ECE 382N-Sec (FA25):

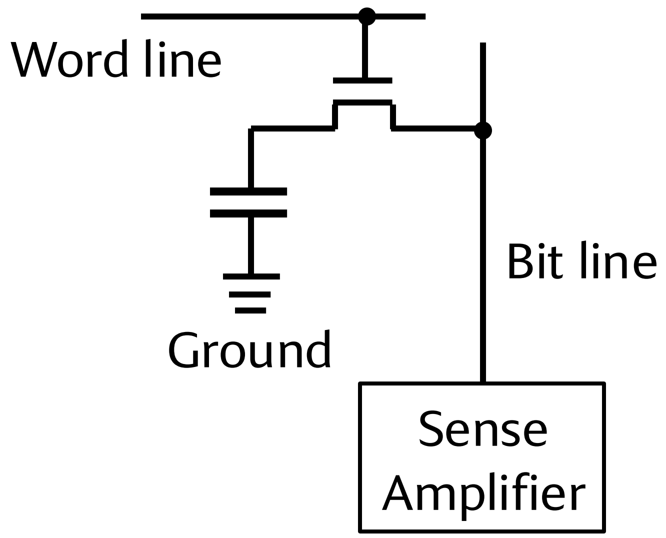
L11: RowHammer

Neil Zhao

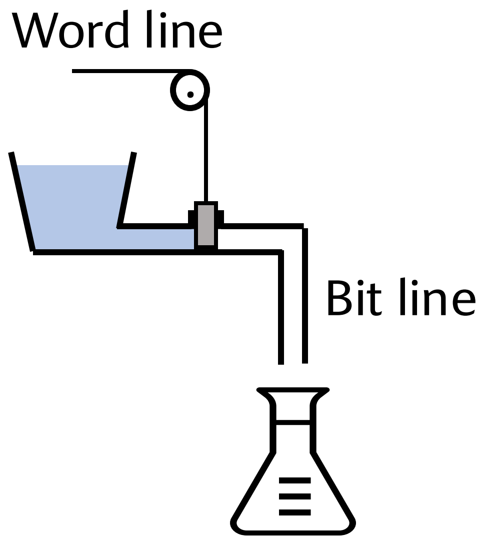
neil.zhao@utexas.edu

DRAM Refresher: A Textbook DRAM Cell

A single DRAM cell
(one-transistor, one-capacitor)



The water bucket analogy
(one-bucket, one-valve)

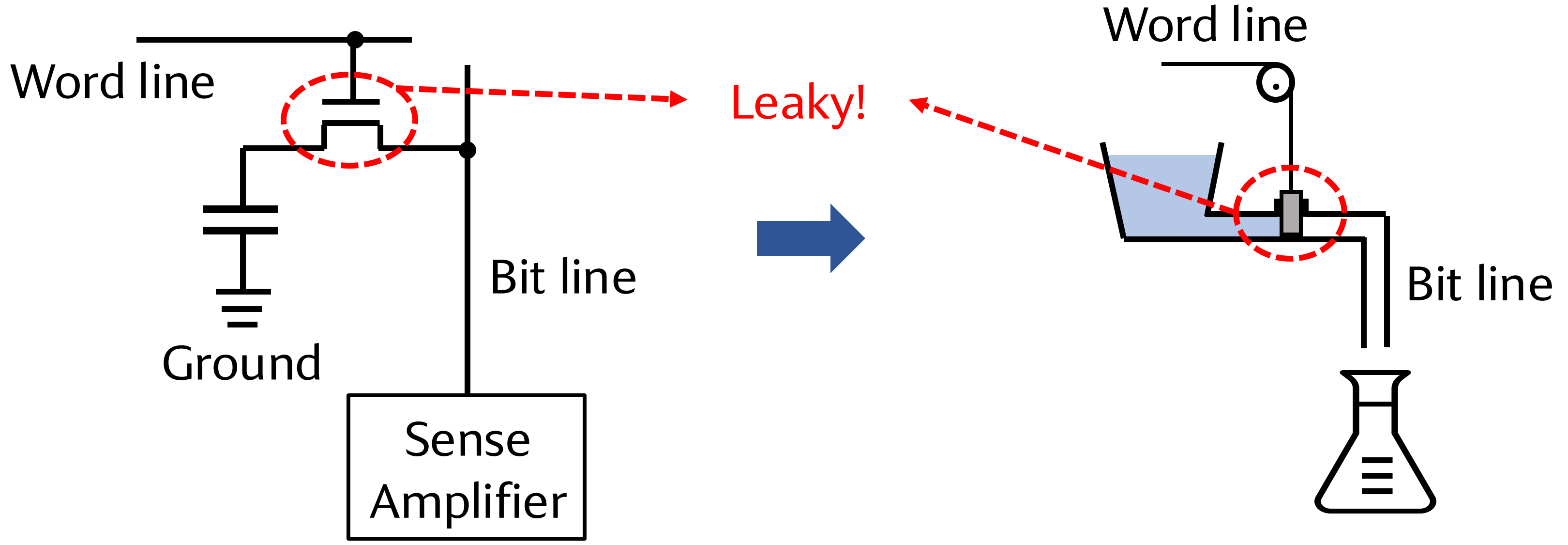


Storing 1 bit. Destructive reads

DRAM Refresher: A Textbook DRAM Cell

A single DRAM cell
(one-transistor, one-capacitor)

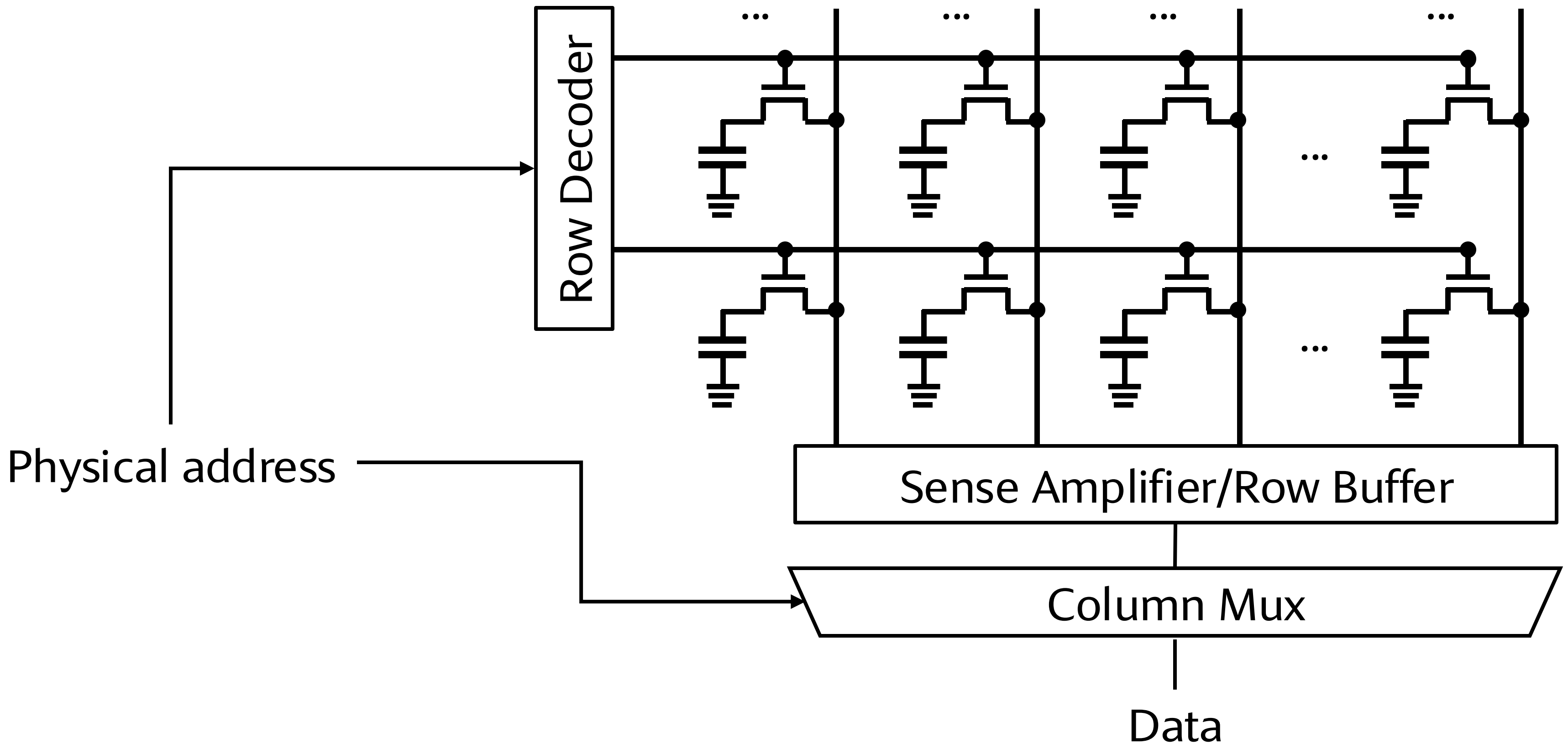
The water bucket analogy
(one-bucket, one-valve)



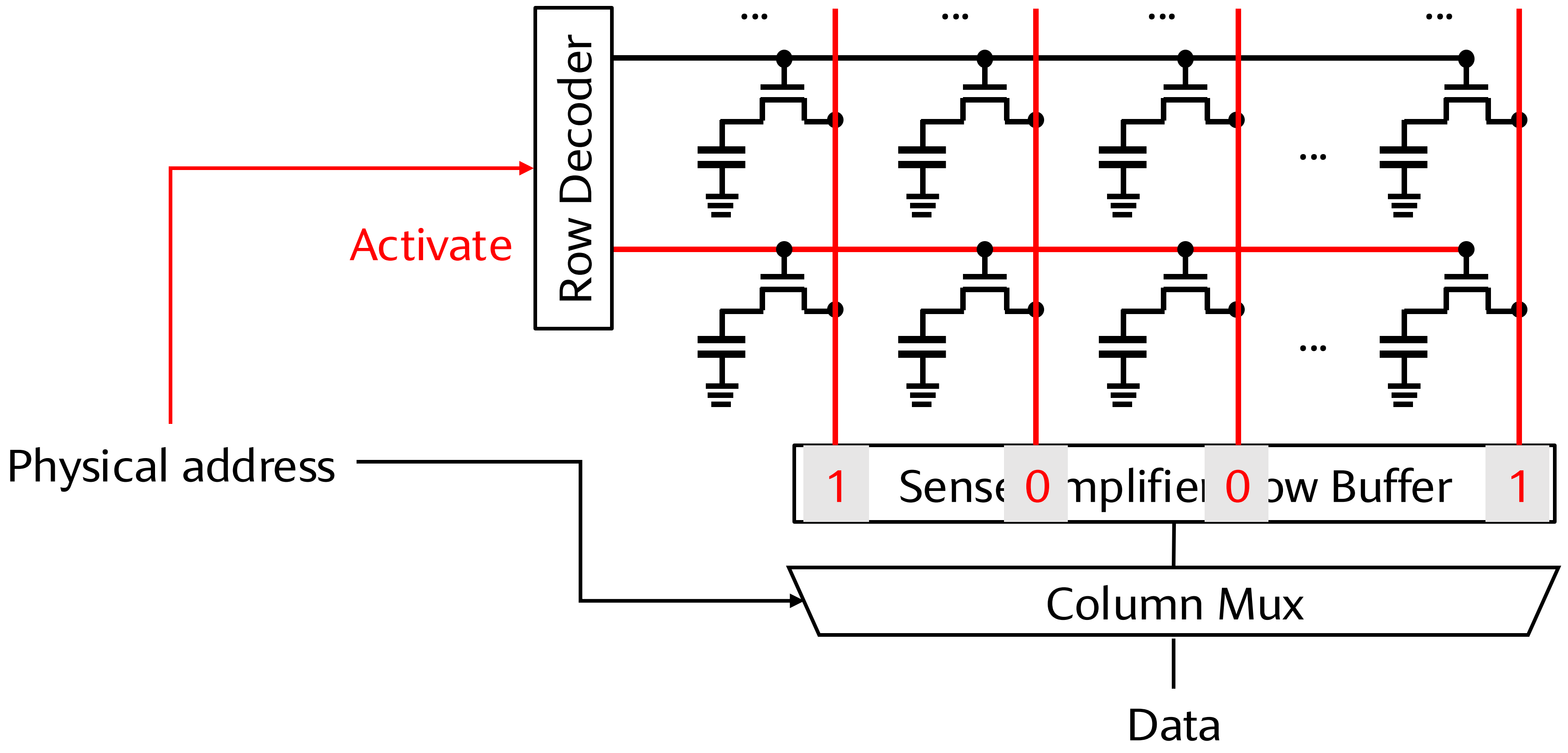
Solution[*]**:** Periodic refreshing (e.g., read it out and write it back every 64ms)

[*]Or spray it with liquid nitrogen!

