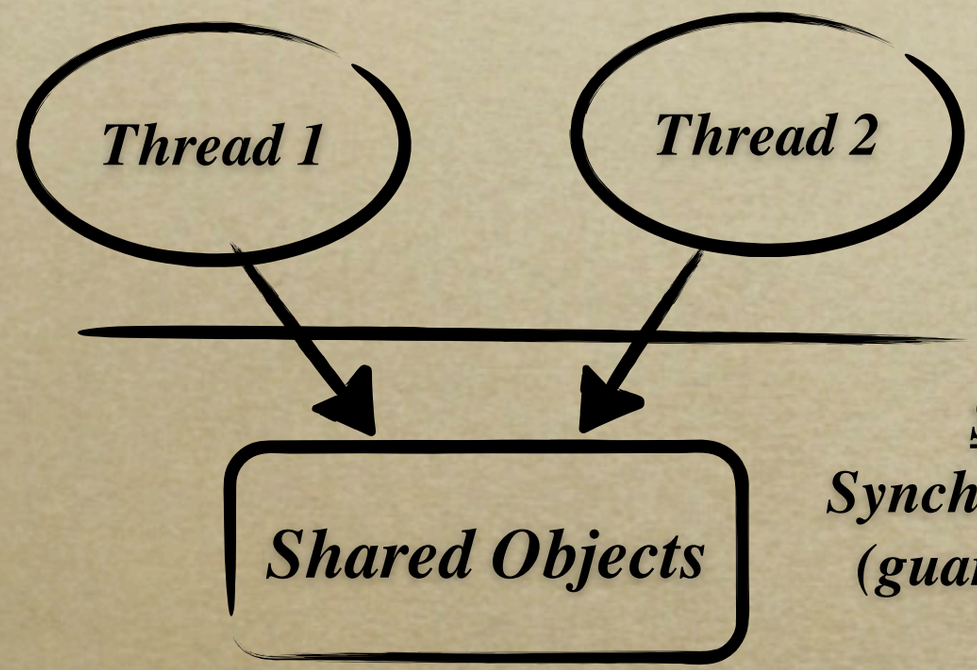
Akka

Building Distributed Systems for
Concurrent, Fault-tolerant and Scalable Java Applications

Problems with Multithreaded Programming

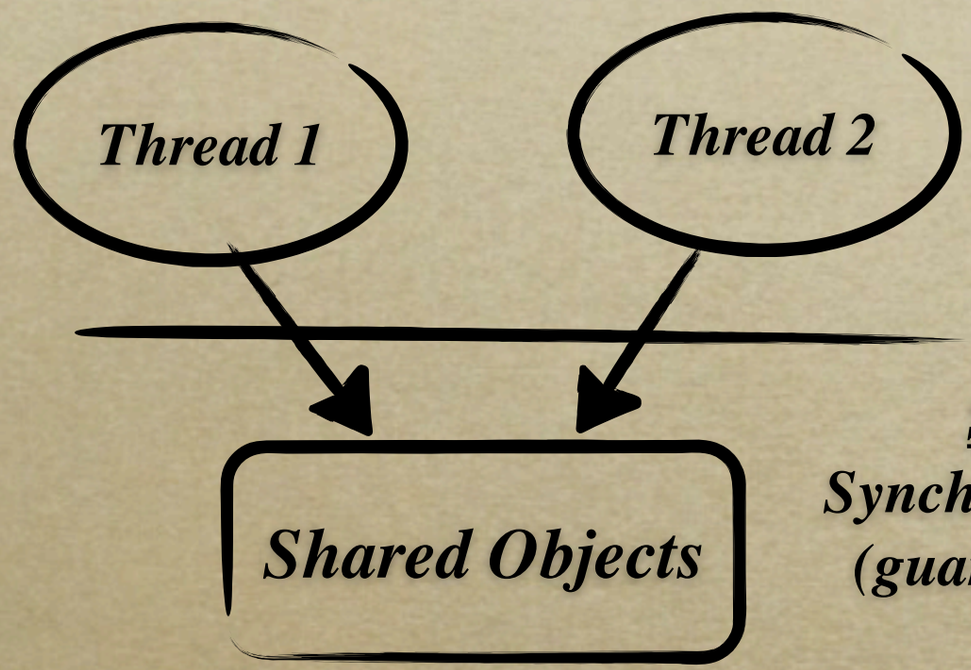
Shared Mutable Objects



Solution:
Synchronized Access
(guarded by locks)

