

Vérifier des reconfigurations : ongoing and future work

Jolan Philippe

IRISA – Université de Rennes (yet)

LIFO – Université d'Orléans (soon)

Journées LMV

LIFO – Université d'Orléans



https://jolanphilippe.github.io/docs/talks/2025_07_17_lmv-verif.pdf

Parcours

Master &

 M1 MIAGE  Université d'Orléans (LMV)

 M.Sc.  Northern Arizona University (SSERL)

 Frédéric Loulgué

Doctorat &

 Ph.D  IMT Atlantique, Nantes (Naomod)

 Gerson Sunye, Massimo Tisi, Hélène Coullon

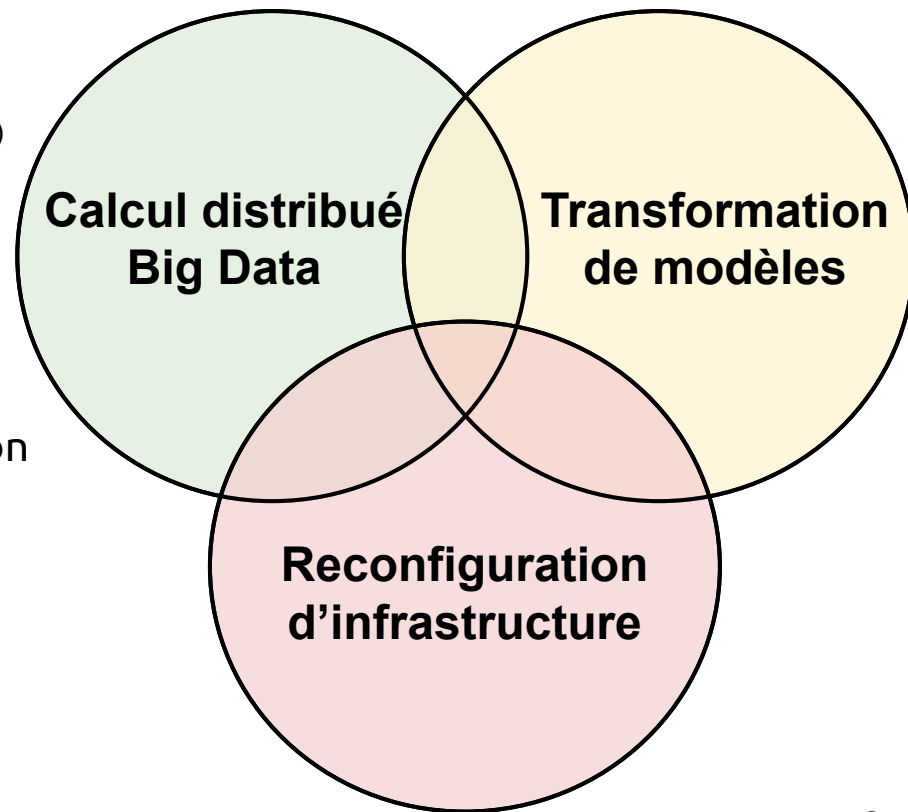
Postdoc &

 IMT Atlantique, Nantes (STACK)

 Hélène Coullon, Charles Prud'Homme

 Université de Rennes (DiverSE)

 Olivier Barais



Thématiques de recherche

Calcul distribué – Big Data

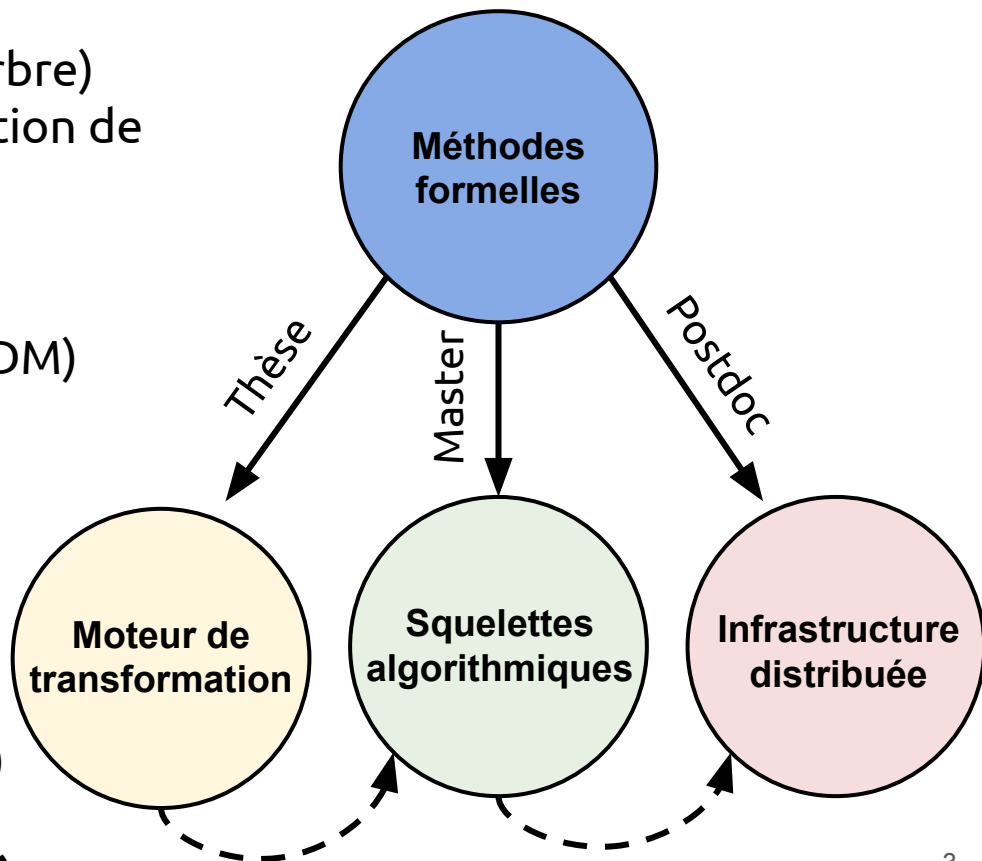
- Structure de données distribuées (arbre)
- **Formalisation (Coq)** et implémentation de squelettes

Transformation distribuée de modèles

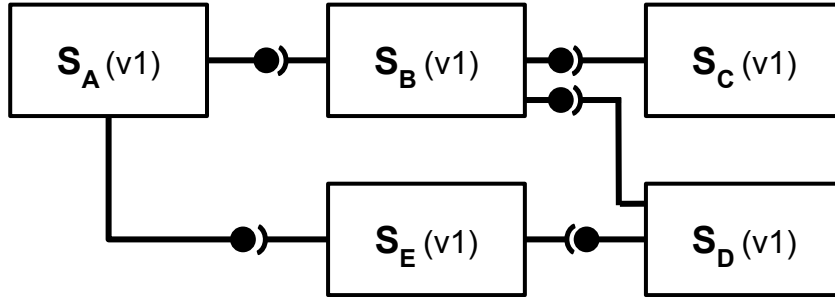
- Ingénierie dirigée par les modèles (IDM)
- Exploration de la **variabilité**
- **Équivalence de sémantique (Coq)**

Reconfiguration d'infrastructure

- **Planification** décentralisée
- Satisfiabilité de reconfiguration
 - **Sémantique du moteur (Maude)**
 - **Formalisation de cycle de vie et modèle SAT (MiniZinc, Choco)**

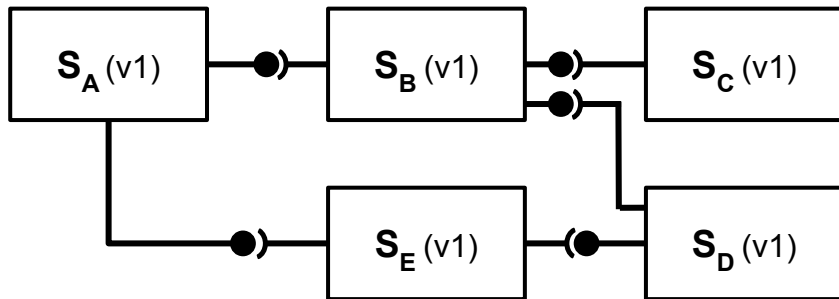


Architecture orientée services



- Approche adoptée dans les systèmes distribués (Cloud)
- Architecture microservices
- Interfaces use/provider pour composition

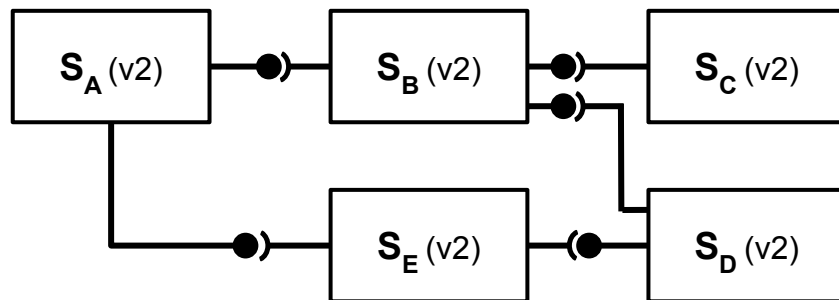
Architecture orientée services



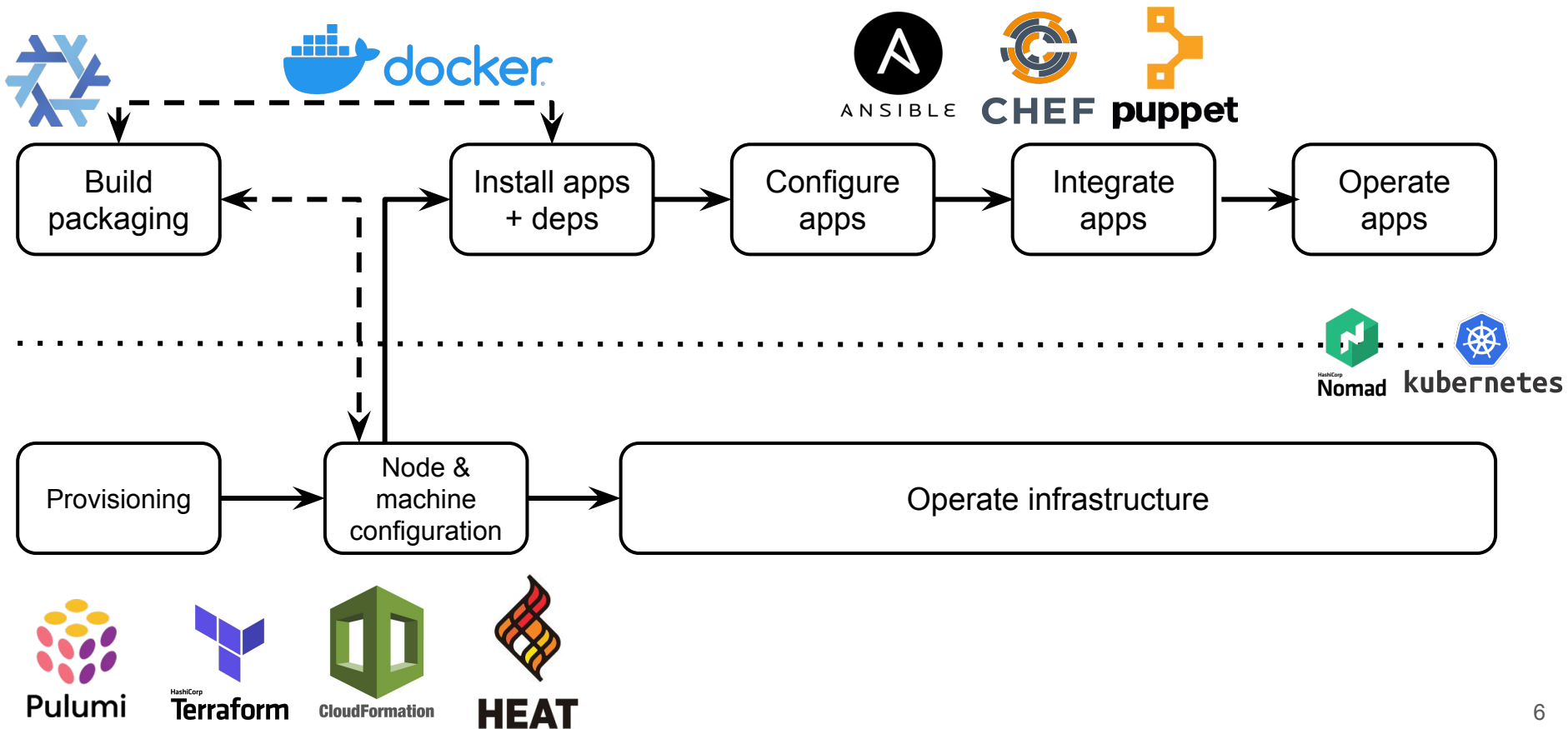
- Approche adoptée dans les systèmes distribués (Cloud)
- Architecture microservices
- Interfaces use/provide pour composition

Reconfiguration d'infrastructure

- Complexes sur les gros assemblages
 - Gérer les dépendances
 - Maintenabilité
 - Décentralisation
- Donner des garanties formelles sur les mécanismes de reconf.



Infrastructure-as-code



Running example

worker



worker



worker

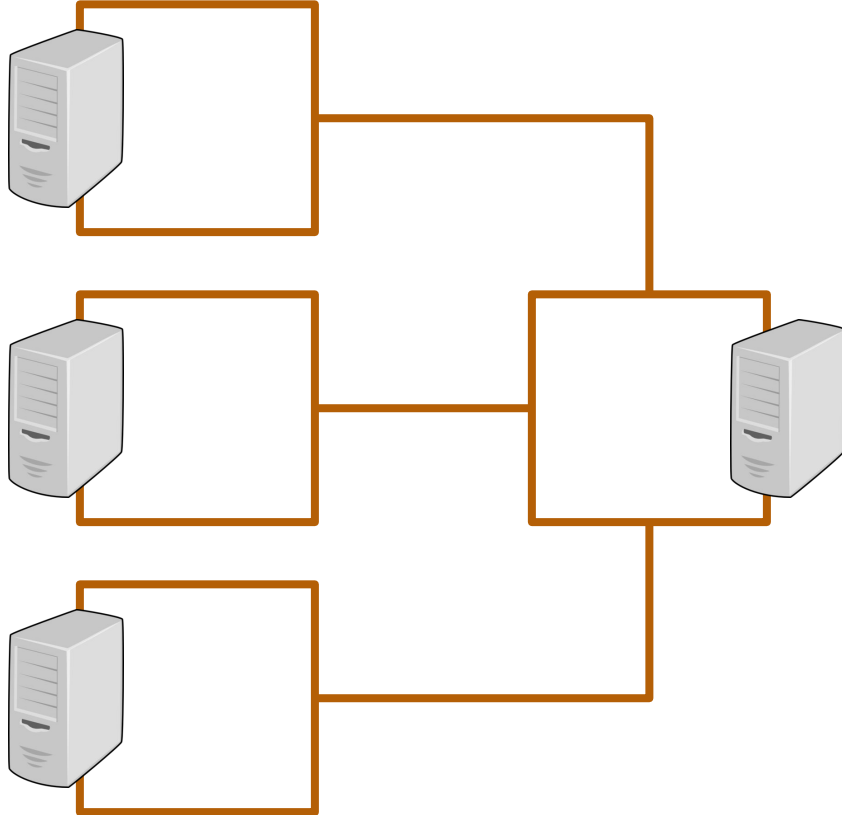


master



**Déploiement et management d'une
base de données distribuées**

Running example

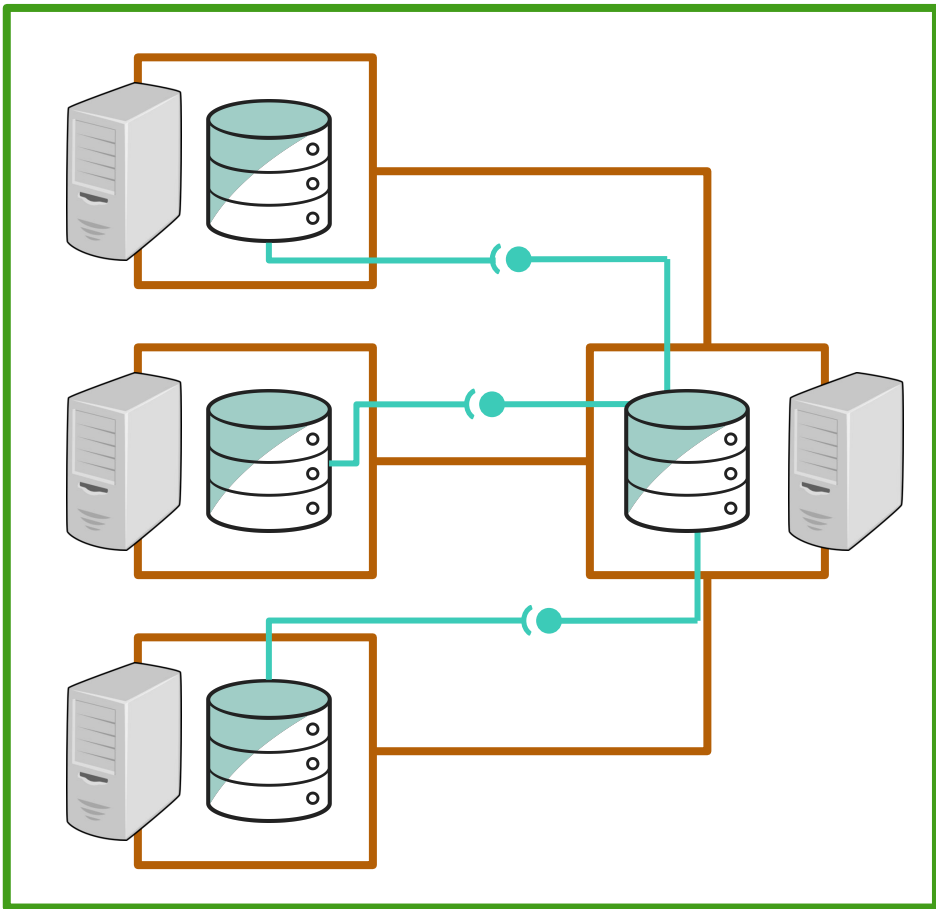


Déploiement et management d'une base de données distribuées

Provisionnement

- Création d'un VPC (subnet)
- Demander des machines
- Définition des VMs/cluster
- Firewall

Running example



Déploiement et management d'une base de données distribuées

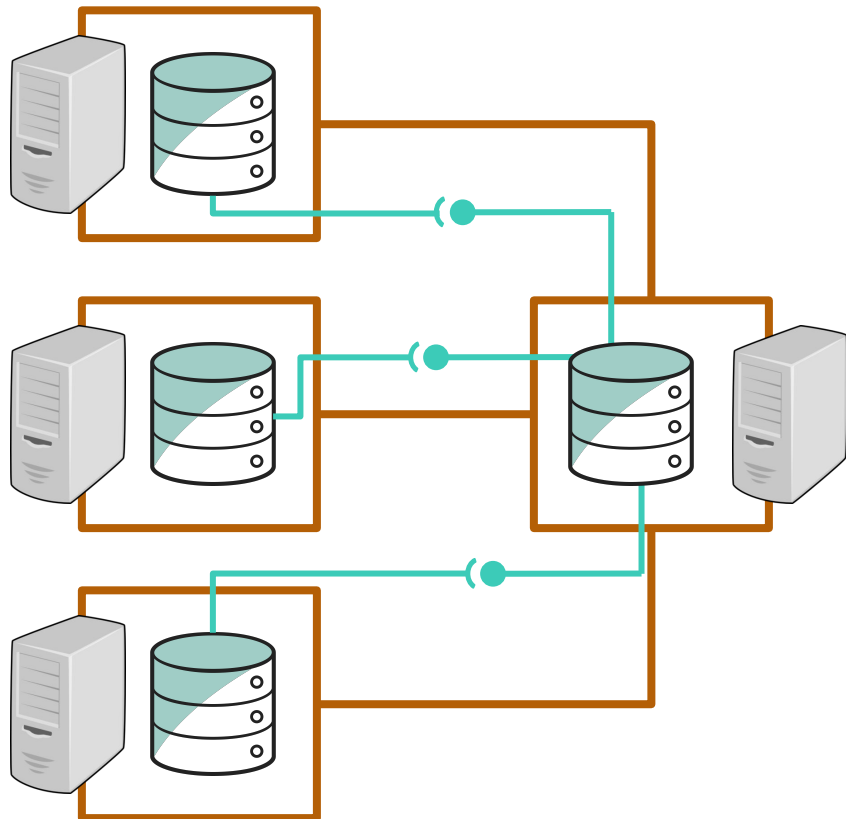
Provisionnement

- Création d'un VPC (subnet)
- Demander des machines
- Définition des VMs/cluster
- Firewall

Configuration management

- Installer Redis (bin + deps)
- Configurer les BDDs
- Bootstrap les BDDs
- Intégrer workers (followers) et master (leader)

Travaux en cours : vérification dans l'IaC



Déploiement et management d'une base de données distribuées

Provisionnement ← VÉRIFIER ?

