Eléments d'Algorithmique

Joël Quinqueton
jq@lirmm.fr
PCSI1, Lycée Joffre

L'informatique (1)

- 10% des investissements des sociétés (hors bâtiments)
- Croissance très rapide du nombre de cadres et techniciens informaticiens [1982-1991]
- Science encore jeune (30 ans)
- Gigantesque cathédrale de constructions matérielles et intellectuelles

Cori, Levy

L'informatique (2)

- Il existe une science de l'informatique
- Plusieurs théories imbriquées
  - logique et calculabilité, algorithmique et analyse d'algorithmes, conception et sémantique des langages de programmation, bases de données, principes des systèmes d'exploitation, architectures des ordinateurs et évaluation de leurs performances, réseaux et protocoles, langages formels et compilation, codes et cryptographie, apprentissage et zero-knowledge algorithms, calcul formel, démonstration automatique, conception et vérification de circuits, vérification et validation de programmes, temps réel et logiques temporelles, traitement d'images et vision, synthèse d'image, robotique, ...

L’informatique (3)

- La jeunesse de l'informatique permet à certains de nier son aspect scientifique
- Mythe du hacker («fous de la programmation»)
  - hacker [...] n. 2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.

L’informatique (4)

- La jeunesse de l'informatique permet à certains de nier son aspect scientifique
- Mythes du hacker («fous de la programmation»)
  - Programmeur préférant ignorer toute considération théorique qui puisse l’aider dans ses constructions souvent très habiles

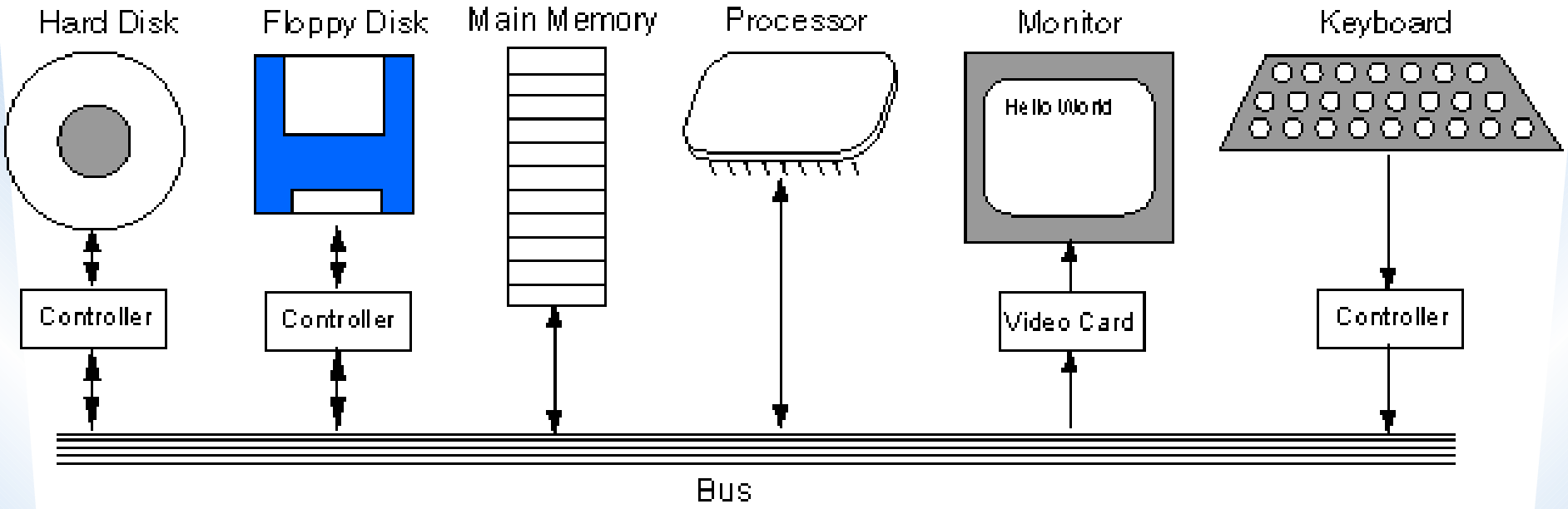
L’informatique (5)

- Une autre caractéristique de l'informatique est le côté instable des programmes
- Les phénomènes continus sont rares en informatique
  - Une panne n'est en général pas le résultat d'une dégradation perceptible. Elle arrive simplement brutalement.