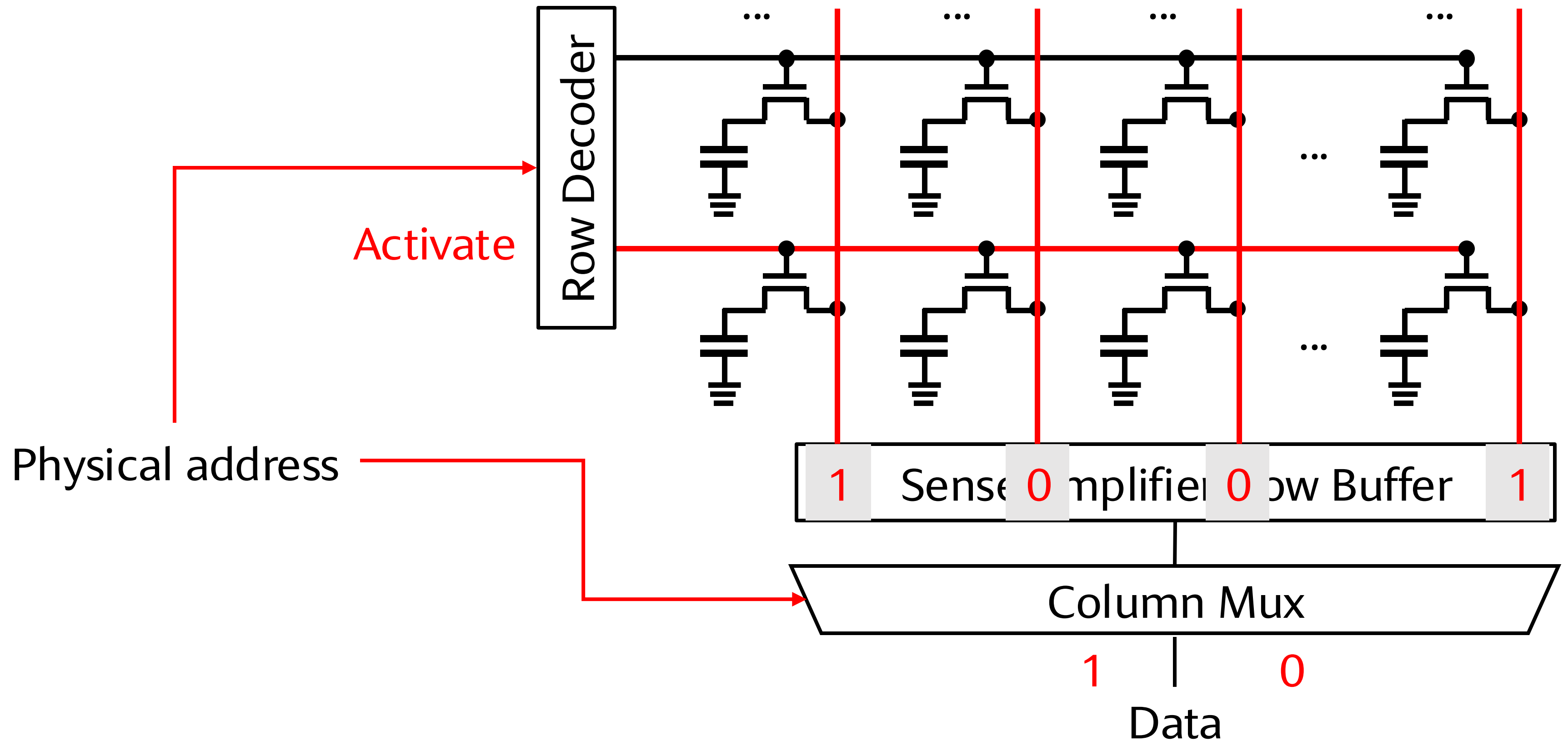
From a Single Cell to a DRAM Array



From a Single Cell to a DRAM Array

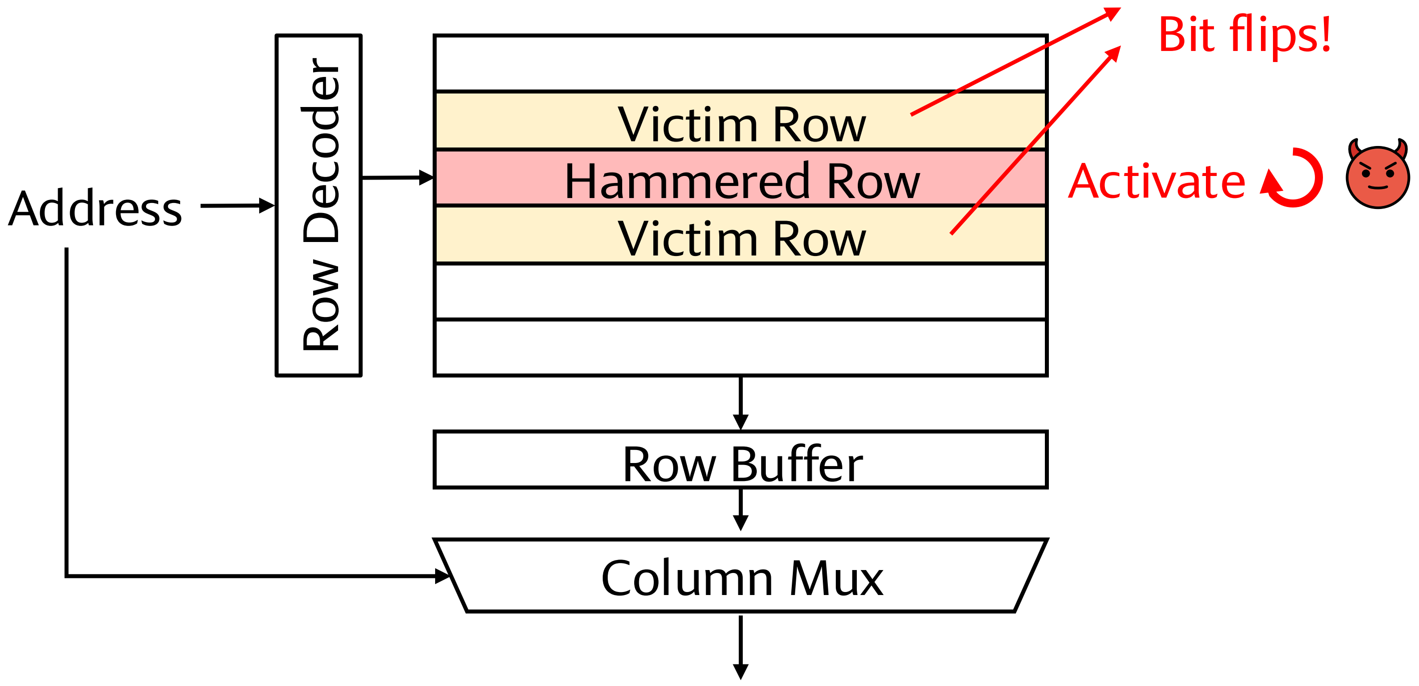


From a Single Cell to a DRAM Array

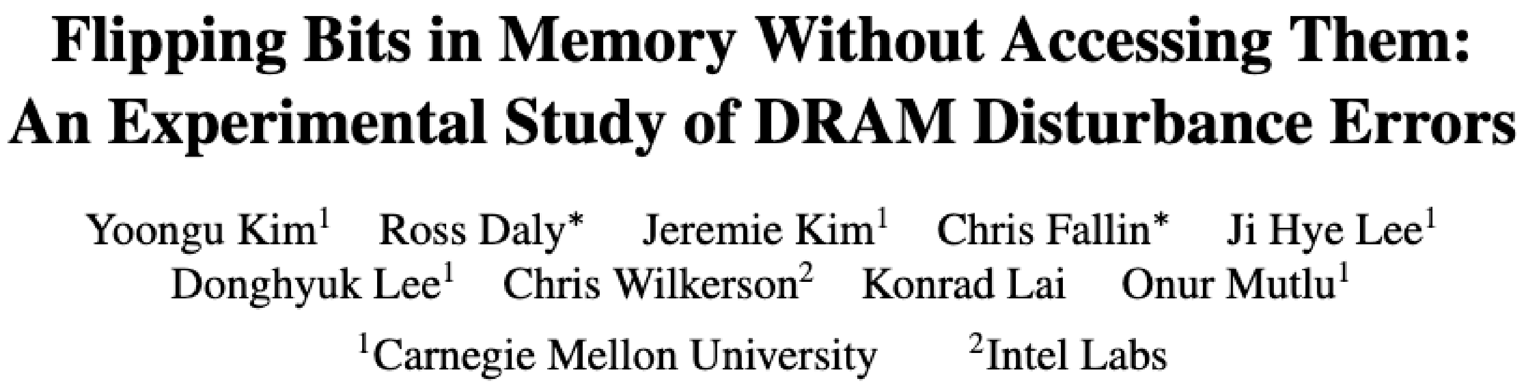


Subsequent accesses to the same row can hit in the row buffer

RowHammer


Bit flips!
Activate
Address
Row Decoder
Victim Row
Hammered Row
Victim Row
Row Buffer
Column Mux

RowHammer*



**Flipping Bits in Memory Without Accessing Them:
An Experimental Study of DRAM Disturbance Errors**

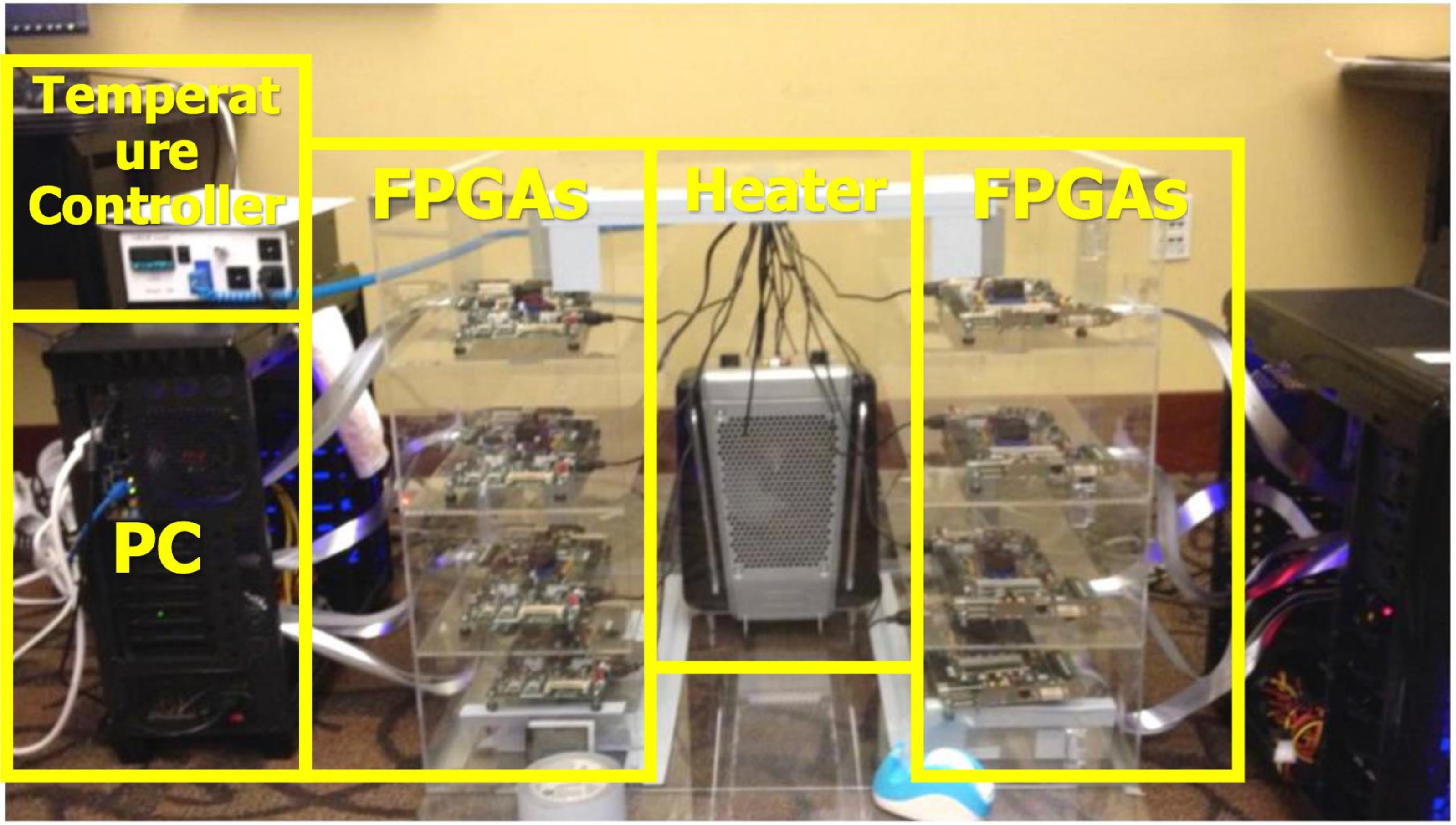
Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]

[1]Carnegie Mellon University [2]Intel Labs

Tested 129 DRAM modules from three vendors (A, B, C)
110 out 129 modules are vulnerable

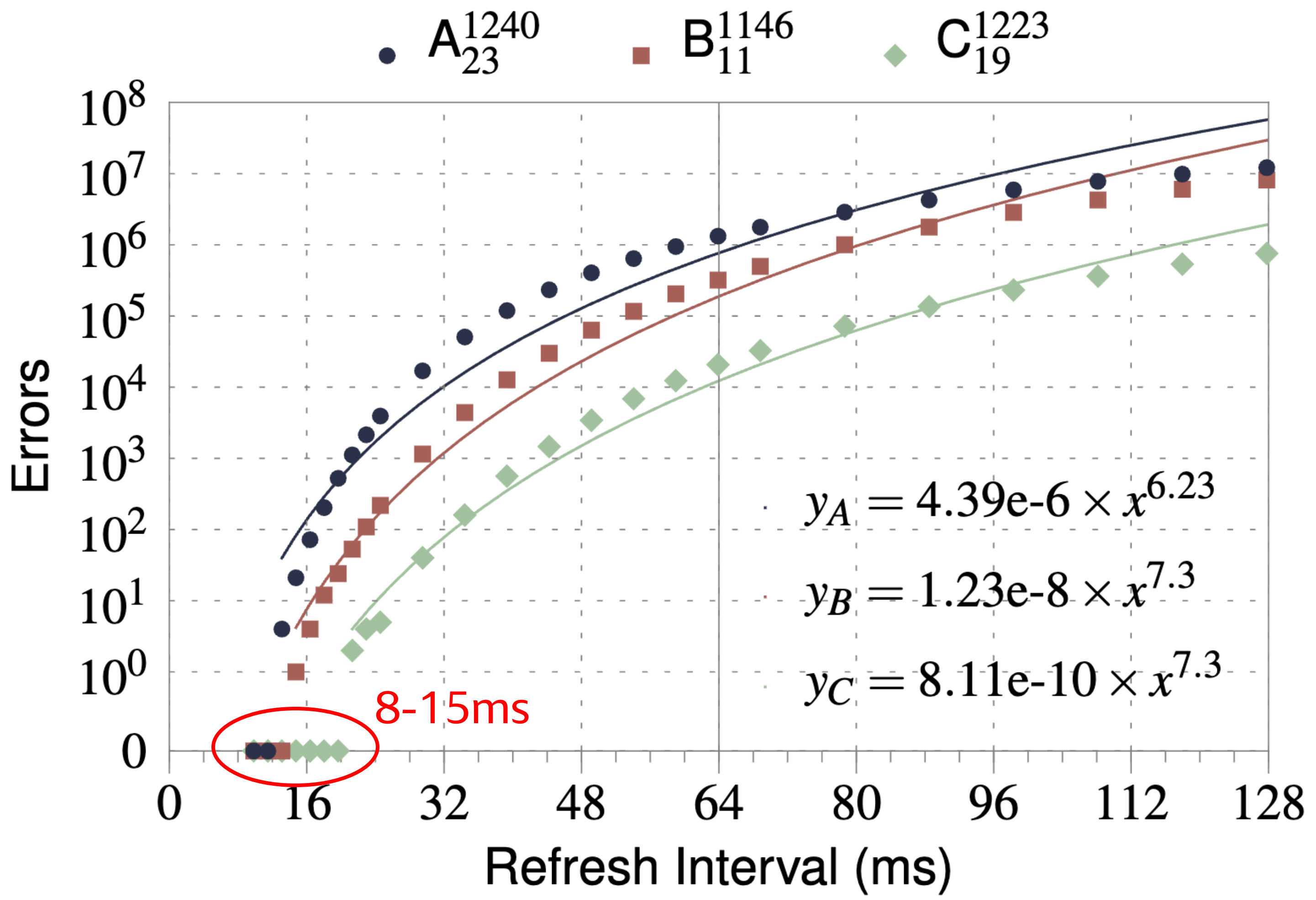
*Based on Kim et al., “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” ISCA’14

RowHammer[*]



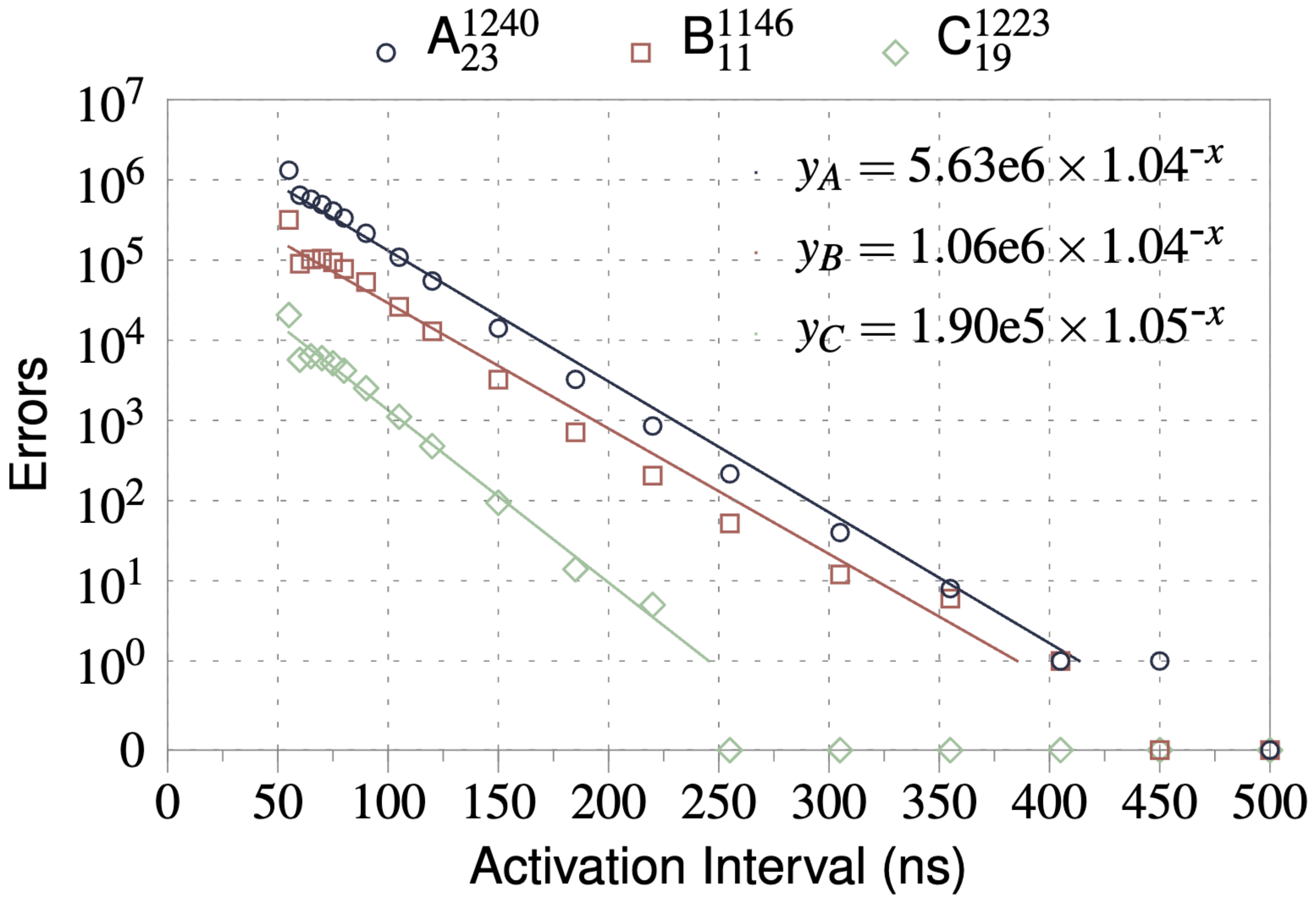
[*]Based on Kim et al., "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA'14
"The Story of RowHammer" by Onur Mutlu

