Jumping Abstraction Layers: Microarchitectural Attacks in JavaScript

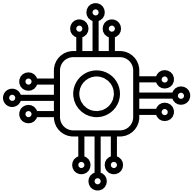
Daniel Gruss

September 18, 2019

Graz University of Technology

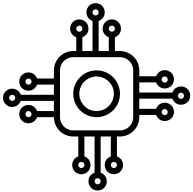
National Geographic


side channel
= obtaining meta-data and
deriving secrets from it
CHANGE MY MIND



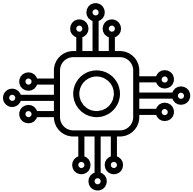
Microarchitecture...

- not architectural state



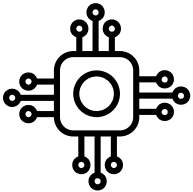
Microarchitecture...

- not architectural state
- not visible to software



Microarchitecture...

- not architectural state
- not visible to software
- hardware specific



Microarchitecture...

- not architectural state
- not visible to software
- hardware specific
- changes with generations

1337 4242

FOOD CACHE

Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

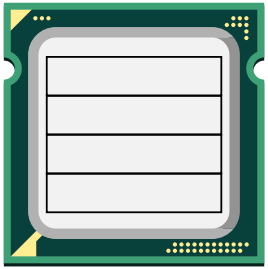
Can store **ALL** kinds of food.

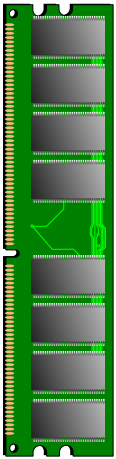
ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

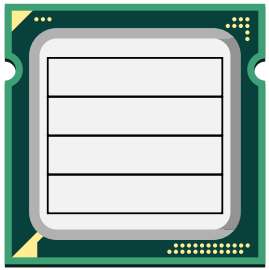
```
printf("%d", i);
printf("%d", i);
```

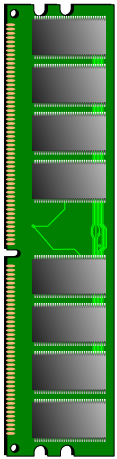




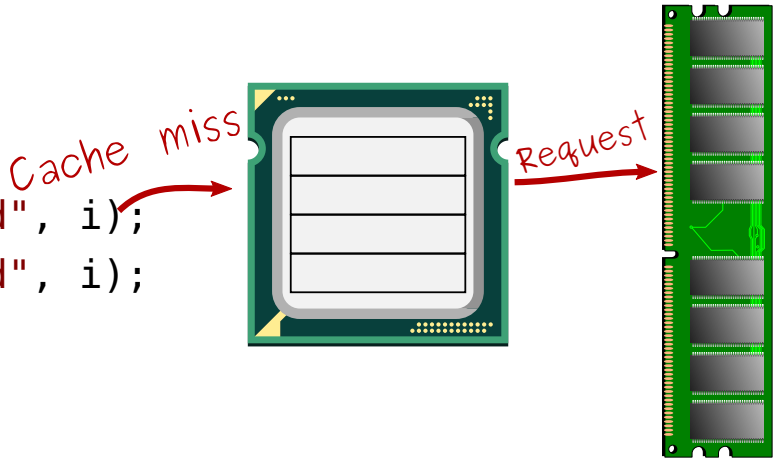
```
printf("%d", i);
printf("%d", i);
```

Cache miss

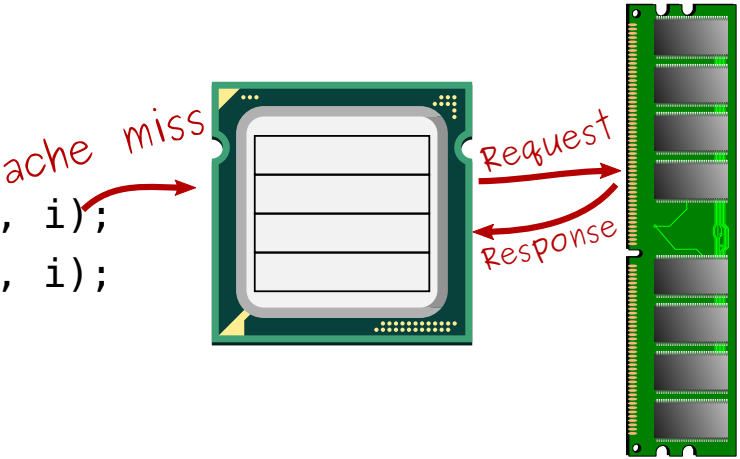




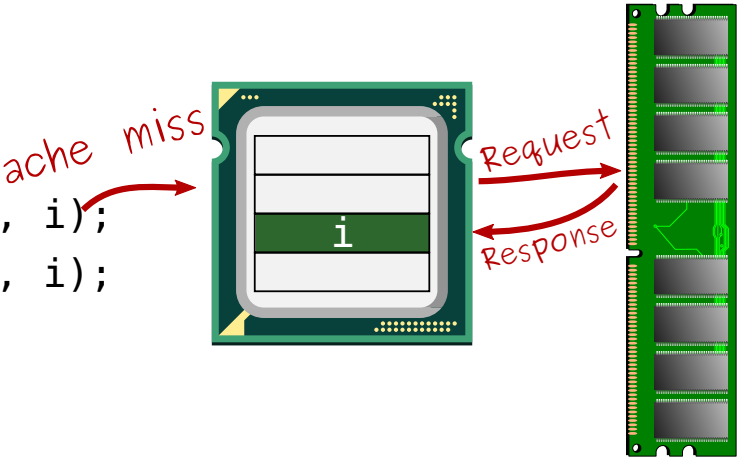
```
printf("%d", i);
printf("%d", i);
```

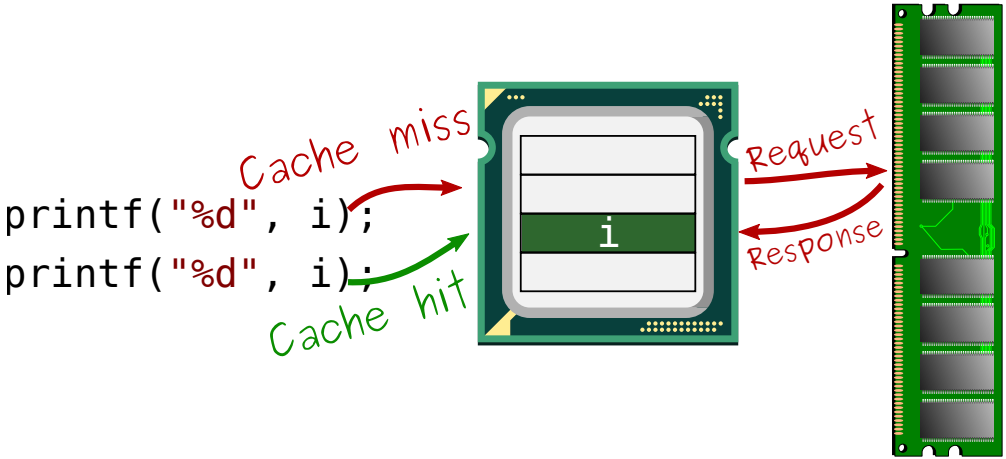


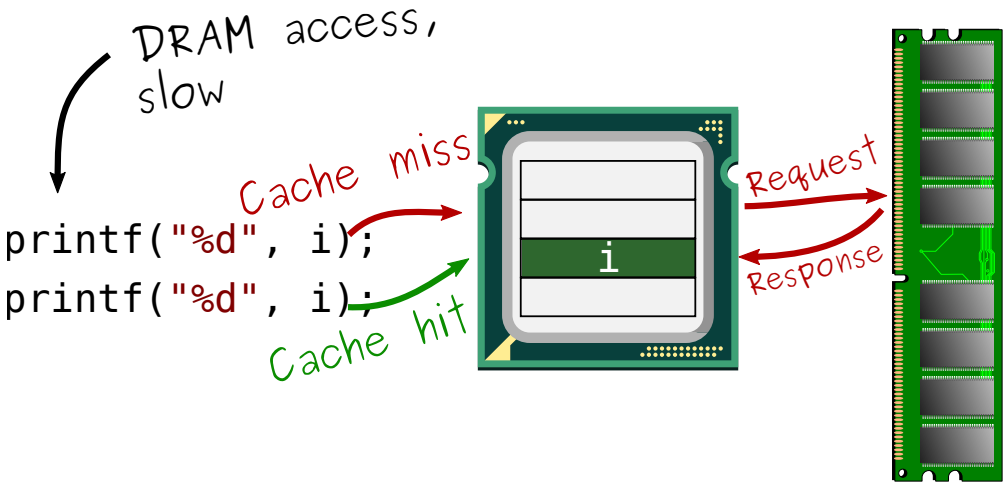
```
printf("%d", i);
printf("%d", i);
```

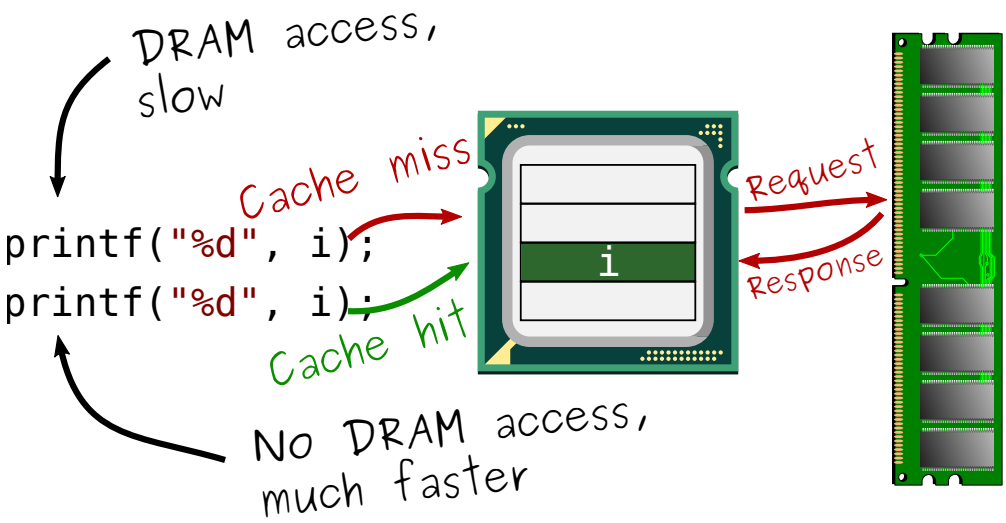


```
printf("%d", i);
printf("%d", i);
```




printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response


DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

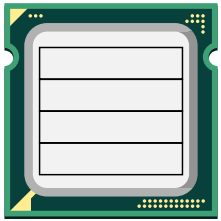

DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access,
much faster

Shared Memory

ATTACKER

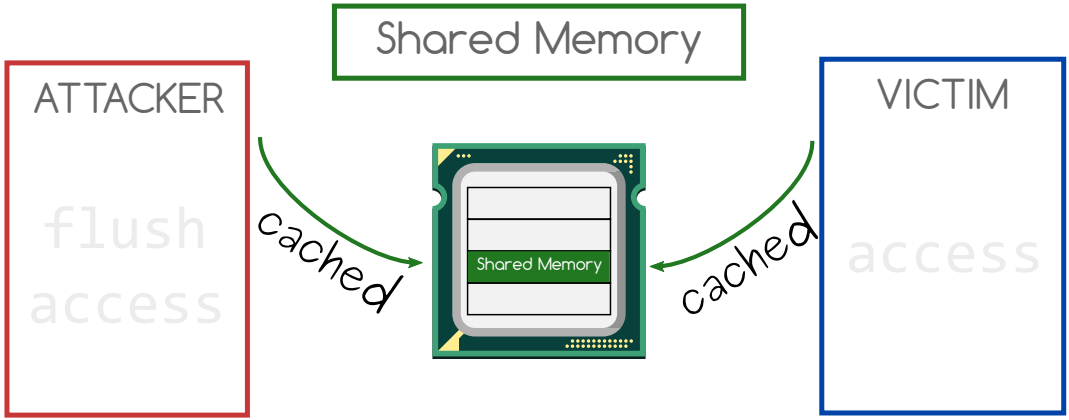
flush

access



VICTIM

access

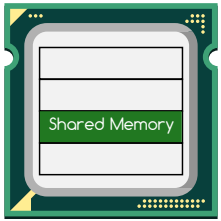

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

Shared Memory

ATTACKER

flush

access



VICTIM

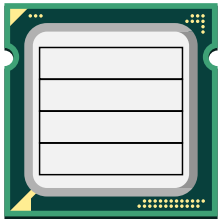
access

Shared Memory

ATTACKER

flush

access



VICTIM

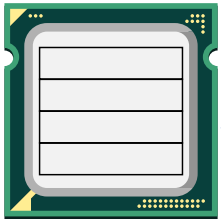
access

Shared Memory

ATTACKER

flush

access



VICTIM

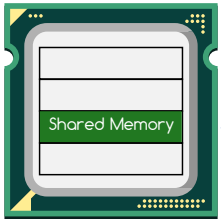
access

Shared Memory

ATTACKER

flush

access



VICTIM

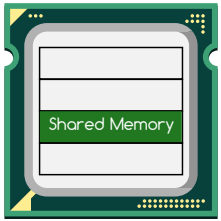
access

Shared Memory

ATTACKER

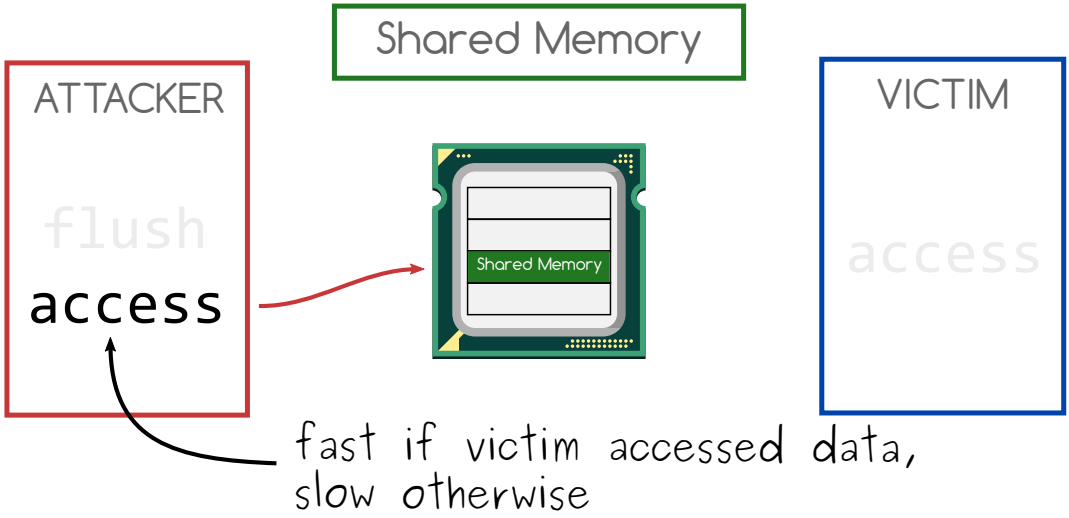
flush

access

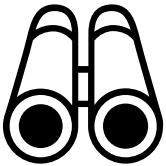


VICTIM

access

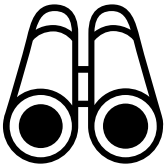

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
fast if victim accessed data,
slow otherwise


