Microarchitectural Attacks in the Cloud

Clémentine Maurice

April 23, 2017—Workshop on Security and Dependability of Multi-Domain Infrastructures, Belgrade, Serbia

Cloud abstracts physical resources to the user

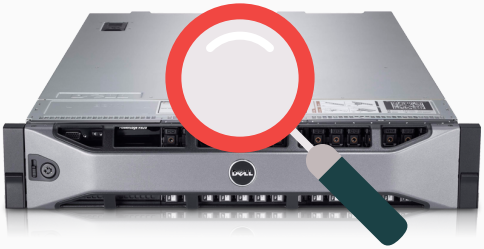
Logical isolation maintained by the hypervisor

Cloud abstracts physical resources to the user

Logical isolation maintained by the hypervisor

But there is a real physical world behind this abstraction

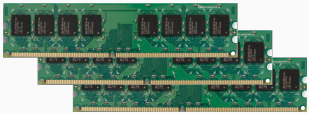
server



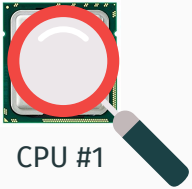
server

CPU #1

CPU #2

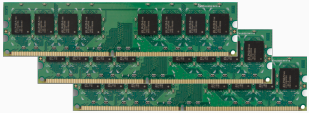


DRAM

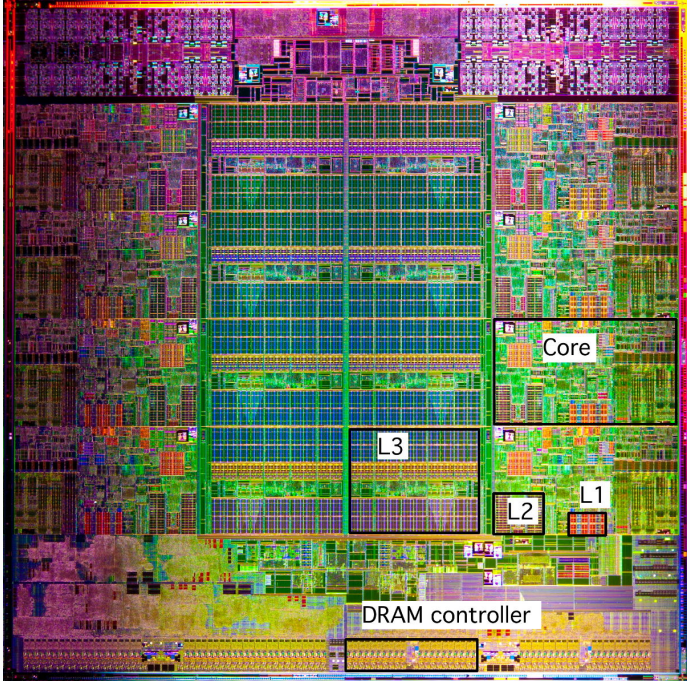


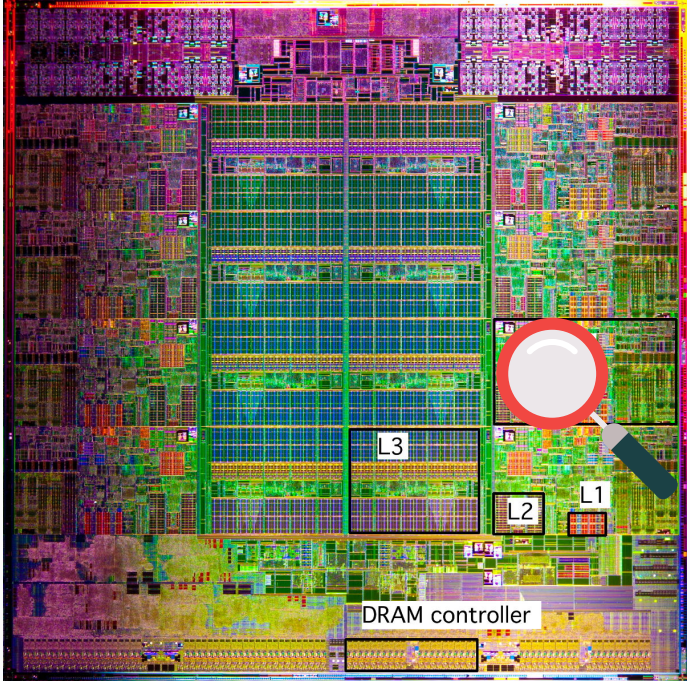
CPU #1

CPU #2



DRAM


Core
L3
L2
L1
DRAM controller


L3
L2
L1
DRAM controller

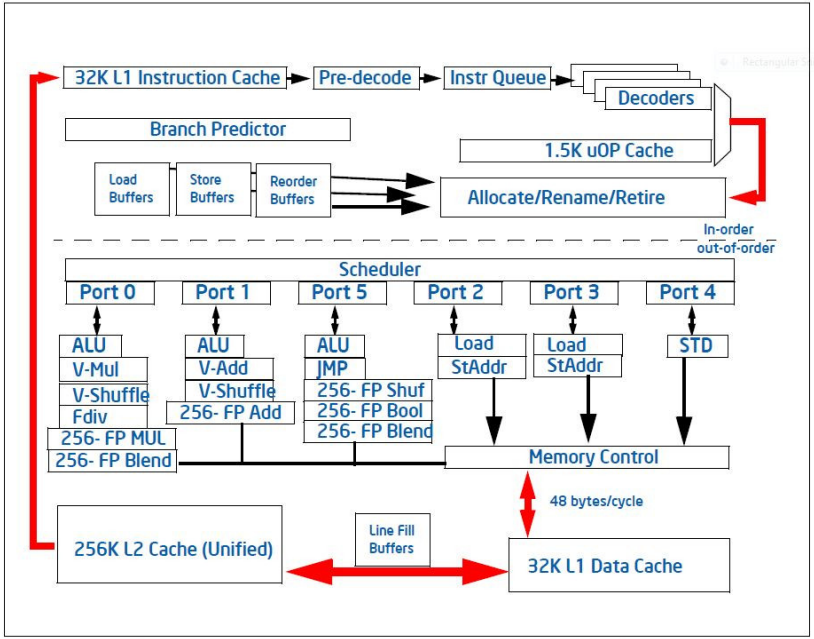
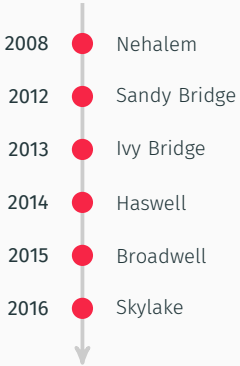


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

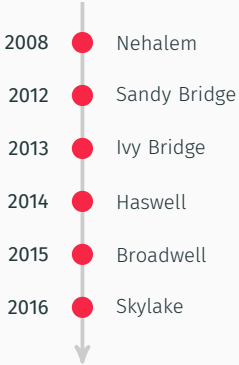
Microarchitectural Side-Channel Attacks

From small optimizations...



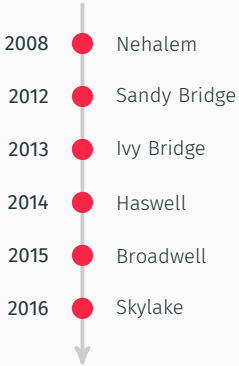
- new microarchitectures yearly

From small optimizations...



- new microarchitectures yearly
- performance improvement $\approx 5\%$

From small optimizations...



- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...

... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations

... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components

... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a victim process via hardware usage

... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a victim process via hardware usage
- pure-software attacks by unprivileged processes

... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a victim process via hardware usage
- pure-software attacks by unprivileged processes
- sequences of benign-looking actions $\rightarrow$ hard to detect

Same-Core Side-Channel Attacks

Hyper-threading

- threads sharing one core share resources: L1, L2 cache, branch predictor

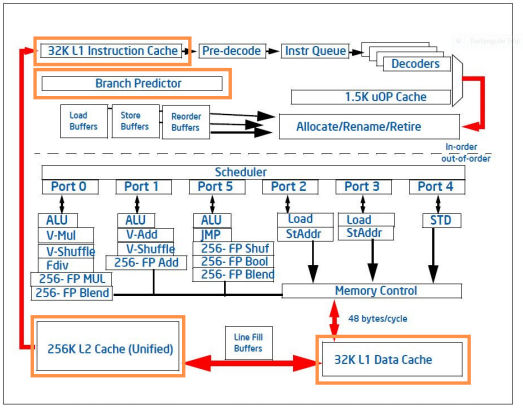


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

Branch prediction (1/2)

- conditional branches → taking the branch or not depends on some condition
- the condition has to be evaluated
- instead of stalling the pipeline → speculative execution of one possible path

→ branch prediction unit predicts the most likely execution path

Branch prediction (2/2)

- branch prediction unit
  - branch target buffer (BTB): cache that stores the target addresses of previously executed branches
  - branch predictor: makes the prediction on the outcome of the branch

Branch prediction (2/2)

- branch prediction unit
  - branch target buffer (BTB): cache that stores the target addresses of previously executed branches
  - branch predictor: makes the prediction on the outcome of the branch

- two things can go wrong
  1. BTB miss
  2. misprediction of the branch
- can be observed by timing penalty or hardware performance counters

Side channels on the branch prediction unit

- attacker and victim processes executed on the same core
- algorithm with secret-dependent path
- detect whether specific branches are taken or not taken

$S \leftarrow A \times B$
$S \leftarrow (S - (S \times N - 1 \mod R) \times N)/R$
if $S > N$ **then**
    $S \leftarrow S - N$
return S

Side channel on the BTB

- attacker forces the eviction of victim's entries in the BTB
- next victim execution → BTB miss → forces a prediction *not-taken*
- if the branch is in fact *taken* → misprediction → update the BTB
- next attacker execution → entries not in the BTB anymore

→ attacker's BTB misses used to deduce victim's branch mispredictions

O. Acıiçmez, J.-P. Seifert, and c. K. Koç. "Predicting secret keys via branch prediction". In: *CT-RSA 2007*. 2007

Side channel on the BTB

- attacker forces the eviction of victim's entries in the BTB
- next victim execution → BTB miss → forces a prediction *not-taken*
- if the branch is in fact *taken* → misprediction → update the BTB
- next attacker execution → entries not in the BTB anymore

→ attacker's BTB misses used to deduce victim's branch mispredictions

- requires to know how the BTB is indexed (undocumented)

O. Acıiçmez, J.-P. Seifert, and c. K. Koç. "Predicting secret keys via branch prediction". In: *CT-RSA 2007*. 2007

Side channel on the BTB

- attacker forces the eviction of victim's entries in the BTB
- next victim execution → BTB miss → forces a prediction *not-taken*
- if the branch is in fact *taken* → misprediction → update the BTB
- next attacker execution → entries not in the BTB anymore

→ attacker's BTB misses used to deduce victim's branch mispredictions

- requires to know how the BTB is indexed (undocumented)
- in most cases micro-benchmarks can extract enough information

O. Acıiçmez, J.-P. Seifert, and c. K. Koç. "Predicting secret keys via branch prediction". In: *CT-RSA 2007*. 2007

Easy solution #1

Possible side channels using components shared by a core?

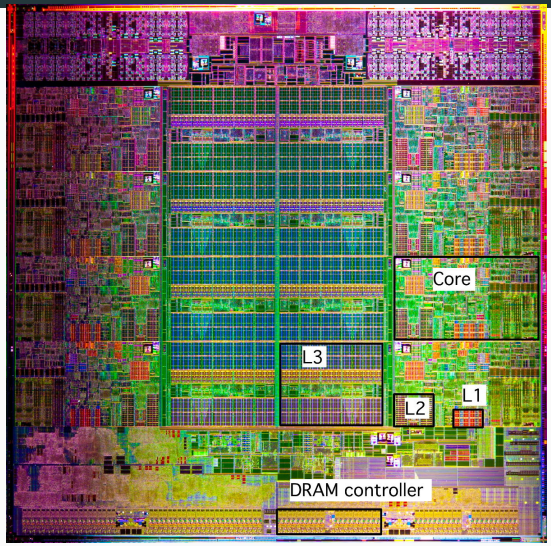
Easy solution #1

Possible side channels using components shared by a core?

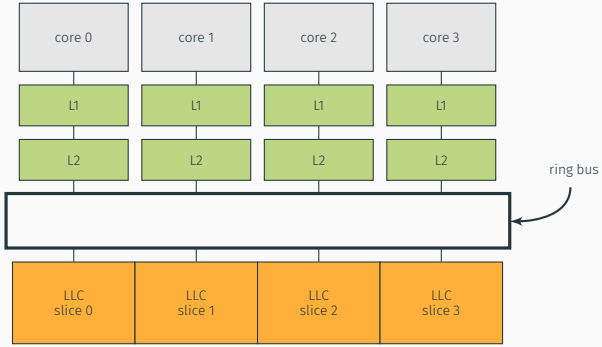
Stop sharing a core!

Cross-Core Side-Channel Attacks

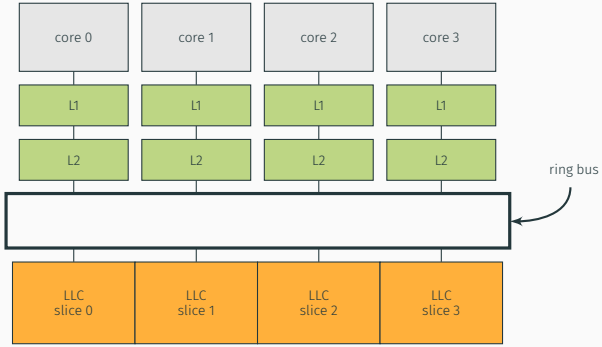
Caches on Intel CPUs



Caches on Intel CPUs

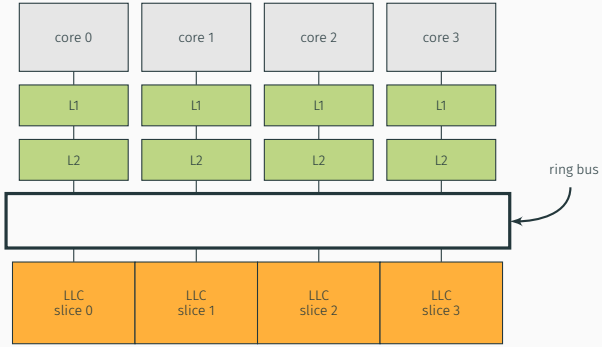


Caches on Intel CPUs



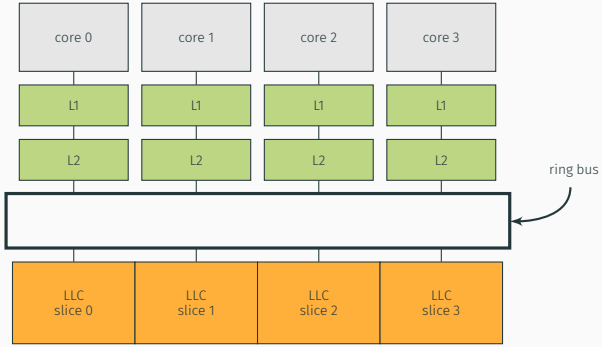
- L1 and L2 are private

Caches on Intel CPUs



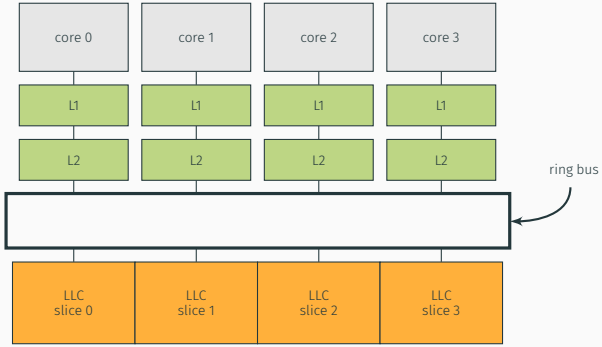
- L1 and L2 are private
- last-level cache