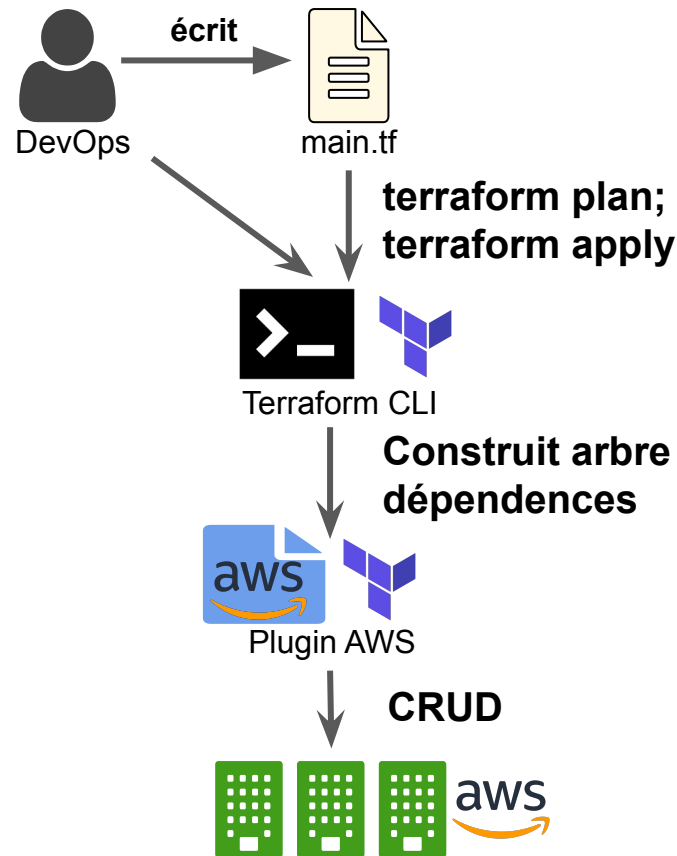
- Création d'un VPC (subnet)
- Demander des machines
- Définition des VMs/cluster
- Firewall

Configuration management ← VÉRIFIER

- Installer Redis (bin + deps)
- Configurer les BDDs
- Bootstrap les BDDs
- Intégrer workers (followers) et master (leader)

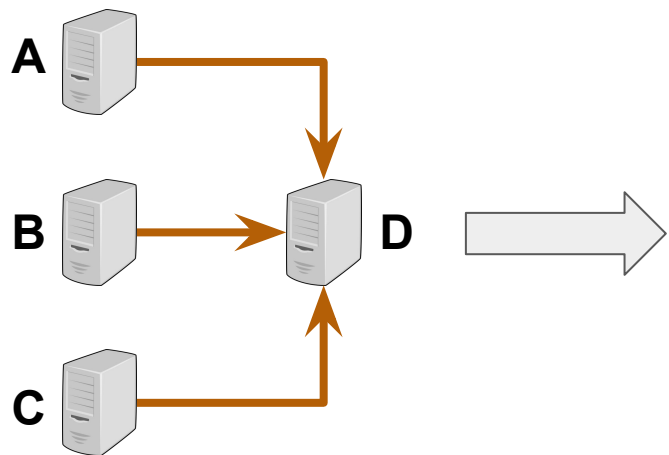
Provisionnement avec Terraform

```
1  resource "aws_vpc" "main" {
2      cidr_block = "10.0.0.0/16"
3  }
4
5  resource "aws_subnet" "public" {
6      vpc_id      = aws_vpc.main.id
7      cidr_block   = "10.0.1.0/24"
8      availability_zone = "us-east-1a"
9  }
10
11 resource "aws_security_group" "redis_sg"
12 {
13     # attributes for security group
14 }
15
16 resource "aws_instance" "redis_nodes" {
17     # attributes for our nodes
18 }
```



Plan de reconfiguration

Arbre de dépendances



Plan de déploiement

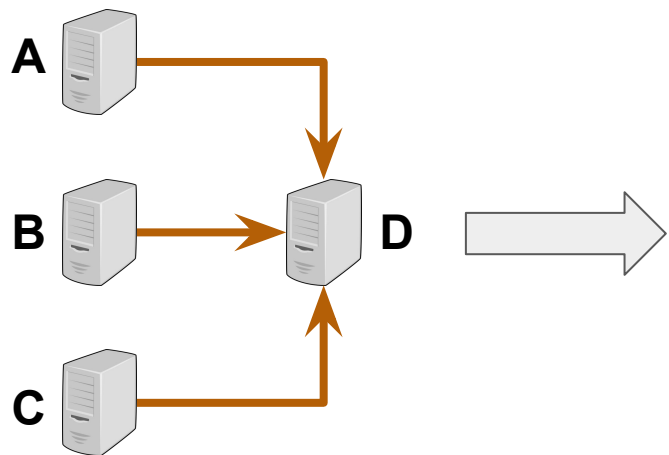
- **Create D; Create A; Create B; Create C**

Plans pour mettre à jour D

- **Update D; Update A; Update B; Update C**
- **Delete A; Delete B; Delete C; Replace D; Create A; Create B; Create C (new)**
With **Replace D** := {Delete D; Create D}
or {Create D; Delete D}

Plan de reconfiguration

Arbre de dépendances



Plan de déploiement

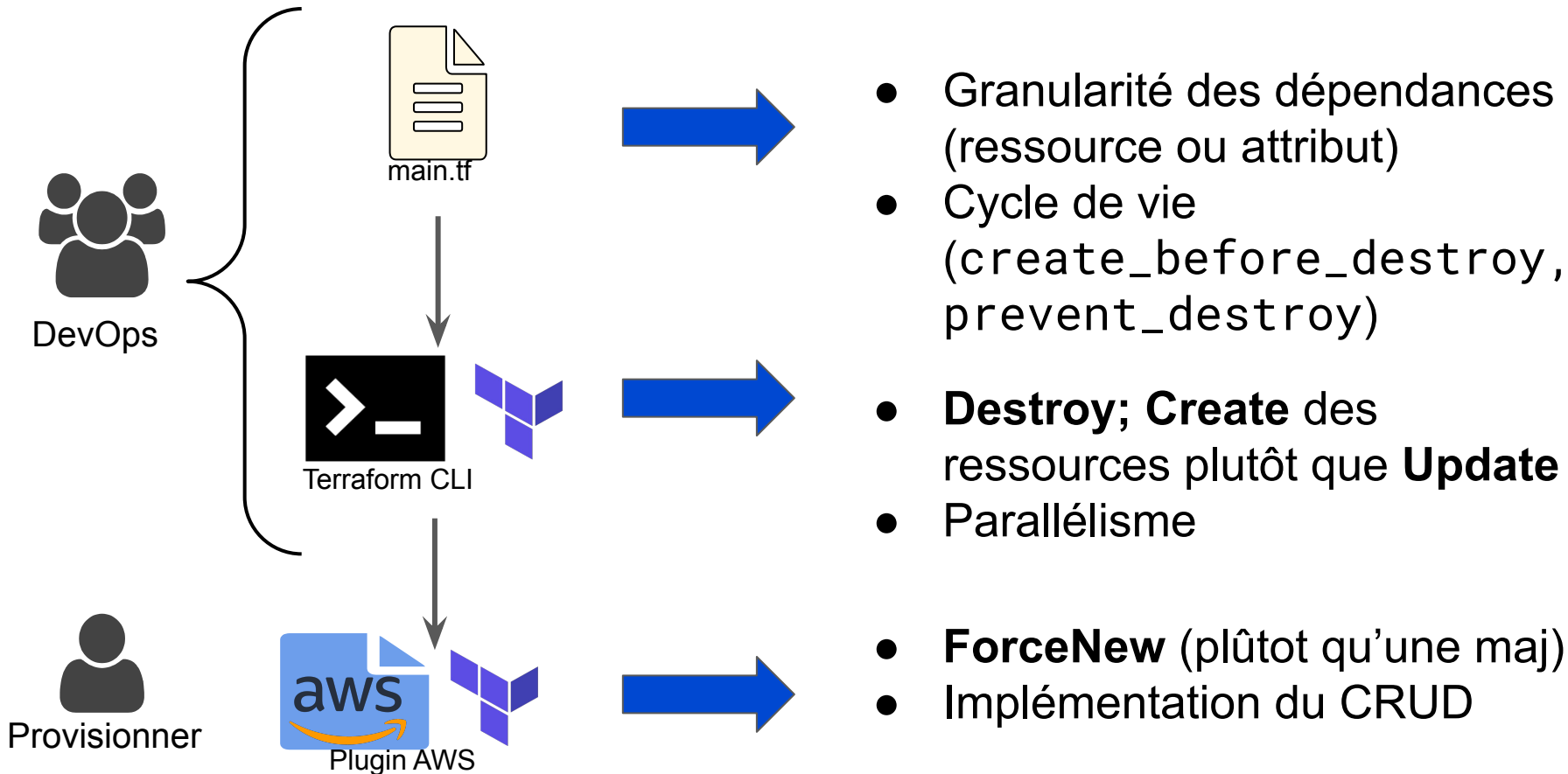
- **Create D; Create A; Create B; Create C**

Plans pour mettre à jour D

- **Update D; Update A; Update B; Update C**
- **Delete A; Delete B; Delete C; Replace D; Create A; Create B; Create C (new)**
With **Replace D** := {Delete D; Create D}
or {Create D; Delete D}

Quelle stratégie adopter ? Quel impact ? Quelles propriétés ?

Grande variabilité des options de déploiement



μProvisioner et BPlan

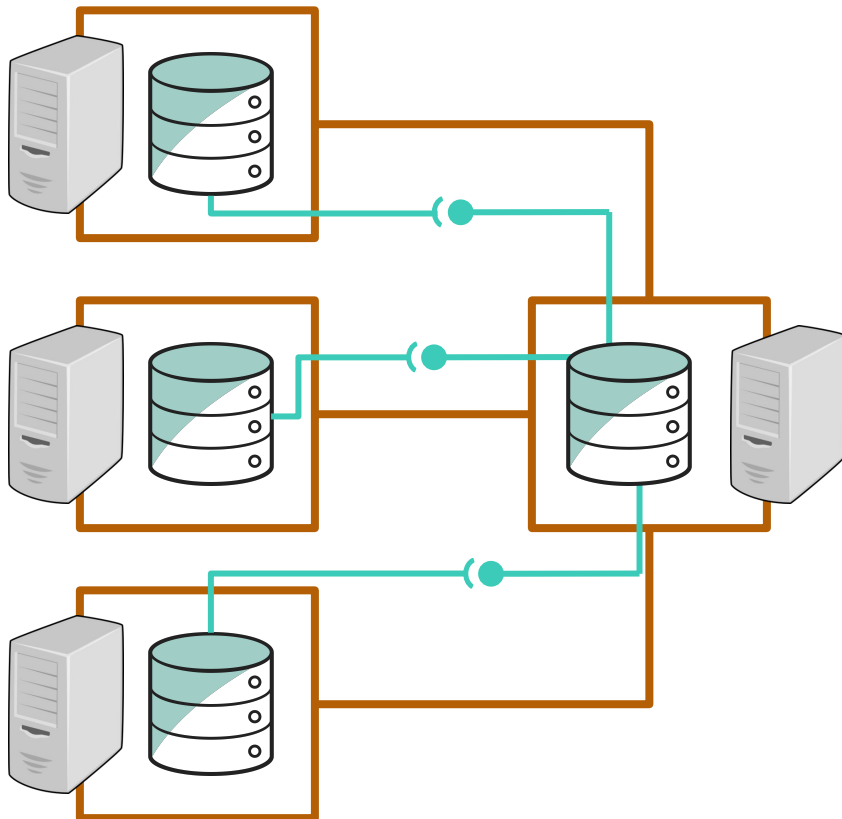
Objectifs

- Permettre un **raisonnement formel** sur les déploiements Terraform pour garantir la **sécurité temporelle** des dépendances entre ressources

Contributions

- Formalisation opérationnelle du comportement de Terraform dans Maude (μProvisioner)
- Propriétés de stabilité des dépendances pendant tout le déploiement.
 - **Stable** : la dépendance est toujours satisfaite.
 - **Estable** : la dépendance est temporairement violée, mais satisfaite à la fin.
 - **Unstable** : la dépendance est absente à la fin \Rightarrow erreur.
- Intégration dans un outil complémentaire (BPlan) à terraform plan

Travaux en cours : vérification dans l'IaC



Déploiement et management d'une base de données distribuées

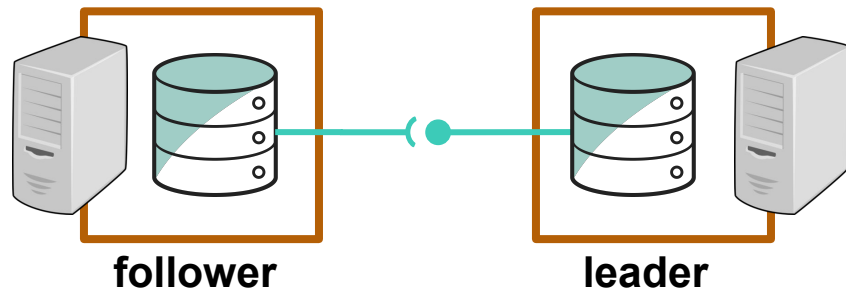
Provisionnement ← **µProvisioner et BPlan**

- Création d'un VPC (subnet)
- Demander des machines
- Définition des VMs/cluster
- Firewall

Configuration management ← **VÉRIFIER ?**

- Installer Redis (bin + deps)
- Configurer les BDDs
- Bootstrap les BDDs
- Intégrer followers et leader

Exemple de déploiement



Plan de déploiement du follower (PF)

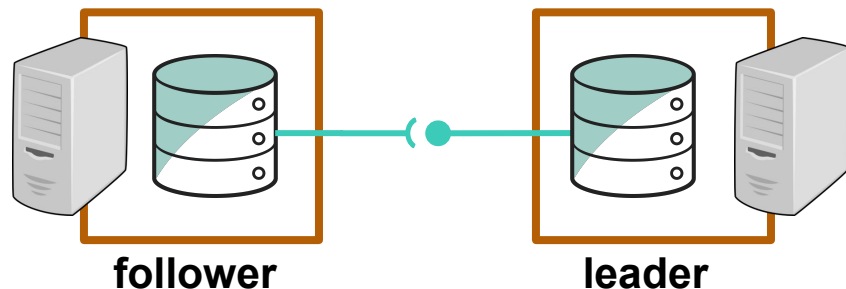
1. Configurer le service
2. S'enregistrer sur le leader
3. Bootstrap la base de données
4. Démarrer le service
5. Exposer son API

Plan de déploiement du leader (PL)

1. Configurer le service
2. Bootstrap la base de données
3. Démarrer le service
4. Exposer son API

- **Granularité composant** : Deploy PL << Deploy PF
- **Granularité cycle de vie** : PL (4) << PF (2)

Exemple de mise à jour



Plan de déploiement du follower (PF)

1. Interrompre le service
2. Faire la mise à jour
3. S'enregistrer sur le leader
4. Démarrer le service
5. Exposer son API

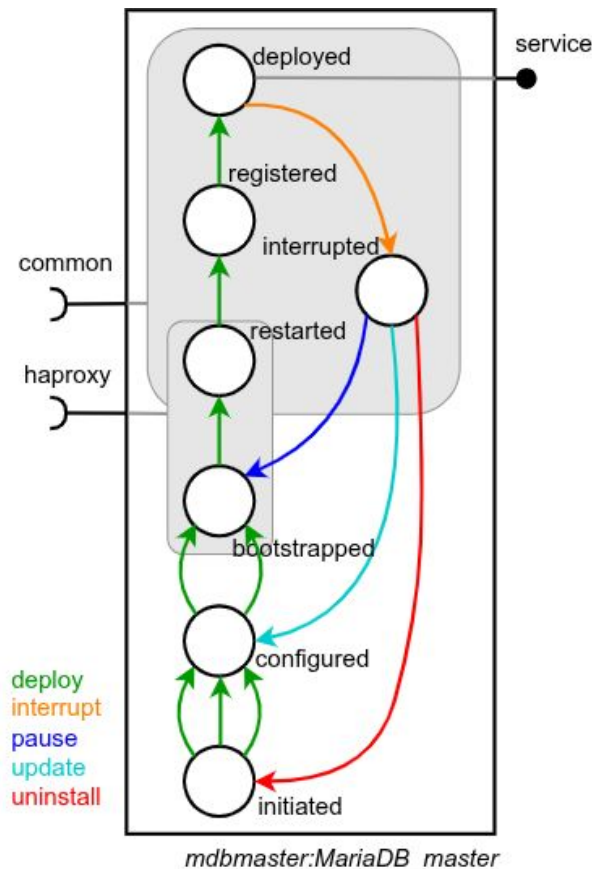
Plan de déploiement du leader (PL)

1. Interrompre le service
2. Faire la mise à jour
3. Démarrer le service
4. Exposer son API

- **Granularité composant** : Destroy PF << Update PL << Create PF
- **Granularité cycle de vie** : PF (1) << PL (1) ; PL (4) << PF (3)

Concerto - Un modèle de reconfiguration

- **Places** : Étapes dans la reconfiguration
- **Comportements** : Interface (gros grains) des actions
- **Transitions** : Actions concrètes (scripts) entre les places, associées à un comportement.
- **Ports** : Fourni (resp. utilise) des informations. Deux types : *provide port*, et *use port*. Les ports sont associés à des transitions et des places.



