

La Réflexivité dans les architectures multi-niveaux : application aux systèmes tolérant les fautes

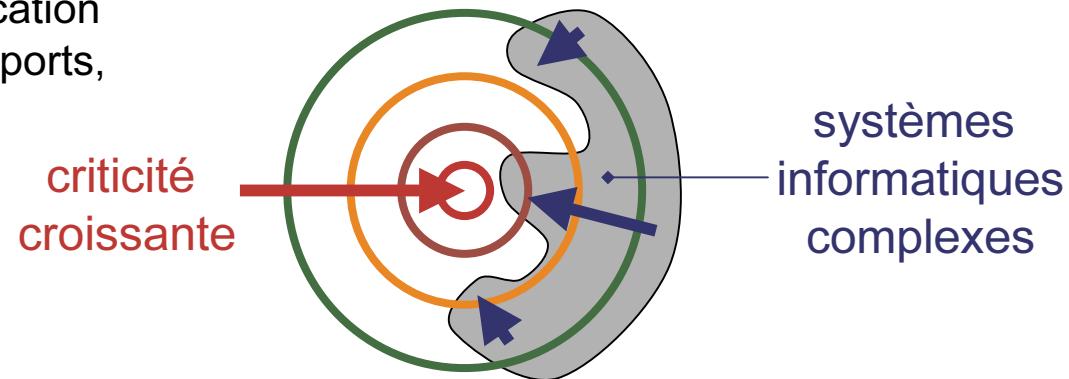
François Taïani



Le contexte

- Des systèmes informatiques de **plus en plus complexes** sont utilisés pour des applications de **plus en plus critiques**.

domaines d'application
(domotique, transports,
médecine, ...)



- Complexité
 - Dans **l'espace** : - plusieurs centaines de **personnes**
- très nombreux **composants** / **fournisseurs**
 - Dans **le temps** : très longs cycles de vie : **10, 20 ans**
- Criticité : défaillance \Rightarrow catastrophe (☹☹ / €€)
 - La **sûreté de fonctionnement** est incontournable.

La Problématique

- La sûreté de fonctionnement est une propriété **globale**.
 - Les « vulnérabilités » portent sur l'ensemble du système.
 - C'est le **comportement global** du système qui compte.
 - Tolérance aux fautes : besoins d'**observation** + **action**
 - capture d'état, maîtrise du non déterminisme, *etc.*
- La maîtrise de la complexité privilégie la **localité**.
 - **Modularité** des composants, **transparence** des mécanismes
 - Mais aussi « **impénétrabilité** » de la réalisation ☹
 - Organisation en « **couches** » **hétérogènes**

La Problématique

- La sûreté de fonctionnement est une propriété **globale**.
 - Les « vulnérabilités » portent sur l'ensemble du système.
 - C'est le **comportement global** du système qui compte.
 - Tolérance aux fautes : besoins d'**observation** + **action**
 - capture d'état, maîtrise du non déterminisme, *etc.*
- La maîtrise de la complexité privilégie la **localité**.
 - **Modularité** des composants, **transparence** des mécanismes
 - Mais aussi « **impénétrabilité** » de la réalisation ☹
 - Organisation en « **couches** » **hétérogènes**

⇒ **Il y a un conflit entre la sûreté de fonctionnement (globalité) et la maîtrise de la complexité (localité)**

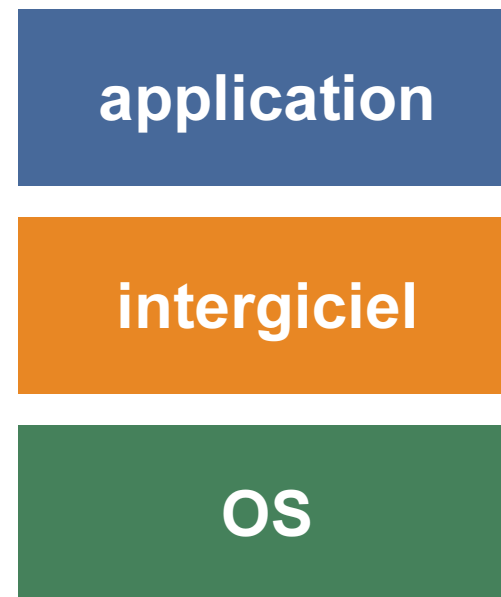
La Problématique

- La sûreté de fonctionnement est une propriété **globale**.
 - Les « vulnérabilités » portent sur l'ensemble du système.
 - C'est le **comportement global** du système qui compte.
 - **Tolérance aux fautes** : besoins d'**observation** + **action**
 - capture d'état, maîtrise du non déterminisme, *etc.*
- La maîtrise de la complexité privilégie la **localité**.
 - **Modularité** des composants, **transparence** des mécanismes
 - Mais aussi « **impénétrabilité** » de la réalisation ☹
 - Organisation en « **couches** » **hétérogènes**

⇒ **Il y a un conflit entre la sûreté de fonctionnement (globalité) et la maîtrise de la complexité (localité)**

La pratique actuelle

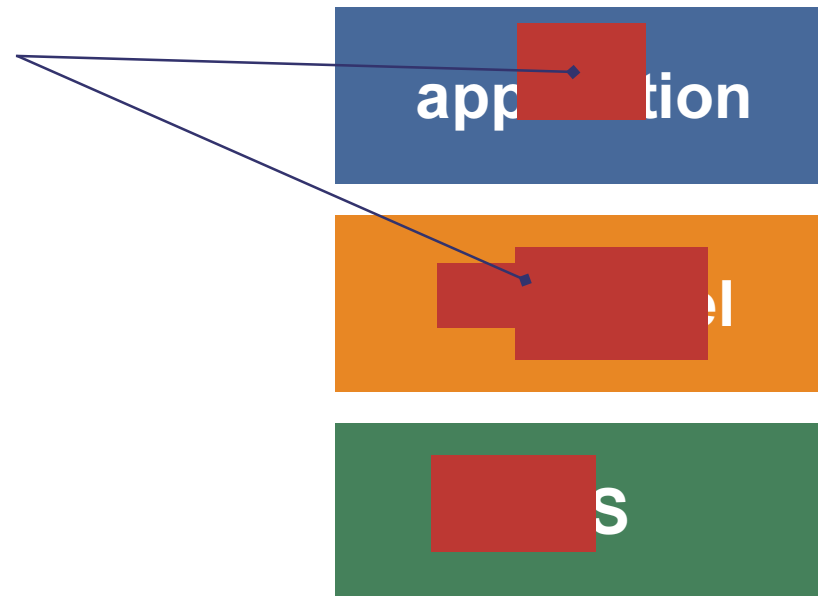
- La plupart des composants du marché ne prennent pas en compte la tolérance aux fautes.



La pratique actuelle

- La plupart des composants du marché ne prennent pas en compte la tolérance aux fautes.
- Des adaptations **sur mesure** sont nécessaires.

« rustines » de tolérance aux fautes

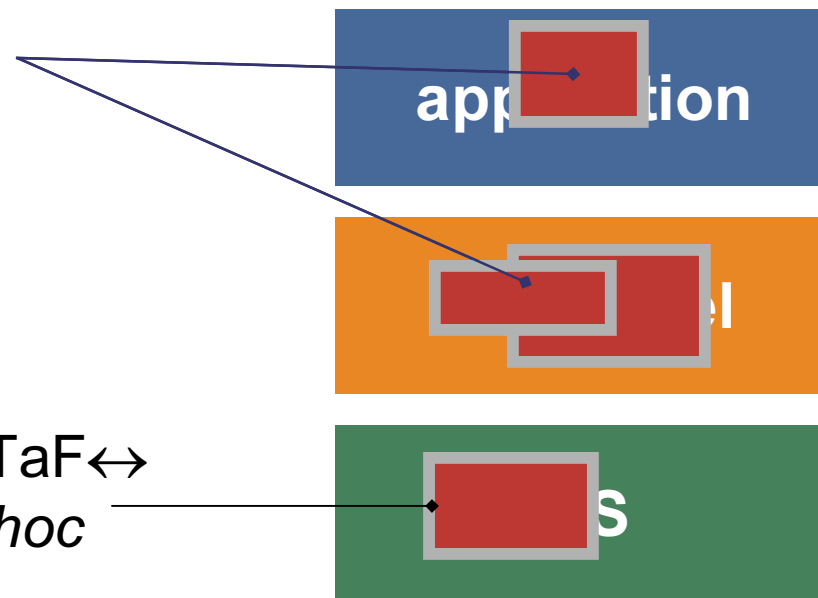


La pratique actuelle

- La plupart des composants du marché ne prennent pas en compte la tolérance aux fautes.
- Des adaptations **sur mesure** sont nécessaires.

« rustines » de tolérance aux fautes

connexion code TaF ↔
code original *ad-hoc*



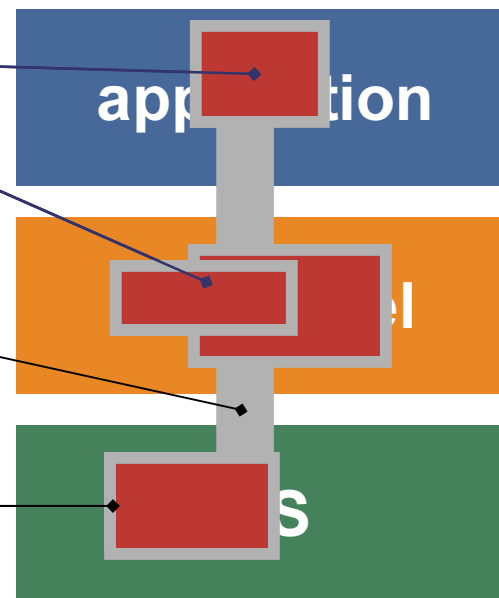
La pratique actuelle

- La plupart des composants du marché ne prennent pas en compte la tolérance aux fautes.
- Des adaptations **sur mesure** sont nécessaires.

« rustines » de tolérance aux fautes

coordination
inter-niveaux
ad hoc

connexion code TaF ↔
code original *ad-hoc*



Les adaptations sur mesure

- Elles ne sont **pas réutilisables**.
 - Elles sont spécifiques aux composants choisis.
 - Le système obtenu est très difficilement maintenable. (€!)
 - La tolérance aux fautes ne peut pas évoluer.
- Elles **dépendent** du **fournisseur**.
 - Elles utilisent souvent des **spécificités** non documentées.
 - Exemple : l'OS VxWorks dans la sonde Pathfinder
 - L'intégrateur doit **collaborer étroitement** avec le fournisseur. (€!)
 - La Nasa avec WindRiver, la SNCF avec ILOG (station d'aiguillage)

Les adaptations sur mesure

- Elles ne sont **pas réutilisables**.
 - Elles sont spécifiques aux composants choisis.
 - Le système obtenu est très difficilement maintenable. (€!)
 - La tolérance aux fautes ne peut pas évoluer.
- Elles **dépendent** du **fournisseur**.
 - Elles utilisent souvent des **spécificités** non documentées.
 - Exemple : l'OS VxWorks dans la sonde Pathfinder
 - L'intégrateur doit **collaborer étroitement** avec le fournisseur. (€!)
 - La Nasa avec WindRiver, la SNCF avec ILOG (station d'aiguillage)

⇒ Architectures « spaghetti » pour un coût très important !

Séparation & transversalité

- Nous recherchons un **paradigme architectural** pour aborder **séparément** la tolérance aux fautes du reste d'un système complexe.

Séparation & transversalité

- Nous recherchons un **paradigme architectural** pour aborder **séparément** la tolérance aux fautes du reste d'un système complexe.
- Or la tolérance aux fautes est un aspect **transversal**.
La plupart des mécanismes sont indépendants du domaine d'application.

Séparation & transversalité

- Nous recherchons un **paradigme architectural** pour aborder **séparément** la tolérance aux fautes du reste d'un système complexe.
- Or la tolérance aux fautes est un aspect **transversal**.
La plupart des mécanismes sont indépendants du domaine d'application.
- Une approche semble prometteuse : la **réflexivité**.
 - Elle sépare les aspects transversaux du reste du système.
 - Déjà utilisée pour la tolérance aux fautes de **petits** systèmes

Séparation & transversalité

- Nous recherchons un **paradigme architectural** pour aborder **séparément** la tolérance aux fautes du reste d'un système complexe.
- Or la tolérance aux fautes est un aspect **transversal**.
La plupart des mécanismes sont indépendants du domaine d'application.
- Une approche semble prometteuse : la **réflexivité**.
 - Elle sépare les aspects transversaux du reste du système.
 - Déjà utilisée pour la tolérance aux fautes de **petits** systèmes

⇒ **La réflexivité permet-elle de mettre en œuvre la tolérance aux fautes dans les systèmes complexes ?**

Plan

Plan

- (A) Qu'est-ce-que la réflexivité ?

Plan

- **(A)** Qu'est-ce-que la réflexivité ?
- **(B)** Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes

Plan

- **(A)** Qu'est-ce-que la réflexivité ?
- **(B)** Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes
- **(C)** Point de vue architectural :
La réflexivité dans les systèmes complexes

Plan

- **(A)** Qu'est-ce-que la réflexivité ?
- **(B)** Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes
- **(C)** Point de vue architectural :
La réflexivité dans les systèmes complexes
- **(D)** Application à un exemple concret :
Réplication multi-traitement d'une plate-forme CORBA / Linux

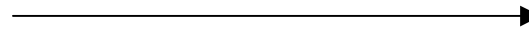
Plan

■ (A) Qu'est-ce-que la réflexivité ?

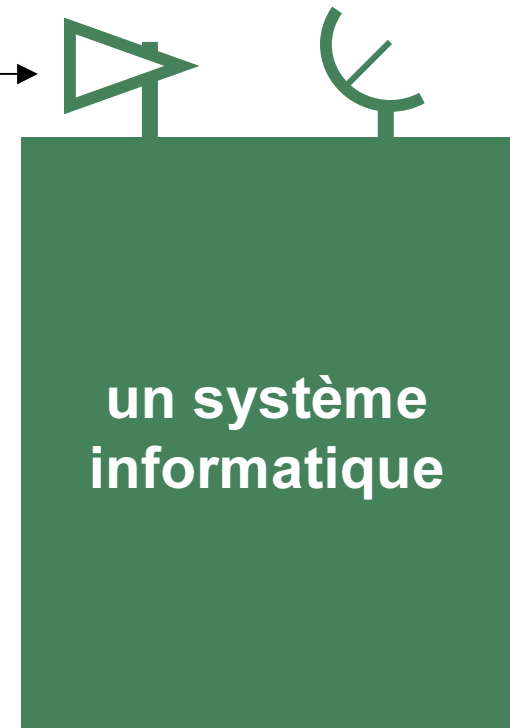
- (B) Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes
- (C) Point de vue architectural :
La réflexivité dans les systèmes complexes
- (D) Application à un exemple concret :
Réplication multi-traitement d'une plate-forme CORBA / Linux

Qu'est-ce que la réflexivité ?

interface fonctionnelle,
visible extérieurement



Monde extérieur

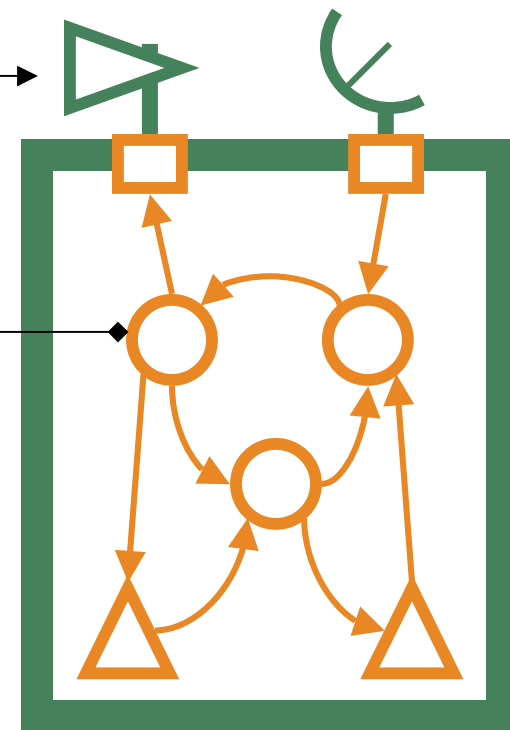


Qu'est-ce que la réflexivité ?

interface fonctionnelle,
visible extérieurement

réalisation interne,
invisible depuis l'extérieur
utilise ○ ; △ ; □ ; →

Monde extérieur



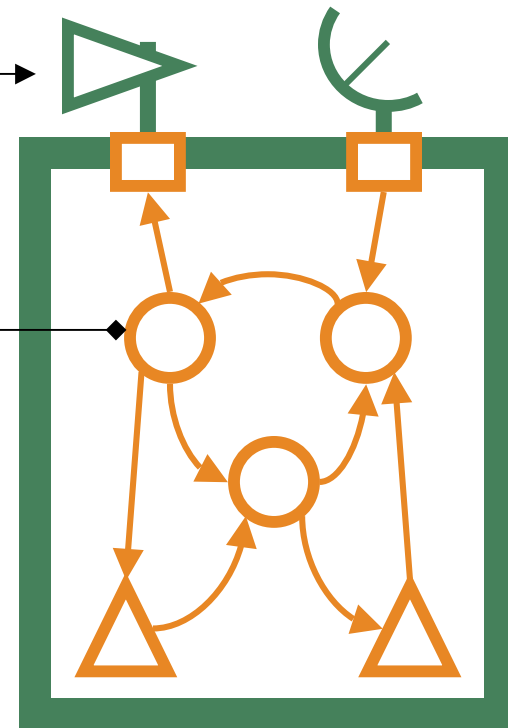
Qu'est-ce que la réflexivité ?

interface fonctionnelle,
visible extérieurement

réalisation interne,
invisible depuis l'extérieur
utilise ○ ; △ ; □ ; →

«modèle de programmation» :
procédures + variables,
objets + méthodes ...

Monde extérieur



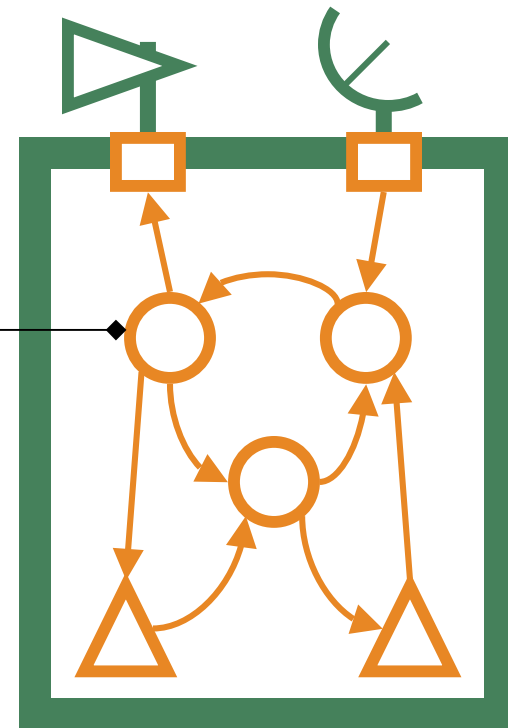
Qu'est-ce que la réflexivité ?

Par défaut : **séparation** stricte entre le **programme** et l'univers sur lequel **il agit**

réalisation interne,
invisible depuis l'extérieur
utilise ○ ; △ ; □ ; →

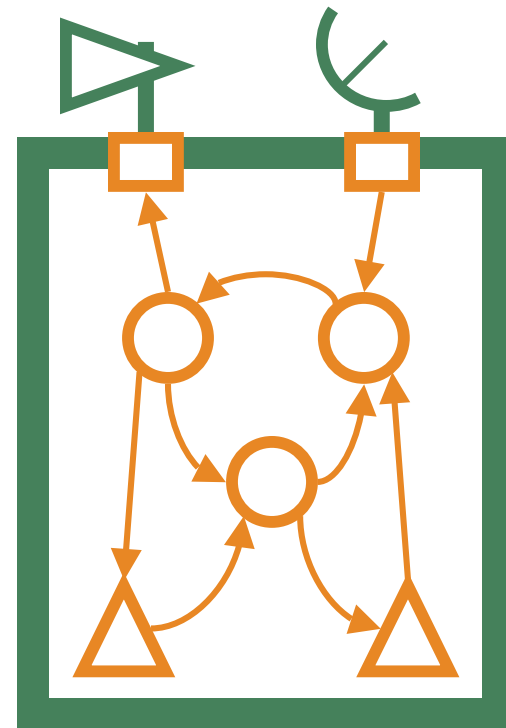
«modèle de programmation» :
procédures + variables,
objets + méthodes ...

Monde extérieur



Qu'est-ce que la réflexivité ?

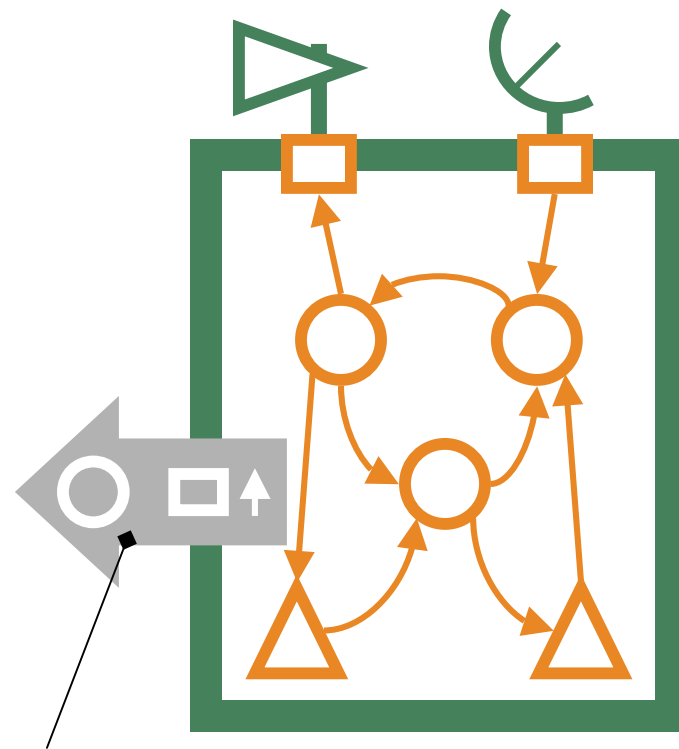
La réflexivité permet de rendre **observable** et **manipulable** l'intérieur d'un système.



Qu'est-ce que la réflexivité ?

La réflexivité permet de rendre **observable** et **manipulable** l'intérieur d'un système.

Comment ? :
en exportant une **représentation** de son fonctionnement



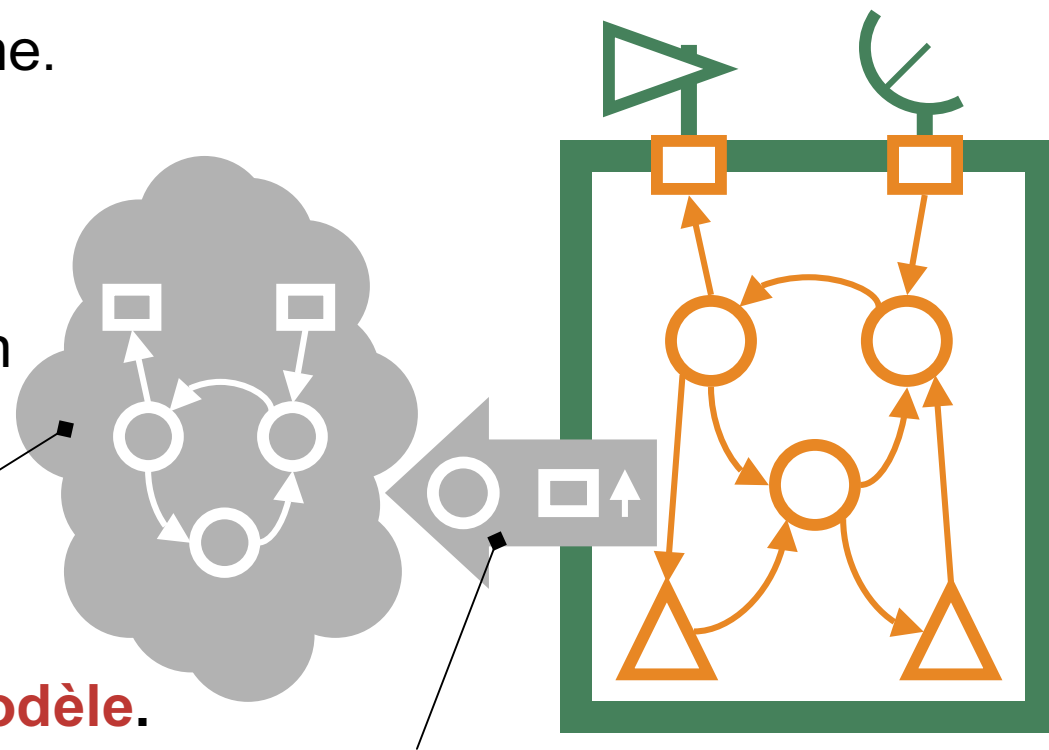
exportation de certains éléments
constitutifs du système

Qu'est-ce que la réflexivité ?

La réflexivité permet de rendre **observable** et **manipulable** l'intérieur d'un système.

Comment ? :
en exportant une **représentation** de son fonctionnement

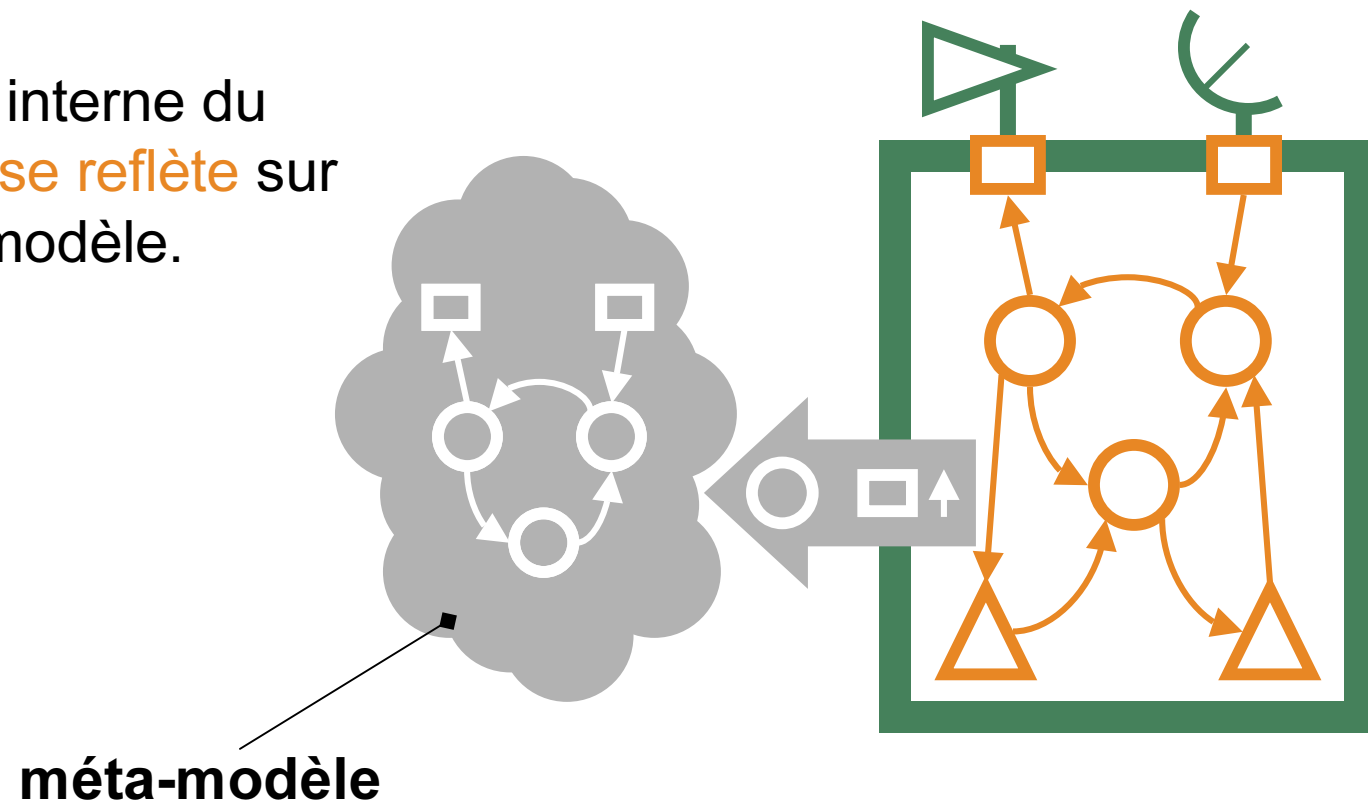
Cette représentation s'appelle un **méta-modèle**.



exportation de certains éléments
constitutifs du système

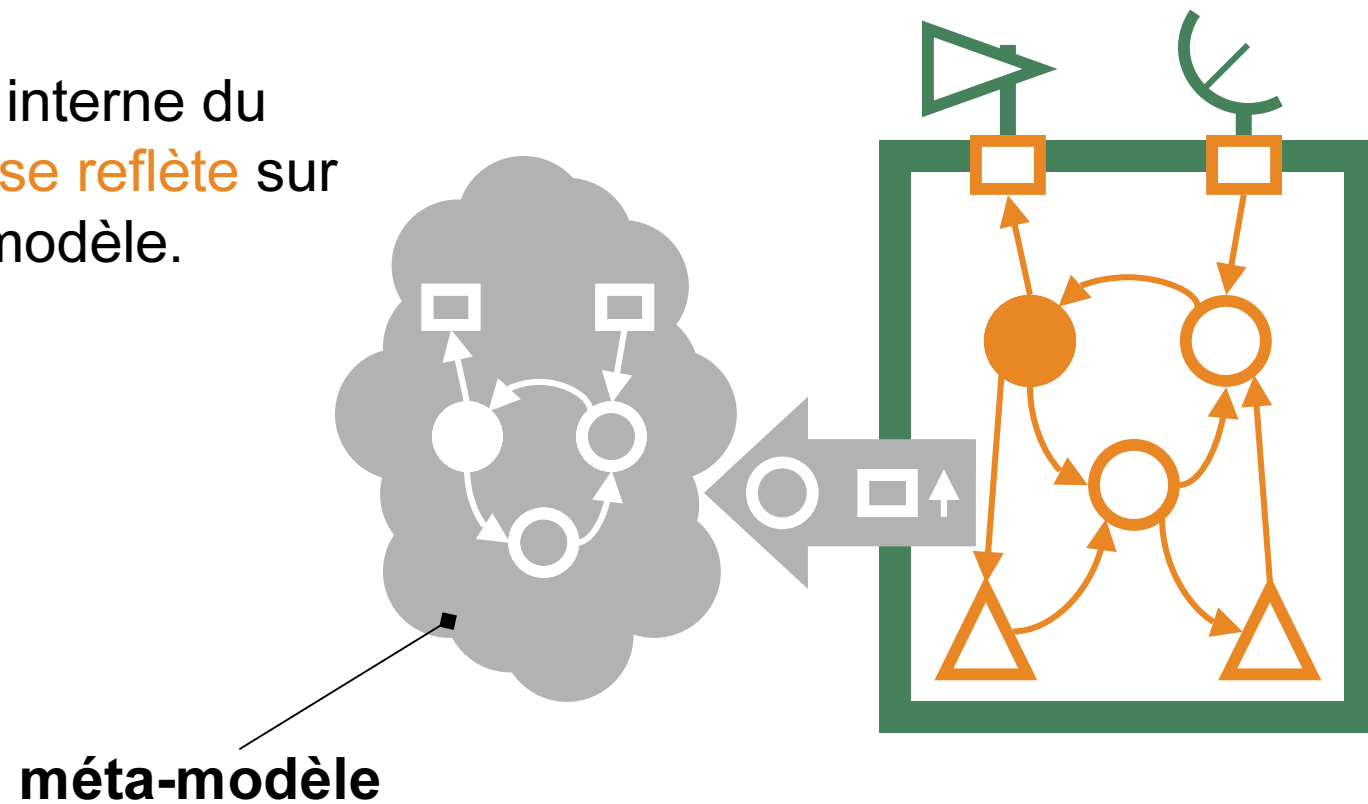
Qu'est-ce que la réflexivité ?

