

Akka :

Play

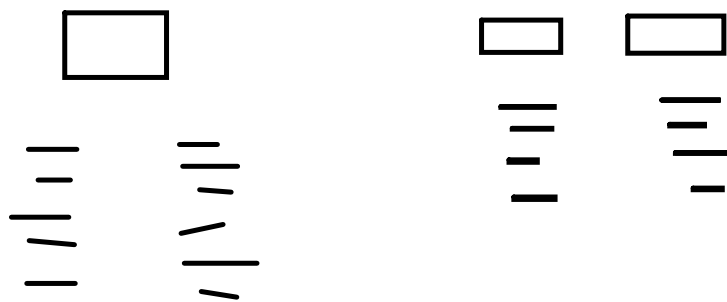
Concurrent Application :

CPU is getting wider : multi-core

Req : Application to sync with CPU arch.

Shared Mutable State

Concurrency V/s parallelism



Concurrency : Multitasking on single core

Parallelism : 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

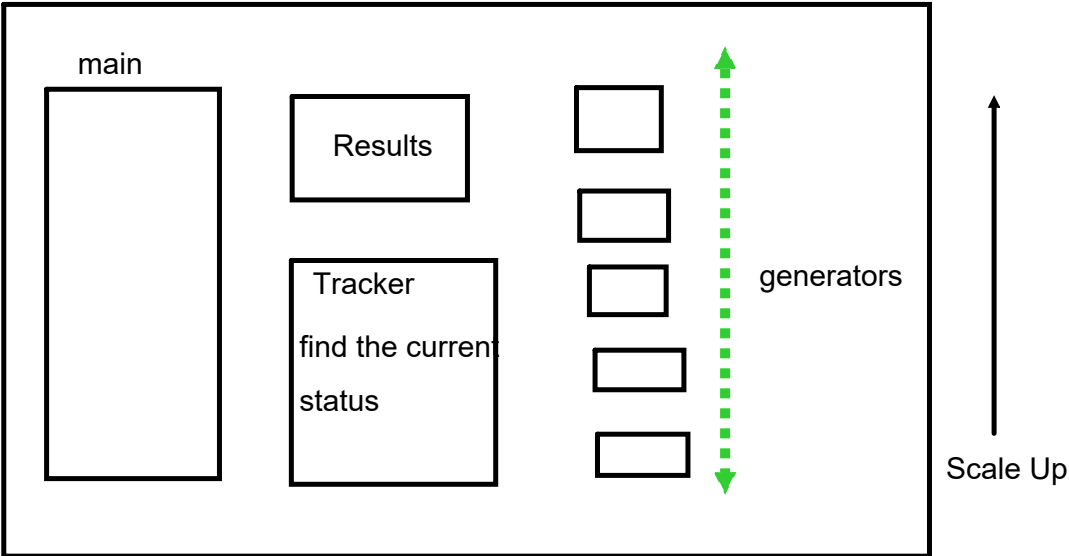
more than one thread try to change state of shared mutable data

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

```
BigInteger.nextProbablePrime()
```

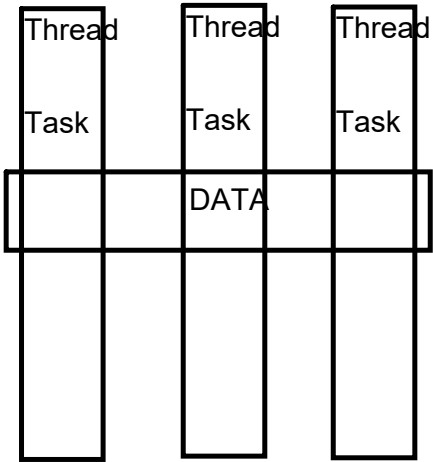
Single Threaded response : 24852 ms.

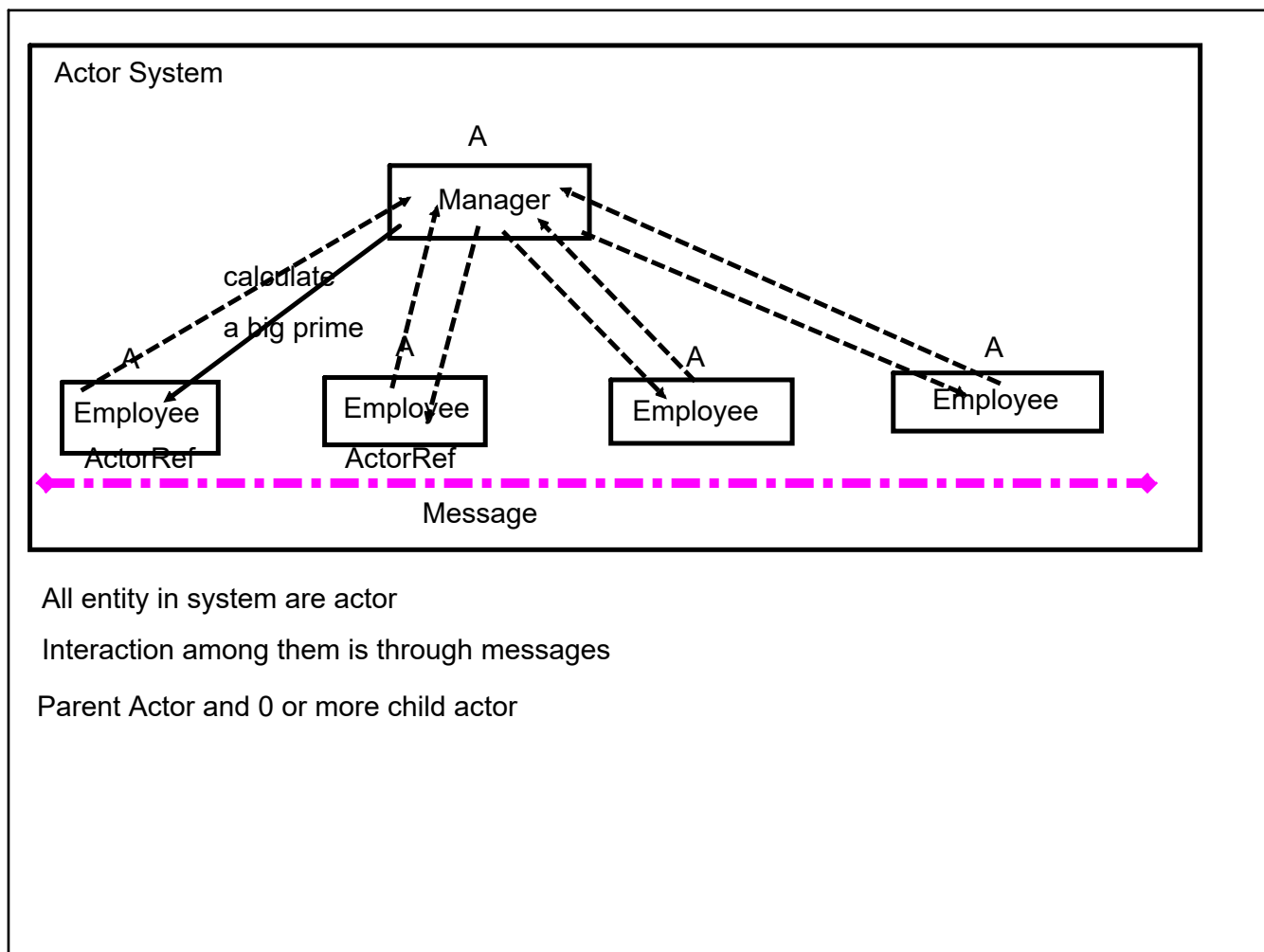
Multi Threaded env : 10957 ms.



