Akka :

Play

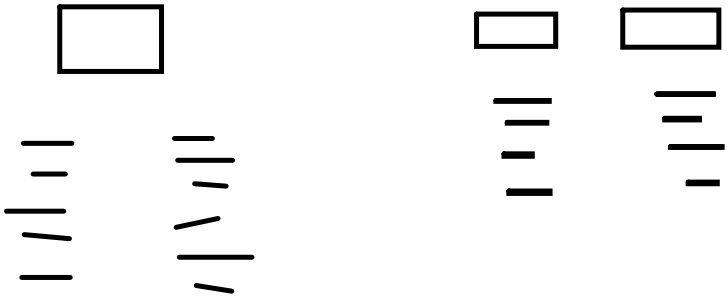
Concurrent Application :

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

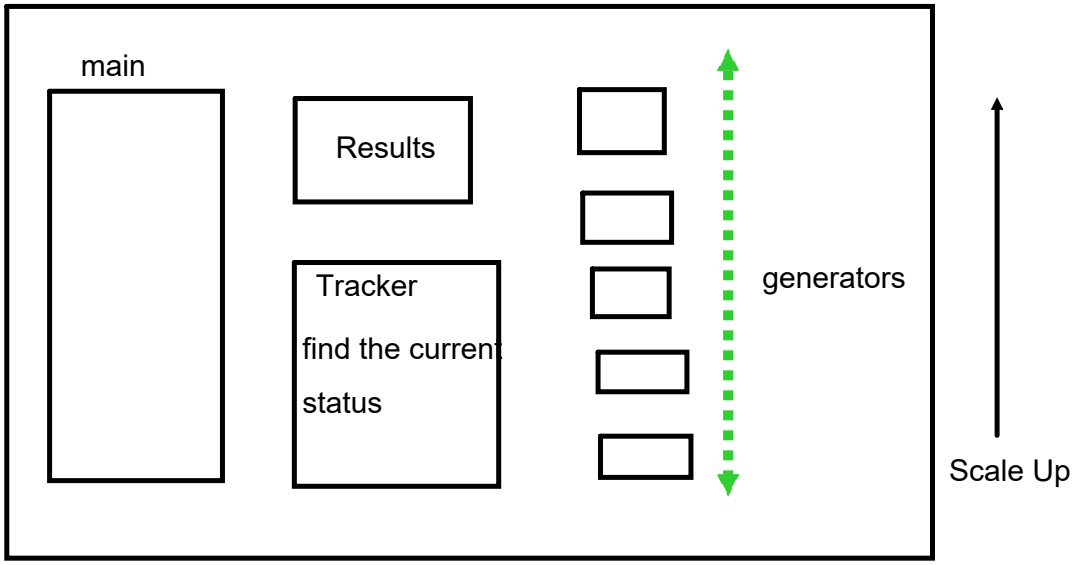
more than one thread try to change state of shared mutable data

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

BigInteger.nextProbablePrime()

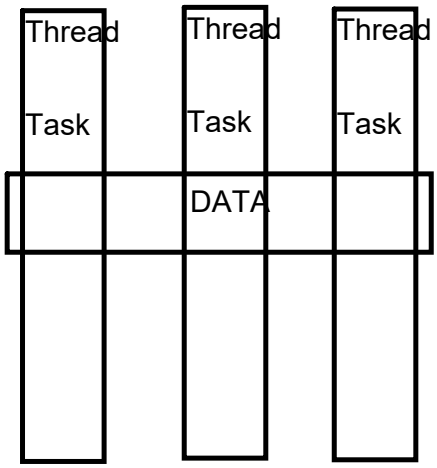
Single Threaded response : 24852 ms.

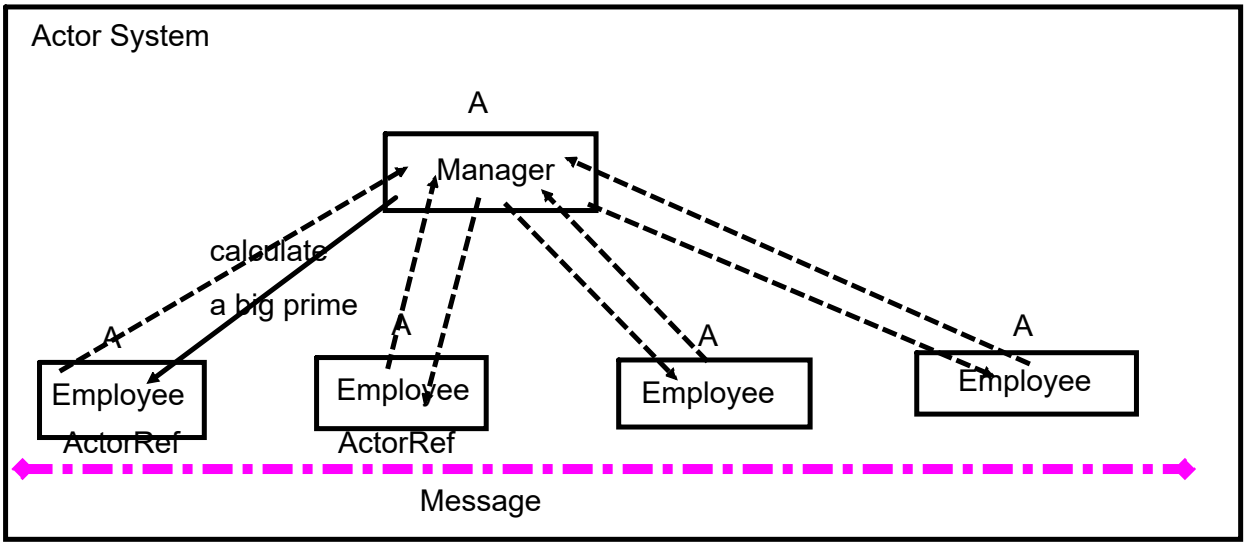
Multi Threaded env : 10957 ms.



