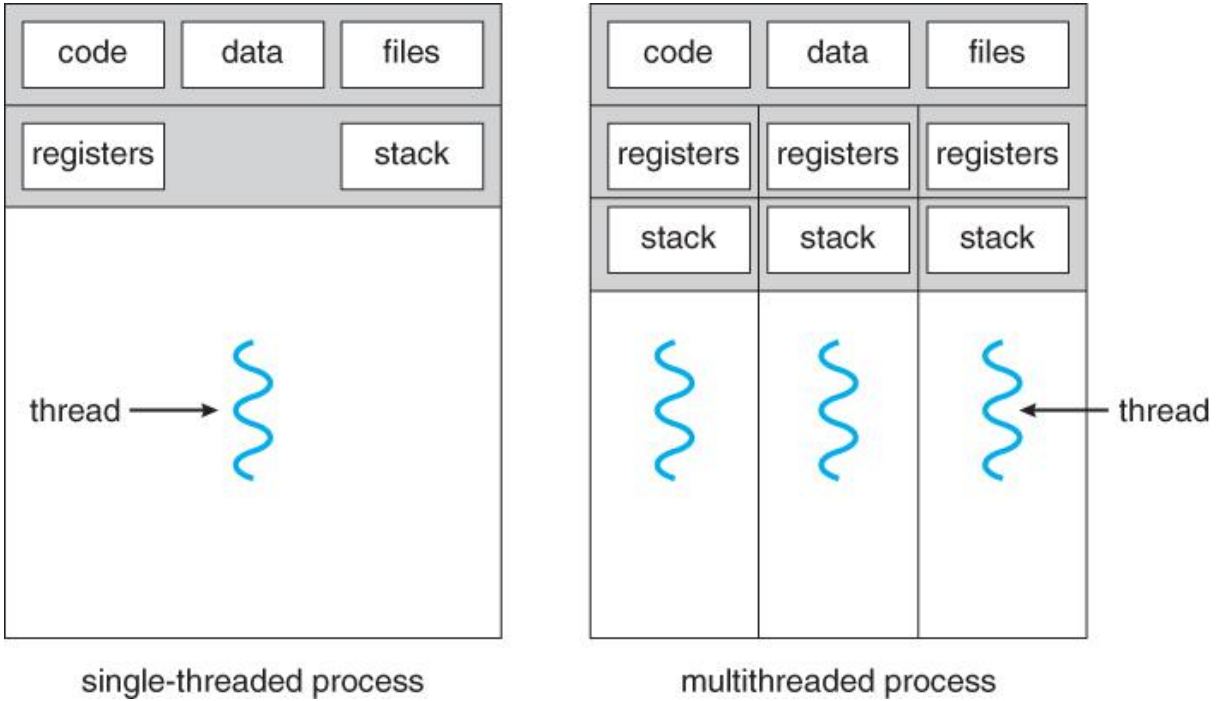
CES

USER INPUT

THREAD 3

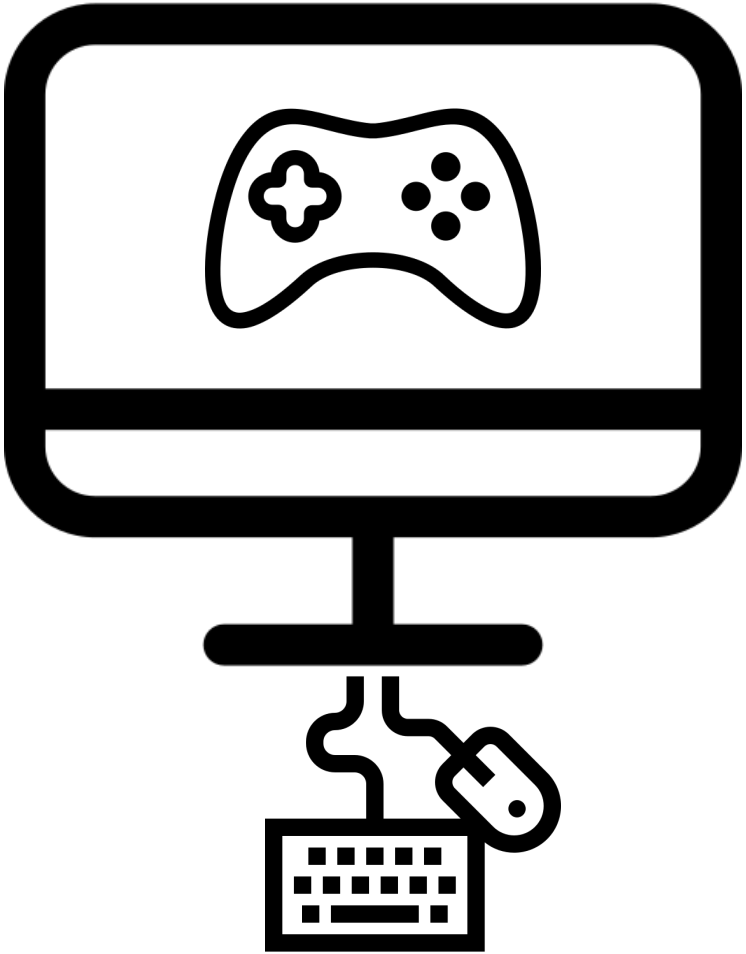
PROC

COMPUTE PROBLEM



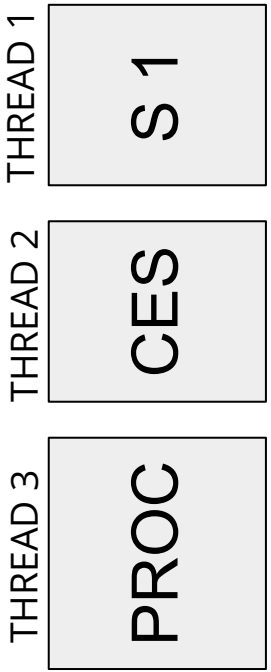
single-threaded process

multithreaded process

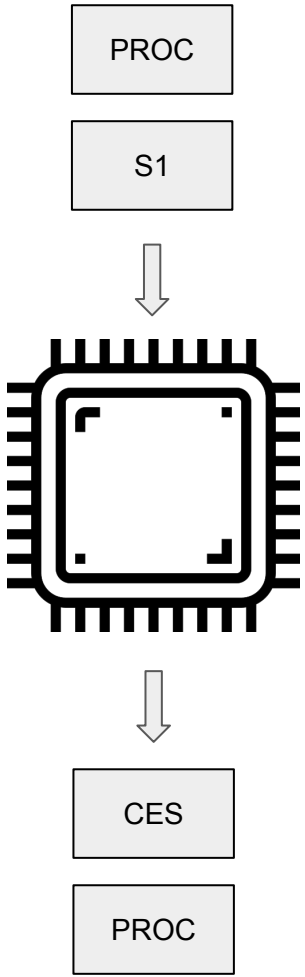


SO
WHAT'S
CONCURRENCY?