Concerto - Un modèle de reconfiguration

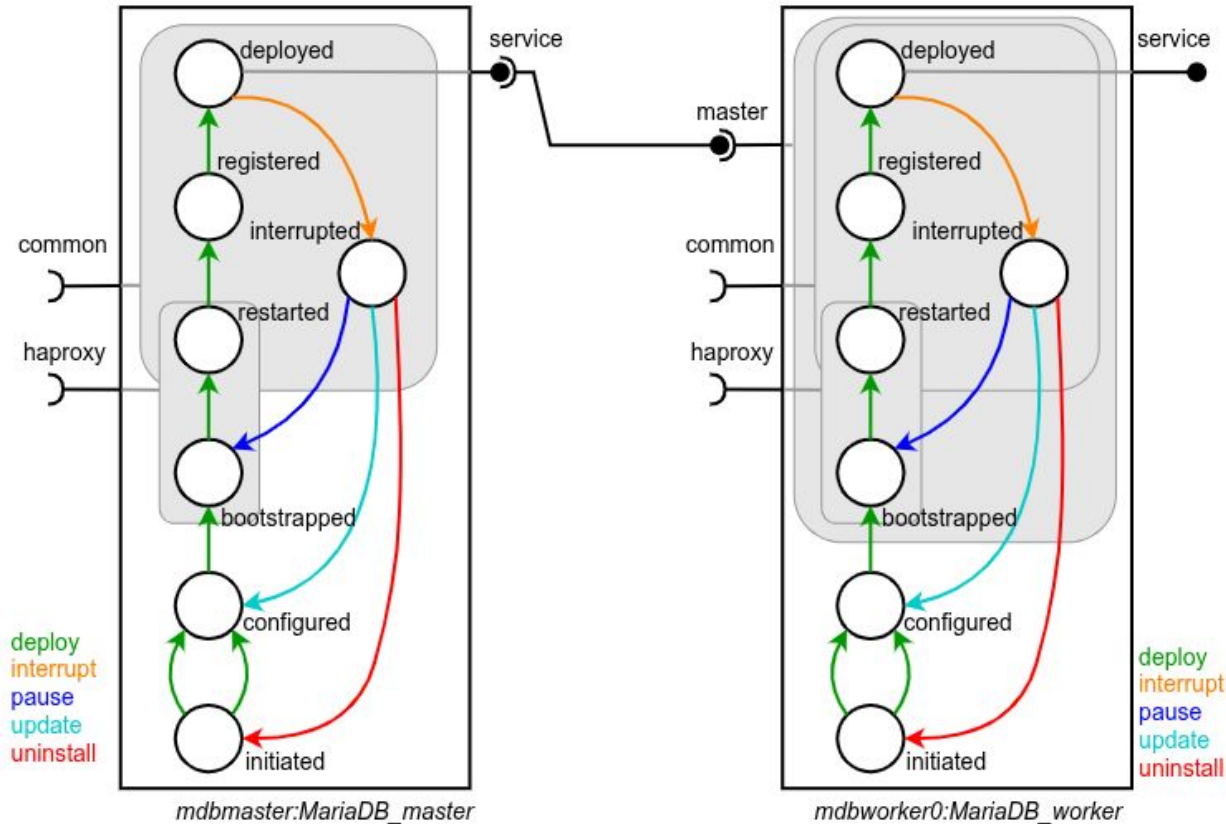
```
class MariaDB_Master(Component):
    def create(self):
        self.places = [ "initiated", "configured", "bootstrapped", "restarted", "registered", "deployed", "interrupted"]
        self.transitions = {
            "configure0": ("initiated", "configured", "deploy", self.configure0),
            "configure1": ("initiated", "configured", "deploy", self.configure1),
            "configure2": ("initiated", "configured", "deploy", self.configure2),
            ...
        }
        self.dependencies = {
            "service": (DepType.PROVIDE, ["deployed"]),
            "haproxy": (DepType.USE, ["bootstrapped", "restarted"]),
            ...
        }
        self.initial_place = 'initiated'
        self.running_place = 'deployed'

    def configure0(self):
        # concrete actions

    def configure1(self):
        # concrete actions

    ...
```

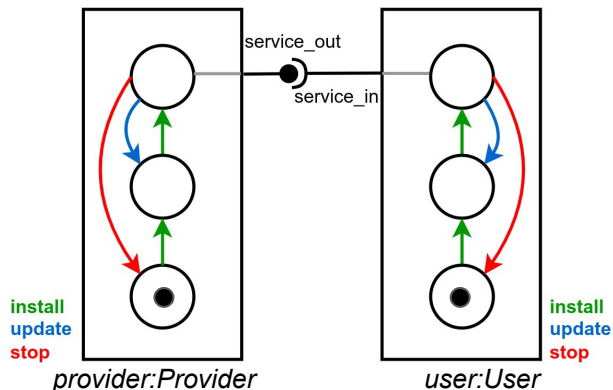
Concerto - Un modèle de reconfiguration



Concerto - Interface utilisateur

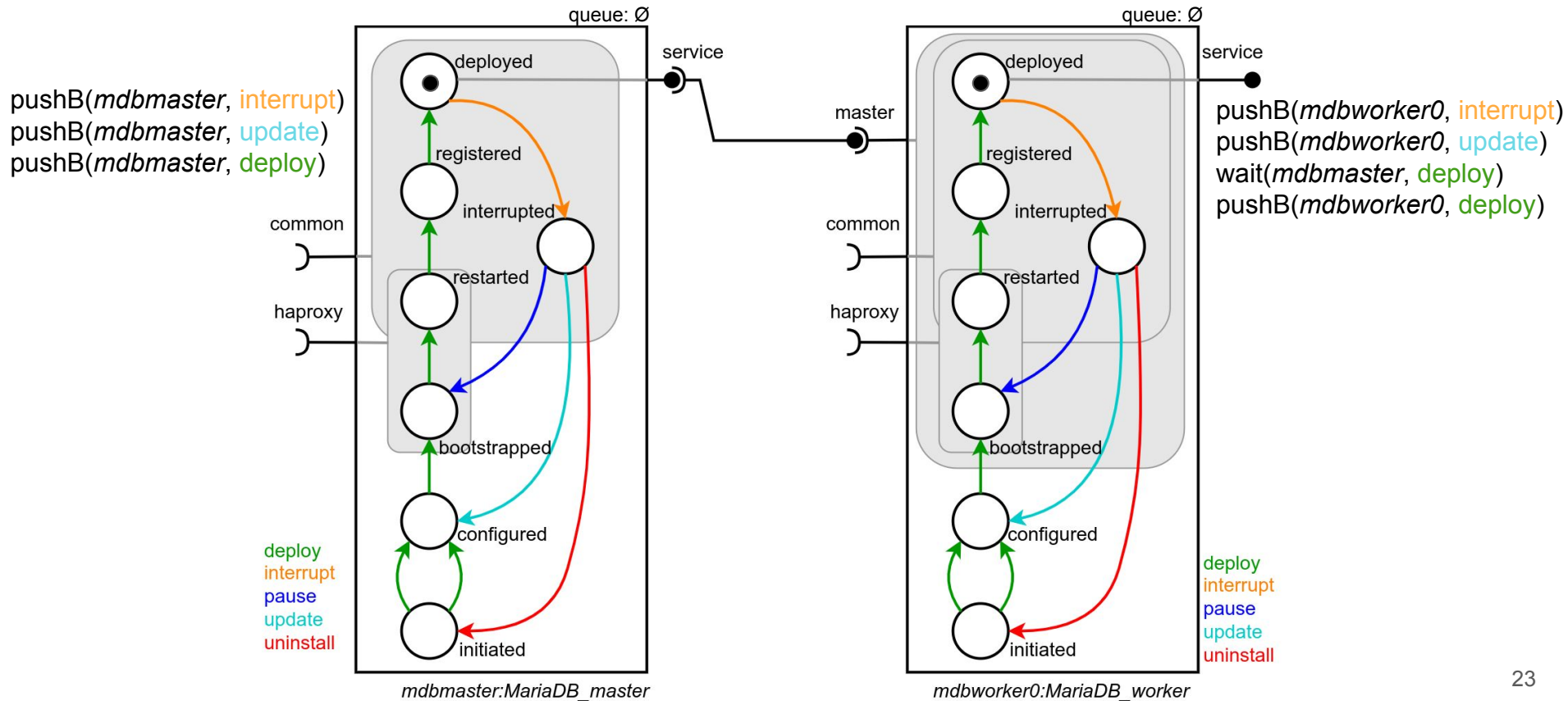
Un **petit language** pour exprimer une reconfiguration dans **Concerto**

- Modifier un assemblage: **add**, **remove**
- Connections entre les composants: **connect**, **disconnect**
- Appliquer des comportements: **pushB** (non bloquant)
- Barrière de synchro. explicite: **wait**

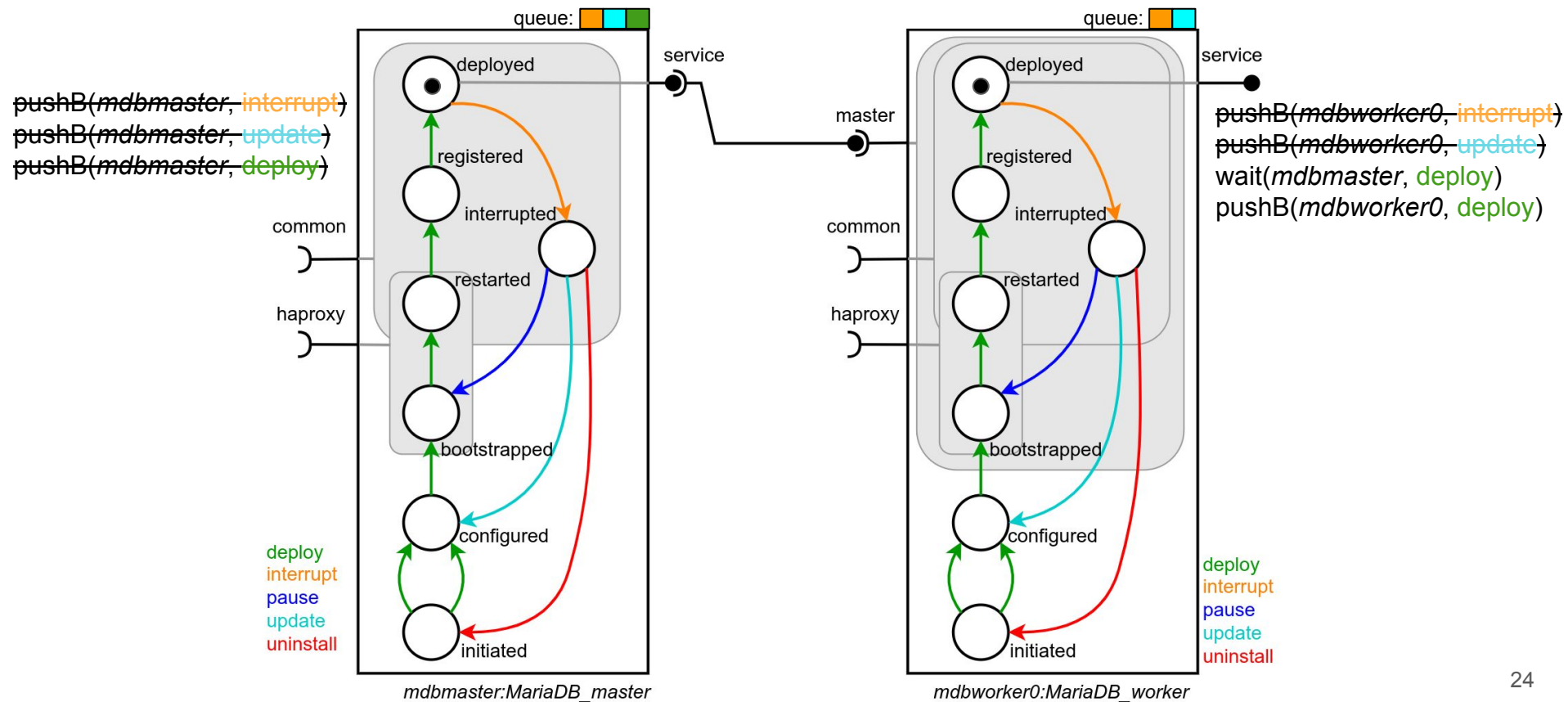


```
add(provider, Provider)
add(user, User)
connect(provider, service_out, user, service_in)
pushB(provider, install)
pushB(user, install)
wait(user, install)
```