Three primary concerns

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

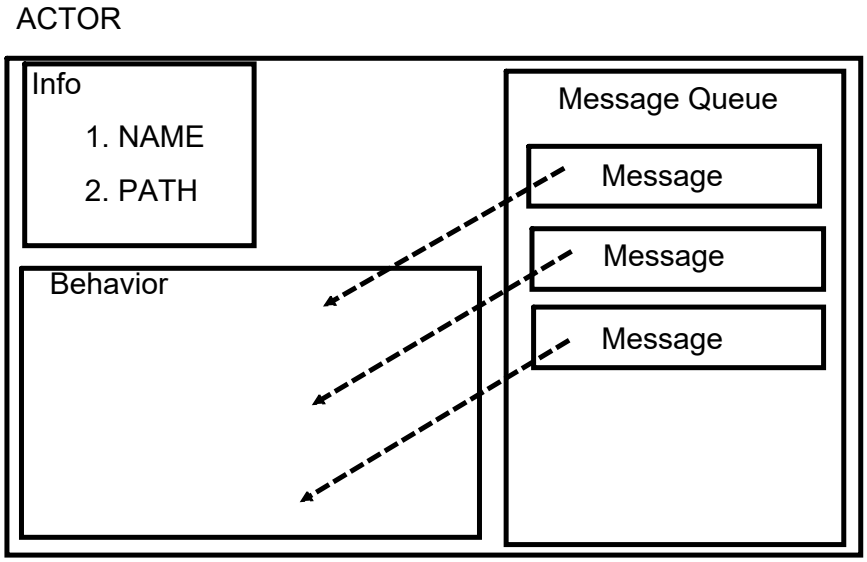

Thread
Task
Thread
Task
Thread
Task
DATA



All entity in system are actor

Interaction among them is through messages

Parent Actor and 0 or more child actor



=>Each Actor has its own thread

=> Actors won't share data

