Go Reactive

Event-Driven,
Scalable,
Resilient &
Responsive
Systems

Jonas Bonér

CTO Typesafe

@jboner

New Tools for a New Era

- The demands and expectations for applications have changed dramatically in recent years

New Tools for a New Era

- The demands and expectations for applications have changed dramatically in recent years
- We need to write applications that manages
  1. Mobile devices
  2. Multicore architectures
  3. Cloud computing environments

New Tools for a New Era

- The demands and expectations for applications have changed dramatically in recent years
- We need to write applications that manages
  1. Mobile devices
  2. Multicore architectures
  3. Cloud computing environments
- Deliver applications that are
  1. Interactive & Real-time
  2. Responsive
  3. Collaborative

Typesafe

New Tools for a New Era

New Tools for a New Era

We to need to build systems that:

New Tools for a New Era

We to need to build systems that:

- *react* to events — **Event-Driven**

Typesafe

New Tools for a New Era

We to need to build systems that:

- *react* to events — **Event-Driven**
- *react* to load — **Scalable**

Typesafe

New Tools for a New Era

We to need to build systems that:

- *react* to events — **Event-Driven**
- *react* to load — **Scalable**
- *react* to failure — **Resilient**

Typesafe

New Tools for a New Era

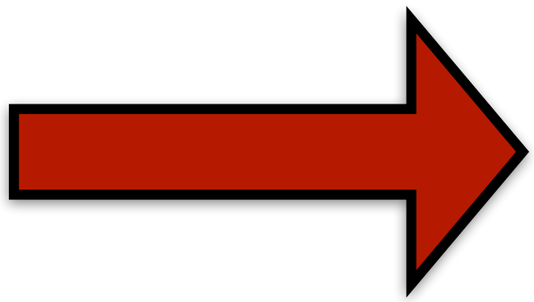
We to need to build systems that:

- *react* to events — **Event-Driven**
- *react* to load — **Scalable**
- *react* to failure — **Resilient**
- *react* to users — **Responsive**

New Tools for a New Era

We to need to build systems that:

- *react* to events — **Event-Driven**
- *react* to load — **Scalable**
- *react* to failure — **Resilient**
- *react* to users — **Responsive**



Reactive Applications

Typesafe

Reactive

"Readily responsive to a stimulus"

- Merriam Webster

The four traits of Reactive

Responsive

Scalable

Resilient

Event-Driven

Typesafe

Event-Driven

"The flow of the program is determined by events"

- Wikipedia

Shared mutable state

Shared mutable state

Together with threads...

Shared mutable state

Together with threads...

Shared mutable state

Together with threads...

...code that is totally non-deterministic

Shared mutable state

Together with threads...

...code that is totally non-deterministic

...and the root of all EVIL

Shared mutable state

Together with threads...

...code that is totally non-deterministic

...and the root of all EVIL

Please, avoid it at all cost

Typesafe

Shared mutable state

Together with threads...

...code that is totally non-deterministic

...and the root of all EVIL

Please, avoid it at all cost

Typesafe

1. Never block

Typesafe

1. Never block

- ...unless you really have to
- Blocking kills scalability (& performance)
- Never sit on resources you don't use
- Use non-blocking IO
- Use lock-free concurrency

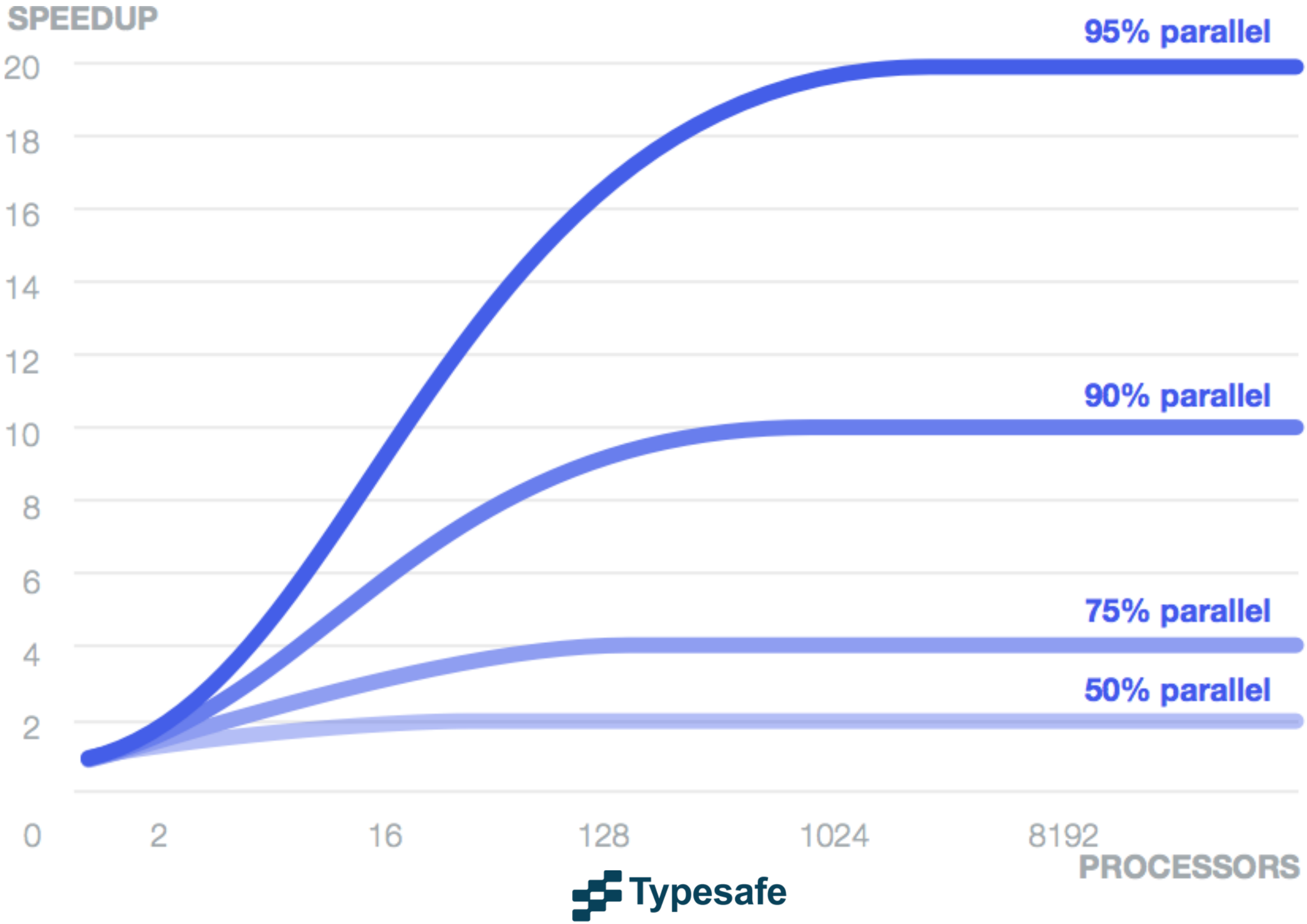
Typesafe

2. Go Async

Design for reactive event-driven systems

- Use asynchronous event/message passing
- Think in workflow, how the events flow in the system
- Gives you
  1. lower latency
  2. better throughput
  3. a more loosely coupled architecture, easier to extend, evolve & maintain