Problems with Multithreaded Programming

Shared Mutable Objects

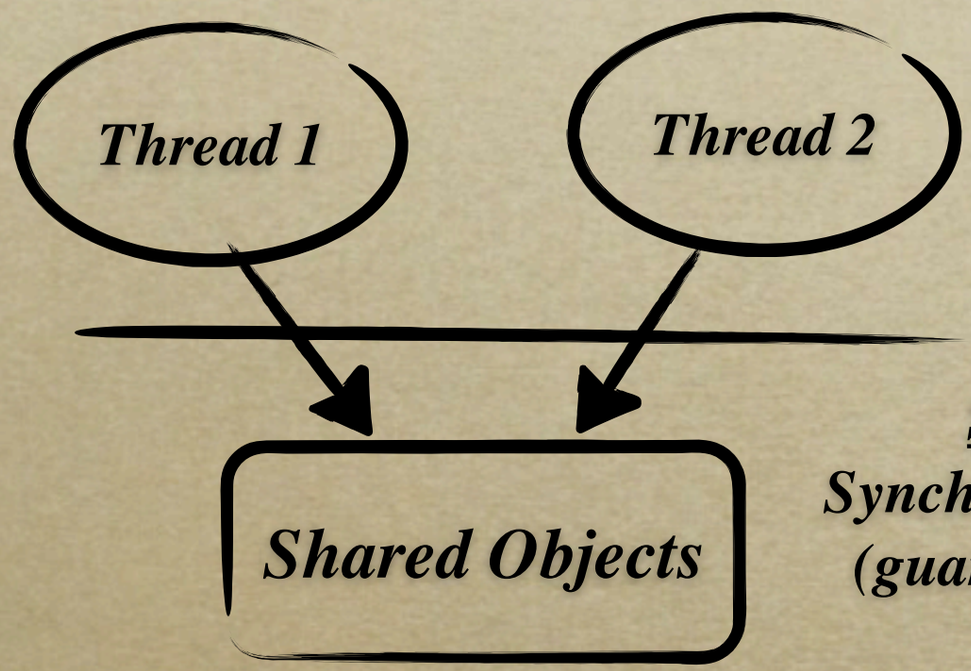


Solution:
Synchronized Access
(guarded by locks)

Change in one part of the system may break it in another part.

Problems with Multithreaded Programming

Shared Mutable Objects



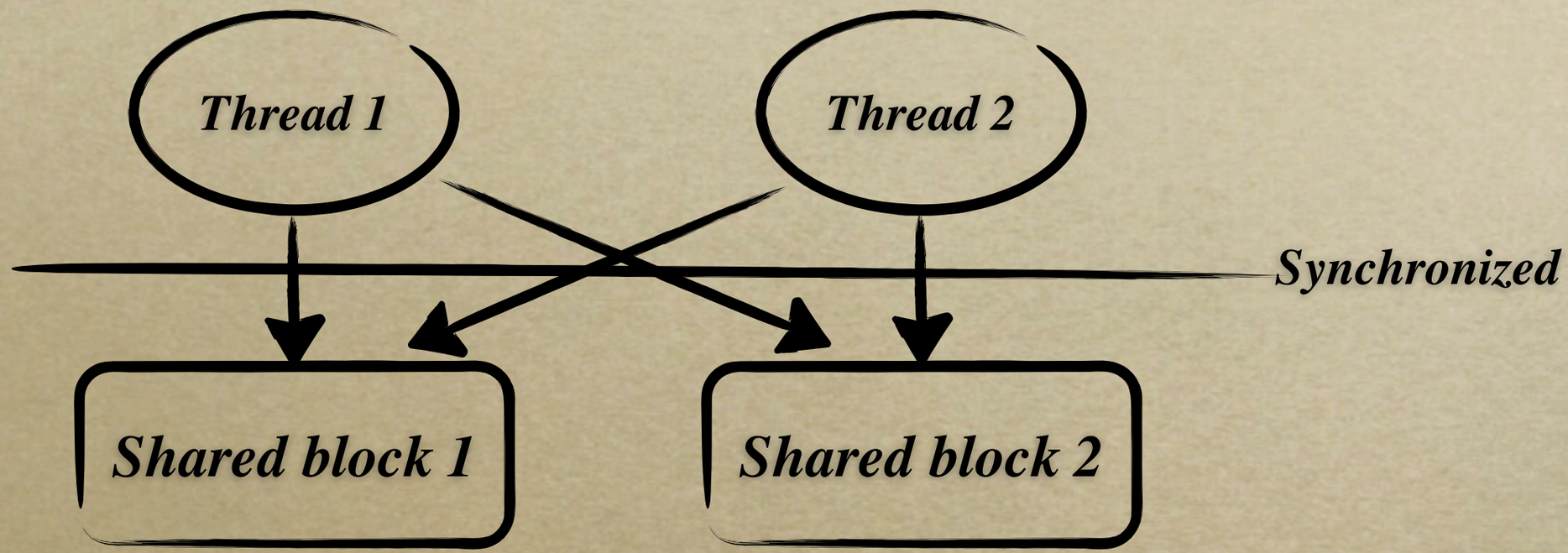
Solution:
Synchronized Access
(guarded by locks)

Change in one part of the system may break it in another part.

