

Rowhammer

Noah Brown (some slides adapted from Andrew Kwong)

Intro

- Rowhammer is a phenomenon that occurs due to electrical interference at the *hardware* level
 - It causes the values of bits to flip (from 0 to 1, or vice-versa)
 - It is *highly* reproducible once found, meaning attackers can exploit it
 - The underlying interference is a *byproduct* of how DRAM works; it cannot be avoided

Outline

- Memory crash course
- What is Rowhammer?
- How do we execute a Rowhammer attack?
- What can we do with Rowhammer?
- How can we protect against Rowhammer?

Memory

A large, empty rectangular box with a black border, positioned below the 'Memory' text. It appears to be a placeholder for content or a diagram.

Memory



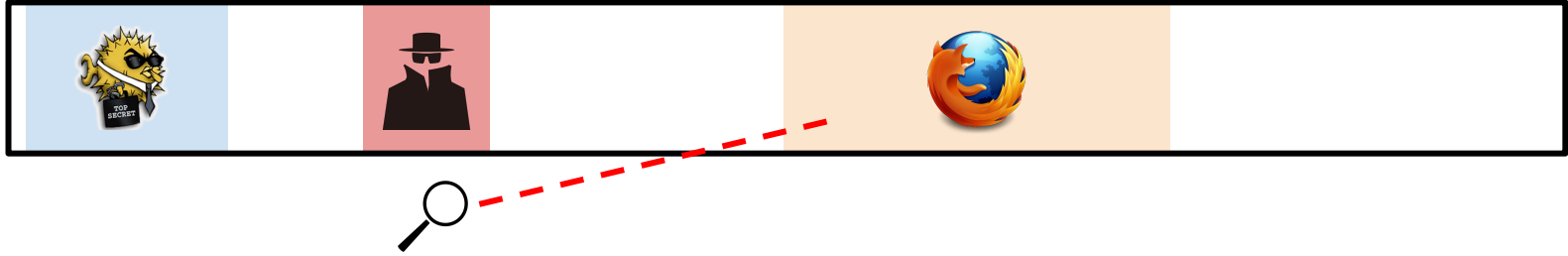
Memory



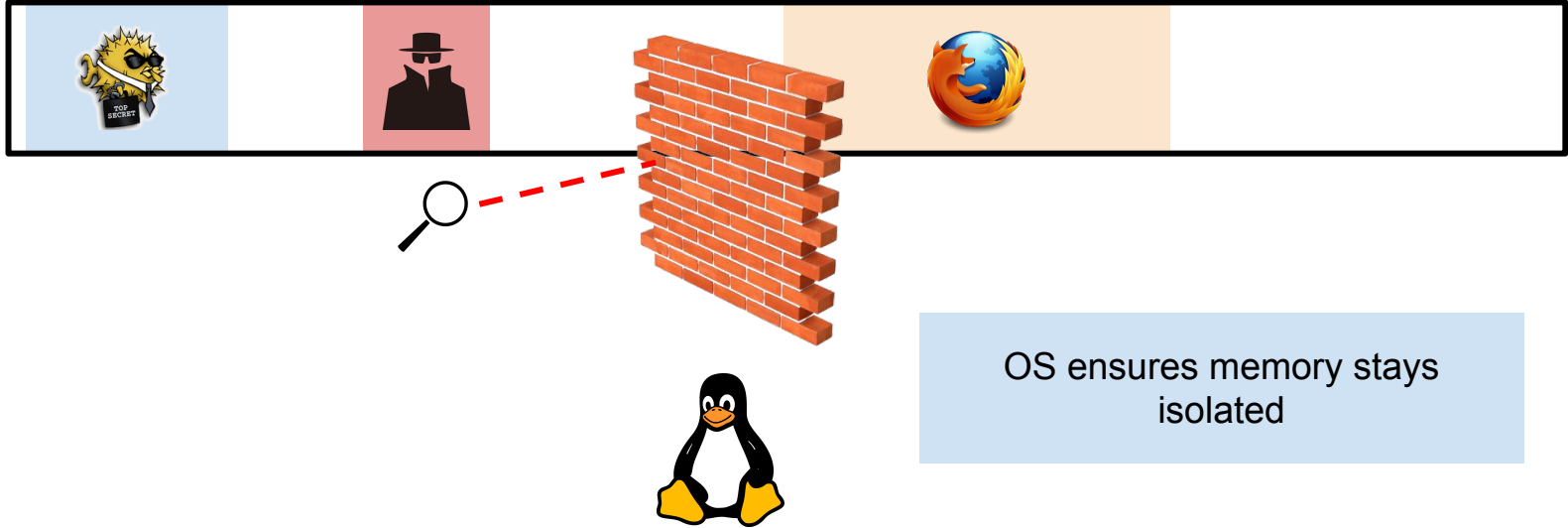
Memory



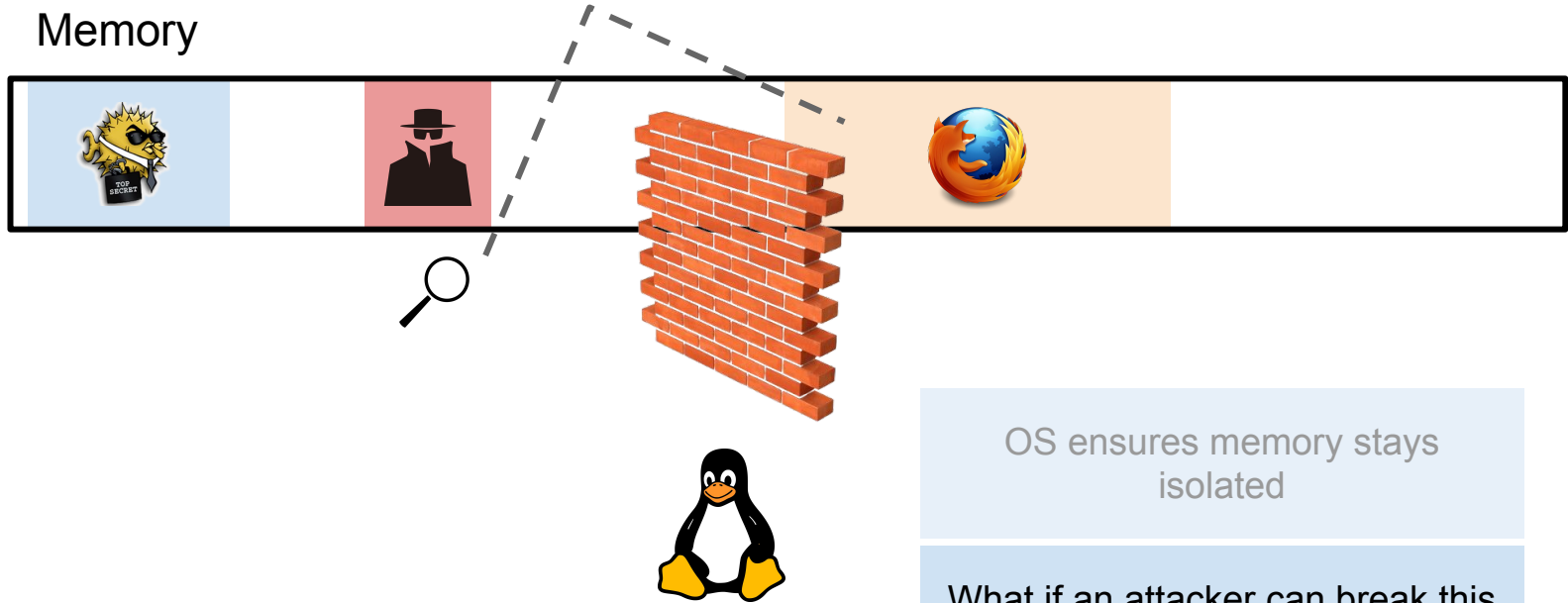
Memory



Memory



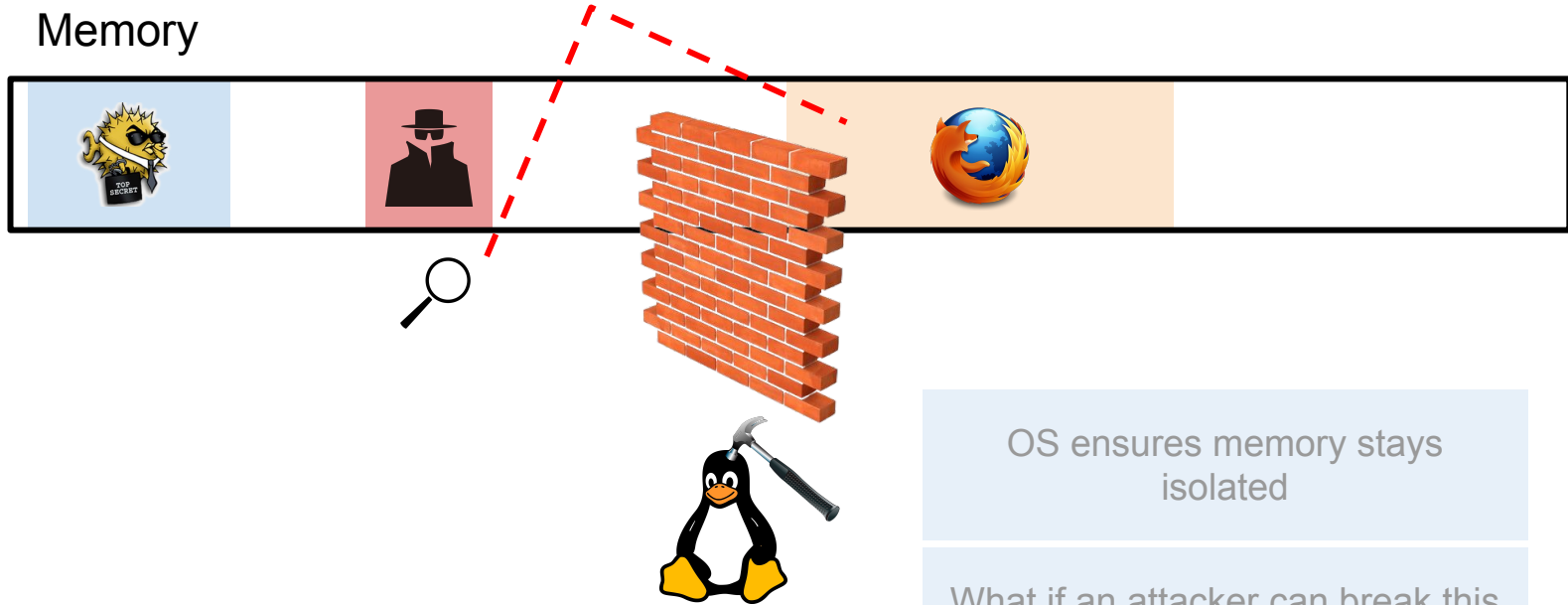
Memory



OS ensures memory stays
isolated

What if an attacker can break this
isolation?

Memory



OS ensures memory stays isolated

What if an attacker can break this isolation?

With Rowhammer, an attacker can!

