Une formalisation fonctionnelle des communications sur la puce

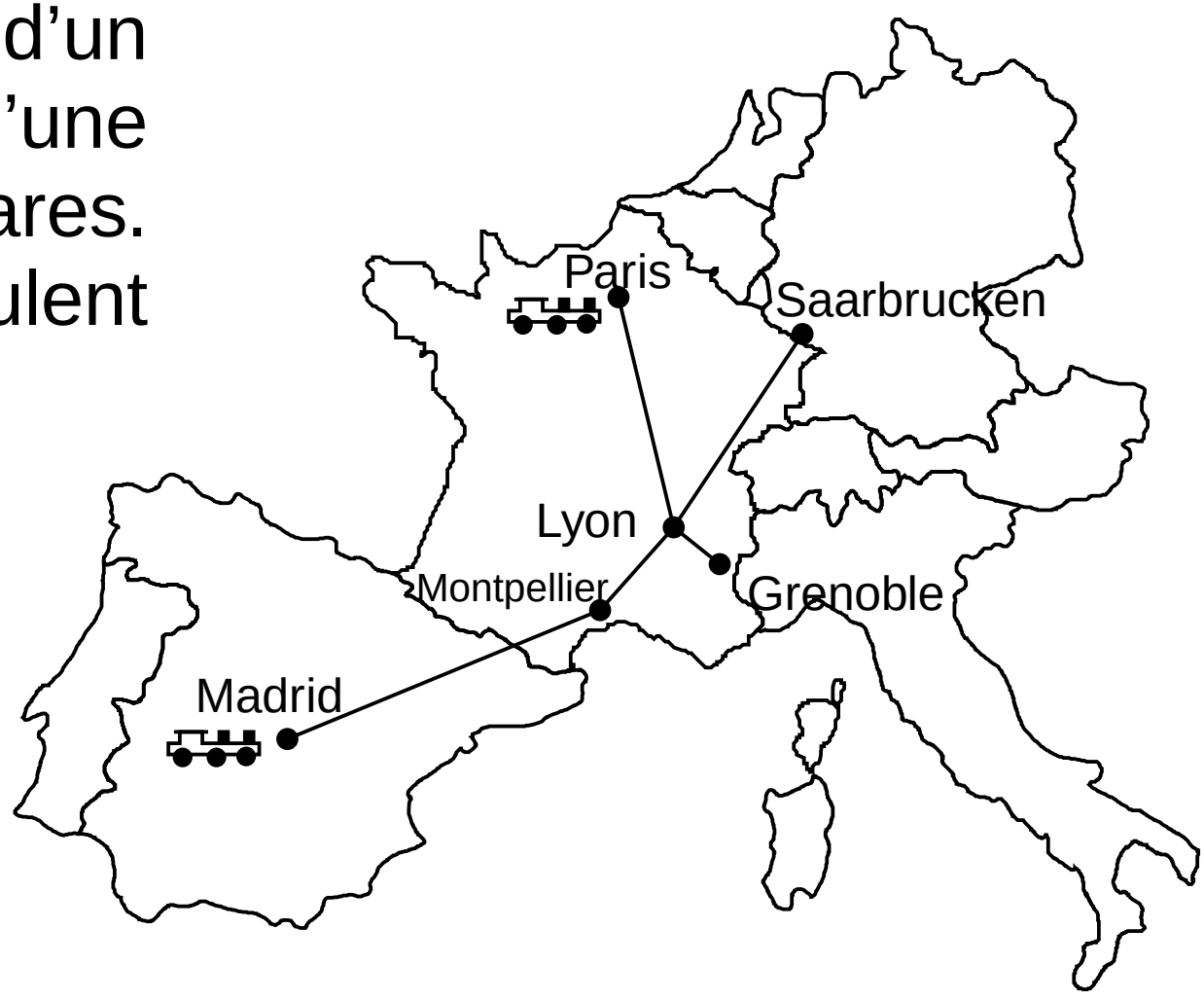
Julien Schmaltz

Thèse dirigée par Dominique Borrione

Laboratoire TIMA - Équipe VDS

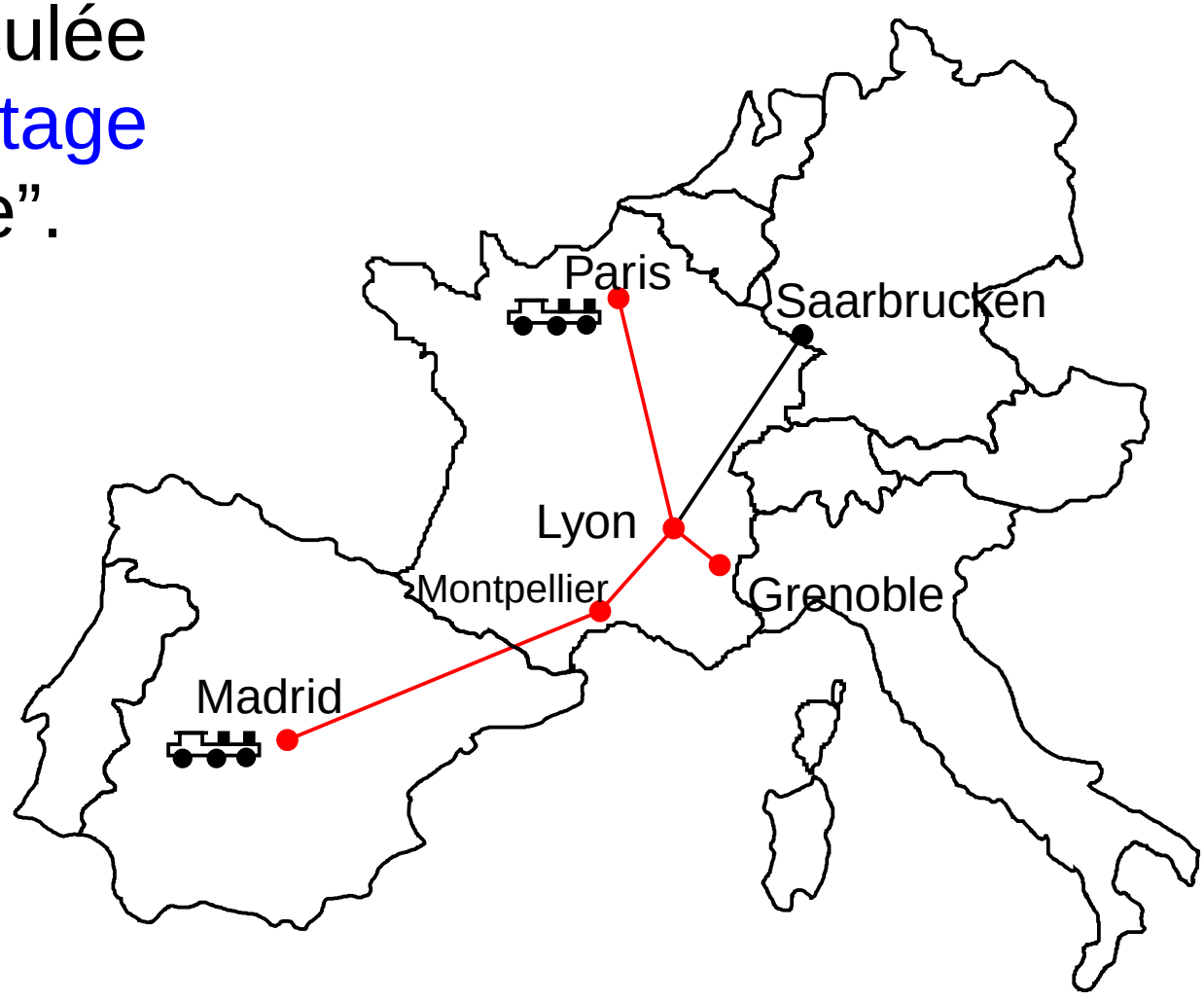
Le réseau ferré

Réseau composé d'un quai par gare et d'une voie entre deux gares. Pierre et José veulent aller à Grenoble.



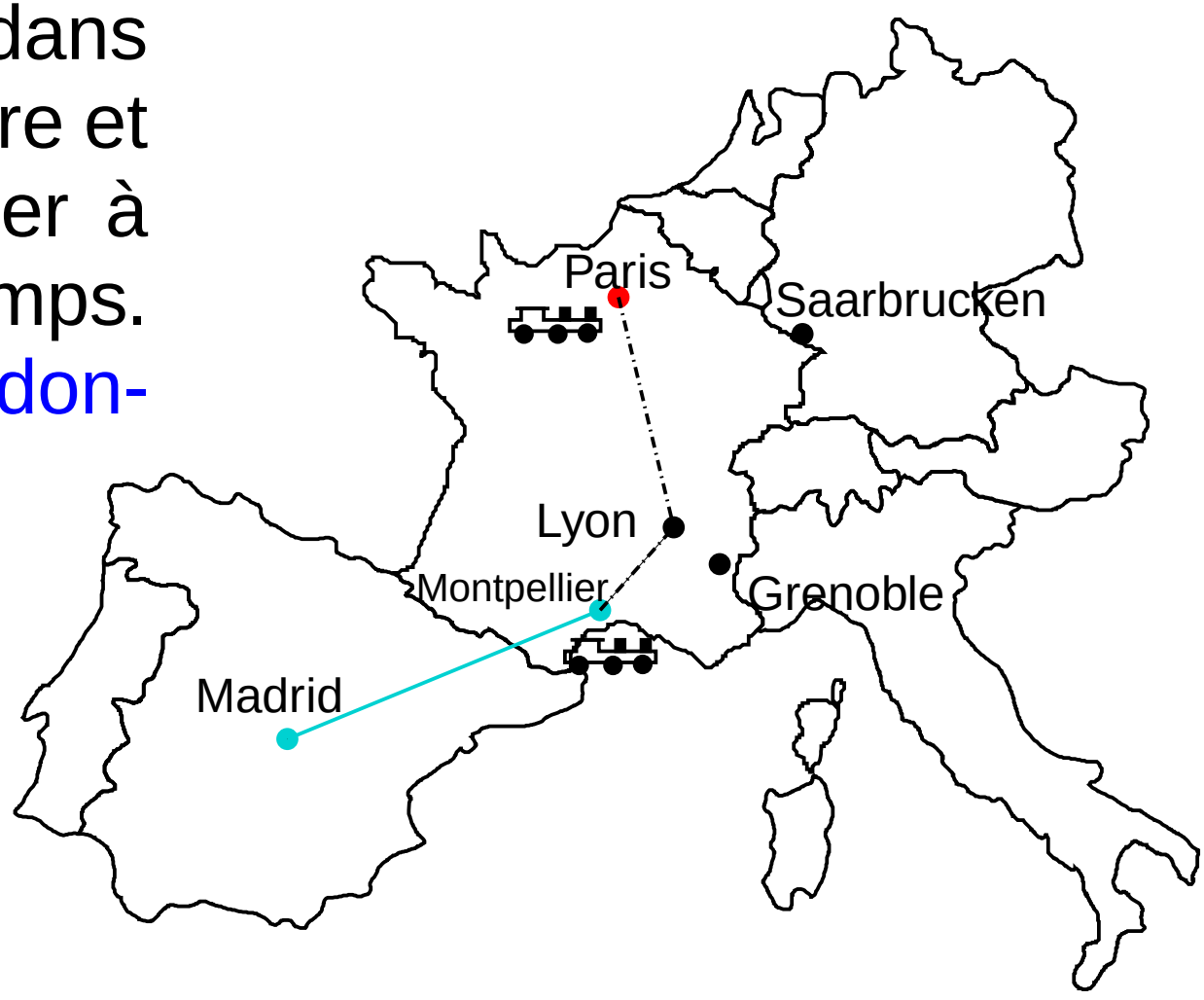
Itinéraire au départ

La route est calculée à l'origine. Le routage est dit "à la source".



Résolution des conflits

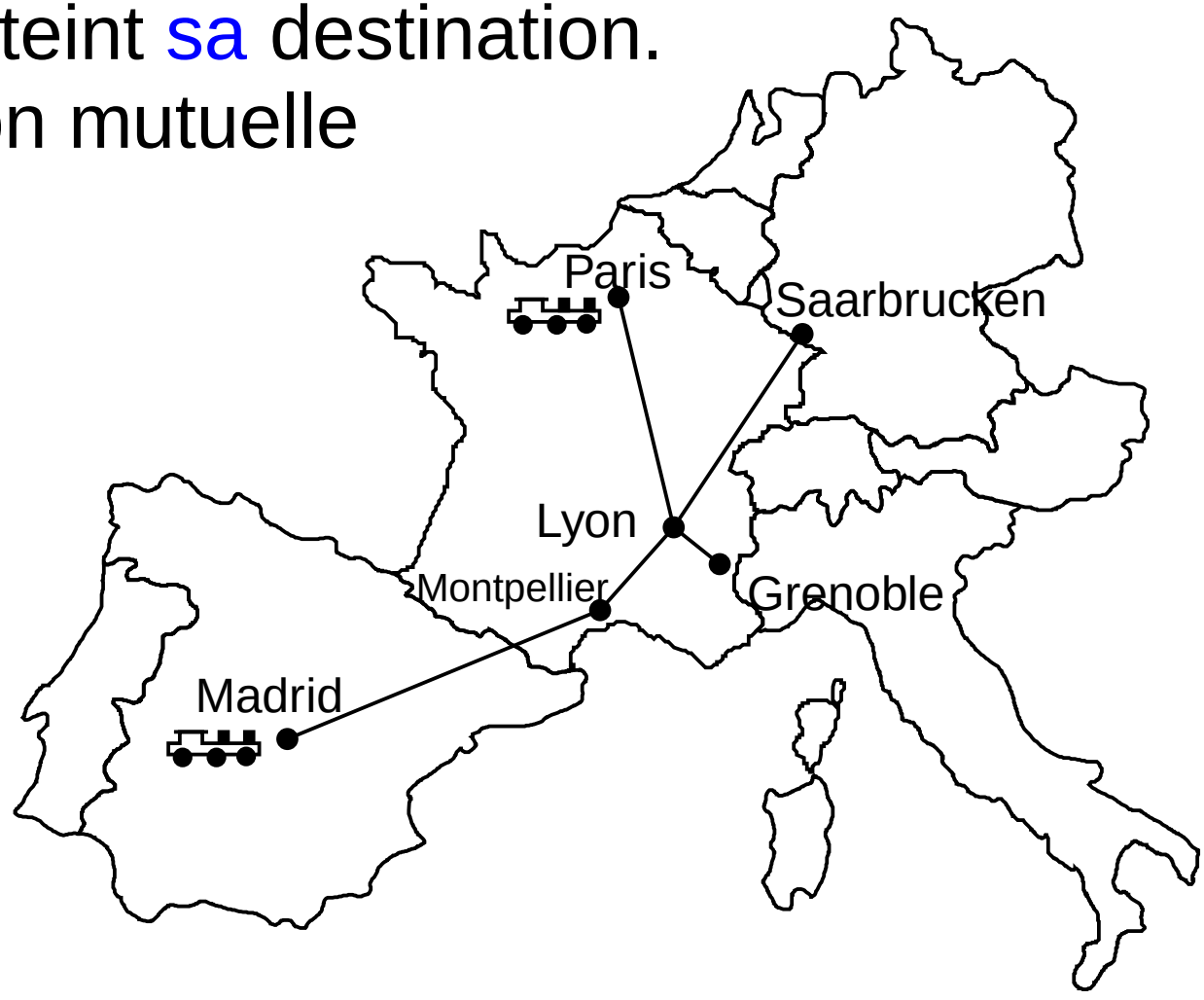
Il y a un seul quai dans chaque gare. Pierre et José veulent entrer à Lyon en même temps. La politique d'ordonnancement résoud le conflit.



Vérification du réseau

Thm. Tout train atteint sa destination.
Preuve : Exclusion mutuelle

et routage correct



Vérification d'un autre réseau

Thm. Tout avion atteint sa destination.
Preuve : Exclusion mutuelle

et routage correct

Le particulier et le général

- Ce qui est particulier **à un** réseau
  - Politique d'ordonnancement
  - Nombre de quais et de voies
- Ce qui est commun **à tout** réseau
  - Structure, routage + ordonnancement
  - Correction du routage et de l'ensemble

Le particulier et le général

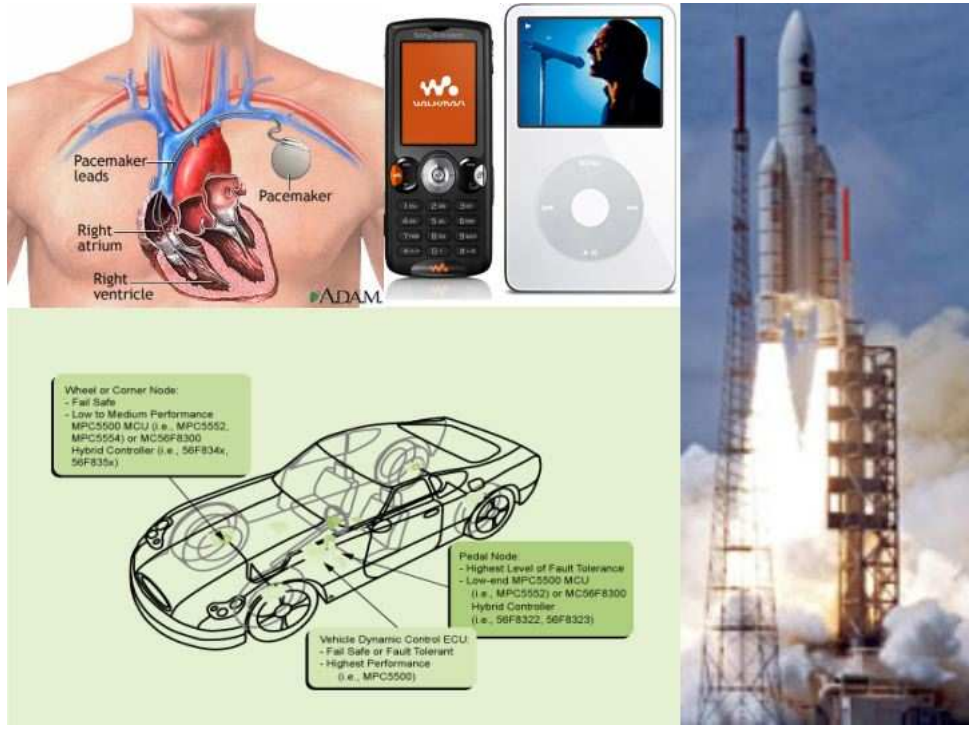
- Ce qui est particulier **à un** réseau
  - Politique d'ordonnancement
  - Nombre de quais et de voies
- Ce qui est commun **à tout** réseau
  - Structure, routage + ordonnancement
  - Correction du routage et de l'ensemble

Ma thèse :
un modèle **formel** d'un réseau **en général**

Plan

- Puces, vérification et réseaux
- Ma thèse
- Formalisation fonctionnelle : $GeNoC$
- Méthodologie

Les systèmes sur puce



- Omniprésence, systèmes critiques
- Complexité croissante
- Sûreté et bon fonctionnement

TOP 10 des “bugs”

- Pentium FDIV (1995), >450 M$
- Ariane 5 (1996)
- National Cancer Institute, Panama City (2000)
  - Multidata Systems International,
  - Au moins 8 morts,
  - 20 patients avec de graves séquelles
  - Mise en accusation pour homicide involontaire
- . . .

