Vérification algorithmique
pour le développement de systèmes fiables

Michael Blondin

Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```

↓

Vérification

Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```

Vérification

Le logiciel fait-il ce à quoi on s'attend ?

Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```

Vérification

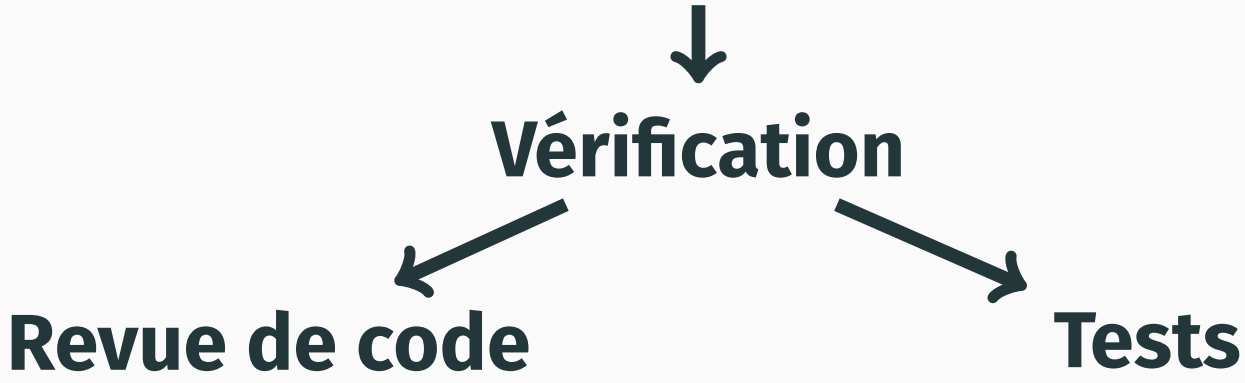
30% à 50% du temps de développement

Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```

Introduction

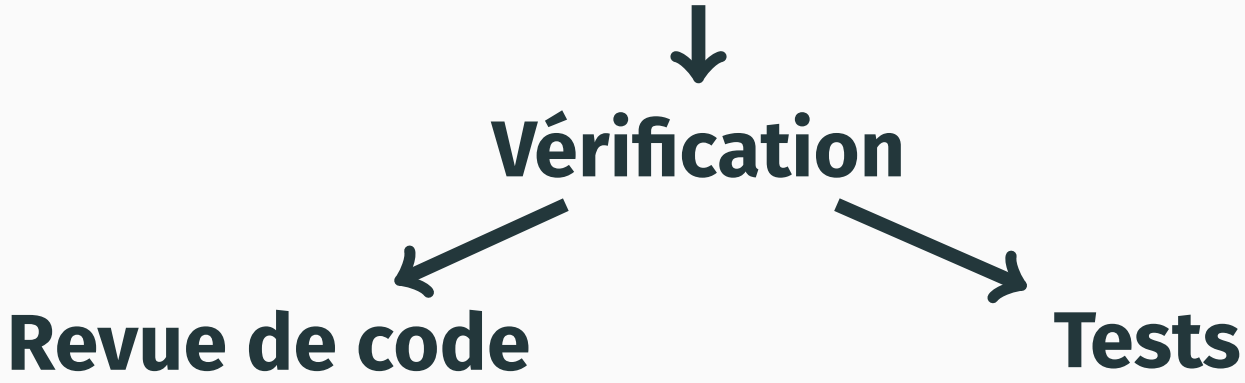
```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```



Corrige la majorité des bogues...

Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```



Corrige la majorité des bogues…

Mais ne peut pas montrer leur absence

et trouve les bogues « prévisibles par un humain »

Introduction

Difficile d’anticiper les erreurs des systèmes concurrents

Introduction

Difficile d'anticiper les erreurs des systèmes concurrents

Mars Pathfinder

Mission compromise (1997)

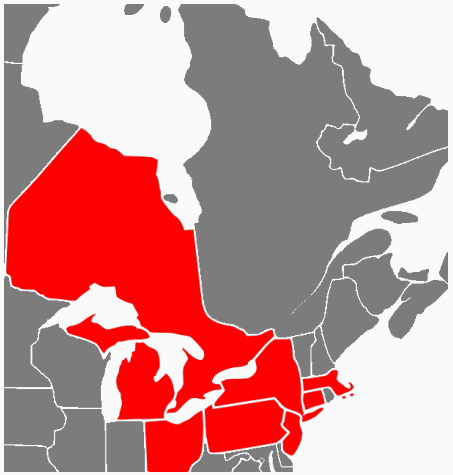
Introduction

Difficile d'anticiper les erreurs des systèmes concurrents

Mars Pathfinder

Mission compromise (1997)

Réseau électrique



Panne généralisée (2003)
10 millions en Ontario
45 millions aux É.-U.

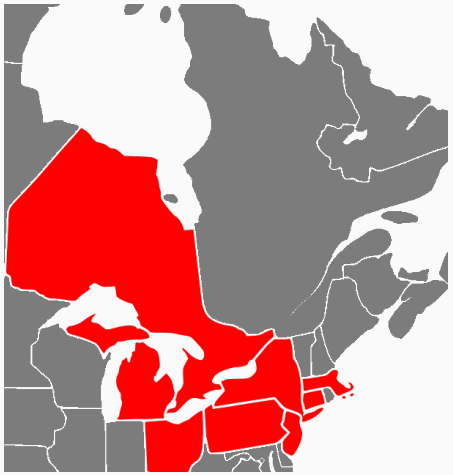
Introduction

Difficile d'anticiper les erreurs des systèmes concurrents

Mars Pathfinder

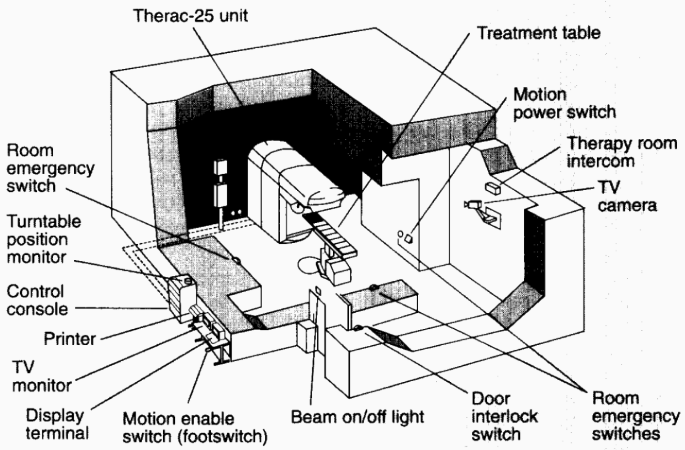
Mission compromise (1997)

Réseau électrique



Panne généralisée (2003)
10 millions en Ontario
45 millions aux É.-U.

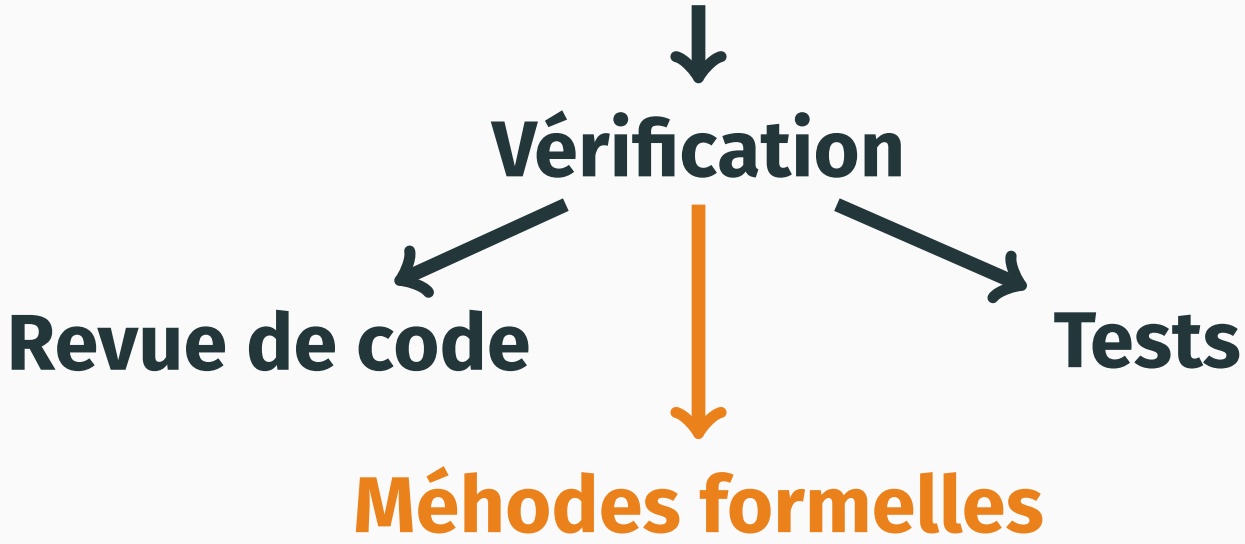
Therac-25 (radiothérapie)



Décès/blessures graves (1985-87)
6 personnes

Introduction

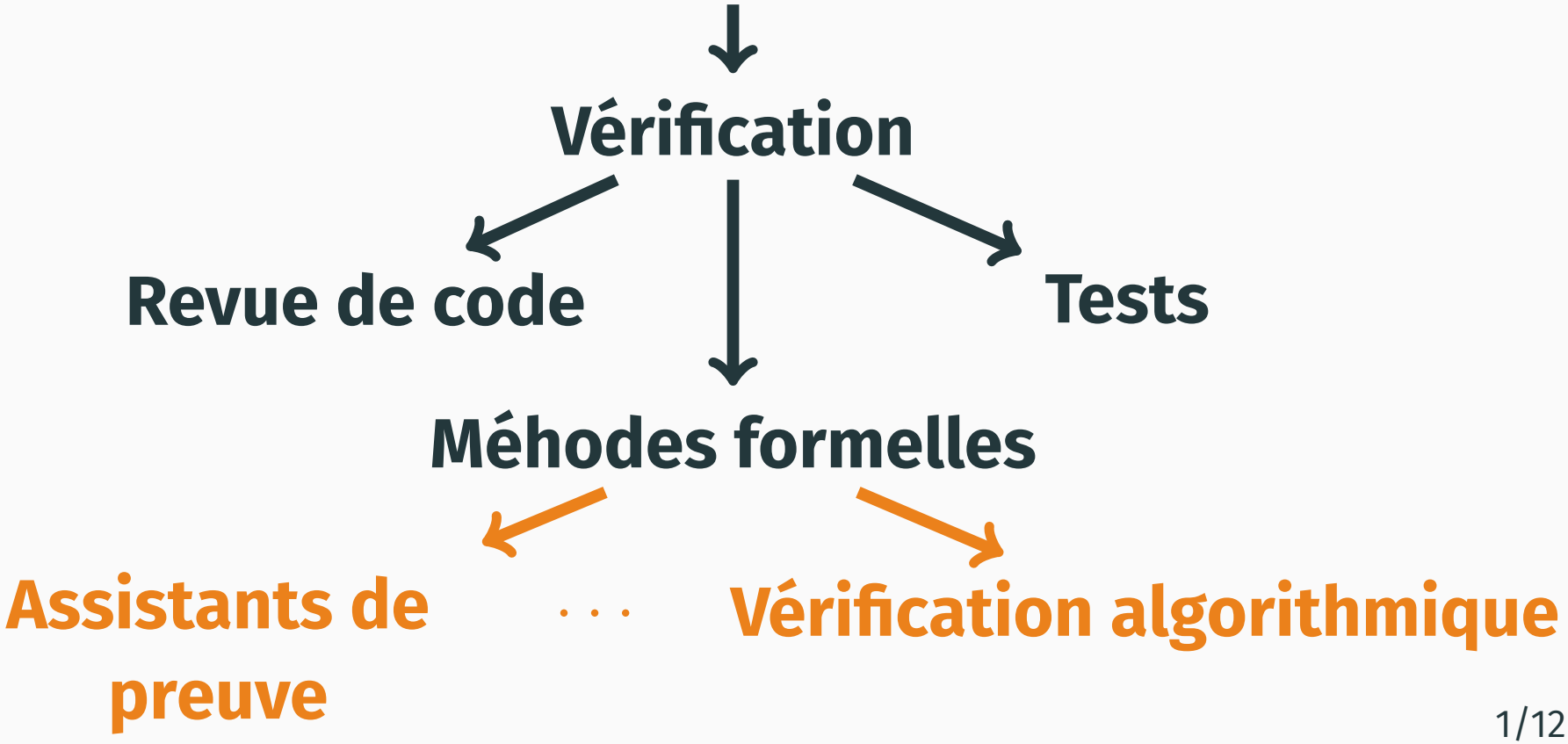
```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```



Rigueur des mathématiques pour prouver correction de systèmes

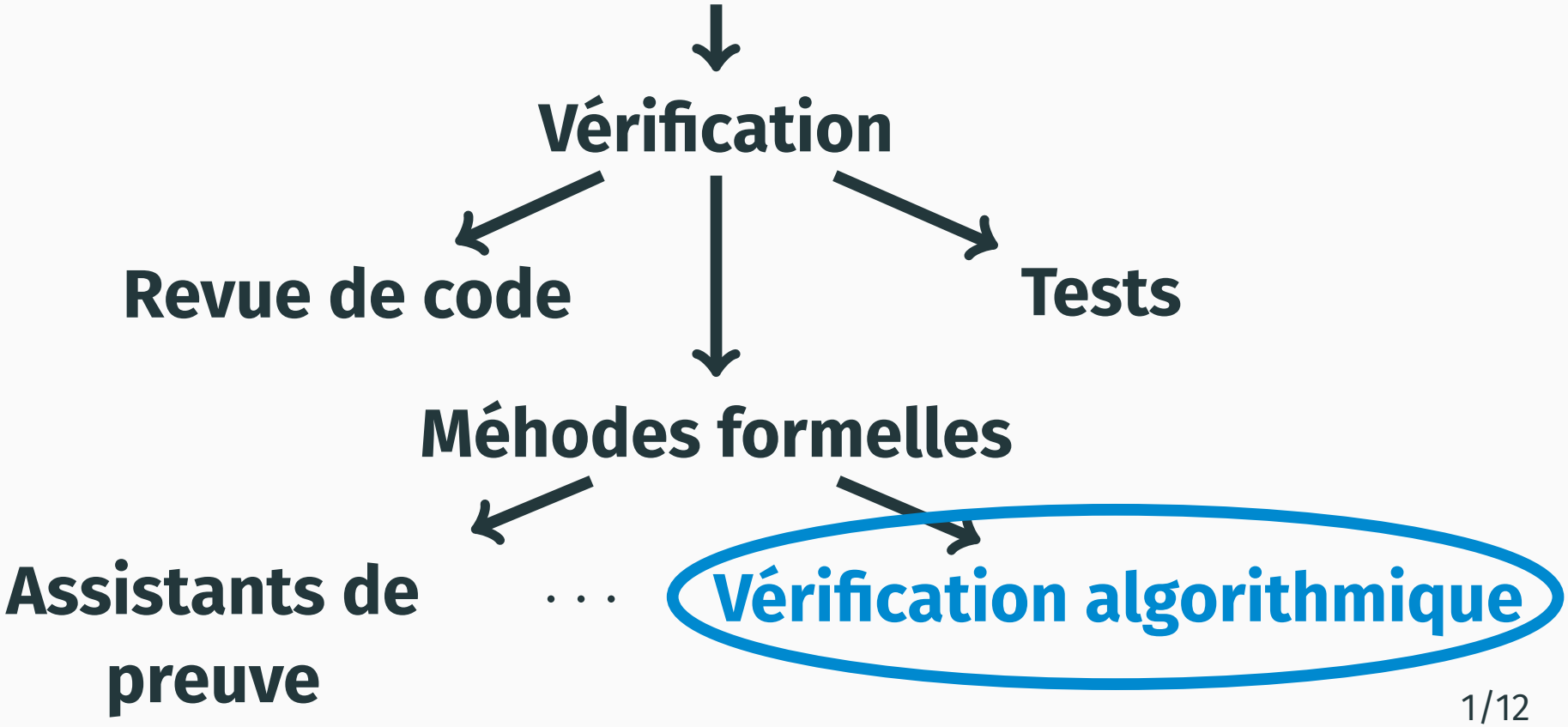
Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```



Introduction

```
for (i = 0; i < 12; i++) {
  uint32_t keyval;
  char buf[3];
  if (OF_getprop(handle, key_names[i],
                 &keyval, sizeof(keyval)) < 0)
    continue;
  // ...
```



Plan

1. **Survol de la vérification algorithmique**
   - **Introduction**
   - **Systèmes finis**
   - **Systèmes infinis**
   - **Systèmes temporisés, probabilistes, etc.**
2. **Avancées récentes en vérification de réseaux dynamiques**
3. **Un peu de publicité !**

1. Vérification algorithmique: un survol

Vérification algorithmique

Vérifier **automatiquement**
qu'un système se comporte comme attendu


Clarke


Emerson


Sifakis

Prix Turing 2007


Pnueli

1996

Vérification algorithmique

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire

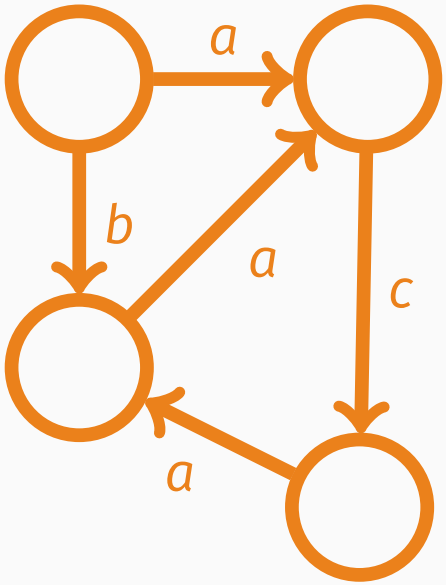
Vérification algorithmique

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire

Système satisfait **spécification ?**

Vérification algorithmique

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire

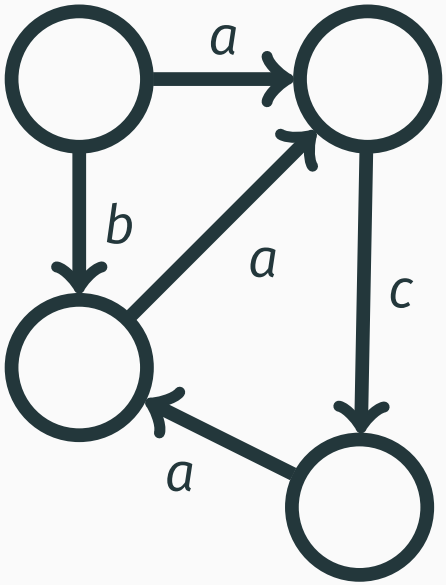


Modélisation

satisfait **spécification ?**

Vérification algorithmique

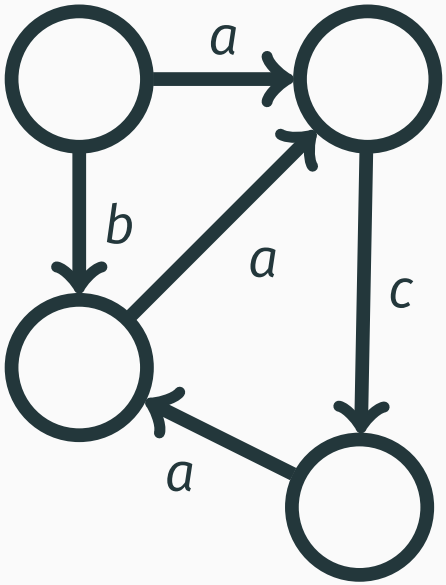
- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire



$\models \varphi$?

Propriété logique

Déterminer à l'aide d'un **algorithme**:



$\models \quad \varphi \quad ?$

Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

Possible d'atteindre les deux sections critiques simultanément ?

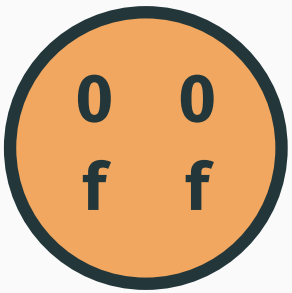
Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```



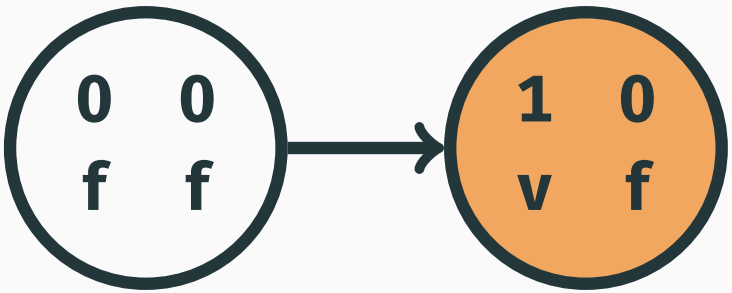
Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```



Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

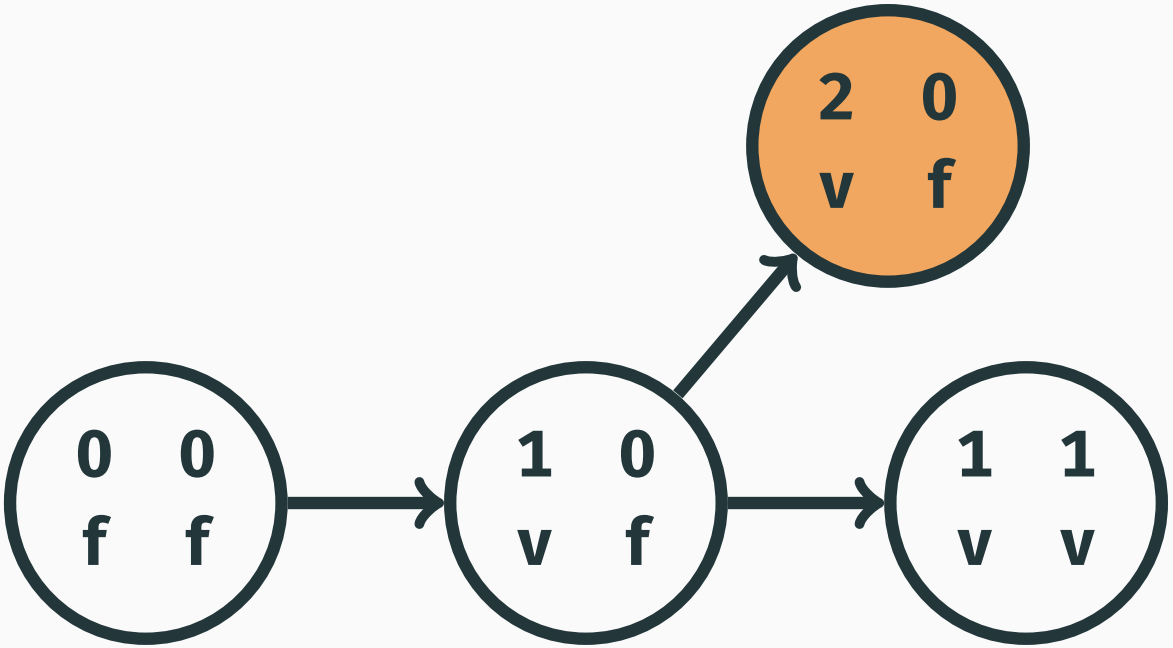
Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```



