Astrée and the static analysis of reactive control programs

Laure Gonnord David Monniaux

2016

Plan

Verimag

Contexte - Système embarqué

Ce qu'on a fait jusqu'à présent :

- Écrit les tâches (leur fonctionnalité)
- Ordonnancé en supposant leur période d'exécution connue.

Crédits

- Isabelle Puaut/ Damien Hardy pour Univ. Rennes
- Exemples et Dessins : Claire Maiza (Grenoble), Damien Hardy (Rennes), Reinhard Wilhelm and Jan Reineke (Univ. Saarland).
- Ordonnancement dans les systèmes temps-réel, coord. Maryline Chetto, chapitre WCET (C. Maiza, P. Raymond, C. Rochange)

Définitions : temps d'exécution pire cas

WCET = Worst Case Execution Time

- Attention à ne pas confondre pire-cas et **estimation** du pire cas.

Dessin D. Hardy (Rennes)

De quoi dépend le temps d'exécution ?

- Des chemins d'exécution dans le programme, qui eux-même dépendent des **entrées** (inconnues).
- De l'architecture matérielle : la durée d'une action dépend du matériel (et de son historique d'exécution).

Verimag

Validation des systèmes temps-réel durs

Que faut-il garantir ?

- Validation au niveau système : connaissance pire cas de l'impact du système (interruptions, etc.)
- Validation au niveau tâche : exec pire cas de chaque tâche individuelle. Le code est considéré en isolation.

Plan

Introduction

WCET d'une tâche

- Analyse de flot
- Analyses haut-niveau : calcul final du WCET
- Analyses bas-niveau : calcul de WCET par bloc

Bornes des coûts de préemption

Challenges de l'estimation du WCET

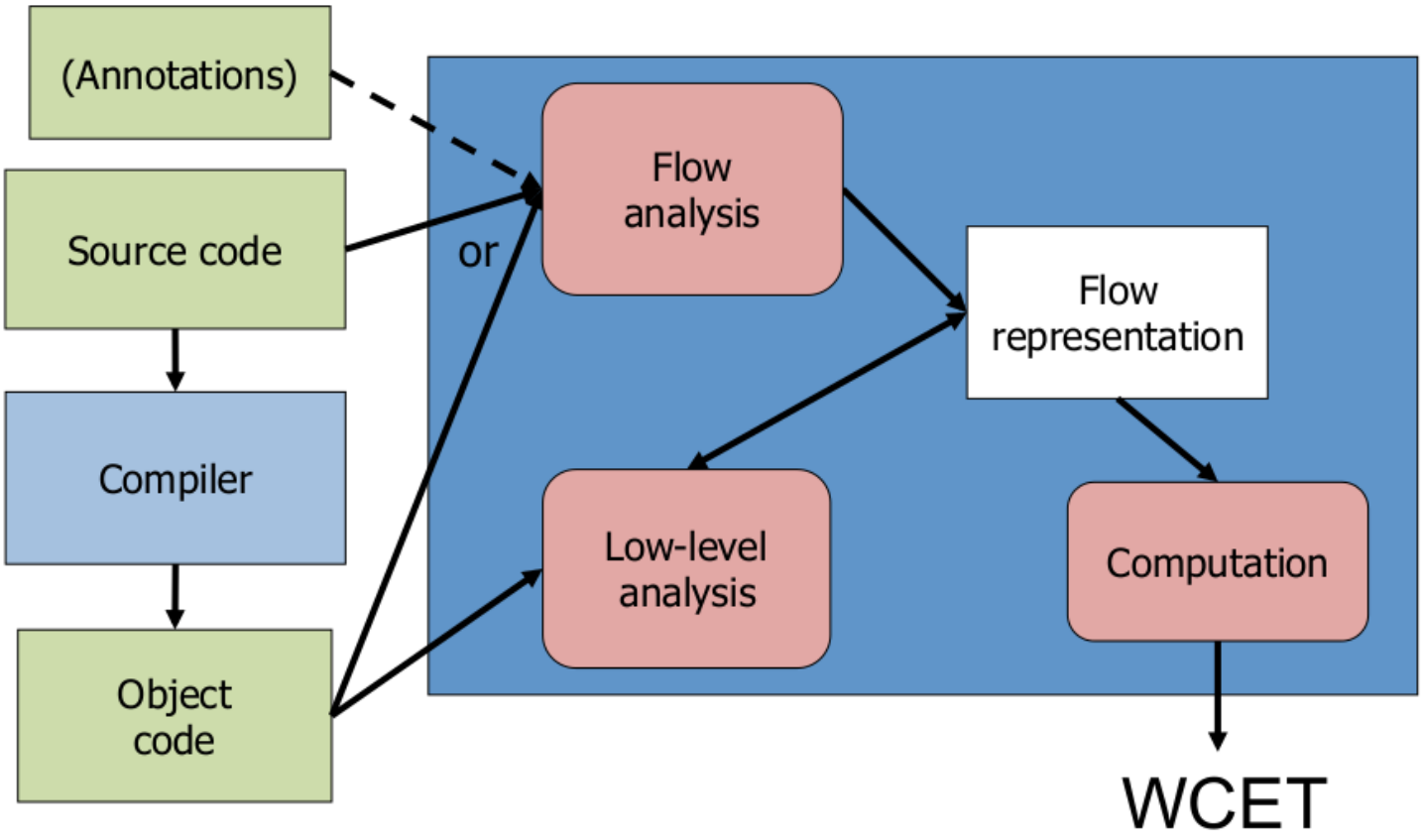
- **Sûreté** (safety) : l'estimation est plus grande que le "vrai" WCET ▶**essentiel** pour les programmes critiques De plus une erreur signifie qu'on se trompe dans l'analyse d'ordonnançabilité après.
- **Précision** : en cas de trop grande surapproximation ▶test d'ordonnançabilité plante ou, trop de ressources utilisées.

Principes des analyses statiques de WCET

- Principe : Analyse basée sur la structure du programme (sans exécution).
- 3 composants principaux :
  1. Récupération du flot du programme
  2. Analyse bas niveau ▶récup d'un WCET par block
     Analyse hardware-dépendante
  3. Calcul final en considérant **tous les chemins**

Verimag

Big picture, récap



Dessin D. Hardy (Rennes) Il manque une flèche entre lowlevel et computation.

Plan

Verimag

Analyses de base/ Autres analyses

Rappel :toutes les analyses sont statiques :

- Extraction d'un graphe de flot (à partir du code source ou du binaire). **assez simple**.
- Calcul du nombre d'itération des boucles. **pas simple**
- (Optionnel) Chemins infaisables. **ça dépend**

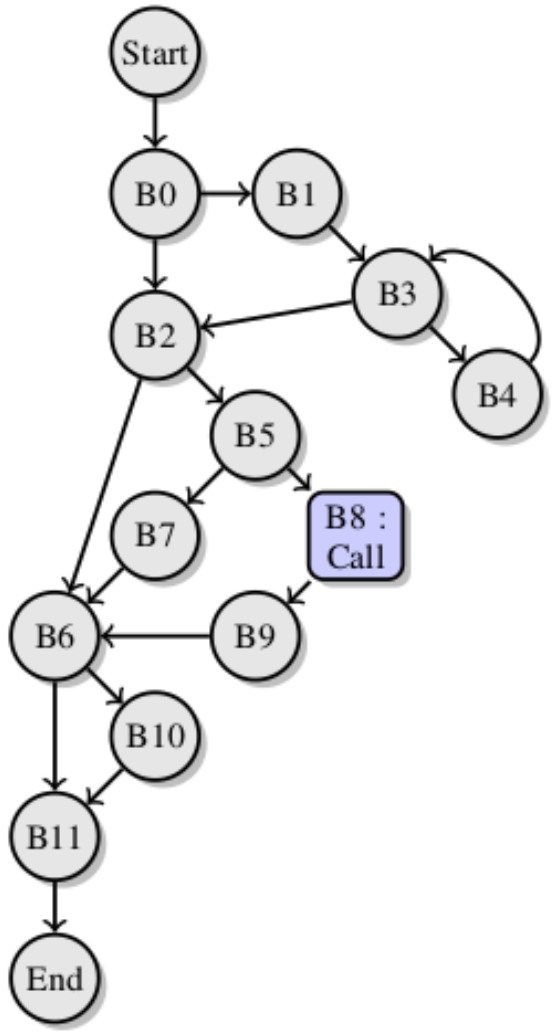
Un programme exemple (step)

Crédit exemple + dessins C. Maiza (Grenoble)

```
void Step () {
  if (init){
    for (i = 0 ; i < 10 ; i ++){
      tab [i]=0;
    }
  }
  if (not (idle)){ // idle et low = etats
    if (low) {     // le systeme n'a plus de batterie
      S=X+Y ;      // X et Y = capteurs
    }
    else {
      insert (X, tab) ;
      S=tab[0]+Y ;
    }
  }
  if (init){
    init=false ;
  }
}
```

Graphe de flot - (programme step)

```
void Step () {
  if (init){
    for (i = 0 ; i < 10 ; i ++){
      tab [i]=0;
    }
  }
  if (not (idle)){
    if (low) {
      S=X+Y ;
    }
    else {
      insert (X, tab) ;
      S=tab[0]+Y ;
    }
  }
  if (init){
    init=false ;
  }
}
```



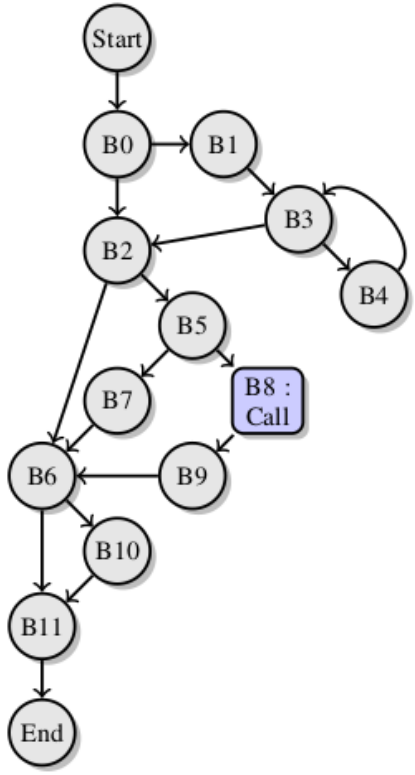
Calcul de borne d'un nid de boucle

```
void Step () {
  if (init){
    for (i = 0 ; i<10 ; i++){
      tab[i]=0;
    }
  }
}
```

Calculer le nombre exact est indécidable. ▶Approximations.
Méthodes plus ou moins syntaxiques.
Voir [Alias10] pour un exemple de telle analyse.

