# Ordonnancement sous contraintes: de la théorie à la pratique

Claude-Guy Quimper


En collaboration avec

THALES

# Mise en contexte

- Thales offre un contrat pour une preuve de concept d'un programme pouvant résoudre des problèmes d'ordonnancement.

- Il ne s'agit pas d'un contrat de recherche.

- Thales a de l'expertise en optimisation, mais pas l'équipe de Québec.

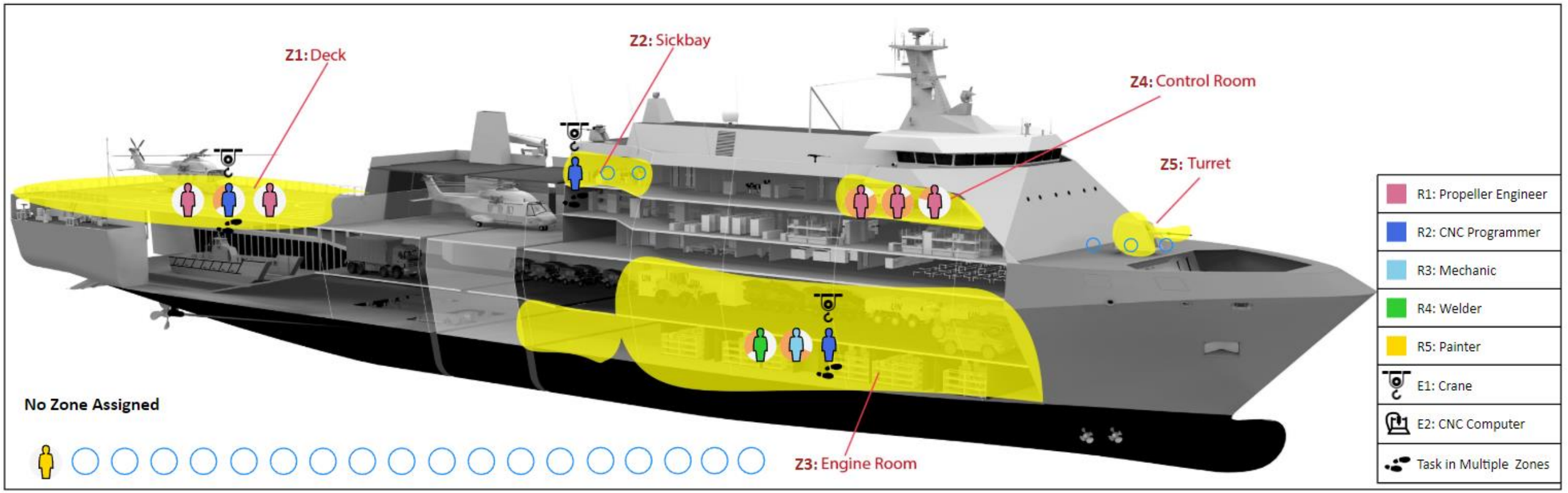
- On veut plusieurs approches: 3 contrats seront octroyés à trois universités.