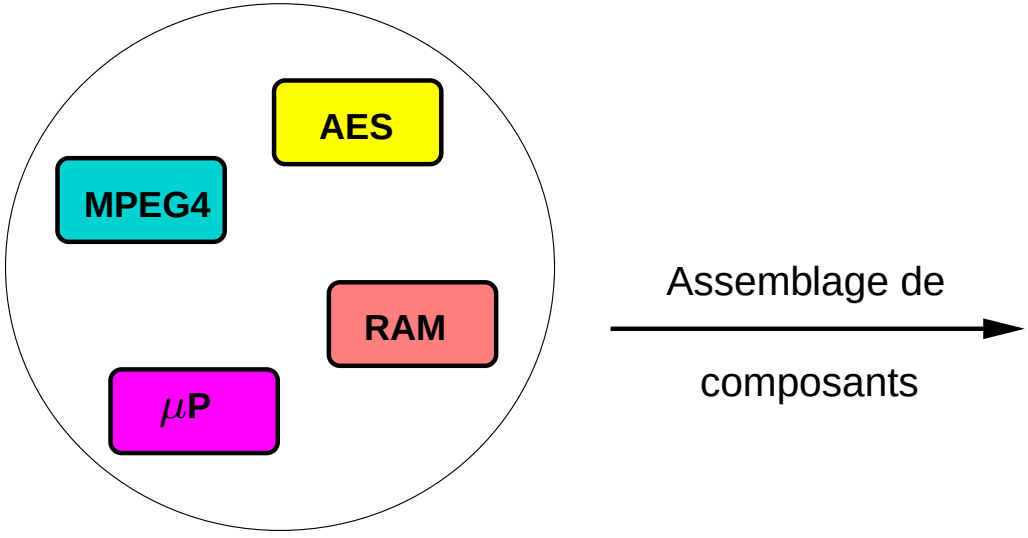
Méthodes de vérification

- Simulation numérique :
  - Test intensif des programmes mais la complexité rend impossible une vérification exhaustive
- Méthodes formelles
  - Preuve pour toutes les entrées et tous les états possibles

“Au lieu de déboguer un programme, on devrait prouver qu’il satisfait sa spécification, et la preuve devrait être vérifiée par un programme informatique”, J. McCarthy, 1962

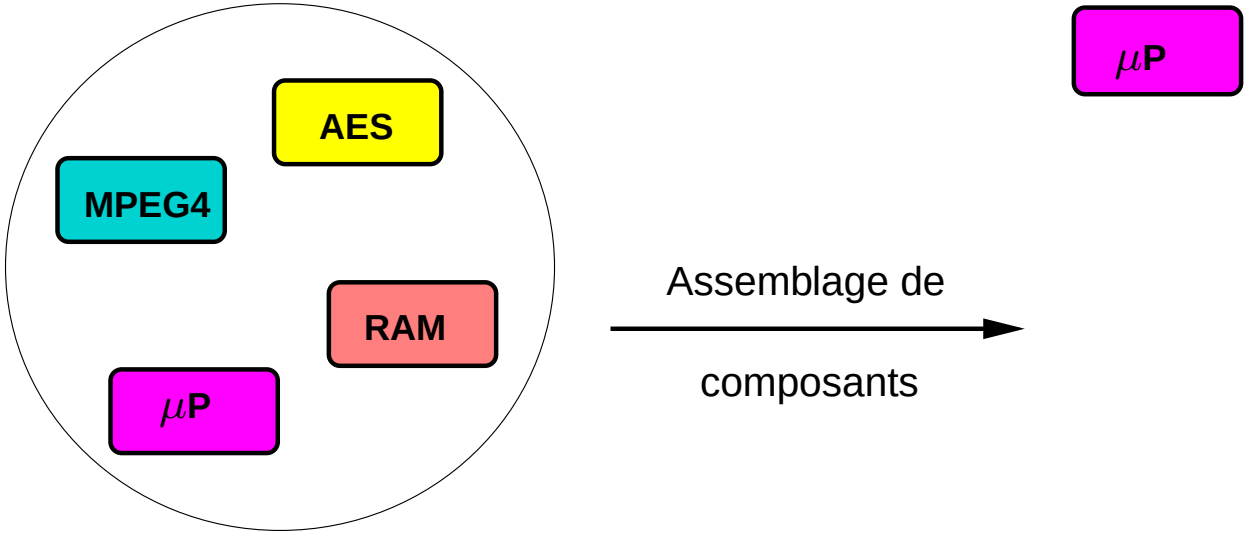
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction



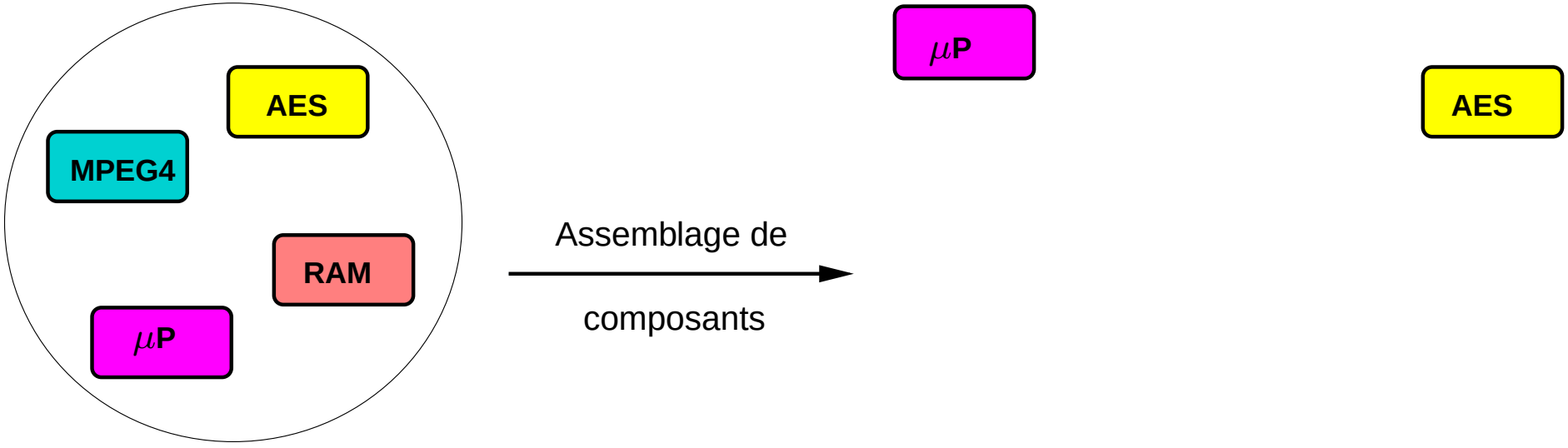
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction



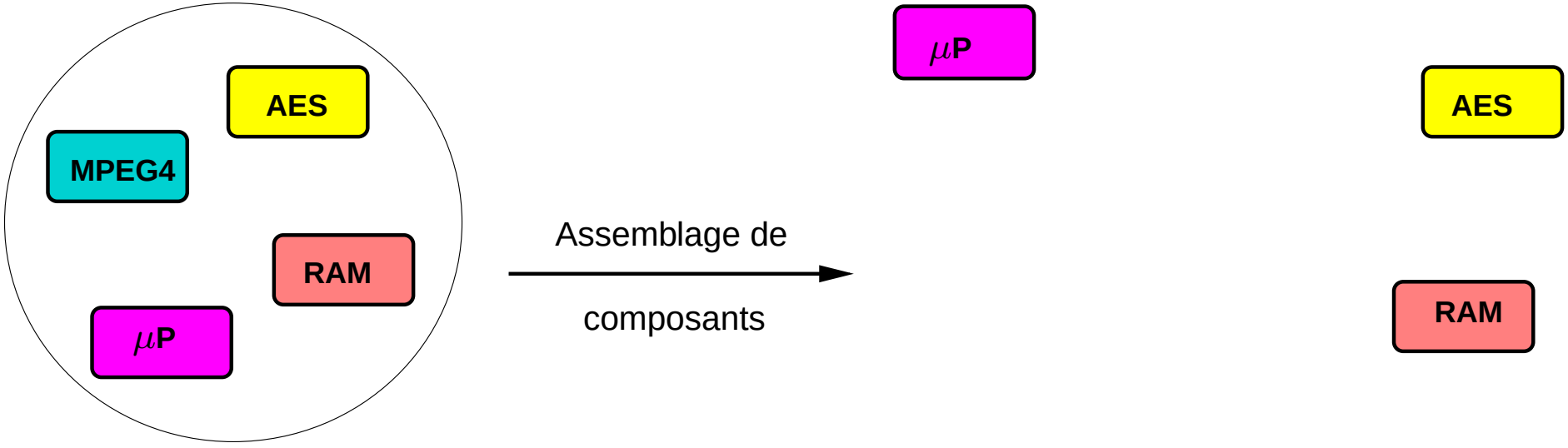
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction



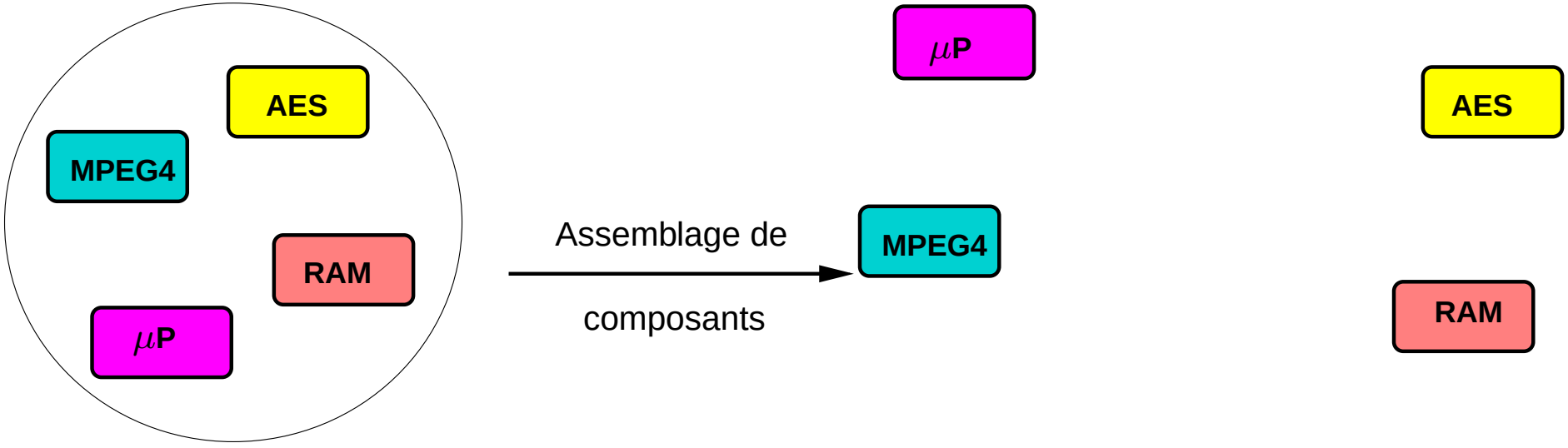
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction



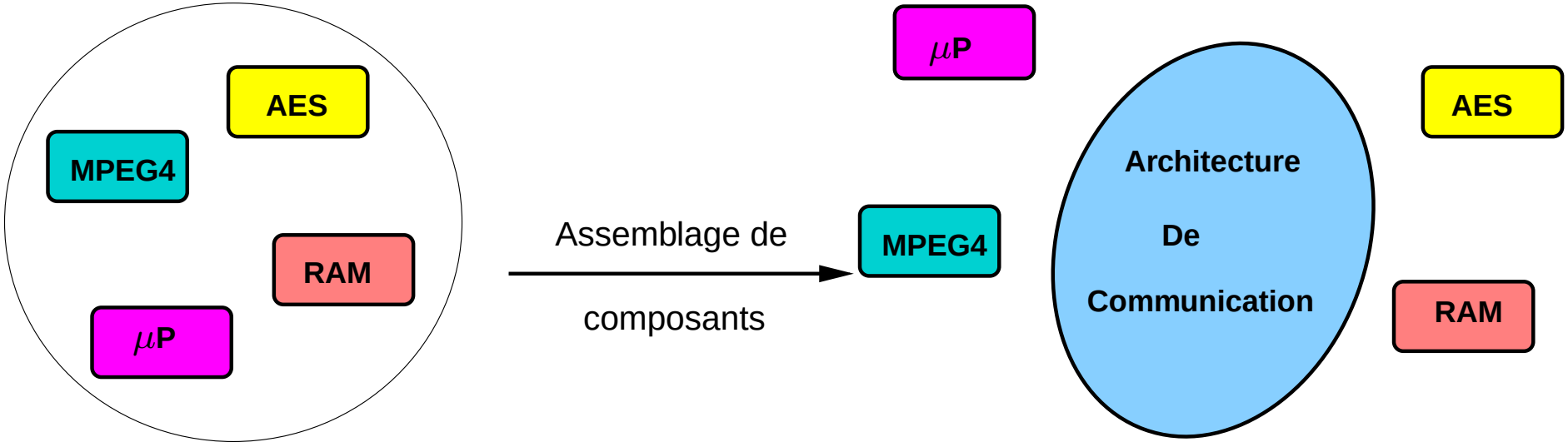
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction



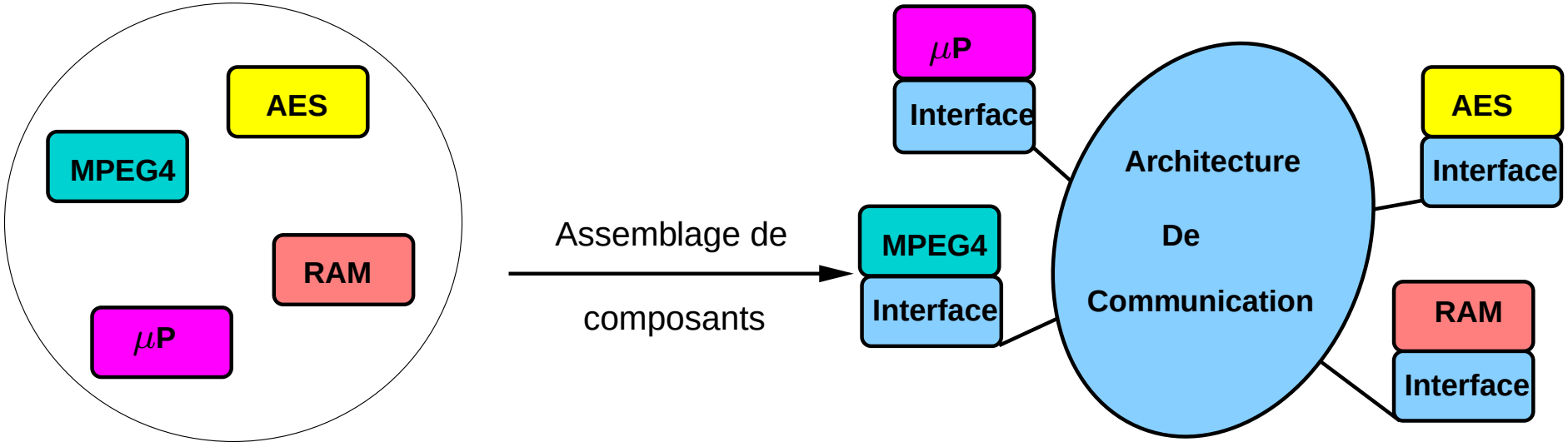
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction



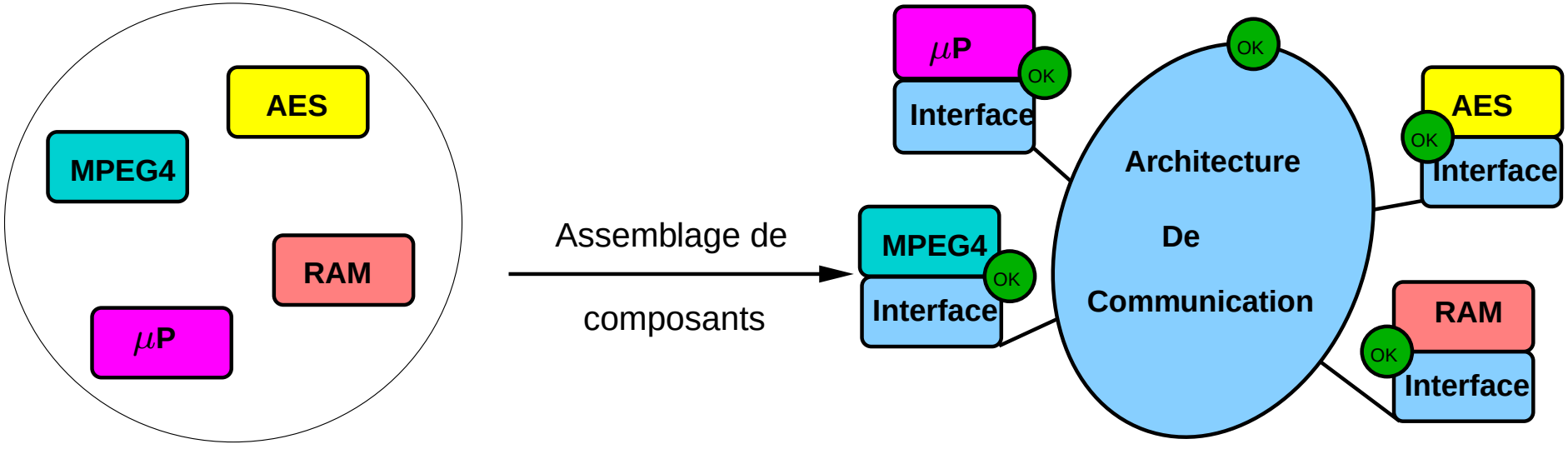
Le paradigme SoC

- Conception orientée plateforme :
  - Réutilisation de composants paramétrés
  - Haut niveau d'abstraction

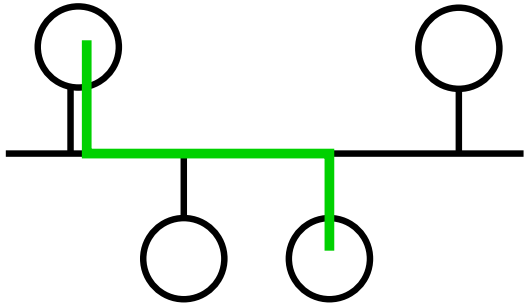


La vérification des SoC's

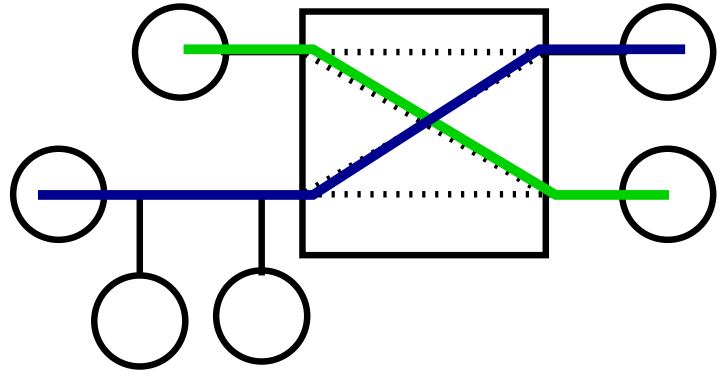
- Preuve de l'assemblage :
  - Preuve de chaque composant
  - Preuve de leur interconnexion



Des bus . . .