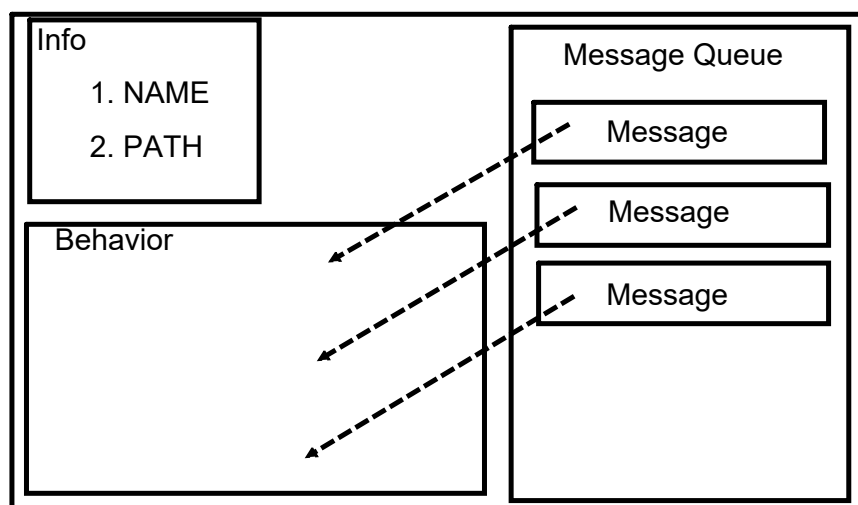
Three primary concerns

1. Data Thread Safe
2. Thread blocking
3. Exception handling





ACTOR



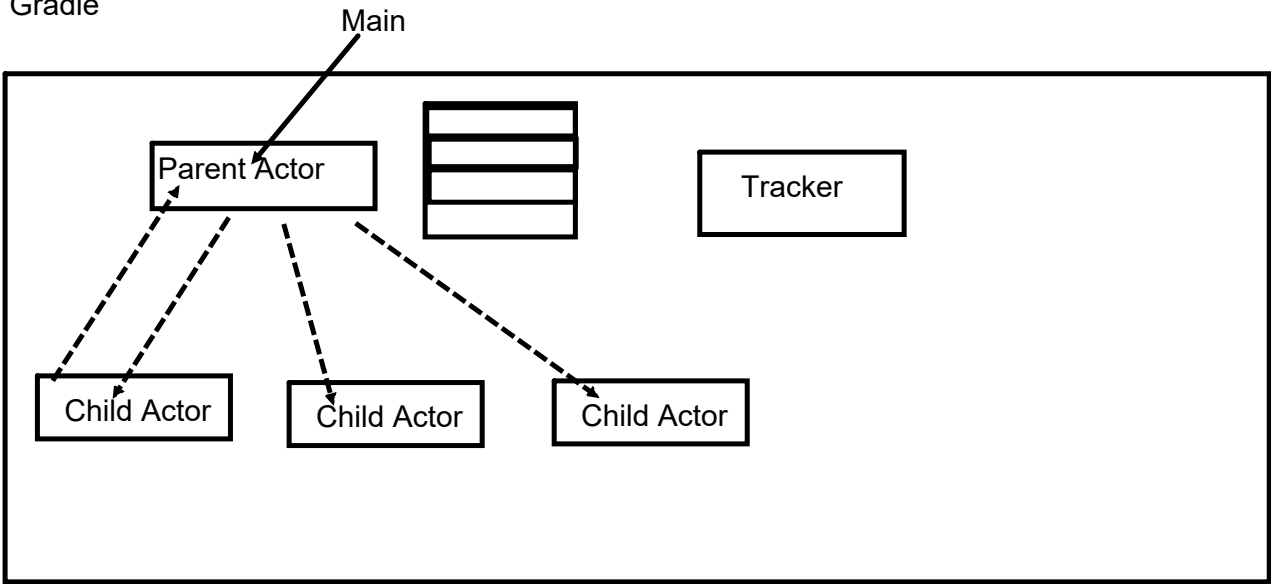
=> Each Actor has its own thread

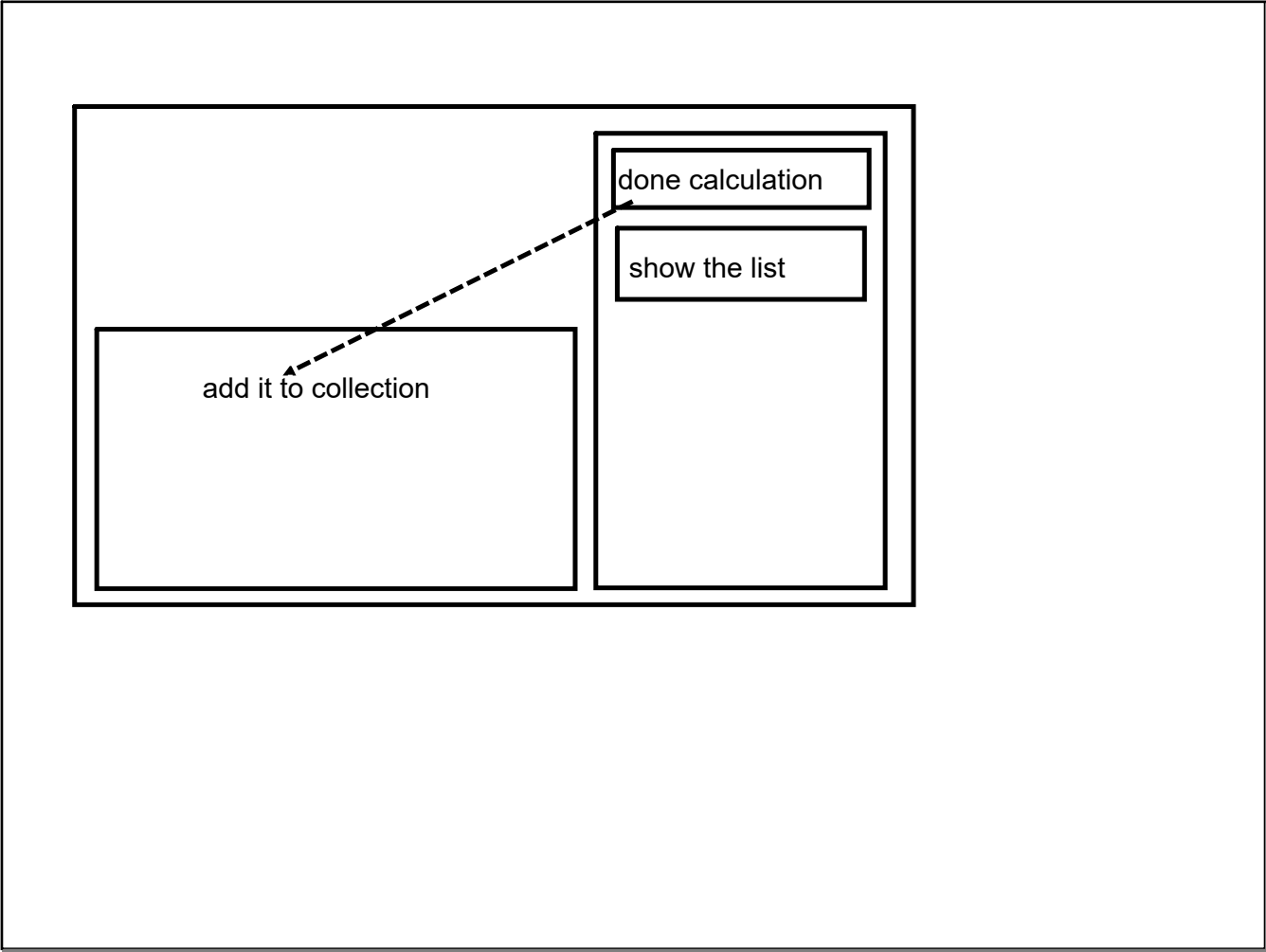