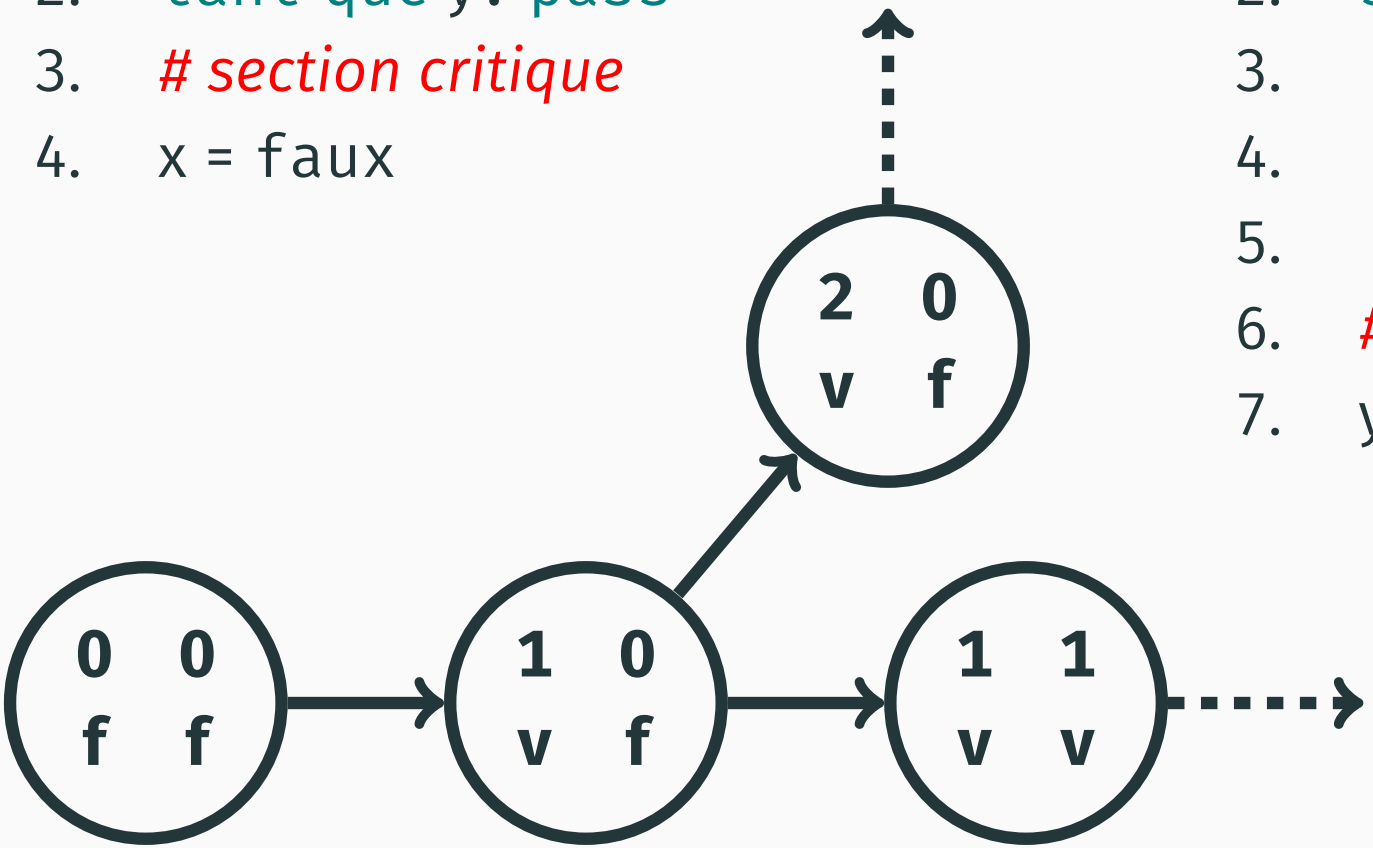
Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.    x = vrai
2.    tant que y: pass
3.    # section critique
4.    x = faux
```

Processus B

```
0. tant que vrai:
1.    y = vrai
2.    si x alors:
3.       y = faux
4.       tant que x: pass
5.       goto 1
6.    # section critique
7.    y = faux
```



Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

non accessible ?

Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

Processus B peut toujours atteindre sa section critique ?

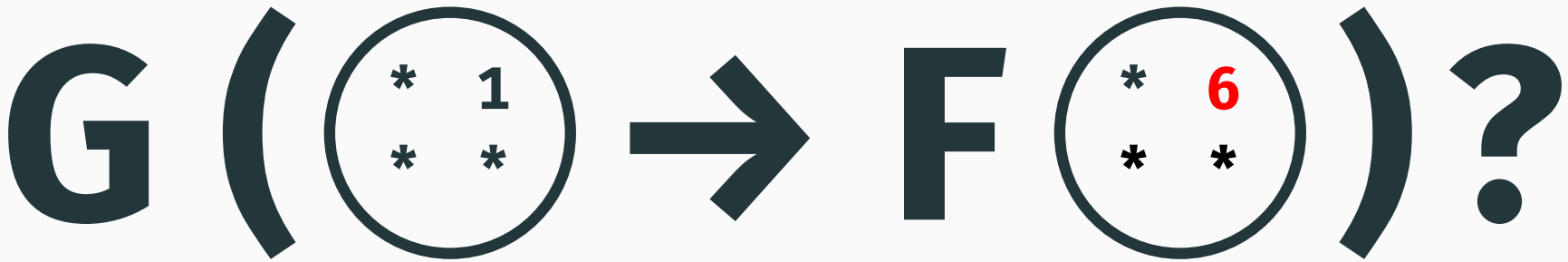
Vérification de systèmes finis

Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```



Vérification de systèmes finis

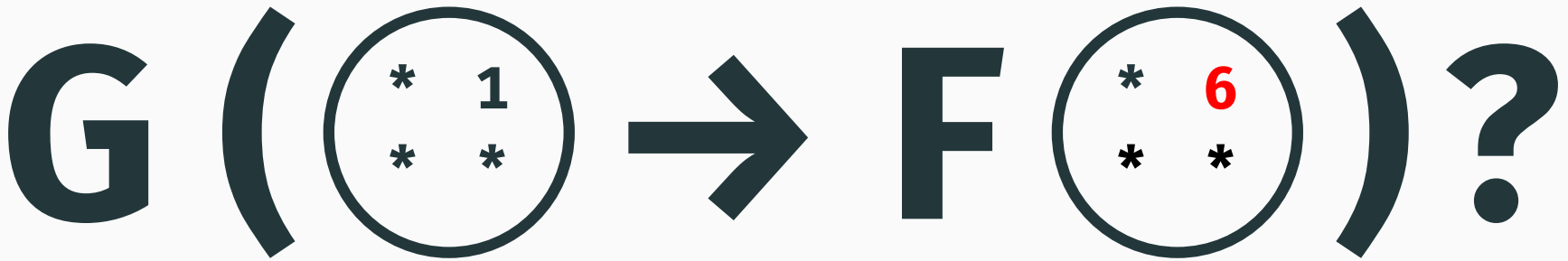
Processus A

```
0. tant que vrai:
1.   x = vrai
2.   tant que y: pass
3.   # section critique
4.   x = faux
```

Processus B

```
0. tant que vrai:
1.   y = vrai
2.   si x alors:
3.     y = faux
4.     tant que x: pass
5.     goto 1
6.   # section critique
7.   y = faux
```

Non...



Vérification de systèmes finis: approche algorithmique

Processus A

```
0. tant que vrai:
1.     x = vrai
2.     tant que y: pass
3.     # section critique
4.     x = faux
```

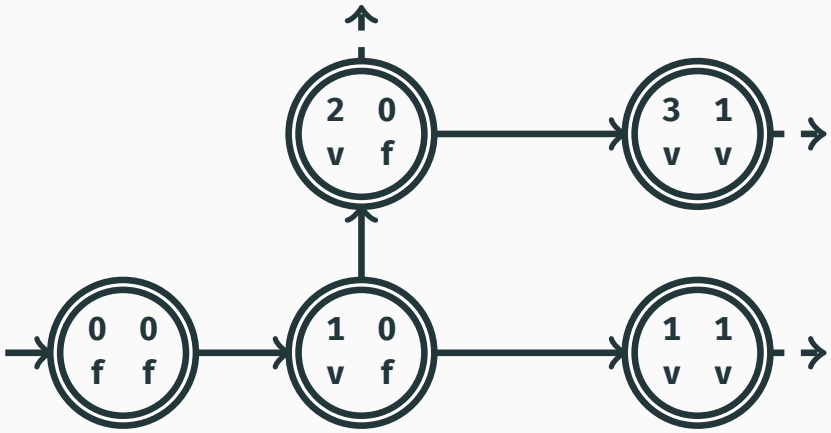
Processus B

