ORACLE APPROVED EDUCATION CENTER

2012 Java Developer Day

JavaTWO專業技術大會

開放、學習、分享、盡在Java Developer Day

Java/Scala Developer

何永琳

ORACLE APPROVED EDUCATION CENTER
2012 Java Developer Day
JavaTWO專業技術大會
Java
開放、學習、分享、盡在Java Developer Day

Introduction to Actor Model & Akka

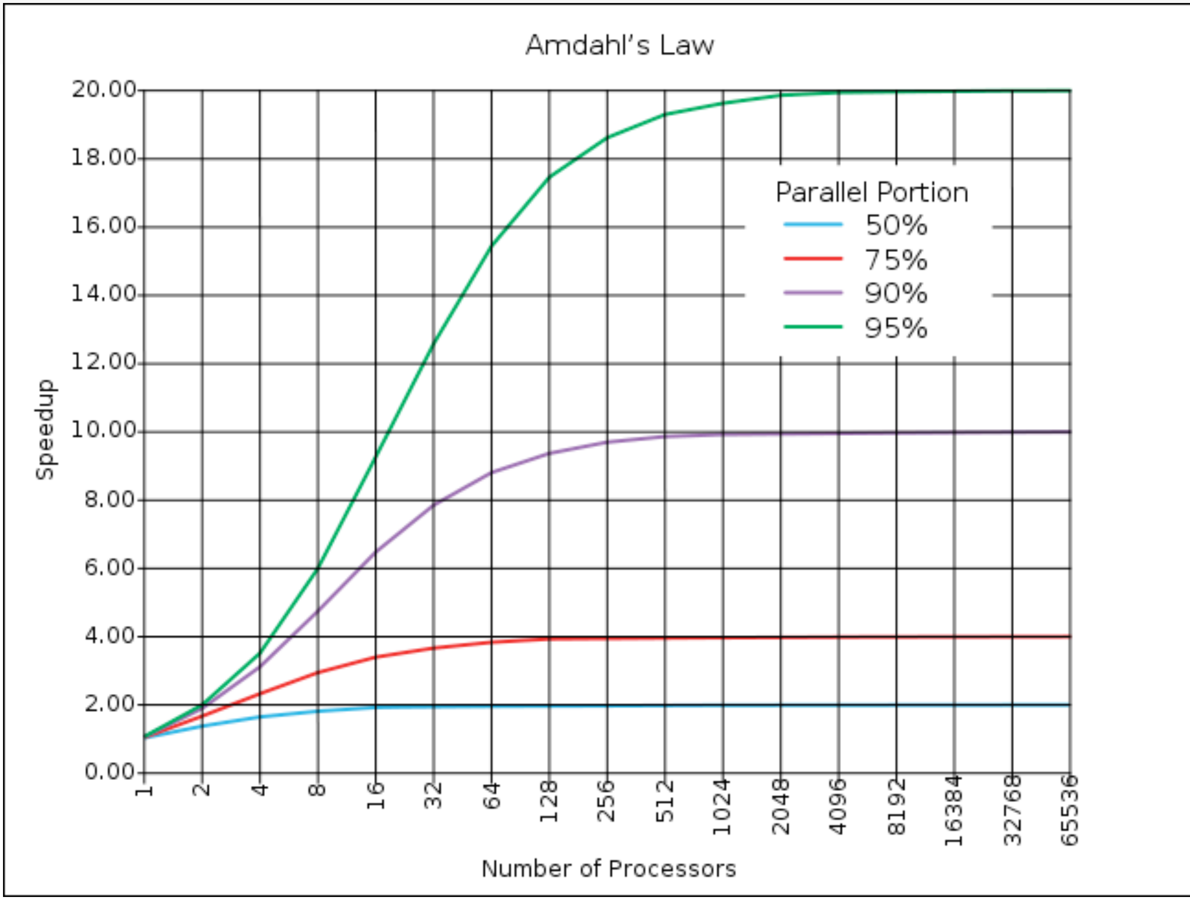
The Challenge

–The clock speed has stopped growing since 2006

–The free lunch is over

–Moore’s Law still applies but only the number of cores in a single chip is increasing.

–The new reality: Amdahl's Law.



ref: http://en.wikipedia.org/wiki/Amdahl's_law

Concurrency and Parallelism

–Concurrency: A condition that exists when at least two threads are making progress. A more generalized form of parallelism that can include time-slicing as a form of virtual parallelism.

–Parallelism: A condition that arises when at least two threads are executing simultaneously.

–Both of them are hard because of shared mutable state.

Issue: Shared Memory Concurrency

- Multithreaded Programs are hard to write and test
  - Non-deterministic
  - Data Race / Race Condition
  - Locks are hard to use
    - too many locks
    - too few locks
    - locks in wrong order

- Poor Performance.
  - False sharing: Cache Line Issue.

The solution

- A new high level programming model
  - easier to understand
  - deterministic
  - no shared/mutable state
  - fully utilize multi-core processors

- Possible Solutions:
  - Functional Programming - Everything is immutable.

```
scala> List(1, 2, 3).par.map(_ + 2)
res: List[Int] = List(3, 4, 5)
```

  - Actor Model - Keep mutable state internal and communicate with each other through asynchronous messages.

A Brief of the Actor Model

- Formalized in 1973 by Carl Hewitt and refined by Gul Agha in mid 80s.
- The first major adoption is done by Ericsson in mid 80s.
  - Invented Erlang and later open-sourced in 90s.
  - Built a distributed, concurrent, and fault-tolerant telcom system which has 99.9999999% uptime

Actor Model

–Actors instead of Objects

–No shared state between actors.

–Asynchronous message passing.