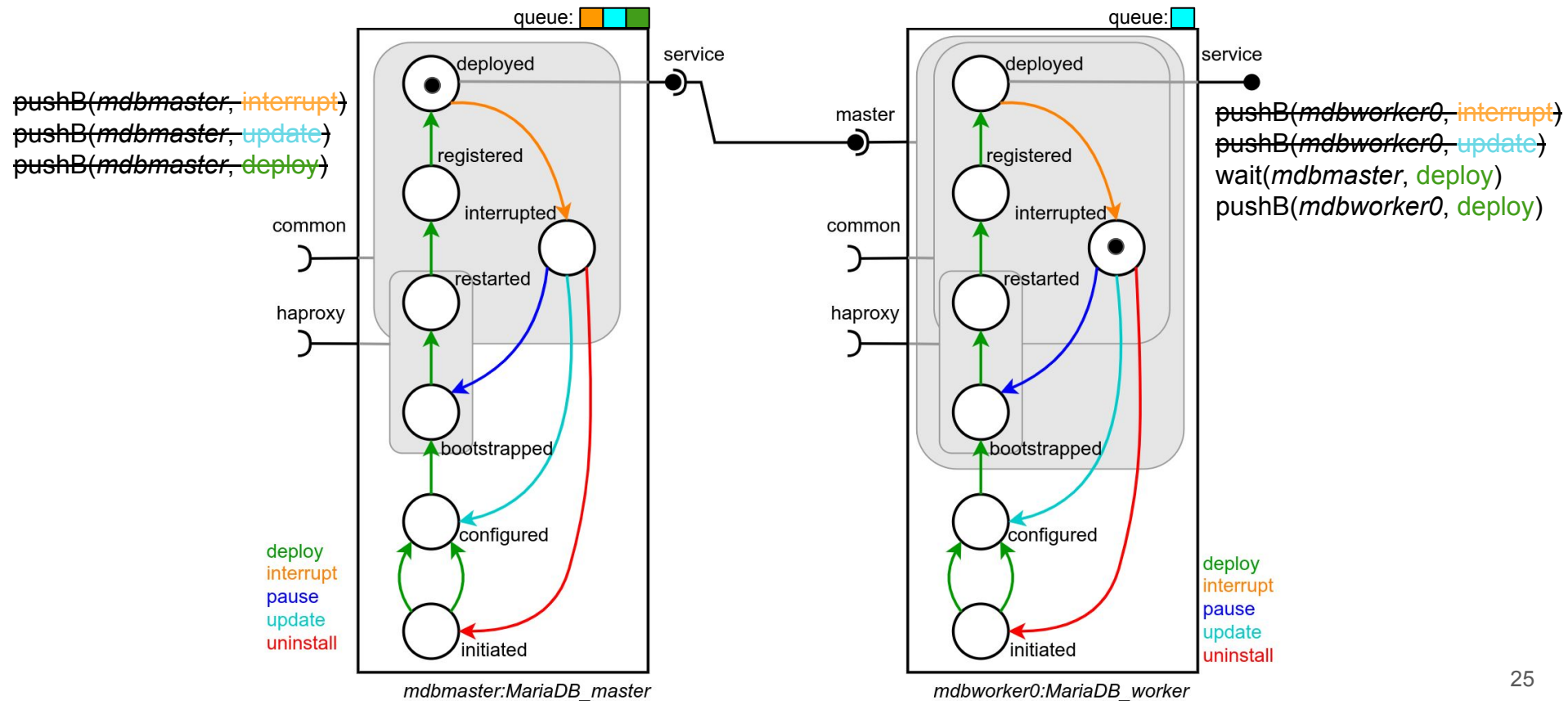
Exemple d'exécution



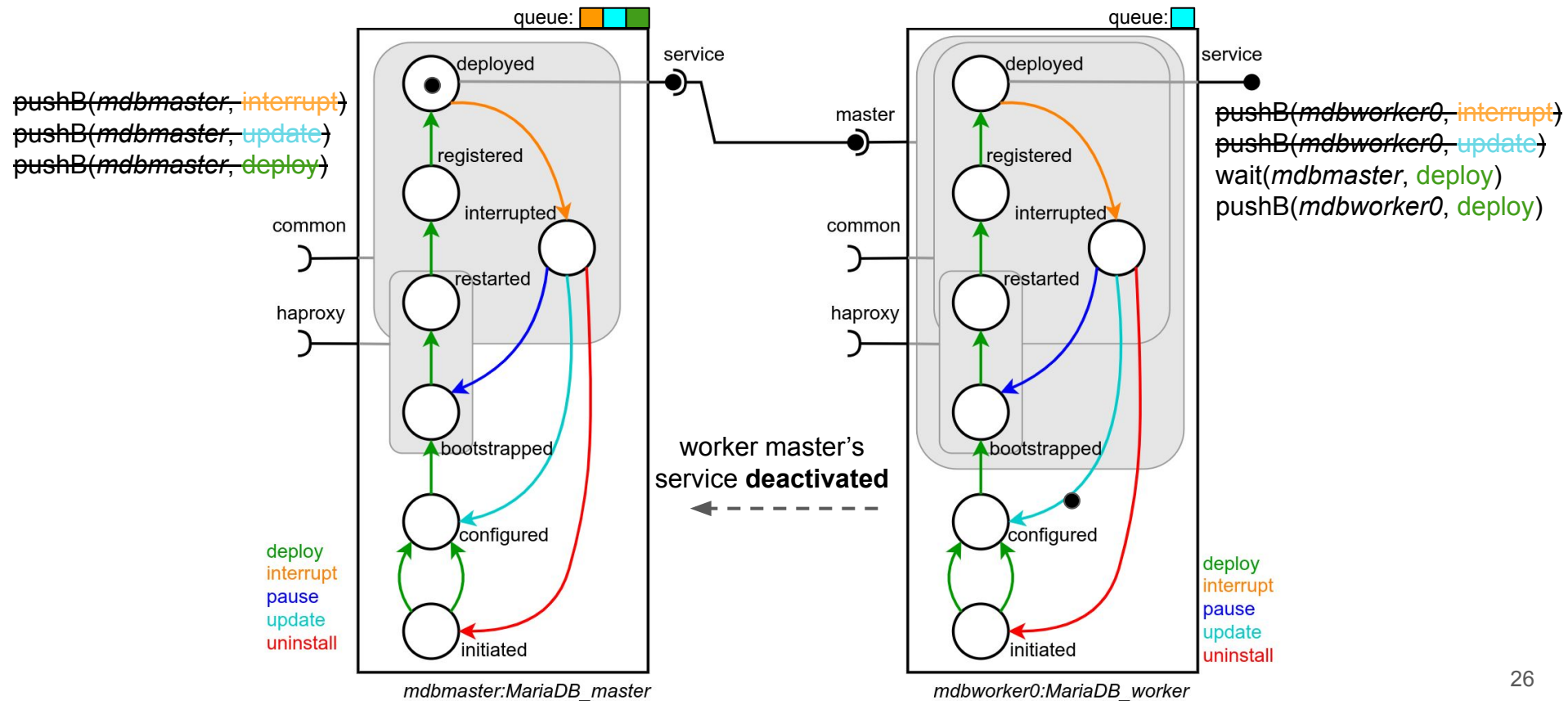
Exemple d'exécution



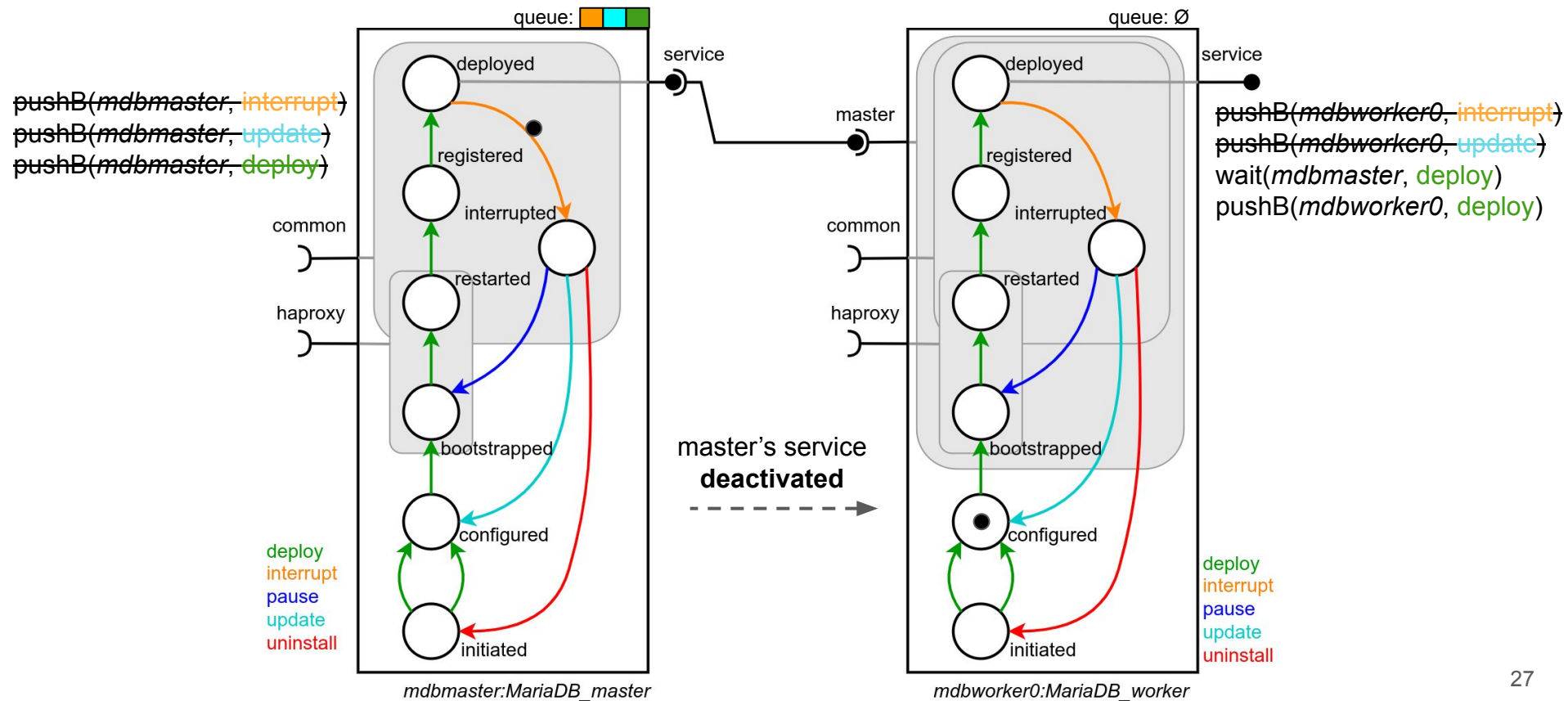
Exemple d'exécution



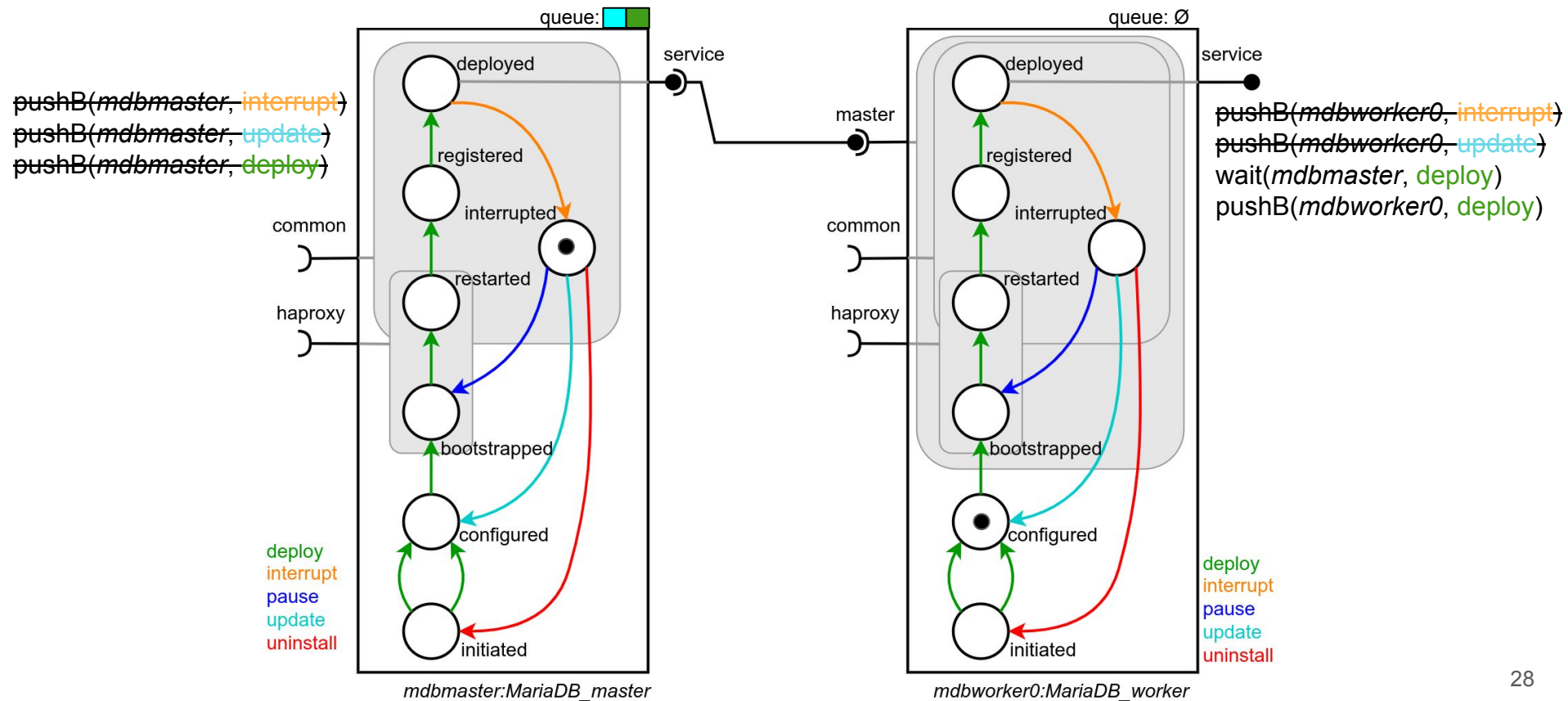
Exemple d'exécution



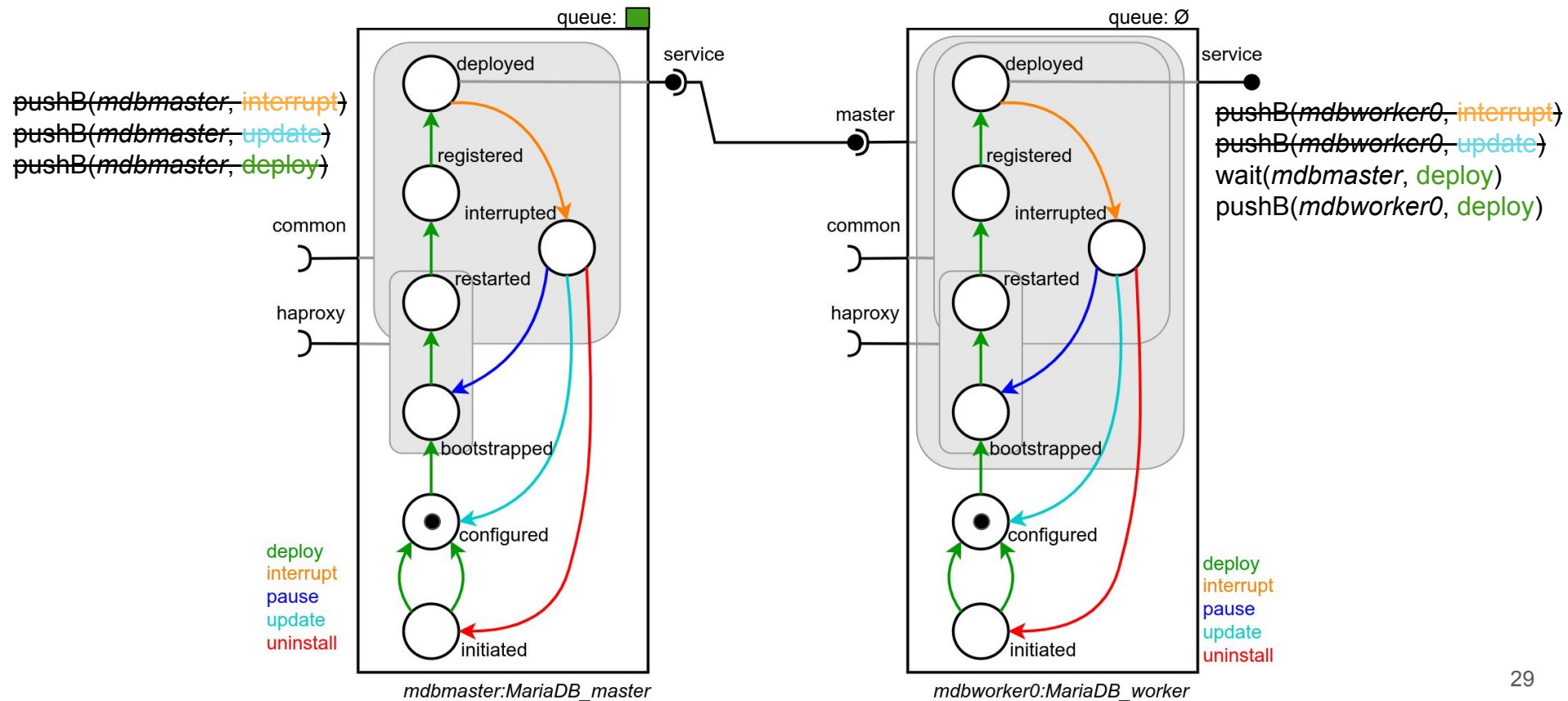
Exemple d'exécution



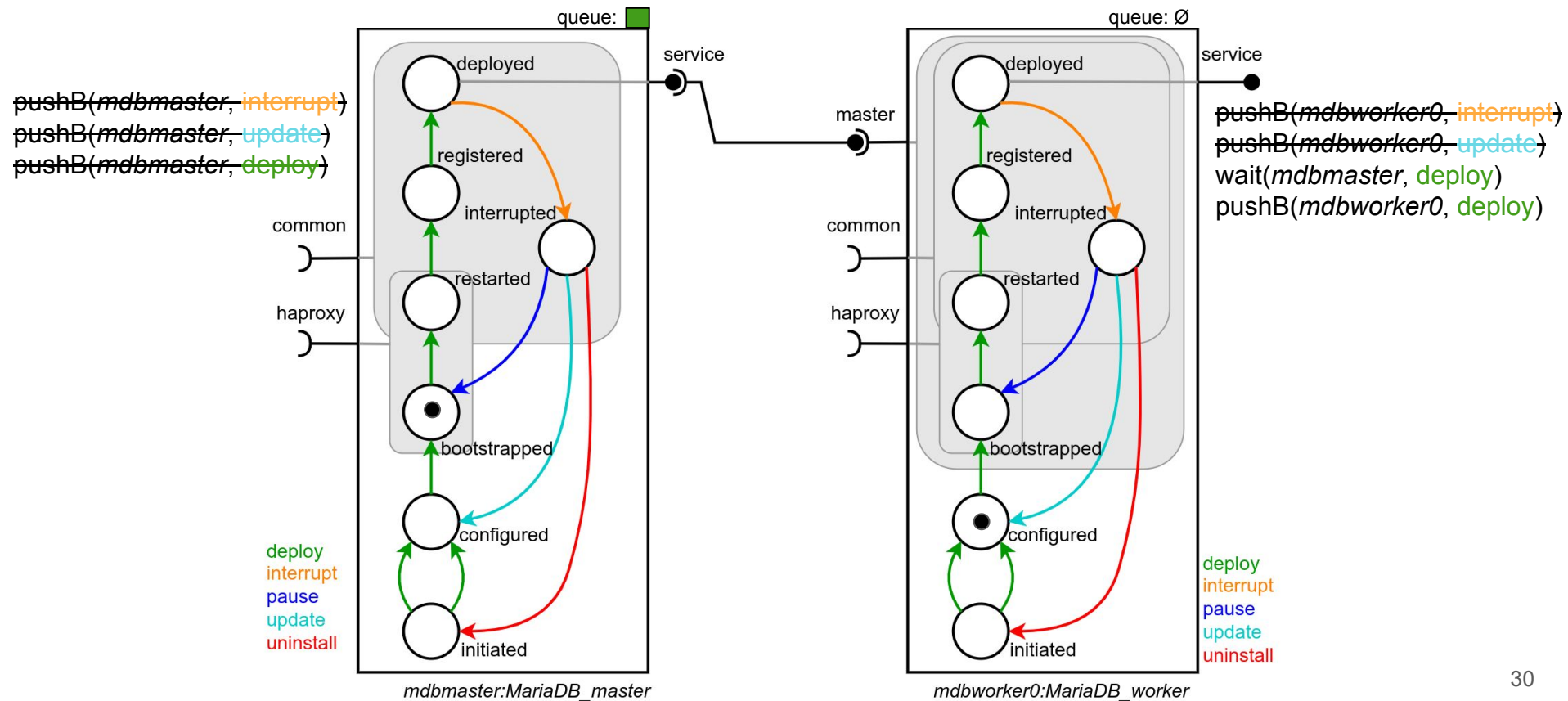
Exemple d'exécution



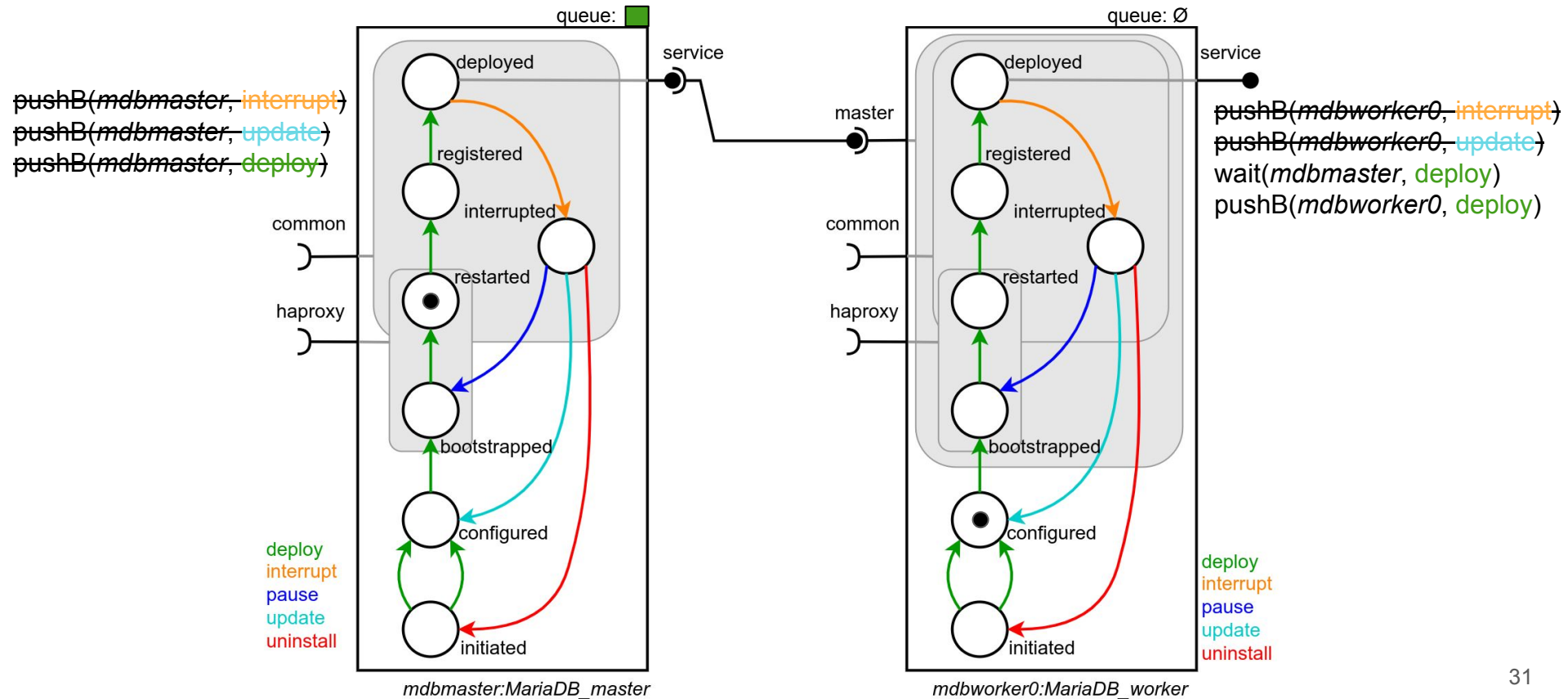
Exemple d'exécution



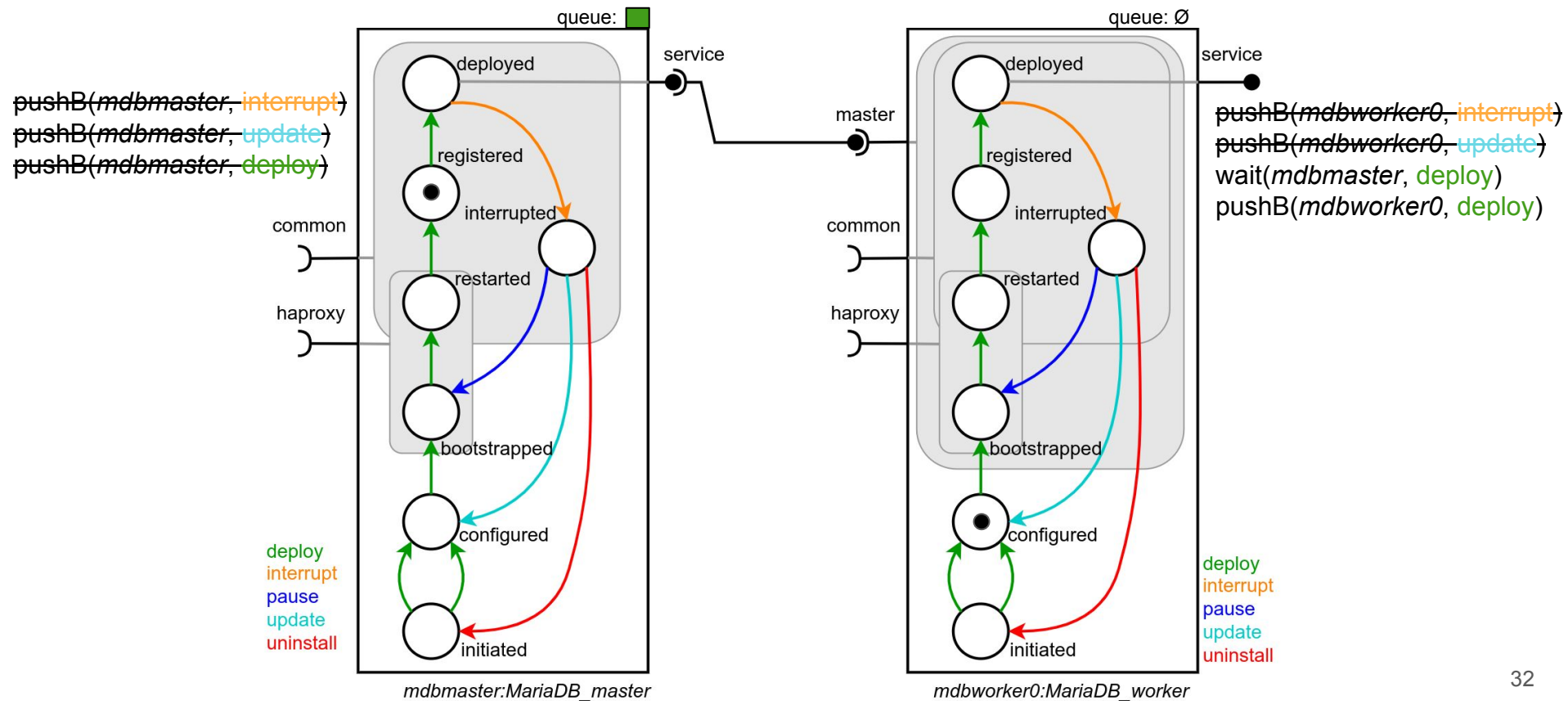
Exemple d'exécution



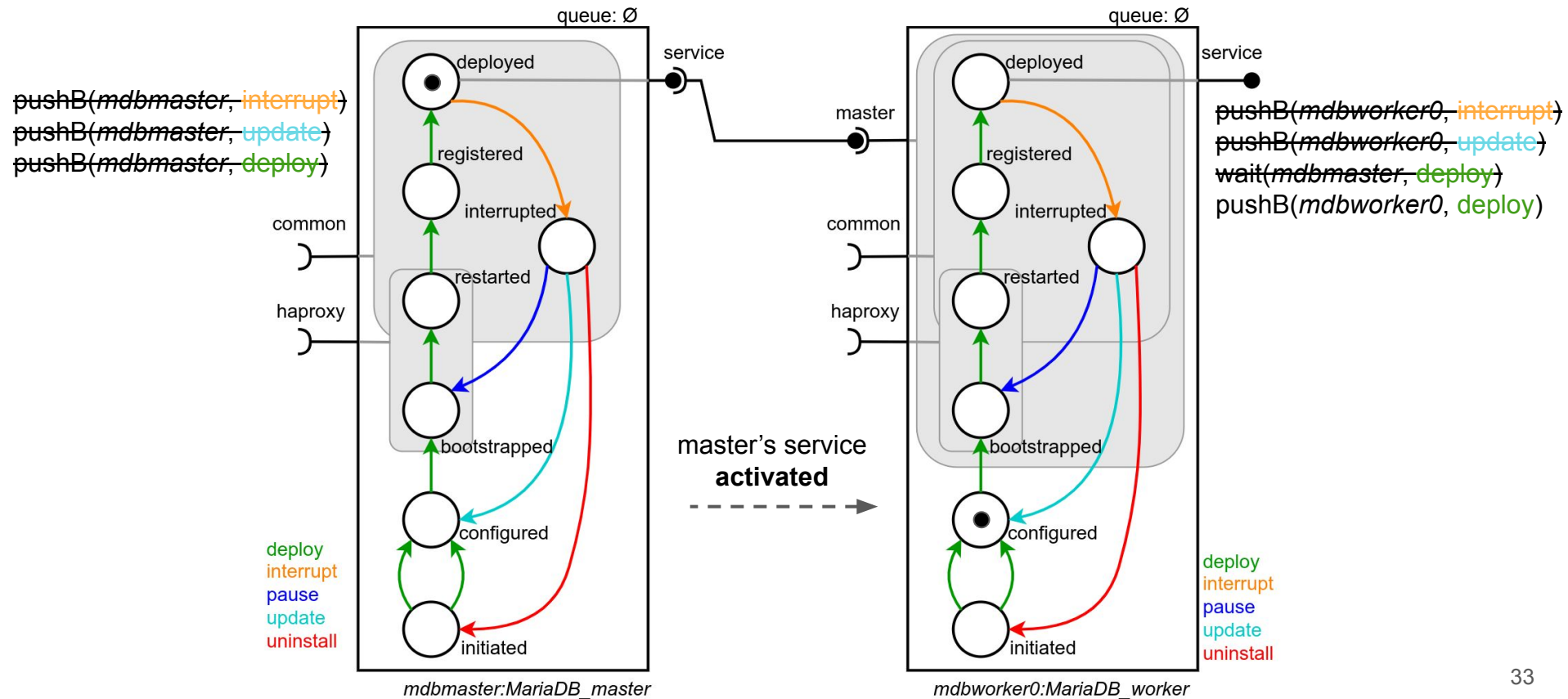
Exemple d'exécution



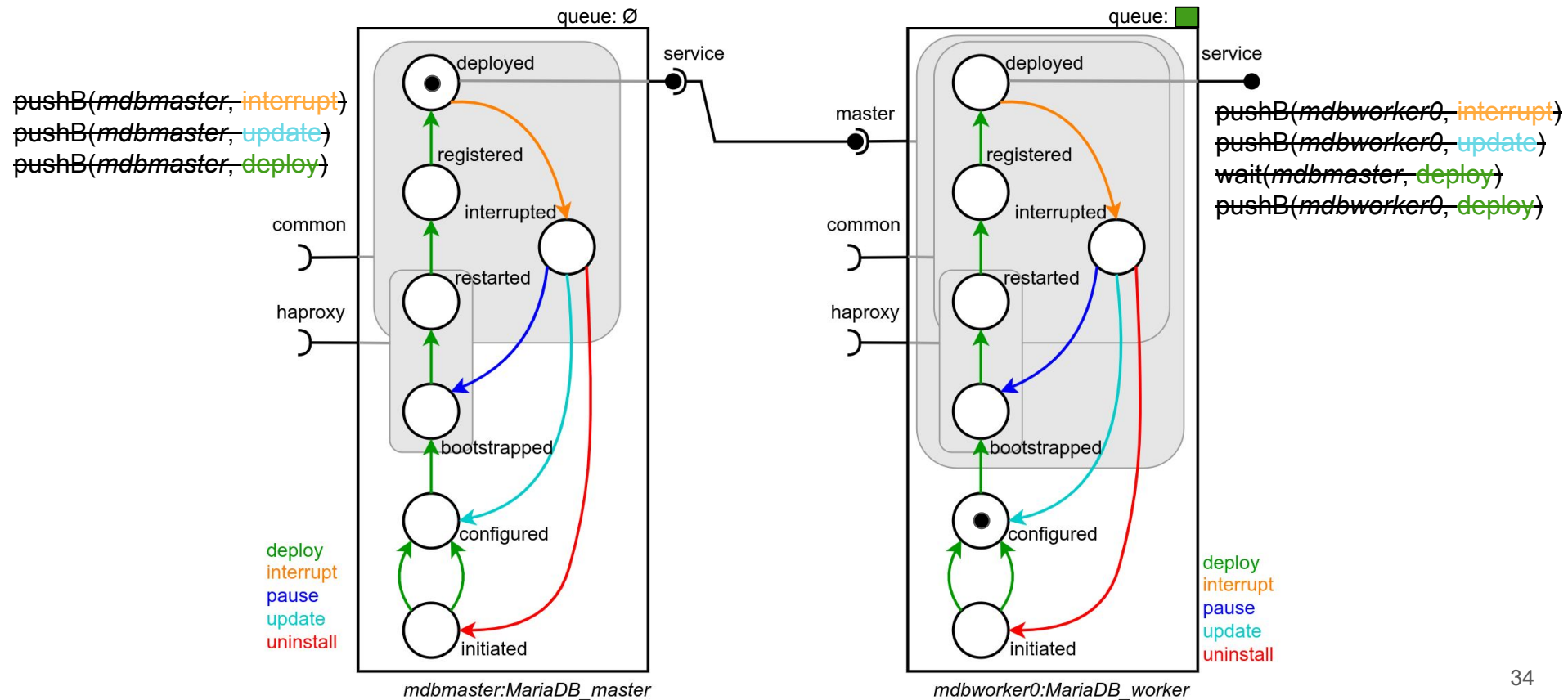
Exemple d'exécution



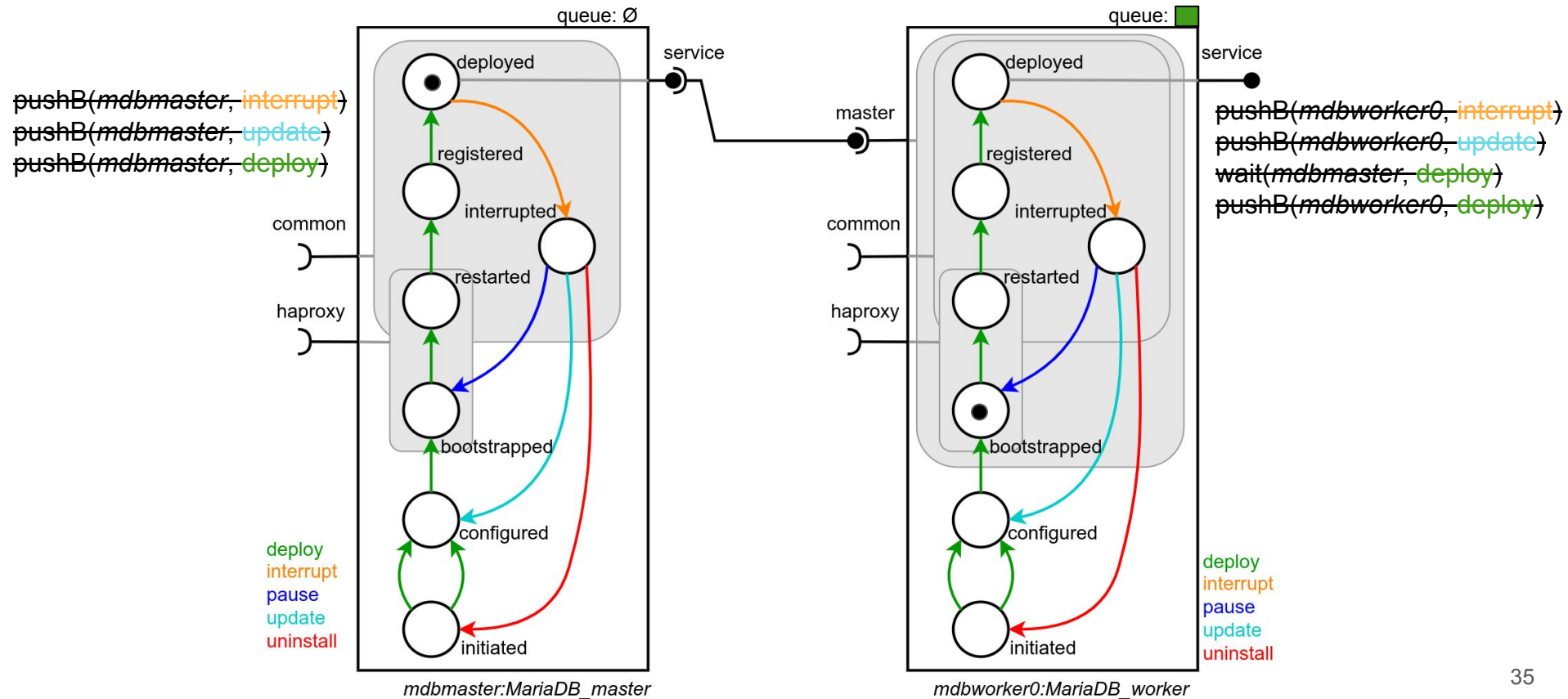
Exemple d'exécution



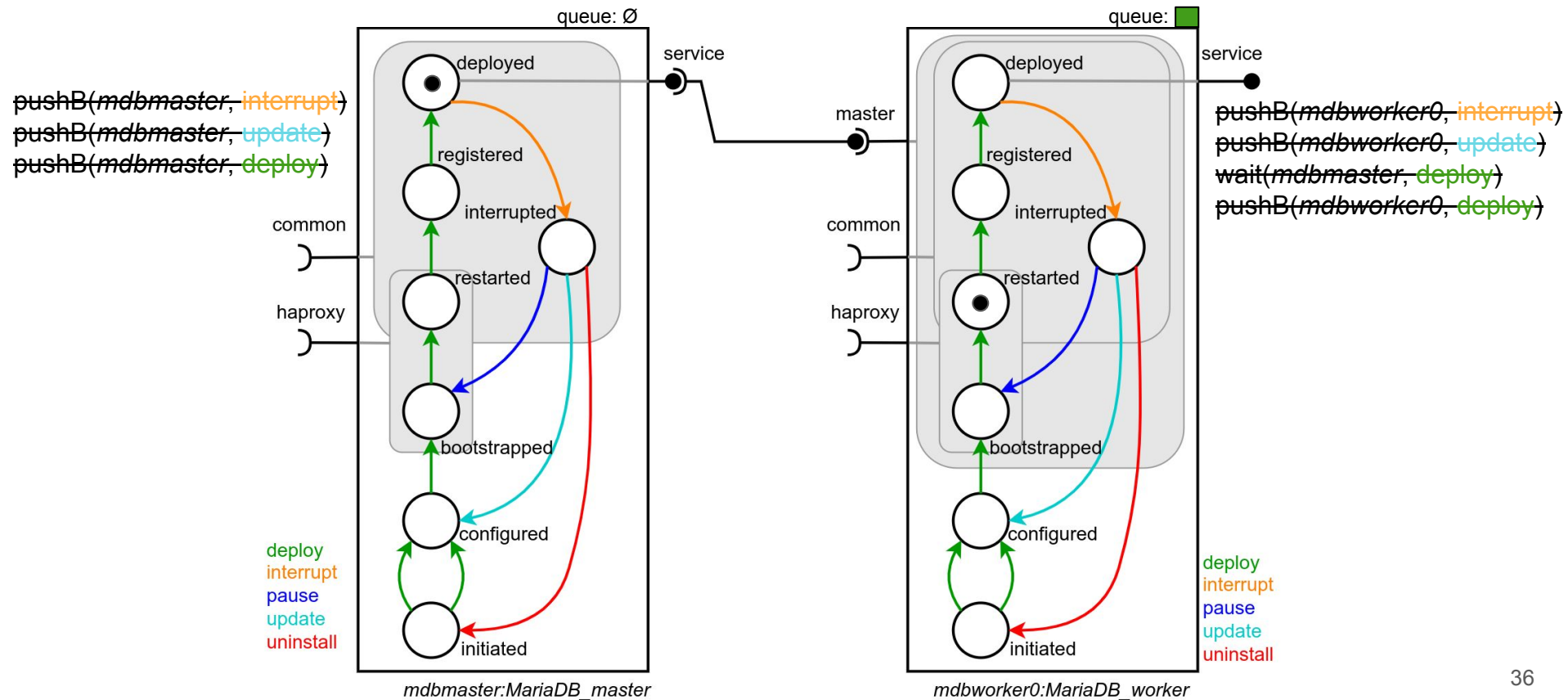
Exemple d'exécution



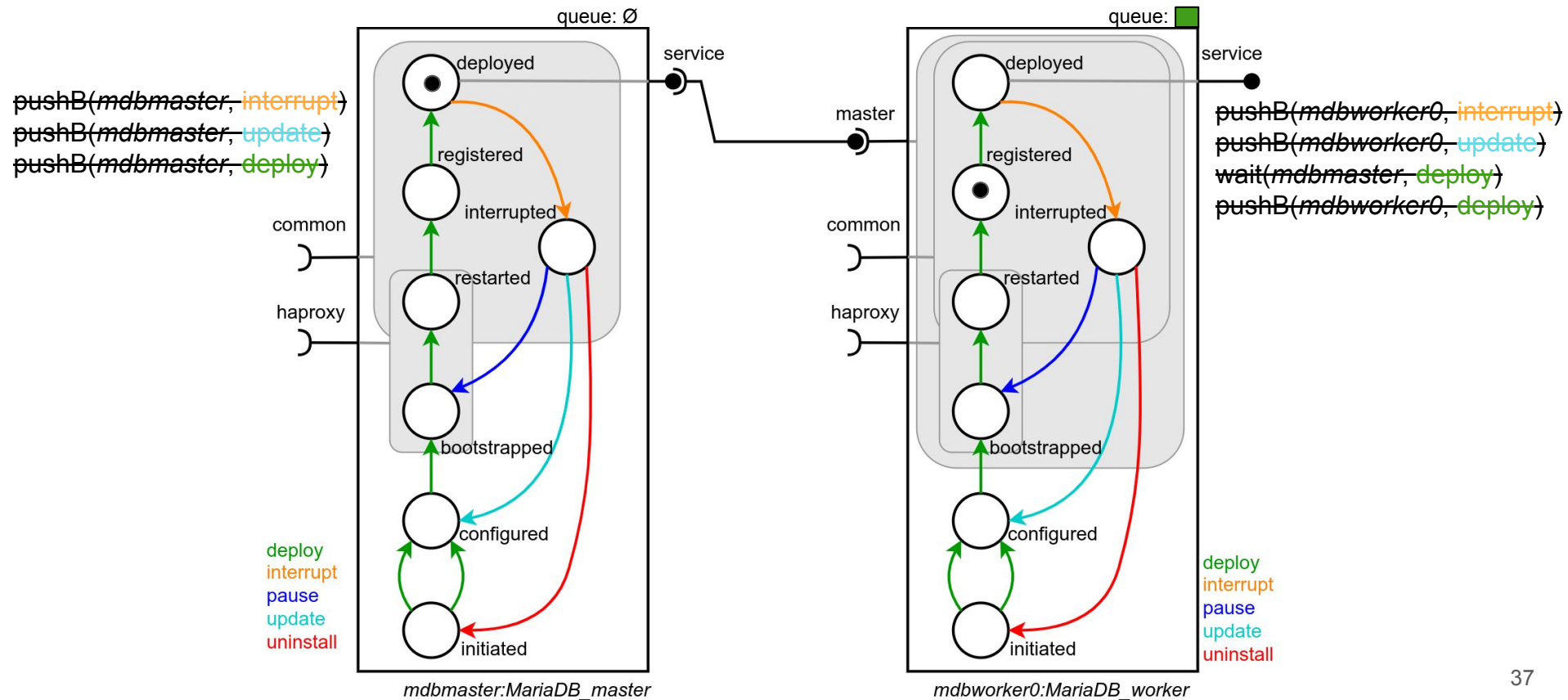
Exemple d'exécution



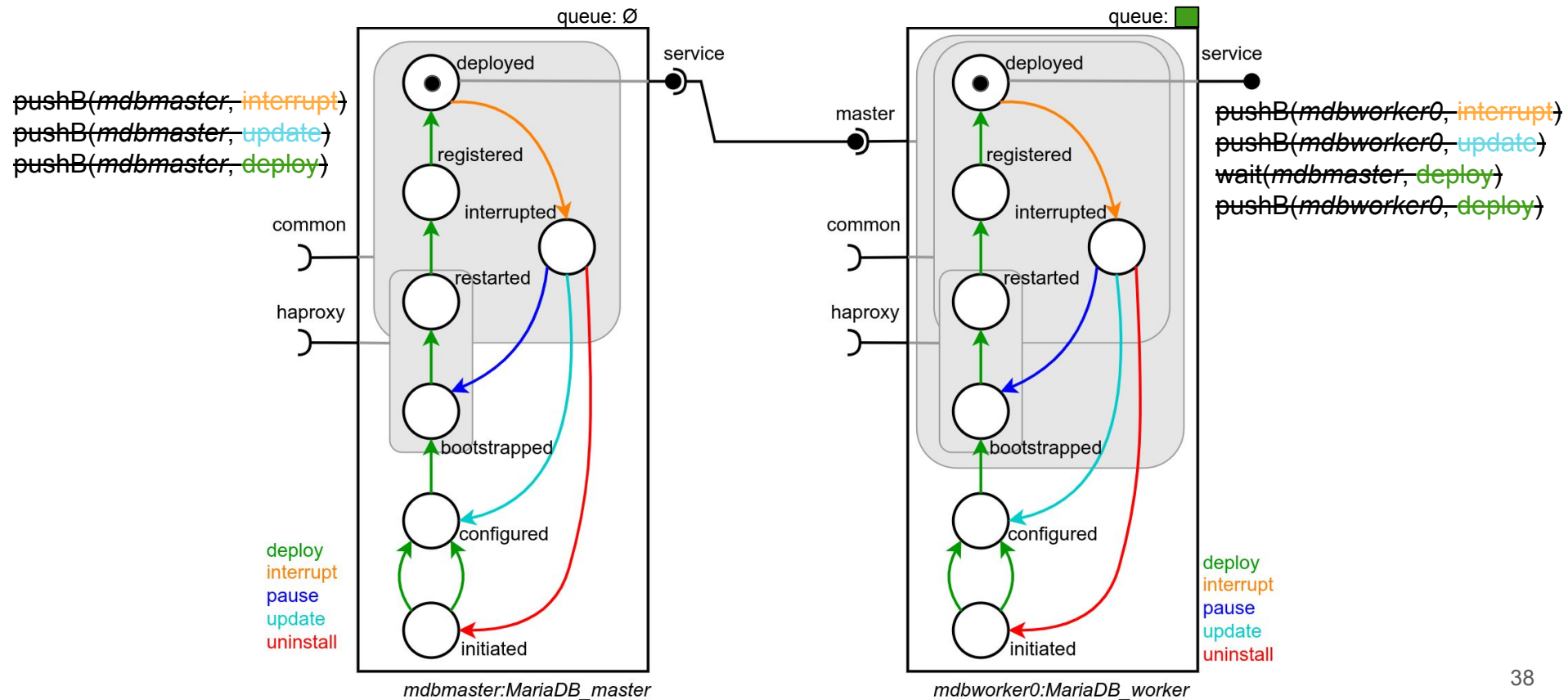
