High Performance Actors

Kiki, principal enterprise architect @ Lightbend

akka

What we mean by “Reactive”

React to Users

Low latency

Real-time / NRT

Responsive

React to Failures

Graceful, Non-catastrophic Recovery

Self-Healing

Resilient

React to Load Variance

Responsive in the face of changing loads

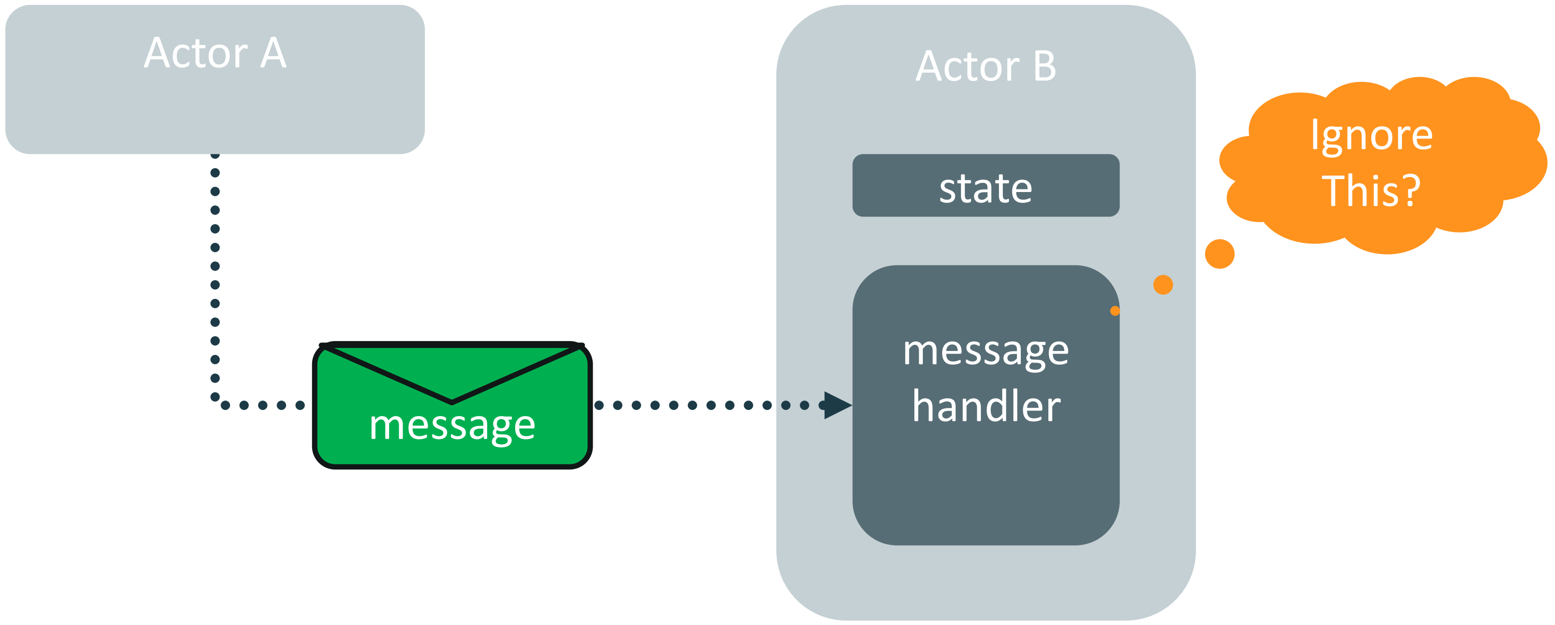
Elastic

The enabler of these characteristics is a Message Driven Model.

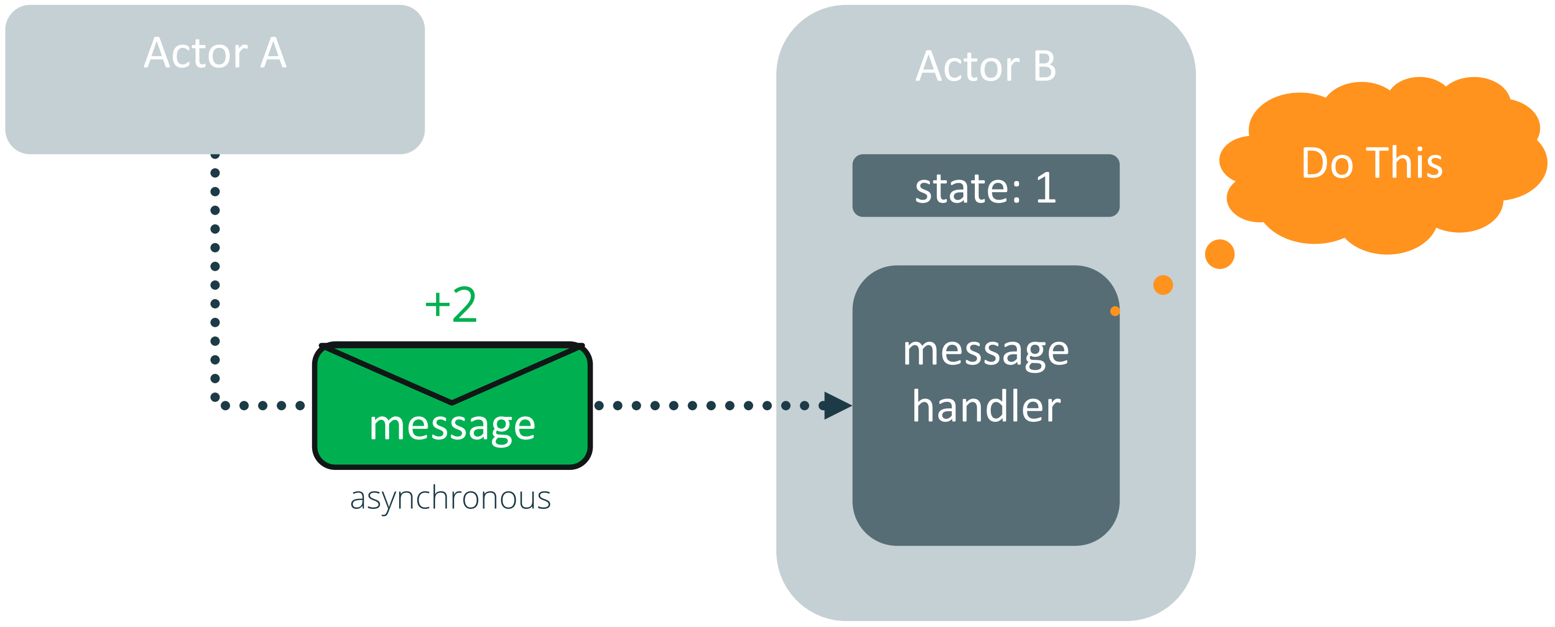
Message Driven Benefits for Speed and Performance

