# GLitch chronicles:

## turning WebGL into a hammer

Pietro Frigo

Cristiano Giuffrida, Herbert Bos, Kaveh Razavi

VU VRIJE UNIVERSITEIT AMSTERDAM

VUSec

# GLitch: what?

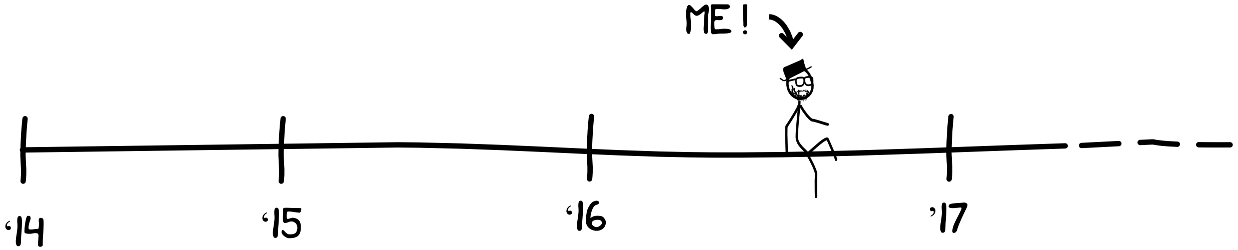
First **Rowhammer** exploit from **JavaScript** on **mobile**

