AN OVERVIEW OF (A)SYNC & (NON-) BLOCKING

...or why is my web-server not responding?

WITH FUNNY FONTS!

EXPERIMENT & REPRODUCE

https://github.com/AntonFagerberg/play-performance

SYNC & BLOCKING CODE

SYNC & BLOCKING CODE

```java
public Result syncBlocking() {
  String username = getUserFromDatabaseBlocking();
  int postCount = getPostCountFromHTTPBlocking();
  int followers = getFollowerCountFromCacheBlocking();

  return Results.ok(hello(username, postCount, followers));
}
```

SYNC & BLOCKING CODE

```java
public Result syncBlocking() {
  String username = getUserFromDatabaseBlocking();
  int postCount = getPostCountFromHTTPBlocking();
  int followers = getFollowerCountFromCacheBlocking();

  return Results.ok(hello(username, postCount, followers));
}
```

200 ms

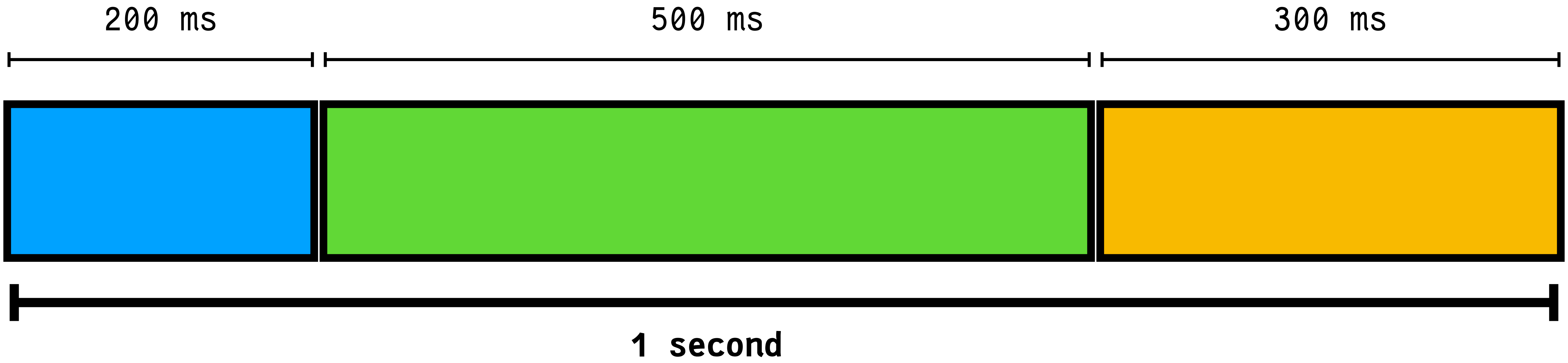
500 ms

300 ms

SYNC & BLOCKING RESPONSE TIME

```
> time curl http://localhost:9000/sync-blocking
Hello Anton, you have 100 posts and 2000 followers!
real  0m1.036s
user  0m0.008s
sys 0m0.005s
```

SYNC & BLOCKING RESPONSE TIME



HTTP LOAD TESTING

```
echo "GET http://localhost:9000/..." |
vegeta attack -rate=10 -duration=10s -timeout=3s
tee results.bin |
vegeta report
```

https://github.com/tsenart/vegeta

SYNC & BLOCKING

-rate=10 -duration=10s -timeout=3s

```
Requests      [total, rate]            100, 10.10
Duration      [total, attack, wait]    12.901466426s, 9.899999s,
3.001467426s
Latencies     [mean, 50, 95, 99, max]  2.210153366s,
2.309412543s, 3.002982981s, 3.003218662s, 3.003276145s
Bytes In      [total, mean]            3315, 33.15
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  65.00%
Status Codes  [code:count]             200:65  0:35
```

Success rate: 65%

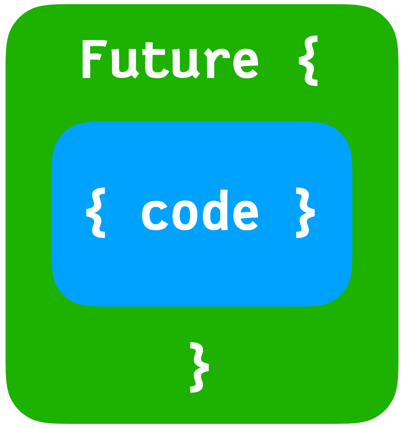
(with Play Framework default config)

ASYNC & BLOCKING CODE

A short and very incomprehensive introduction to doing computations in futures*

[*] CompletionStage, CompletableFuture, ...