Caches on Intel CPUs



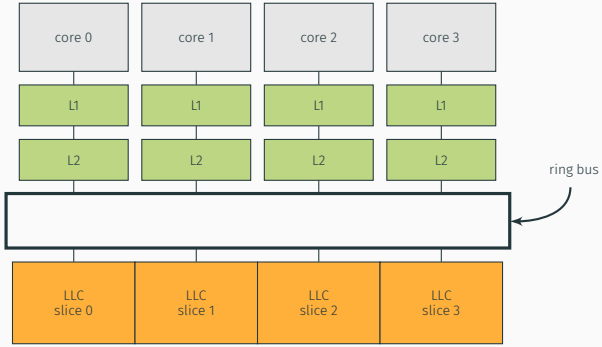
- L1 and L2 are private
- last-level cache
  - divided in slices

Caches on Intel CPUs



- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores

Caches on Intel CPUs

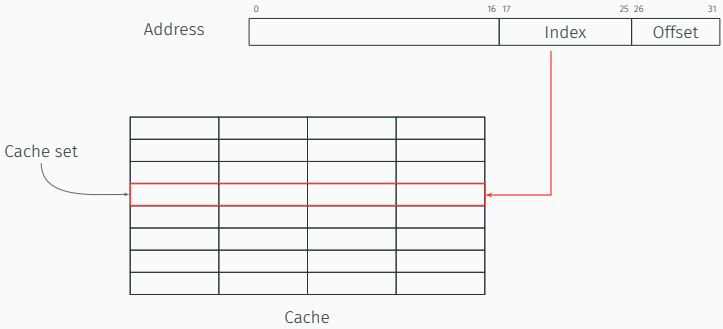


- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Set-associative caches

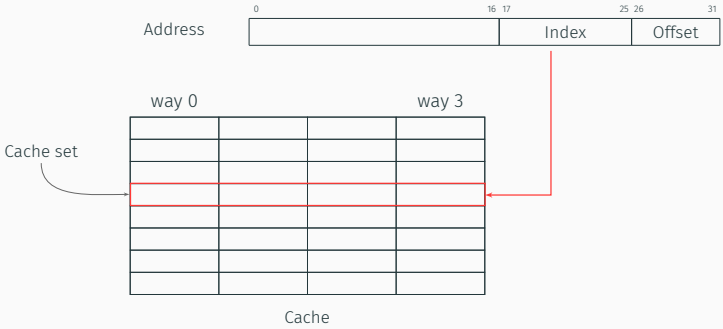
Cache

Set-associative caches



Data loaded in a specific set depending on its address

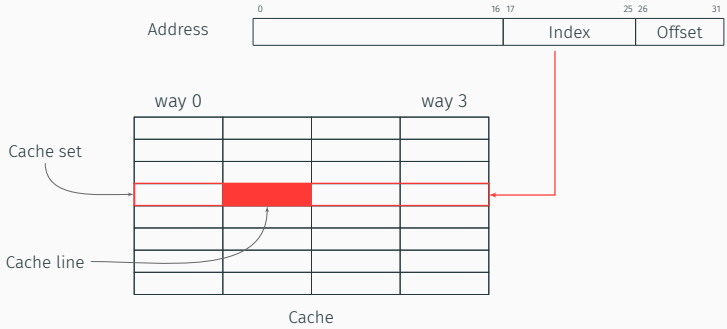
Set-associative caches



Data loaded in a specific set depending on its address

Several ways per set

Set-associative caches



Data loaded in a specific set depending on its address

Several ways per set

Cache line loaded in a specific way depending on the replacement policy

Cache attacks

- caches improve performance

Cache attacks

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches

Cache attacks

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches
- different timings for memory accesses

Cache attacks

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
    1. data is cached → cache hit → fast

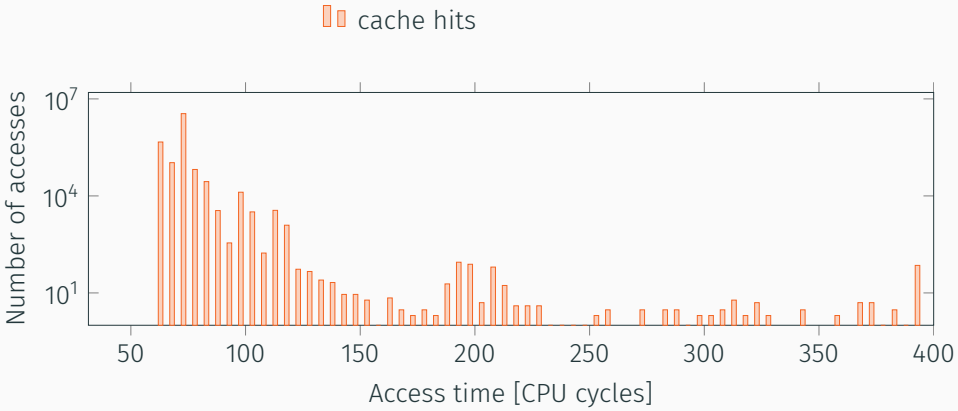
Cache attacks

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  1. data is cached → cache hit → fast
  2. data is not cached → cache miss → slow

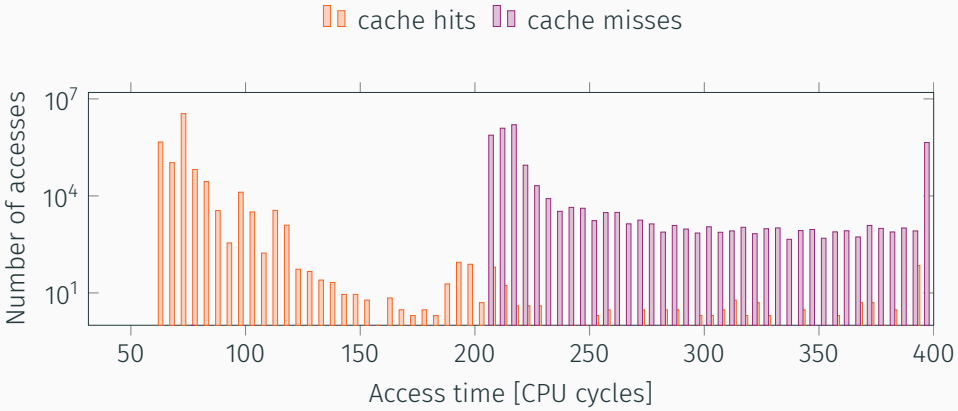
Cache attacks

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  1. data is cached → cache hit → fast
  2. data is not cached → cache miss → slow
- cache attacks leverage this timing difference

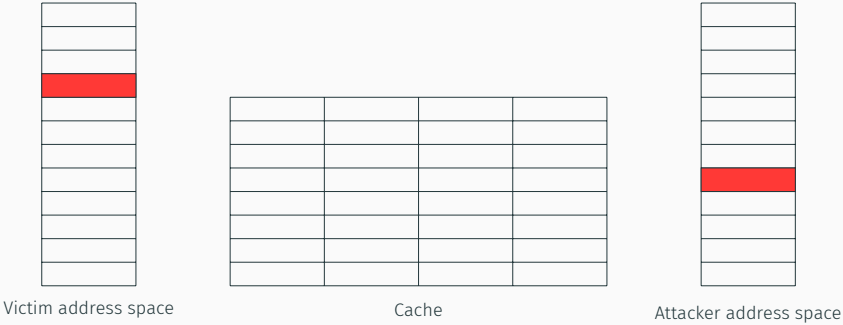
Timing differences



Timing differences

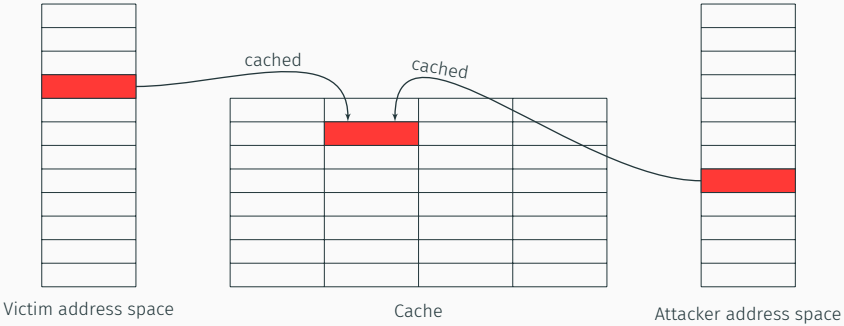


Cache attacks: Flush+Reload



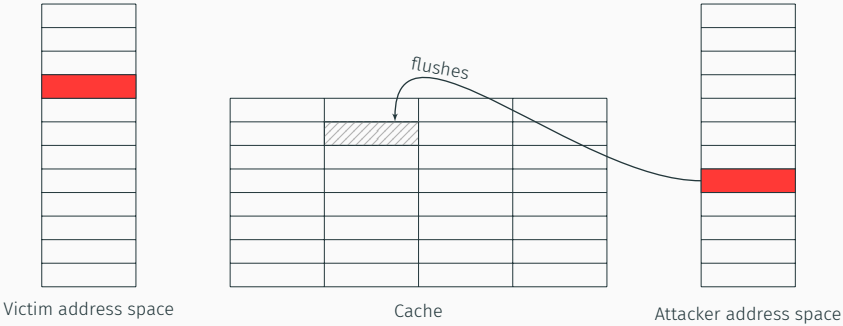
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

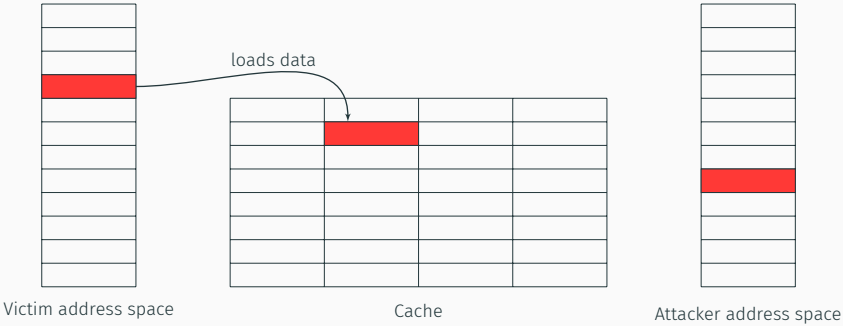
Cache attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

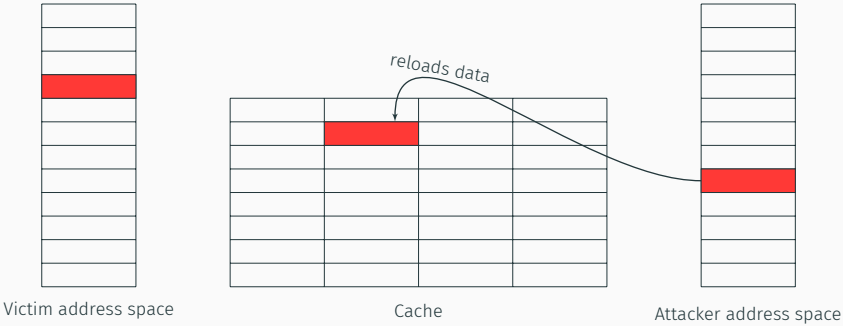


Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Cache attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker reloads the data

Flush+Reload: Applications

- cross-VM side channel attacks on crypto algorithms
  - RSA: 96.7% of secret key bits in a single signature
  - AES: full key recovery in 30000 dec. (a few seconds)

Y. Yarom and K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014

B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. “A Faster and More Realistic Flush+Reload Attack on AES”. . In: *COSADE'15*. 2015

D. Gruss, R. Spreitzer, and S. Mangard. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security*. 2015

`https://github.com/IAIK/cache_template_attacks`

Flush+Reload: Applications

- cross-VM side channel attacks on crypto algorithms
  - RSA: 96.7% of secret key bits in a single signature
  - AES: full key recovery in 30000 dec. (a few seconds)

- Cache Template Attacks: automatically finds information leakage

  $\rightarrow$ side channel on keystrokes and AES T-tables implementation

---

Y. Yarom and K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014

B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. “A Faster and More Realistic Flush+Reload Attack on AES”. . In: *COSADE'15*. 2015

D. Gruss, R. Spreitzer, and S. Mangard. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security*. 2015

`https://github.com/IAIK/cache_template_attacks`

Flush+Reload: Pros and cons

- fine granularity: 1 cache line (64 Bytes)

Flush+Reload: Pros and cons

- fine granularity: 1 cache line (64 Bytes)
- but requires shared memory

Flush+Reload: Pros and cons

- fine granularity: 1 cache line (64 Bytes)
- but requires shared memory

→ memory deduplication between VMs

Easy solution #2

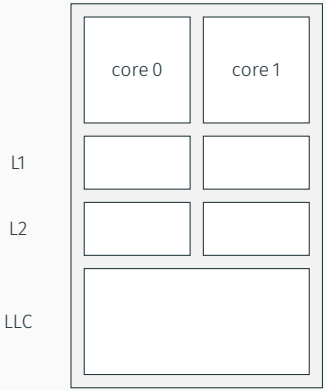
Possible side channels using memory deduplication?

Easy solution #2

Possible side channels using memory deduplication?

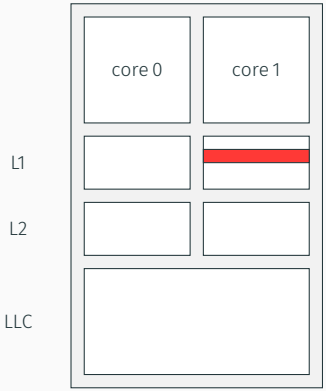
Disable memory deduplication!

Inclusive property



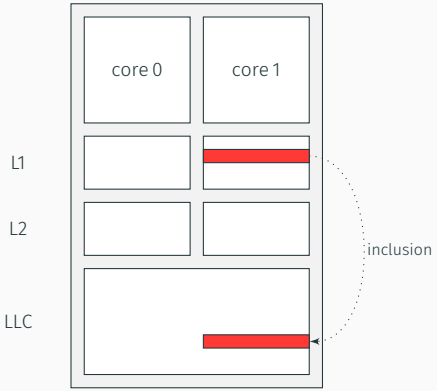
- inclusive LLC: superset of L1 and L2

Inclusive property



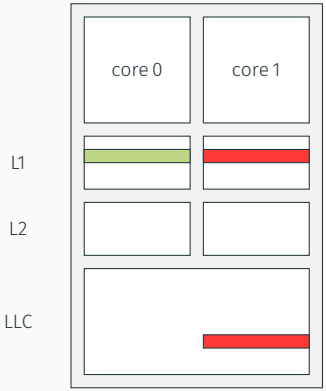
- inclusive LLC: superset of L1 and L2

Inclusive property



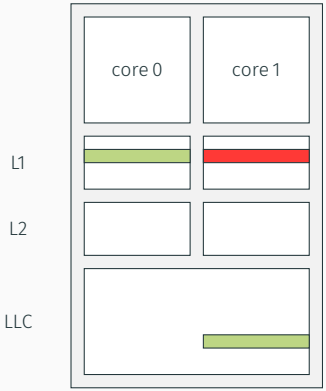
- inclusive LLC: superset of L1 and L2

Inclusive property



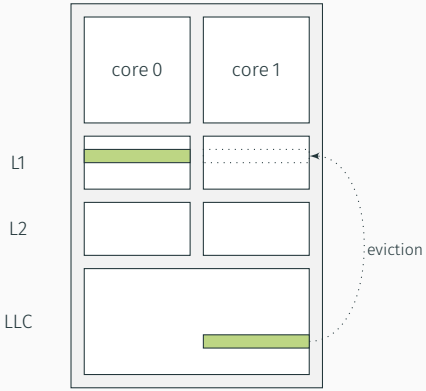
- inclusive LLC: superset of L1 and L2

Inclusive property



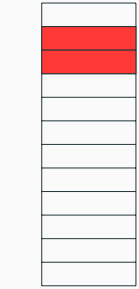
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2