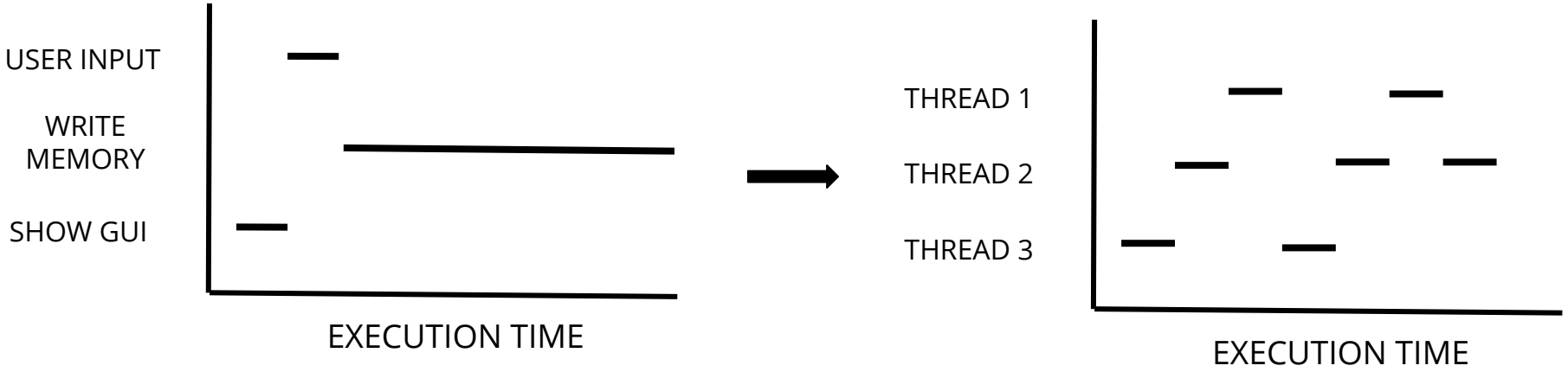
Concurrency consists in the ability of **dealing with multiple task simultaneously.**



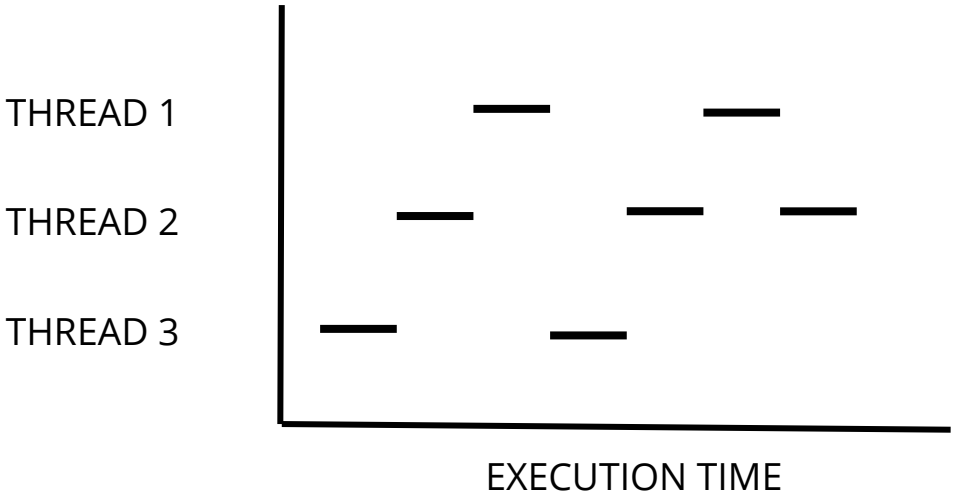
Starting and running them in **overlapping time periods**, which means the application could but **doesn't have to execute tasks in the same time period.**



