Optimisation logicielle & matérielle

L'informatique face à l'écoconception

Loïc Rouquette
EPITA - Laboratoire de Recherche de l'EPITA (LRE)
26 octobre 2026

Optimisation logicielle & matérielle © 2026 Loïc Rouquette is licensed under CC BY 4.0.
To view a copy of this license, visit https ://creativecommons.org/licenses/by/4.0/.

Table des matières

01 Introduction

02 Optimiser le logicielHack, HVVM & l'énergie des langages

03 Optimiser le runtimeLe projet Valhalla (OpenJDK/Oracle)

04 Optimiser le matérielFPGA & ASIC dans les clouds

05 Exploiter autrementIA Générative, PUE & carbon-aware computing

06 Circularité & SobriétéSeconde vie du matériel et des données

07 Synthèse

licensed under CC BY 4.0

01 Introduction

La fin du progrès « gratuit »

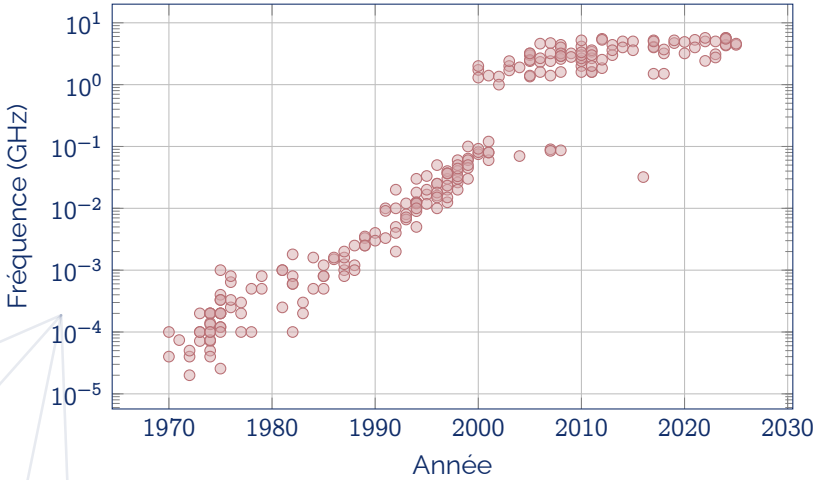


Figure 1 : Puissance des microprocesseurs de 1970 à nos jours Wikipedia contributors, 2026a.

Fournir de nouveaux services à puissance de calcul équivalent.

licensed under CC BY 4.0

Quelques chiffres clés ADEME, 2023

- Empreinte carbone du numérique **29,5 Mt CO_2 éq.** soit **4,4%** de l'empreinte carbone nationale ;

licensed under CC BY 4.0

Quelques chiffres clés ADEME, 2023

- Empreinte carbone du numérique **29,5 Mt CO_2 éq.** soit **4,4%** de l'empreinte carbone nationale ;
- Consommation électrique : **51,5 TWh**, soit **11 %** de la consommation nationale (chiffre incluant les datacenters étrangers nécessaires aux usages français) ;

licensed under CC BY 4.0

Quelques chiffres clés ADEME, 2023

- Empreinte carbone du numérique **29,5 Mt CO_2 éq.** soit **4,4%** de l'empreinte carbone nationale ;
- Consommation électrique : **51,5 TWh**, soit **11 %** de la consommation nationale (chiffre incluant les datacenters étrangers nécessaires aux usages français) ;
- Répartition de l'impact : terminaux (fabrication) dominent, puis les datacenters et le réseaux. L'**écoconception** généralisée pourrait réduire la consommation des ressources de **15 %**.

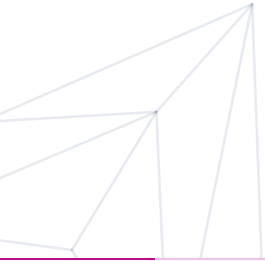
licensed under CC BY 4.0

02

Optimiser le logiciel Hack, HVVM & l'énergie des langages

Contexte

- Facebook naît en **PHP** en 2004. : langage interprété, pensé pour la rapidité de développement, pas la performance brute.



licensed under CC BY 4.0

- Facebook naît en **PHP** en 2004. : langage interprété, pensé pour la rapidité de développement, pas la performance brute.
- L'entreprise passe de **0 utilisateurs** en 2004 à **+100M utilisateurs** en 2008 (Wilberding et Wells, 2019).

