A journey into the secret flaws of in-DRAM RowHammer mitigations

Emanuele Vannacci
Pietro Frigo

Vrije Universiteit Amsterdam

Who are we?

Emanuele Vannacci (@vanema94)
Pietro Frigo (@pit_frg)

- PhD students @VUSec
- Extensive experience with RowHammer

VUSec

What's it about?

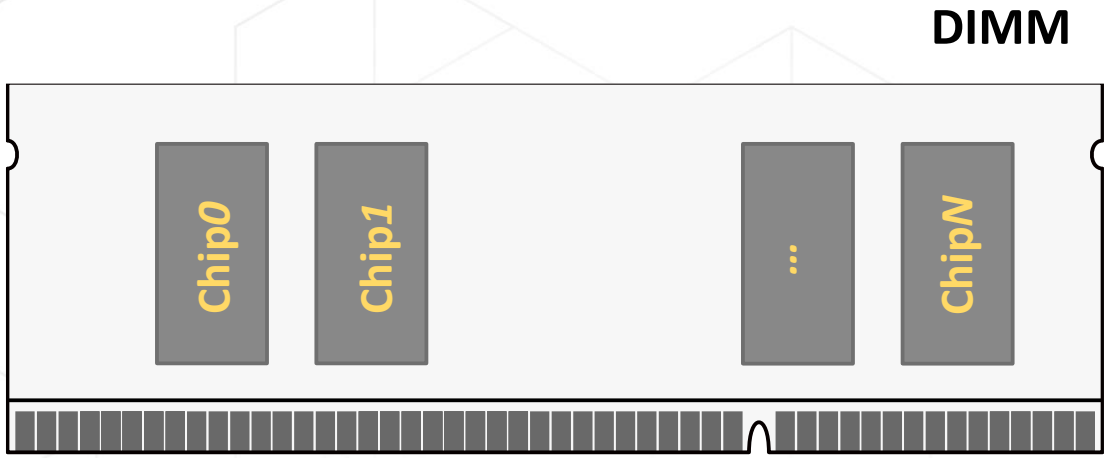
RowHammer


Defenses vol. 1&2

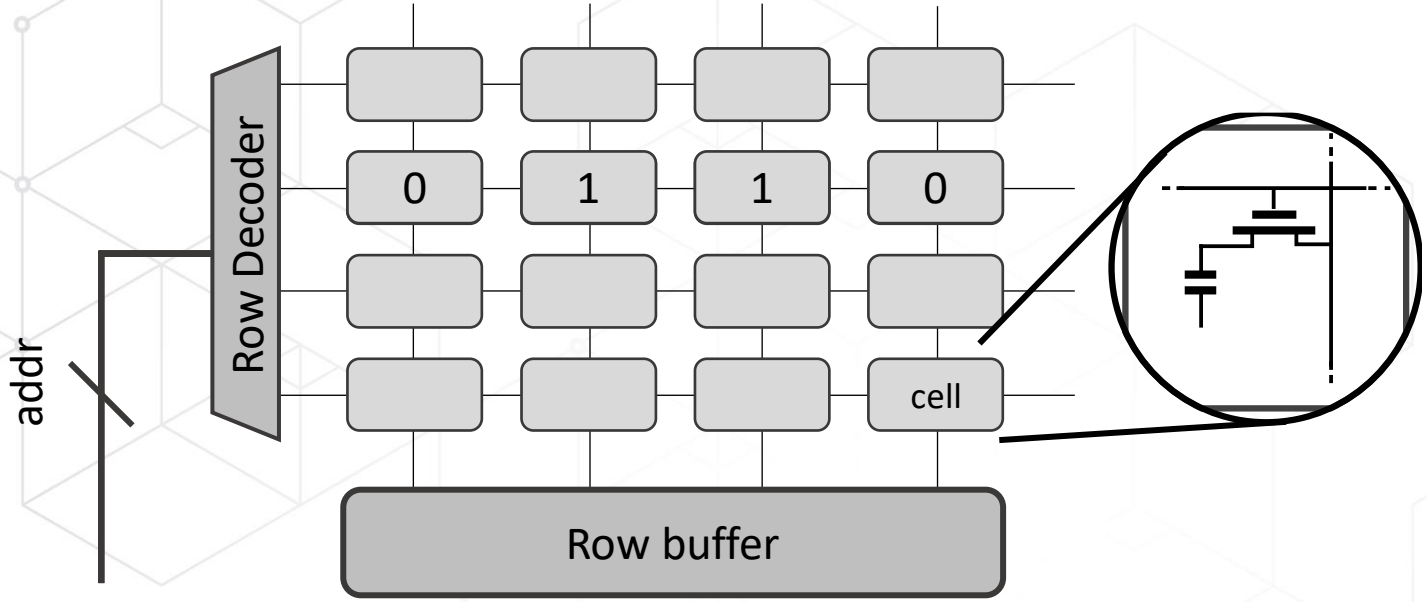

in-DRAM DDR4 mitigations


How we broke them

DRAM

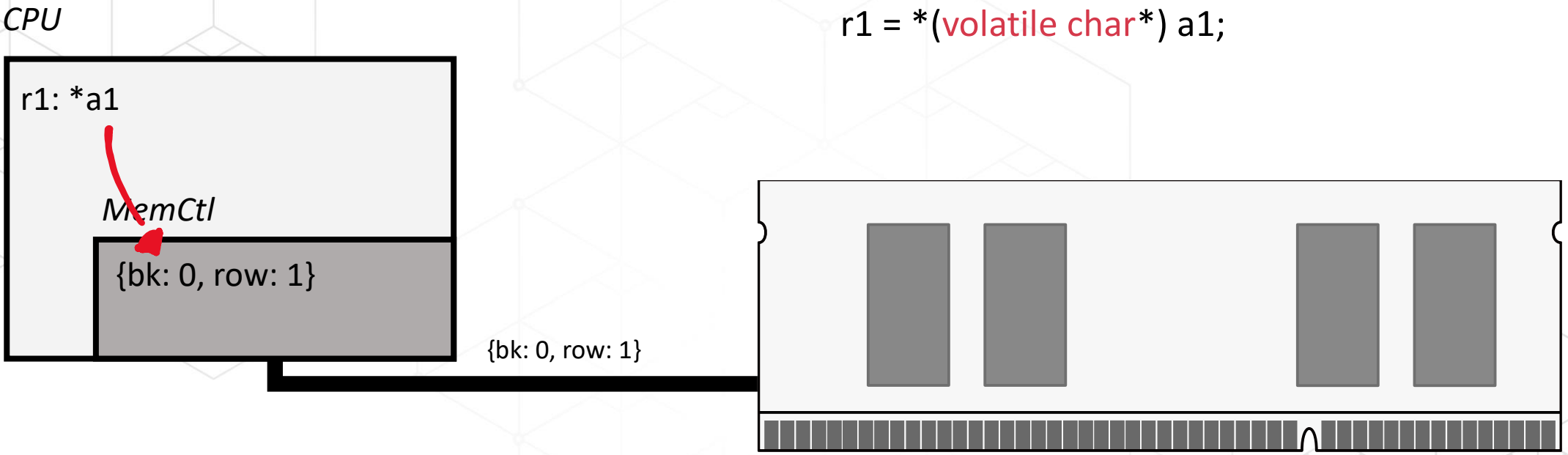


DRAM - Bank

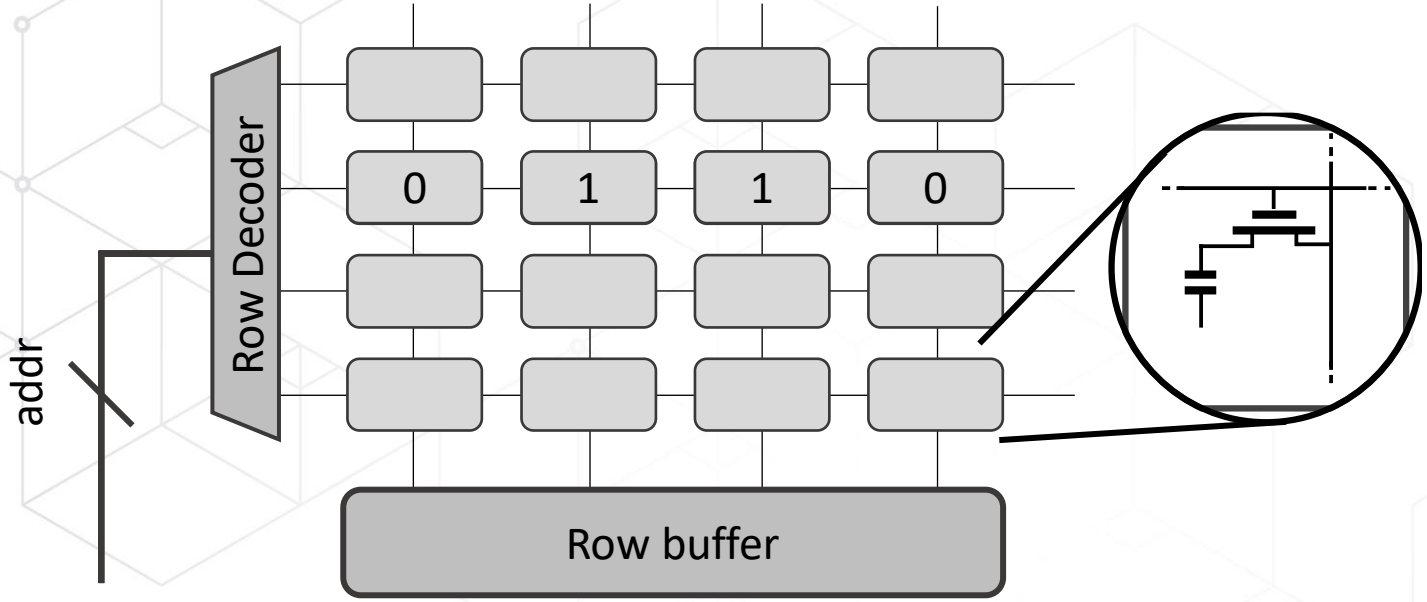


DRAM

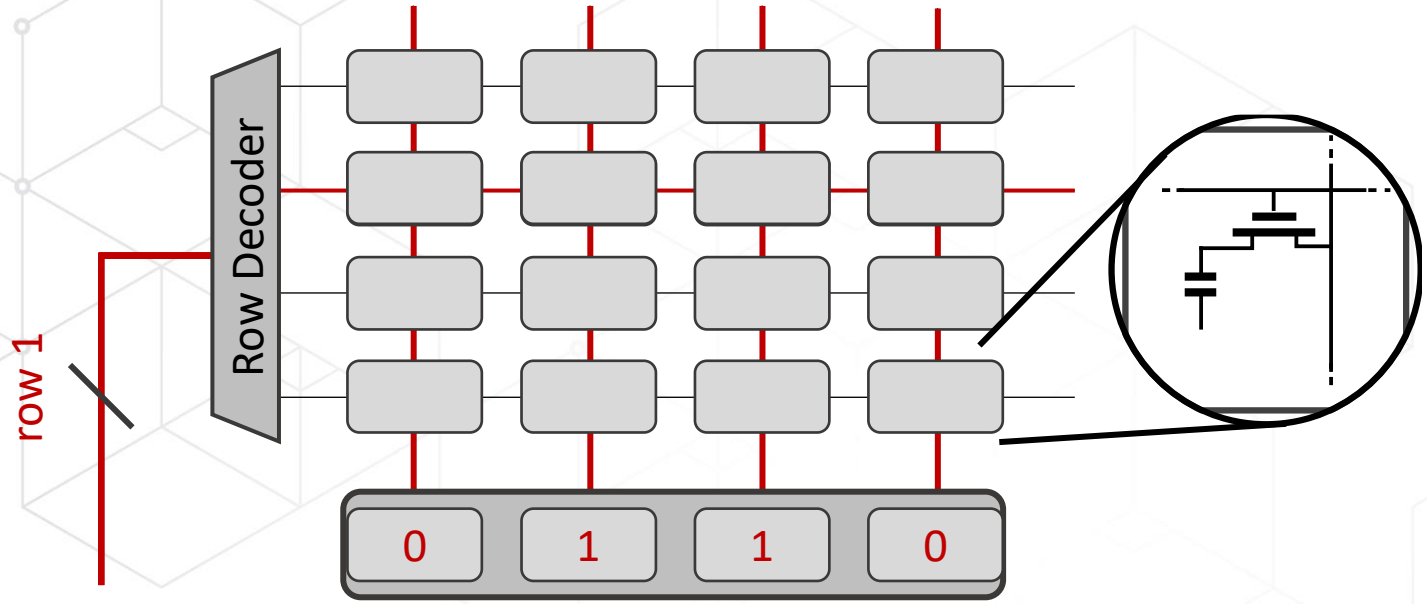
r1 = *(volatile char*) a1;