1. Operations may occur serially -> poor performance
2. Locks
-- do not compose -> hard to design applications
-- do not scale well -> block the execution
-- hard to get in right order and error recovery is complicated

Problems with Multithreaded Programming

Deadlocks



Thread 1 has obtained the lock to synchronized block 1
and
Thread 2 has obtained the lock to synchronized block 2

Problems with Multithreaded Programming

Scalability

Managing multiple threads in a single JVM → ***Challenging***

Scaling the application across multiple JVMs?

Problems with Multithreaded Programming

Scalability

Managing multiple threads in a single JVM → ***Challenging***

Scaling the application across multiple JVMs?

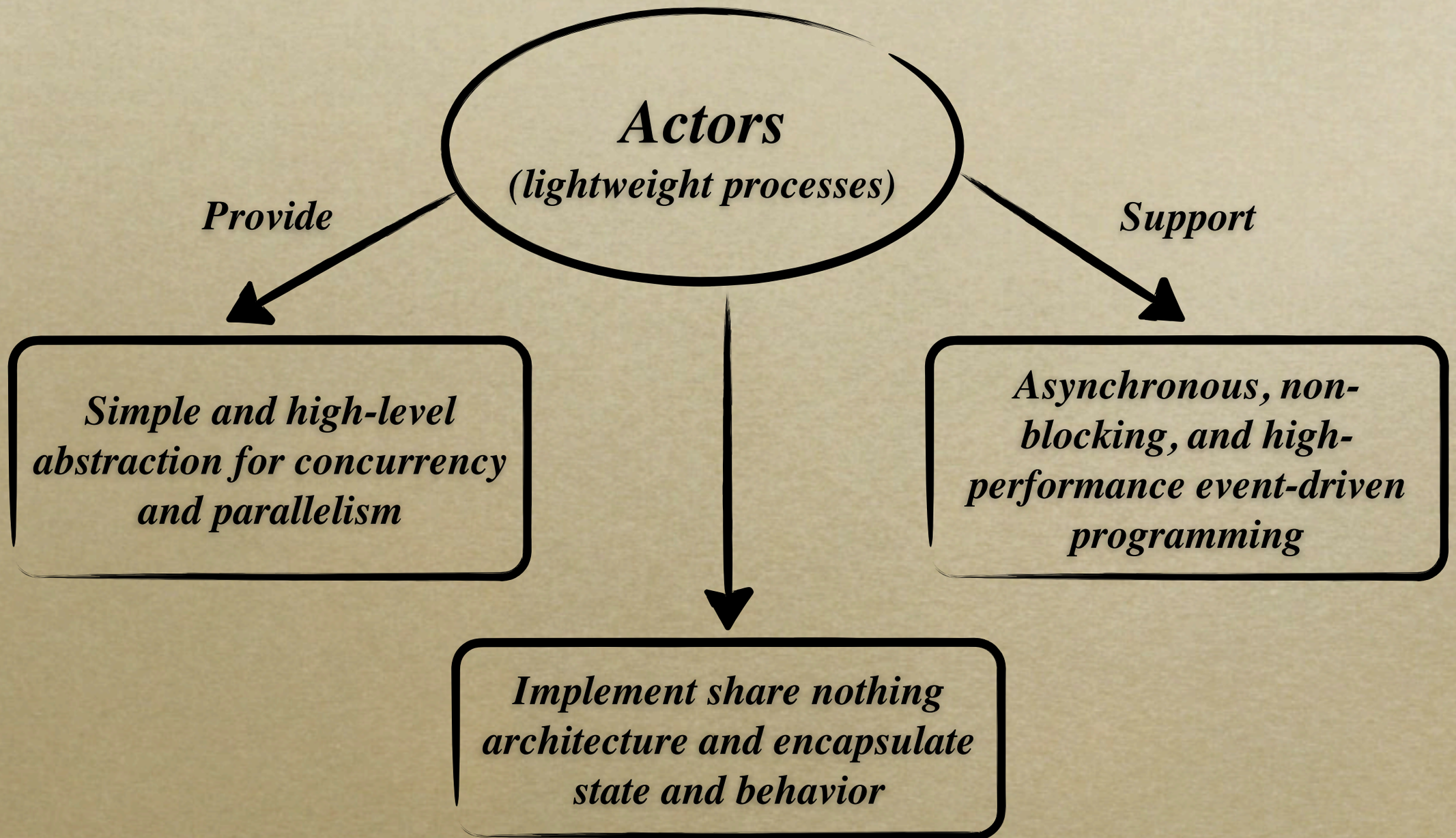
↓

Shared state stored in database
and
Relying on database to manage the concurrent access

Akka: A Solution for Concurrency, Fault-tolerance and Scalability

- ***Open source toolkit and runtime that runs on JVM***
- ***Written in Scala***
- ***May use Scala or Java to call libraries and features***
- ***High level abstraction for concurrency: Actors combined with software transaction memory (SMT) to implement atomic message-passing***

