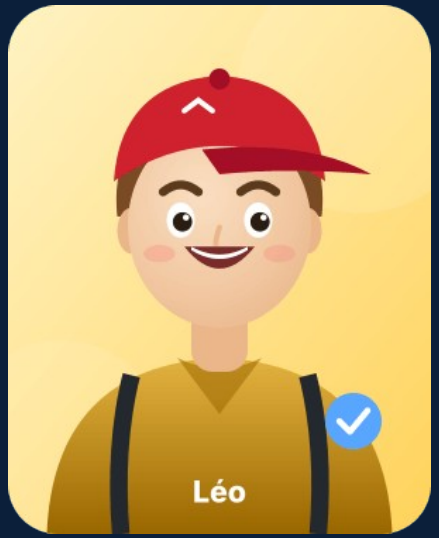
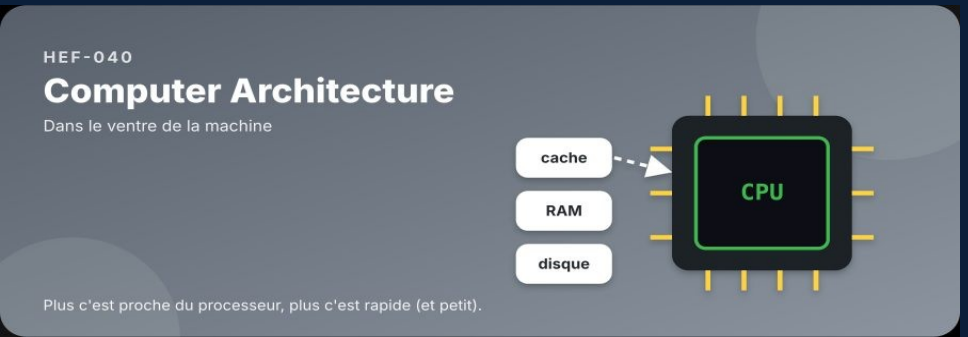
HEF-040 · MISSION 04 / 10

La hiérarchie mémoire

Plus c'est rapide, plus c'est petit et cher. D'où une pyramide de mémoires.





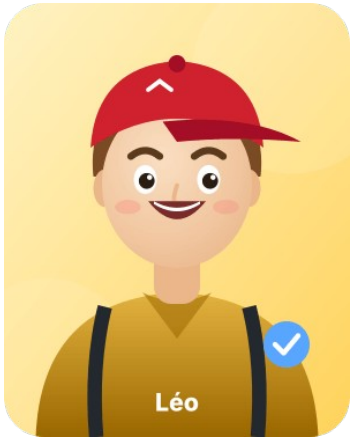
Badge : Gardien·ne des niveaux

70 min · débutant

L'histoire

Le serveur de HashBox charge 8 Go de données au démarrage. Kodi s'étonne : « Pourquoi mon disque SSD, déjà rapide, est-il des milliers de fois plus lent que la mémoire ? Et pourquoi la mémoire est-elle elle-même beaucoup plus lente que le processeur ? »

> *« Parce qu'il y a un compromis entre vitesse, taille et prix, dit Sam. On empile donc plusieurs niveaux. »*



Tu découvres la pyramide qui organise toute la mémoire d'un ordinateur.

Ta mission

À la fin, je serai capable de…

1. citer les niveaux de la hiérarchie mémoire
2. donner l'ordre de grandeur des latences de chaque niveau
3. expliquer le compromis vitesse / capacité / coût
4. convertir des latences en échelle humaine pour les comparer
5. expliquer pourquoi la mémoire est volatile ou persistante

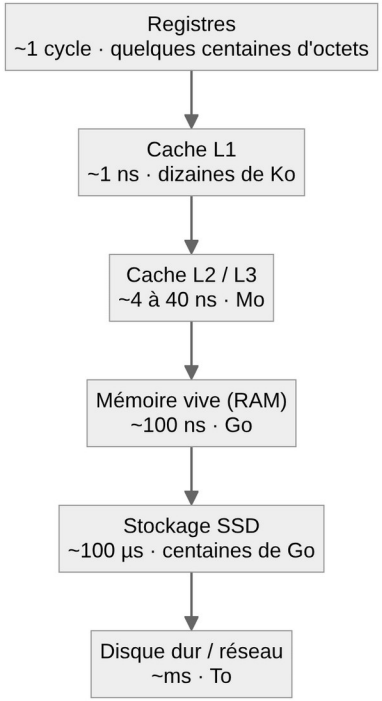
Défi d'échauffement

Défi d'échauffement

Tu cuisines. Les épices que tu utilises à chaque plat sont **dans ta main**, les ingrédients du jour **sur le plan de travail**, les provisions **dans le placard**, le stock en gros **à la cave**, et le reste **au supermarché**. Pourquoi ne mets-tu pas tout dans ta main ? Quel est le prix de chaque niveau ?