TIMING...
IT'S ALL ABOUT
imgflip.com



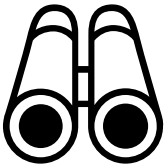
First side-channel attack in JavaScript

- Stone et al. (2013): HTML5 pixel perfect attacks



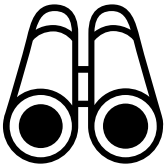
First side-channel attack in JavaScript

- Stone et al. (2013): HTML5 pixel perfect attacks
- Use high-resolution timer



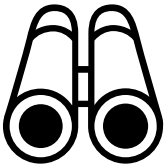
First side-channel attack in JavaScript

- Stone et al. (2013): HTML5 pixel perfect attacks
- Use high-resolution timer
- Timing redraw events (visited, ...)



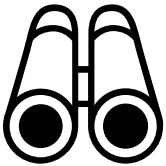
First side-channel attack in JavaScript

- Stone et al. (2013): HTML5 pixel perfect attacks
- Use high-resolution timer
- Timing redraw events (visited, ...)
- SVG filter timing for pixels (known since 2011)



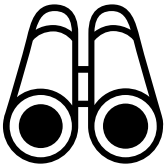
First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox



First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox
- Timing of memory accesses



First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox
- Timing of memory accesses
- Data cached or not

FANTASTIC TIMERS AND WHERE TO FIND THEM

HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT

- We need a high-resolution timer

- We need a high-resolution timer
- Native: `rdtsc`

- We need a high-resolution timer
- Native: `rdtsc`
- JavaScript: `performance.now()`

- We need a high-resolution timer
- Native: `rdtsc`
- JavaScript: `performance.now()`

performance.now()

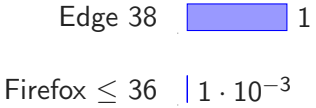
[...] represent times as floating-point numbers with up to microsecond precision.

— Mozilla Developer Network

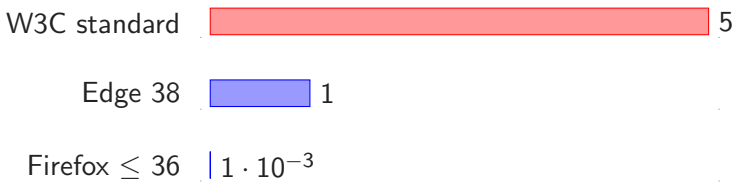
...up to microsecond precision?

Firefox ≤ 36	$1 \cdot 10^{-3}$

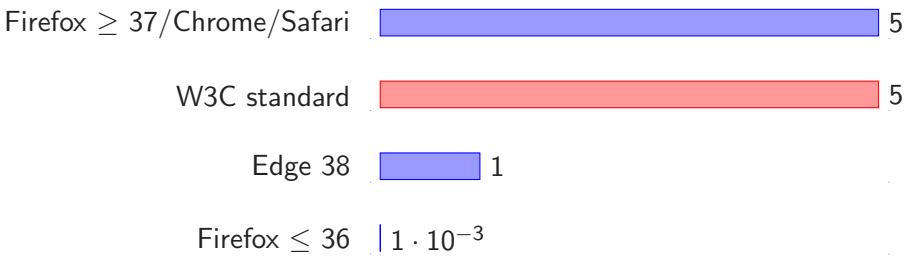
...up to microsecond precision?



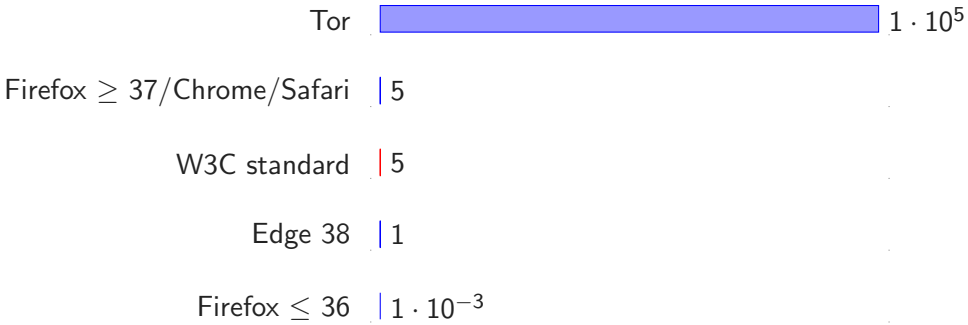
...up to microsecond precision?



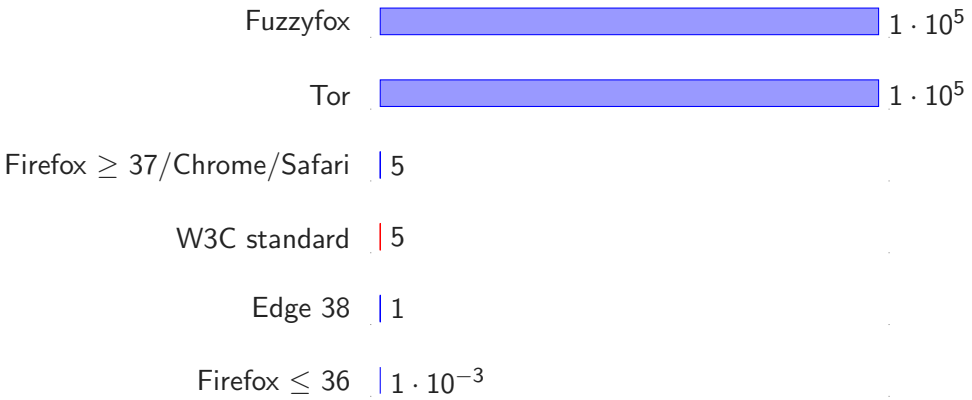
...up to microsecond precision?



...up to microsecond precision?



...up to microsecond precision?



We need a higher resolution

- Current precision can't measure cycle differences

We need a higher resolution

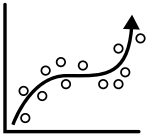
- Current precision can't measure cycle differences
- Two options

We need a higher resolution

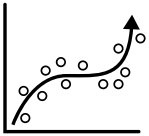
- Current precision can't measure cycle differences
- Two options
- Recover a higher resolution

We need a higher resolution

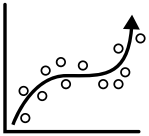
- Current precision can't measure cycle differences
- Two options
- Recover a higher resolution
- Build our own high-resolution timer



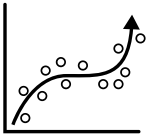
- Measure how often we can increment a variable between two timer ticks



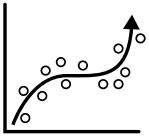
- Measure how often we can increment a variable between two timer ticks
- Average number of increments is the interpolation step



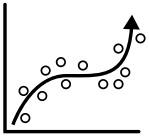
- Measure how often we can increment a variable between two timer ticks
- Average number of increments is the interpolation step
- To measure with high resolution:



- Measure how often we can increment a variable between two timer ticks
- Average number of increments is the interpolation step
- To measure with high resolution:
  - Start measurement at clock edge



- Measure how often we can increment a variable between two timer ticks
- Average number of increments is the interpolation step
- To measure with high resolution:
  - Start measurement at clock edge
  - Increment a variable until next clock edge

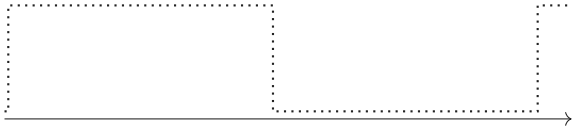


- Measure how often we can increment a variable between two timer ticks
- Average number of increments is the interpolation step
- To measure with high resolution:
  - Start measurement at clock edge
  - Increment a variable until next clock edge
- Highly accurate: 500 ns (Firefox/Chrome), 15 µs (Tor)

- We can get a higher resolution for a classifier only

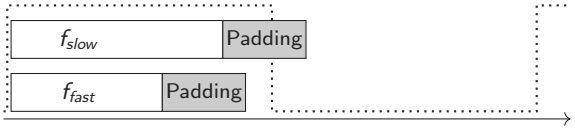
Recovering resolution - Edge thresholding

- We can get a higher resolution for a classifier only
- Often sufficient to see which of two functions takes longer



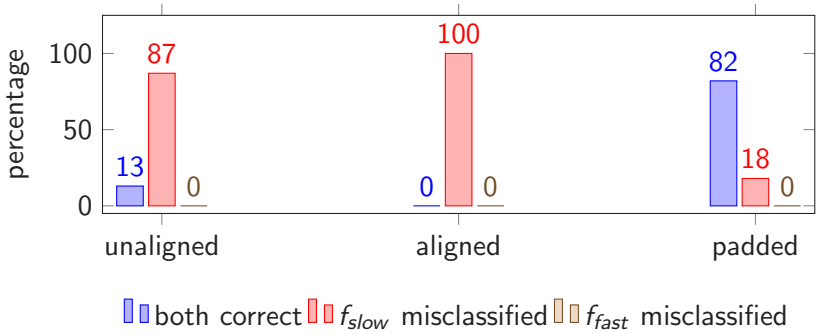
- We can get a higher resolution for a classifier only
- Often sufficient to see which of two functions takes longer

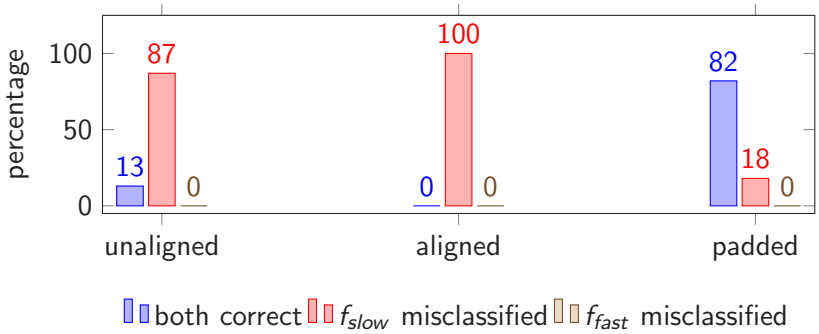
- We can get a higher resolution for a classifier only
- Often sufficient to see which of two functions takes longer



- Edge thresholding: apply padding such that the slow function crosses one more clock edge than the fast function.