Inclusive property

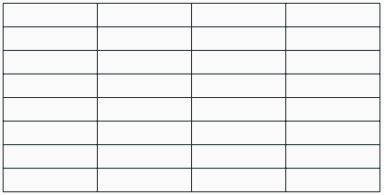


- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core

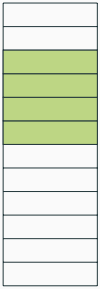
Cache attacks: Prime+Probe



Victim address space

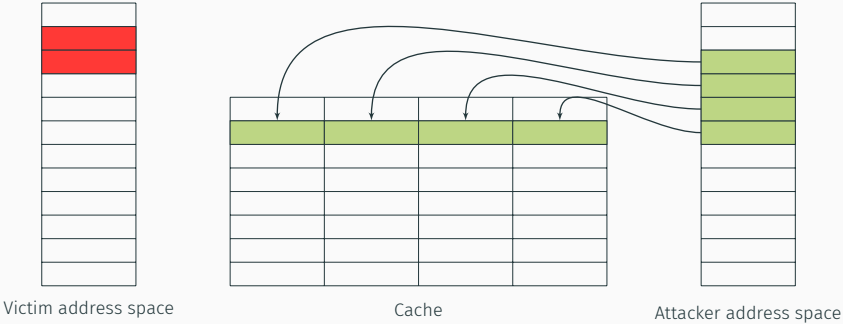


Cache



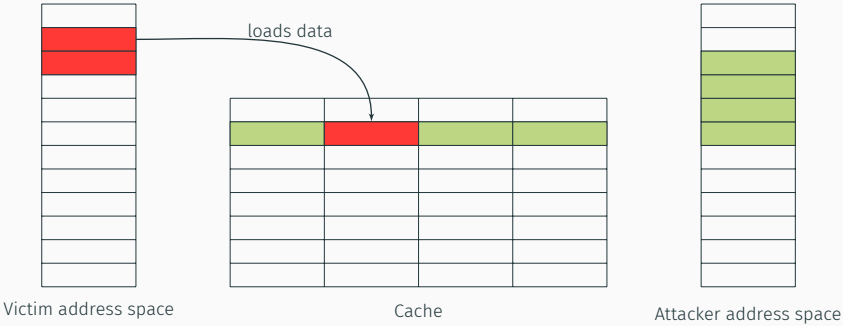
Attacker address space

Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

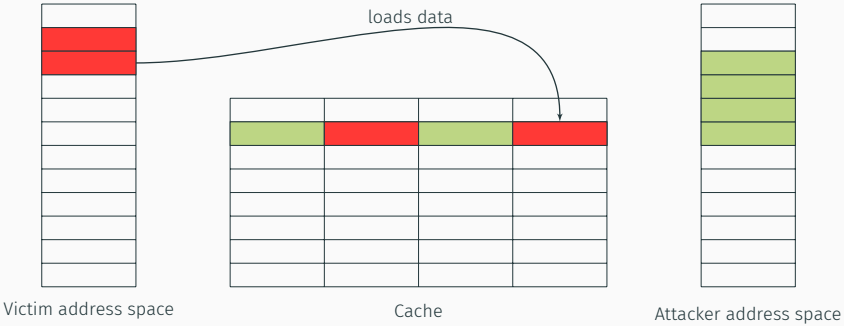
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

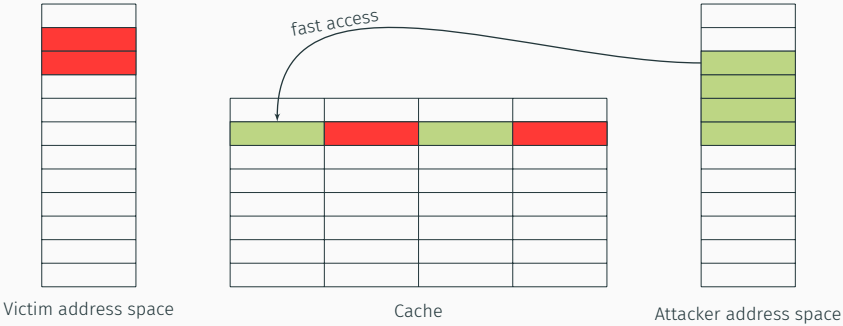
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

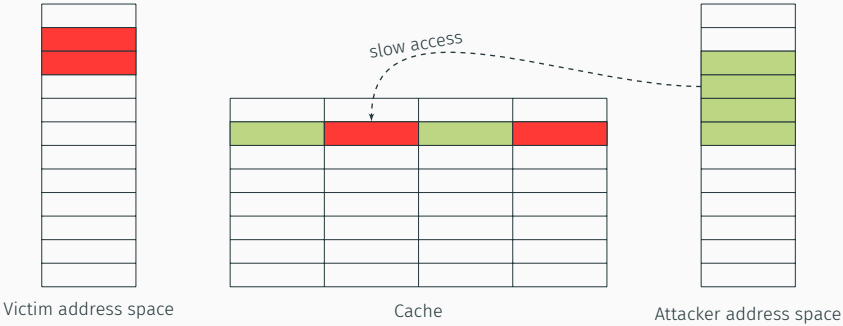


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

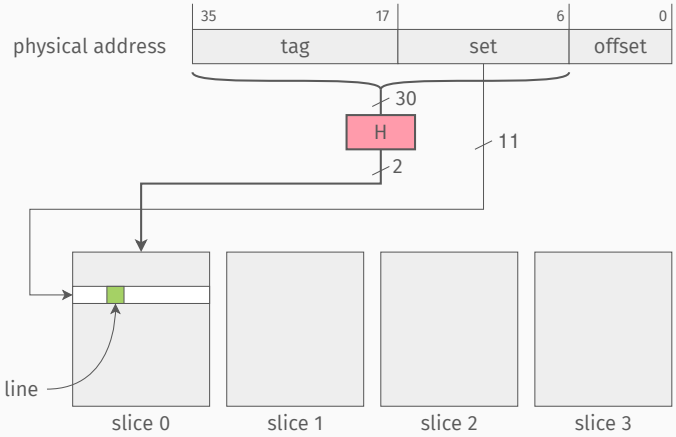
Step 3: Attacker probes data to determine if set has been accessed

Challenges with Prime+Probe

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

Last-level cache addressing



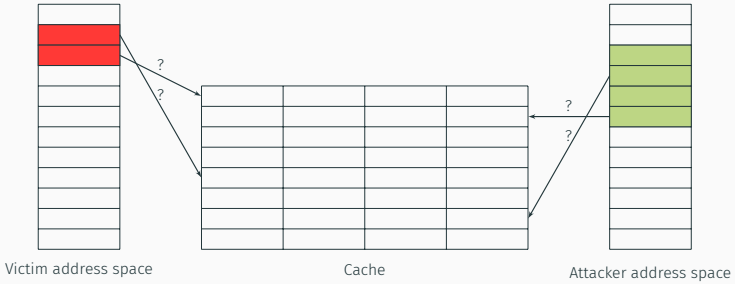
Last-level cache addressing

- last-level cache $\rightarrow$ as many slices as cores
- undocumented hash function that maps a physical address to a slice
- designed for performance

For 2^k slices:

Prime+Probe on recent procesors?

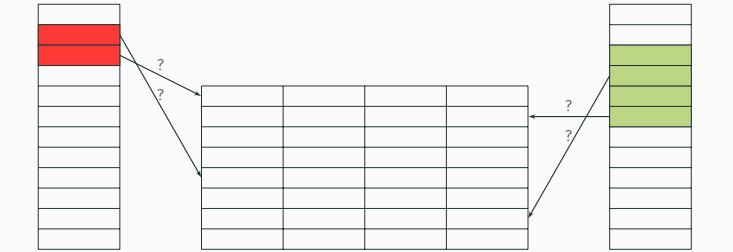
Undocumented function → impossible to target a set



C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: *RAID’15*. 2015

Prime+Probe on recent procesors?

Undocumented function → impossible to target a set



→ We reverse-engineered the function!

C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: *RAID'15*. 2015

Prime+Probe: Applications

- cross-VM side channel attacks on crypto algorithms:
  - El Gamal (sliding window): full key recovery in 12 min.
- tracking user behavior in the browser, in JavaScript
- covert channels between virtual machines in the cloud

F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. “Last-Level Cache Side-Channel Attacks are Practical”. In: *S&P’15*. 2015.

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015.

C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.

Covert channels in the cloud

- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs

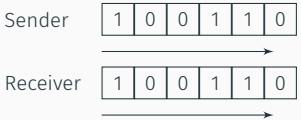
Covert channels in the cloud

- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine

Covert channels in the cloud

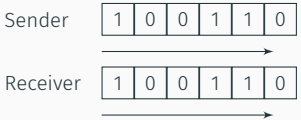
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine
- solution? “Just use error-correcting codes”

Why can’t we just use error correcting codes?

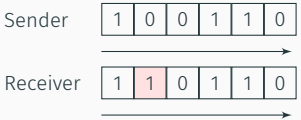


(a) Transmission without errors

Why can’t we just use error correcting codes?

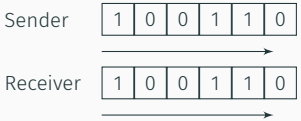


(a) Transmission without errors

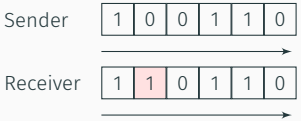


(b) Noise: substitution error

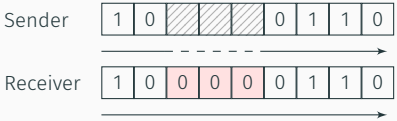
Why can’t we just use error correcting codes?



(a) Transmission without errors

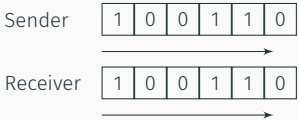


(b) Noise: substitution error

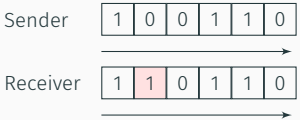


(c) Sender descheduled: insertions

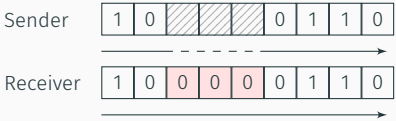
Why can’t we just use error correcting codes?



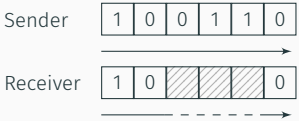
(a) Transmission without errors



(b) Noise: substitution error



(c) Sender descheduled: insertions



(d) Receiver descheduled: deletions

Our robust covert channel

- physical layer:
  - transmits words as a sequence of ‘0’s and ‘1’s
  - deals with synchronization errors
- data-link layer:
  - divides data to transmit into packets
  - corrects the remaining errors

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ’0’ doing nothing
  → lines of the receiver still in cache → fast access

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ’0’ doing nothing
  → lines of the receiver still in cache → fast access
- sender transmits ’1’ accessing addresses in the set
  → evicts lines of the receiver → slow access

Eviction set generation

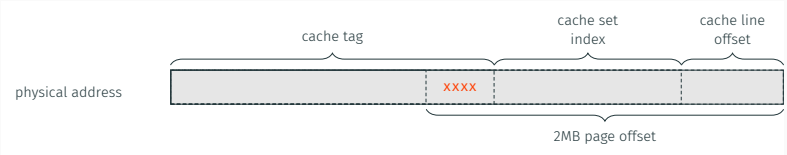
- need a set of addresses in the same cache set and same slice

Eviction set generation

- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

Eviction set generation

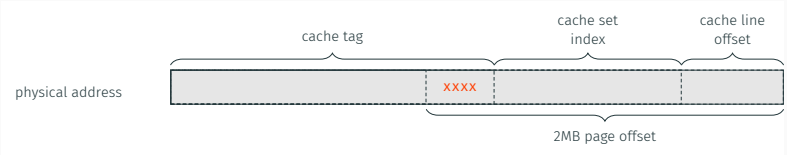
- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address



- we can build a set of addresses in the same cache set and same slice

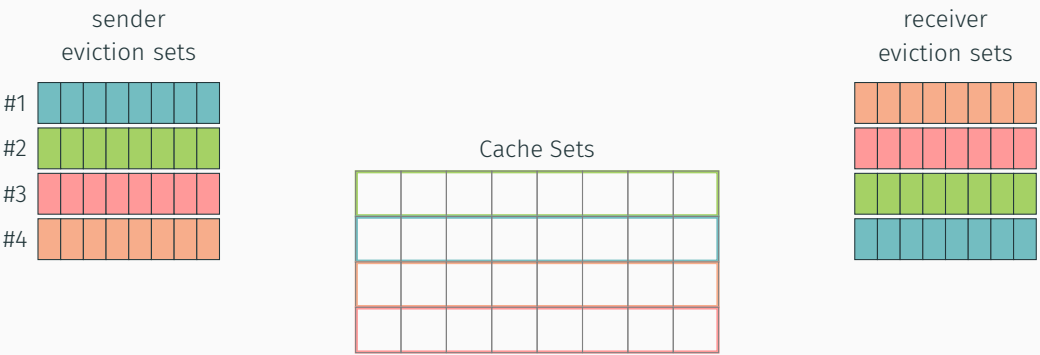
Eviction set generation

- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

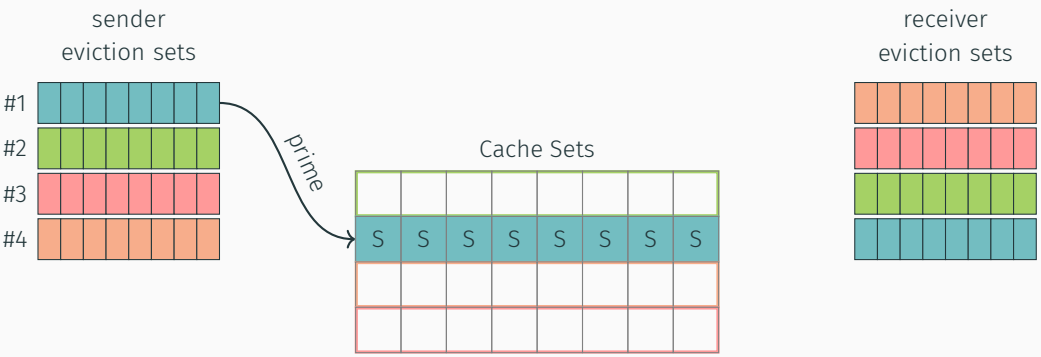


- we can build a set of addresses in the same cache set and same slice
- without knowing which slice