L'activité interne du système **se reflète** sur le méta-modèle.



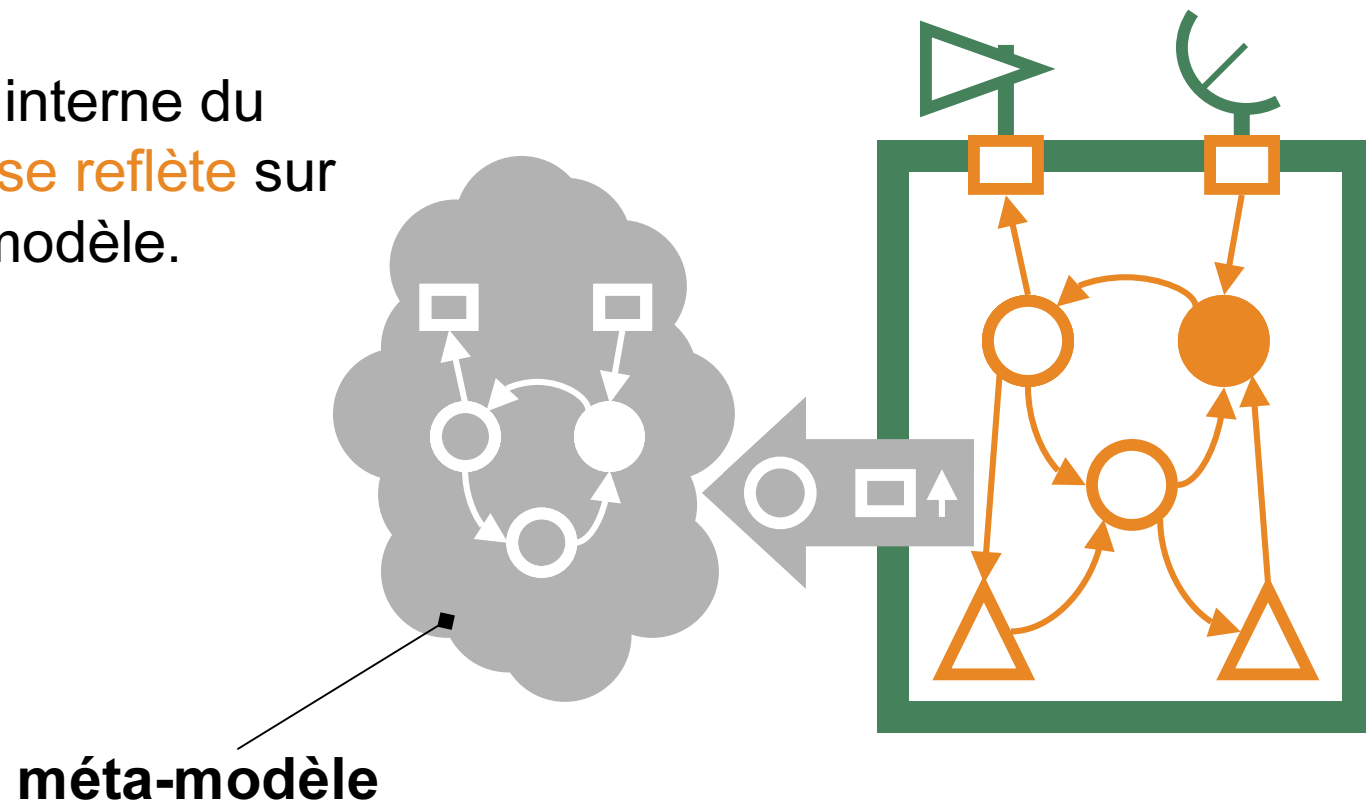
Qu'est-ce que la réflexivité ?

L'activité interne du système **se reflète** sur le méta-modèle.



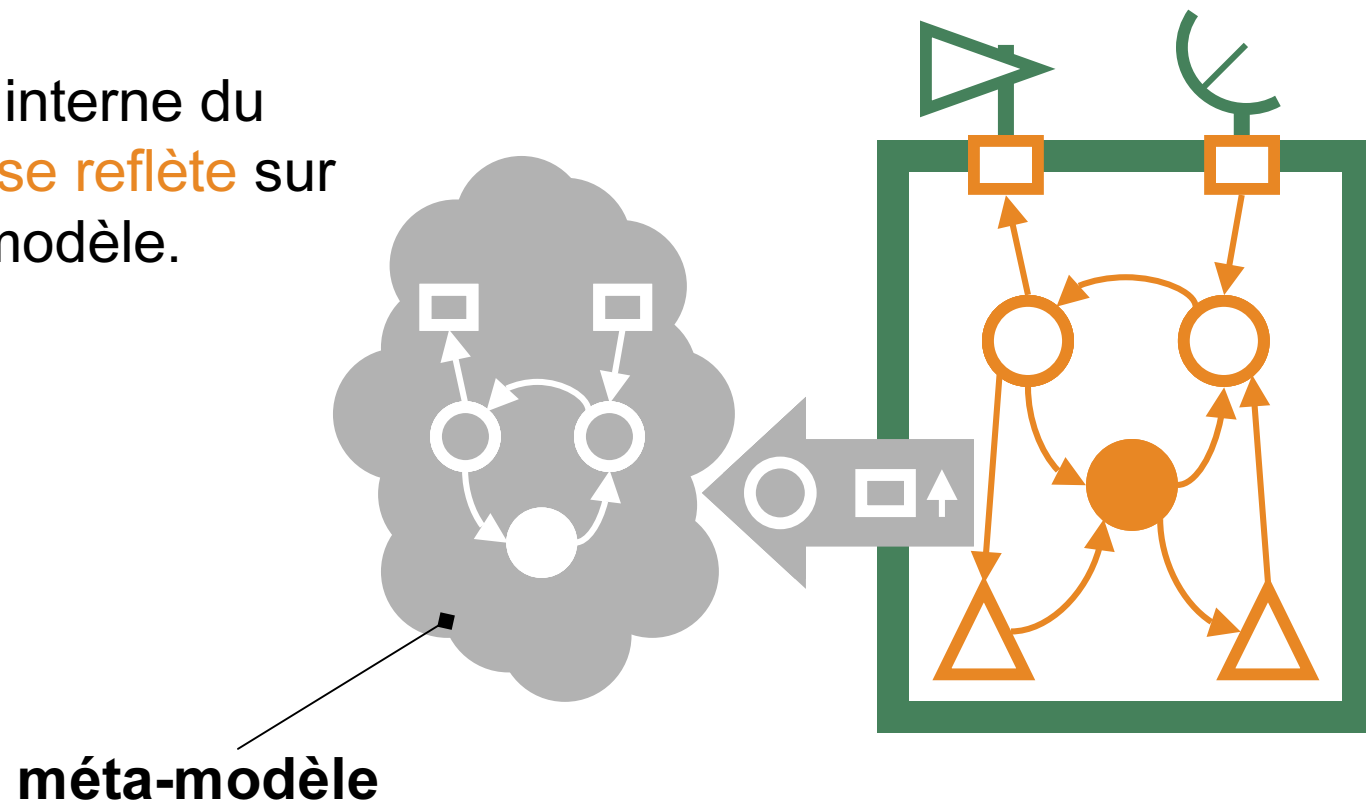
Qu'est-ce que la réflexivité ?

L'activité interne du système **se reflète** sur le méta-modèle.



Qu'est-ce que la réflexivité ?

L'activité interne du système **se reflète** sur le méta-modèle.

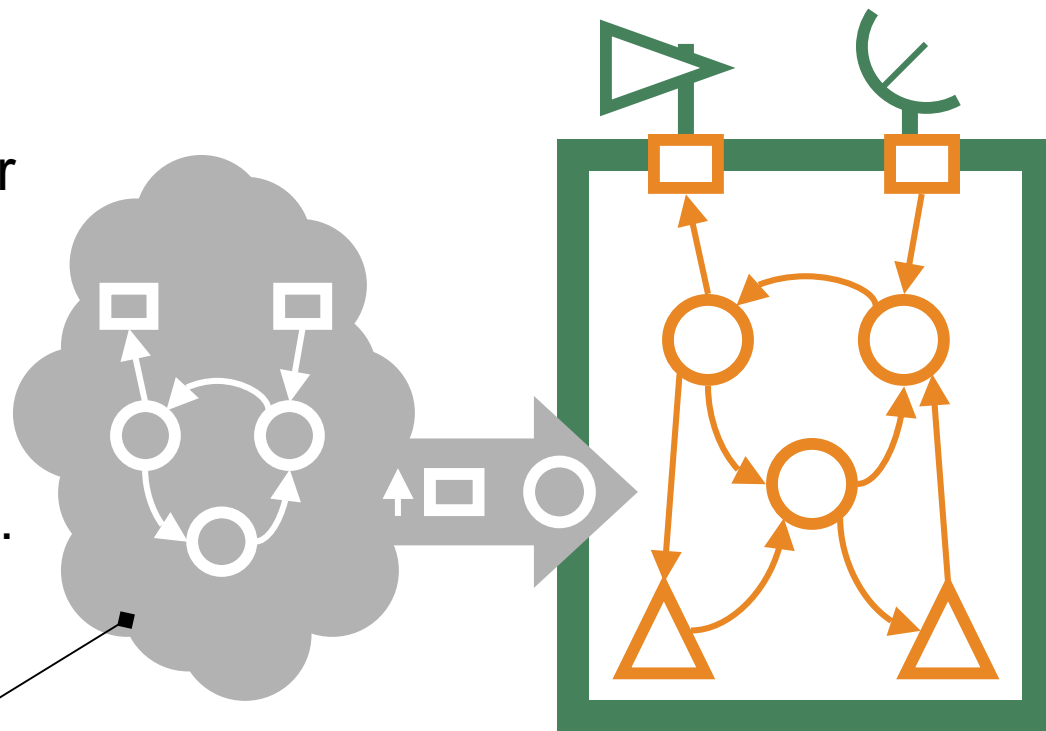


Qu'est-ce que la réflexivité ?

L'activité interne du système se **reflète** sur le méta-modèle.

Toute manipulation du méta-modèle se **reflète** sur le système.

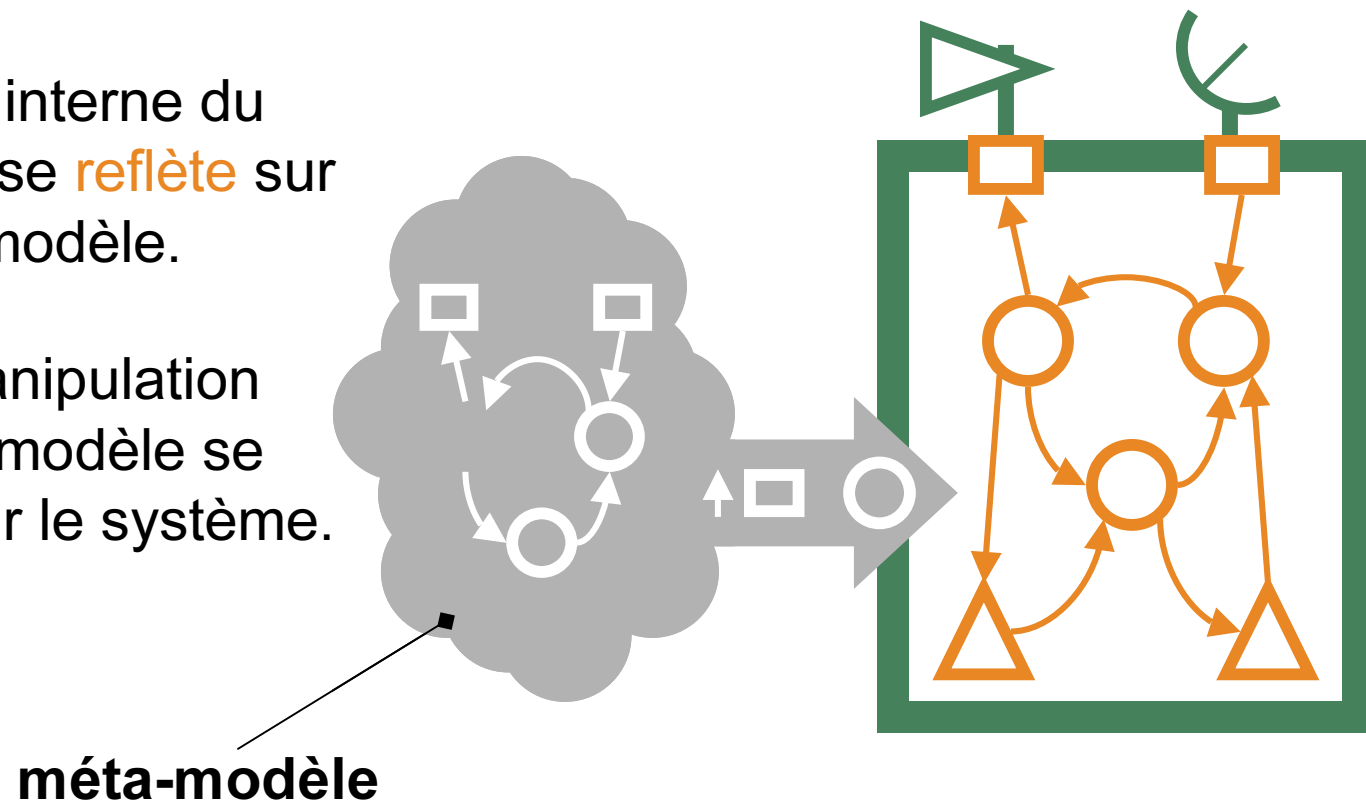
méta-modèle



Qu'est-ce que la réflexivité ?

L'activité interne du système se **reflète** sur le méta-modèle.

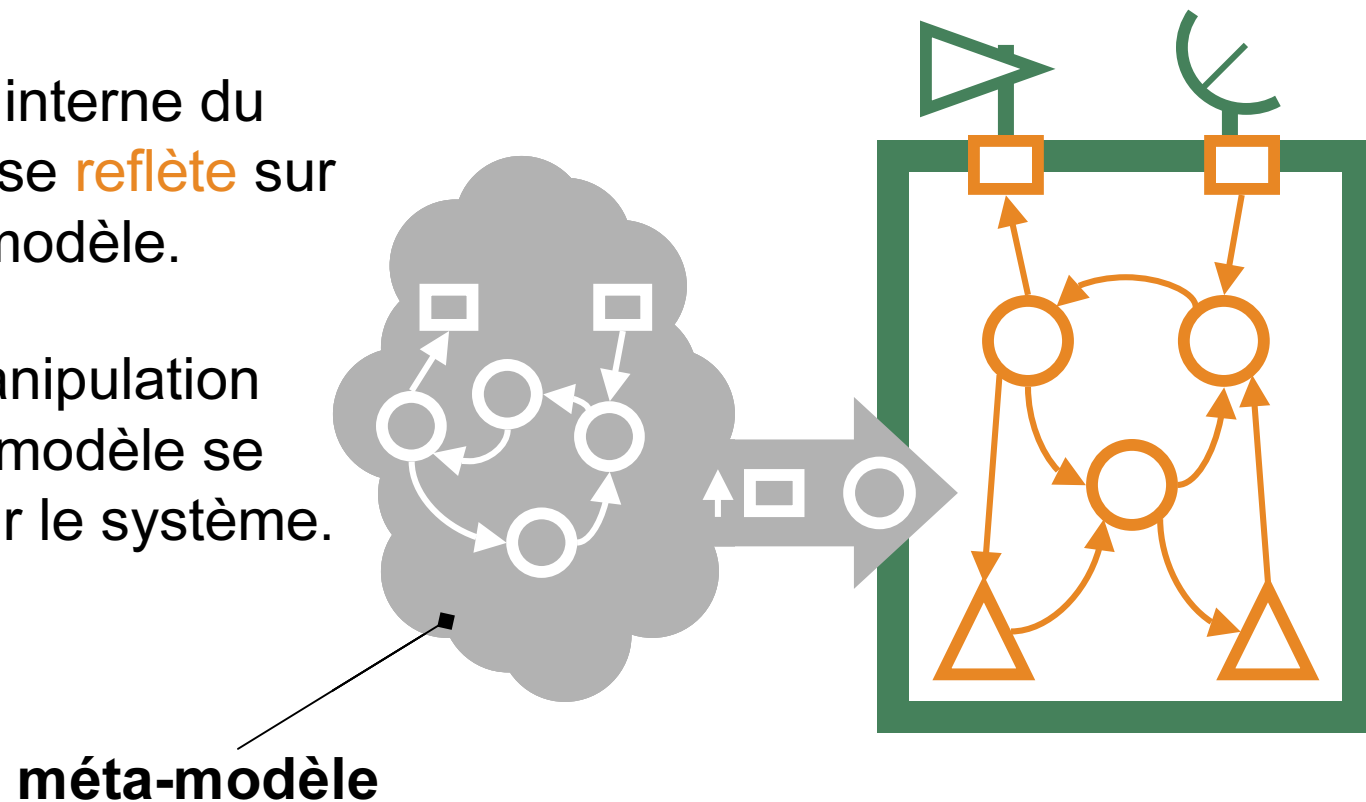
Toute manipulation du méta-modèle se **reflète** sur le système.



Qu'est-ce que la réflexivité ?

L'activité interne du système se **reflète** sur le méta-modèle.

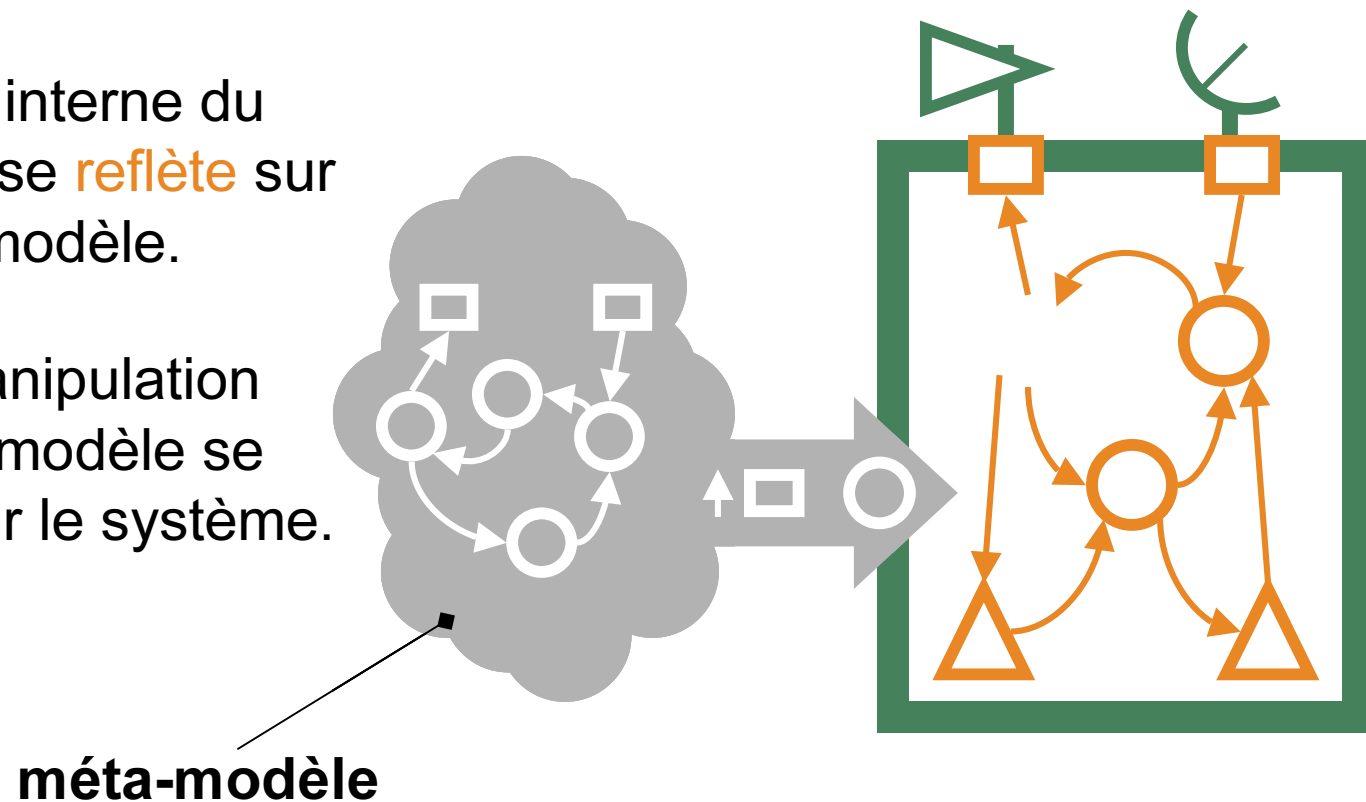
Toute manipulation du méta-modèle se **reflète** sur le système.



Qu'est-ce que la réflexivité ?

L'activité interne du système se **reflète** sur le méta-modèle.

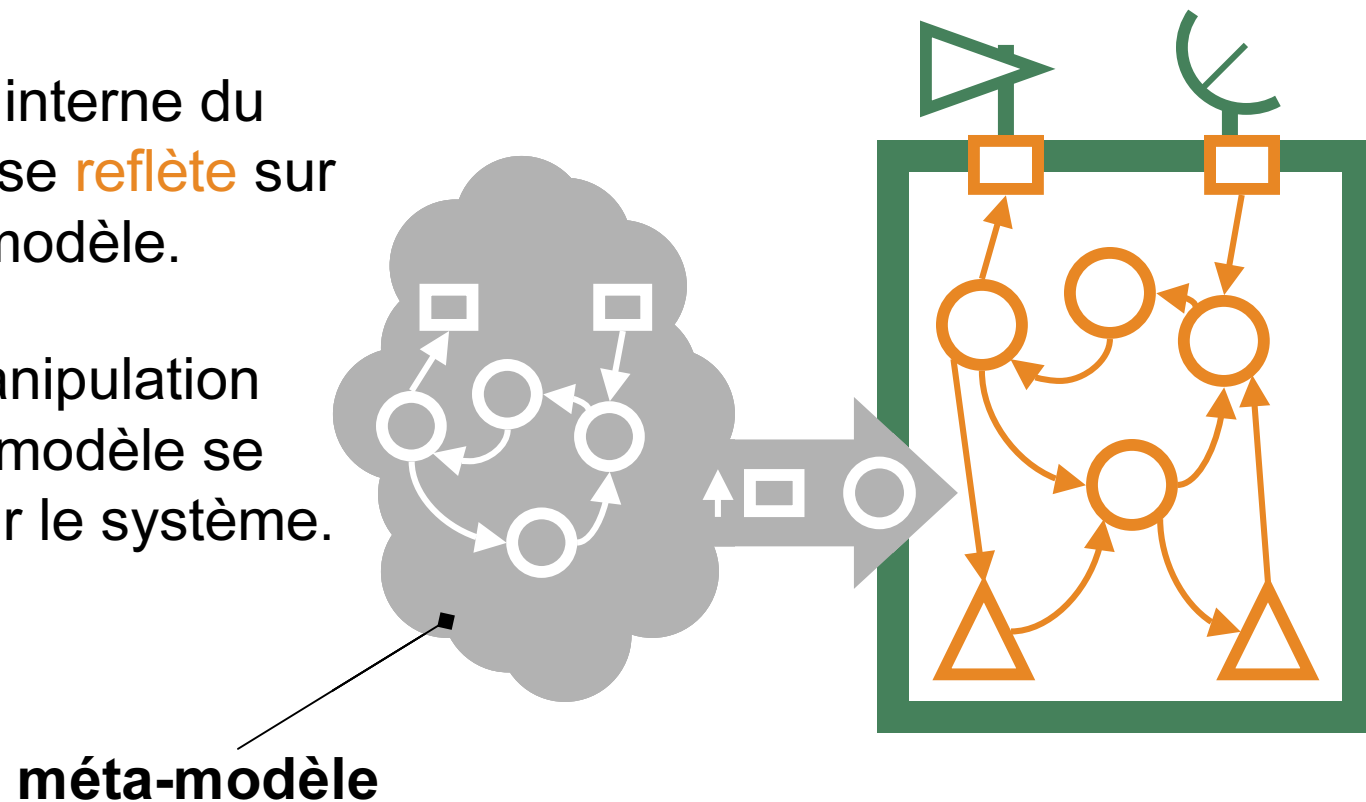
Toute manipulation du méta-modèle se **reflète** sur le système.



Qu'est-ce que la réflexivité ?

L'activité interne du système se **reflète** sur le méta-modèle.

Toute manipulation du méta-modèle se **reflète** sur le système.

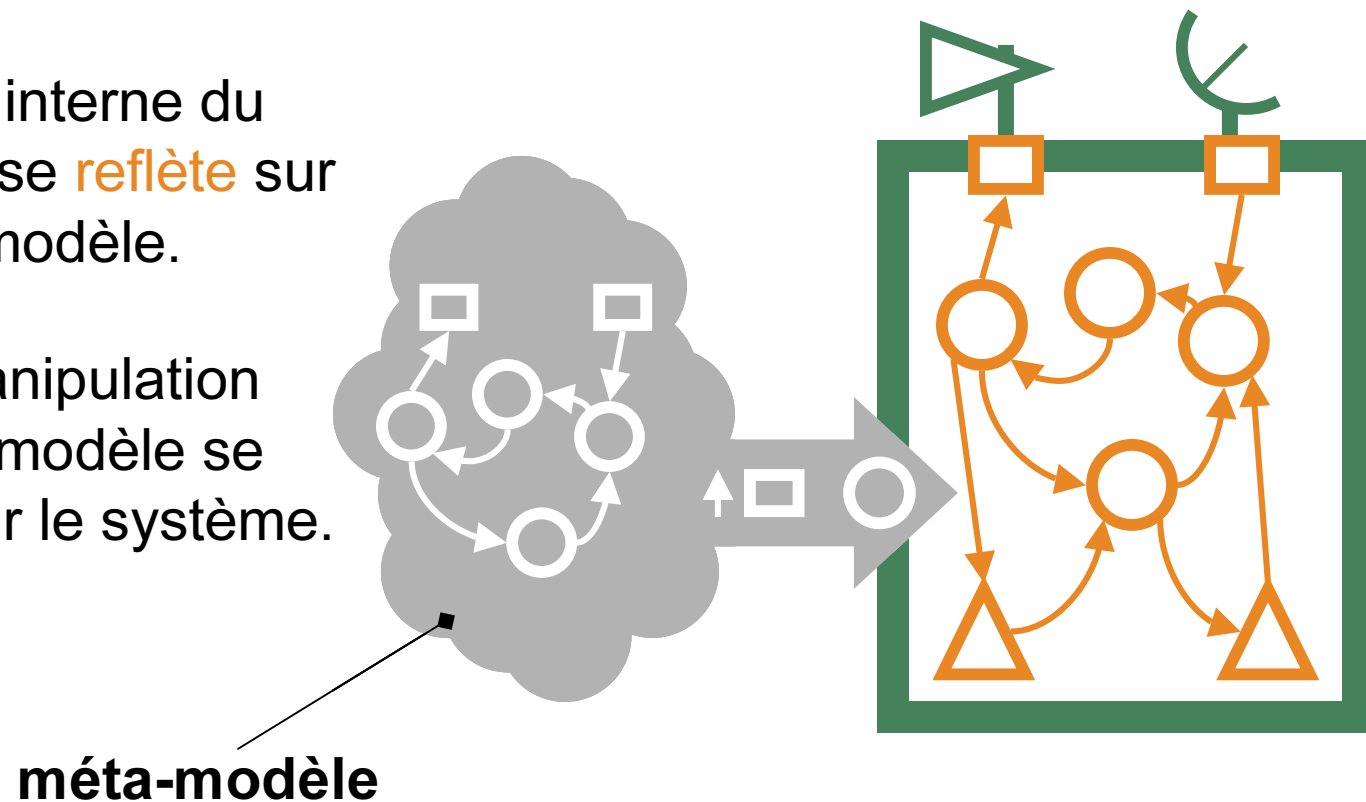


Qu'est-ce que la réflexivité ?

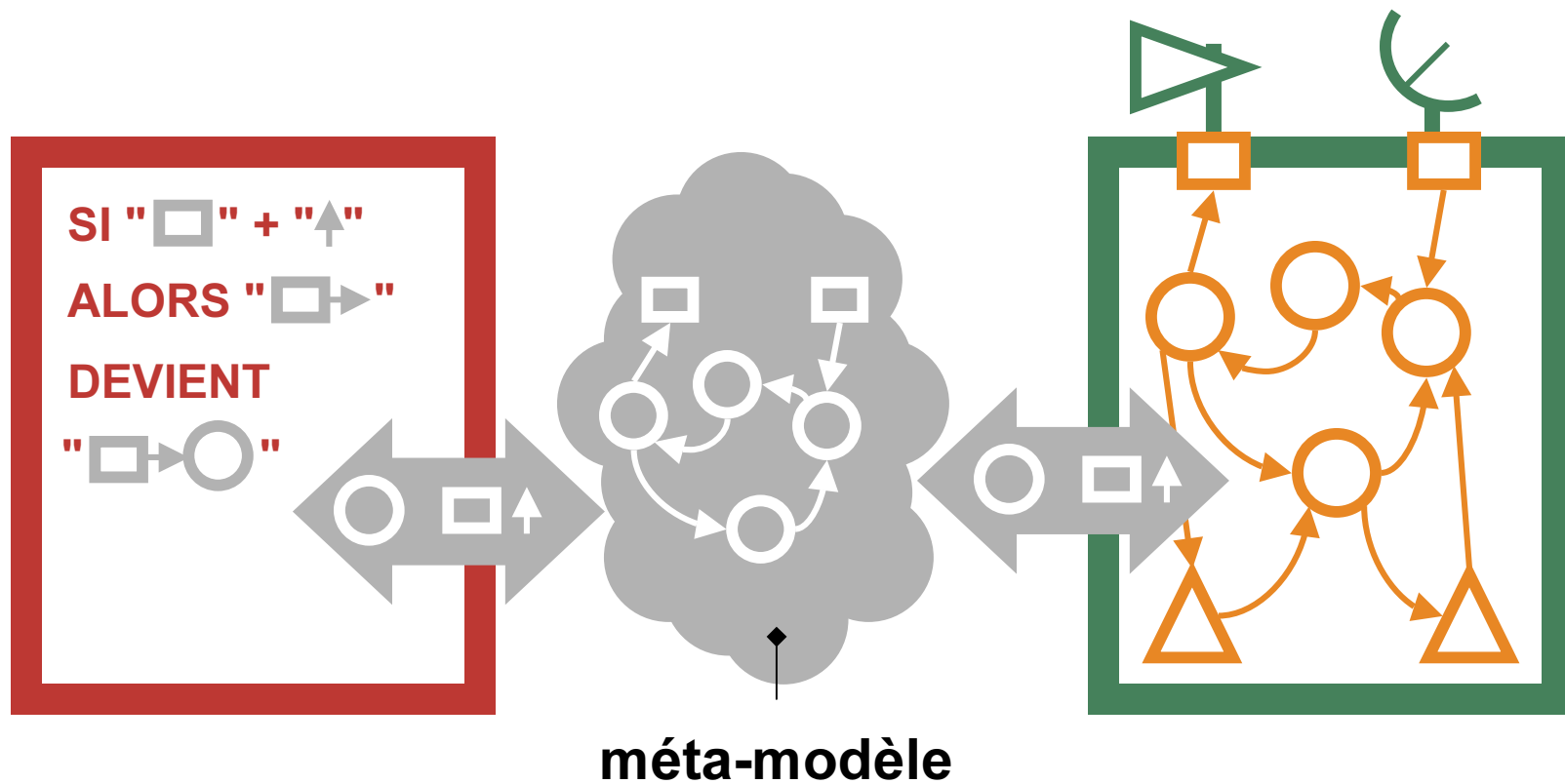
Par la réflexivité, le **programme** devient un **objet** qui peut être **manipulé**.

L'activité interne du système se **reflète** sur le méta-modèle.

