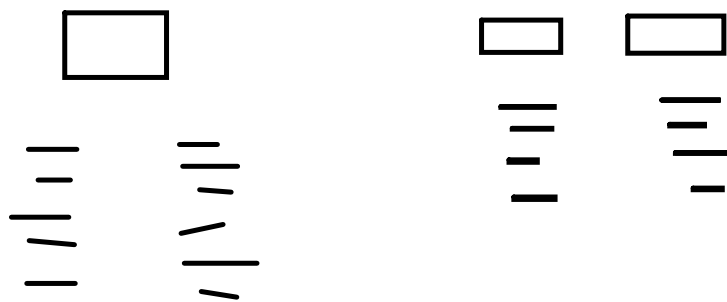Akka :

Play

Concurrent Application :

CPU is getting wider : multi-core

Req : Application to sync with CPU arch.

Shared Mutable State

Concurrency V/s parallelism

Concurreny : Multitasking on single core

Parrellelism : Multiple Threads running on multicore processor

Asynchronous V/s synchronous

sync : caller waits

async : caller may proceed and called method may inform back using callback or future or message

Blocking V/s Non-Blocking

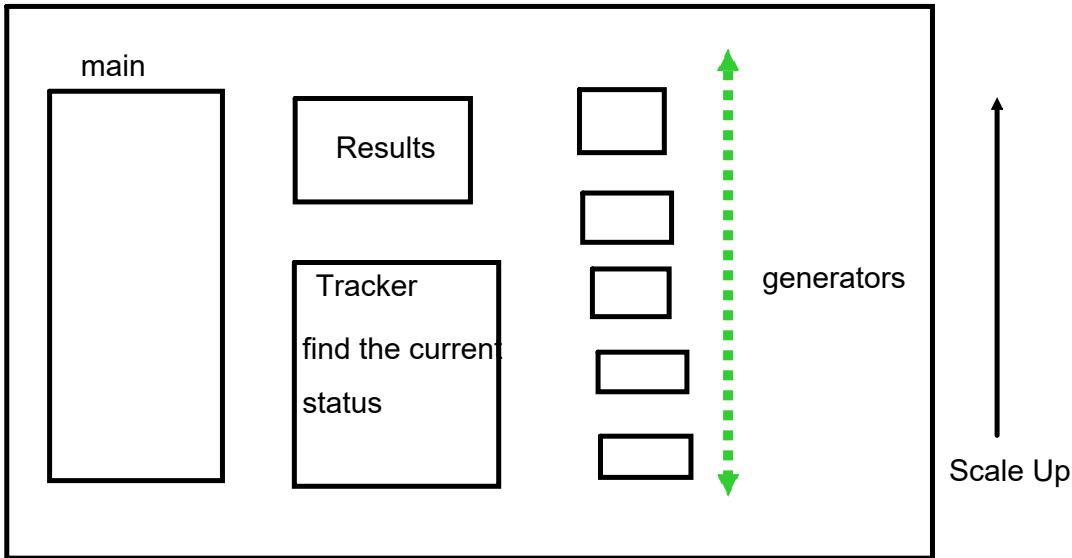one thread delaying other

Race Condition

more than one thread try to change state of shared mutable data

USe - Case : Create a sorted set of 20 probable prime big integers

BigInteger.nextProbablePrime()

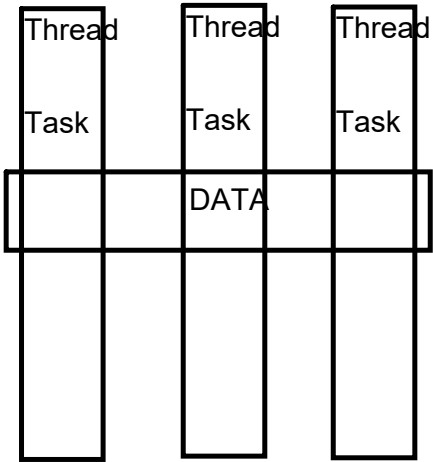Single Threaded response : 24852 ms.

Multi Threaded env : 10957 ms.