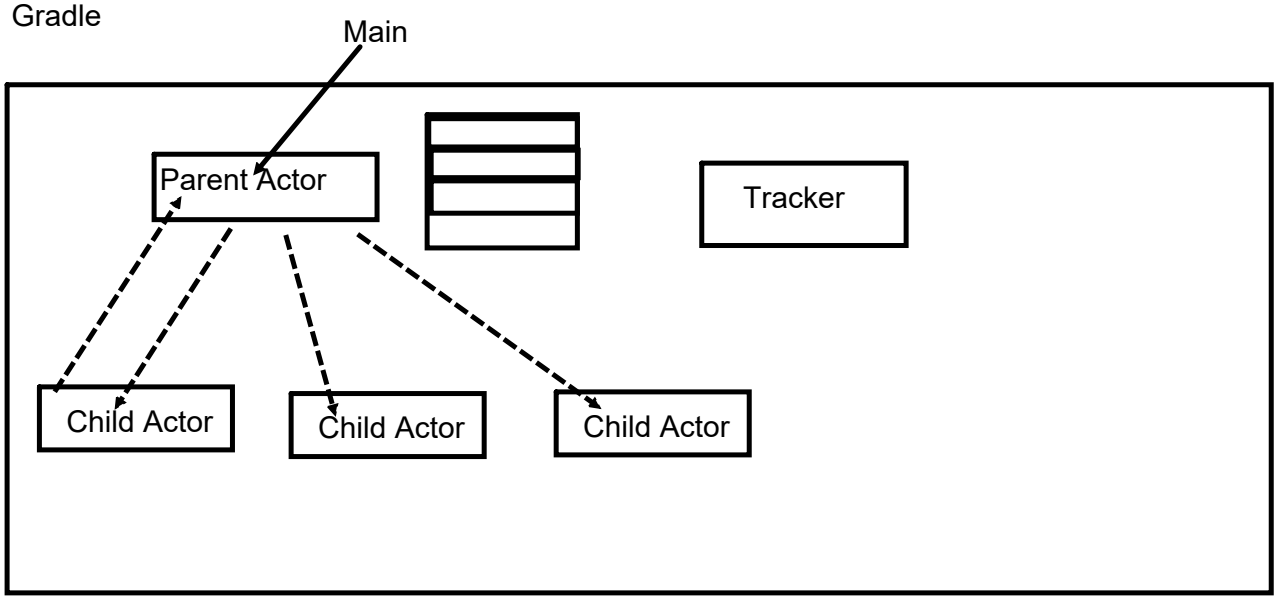
=> Messages must be immutable

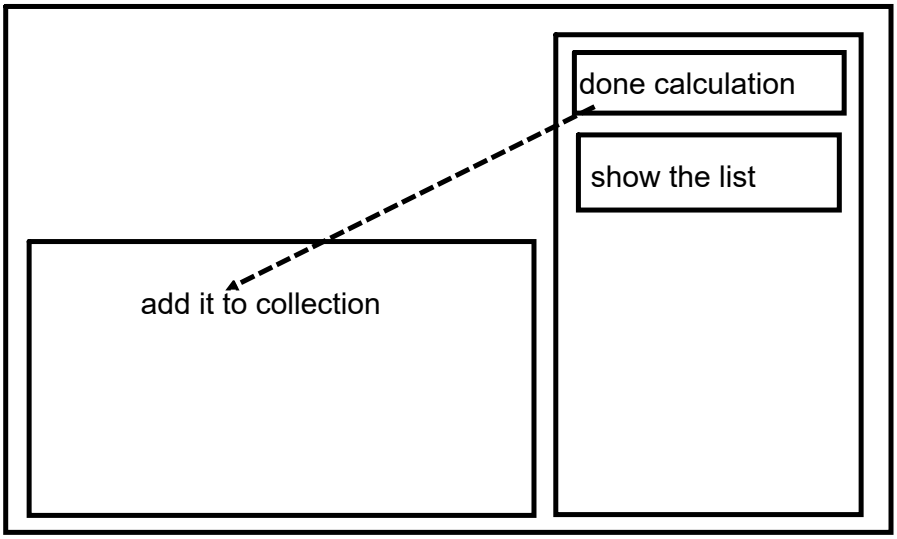
=> Messages are processed one at a time

SBT : scala build tool

Maven

Gradle



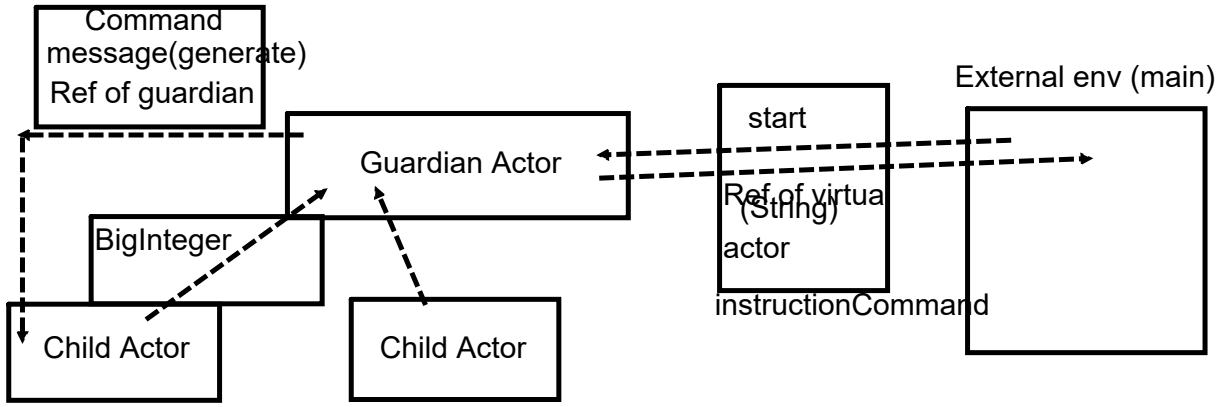

done calculation
show the list
add it to collection

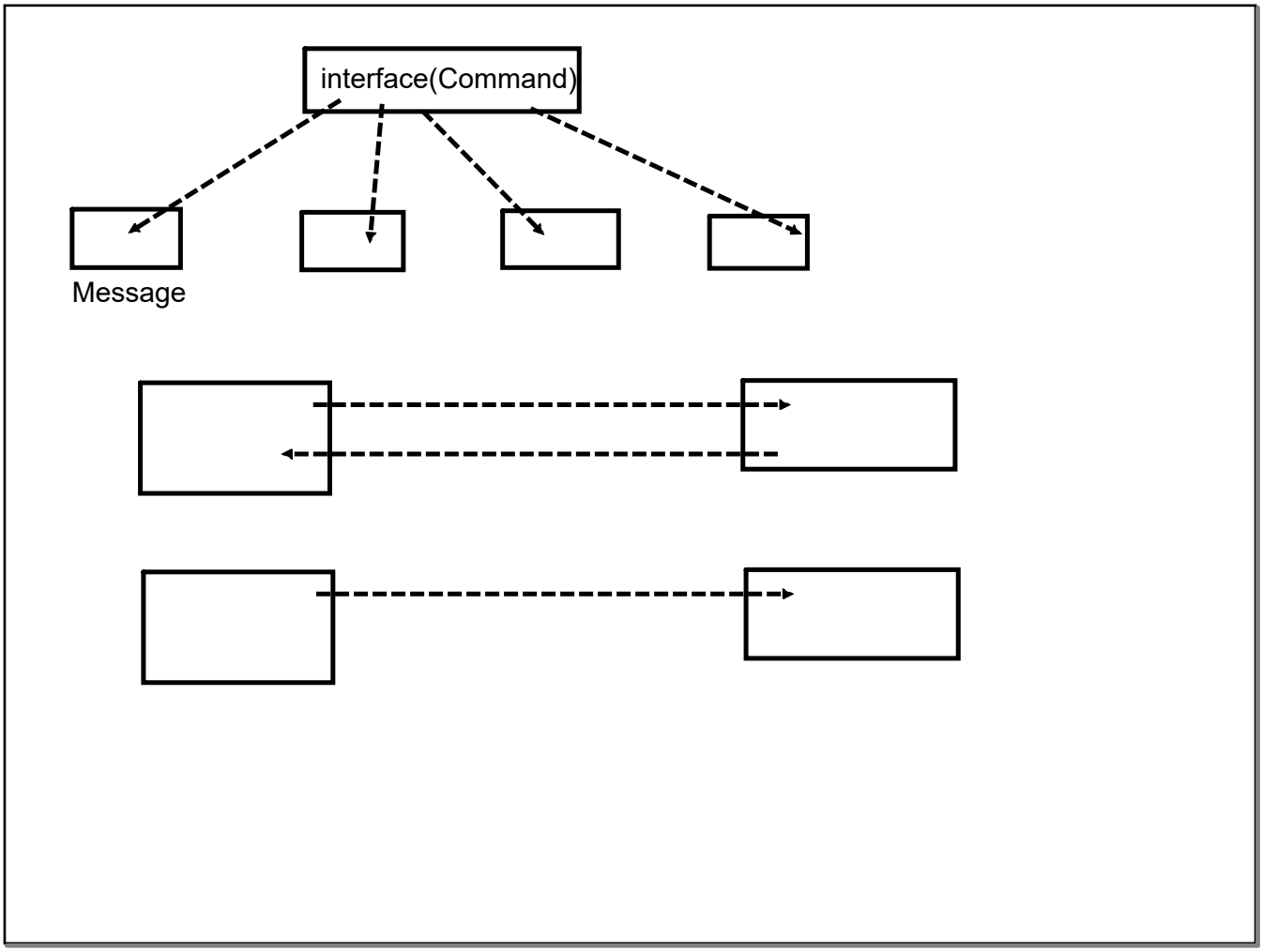
Actor

Guardian Actor

PrimeGenerator