- Fast decisions
- Fast failures
- Fast progress with Concurrency & Parallelism

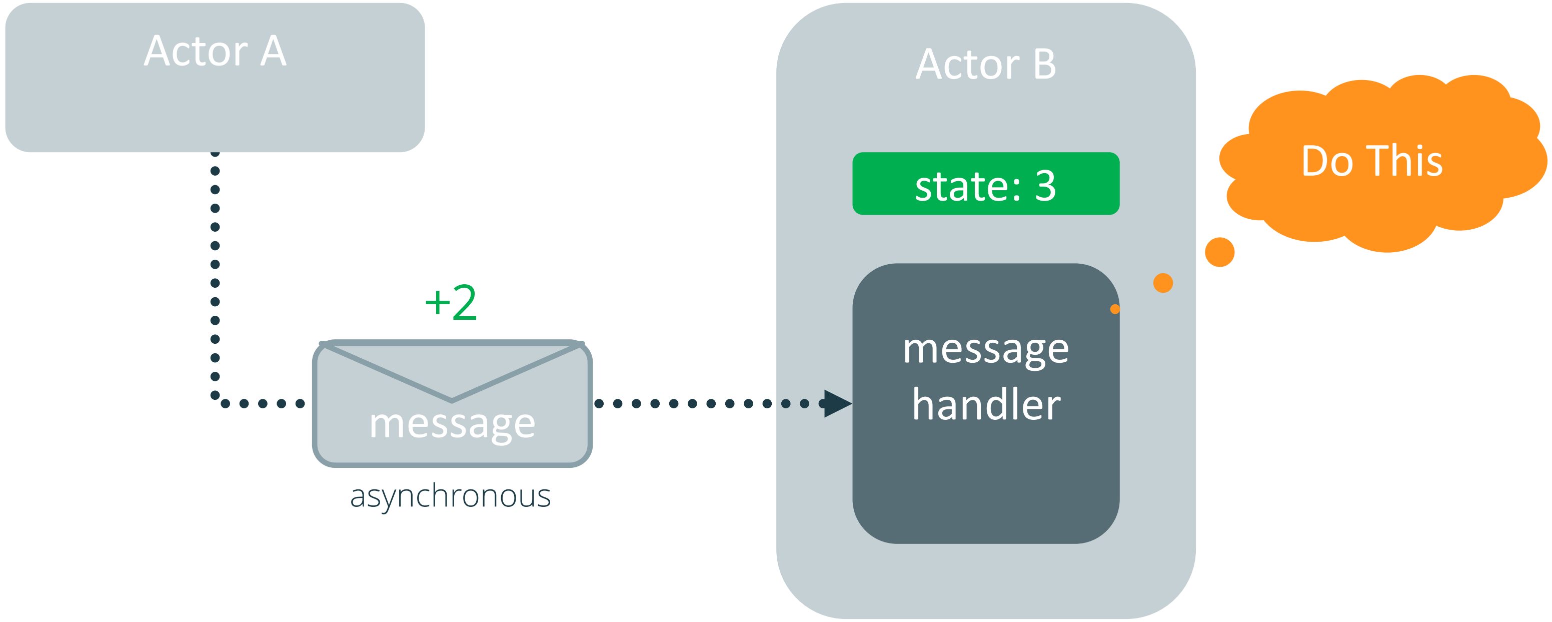
Make a local decision



Make a local decision



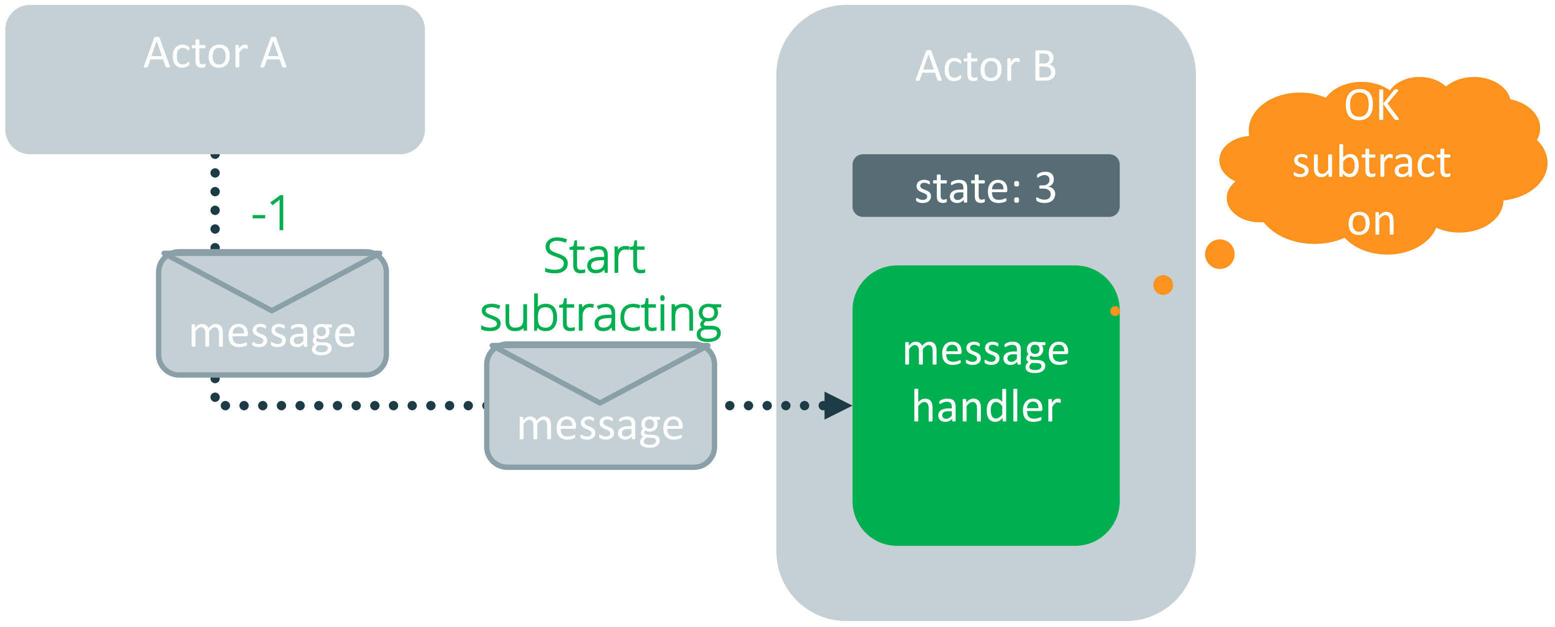
Make a local decision



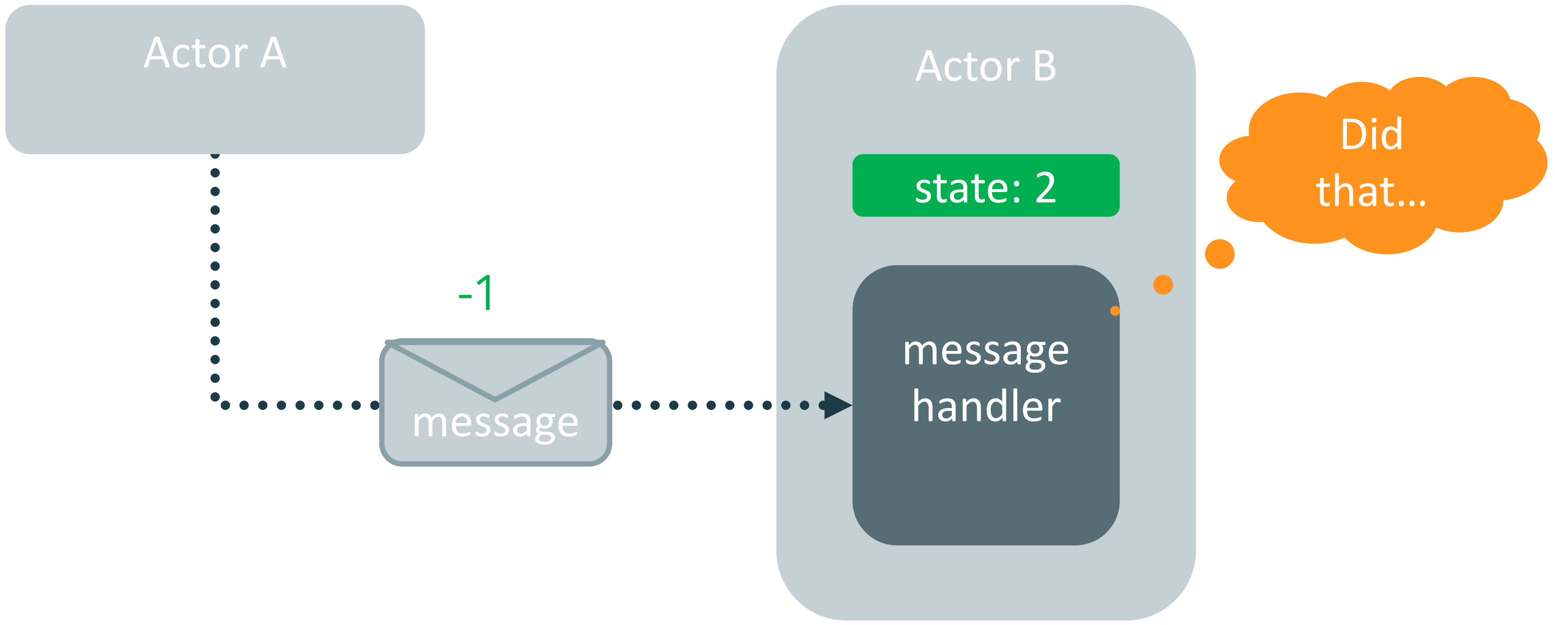
Local decision code

```
@Override
public Receive createReceive() {
  return receiveBuilder()
    .match(String.class, s -> {
      log.info("Received String message: {}", s);
    })
    .matchAny(o -> log.info("received unknown message"))
    .build();
}
```