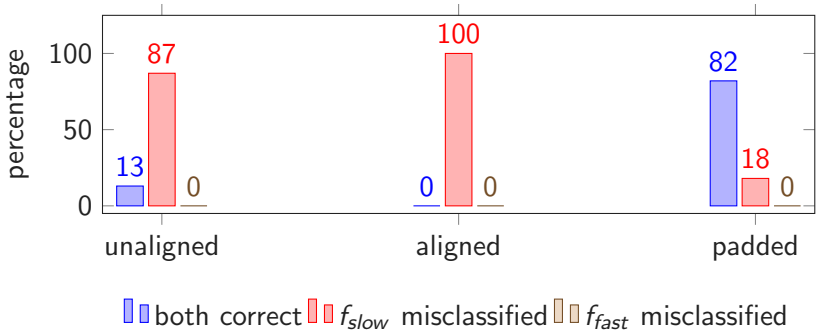
Recovering resolution - Edge thresholding





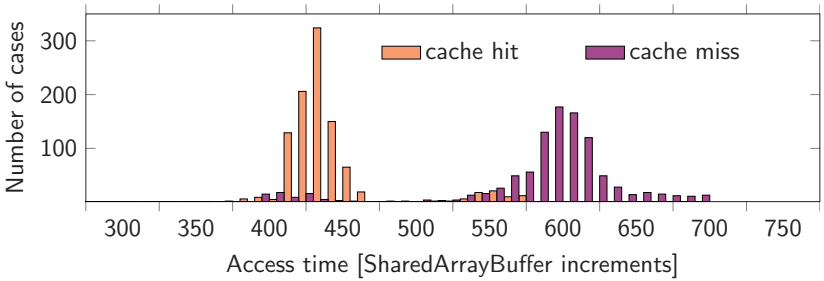
- Yields nanosecond resolution

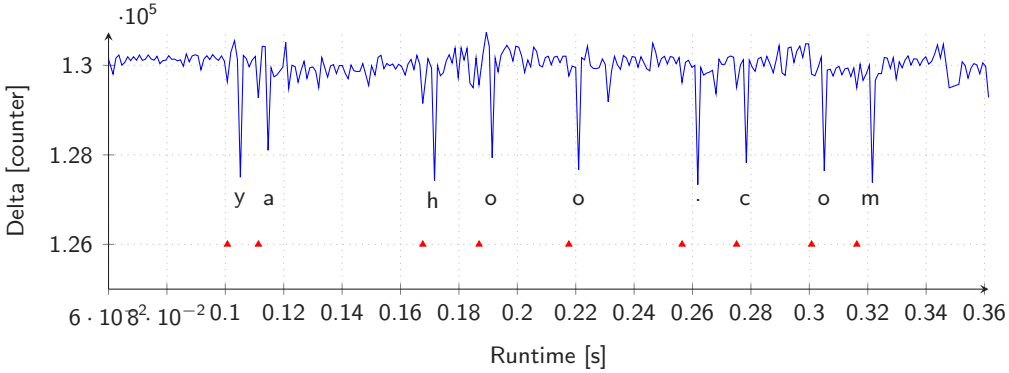
Recovering resolution - Edge thresholding



- Yields nanosecond resolution
- Firefox/Tor (2 ns), Edge (10 ns), Chrome (15 ns)

Timer evaluation




$\cdot 10^5$
1.3
1.28
1.26
Delta [counter]
y a h o o · c o m
$6 \cdot 10^{-2}$ $8 \cdot 10^{-2}$ 0.1 0.12 0.14 0.16 0.18 0.2 0.22 0.24 0.26 0.28 0.3 0.32 0.34 0.36
Runtime [s]

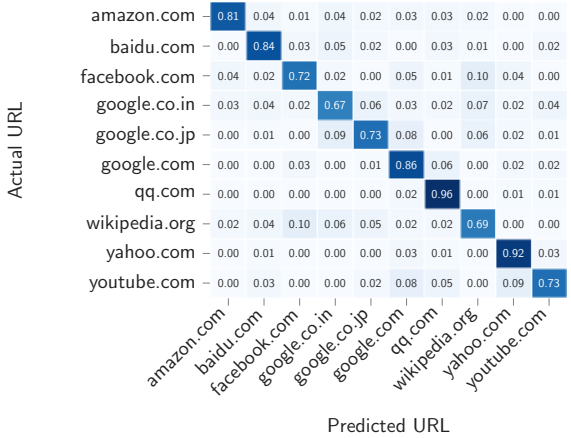


Figure 1: Confusion matrix for URL input.

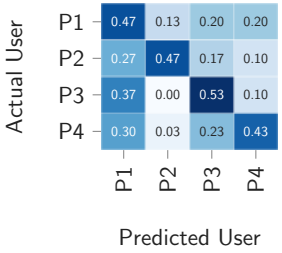


Figure 2: Confusion matrix for input by different users.

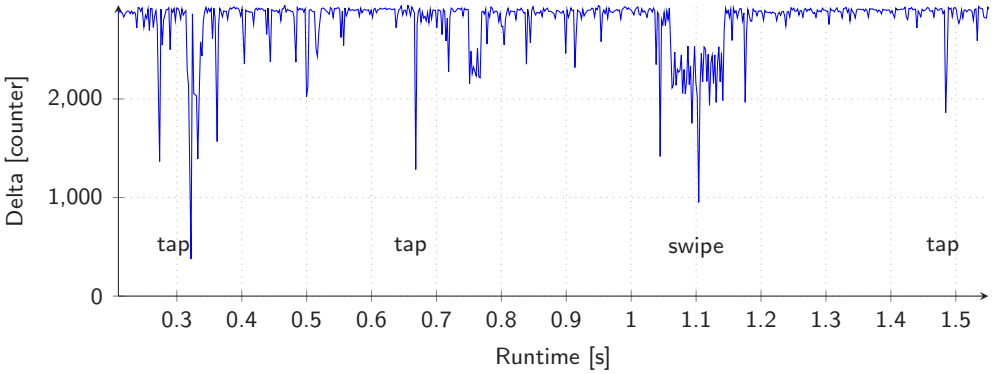


Figure 3: Keystroke timing on Google Nexus 5.

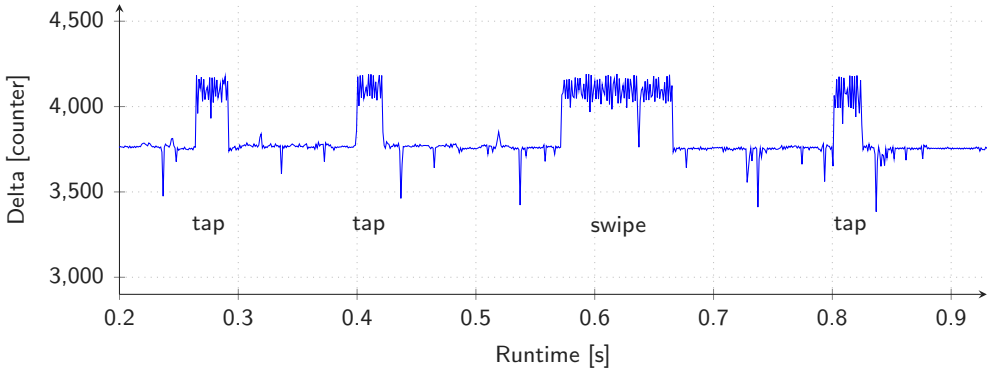


Figure 4: Keystroke timing on Xiaomi Redmi Note 3.

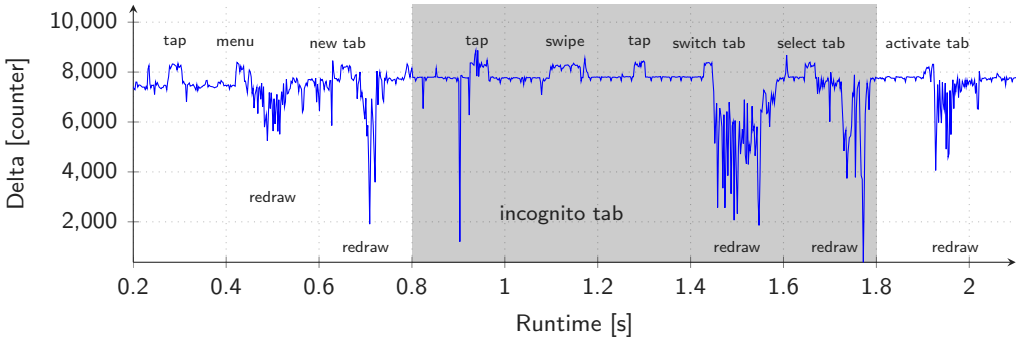


Figure 5: Chrome on Xiaomi Redmi Note 3.

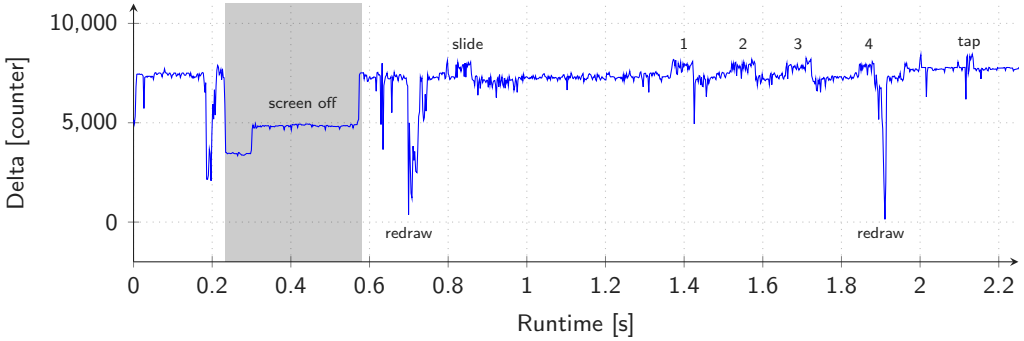


Figure 6: Firefox on Xiaomi Redmi Note 3.

- Timers were always the main focus

Attack requirements

- Timers were always the main focus
- Reducing timer resolution is not sufficient

Attack requirements

- Timers were always the main focus
- Reducing timer resolution is not sufficient
- Timers can (always) be built

Attack requirements

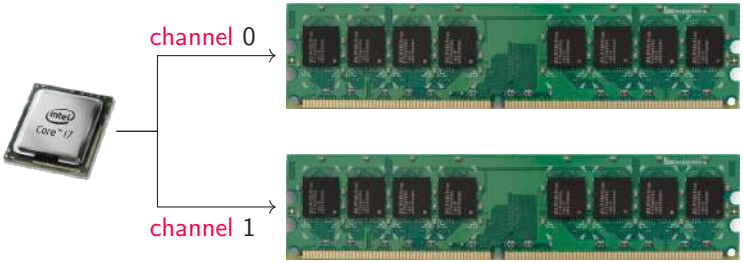
- Timers were always the main focus
- Reducing timer resolution is not sufficient
- Timers can (always) be built
- Some attacks do not require timers at all

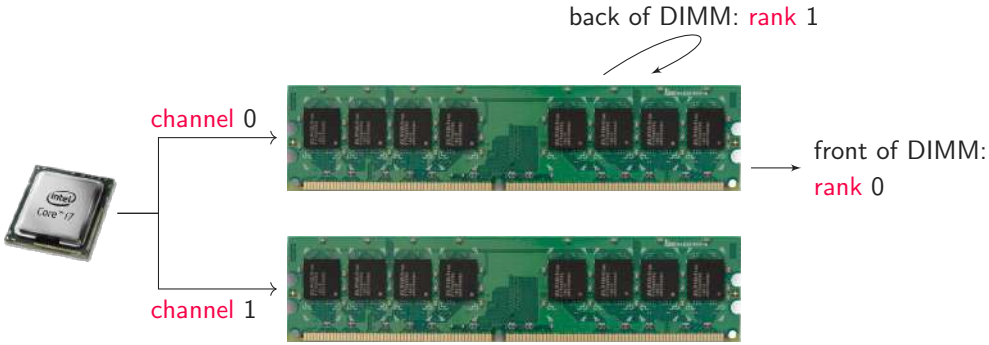
Attack requirements

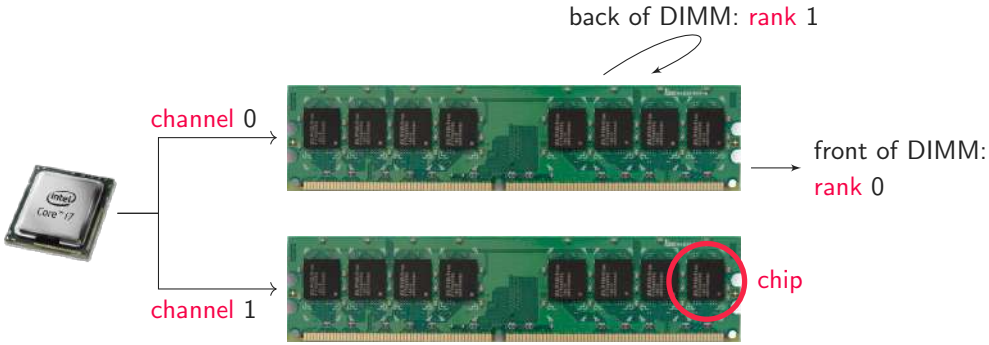
- Timers were always the main focus
- Reducing timer resolution is not sufficient
- Timers can (always) be built
- Some attacks do not require timers at all
- Important to understand requirements before designing countermeasures


