Introduction to Akka with Scala

Piotr Trzpil @ Warsjawa 2014

The plan

- Scala in a pill
- Akka overview
- A problem to solve: processing logs
- Part 1: Actor basics
- Part 2: Actor lifecycle
- Part 3: Routing and remoting

Case classes, objects, pattern matching

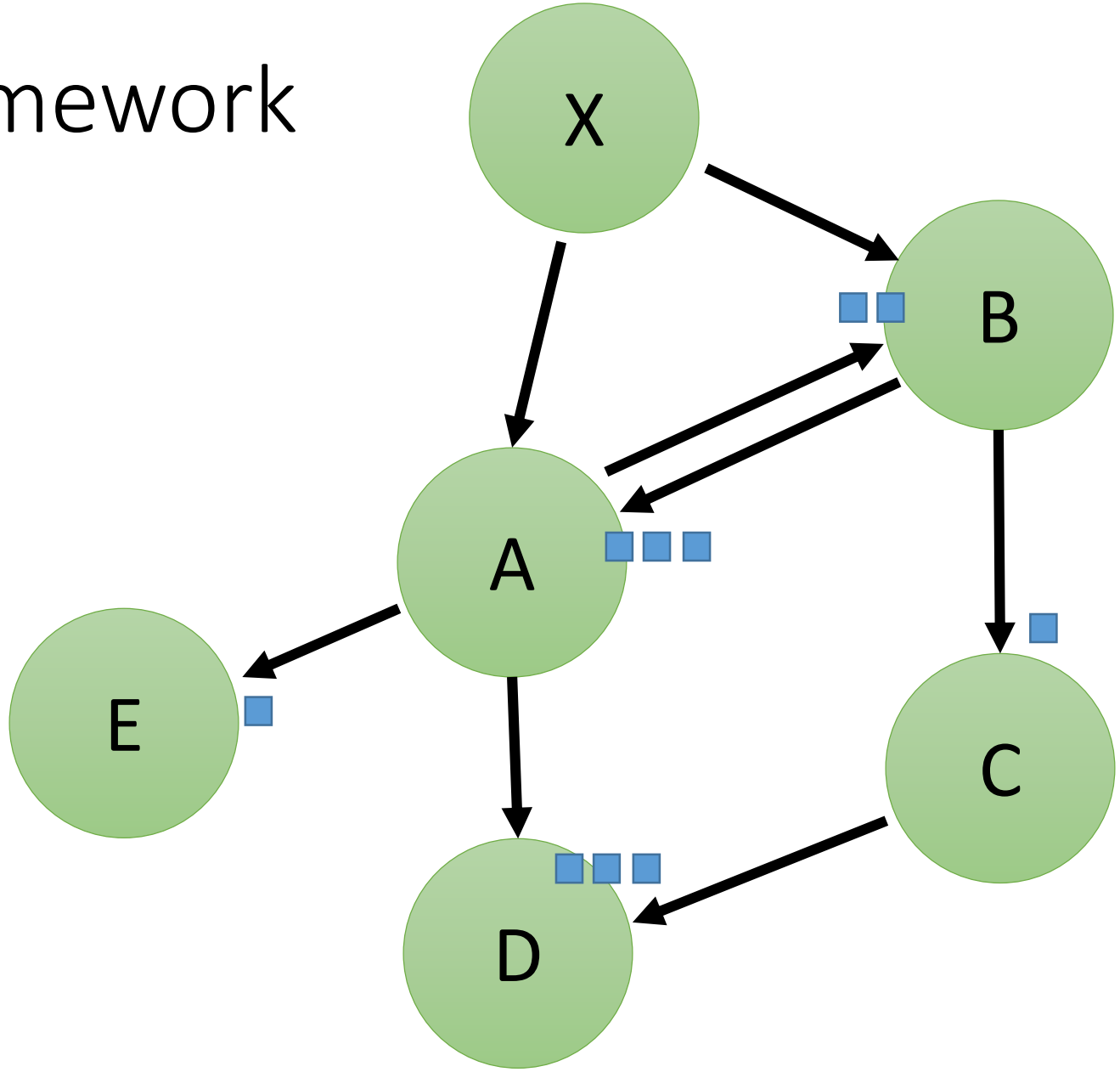
```
case class Thing(name: String)

object PatternMatchingTests {
    def checkList(things : List[Thing]) = things match {
        case List() => println("empty")
        case List(Thing("a thing")) => println("specific")
        case List(Thing(someName)) => println(someName)
        case List(Thing(n)) if n.size > 4 => println("large")
        case _ => println("anything")
    }
}
```

Akka – an actor framework

Actor system:

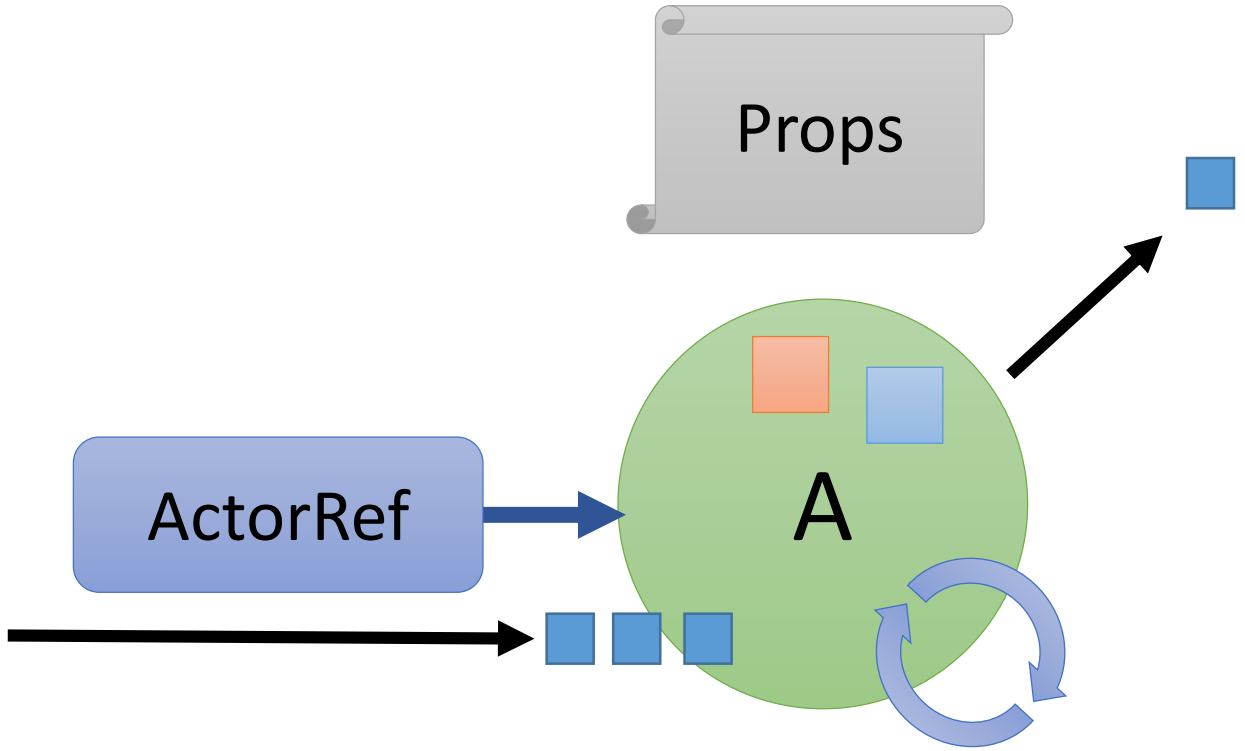
- An object model composed of actors
- Communication by asynchronous message passing
- Message delivery (in general) not guaranteed
- Java & Scala API

