ARCHITECTURES DE CALCUL MODERNES

…et pourquoi un développeur devrait s'en soucier

Hadrien Grasland

LAL – Orsay

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

La fin d’une époque

- Nous sortons d’une ère de performance « gratuite »
  - Il suffisait de racheter du matériel pour accélérer son code…
  - …ou de mettre à jour / mieux configurer son compilateur…
  - …donc on enseignait peu l’art de la performance logicielle (à part les bases de complexité algorithmique)

La fin d'une époque

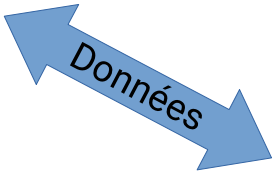
- Nous sortons d'une ère de performance « gratuite »
  - Il suffisait de racheter du matériel pour accélérer son code…
  - …ou de mettre à jour / mieux configurer son compilateur…
  - …donc on enseignait peu l'art de la performance logicielle (à part les bases de complexité algorithmique)
- Il va falloir à nouveau se retrousser les manches
  - Le matériel moderne est riche en pièges de performance
  - Certaines fonctionnalités nécessitent un support explicite
  - Il est donc important d'en avoir un bon **modèle conceptuel**

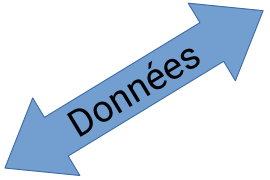
Avertissement

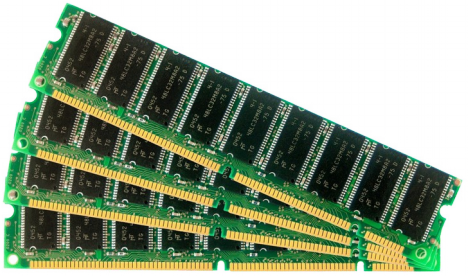
- Dans ce qui suit, nous nous concentrerons sur les **calculs**…
  - Donc principalement les affaires des CPUs (ou des GPUs)
- …mais n'oubliez pas qu'il n'y a pas que le calcul dans la vie
  - On pourrait remplir un deuxième séminaire avec la vie secrète des **E/S disques, réseau**…

Modèle de von Neumann

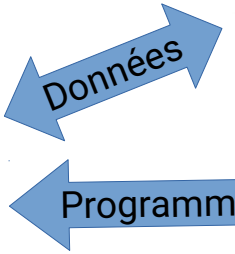
- C’est le modèle conceptuel classique des programmeurs
- Il représentait bien les premiers CPUs (~1960)

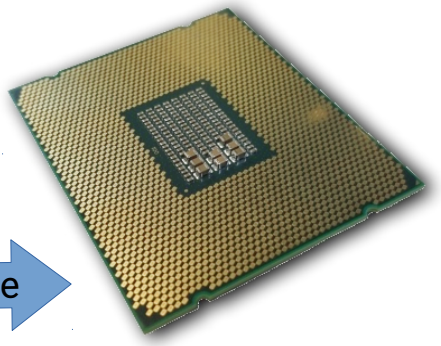






Mémoire principale





Processeur

Entrées/Sorties

Von Neumann en action

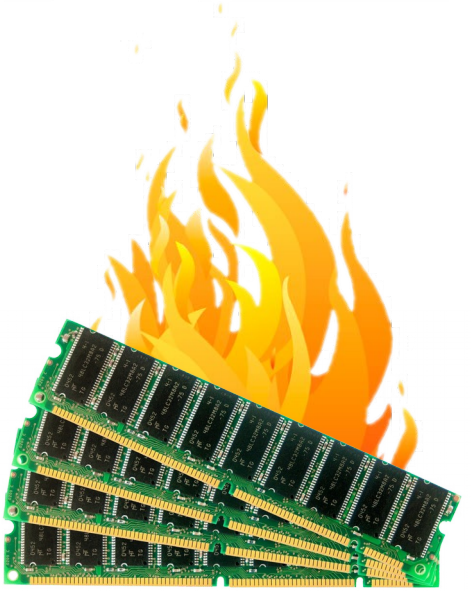
- Calculons la norme d'un vecteur 2D $n = \sqrt{x^2 + y^2}$
  - x ← Entrée
  - x2 ← x * x
  - y ← Entrée
  - y2 ← y * y
  - n2 ← x2 + y2
  - n ← sqrt(n2)
  - Sortie ← n

Von Neumann en action

- Calculons la norme d'un vecteur 2D $n = \sqrt{x^2 + y^2}$
  - x ← Entrée (Mémoire, E/S)
  - x2 ← x * x (Mémoire, processeur)
  - y ← Entrée (Mémoire, E/S)
  - y2 ← y * y (Mémoire, processeur)
  - n2 ← x2 + y2 (Mémoire, processeur)
  - n ← sqrt(n2) (Mémoire, processeur)
  - Sortie ← n (Mémoire, E/S)

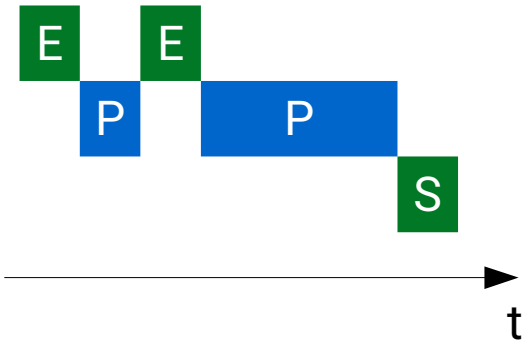
Von Neumann en action

- Calculons la norme d'un vecteur 2D $n = \sqrt{x^2 + y^2}$
  - x ← Entrée (Mémoire, E/S)
  - x2 ← x * x (Mémoire, processeur)
  - y ← Entrée (Mémoire, E/S)
  - y2 ← y * y (Mémoire, processeur)
  - n2 ← x2 + y2 (Mémoire, processeur)
  - n ← sqrt(n2) (Mémoire, processeur)
  - Sortie ← n (Mémoire, E/S)