=> Actors won't share data

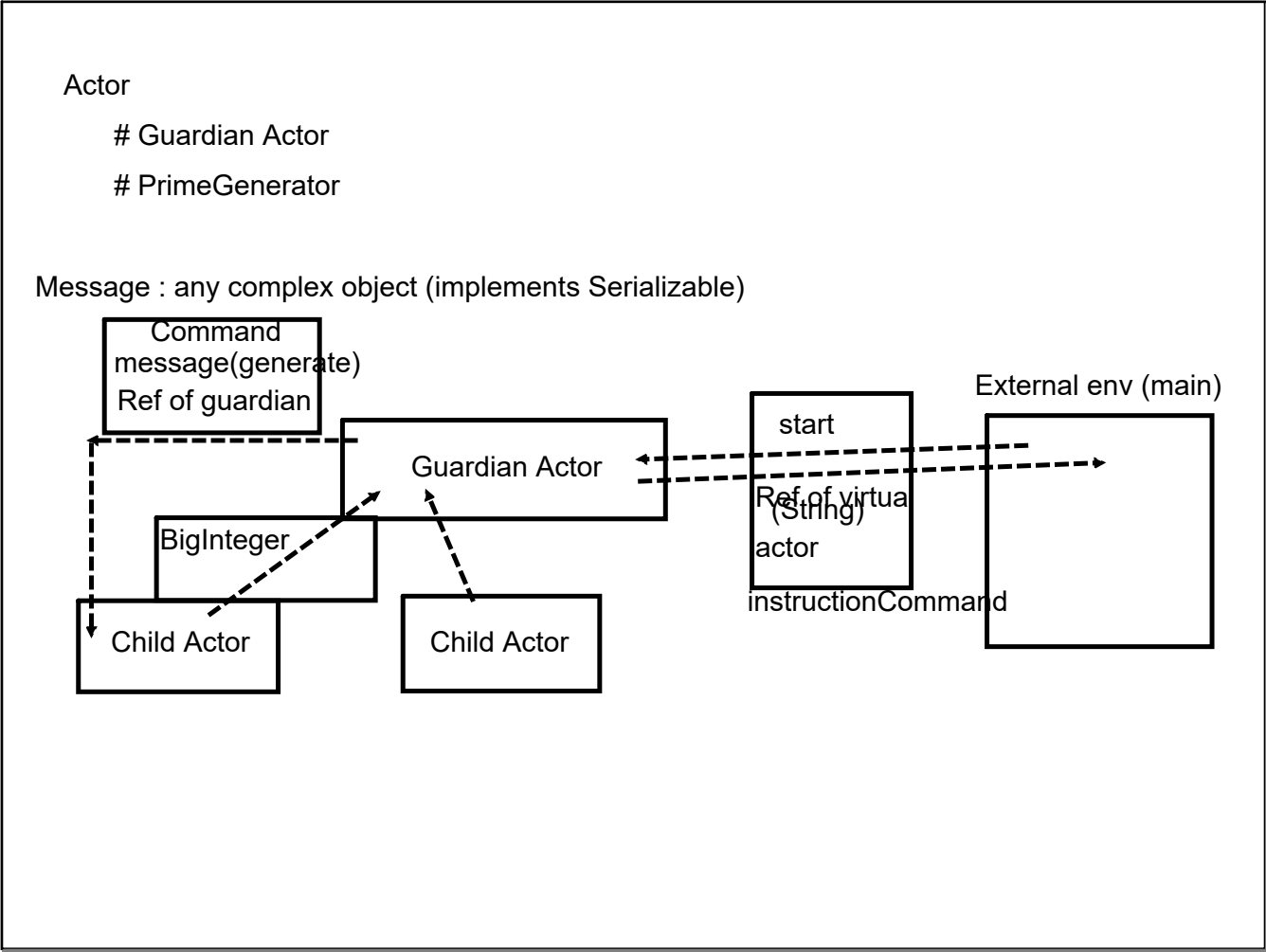
=> Messages must be immutable

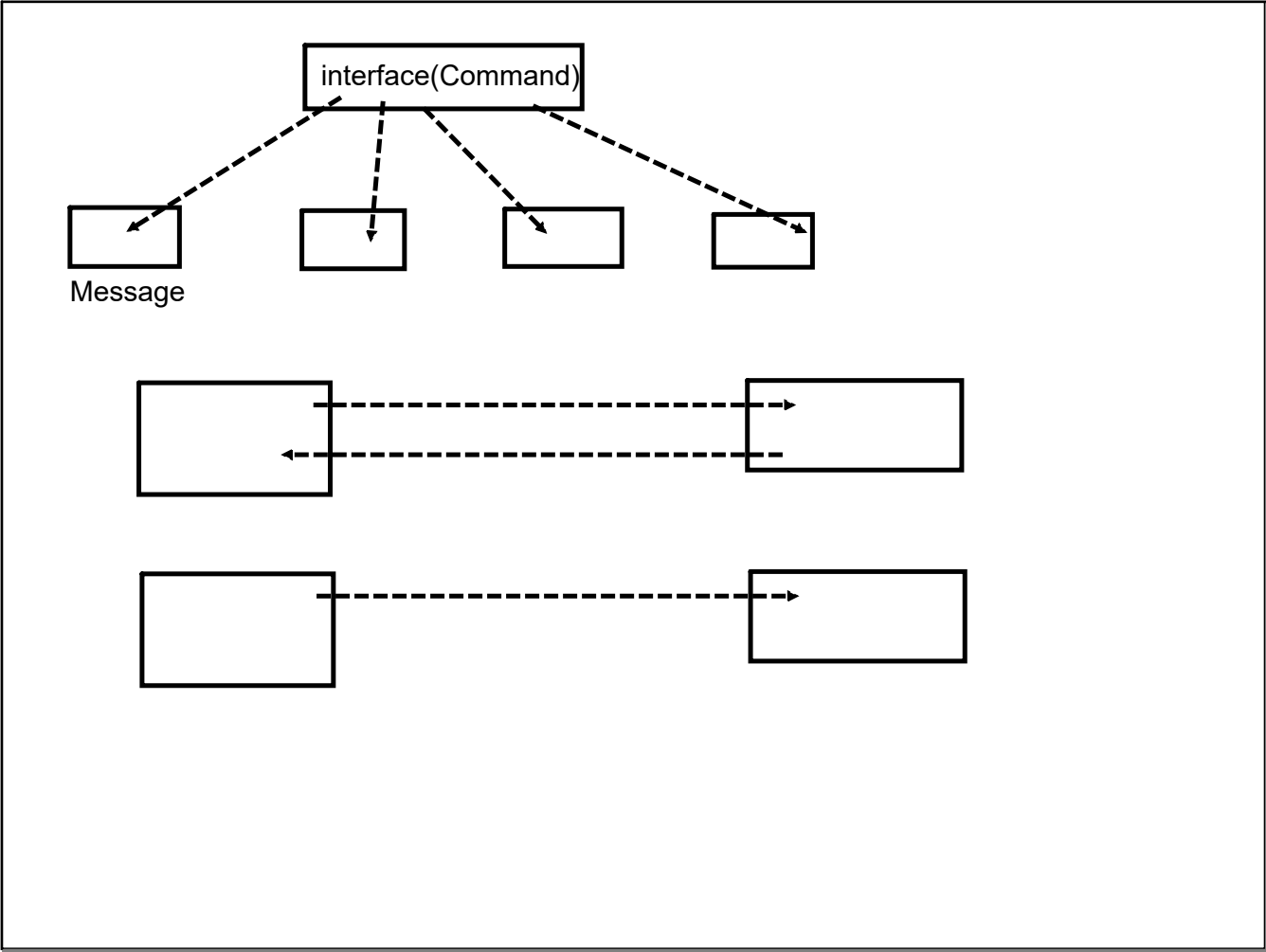
=> Messages are processed one at a time

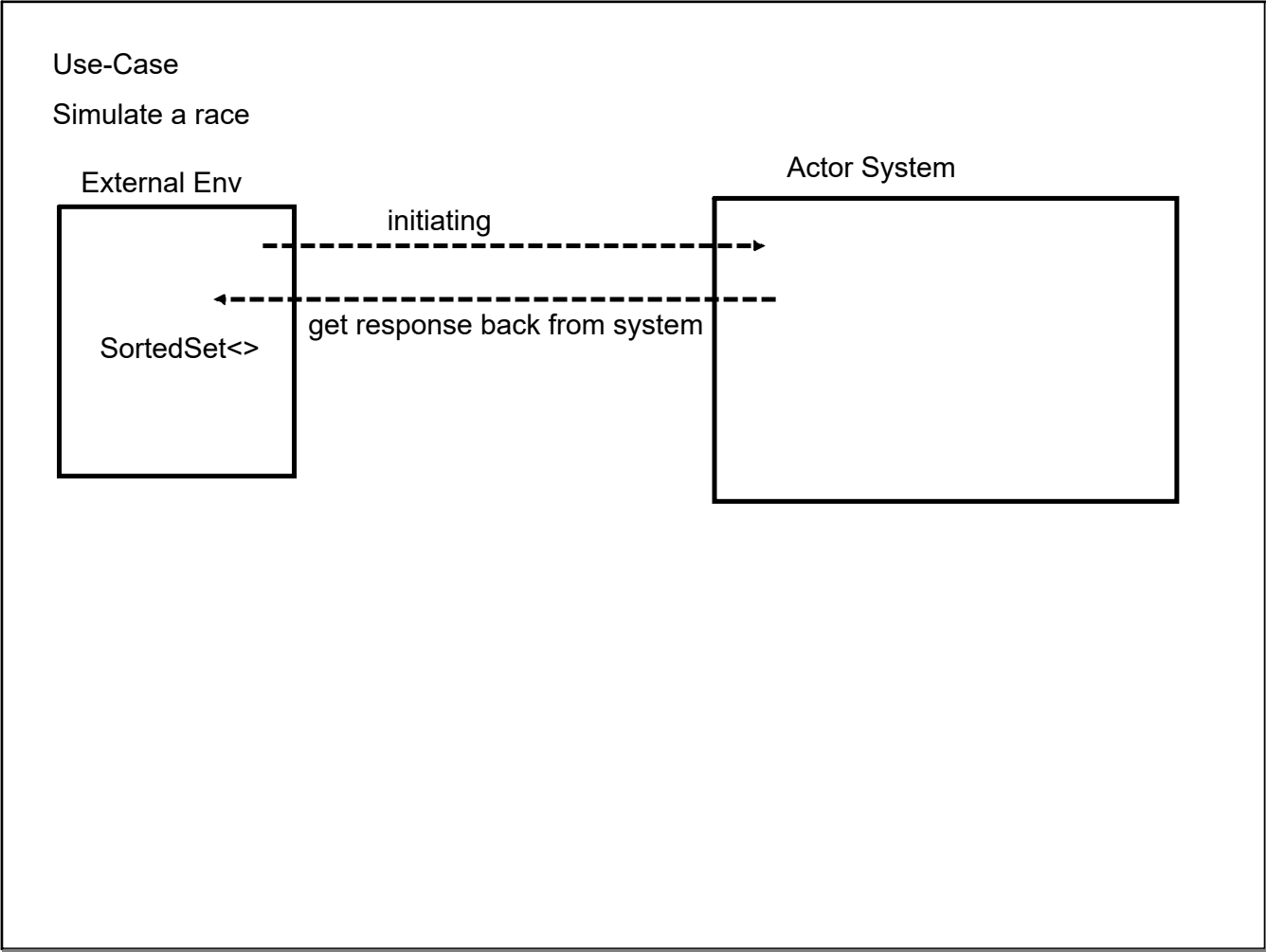
SBT : scala build tool
Maven