<u>Microservices in one app</u>



An actor

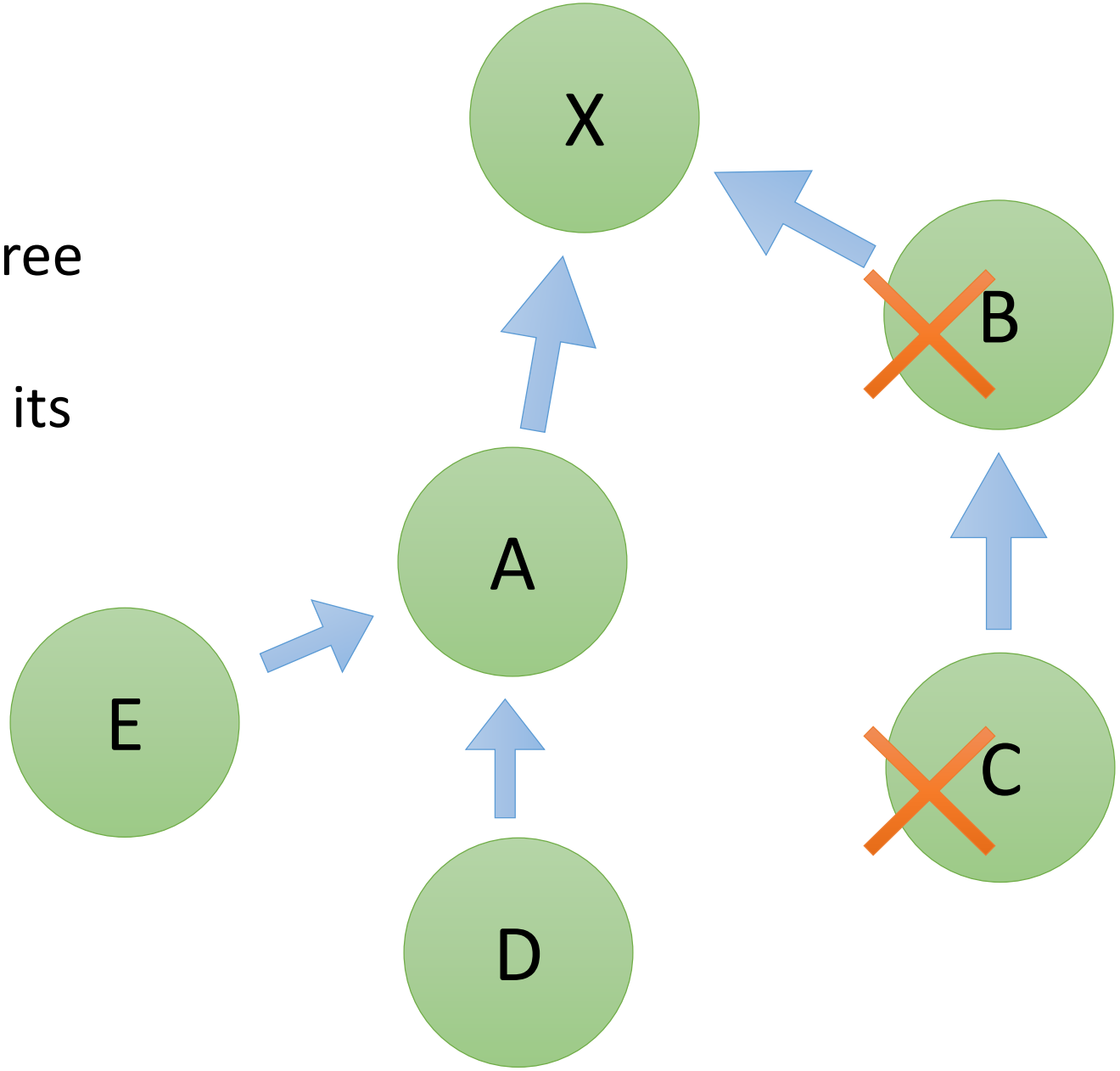
- Reacts to messages
- Is a ‘living’ object
- Has a:
  - Name
  - Location
  - Mutable state
  - Lifecycle
  - A mailbox
  - A parent
  - Optionally, children
  - Changeable behavior



Internally synchronous!