main

Results

Tracker

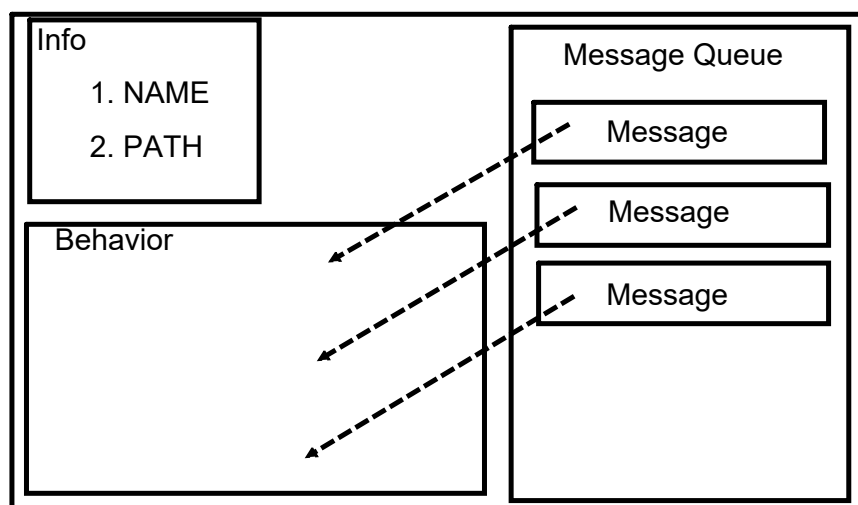find the current

status

generators

Scale Up

Three primary concerns

1. Data Thread Safe

2. Thread blocking

3. Exception handling

Thread	Thread	Thread
Task	Task	Task
	DATA	

Actor System

A

Manager

calculate

a big prime

A

Employee
ActorRef

A

Employee
ActorRef

A

Employee

A

Employee

Message

All entity in system are actor

Interaction among them is through messages

Parent Actor and 0 or more child actor

ACTOR

Info
1. NAME
2. PATH

Behavior

Message Queue

Message

Message

Message

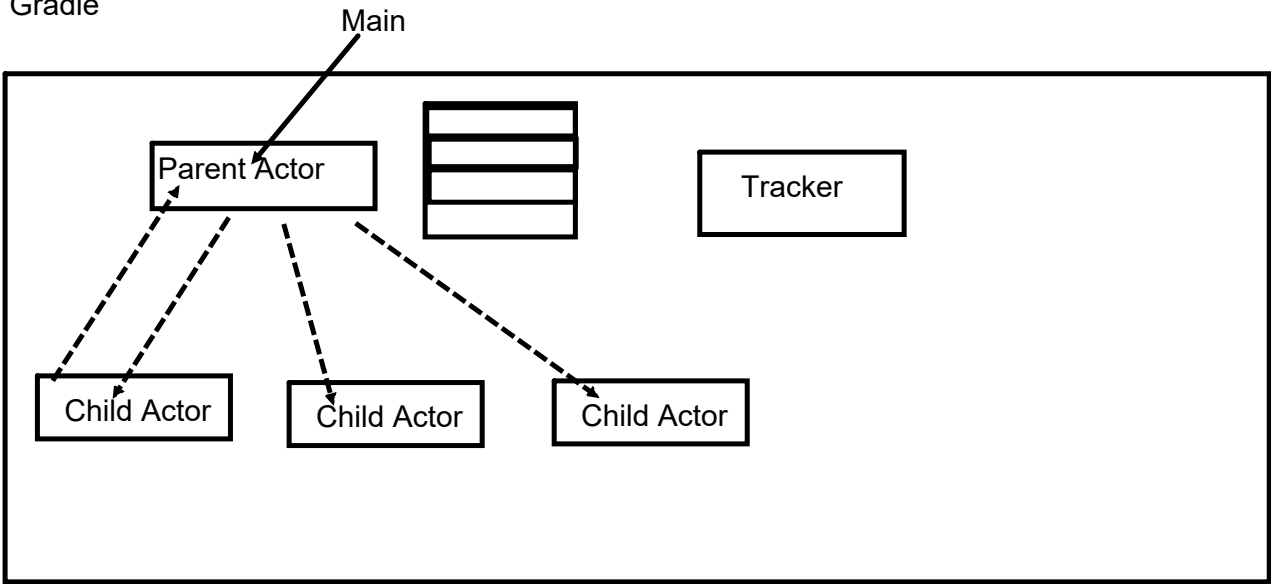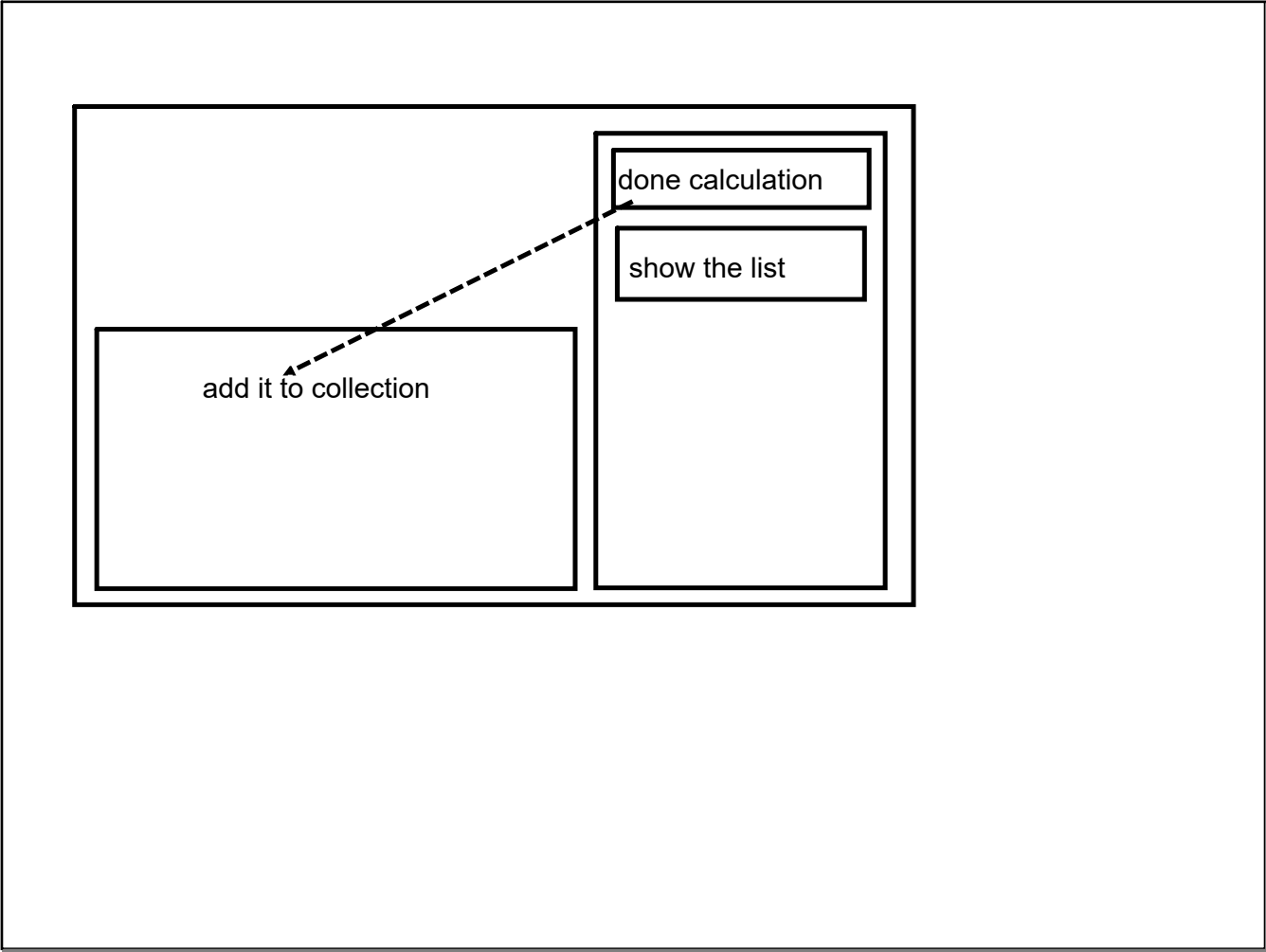=>Each Actor has its own thread
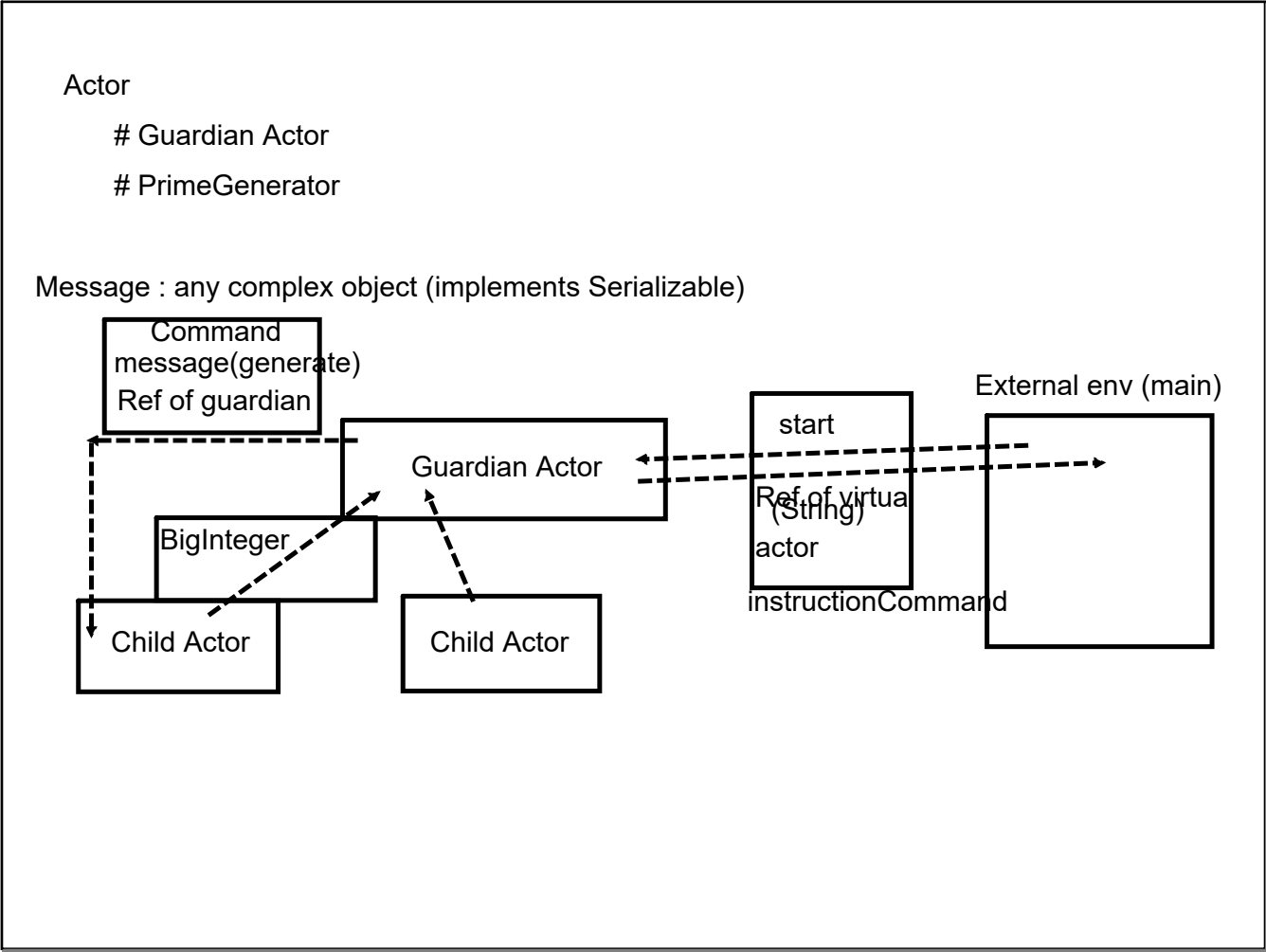
=> Actors won't share data