# Plan

- Refit Optimizer
- Quelle méthode de résolution choisir?
- Choix technologique

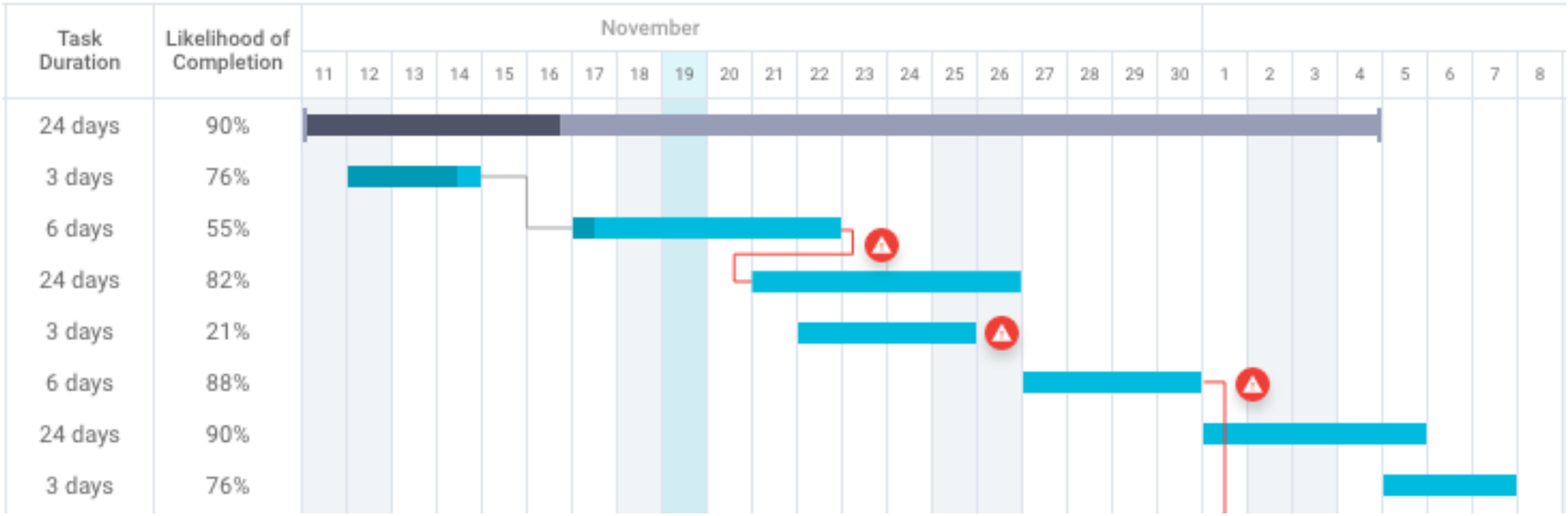
# Refit Optimizer

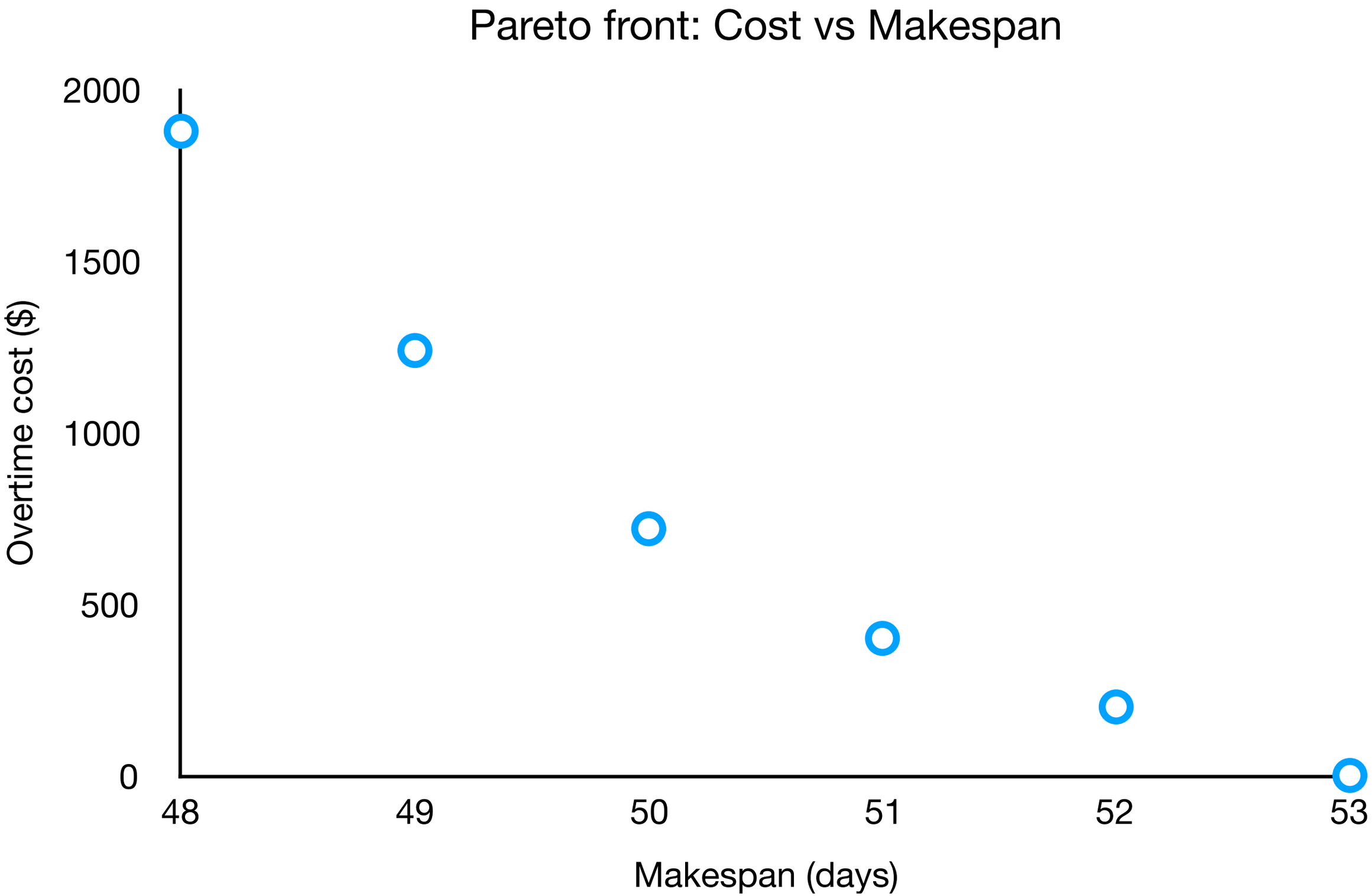
# Refit Optimizer




SIMWELL
Your Simulation Technology Partner

# Refit Optimizer



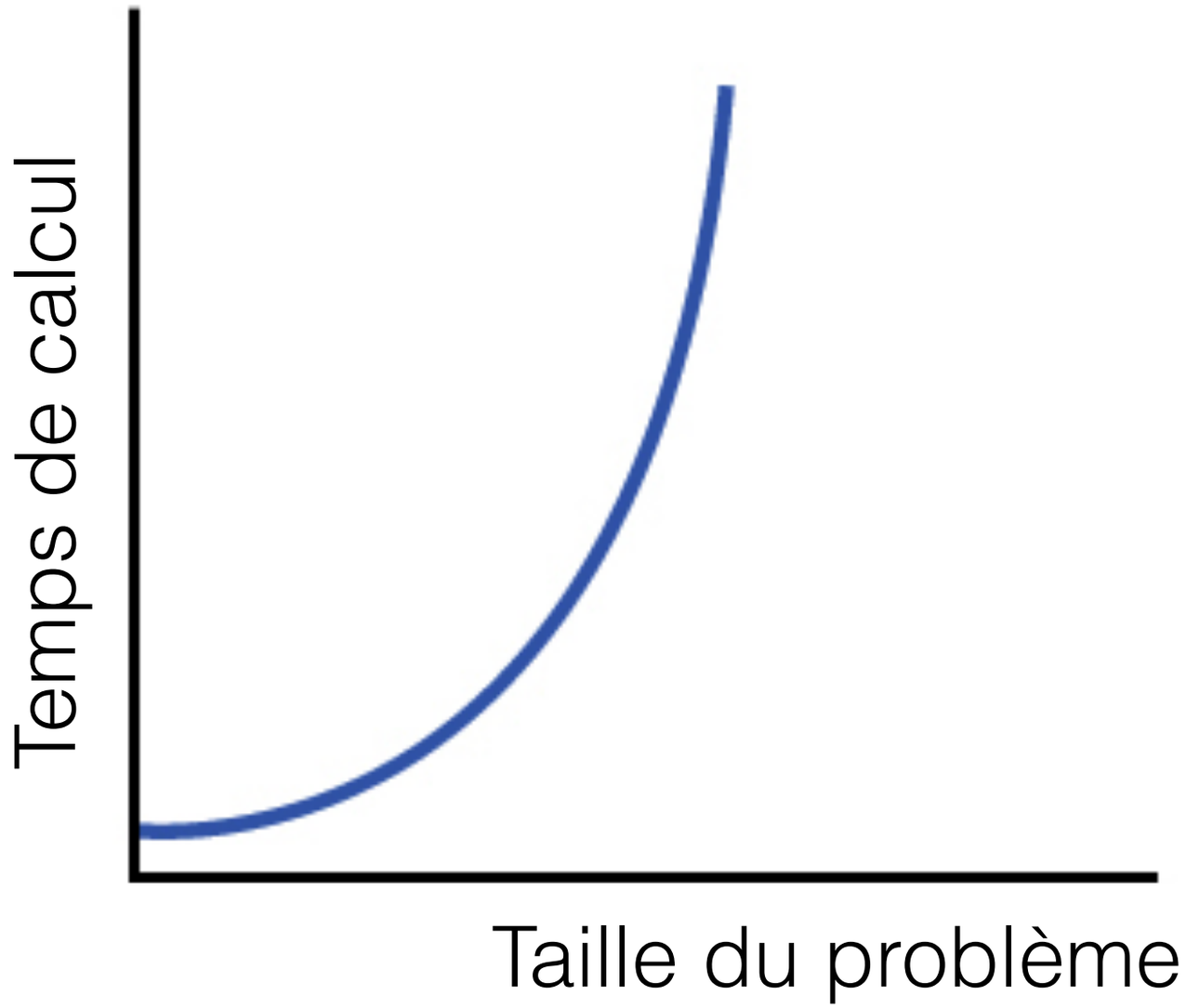

Pareto front: Cost vs Makespan
Overtime cost ($)
2000
1500
1000
500
0
48
49
50
51
52
53
Makespan (days)

# En résumé

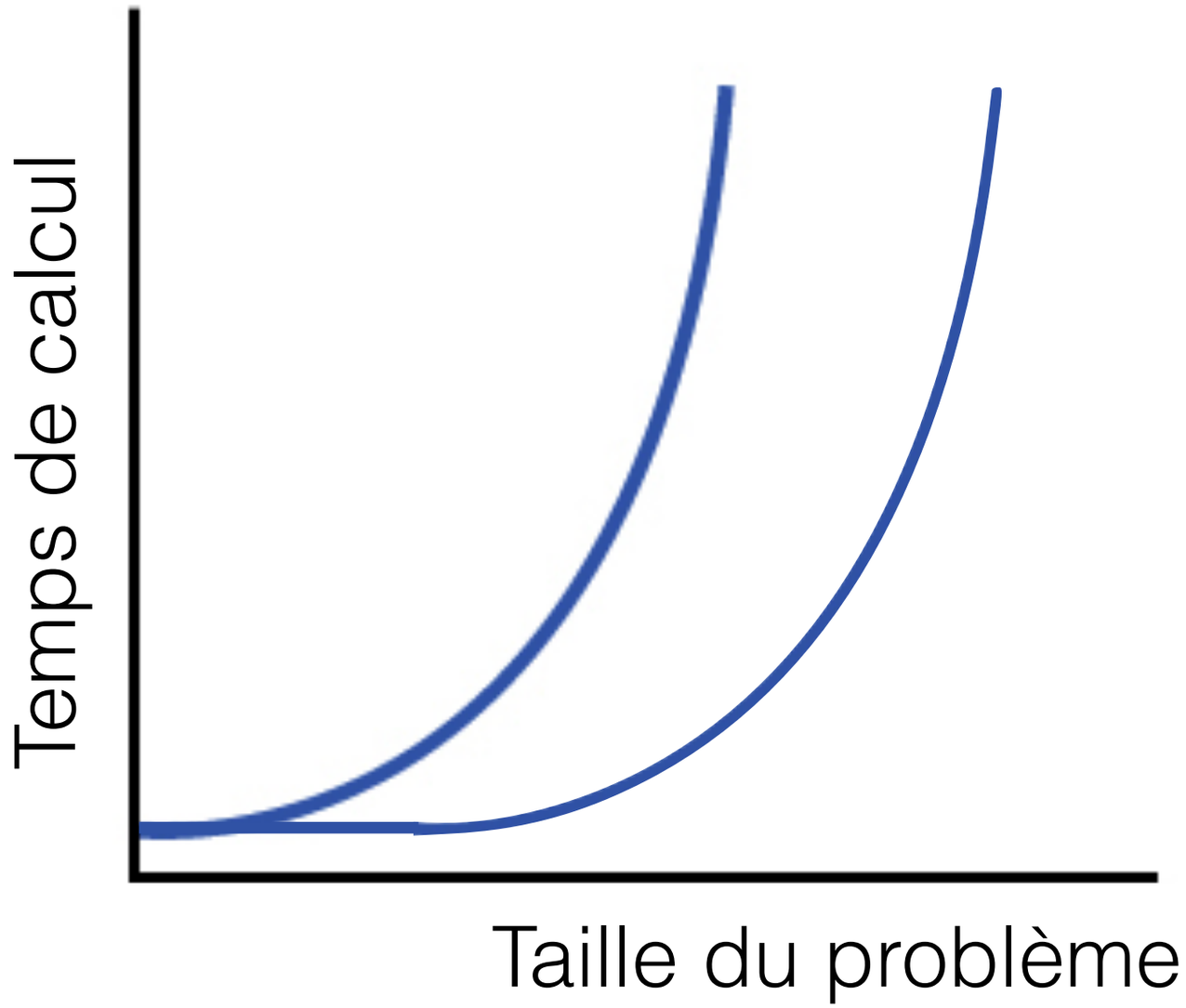
- Resource Constrained Project Scheduling System
- Plusieurs ressources cumulatives
- Contraintes de prédécesseurs
- Échéances
- Contraintes de calendriers
- Objectifs:
  - Makespan
  - Coût en heures supplémentaires
  - Robustesse

# Comment gérer les attentes?

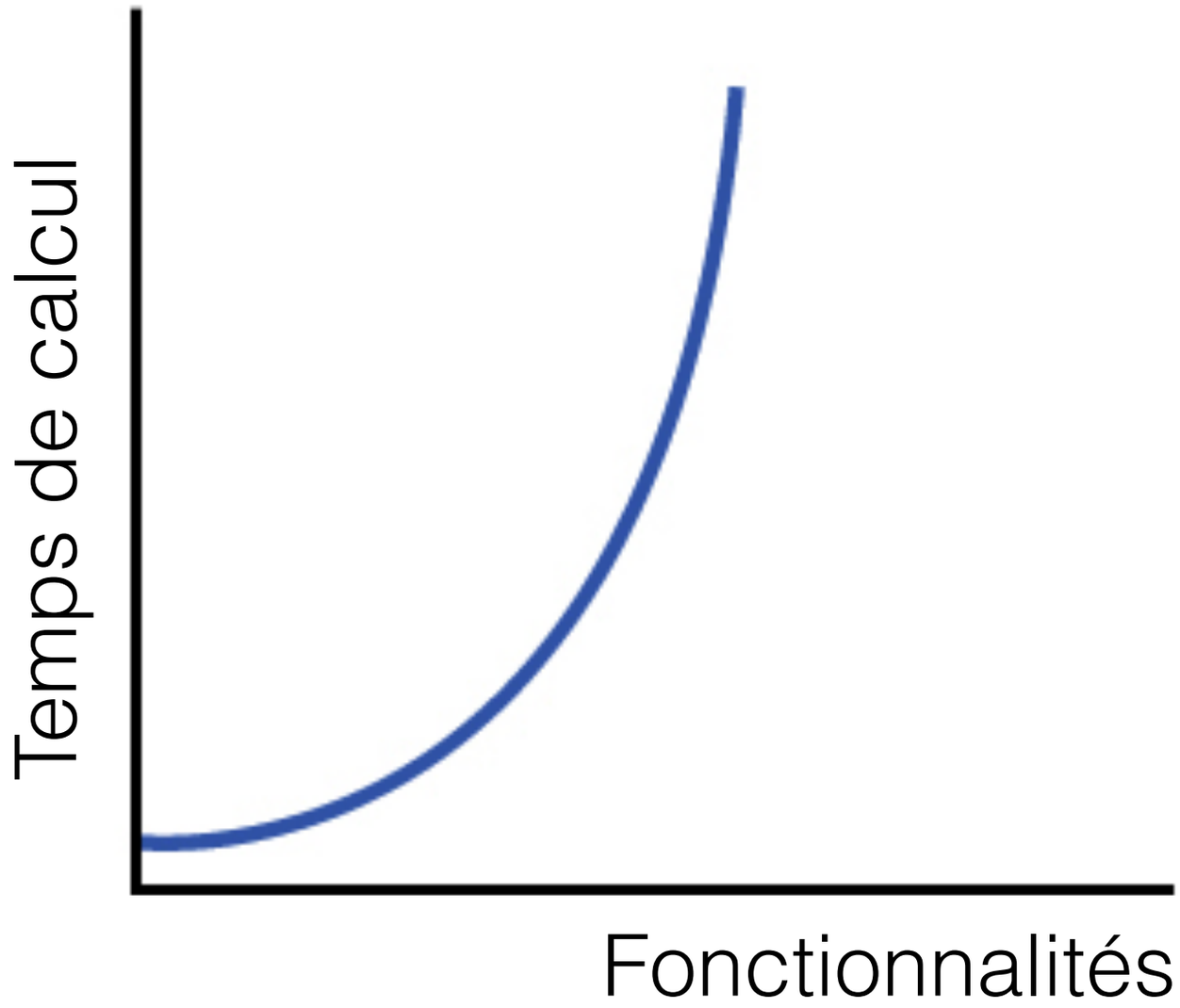
# Comment gérer les attentes?



