# De bas en haut : attaques sur la microarchitecture depuis un navigateur web

Clémentine Maurice, Graz University of Technology

8 Juin 2017—SSTIC, Rennes, France


Clémentine Maurice
PhD depuis octobre 2015
de Rennes, France

postdoc à TU Graz, Autriche
équipe Secure Systems

+ équipe Secure Systems : Daniel Gruss, Michael Schwarz, Peter Pessl

# Introduction

- infrastructure logicielle sécurisée $\neq$ exécution sécurisée

# Introduction

- infrastructure logicielle sécurisée $\neq$ exécution sécurisée
- fuites d'informations dues au matériel

# Introduction

- infrastructure logicielle sécurisée $\neq$ exécution sécurisée
- fuites d'informations dues au matériel
- vulnérabilités exploitables à haut niveau

# Introduction

- infrastructure logicielle sécurisée $\neq$ exécution sécurisée
- fuites d'informations dues au matériel
- vulnérabilités exploitables à haut niveau
- comme un navigateur web

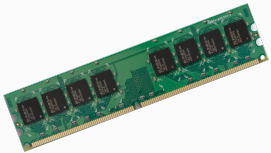
# Introduction

- infrastructure logicielle sécurisée $\neq$ exécution sécurisée
- fuites d'informations dues au matériel
- vulnérabilités exploitables à haut niveau
- comme un navigateur web
- parce que JavaScript c'est juste du code qui s'exécute sur votre machine :)

# Sommaire

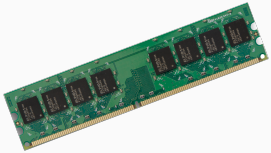
# Sommaire

1. C’est quoi un side channel sur la microarchitecture ?
2. Comment j’utilise ma DRAM pour faire un canal caché ?



# Sommaire

1. C’est quoi un side channel sur la microarchitecture ?
2. Comment j’utilise ma DRAM pour faire un canal caché ?
3. Comment je fais ça en JavaScript ?!



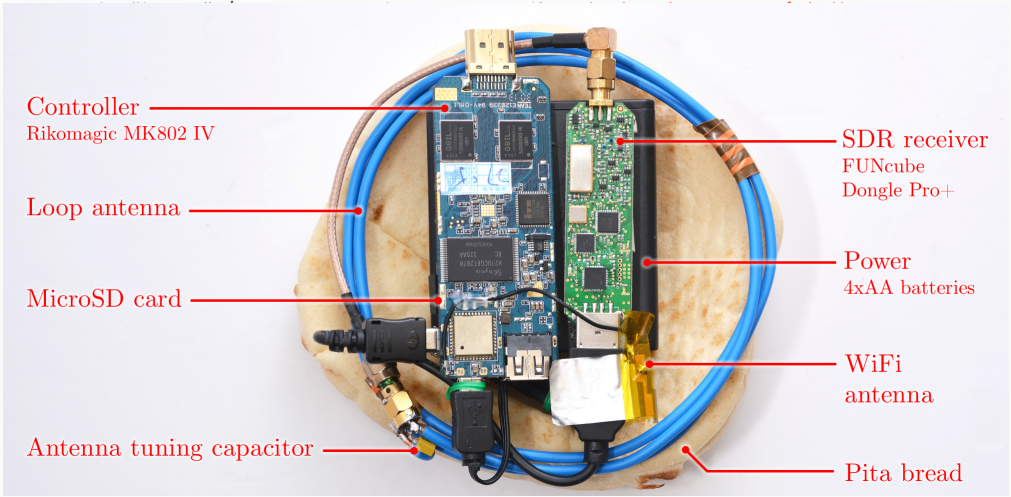
# Sources des fuites

- pas de “bugs” / erreurs $\rightarrow$ beaucoup d’optimisations matérielles

# Sources des fuites

- pas de “bugs” / erreurs $\rightarrow$ beaucoup d’optimisations matérielles
- par la consommation de courant, le rayonnement électromagnétique

# Sources des fuites



# Sources des fuites

- pas de “bugs” / erreurs → beaucoup d’optimisations matérielles
- par la consommation de courant, le rayonnement électromagnétique
  → attaques ciblées, accès physique requis

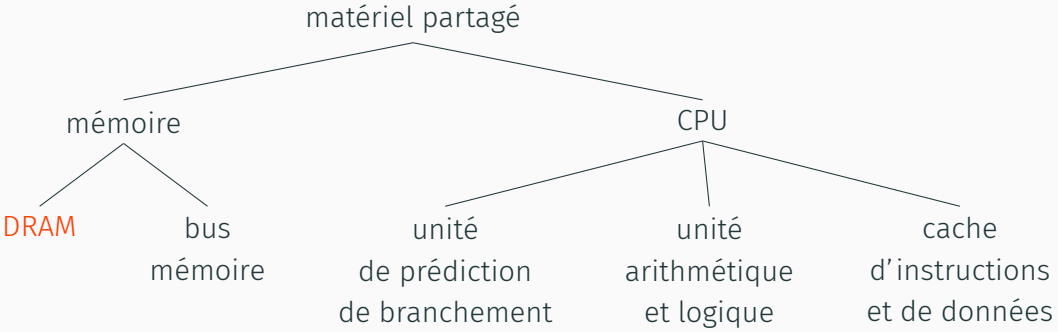
# Sources des fuites

- pas de “bugs” / erreurs → beaucoup d’optimisations matérielles
- par la consommation de courant, le rayonnement électromagnétique
  → attaques ciblées, accès physique requis
- par le matériel partagé et la microarchitecture

# Sources des fuites

- pas de “bugs” / erreurs → beaucoup d’optimisations matérielles
- par la consommation de courant, le rayonnement électromagnétique
  → attaques ciblées, accès physique requis
- par le matériel partagé et la microarchitecture
  → pas d’accès physique requis

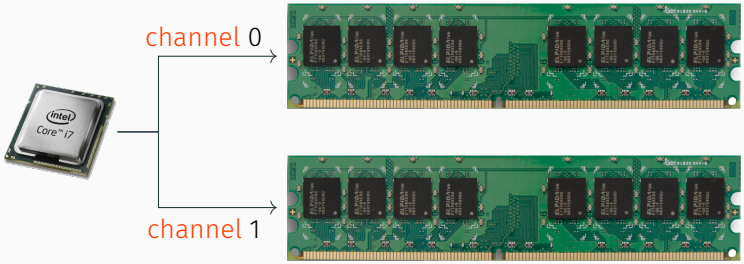
# Matériel partagé



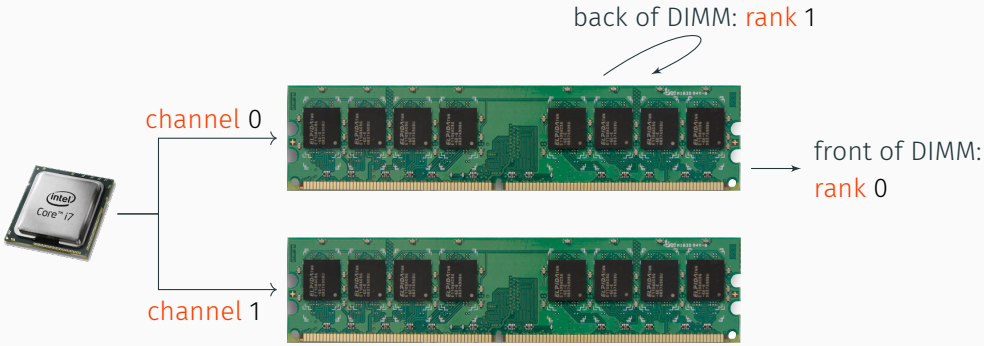
# DRAM et side channels

# Organisation de la DRAM

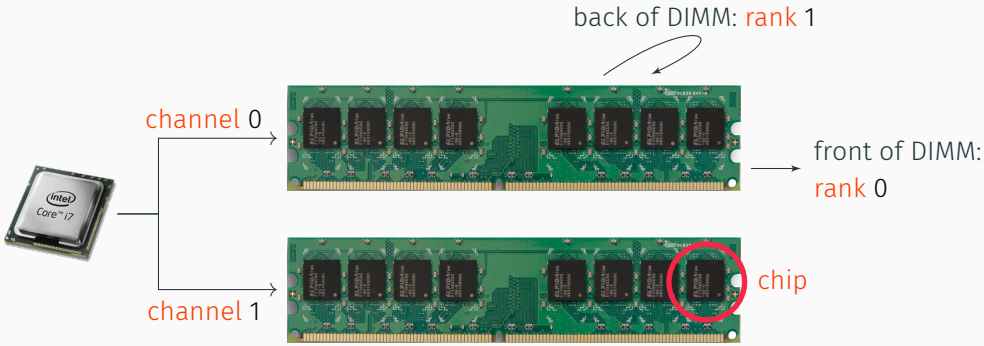
# Organisation de la DRAM



# Organisation de la DRAM



# Organisation de la DRAM



# Organisation de la DRAM

chip


bank 0

row 0
row 1
row 2
...
row 32767

row buffer

# Organisation de la DRAM

chip


bank 0

row 0
row 1
row 2
...
row 32767

row buffer

64k cellules
1 condensateur,
1 transitor chacune

# Row buffer de la DRAM

- la DRAM ne peut que lire des *rows* entiers

# Row buffer de la DRAM

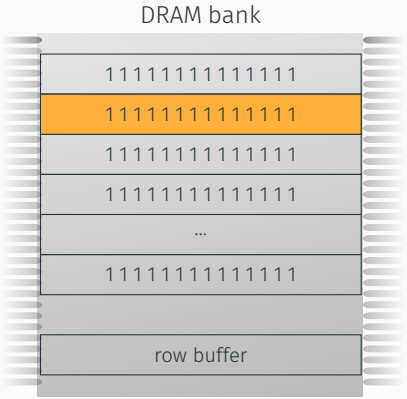
- la DRAM ne peut que lire des *rows* entiers
- condensateurs se déchargent quand on “lit les bits”
- bufferiser les bits quand on les lit
- réécrire les bits dans les cellules quand on a fini

# Row buffer de la DRAM

- la DRAM ne peut que lire des *rows* entiers
- condensateurs se déchargent quand on “lit les bits”
- bufferiser les bits quand on les lit
- réécrire les bits dans les cellules quand on a fini

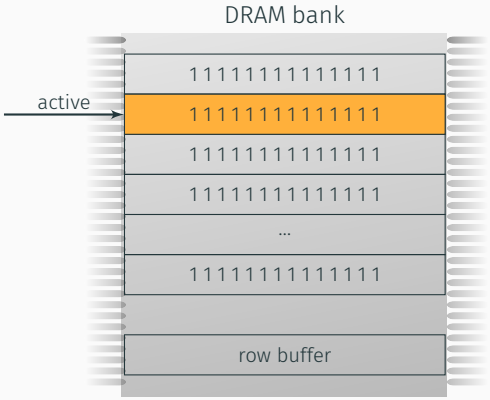
→ row buffer

# Comment on lit sur la DRAM ?


CPU veut accéder au row 1

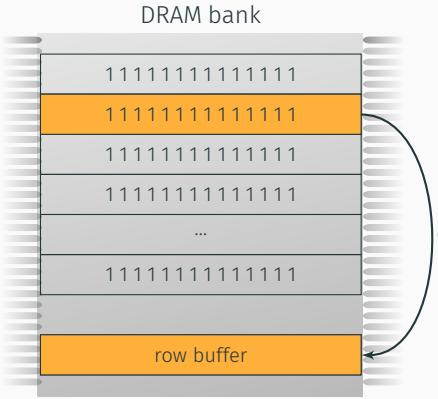
# Comment on lit sur la DRAM ?