DRAM Organization

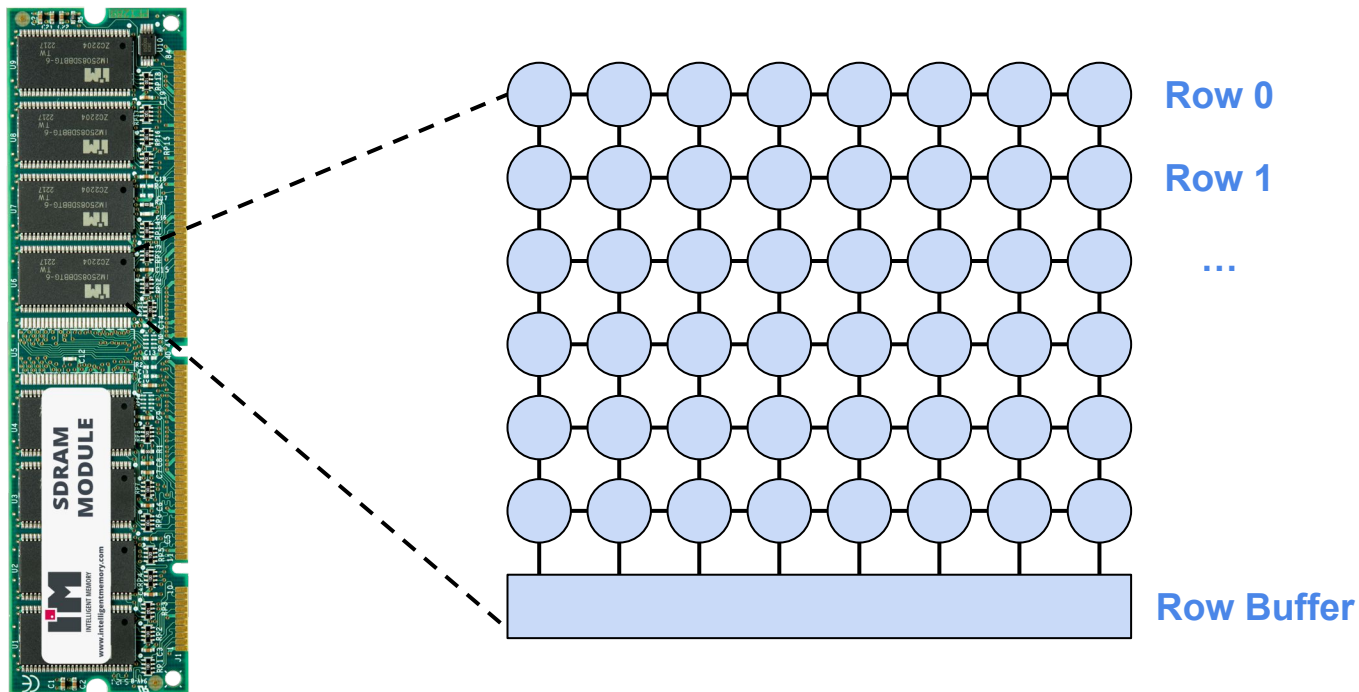
Bank

Bank

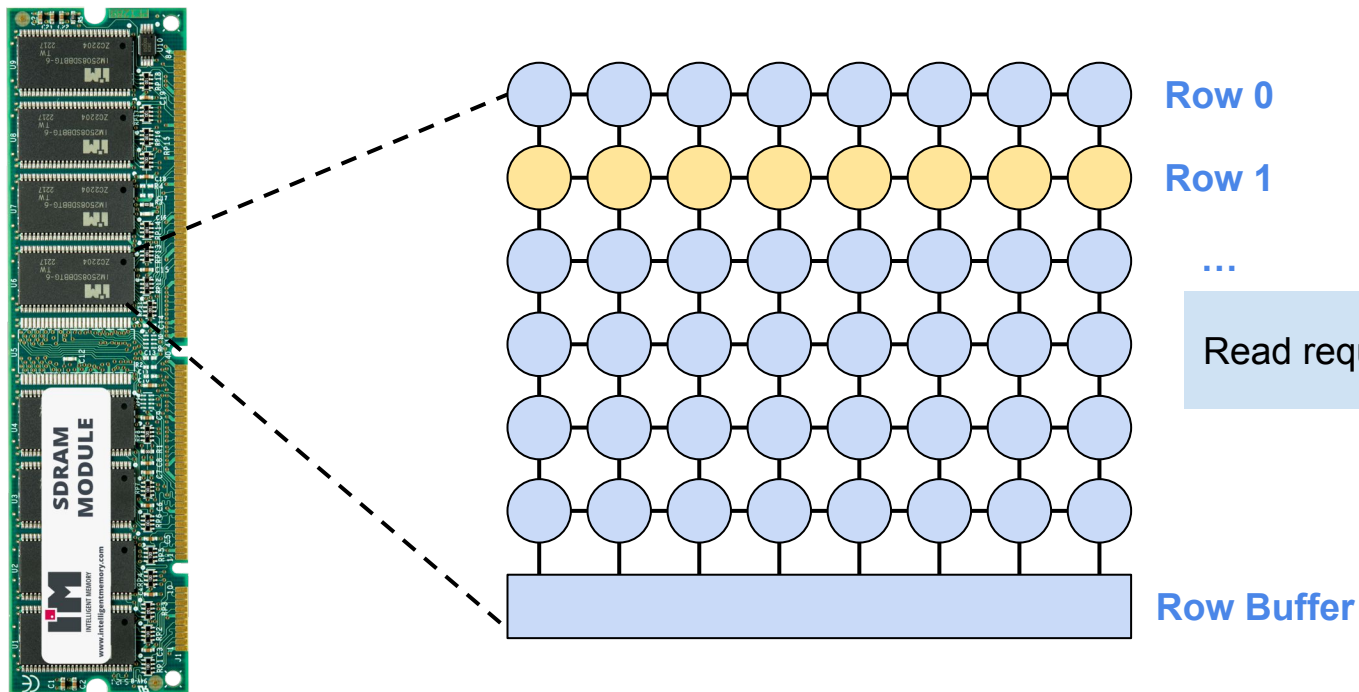
Bank



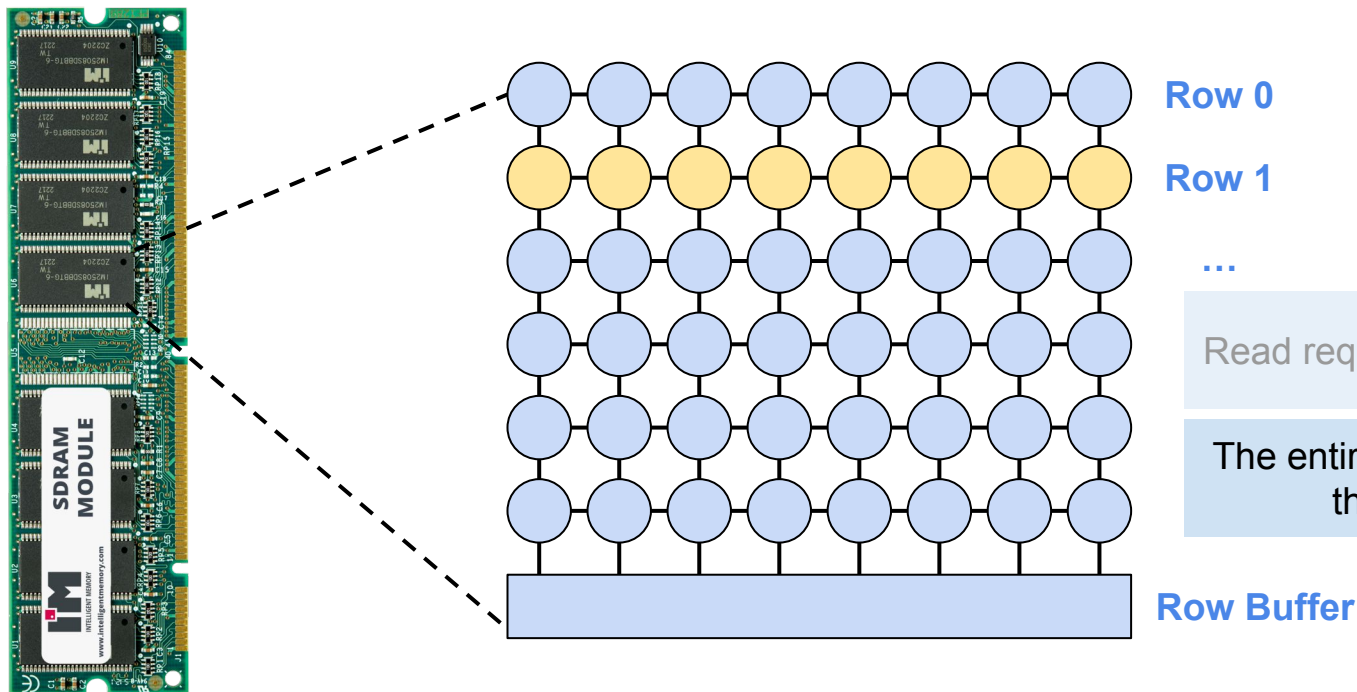
DRAM Organization



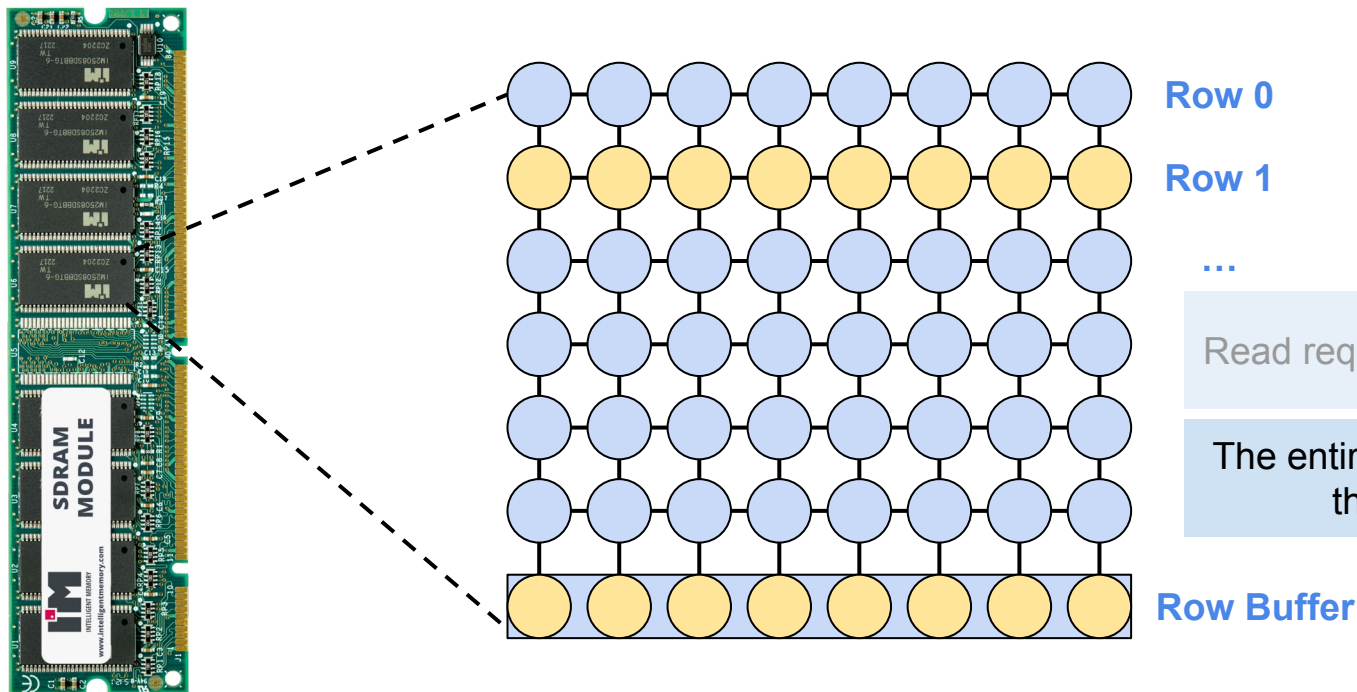
DRAM Organization



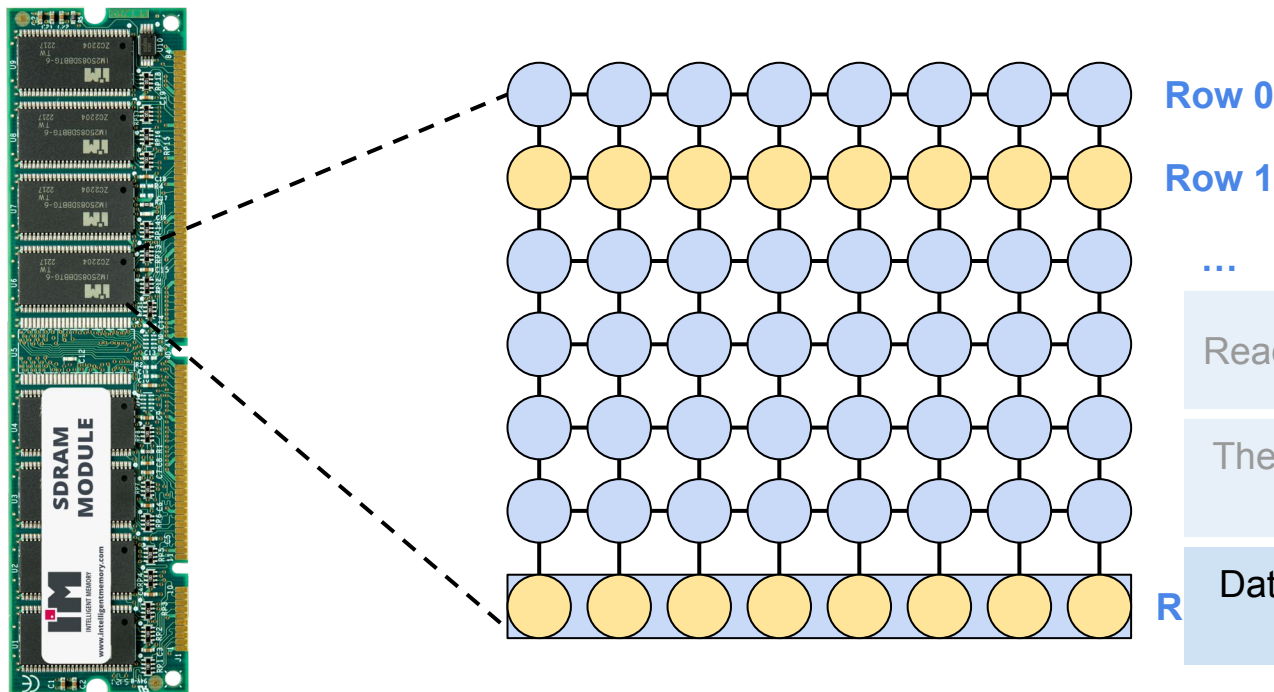
DRAM Organization



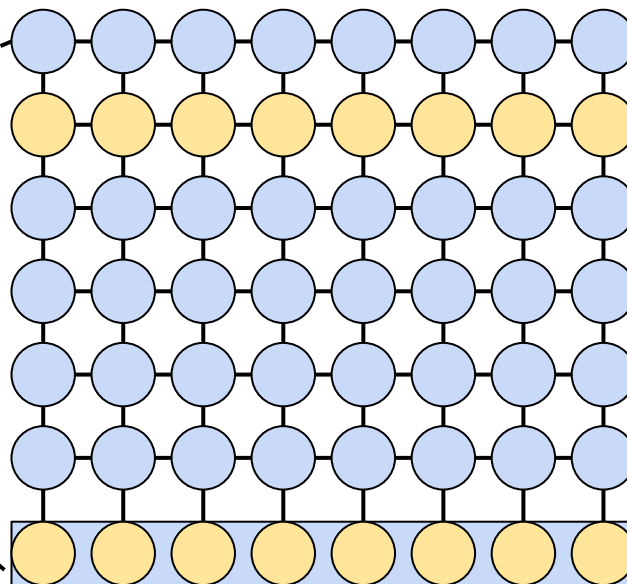
DRAM Organization



DRAM Organization



DRAM Organization



Row 0

Row 1

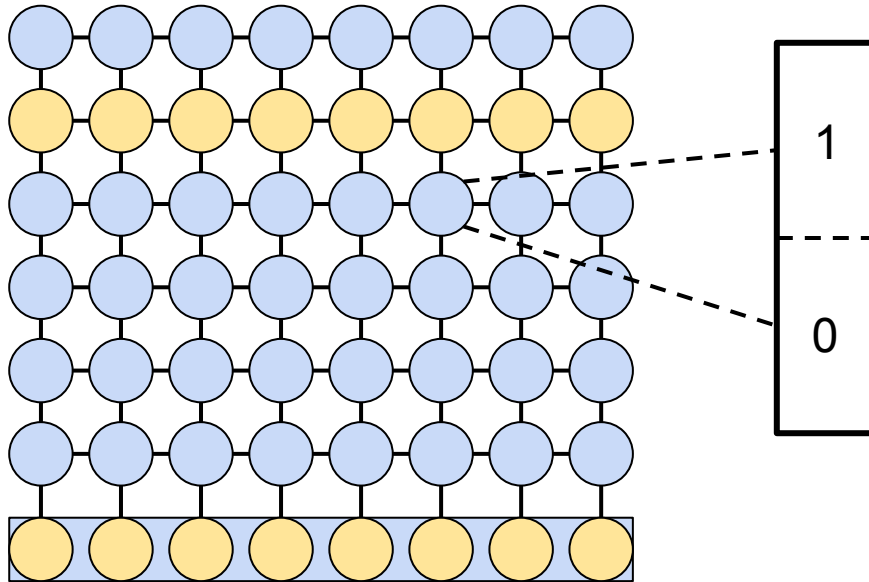
...

Read request is sent to a row

The entire row is copied into the row buffer

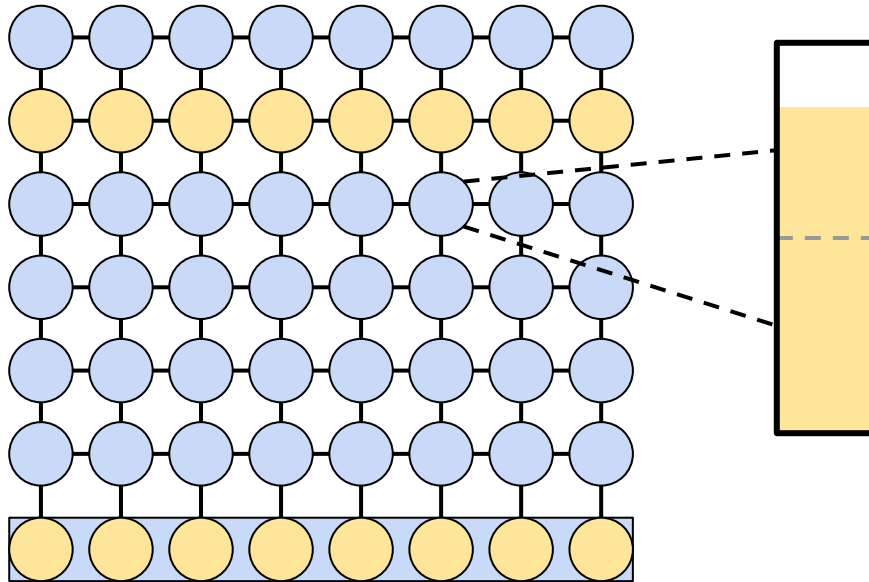