Toute manipulation du méta-modèle se **reflète** sur le système.

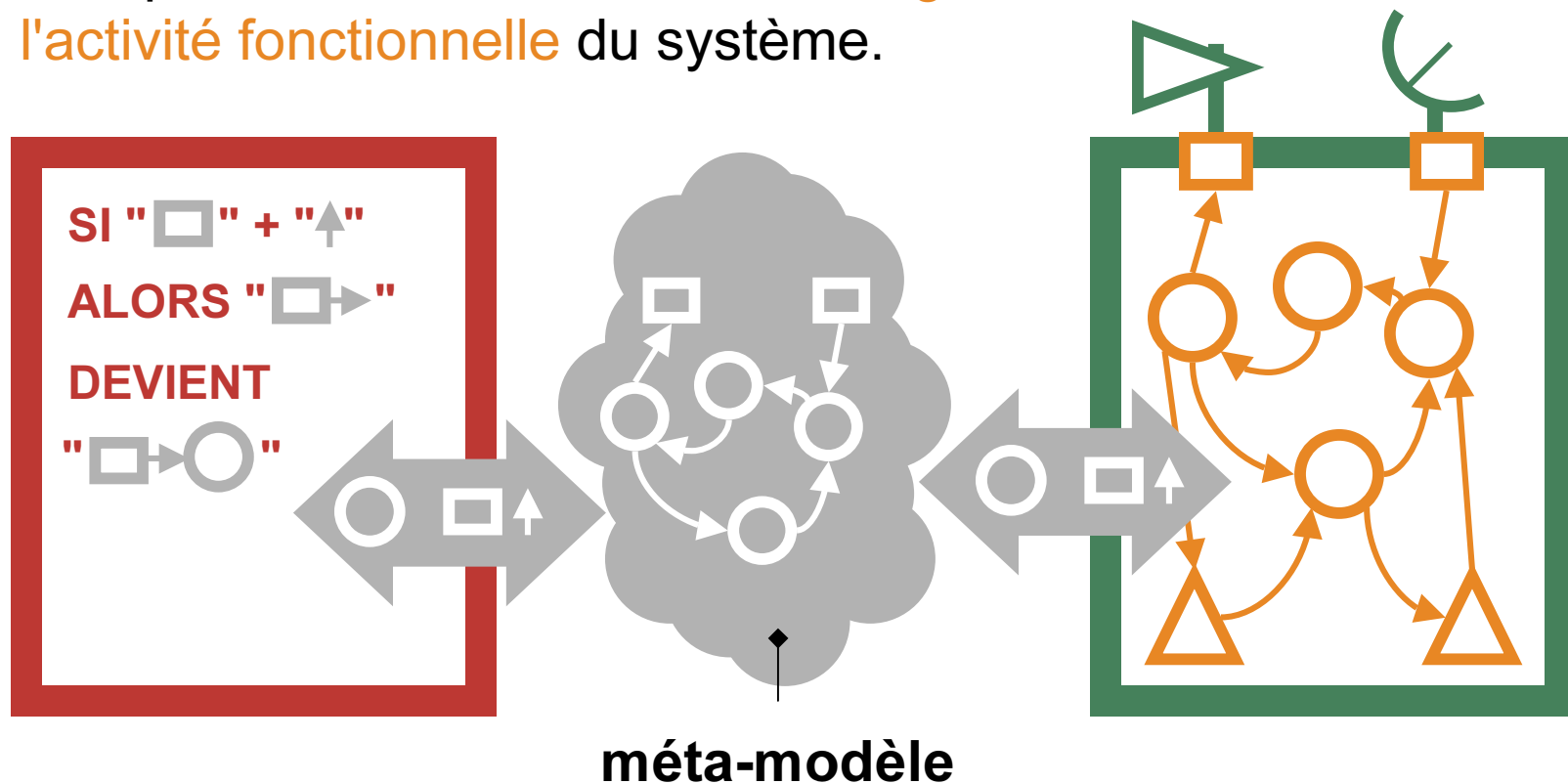


Réflexivité et orthogonalité



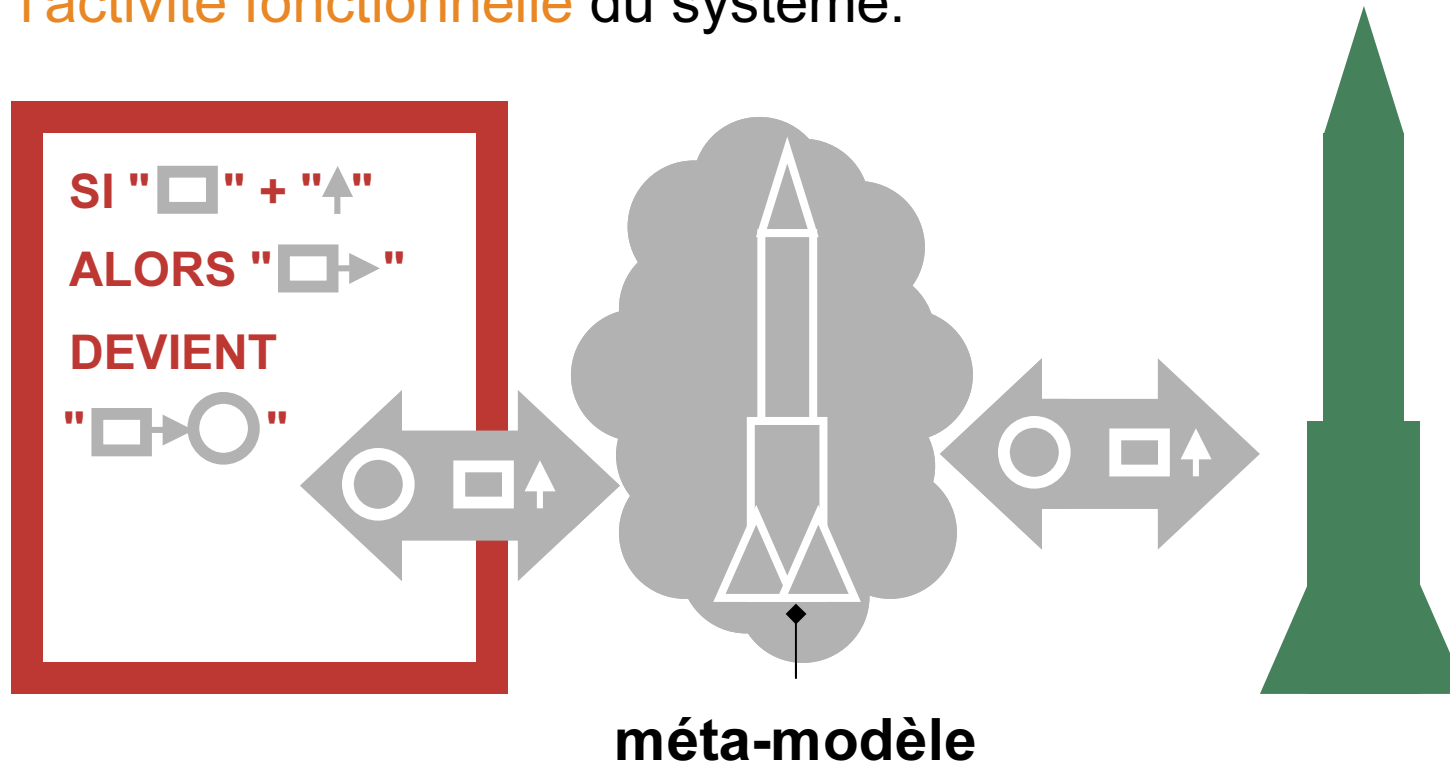
Réflexivité et orthogonalité

La **réflexivité** permet de définir des traitements qui manipulent le niveau de base **orthogonalement** à l'**activité fonctionnelle** du système.



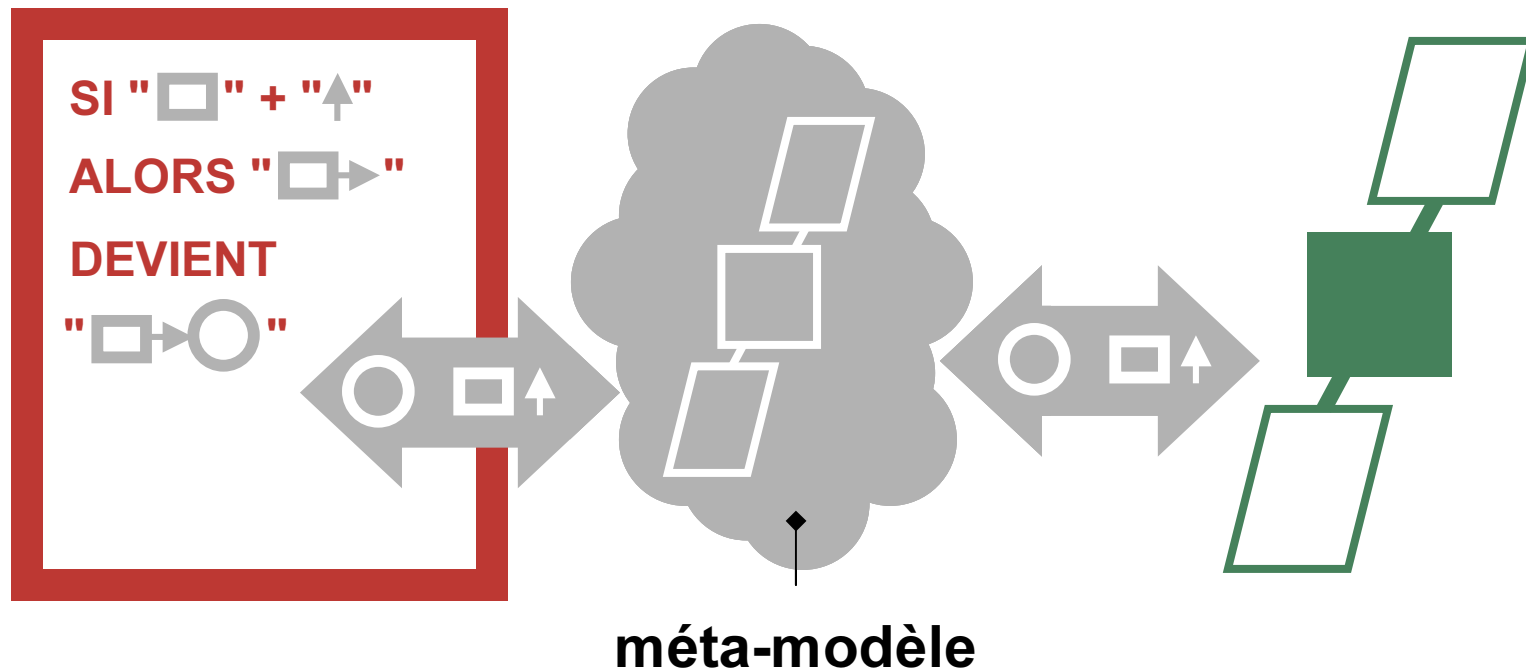
Réflexivité et orthogonalité

La **réflexivité** permet de définir des traitements qui manipulent le niveau de base **orthogonalement** à l'**activité fonctionnelle** du système.



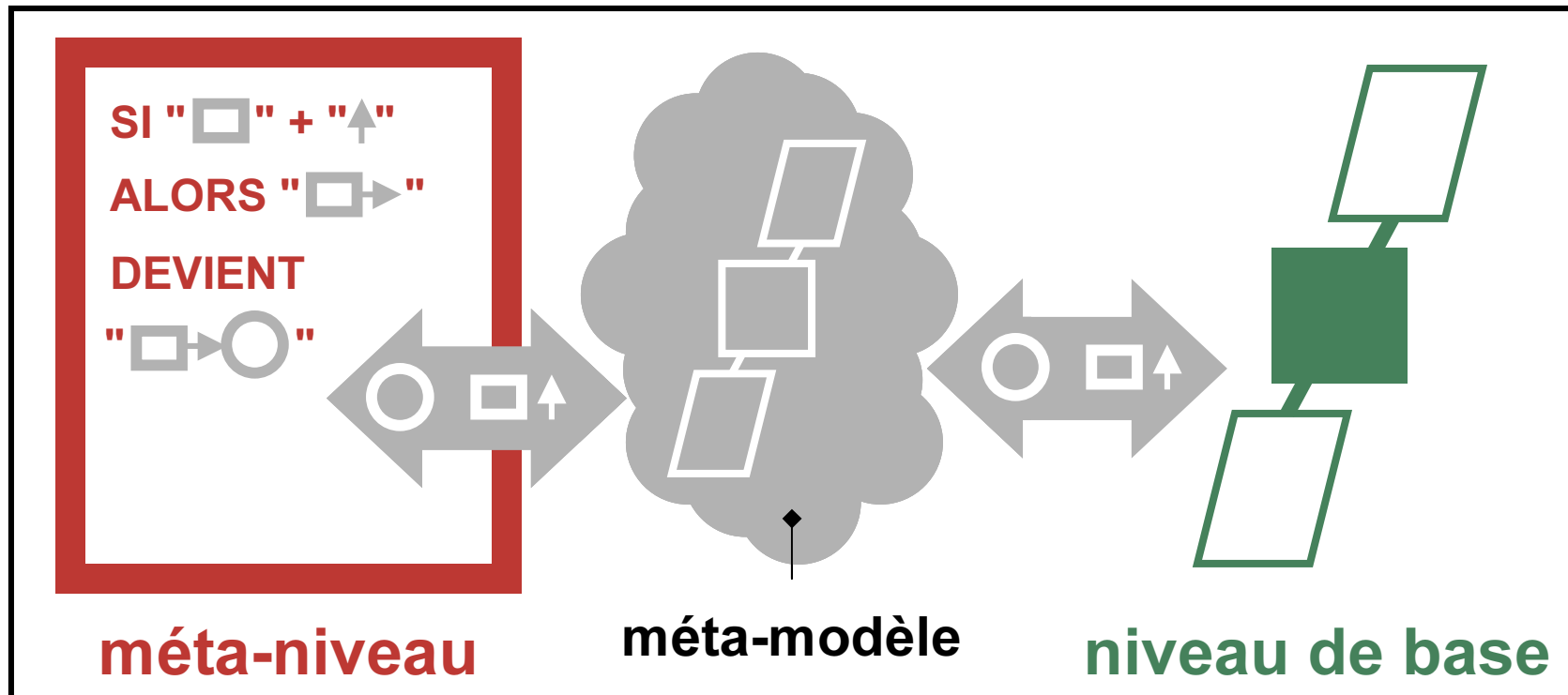
Réflexivité et orthogonalité

La **réflexivité** permet de définir des traitements qui manipulent le niveau de base **orthogonalement** à l'**activité fonctionnelle** du système.



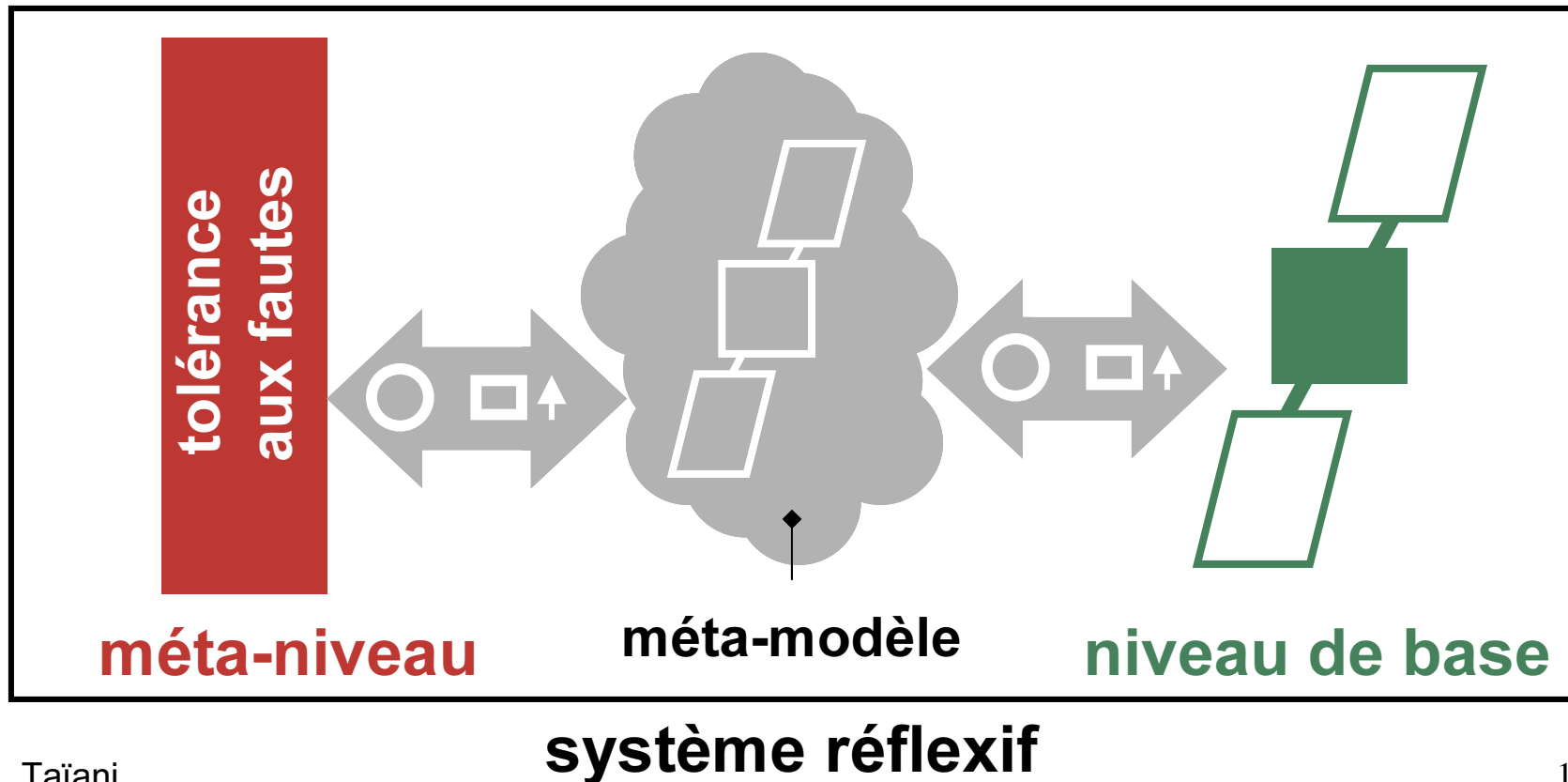
Réflexivité et orthogonalité

La **réflexivité** permet de définir des traitements qui manipulent le niveau de base **orthogonalement** à l'**activité fonctionnelle** du système.



Réflexivité et orthogonalité

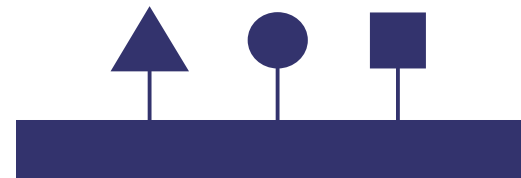
⇒ La réflexivité permet de rendre la tolérance aux fautes indépendante des fonctionnalités du système.



Architecture multi-niveaux

- Réaliser une architecture réflexive requiert de savoir où trouver **quelles** informations dans le système.

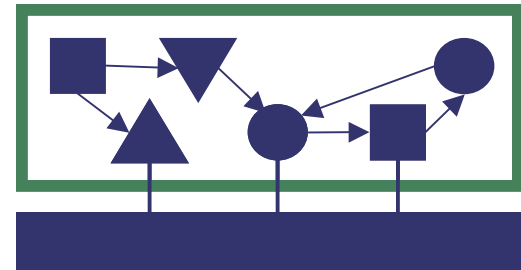
langage machine
(interruptions,
opérations logiques, ...)



Architecture multi-niveaux

- Réaliser une architecture réflexive requiert de savoir où trouver **quelles** informations dans le système.

langage machine
(interruptions,
opérations logiques, ...)

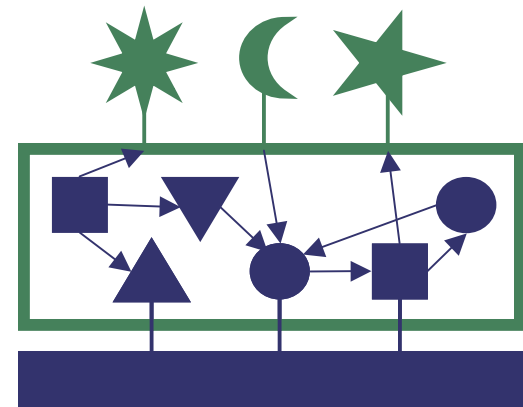


Architecture multi-niveaux

- Réaliser une architecture réflexive requiert de savoir où trouver **quelles** informations dans le système.

primitives noyau
(synchronisation,
gestion de la mémoire...)

langage machine
(interruptions,
opérations logiques, ...)



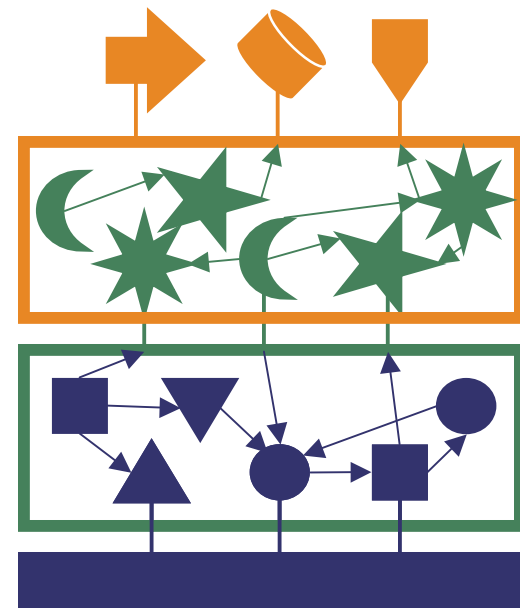
Architecture multi-niveaux

- Réaliser une architecture réflexive requiert de savoir où trouver **quelles** informations dans le système.

services intergiciels
(appels à distance, etc...)

primitives noyau
(synchronisation,
gestion de la mémoire...)

langage machine
(interruptions,
opérations logiques, ...)



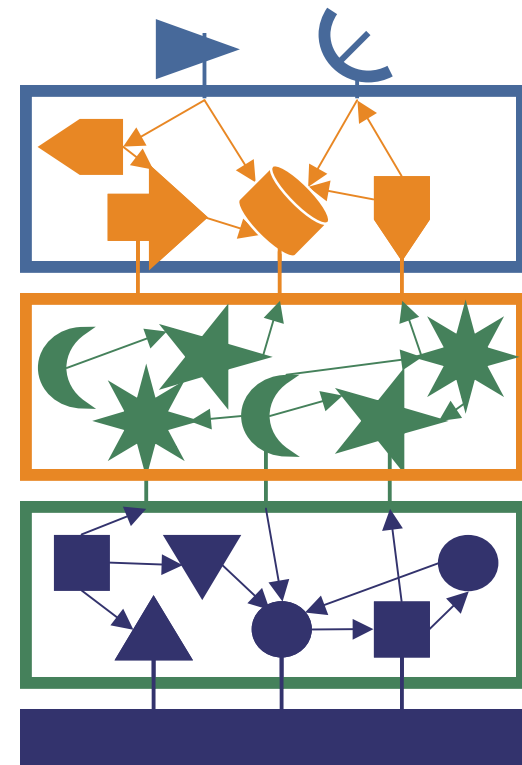
Architecture multi-niveaux

- Réaliser une architecture réflexive requiert de savoir où trouver **quelles** informations dans le système.

services intergiciels
(appels à distance, etc...)

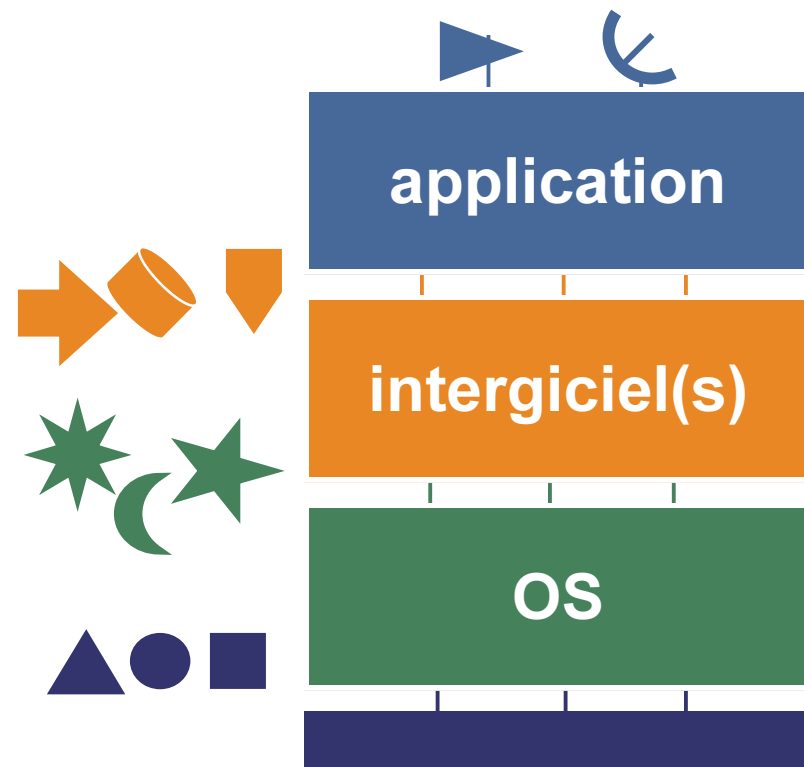
primitives noyau
(synchronisation,
gestion de la mémoire...)

langage machine
(interruptions,
opérations logiques, ...)



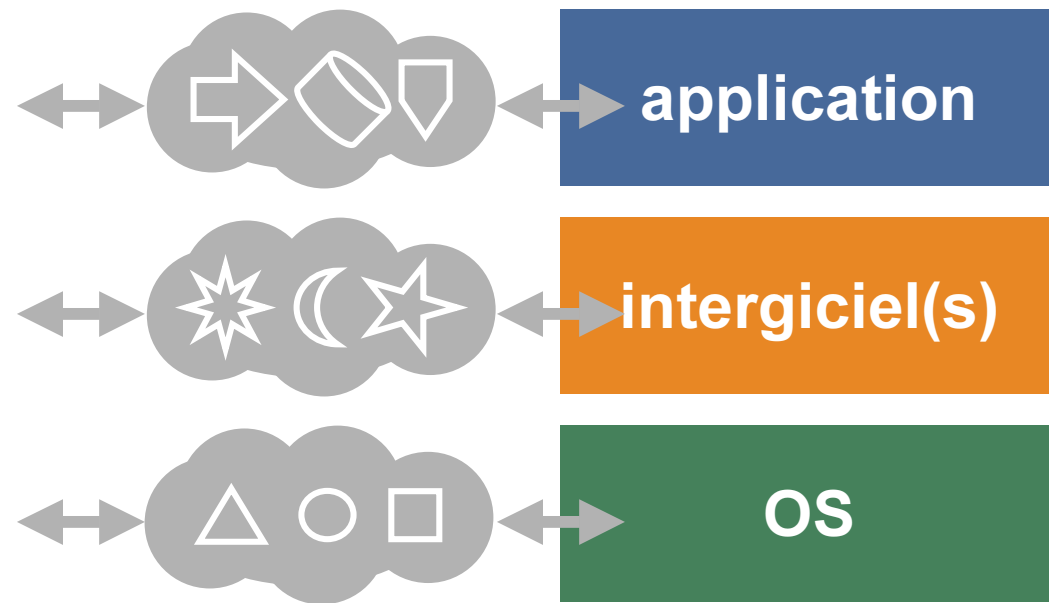
Architecture multi-niveaux

- Réaliser une architecture réflexive requiert de savoir où trouver **quelles** informations dans le système.
- Systèmes complexes : plusieurs niveaux **hétérogènes**
- Les activités des différents niveaux sont **liées** entre elle, mais ce «couplage» est **caché**.



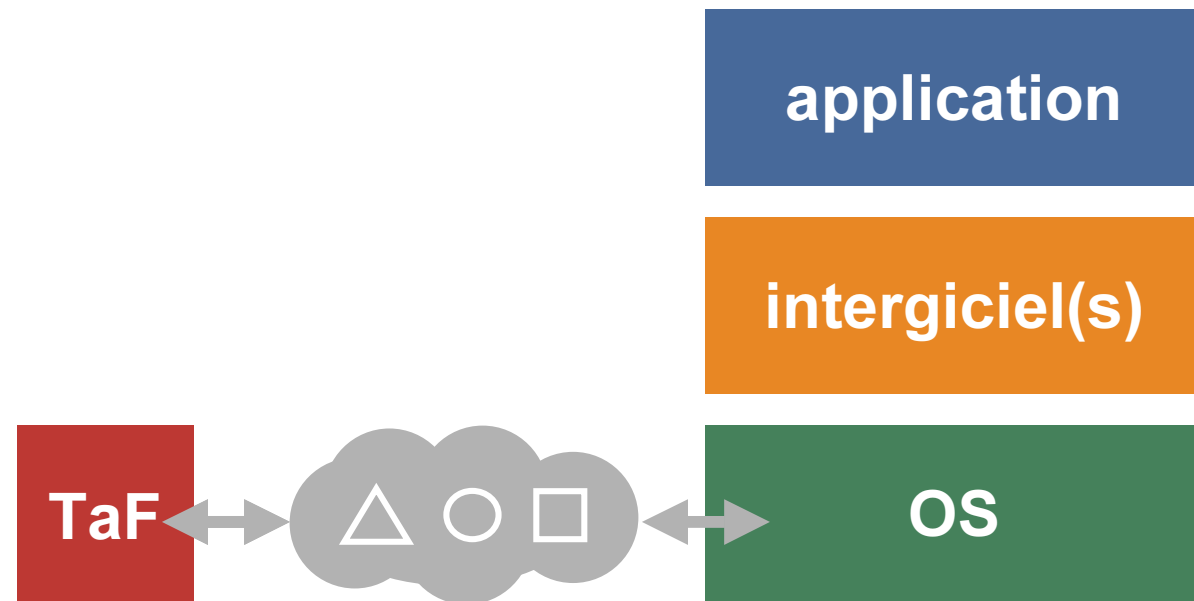
Réflexivité et Tolérance aux Fautes

- Dans les systèmes complexes :
 - Chaque composant a son propre modèle de programmation.
 - Les méta-modèles obtenus sont hétérogènes + incompatibles.



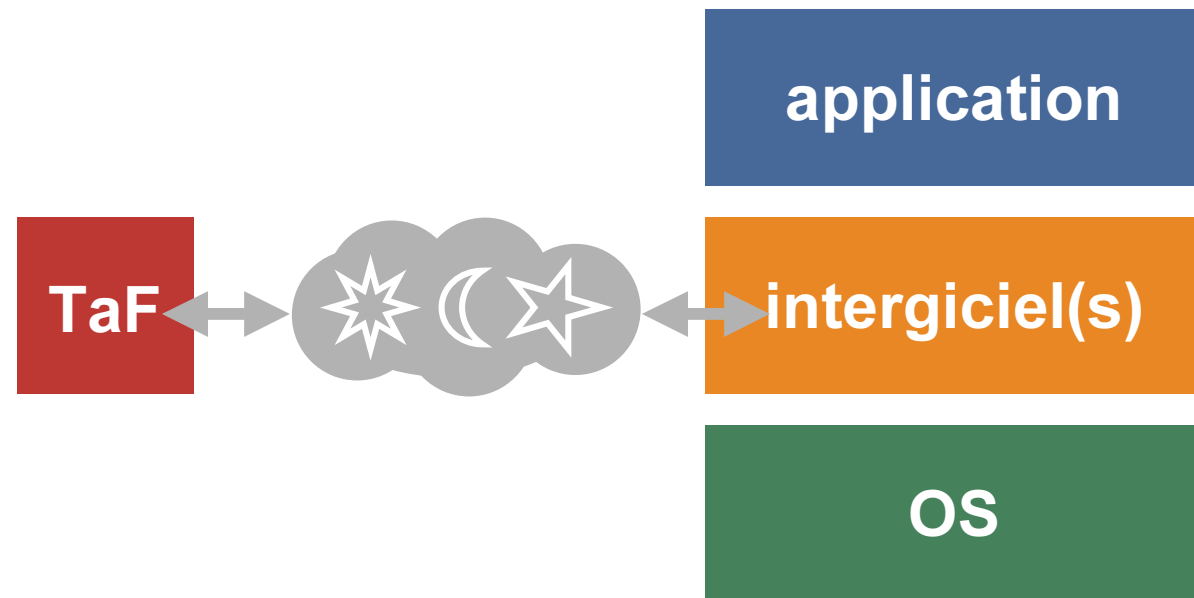
Réflexivité et Tolérance aux Fautes

- Jusqu'à maintenant, dans les systèmes complexes :
 - Un seul niveau d'abstraction était pris en compte.
 - La tolérance aux fautes réalisable était limitée.