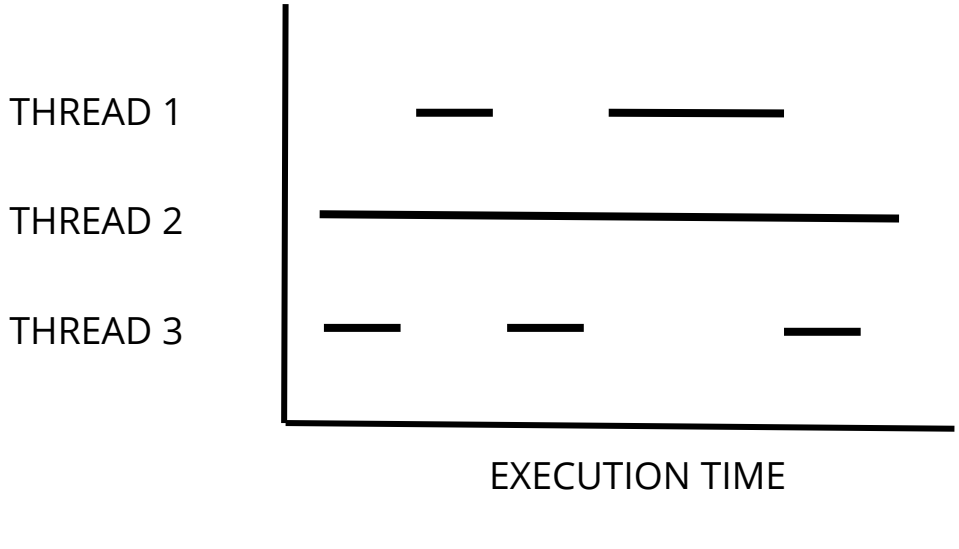
SINGLE CORE CONCURRENCY

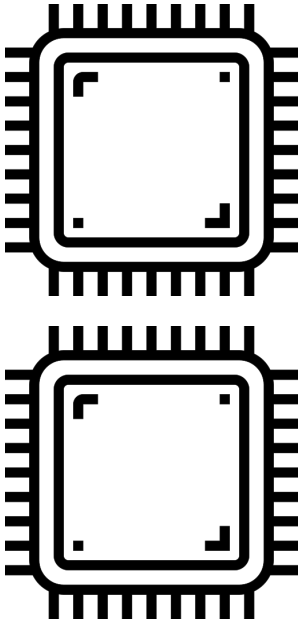


MULTIPLE CORE CONCURRENCY



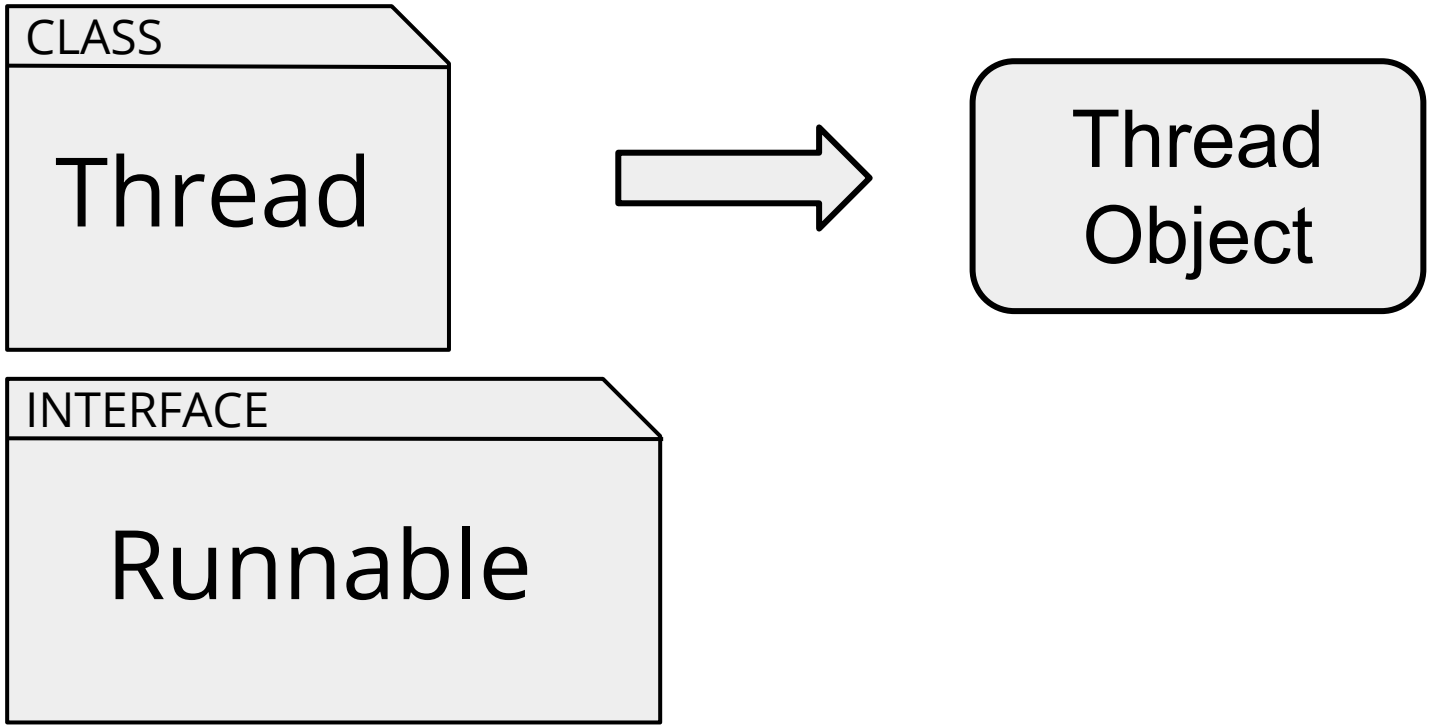
PARALLELISM



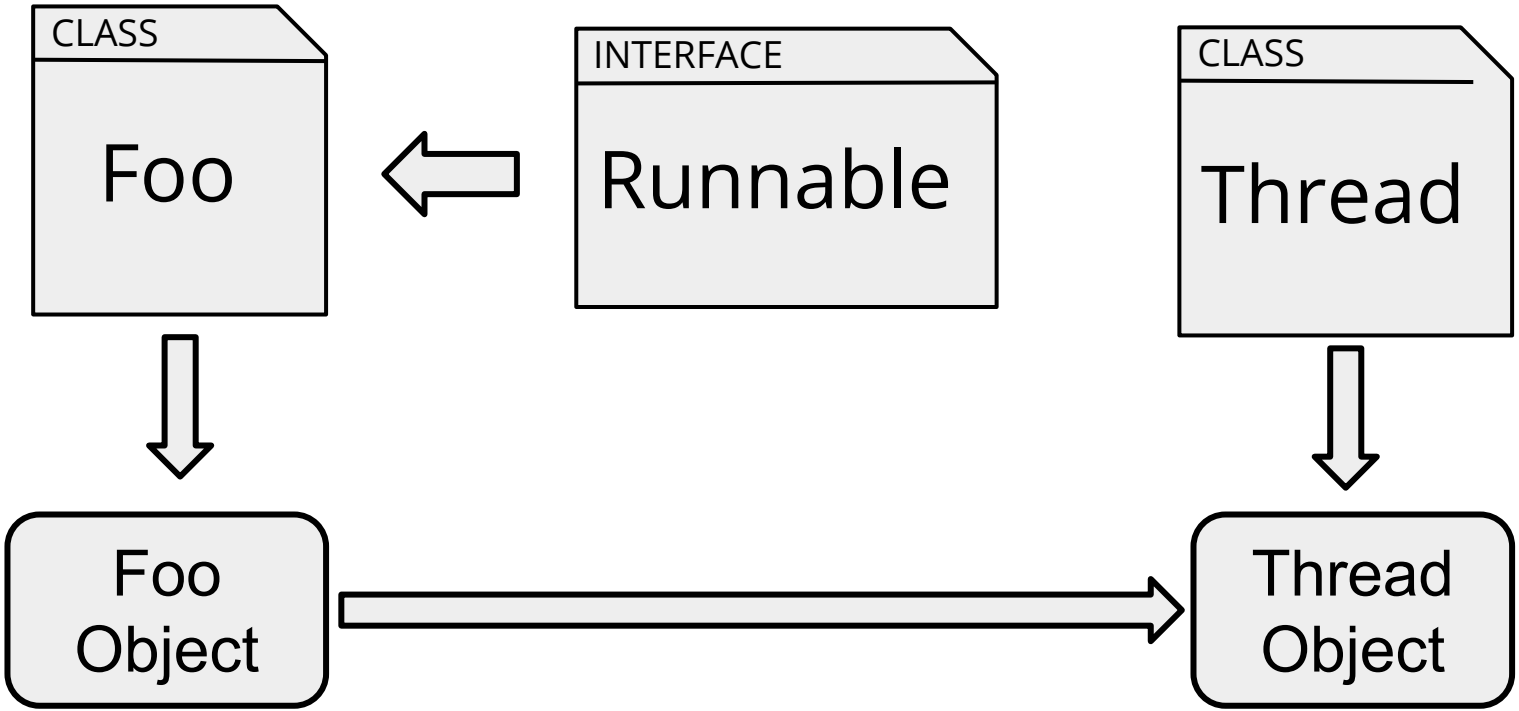