À l'échelle : centaines de milliards de requêtes/mois — l'interprétation classique ne tient plus.

licensed under CC BY 4.0

Trajectoire : de l'interprétation à la compilation

Année	Outil	Idée
2004	PHP	Facilité de développement

licensed under CC BY 4.0

Trajectoire : de l'interprétation à la compilation

Année	Outil	Idée
2004	PHP	Facilité de développement
2010	HipHop	Compilateur PHP → C++ : le code est traduit une fois et exécuté plus vite

licensed under CC BY 4.0

Trajectoire : de l'interprétation à la compilation

Année	Outil	Idée
2004	PHP	Facilité de développement
2010	HipHop	Compilateur PHP → C++ : le code est traduit une fois et exécuté plus vite
2011	HHVM	Machine virtuelle avec **JIT** : compilation à la volée du bytecode HHBC en code machine

licensed under CC BY 4.0

Trajectoire : de l'interprétation à la compilation

Année	Outil	Idée
2004	PHP	Facilité de développement
2010	HipHop	Compilateur PHP → C++ : le code est traduit une fois et exécuté plus vite
2011	HHVM	Machine virtuelle avec **JIT** : compilation à la volée du bytecode HHBC en code machine
2014	Hack	Nouveau langage à typage graduel, conçu au-dessus de HHVM

licensed under CC BY 4.0

HHVM + Hack : les leviers techniques

HHVM

Traduction du bytecode HHBC en **code machine** pour les chemins « chauds ».
Spécialisation par type d'arguments : des requêtes typées ⇒ du code machine quasi statique.

Hack

Annotations de types vérifiées par le **typechecker**.
Erreurs attrapées **avant** l'exécution, code plus prévisible.
Prévisibilité ⇒ le JIT peut optimiser plus agressivement.

licensed under CC BY 4.0

Comparaison (Codex, 2023)

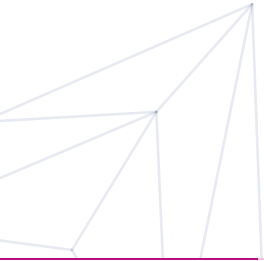
```
# Traditional PHP 7.4
php benchmark.php
# Fibonacci(35) = 9227465
# Execution time: 2.3491 seconds
# Memory usage: 2.00 MB

# HHVM
hhvm benchmark.php
# Fibonacci(35) = 9227465
# Execution time: 0.8734 seconds
# Memory usage: 4.00 MB
```


licensed under CC BY 4.0

Gains

Réduction **massive** du nombre de serveurs pour un même trafic.
Ordre de grandeur : nombre de machines divisé par **2** à **10** selon les services.



licensed under CC BY 4.0

Gains

Réduction **massive** du nombre de serveurs pour un même trafic.
Ordre de grandeur : nombre de machines divisé par **2** à **10** selon les services.

Note : Arrivée du JIT dans PHP 8.0

licensed under CC BY 4.0

L'énergie dépend du langage

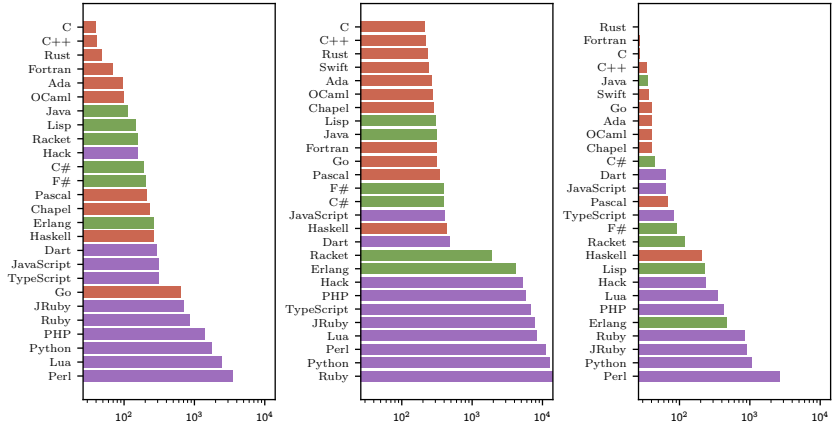


Figure 2 : Consommation énergétique par langage et type d'exécution PEREIRA et al., 2017.

