Understanding Akka Cluster through Raspberry-Pi visualisation

Workshop overview

- Brief intro to Clustering & Akka Clustering
- Description of the Raspberry Pi-based Cluster hardware
- Building & Running the "*Exercises*"
- ***Your experiments***

Workshop prerequisites

- Have the following installed on your laptop:
  - JDK 8
  - sbt
  - Docker
  - A (multi-window) terminal (such as iTerm2 on MacOS)
  - Clone the Pi-Akka-Cluster repo

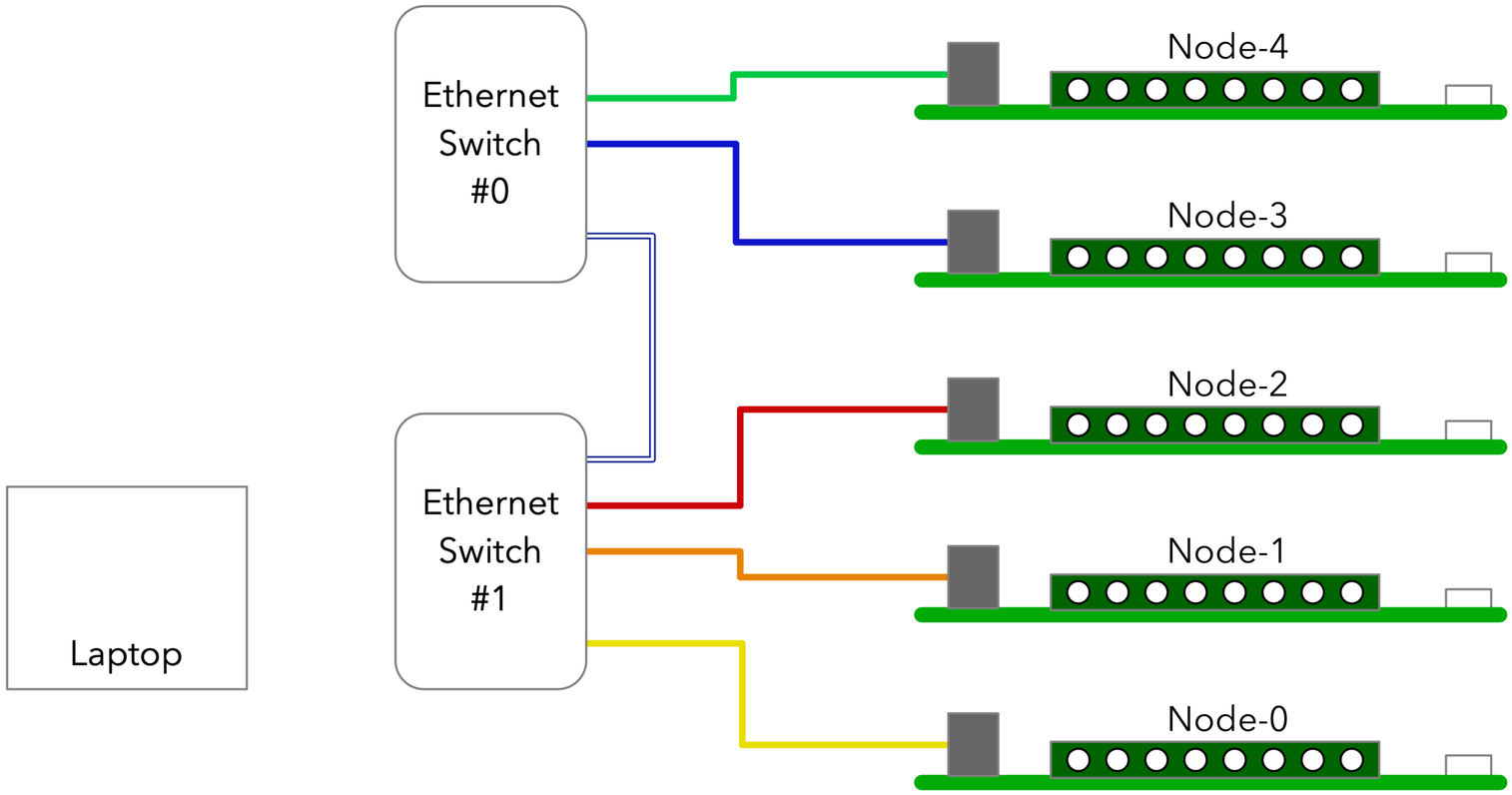
    ```
    git clone git@github.com:lightbend/Pi-Akka-Cluster.git
    cd Pi-Akka-Cluster
    ```

  - IntelliJ IDEA with Scala plugin
- Ability to connect you laptop to a Wifi network
- Set-up key-pair to allow for password-less login on the Pi's

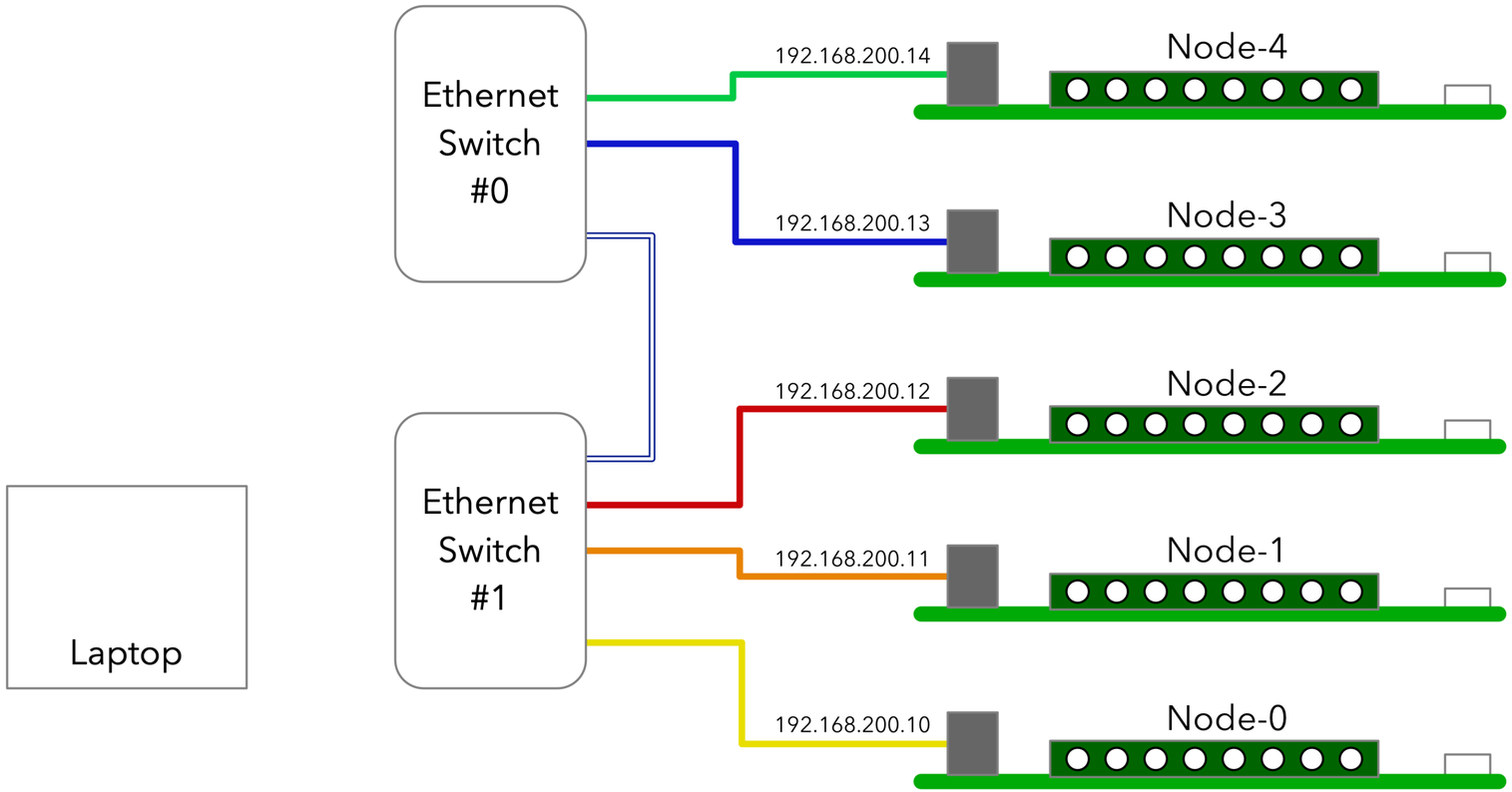
Setting up your Cluster

- Unpacking and wiring it all together
- Configuring your laptop
  - Set-up the traveller router
  - Set-up password-less login
- Give it a spin to test if everything works correctly