CPU veut accéder au row 1

$\rightarrow$ row 1 activé

# Comment on lit sur la DRAM ?

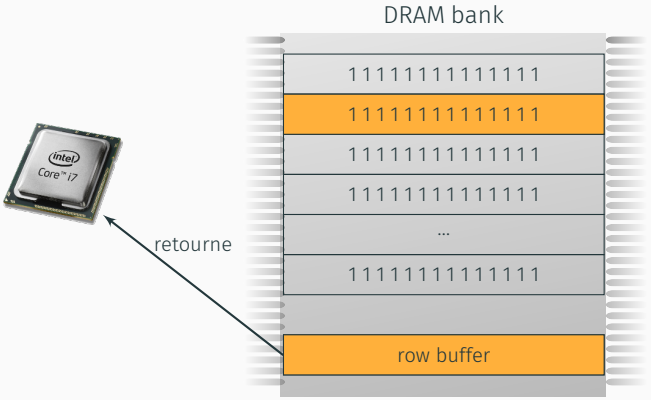




CPU veut accéder au row 1

→ row 1 activé

→ row 1 copié au row buffer

# Comment on lit sur la DRAM ?

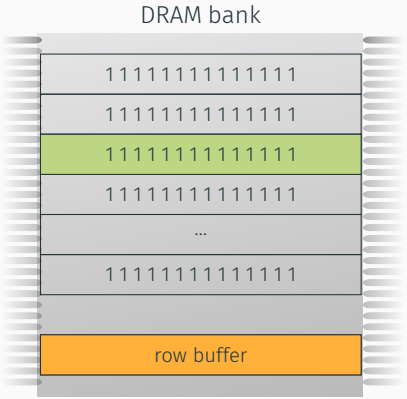



CPU veut accéder au row 1

→ row 1 activé

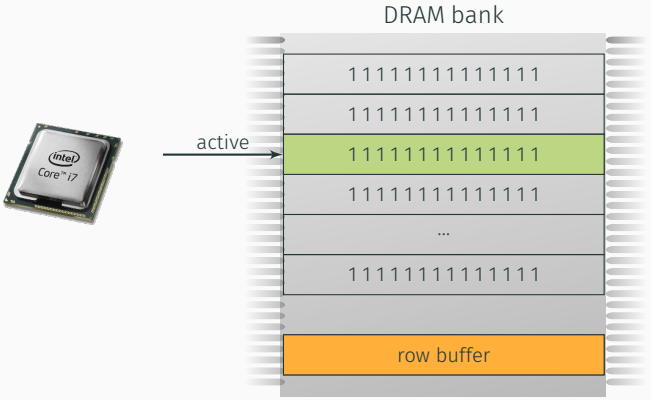
→ row 1 copié au row buffer

# Comment on lit sur la DRAM ?


CPU veut accéder au row 2

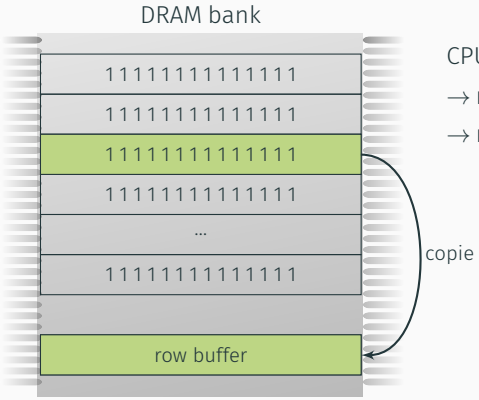
# Comment on lit sur la DRAM ?




CPU veut accéder au row 2

$\rightarrow$ row 2 activé

# Comment on lit sur la DRAM ?

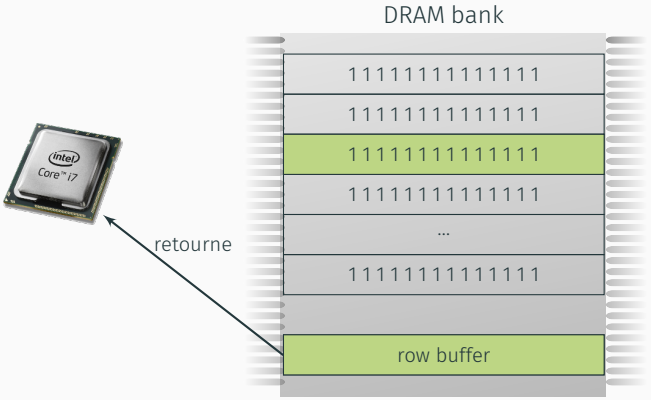




CPU veut accéder au row 2

$\rightarrow$ row 2 activé

$\rightarrow$ row 2 copié au row buffer

# Comment on lit sur la DRAM ?

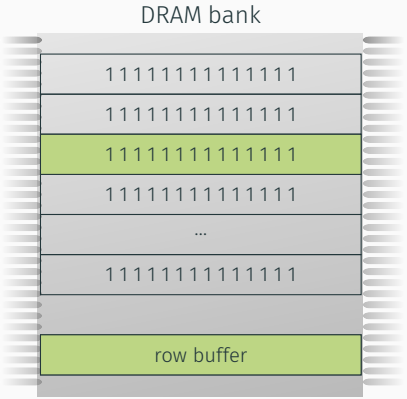



CPU veut accéder au row 2

→ row 2 activé

→ row 2 copié au row buffer

# Comment on lit sur la DRAM ?

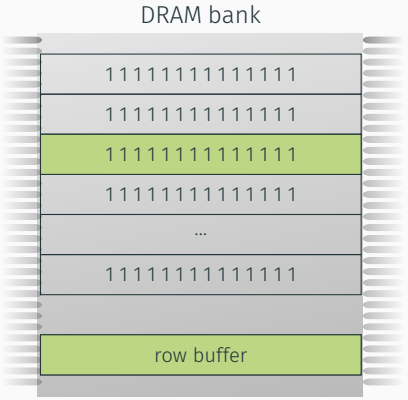

CPU veut accéder au row 2

→ row 2 activé

→ row 2 copié au row buffer

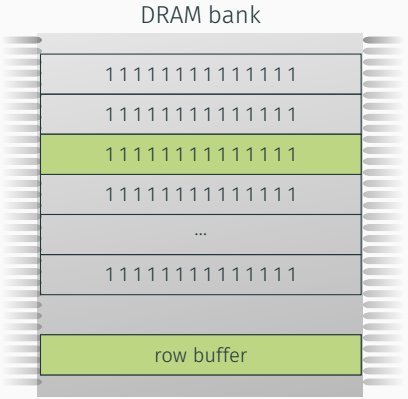
→ lent (row conflict)

# Comment on lit sur la DRAM ?


CPU veut accéder au row 2 (encore)

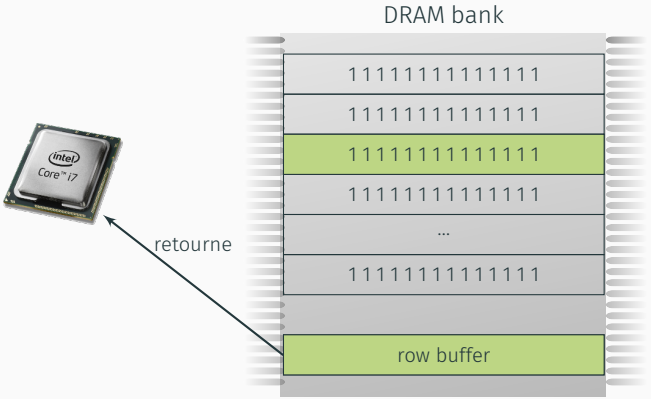
# Comment on lit sur la DRAM ?


CPU veut accéder au row 2 (encore)

→ row 2 déjà dans le row buffer

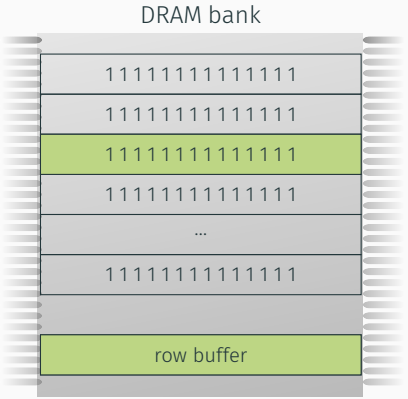
# Comment on lit sur la DRAM ?




CPU veut accéder au row 2 (encore)

→ row 2 déjà dans le row buffer

# Comment on lit sur la DRAM ?

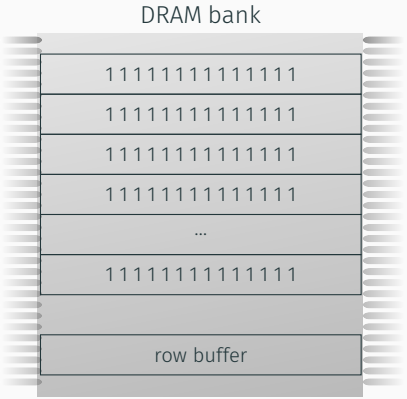




CPU veut accéder au row 2 (encore)

→ row 2 déjà dans le row buffer

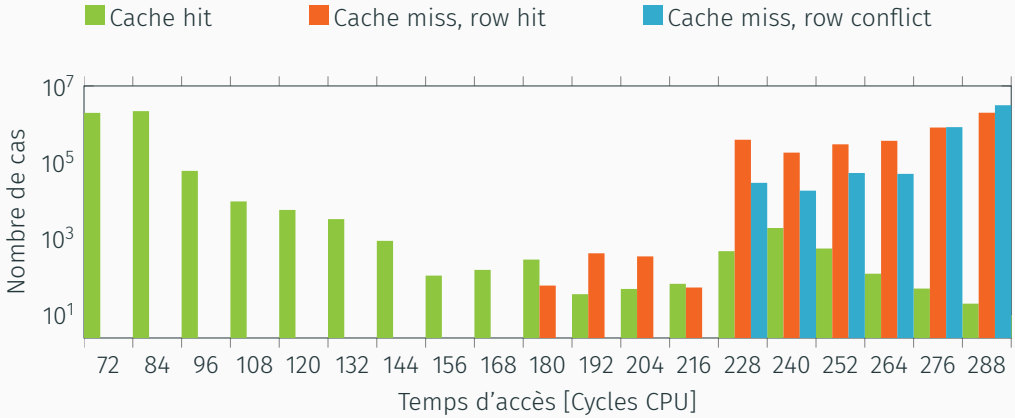
→ rapide (row hit)

# Comment on lit sur la DRAM ?

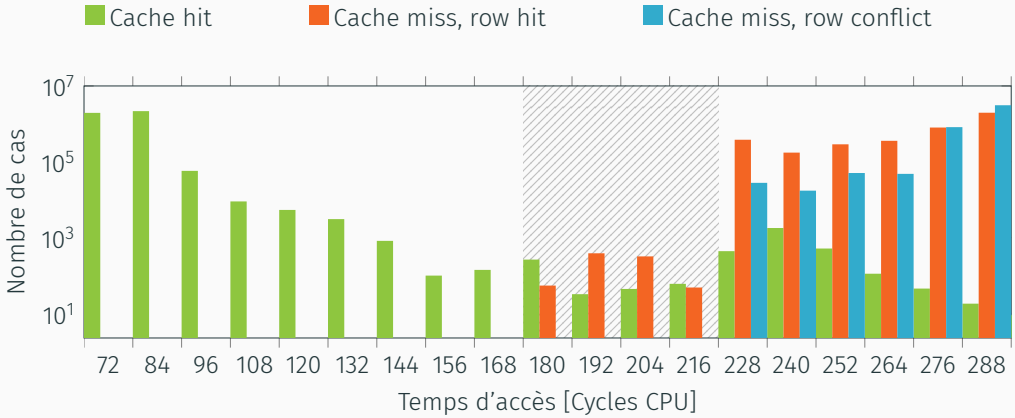




row buffer = cache

# Différences de timing de la DRAM



# Différences de timing de la DRAM



# Side channels sur la DRAM ?

- les row buffers sont des caches

# Side channels sur la DRAM ?

- les row buffers sont des caches
- on observe des différences de timing

# Side channels sur la DRAM ?

- les row buffers sont des caches
- on observe des différences de timing
- comment on les exploite ?