Actors in Akka



“Let it Crash” Model for Fault-tolerance

standard java application

critically important
state is guarded by
try/catch blocks

```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

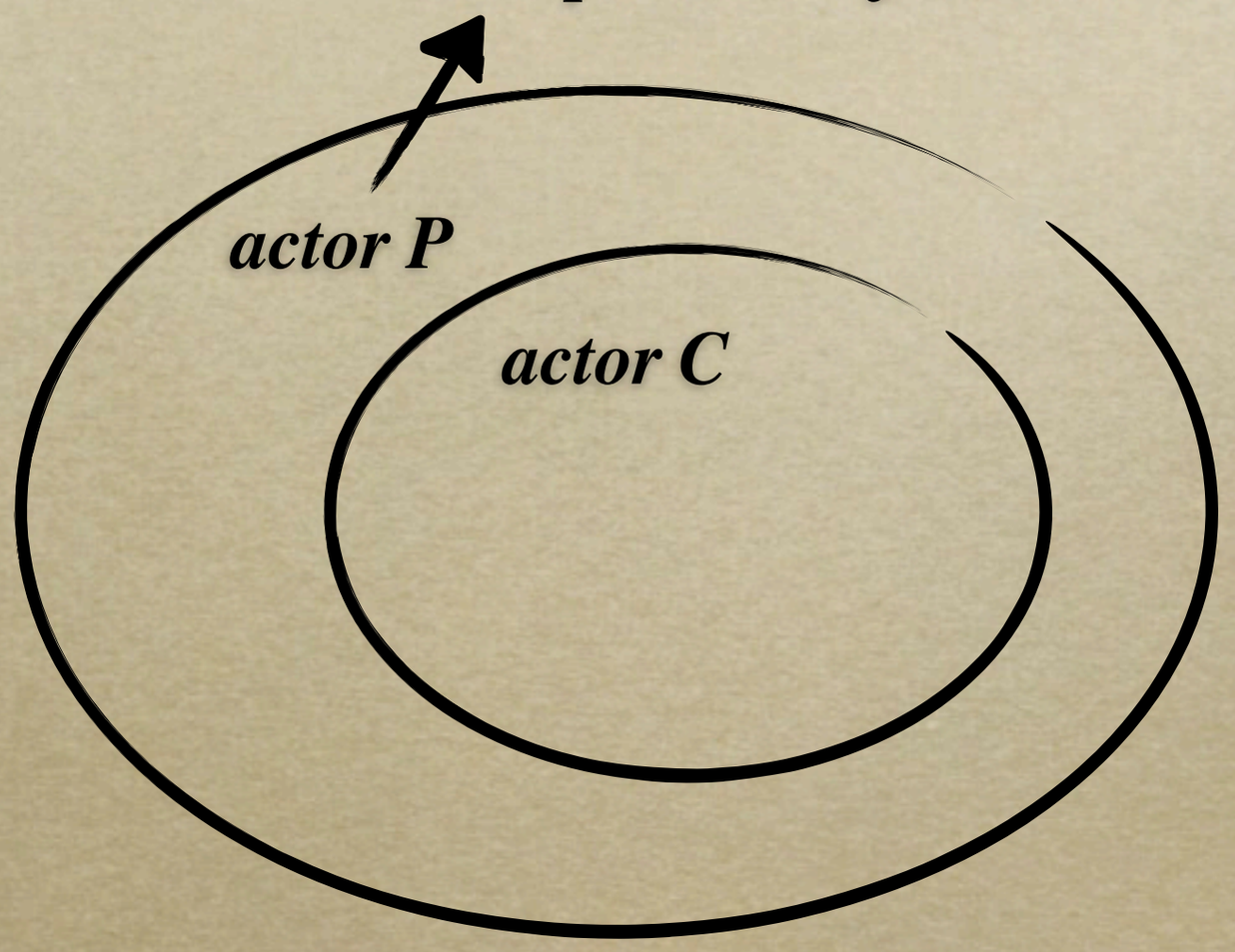
“Let it Crash” Model for Fault-tolerance

standard java application