L’informatique (6)

- Une autre caractéristique de l'informatique est le côté instable des programmes
  - C'est ce côté *exact* de l'informatique qui est très attrayant
  - En informatique, il y a peu de solutions approchées
  - En informatique, il y a une certaine notion de l'*exactitude*

Architecture d'un ordinateur (1)

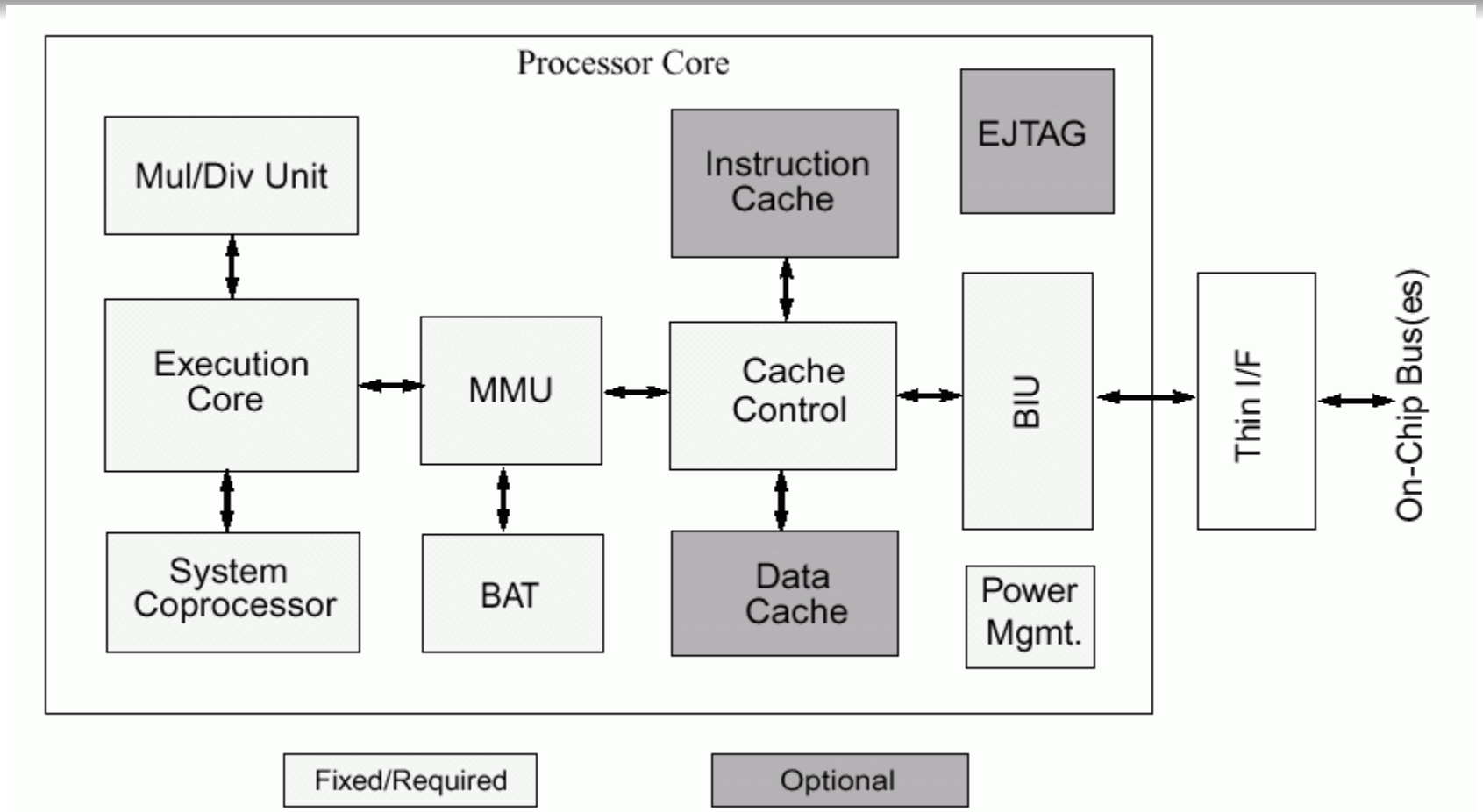


Main Components of a Computer System

Architecture d'un ordinateur (2)

- Toutes les opérations effectuées en machine sont exécutées *par le processeur*
- *Aucune* opération n'est faite en mémoire principale
- Le processeur opère sur des données préalablement stockée en mémoire

Architecture d'un ordinateur (3)



Processeur MIPS 32 4kP