```
0. tant que vrai:
1.     y = vrai
2.     si x alors:
3.         y = faux
4.         tant que x: pass
5.         goto 1
6.     # section critique
7.     y = faux
```

$\models$

Processus B peut toujours atteindre sa section critique

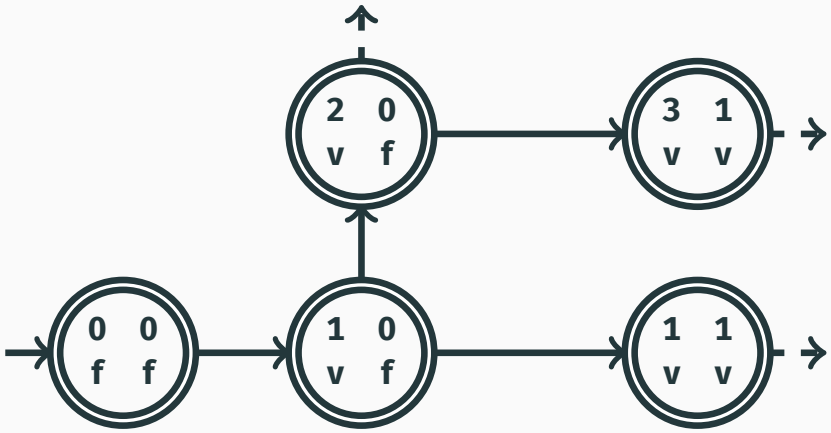
Vérification de systèmes finis: approche algorithmique



$\models$

Processus B peut toujours atteindre sa section critique

Vérification de systèmes finis: approche algorithmique

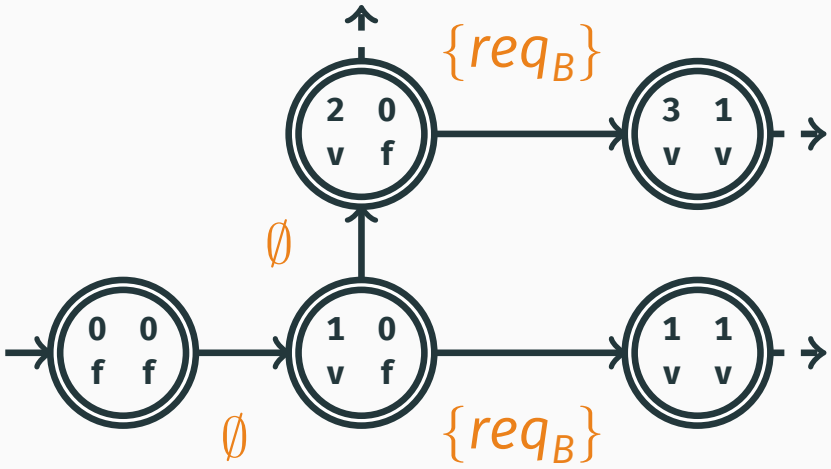


$\models$

Processus B peut toujours atteindre sa section critique

$$P = \{req_B, crit_B\}$$

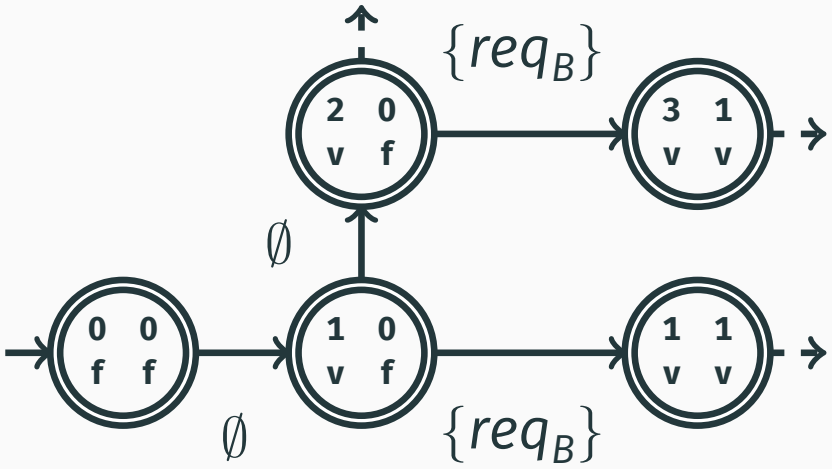
Vérification de systèmes finis: approche algorithmique



$\models$ Processus B peut toujours atteindre sa section critique

$$P = \{req_B, crit_B\}$$

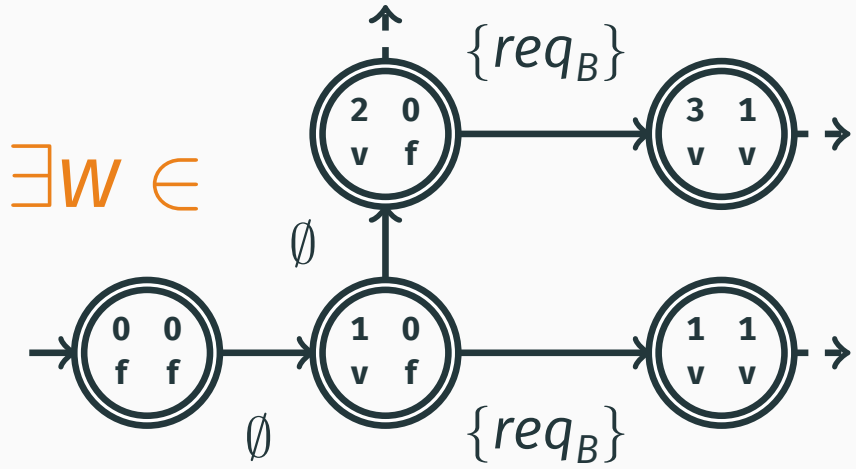
Vérification de systèmes finis: approche algorithmique



$\models$ $\mathbf{G}(req_B \rightarrow \mathbf{F}\, crit_B)$

$$P = \{req_B, crit_B\}$$

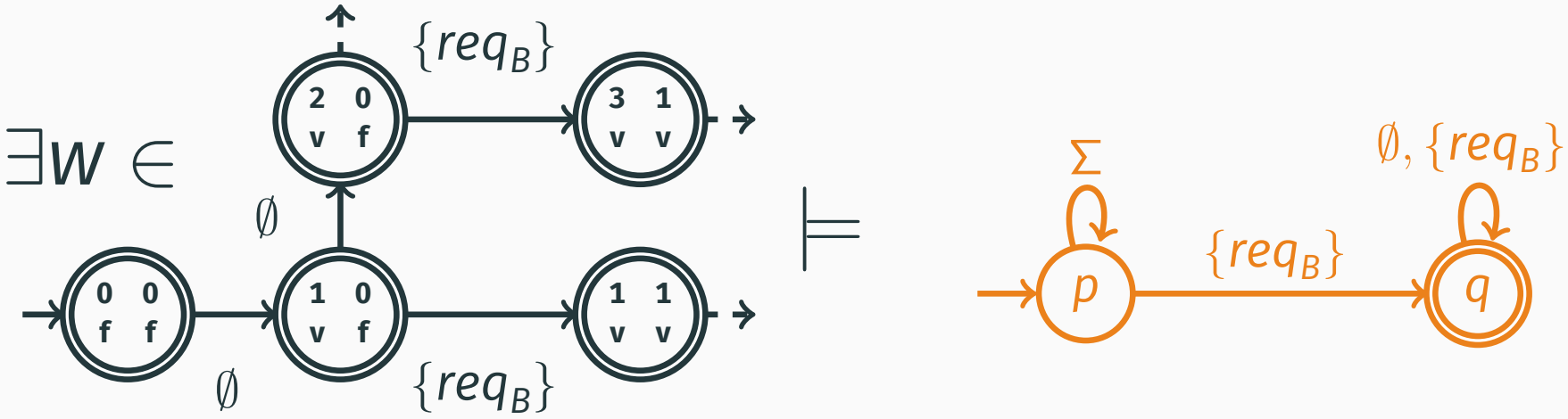
Vérification de systèmes finis: approche algorithmique



$$\models \quad \neg \mathbf{G}(req_B \rightarrow \mathbf{F}\, crit_B)$$

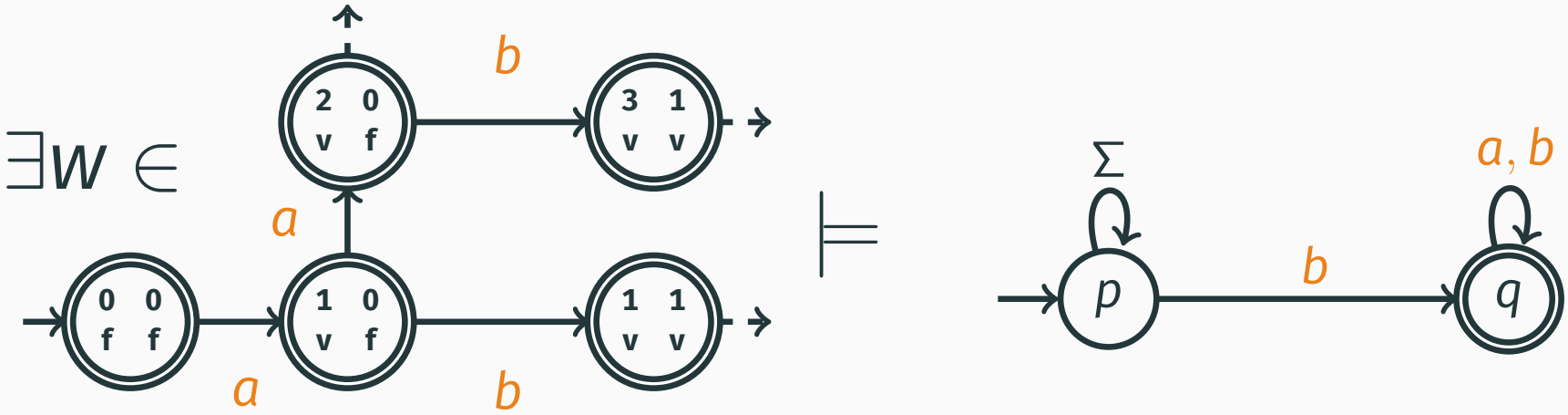
$$P = \{req_B, crit_B\}$$

Vérification de systèmes finis: approche algorithmique



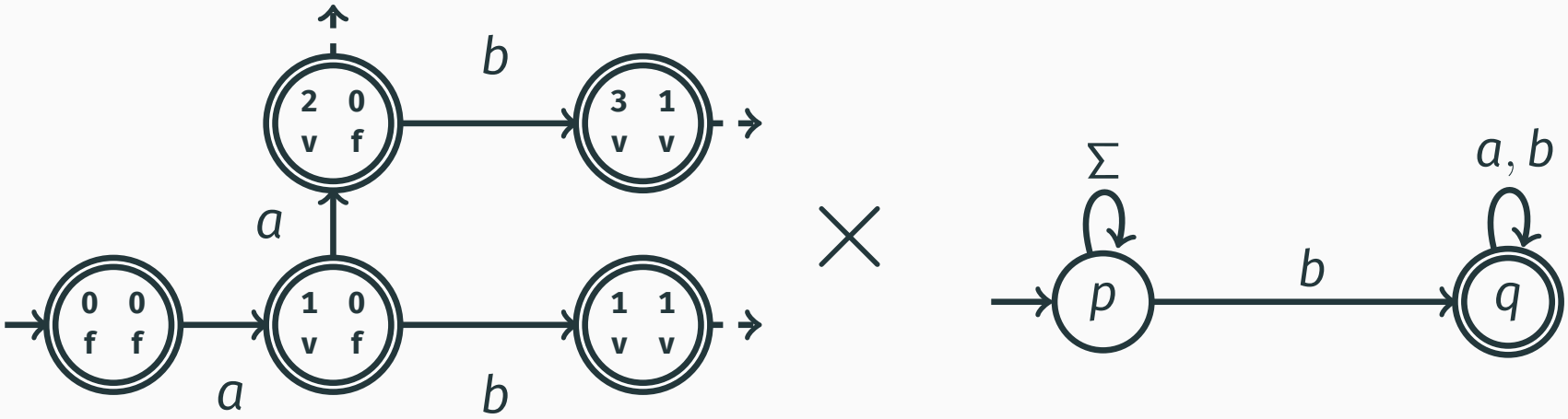
$$\Sigma = \{\emptyset, \{req_B\}, \{crit_B\}, \{req_B, crit_B\}\}$$

Vérification de systèmes finis: approche algorithmique



$$\Sigma = \{a, b, c, d\}$$

Vérification de systèmes finis: approche algorithmique

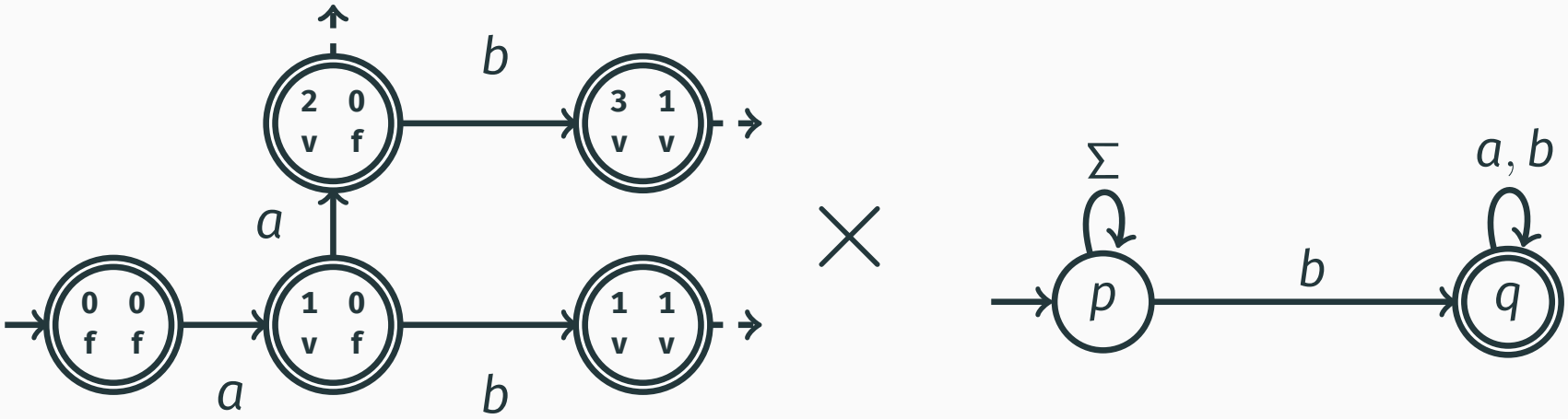