SIDE CHANNELS?
I WOULD NEED TO BREAK SOME ISOLATION

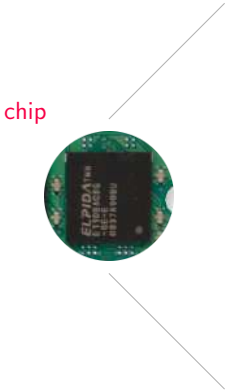
DRAM organization

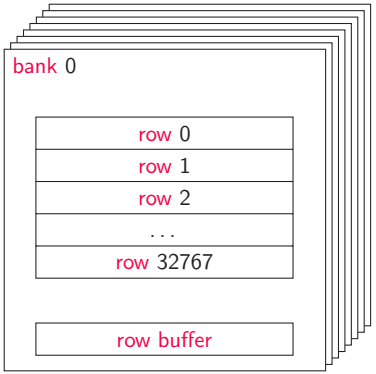



back of DIMM: rank 1
channel 0
front of DIMM:
rank 0
channel 1


back of DIMM: rank 1
channel 0
front of DIMM:
rank 0
chip
channel 1

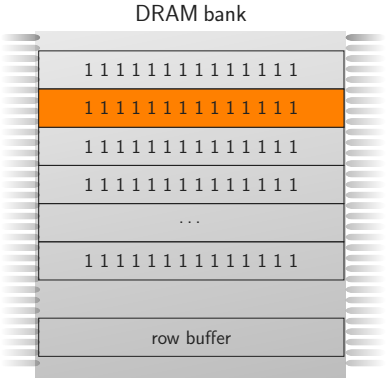
DRAM organization



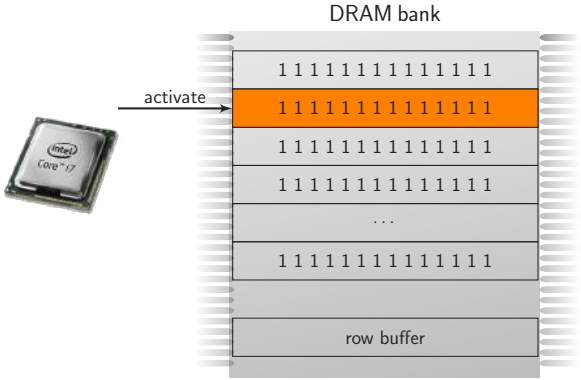


- bits in cells in rows
- access: activate row, copy to row buffer

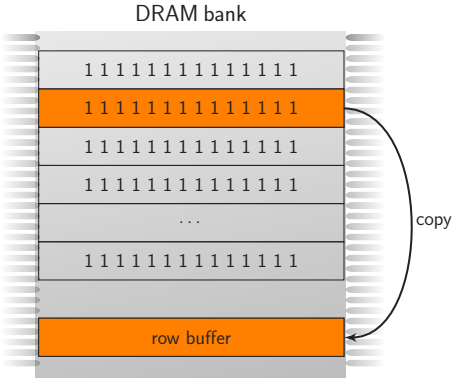
How reading from DRAM works



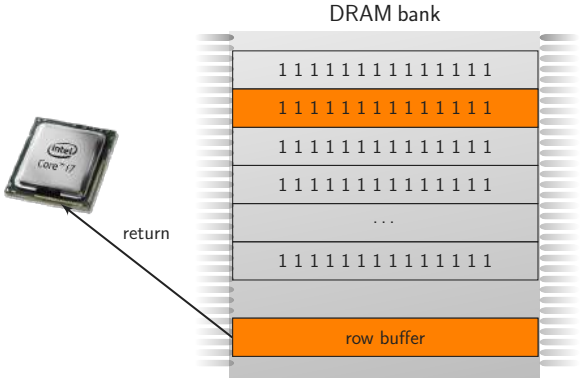
How reading from DRAM works



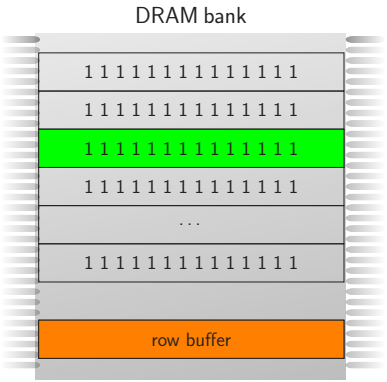
How reading from DRAM works



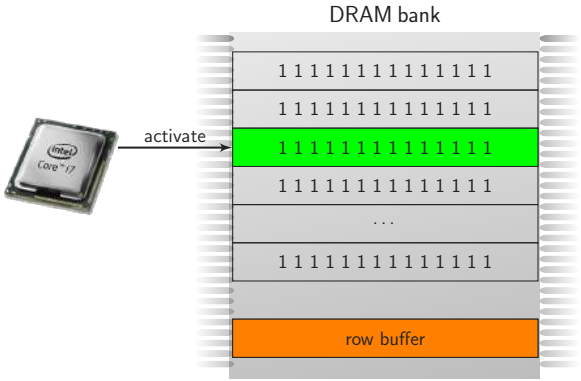
How reading from DRAM works



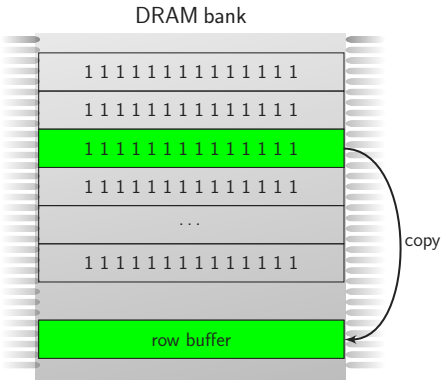
How reading from DRAM works



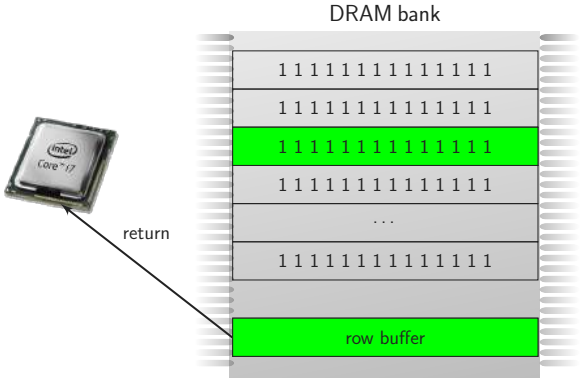
How reading from DRAM works



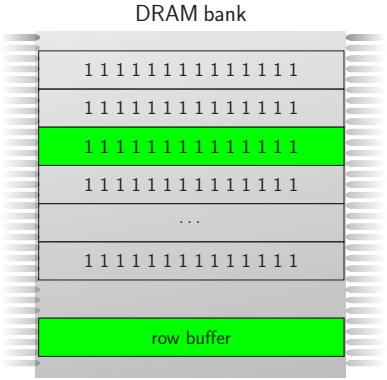
How reading from DRAM works



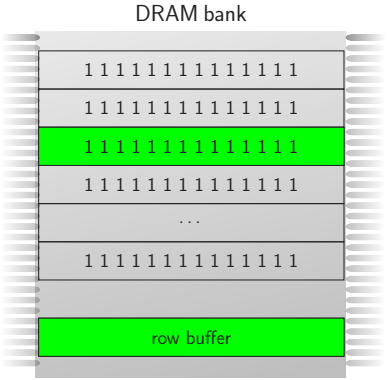
How reading from DRAM works



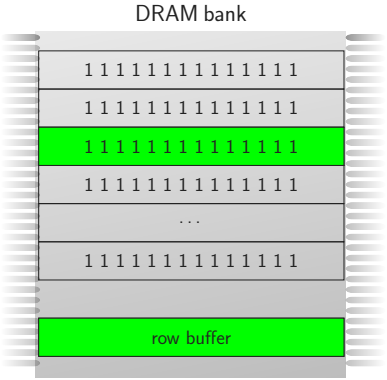
How reading from DRAM works



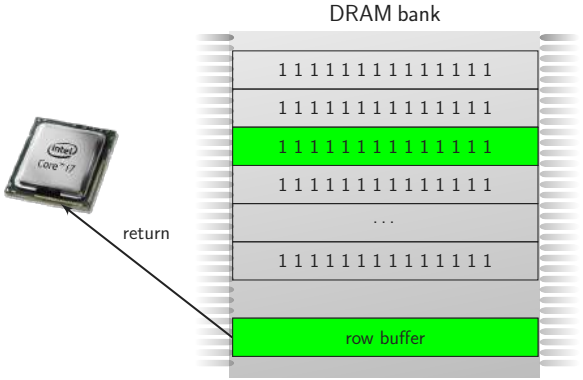
How reading from DRAM works



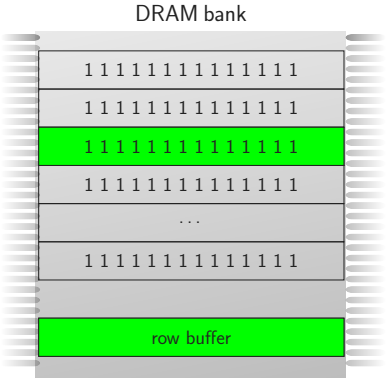
How reading from DRAM works



How reading from DRAM works

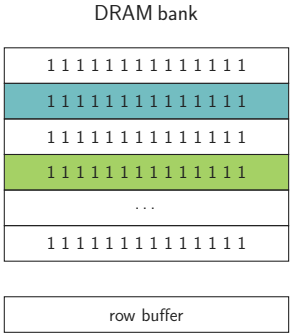


How reading from DRAM works

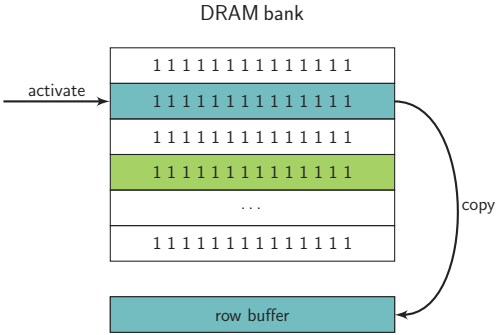



ONE DOES NOT SIMPLY
ROWHAMMER IN JAVASCRIPT

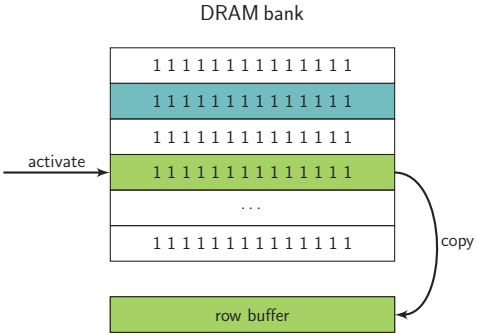
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



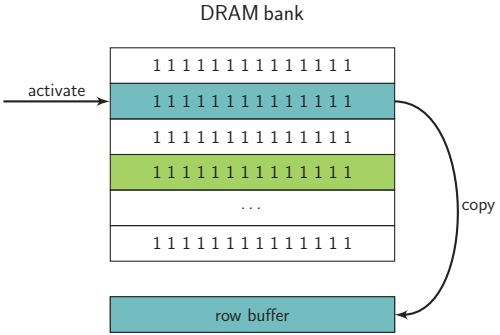
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



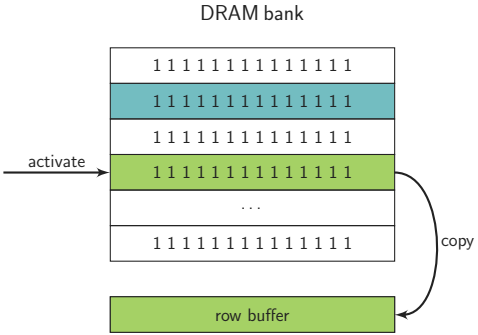
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



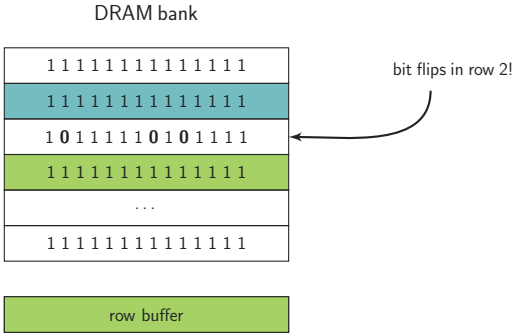
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



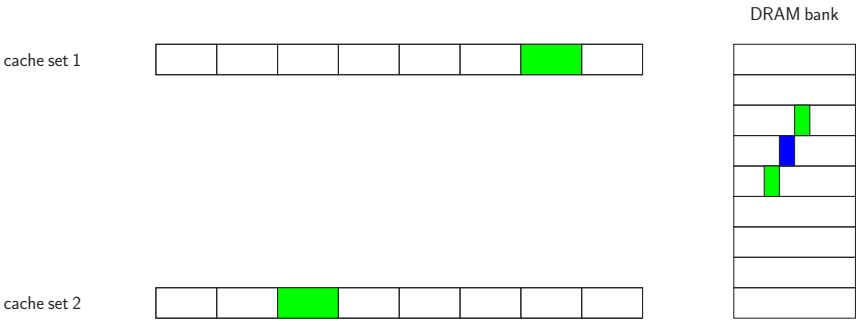
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



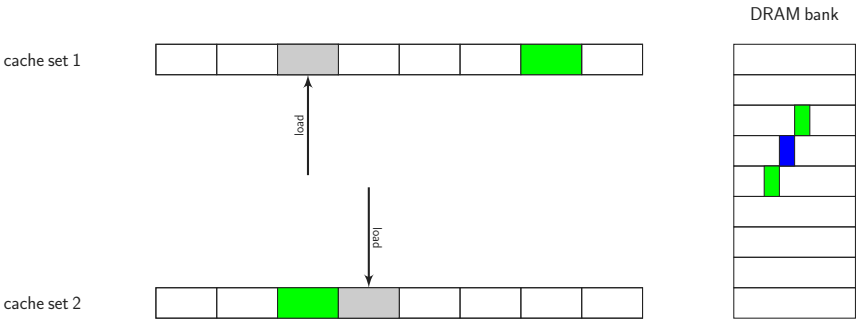
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