Architecture d'un ordinateur (4)

- La mémoire est une composante constituée de circuits spécialisés dans la localisation de mots mémoire à partir d'une adresse
- Une adresse = un nombre = un endroit où se trouve une donnée

Architecture d'un ordinateur (5)

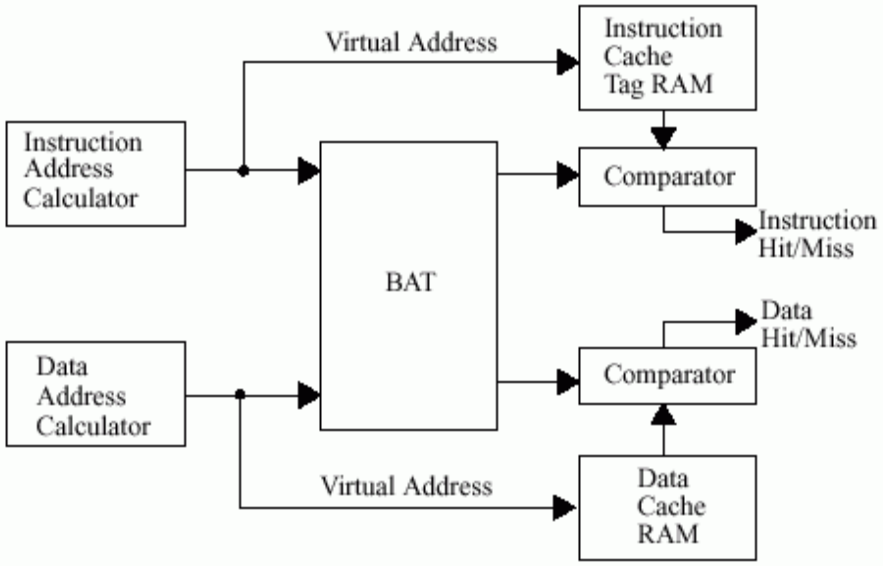


Figure 4 Address Translation During a Cache Access

Segment	Virtual Address Range	Cacheability
KSeg1	0xA000_0000- 0xBFFF_FFFF	Always uncacheable
KSeg2	0xC000_0000- 0xDFFF_FFFF	Controlled by K23 field (bits 30:28) of the Config register. See Table 4 for mapping.
KSeg3	0xE000_0000- 0xFFFF_FFFF	Controlled by K23 field (bits 30:28) of the Config register. See Table 4 for mapping.

Architecture d'un ordinateur (6)

