# Optimisation d'un algorithme de calcul de bornes de latences pour accélérer la génération de preuves

Florian Delage

La Prépa des INP – Nancy

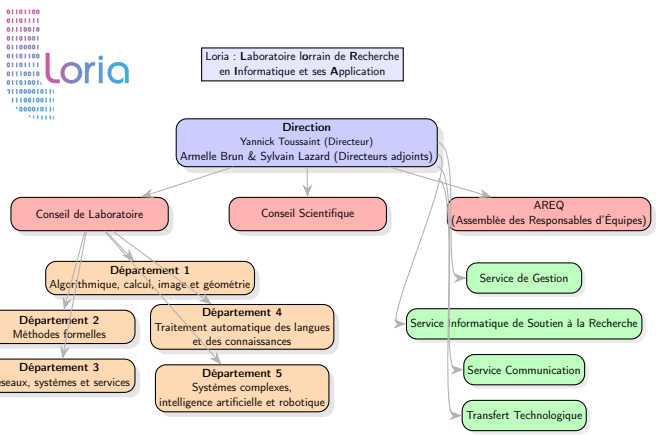
13 juin 2025

# Table des Matières

# Table des Matières

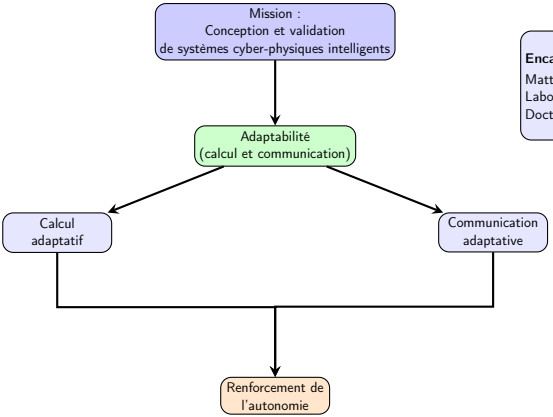
# Contexte

# Le Loria




- Créé en 1997
- 28 équipes
- 5 départements
- 500 personnes

# Simbiot




**Encadrant 1**
Matthieu Amet
Laboratoire : Loria (UL)
Doctorant

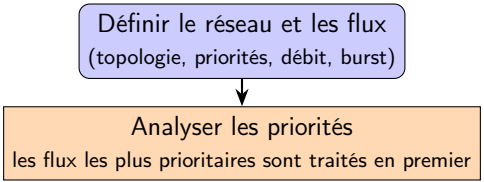
**Encadrant 2**
Ludovic Thomas
Laboratoire : Loria (CNRS)
Chargé de Recherche

# Table des Matières

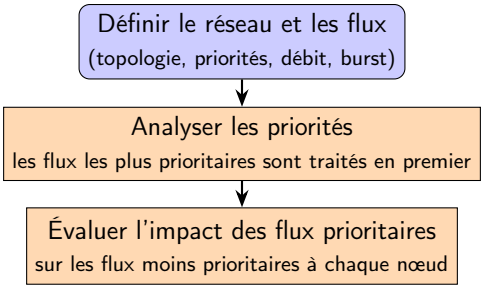
# Algorithme TFA

Définir le réseau et les flux
(topologie, priorités, débit, burst)