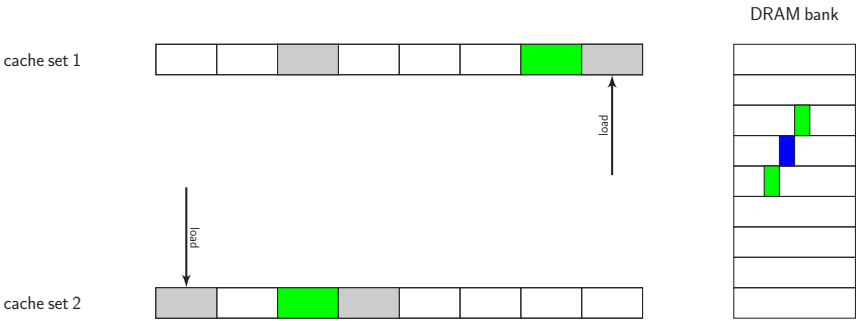
Hammering with cache eviction

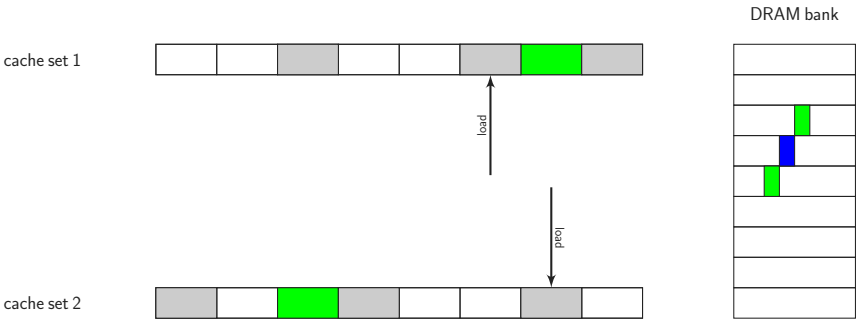


Hammering with cache eviction

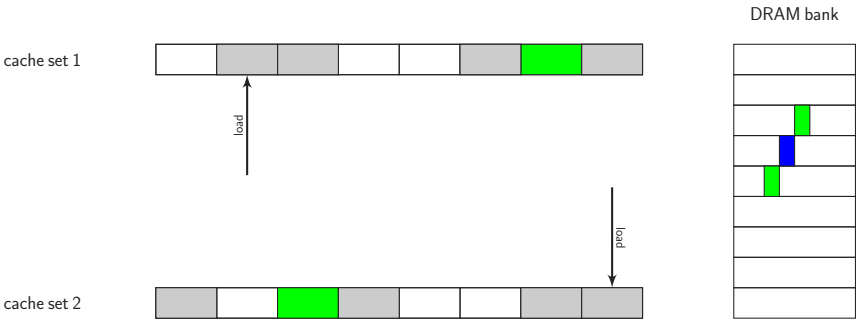


Hammering with cache eviction

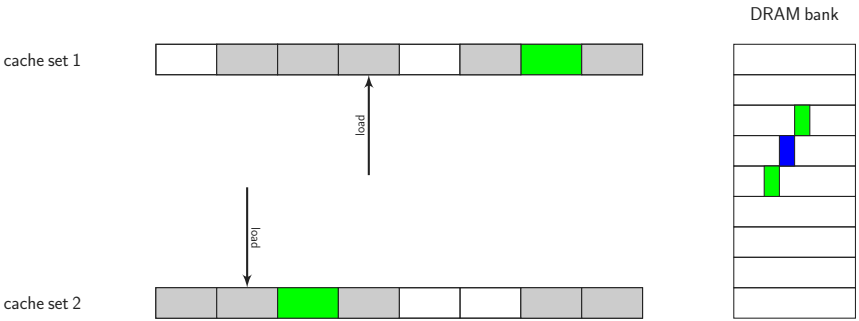


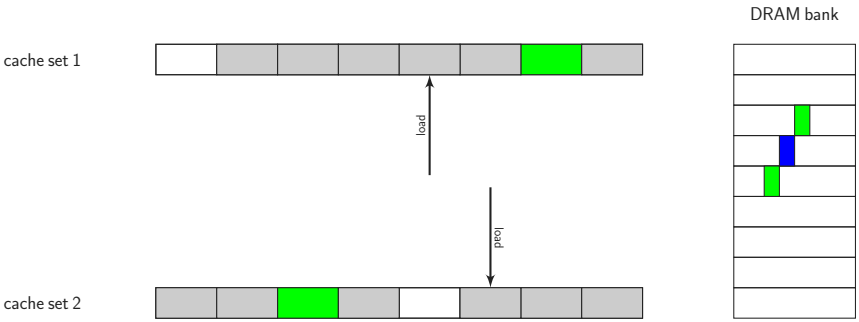

DRAM bank
cache set 1
load
load
cache set 2

Hammering with cache eviction

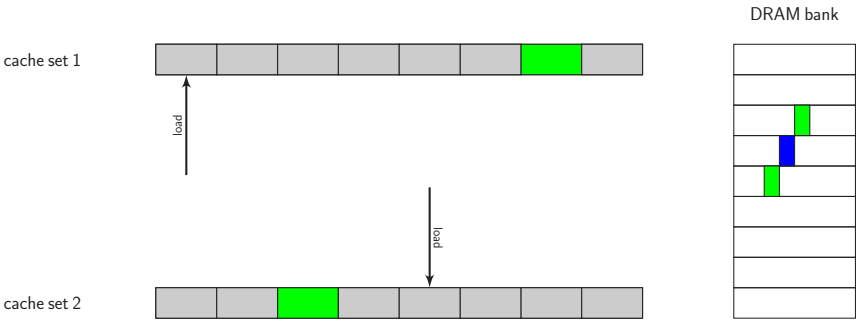


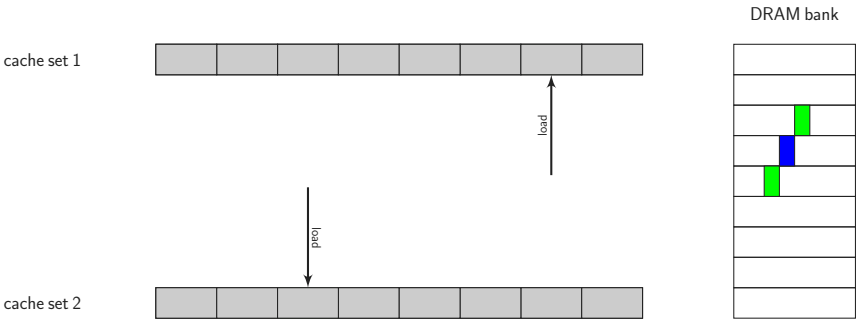
Hammering with cache eviction



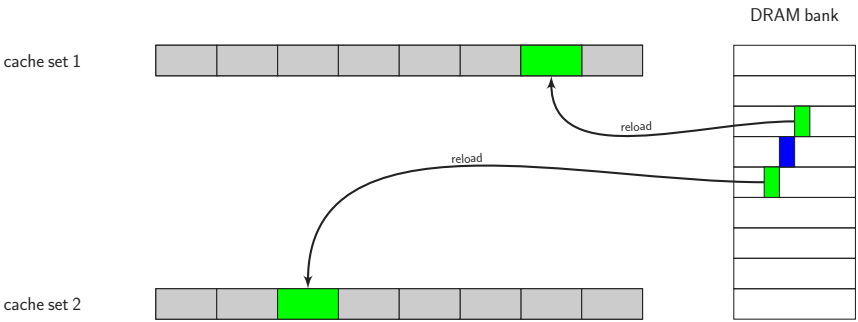

DRAM bank
cache set 1
load
load
cache set 2

Hammering with cache eviction

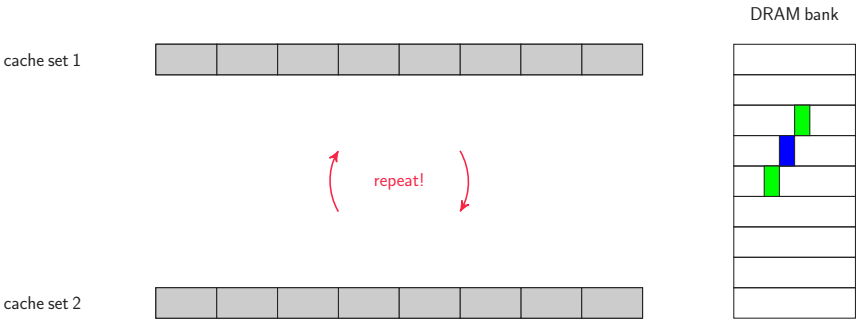



DRAM bank
cache set 1
load
load
cache set 2

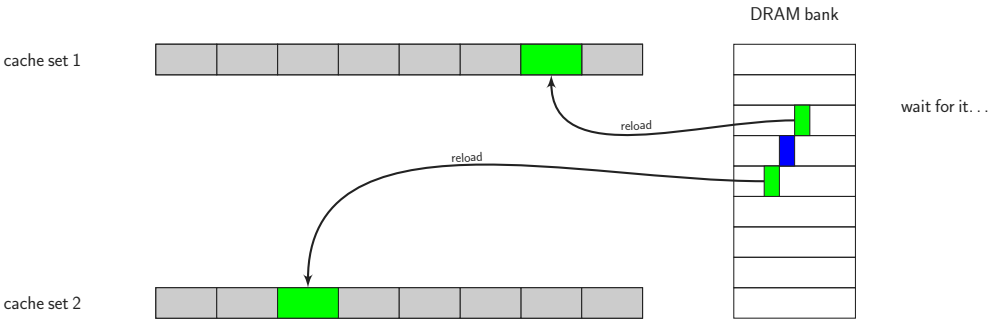
Hammering with cache eviction



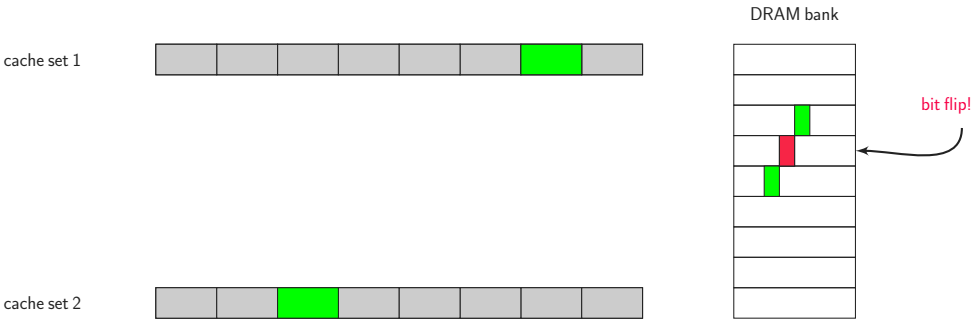
Hammering with cache eviction



Hammering with cache eviction



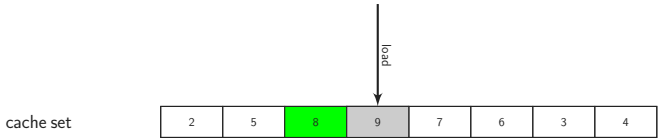
Hammering with cache eviction



"LRU eviction" memory accesses: n accesses for an n-way cache

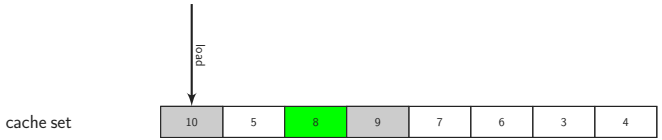
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



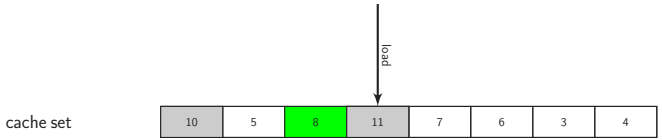
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



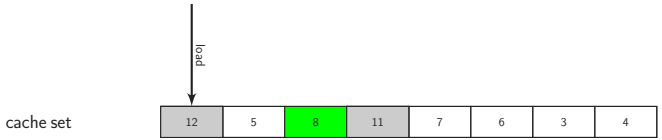
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



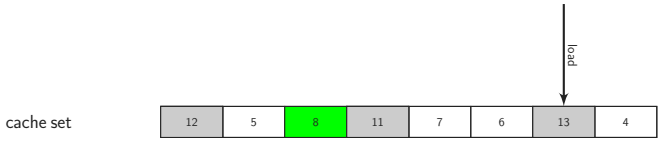
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



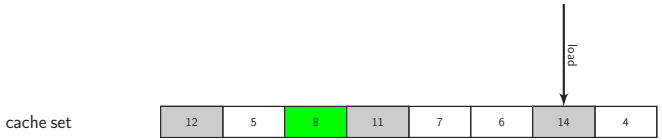
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



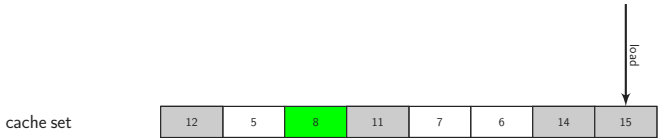
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



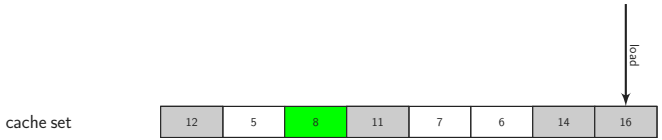
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache



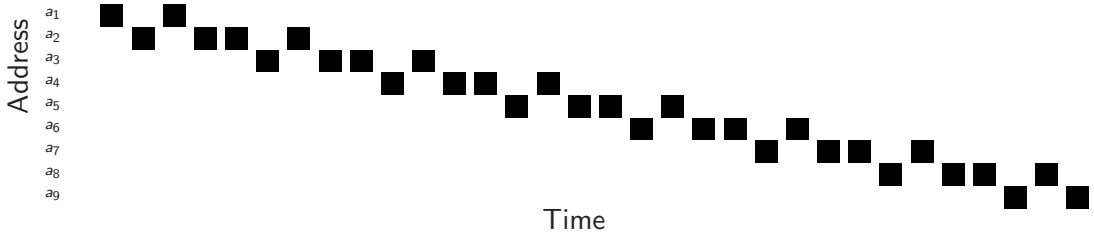
- no LRU replacement

"LRU eviction" memory accesses: n accesses for an n-way cache

- no LRU replacement
- only 75% success rate on Haswell

"LRU eviction" memory accesses: n accesses for an n-way cache

- no LRU replacement
- only 75% success rate on Haswell
- more accesses → higher success rate, but too slow



$\rightarrow$ fast and effective on Haswell: eviction rate >99.97%

- represent accesses as a sequence: $1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, \ldots$
- what can improve eviction rates?

Cache eviction strategy: New representation

- represent accesses as a sequence: $1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, \ldots$
- what can improve eviction rates?