DRAM - Bank

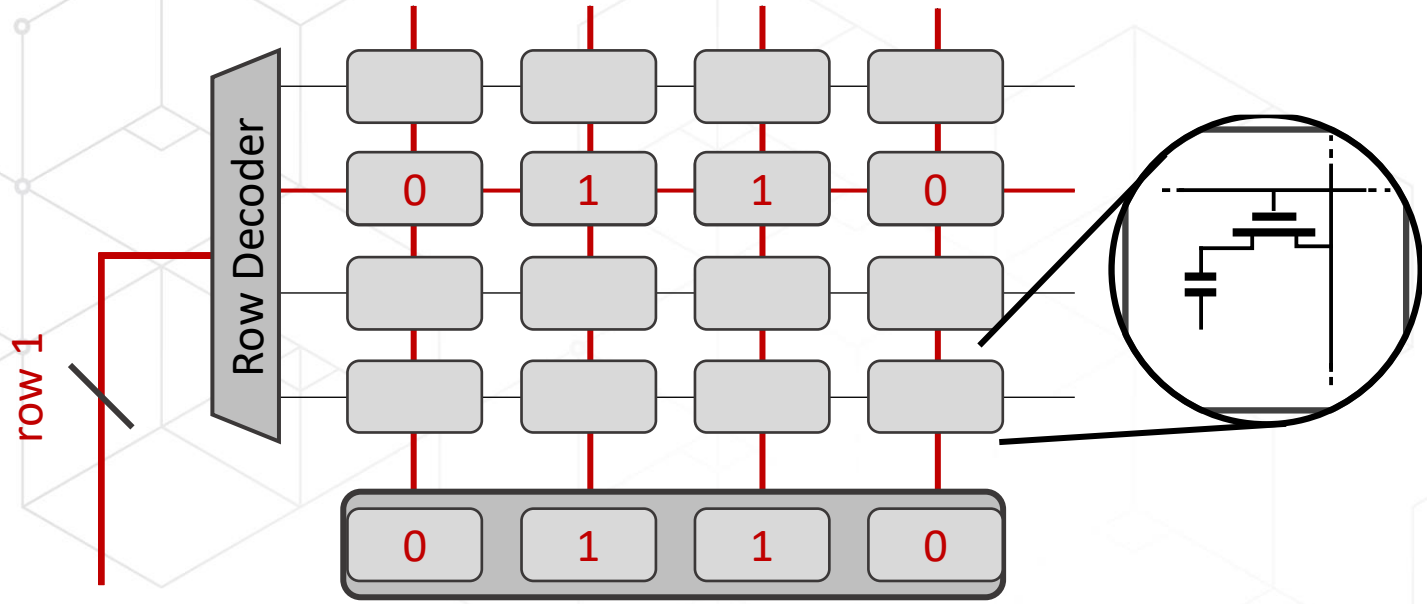


DRAM - Bank



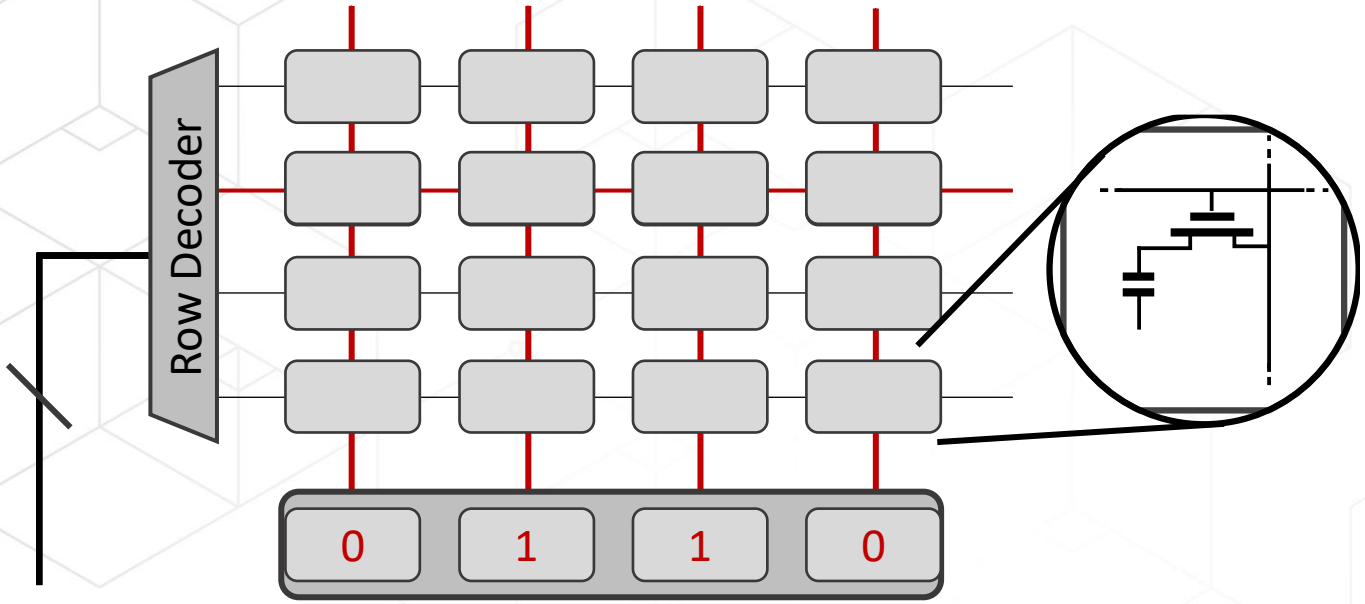
- Activate

DRAM - Bank



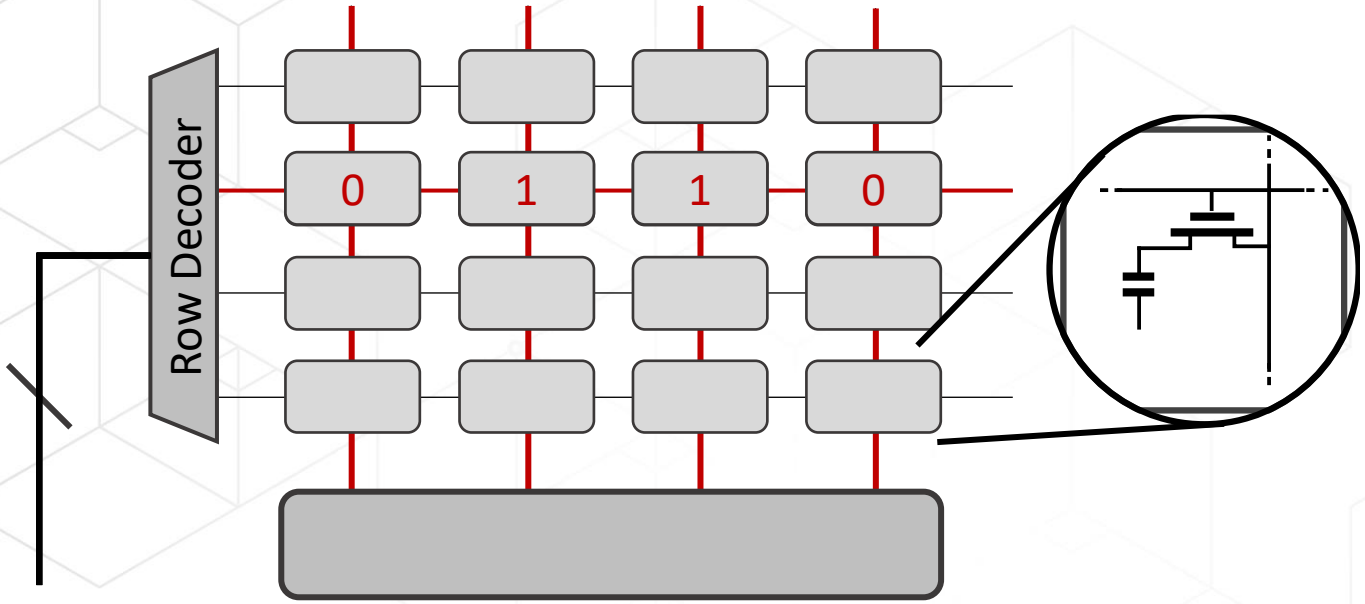
- Activate
- **Precharge**

DRAM - Bank



- Activate
- Precharge
- **Refresh**

DRAM - Bank



- Activate
- Precharge
- **Refresh**

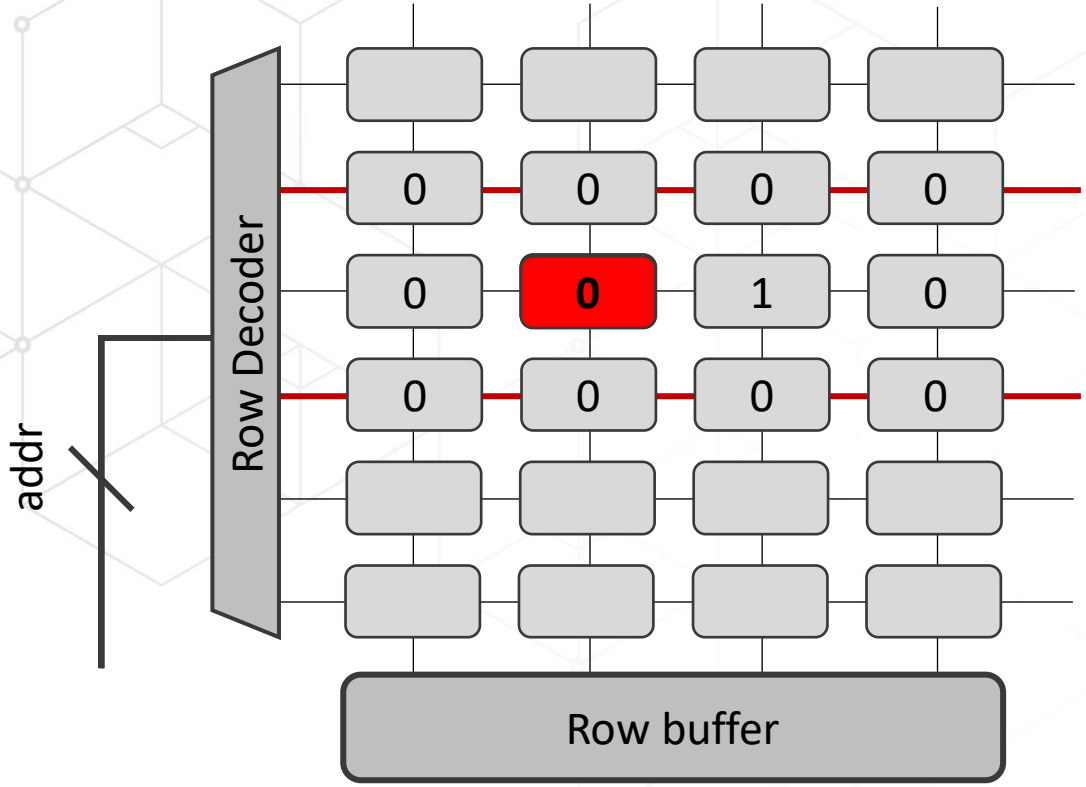
~~Refresh **every** row **every** 64ms~~

Refresh **some** rows **every** 7.8µs (64ms/8192)