Système ne satisfait pas la propriété

$\Longleftrightarrow$

Les deux automates acceptent un mot en commun

Vérification de systèmes finis: approche algorithmique



Outils: SPIN, NuSMV, etc.

Nombre d'états: 10^9

parfois $[10^{20}, 10^{476}]$

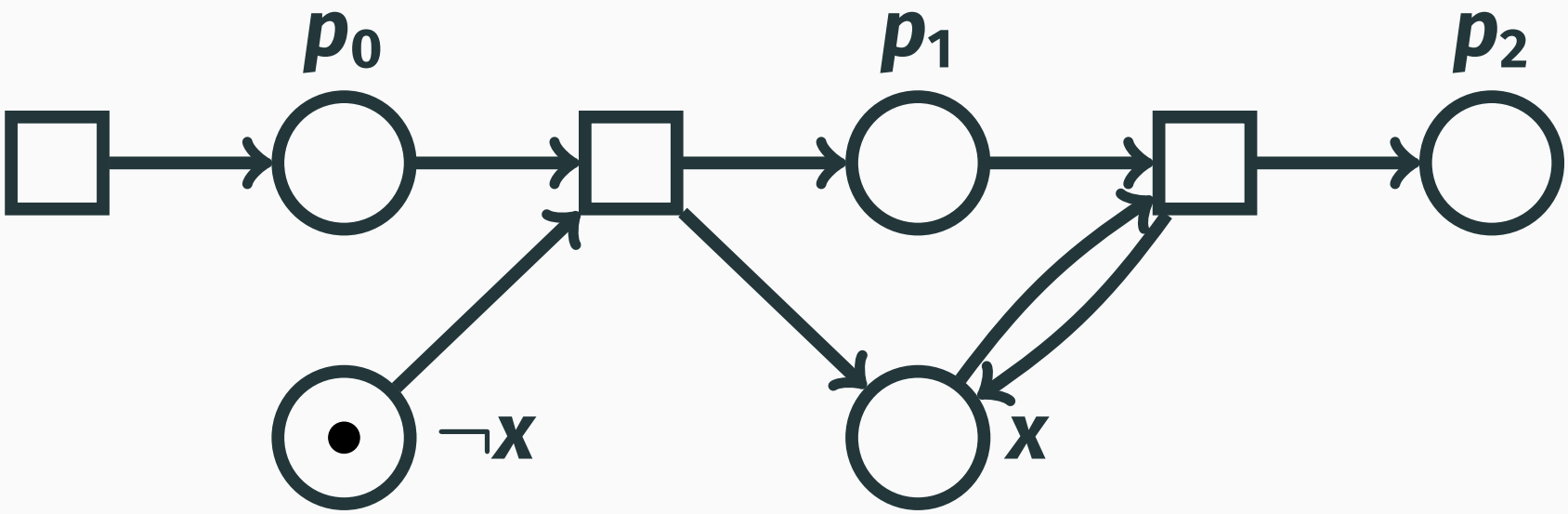
Si des processus peuvent être créés...

...alors il y a une infinité de configurations!

Cherchons à déterminer si plusieurs processus peuvent atteindre la ligne 2

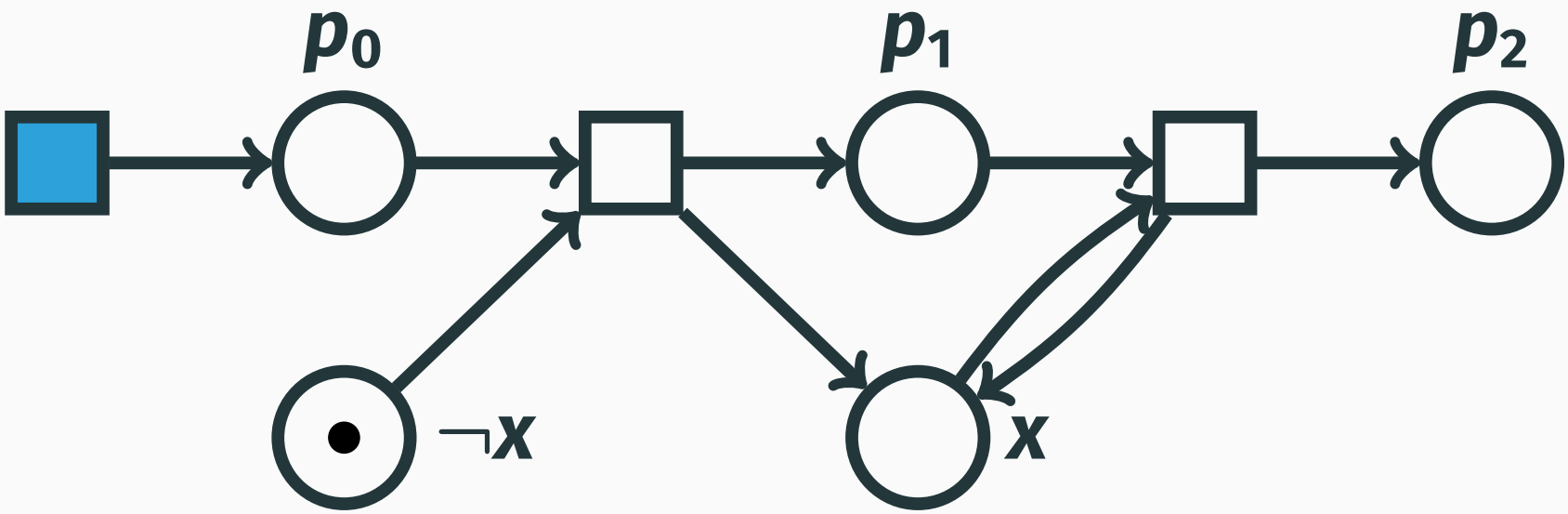
```
0. si ¬x alors: x = vrai, sinon: goto 0
1. tant que ¬x: pass
2. # cible
```

Vérification de systèmes infinis



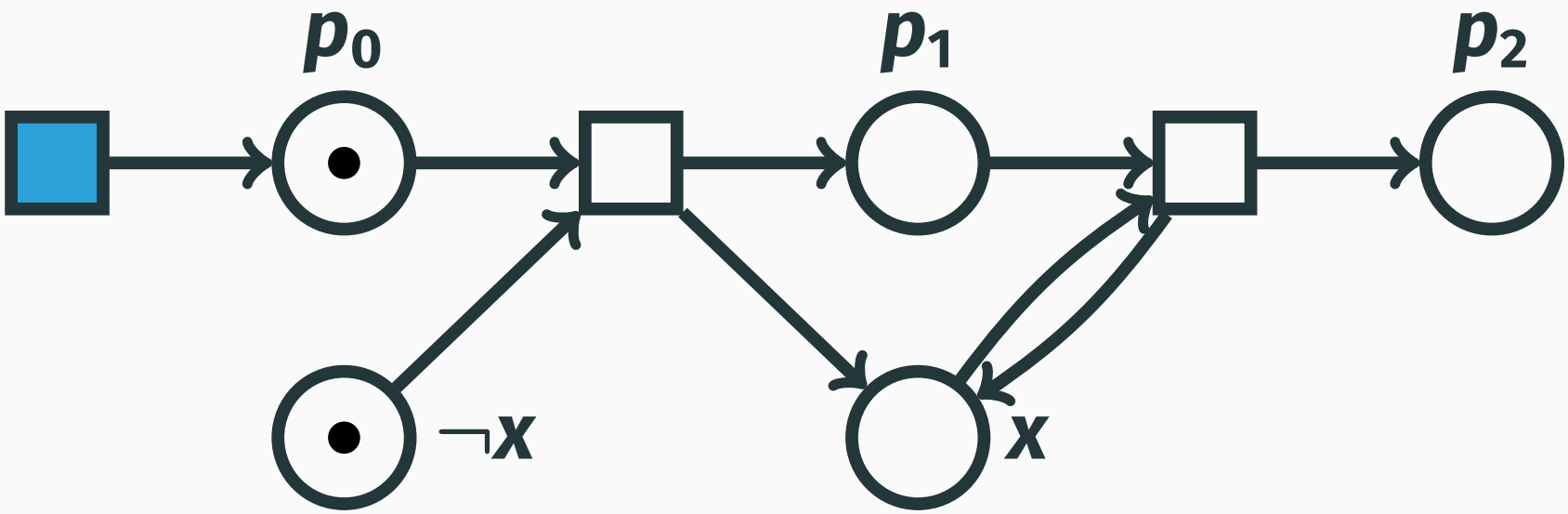
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



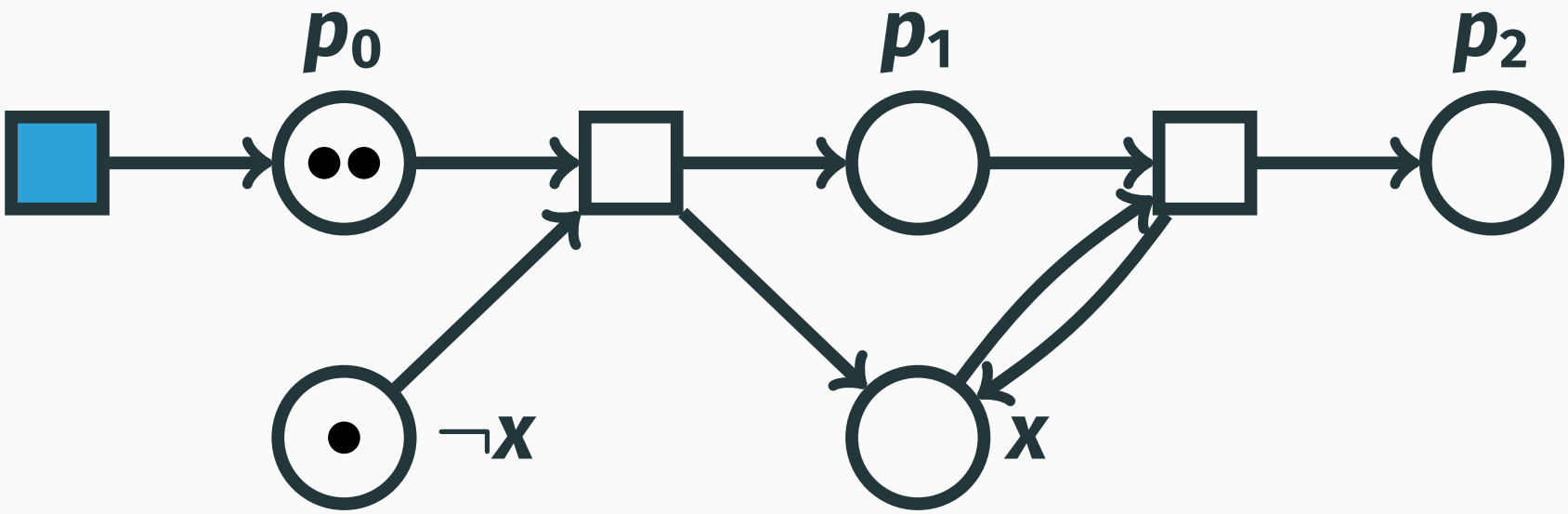
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



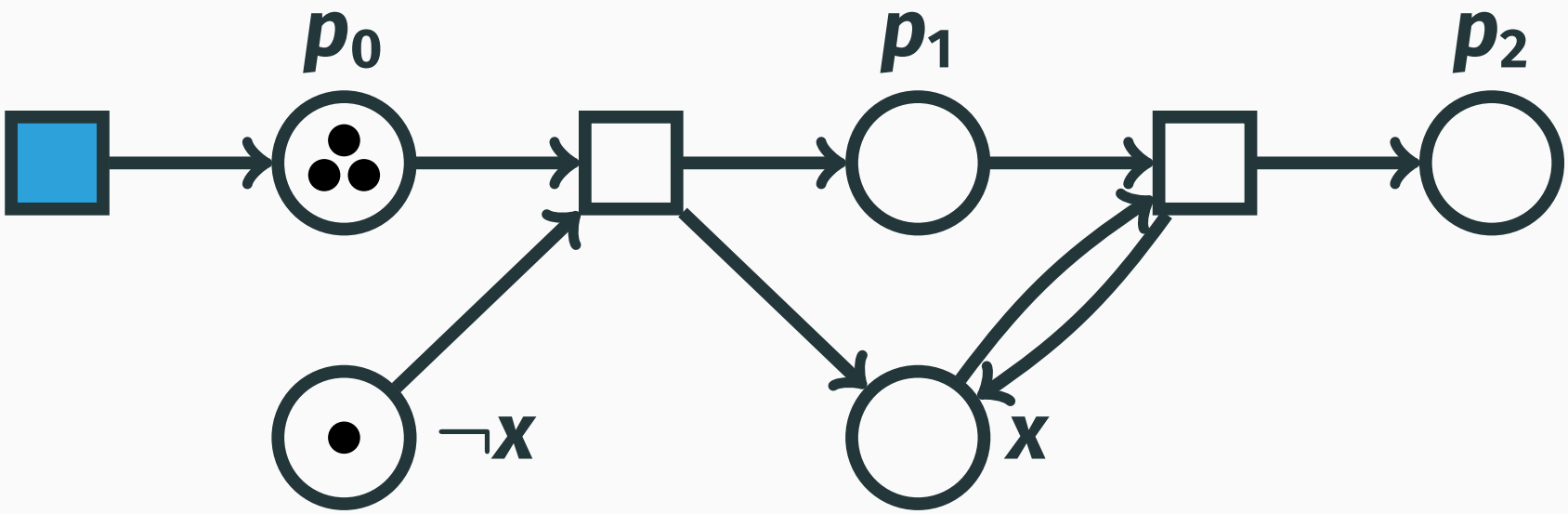
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



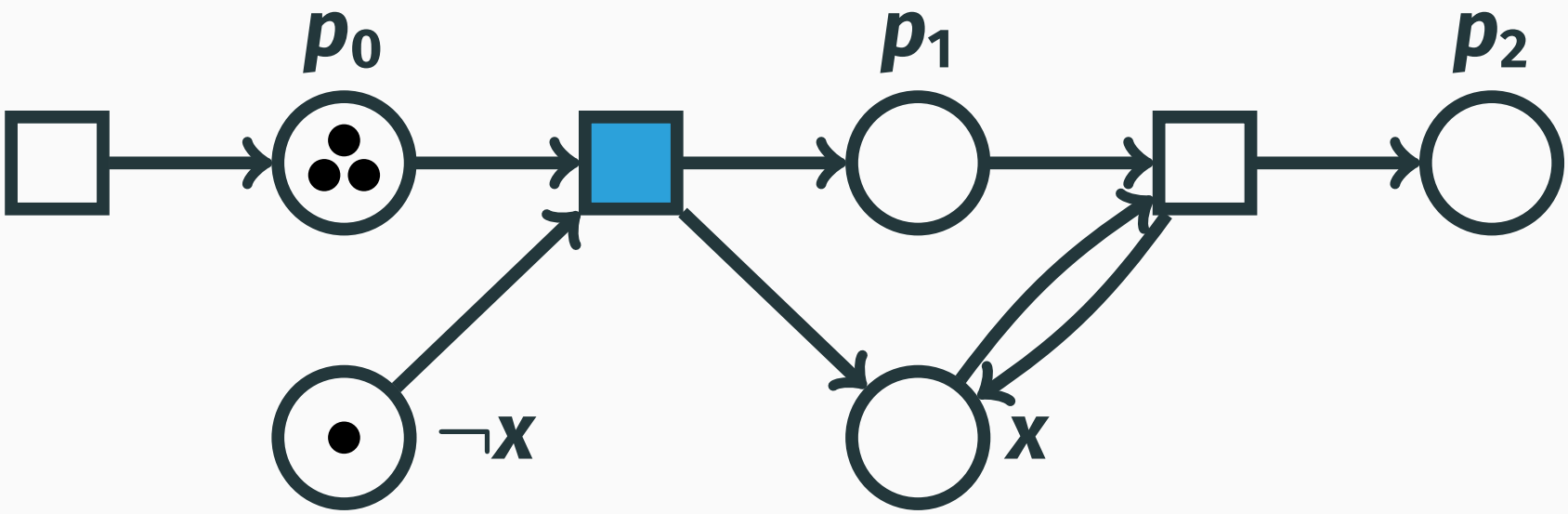
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



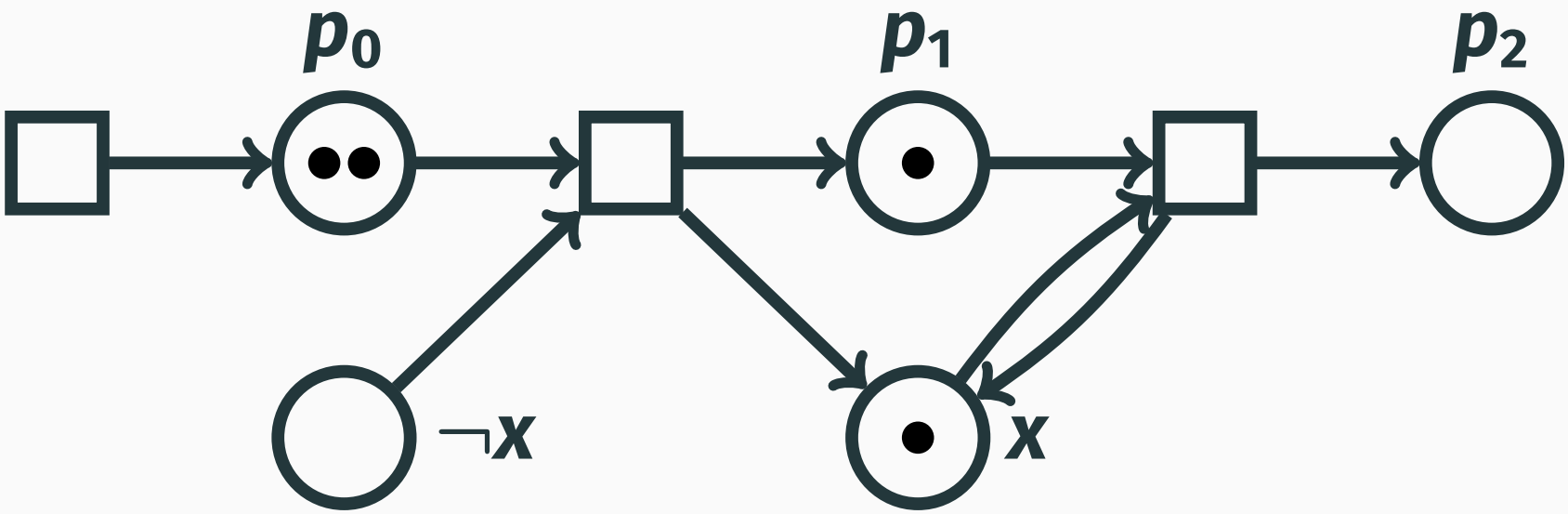
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



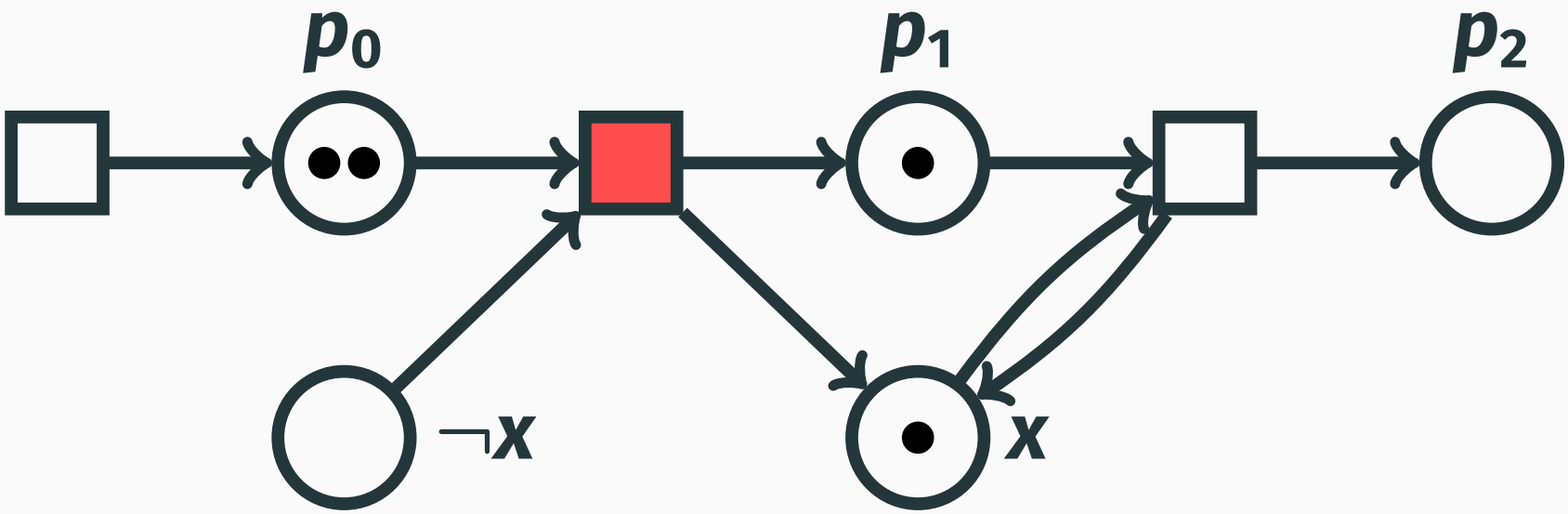
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



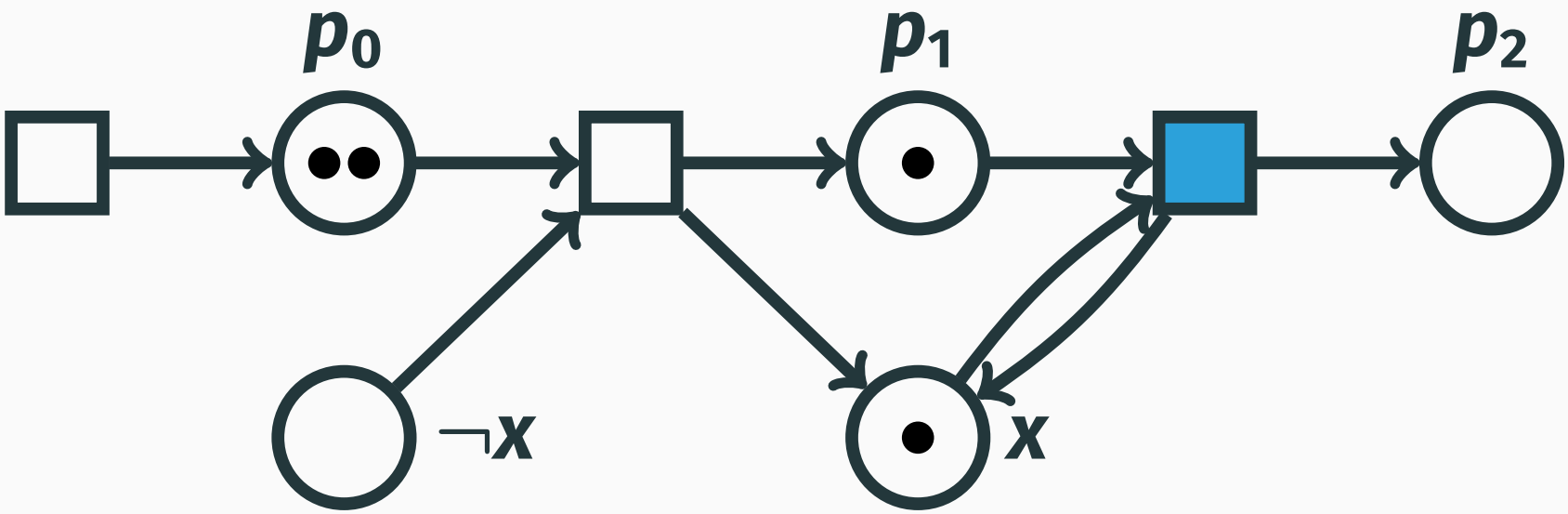
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



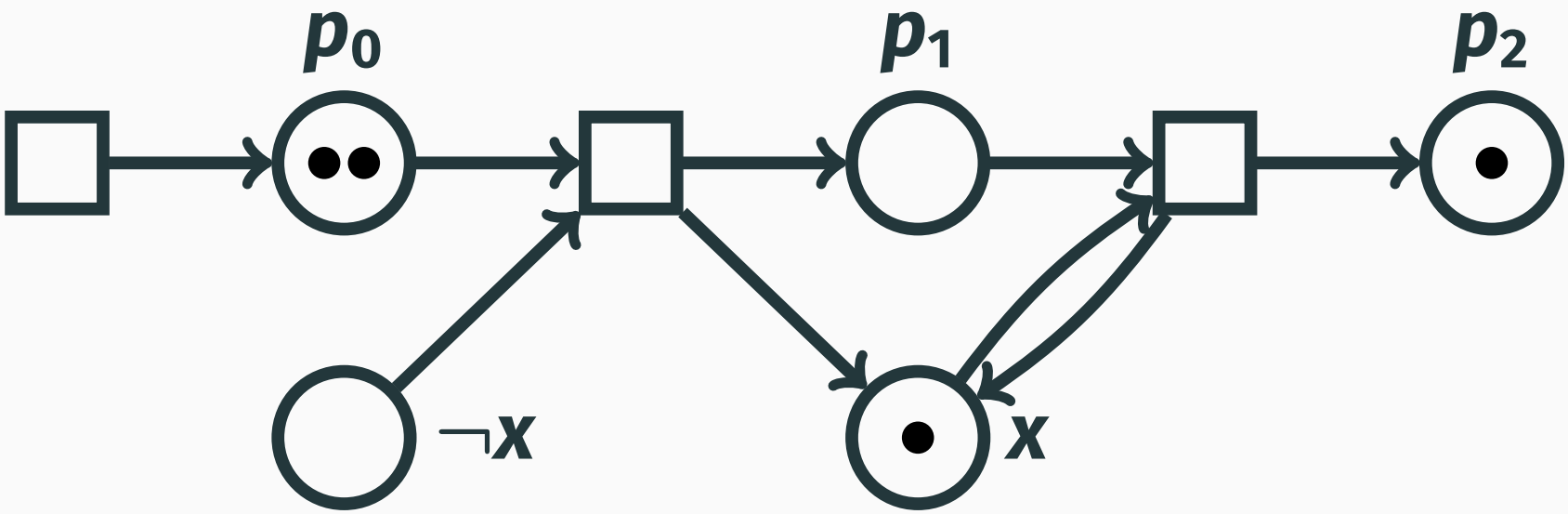
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



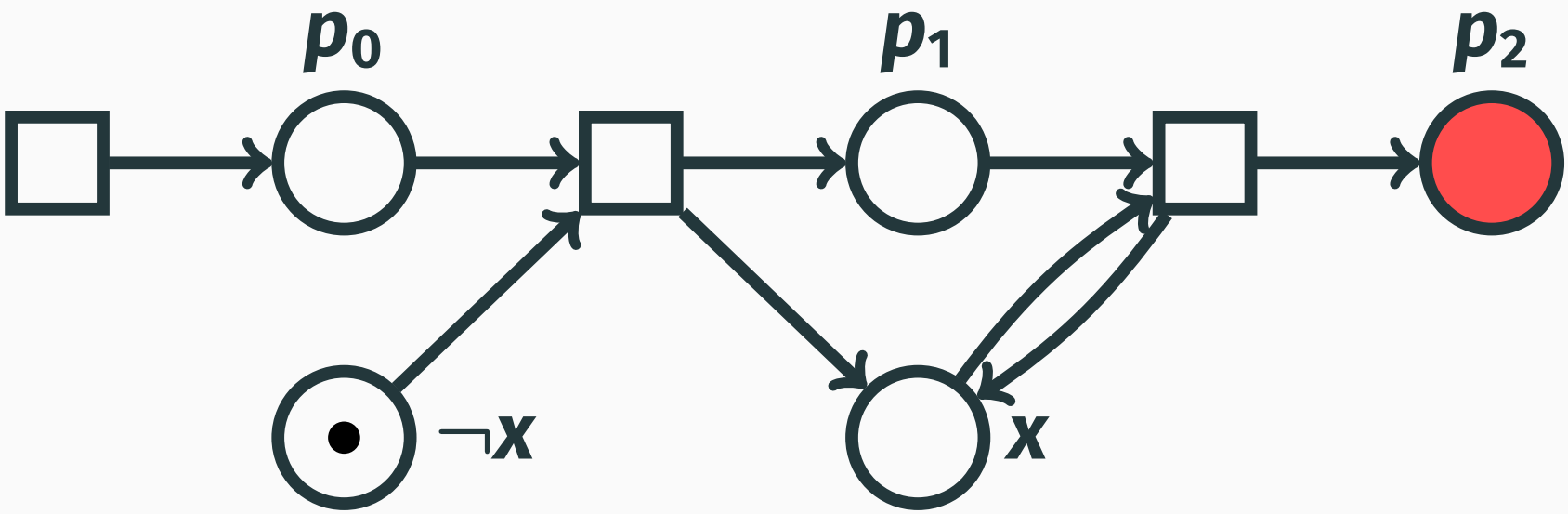
0. `si ¬x alors: x = vrai, sinon: goto 0`
1. `tant que ¬x: pass`
2. *# cible*

Vérification de systèmes infinis



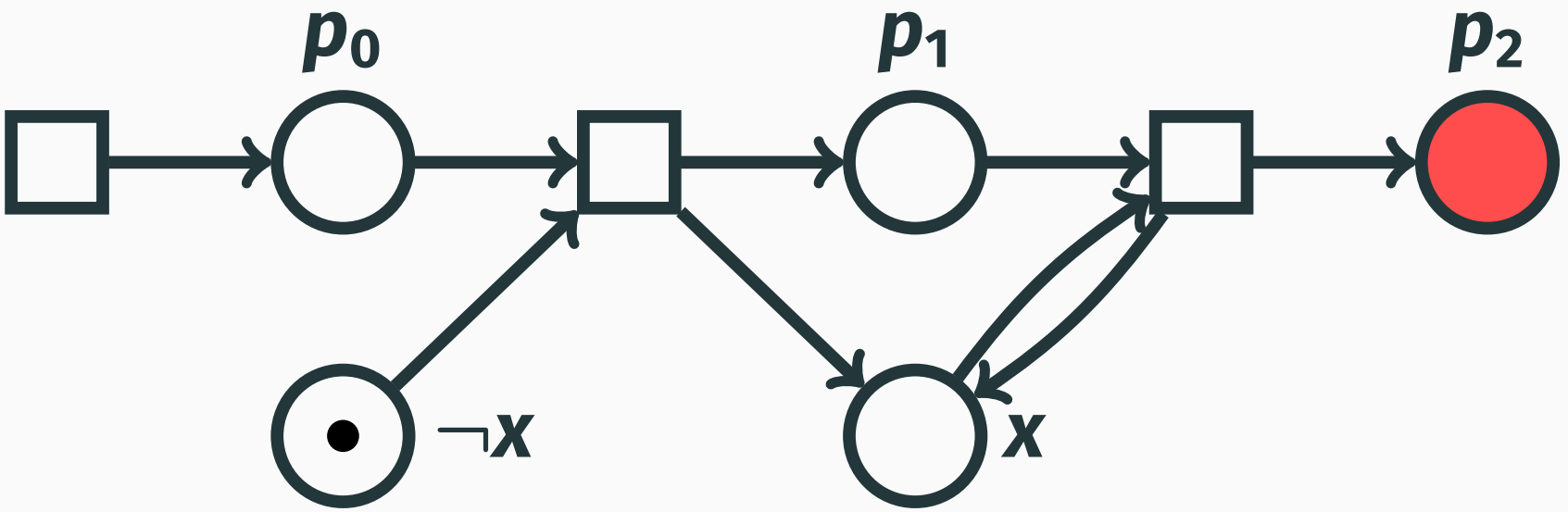