Gradle











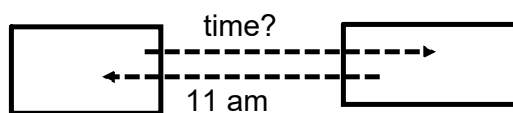
Actor interaction patterns

Fire n Forget

we send a message but does not waits for response

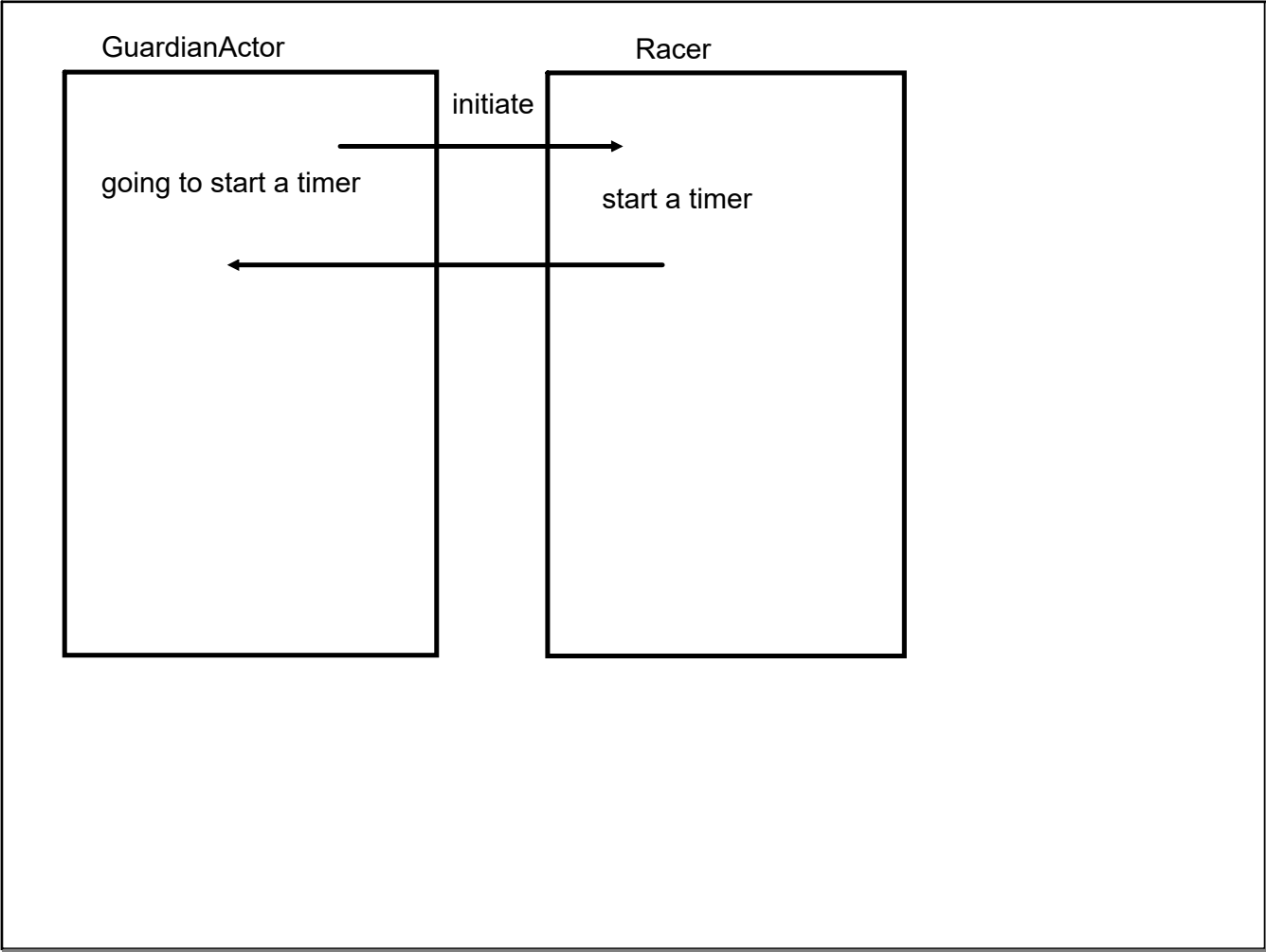
Request-Response (Request-Adapter response : ActorRef)