JAVA
CONCURRENCY

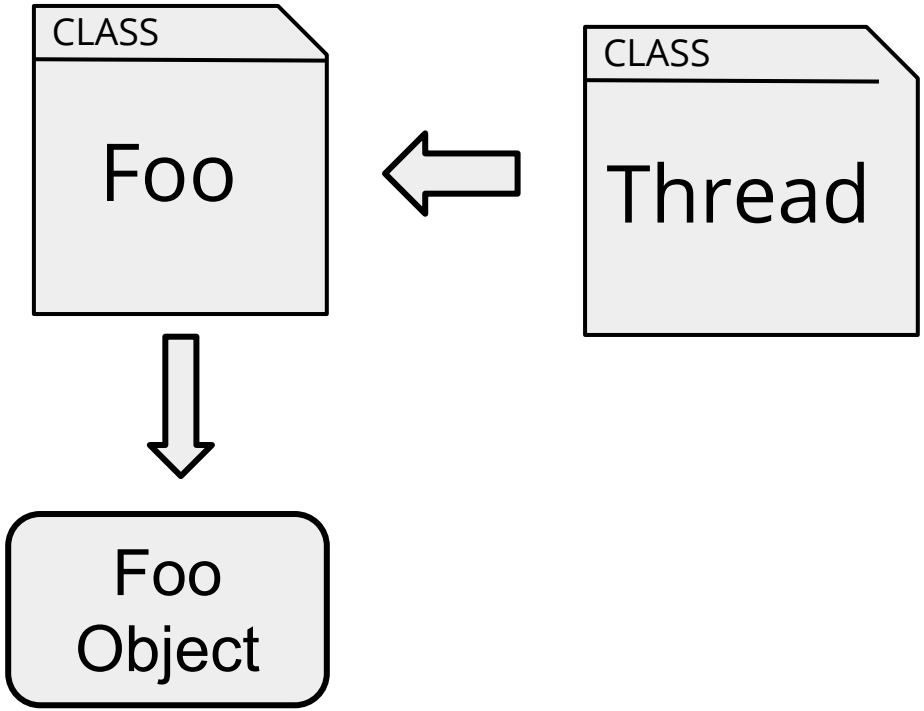
THREAD OBJECTS



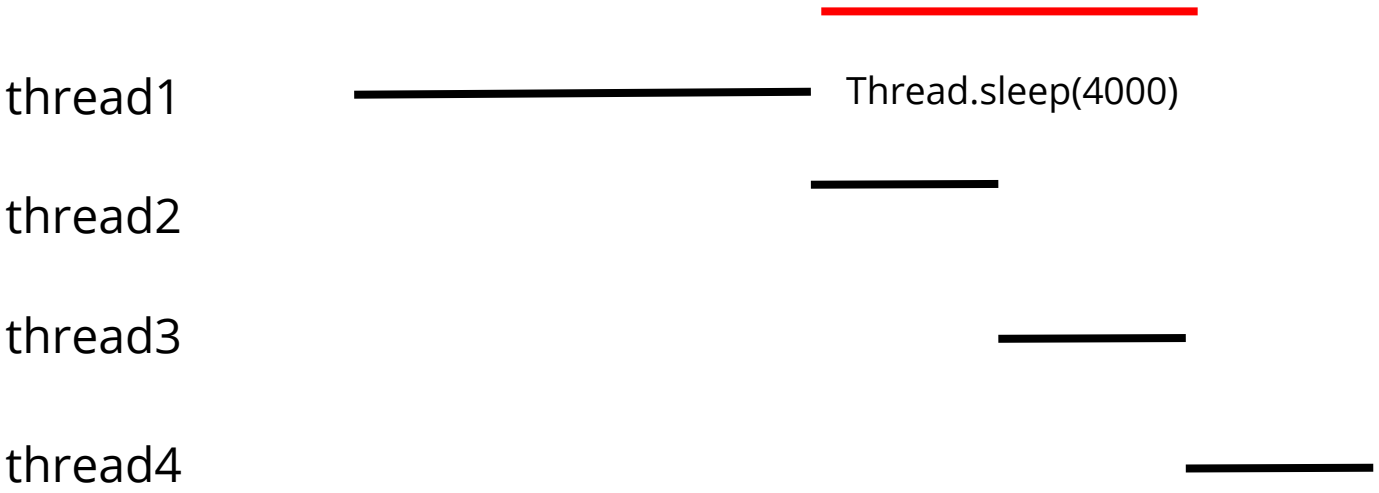
DEFINING AND CREATING A THREAD I



DEFINING AND CREATING A THREAD II



THREAD SLEEP

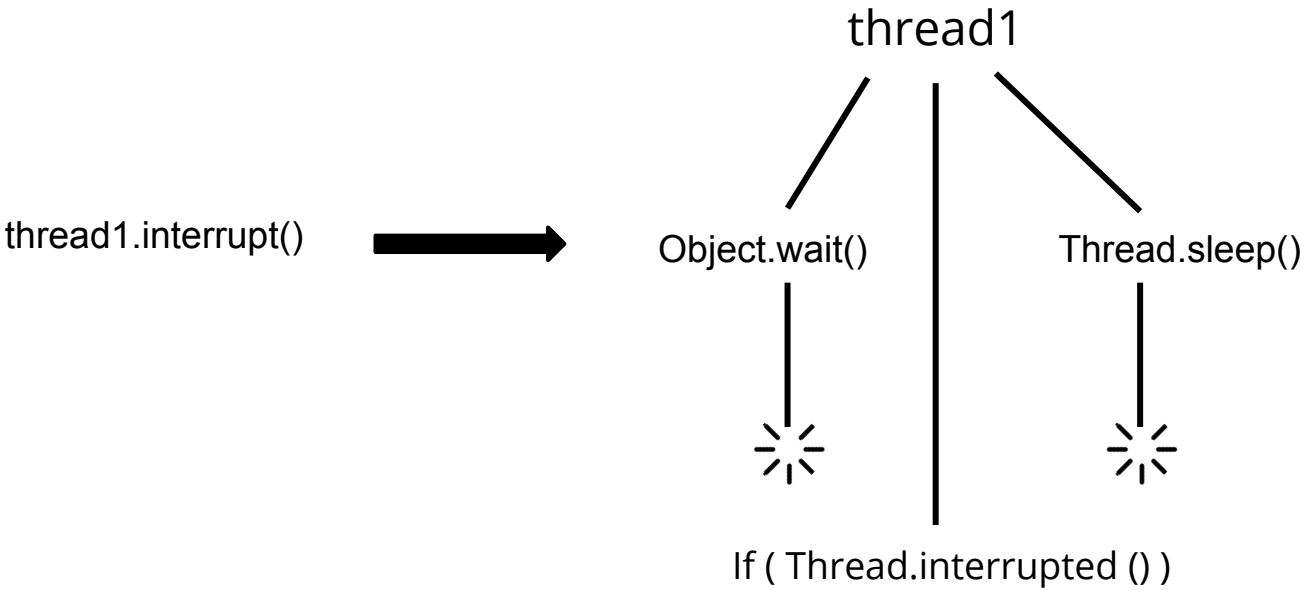