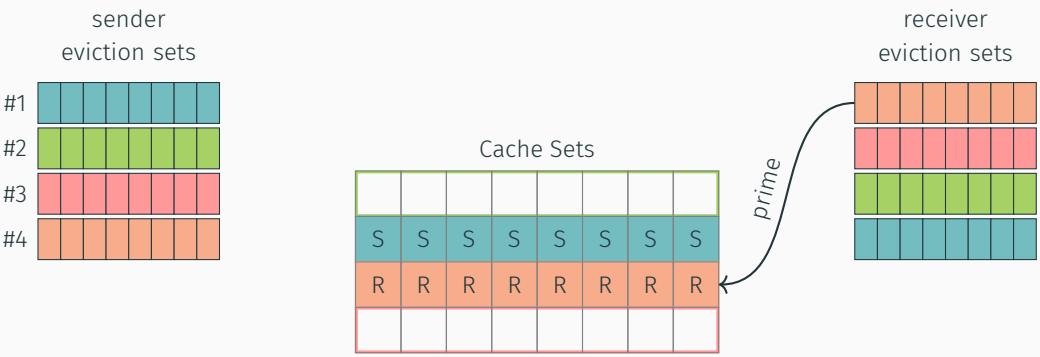
Jamming agreement



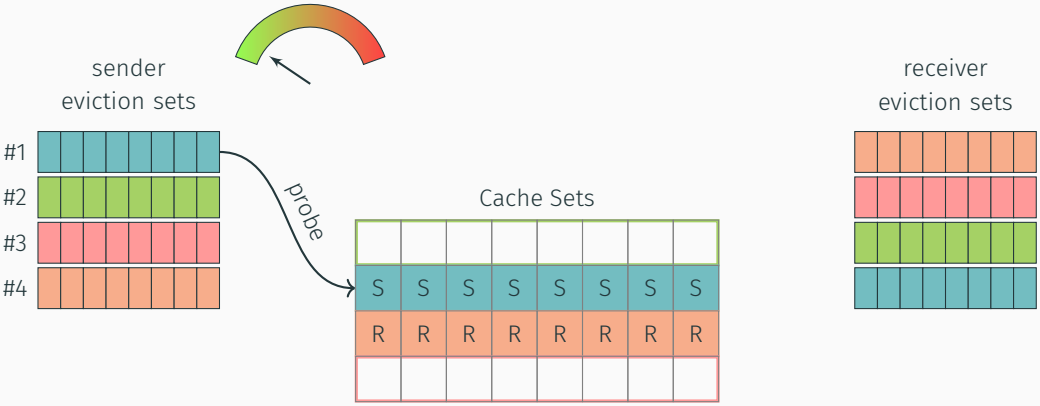
Jamming agreement



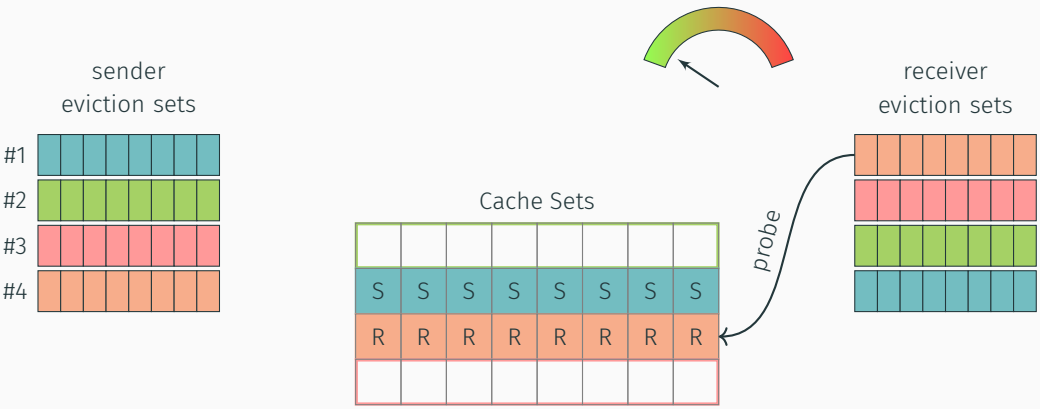
Jamming agreement



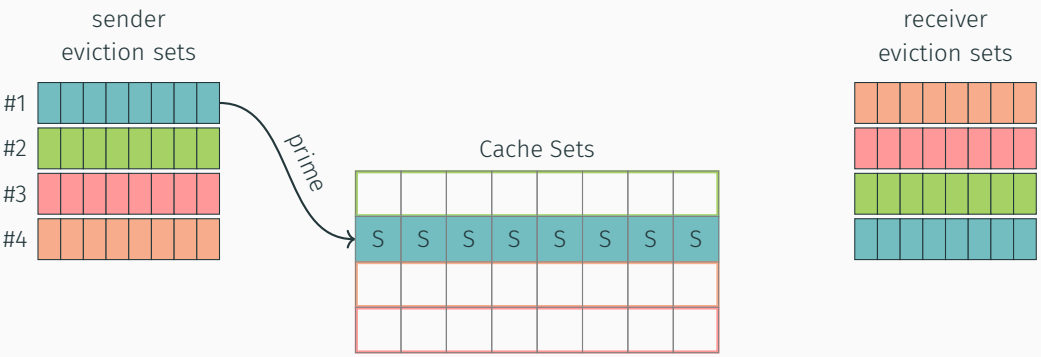
Jamming agreement



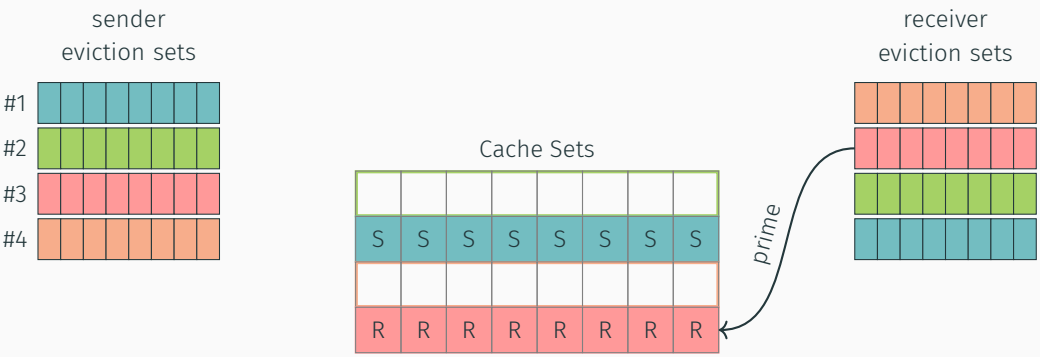
Jamming agreement



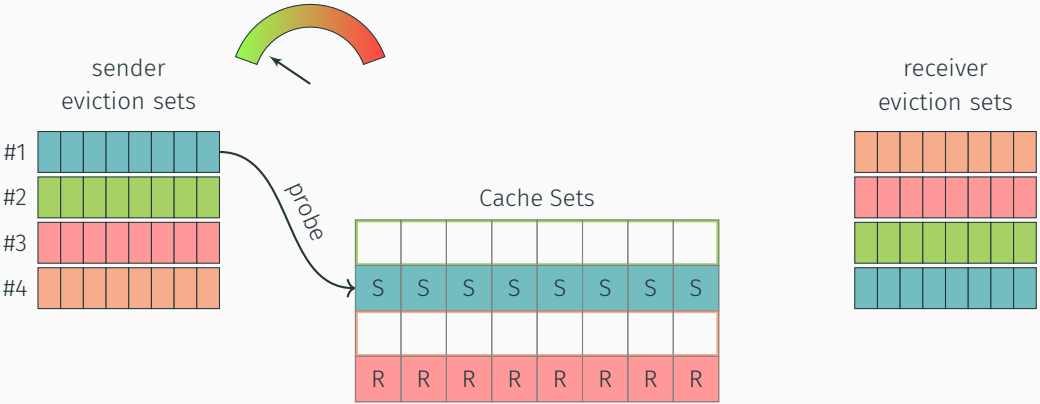
Jamming agreement



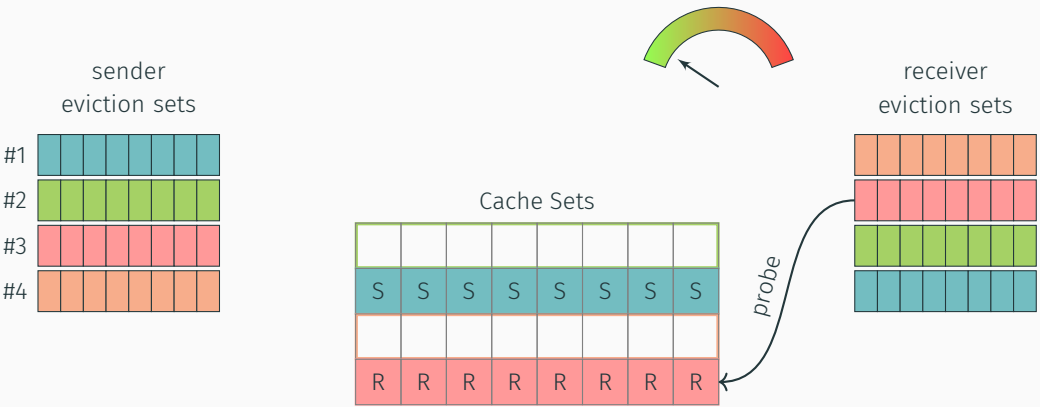
Jamming agreement



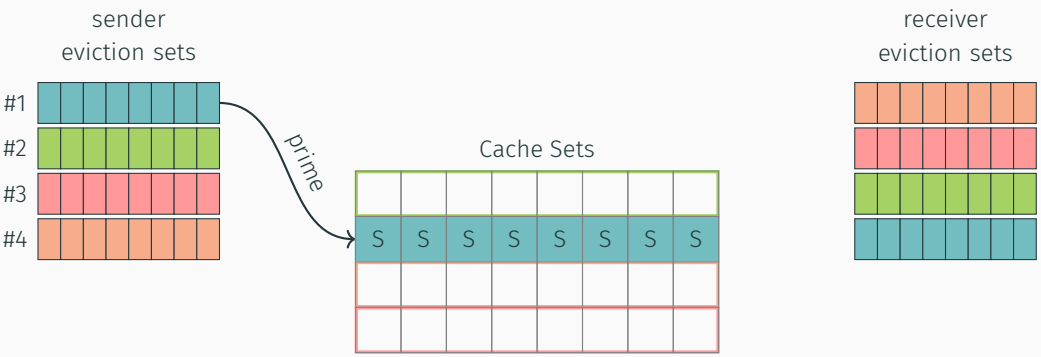
Jamming agreement



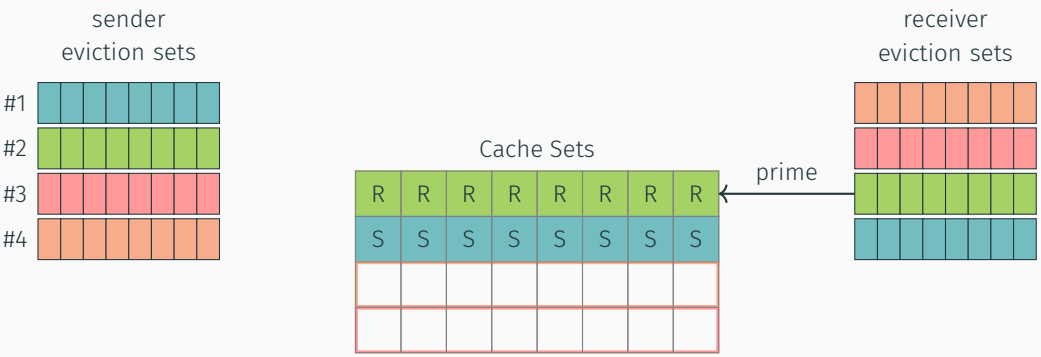
Jamming agreement



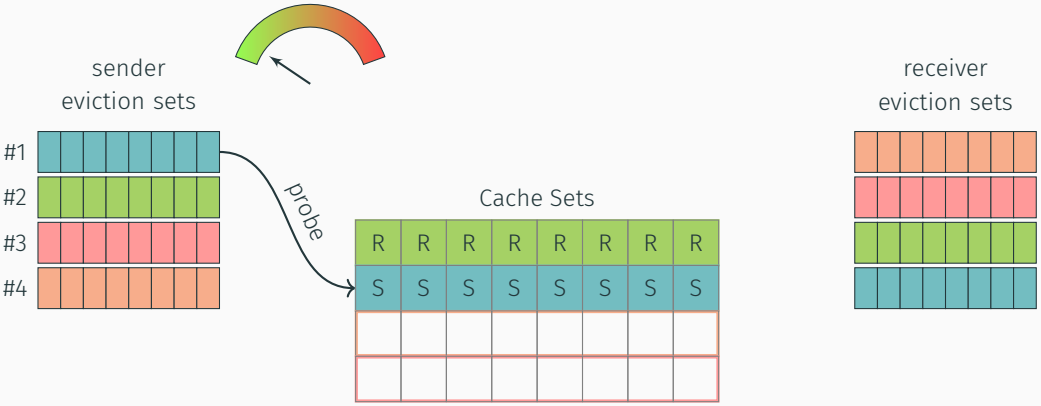
Jamming agreement



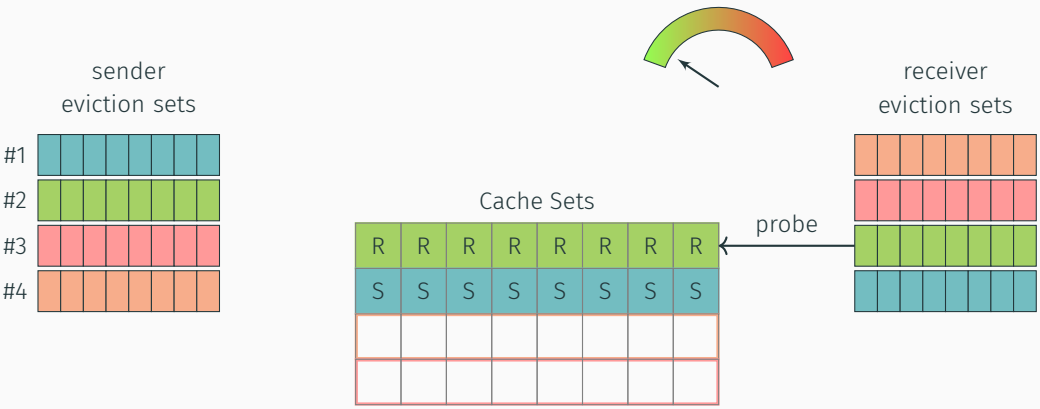
Jamming agreement



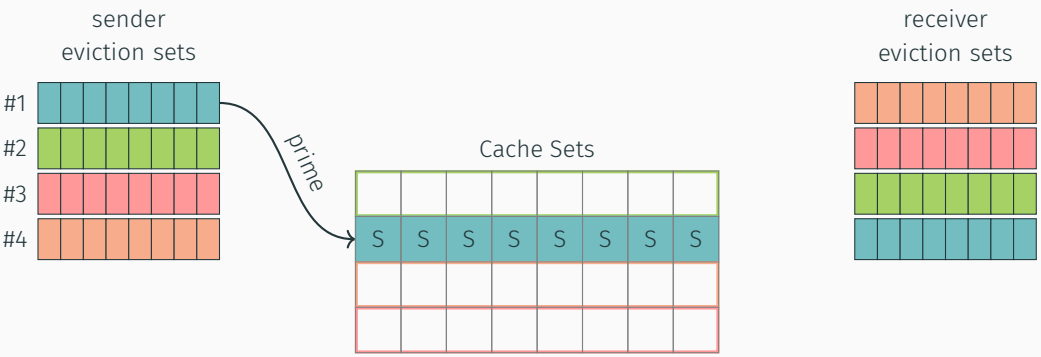
Jamming agreement



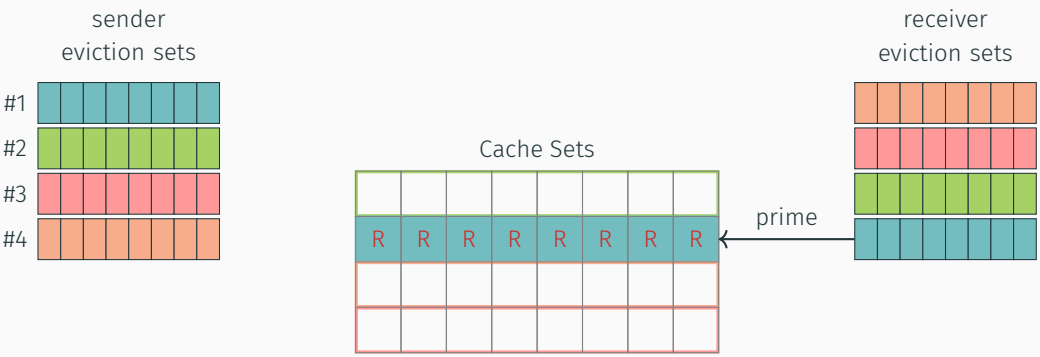
Jamming agreement



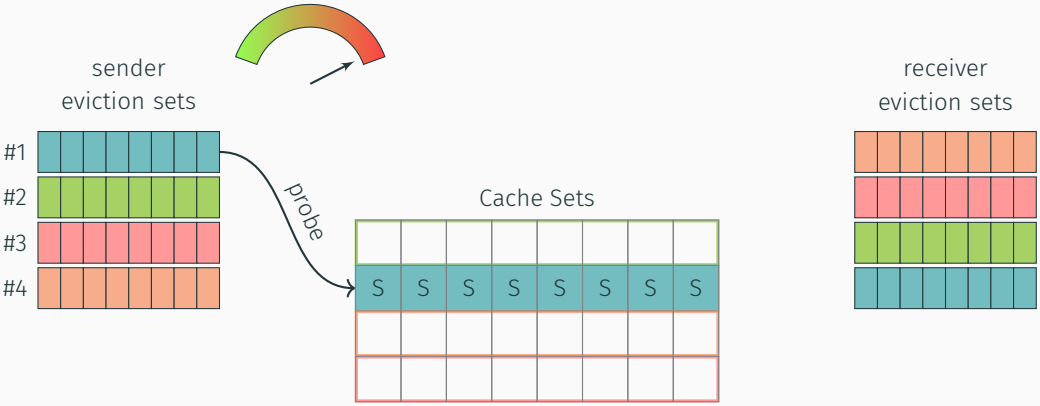
Jamming agreement



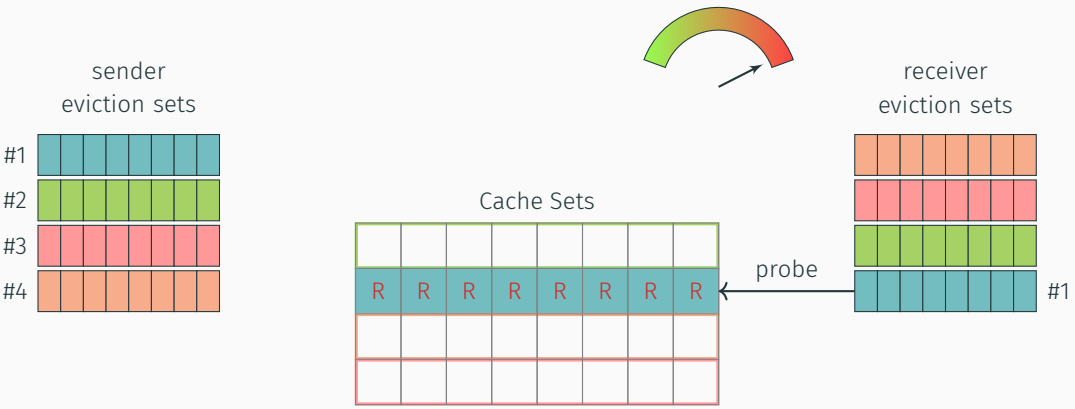
Jamming agreement



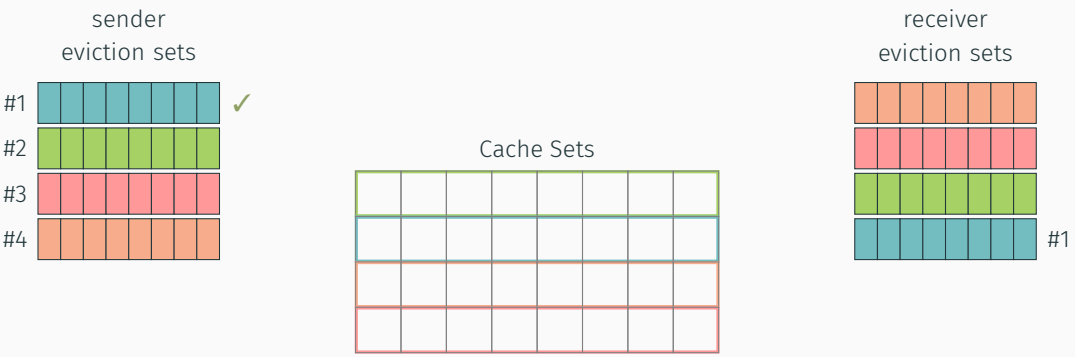
Jamming agreement



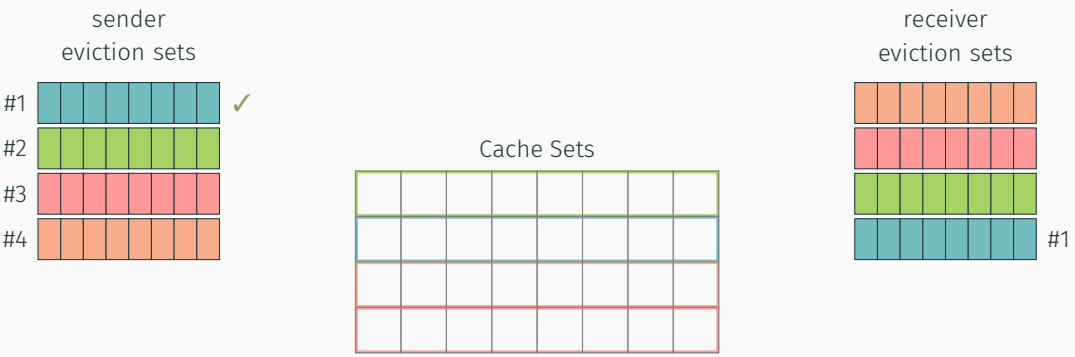
Jamming agreement



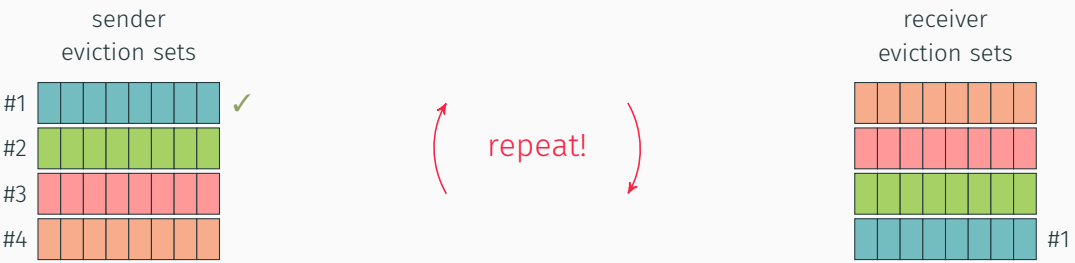
Jamming agreement



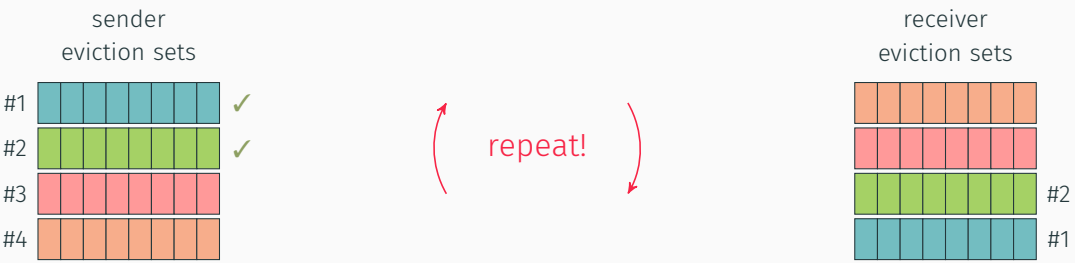
Jamming agreement



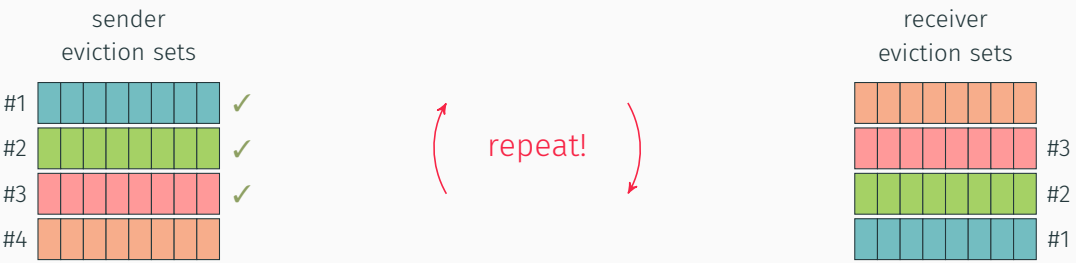
Jamming agreement



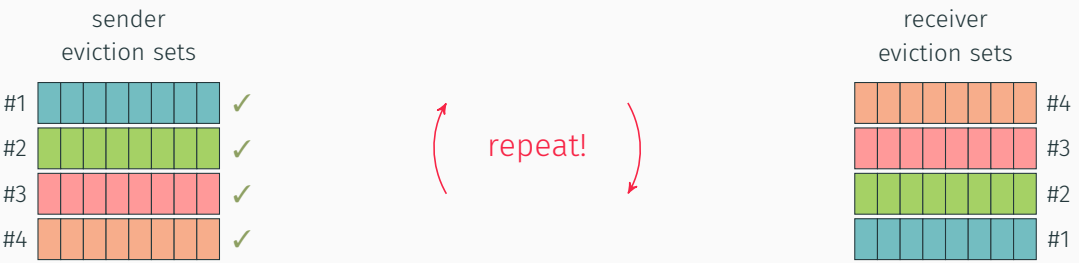
Jamming agreement



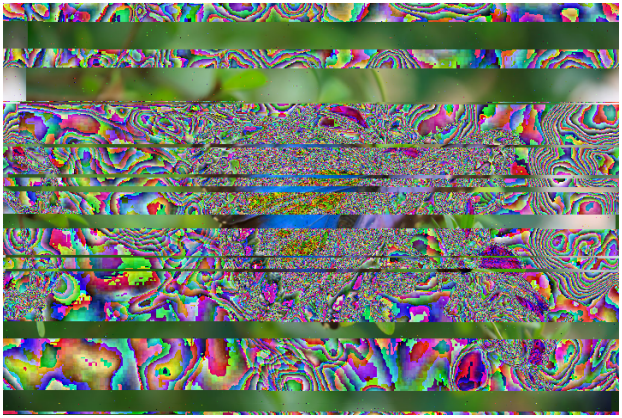
Jamming agreement



Jamming agreement



Sending the first image



Handling synchronization errors

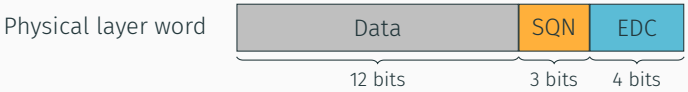
Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)

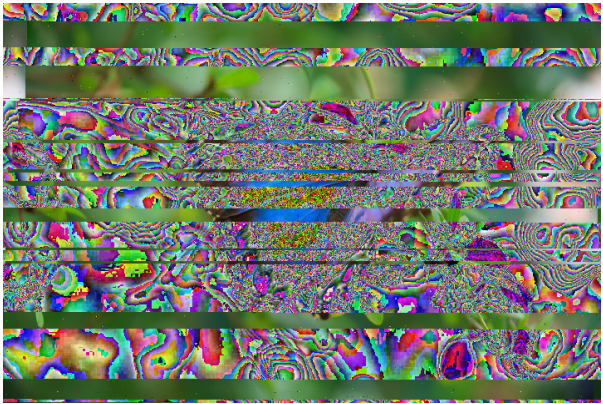
Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)
- '0'-insertion errors: error detection code → Berger codes
  - appending the number of '0's in the word to itself

  → property: a word cannot consist solely of '0's



Synchronization (before)



Synchronization (after)

Synchronization (after)