# Side channels sur la DRAM ?

- les row buffers sont des caches
- on observe des différences de timing
- comment on les exploite ?
- on cible les addresses dans le même channel, rank et bank

# Side channels sur la DRAM ?

- les row buffers sont des caches
- on observe des différences de timing
- comment on les exploite ?
- on cible les addresses dans le même channel, rank et bank
- mais les fonctions de mapping de la DRAM ne sont pas documentées

# Side channels sur la DRAM ?

- les row buffers sont des caches
- on observe des différences de timing
- comment on les exploite ?
- on cible les addresses dans le même channel, rank et bank
- mais les fonctions de mapping de la DRAM ne sont pas documentées

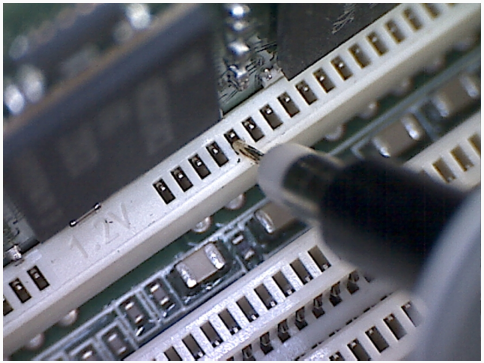
$\rightarrow$ on n'a qu'à les reverse !

P. Pessl et al. "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks". In: *USENIX Security Symposium*. 2016

Let’s reverse-engineer the DRAM!

# Approche physique

- 1e phase : probe physique et oscilloscope



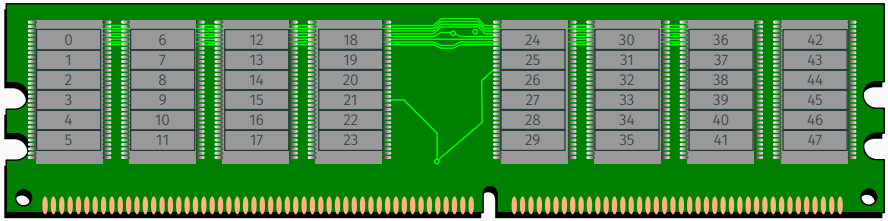
# Approche automatisée

- 2e phase : reverse entièrement logiciel et automatisé

# Approche automatisée

- 2e phase : reverse entièrement logiciel et automatisé
- utilise les différences de timing des row conflicts

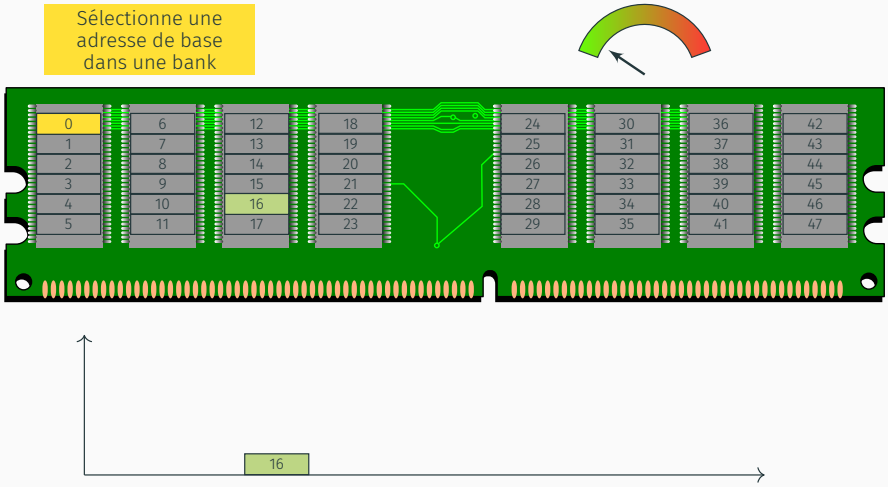
# Approche automatisée




Bank différente

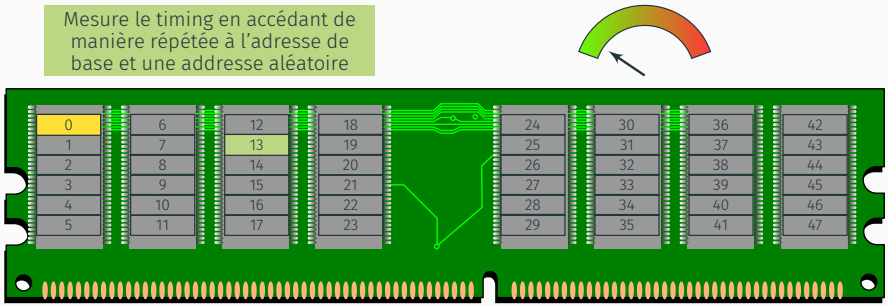
Même bank

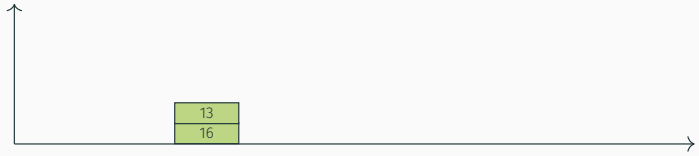
# Approche automatisée



# Approche automatisée

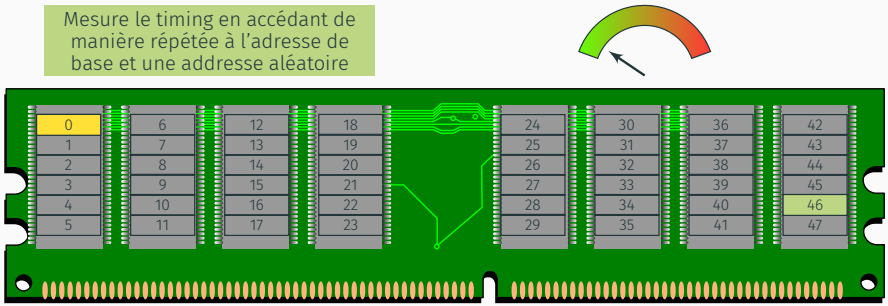
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

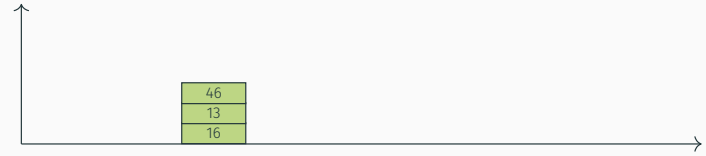


# Approche automatisée

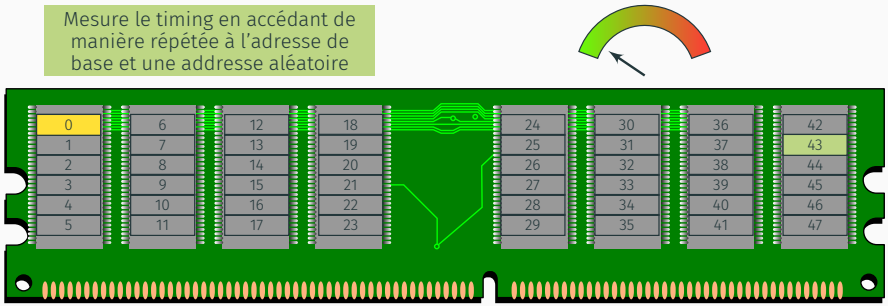
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

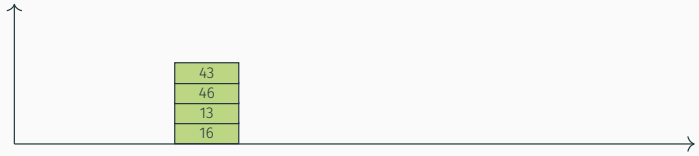


# Approche automatisée

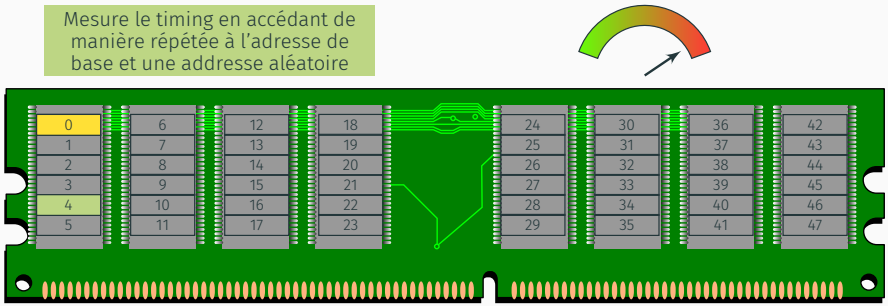
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