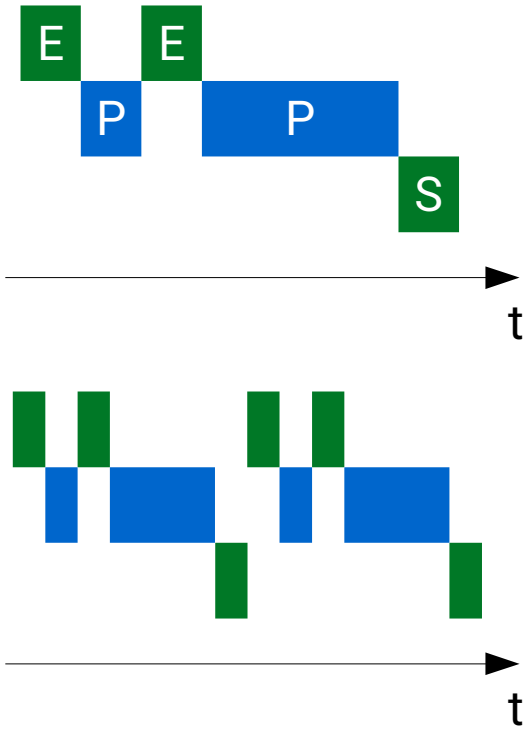
Von Neumann en action

- Calculons la norme d'un vecteur 2D $n = \sqrt{x^2 + y^2}$
  - x ← Entrée (Mémoire, E/S)
  - x2 ← x * x (Mémoire, processeur)
  - y ← Entrée (Mémoire, E/S)
  - y2 ← y * y (Mémoire, processeur)
  - n2 ← x2 + y2 (Mémoire, processeur)
  - n ← sqrt(n2) (Mémoire, processeur)
  - Sortie ← n (Mémoire, E/S)



Von Neumann en action

- Calculons la norme d'un vecteur 2D $n = \sqrt{x^2 + y^2}$
  - x ← Entrée (Mémoire, E/S)
  - x2 ← x * x (Mémoire, processeur)
  - y ← Entrée (Mémoire, E/S)
  - y2 ← y * y (Mémoire, processeur)
  - n2 ← x2 + y2 (Mémoire, processeur)
  - n ← sqrt(n2) (Mémoire, processeur)
  - Sortie ← n (Mémoire, E/S)



Limites de ce modèle

- Une machine de von Neumann « pure » serait **inefficace**
  - La mémoire principale et les E/S limitent la performance
  - Les ressources systèmes s’attendent mutuellement

Limites de ce modèle

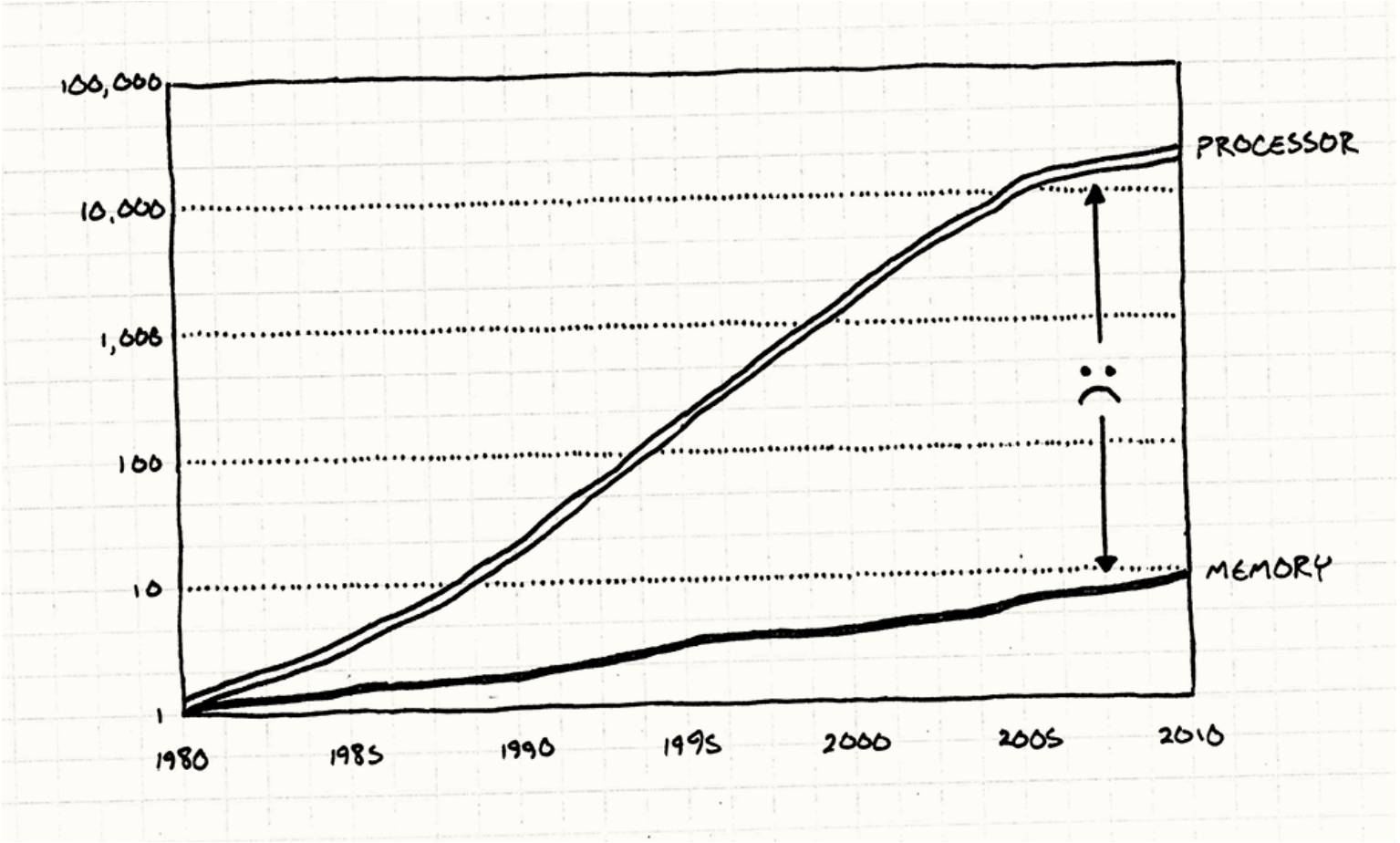
- Une machine de von Neumann « pure » serait **inefficace**
  - La mémoire principale et les E/S limitent la performance
  - Les ressources systèmes s'attendent mutuellement
- La performance du CPU n'y dépend que de sa **fréquence**…
  - …qui a des limites physiques (thermiques), déjà atteintes