we send a message and expects the response



must have a response
must be time bounded

ask pattern

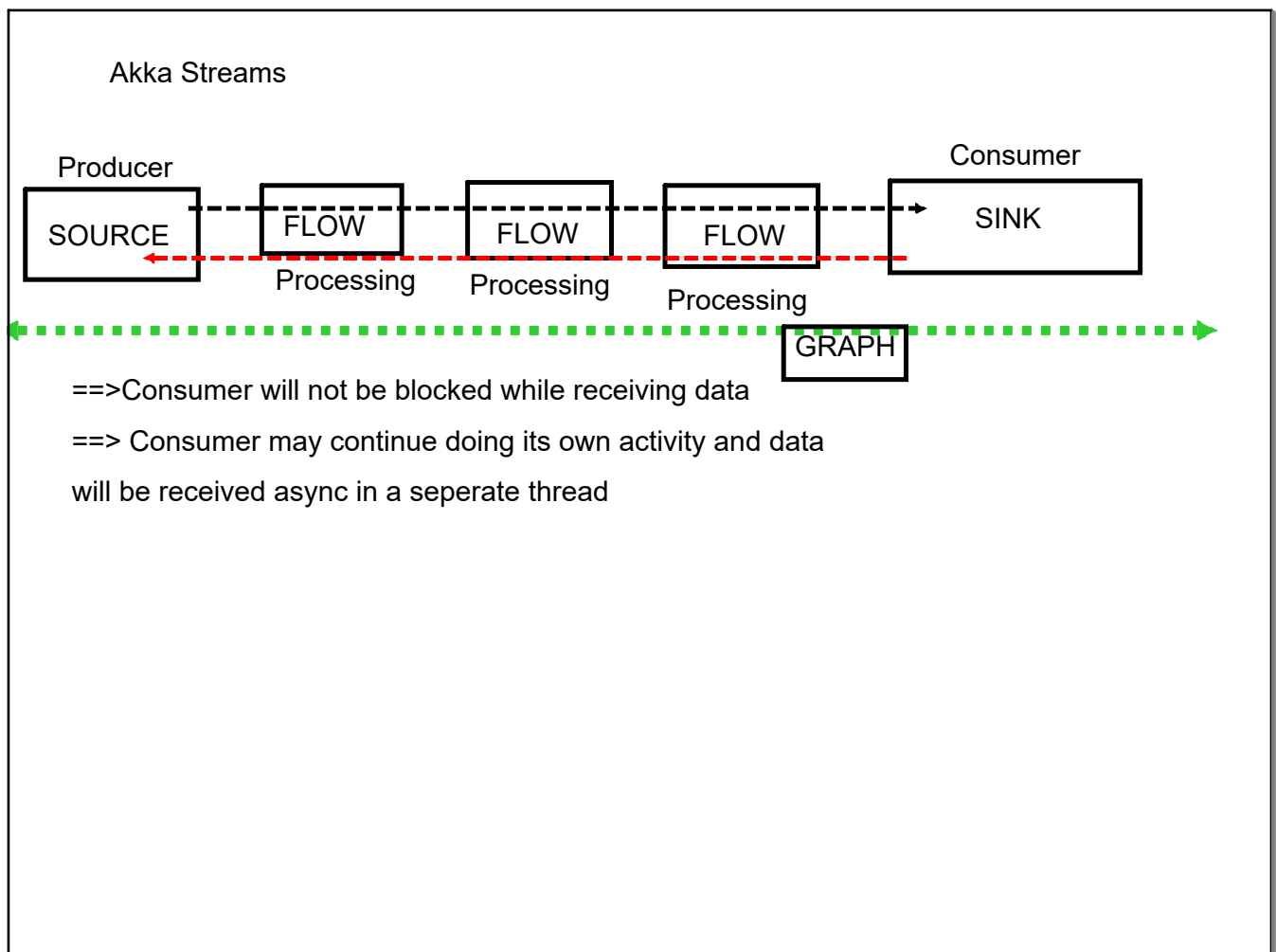


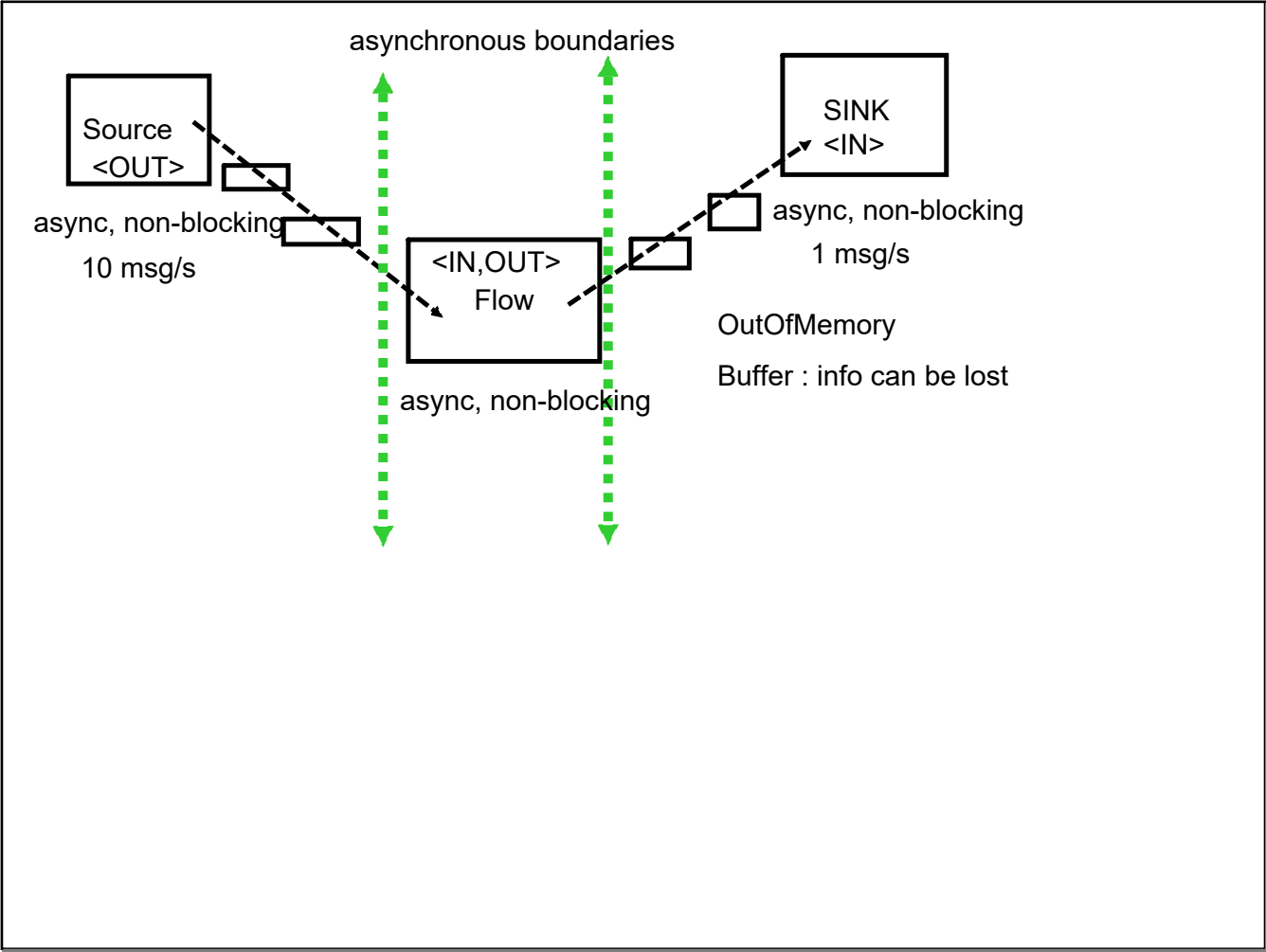
Writing unit test cases...