$\rightarrow$ adding more *unique* addresses

$\rightarrow$ *more* accesses to the same addresses

- indistinguishable $\rightarrow$ *balanced* number of accesses

Cache eviction strategy: Notation (1)

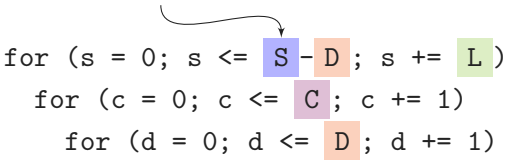
Write eviction strategies as: *P-C-D-L-S*

```
for (s = 0; s <= S-D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

Cache eviction strategy: Notation (1)

Write eviction strategies as: *P-C-D-L-S*

S: total number of different addresses
(= set size)

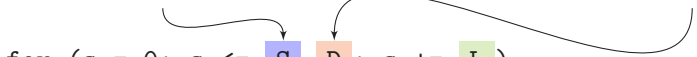


Cache eviction strategy: Notation (1)

Write eviction strategies as: *P-C-D-L-S*

S: total number of different addresses (= set size)

D: different addresses per inner access loop



```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

Cache eviction strategy: Notation (1)

Write eviction strategies as: *P-C-D-L-S*

S: total number of different addresses (= set size)

D: different addresses per inner access loop

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

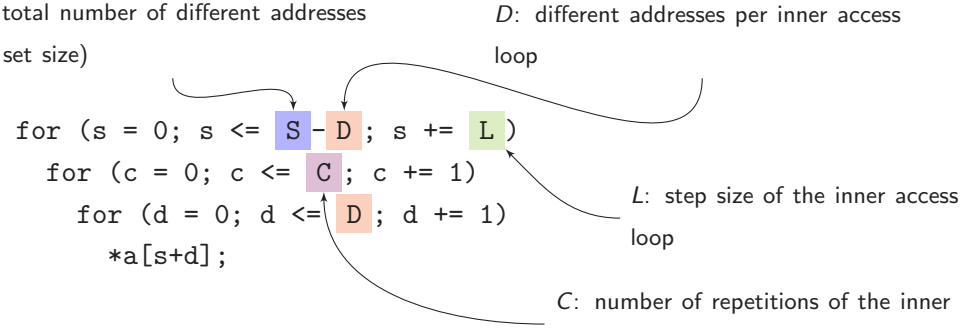
L: step size of the inner access loop

Cache eviction strategy: Notation (1)

Write eviction strategies as: *P-C-D-L-S*

S: total number of different addresses (= set size)

D: different addresses per inner access loop



Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S-D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

Cache eviction strategy: Notation (2)

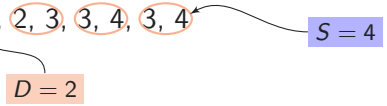
```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

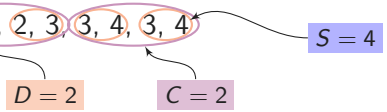
- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4



Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

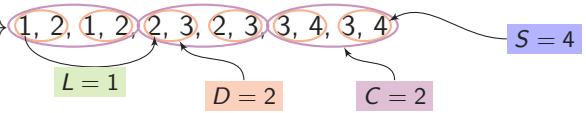
- *P*-2-2-1-4 → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4



Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

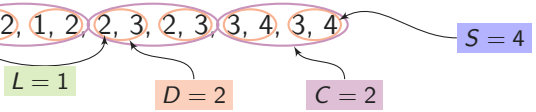
- P-2-2-1-4 → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4



Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

- *P*-2-2-1-4 → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4



- *P*-1-1-1-4 → 1, 2, 3, 4 → LRU eviction with set size 4

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17		
P-1-1-1-20	20		

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	
P-1-1-1-20	20	99.82% ✓	

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34		

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64		

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	

[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	180 ns ✓

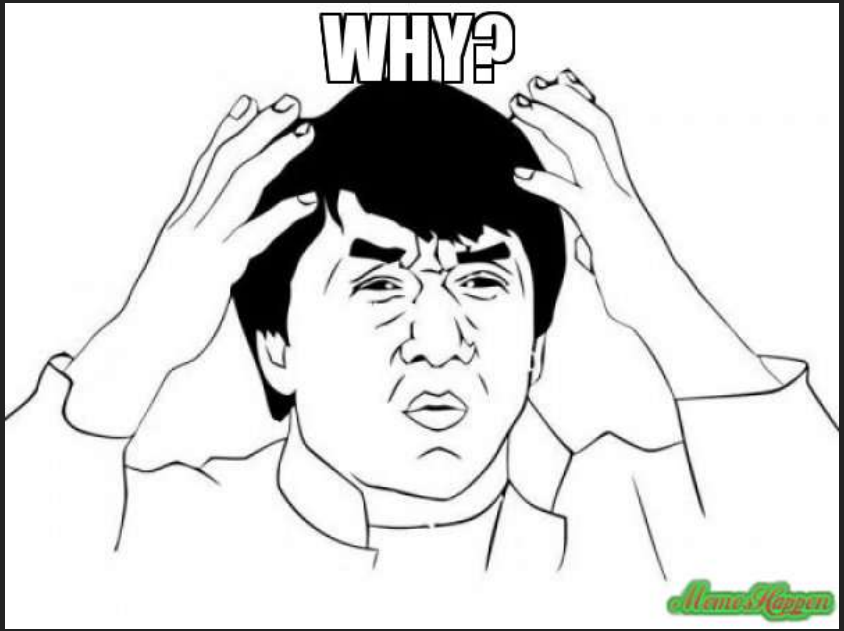
[1] Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	180 ns ✓

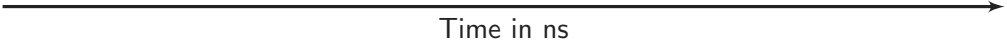
→ more accesses, smaller execution time?

[1] Executed in a loop, on a Haswell with a 16-way last-level cache


WHY?
MemeHappen

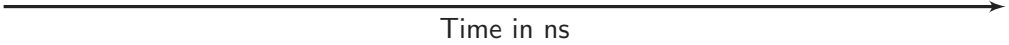
P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)



P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

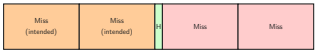
Time in ns

P-1-1-1-17 (17 accesses, 307ns)

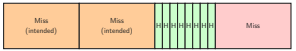
P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)



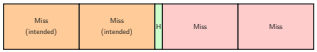
P-2-1-1-17 (34 accesses, 191ns)



Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

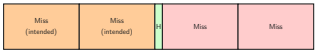


P-2-1-1-17 (34 accesses, 191ns)

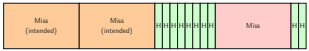
Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

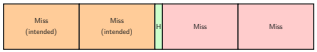


P-2-1-1-17 (34 accesses, 191ns)



Time in ns

P-1-1-1-17 (17 accesses, 307ns)

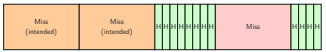


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)



Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

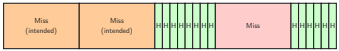
P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

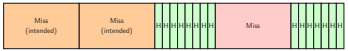


Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

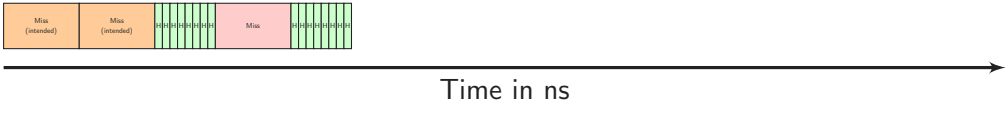
P-2-1-1-17 (34 accesses, 191ns)



Time in ns

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)



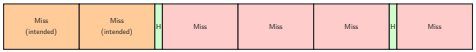
P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

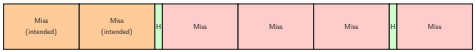


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

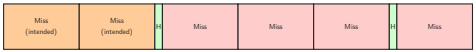
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

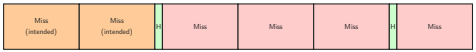


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

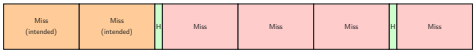
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

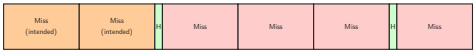


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

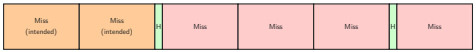


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

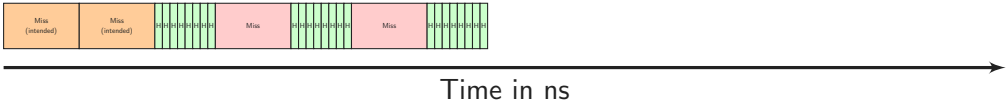
P-2-1-1-17 (34 accesses, 191ns)

Time in ns

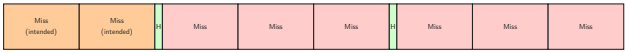
Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

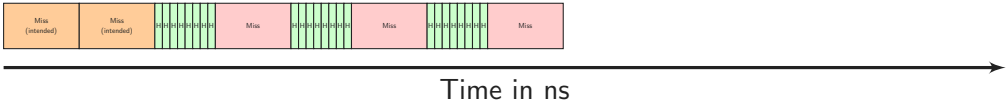
P-2-1-1-17 (34 accesses, 191ns)



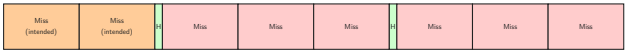
P-1-1-1-17 (17 accesses, 307ns)



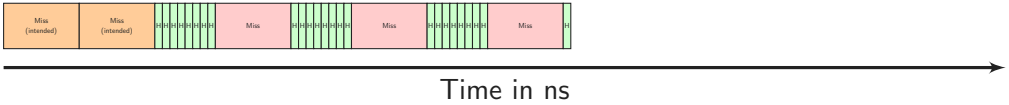
P-2-1-1-17 (34 accesses, 191ns)



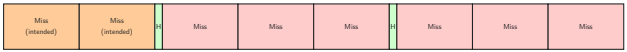
P-1-1-1-17 (17 accesses, 307ns)



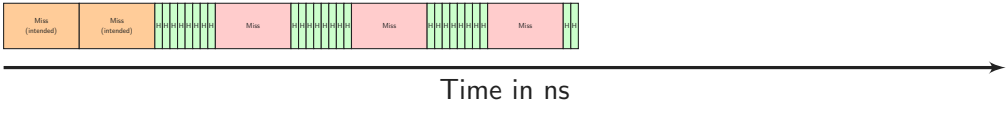
P-2-1-1-17 (34 accesses, 191ns)



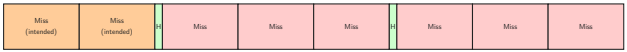
P-1-1-1-17 (17 accesses, 307ns)



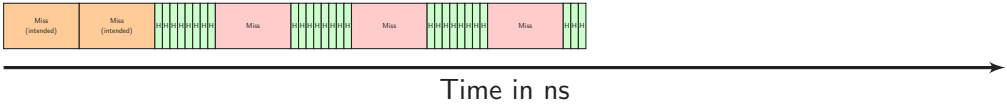
P-2-1-1-17 (34 accesses, 191ns)



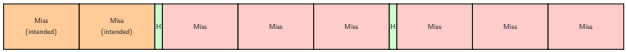