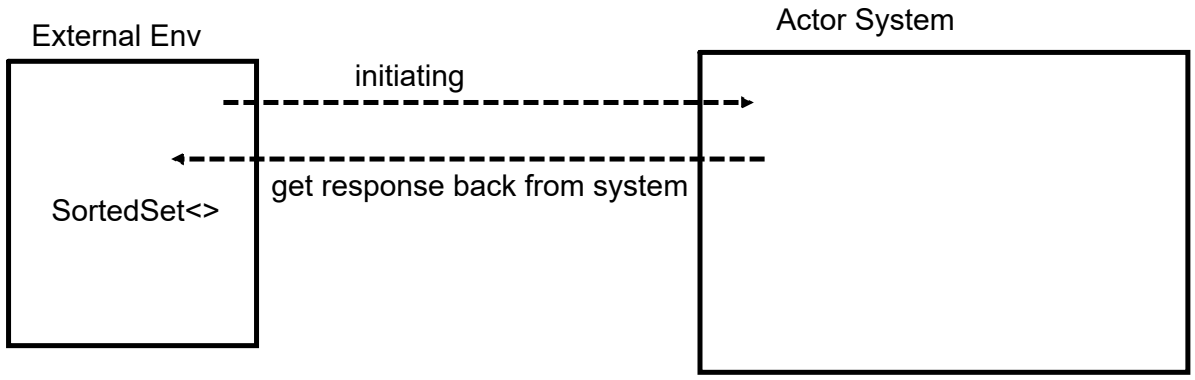
Message : any complex object (implements Serializable)




interface(Command)
Message

Use-Case

Simulate a race



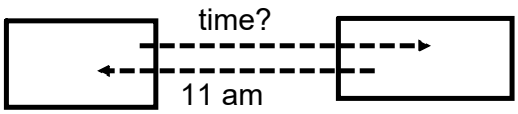
Actor interaction patterns

Fire n Forget

we send a message but does not waits for response

Request-Response (Request-Adapter response : ActorRef)

