Tech Stack Intro

LIU Shufa

2016.3

Incomplete tech stack

- Mybatis
- Spring
- Akka
- Finagle/Thrift
- Jesque
- Zk/Consul
- Miscellanies
  - Guava, Lombok

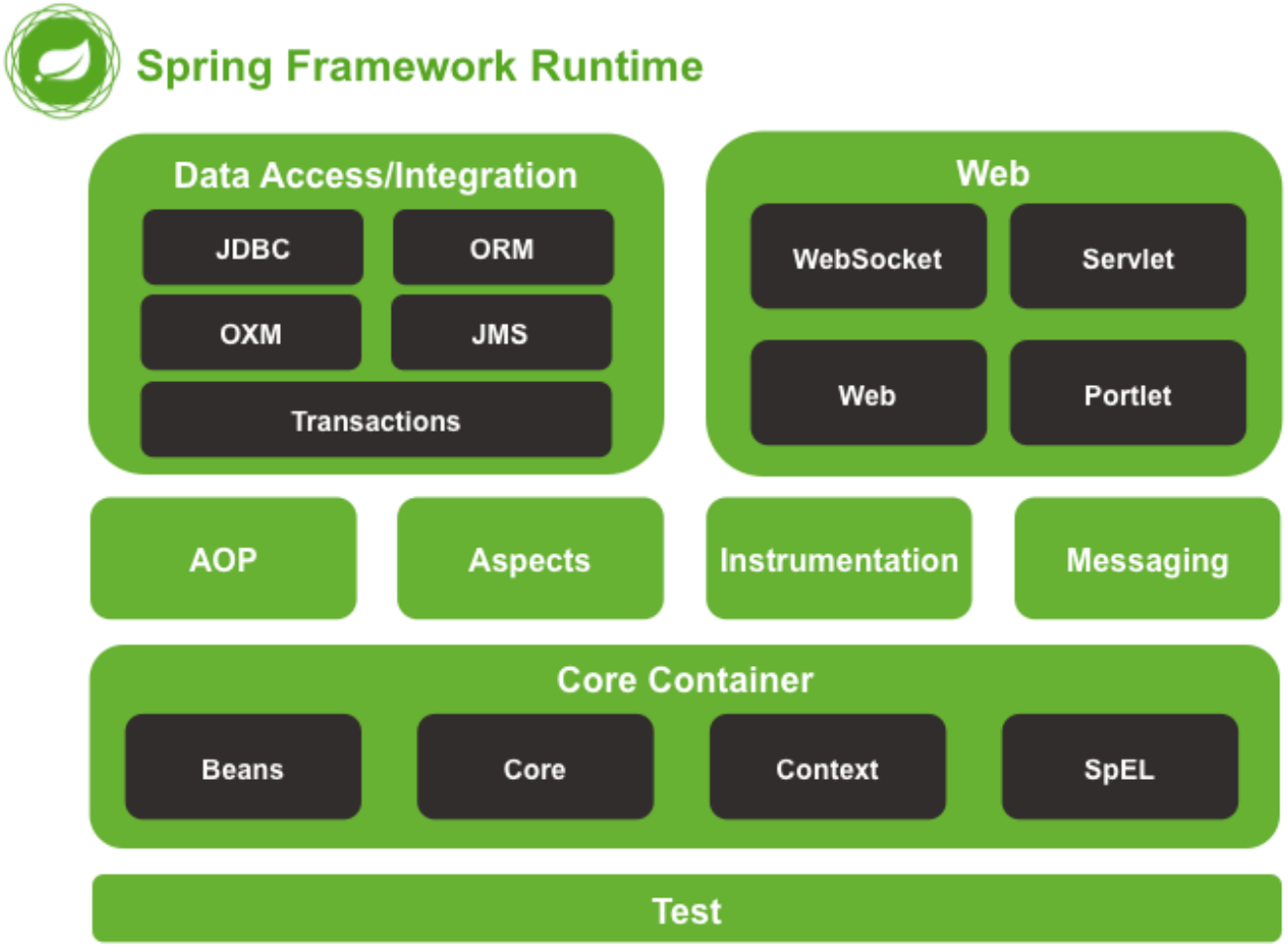
Mybatis

- Mybatis migration
  - mvn migration:up
- Mybatis generator
  - mvn mybatis-generator:generate
- Mybatis ORM

Spring

- Comprehensive library/framework in Java:
- Dependency Injection
- Aspect Oriented Programming
- MVC web framework

- Spring-boot: quick start for spring application


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Akka

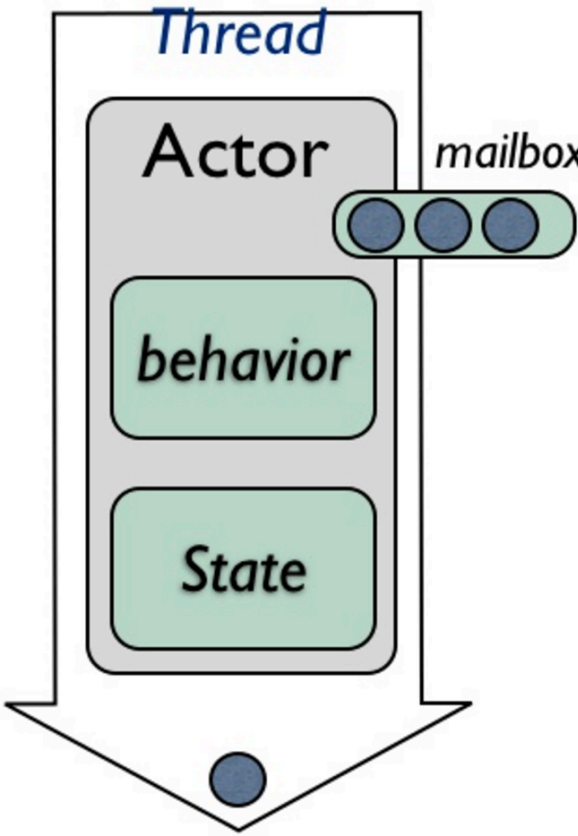
Actor Model

- Is a computational entity
  - State
  - Behavior
  - Message
- When received a message, an actor can
  - Create other actors
  - Send message to other actors
  - Change behavior