Actor

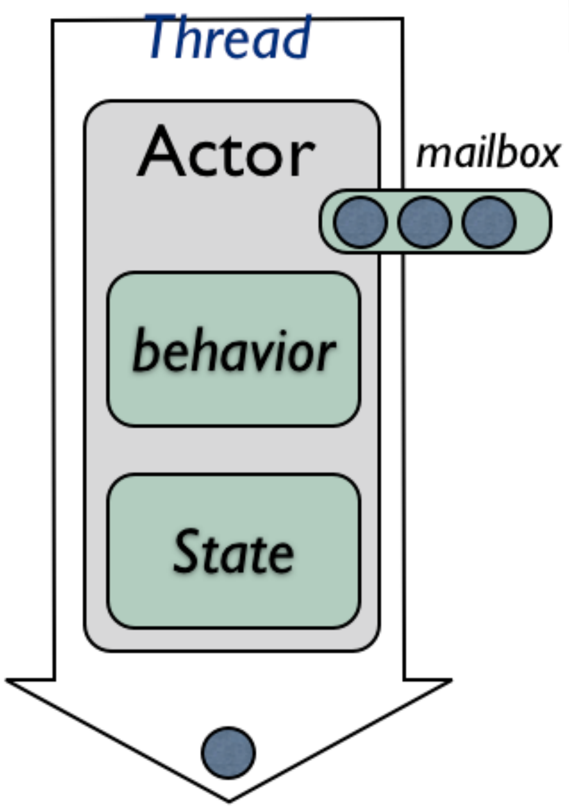
–Lightweight object.

–Keep state internally

–Asynchronous and non-blocking

–Messages are kept in mailbox and processed in order.

–Massive scalable and lighting fast because of the small call stack.



Introduce Akka

–Founded by Jonas Boner and now part of Typesafe stack.

–Actor implementation on JVM.

–Java API and Scala API

–Support Remote Actor

–Modules: akka-camel, akka-spring, akka-zeromq

Define Actor

```java
import akka.actor.UntypedActor;

public class Counter extends UntypedActor {

  private int count = 0;

  public void onReceive(Object message) throws Exception {
    if (message.equals("increase") {
      count += 1;
    } else if (message.equals("get") {
      getSender().tell(new Result(count));
    } else {
      unhandled(message);
    }
  }
}
```

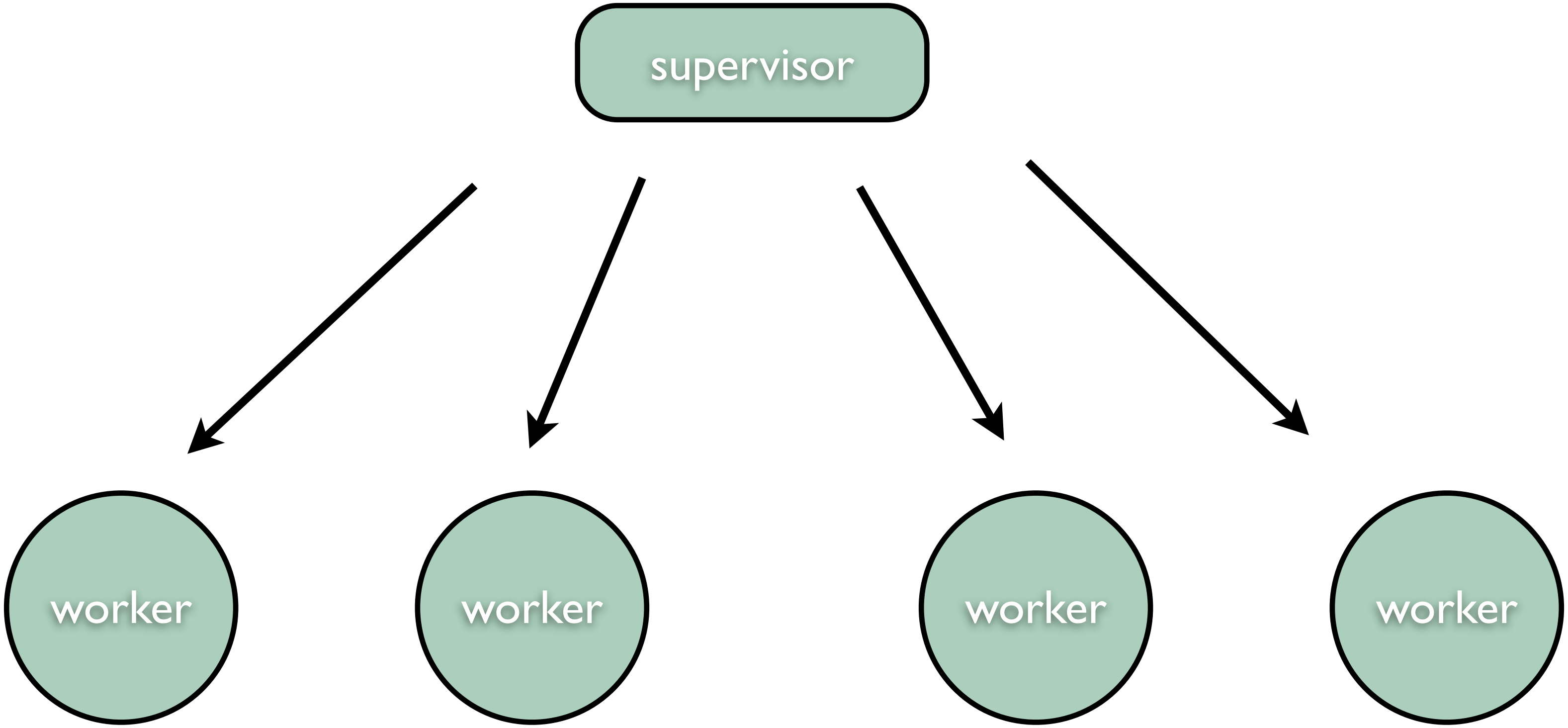
Create And Send Message

```java
// Create an Akka system
ActorSystem system = ActorSystem.create("MySystem");

// create a counter
final ActorRef counter =
    system.actorOf(new Props(Counter.class), "counter");

// send message to the counter
counter.tell("increase");
Future<Object> count = ask(counter, "get");
```

