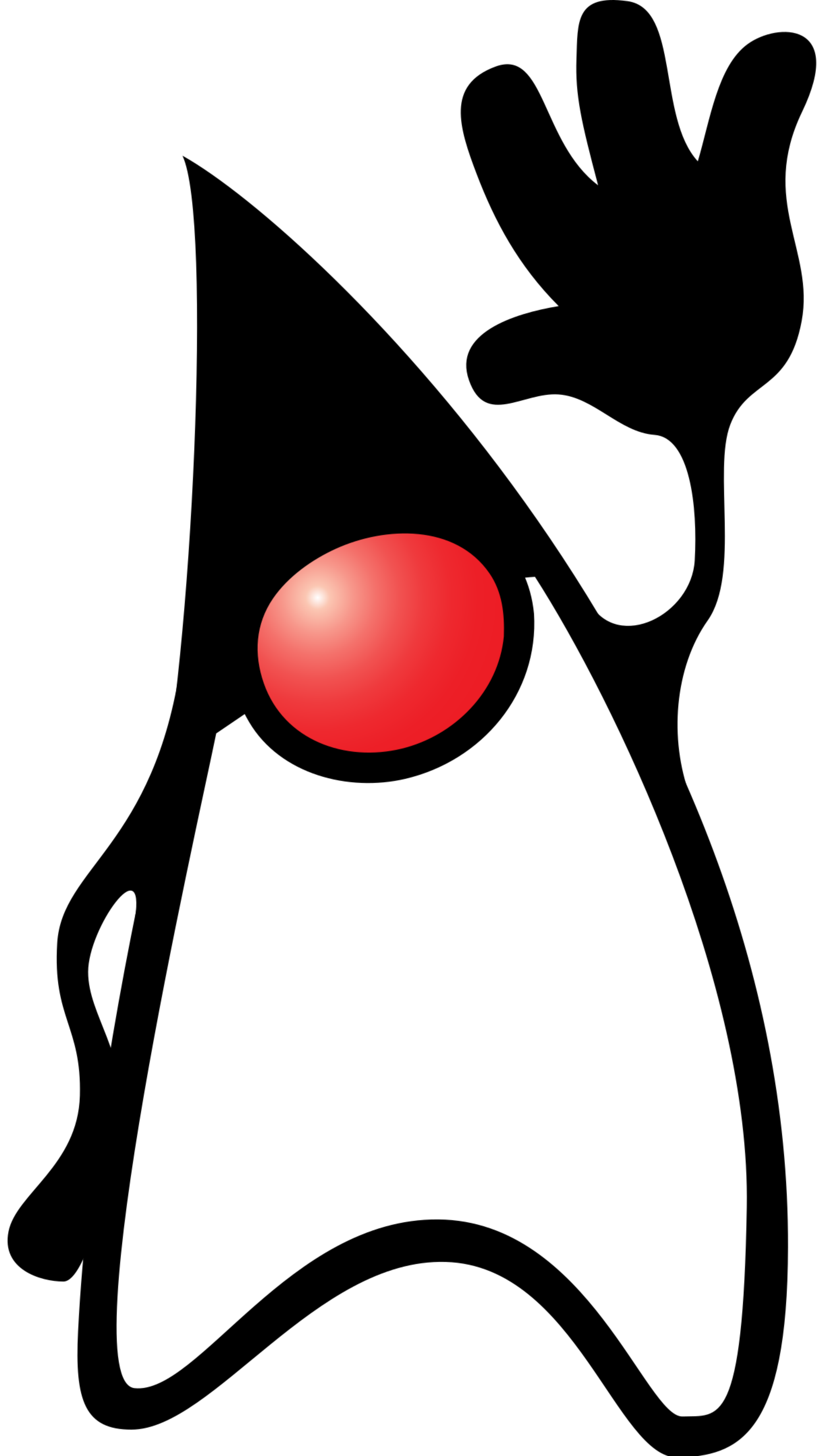
MR SADAM HUSSAIN

EXTENDED JAVA

Multithreading: Overview

ADAPTED FROM DR EDWARD ANSTEAD



THIS WEEK’S LECTURE

Overview of multitasking and threading

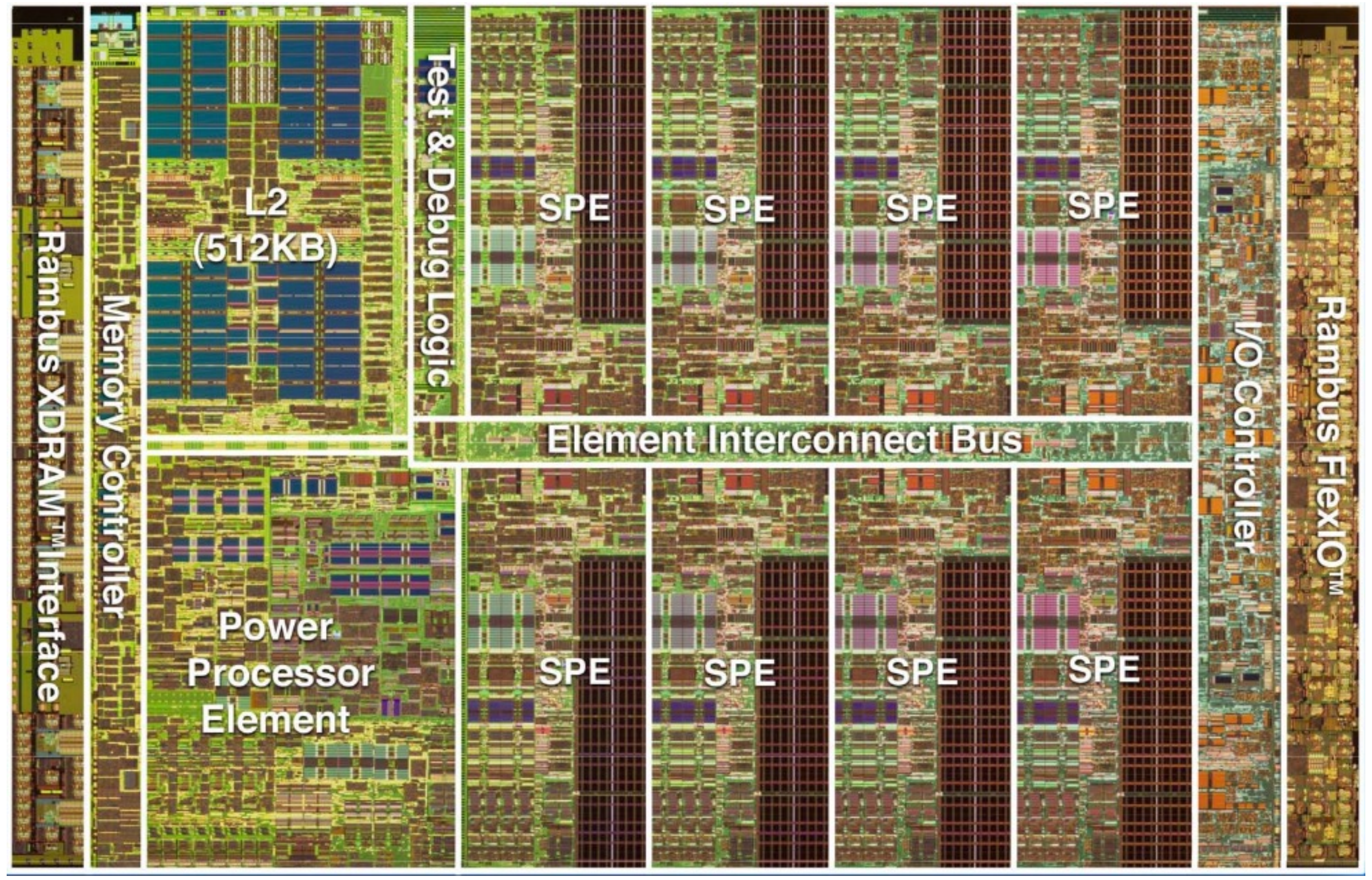
Java Thread class and Runnable interface

Thread synchronisation

Inter-thread communication

MULTITASKING

- Modern operating systems multitask supporting,
  - multiple simultaneous users
  - multiple simultaneous processes (programs)
- This is achieved by,
  - time slicing
  - multicore processors (21st Century)
    - i.e. processes on Activity monitor



MULTITHREADING

- A thread is the smallest unit of dispatchable code,
  - A process is divided into multiple threads
  - Makes use of idle CPU time, for better performance and experience
  - Makes use of multiple processor cores
- For example
  - You can write email while your mailbox is synchronising with the server
  - Games load new assets while detecting user input
  - ???

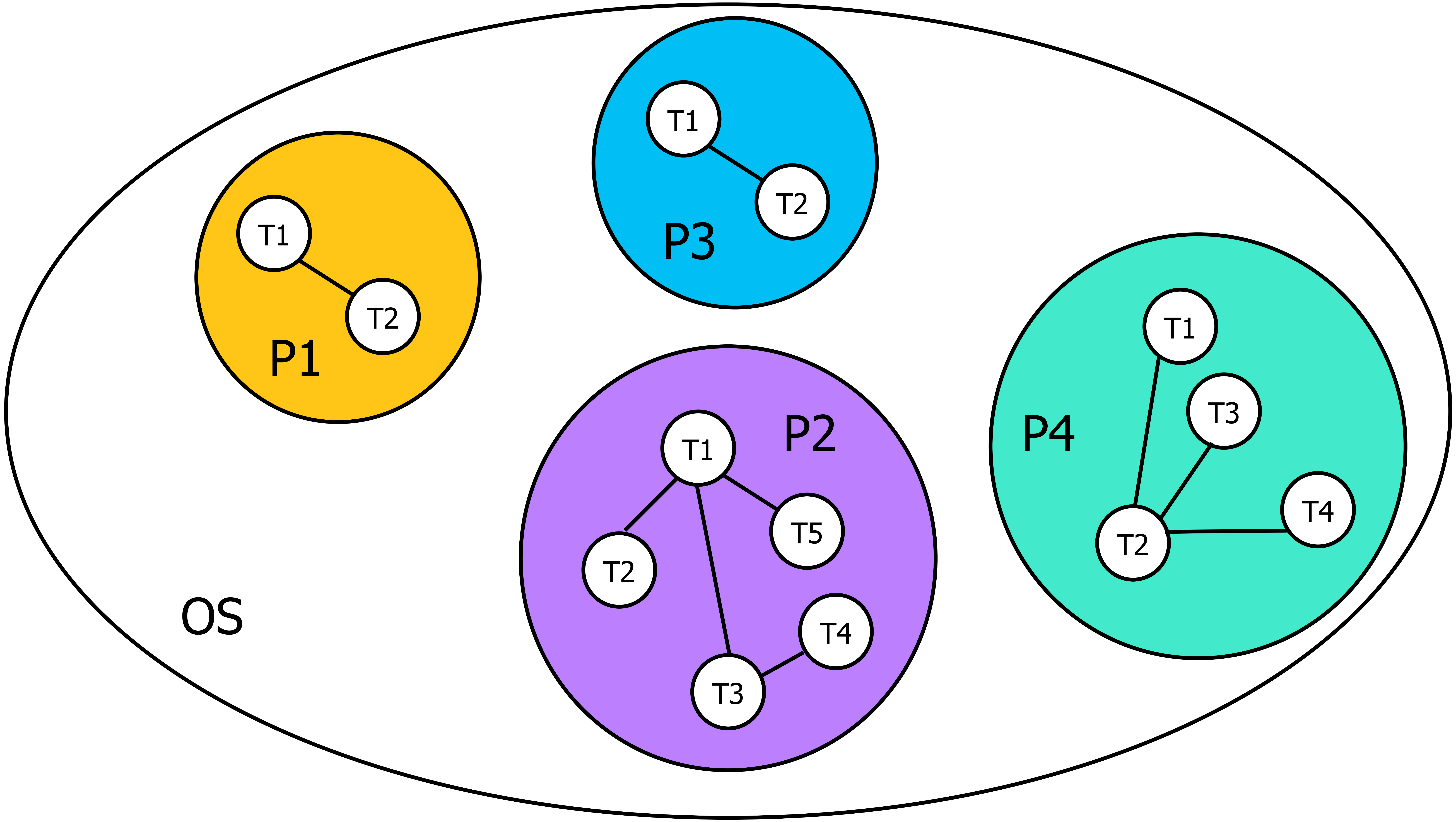
MULTITASKING

- Process based (switching user to user)
- Individual address space
- Processes are heavyweight
- interprocess communication is slow and complex (sockets, message bus, COM+, NSProxy)
- Process switching is slow (mapping memory, loading registers etc.)
- Secure (web browsers, i.e. Safari and google doc https://blog.chromium.org/2008/09/multi-process-architecture.html

MULTITHREADING

- Thread based (divide single program)
- shared address space (threads), individual address space for tasks
- threads are lightweight
- Interthread communication is straightforward

