akka
cluster management
& split brain resolution

Niko Will, innoQ
@n1ko_w1ll

about me

- Living in south Germany (Lake of Constance)
- Developer since 2005
- Consultant at innoQ since 2017
- follow me on Twitter: @n1ko_w1ll

agenda

- why a cluster
- akka-cluster
  - setting the ground
  - membership lifecycle (joining / leaving)
- split brain resolution
  - cluster events & cluster state
  - membership lifecycle (unreachable / weakly up / down)

why a cluster

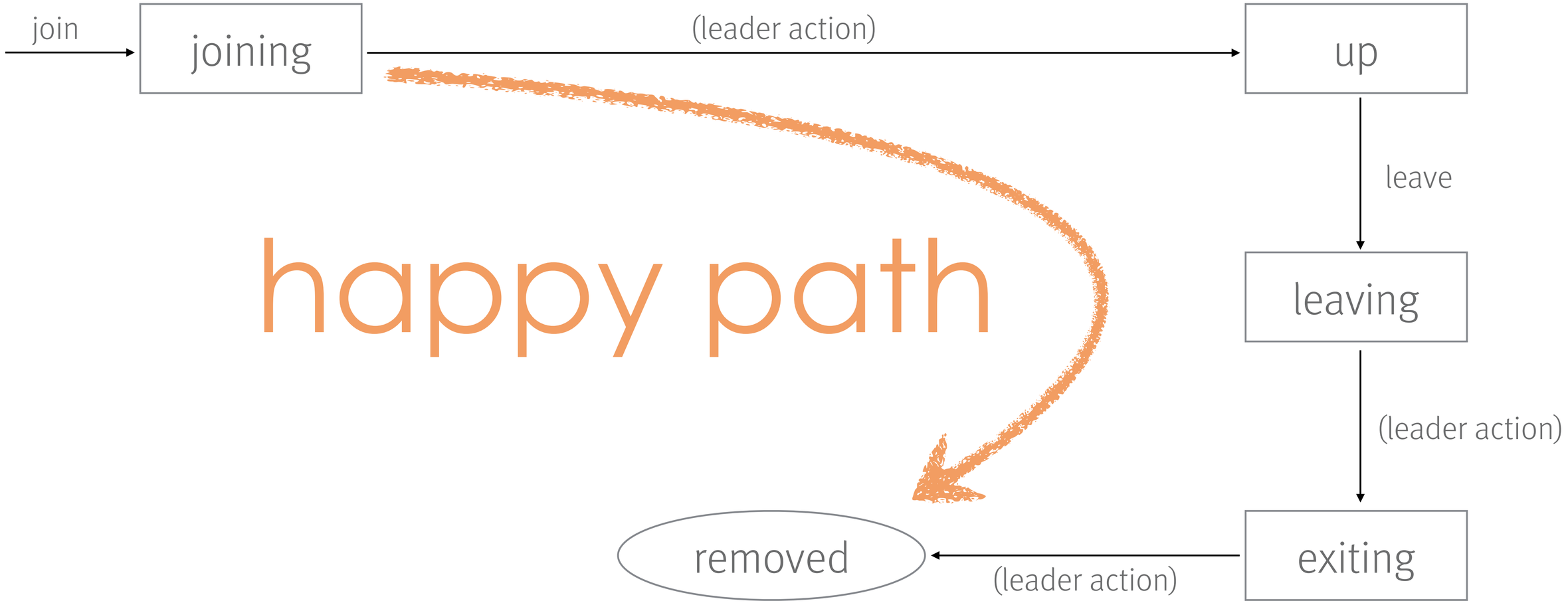
- compute power
- state does not fit in memory
- fault-tolerance

akka-cluster

akka-cluster

- set of member nodes
- membership state is a CRDT
- communicated via Gossip
  - Convergence
  - Failure Detector
  - Leader
  - Seed Nodes

membership lifecycle



Source: http://doc.akka.io/docs/akka/current/common/cluster.html

@n1ko_w1ll

joining

seed nodes

- config
  - akka.cluster.seed-nodes
- manually
  - JMX
  - HTTP API (or command line tool before akka 2.5)
- programmatically

```
Cluster(system).joinSeedNodes(seedNodes)
```

wait for cluster

- leader changes member from "Joining" to „Up"
  - akka.cluster.min-nr-of-members
  - akka.cluster.role.‹role›.min-nr-of-members
- start processing jobs / messages when „Up“

```
Cluster(system).registerOnMemberUp {
  system.actorOf(Props(classOf[FactorialFrontend], upToN, true),
    name = "factorialFrontend")
}
```

leaving

stopping actor system

- leave cluster gracefully
  - manually with JMX or HTTP API (cluster management tool)
  - programmatically with

```
val cluster = Cluster(system)
cluster.leave(cluster.selfAddress)
```