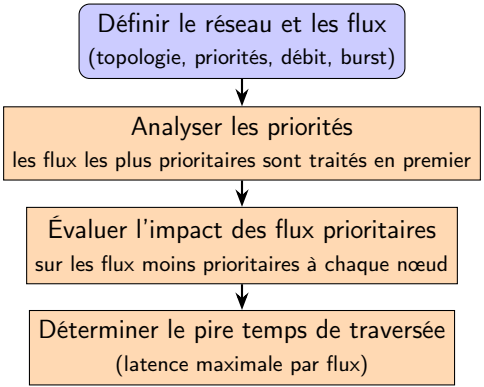
# Algorithme TFA



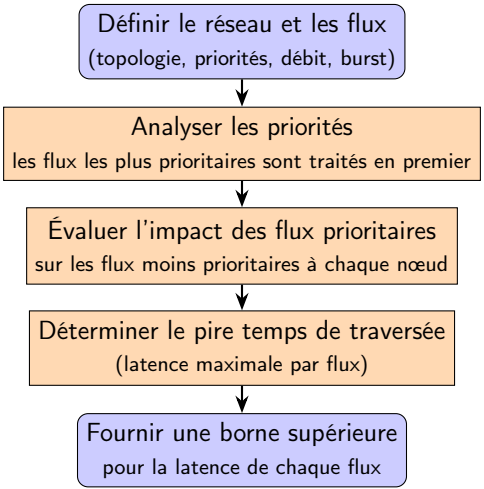
# Algorithme TFA



# Algorithme TFA



# Algorithme TFA

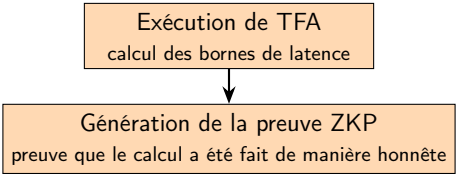


# Zero-Knowledge Proofs (ZKPs)

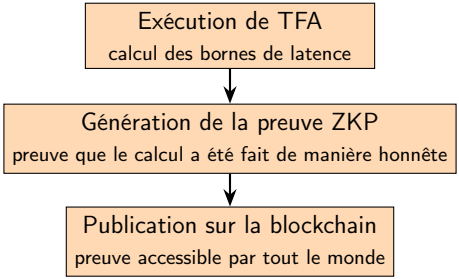
# Utilité des preuves

Exécution de TFA
calcul des bornes de latence

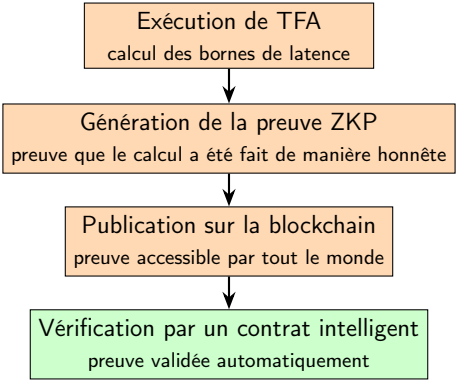
# Utilité des preuves



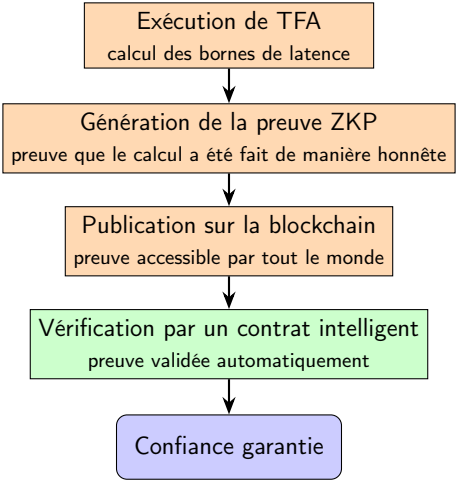
# Utilité des preuves



# Utilité des preuves



# Utilité des preuves



# Table des Matières

# Conversion des flottants vers des entiers

**Entiers**

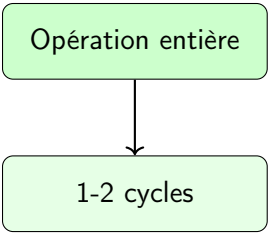