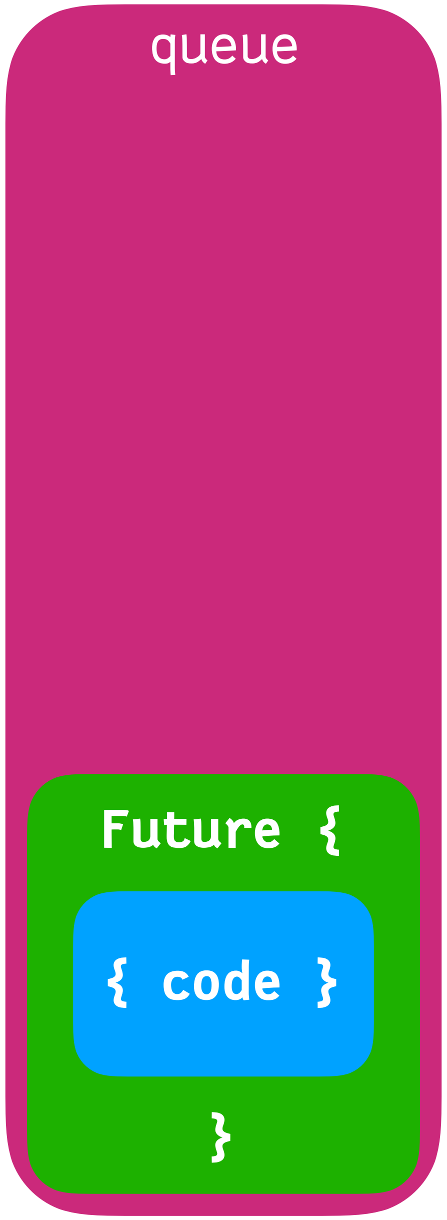
{ code }



Future {

{ code }

}

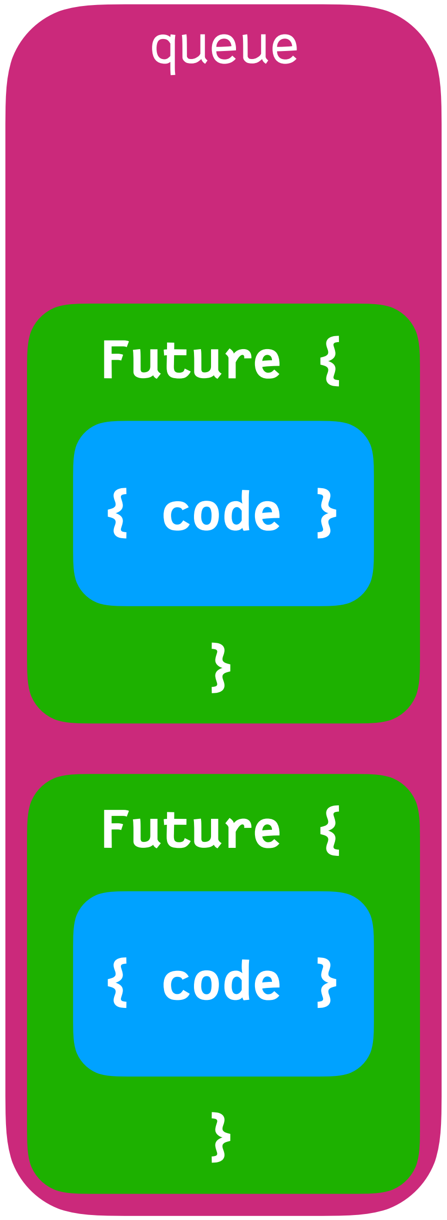


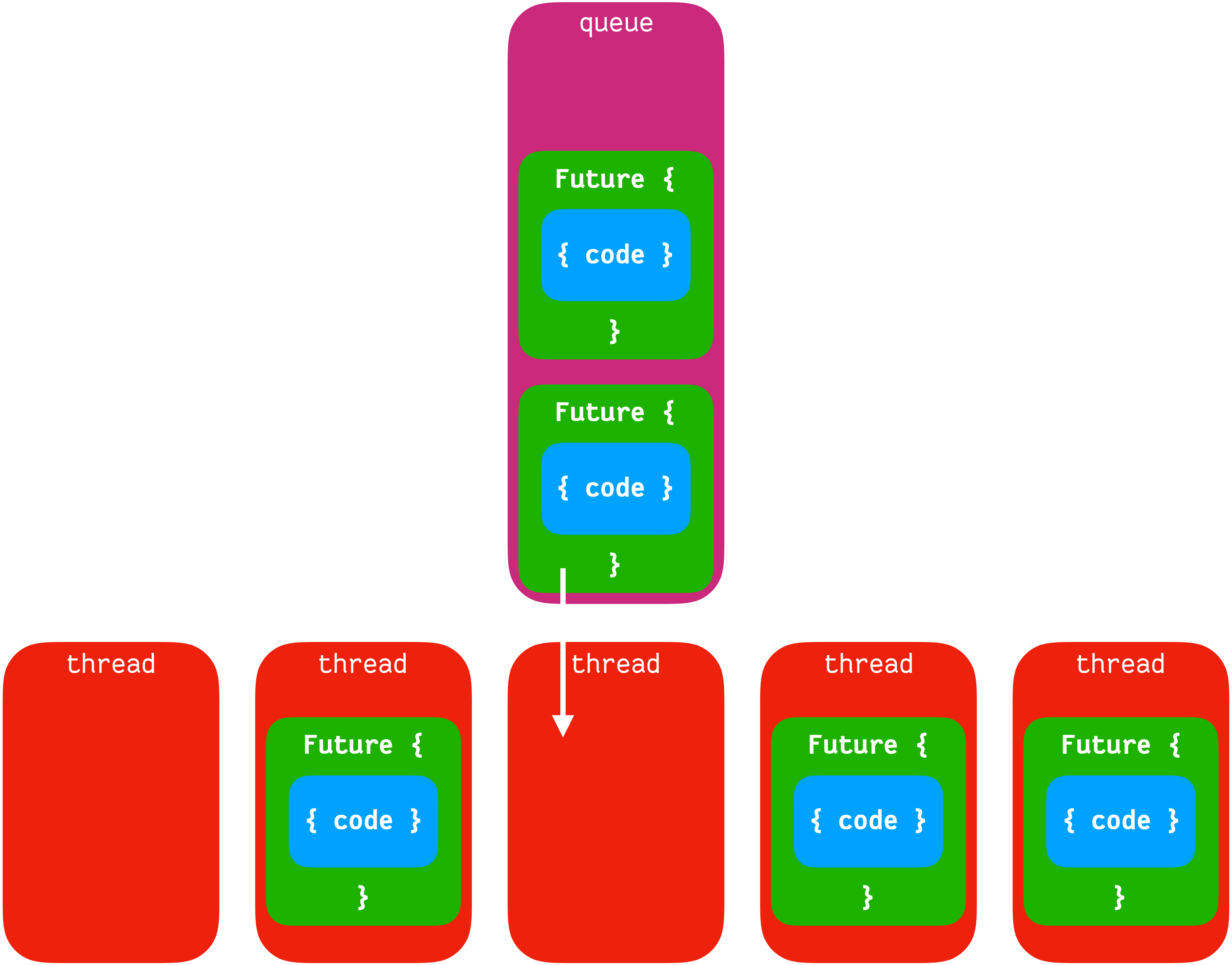
queue

Future {

{ code }

}


queue
Future {
{ code }
}
Future {
{ code }
}


queue
Future {
{ code }
}
Future {
{ code }
}
thread
thread
Future {
{ code }
}
thread
thread
Future {
{ code }
}
thread
Future {
{ code }
}

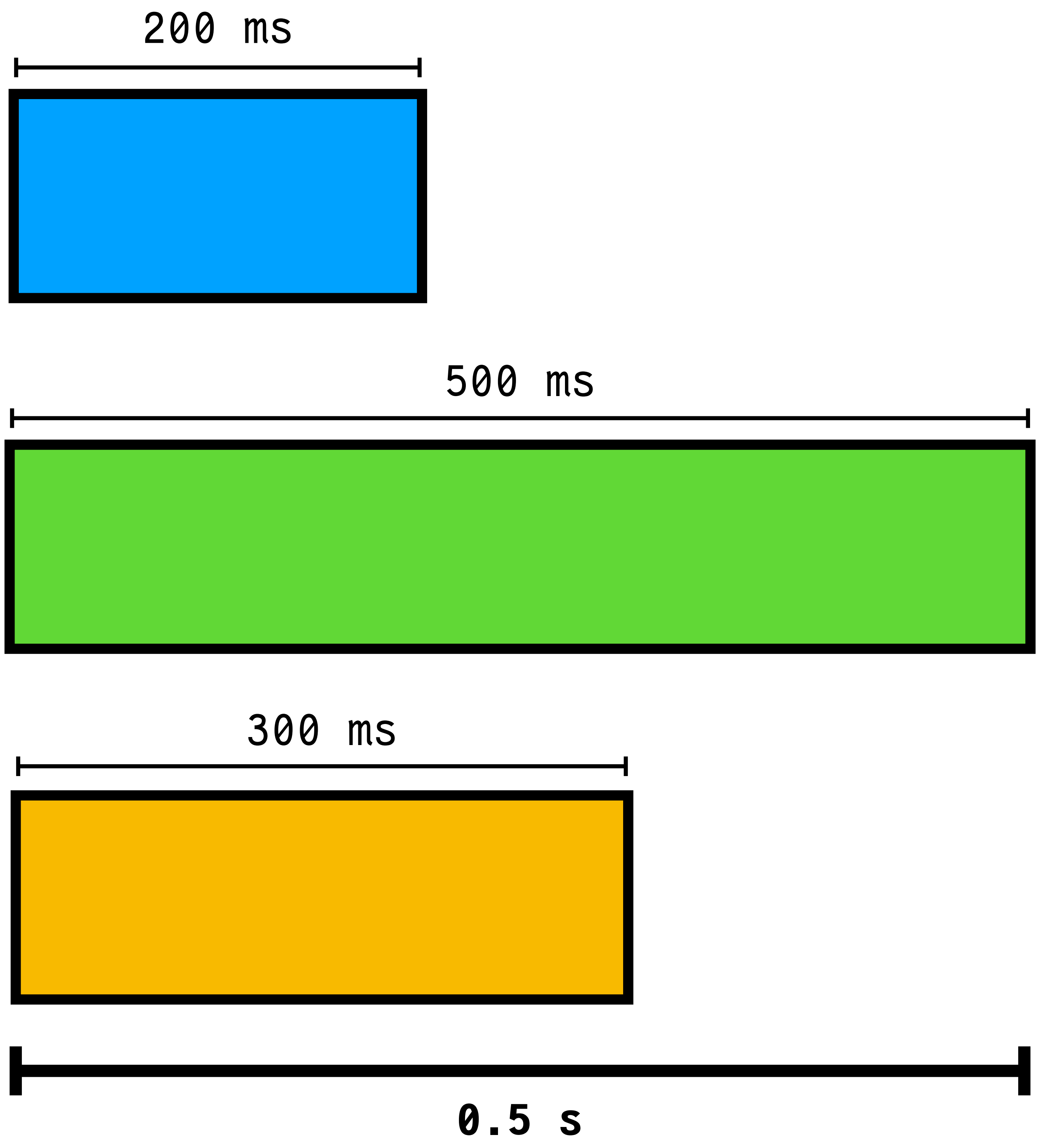
ASYNC & BLOCKING CODE

```
public CompletionStage<Result> asyncBlocking() {
  CompletionStage<String> user =
    CompletableFuture.supplyAsync(this::getUserFromDatabaseBlocking);

  CompletionStage<Integer> postCount =
    CompletableFuture.supplyAsync(this::getPostCountFromHTTPBlocking);

  CompletionStage<Integer> followerCount =
    CompletableFuture.supplyAsync(this::getFollowerCountFromCacheBlocking);

  return user.thenComposeAsync(username ->
    postCount.thenComposeAsync(posts ->
      followerCount.thenApplyAsync(followers ->
        hello(username, posts, followers)
      )
    )
  ).thenApplyAsync(Results::ok, exec);
}
```

ASYNC & BLOCKING RESPONSE TIME

```
time curl http://localhost:9000/async-blocking
Hello Anton, you have 100 posts and 2000 followers!
real  0m0.528s
user  0m0.009s
sys 0m0.006s
```

