SeMaFoR Project
& Planning for Decentralized Reconfiguration

Jolan PHILIPPE

PostDoc - SeMaFoR project

Hélène COULLON
(STACK)
SeMaFoR, Ballet

Charles PRUD'HOMME
(TASC)
SeMaFoR, Ballet

Antoine OMOND
(STACK, UiT)
Ballet

Issam Rais
(UiT - Ballet)
Ballet

Thomas LEDOUX
(STACK)
SeMaFoR

Hugo BRUNELIERE
(Naomod)
SeMaFoR

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

STACK

SeMaFoR
Self Management of Fog Resources

August 2nd, 2023

Inria

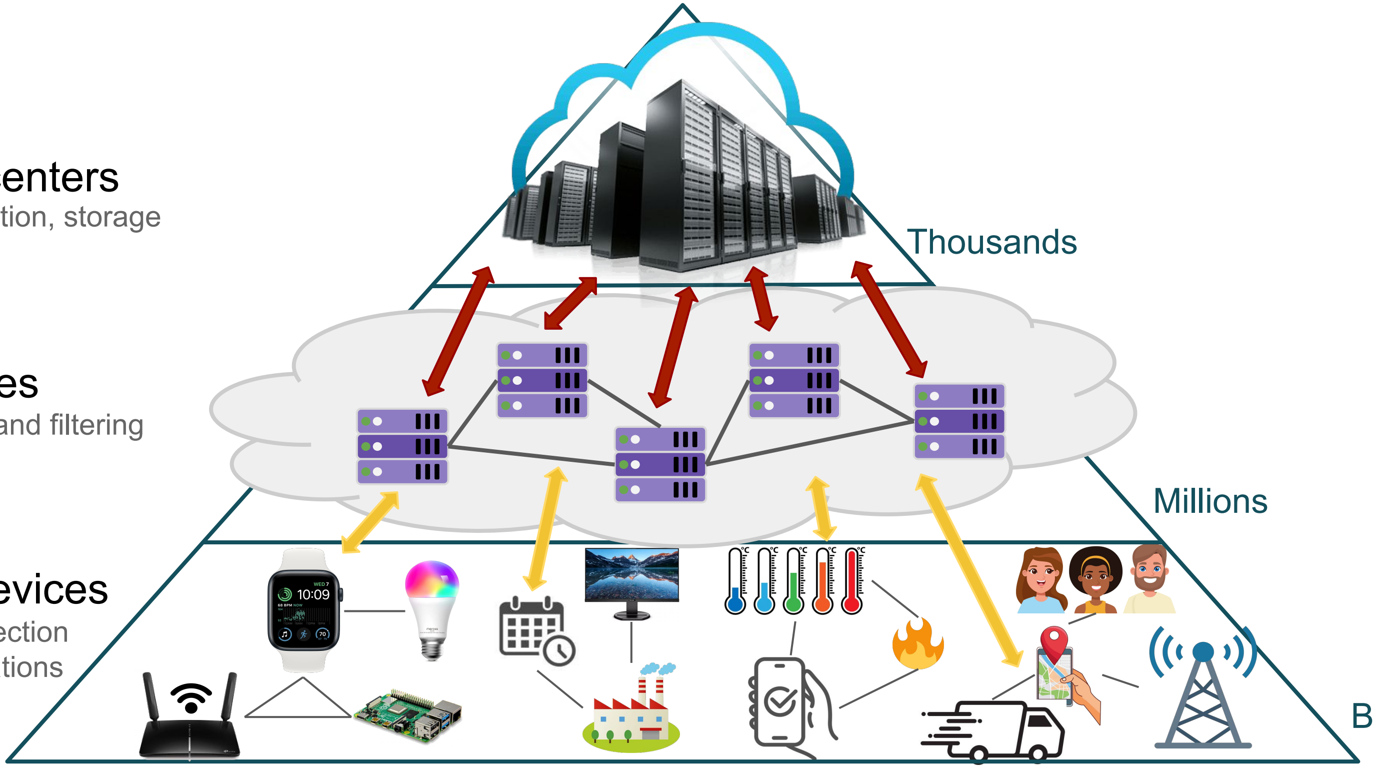
LS2N

ANR
PROJET FINANCÉ PAR L'ANR - PROJECT FUNDED BY THE ANR

Outline

1. SeMaFoR and Postdoc subject
2. Background about reconfiguration
3. CP model for local problem
4. Communication protocol for global problem
5. Integration of the solution
6. Use cases and performances
7. Concluding remarks

Context: Fog Architectures



Cloud data centers
Non-real-time data action, storage

Thousands

Fog nodes
Local data analysis and filtering

Millions

Edge / IoT devices
Local data collection
+ small calculations

Billions

“The Fog extends the Cloud to be closer to the thing that produce and act on IoT data” [Cisco, mar. 2015]

SeMaFoR Project

Problem

- How to administrate a Fog infrastructure?
(size, reliability, dynamic, heterogeneous,...)

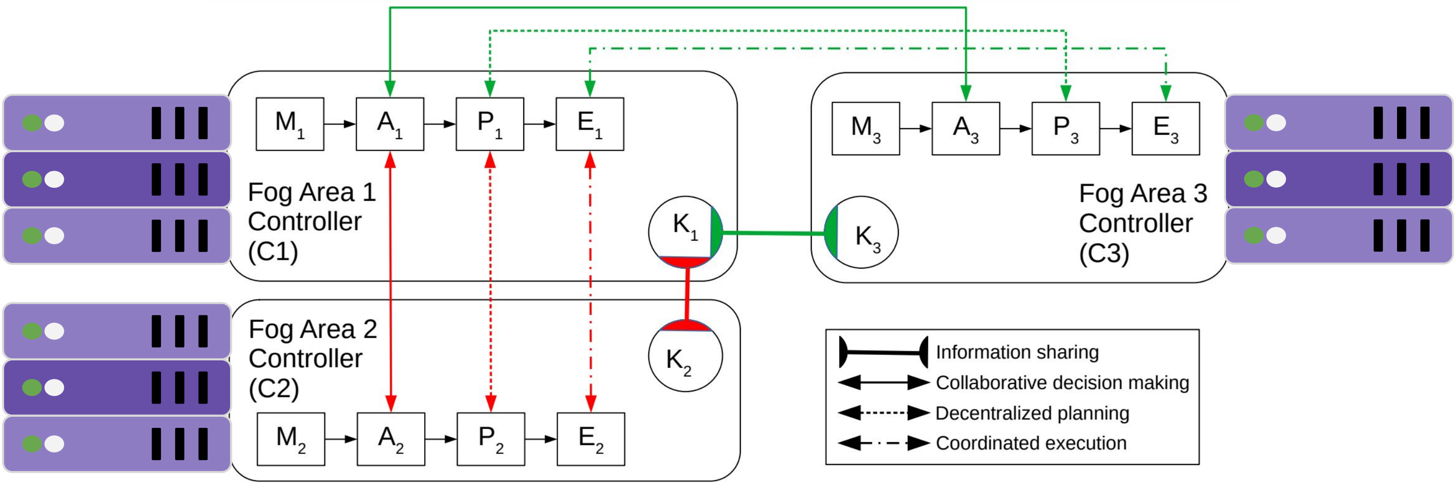
Objectives [SeMaFoR, 2023]

- Designing and developing a decentralized, generic solution for self-administration of resources.
- Coordinate a fleet of autonomous controllers in a distributed manner, with each controller having a local view of its resources.

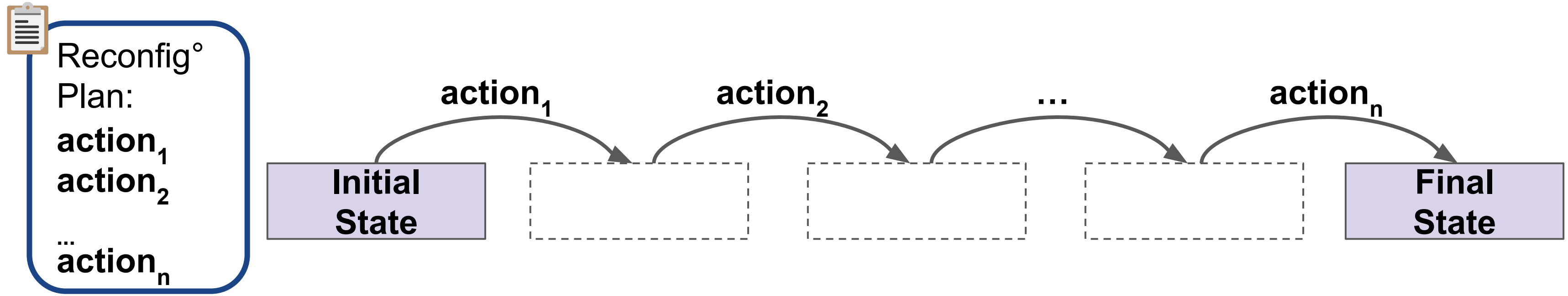
SeMaFoR proposal for controller coordination

MAPE-K [IBM, 2006]**:** ***Coordinated Control Pattern*** model

- **M**onitor its state and the state of the environment 👁️
- **A**nalyze to decide which state to reach 🎯
- <u>**P**lan the reconfiguration</u> 📋
- **E**xecute the reconfiguration to reach the new state ⚙️
  ○ **K**nowledge that is common, to take a decision



Decentralized reconfiguration

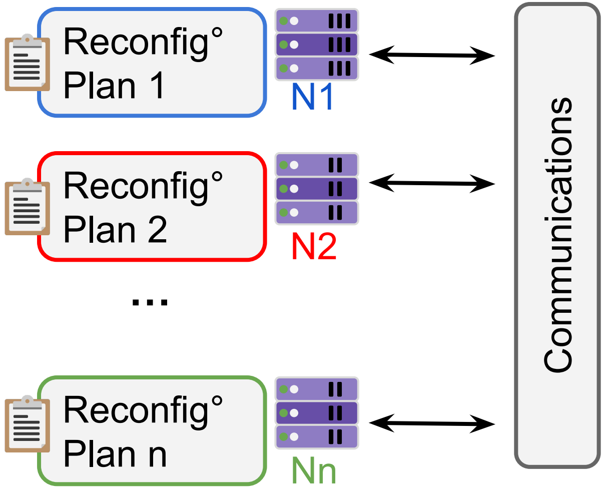


Postdoc objectives:

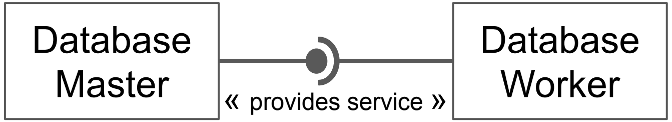
- ➢ Infer reconfiguration actions (MAPE-K)
- ➢ Optimal overall reconfiguration

Challenges:

- ■ Locally: partial view of the system
- ■ Collaboration with other nodes



Deployment example



Machine 1:

Database Master (DM)

1. Initiate
2. Configure
3. Prepare the service
4. Start the service

Machine 2:

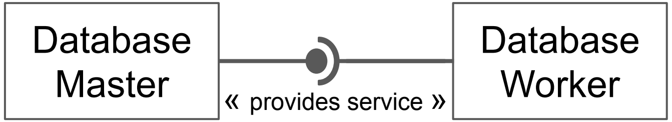
Database Worker (DW)

1. Initiate
2. Configure
3. Prepare the service
4. Register the service
5. Start the service

Component granularity: DM $\ll$ DW
Lifecycle granularity: DM(4) $\ll$ DW(4)

Reconfiguration example: Update DM

Goal: must update
+ end at running state



Goal: end at running state

Machine 1:

Database Master (DM)

1. Interrupt the service
2. Update
3. Prepare the service
4. Start the service

Machine 2:

Database Worker (DW)

1. Interrupt the service
2. Unregister and update
3. Prepare the service
4. Register the service
5. Start the service

Lifecycle granularity: DW(2) ≪ DM(1), DM(4) ≪ DW(4)

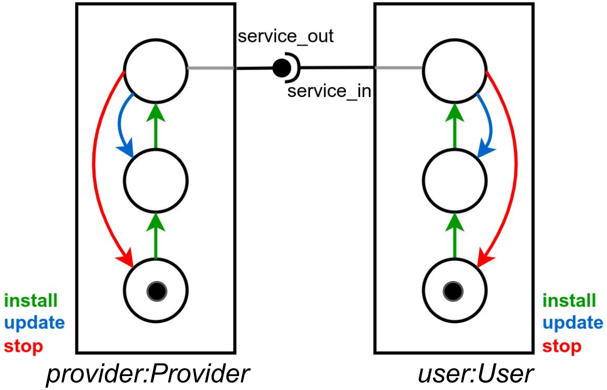
Objective

- Formalize reconfiguration goals
- Inference of local behaviors
  - No deadlock
  - Target a reconfiguration engine
- Decentralized planning
  - Constraint spreading
  - Communication protocol

Concerto

Concerto: A reconfiguration <u>language</u> for distributed systems

- Involved components and their life-cycle
- Interactions / connections between components
- Changes in the component



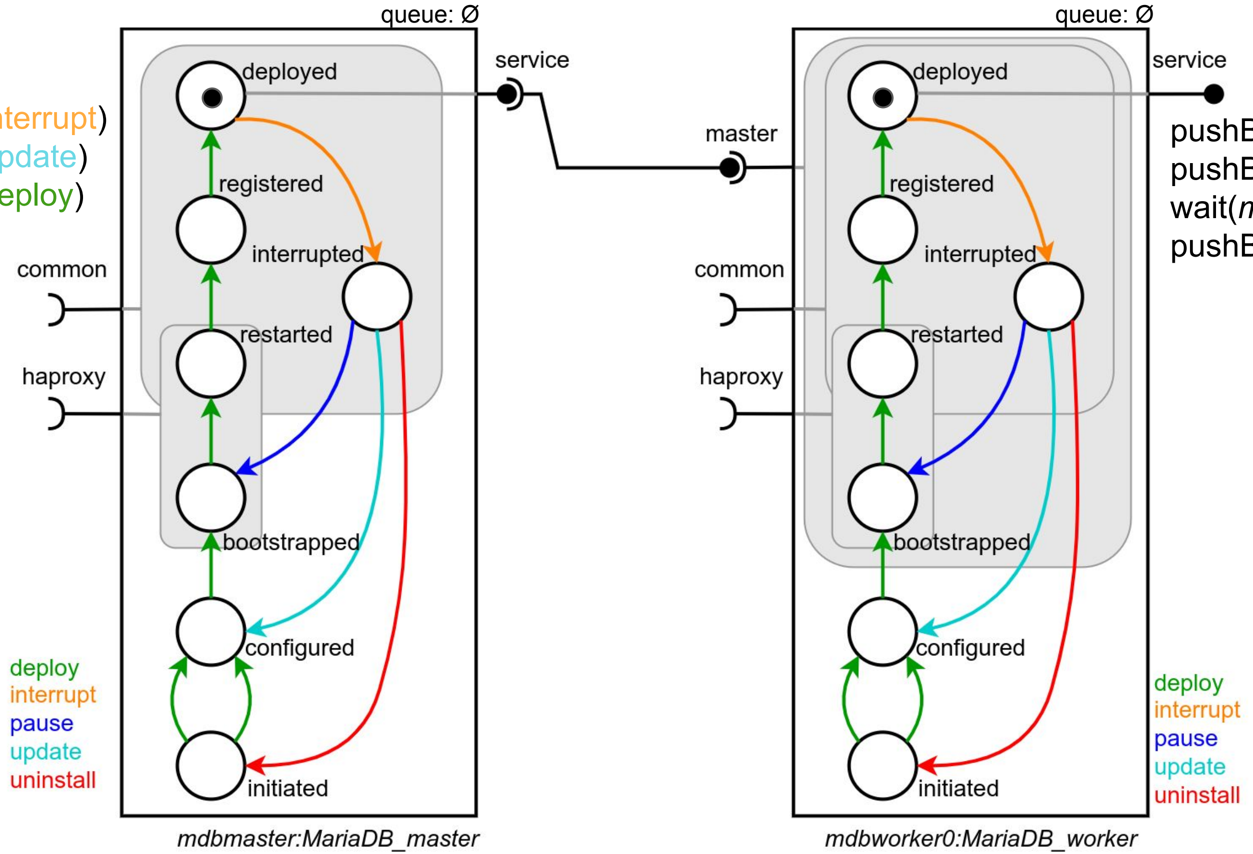
```
add(provider, Provider)
add(user, User)
connect(provider, service_out,
        user, service_in)
pushB(provider, install)
pushB(user, install)
wait(user, install)
```

2 levels of concurrency:

- **Within component**
- **Between components**

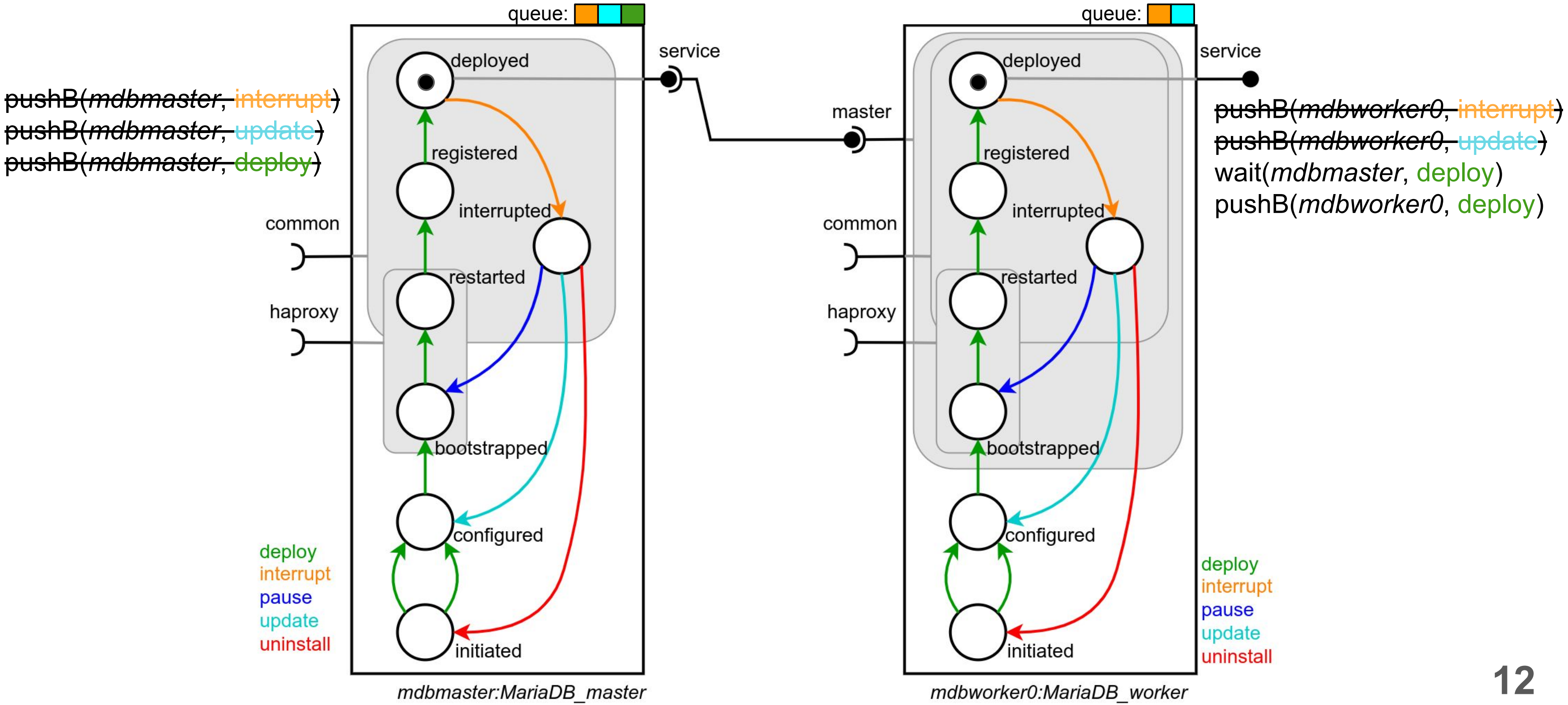
Concurrent execution

pushB(*mdbmaster*, interrupt)
pushB(*mdbmaster*, update)
pushB(*mdbmaster*, deploy)



pushB(*mdbworker0*, interrupt)
pushB(*mdbworker0*, update)
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution

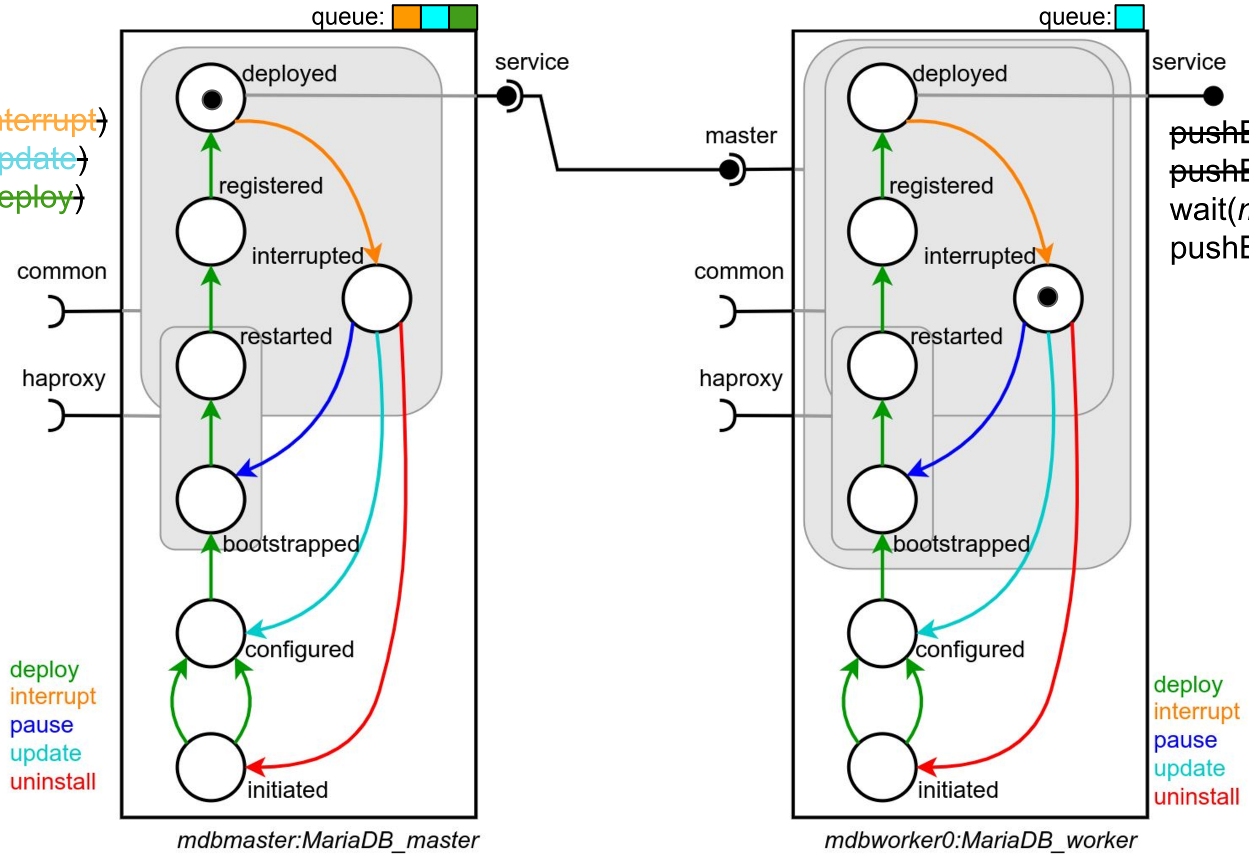


~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



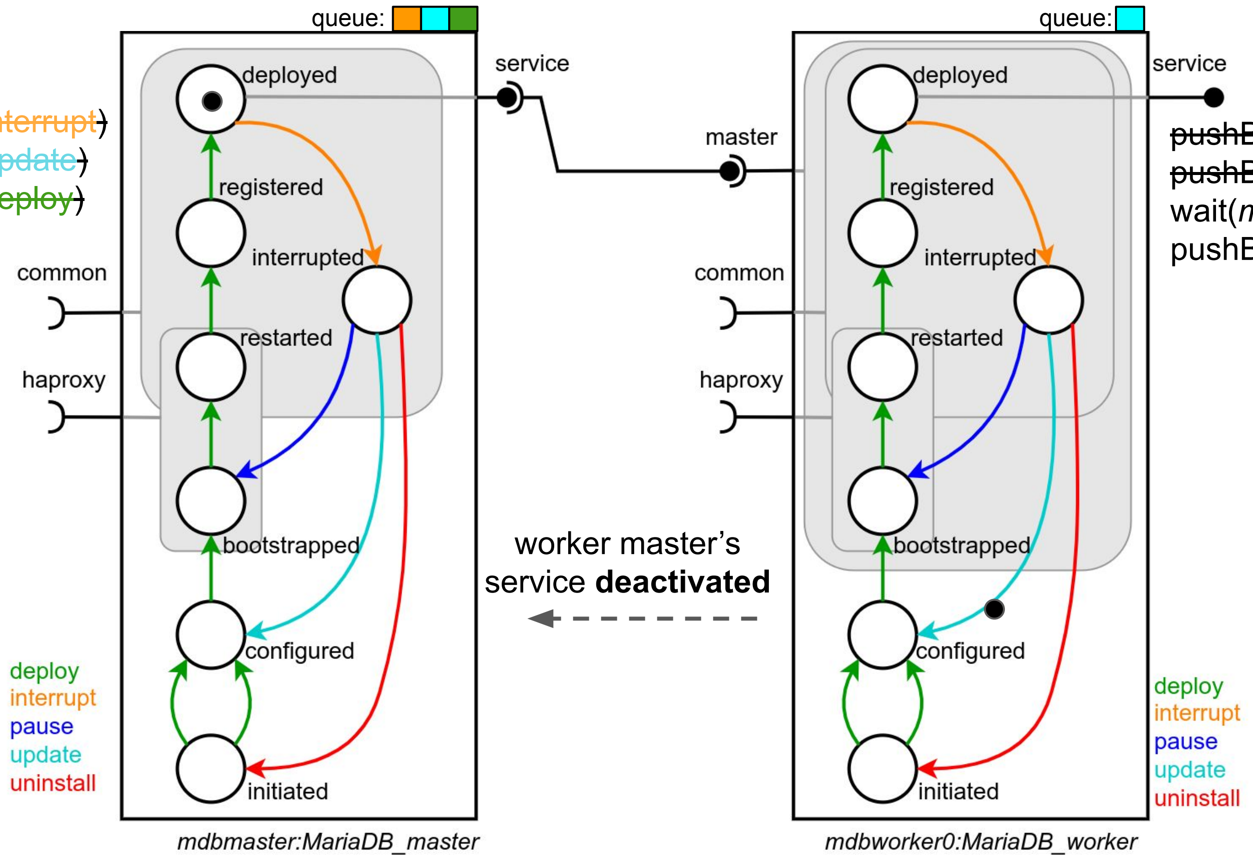
~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~

~~pushB(*mdbmaster*, update)~~

~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~

~~pushB(*mdbworker0*, update)~~

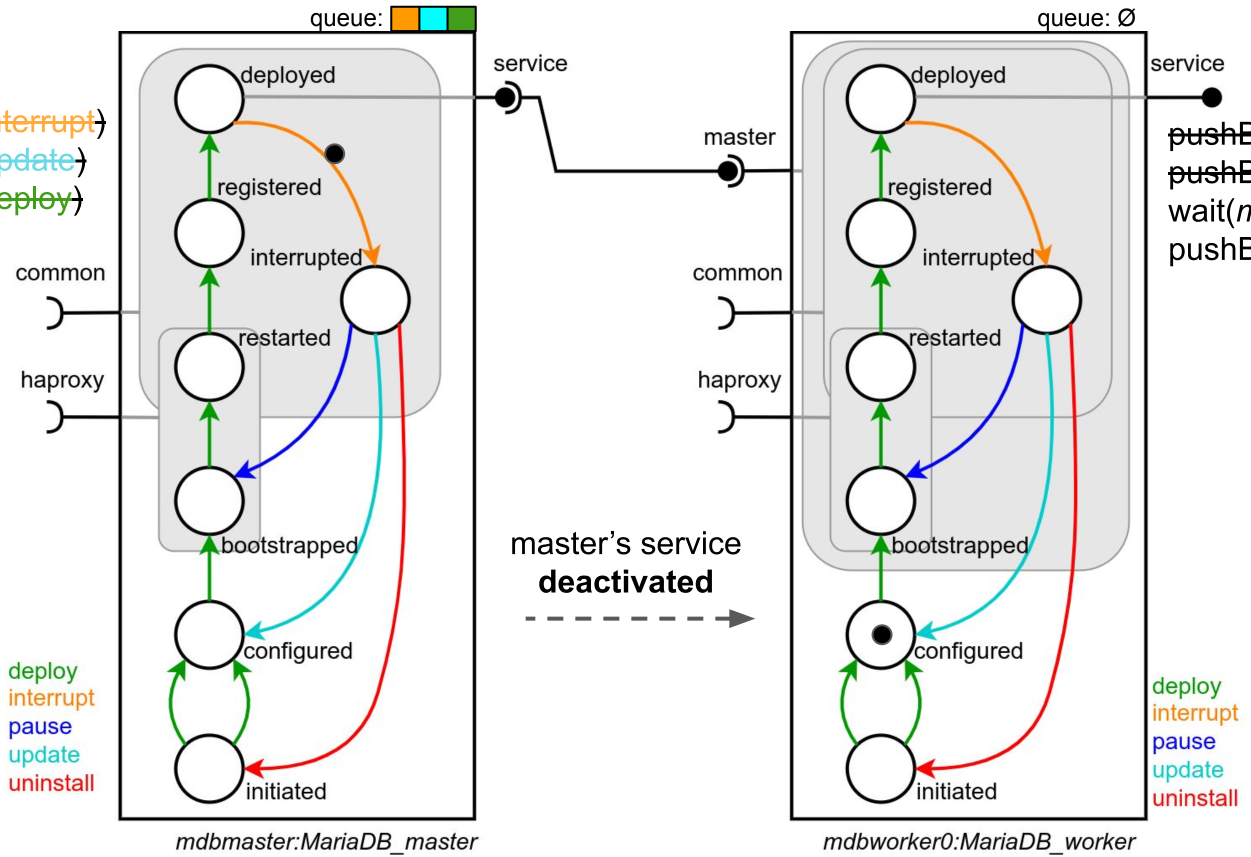
wait(*mdbmaster*, deploy)

pushB(*mdbworker0*, deploy)

worker master's service **deactivated**

Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

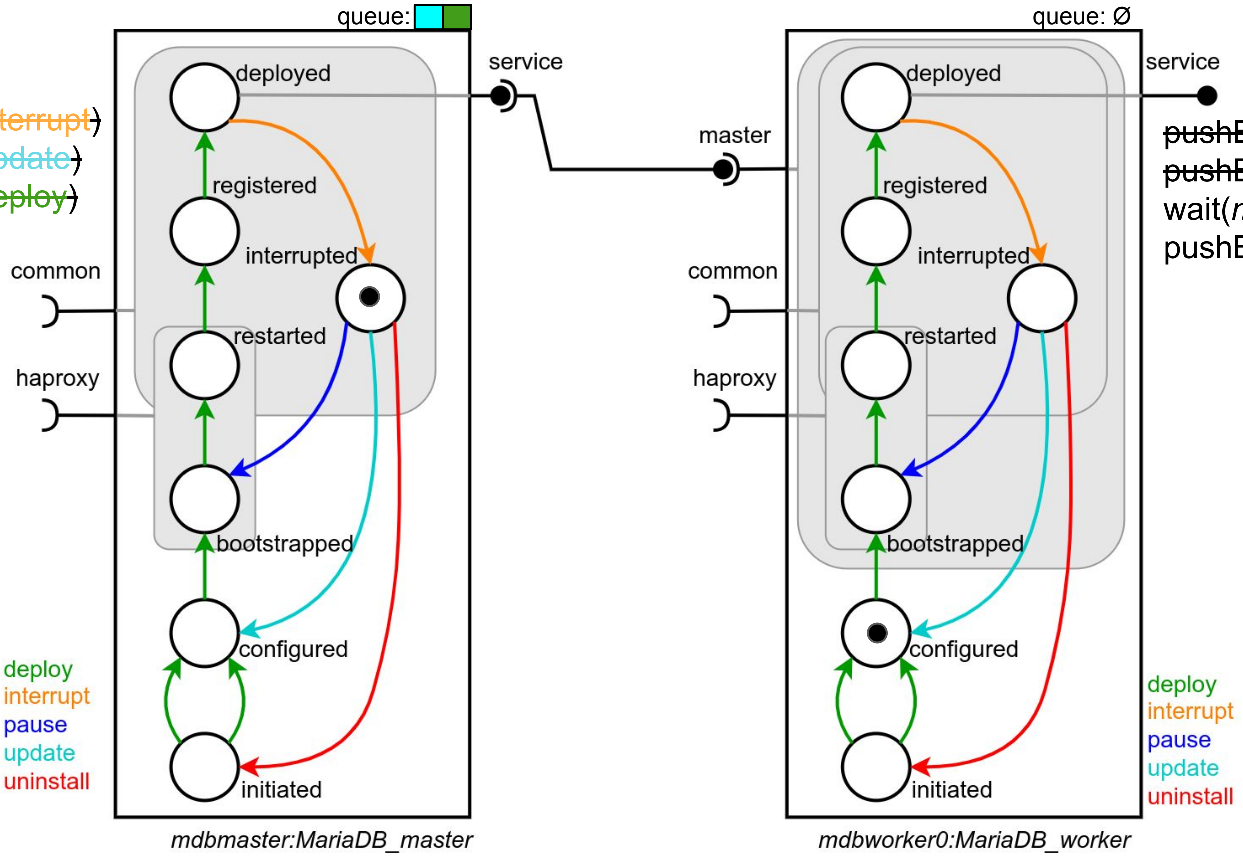


master's service **deactivated**

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

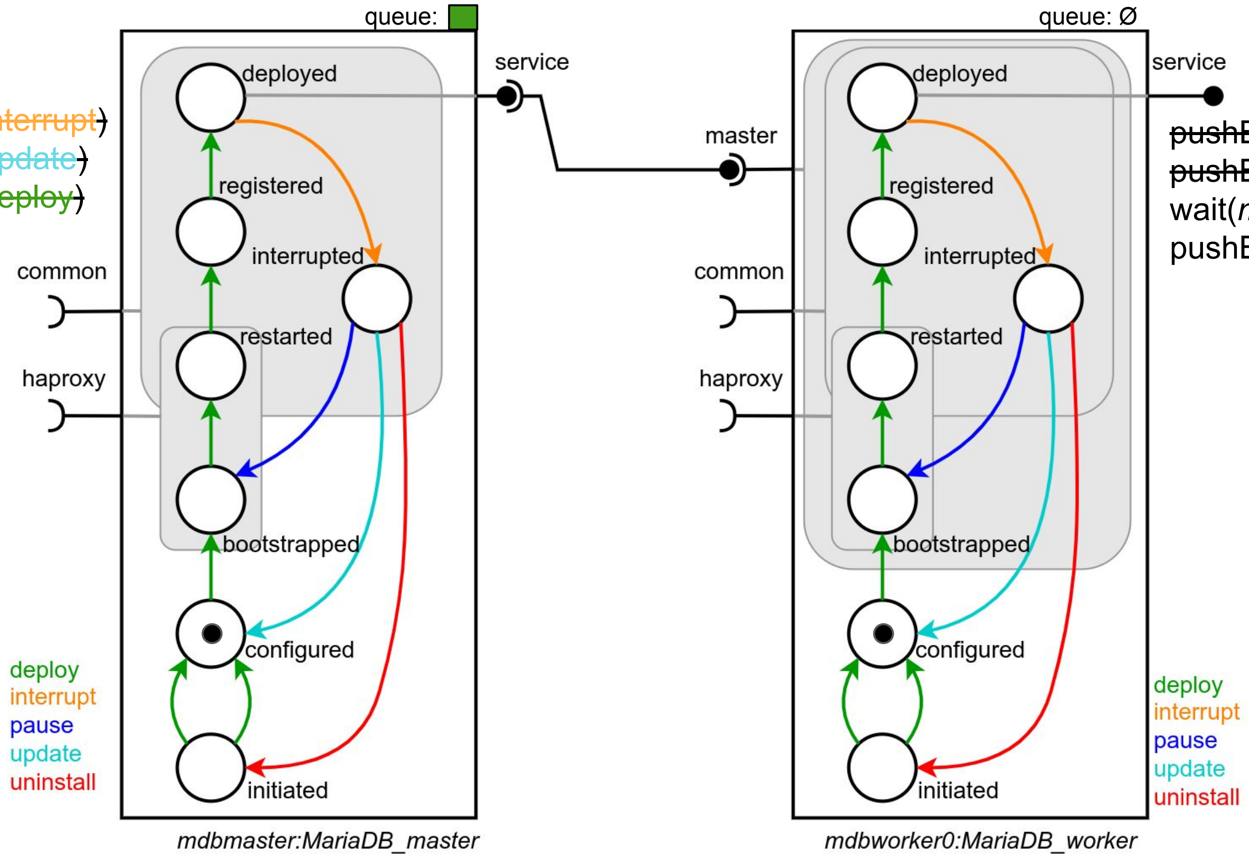
Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution

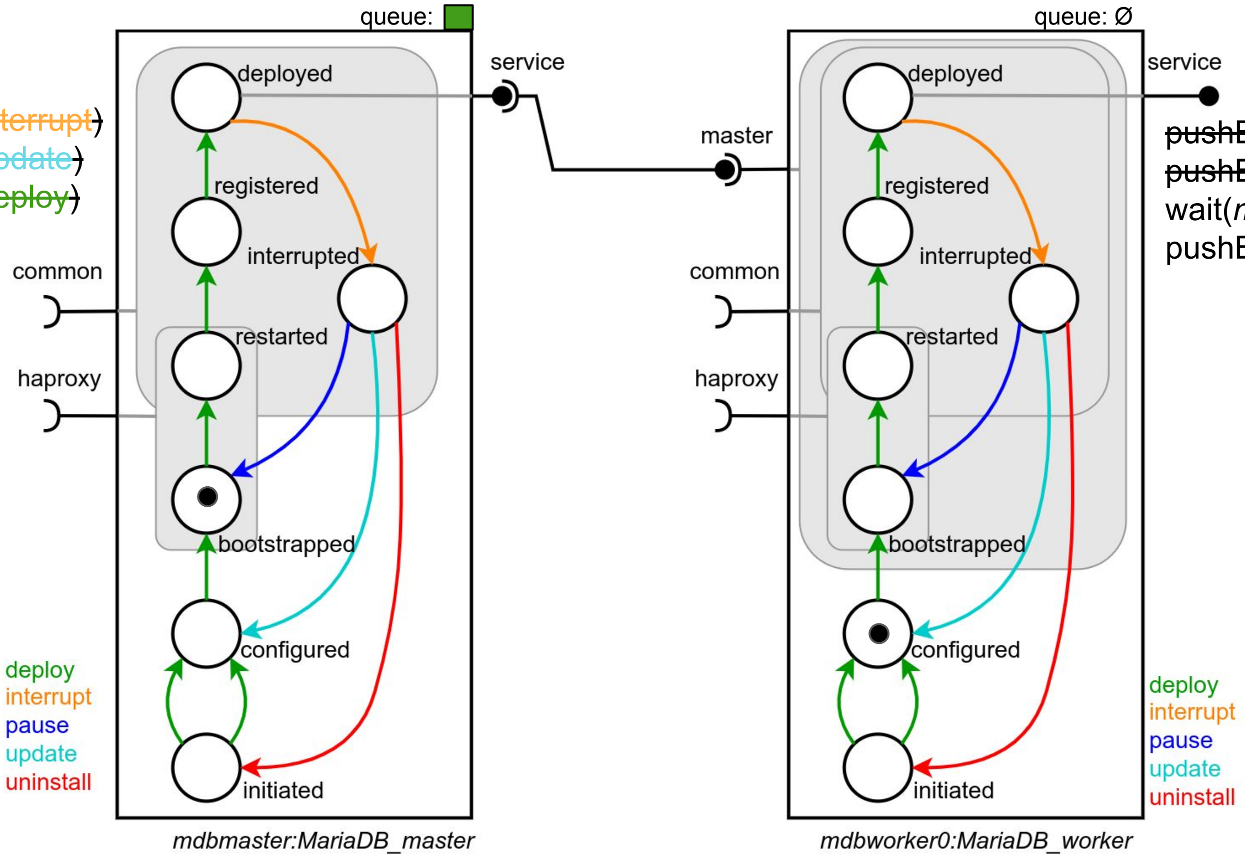


~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution

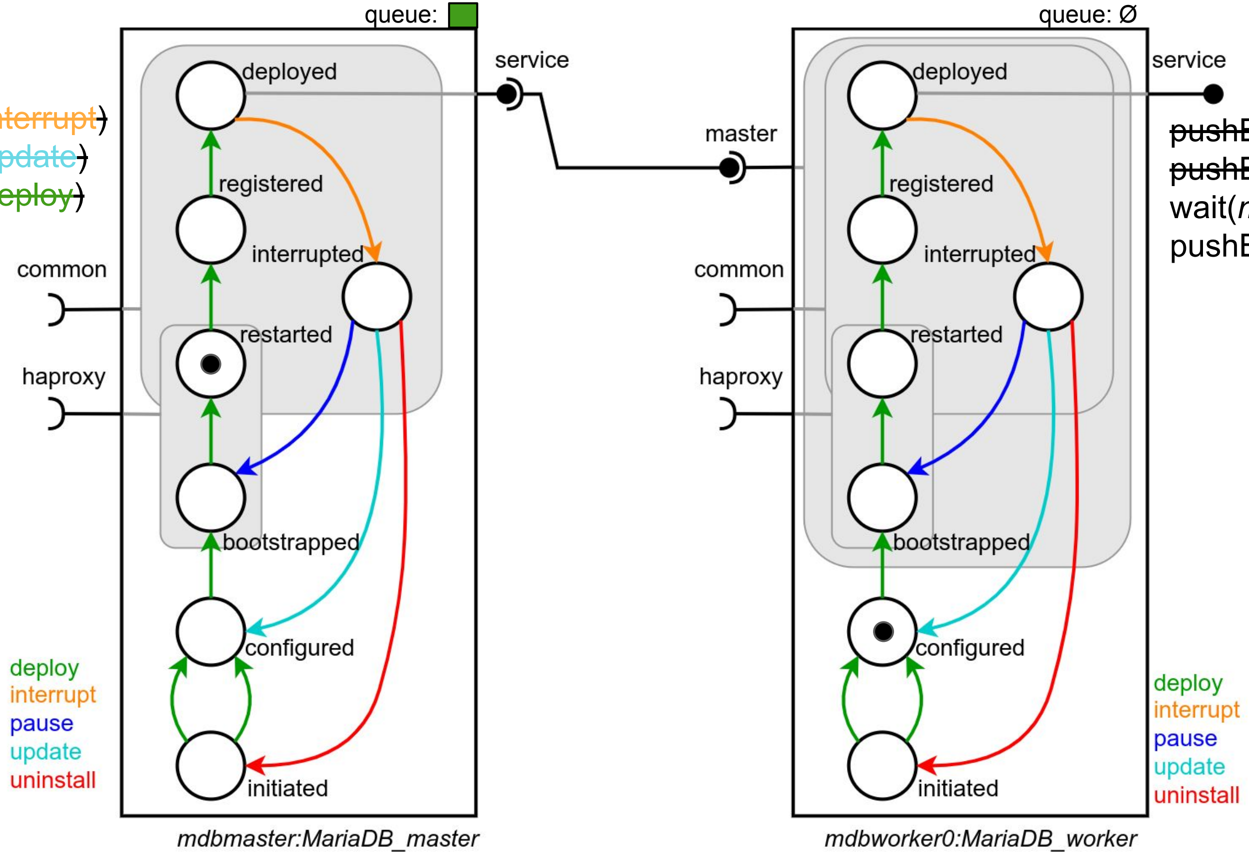
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution

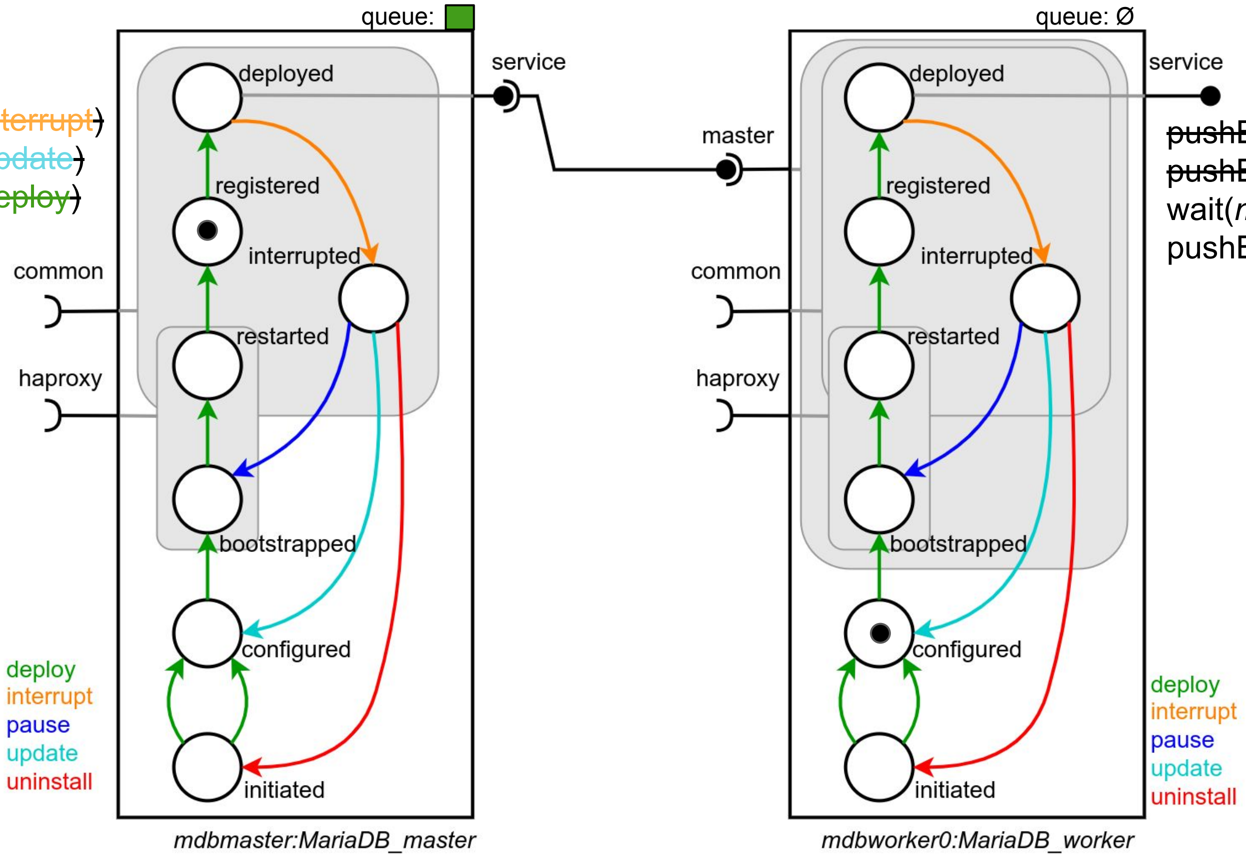
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

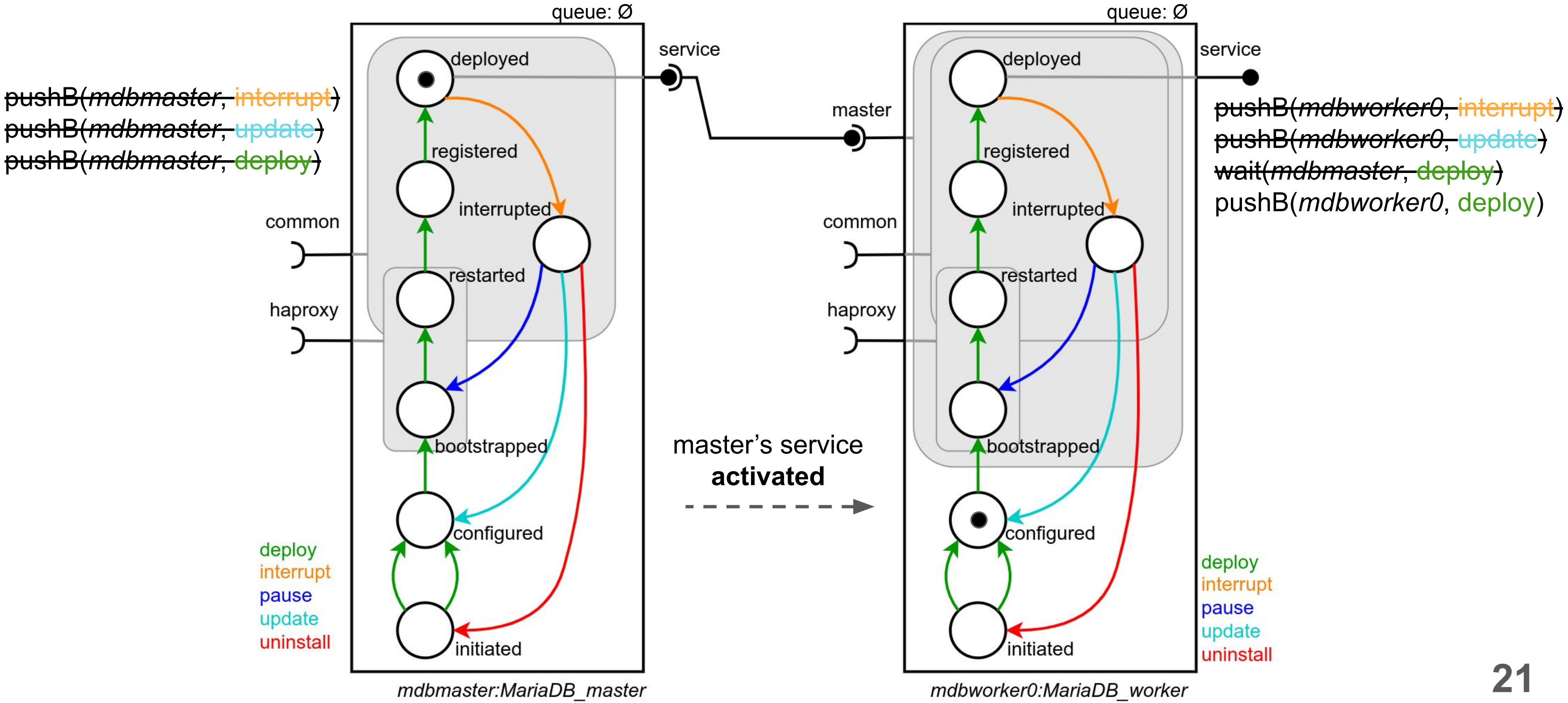
Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
wait(*mdbmaster*, deploy)
pushB(*mdbworker0*, deploy)

Concurrent execution



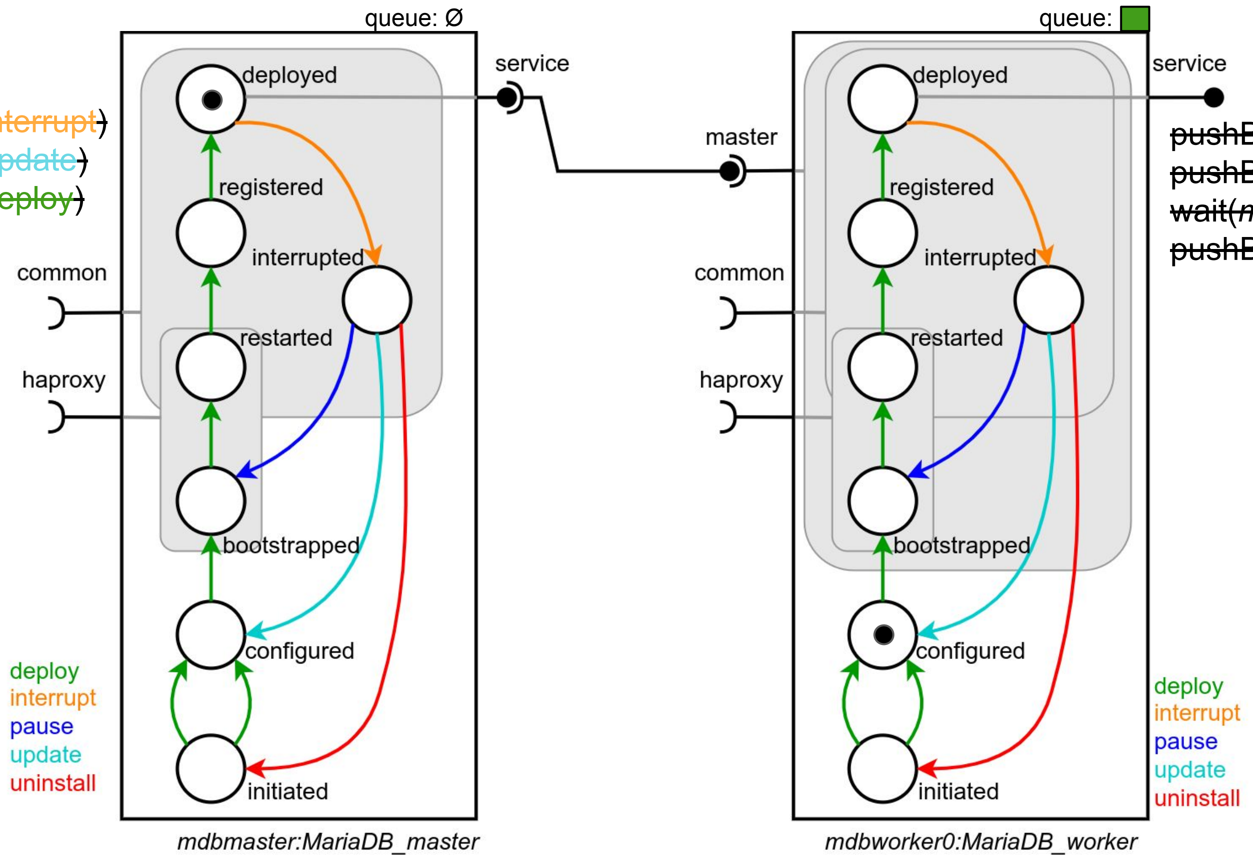
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
pushB(*mdbworker0*, deploy)

master's service **activated**

Concurrent execution

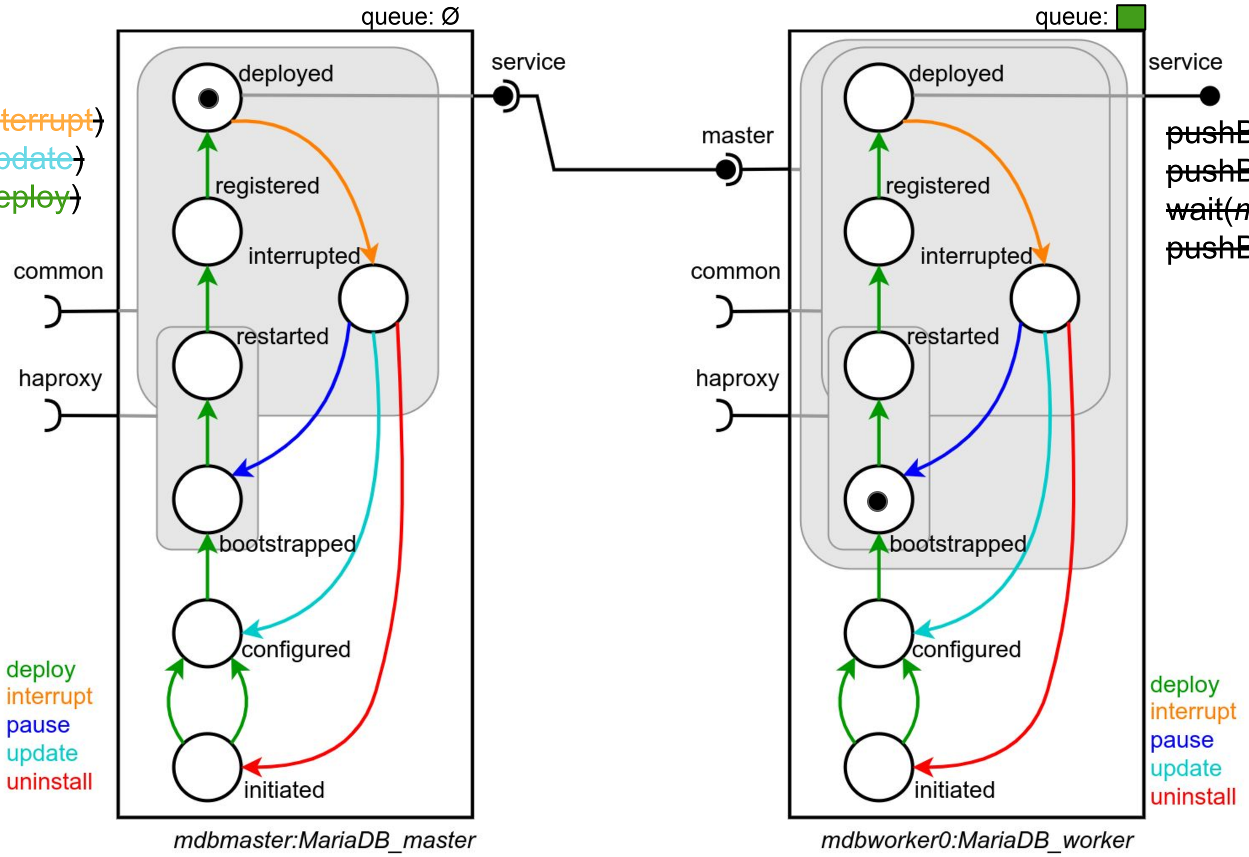
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
~~pushB(*mdbworker0*, deploy)~~

Concurrent execution

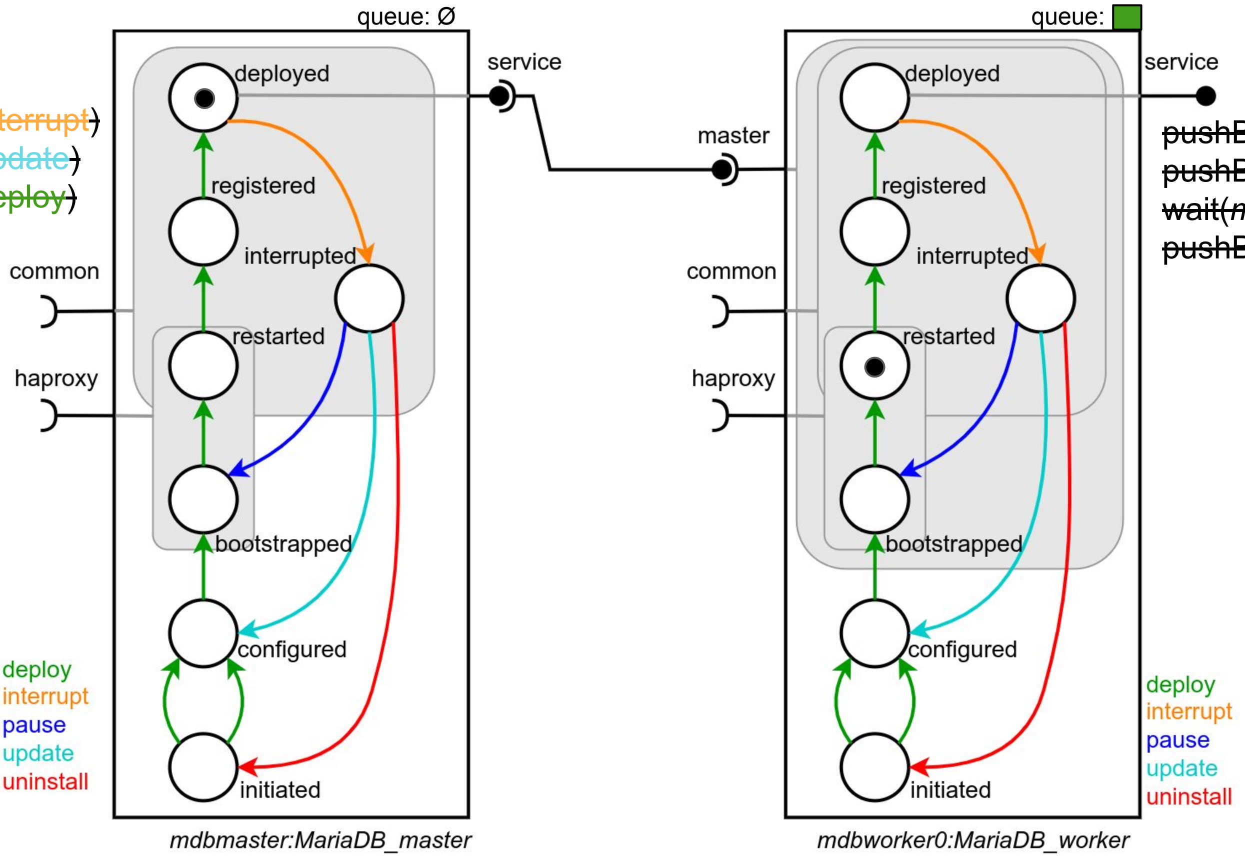
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
~~pushB(*mdbworker0*, deploy)~~

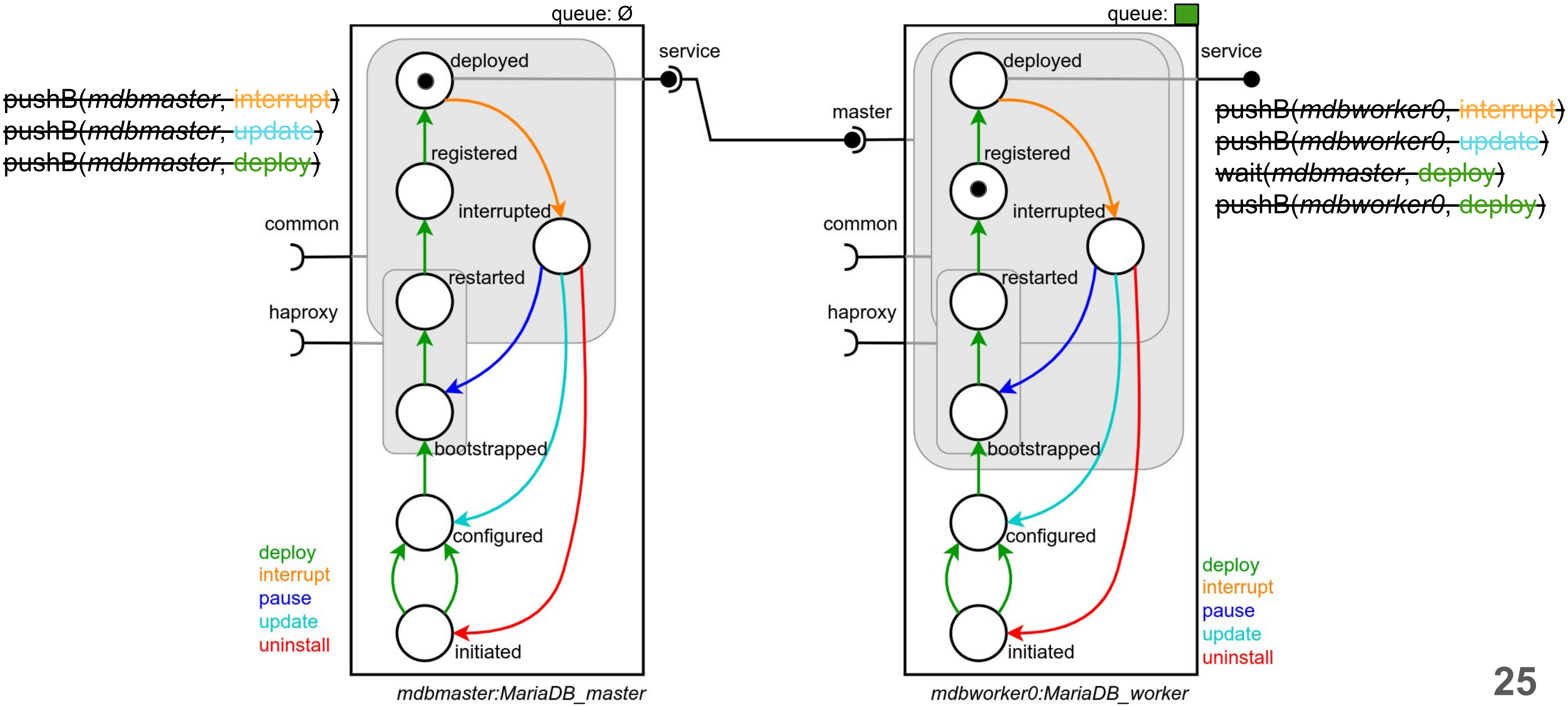
Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
~~pushB(*mdbworker0*, deploy)~~