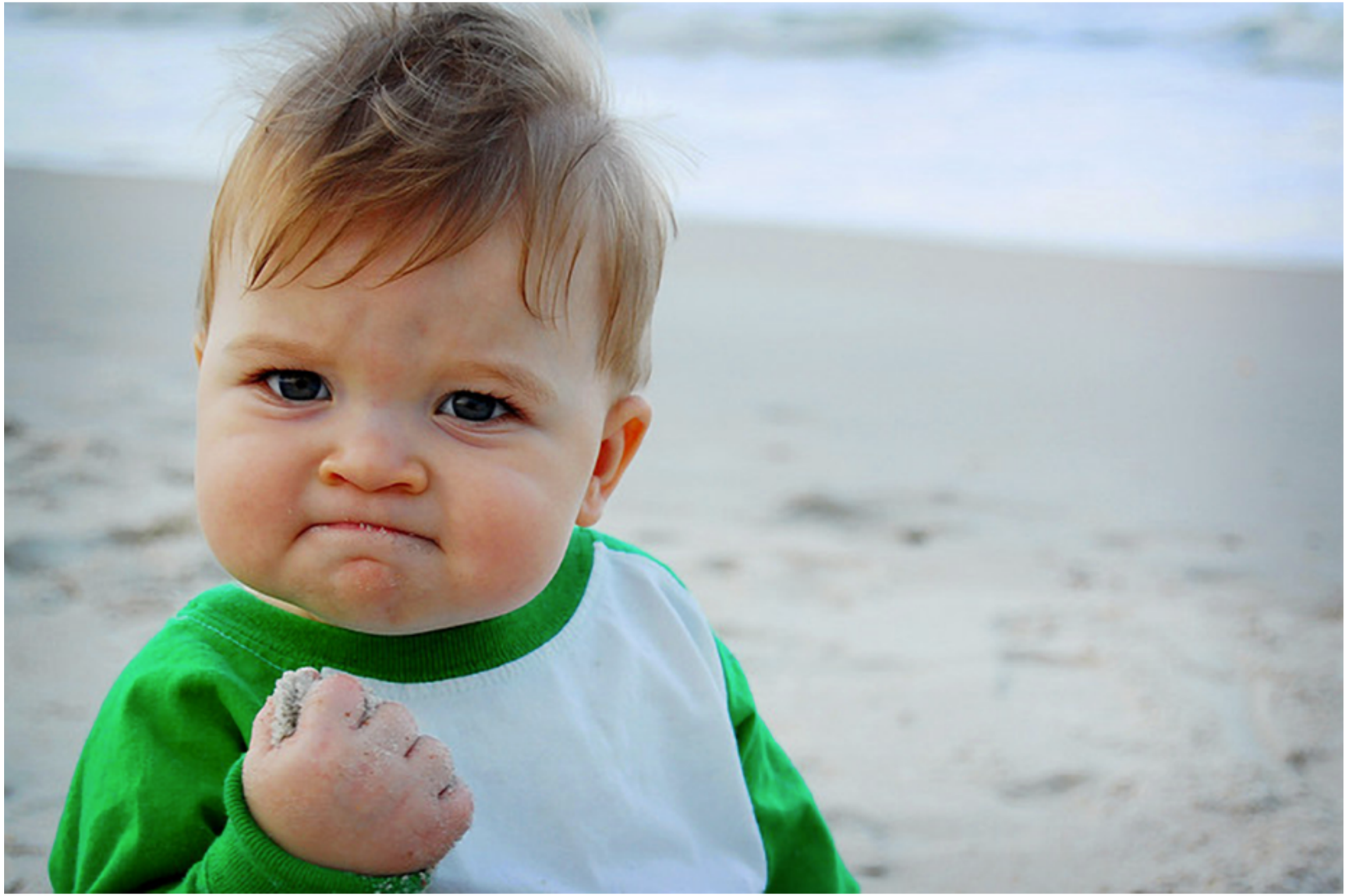
```
Runtime.getRuntime().addShutdownHook(...)
```