Actor

- An actor changes state and behavior only by received messages
- An actor processes messages sequentially
- An actor is thread safe

- Concurrency is achieved by multiple actors



Akka

- Actor implementation on JVM (Scala/Java API)
- Light-weight actors, high performance
  - 50m/s messages on a single machine
  - 2.5m actors per GB heap
- Distribution (remoting)
- Fault tolerance (supervision)

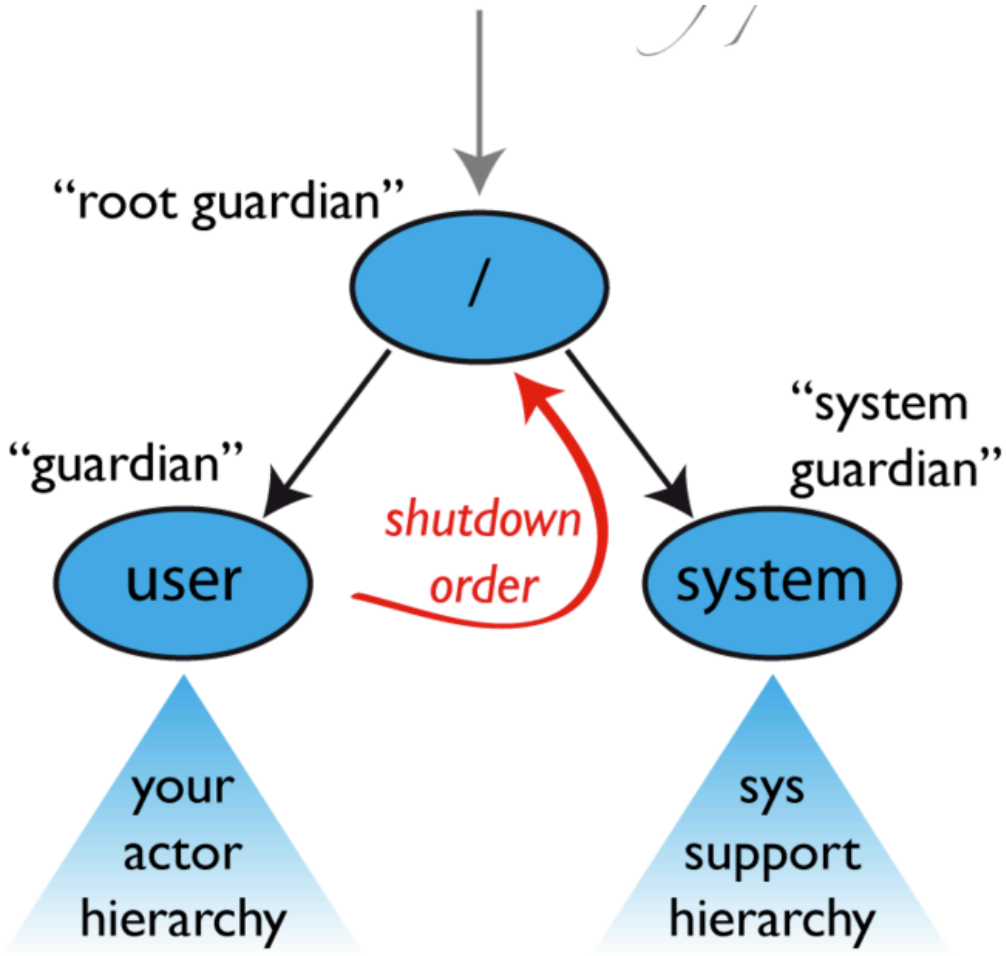
Actor

```
public class ExeReportActor extends UntypedActor {
    private int count = 0;
    @Override
    public SupervisorStrategy supervisorStrategy() {
      return new OneForOneStrategy(10, Duration.create("1 minute"),
        new Function<Throwable, Directive>() {
          @Override
          public Directive apply(Throwable t) {
            return restart();
          }
        };
    }
   @Override
   public void onReceive(Object message) throws Exception {
      count++;
      if (message instanceof ExecutionReport) {
        childrenActor.tell(message, this);
      }
   }
}
```