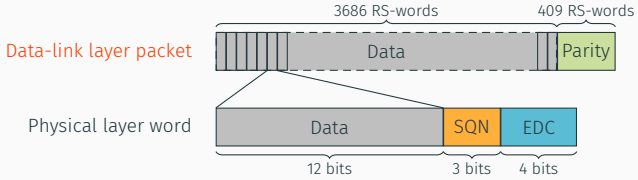
Synchronization (after)

Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors

Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors
- RS word size = physical layer word size = 12 bits
- packet size = $2^{12} - 1 = 4095$ RS words
- 10% error-correcting code: 409 parity and 3686 data RS words



Error correction (after)

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`

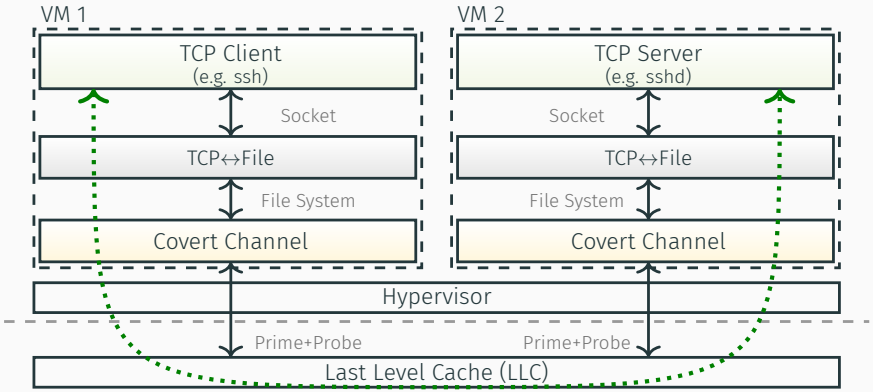
Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`
Amazon EC2	45.25 KBps	0.00%	–

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`
Amazon EC2	45.25 KBps	0.00%	–
Amazon EC2	45.09 KBps	0.00%	web server serving files on sender VM
Amazon EC2	42.96 KBps	0.00%	`stress -m 2` on sender VM
Amazon EC2	42.26 KBps	0.00%	`stress -m 1` on receiver VM
Amazon EC2	37.42 KBps	0.00%	web server on all 3 VMs, `stress -m 4` on 3rd VM, `stress -m 1` on sender and receiver VMs
Amazon EC2	34.27 KBps	0.00%	`stress -m 8` on third VM

Building an SSH connection



SSH evaluation

Between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on SSH server VM	✓
Web server on all VMs	✓
`stress -m 1` on server side	unstable

SSH evaluation

Between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on SSH server VM	✓
Web server on all VMs	✓
`stress -m 1` on server side	unstable

Telnet also works with occasional corrupted bytes with `stress -m 1`

Easy solution #3

Possible side channels using components shared by a CPU?

Easy solution #3

Possible side channels using components shared by a CPU?

Stop sharing a CPU!

Cross-CPU Side-Channel Attacks

What if?

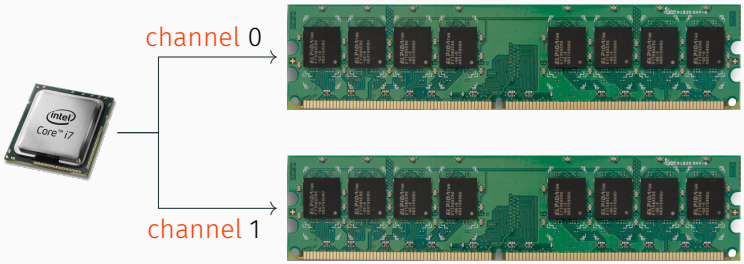
- VMs located on the same physical machine, but on separate CPUs?

What if?

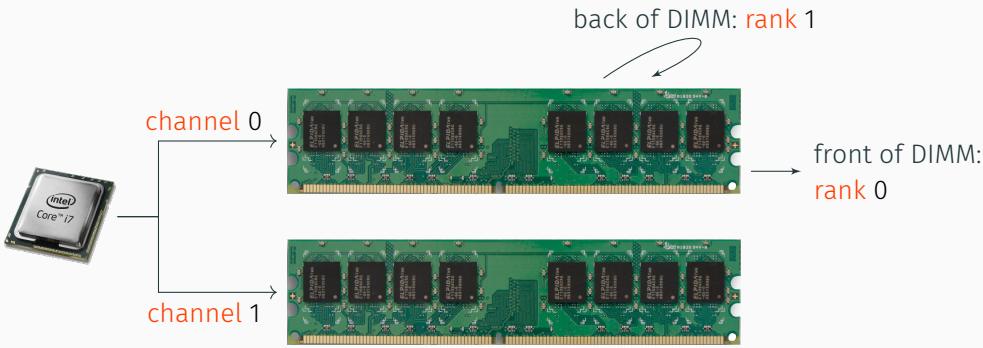
- VMs located on the same physical machine, but on separate CPUs?
- they share some DRAM!