03

Optimiser le runtime
Le projet Valhalla
(OpenJDK/Oracle)

Rappels (OpenJDK, 2026)

- En Java, tout objet a une **identité** (en-tête, référence, verrouillage possible).
- Problème : comment sont stockés les tableaux d'objets ?

licensed under CC BY 4.0

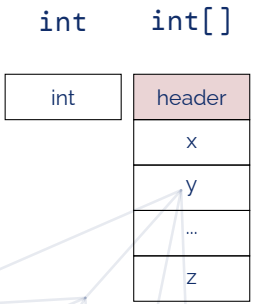
int

`int` `int[]`

int

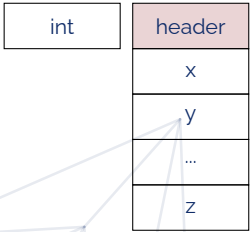


int



int

`int` `int[]`



Point

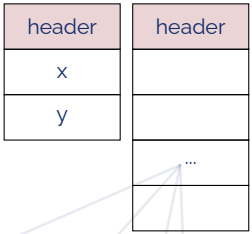
`Point` `Point[]`

header
x
y



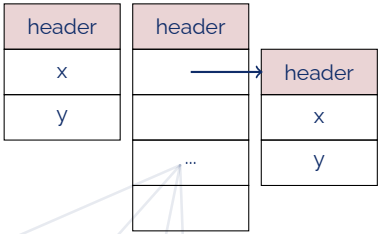
Point

Point Point[]



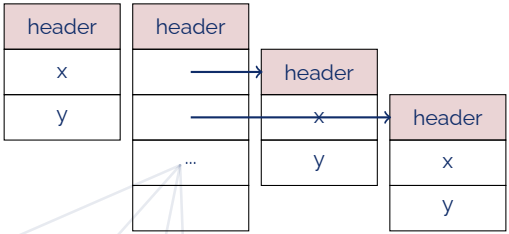
Point

`Point` `Point[]`



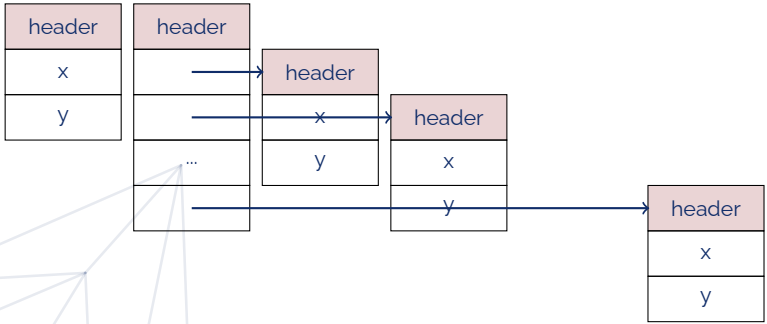
Point

Point Point[]



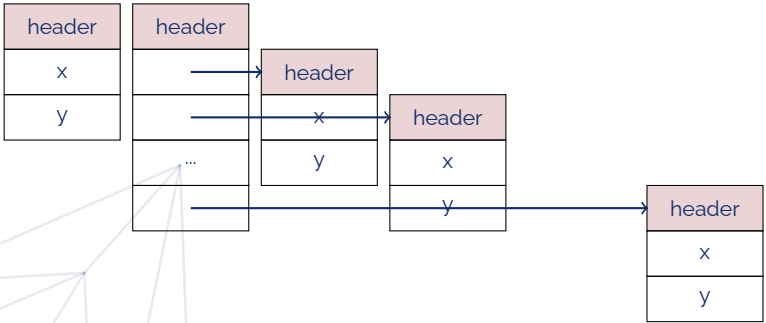
Point

Point Point[]



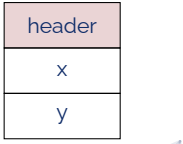
Point

`Point` `Point[]`



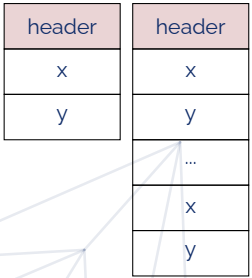
`Point` (with Valhalla)

`Point` `Point[]`



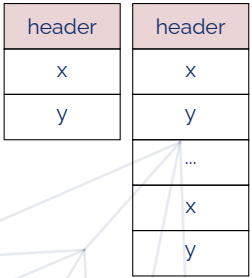
`Point` (with Valhalla)