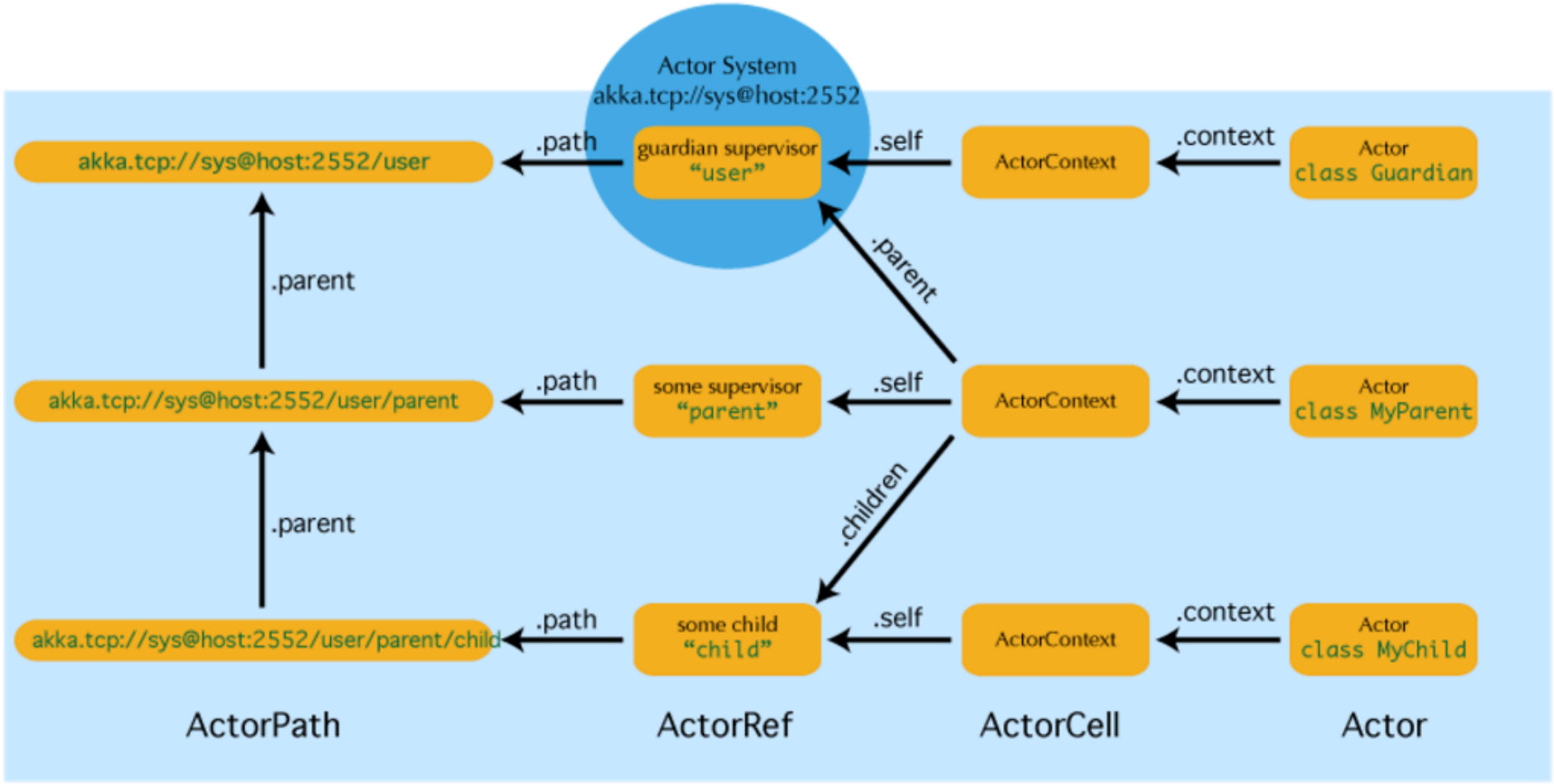
Hierarchical structure

- Parent creates and supervises children
  - Parent(supervisor) can resume, restart, stop them when they encountered failures
  - Restart: create new actor instance and replace old one
- Error kernel pattern(let it crash)
  - Dispatch (dangerous) sub-task to children
  - Handle their failures
- Like a corporation in human society

Actor System

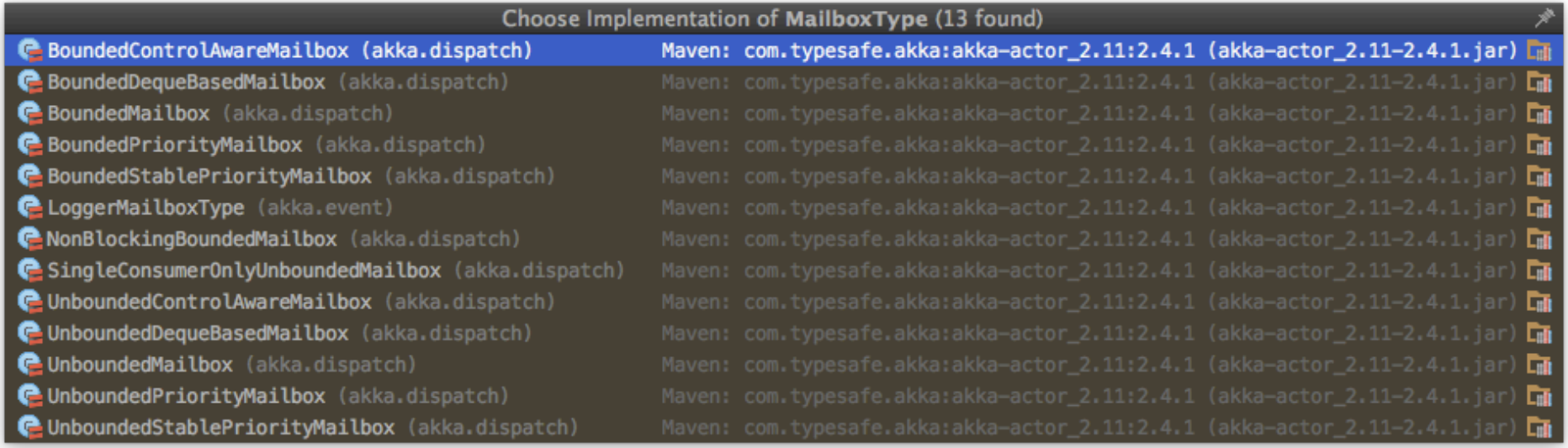


Location transparency



Mailbox

- each actor has a mailbox, which is a message queue, stores received messages
- Different implements
  - Default is UnboundedMailbox