PROCESSES CONTAIN THREADS

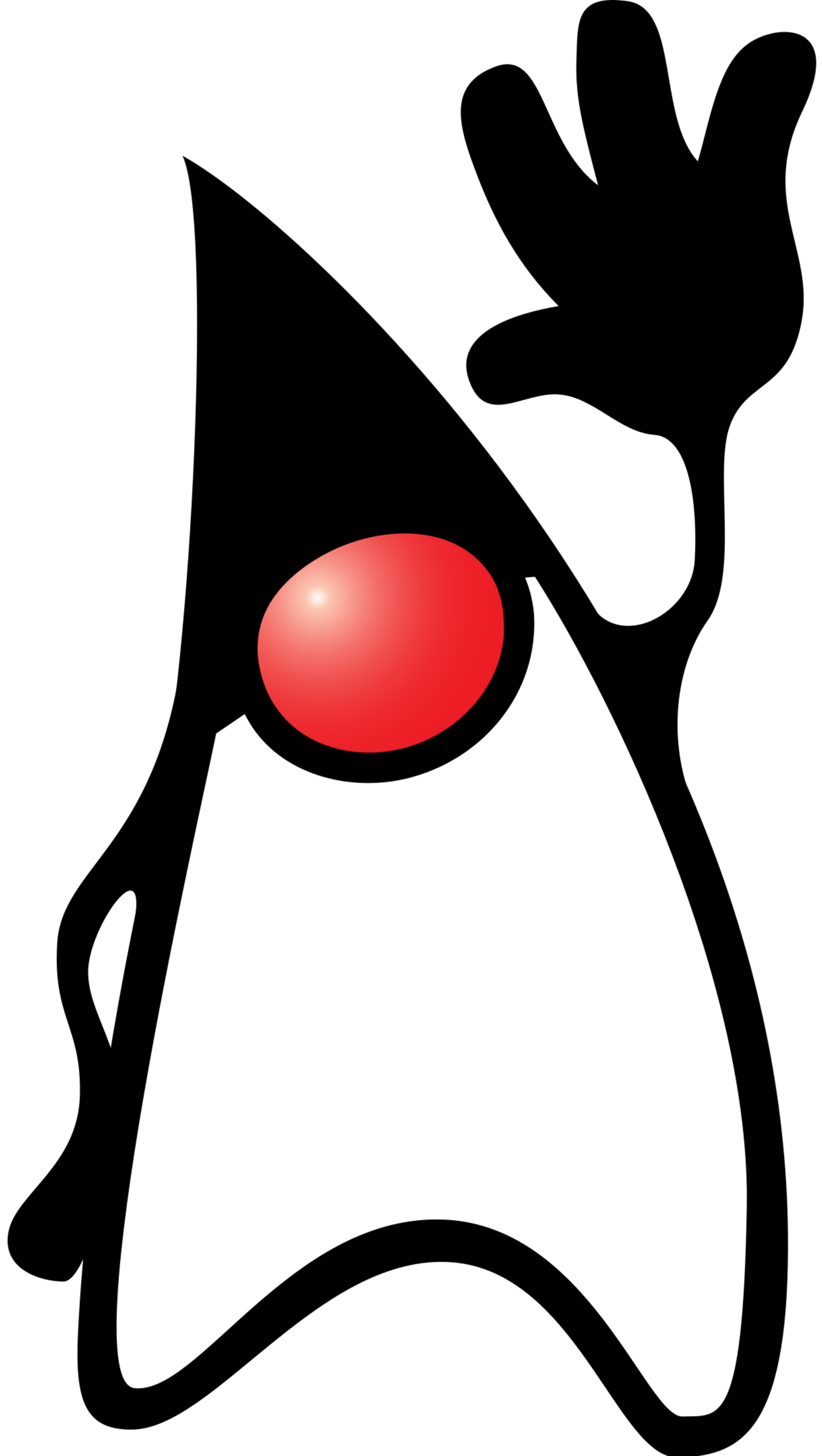


MR SADAM HUSSAIN

EXTENDED JAVA

Multithreading:
Thread Class and Runnable Interface

ADAPTED FROM DR EDWARD ANSTEAD



EXAMPLE: SINGLE THREADED PROGRAM

RUNNABLE INTERFACE

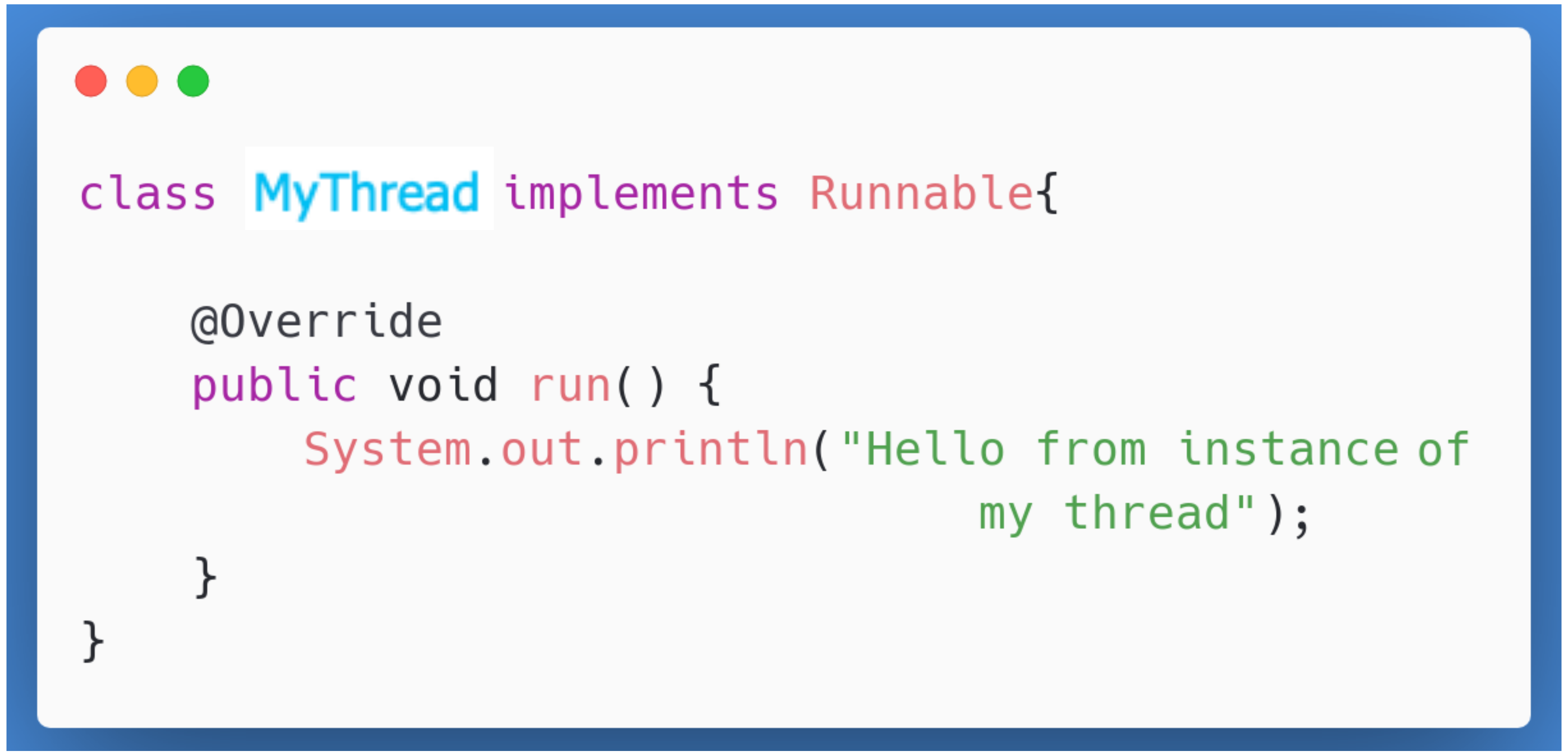
Java includes support for writing multithreaded programs. There are a couple of approaches for implementing multithreaded code, the first is the interface **`Runnable`**

- Classes that implement runnable need to implement the run method.
- Runnable is a functional interface, so we could use a Lambda expression.

RUNNABLE INTERFACE

Java includes support for writing multithreaded programs. There are a couple of approaches for implementing multithreaded code, the first is the interface

- Classes that implement runnable need to implement the run method. MyThread
- Runnable is a functional interface, so we could use a Lambda expression.



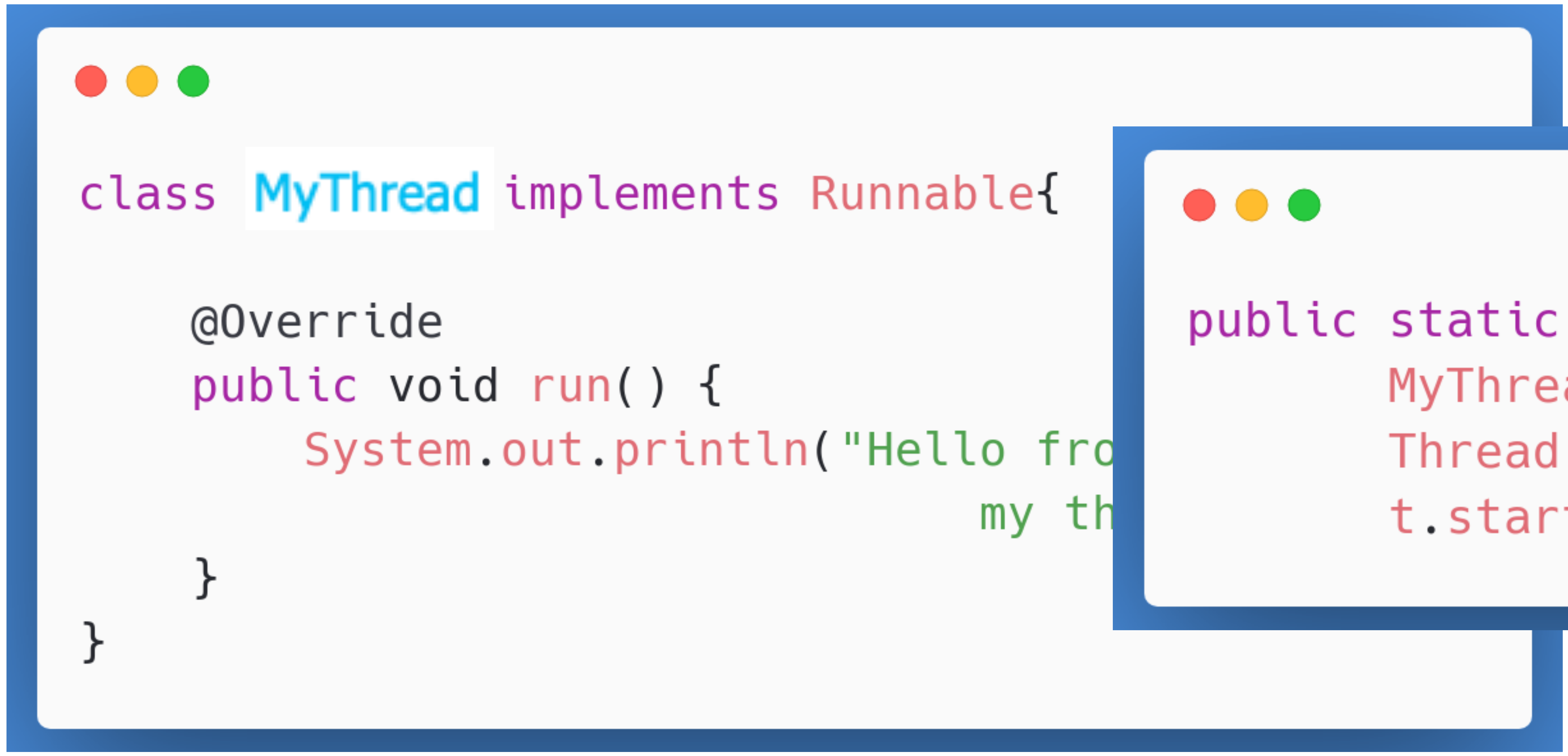
```
class MyThread implements Runnable{

    @Override
    public void run() {
        System.out.println("Hello from instance of
                            my thread");
    }
}
```

RUNNABLE INTERFACE

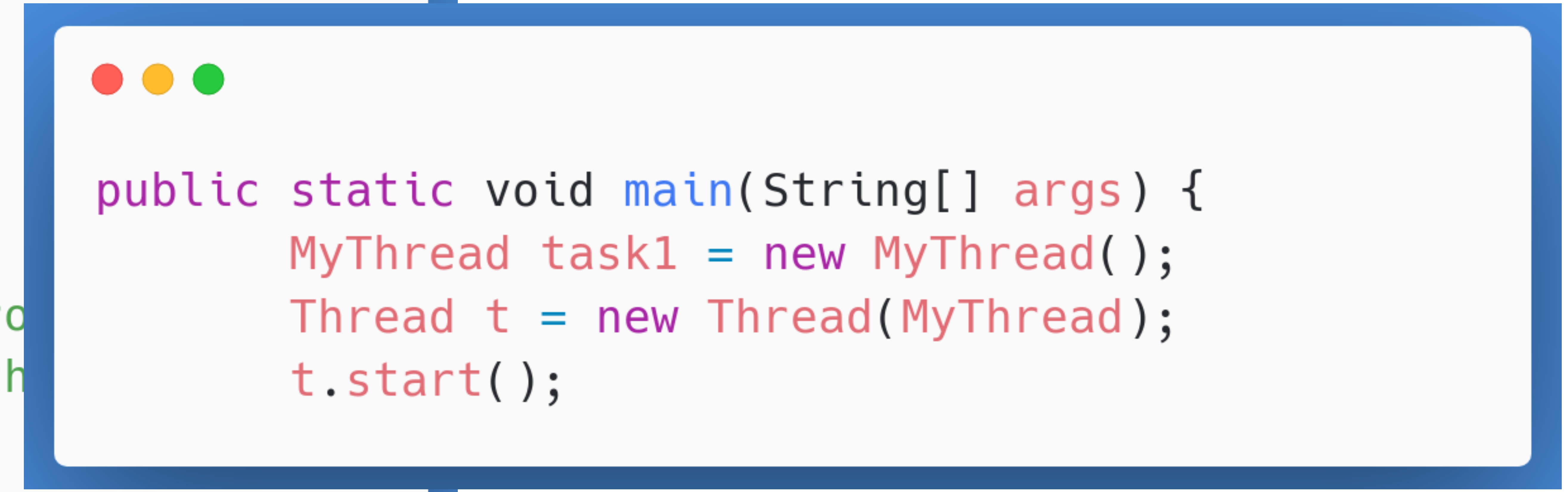
Java includes support for writing multithreaded programs. There are a couple of approaches for implementing multithreaded code, the first is the interface **`Runnable`**

