Akka 101

...

고재도

+jeado.ko
haibane84@gmail.com

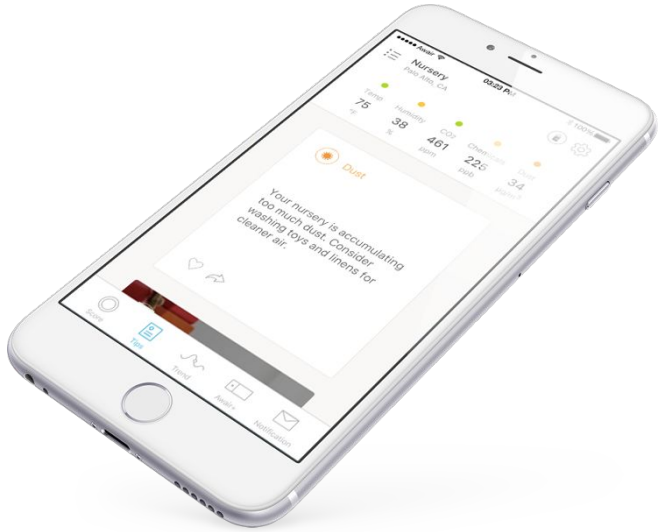
- “Awair” Software Engineer
- “kt” IoT Platform Researcher
- “http://webframeworks.kr” AngularJS Tutorial Contributor
- “Google Developer Expert”
- “시작하세요 AngularJS wikibooks” Author

Smart Sensing

Awair's sensors are designed and tested to accurately identify the five keys factors of air quality. Each sensor is strategically placed to ensure optimal airflow and consistent readings.

https://getawair.com

You can customize the triggers to automatically activate your attached device.



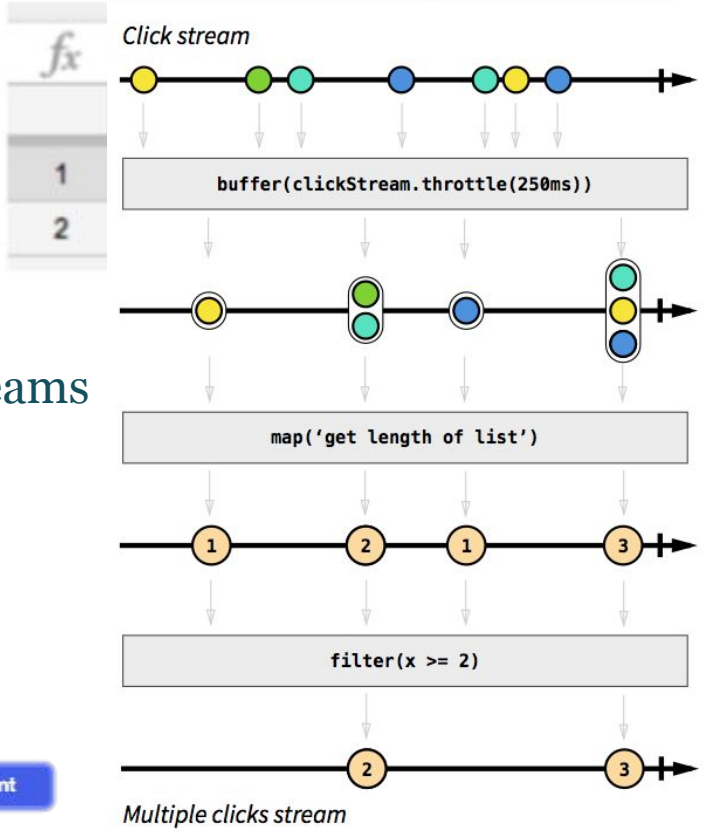
Intro

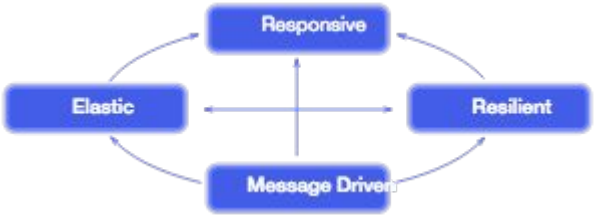
Reactive


??????????????
#TeachersSeries

Reactive

- Reactive as Responsive
  Quickly prompt to user actions
- Reactive Data binding
  Declarative, as opposed to Imperative
  a = b+3
  e.g. Spreadsheets
- Reactive streams as Asynchronous Data Streams
- Reactive as Reactive Manifesto