RowHammer – Basic Characteristics*



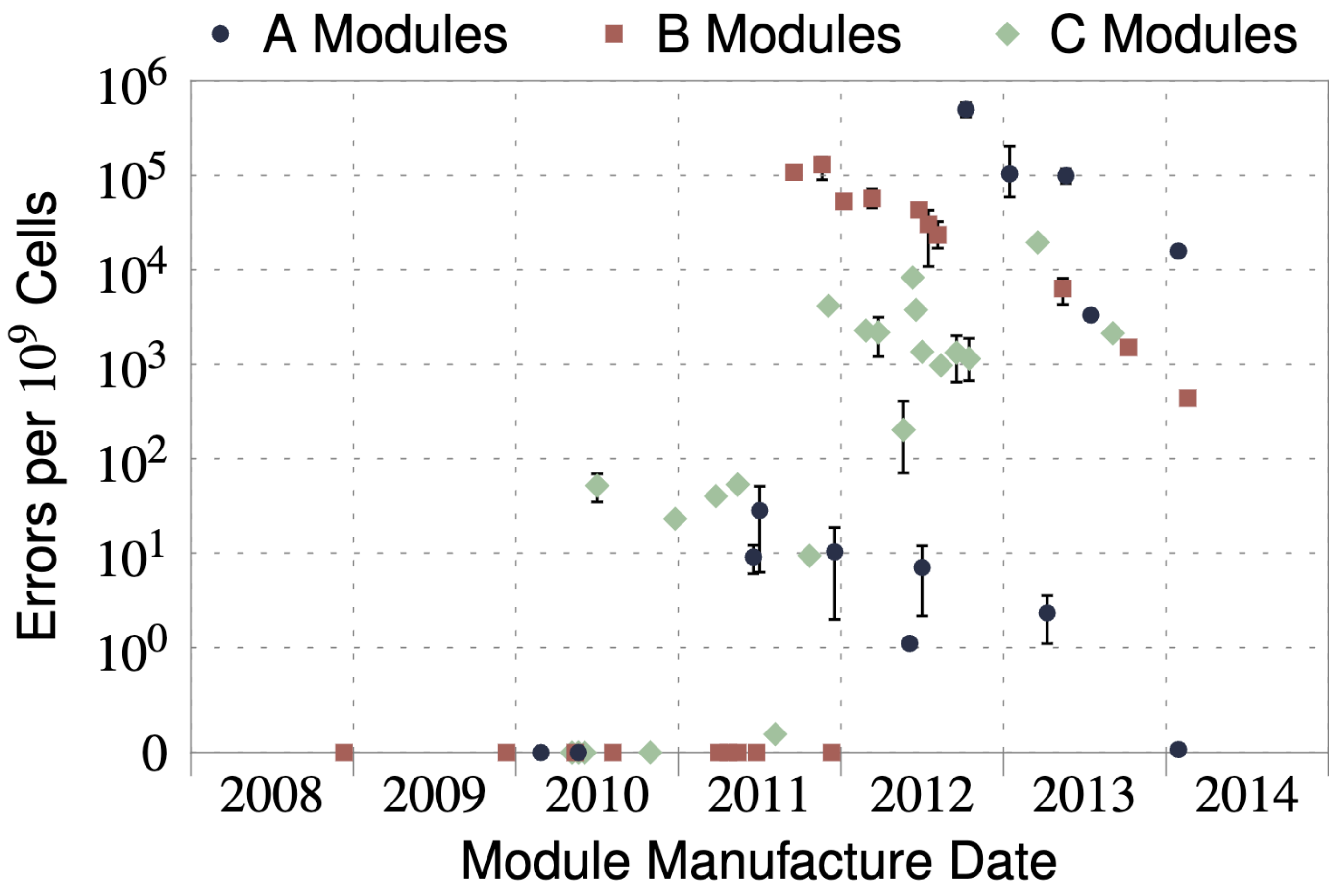
*Kim et al., "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA '14

RowHammer – Basic Characteristics*



*Kim et al., "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA '14

RowHammer – Basic Characteristics*



*Kim et al., "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA '14

RowHammer – Root Cause

- When activating a row, its adjacent rows are slightly activated
    - Adjacent rows lose a small amount of charge
    - Enough activations ⇒ Lose enough charge ⇒ Bit flip

- But still, why? Hypotheses from Kim et al.:
    - Electromagnetic coupling
    - Bridging
    - Hot-carrier damage

RowHammer – Other Characteristics*

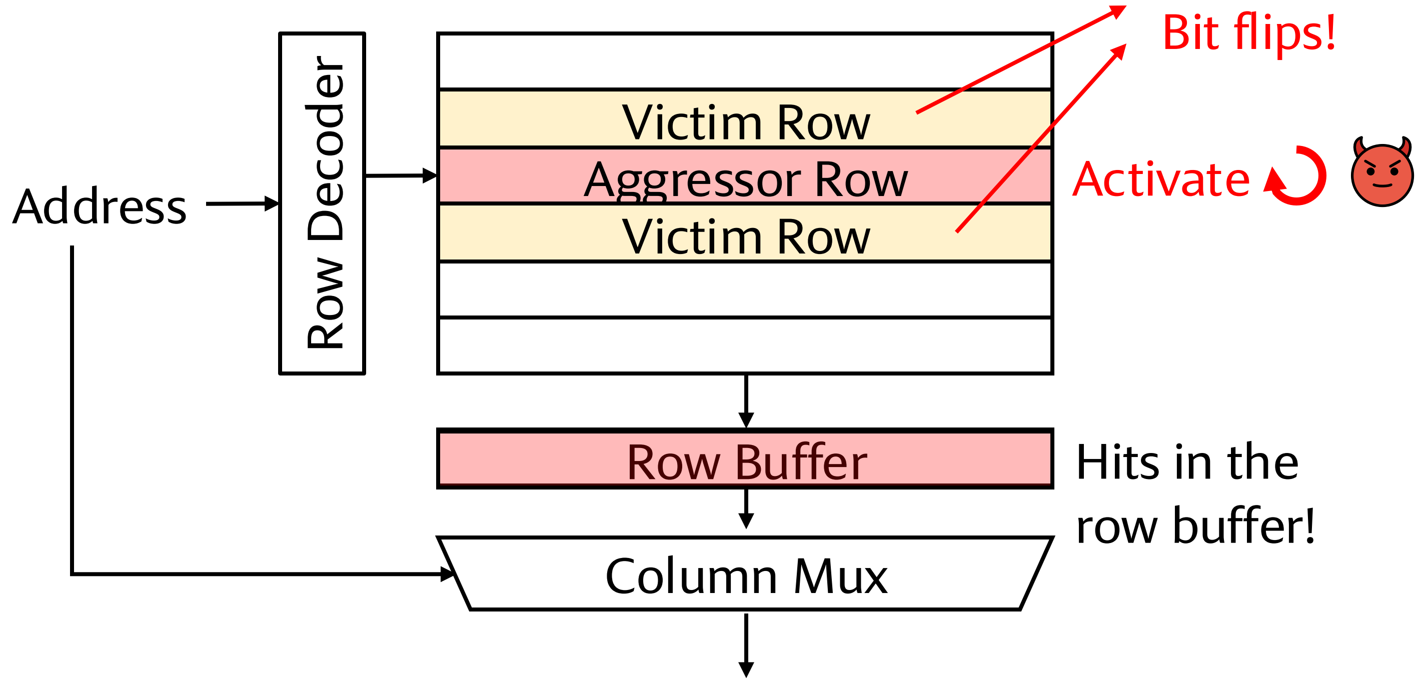
- Bit flips are repeatable
- Bit flip direction:
  - Module A: 1 → 0 (49.9%)
  - Module B: 1 → 0 (92.8%)
  - Module C: 1 → 0 (97.1%)
- Data-dependent pattern:

>10x more bit flips!

~Solid	Solid	Row Stripe
00000	11111	11111
00000	11111	00000
00000	11111	11111
00000	11111	00000

*Based on Kim et al., “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” ISCA’14

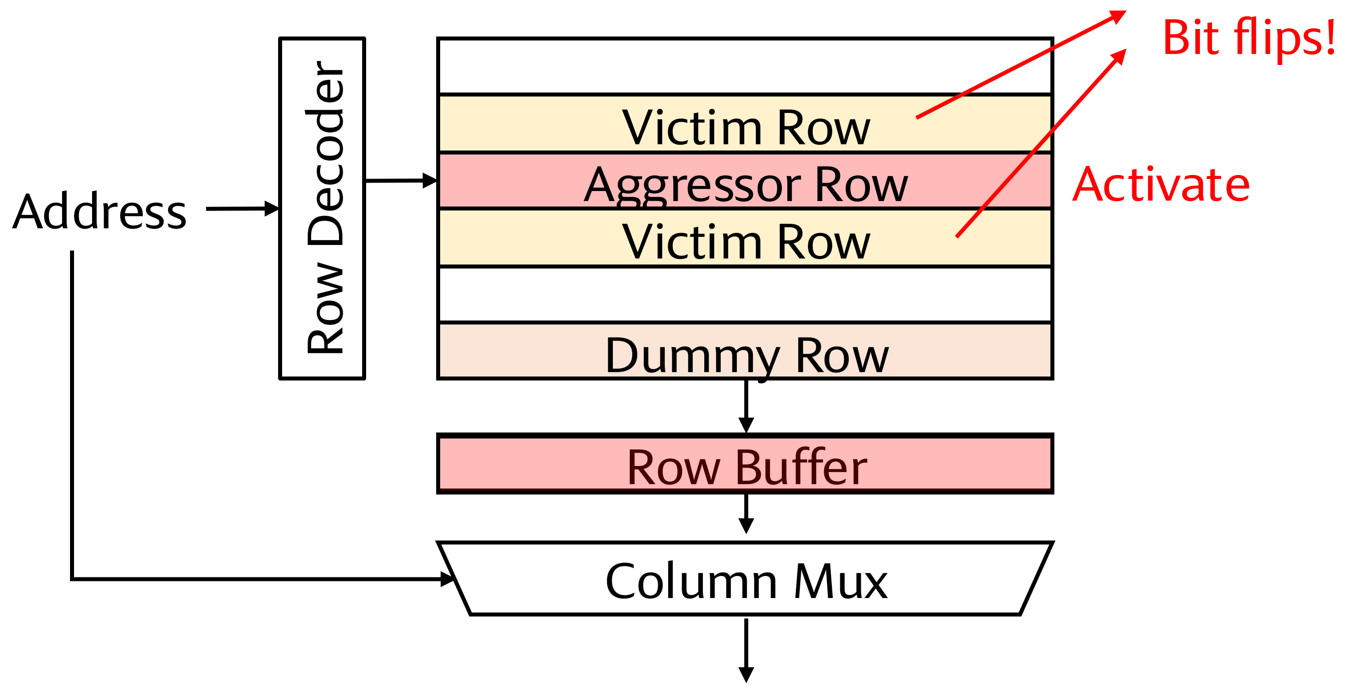
Hammering Sequence



```
while (true) {
  load hammer_row;

  clflush hammer_row;
  mfence;
}
```

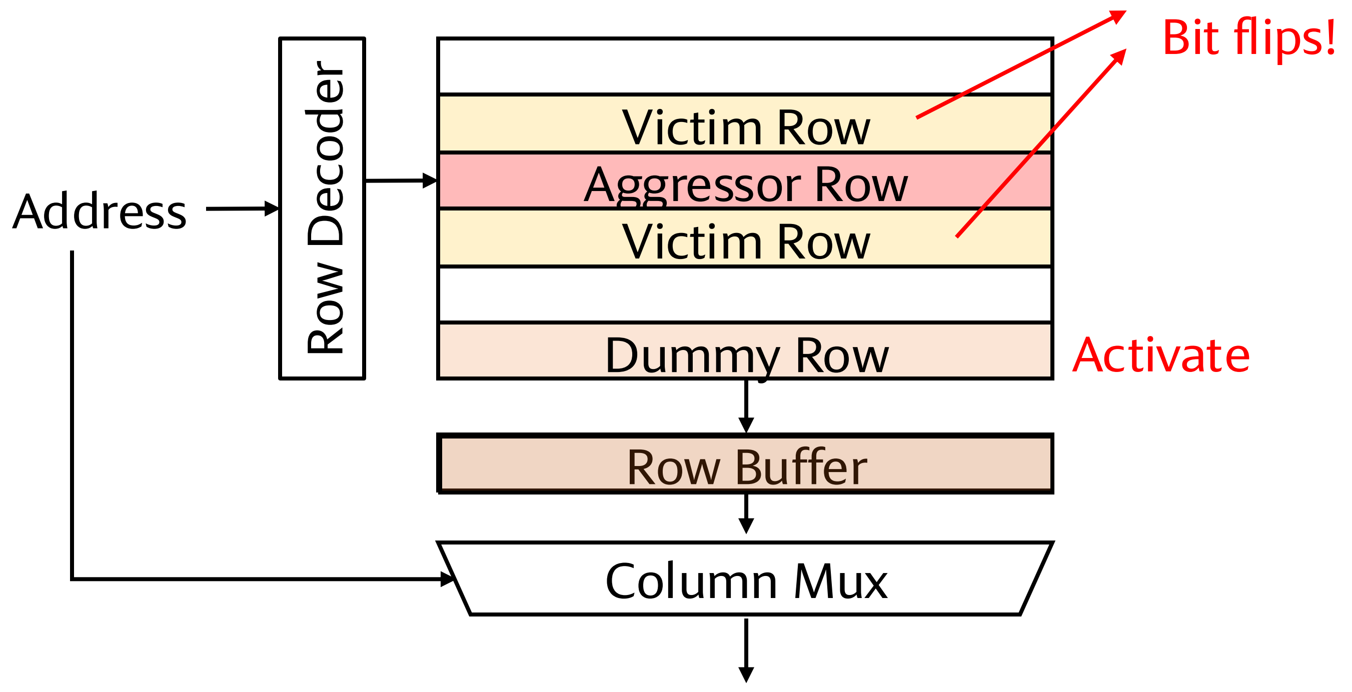
Hammering Sequence



```
while (true) {
  load hammer_row;
  load dummy_row;

  clflush hammer_row;
  clflush dummy_row;
  mfence;
}
```