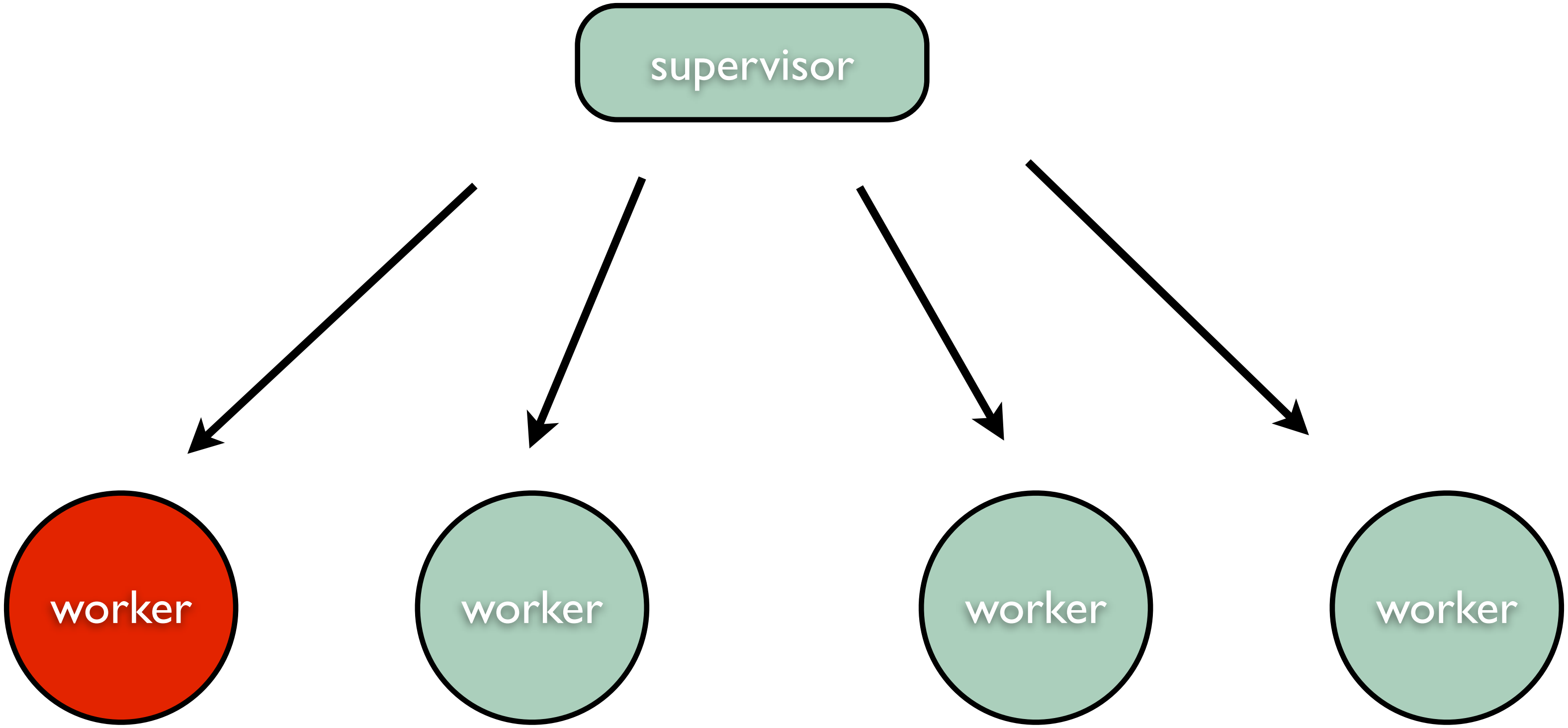
More on the Futures

```
// build a model for a EC site.
def doSearch(userId: String, keyword: String) {

  val sessionFuture = ask(sessionManager, GetSession(userId))
  val adFuture = ask(advertiser, GetAdvertisement)
  val resultFuture = ask(searcher, Search(keyword))

  val recommFuture = sessionFuture.map {
    session => ask(recommender,  Get(keyword, session))
  }

  val responseFuture = for {
    ad: Advertisement      <- adFuture
    result: SearchResult   <- resultFuture
    recomm: Recommendation <- recommFuture
  } yield new Model(ad, result, recomm)
  return responseFuture.get
}
```