Limites de ce modèle

- Une machine de von Neumann « pure » serait **inefficace**
  - La mémoire principale et les E/S limitent la performance
  - Les ressources systèmes s'attendent mutuellement
- La performance du CPU n'y dépend que de sa **fréquence**…
  - …qui a des limites physiques (thermiques), déjà atteintes
- Comment dépasser ces limites ?
  - Utiliser des **mémoires plus rapides** (caches)
  - Avoir des **entrées-sorties autonomes** (DMA, non-bloquant)
  - Faire **plusieurs calculs à la fois** (parallélisme, concurrence)

Accélérer la mémoire

Un goulot d'étranglement



http://gameprogrammingpatterns.com/data-locality.html

Comment aller plus vite ?

- Le programmeur voudrait une mémoire qui soit :
  - aussi rapide que le CPU (débit, et surtout **latence**) ;
  - vaste (capacité) et bon marché (capacité/€) ;
  - non-volatile, économe en énergie…
- Mais le monde est fait de compromis :
  - plus rapide = plus petit, plus cher, plus gourmand...

Comment aller plus vite ?

- Le programmeur voudrait une mémoire qui soit :
  - aussi rapide que le CPU (débit, et surtout **latence**) ;
  - vaste (capacité) et bon marché (capacité/€) ;
  - non-volatile, économe en énergie…
- Mais le monde est fait de compromis :
  - plus rapide = plus petit, plus cher, plus gourmand...
- Une solution : combiner **différents types** de mémoires

Hiérarchie mémoire

- L'ordinateur moderne combine de nombreuses mémoires :
  - registres CPU (1 cycle) ;
  - caches (~3, 15, 45 cycles) ;
  - DRAM principale (> 100 cycles) ;
  - disque dur, SSD flash (> 10^5 cycles) ;
  - accès internet aux données (> 10^7 cycles).

Petit et rapide

Grand et lent

Hiérarchie mémoire

- L'ordinateur moderne combine de nombreuses mémoires :
  - registres CPU (1 cycle) ;
  - caches (~3, 15, 45 cycles) ;
  - DRAM principale (> 100 cycles) ;
  - disque dur, SSD flash (> 10^5 cycles) ;
  - accès internet aux données (> 10^7 cycles).

Petit et rapide

Grand et lent

- On veut **placer** les données au meilleur endroit possible
  - Si possible avant même d'en avoir besoin…
  - …et en minimisant l'intervention du programmeur

Principe d’un cache

- On copie une information dans une mémoire plus rapide
  - Les **lectures** sont ainsi accélérées…
  - …mais les **écritures** doivent être synchronisées
  - On peut éviter cette opération en modifiant peu l’état **partagé**

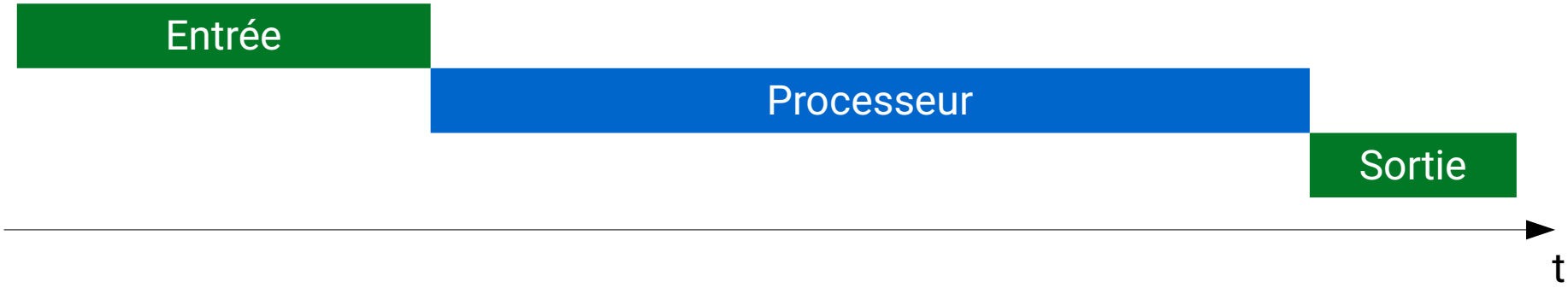
Principe d'un cache

- On copie une information dans une mémoire plus rapide
  - Les **lectures** sont ainsi accélérées…
  - …mais les **écritures** doivent être synchronisées
  - On peut éviter cette opération en modifiant peu l'état **partagé**
- La sélection de l'info en cache se base sur 3 éléments
  - Instructions du programmeur (code ordinaire et « prefetch »)
  - Localité **temporelle** (la donnée N servira plusieurs fois)
  - Localité **spatiale** (on utilise N et N+1 en même temps)