INTERRUPTS

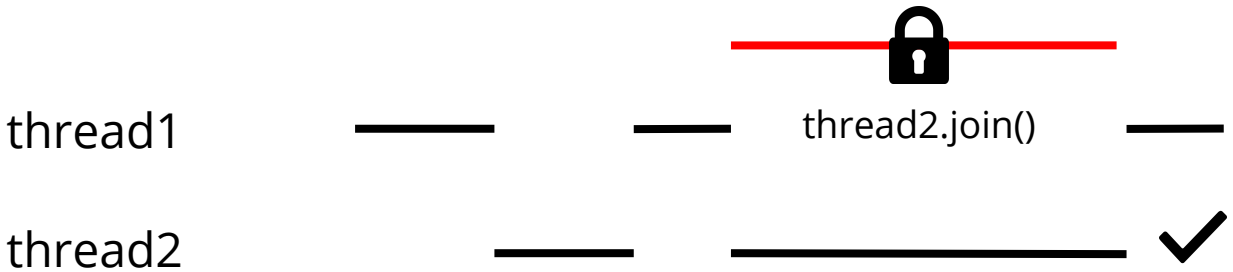
thread1

thread2

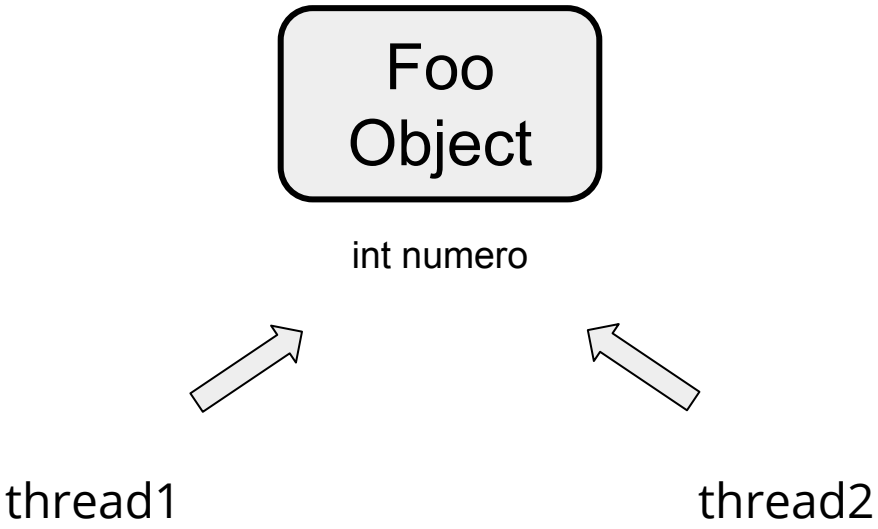
thread1.interrupt()


thread1
thread1.interrupt()
Object.wait()
Thread.sleep()
If (Thread.interrupted ())