- Classes that implement runnable need to implement the run method.
- Runnable is a functional interface, so we could use a Lambda expression.



```
class MyThread implements Runnable{

    @Override
    public void run() {
        System.out.println("Hello fro
                                  my th
    }
}
```

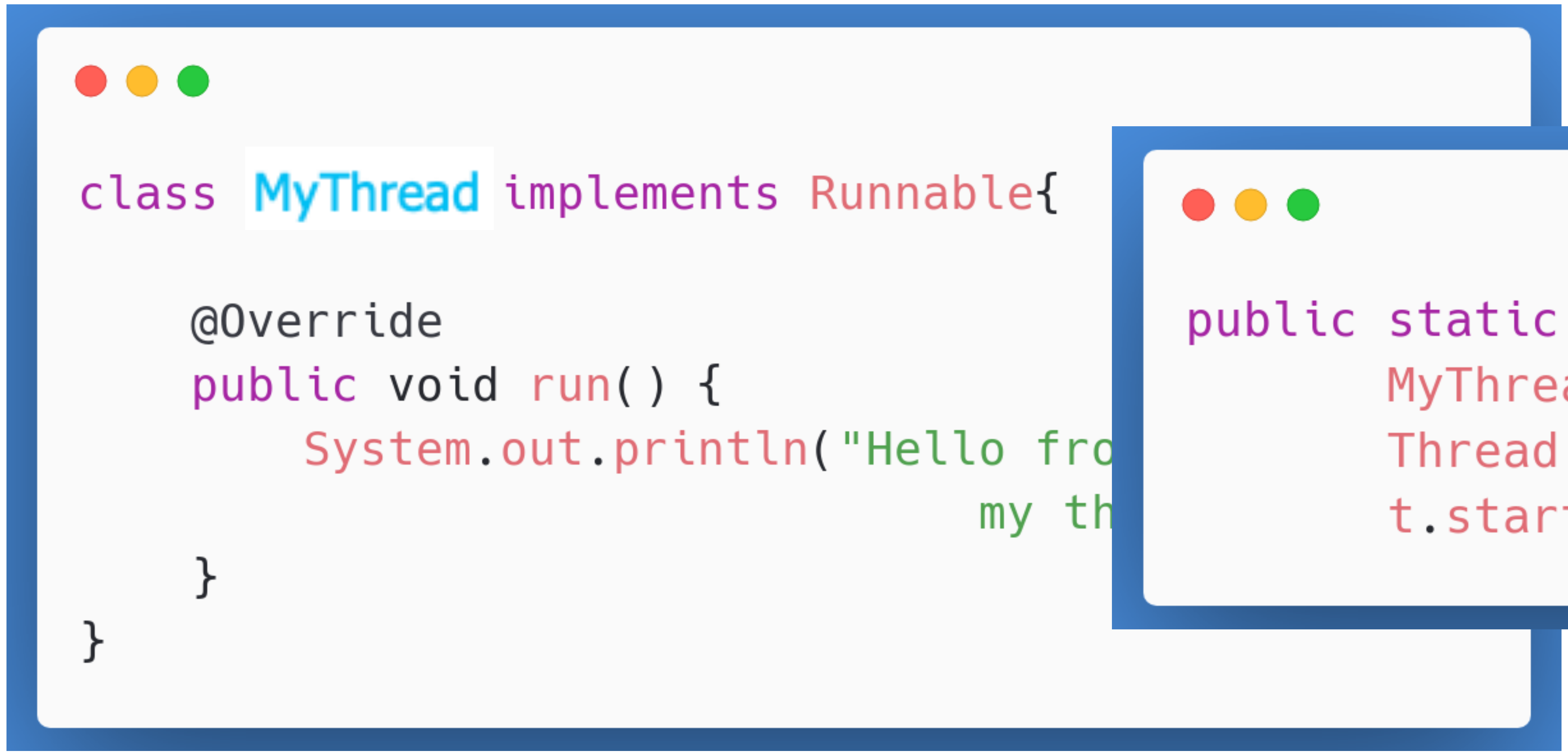


```
public static void main(String[] args) {
       MyThread task1 = new MyThread();
       Thread t = new Thread(MyThread);
       t.start();
```

RUNNABLE INTERFACE

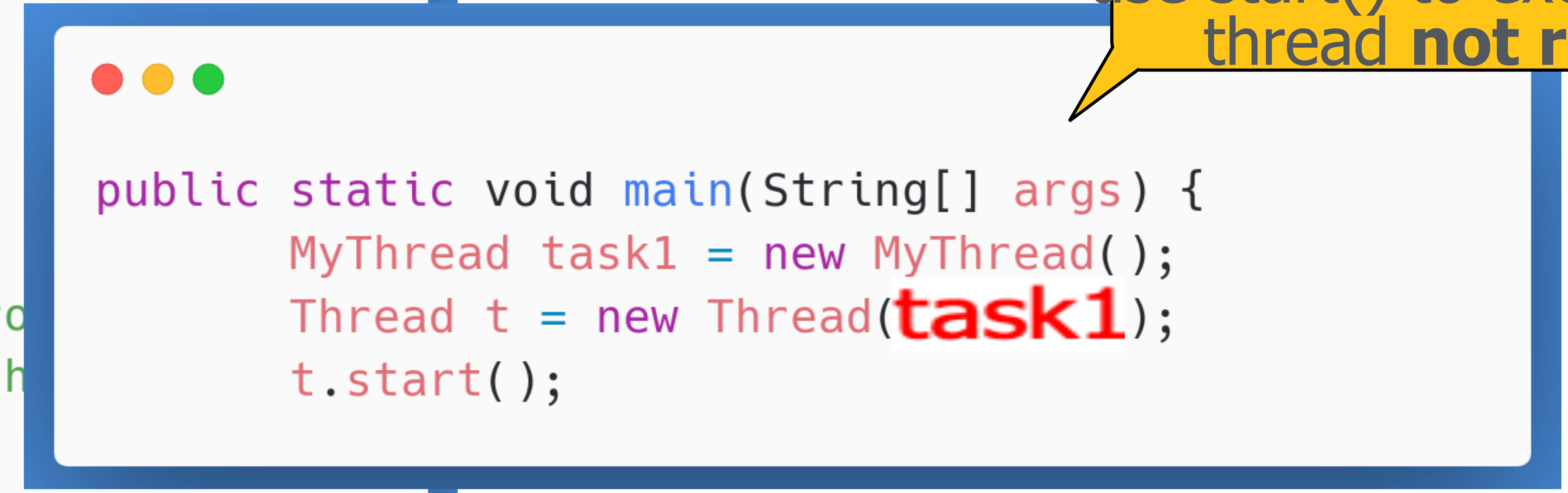
Java includes support for writing multithreaded programs. There are a couple of approaches for implementing multithreaded code, the first is the interface **`Runnable`**

- Classes that implement runnable need to implement the run method.
- Runnable is a functional interface, so we could use a Lambda expression.



```
class MyThread implements Runnable{

    @Override
    public void run() {
        System.out.println("Hello fro
                          my th
    }
}
```

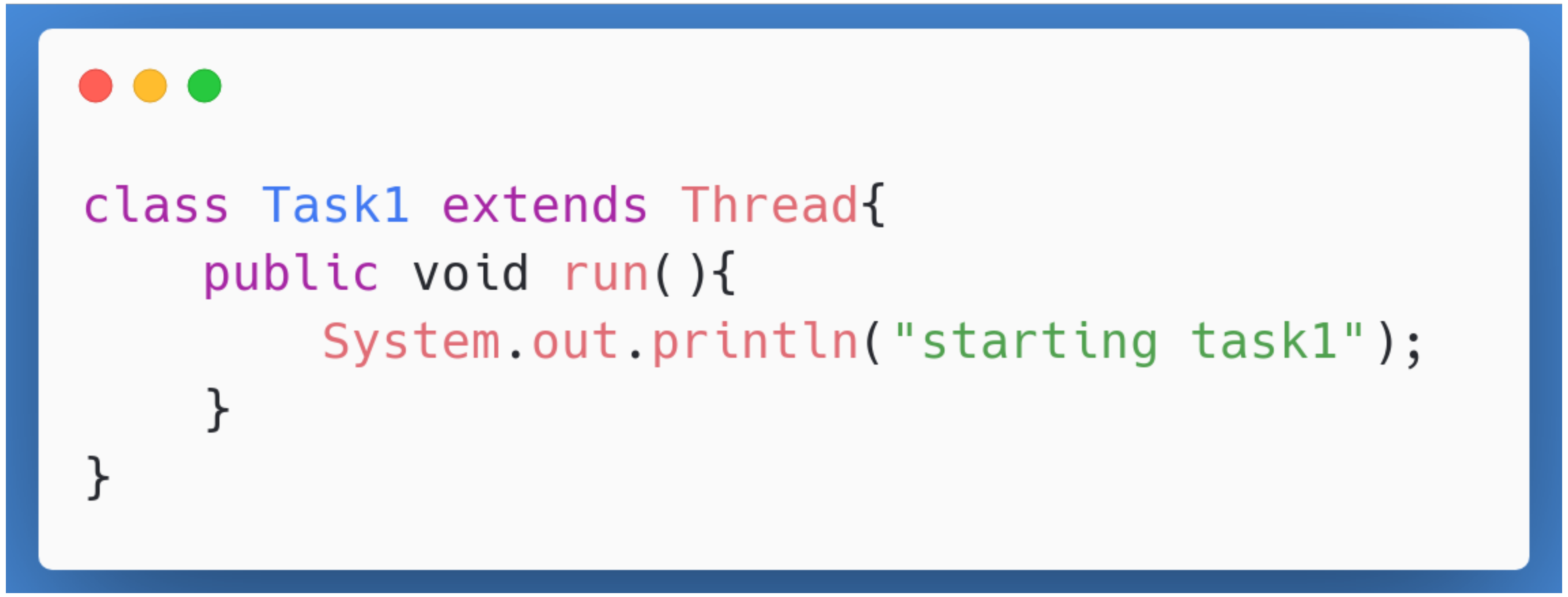


```
public static void main(String[] args) {
       MyThread task1 = new MyThread();
       Thread t = new Thread(task1);
       t.start();
```

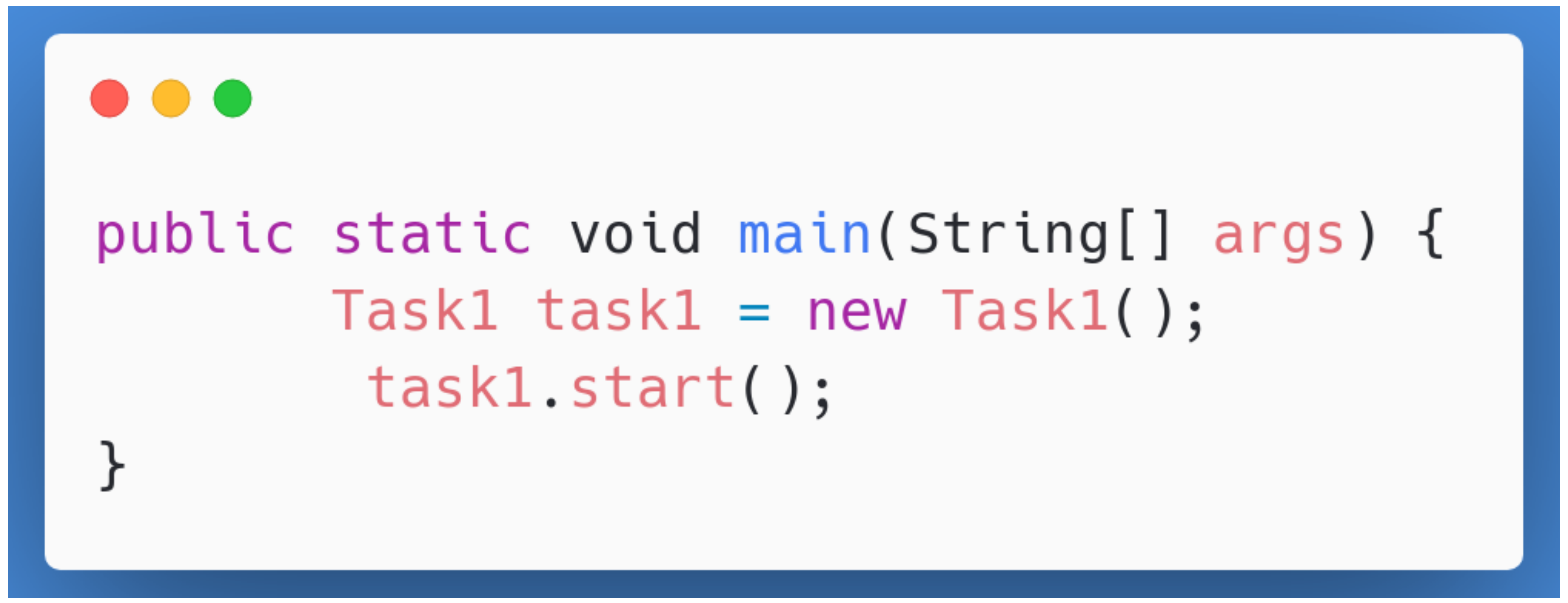
use start() to execute the thread **not run()**