Exemple d'exécution



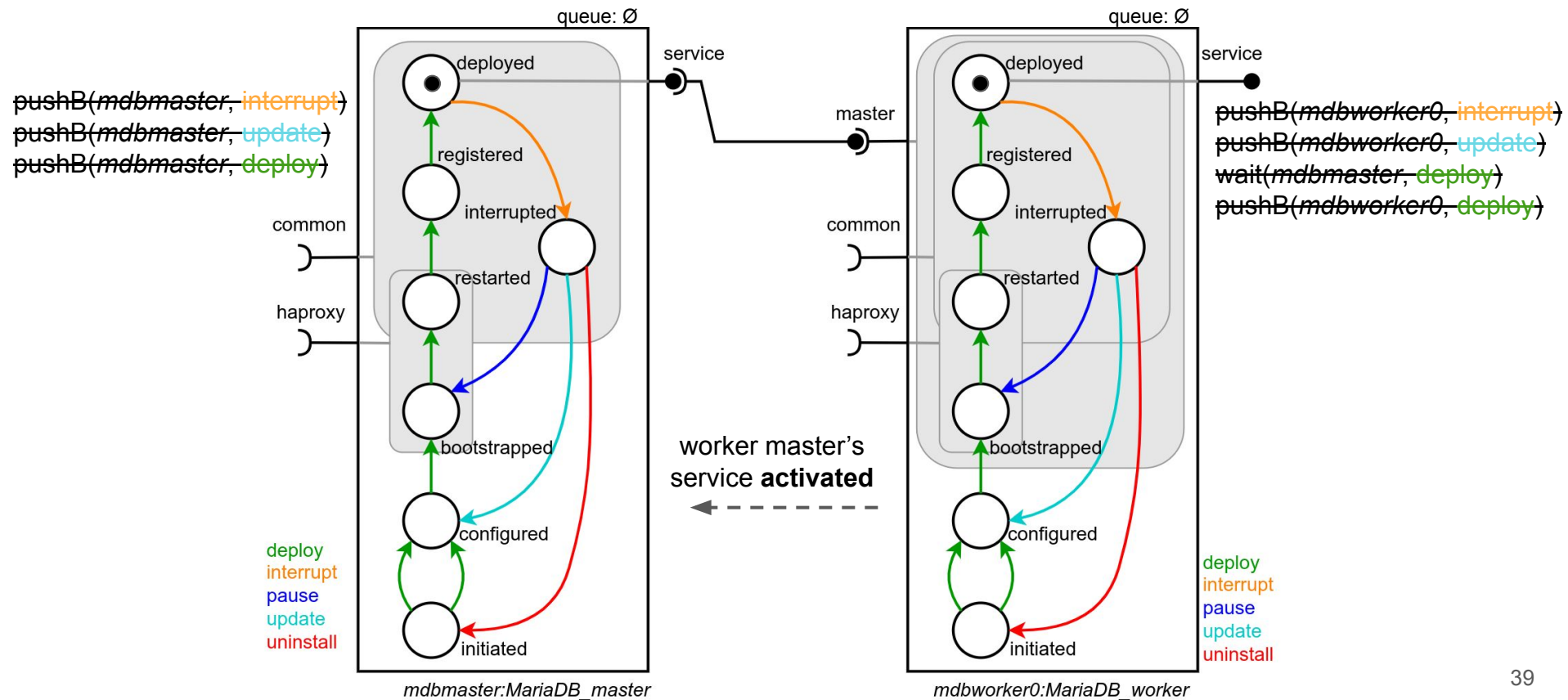
Exemple d'exécution



Exemple d'exécution

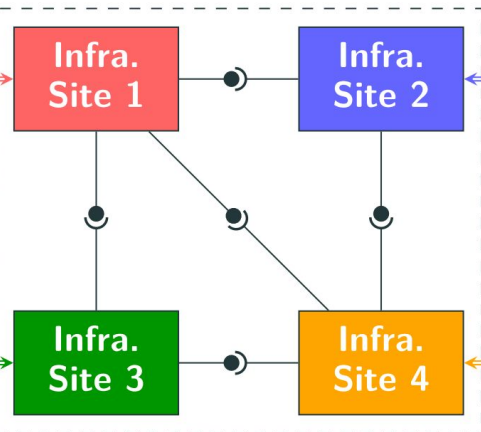
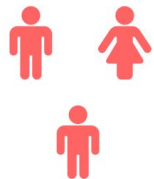


Exemple d'exécution

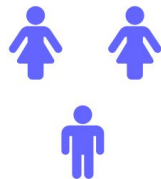


Décentralisation de Concerto : Concerto-D

DevOps team



DevOps team



DevOps team



Multi-site infrastructure

➤ Décentralisation de la reconfiguration.
1 noeud = 1 programme

➤ Communications explicites inter-composants

➤ Plus adapté aux environnements contraints, plus de robustesse

Sémantique opérationnelle et model-checking

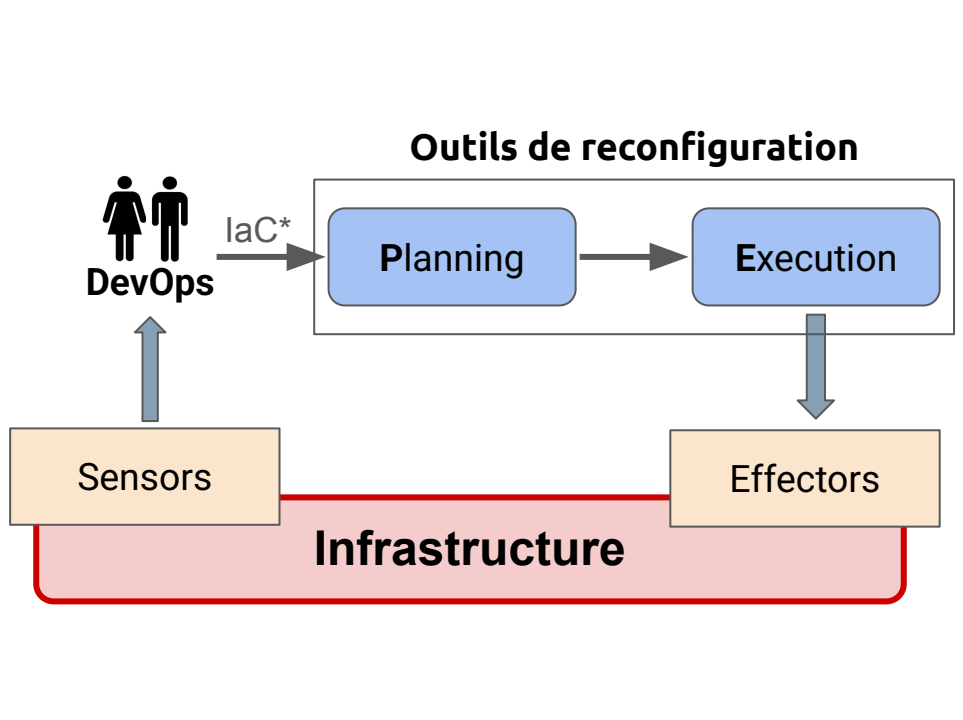
Objectifs

- Fournir une sémantique formelle, mécanisée et exécutable de Concerto-D pour raisonner sur l'exécution des comportements

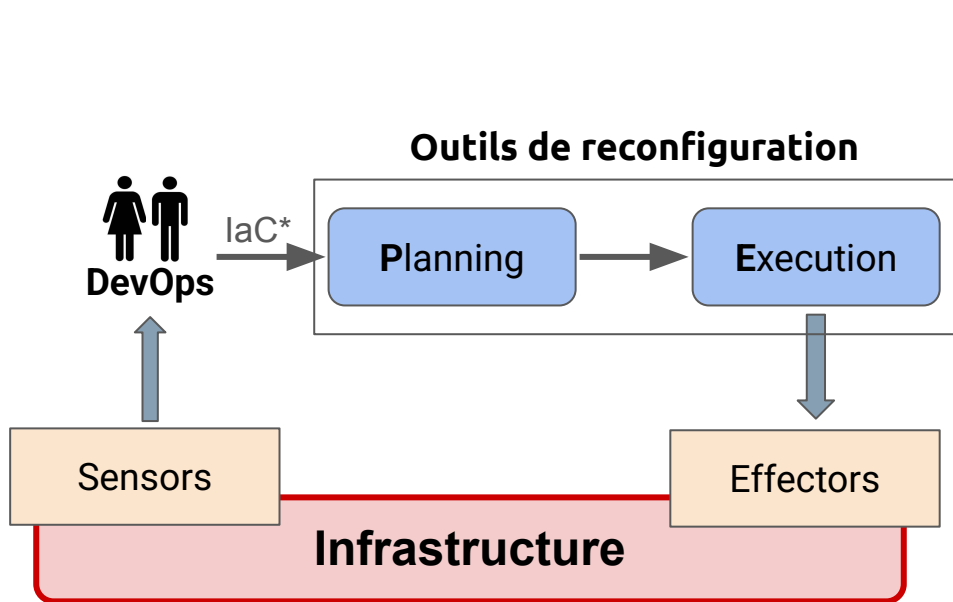
Contributions

- Sémantique formelle de CONCERTO-D