- (with SBR, others will down unreachable members)

cleanup

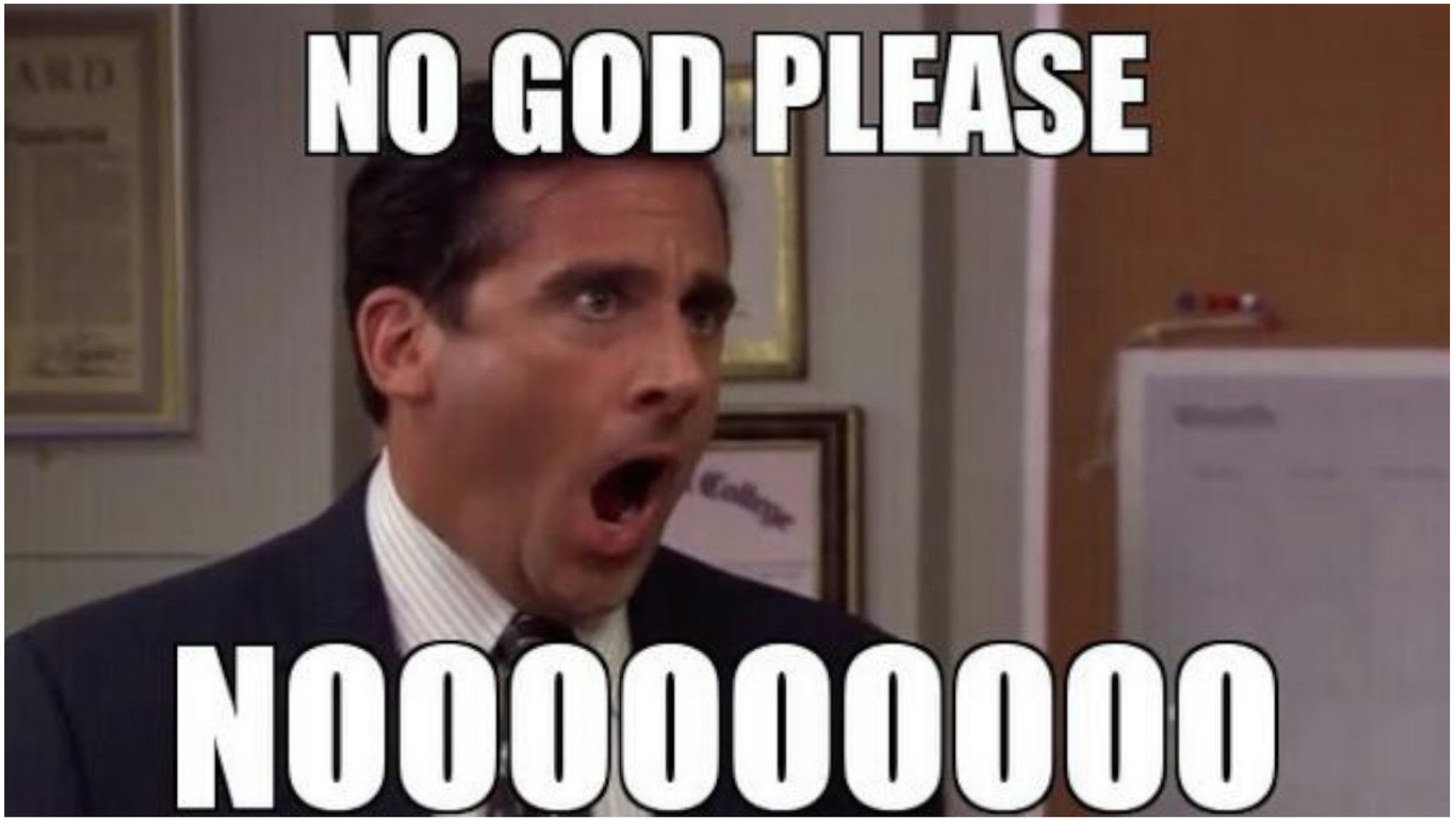
```
Cluster(system).registerOnMemberRemoved {
 system.registerOnTermination(System.exit(0))
 system.terminate()

 new Thread {
  override def run(): Unit = {
   if (Try(Await.ready(system.whenTerminated, 10.seconds)).isFailure)
    System.exit(-1)
  }
 }.start()
}
```

- included in akka 2.5 (Coordinated Shutdown)