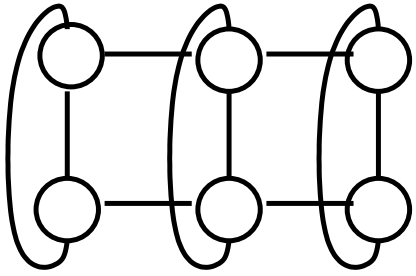
Bus partagé



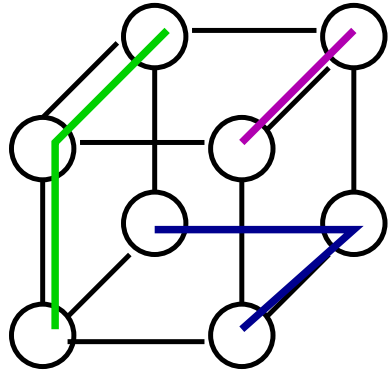
Bus matriciel

- (+) Faible surface
- (+) Simplicité
- (-) Difficilement extensible
- (-) Faible parallélisme

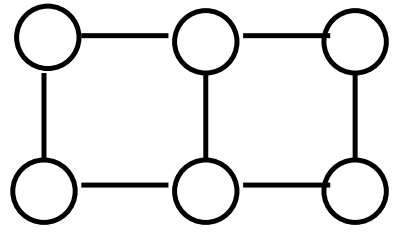
. . . aux réseaux



Tore



Cube



Grille

- (+) Facilement extensible
- (+) Haut niveau de parallélisme
- (-) Complexité
- (-) Augmentation de la surface

Les méthodes formelles

- Recherche des “bugs”
  - Méthodes algorithmiques (automatiques)
  - Contre-exemples
  - Systèmes non paramétrés
  - Bas niveau d’abstraction
- Preuve de l’absence de “bugs”
  - Méthodes déductives
  - Assistants de preuve (non automatiques)
  - Systèmes paramétrés
  - Haut niveau d’abstraction

Vérification formelle des NoC's

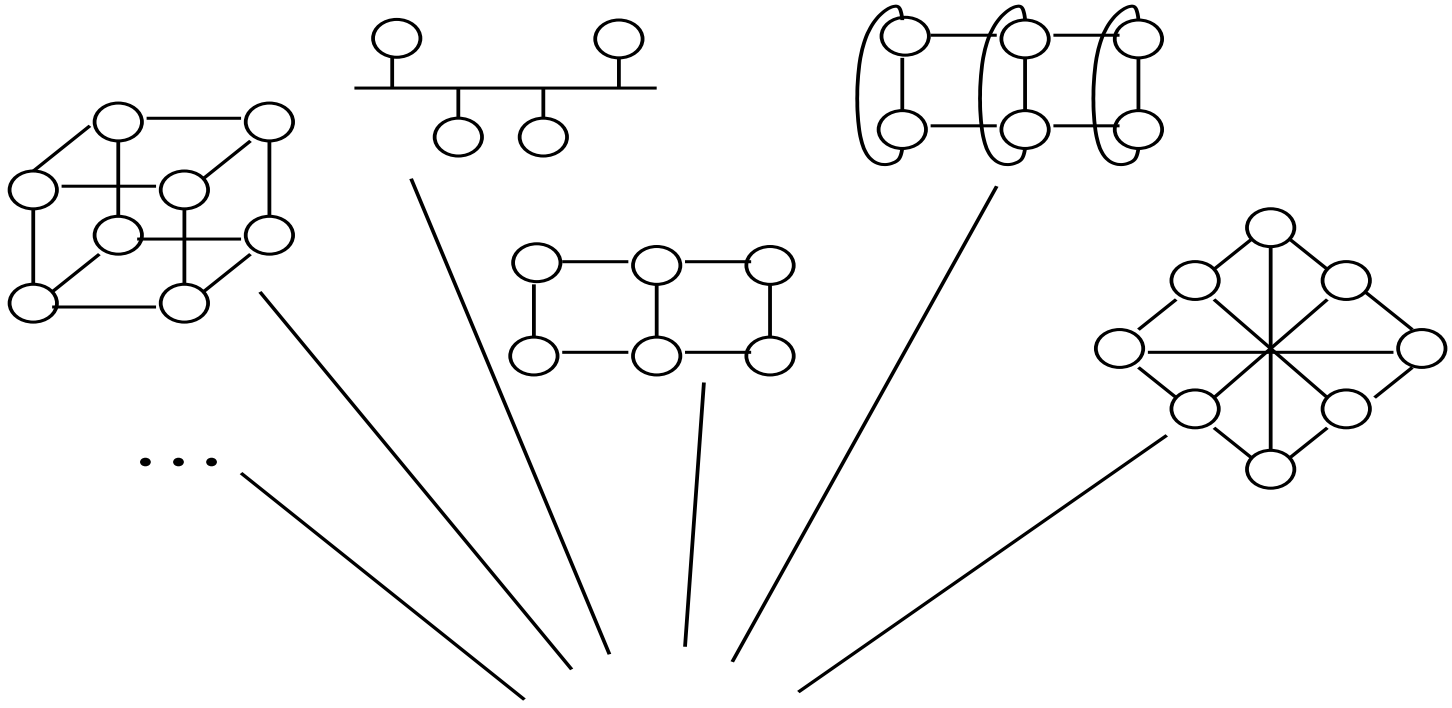
- Bus AMBA par vérification de modèles (Roychoudhury *et al.*, 2003)
- Bus AMBA par vérif. de mod. et HOL (Amjad, 2004)
- Protocole Æthereal de Philips en utilisant PVS (Gebremichael *et al.*, 2005)
  - Bas niveau d'abstraction
  - Uniquement des cas particuliers
  - Pas de méthode générale

Plan

- Puces, réseaux et “bugs”
- Ma thèse
- Formalisation fonctionnelle : $GeNoC$
- Méthodologie

Objectif global

Un modèle pour **toute** architecture



Contribution

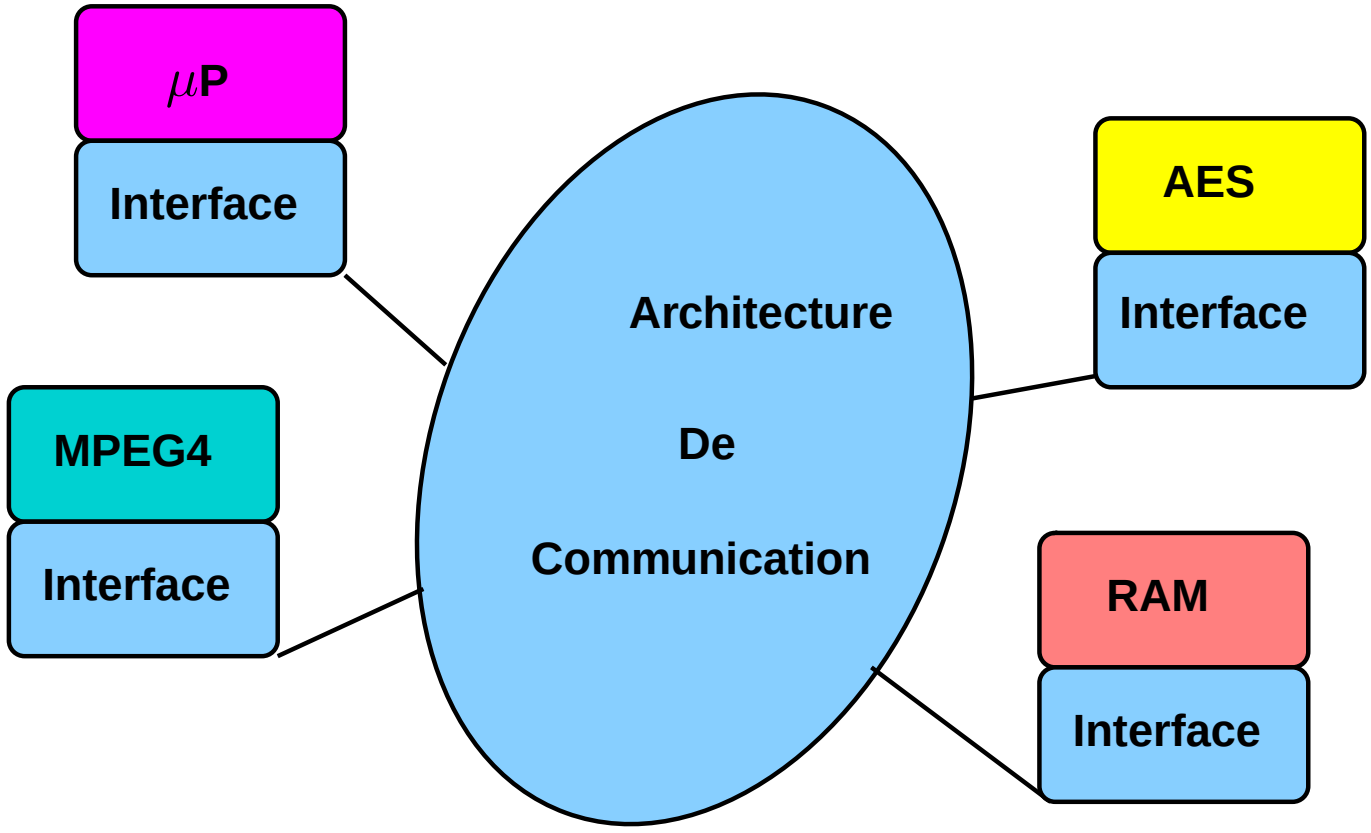
Une formalisation fonctionnelle des communications : $GeNoC$ (Generic Network on Chip)

- Identification des composantes essentielles et de leurs propriétés
- Formalisation des interactions entre ces composantes
- Correction de l'ensemble est une conséquence des propriétés essentielles des composantes
- Méthodologie supportée par des outils "automatiques"

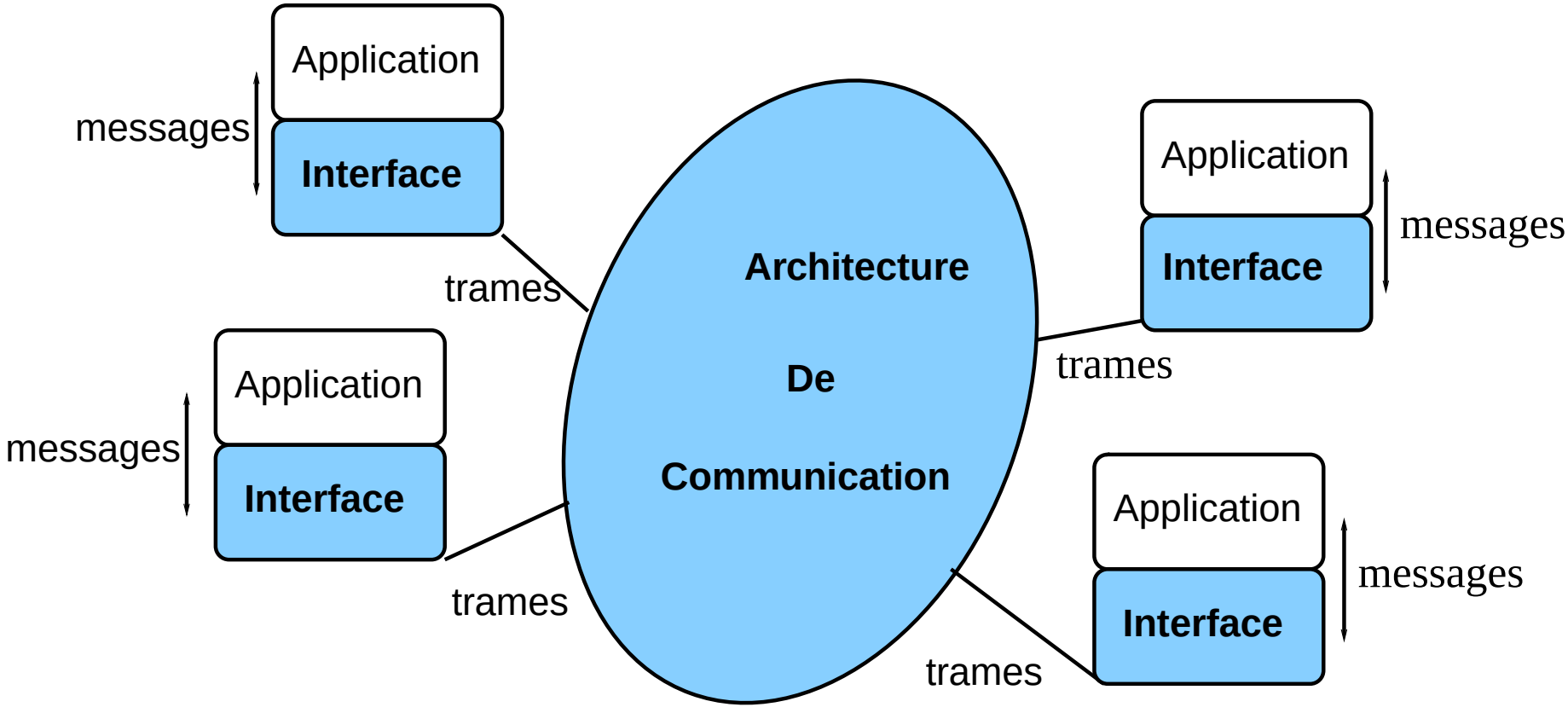
Plan

- Puces, réseaux et “bugs”
- Ma thèse
- Formalisation fonctionnelle : $GeNoC$
- Méthodologie

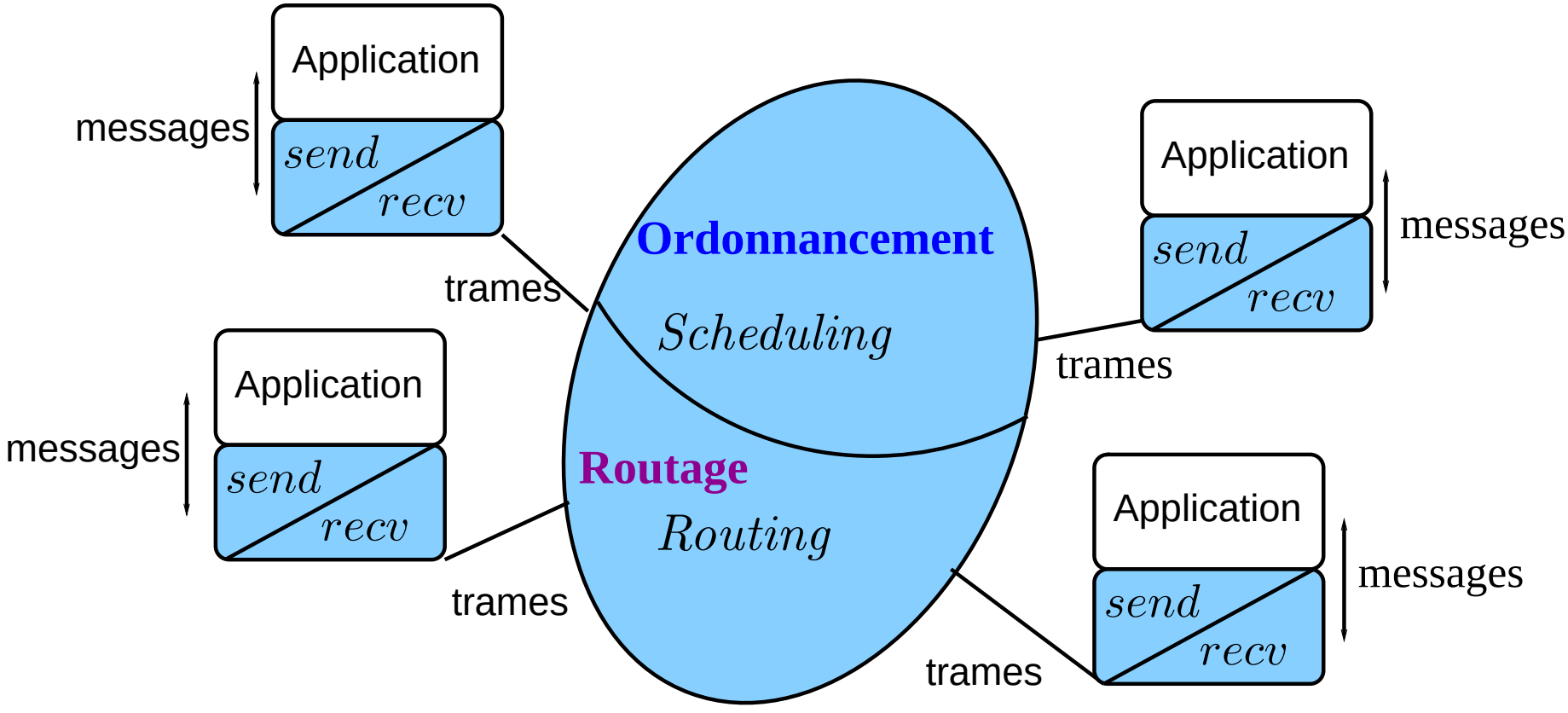
Principes de la généralisation



Principes de la généralisation

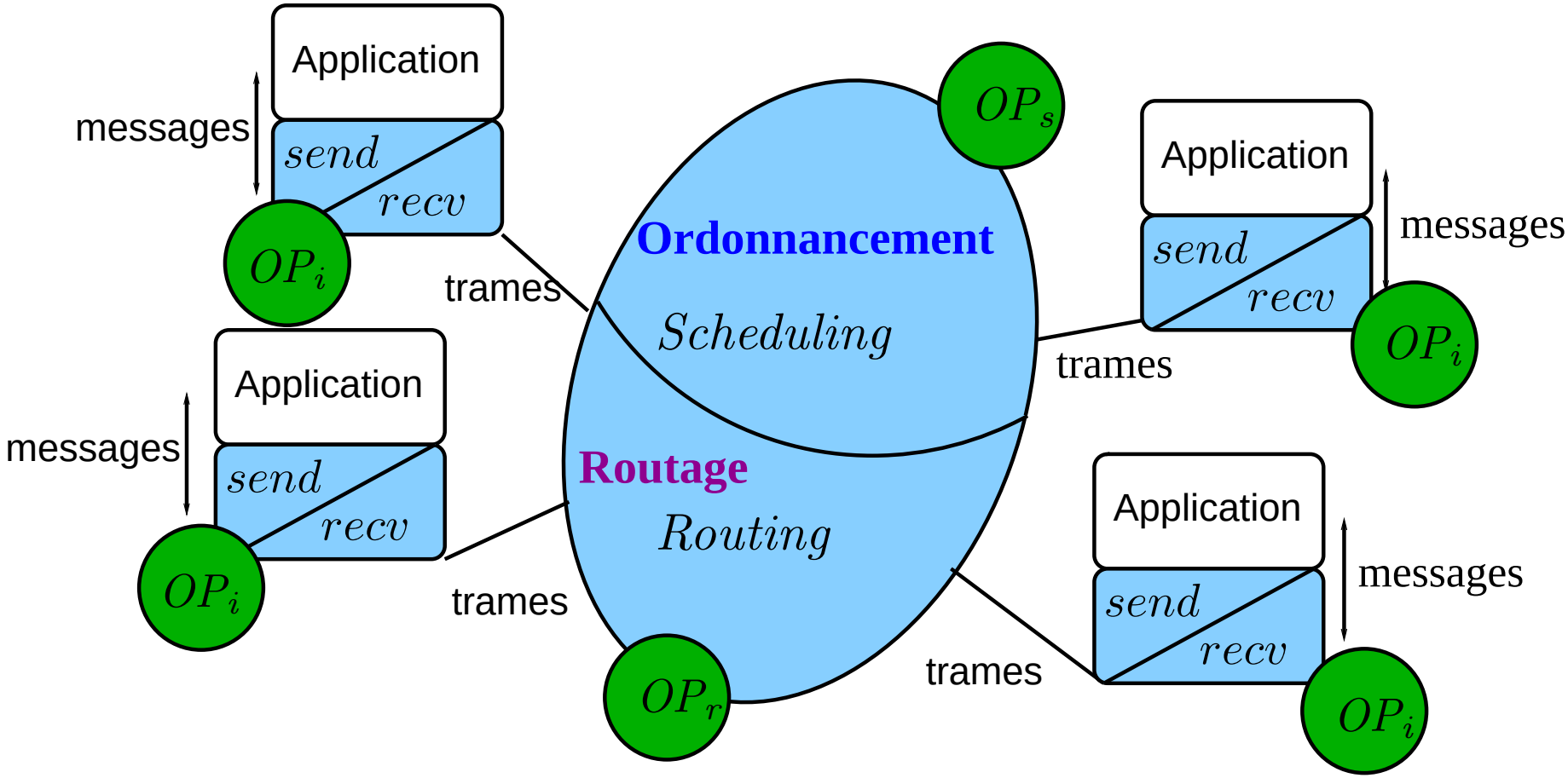


Modélisation fonctionnelle



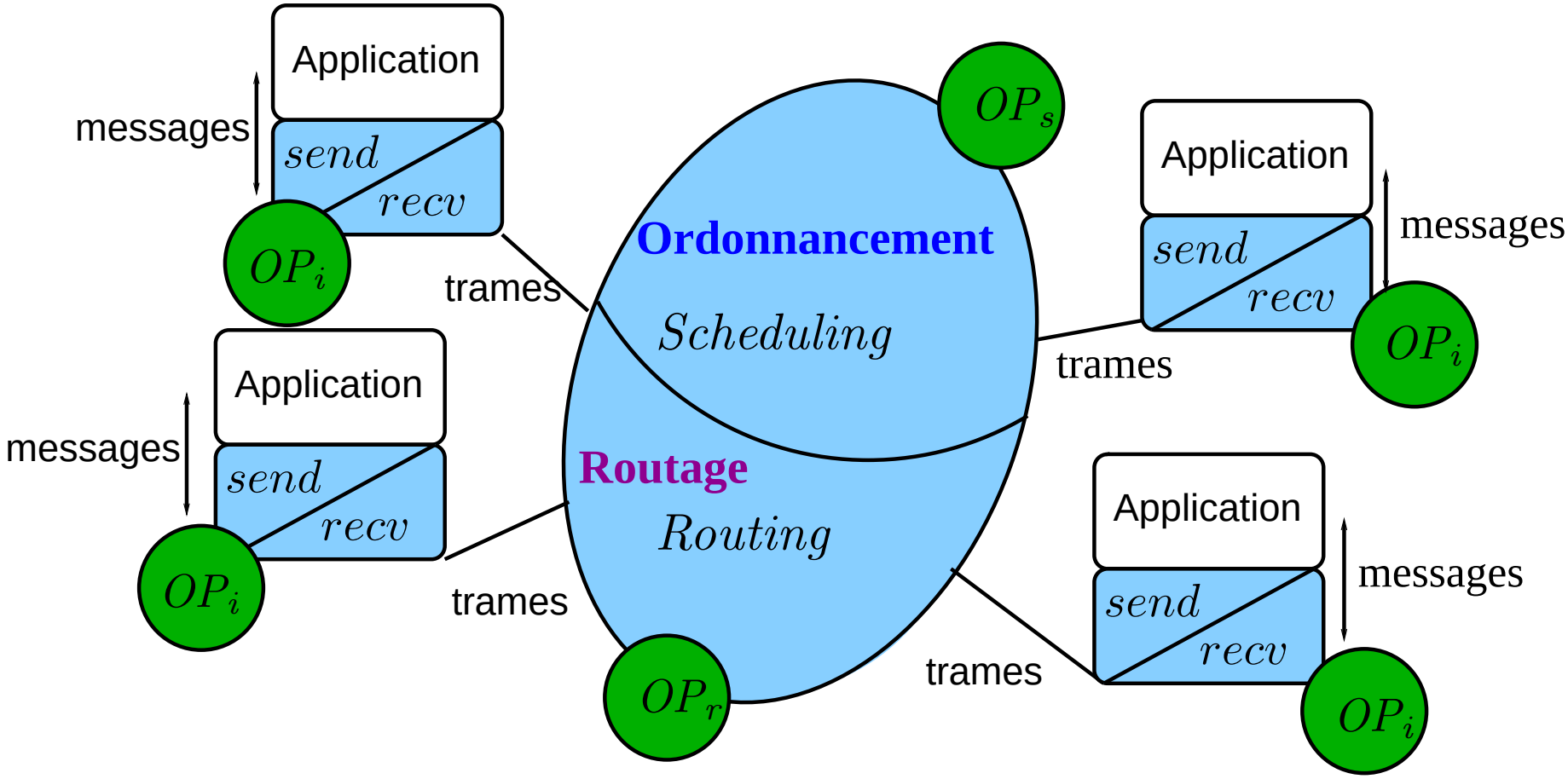
Système $= \mathcal{F}(Routing, Scheduling, recv, send)$