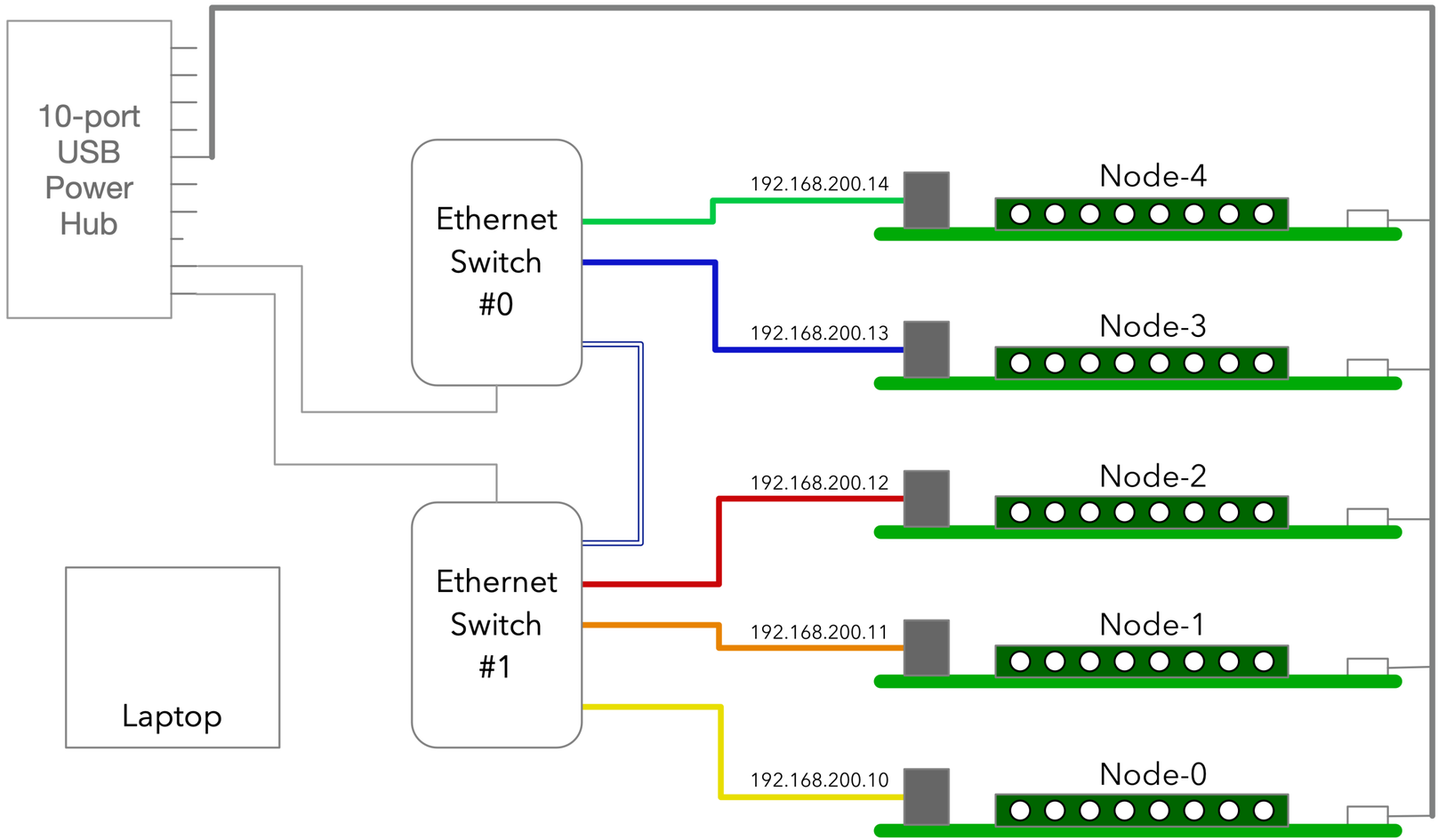
Raspberry-Pi Cluster set-up



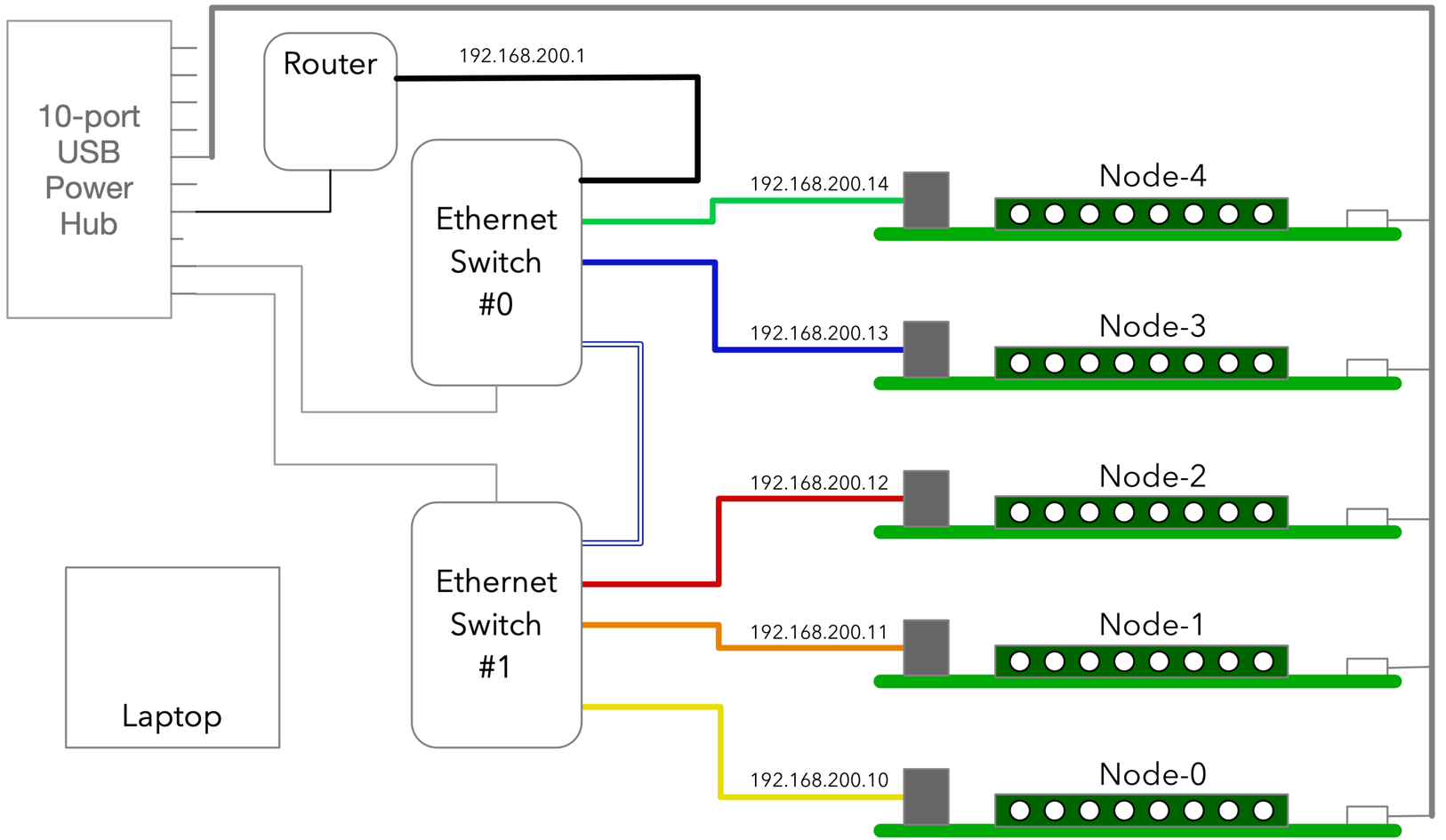
Raspberry-Pi Cluster set-up



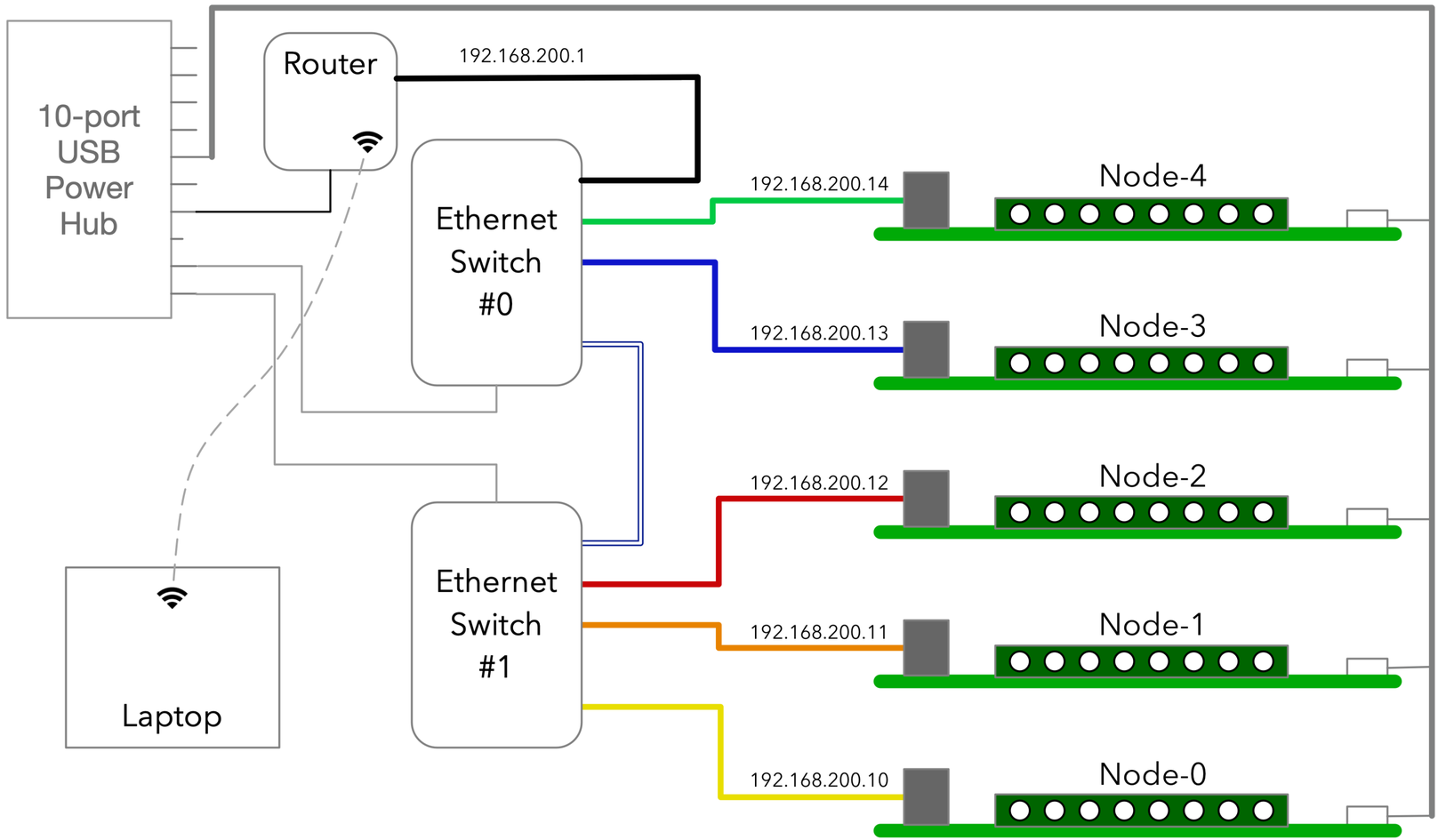
Raspberry-Pi Cluster set-up



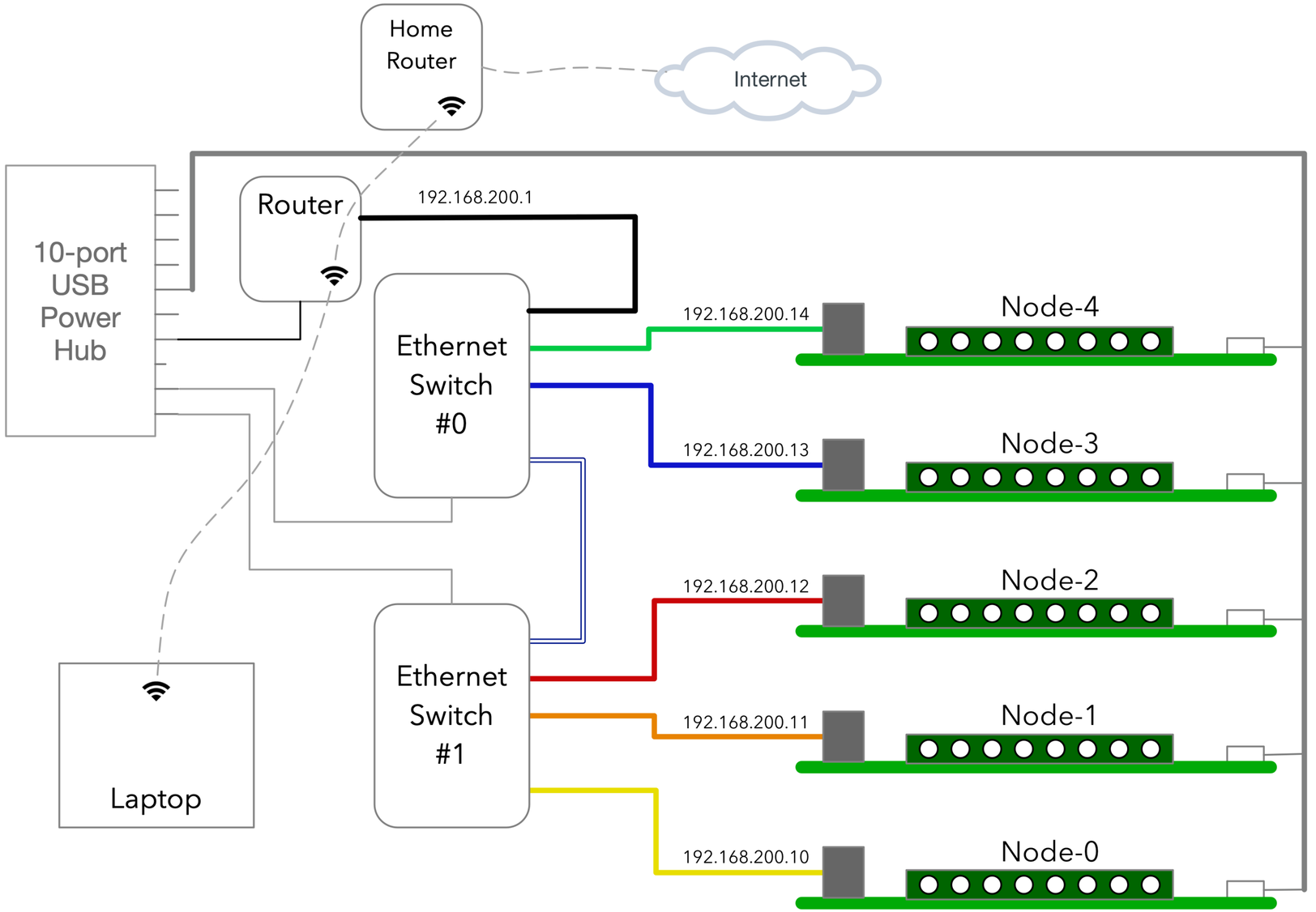
Raspberry-Pi Cluster set-up



Raspberry-Pi Cluster set-up



Raspberry-Pi Cluster set-up



Configuring your laptop

- Follow the set-up instructions here
- Run the first exercise in the workshop to verify your configuration

```
sbt assembly
./copy 0
```

- Login on each node with account akkapi and run the exercise

```
./run 0
```

- The code in the first exercise will flash the LEDs on the LED strip connected to the node

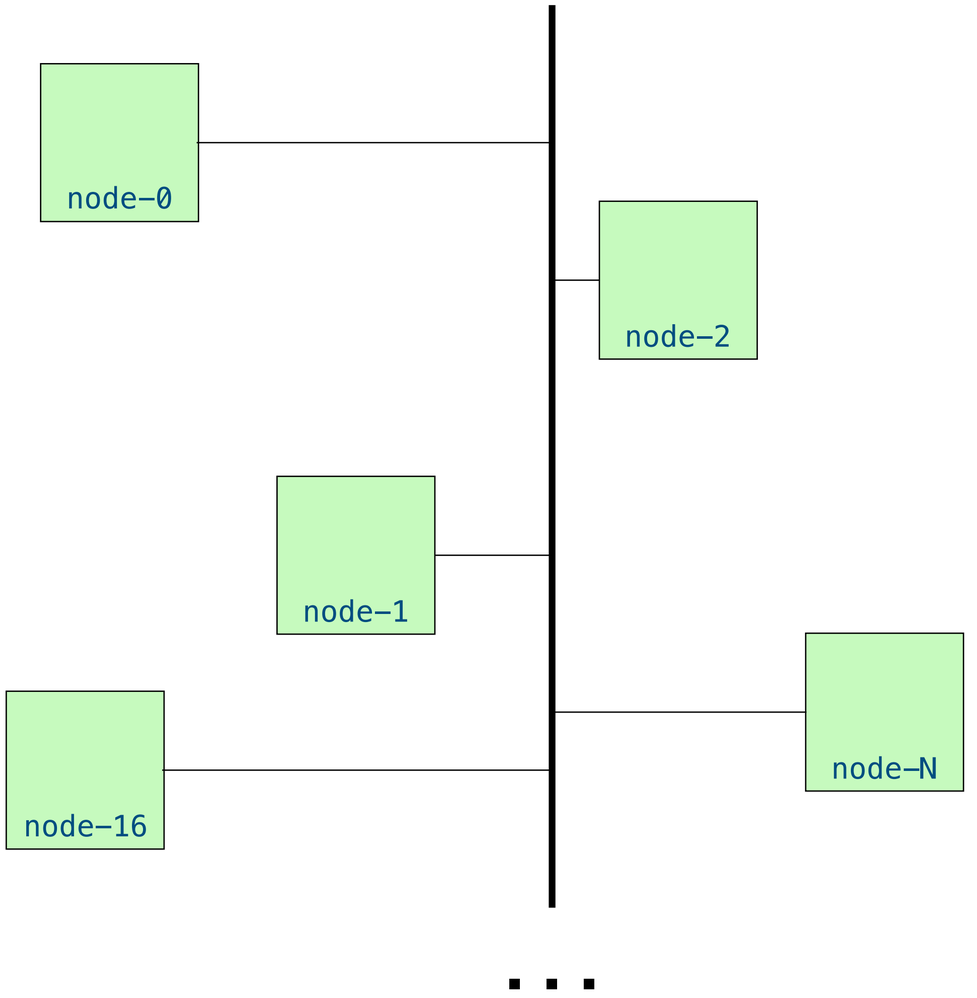
Configuring your laptop

- The exercises are in a so-called multi-project sbt build
- The exercises are in folders named *exercise_ddd_…*
- Fire-up *sbt* in the cloned project root folder
- Select the project by running the *project* command
- run the “*man e”* command for exercise instructions
- Do a clean build by running *clean* and *assembly*
- Copy the generated artefacts by using the *copy* command
- *ssh* to the different nodes using the akkapi account
- Run the code using the *run* command followed by the exercise number