```
critically important
state is guarded by
try/catch blocks
```

```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

Akka



supervisor hierarchy

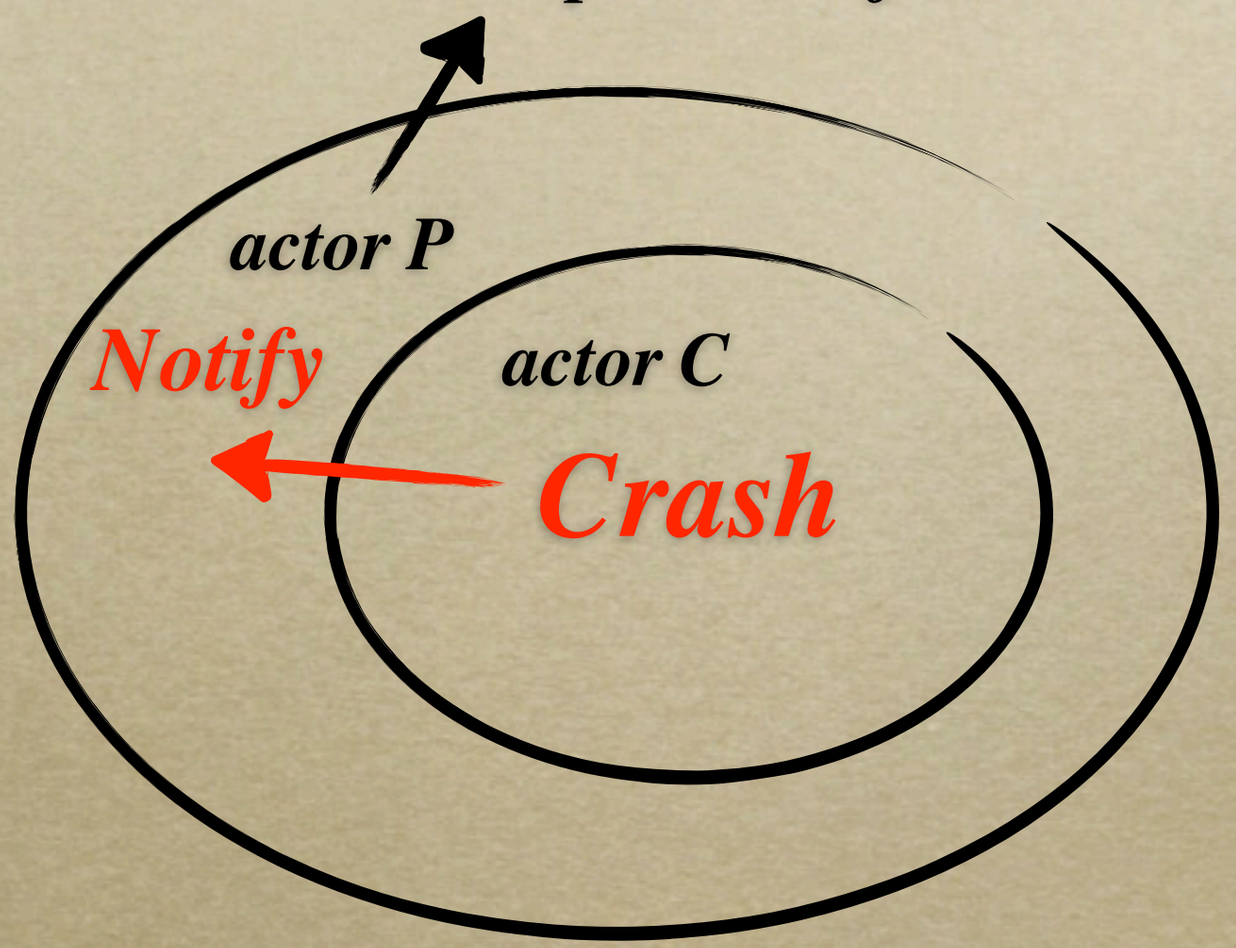
“Let it Crash” Model for Fault-tolerance

standard java application

```
critically important
state is guarded by
try/catch blocks
```