```
0. si ¬x alors: x = vrai, sinon: goto 0
1. tant que ¬x: pass
2. # cible
```

Vérification de systèmes infinis



$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & \end{pmatrix} \xrightarrow{*} \geq \begin{pmatrix} 0 & 0 & 2 \\ 0 & 0 & \end{pmatrix} \; ?$$

Vérification de systèmes infinis



Vérifiable algorithmiquement !

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & \end{pmatrix} \xrightarrow{*} \geq \begin{pmatrix} 0 & 0 & \mathbf{2} \\ 0 & 0 & \end{pmatrix} \; ?$$

Vérification quantitative

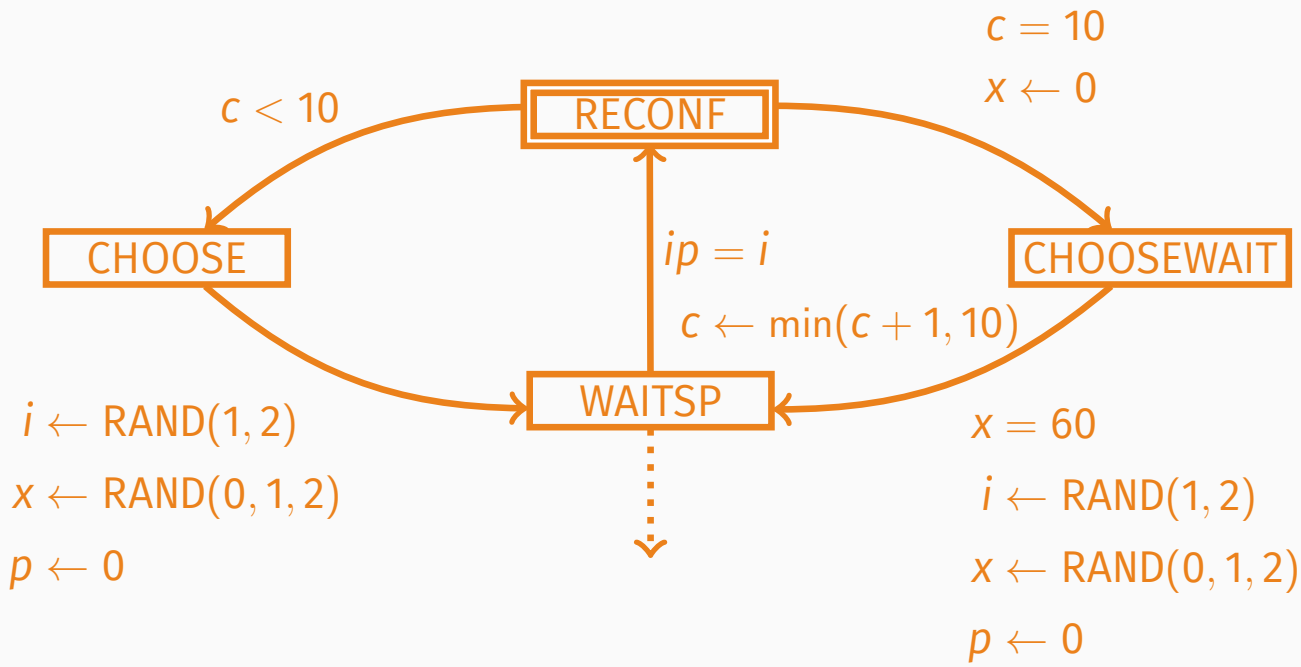
Protocole IPv4 Zeroconf: *Un appareil, désormais appelé hôte, qui souhaite configurer une adresse IP sélectionne d'abord aléatoirement une adresse parmi un ensemble de 65024 adresses. L'hôte envoie ensuite quatre paquets ARP, appelés sondes, à tous les autres hôtes du réseau. Les sondes contiennent l'adresse IP sélectionnée par l'hôte [...] et sont envoyées toutes les deux secondes [...]*

Vérification quantitative

Protocole IPv4 Zeroconf: *Un appareil, désormais appelé hôte, qui souhaite configurer une adresse IP sélectionne d'abord aléatoirement une adresse parmi un ensemble de 65024 adresses. L'hôte envoie ensuite quatre paquets ARP, appelés sondes, à tous les autres hôtes du réseau. Les sondes contiennent l'adresse IP sélectionnée par l'hôte [...] et sont envoyées toutes les deux secondes [...]*

Vérification quantitative

Protocole IPv4 Zeroconf: *Un appareil, désormais appelé hôte, qui souhaite configurer une adresse IP sélectionne d'abord aléatoirement une adresse parmi un ensemble de 65024 adresses. L'hôte envoie ensuite quatre paquets ARP, appelés sondes, à tous les autres hôtes du réseau. Les sondes contiennent l'adresse IP sélectionnée par l'hôte [...] et sont envoyées toutes les deux secondes [...]*

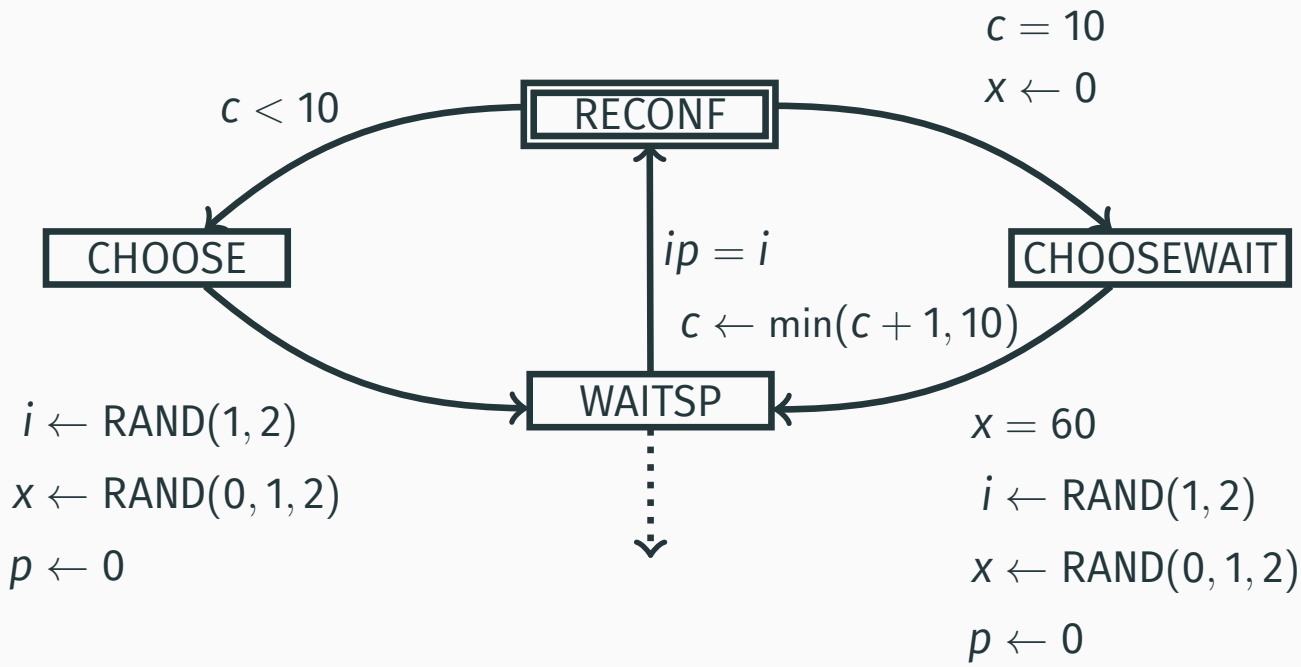


Vérification quantitative

Possible de vérifier algorithmiquement les systèmes avec:

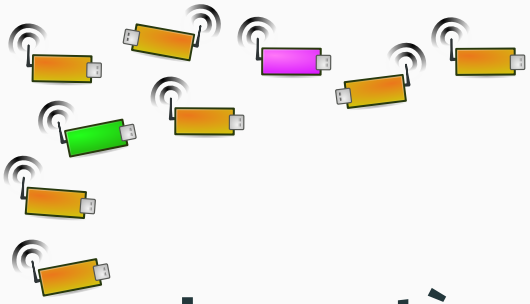
- **probabilités, coûts (processus de décision markovien)**
- **temps, horloges (automates temporisés)**

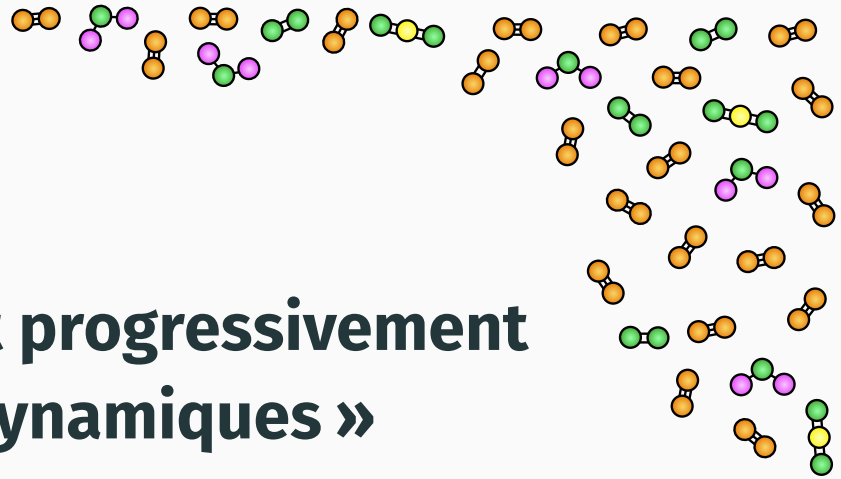
Outils: UPPAAL, PRISM, Storm, etc.



2. Vérification de réseaux dynamiques

Réseaux dynamiques



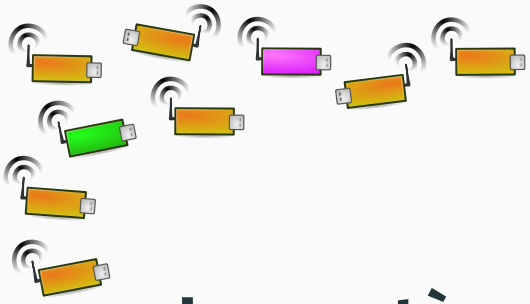


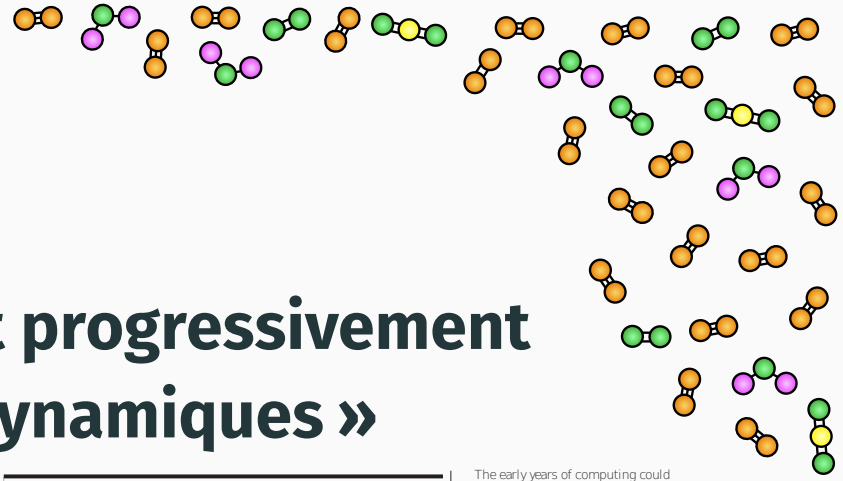
Les systèmes traditionnels font progressivement place aux « réseaux dynamiques »

Par ex.

- Réseaux de capteurs sans fil
- Essaims de robots
- Ordinateurs moléculaires

Réseaux dynamiques





Les systèmes traditionnels font progressivement place aux « réseaux dynamiques »

Par ex.

- Réseaux de capteurs sans fil
- Essaims de robots
- Ordinateurs moléculaires

The challenge of computing in a highly dynamic environment.

BY OTHON MICHAIL AND PAUL G. SPIRAKIS

Elements of the Theory of Dynamic Networks

A DYNAMIC NETWORK is a network that changes with time. Nature, society, and the modern communications landscape abound with examples. Molecular interactions, chemical reactions, social relationships and interactions in human and animal populations, transportation networks, mobile wireless devices, and robot collectives form only a small subset of the systems whose dynamics can be naturally modeled and analyzed by some sort of dynamic network. Though many of these systems have always existed, it was not until recently the need for a formal treatment that would consider *time* as an integral part of the network has been identif ed. Computer science is leading this major shift, mainly driven by the advent of low-cost wireless communication devices and the development of eff cient wireless communication protocols.

The early years of computing could be characterized as the era of staticity and of the relatively predictable; centralized algorithms for (combinatorial optimization) problems concerning static instances, as is that of f nding a minimum cost traveling salesman tour in a complete weighted graph, computability questions in cellular automata, and protocols for distributed tasks in a static network. Even when changes were considered, as is the case in fault-tolerant distributed computing, the dynamics were usually suff ciently slow to be handled by conservative approaches, in principle too weak to be useful for highly dynamic systems. An exception is the area of online algorithms, where the input is not known in advance and is instead revealed to the algorithm during its course. Though the original motivation and context of online algorithms is not related to dynamic networks, the existing techniques and body of knowledge of the former may prove very useful in tackling the high unpredictability inherent in the latter.