A brief introduction to Akka Cluster

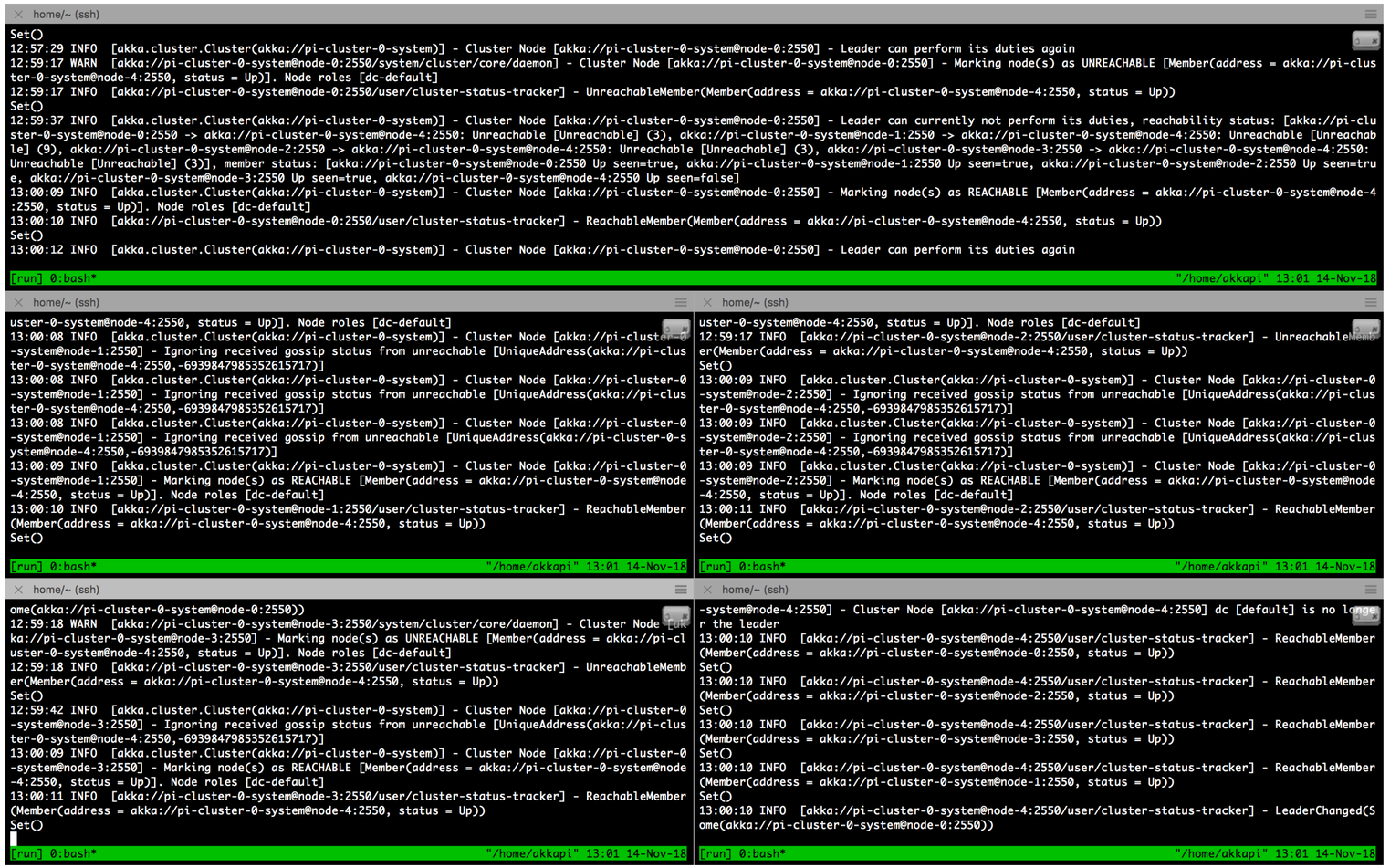
Why Clustering?

Build Resilient, Elastic [Stateful] Services

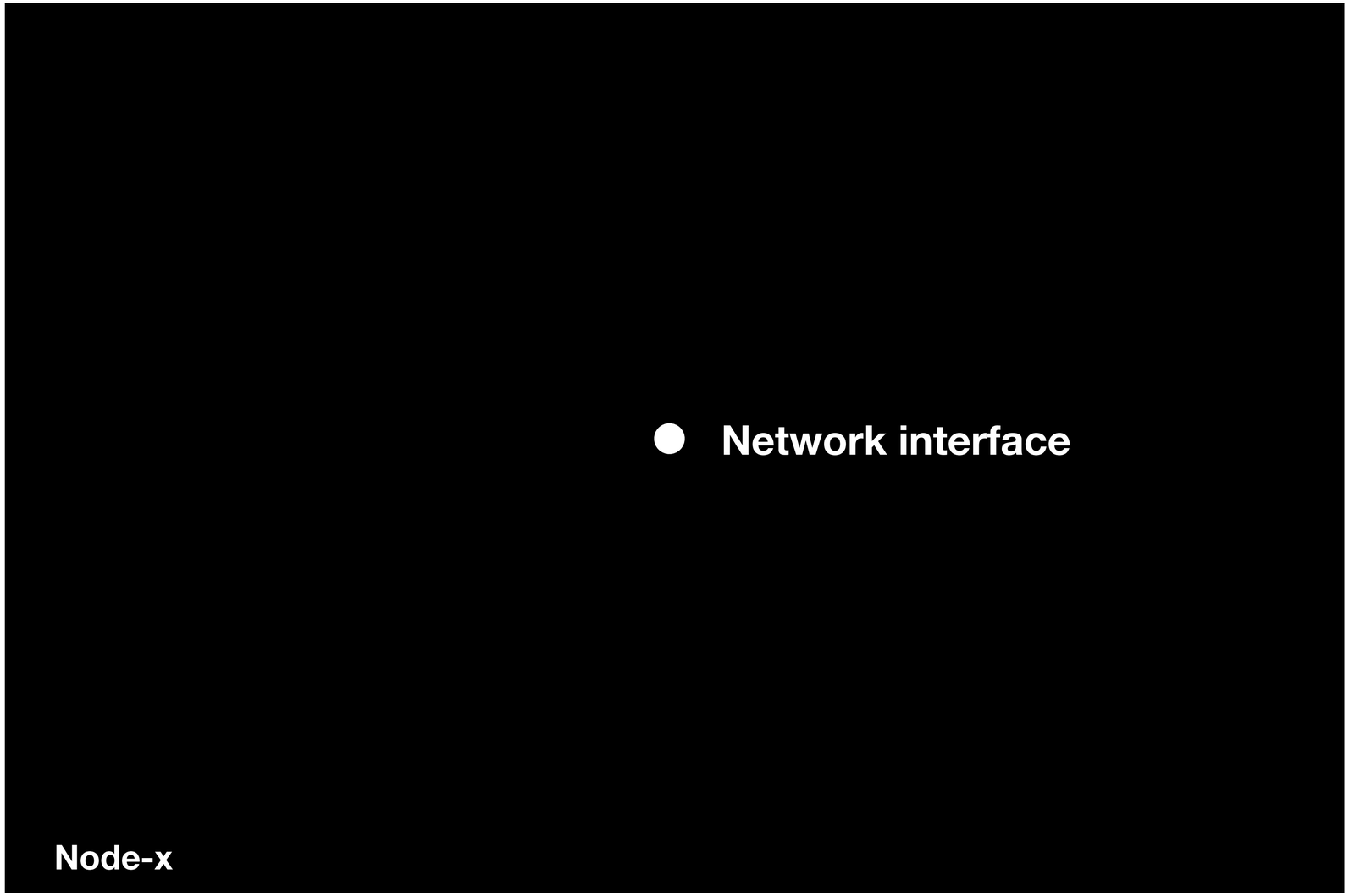


Afraid of Clustering too?

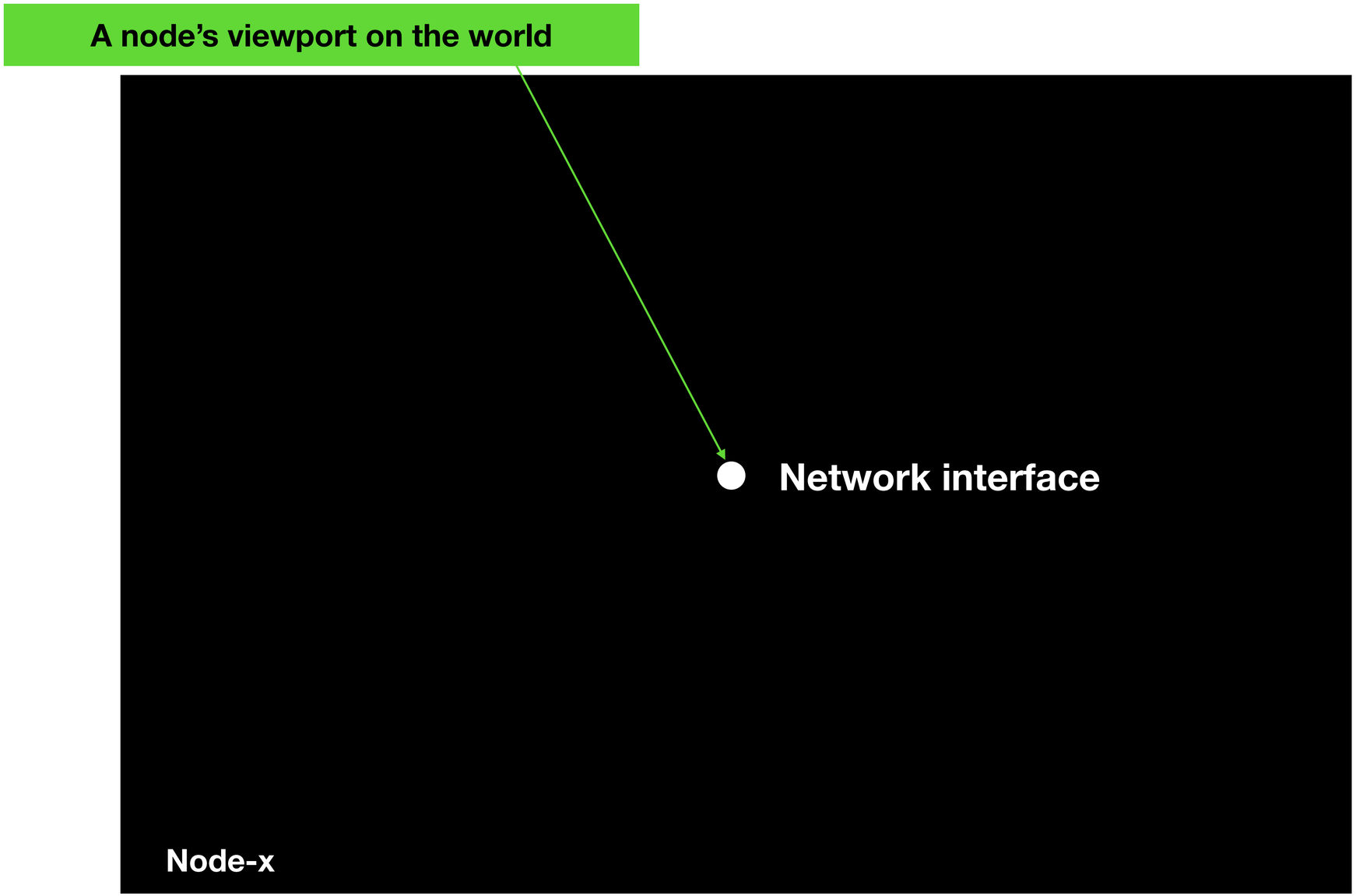
- Documentation is a challenge…
- Browsing the logs & correlating events is another challenge…



What’s tough about understanding Clustering?



What’s tough about understanding Clustering?