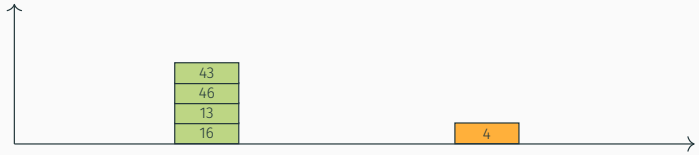


# Approche automatisée

Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire



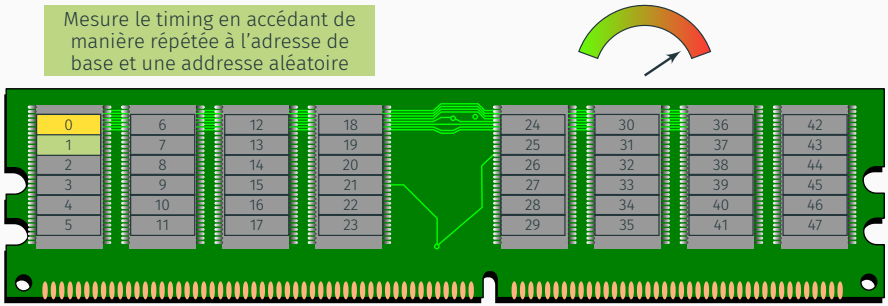



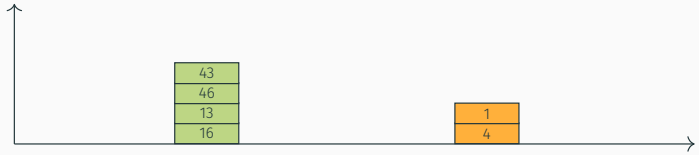

Bank différente

Même bank

# Approche automatisée

Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire



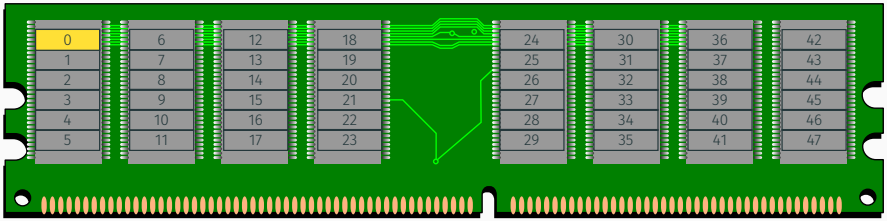



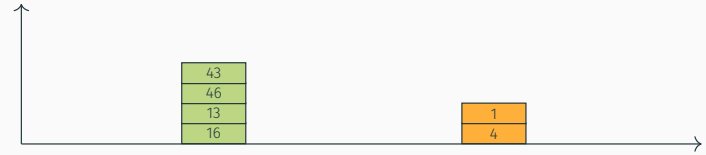

Bank différente

Même bank

# Approche automatisée

Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

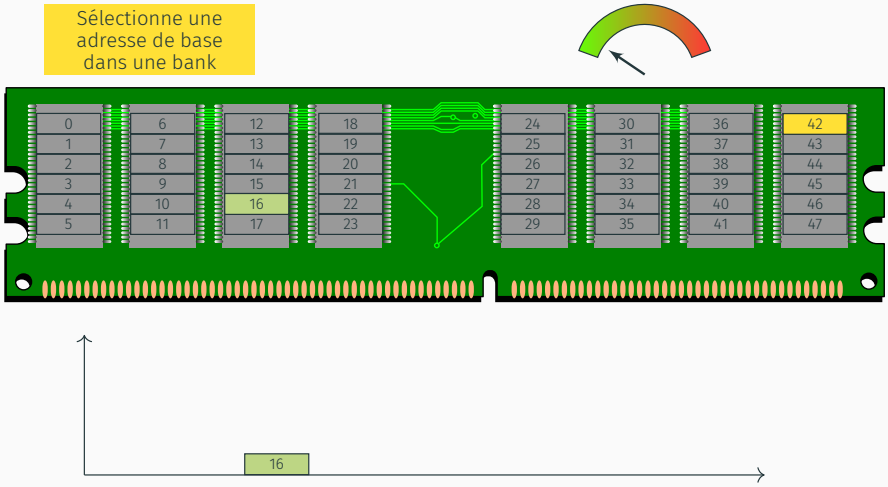



Bank différente

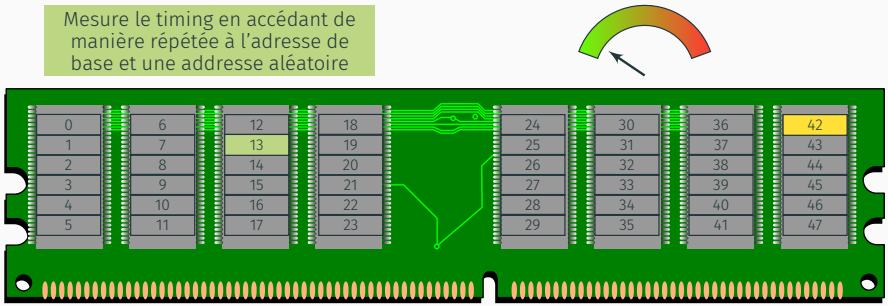
Même bank

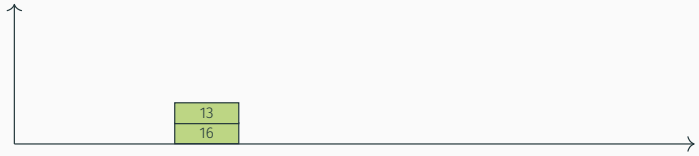
# Approche automatisée



# Approche automatisée

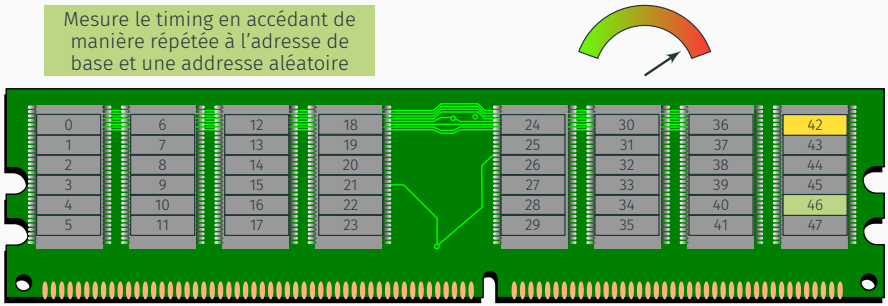
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

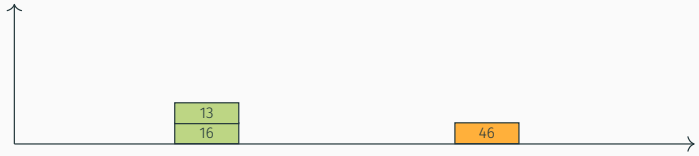


# Approche automatisée

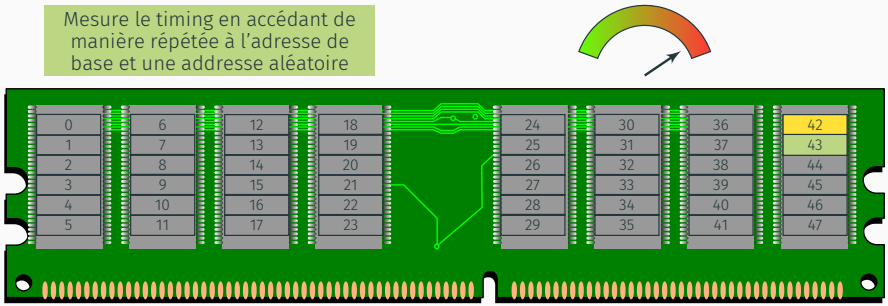
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

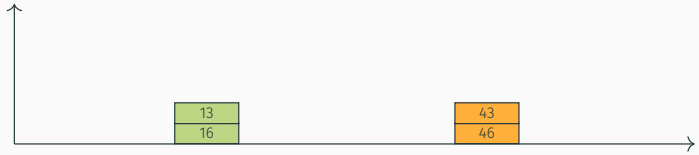


# Approche automatisée

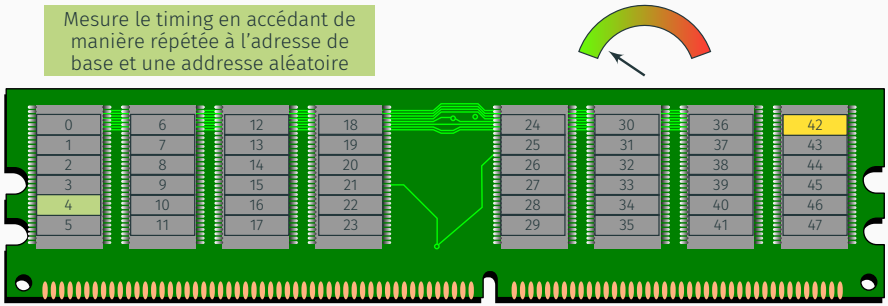
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

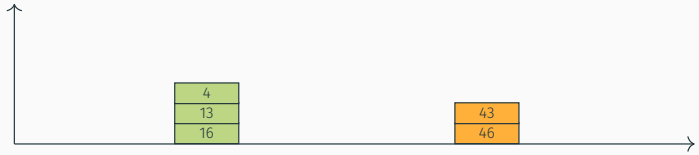


# Approche automatisée

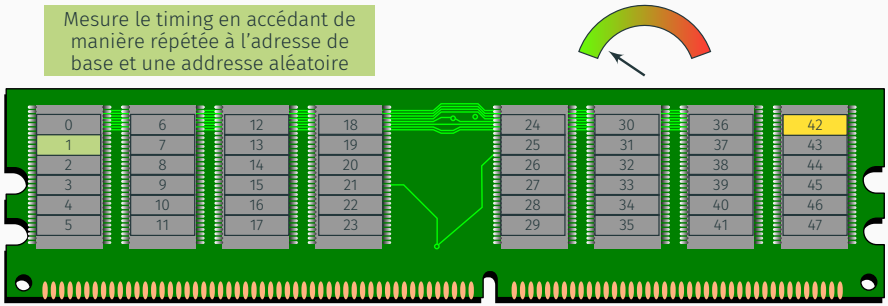
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

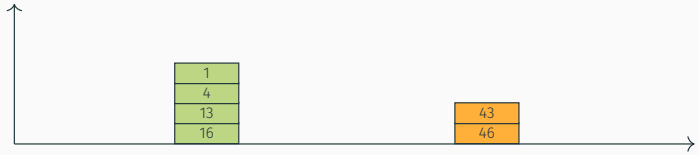


# Approche automatisée

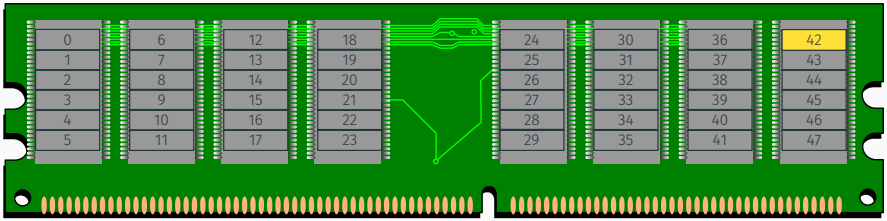
Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire

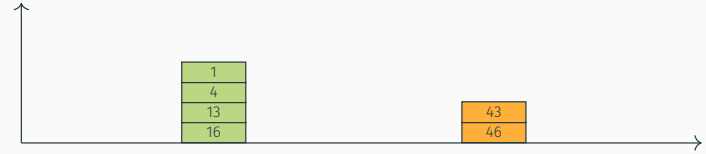


# Approche automatisée

Mesure le timing en accédant de manière répétée à l'adresse de base et une addresse aléatoire



# Trouver les fonctions (1)

- on a un ensemble d'adresses qui map chaque bank

# Trouver les fonctions (1)

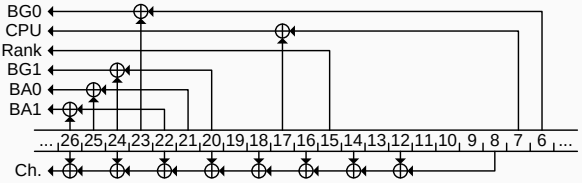
- on a un ensemble d'adresses qui map chaque bank
- approche #1 : système d'équations linéaires
  $\rightarrow$ la solution donne les bits utilisés dans les fonctions de mapping

# Trouver les fonctions (1)

- on a un ensemble d'adresses qui map chaque bank
- approche #1 : système d'équations linéaires
  → la solution donne les bits utilisés dans les fonctions de mapping
- approche #2 : brute-force les fonctions XOR et vérifier
  → rapide quand même (quelques secondes)

# Trouver les fonctions (2)

- toolkit pour reverse les fonctions de manière automatisée
- par ex. sur notre serveur Haswell-EP




https://github.com/IAIK/drama

Qu’est-ce qu’on fait maintenant ?