=> Messages must be immutable

=> Messages are processed one at a time

SBT : scala build tool

Maven

Gradle

Main

Parent Actor

Tracker

Child Actor

Child Actor

Child Actor

done calculation

show the list

add it to collection

Actor

　　　# Guardian Actor

　　　# PrimeGenerator

Message : any complex object (implements Serializable)

Command
message(generate)
Ref of guardian

Guardian Actor

start

Ref of virtual
(String)
actor

instructionCommand

External env (main)

BigInteger

Child Actor

Child Actor

interface(Command)

Message

Use-Case

Simulate a race

External Env   Actor System

initiating

get response back from system
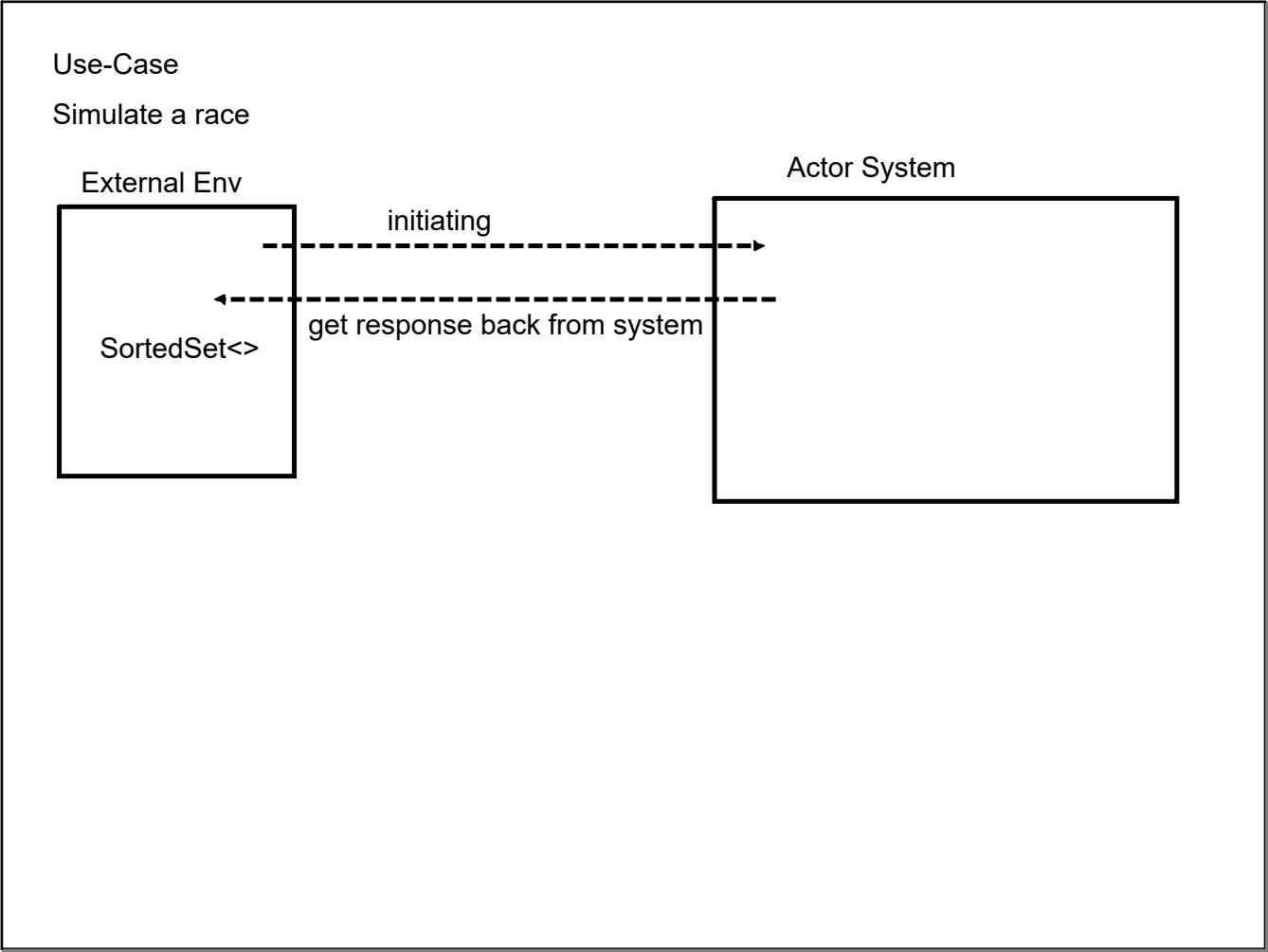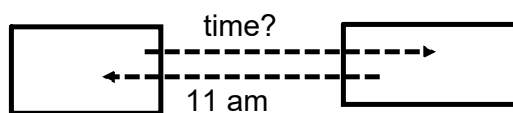
SortedSet<>

Actor interaction patterns

Fire n Forget

   we send a message but does not waits for response

Request-Response (Request-Adapter response : ActorRef)

  we send a message and expects the response

time?