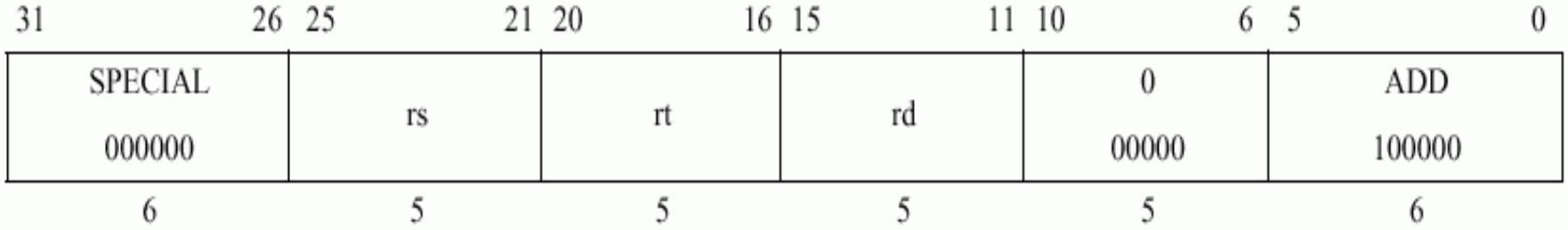
- Le processeur sait faire un nombre fini d'opérations
- Tout programme doit être traduit en une suite d'opérations qu'il sait exécuter
- Ces opérations sont souvent appelées *instructions machines*

Architecture d'un ordinateur (7)

- Pour additionner deux nombres *m* et *n.*
  - Amener l'*adresse de m* dans un *registre*
  - Amener dans un second registre le contenu de la mémoire vive à cette adresse
  - Idem pour le second nombre *n*
  - Exécuter l'addition (module arithmétique)
  - Amener dans un registre l'adresse où stocker le résultat
  - Le stocker dans la mémoire vive

Architecture d'un ordinateur (8)

Add Word **ADD**



Format: `ADD rd, rs, rt` **MIPS32**

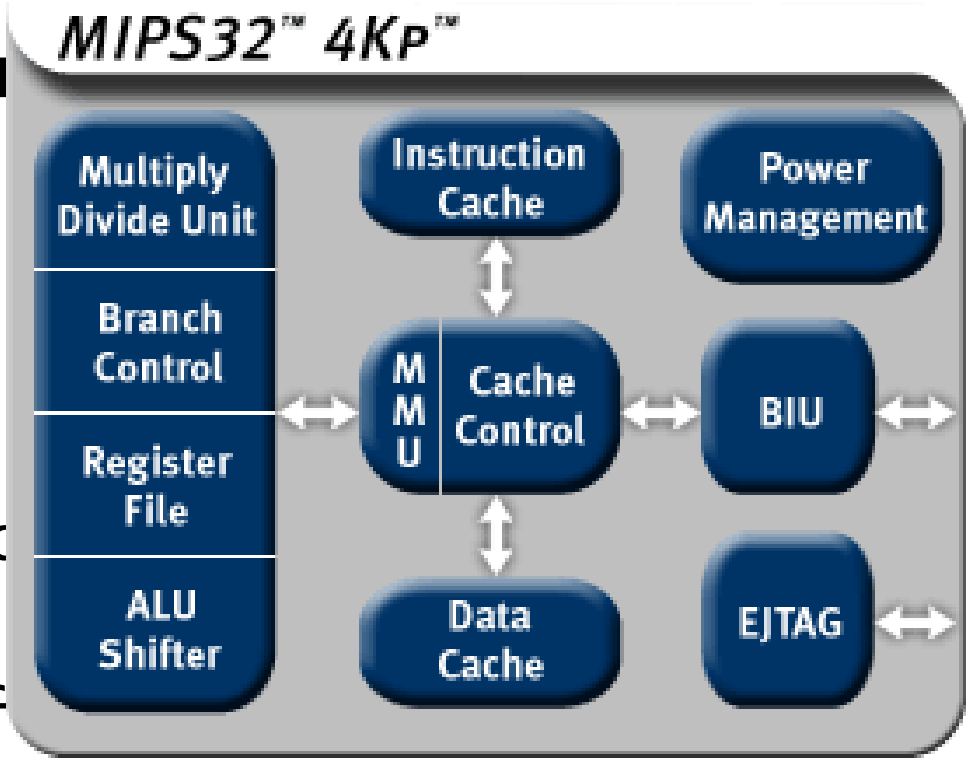
Purpose:

To add 32-bit integers. If an overflow occurs, then trap.