# DRAMA: DRAM Addressing attacks

- inférer les accès d'une victime similairement aux attaques sur les caches

# DRAMA: DRAM Addressing attacks

- inférer les accès d'une victime similairement aux attaques sur les caches
- marche entre VMs, entre coeurs, et entre CPUs

# DRAMA: DRAM Addressing attacks

- inférer les accès d'une victime similairement aux attaques sur les caches
- marche entre VMs, entre coeurs, et entre CPUs
- canaux cachés et attaques par canal auxiliaire

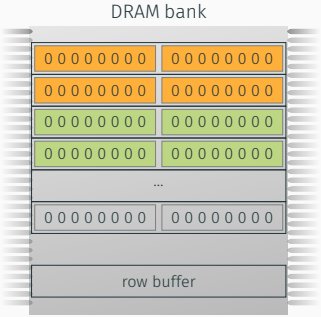
# DRAMA: DRAM Addressing attacks

- inférer les accès d'une victime similairement aux attaques sur les caches
- marche entre VMs, entre coeurs, et entre CPUs
- canaux cachés et attaques par canal auxiliaire
- canal caché : deux processus communiquent entre eux
  - alors qu'ils n'y sont pas autorisés, par ex. entre VMs

# DRAMA: DRAM Addressing attacks

- inférer les accès d'une victime similairement aux attaques sur les caches
- marche entre VMs, entre coeurs, et entre CPUs
- canaux cachés et attaques par canal auxiliaire
- canal caché : deux processus communiquent entre eux
  - alors qu'ils n'y sont pas autorisés, par ex. entre VMs
- attaque par canal auxiliaire : un processus malicieux espionne des processus bénins
  - par ex. espionne les frappes de clavier

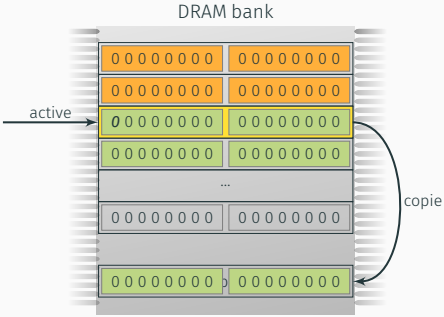
# Canal caché DRAMA


émetteur et récepteur choisissent une bank

récepteur accède continuellement au row *i*

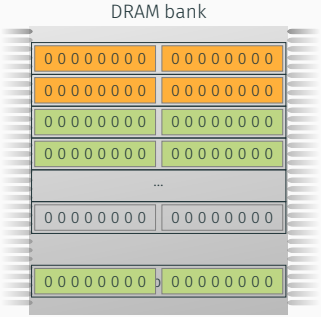
# Canal caché DRAMA


émetteur et récepteur choisissent une bank

récepteur accède continuellement au row *i*

# Canal caché DRAMA

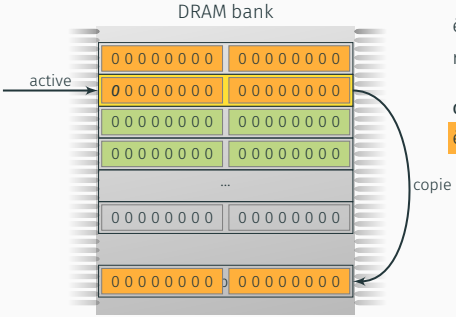




émetteur et récepteur choisissent une bank

récepteur accède continuellement au row *i*

**cas #1 : émetteur transmet 1**

# Canal caché DRAMA

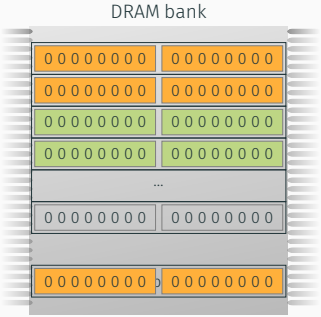

émetteur et récepteur choisissent une bank

récepteur accède continuellement au row $i$

**cas #1 : émetteur transmet 1**

émetteur accède au row $j \neq i$

# Canal caché DRAMA

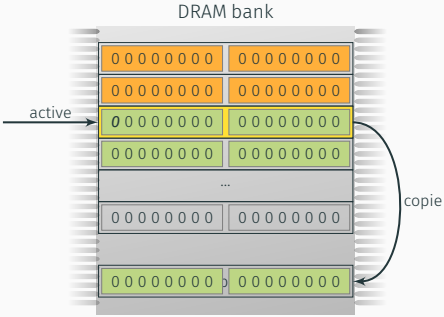

émetteur et récepteur choisissent une bank

récepteur accède continuellement au row $i$

**cas #1 : émetteur transmet 1**

émetteur accède au row $j \neq i$

# Canal caché DRAMA


émetteur et récepteur choisissent une bank

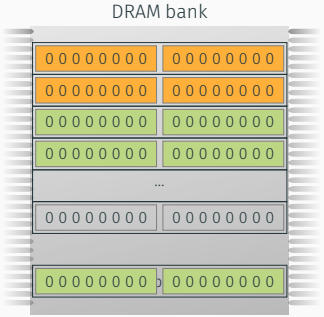
récepteur accède continuellement au row $i$

**cas #1 : émetteur transmet 1**

émetteur accède au row $j \neq i$

prochain accès récepteur $\rightarrow$ copie

# Canal caché DRAMA


émetteur et récepteur choisissent une bank

récepteur accède continuellement au row $i$

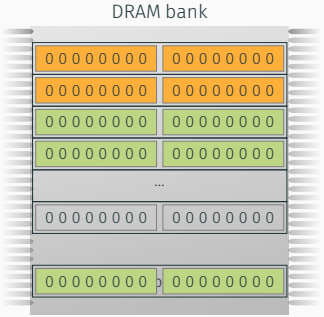
**cas #1 : émetteur transmet 1**

émetteur accède au row $j \neq i$

prochain accès récepteur $\rightarrow$ copie

$\rightarrow$ **lent**

# Canal caché DRAMA

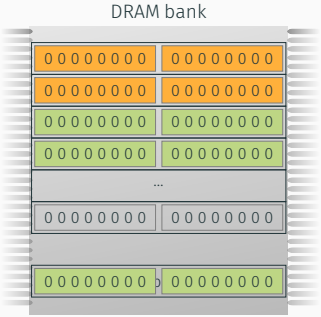




émetteur et récepteur choisissent une bank

récepteur accède continuellement au row *i*

**cas #2 : émetteur transmet 0**

# Canal caché DRAMA

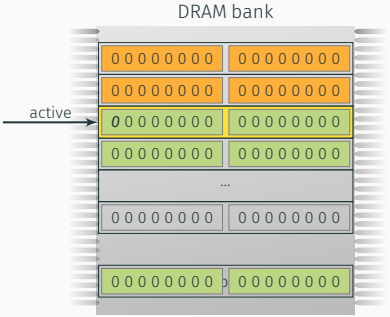

émetteur et récepteur choisissent une bank

récepteur accède continuellement au row *i*

**cas #2 : émetteur transmet 0**

l'émetteur ne fait rien

# Canal caché DRAMA


émetteur et récepteur choisissent une bank

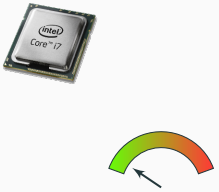
récepteur accède continuellement au row *i*

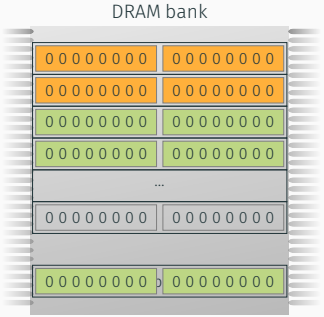
**cas #2 : émetteur transmet 0**

l'émetteur ne fait rien

prochain accès récepteur → déjà dans le buffer

# Canal caché DRAMA




émetteur et récepteur choisissent une bank

récepteur accède continuellement au row *i*

**cas #2 : émetteur transmet 0**

l'émetteur ne fait rien

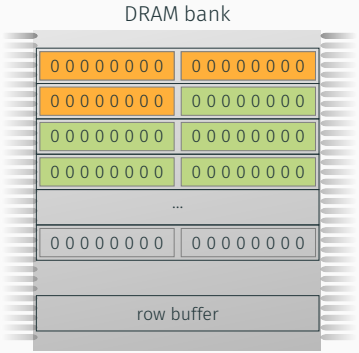
prochain accès récepteur → déjà dans le buffer

→ **rapide**

Deux applications peuvent communiquer
entre elles de manière cachée

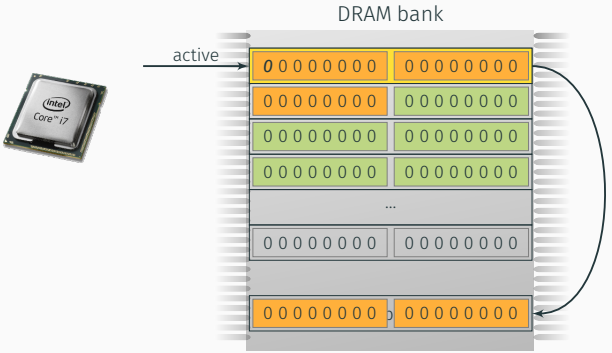
Mais est-ce qu'on peut utiliser ça pour espionner ?