Obligations de preuve



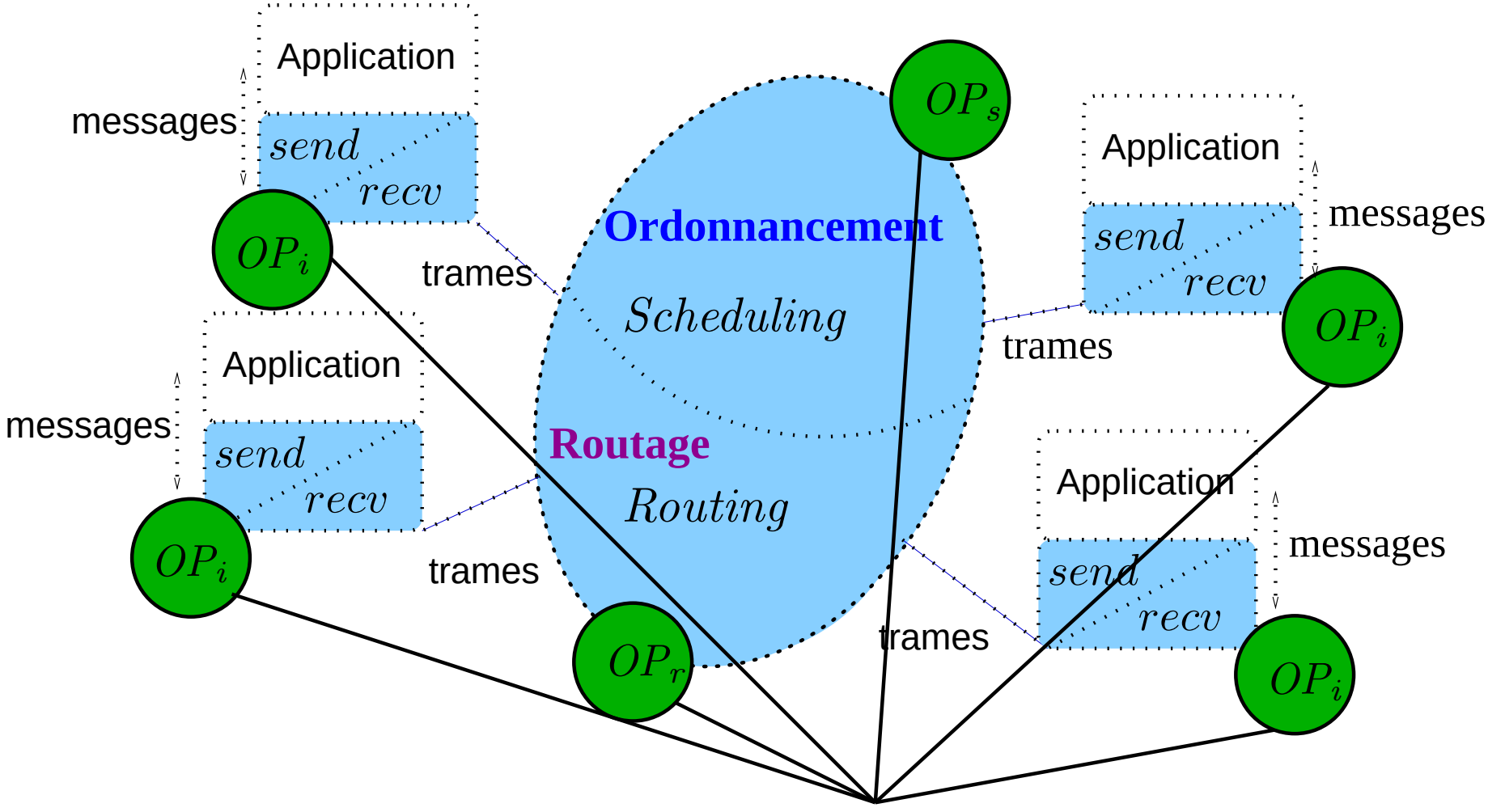
Système $= \mathcal{F}(Routing, Scheduling, recv, send)$

Le théorème sur l'ensemble



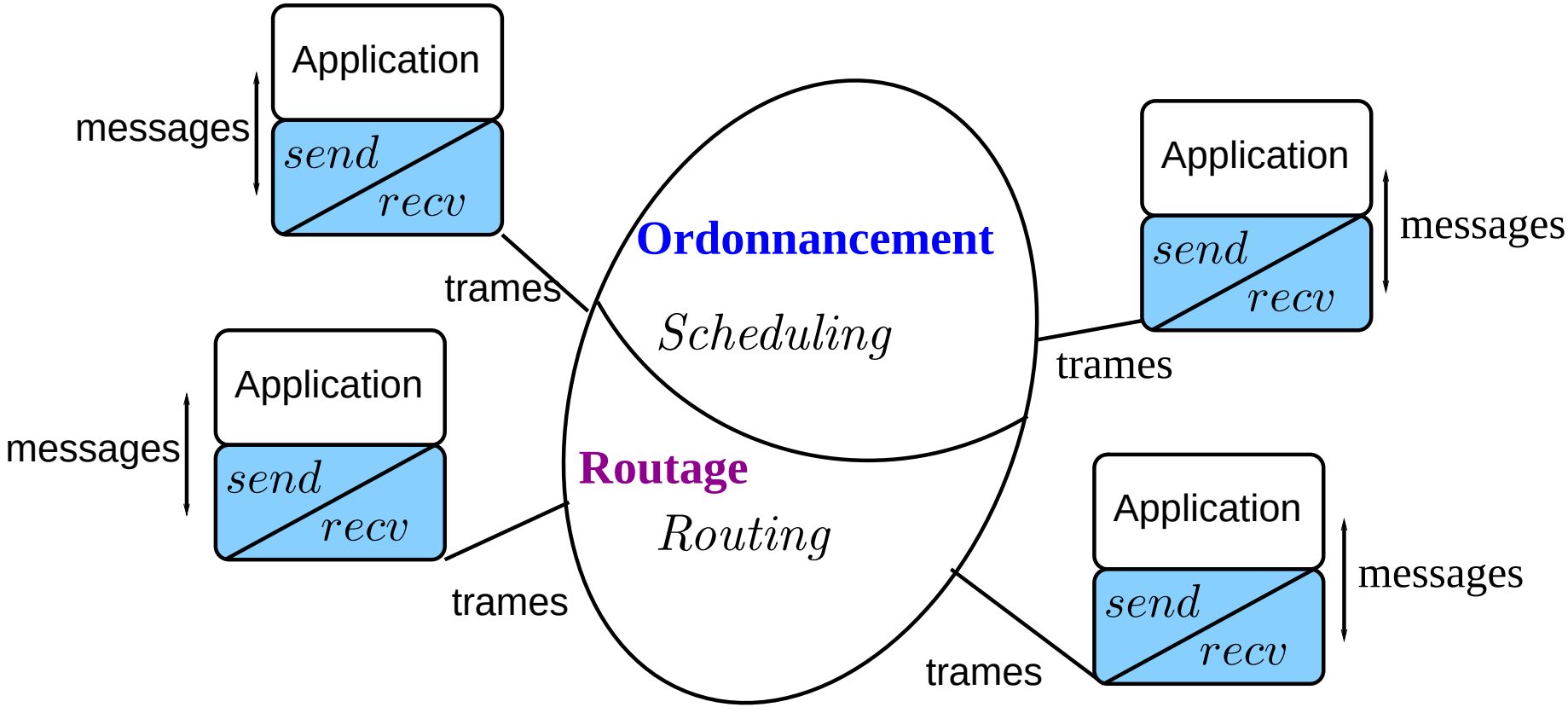
Thm : tout message atteint sa destination

Le théorème sur l'ensemble



Thm : tout message atteint sa destination

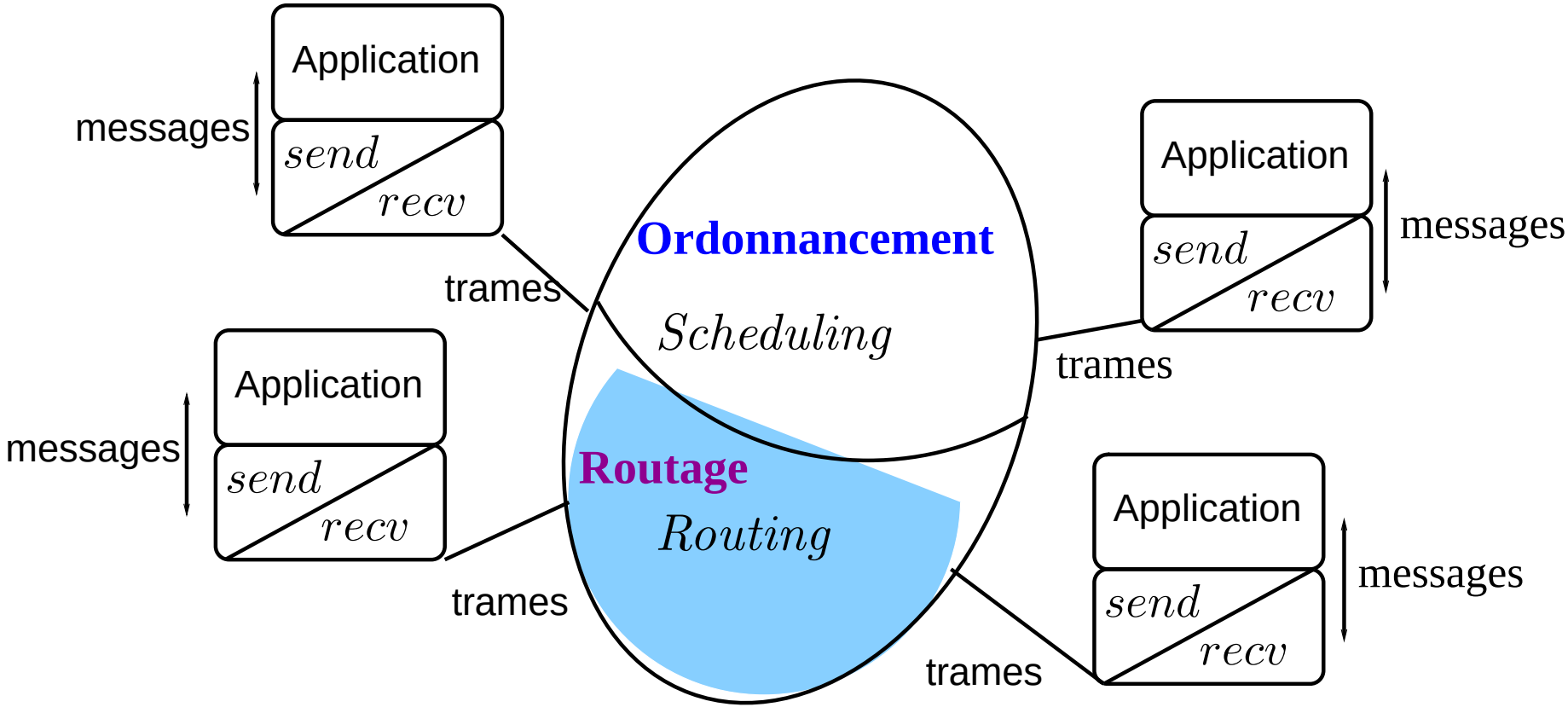
Les interfaces



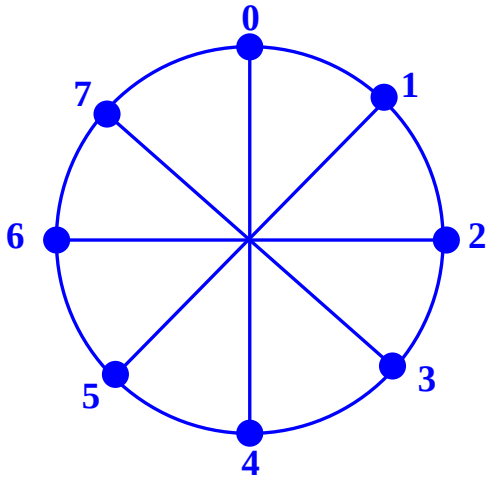
Les interfaces

- Modélisation
  - Lien avec le modèle OSI (*e.g.* couches 1 à 4)
  - La fonction $send$ produit des trames à partir de messages
  - La fonction $recv$ consomme les trames
- Obligation de preuve
  - La composition $recv \circ send$ est une identité

Le routage


Application
messages
send
recv
trames
Ordonnancement
Scheduling
Routage
Routing
Application
messages
send
recv
trames
Application
messages
send
recv
trames
Application
messages
send
recv
trames

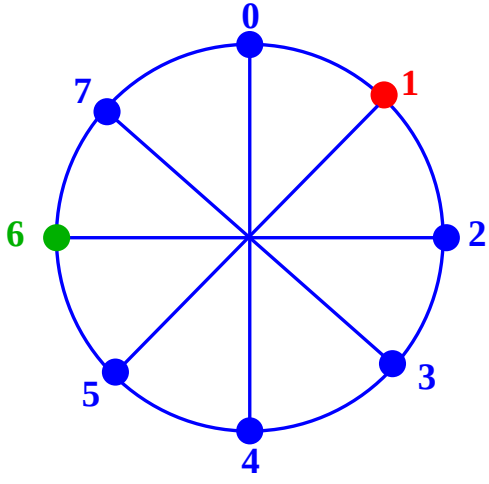
Le réseau Octagon



- 8 nœuds
- Extensible à $4 * i$
- Liens bidirectionaux
- Routage selon le plus court chemin

- Conçu par STMicroelectronics, ref : DAC'01 et IEEE Micro 2002 par F. Karim *et al*.

Algorithme de routage



$RelAd = (dest - current) \bmod 8$

if $RelAd = 0$

then stop

elsif $RelAd = 1 \vee 2$

then go clockwise

elsif $RelAd = 6 \vee 7$

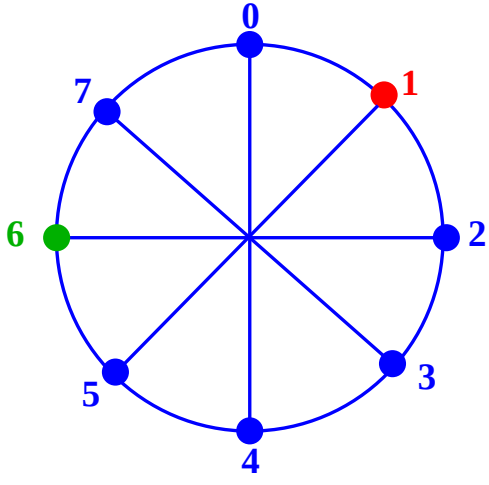
then go counter clockwise

else go across

endif

Exemple : route de 1 vers 6

Algorithme de routage



$RelAd = (6 - 1) \bmod 8$

if $RelAd = 0$

then stop

elsif $RelAd = 1 \vee 2$

then go clockwise

elsif $RelAd = 6 \vee 7$

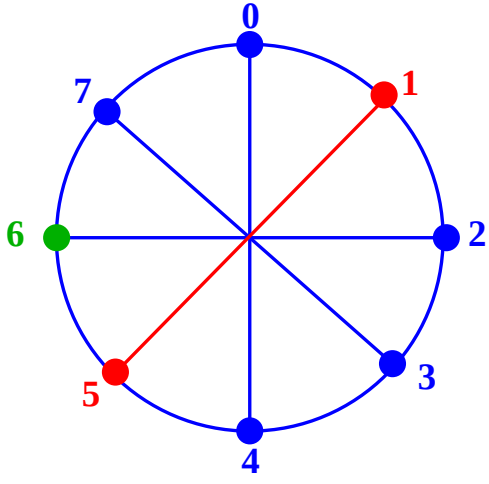
then go counter clockwise

else go across

endif

Exemple : route de 1 vers 6

Algorithme de routage



$RelAd = (6 - 5) \bmod 8$

if $RelAd = 0$

then stop

elsif $RelAd = 1$

then go clockwise

elsif $RelAd = 6 \vee 7$

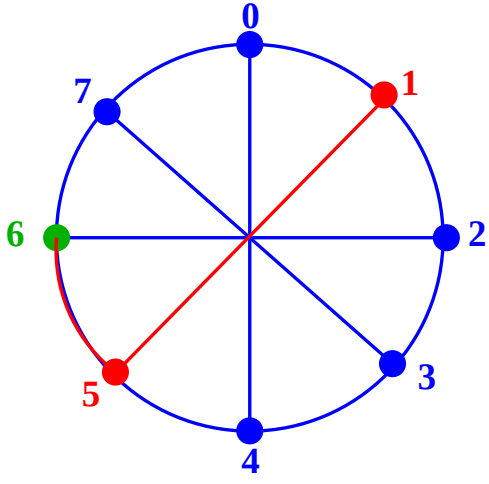
then go counter clockwise

else go across

endif

Exemple : route de 1 vers 6

Algorithme de routage



$RelAd = (6 - 6) \bmod 8$

if $RelAd = 0$

then stop

elsif $RelAd = 1 \vee 2$

then go clockwise

elsif $RelAd = 6 \vee 7$

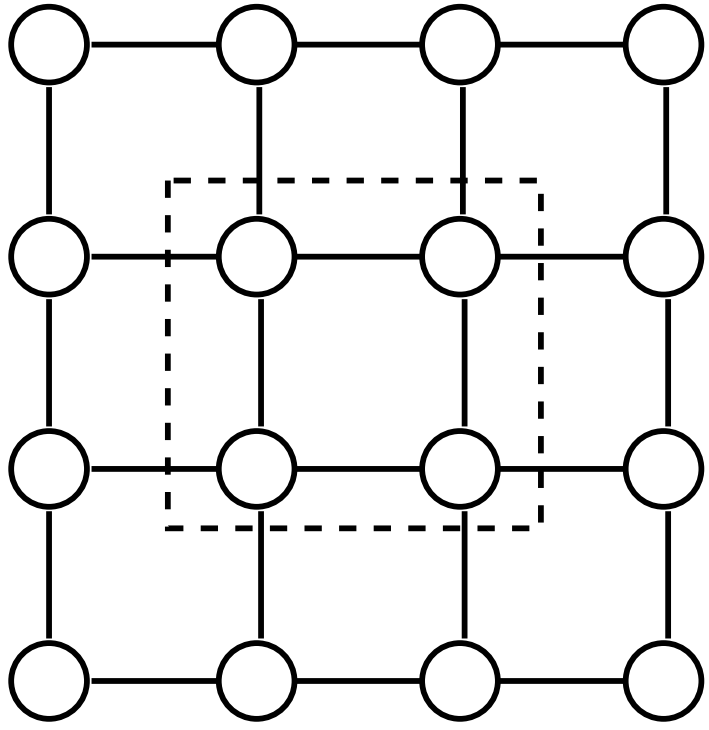
then go counter clockwise

else go across

endif

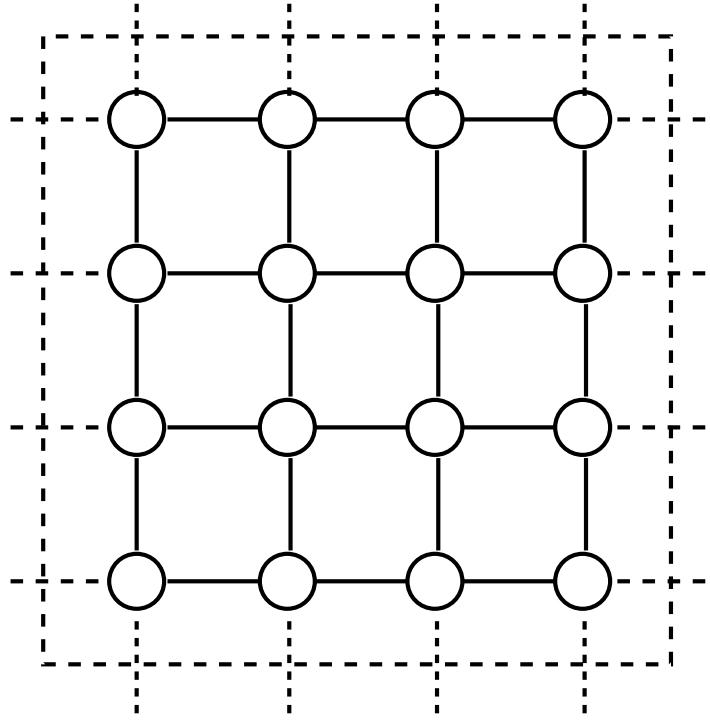
Exemple : route de 1 vers 6

Régularisation des architectures



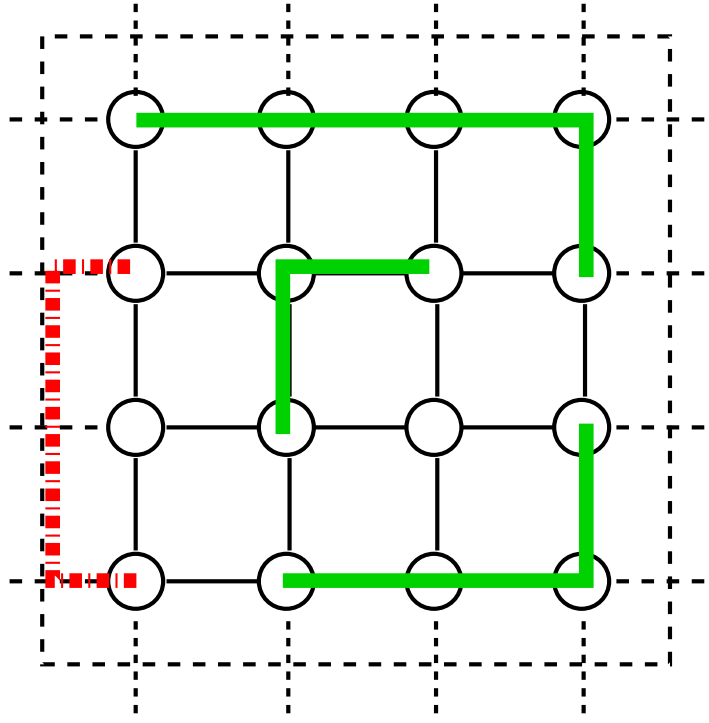
Mais : la plupart des structures sont irrégulières . . .