Calcul de chemins infaisables (infeasible paths)

```
void Step () {
  if (init){
    ...
    }
  } ...
  if (low) {  ...}
  }
  if (init){
    ...
  }
}
```



- `init` inchangé entre les deux tests $\Rightarrow$ transitions mutuellement exclusives (B0-B1 et B6-B11)
- Si on sait `init => low`, alors on peut enlever un long chemin. (les outils utilisent des annotations)

▶Voir [Henry14] pour un exemple de tel analyse. On gagne beaucoup.

What's next ?

1. Calcul du WCET par bloc (plus tard).
2. On a le flot (ou l'AST) et des WCET par bloc ▶calculer le WCET total.

On suppose des architectures simples, où la durée d'une instruction dépend seulement de l'instruction et des opérandes. Pas d'*overlap* entre les instructions.

Plan

Verimag

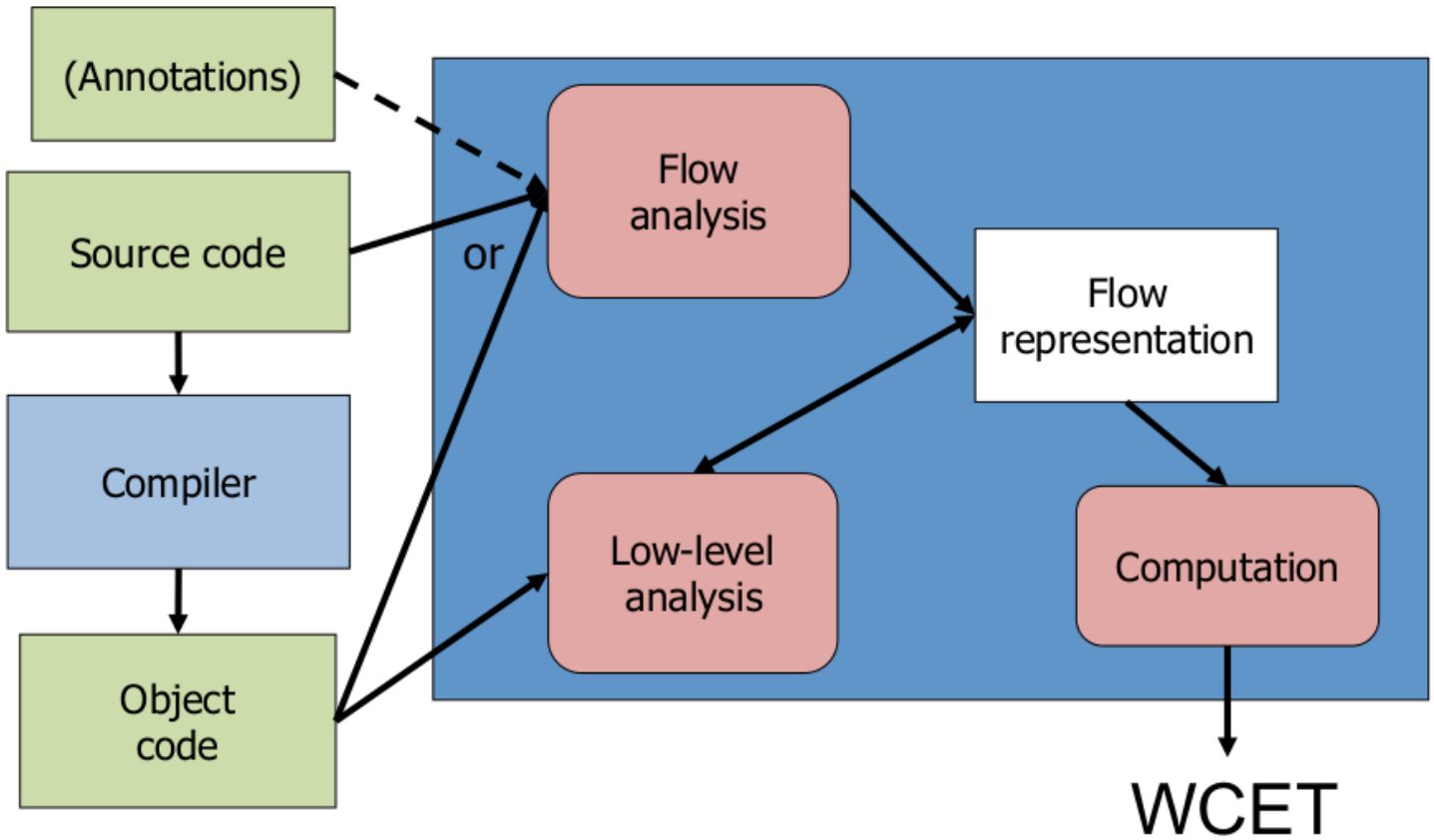
Rappel du problème

Le problème

On a le flot (ou l'AST) et des WCET par bloc. On veut calculer le WCET total.

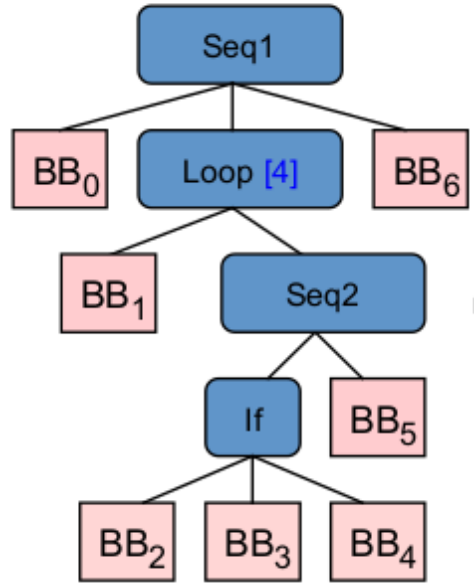
- 2 méthodes

Sur la récap.


(Annotations)
Source code
Compiler
Object code
or
Flow analysis
Low-level analysis
Flow representation
Computation
WCET

Méthode 1 : sur l'AST

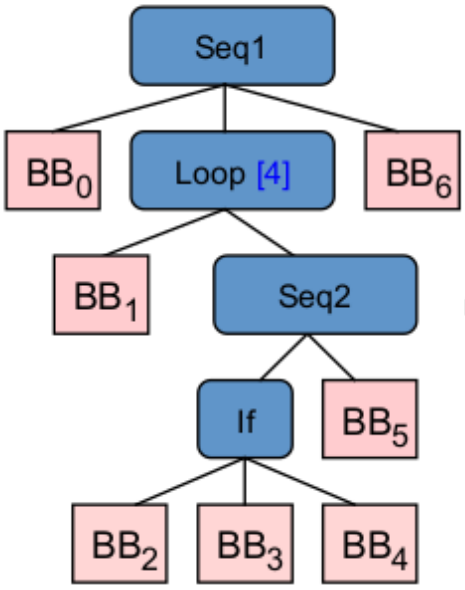
```
int x, p=0; i=0;
for (x=0; x<4;x++){
  if (i mod 2) p++
  else i++;
}
```



On dispose des WCET des blocs, puis :

- WCET d'une séquence : somme des WCET
- WCET (if) : WCET(test) + max des 2 WCET des branches
- WCET(while) : nbboucles * (WCET(test) + WCET (body) + WCET (inc))

Méthode 1 : exemple



Timing schema

WCET(If) = WCET_BB2 + **max**(WCET_BB3 , WCET_BB4)
WCET(Seq2) = **WCET(If)** + WCET_BB5
WCET(Loop) = 4 * (WCET_BB1 + **WCET(Seq2)**) + WCET_BB1
WCET(Seq1) = WCET_BB0 + **WCET(Loop)** +WCET_BB_6

Méthode 1 : avantages/inconvénients

- Avantages : pas cher ! passe bien à l'échelle, retour au source facile.
- Inconvénients : pas compatible avec des optimisations "agressives" du compilateur.

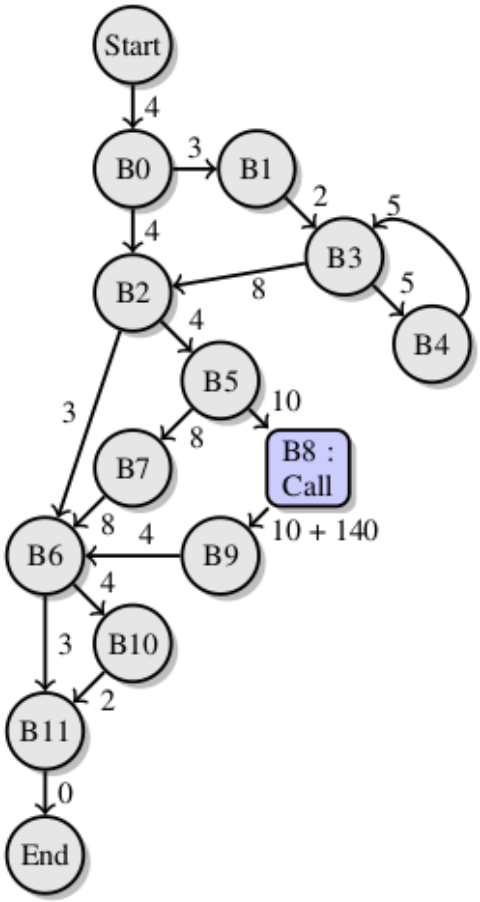
Méthode 2 : sur le CFG - IPET

IPET = Implicit path enumeration technique
On écrit une instance d'un **problème linéaire en nombre entiers** (ILP) :