Change Behavior



Change Behavior



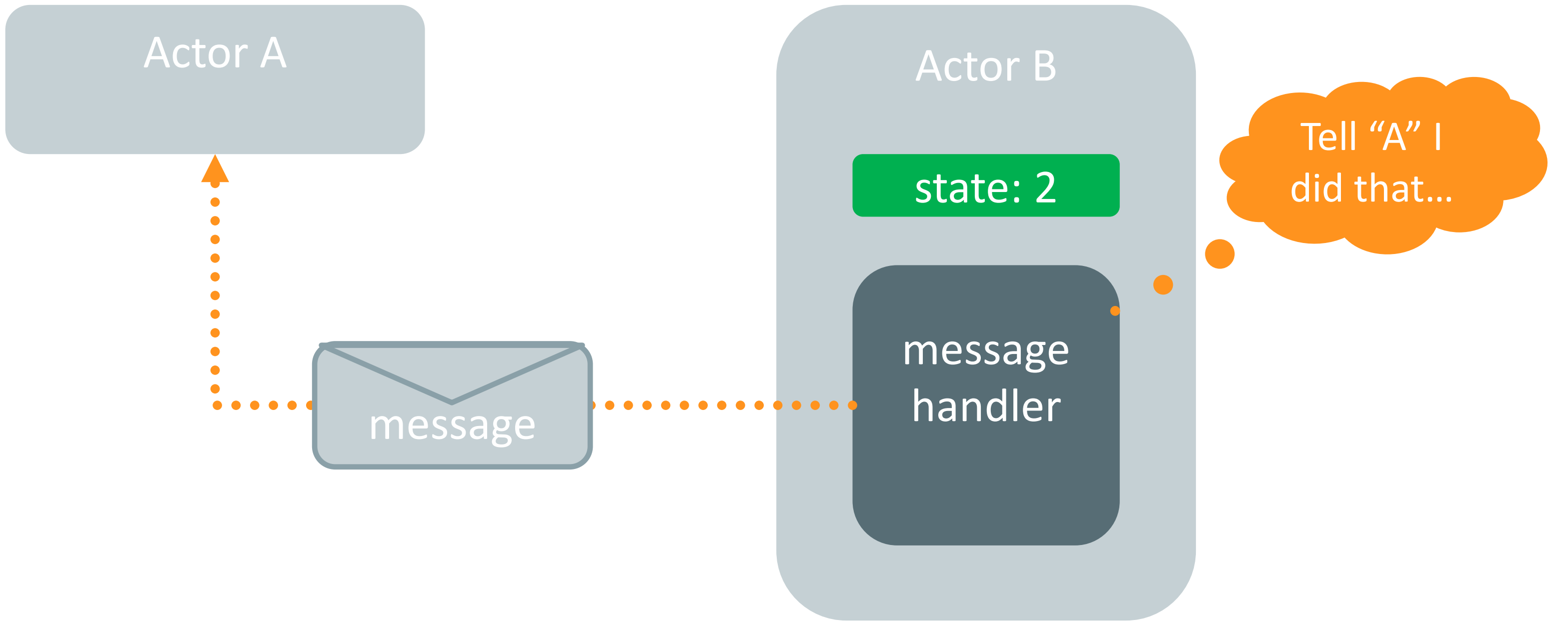
Change behavior code

```java
private AbstractActor.Receive angry;
private AbstractActor.Receive happy;

public HotSwapActor() {
  angry =
    receiveBuilder()
      .matchEquals("foo", s -> {
        getSender().tell("I am already angry?", getSelf());
      })
      .matchEquals("bar", s -> {
        getContext().become(happy);
      })
      .build();

  happy = receiveBuilder()
    .matchEquals("bar", s -> {
      getSender().tell("I am already happy :-)", getSelf());
    })
    .matchEquals("foo", s -> {
      getContext().become(angry);
    })
    .build();
}
```

Change behavior code (cont'd)

```
@Override
public Receive createReceive() {
  return receiveBuilder()
    .matchEquals("foo", s ->
      getContext().become(angry)
    )
    .matchEquals("bar", s ->
      getContext().become(happy)
    )
    .build();
}
```