➡ tREFi

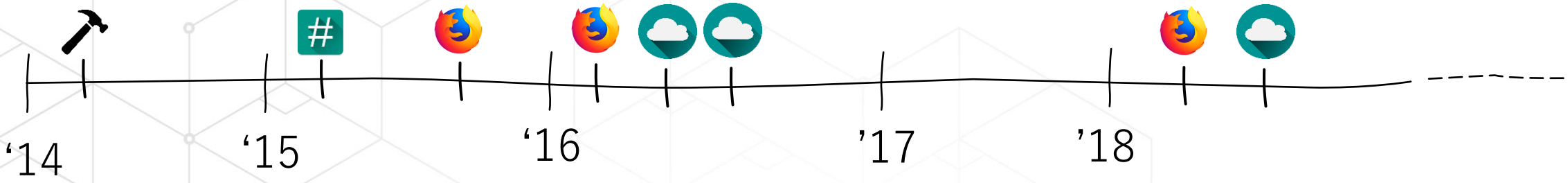
Rowhammer

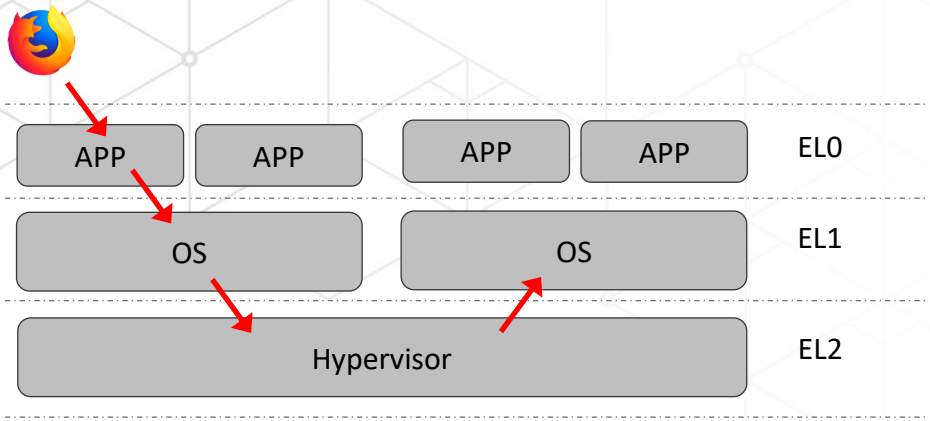
Rowhammer

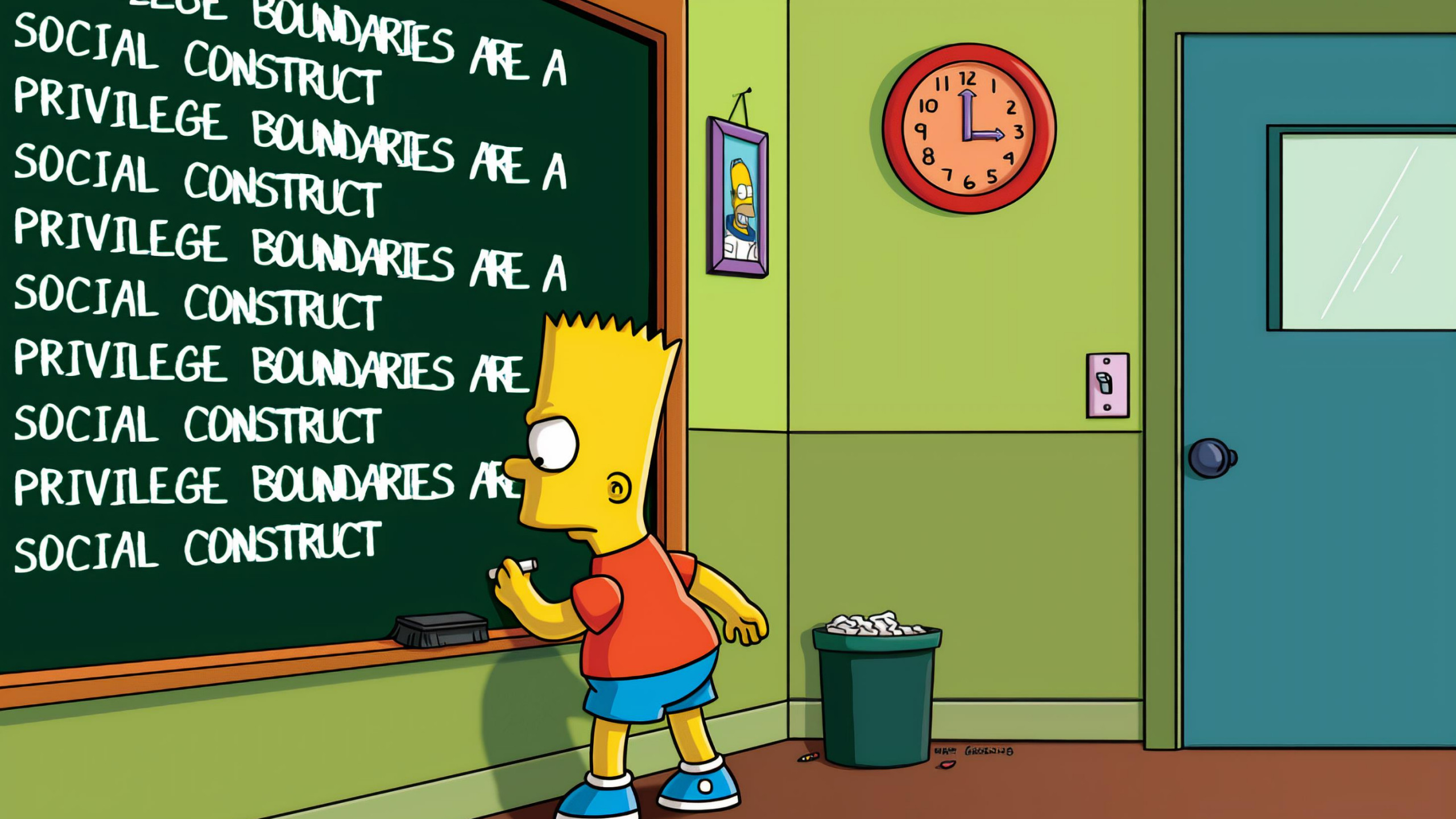


REPEATABLE!

Exploiting RowHammer






SOCIAL CONSTRUCT
PRIVILEGE BOUNDARIES ARE A
SOCIAL CONSTRUCT
PRIVILEGE BOUNDARIES ARE A
SOCIAL CONSTRUCT
PRIVILEGE BOUNDARIES ARE A
SOCIAL CONSTRUCT
PRIVILEGE BOUNDARIES ARE
SOCIAL CONSTRUCT
PRIVILEGE BOUNDARIES ARE
SOCIAL CONSTRUCT

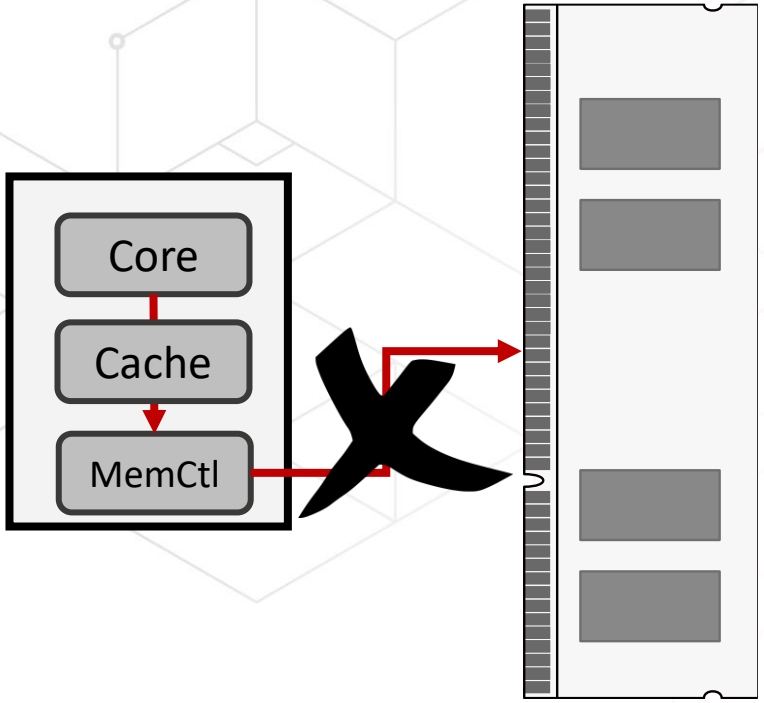
Memory integrity is **dead**

How do we get it back?

Software Defenses

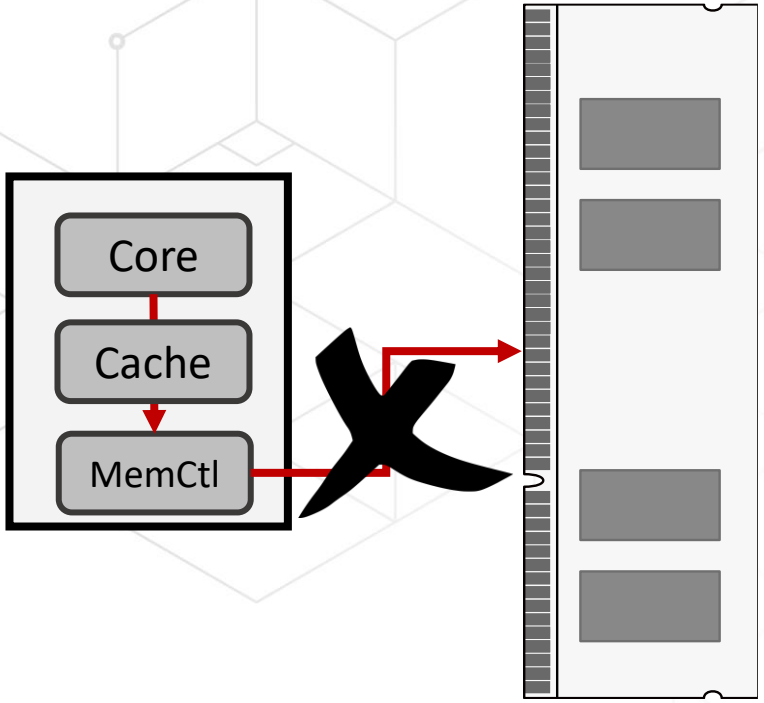
- Disabling flushing instructions
- Tracing via PMU
- Physical memory separation

clflush() …no more




```
for (r in N):
        *(volatile char*) row1
        *(volatile char*) row3
        clflush(row1)
        clflush(row3)
```

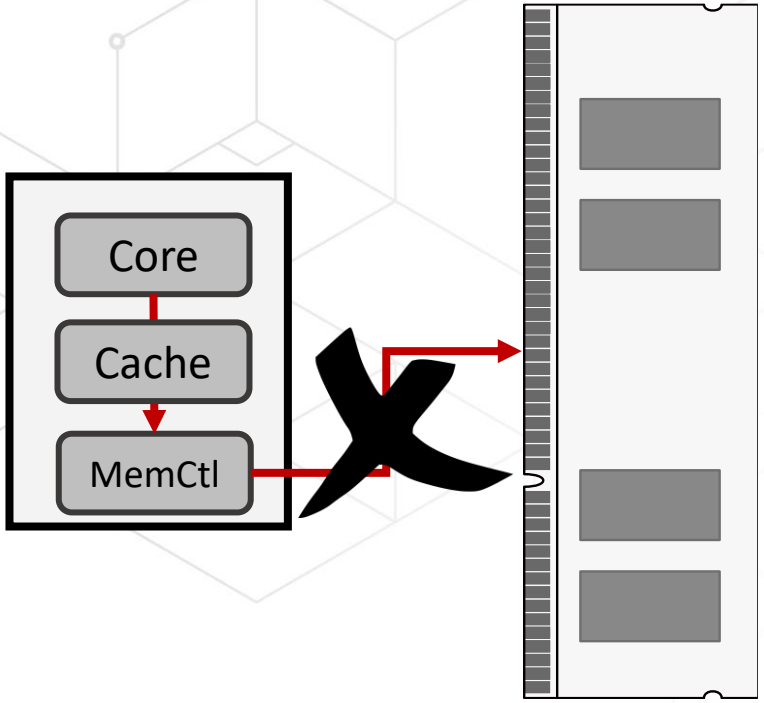
clflush() …no more



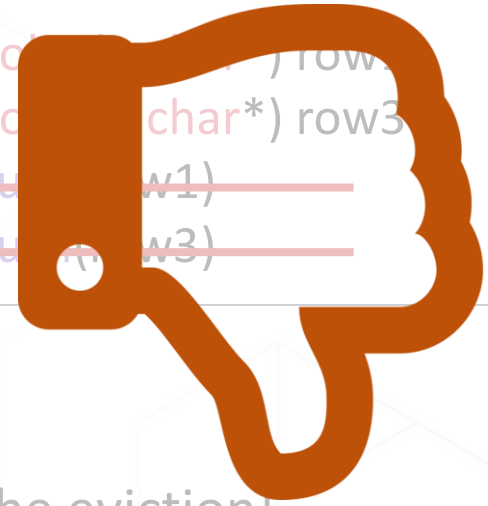

```
for (r in N):
        *(volatile char*) row1
        *(volatile char*) row3
        clflush(row1)
        clflush(row3)
```

Cache eviction!

clflush() …no more




```
for (r in N):
        *(vo                ) row
        *(vo          char*) row3
        clflu          w1)
        clflu          w3)
```

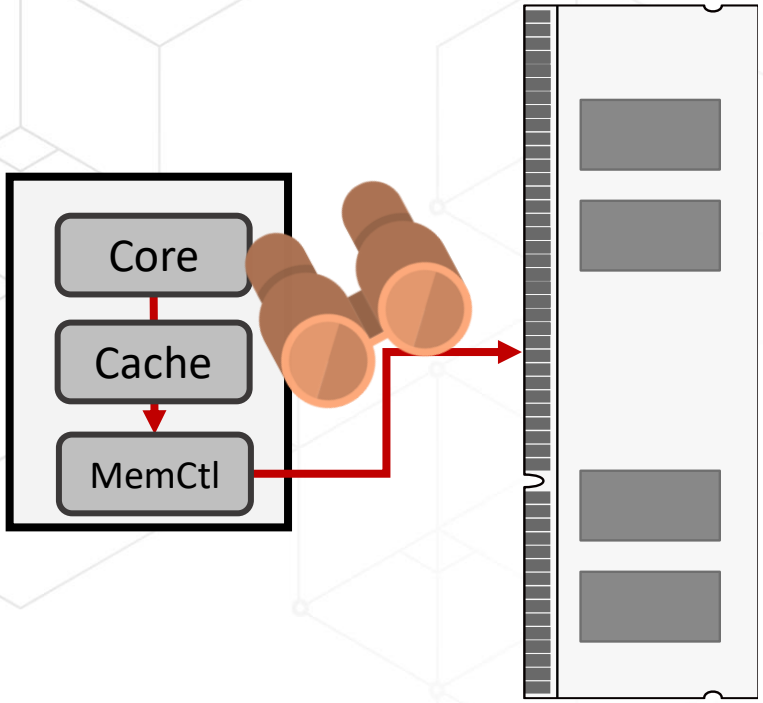


Cache eviction!