11 am

ask pattern

must have a response

must be time bounded

GuardianActor   Racer

initiate

going to start a timer   start a timer
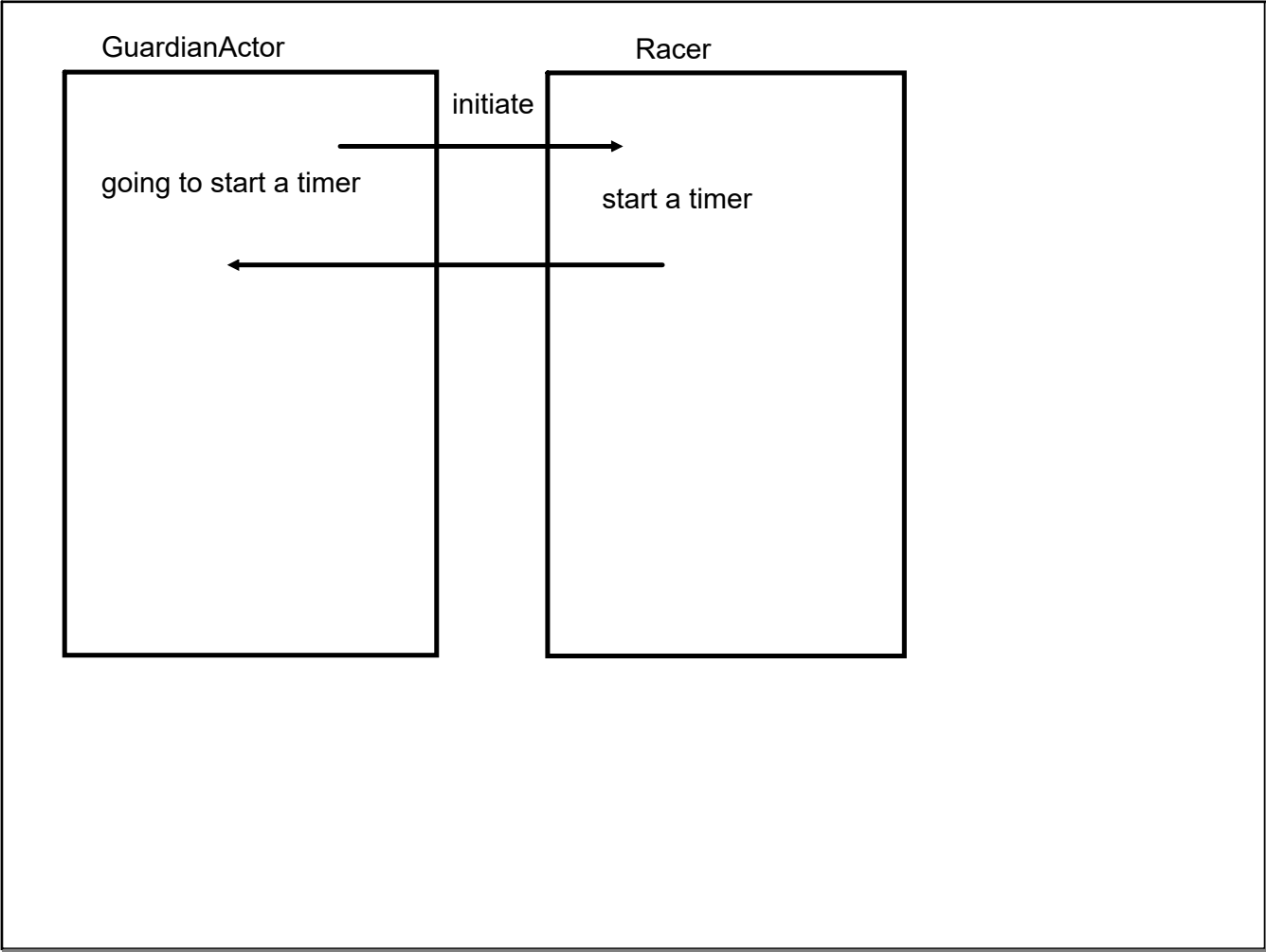
Writing unit test cases...

Test the actual actor

create a virtual guardian actor

popular testing is through logs

Akka Streams

Consumer

Producer

SOURCE	FLOW	FLOW	FLOW	SINK

Processing   Processing

Processing

GRAPH

==>Consumer will not be blocked while receiving data

==> Consumer may continue doing its own activity and data

will be received async in a seperate thread

asynchronous boundaries

Source
<OUT>

async, non-blocking

10 msg/s

<IN,OUT>
Flow

async, non-blocking

SINK
<IN>

async, non-blocking

1 msg/s

OutOfMemory

Buffer : info can be lost

BackPressures

Consumer defines flow of data

asynchronous boundaries

Source

async, non-blocking

10 msg/s

Flow

2 msg

async, non-blocking

SINK

1 msg/s

2 more msg

Back-Pressure

Source -> large number of integer

Flow -> transform int to string

Sink -> consume(print) those string

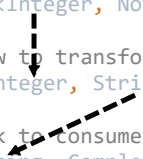 Akka Stream are lazy :