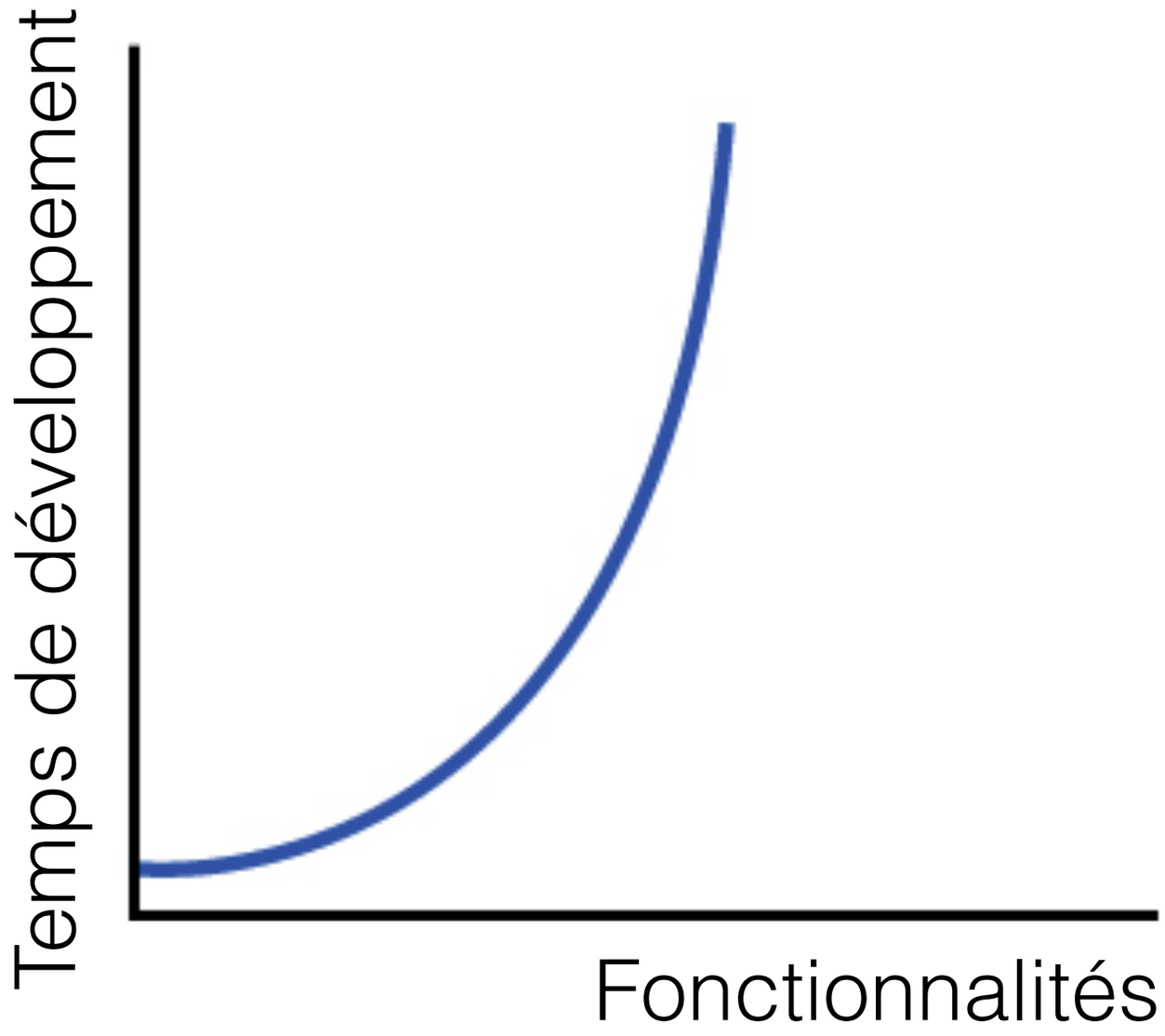
# Comment gérer les attentes?



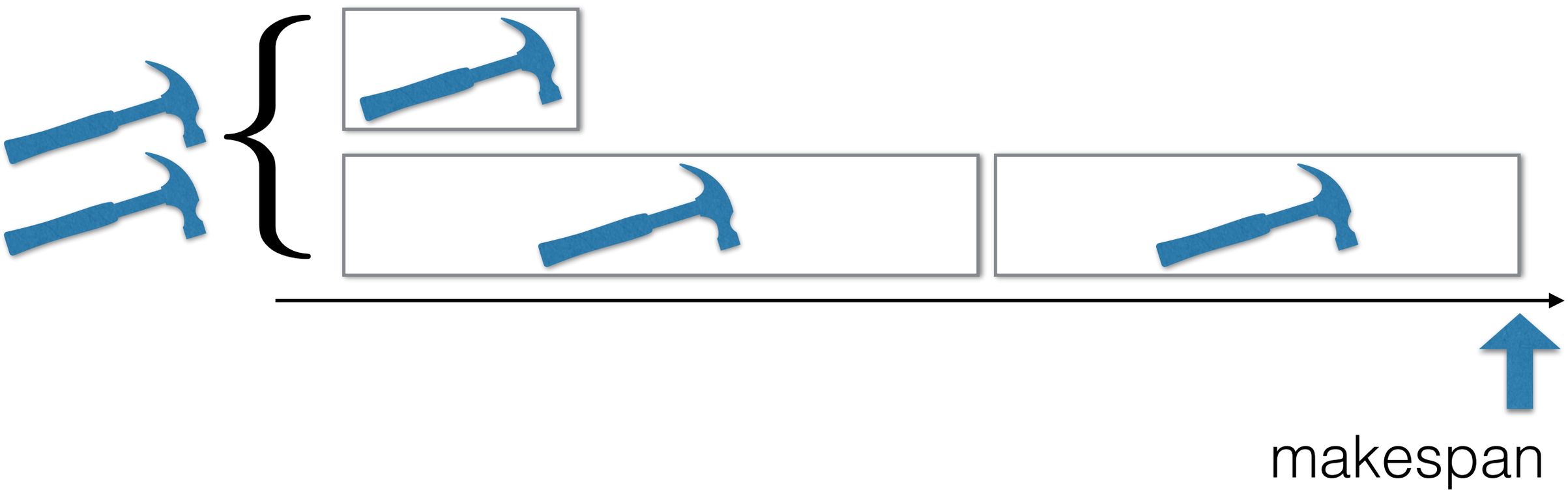
# Comment gérer les attentes?



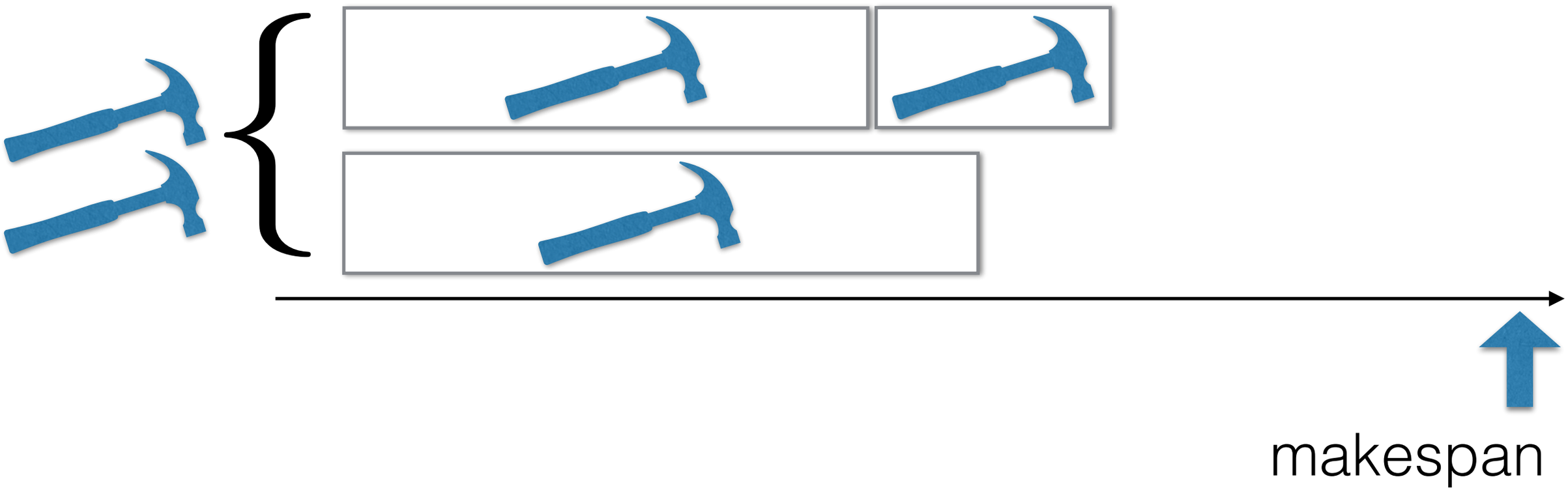
# Comment gérer les attentes?