R Row buffer acts as a cache: until a new row is activated, requests are fulfilled directly from the row buffer

Rowhammer



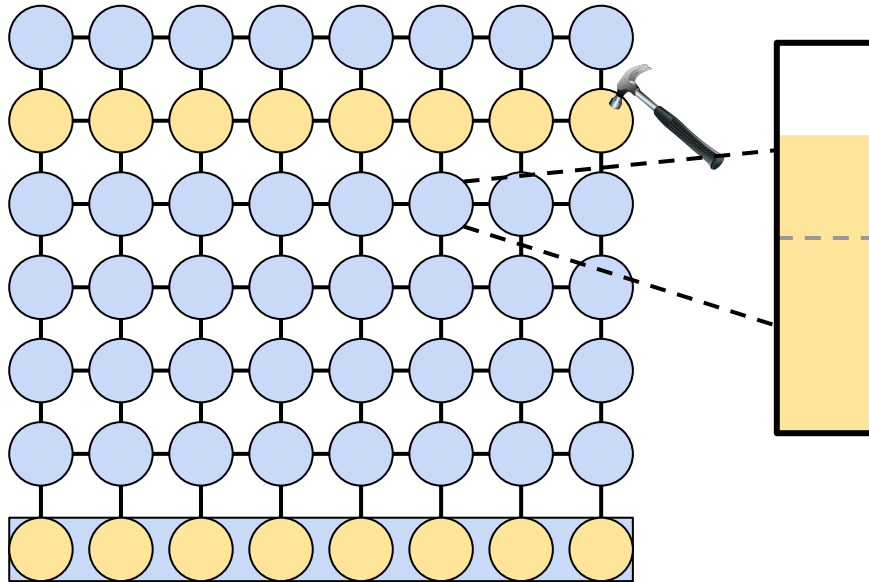
The value of a cell (bit) is stored as charge or no charge in cell

Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

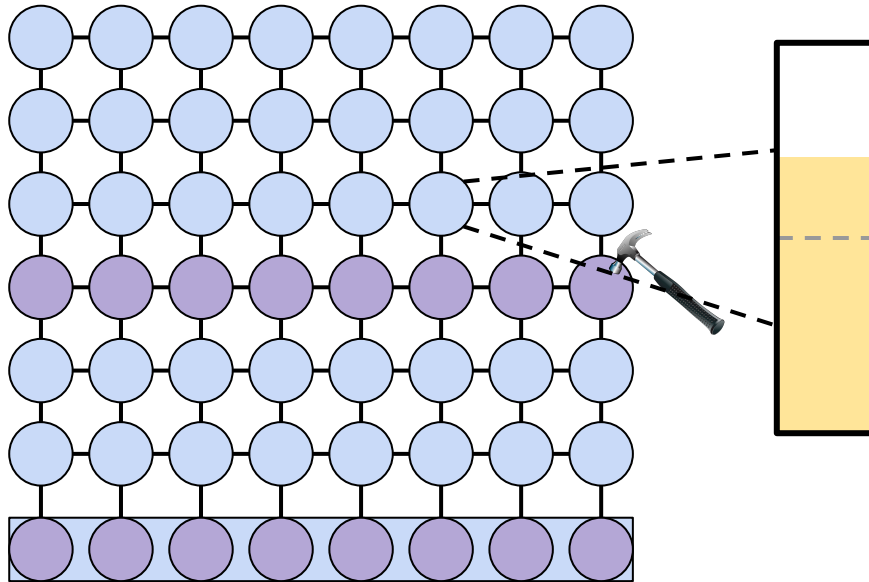
Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

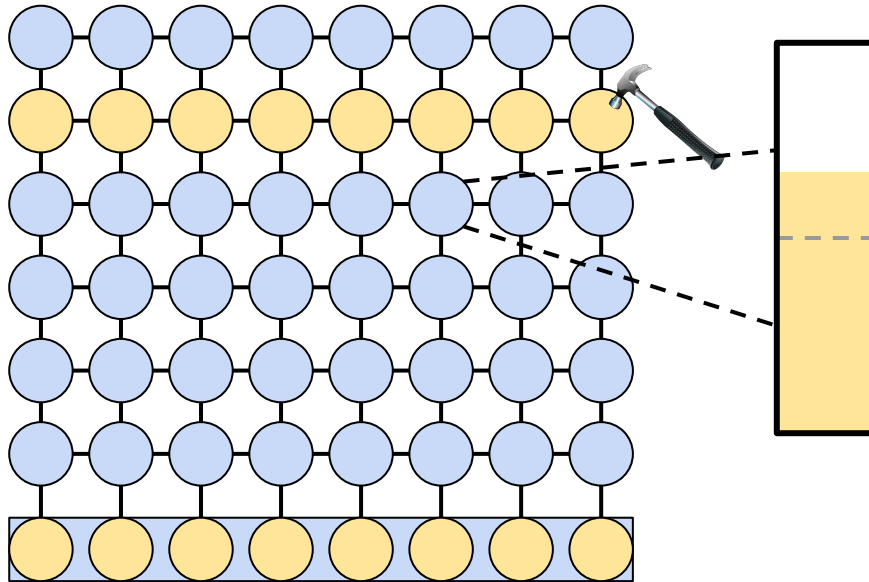
Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

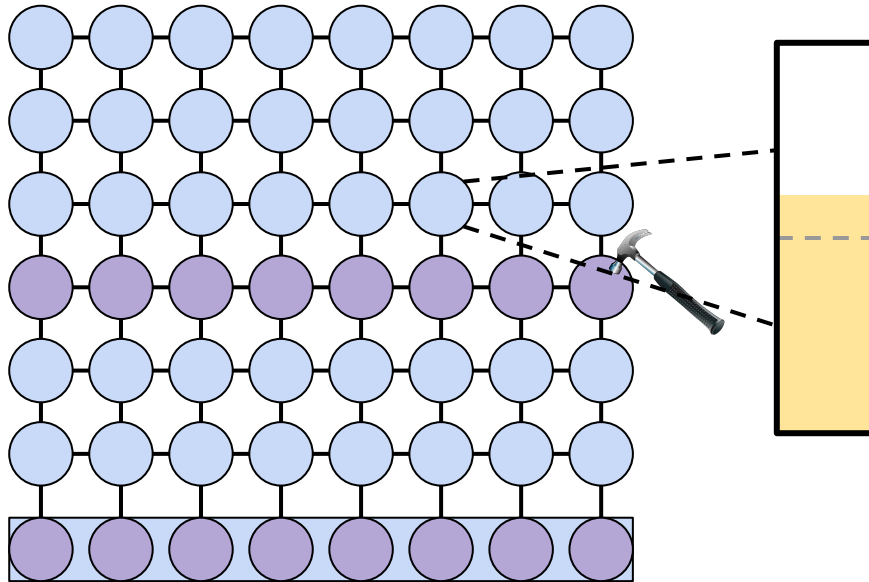
Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

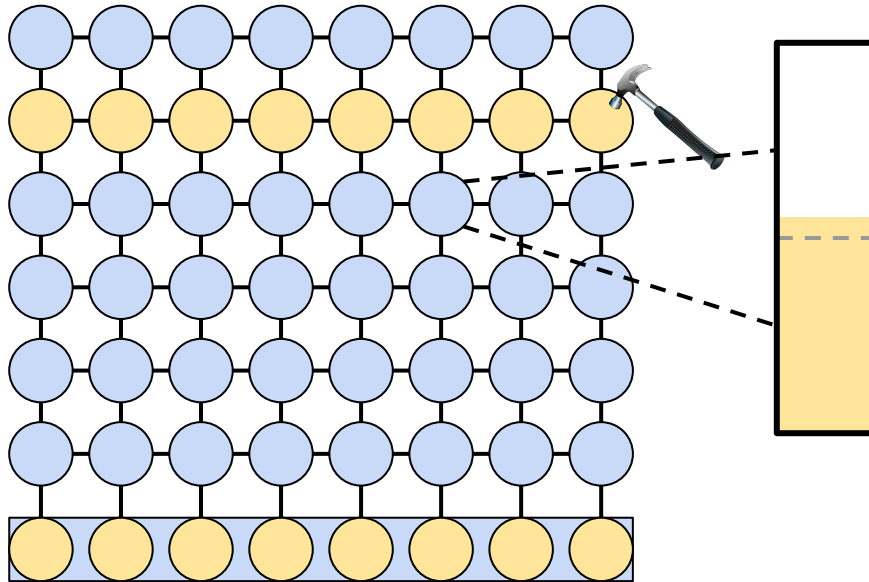
Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