Software Defenses

- Disabling flushing instructions
- Tracing via PMU
- Physical memory separation

Tracing via PMU



Tracing via PMU

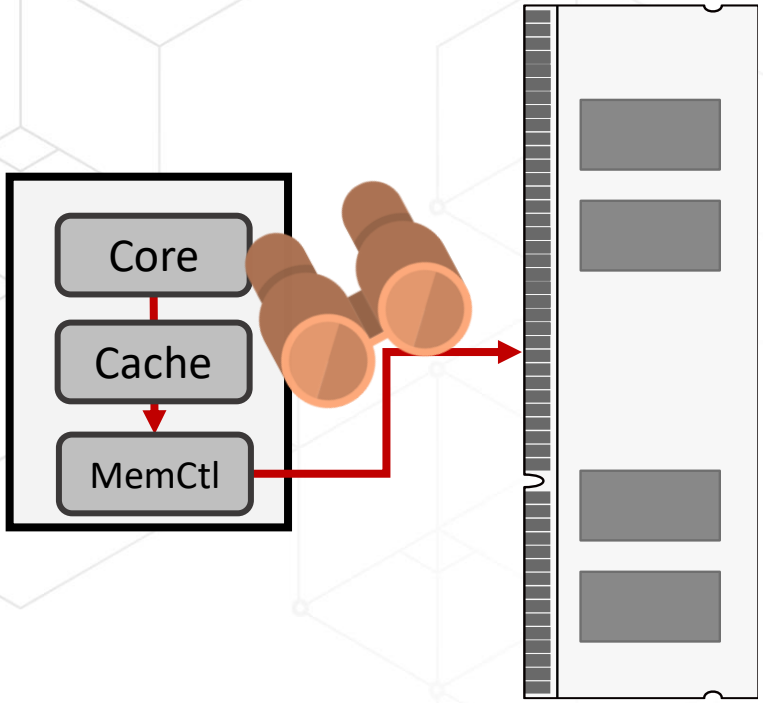
Table 19-1. Architectural Performance Events

Event Num.	Event Mask Name	Umask Value	Description
3CH	UnHalted Core Cycles	00H	Counts core clock cycles whenever the logical processor is in C0 state (not halted). The frequency of this event varies with state transitions in the core.
3CH	UnHalted Reference Cycles[1]	01H	Counts at a fixed frequency whenever the logical processor is in C0 state (not halted).
C0H	Instructions Retired	00H	Counts when the last uop of an instruction retire
2EH	LLC Reference	4FH	Counts requests originating from th the last level on-die cache
2EH	LLC Misses	41H	Counts each cach

Table 19-6. Performance Events of the Processor Core Supported by Skylake, Kaby Lake and Coffee Lake Microarchitectures (Contd.)

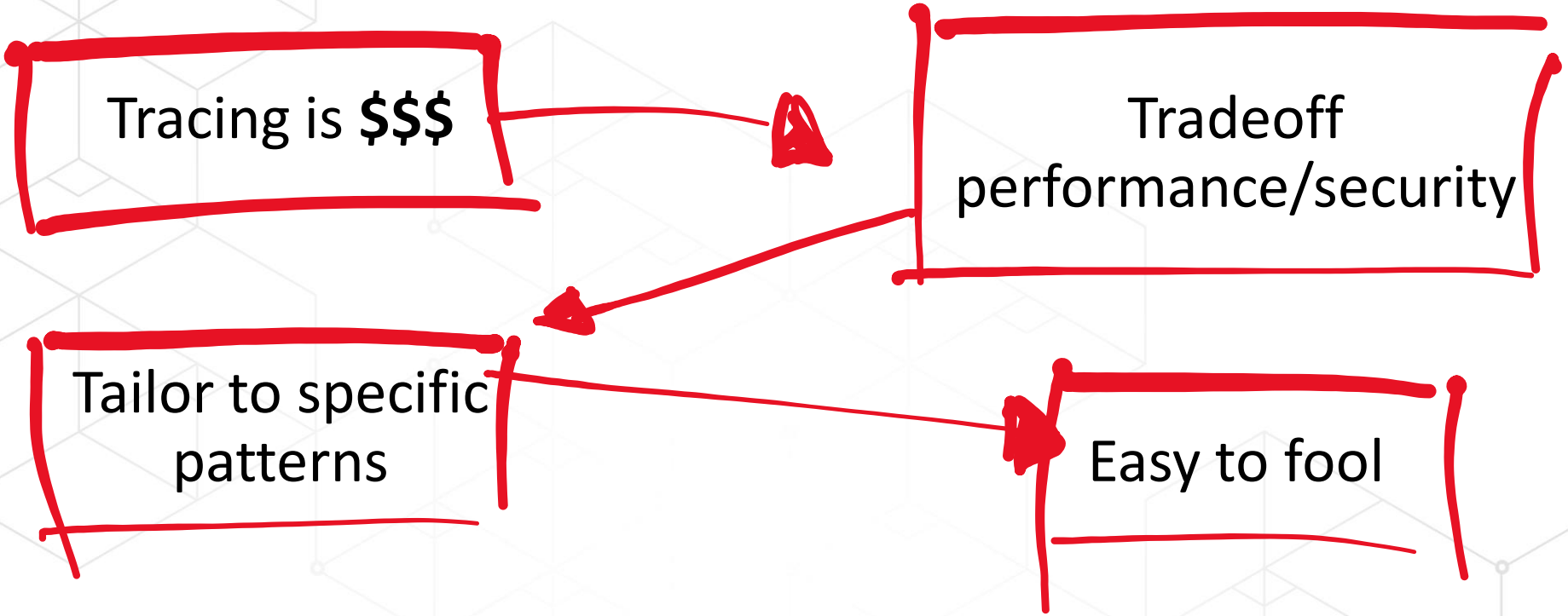
Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
D1H	20H	MEM_LOAD_RETIRED.L3_MISS	Retired load instructions missed L3. Excludes unknown data source.	PSDLA

Tracing via PMU

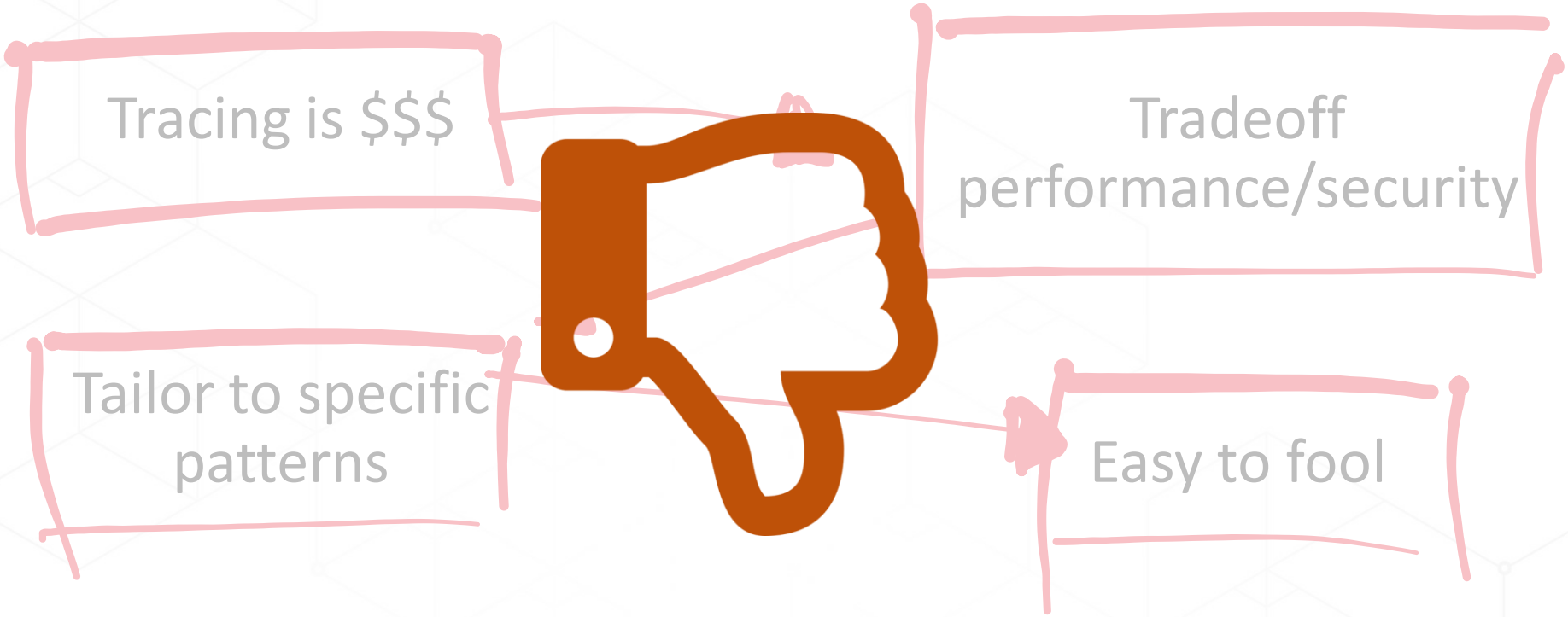


Refresh rows
being targeted

Tracing via PMU



Tracing via PMU

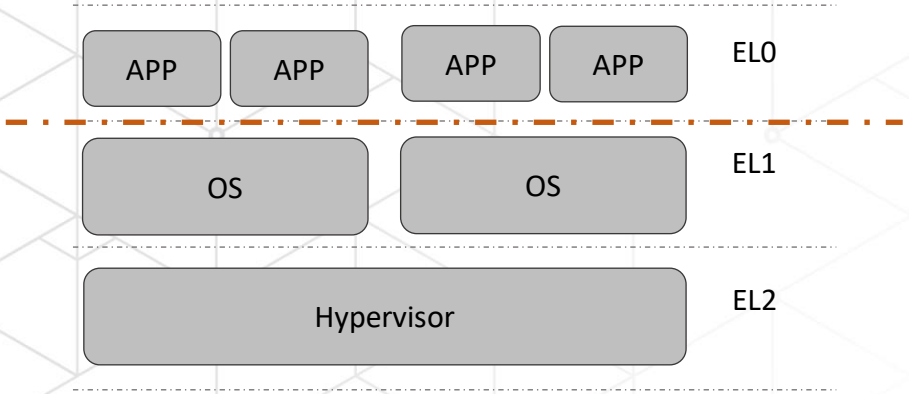



CAN WE... AT LEAST...
STOP EXPLOITATION?