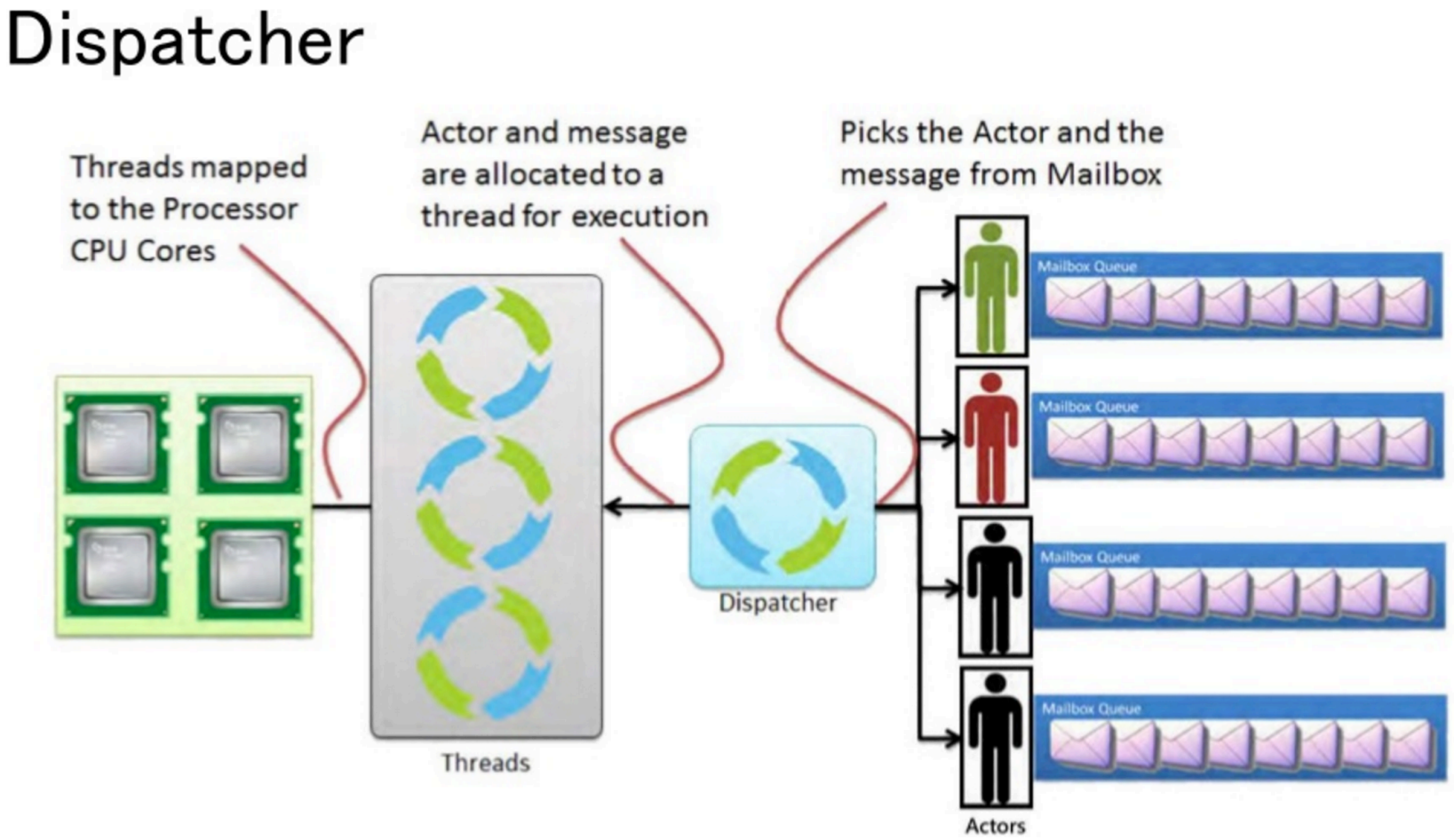


Dispatchers

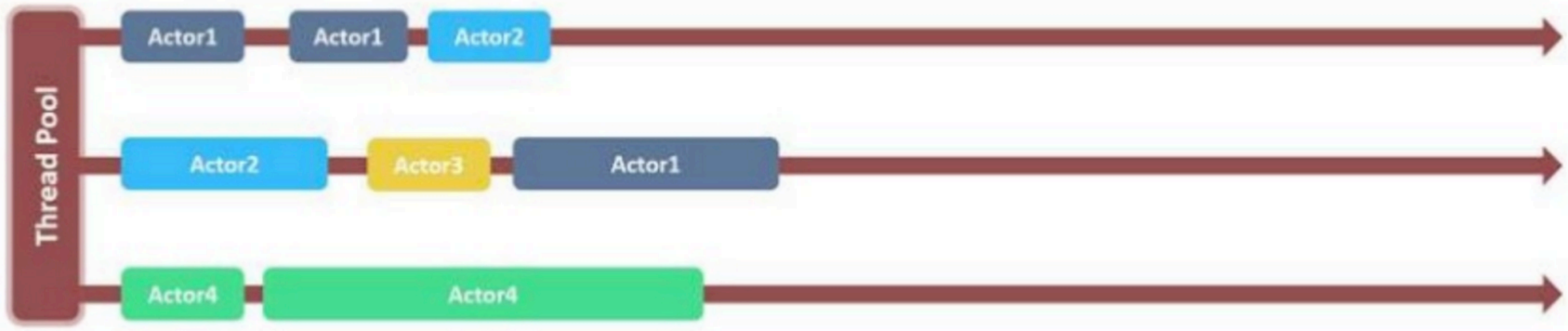
- Dispatchers
  - Dispatcher:
    - a set of actors bind to a thread pool
  - PinnedDispatcher
    - A actor bind to a thread
  - CallingThreadDispatcher
    - Actor run on current threa

```
blocking-io-dispatcher {
  type = Dispatcher
  executor = "thread-pool-executor"
  thread-pool-executor {
    fixed-pool-size = 32
  }
  throughput = 1
}
```

Dispatcher


Dispatcher
Threads mapped to the Processor CPU Cores
Actor and message are allocated to a thread for execution
Picks the Actor and the message from Mailbox
Mailbox Queue
Mailbox Queue
Mailbox Queue
Mailbox Queue
Dispatcher
Threads
Actors