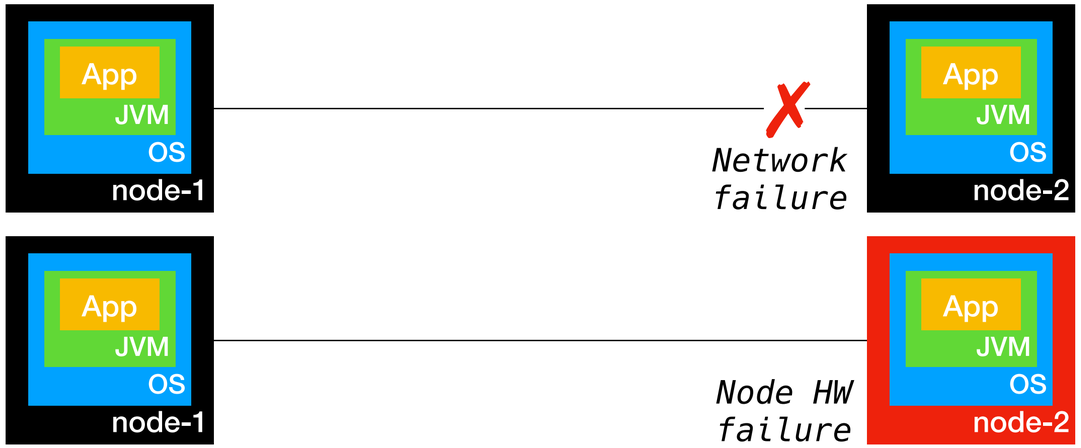


Indistinguishable failure scenarios

node-1 sees node-2 as unreachable

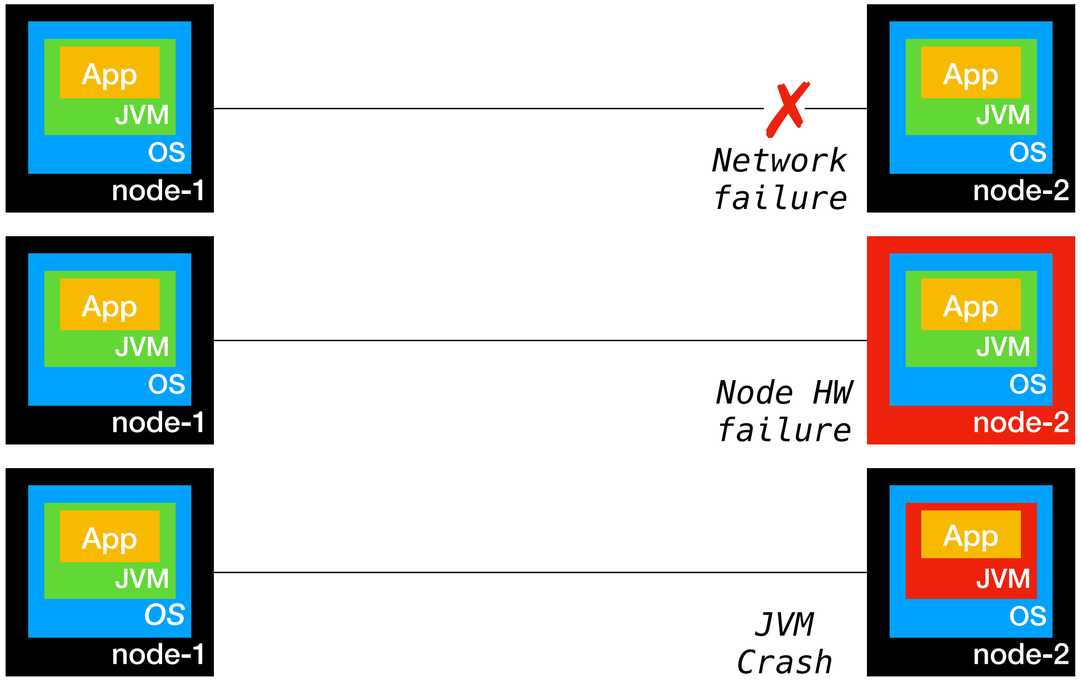
Indistinguishable failure scenarios

node-1 sees node-2 as unreachable



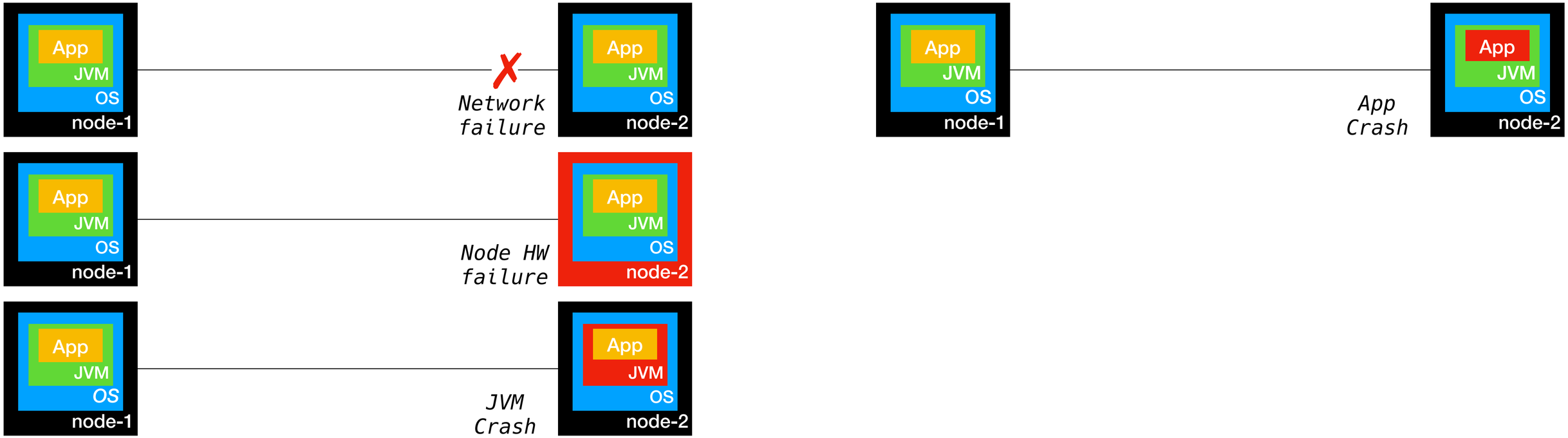
Indistinguishable failure scenarios

node-1 sees node-2 as unreachable



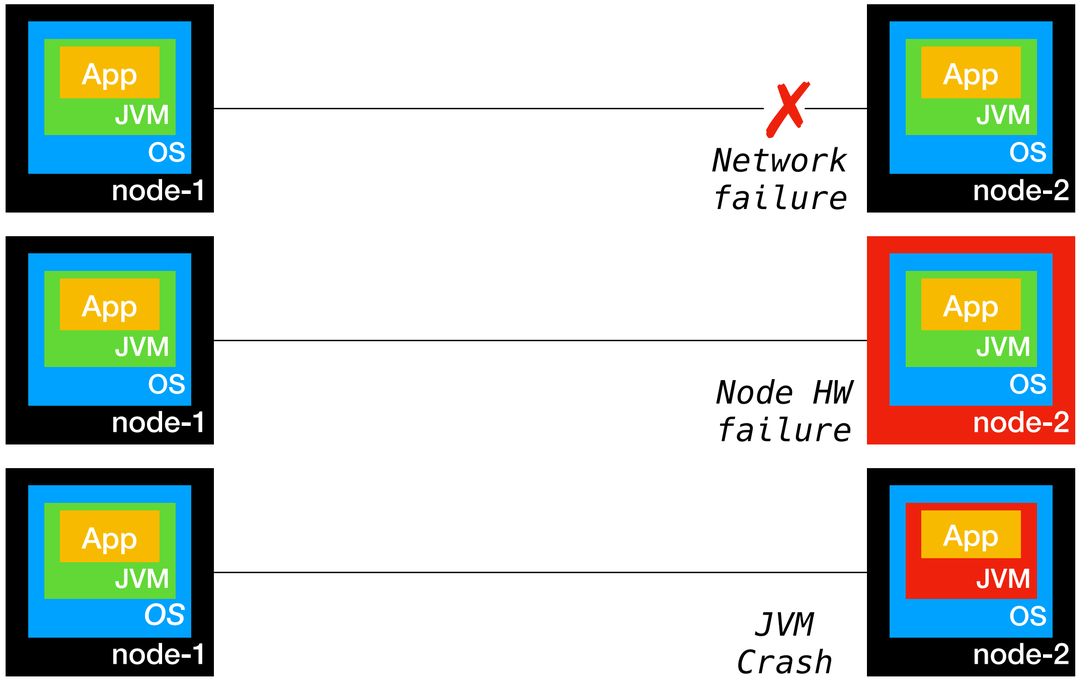
Indistinguishable failure scenarios

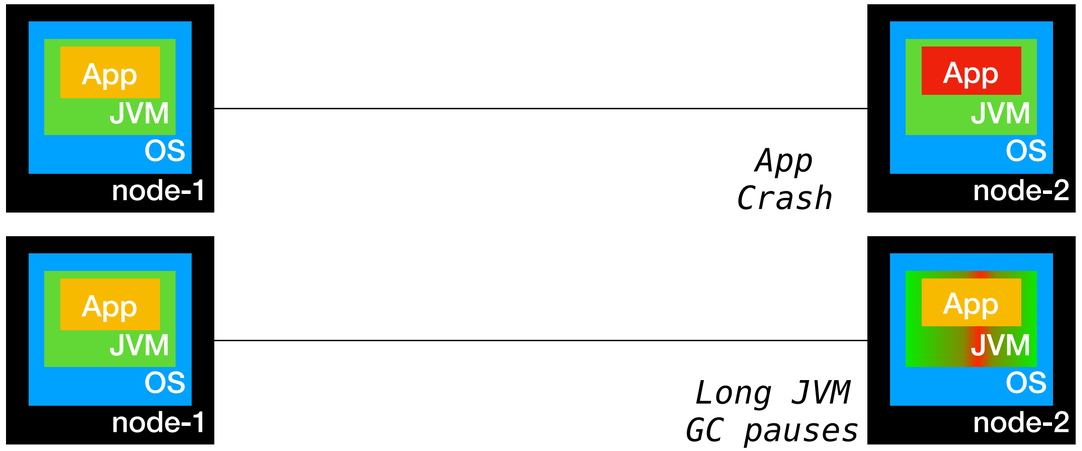
node-1 sees node-2 as unreachable



Indistinguishable failure scenarios

node-1 sees node-2 as unreachable





Akka Cluster Definition

Akka Cluster provides a fault-tolerant decentralized peer-to-peer based cluster membership service with no single point of failure or single point of bottleneck. It does this using gossip protocols and an automatic failure detector.

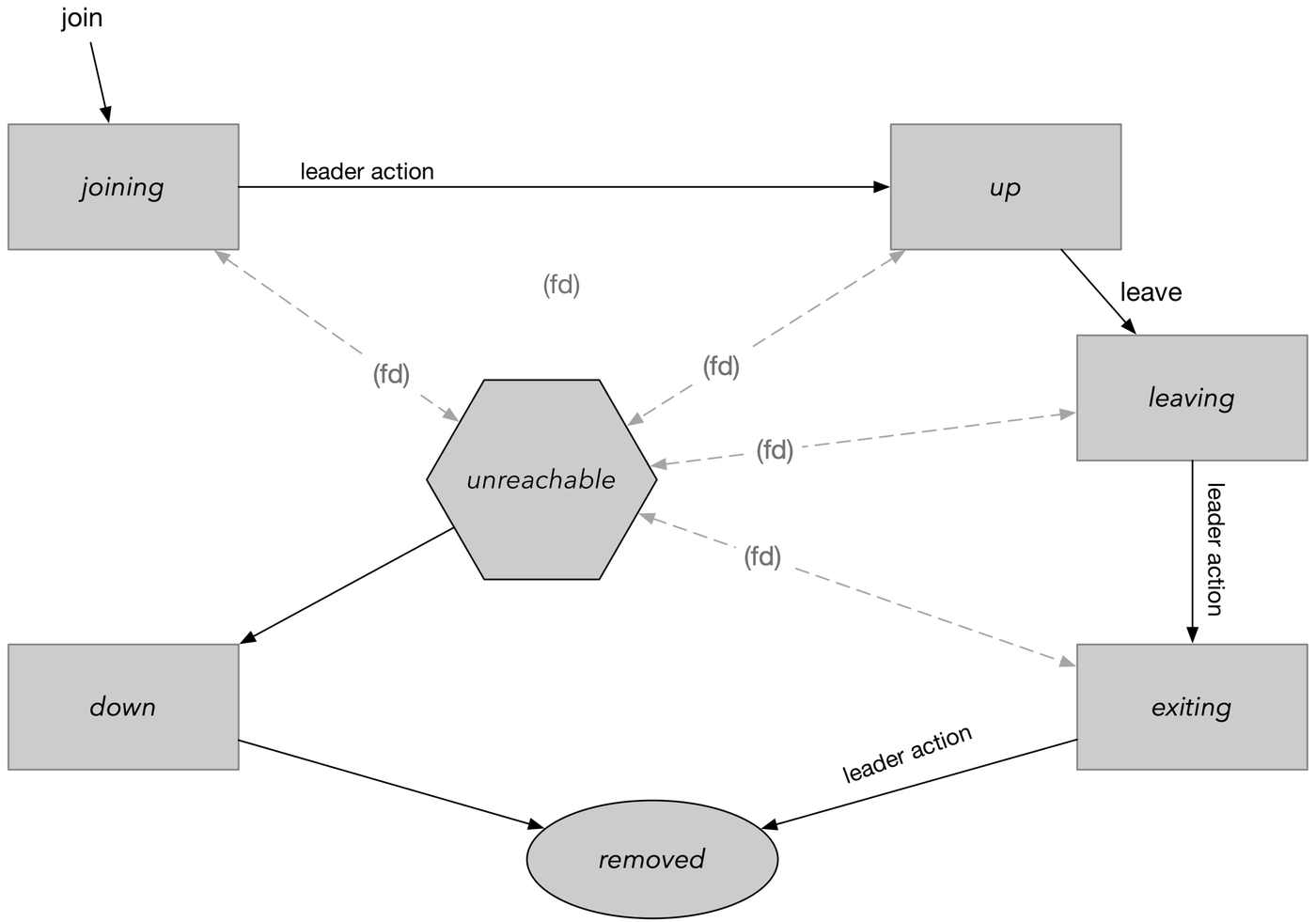
Akka Cluster

- Peer-to-peer based
- Nodes disseminate cluster state via gossipping
- Failure detection via *phi accrual failure detector*
- Every partition has a leader:
  - Dynamic role
  - Decides on key node state transitions
- Akka Cluster Singleton
  - Component which is running on one and only one node at a time

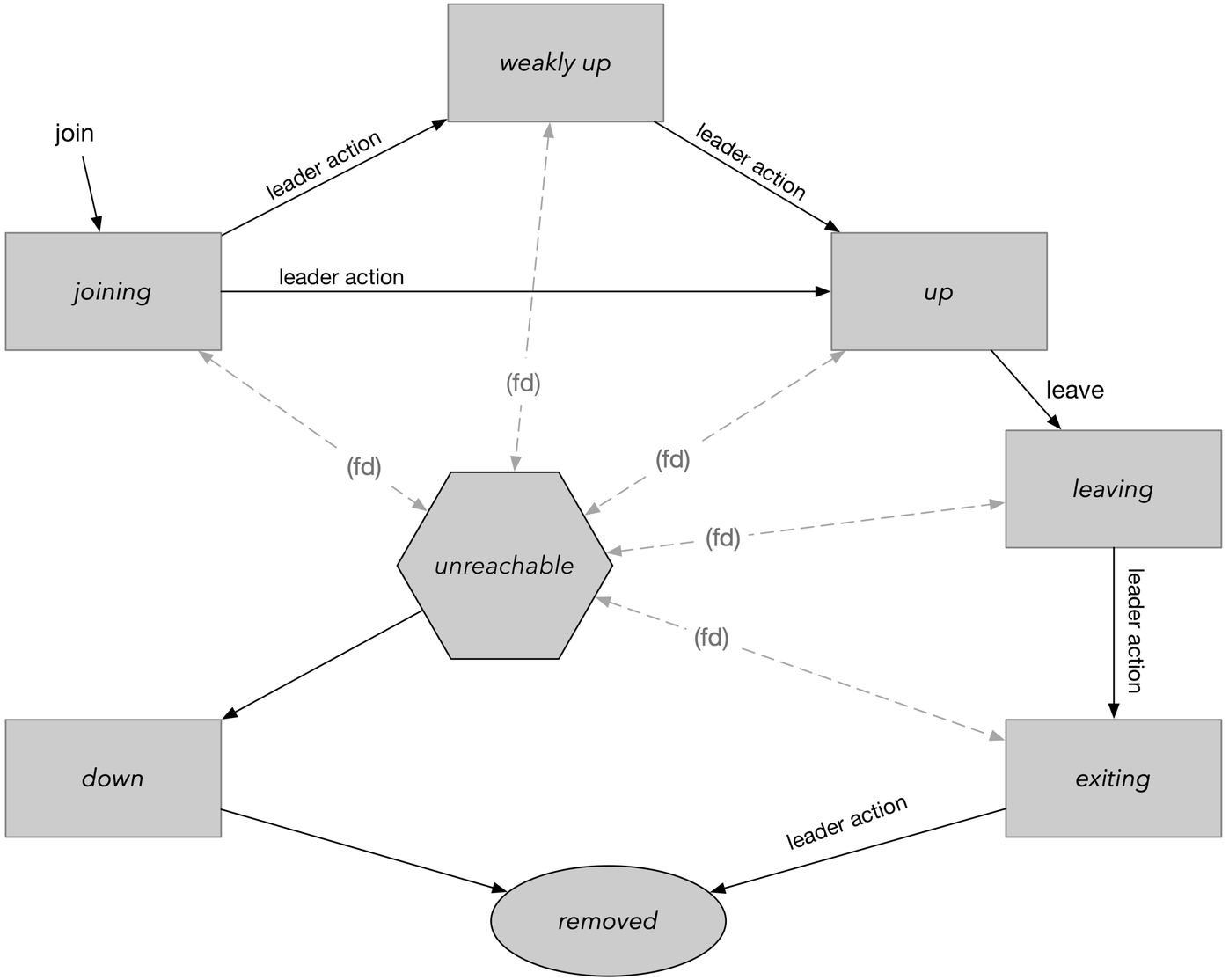
Akka Cluster membership

- A cluster is made up from a set of actor systems forming member nodes:
- An actor system can only be a member of one cluster at a time
- All actor systems within a cluster must have the same name
- Member nodes can join or leave a cluster any time:
- Joining can happen automatically using:
  - seed nodes
  - programatically
  - manually or via JMX
  - Cluster Http Management
  - Akka Cluster Bootstrap / Discovery
- Shutting down an actor system makes it leave a cluster, but a member node can also leave programatically or manually