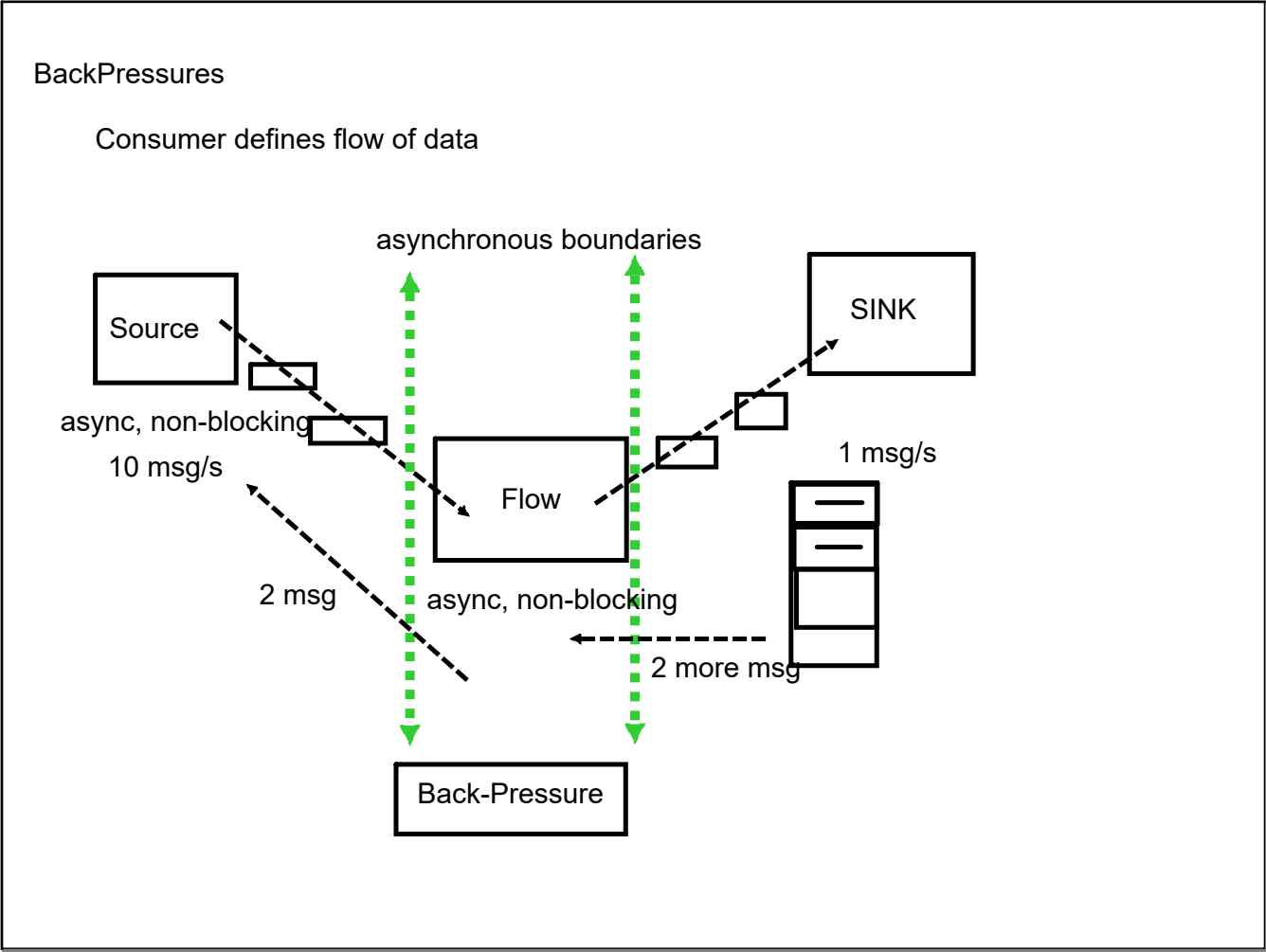
Test the actual actor

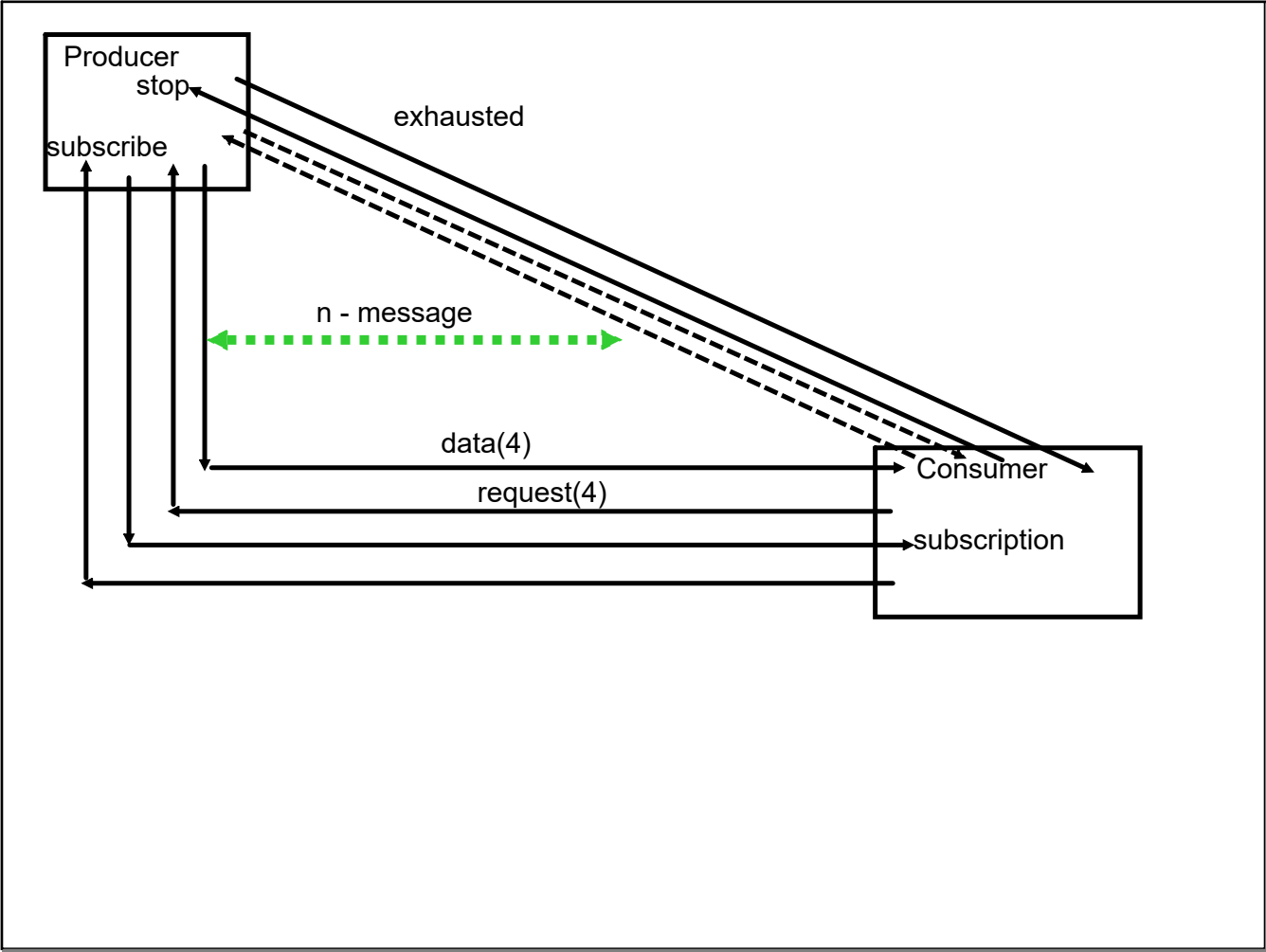
create a virtual guardian actor

popular testing is through logs









Source -> large number of integer

Flow -> transform int to string

Sink -> consume(print) those string

Akka Stream are lazy :