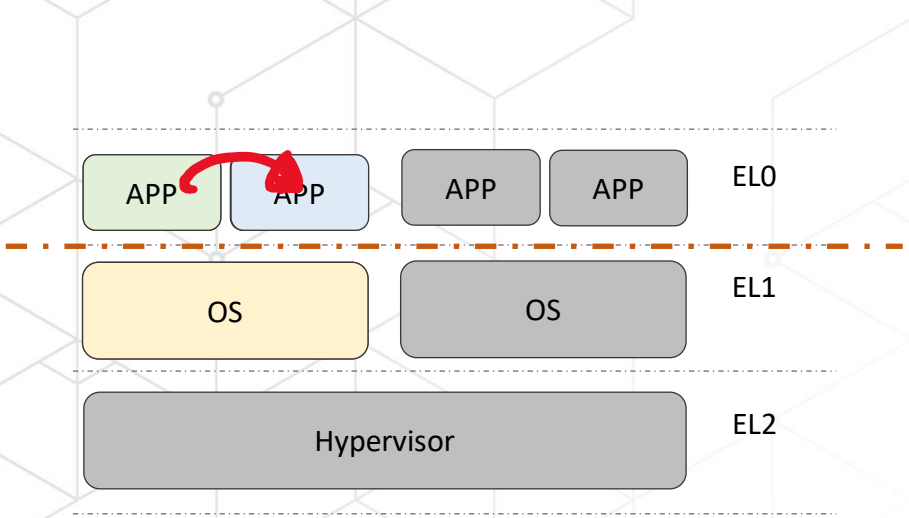
Software Defenses

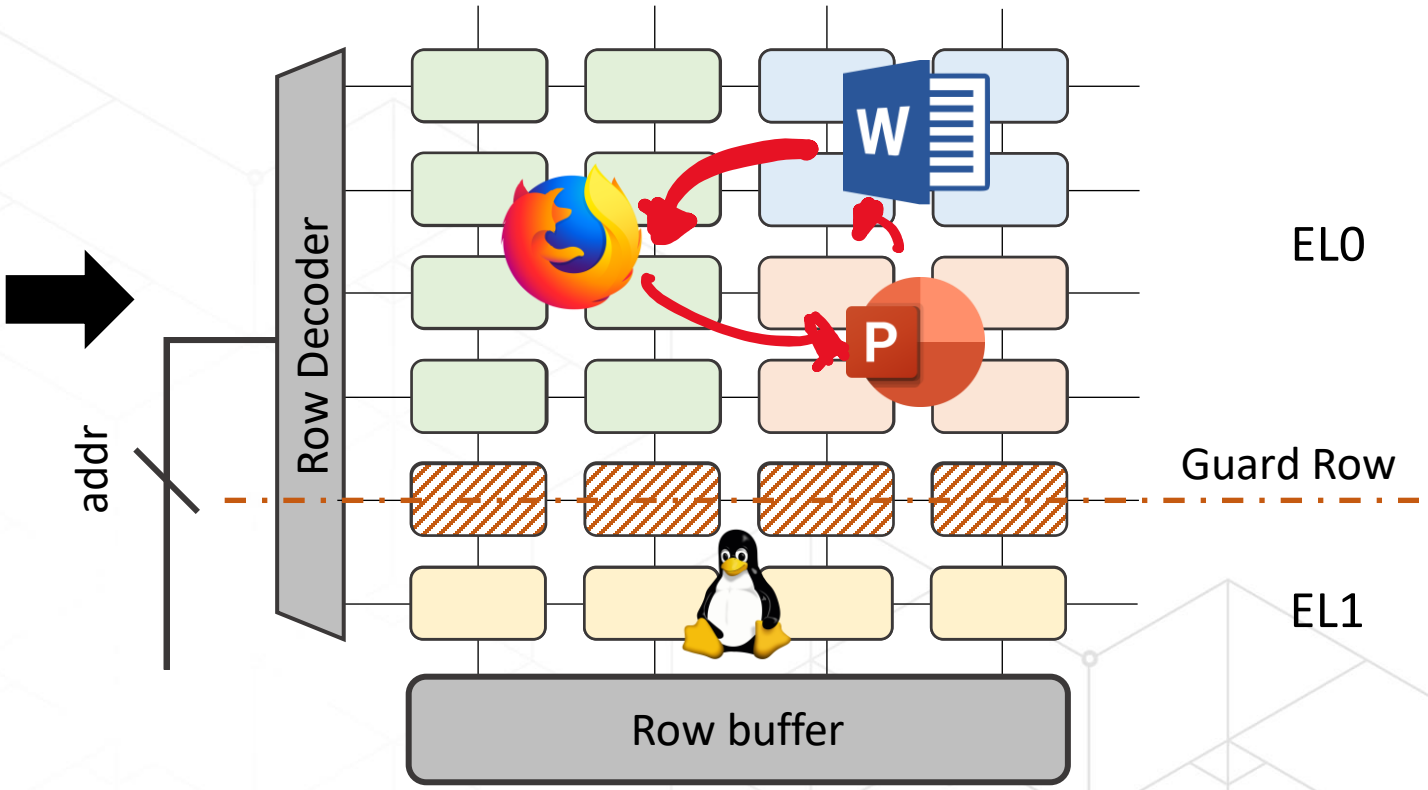
- Disabling flushing instructions
- Tracing via PMU
- Physical memory separation