Amdahl's Law

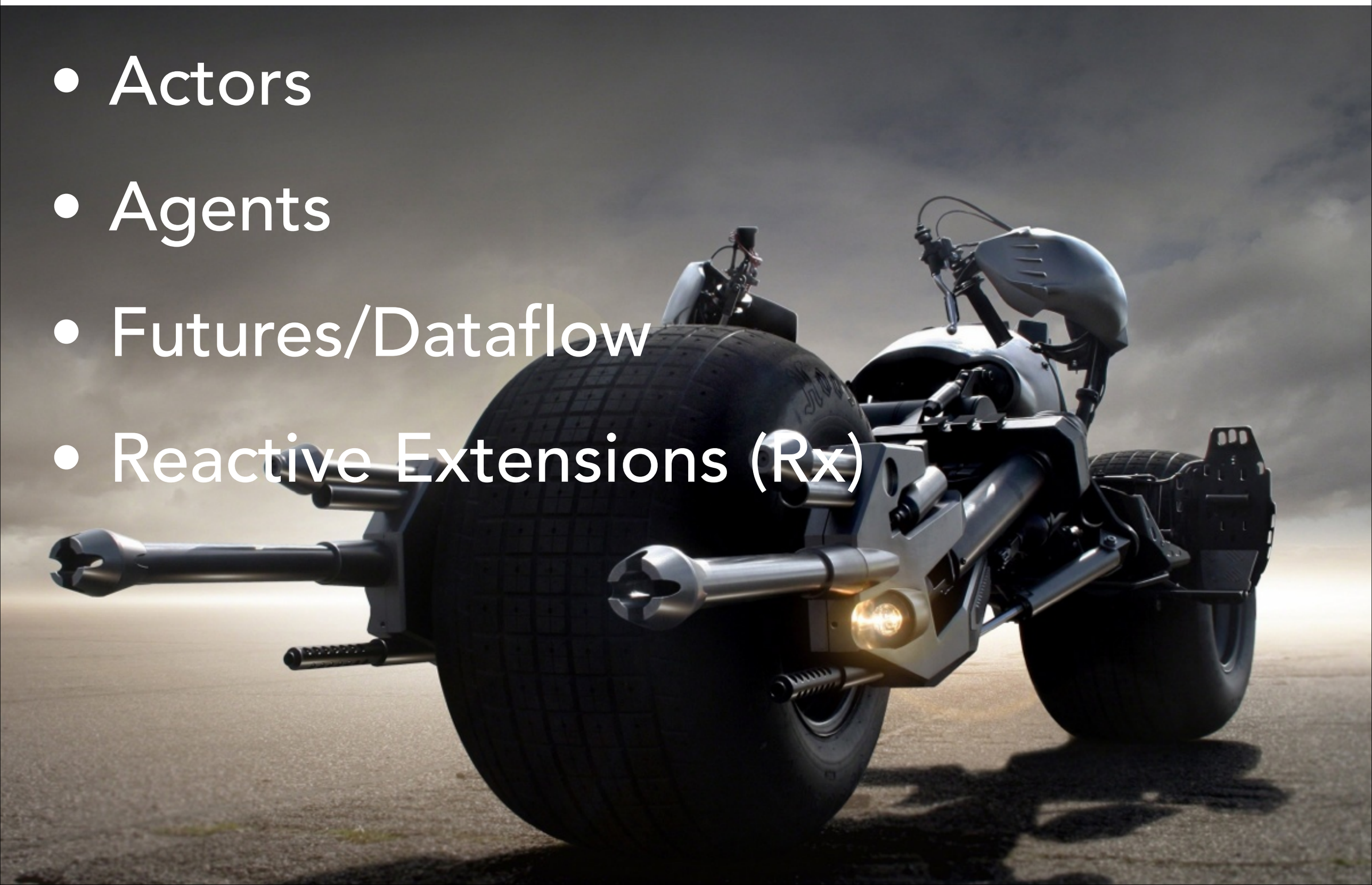


Typesafe

Amdahl's Law


SPEEDUP
20
18
16
14
12
10
8
6
4
2
0
2
16
128
1024
8192
PROCESSORS
parallel
75% parallel
50% parallel
Needs to be
Reactive from
ToP to BoTToM
Typesafe

You deserve better tools



- Actors
- Agents
- Futures/Dataflow
- Reactive Extensions (Rx)

Actors

- Share NOTHING
- Isolated lightweight event-based processes
- Each actor has a mailbox (message queue)
- Communicates through asynchronous & non-blocking message passing
- Location transparent (distributable)
- Supervision-based failure management
- Examples: Akka & Erlang

Actors in Akka

```
public class Greeting implements Serializable {
 public final String who;
 public Greeting(String who) { this.who = who; }
}

public class GreetingActor extends UntypedActor {

  public void onReceive(Object message) {
   if (message instanceof Greeting)
    println("Hello " + ((Greeting) message).who);
 }
}
```

Typesafe

Actors in Akka

Define the message(s) the Actor should be able to respond to

```
public class Greeting implements Serializable {
 public final String who;
 public Greeting(String who) { this.who = who; }
}

public class GreetingActor extends UntypedActor {

  public void onReceive(Object message) {
   if (message instanceof Greeting)
    println("Hello " + ((Greeting) message).who);
 }
}
```

Typesafe

Actors in Akka

Define the message(s) the Actor should be able to respond to

Define the Actor class

```
public class Greeting implements Serializable {
 public final String who;
 public Greeting(String who) { this.who = who; }
}

public class GreetingActor extends UntypedActor {

  public void onReceive(Object message) {
   if (message instanceof Greeting)
    println("Hello " + ((Greeting) message).who);
  }
}
```

Actors in Akka

Define the message(s) the Actor should be able to respond to

```
public class Greeting implements Serializable {
  public final String who;
  public Greeting(String who) { this.who = who; }
}

public class GreetingActor extends UntypedActor {

  public void onReceive(Object message) {
    if (message instanceof Greeting)
      println("Hello " + ((Greeting) message).who);
  }
}
```

Define the Actor class

Define the Actor's behavior

Agents

- Reactive memory cells
- Send a update function to the Agent, which

  1. adds it to an (ordered) queue, to be

  2. applied to the Agent async & non-blocking

- Reads are “free”, just dereferences the Ref
- Composes nicely
- Examples: Clojure & Akka

Agents in Akka

```
val agent = Agent(5)
agent send (x => x + 1)
agent send (x => x * 2)
```

Futures/Dataflow

- Allows you to spawn concurrent computations and work with the not yet computed results
- Write-once, Read-many
- Freely sharable
- Allows non-blocking composition
- Monadic (composes in for-comprehensions)
- Build in model for managing failure

Futures in Scala

```
val result1 = future { ... }
val result2 = future { ... }
val result3 = future { ... }

val sum = for {
   r1 <- result1
   r2 <- result2
   r3 <- result3
} yield { r1 + r2 + r3 }
```

Typesafe

Reactive Extensions (Rx)

- Extend Futures with the concept of a Stream
- Composable in a type-safe way
- Event-based & asynchronous
- Observable ⇛ Push Collections
  - onNext(T), onError(E), onCompleted()
  - Compared to Iterable ⇛ Pull Collections
- Examples: Rx.NET, RxJava, RxJS etc.

RxJava

```
getDataFromNetwork()
   .skip(10)
   .take(5)
   .map({ s -> return s + " transformed" })
   .subscribe({ println "onNext => " + it })
```

Typesafe

Scalable

"Capable of being easily expanded or upgraded on demand"

- *Merriam Webster*

Distributed systems is the

new normal

Typesafe

Distributed systems is the

new normal

You already have a distributed system,

whether you want it or not

Typesafe

Distributed systems is the

new normal

You already have a distributed system,

whether you want it or not

Mobile

SQL Replication

NOSQL DBs

Cloud Services

Typesafe

What is the essence of distributed computing?

What is the essence of distributed computing?

It's to try to overcome that

1. Information travels at the speed of light
2. Independent things fail independently

Why do we need it?

Why do we need it?

Scalability

When you outgrow
the resources of
a single node

Why do we need it?

Scalability

When you outgrow the resources of a single node

Availability

Providing resilience if one node fails

Why do we need it?

Scalability

When you outgrow
the resources of
a single node

Availability

Providing resilience if
one node fails

Rich stateful clients

The problem?

The problem?

It is still

Very Hard

Typesafe

No difference

Between a

Slow node

and a

Dead node

Typesafe

The network is

Inherently Unreliable

Typesafe

The network is

Inherently Unreliable

http://aphyr.com/posts/288-the-network-is-reliable

Typesafe

Fallacies

Peter Deutsch's

8 Fallacies

of Distributed Computing

Typesafe

Fallacies

Peter Deutsch's

8 Fallacies of Distributed Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Typesafe

Graveyard of distributed systems

- Distributed Shared Mutable State
  - $EVIL^{N}$ (where N is number of nodes)

Graveyard of distributed systems

- Distributed Shared Mutable State
  - $EVIL^N$ (where N is number of nodes)
- Serializable Distributed Transactions

Typesafe

Graveyard of distributed systems

- Distributed Shared Mutable State
  - $EVIL^N$ (where N is number of nodes)
- Serializable Distributed Transactions
- Synchronous RPC

Graveyard of distributed systems

- Distributed Shared Mutable State
  - $EVIL^{N}$ (where N is number of nodes)
- Serializable Distributed Transactions
- Synchronous RPC
- Guaranteed Delivery

Graveyard of distributed systems

- Distributed Shared Mutable State
  - EVIL^{N} (where N is number of nodes)
- Serializable Distributed Transactions
- Synchronous RPC
- Guaranteed Delivery
- Distributed Objects

Typesafe

Graveyard of distributed systems

- Distributed Shared Mutable State
  - EVIL^{N} (where N is number of nodes)
- Serializable Distributed Transactions
- Synchronous RPC
- Guaranteed Delivery
- Distributed Objects
  - *"Sucks like an inverted hurricane" - Martin Fowler*

Typesafe

Instead

Instead

Embrace the Network

Use

Asynchronous

Message

Passing

and be done with it

Typesafe

We need

Location Transparency

Typesafe

What is

Location Transparency?

```
// send message to local actor
ActorRef localGreeter = system.actorOf(
    new Props(GreetingActor.class), “greeter");

localGreeter.tell(“Jonas”);
```