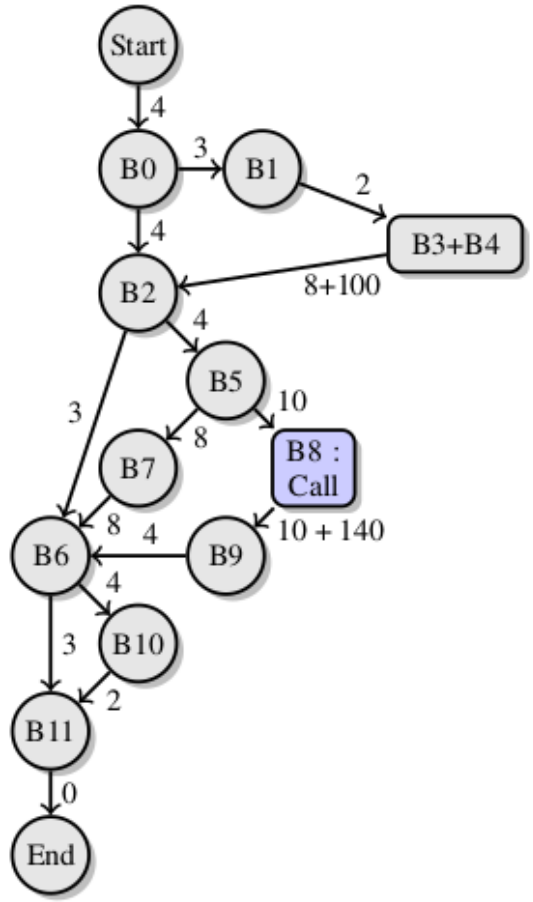
- Données : T_i les WCET des blocs.
- Une variable numérique $x_{i,j}$ par transition qui compte le nombre de fois où l'on passe par celle-ci.
- La fonction objectif est $Max \sum f_i T_i$.
- Contraintes : pour tout bloc, somme des entrants = somme des sortants.
- Cas des boucles : rajouter le nb max dans les contraintes
- Les infos de flot rajoutent des contraintes.

Méthode 2 : exemple

Exemple C. Maiza, livre “Ordonnancement pour les systèmes temps-réel”



(a) CFG + poids



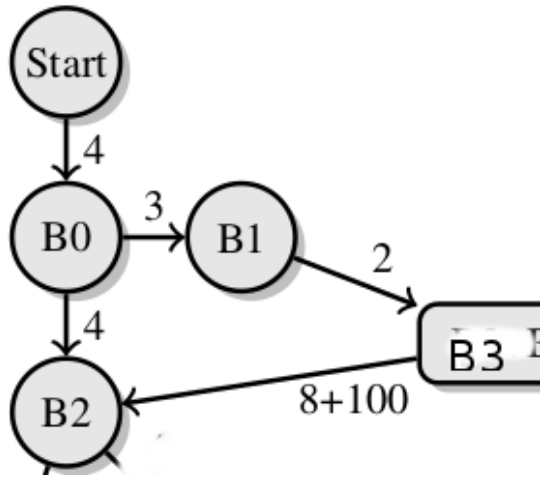
(b) CFG + poids + résumés

- Exo : écrire le (début) du Programme Linéaire.

Méthode 2 : problème LP simplifié et lpsolve

(LP solve format)

```
max: 4 xs0 + 3 x01 + 4 x02
+ 0 * x2e +  2 x13 + 108 x32;
xs0 = 1 ;
x2e = 1 ;
x0 = x01 + x02 ;
x01 = x13 ;
x13 = x32 ;
x2e = x02 + x32 ;
```



- Utilisation de `lpsolve` (si il n'y a pas de boucle la solution est en 0/1) :

```
$ lp_solve mif25.lp

Value of objective function: 117

Actual values of the variables:
xs0                             1
x01                             1
x02                             0
...
```

Méthode 2 : avantages/inconvénients

- Avantages : supporte tous les types de graphes de flots, super optimisés ou complétement instructurés ...
- Inconvénients : coûte plus cher, moins de feed-back, les annotations sont difficiles à propager (dans le cas binaire).

- Utilisé dans la plupart des outils commerciaux/académiques
- Does not take into account the semantics of the program!

Verimag

With program semantics

```
if (clock % 24 == 3) {
... /* A */
}
...
if (clock % 3 == 1) {
... /* B */
}
clock++;
```

A and B cannot be executed in the same clock cycle.
But the above methods will consider time(A)+time(B)!

Optimization modulo theory

1. Encode possible execution traces (or superset) as SMT problem
2. Add timing for basic blocks
3. Maximize timing over execution traces

Produces exponential behavior in SMT-solvers if no precautions taken when encoding timing.

Henry et al., *How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics*, LCTES 2014

Verimag

Plan

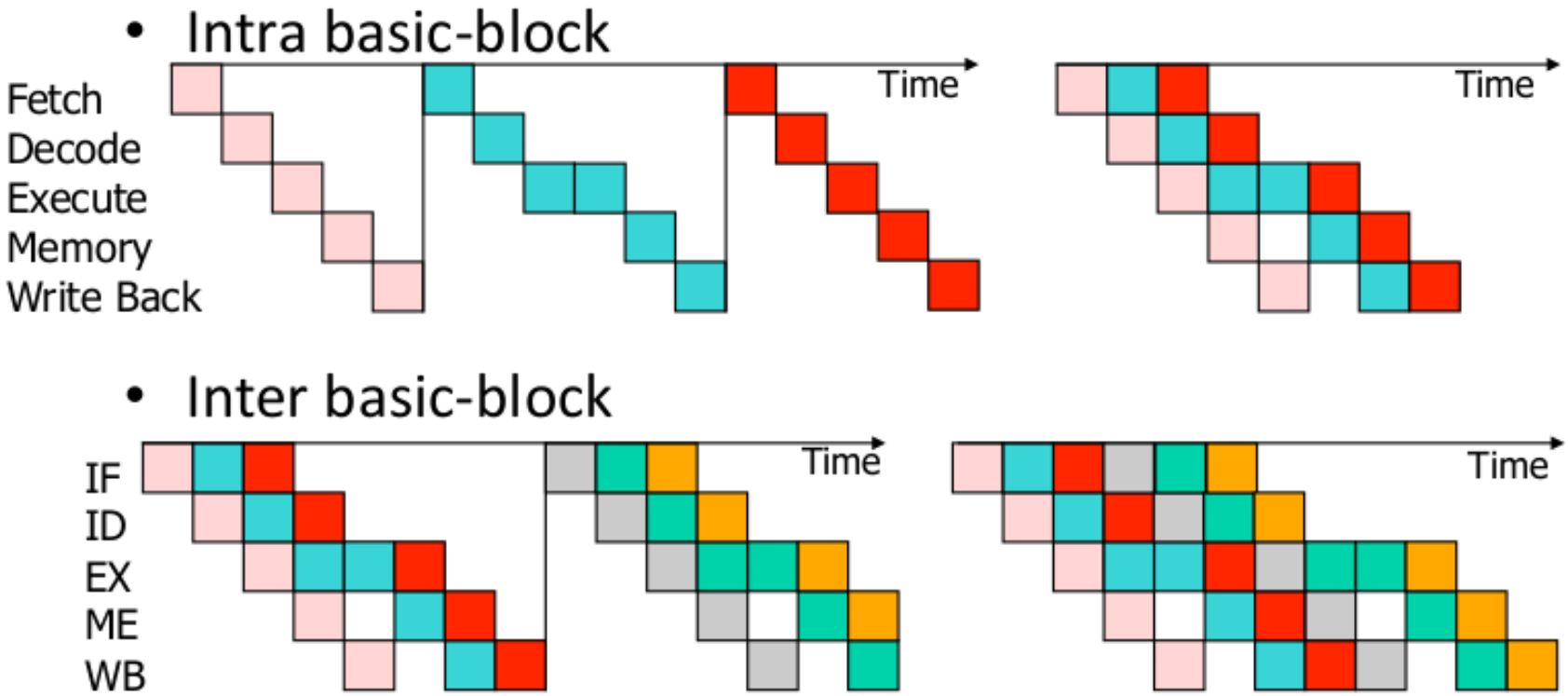
Verimag

Problématiques

- Effets locaux : pipeline (au niveau instruction)
- Effets globaux : cache, prédicteurs de branchement (il faut une connaissance entière du code).

Verimag

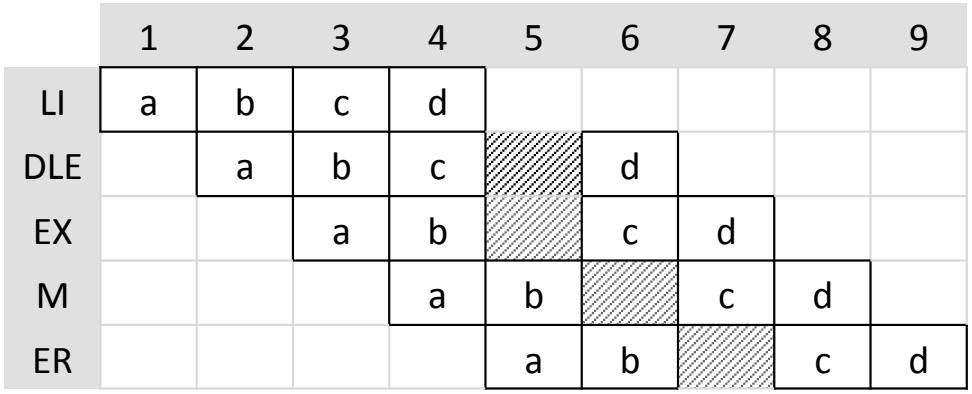
Pipeline 1/3



crédit image D. Hardy (Rennes)

Pipeline 2/3 : intra-basic blocks

Modification du calcul de WCET d'un bloc




```
a: add r0,r15,#128
b: ldrb r1,[r]
c: cmp r1,#0
d: bne _low0
```

▶On prend en compte le pipeline dans le WCET d'un bloc de base.

crédit image C. Maiza (Grenoble)