Memory separation

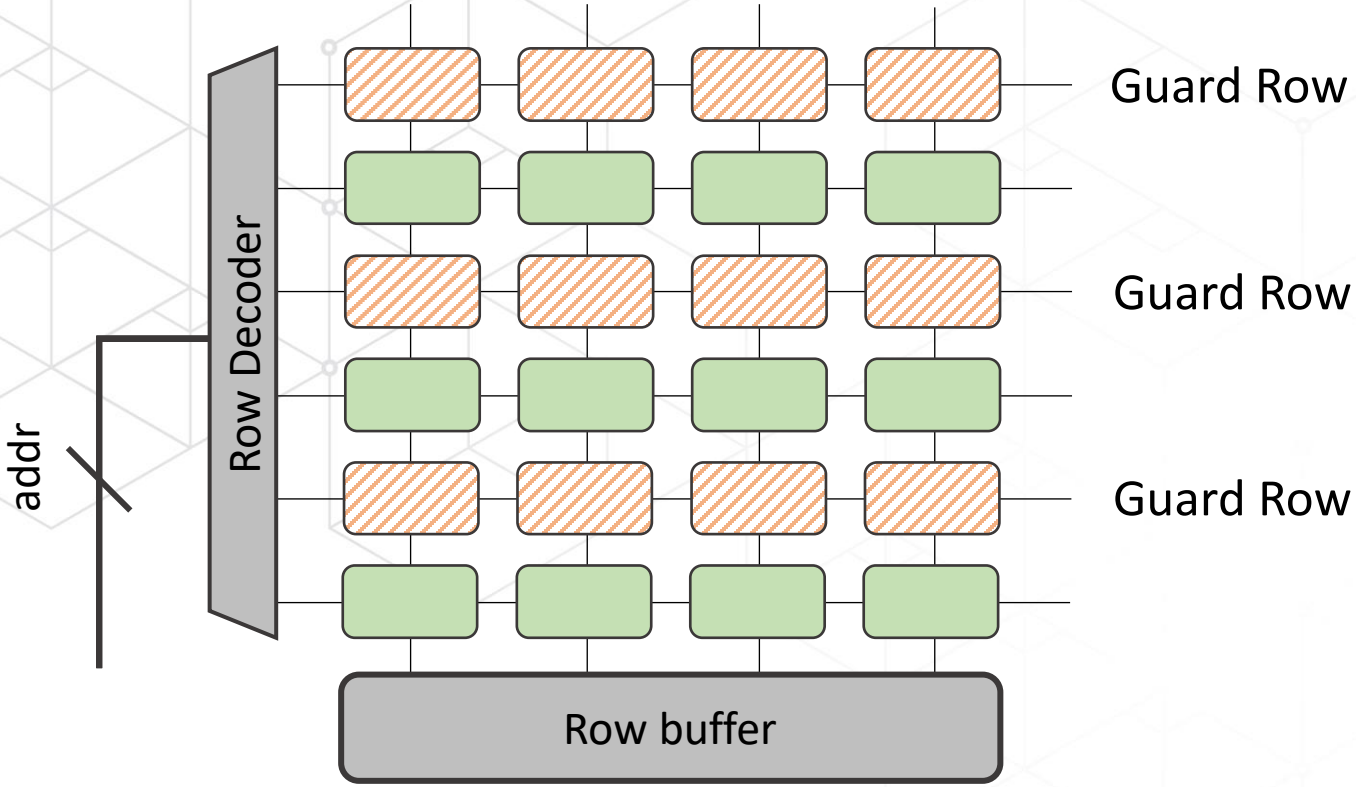


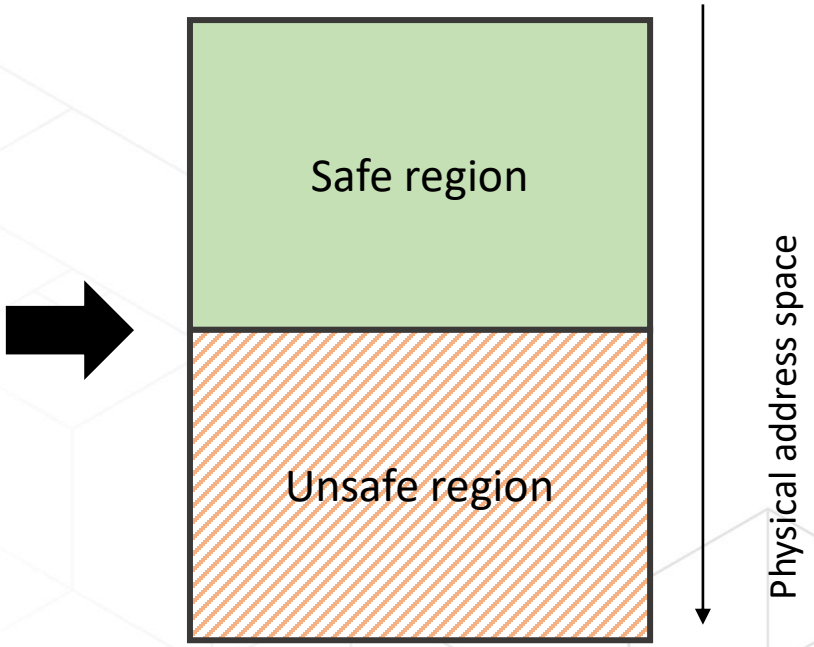
Memory separation





Memory separation

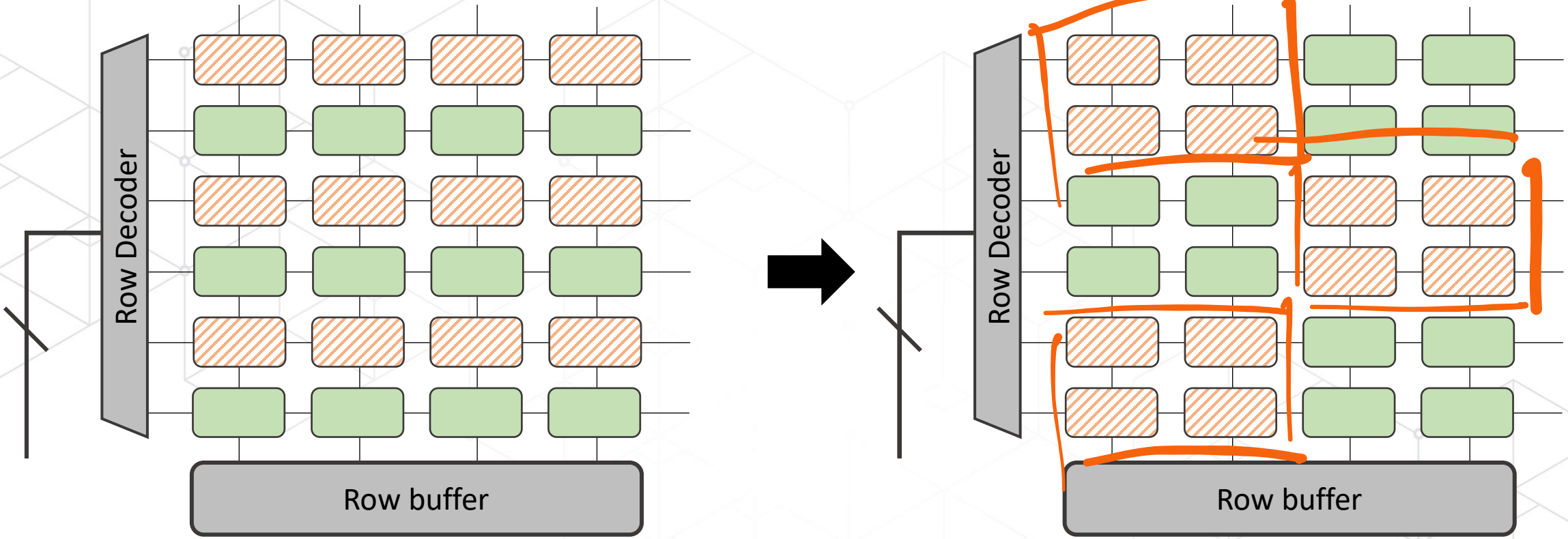




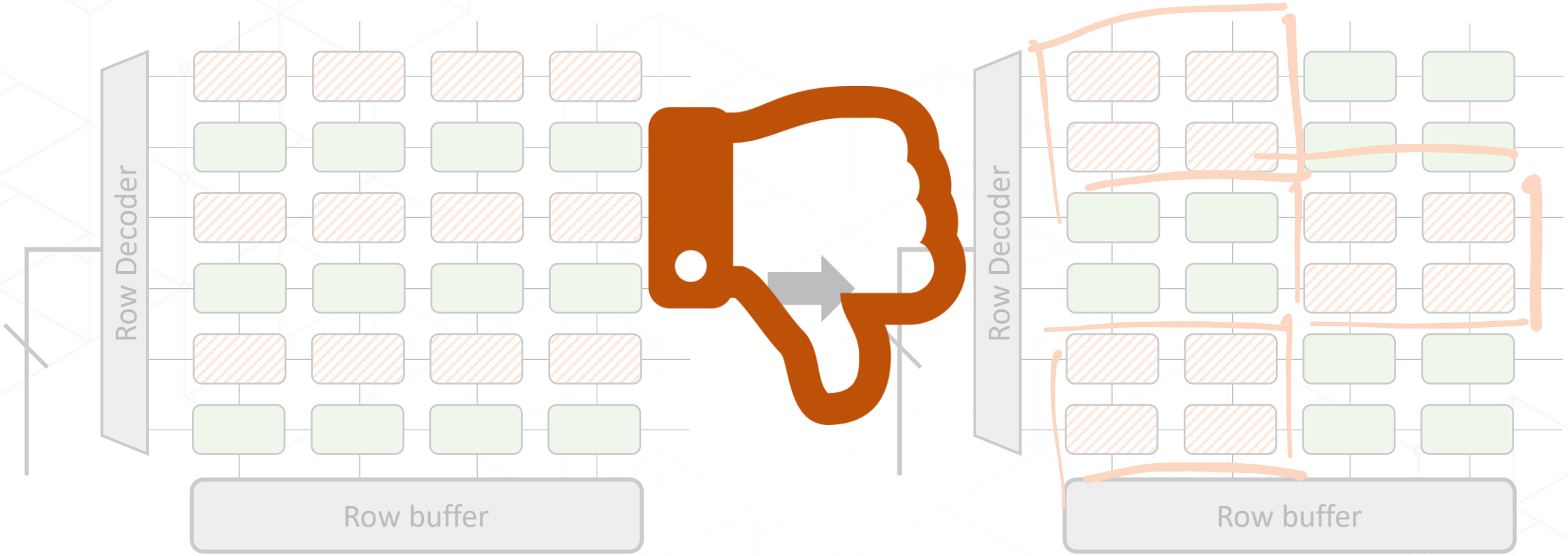
Limitations

- Bit flips can occur on rows further away
- Unknown memory geometry

Unknown geometry



Unknown geometry



Software Defenses

- Disabling flushing instructions
- Tracing via PMU
- Physical memory separation

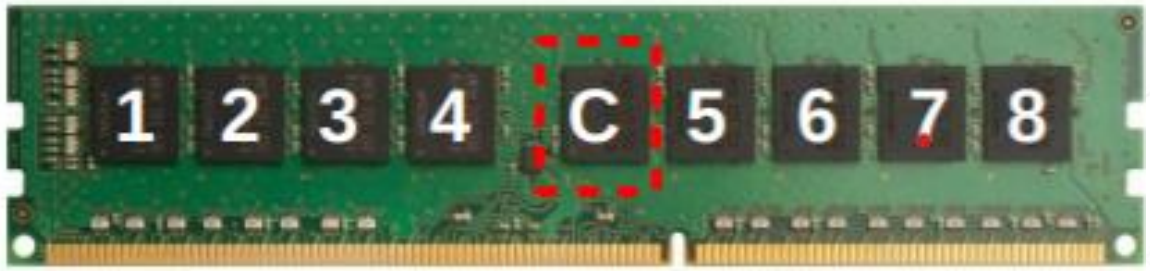
Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

ECC



- ECC DIMMs used in server systems
- SECDED
  - single error correction and double-bit error detection
  - 3 bit flips: potentially undetectable and uncorrectable (ECCploit)

L. Cojocar et al, “Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks,” in IEEE S&P, 2019

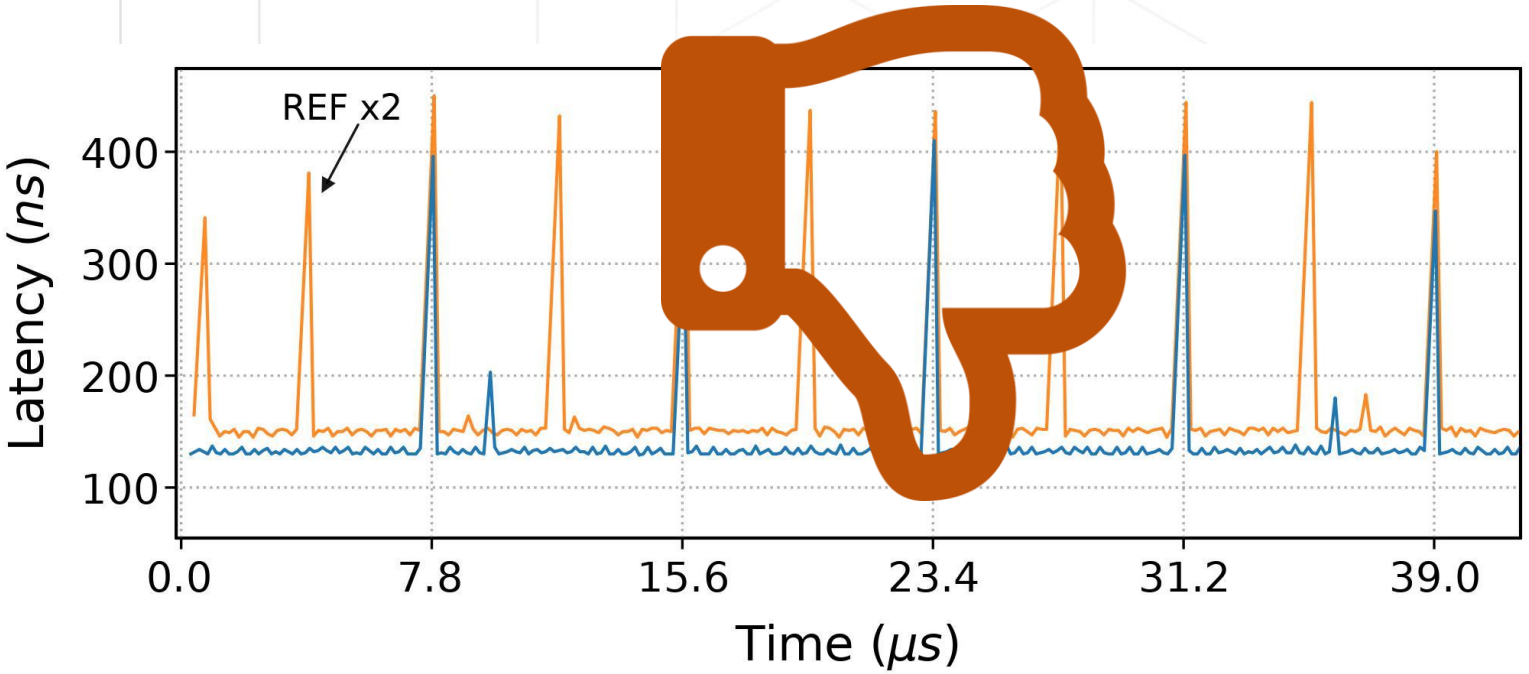
Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

Double refresh rate



Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

PARA

- Probabilistic Adjacent Row Activation
- The MC activates the adjacent rows with a probability p (<< 1) after a row is closed
- Stateless
- No significant overhead
- Memory geometry is unknown

Y. Kim et al, “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” in ISCA, 2014

Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

Pseudo Target Row Refresh

- From “Thoughts on Intel® Xeon® E5-2600 v2 Product Family Performance Optimisation”

For best performance, use pTRR compliant DDR3 DIMMs and enable the pTRR feature.

- When non-pTRR compliant DIMMs are used, the E5-2600 v2 system defaults into double refresh mode, which has longer memory latency/DIMM access latency and can lower memory bandwidth by up to 2-4%.

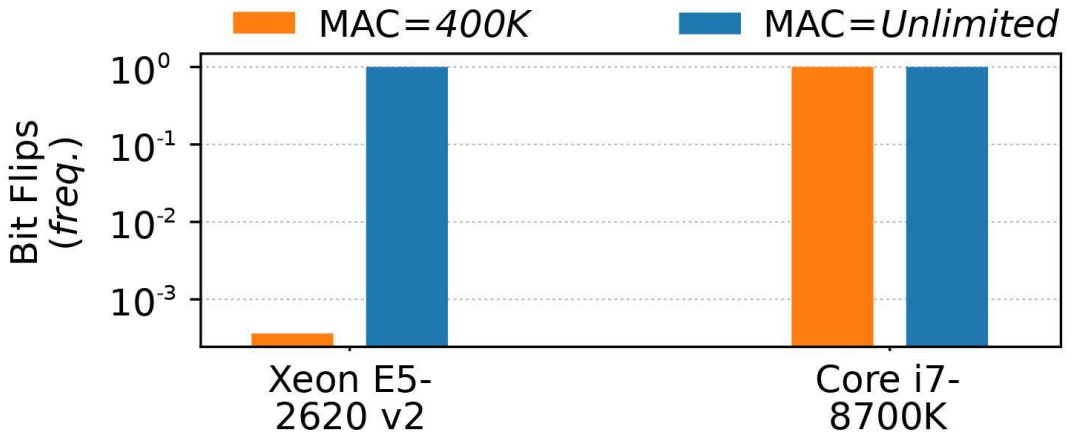
Key idea:

- The MC **monitors** rows activations and performs **targeted** refreshes on the victim rows

Pseudo Target Row Refresh

Compliant DIMMs?

- **The Serial Presence Detect (SPD)** contains **the Maximum Activation Count (MAC)**
  - **Untested**
  - **Unlimited**
  - **A discrete value (e.g. 300K)**



Defenses vol. 2

- Error-correcting codes (ECC)
- Refresh based mitigations
  - Double refresh rate
  - PARA
  - pTRR
  - TRR

In-DRAM mitigations

The DDR4 landscape

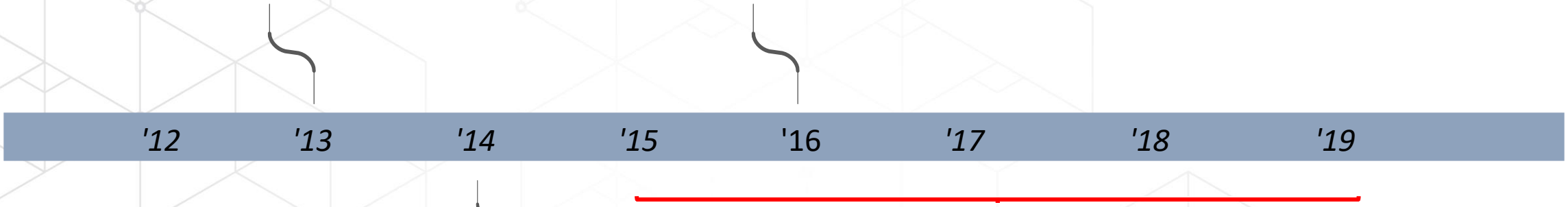
Timeline

pTRR DDR3

Intel reports pTRR on DDR3 server systems

In-DRAM TRR

Earliest manufacturing date of RH-free DRAM modules



DIMMs we focused on

pTRR DDR4

First DDR4 generation is pTRR protected

Target Row Refresh (TRR)

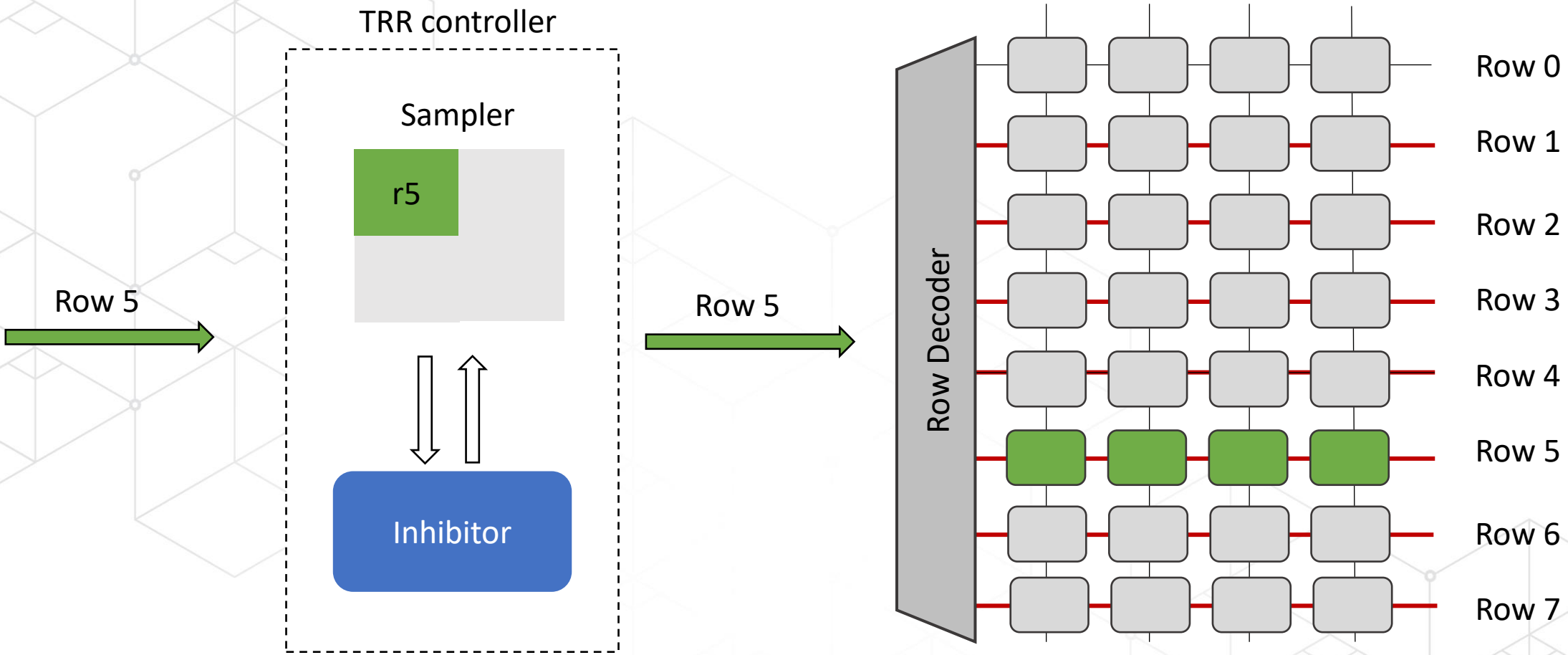
- TRR-like mitigations track rows activations and prevent errors
  - Errors prevention by targeted refresh commands
- No Memory Controller support
- Embedded in the DRAM circuitry