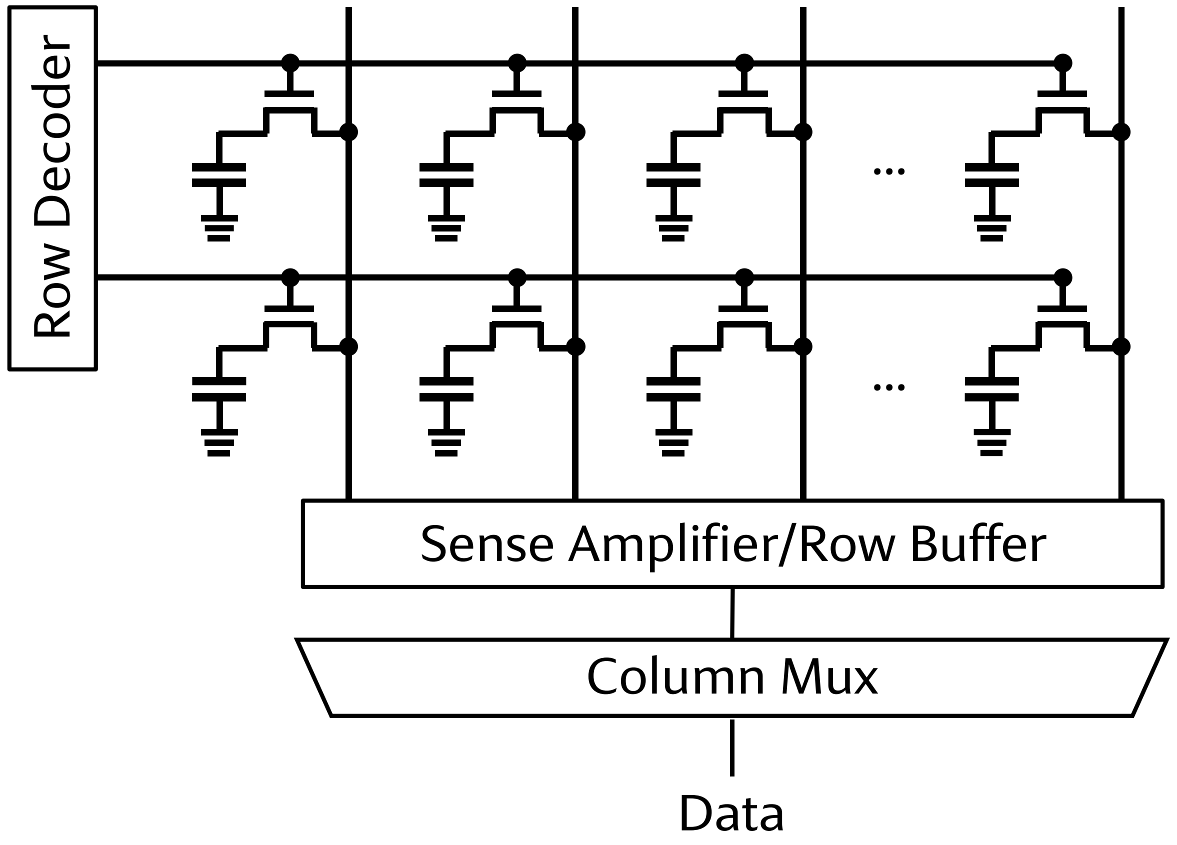
Hammering Sequence



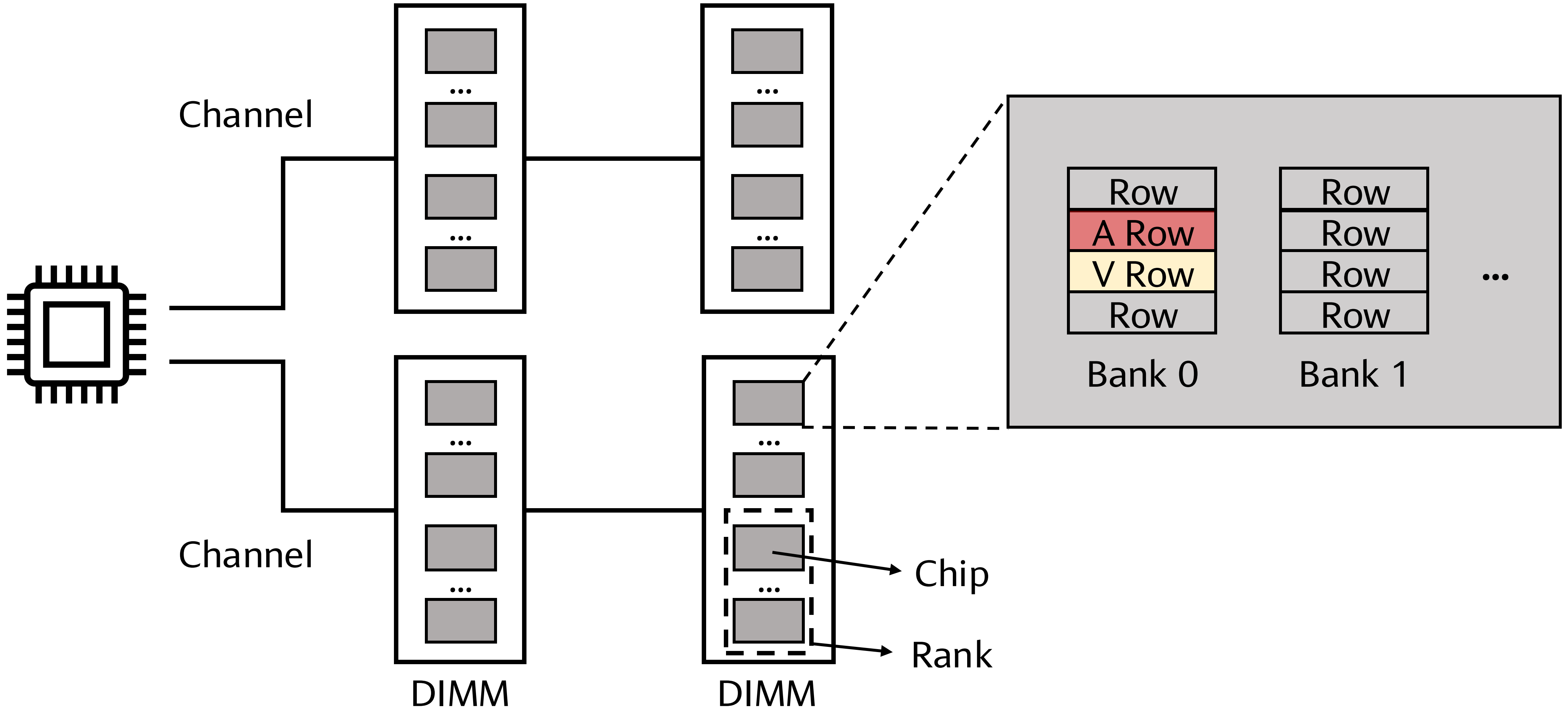
```
while (true) {
  load hammer_row;
  load dummy_row;

  clflush hammer_row;
  clflush dummy_row;
  mfence;
}
```

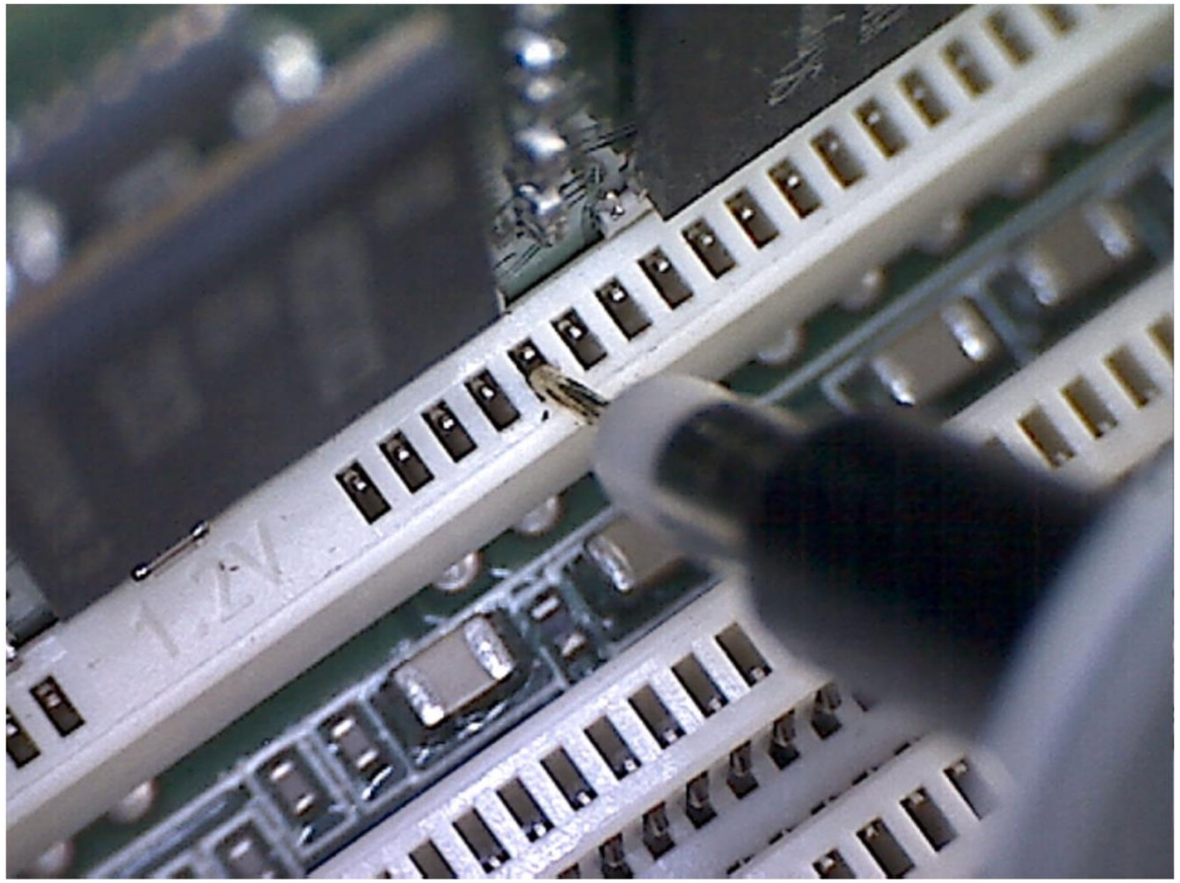
DRAM Mapping Function



DRAM Mapping Function



Recover the Mapping Function



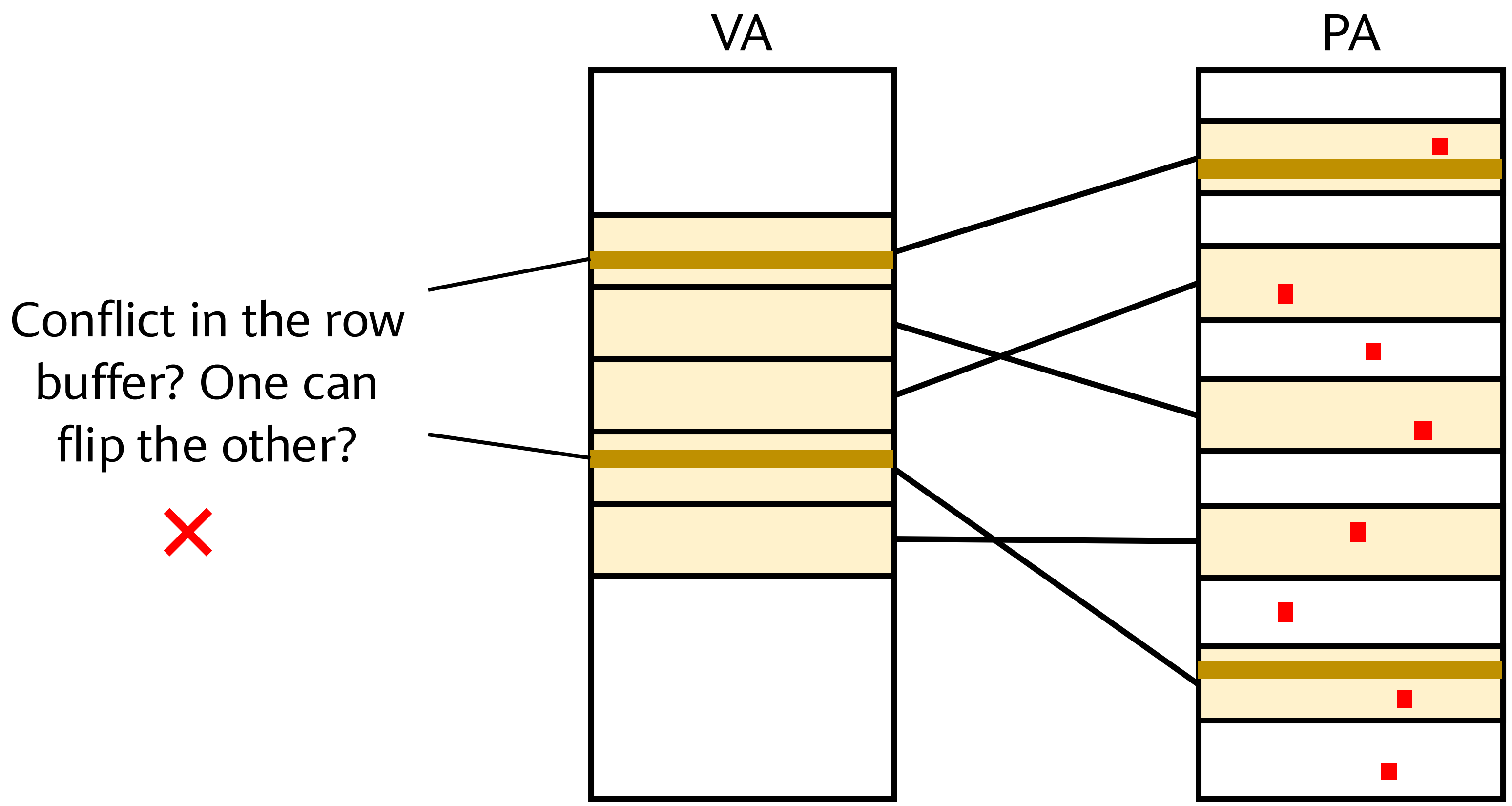
Method 1 (physical access): Repeated accesses to a known physical address, observe which rank and bank are used through **physical probing**

Method 2 (software-only): Detect addresses that are mapped to the same bank through row buffer conflicts
⇒ Timing difference

Source: Pessl et al, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," USENIX '16

Finding Aggressor

Observation: Bit flips are repeatable

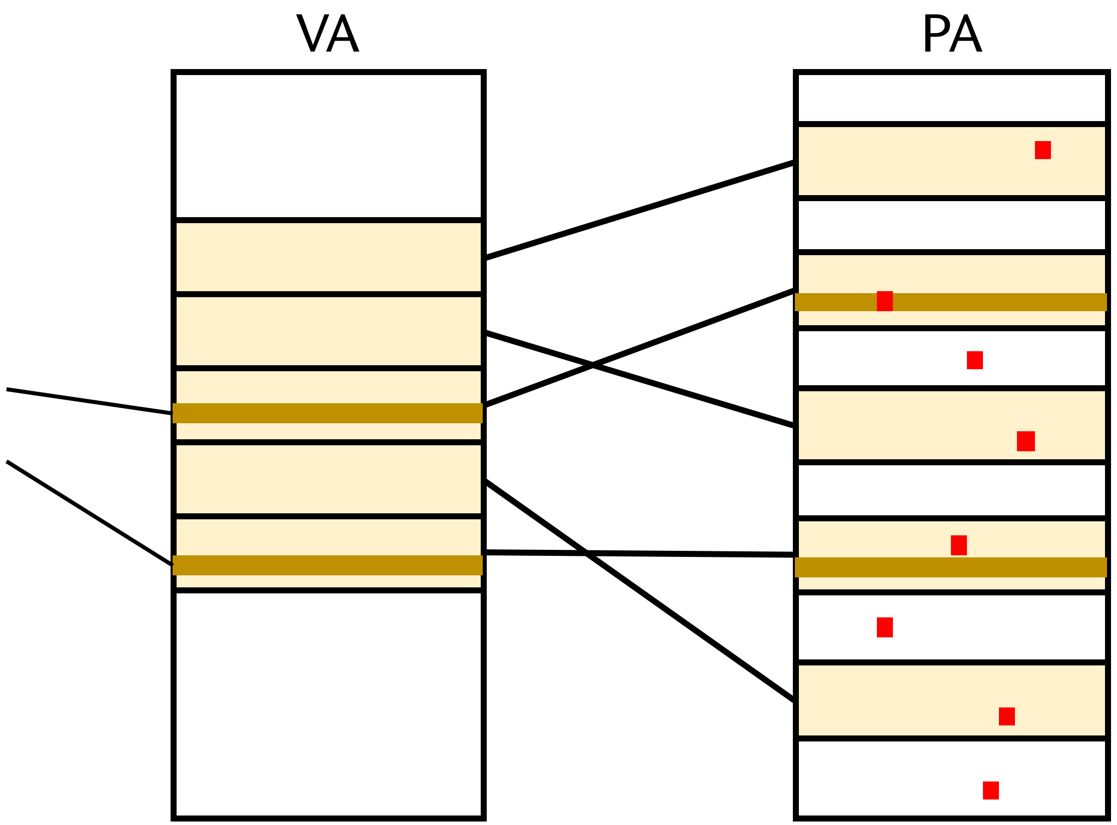


Finding Aggressor

Observation: Bit flips are repeatable

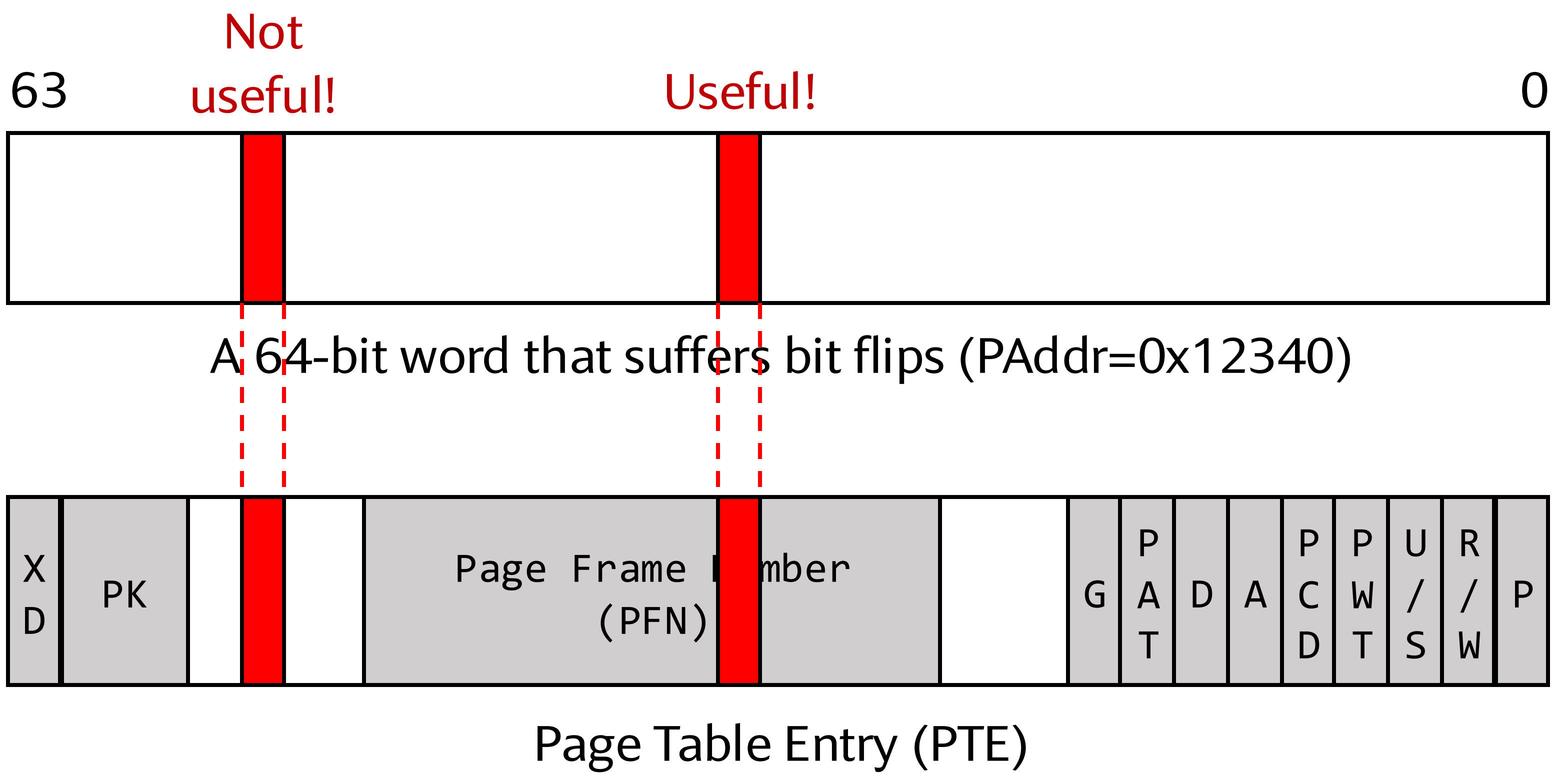
Conflict in the row buffer? One can flip the other?