P-1-1-1-17 (17 accesses, 307ns)



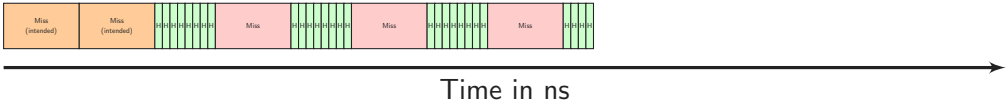
P-2-1-1-17 (34 accesses, 191ns)



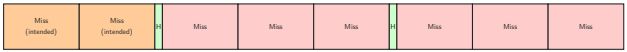
P-1-1-1-17 (17 accesses, 307ns)



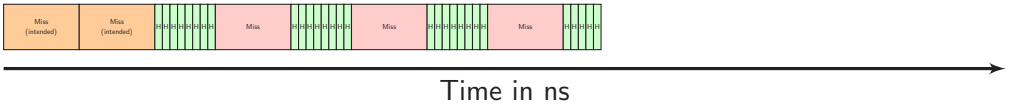
P-2-1-1-17 (34 accesses, 191ns)



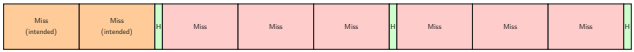
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)



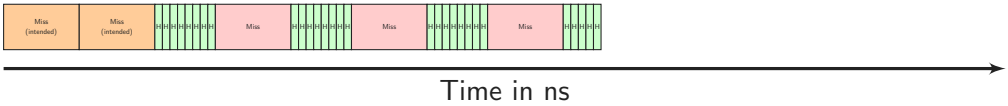
P-1-1-1-17 (17 accesses, 307ns)



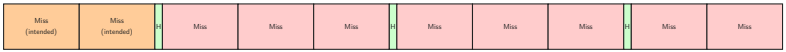
P-2-1-1-17 (34 accesses, 191ns)

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)



P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

P-1-1-1-17 (17 accesses, 307ns)

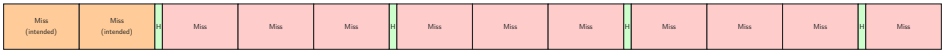
P-2-1-1-17 (34 accesses, 191ns)

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

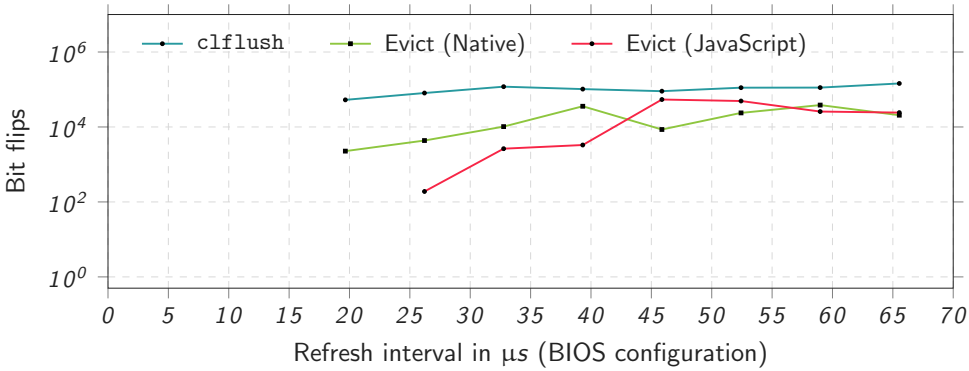


Figure 7: Number of bit flips within 15 minutes.

20: 12
30: 9
40: 1
50: 0
60: 1
70: 2
80: 199
90: 76
00: 72
10: 231
20: 572
250
] Found flip (254 != 255) at array index 340021386 when hammering indices 339881984 and 340156416

] Found flip (239 != 255) at array index 340022176 when hammering indices 339881984 and 340156416

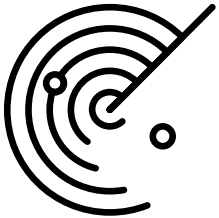
] Found flip (191 != 255) at array index 340023138 when hammering indices 339881984 and 340156416

] Found flip (254 != 255) at array index 340025146 when hammering indices 339881984 and 340156416

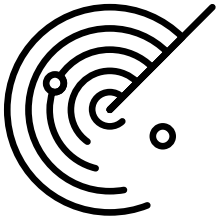
ROOT privileges for web apps!

JavaScript zero

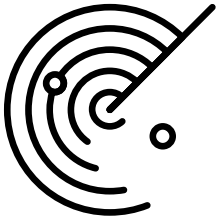
REAL JavaScript AND ZERO SIDE-CHANNEL ATTACKS



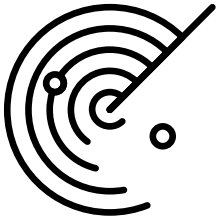
- Currently 11 microarchitectural and side-channel attacks in JavaScript



- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack

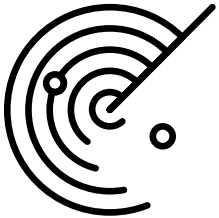


- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
- Results in 5 categories



- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
- Results in 5 categories
  - Memory addresses
  - Accurate timing
  - Multithreading
  - Shared data
  - Sensor API

Identify requirements



- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
- Results in 5 categories
  - Memory addresses
  - Accurate timing
  - Multithreading
  - Shared data
  - Sensor API
- Every attack is in at least one category

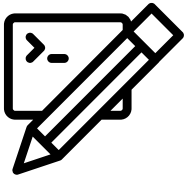
Attacks and Categories

	Mem. addrs.	Accurate timing	Multithreading	Shared data	Sensor API
Rowhammer.js	●	●	○	○	○
Practical Memory Deduplication Attacks in Sandboxed Javascript	◐	●	○	○	○
Fantastic Timers and Where to Find Them	●	●†	◐	◐	○
ASLR on the Line	●	●†	◐	◐	○
The spy in the sandbox	◐	●	○	○	○
Loophole	○	◐	●	○	○
Pixel perfect timing attacks with HTML5	○	●†	◐	◐	○
The clock is still ticking	○	●	◐	○	○
Practical Keystroke Timing Attacks in Sandboxed JavaScript	○	◐†	●	◐	○
TouchSignatures	○	○	○	○	●
Stealing sensitive browser data with the W3C Ambient Light Sensor API	○	○	○	○	●

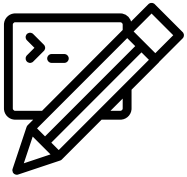
† If accurate timing is not available, it can be approximated using a combination of multithreading and shared

Defense \ Prevents	Rowhammer.js	Page Deduplication	DRAM Covert Channel	Anti-ASLR	Cache Eviction	Keystroke Timing	Browser
Buffer ASLR	○	◐	○	●	●	○	○
Array preloading	●	○	●	○	○	○	○
Non-deterministic array	●	◐	◐	●	●	○	○
Array index randomization	○	●	○	●	○	○	○
Low-resolution timestamp	○	◐	○	○	○	◐	◐
Fuzzy time	○	◐*	○	○*	○	●*	●*
`WebWorker` polyfill	○	○	●	●	●	●	○
Message delay	○	○	○	○	○	◐	◐
Slow `SharedArrayBuffer`	○	○	●	◐	●	○	○
No `SharedArrayBuffer`	○	○*	●	●*	●	○*	○*
Summary	●	●	●	●	●	●	●

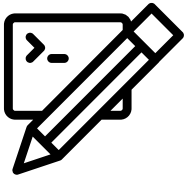
Prevented (●), partly prevented / more difficult (◐), not prevented (○). A star (*) means the combination is necessary.



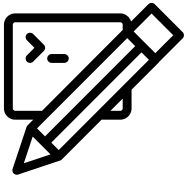
- Ideally → browser core



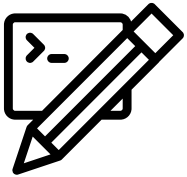
- Ideally → browser core
- Maintaining a fork is hard



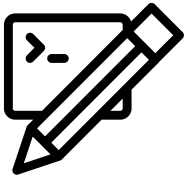
- Ideally → browser core
- Maintaining a fork is hard
- Generic solution for multiple browsers



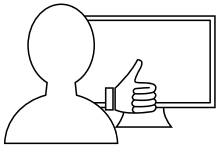
- Ideally → browser core
- Maintaining a fork is hard
- Generic solution for multiple browsers
- Parsing JavaScript is hard



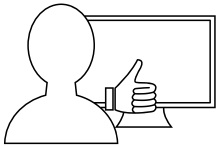
- Ideally → browser core
- Maintaining a fork is hard
- Generic solution for multiple browsers
- Parsing JavaScript is hard
- Implementation in JavaScript → Virtual machine layering



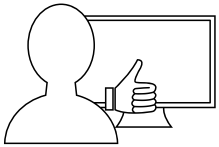
- Ideally → browser core
- Maintaining a fork is hard
- Generic solution for multiple browsers
- Parsing JavaScript is hard
- Implementation in JavaScript → Virtual machine layering
- Proof-of-concept → browser extension



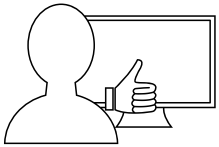
- Affects user experience? E.g., disable multithreading



- Affects user experience? E.g., disable multithreading
- Select pre-defined protection level

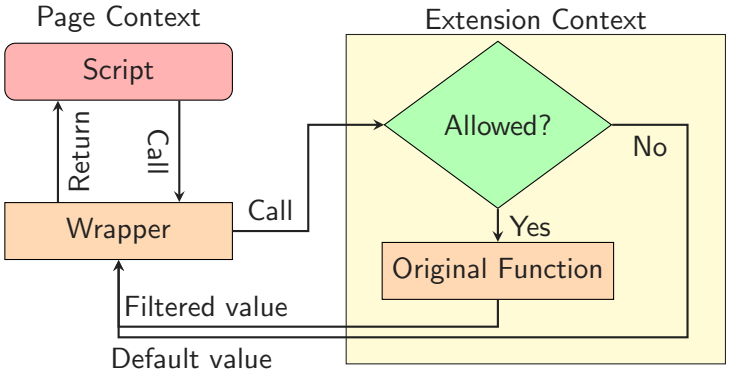


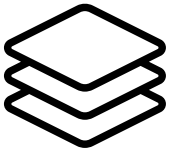
- Affects user experience? E.g., disable multithreading
- Select pre-defined protection level
- Protection levels $\rightarrow$ combinations of defenses



- Affects user experience? E.g., disable multithreading
- Select pre-defined protection level
- Protection levels $\rightarrow$ combinations of defenses
- Each defense is disabled, enabled, or prompts

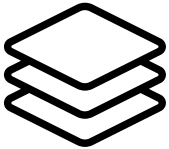
- Functions and properties are replaced by wrappers





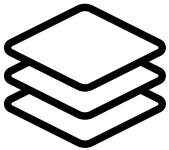
- Functions can be re-defined in JavaScript