ASYNC & BLOCKING RESPONSE TIME



ASYNC & BLOCKING

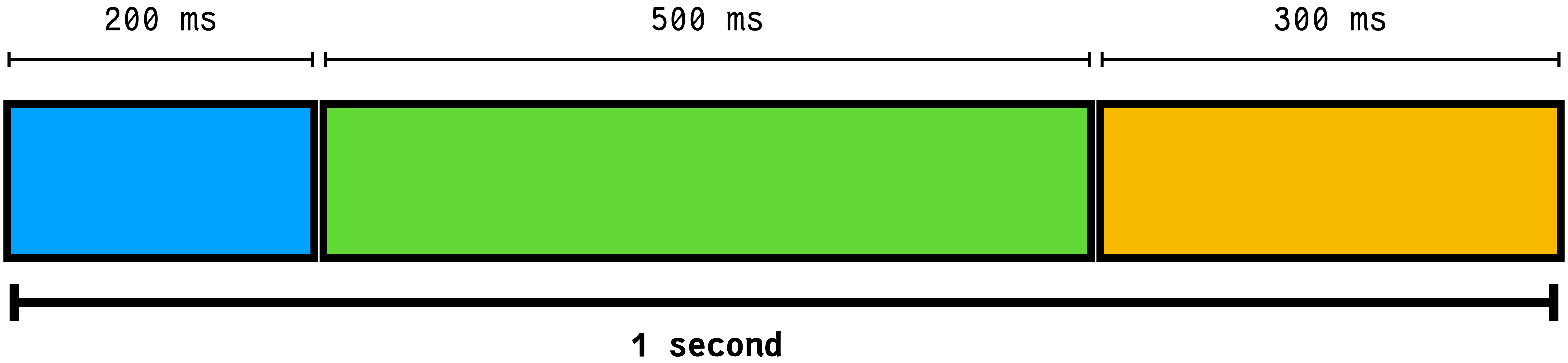
-rate=10 -duration=10s -timeout=3s

```
Requests      [total, rate]            100, 10.10
Duration      [total, attack, wait]    12.903317112s, 9.899999s, 3.003318112s
Latencies     [mean, 50, 95, 99, max]  2.835021103s, 3.002784174s,
3.003920555s, 3.004013481s, 3.004443918s
Bytes In      [total, mean]            561, 5.61
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  11.00%
Status Codes  [code:count]             200:11  0:89
```

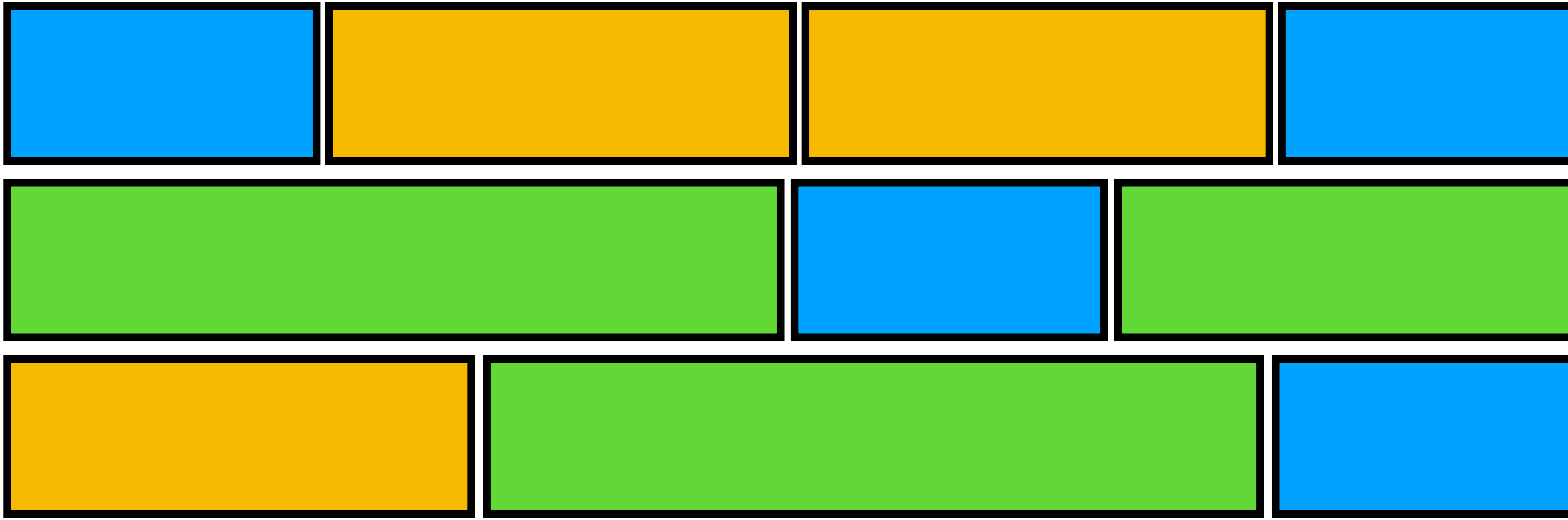
Success rate: 11%

(with Play Framework default config)