DRAM organization

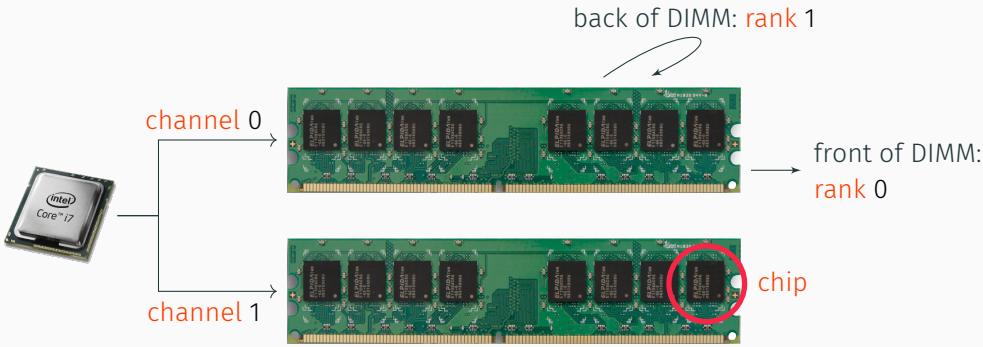
DRAM organization



DRAM organization

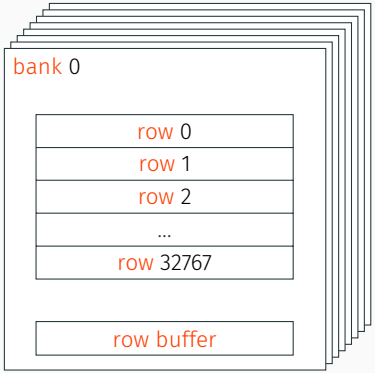


DRAM organization



DRAM organization

chip



→ bits in cells in rows

DRAM row buffer

- DRAM internally is only capable of reading entire rows

DRAM row buffer

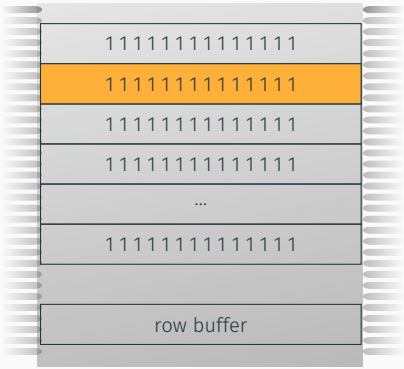
- DRAM internally is only capable of reading entire rows
- capacitors in cells discharge when you “read the bits”
- buffer the bits when reading them from the cells
- write the bits back to the cells when you’re done

DRAM row buffer

- DRAM internally is only capable of reading entire rows
- capacitors in cells discharge when you “read the bits”
- buffer the bits when reading them from the cells
- write the bits back to the cells when you’re done

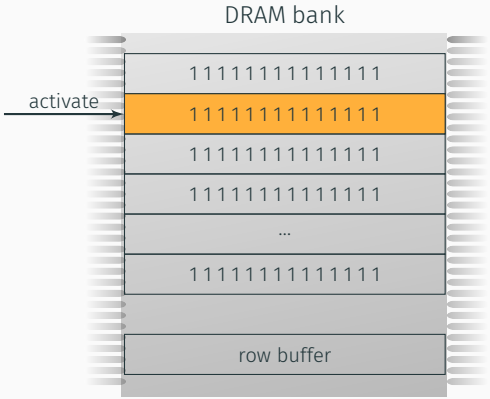
→ row buffer

How reading from DRAM works



CPU wants to access row 1

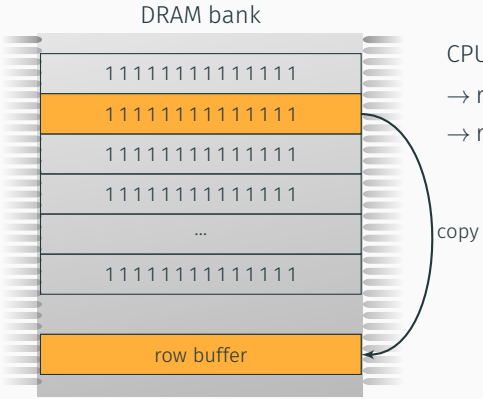
How reading from DRAM works



CPU wants to access row 1

→ row 1 activated

How reading from DRAM works

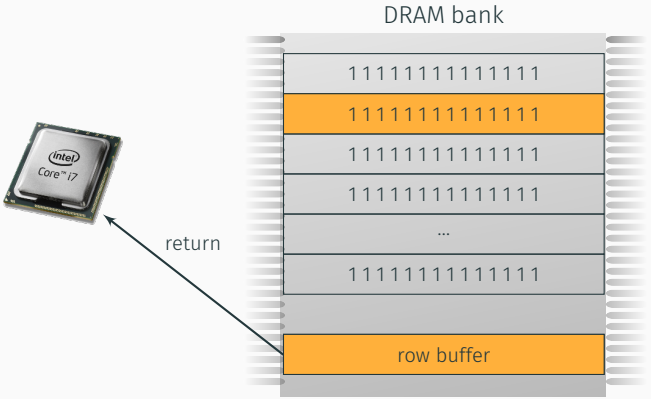


CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer

How reading from DRAM works

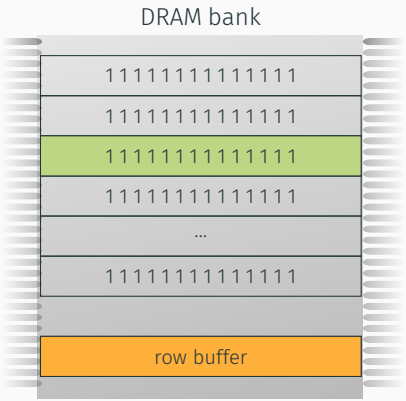


CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer

How reading from DRAM works



CPU wants to access row 2

How reading from DRAM works

activate

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1

...

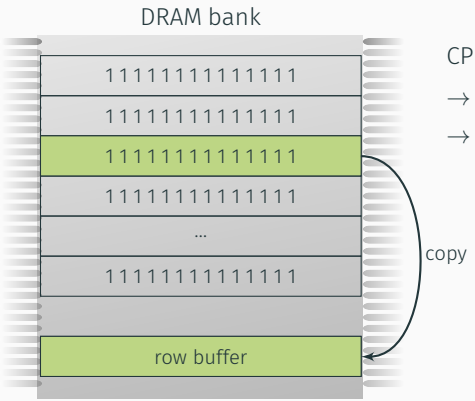
1 1 1 1 1 1 1 1 1 1 1 1 1 1

row buffer

CPU wants to access row 2

$\rightarrow$ row 2 activated

How reading from DRAM works

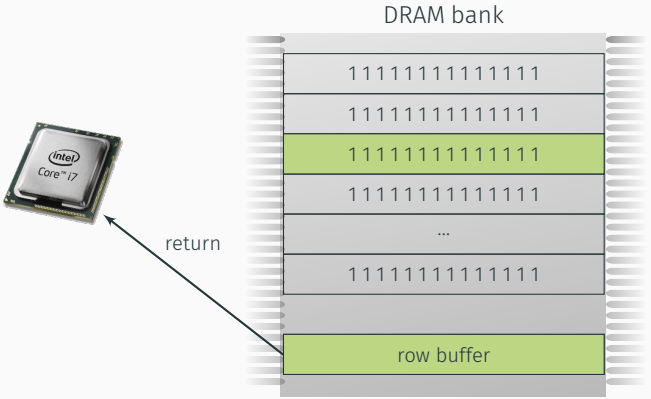


CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

How reading from DRAM works

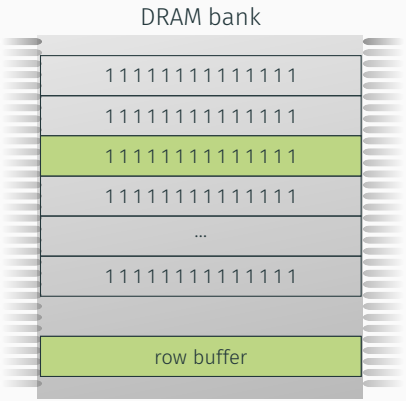


CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

How reading from DRAM works



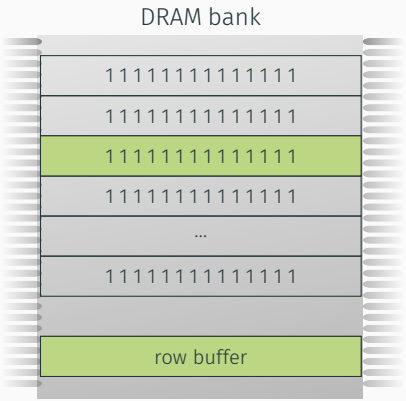
CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

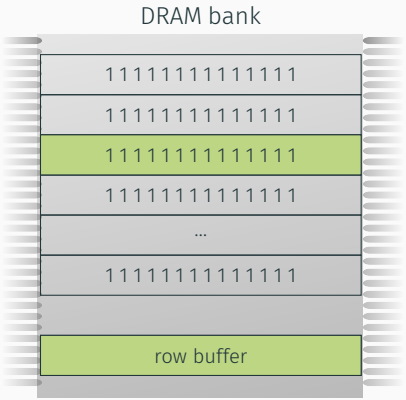
→ slow (row conflict)

How reading from DRAM works



CPU wants to access row 2—again

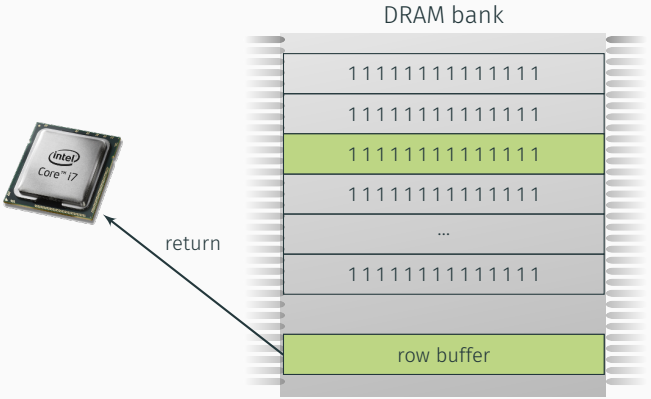
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

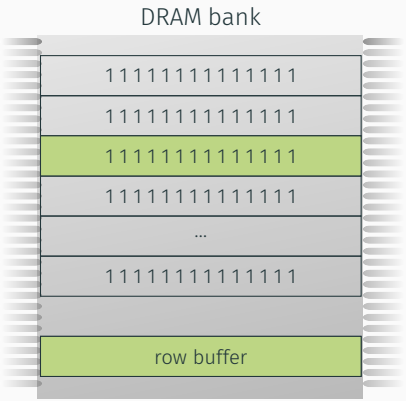
How reading from DRAM works



CPU wants to access row 2—again

$\rightarrow$ row 2 already in row buffer

How reading from DRAM works

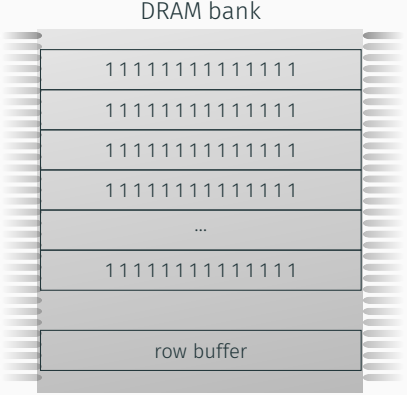


CPU wants to access row 2—again

→ row 2 already in row buffer

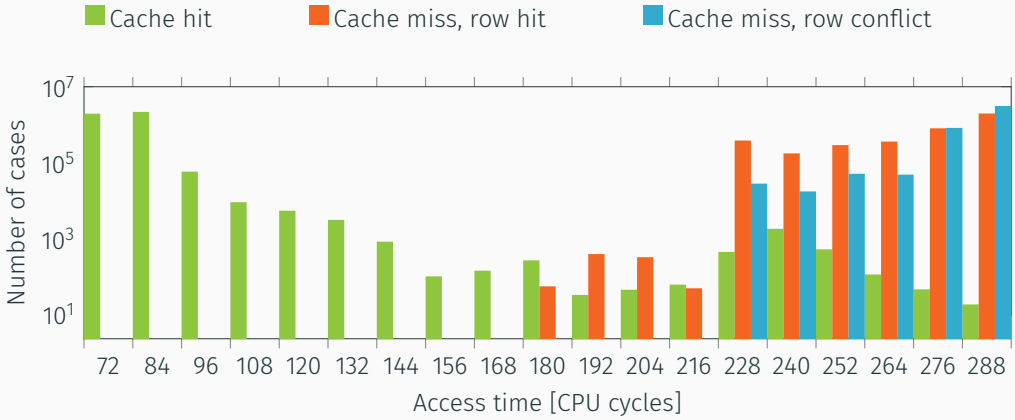
→ fast (row hit)

How reading from DRAM works

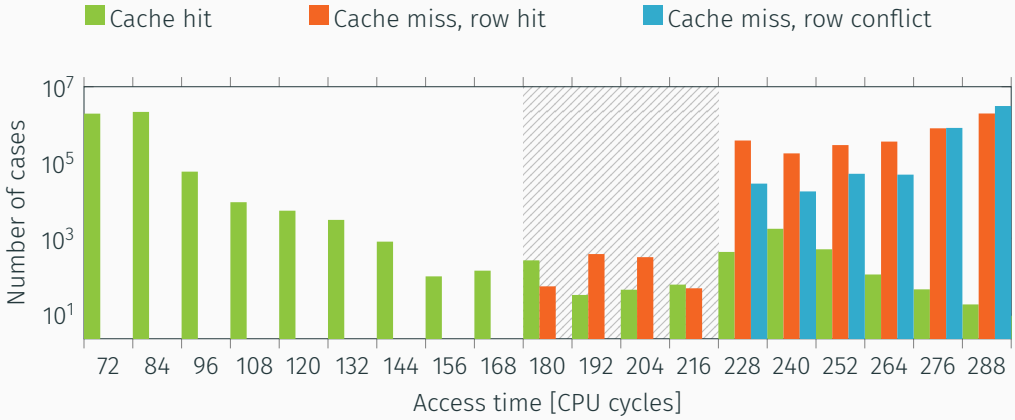


row buffer = cache

DRAM timing differences



DRAM timing differences



DRAM side channels?

- row buffers are caches

DRAM side channels?

- row buffers are caches
- we can observe timing differences

DRAM side channels?

- row buffers are caches
- we can observe timing differences
- how to exploit these timing differences?

DRAM side channels?

- row buffers are caches
- we can observe timing differences
- how to exploit these timing differences?
- target addresses in the same channel, rank and bank

DRAM side channels?

- row buffers are caches
- we can observe timing differences
- how to exploit these timing differences?
- target addresses in the same channel, rank and bank
- but DRAM mapping functions are undocumented

DRAM side channels?

- row buffers are caches
- we can observe timing differences
- how to exploit these timing differences?
- target addresses in the same channel, rank and bank
- but DRAM mapping functions are undocumented

→ we reverse-engineered them!