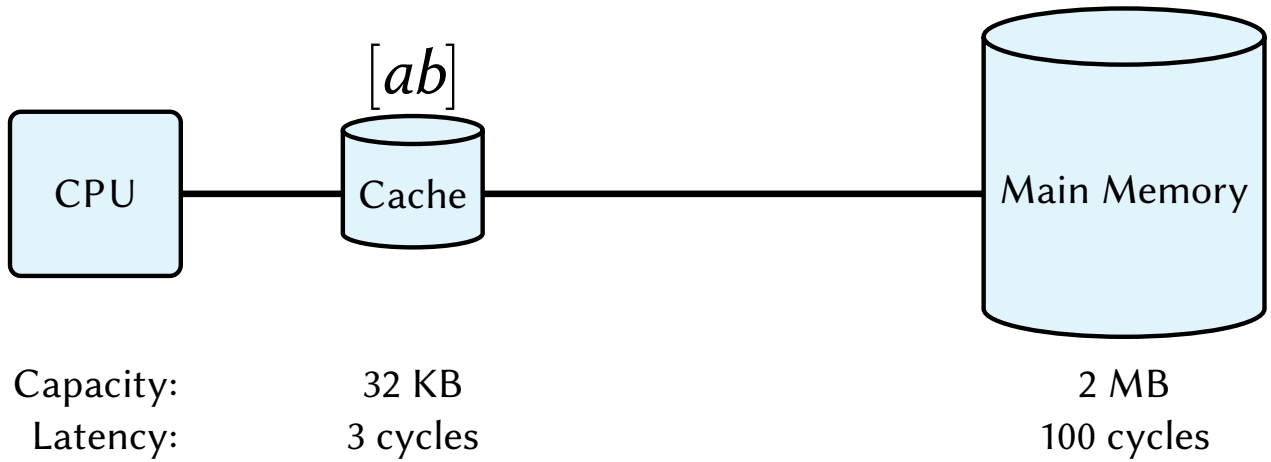
Pipeline 3/3 : inter-basic blocks

Modification du calcul global du WCET :

- (méthode AST) on remplace l'addition des WCET par une addition plus "intelligente"
- (méthode CFG) on ajoute des contraintes négatives sur certaines transitions dans le problème LP.

Caches, rappels d'archi

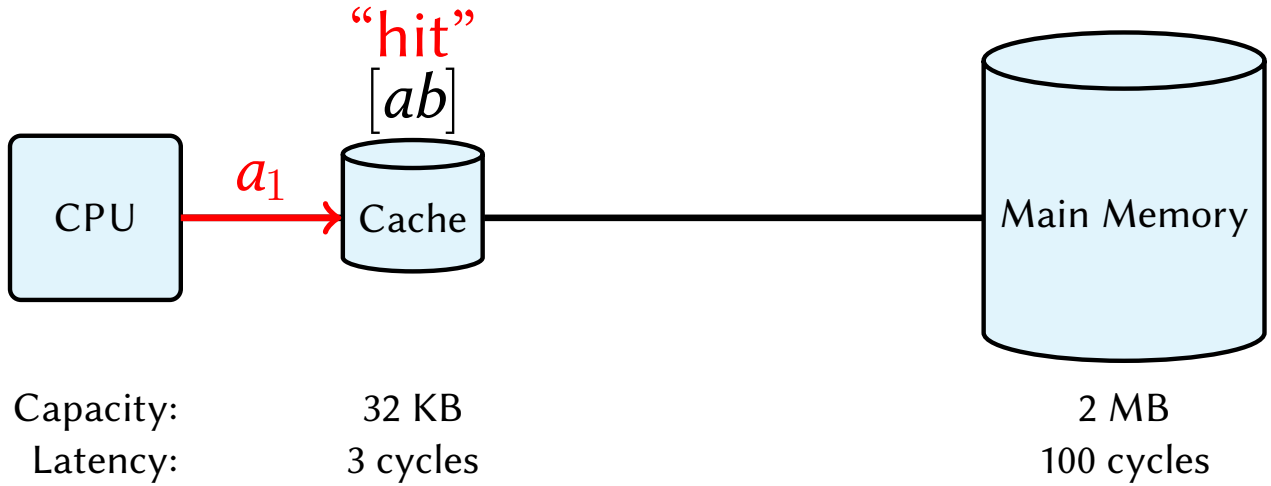
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

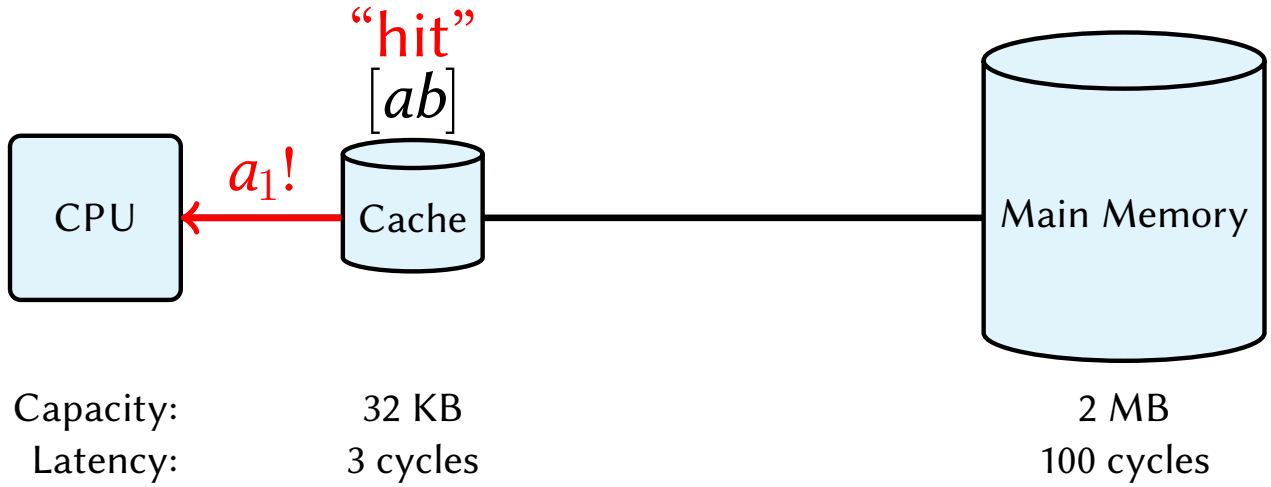
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

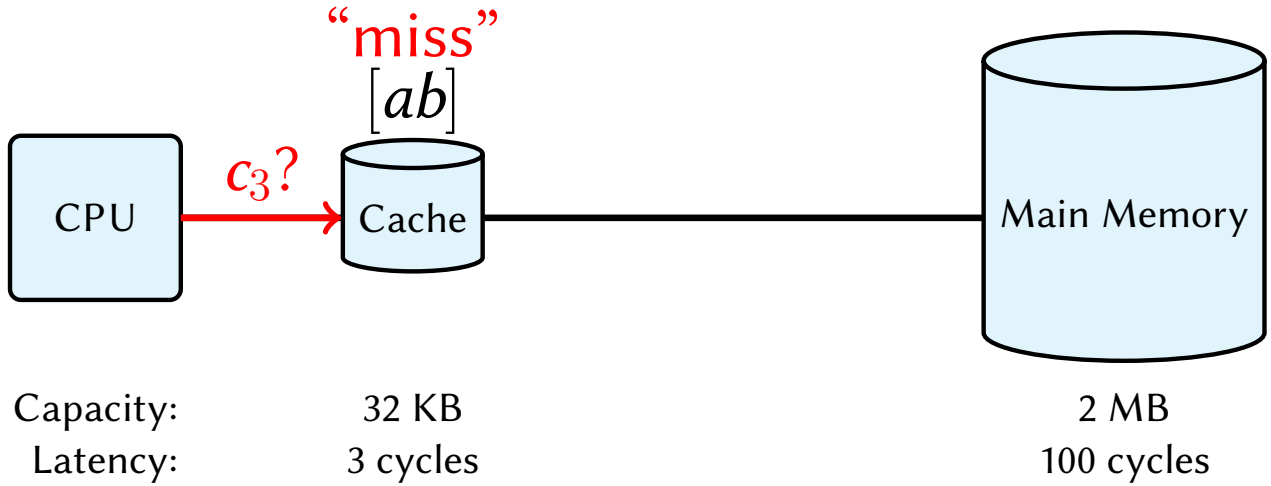
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

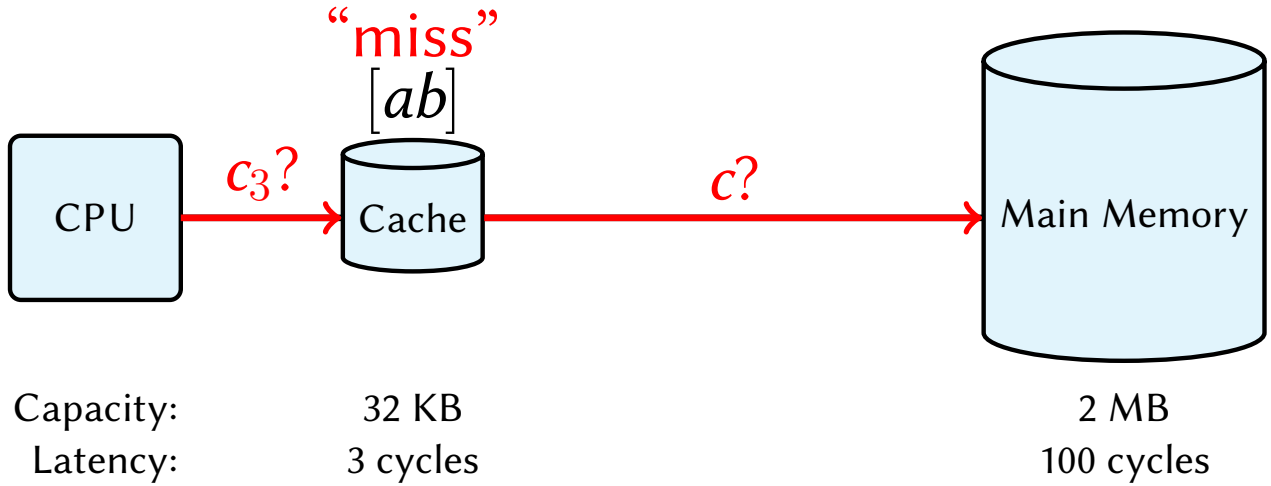
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