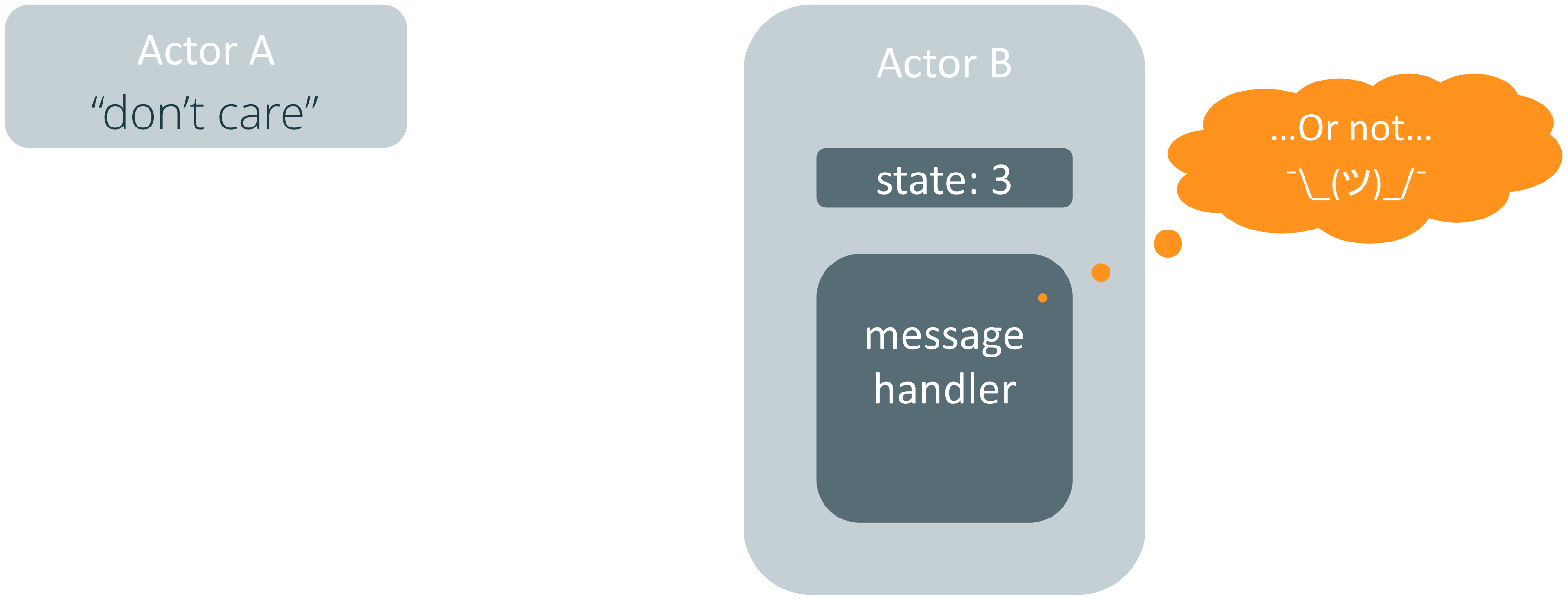
Send more messages



Send more messages code

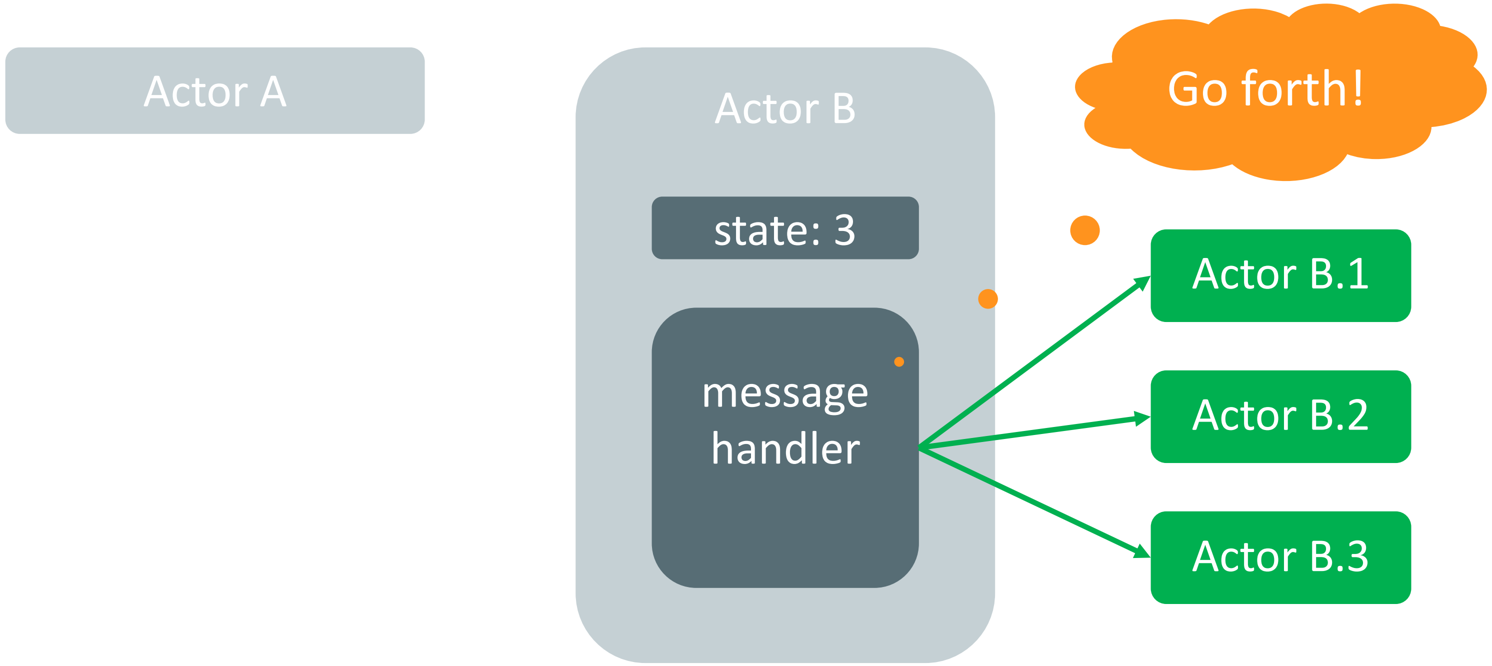
```
@Override
public Receive createReceive() {
  return receiveBuilder()
    .match(Integer.class, i -> {
      getSender().tell(i + magicNumber, getSelf());
    })
    .build();
}
```

Or chose not to



Trust but verify

Create More Actors



Create more actors code

Java

```java
public class SomeOtherActor extends AbstractActor {

  ActorRef demoActor = getContext().actorOf(DemoActor.props(42), "demo");
  // ...
}
```