SYNC & BLOCKING RESPONSE TIME



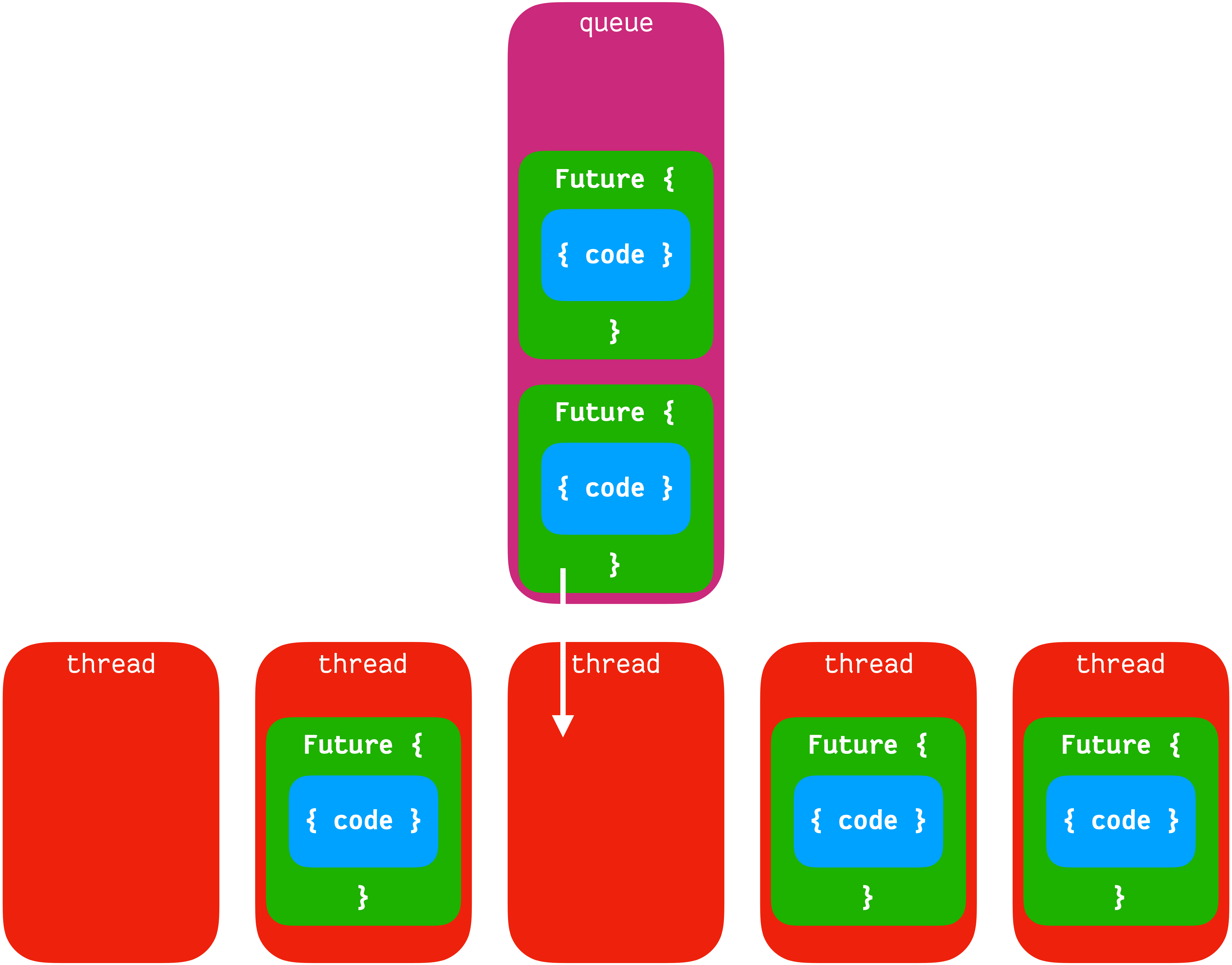
ASYNC & BLOCKING



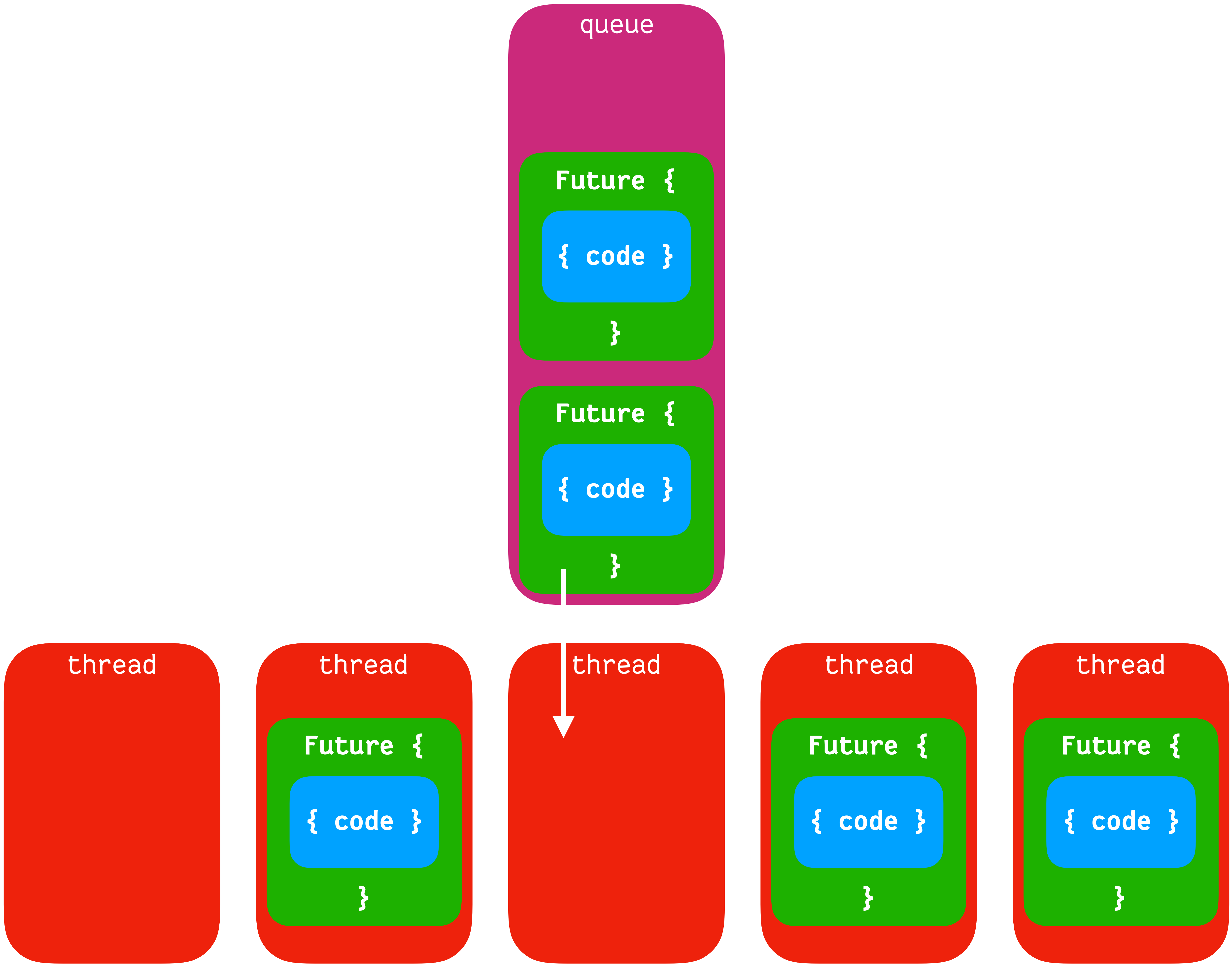
(still wasted resources)

Lesson 1: you can not make blocking code non-blocking

(without re-writing it)


queue
Future {
{ code }
}
Future {
{ code }
}
thread
thread
Future {
{ code }
}
thread
thread
Future {
{ code }
}
thread
Future {
{ code }
}

Lesson 2: a thread will execute one future until it is done



queue

Future {

{ code }

}

Future {

{ code }

}

thread

thread

Future {

{ code }

}

thread

thread

Future {

{ code }

}

thread

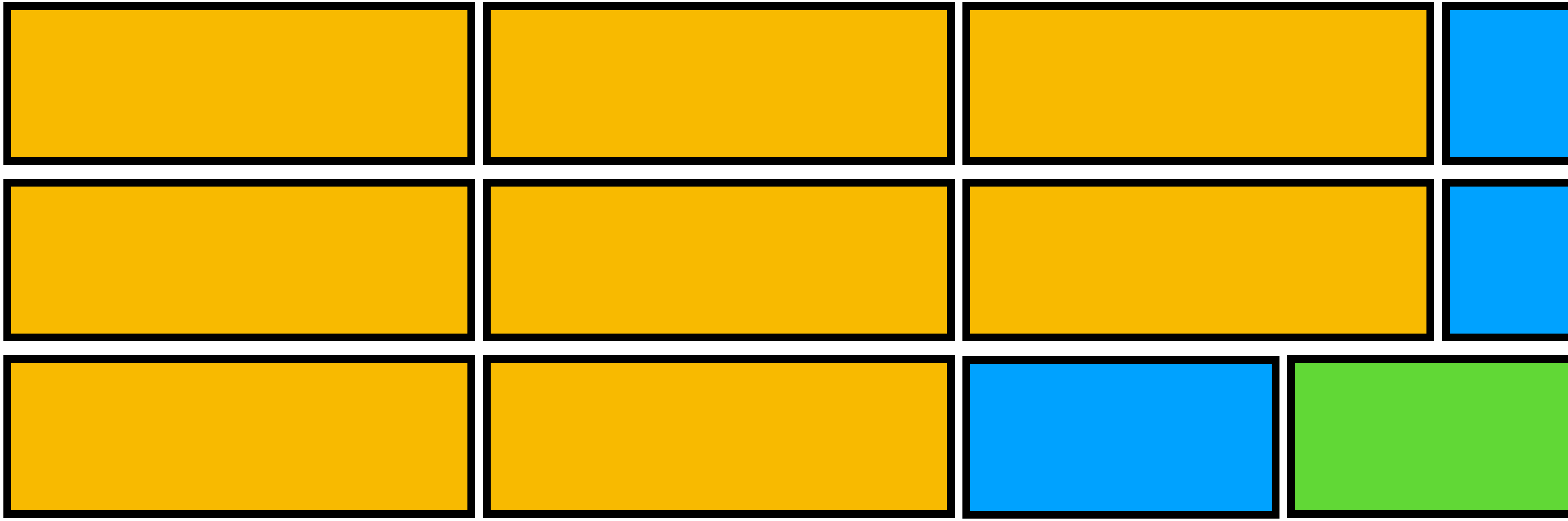
Future {

{ code }

}

Lesson 3: futures can not be cancelled

ASYNC & BLOCKING



(unfortunate scheduling)

We have just moved the blocking to other threads and shuffeled the computations around

THINGS THAT BLOCK

- Thread.sleep
- Future.get
- Await.until
- JDBC
- (Heavy computations)
- etc...

(Be ware of other side-effect calls: reading a file / HTTP call / external system and so on...)

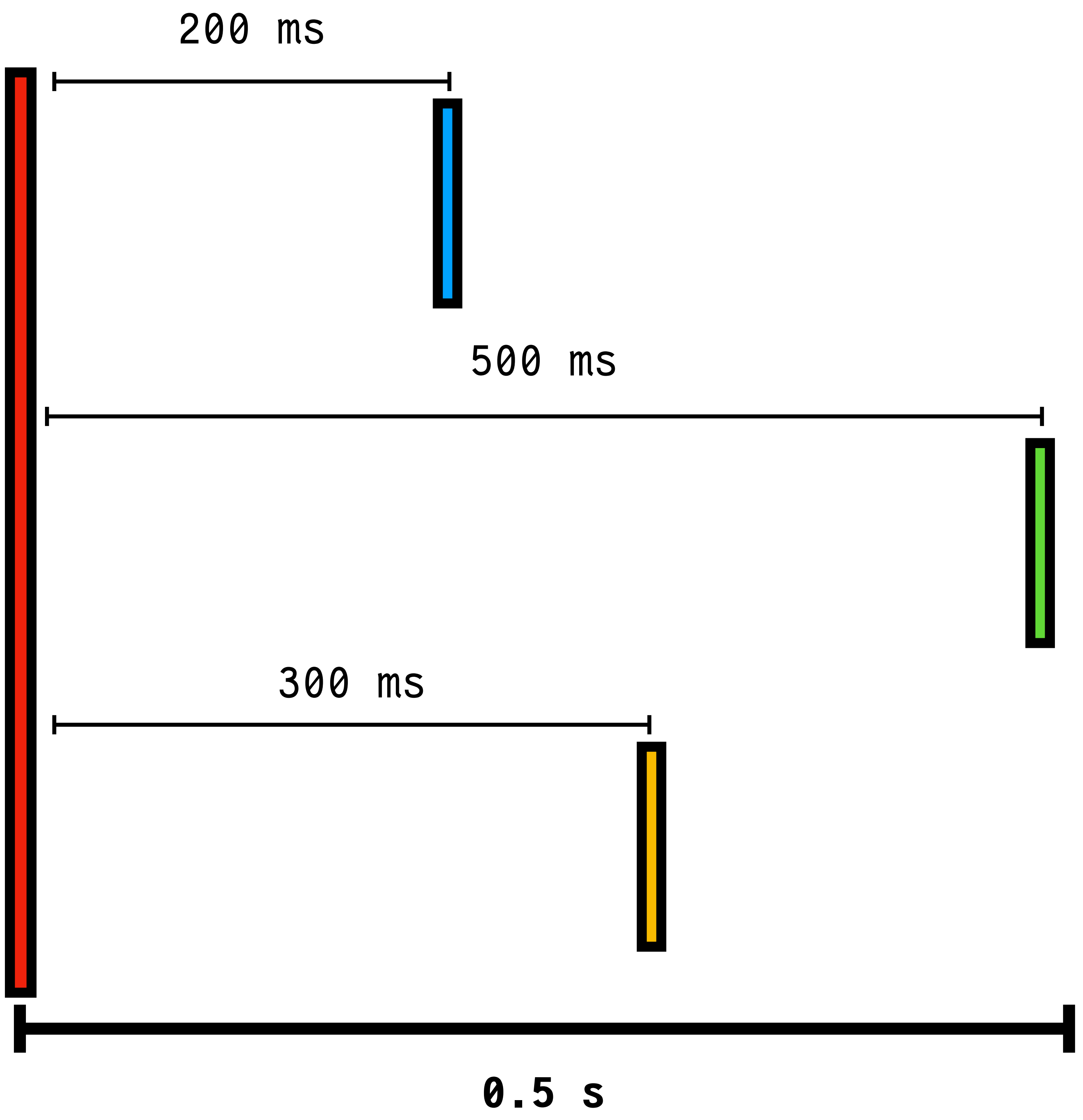
ASYNC & NON-BLOCKING CODE

```
public CompletionStage<Result> asyncNonBlocking() {
  CompletionStage<String> user =
    getUserFromDatabaseNonBlocking();

  CompletionStage<Integer> postCount =
    getPostCountFromHTTPNonBlocking();

  CompletionStage<Integer> followerCount =
    getFollowerCountFromCacheNonBlocking();

  return user.thenComposeAsync(username ->
    postCount.thenComposeAsync(posts ->
      followerCount.thenApplyAsync(followers ->
        hello(username, posts, followers)
      )
    )
  ).thenApplyAsync(Results::ok, exec);
}
```