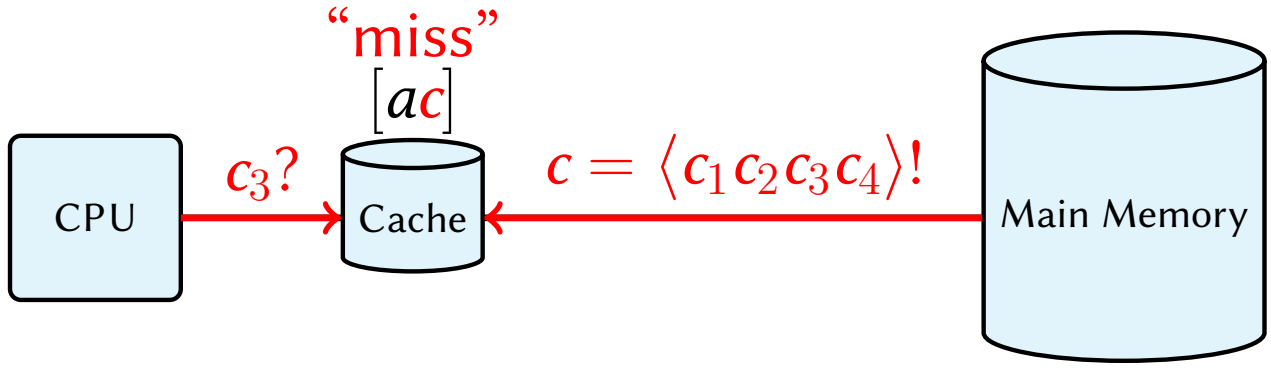
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

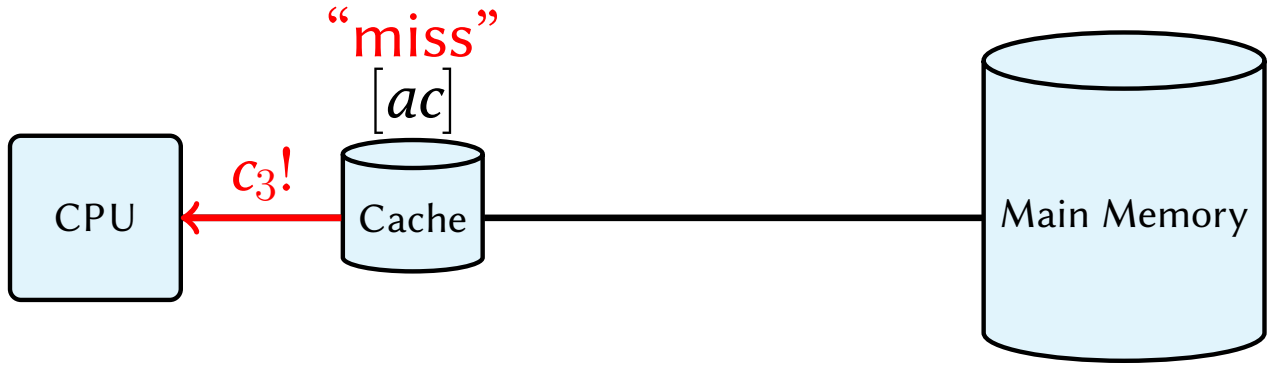
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

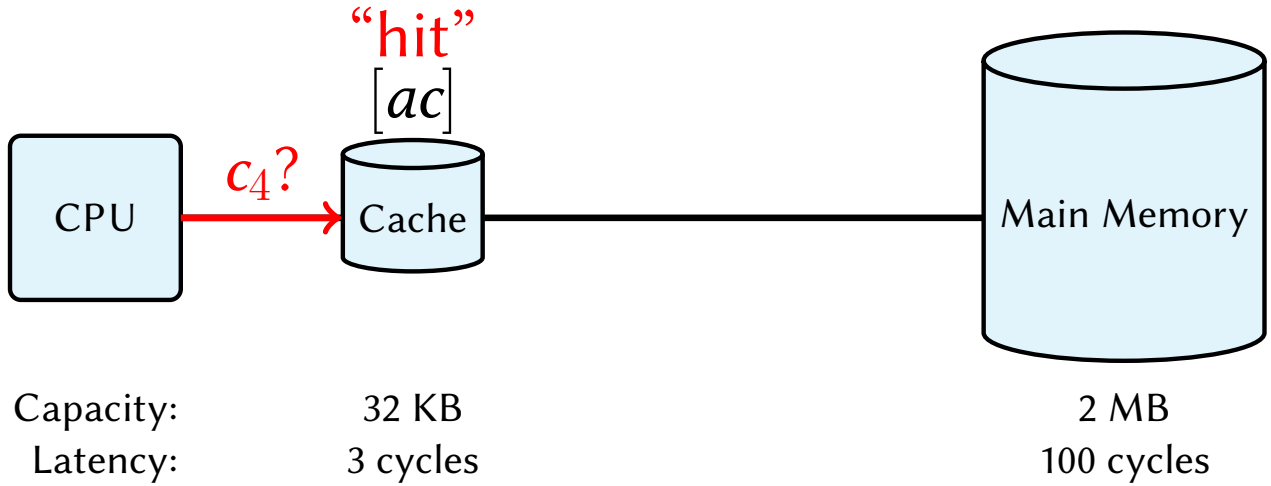
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Caches, rappels d'archi

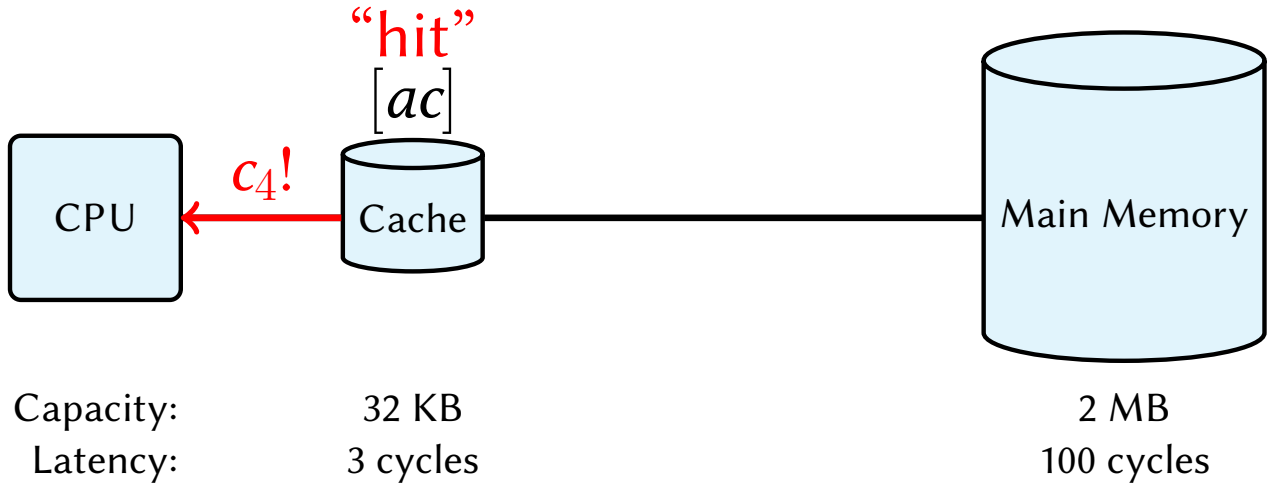
Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm



- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

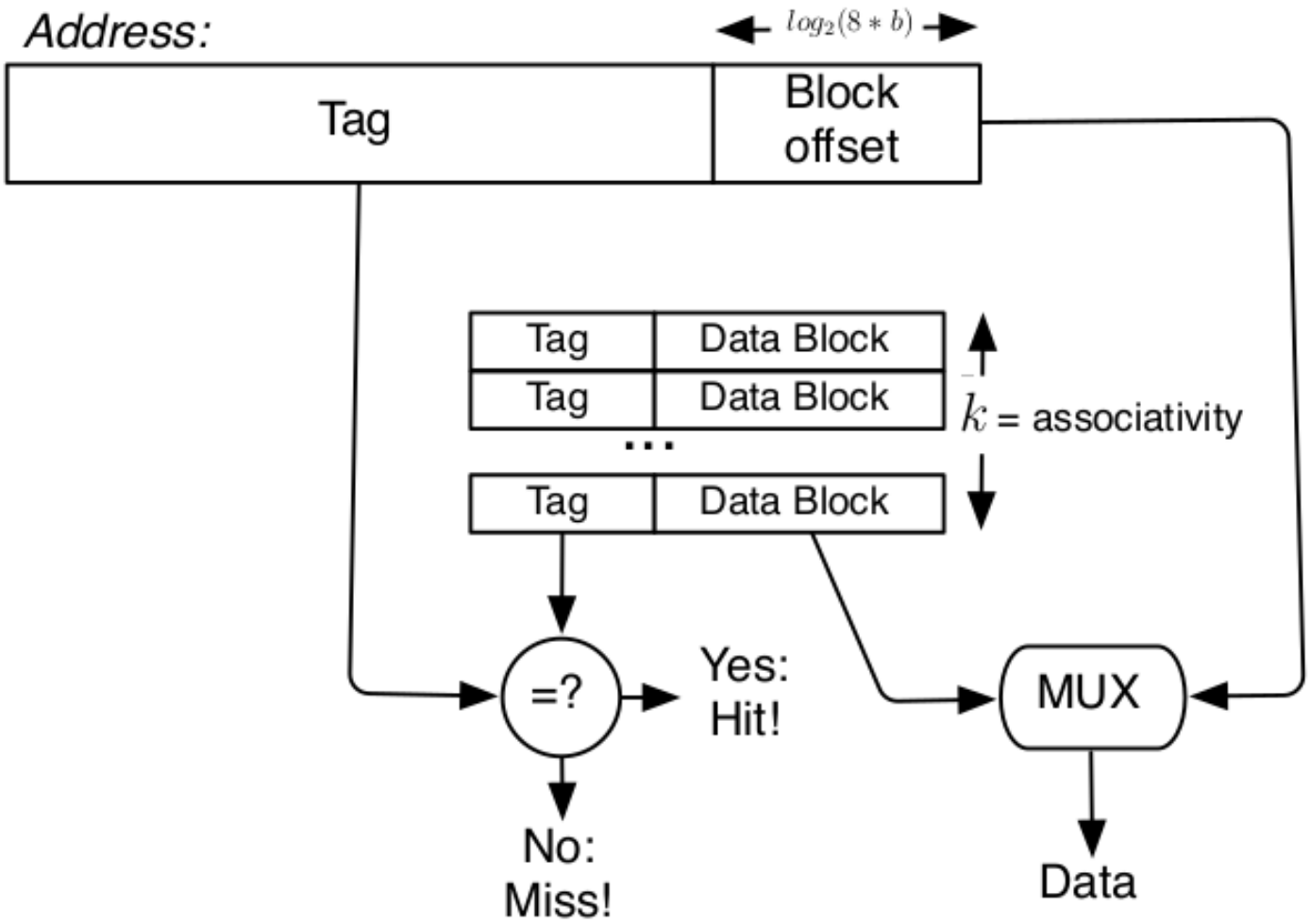
Caches, rappels d'archi

Crédit images cache + analyses de cache : J. Reineke, R. Wilhelm

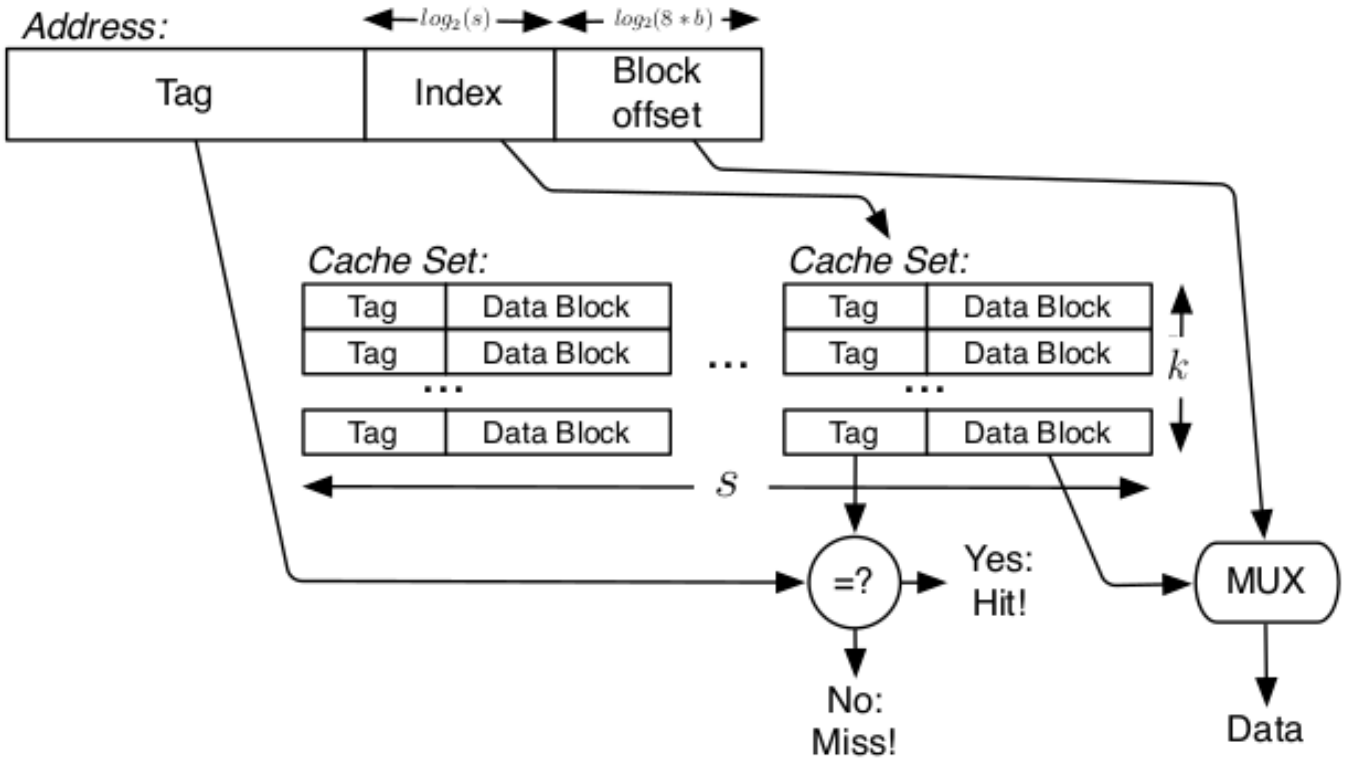


- Petites parties de mémoire rapides qui permettent d'exploiter la **localité spatiale** et temporelle.

Différents types de caches 1/3



Différents types de caches 2/3



Différents types de caches 3/3

Politiques de remplacement :

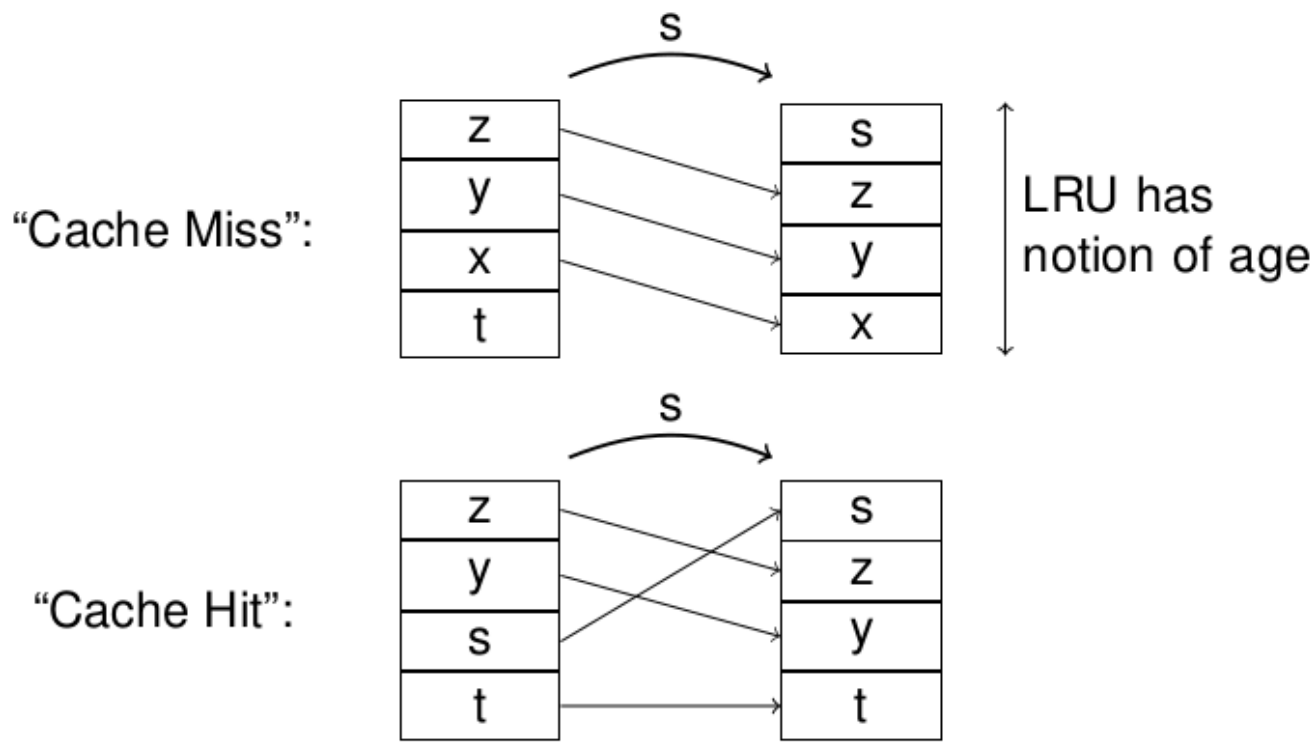
- Least-Recently-Used (LRU) : Pentium 1, MIPS
- First-In First-Out (FIFO) : Intel XS CALE , ARM9, ARM11, ...
- Pseudo-LRU (PLRU) : Intel Pentium II-IV et POWER PC 75 X (voir plus tard)
- Most-Recently-Used (MRU) : meilleur dans le cas de “repeated scans over large datasets”

▶On veut prédire les “hit” et les “miss”.

Cache partitionnés en « sets » = compositions de caches totalement associatifs.