# Attaque par canal auxiliaire DRAMA


espion et victime partagent un row *i*

# Attaque par canal auxiliaire DRAMA

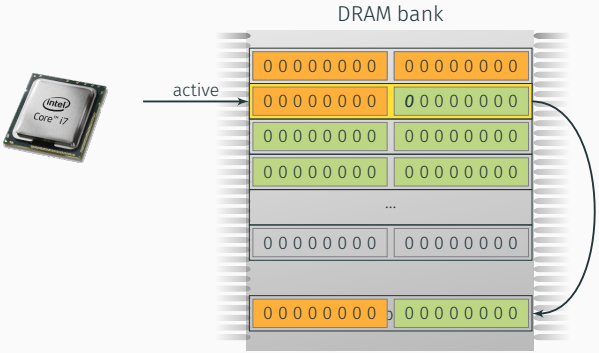



espion et victime partagent un row $i$

**cas #1**

espion accède au row $j \neq i$, copie

# Attaque par canal auxiliaire DRAMA



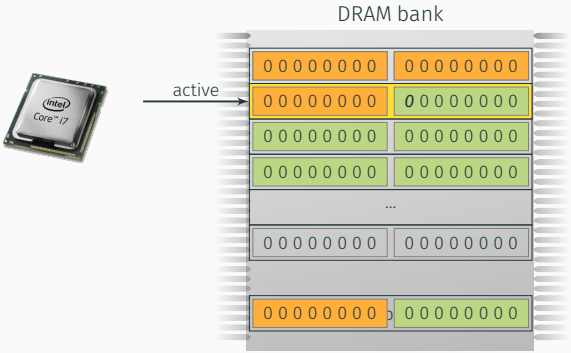

espion et victime partagent un row $i$

**cas #1**

espion accède au row $j \neq i$, copie

**victime accède** au row $i$, copie

# Attaque par canal auxiliaire DRAMA




espion et victime partagent un row $i$

**cas #1**

espion accède au row $j \neq i$, copie

**victime accède** au row $i$, copie

espion accède au row $i$, pas de copie

# Attaque par canal auxiliaire DRAMA


espion et victime partagent un row $i$

**cas #1**

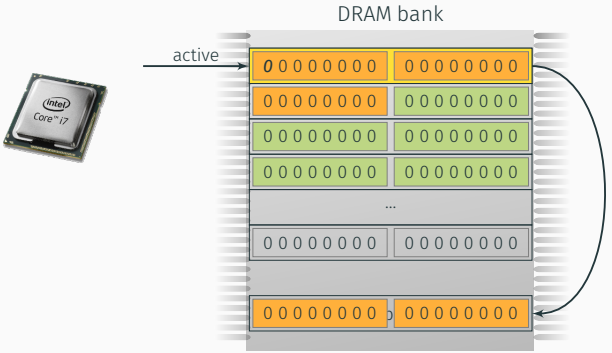
espion accède au row $j \neq i$, copie

**victime accède** au row $i$, copie

espion accède au row $i$, pas de copie

$\rightarrow$ **rapide**

# Attaque par canal auxiliaire DRAMA

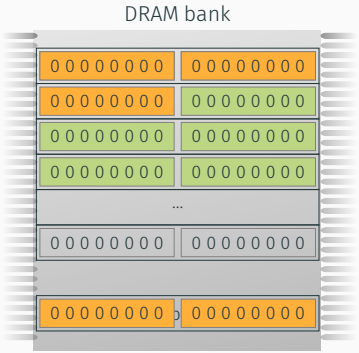



espion et victime partagent un row $i$

**cas #2**

espion accède au row $j \neq i$, copie

# Attaque par canal auxiliaire DRAMA

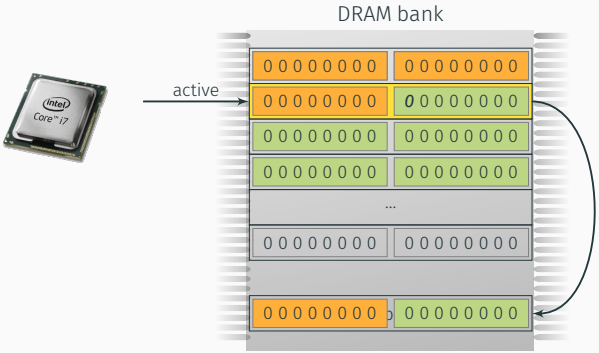

espion et victime partagent un row $i$

**cas #2**

espion accède au row $j \neq i$, copie

**pas d'accès de la victime** au row $i$

# Attaque par canal auxiliaire DRAMA



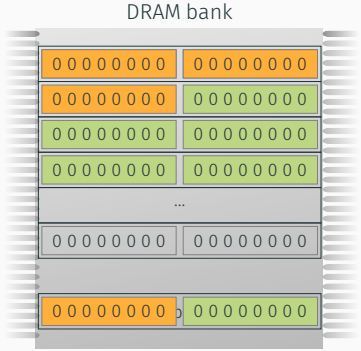

espion et victime partagent un row $i$

**cas #2**

| |
|---|
| espion accède au row $j \neq i$, copie |
| **pas d'accès de la victime** au row $i$ |
| espion accède au row $i$, copie |

# Attaque par canal auxiliaire DRAMA


DRAM bank




espion et victime partagent un row $i$

**cas #2**

espion accède au row $j \neq i$, copie

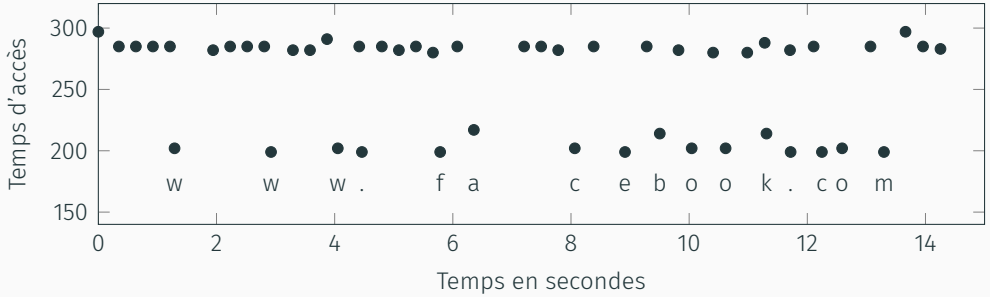
**pas d'accès de la victime** au row $i$

espion accède au row $i$, copie

$\rightarrow$ **lent**

# Espionner les frappes de clavier sur la barre URL de Firefox

- attaque “template”
  - allouer une large fraction de la mémoire pour partager un row avec la victime
  - profiler la mémoire et sauver les ratios de row hit pour chaque adresse



On va devoir écrire un tas de code en C
Au moins on est saufs avec JavaScript !

Member Rowhammer.js?

# Canaux cachés sur la DRAM en JavaScript ?

# Pourquoi JavaScript ?

- code exécuté dans une sandbox

# Pourquoi JavaScript ?

- code exécuté dans une sandbox
- donc ne peut rien faire de méchant, n'est-ce pas ?

# Pourquoi JavaScript ?

- code exécuté dans une sandbox
- donc ne peut rien faire de méchant, n'est-ce pas ?
- sauf que les side channels ne font que des opérations bénignes

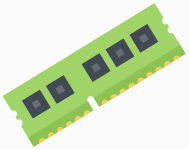
# Pourquoi JavaScript ?

- code exécuté dans une sandbox
- donc ne peut rien faire de méchant, n'est-ce pas ?
- sauf que les side channels ne font que des opérations bénignes
    1. accéder à sa propre mémoire

# Pourquoi JavaScript ?

- code exécuté dans une sandbox
- donc ne peut rien faire de méchant, n'est-ce pas ?
- sauf que les side channels ne font que des opérations bénignes
  1. accéder à sa propre mémoire
  2. mesurer le temps

# Challenges avec JavaScript



1. Pas de connaissance des adresses physiques

2. Pas d'instruction pour flusher le cache

3. Pas de timers haute résolution

# #1. Pas de connaissance des adresses physiques

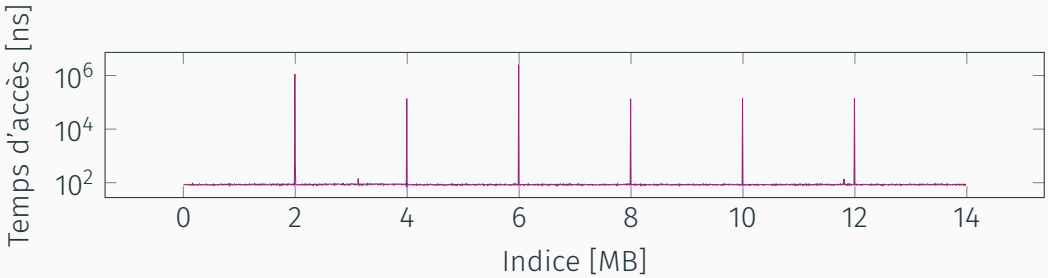
- optimisation OS : utilise des Transparent Huge Pages (THP, 2MB pages)
- 21 derniers bits (2MB) de l'adresse physique
- = 21 derniers bits (2MB) de l'adresse virtuelle

# #1. Pas de connaissance des adresses physiques

- optimisation OS : utilise des Transparent Huge Pages (THP, 2MB pages)
- 21 derniers bits (2MB) de l'adresse physique
- = 21 derniers bits (2MB) de l'adresse virtuelle

→ quels indices d'un tableau JS?

# #1. Obtenir le début d'une THP



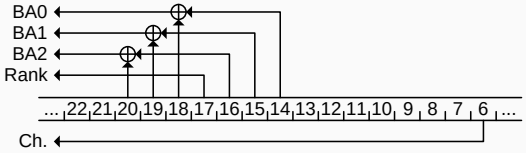

- pages physiques des THPs mappées à la demande

→ page fault au premier accès d'une THP allouée

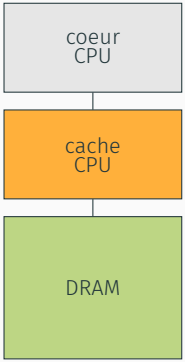
D. Gruss et al. "Practical Memory Deduplication Attacks in Sandboxed JavaScript". In: *ESORICS'15*. 2015.

# #1. Choisir les adresses physiques

- on connaît les 21 derniers bits des adresses physiques
- suffisant pour la plupart des systèmes, par ex. Sandy Bridge avec DDR3



# #2. Pas d'instruction pour flusher le cache

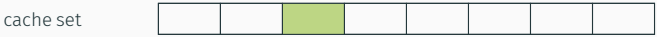



- mesurer les timings de la DRAM
- seuls les accès non en cache atteignent la DRAM
- pas d'instruction `clflush`

→ élimine les lignes avec d'autres accès mémoires

# #2. Contourner le cache CPU : Idée de base

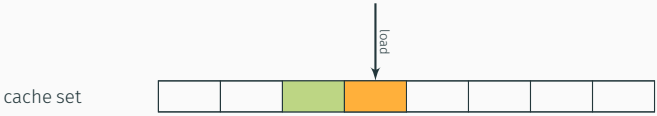
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

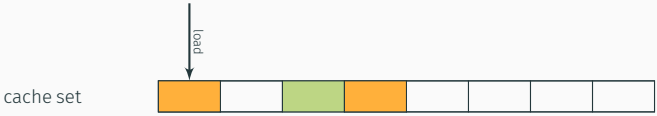
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

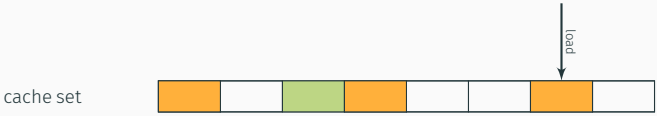
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

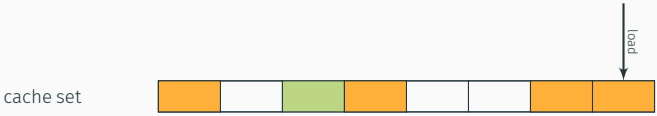
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

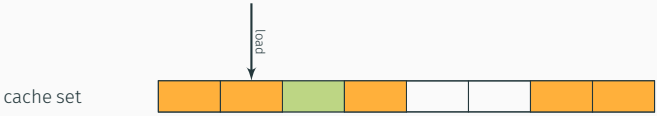
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

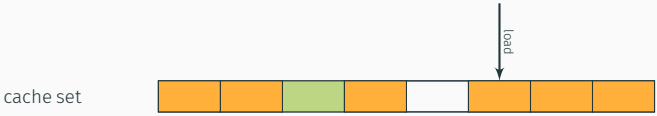
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

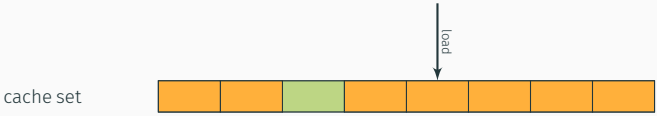
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

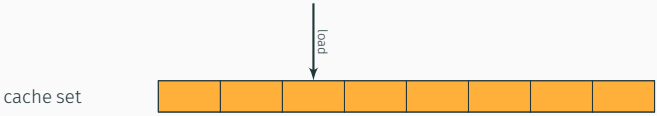
- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

- éliminer la ligne de cache en utilisant seulement des accès mémoire




D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA’16*. 2016.

# #2. Contourner le cache CPU : Idée de base

- éliminer la ligne de cache en utilisant seulement des accès mémoire


- c'est plus compliqué que ça : la politique de remplacement n'est pas LRU

D. Gruss et al. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA'16*. 2016.

# #2. Contourner le cache CPU : Idée de base

- éliminer la ligne de cache en utilisant seulement des accès mémoire


- c'est plus compliqué que ça : la politique de remplacement n'est pas LRU
- mais on a déjà résolu ce problème :)

D. Gruss et al. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA'16*. 2016.

# #3. Timers haute résolution ?

- mesure de différences de timing faibles : besoin d'un timer haute résolution

# #3. Timers haute résolution ?

- mesure de différences de timing faibles : besoin d'un timer haute résolution
- natif : `rdtsc`, timestamp en cycles CPU

# #3. Timers haute résolution ?

- mesure de différences de timing faibles : besoin d'un timer haute résolution
- natif : `rdtsc`, timestamp en cycles CPU
- JavaScript : `performance.now()` a la plus fine résolution

# #3. Timers haute résolution ?

- mesure de différences de timing faibles : besoin d'un timer haute résolution
- natif : `rdtsc`, timestamp en cycles CPU
- JavaScript : `performance.now()` a la plus fine résolution

**`performance.now()`**

[…] represent times as floating-point numbers with up to microsecond precision.

— Mozilla Developer Network

# Timers haute résolution en JavaScript

# C’était mieux avant

- avant septembre 2015 : `performance.now()` a une résolution à la nanoseconde près

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015.