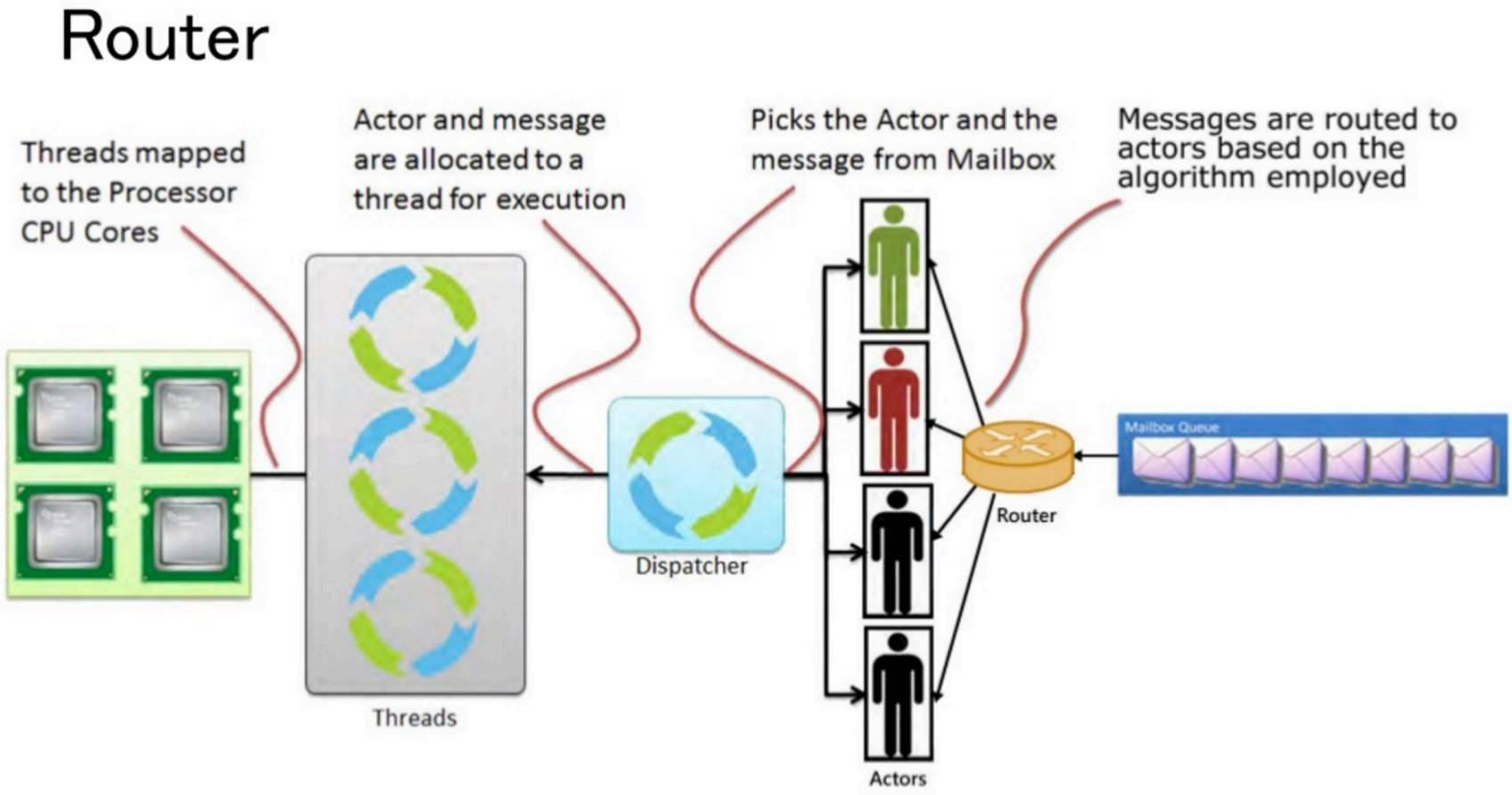
Dispatcher


Thread Pool
Actor1
Actor1
Actor2
Actor2
Actor3
Actor1
Actor4
Actor4

Routing

- Routing
  - Two kinds of routers:
    - Pool : router creates routees as children actors and remove them if they terminate
    - Group: routees are created outside router
  - Route logic: round robin, load balance, random, consistent hash, broadcast…

Routing


Router
Threads mapped to the Processor CPU Cores
Actor and message are allocated to a thread for execution
Picks the Actor and the message from Mailbox
Messages are routed to actors based on the algorithm employed
Mailbox Queue
Router
Dispatcher
Threads
Actors

Create and config instances

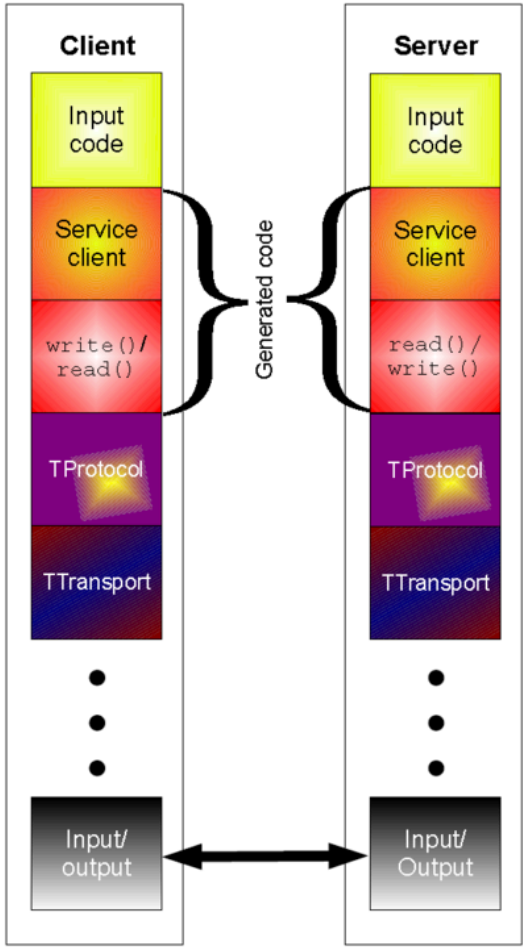
```
Props props = Props.create(ExeReportActor.class)
        .withRouter(new RoundRobinPool(10))
        .withDispatcher("blocking-io-dispatcher")
        .withMailbox("bounded-mailbox");

ActorRef exeReportActor = context.actorOf(props, "exeReportActor");
```

application.conf:

```
akka {
        actor {
                bounded-mailbox {
                  mailbox-type = "akka.dispatch.BoundedMailbox"
                  mailbox-capacity = 1000
                  mailbox-push-timeout-time = 10s
                }
        }
}
```