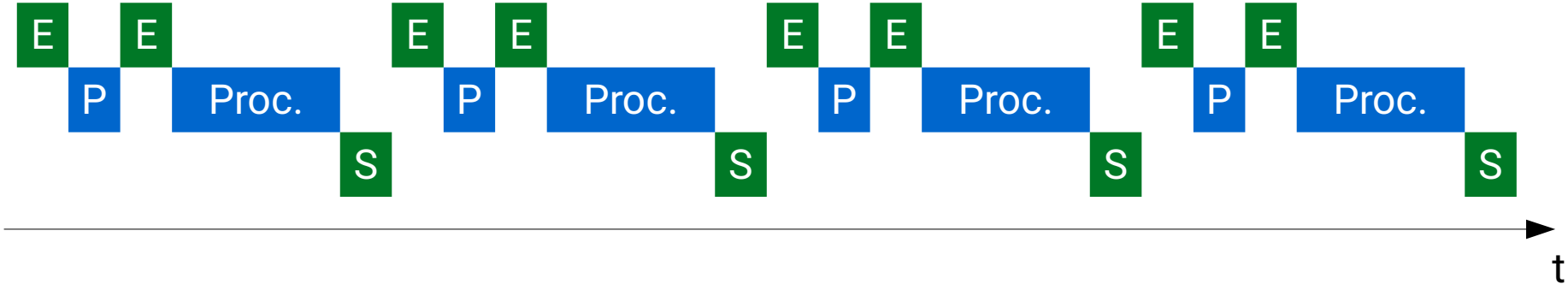
Ne pas attendre les E/S

Effet des entrées/sorties

- Un style de programme en exécution :



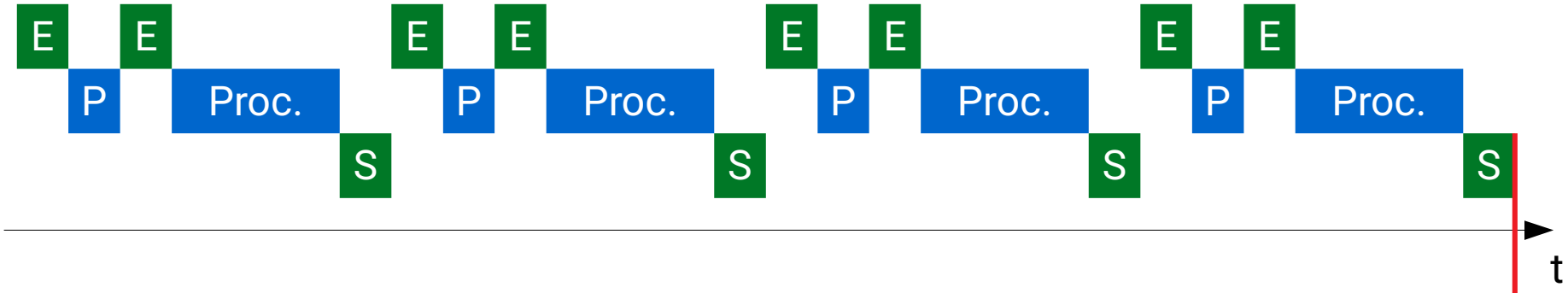
- Un autre style de programme :



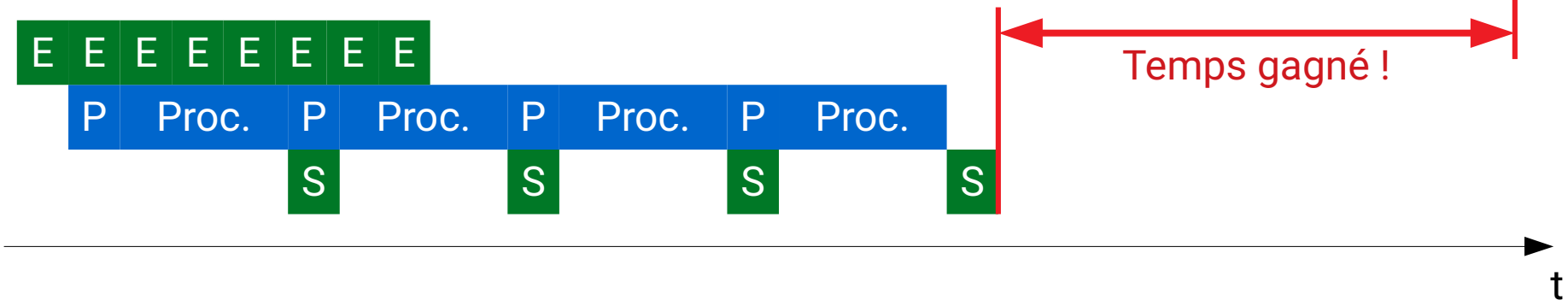
- Quelle différence ? Comment faire mieux ?

Intérêt des E/S autonomes

- Avec des entrées/sorties autonomes, ceci…



- …peut devenir cela :



- Le matériel le permet (DMA), mais sous conditions

Conditions d’E/S autonomes

- Les entrées doivent être commandées à l’avance
  - Par le programmeur (requêtes **asynchrone**)…
  - …ou par le système d’exploitation (lectures **spéculatives**)
  - Les mécanismes sont similaires à la gestion d’un cache

Conditions d'E/S autonomes

- Les entrées doivent être commandées à l'avance
  - Par le programmeur (requêtes **asynchrone**)…
  - …ou par le système d'exploitation (lectures **spéculatives**)
  - Les mécanismes sont similaires à la gestion d'un cache
- Le programme doit éviter d'attendre le matériel
  - Ne demander que ce dont on a besoin (à l'OS/lib d'optimiser)
  - Avoir des tampons d'entrée/sortie assez grands
  - Préférer les interfaces **non bloquantes** (epoll, boost::asio…)
  - Isoler les vieilles APIs bloquantes dans des threads