- Modèle de communication : sync. inter-nœuds (e.g. statut de port, fin de comportement).
- Implémentation en Maude pour faire du model-checking avec LTL

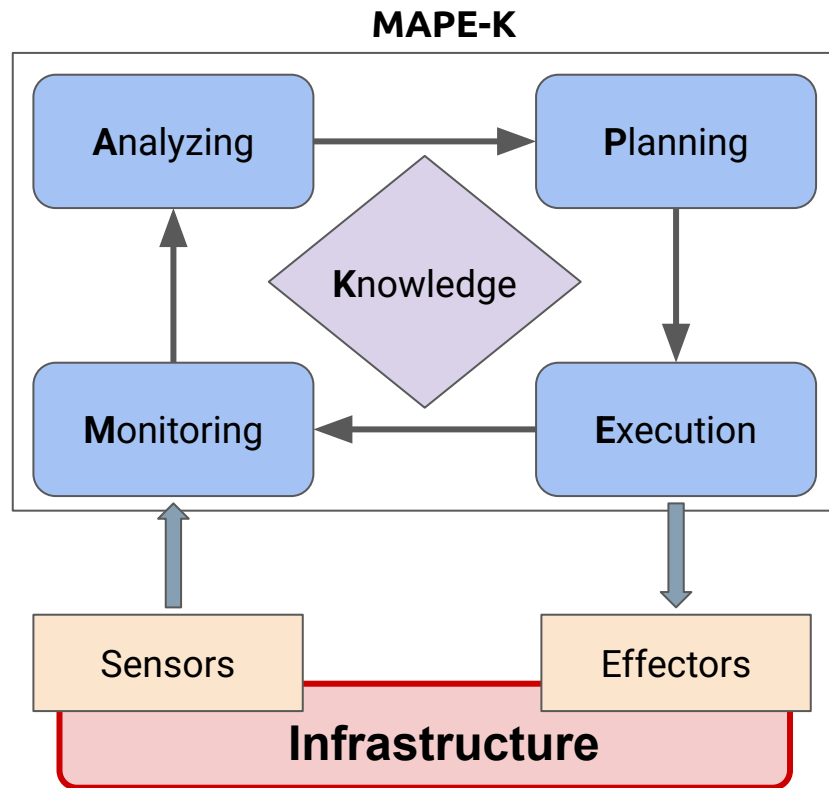
Vers des reconfigurations autonomes



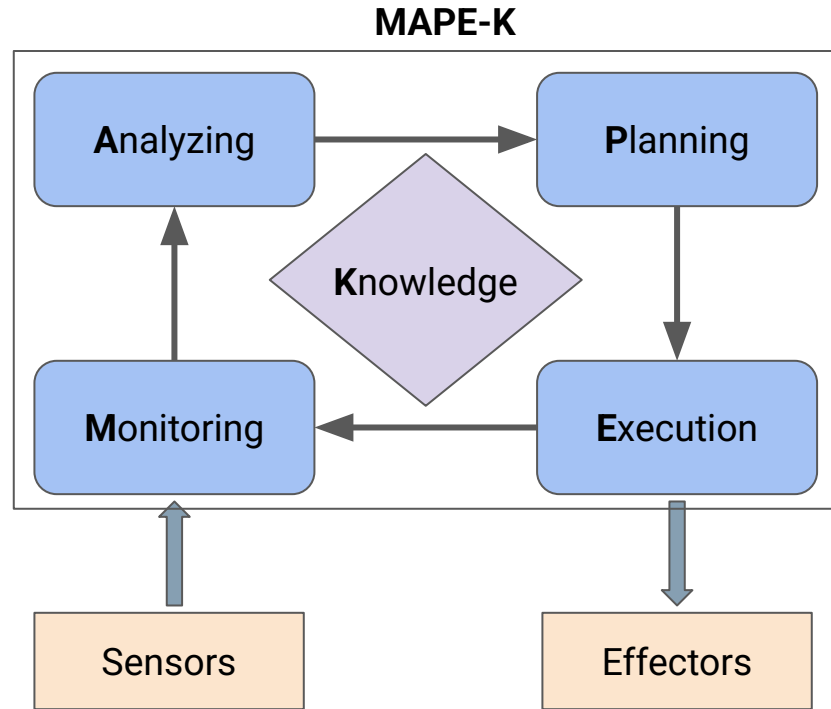
Vers des reconfigurations autonomes







*Infrastructure-as-code






Reconfiguration autonome de systèmes



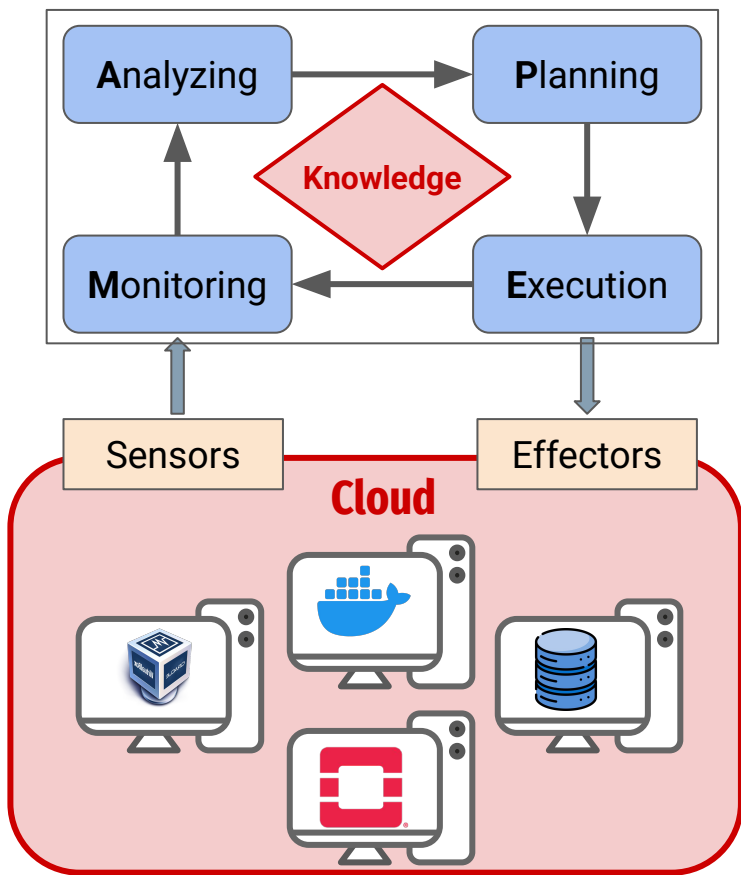
Boucle MAPE

-  **Monitoring** : Surveillance du système
-  **Analyzing** : Analyse de l'état, définition d'un état cible en fonction des ressources
-  **Planning** : Synthèse des actions de reconfiguration
-  **Execution** : Mise en œuvre des actions

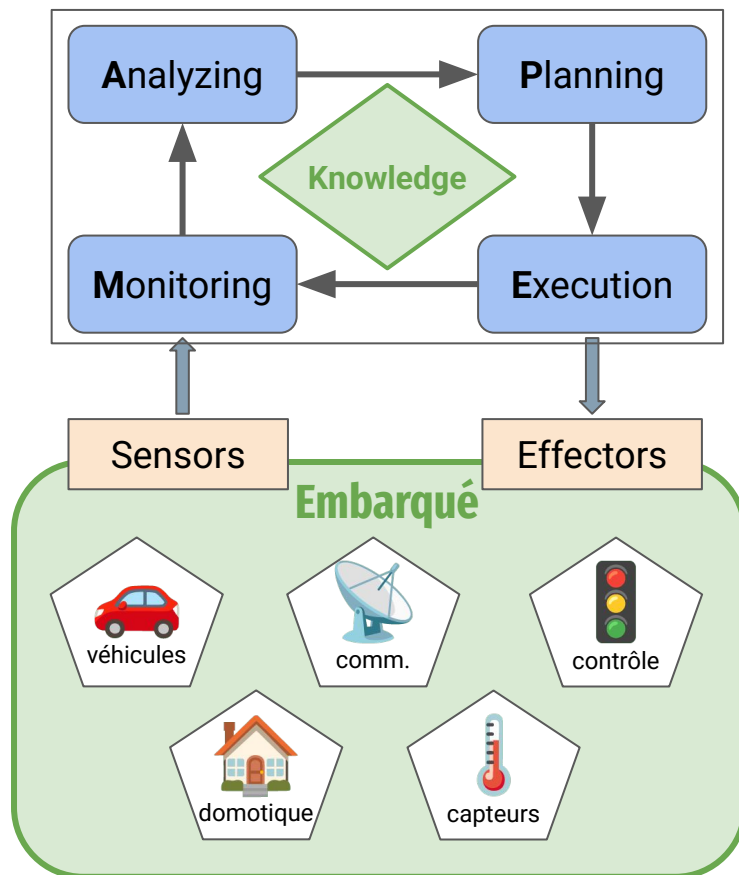
Knowledge

-  **Topologie d'architecture** avec un ADL (Architecture Description Language)
-  **Modèle** de consommation
-  **Contraintes**