Concurrent execution



~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

queue: ∅

mdbmaster:MariaDB_master

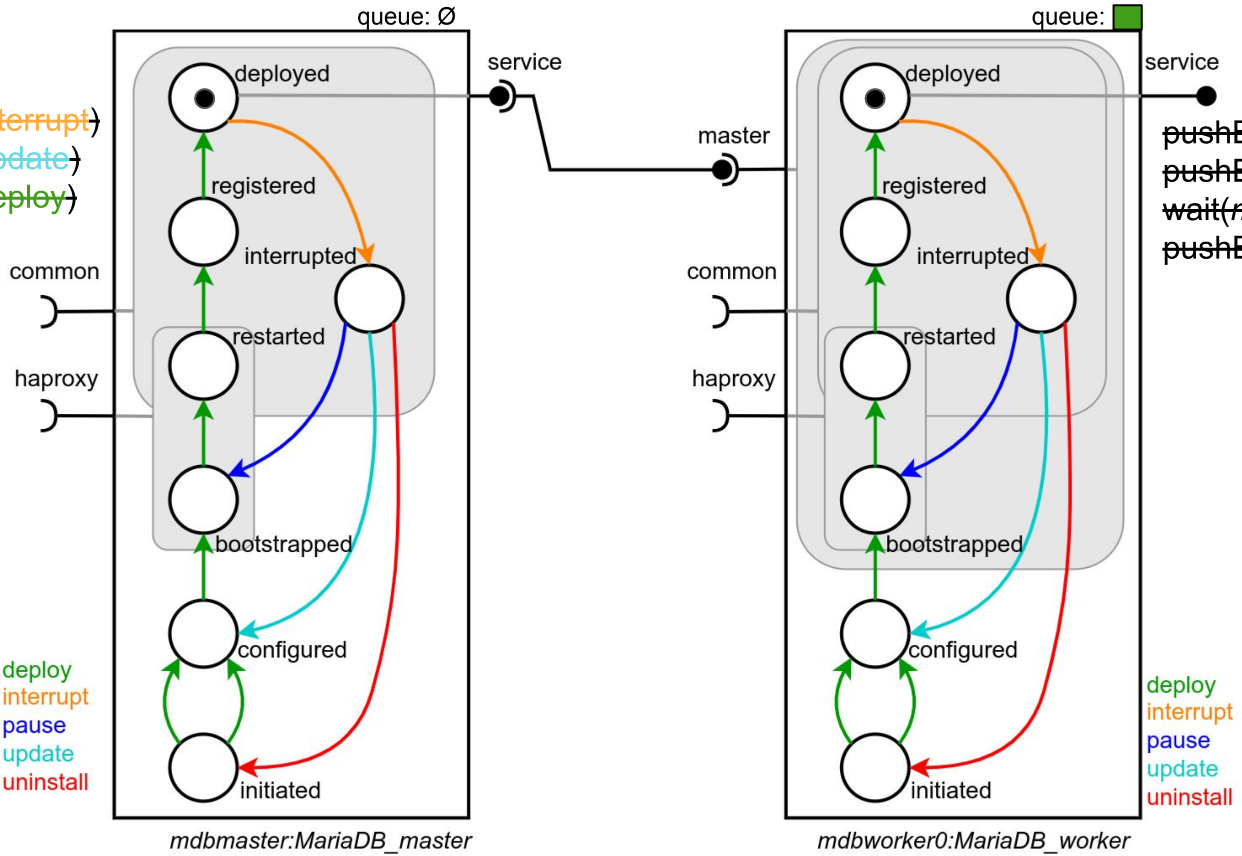
queue: 🟩

mdbworker0:MariaDB_worker

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
~~pushB(*mdbworker0*, deploy)~~

Concurrent execution

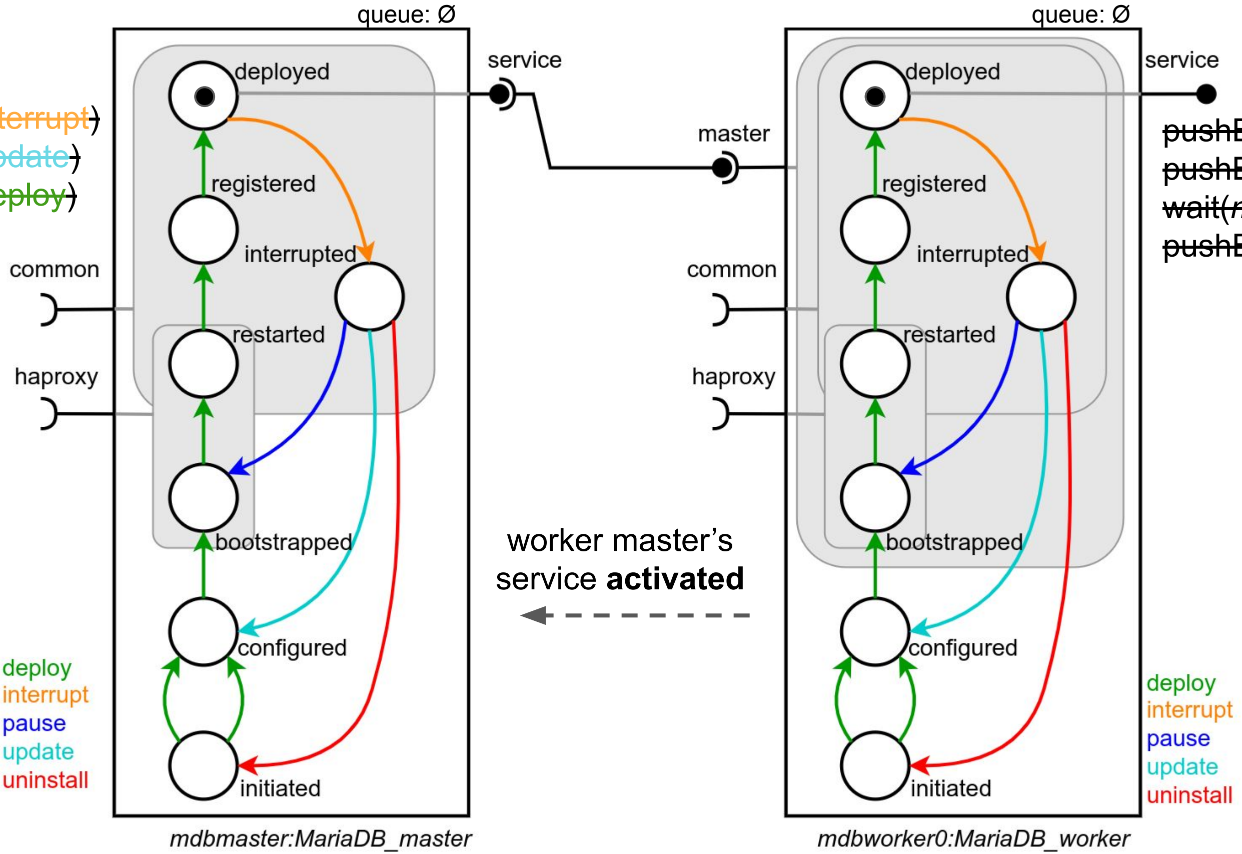
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
~~pushB(*mdbworker0*, deploy)~~

Concurrent execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

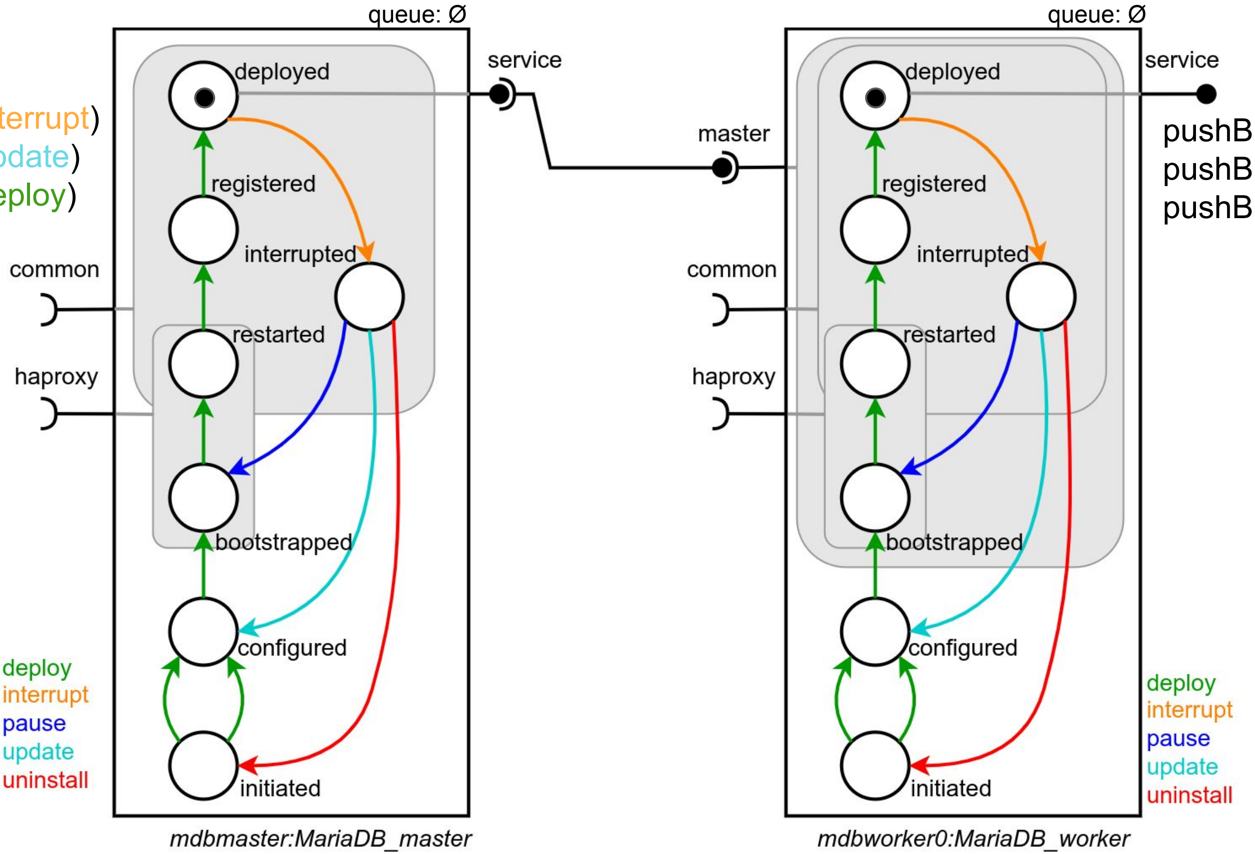


worker master’s
service **activated**

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~wait(*mdbmaster*, deploy)~~
~~pushB(*mdbworker0*, deploy)~~

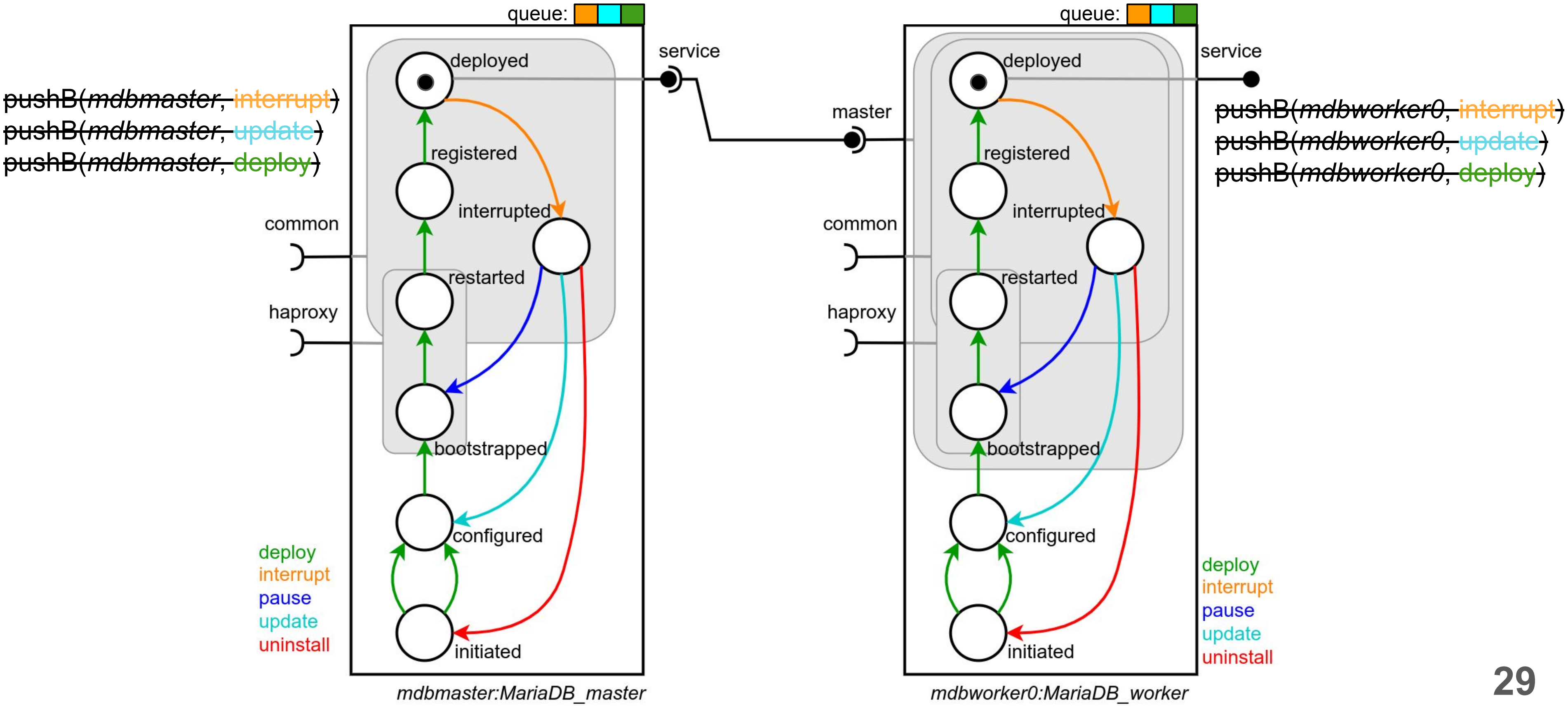
Failing execution

pushB(*mdbmaster*, interrupt)
pushB(*mdbmaster*, update)
pushB(*mdbmaster*, deploy)



pushB(*mdbworker0*, interrupt)
pushB(*mdbworker0*, update)
pushB(*mdbworker0*, deploy)

Failing execution

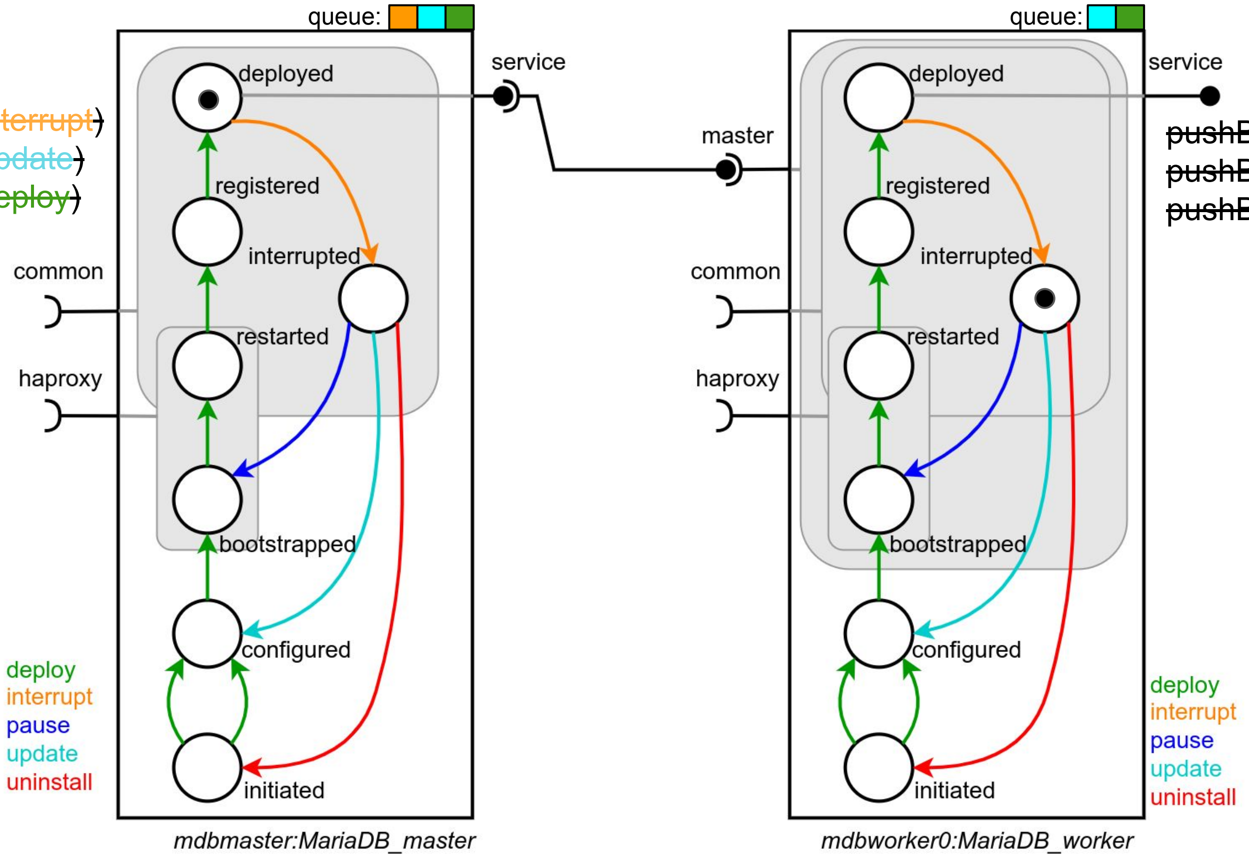


~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~pushB(*mdbworker0*, deploy)~~

Failing execution

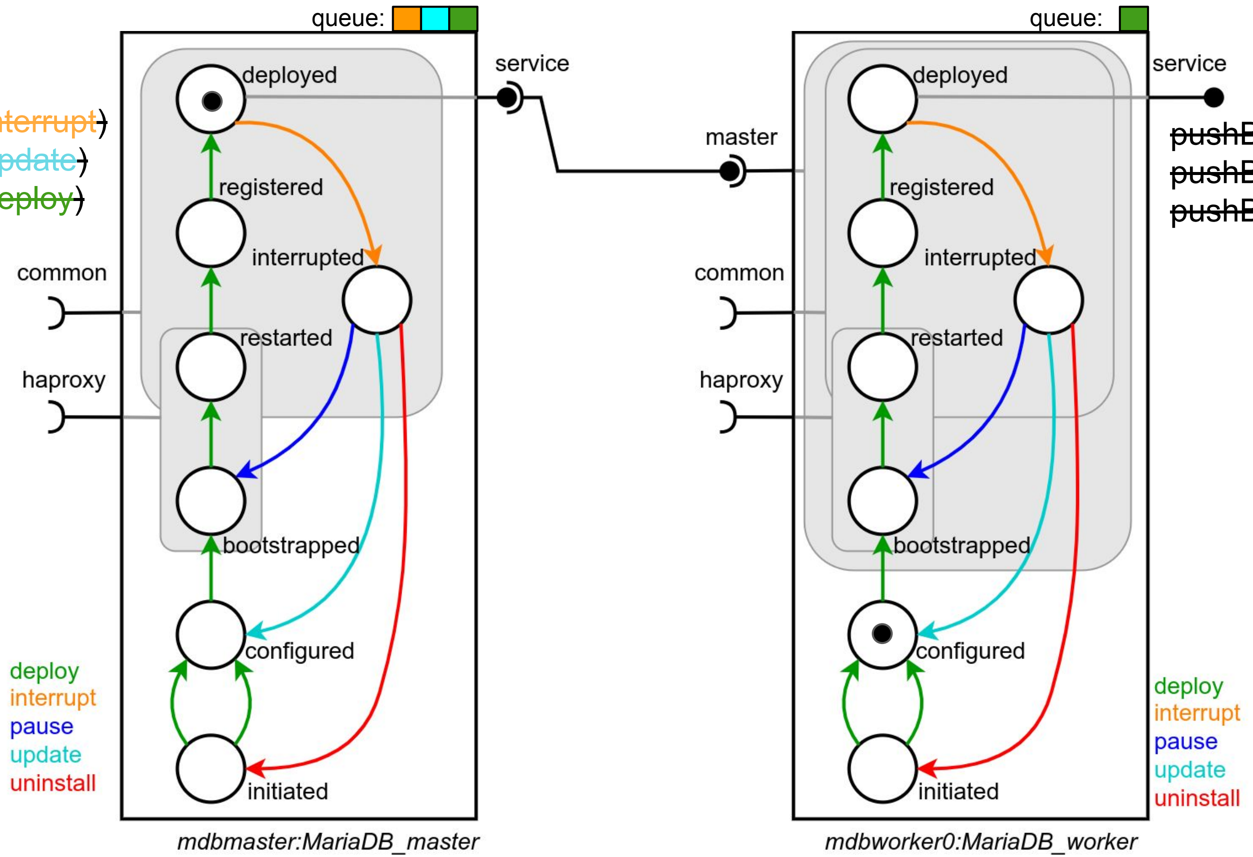
~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~pushB(*mdbworker0*, deploy)~~