Source -> 100 records of employees

Flow -> will update employee records

Sink -> consume(print) those record

```java
    // source will give out large number of integers
     Source<Integer, NotUsed> source = Source.range(0, 20000000);

     // flow to transform integer into string
     Flow<Integer, String, NotUsed> flow = Flow.fromFunction(val-> val.toString());

     // sink to consume string  (display them)
   Sink<String, CompletionStage<Done>> sink = Sink.foreach(System.out::println);
```
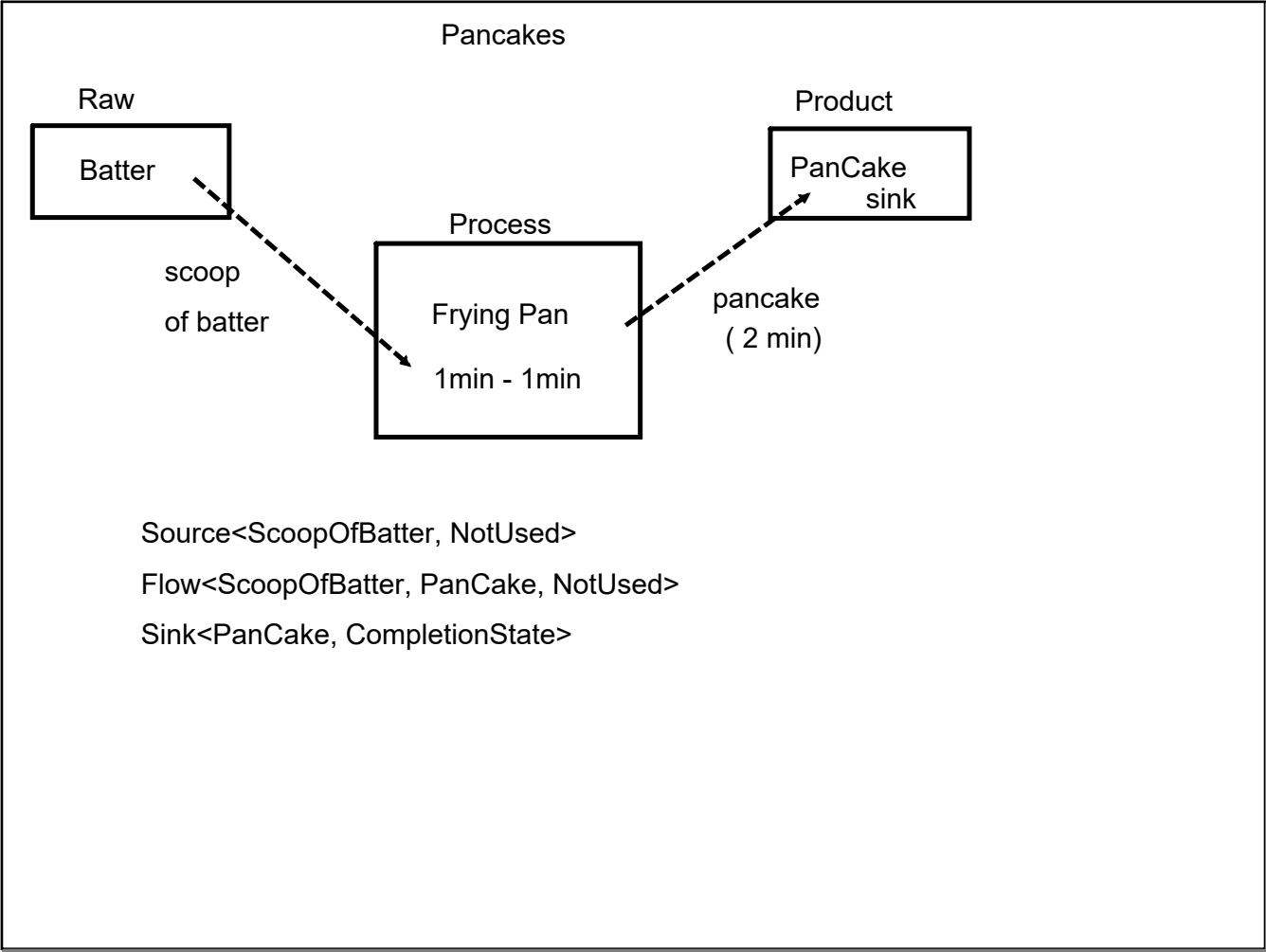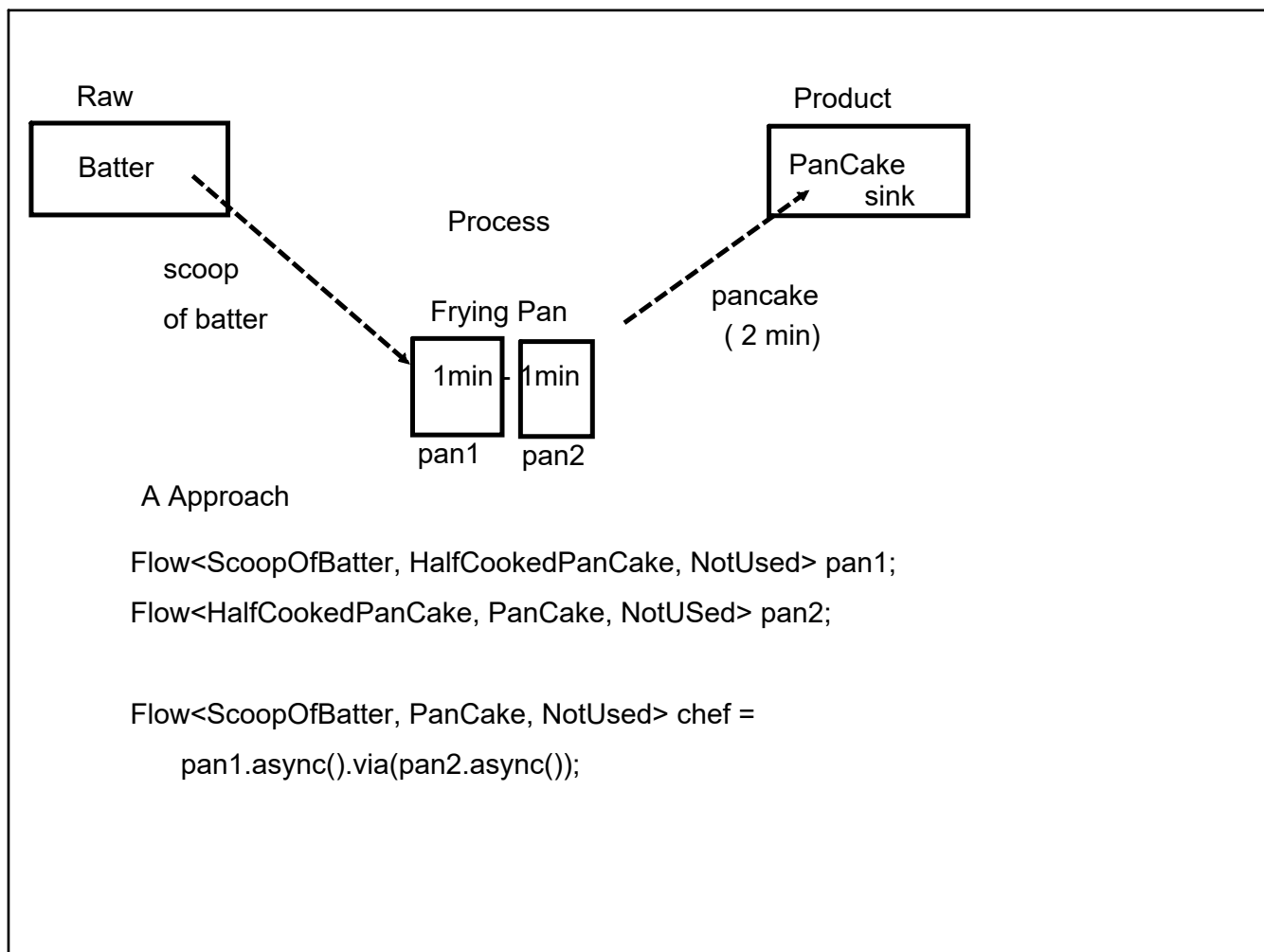
Stages can be expressed with fluent API