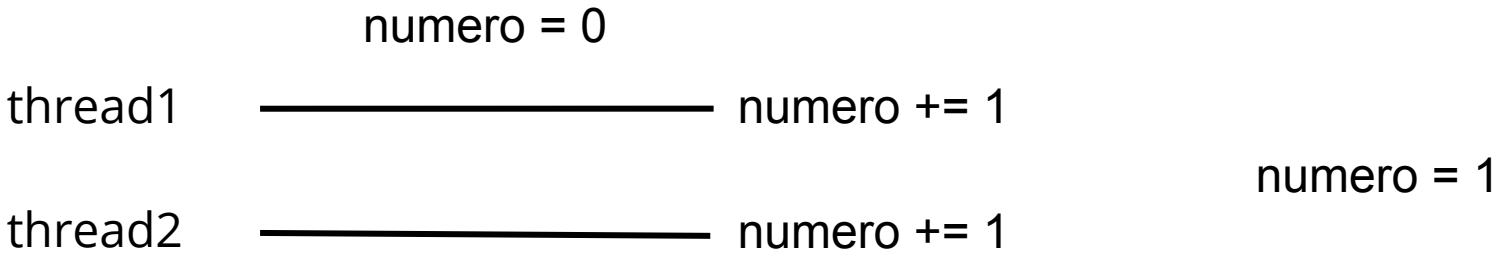
JOINS



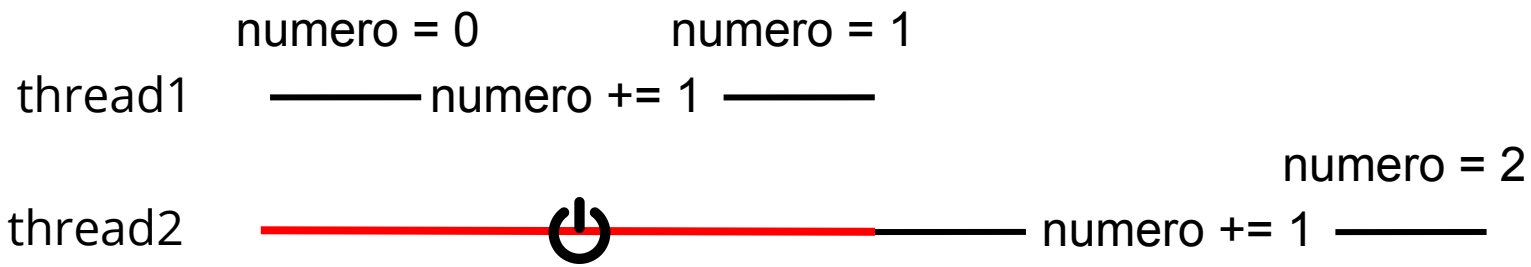
SYNCHRONIZATION



RACE CONDITION



SOLVING RACE CONDITION WITH SYNCHRONIZATION



WAYS OF DECLARING SYNCHRONIZATION

FIRST METHOD

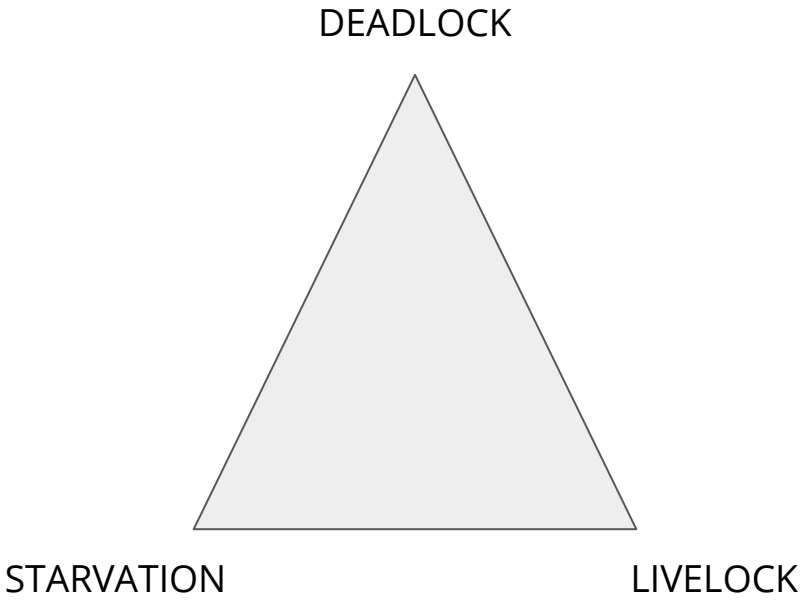
```
public synchronized void run(){

    …

}
```

SECOND METHOD

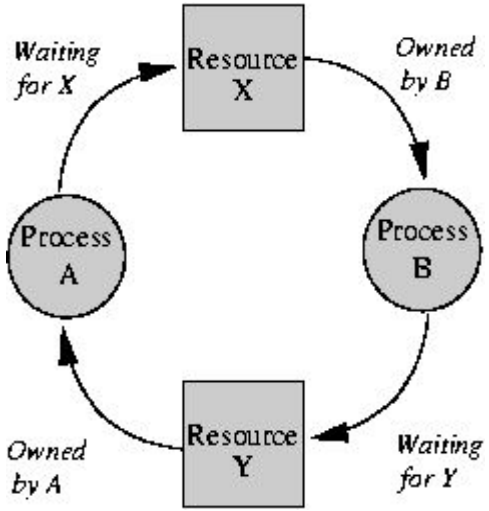
```
public void run(){

    synchronized(this){

        …


    }
}
```

LIVENESS OF A CONCURRENT APPLICATION

“THE UNHOLY TRINITY OF LIVENESS”



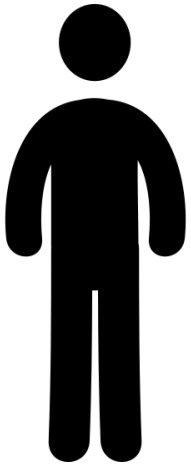
DEADLOCK



It’s all about resource management!

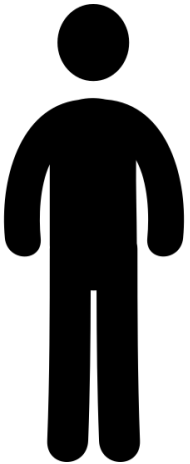
“THE GREETING PROBLEM”

GASTON



They’re friends!
Yay!

ALPHONSE



“THE GREETING PROBLEM”

RULES OF COURTESY:

- Bow when greeting.
- Do not leave the bow until your friend bows back.

But what if they started the bow at the same time?