...and then things go south

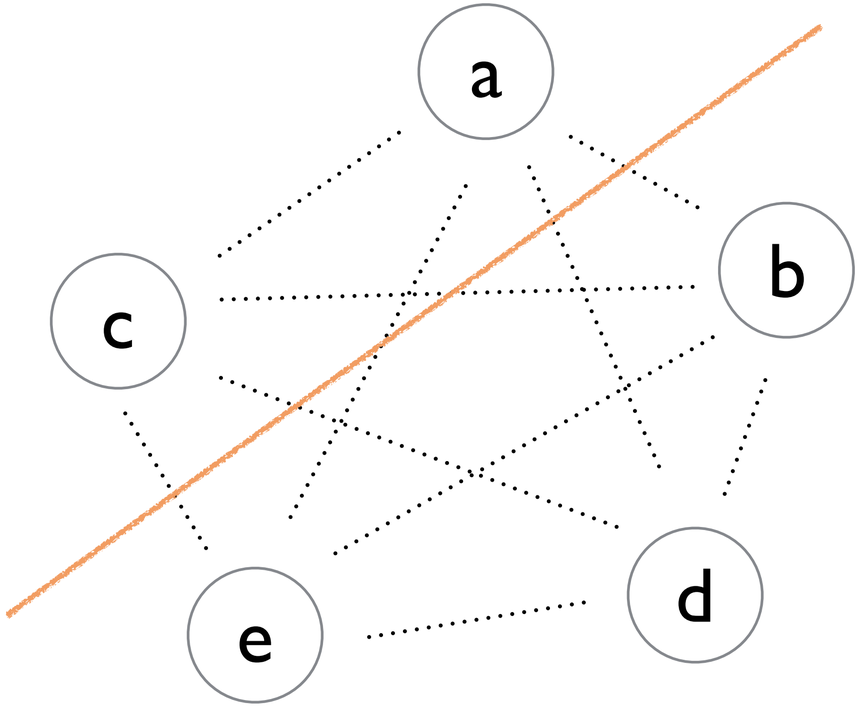
```
 [info] [INFO] [04/05/2017 10:42:22.753] [ClusterSystem-akka.actor.default-
dispatcher-14] [akka.cluster.Cluster(akka://ClusterSystem)]

 Cluster Node [akka.tcp://ClusterSystem@127.0.0.1:2551] -
  Leader can currently not perform its duties,
 reachability status: [
  akka.tcp://ClusterSystem@127.0.0.1:2551 ->
   akka.tcp://ClusterSystem@127.0.0.1:64768: Unreachable [Unreachable] (1)
 ],
 member status: [
  akka.tcp://ClusterSystem@127.0.0.1:2551 Up seen=true,
  akka.tcp://ClusterSystem@127.0.0.1:64768 Up seen=false
 ]
```



@n1ko_w1ll

network partitions

- CAP theorem
  - Consistency
  - Availability
  - **Partition tolerance**



split brain resolution

split brain resolution

- Split Brain Resolver available with Lightbend Subscription
  - Static Quorum
  - Keep Majority
  - Keep Oldest
  - Keep Referee

BYOSPR

bring your own split brain resolver

@n1ko_w1ll

register for cluster events

```
class SimpleClusterListener extends Actor with ActorLogging {
  val cluster = Cluster(context.system)

  override def preStart(): Unit = {
    // subscribe to cluster changes
    cluster.subscribe(self, initialStateMode = InitialStateAsEvents,
      classOf[ClusterDomainEvent])
  }

  override def postStop(): Unit = cluster.unsubscribe(self)

  def receive = ???
}
```

register for cluster events

```
def receive = {
  case state: CurrentClusterState =>
    log.info("Cluster state is: {}", state)
  case MemberUp(member) =>
    log.info("Member is up: {}", member.address)
  case MemberWeaklyUp(member) =>
    log.info("Member is weakly up: {}", member.address)
  case UnreachableMember(member) =>
    log.info("Member detected as unreachable: {}", member)
  case MemberRemoved(member, prevStatus) =>
    log.info("Member is removed: {} after {}“, member.address, prevStatus)
  case _: ClusterDomainEvent => // ignore
}
```