Analyses de cache pour LRU, principes

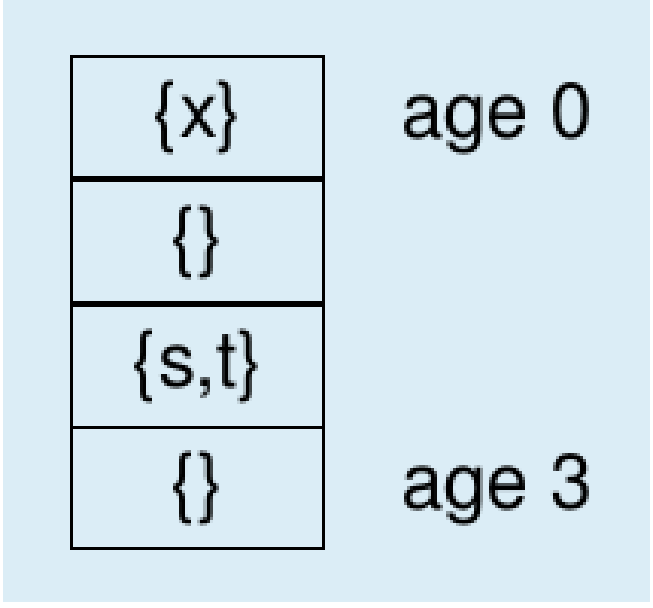
Comportement **concret** pour LRU :



Must analysis : prédire les cache hits

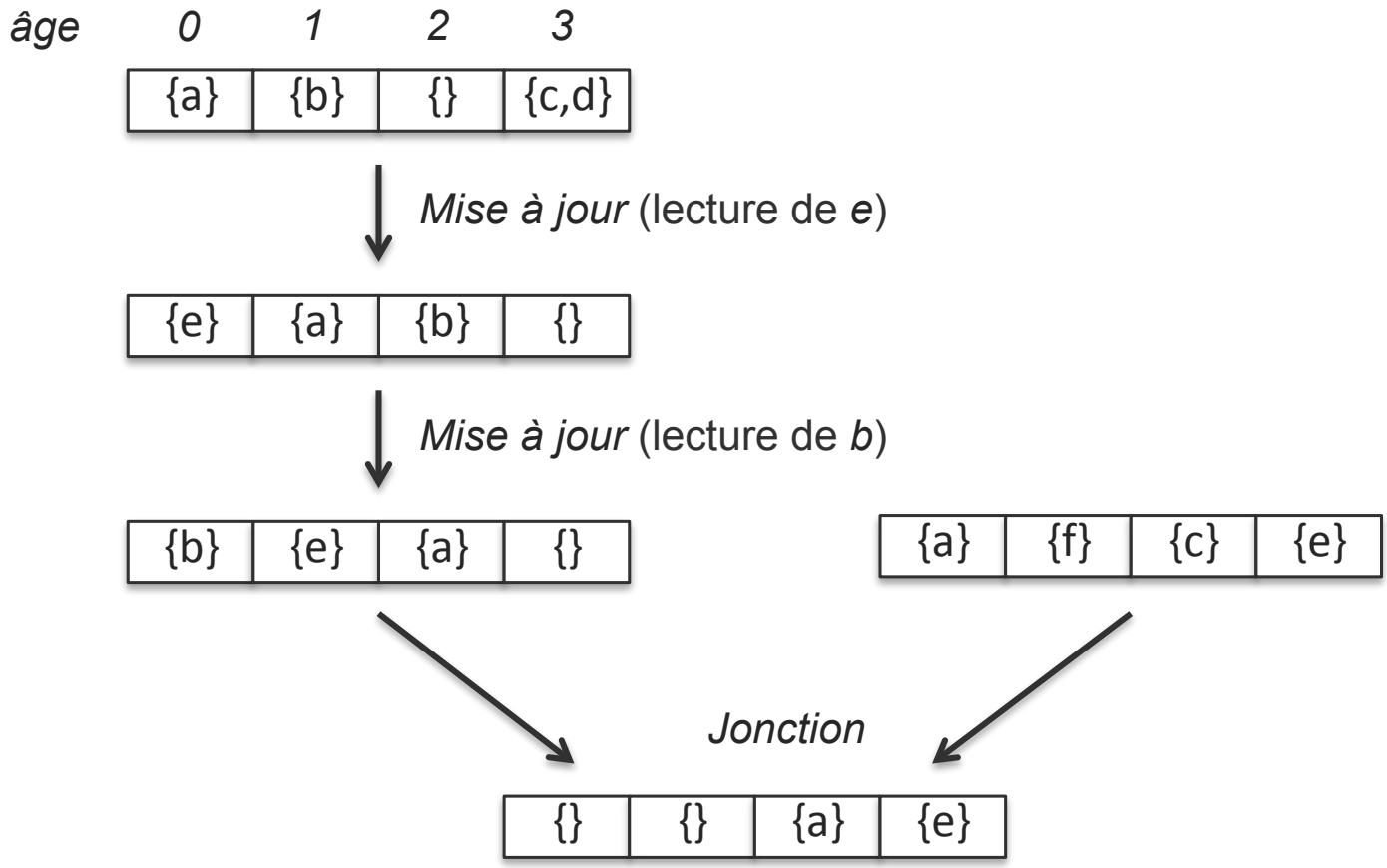
Idée : on maintient des bornes supérieures sur les âges du contenu du cache (si cette borne est inférieure à l'associativité, cela signifie que la donnée est **vraiment** dans le cache)

Exemple de **valeur abstraite**:



- x a un âge 0, $age(s) \leq 2$, ...

Must analysis, opérations

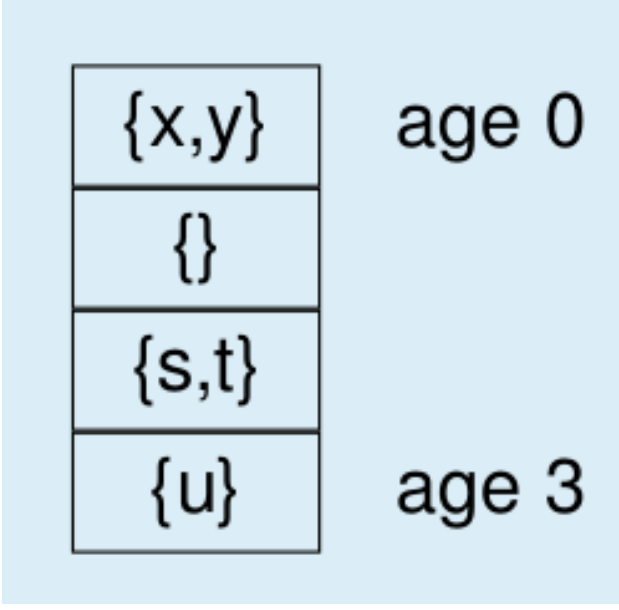


▶ Itérer jusqu'à point fixe. Recherche active pour augmenter la précision. Cette technique s'appelle l'interprétation abstraite. C'est le couteau suisse des analyses statiques.

May analysis : prédire les cache misses (défauts)

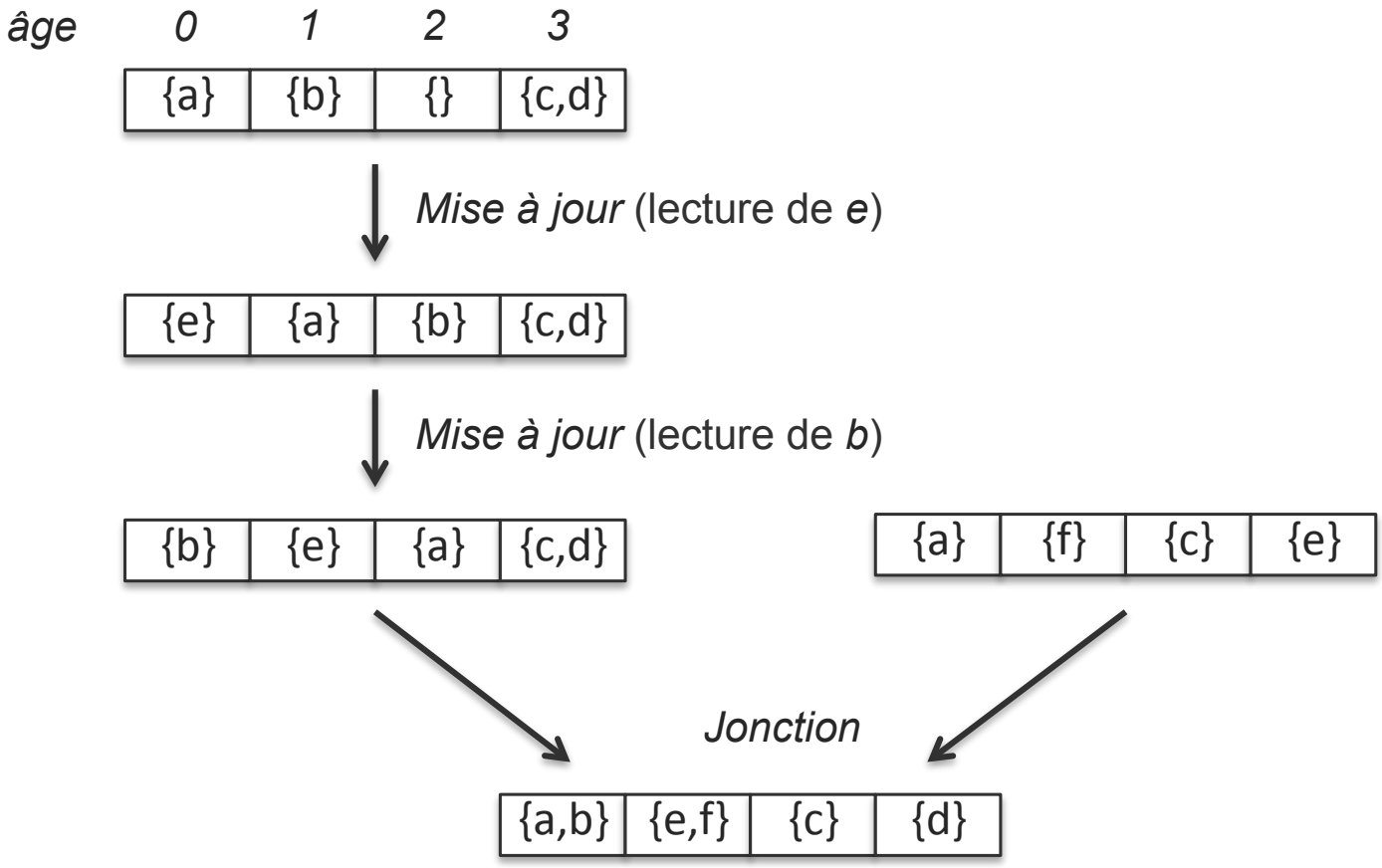
Idée : on maintient des bornes inférieures des âges du contenu du cache (si $\geq$ associativité, cela signifie que la donnée n'est **vraiment pas** dans le cache)

Exemple de **valeur abstraite**:



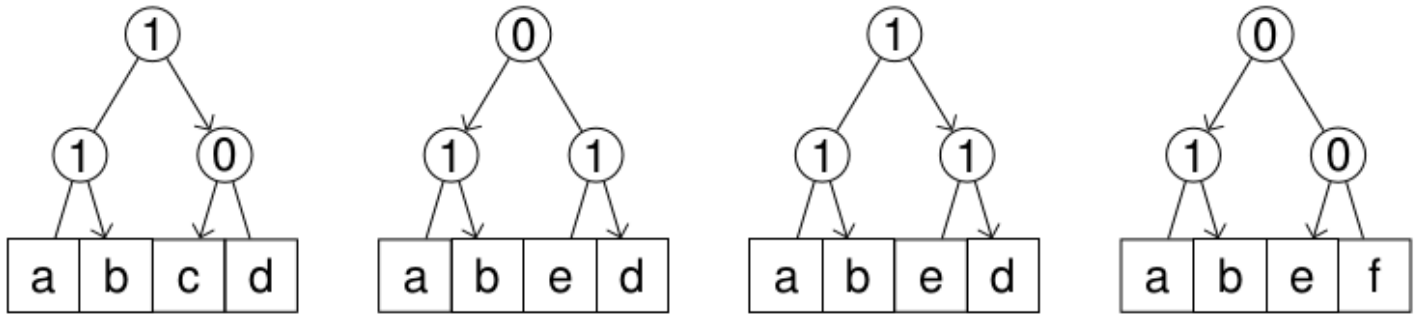
- x, y ont un âge ≥ 0, $age(s, t) \geq 2$, ...