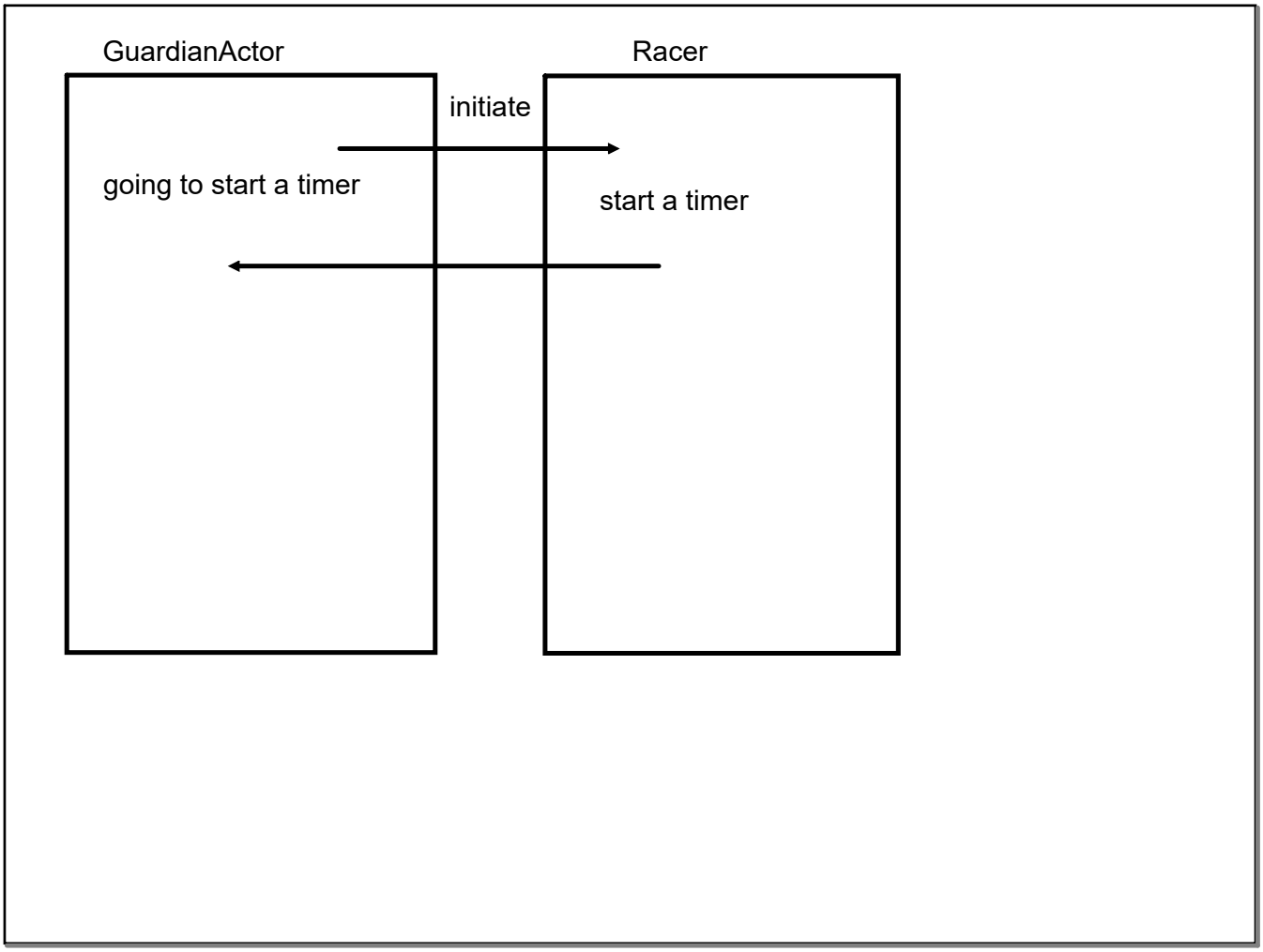
we send a message and expects the response



must have a response

must be time bounded

ask pattern


GuardianActor
Racer
initiate
going to start a timer
start a timer

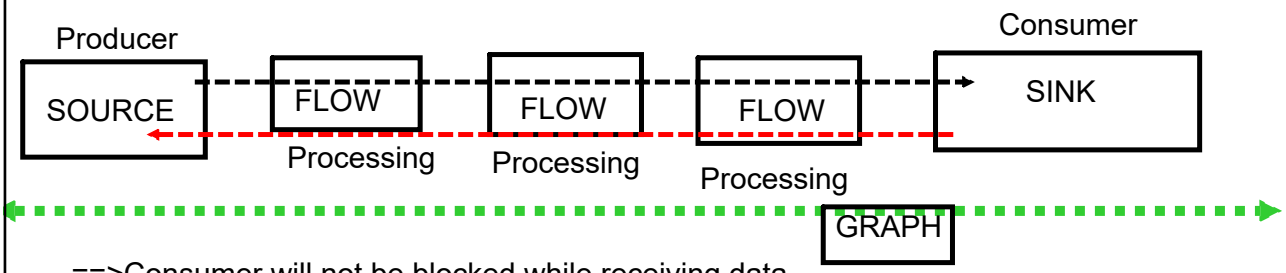
Writing unit test cases...