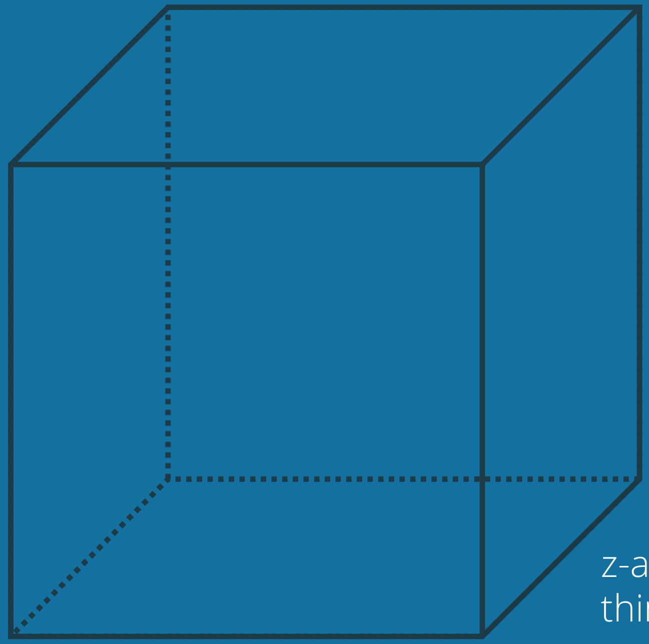
Create Actors Liberally

Akka actors have a small memory footprint; ~2.5 million actors per GB of heap.

Akka Abstractions

- Akka Streams
- Akka Cluster
- Akka Http

Scalability Cube: 3 dimensions of scalability



Y-axis:
scale via
functional
compositi
on

x-axis: horizontal scaling via
duplication and load balancing

z-axis: split similar
things via data
partitioning/sharding

never block
always asynchronous

Non-blocking

Actor messaging, default mailbox, work scheduling is non blocking.

Asynchronous

Actors are asynchronous by nature: an actor can progress after a message send without waiting for the actual delivery to happen.

Dispatcher

- Selfless: Doesn’t hog resources
- Responsible for assigning threads
- Only assigns when necessary
- Idle actors do not use threads
- Dispatcher always paired up with an executor which will define what kind of thread pool model is used to support the actors
- CPU caches likely each time actors are assigned threads and warmed caches are one of your best friend for high performance.

Fork Join Executor - from 1.4 million to 20 million messages per second

…and then from 20 million to 50 million messages per second

throughput vs fairness

Even faster in actors

- CAS compare and swap
    - Atomic values, adders and accumulators

Mechanical Sympathy

What kinds of machines, with how many cores, will run this application? How CPU-bound are the tasks being performed by my actors? How many threads can I realistically expect to run concurrently for this application on those machines?

Akka Streams

Implementation of Reactive Streams, concerned with applying backpressure when producers are faster than consumers.

Artery is designed from the ground up to support high-throughput in the magnitude of 1 million messages per second and low-latency in the magnitude of 100 microseconds.

Artery Aeron Performance

- 630,239 messages/s with message payload of 100 bytes
- 8,245 messages/s with messages payload of 10,000 bytes
- Round trip latency at a message rate of 10,000 messages/s:
  - 50%ile: 155 μs,
  - 90%ile: 173 μs
  - 99%ile: 196 μs

**two m4.4xlarge EC2 instances (1 Gbit/s bandwidth)*

Aeron efficient, guaranteed transport.

Designed to work with low-latency, high-throughput systems.

Aeron “Busy Spinning”

Message offers or polls are retried until successful.

Tuning: CPU Usage vs Latency

```
# Values can be from 1 to 10, where 10 strongly prefers low latency
# and 1 strongly prefers less CPU usage
akka.remote.artery.advanced.idle-cpu-level = 1
```

Aeron Interaction Pattern

The unidirectional nature of Aeron channels fits nicely with the Akka peer-to-peer communication model.

ByteBuffer Based Serialization

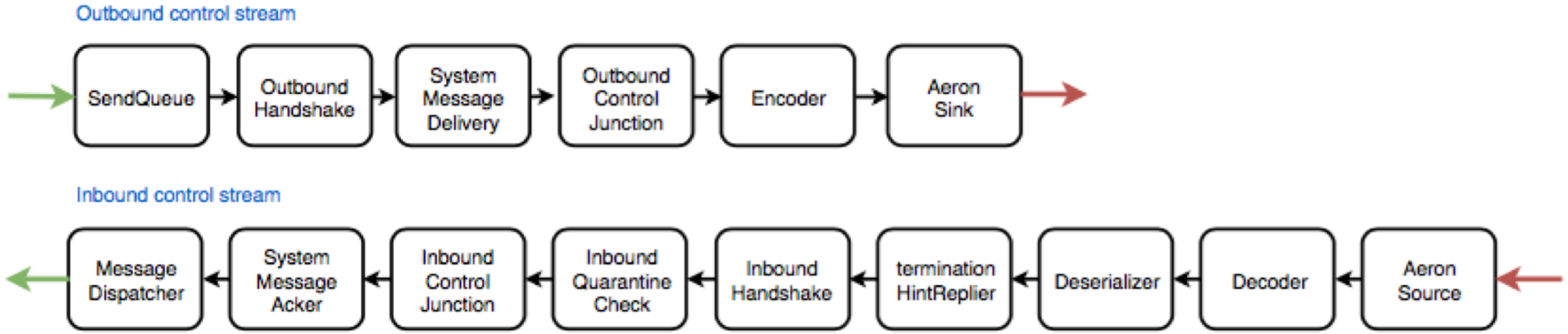
ByteBuffer Serializer Interface

```
interface ByteBufferSerializer {
  /**
   * Serializes the given object into the `ByteBuffer`.
   */
  void toBinary(Object o, ByteBuffer buf);

  /**
   * Produces an object from a `ByteBuffer`, with an optional type-hint;
   * the class should be loaded using ActorSystem.dynamicAccess.
   */
  Object fromBinary(ByteBuffer buf, String manifest);
}
```