✓



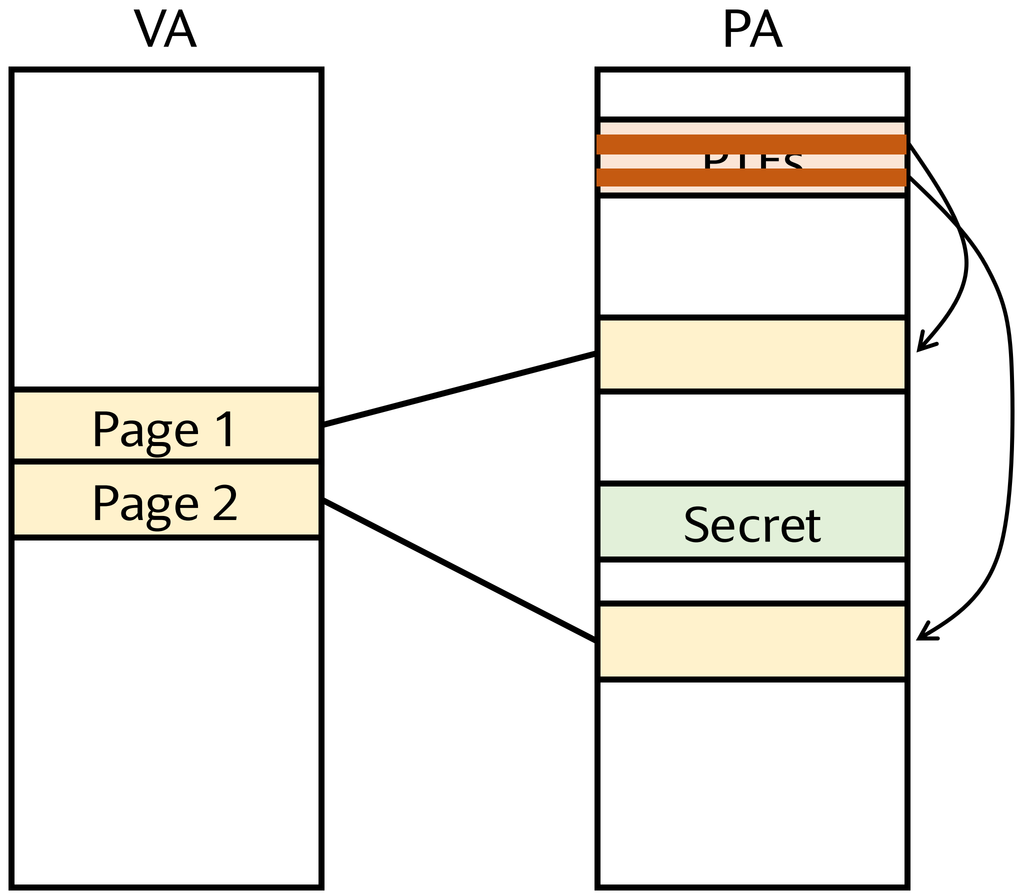
Is This Bit Flip Useful?

Goal: Corrupt the page-frame number of a page-table entry (PTE)



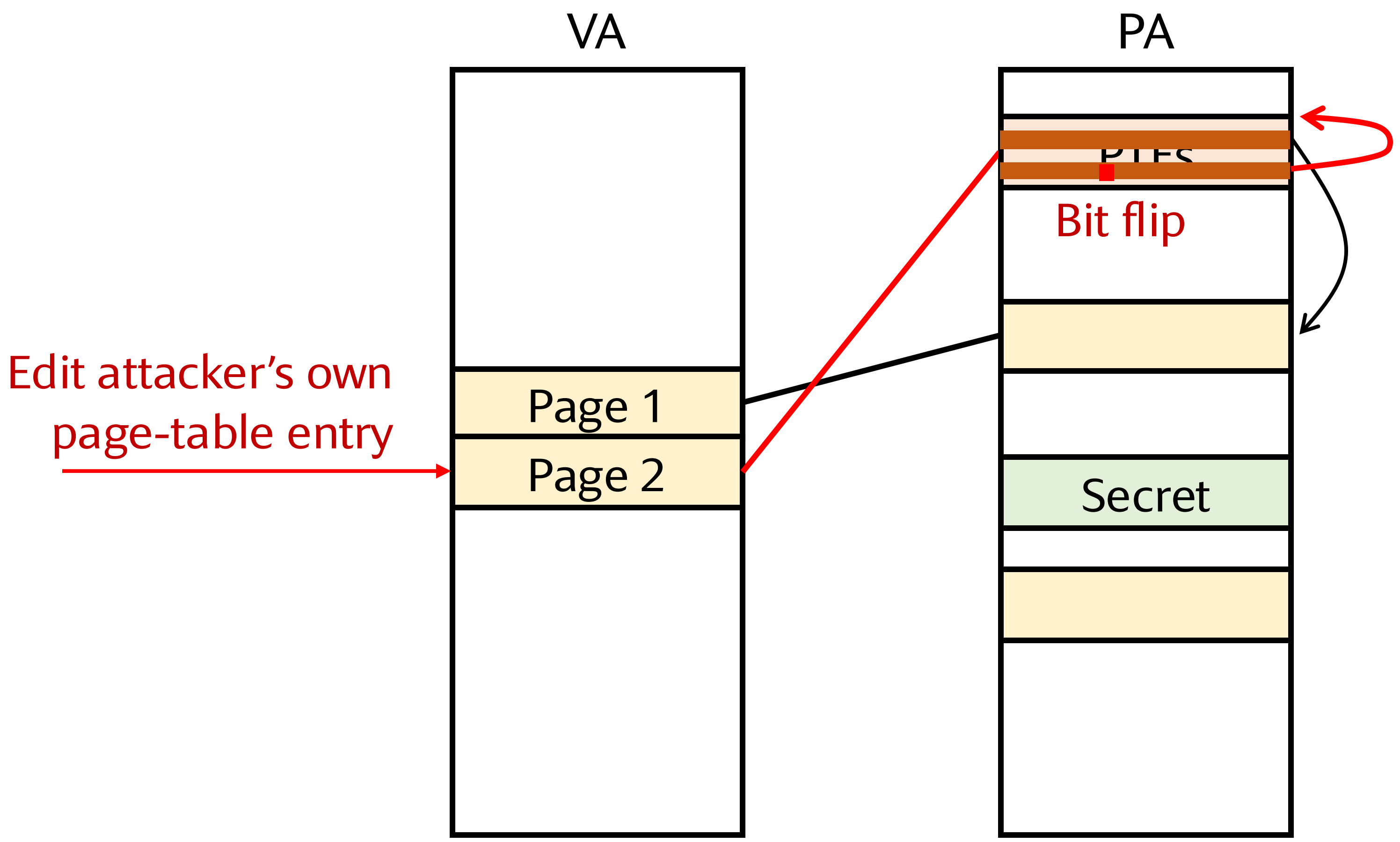
[*]https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

Goal: Corrupt the Page Table Entry



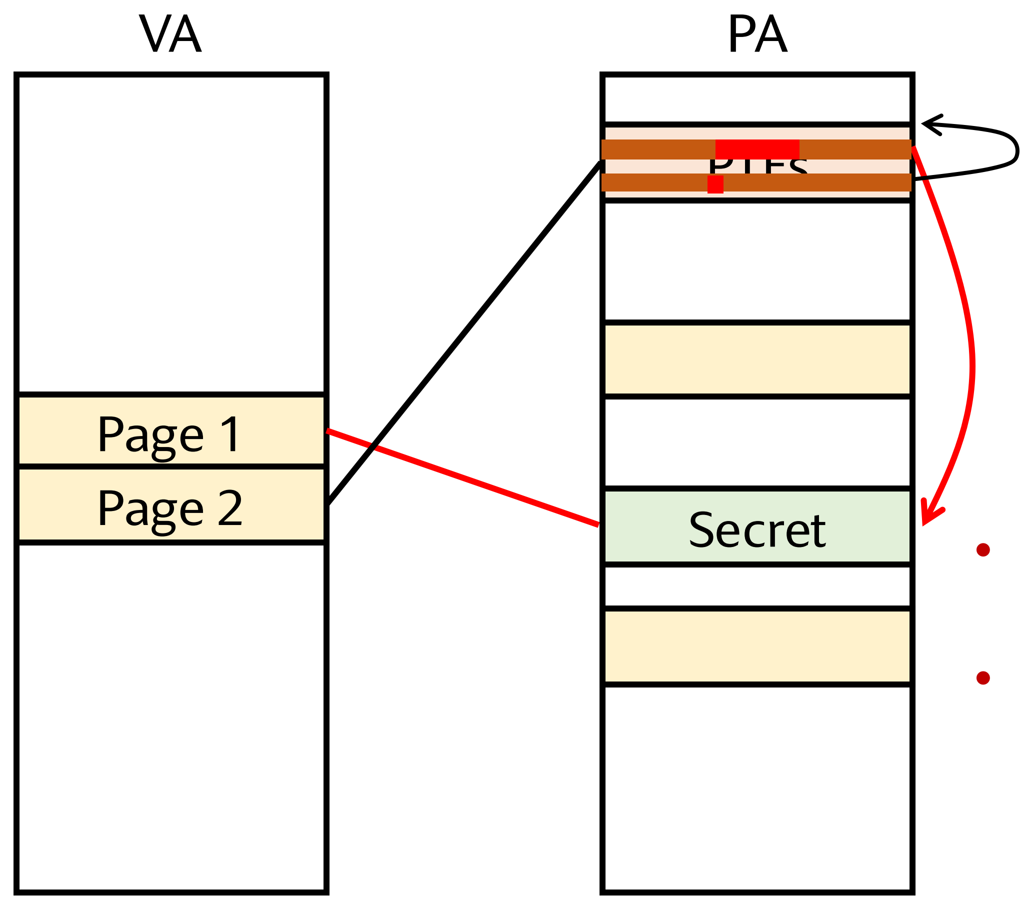
*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

Goal: Corrupt the Page Table Entry



*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

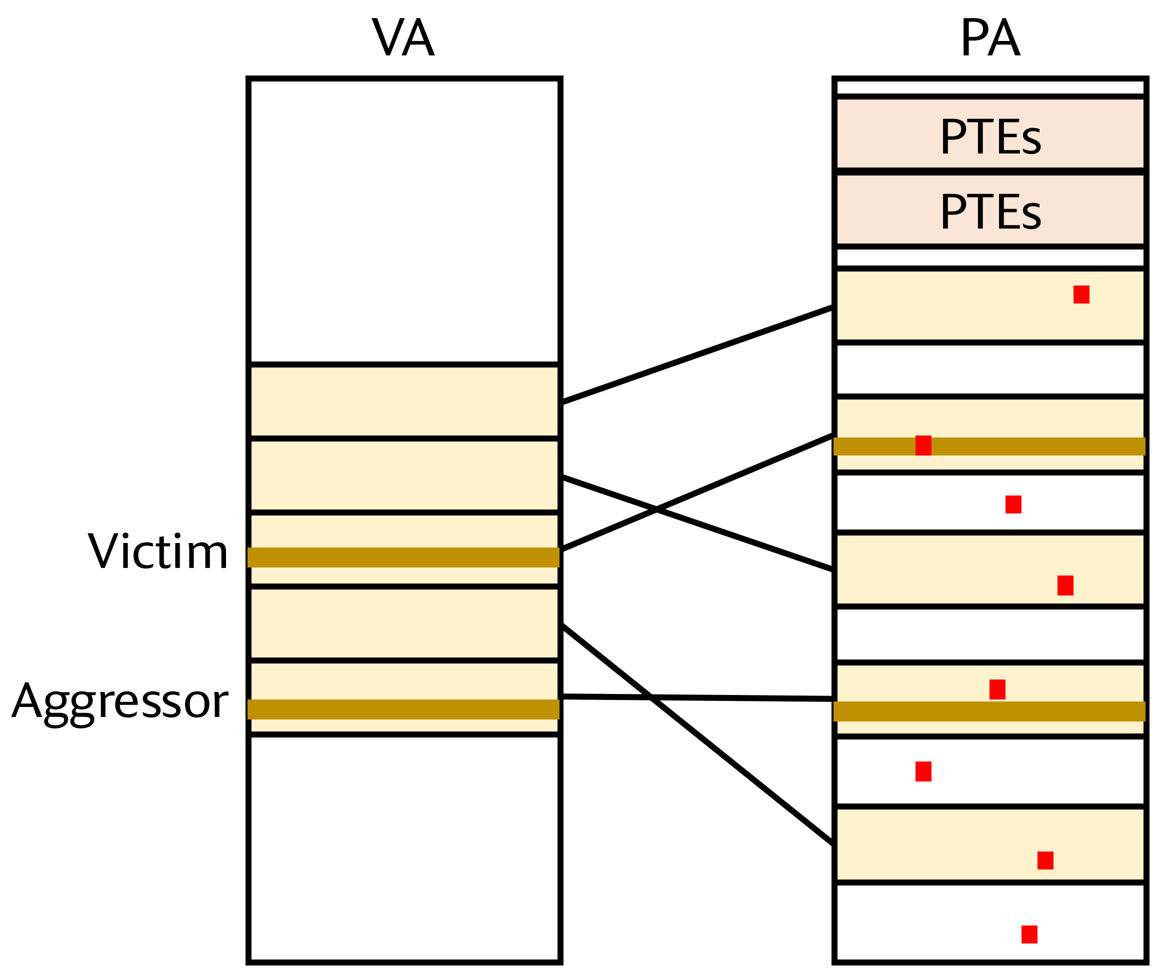
Goal: Corrupt the Page Table Entry



- How to trick the OS to put the PTE at a vulnerable address?
- How to ensure that the corrupted PTE points to a leaf page-table page?

