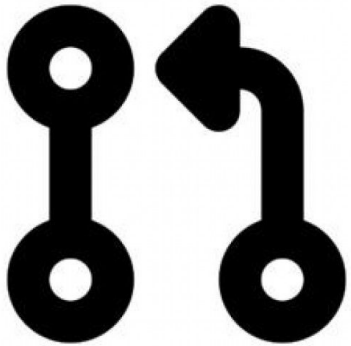


MultiThreading & Actors

Diego Pacheco

About me...

- ❑ Cat's Father
- ❑ Principal Software Architect
- ❑ Agile Coach
- ❑ SOA/Microservices Expert
- ❑ DevOps Practitioner
- ❑ Speaker
- ❑ Author

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

https://diegopacheco.github.io/

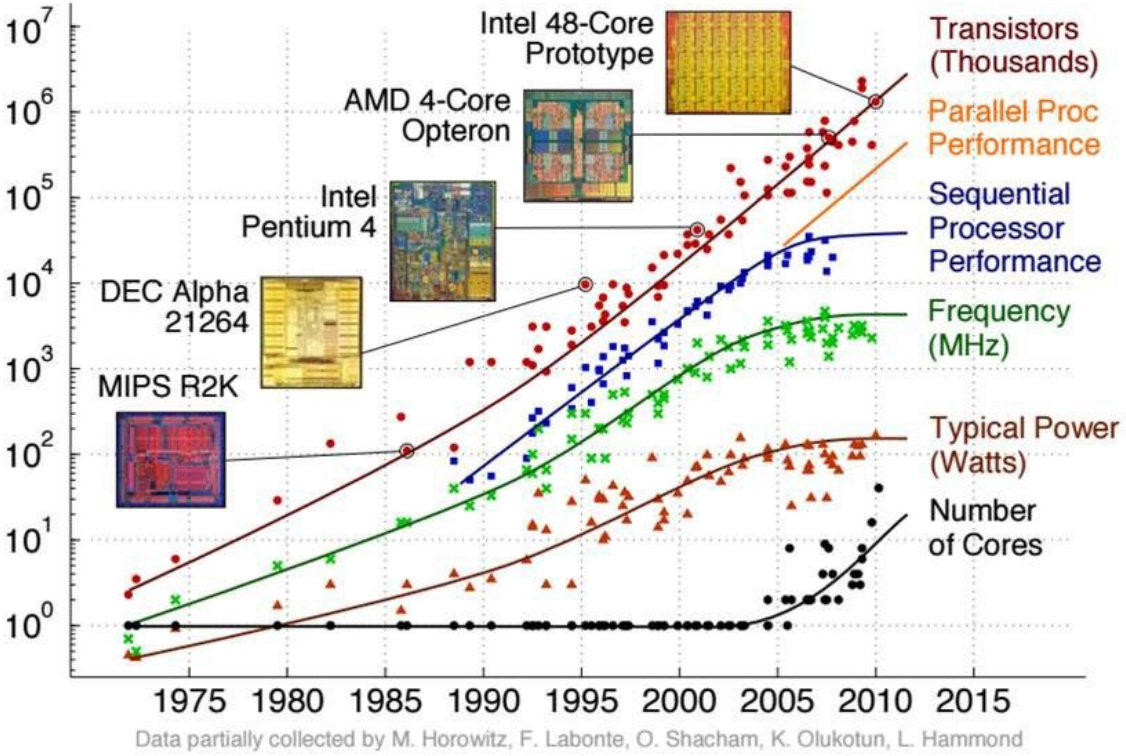
I don't need to worry about Threads. Really? Wrong!

Moore's Law of transistor density is still going, but the "**Moore's Law**" of clock speed has hit a wall. ... Instead, we increase computational throughput by using those transistors to pack multiple **processors** onto the same chip. This is referred to as multicore.

Moore's Law and Multicore - Oregon State University

web.engr.oregonstate.edu › Handouts › moores.law.and.multicore.2pp.pdf

Moore Law and the need for Threads



Reduce Latency Matters.