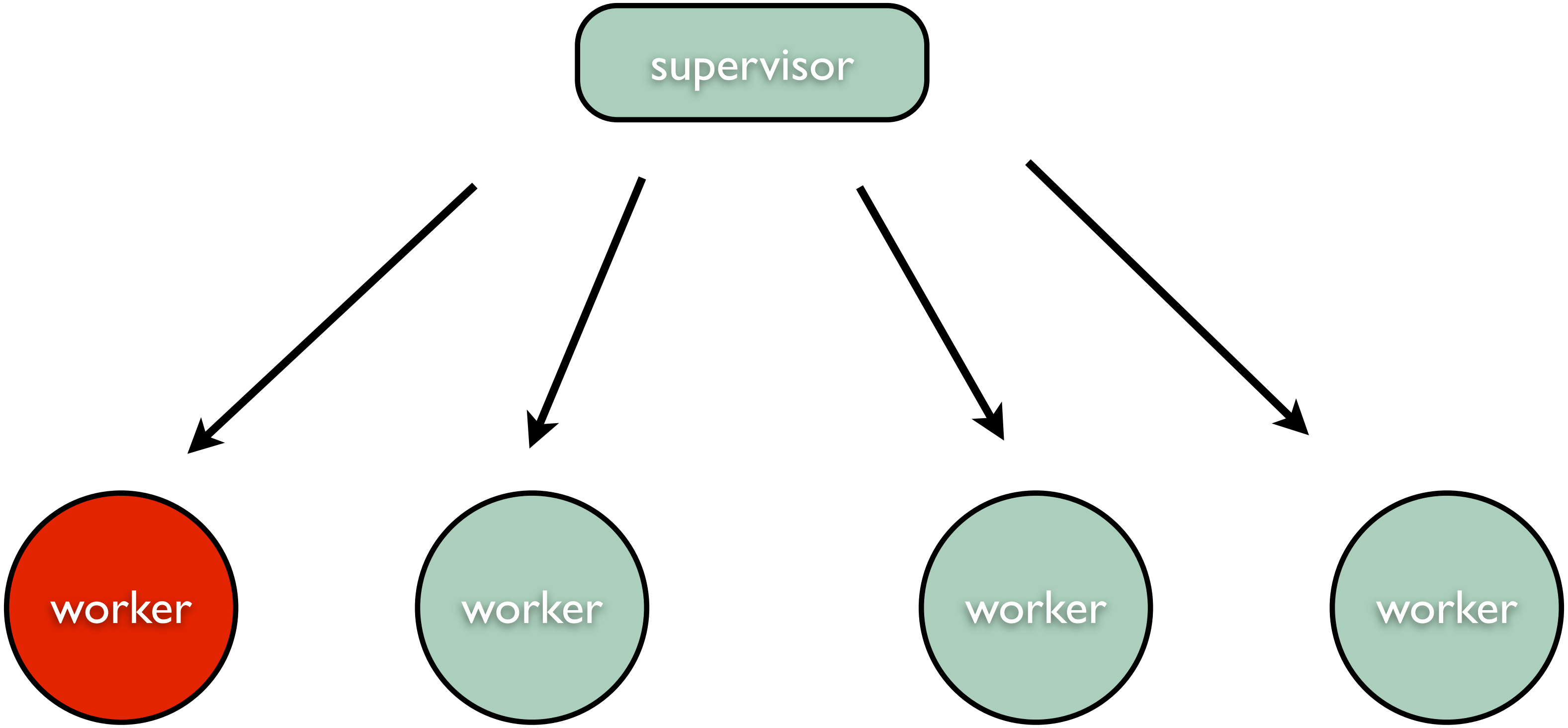
Fault Tolerance in Akka



Fault Tolerance in Akka

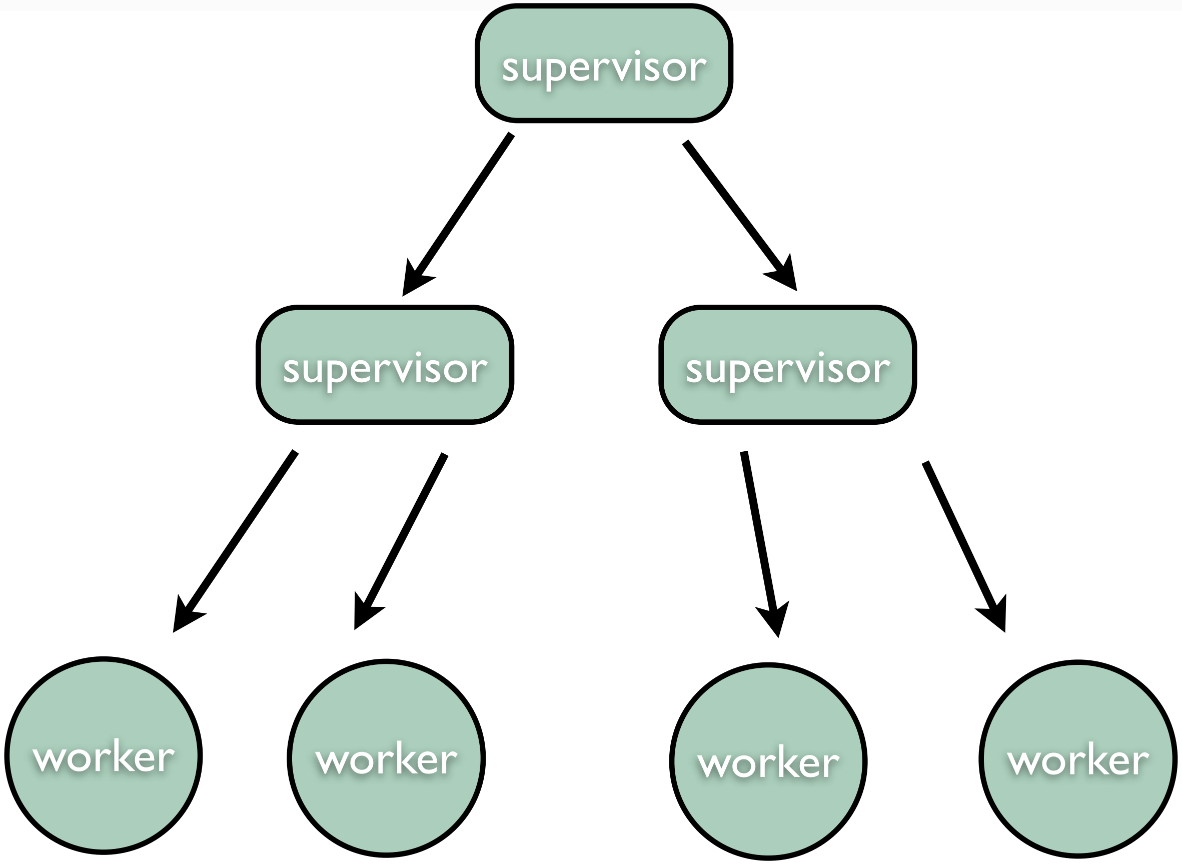


Fault Tolerance in Akka

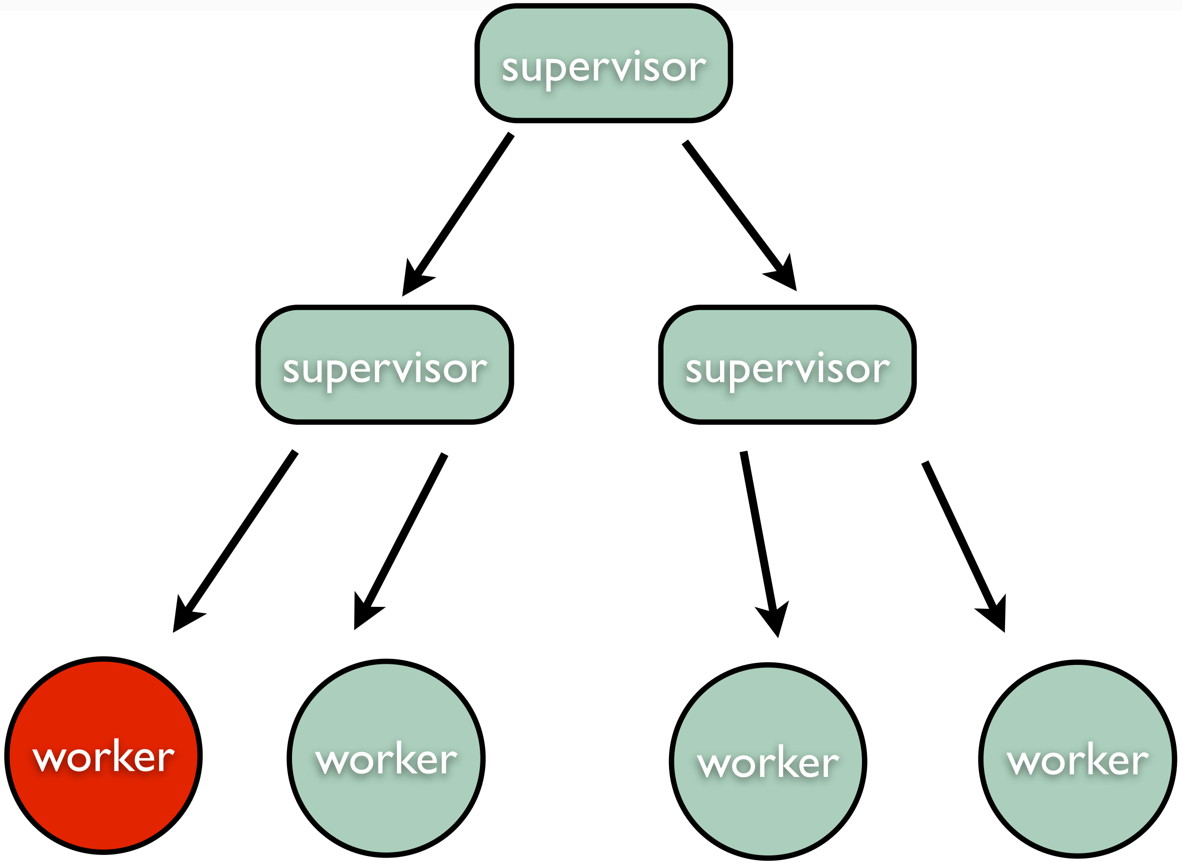


- One-For-One restart strategy
- One-For-All restart strategy

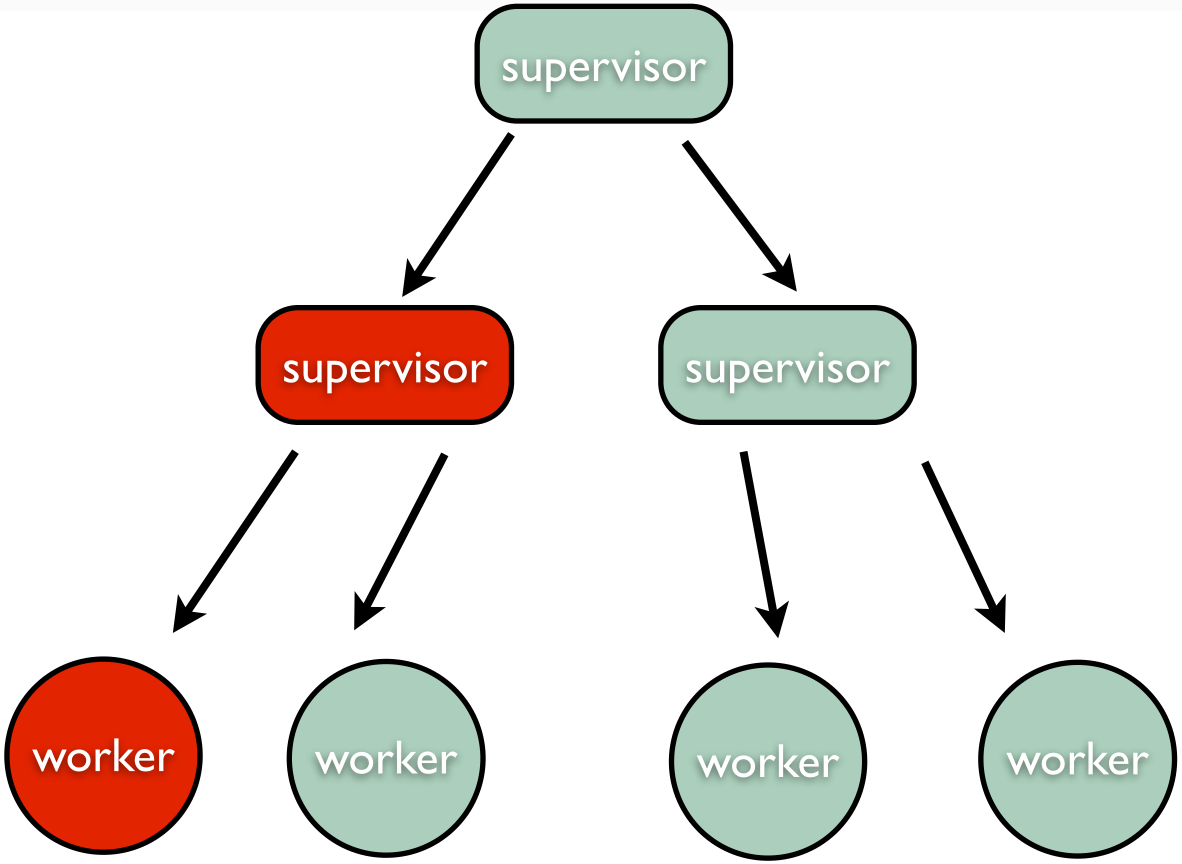
Fault Tolerance in Akka



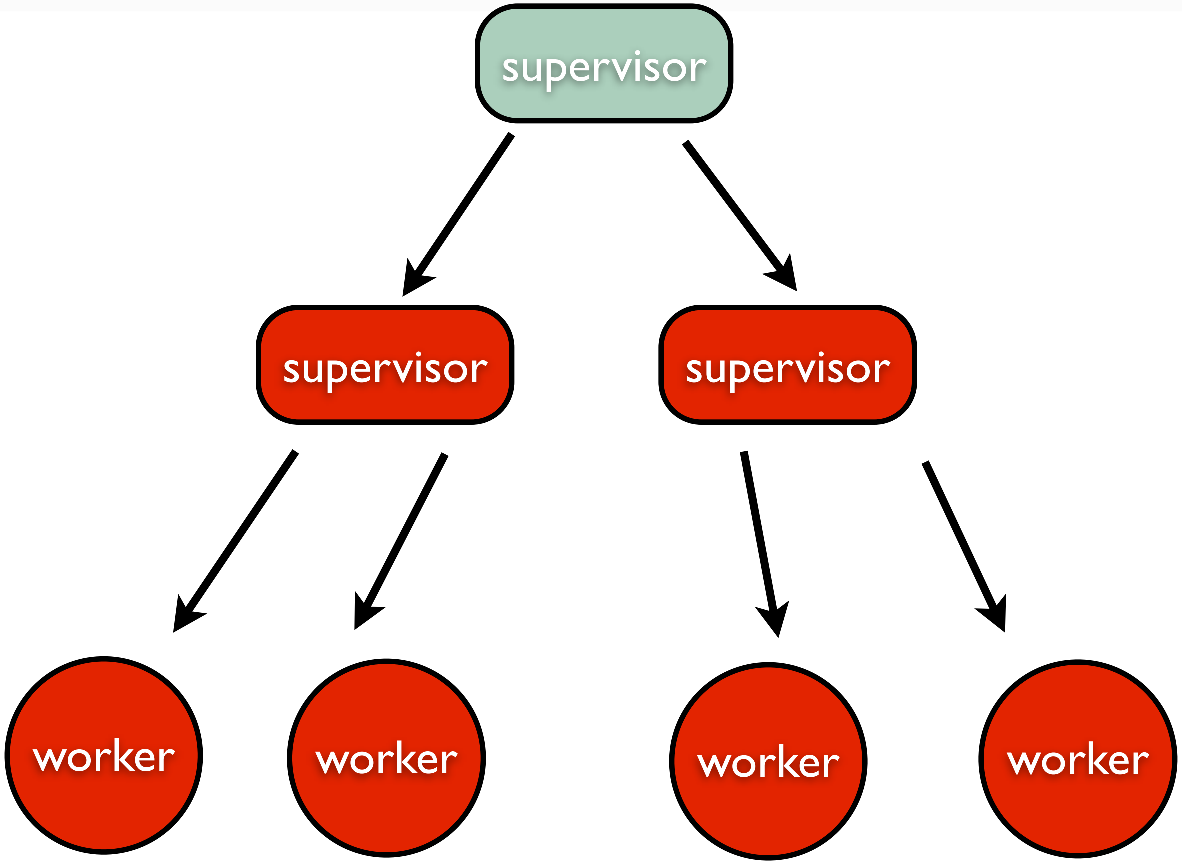
Fault Tolerance in Akka



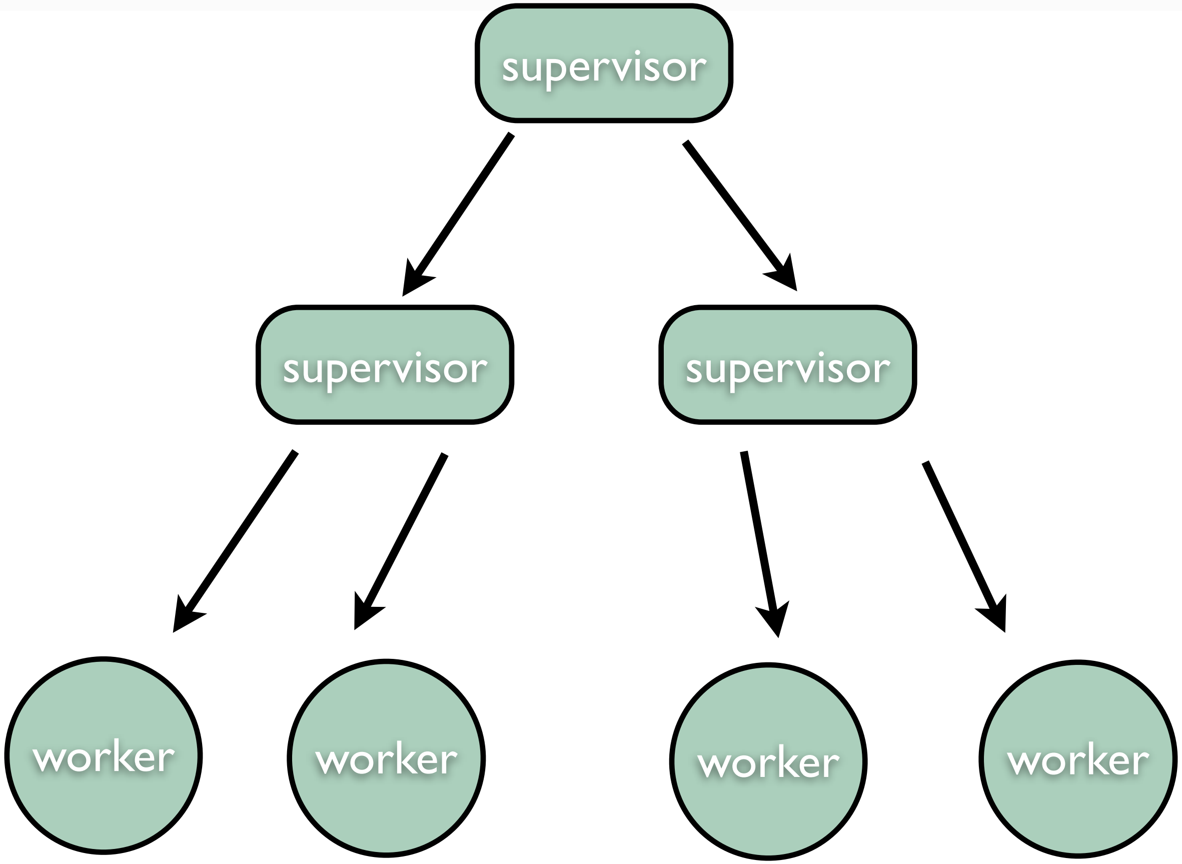
Fault Tolerance in Akka



Fault Tolerance in Akka



Fault Tolerance in Akka