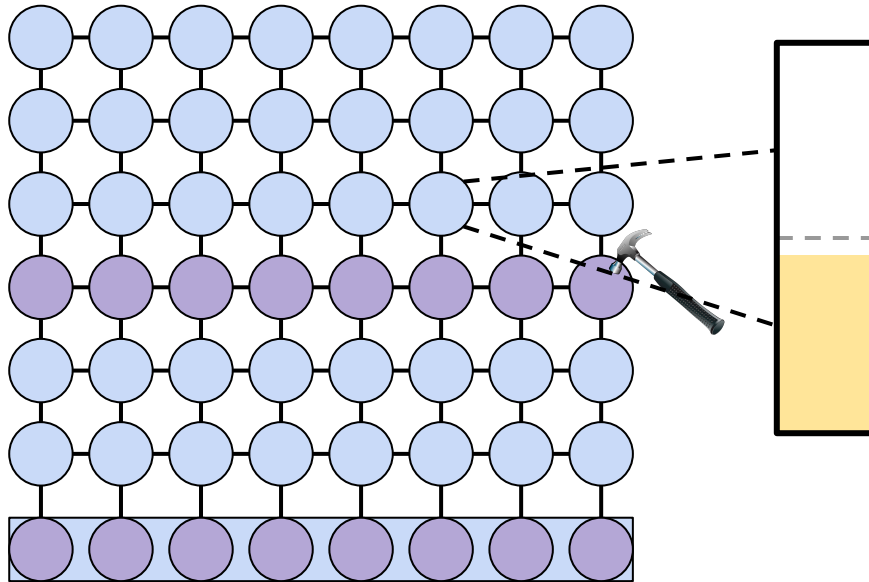
Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

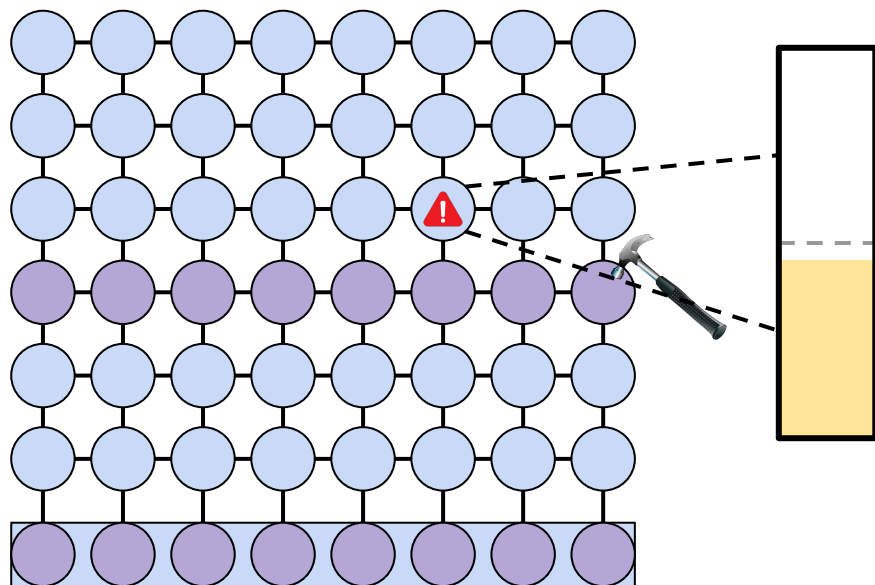
Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

Rowhammer

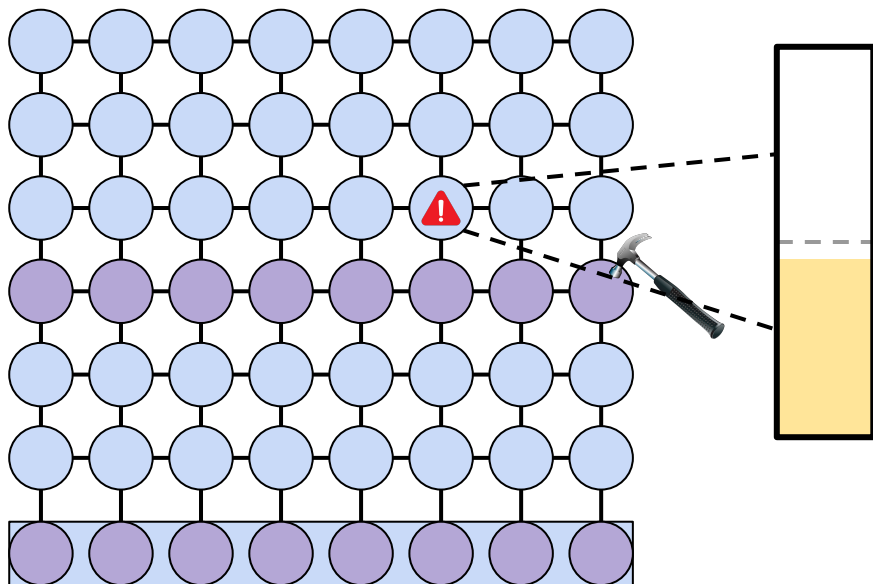


The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

If enough charge leaks, a bit can flip! (Kim et al., SIGARCH '14)

Rowhammer



The value of a cell (bit) is stored as charge or no charge in cell

Reading a row causes charge to leak out of adjacent cells

If enough charge leaks, a bit can flip! (Kim et al., SIGARCH '14)

An attacker controlling two rows can write to the row between, **EVEN IF** someone else's memory is there

Naive Protection

- Even before Rowhammer, this charge-leakage issue was known
- Thus, a *refresh* is issued periodically to all rows (~64ms)
 - A refresh restores charge back to its full value (or lack)
- Rowhammer accelerates the charge leakage **beyond** what these periodic refreshes can identify
 - In other words, if you Rowhammer a 1 to a 0 before a refresh happens, the bit is permanently flipped

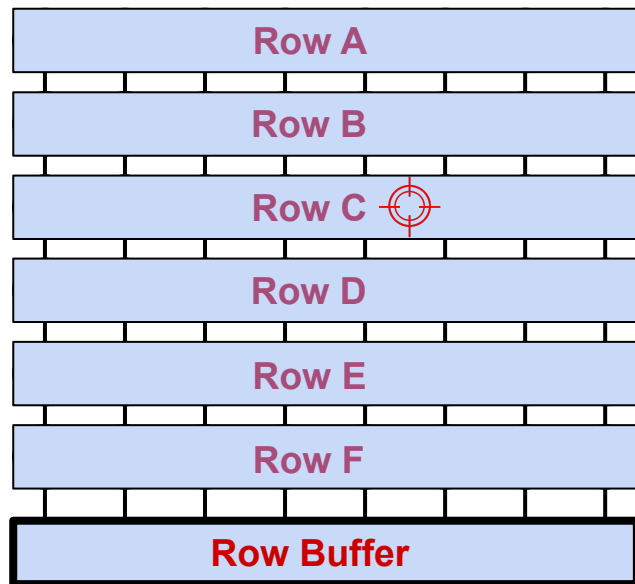
How do we Rowhammer?

- There are some challenges involved:
 - How do we even hammer? We need to make sure we are repeatedly accessing rows in memory
 - We don't actually know which addresses map to which rows. How can we figure it out?
 - How do we map victim data to vulnerable cells?

How do we Rowhammer?

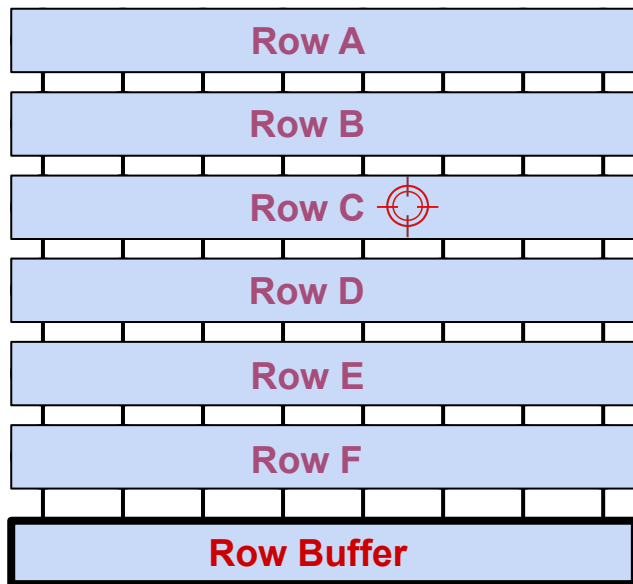
- There are some challenges involved:
 - How do we even hammer? We need to make sure we are repeatedly accessing rows in memory
 - We don't actually know which addresses map to which rows. How can we figure it out?
 - ~~How do we map victim data to vulnerable cells?~~

Rowhammer



We want to flip a bit in Row C

Rowhammer

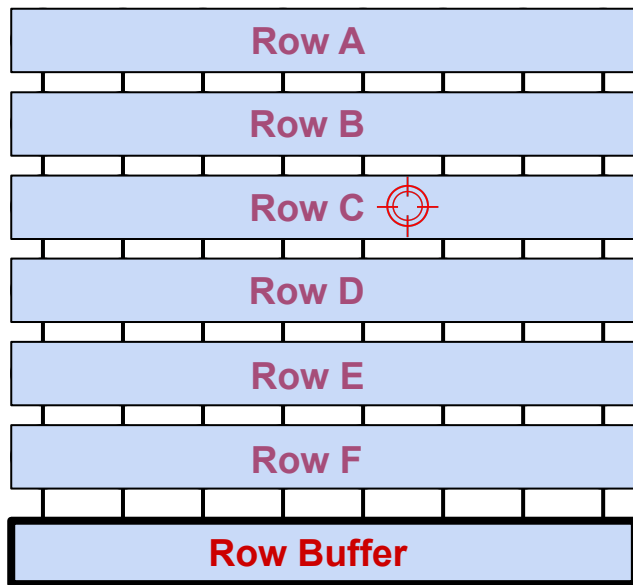


```
loop:
  mov (Row B), %eax
  jmp loop
```

We want to flip a bit in Row C

We can't just access Row B over and over

Rowhammer



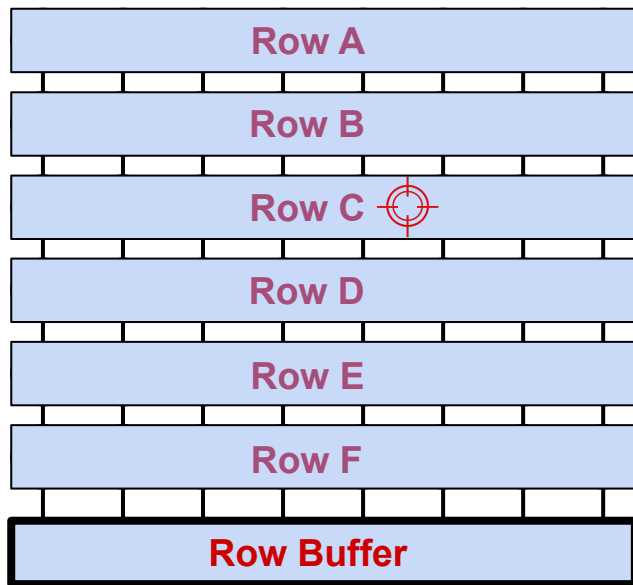
```
loop:  
  mov (Row B), %eax  
  jmp loop
```

We want to flip a bit in Row C

We can't just access Row B over and over

Why won't this hit Row B **in memory** repeatedly?
(Hint: remember the multi-level memory hierarchy)

Rowhammer



```
loop:  
  mov (Row B), %eax  
  jmp loop
```

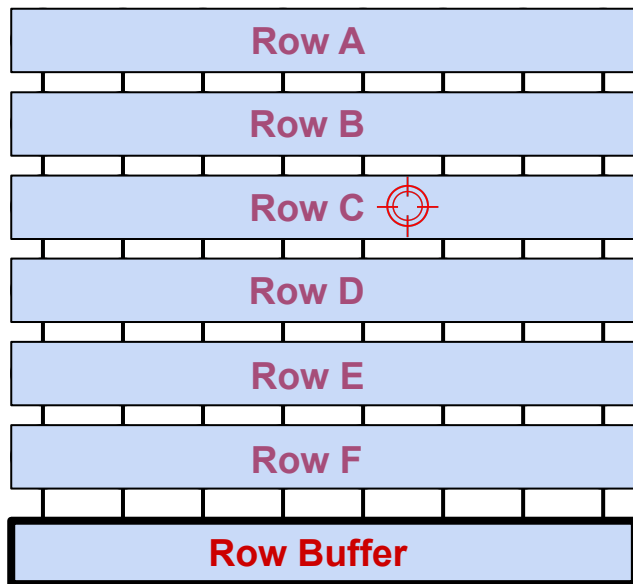
We want to flip a bit in Row C

We can't just access Row B over and over

Why won't this hit Row B **in memory** repeatedly?
(Hint: remember the multi-level memory hierarchy)

We'd hit the cache over and over.