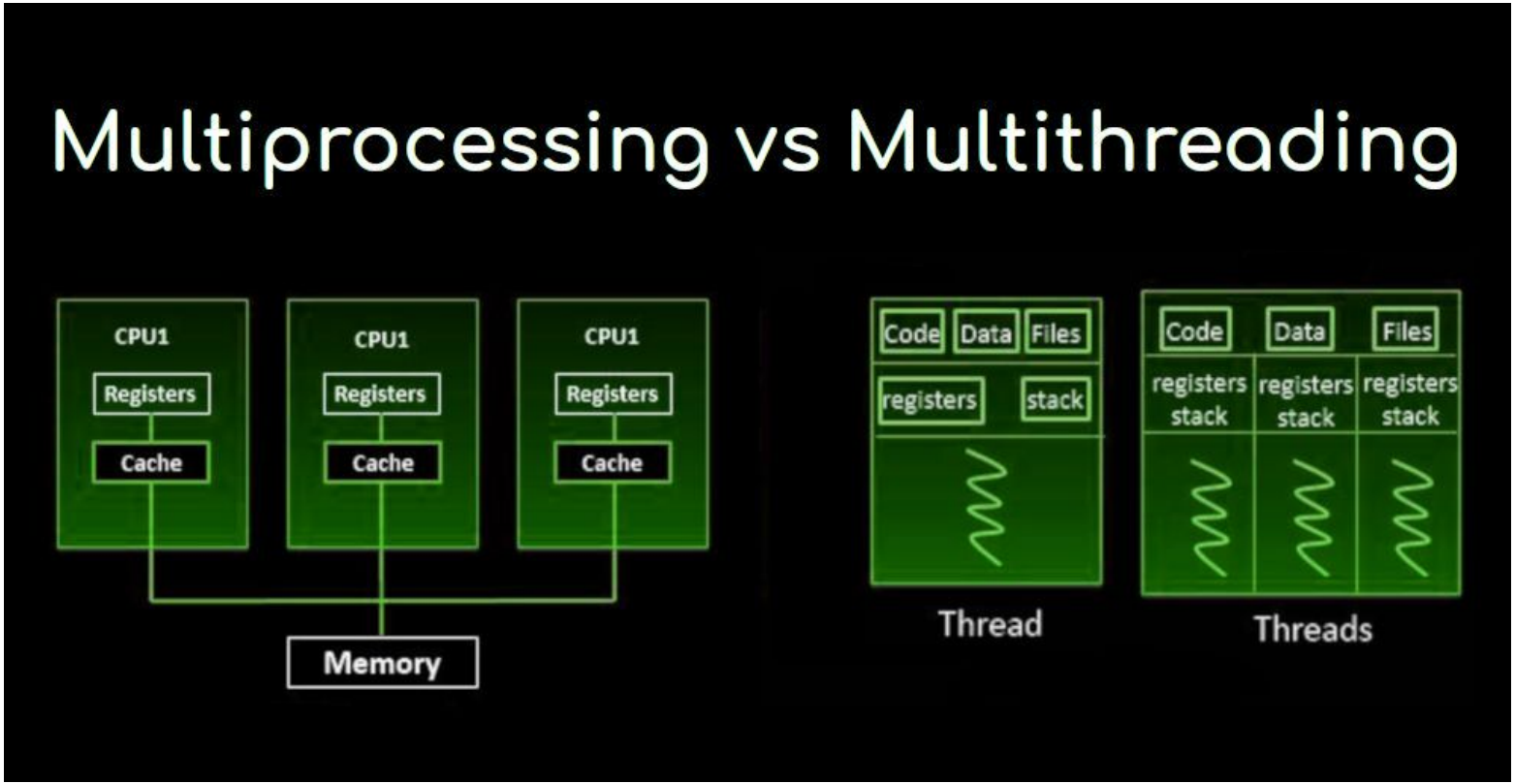
+500 ms ➔ -20% traffic

@ Google

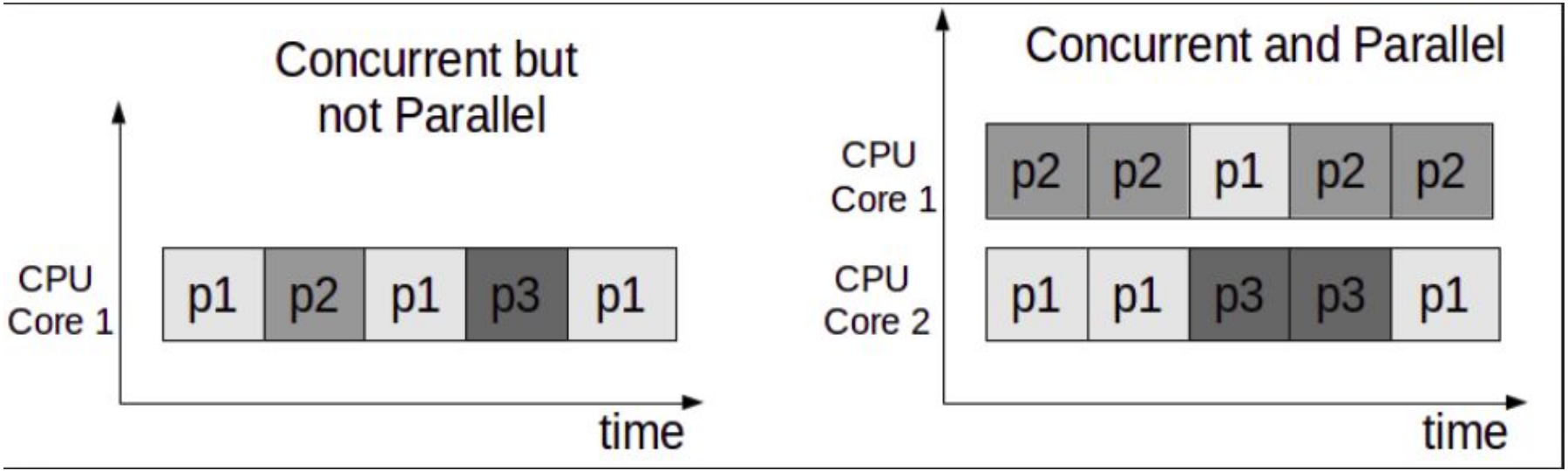
+100 ms ➔ -1% sales

@ Amazon

Multiprocessing vs multithreading



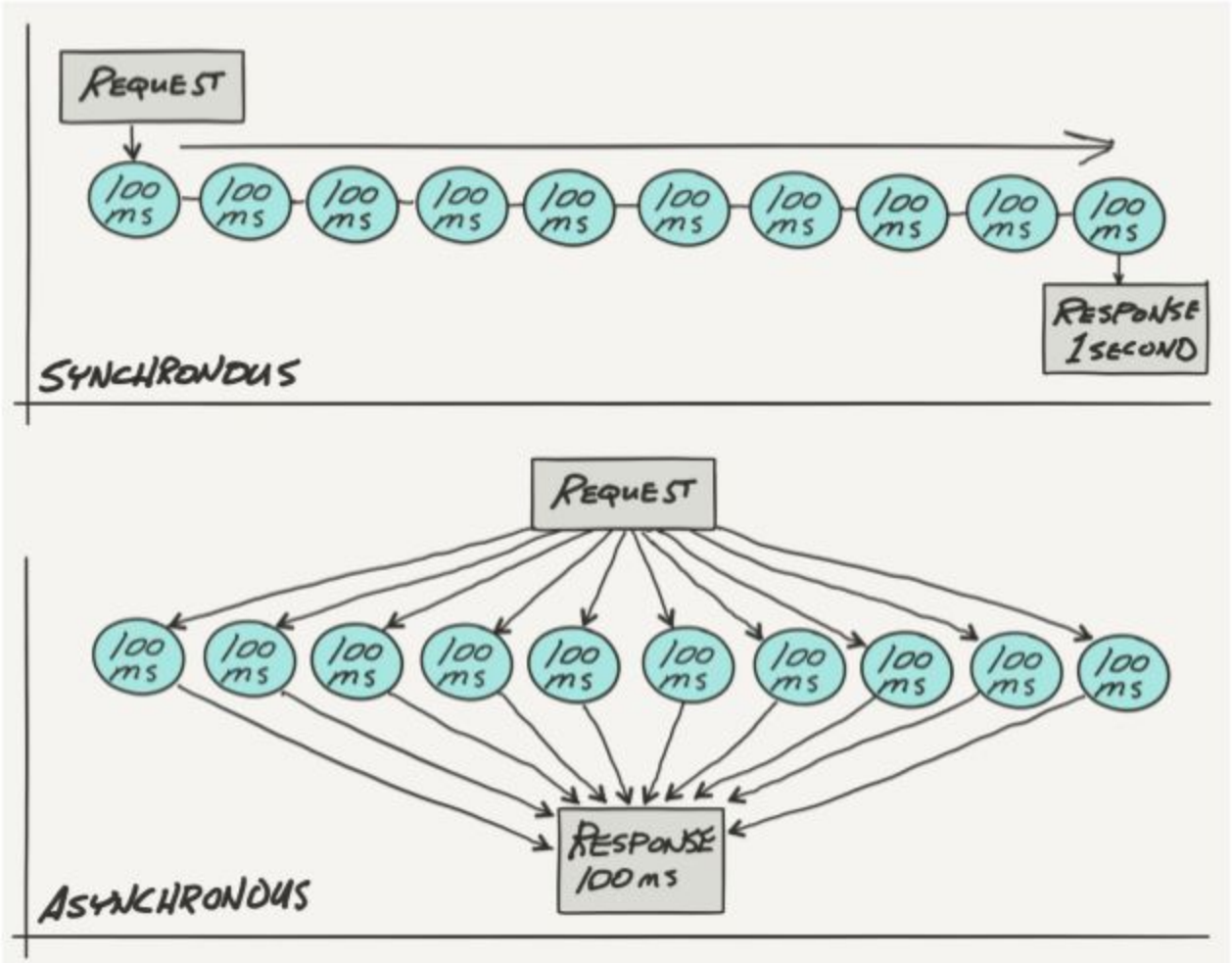
Concurrent Vs Parallel



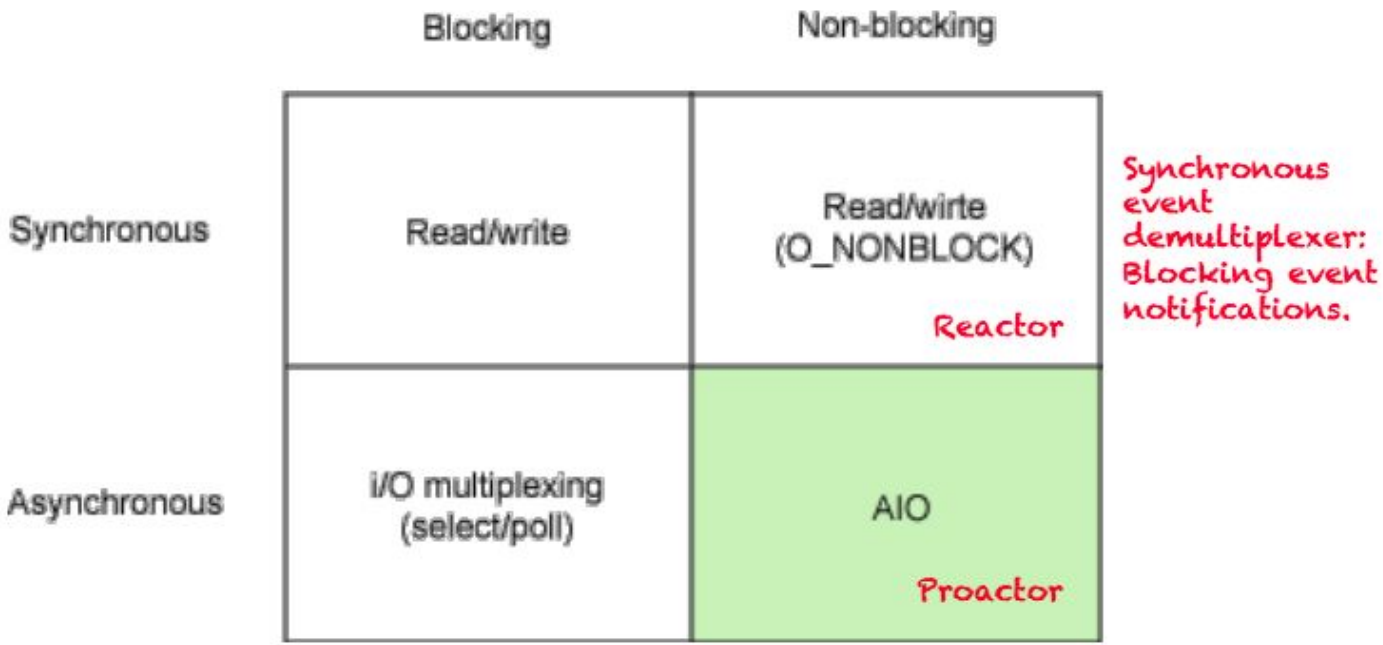
How Should I pick Concurrency or Parallelism ?