Fault Tolerance in Akka

```
public class MySupervisor extends UntypedActor {
  // Restart the child if it throws ServiceUnavailable
  private static SupervisorStrategy strategy =
    new OneForOneStrategy(3, Duration.parse(“5 seconds”),
    new Function<Throwable, Directive>() {
      @Override
      public Directive apply(Throwable t) {
        if (t instanceof IOException) {
          return restart();
        } else {
          return escalate();
        }
      }
    });

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

Remote Actor

–Actors are location transparent and distributable by design.

–All Actors can be remote actor through configuration without any code changes.

–Sending message to a remote Actor is as simple as sending message to local Actor.

–Messages are serialized through java serialization, Protocol Buffer serializer or custom serializer. The desired behavior is configurable in the config file.

Remote Actor

```java
// define a remote address
Address addr =
  new Address("serializer", "MySystem", "host", 1234);

// initialize an actor on remote host programmatically.
ActorRef ref = system.actorOf(
  new Props(Counter.class)
    .withDeploy(
      new Deploy(new RemoteScope(addr)
    )
  )
);
```

Routing & Clustering

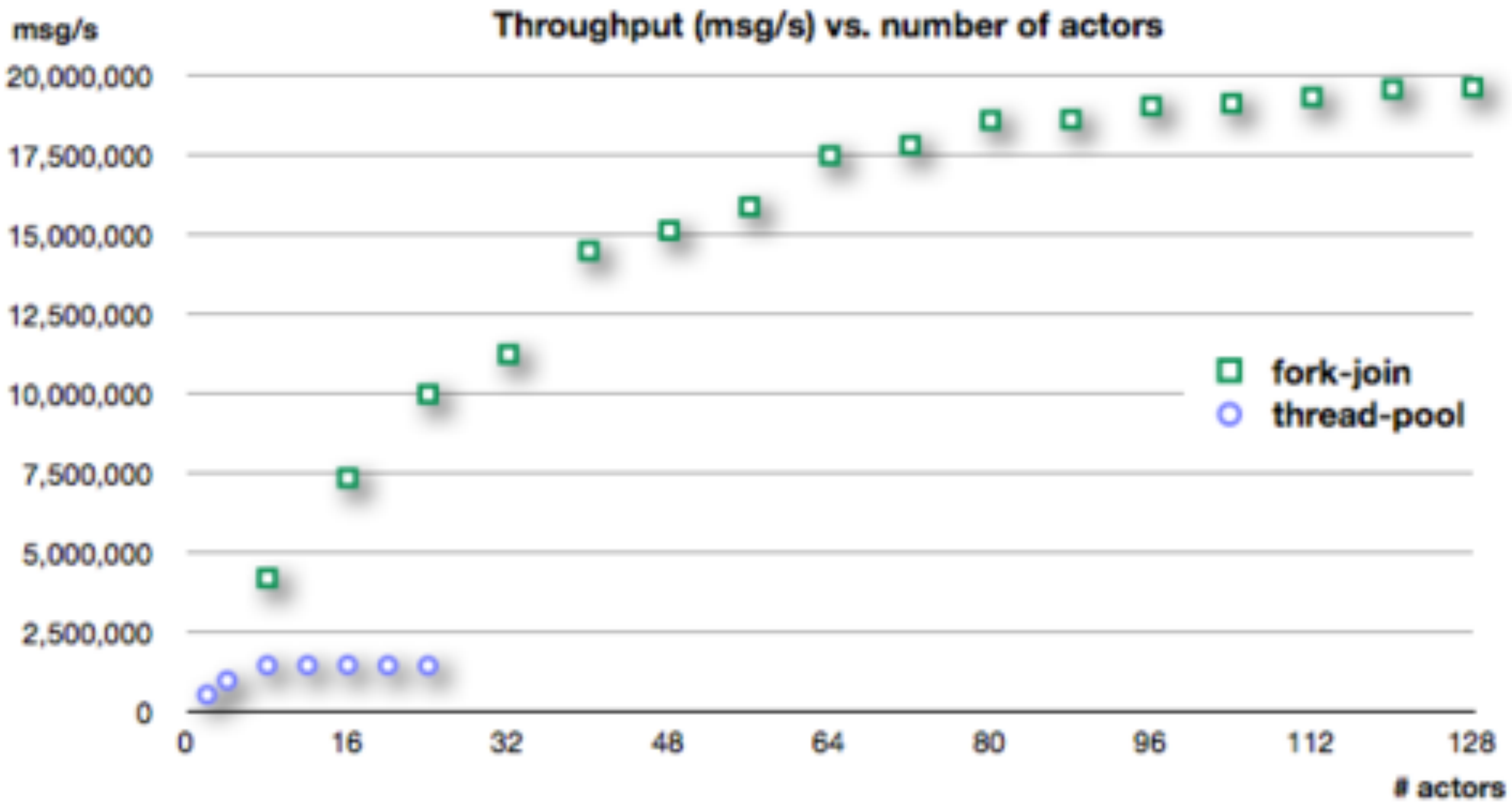
–Clustering support is still under construction and will be available in 2.1 release.

–A Router routes incoming messages to outbound actors.

- – RoundRobinRouter
- – RandomRouter
- – SmallestMailboxRouter
- – BroadcastRouter
- – ScatterGatherFirstCompletedRouter