EXTENDING THREAD

- Alternatively extend the thread class
- You would typically override the run method
  - Confusingly, Thread implements runnable too
- You only then need to create a new instance of your Thread



```java
class Task1 extends Thread{
    public void run(){
        System.out.println("starting task1");
    }
}
```



```java
public static void main(String[] args) {
        Task1 task1 = new Task1();
         task1.start();
}
```

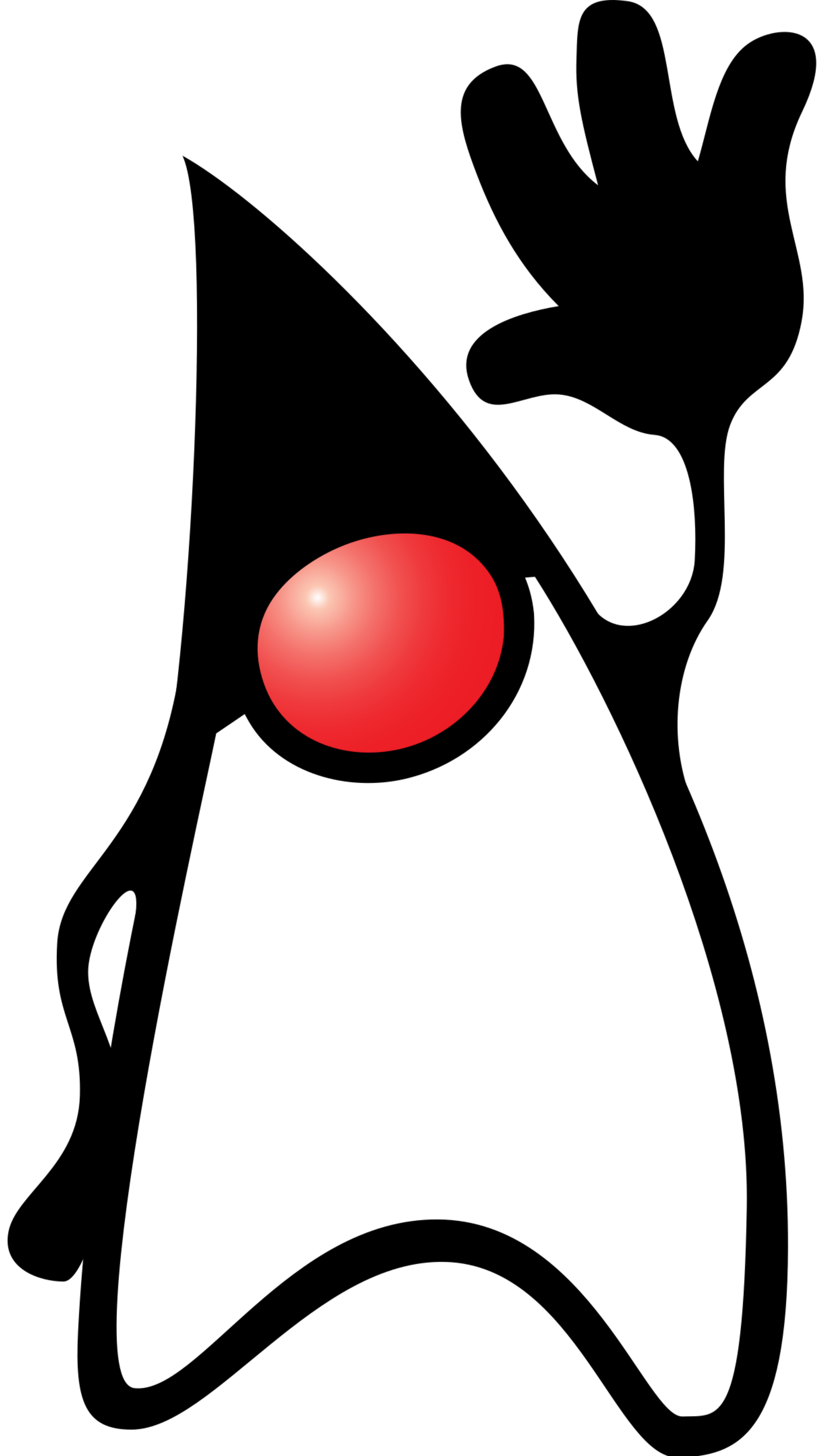
EXAMPLE: SIMPLE THREADING

MR SADAM HUSSAIN

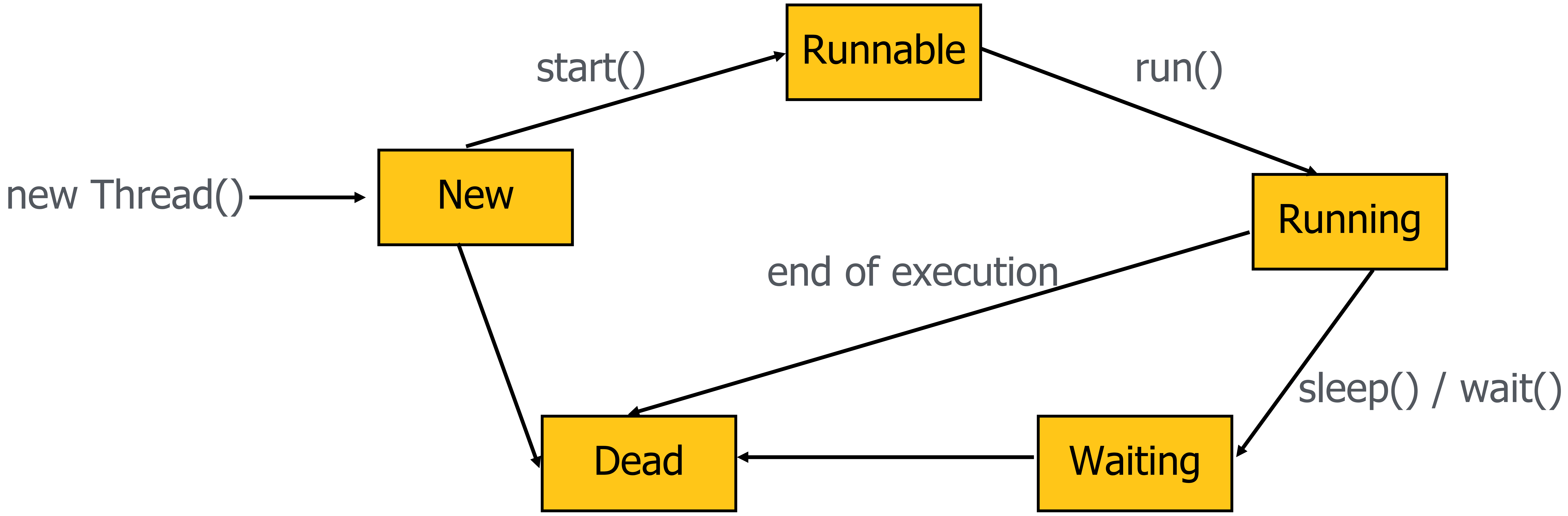
EXTENDED JAVA

Multithreading: Thread Lifecycle

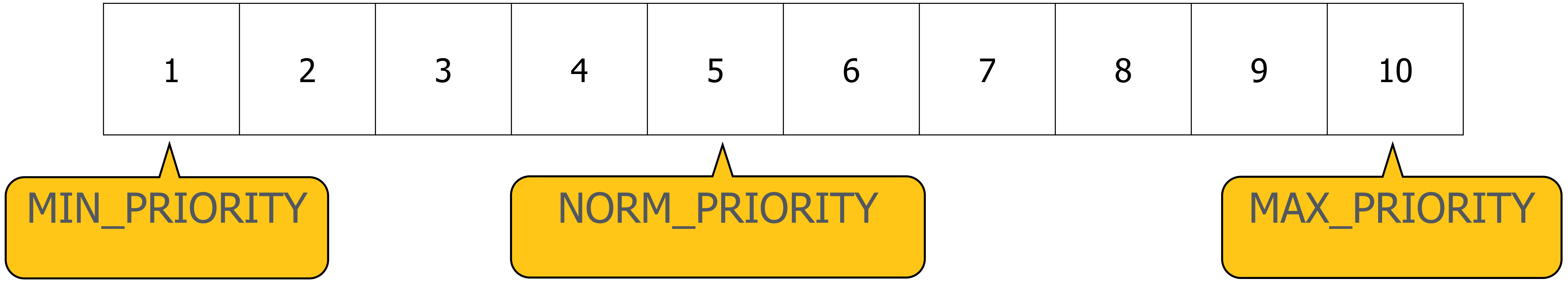
ADAPTED FROM DR EDWARD ANSTEAD



THREAD LIFECYCLE



THREAD PRIORITY

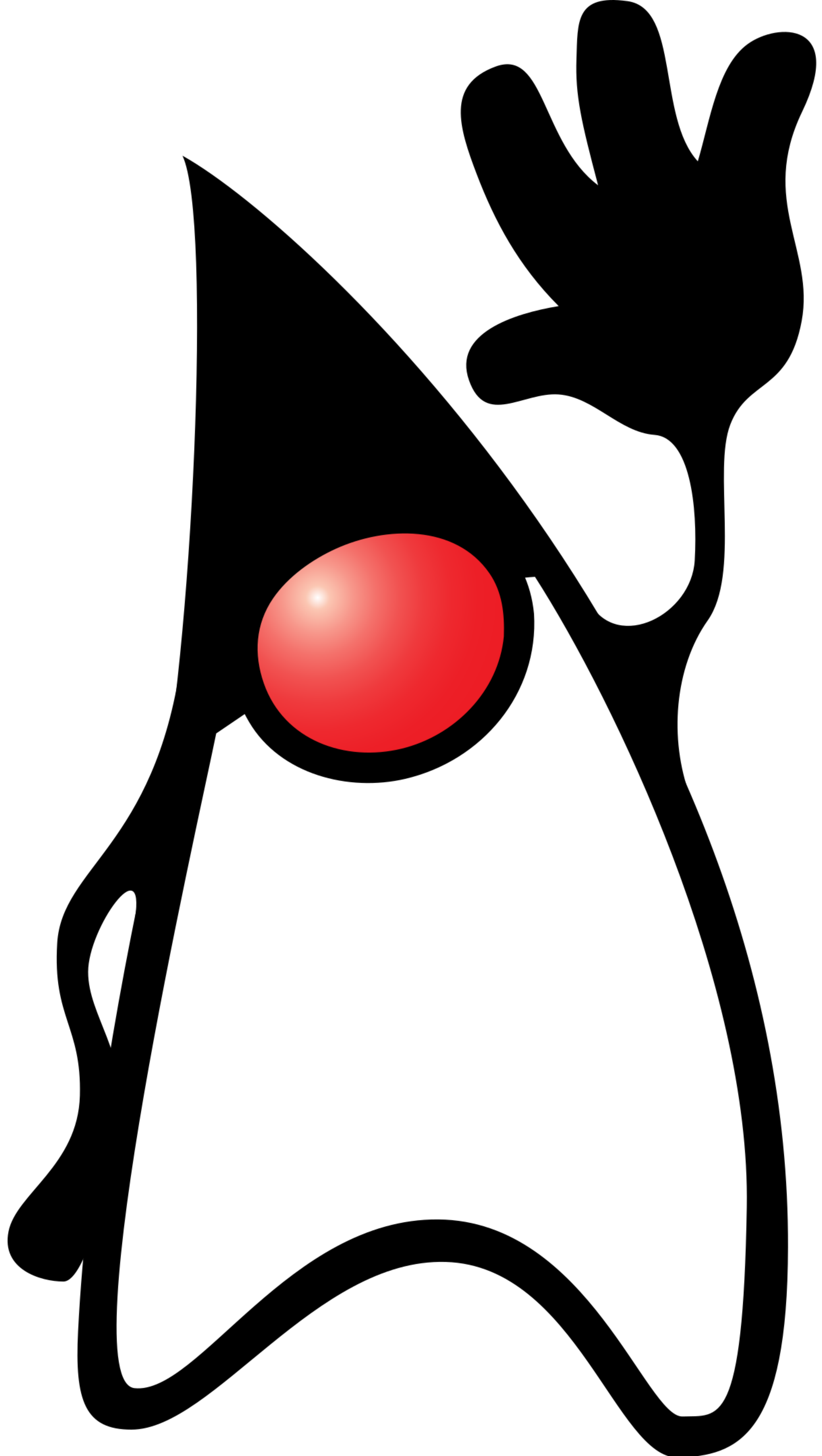


MR SADAM HUSSAIN

EXTENDED JAVA