Opération entière

**Flottants**

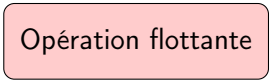
Opération flottante

# Conversion des flottants vers des entiers

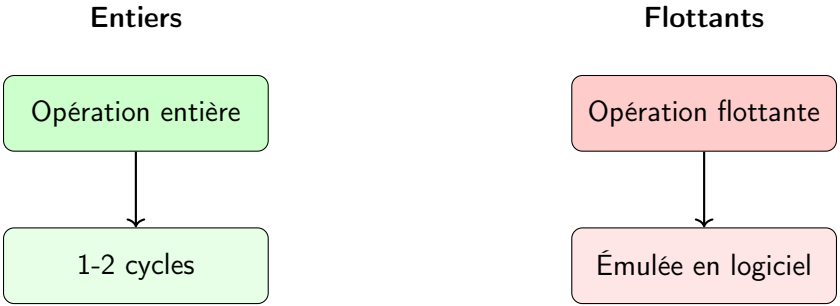
**Entiers**



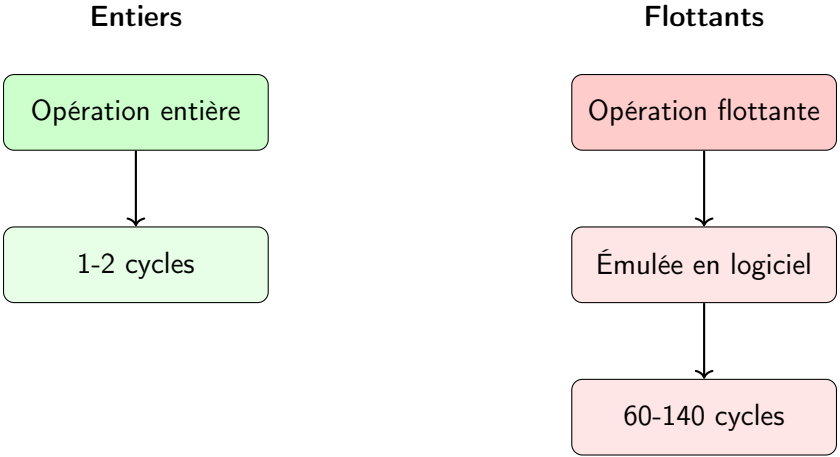

**Flottants**



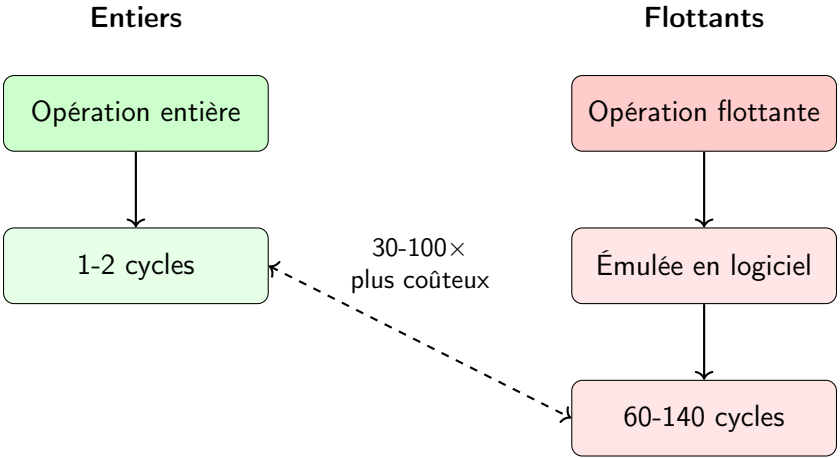
# Conversion des flottants vers des entiers



# Conversion des flottants vers des entiers

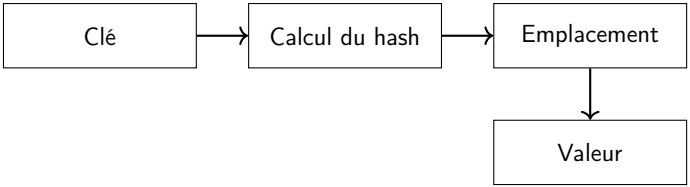


# Conversion des flottants vers des entiers



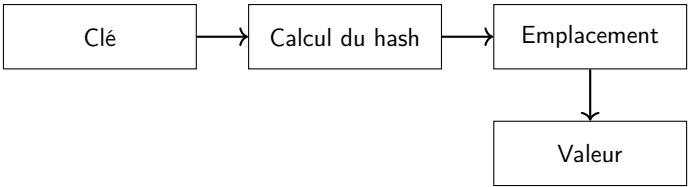
# Structures de hachage : HashMap et HashSet