Projet de recherche

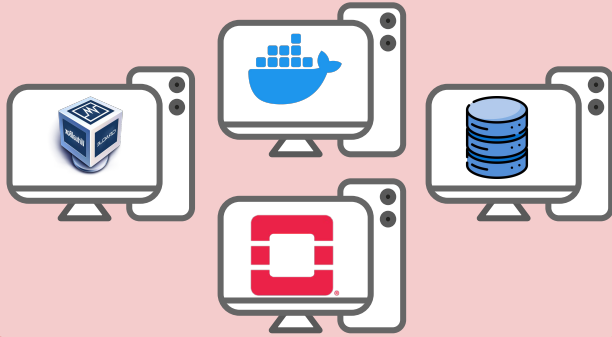


≠

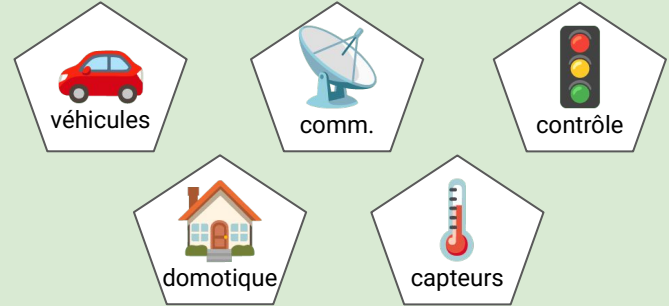


Projet de recherche

Cloud



Embarqué



Knowledge MAPE-K

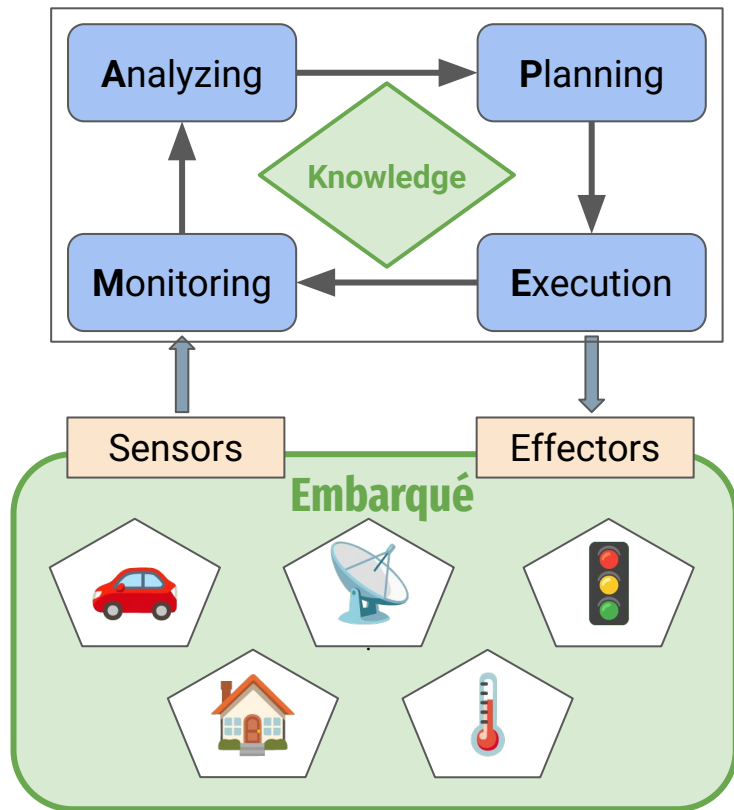
- Nombre de noeuds physiques/virtuels
- Placement de services
- Capacité des noeuds
- Cycles de vie
- ...



Knowledge MAPE-K

- Topologie matérielle
- Contrainte temporelles
- Consommation énergétique
- Politique de reconfiguration
- ...

Reconfiguration sûre pour les systèmes embarqués



Problématiques



Base **K** trop pauvre. Manque d'uniformité et de formalisme :

- des **propriétés extra-fonctionnelles**
- des **événements** endogènes (internes) et exogènes (externes)
- de la **variabilité**
- etc.



Reconfiguration de systèmes critiques

- risque **sécurité**
- **consommation** énergétique
- perte de **cohérence**
- etc.

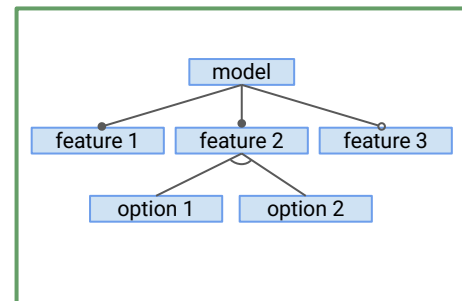
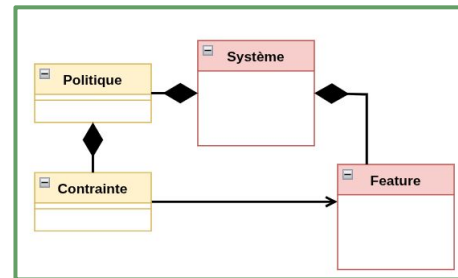
Axe de recherche - Modélisation

Objectif: Enrichissement de la base de connaissance

- Formaliser des **politiques d'adaptation**, à différents grains, pour systèmes embarqués
- Étendre l'**expressivité** autour de la variabilité des systèmes
- Permettre une **analyse, composition** et **vérification** face à des propriétés critiques

Contribution envisagée

- Définition d'un **modèle** (e.g., ADL) unifié + un outillage pour raisonner
 - **Politiques d'adaptation** pour face aux **événements**
- Expression de la variabilité via un **langage**
 - **Verification** d'UVL ?



Axe de recherche - Adaptation

Objectif: Optimisation des décisions d'adaptation

- **Intégrer** le nouveau *knowledge* dans la phase d'**analyse et de plan**
- Prendre des décisions d'**adaptation sûres, efficaces et conformes** aux politiques d'adaptation

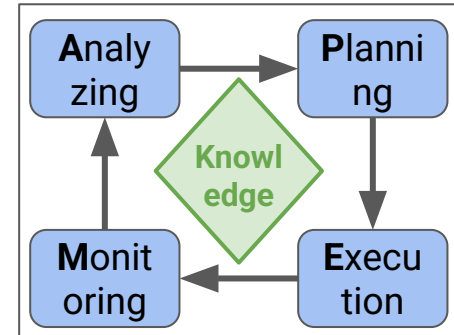
Contribution envisagée

- Considérer le **modèle** de Knowledge pour l'**intégrer** dans des **décisions** de reconf. existants
- **Formaliser** la prise de décision et assurer des **décisions sûres**
- Sur le long terme : Considération de l'**incertitude**; Modèle d'**apprentissage**; Approche probabiliste

Politiques d'adaptation

Exigences
utilisateur

Exigences
système



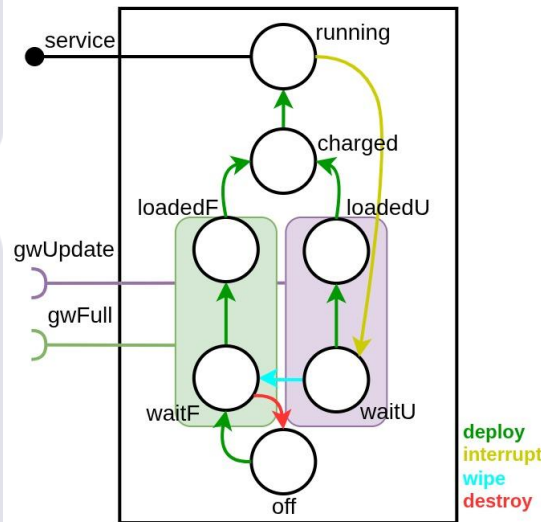
Axe de recherche - Sûreté

Objectif: Garantie de sûreté

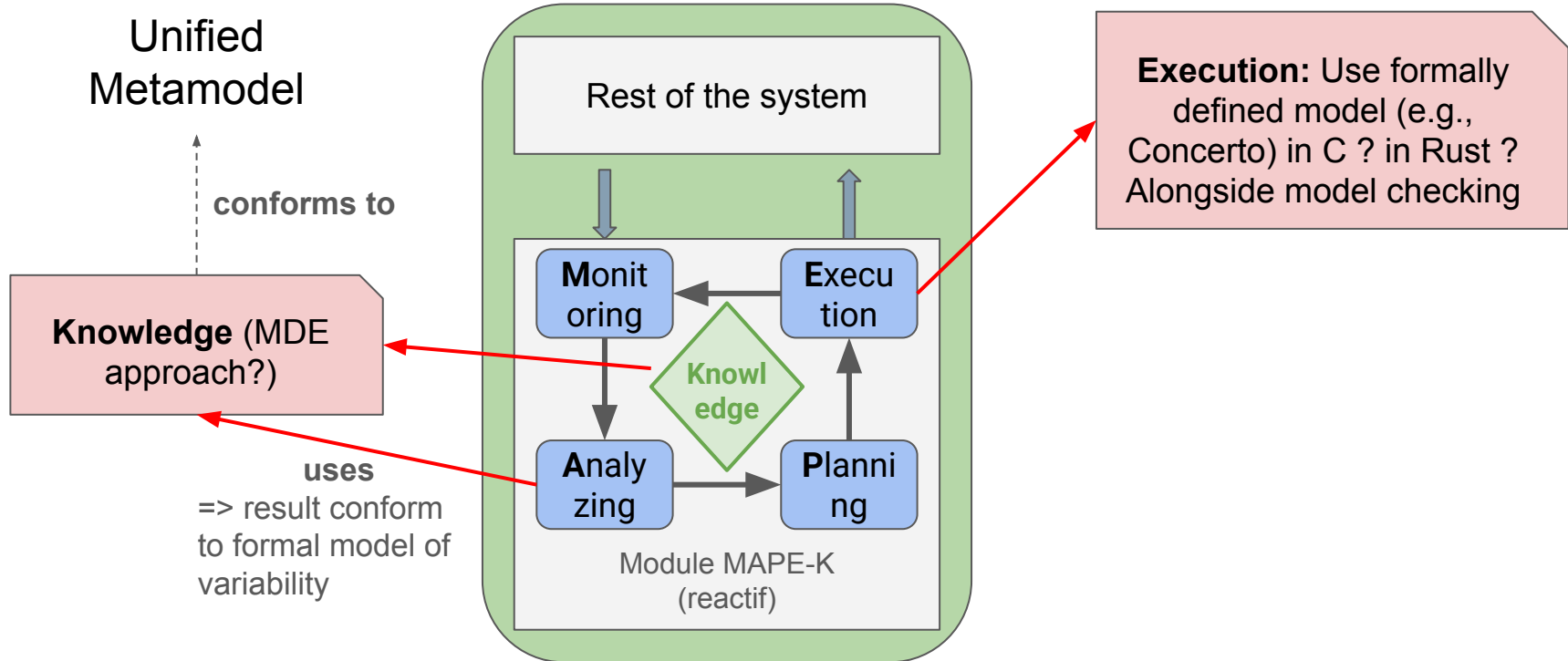
- **Garantir** la sûreté des reconfigurations
- **Automatiser** et **certifier** l'exécution de la reconfiguration

Contribution envisagée

- Utiliser des **modèles** et **langages** de reconfiguration existant (e.g., Concerto) pour **vérifier** les propriétés critiques des processus (e.g., absence deadlock)
- **Formaliser** ce processus et **extraire du code certifié** ?
- **Intégrer** ce modèle reconfiguration sûre dans des systèmes réactifs adaptables (e.g., CPS et IoT energy-aware)



Projet de recherche



Collaboration dans LMV



ANR ForCoala : Langages de configuration et vérification des reconfigurations
([F. Loulergue](#), [O. Proust](#))



APR-IA AcceptAlgo : Acceptabilité des algorithmes
([W. Bousdira-Semmar](#), [A. Ed-Dbali](#), [J. Ischard](#), [F. Loulergue](#))



Programmation réactive ([J. Ischard](#), [F. Dabrowski](#), [F. Loulergue](#))



(Vérification en crypto - [Yohan Boichut](#))

Contribuer à long terme: Un kernel reconfigurable vérifié



Vérification de modules Rust (thèse de [F. Groult](#))

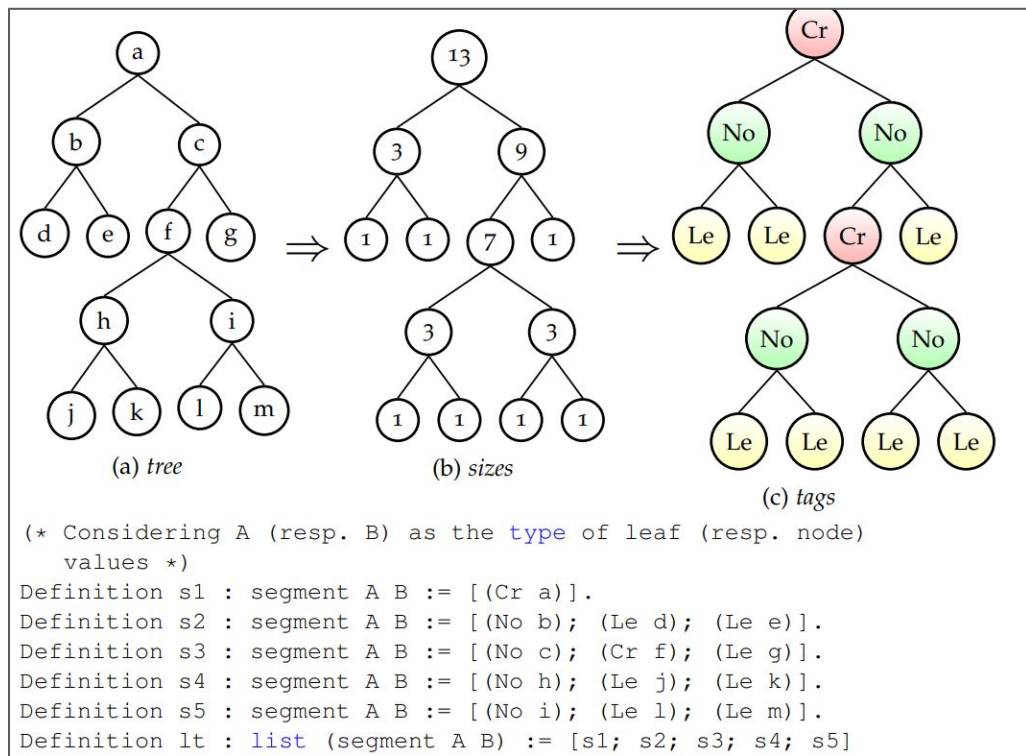


Vérification de modules pour Contiki avec Frama-C ([F. Loulergue](#),
collaboration avec le [CEA](#))

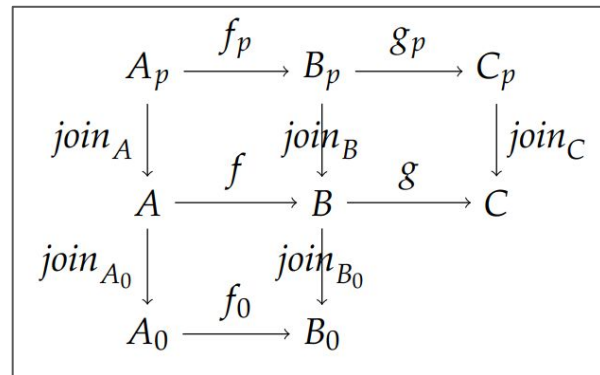
This slide intentionally left blank

Backup

- [Arbre binaire dans SyDPaCC](#)
- [Transformation de modèles distribuée et sémantique](#)
- [Coq2Spark](#)
- [Ballet](#)
- [Concerto et modélisation de cycle de vie pour SAT](#)
- [Explicabilité UNSAT](#)
- [MAPE-K décentralisé](#)
- [SONAR](#)



Linéarisation d'arbres binaires (en Coq)



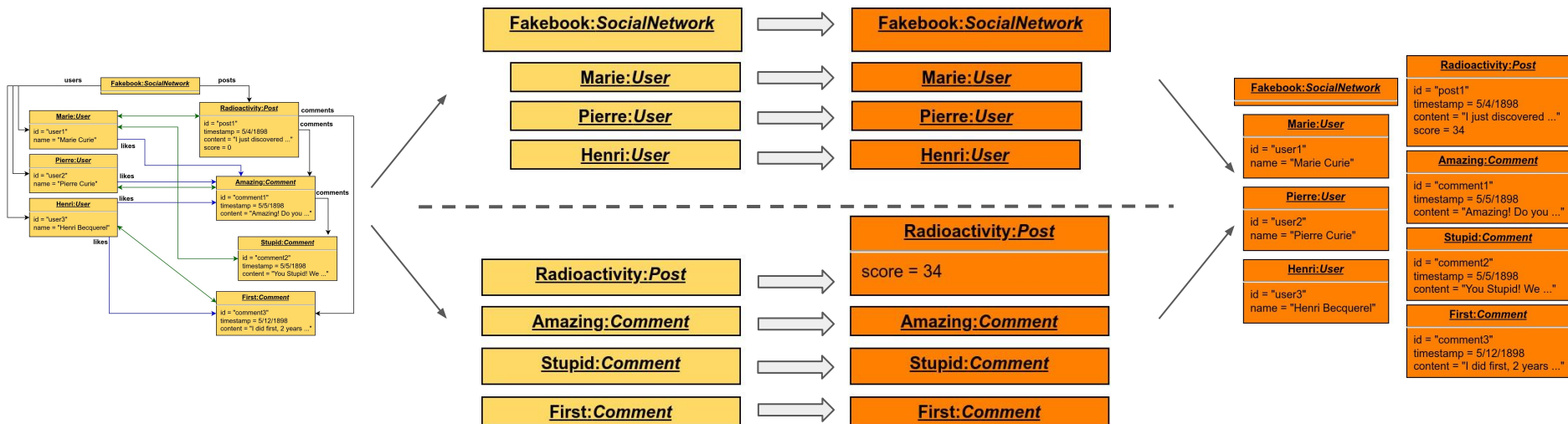
Equivalence horizontale et verticale dans SyDPaCC

- Travaux publiés**

SAC'18

Avec Frédéric LOULERGUE

Transformation de modèle distribuée



Data-distribriuted strategy: (Map-Reduce phase)

- Input **elements** are **distributed**
- Input **model** is **broadcasted**

As output:

- Instantiated **output model elements**
- **Trace-links** (mapping input-output)