Pancakes

Raw

Batter

Product

PanCake
sink

Process

Frying Pan

1min - 1min

scoop
of batter

pancake
(2 min)

Source<ScoopOfBatter, NotUsed>

Flow<ScoopOfBatter, PanCake, NotUsed>

Sink<PanCake, CompletionState>

Raw

Batter

Product

PanCake
sink

Process

scoop
of batter

Frying Pan

1min — 1min

pan1   pan2

pancake
(2 min)

A Approach

Flow<ScoopOfBatter, HalfCookedPanCake, NotUsed> pan1;

Flow<HalfCookedPanCake, PanCake, NotUSed> pan2;

Flow<ScoopOfBatter, PanCake, NotUsed> chef =
    pan1.async().via(pan2.async());
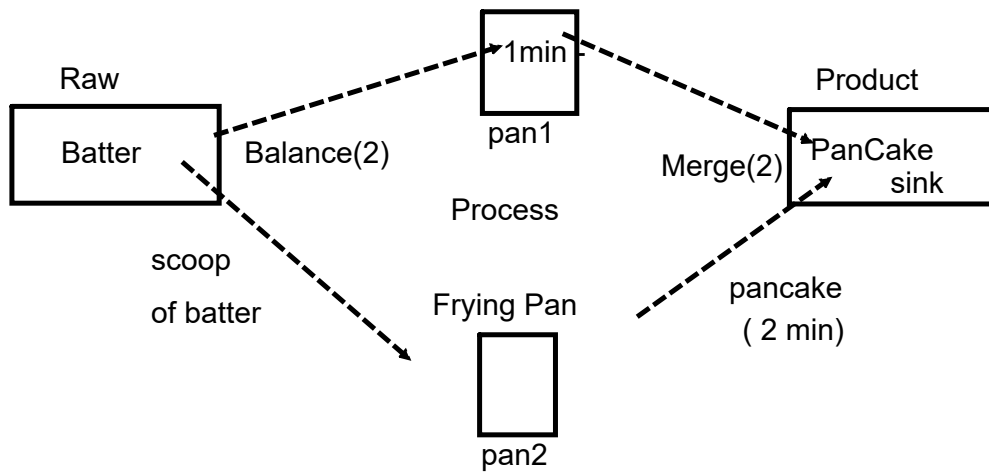
Approach B

```
Flow<ScoopOfBatter, Pancake, NotUsed> pan;

Flow<ScoopOfBatter, Pancake, NotUSed> chef =
    Flow.fromGraph(GraphDSL.create(builder ->{
        // create balancer
        UniformFanOutShape<ScoopOfBatter, ScoopOfBatter> dispatchBatter
            = builder.add(Balance.create(2)); // number of flow to feed
        UniformFanInShape<PanCake, PAncake> mergePancake
            = builder.add(Merge.create(2)); // number of flows from which to get
contents
        // connect actual working
        // first pan
builder.from(dispatchBatter.out(0)).via(builder.add(pan).toInlet(mergePancake.in(0));
        // second pan
builder.from(dispatchBatter.out(1)).via(builder.add(pan).toInlet(mergePancake.in(1));
    }
```

Play Framework :

    highly productive

        1. development :

        2. non-blocking RESTful by default

Non-blocking (Reactive behavior) : server implementation : netty

"full stack framework" : documented pattern of how to use libraries

clean implementation

Tools : build tool : Maven,gradle

SBT : Scala build tool

```
guice : Core context factory implementation : Dependency injection
guice ~ bean factory
```

Action method of controller returns an HTTP Response wrapping + HTTP Header

    1. raw data

    2. Raw HTML

    3. JSON

    4. Streams

    5. Rendered HTML

return : helper methods to represent http status

    eg : ok(), badrequest(), notfound(), created(), found(),

    status(413, <content>);