**HashMap**

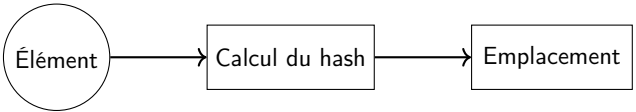


# Structures de hachage : HashMap et HashSet

**HashMap**

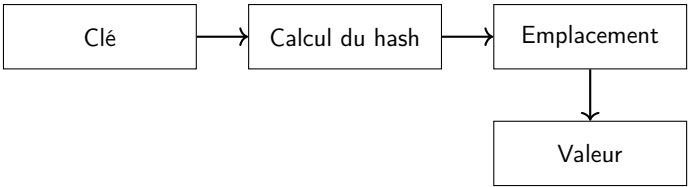



**HashSet**

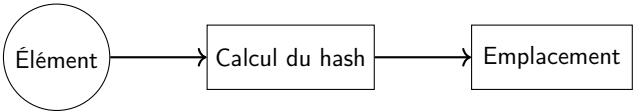


# Structures de hachage : HashMap et HashSet

**HashMap**

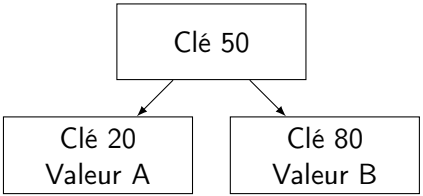



**HashSet**



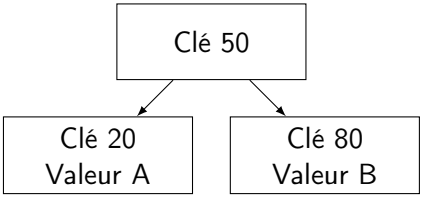

- **HashMap** associe des clés à des valeurs.
- **HashSet** stocke uniquement les valeurs.
- Tous deux utilisent une fonction de hachage.

**BTreeMap**

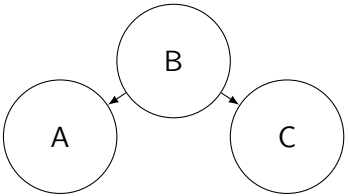


# Structures arborescentes : BTreeMap et BTreeSet

**BTreeMap**

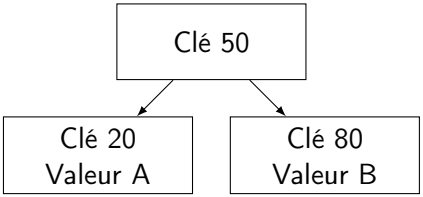



**BTreeSet**

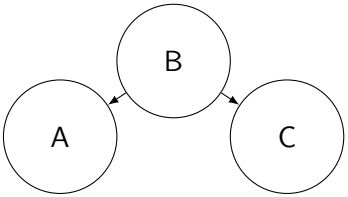


# Structures arborescentes : BTreeMap et BTreeSet

**BTreeMap**




**BTreeSet**




**Résumé :**

- **BTreeMap** trie les clés avec leurs valeurs.
- **BTreeSet** trie seulement les valeurs.
- Recherche et insertion par comparaison.

# Réduction du nombre de variables

**Code non optimisé**

**Code optimisé**

# Réduction du nombre de variables

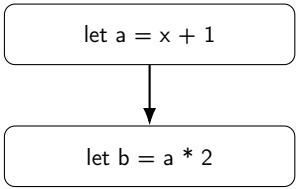
**Code non optimisé**

```
let a = x + 1
```