Multithreading:
Dining Philosophers problem

ADAPTED FROM DR EDWARD ANSTEAD



THE DINING PHILOSOPHERS PROBLEM

EDSGER W. DIJKSTRA

THE DINING PHILOSOPHERS PROBLEM

EATING

THINKING

THE DINING PHILOSOPHERS PROBLEM

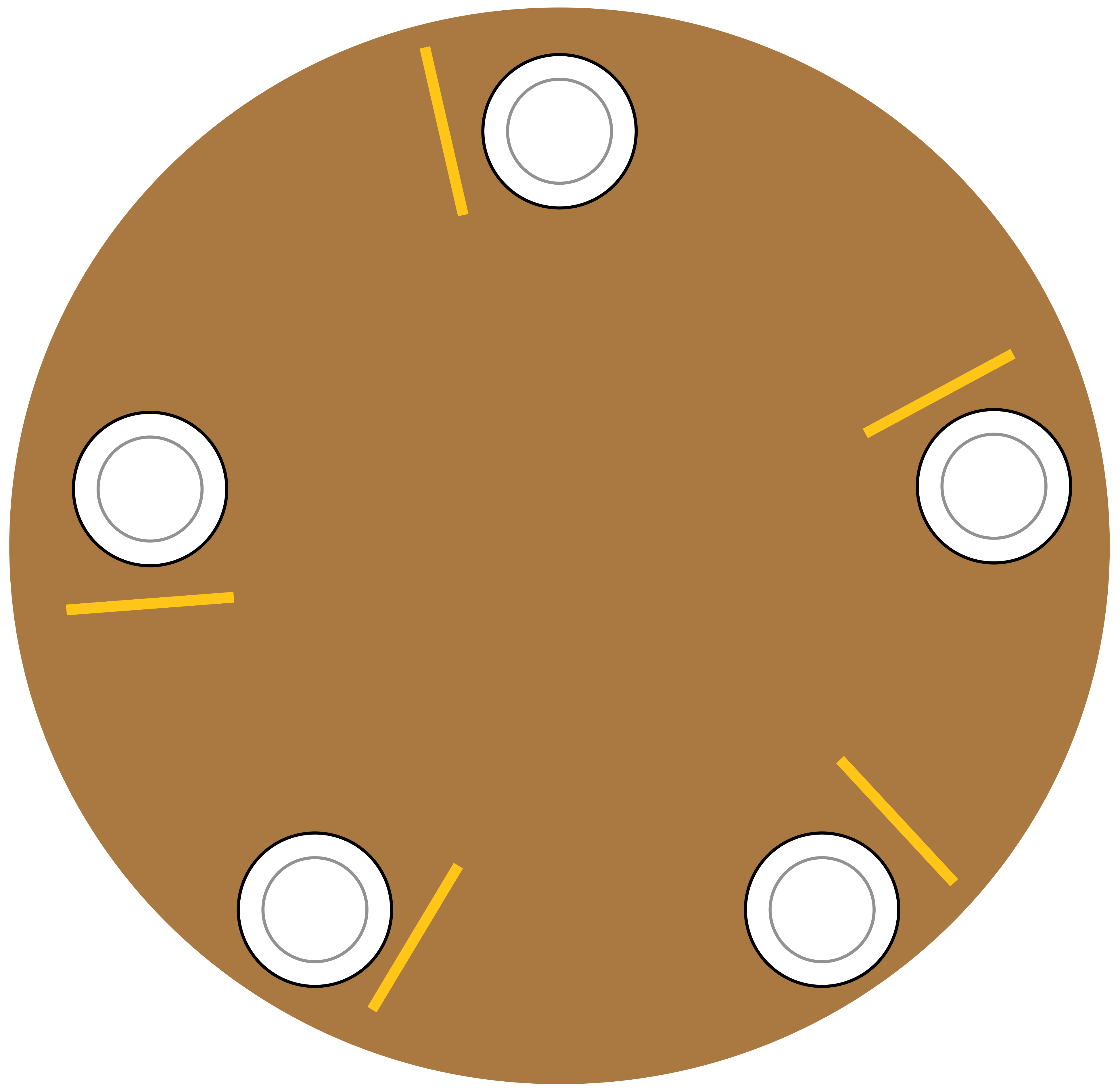
'NOM' CHOMSKY

EATING

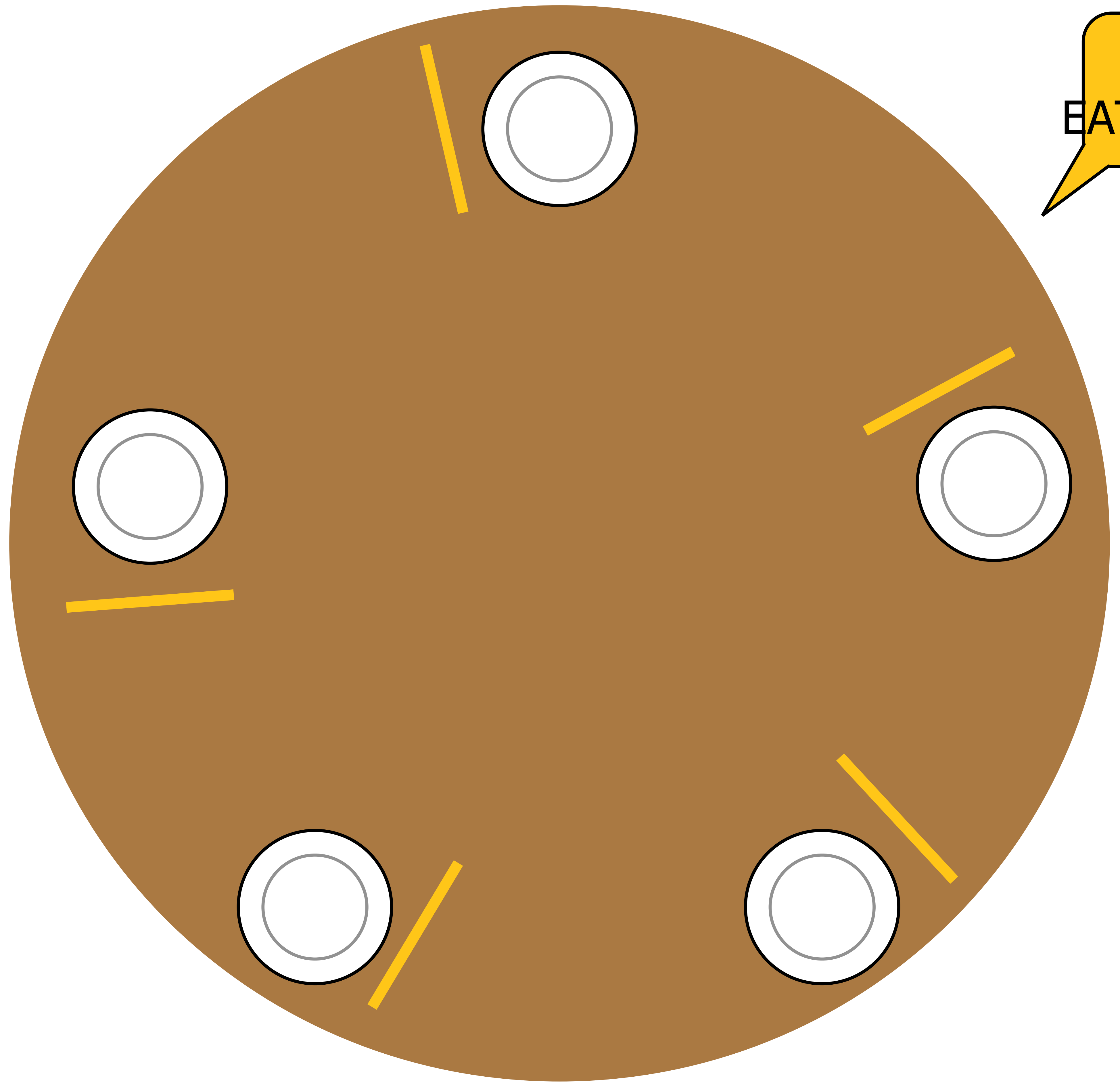
NOAM CHOMSKY

THINKING

THE DINING PHILOSOPHERS PROBLEM

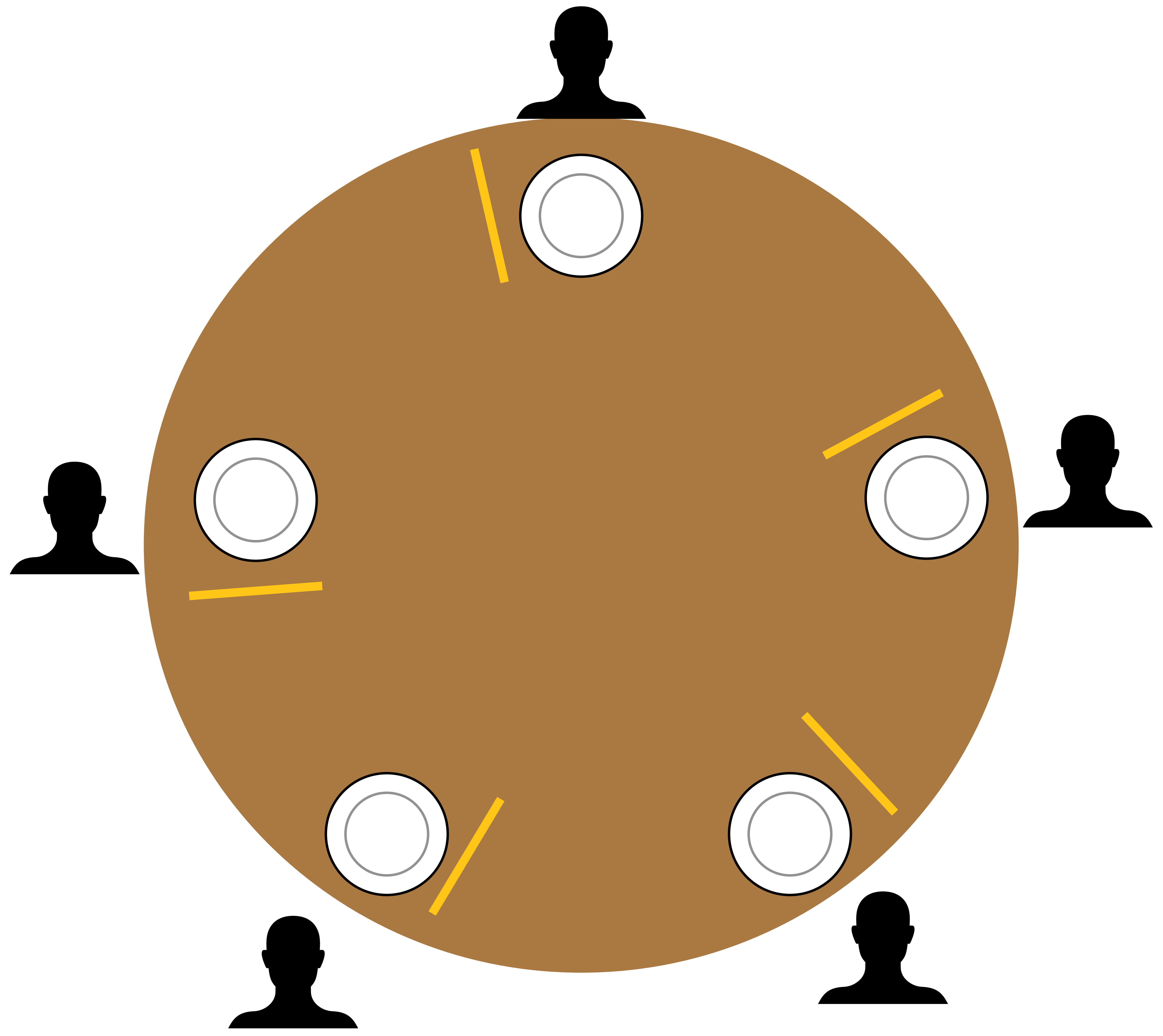


THE DINING PHILOSOPHERS PROBLEM

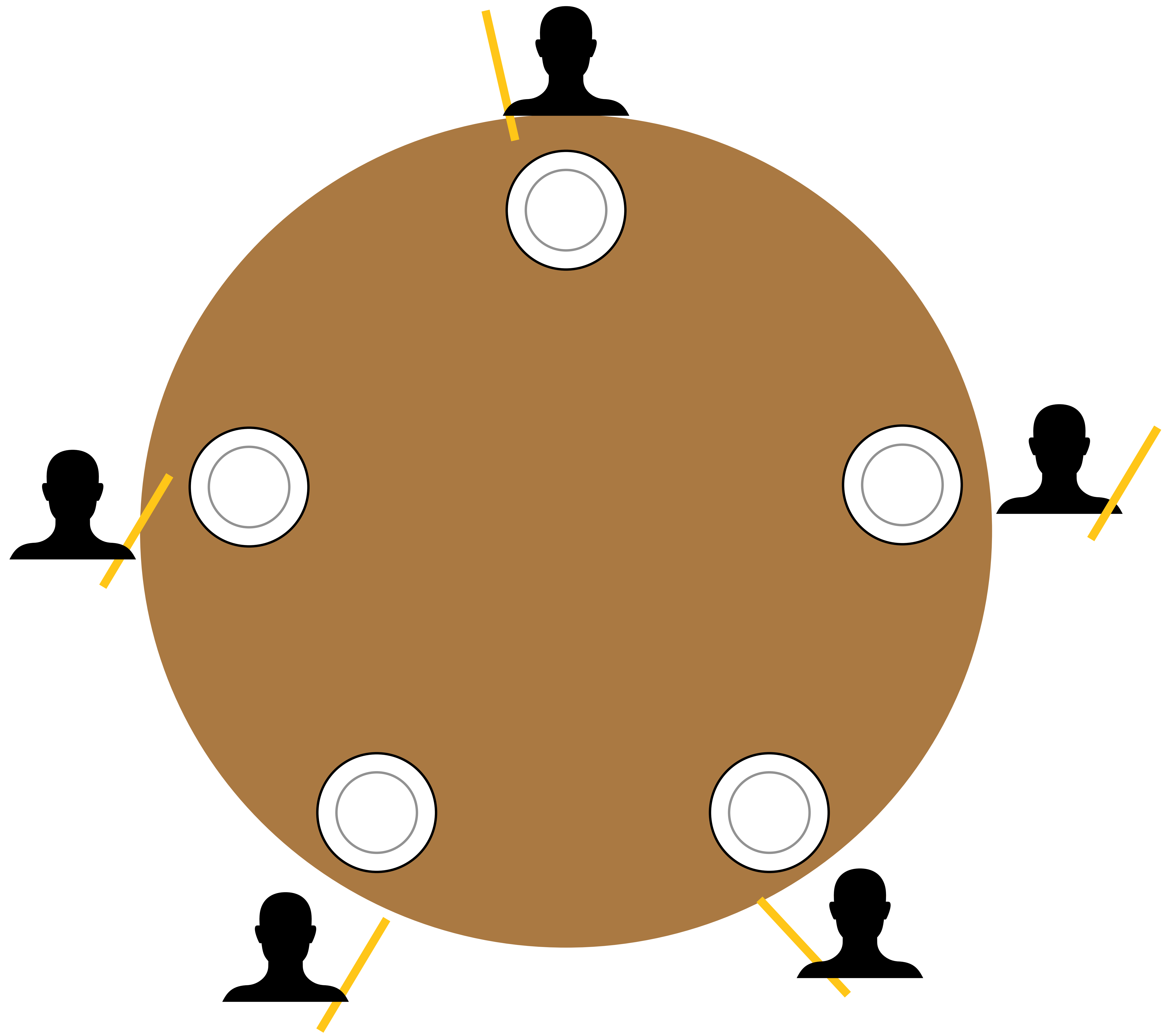


EATING REQUIRES 2 CHOPSTICKS

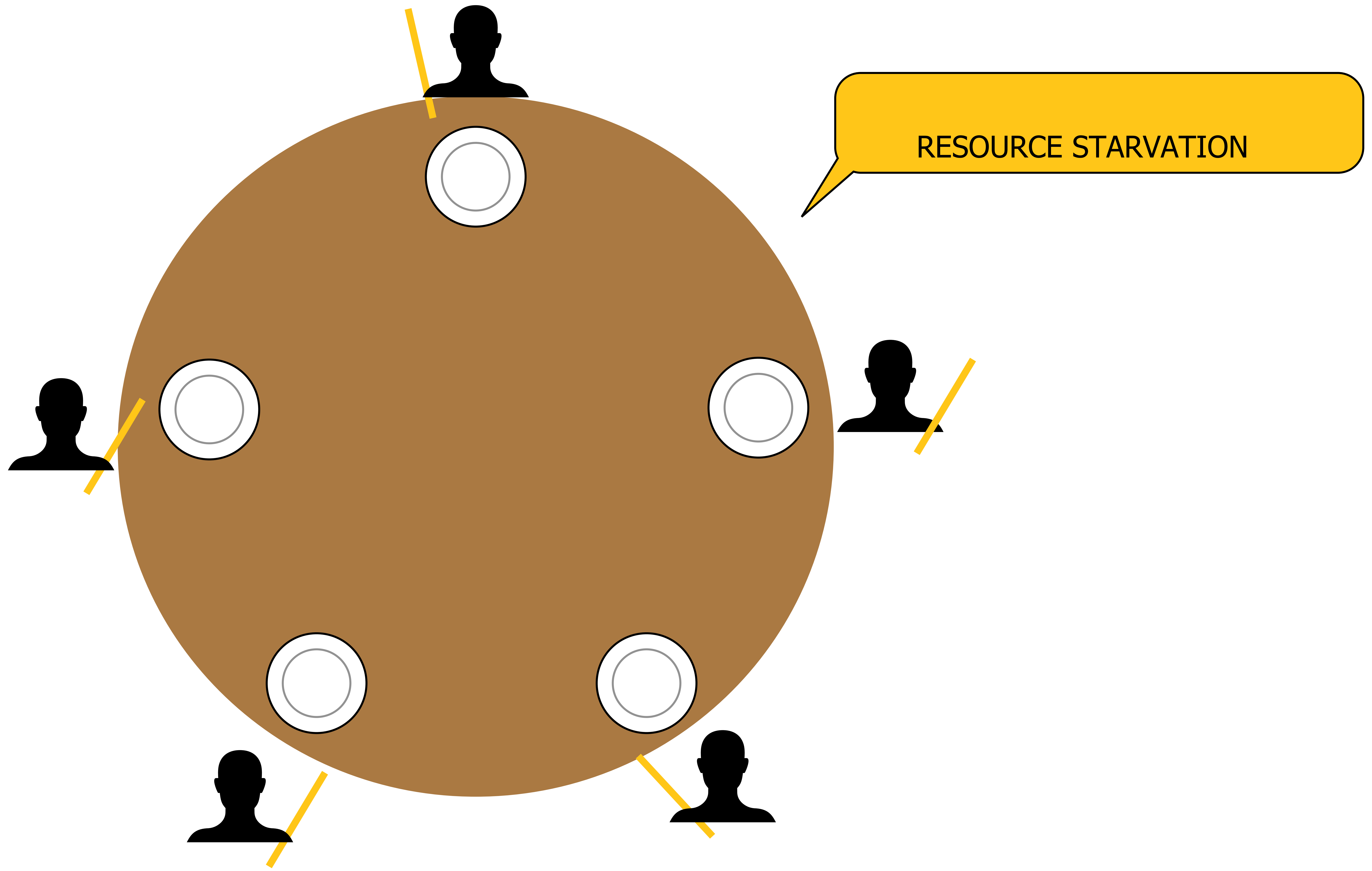
THE DINING PHILOSOPHERS PROBLEM



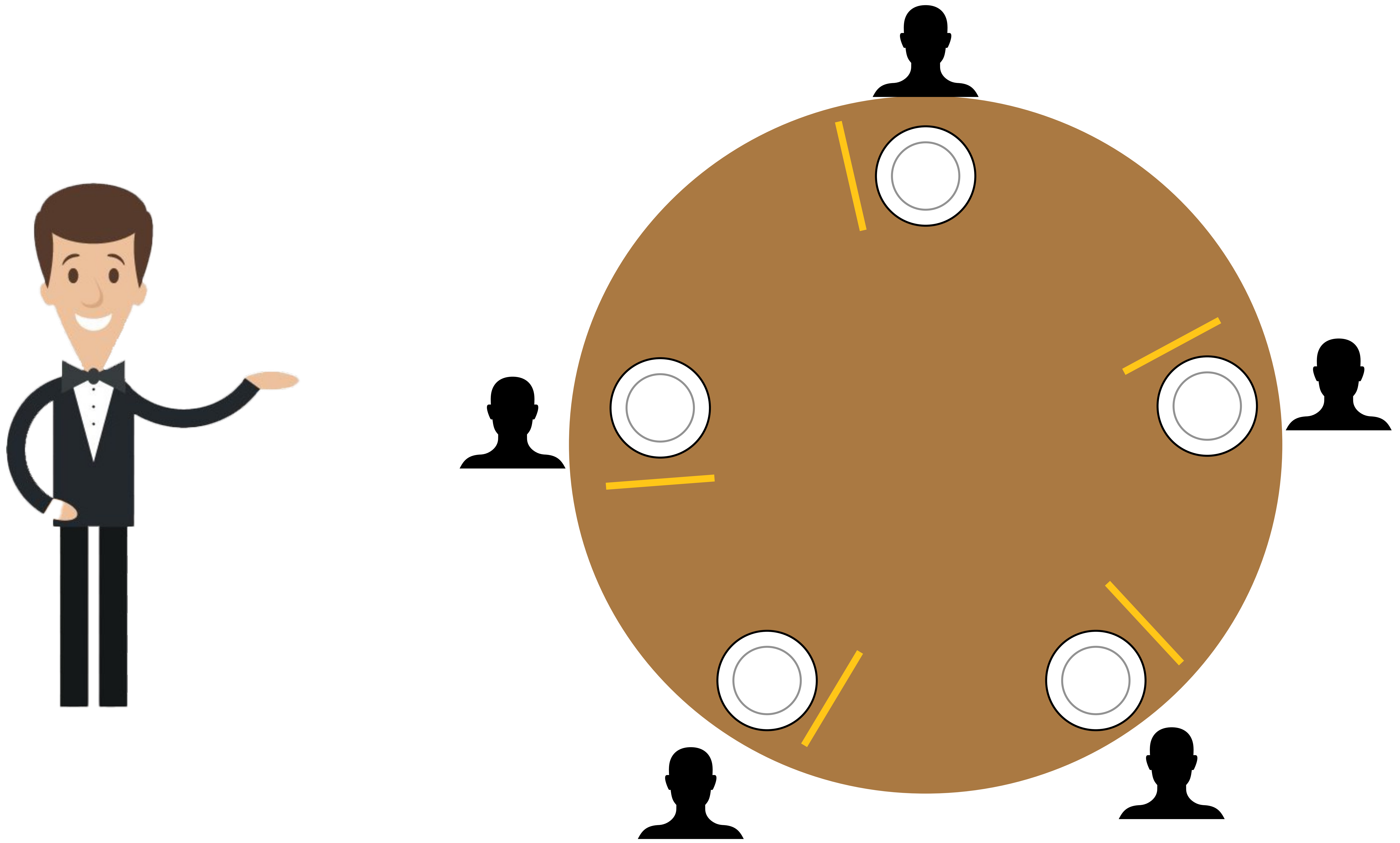
THE DINING PHILOSOPHERS PROBLEM



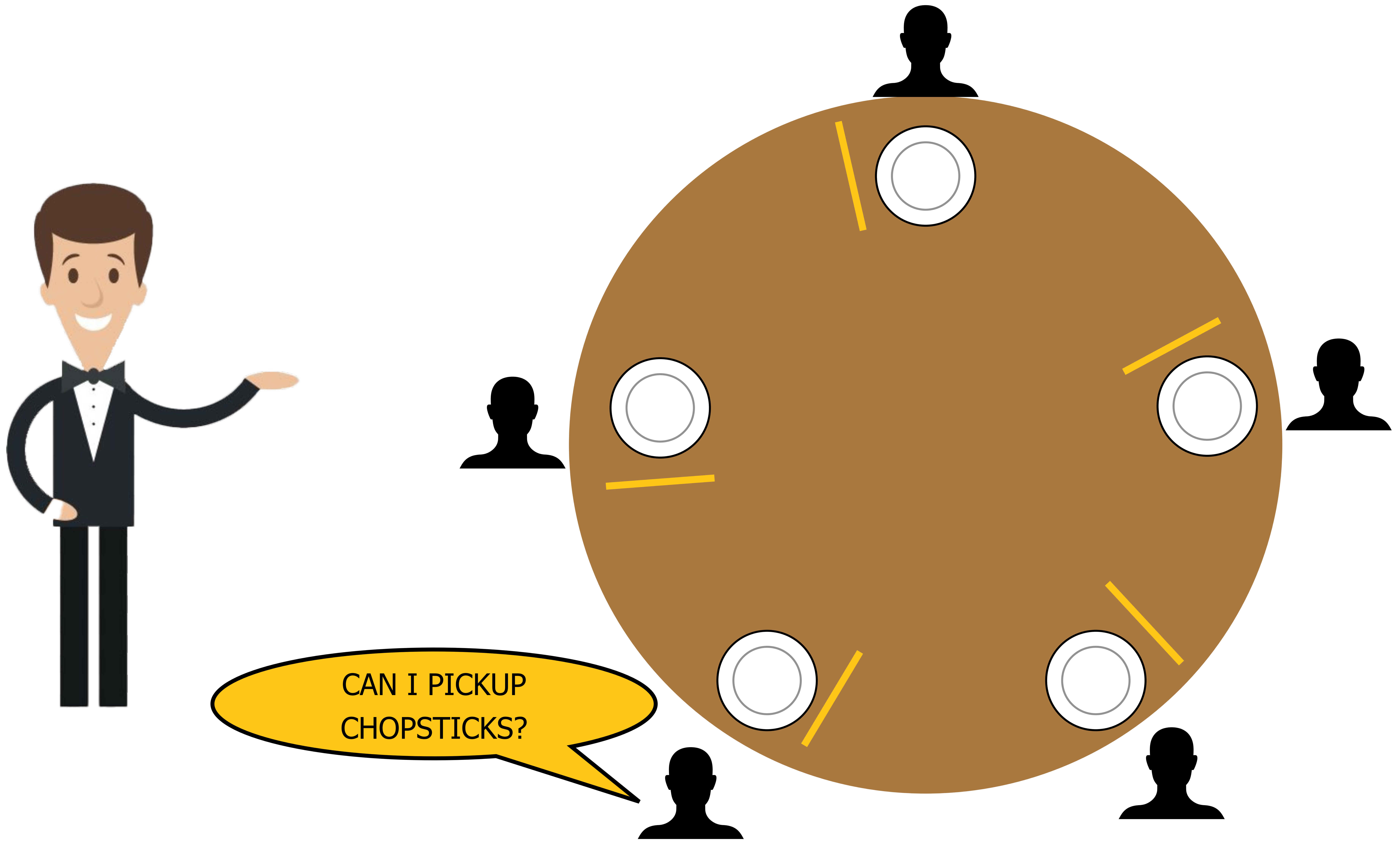
THE DINING PHILOSOPHERS PROBLEM



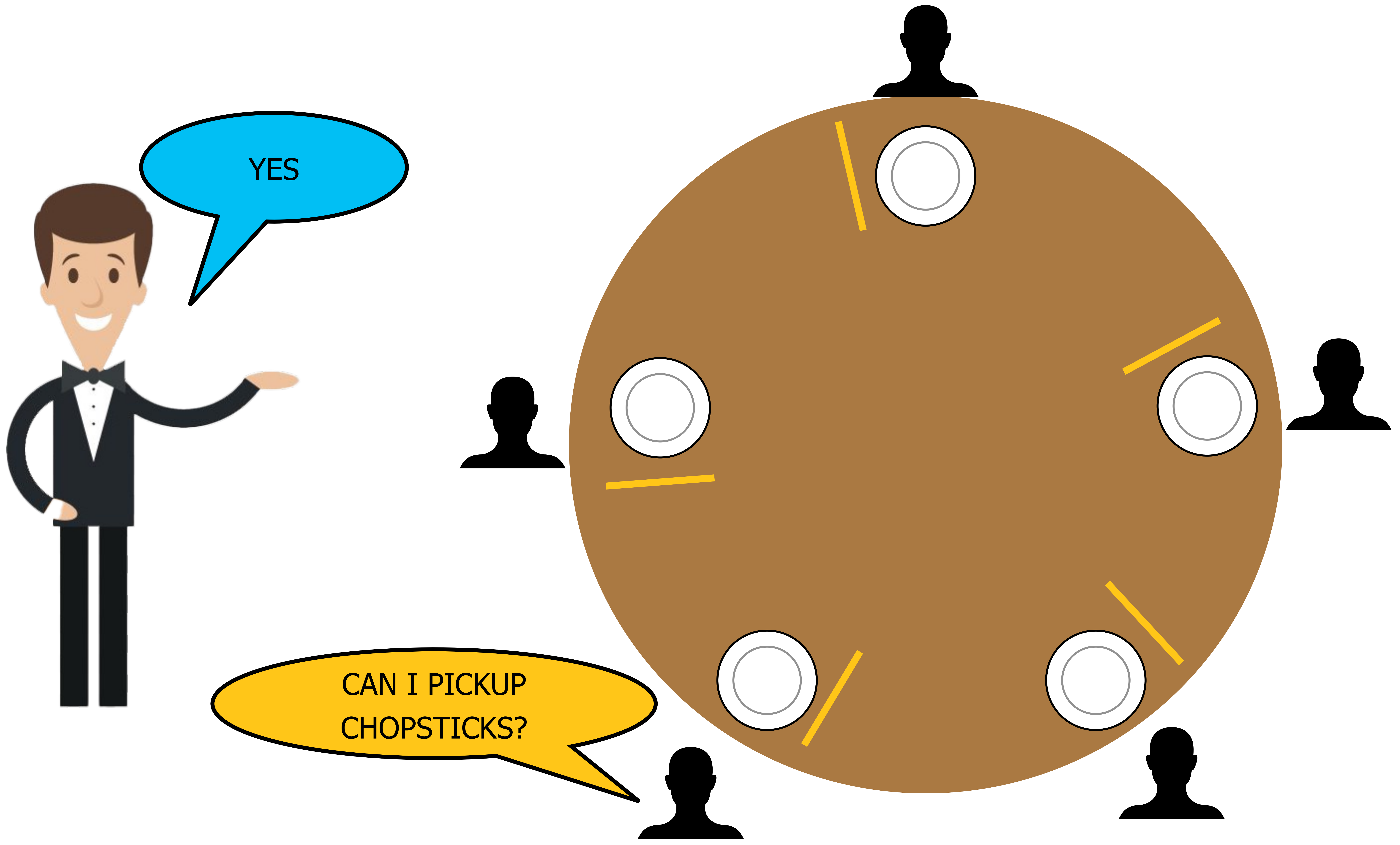
THE DINING PHILOSOPHERS PROBLEM