Akka Cluster State Diagram (simplified)



Akka Cluster State Diagram



Akka Cluster Seed Nodes

- Seed nodes are initial contact points for joining nodes:
  - An actor system tries to automatically join by contacting the seed nodes
  - In order to form a cluster, the first seed node must be started
  - Specify seed nodes via the *akka.cluster.seed-nodes* configuration setting

Akka Cluster Events

```
Cluster(system).subscribe(
      listener,
      InitialStateAsEvents,
      classOf[MemberEvent],
      classOf[ReachabilityEvent],
      classOf[LeaderChanged]
)
```

- You can subscribe an actor to cluster change notifications
- The most interesting event types are:
  - MemberEvent: cluster member status change Joining, Up, Weakly-Up, Leaving, Exiting, Removed
  - LeaderChanged: changes in leader role in cluster/cluster partition
  - ReachabilityEvent: changed reachability of a cluster member

Akka Cluster Events

- In order to enable Akka Cluster, use the ClusterActorRefProvider
- Set-up remoting transport (Netty TCP in example below)

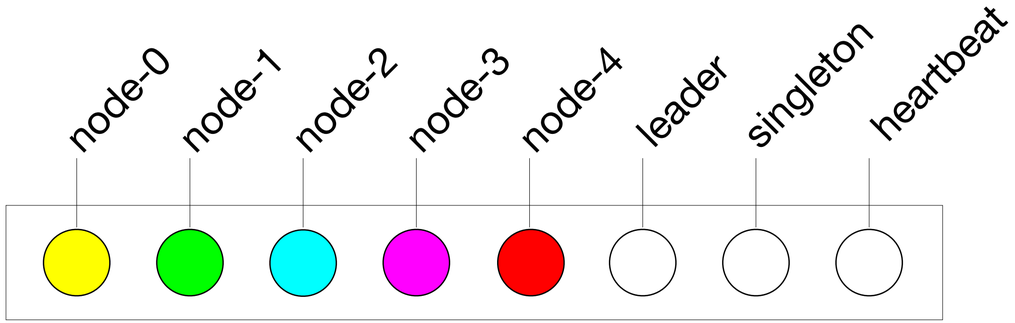
```
akka {
  actor {
    provider = akka.cluster.ClusterActorRefProvider
  }
  cluster {
    seed-nodes = [
      "akka.tcp://my-akka-cluster-system@node-1:2550",
      "akka.tcp://my-akka-cluster-system@node-2:2550"
    ]
  }
  remote {
    enabled-transports          = [akka.remote.netty.tcp]
    netty.tcp {
      hostname = "127.0.0.1"
      port     = 2550
    }
  }
}
```