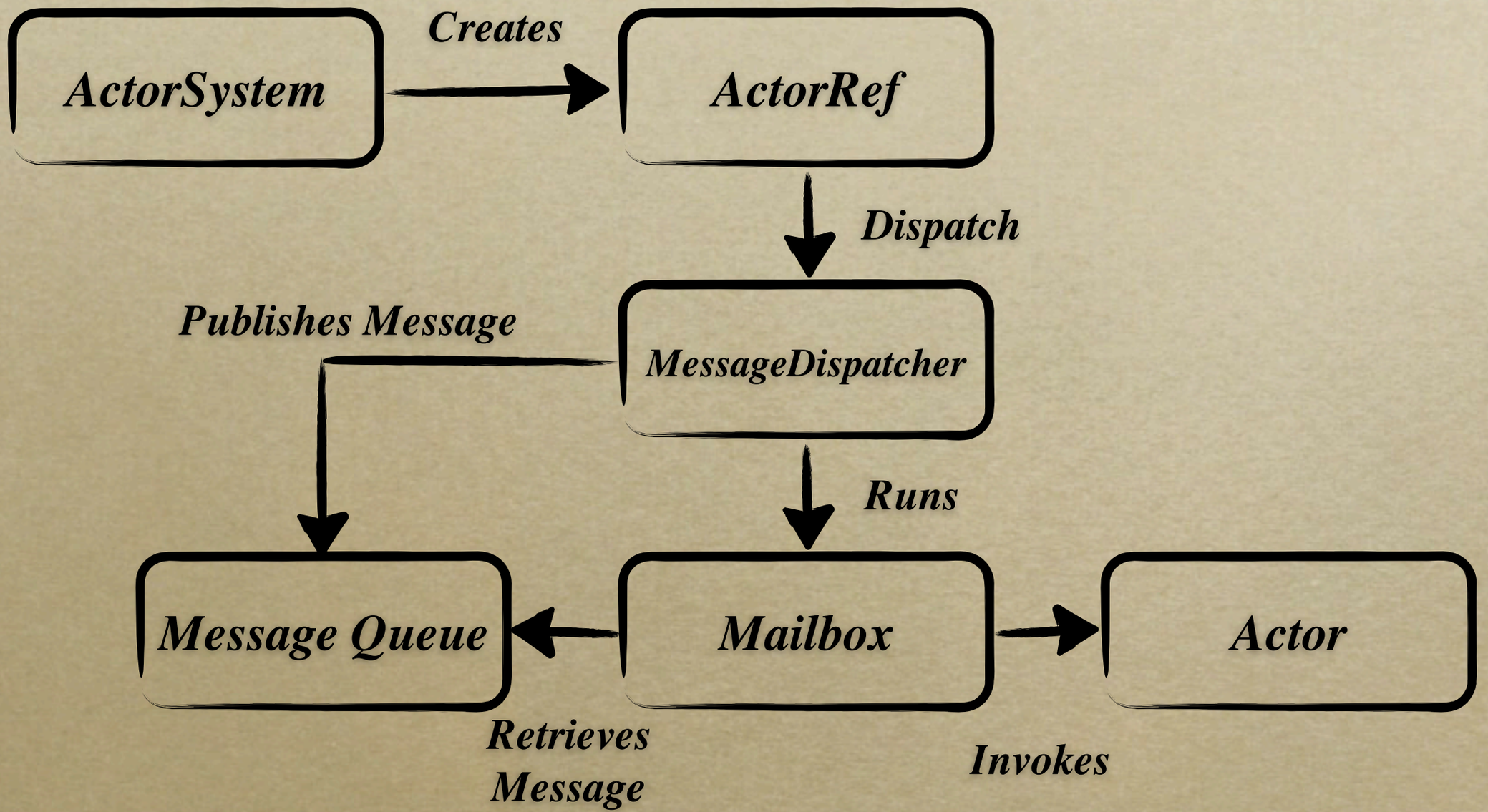
```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

Akka



The Actor Model

The Actor Model



Some More Advantages

MessageDispatcher

Load balancing

-- can maintain a thread pool having limited number of threads.

-- can be configured with one-to-one mapping of threads to actors.

Some More Advantages

MessageDispatcher

Load balancing

-- can maintain a thread pool having limited number of threads.
-- can be configured with one-to-one mapping of threads to actors.

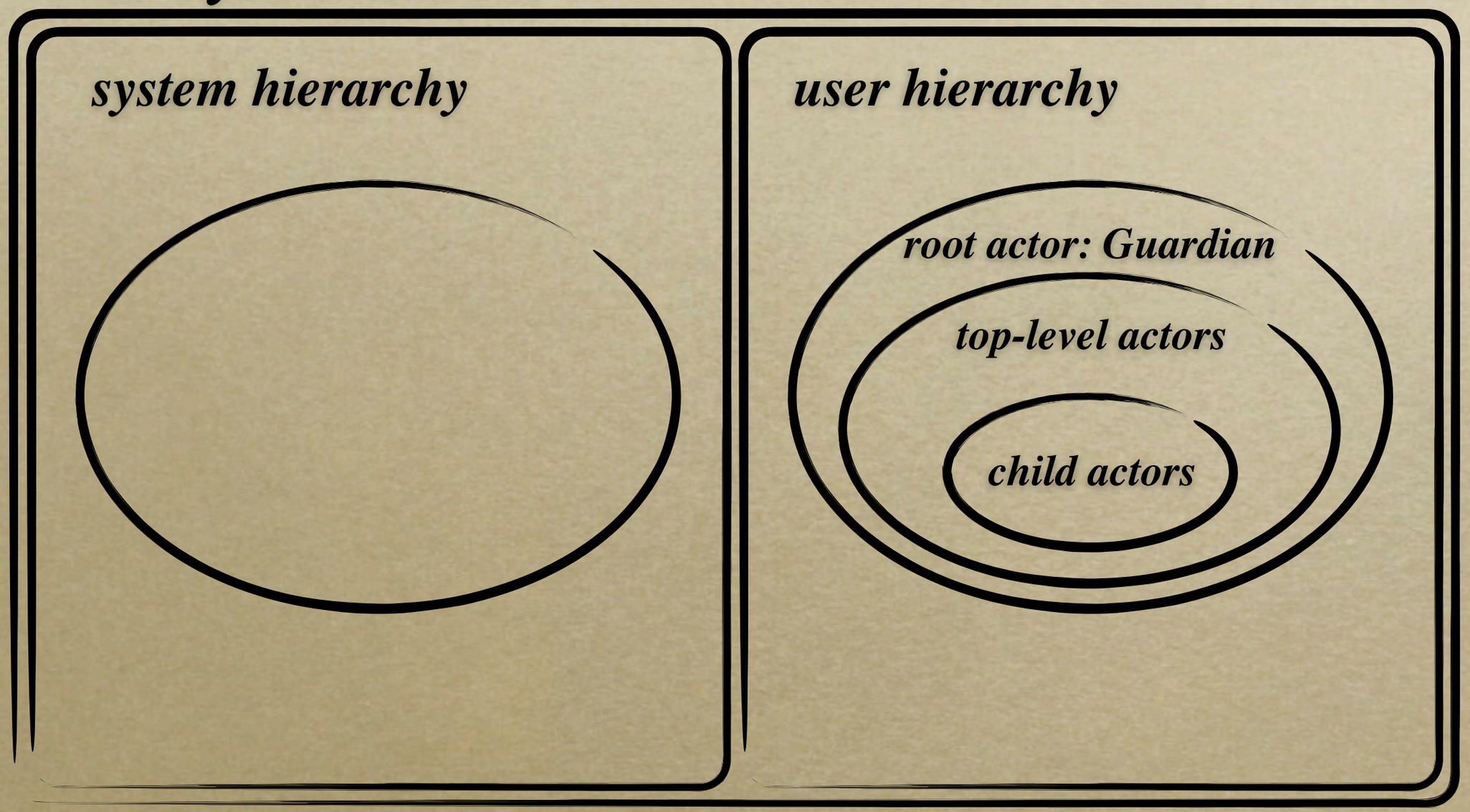
ActorRef

Horizontal scalability

Local ActorRef

Remote ActorRef

Actor Hierarchies



Actor hierarchy

Example: Computing Prime Numbers

- *Create a master actor*
- *Create a round-robin router to distribute work across multiple worker actors*
- *Communicate between worker actors and the master actor*
- *Communicate between the master actor and a listner*