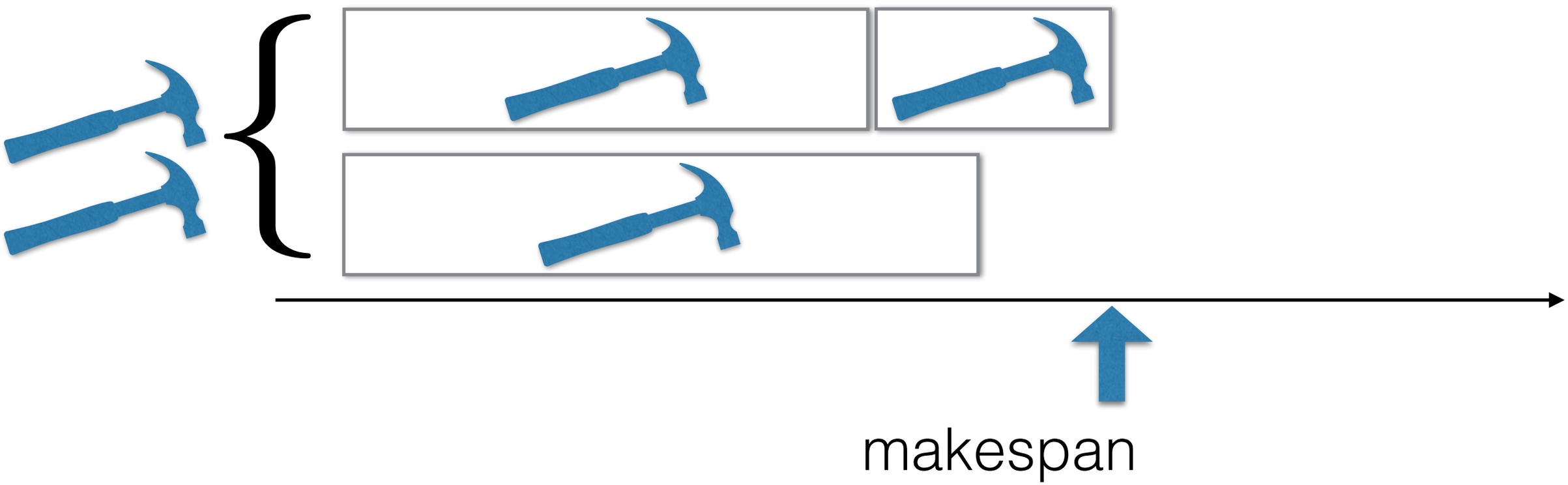
# Simplification du problème



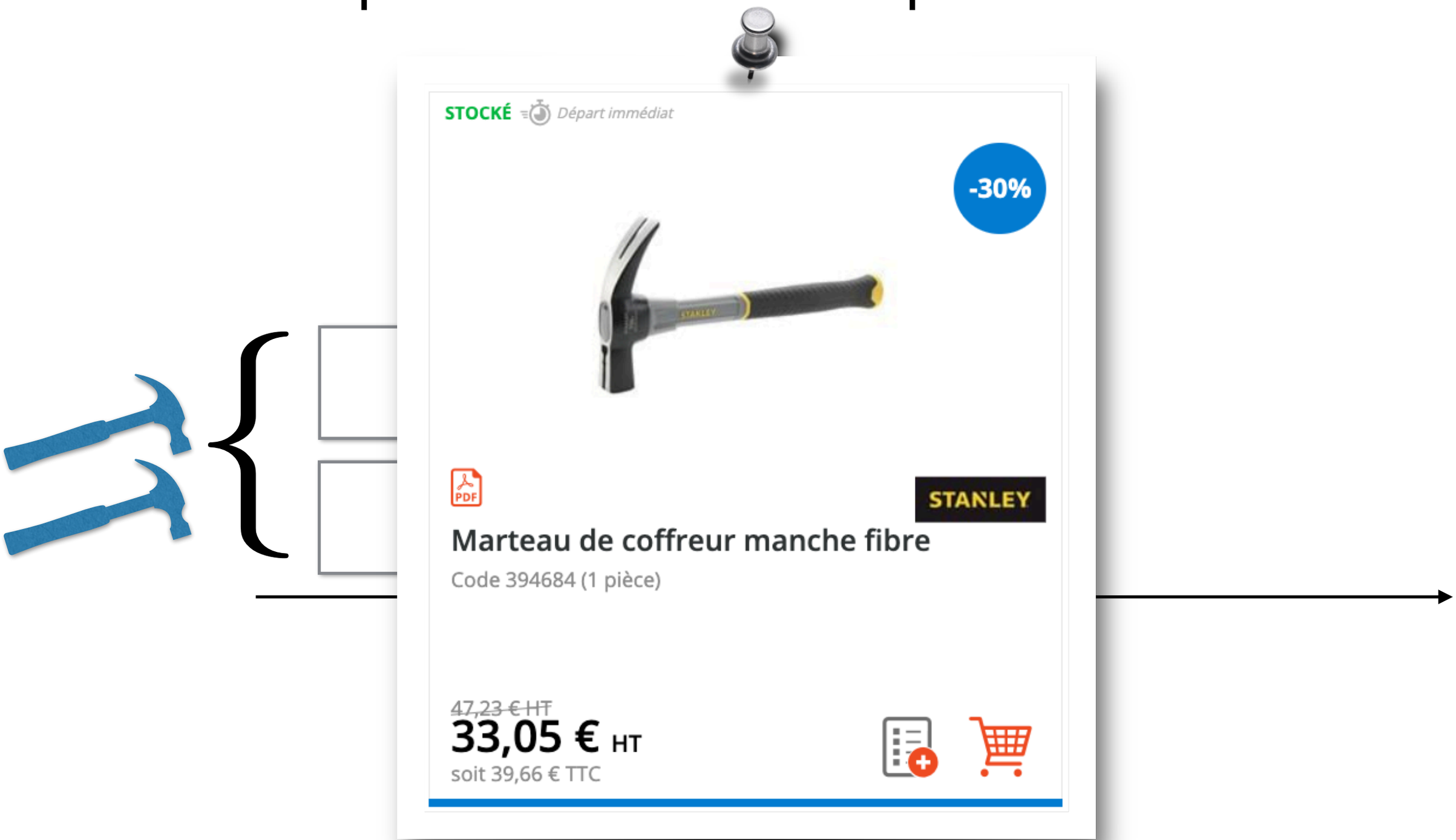
# Simplification du problème



# Simplification du problème



# Simplification du problème



# Simplification du problème

- Toutes les tâches n’ont pas à être ordonnancées
- Toutes les ressources n’ont pas à être considérées
- Les durées des tâches peuvent être arrondies

# Quelle méthode de résolution choisir?

# Choix de la méthode

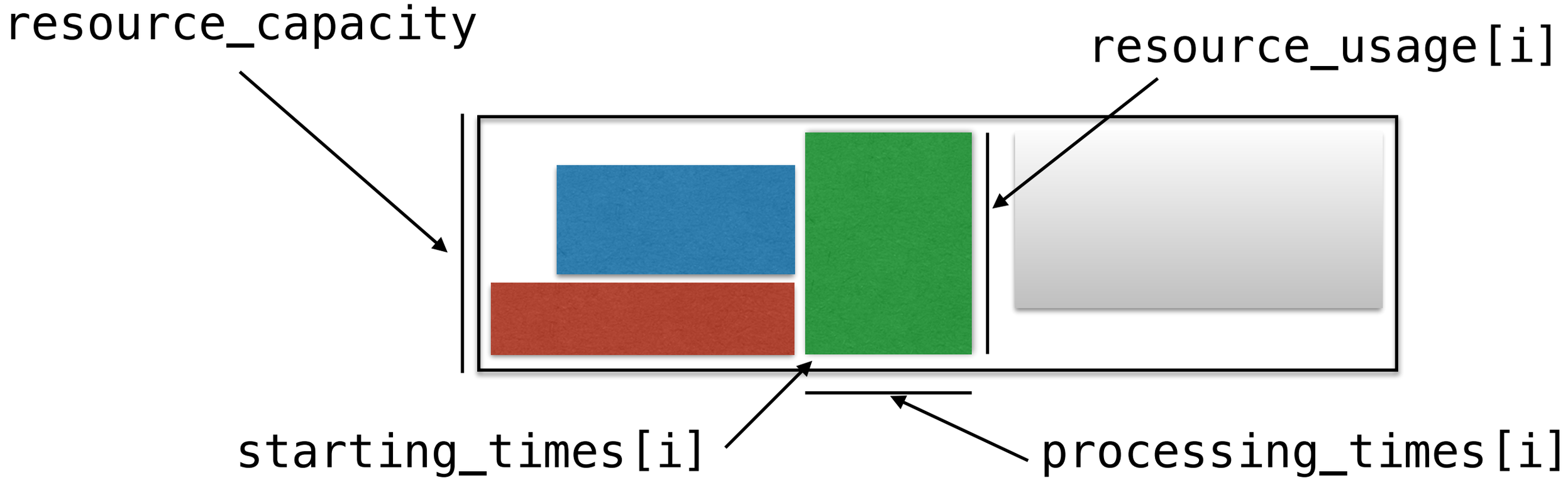
- Les compagnies n'ont pas de parti pris
- Utilisez la technologie qui semble la mieux adaptée
- Utilisez la technologie que vous maîtrisez le mieux

# Pourquoi la programmation par contraintes?

- Simplicité
- Performance

# Facilité de modélisation

```
constraint cumulative(starting_times,
                      processing_times,
                      resource_usage,
                      resource_capacity);
```



# Qu'en est-il des programmes à nombres entiers?

- **Modèle indexé par le temps**:
  nH variables binaires
  n + H contraintes

- **Modèle par événements**:
  $n^2$ variables binaires
  n + 1 variables continues
  $(n-1)(3+n^2/2) + n^2 + n$ contraintes

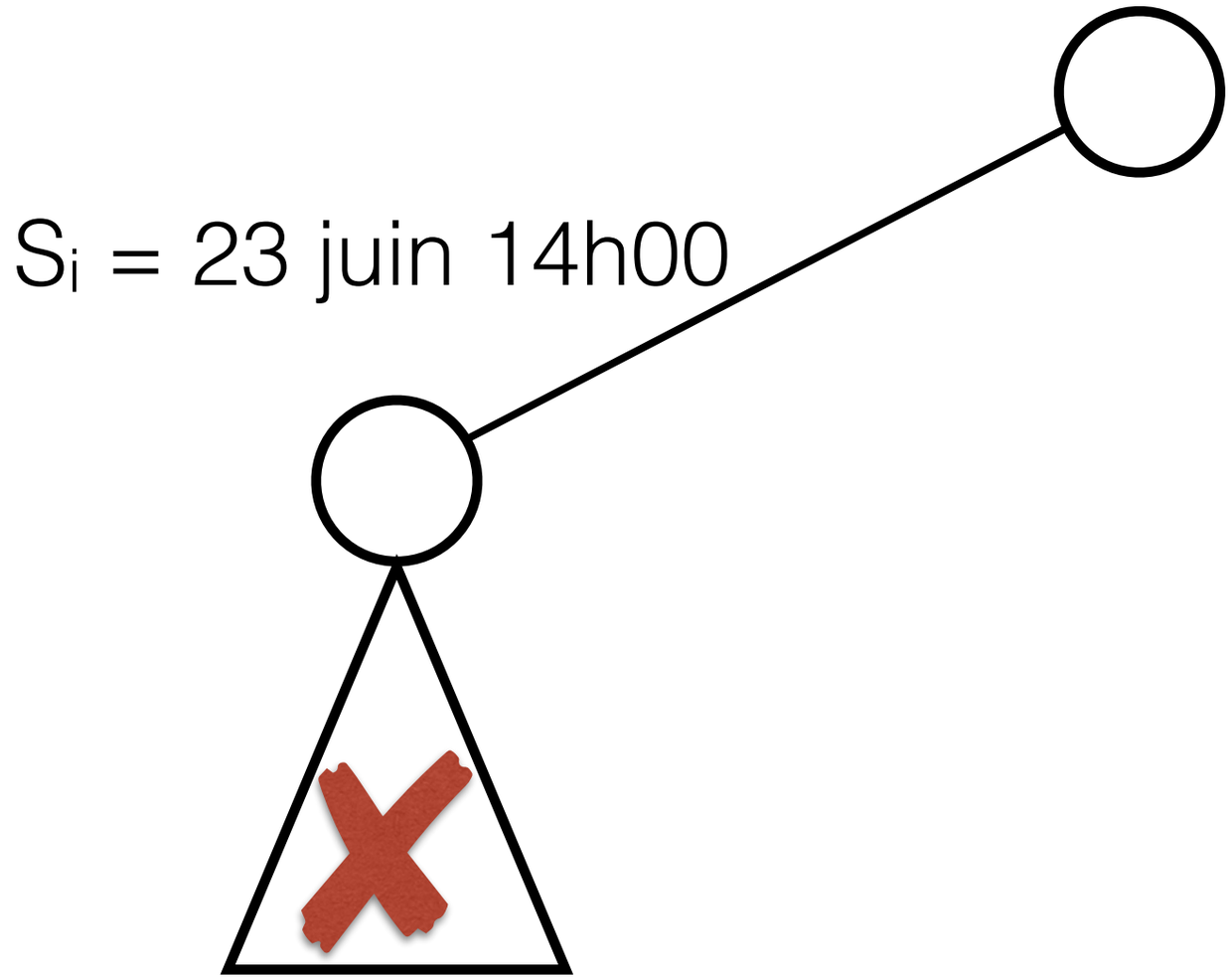
# Efficacité

- En programmation par contraintes, les principaux algorithmes de filtrage s'exécutent en O(n log n).

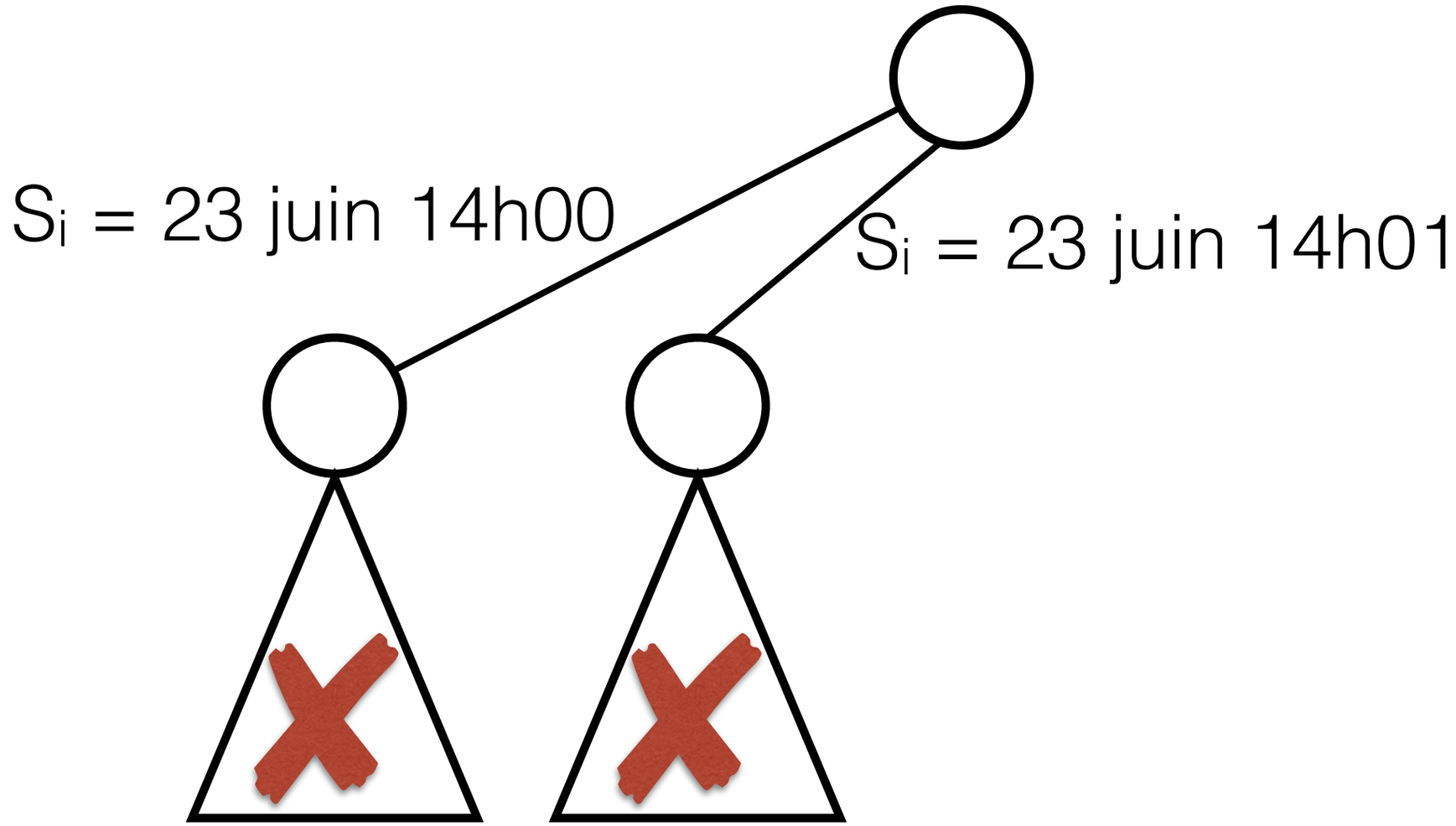
- La programmation en nombres entiers fait elle aussi du filtrage… à sa manière.