What is

Location Transparency?

```
// send message to local actor
ActorRef localGreeter = system.actorOf(
    new Props(GreetingActor.class), “greeter");

localGreeter.tell(“Jonas”);
```

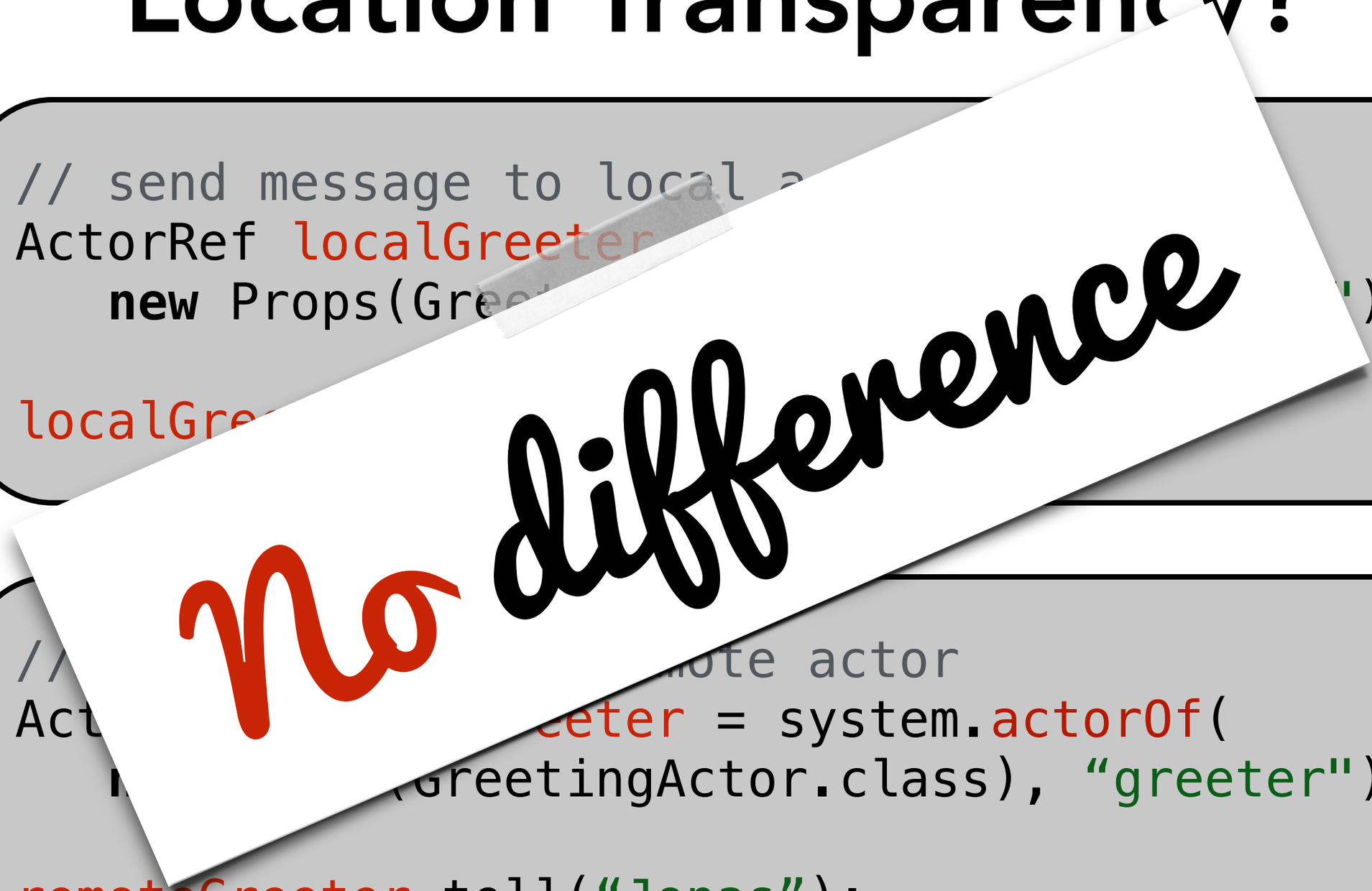
```
// send message to remote actor
ActorRef remoteGreeter = system.actorOf(
    new Props(GreetingActor.class), “greeter");

remoteGreeter.tell(“Jonas”);
```

What is Location Transparency?

```
// send message to local a
ActorRef localGreeter
    new Props(Gre                    ");

localGre
```




```
//                    mote actor
Act          eeter = system.actorOf(
    n          (GreetingActor.class), “greeter");

remoteGreeter.tell(“Jonas”);
```

Typesafe

Divide & Conquer

Partition

for scale

Replicate

for resilience

Typesafe


Typesafe

Share
Nothing

Share

Nothing

Asynchronous

Communication

Share
Nothing

Loose
Coupling

Asynchronous
Communication

Typesafe

Share
Nothing

Loose
Coupling

Asynchronous
Communication

Location
Transparency

Share
Nothing

Loose
Coupling

Asynchronous
Communication

Location
Transparency

No limit to scalability

Typesafe

Share
Nothing

Loose
Coupling

Asynchronous
Communication

Location
Transparency

close to at least

No limit to scalability

Typesafe

Resilience

"The ability of a substance or object to spring back into shape."

"The capacity to recover quickly from difficulties."

- *Merriam Webster*

Failure Recovery in Java/C/C# etc.

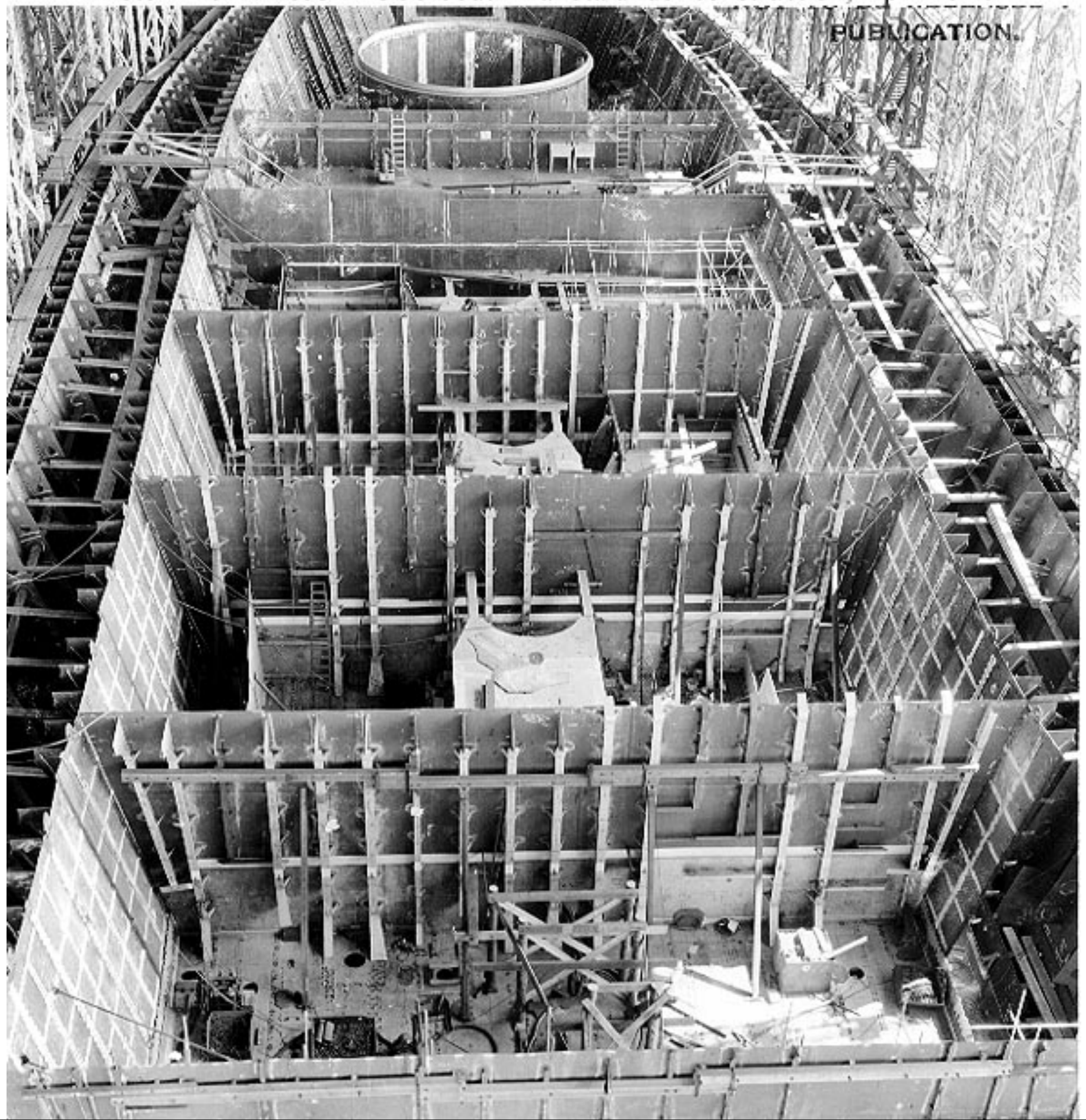
Failure Recovery in Java/C/C# etc.