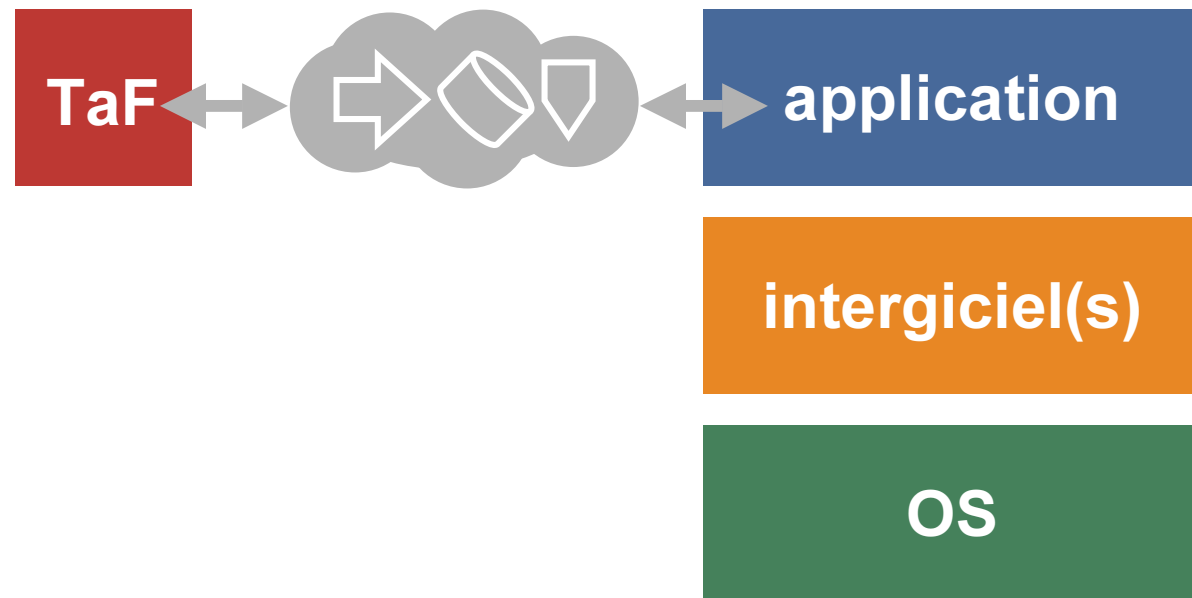
Réflexivité et Tolérance aux Fautes

- Jusqu'à maintenant, dans les systèmes complexes :
 - Un seul niveau d'abstraction était pris en compte.
 - La tolérance aux fautes réalisable était limitée.



Réflexivité et Tolérance aux Fautes

- Jusqu'à maintenant, dans les systèmes complexes :
 - Un seul niveau d'abstraction était pris en compte.
 - La tolérance aux fautes réalisable était limitée.



Reformulation de la problématique

- Notre objectif :

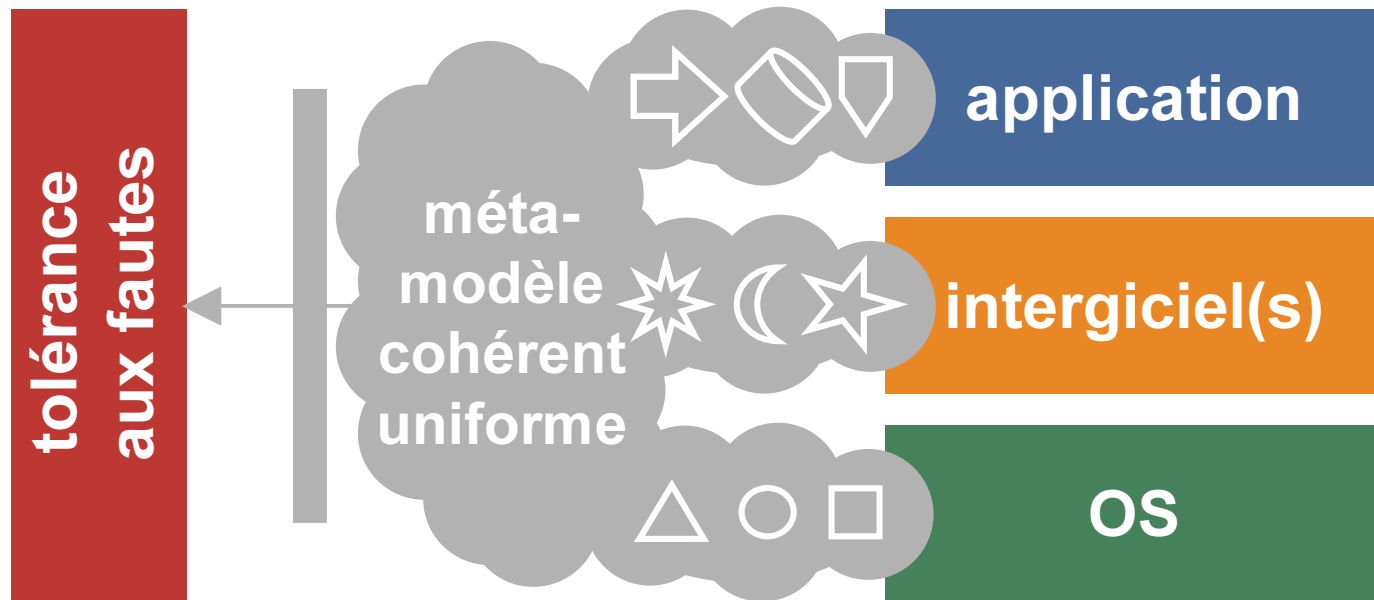
Une tolérance aux fautes qui prenne en compte le système dans son **intégralité**



Reformulation de la problématique

- Notre objectif :

Une tolérance aux fautes qui prenne en compte le système dans son **intégralité**



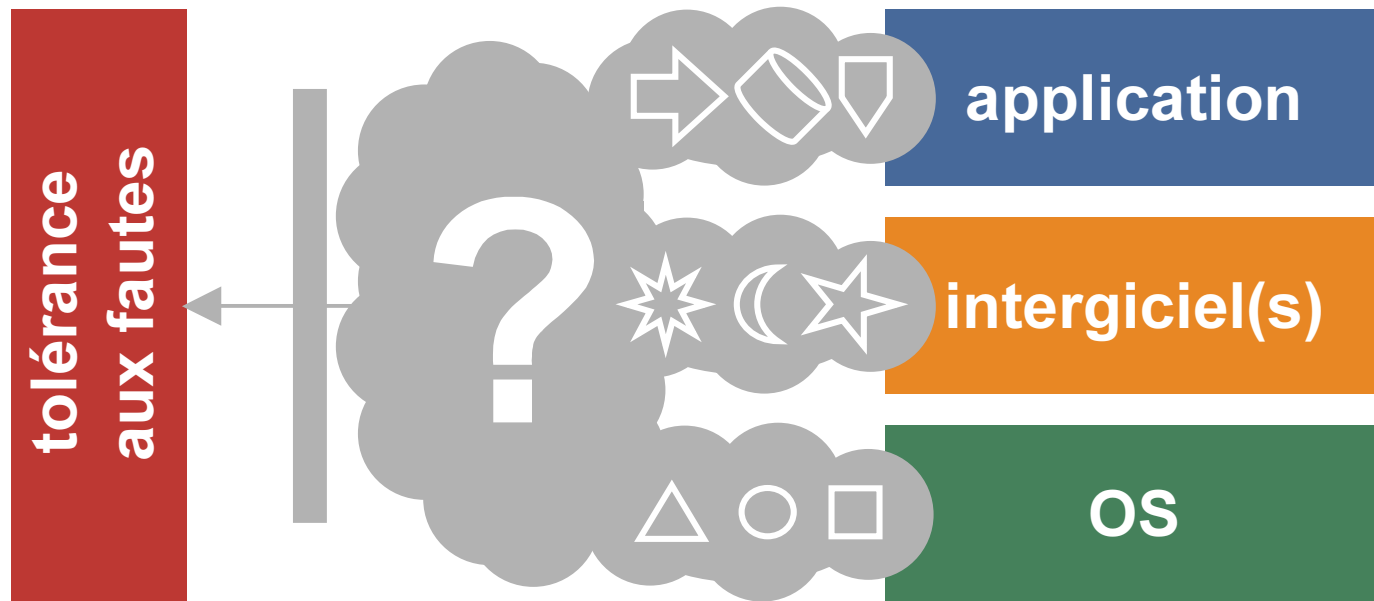
Reformulation de la problématique

⇒ **Comment construire un méta-modèle cohérent et uniforme offrant une vue globale d'un système complexe ?**



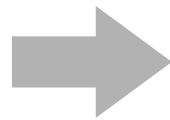
Reformulation de la problématique

Quels besoins pour la tolérance aux fautes ?

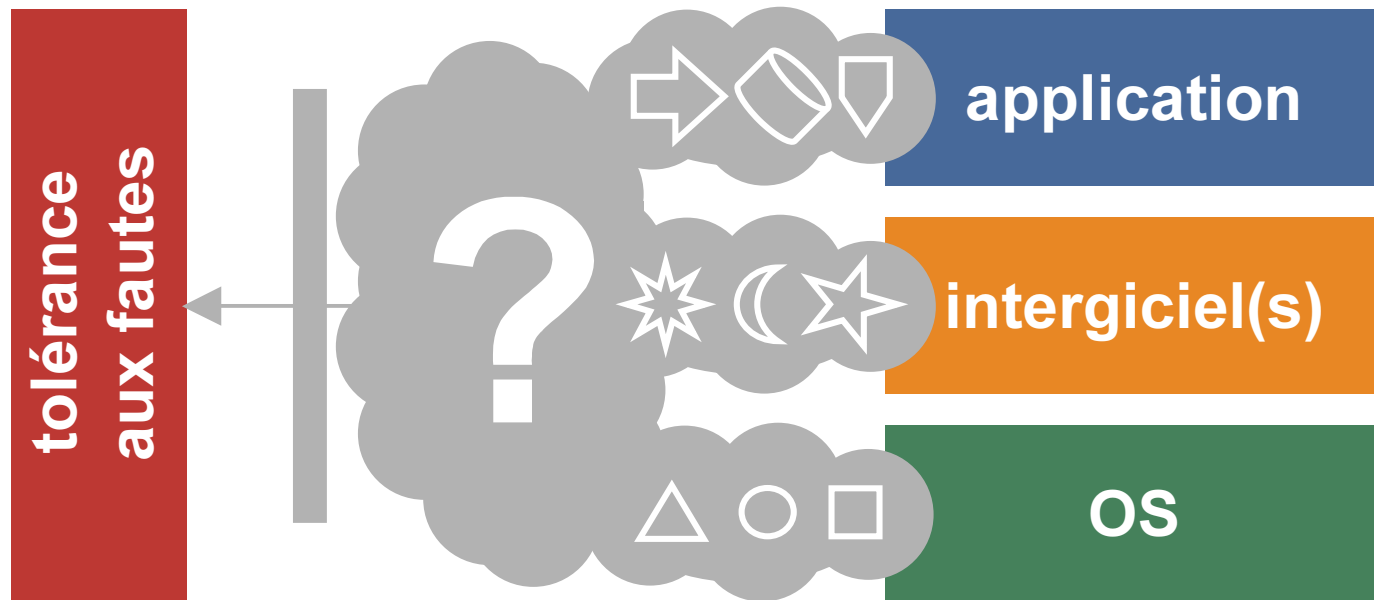


Reformulation de la problématique

Quels besoins pour la tolérance aux fautes ?

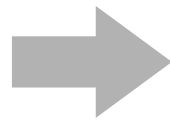


Quels éléments exporter ?
Comment les combiner ?

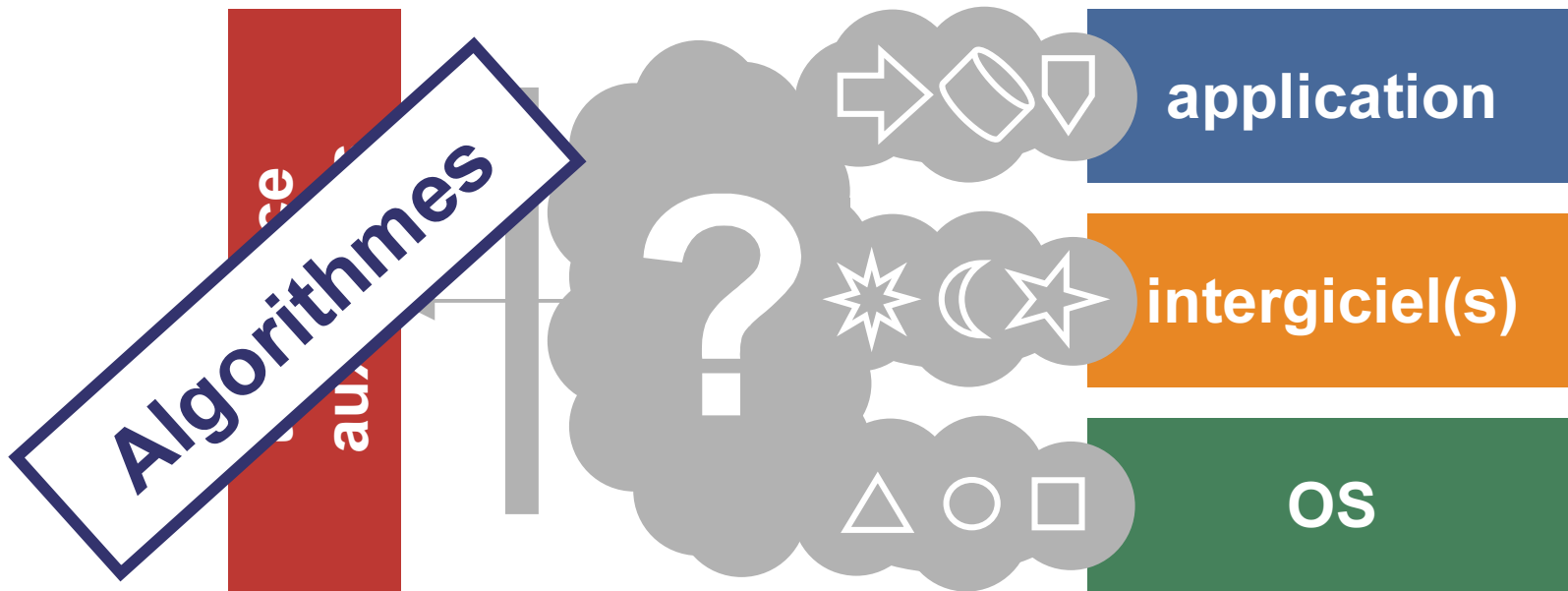


Reformulation de la problématique

Quels besoins pour la tolérance aux fautes ?

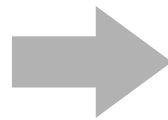


Quels éléments exporter ?
Comment les combiner ?

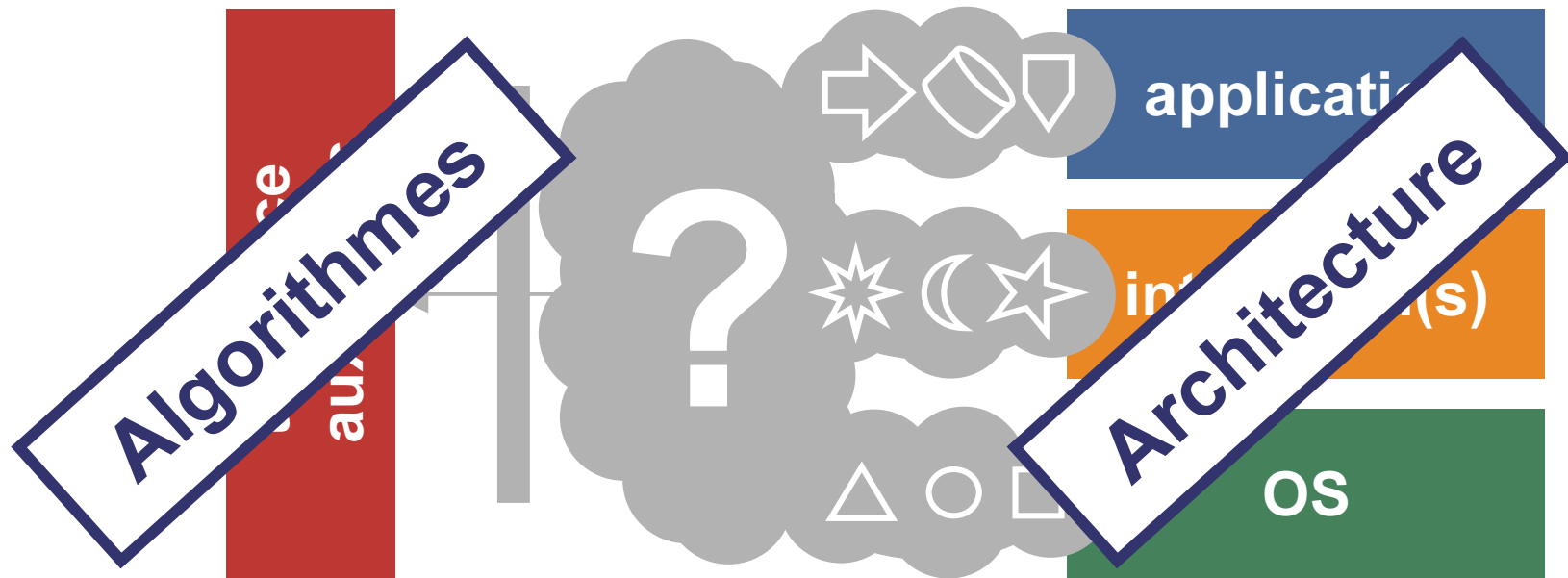


Reformulation de la problématique

Quels besoins pour la tolérance aux fautes ?

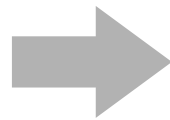


Quels éléments exporter ?
Comment les combiner ?

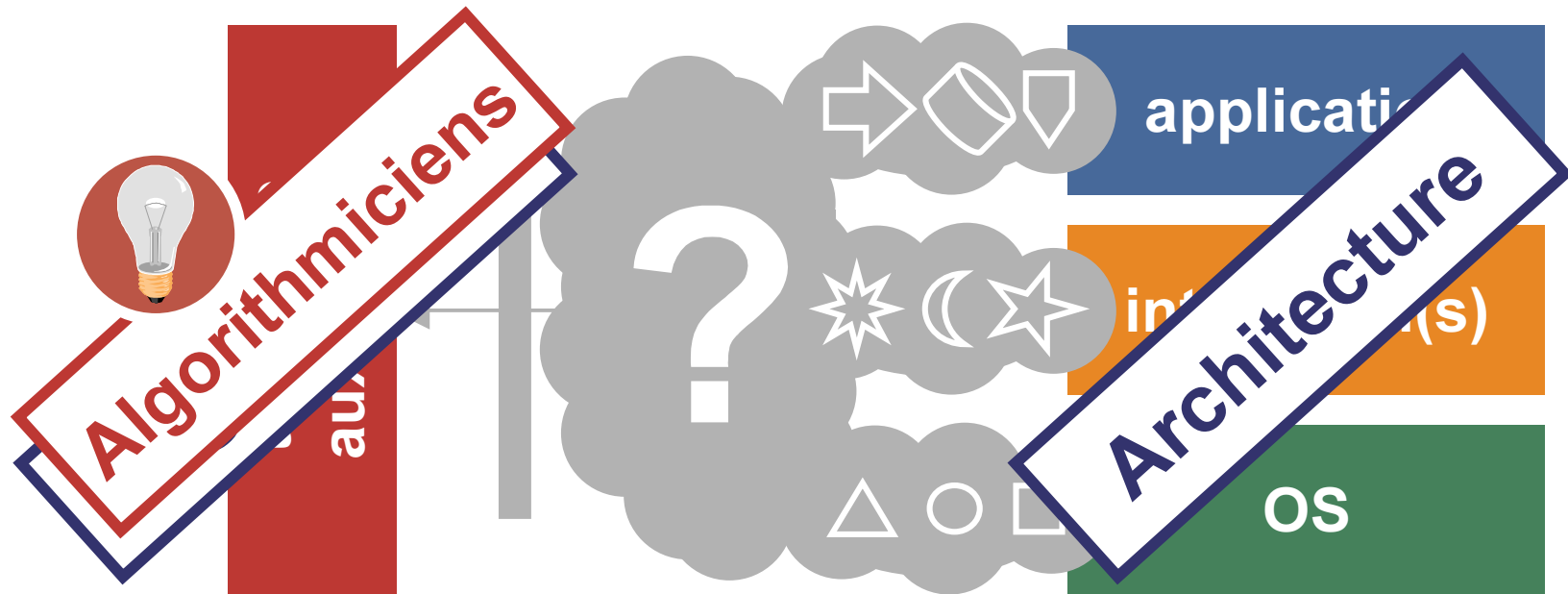


Reformulation de la problématique

Quels besoins pour la tolérance aux fautes ?

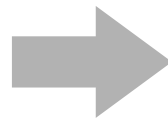


Quels éléments exporter ?
Comment les combiner ?

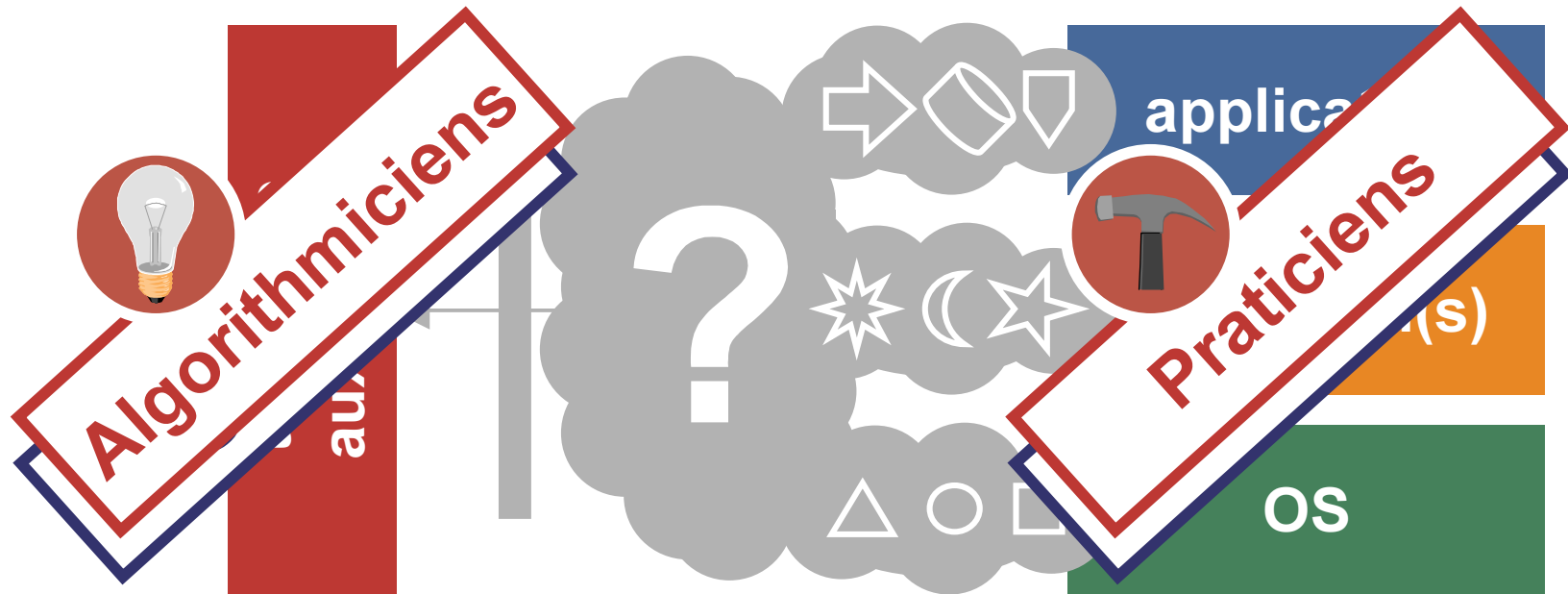


Reformulation de la problématique

Quels besoins pour la tolérance aux fautes ?



Quels éléments exporter ?
Comment les combiner ?



Plan

- (A) Qu'est-ce-que la réflexivité ?

- (B) Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes

- (C) Point de vue architectural :
La réflexivité dans les systèmes complexes

- (D) Application à un exemple concret :
Réplication multi-traitement d'une plate-forme CORBA / Linux

Besoins opérationnaires & performances

Besoins opératoires & performances

- Un algorithme de tolérance aux fautes générique :
 - Utilise un modèle de calcul de **haute abstraction** ;
 - Ex. : machines à états, processus, messages asynchrones
 - **Observe et agit** sur ce modèle de calcul.
 - Ex. : interception des messages, restauration de l'état d'un processus
 - Notions de **capacités** d'action et d'observation

Besoins opératoires & performances

- Un algorithme de tolérance aux fautes générique :
 - Utilise un modèle de calcul de **haute abstraction** ;
 - Ex. : machines à états, processus, messages asynchrones
 - **Observe et agit** sur ce modèle de calcul.
 - Ex. : interception des messages, restauration de l'état d'un processus
 - Notions de **capacités** d'action et d'observation

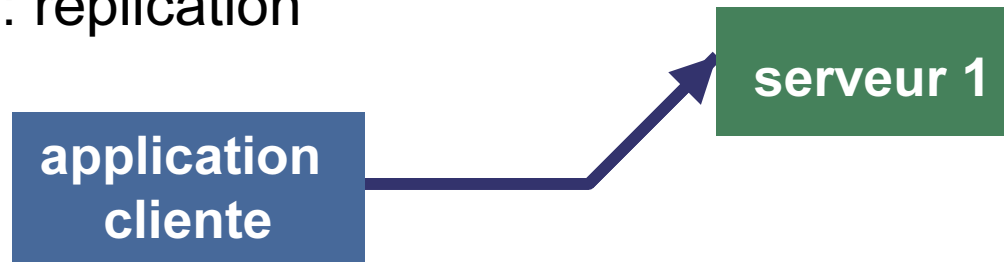
- Les capacités d'observation peuvent être **imparfaites**.
 - Mais trop d'approximation rend les performances inacceptables.
 - Dans un **système complexe** : la **précision est difficile** à obtenir dans une seule couche.

Besoins opératoires & performances

- Un algorithme de tolérance aux fautes générique :
 - Utilise un modèle de calcul de **haute abstraction** ;
 - Ex. : machines à états, processus, messages asynchrones
 - **Observe et agit** sur ce modèle de calcul.
 - Ex. : interception des messages, restauration de l'état d'un processus
 - Notions de **capacités** d'action et d'observation
- Les capacités d'observation peuvent être **imparfaites**.
 - Mais trop d'approximation rend les performances inacceptables.
 - Dans un **système complexe** : la **précision est difficile** à obtenir dans une seule couche.
- **La qualité des capacités d'action et d'observation d'un algorithme doit être maîtrisée.**

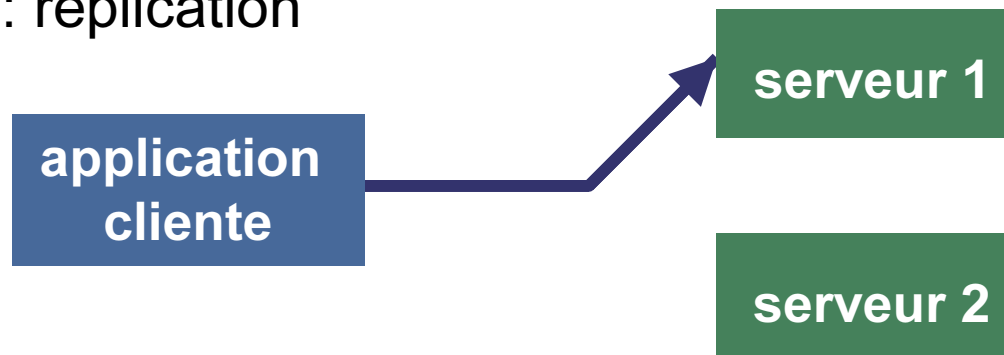
Mieux maîtriser les besoins de la TaF

- Notre proposition : **expliciter** les besoins en définissant l'**empreinte réflexive** d'une famille d'algorithmes
 - Ensemble des capacités réflexives requises par la famille
 - **Découple** cœur algorithmique / besoins opératoires concrets
 - Indépendante de toute architecture concrète
- Exemple : réplication



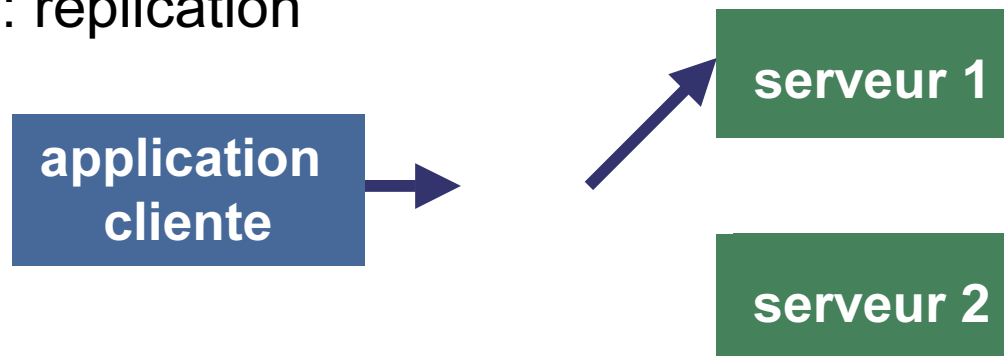
Mieux maîtriser les besoins de la TaF

- Notre proposition : **expliciter** les besoins en définissant l'**empreinte réflexive** d'une famille d'algorithmes
 - Ensemble des capacités réflexives requises par la famille
 - **Découple** cœur algorithmique / besoins opératoires concrets
 - Indépendante de toute architecture concrète
- Exemple : réplication



Mieux maîtriser les besoins de la TaF

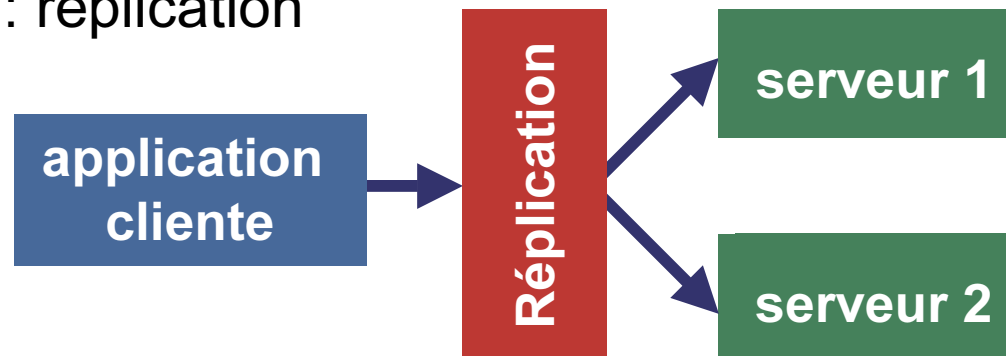
- Notre proposition : **expliciter** les besoins en définissant l'**empreinte réflexive** d'une famille d'algorithmes
 - Ensemble des capacités réflexives requises par la famille
 - **Découple** cœur algorithmique / besoins opératoires concrets
 - Indépendante de toute architecture concrète
- Exemple : réplication



Mieux maîtriser les besoins de la TaF

- Notre proposition : **expliciter** les besoins en définissant l'**empreinte réflexive** d'une famille d'algorithmes
 - Ensemble des capacités réflexives requises par la famille
 - **Découple** cœur algorithmique / besoins opératoires concrets
 - Indépendante de toute architecture concrète

- Exemple : réplication



- empreinte réflexive : a) interception des messages vers serveurs
b) transfert d'état
c) gestion du non déterminisme ... etc.