Rowhammer



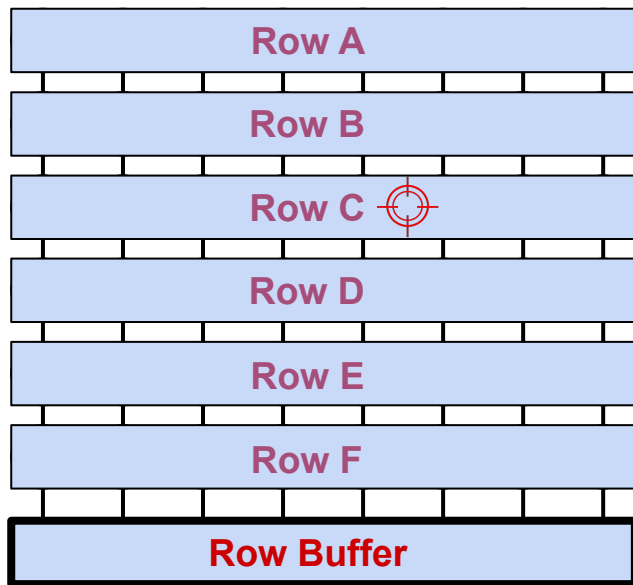
```
loop:
  mov (Row B), %eax
  clflush(Row B)
  jmp loop
```

We want to flip a bit in Row C

We can't just access Row B over and over

Even with flushing, this won't activate Row B over and over

Rowhammer



```
loop:
  mov (Row B), %eax
  clflush(Row B)
  jmp loop
```

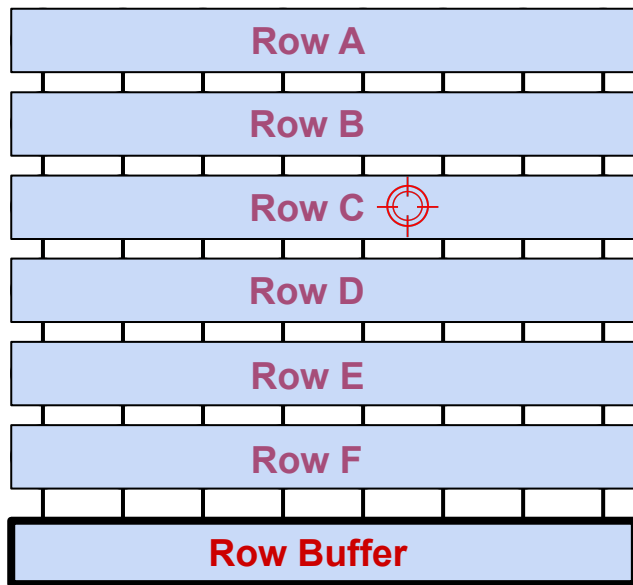
We want to flip a bit in Row C

We can't just access Row B over and over

Even with flushing, this won't activate Row B over and over

Why not?
(Hint: remember there's a cache of sorts in the bank)

Rowhammer



```
loop:
  mov (Row B), %eax
  clflush(Row B)
  jmp loop
```

We want to flip a bit in Row C

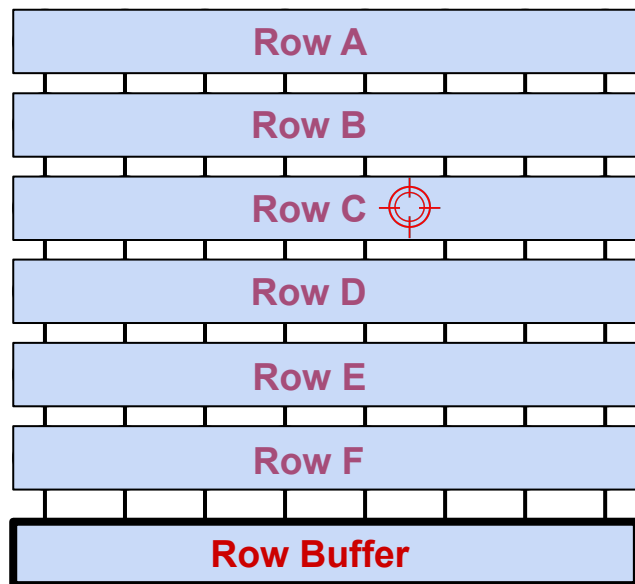
We can't just access Row B over and over

Even with flushing, this won't activate Row B over and over

Why not?
(Hint: remember there's a cache of sorts in the bank)

We'd hit the row buffer over and over.

Rowhammer



```
loop:
  mov (Row B), %eax
  mov (Row D), %eax
  clflush(Row B)
  clflush(Row D)
  jmp loop
```

We want to flip a bit in Row C

We can't just access Row B over and over

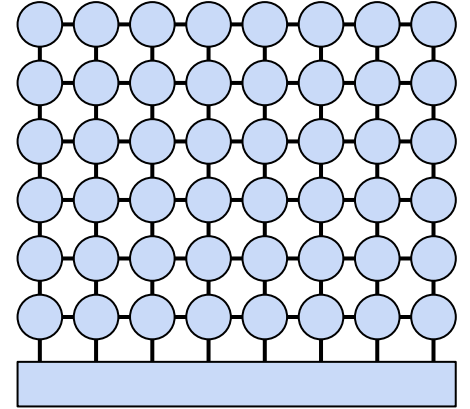
Even with flushing, this won't activate Row B over and over

Alternating Rows B and D bypasses the Row Buffer every time

How do we find rows in the same bank?

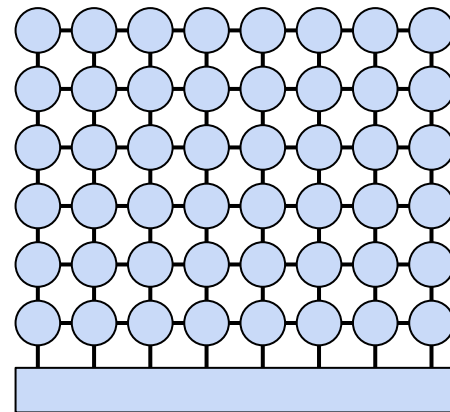
- So now we know how to Rowhammer properly
- But in the real world, we can't just access rows
 - We only see *physical addresses*, that are mapped to rows somehow

DRAM Address Reverse Engineering



DRAM Address Reverse Engineering

Page tables translate virtual addresses to physical ones

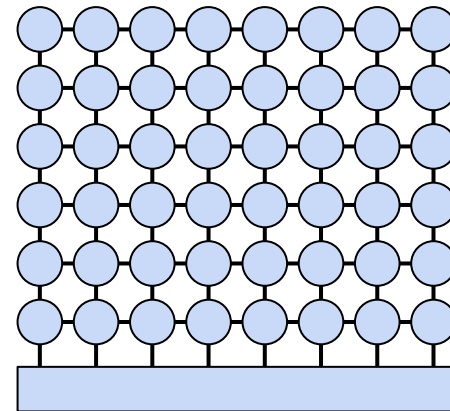


DRAM Address Reverse Engineering

Page tables translate virtual addresses to physical ones



Physical addresses are translated to DRAM locations via **function mappings**

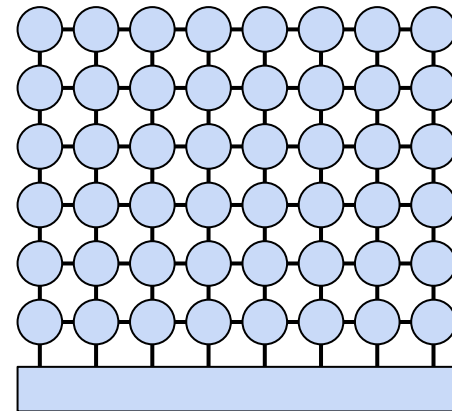


DRAM Address Reverse Engineering

Page tables translate virtual addresses to physical ones



We need to figure out these functions ourselves!

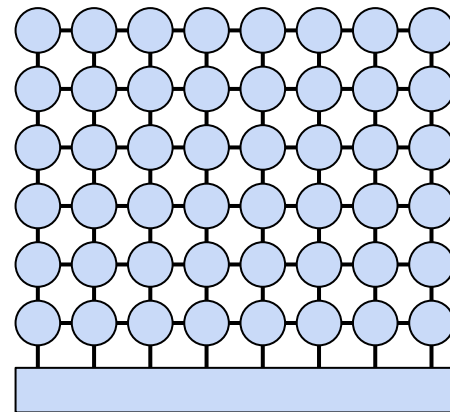


DRAM Address Reverse Engineering

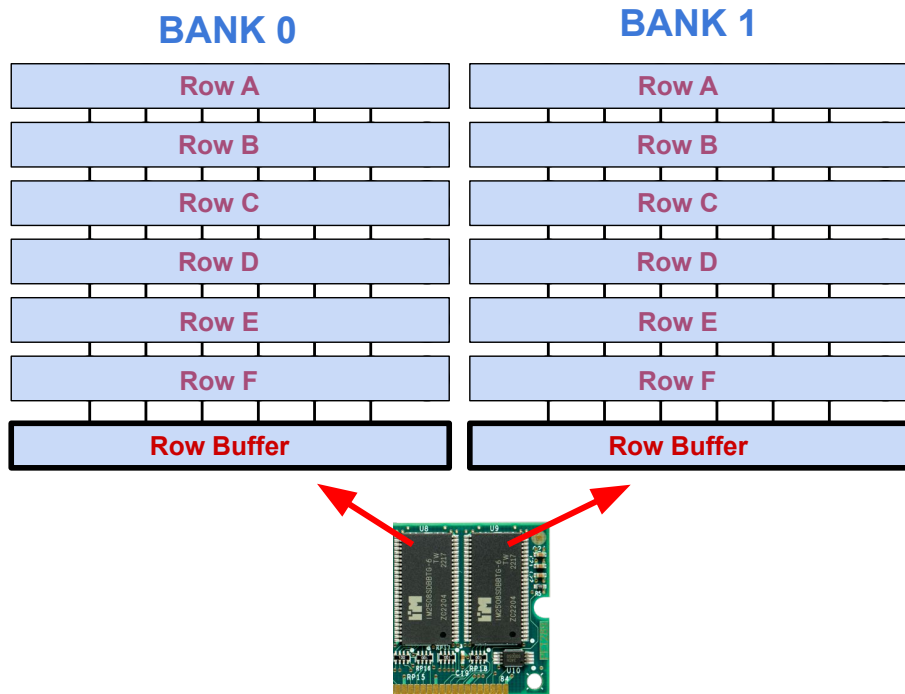
Page tables translate virtual addresses to physical ones



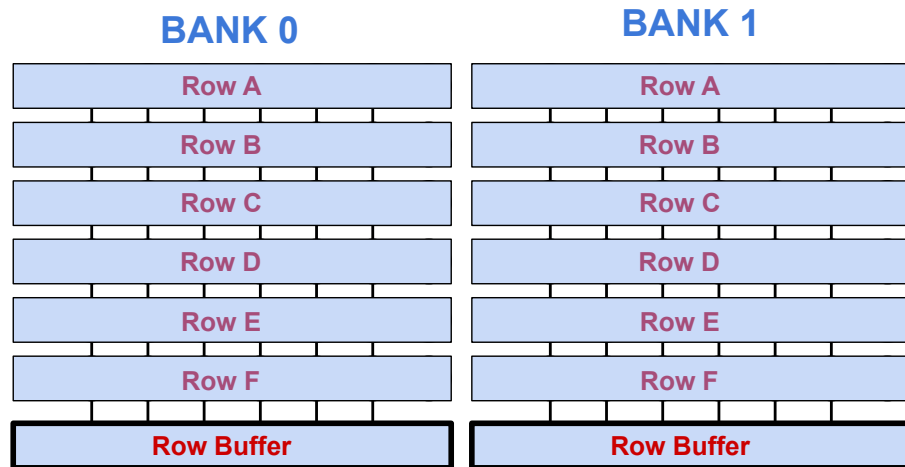
We use a *side-channel* to figure out what bank/row every address maps to