Description: `rd ← rs + rt`

Architecture d'un ordinateur (8)

- Un processeur est co
  modules spécialisés
  - Module arithmétique
  - Contrôle de flux
  - Calcul d'adresse
  - Traitement des interac
    périphériques
- Deux processeurs dif
  des instructions machine dont il dispose



Interprétation/Compilation

- Les langages de programmation de haut niveau permettent de s'abstraire du matériel
- La compilation consiste à traduire un langage de haut niveau en langage machine

```
Proc(x) if x>0 then error("Hello World!");
```



```
00011011011010010100 000110110101
11011010010010010100 001110110110
00110011011010010100 000110110111
…
```

Algorithmique (1)

- Un programme présente deux aspects
  - un contenu et une forme
  - un sens et une grammaire
- Pour l'ordinateur, il suffit que le programme soit correct au niveau de la forme (la syntaxe)

Algorithmique (2)

- L’ordinateur effectuera toujours les manipulations commandées par un programme syntaxiquement correct
  - La première étape de la compilation consiste à vérifier que le programme écrit est syntaxiquement correct
  - Si l'on fait une erreur de syntaxe, le compilateur affiche un message d'erreur

Algorithmique (3)

- La cohérence du programme (du contenu) n'est pas examinée ou évaluée par le compilateur
- L'analyse du problème à traiter, la preuve de la cohérence et de la pertinence de sa solution sont préalable à l’écriture du programme

Algorithmique (4)

- On doit
  - fixer l'objectif du programme
  - établir la liste des données à manipuler et des opérations à exécuter, et les ordonner.
- La description de la suite des opérations élémentaires ordonnées capables de résoudre le problème posé constitue un *algorithme*
- *Al-Khwārizmī* (783-850)

Exemple du tri

- Techniques de tris élémentaires
  - Qu'est-ce qu'un tri?
  - On suppose qu'on se donne une suite de *N* nombres entiers et on veut les ranger en ordre croissant au sens large.

Ainsi, pour $N = 10$, la suite

18, 3, 10, 25, 9, 3, 11, 13, 23, 8

devra devenir

3, 3, 8, 9, 10, 11, 13, 18, 23, 25

Tri par sélection

- On trie N nombres entiers, rangés dans un tableau
- Tri par sélection (le plus simple)
  - Trouver l'emplacement de l'élément le plus grand du tableau : l'entier m tel que, pour tout i, $a_m \geq a_i$
  - On échange les éléments a_N et a_m
  - Puis on recommence ces opérations sur la suite a_1, a_2, ..., a_{N-1}

Tri par sélection

- Cette solution utilise la solution à un sous-problème
  - La recherche de la *position* du plus grand élément d'un tableau

```
indElmtMax = 0 // pos de l'élmt max courant
j = 0
tant que j < N faire {
  si (T[j] > T[indElmtMax])
    alors indElmtMax = j
  j = j + 1
}
```

Tri par sélection

- Ses performances ?
  - Evaluation du *coût* de l 'algorithme
  - Combien d'opérations doit-on effectuer pour trier un tableau de *N* entiers ?