current cluster state

```
Cluster(system).state
```

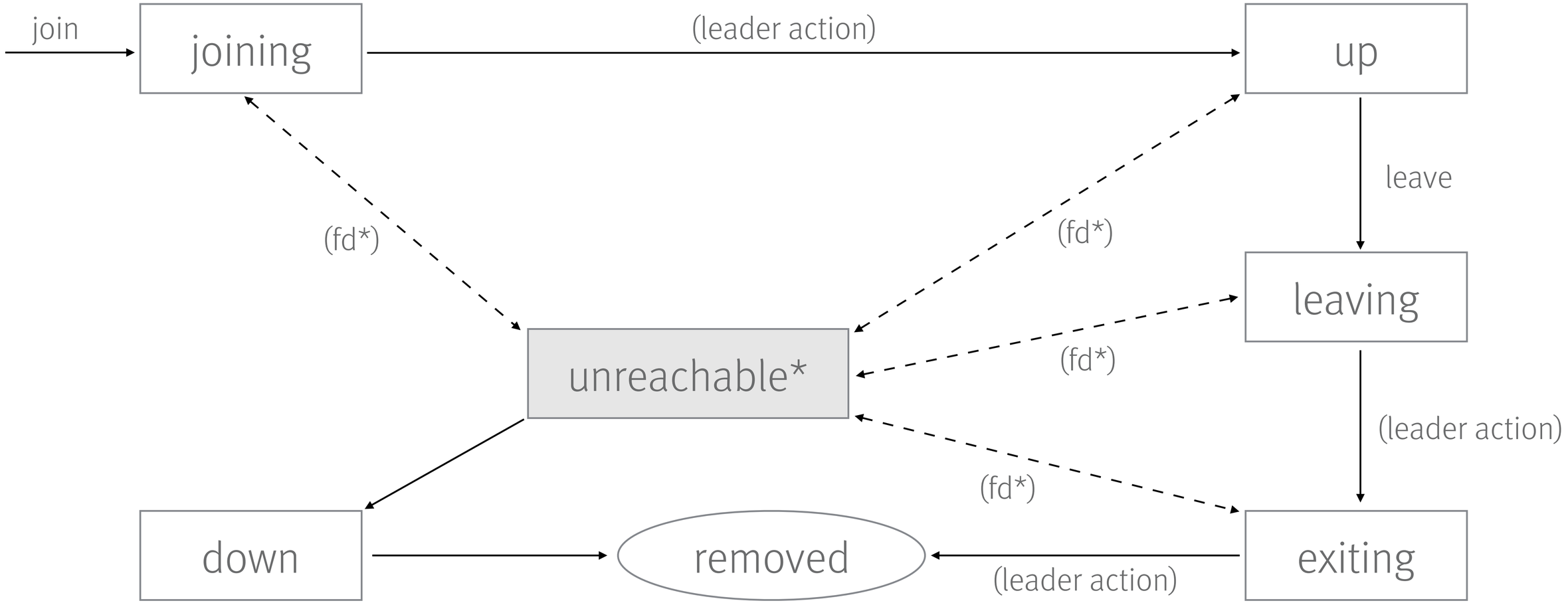
```
final case class CurrentClusterState(
  members:       immutable.SortedSet[Member]  = immutable.SortedSet.empty,
  unreachable:   Set[Member]                  = Set.empty,
  seenBy:        Set[Address]                 = Set.empty,
  leader:        Option[Address]              = None,
  roleLeaderMap: Map[String, Option[Address]] = Map.empty) {
  ...
}
```

health indicator

- expose cluster health state
  - as addition to akka cluster HTTP management (since version 2.5)
  - helps monitoring cluster
  - healthy if no unreachable members

```
Cluster(system).state.unreachable.isEmpty
```

membership lifecycle



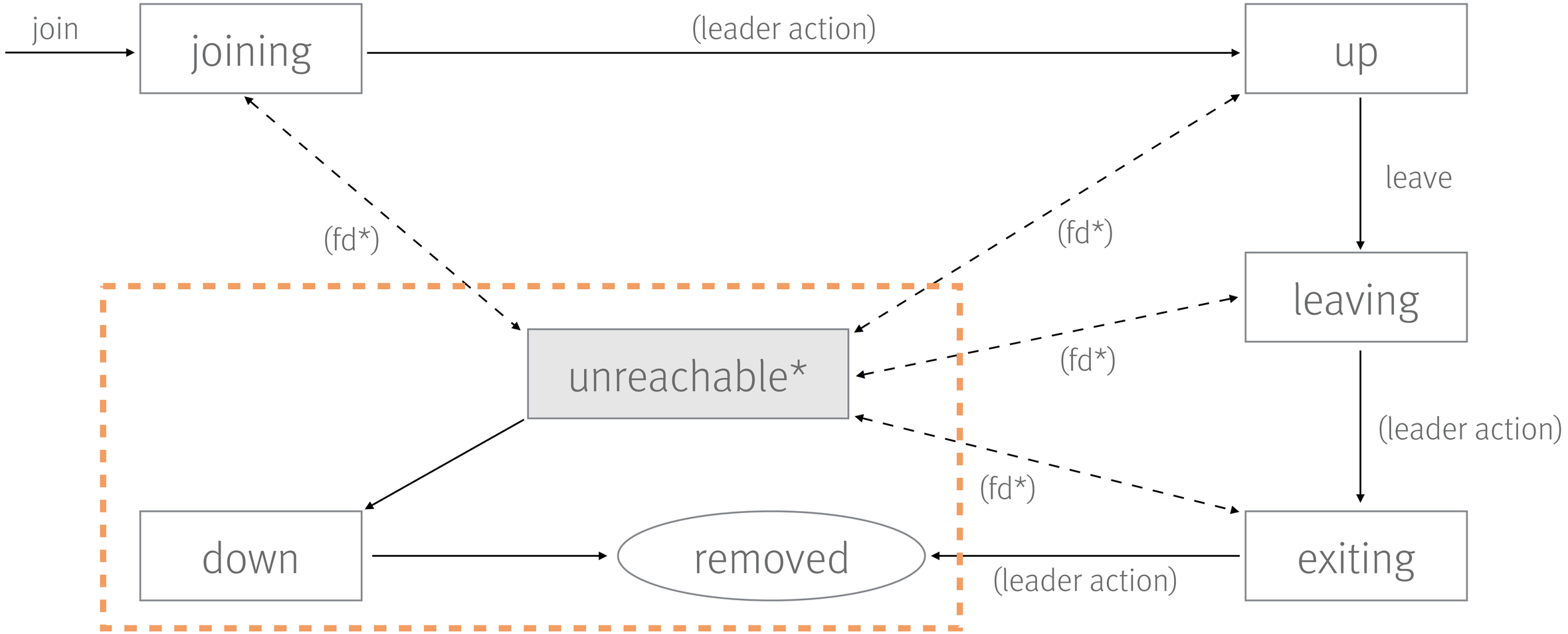
Source: http://doc.akka.io/docs/akka/current/common/cluster.html