**Fastest** JS-based Rowhammer exploit

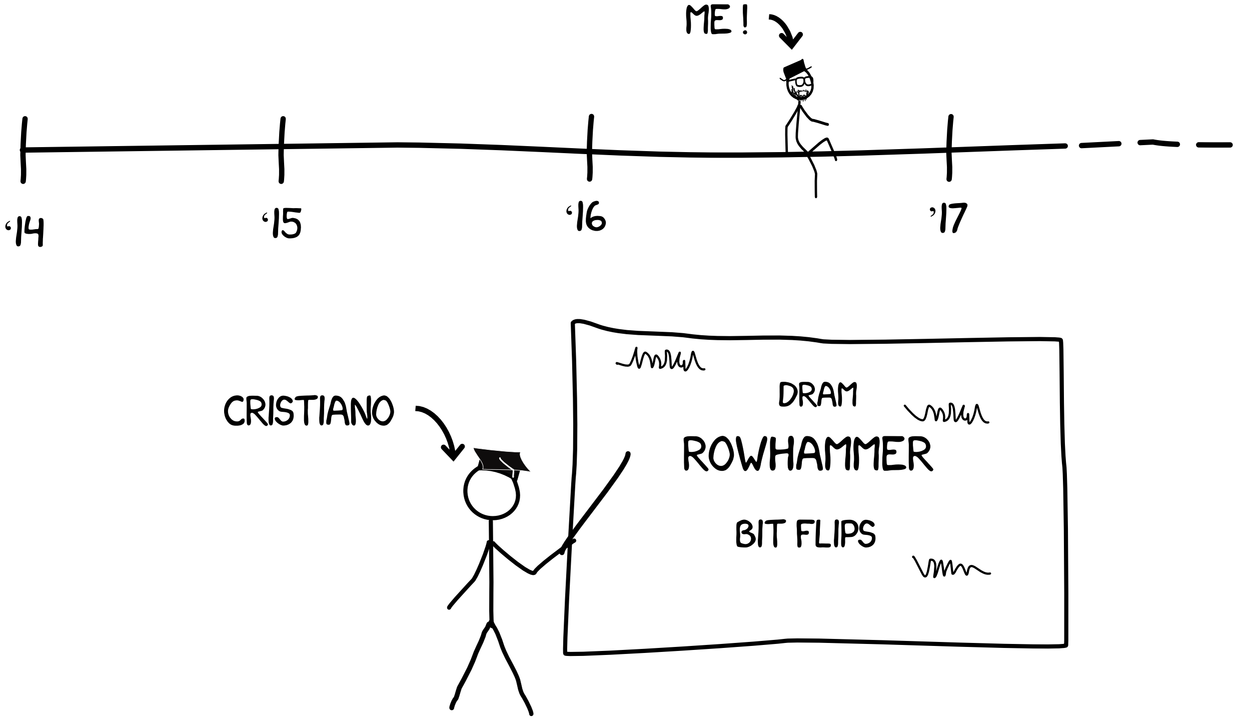
First **GPU-accelerated** bit flip