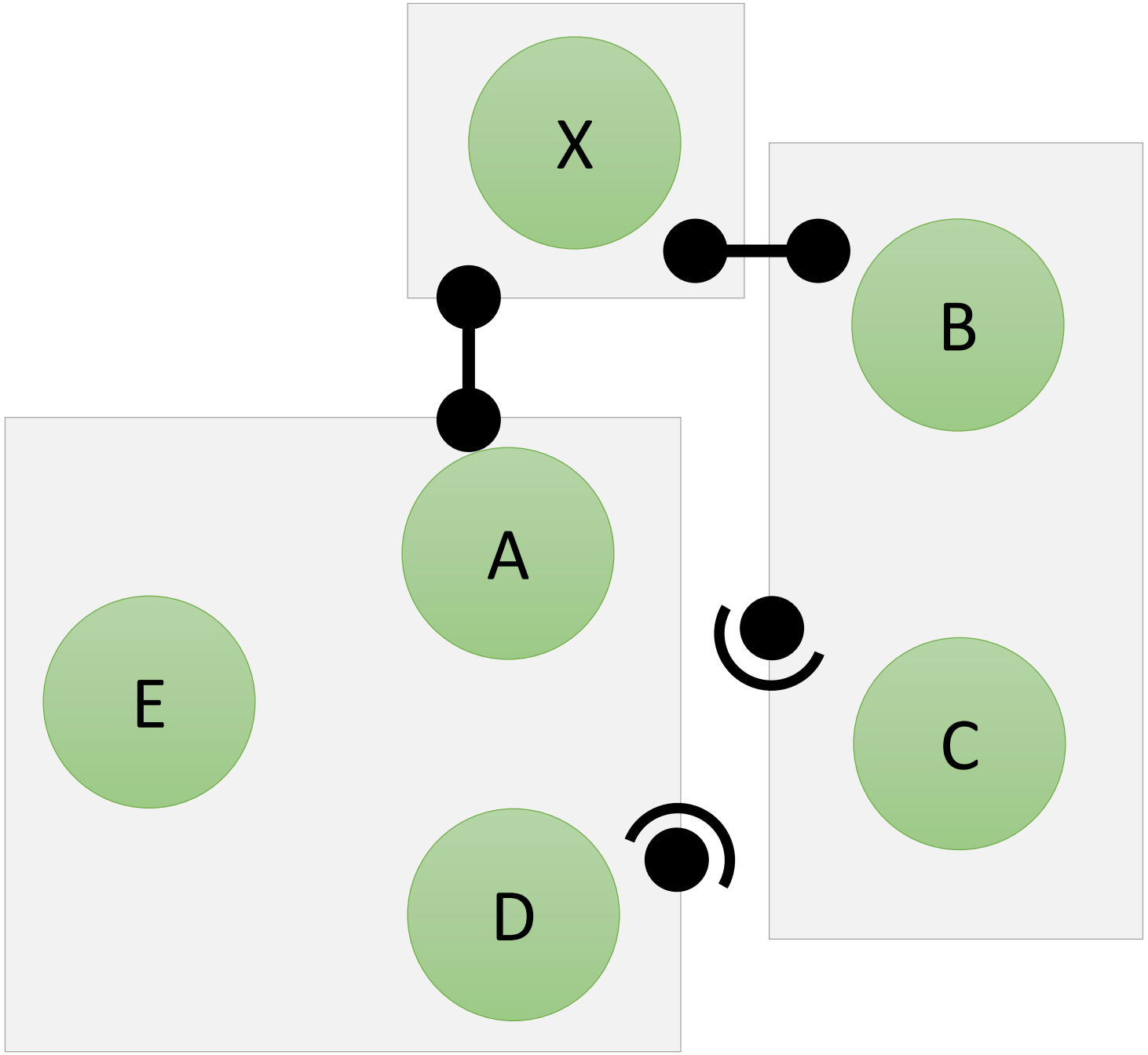
Actor hierarchy

- Actors form a supervision tree
- A child's life is governed by its parent
- Failure is local
- Failure will happen…



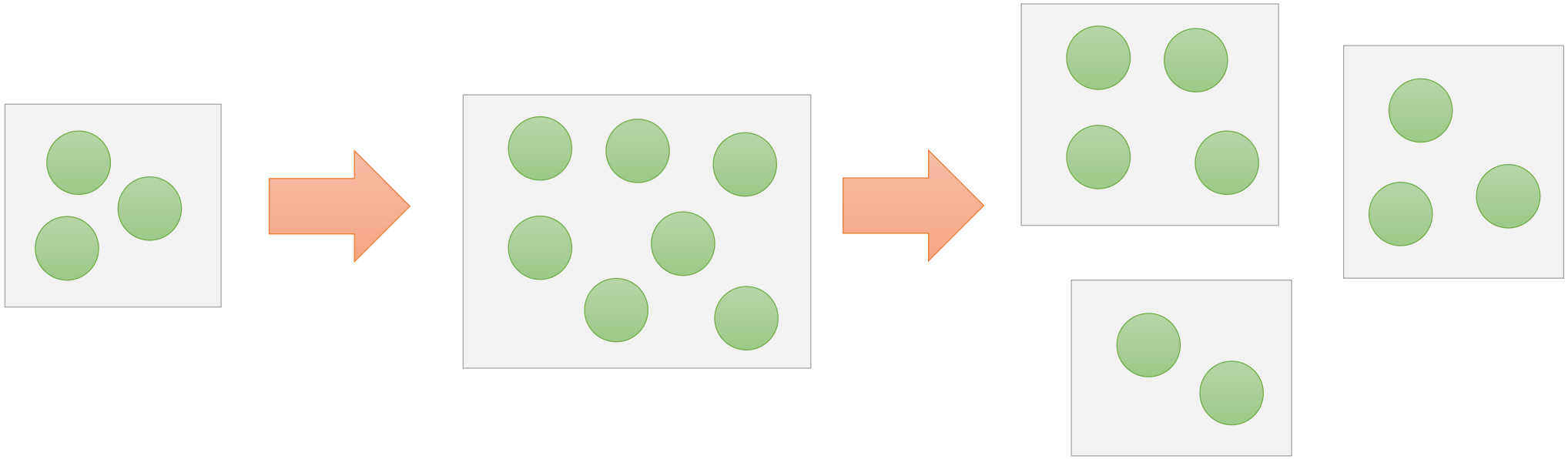
Actor’s location

- Same JVM, other JVM, other machine
- Can be transparent to the application



What does it all give?

- Actors are independent processing cells
- It’s fast, non blocking
- It gives a horizontally scalable application architecture
- Composes well with regular objects