ASYNC & BLOCKING RESPONSE TIME

```
time curl http://localhost:9000/async-non-blocking
Hello Anton, you have 100 posts and 2000 followers!
real0m0.543s
user0m0.008s
sys 0m0.006s
```

ASYNC & BLOCKING RESPONSE TIME



Lesson four: computations can be delayed without thread blocking

wait / database locks etc...

ASYNC & NONBLOCKING

```
-rate=10 -duration=10s -timeout=3s
```

```
Requests      [total, rate]             100, 10.10
Duration      [total, attack, wait]     10.416300916s, 9.899999s, 516.301916ms
Latencies     [mean, 50, 95, 99, max]  515.904367ms, 515.723494ms, 517.374616ms,
517.70094ms, 517.967775ms
Bytes In      [total, mean]             5100, 51.00
Bytes Out     [total, mean]             0, 0.00
Success       [ratio]                   100.00%
Status Codes  [code:count]              200:100
```

Success rate: 100%

(with Play Framework default config)

ASYNC & NONBLOCKING

-rate=1000 -duration=10s -timeout=3s

```
Requests      [total, rate]            10000, 1000.10
Duration      [total, attack, wait]    10.513059072s, 9.998999s, 514.060072ms
Latencies     [mean, 50, 95, 99, max]  516.168677ms, 515.920758ms, 520.816972ms,
521.963321ms, 528.467238ms
Bytes In      [total, mean]            510000, 51.00
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  100.00%
Status Codes  [code:count]             200:10000
```

Success rate: 100%

(with Play Framework default config)

DEFAULT PLAY FRAMEWORK EXECUTOR CONFIGURATION

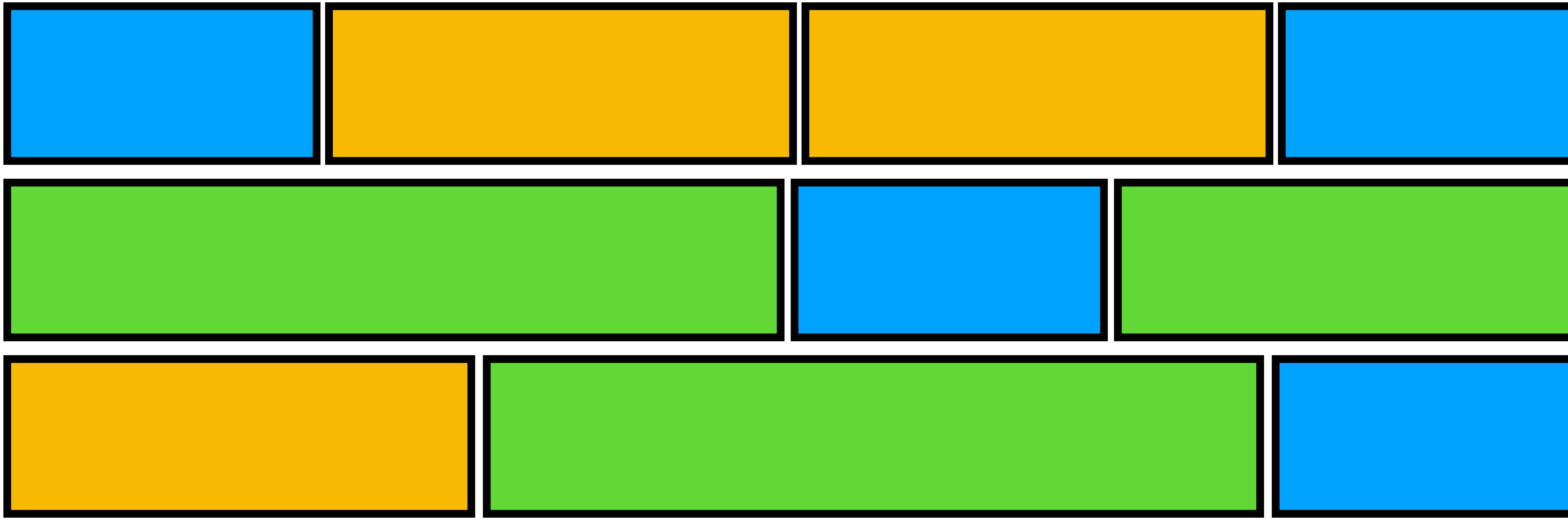
```
akka {
  actor {
    default-dispatcher {
      fork-join-executor {
        parallelism-factor = 1.0
        parallelism-max = 24
        task-peeking-mode = LIFO
      }
    }
  }
}
```

https://github.com/playframework/playframework/issues/7242

MY LAPTOP

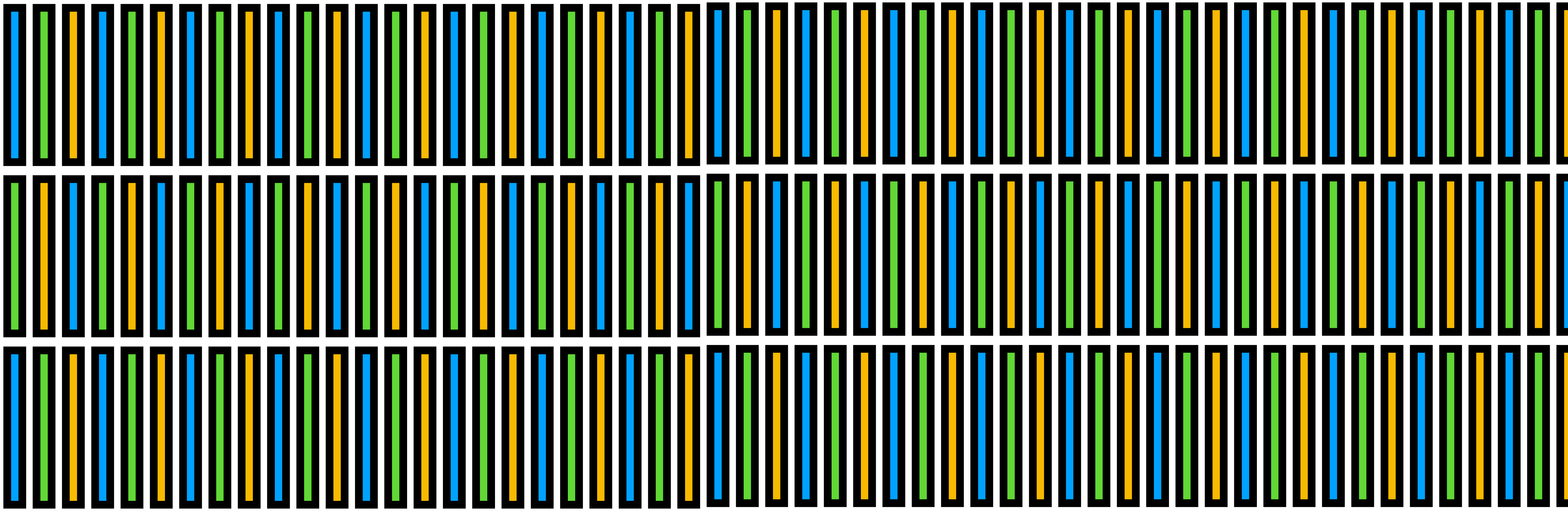
- 4 cores + hyper-threading = 8 threads
- Blocking sync code example
- 9 concurrent visitors
- Entire app will block for 1 second for one of the visitor
- Aggressive non-blocking thread pool configuration

ASYNC & BLOCKING



(wasted resources)

ASYNC & NONBLOCKING



WHAT ABOUT SERVLETS?