Thrift

- Cross-language serialization & RPC framework
- Transport layer
  - TSocket: blocking IO
  - TFramedTransport: non-blocking
- Protocol layer
  - TBinaryProtocol
  - TCompactProtocol
  - TJSONProtocol
- Servers
  - TSimpleServer/TThreadPoolServer
  - TNonblockingServer/THsHaServer



Thrift type system

- Base Types
  - Bool, byte, i16, i32, i64, double, string, enum, binary
- Struct
- Container
  - List, set, map
- Exception
- Service
- Thrift File Example

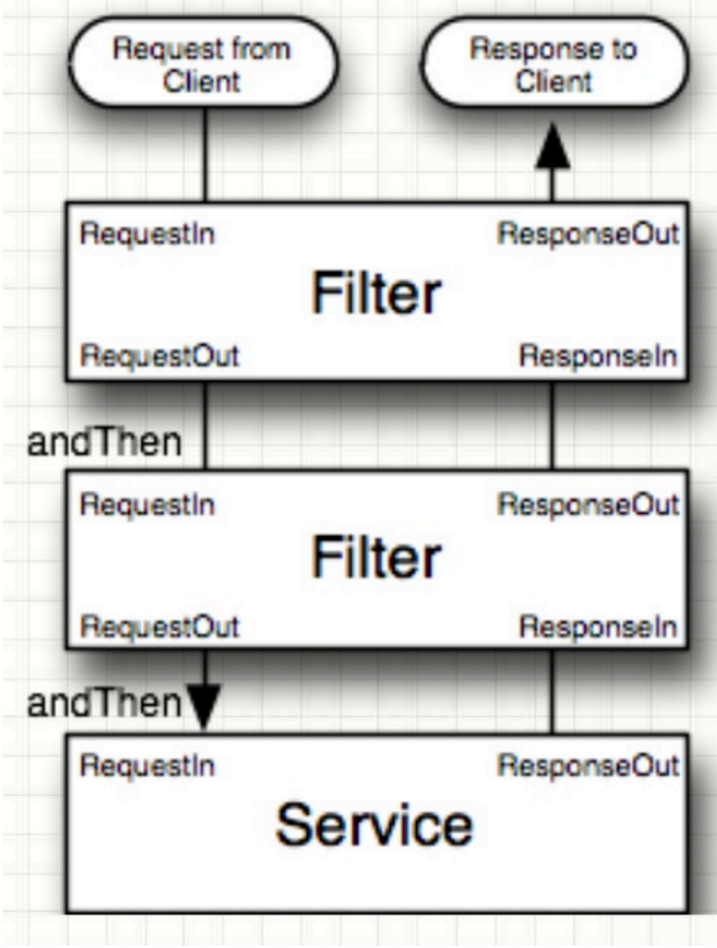
Finagle

Finagle

- Twitter open-sourced fault tolerant, extensible RPC system
- High performance servers & clients
- Support multiple protocols
  - http,
  - thrift
  - mysql
  - redis,
  - memcached…

Finagle

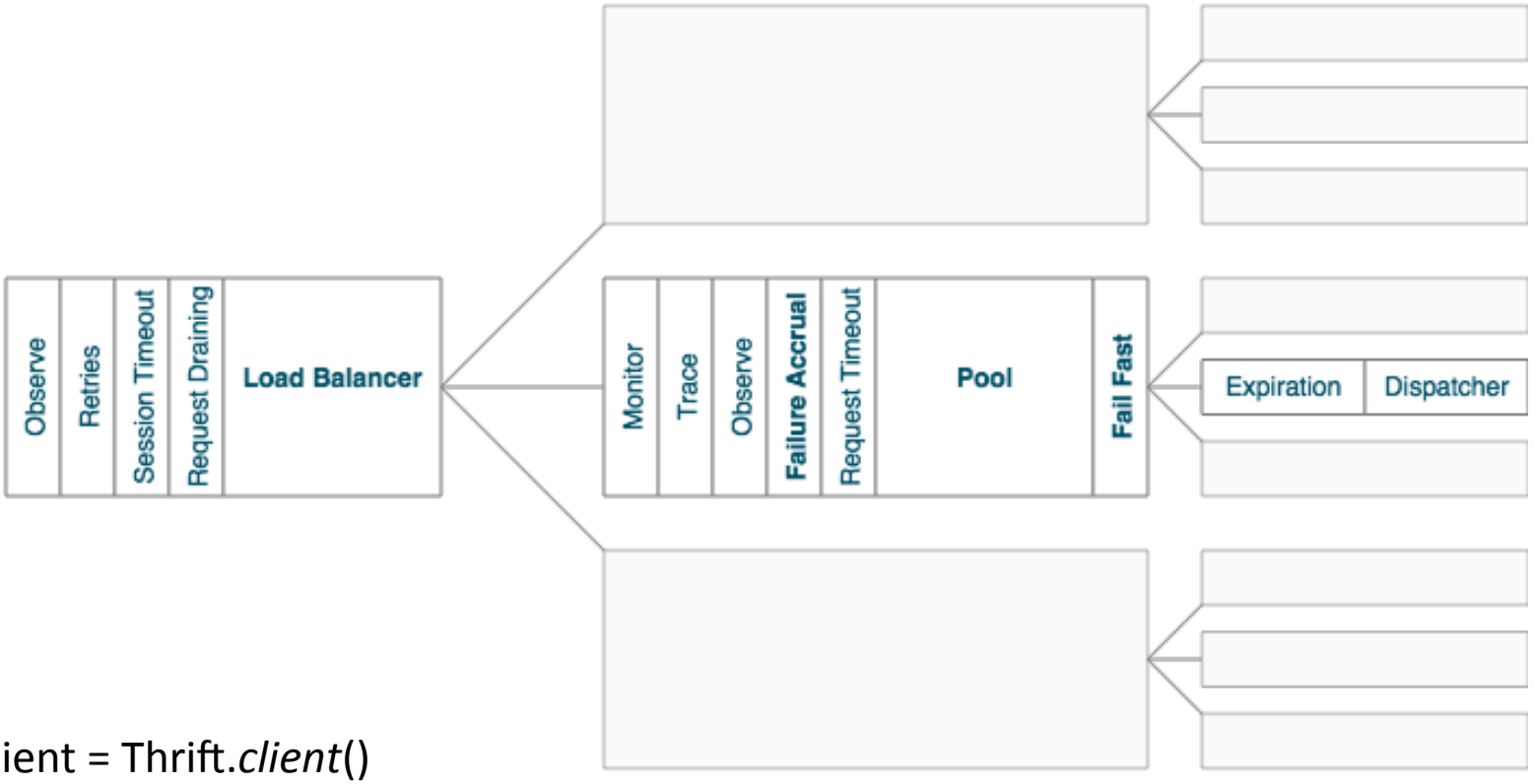
- Modules based implementation
- Future
- Service[Req, Rep]
  - *Req => Future[Rep]*
- Filter
  - *(* MyService *)*
  - *[ReqIn -> (ReqOut -> RepIn) -> RepOut]*
  - *service: String => Int*
  - *filter: ThriftIn => (String => Int) => ThriftOut*
  - *filter andThen service: ThriftIn => ThriftOut*