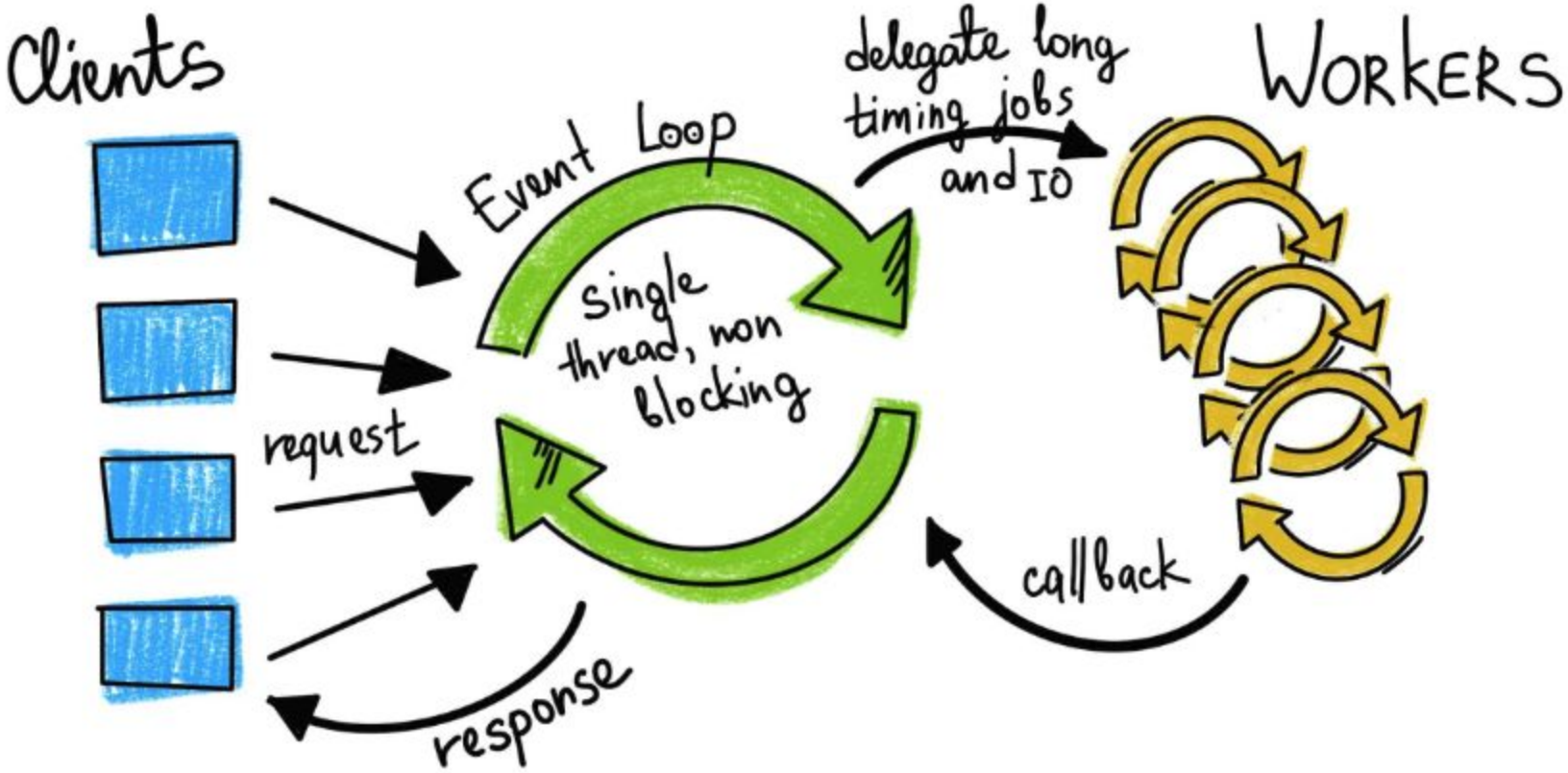
I/O Bound == Concurrency

CPU Bound == Parallelism


REQUEST
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
RESPONSE
1 SECOND
SYNCHRONOUS
REQUEST
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
100 ms
RESPONSE
100 ms
ASYNCHRONOUS

Block vs Non-Blocking IO




Clients
Event Loop
single thread, non blocking
delegate long timing jobs and IO
Workers
request
response
callback

Let's not paint a "Rose Picture" on Async

- Potentially you can have:
  - Memory Leaks
  - Race Conditions
  - Complex state Machines
  - Callback Hells (Look NodeJS)
  - Error Handler can be difficult (look RxJava)

A Simple Thread in Java

```java
package threads;

public class SimpleThreads{
    Run | Debug
    public static void main(String[] args) throws Exception{

        Runnable myCode = new Runnable(){
            @Override
            public void run() {
                System.out.println("Running " +  Thread.currentThread().getName());
                System.out.println("DONE");
            }
        };

        Thread t = new Thread(myCode);
        t.setName("My-First-Thread");
        t.start();
        t.join();
    }
}
```

Thread issues

- ❑ No Language has a great abstraction to model TIME.
- ❑ Deadlocks
- ❑ Race Conditions
- ❑ Memory Inconsistency Errors (shared State)
- ❑ Bugs & Troubleshooting Issues

Oopsie! Something went wrong...

```
static class UnsafeGlobalUnsafeCounter{
    private Integer counter = 0;
    public  Integer getCount(){ return counter; }
    public  Integer inc(){ return ++counter; }
}
```

Oopsie! Something went wrong...

```
Run | Debug
public static void main(String[] args) throws Exception {

    UnsafeGlobalUnsafeCounter counter = new UnsafeGlobalUnsafeCounter();

    Runnable code = new Runnable(){
        @Override
        public void run() {
            for(int i=1;i<=10;i++){
                System.out.println("Tread " + Thread.currentThread().getName() + " = " +
                counter.inc());
            }
        }
    };
    Thread t1 = new Thread(code,"t1");
    Thread t2 = new Thread(code,"t2");
    t1.start();
    t2.start();
    t1.join();
    t2.join();

}
```

Synchronized to rescue...

```
static class GlobalSafeCounter{
    private Integer counter = 0;
    public  Integer getCount(){ return counter; }
    public  synchronized Integer inc(){ return ++counter; }
}
```

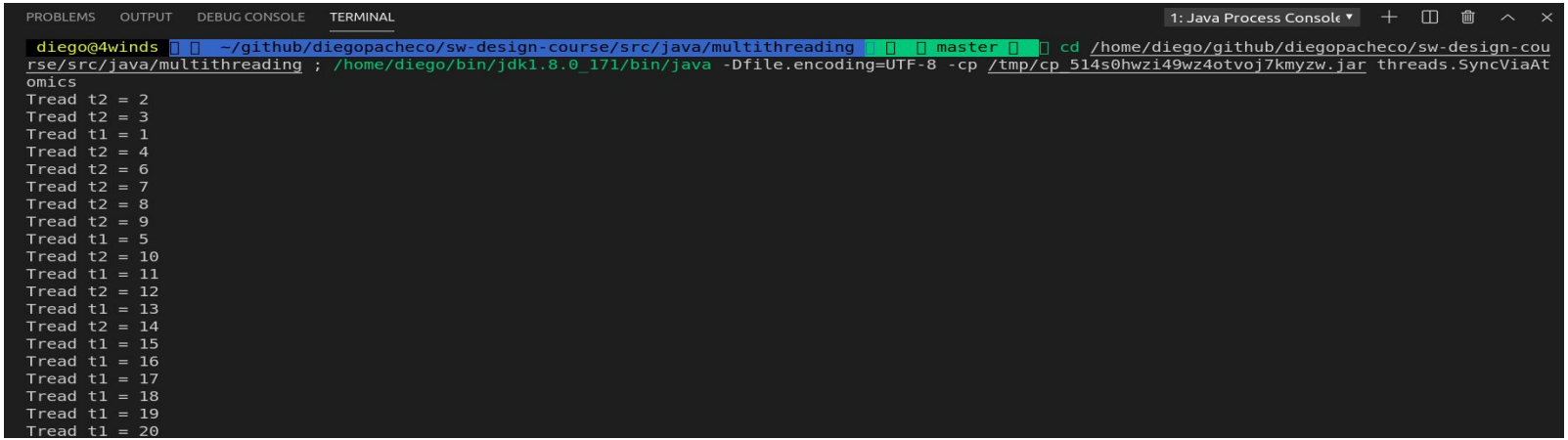
```
diego@4winds  ~/github/diegopacheco/sw-design-course/src/java/multithreading  master  /home/diego/bin/jdk1.8.0_171/bin/java -Dfile.enc
oding=UTF-8 -cp /tmp/cp_514s0hwzi49wz4otvoj7kmyzw.jar threads.SyncMainFixed
Tread t2 = 1
Tread t1 = 2
Tread t2 = 3
Tread t1 = 4
Tread t2 = 5
Tread t1 = 6
Tread t2 = 7
Tread t1 = 8
Tread t2 = 9
Tread t1 = 10
Tread t2 = 11
Tread t1 = 12
Tread t2 = 13
Tread t1 = 14
Tread t2 = 15
Tread t1 = 16
Tread t2 = 17
Tread t1 = 18
Tread t2 = 19
Tread t1 = 20
```