Cluster State visualisation

Show the state of all nodes,
on each node,
as seen by that node

Cluster State visualisation

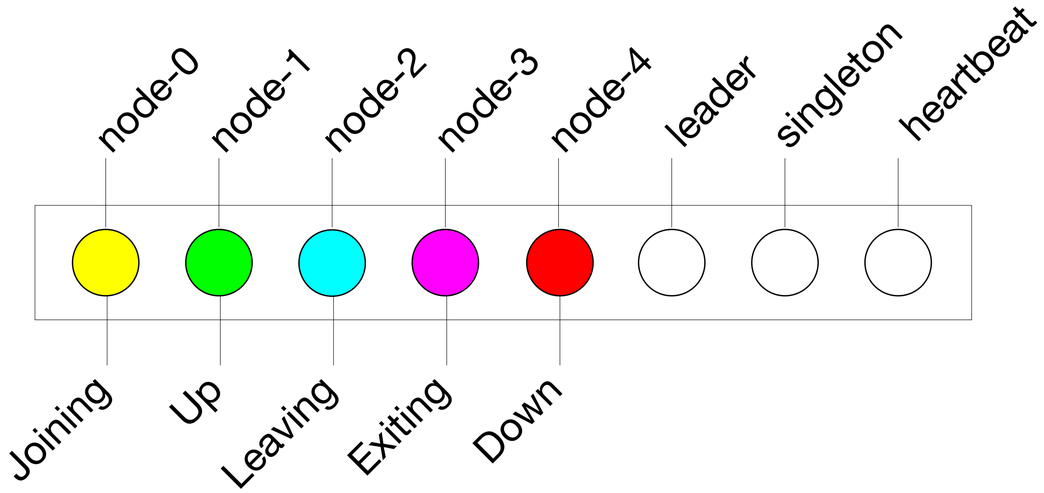
LED legend



Purplish white: Node is unreachable

Cluster State visualisation

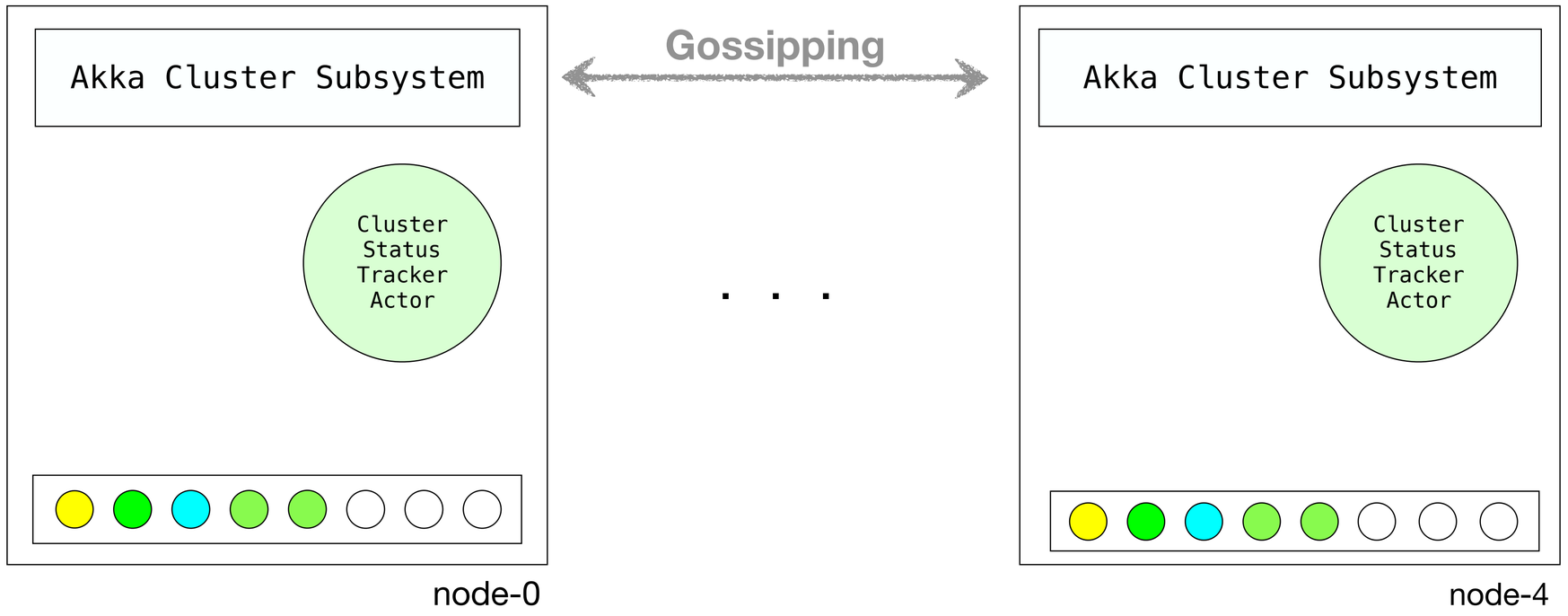
LED legend



Purplish white: Node is unreachable

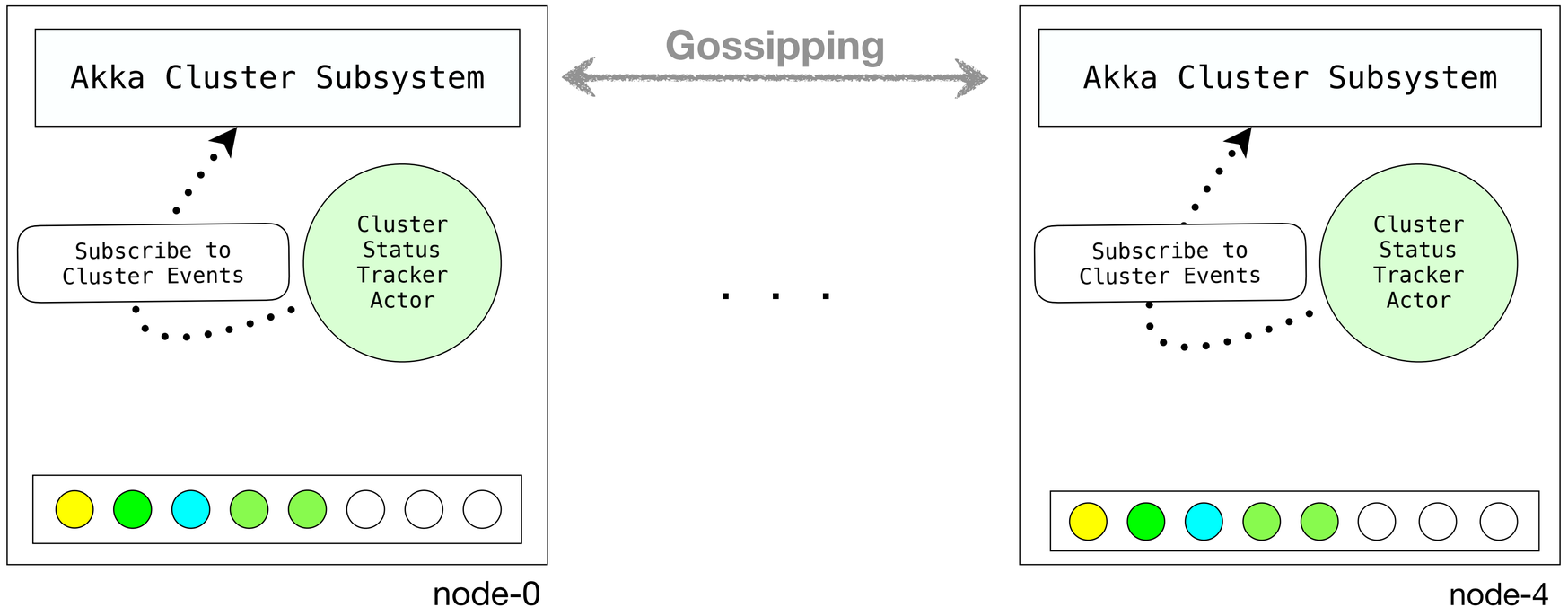
Cluster State visualisation

Behind the screens



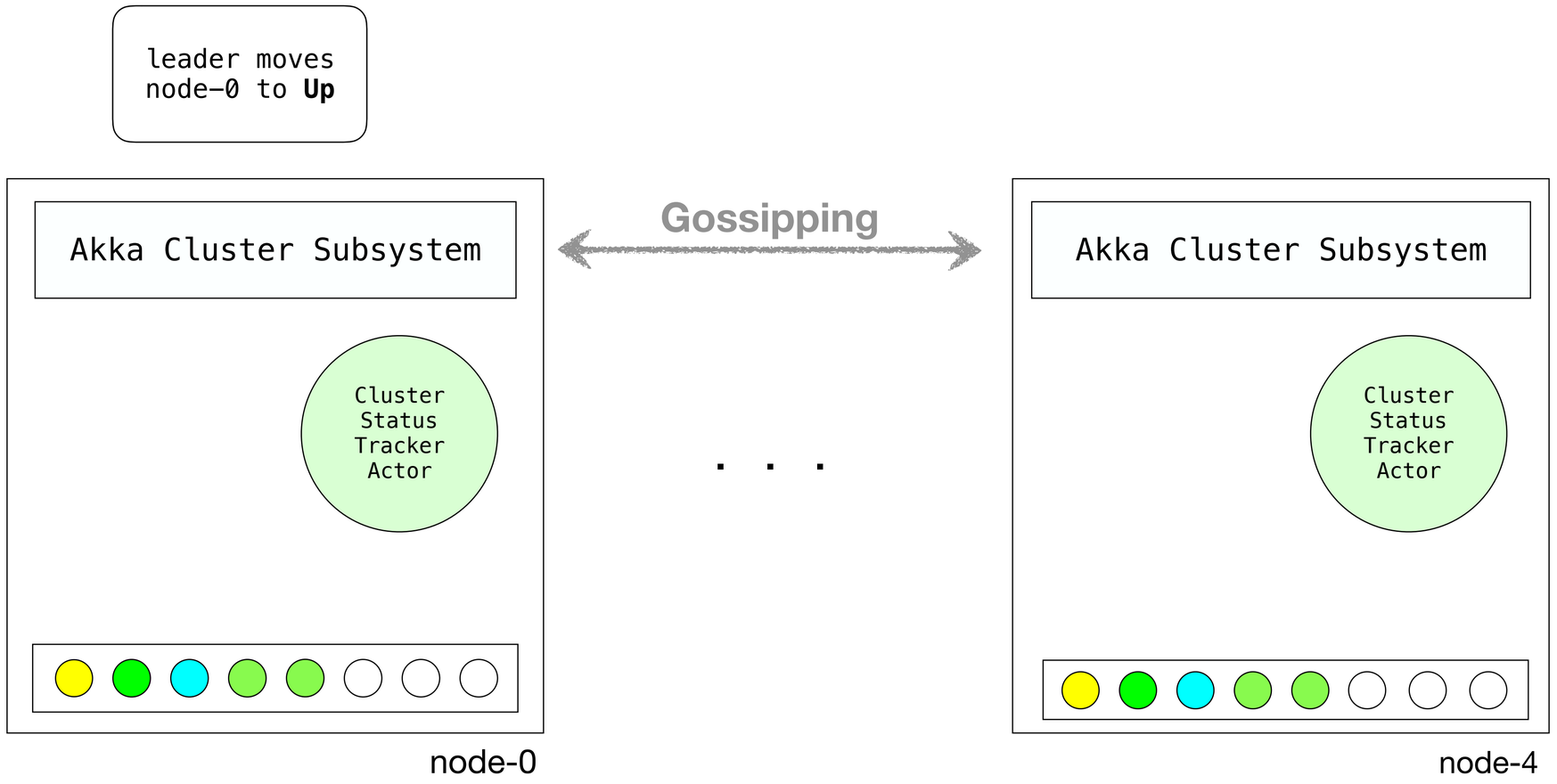
Cluster State visualisation

Behind the screens



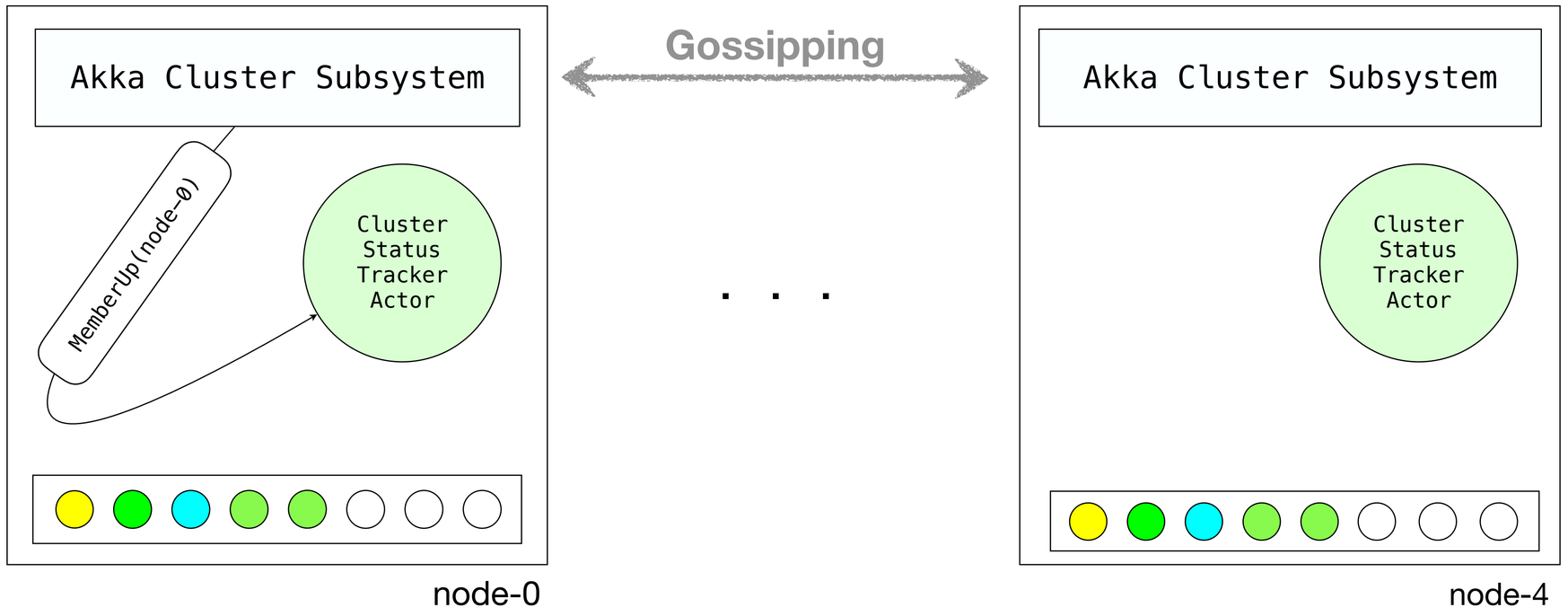
Cluster State visualisation

Behind the screens



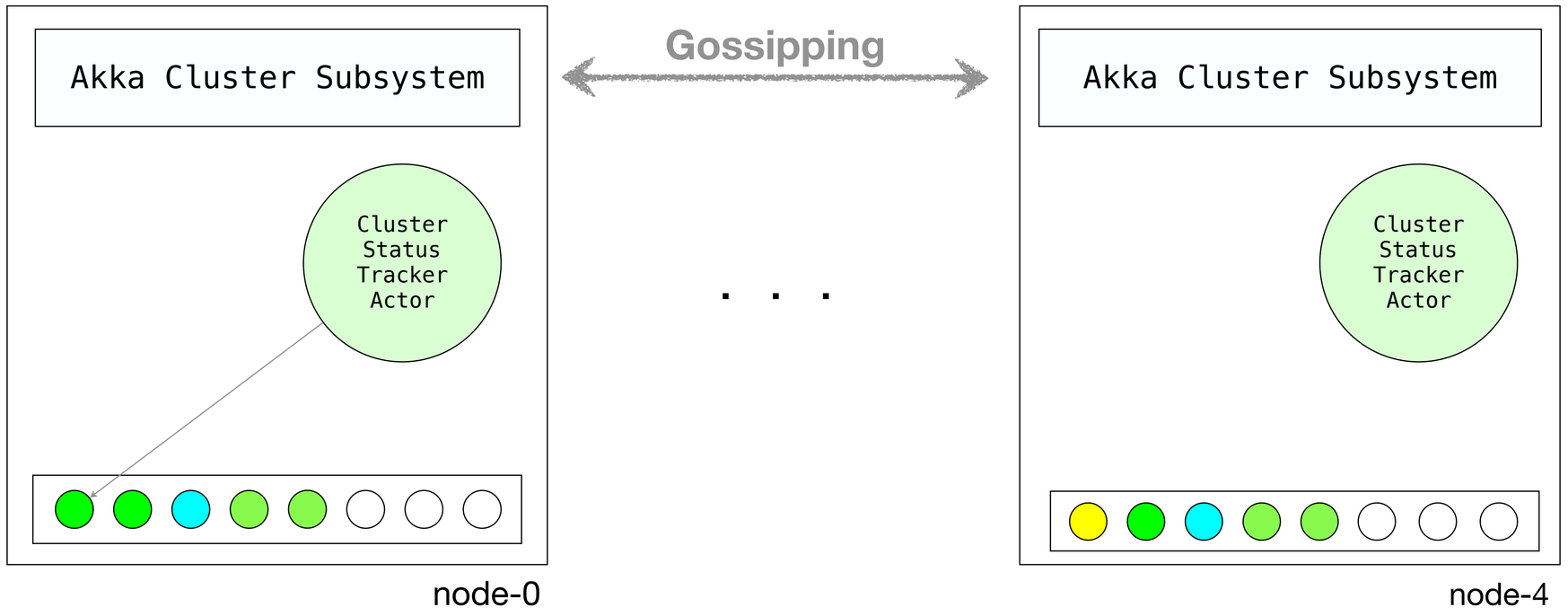
Cluster State visualisation

Behind the screens



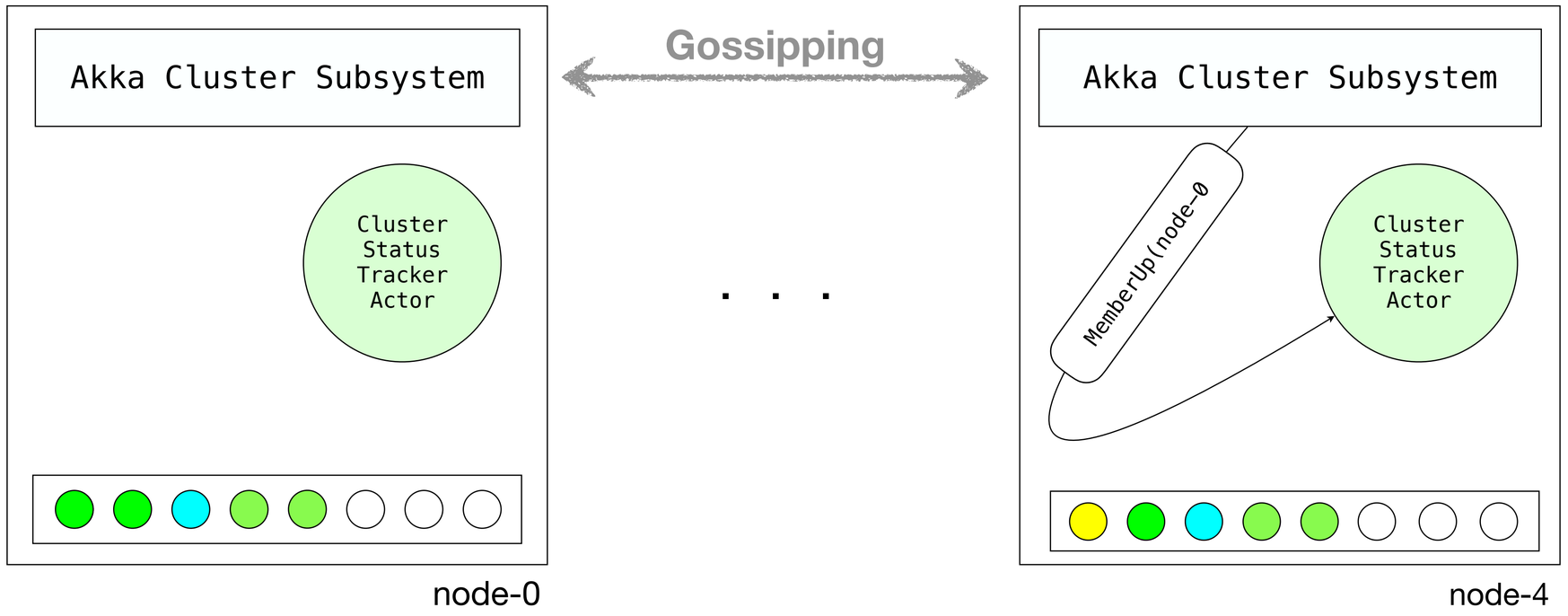
Cluster State visualisation

Behind the screens



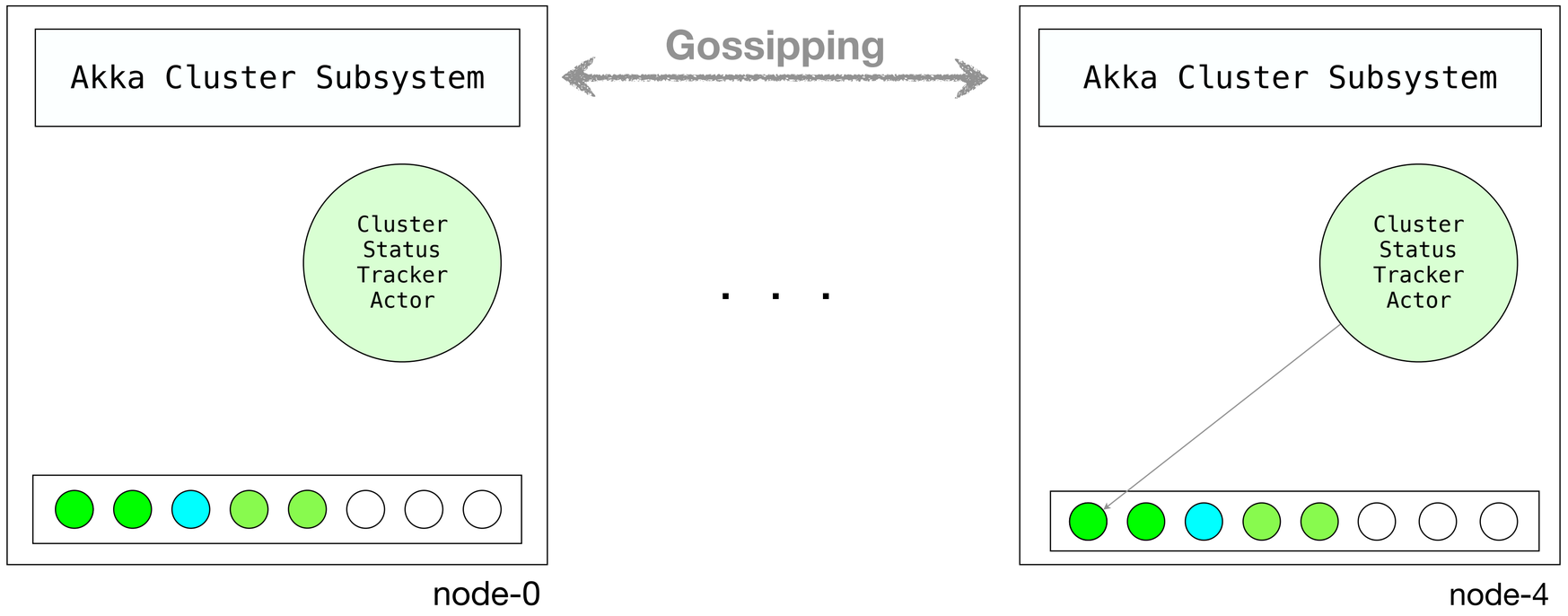
Cluster State visualisation

Behind the screens



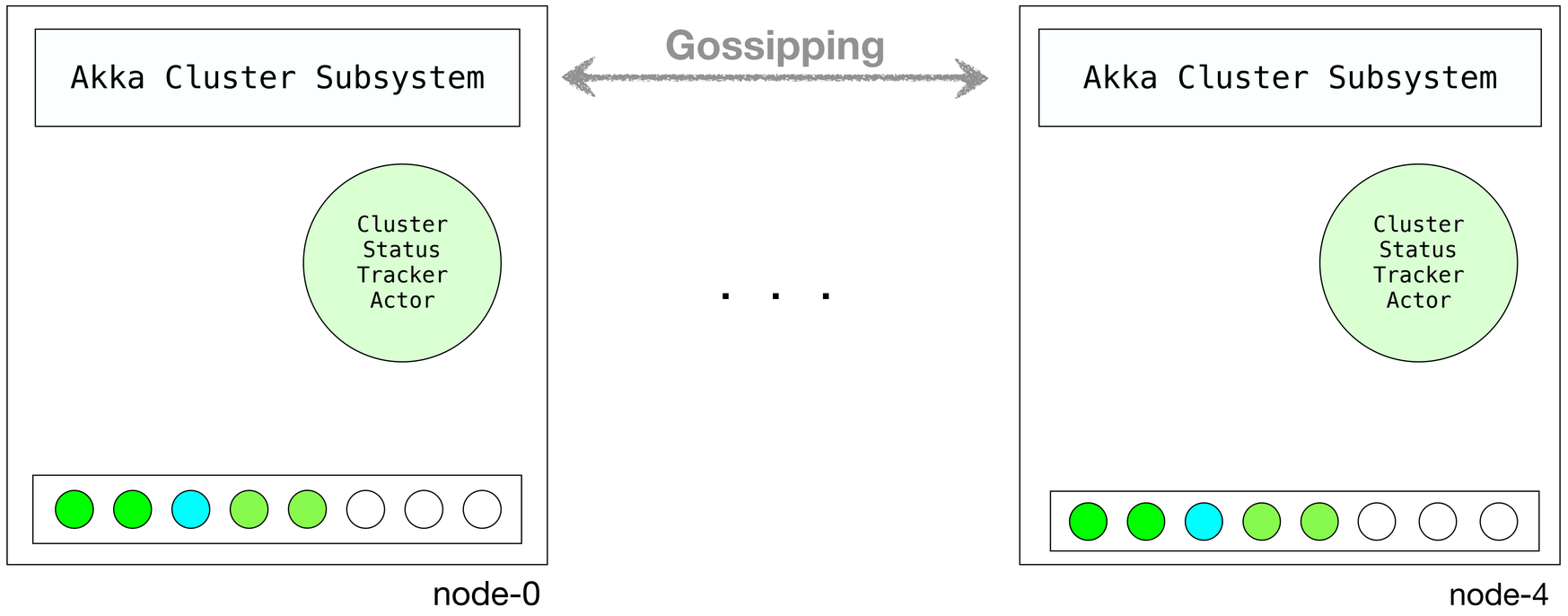
Cluster State visualisation

Behind the screens

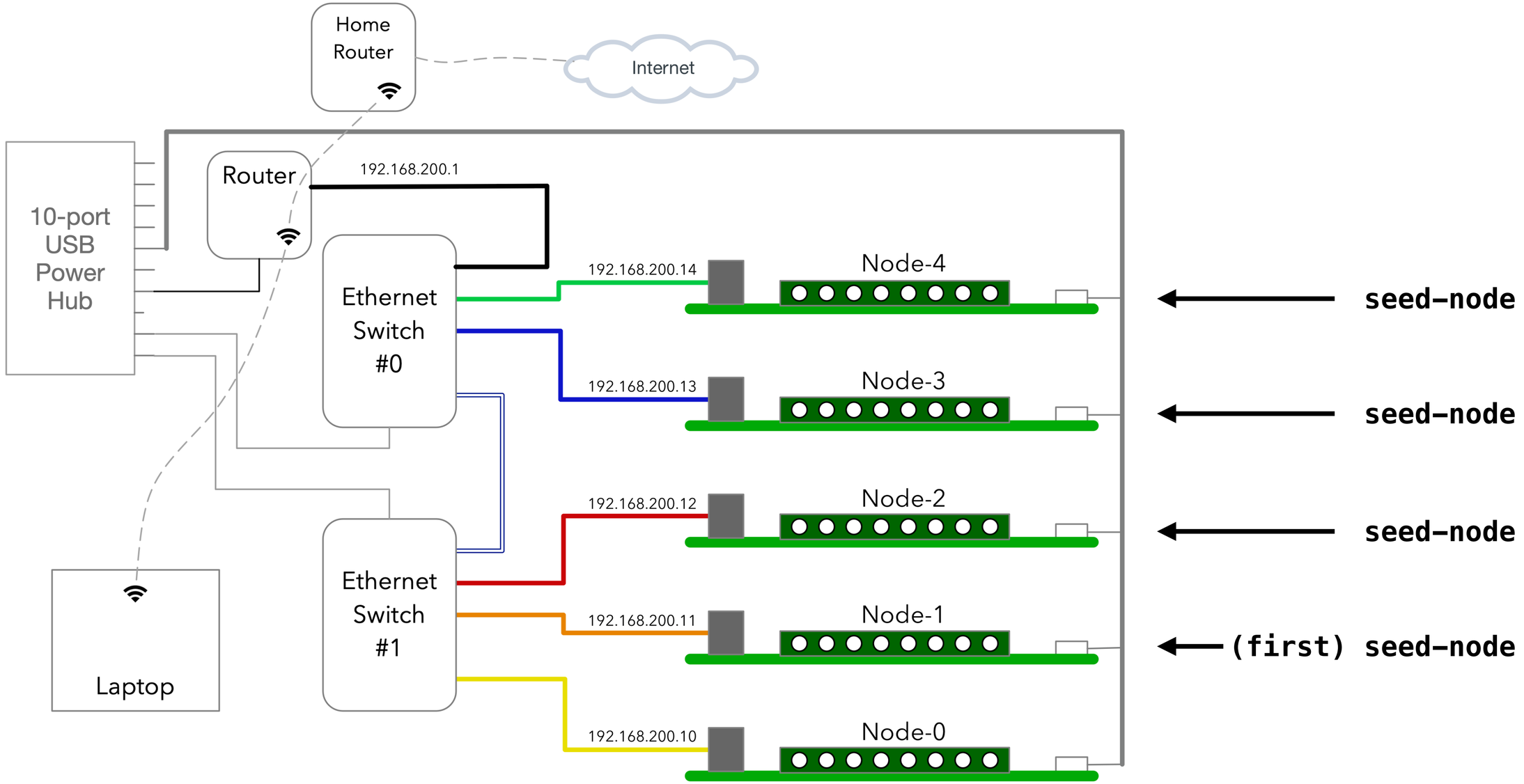


Cluster State visualisation

Behind the screens



Akka Cluster formation - seed-node approach



Akka (Cluster) modules

- Akka Cluster
  - Akka Cluster Singleton
  - Akka Cluster Sharding
  - Akka Cluster Multi-DC
  - Akka Cluster HTTP Management
  - Akka Distributed Data
  - Akka Distributed Pub/Sub
  - Akka Split Brain Resolver (not open source)
  - Akka Cluster Bootstrap / Discovery
- Akka
  - Akka Persistence
  - Akka Persistence Query
  - Akka Streams
  - Akka HTTP
  - Alpakka

Exercises

Basic Akka Cluster formation

- We will use so-called seed nodes to assist in the process of cluster formation
- We pass the list of seed nodes to cluster nodes via configuration
- Relative ordering of the seed nodes in important
  - The first seed node bootstraps the cluster formation process by joining itself
  - Make sure that all nodes use exactly the same seed-node configuration!

EXERCISE - Basic cluster formation

- In this exercise, we explore basic cluster formation
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster base >
```

Basic Akka Cluster formation

- By default, a Netty TCP based Akka remoting transport is enabled
- A high performance and high throughput remote transport named Artery TCP was added. It will become the default in Akka 2.6
  - Separate TCP channels for user and system messages
  - Akka Streams based
  - Lower latency
  - Higher througput
- Both Netty TCP and Artery TCP can be configured with TLS enabled
- There's also Artery UDP
  - No TLS support
  - Not covered in this workshop

Artery TCP - Configuration

- Switching from Netty TCP based Akka remote transport to Artery TCP is done via configuration:

```
akka {
    remote {
    artery {
      transport = tcp
      enabled = on
      canonical {
        hostname = "10.11.12.13"
        port = 2550
      }
    }
  }
}
```

- Note that the protocol type in URI's change from *akka.tcp* to *akka*
- Example: *akka://my-cluster-system@node-3:2550*

EXERCISE - Enable Artery TCP

- In this exercise, we will switch from the default *Netty TCP* based remote transport to *Artery TCP*
- Make sure your *sbt* prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster base move to artery tcp >
```

Basic Akka Cluster formation

- Nodes can only join a cluster and move to the *Up* state when the cluster is in a so-called *converged* state
- When one or more nodes that were part of a *converged* cluster are *unreachable*, the cluster is not in a *converged* state
- In a *non-converged* state, a *leader*
  - cannot move a node from the *Joining* state to the *Up* state
  - can move a node to the so-called *Weakly-Up* state
- The *Weaky-Up* behaviour is enabled by default
- Disable *Weakly-Up* via configuration:

```
akka.cluster.allow-weakly-up-members = off