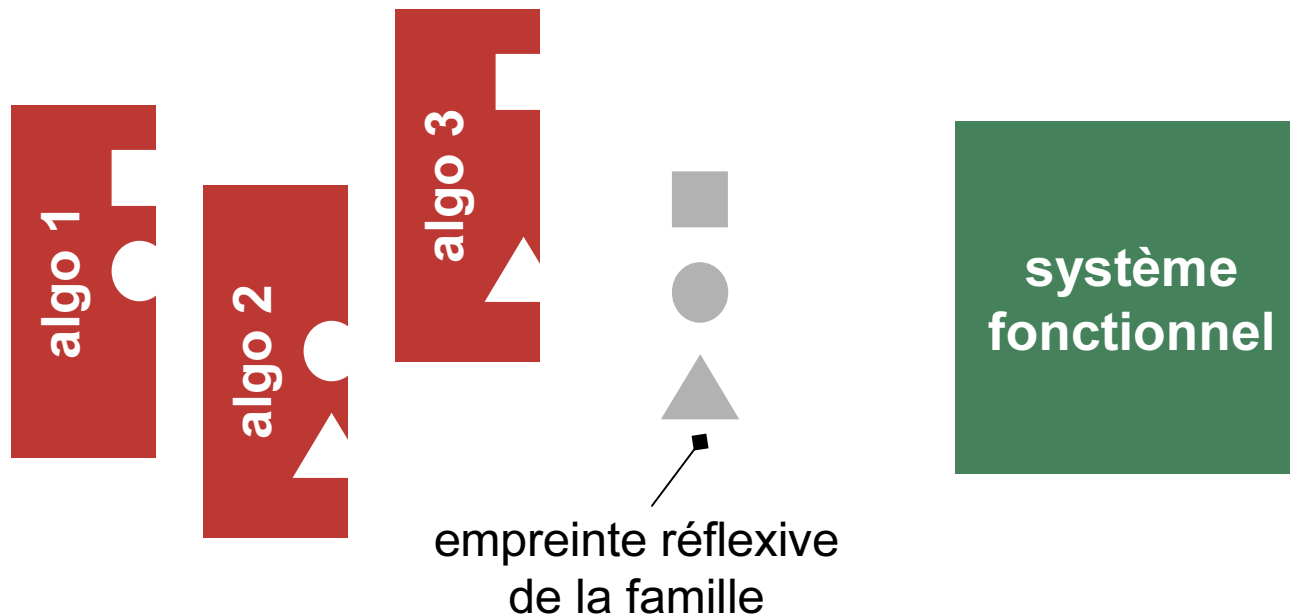
Empreintes réflexives & adaptabilité

- **Empreinte** réflexive d'une **famille** de mécanismes
 - **Découple** le choix de l'algorithme de ses besoins opératoires.



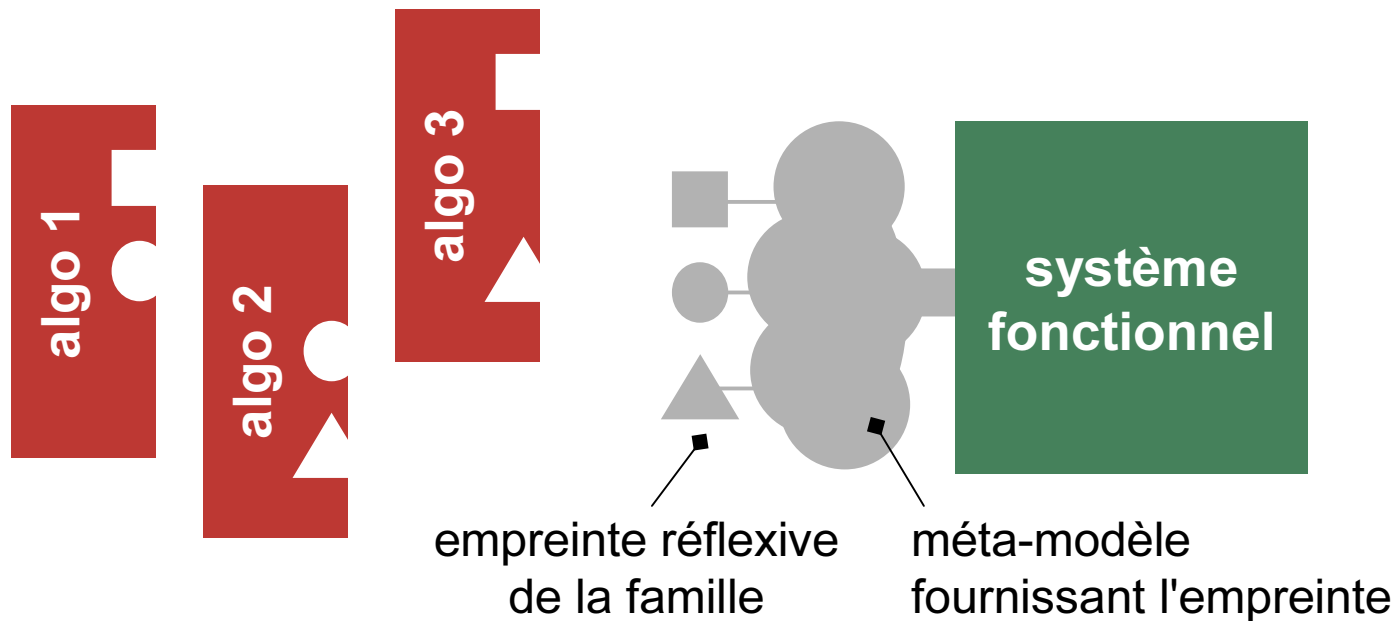
Empreintes réflexives & adaptabilité

- Empreinte réflexive d'une famille de mécanismes
 - Découple le choix de l'algorithme de ses besoins opératoires.



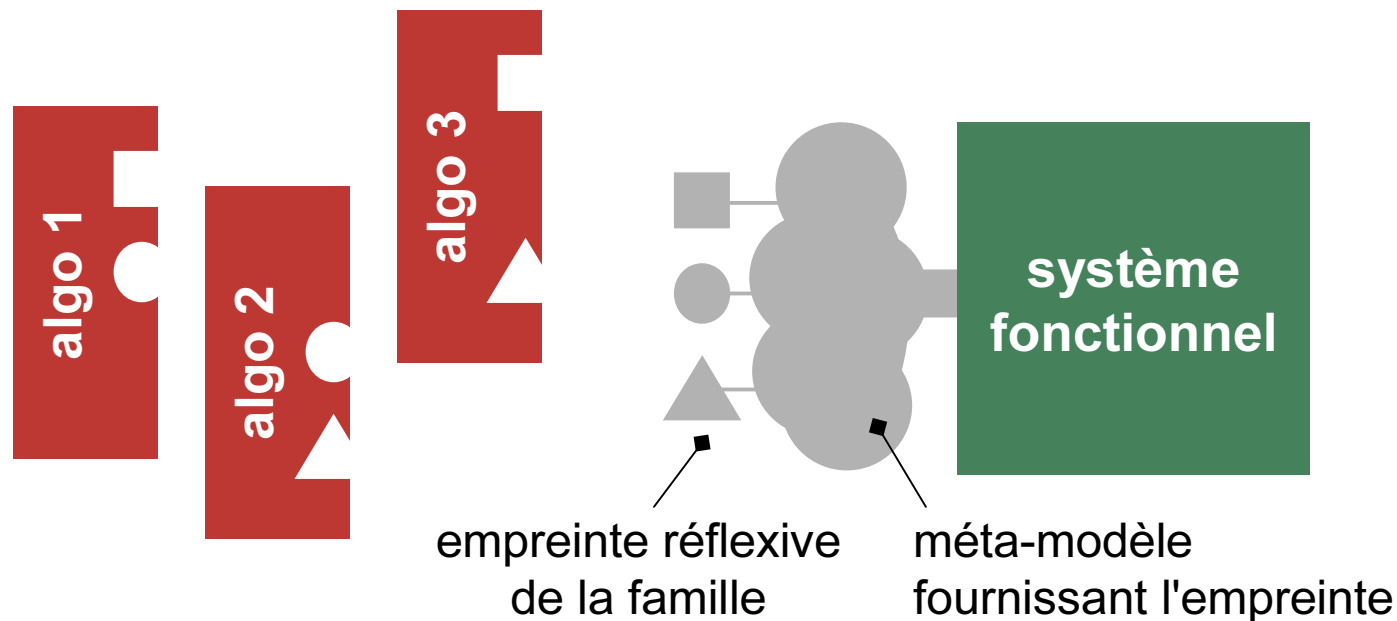
Empreintes réflexives & adaptabilité

- **Empreinte** réflexive d'une **famille** de mécanismes
 - **Découple** le choix de l'algorithme de ses besoins opératoires.



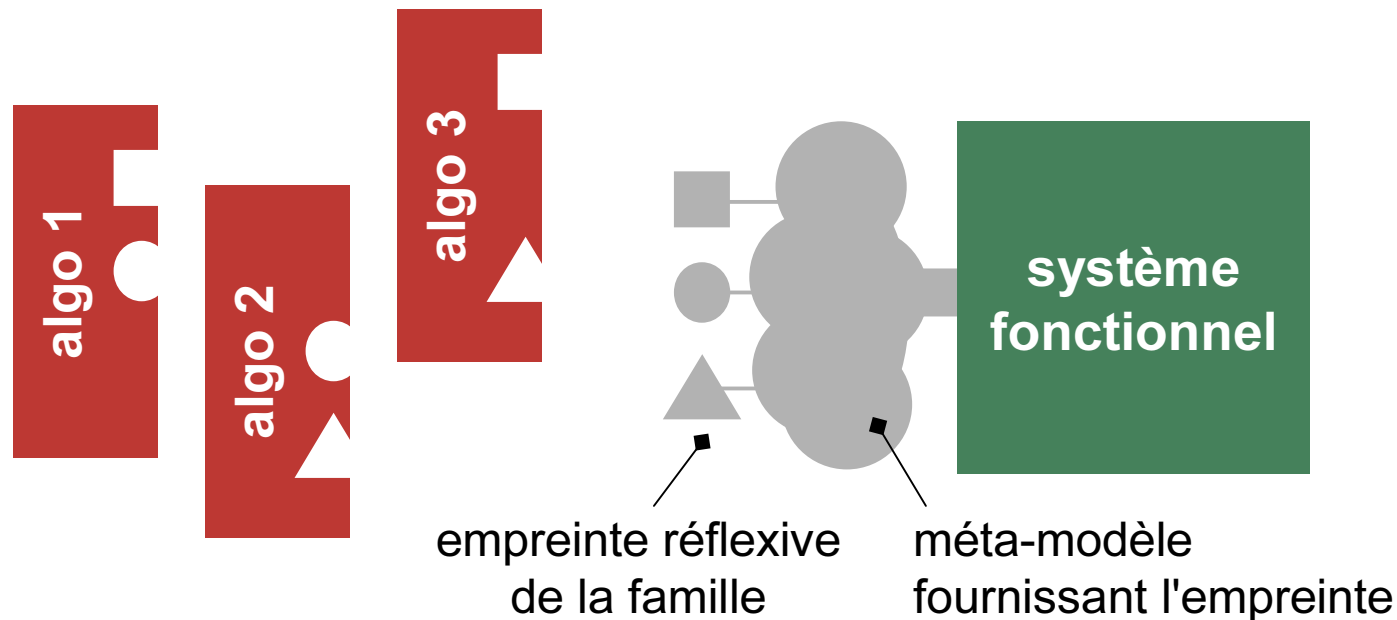
Empreintes réflexives & adaptabilité

- **Empreinte** réflexive d'une **famille** de mécanismes
 - **Découple** le choix de l'algorithme de ses besoins opératoires.
 - Réutilisation de l'instrumentation \Rightarrow **meilleure qualité**, coûts \searrow



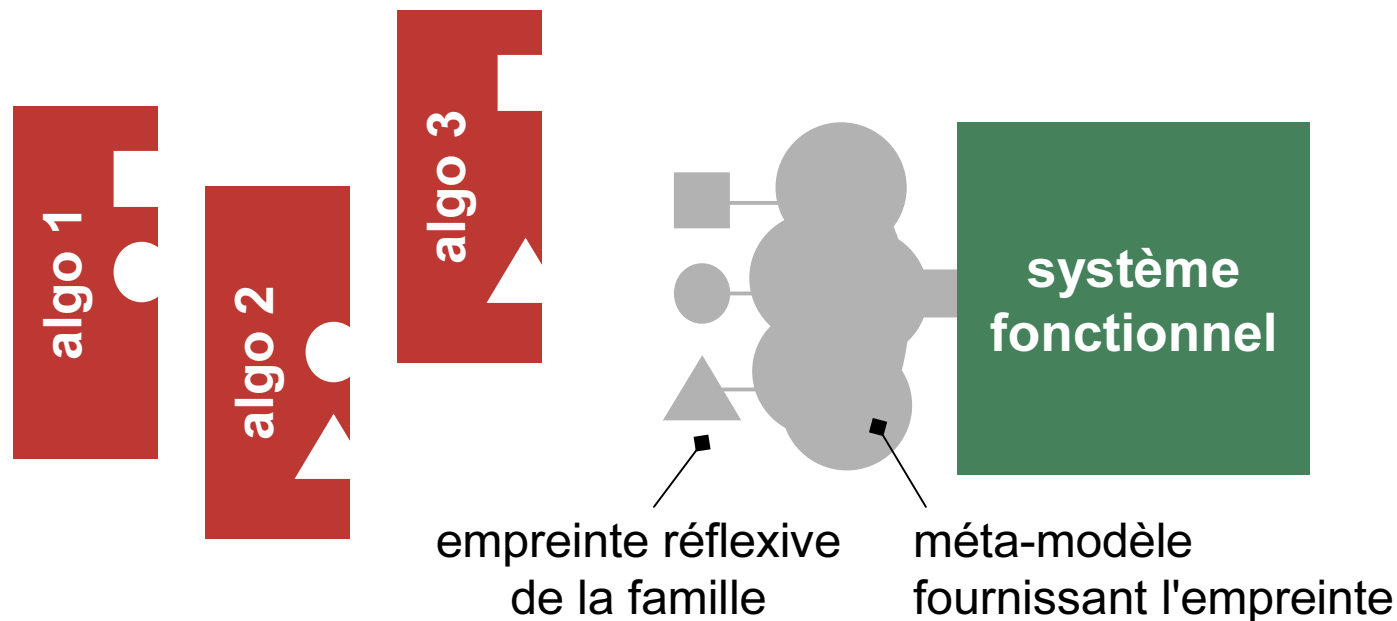
Empreintes réflexives & adaptabilité

- **Empreinte** réflexive d'une **famille** de mécanismes
 - **Découple** le choix de l'algorithme de ses besoins opératoires.
 - Réutilisation de l'instrumentation \Rightarrow **meilleure qualité**, coûts \searrow
 - Rend les algorithmes **adaptables** en cours de développement.



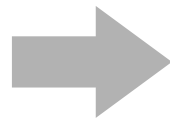
Empreintes réflexives & adaptabilité

- **Empreinte** réflexive d'une **famille** de mécanismes
 - **Découple** le choix de l'algorithme de ses besoins opératoires.
 - Réutilisation de l'instrumentation \Rightarrow **meilleure qualité**, coûts \searrow
 - Rend les algorithmes **adaptables** en cours de développement.
 - Ouvre la voie à l'adaptation dynamique.

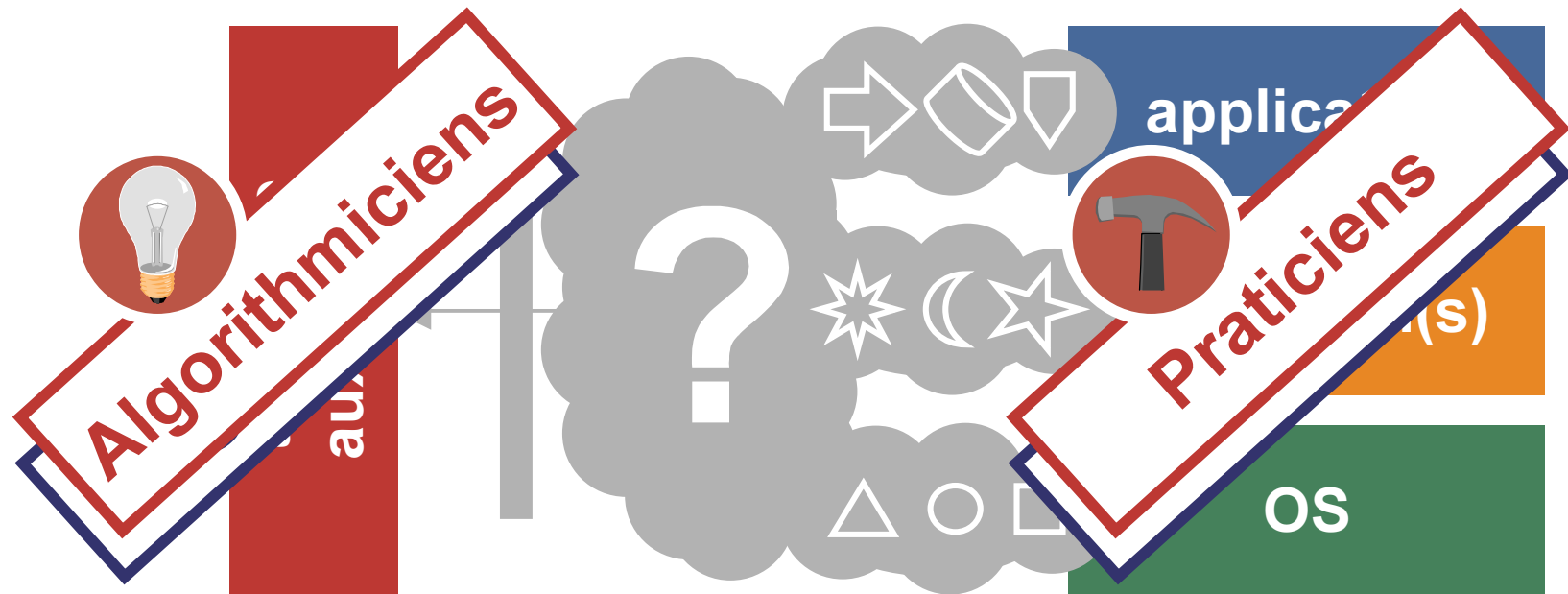


Retour sur la problématique

Quels besoins pour la tolérance aux fautes ?



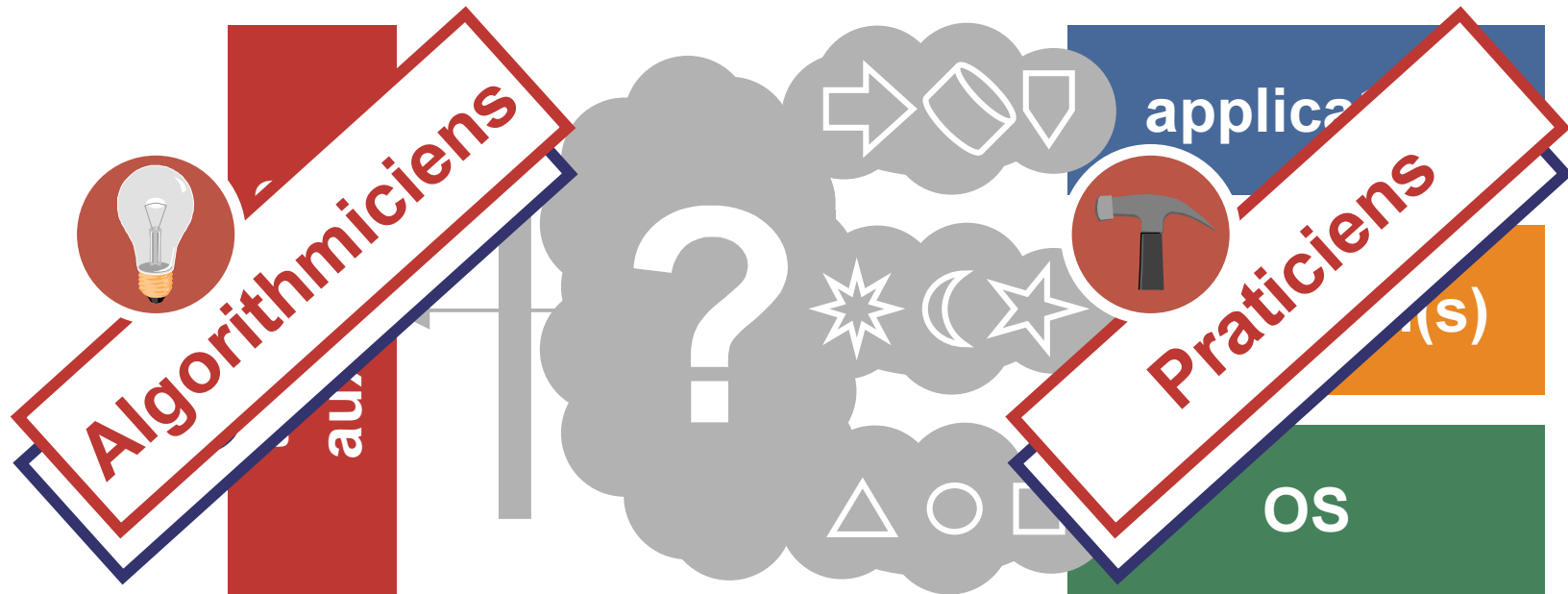
Quels éléments exporter ?
Comment les combiner ?



Retour sur la problématique

Les empreintes réflexives spécifient les besoins réflexifs pour rendre la TaF adaptable.

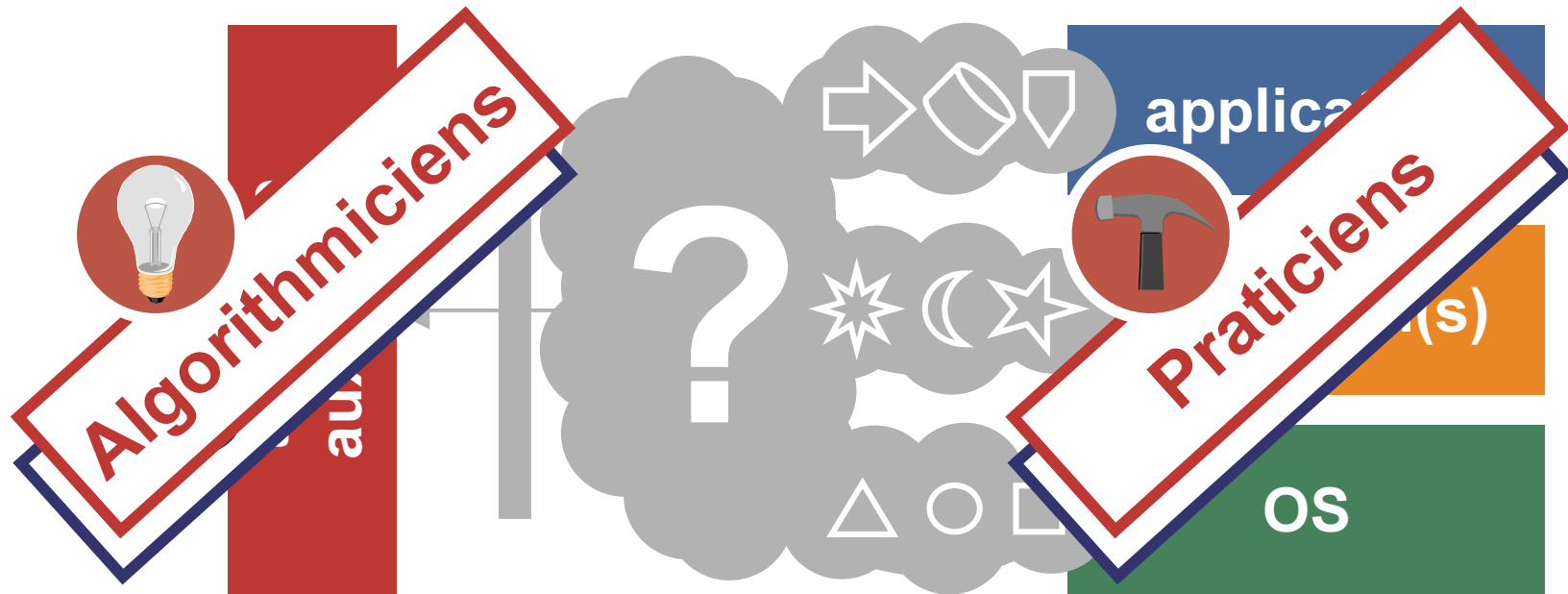
Quels éléments exporter ?
Comment les combiner ?



Retour sur la problématique

Les empreintes réflexives spécifient les besoins réflexifs pour rendre la TaF adaptable.

⇒ Avec quels éléments concrets réaliser une empreinte ?

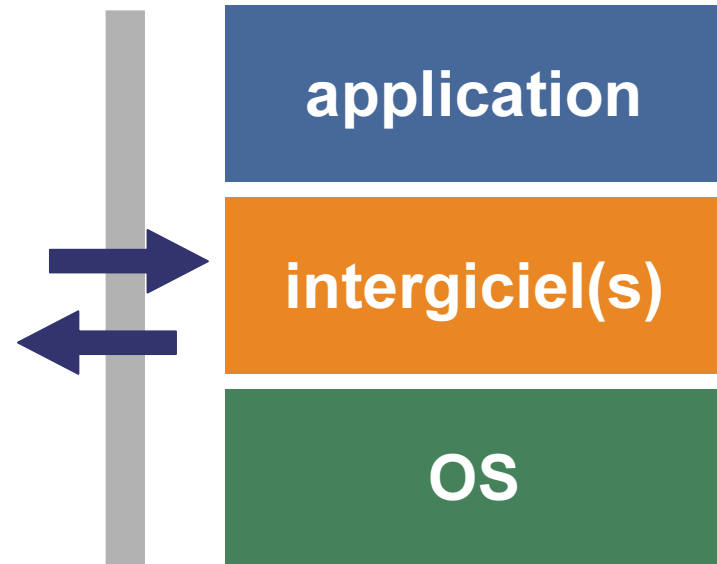


Plan

- (A) Qu'est-ce-que la réflexivité ?
- (B) Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes
- (C) Point de vue architectural :
La réflexivité dans les systèmes complexes
- (D) Application à un exemple concret :
Réplication multitraitements d'un serveur CORBA

Abstraction et information

- Les niveaux d'abstraction sont hétérogènes.
⇒ L'information disponible est hétérogène.
- Hauts niveaux :
 - 😊 Sémantique riche
 - ☹ Mais de l'information manque.
- Bas niveaux :
 - 😊 Information complète
 - ☹ Mais la sémantique manque.

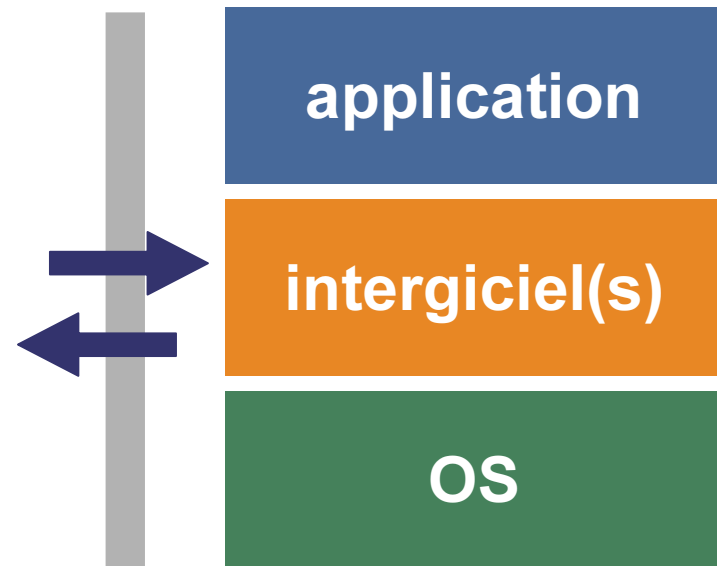


Abstraction et information

- Les niveaux d'abstraction sont hétérogènes.
⇒ L'information disponible est hétérogène.

- Hauts niveaux :
 - 😊 Sémantique riche
 - ☹ Mais de l'information manque.

- Bas niveaux :
 - 😊 Information complète
 - ☹ Mais la sémantique manque.



- **Les approches mono-niveaux ne fonctionnent pas :**
 - ➔ Manque d'information : impossibilité de réaliser la TaF
 - ➔ Manque de sémantique ⇒ coûts de performance trop élevés

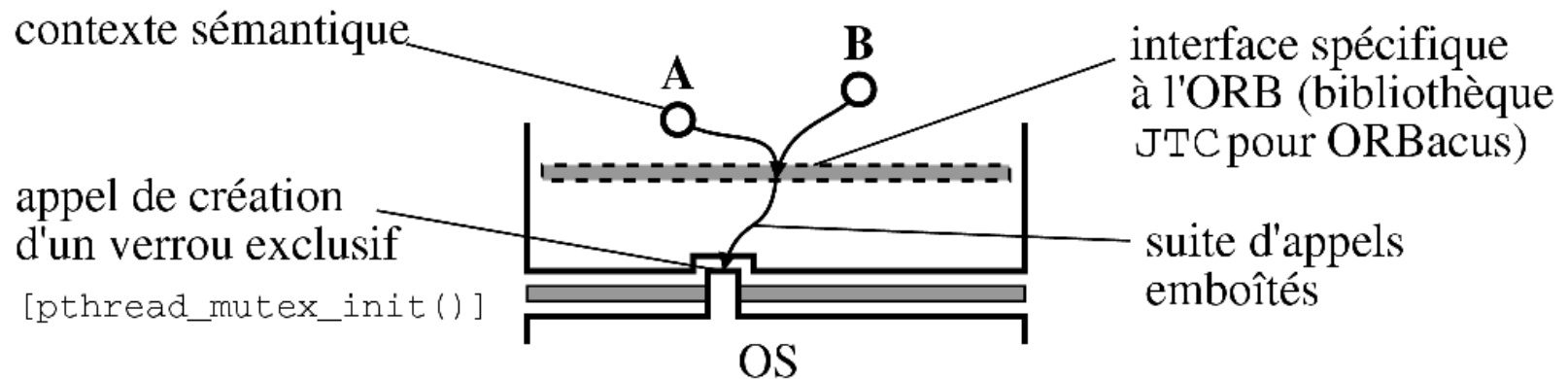
Transcendance et interception

- La **logique globale** du système **transcende** les **bas niveaux**.

→ Par exemple : connexion réseau (socket) au niveau OS

- Soit début d'une requête CORBA
- Soit initialisation d'une fenêtre X11

- Notion de **contexte sémantique** : exemple bus à objets



- Instrumentation par interception des bas niveaux :

Nécessité de savoir pour le compte de qui on intercepte.

Utilisation des liens inter-niveaux

Pour réaliser une empreinte réflexive la compréhension des **liens entre les niveaux** (transcendance) permet de **combiner l'intelligence** des hauts niveaux avec **la force brute** des bas niveaux.