Synchronized to rescue...

```
Run | Debug
public static void main(String[] args) throws Exception{
    GlobalSafeCounter counter = new GlobalSafeCounter();
    Runnable code = new Runnable(){
        @Override
        public void run() {
            for(int i=1;i<=10;i++){
                System.out.println("Tread " + Thread.currentThread().getName() + " = " +
                counter.inc());
            }
        }
    };
    Thread t1 = new Thread(code,"t1");
    Thread t2 = new Thread(code,"t2");
    t1.start();
    t2.start();
    t1.join();
    t2.join();
}
```

Atomics to rescue...

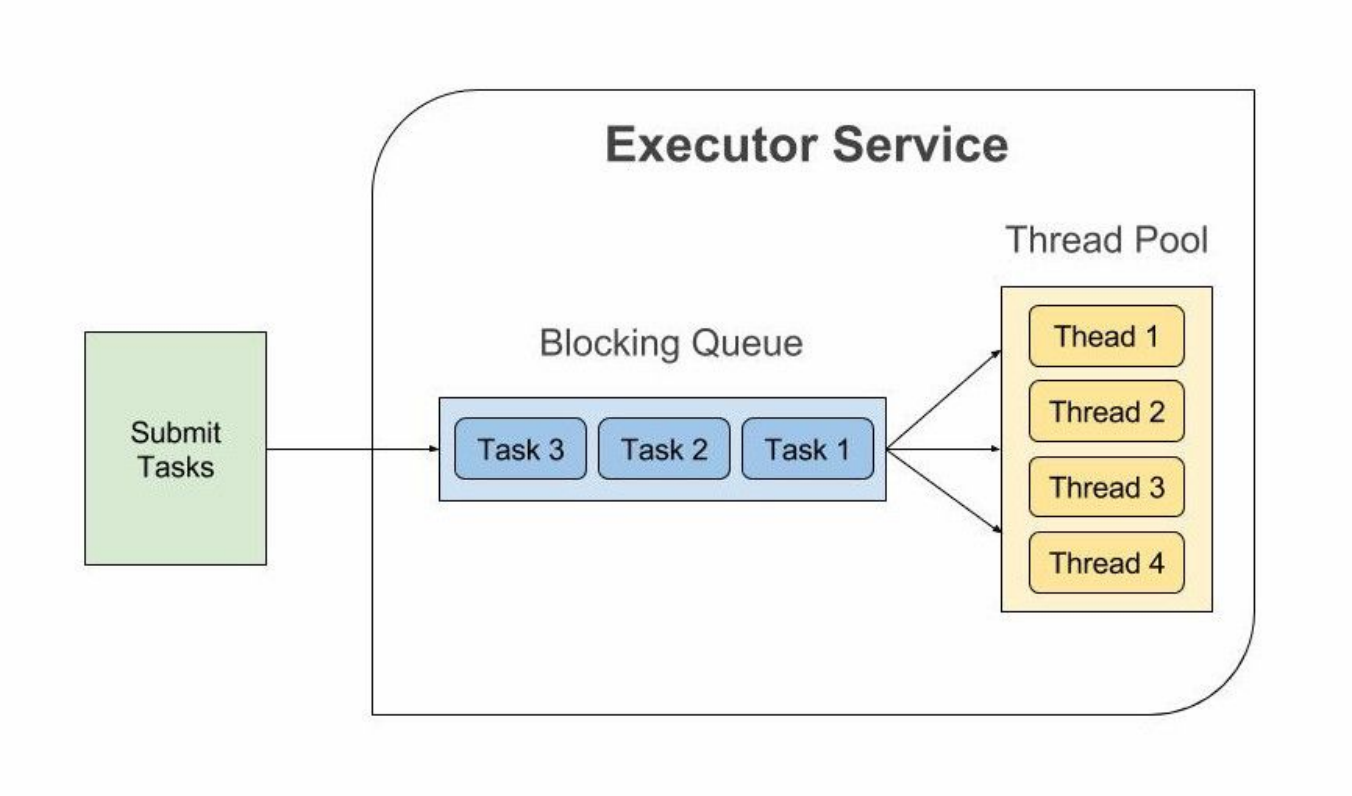
```java
static class GlobalSafeAtomicCounter{
    private AtomicInteger counter = new AtomicInteger(0);
    public  Integer getCount(){ return counter.get(); }
    public  Integer inc(){ return counter.incrementAndGet(); }
}
```



Atomics to rescue...

```
public static void main(String[] args) throws Exception{
    GlobalSafeAtomicCounter counter = new GlobalSafeAtomicCounter();
    Runnable code = new Runnable(){
        @Override
        public void run() {
            for(int i=1;i<=10;i++){
                System.out.println("Tread " + Thread.currentThread().getName() + " = " +
                counter.inc());
            }
        }
    };
    Thread t1 = new Thread(code,"t1");
    Thread t2 = new Thread(code,"t2");
    t1.start();
    t2.start();
    t1.join();
    t2.join();
}
```

Executors

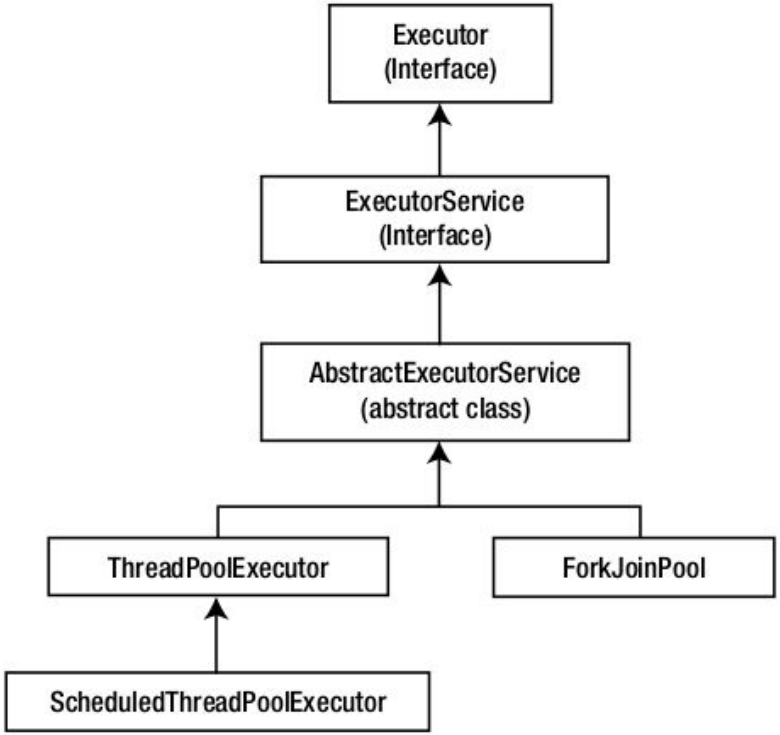

Executor Service
Thread Pool
Blocking Queue
Submit Tasks
Task 3
Task 2
Task 1
Thead 1
Thread 2
Thread 3
Thread 4