```

- After cluster convergence, *Weakly-Up* members will be moved to the *Up* state by the *leader*
- *Weakly-Up* members:
  - should not be taken into account to make quorum decisions
  - can take part in running some user services

EXERCISE - Weakly-Up nodes

- In this exercise, we will demonstrate the occurrence of so-called *Weakly-Up* members
- Make sure your *sbt* prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster weakly up >
```

Akka Management - Cluster HTTP Management

- Use Akka Cluster HTTP Management to manage Cluster nodes:
  - *HTTP GET:* Returns state of Cluster or a specified node in a Cluster
  - *HTTP POST:* Executes JOIN operation in cluster for a provided address
  - *HTTP DELETE:* Executes LEAVE operation for a provided address
  - *HTTP PUT:* Executes LEAVE or DOWN operation for a provided address
- Note that all operations that can modify the state of a cluster (POST, DELETE, PUT) are disabled by default
  - Add the following setting to enable:

```
akka.management.http.route-providers-read-only = false
```

Cluster HTTP Management - Get cluster status

- Example GET request with JSON response

```
$ curl --request GET node-0:8558/cluster/members | jq
{ "selfNode":"akka://pi-cluster-0-system@node-0:2550",
  "leader":"akka://pi-cluster-0-system@node-0:2550",
  "oldest":"akka://pi-cluster-0-system@node-1:2550",
  "unreachable":[],
  "members":[
    { "node":"akka://pi-cluster-0-system@node-0:2550",
      "nodeUid":"-5739973322783592 32",
      "status":"Up",
      "roles":["dc-default"]
    },
    { "node":"akka://pi-cluster-0-system@node-1:2550",
      "nodeUid":"-6319386035827941279",
      "status":"Up",
      "roles":["dc-default"]
    }
  ]
}%
```

Cluster HTTP Management - Operations on members

- Following are a few examples of operations on Cluster members using HTTP DELETE and HTTP PUT requests

```
$ curl --request DELETE \
    node-0:8558/cluster/members/akka://pi-cluster-0-system@node-0:2550
{"message":"Leaving akka://pi-cluster-0-system@node-0:2550"}%
$ curl --request DELETE \
    node-1:8558/cluster/members/akka://pi-cluster-0-system@node-1:2550
{"message":"Leaving akka://pi-cluster-0-system@node-1:2550"}
```

```
$ curl --request PUT -d operation=DOWN \
    node-0:8558/cluster/members/akka://pi-cluster-0-system@node-1:2550
{"message":"Leaving akka://pi-cluster-0-system@node-1:2550"}%

$ curl --request PUT -d operation=DOWN \
    node-0:8558/cluster/members/akka://pi-cluster-0-system@node-0:2550
{"message":"Leaving akka://pi-cluster-0-system@node-0:2550"}%
```

Akka Cluster Singleton

- An Akka Cluster Singleton is an actor of which there's at most one instance running within an Akka Cluster at any moment in time
  - Akka Cluster will create an instance of the Akka Cluster Singleton on one of the nodes in a converged cluster
  - When the node on which the Cluster Singleton is downed or removed, Akka cluster will re-instantiate the Singleton on another node, if any
  - In normal use cases, an Akka Cluster Singleton will hold state
  - If that state should survive a re-instantiation of the Cluster Singleton, the state should be persisted (and recovered)

EXERCISE - Akka Cluster Singleton & HTTP Management

- In this exercise, we will add a cluster singleton to our application
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster singleton >
```

Akka Management - Cluster HTTP Management

- There is a setting `akka.cluster.auto-down-unreachable-after` taking a duration as value
  - ***DON'T USE IT !!!***
  - It's there primarily to use it in Akka test scenarios
  - Auto-downing will lead to so-called *Cluster Split-brain syndrome*
  - The occurrence of a split-brain will most probably lead to data corruption
- Let's illustrate what can happen when this feature is enabled...