Side-Channel Example



Side-Channel Example



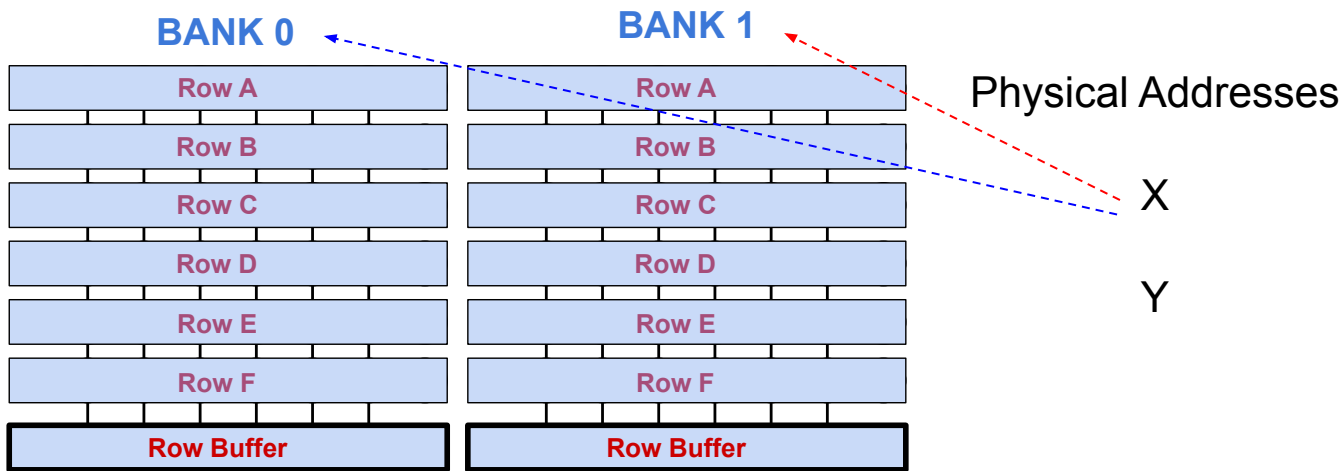
Physical Addresses

X

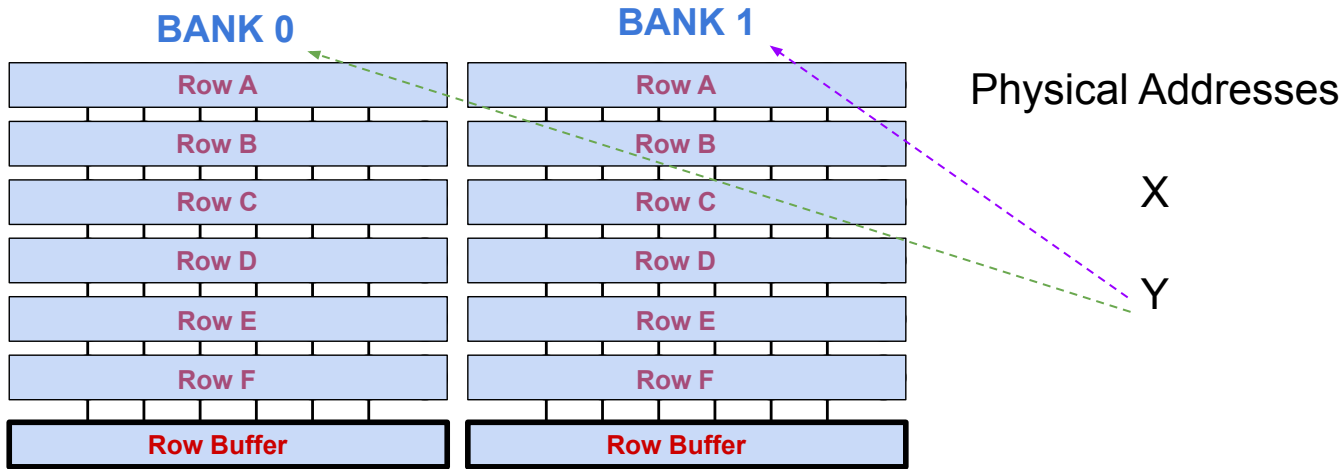
Y

We only see *physical addresses* X/Y to start—we don't know which bank they are a part of

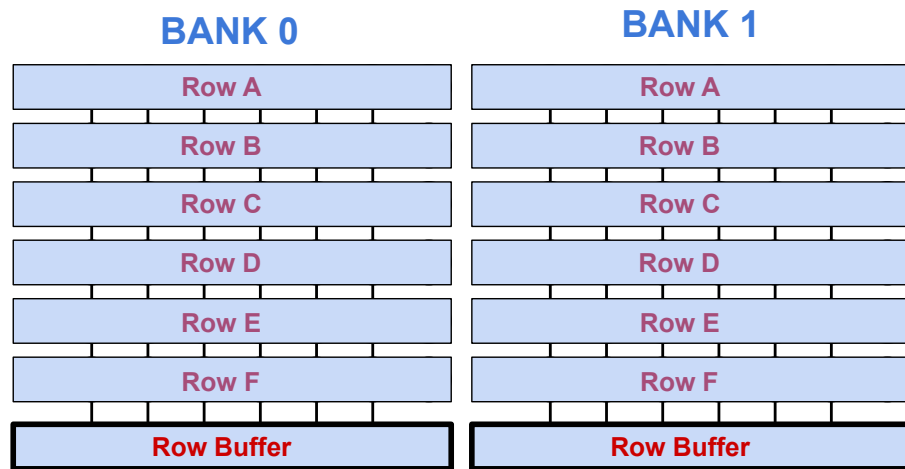
Side-Channel Example



Side-Channel Example



Side-Channel Example



X - ?

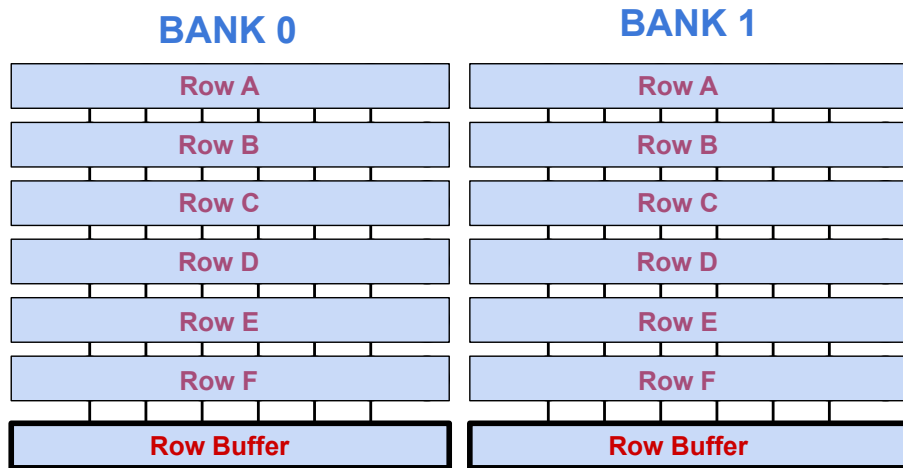
Y - ?

```
loop:
    mov (X), %eax
    mov (Y), %eax
    clflush(X)
    clflush(Y)
    jmp loop
```

Which of the following options do you think is correct? (hint: remember the row buffer!)

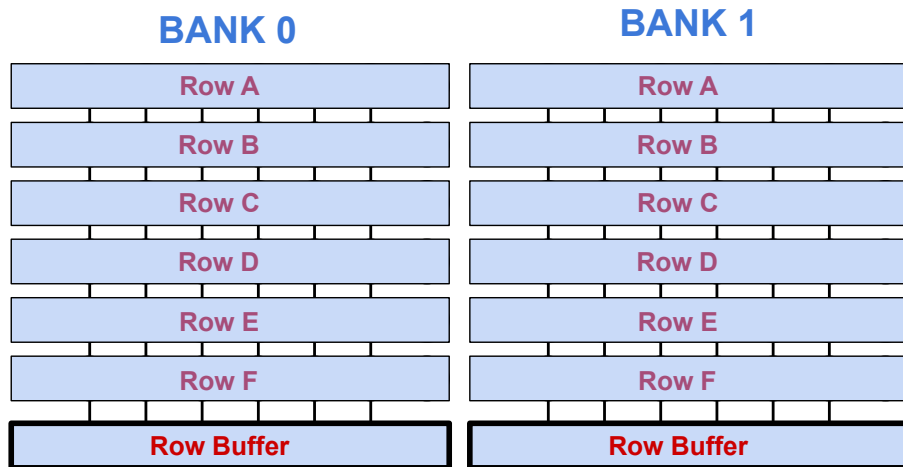
If X and Y are rows in *different* banks, alternating accesses to X and Y should be **(FASTER/SLOWER)** than if X and Y were in the *same* bank

Side-Channel Example



We can access rows in *different* banks very quickly

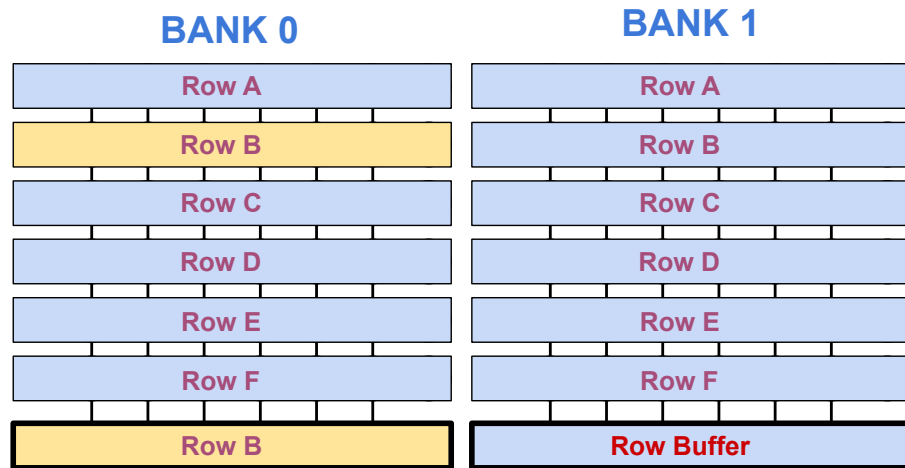
Side-Channel Example



We can access rows in *different* banks very quickly

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 1, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 1, Row E)
    jmp loop
```

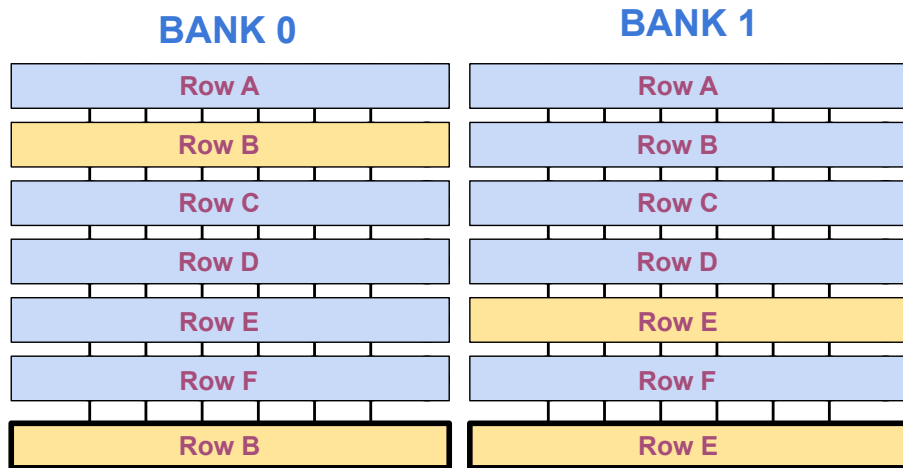
Side-Channel Example



We can access rows in *different* banks very quickly

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 1, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 1, Row E)
    jmp loop
```