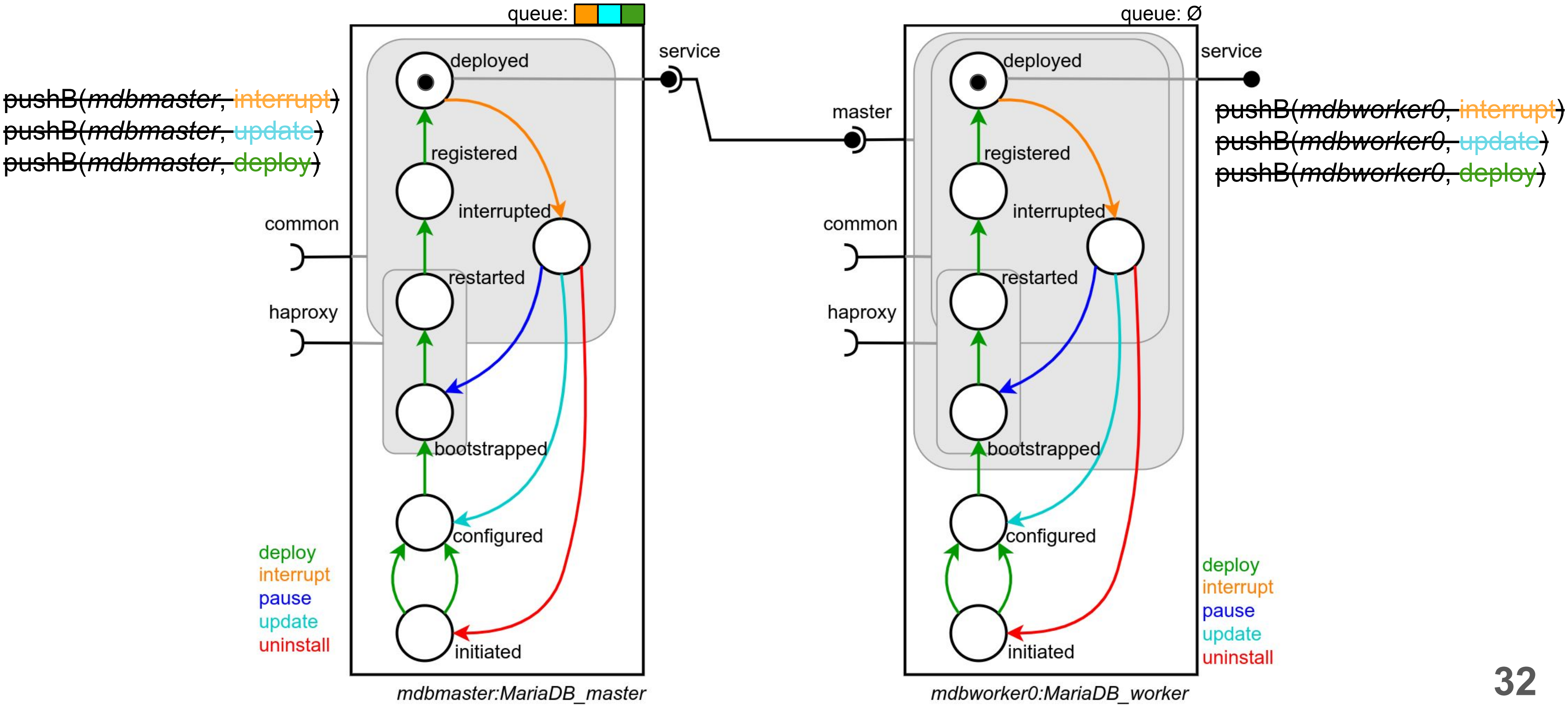
Failing execution

~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~



~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~pushB(*mdbworker0*, deploy)~~

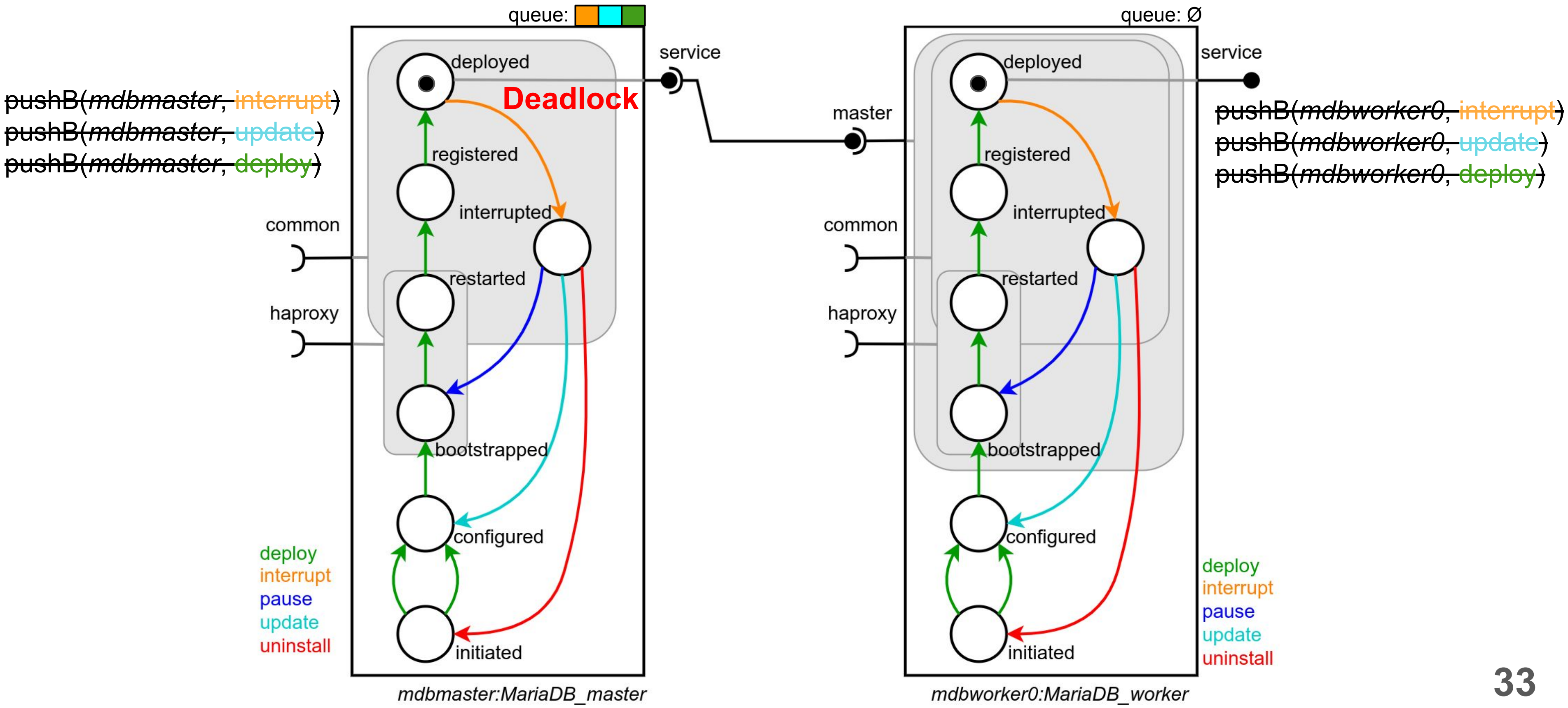
Failing execution



~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~pushB(*mdbworker0*, deploy)~~

Failing execution



~~pushB(*mdbmaster*, interrupt)~~
~~pushB(*mdbmaster*, update)~~
~~pushB(*mdbmaster*, deploy)~~

~~pushB(*mdbworker0*, interrupt)~~
~~pushB(*mdbworker0*, update)~~
~~pushB(*mdbworker0*, deploy)~~

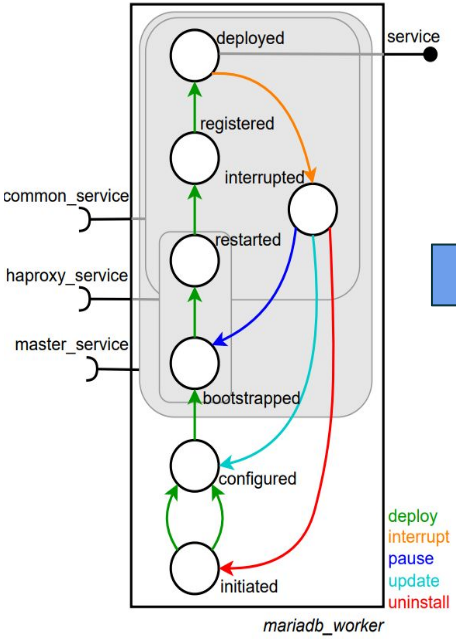
Global idea

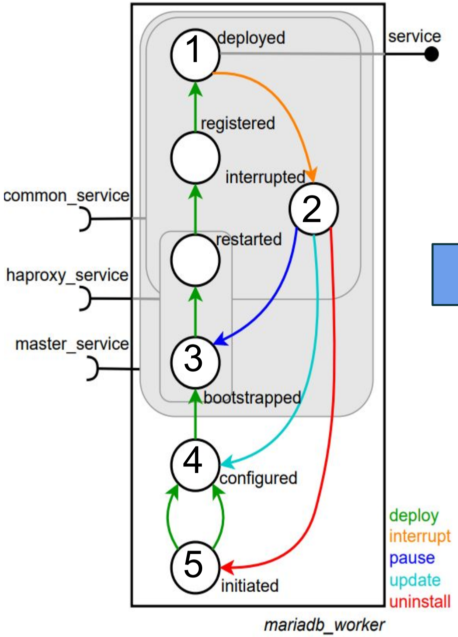
1. Model component life-cycle as automata

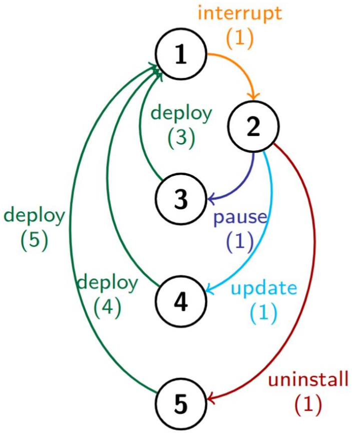
2. Find a word on the automata
3. Constraint the word to match reconfiguration goals
   a. Behavior
   b. Port status
   c. Reached state
4. Rumor-spreading approach for constraint propagation
   a. Adding constraint for synchronization needs

Model for local reconfiguration

- **Input:** Life-cycle of component,
  its status (running, initiated)
- **Output:** A set of behaviors



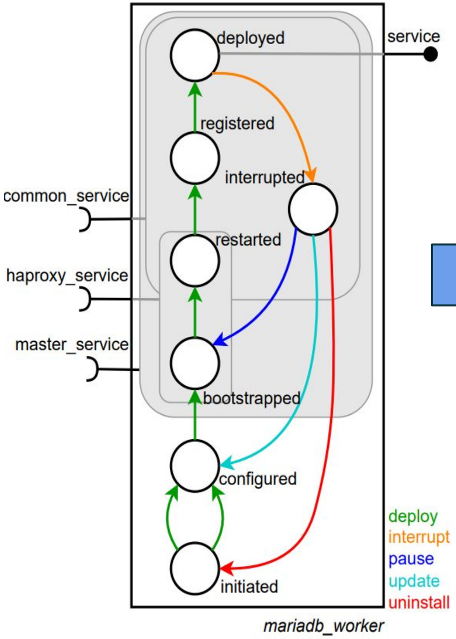


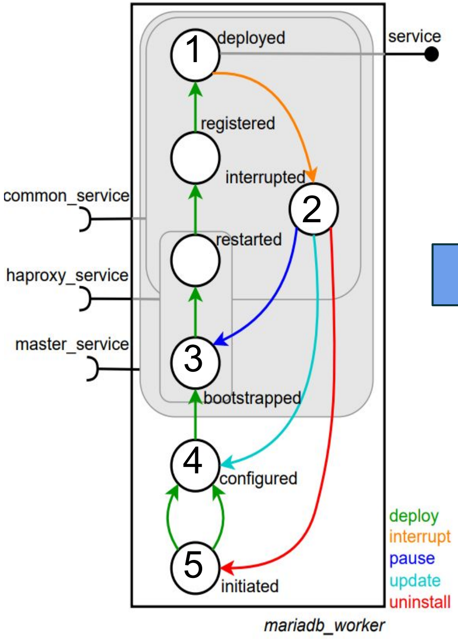


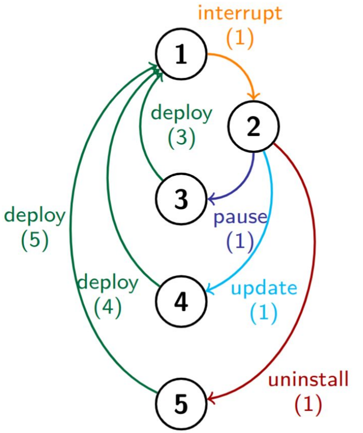
state	1	2	3	4	5
common_service	✓	✓	×	×	×
haproxy_service	×	×	✓	×	×
master_service	✓	✓	✓	×	×
service	✓	×	×	×	×

Model for local reconfiguration

- **Find a word, is equivalent to find a sequence**

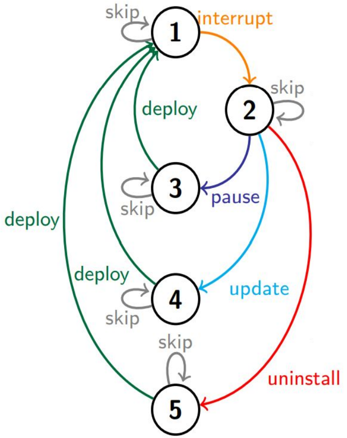






state	1	2	3	4	5
common_service	✓	✓	×	×	×
haproxy_service	×	×	✓	×	×
master_service	✓	✓	✓	×	×
service	✓	×	×	×	×

Local goal: Model CP



state	1	2	3	4	5
common_service	✓	✓	×	×	×
haproxy_service	×	×	✓	×	×
master_service	✓	✓	✓	×	×
service	✓	×	×	×	×

Minimize C subject to

$$\textsc{Regular}(B, \Pi, s_{init}, S_{goal}) \tag{1}$$

$$s_{i+1} = inc_{\Pi}[s_i][b_i], \ \forall i \in 1..n \tag{2}$$

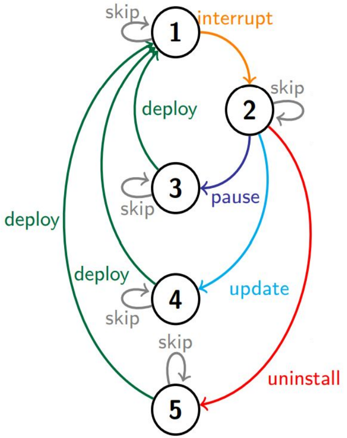
$$\textsc{Count}(b, B, >, 0) \tag{3}$$

$$status(p, s_{n+1}) = \Gamma_p \tag{4}$$

$$\text{where} \quad \begin{aligned} &\Gamma_p \in \{\texttt{active}, \texttt{inactive}\} \\ &c_i = cost(s_i, b_i), \ \forall i \in 1..n \\ &C = \textsc{Sum}([c_i \mid i \in 1..n]) \end{aligned}$$

- a word in the automaton
- a record of states
- a port tracking
- a set of constraints (reconf. objectives)

Example



state	1	2	3	4	5
common_service	✓	✓	×	×	×
haproxy_service	×	×	✓	×	×
master_service	✓	✓	✓	×	×
service	✓	×	×	×	×

- Reconf. Objectives
  - ○ $\boldsymbol{s_1 = s1}$ current state
  - ○ $\boldsymbol{s_{n+1} = s1}$ targeting state
  - ○ ***Count(update, B, >, 0)*** mandatory behavior

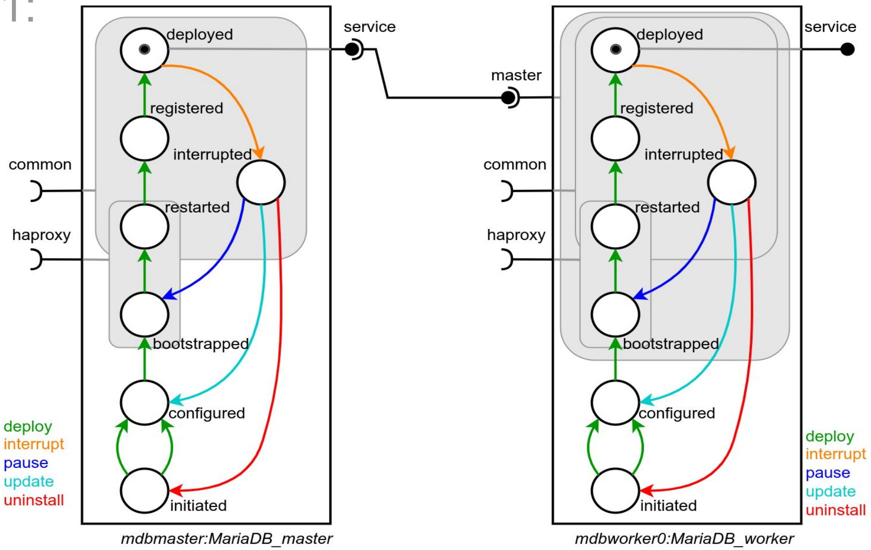
- Output:
  - ○ *B* = [interrupt, update, deploy, skip, …, skip]
  - ○ *S* = [s1, s2, s4, s1, s1, ..., s1]
  - ○ *C* = [...]

- Side-effect: [s1, s2, s4, s1, ..., s1]
  - ○ common_service: [✓ ,✓ , x, ✓ , …, ✓]
  - ○ haproxy_service: [x , x , x, x , …, x]
  - ○ master_service: [✓ ,✓ , x, ✓ , …, ✓]
  - ○ service: [✓ , x, x, ✓ , …, ✓]

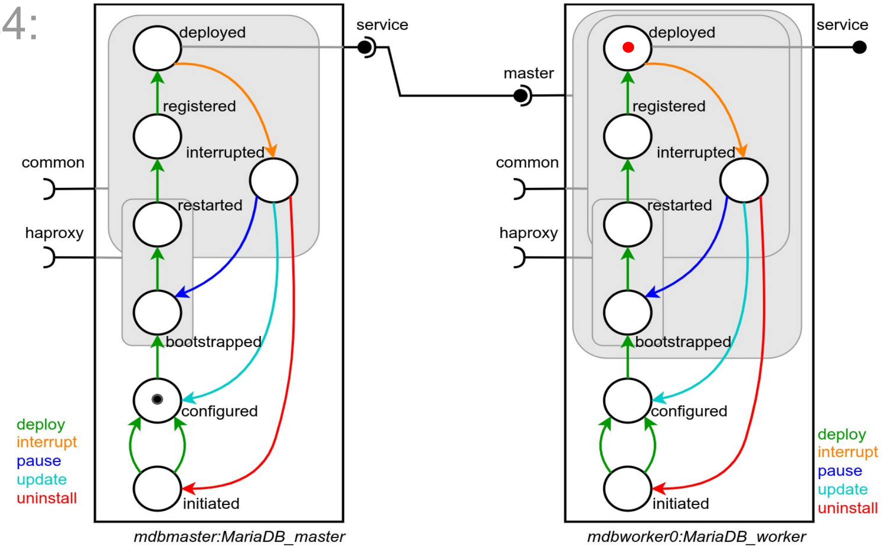
Need of constraint propagation

- Side-effect: [s1, s2, s4, s1, ..., s1]
  - common_service: [✓,✓, x, ✓, …, ✓]
  - haproxy_service: [x , x , x, x , …, x]
  - master_service: [✓,✓, x, ✓, …, ✓]
  - service: [✓, x, x, ✓, …, ✓]

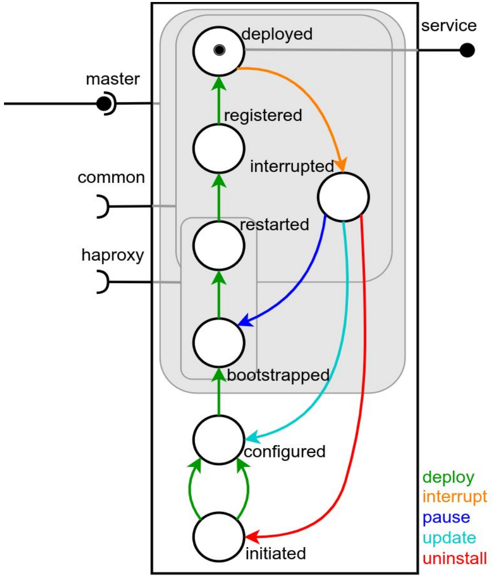
s1:



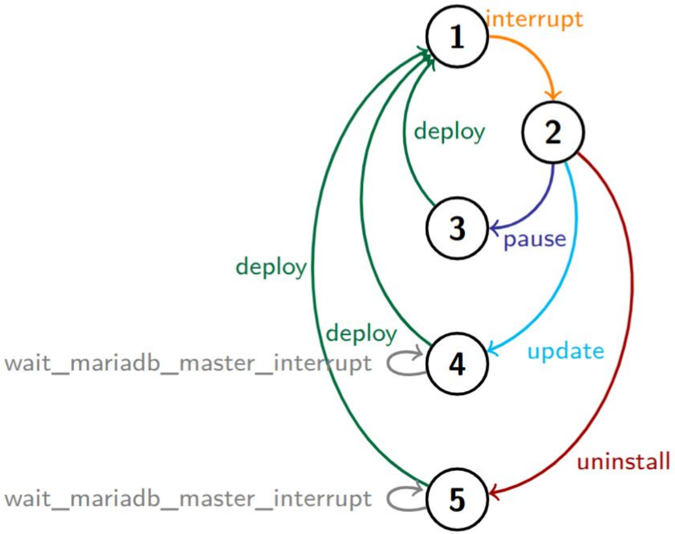
s4:



Local goal: CP on automaton

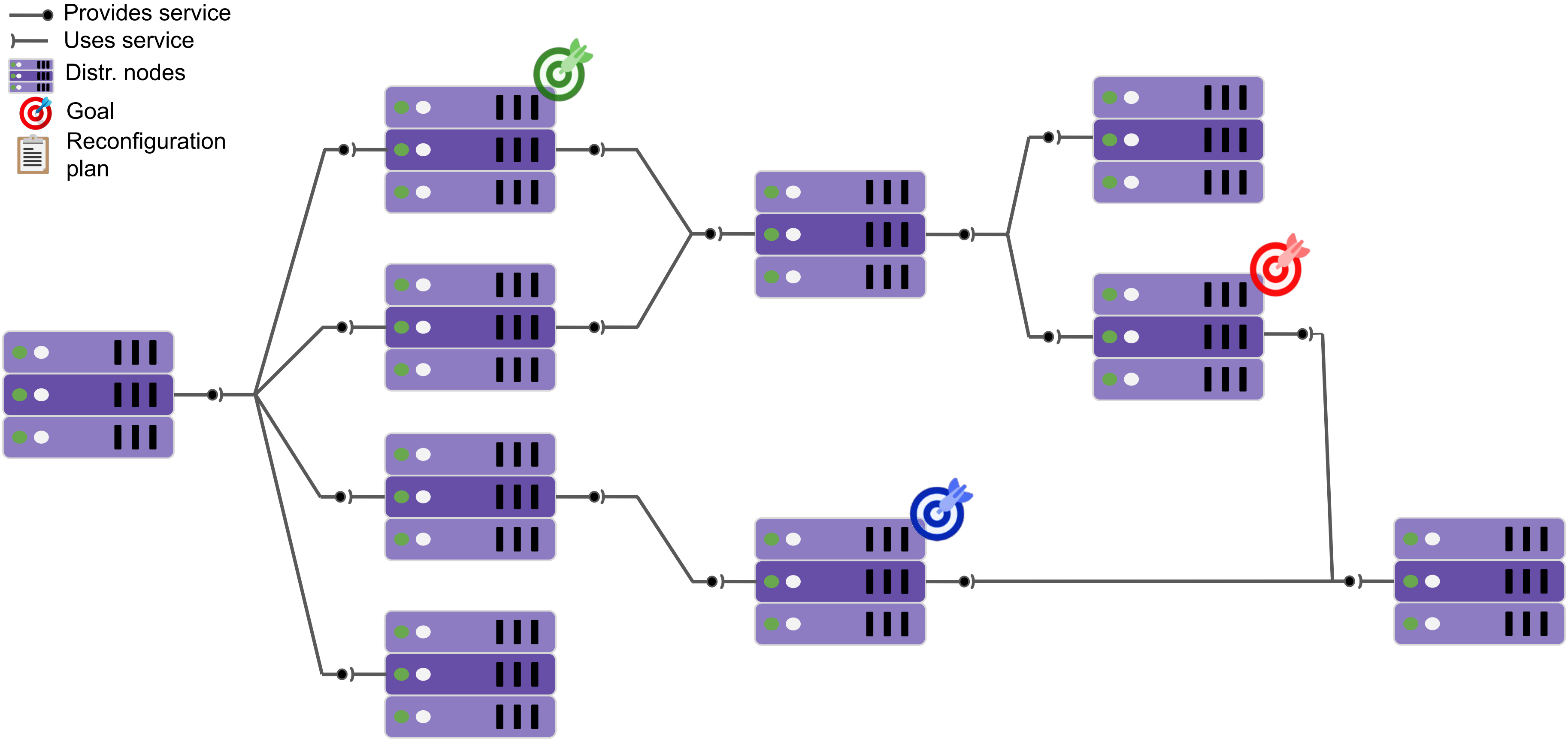


mdbworker0:MariaDB_worker

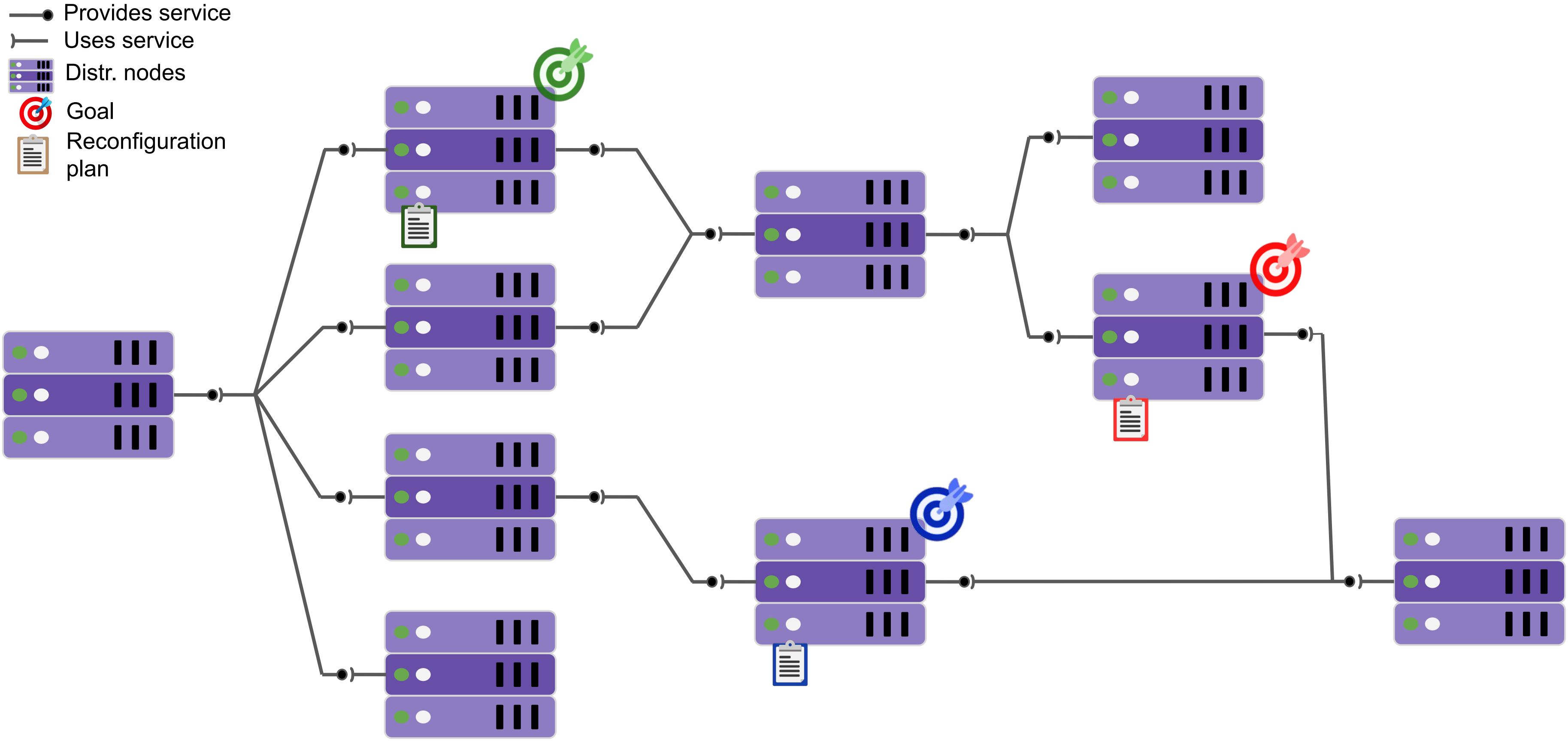


- Enriched automata
  - Add transitions on state validating port requirement
- Additional constraints
  - **Count(wait_comp_bhv, B, >, 0)** mandatory synchronization
- Example
  - **Count(wait_mariadb_master_update, B, >, 0)**

Sharing constraints: Local goal



Sharing constraints: Local decision



—● Provides service

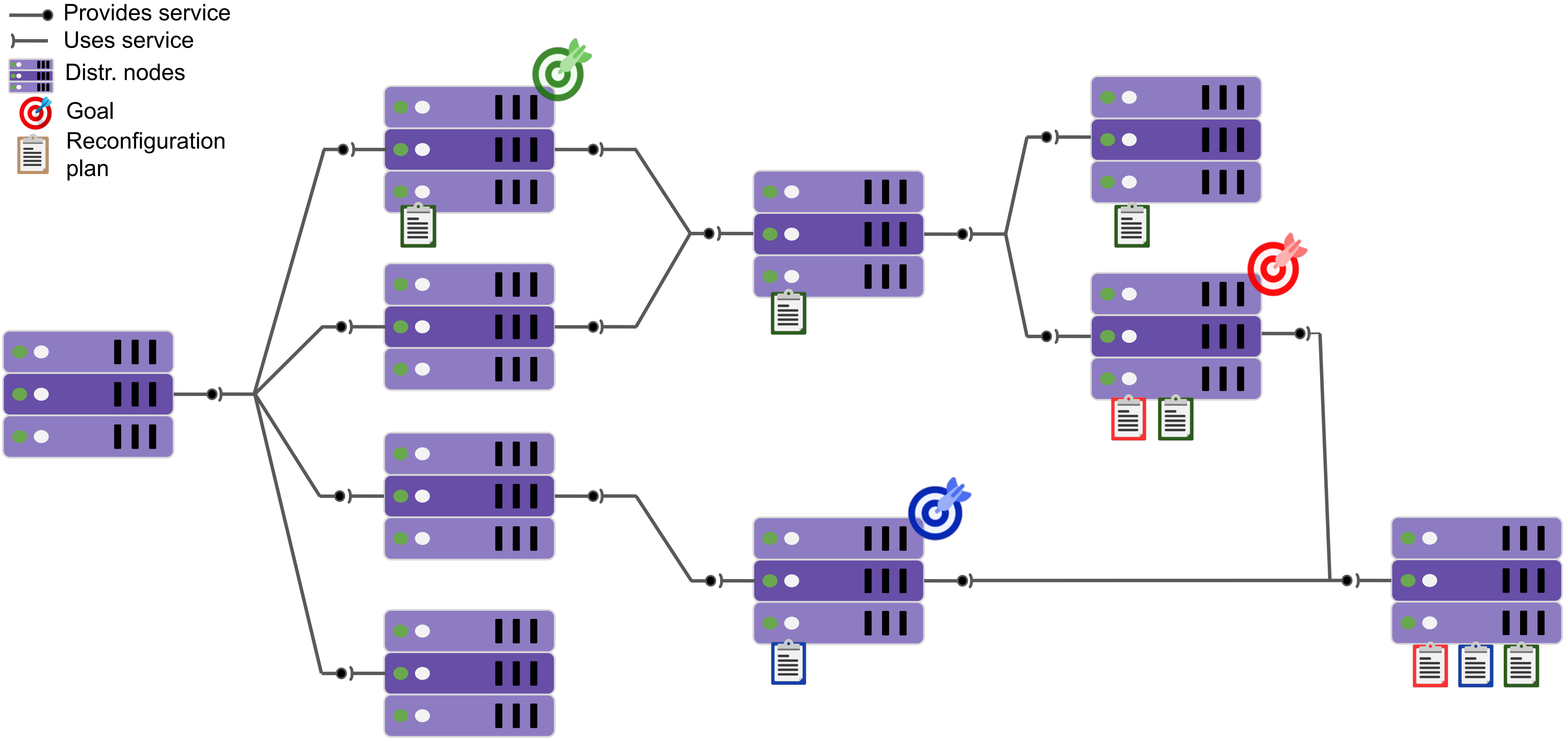
)— Uses service

Distr. nodes

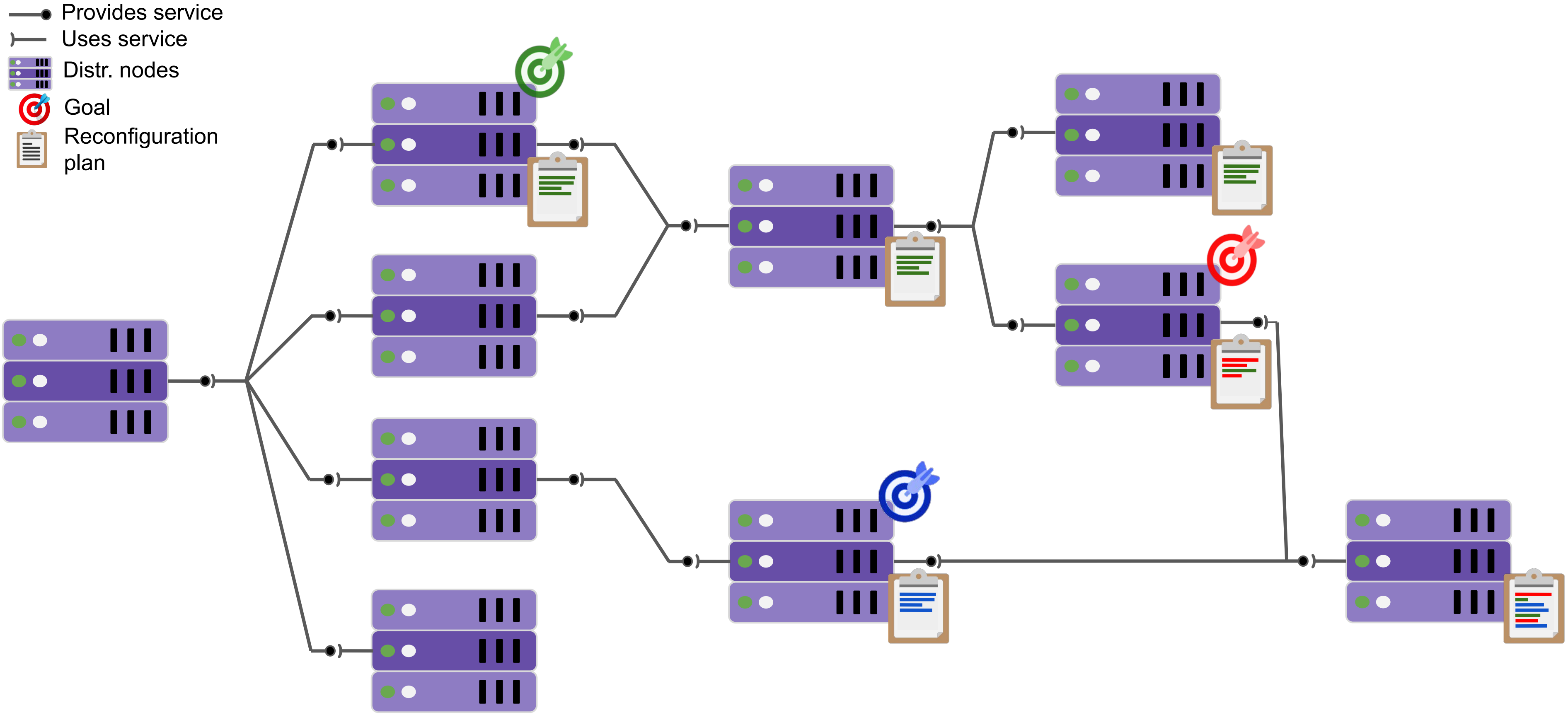
Goal

Reconfiguration plan

Sharing constraints: Local decision propagation



Sharing constraints: Local plan (Sync + Optimization)