# The chronicles of GLitch

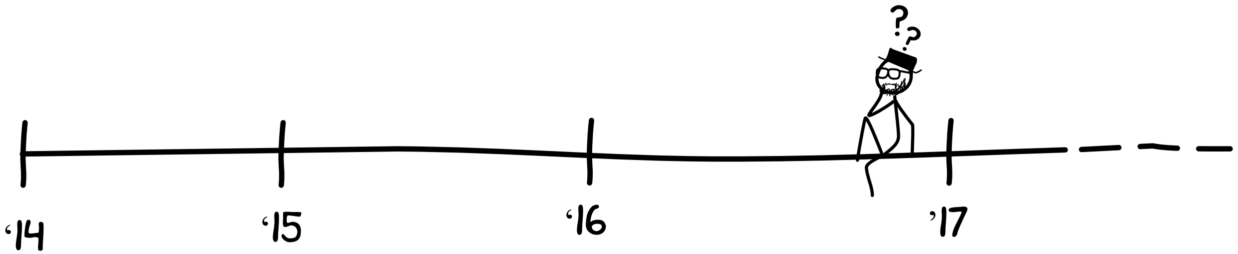
# GLitch: the chronicles

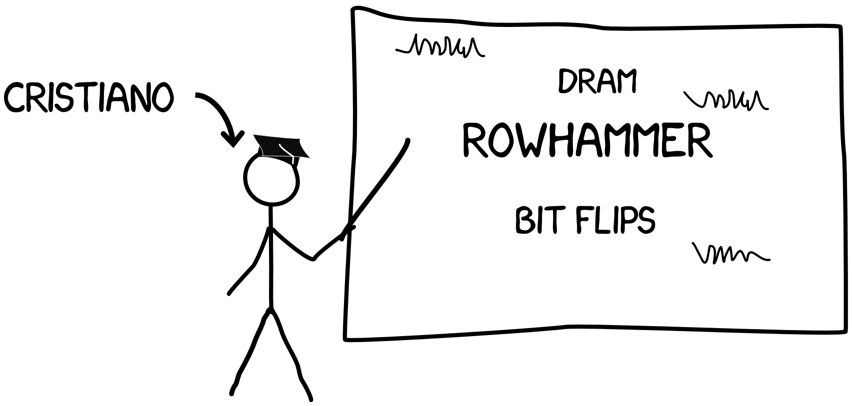


# GLitch: the chronicles

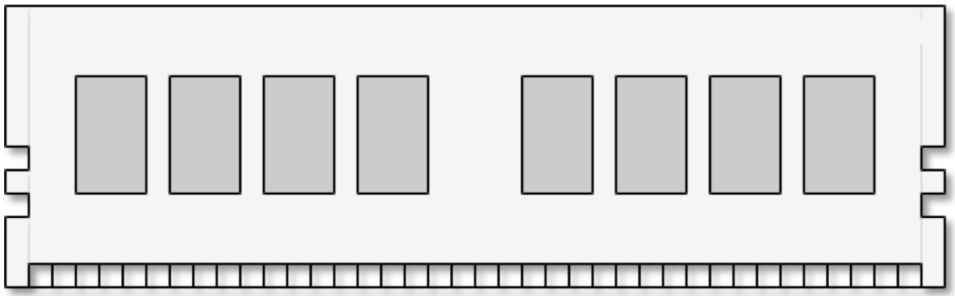