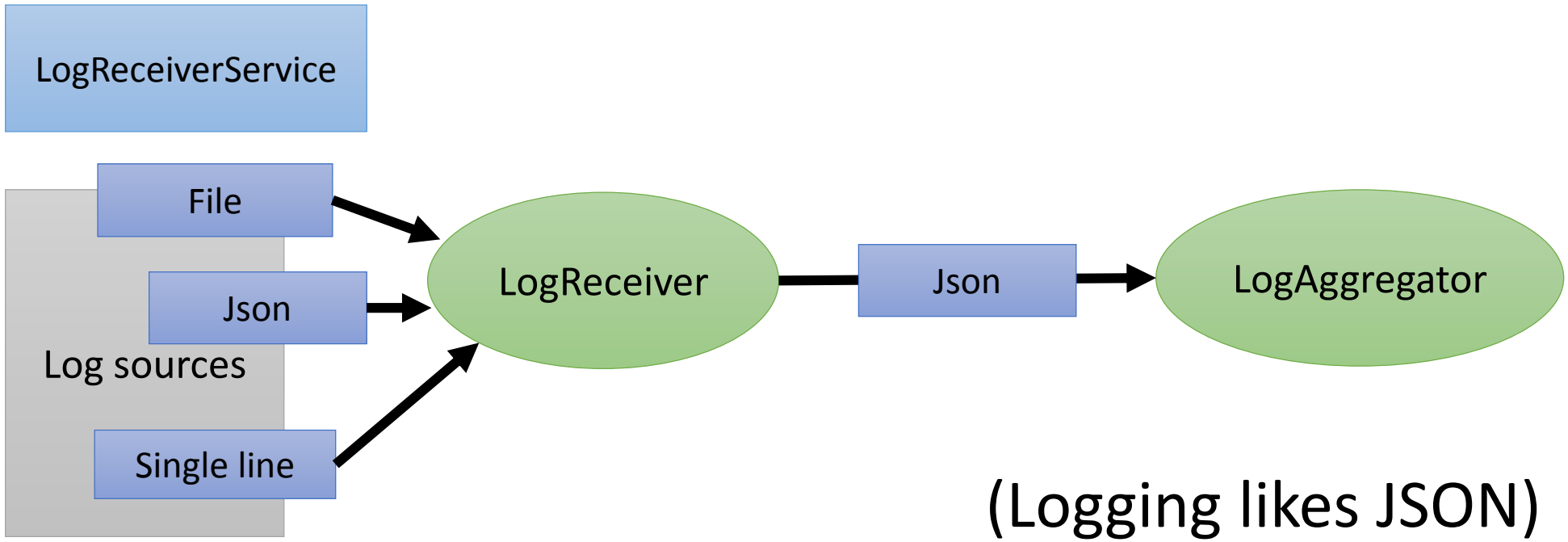


Spray == Akka HTTP

- Lightweight, fast, layered HTTP library (+server)
- Layers: TCP -> HTTP server -> Routing
- Built on actors
- Will merge with Akka
- Composable, declarative DSL for routes

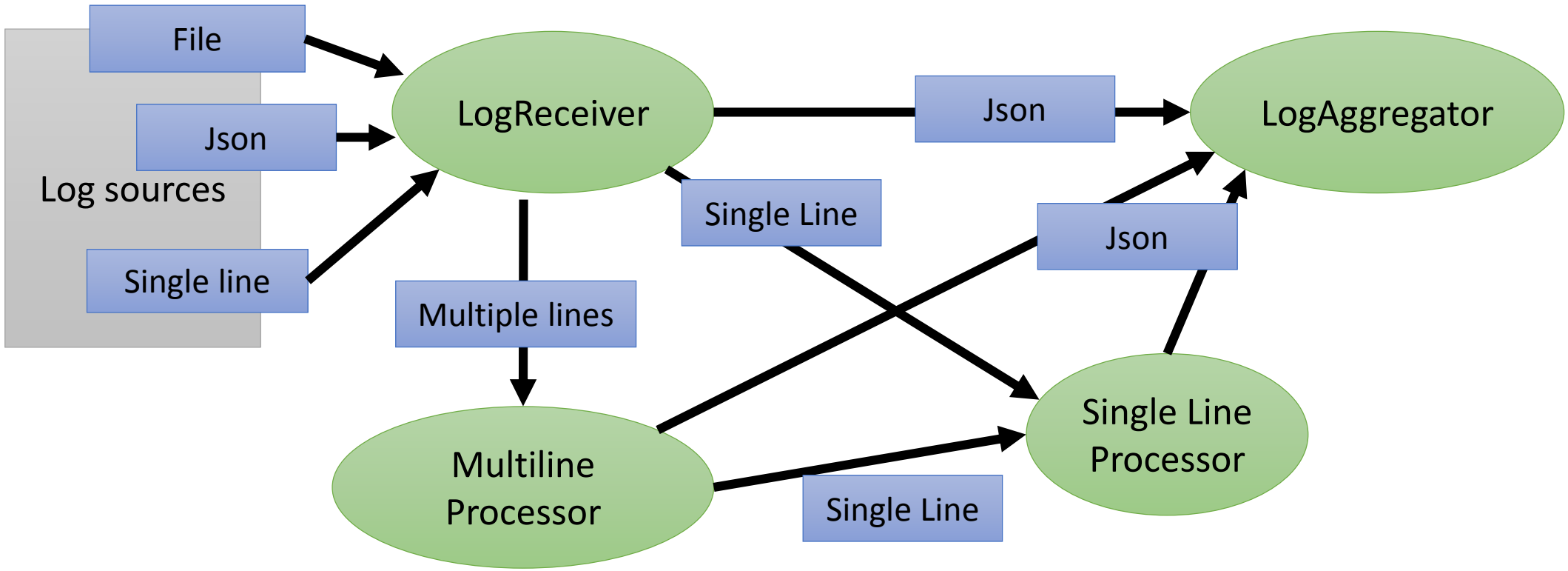
```
val routeLine =
  path("log-line" / Segment) { appName =>
    post {
      handleWith { line: String =>
        s"Got it: $line from app: $appName"
      }
    }
  }
```