unreachable members

- detected by akka cluster failure detector
  - akka.cluster.failure-detector
- „leader can not perform its duties“
  - no new members can join
  - does not influence running members
- unreachable members are still part of the cluster
  - their responsibilities will not failover (e.g. singletons or shards)

downing

membership lifecycle



Source: http://doc.akka.io/docs/akka/current/common/cluster.html

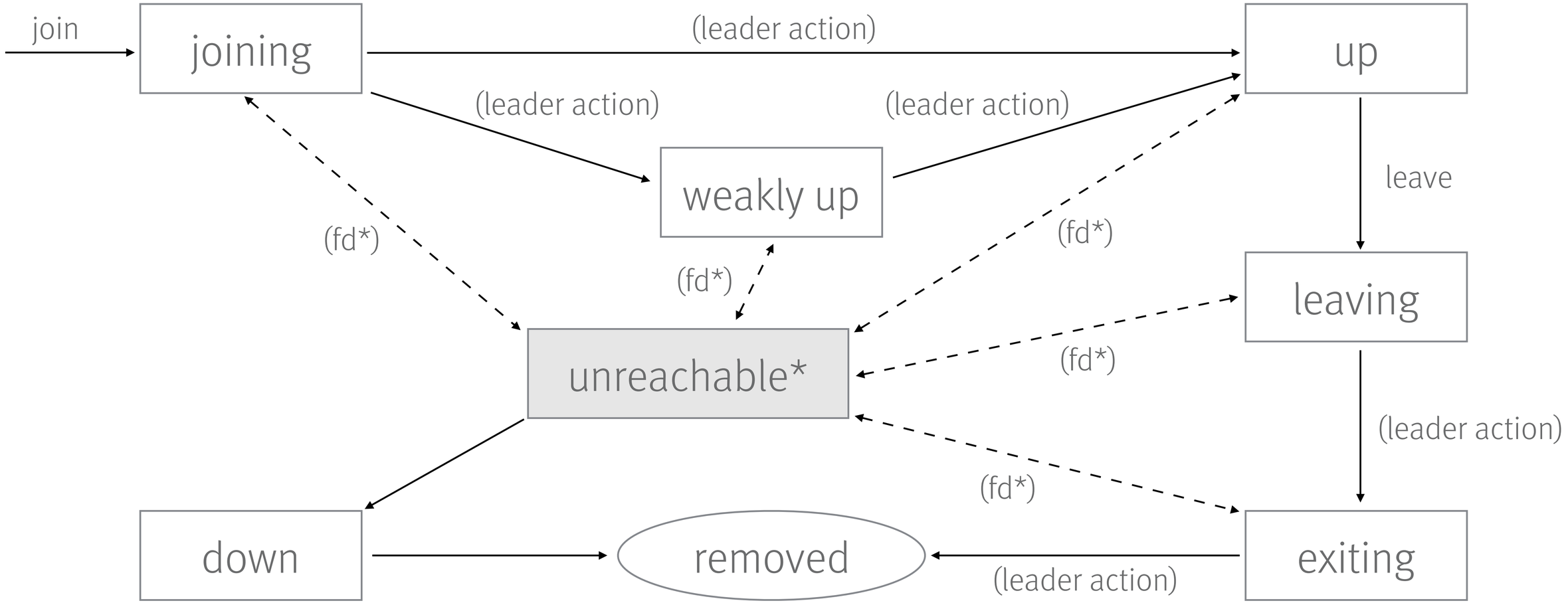
downing

- will remove member from cluster
  - cluster will take over their responsibilities
- auto-downing (development only)
  - mark all unreachable as DOWN
  - network partition will lead to several clusters
- every member can mark itself and others as DOWN

```
Cluster(system).down(nodeAddress)
```