Test the actual actor

create a virtual guardian actor

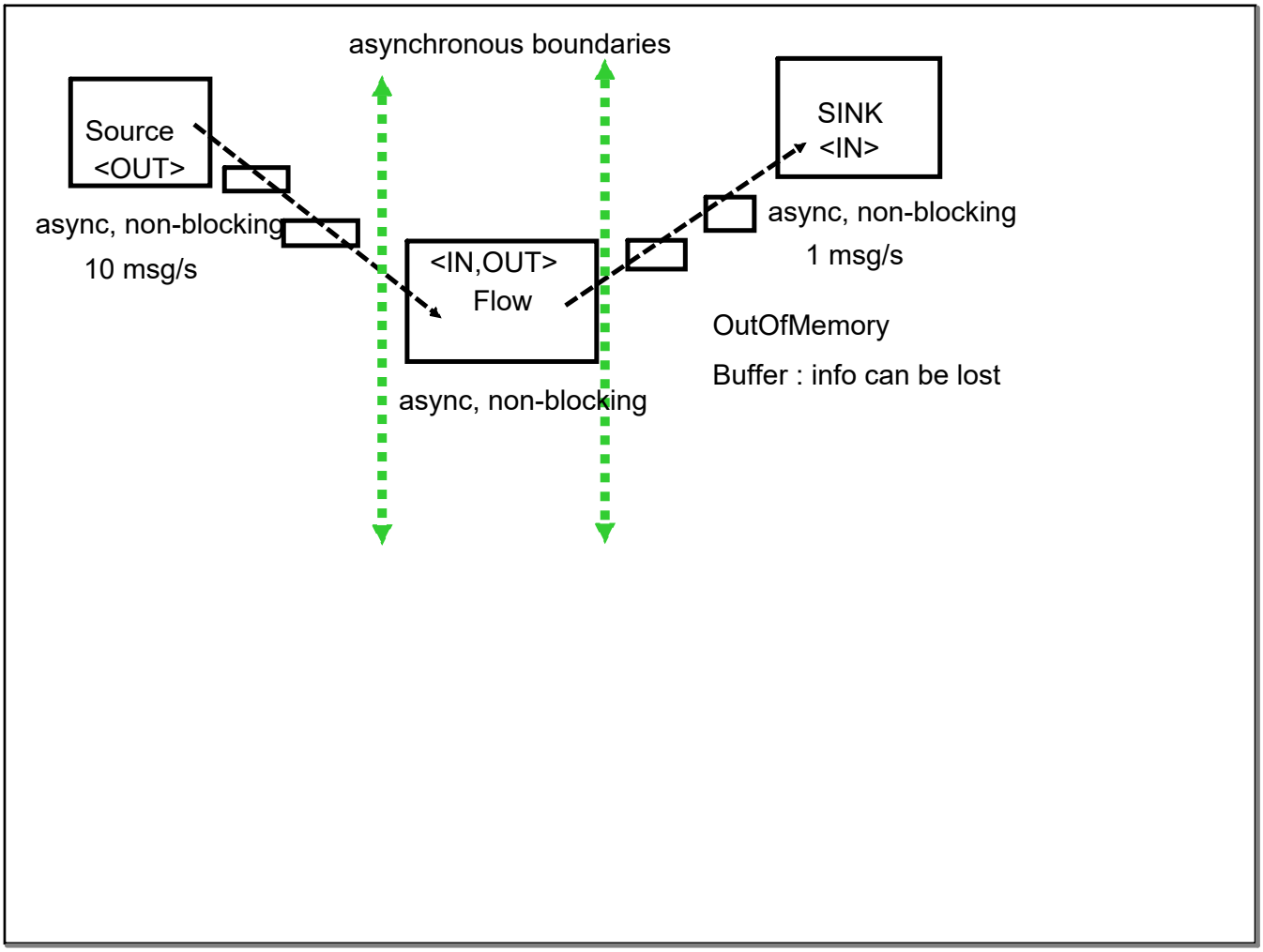
popular testing is through logs

Akka Streams



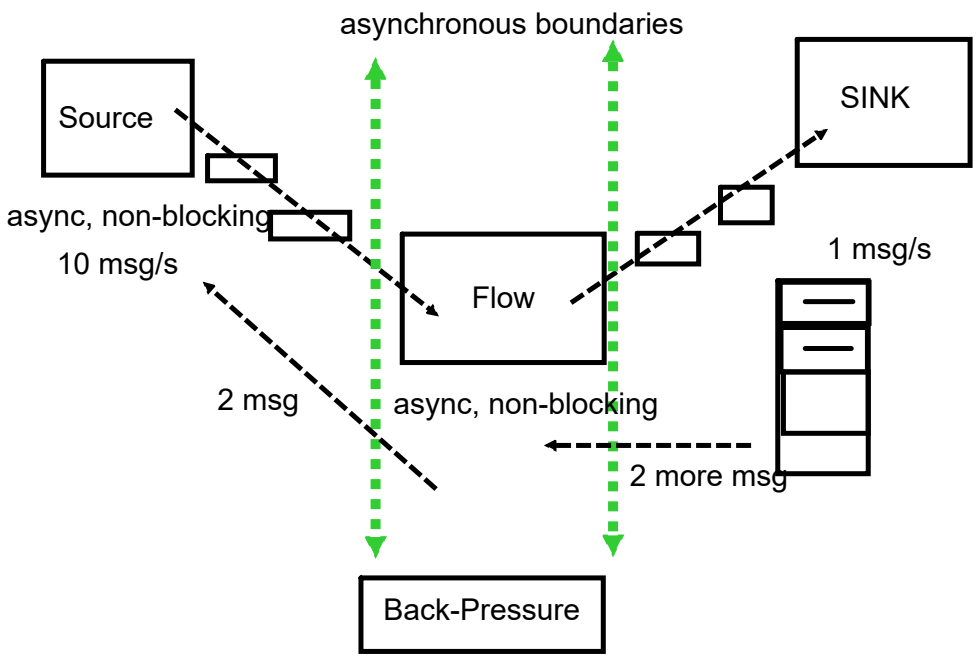
==>Consumer will not be blocked while receiving data