P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security Symposium*. 2016

DRAMA: DRAM Addressing attacks

- infer behavior from memory accesses similarly to cache attacks

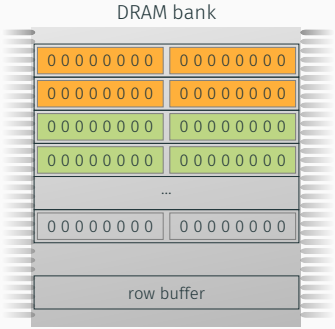
DRAMA: DRAM Addressing attacks

- infer behavior from memory accesses similarly to cache attacks
- works across VMs, across cores, across CPUs

DRAMA: DRAM Addressing attacks

- infer behavior from memory accesses similarly to cache attacks
- works across VMs, across cores, across CPUs
- covert channels and side-channel attacks

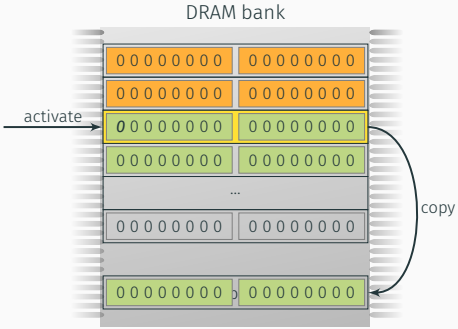
DRAMA covert channel



sender and receiver agree on one bank

receiver continuously accesses a row *i*

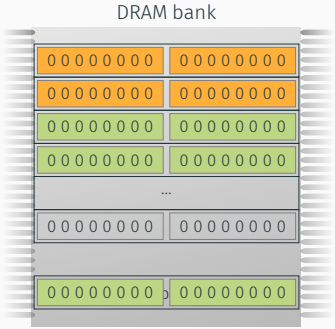
DRAMA covert channel



sender and receiver agree on one bank

receiver continuously accesses a row *i*

DRAMA covert channel

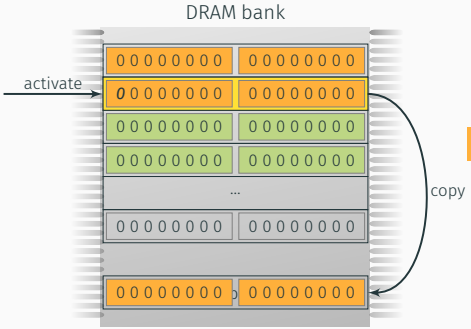


sender and receiver agree on one bank

receiver continuously accesses a row *i*

case #1: sender transmits 1

DRAMA covert channel



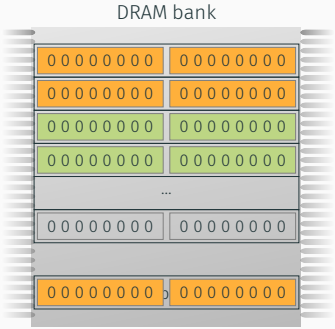
sender and receiver agree on one bank

receiver continuously accesses a row i

case #1: sender transmits 1

sender accesses row $j \neq i$

DRAMA covert channel



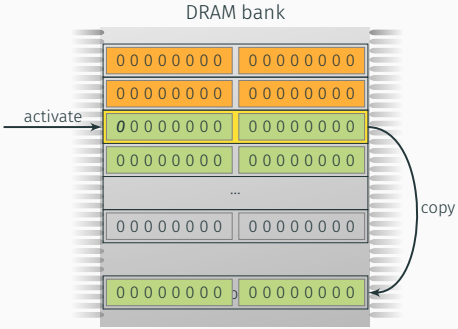
sender and receiver agree on one bank

receiver continuously accesses a row i

case #1: sender transmits 1

sender accesses row $j \neq i$

DRAMA covert channel



sender and receiver agree on one bank

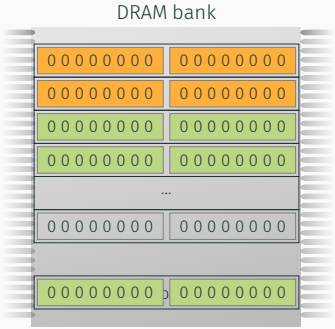
receiver continuously accesses a row i

case #1: sender transmits 1

sender accesses row $j \neq i$

next receiver access $\rightarrow$ copy row buffer

DRAMA covert channel



sender and receiver agree on one bank

receiver continuously accesses a row i

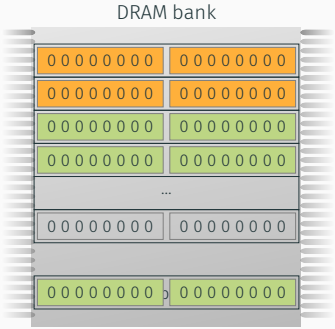
case #1: sender transmits 1

sender accesses row $j \neq i$

next receiver access $\rightarrow$ copy row buffer

$\rightarrow$ **slow**

DRAMA covert channel

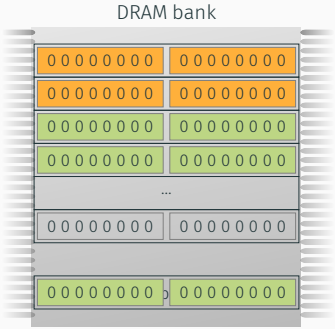


sender and receiver agree on one bank

receiver continuously accesses a row *i*

case #2: sender transmits 0

DRAMA covert channel



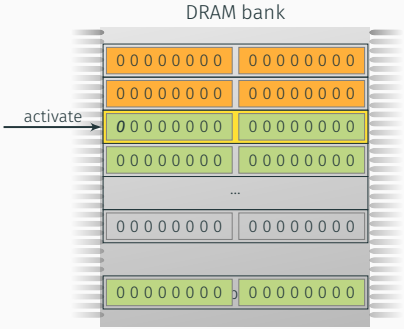
sender and receiver agree on one bank

receiver continuously accesses a row *i*

case #2: sender transmits 0

sender does nothing

DRAMA covert channel



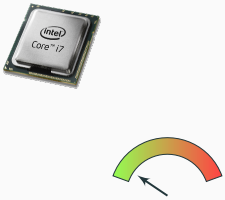
sender and receiver agree on one bank

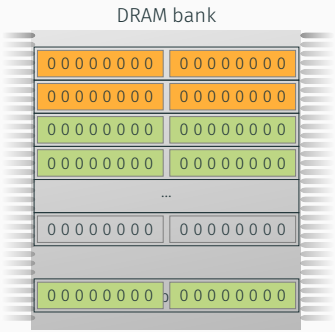
receiver continuously accesses a row *i*

case #2: sender transmits 0

sender does nothing
next receiver access → already in buffer

DRAMA covert channel





sender and receiver agree on one bank

receiver continuously accesses a row *i*

case #2: sender transmits 0

sender does nothing

next receiver access → already in buffer

→ **fast**

DRAMA covert channel: Evaluation

- native environment: 1.6 Mbps
- cross-VM: 596 Kbps

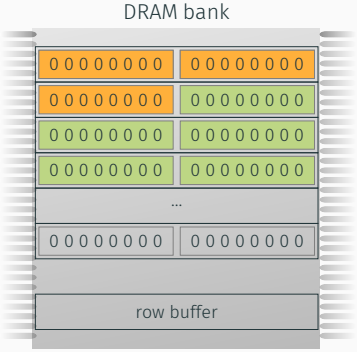
DRAMA covert channel: Evaluation

- native environment: 1.6 Mbps
- cross-VM: 596 Kbps
- also implemented in JavaScript

DRAMA covert channel: Evaluation

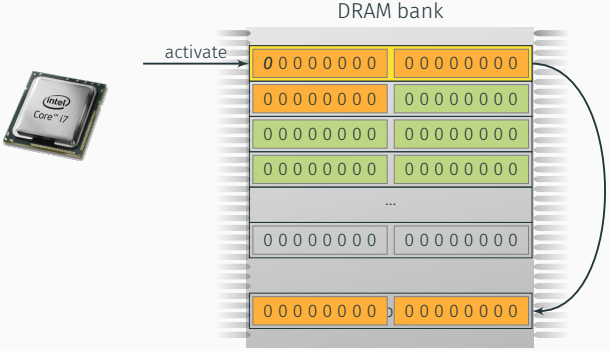
- native environment: 1.6 Mbps
- cross-VM: 596 Kbps
- also implemented in JavaScript
  - sender inside a VM
  - receiver runs in JavaScript in the browser on the host
  - 11 bps

DRAMA side-channel attacks



spy and victim share a row *i*

DRAMA side-channel attacks

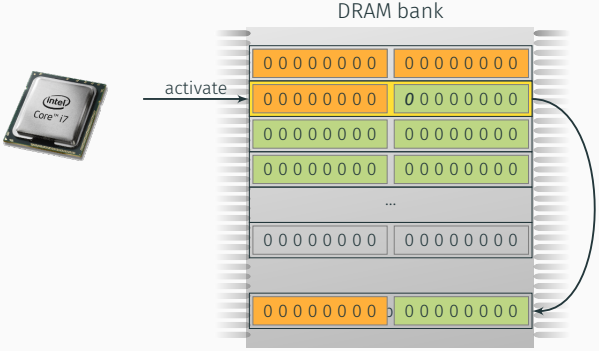


spy and victim share a row i

case #1

spy accesses row $j \neq i$, copy to row buffer

DRAMA side-channel attacks



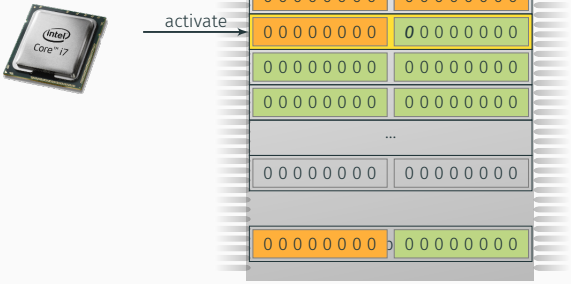
spy and victim share a row i

case #1

spy accesses row $j \neq i$, copy to row buffer

victim accesses row i, copy to row buffer

DRAMA side-channel attacks



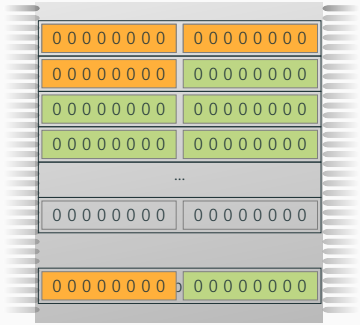
spy and victim share a row i

case #1

spy accesses row $j \neq i$, copy to row buffer
victim accesses row i, copy to row buffer
spy accesses row i, no copy

DRAMA side-channel attacks

DRAM bank



spy and victim share a row i

case #1

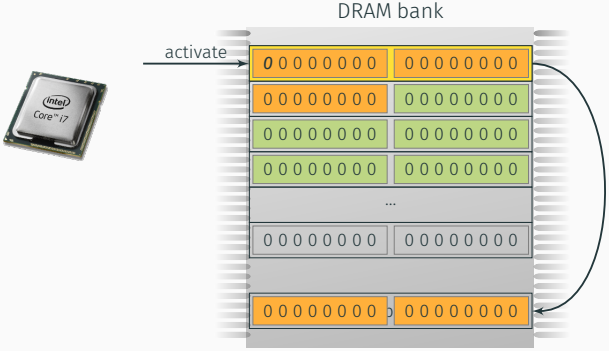
spy accesses row $j \neq i$, copy to row buffer

victim accesses row i, copy to row buffer

spy accesses row i, no copy

$\rightarrow$ **fast**

DRAMA side-channel attacks



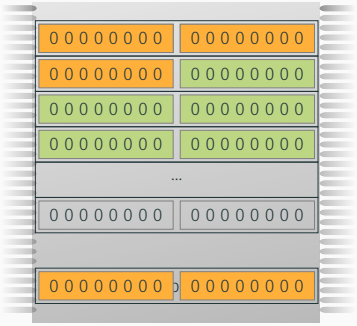
spy and victim share a row i

case #2

spy accesses row $j \neq i$, copy to row buffer

DRAMA side-channel attacks

DRAM bank



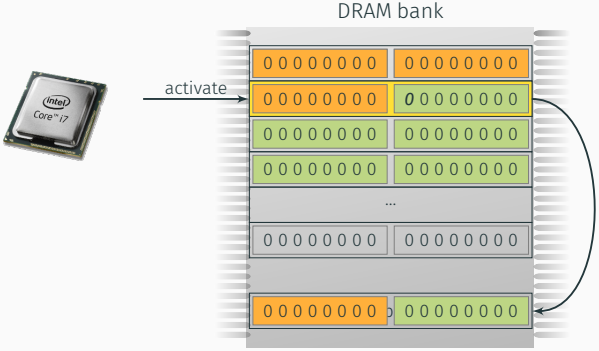
spy and victim share a row i

case #2

spy accesses row $j \neq i$, copy to row buffer

no victim access on row i

DRAMA side-channel attacks



spy and victim share a row *i*

case #2

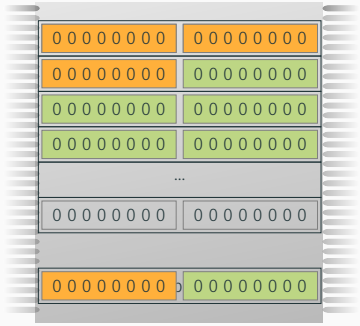
spy accesses row $j \neq i$, copy to row buffer

no victim access on row *i*

spy accesses row *i*, copy to row buffer

DRAMA side-channel attacks

DRAM bank



spy and victim share a row i

case #2

spy accesses row $j \neq i$, copy to row buffer

no victim access on row i

spy accesses row i, copy to row buffer

$\rightarrow$ **slow**

Two related questions

- what is the chance we can share a row with important victim data?

Two related questions

- what is the chance we can share a row with important victim data?
- what kind of spatial accuracy will we get?

Sharing a DRAM row?

- the smallest unit of physical memory is one page

Sharing a DRAM row?

- the smallest unit of physical memory is one page
- pages are usually 4 KB

Sharing a DRAM row?

- the smallest unit of physical memory is one page
- pages are usually 4 KB
- DRAM rows are usually 8 KB

Sharing a DRAM row?

- the smallest unit of physical memory is one page
- pages are usually 4 KB
- DRAM rows are usually 8 KB
- we need the victim's address and our address in the same row

Sharing a DRAM row?

- the smallest unit of physical memory is one page
- pages are usually 4 KB
- DRAM rows are usually 8 KB
- we need the victim's address and our address in the same row
- if you say that two pages share one row you are not wrong...

Sharing a DRAM row?

- the smallest unit of physical memory is one page
- pages are usually 4 KB
- DRAM rows are usually 8 KB
- we need the victim's address and our address in the same row
- if you say that two pages share one row you are not wrong...
- ... but not right either

Sharing a DRAM row?

- the smallest unit of physical memory is one page
- pages are usually 4 KB
- DRAM rows are usually 8 KB
- we need the victim's address and our address in the same row
- if you say that two pages share one row you are not wrong...
- ... but not right either
- why?