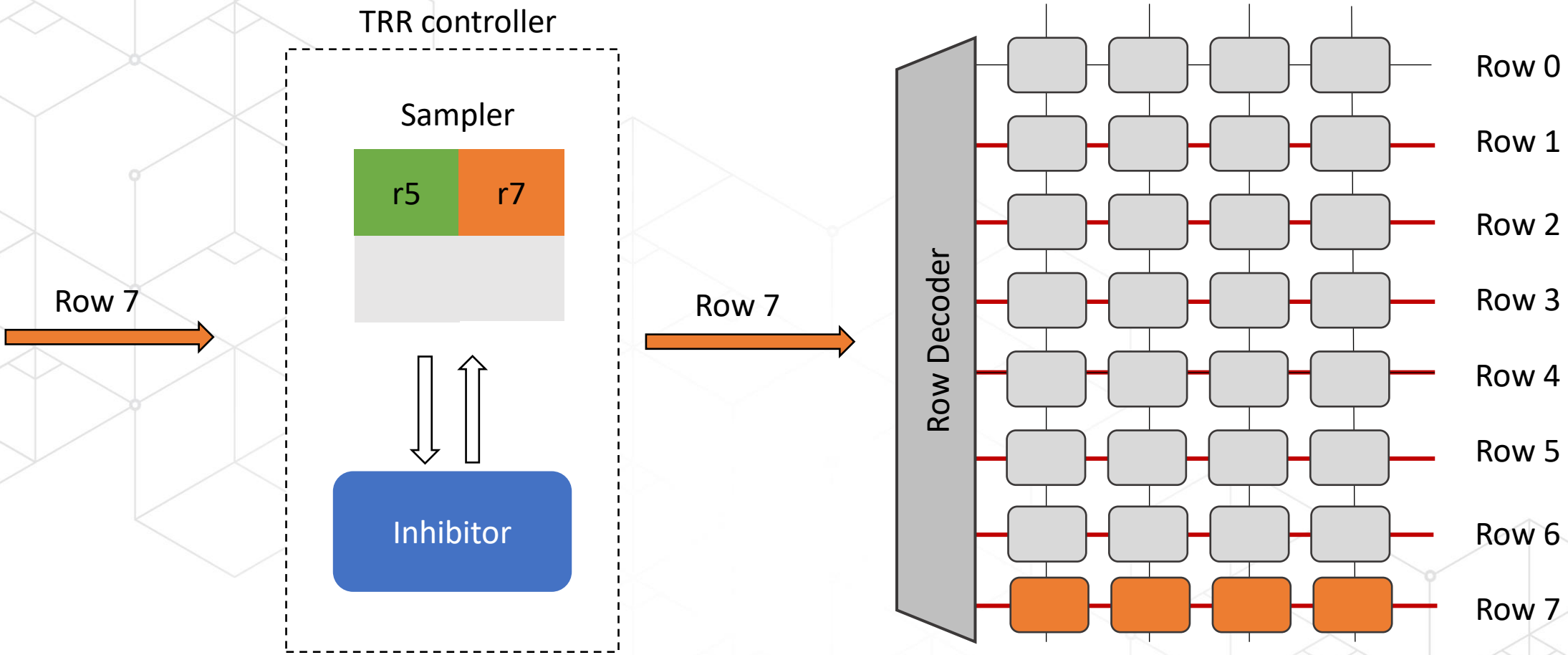
Target Row Refresh (TRR)

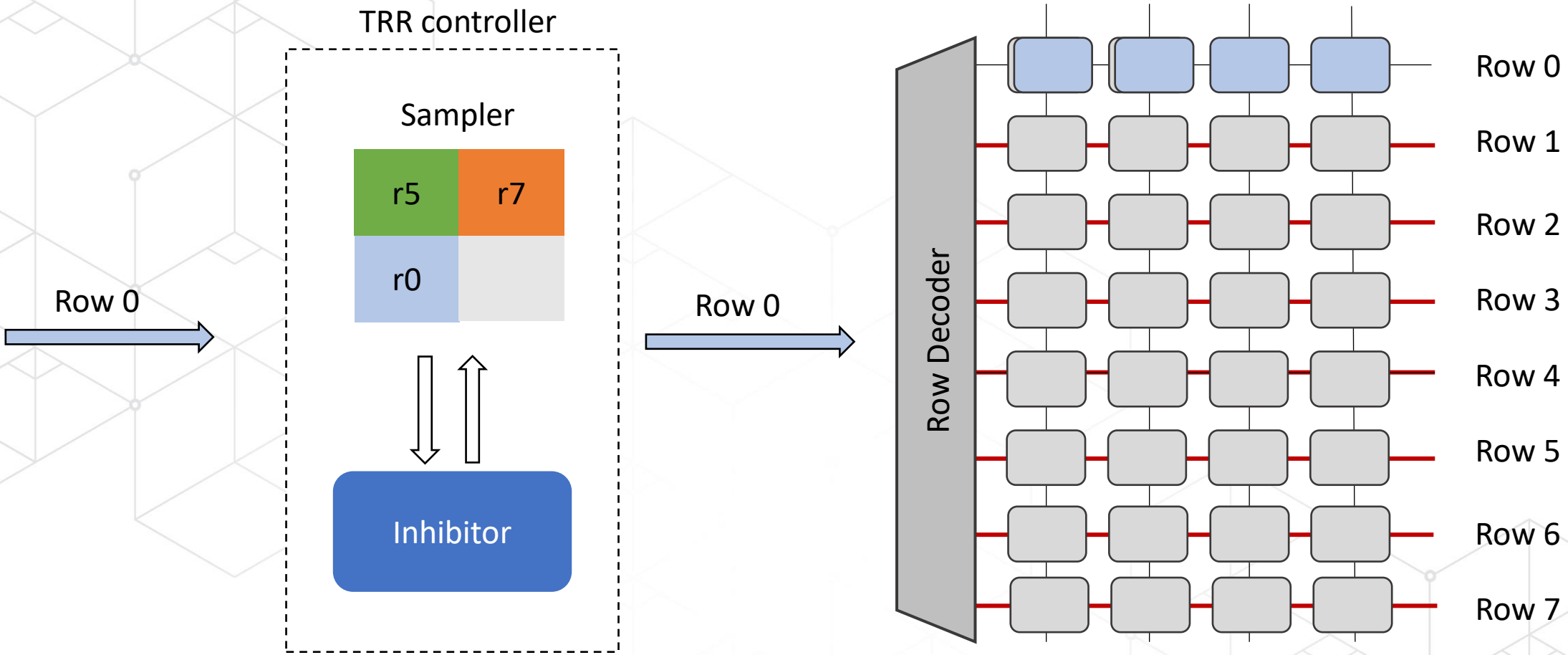
- Removed from JEDEC DDR4 standard
- Memory vendors advertise RowHammer-free memory modules
  - Many possible implementations!
  - Security by obscurity
- No real evaluation

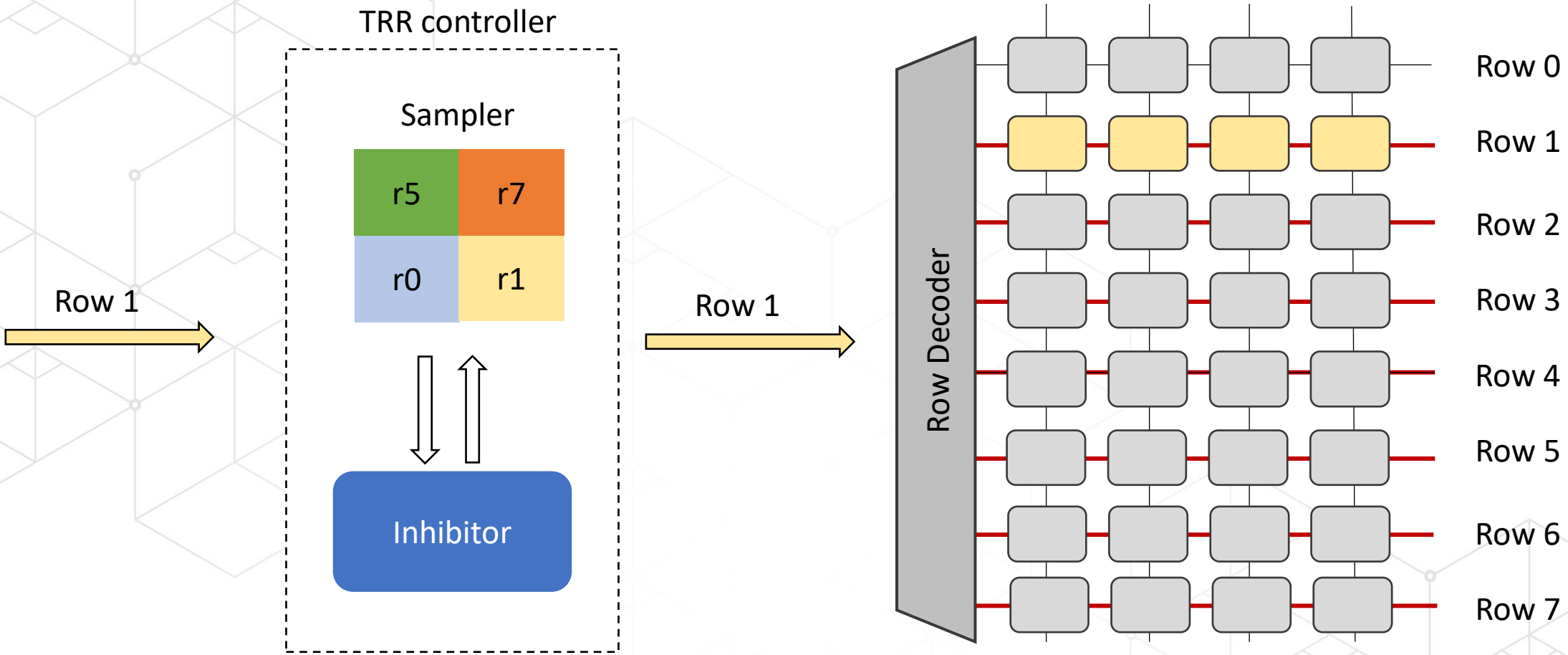
Abstractions

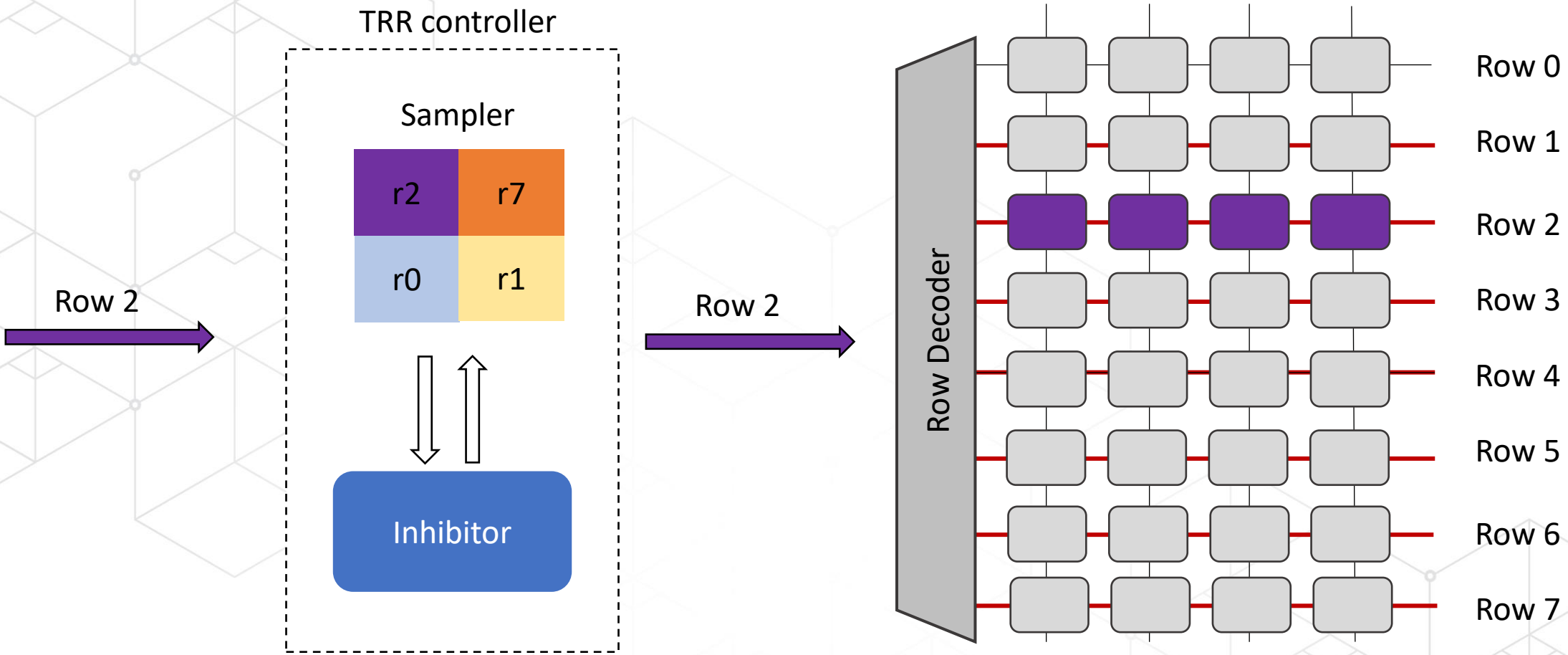
- **Sampler**
  - Row activations monitoring
  - It specifies which rows must be refreshed
- **Inhibitor**
  - Refresh
  - Remapping