`Point` `Point[]`



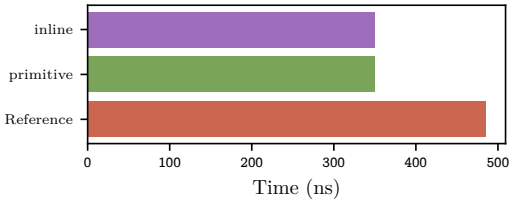
`Point` (with Valhalla)

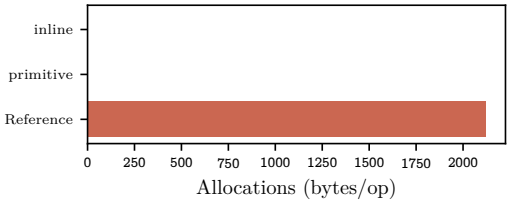
`Point` `Point[]`



Local variable (Kuksenko, 2020)

```
int count(Complex c) {
    Complex z = c;
    for (int i = 1; i < MAX_ITERATION; i++) {
        if (z.modulus() >= 4.0) return i;
        z = z.square().add(c);
    }
    return MAX_ITERATION;
}
```





Mandelbrot

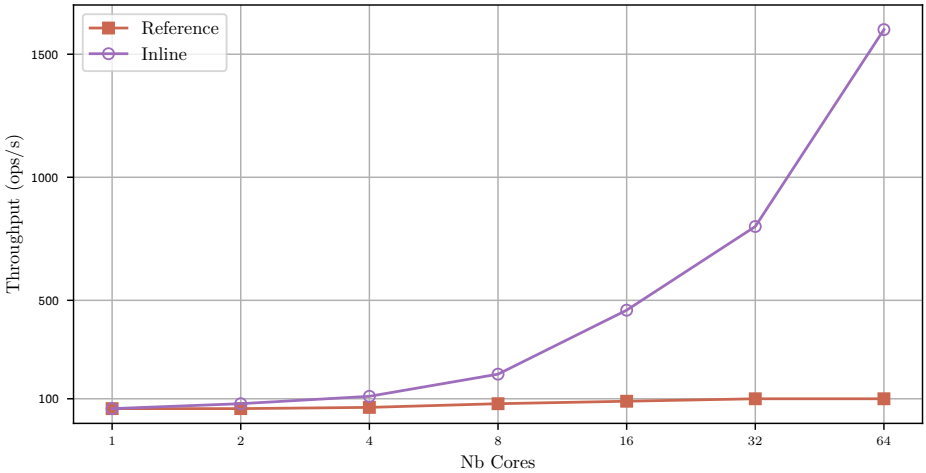


Figure 3 : Débit (en ops/s) en fonction du nombre de cœurs (KUKSENKO, 2020).

licensed under CC BY 4.0

Constat

- Moins d'allocation $\Longrightarrow$ moins de GC $\Longrightarrow$ moins de CPU $\Longrightarrow$ moins de serveur pour la même charge ;

licensed under CC BY 4.0

Constat

- Moins d'allocation $\Longrightarrow$ moins de GC $\Longrightarrow$ moins de CPU $\Longrightarrow$ moins de serveur pour la même charge ;
- Moins de cache misses $\Longrightarrow$ moins de cycles $\Longrightarrow$ moins de watt.

licensed under CC BY 4.0

04

Optimiser le matériel FPGA & ASIC dans les clouds

Le double mur du CPU générique

- Loi de Moore au ralenti ; fin de la loi de Dennard : le CPU générique ne gagne persque plus de performance ;



licensed under CC BY 4.0

Le double mur du CPU générique

- Loi de Moore au ralenti ; fin de la loi de Dennard : le CPU générique ne gagne persque plus de performance ;
- Énorme dépense d'énergie en **flexibilité** : décodage, spéculations, caches, *etc.* ;

licensed under CC BY 4.0

Le double mur du CPU générique

- Loi de Moore au ralenti ; fin de la loi de Dennard : le CPU générique ne gagne persque plus de performance ;
- Énorme dépense d'énergie en **flexibilité** : décodage, spéculations, caches, *etc.* ;
- La **spécialisation matérielle** est la réponse de l'industrie à ce double mur.