Le concept

La pyramide (suite)



Le concept

La pyramide (suite)

Description du schéma : une pyramide de six niveaux, du plus rapide et petit en haut (registres) au plus lent et grand en bas (disque dur, réseau). Chaque niveau est plus capacieux et plus lent que le précédent.

Les ordres de grandeur (approximatifs)

Niveau	Latence typique	Capacité typique	Persistant ?
Registre	< 1 ns	~1 Ko	non
Cache L1	~1 ns	32 à 64 Ko par cœur	non
Cache L2	~4 ns	256 Ko à 2 Mo par cœur	non
Cache L3	~10 à 40 ns	8 à 64 Mo partagés	non
RAM	~100 ns	8 à 128 Go	non (volatile)

Le concept

Les ordres de grandeur (approximatifs) (suite)

Niveau	Latence typique	Capacité typique	Persistant ?
SSD	~20 à 100 µs	256 Go à plusieurs To	oui
Disque dur	~5 à 10 ms	plusieurs To	oui
Réseau (même continent)	~10 à 100 ms	:	:

(1 ns = un milliardième de seconde ; 1 µs = un millionième ; 1 ms = un millième.)

Volatile ou persistant

- **Volatile** : le contenu disparaît sans courant (registres, caches, RAM).
- **Persistant** : le contenu reste éteint (SSD, disque dur).

Le concept

Pourquoi une hiérarchie ?

Les technologies rapides sont **chères** et **petites**. On combine donc de petits niveaux rapides et de grands niveaux lents, en comptant sur le fait que les programmes **réutilisent** souvent les mêmes données (localité : mission suivante).

Mettre à l'échelle humaine

Si 1 ns durait 1 seconde :

```
niveaux = [("registre / L1", 1), ("L2", 4), ("L3", 20), ("RAM", 100),
           ("SSD", 100_000), ("disque dur", 8_000_000), ("réseau", 80_000_000)]

for nom, ns in niveaux:
    secondes = ns          # 1 ns -> 1 s
    if secondes < 60:
```

Le concept

Mettre à l'échelle humaine (suite)

```
        txt = f"{secondes} s"
    elif secondes < 3600 * 24:
        txt = f"{secondes / 3600:.1f} heures"
    else:
        txt = f"{secondes / 86400:.0f} jours"
    print(f"{nom:14s} {txt}")
```

🧠 ***Astuce*** *: si le cache était une seconde, la RAM serait une minute et demie, le SSD plus d'une journée, le disque dur près de 3 mois.*

Exemple pas à pas

Problème : combien de temps pour lire 1 million d'entiers si chaque lecture doit aller chercher la donnée à un niveau différent ?

```
lectures = 1_000_000
latences_ns = {"cache L1": 1, "RAM": 100, "SSD": 100_000, "disque dur": 8_000_000}

for nom, ns in latences_ns.items():
    total_s = lectures * ns / 1e9
    print(f"{nom:11s} {total_s:12.3f} s")
# cache L1        0.001 s
# RAM             0.100 s
# SSD           100.000 s
# disque dur   8000.000 s
```

Lecture : le même million de lectures prend 1 ms depuis le cache, 0,1 s en RAM, plus d'une minute et demie sur SSD si chaque lecture est un accès aléatoire isolé, et plus de deux heures sur disque dur. (En pratique, les lectures séquentielles sont groupées et beaucoup plus rapides : on y revient en mission 6.)

Exemple pas à pas (suite)

Leçon : la performance d'un programme dépend moins de la vitesse du CPU que de **l'endroit où se trouvent les données**.

À toi, avec aide

À toi, avec aide

Complète le tableau. Pour chaque situation, dis dans quel niveau se trouve probablement la donnée et justifie :

Situation	Niveau
La variable `i` d'une boucle `for` qui vient d'être modifiée	?
Un tableau de 100 Mo que le programme parcourt	?
Le fichier de configuration lu au démarrage	?
La page web que tu viens de charger	?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Une valeur qui change à chaque tour est souvent gardée dans un registre.

Indice 2

100 Mo ne tient pas dans un cache L1 de 32 Ko.

Indice 3

Un fichier est persistant : il vient du SSD ou du disque.

À toi, avec aide · réponse

Réponse

Situation	Niveau
Variable `i` d'une boucle	**Registre** (ou cache L1) : valeur utilisée en continu
Tableau de 100 Mo	**RAM**, avec des morceaux qui transitent par les caches L3/L2/L1 pendant le parcours
Fichier de configuration lu au démarrage	**SSD / disque** à la première lecture, puis **RAM** (cache du système d'exploitation)
Page web chargée	**Réseau** pour la première fois, puis **RAM** (et éventuellement cache disque du navigateur)

À toi, en solo

Écris `temps_total(acces)` qui calcule le temps d'un programme à partir du nombre d'accès à chaque niveau, par exemple `{"L1": 9_000_000, "RAM": 900_000, "SSD": 100}`.