# GLitch: the chronicles

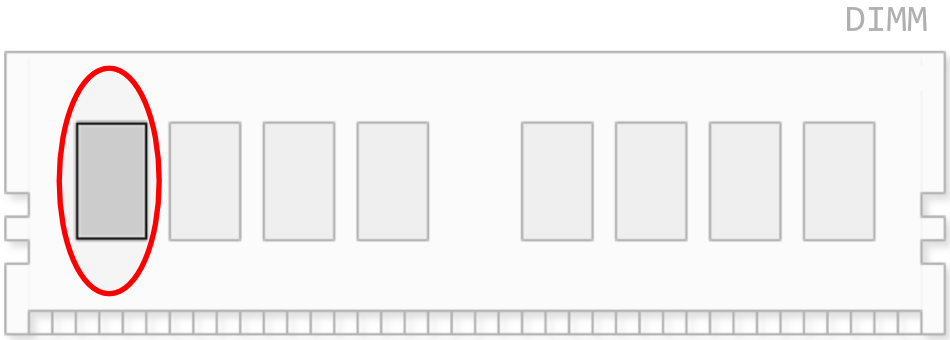


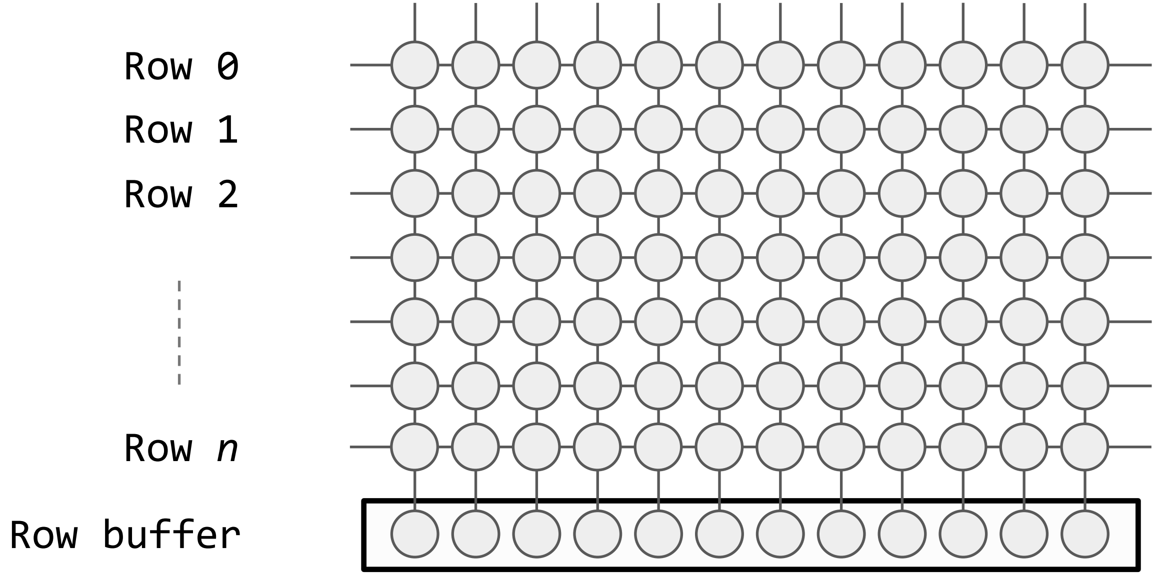