```java
ActorRef router = system.actorOf(
  new Props(ExampleActor.class)
    .withRouter(new RoundRobinRouter(5))
);
```

Performance



ref: http://letitcrash.com/post/17607272336/scalability-of-fork-join-pool

Use Cases

- Event driven messaging system
- Stock trend analysis and simulation.
- Rule based engine.
- Multiplayer online games.

case study -
twitter-like messaging service

Messaging Service.

–Publisher
- keeps a list of reference to subscribers.
- when it receives a message, it will forward the message to subscribers.

–Subscribers
- stores received messages.

Protocol Classes

```java
public class Message implements Serializable {
    public final String sender;
    public final String message;
    public final DateTime createDate;
    //skipped...
}
public class GetMessages implements Serializable {
    public final DateTime since;
    //skipped...
}
public class Subscribe implements Serializable {
    public final ActorRef subscriber;
    //skipped...
}
```

The Actor

```
public class PubSubscriber extends UntypedActor {
  private final String name;
  private final List<Message> received = Lists.newArrayList();
  private final Set<ActorRef> subscribers = Sets.newHashSet();
  public PubSubscriber(String name) {
    this.name = name;
  }
  public void onReceive(Object message) {
    if (message instanceof Subscribe) {
      subscribers.add(((Subscribe) message).subscriber);
    } else if (message instanceof Message) {
      Message msg = (Message) message;
      // if sender is self, forward the message to subscriber.
      if (Objects.equal(msg.sender, name)) {
        for (ActorRef subscriber: subscribers) {
          subscriber.tell(msg);
        }
      } else {
        received.add((Message) message);
      }
```

The Actor

```
} else if (message instanceof GetMessages) {
    final DateTime since = ((GetMessages) message).since;
    Iterable<Message> ret = Iterables.filter(received,
        new Predicate<Message>() {
        @Override
        public boolean apply(@Nullable Message message) {
          return message.createDate.isAfter(since);
        }
      });
      getSender().tell(ret);
    } else {
      unhandled(message);
    }
  }
}
```

External Interface

–Akka-Camel

```
class JettyAdapter extends Consumer with ActorLogging {

  def endpointUri = "jetty:http://localhost:8080/"

  override def receive = {
    case CamelMessage(body, headers) => {
      headers.get("op") match {
        case Some("msg")   => handleMessagingOp(headers)
        case Some("get")   => handleGetOp(headers)
        case op            => handleUnsupportedOp(op)
      }
    }
  }
```

External Interface

```
private def handleMessagingOp(headers: Map[String, Any]) {
  val tweetOption = for(
    name  <- headers.get("name");
    msg  <- headers.get("msg")
  ) yield new Message(name.toString, msg.toString, DateTime.now)

  tweetOption match {
    case Some(message) => {
      findOrCreateActorRef(msg).forward(message)
   }
   case None => {
     sender ! "Unable to perform Action."
   }
}}
private def findOrCreateActorRef(name: String): ActorRef = {
  val pubsub = context.actorFor(name)
  if (pubsub.isTerminated) {
    context.actorOf(Props(new PubSubscriber(name)), name = name)
  } else { pubsub }
}
```

Handle Server Shutdown

–When server stops, we need to persist state to external storage.

- actors' state
- unprocessed messages in mail boxes.

–For actor's state, you can implement preStart and postStop method to persiste state to external storage.

–For unprocessed message, Akka provides durable mail box backed by local file system.

Going Remote.

- There is no code changes to the PubSubscriber or protocol classes.
  - The protocol classes are serializable and immutable already.
  - The subscriber reference, the ActorRef, is remote ready too.

- The only missing piece is the one connects the actors. We need to rewrite the findOrCreateActor() method.
  - In Akka 2.1 release, it will provide a new cluster module to solve this issue.

Q&A

yunglin@gmail.com
twitter: @yunglinho