Source -> 100 records of employees

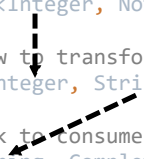
Flow -> will update employee records

Sink -> consume(print) those record

```
// source will give out large number of integers
Source<Integer, NotUsed> source = Source.range(0, 200000000);

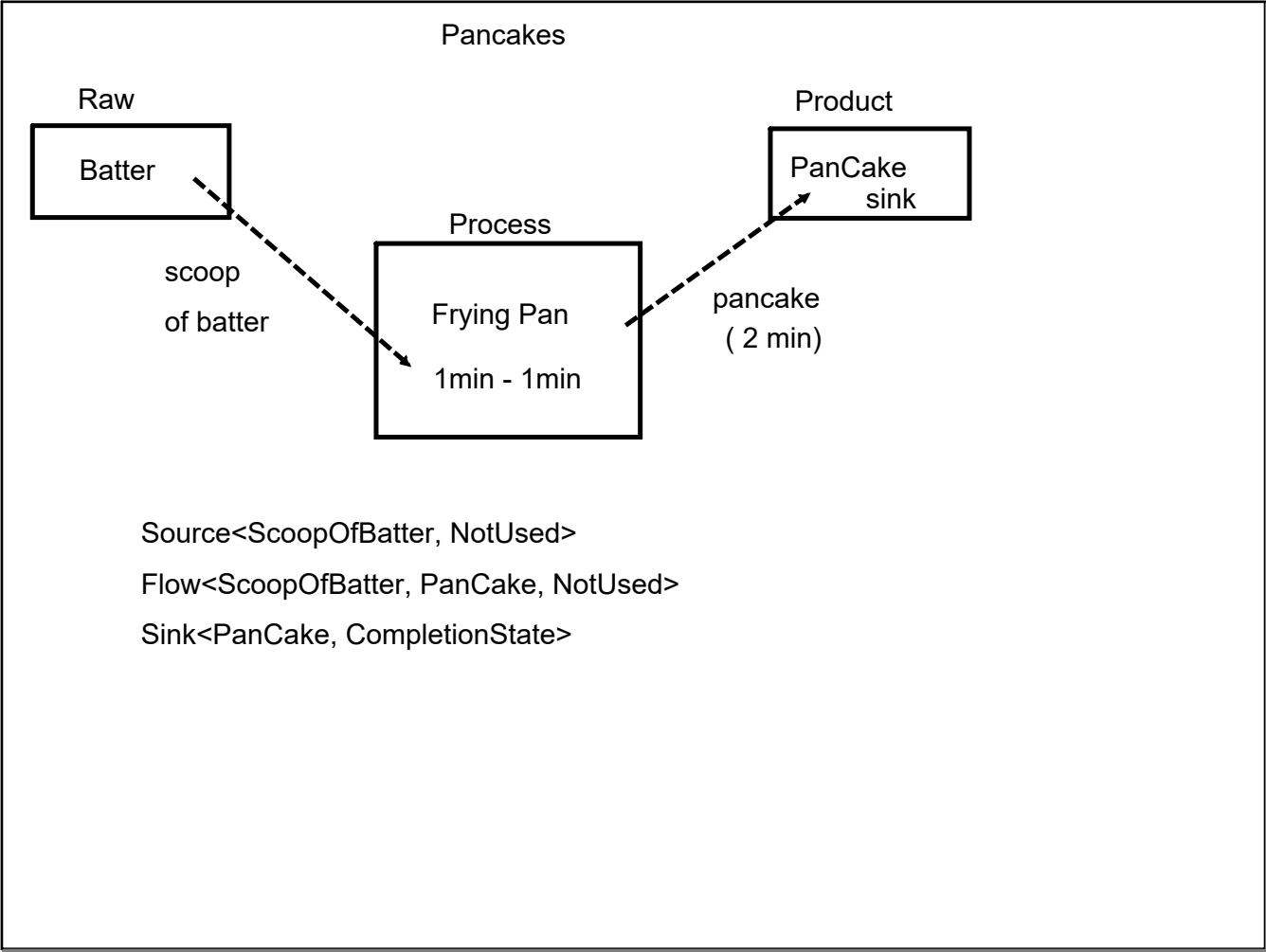
// flow to transform integer into string
Flow<Integer, String, NotUsed> flow = Flow.fromFunction(val-> val.toString());

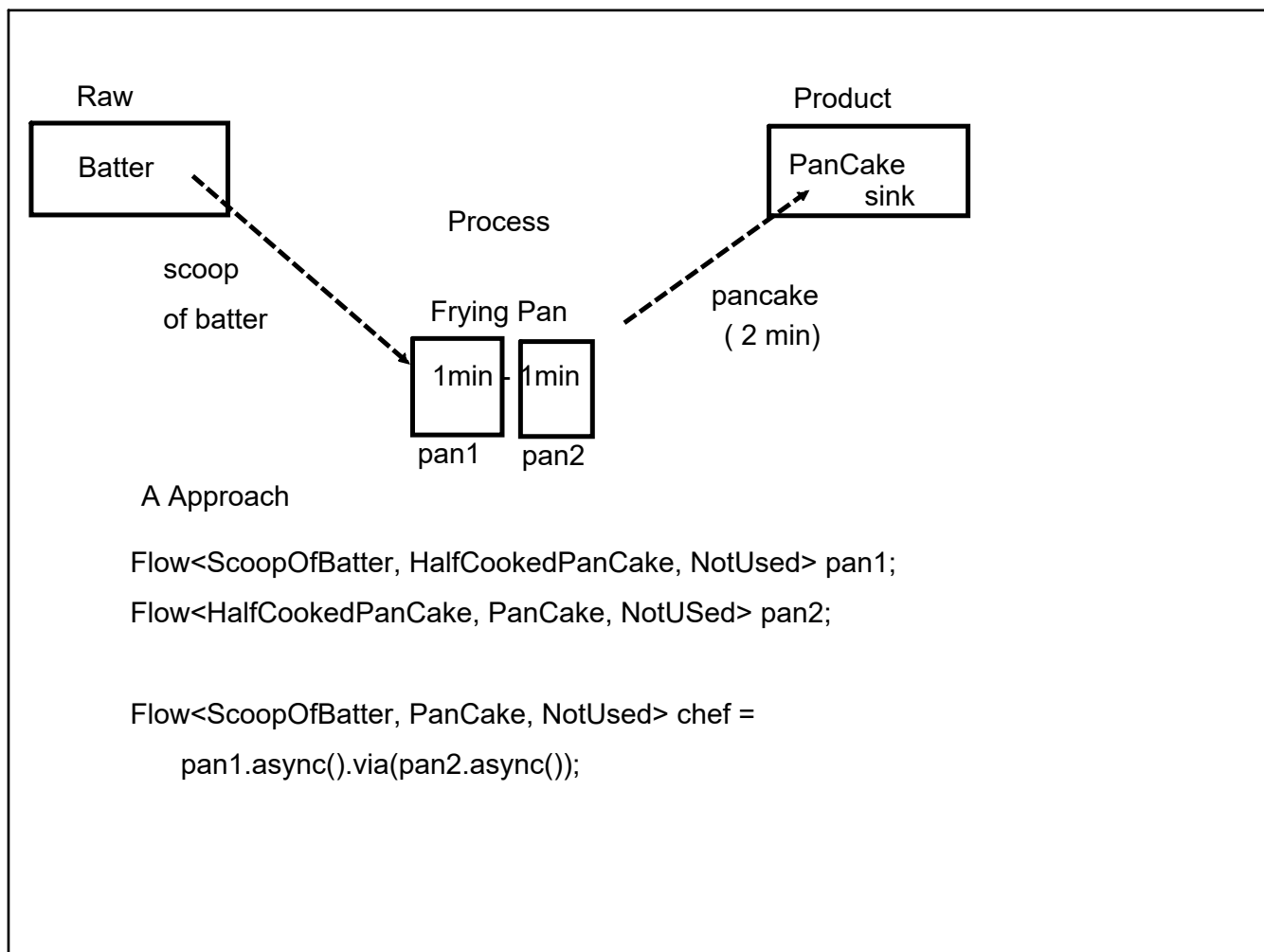
// sink to consume string (display them)
Sink<String, CompletionStage<Done>> sink = Sink.foreach(System.out::println);
```