weakly up

membership lifecycle



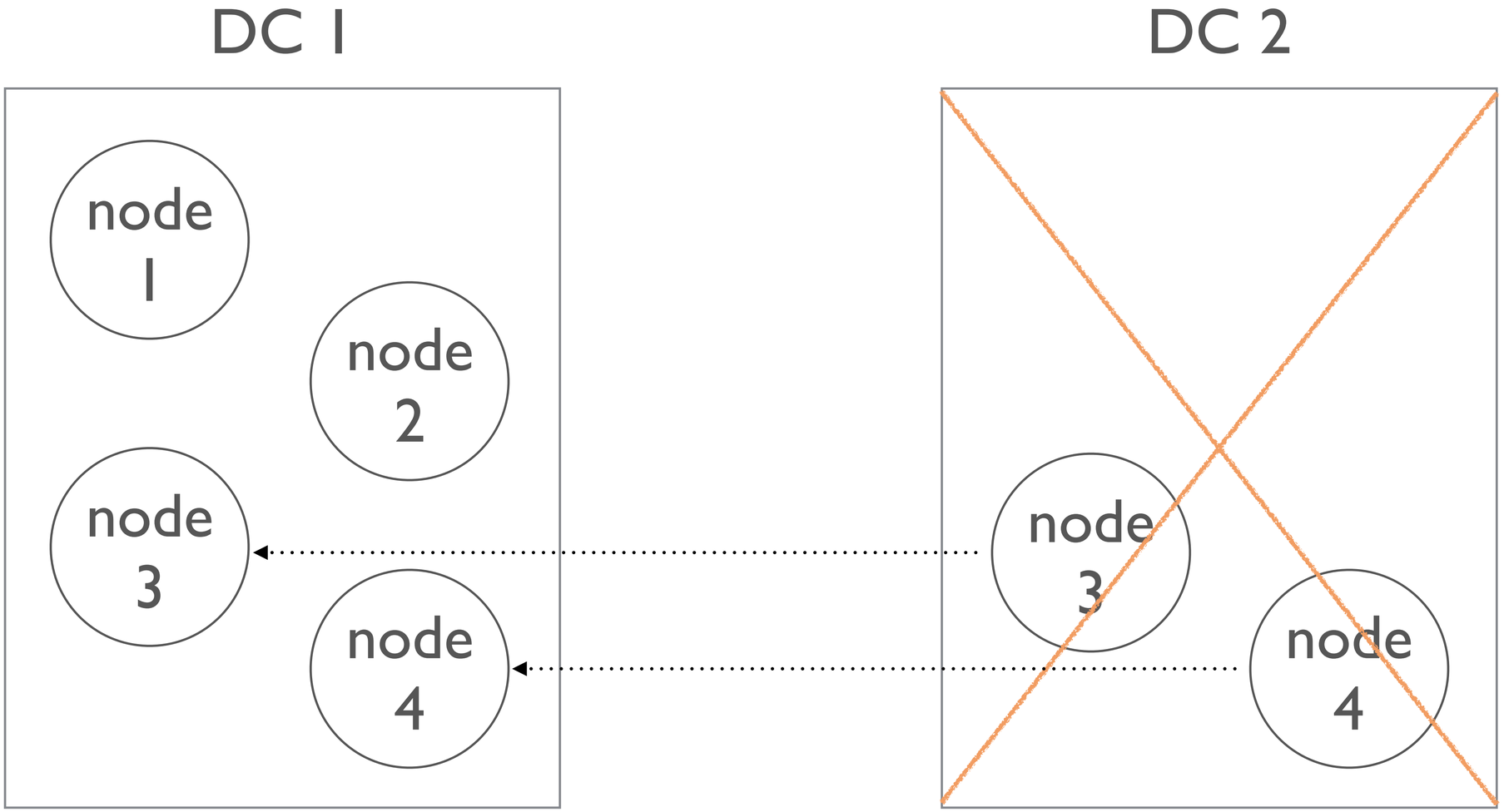
Source: http://doc.akka.io/docs/akka/current/common/cluster.html