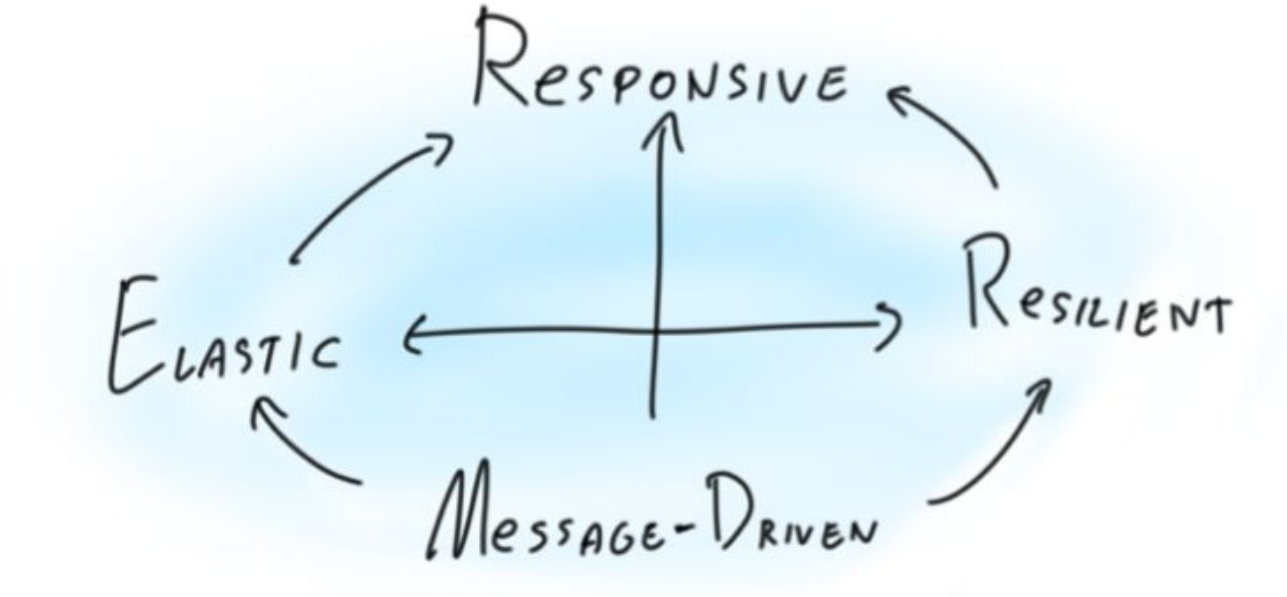
Reactive Manifesto

Responsive -> Low latency

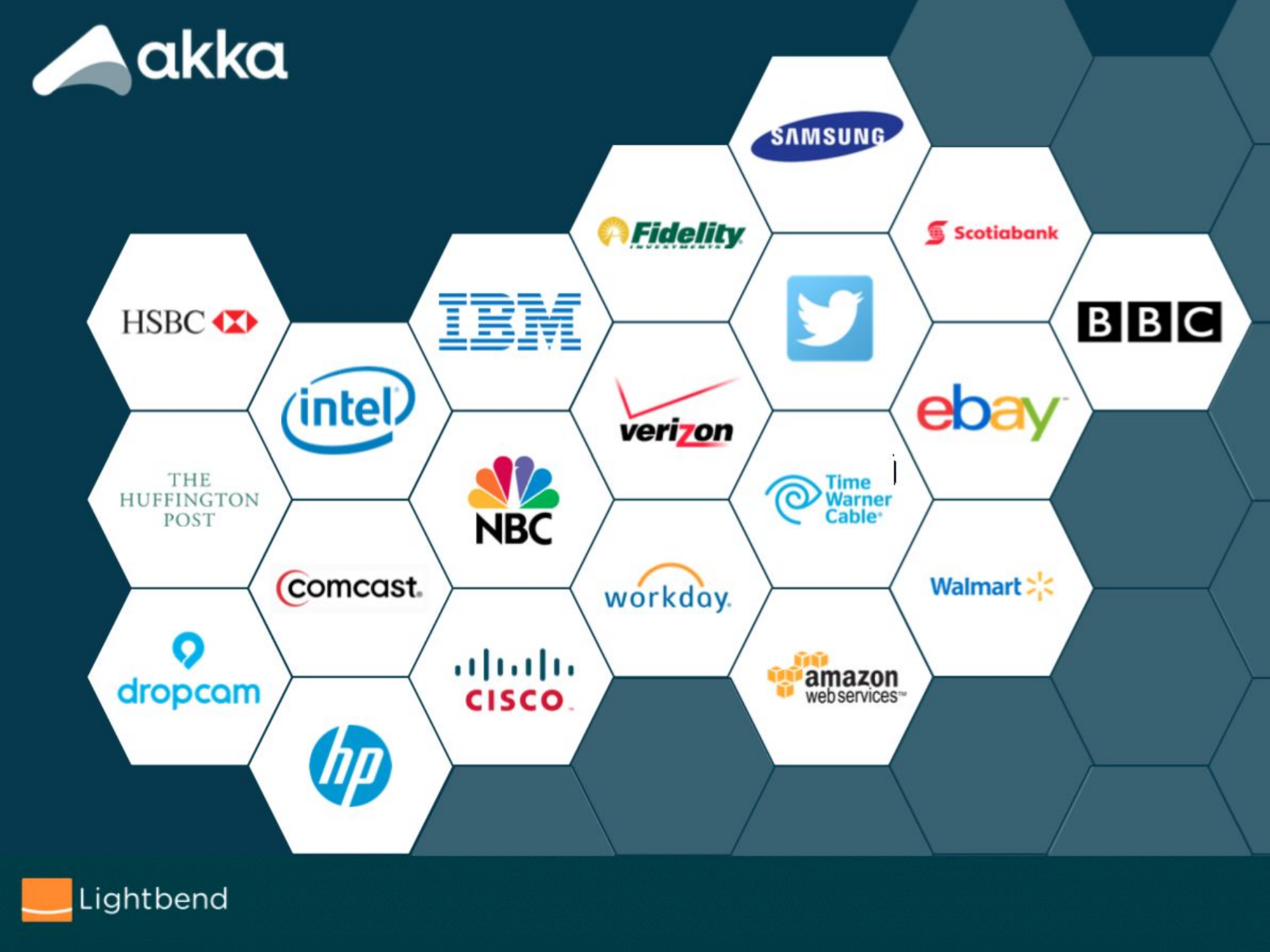
Resilient -> Stay responsive on failure by **Failure Detectors**

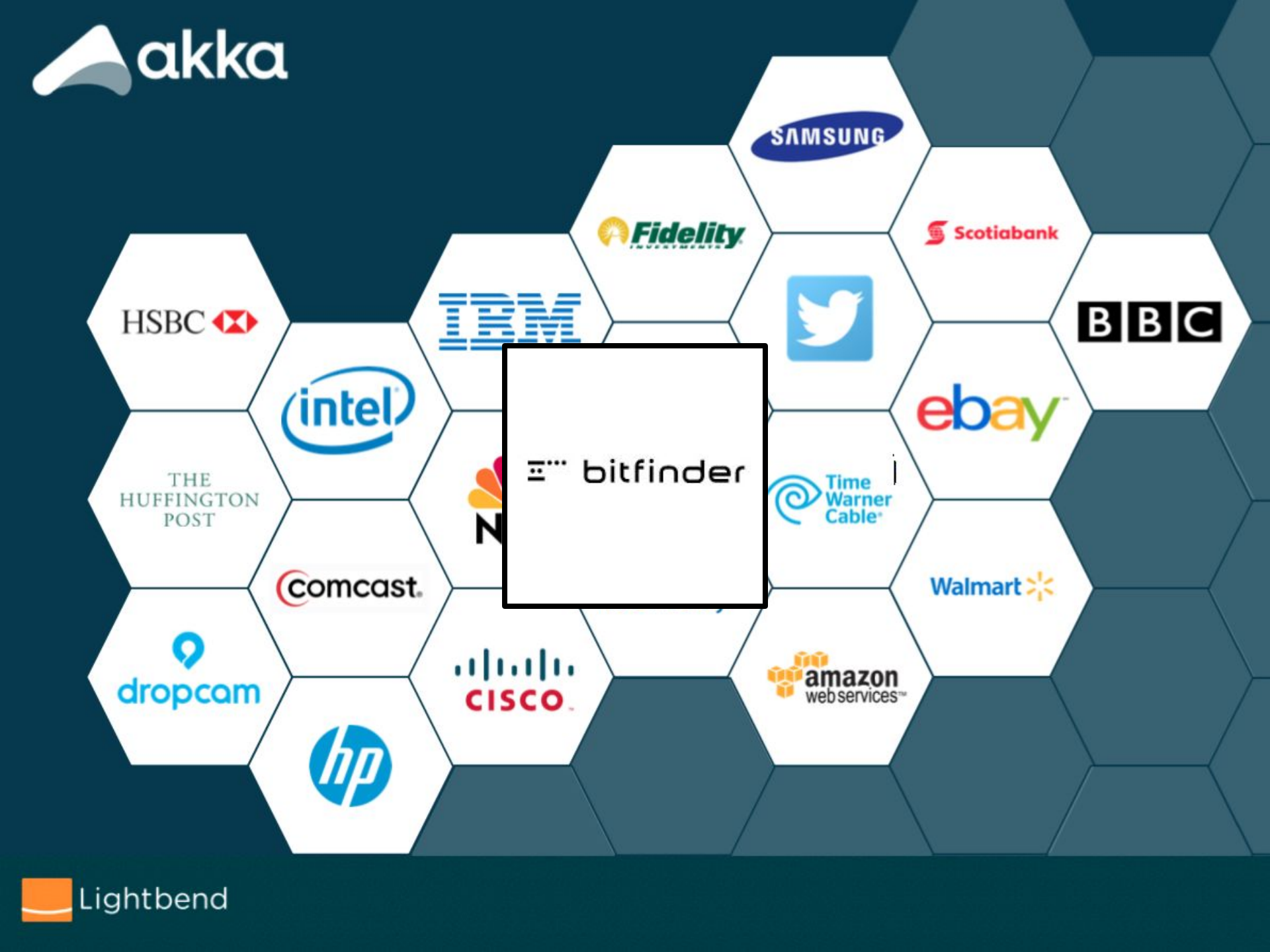
Elastic -> Scale as needed by **Cluster + Sharding**