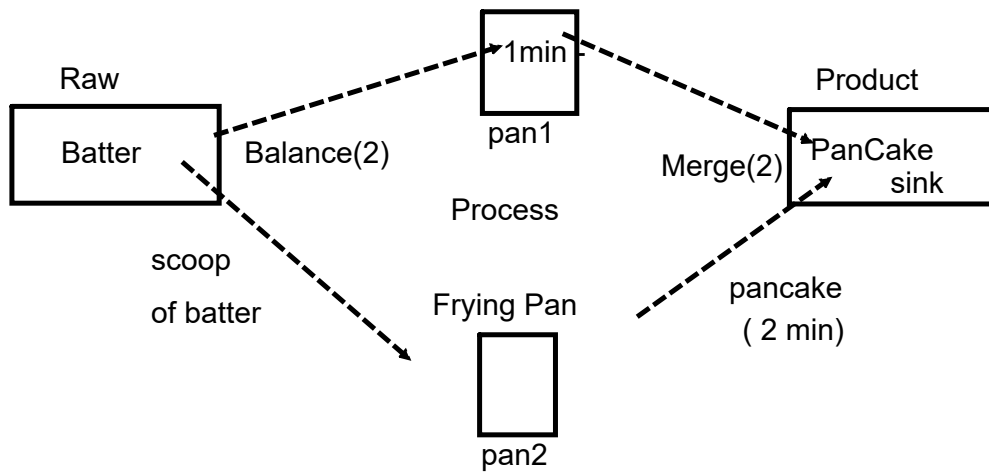


Stages can be expressed with fluent API









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

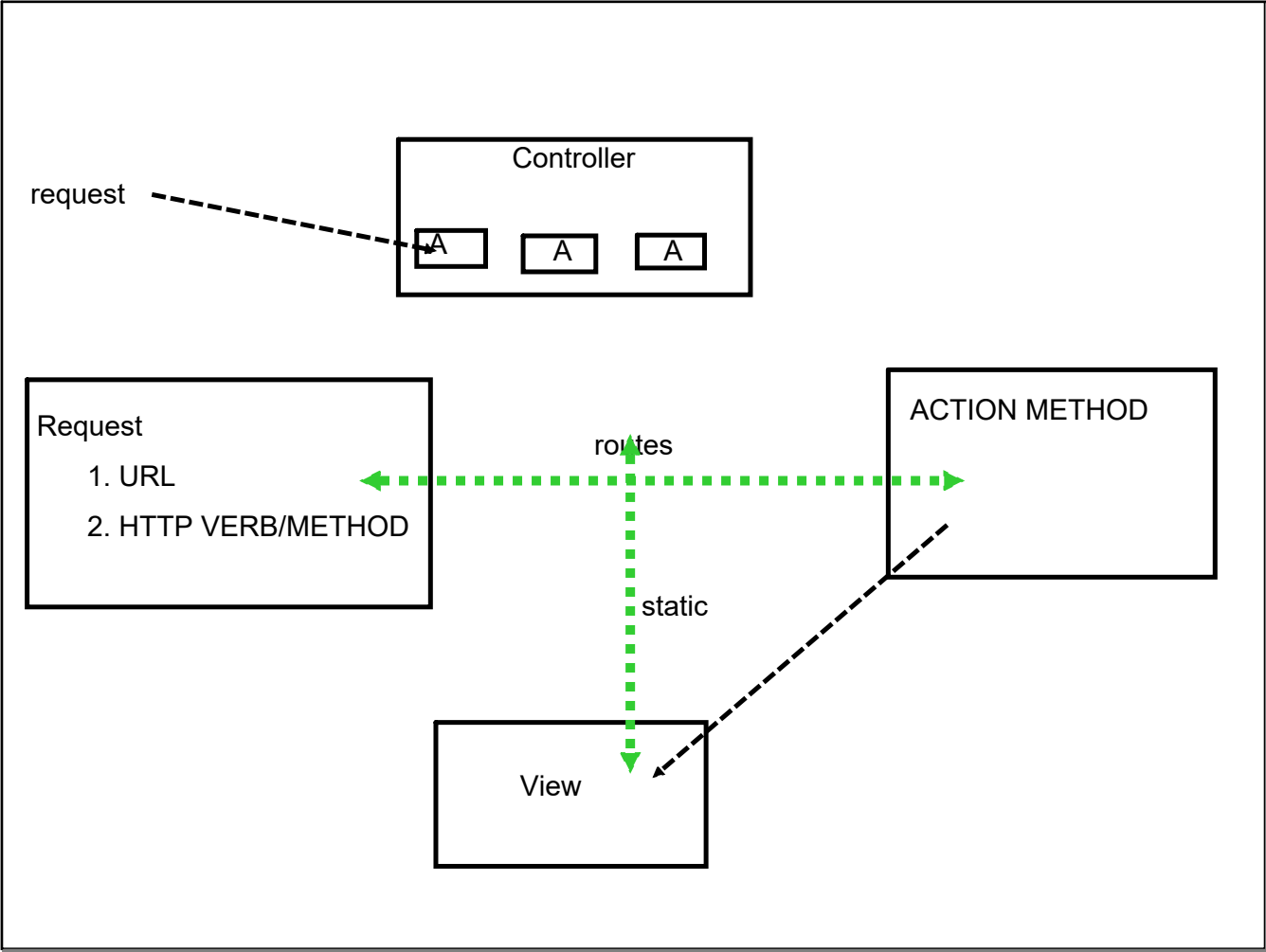
ctx(); HTTP Context

session()

request();

response();

flash() :



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



Routes

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

/student/3

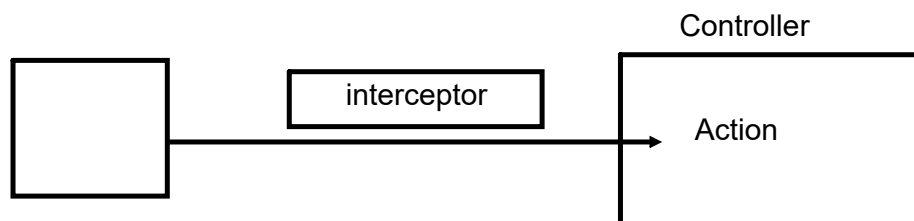
/student?id=3

```
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



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