Finagle Server


```
server = Thrift.server()
        .withLabel("oes_server")
        .withRequestTimeout(Duration.fromSeconds(10))
        .serveIface(new InetSocketAddress(this.port), service);

server.announce("zk!127.0.0.1:2181!/oes_servers!0")
```

Finagle Client



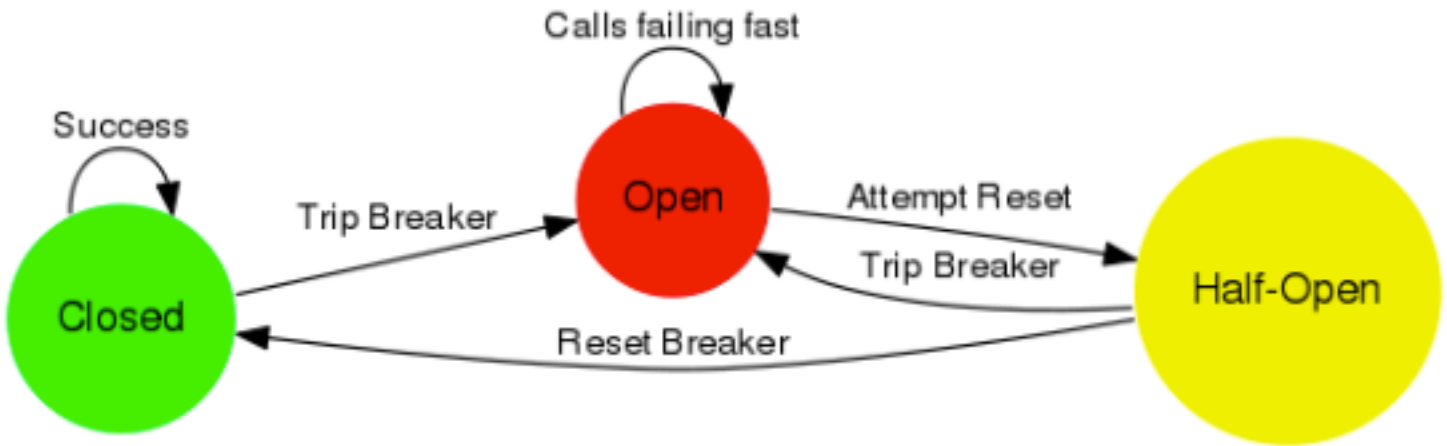

```
client = Thrift.client()
        .withLabel("oes_client")
        .withSessionPool().minSize(10)
        .withSessionQualifier().noFailFast()
        .withRequestTimeout(Duration.fromSeconds(10))
        .newIface(serverAddr, "oes_client", OESClient.FutureIface.class);
```

Finagle Client

- Observe/Monitor/Trace
- Service Discovery/name resolve
  - zk!127.0.0.1:2181!/my/zk/path!0
- Load Balancing
  - P2C + least loaded
- Retry: RetryBudget/RetryBackoff
- Timeout
  - sessionTimeout, requestTimeout
- Circuit breaker: fail fast/failure accrual
- Connection pool

Circuit Breaker

- used to provide stability and prevent cascading failures in distributed systems.
- Example: web service with DB overloaded



Finagle-thrift

- Use scrooge to generate code
- Don’t support multiple services on one port
  - Need implement MultiplexedFinagleService

- Register service through zookeeper

Jesque

Jesque

- Implementation of Resque in Java
- Resque is GitHub open-sourced job queue based on redis
- GitHub use resque to process jobs:
  - warming caches
  - building tarballs
  - firing off web hooks
  - building graphs
  - deleting users
  - updating search index
  - …

Jesque Client

- Normal job

```
Client client = new ClientImpl(new ConfigBuilder().build());
Job job = new Job(className,   Object… args);
String queueName = “test:normal”;
client.enqueue(queueName, job);
```