Message-Driven -> Async messages as only communication between components by **Actors + Streams**




akka


akka
SAMSUNG
Fidelity
Scotiabank
HSBC
IBM
BBC
intel
verizon
ebay
THE HUFFINGTON POST
NBC
Time Warner Cable
comcast
workday
Walmart
dropcam
CISCO
amazon web services
hp
Lightbend


akka
SAMSUNG
Fidelity
Scotiabank
HSBC
IBM
BBC
intel
ebay
bitfinder
THE HUFFINGTON POST
Time Warner Cable
comcast
Walmart
dropcam
CISCO
amazon web services
hp
Lightbend

REACTIVE PLATFORM » FOR DEVELOPMENT » AKKA

LEARN MORE

Actor-based message-driven runtime

Akka is an actor-based message-driven runtime for managing concurrency, elasticity and resilience on the JVM with support for both Java and Scala.

In Akka, the communication between services uses messaging primitives that optimize for CPU utilization, low latency, high throughput and scalability—hardened from years of contributions from the open source community.

Akka embraces the reality of unplanned errors and adopts a pragmatic "Let It Crash" philosophy using supervision and self-healing to ensure impacted components are reset to a stable state and restarted upon failure.

> We're delivering the future of money to over 148 million people worldwide with Akka.
>
> **— PayPal**

Make building powerful concurrent & distributed applications **simple**.

Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient **message-driven** applications on the JVM

Actors – simple & high performance concurrency
Cluster / **Remoting** – location transparency, resilience
Cluster Sharding – and more prepackaged patterns

Streams – back-pressured stream processing

Persistence – Event Sourcing

HTTP – complete, fully async and reactive HTTP Server
Official **Kafka**, **Cassandra**, **DynamoDB integrations**, tons more in the community

Complete **Java** & **Scala APIs** for all features (since day 1)
Typed coming soon…

Actor

Actor Model Frameworks / Languages

- Erlang / Elixer
- Akka (JVM/Akka.net)
- Orleans (.NET)
- CAF (C++)
- Celluloid (Ruby)
- Pulsar (Python)

Actor Model

- 1973, publication by Carl Hewitt, Peter Bishop, and Richard Steiger titled *A Universal Modular Actor Formalism for Artificial Intelligence*
- 1986, Erlang adopted the actor model as the foundation for both concurrent programming and distributed programming
- 2009, Jones Bonèr created the Akka framework as an Erlang-inspired Scala implementation of the actor model.

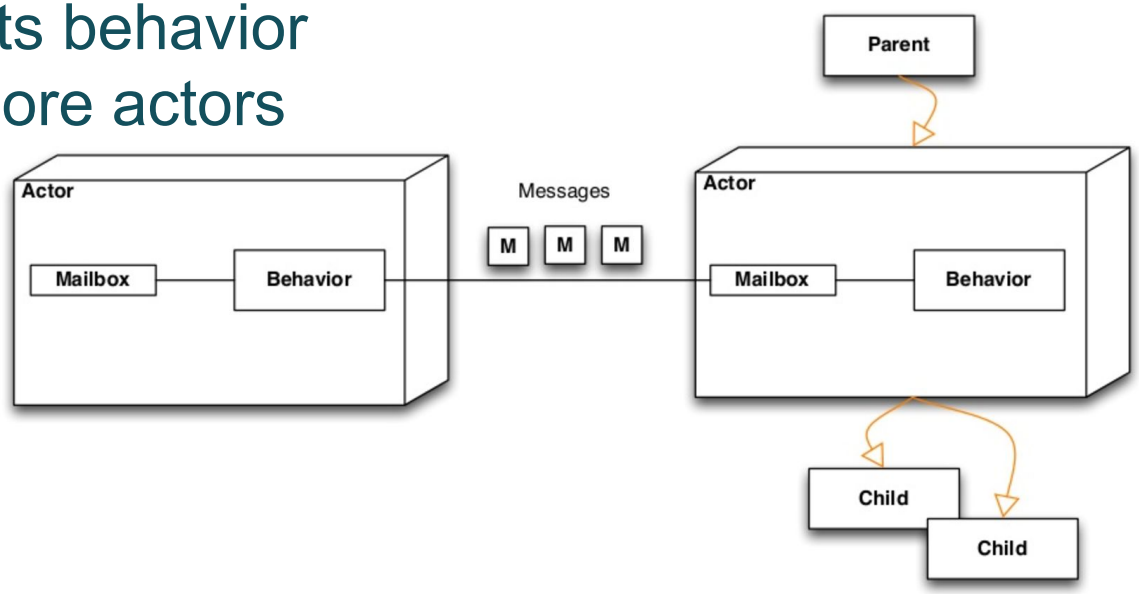
행위자 모델 또는 **액터 모델**(actor model)은 컴퓨터 과학에서 행위자를 병행 연산의 범용적 기본 단위로 취급하는 병행 컴퓨팅의 수학적 모델이다. 행위자가 받는 메시지에 대응하여, 행위자는 자체적인 결정을 하고 더 많은 행위자를 만들며, 더 많은 메시지를 보내고, 다음에 받을 메시지에 대한 응답 행위를 결정할 수 있다. 행위자는 개인 상태를 수정할 수 있지만, 메시지를 통해서만 서로에게 영향을 줄 수 있다. (락의 필요성을 제거함)
[Wikipedia]