- Les opérations de pivot sur une matrice de $O(n^2)$ colonnes et $O(n^3)$ lignes sont très coûteuses.

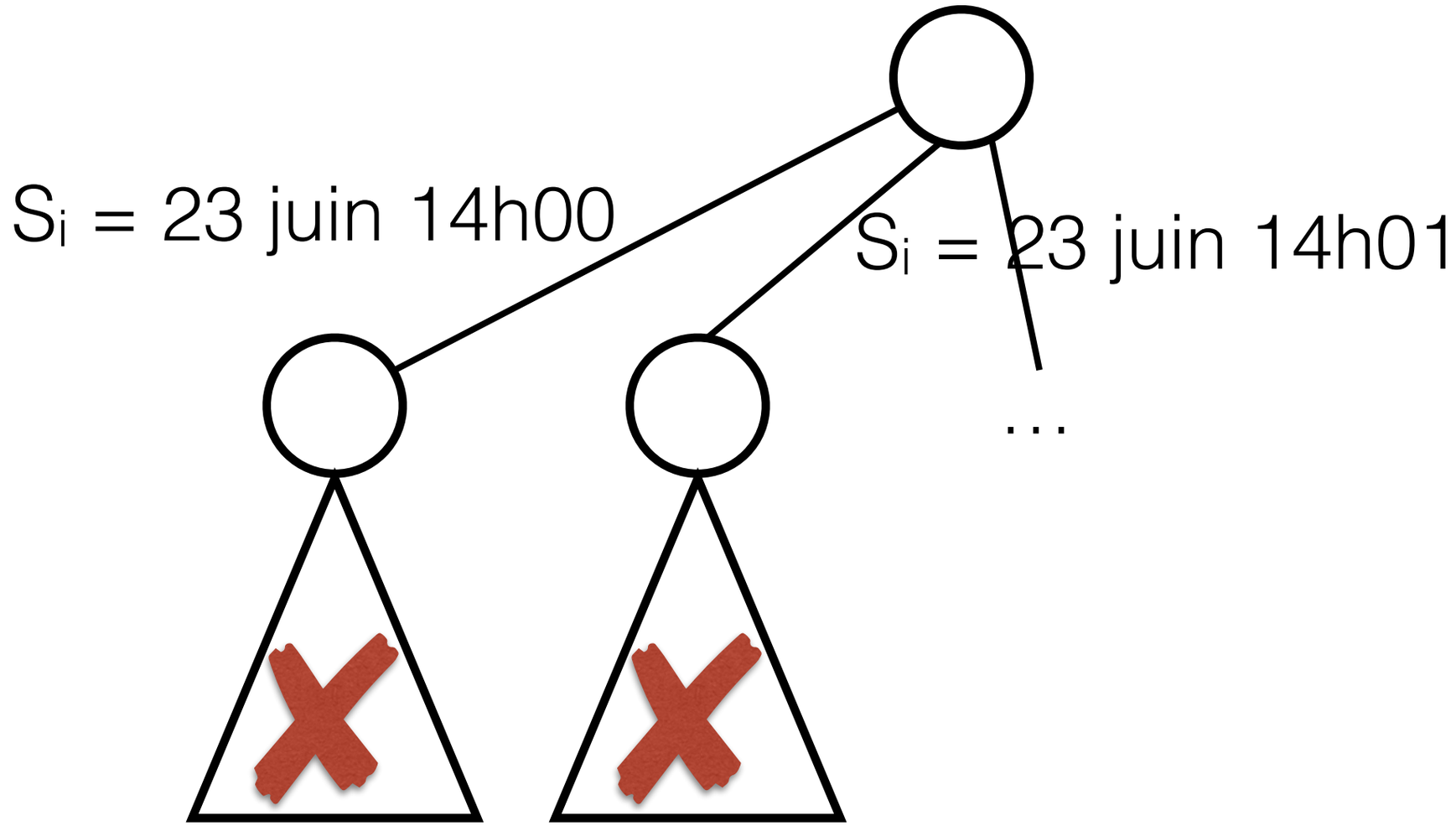
# Efficacité de la recherche



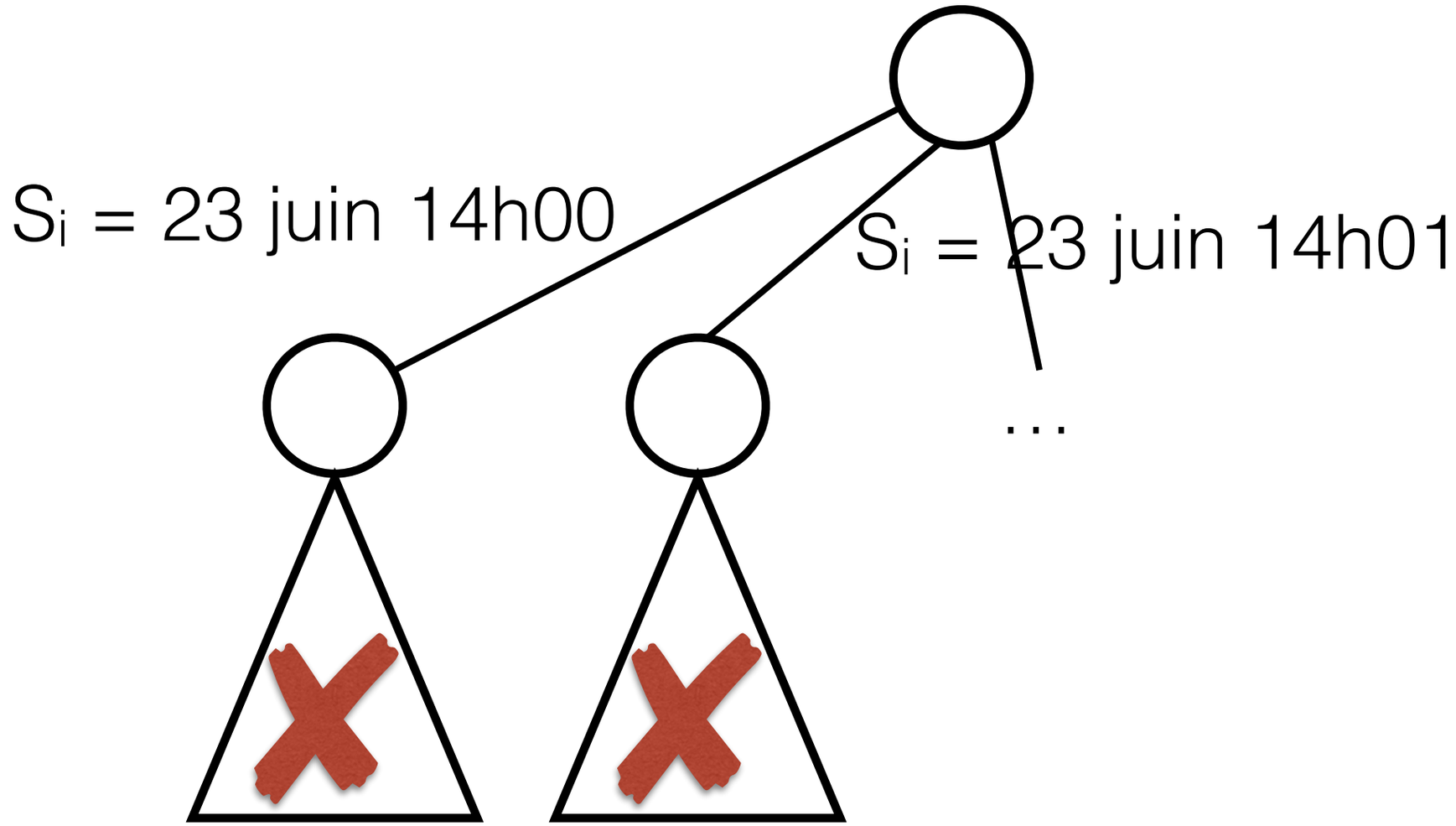
# Efficacité de la recherche



# Efficacité de la recherche

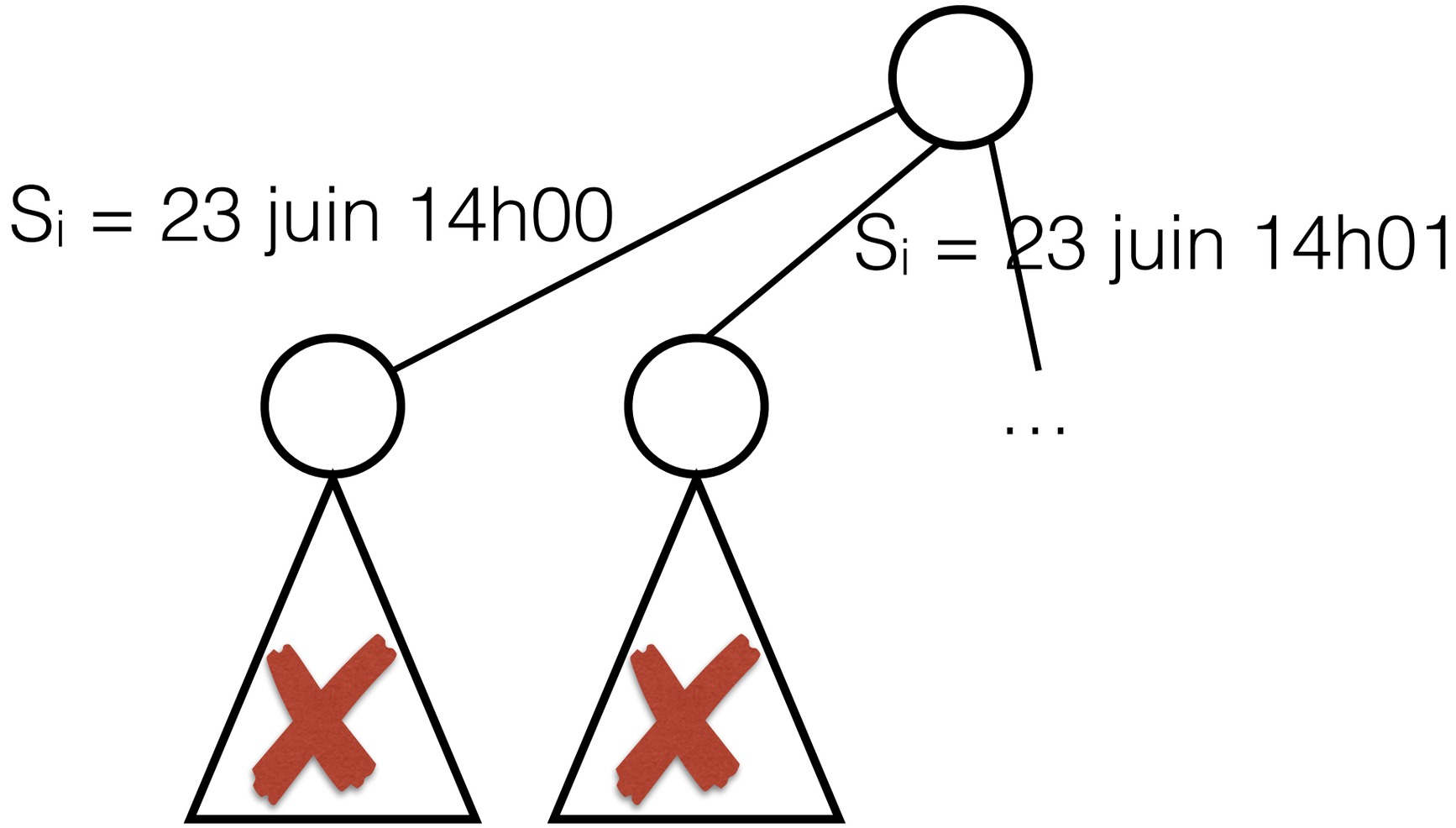


# Efficacité de la recherche




- Ne pas brancher sur la même tâche (setTimes)

# Efficacité de la recherche




- Ne pas brancher sur la même tâche (setTimes)
- Nogood: $S_i$ > 24 juin 23h59

# Nogood vs Coupe

- Si la tâche A ne commence pas avant le 23 juin, il faudra repousser l'exécution de la tâche B au 25 juin.