Régularisation des architectures



. . . elles sont “régularisées” en considérant que tous les nœuds sont identiques.

Régularisation des architectures



Cette régularisation est valide si toute route n'emprunte que des nœuds appartenant au réseau irrégulier initial.

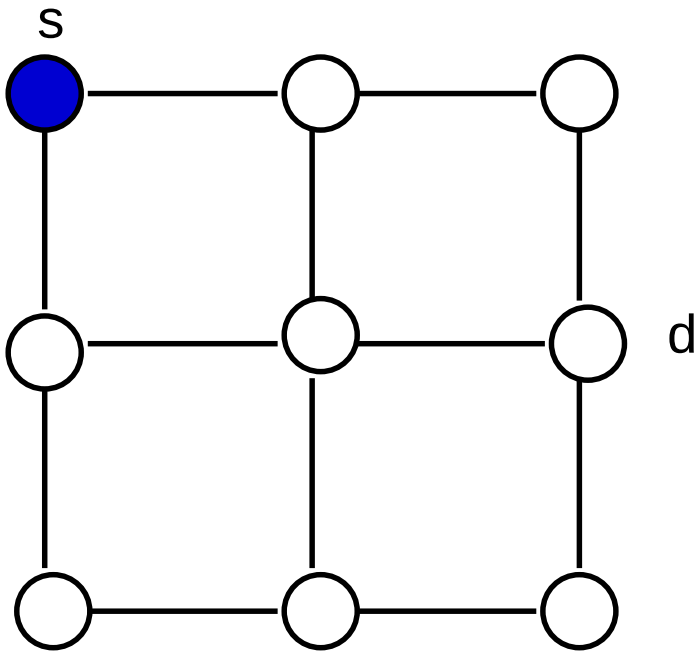
Logique de routage

- Identique à tous les nœuds
- Détermine les déplacements unitaires possibles
- Représentée par une fonction $\mathcal{L}$

$$prochain = \mathcal{L}(courant, destination)$$

- Principe de modélisation
  - Calcul de toutes les routes autorisées par la logique de routage

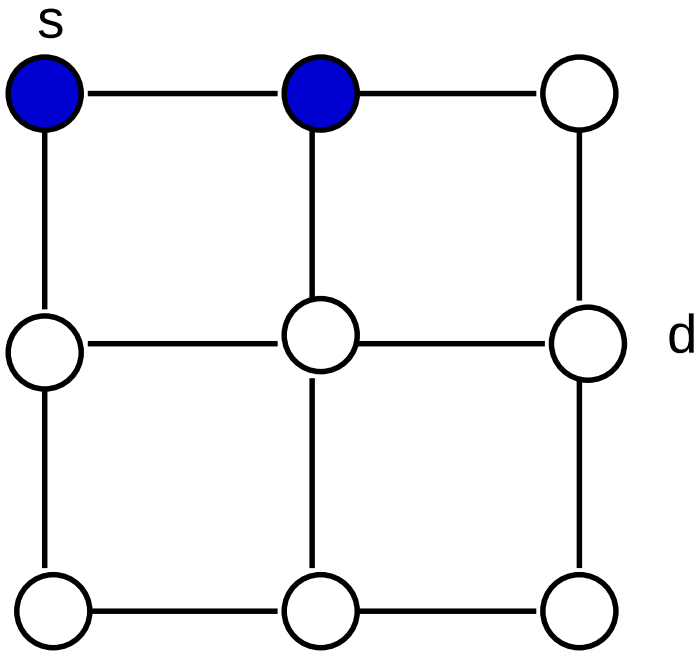
Routage : cas déterministe



Route =

s,

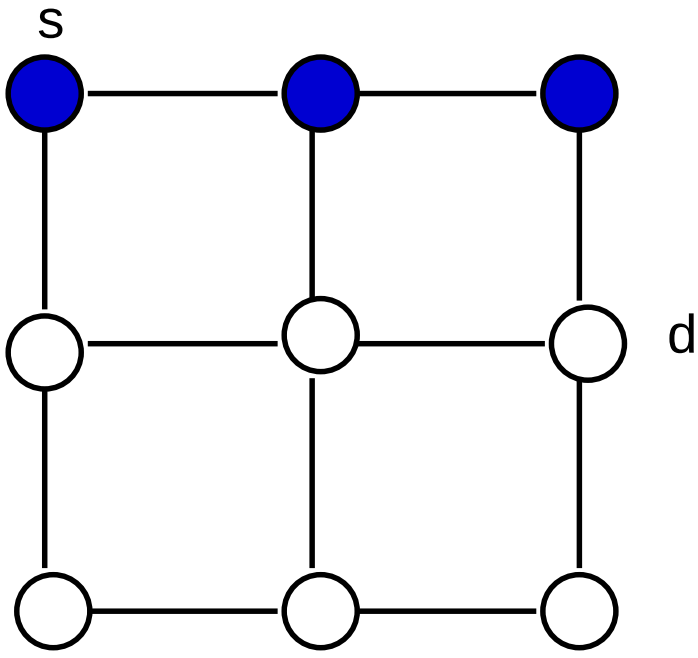
Routage : cas déterministe



Route =

s, $\mathcal{L}(s, d)$,

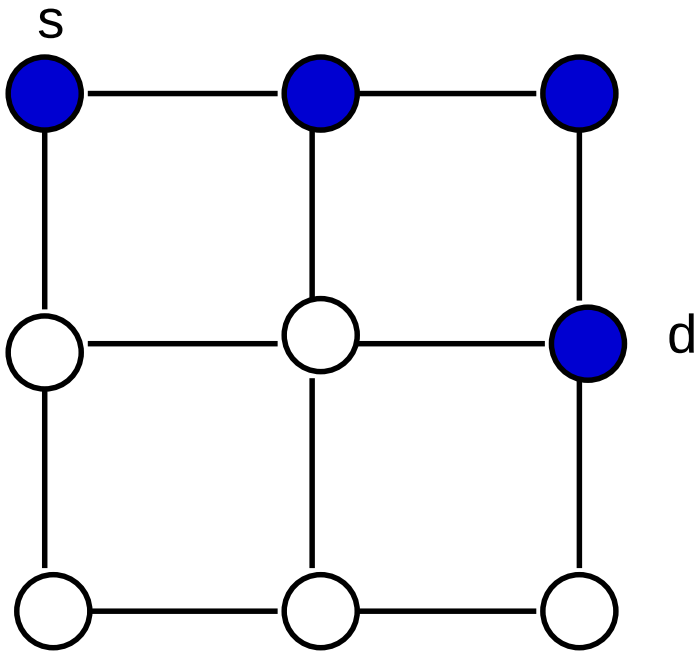
Routage : cas déterministe



Route =

s, $\mathcal{L}(s,d)$, $\mathcal{L}(\mathcal{L}(s,d),d)$,

Routage : cas déterministe

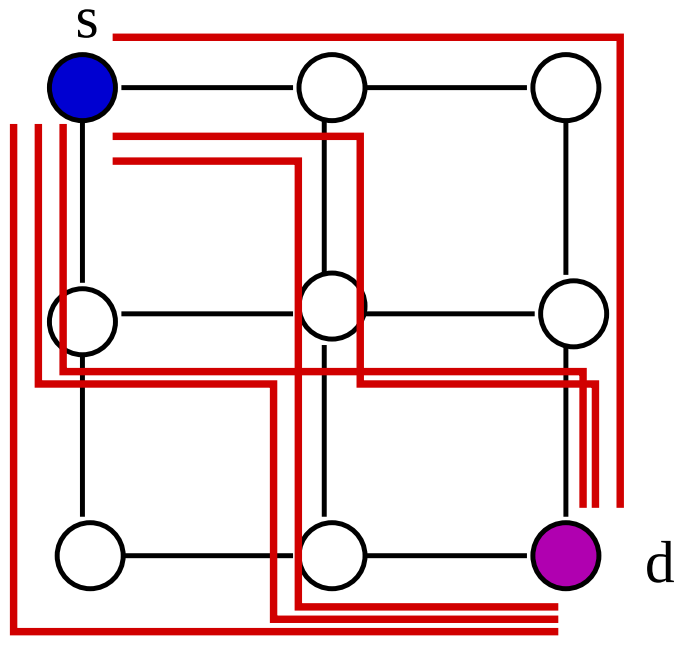


Route =

s, $\mathcal{L}(s,d)$, $\mathcal{L}(\mathcal{L}(s,d),d)$, d

Routage : cas adaptatif

Routage adaptatif et minimal

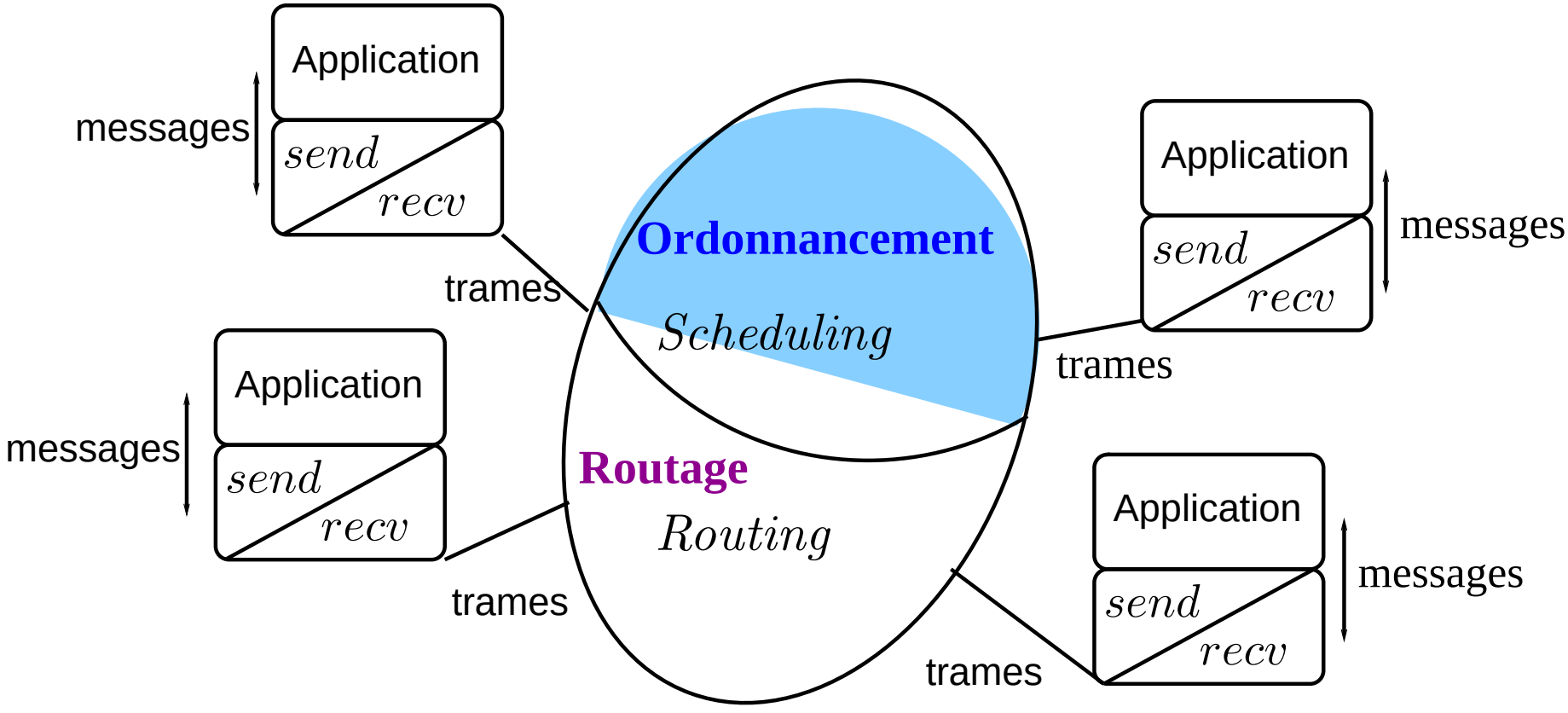


Toutes les routes = les routes vers la droite
+ les routes vers le bas

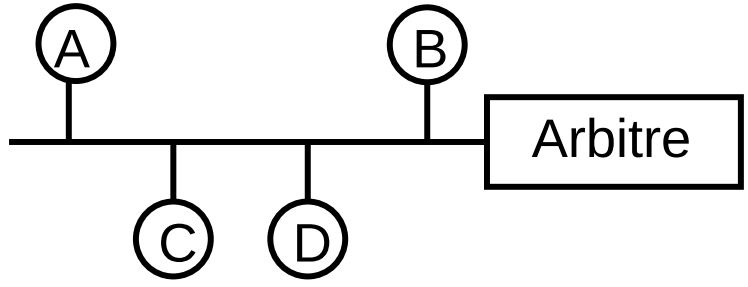
Routage : correction

- Terminaison du routage
  - Distance décroit à chaque “saut”
- Correction du routage
  - Route va de l’origine à la destination
  - Nœuds de la route existent

L'ordonnancement

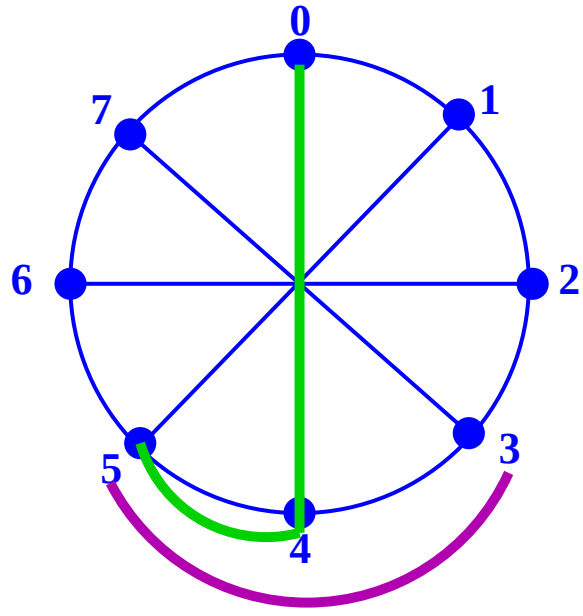


Arbitrage de bus



- Si A et B veulent émettre, l'arbitre tranche
- L'arbitre préserve l'exclusion mutuelle à tout instant

Commutation par circuits

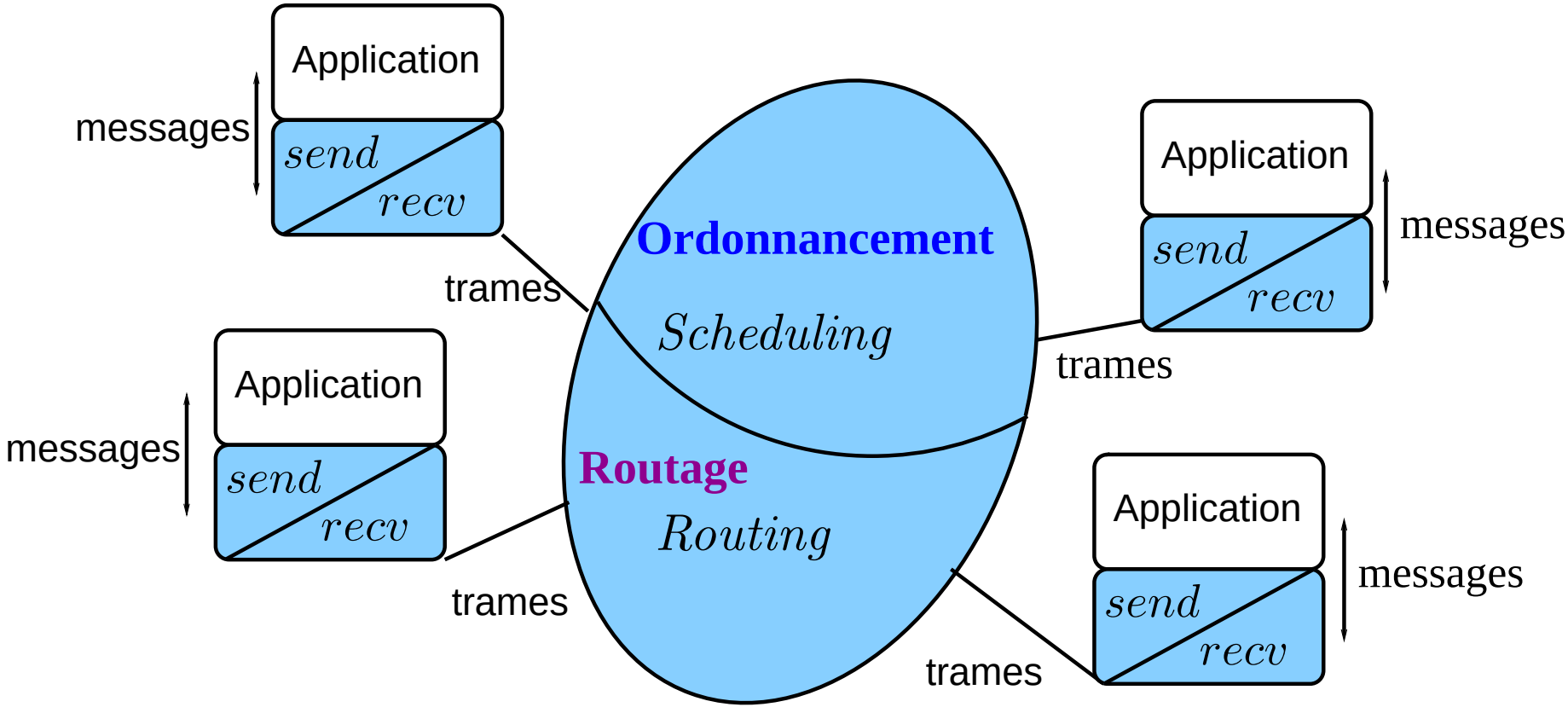


- Réservation des nœuds avant émission
- A tout instant, un nœud n'appartient qu'à un seul circuit

Fonction *Scheduling*

- Modélisation
  - Préservation d'un invariant
  - 2 instants : "maintenant" et "plus tard"