Tri par sélection

- Coût de l'algorithme

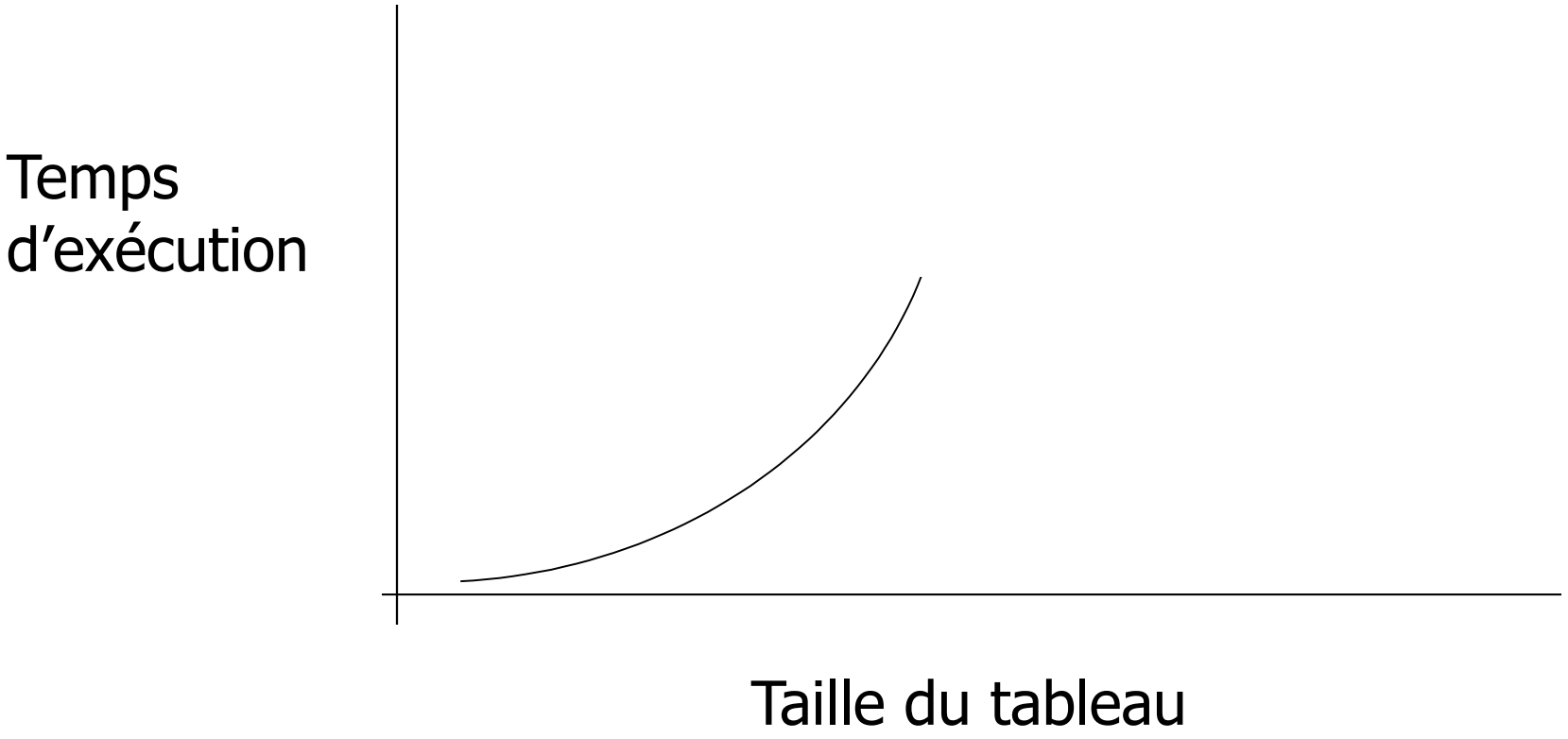
$$N + \sum_{i=0}^{N-2} N - i - 1 = N(N-1)/2$$

Echanges

Comparaisons

Tri par sélection

- Coût de l'algorithme



Autres algorithmes de tri

- Tri par sélection
- Tri par propagation ou « à bulles »
- Tri par insertion
- Tri rapide
- Tri par fusion
- voir `http://interstices.info/jcms/c_6973/les-algorithmes-de-tri`

Tri « à bulles »

- Parcourir le tableau de 1 à N
- Permuter toute paire d'éléments consécutifs non ordonnés
- Le plus grand se retrouve en N
- Recommencer la procédure de 1 à N-1

```
procedure triBulle(entier[] tab)
   entier i, k;
   entier tmp;
   pour (i de N à 2 en décrémentant de 1) faire
      pour (k de 1 à i-1 en incrémentant de 1) faire
         si (tab[k] > tab[k+1]) alors
            tmp <- tab[k];
            tab[k] <- tab[k+1];
            tab[k+1] <- tmp;
         fin si
      fin pour
   fin pour
fin procedure
```