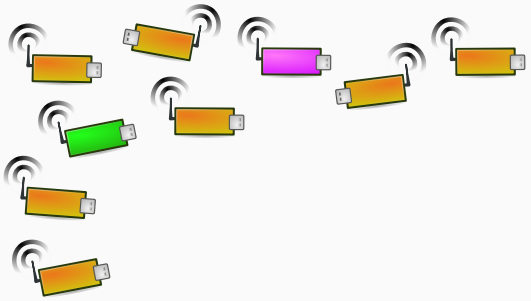
In contrast, we are rapidly approaching, if not already there, the era of *dynamicity* and of the highly *unpredictable*. According to some latest reports, the number of mobile-only Internet users has already exceeded the number of desktop-only Internet users and

» key insights

- **We are rapidly approaching the era of dynamicity and of the highly unpredictable. A great variety of modern networked systems are highly dynamic both in space and time.**
- **Theory will continue sitting at the center of progress in our science and its necessity toward our understanding of dynamic networks is already evident.**
- **Many traditional approaches and measures for static networks are not adequate for dynamic networks. There is already strong evidence that there is room for the development of a rich theory.**
- **Despite the considerable recent progress discussed in this article, we do not yet really know how to compute in highly dynamic environments.**

IMAGE BY ANTHUSOV MARTIN

Réseaux dynamiques



Les méthodes de vérification sont peu adaptées à leur

- nombre arbitraire d'agents (vérification paramétrée)
- réorganisation constante
- comportements probabilistes

Protocoles de population

Angluin *et al.* 2004

- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus


Angluin


Aspnes


Diamadi

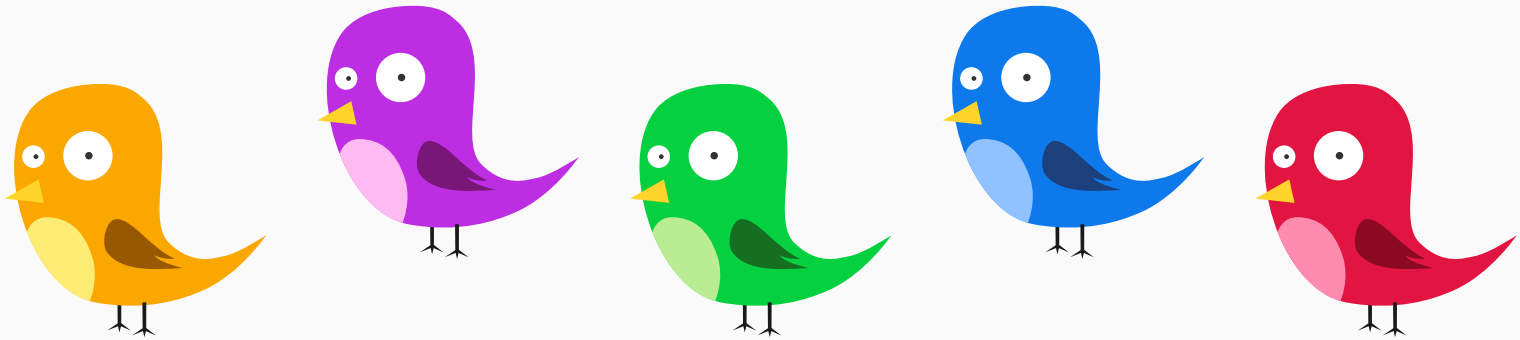

Fischer


Peralta

Protocoles de population

Angluin *et al.* 2004

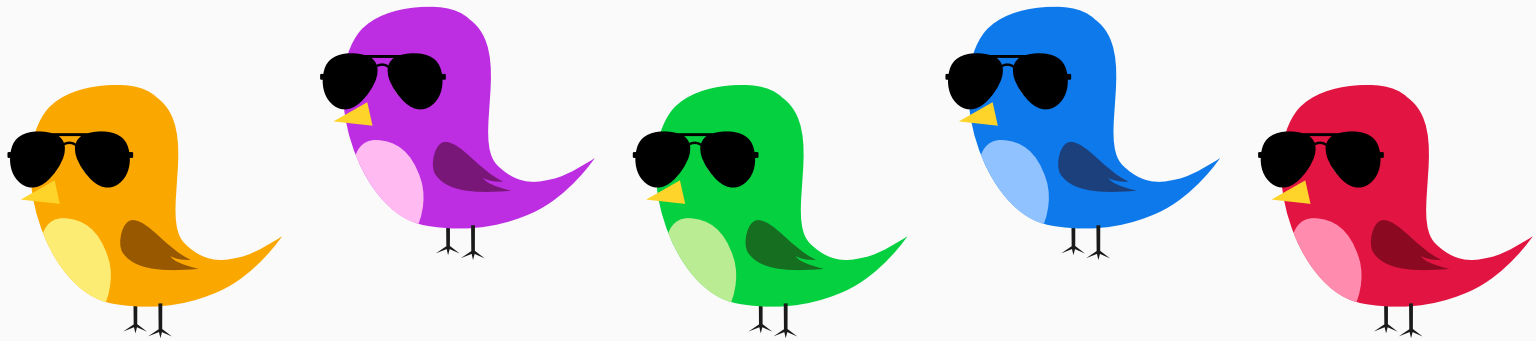
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

Angluin *et al.* 2004

- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

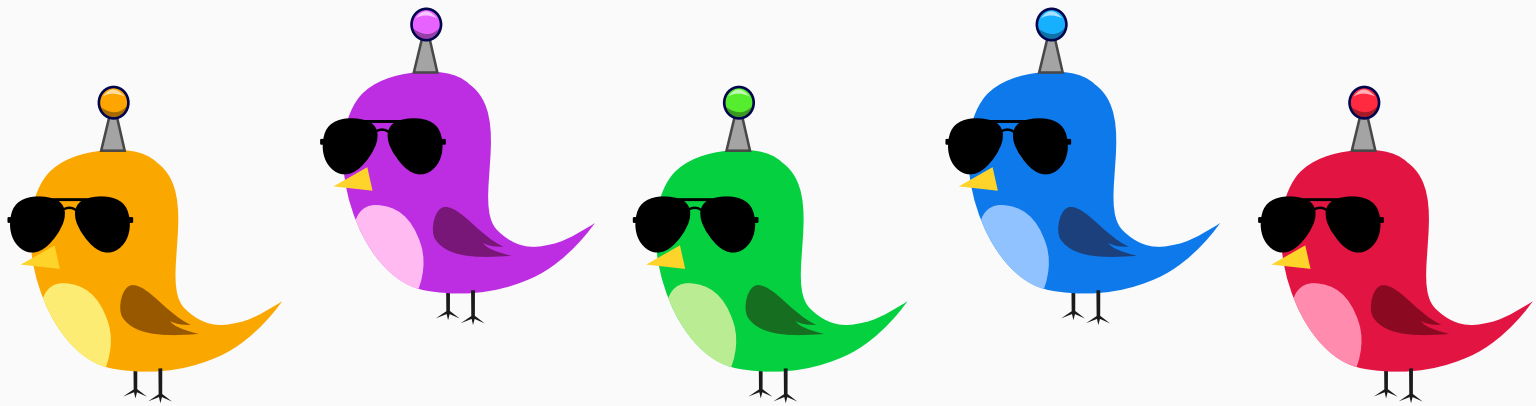
Angluin *et al.* 2004

- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus

Protocoles de population

Angluin *et al.* 2004

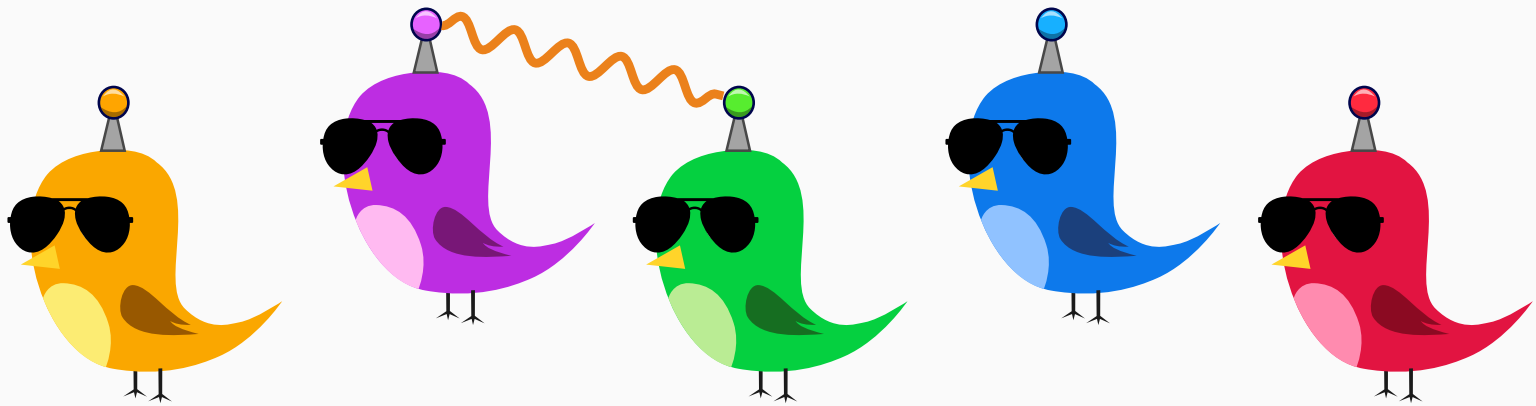
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

Angluin *et al.* 2004

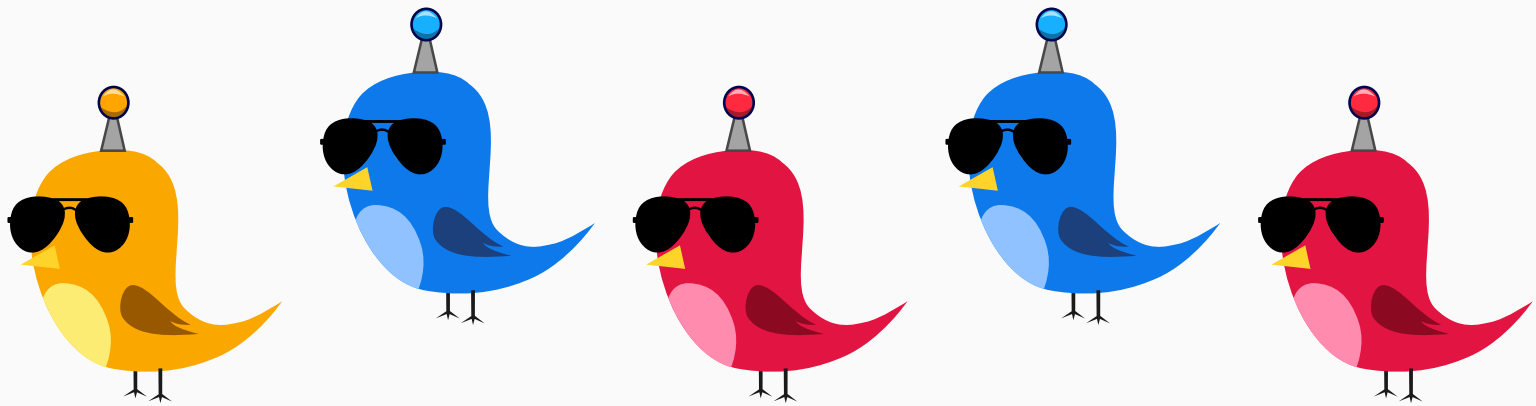
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

Angluin *et al.* 2004

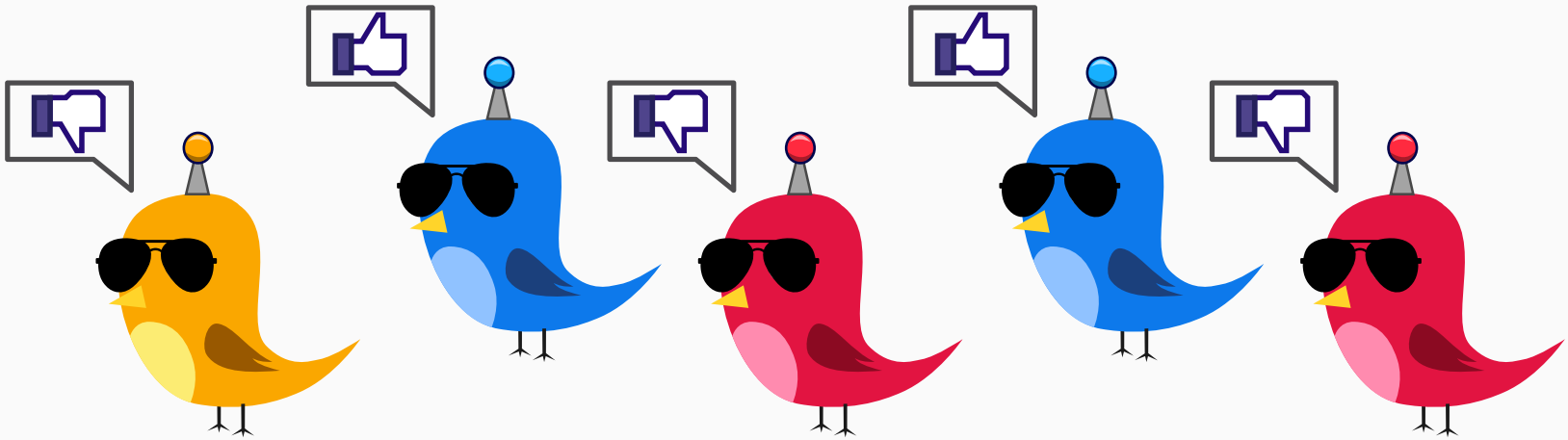
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

Angluin *et al.* 2004

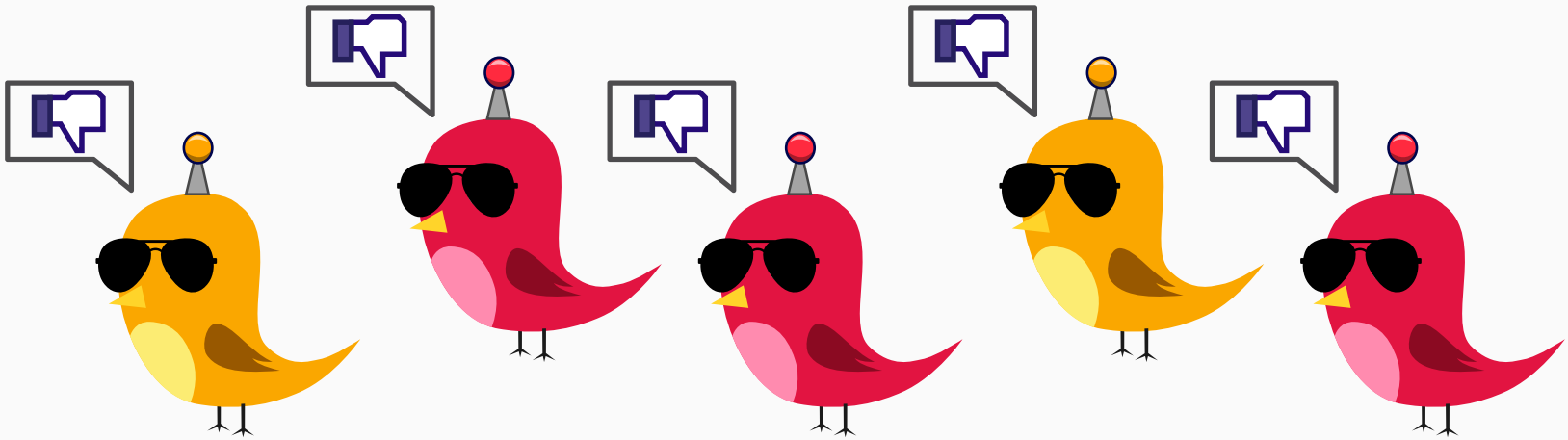
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

Angluin *et al.* 2004

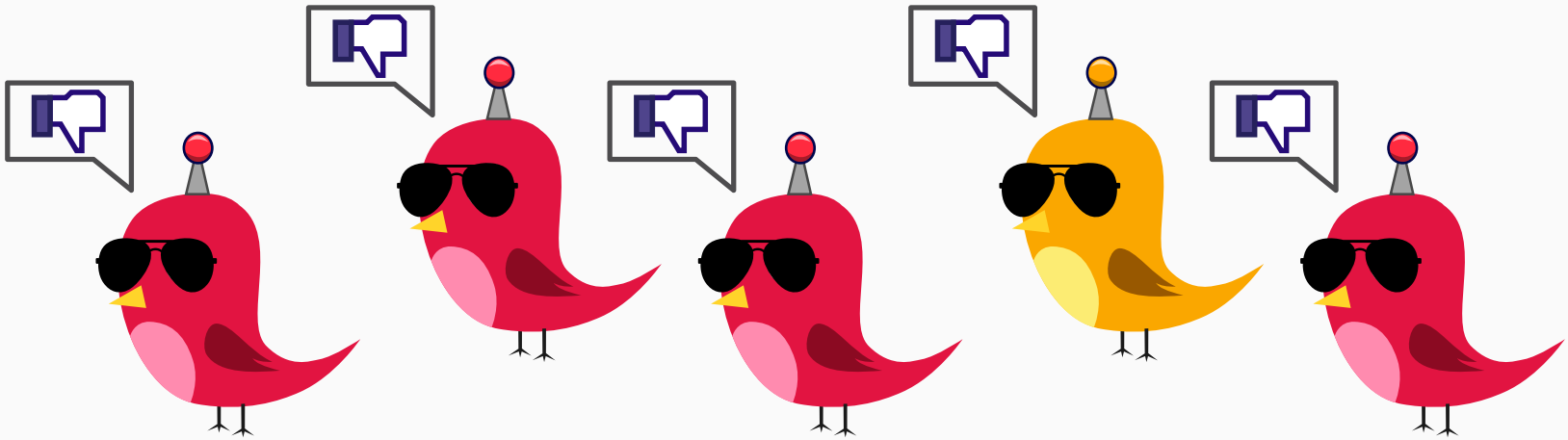
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

Angluin *et al.* 2004

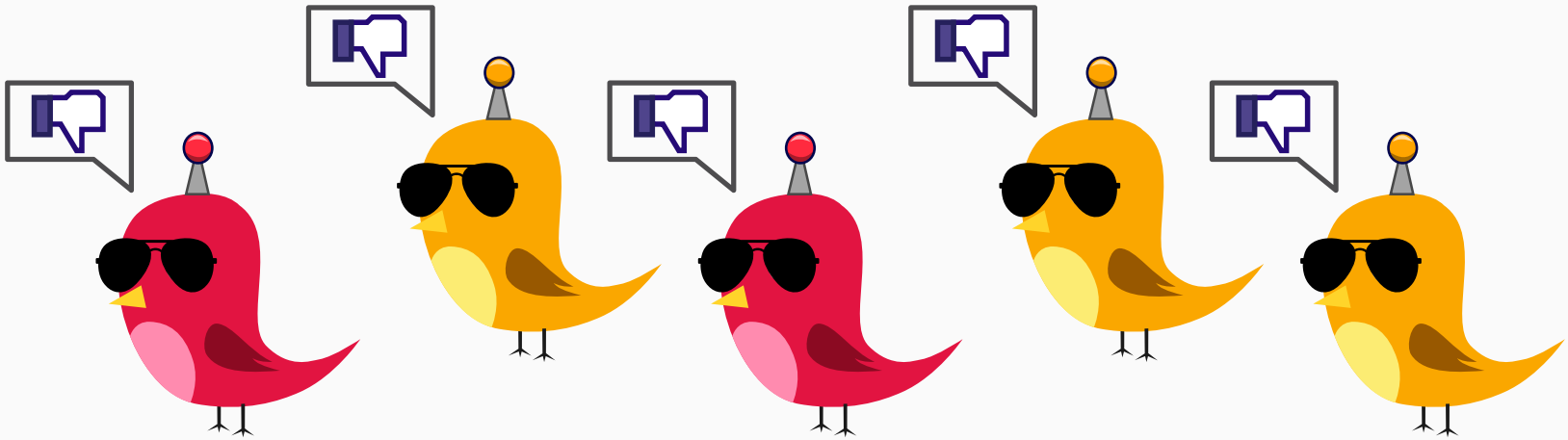
- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population

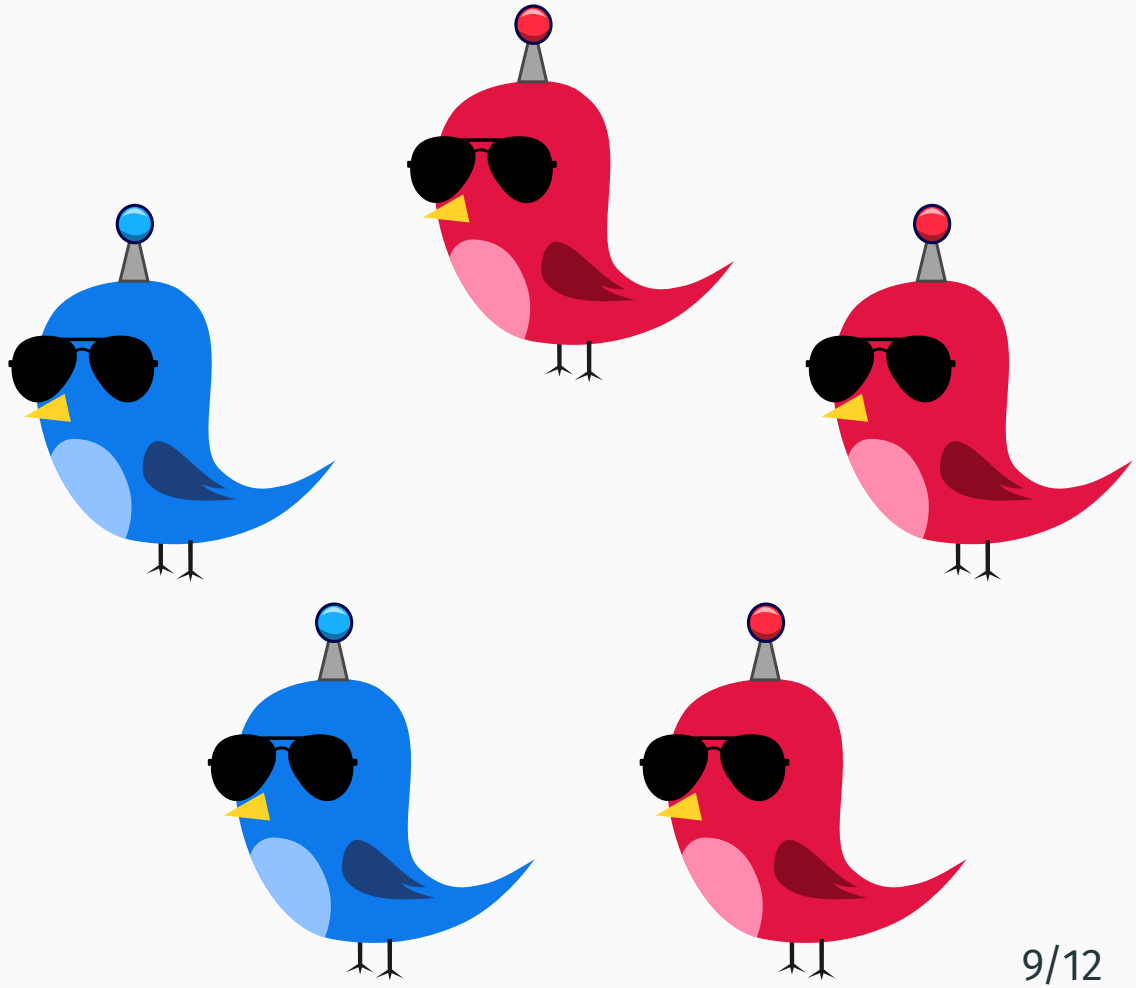
Angluin *et al.* 2004

- agents mobiles anonymes avec très peu de ressources
- les agents intéragissent en paires
- chaque agent a une opinion vrai/faux
- les agents arrivent éventuellement à un consensus



Protocoles de population: exemple

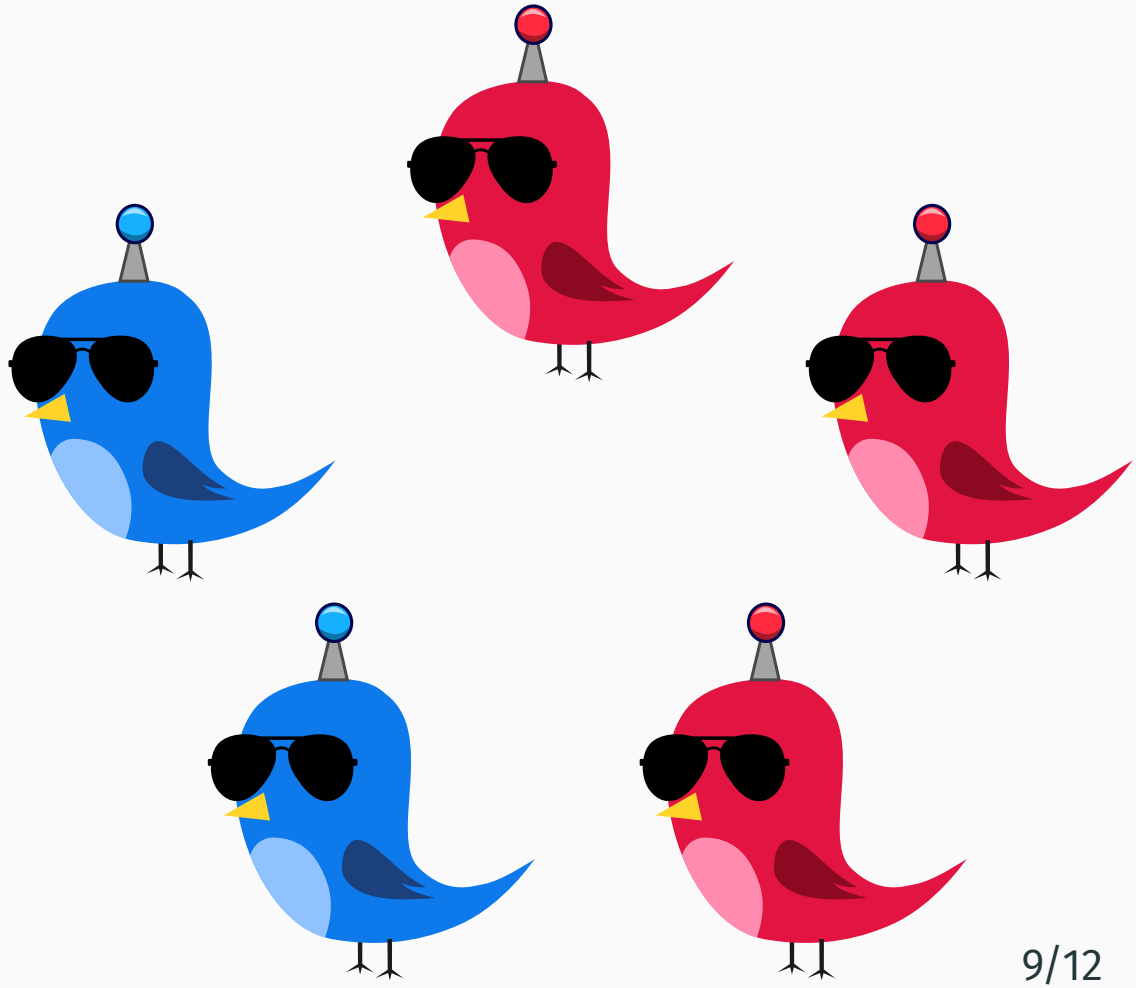
Plus d'agents bleus que d'agents rouges ?



Plus d'agents bleus que d'agents rouges ?

Interactions :

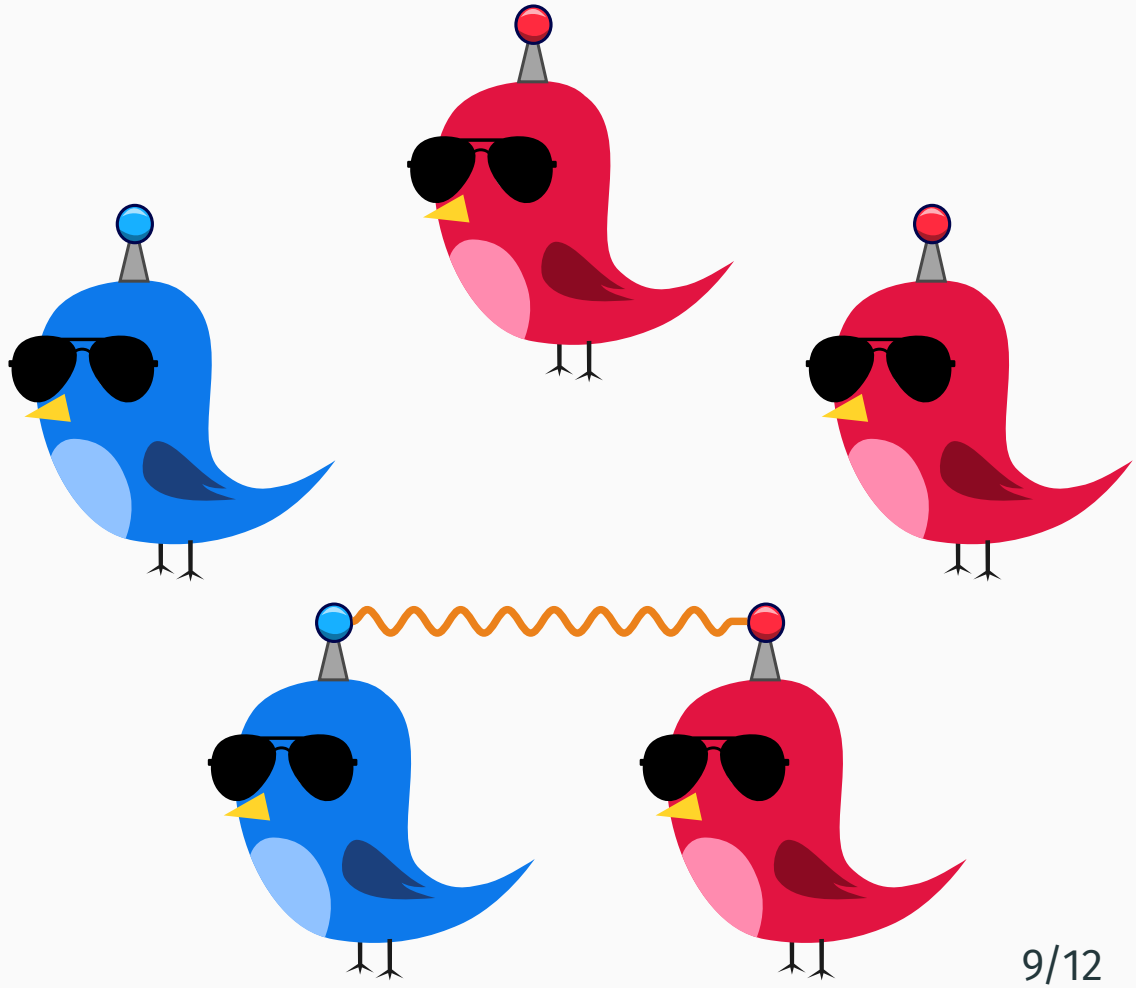
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

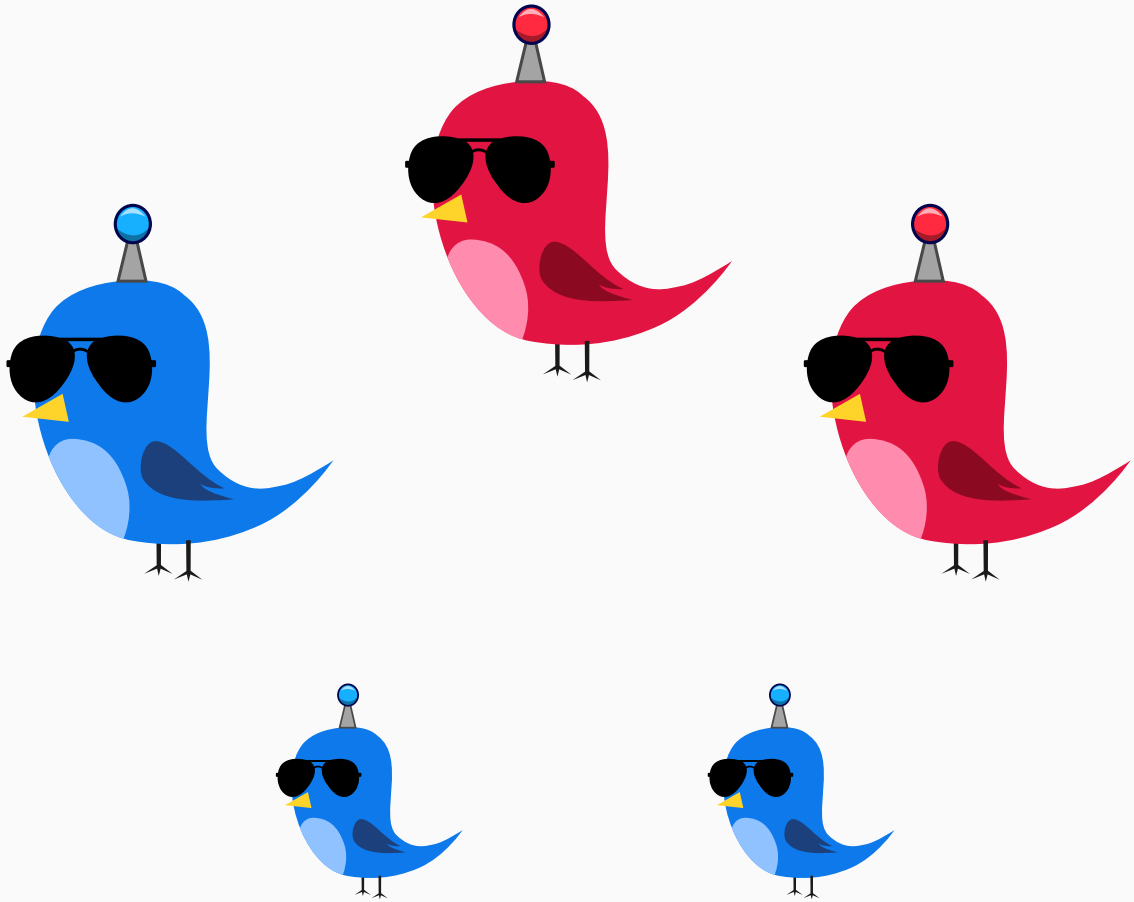
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

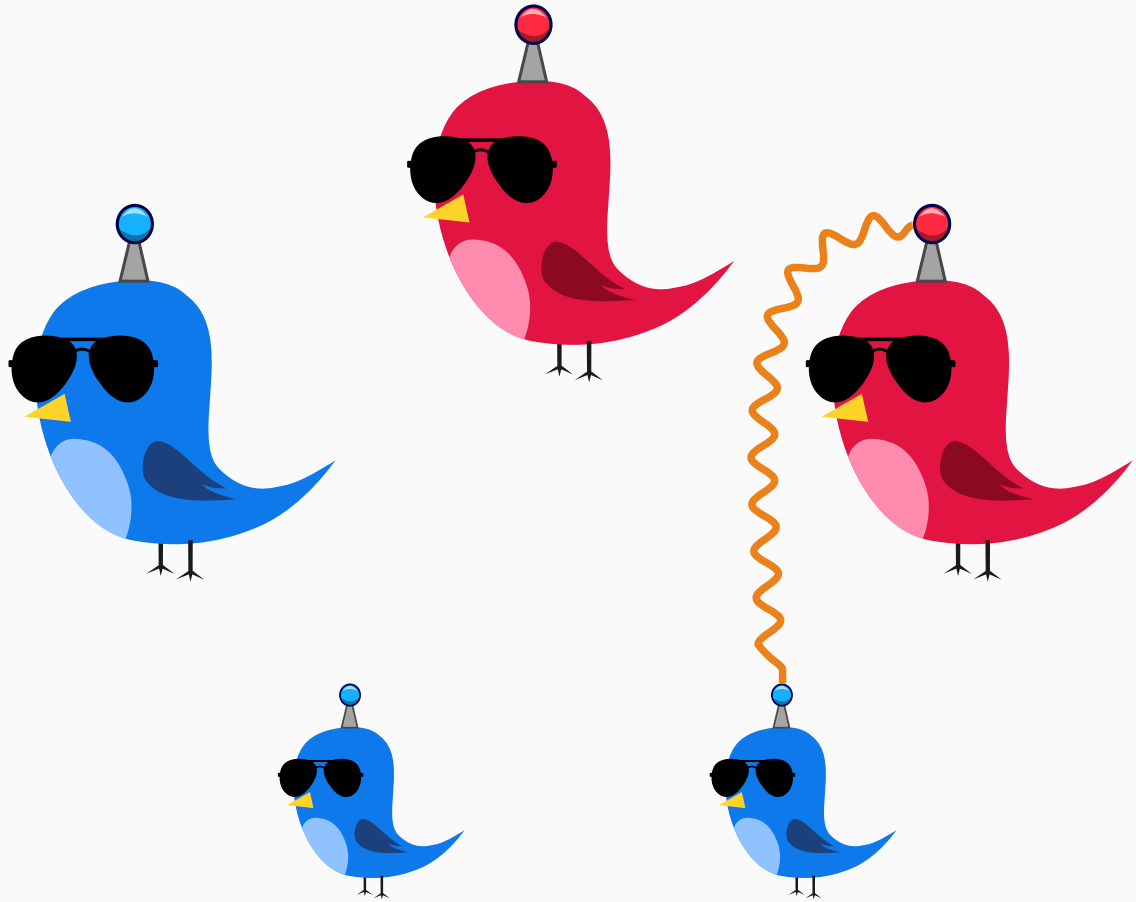
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

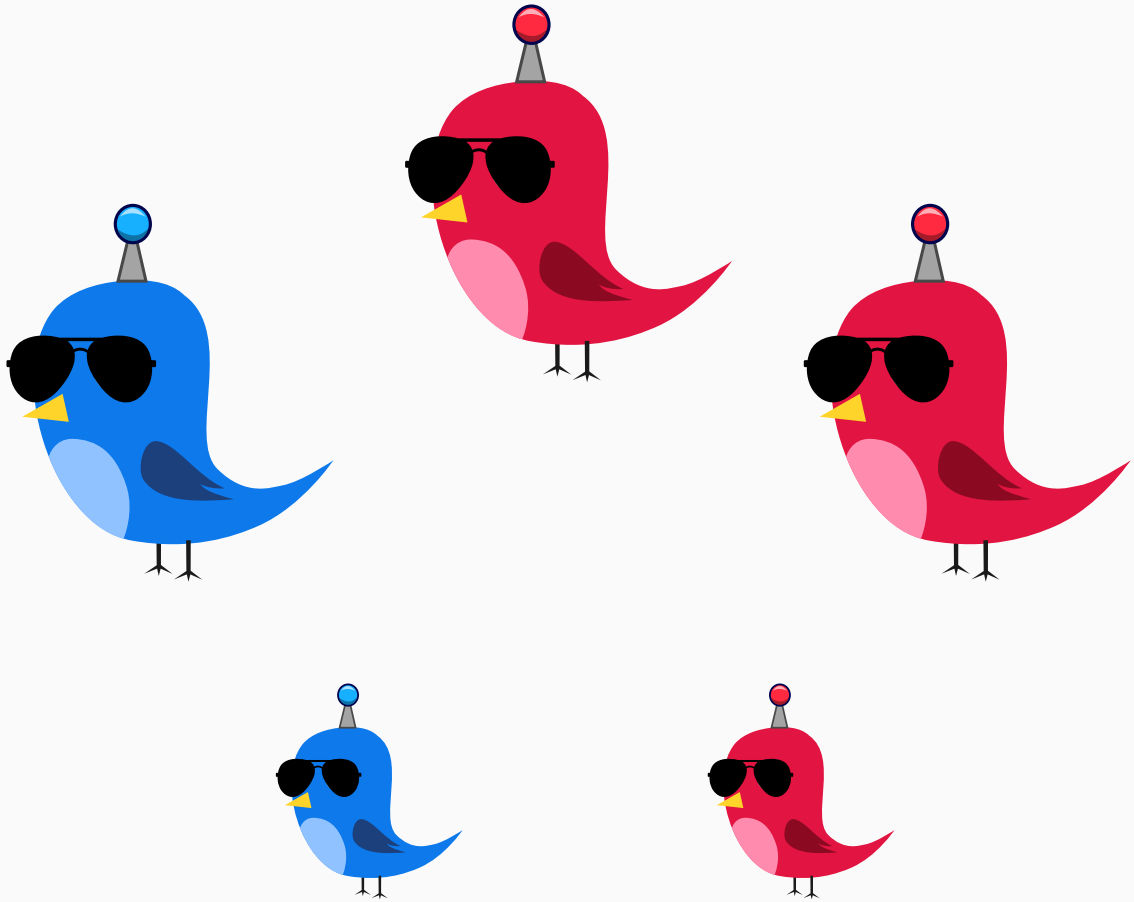
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

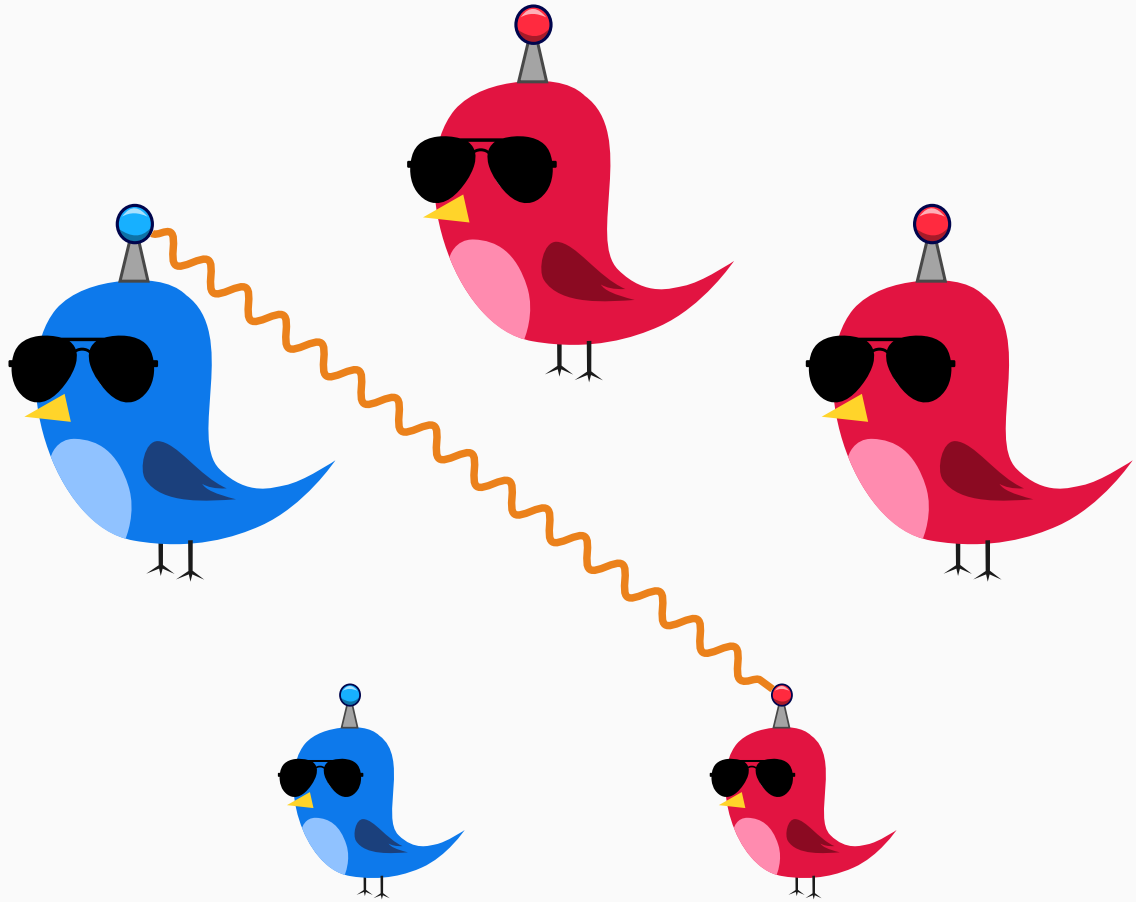
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

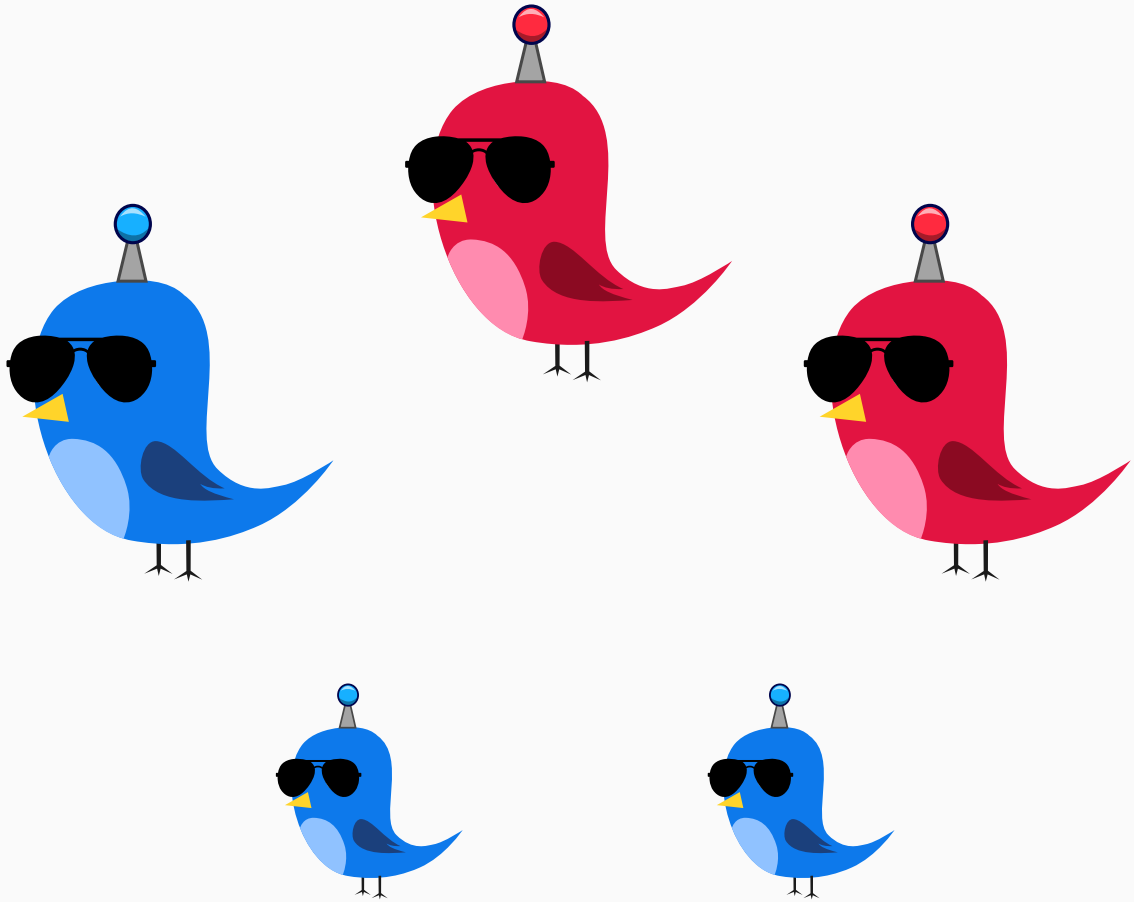
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

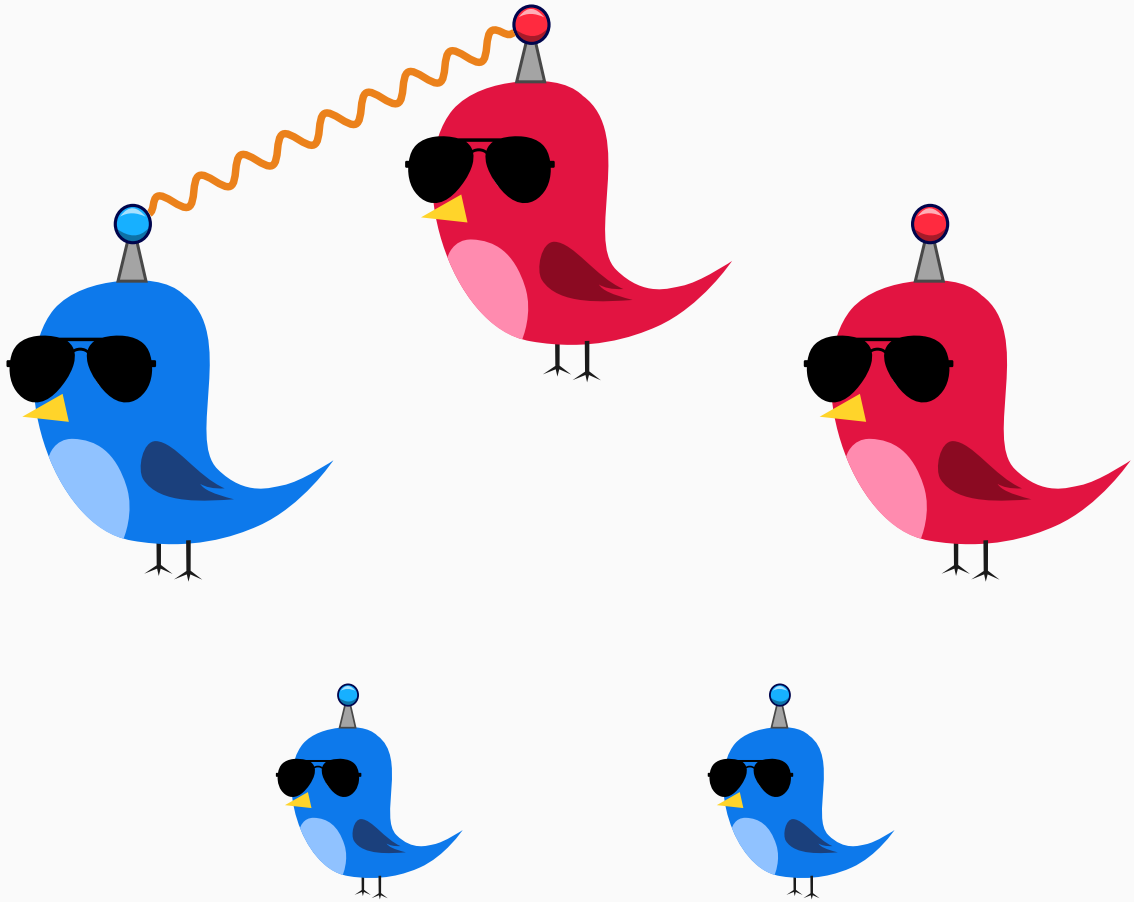
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur

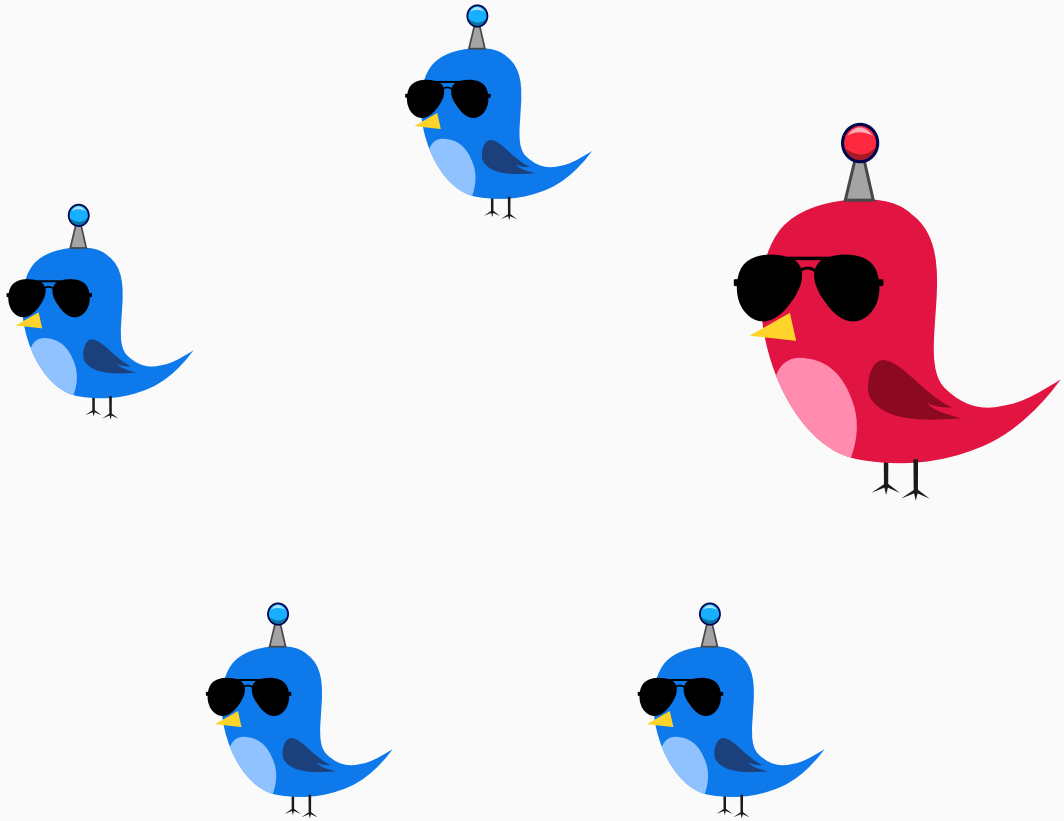


Protocoles de population: exemple

Plus d'agents bleus que d'agents rouges ?

Interactions :

- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur

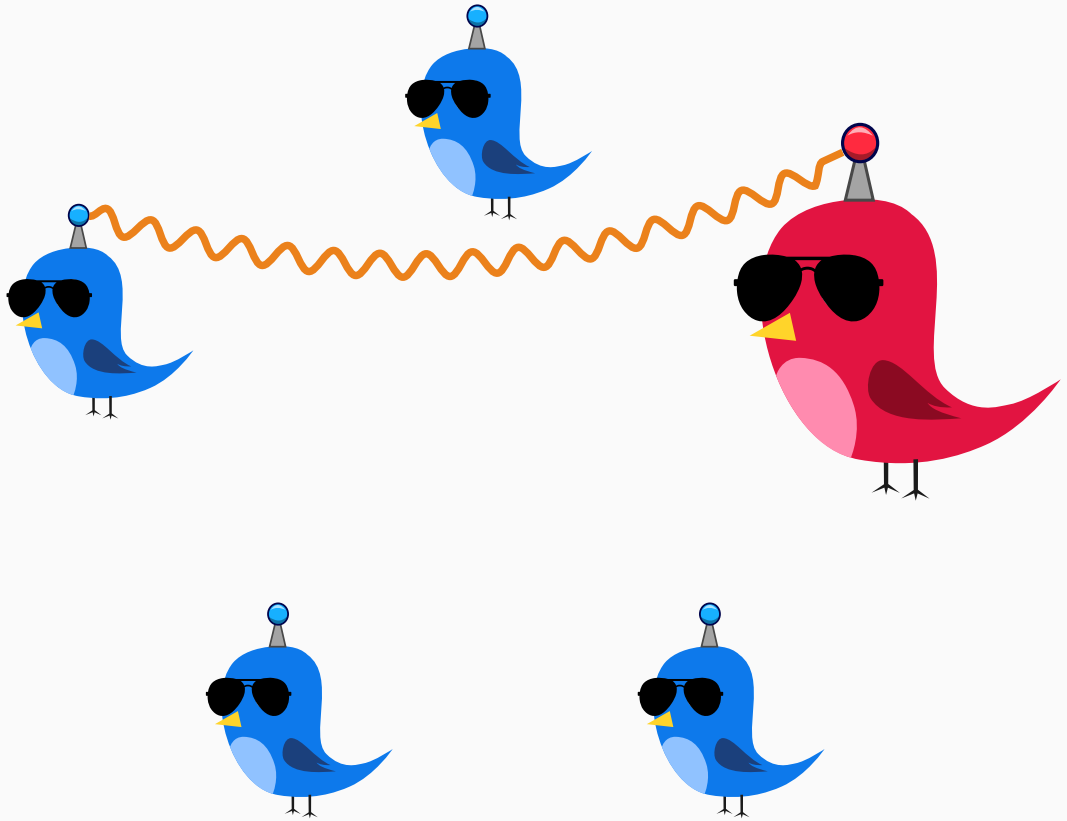


Protocoles de population: exemple

Plus d'agents bleus que d'agents rouges ?

Interactions :

- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur

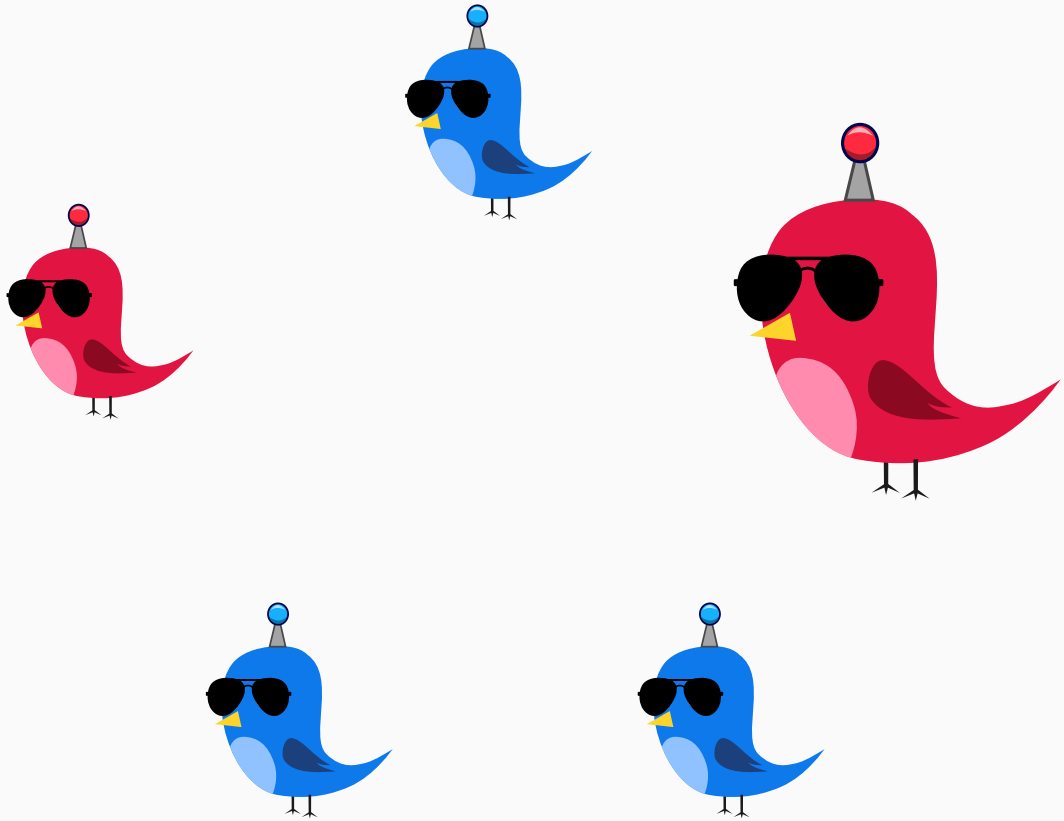


Protocoles de population: exemple

Plus d'agents bleus que d'agents rouges ?

Interactions :

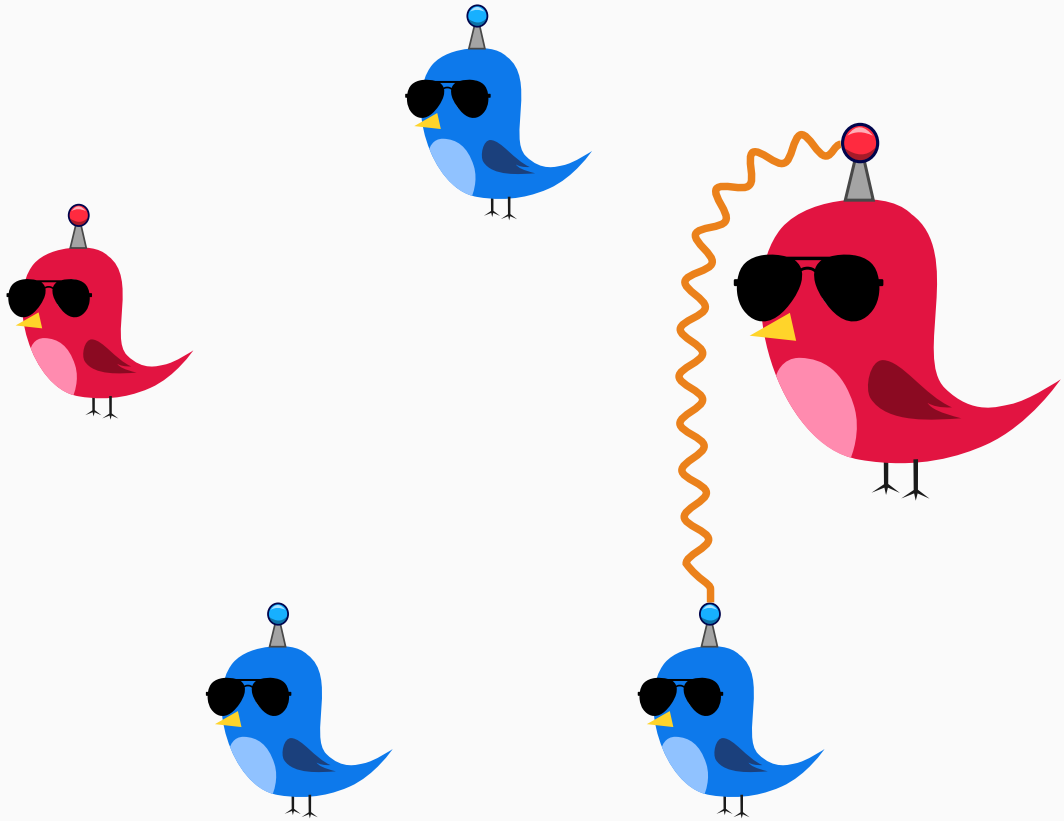
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

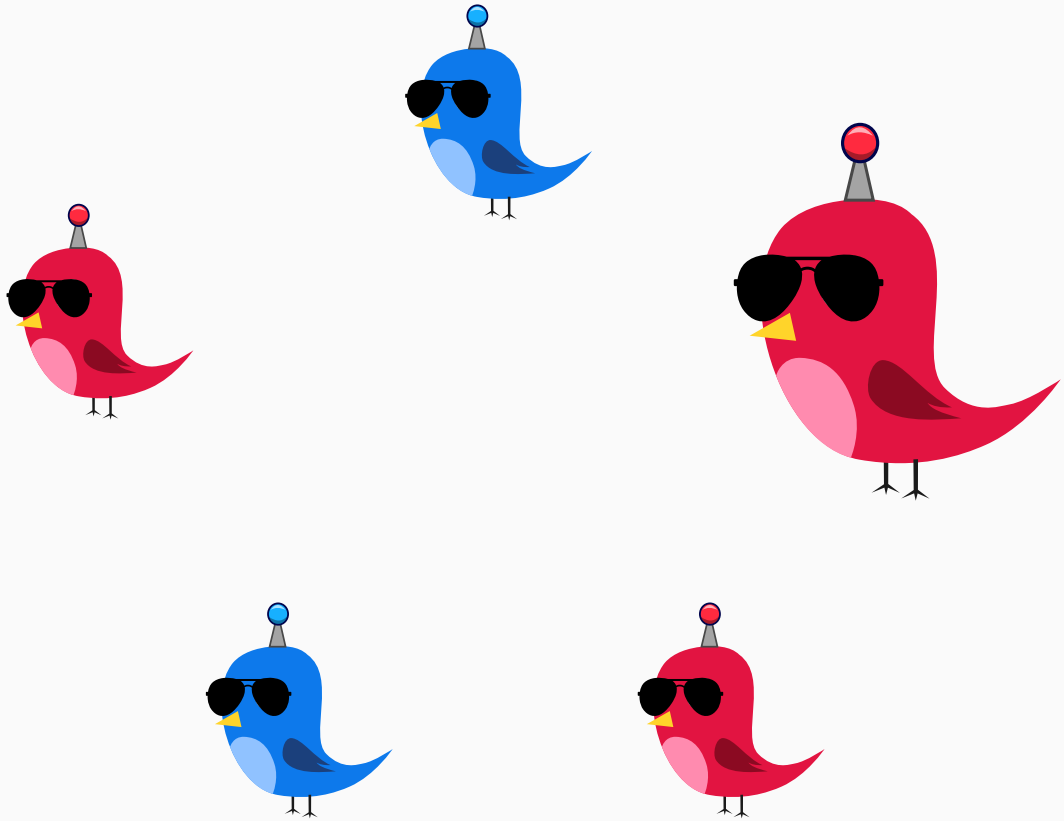
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

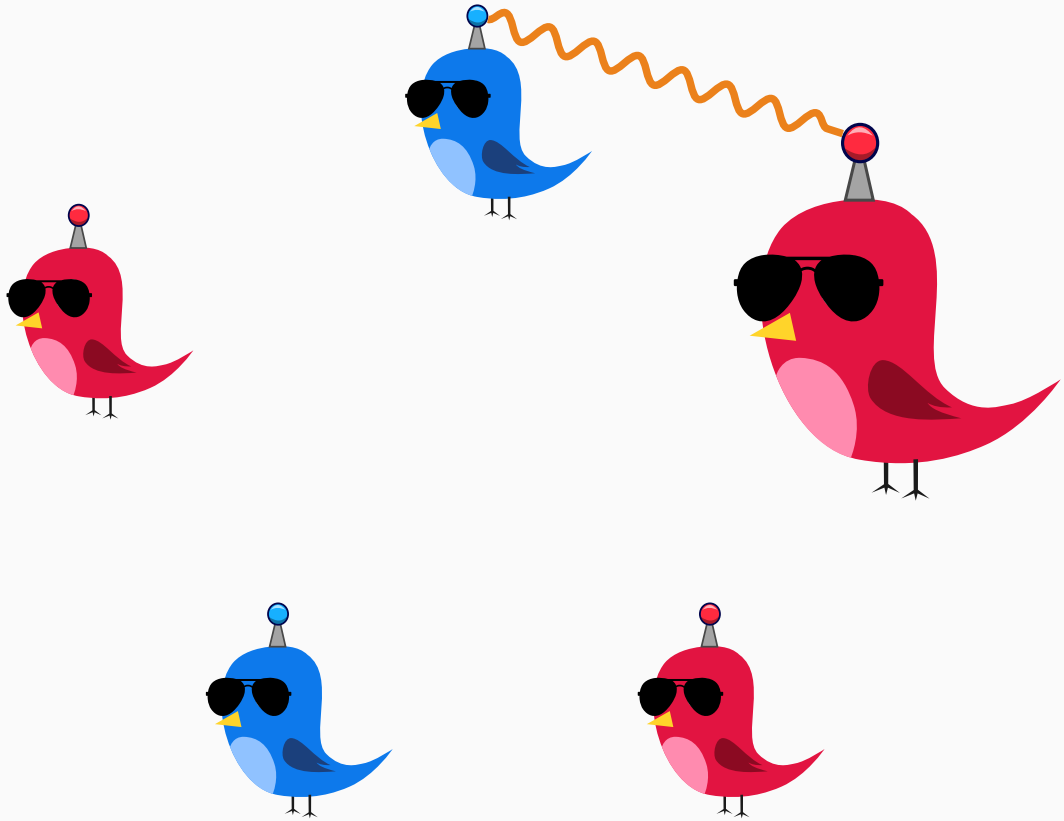
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

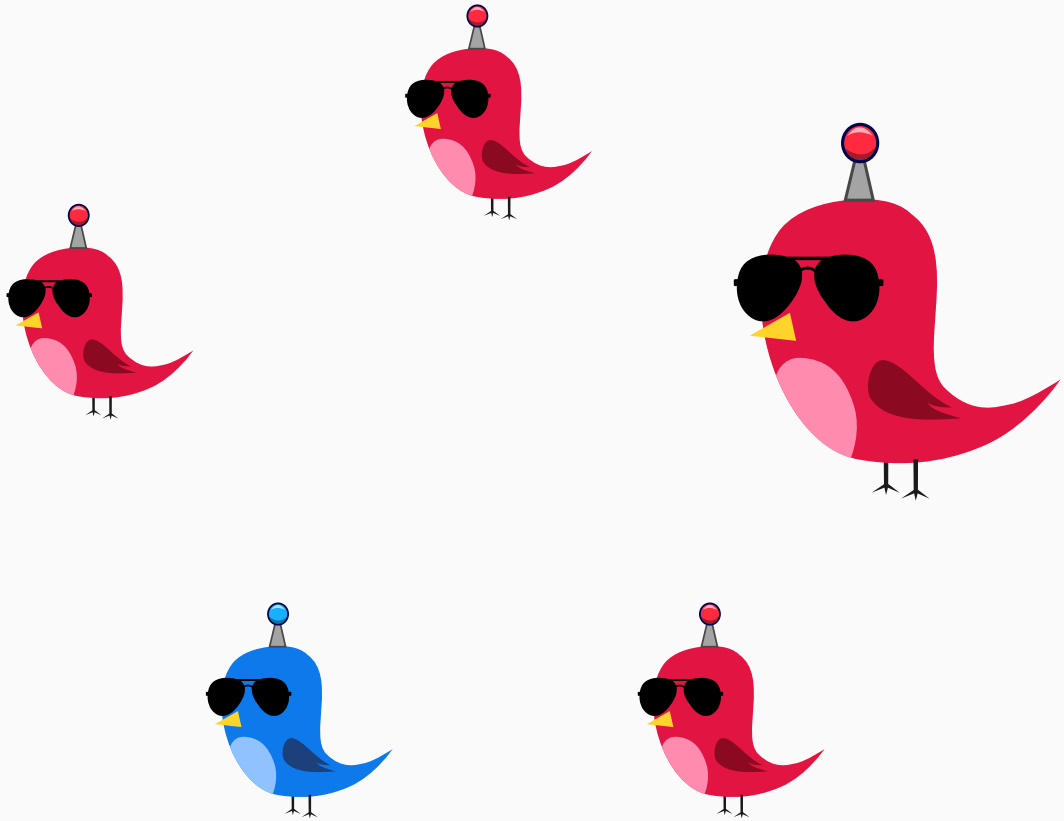
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

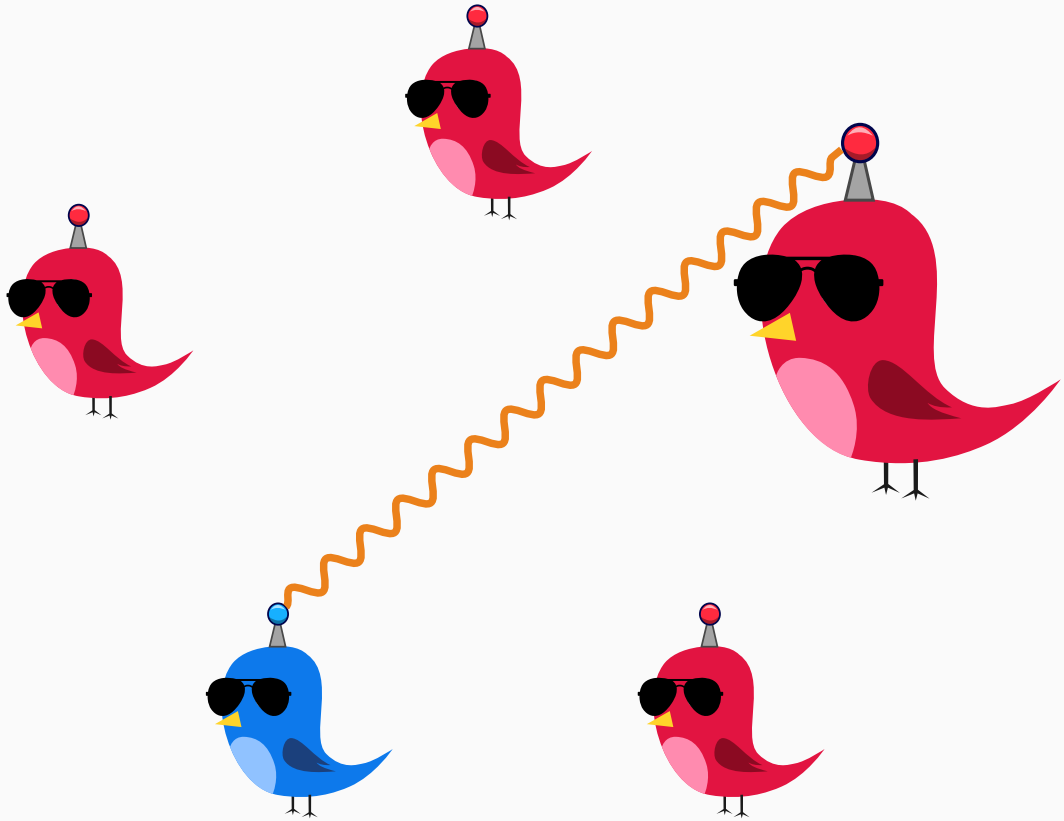