Le système

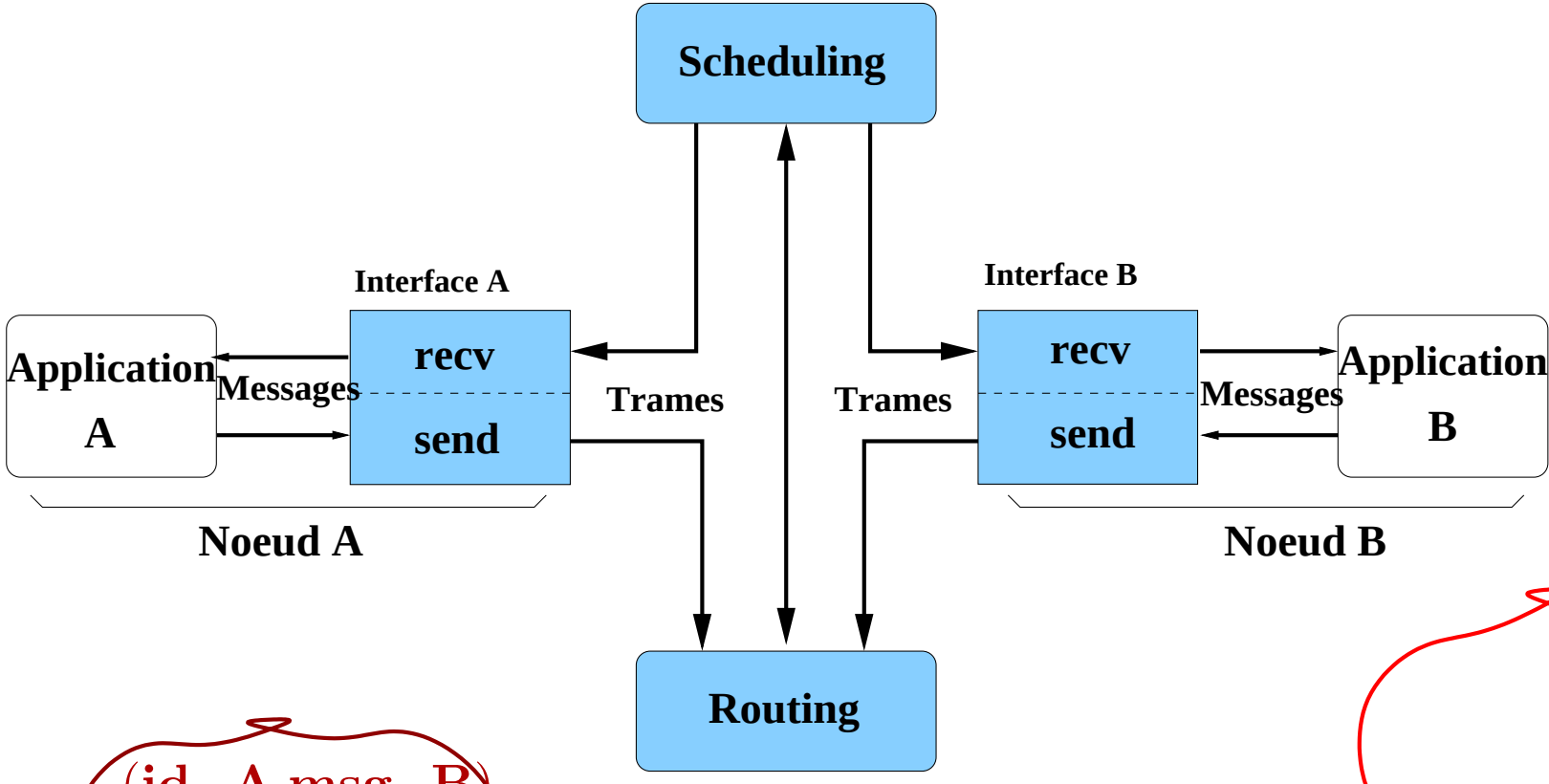


Système $= \mathcal{F}(Routing, Scheduling, recv, send)$

Modélisation de l'ensemble

- La fonction $GeNoC$
  - Prend la liste des communications en attente
  - Retourne la liste des résultats et celle des communications avortées
- Les transactions
  - Une transaction représente une communication en attente, soit l'intention de A d'envoyer msg à B
  - C'est un quadruplet $(id\ A\ msg\ B)$

La fonction $GeNoC$



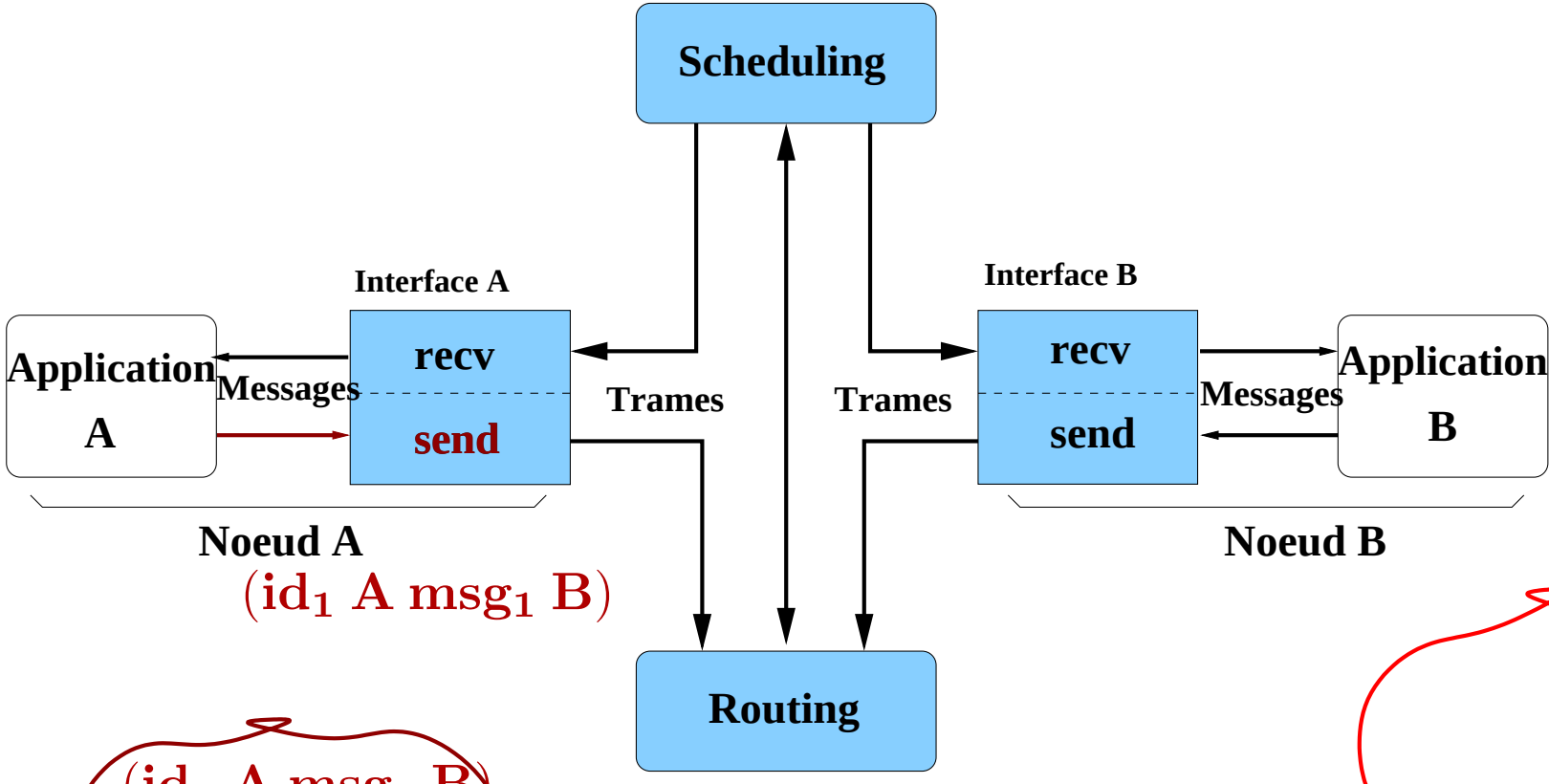
$(id_1\ A\ msg_1\ B)$
$(id_2\ D\ msg_2\ T)$
$(id_3\ F\ msg_3\ E)$
$(id_4\ R\ msg_4\ Z)$