Tri par insertion

- les (i-1) premières cartes, tab[1],..., tab[i-1] sont supposées triées
- placer la i^{éme} carte, tab[i], à sa place parmi les (i-1) déjà triées
- Répéter pour i de 2 jusqu'à N

```
procedure triInsertion(entier[] tab)
   entier i, k;
   entier tmp;
   pour (i de 2 à N en incrémentant de 1)
faire
      tmp <- tab[i];
      k <- i;
      tant que (k > 1 et tab[k - 1] > tmp) faire
         tab[k] <- tab[k - 1];
         k <- k - 1;
      fin tant que
   tab[k] <- tmp;
   fin pour
fin procedure
```

Tri rapide (1)

- Pivot = élément au hasard dans le tableau.
- Éléments <pivot au début
- Éléments >pivot à la fin

```
fonction partition(entier[] tab, entier debut, entier fin, entier indicePivot)
retourne un entier
  entier i;
  entier pivot <- tab[indicePivot];
  entier k <- debut;
  entier tmp;
  pour (i de debut à fin en incrémentant de 1) faire
    si (tab[i] < pivot) alors
      tmp <- tab[i];
      tab[i] <- tab[k];
      tab[k] <- tmp;
      k <- k + 1;
    fin si
  fin pour
  tab[k] <- pivot;
  retourner k;
fin fonction
```

Tri rapide (2)

- placer la valeur de pivot à sa place définitive
- répéter ensuite récursivement la procédure sur chacune des partitions créées

```
procedure triRapideR(entier[] tab, entier debut, entier fin)
   si (fin > debut) alors
      pivot0 <- entier aléatoire entre 1 et N;
      entier indicePivot <- partition(tab, debut, fin, pivot0);
      triRapideR(tab, debut, indicePivot - 1);
      triRapideR(tab, indicePivot + 1, fin);
   fin si
fin procedure

procedure triRapide(entier[] tab)
   triRapideR(tab, 1, N);
fin procedure
```

Tri par fusion (1)

- Choisir le pivot
- Obtenir deux suites d'éléments triés, de longueurs respectives L1 et L2

```
procedure fusion(entier[] tab, entier[] tmp, entier debut,
entier mil, entier fin)
  entier k;
  entier i <- debut;
  entier j <- mil + 1;
  pour (k de debut à fin en incrémentant de 1) faire
    si ((j > fin) ou (i <= mil et tab[i] < tab[j])) alors
      tmp[k] <- tab[i];
      i <- i + 1;
    sinon
      tmp[k] <- tab[j];
      j <- j + 1;
    fin si
  fin pour
  pour (k de debut à fin en incrémentant de 1) faire
  tab[k] <- tmp[k];
  fin pour
fin procedure
```

Tri par fusion (2)

- obtenir une troisième suite d'éléments triés de longueur L1 + L2, par « interclassement » (ou fusion) des deux précédentes suites

```
procedure triFusionR(entier[] tab, entier[] tmp, entier debut, entier fin)
  si (debut < fin) alors
    entier milieu <- (debut + fin)/2;
    triFusionR(tab, tmp, debut, milieu);
    triFusionR(tab, tmp, milieu + 1, fin);
    fusion(tab, tmp, debut, milieu, fin);
  fin si
fin procedure

procedure triFusion(entier[] tab)
  entier[] tmp <- tableau de taille N;
  triFusionR(tab, tmp, 1, N);
fin procedure
```

Étude de complexité

- Tri par sélection, Tri « à bulles »:
  - Complexité quadratique (dans le pire des cas)
- Tri par insertion
  - Id, mais très rapide si la liste est déjà triée
- Tri rapide
  - N^2 (pire des cas), $N*\log_2(N)$ (moyenne)
- Tri par fusion
  - $N*\log_2(N)$ (pire des cas) mais coût en mémoire

Conclusion

- Intérêt de la complexité: vitesse et taille du problème
- Si les machines vont 10 fois plus vite:
  - $N*\log_2(N)$: ~10 fois plus grand
  - N^2: ~3 fois plus grand
  - N^4: ~2 fois plus grand
  - …