Actor Model

A method of concurrency in which the universal primitive is an actor

Actor

- Concurrency primitive
- Persistent
- Encapsulate internal state
- Actors interact exclusively via asynchronous messages
  - Actor sends messages
  - Actor change its state
  - Actor change its behavior
  - Actor create more actors

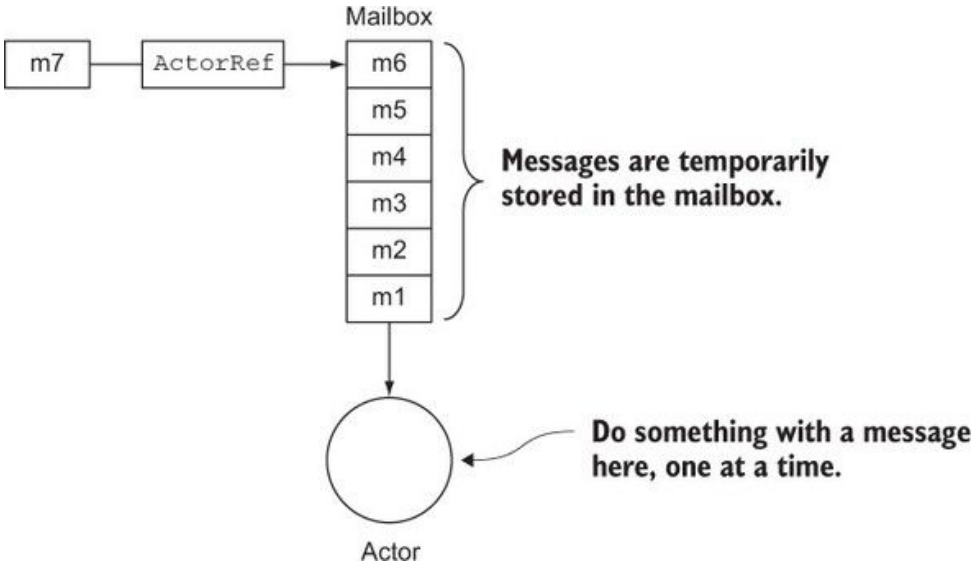


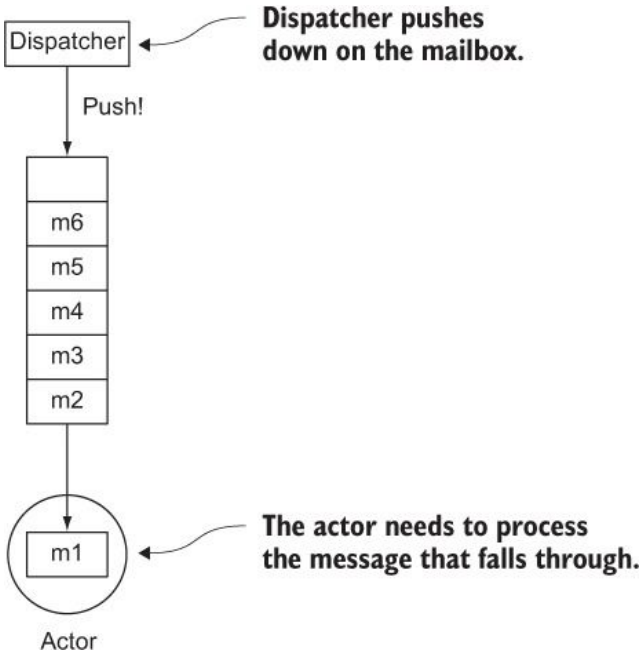
Actor can ...

- Actor sends messages and response
- Actor change its state
- Actor change its behavior
- Actor create more actors
- Actor process exactly one message at a time