licensed under CC BY 4.0

Le Continuum de Spécialisation

	Perf/W	Fonctionnement	Rigidité/Coût
CPU	O	Dynamique	$

licensed under CC BY 4.0

Le Continuum de Spécialisation

	Perf/W	Fonctionnement	Rigidité/Coût
CPU	O	Dynamique	$
GPU	OO	Parallèle Massif	$ $

licensed under CC BY 4.0

Le Continuum de Spécialisation

	Perf/W	Fonctionnement	Rigidité/Coût
CPU	O	Dynamique	$
GPU	OO	Parallèle Massif	$ $
FPGA	O O O	Reprogrammable	$ $ $

licensed under CC BY 4.0

Le Continuum de Spécialisation

	Perf/W	Fonctionnement	Rigidité/Coût
CPU	O	Dynamique	$
GPU	OO	Parallèle Massif	$ $
FPGA	O O O	Reprogrammable	$ $ $
ASIC	O O O O OO	Câblé dans la puce	$ $ $ $ $ $ $ $

licensed under CC BY 4.0

Le TPU : archetype de l'ASIC

Tensoir Processing Unit : TPU

- ASIC conçu pour l'**inférence** puis l'**entraînement** ;
- **matrice de multiplcateurs-accumulateurs** (*systolic array*) ; cœur dédié aux produits matriciels ;
- **quantification** `int8` : calcul en précision réduite → moins d'énergie, plus de débit ;
- **abandon de la flexivilité généraliste**.

	Date de sortie	Mem. Bandwidth (GB/s)	Performance (10^9 ops/s)	Efficiency (10^9 ops/W)
v1	2015	34	23	0.31
v2	2017	660	45	0.16
v3	2018	900	123	0.56
v4	2021	1200	275	1.62
v5e	2023	819	393 (*int8*)	N/A
v5p	2023	2765	918 (*int8*)	N/A
v6e	2024	1640	1836 (*int8*)	N/A
v7	2025	7370	4614 (*fp8*)	4.7
v8t/v8i	2026	6500/8500	12600 (*fp4*) / 10100 (*fp4*)	N/A

Table 1 : (Wikipedia contributors, 2026b)

licensed under CC BY 4.0

Le TPU : archetype de l'ASIC

Résultats

- **Gros gains de perf/W vs CPU/GPU** pour les charges IA ;
- Alimente l'**entraînement et l'inférence massifs** - dont l'**IA générative de Google** ;
- **Usage interne** (datacenters Google) + **Commercialisation via Cloud**.

licensed under CC BY 4.0

Azure Accelerated Networking (Firestone et al., 2018; Putnam et al., 2014)

- Implémentation de la stack réseau par FPGA *SmartNIC/FGPA as a NIC*[1] ;
- **Reconfigurable après fabrication** : moins de risque d'**obsolescence** qu'un ASIC ;
- **Plus efficace qu'un CPU, moins qu'un ASIC** : positionnement au milieu du continuum ;
- Idéal quand la **charge évolue vite** ou que les fonctionnalités peuvent évoluer.

[1] NIC : *Network Interface Card*

licensed under CC BY 4.0

Amazon : le chaîne complète

Nitro

- Idem à Azure Accelerated Network : Décharge de virtualisation / réseau / stockage du CPU vers des **puces dédiées** ;
- Le CPU générique cesse de dépenser son énergie à gérer l'infrastructure $\Longrightarrow$ plus de cycles au profit des clients.

Graviton

- CPU Arm sur mesure, conçu par AWS pour ses propres charges ;
- Gains de l'orgde de **20-40 %** de perf/€ selon les charges.

Inferentia / Trainium

- Idem que les TPUs de Google.

licensed under CC BY 4.0

Nuance d'écoconception

- Un **ASIC** se justifie uniquement à **très (très) grande échelle** ;
  - Coût de conception / fabrication (*Non-recurring engineering*) ;
  - Risque d'**obsolescence**.

licensed under CC BY 4.0

05

Exploiter autrement IA Générative, PUE & carbon-aware computing

Green AI ? Le coût énergétique de l'IA

	Entraînement	Inférence
Intensité	Très élevée	Légère
Fréquence	Ponctuelle	Des milliards de requêtes
Horizon	Une fois par modèle	Toute la durée de vie