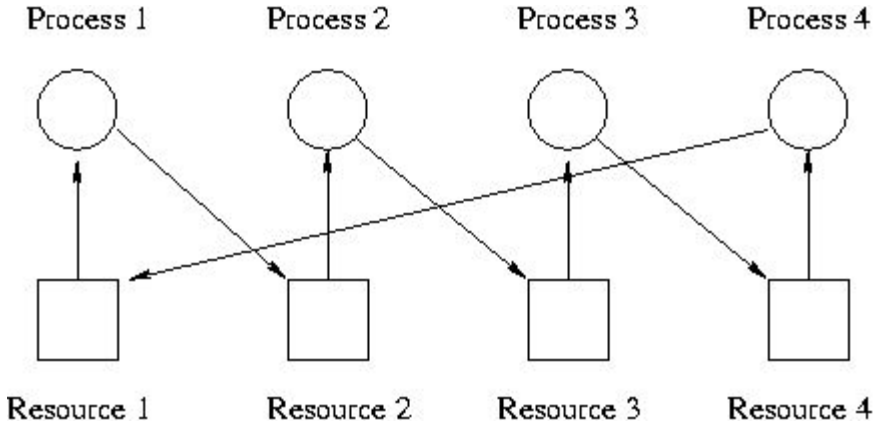
“THE GREETING PROBLEM”

```
public synchronized void bow(Friend bower) {
        // Notices Friend bow
        bower.bowBack(this);
}
public synchronized void bowBack(Friend bower) {
        // Notices Friend bowing back and exits routine
}
```

```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s"
                + "  has bowed to me!%n",
                this.name, bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s"
                + " has bowed back to me!%n",
                this.name, bower.getName());
        }
    }

    public static void main(String[] args) {
        final Friend alphonse =
            new Friend("Alphonse");
        final Friend gaston =
            new Friend("Gaston");
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

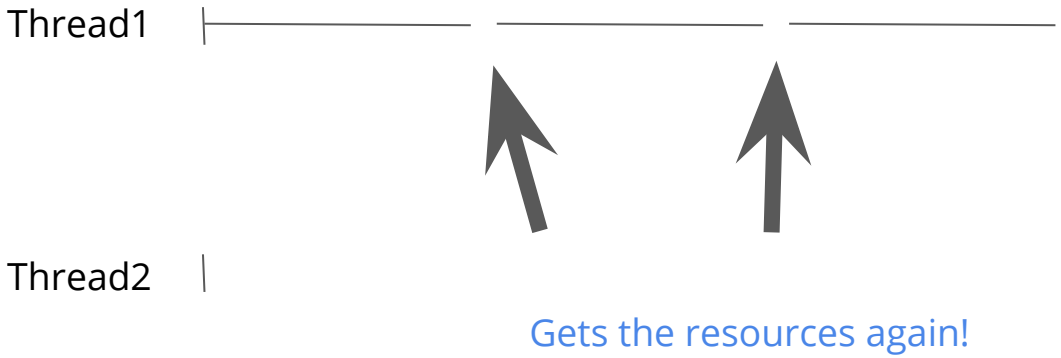
DEADLOCK

- Not always that simple.
- Usually a silent killer.
- Your app will freeze!!!



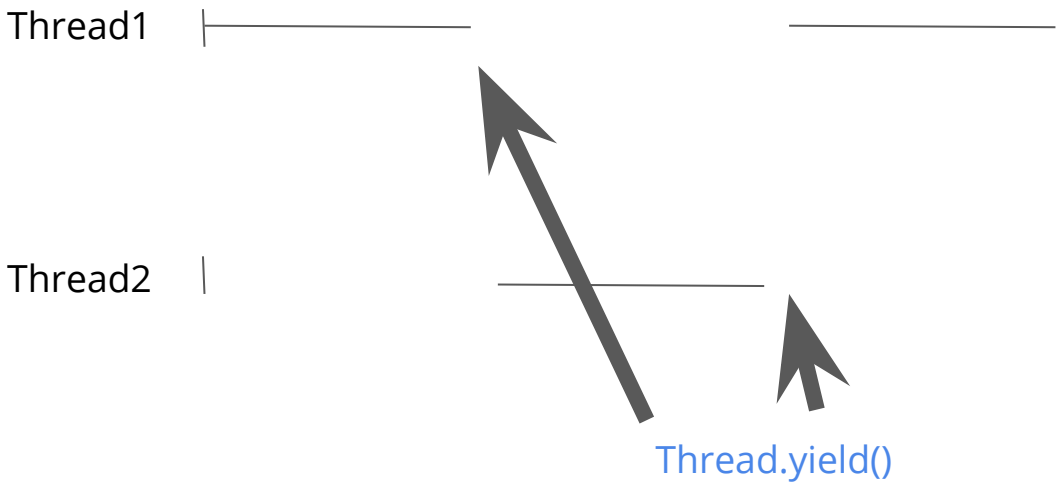
CAREFUL WITH WHO OWNS WHOM’S RESOURCES

STARVATION



A SCHEDULING POLICY IS NEEDED

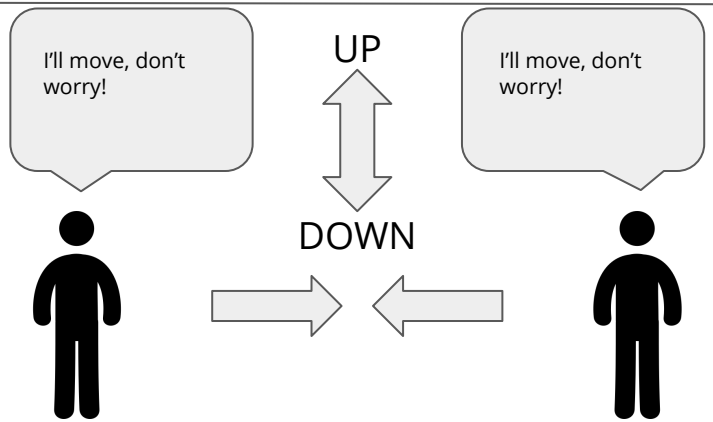
STARVATION



This works too...

LIVELOCK

"The corridor problem"



GUARDED BLOCKS

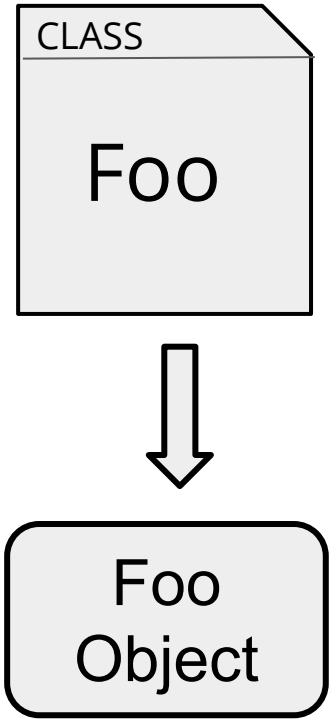
GUARDED BLOCKS

```
public void guardedJoy() {
        while (!joy) {}
        System.out.println("Joy achieved!");
}
```