time for our own SBR

split brain resolution

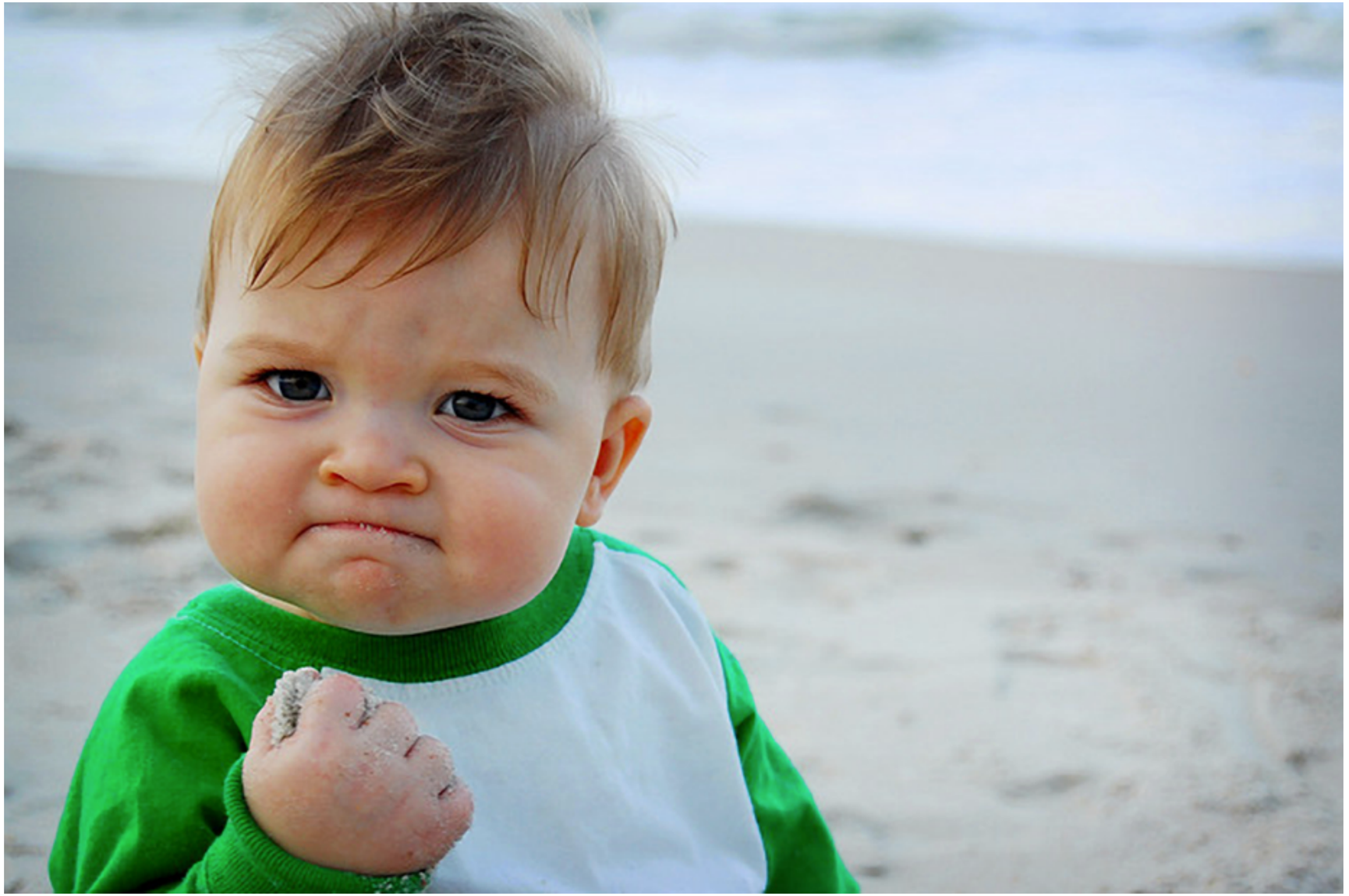
- write a custom `DowningProvider`
  - `akka.cluster.downing-provider-class`
  - `akka.cluster.auto-down-unreachable-after`
- custom strategies using cluster state
  - reachable vs. unreachable members
- member state `WEAKLY_UP`
  - is member known?
  - is known member reachable?

custom SBR



```
def scheduleMajorityCheck() = {
  check.foreach(_.cancel())
  check = Some(scheduler.scheduleOnce(7 seconds, self, CheckForMajority))
}
def receive = {
   case CheckForMajority if cluster.state.unreachable.isEmpty => check = None
   case CheckForMajority =>
     check = None
     val unreachable = cluster.state.unreachable
     val reachable = cluster.state.members.diff(unreachable)
     if (reachable.size > unreachable.size)
       unreachable.map(_.address).foreach(cluster.down)
     else if (reachable.size < unreachable.size)
       reachable.map(_.address).foreach(cluster.down)
```

```
    case MemberUp(member) =>
      knownAddresses += getHostname(member) -> member
      scheduleMajorityCheck()
    case MemberWeaklyUp(member) =>
      knownAddresses.get(getHostname(member))
        .filter(cluster.state.unreachable.contains)
        .map(_.address).foreach(cluster.down)
      scheduleMajorityCheck()
    case MemberRemoved(member, _) =>
      knownAddresses -= getHostname(member)
      scheduleMajorityCheck()
    case UnreachableMember(member) =>
      scheduleMajorityCheck()
}
```



Thank you.

Questions?

Comments

@n1ko_w1ll

Niko Will

niko.will@innoq.com

innoQ

www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
Phone: +49 2173 3366-0

Kreuzstraße 16
80331 München
Germany
Phone: +49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116