*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

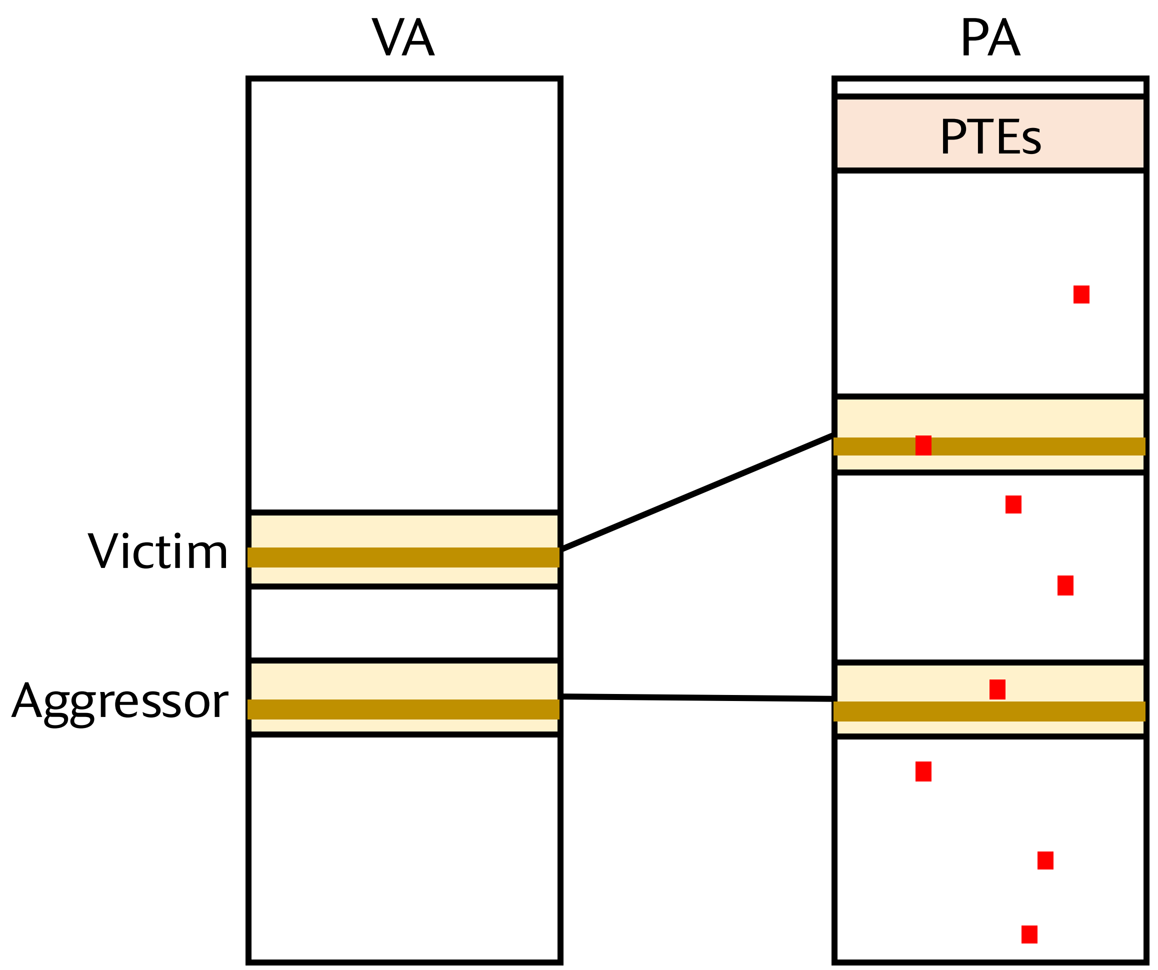
Memory Massaging



- Record the aggressor and victim addresses

*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

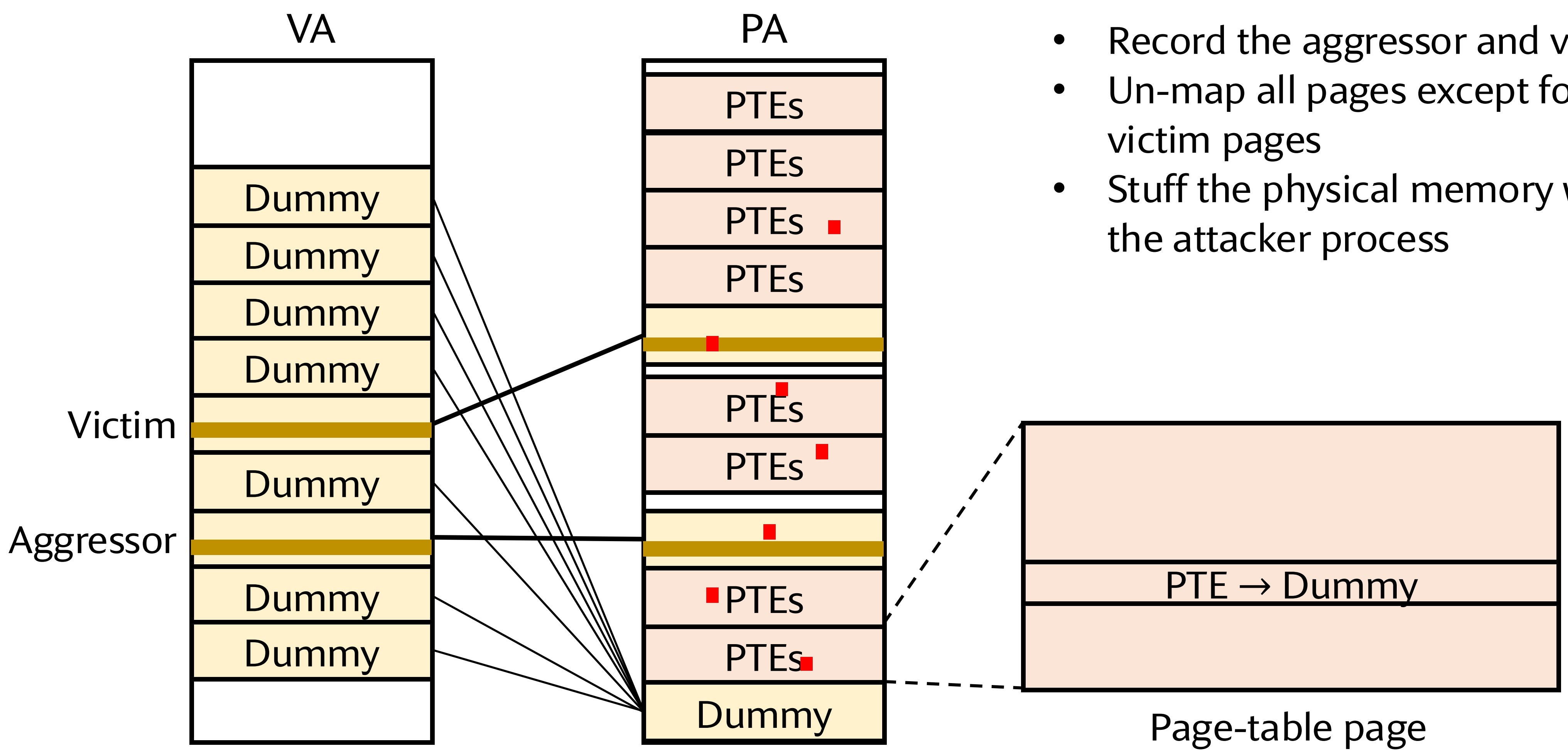
Memory Massaging



- Record the aggressor and victim addresses
- Un-map all pages except for aggressor and victim pages

*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

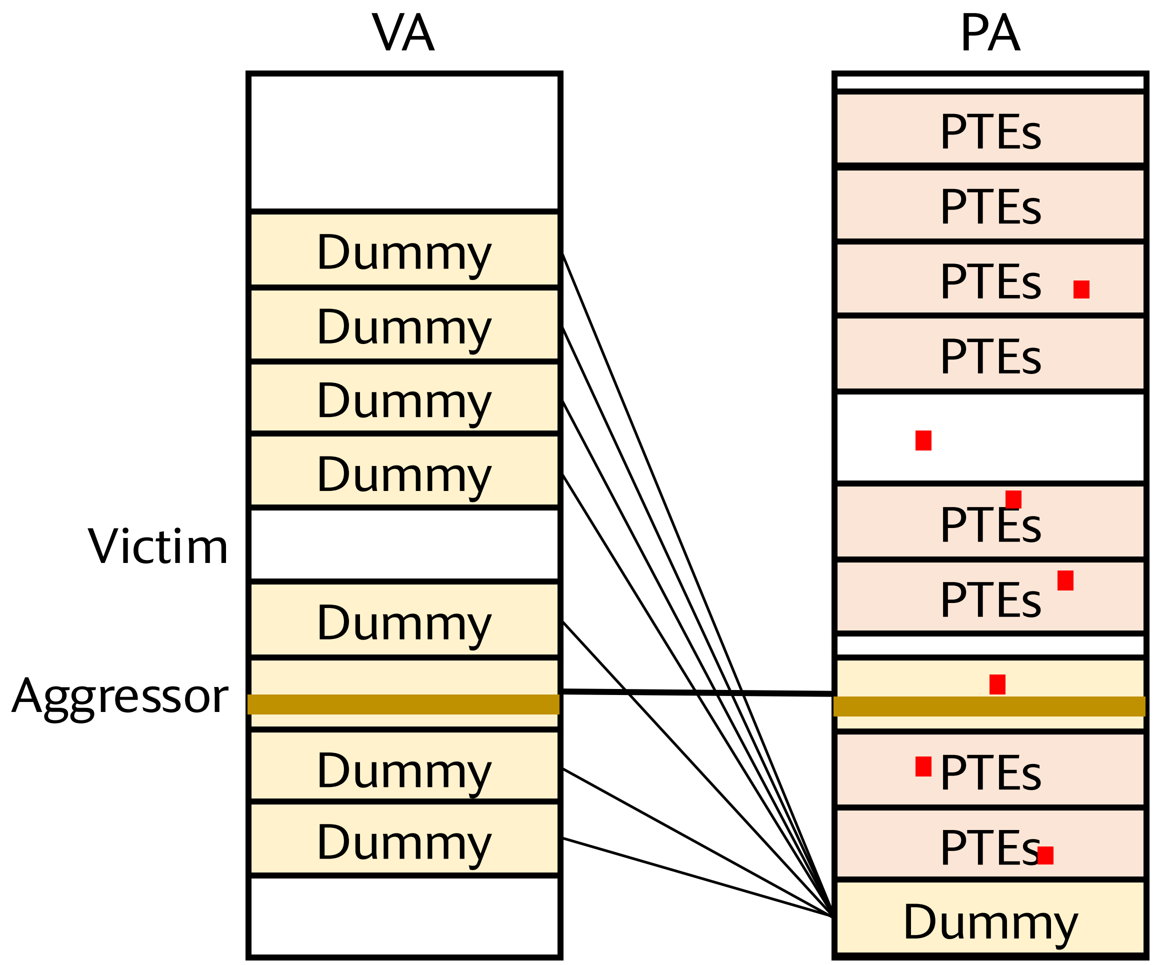
Memory Massaging



- Record the aggressor and victim addresses
- Un-map all pages except for aggressor and victim pages
- Stuff the physical memory with the PTEs of the attacker process

*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

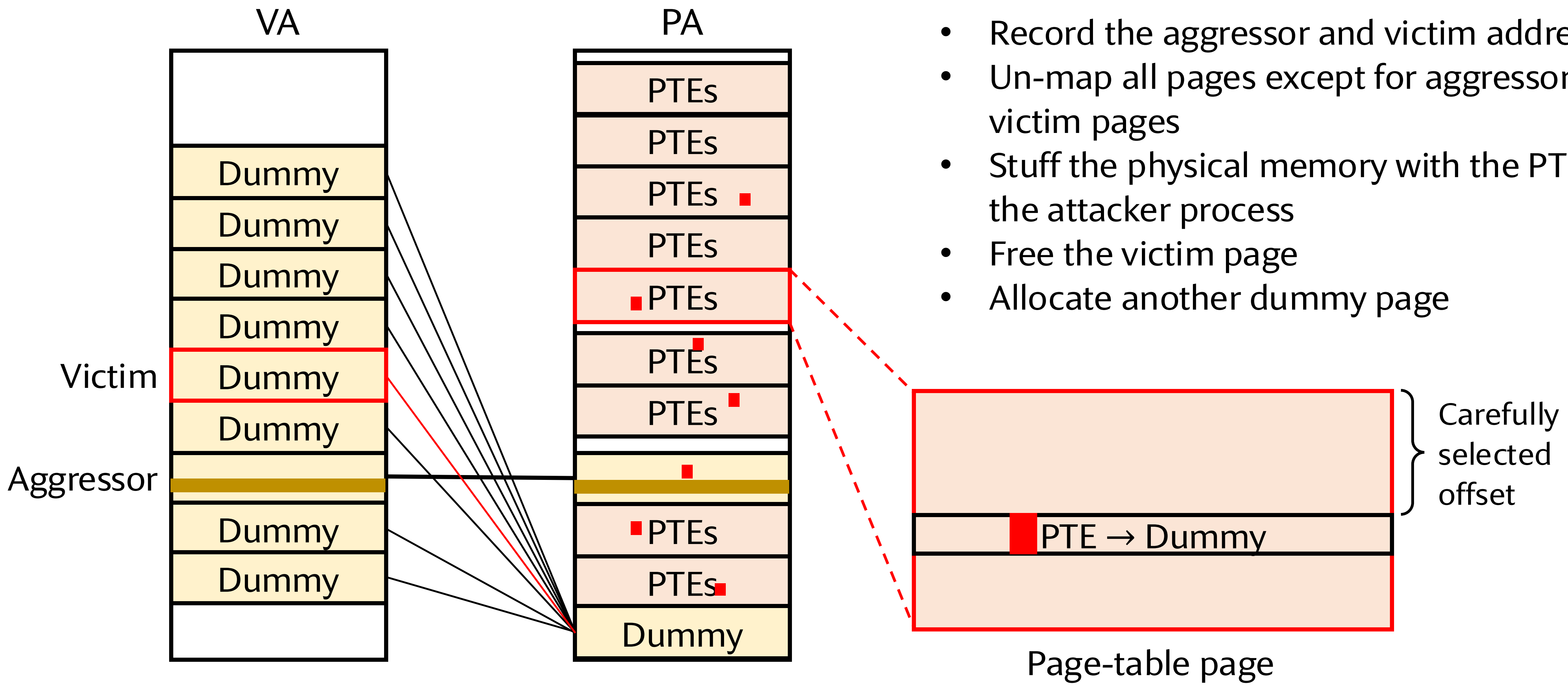
Memory Massaging



- Record the aggressor and victim addresses
- Un-map all pages except for aggressor and victim pages
- Stuff the physical memory with the PTEs of the attacker process
- Free the victim page

*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

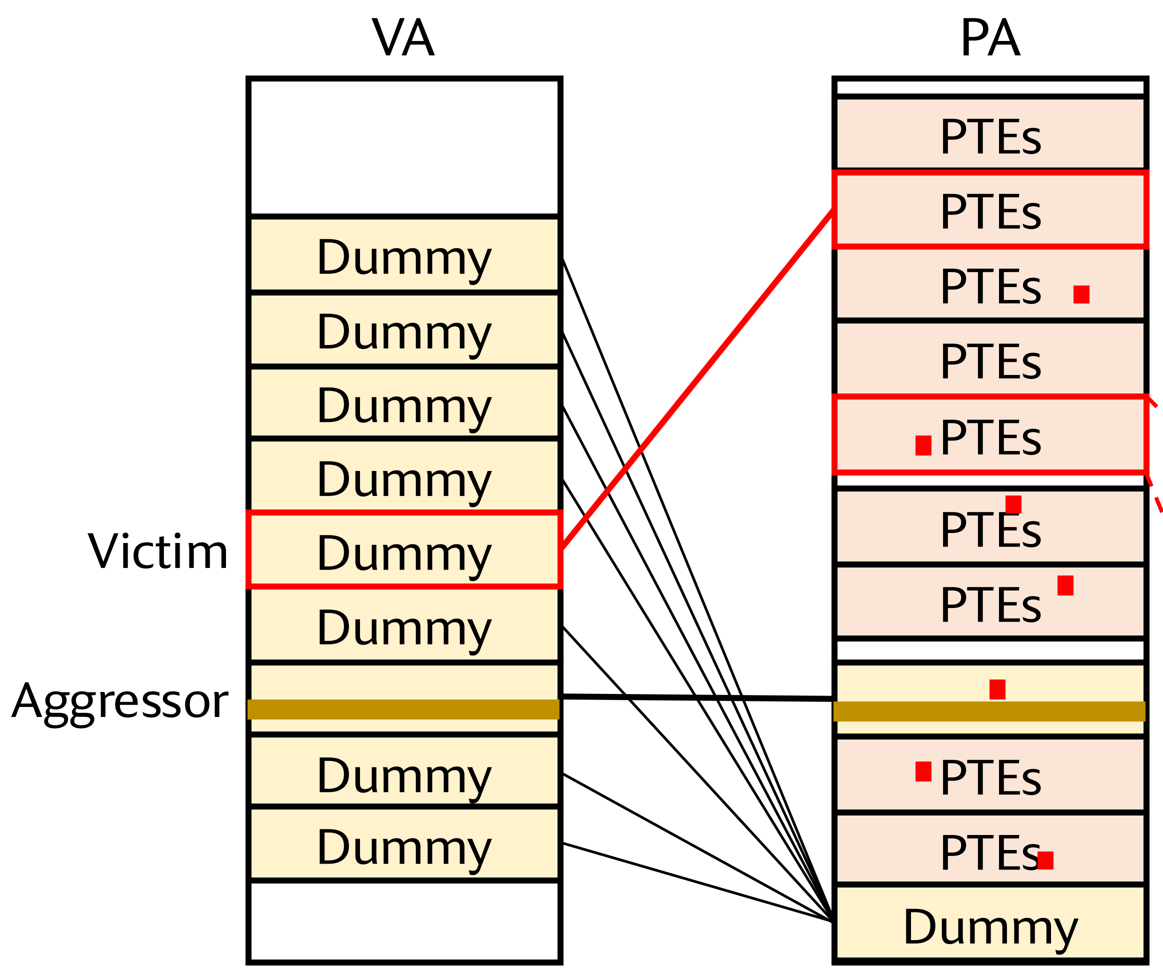
Memory Massaging



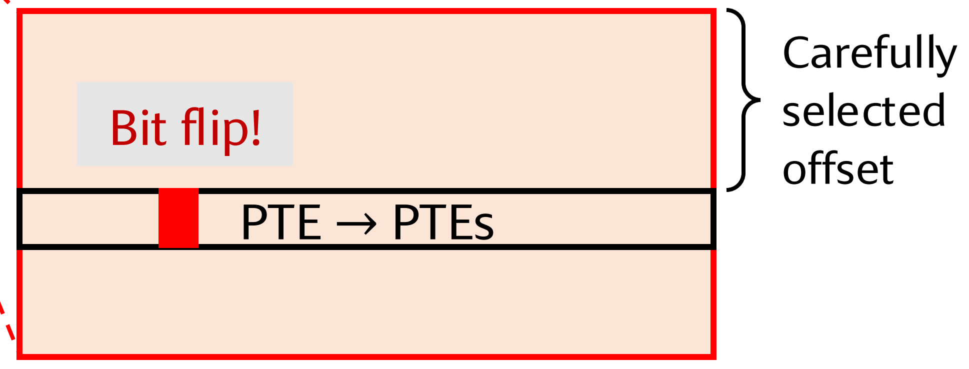
- Record the aggressor and victim addresses
- Un-map all pages except for aggressor and victim pages
- Stuff the physical memory with the PTEs of the attacker process
- Free the victim page
- Allocate another dummy page