- You are given a SINGLE thread of control
- If this thread blows up you are screwed
- So you need to do all explicit error handling WITHIN this single thread
- To make things worse - errors do not propagate between threads so there is NO WAY OF EVEN FINDING OUT that something have failed
- This leads to DEFENSIVE programming with:
  - Error handling TANGLED with business logic
  - SCATTERED all over the code base

Failure Recovery in Java/C/C# etc.

- You are given a SINGLE th
- If this thre
- Th
  - ENSIVE programming with:
    - Error handling TANGLED with business logic
    - SCATTERED all over the code base

Photo # 19-N-28532 USS South Dakota under construction, April 1940



The Right Way

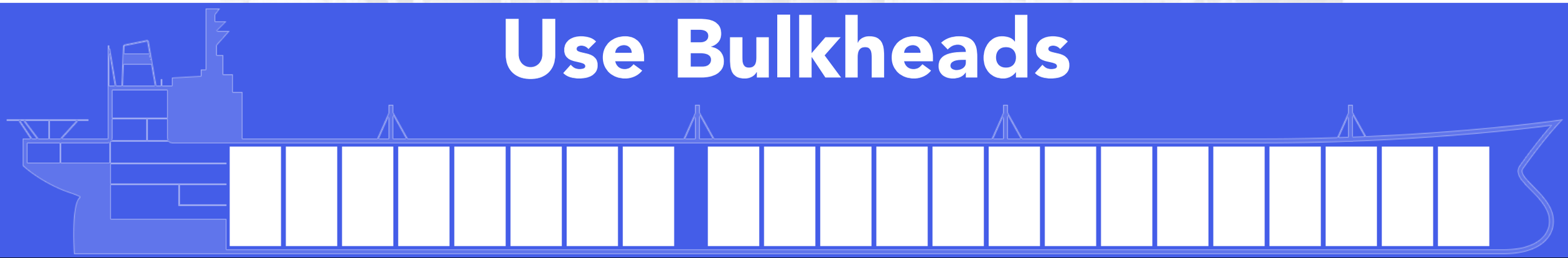
The Right Way

- Isolate the failure
- Compartmentalize
- Manage failure locally
- Avoid cascading failures

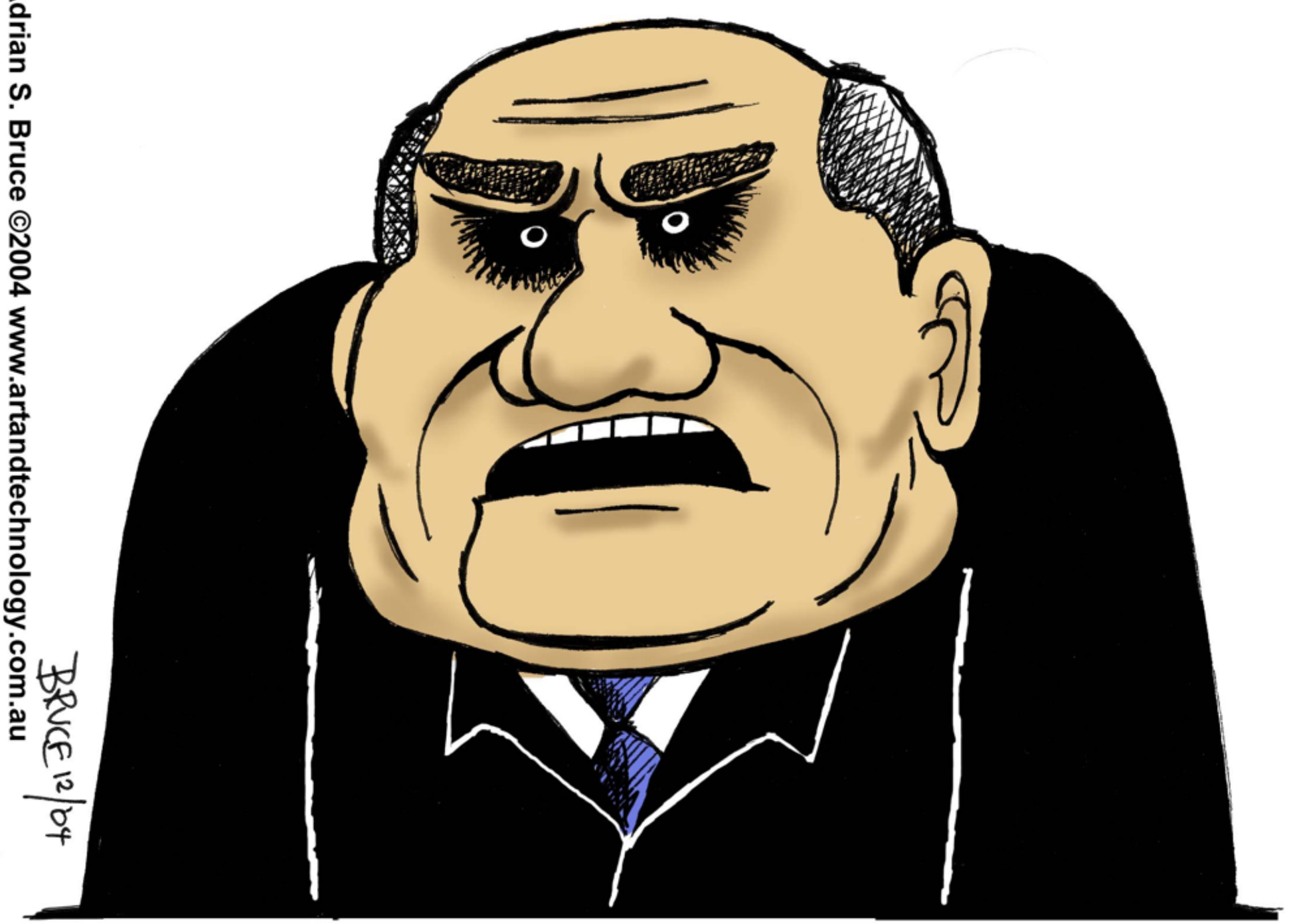
The Right Way

- Isolate the failure
- Compartmentalize
- Manage failure locally
- Avoid cascading failures

Use Bulkheads



...together with supervision



THE BEATINGS WILL CONTINUE UNTIL MORALE IMPROVES

...together with supervision

1. Use Isolated lightweight processes (compartments)
2. Supervise these processes
    1. Each process has a supervising parent process
    2. Errors are reified and sent as (async) events to the supervisor
    3. Supervisor manages the failure - can kill, restart, suspend/resume

- Same semantics local as remote
- Full decoupling between business logic & error handling
- Build into the Actor model

Supervision in Akka

Every single actor has a
default supervisor strategy.
Which is usually sufficient.
But it can be overridden.

Supervision in Akka

Every single actor has a
default supervisor strategy.
Which is usually sufficient.
But it can be overridden.

```
class Supervisor extends UntypedActor {
  private SupervisorStrategy strategy = new OneForOneStrategy(
    10,
    Duration.parse("1 minute"),
    new Function<Throwable, Directive>() {
      @Override public Directive apply(Throwable t) {
        if (t instanceof ArithmeticException)       return resume();
        else if (t instanceof NullPointerException) return restart();
        else                                        return escalate();
    }
  });

  @Override public SupervisorStrategy supervisorStrategy() {
```

Typesafe

akka

Supervision in Akka

```
class Supervisor extends UntypedActor {
  private SupervisorStrategy strategy = new OneForOneStrategy(
    10,
    Duration.parse("1 minute"),
    new Function<Throwable, Directive>() {
      @Override public Directive apply(Throwable t) {
        if (t instanceof ArithmeticException)        return resume();
        else if (t instanceof NullPointerException) return restart();
        else                                        return escalate();
    }
  });

  @Override public SupervisorStrategy supervisorStrategy() {
    return strategy;
  }

  ActorRef worker = context.actorOf(new Props(Worker.class));

  public void onReceive(Object message) throws Exception {
    if (message instanceof Integer) worker.forward(message);
  }
}
```

Typesafe

akka

Responsive

"Quick to respond or react appropriately"