Parallélismes

Un problème de riche

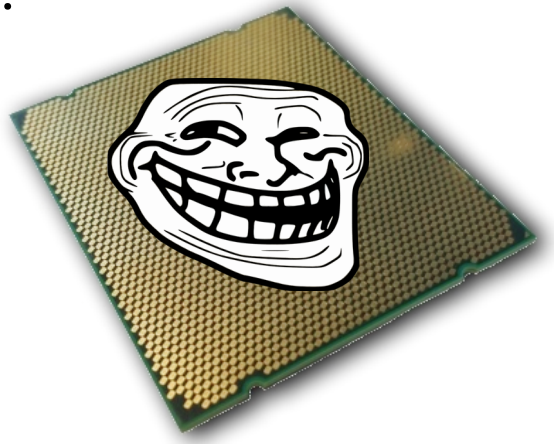
- Il arrive parfois que ce soit le **processeur** qui est limitant
  - Ne pas oublier de mesurer ce qui se passe !

Un problème de riche

- Il arrive parfois que ce soit le **processeur** qui est limitant
  - Ne pas oublier de mesurer ce qui se passe !
- Comment calculer plus vite ?
  - Augmenter la fréquence
  - Traiter plus de données à la fois
  - Traiter plusieurs tâches en parallèle
  - Traiter chaque tâche plus efficacement

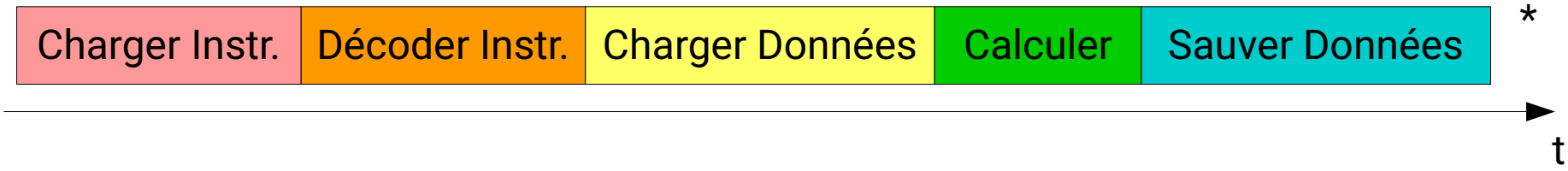
Un problème de riche

- Il arrive parfois que ce soit le **processeur** qui est limitant
  - Ne pas oublier de mesurer ce qui se passe !
- Comment calculer plus vite ?
  - ~~Augmenter la fréquence~~ (~1980-2005)
  - Traiter plus de données à la fois
  - Traiter plusieurs tâches en parallèle
  - Traiter chaque tâche plus efficacement



Le pipelining

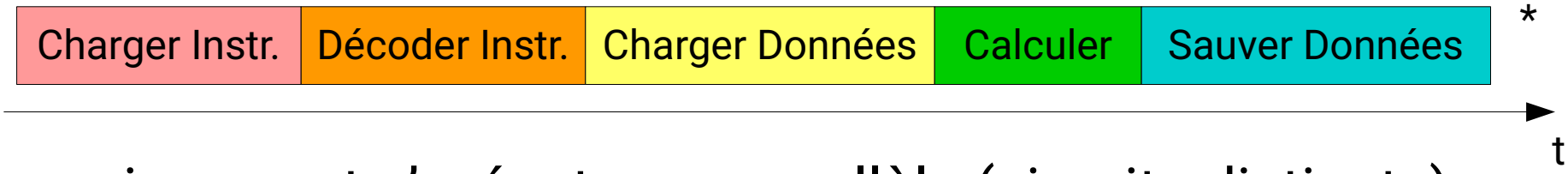
- Une instruction CPU se décompose en plusieurs parties…



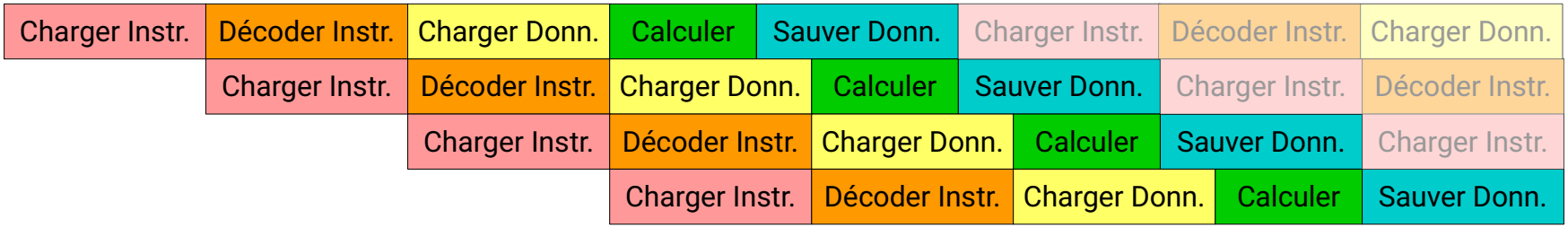
* Modèle très simplifié, une instruction d'un CPU moderne peut s'exécuter en plusieurs dizaines, voire centaines d'étapes!