- One thread per request
- Hundreds of threads

SYNC & BLOCKING WITH 1000 THREADS

```
-rate=1000 -duration=10s -timeout=3s
```

```
Requests      [total, rate]            10000, 1000.10
Duration      [total, attack, wait]    11.061748299s, 9.998999s, 1.062749299s
Latencies     [mean, 50, 95, 99, max]  1.030605789s, 1.011834459s, 1.057429836s,
1.538313379s, 3.000731805s
Bytes In      [total, mean]            509847, 50.98
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  99.97%
Status Codes  [code:count]             200:9997  0:3
```

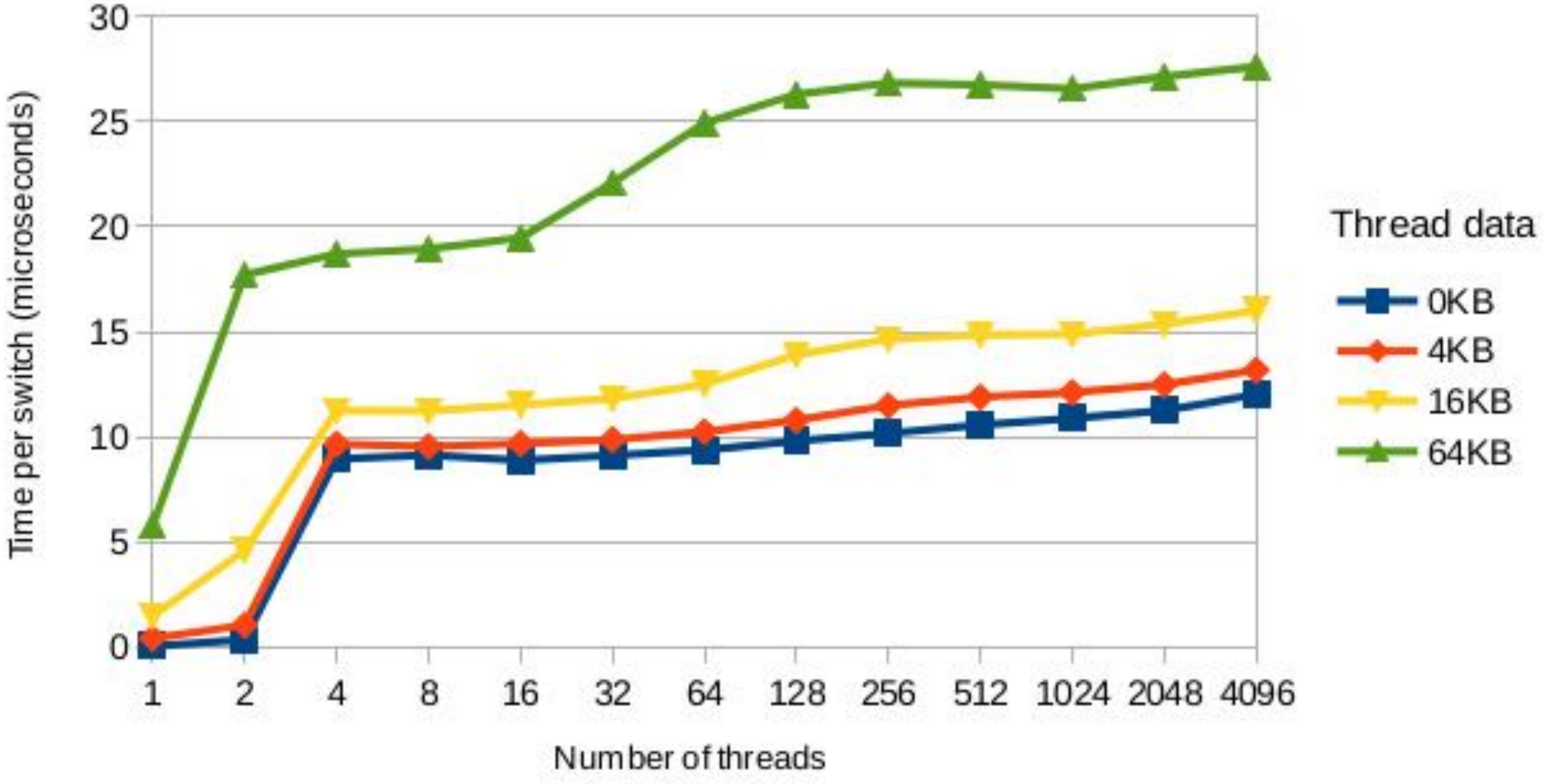
Success rate: 99.97%

BUT SOONER OR LATER...

```
java.lang.OutOfMemoryError: unable to create new native thread
    at java.lang.Thread.start0(Native Method)
    at java.lang.Thread.start(Thread.java:714)
    at io.netty.util.concurrent.SingleThreadEventExecutor.shutdownGracefully(SingleThreadEventExecutor.java:587)
    at io.netty.util.concurrent.MultithreadEventExecutorGroup.shutdownGracefully(MultithreadEventExecutorGroup.java:146)
    at org.asynchttpclient.netty.channel.ChannelManager.close(ChannelManager.java:365)
    at org.asynchttpclient.DefaultAsyncHttpClient.close(DefaultAsyncHttpClient.java:96)
    at play.libs.ws.ahc.AhcWSClient.close(AhcWSClient.java:43)
    at play.libs.ws.ahc.AhcWSAPI.lambda$new$1(AhcWSAPI.java:32)
    at play.libs.ws.ahc.AhcWSAPI$$Lambda$2/131096911.call(Unknown Source)
    at play.api.inject.ApplicationLifecycle$$anonfun$addStopHook$1.apply(ApplicationLifecycle.scala:67)
```