# Nogood vs Coupe

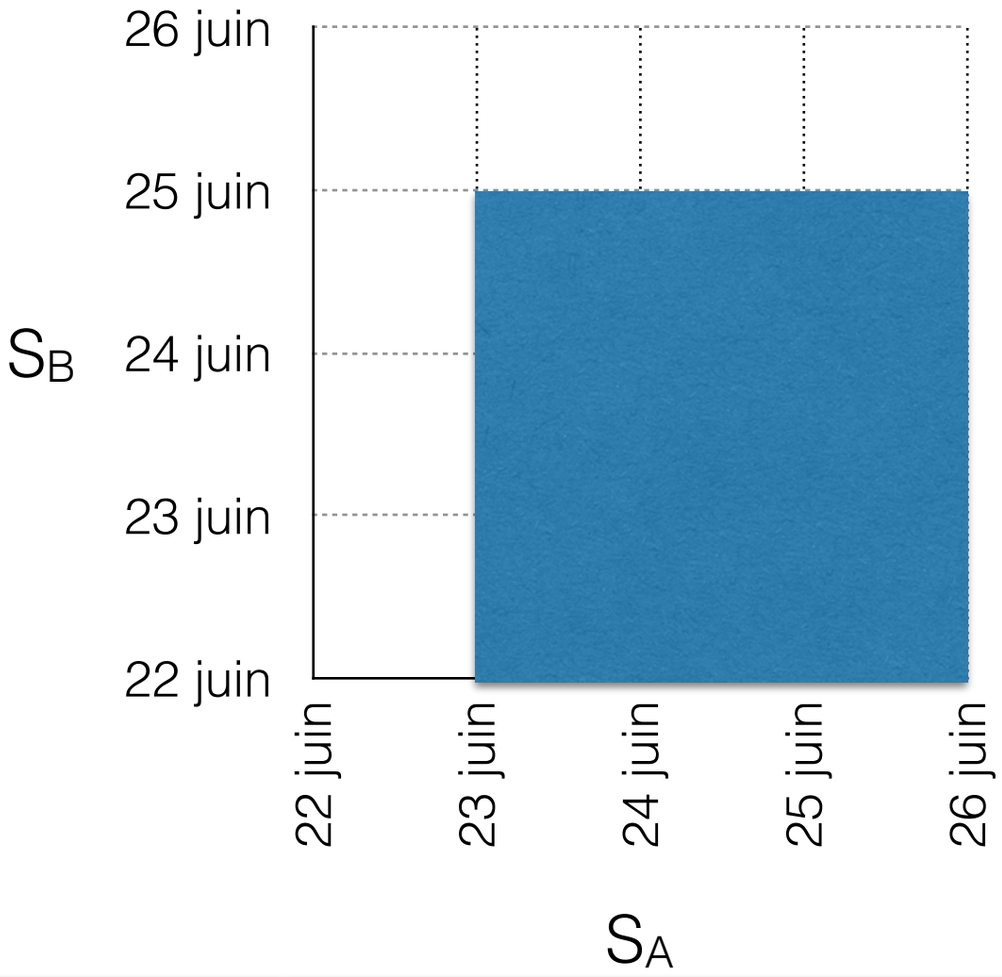
- Si la tâche A ne commence pas avant le 23 juin, il faudra repousser l'exécution de la tâche B au 25 juin.

$S_A \geq 23 \text{ juin} \Longrightarrow S_B \geq 25 \text{ juin}$

# Nogood vs Coupe

- Si la tâche A ne commence pas avant le 23 juin, il faudra repousser l'exécution de la tâche B au 25 juin.

$S_A \geq 23 \text{ juin} \Longrightarrow S_B \geq 25 \text{ juin}$

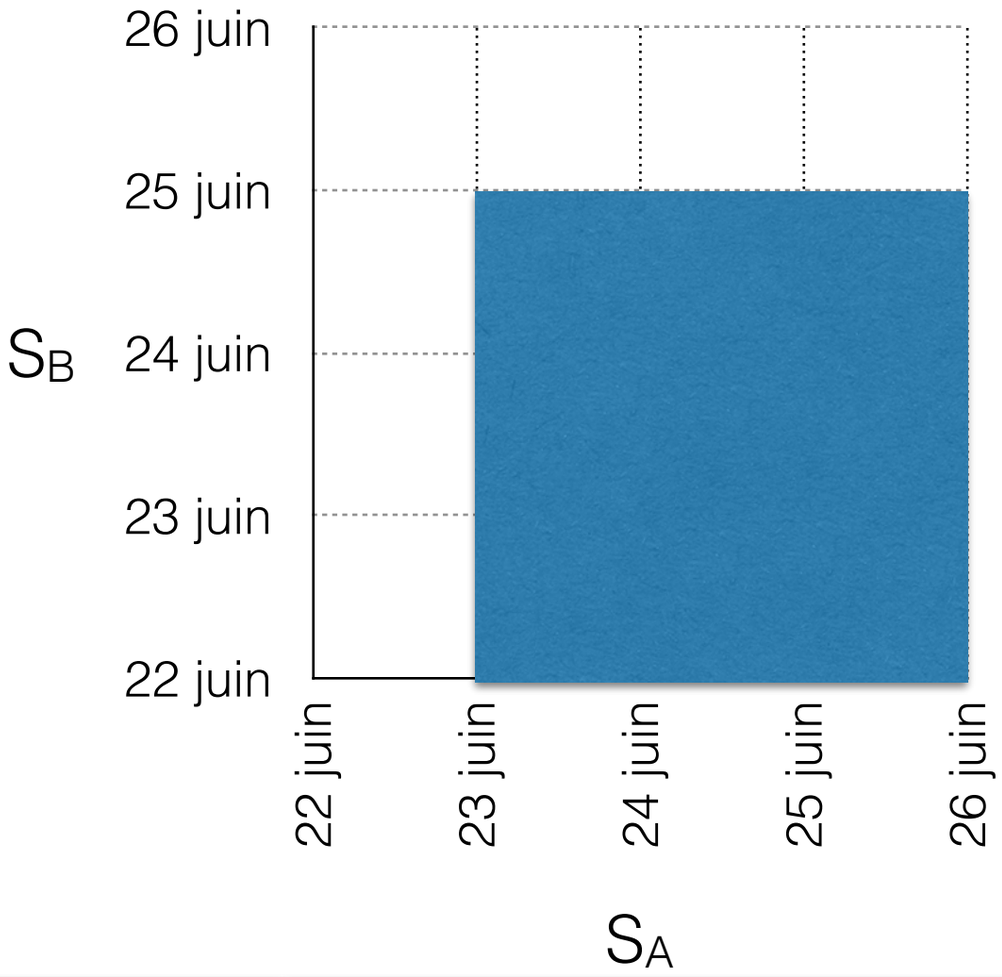


# Nogood vs Coupe

- Si la tâche A ne commence pas avant le 23 juin, il faudra repousser l'exécution de la tâche B au 25 juin.