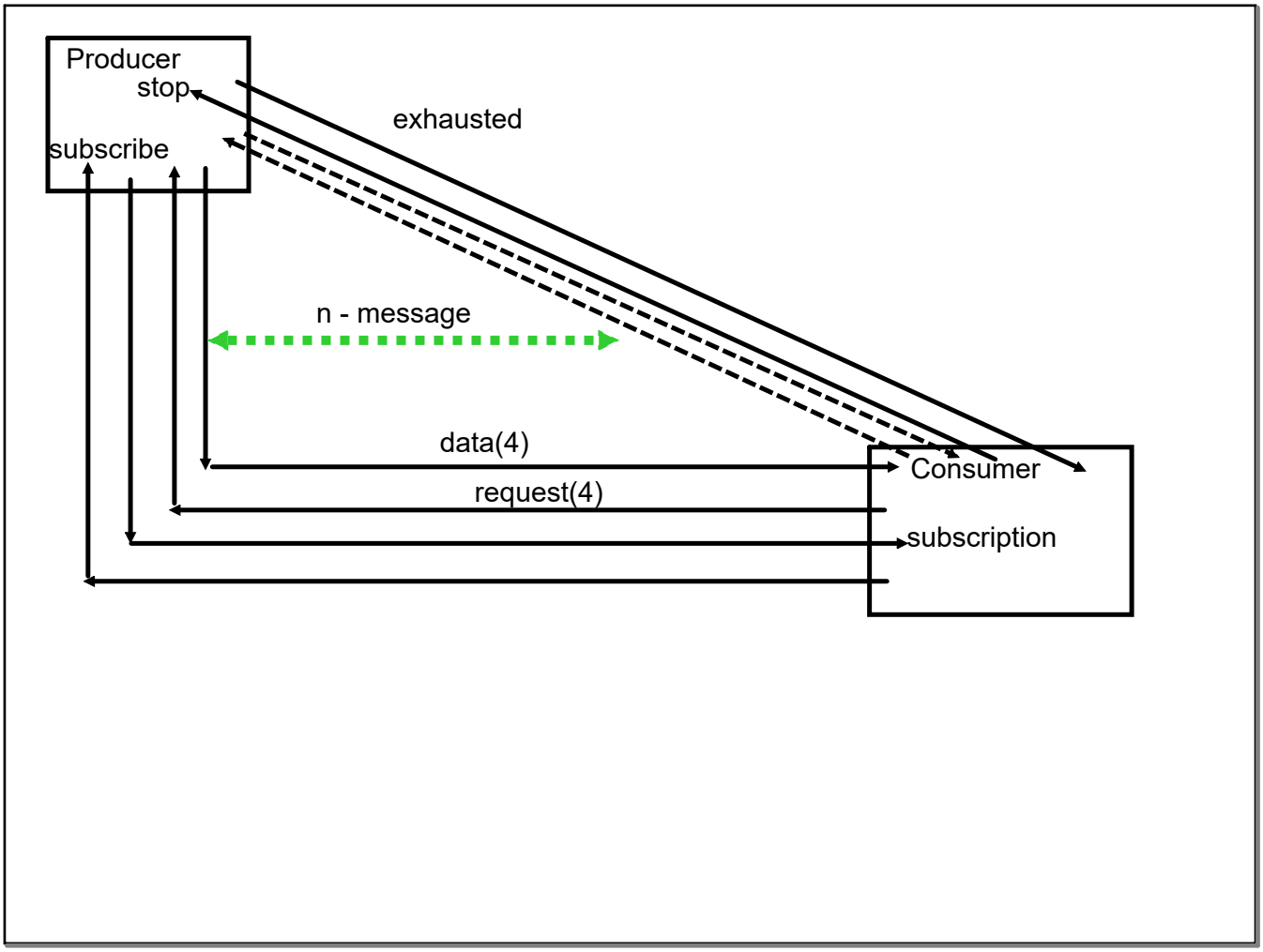
==> Consumer may continue doing its own activity and data will be received async in a seperate thread


asynchronous boundaries
Source
<OUT>
SINK
<IN>
async, non-blocking
10 msg/s
<IN,OUT>
Flow
async, non-blocking
1 msg/s
OutOfMemory
Buffer : info can be lost
async, non-blocking

BackPressures

Consumer defines flow of data




Producer
stop
subscribe
exhausted
n - message
data(4)
request(4)
Consumer
subscription

Source -> large number of integer

Flow -> transform int to string

Sink -> consume(print) those string

Akka Stream are lazy :