`https://www.mozilla.org/en-US/security/advisories/mfsa2015-114/`

# C’était mieux avant

- avant septembre 2015 : `performance.now()` a une résolution à la nanoseconde près
- Oren et al. on démontré une attaque sur le cache CPU en JavaScript

---

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015.

`https://www.mozilla.org/en-US/security/advisories/mfsa2015-114/`

# C’était mieux avant

- avant septembre 2015 : `performance.now()` a une résolution à la nanoseconde près
- Oren et al. on démontré une attaque sur le cache CPU en JavaScript
- “fixé” depuis Firefox 41 : arrondi à 5 µs

---

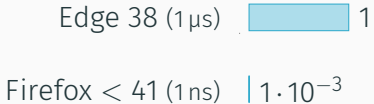
Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015.

`https://www.mozilla.org/en-US/security/advisories/mfsa2015-114/`

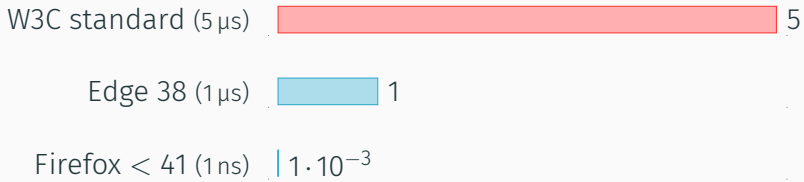
# À la microseconde près ?

Firefox $< 41$ (1 ns) | $1 \cdot 10^{-3}$

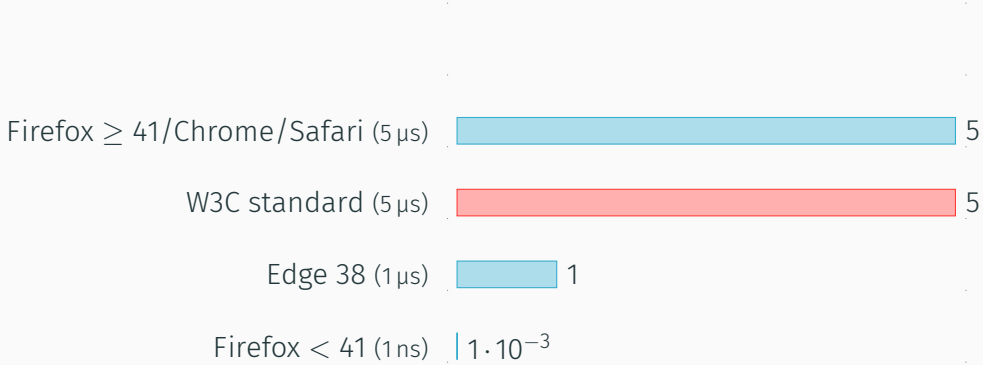
# À la microseconde près ?



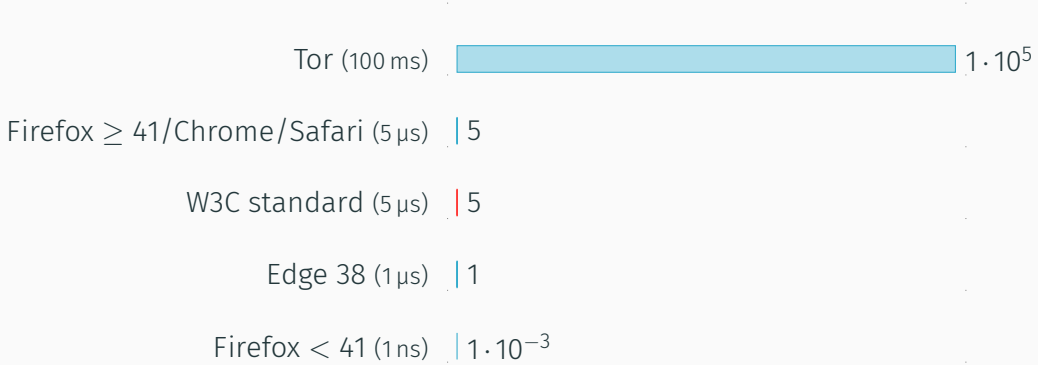
# À la microseconde près ?



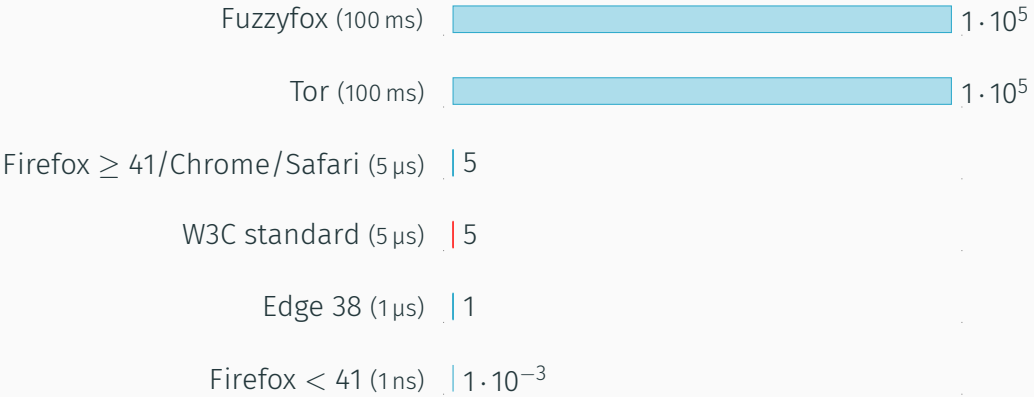
# À la microseconde près ?



# À la microseconde près ?



# À la microseconde près ?




D. Kohlbrenner et al. “Trusted Browsers for Uncertain Times”. In: *USENIX Security Symposium*. 2016

# On peut faire mieux !

- une résolution à la microseconde près ce n'est pas assez

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

# On peut faire mieux !

- une résolution à la microseconde près ce n'est pas assez
- deux approches

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

# On peut faire mieux !

- une résolution à la microseconde près ce n'est pas assez
- deux approches
    1. récupérer une haute résolution depuis le timer disponible

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

# On peut faire mieux !

- une résolution à la microseconde près ce n'est pas assez
- deux approches
  1. récupérer une haute résolution depuis le timer disponible
  2. construire notre propre timer haute résolution

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

# Récupérer une haute résolution : Interpolation

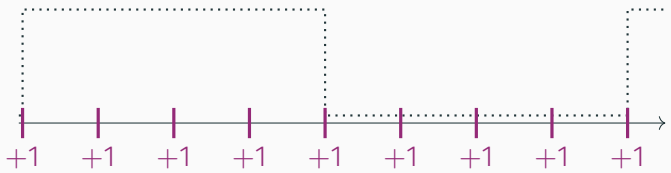
- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge

# Récupérer une haute résolution : Interpolation

- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge

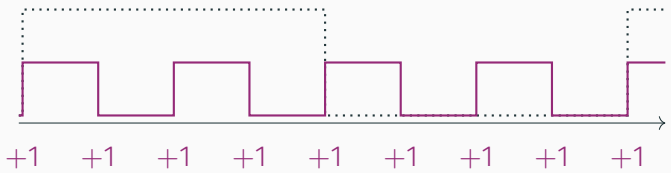
# Récupérer une haute résolution : Interpolation

- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge



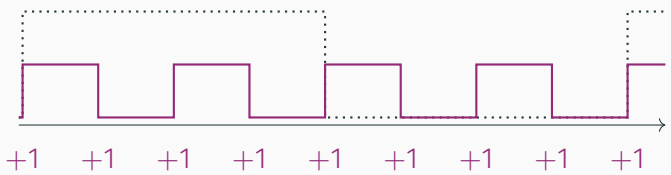
# Récupérer une haute résolution : Interpolation

- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge



# Récupérer une haute résolution : Interpolation

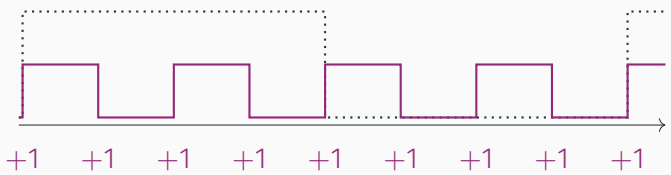
- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge




- pour mesurer avec une haute résolution

# Récupérer une haute résolution : Interpolation

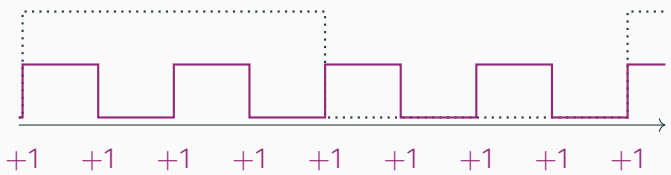
- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge




- pour mesurer avec une haute résolution
  - commencer la mesure au front d'horloge

# Récupérer une haute résolution : Interpolation

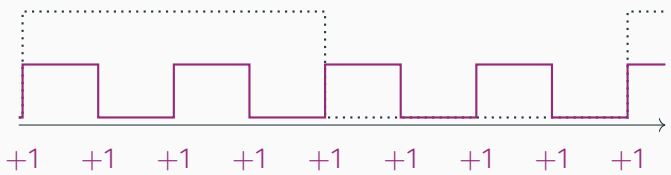
- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge




- pour mesurer avec une haute résolution
  - commencer la mesure au front d'horloge
  - incrémenter une variable jusqu'au prochain front

# Récupérer une haute résolution : Interpolation

- mesurer la fréquence d'incrément d'une variable entre deux tics d'horloge




- pour mesurer avec une haute résolution
  - commencer la mesure au front d'horloge
  - incrémenter une variable jusqu'au prochain front
- Firefox/Chrome : 500 ns, Tor : 15 µs

# Récupérer une haute résolution : Padding

- souvent suffisant de savoir quelle fonction est la plus lente

# Récupérer une haute résolution : Padding

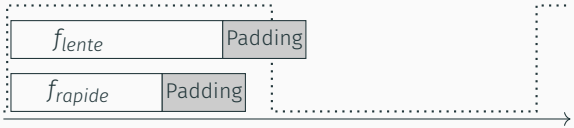
- souvent suffisant de savoir quelle fonction est la plus lente

# Récupérer une haute résolution : Padding

- souvent suffisant de savoir quelle fonction est la plus lente

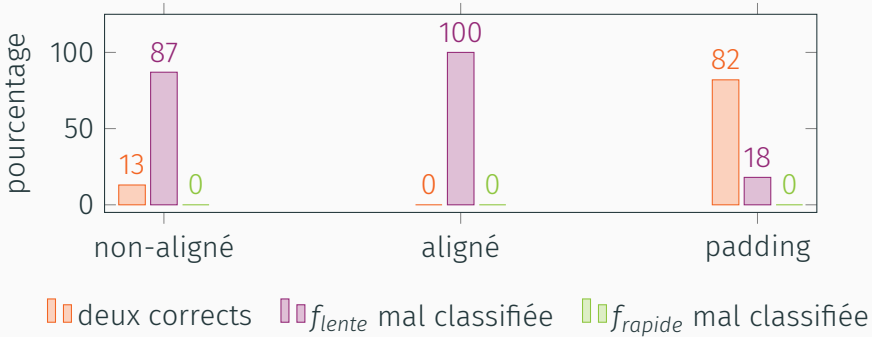
# Récupérer une haute résolution : Padding