—● Provides service

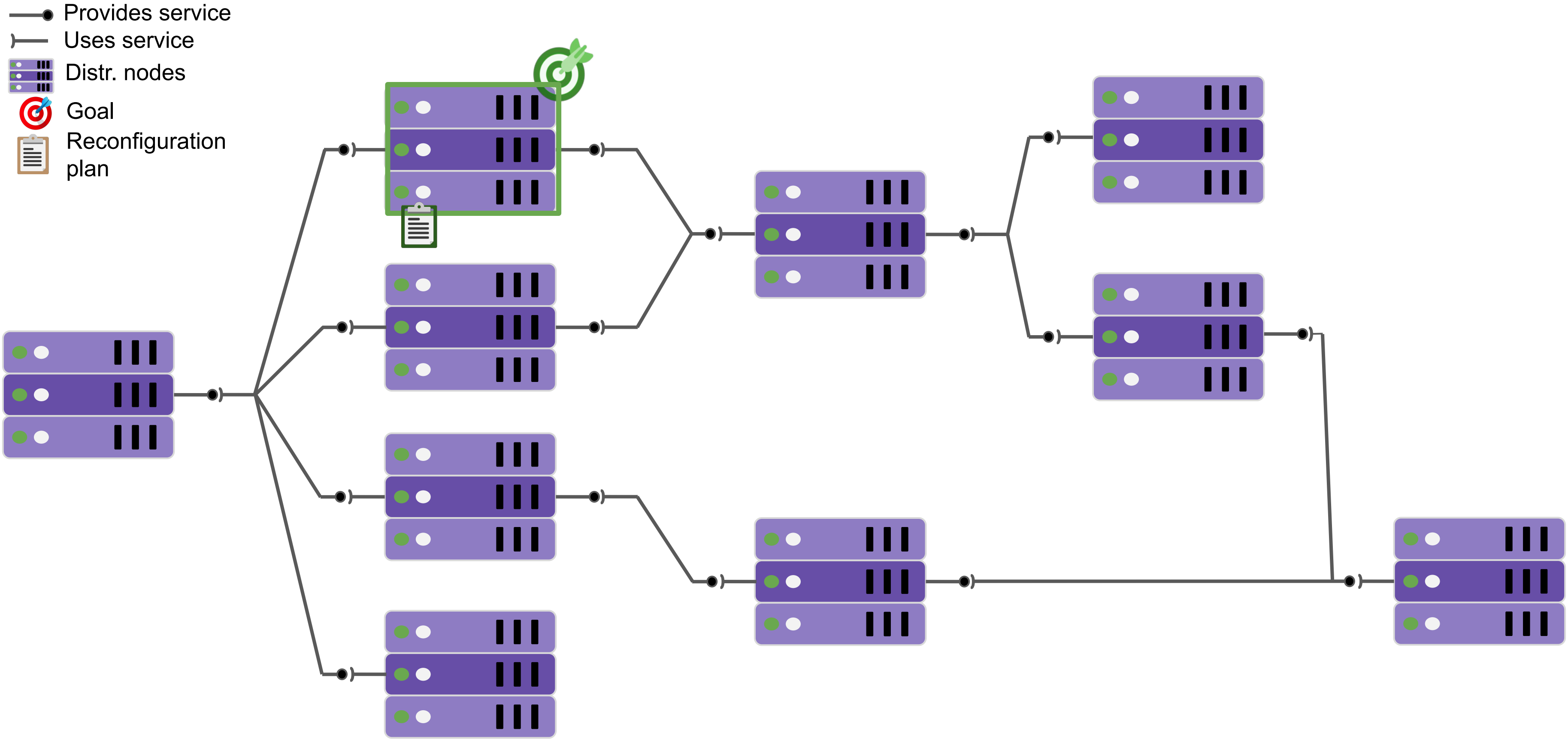
)— Uses service

Distr. nodes

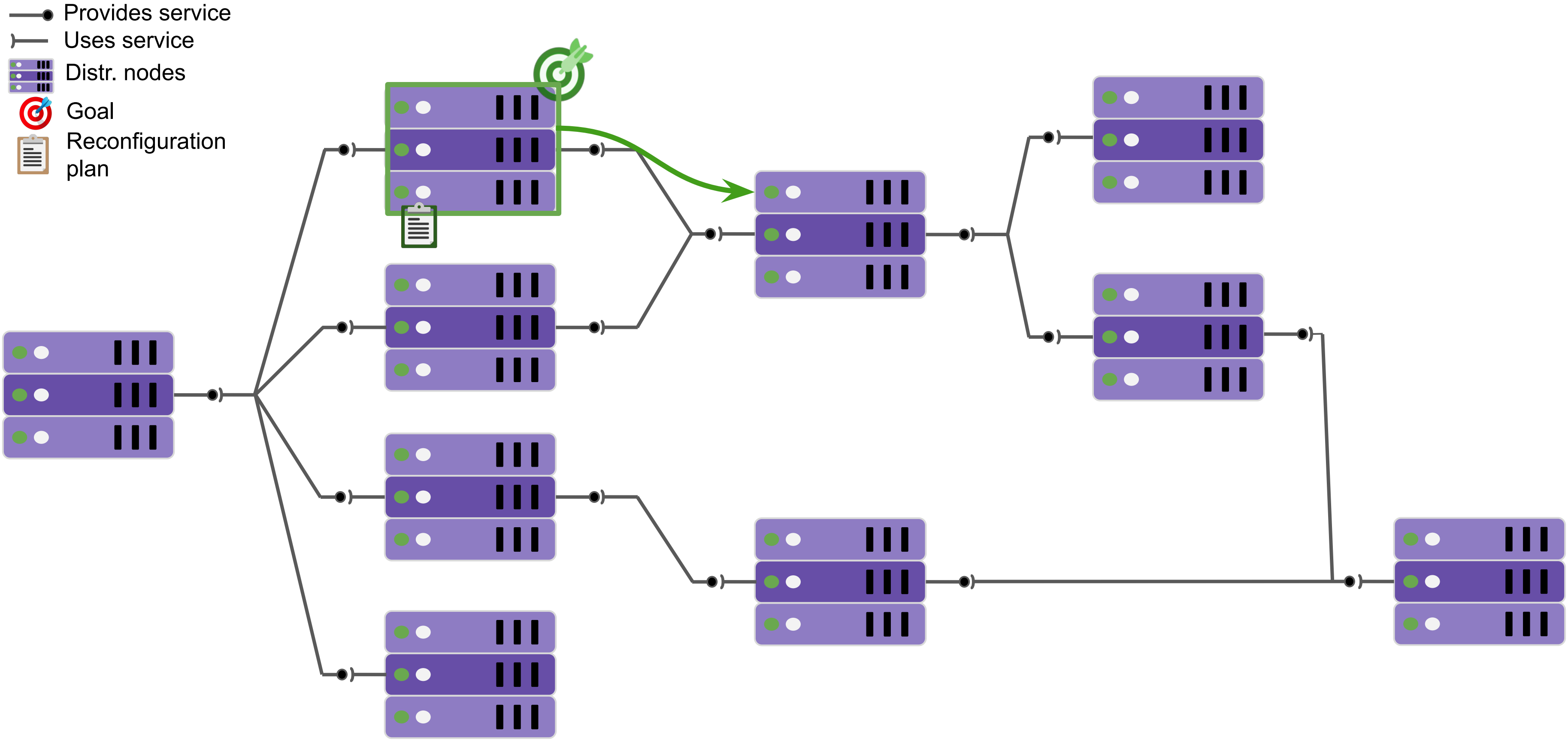
Goal

Reconfiguration plan

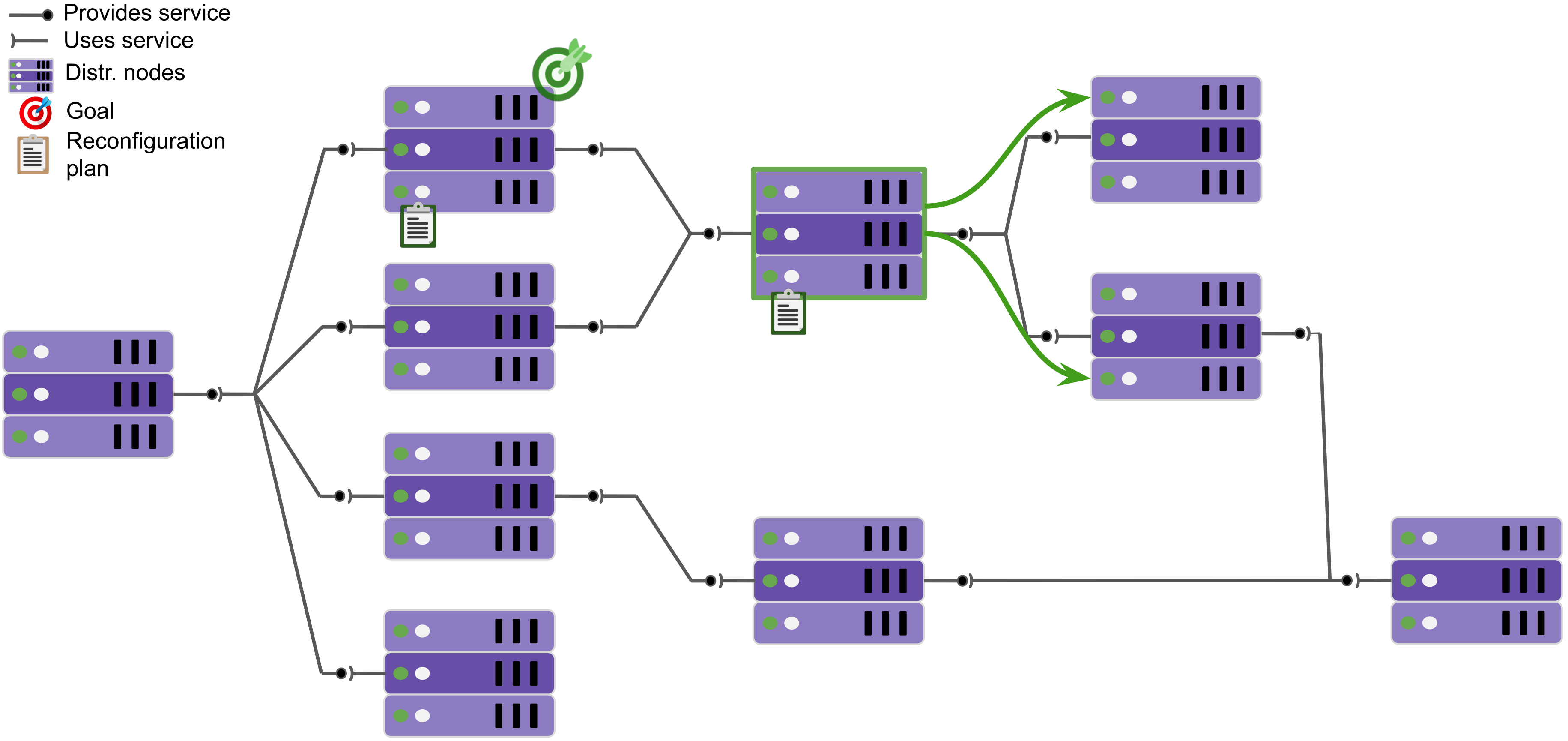
Communication protocol. Step I: Propose



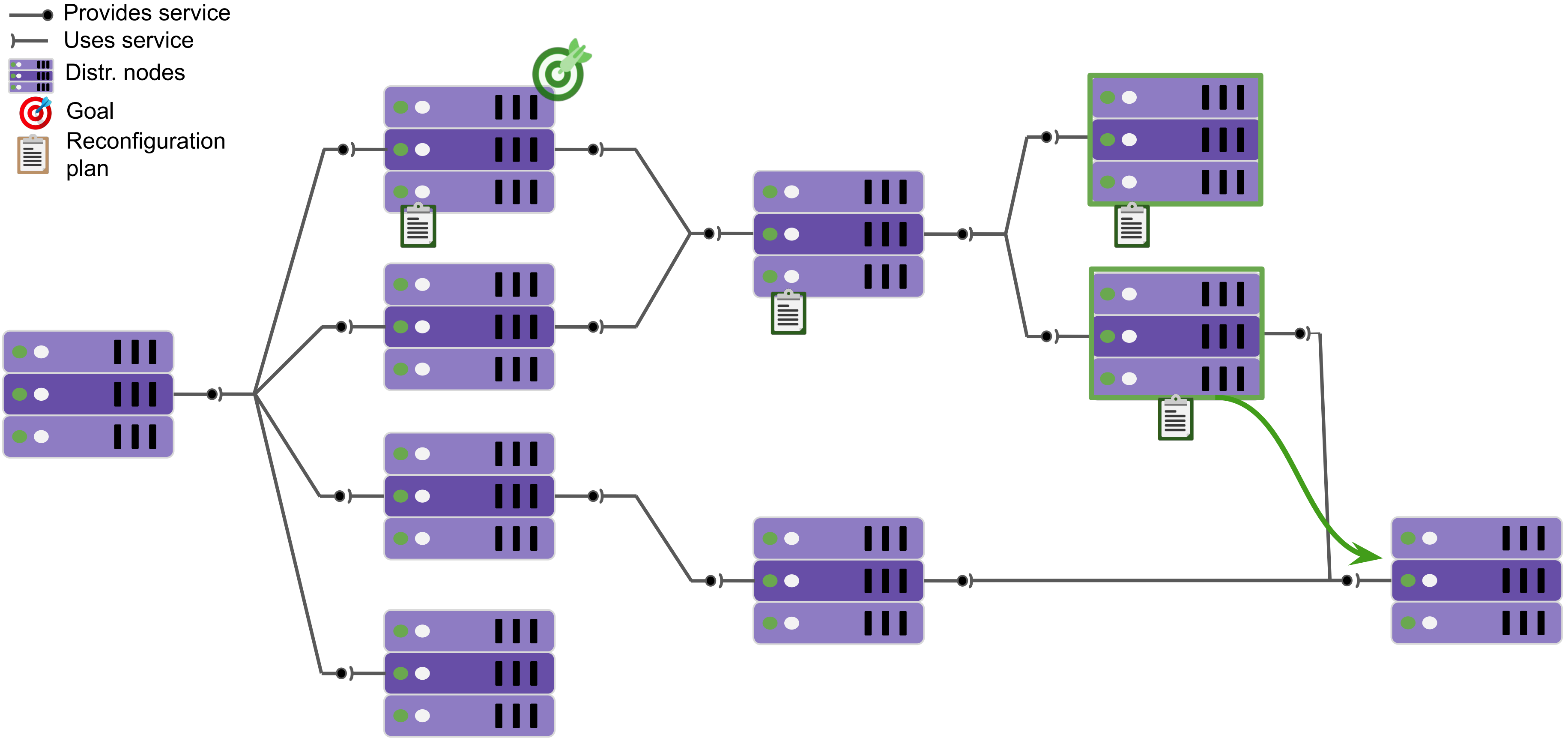
Communication protocol. Step I: Propose



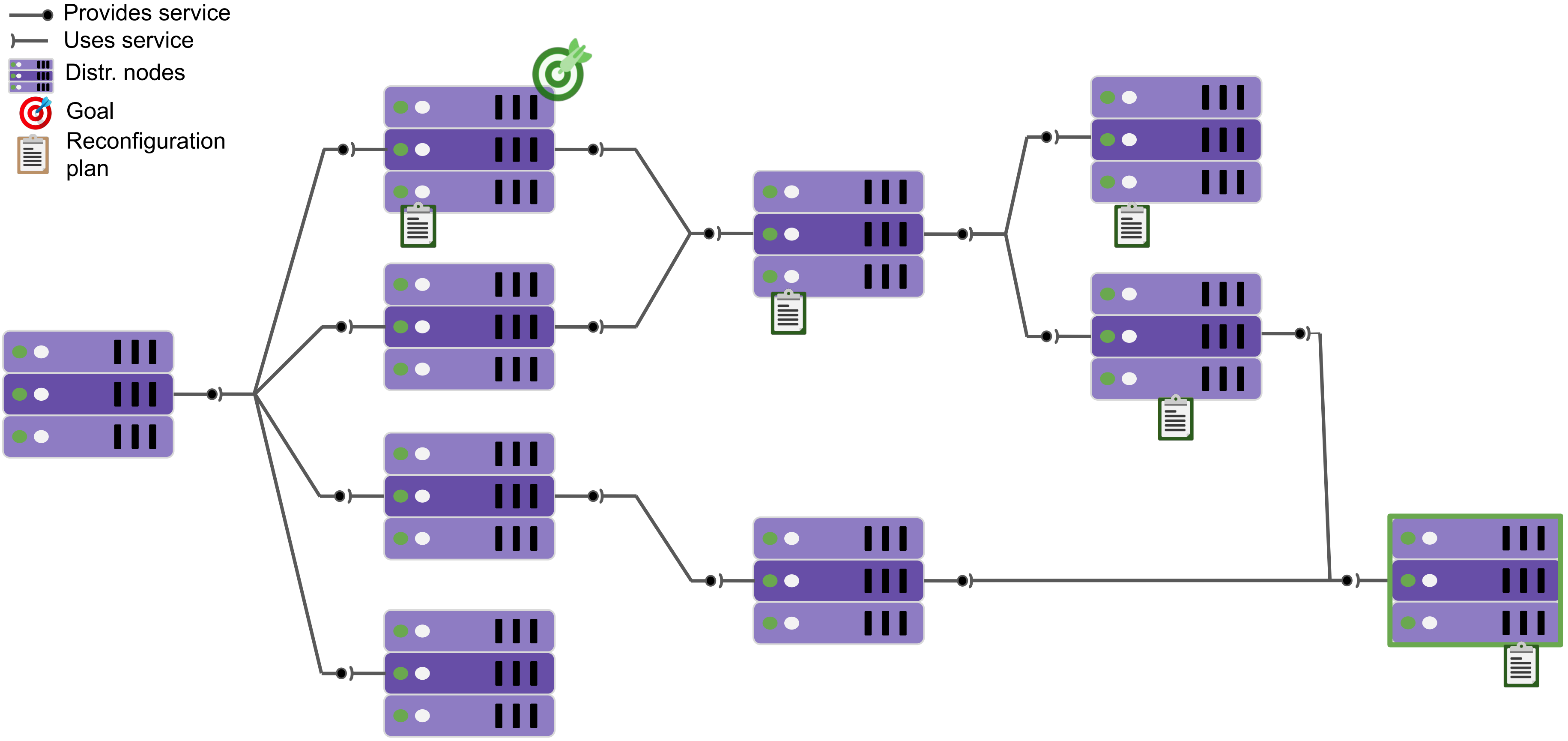
Communication protocol. Step I: Propose



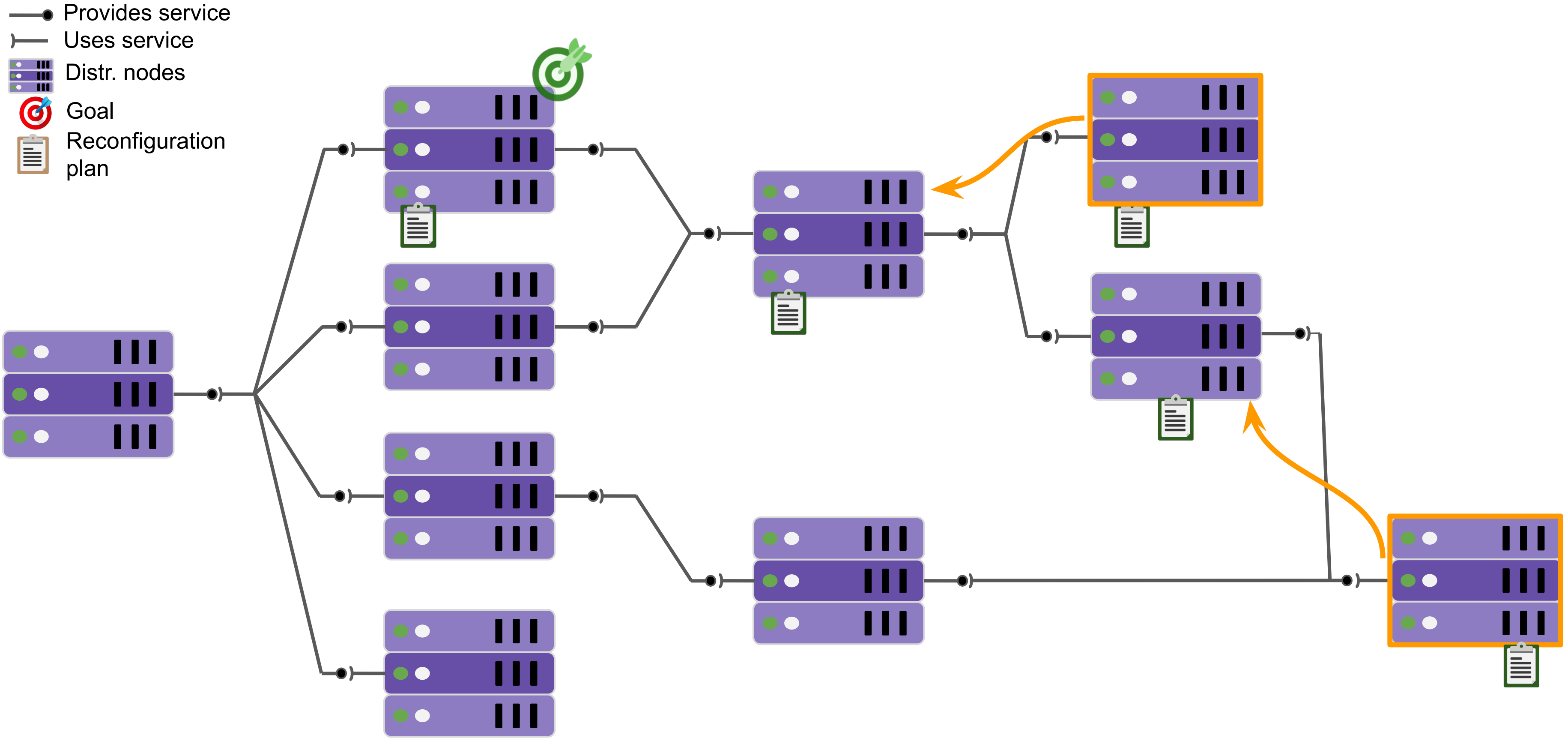
Communication protocol. Step I: Propose



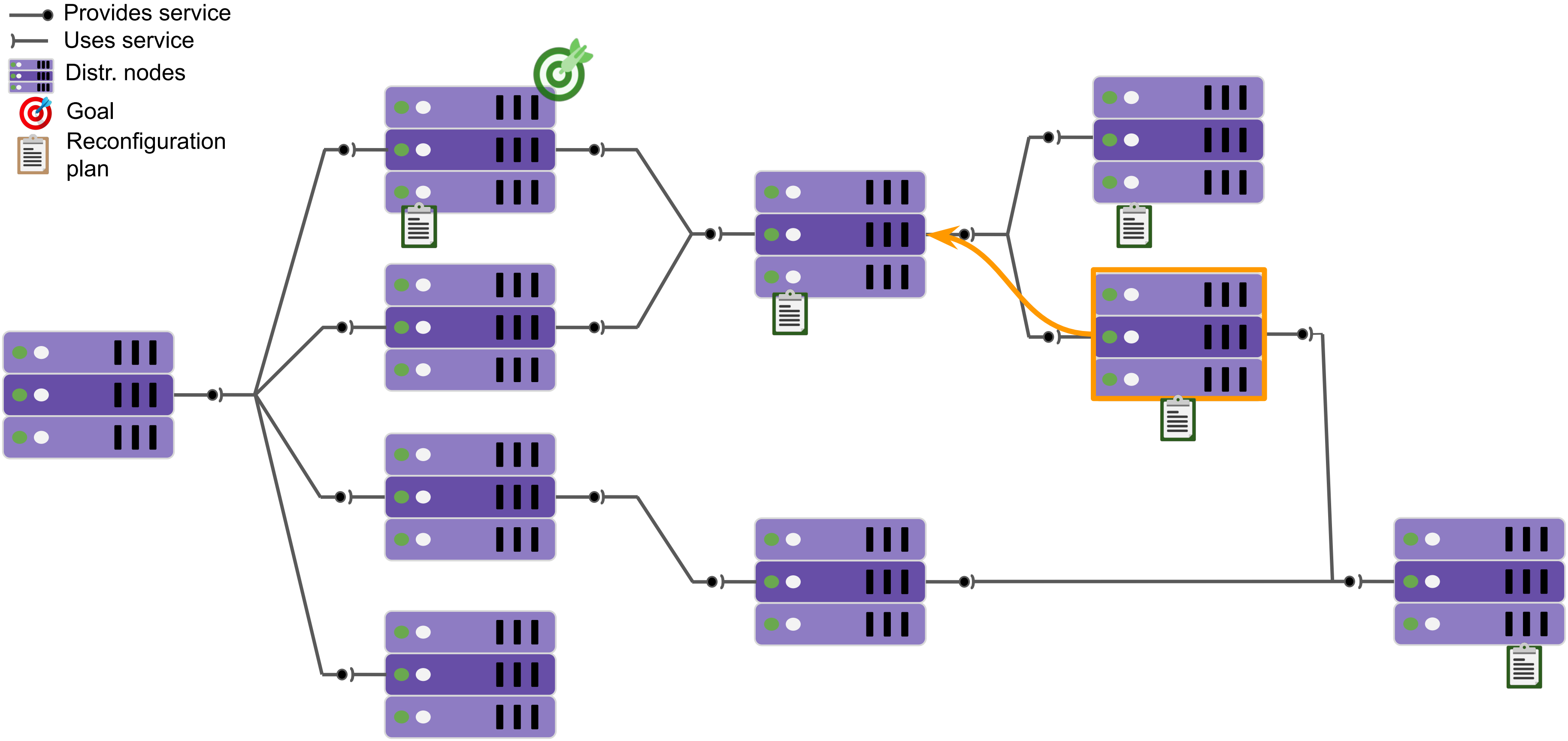
Communication protocol. Step I: Propose



Communication protocol. Step II: Send ack



Communication protocol. Step II: Send ack



—● Provides service

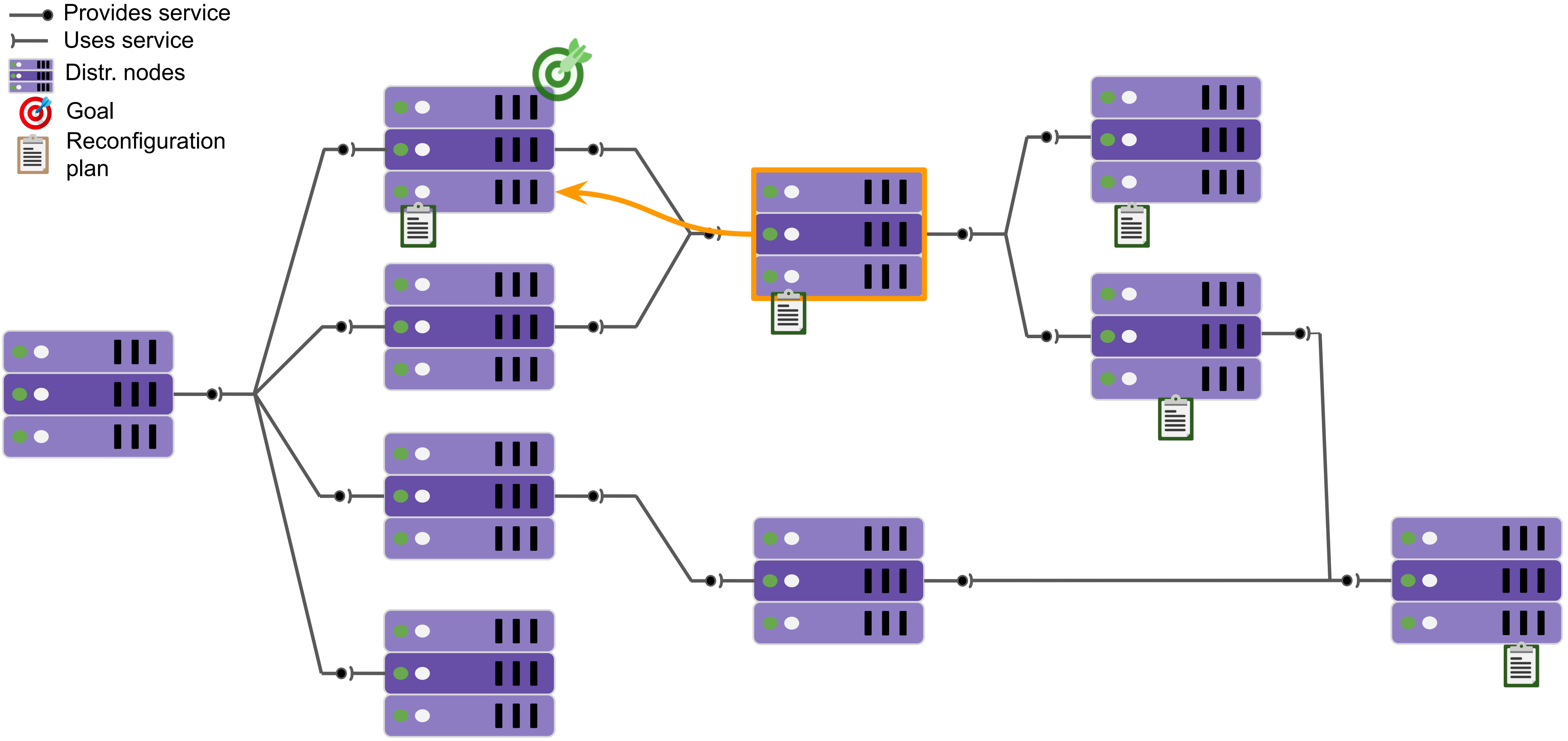
)— Uses service

Distr. nodes

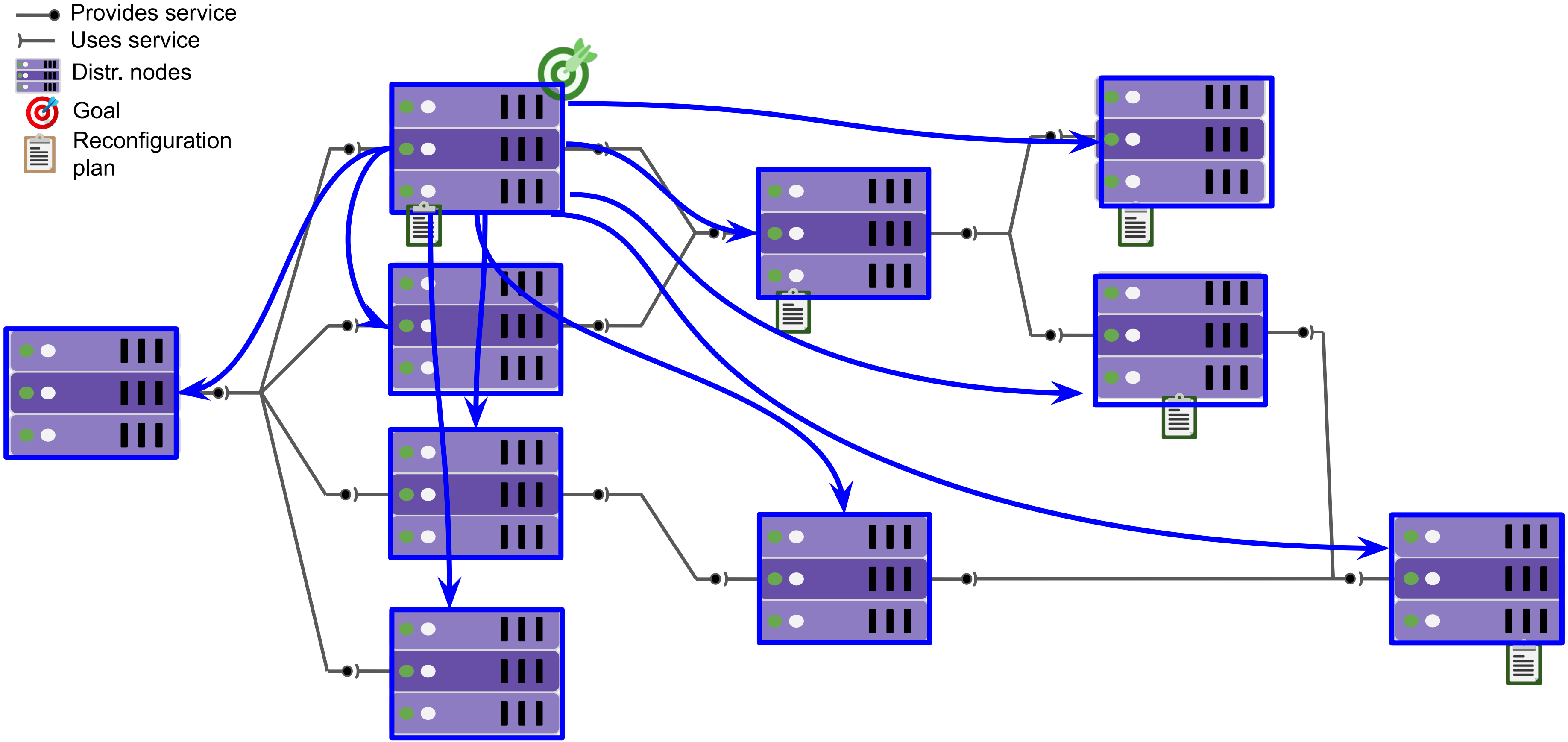
Goal

Reconfiguration plan

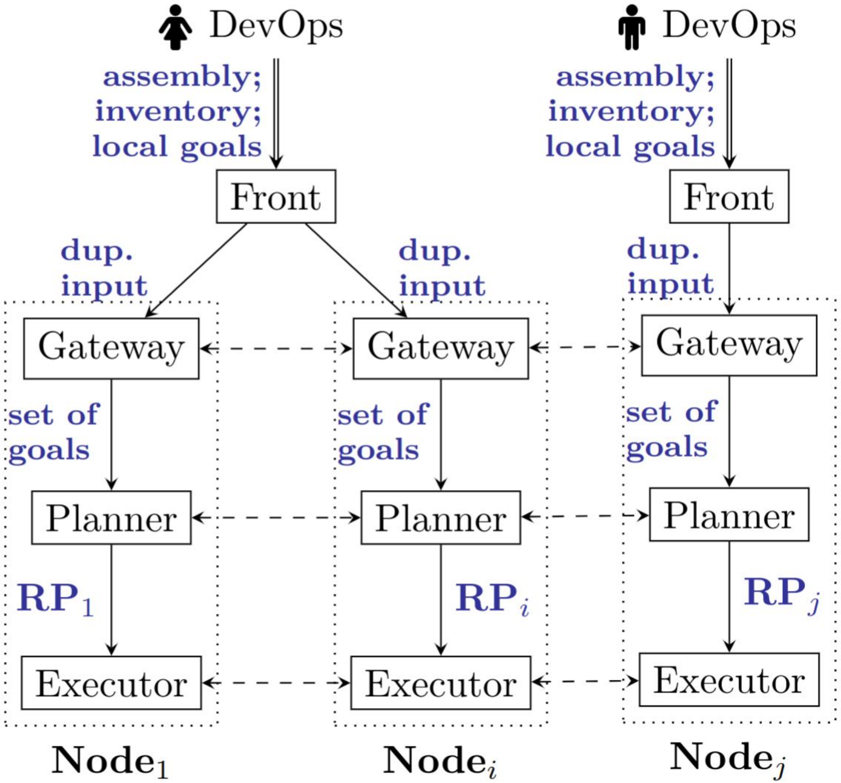
Communication protocol. Step II: Send ack



Communication protocol. Step III: Global ack



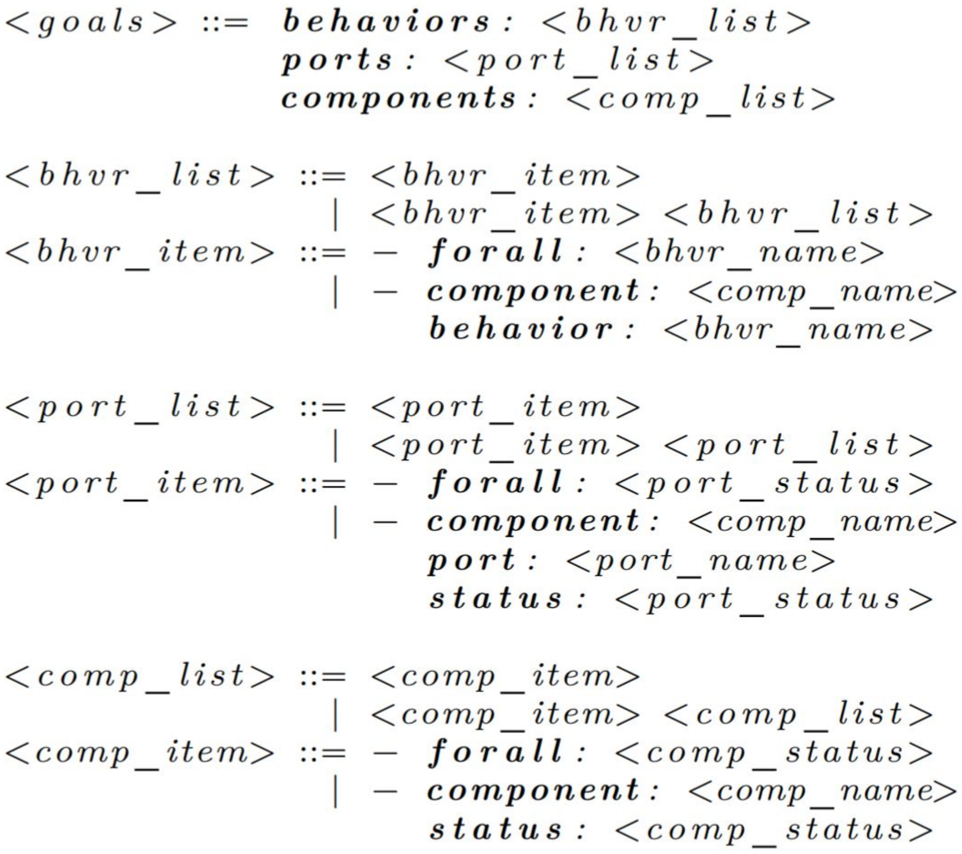
Integration within Ballet



Integrated in **Ballet** (8400 LoC), a DevOps declarative reconfiguration tool

- A high-level language for defining goals
- A decentralized planner
  - MiniZinc
  - GeCode
- A decentralized execution (inspired by Concerto)

Find a plan from goals



```
<goals> ::= behaviors: <bhvr_list>
            ports: <port_list>
            components: <comp_list>

<bhvr_list> ::= <bhvr_item>
              | <bhvr_item> <bhvr_list>
<bhvr_item> ::= - forall: <bhvr_name>
              | - component: <comp_name>
                  behavior: <bhvr_name>

<port_list> ::= <port_item>
              | <port_item> <port_list>
<port_item> ::= - forall: <port_status>
              | - component: <comp_name>
                  port: <port_name>
                  status: <port_status>

<comp_list> ::= <comp_item>
              | <comp_item> <comp_list>
<comp_item> ::= - forall: <comp_status>
              | - component: <comp_name>
                  status: <comp_status>
```

Novel language for goal definition:

- Mandatory behavior(s)
- Status for port(s)
- Status of a component (specific, “running”, “initial”)

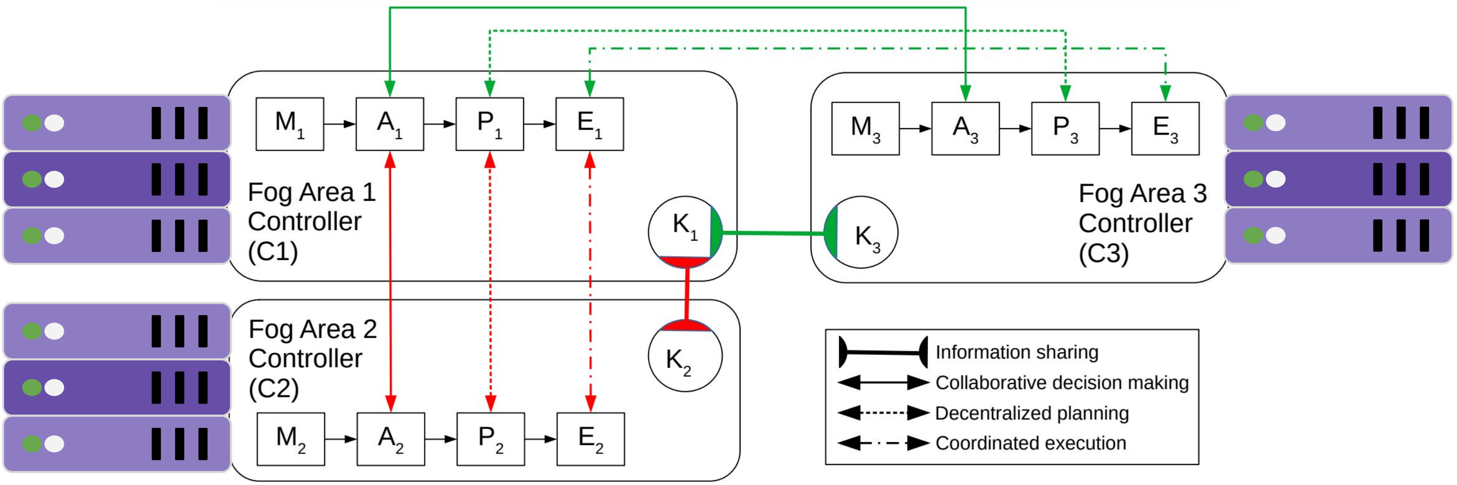
SeMaFoR proposal for controller coordination

MAPE-K [IBM, 2006]**:** ***Coordinated Control Pattern*** model

- **M**onitor its state and the state of the environment 👁️
- **A**nalyze to decide which state to reach 🎯

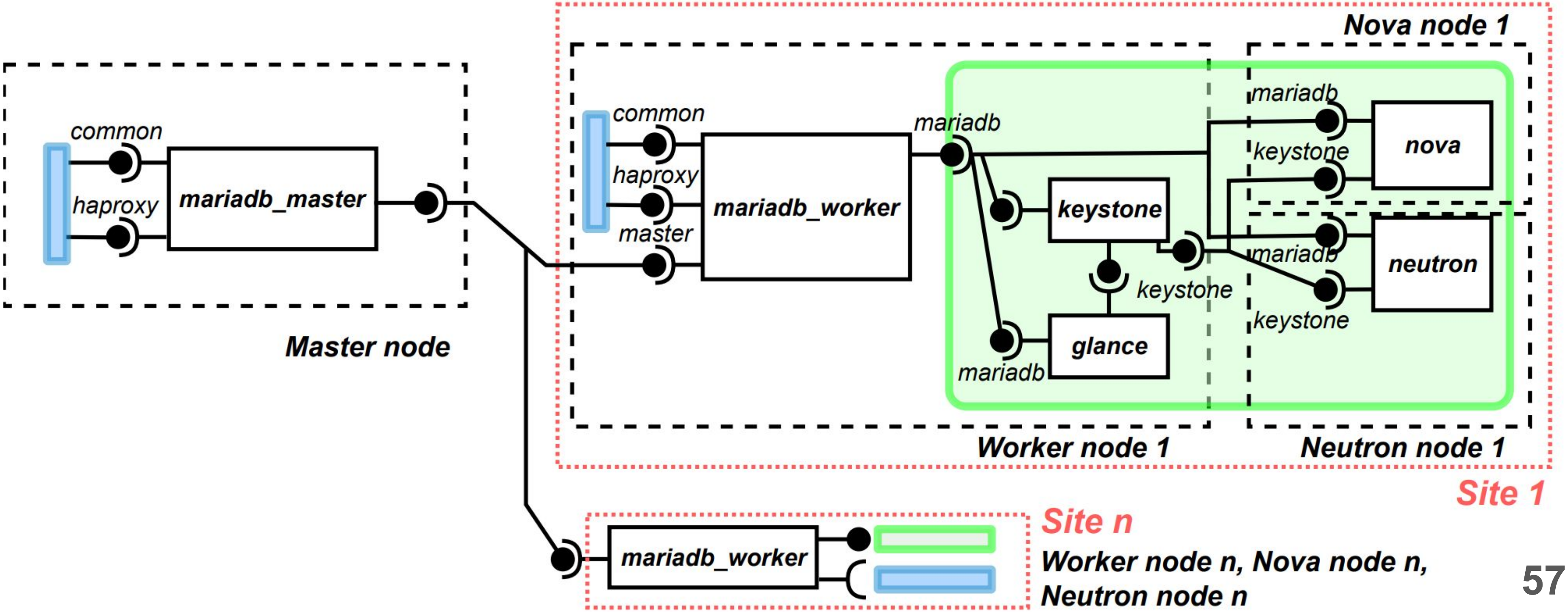
Ballet {
- **P**lan the reconfiguration 📋
- **E**xecute the reconfiguration to reach the new state ⚙️

  ○ **K**nowledge that is common, to take a decision



Example of model and reconfiguration goal: OpenStack

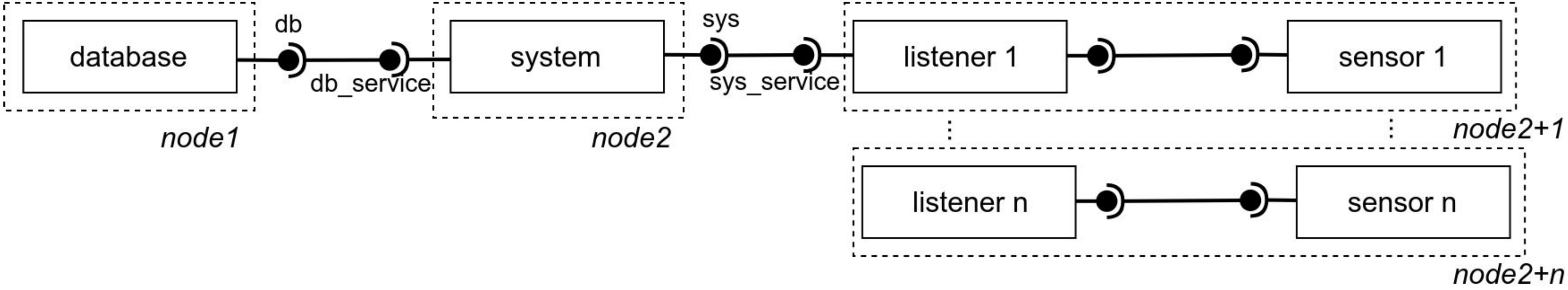
- Cluster of MariaDB in OpenStack (from real case)
  - OpenStack = deployment solution
  - Many components, scalability opportunity



Example of model and reconfiguration goal: CPS