Executors

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;

public class ExecutorServiceMain{

    static class GlobalSafeCounter{
        private AtomicInteger counter = new AtomicInteger(0);
        public  Integer getCount(){ return counter.get(); }
        public  Integer inc(){ return counter.incrementAndGet(); }
    }

    Run | Debug
    public static void main(String[] args) throws Exception{
        GlobalSafeCounter counter = new GlobalSafeCounter();
        ExecutorService executorService = Executors.newFixedThreadPool(10);
        for(int i = 0; i < 10; i++) {
            executorService.submit(() -> counter.inc());
        }
        executorService.shutdown();
        executorService.awaitTermination(60, TimeUnit.SECONDS);
        System.out.println("Final count is : " + counter.getCount());
    }
}
```

- Safe if your code is safe.
- Better Abstraction then Direct Threads.
- Don't forget to shutdown.

Executors



- Less worries.
- Thread Management
- Works with Queues
- Your code still need to be Thread SAFE.
- Create ONCE, re-use as much as you can.
- Be Careful with "clone".

Executors

```
3   import java.util.concurrent.Callable;
4   import java.util.concurrent.Executors;
5   import java.util.concurrent.Future;
6   import java.util.concurrent.ScheduledExecutorService;
7   import java.util.concurrent.TimeUnit;
8
9   public class ScheduledExecutorsMain {
        Run | Debug
10      public static void main(String[] args) throws Exception{
11
12          Callable<String> callableTask = new Callable<String>() {
13                  @Override
14                  public String call() throws Exception {
15                      System.out.println("Printing *** ");
16                      return "Printing *** ";
17                  }
18          };
19
20          ScheduledExecutorService executorService = Executors.newSingleThreadScheduledExecutor();
21          Future<String> resultFuture = executorService.schedule(callableTask, 1, TimeUnit.SECONDS);
22          System.out.println("After 1 sec Delay - we got: " + resultFuture.get());
23          executorService.shutdown();
24      }
25  }
```

Executors

```java
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class ScheduledExecutorsMain2 {
    Run | Debug
    public static void main(String[] args) throws Exception {

        Runnable runnableTask = new Runnable(){
            @Override
            public void run() {
                System.out.println("Printing *** ");
            }
        };

        ScheduledExecutorService executorService = Executors.newSingleThreadScheduledExecutor();
        executorService.scheduleAtFixedRate(runnableTask, 0, 1, TimeUnit.SECONDS);
        Thread.sleep(4000L);
        executorService.shutdownNow();
        System.out.println("END");
    }
}
```

Executors

```
diego@4winds  ~/github/diegopacheco/sw-design-course/src/java/multithreading  master  /home/diego/bin/jdk1
.8.0_171/bin/java -Dfile.encoding=UTF-8 -cp /tmp/cp_514s0hwzi49wz4otvoj7kmyzw.jar threads.ScheduledExecutorsMain2
Printing ***
Printing ***
Printing ***
Printing ***
Printing ***
END
diego@4winds  ~/github/diegopacheco/sw-design-course/src/java/multithreading  master
```

Locks

```
diego@4winds  ~/github/diegopacheco/sw-design-course/src/java/multithreading  master  cd /home/diego/github/diegopacheco/sw-design-course/src/java/multithreading ; /home/diego/bin/jdk1.8.0_171/bin/java -Dfile.encoding=UTF-8 -cp /tmp/cp_514s0hwzi49wz4otvoj7kmyzw.jar threads.LocksMain
Tread t2 = 2
Tread t1 = 1
Tread t2 = 3
Tread t1 = 4
Tread t2 = 5
Tread t1 = 6
Tread t2 = 7
Tread t1 = 8
Tread t2 = 9
Tread t1 = 10
Tread t2 = 11
Tread t1 = 12
Tread t2 = 13
Tread t1 = 14
Tread t2 = 15
Tread t1 = 16
Tread t2 = 17
Tread t1 = 18
Tread t2 = 19
Tread t1 = 20
```

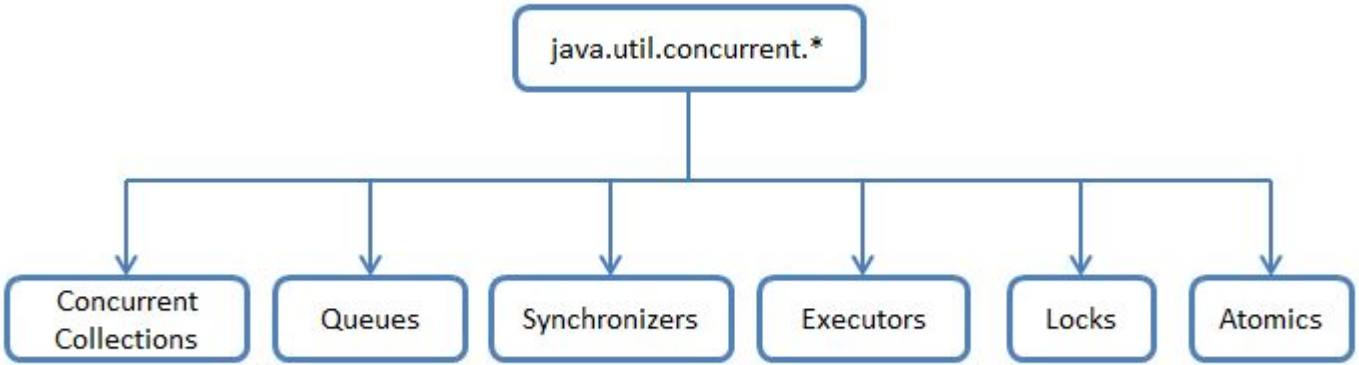
Locks