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

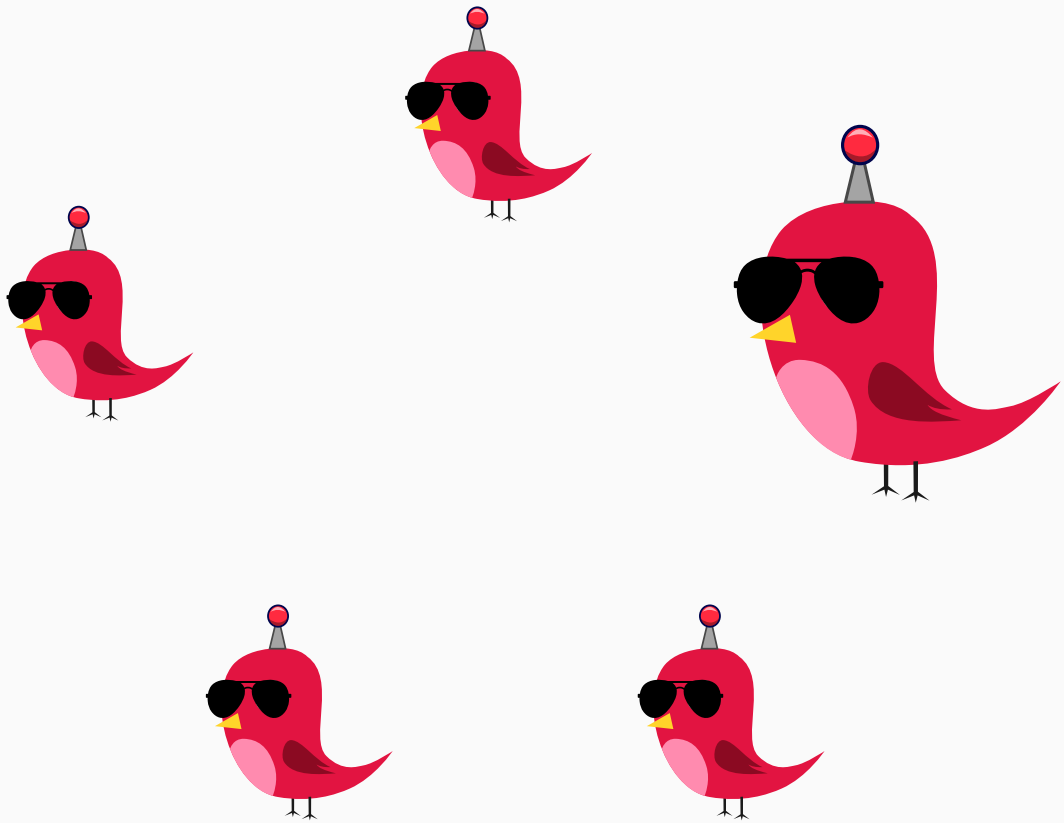
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

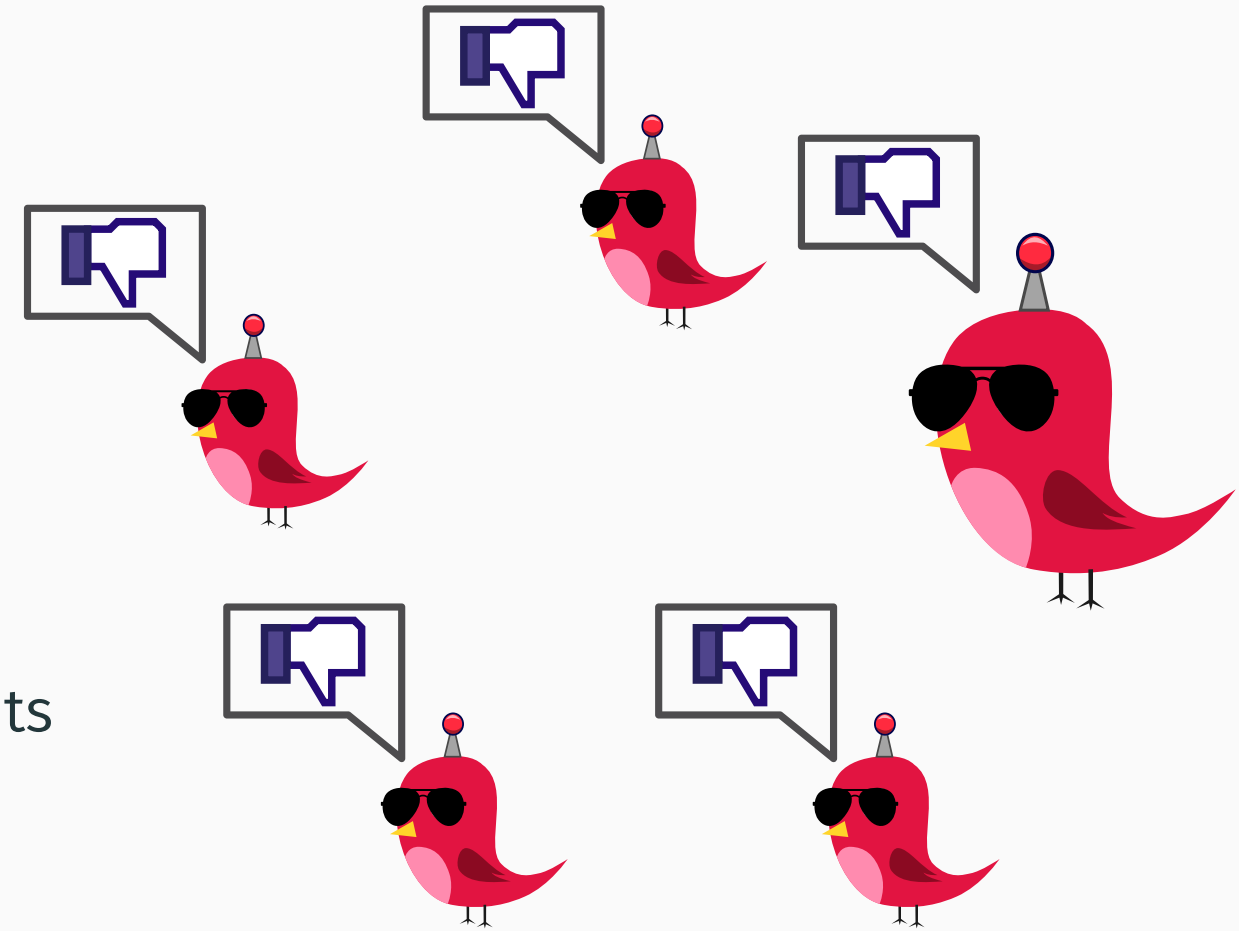
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Plus d'agents bleus que d'agents rouges ?

Interactions :

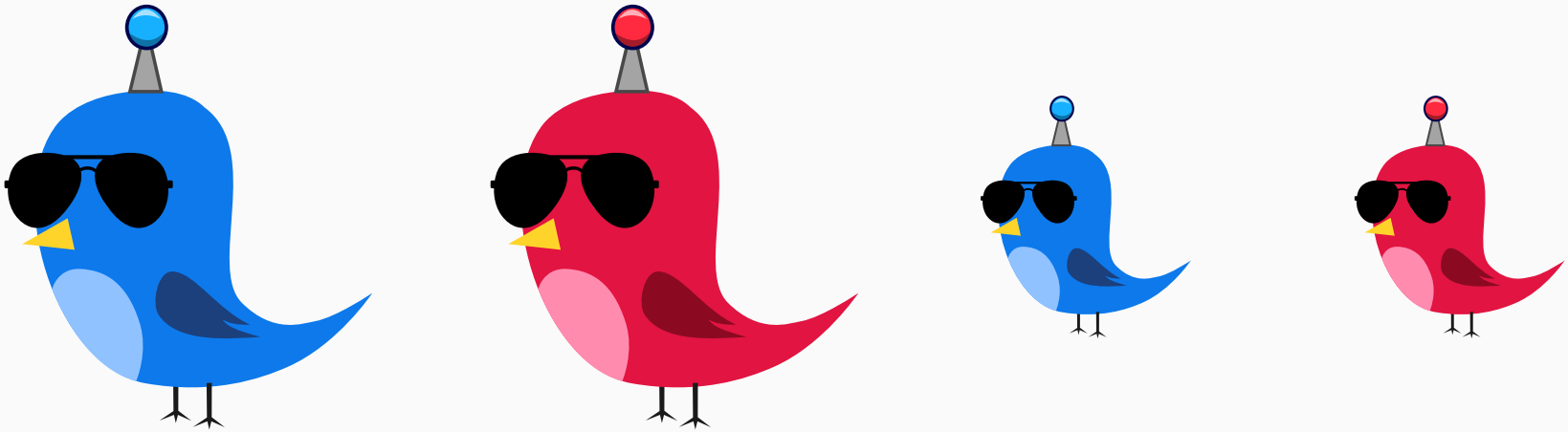
- Deux grands agents deviennent petits et bleus
- Les grands agents convertissent les petits à leur couleur



Démonstration

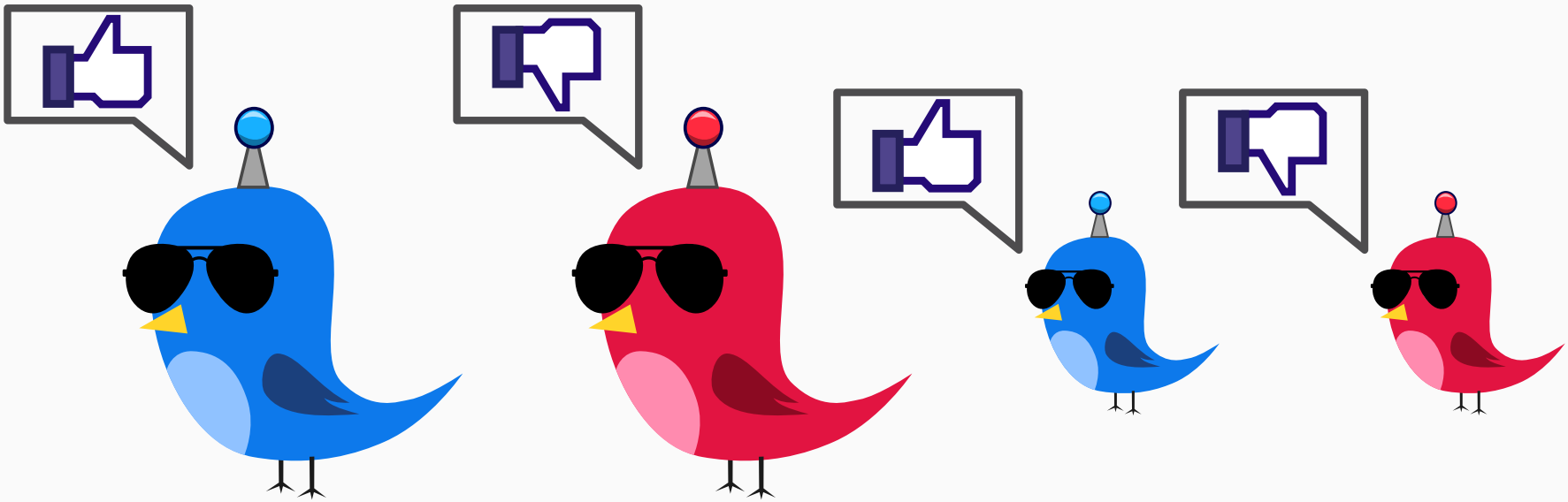
Protocoles de population: modèle formel

- *États* : ensemble fini Q
- *Opinions* : $O \colon Q \to \{\texttt{faux}, \texttt{vrai}\}$
- *États initiaux* : $I \subseteq Q$
- *Transitions* : $T \subseteq Q^2 \times Q^2$



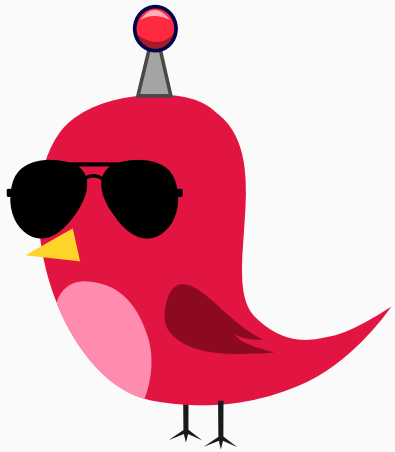
Protocoles de population: modèle formel

- *États* : ensemble fini Q
- *Opinions* : $O \colon Q \to \{\texttt{faux}, \texttt{vrai}\}$
- *États initiaux* : $I \subseteq Q$
- *Transitions* : $T \subseteq Q^2 \times Q^2$



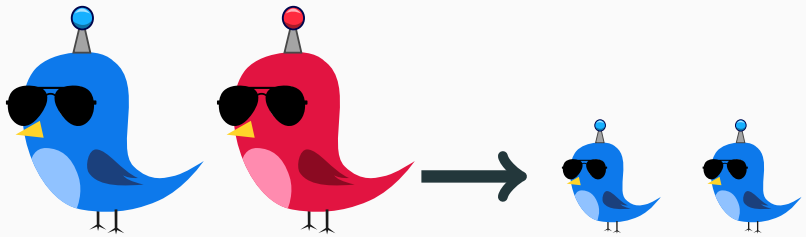
Protocoles de population: modèle formel

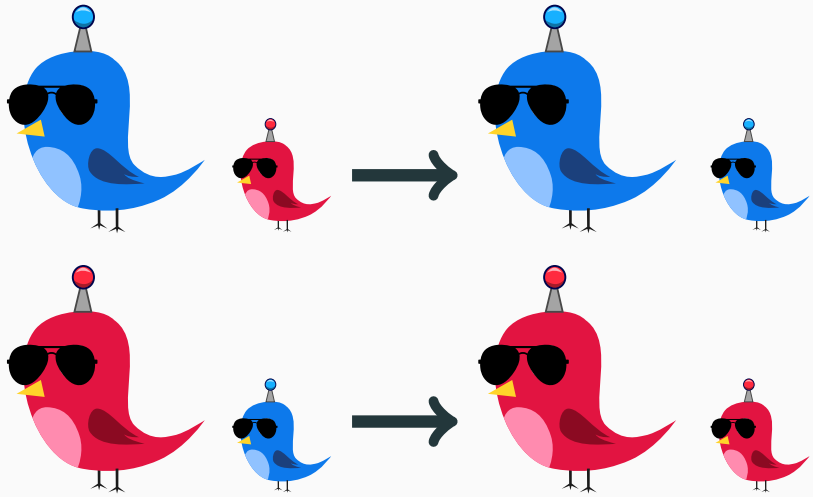
- *États* : ensemble fini Q
- *Opinions* : $O \colon Q \to \{\texttt{faux}, \texttt{vrai}\}$
- *États initiaux* : $I \subseteq Q$
- *Transitions* : $T \subseteq Q^2 \times Q^2$



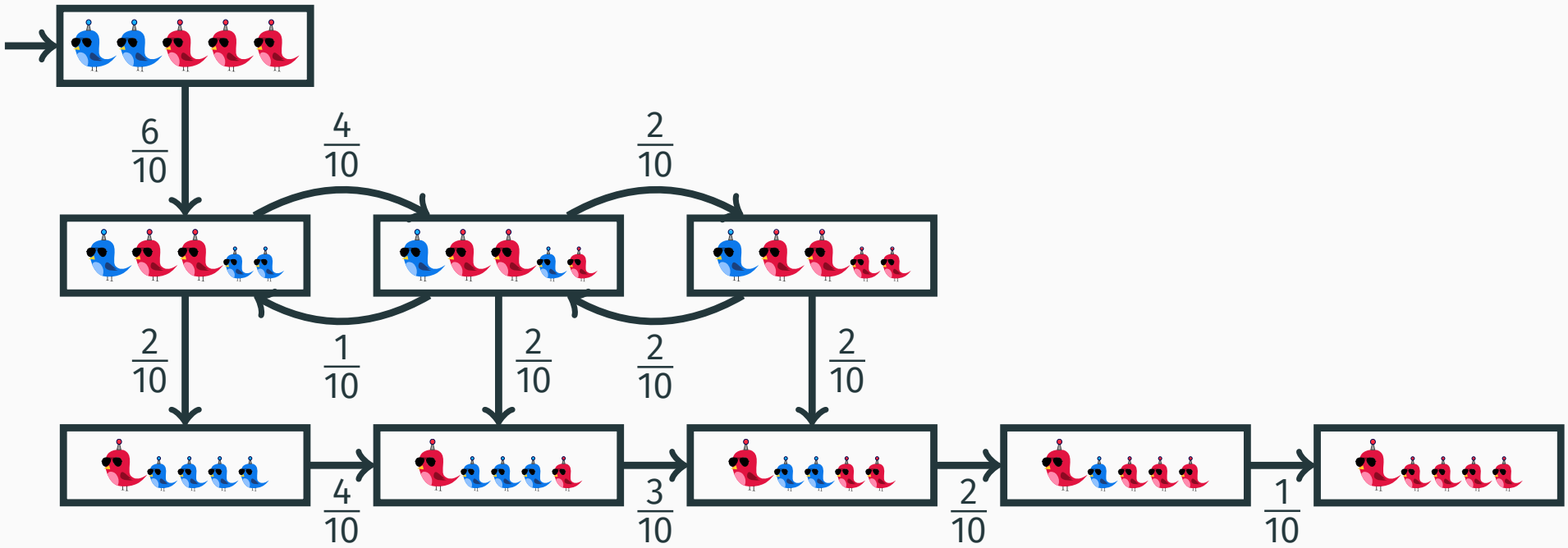
Protocoles de population: modèle formel

- *États* : ensemble fini Q
- *Opinions* : $O \colon Q \to \{\texttt{faux}, \texttt{vrai}\}$
- *États initiaux* : $I \subseteq Q$
- *Transitions* : $T \subseteq Q^2 \times Q^2$



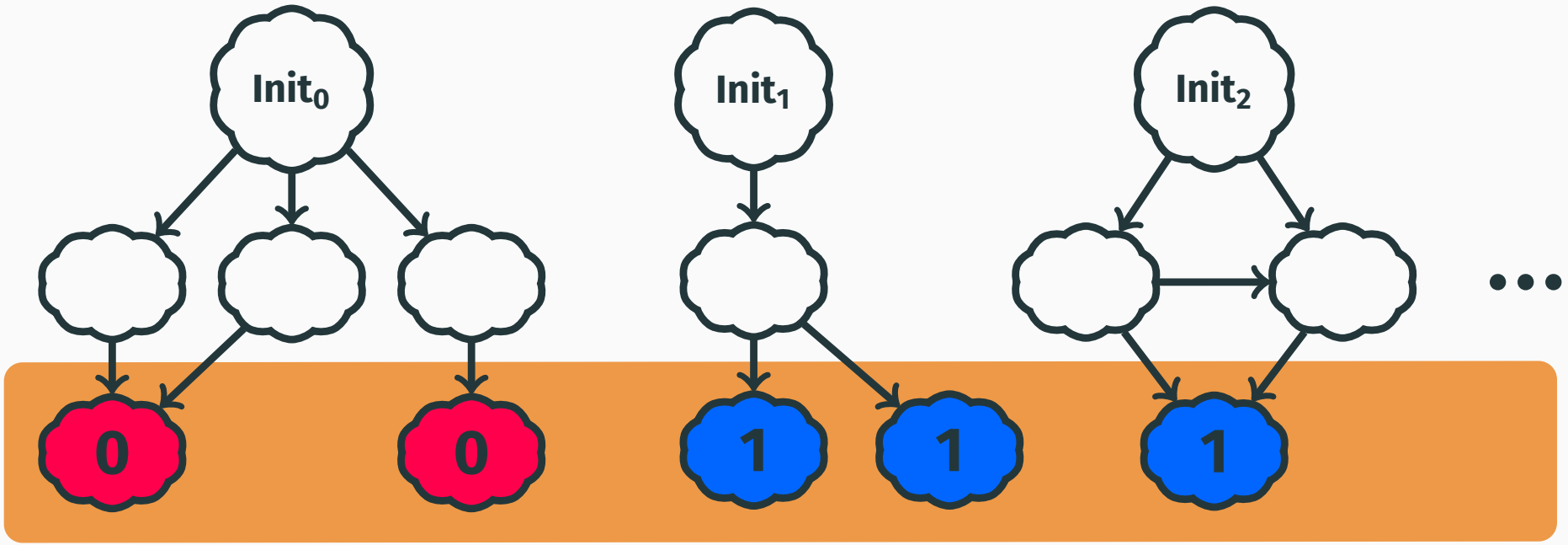


Graphe des configurations (chaîne de Markov):



Un protocole calcule un prédicat $f\colon \mathbb{N}^I \to \{0,1\}$

si les exécutions arrivent à un **consensus**
avec probabilité 1



Protocoles de population: vérification

Les protocoles deviennent complexes, même pour B ≥ R

Fast and Exact Majority in Population Protocols

Dan Alistarh, Rati Gelashvili, Milan Vojnović

February 2015

/* Functions mapping states to integers */

1 $weight(x) = \begin{cases} |x| & \text{if } x \in StrongStates \text{ or } x \in WeakStates; \\ 1 & \text{if } x \in IntermediateStates. \end{cases}$

2 $sgn(x) = \begin{cases} 1 & \text{if } x \in \{+0, 1_d, \ldots, 1_1, 3, 5, \ldots, m\}; \\ -1 & \text{otherwise.} \end{cases}$

3 $value(x) = sgn(x) \cdot weight(x)$

/* Functions for rounding state interactions */

4 $\phi(x) = -1_1$ if $x = -1$; 1_1 if $x = 1$; x, otherwise

5 $R_\downarrow(k) = \phi(k$ if k odd integer, $k-1$ if k even$)$

6 $R_\uparrow(k) = \phi(k$ if k odd integer, $k+1$ if k even$)$

7 $Shift\text{-}to\text{-}Zero(x) = \begin{cases} -1_{j+1} & \text{if } x = -1_j \text{ for some index } j < d \\ 1_{j+1} & \text{if } x = 1_j \text{ for some index } j < d \\ x & \text{otherwise.} \end{cases}$

8 $Sign\text{-}to\text{-}Zero(x) = \begin{cases} +0 & \text{if } sgn(x) > 0 \\ -0 & \text{oherwise.} \end{cases}$

9 **procedure** update$\langle x, y \rangle$

10 **if** $(weight(x) > 0$ *and* $weight(y) > 1)$ ***or*** $(weight(y) > 0$ *and* $weight(x) > 1)$ **then**

11 $x' \leftarrow R_\downarrow\left(\frac{value(x)+value(y)}{2}\right)$ and $y' \leftarrow R_\uparrow\left(\frac{value(x)+value(y)}{2}\right)$

12 **else if** $weight(x) \cdot weight(y) = 0$ *and* $value(x) + value(y) > 0$ **then**

13 **if** $weight(x) \neq 0$ **then** $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$ and $y' \leftarrow Sign\text{-}to\text{-}Zero(x)$

14 **else** $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$ and $x' \leftarrow Sign\text{-}to\text{-}Zero(y)$

15 **else if** $(x \in \{-1_d, +1_d\}$ *and* $weight(y) = 1$ *and* $sgn(x) \neq sgn(y))$ ***or***

16 $(y \in \{-1_d, +1_d\}$ *and* $weight(x) = 1$ *and* $sgn(y) \neq sgn(x))$ **then**

17 $x' \leftarrow -0$ and $y' \leftarrow +0$

18 **else**

19 $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$ and $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$

Protocoles de population: vérification

Les protocoles deviennent complexes, même pour B ≥ R

Fast and Exact Majority in Population Protocols

Dan Alistarh, Rati Gelashvili, Milan Vojnović

February 2015

/* Functions mapping states to integers */

1 $weight(x) = \begin{cases} |x| & \text{if } x \in StrongStates \text{ or } x \in WeakStates; \\ 1 & \text{if } x \in IntermediateStates. \end{cases}$

2 $sgn(x) = \begin{cases} 1 & \text{if } x \in \{+0, 1_d, \ldots, 1_1, 3, 5, \ldots, m\}; \\ -1 & \text{otherwise.} \end{cases}$

3 $value(x) = sgn(x) \cdot weight(x)$

/* Functions for rounding state interactions */

4 $\phi(x) = -1_1$ if $x = -1$; 1_1 if $x = 1$; x, otherwise