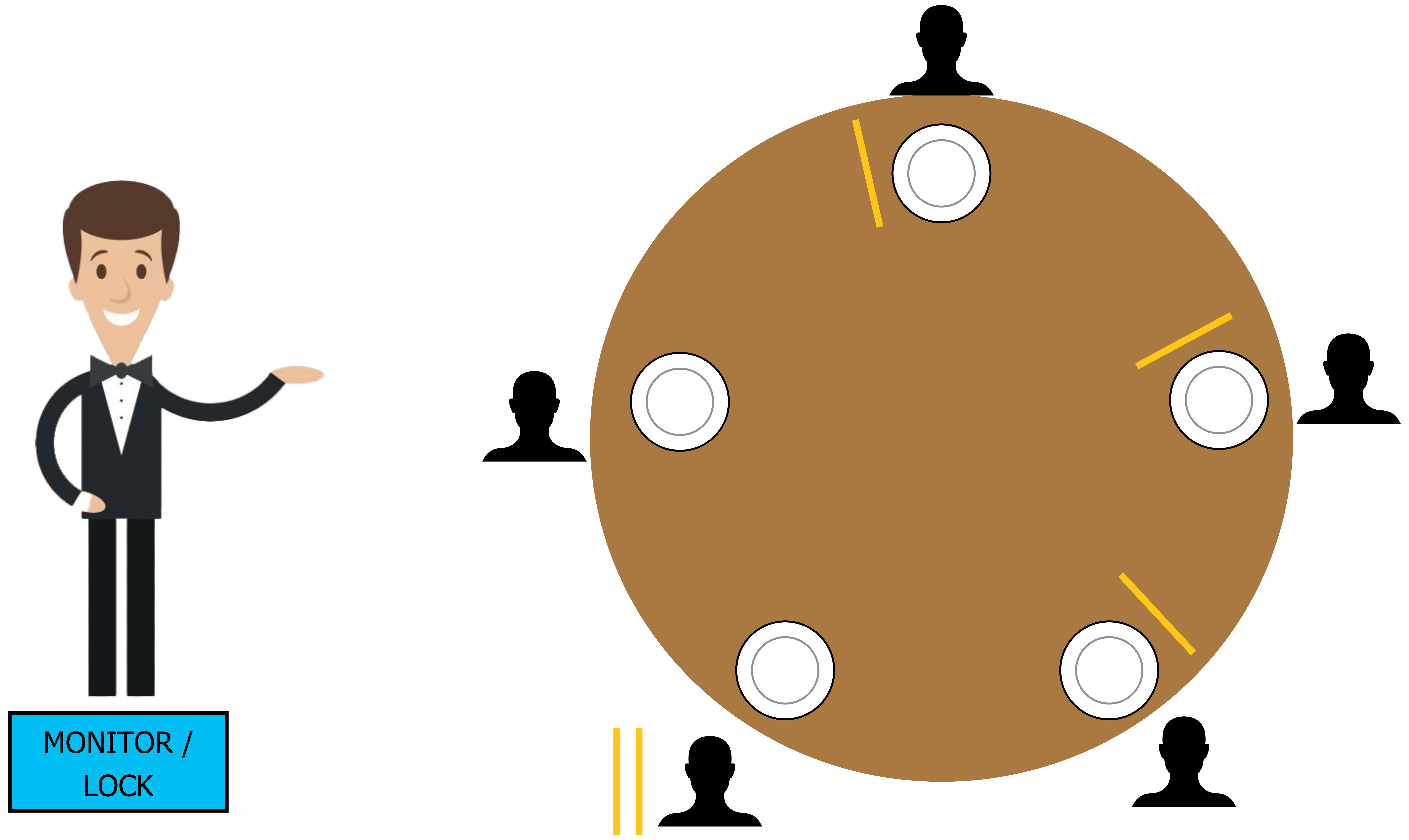
THE DINING PHILOSOPHERS PROBLEM



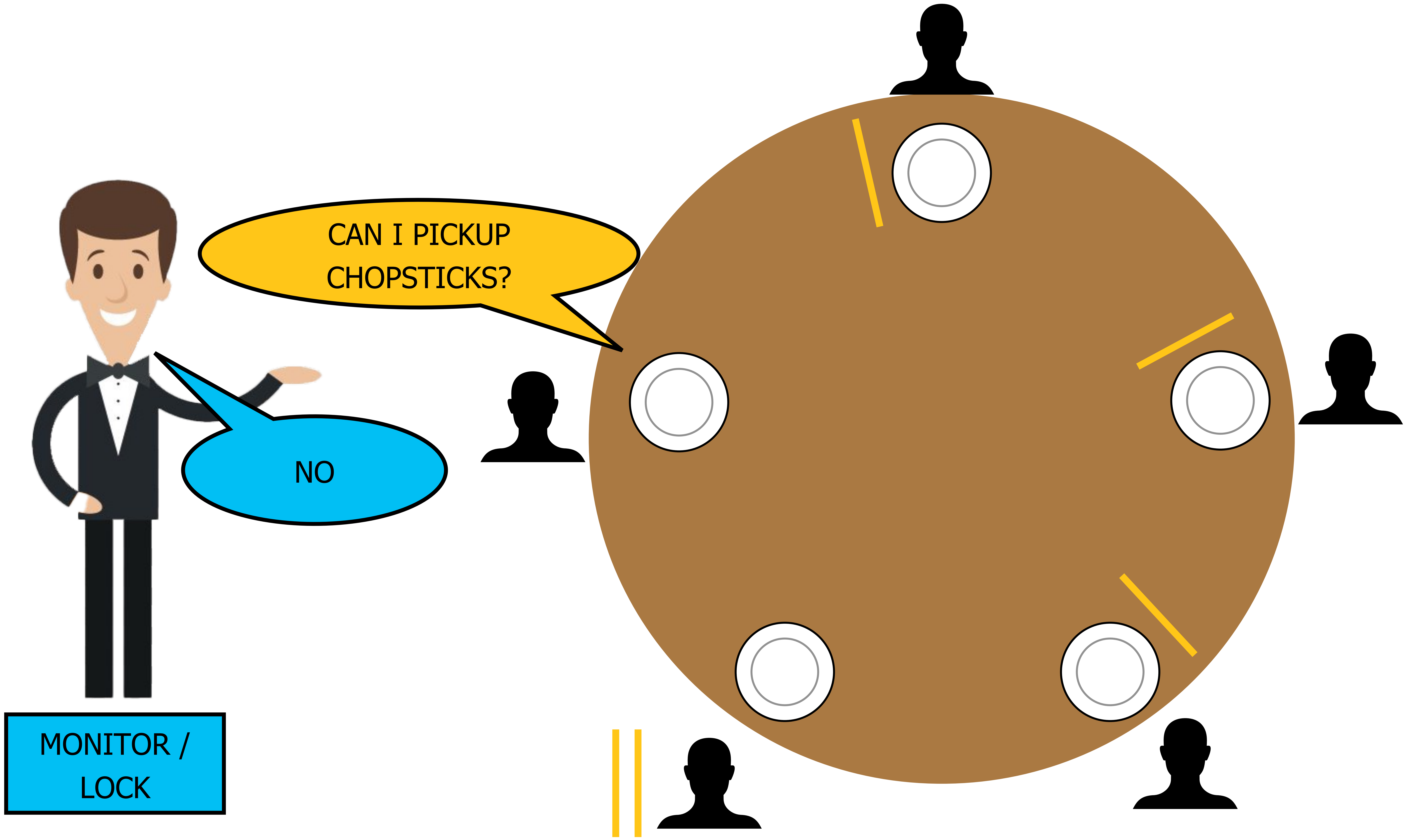
THE DINING PHILOSOPHERS PROBLEM



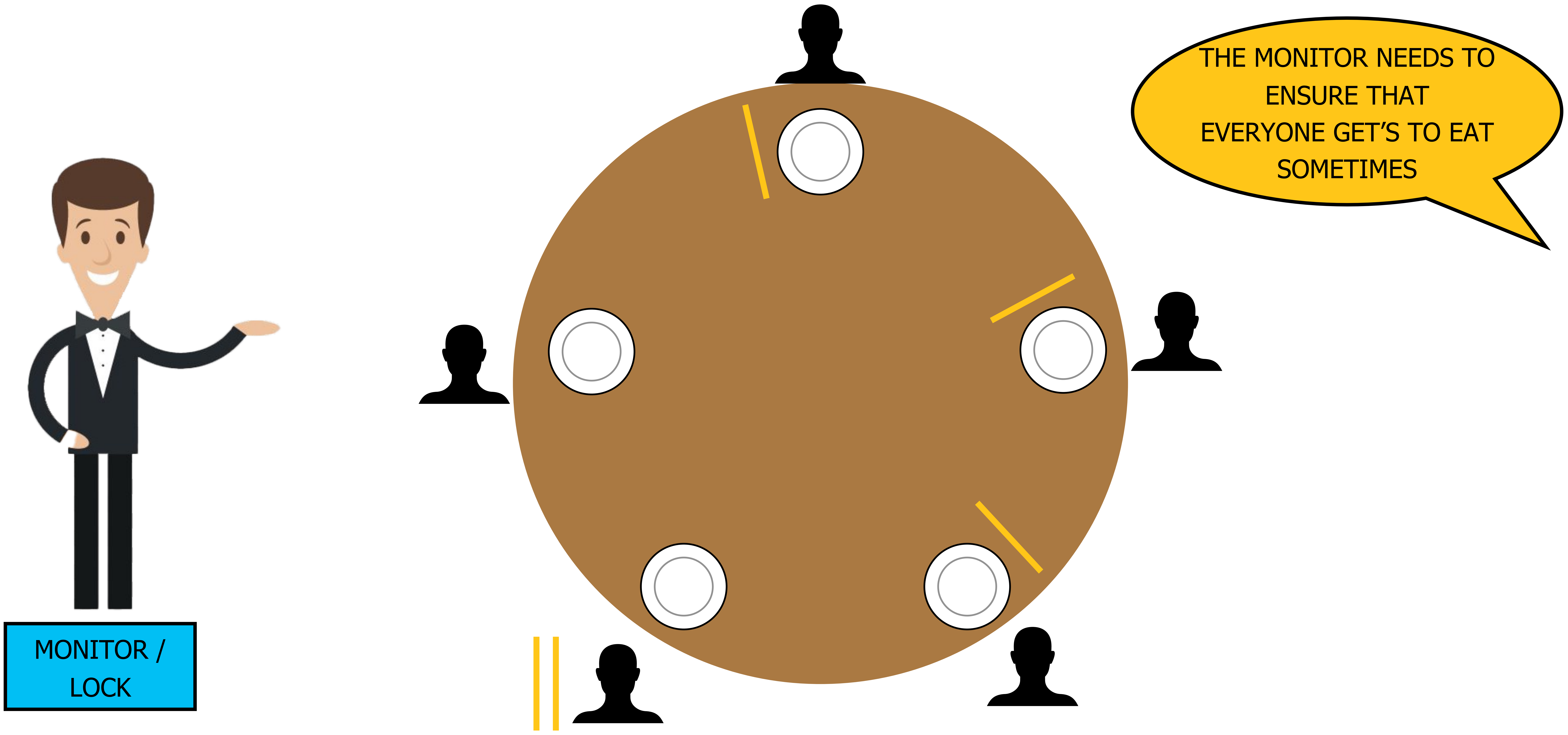
THE DINING PHILOSOPHERS PROBLEM



THE DINING PHILOSOPHERS PROBLEM



THE DINING PHILOSOPHERS PROBLEM

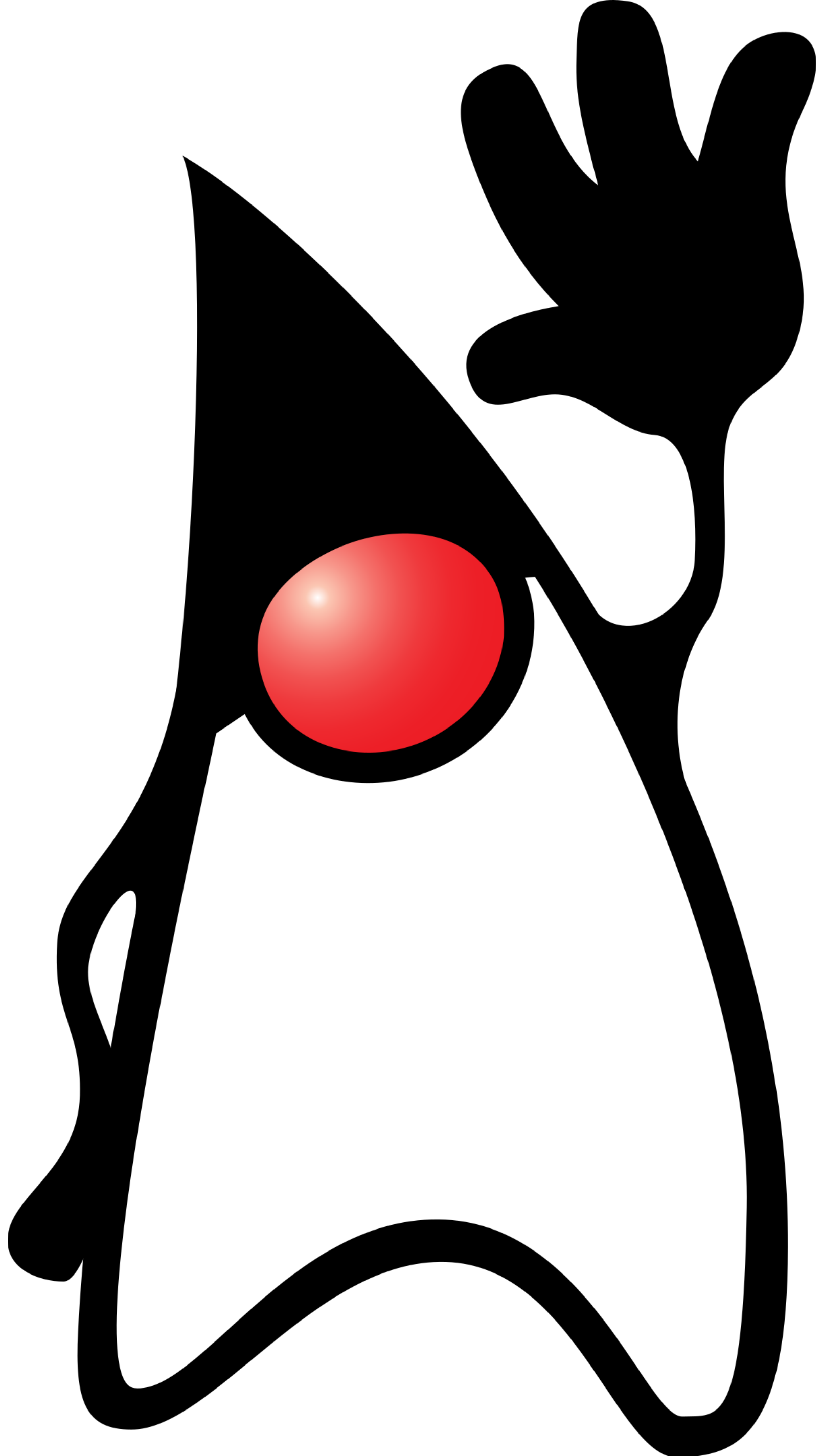

THE MONITOR NEEDS TO ENSURE THAT EVERYONE GET'S TO EAT SOMETIMES
MONITOR / LOCK

MR SADAM HUSSAIN

EXTENDED JAVA

Multithreading: Synchronisation

ADAPTED FROM DR EDWARD ANSTEAD



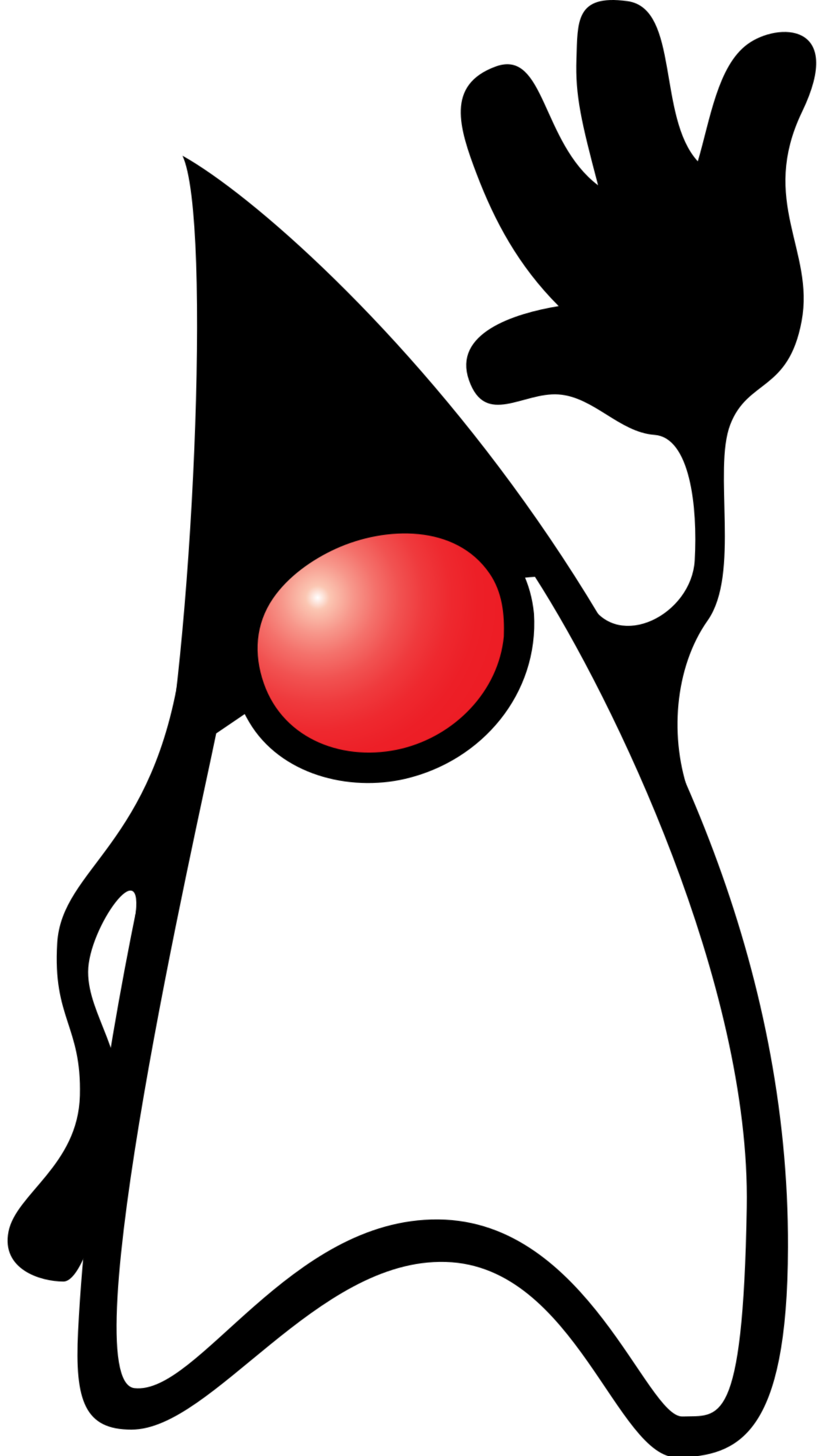
EXAMPLE SYNCHRONISATION

MR SADAM HUSSAIN

EXTENDED JAVA

Multithreading:
Inter-thread Communication

ADAPTED FROM DR EDWARD ANSTEAD



THE OBJECT CLASS - FROM INHERITANCE LECTURE (WEEK 8)

- Implicitly all classes inherit the object class