- Merriam Webster

Keep

latency consistent

1. Blue sky scenarios
2. Traffic bursts
3. Failures

Keep

latency consistent

1. Blue sky scenarios
2. Traffic bursts
3. Failures

The system should

always be responsive

Typesafe

Use Back Pressure

Bounded queues with backoff strategies

Use Back Pressure

Bounded queues with backoff strategies

Respect Little's Law:

$$L = \lambda W$$

Queue Length = Arrival Rate * Response Time

Use Back Pressure

Bounded queues with backoff strategies

Respect Little's Law:

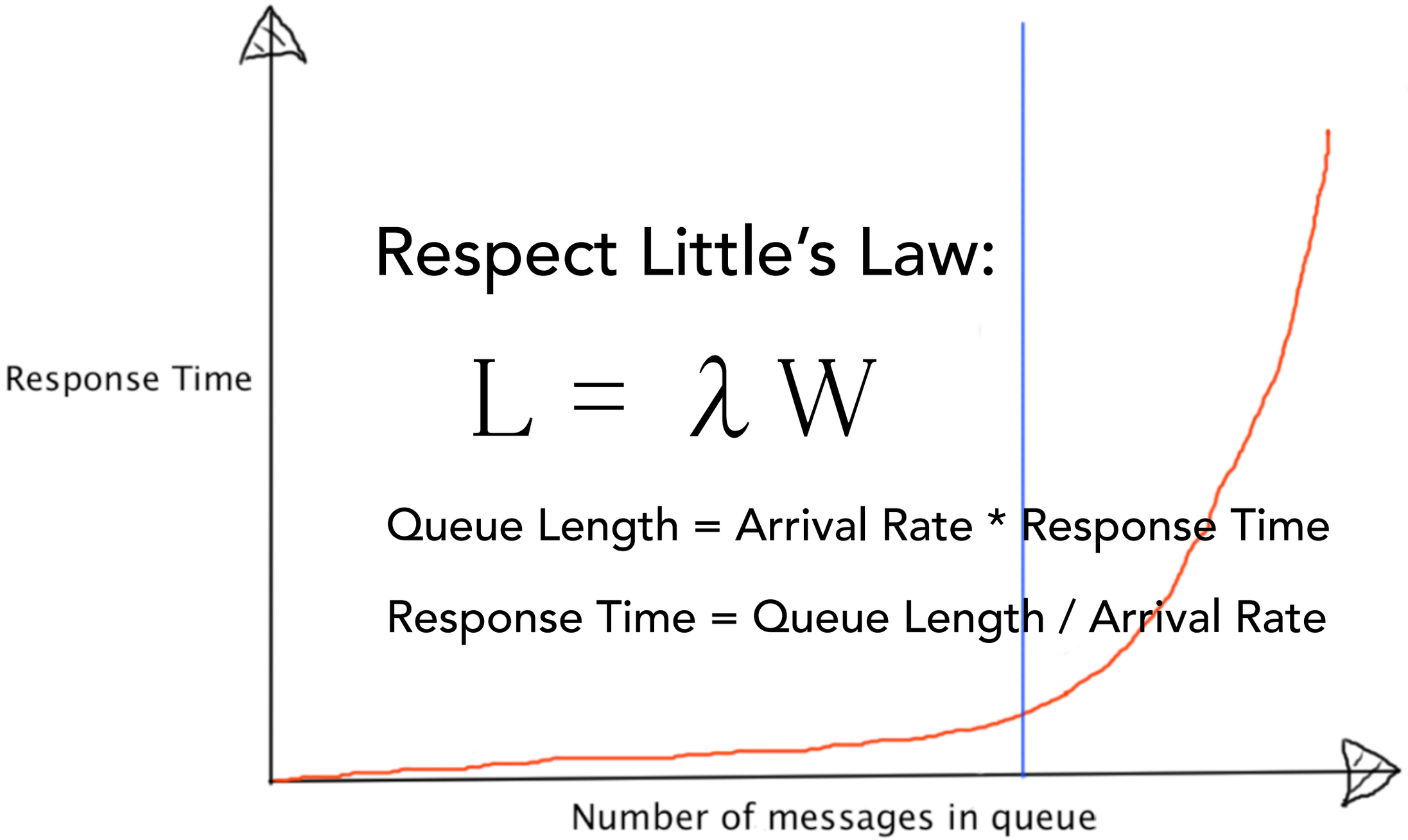
$$L = \lambda W$$

Queue Length = Arrival Rate * Response Time

Response Time = Queue Length / Arrival Rate

Use Back Pressure

Bounded queues with backoff strategies



Respect Little's Law:

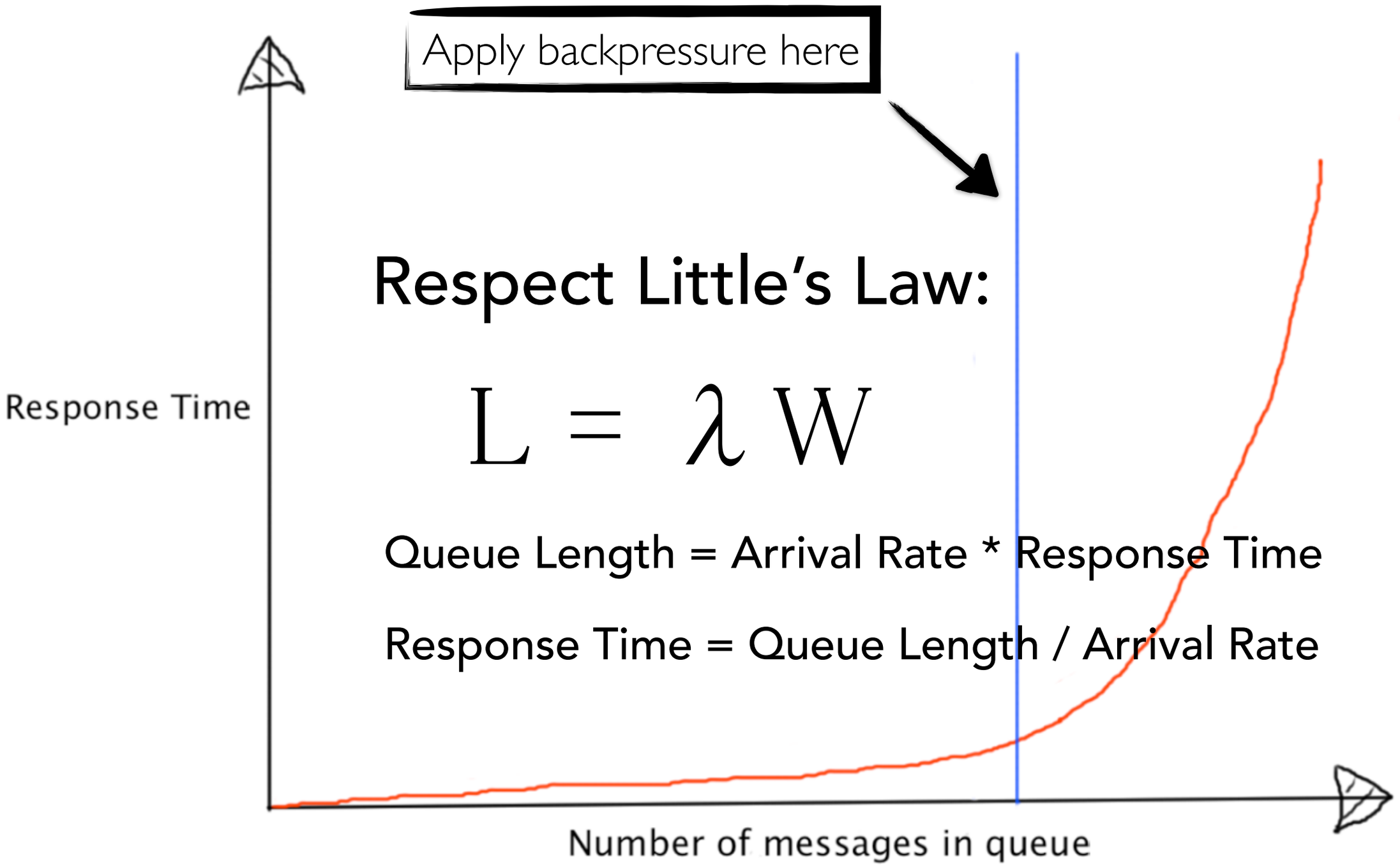
$$L = \lambda W$$

Queue Length = Arrival Rate * Response Time

Response Time = Queue Length / Arrival Rate

Use Back Pressure

Bounded queues with backoff strategies



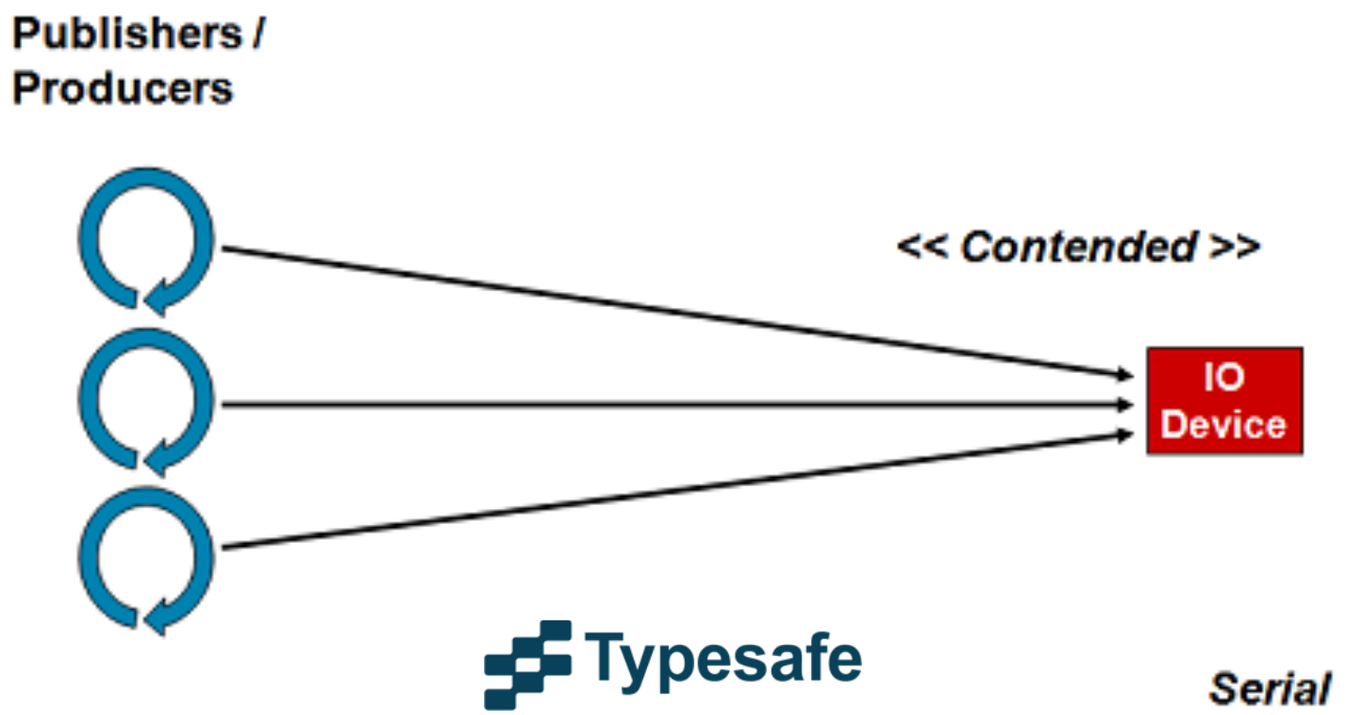
Respect Little's Law:

$$L = \lambda W$$

Queue Length = Arrival Rate * Response Time

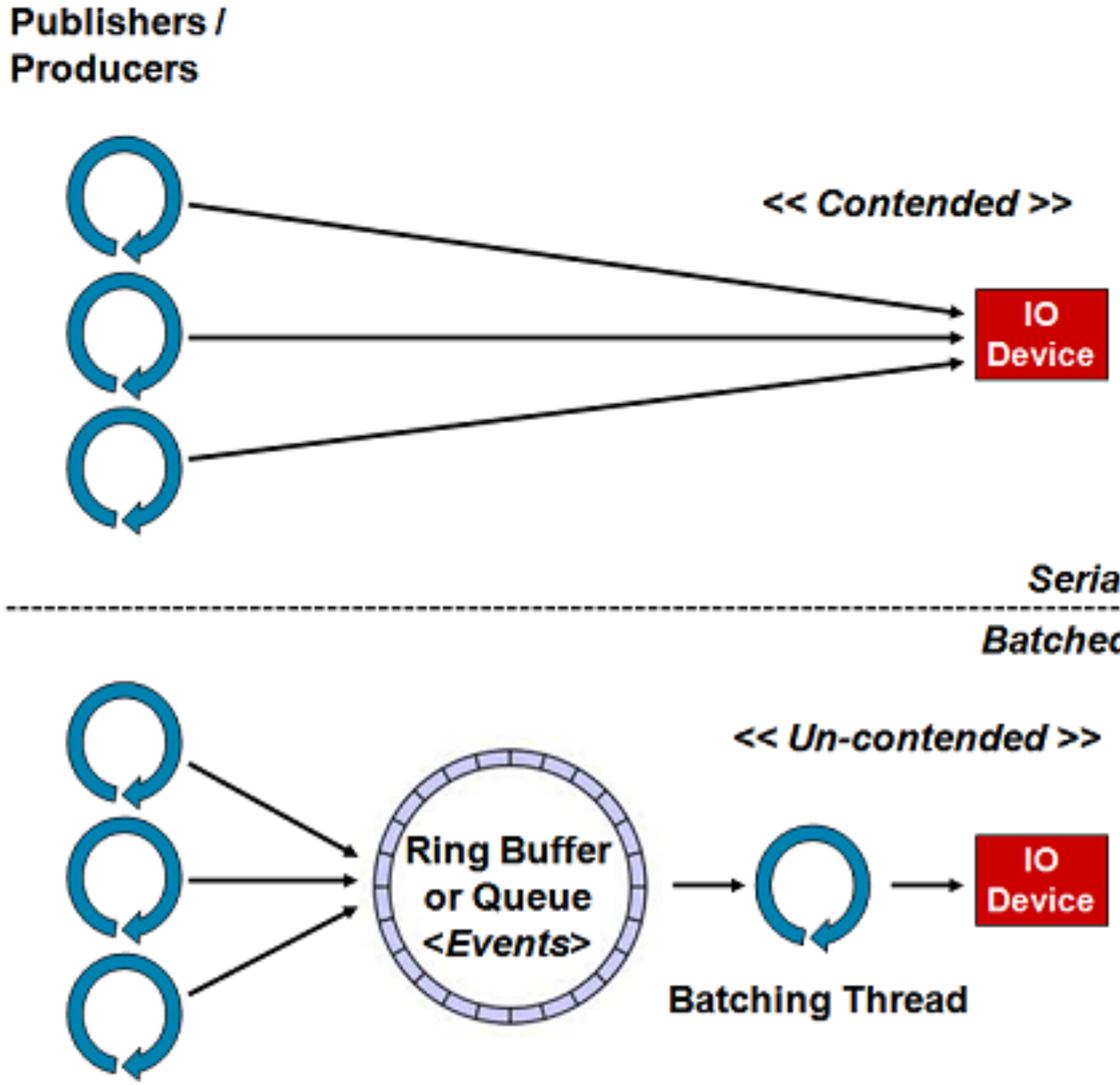
Response Time = Queue Length / Arrival Rate

Smart Batching



Smart Batching
http://bit.ly/smartbatching
By Martin Thompson

Smart Batching



Typesafe

Smart Batching
http://bit.ly/smartbatching
By Martin Thompson

Reactive Web & Mobile Apps

Typesafe

Reactive Web & Mobile Apps

1. Reactive Request

Async & Non-blocking Request & Response

Reactive Web & Mobile Apps

1. Reactive Request

Async & Non-blocking Request & Response

2. Reactive Composition

Reactive Request + Reactive Request + ...

Typesafe

Reactive Web & Mobile Apps

1. Reactive Request

   Async & Non-blocking Request & Response

2. Reactive Composition

   Reactive Request + Reactive Request + ...

3. Reactive Push

   Stream Producer

Typesafe

Reactive Web & Mobile Apps

1. Reactive Request

   Async & Non-blocking Request & Response

2. Reactive Composition

   Reactive Request + Reactive Request + ...

3. Reactive Push

   Stream Producer

4. 2-way Reactive (Bi-Directional Reactive Push)

Typesafe

Reactive Web & Mobile Apps

1. Reactive Request

   Async & Non-blocking Request & Response

2. Reactive Composition

   Reactive Request + Reactive Request + ...

3. Reactive Push

   Stream Producer

4. 2-way Reactive (Bi-Directional Reactive Push)

Typesafe

Reactive Web & Mobile Apps

1. Reactive Request

Async & Non-blocking Request & Response

2. Reactive Composition

Reactive Request + Reactive Request + ...