```
static class GlobalSafeCounterWithLocks{
    ReentrantLock lock = new ReentrantLock();
    private Integer counter = 0;
    public  Integer getCount(){ return counter; }
    public  Integer inc(){
        lock.lock();
        try {
            return ++counter;
        } finally {
            lock.unlock();
        }
    }
}
```

Locks

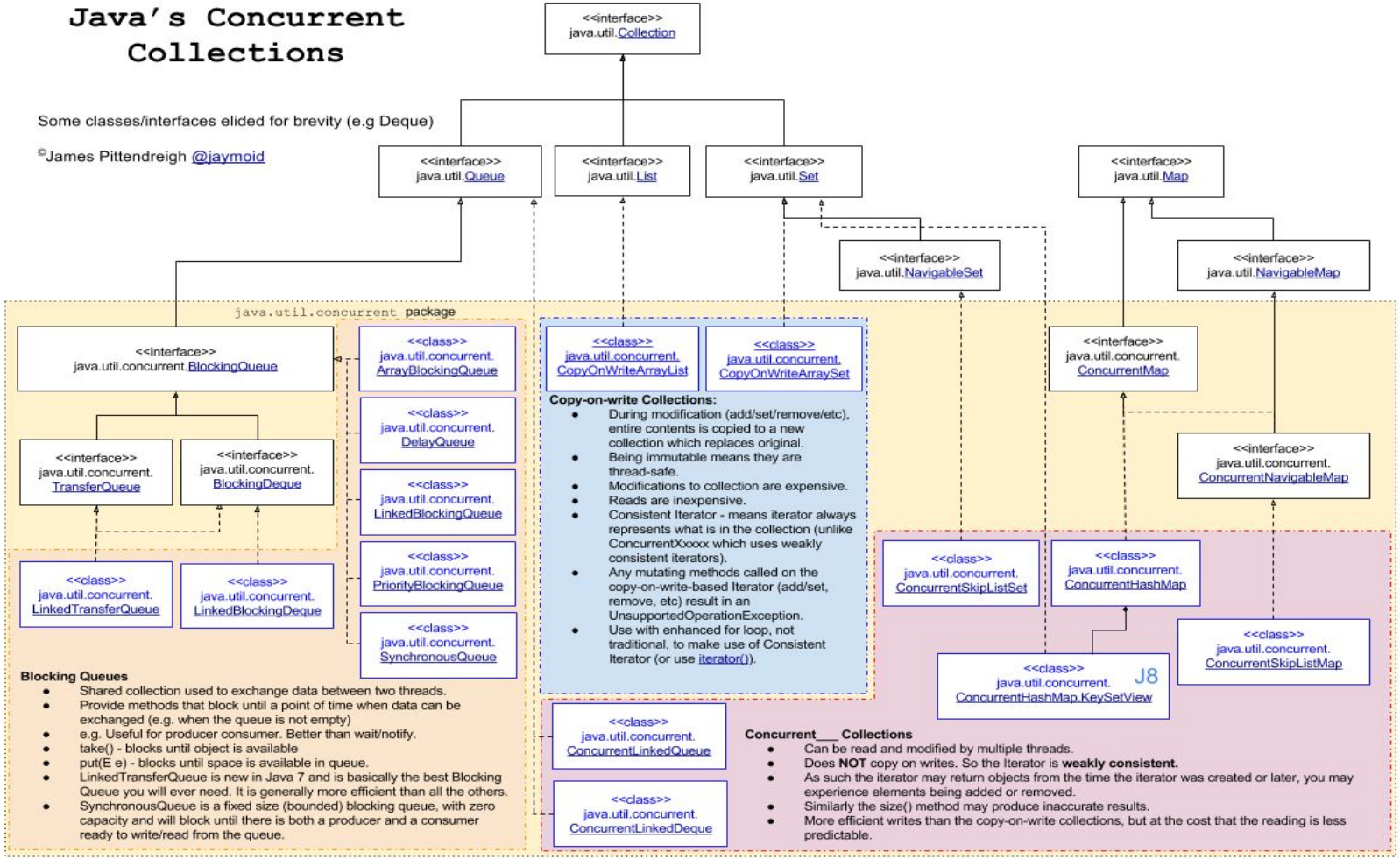
```
Run | Debug
public static void main(String[] args) throws Exception{
    GlobalSafeCounterWithLocks counter = new GlobalSafeCounterWithLocks();
    Runnable code = new Runnable(){
        @Override
        public void run() {
            for(int i=1;i<=10;i++){
                System.out.println("Tread " + Thread.currentThread().getName() + " = " +
                counter.inc());
            }
        }
    };
    Thread t1 = new Thread(code,"t1");
    Thread t2 = new Thread(code,"t2");
    t1.start();
    t2.start();
    t1.join();
    t2.join();
}
```

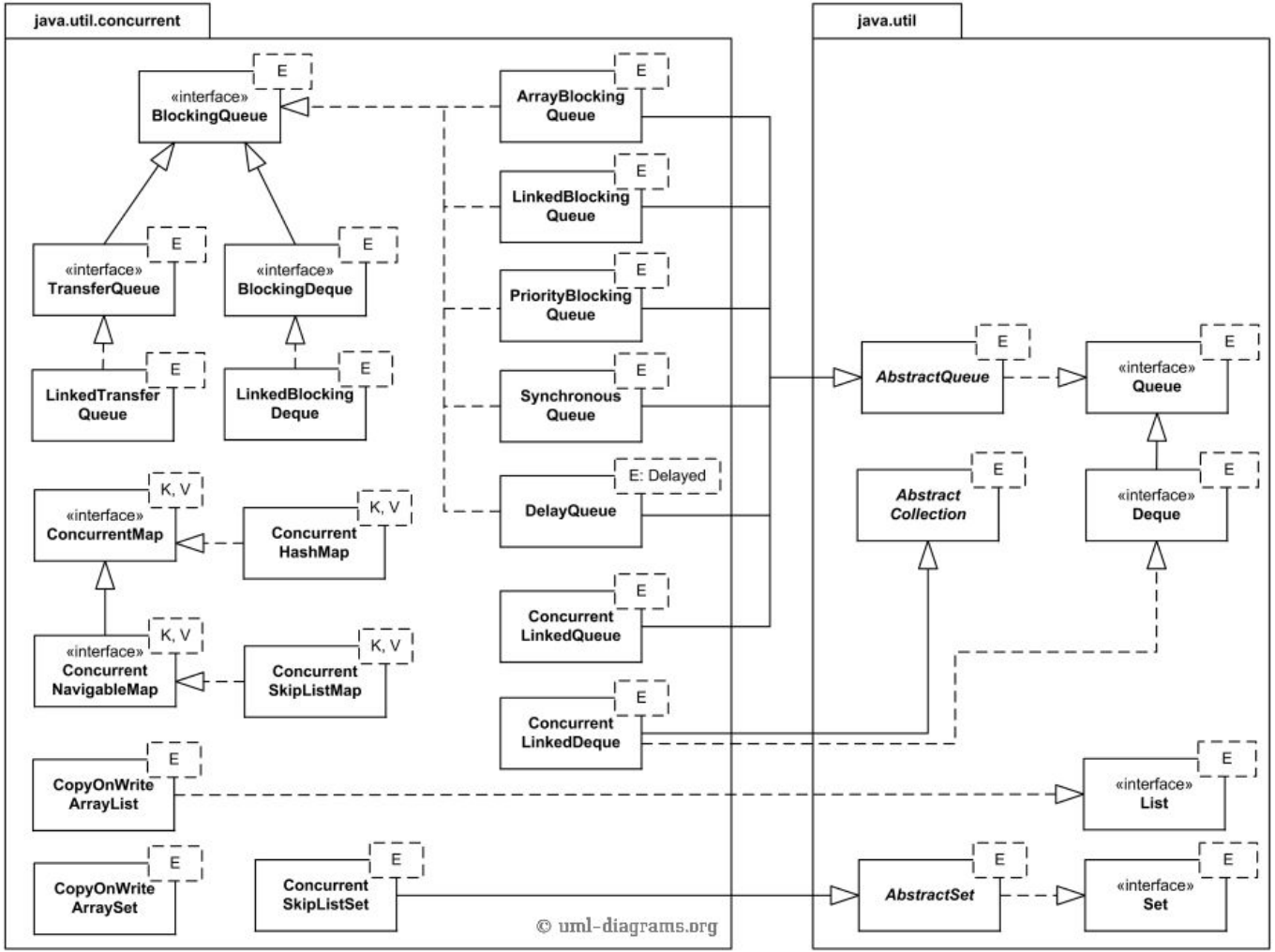
Java Concurrent Collections



Java's Concurrent Collections

Some classes/interfaces elided for brevity (e.g Deque)




java.util.concurrent
java.util
«interface»
BlockingQueue
E
ArrayBlocking
Queue
LinkedBlocking
Queue
PriorityBlocking
Queue
Synchronous
Queue
DelayQueue
E: Delayed
Concurrent
LinkedQueue
Concurrent
LinkedDeque
«interface»
TransferQueue
«interface»
BlockingDeque
LinkedTransfer
Queue
LinkedBlocking
Deque
«interface»
ConcurrentMap
K, V
Concurrent
HashMap
«interface»
Concurrent
NavigableMap
Concurrent
SkipListMap
CopyOnWrite
ArrayList
CopyOnWrite
ArraySet
Concurrent
SkipListSet
AbstractQueue
«interface»
Queue
Abstract
Collection
«interface»
Deque
«interface»
List
AbstractSet
«interface»
Set
© uml-diagrams.org

Threads Exercise (with Java >= 8)

Constraints: You cannot use frameworks, only the "things" you learned in this class. Use Java.

1. Build a concurrent program with prints the time table of 7 (7x1, 7x2, 7x3... 7x10) all in parallel and when it ends print the results in order.
2. Build a concurrent program which calculates the factorial from 1 to 200 in parallel. Print the results at the end when all calculations are done.
3. Build a concurrent program that count words in a file wich has 10mb. The counting must happen in parallel as well.
4. Design and code a Worker from sratch, you should have a Queue and manage threads for the user. You cannot use frameworks on any high level features like Service Executors or Java Thread Pools - create your own.

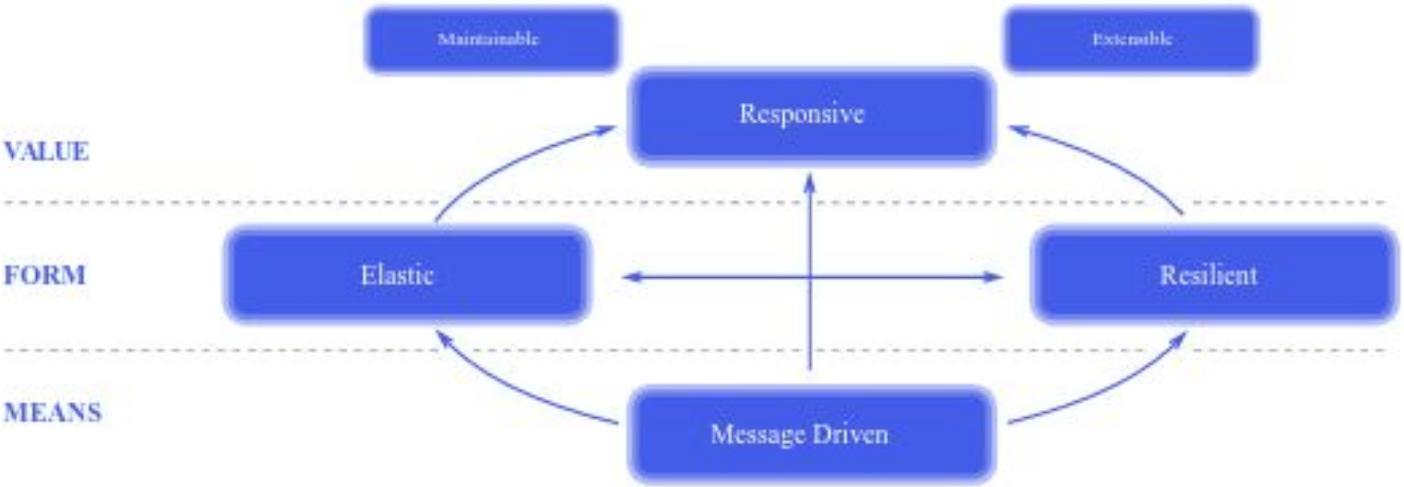
Actors & Akka

Reactive Stream

- Standard Async Stream Processing
- Non-Blocking back pressure
- Equivalent Reactive Stream -> JDK9 java.util.concurrent.Flow
- Handle Stream of Data Issues
  - Resource Consumption (i.g: Flood of Fast Data)
  - Right Async code in order to Handle multiple CPU Cores
- Governance or Stream Data Exchanges (avoid arbitrary buffering)

https://www.reactive-streams.org/

Reactive Manifesto (Value from Non-Blocking + Async)



https://www.reactivemanifesto.org/

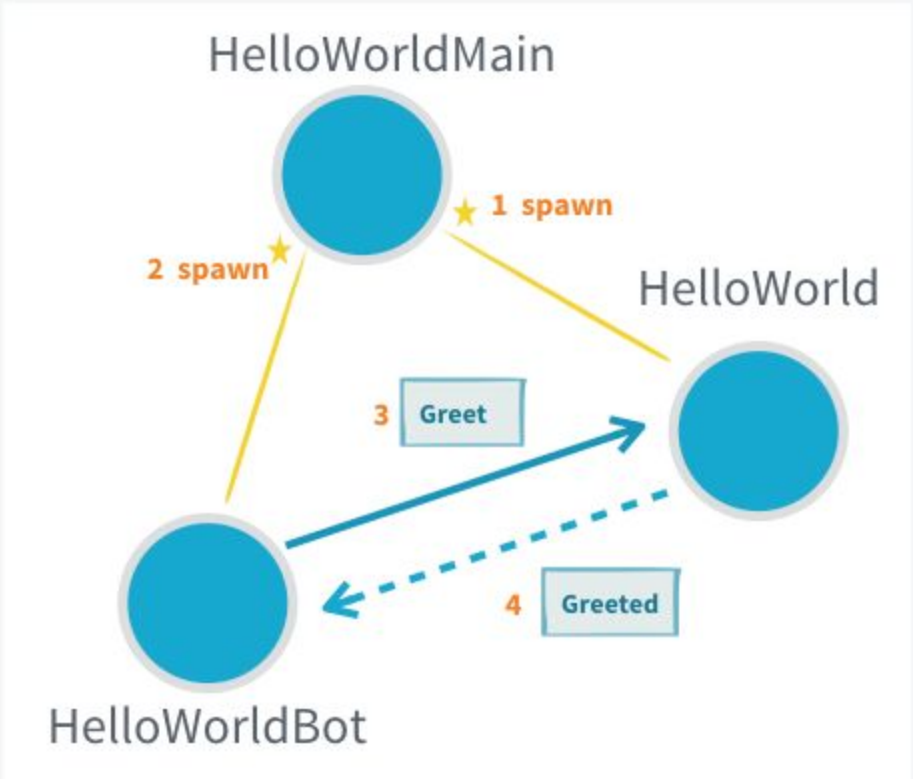
Actors & Akka

- Mature Actor ecosystem based on Erlang / OTP
- "Simple" concurrent Distributed Systems
- Resilient By Design
- Implement Reactive Streams principles
- High Performance & Scalability
  - Up to 50 million msg/sec on a single machine.
  - Small memory footprint; ~2.5 million actors 1GB.

Actors Issues

- Actors Code Reuse is complicated.
- Not meant for all kinds of problems.
- Does not work well on a "PURE" RPC system. Since actors systems are similar to state machines.
- Can be much more complex than regular service development.
- Testing is Painful (being improved) but still.

Typed Actors with Scala



akka

Typed Actors with Scala

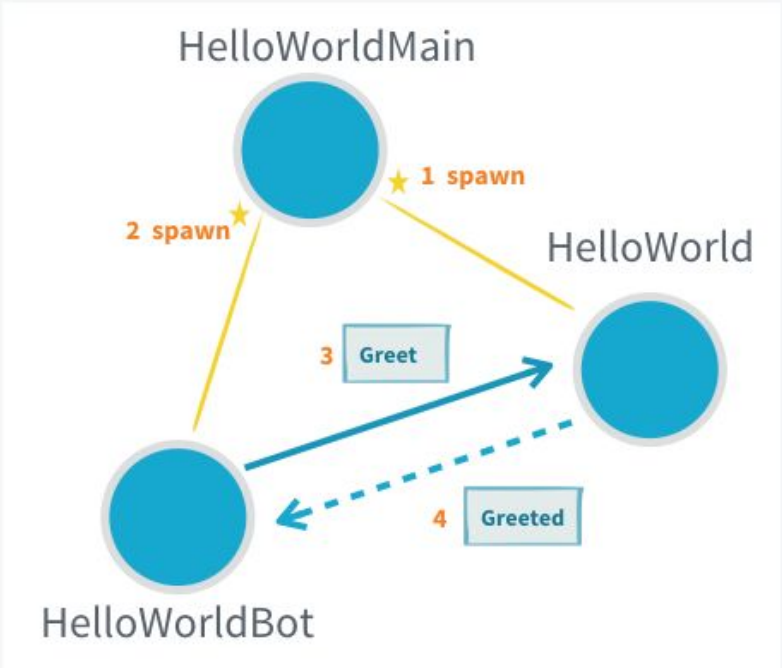
build.sbt