Aeron Message Channels

User Messages
Internal Control Messages

Control stream



All messages are not created equally

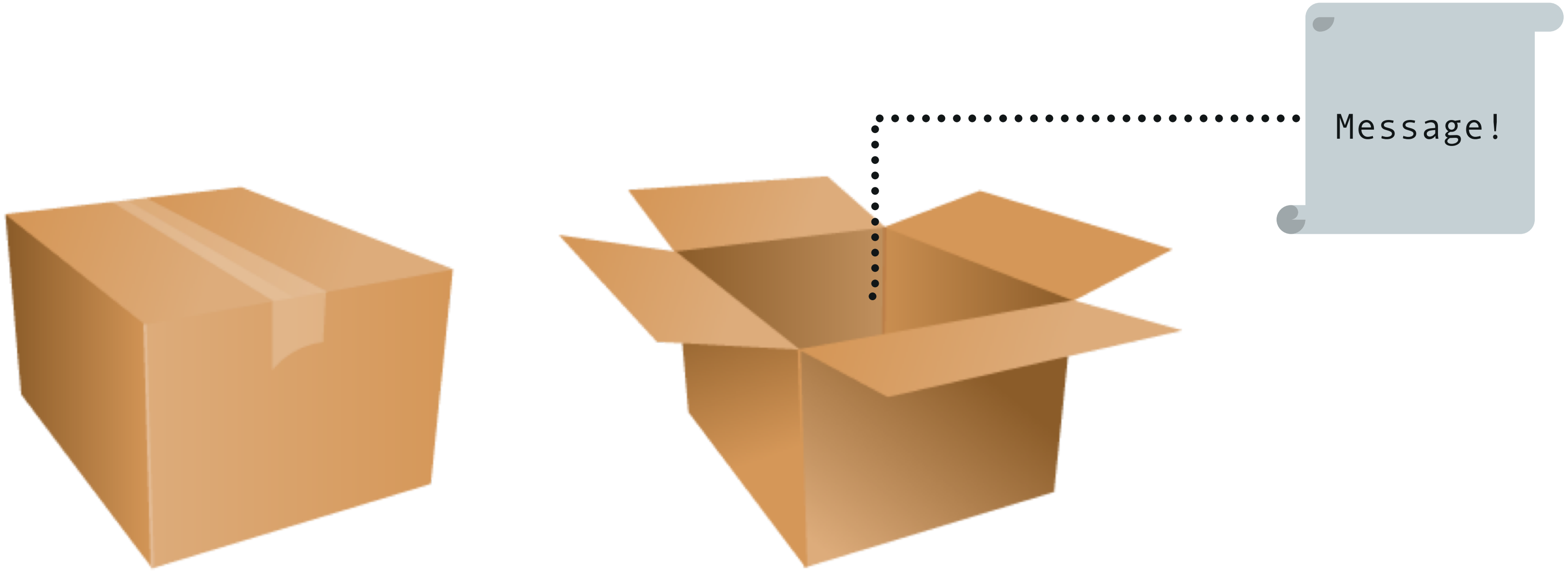
User Messages

Internal Control Messages

Optional: Large Messages

Actor Path Compression

Actor Path Compression



Compression Algorithm

Receiver initiates algorithm

Advertises compression table

Remote sender uses advertised table

Built-in Flight-Recorder

Flight Recorder

- Fixed size file
- This file is crash resistant
- Very low overhead

Akka Cluster

Akka Cluster

- Gossip Protocol
- Adaptive load balancing

Akka Cluster: Gossip

The cluster membership used in Akka is based on Amazon’s Dynamo system and particularly the approach taken in Basho’s’ Riak distributed database.

Akka Cluster: Gossip

The cluster membership used in Akka is based on Amazon's Dynamo system and particularly the approach taken in Basho's' Riak distributed database.

Akka Cluster: Gossip Convergence

Information about the cluster converges locally at a node at certain points in time.