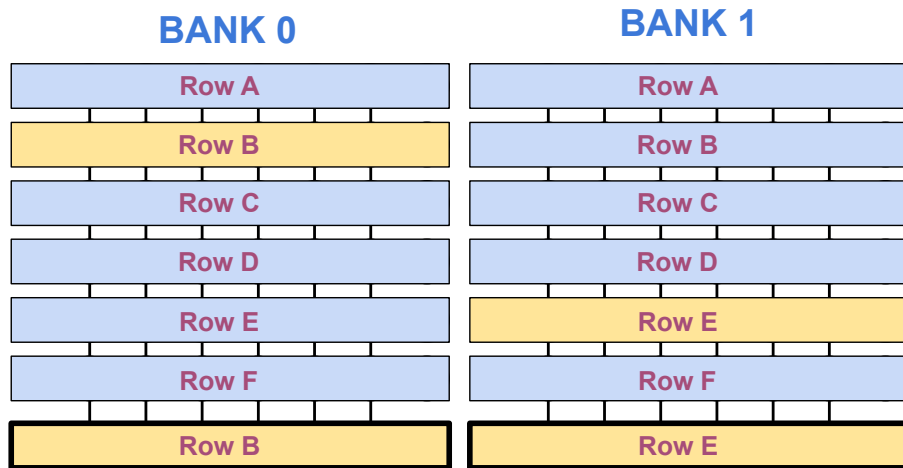
Side-Channel Example



We can access rows in *different* banks very quickly

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 1, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 1, Row E)
    jmp loop
```

Side-Channel Example

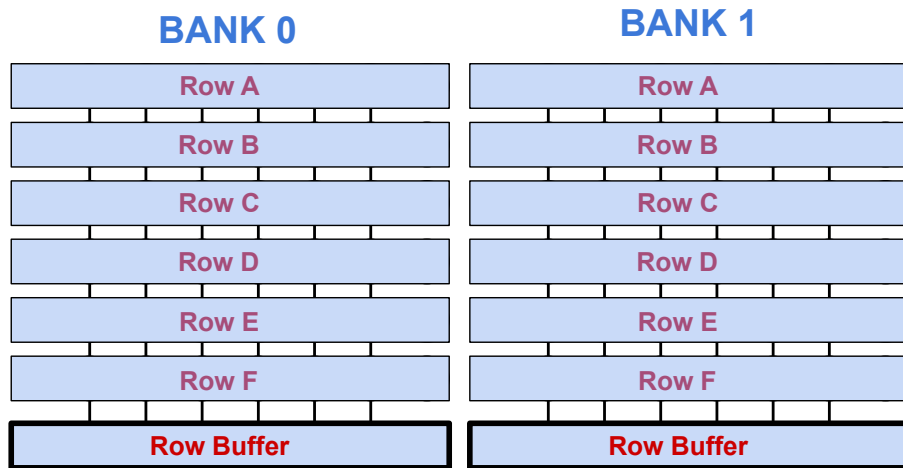


We can access rows in *different* banks very quickly

...because we keep hitting the row buffer!

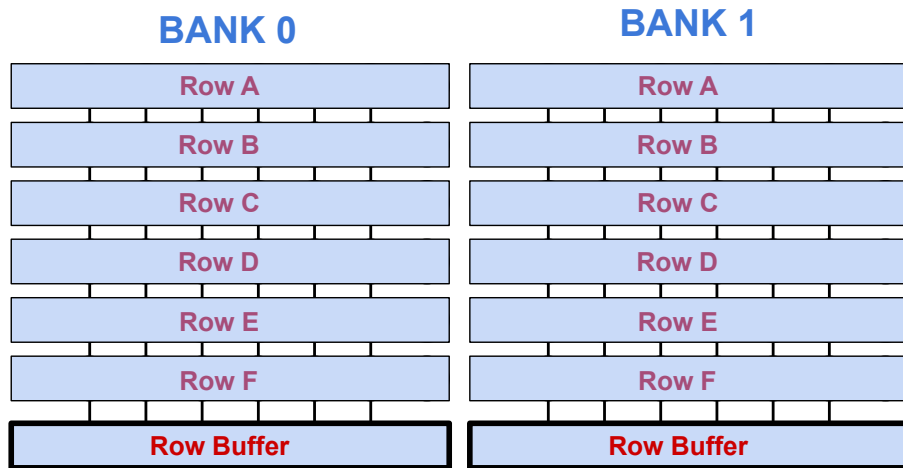
```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 1, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 1, Row E)
    jmp loop
```

Side-Channel Example



Accessing rows in the *same* bank is very slow

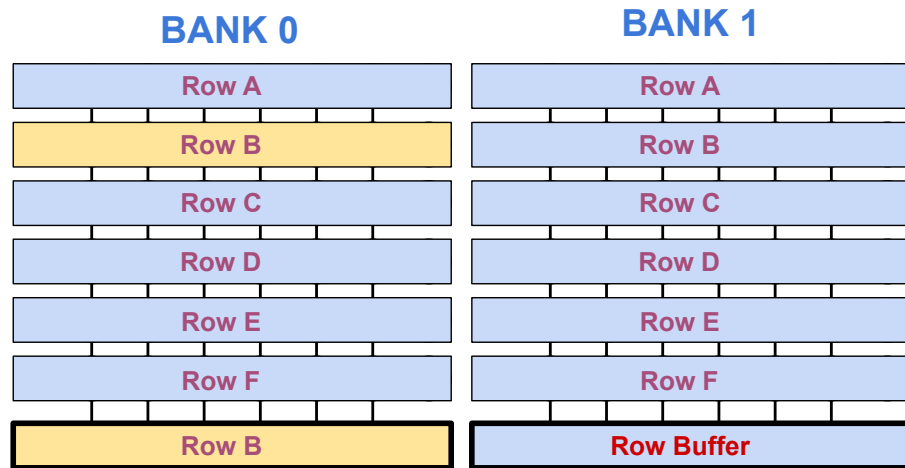
Side-Channel Example



Accessing rows in the *same* bank is very slow

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 0, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 0, Row E)
    jmp loop
```

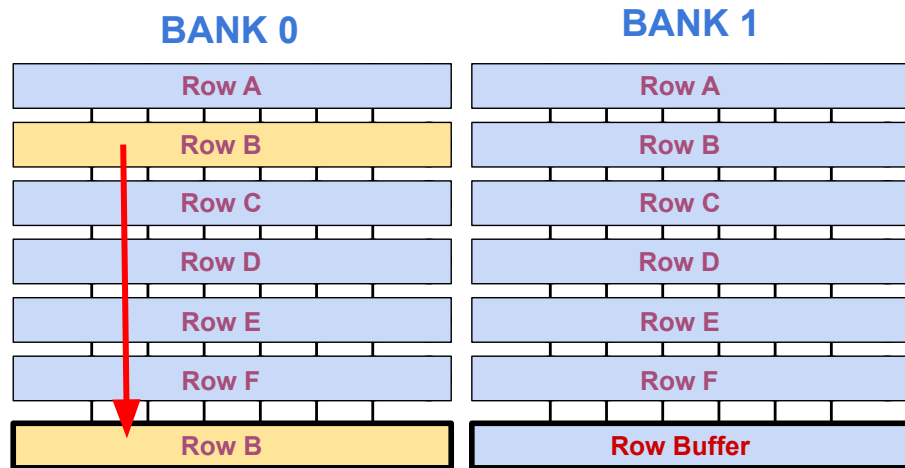
Side-Channel Example



Accessing rows in the *same* bank is very slow

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 0, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 0, Row E)
    jmp loop
```

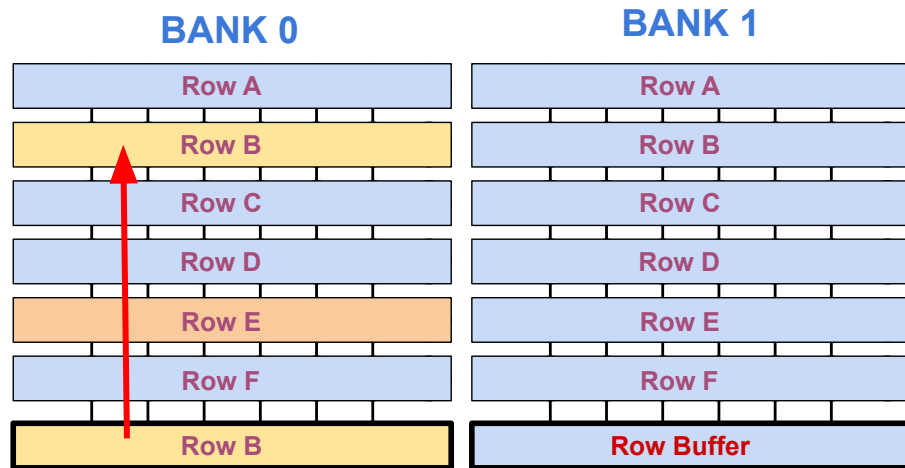
Side-Channel Example



Accessing rows in the *same* bank is very slow

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 0, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 0, Row E)
    jmp loop
```

Side-Channel Example

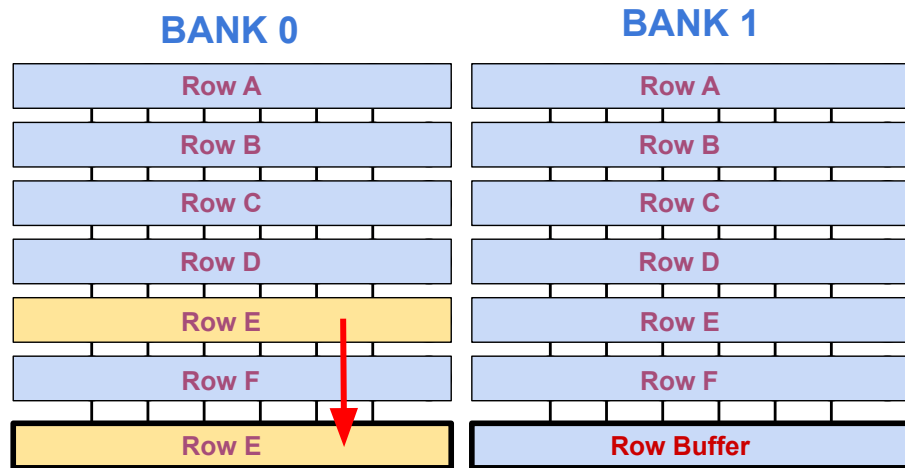


Accessing rows in the *same* bank is very slow

...because the rows are fighting for the same row buffer

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 0, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 0, Row E)
    jmp loop
```

Side-Channel Example

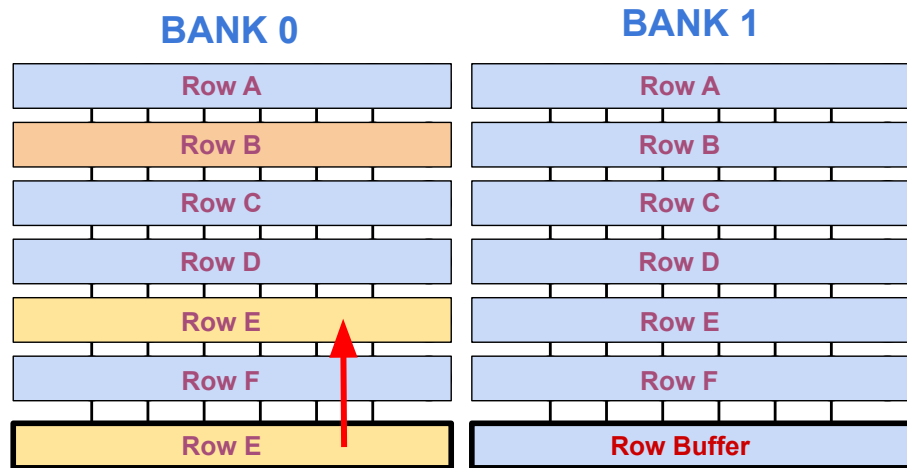


Accessing rows in the *same* bank is very slow

...because the rows are fighting for the same row buffer

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 0, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 0, Row E)
    jmp loop
```

Side-Channel Example



Accessing rows in the *same* bank is very slow

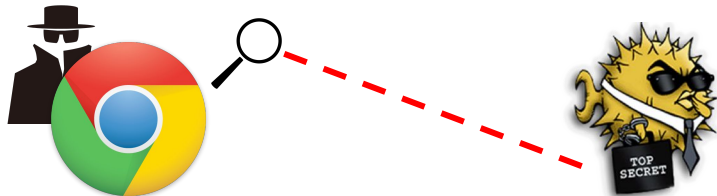
...because the rows are fighting for the same row buffer

```
loop:
    mov (Bank 0, Row B), %eax
    mov (Bank 0, Row E), %eax
    clflush(Bank 0, Row B)
    clflush(Bank 0, Row E)
    jmp loop
```

Finding Addresses $\longleftrightarrow$ Rows

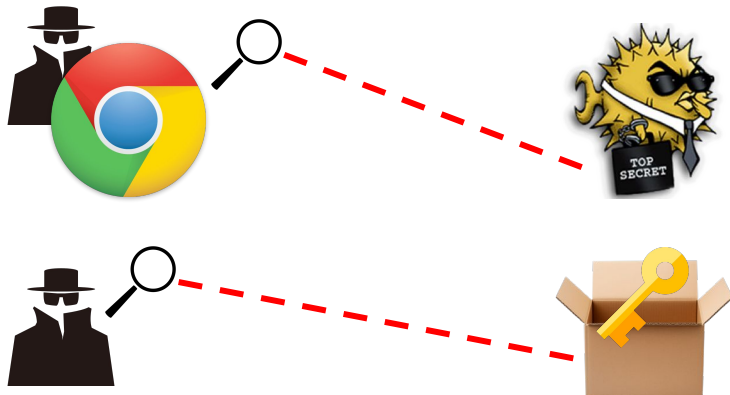
- We repeat that process for large amounts of memory, then use some math to figure out the exact functions
- The bottom line: **by using a timing side-channel, we can find rows that map to the same bank**
- This allows us to Rowhammer effectively!