```java
package com.geekcap.akka.prime.message;

import java.io.Serializable;

public class NumberRangeMessage implements Serializable
{
    private long startNumber;
    private long endNumber;

    public NumberRangeMessage(long startNumber, long endNumber) {
        this.startNumber = startNumber;
        this.endNumber = endNumber;
    }


    public long getStartNumber() {
        return startNumber;
    }

    public void setStartNumber(long startNumber) {
        this.startNumber = startNumber;
    }

    public long getEndNumber() {
        return endNumber;
    }

    public void setEndNumber(long endNumber) {
        this.endNumber = endNumber;
    }
}
```

```java
package com.geekcap.akka.prime.message;

import java.util.ArrayList;
import java.util.List;

public class Result
{
    private List<Long> results = new ArrayList<Long>();

    public Result()
    {
    }

    public List<Long> getResults()
    {
        return results;
    }
}
```

```java
package com.geekcap.akka.prime;

import akka.actor.UntypedActor;
import com.geekcap.akka.prime.message.NumberRangeMessage;
import com.geekcap.akka.prime.message.Result;

public class PrimeWorker extends UntypedActor
{
    /**
     * Invoked by the mailbox when it receives a thread timeslice and a message is
     *
     * @param message   The message to process
     */
    public void onReceive( Object message )
    {
        // We only handle NumberRangeMessages
        if( message instanceof NumberRangeMessage )
        {
            // Cast the message to a NumberRangeMessage
            NumberRangeMessage numberRangeMessage = ( NumberRangeMessage )message;
            System.out.println( "Number Rage: " + numberRangeMessage.getStartNumber() + " to " + numberRangeMessage.getEndNumber() );

            // Iterate over the range, compute primes, and return the list of numbers that are prime
            Result result = new Result();
            for( long l = numberRangeMessage.getStartNumber(); l <= numberRangeMessage.getEndNumber(); l++ )
            {
                if( isPrime( l ) )
                {
                    result.getResults().add( l );
                }
            }

            // Send a notification back to the sender
            getSender().tell( result, getSelf() );
        }
        else
        {
            // Mark this message as unhandled
            unhandled( message );
        }
    }
```

```java
    /**
     * Returns true if n is prime, false otherwise
     *
     * @param n     The long to check
     *
     * @return      True if n is prime, false otherwise
     */
    private boolean isPrime( long n )
    {
        if( n == 1 || n == 2 || n == 3 )
        {
            return true;
        }

        // Is n an even number?
        if( n % 2 == 0 )
        {
            return false;
        }

        //if not, then just check the odds
        for( long i=3; i*i<=n; i+=2 )
        {
            if( n % i == 0)
            {
                return false;
            }
        }
        return true;
    }
}
```