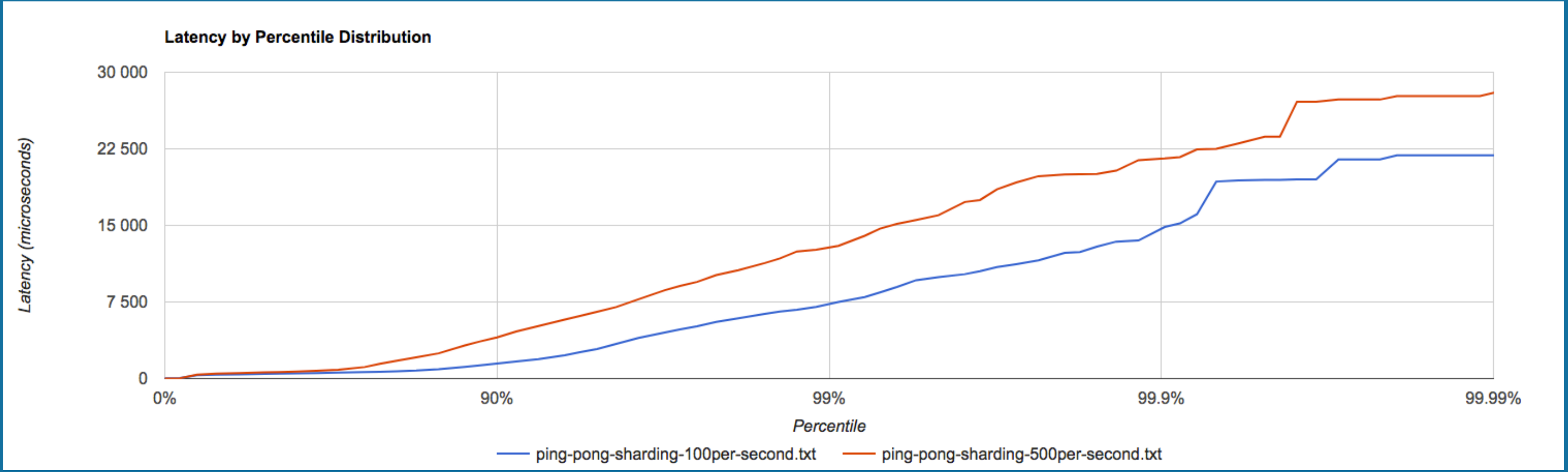
Akka Cluster: Gossip Protocol

A variation of push-pull gossip is used to reduce the amount of gossip information sent around the cluster.

Akka Cluster: Sharding performance

Cluster Sharding can be scaled in a nearly linear fashion. It can easily handle millions of Actors without much regression in messaging speed or message loss.

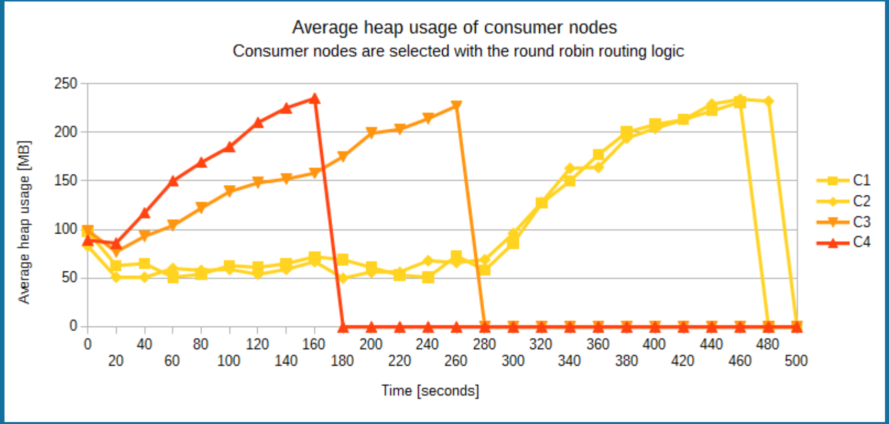
20 cluster nodes and 10,000 clustered actors. First we measured the latency given a message throughput of 100 pings per second, and then given a throughput of 500 pings per second.

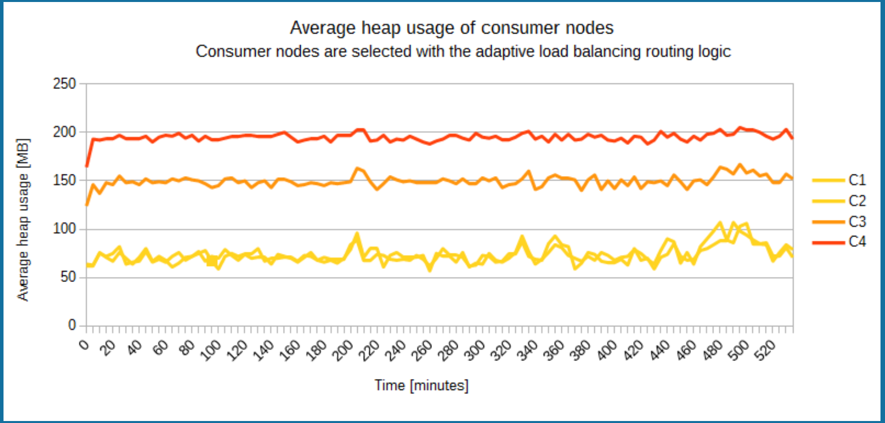


Akka Cluster: Adaptive Load Balancing

The AdaptiveLoadBalancingPool / AdaptiveLoadBalancingGro
up performs load balancing of messages to cluster nodes
based on the cluster metrics data.

Akka Cluster: Adaptive Load Balancing





http://blog.kamkor.me/Akka-Cluster-Load-Balancing/

A user’s story

PayPal

"Powered by Akka and Scala, squbs has already provided very high-scale results with a low infrastructure footprint:
our applications are able to serve over a billion hits a day with as little as 8 VMs and 2 vCPU each."

PayPal

"Akka helps our systems stay responsive even at 90% CPU, very uncharacteristic for our older architectures, and provides for transaction densities never seen before. Batches or micro-batches do their jobs in one-tenth of the time it took before. With wider adoption, we will see this kind of technology being able to reduce cost and allow for much better organizational growth without growing the compute infrastructure accordingly."

Lightbend

PayPal

"Not only do we achieve much larger transaction rates with the same hardware infrastructure, but we also often cut down code by 80% for the same functionality when compared to equivalent imperative code."

LinkedIn

Presence Platform display real-time presence status for a member’s connections across LinkedIn messaging, feed, notifications, etc. on both mobile and web for hundreds of millions of LinkedIn members across the globe.

Is it really so Difficult?

`Akka makes the difficult possible`

How can I be confident?

- The Actor model is proven
- Akka is designed from ground up for performance on the JVM
- Akka scales on all axis X, Y, Z
- Other companies have benefited from Akka
- You can confidently focus on your core competencies while using Akka.

Getting started with Akka

- Akka.io / akka guides
- Talk to users, gitter chat
- Tweet the akka team: @akkateam
- For fun tweet me: @kikisworldrace