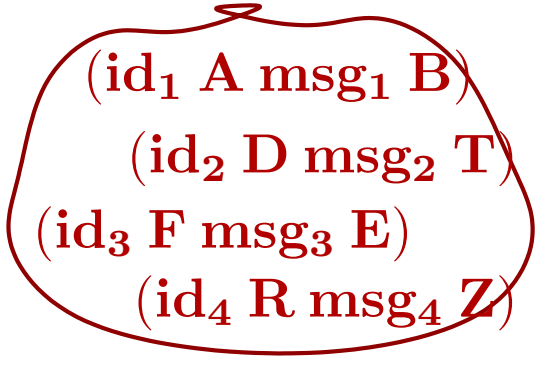
Transactions

Résultats

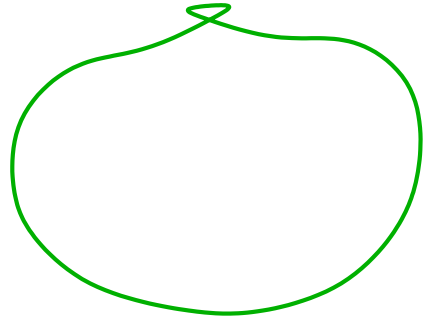
Missives Avortées

De la transaction à la missive

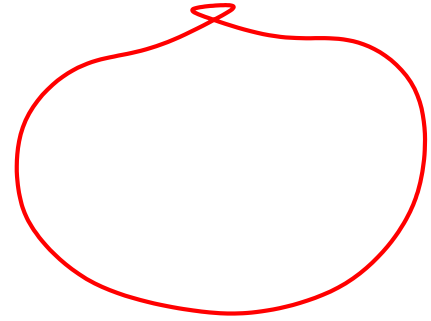




Transactions

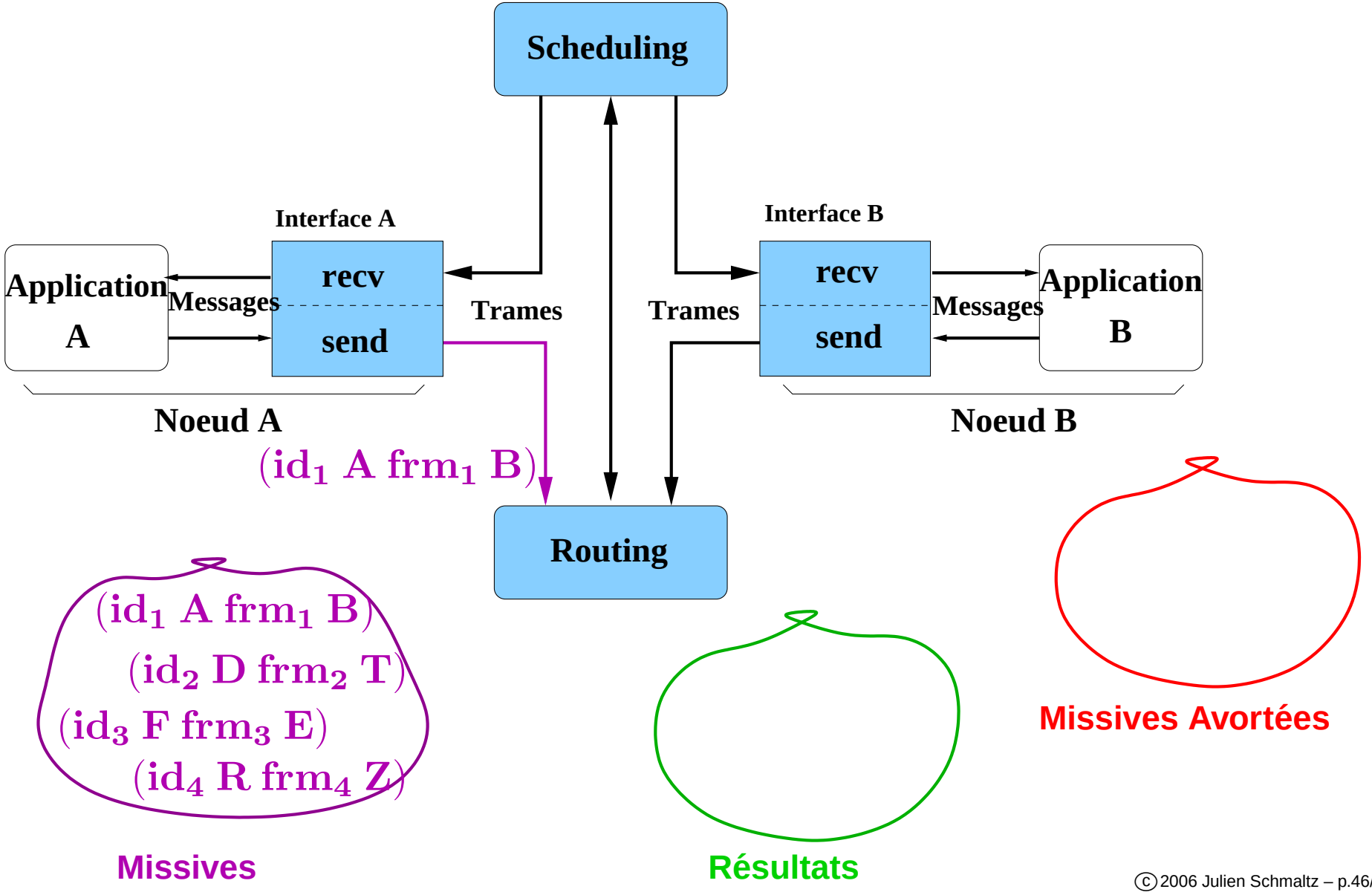


Résultats

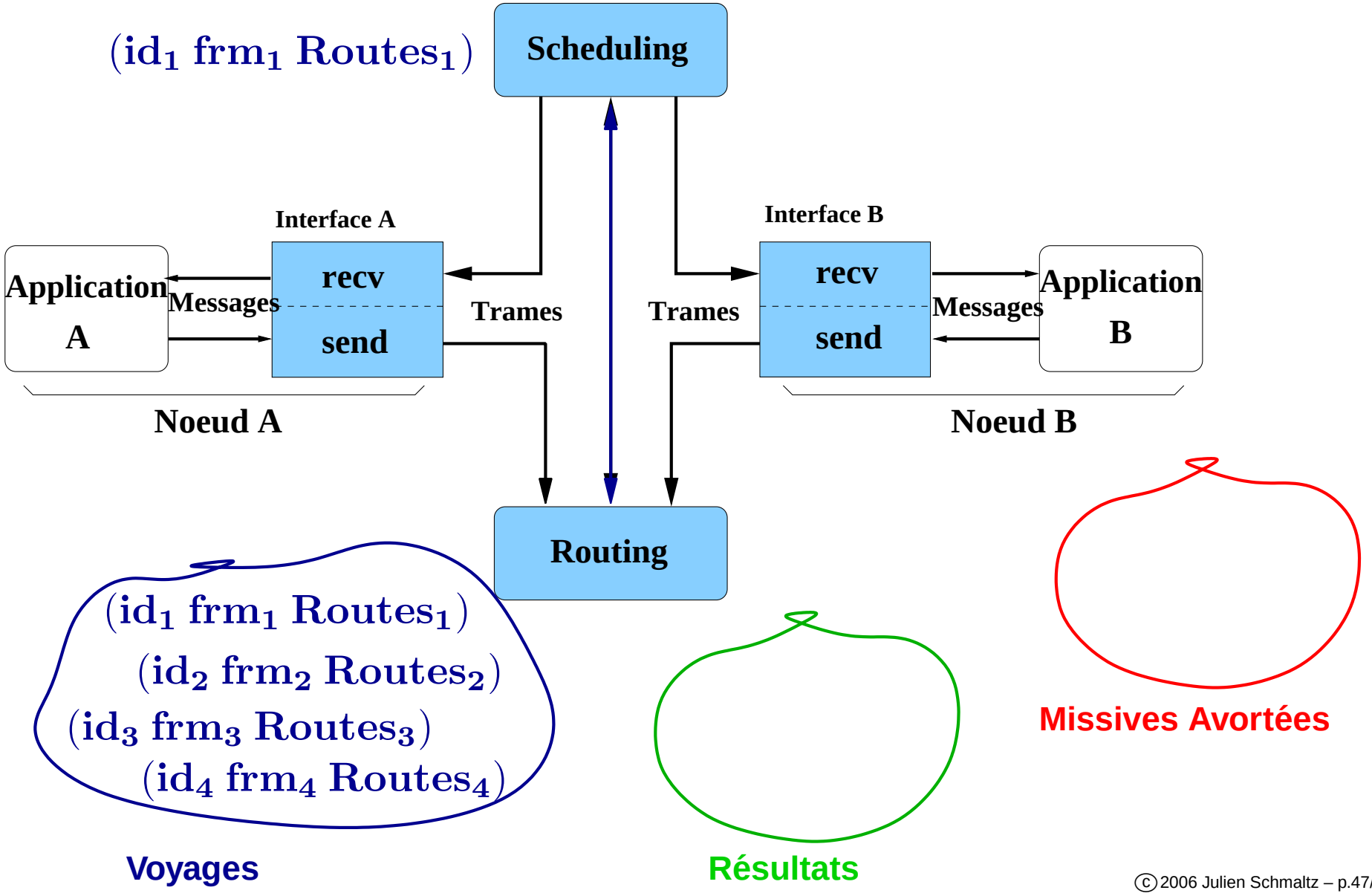


Missives Avortées

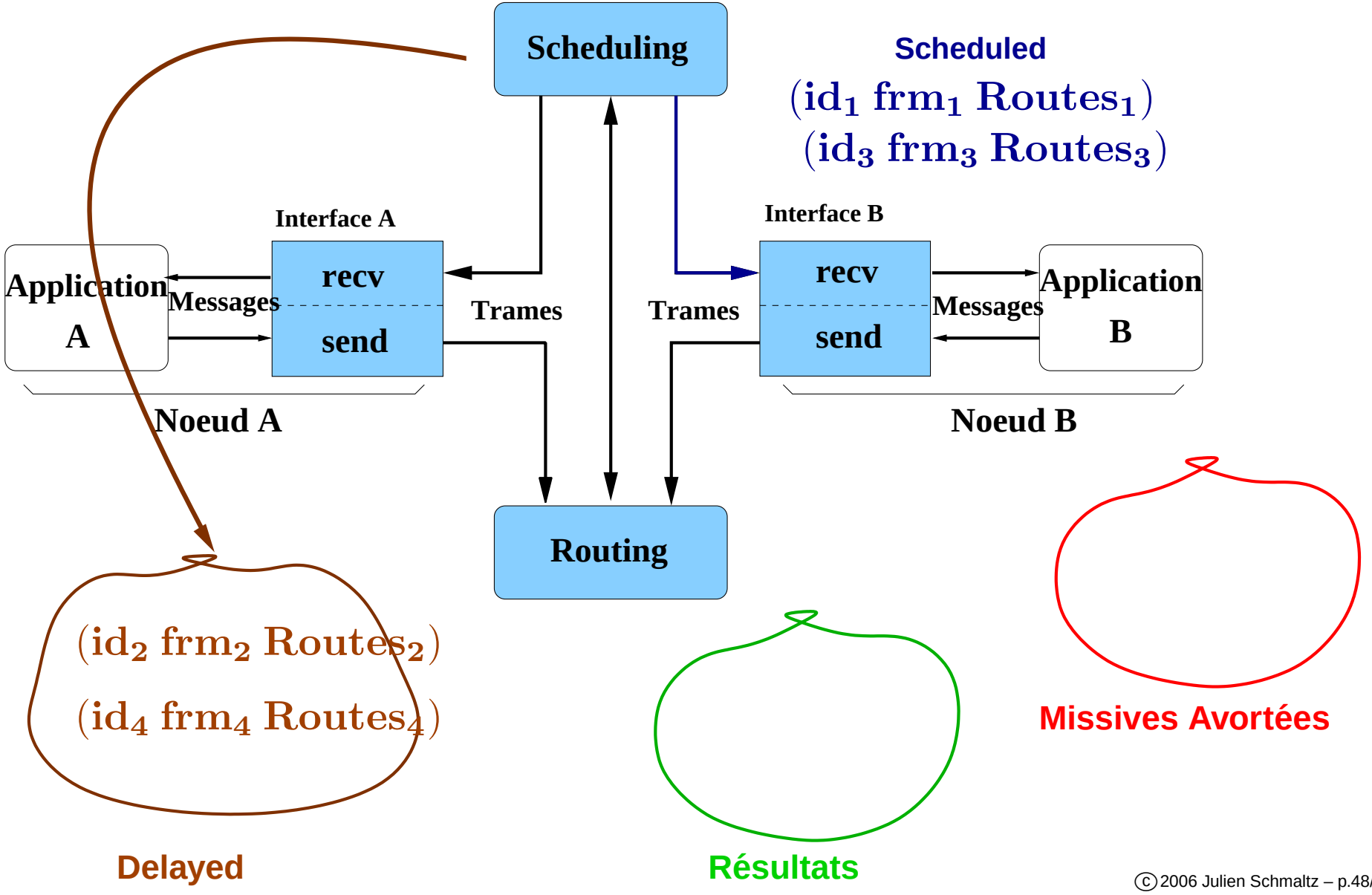
De la transaction à la missive



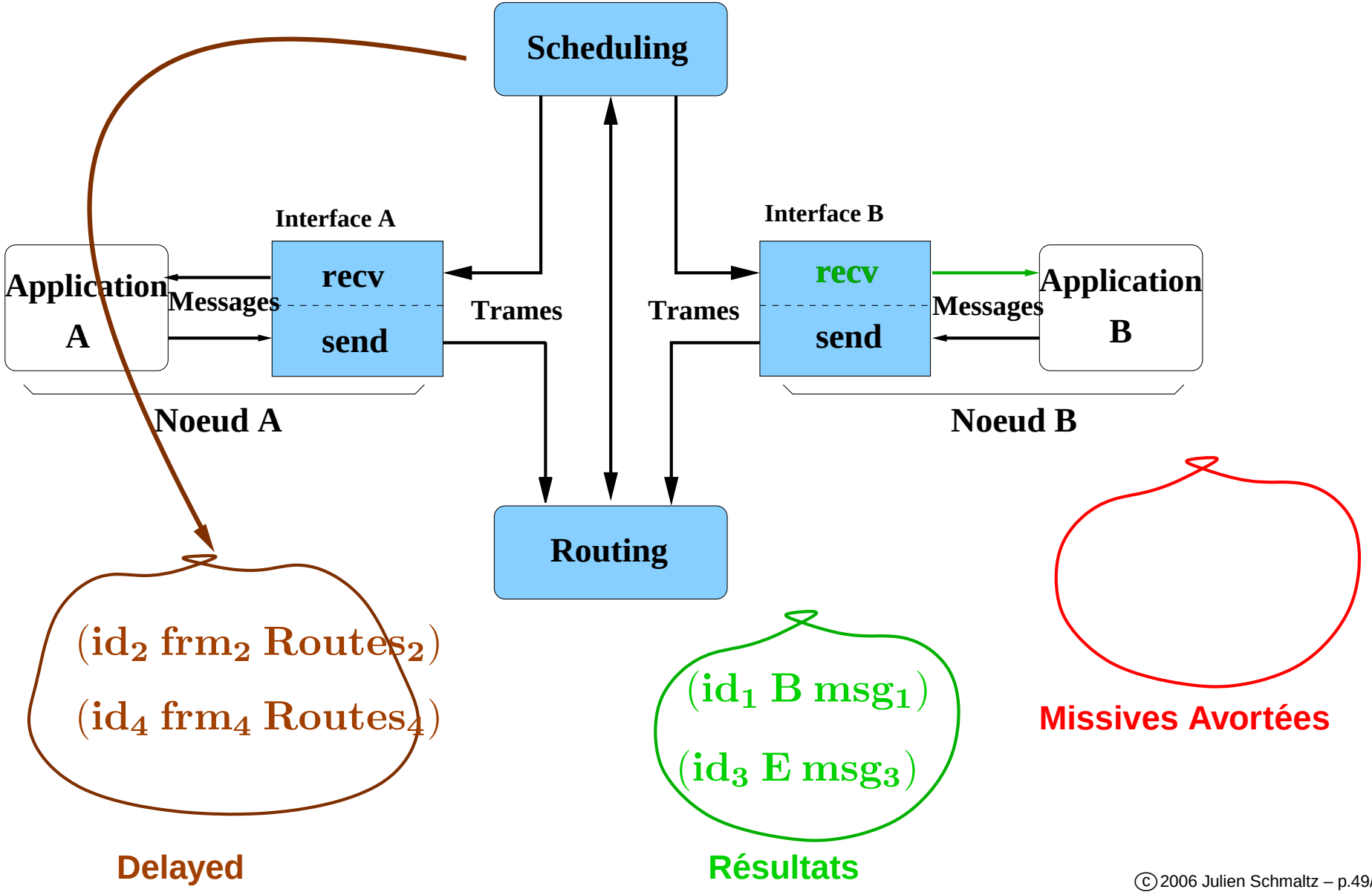
Routage


$(id_1\ frm_1\ Routes_1)$
Scheduling
Interface A
Interface B
Application A
Messages
recv
send
Trames
Trames
recv
send
Messages
Application B
Noeud A
Noeud B
Routing
$(id_1\ frm_1\ Routes_1)$
$(id_2\ frm_2\ Routes_2)$
$(id_3\ frm_3\ Routes_3)$
$(id_4\ frm_4\ Routes_4)$
Missives Avortées
Voyages
Résultats

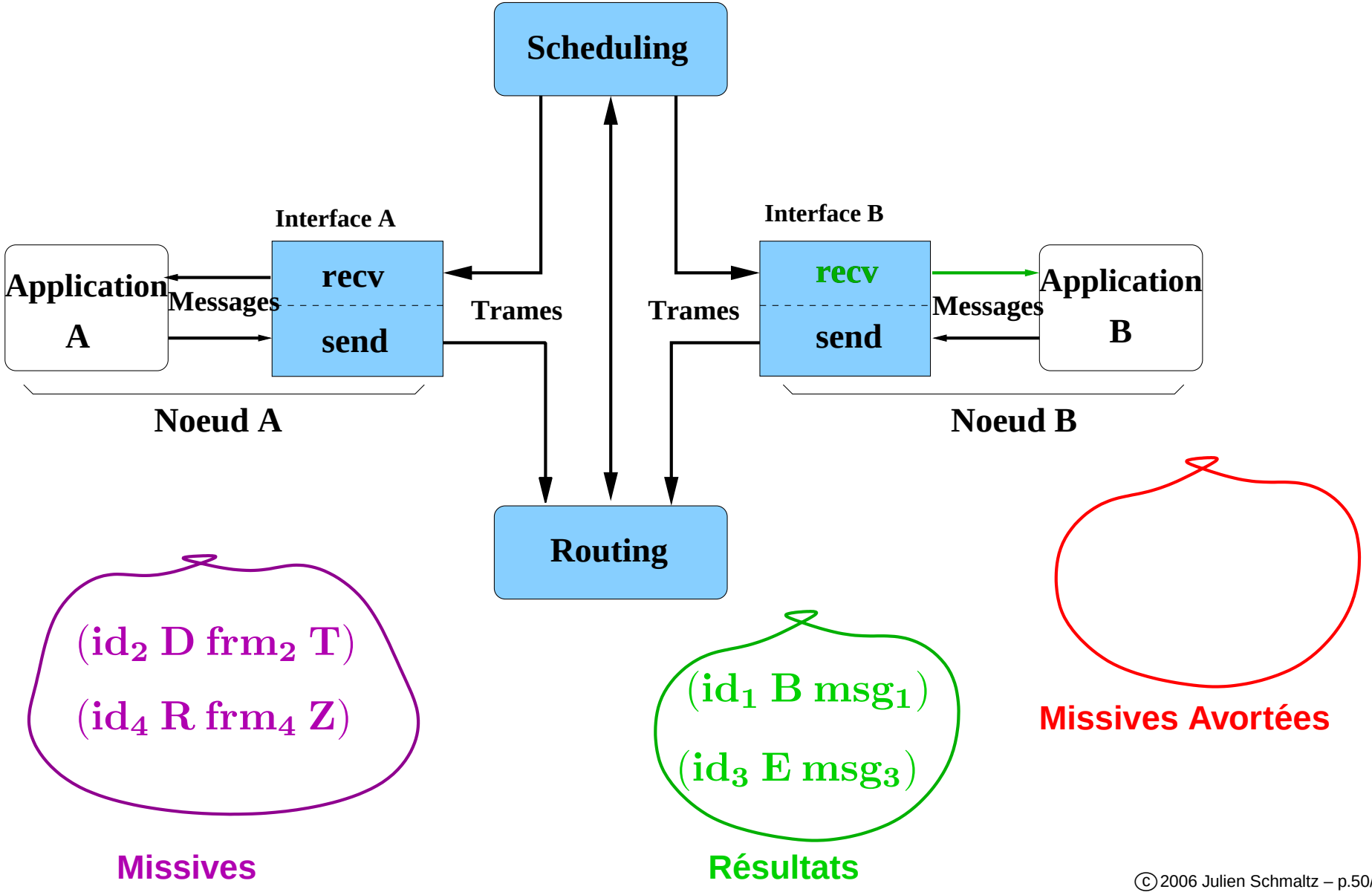
Ordonnancement



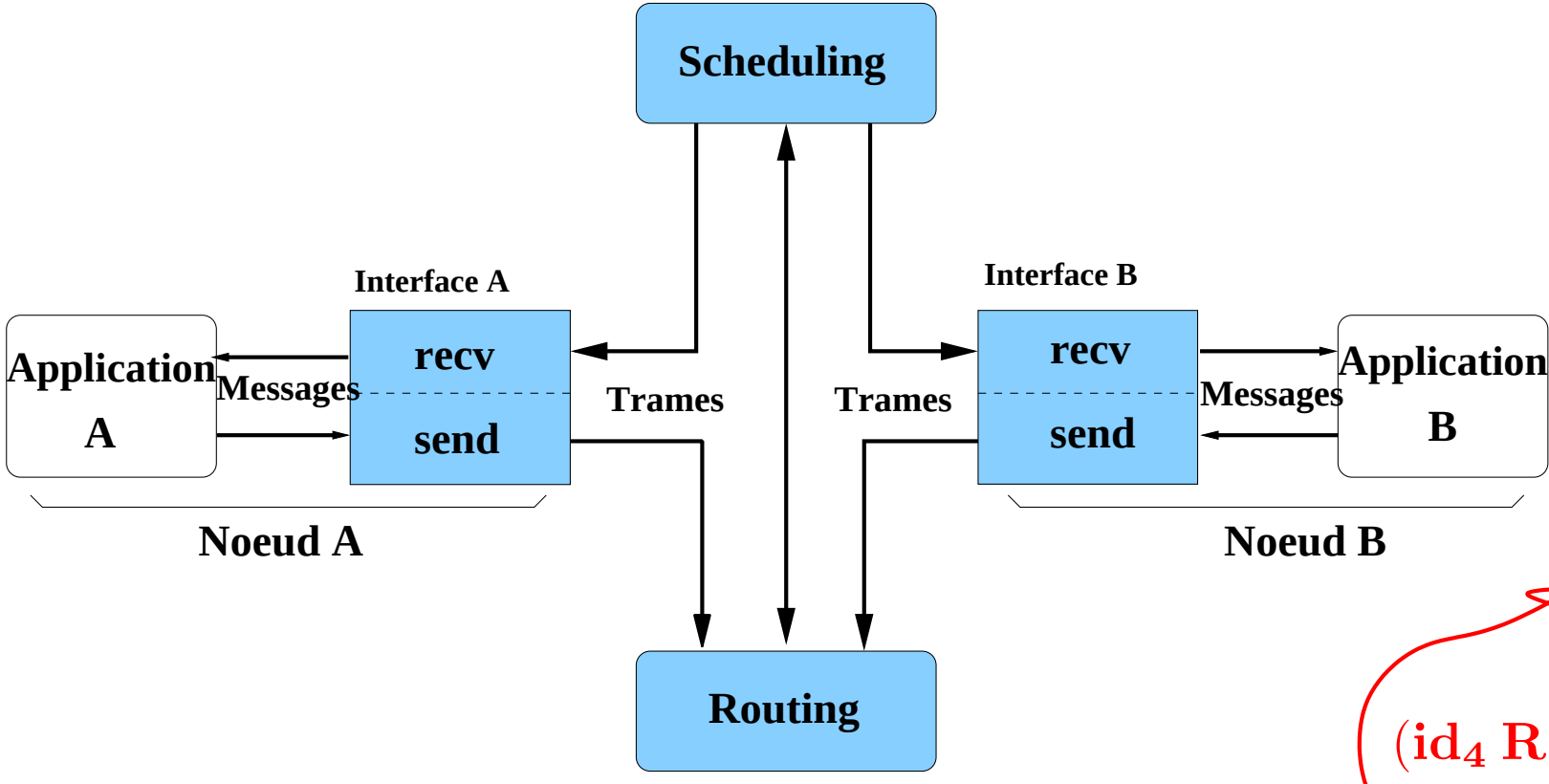
Résultats


Scheduling
Interface A
Interface B
Application A
Messages
recv
send
Trames
Trames
recv
send
Messages
Application B
Noeud A
Noeud B
Routing
(id2 frm2 Routes2)
(id4 frm4 Routes4)
(id1 B msg1)
(id3 E msg3)
Missives Avortées
Delayed
Résultats

Missives avortées



Missives avortées



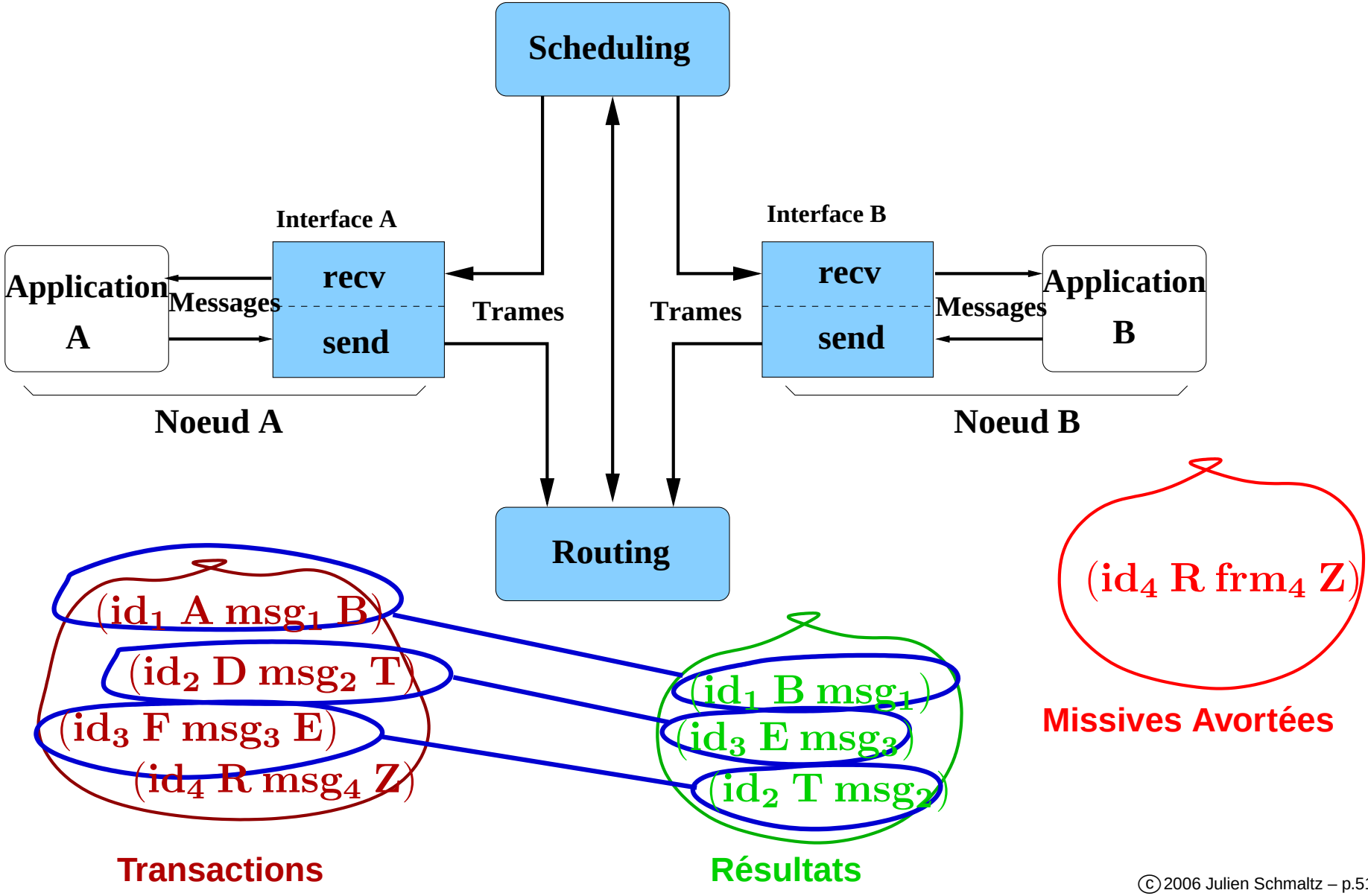
$(\mathrm{id}_4\ \mathrm{R}\ \mathrm{frm}_4\ \mathrm{Z})$

Missives Avortées

$(\mathrm{id}_1\ \mathrm{B}\ \mathrm{msg}_1)$
$(\mathrm{id}_3\ \mathrm{E}\ \mathrm{msg}_3)$
$(\mathrm{id}_2\ \mathrm{T}\ \mathrm{msg}_2)$

Résultats

Critère de correction



Terminaison

La fonction $GeNoC$ est récursive, on doit prouver l'arrêt des calculs :

- C'est un prérequis aux raisonnements automatiques
- C'est nécessaire pour assurer la vivacité du système

Pour assurer l'arrêt des appels récursifs, chaque nœud possède un nombre fini de tentatives pour effectuer les transactions en attente à ce nœud.

Définition formelle

À partir d'une liste de transactions, $\mathcal{T}$, de l'ensemble des nœuds du réseau $NodeSet$ et d'une liste de nombres de tentatives att, la fonction $GeNoC$ produit :

- La liste $\mathcal{R}$ des résultats
- La liste $\mathcal{A}$ des missives avortées

$$GeNoC : \mathcal{D}_{\mathcal{T}} \times GenNodeSet \times AttLst \rightarrow \mathcal{D}_{\mathcal{R}} \times \mathcal{D}_{\mathcal{T}}$$
$$(\mathcal{T}, NodeSet, att) \mapsto (\mathcal{R}, \mathcal{A})$$

Critère de correction

$$\forall res \in \mathcal{R},$$

$$\exists! trans \in \mathcal{T}, \begin{cases} & Id_{\mathcal{R}}(res) = Id_{\mathcal{T}}(trans) \\ \wedge & Msg_{\mathcal{R}}(res) = Msg_{\mathcal{T}}(trans) \\ \wedge & Dest_{\mathcal{R}}(res) = Dest_{\mathcal{T}}(trans) \end{cases}$$

Critère de correction

$$\forall res \in \mathcal{R},$$

$$\exists! trans \in \mathcal{T}, \begin{cases} & Id_{\mathcal{R}}(res) = Id_{\mathcal{T}}(trans) \\ \wedge & Msg_{\mathcal{R}}(res) = Msg_{\mathcal{T}}(trans) \\ \wedge & Dest_{\mathcal{R}}(res) = Dest_{\mathcal{T}}(trans) \end{cases}$$

En français !

Pour chaque résultat res, il existe une unique transaction $trans$ telle que $trans$ et res ont les mêmes identifiant, message, et destination.