The problem to crunch – log processing

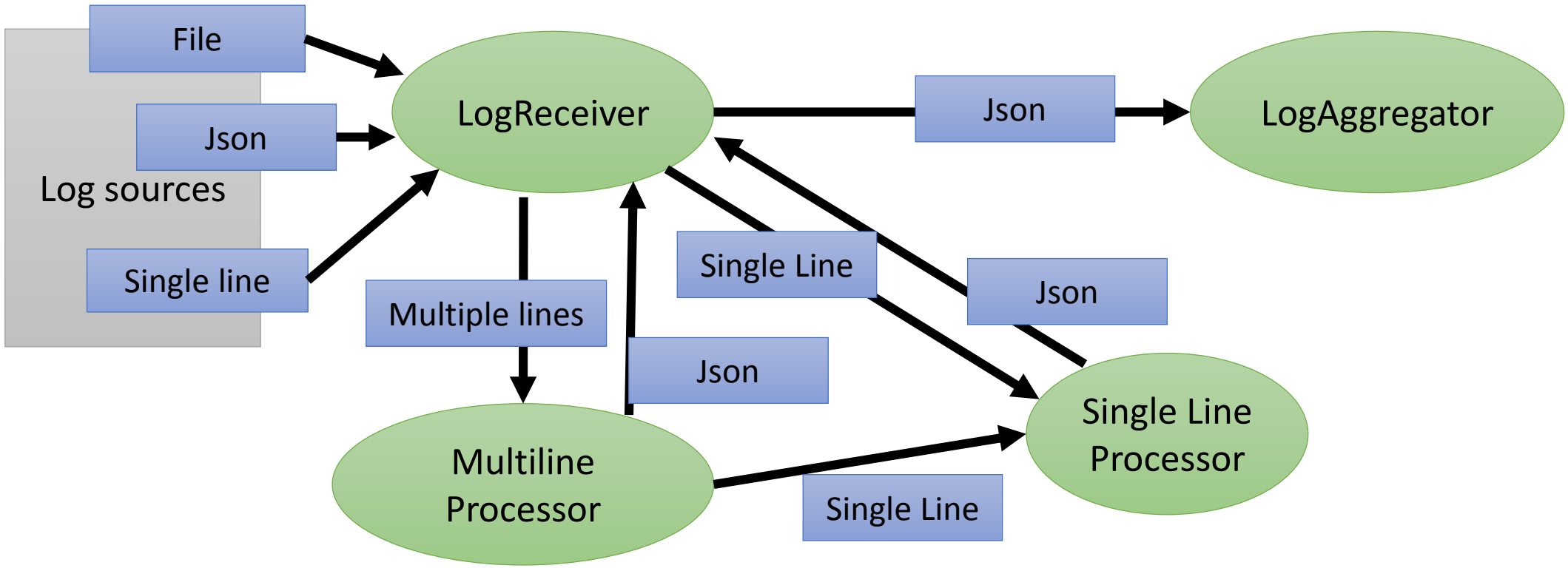


(Logging likes JSON)