$S_A \geq$ 23 juin $\Longrightarrow S_B \geq$ 25 juin

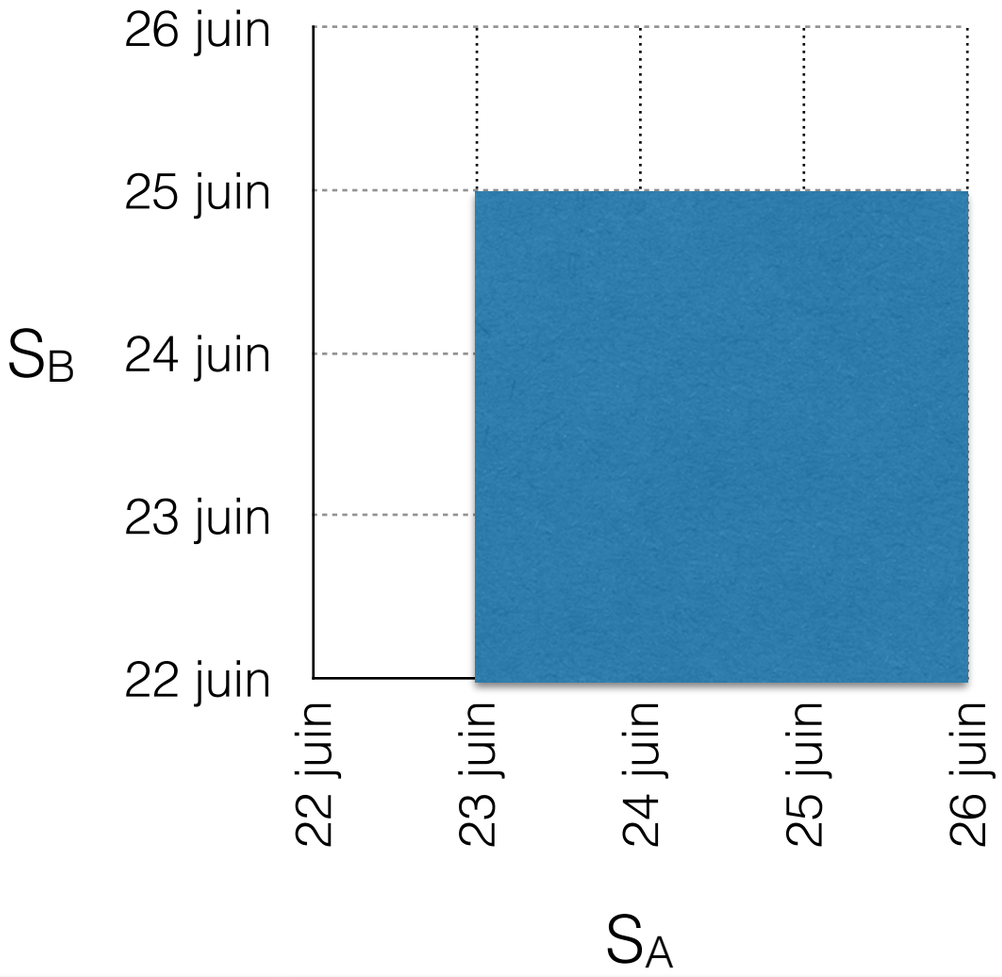
$S_A - S_B \leq 1$ jour



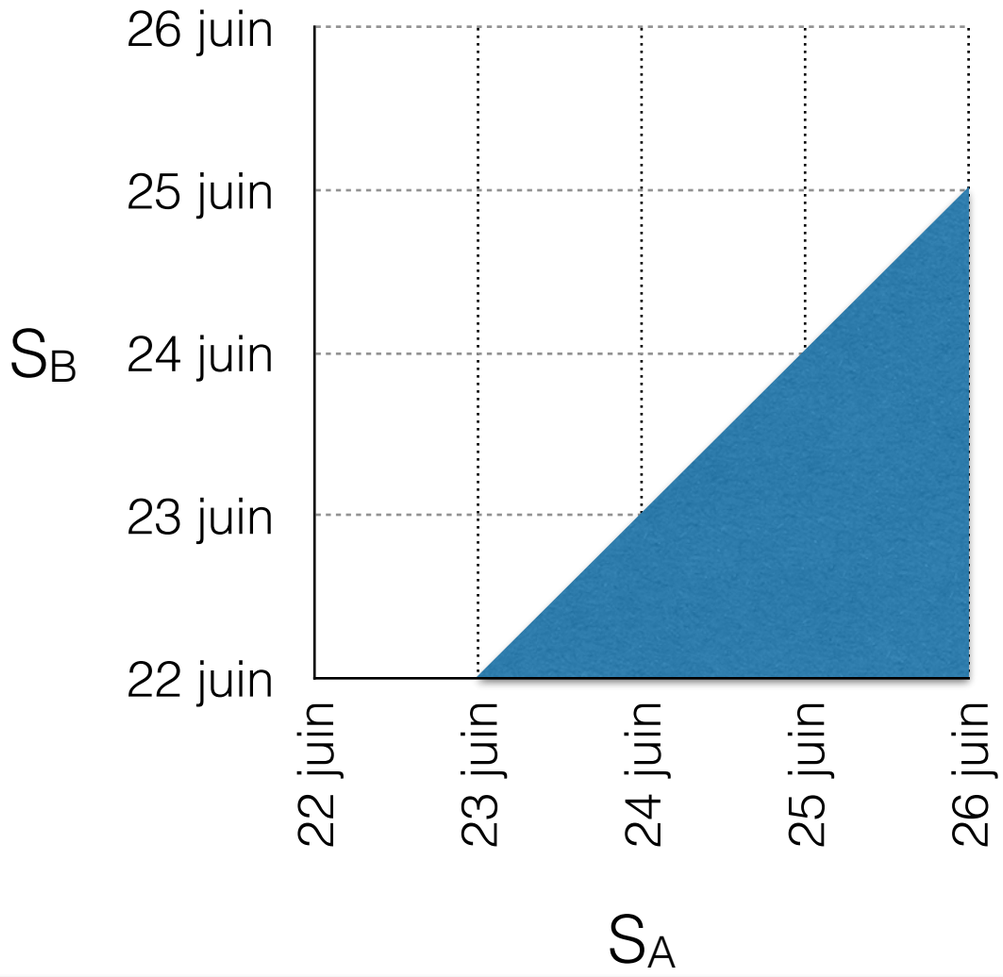
# Nogood vs Coupe

- Si la tâche A ne commence pas avant le 23 juin, il faudra repousser l'exécution de la tâche B au 25 juin.

$S_A \geq$ 23 juin $\Longrightarrow S_B \geq$ 25 juin




$S_A - S_B \leq$ 1 jour



# Synergie heuristique / filtrage

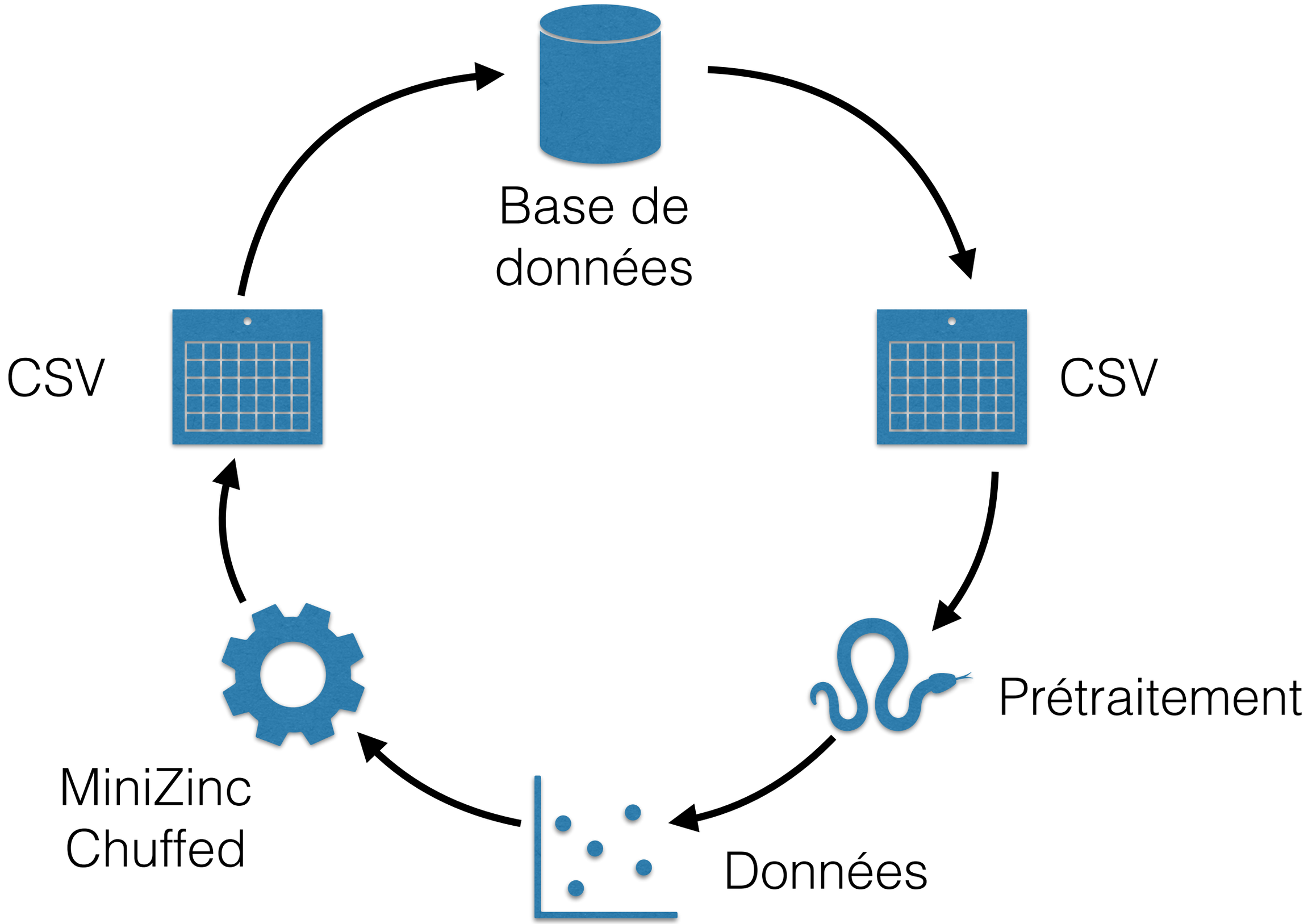
- L'heuristique *smallest* choisit la variable avec la plus petite valeur dans son domaine et branche sur cette paire variable / valeur.

- En ordonnancement, les algorithmes de filtrage ciblent justement ces valeurs.

# Choix technologique

# Technologies utilisées

- Langage de modélisation: MiniZinc
  - Le développement est beaucoup plus rapide
  - Permet de rectifier votre choix de solveur rapidement
    - Outils pour visualiser l’arbre de recherche
    - Outils pour détecter les contraintes contradictoires
    - Heuristiques de branchement sont limités
- Solveur: Chuffed