# DRAM



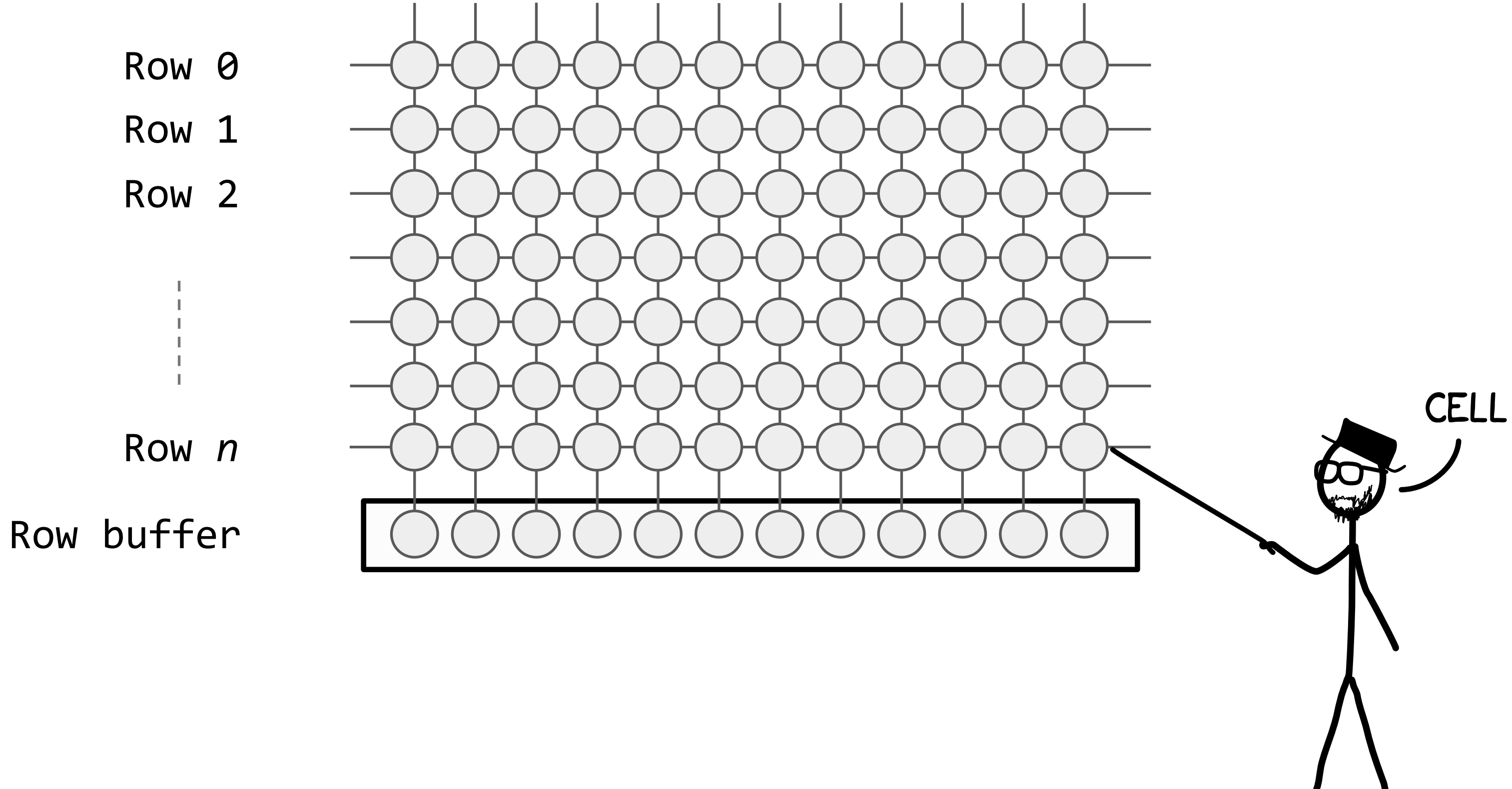
# DRAM



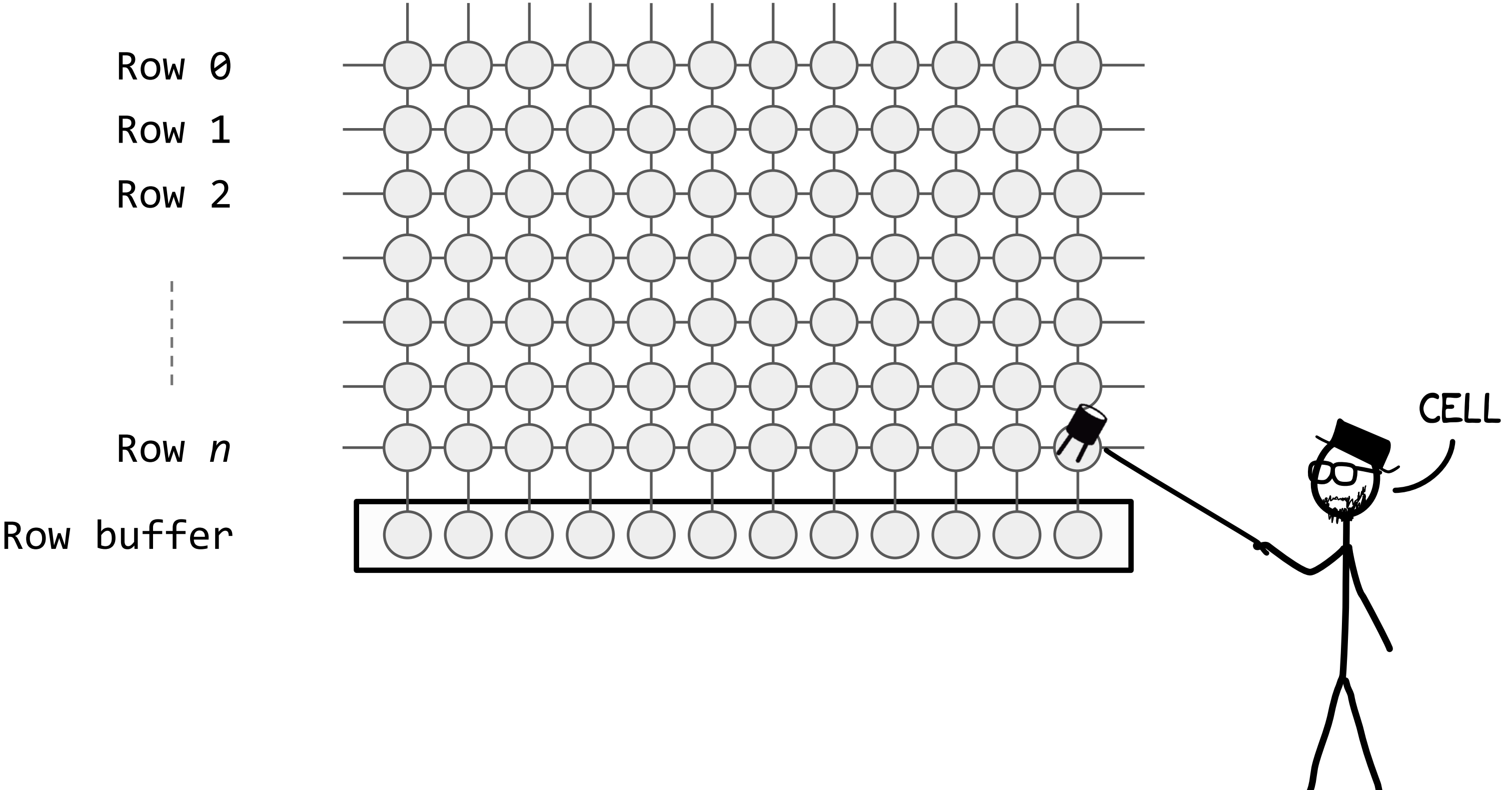