EXERCISE - The perils of auto-downing

- In this exercise, we will observe what happens when a network partition occurs while the auto-downing "feature" is enabled
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster the perils of auto downing >
```

EXERCISE - Disable Weakly-Up

- In this exercise, we disable the *Weakly-Up* feature and observe the new behaviour
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster weakly up disabled >
```

Akka Split-Brain resolver

- An Akka Cluster leader will not move nodes to the *Up* or *Down* state when the cluster is not in a converged state
- The only ways to move a node to the *Down* state are:
  - Programmatically using the Akka Cluster API
  - Via JMX or the command line
  - Using Akka HTTP Management
- Making the right decision to *Down* a node is complex. Doing it incorrectly can lead to split-brain situation
- Alternative is to utilise a so-called Split-Brain Resolver (*SBR*)
- Lightbend's *SBR* implements many resolving *strategies*:
  - Keep Majority
  - Static Quorum
  - Keep Referee Node
  - Keep Oldest Node
  - Down All

EXERCISE - Split-Brain Resolver - Keep Majority

- In this exercise, we configure the Split Brain Resolver with a Keep Majority resolver strategy
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > split brain resolver keep majority >
```

EXERCISE - Split-Brain Resolver - Static Quorum

- In this exercise, we configure the Split Brain Resolver with a Static Quorum resolver strategy
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > split brain resolver static quorum >
```

EXERCISE - Split-Brain Resolver - Keep Referee

- In this exercise, we configure the Split Brain Resolver with a Keep Referee resolver strategy
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > split brain resolver keep referee >
```

EXERCISE - Split-Brain Resolver - Keep Oldest

- In this exercise, we configure the Split Brain Resolver with a Keep Oldest resolver strategy
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > split brain resolver keep oldest >
```

EXERCISE - Split-Brain Resolver - Down All

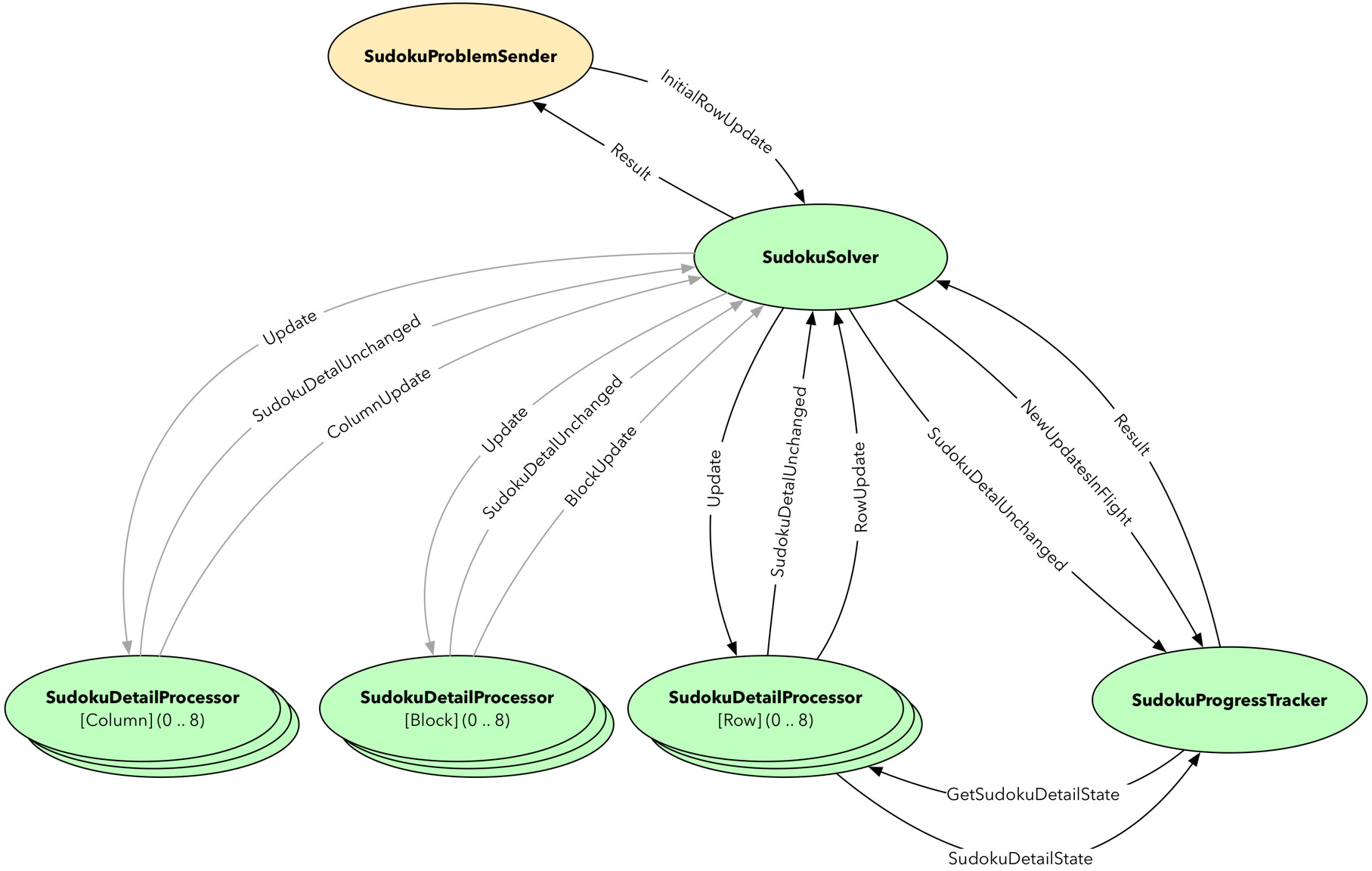
- In this exercise, we configure the Split Brain Resolver with a Down All resolver strategy
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > split brain resolver down all >
```

Clustered Applications

- Cluster computing is a popular way to distribute work, and actors are a natural fit for clustered computations.
- Akka actors paired with Akka Cluster allow you to scale problems both vertically and horizontally.
- Vertically with actors coordinating on a node to perform work, and horizontally with work spread across nodes in a cluster.
- *There's a third axis of scaling which comes with actor clusters using sharding to distribute work*.
- Up until now, we've been understanding clustered nodes, membership states and how these change in the akka cluster life cycle. Now, we will begin to look at how to run clustered applications to achieve work distribution with automatic balancing between the nodes.
- Understanding the cluster membership state will help understand how work will be balanced during different states.

Clustered Sudoku Solver



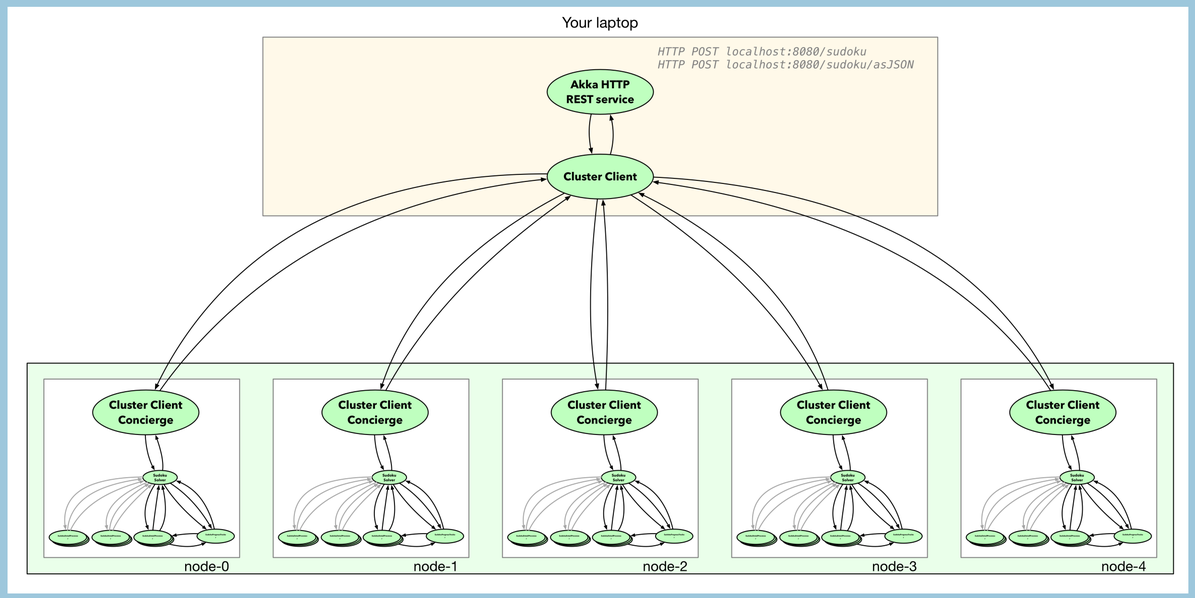
EXERCISE - Clustered Sudoku Solver

- In this exercise, we'll run an application, one that solves sudoku problems, on each node in the cluster.
- The application has two main parts; Actors that coordinate to solve the problem, and an Actor that repeatedly sends a sudoku problem to the solver
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > clustered sudoku solver >
```

EXERCISE - Add Cluster Client

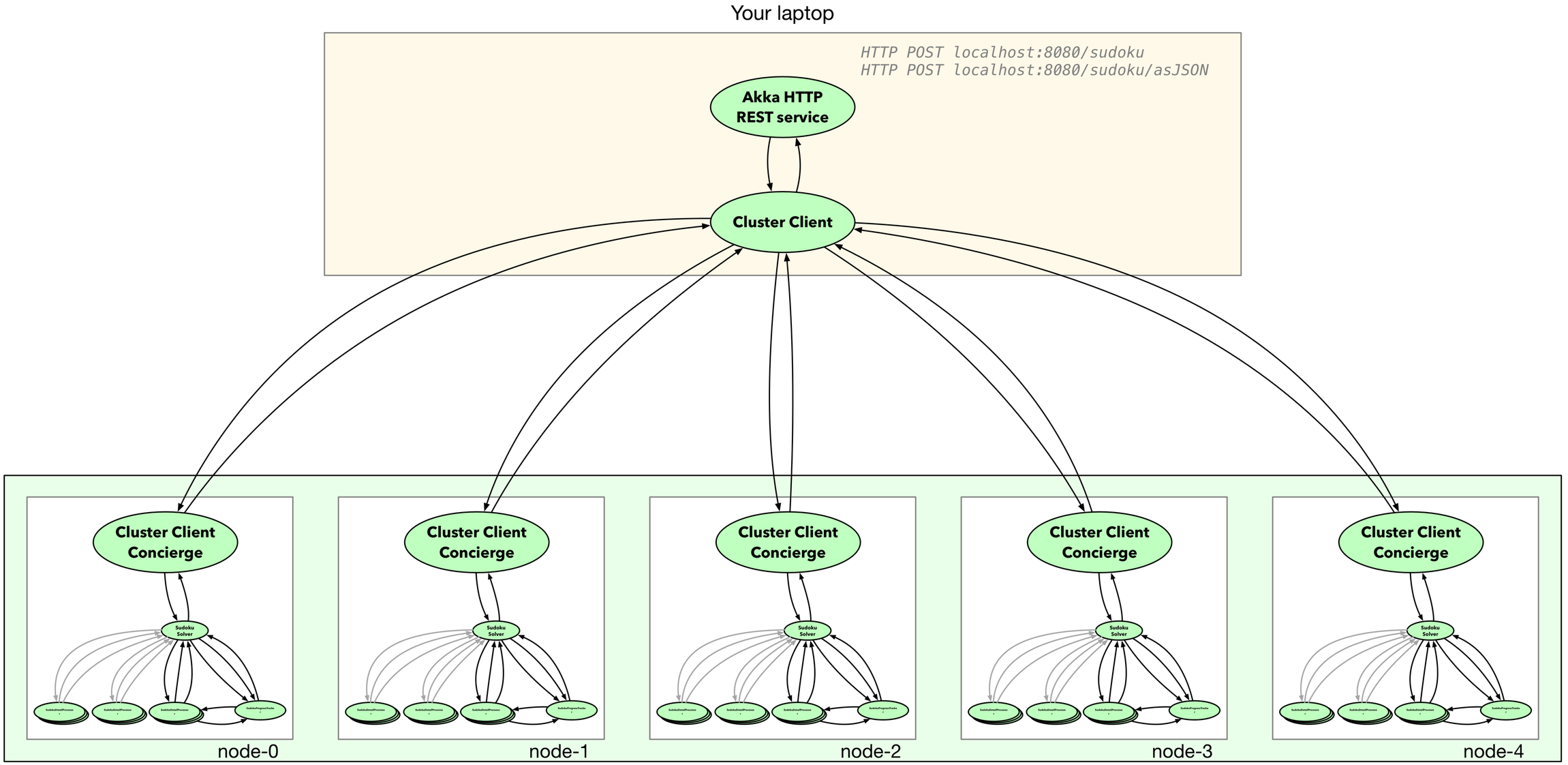
- In this exercise, we'll be running the sudoku solver again on all the nodes.
- Instead of having an actor on the node send a sudoku problem, we'll use cluster client to have an actor on a remote machine (your laptop) send problems to the cluster.



- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > add cluster client >
```

Clustered Sudoku Solver



Clustered Sudoku Solver Cluster - Client Enabled

- In this exercise, we'll be running the sudoku solver again on all the nodes.
- Instead of having an actor on the node send a sudoku problem, we'll use cluster client to have an actor on a remote machine (your laptop) send problems to the cluster.
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > clustered sudoku solver cluster client enabled >
```

Monitoring Reactive Applications - I

- Distributed systems are complex and have many moving parts, much of which are asynchronous and run in parallel.
- Lightbend Telemetry breaks capture down into composable parts that will provide better insight into your system.
- Currently Telemetry provides instrumentation (metrics, events or traces) for the following feature sets:
  - Akka
  - Akka-HTTP
  - Akka-Streams
  - Scala Futures
  - Lagom
  - Play

Monitoring Reactive Applications - II

- Monitoring clustered applications involves monitoring both the cluster state and metrics specific to the application running on the cluster.
- Monitoring Akka applications is made easier using *Lightbend Telemetry*, a tool that collects specific telemetry related to Reactive Applications.
- In addition to cluster metrics, you'll see specific metrics around actors.
- The full list can be found in Cinnamon docs

Enable Lightbend Telemetry

- Lightbend Telemetry is enabled by adding the following settings to your configuration

```
cinnamon {
  akka.cluster {
    domain-events = on
    member-events = on
    singleton-events = on
    shard-region-info = on
  }
  akka.actors {
    "/user/*" {
      report-by = instance
    }
  }
  prometheus {
    exporters += http-server

    http-server {
      host = "0.0.0.0"
    }
  }
}
```

Akka Cluster Bootstrap & Akka Discovery

- Akka Cluster Bootstrap: flexible alternative to *seed-node* based cluster initialisation.
  - Works in concert with a discovery service
    - Via configuration files - close to the seed-node approach but ordering in config is irrelevant.
    - Via *akka-dns* using *DNS A* or *DNS SRV* records

EXERCISE - Akka Cluster Bootstrap with Discovery via configuration

- In this exercise, we will experiment with Akka Cluster Bootstrap in combination with a set of statically configured discovery end-points
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster singleton akka bootstrap discovery via akka dns >
```

EXERCISE - Akka Cluster Bootstrap with Discovery via akka-dns

- In this exercise, we will experiment with Akka Cluster Bootstrap in combination with Akka Discovery via akka-dns
- Make sure your sbt prompt looks like:

```
man [e] > Pi-Akka-Cluster > cluster singleton akka bootstrap discovery via config >
```