```
name := "actors-scala"
version := "1.0"
scalaVersion := "2.13.1"

libraryDependencies += "com.typesafe.akka" %% "akka-actor" % "2.6.0"
libraryDependencies += "com.typesafe.akka" %% "akka-actor-typed" % "2.6.0"
libraryDependencies += "ch.qos.logback" % "logback-classic" % "1.1.3" % Runtime
libraryDependencies += "com.typesafe.akka" %% "akka-testkit" % "2.6.0" % Test

```

```
$ sbt compile
$ sbt run
```

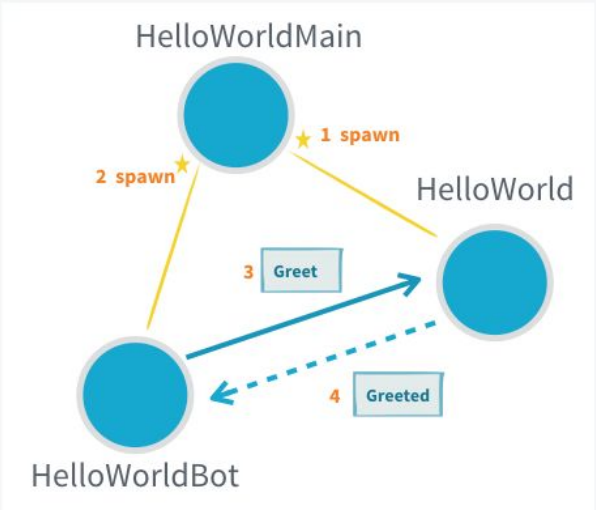
Messages == Part of The protocol (Actors are the other part)