- L'inférence est **souvent le poste dominant** sur la durée de vie du modèle
- Un coût faible × un volume massif = un total considérable

licensed under CC BY 4.0

Les leviers d'optimisation

- **Quantification** (int8 / int4) : division de l'énergie consommée par inférence
- **Batching des requêtes** : mutualiser les calculs entre requêtes simultanées
- **Modèles plus petits et spécialisés** : distillation vers des modèles ciblés
- **Cache de résultats** : éviter de recalculer des inférences identiques
- **Dimensionnement du modèle à la tâche réelle** : ne pas sur-qualifier le besoin

licensed under CC BY 4.0

Local vs Cloud

Edge (téléphone, caméra, *etc.*)

- Économise le transport réseau ;
- Permet d'éteindre quand inutile ;
- Matériel moins efficace qu'un ASIC de datacenter ;
- Gestion plus simples des données sensibles ;
- Multiplication des puces.

Datacenter

- ASIC très efficaces (TPU, Inferentia) ;
- Mutualisation et optimisation à grande échelle ;
- Gestion externe des données sensibles.

La réponse est **dépendante du cas d'usage** : volume de requêtes, latence requise, contraintes réseau.

licensed under CC BY 4.0

PUE - Power Usage Effectiveness

PUE

$$PUE = \frac{\text{Énergie totale du site}}{\text{Énergie utile IT}}$$

$PUE = 1$: **limite théorique**, toute l'énergie va à l'informatique.
Hyperscalers : $PUE \approx 1.1$ pour les meilleurs sites.
Salles on-premise classiques : bien au délà.

licensed under CC BY 4.0

L'eau : la dimension oubliée

- Le **refroidissement évaporatoire** consommme de l'eau potable à grande échelle ;
- L'empreinte d'un datacenter **n'est pas que carbone** ;
- À croiser avec les contraintes locales : stress hydrique des régions d'implantation.

licensed under CC BY 4.0

Carbon-aware computing (Radovanović, 2020)

Principe : déplacer les charges dlexibles **dans le temps et l'espace**

- Charges concernées : batch, entraînement, indexation
- Objectif : viser les créneaux et régions où l'électricité est **la plus bas-carbone**
- Exemple emblématique : la plateforme **carbon-aware** de Google qui planifie certaines charges selon le mix électrique horaire.

licensed under CC BY 4.0

06

Circularité & Sobriété
Seconde vie du matériel et des données

Où est l'empreinte d'un serveur ?

60 - 80 % de l'empreinte se situe dans la fabrication :

- Métaux critiques
- Eau
- Énergie de production

Le serveur le plus sobre est souvent celui qu'on n'achète pas.

licensed under CC BY 4.0

Prolonger la durée de vie

Les hyperscalers sont passés d'un cycle de ≈ **3 ans** à **5-6 ans** pour leurs serveurs.
Chaque année de réutilisation évite une nouvelle fabrication – et donc 60-80 % d'empreinte amont.

licensed under CC BY 4.0

Est-ce toujours bon ?

Données : un serveur neuf coûte $C_b \approx 1$ à 3 tCO_2eq en fabrication ; il consomme en charge P_n watts et l'ancien P_o watts.

$$T_{\text{breakeven}} = \frac{C_b}{\Delta_P \times CI_{\text{grid}}}$$

avec CI_{grid} l'intensité carbone du réseau.

- 43.48 tCO_2eq pour la France
- 296.02 tCO_2eq pour l'Allemagne
- 338.91 tCO_2eq pour les États-Unis

(Low-Carbon Power, 2026)

licensed under CC BY 4.0

Reconditionné, Cascade d'usage & Retrofit

- **Marché du reconditionné, cascade d'usage :** donner une seconde vie à des serveurs sortis de production (*e.g.* prod, pre-prod, dev, stockage)
- **Rétrofit :** moderniser l'existant (composants, mémoire) plutôt que de tout remplacer.

licensed under CC BY 4.0

Le logiciel fait vieillir le matériel

Le matériel n'est pas le seul à vieillir : le **software bloat** rend les « vieilles » machines inutilisables :

- poids des pages web $\times 10$ en 10 ans ($\times 180$ en 30 ans) ;
- applications mobiles obèses ;
- OS de plus en plus exigeants.

licensed under CC BY 4.0

Les leviers

- **OS légers :** allégements Linux pour redonner vie aux vieilles machiens ;
- **ROM alternatives et support logiciel long (LTS)** : *e.g.* écosystème Fairphone / LineageOS côté mobiel ;
- **Logiciels compatibles avec le matériel ancien** : code open source ;
- **Éviter les dépendances superflues :** chaque dépendance a un coût, en ressources comme en maitenance.