Le pipelining

- Une instruction CPU se décompose en plusieurs parties…



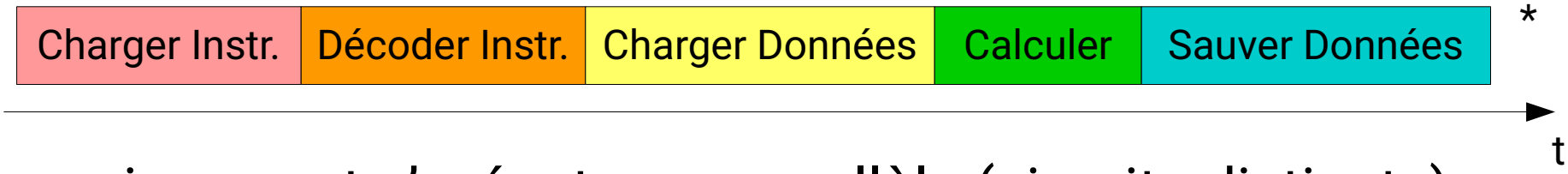
- …qui peuvent s'exécuter en parallèle (circuits distincts)



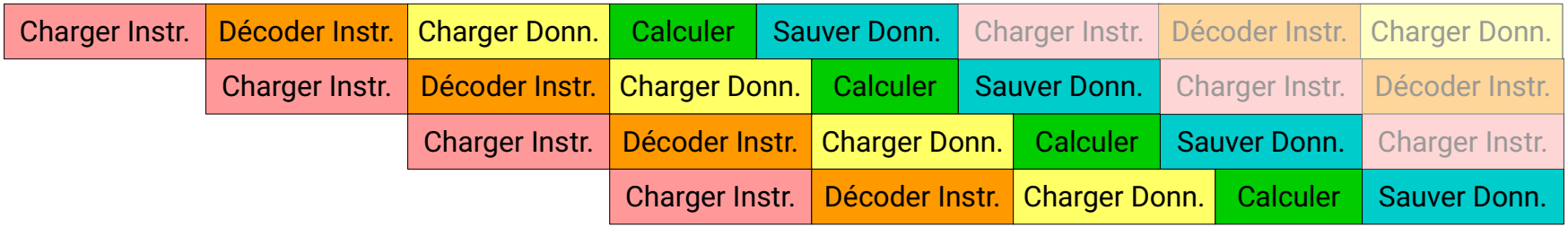
* Modèle très simplifié, une instruction d'un CPU moderne peut s'exécuter en plusieurs dizaines, voire centaines d'étapes!

Le pipelining

- Une instruction CPU se décompose en plusieurs parties…



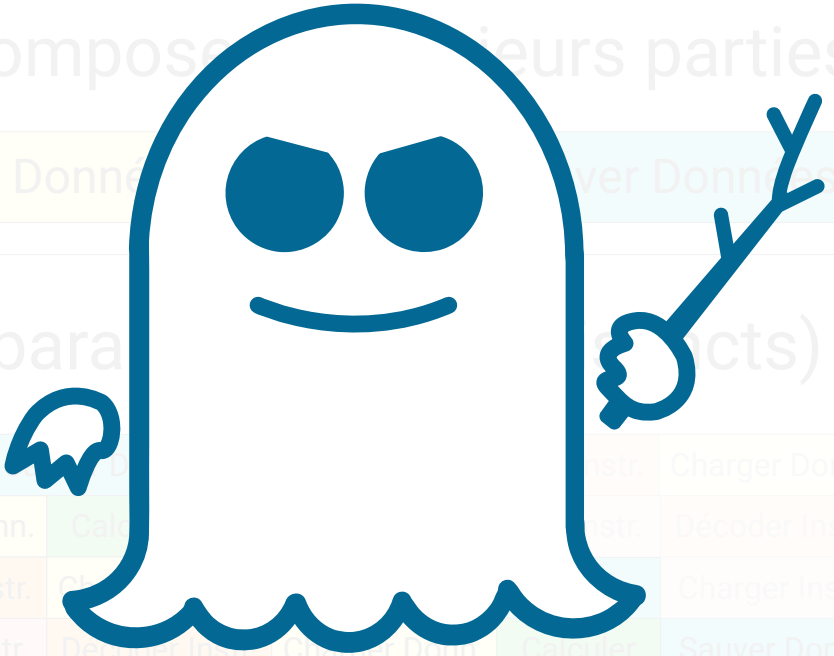
- …qui peuvent s'exécuter en parallèle (circuits distincts)



- Requiert une logique de code prévisible par le CPU
  - Il n'est pas si bête : prédiction de branche, spéculation…
  - …mais un « if » mal prédit coûte cher (parfois ~300 cycles !)

* Modèle très simplifié, une instruction d'un CPU moderne peut s'exécuter en plusieurs dizaines, voire centaines d'étapes!