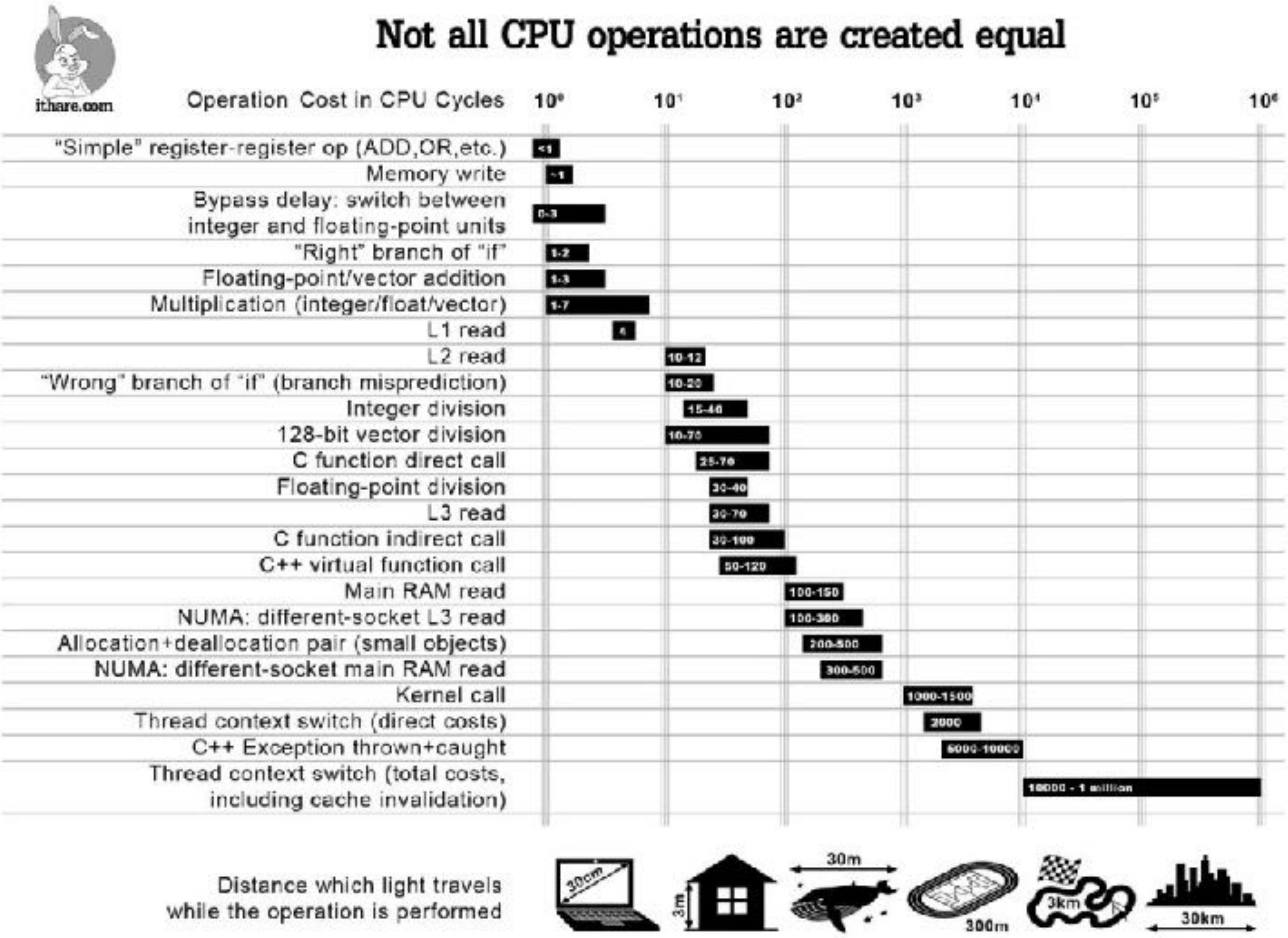
BLOCKING VS NON-BLOCKING PERFORMANCE

THREAD CONTEXT SWITCH



https://www.ibm.com/developerworks/library/j-jvmc3/index.html

THREAD CONTEXT SWITCH



https://manuel.bernhardt.io/2017/05/15/akka-anti-patterns-blocking/

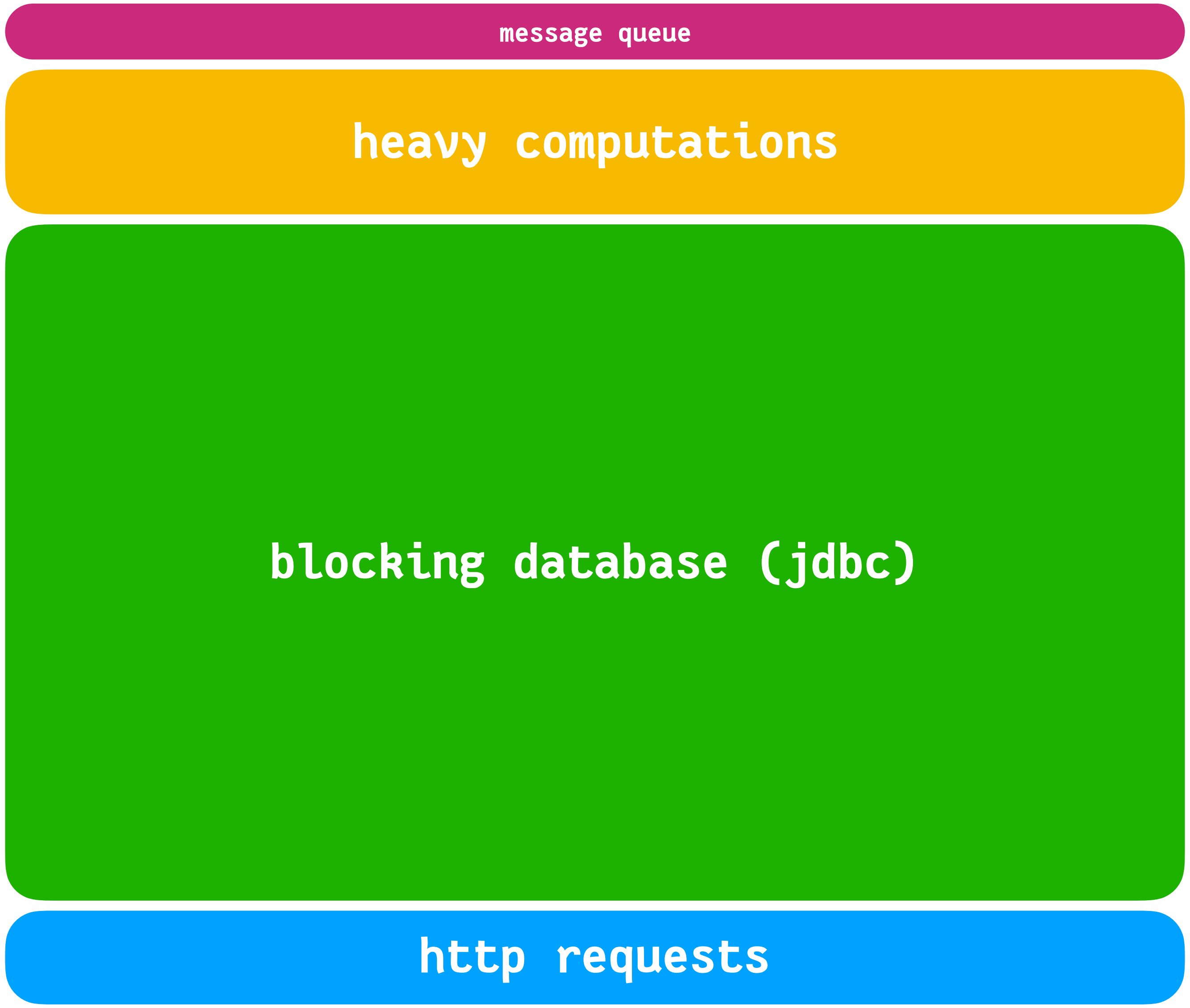
BLOCKING VS NON-BLOCKING PERFORMANCE

- Wasted resources
- Async behaviors
- Incoming requests limited

WHAT TO DO WHEN WE MUST BOCK?

- Blocking APIs
- Heavy computations

DEDICATED THREAD POOLS



message queue

heavy computations

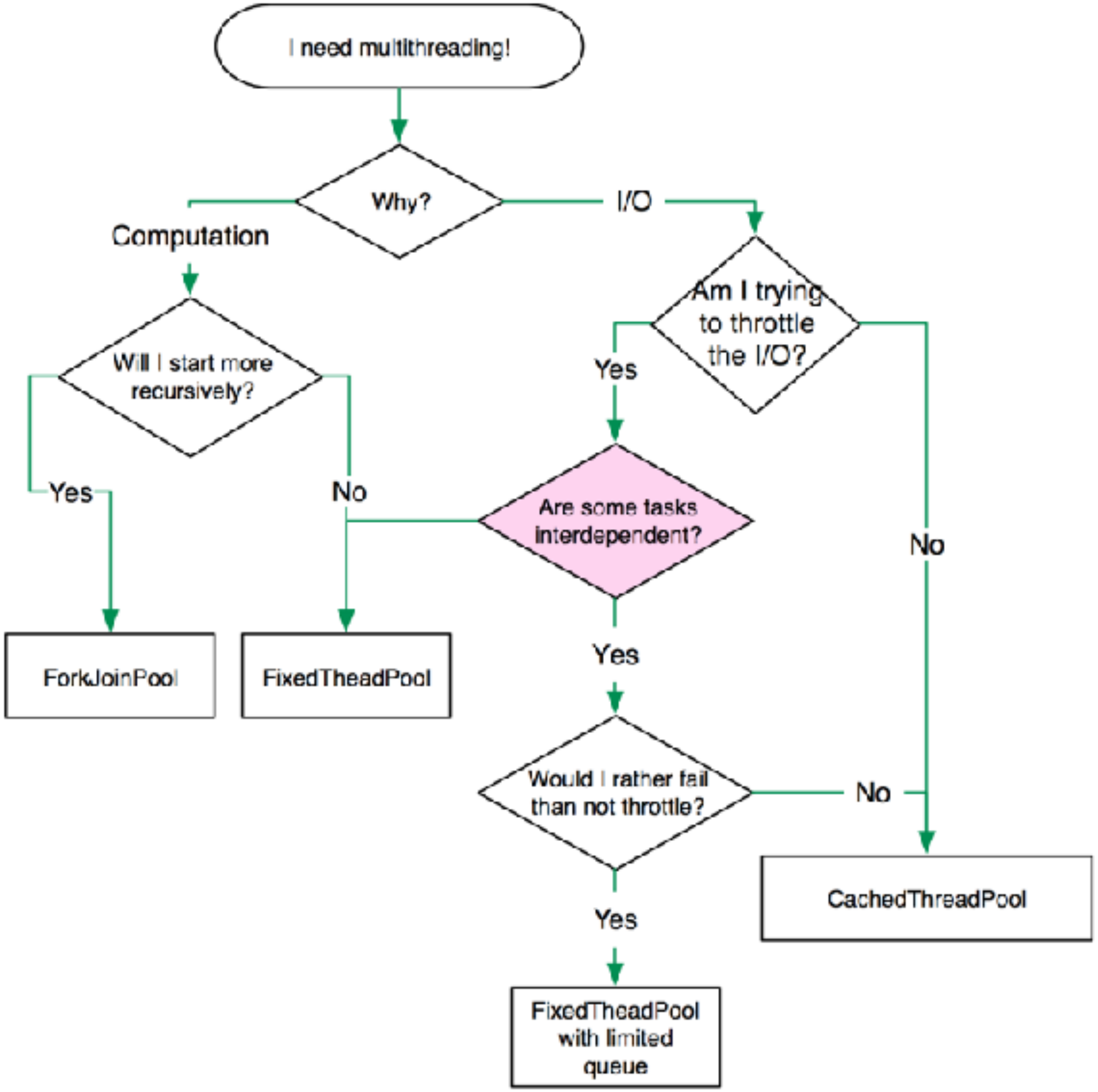
blocking database (jdbc)

http requests

EXECUTOR / EXECUTION CONTEXT

```java
private MyExecutionContext myExecutionContext;

@Inject
public Application(MyExecutionContext myExecutionContext) {
  this.myExecutionContext = myExecutionContext;
}

public CompletionStage<Result> index() {
  // Wrap an existing thread pool, using the context
  // from the current thread
  Executor myEc = HttpExecution.fromThread(
    (Executor) myExecutionContext
  );

  return supplyAsync(() ->
    intensiveComputation(), myEc
  ).thenApplyAsync(i -> ok("Got result: " + i), myEc);
}

public int intensiveComputation() { return 2;}
```