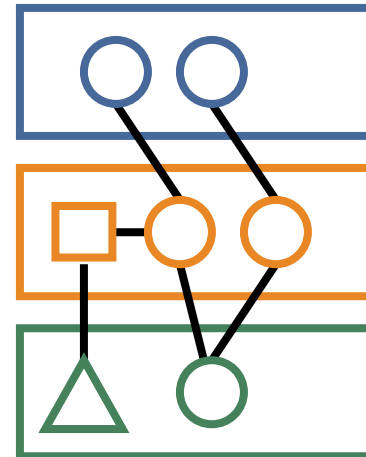
Application

Intergiciel(s)

OS

Utilisation des liens inter-niveaux

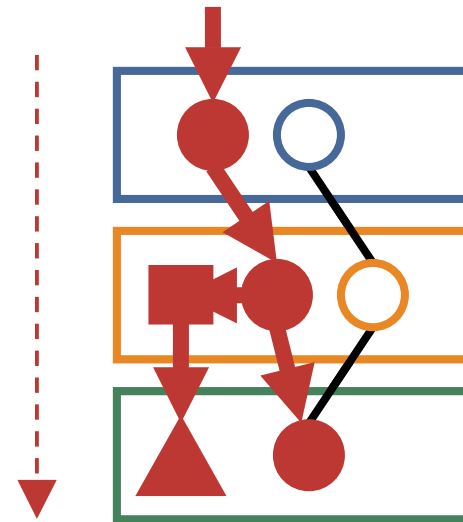
Pour réaliser une empreinte réflexive la compréhension des **liens entre les niveaux** (transcendance) permet de **combiner l'intelligence** des hauts niveaux avec **la force brute** des bas niveaux.



Utilisation des liens inter-niveaux

Pour réaliser une empreinte réflexive la compréhension des **liens entre les niveaux** (transcendance) permet de **combinaison** l'**intelligence** des hauts niveaux avec **la force brute** des bas niveaux.

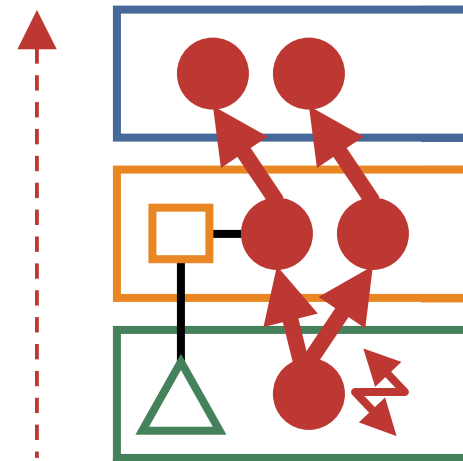
- Utilisation descendante
 - capture d'état
 - maîtrise du non-déterminisme



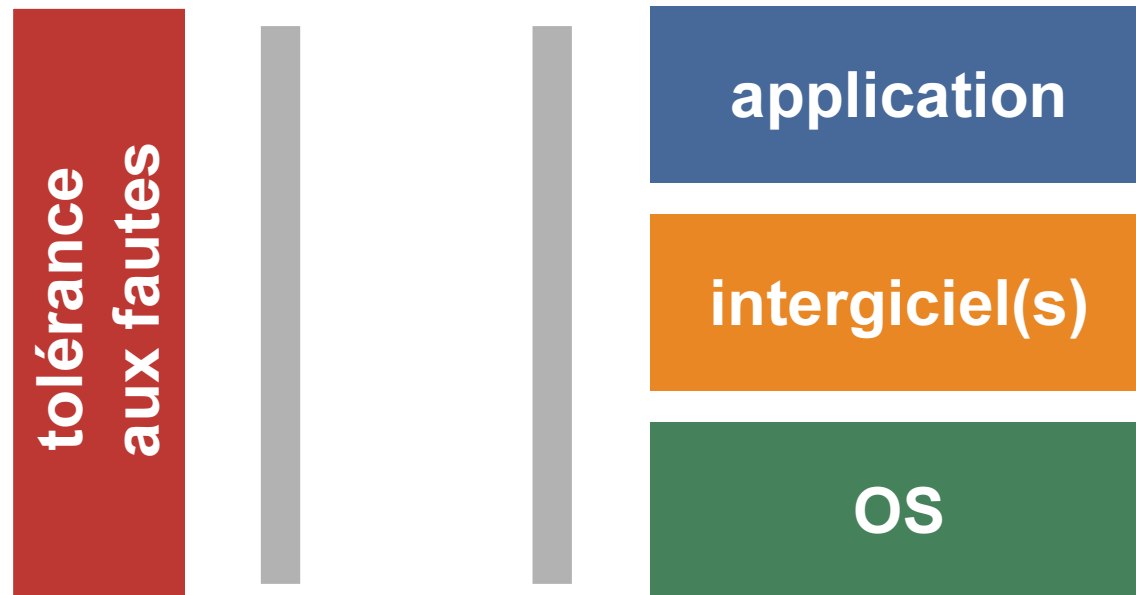
Utilisation des liens inter-niveaux

Pour réaliser une empreinte réflexive la compréhension des **liens entre les niveaux** (transcendance) permet de **combiner l'intelligence** des hauts niveaux avec **la force brute** des bas niveaux.

- Utilisation descendante
 - capture d'état
 - maîtrise du non-déterminisme
- Utilisation ascendante
 - analyse des propagations d'erreurs
 - recouvrement par reprise avant

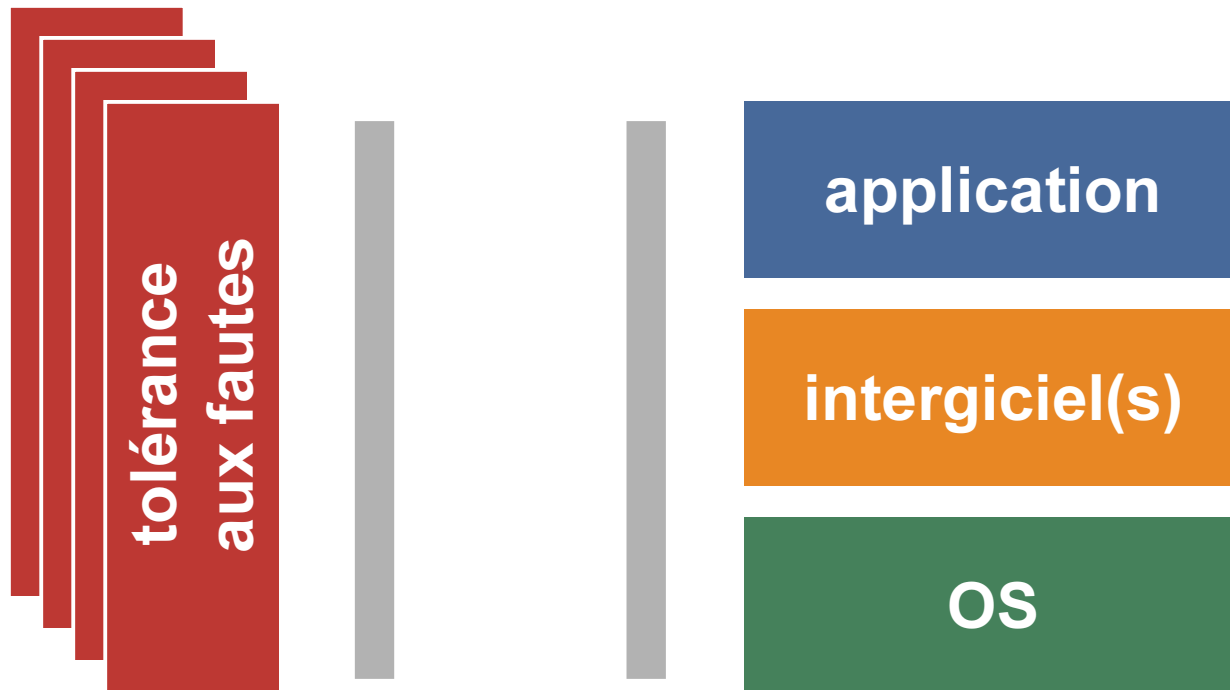


La démarche

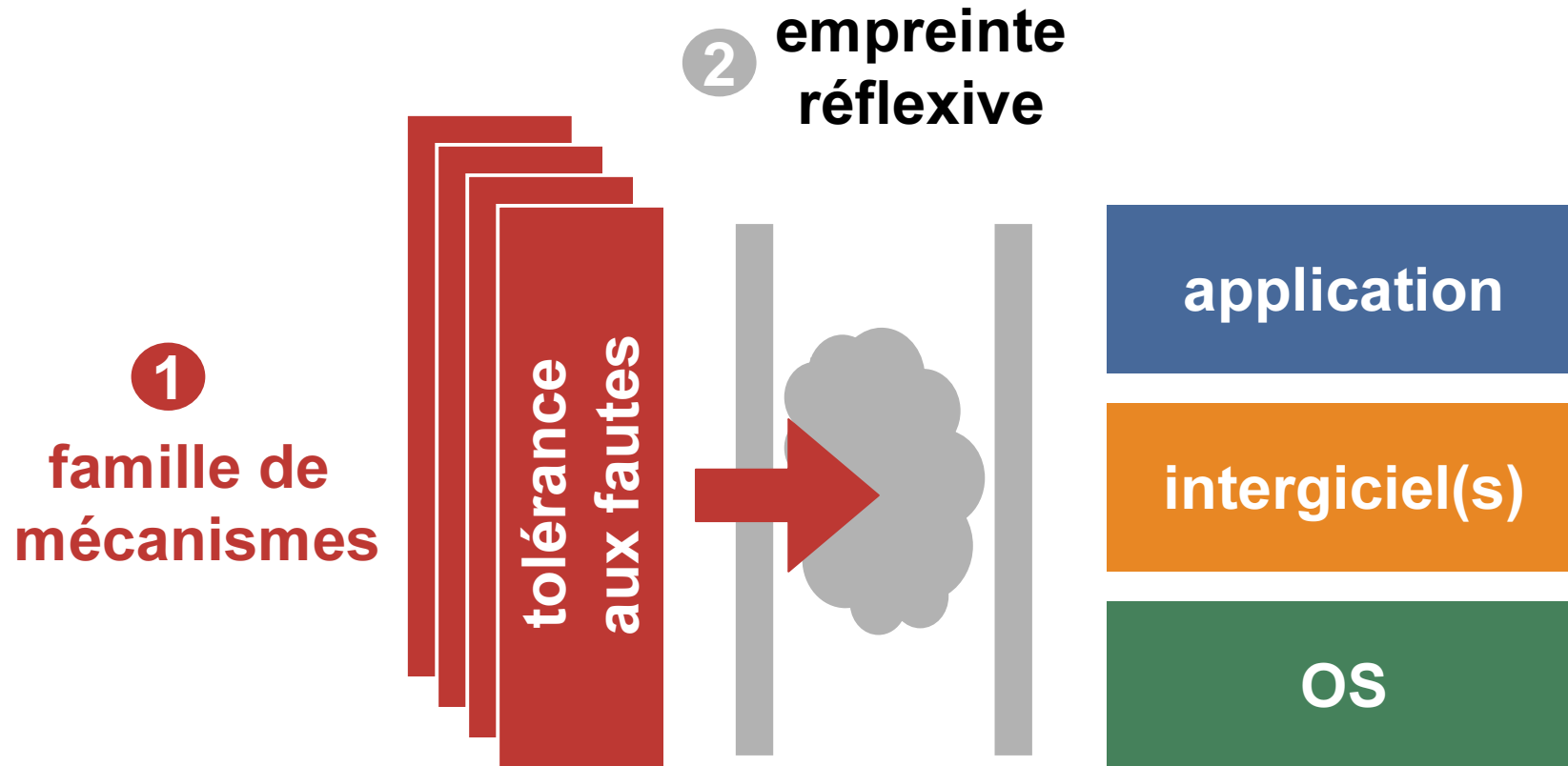


La démarche

1
famille de
mécanismes



La démarche

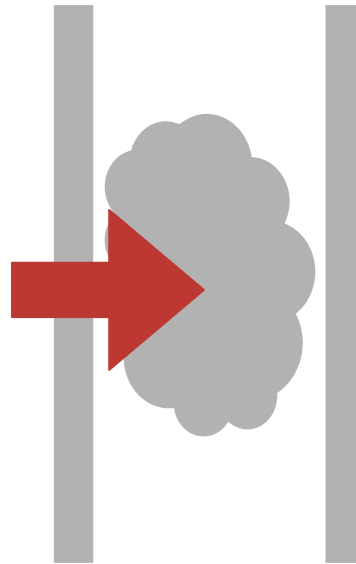


La démarche

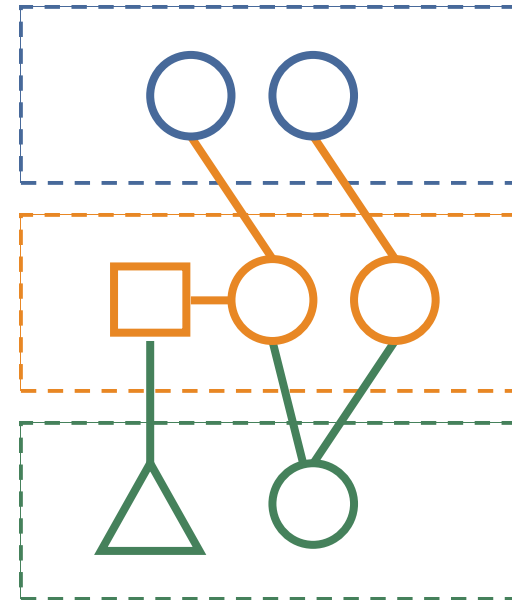
1
famille de
mécanismes



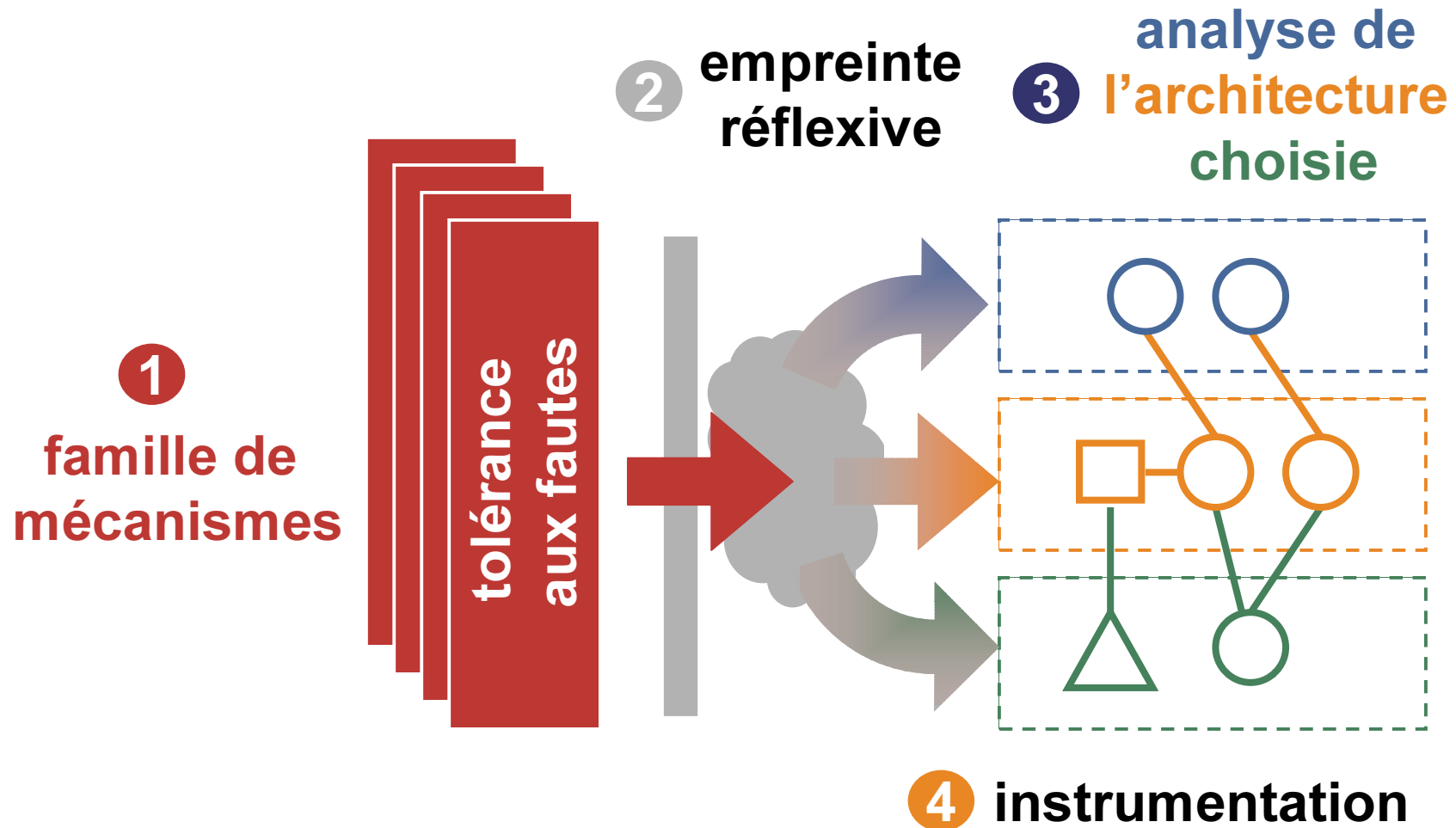
2 empreinte
réflexive



3 analyse de
l'architecture
choisie



La démarche



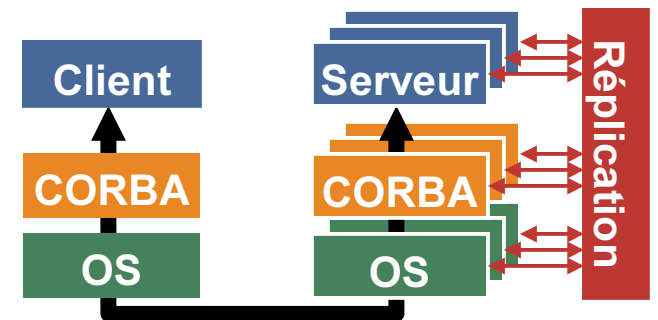
Plan

- (A) Qu'est-ce-que la réflexivité ?
- (B) Point de vue algorithmique :
Besoins réflexifs de la tolérance aux fautes
- (C) Point de vue architectural :
La réflexivité dans les systèmes complexes

- (D) Application à un exemple concret :
**Réplication multi-traitement d'une plate-forme
CORBA / Linux**

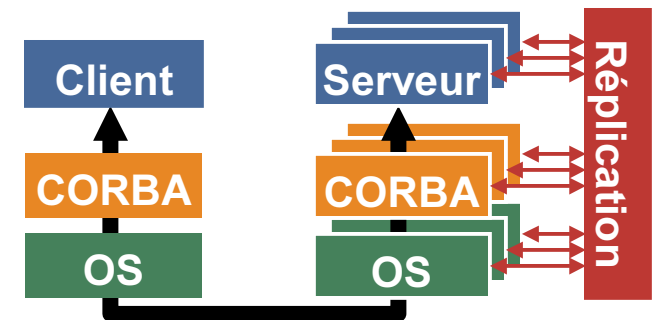
Exemple concret : réplication & multitraitement

- Objectif : réplication transparente d'une application CORBA
 - Multi-niveaux : **POSIX** (Linux + biblio.) + **CORBA** (ORBacus)
 - Multitraitement : traitement concurrent des requêtes
 - À réserve de brins : limitation de la concurrence



Exemple concret : réplication & multitraitement

- Objectif : réplication transparente d'une application CORBA
 - Multi-niveaux : **POSIX** (Linux + biblio.) + **CORBA** (ORBacus)
 - Multitraitement : traitement concurrent des requêtes
 - À réserve de brins : limitation de la concurrence



⇒ Mise en pratique des 4 étapes de la démarche





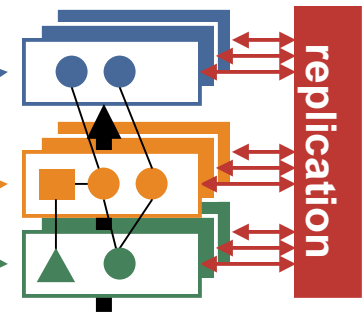
Exemple concret : réplication & multitraitements

- Objectif : réplication transparente d'une application CORBA
 - Multi-niveaux : **POSIX** (Linux + biblio.) + **CORBA** (ORBacus)
 - Multitraitements : traitement concurrent des requêtes
 - À réserve de brins : limitation de la concurrence

- *Problème 1: capture & restauration de l'état*

- État applicatif

- État de l'intergiciel + et de l'OS

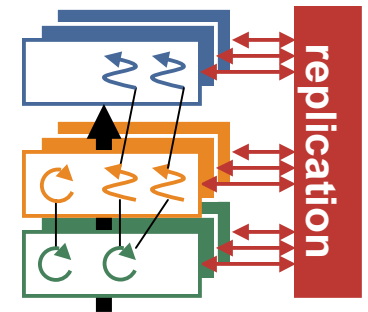




Exemple concret : réplication & multitraitement

- Objectif : réplication transparente d'une application CORBA
 - Multi-niveaux : **POSIX** (Linux + biblio.) + **CORBA** (ORBacus)
 - Multitraitement : traitement concurrent des requêtes
 - À réserve de brins : limitation de la concurrence

- *Problème 1*: capture & restauration de l'état
 - État applicatif
 - État de l'intergiciel + et de l'OS



- *Problème 2*: maîtrise du non-déterminisme
 - Hypothèse: multitraitement seule source de non-déterminisme
 - Comment **répliquer** les **décisions non-déterministes** ?



Empreinte de la réplication

Facettes réflexives

	<i>Communication</i>	<i>Exécution</i>	<i>État</i>
Capacités d'observation	RequestReception RequestSending ReplySending ReplyReception getRequestContent getReplyContent	ExecutionPointStart ExecutionPointEnd ExecutionPointReach NonDeterministicFlowChange getExecutionPoint	NonDeterministicPlatformCall getServerState getPlatformState
Capacités d'action	doSend doReceive piggyBackDataOnMsg	createExecutionPoint setExecutionPoint forceResultOfFlowChange	forceResultOfPlatformCall setServerState setPlatformSate



Empreinte de la réplication

Facettes réflexives

	<i>Communication</i>	<i>Exécution</i>	<i>État</i>
Capacités d'observation	RequestReception RequestSending ReplySending ReplyReception getRequestContent getReplyContent	ExecutionPointStart ExecutionPointEnd ExecutionPointReach NonDeterministicFlowChange getExecutionPoint	NonDeterministicPlatformCall getServerState getPlatformState
Capacités d'action	doSend doReceive piggyBackDataOnMsg	createExecutionPoint setExecutionPoint forceResultOfFlowChange	forceResultOfPlatformCall setServerState setPlatformSate

Concrétisation de l'empreinte

- Tous les éléments d'une architecture n'ont pas la même « permanence ».
 - Les interfaces (CORBA, POSIX) restent.
 - Les implémentations (ORBacus, GNU/Linux) changent.
- **Concrétisation** de l'empreinte réflexive en **2 temps** :



Analyse : modélisation des liens **CORBA** ↔ **POSIX**

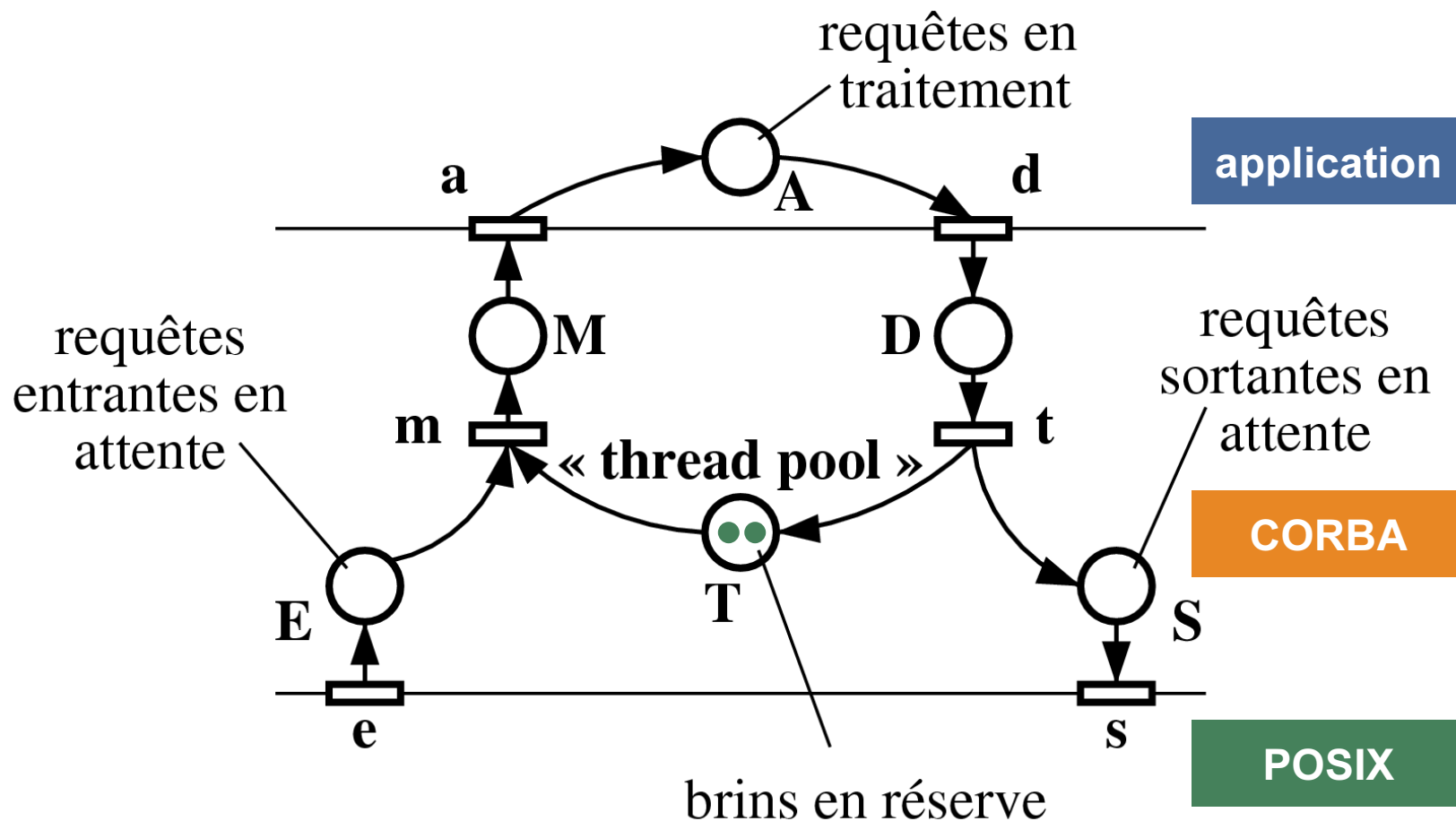
- En factorisant notre connaissance de plusieurs ORB
- Indépendamment de toute implémentation particulière
- Détermine où réaliser les capacités réflexives de l'empreinte