```java
package com.geekcap.akka.prime;

import akka.actor.ActorRef;
import akka.actor.Props;
import akka.actor.UntypedActor;
import akka.routing.RoundRobinRouter;
import com.geekcap.akka.prime.message.NumberRangeMessage;
import com.geekcap.akka.prime.message.Result;

import java.util.List;

public class PrimeMaster extends UntypedActor
{
    private final ActorRef workerRouter;
    private final ActorRef listener;

    private final int numberOfWorkers;
    private int numberOfResults = 0;

    private Result finalResults = new Result();

    public PrimeMaster( final int numberOfWorkers, ActorRef listener )
    {
        // Save our parameters locally
        this.numberOfWorkers = numberOfWorkers;
        this.listener = listener;

        // Create a new router to distribute messages out to 10 workers
        workerRouter = this.getContext()
                .actorOf( new Props(PrimeWorker.class )
                        .withRouter( new RoundRobinRouter( numberOfWorkers )), "workerRouter" );
    }

    @Override
    public void onReceive( Object message )
    {
        if( message instanceof NumberRangeMessage )
        {
            // We have a new set of work to perform
            NumberRangeMessage numberRangeMessage = ( NumberRangeMessage )message;

            // Just as a demo: break the work up into 10 chunks of numbers
            long numberOfNumbers = numberRangeMessage.getEndNumber() - numberRangeMessage.getStartNumber();
            long segmentLength = numberOfNumbers / 10;
```

```
            for( int i=0; i<numberOfWorkers; i++ )
            {
                // Compute the start and end numbers for this worker
                long startNumber = numberRangeMessage.getStartNumber() + ( i * segmentLength );
                long endNumber = startNumber + segmentLength - 1;

                // Handle any remainder
                if( i == numberOfWorkers - 1 )
                {
                    // Make sure we get the rest of the list
                    endNumber = numberRangeMessage.getEndNumber();
                }

                // Send a new message to the work router for this subset of numbers
                workerRouter.tell( new NumberRangeMessage( startNumber, endNumber ), getSelf() );
            }
        }
        else if( message instanceof Result )
        {
            // We received results from our worker: add its results to our final results
            Result result = ( Result )message;
            finalResults.getResults().addAll( result.getResults() );

            if( ++numberOfResults >= 10 )
            {
                // Notify our listener
                listener.tell( finalResults, getSelf() );

                // Stop our actor hierarchy
                getContext().stop( getSelf() );
            }

        }
        else
        {
            unhandled( message );
        }
    }
}
```

```java
package com.geekcap.akka.prime;

import akka.actor.UntypedActor;
import com.geekcap.akka.prime.message.Result;


public class PrimeListener extends UntypedActor
{
    @Override
    public void onReceive( Object message ) throws Exception
    {
        if( message instanceof Result )
        {
            Result result = ( Result )message;

            System.out.println( "Results: " );
            for( Long value : result.getResults() )
            {
                System.out.print( value + ", " );
            }
            System.out.println();

            // Exit
            getContext().system().shutdown();
        }
        else
        {
            unhandled( message );
        }
    }
}
```

```java
package com.geekcap.akka.prime;

import akka.actor.*;
import com.geekcap.akka.prime.message.NumberRangeMessage;

public class PrimeCalculator
{
    public void calculate( long startNumber, long endNumber )
    {
        // Create our ActorSystem, which owns and configures the classes
        ActorSystem actorSystem = ActorSystem.create( "primeCalculator" );

        // Create our listener
        final ActorRef primeListener = actorSystem.actorOf( new Props( PrimeListener.class ), "primeListener" );

        // Create the PrimeMaster: we need to define an UntypedActorFactory so that we can control
        // how PrimeMaster instances are created (pass in the number of workers and listener reference
        ActorRef primeMaster = actorSystem.actorOf( new Props( new UntypedActorFactory() {
            public UntypedActor create() {
                return new PrimeMaster( 10, primeListener );
            }
        }), "primeMaster" );

        // Start the calculation
        primeMaster.tell( new NumberRangeMessage( startNumber, endNumber ) );
    }

    public static void main( String[] args )
    {
        if( args.length < 2 )
        {
            System.out.println( "Usage: java com.geekcap.akka.prime.PrimeCalculator <start-number> <end-number>" );
            System.exit( 0 );
        }

        long startNumber = Long.parseLong( args[ 0 ] );
        long endNumber = Long.parseLong( args[ 1 ] );

        PrimeCalculator primeCalculator = new PrimeCalculator();
        primeCalculator.calculate( startNumber, endNumber );
    }
}
```

Exercise

- *Find GCD for a given set of numbers by creating 2 worker_actors and dividing the load among them equally.*
- *Report the time for computation taken by each worker_actor.*
- *Use Akka release 2.2.1*