CHOOSING AN EXECUTORSERVICE


I need multithreading!
Why?
Computation
I/O
Will I start more recursively?
Yes
No
Am I trying to throttle the I/O?
Yes
No
Are some tasks interdependent?
Yes
Would I rather fail than not throttle?
No
Yes
ForkJoinPool
FixedTheadPool
CachedThreadPool
FixedTheadPool with limited queue

http://blog.jessitron.com/2014/01/choosing-executorservice.html

DEADLOCKS

BLOCKING (SCALA)

```
Future {
  blocking {
    ...
  }
}
```

if needed, add extra threads

(doesn’t work with all execution contexts)

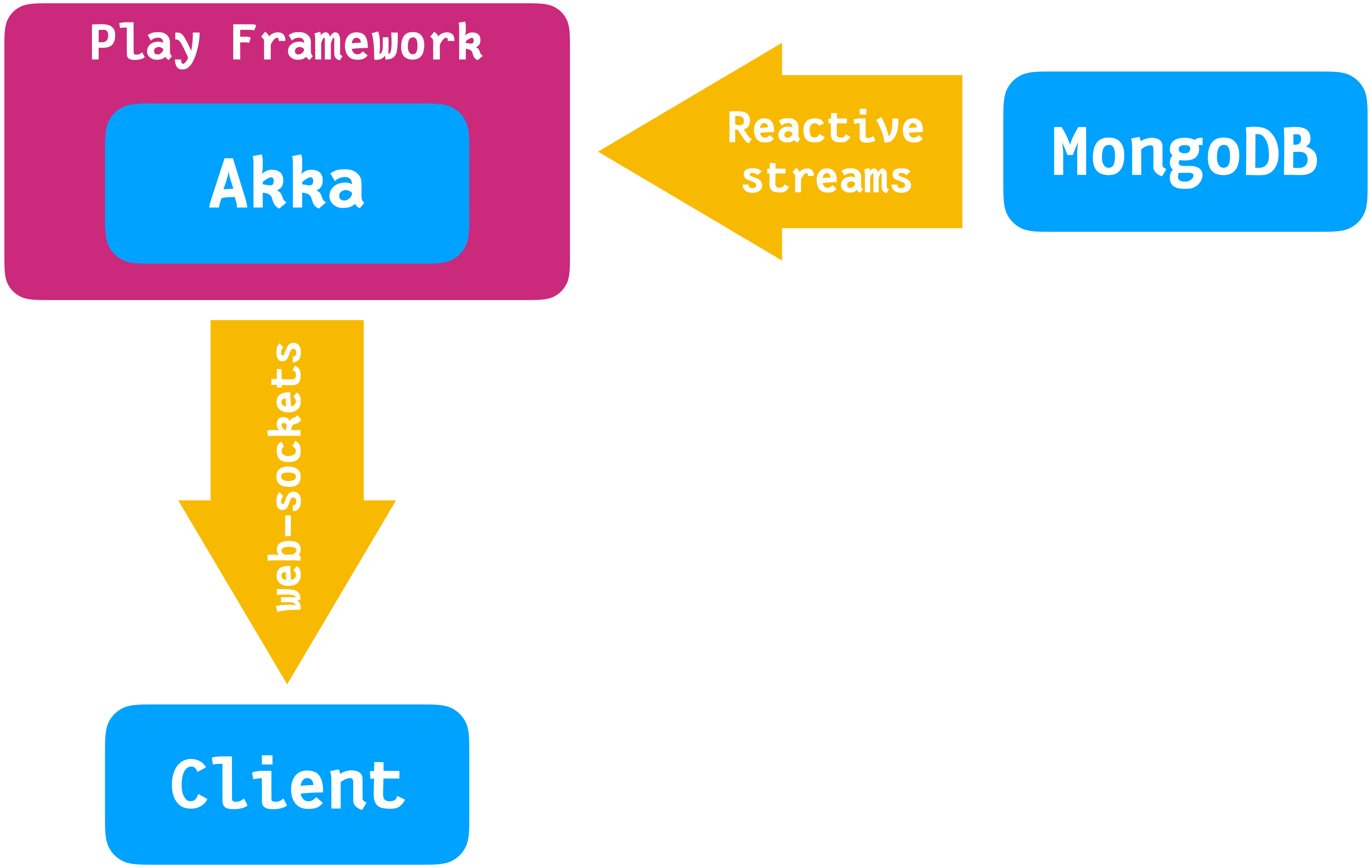
OTHER CONCURRENCY ABSTRACTIONS

MONGODB (REACTIVE STREAMS)

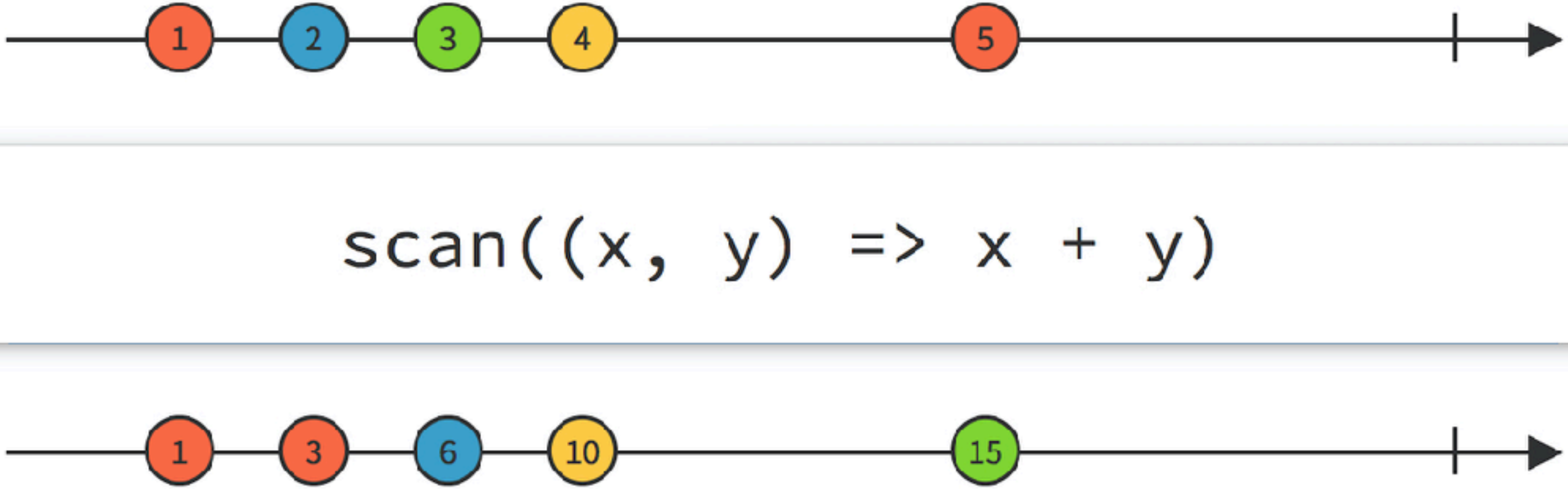
```
publisher.subscribe(new Subscriber<T>() {

    @Override
    public void onNext(final T thing) {
        -
    }

    @Override
    public void onError(final Throwable t) {
        -
    }

    @Override
    public void onComplete() {
        -
    }

});
```

WHAT I'VE USED REACTIVESTREAMS FOR

- (No real good use case)
- "Classic HTTP" removes much benefit
- No big data sets
- Usually converted to futures


Play Framework
Akka
Reactive streams
MongoDB
web-sockets
Client

REACTIVEX - GOOD DIAGRAMS



```
scan((x, y) => x + y)
```

http://reactivex.io

AKKA (ACTORS)

```java
public class WellStructuredActor extends AbstractActor {

  public static class Msg1 {}
  public static class Msg2 {}
  public static class Msg3 {}

  @Override
  public Receive createReceive() {
    return receiveBuilder()
      .match(Msg1.class, this::receiveMsg1)
      .match(Msg2.class, this::receiveMsg2)
      .match(Msg3.class, this::receiveMsg3)
      .build();
  }

  private void receiveMsg1(Msg1 msg) {
    // actual work
  }

  private void receiveMsg2(Msg2 msg) {
    // actual work
  }

  private void receiveMsg3(Msg3 msg) {
    // actual work
  }

}
```

WHAT I'VE USED AKKA (ACTORS) FOR

- Polling external systems
- Code that needs retries
- Scheduling — future / periodic
- Web-sockets (server side)
- (Elixir / Erlang)

FINAL THOUGHT

> "If everything feels like it is getting more complicated, that means you are understanding the problem better."
>
> —Anil Dash