Instrumentation concrète sur **ORBacus** + **GNU/Linux**

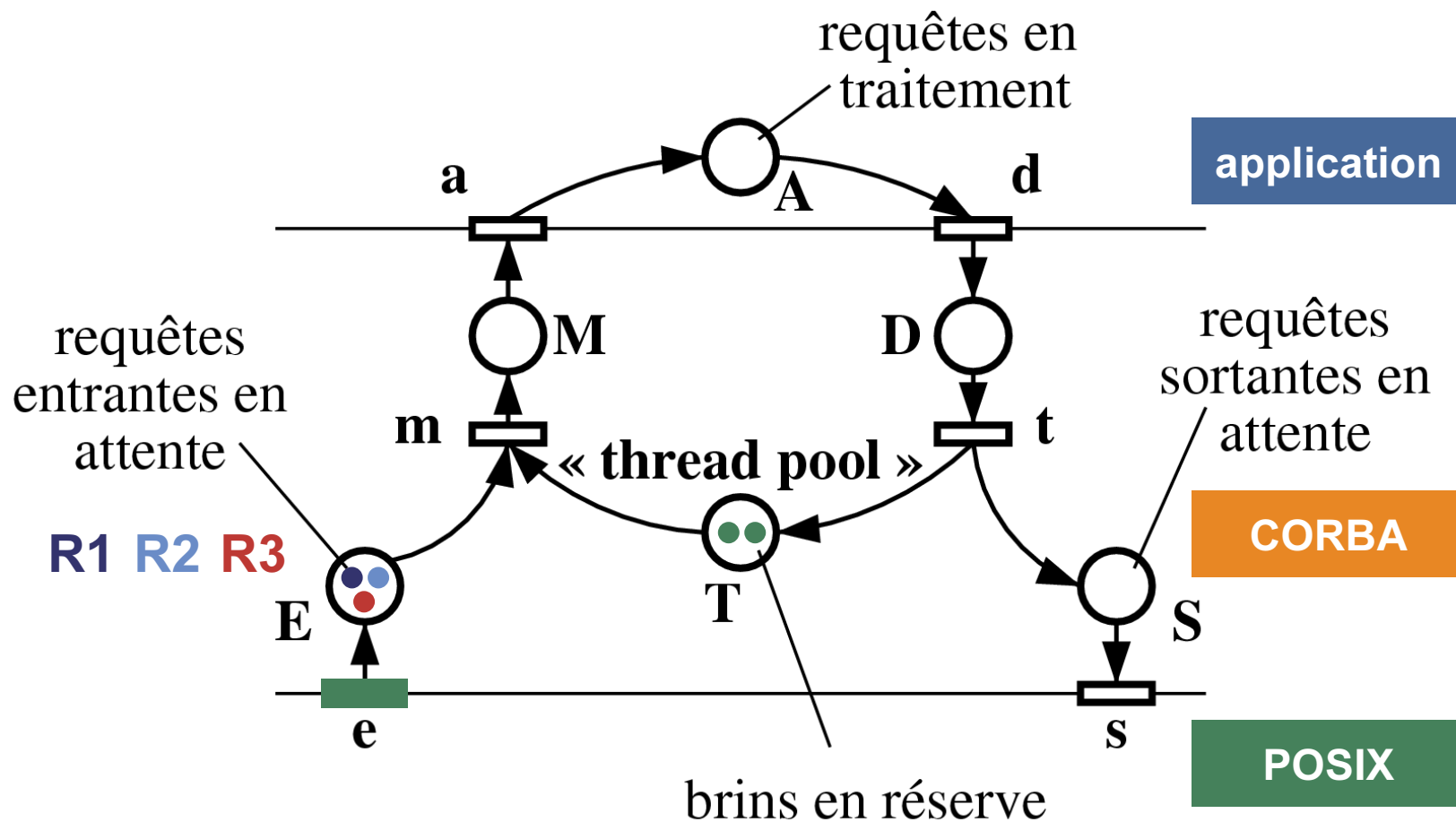


Modélisation des liens POSIX \leftrightarrow CORBA



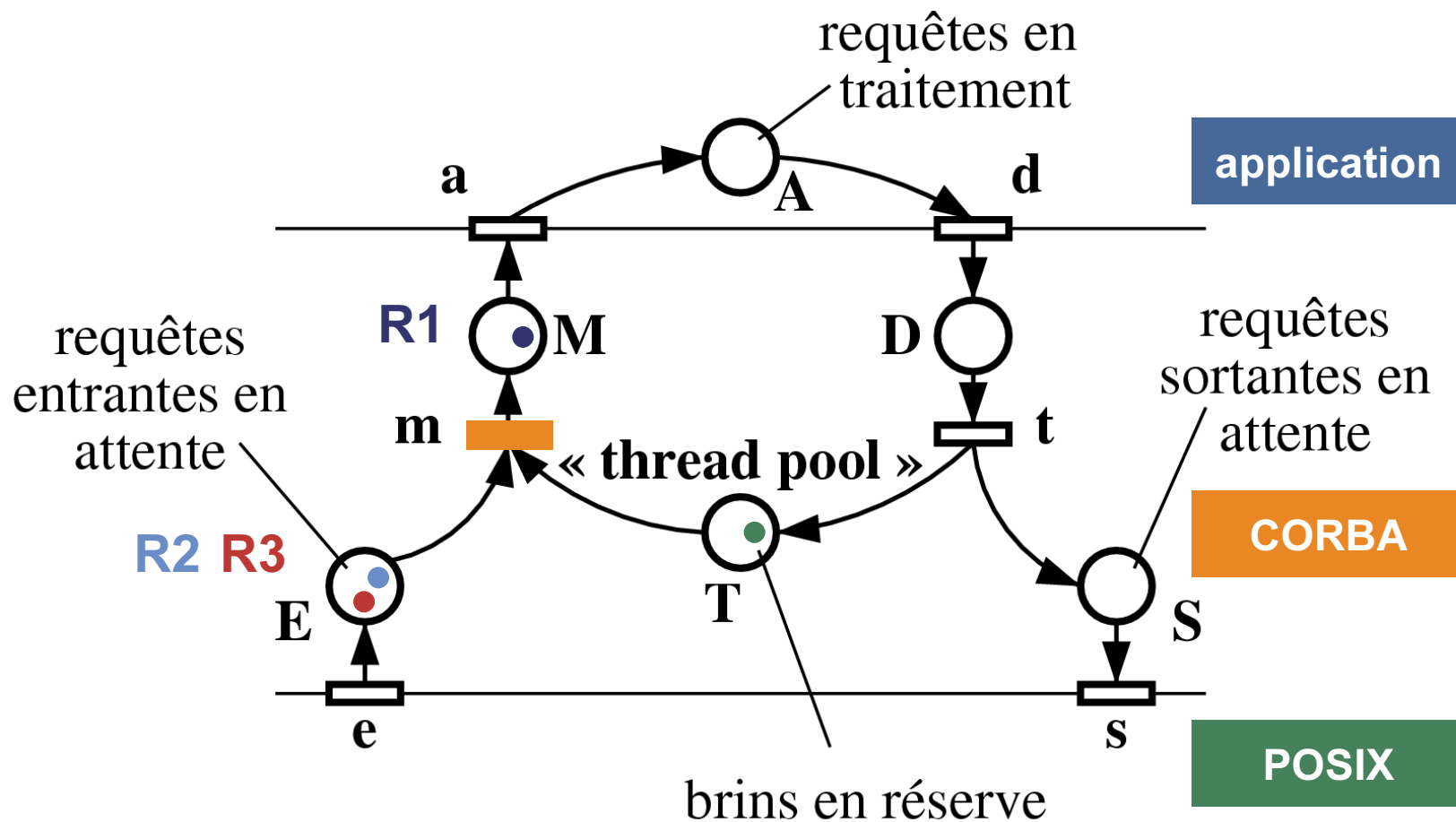


Modélisation des liens POSIX ↔ CORBA



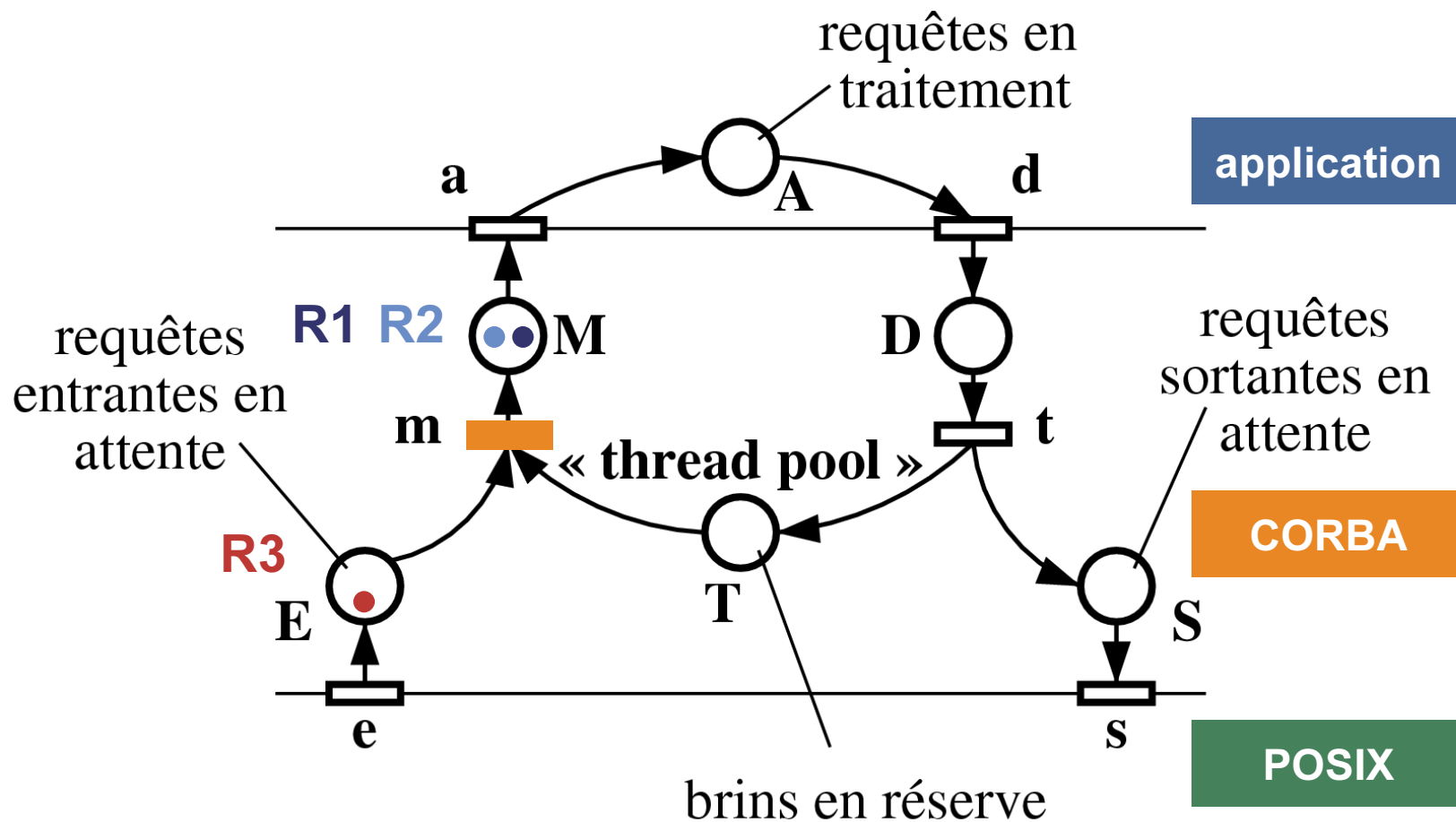


Modélisation des liens POSIX ↔ CORBA



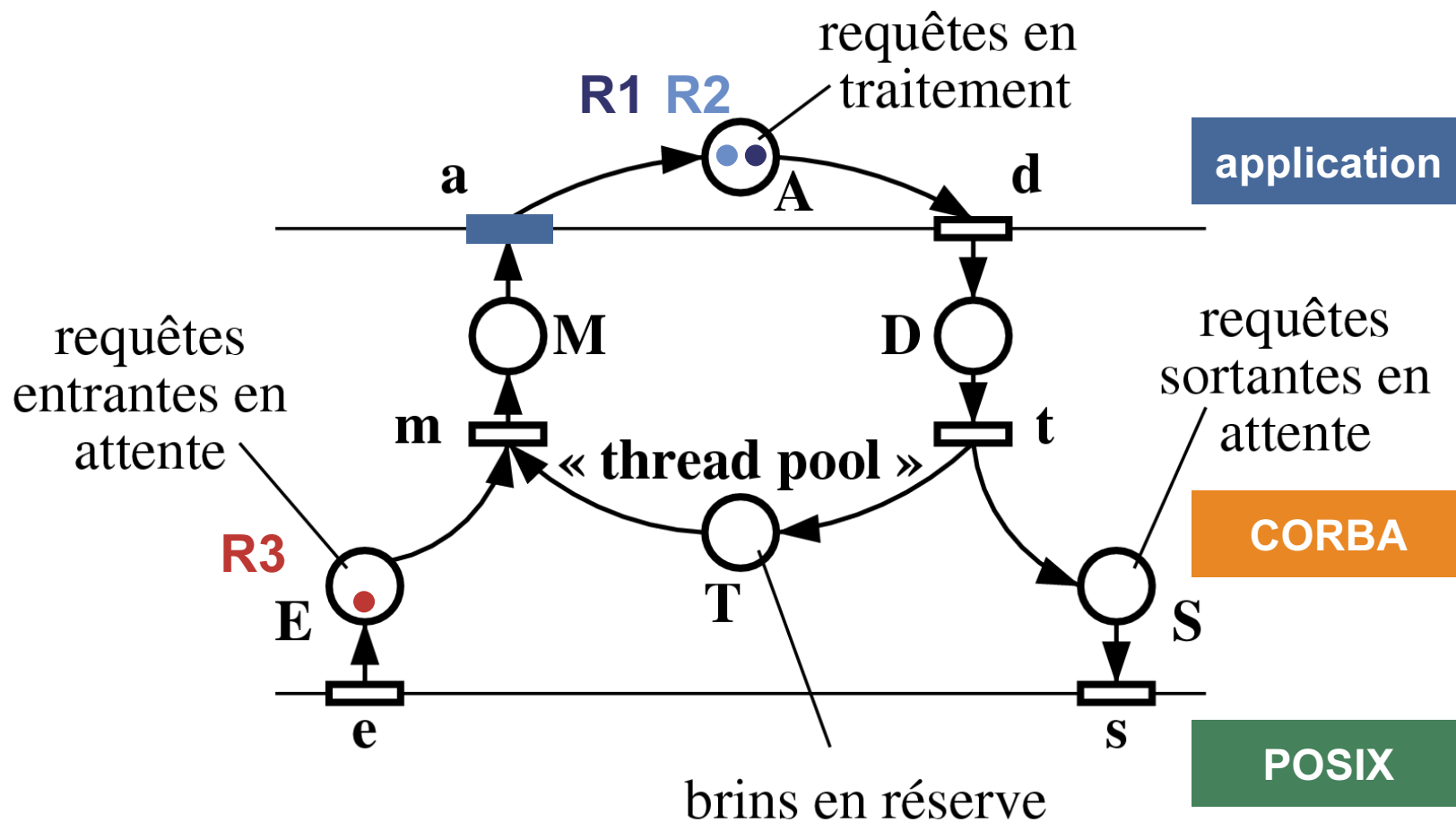


Modélisation des liens POSIX \leftrightarrow CORBA



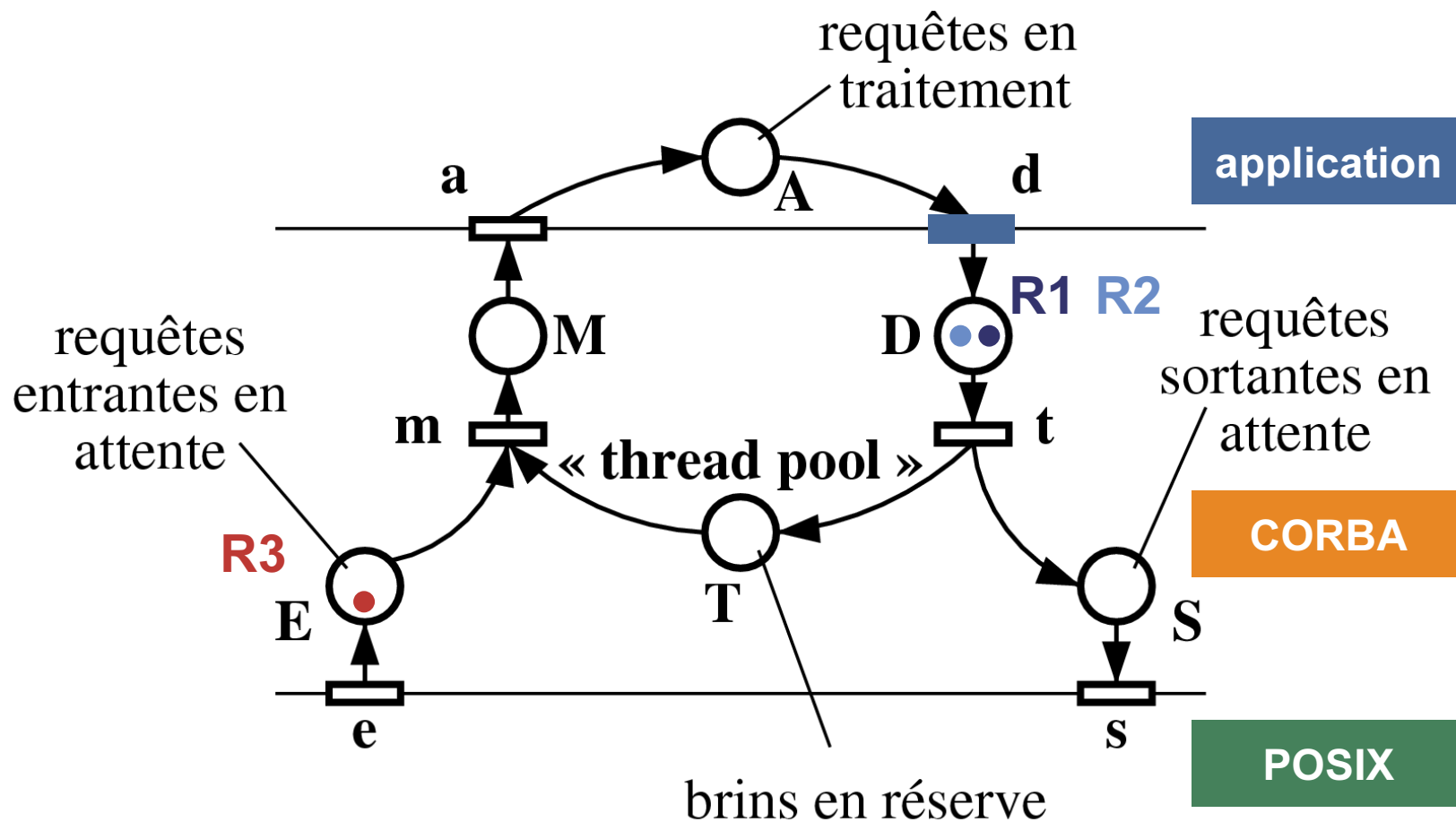


Modélisation des liens POSIX \leftrightarrow CORBA



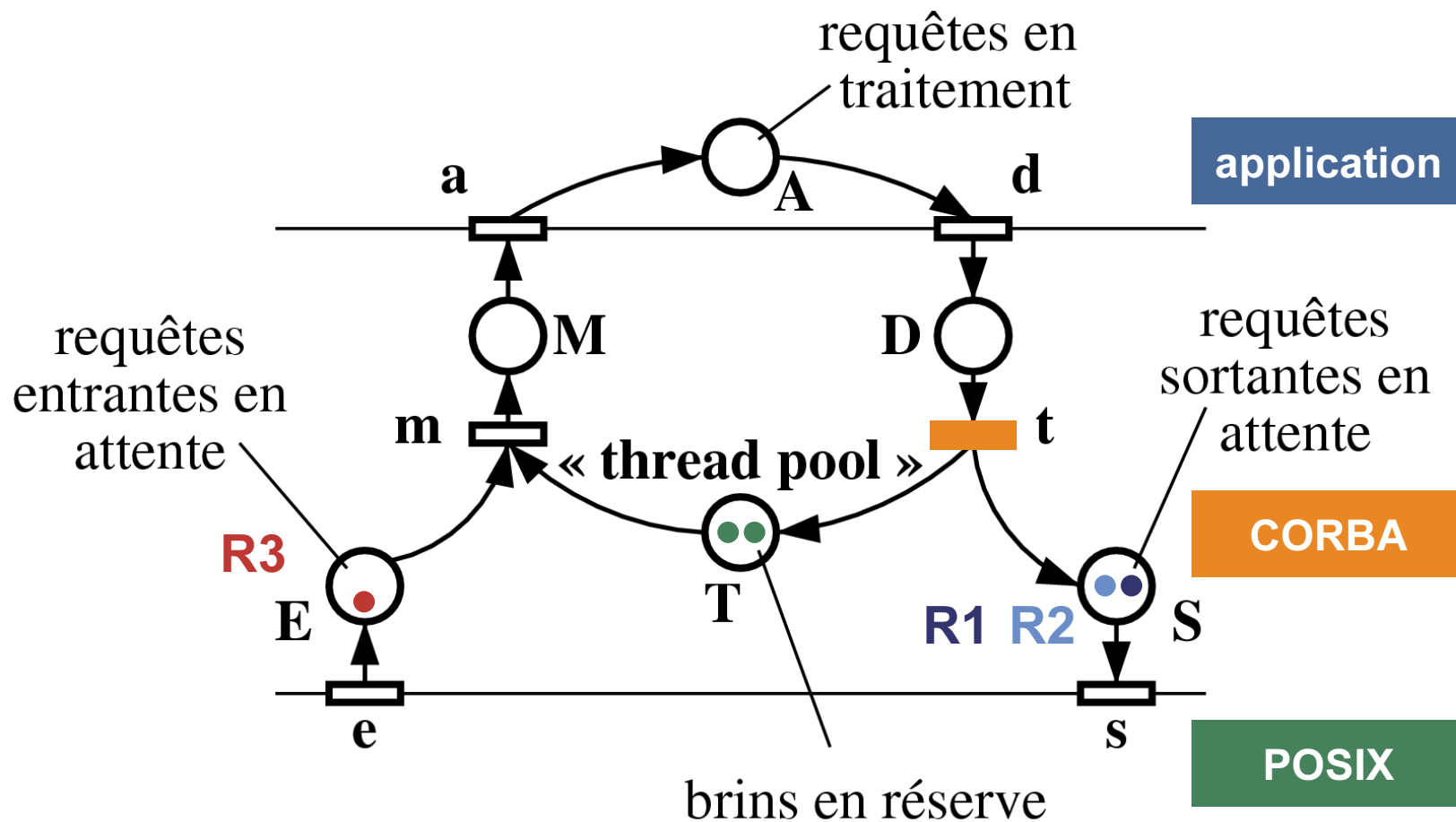


Modélisation des liens POSIX ↔ CORBA



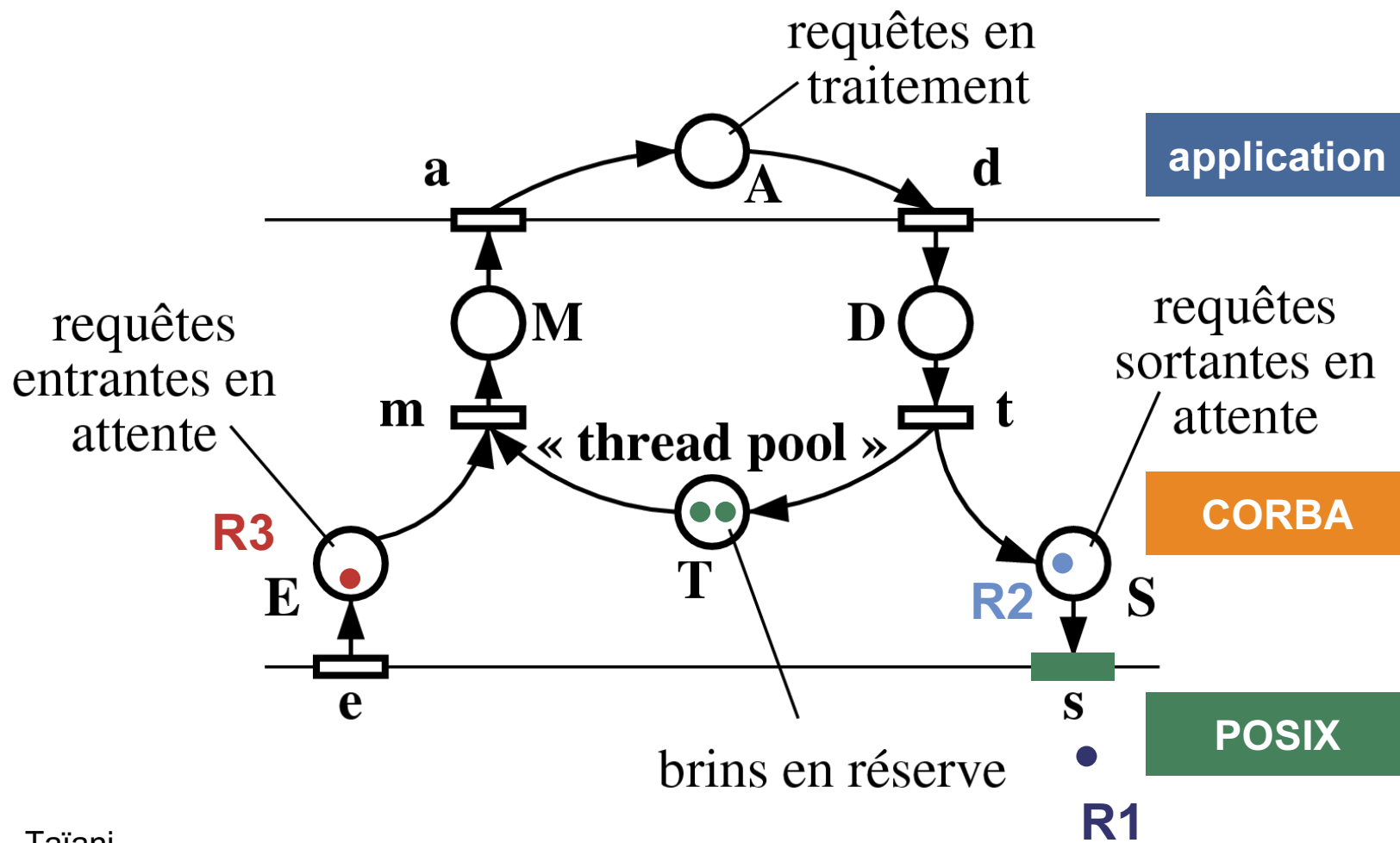


Modélisation des liens POSIX ↔ CORBA



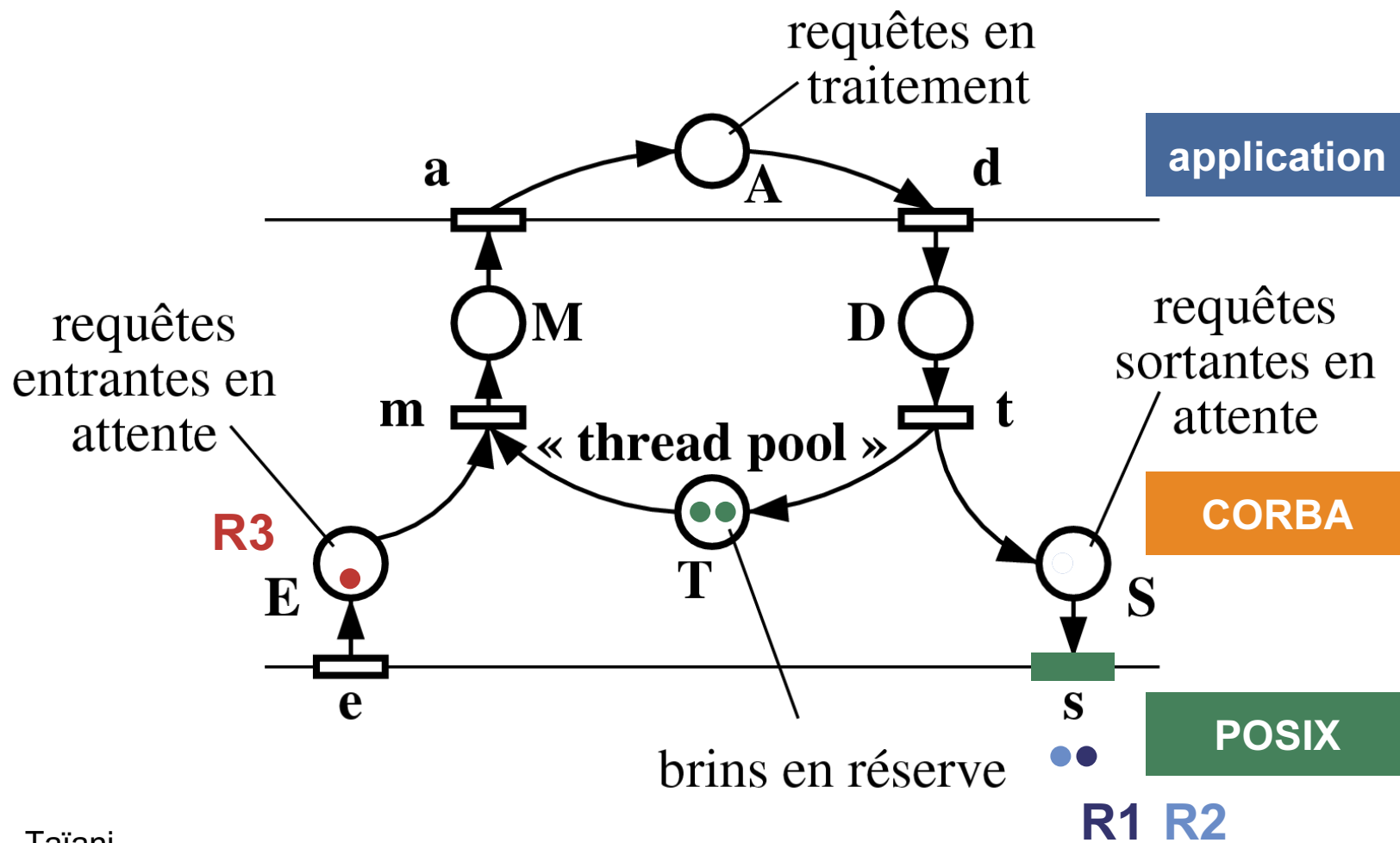


Modélisation des liens POSIX ↔ CORBA



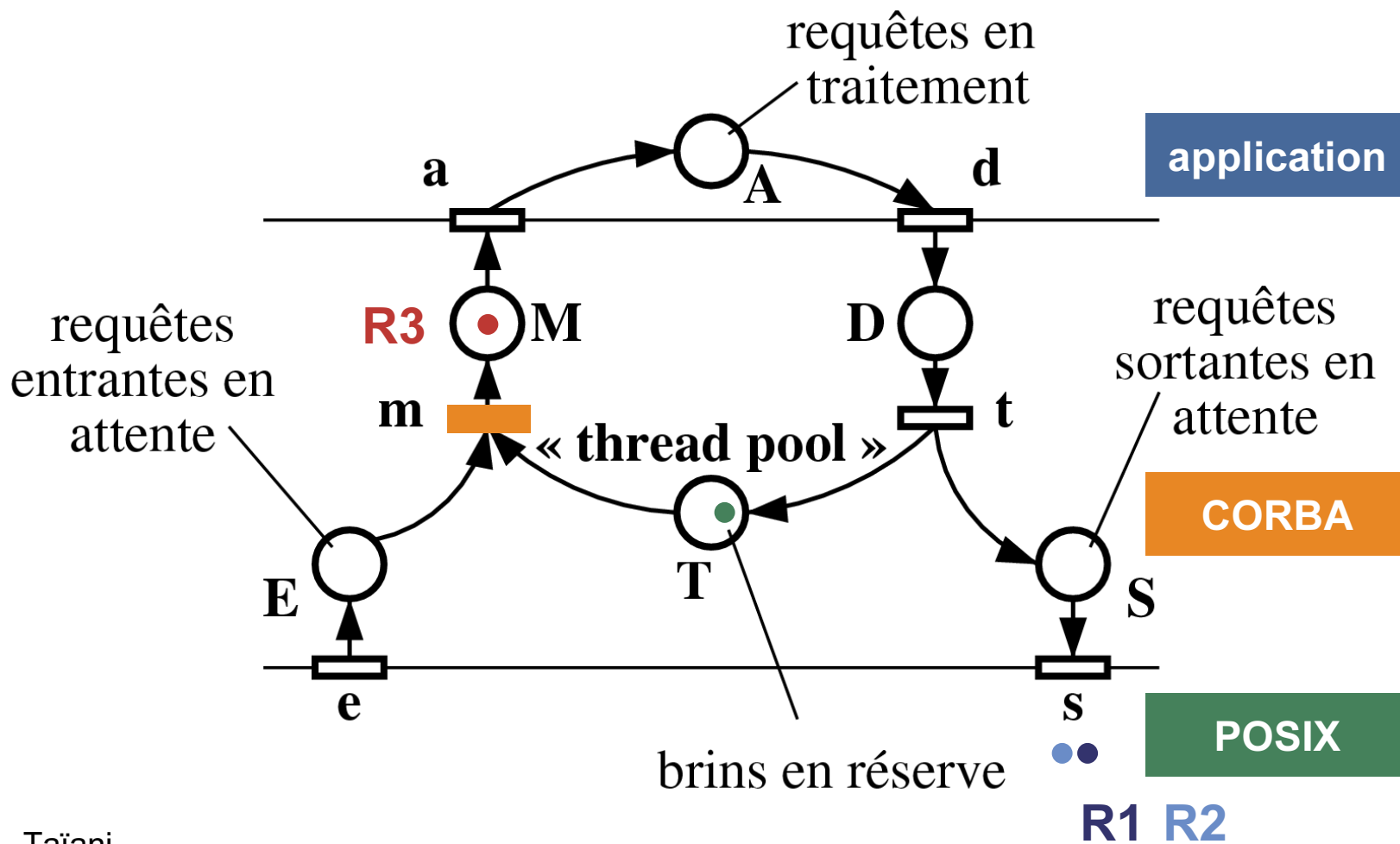


Modélisation des liens POSIX ↔ CORBA





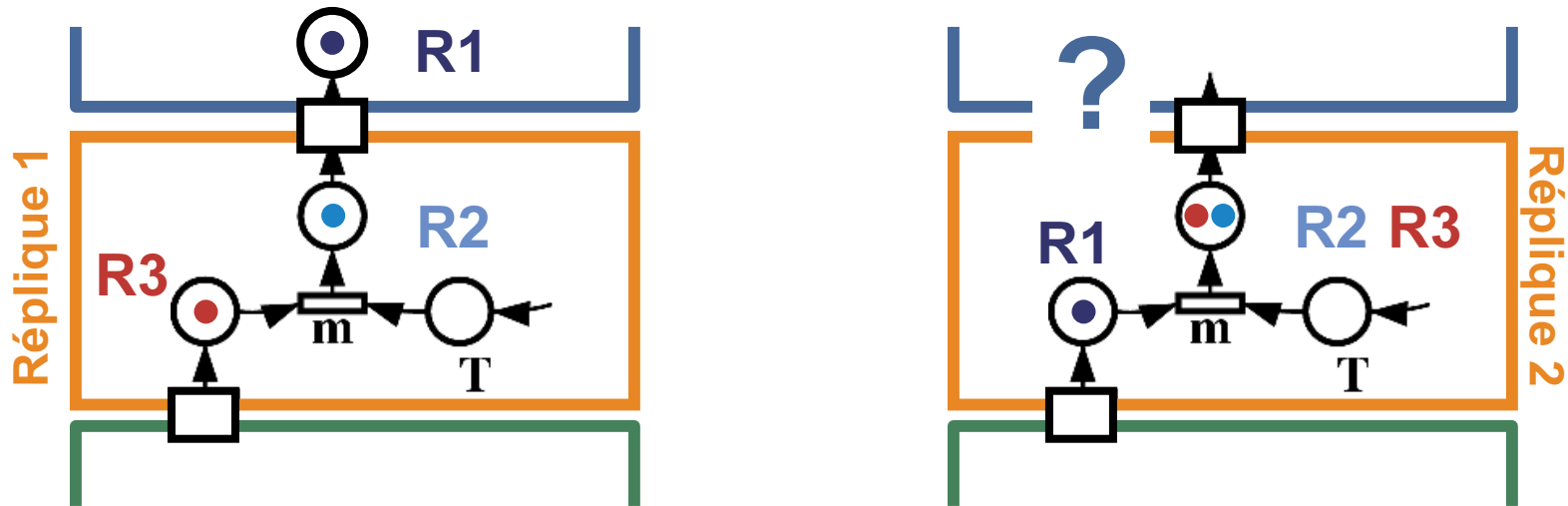
Modélisation des liens POSIX \leftrightarrow CORBA





Réplication du déterminisme : Niveau applicatif seul

- L'ordonnancement des requêtes par l'intergiciel est **arbitraire**.





Réplication du déterminisme : Niveau applicatif seul

- L'**ordonnancement** des requêtes par l'intergiciel est **arbitraire**.