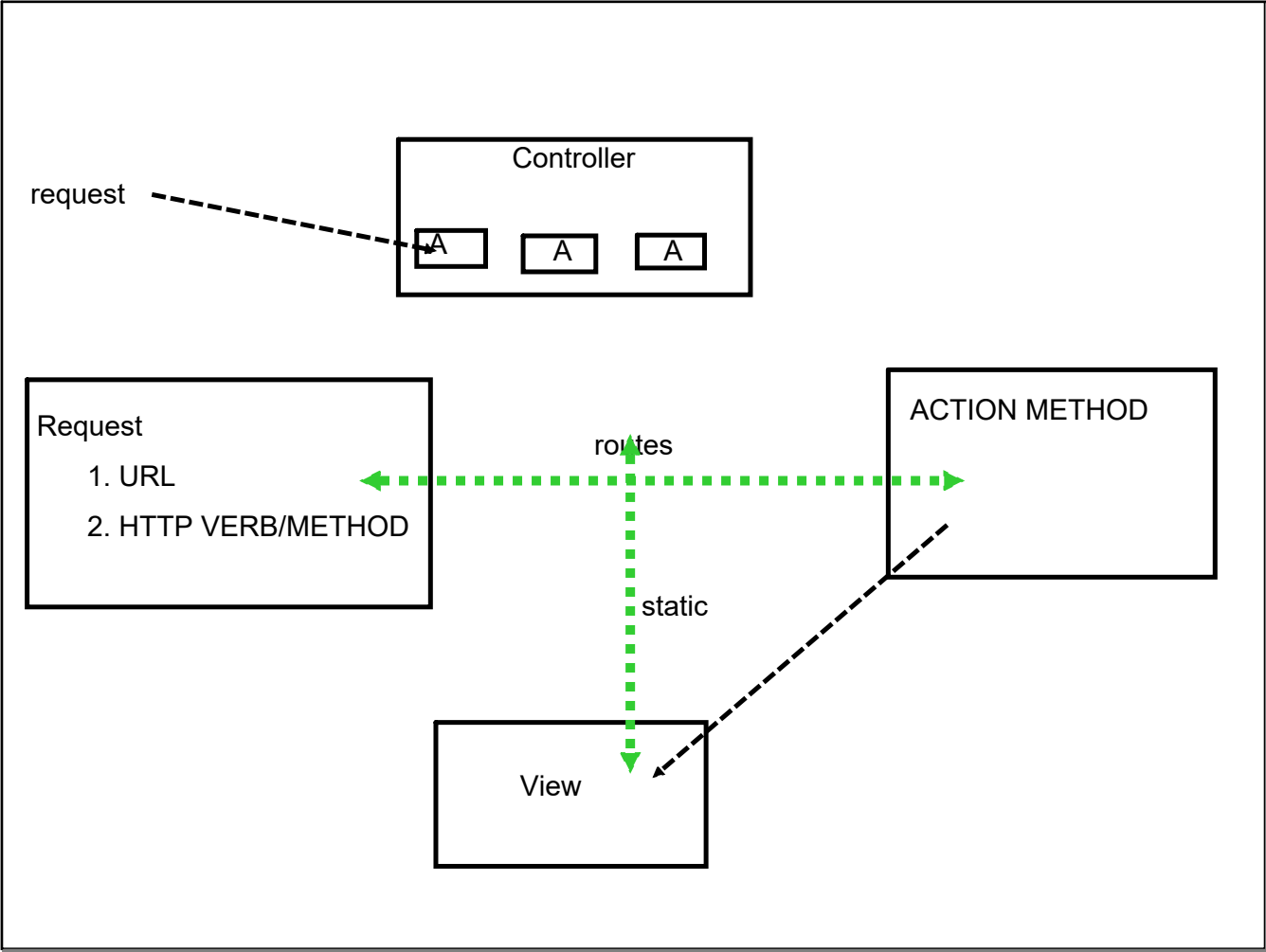Convenience methods : Controller

    help us interact with http env

ctx(); HTTP Context

session()

request();

response();

flash() :

request

Controller

A   A   A

Request
    1. URL
    2. HTTP VERB/METHOD

routes

ACTION METHOD

static

View

View: Server-Side Templates:

    Templating system : Twirl (Scala templates)

    <file>.scala.html

Each view file ~ function

Eg:

    /app/views/index.scala.html   ~ views.html.index()

    /app/views/show.scala.html   ~ views.html.show()

Rendered html : views.html.index.render();

Use-Case : Product

@() // used for receiving argument of view method

// argument would be passed by

     1. Controller Action method

     2. Another view method

Twirl : does not calls getter/setter auto

Jackson (REST) : will going to call auto

Recommended : to use reverse routing of links (scala code)

    Resolve issue of relative url

Twirl provides scala ui helper methods to create ui

Helper methods : allows to map HTML tags with Model object

// helper method will get and set values from/to model object

Action Method   View

Create A Virtual FORM

using MODEL object

HTML Form

(helper method can map

virtual form with HTML form)

Routes

     <VERB> <PATH/URL> <CONTROLLER_CLASS.METHOD>

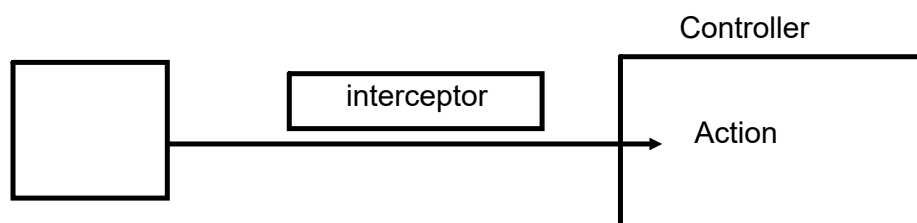 /student/3

 /student?id=3

```
GET     /assets/*file              controllers.Assets.versioned(path="/public", file: Asset)
```

 All static res are compiled and saved in /assets folder for runtime access

Interceptors

Pre-Post processing interceptors

Controller

interceptor

Action

Action.Simple : need to implement call method

Interceptors can be applied to method as well as class level

Multiple interceptor chain themselves

Server Side Validation

ORM framework
    Hibernate : h2Console

Dependency management

    build.sbt

dependency (Play Java)

    1. jdbc

    2. javaJdbc

    3. javaJpa

 dependency (External)

    <groupId> % <ArtifactId> % <version>

    1. h2database

    2. hibernate

    3. mysql

application.conf

#configuration details for DB - Driver/URL etc

#conf related with connection pool

 Configuration file for hibernate (xml)

 : conf/META-INF/persistence.xml

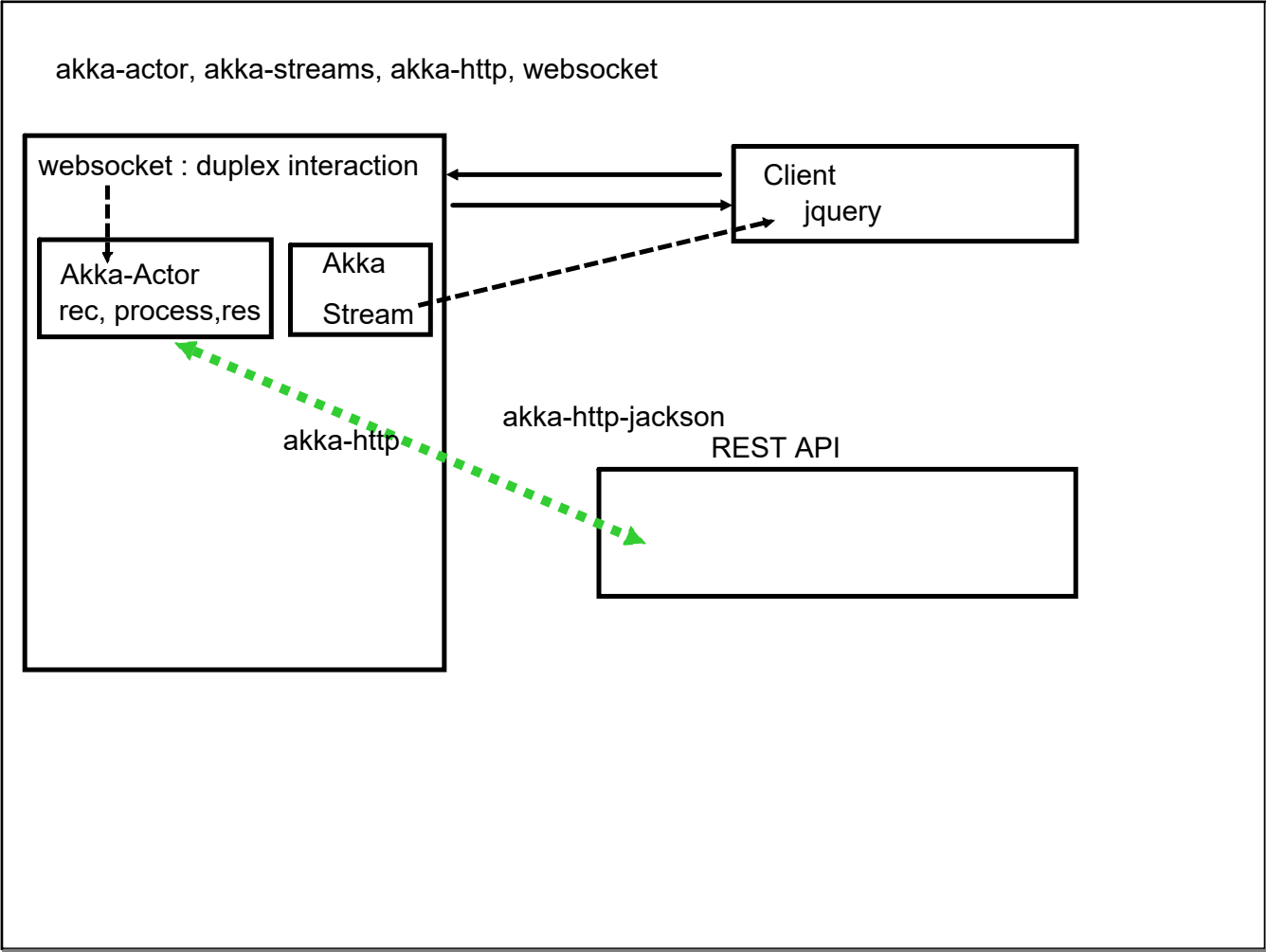 # need to mention this conf file in build.sbt