*https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

Memory Massaging

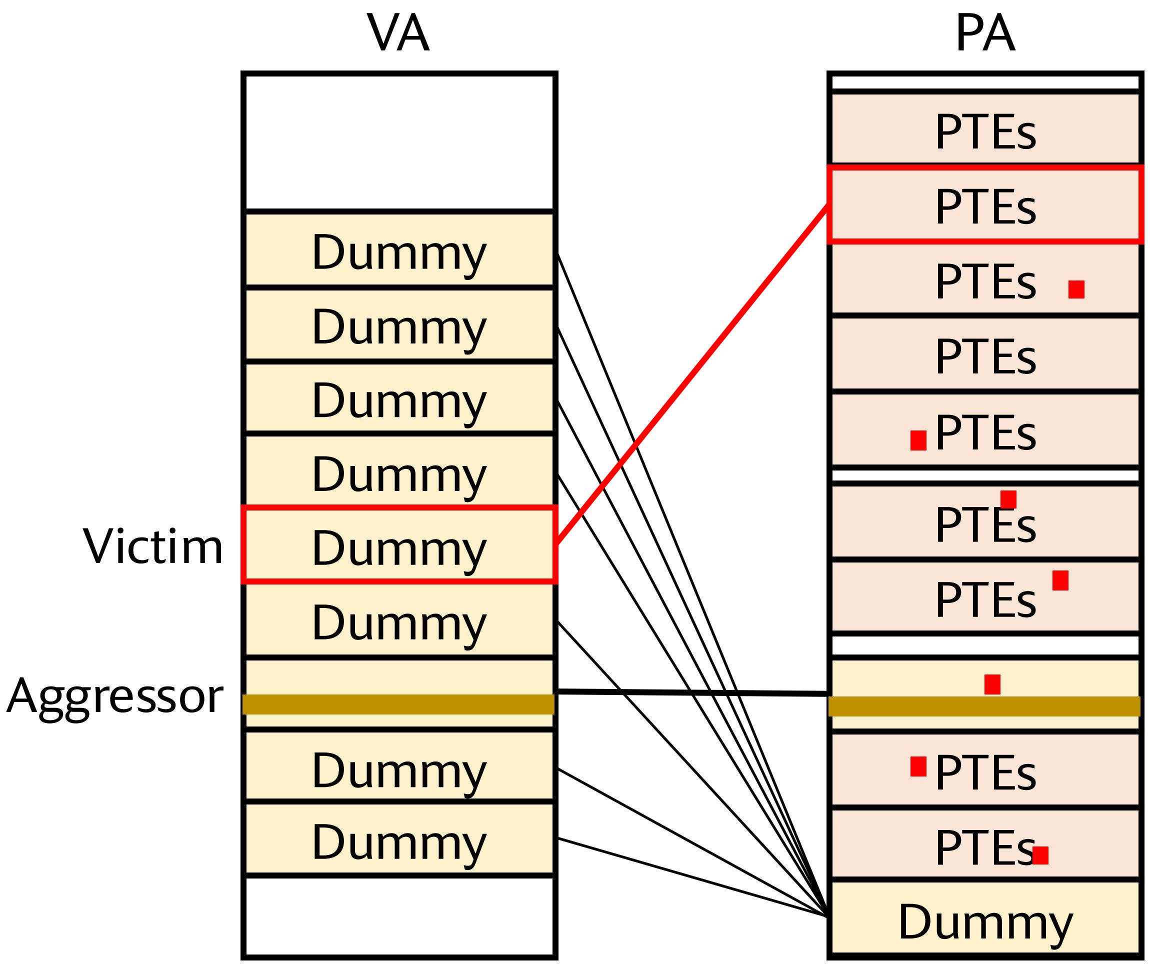


- Record the aggressor and victim addresses
- Un-map all pages except for aggressor and victim pages
- Stuff the physical memory with the PTEs of the attacker process
- Free the victim page
- Allocate another dummy page
- Hammer with the aggressor



Page-table page

Memory Massaging



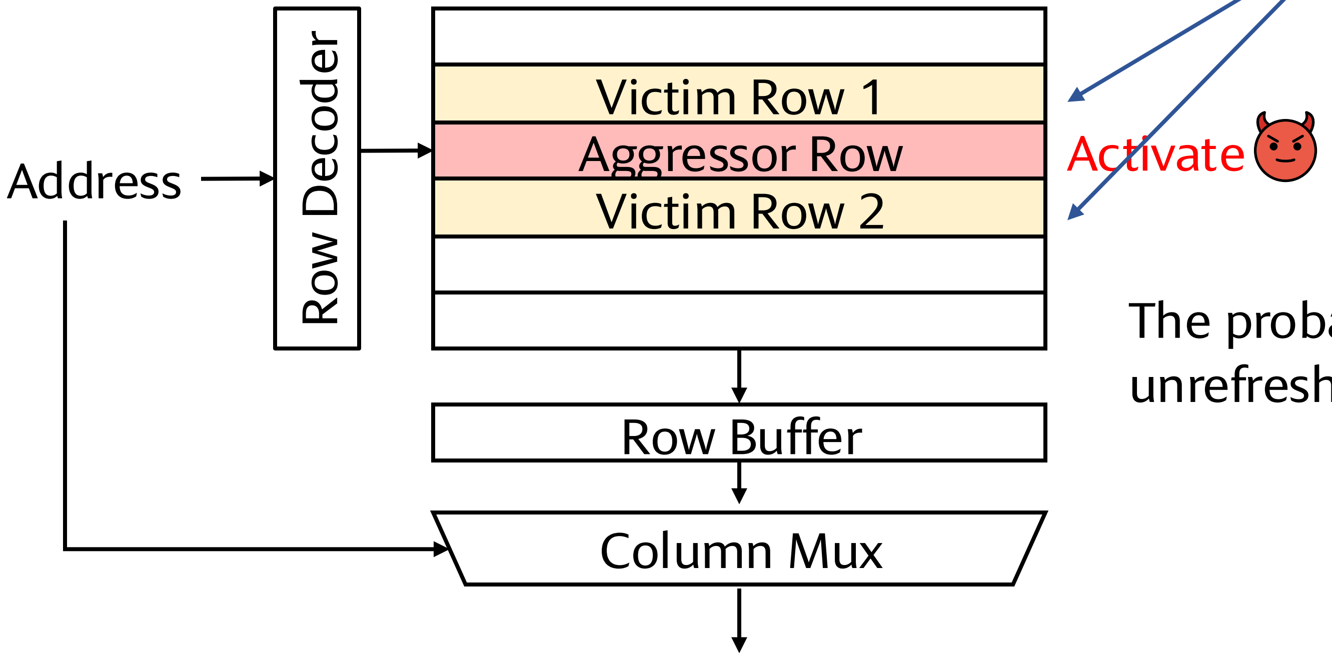
- Record the aggressor and victim addresses
- Un-map all pages except for aggressor and victim pages
- Stuff the physical memory with the PTEs of the attacker process
- Free the victim page
- Allocate another dummy page
- Hammer with the aggressor

⇒ The attacker process can write to its own page table entries
⇒ Access arbitrary physical address

Naïve Defenses Against RowHammer

- Build better chips
  - ⇒ Cost
- Error Correction Code (ECC)
  - ⇒ Cannot correct multi-bit errors, storage overhead
- Reduced refresh interval
  - ⇒ Increase the “downtime” due to refreshing
- Removing clflush
  - ⇒ Eviction sets
- …

Probabilistic Adjacent Row Activation (PARA)



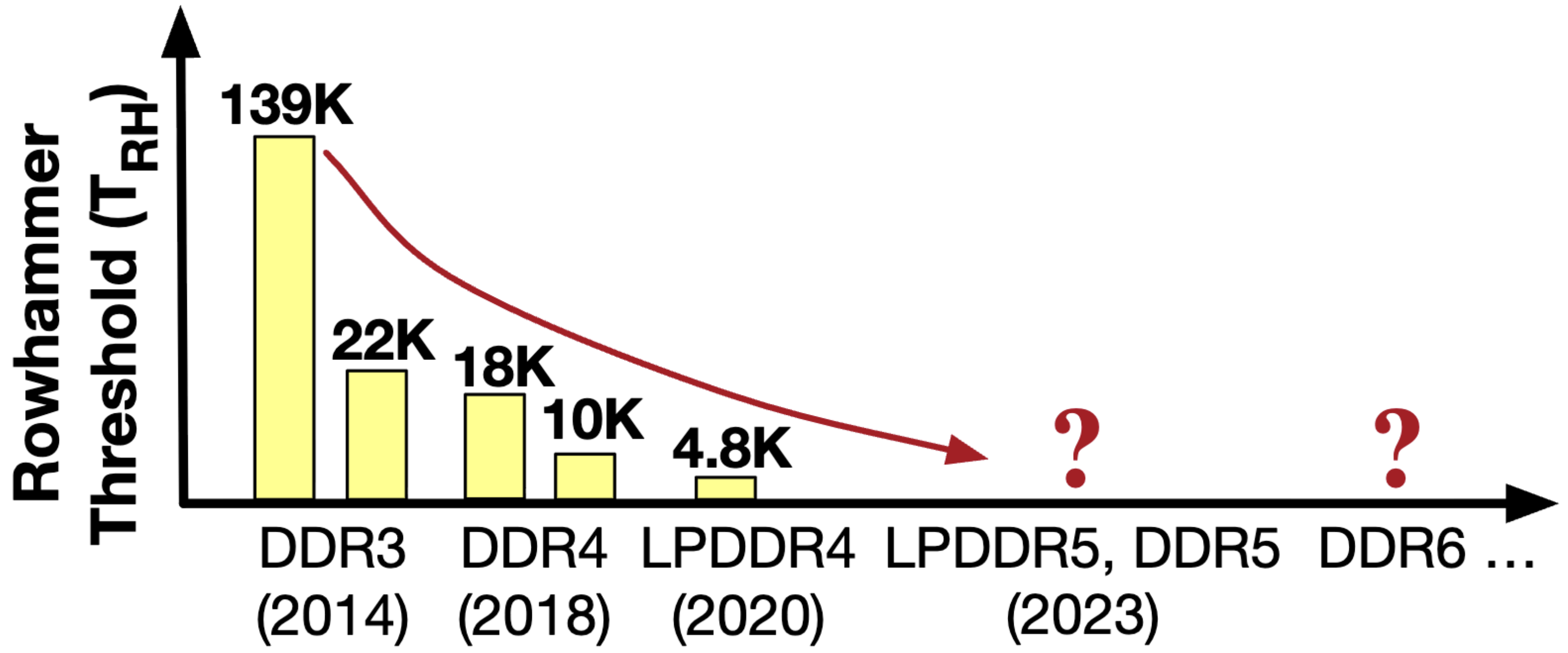
Refresh an adjacent row with probability p
⇒ Randomly pick one of the adjacent rows

The probability that victim row 1 remains unrefreshed after N aggressor row activations:

$$\left(1 - \frac{p}{2}\right)^N$$

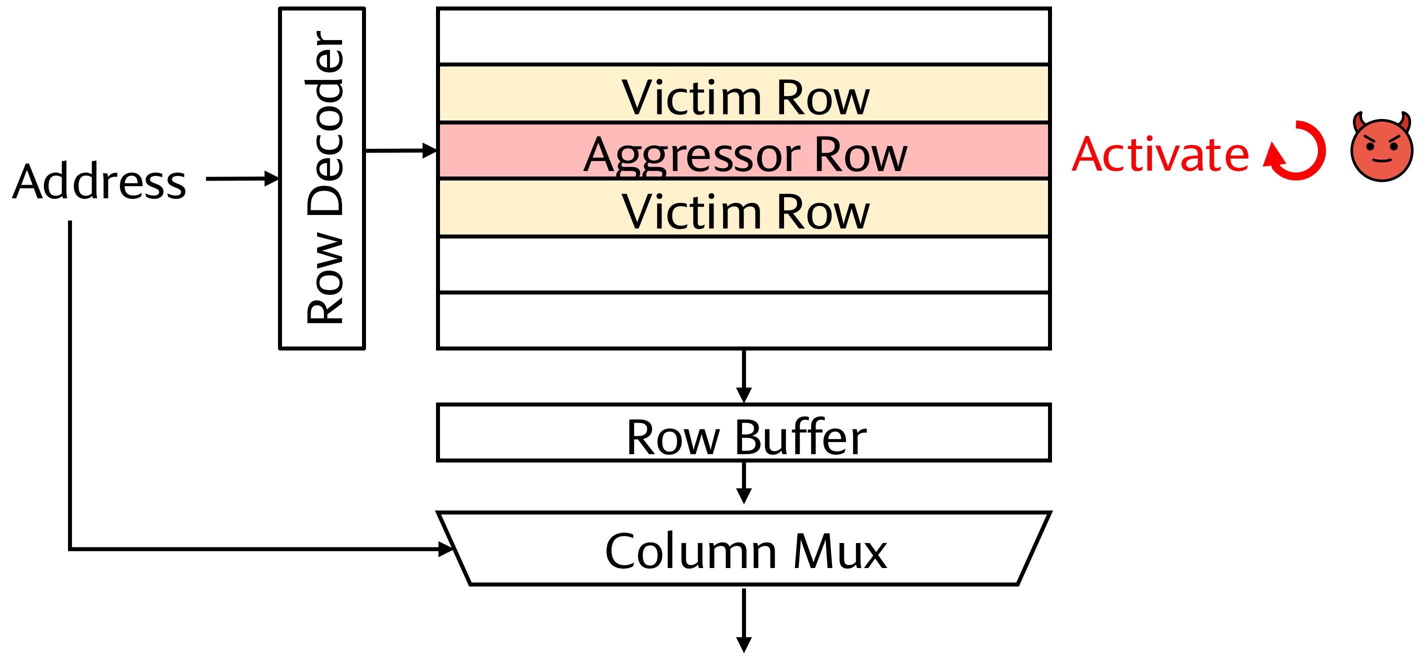
Larger p ⇒ stronger protection, higher performance overhead