licensed under CC BY 4.0

Dark data : stocker sans jamais relire

Une grande part des donénes stockées **n'est jamais relue**.
Mais chaque **sauvegarde, réplication** et **transfert** la déplace – et la **refacture en énergie**.
Politique de données :

- Définir une **durée de vie** des données ;
- **Archivage froid** plutôt que stockage actif permanent ;
- **Suppression effective** en fin de vie.

licensed under CC BY 4.0

Réseau alléger ce qui transite

- **Compression :** Brotli, formats binaires (protobuf/CBOR) vs JSON ;
- **Dimensionnement des images :** la bonne taille, pas la taille maximale ;
- **Qualité vidéo adaptée** au besoin réel.

Pour rappel les images représents **60 - 80 %** du poids d'une page web ; Les vidéos représentent **80 %** du traffic internet mondial.

licensed under CC BY 4.0

07

Synthèse

Les grands arbitrages

- **Usage vs Fabrication :** le seuil de rentabilité contrebalance l'attrait du matériel neuf « plus efficace » ;
- **Paradoxe de Jevons :** les gains d'efficacité sont souvent réinvestis dans plus de consommation (effet rebond) – l'IA générative en est l'exemple actuel ;
- **Efficacite ≠ Sobriété :** optimiser un service inutile reste du gaspillage. Ordre des leviers : remettre en cause l'usage (sobriété), puis écoconcevoir, puis optimiser.

licensed under CC BY 4.0

Le boîte à outils de l'ingénieur en 2026

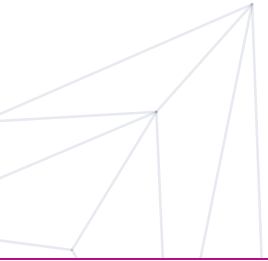
- **Mesurer avant d'optimiser :** profilage (perf, flamegraphs), compteurs RAPL/énergie, Scaphandre pour la mesure par process ;
- **Outil d'écoconception logicielle :** référentiel **RGESN** (GreenIT), analyseurs statisque de types EcoCode/CodeCarbon, Cloud Carbon Footprint, méthodologie **Software Carbon Intensity** de la Green Software Foundation ;
- **Leviers concerts pour les développeurs :** choix de langage et typage, structures de données cache-friendly, éviter la dark data, dimensionner les ressources.

licensed under CC BY 4.0

Sobriété fonctionnelle et architecture centrée sur l'usage

La question qui précède toutes les autres : Cette fonctionnalité a-t-elle un réel intérêt pour les utilisateurs ?

- Un chatbot IA pour une FAQ de 10 questions ?

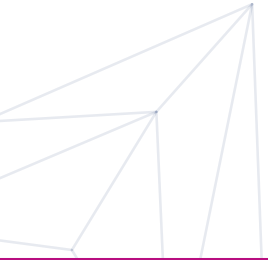


licensed under CC BY 4.0

Sobriété fonctionnelle et architecture centrée sur l'usage

La question qui précède toutes les autres : Cette fonctionnalité a-t-elle un réel intérêt pour les utilisateurs ?

- Un chatbot IA pour une FAQ de 10 questions ?
- Des notifications push redondantes ?



licensed under CC BY 4.0

Sobriété fonctionnelle et architecture centrée sur l'usage

La question qui précède toutes les autres : Cette fonctionnalité a-t-elle un réel intérêt pour les utilisateurs ?

- Un chatbot IA pour une FAQ de 10 questions ?
- Des notifications push redondantes ?
- La vidéo en autplay ?



licensed under CC BY 4.0

Sobriété fonctionnelle et architecture centrée sur l'usage

La question qui précède toutes les autres : Cette fonctionnalité a-t-elle un réel intérêt pour les utilisateurs ?

- Un chatbot IA pour une FAQ de 10 questions ?
- Des notifications push redondantes ?
- La vidéo en autplay ?

Architecture centrée sur l'usage : dimensionner au besoin réel.

licensed under CC BY 4.0

Sobriété fonctionnelle et architecture centrée sur l'usage

La question qui précède toutes les autres : Cette fonctionnalité a-t-elle un réel intérêt pour les utilisateurs ?