Play JPA uses Actor System, thus does not expose EntityManager
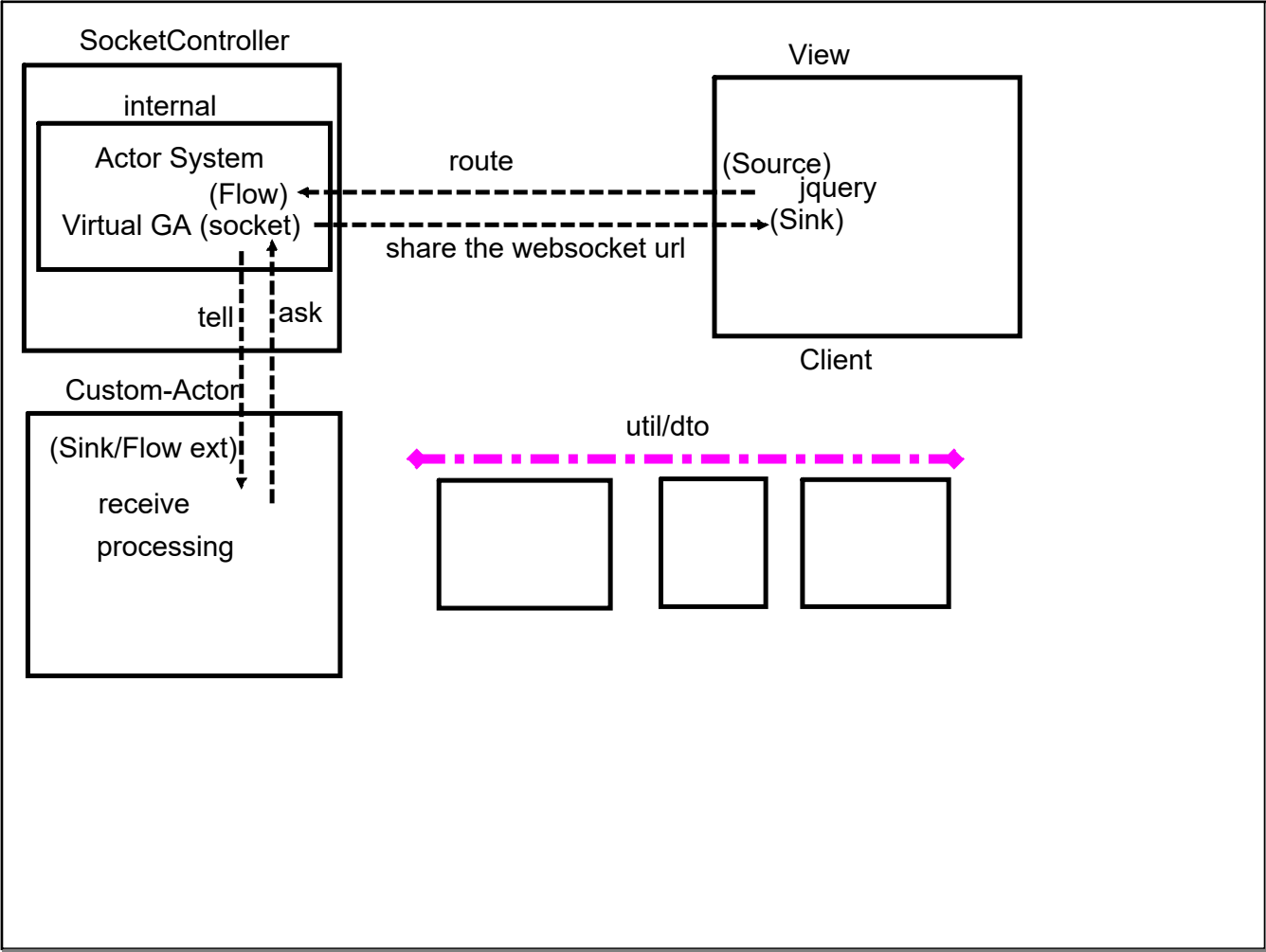
directly

instead exposes JPAApi

REST Controller : Play framework

    async, non-blocking

```
{
    isSuccessfull : true,
    body : {

    }
}
```

Util class will prepare response (ResponseDesigner)

akka-actor, akka-streams, akka-http, websocket

websocket : duplex interaction

Client
    jquery

Akka-Actor
rec, process,res

Akka
Stream

akka-http-jackson
REST API

akka-http

SocketController

internal

Actor System

(Flow)

route

Virtual GA (socket)

share the websocket url

tell   ask

Custom-Actor

(Sink/Flow ext)

receive

processing

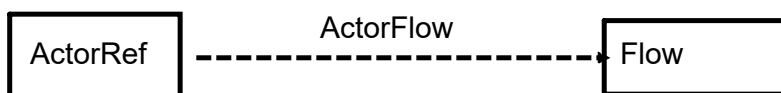View

(Source)

jquery

(Sink)

Client

util/dto

Play's Websocket is build around AkkaStreams

WebSocket : modelled like a FLOW

Websocket will implemented by Actor (ActorRef)

jsonplaceholder.typicode.com/posts/<id>

Add Dependecny

    akka-actor

    akka-stream

    http