Base de
données
CSV
CSV
Prétraitement
Données
MiniZinc
Chuffed

# Comment passer à l'échelle?

**Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems**

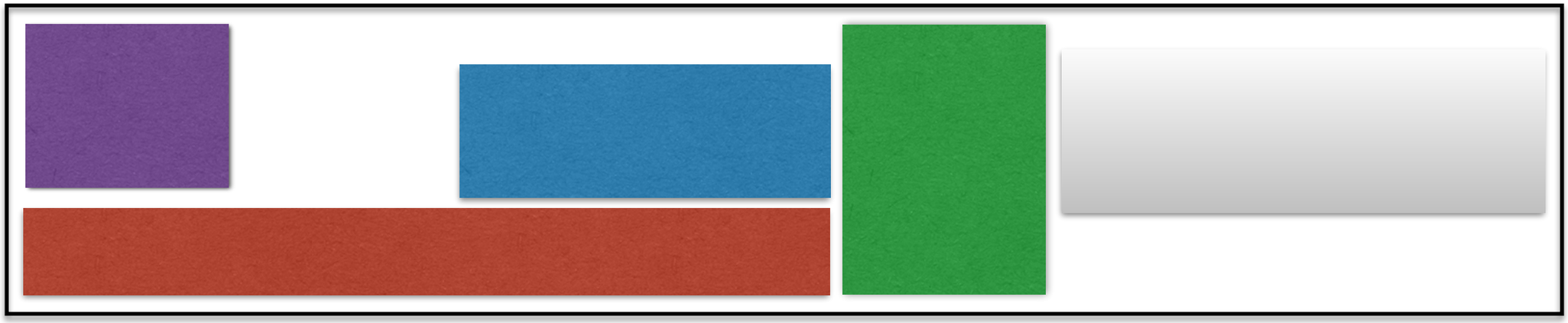
Paul Shaw*

ILOG S.A.
9, rue de Verdun, BP 85
94253 Gentilly Cedex, FRANCE.
shaw@ilog.fr

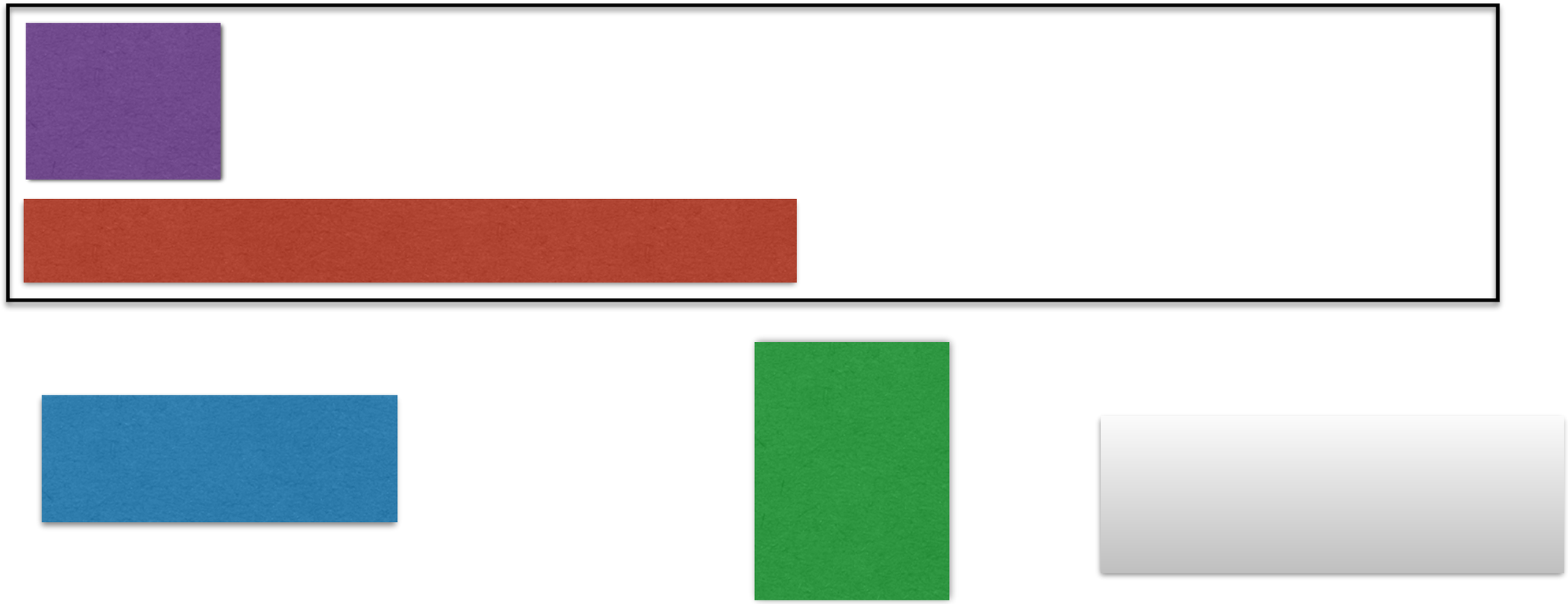
**Abstract.** We use a local search method we term Large Neighbourhood Search (LNS) to solve vehicle routing problems. LNS is analogous to the shuffling technique of job-shop scheduling, and so meshes well with constraint programming technology. LNS explores a large neighbourhood of

CP'98

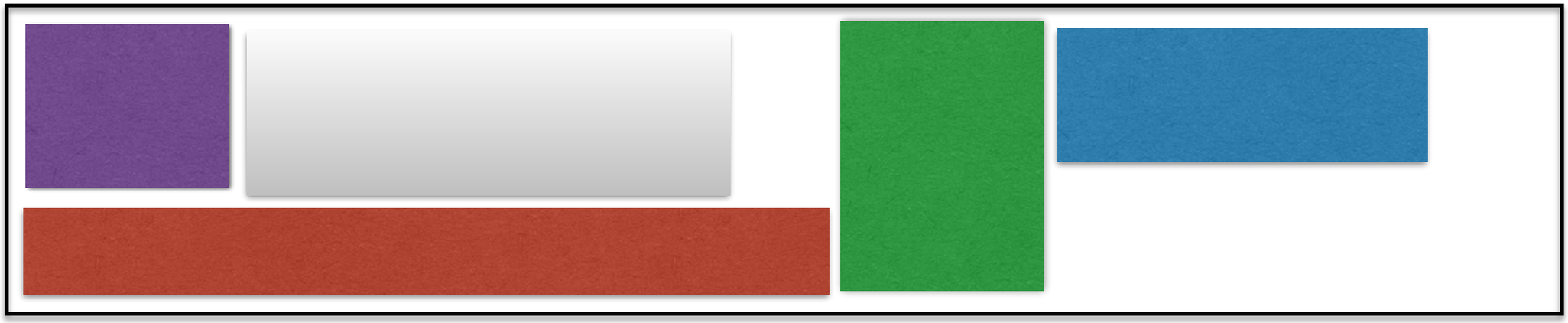
# Large Neigbourhood Search



# Large Neigbourhood Search



# Large Neigbourhood Search



# Comment programmer un LNS?

*Actes JFPC 2017*

## Une simple heuristique pour rapprocher DFS et LNS pour les COP

**Julien Vion** **Sylvain Piechowiak**

Université de Valenciennes et du Hainaut Cambrésis
LAMIH CNRS UMR 8201
{julien.vion, sylvain.piechowiak}@univ-valenciennes.fr

**Résumé**

Dans cet article, nous montrons comment une combinaison de stratégies de branchement et de redémarrages pour la recherche en profondeur d'abord (DFS) permet de reproduire le fonctionnement de la recherche par grand voisinage (LNS) pour la résolution de problèmes d'optimisation à contraintes, ce qui permet de rapprocher considérablement les deux techniques. En particulier, nous pouvons implémenter une stratégie DFS qui bénéficie des propriétés de passage à l'échelle de LNS tout en étant capable de prouver l'optimalité des solutions.

hybride [16] qui réalise des « déplacements » itératifs de manière similaire à une recherche locale, mais utilise une DFS et la propagation de contraintes pour améliorer la meilleure solution connue [17]. L'idée de LNS est de *relâcher* la meilleure solution connue en restaurant le domaine d'une partie de ses variables. Le « *fragment* » obtenu est alors *réoptimisé* par une DFS. Comme la plupart des stratégies incomplètes, LNS passe bien mieux à l'échelle qu'une DFS classique, mais elle ne peut pas prouver l'optimalité d'une solution (ou l'inconsistance d'un problème). De plus le choix des fragments à réoptimiser est une tâche difficile

JFPC 2017

## Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers

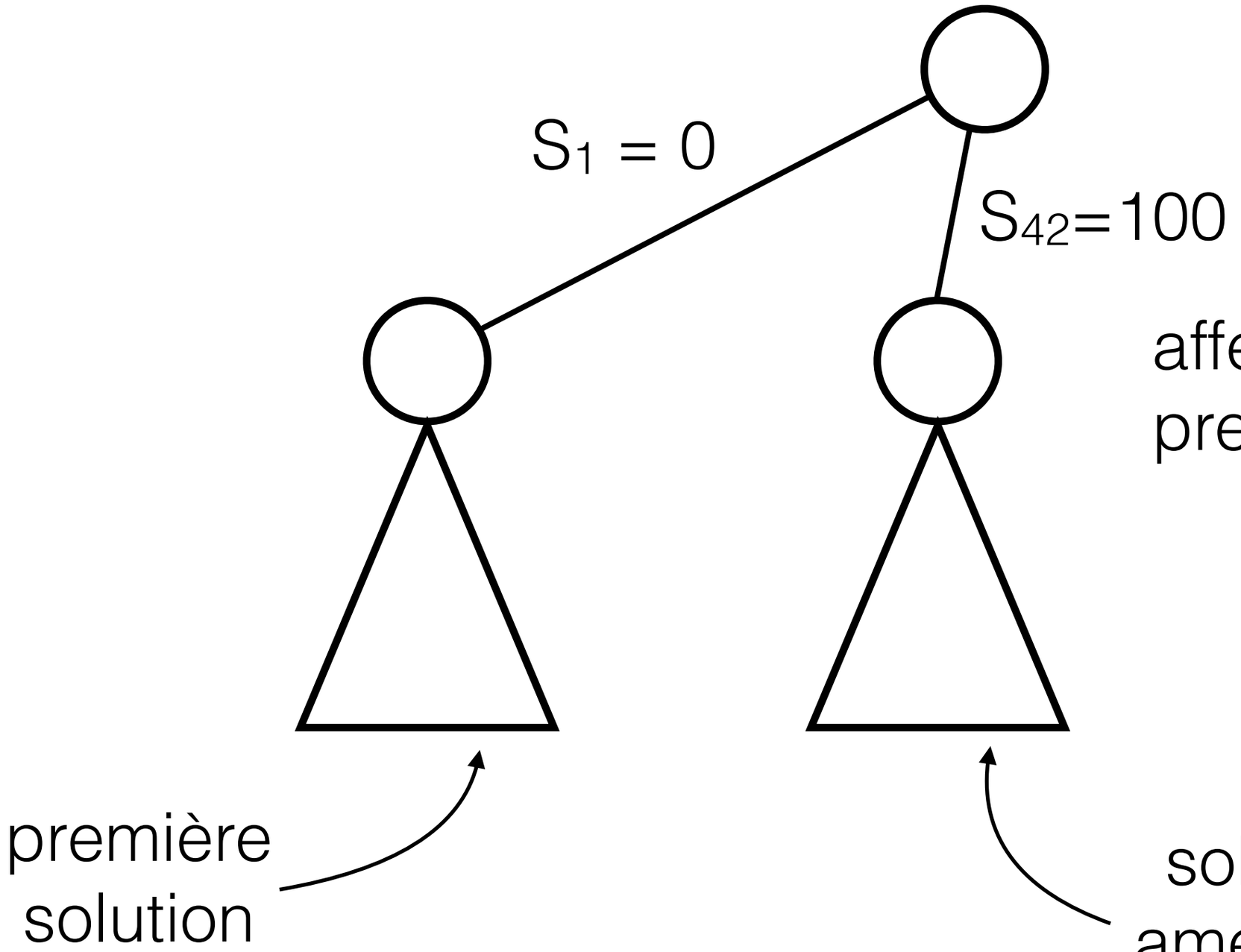
Emir Demirović(✉), Geoffrey Chu, and Peter J. Stuckey

School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
{emir.demirovic,pstuckey}@unimelb.edu.au

**Abstract.** Large neighbourhood search, a meta-heuristic, has proven to be successful on a wide range of optimisation problems. The algorithm repeatedly generates and searches through a neighbourhood around the current best solution. Thus, it finds increasingly better solutions by solv-

CP 2018

# Comment programmer un LNS?



# Performances

# Performances

Pour une instance de 100 tâches

# Performances

Pour une instance de 100 tâches

| | | |
|---|---|---|
| Programmation à nombres entiers | 2h - 3h | Solution optimale |

# Performances

Pour une instance de 100 tâches

| | | |
|---|---|---|
| Programmation à nombres entiers | 2h - 3h | Solution optimale |
| Heuristiques | 0s | Solution non réalisable |

# Performances

Pour une instance de 100 tâches

| | | |
|---|---|---|
| Programmation à nombres entiers | 2h - 3h | Solution optimale |
| Heuristiques | 0s | Solution non réalisable |
| Programmation par contraintes | 1m - 2m | Solution optimale |

# Performances

# Performances

Pour une instance de 800 tâches

# Performances

Pour une instance de 800 tâches

Programmation à nombres entiers

# Performances

Pour une instance de 800 tâches

| | | |
|---|---|---|
| Programmation à nombres entiers | | |
| Heuristiques | 0s | Solution non réalisable |

# Performances

Pour une instance de 800 tâches

| | | |
|---|---|---|
| Programmation à nombres entiers | | |
| Heuristiques | 0s | Solution non réalisable |
| Programmation par contraintes | 15m | Solution réalisable |

# Conclusion

- La programmation par contraintes est une technologie mature.
- Il faut toutefois comprendre son fonctionnement pour tirer avantage de ses forces.
- Les ressources cumulatives représentent un défi pour la programmation à nombres entiers et les méthodes heuristiques.
- Il est important de penser à l'interaction entre l'heuristique de recherche, le filtrage et la génération des no-goods.