From discussion with STR team

- CPS
  - Listen birds sounds
  - Communicate with a gateway
  - Need to be reconfigured
    - Update of system
    - Change freq. of observation



Planner metrics

assembly	scenario	# sites	# inf constraint	# gen instructions	# total msgs
openstack	deploy	n	7 + 11 * n	7 + 11 * n	0
openstack	update	n	3 + 20 * n	8 + 11 * n	9 * n
cps	deploy	n	2 + 2 * n	2 + 2 * n	0
cps	update	n	3 + 2 * n	6 + 6 * n	1 + 5 * n

n = 10

assembly	scenario	# sites	# inf constraint	# gen instructions	# total msgs
openstack	deploy	10	117	117	0
openstack	update	10	203	118	90
cps	deploy	10	22	22	0
cps	update	10	23	66	51

Performances: Deployment

openstack™

assembly	# sites	avg planner (std)	avg executor (std)	Ballet	Muse	Gain
openstack	1	1.69s (0.11)	306.02s (0.32)	307.71s	536.57s	42.65%
openstack	2	1.78s (0.07)	306.09 (0.15)	307.86s	536.69s	42.64%
openstack	5	1.77s (0.07)	306.19 (0.11)	307.97s	537.09s	42.66%
openstack	10	2.02 (0.04)	306.14 (0.21)	308.17s	537.64s	42.68%

assembly	# sites	avg planner (std)	avg executor (std)	Ballet	Muse	Gain
cps	1	0.48s (0.06)	24.36s (0.06)	24.84s	66.18s	62.47%
cps	2	0.46s (0.06)	24.32s (0.22)	24.78s	66.29s	62.62%
cps	5	0.47s (0.06)	38.03s (0.25)	38.5s	84.35s	54.36%
cps	10	0.49s (0.08)	38.02s (0.22)	38.5s	84.81s	54.6%
cps	15	0.49s (0.07)	37.98s (0.20)	38.47s	93.02s	58.64%

Performances: Update

assembly	# sites	avg planner (std)	avg executor (std)	Ballet	Muse	Gain
openstack	1	3.36s (0.43)	416.84s (0.28)	420.2s	555.56s	24.37%
openstack	2	4.39s (1.72)	416.9s (0.22)	421.3s	555.7s	24.18%
openstack	5	6.05s (0.91)	417.17s (0.12)	423.22s	556.08s	23.89%
openstack	10	5.97s (0.68)	417.46s (0.21)	423.43s	556.77s	23.95%

assembly	# sites	avg planner (std)	avg executor (std)	Ballet	Muse	Gain
cps	1	1.63s (0.37)	60.35s (0.15)	61.99s	106.92s	42.03%
cps	2	1.83s (0.46)	60.44s (0.14)	62.27s	107.02s	41.81%
cps	5	3.24s (0.94)	79.10s (0.21)	82.34s	150.77s	45.39%
cps	10	5.72s (0.81)	79.09s (0.20)	84.31s	151.38s	43.98%
cps	15	8.05s (1.65)	79.40s (0.22)	87.44s	151.89s	42.43%

Concluding remarks

Postdoc contributions

- SeMaFoR project and Ballet
- Infer reconfiguration actions (CP-based approach)
- Communication protocol

Target applications:

- (SeMaFoR) Smart cities, smart buildings, smart factories, etc.
- OpenStack, and CPS

Perspectives:

- Model-Driven Engineering approach for determining objectives
- Experiments on more topologies
- Formalization of Planner + Executor in Why3 for correctness

References:

[Cisco, mar. 2015]	Maher Abdelshkour. From Cloud to Fog Computing. Cisco, 2015
[IBM, 2006]	A. Computing et al. An architectural blueprint for autonomic computing. IBM White Paper, 2006.
[SeMaFoR, 2023]	SeMaFoR - Self-Management of Fog Resources with Collaborative Decentralized Controllers
[Robillard, apr. 2022]	Simon Robillard et al. SMT-Based Planning Synthesis for Distributed System Reconfigurations. FASE 2022