Puis réponds avec le programme : si 99 % des accès sont servis par le cache L1 et 1 % par la RAM, quelle est la latence **moyenne** par accès ? Et si le taux de succès du cache tombe à 90 % ?

À toi, en solo · réponse

Pistes de correction

```
LATENCES_NS = {"L1": 1, "L2": 4, "L3": 20, "RAM": 100, "SSD": 100_000, "HDD": 8_00
   0_000}

def temps_total(acces):
    return sum(LATENCES_NS[niveau] * n for niveau, n in acces.items()) / 1e9

def latence_moyenne(taux_succes_l1):
    return taux_succes_l1 * LATENCES_NS["L1"] + (1 - taux_succes_l1) * LATENCES_NS
   ["RAM"]

print(temps_total({"L1": 9_000_000, "RAM": 900_000, "SSD": 100}), "s")
print(latence_moyenne(0.99), "ns")   # 1.99 ns
print(latence_moyenne(0.90), "ns")   # 10.9 ns
```

Passer de 99 % à 90 % de succès multiplie la latence moyenne par plus de 5. Un petit déficit de cache peut donc ralentir énormément un programme.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Penser que la RAM est « rapide »	Elle est ~100 fois plus lente que le cache	Compter les latences en nanosecondes
Confondre volatile et persistant	Données perdues à l'extinction	Sauvegarder sur un support persistant
Ignorer l'ordre de grandeur	On optimise la mauvaise chose	Mémoriser : ns, µs, ms
Croire que plus de RAM règle tout	Le goulot est peut-être ailleurs	Mesurer avant d'investir

Mini-boss

Mini-boss

Budget de latence. Une page web de HashBox fait, pour s'afficher : 200 000 accès au cache L1, 20 000 accès à la RAM, 30 lectures SSD, 3 requêtes réseau vers une base distante (30 ms chacune).

1. Calcule le temps passé à chaque niveau, avec un programme.
2. Quel niveau domine ? Quelle est la part de chaque niveau dans le total ?
3. Propose deux améliorations (par exemple regrouper les requêtes réseau, mettre en cache des lectures) et calcule le gain.
4. Applique la loi d'Amdahl : si on accélère 10 fois les accès cache, que gagne-t-on au total ?

Mini-boss · réponse

Pistes de correction

```
L = {"L1": 1, "RAM": 100, "SSD": 100_000, "reseau": 30_000_000}
acces = {"L1": 200_000, "RAM": 20_000, "SSD": 30, "reseau": 3}
temps = {k: acces[k] * L[k] / 1e9 for k in acces}
total = sum(temps.values())
for k, t in temps.items():
    print(f"{k:7s} {t * 1000:9.3f} ms  {100 * t / total:5.1f} %")

# amélioration : 1 seule requête réseau, SSD servi par la RAM
acces2 = {"L1": 200_000, "RAM": 20_030, "SSD": 0, "reseau": 1}
total2 = sum(acces2[k] * L[k] / 1e9 for k in acces2)
print(round(total * 1000, 1), "ms ->", round(total2 * 1000, 1), "ms")

# Amdahl : cache 10x plus rapide
p = temps["L1"] / total
print("gain global :", round(1 / ((1 - p) + p / 10), 3), "x")
```

Le réseau domine (environ 95 % du temps). Accélérer le cache n'apporte presque rien : la bonne cible est de réduire les allers-retours réseau.

Quiz éclair

Quiz éclair

1. Le niveau le plus rapide est :

A. les registres
B. la RAM
C. le SSD

2. La RAM est :

A. persistante
B. volatile
C. aussi rapide que le cache

3. Plus un niveau de mémoire est rapide, plus il est généralement :

A. grand et bon marché

Quiz éclair

Quiz éclair (suite)

B. lent
C. petit et cher

4. 🔁 *Révision mission 03* : Le registre PC contient :

A. l'adresse de la prochaine instruction
B. le résultat du dernier calcul
C. la taille de la mémoire

5. 🔁 *Révision mission 02* : Un ISA est :

A. un type de mémoire
B. la liste des instructions qu'un processeur comprend
C. un langage de haut niveau

Quiz éclair · réponse

Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

Je réfléchis

Je réfléchis

1. Quel équivalent de la hiérarchie mémoire existe dans ta façon de ranger tes affaires ou tes documents ?
2. Pourquoi connaître les ordres de grandeur évite-t-il des erreurs de conception ?

À retenir

À retenir

1. Plus rapide = plus petit et plus cher.
2. Registres, L1, L2, L3, RAM, SSD, disque, réseau.
3. Volatile (registres, cache, RAM) / persistant (SSD, disque).
4. Les écarts de latence se mesurent en facteurs de 10 à 100 000.
5. La performance dépend de l'endroit où sont les données.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Gardien·ne des niveaux

Prochaine mission : Cache et localité