5 $R_\downarrow(k) = \phi(k$ if k odd integer, $k-1$ if k even$)$

6 $R_\uparrow(k) = \phi(k$ if k odd integer, $k+1$ if k even$)$

7 $Shift\text{-}to\text{-}Zero(x) = \begin{cases} -1_{j+1} & \text{if } x = -1_j \text{ for some index } j < d \\ 1_{j+1} & \text{if } x = 1_j \text{ for some index } j < d \\ x & \text{otherwise.} \end{cases}$

8 $Sign\text{-}to\text{-}Zero(x) = \begin{cases} +0 & \text{if } sgn(x) > 0 \\ -0 & \text{oherwise.} \end{cases}$

9 **procedure** update$\langle x, y \rangle$

10 **if** $(weight(x) > 0$ *and* $weight(y) > 1)$ ***or*** $(weight(y) > 0$ *and* $weight(x) > 1)$ **then**

11 $x' \leftarrow R_\downarrow\left(\frac{value(x)+value(y)}{2}\right)$ and $y' \leftarrow R_\uparrow\left(\frac{value(x)+value(y)}{2}\right)$

12 **else if** $weight(x) \cdot weight(y) = 0$ *and* $value(x) + value(y) > 0$ **then**

13 **if** $weight(x) \neq 0$ **then** $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$ and $y' \leftarrow Sign\text{-}to\text{-}Zero(x)$

14 **else** $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$ and $x' \leftarrow Sign\text{-}to\text{-}Zero(y)$

15 **else if** $(x \in \{-1_d, +1_d\}$ *and* $weight(y) = 1$ *and* $sgn(x) \neq sgn(y))$ ***or***

16 $(y \in \{-1_d, +1_d\}$ *and* $weight(x) = 1$ *and* $sgn(y) \neq sgn(x))$ **then**

17 $x' \leftarrow -0$ and $y' \leftarrow +0$

18 **else**

19 $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$ and $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$

Possible d'éviter les erreurs en vérifiant automatiquement qu'un protocole calcule:

- **un prédicat**
- **le prédicat attendu**

Protocoles de population: vérification

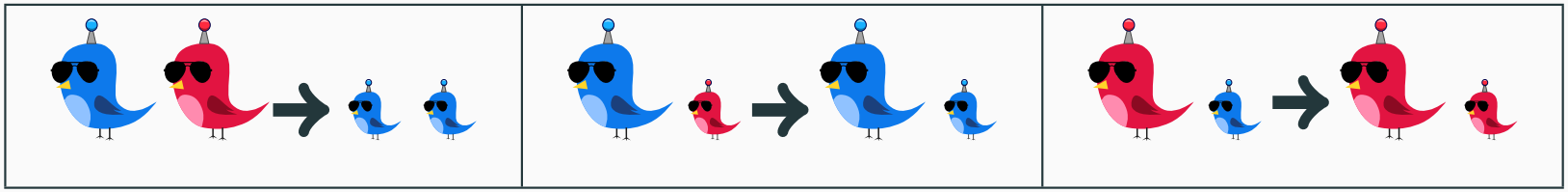
- Il existe un algorithme pour la vérification des protocoles de population (Esparza *et al.* 2015)
- Complexité : $\Omega\left(2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}\, n \text{ fois}}\right)$ (Czerwinski *et al.* 2018)
- Algorithme « non constructif »
- Les outils de vérification existants peuvent seulement vérifier des populations de taille fixe

Protocoles de population: vérification

- Il existe un algorithme pour la vérification des protocoles de population (Esparza *et al.* 2015)
- Complexité : $\Omega\left(2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}\, n \text{ fois}}\right)$ (Czerwinski *et al.* 2018)
- Algorithme « non constructif »
- Les outils de vérification existants peuvent seulement vérifier des populations de taille fixe

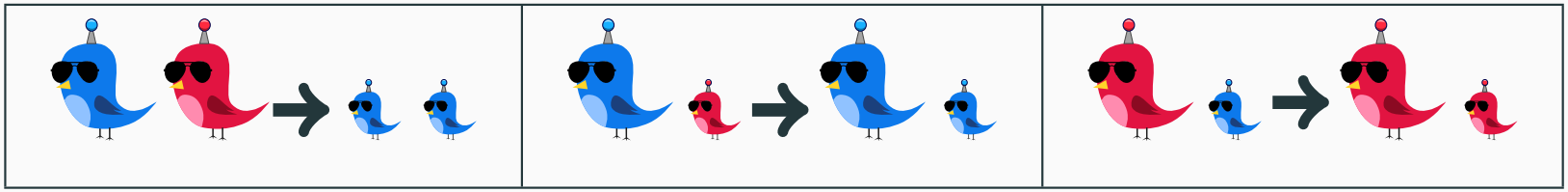
Défi : vérifier toutes les tailles dans un temps raisonnable

La plupart des protocoles sont conçus en strates:



Protocoles de population: vérification

La plupart des protocoles sont conçus en strates:



Possible de:

- Vérifier ces protocoles efficacement!
- Identifier les strates automatiquement!

Protocoles de population: vérification

Peregrine: Haskell + Z3 (Microsoft Research)

peregrine.model.in.tum.de

Vérifie qu'un protocole:

- **est sous forme de strates**
- **calcule un prédicat**
- **calcule un prédicat donné**

Protocoles de population: vérification

Protocole	Prédicat	# états	# trans.	Temps (secs.)
Majorité [a]	$x \geq y$	4	4	0.1
Diffusion [b]	$x_1 \vee \cdots \vee x_n$	2	1	0.1
Inéq. linéaire [c]	$\sum a_i x_i \geq 9$	75	2148	2376
Modulo [c]	$\sum a_i x_i = 0 \bmod 70$	72	2555	3177
Seuil [d]	$x \geq 50$	51	1275	182
Seuil [b]	$x \geq 325$	326	649	3471
Seuil [e]	$x \geq 10^7$	37	155	19

[a] Draief *et al.* 2012
[b] Clément *et al.* 2011
[c] Angluin *et al.* 2006
[d] Chatzigiannakis *et al.* 2010
[e] Offtermatt 2017

Protocoles de population: vérification

Par ex. si taille = 1000 :

PRISM prend 1 heure pour vérifier 1 configuration sur 499 500

Protocole	Prédicat	# états	# trans.	Temps (secs.)
Majorité [a]	$x \geq y$	4	4	0.1
Diffusion [b]	$x_1 \vee \cdots \vee x_n$	2	1	0.1
Inéq. linéaire [c]	$\sum a_i x_i \geq 9$	75	2148	2376
Modulo [c]	$\sum a_i x_i = 0 \bmod 70$	72	2555	3177
Seuil [d]	$x \geq 50$	51	1275	182
Seuil [b]	$x \geq 325$	326	649	3471
Seuil [e]	$x \geq 10^7$	37	155	19

[a] Draief *et al.* 2012
[b] Clément *et al.* 2011
[c] Angluin *et al.* 2006
[d] Chatzigiannakis *et al.* 2010
[e] Offtermatt 2017

Démonstration

3. Un peu de publicité

Publicité

IGL752 – Techniques de vérification et de validation

- Offert à la prochaine session (Hiver 2019)
- Offert à tous les cycles d'études
- Préalables: MAT115 et IFT436
- Inscrivez-vous!

Publicité

Projets de recherche

- Vérification efficace de systèmes infinis
- Automates, logique, algorithmique, complexité, etc.
- Cours connexes: IFT313, IGL501, IFT503
- Collaborations:

Merci !