# DRAM



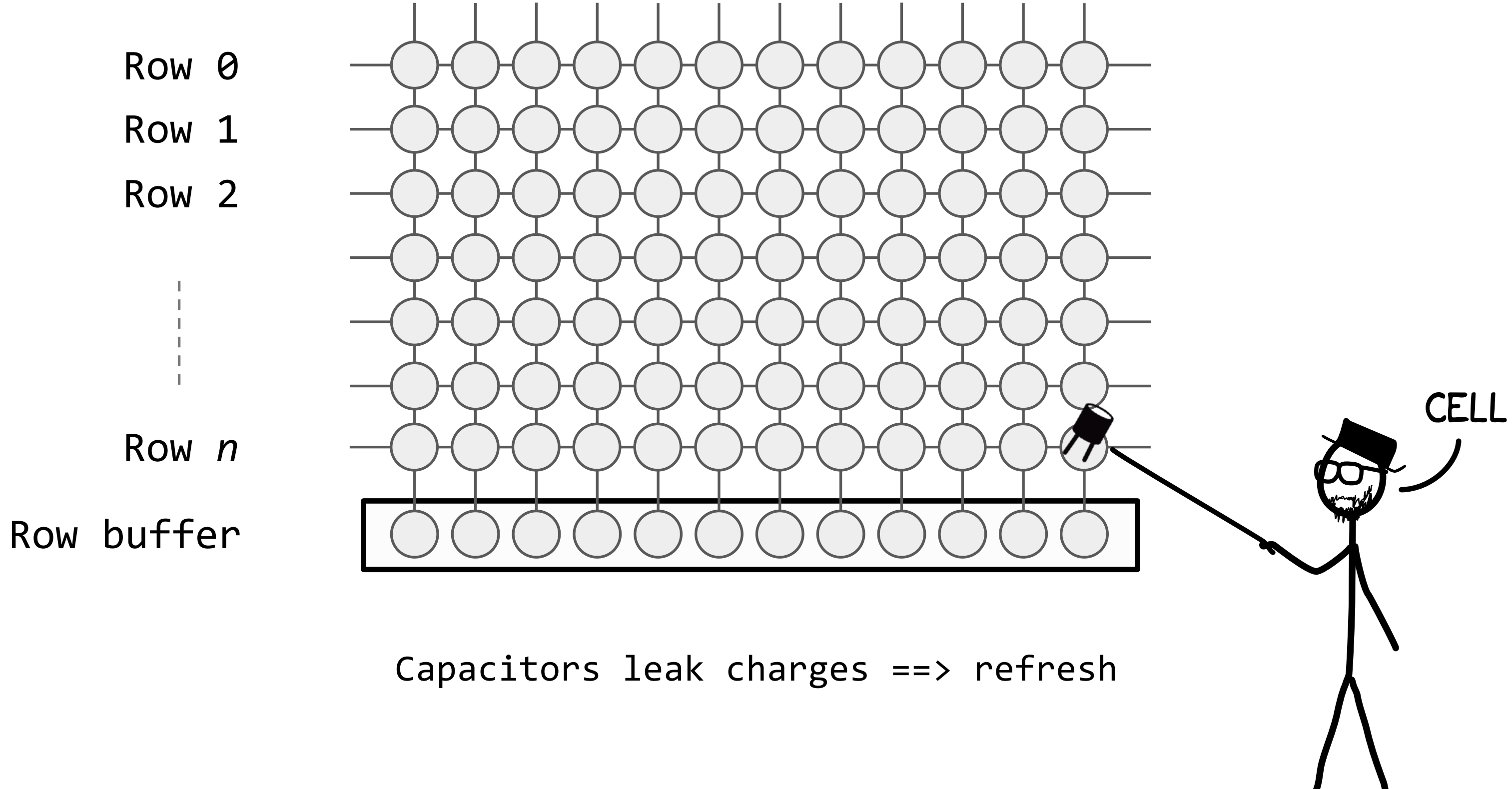
# DRAM



# DRAM

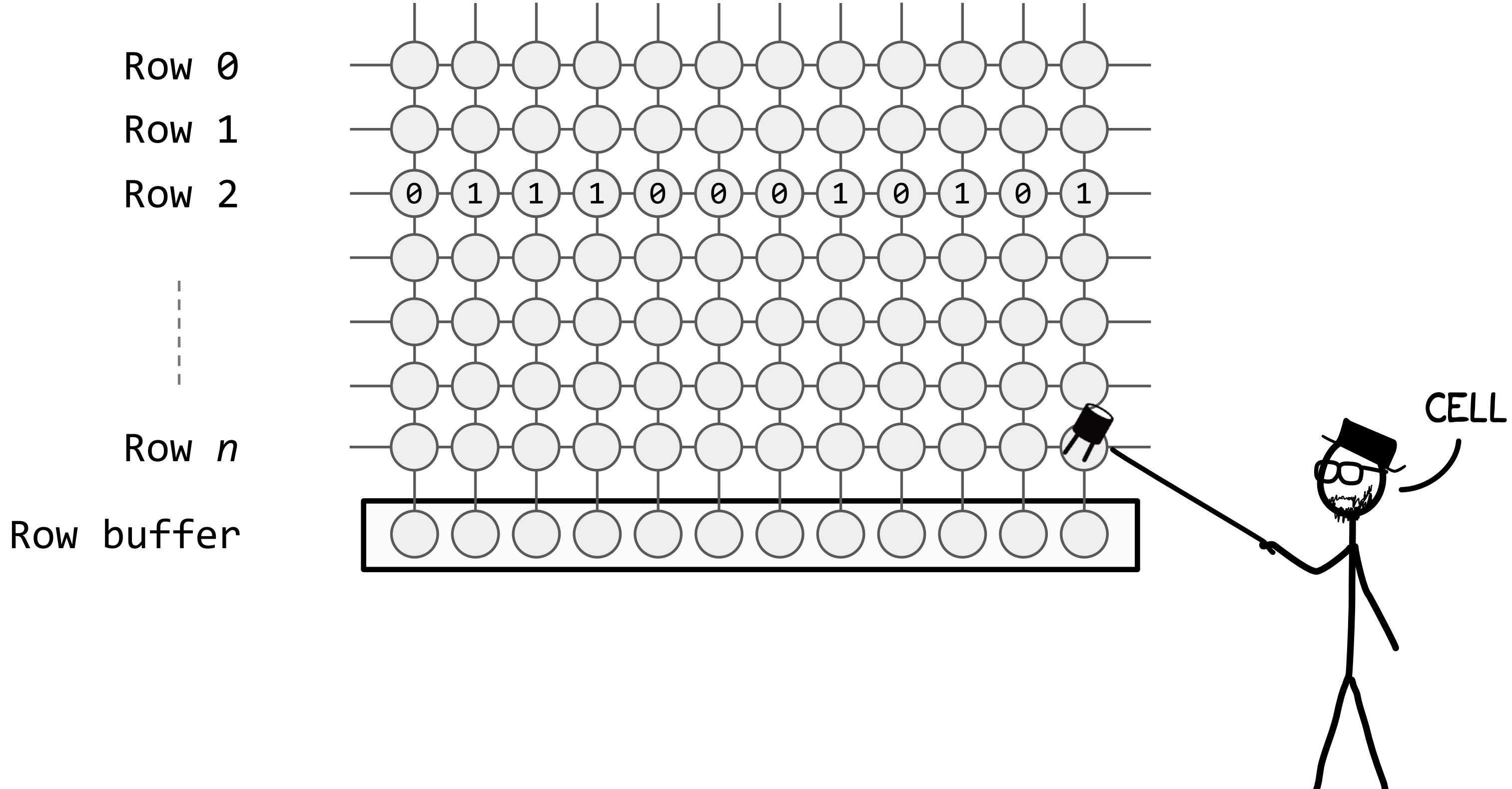