Source -> 100 records of employees

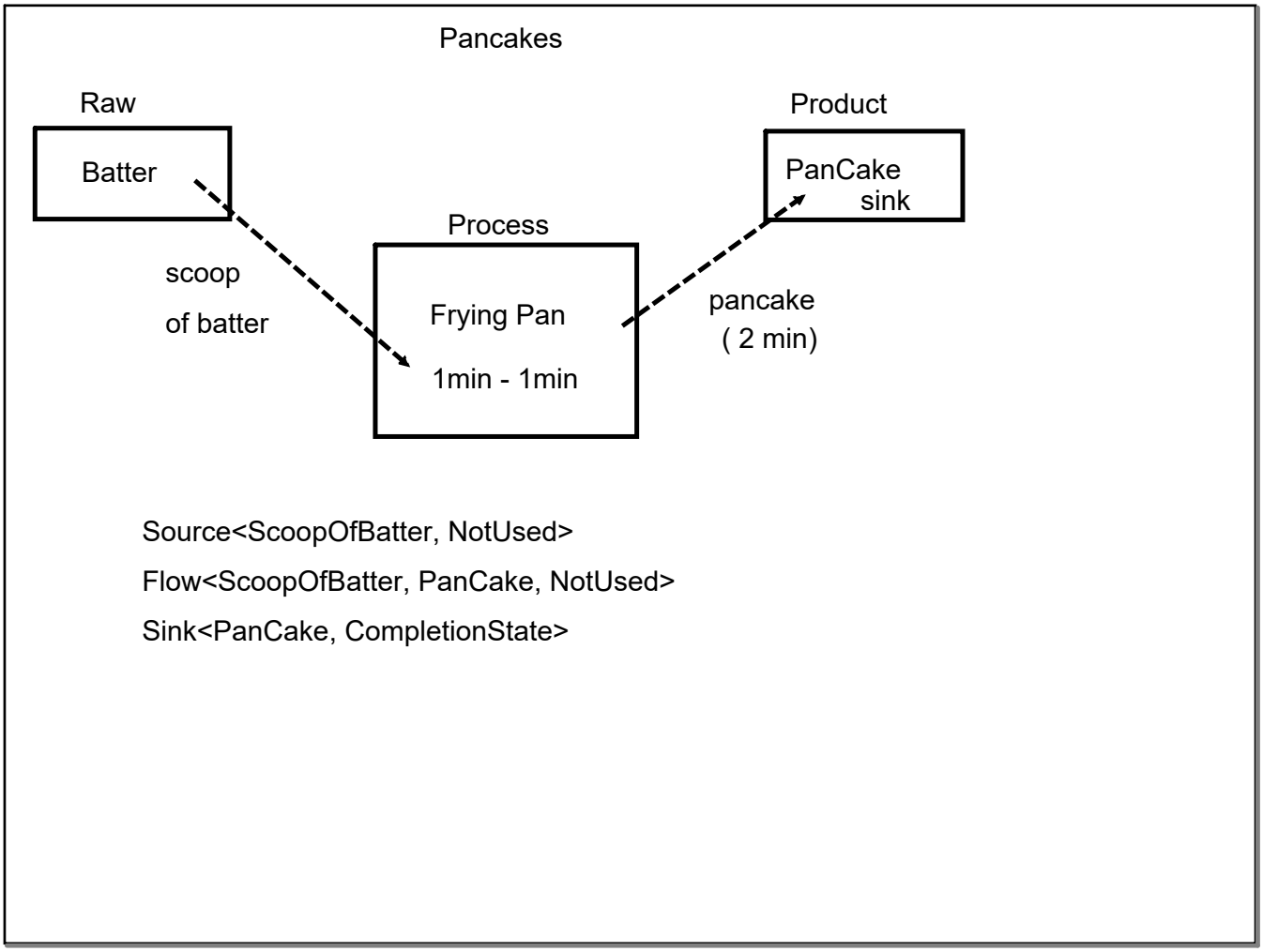
Flow -> will update employee records

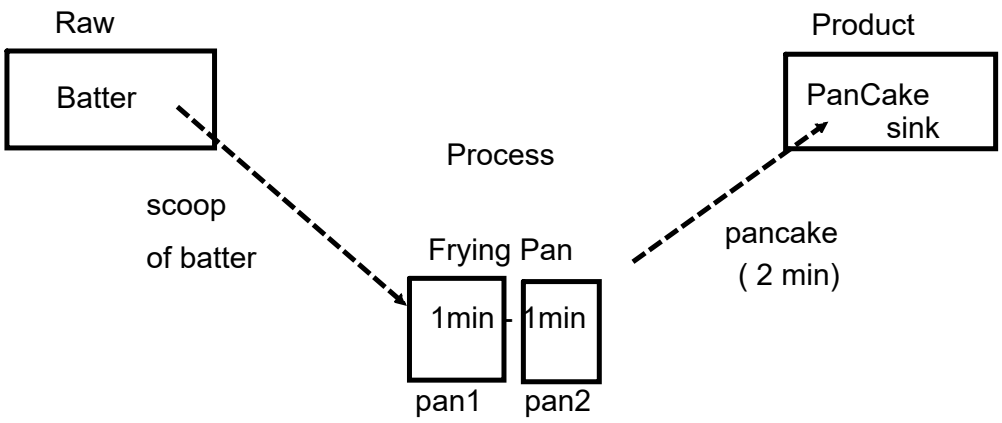
Sink -> consume(print) those record

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

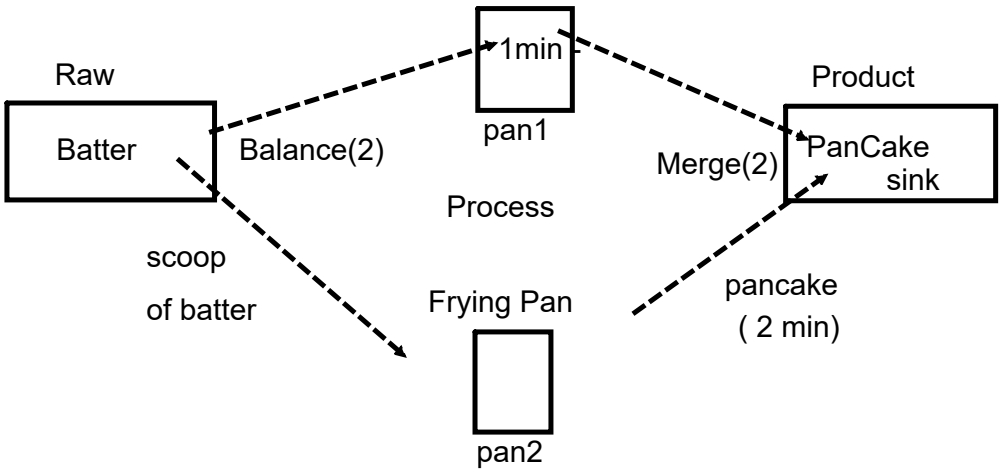

RS Library A
RS Library B


Pancakes
Raw
Batter
Product
PanCake
sink
Process
scoop
of batter
Frying Pan
1min - 1min
pancake
(2 min)
Source<ScoopOfBatter, NotUsed>
Flow<ScoopOfBatter, PanCake, NotUsed>
Sink<PanCake, CompletionState>



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
```