3. Reactive Push

Stream Producer

4. 2-way Reactive (Bi-Directional Reactive Push)

Enables Reactive UIs

1. Interactive
2. Data Synchronization
3. “Real-time” Collaboration

Typesafe

Reactive Composition in Play

```
public class Application extends Controller {
  public static Result index() {
    return ok(index.render("Your new application is ready."));
  }
}
```

Typesafe

Reactive Composition in Play

```
public class Application extends Controller {
  public static Result index() {
    return ok(index.render("Your new application is ready."));
  }
}
```

standard non-reactive request

Reactive Composition in Play

```
public class Application extends Controller {
  public static Result index() {
    return ok(index.render("Your new application is ready."));
  }
}
```

standard non-reactive request

```
 def get(symbol: String): Action[AnyContent] = Action.async {
   for {
     tweets     <- getTweets(symbol)
     sentiments <- Future.sequence(loadSentiments(tweets.json))
   } yield Ok(toJson(sentiments))
}
```

Reactive Composition in Play

```
public class Application extends Controller {
  public static Result index() {
    return ok(index.render("Your new application is ready."));
  }
}
```

standard non-reactive request

```
 def get(symbol: String): Action[AnyContent] = Action.async {
   for {
     tweets      <- getTweets(symbol)
     sentiments <- Future.sequence(loadSentiments(tweets.json))
   } yield Ok(toJson(sentiments))
}
```

fully reactive non-blocking request composition

typesafe

Get Started – Typesafe Platform – Typesafe

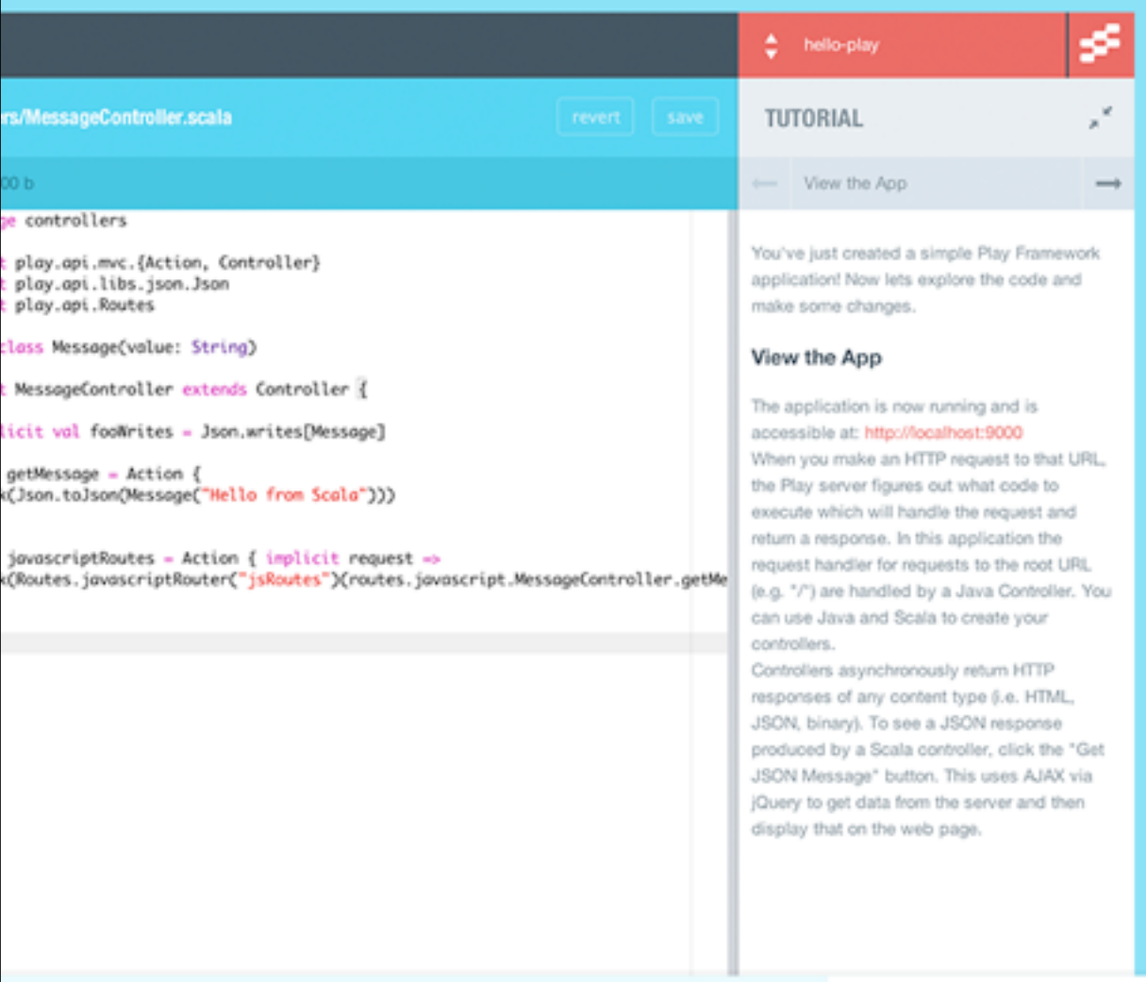
typesafe.com/platform/getstarted

Reader

Typesafe

TYPESAFE PLATFORM HOW WE HELP COMPANY BLOG

TYPESAFE PLATFORM / **GET STARTED**



Typesafe Activator helps you get started with the Typesafe Platform, Play Framework, Akka & Scala

Download Activator

JDK6+ | 203MB
Activator is in Developer Preview

JVM - Based Runtime

Akka

Play

Typesafe Console

① Download Activator, and unzip the file

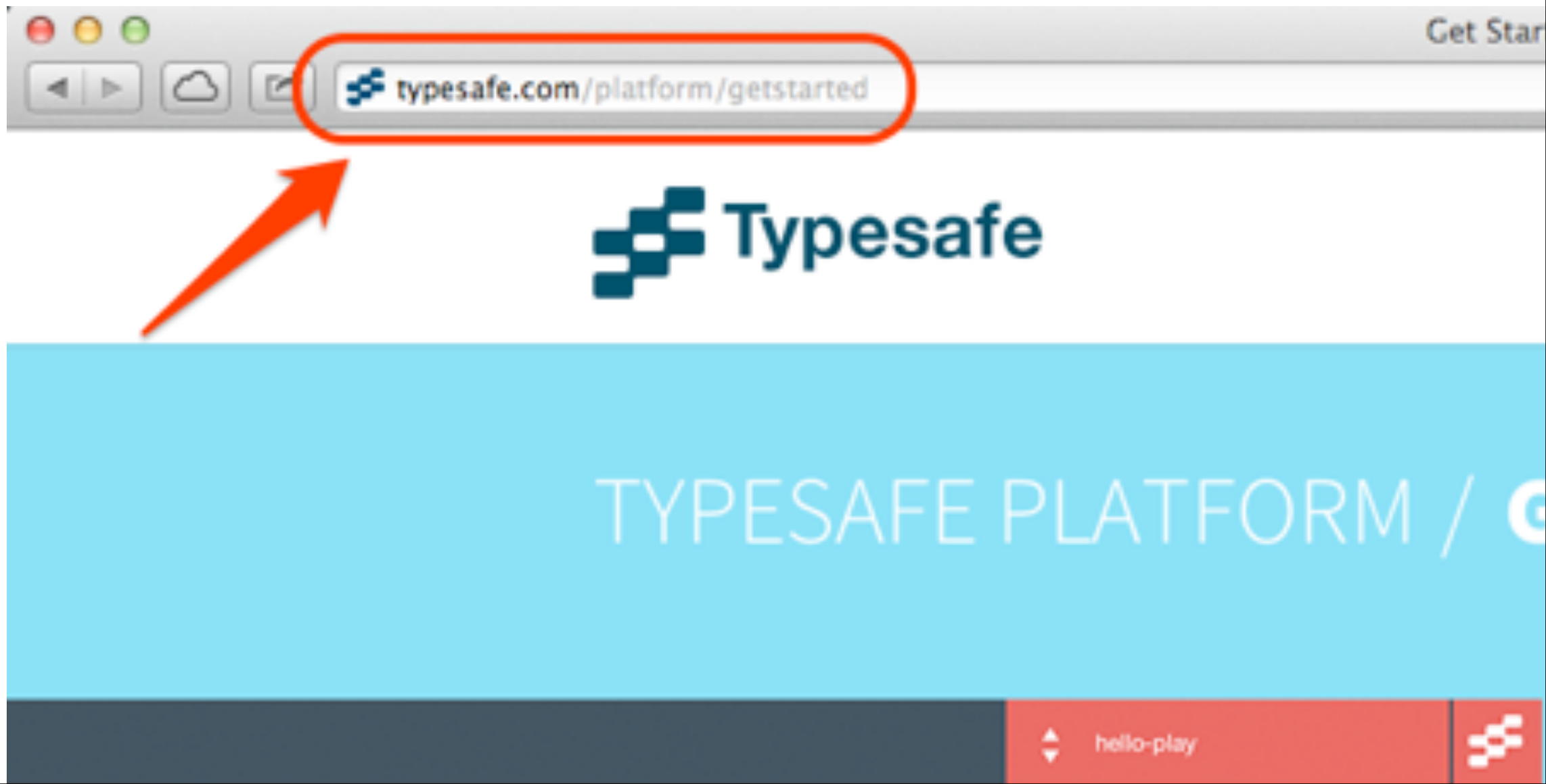
Download Developer Preview 0.1.1 (203MB).