The Trend of RowHammer Threshold

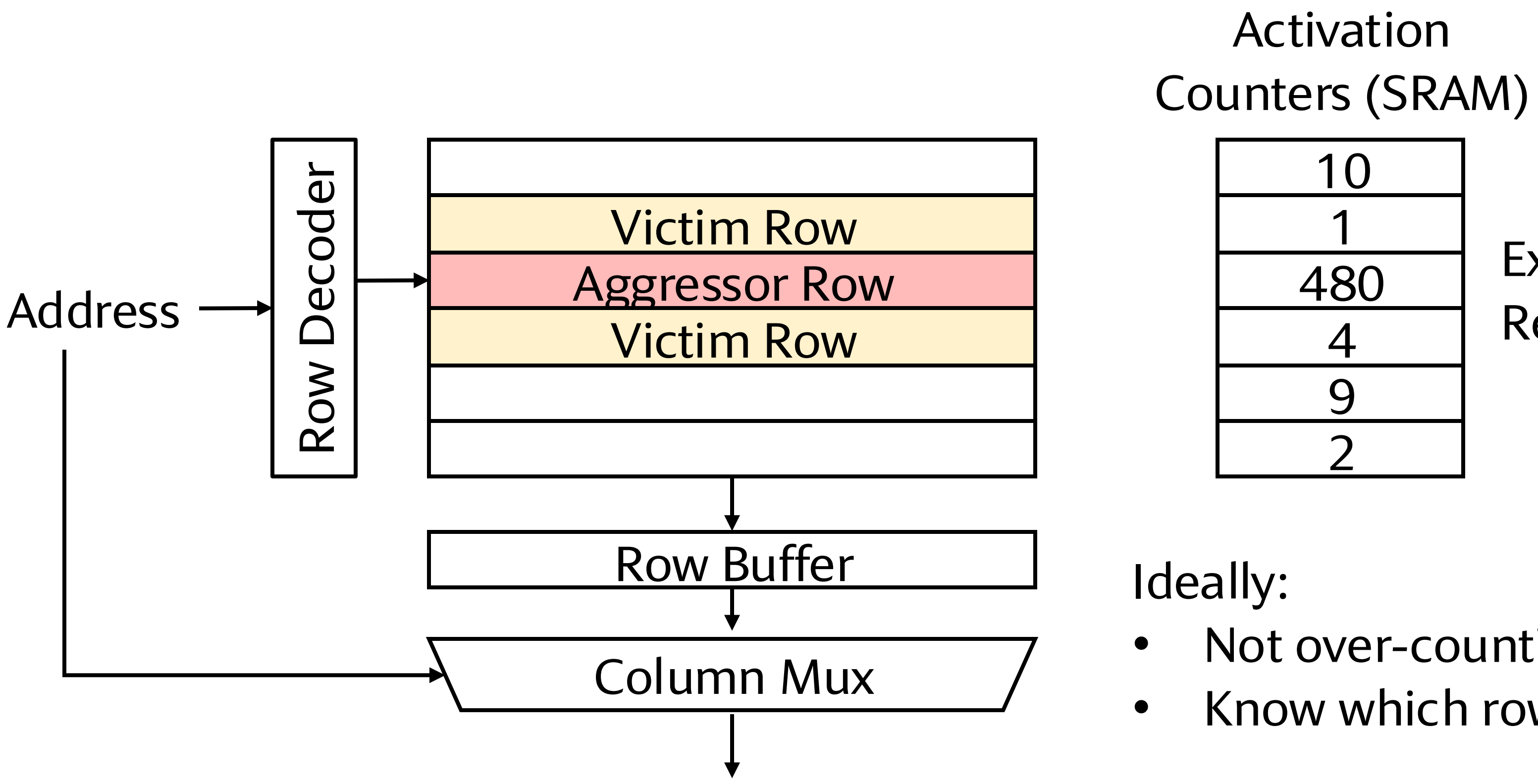


*Saxena et al., “START: Scalable Tracking for Any Rowhammer Threshold,” HPCA ’24

Counter-Based Mitigation



Counter-Based Mitigation

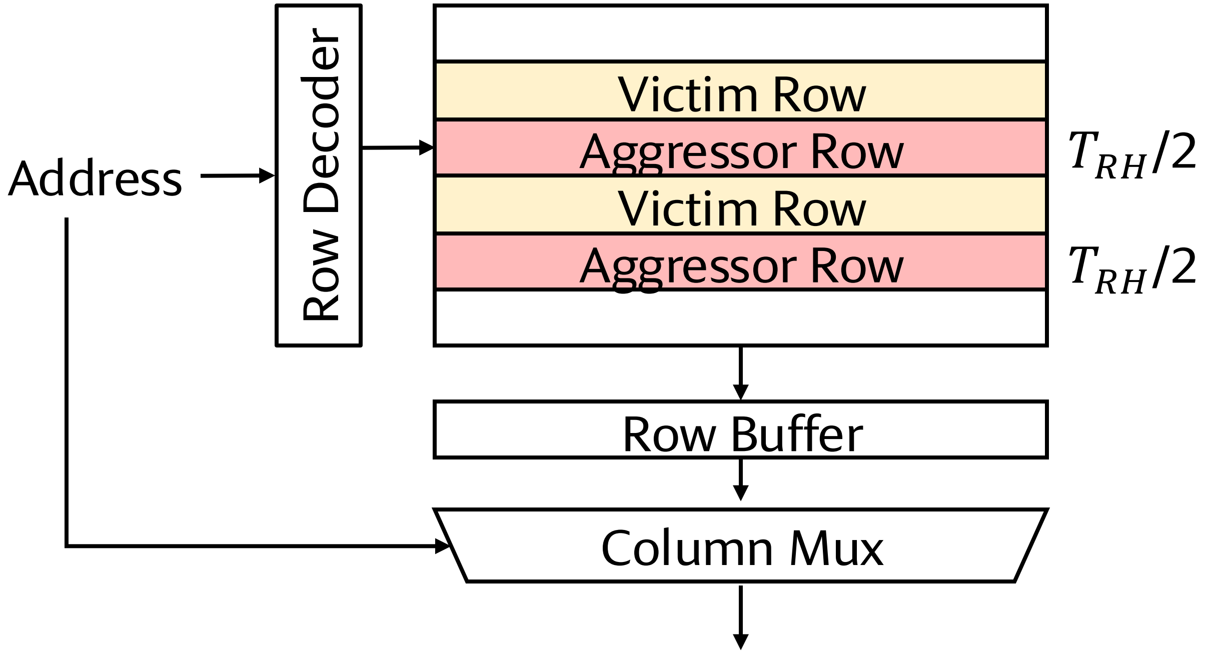


Exceeding a threshold T?
Refresh victim rows

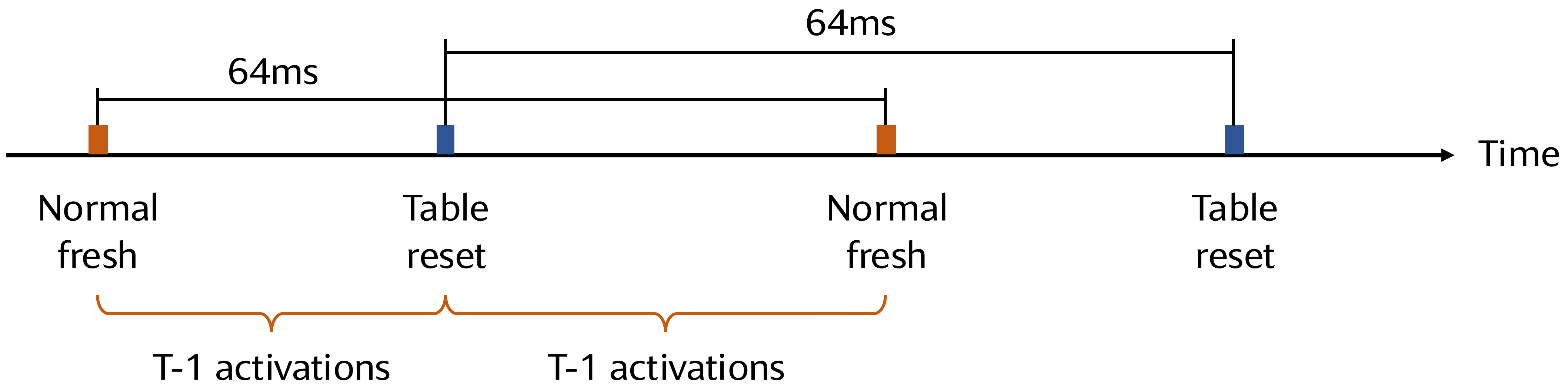
Ideally:

- Not over-counting
- Know which rows to mitigate

How to Set T - Factor 1: Double-Sided RowHammer



How to Set T - Factor 2: Refresh and Reset Timing



$$2(T-1) < \frac{T_{RH}}{2} \Rightarrow T < \frac{T_{RH}}{4} + 1$$

With $T_{RH} = 50k \Rightarrow T \approx 12.5k$

Graphene: Strong yet Lightweight Row Hammer Protection

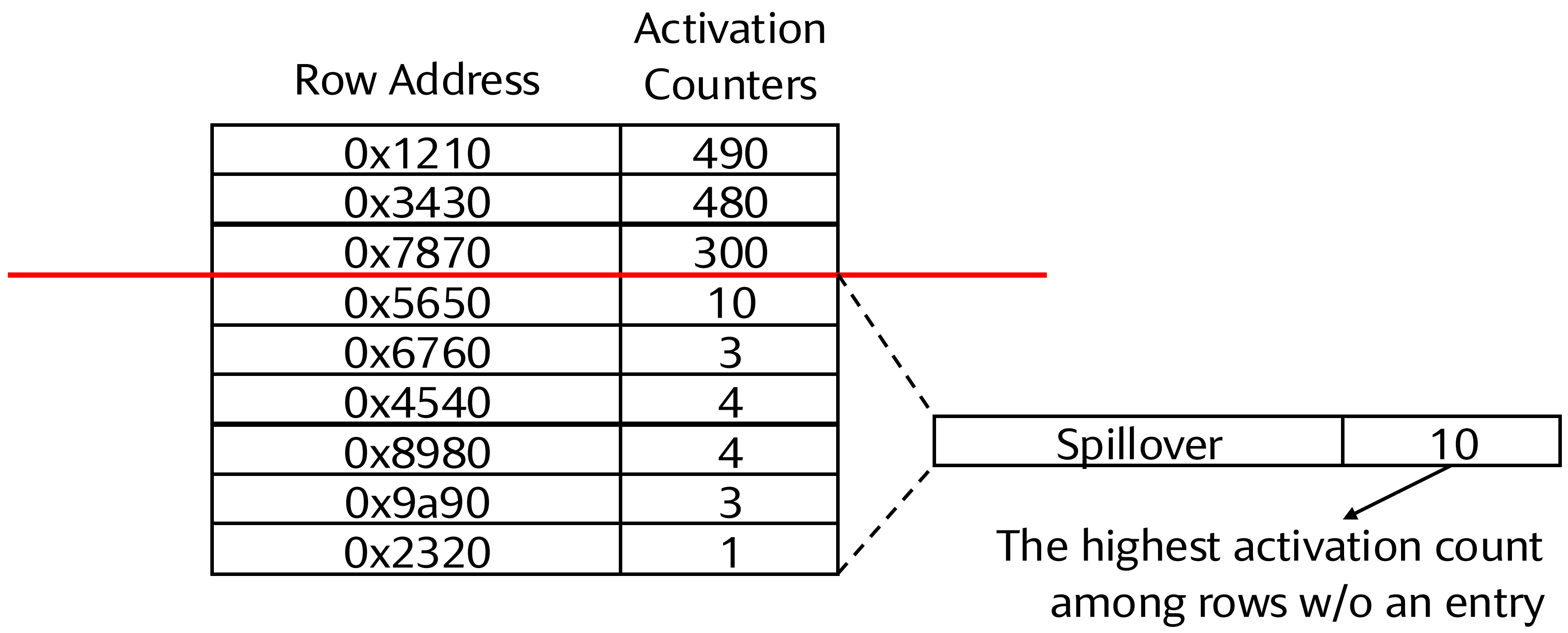
Insight: Only track the most activated rows

Row Address	Activation Counters
0x1210	490
0x2320	1
0x3430	480
0x4540	4
0x5650	10
0x6760	3
0x7870	300
0x8980	4
0x9a90	3

*Park et al., “Graphene: Strong yet Lightweight Row Hammer Protection,” MICRO ’20

Graphene: Strong yet Lightweight Row Hammer Protection

Insight: Only track the most activated rows



Row Address	Activation Counters
0x1210	490
0x3430	480
0x7870	300
0x5650	10
0x6760	3
0x4540	4
0x8980	4
0x9a90	3
0x2320	1

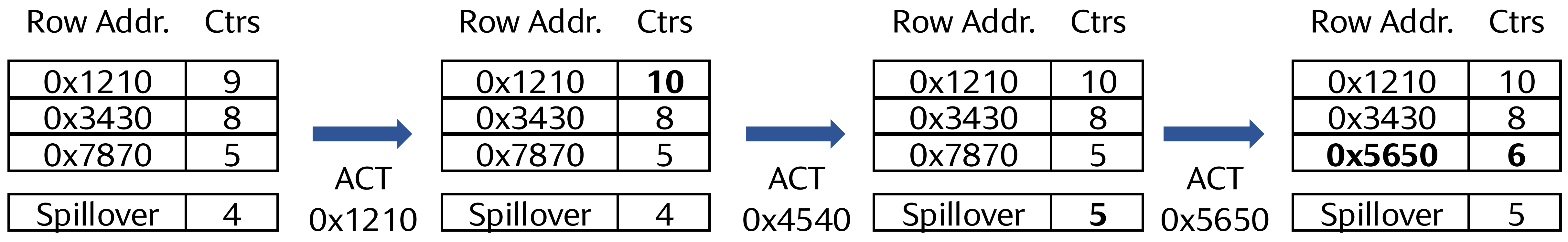
Spillover	10

The highest activation count among rows w/o an entry

[*] Park et al., “Graphene: Strong yet Lightweight Row Hammer Protection,” MICRO ’20

Graphene: Strong yet Lightweight Row Hammer Protection

Misra-Gries algorithm



Row Addr.	Ctrs
0x1210	9
0x3430	8
0x7870	5
Spillover	4

Row Addr.	Ctrs
0x1210	**10**
0x3430	8
0x7870	5
Spillover	4

Row Addr.	Ctrs
0x1210	10
0x3430	8
0x7870	5
Spillover	**5**

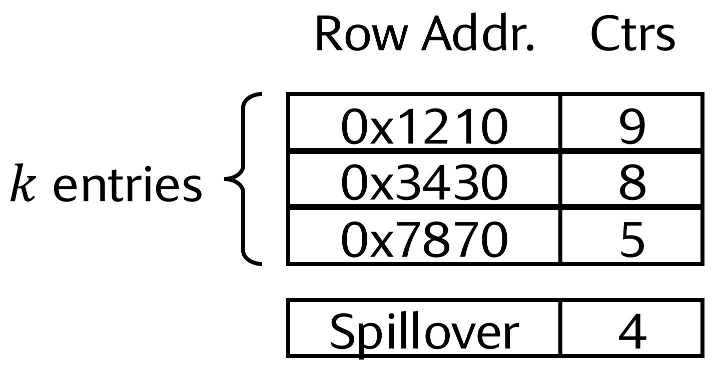
Row Addr.	Ctrs
0x1210	10
0x3430	8
0x5650	**6**
Spillover	5

Invariant: spillover counter <= smallest count from the table

Can over-count ⇒ Conservative but safe

*Park et al., "Graphene: Strong yet Lightweight Row Hammer Protection," MICRO '20

Number of Counters



Row Addr.	Ctrs
0x1210	9
0x3430	8
0x7870	5
Spillover	4

Misra-Gries algorithm guarantees:

- With k entries, after N_{act} activations:
  Row addresses with more than $N_{act}/(k+1)$ counts are tracked in the table (instead of being tracked by the spillover counter)
- We want $N_{act}/(k+1) < T \Rightarrow N_{ctr} > \frac{N_{act}}{T} - 1$

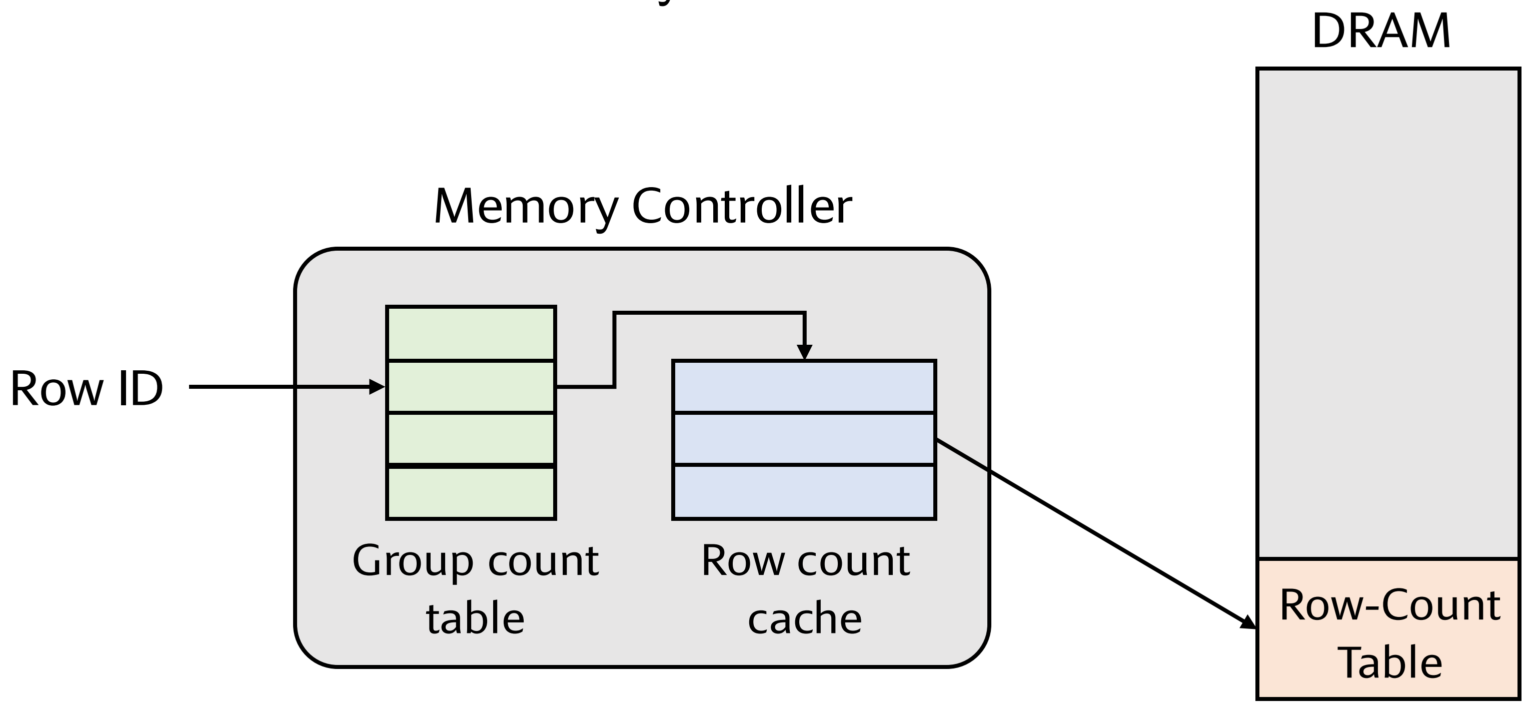
N_{act} between refreshes is $1360k$, $T = 12.5k$, $N_{ctr} = 108$

*Park et al., “Graphene: Strong yet Lightweight Row Hammer Protection,” MICRO ’20

Hydra: Hybrid Tracking

The maximum number of activations between refreshes is bounded:

- Access many rows few times
- Access few rows many times



*Qureshi et al., "Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking," ISCA '22