- Provides the following methods

Method	Final?	Purpose
`Object clone()`		Create a new identical object
`boolean equals(Object object)`		Determine whether one object is equal to another
`Class <?> getClass()`	Y	Obtain the class of an object at runtime
`int hashCode()`		return the hash code associated with the invoking object
`String toString()`		Return a string that describes the object
`void notify()`	Y	Part of Java's threading system
`void notifyAll()`	Y	

`void wait()`
`void wait(long milliseconds)`
`void wait (long milliseconds, int nanoseconds)`
	Y	

THE OBJECT CLASS - FROM INHERITANCE LECTURE (WEEK 8)

- Implicitly all classes inherit the object class
- Provides the following methods

Method	Final?	Purpose
`Object clone()`		Create a new identical object
`boolean equals(Object object)`		Determine whether one object is equal to another
`Class <?> getClass()`	Y	Obtain the class of an object at runtime
`int hashCode()`		return the hash code associated with the invoking object
`String toString()`		Return a string that describes the object
`void notify()`	Y	Part of Java's threading system
`void notifyAll()`	Y	

`void wait()`
`void wait(long milliseconds)`
`void wait (long milliseconds, int nanoseconds)`
	Y	

INTERPROCESS COMMUNICATION

Method	Final?	Purpose
`void notify()`	Y	Wake up a single thread that is waiting on an objects monitor
`void notifyAll()`	Y	Wake up all threads that are waiting on an objects monitor

`void wait()`
`void wait(long milliseconds)`
`void wait (long milliseconds, int nanoseconds)`
	Y	Cause thread to wait until another calls notify or until timeout

EXAMPLE: INTERPROCESS COMMUNICATION