```
final case class Greet(whom: String, replyTo: ActorRef[Greeted])
final case class Greeted(whom: String, from: ActorRef[Greet])
```



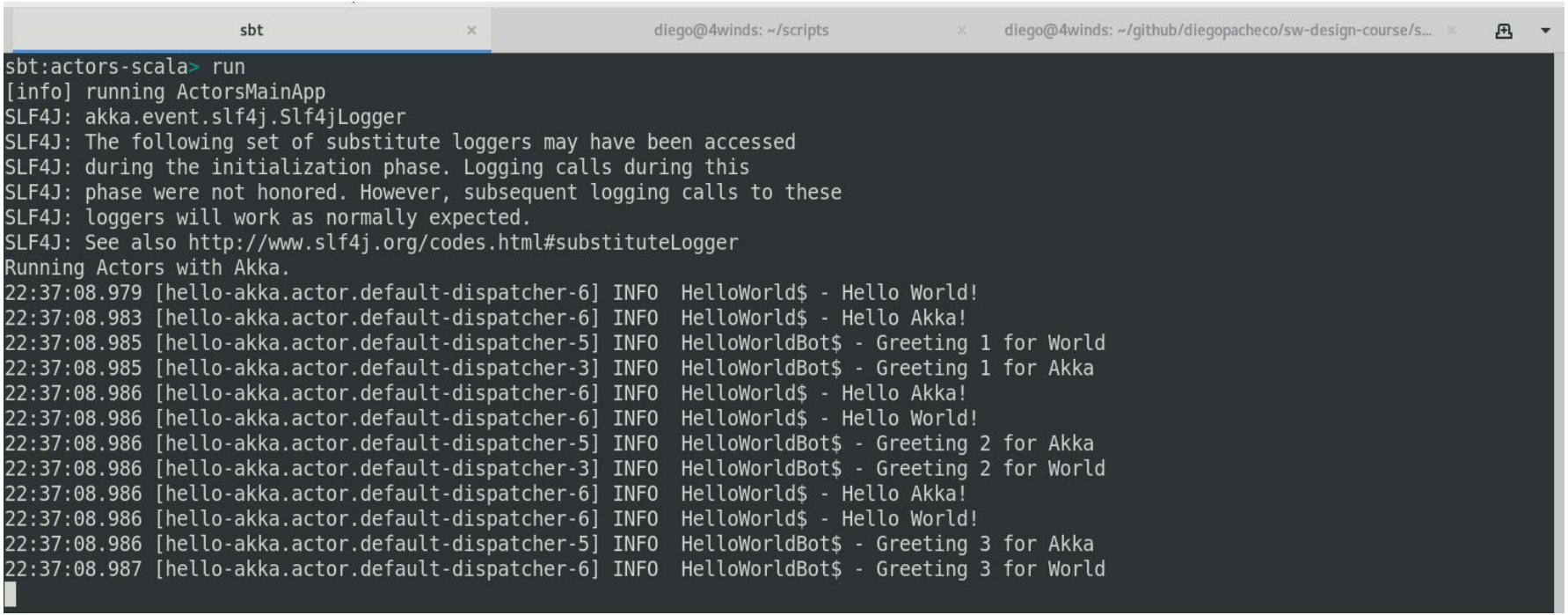
Create the Actor System and Run

```
53  object ActorsMainApp extends App {
54      val system: ActorSystem[HelloWorldMain.Start] = ActorSystem(HelloWorldMain(), "hello")
55      system ! HelloWorldMain.Start("World")
56      system ! HelloWorldMain.Start("Akka")
57      println("Running Actors with Akka.")
58  }
```




```
sbt:actors-scala> run
[info] running ActorsMainApp
SLF4J: akka.event.slf4j.Slf4jLogger
SLF4J: The following set of substitute loggers may have been accessed
SLF4J: during the initialization phase. Logging calls during this
SLF4J: phase were not honored. However, subsequent logging calls to these
SLF4J: loggers will work as normally expected.
SLF4J: See also http://www.slf4j.org/codes.html#substituteLogger
Running Actors with Akka.
22:37:08.979 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorld$ - Hello World!
22:37:08.983 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorld$ - Hello Akka!
22:37:08.985 [hello-akka.actor.default-dispatcher-5] INFO  HelloWorldBot$ - Greeting 1 for World
22:37:08.985 [hello-akka.actor.default-dispatcher-3] INFO  HelloWorldBot$ - Greeting 1 for Akka
22:37:08.986 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorld$ - Hello Akka!
22:37:08.986 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorld$ - Hello World!
22:37:08.986 [hello-akka.actor.default-dispatcher-5] INFO  HelloWorldBot$ - Greeting 2 for Akka
22:37:08.986 [hello-akka.actor.default-dispatcher-3] INFO  HelloWorldBot$ - Greeting 2 for World
22:37:08.986 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorld$ - Hello Akka!
22:37:08.986 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorld$ - Hello World!
22:37:08.986 [hello-akka.actor.default-dispatcher-5] INFO  HelloWorldBot$ - Greeting 3 for Akka
22:37:08.987 [hello-akka.actor.default-dispatcher-6] INFO  HelloWorldBot$ - Greeting 3 for World
```

Typed Actors with Scala



Typed Actors with Scala ~ HelloWorld

```scala
import akka.actor.typed.scaladsl.Behaviors
import akka.actor.typed.scaladsl.LoggerOps
import akka.actor.typed.{ ActorRef, ActorSystem, Behavior }

object HelloWorld {
  final case class Greet(whom: String, replyTo: ActorRef[Greeted])
  final case class Greeted(whom: String, from: ActorRef[Greet])

  def apply(): Behavior[Greet] = Behaviors.receive { (context, message) =>
    context.log.info("Hello {}!", message.whom)
    message.replyTo ! Greeted(message.whom, context.self)

    // because we dont need change stae
    Behaviors.same
  }
}
```

Typed Actors with Scala ~ HelloWorldBot

```scala
object HelloWorldBot {

  def apply(max: Int): Behavior[HelloWorld.Greeted] = {
    bot(0, max)
  }

  private def bot(greetingCounter: Int, max: Int): Behavior[HelloWorld.Greeted] =
    Behaviors.receive { (context, message) =>
      val n = greetingCounter + 1
      context.log.info2("Greeting {} for {}", n, message.whom)
      if (n == max) {
        Behaviors.stopped
      } else {
        message.from ! HelloWorld.Greet(message.whom, context.self)
        bot(n, max)
      }
    }
}
```

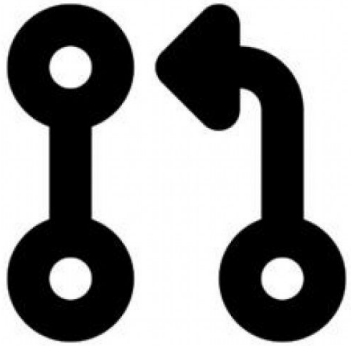
Typed Actors with Scala ~ HelloWorldMain

```scala
object HelloWorldMain {

  final case class Start(name: String)

  def apply(): Behavior[Start] =
    Behaviors.setup { context =>
      val greeter = context.spawn(HelloWorld(), "greeter")

      Behaviors.receiveMessage { message =>
        val replyTo = context.spawn(HelloWorldBot(max = 3), message.name)
        greeter ! HelloWorld.Greet(message.name, replyTo)
        Behaviors.same
      }
    }
}
```

Actors Exercise (with Akka >= 2.6)

Constraints: You cannot use frameworks besides Akka, you need to code with Scala >= 2.13

1. Build a Chat application Akka: No Ui is needed. You need to have Messages, ChatRooms, People and people need to be able to be in multiple chat rooms at same time. Chat Rooms has a limit of 10 people.
2. Build a Chat application Akka - part 2: Enhance your chat application and now create an Admin role where a user could become an Admin and be able to KICK any user from any chat room.
3. Build a Chat application Akka - part 3: Enhance your chat application and now create observability for the system, you need to have live tracking of: Number of messages, Number of chat rooms, top 3 people who sends more message, top 3 chat rooms with more messages. Provide a print operation for the users see the metrics.
3. Build a Chat application Akka - part 4: Enhance your chat application and now Expose operations via REST interface(for this homework you can add more frameworks and libs).



MultiThreading & Actors

Diego Pacheco