Akka Actor

ActorRef, mailbox, actor, dispatcher





이미지 출처: akka in action

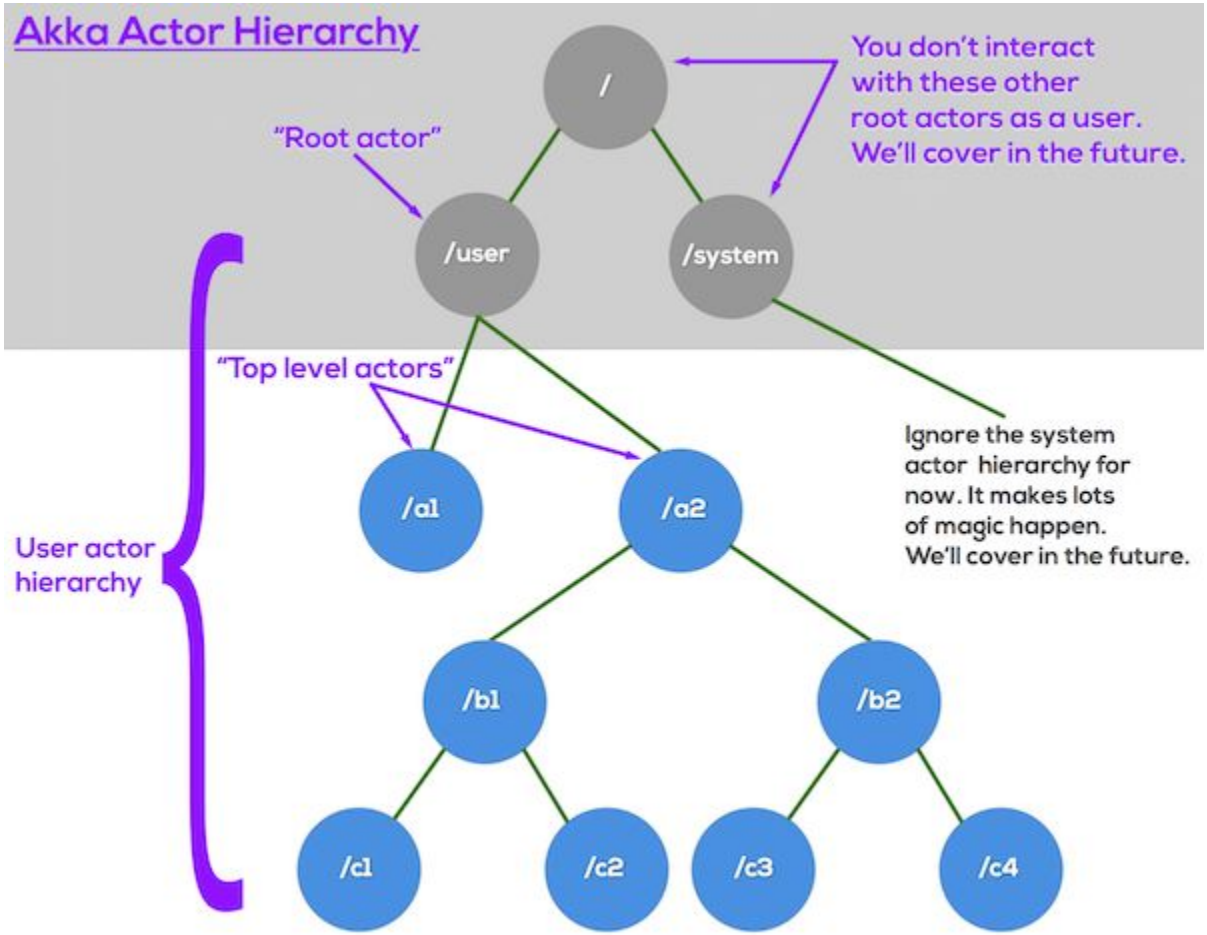
Akka Actor Path

- Identifies an Actor
- May also represent a proxy / forwarder to an Actor
- Contains location and transport information
- Location transparency
  - one path may represent many actors (router pool)
  - one actor may have many addresses (cluster)

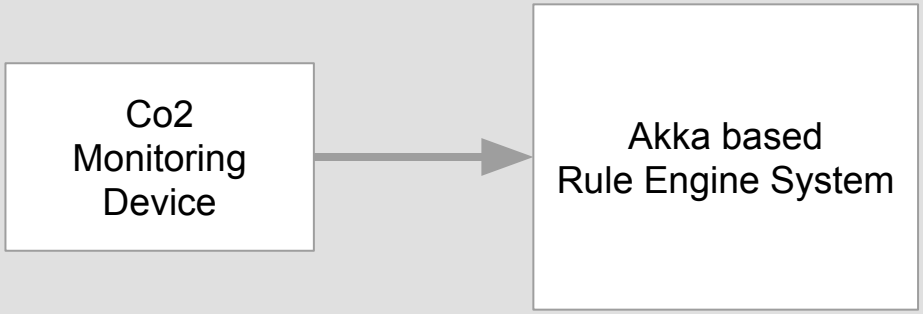
Supervision

The running state of an actor is monitored and managed by another actor

Akka Actor Hierarchy



Demo


Co2
Monitoring
Device
Akka based
Rule Engine System

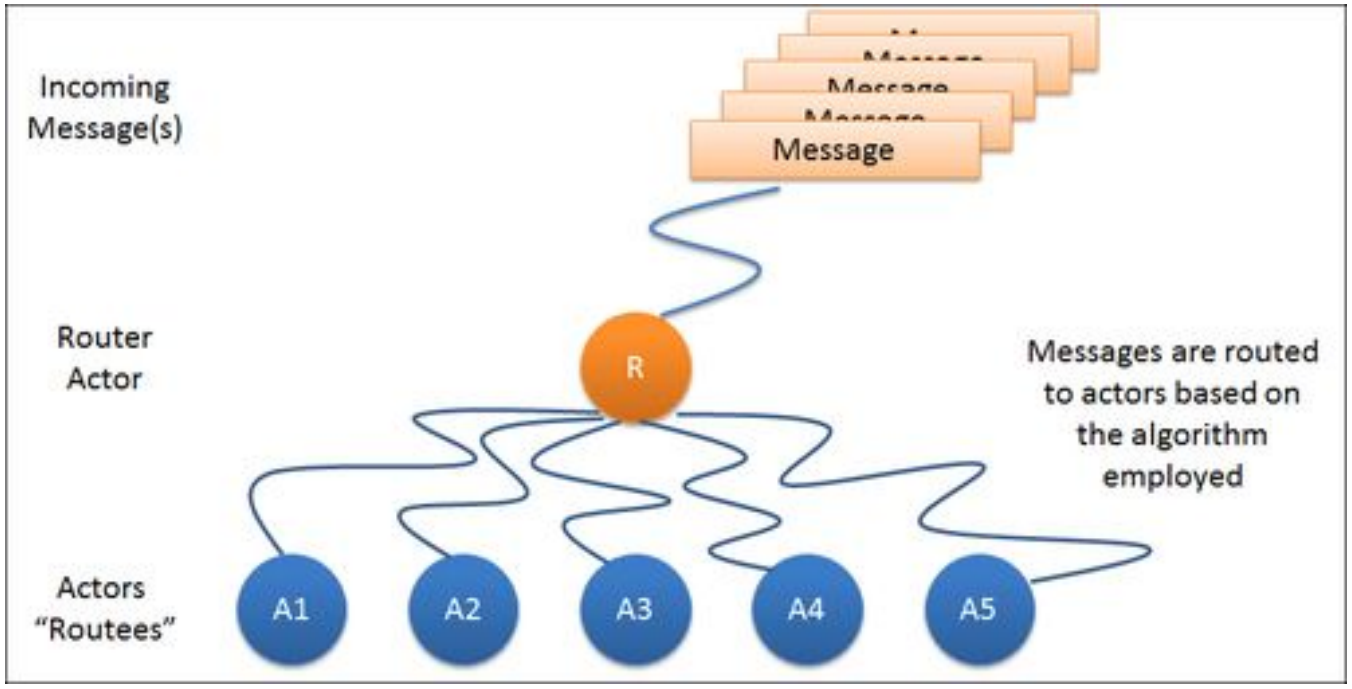
Scale up with router

Router

Routers

- Pool - The router creates routees as child actors and removes them from the router if they terminate.
- Group - The routee actors are created externally to the router and the router sends messages to the specified path using actor selection, without watching for termination.

  * a pool needs the number of routee instances, and a group needs a list of routee paths.



이미지 출처: akka essentials

Routers

```
1.  akka.actor.deployment {
2.   /parent/router1 {
3.     router = round-robin-pool
4.     nr-of-instances = 5
5.   }
6.  }
```

```
1.  val poolRouter: ActorRef =
2.   context.actorOf(FromConfig.props(Props[Worker]), "router1")
```

```
1.  val router2: ActorRef =
2.   context.actorOf(RoundRobinPool(5).props(Props[Worker]), "router2")
```

Routers

```
1.  akka.actor.deployment {
2.   /parent/router3 {
3.     router = round-robin-group
4.     routees.paths = ["/user/workers/w1", "/user/workers/w2",
    "/user/workers/w3"]
5.   }
6.  }
```