Le pipelining

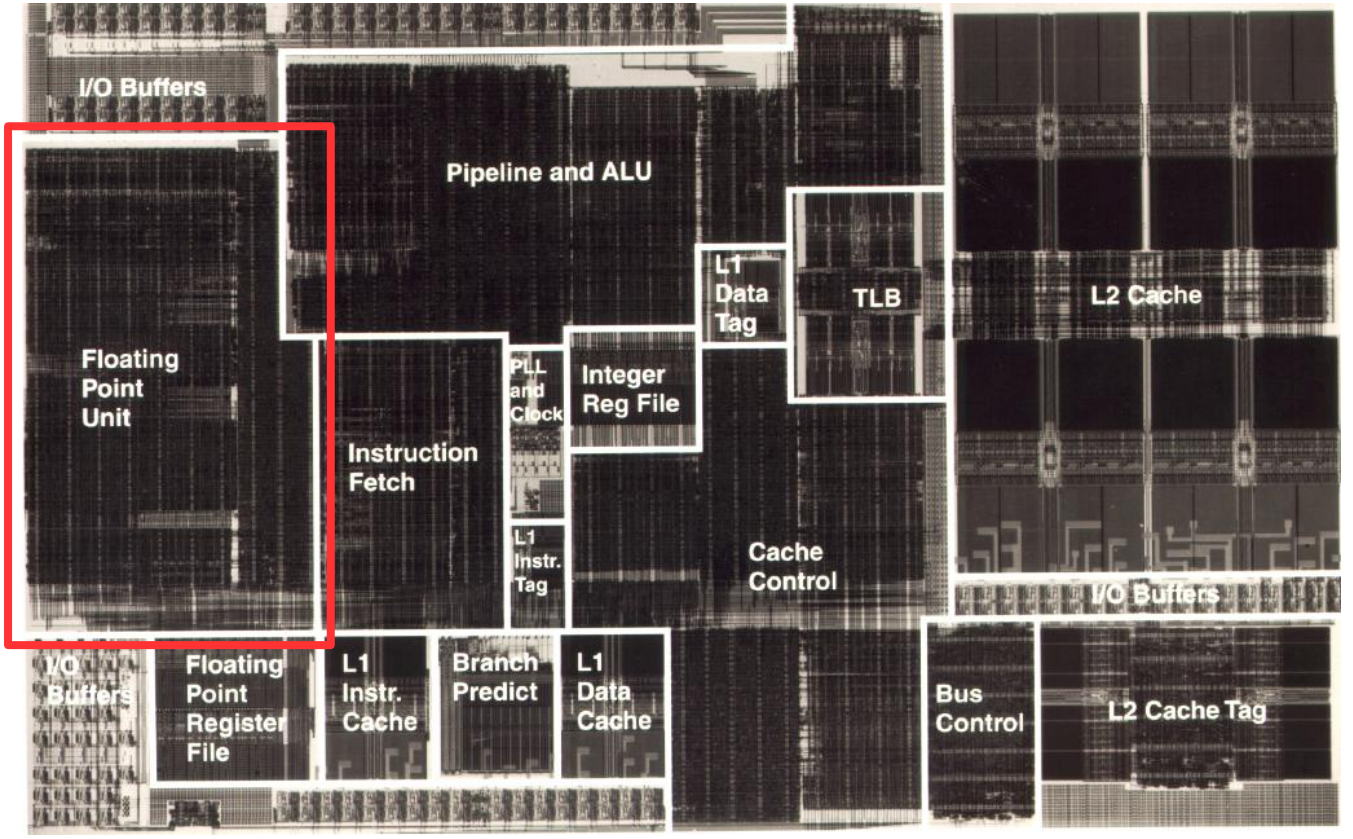


- Requiert une logique de code prévisible par le CPU
  - Il n'est **pas si bête** : prédiction de branche, spéculation…
  - …mais un « if » mal prédit coûte cher (parfois ~300 cycles !)

* Modèle très simplifié, une instruction d'un CPU moderne peut s'exécuter en plusieurs dizaines, voire centaines d'étapes!

Et le « vrai » calcul parallèle ?

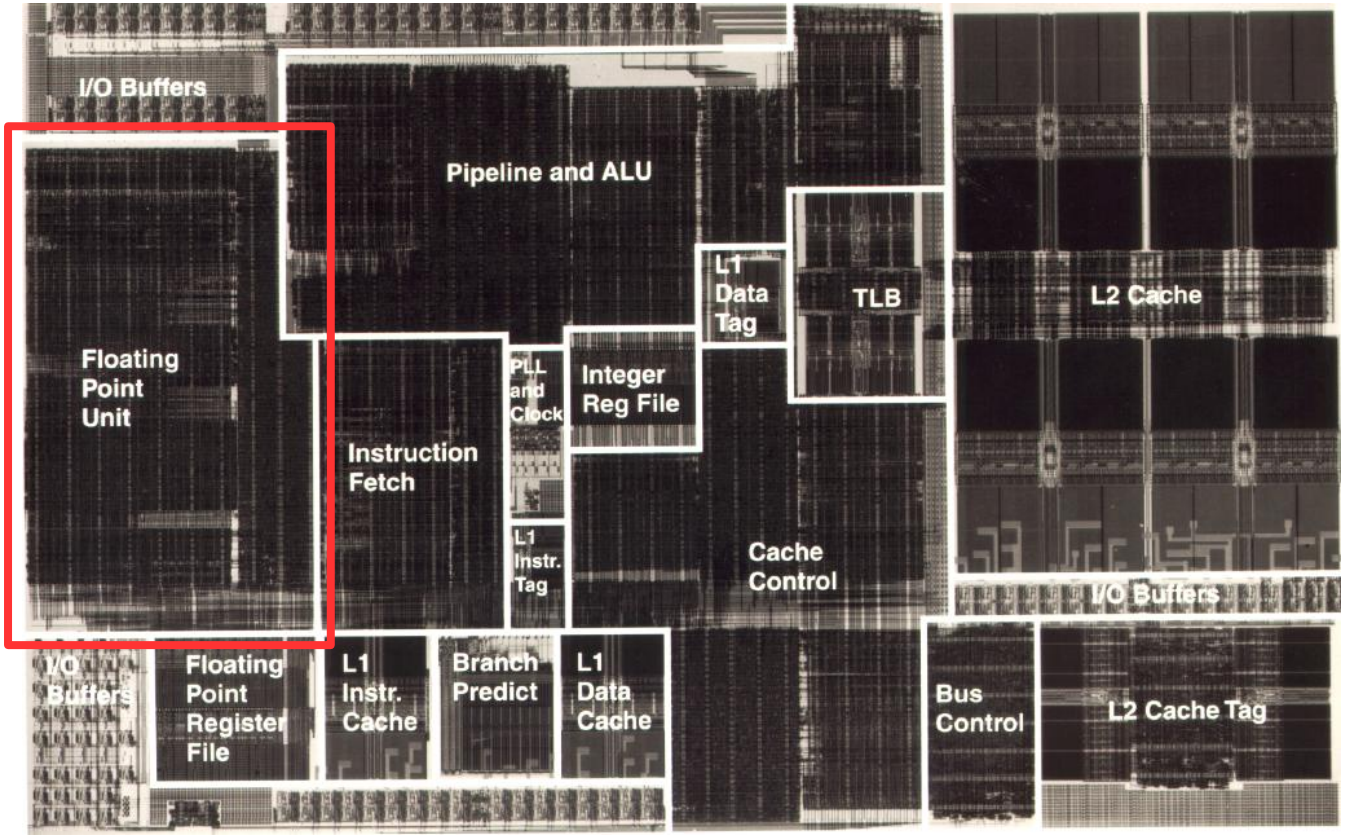
- Dans un CPU, les unités de calcul ne coûtent pas si cher…



https://en.wikichip.org/wiki/exponential_technology/x704

Et le « vrai » calcul parallèle ?