May analysis



Au delà - Pseudo LRU

"Sélectionner un élément qui n'a probablement pas été accédé récemment" : arbres de recherche (0 = gauche, 1 = droite)



Initial cache-set state $[a, b, c, d]_{110}$.

After a miss on e. State: $[a, b, e, d]_{011}$.

After a hit on a. State: $[a, b, e, d]_{111}$.

After a miss on f. State: $[a, b, e, f]_{010}$.

- le voisinage est recalculé $\Rightarrow$ b reste dans le cache.

Plan

La préemption, pourquoi ? (rappel)

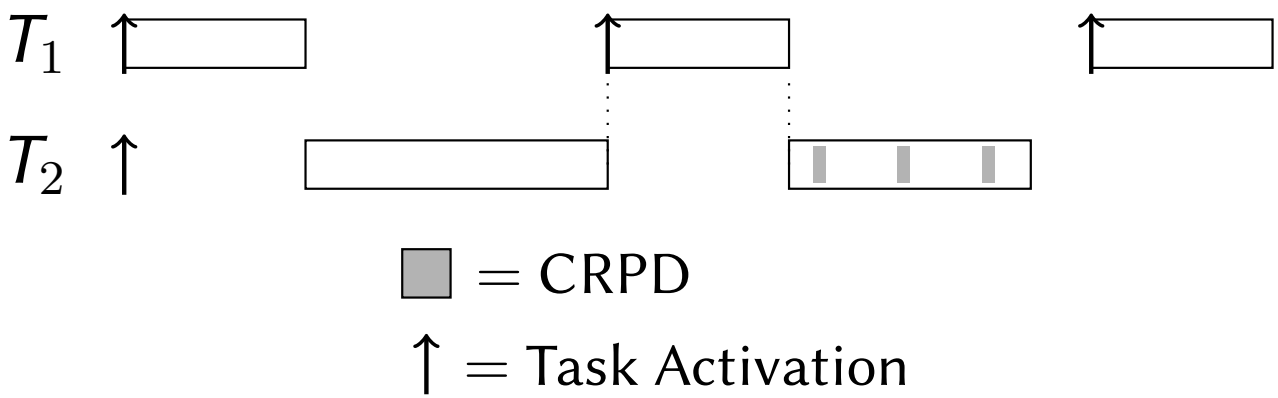
- Souvent la préemption permet l'ordonnançabilité d'un ensemble de tâches.
- Souvent les tâches à échéance courte ne sont pas ordonnançables sans préemption.

Exemple : Soient les tâches T_1 et T_2, de périodes $P_1 = 2, P_2 = 8$, échéances $D_1 = P_1, D_2 = P_2$, et temps d'exécution $C_1 = 1, C_2 = 3$.

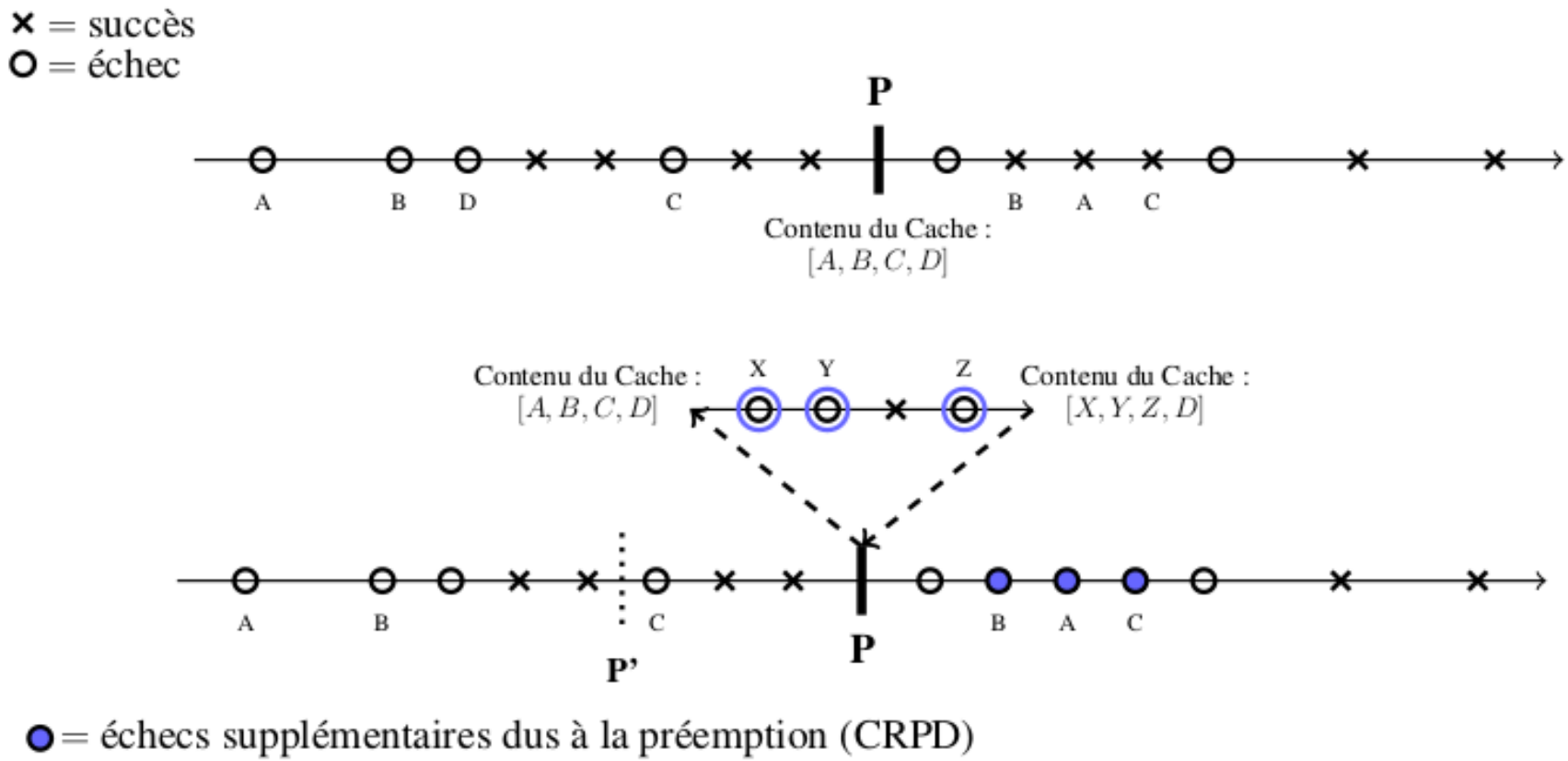
Dessins !

La préemption n'est pas gratuite

- La tâche qui préempte perturbe les états jouant sur la performance, comme les caches et les pipelines.
- Lors de la reprise d'exécution de la tâche prémptée, cette perturbation cause des défauts de cache supplémentaires.
- On appelle le temps supplémentaire dû à ces défaut de cache le *cache-related preemption delay* (CRPD).



Exemple d'effet de la préemption



- ici : $CRPD = 3 * cost(miss)$

Image C. Maiza - livre "Ordonnancement des systèmes temps-réel."

Où prendre en compte le coût de préemption ?

- Intégrer dans l'analyse de WCET: [Schneider2000]
  - supposer des cache misses partout
  - usage facile dans les analyses d'ordonnançabilité. (mais très pessimiste)
- WCET Analysis + CRPD Analysis: [Altmeyer2012]
  - $WCET_{bound} + n \cdot CRPD_{bound} \geq$ temps d'exécution avec "jusqu'à n préemptions".
  - plus précis mais très peu d'analyses d'ordonnançabilité savent prendre en compte.

Analyses de CRPD, quoi faire?

- Tâche préemptée:
  combien de blocs *utiles* dans le cache ?
- Tâche qui préempte:
  quel dommage cause-t-elle sur le cache de *n'importe quelle autre* tâche ?
- Les deux :
  quel dommage la tâche qui préempte cause-t-elle sur le cache utile de la préemptée ?

Conclusion

L'analyse de WCET est un domaine de recherche très foisonnant, en particulier :

- Prise en compte des caractéristiques architecturales plus complexes;
- Le couplage avec l'ordonnancement;
- Les machines parallèles.

Bibliography I

Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord.
Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs.
In *Static Analysis Symposium*, Perpignan, France, September 2010.

Sebastian Altmeyer, Robert I. Davis, and Claire Maiza.
Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems.
Real-Time Systems, 48(5):499–526, 2012.

Bibliography II

Julien Henry, Mihail Asavoae, David Monniaux, and Claire Maiza.
How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics.
In Youtao Zhang and Prasad Kulkarni, editors, *LCTES*, pages 43–52. ACM, 2014.

Joerg Schneider.
Cache and pipeline sensitive fixed priority scheduling for preemptive real-time systems.
In *Real-Time Systemes Symposium*, 2000.