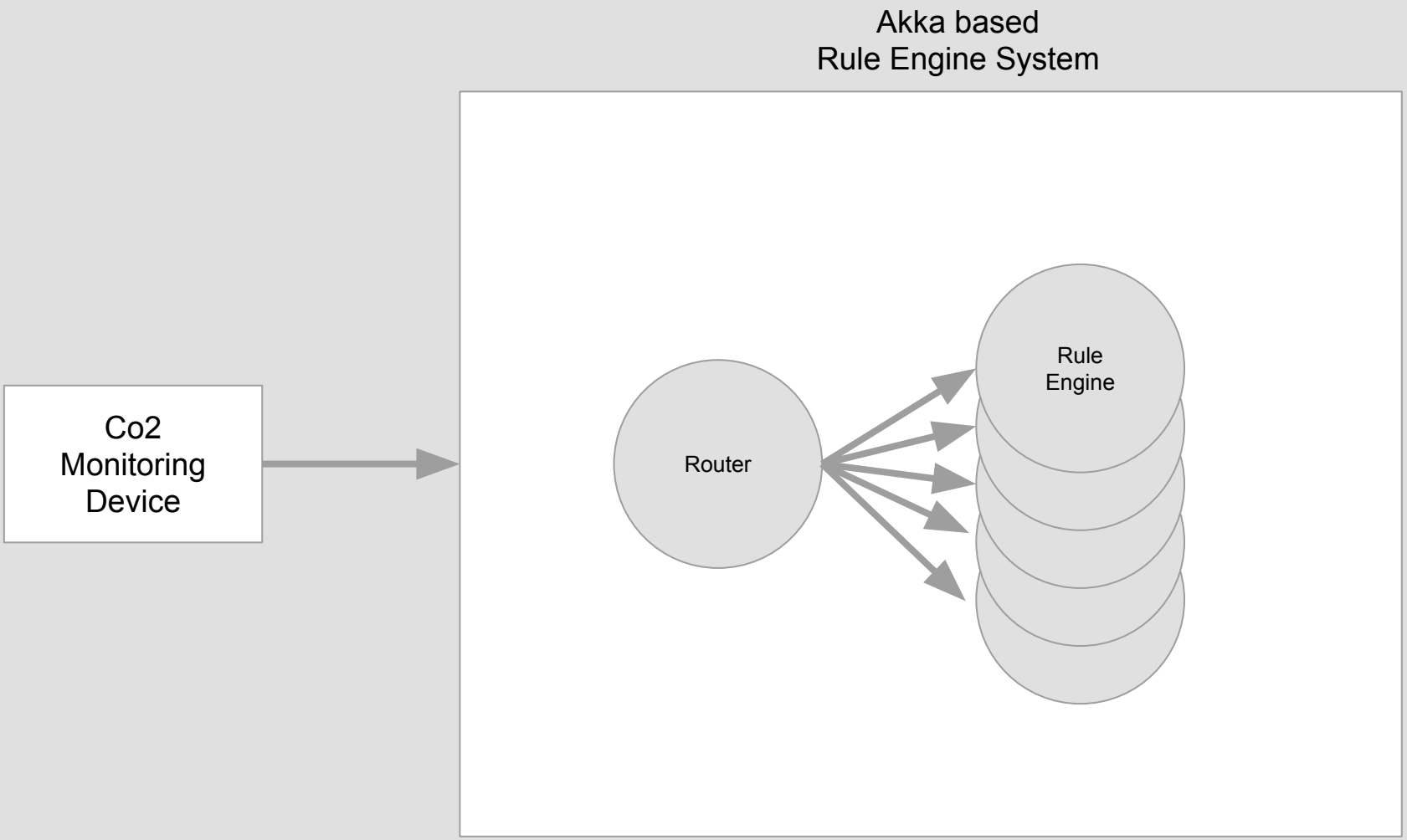
```
1.  val router3: ActorRef =
2.   context.actorOf(FromConfig.props(), "router3")
```

```
1.  val router4: ActorRef =
2.   context.actorOf(RoundRobinGroup(paths).props(), "router4")
```

Routing Logic

- **RoundRobinPool and RoundRobinGroup**
- **RandomPool and RandomGroup**
- **BalancingPool**
- **SmallestMailboxPool**
- **BroadcastPool and BroadcastGroup**
- **ScatterGatherFirstCompletedPool and ScatterGatherFirstCompletedGroup**
- **TailChoppingPool and TailChoppingGroup**
- **ConsistentHashingPool and ConsistentHashingGroup**