Les obligations de preuve

- Interfaces
  - La composition $recv \circ send$ est une identité
- Routage $(id\ A\ frm\ B) \mapsto (id\ frm\ Routes)$
  - Correspondance missive/voyage
    - Même trame et même identifiant
    - Route va de l'origine à la destination
- Ordonnancement
  - Exclusion mutuelle entre $Scheduled$ et $Delayed$
  - Pas de nouveaux identifiants
  - Préservation des trames et de la correction des routes

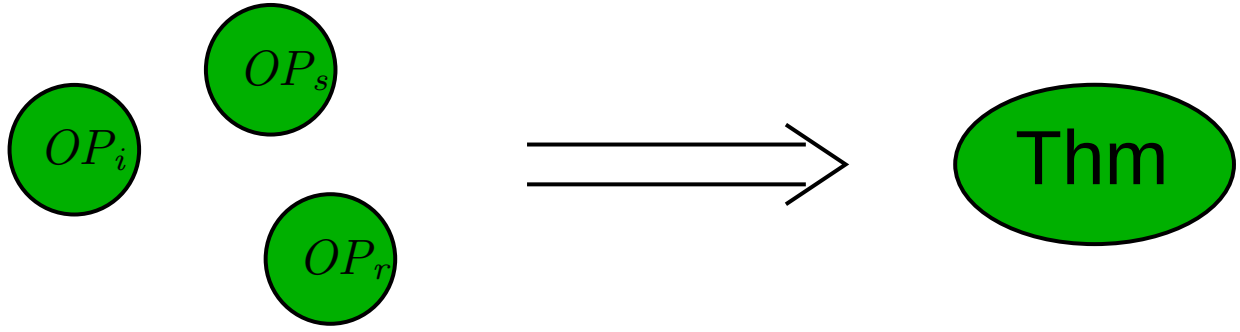
Preuve du théorème

- Correction du routage + préservation
  - $\longrightarrow$ bonne destination
- Non modification des trames
  - $\longrightarrow$ tout message est obtenu par la composition $recv \circ send$
- Correction des interfaces
  - $\longrightarrow$ message reçu = message émis
- Exclusion entre $Scheduled$ et $Delayed$ + pas de nouveaux identifiants
  - $\longrightarrow$ découpage de la preuve en deux

Plan

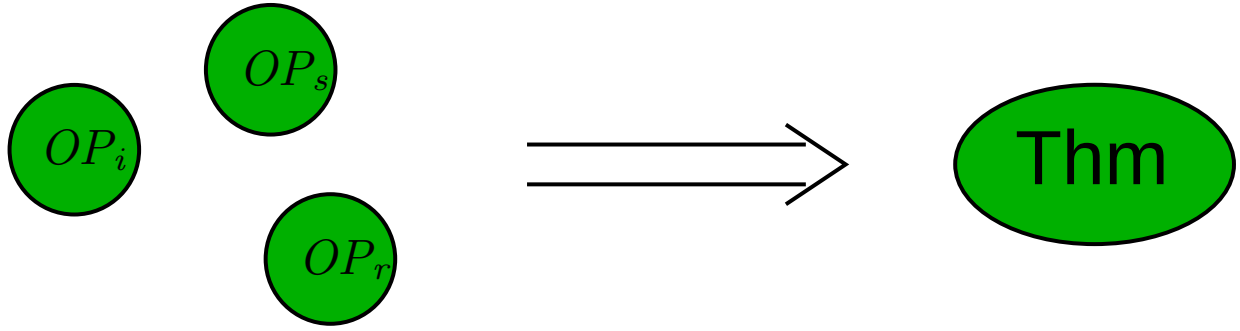
- Puces, réseaux et “bugs”
- Ma thèse
- Formalisation fonctionnelle : $GeNoC$
- Méthodologie

$GeNoC$ et logique (d'ordre 2)

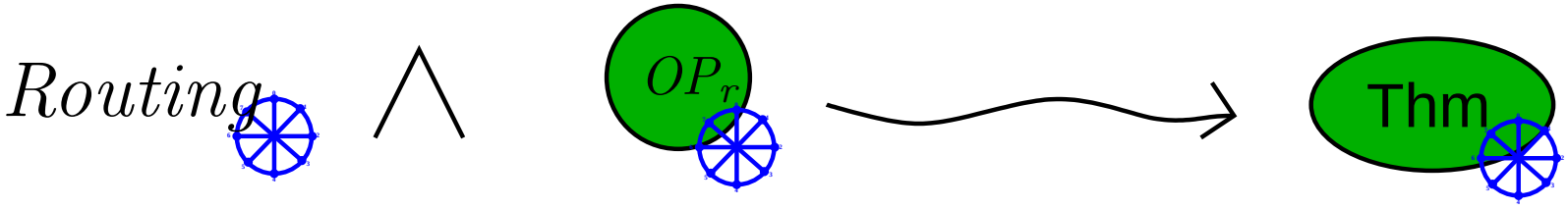


$$\forall recv, send, Routing, Scheduling$$

$GeNoC$ et logique (d'ordre 2)



Exemple :

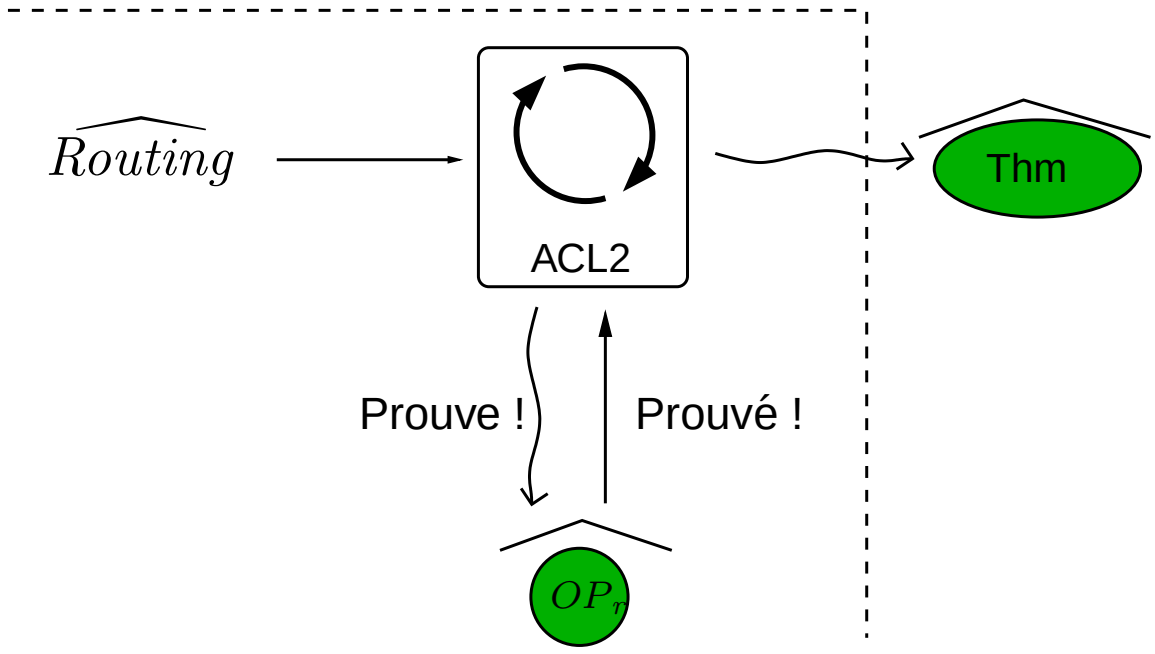


"Instanciation" fonctionnelle de l'Octagon

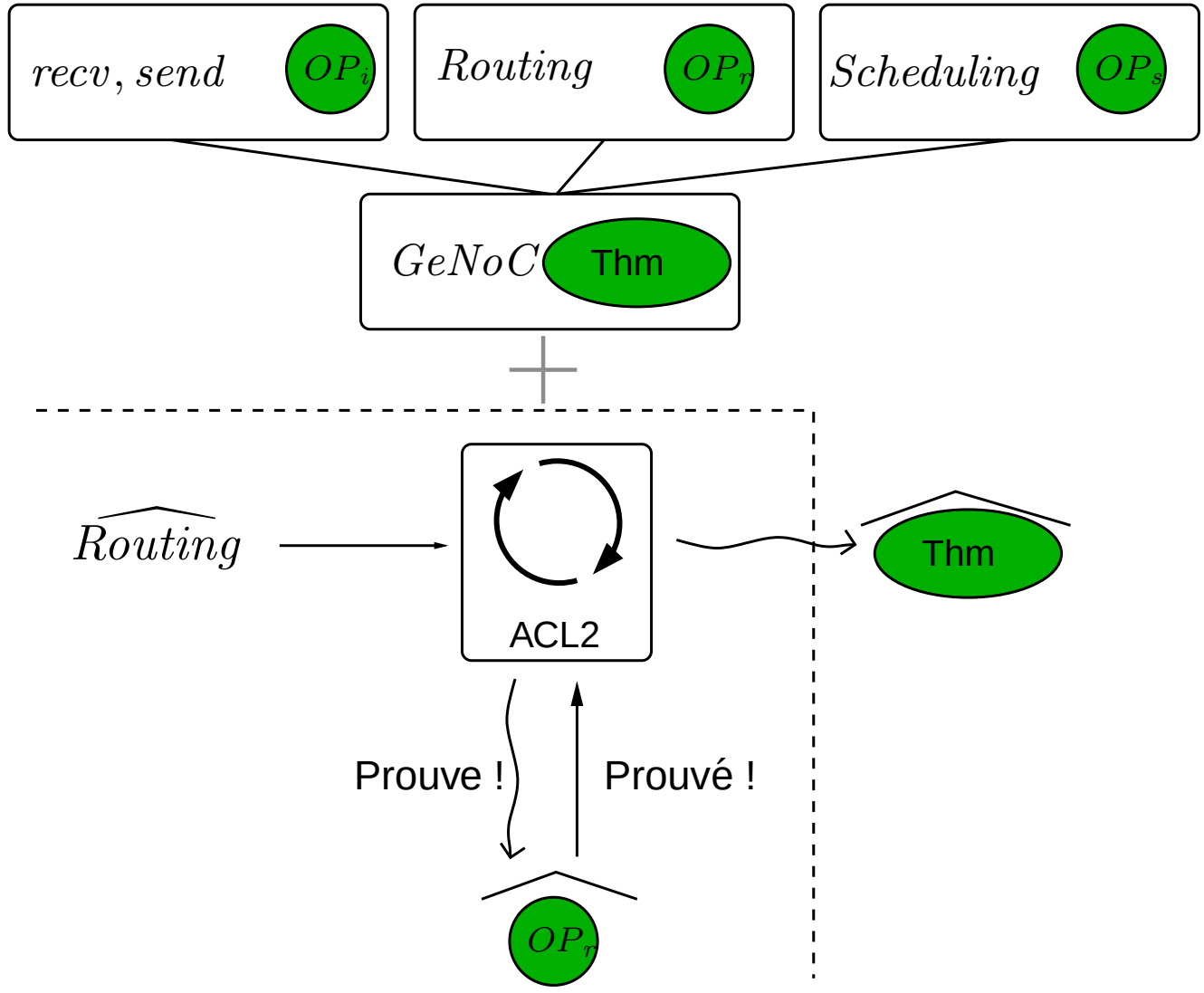
ACL2

- “A Computational Logic for Applicative Common LISP”
  - Langage de programmation (LISP)
    - Exécutable (vitesse $\sim$ programmes C)
  - Logique mathématique (1^{er} ordre)
  - Assistant de preuve (heuristiques)
- Applications matérielles et logicielles
  - Microprocesseurs (*e.g.* AMD)
  - JVM
  - Compilateurs, OS, . . .

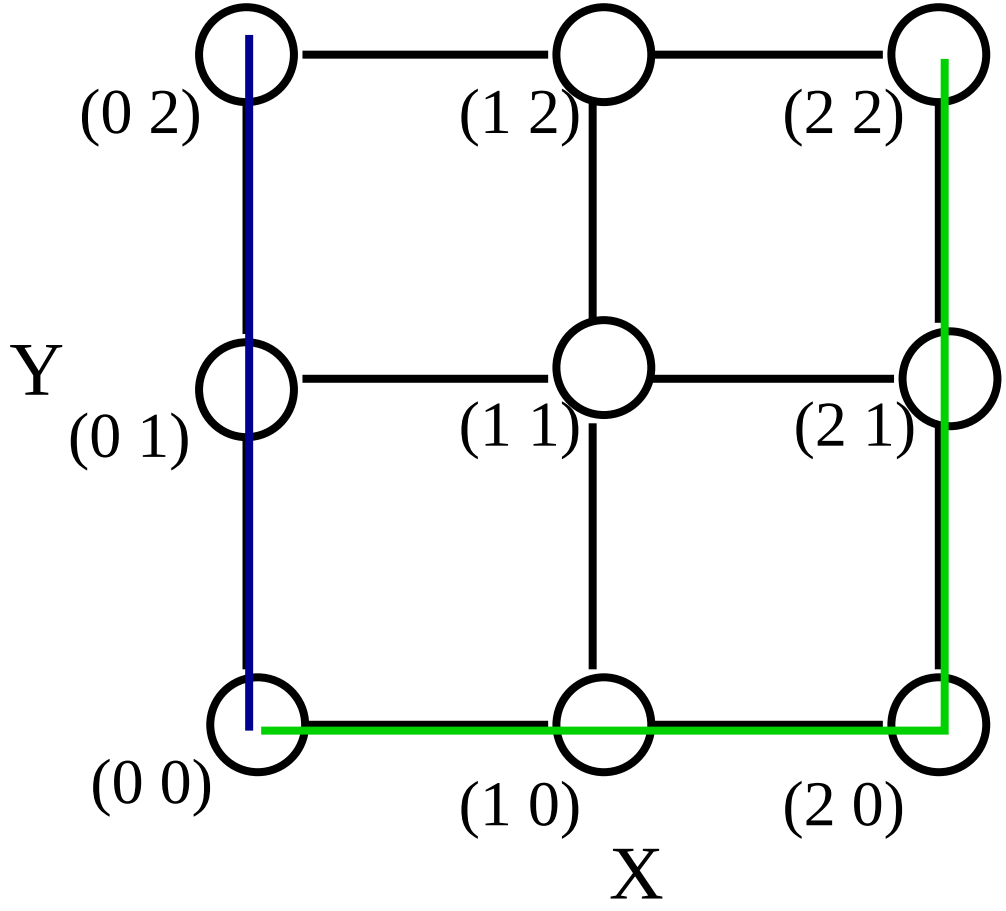
Méthode systématique



Méthode systématique



Routage en XY

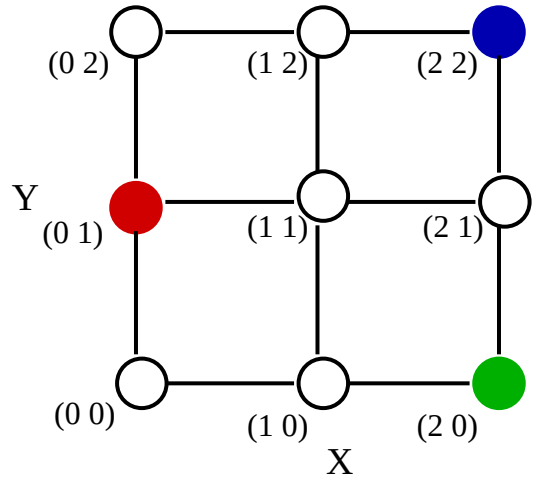


Déplacements selon l'axe X puis selon l'axe Y.

Définition du routage en XY

$$\mathcal{L}_{xy}(s,d) \triangleq$$

$$\begin{cases} d & \textbf{si} \quad s = d \\ (s_x + 1, s_y) & \textbf{si} \quad s_x < d_x \\ (s_x - 1, s_y) & \textbf{si} \quad s_x > d_x \\ (s_x, s_y + 1) & \textbf{si} \quad s_x = d_x \wedge \ s_y < d_y \\ (s_x, s_y - 1) & \textbf{si} \quad s_x = d_x \wedge s_y > d_y \end{cases}$$



- Distance

$$dist_{xy}(s,d) = |d_x - s_x| + |d_y - s_y|$$

Preuve du routage en XY

- La distance décroit
- Validité des routes
  - Même trame et même identifiant
  - Route va de l'origine à la destination
  - Nœuds de la route existent

Seule la preuve que les nœuds des routes existent nécessite de guider ACL2.
14 fonctions, 49 théorèmes, 615 lignes de code, temps de preuve $\sim$ 25 secondes

Preuve de l'Octagon

- La distance décroit
- Validité des routes
  - Même trame et même identifiant
  - Route va de l'origine à la destination
  - Nœuds de la route existent

Difficulté liée à l'arithmétique.
21 fonctions, 64 théorèmes, 1325 lignes de code, temps de preuve $<$740 secondes

Conclusions

- Modèle formel et générique
- Réalisation du modèle dans ACL2
- Concrétisations variées pour validation

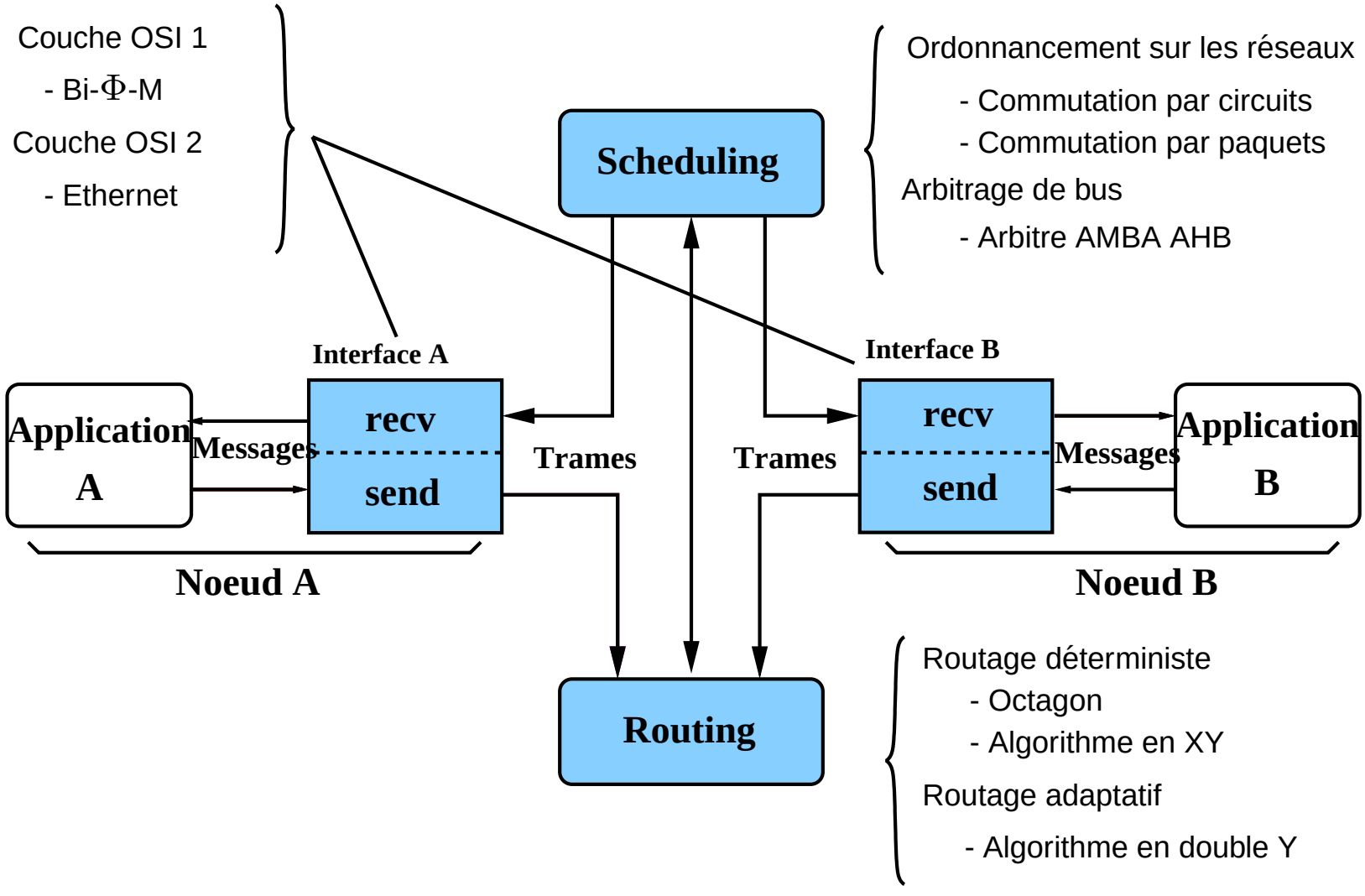
Modèle générique : $GeNoC$

- Identifie les composantes essentielles de **toute** architecture de communication
- Exprime leurs propriétés essentielles
- Formalise la propriété globale satisfaite par l'interaction des composantes comme une conséquence des propriétés des composantes

Réalisation du modèle dans ACL2

- 1864 lignes de code, 71 fonctions et 119 théorèmes
  - Preuve générale : 62 fonctions, 77 théorèmes, 1357 lignes de code
  - Les modules : 9 fonctions, 42 théorèmes, 507 lignes de code
- Génération automatique des obligations de preuve

Validation de $GeNoC$



Perspectives

- Extensions de $GeNoC$
- Vérification persistante des systèmes

Extensions de $GeNoC$

$GeNoC$ n'est qu'une étape ...

- Structures maître/esclave
- Ajout des files et des canaux
- Algorithmes non minimaux
- Interblocages
- Notion explicite du temps
- ...

Objectif : se rapprocher du RTL

Stage M2R (A. Helmy)

Vérification persistante

… vers un but nécessaire.

Un théorème pour **un** système complet, soit la preuve :

- Des applications
- Des microprocesseurs
- Des compilateurs
- Des systèmes d'exploitation
- Réseau d'interconnexion

Post-doc, projet Verisoft, Université du Saarland

Remerciements

- Université du Texas à Austin, W.A. Hunt
- Région Rhône-Alpes, bourse EURODOC