- souvent suffisant de savoir quelle fonction est la plus lente

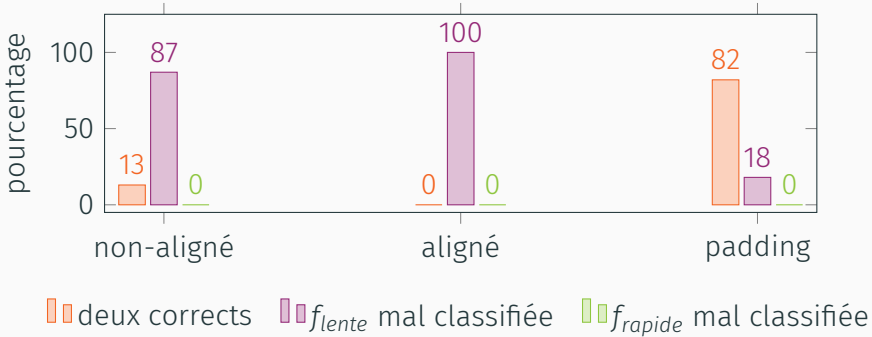



→ padding pour que la fonction lente traverse un front d'horloge de plus que la rapide

# Récupérer une haute résolution : Padding

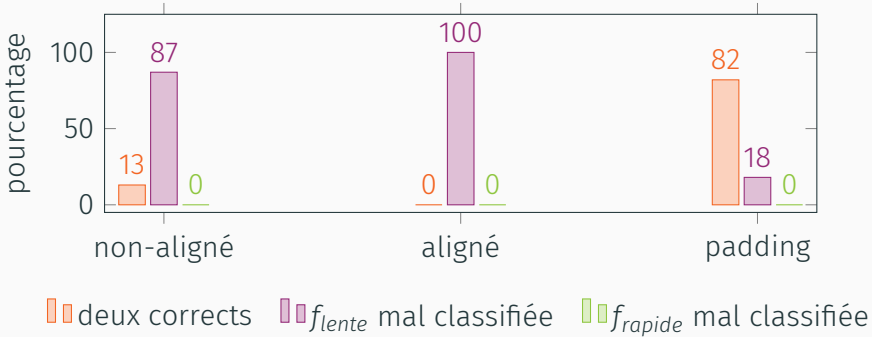


# Récupérer une haute résolution : Padding




- résolution à la nanoseconde près

# Récupérer une haute résolution : Padding




- résolution à la nanoseconde près
- Firefox/Tor : 2 ns, Edge : 10 ns, Chrome : 15 ns

# Construire un timer

- objectif : compteur qui ne bloque pas le thread principal

# Construire un timer

- objectif : compteur qui ne bloque pas le thread principal
- baseline setTimeout : 4 ms (sauf Edge : 2 ms)

# Construire un timer

- objectif : compteur qui ne bloque pas le thread principal
- baseline setTimeout : 4 ms (sauf Edge : 2 ms)
- animation CSS → augmenter la largeur de l'élément le plus vite possible

# Construire un timer

- objectif : compteur qui ne bloque pas le thread principal
- baseline setTimeout : 4 ms (sauf Edge : 2 ms)
- animation CSS → augmenter la largeur de l'élément le plus vite possible
- timestamp = largeur de l'élément

# Construire un timer

- objectif : compteur qui ne bloque pas le thread principal
- baseline setTimeout : 4 ms (sauf Edge : 2 ms)
- animation CSS → augmenter la largeur de l'élément le plus vite possible
- timestamp = largeur de l'élément
- mais : animations limitées à 60 fps → résolution de 16 ms

# Construire un timer : Web worker

- JavaScript peut créer des nouveaux threads appelés web worker

# Construire un timer : Web worker

- JavaScript peut créer des nouveaux threads appelés web worker
- web worker communique en utilisant du passage de message

# Construire un timer : Web worker

- JavaScript peut créer des nouveaux threads appelés web worker
- web worker communique en utilisant du passage de message
- laisser le worker compter et demander le timestamp dans le thread principal

# Construire un timer : Web worker

- JavaScript peut créer des nouveaux threads appelés web worker
- web worker communique en utilisant du passage de message
- laisser le worker compter et demander le timestamp dans le thread principal
- possibilités : `postMessage`, `MessageChannel` or `BroadcastChannel`

# Construire un timer : Web worker

- JavaScript peut créer des nouveaux threads appelés web worker
- web worker communique en utilisant du passage de message
- laisser le worker compter et demander le timestamp dans le thread principal
- possibilités : `postMessage`, `MessageChannel` or `BroadcastChannel`
- résolution : microseconde (même sur Tor et Fuzzyfox)

# Construire un timer : Web worker

- feature expérimentale pour partager des données : `SharedArrayBuffer`

# Construire un timer : Web worker

- feature expérimentale pour partager des données : `SharedArrayBuffer`
- pas d'overhead du passage de message

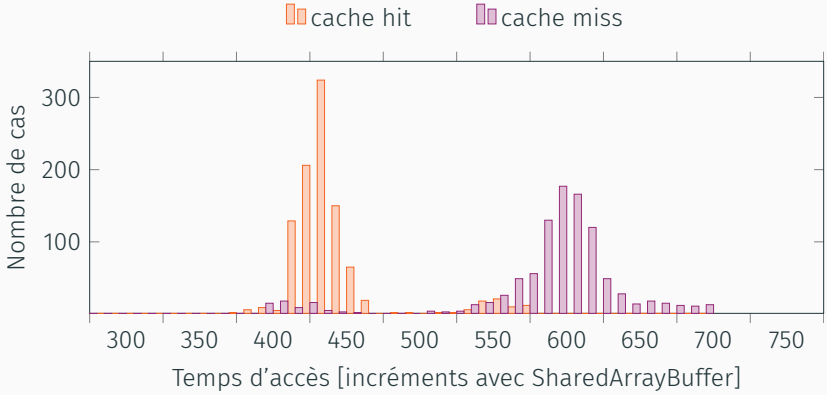
# Construire un timer : Web worker

- feature expérimentale pour partager des données : `SharedArrayBuffer`
- pas d'overhead du passage de message
- un worker dédié pour incrémenter une variable partagée

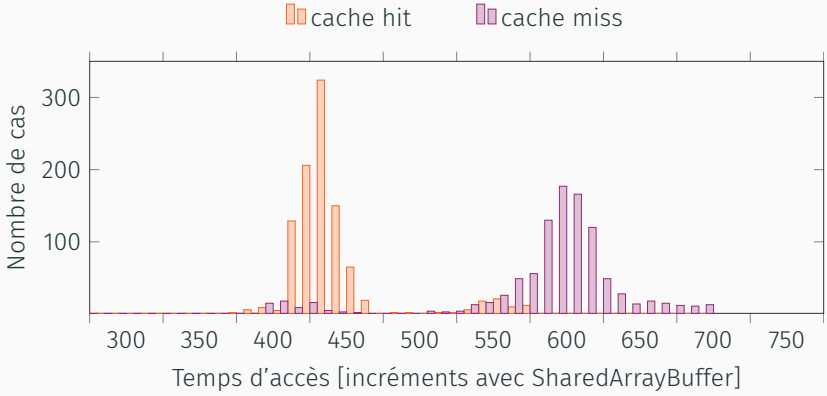
# Construire un timer : Web worker

- feature expérimentale pour partager des données : `SharedArrayBuffer`
- pas d'overhead du passage de message
- un worker dédié pour incrémenter une variable partagée
- Firefox/Fuzzyfox : 2 ns, Chrome : 15 ns

# Construire un timer : Suffisamment bon ?



# Construire un timer : Suffisamment bon ?




→ on peut distinguer des cache hits de cache misses (seulement ≈ 150 cycles de différence) !

# Take-away

# Bonus : On peut faire quoi d'autre avec ça ?

- l'idée n'est pas nouvelle : Wray (1992)
- on l'a aussi exploitée dans d'autres contextes
  - sur ARM
  - dans une enclave SGX

J. C. Wray. "An analysis of covert timing channels". In: *Journal of Computer Security* 1.3-4 (1992), pp. 219–232.
M. Lipp et al. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016.
M. Schwarz et al. "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: *DIMVA'17*. 2017.

# Canaux cachés sur la DRAM en JavaScript !

# Setup

- émetteur : application native dans une VM

# Setup

- émetteur : application native dans une VM
- récepteur : JavaScript dans une page web sur le host

# Setup

- émetteur : application native dans une VM
- récepteur : JavaScript dans une page web sur le host
- émetteur et récepteur choisissent la même bank

# Setup

- émetteur : application native dans une VM
- récepteur : JavaScript dans une page web sur le host
- émetteur et récepteur choisissent la même bank
- émetteur et récepteur choisissent un row différent dans cette bank

# Setup

- émetteur : application native dans une VM
- récepteur : JavaScript dans une page web sur le host
- émetteur et récepteur choisissent la même bank
- émetteur et récepteur choisissent un row différent dans cette bank
- émetteur transmet 0 en ne faisant rien et 1 en causant un row conflict

# Setup

- émetteur : application native dans une VM
- récepteur : JavaScript dans une page web sur le host
- émetteur et récepteur choisissent la même bank
- émetteur et récepteur choisissent un row différent dans cette bank
- émetteur transmet 0 en ne faisant rien et 1 en causant un row conflict
- récepteur mesure le temps d'accès pour son row : rapide $\rightarrow$ 0, lent $\rightarrow$ 1

# Envoyer des paquets


- communication basée sur des paquets de 11 bits, avec 5 bits de données

# Envoyer des paquets


- communication basée sur des paquets de 11 bits, avec 5 bits de données
- paquet commence avec un préambule de 2 bits

# Envoyer des paquets


- communication basée sur des paquets de 11 bits, avec 5 bits de données
- paquet commence avec un préambule de 2 bits
- intégrité des données vérifiée par un code de détection d'erreurs

# Envoyer des paquets


- communication basée sur des paquets de 11 bits, avec 5 bits de données
- paquet commence avec un préambule de 2 bits
- intégrité des données vérifiée par un code de détection d'erreurs
- bit de sequence indique si retransmission ou nouveau paquet

# Évaluation

- transmission $\approx$ 11 bits/s

# Évaluation

- transmission $\approx$ 11 bits/s
- peut être améliorée en utilisant

# Évaluation

- transmission $\approx$ 11 bits/s
- peut être améliorée en utilisant
  - moins de retransmission

# Évaluation

- transmission $\approx$ 11 bits/s
- peut être améliorée en utilisant
  - moins de retransmission
  - des codes correcteurs d'erreurs

# Évaluation

- transmission $\approx$ 11 bits/s
- peut être améliorée en utilisant
  - moins de retransmission
  - des codes correcteurs d'erreurs
  - multithreading $\rightarrow$ plusieurs banks en parallèle

# Évaluation

- transmission $\approx$ 11 bits/s
- peut être améliorée en utilisant
  - moins de retransmission
  - des codes correcteurs d'erreurs
  - multithreading $\rightarrow$ plusieurs banks en parallèle
- code natif : 596 kbit/s entre CPUs et entre VMs

# Conclusion

# Conclusion

- fuites d'informations dues au matériel

# Conclusion

- fuites d'informations dues au matériel
- vulnérabilités exploitables au niveau d'un navigateur web

# Conclusion

- fuites d'informations dues au matériel
- vulnérabilités exploitables au niveau d'un navigateur web
- faire tourner un script arbitraire en JavaScript permet de construire des timers haute résolution

# Conclusion

- fuites d'informations dues au matériel
- vulnérabilités exploitables au niveau d'un navigateur web
- faire tourner un script arbitraire en JavaScript permet de construire des timers haute résolution
- contre-mesures compliquées sans réduire les fonctionnalités ou la performance

# Merci SSTIC !

Contact

clementine@cmaurice.fr
@BloodyTangerine

# De bas en haut : attaques sur la microarchitecture depuis un navigateur web

Clémentine Maurice, Graz University of Technology

8 Juin 2017—SSTIC, Rennes, France