**Code optimisé**

# Réduction du nombre de variables

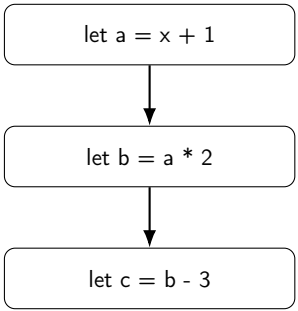
**Code non optimisé**




**Code optimisé**

# Réduction du nombre de variables

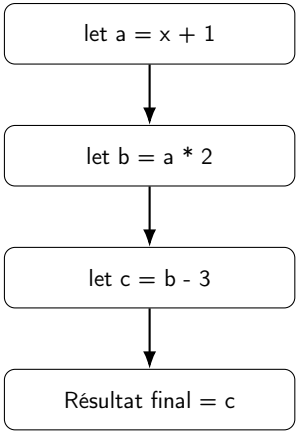
**Code non optimisé**




**Code optimisé**

# Réduction du nombre de variables

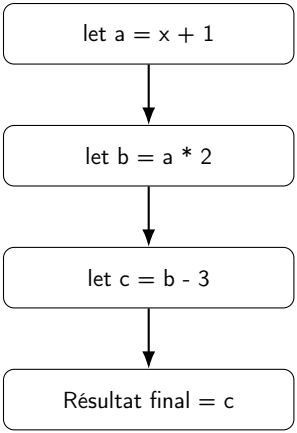
**Code non optimisé**




**Code optimisé**

# Réduction du nombre de variables

**Code non optimisé**

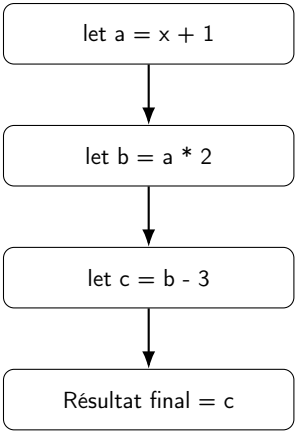



**Code optimisé**

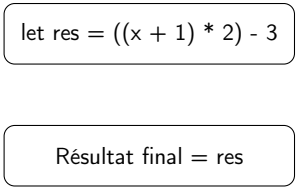
```
let res = ((x + 1) * 2) - 3
```

# Réduction du nombre de variables

**Code non optimisé**




**Code optimisé**



# Mémoïsation

# Table des Matières

# Table des Matières

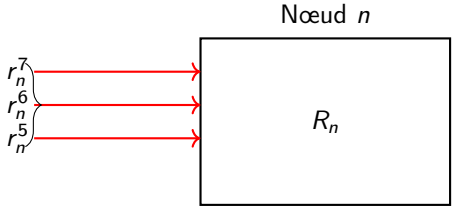
# Table des Matières

# Table des Matières

**Principe.** L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

**Principe.** L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.

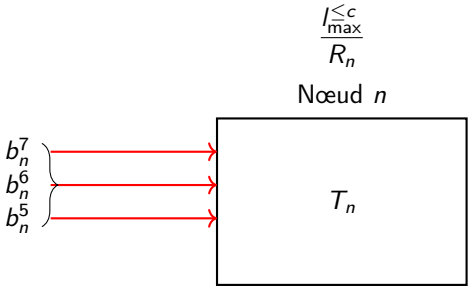



**Débit restant** pour $c = 4$ :

$$R_n^c = R_n - \sum_{c'>c} r_n^{c'}$$

**Principe.** L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.

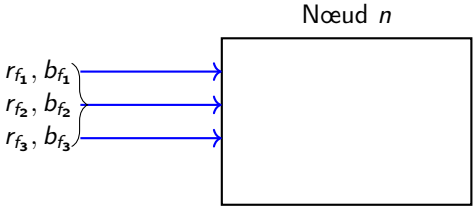



**Latence de service** pour $c = 4$ :

$$T_n^c = T_n + \sum_{c' > c} \frac{b_n^{c'}}{R_n^c} + \frac{I_{\max}^{\leq c}}{R_n}$$

**Principe.** L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.
- Calcule le **débit et le burst**.

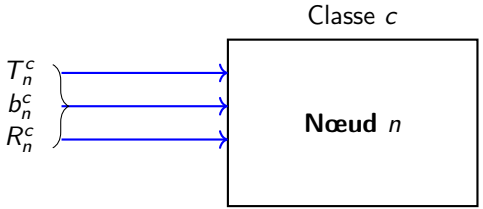



**Débit et burst** pour $c$ :

$$r_n^c = \sum_{f \in c} r_f \qquad b_n^c = \sum_{f \in c} b_f$$

**Principe.** L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.
- Calcule le **débit et le burst**.
- Calcule la **borne de latence**




**Borne de latence pour $c$ :**

$$D_n^c = T_n^c + \frac{b_n^c}{R_n^c}$$

**Principe.** L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.
- Calcule le **débit et le burst**.
- Calcule la **borne de latence**
- Propage le **burst** aux nœuds suivants.