Extract the downloaded zip file to your system.

② Start Activator's UI

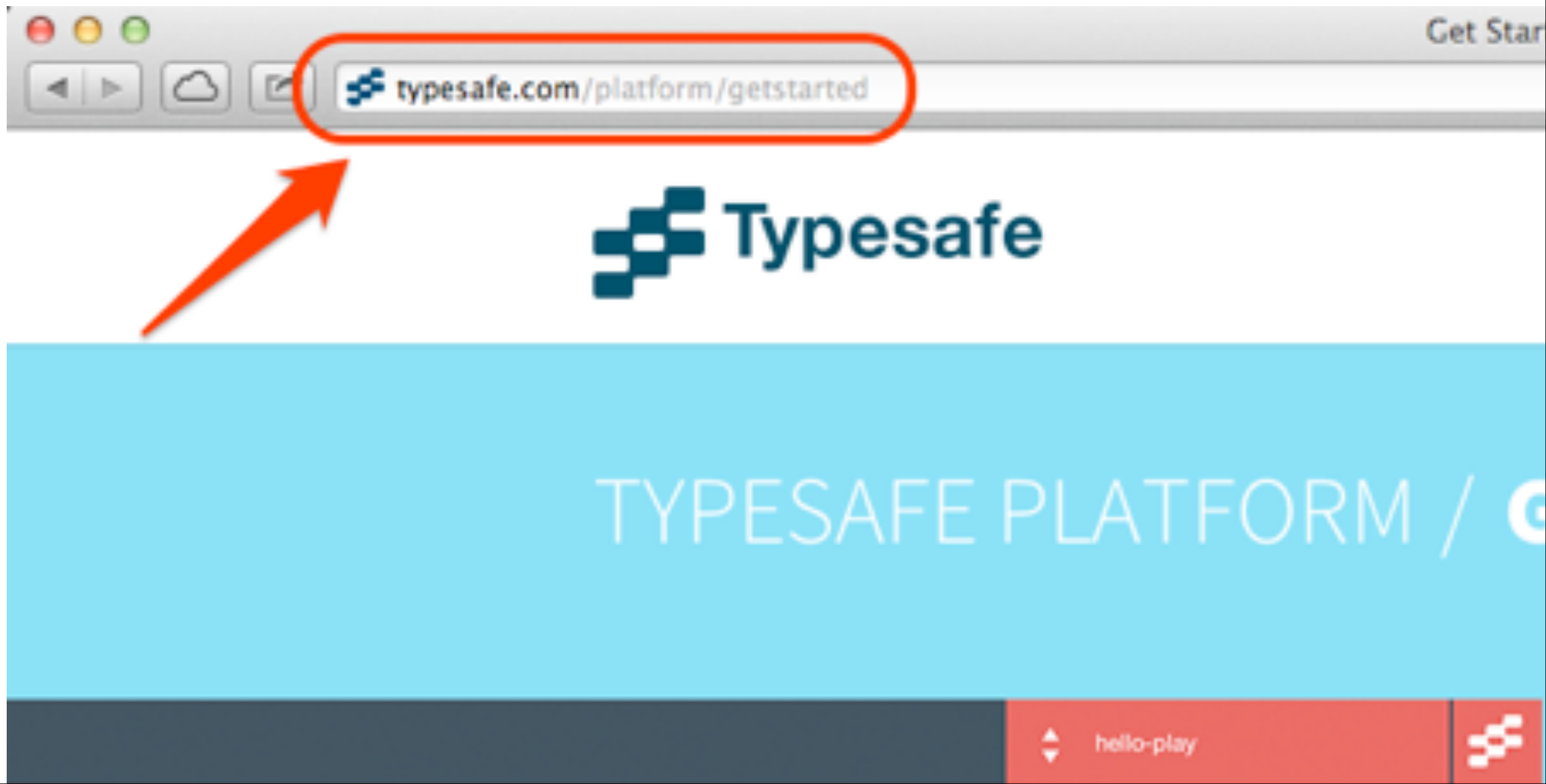
In Finder, navigate to the directory that Activator was extracted to, right-

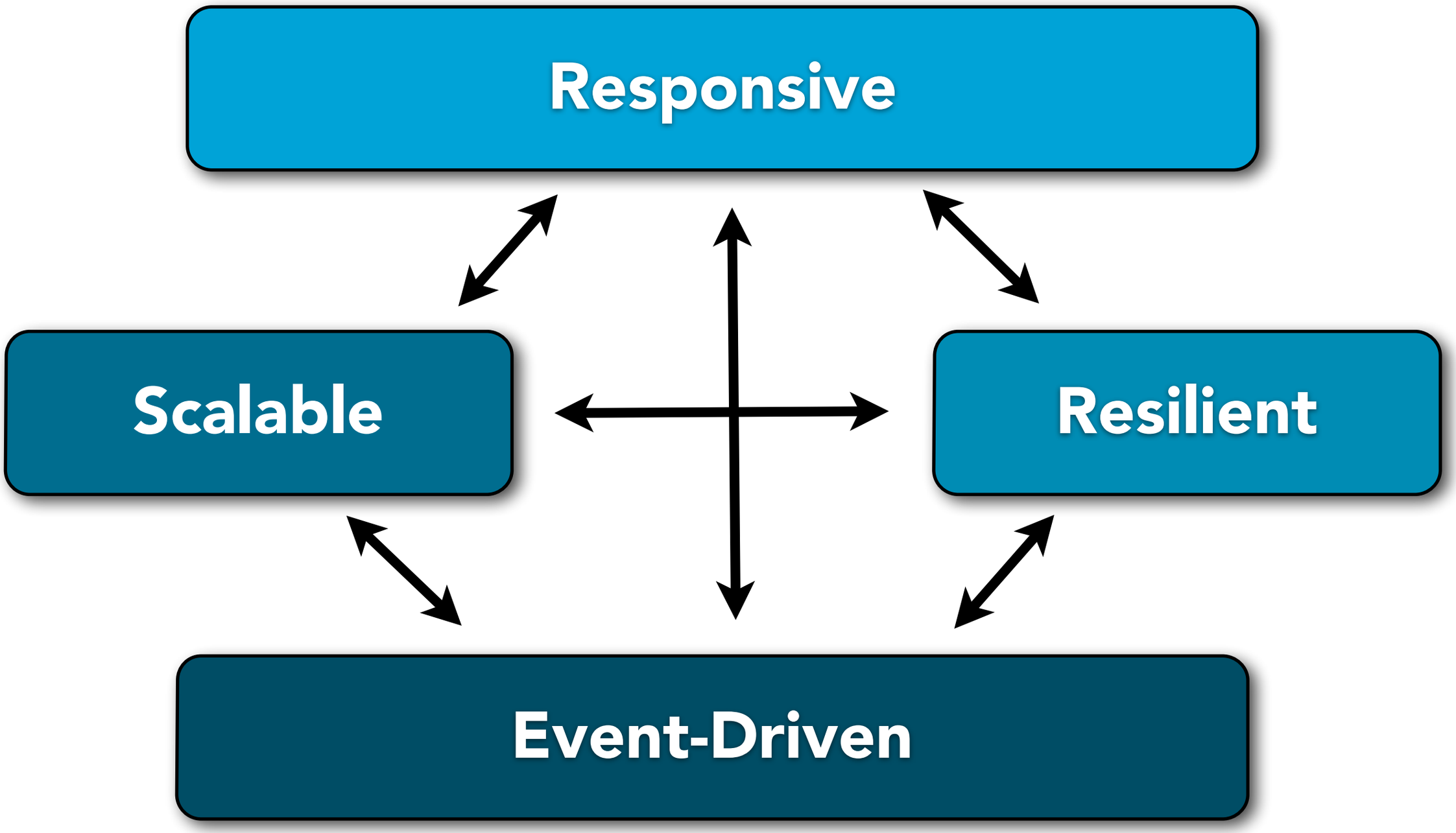
http://typesafe.com/platform/getstarted



Demo time

http://typesafe.com/platform/getstarted




Responsive
Scalable
Resilient
Event-Driven

http://reactivemanifesto.org

The Reactive Manifesto

Published on September 23 2013. (v1.1) *Table of Contents*

Download as PDF

Suggest improvements

Sign the manifesto

3156 people already signed (Full list)

Go Reactive

Email: jonas@typesafe.com
Web: typesafe.com
Twtr: @jboner

Typesafe