- Un chatbot IA pour une FAQ de 10 questions ?
- Des notifications push redondantes ?
- La vidéo en autplay ?

Architecture centrée sur l'usage : dimensionner au besoin réel.
Éviter le **traitement synchrone inutile** : certains mails n'ont pas besoin de partir dans les 50ms – reporter, batcher, lisser.

licensed under CC BY 4.0

Sobriété fonctionnelle et architecture centrée sur l'usage

La question qui précède toutes les autres : Cette fonctionnalité a-t-elle un réel intérêt pour les utilisateurs ?

- Un chatbot IA pour une FAQ de 10 questions ?
- Des notifications push redondantes ?
- La vidéo en autplay ?

Architecture centrée sur l'usage : dimensionner au besoin réel.
Éviter le **traitement synchrone inutile** : certains mails n'ont pas besoin de partir dans les 50ms – reporter, batcher, lisser.
Effacement énergétique : prolongement de carbon-aware, passer en mode dégradé quand le mix électrique est mauvais.

licensed under CC BY 4.0

Des Questions ?



Bibliographie i

ADEME. (2023). Numérique et environnement : entre opportunités et nécessaire sobriété [[Online ; accessed 26-October-2026]]. https://www.ademe.fr/presse/communique-national/numerique-environnement-entre-opportunites-et-necessaire-sobriete/

Codex, A. C. (2023, avril). *PHP and HHVM : Boosting PHP Performance with the HipHop Virtual Machine* [Consulté le 26 octobre 2026]. Reintech. https://reintech.io/blog/boosting-php-performance-with-hhvm

Firestone, D., Putnam, A., Angepat, H., Chiou, D., Caulfield, A., Chung, E., Humphrey, M., Ovtcharov, K., Padhye, J., Burger, D., Maltz, D., Greenberg, A., Mundkur, S., Dabagh, A., Andrewartha, M., Bhanu, V., Chandrappa, H. K., Chaturmohta, S., Lavier, J., … Maltz, D. A. (2018).Azure Accelerated Networking : SmartNICs in the Public Cloud. *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. https://www.microsoft.com/en-us/research/publication/azure-accelerated-networking-smartnics-public-cloud/

Bibliographie ii

- Kuksenko, S. (2020, juillet). *Project Valhalla : Performance Aspects* [Consulté le 26 octobre 2026]. [Presentation slides, OpenJDK]. https://cr.openjdk.org/~skuksenko/valhalla/talk/vp.pdf
- Low-Carbon Power. (2026). Carte de l'intensité carbone de l'électricité (gCO2eq/kWh) [Consulté le 26 octobre 2026]. https://lowcarbonpower.org/fr/map-gCO2eq-kWh
- OpenJDK. (2026). Project Valhalla [Consulté le 26 octobre 2026]. *Oraclethe OpenJDK Community*. https://openjdk.org/projects/valhalla/
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2017).Energy efficiency across programming languages : how do energy, time, and memory relate? *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, 256-267. https://doi.org/10.1145/3136014.3136031

 licensed under CC BY 4.0

Bibliographie iii

- Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G. P., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Larus, J., Peterson, E., Pope, S., … Burger, D. (2014).A reconfigurable fabric for accelerating large-scale datacenter services. *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 13-24. https://doi.org/10.1109/ISCA.2014.6853195
- Radovanović, A. (2020, avril). *Our Data Centers Now Work Harder When the Sun Shines and Wind Blows* [Consulté le 26 octobre 2026]. Google Cloud Blog. https://cloud.google.com/blog/topics/sustainability/googles-approach-to-carbon-aware-data-center
- Wikipedia contributors. (2026a). Microprocessor chronology — Wikipedia, The Free Encyclopedia [[Online; accessed 26-October-2026]]. https://en.wikipedia.org/wiki/%7BM%7Dicroprocessor%5C_chronology
- Wikipedia contributors. (2026b). Tensor Processing Unit — Wikipedia, The Free Encyclopedia [[Online; accessed 26-October-2026]]. https://en.wikipedia.org/wiki/Tensor_Processing_Unit

 licensed under CC BY 4.0

Bibliographie iv



Wilberding, K., & Wells, G. (2019).Facebook's Timeline : 15 Years In [Consulté le 6 février 2019]. *The Wall Street Journal*. https://www.wsj.com/

 licensed under CC BY 4.0