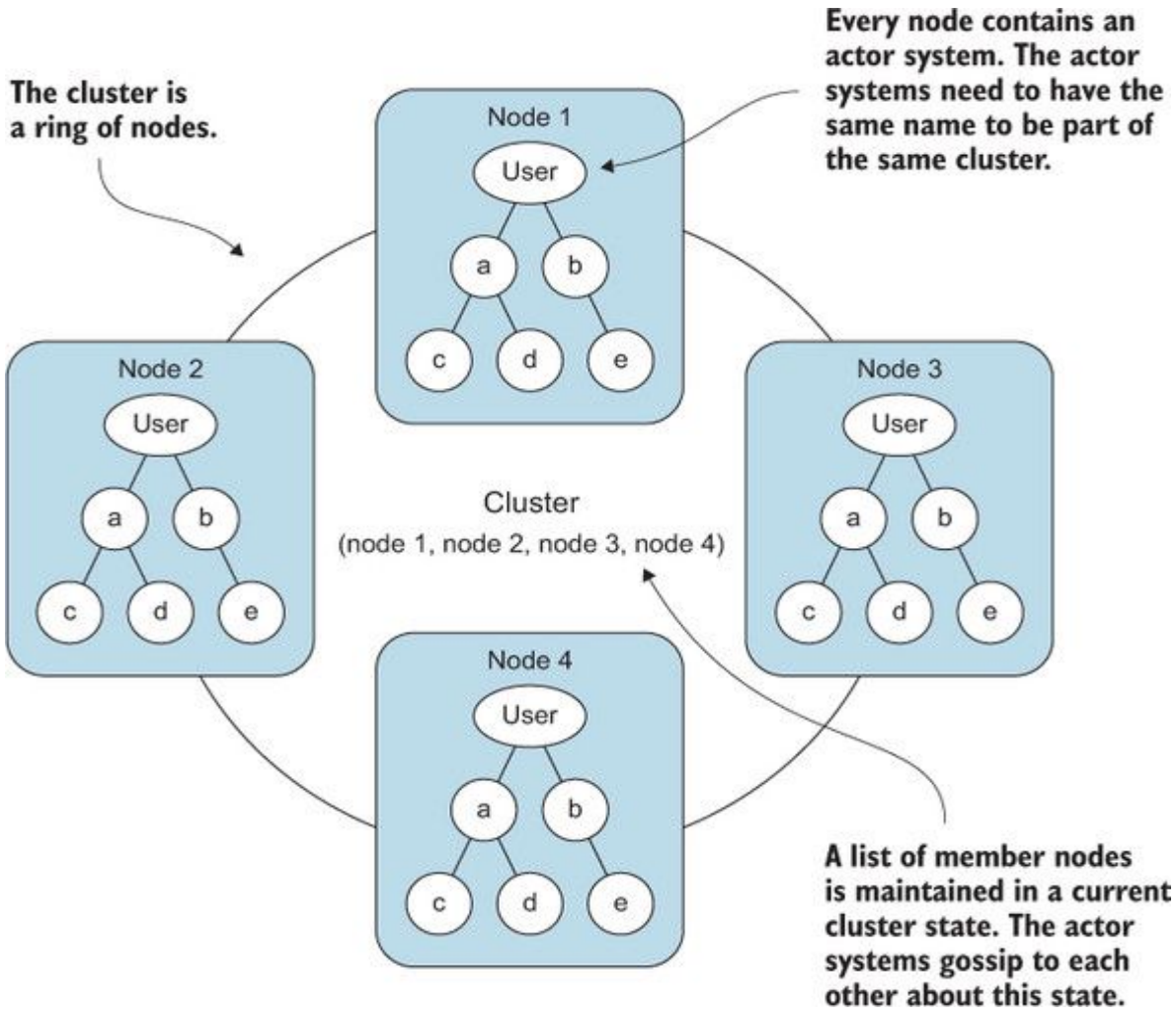
Demo


Akka based
Rule Engine System
Co2
Monitoring
Device
Router
Rule
Engine

Scale out with cluster

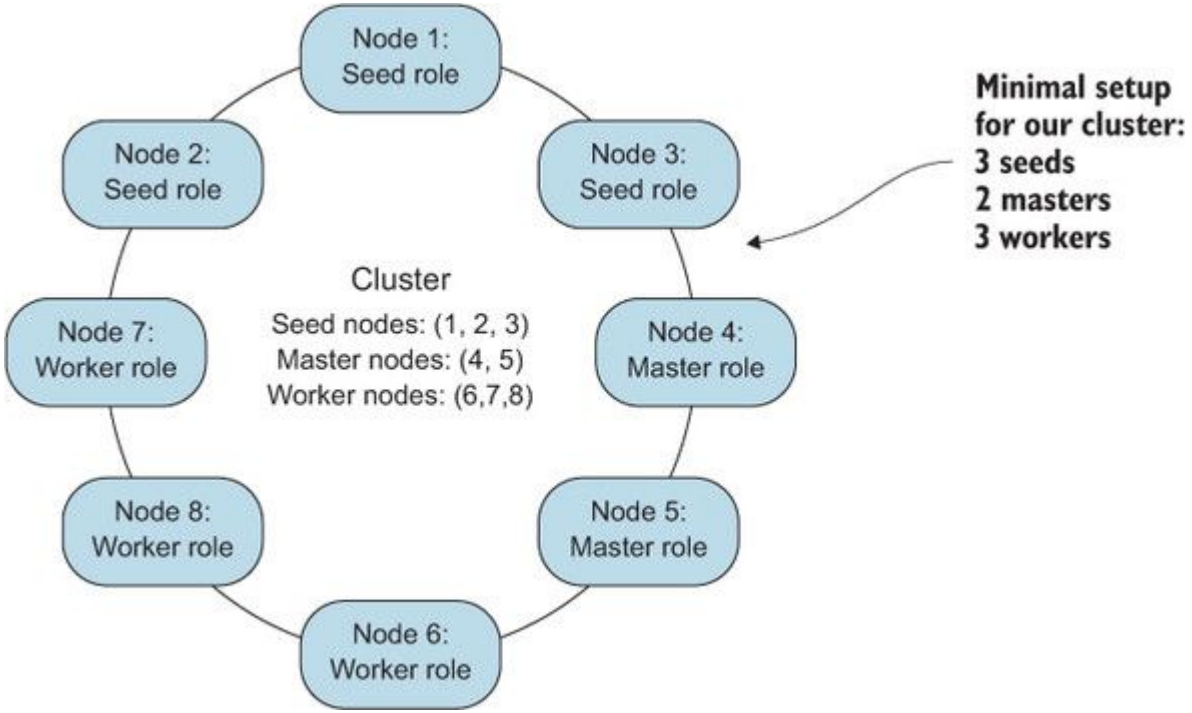
Akka Cluster

Akka Cluster



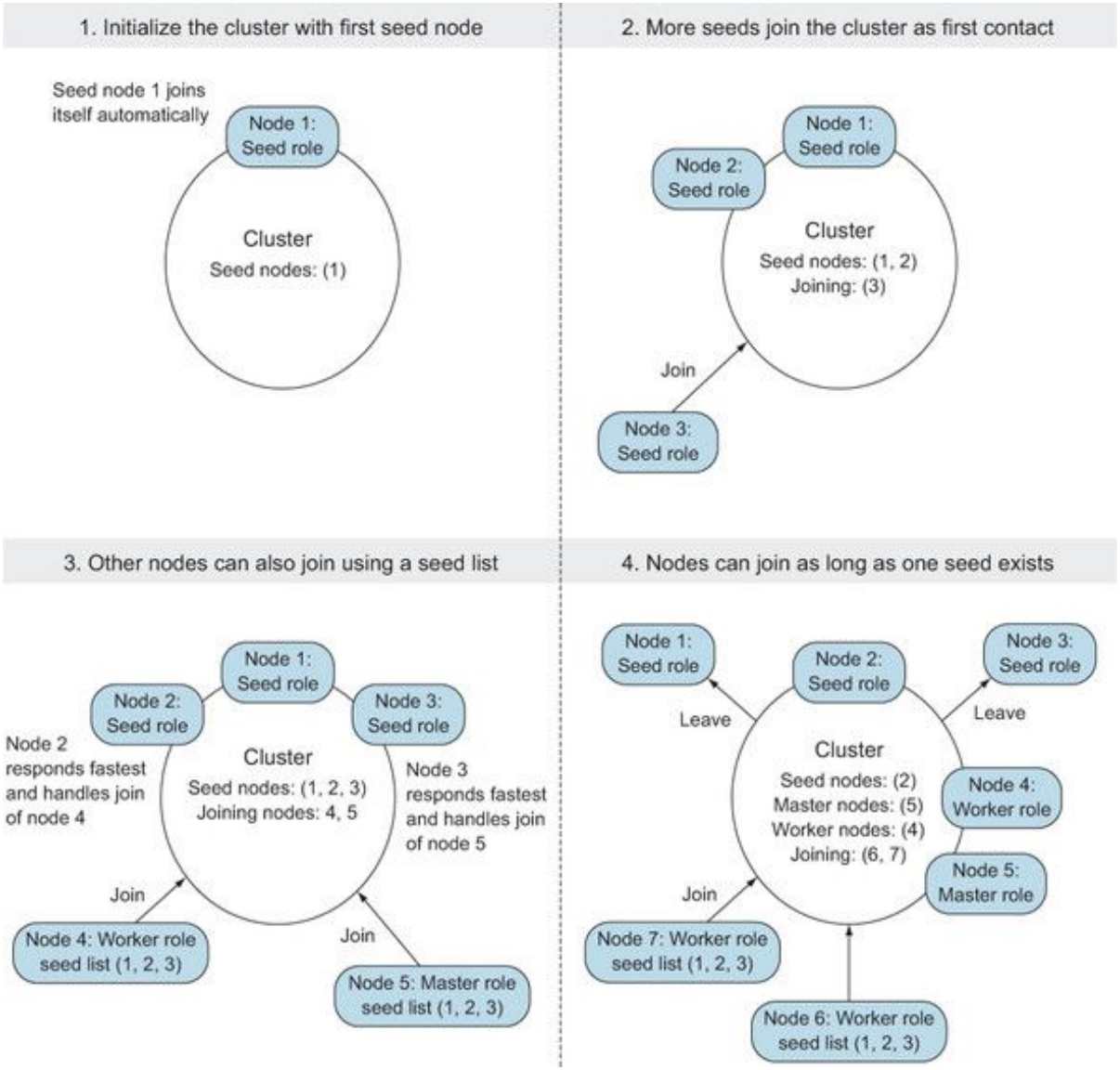
이미지 출처: akka in action

Akka Cluster Membership



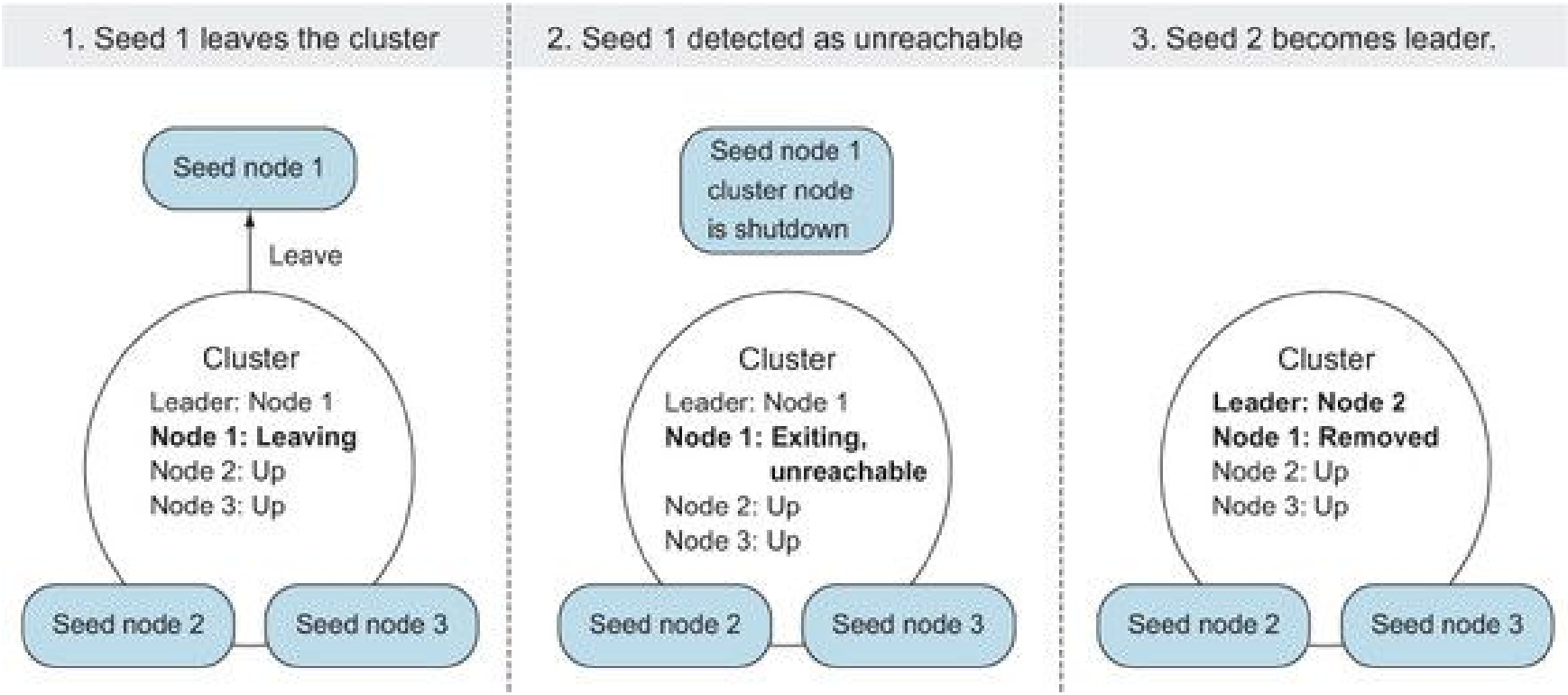
이미지 출처: akka in action

Akka Cluster Joining the cluster



이미지 출처: akka in action

Akka Cluster Leaving and Exiting




```
[Cluster(akka://words)] Cluster Node [akka.tcp://words@127.0.0.1:2552]
- Marking exiting node(s) as UNREACHABLE
[Member(address = akka.tcp://words@127.0.0.1:2551, status = Exiting)].
This is expected and they will be removed.

[Cluster(akka://words)] Cluster Node [akka.tcp://words@127.0.0.1:2552]
- Leader is removing exiting node [akka.tcp://words@127.0.0.1:2551]
```

Leader removes the exiting node

Exiting seed node has the Exiting state

이미지 출처: akka in action

Demo

Anti Use-Cases

- **Non-concurrent system**
- **None-concurrent communication is involved**
- **There is no mutable state**

```
01| actor Foo {
02|   def receive = {
03|     case FooRequest(x) =>
04|       val x = database.runQuery("select * from foo where", x)
05|       val y = redis.get(x.fookey)
06|       sender sendMsg computeResponse(x, y)
07|     }
08|   }
09| }
```

```
01| def fooResult(x) = Future {
04|   val x = database.runQuery("select * from foo where", x)
05|   val y = redis.get(x.fookey)
06|   computeResponse(x, y)
07| }
```

출처 : Introduction to the Actor Model for Concurrent Computation: Tech Talks @ AppNexus

References

Books

- akka in action
- mastering akka
- leaning akka
- akka essentials

Talks

- Introduction to the Actor Model for Concurrent Computation: Tech Talks @ AppNexus
- Introduction to Akka Actors with Java 8
- OpenCredo: Spring Boot Microservices vs Akka Actor Cluster by Lorenzo Nicora

Docs

- akka.io documentation

Thanks