- Delayed job

```
client.delayedEnqueue(“test:dalayed”, job, now() + 1000);
client.removeDelayedEnqueue(“test:dalayed”, job);
```

- Recurring job

```
client.recurringEnqueue(“test:recurring”, job, now() + 1000, 5000);
```

Jesque Worker

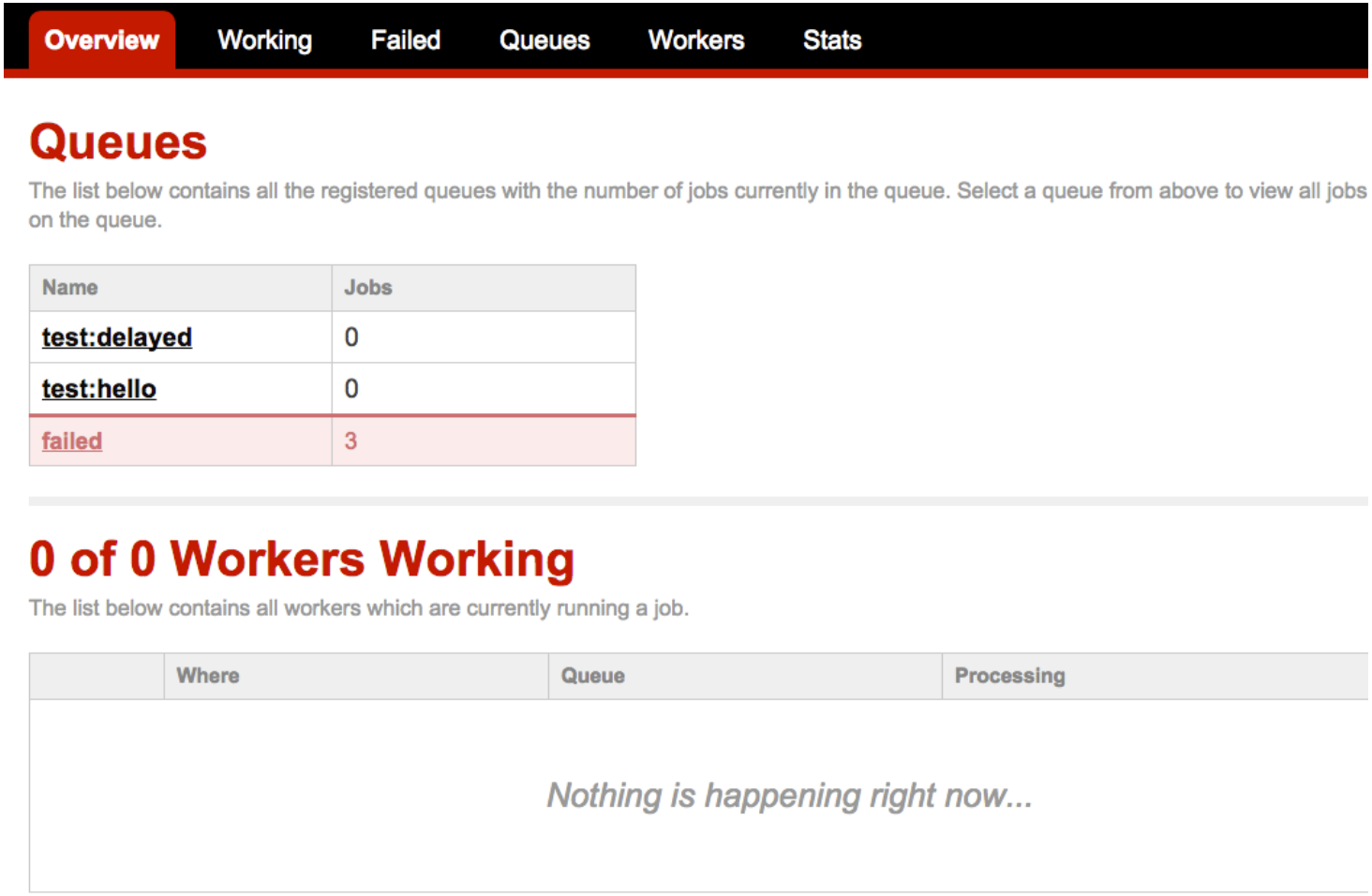
- Job runner implement Runnable/Callable

```
public static class HelloJob implements Runnable {
 private String name;
 public HelloJob(String name) { this.name = name;}
        @Override
        public void run() {
         System.out.println("hello " + );
        }
}
```

- Worker register queues & runners

```
Map<String, Class<?>> jobMap = new HashMap<>();
jobMap.put(jobType, HelloJob.class);
MapBasedJobFactory jobFactory = new MapBasedJobFactory(jobMap);
Worker worker = new WorkerImpl(config, Arrays.asList("test:normal", "test:delayed"), jobFactory);
executorService.submit(worker);
```

Jesque Web


Overview
Working
Failed
Queues
Workers
Stats
Queues
The list below contains all the registered queues with the number of jobs currently in the queue. Select a queue from above to view all jobs on the queue.
Name
Jobs
test:delayed
0
test:hello
0
failed
3
0 of 0 Workers Working
The list below contains all workers which are currently running a job.
Where
Queue
Processing
Nothing is happening right now...

Lombok

- Avoid boilerplate code for value class
- Auto code generator
- Intellij Idea plugin

```
@Data
public class Person {
    private String name;
    private int age;
}
```

Google Guava

- Collections
  - Immutable collections
  - New collection types
    - Multiset, multimap, BiMap…
  - Utility classes
- Cache

```
CacheBuilder.newBuilder().maximumSize(1000)
   .expireAfterWrite(10,   TimeUnit.MINUTES).build(
            new CacheLoader<Key, Value>() {
                  public Value load(Key key) {
                    return dao.get(key);
                  }
            }
      );
```

Google Guava

- Concurrency
- Strings
- Primitives
- Reflection
- …