- Dans un CPU, les unités de calcul ne coûtent pas si cher…



https://en.wikichip.org/wiki/exponential_technology/x704

…on peut très bien en rajouter tout en partageant le reste

De la vectorisation au multicoeur

- Vectorisation : Une opération agit sur plusieurs données
  - Exemple : a[0..4] = b[0..4] + c[0..4] → Une instruction CPU
  - Le code machine doit utiliser explicitement ces instructions

De la vectorisation au multicoeur

- Vectorisation : Une opération agit sur plusieurs données
  - Exemple : a[0..4] = b[0..4] + c[0..4] → Une instruction CPU
  - Le code machine doit utiliser explicitement ces instructions
- Superscalaire : Plusieurs instructions **indépendantes** d'un même programme simultanément
  - Le code doit s'y prêter : (a+b) + (c+d) ≠ (((a+b)+c)+d)

De la vectorisation au multicoeur

- Vectorisation : Une opération agit sur plusieurs données
  - Exemple : a[0..4] = b[0..4] + c[0..4] → Une instruction CPU
  - Le code machine doit utiliser explicitement ces instructions
- Superscalaire : Plusieurs instructions **indépendantes** d'un même programme simultanément
  - Le code doit s'y prêter : (a+b) + (c+d) ≠ (((a+b)+c)+d)
- SMT/Hyper-threading : Remplir des « trous » du pipeline (ex : requête cache) avec le code d'une autre tâche

De la vectorisation au multicoeur

- Vectorisation : Une opération agit sur plusieurs données
  - Exemple : a[0..4] = b[0..4] + c[0..4] → Une instruction CPU
  - Le code machine doit utiliser explicitement ces instructions
- Superscalaire : Plusieurs instructions **indépendantes** d'un même programme simultanément
  - Le code doit s'y prêter : (a+b) + (c+d) ≠ (((a+b)+c)+d)
- SMT/Hyper-threading : Remplir des « trous » du pipeline (ex : requête cache) avec le code d'une autre tâche
- Multi-coeur : Exécution simultanée de plusieurs tâches

Le retour du coprocesseur

- Les CPU modernes sont très complexes
  - Fort accent sur la performance du code séquentiel
  - Gros caches, longs pipelines, prédictions, spéculations…
  - Autant de transistors perdus pour le calcul !
- C’est inutile pour des calculs simples (ex : algèbre linéaire)
  - Émergence de matériel de calcul simplifié (DSPs, GPUs…)
  - Priorités : débit mémoire, vectorisation et parallélisme
  - Nouveau problème : les communications avec le CPU

En guise de conclusion

Résumons

- Le modèle de von Neumann ne nous dit pas tout
  - Le matériel informatique moderne va **beaucoup** plus loin
- Il faut plus qu'un bon compilateur pour exploiter tout ça
  - **Cache :** Structures de données optimisées pour la localité
  - **DMA :** Interfaces non-bloquantes ou threads d'E/S dédiés
  - **Pipeline, coprocesseur :** Simplification de la logique du code
  - **Vectorisation :** Bibliothèques spécialisées (BLAS, Eigen…)
  - **Coeurs, threads :** Décomposition du code en tâches

Que peut-on faire ? (1)

- Tout d’abord, ne pas nuire
  - Préparer un bon jeu de tests avant d’optimiser
  - Gare aux pointeurs (chaînage, tableau de tableau, vtables)
  - Attention aux interfaces… (E/S bloquantes, travail inutile)
  - …ainsi qu’aux infrastructures (interpréteurs, piles de VMs, ramasse-miettes, « Global Interpreter Lock »)

Que peut-on faire ? (1)

- Tout d’abord, ne pas nuire
  - Préparer un bon jeu de tests avant d’optimiser
  - Gare aux pointeurs (chaînage, tableau de tableau, vtables)
  - Attention aux interfaces… (E/S bloquantes, travail inutile)
  - …ainsi qu’aux infrastructures (interpréteurs, piles de VMs, ramasse-miettes, « Global Interpreter Lock »)
- Penser au matériel quand on programme
  - Est-ce que j’exploite les vecteurs, les coeurs, le DMA… ?
  - Mes outils ont-ils une implémentation efficace ?
  - Et si c’est le cas, est-ce celle que je l’utilise ?

Que peut-on faire ? (2)

- Savoir à quoi on passe son temps pour se focaliser dessus
  - perf (Linux), Instruments (macOS), WPA/xPerf (Windows)
  - VTune (Intel), CodeXL (AMD), NVVP (Nvidia)
- Ecrire des programmes à plus haut niveau
  - Troquer ses boucles pour des concepts plus abstraits (algèbre linéaire, itérateurs, programmation fonctionnelle…)
  - Ils permettent davantage d'optimisations automatiques
  - Bien utilisés, ils clarifient aussi le code (et donc ses soucis)
- Pour des nombres absolus, consulter la spec matérielle

Pour finir

- La loi de Moore a autorisé un certain laisser-aller
- Le mur des GHz nous pousse à revoir nos habitudes
- Les débuts sont difficiles, mais l'expertise grandit vite :)

- Quelques références, hélas très copieuses :
  - Hennessy & Patterson, « Computer Architecture: A Quantitative Approach » (architecture matérielle)
  - www.brendangregg.com (analyse de performances Linux)

Merci de votre attention