```
1 var original_reference = window.performance.now;
2 window.performance.now = function() { return 0; };
3
```



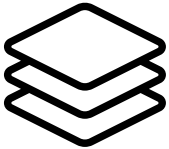
- Functions can be re-defined in JavaScript

```
1 var original_reference = window.performance.now;
2 window.performance.now = function() { return 0; };
3
4 // call the new function (via function name)
5 alert(window.performance.now()); // == alert(0)
6
```



- Functions can be re-defined in JavaScript

```
1 var original_reference = window.performance.now;
2 window.performance.now = function() { return 0; };
3
4 // call the new function (via function name)
5 alert(window.performance.now()); // == alert(0)
6
7 // call the original function (only via reference)
8 alert(original_reference.call(window.performance));
```

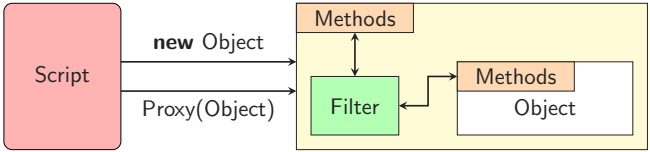


- Functions can be re-defined in JavaScript

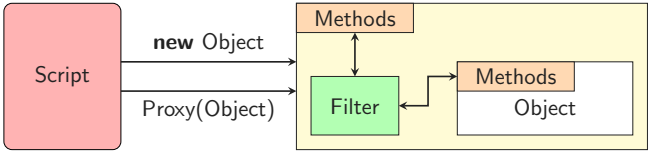
```
1 var original_reference = window.performance.now;
2 window.performance.now = function() { return 0; };
3
4 // call the new function (via function name)
5 alert(window.performance.now()); // == alert(0)
6
7 // call the original function (only via reference)
8 alert(original_reference.call(window.performance));
```

- Properties can be replaced by accessor properties

- Objects are proxied

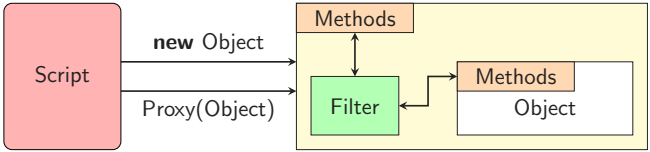


- Objects are proxied



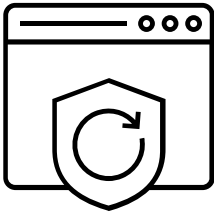
- All properties and functions are handled by the original object

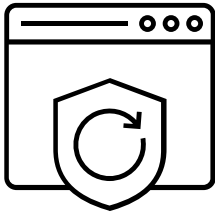
- Objects are proxied



- All properties and functions are handled by the original object
- Functions and properties can be overwritten in the proxy object

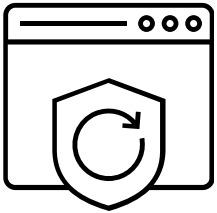
- Attacker tries to circumvent JavaScript Zero





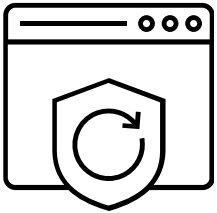
- Attacker tries to circumvent JavaScript Zero
- Self protection is necessary if implemented in JavaScript

Self Protection



- Attacker tries to circumvent JavaScript Zero
- Self protection is necessary if implemented in JavaScript
- Use closures to hide all references to original functions

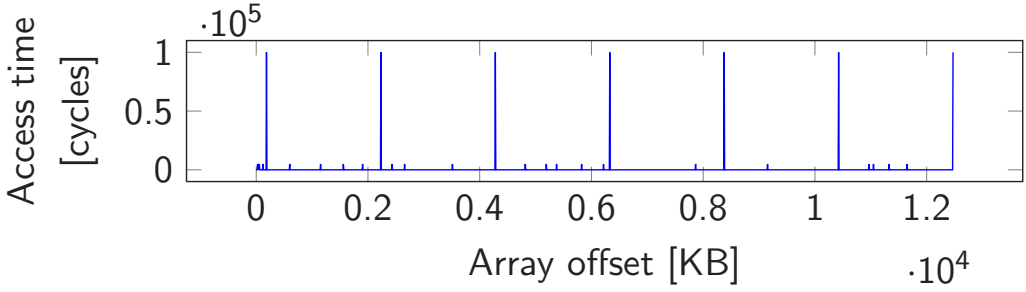
```
1 (function() {
2 // original is only accessible in this scope
3 var original = window.performance.now;
4 window.performance.now = ...
5 })();
```

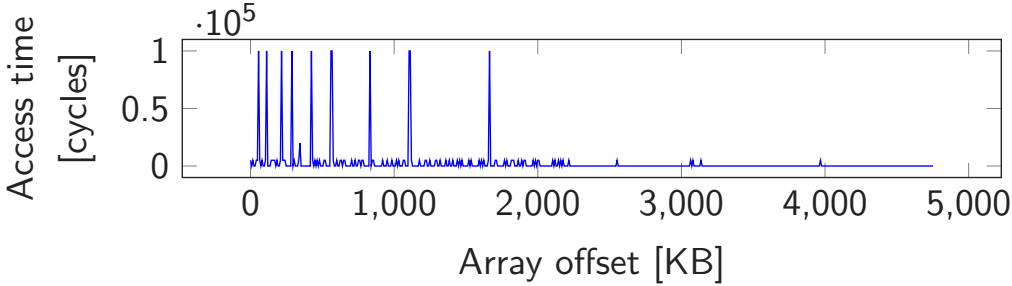


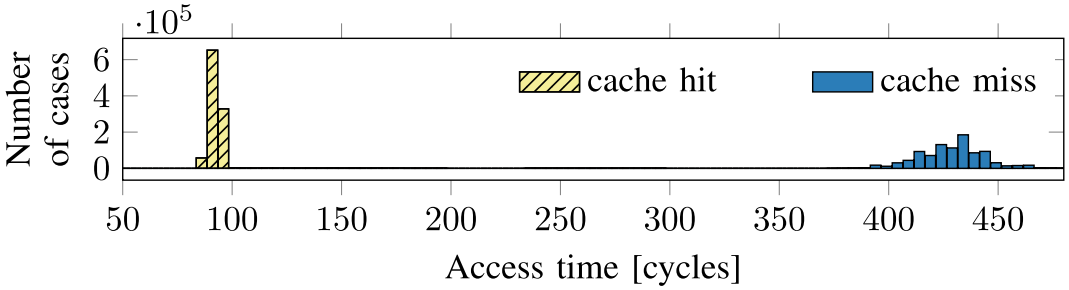
- Attacker tries to circumvent JavaScript Zero
- Self protection is necessary if implemented in JavaScript
- Use closures to hide all references to original functions

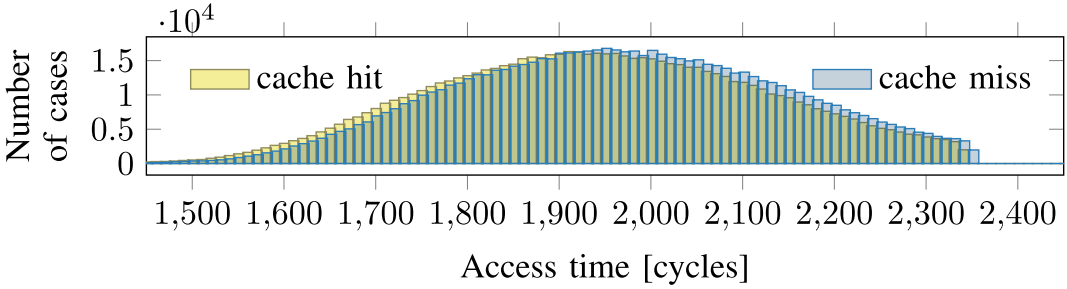
```
1 (function() {
2 // original is only accessible in this scope
3 var original = window.performance.now;
4 window.performance.now = ...
5 })();
```

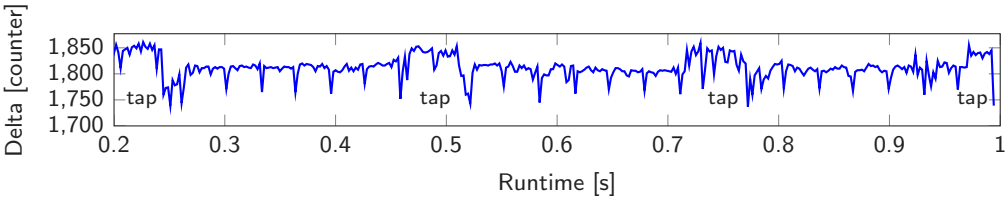
- Prevent objects from being modified: `Object.freeze`

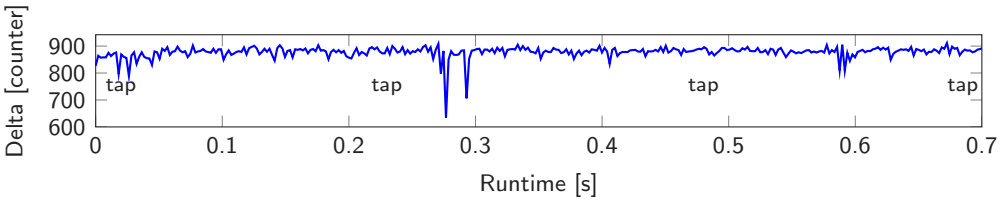

Access time
[cycles]
·10^5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2
Array offset [KB]
·10^4

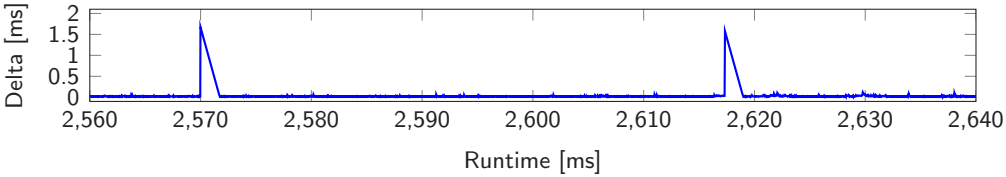

Access time [cycles]
$\cdot 10^5$
1
0.5
0
0
1,000
2,000
3,000
4,000
5,000
Array offset [KB]

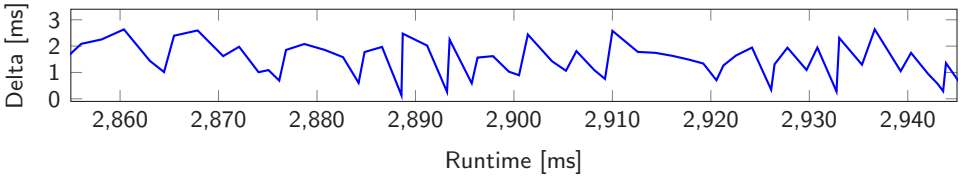

$\cdot 10^5$
Number of cases
6
4
2
0
cache hit
cache miss
50
100
150
200
250
300
350
400
450
Access time [cycles]

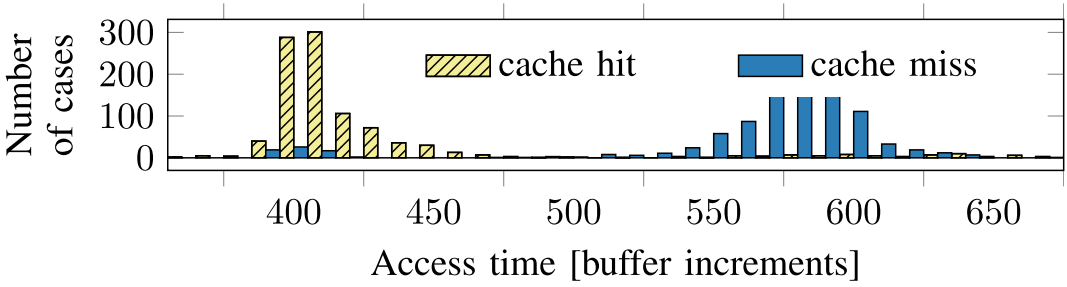

$\cdot 10^4$
Number of cases
1.5
1
0.5
0
cache hit
cache miss
1,500 1,600 1,700 1,800 1,900 2,000 2,100 2,200 2,300 2,400
Access time [cycles]

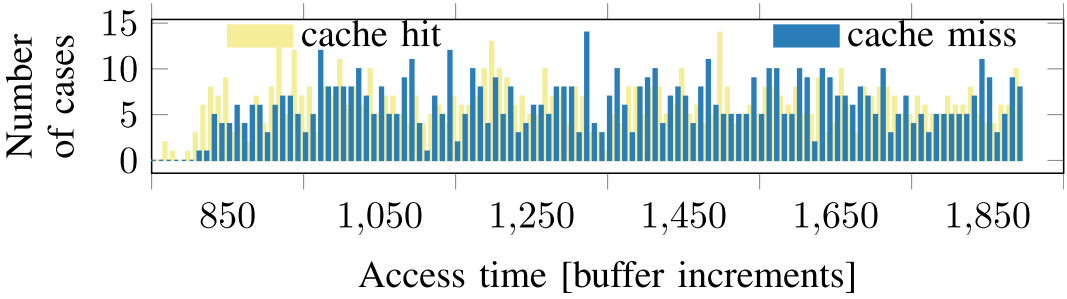

Delta [counter]
1,850
1,800
1,750
1,700
tap
tap
tap
tap
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Runtime [s]

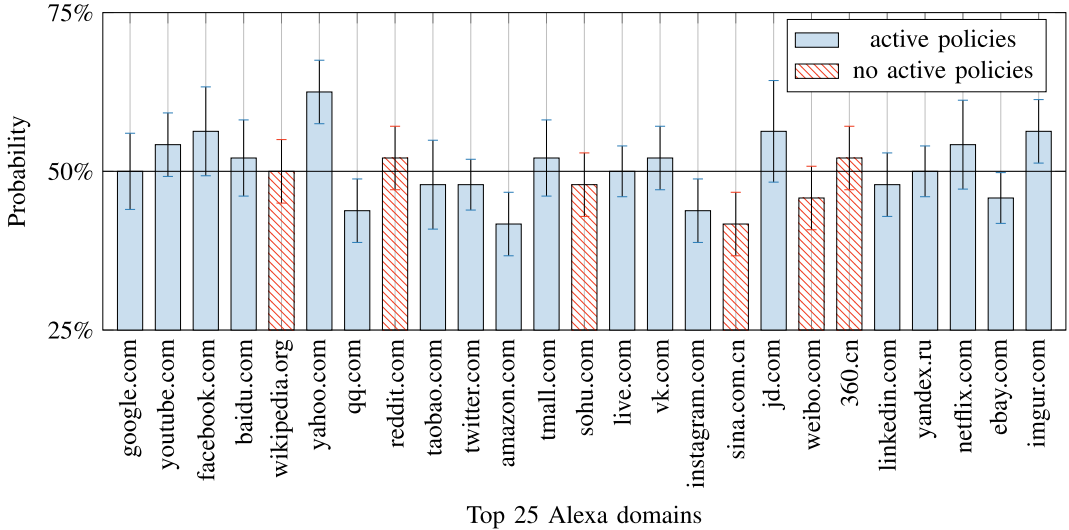

Delta [counter]
900
800
700
600
tap
tap
tap
tap
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Runtime [s]


Delta [ms]
2
1.5
1
0.5
0
2,560
2,570
2,580
2,590
2,600
2,610
2,620
2,630
2,640
Runtime [ms]


Delta [ms]
3
2
1
0
2,860
2,870
2,880
2,890
2,900
2,910
2,920
2,930
2,940
Runtime [ms]

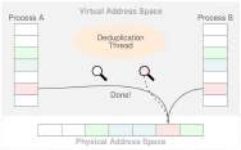

Number of cases
300
200
100
0
cache hit
cache miss
400
450
500
550
600
650
Access time [buffer increments]


Number of cases
15
10
5
0
cache hit
cache miss
850
1,050
1,250
1,450
1,650
1,850
Access time [buffer increments]

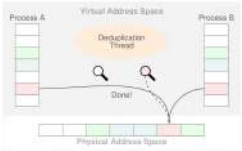

75%
50%
25%
Probability
active policies
no active policies
google.com
youtube.com
facebook.com
baidu.com
wikipedia.org
yahoo.com
qq.com
reddit.com
taobao.com
twitter.com
amazon.com
tmall.com
sohu.com
live.com
vk.com
instagram.com
sina.com.cn
jd.com
weibo.com
360.cn
linkedin.com
yandex.ru
netflix.com
ebay.com
imgur.com
Top 25 Alexa domains


</>
JS

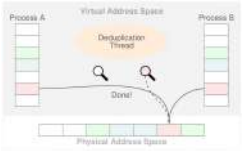

</>
JS


Process A
Virtual Address Space
Process B
Deduplication Thread
Done!
Physical Address Space

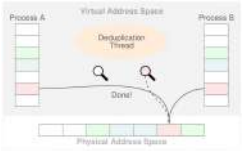

</>
JS


Virtual Address Space
Process A
Process B
Deduplication Thread
Done!
Physical Address Space

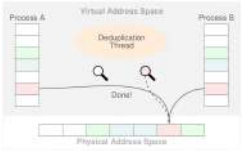

</>
JS


Process A
Virtual Address Space
Process B
Deduplication Thread
Done!
Physical Address Space

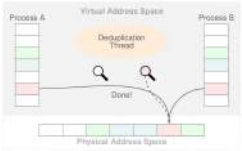

</>
JS

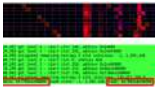

Process A
Virtual Address Space
Process B
Deduplication Thread
Done!
Physical Address Space


</>
JS

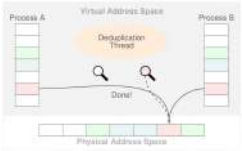

Process A
Virtual Address Space
Process B
Deduplication Thread
Done!
Physical Address Space

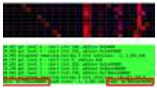

</>
JS


Process A
Virtual Address Space
Process B
Deduplication Thread
Done!
Physical Address Space

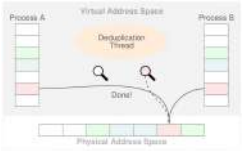


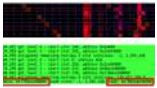

JS


Process A
Virtual Address Space
Process B
Deduplication Thread
Done!
Physical Address Space




SPECTRE





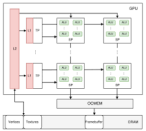
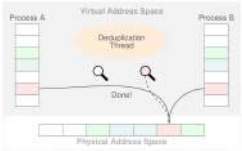
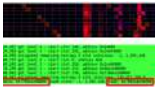


Fig. 2: Building blocks of an integrated GPU





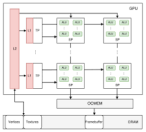
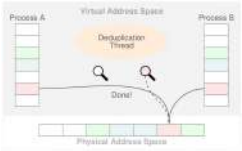
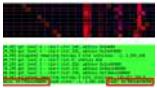


Fig. 2: Building blocks of an integrated GPU





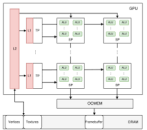


Fig. 2: Building blocks of an integrated GPU

- Jumping Abstraction Layers becomes easier

Conclusion

- Jumping Abstraction Layers becomes easier
- New attacks often also in JavaScript

- Jumping Abstraction Layers becomes easier
- New attacks often also in JavaScript
- We need cross-layer solutions

Jumping Abstraction Layers: Microarchitectural Attacks in JavaScript

Daniel Gruss

September 18, 2019

Graz University of Technology