Accuracy (1)

- a 4 KB page might not be in a single row

Accuracy (1)

- a 4 KB page might not be in a single row
- depending on the functions, a page can be distributed over multiple rows

Accuracy (1)

- a 4 KB page might not be in a single row
- depending on the functions, a page can be distributed over multiple rows
- this is the case if address bits 0 to 11 are used for the mapping

Accuracy (1)

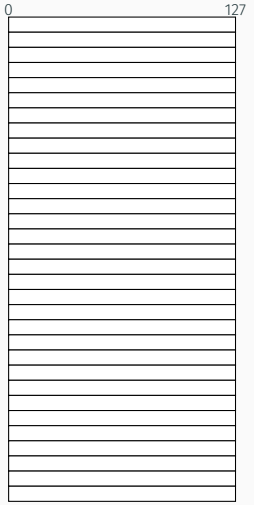
- a 4 KB page might not be in a single row
- depending on the functions, a page can be distributed over multiple rows
- this is the case if address bits 0 to 11 are used for the mapping
- e.g., Skylake uses low bits for channel and bank group

Accuracy (1)

- a 4 KB page might not be in a single row
- depending on the functions, a page can be distributed over multiple rows
- this is the case if address bits 0 to 11 are used for the mapping
- e.g., Skylake uses low bits for channel and bank group

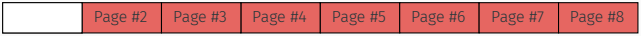
$\rightarrow$ one physical page is distributed over 4 rows

Accuracy (2)



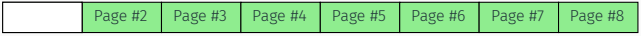
8 KB row *x* in BG0 (1) and channel (1)

8 KB row *x* in BG0 (0) and channel (1)

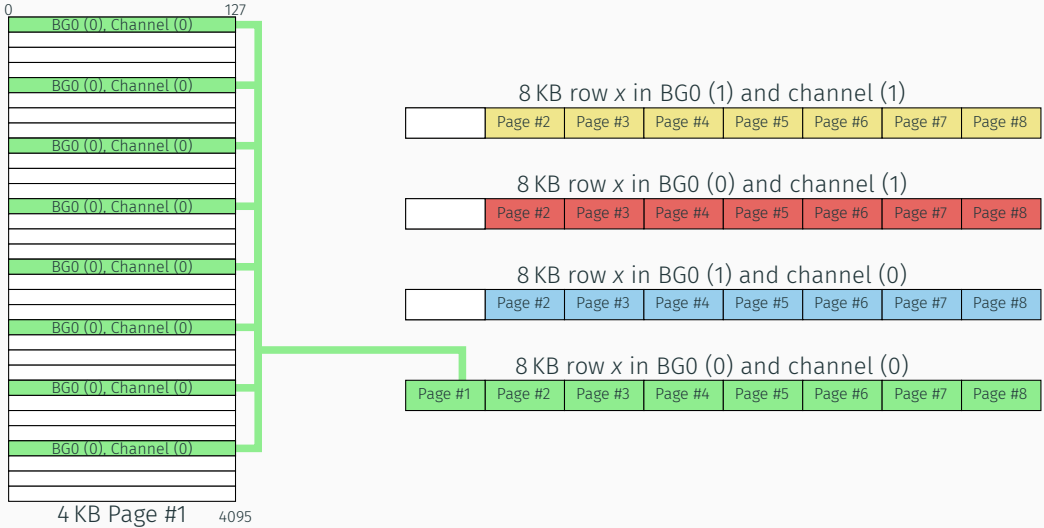


8 KB row *x* in BG0 (1) and channel (0)

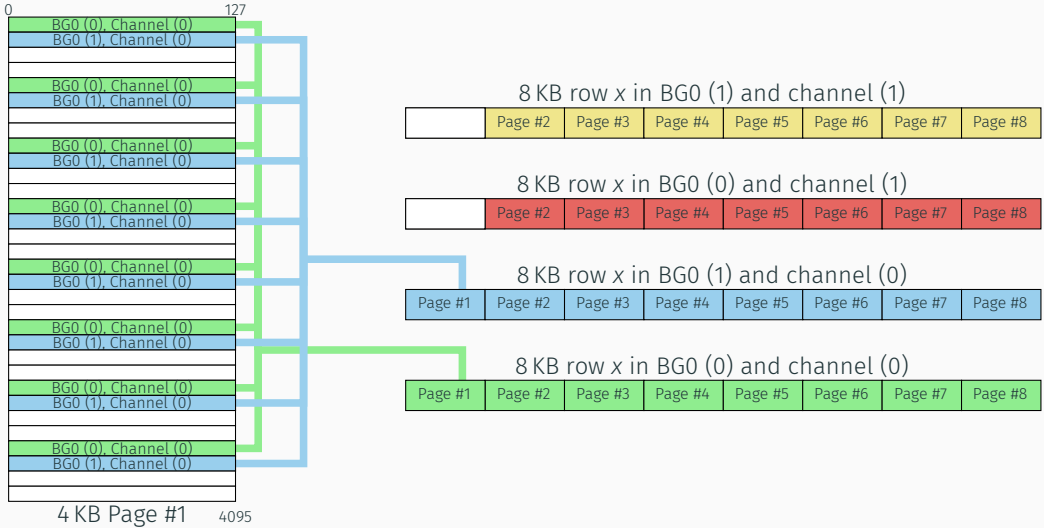
8 KB row *x* in BG0 (0) and channel (0)



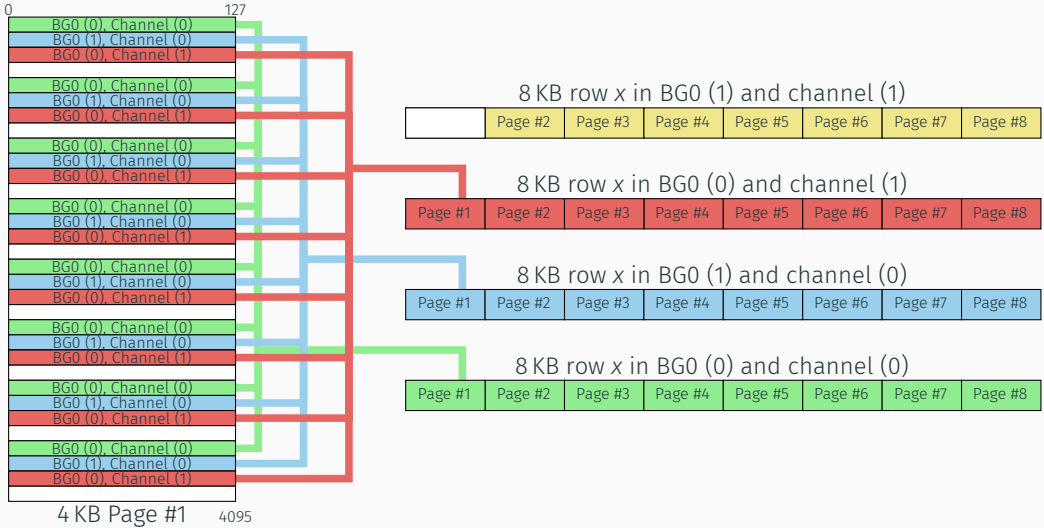
Accuracy (2)


0
127
BG0 (0), Channel (0)
BG0 (0), Channel (0)
BG0 (0), Channel (0)
BG0 (0), Channel (0)
BG0 (0), Channel (0)
BG0 (0), Channel (0)
BG0 (0), Channel (0)
BG0 (0), Channel (0)
4 KB Page #1
4095
8 KB row x in BG0 (1) and channel (1)
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8
8 KB row x in BG0 (0) and channel (1)
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8
8 KB row x in BG0 (1) and channel (0)
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8
8 KB row x in BG0 (0) and channel (0)
Page #1
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8

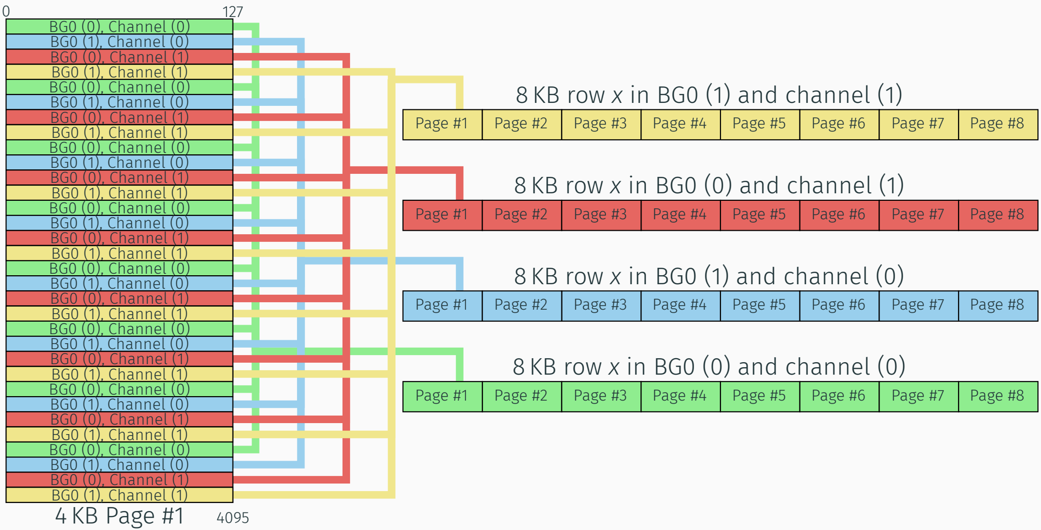
Accuracy (2)


0
127
BG0 (0), Channel (0)
BG0 (1), Channel (0)
4 KB Page #1
4095
8 KB row x in BG0 (1) and channel (1)
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8
8 KB row x in BG0 (0) and channel (1)
8 KB row x in BG0 (1) and channel (0)
Page #1
8 KB row x in BG0 (0) and channel (0)

Accuracy (2)


0
127
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
4 KB Page #1
4095
8 KB row x in BG0 (1) and channel (1)
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8
8 KB row x in BG0 (0) and channel (1)
Page #1
8 KB row x in BG0 (1) and channel (0)
8 KB row x in BG0 (0) and channel (0)

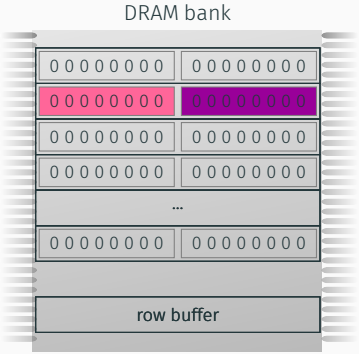
Accuracy (2)


0
127
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (1)
4 KB Page #1
4095
8 KB row *x* in BG0 (1) and channel (1)
8 KB row *x* in BG0 (0) and channel (1)
8 KB row *x* in BG0 (1) and channel (0)
8 KB row *x* in BG0 (0) and channel (0)
Page #1
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8

Accuracy (3)

Number of pages per row depends on DRAM configuration and CPU architecture

Sandy Bridge /w 1 DIMM



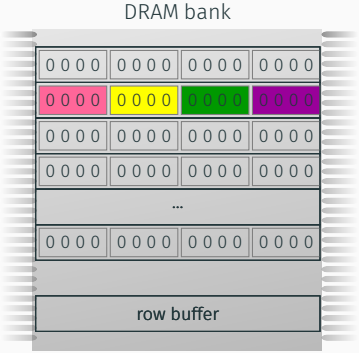
Sandy Bridge /w 1 DIMM

→ 2 pages per row

Accuracy (3)

Number of pages per row depends on DRAM configuration and CPU architecture

Ivy Bridge /w 2 DIMM



Sandy Bridge /w 1 DIMM

→ 2 pages per row

Ivy Bridge /w 2 DIMM

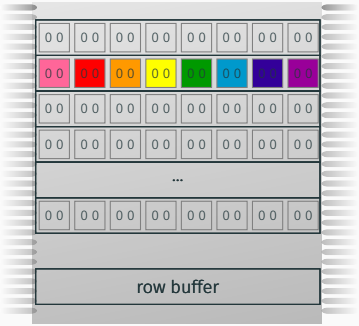
→ 4 pages per row

Accuracy (3)

Number of pages per row depends on DRAM configuration and CPU architecture

Skylake /w 2 DIMM

DRAM bank



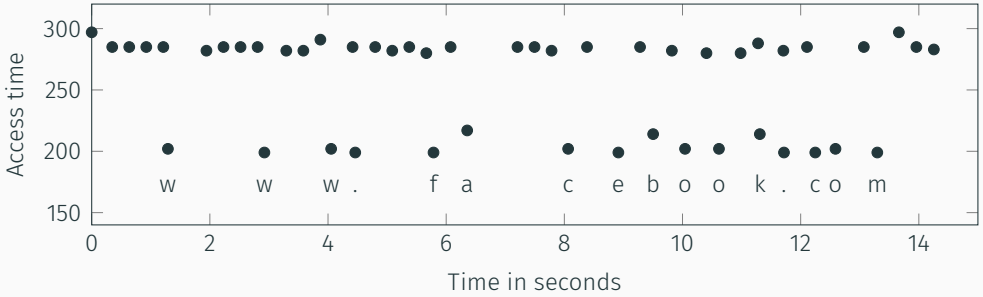
Sandy Bridge /w 1 DIMM
→ 2 pages per row

Ivy Bridge /w 2 DIMM
→ 4 pages per row

Skylake /w 2 DIMM
→ 8 pages per row

Spying on keystrokes

- side-channel: template attack
  - allocate a large fraction of memory to be in a row with the victim
  - profile memory and record row-hit ratio for each address



Easy solution #4

Possible side channels using components shared by a machine?

Easy solution #4

Possible side channels using components shared by a machine?

Now what?

There’s more!

There's more!

Sharing hardware does not only happen in the cloud

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications". In: *CCS'15*. 2015

M. Schwarz, C. Maurice, D. Gruss, and S. Mangard. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript ". In: *FC'17*. 2017

M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016

M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: *DIMVA'17*. To appear. 2017

There's more!

Sharing hardware does not only happen in the cloud

- JavaScript is essentially code execution

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications". In: *CCS'15*. 2015

M. Schwarz, C. Maurice, D. Gruss, and S. Mangard. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript ". In: *FC'17*. 2017

M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016

M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: *DIMVA'17*. To appear. 2017

There's more!

Sharing hardware does not only happen in the cloud

- JavaScript is essentially code execution
- same problems on mobile devices

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications". In: *CCS'15*. 2015

M. Schwarz, C. Maurice, D. Gruss, and S. Mangard. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript ". In: *FC'17*. 2017

M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016

M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: *DIMVA'17*. To appear. 2017

There's more!

Sharing hardware does not only happen in the cloud

- JavaScript is essentially code execution
- same problems on mobile devices
- and between SGX enclaves

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications". In: *CCS'15*. 2015

M. Schwarz, C. Maurice, D. Gruss, and S. Mangard. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript ". In: *FC'17*. 2017

M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016

M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: *DIMVA'17*. To appear. 2017

On the difficulty of finding countermeasures

- cryptographic algorithms

C. Pereida García, B. B. Brumley, and Y. Yarom. “Make Sure DSA Signing Exponentiations Really are Constant-Time”. In: *CCS'16*. 2016

On the difficulty of finding countermeasures

- cryptographic algorithms $\rightarrow$ today we know how to write good algorithms

C. Pereida García, B. B. Brumley, and Y. Yarom. "Make Sure DSA Signing Exponentiations Really are Constant-Time". In: *CCS'16*. 2016

On the difficulty of finding countermeasures

- cryptographic algorithms $\rightarrow$ today we know how to write good algorithms
  - most of the time

C. Pereida García, B. B. Brumley, and Y. Yarom. "Make Sure DSA Signing Exponentiations Really are Constant-Time". In: *CCS'16*. 2016

On the difficulty of finding countermeasures

- cryptographic algorithms → today we know how to write good algorithms
  - most of the time
- other problems: keystroke timing attacks, attacks against ASLR...

C. Pereida García, B. B. Brumley, and Y. Yarom. "Make Sure DSA Signing Exponentiations Really are Constant-Time". In: *CCS'16*. 2016

On the difficulty of finding countermeasures

- cryptographic algorithms → today we know how to write good algorithms
  - most of the time
- other problems: keystroke timing attacks, attacks against ASLR...
- no hardware sharing is not an option

C. Pereida García, B. B. Brumley, and Y. Yarom. "Make Sure DSA Signing Exponentiations Really are Constant-Time". In: *CCS'16*. 2016

On the difficulty of finding countermeasures

- cryptographic algorithms → today we know how to write good algorithms
  - most of the time
- other problems: keystroke timing attacks, attacks against ASLR...
- no hardware sharing is not an option

How to counter attacks based on hardware optimizations without decreasing performance brought by these optimizations?

C. Pereida García, B. B. Brumley, and Y. Yarom. "Make Sure DSA Signing Exponentiations Really are Constant-Time". In: *CCS'16*. 2016

Conclusion

Conclusion

- more a problem of CPU design than Instruction Set Architecture

---

http://cs.adelaide.edu.au/~yval/Mastik/

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to find → lots of undocumented hardware

---

http://cs.adelaide.edu.au/~yval/Mastik/

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to find $\rightarrow$ lots of undocumented hardware
- not that complicated to run $\rightarrow$ more automated attacks and frameworks

---

`http://cs.adelaide.edu.au/~yval/Mastik/`

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to find → lots of undocumented hardware
- not that complicated to run → more automated attacks and frameworks
- hard to patch → issues linked to performance optimizations

http://cs.adelaide.edu.au/~yval/Mastik/

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to find → lots of undocumented hardware
- not that complicated to run → more automated attacks and frameworks
- hard to patch → issues linked to performance optimizations
- quick fixes don't work

---

http://cs.adelaide.edu.au/~yval/Mastik/

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to find → lots of undocumented hardware
- not that complicated to run → more automated attacks and frameworks
- hard to patch → issues linked to performance optimizations
- quick fixes don't work
- more work needed before having satisfying solutions

---

http://cs.adelaide.edu.au/~yval/Mastik/

Thank you!

Contact

- clementine@cmaurice.fr
- @BloodyTangerine

Microarchitectural Attacks in the Cloud

Clémentine Maurice

April 23, 2017—Workshop on Security and Dependability of Multi-Domain Infrastructures, Belgrade, Serbia