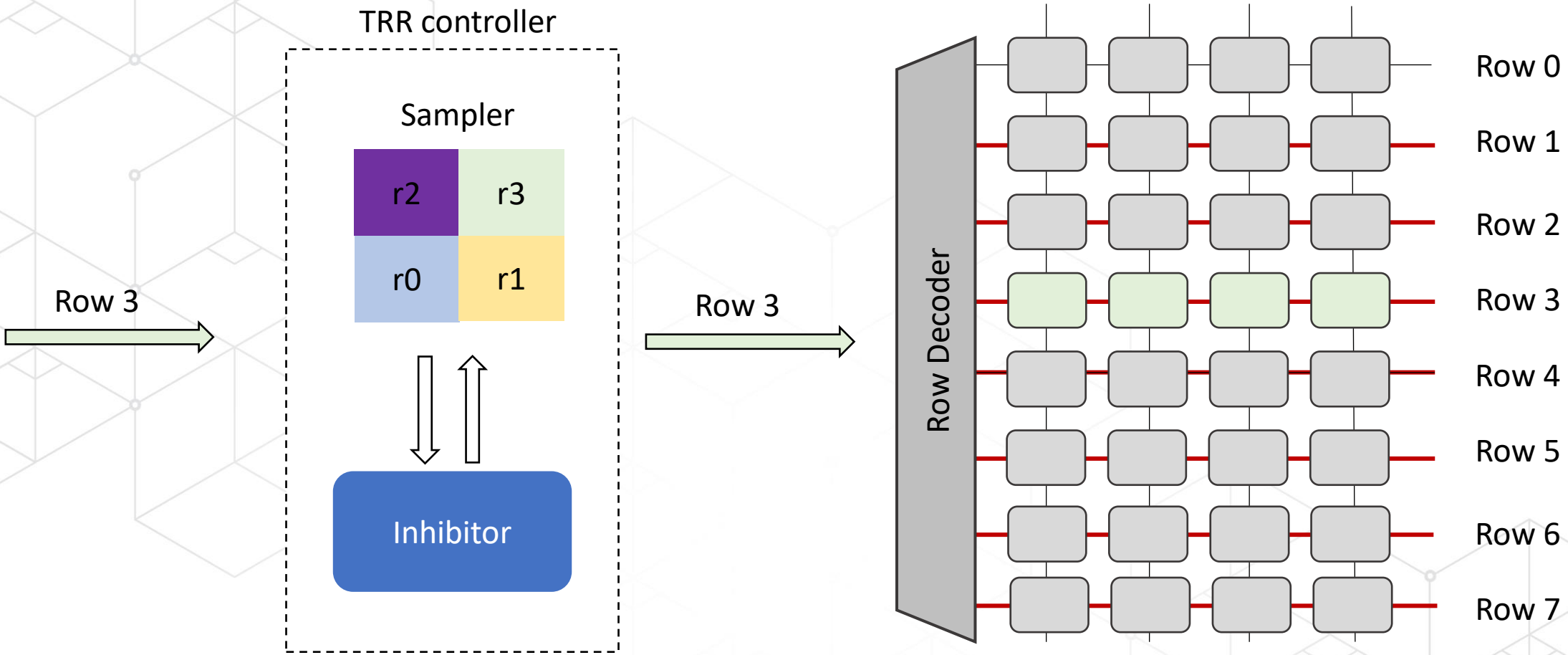

TRR controller
Sampler
r5
Row 5
Inhibitor
Row 5
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7

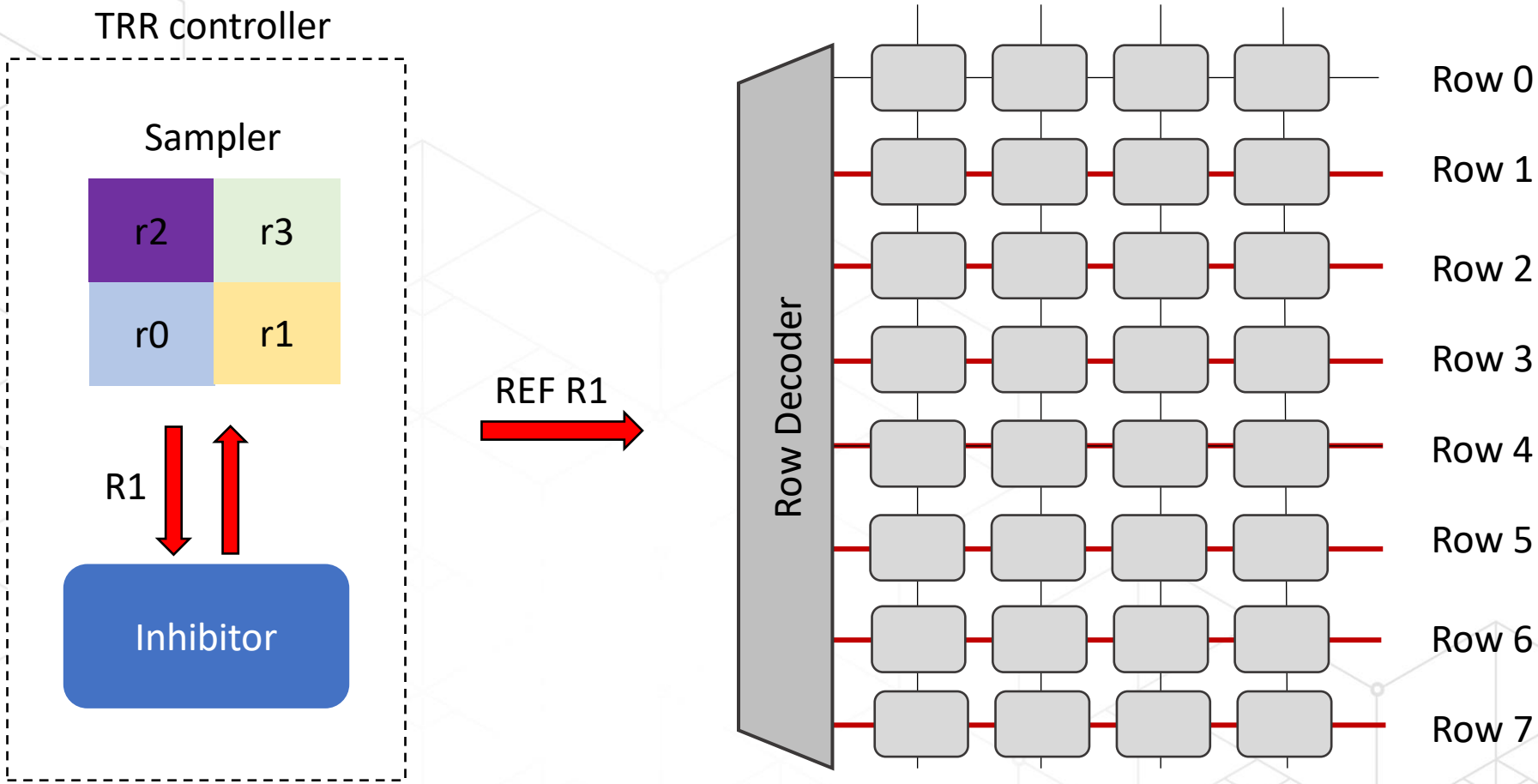

TRR controller
Sampler
r5
r7
Row 7
Inhibitor
Row 7
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7


TRR controller
Sampler
r5
r7
r0
Inhibitor
Row 0
Row 0
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7


TRR controller
Sampler
r5
r7
r0
r1
Inhibitor
Row 1
Row 1
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7


TRR controller
Sampler
r2
r7
r0
r1
Inhibitor
Row 2
Row 2
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7


TRR controller
Sampler
r2
r3
r0
r1
Row 3
Row 3
Inhibitor
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7


TRR controller
Sampler
r2
r3
r0
r1
R1
Inhibitor
REF R1
Row Decoder
Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7

Reverse Engineering

Challenges

from the CPU side not feasible

high interface

- We would need a

SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

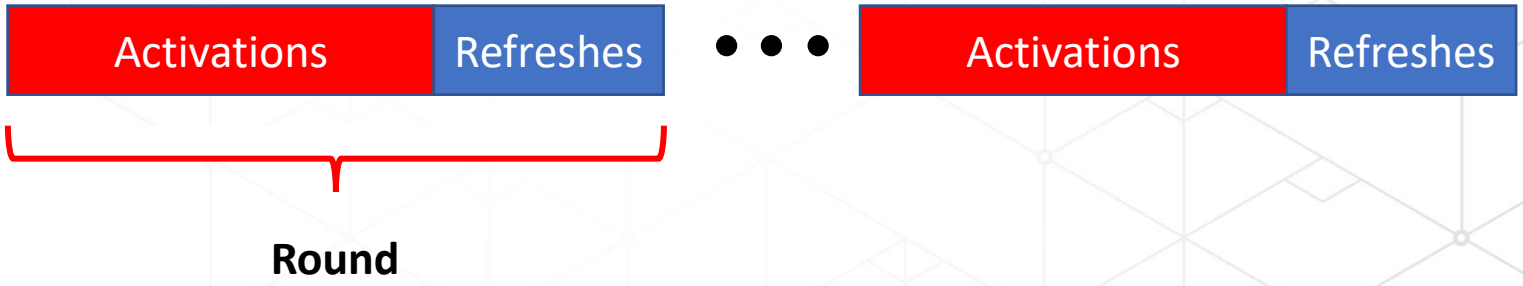
H. Hassan et al., "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in HPCA, 2017

Reverse Engineering

- **What?** The sampler size!
- **How?**
  - Issuing specific commands sequences by SoftMC to the DIMM
  - Targeting more then 2 aggressor rows!
  - **The Many-sided RowHammer**
- **Why?** To lead the sampler to discard a few rows

Methodology

- Pick **N** aggressor rows
- Perform a series of hammers (activations of aggressors)
  - **8K activations**
- After each series of hammers, issue **R refreshes**
- **10 Rounds**



Case study

- The TRR mitigation acts on every refresh command
- The mitigation can sample more than one aggressor per refresh interval
- The mitigation can refresh only a single victim within a refresh operation
- Sweeping the number of refresh operations and aggressor rows reveals the sampler size
- The sampling mechanism is affected by the addresses of aggressor rows

Findings

- The DDR4 substrate is much more vulnerable!
  - Bit flips with less then 50K activations per aggressor
- The sampler can be overfilled
  - Victims rows may not be properly refreshed by the Inhibitor
- Sampler properties
  - Timing-based
  - Frequency-based


ONE PROBLEM SOLVED…
ONE MILLION PROBLEMS LEFT

TRRespass

TRRespass: The RowFuzzer

- The first row fuzzer
  - Black Box fuzzing
  - Scalable approach for testing
- Randomizing hammering pattern
  - # Aggressors
  - Aggressor location


BIT FLIPS...
LIGHTYEAR
BIT FLIPS EVERYWHERE

TRRespass: The RowFuzzer

- TRR not secure at 100%
  - Discretely effective against state-of-the-art hammering patterns
  - Vulnerable to novel patterns

What if combined with other kind of defenses?

- Double refresh rate: still flippy!
- ECC: Not tested
  - Many-sided RH usually causes multiple flips

Recap

- Software mitigations
  - High overhead
  - Lack of memory geometry information
- Hardware mitigations
  - Hardly deployable
  - Fragmented solutions
  - Missing a standard

Conclusions

- DDR4 device even more vulnerable than previous versions
- All major vendors are affected:
  - 90% of the market
- Fuzzing techniques are helpful
- After almost 10 years RowHammer is still a problem
- No prompt mitigation available

Thank You!

HITB LOCKDOWN 002
livestream

Pietro Frigo p.frigo@vu.nl, Emanuele Vannacci e.vannacci@vu.nl