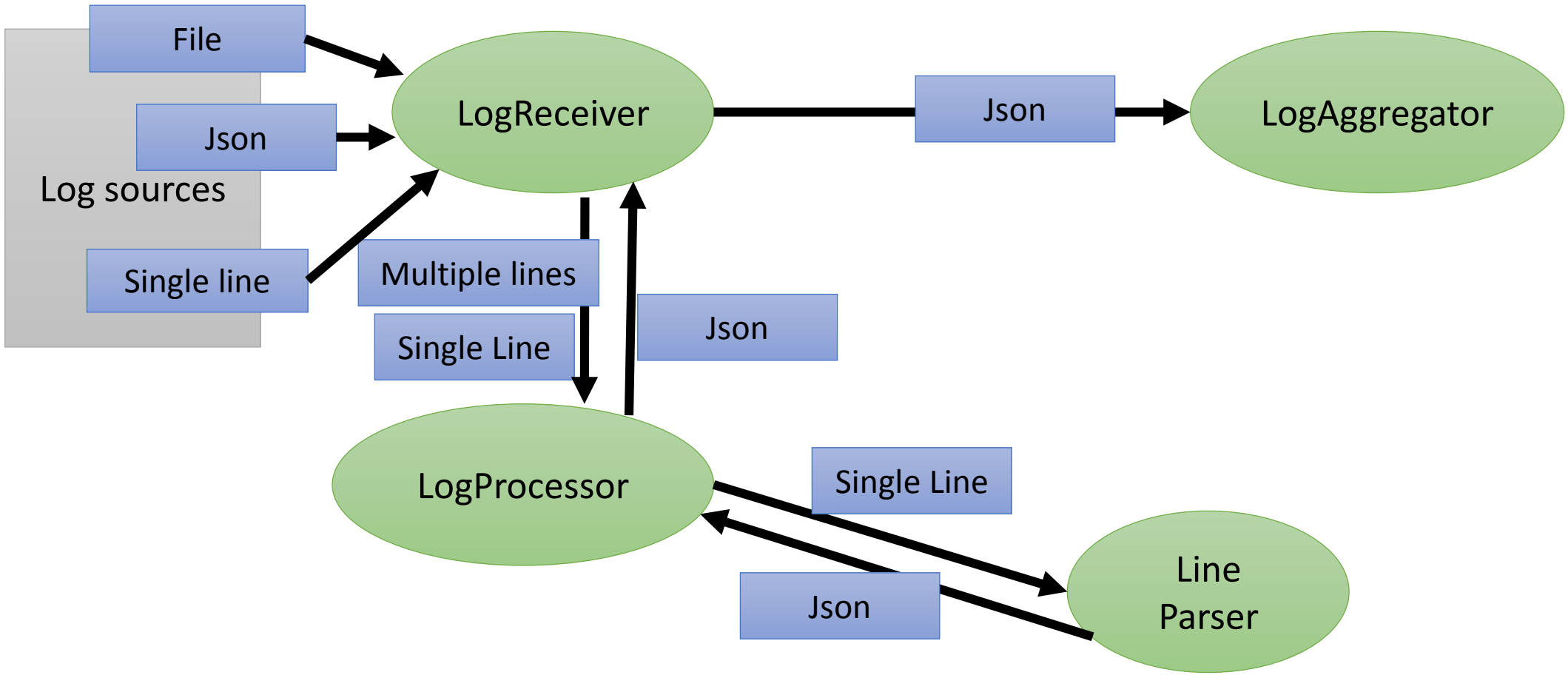
Possible Architecture



Another possibility

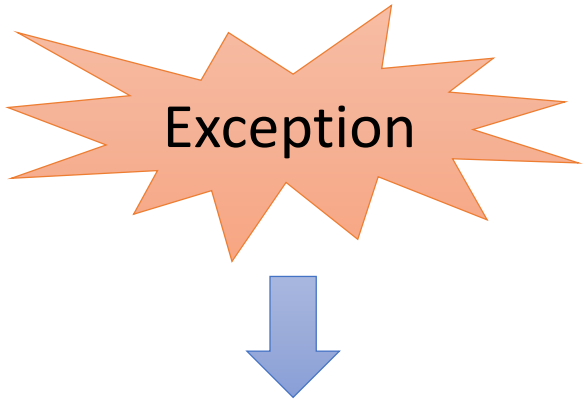


Another one – concepts changed

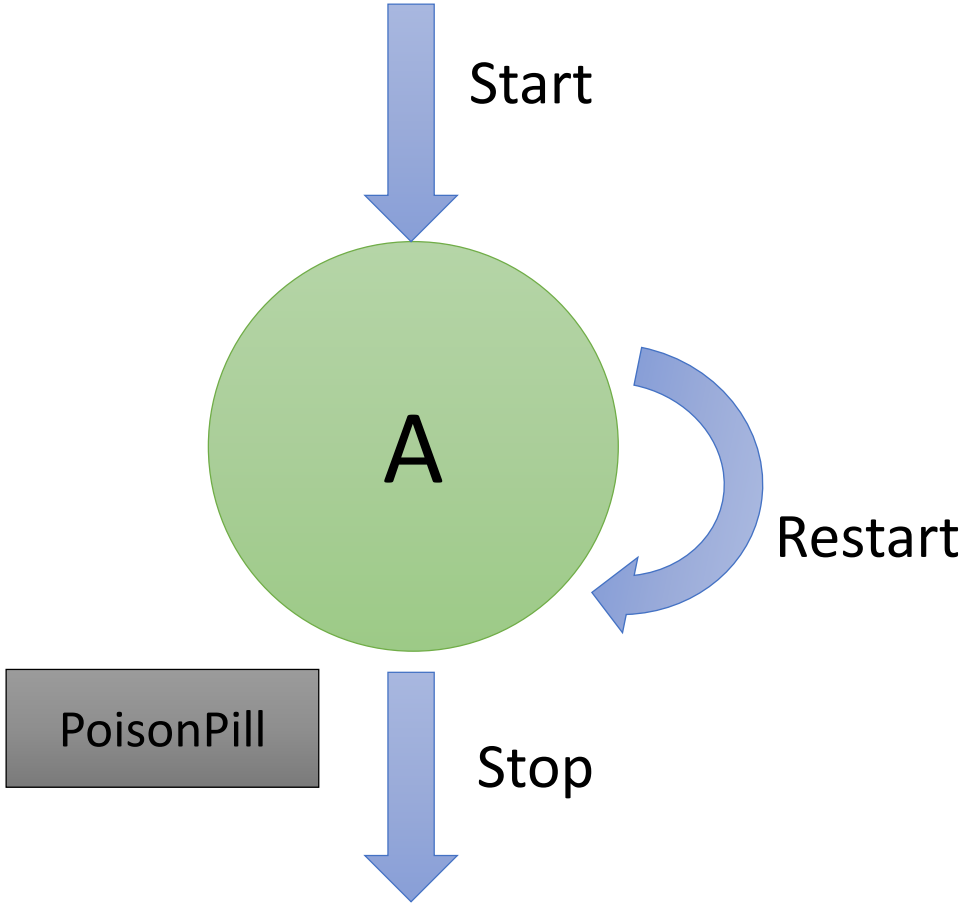


Actor lifecycle

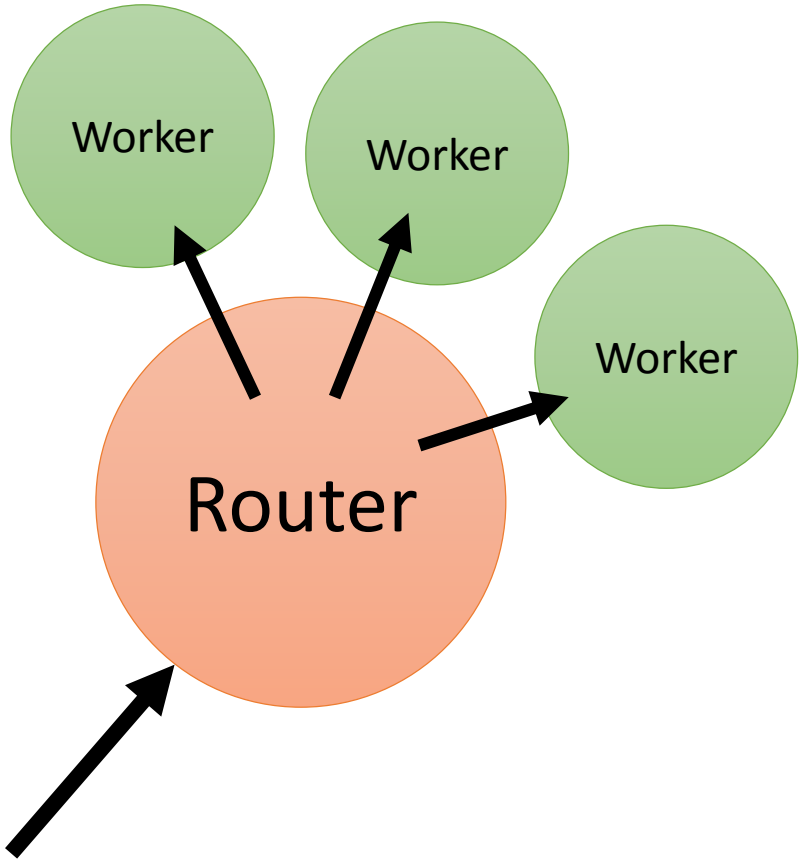
Supervised by actor's parent



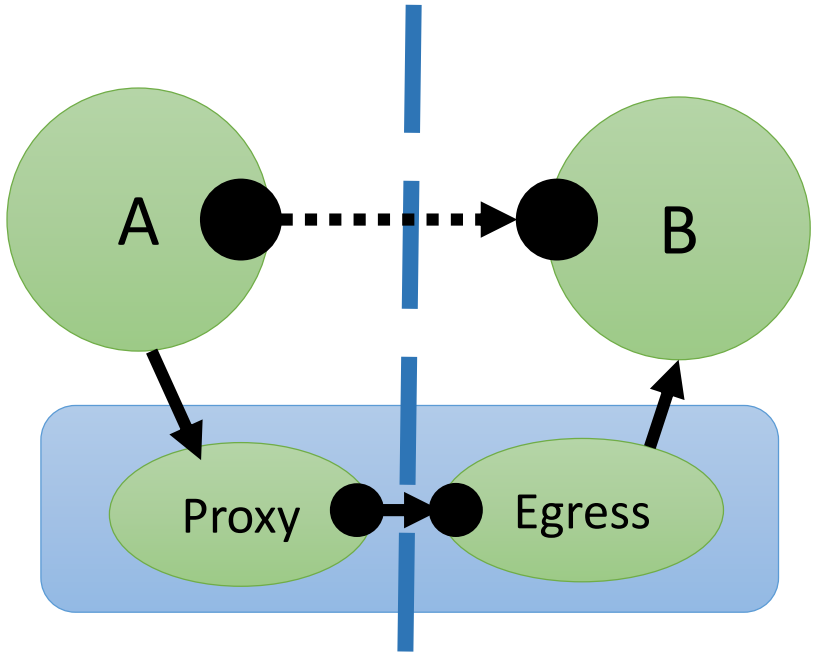
- Stop the child?
- Restart the child?
- Restart all children?
- Escalate (throw to parent)?



Routers



ReliableProxy



Summary

- A framework for asynchronous processing
- Scalability at the core
- No threads
- No blocking
- If you can, delegate it
- Ensure single responsibility

Not covered

- Typed Actors
- Agents
- FSM
- Persistence
- Clustering