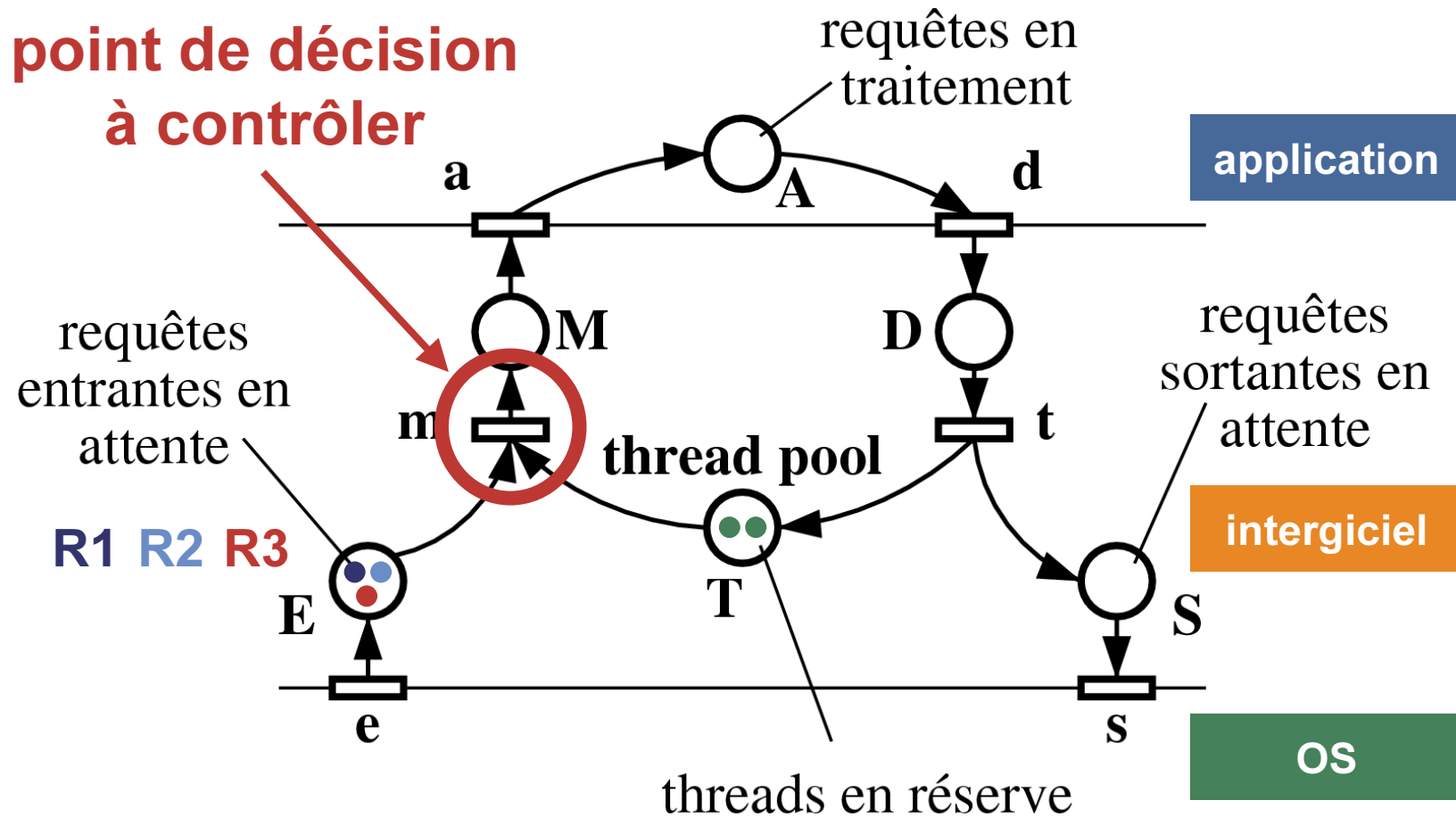


- La réplication de l'ordonnancement observé au **niveau applicatif** aboutit à un **blocage**
 - dû à la réserve de brins (ici de taille 2).
 - Les décisions d'ordonnancement internes à l'intergiciel ne peuvent être contrôlées depuis l'application.

Appli
?
?



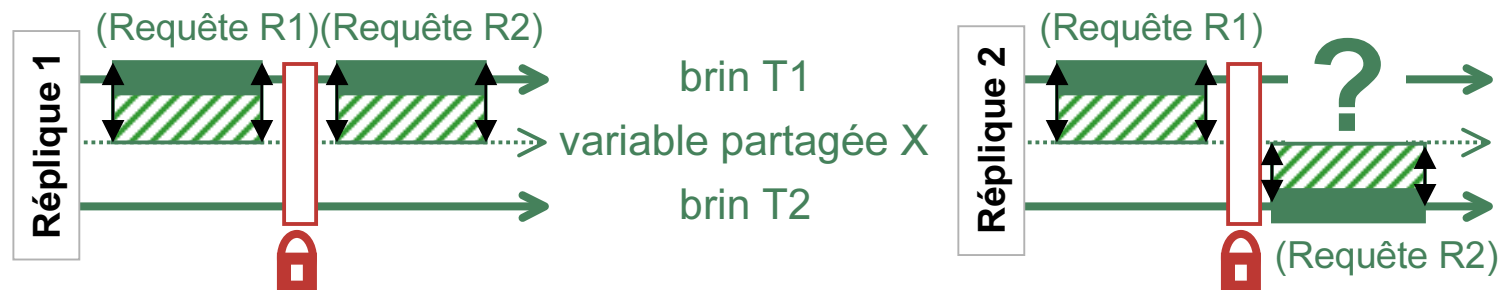
Identification du problème





Niveau OS seul

- L'allocation des verrous de l'OS peut être contrôlée :
 - Même ordonnancement des brins sur toutes les répliques
 - Les requêtes sont transmises / traitées dans le même ordre.
 - 😊 Toutes les répliques ont le même état : **cohérence des répliques** (hypothèse: MT = seule source de non-déterminisme)
- ☹️ Mais cette solution **sur-contraint** l'exécution des répliques :
 - Impossible de relier l'activité de l'OS à celle de l'intergiciel
 - Nécessité de répliquer **toutes** les décisions sur les verrous



non-équivalent ➡ **réplication de toutes les décisions**



Réplication « intelligente »

- Utilisation des sémantiques **intergicielle** et **applicative** :
 - ➔ La réification de l'activité de **l'intergiciel** de de **l'application** donne sens à l'activité du **système d'exploitation**.
 - ➔ Utilisation **optimisée** des capacités réflexives de l'OS
- Exemple: réserve de brins
 - ➔ Savoir quel brin exécute quelle requête n'a pas d'importance.
 - ➔ Les deux exécutions suivantes sont équivalentes :

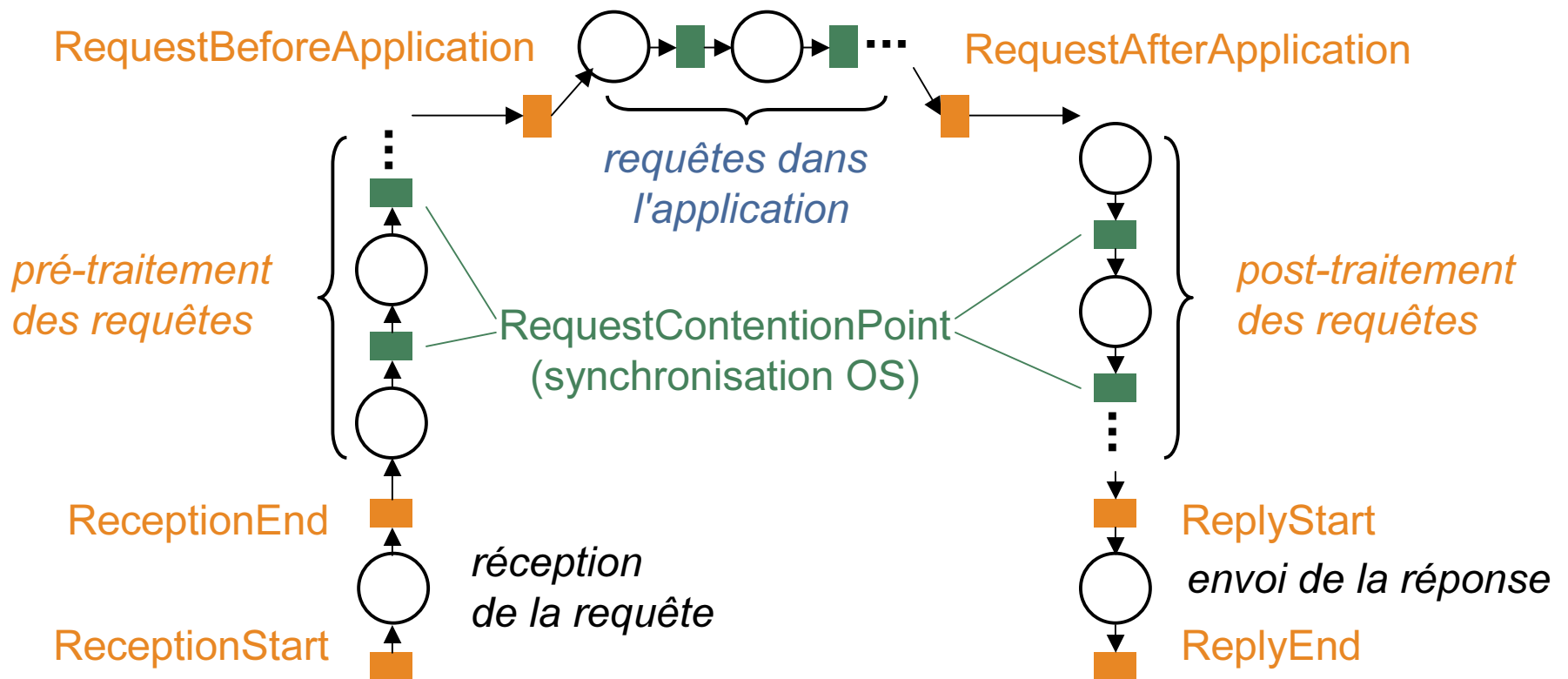
Cet ordonnancement n'a pas besoin d'être répliqué.





Le méta-modèle obtenu

- Structuré autour du **cycle de vie** d'une requête CORBA
 - ➔ Agrège les synchronisations OS au cycle de vie des requêtes



Appli

CORBA

OS



La méta-interface

```
class Request ;
class Thread ;
class StackChunk ;
class ReifiedEvent ;
class RequestLifeCycleEvent extends ReifiedEvent {
    public Request reifiedRequest ;
    public Thread reifyingThread ;
}
class BeginOfRequestReception extends RequestLifeCycleEvent ;
class EndOfRequestReception extends RequestLifeCycleEvent ;
class RequestBeforeApplication extends RequestLifeCycleEvent ;
class RequestAfterApplication extends RequestLifeCycleEvent ;
class BeginOfRequestResultSend extends RequestLifeCycleEvent ;
class EndOfRequestResultSend extends RequestLifeCycleEvent ;
class RequestContentionPoints extends RequestLifeCycleEvent ;

class IntercessionCommand ;
class ContinueExecution extends IntercessionCommand ;
class SkipCallToApplication extends IntercessionCommand ;

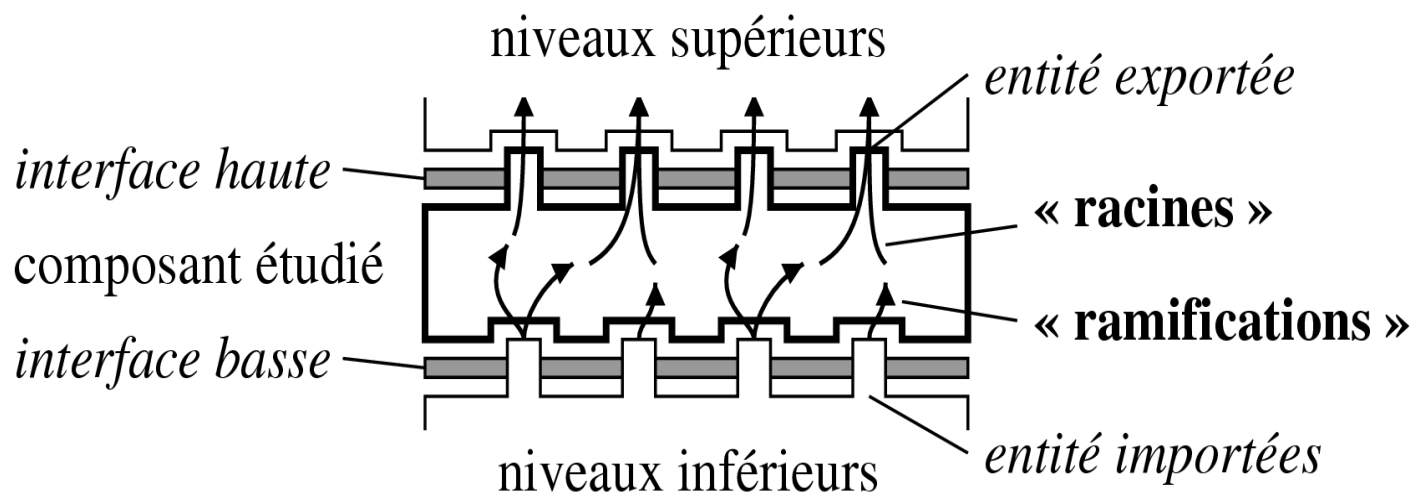
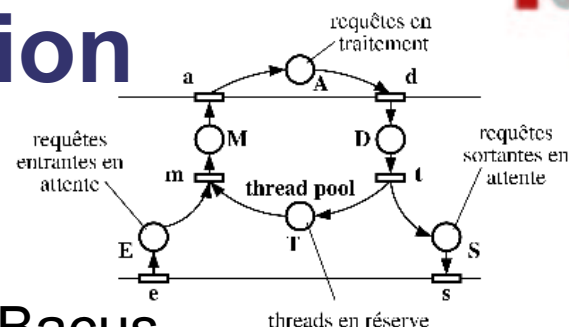
interface MetaLevel {
    IntercessionCommand reifyEventToMetaSynchronous (ReifiedEvent e);
}
interface BaseLevel {
    State captureApplicationState ();
    void restoreApplicationState (State s);
    StackChunk captureApplicationStack (Thread t);
    void restoreApplicationStack (Thread t, StackChunk stack) ;
    void InjectRequestAtCommuncationLevel (Request r);
}
```

Taïani



Instrumentation

- Le modèle d'analyse est générique.
- Instrumentation sur GNU/Linux – ORBacus
 - Requiert de relier les **composants concrets** au **modèle générique**.
 - **Rétro-conception** complexe : ORBacus = 110 000 LdCode
 - Activité **d'abstraction** (outil spécialisé, **CosmOpen**, cf. Annexe B)
 - Ciblage des interfaces : « racines » / « ramifications »





Notre appareil instrumental

- **CosmOpen** : Un outils d'analyse semi-automatique
 - Nous l'avons développé spécifiquement pour notre thèse.
 - Il est basé sur des opérations de **manipulation de graphes**.
 - Il permet des analyses **structurelles & comportementales**.
 - Il gère des graphes de **très grandes tailles**.
 - Trace ORBacus : 2066 appels enregistrés ⇒ **2066 nœuds**
 - **Distribué librement** : <http://www.laas.fr/~ftaiani/7-software>
- Composant de capture :
 - Capture structurelle : 4280 lignes C++ (avec Doxygen)
 - Capture comportementale : 1660 lignes C++ (avec gdb)
- Composant d'analyse : 17010 lignes en Java
- **CosmOpen** **22950** lignes de code

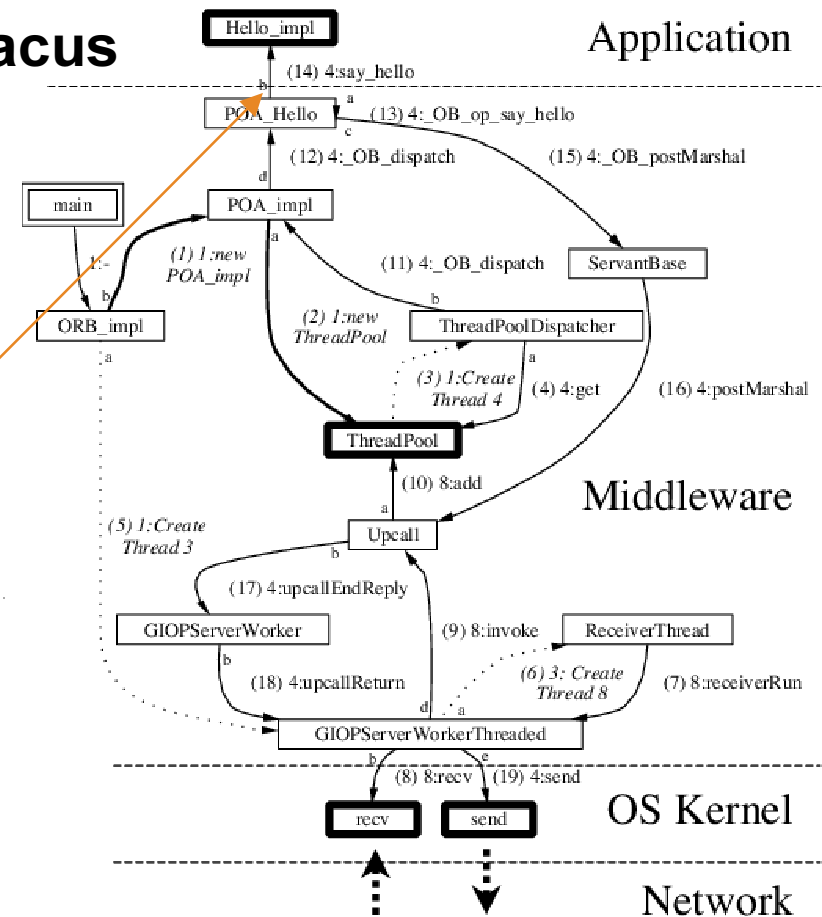


Instrumentation

■ Modèle comportemental d'ORBacus

- fourni par **CosmOpen**
- donne du sens aux actions de l'OS pour l'application et l'ORB
- permet d'identifier les points d'instrumentation.

RequestBeforeApplication





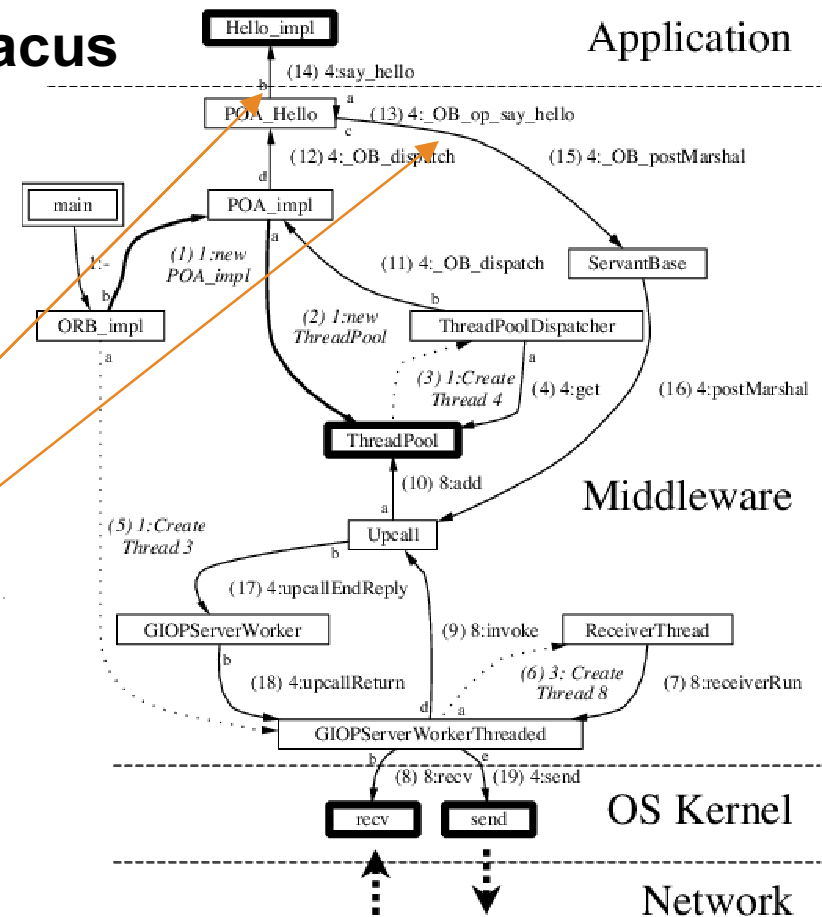
Instrumentation

■ Modèle comportemental d'ORBacus

- fourni par **CosmOpen**
- donne du sens aux actions de l'OS pour l'application et l'ORB
- permet d'identifier les points d'instrumentation.

RequestBeforeApplication

RequestAfterApplication





Instrumentation

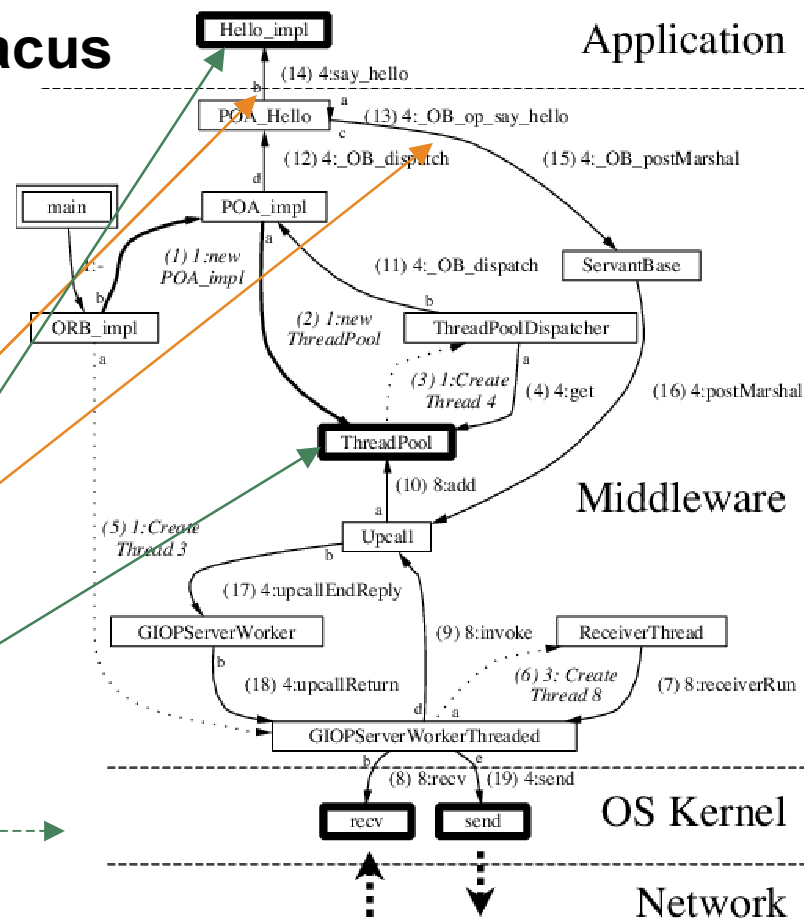
■ Modèle comportemental d'ORBacus

- fourni par **CosmOpen**
- donne du sens aux actions de l'OS pour l'application et l'ORB
- permet d'identifier les points d'instrumentation.

RequestBeforeApplication

RequestAfterApplication

RequestContentionPoint





Instrumentation

- Bibliothèque OO **générique** d'interception OS
 - 6590 lignes en C++
 - classes pour intercepter verrous et connexions
 - MetaMutex, MetaSocket
 - supporte la transcendance par « talonnage » des brins
 - MetaThreadInfo, ThreadMetaMutex, ThreadMetaSocket

- Bibliothèque OO **générique** d'interception **multi-niveaux**
 - 1460 lignes en C++
 - utilise l'interception OS pour réaliser le méta-modèle
 - RequestContentionPoint, MetaRequestLifeCycle

- **Instrumentation** du **code original** d'ORBacus
 - Extrêmement peu intrusif : 20 lignes ajoutées
 - **0,02 % du code original**

Ce que nous a appris l'exemple

- La méta-interface obtenue est **cohérente** et **homogène**.
 - Elle gère le **non-déterminisme** et la **capture d'état**.
- Nous avons implanté la partie sur le non-déterminisme.
- **Efficace** : par exemple réplique de la **synchronisation**.
Durant le traitement d'une requête dans ORBacus :
 - **203** opérations de synchronisation sont observées.
 - Seules **3** sont interceptées dans l'ORB (**gain : x 67**).
 - Notre analyse assure que ces 3 interceptions sont **suffisantes** pour assurer la **cohérences** de l'ORB.
- **Très peu intrusif** : 20 lignes modifiées sur 110 000
- **Réutilisable** : **outil** CosmOpen, **bibliothèques** génériques

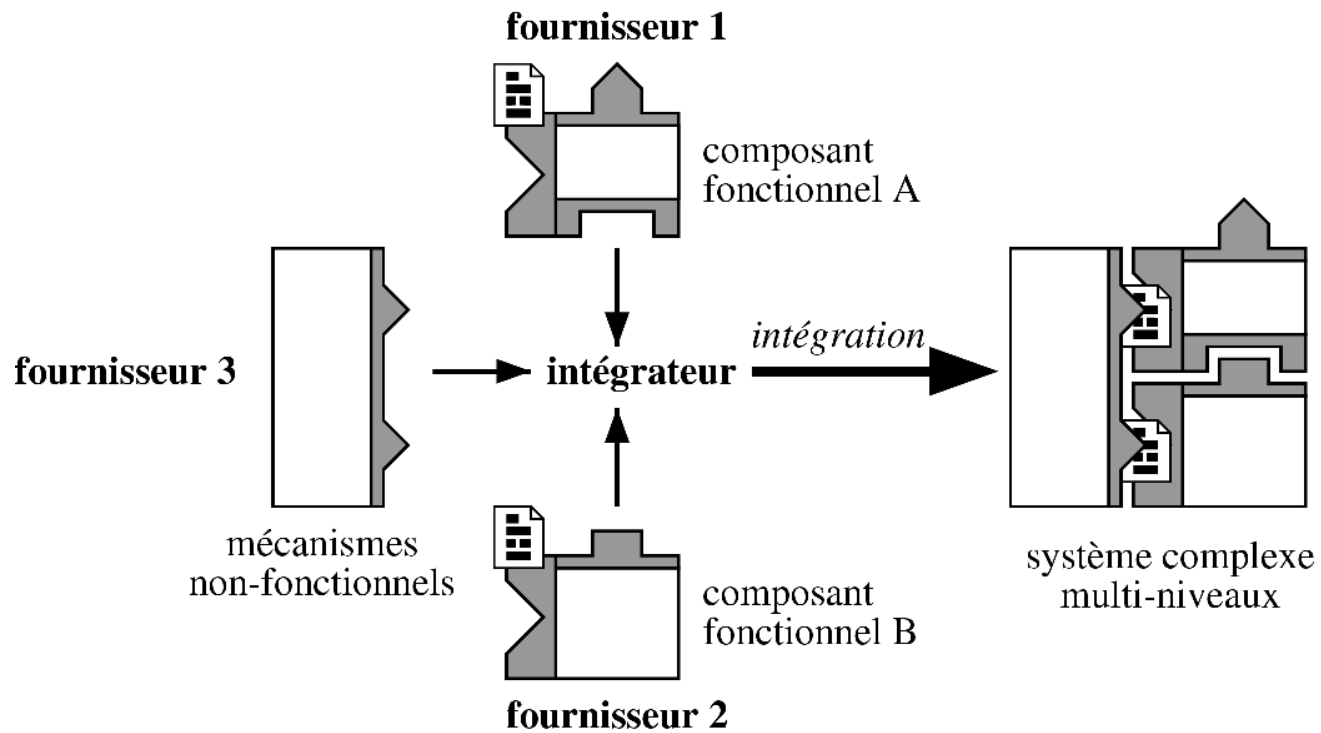
Conclusions

- La réalisation de la **tolérance aux fautes transversale** rentre en **conflit** avec les natures multi-composants & multi-couches des **systèmes complexes** utilisés aujourd'hui.
- Notre proposition pour lever ce conflit :
La réflexivité multi-niveaux :
 - Combine des capacités d'action et d'observation des **hautes** et **basses** couches en une vision **globale** du système
- Nous avons **validé** cette proposition dans la **pratique**.
 - **Analyse** de composants industriels (GNU/Linux + Orbacus) par un outil de rétro-conception (**CosmOpen**)
 - **Implantation** d'une méta-interface pour la réplication sur une architecture POSIX / CORBA

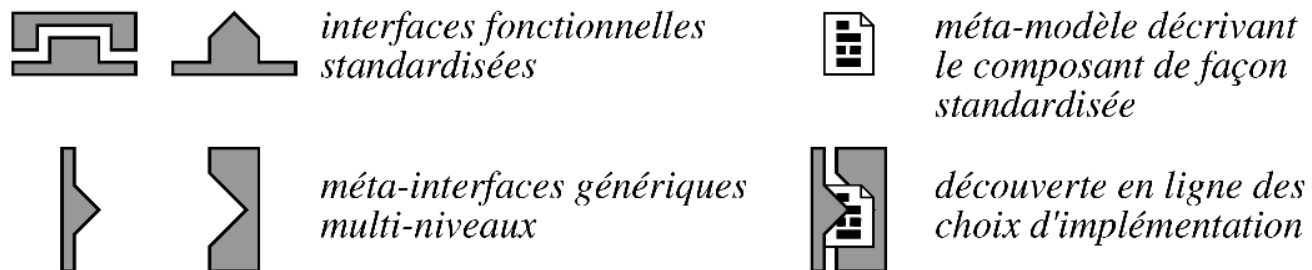
Perspectives

- Cette thèse est un premier pas.
- Pour aller plus loin : **nouveau modèle à composants**
- Renoncer à la stricte séparation interface / implémentation
- Ouverture partielle & maîtrisée par exportation d'un **méta-modèle** dans un **format normalisé**
- **Déjà réalité** pour certaines applications :
 - **microélectronique** : synthèse modulaire de circuits intégrés
 - **sécurité informatique** : codes mobiles porteurs de preuve

Notre vision



Légende :



Les algorithmes de TaF

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Leftrightarrow déduction pessimiste

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Rightarrow déduction pessimiste
 - L'algorithme est peu efficace, mais reste correct.

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Leftrightarrow déduction pessimiste
 - L'algorithme est peu efficace, mais reste correct.

- Impact pour le praticien

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Leftrightarrow déduction pessimiste
 - L'algorithme est peu efficace, mais reste correct.
- Impact pour le praticien
 - Des capacités d'observation bon marché peuvent être utilisées.

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Leftrightarrow déduction pessimiste
 - L'algorithme est peu efficace, mais reste correct.

- Impact pour le praticien
 - Des capacités d'observation bon marché peuvent être utilisées.
 - Mais les performances sont mauvaises.

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Leftrightarrow déduction pessimiste
 - L'algorithme est peu efficace, mais reste correct.

- Impact pour le praticien
 - Des capacités d'observation bon marché peuvent être utilisées.
 - Mais les performances sont mauvaises.
 - Or dans un système complexe, le « floue » augmente.

Les algorithmes de TaF

- Ils tolèrent des capacités d'observation imparfaites.
 - Bcp. d'algorithmes garantissent des conditions de cohérence.
 - Exemple : capture d'état, synchronisation de messages
 - Ils contraignent le système parmi ses exécutions possibles.
 - Reprise depuis un état donné, ré-ordonnancement de messages
 - Le niveau de « contrainte » dépend des informations perçues.
 - Perception « floue » \Leftrightarrow déduction pessimiste
 - L'algorithme est peu efficace, mais reste correct.
- Impact pour le praticien
 - Des capacités d'observation bon marché peuvent être utilisées.
 - Mais les performances sont mauvaises.
 - Or dans un système complexe, le « floue » augmente.
 - Les performances deviennent inacceptables.