Rowhammer



An attacker can escape
browser sandboxes...
(Seaborn and Dullien, '15)

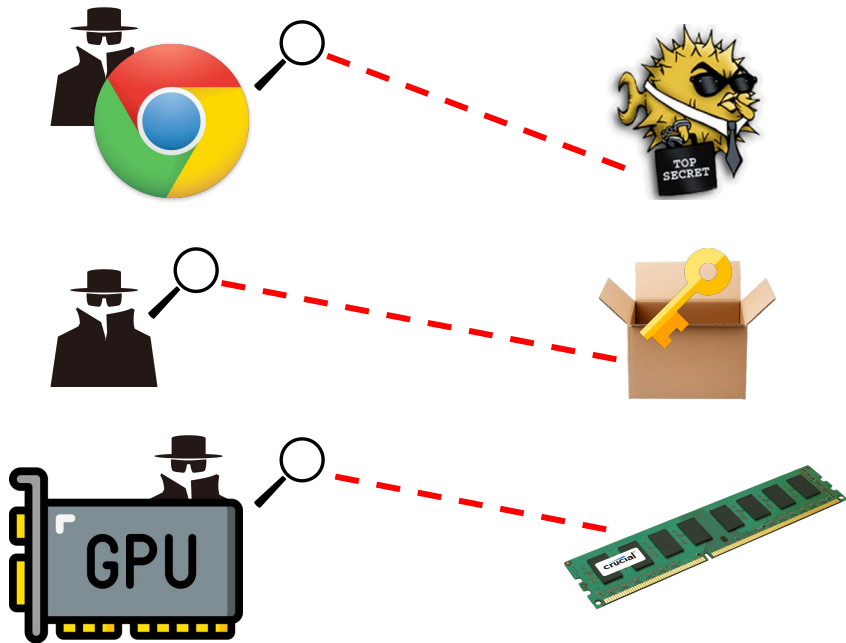
Rowhammer



An attacker can escape browser sandboxes...
(Seaborn and Dullien, '15)

...read private RSA keys...
(Kwong et al., IEEE SP '20)

Rowhammer



An attacker can escape browser sandboxes...
(Seaborn and Dullien, '15)

...read private RSA keys...
(Kwong et al., IEEE SP '20)

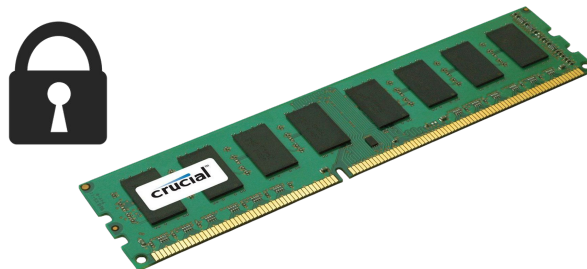
...and even change host memory from the GPU!
(Hu et al., IEEE SP '26)

Current Mitigations

Software-side
(Brasser et al., SEC '17)



Hardware-side
(Hassan et al., MICRO '21)

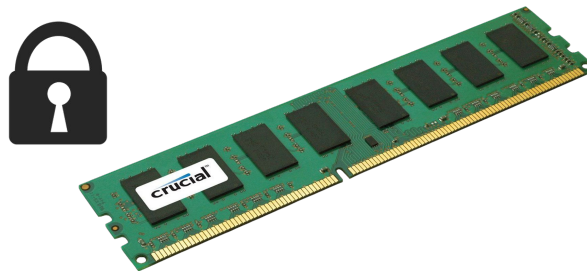


Current Mitigations

Software-side
(Brasser et al., SEC '17)



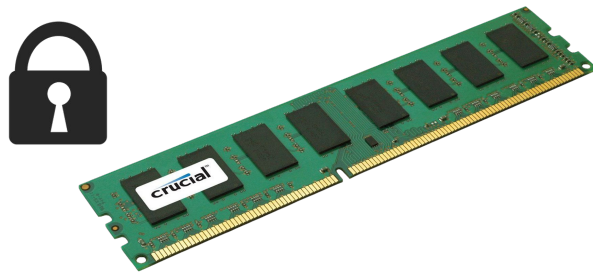
Hardware-side
(Hassan et al., MICRO '21)



Nearly every DDR4+ module
today has mitigations!

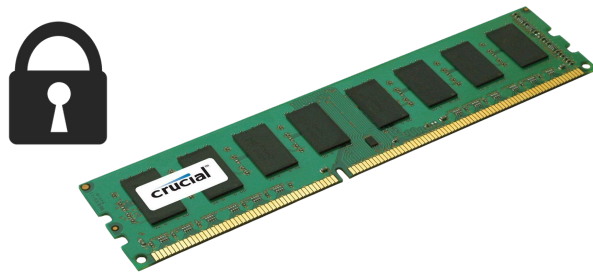
Current Mitigations

Future work was able to break these mitigations...



Current Mitigations

...on SW side...
(Cheng et al., TDSC '19)

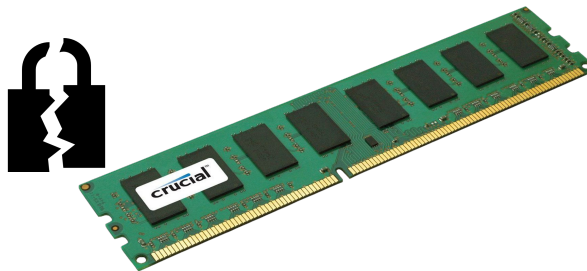


Current Mitigations

...on SW side...
(Cheng et al., TDSC '19)



...and on HW side!
(Frigo et al., IEEE SP '20)



TRR

Memory Controller (MC) keeps track of frequently-activated rows



Row #	Activations
6	2800
17	2200
10	1600
14	2400
4	1900

TRR

Memory Controller (MC) keeps track of frequently-activated rows

When a row is activated enough times, refresh row's neighbors



Row #	Activations
6	2800
17	2200
10	1600
14	2400
4	1900

TRR

Memory Controller (MC) keeps track of frequently-activated rows

When a row is activated enough times, refresh row's neighbors

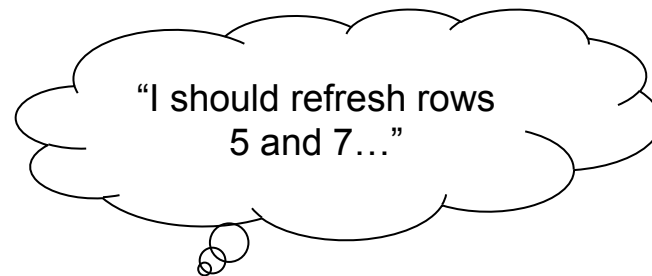


Row #	Activations
6	2800
17	2200
10	1600
14	2400
4	1900

TRR

Memory Controller (MC) keeps track of frequently-activated rows

When a row is activated enough times, refresh row's neighbors



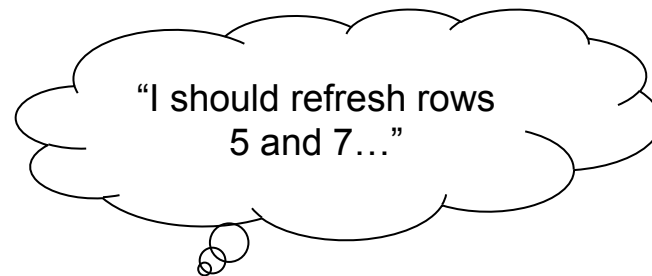
Row #	Activations
6	2800

TRR

Memory Controller (MC) keeps track of frequently-activated rows

When a row is activated enough times, refresh row's neighbors

TRR effectively squashed first class of Rowhammer attacks



Row #	Activations
6	2800

TRRespass

The TRR mechanism can be overwhelmed

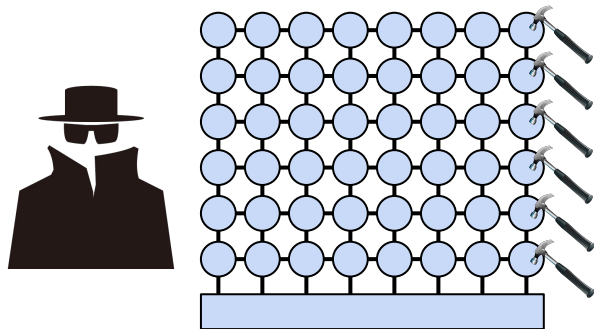


Row #	Activations

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

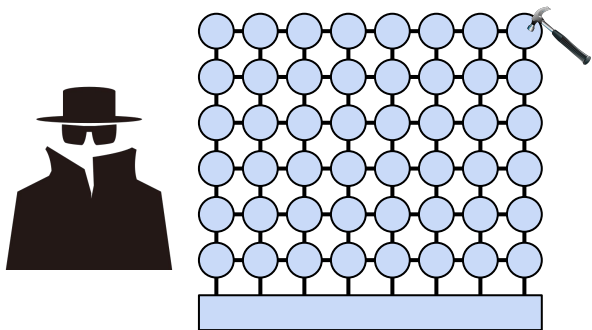


Row #	Activations

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

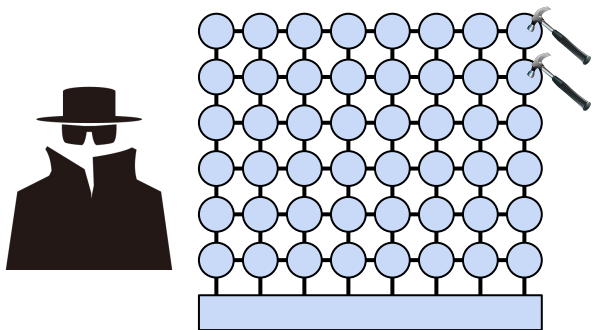


Row #	Activations
0	1

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

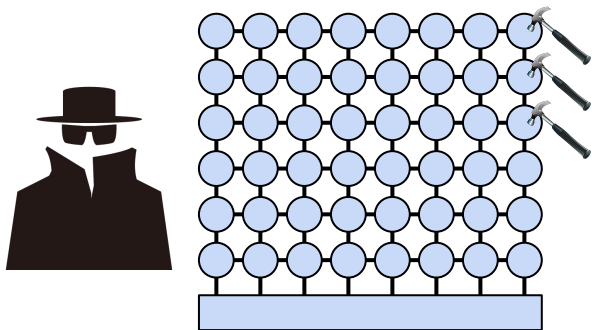


Row #	Activations
0	1
1	1

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

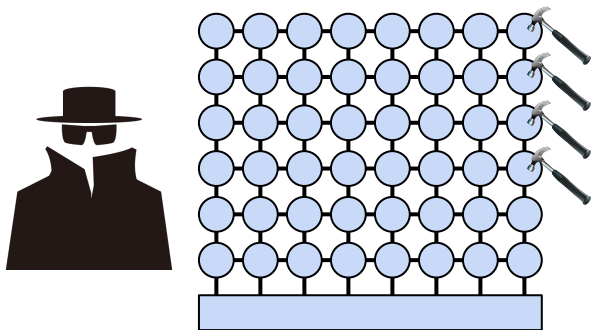


Row #	Activations
0	1
1	1
2	1

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

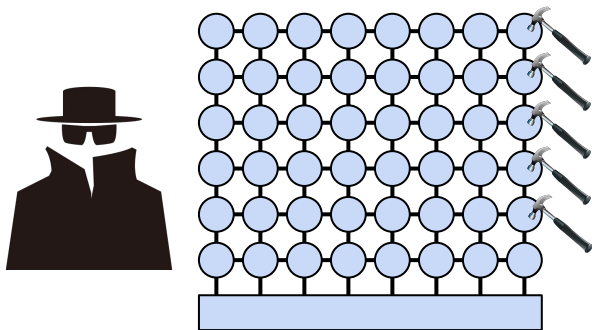


Row #	Activations
0	1
1	1
2	1
3	1

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

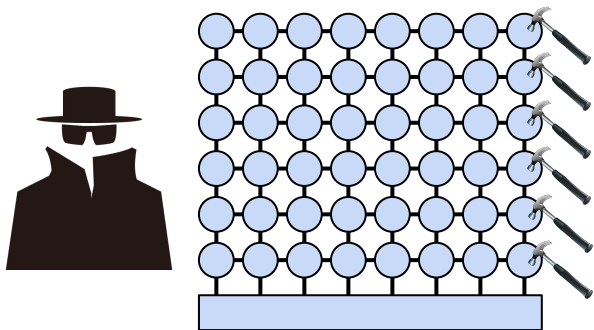


Row #	Activations
0	1
1	1
2	1
3	1
4	1

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)

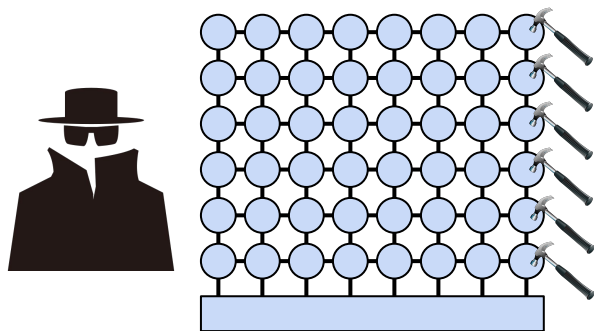


Row #	Activations
0	1
1	1
2	1
3	1
4	1

TRRespass

The TRR mechanism can be overwhelmed

When attacking, hammer *many* rows, not just 2 (Frigo et al., IEEE SP '20)



Row #	Activations
0	1
1	1
2	1
3	1
4	1

Closing Remarks

- Rowhammer occurs when memory is repeatedly accessed, causing electrical interference that flips bits indirectly
- Bugs/vulnerabilities can be patched; Rowhammer cannot
 - ...it is a byproduct of *how* DRAM works
- Thus, clever mitigations are required

Questions?