Parallelizable CoqTL

Increase parallelization

1. Two distinct phases : **instantiate** and **apply**
 - Defined as map-reduce phases
2. Iterate on rules instead of source patterns
 - Avoid unnecessary computations
3. Iterate on trace for apply instead of source patterns
 - Reuse intermediate results while everything is redefined in CoqTL

	spec (loc)	cert (loc)	effort (man-days)
1.	69	484	10
2.	42	487	7
3.	69	520	4

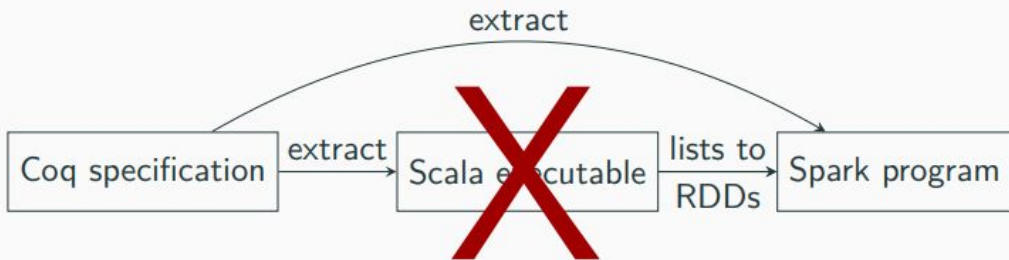


• Travaux publiés

SLE'21

Avec Gerson SUNYE, Massimo TISI et H  l  ne COULLON

Towards Verified Scalable Parallel Computing with Coq and Spark



Extract Coq code into Spark program

- Formalize Spark's distributed structure (i.e., RDD) in Coq
- Formalize computation on RDDs
- Prove the equivalence between function on lists and on RDDs

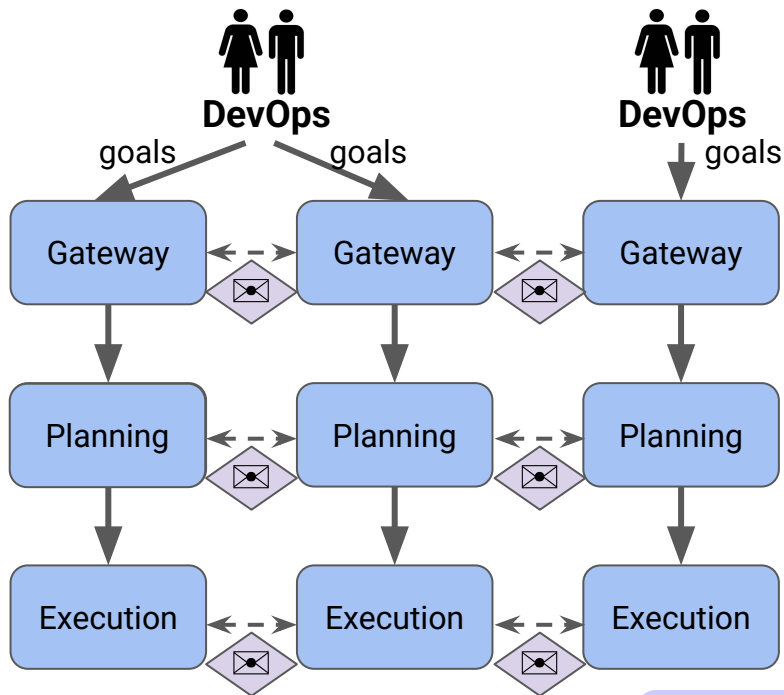
- **Travaux publiés**

FTfJP'23
(ECOOP)

Avec Frédéric LOULERGUE



Ballet : Fast Choreography of Cross-DevOps Reconfiguration



- Conception d'un **outil de reconf. BALLET**
 - Language **déclaratif**
 - **Planification** décentralisée avec **analyse de satisfaisabilité**
 - **Exécution** de la reconfiguration
- **Formalisation du moteur** pour faire de la vérification de modèle
- Analyse **SAT** des reconf + explicabilité **UNSAT**

- **Pour en savoir plus**
- **Soumettre à la revue**

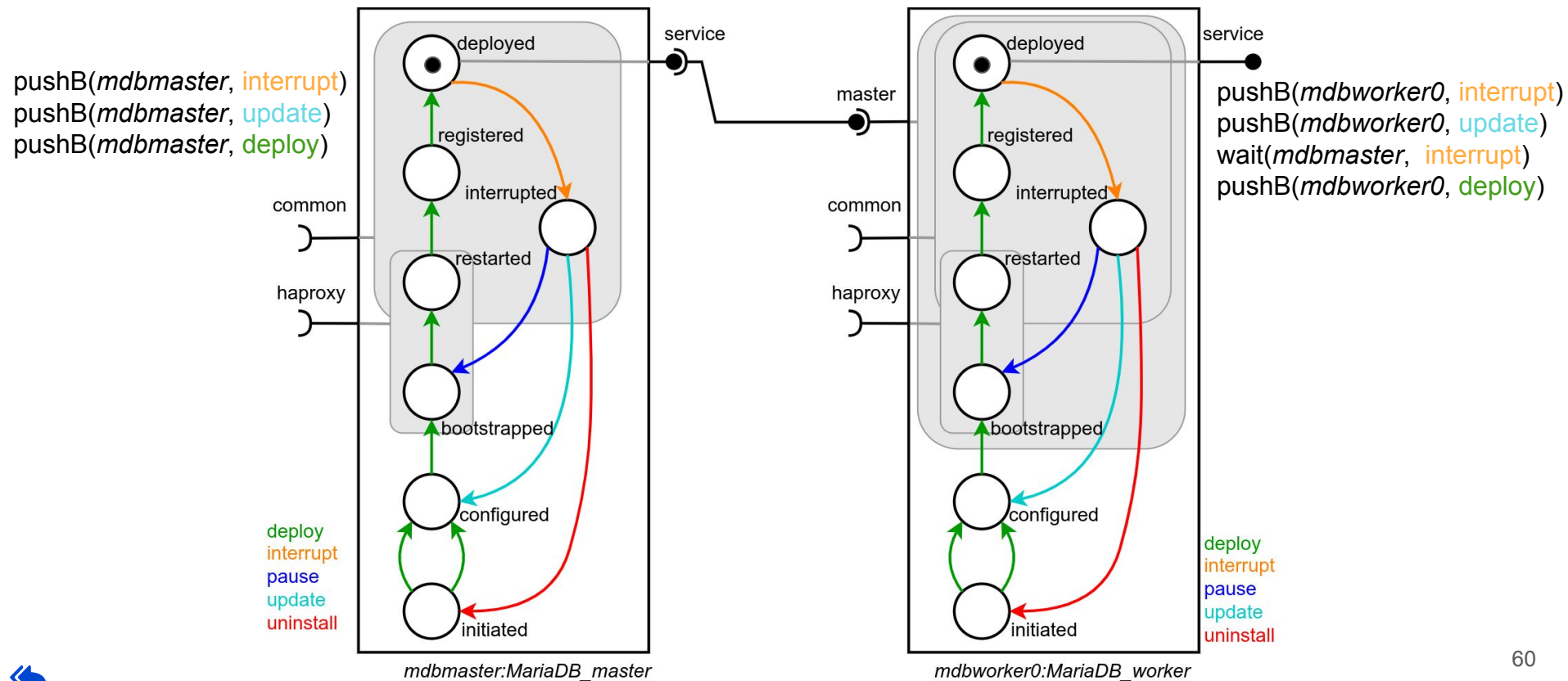
SANER'24

ICE'24
(DisCoTec)

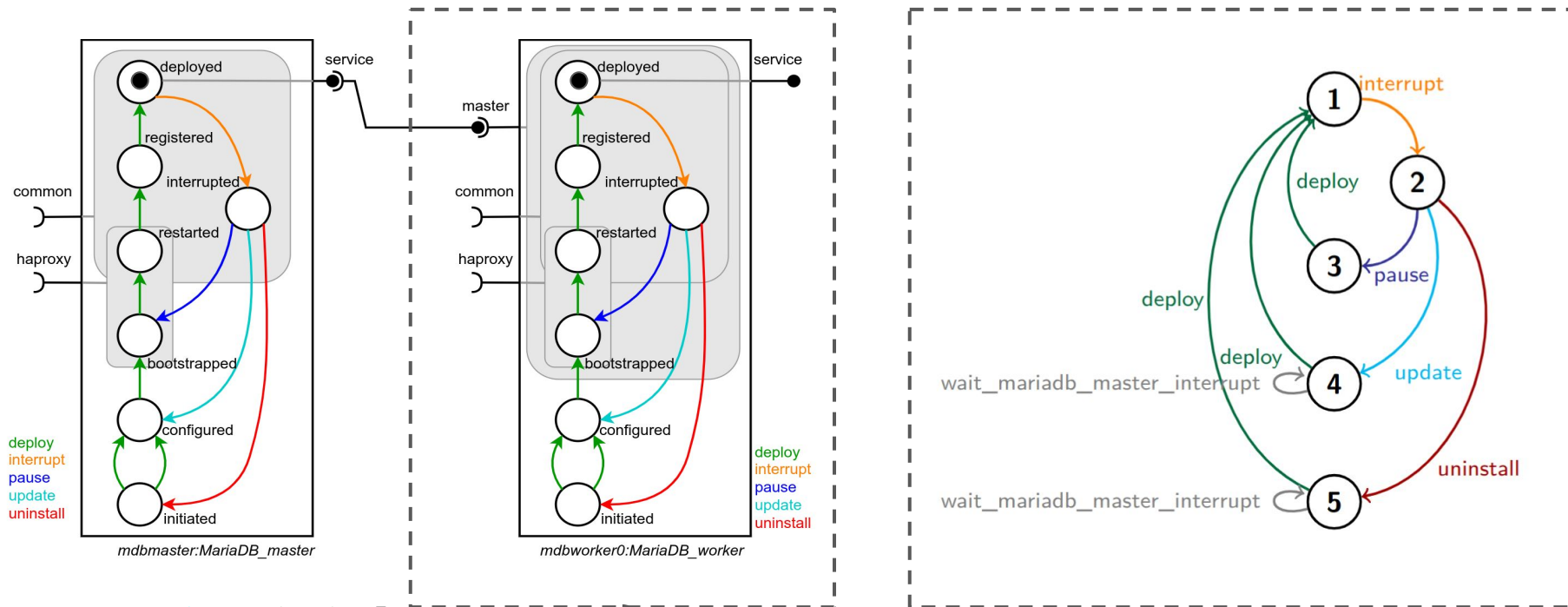
FGCS

JLAMP

Ballet : Utilisation de Concerto-D pour reconfiguration



Ballet : Cycle de vie, modèle SAT et messages-passing

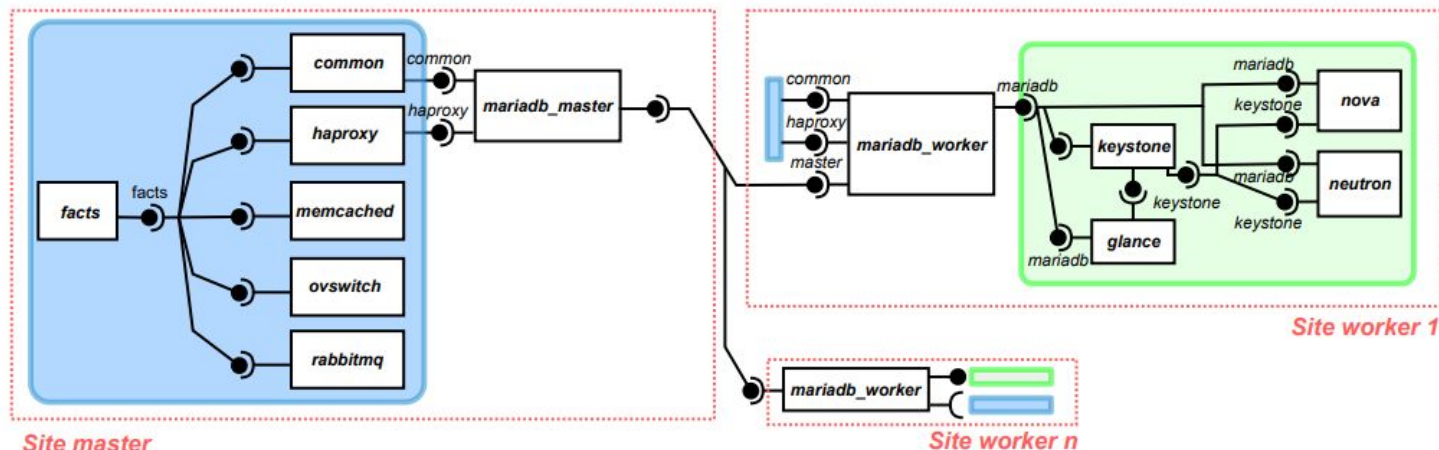


[interrupt; update; deploy]



(from:mdbmaster, to:mdbworker0, what:service, how:disabled, when:interrupt)

Diffusion de contraintes : Multi-site OpenStack



SAT-case goals

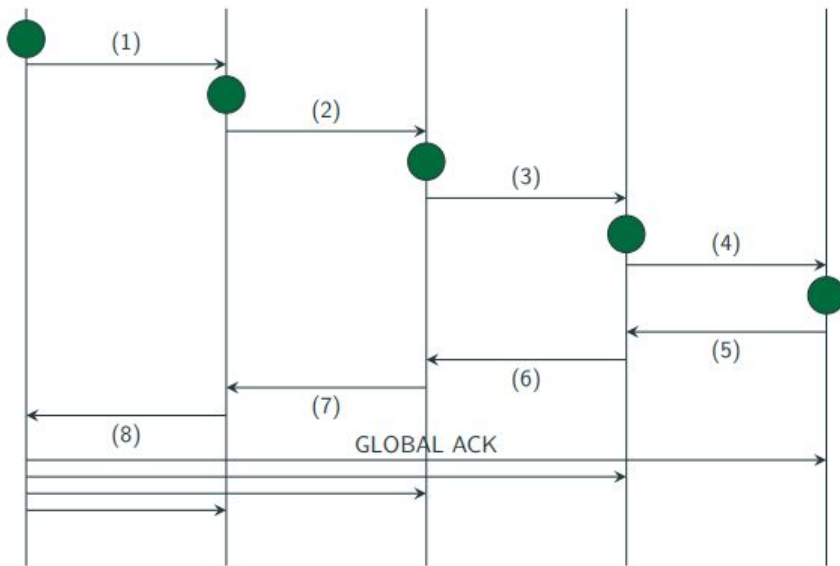
- Update **Site master**'s **common** from v_1 to v_2
- all components end running

UNSAT-case goals

- Update **Site master**'s **common** from v_1 to v_2
- Update **Site worker 1**'s **nova** from v_1 to v_3
- all components end running

Diffusion de contraintes : reconf. SAT

cmnmaster mdbmaster mdbworker1 ksworker1 novaworker1



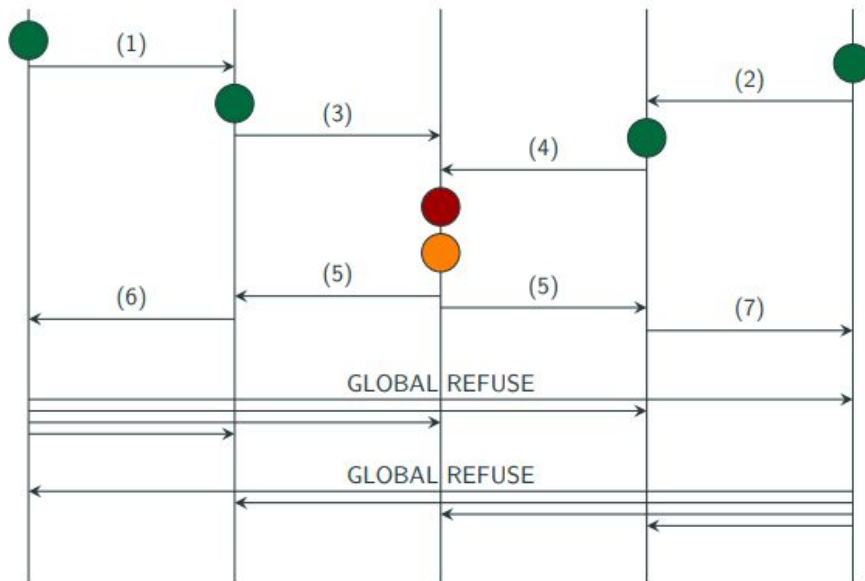
● local solve (SAT)

message := (source, port, status, final)

- (1) (cmnmaster, service(v2), enabled, True)
- (2) (mdbmaster, service(v2), enabled, True)
- (3) (mdbworker1, service(v2), enabled, True)
- (4) (ksworker1, service(v2), enabled, True)
- (5) ACK (4)
- (6) ACK (3)
- (7) ACK (2)
- (8) ACK (1)

Diffusion de contraintes : reconf. UNSAT

cmnmaster mdbmaster mdbworker1 ksworker1 novaworker1



● local solve (SAT) ● QuickXplain

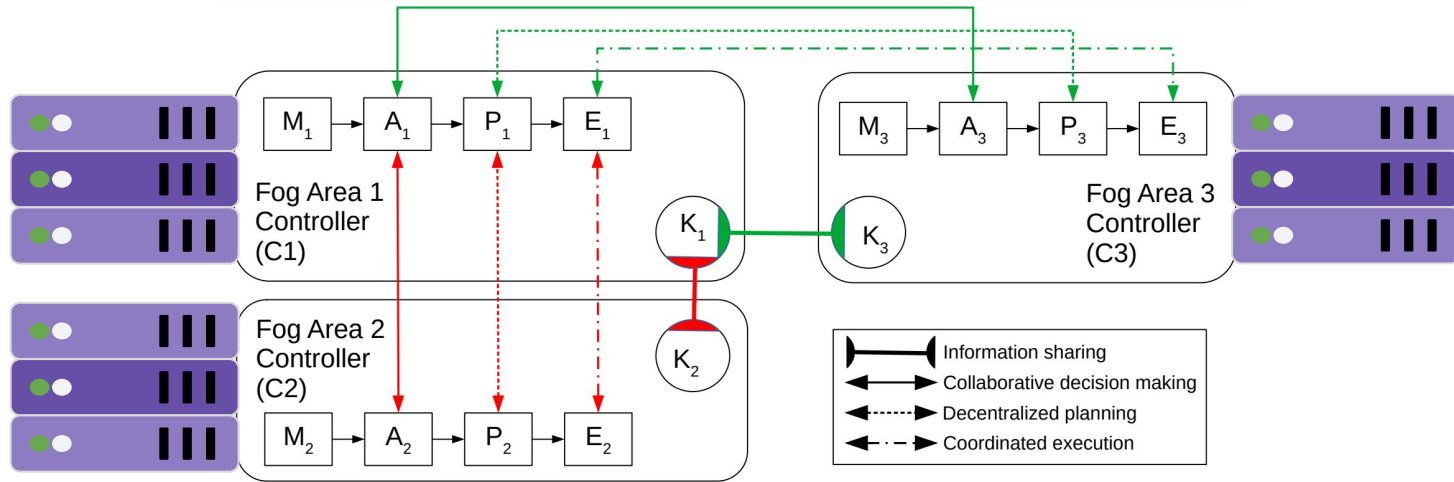
● local solve (UNSAT)

message := (source, port, status, final)

- (1) (cmnmaster, service(v2), enabled, True)
- (2) (novaworker1, service(v3), enabled, True)
- (3) (mdbmaster, service(v2), enabled, True)
- (4) (ksworker1, service(v3), enabled, True)
- (5) REFUSE caused by (3) and (4)
- (6) REFUSE caused by (3), (4) and (1)
- (7) REFUSE caused by (3), (4) and (2)

SeMaFoR : Self Management of Fog Resources

- MAPE-K décentralisé



- Intégrer P et E depuis Ballet

Avec Matthieu RAKOTOJAONA RAINIMANGAVELO, Hélène COULLON, Thomas LEDOUX, Hugo BRUNELIERE, Charles PRUD'HOMME, et Jonathan LEJEUNE

SONAR : Sound Observation Network with Automatic Reconfiguration