```
public synchronized void guardedJoy() {
        while (!joy) {
                try {
                        wait();
                } catch (InterruptedException e){}
        }
        System.out.println("Joy and efficiency
        achieved!");
}
```

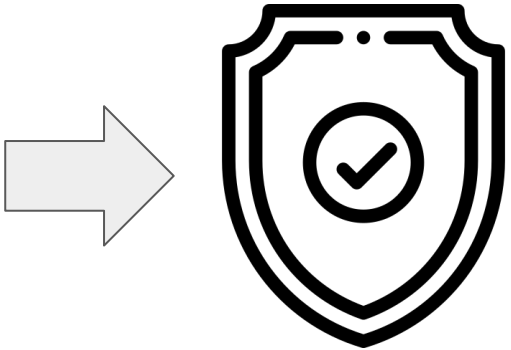
IMMUTABLE OBJECTS

IMMUTABLE OBJECTS



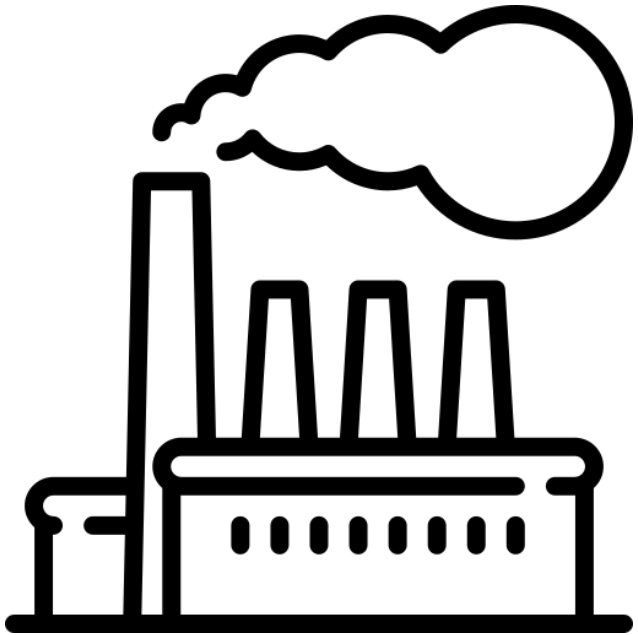
- No setter methods.
- Methods can’t modify members.
- All fields **final** and **private.**
- Override disabled for subclasses.

- Ensure **no shared references** to mutable objects in constructor.



IMMUTABLE OBJECTS

Factories

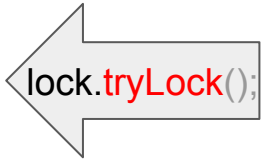


Are Cool

HIGH LEVEL CONCURRENCY OBJECTS

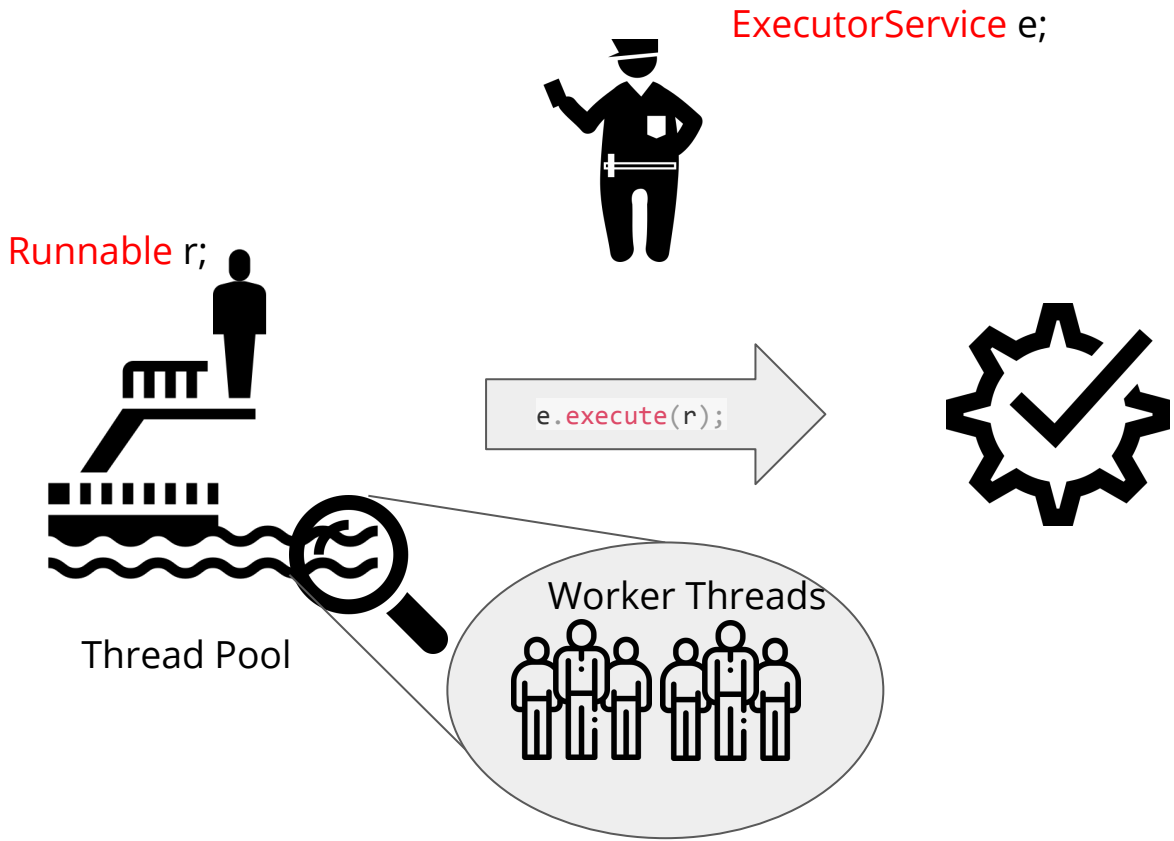
LOCK OBJECTS

Have a Lock™ ?



TRY IT!!

EXECUTORS


ExecutorService e;
Runnable r;
e.execute(r);
Worker Threads
Thread Pool

BIBLIOGRAPHY

- An introduction to parallel programming - Peter Pacheco
- https://docs.oracle.com/javase/tutorial/essential/concurrency/
- Learning Concurrency in Python - Elliot Forbes
- Icons in presentation made by https://www.freepik.com/ from https://www.flaticon.com/ is licensed by http://creativecommons.org/licenses/by/3.0/ CC 3.0 BY.