# DRAM

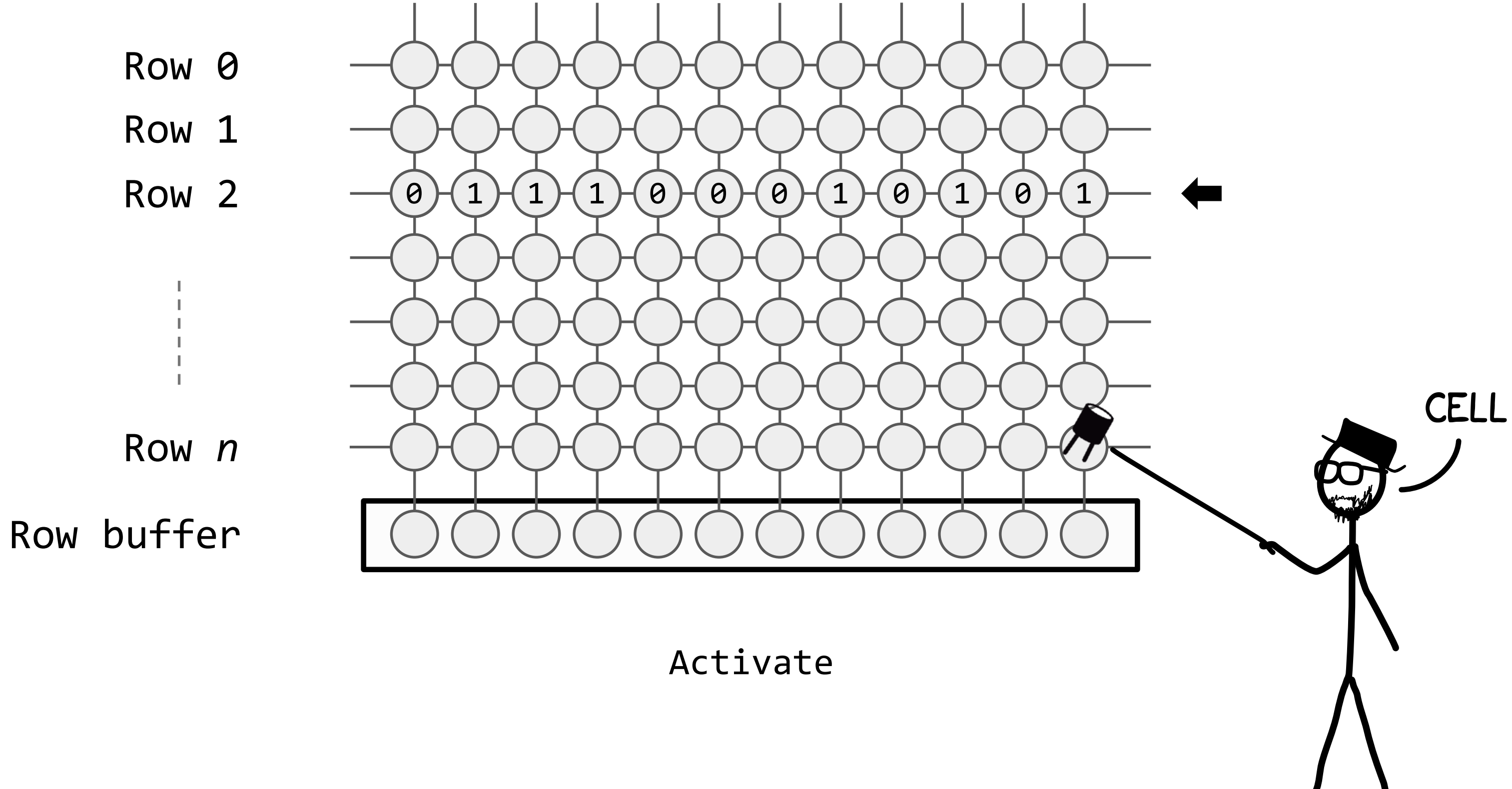



Capacitors leak charges ==> refresh

# DRAM

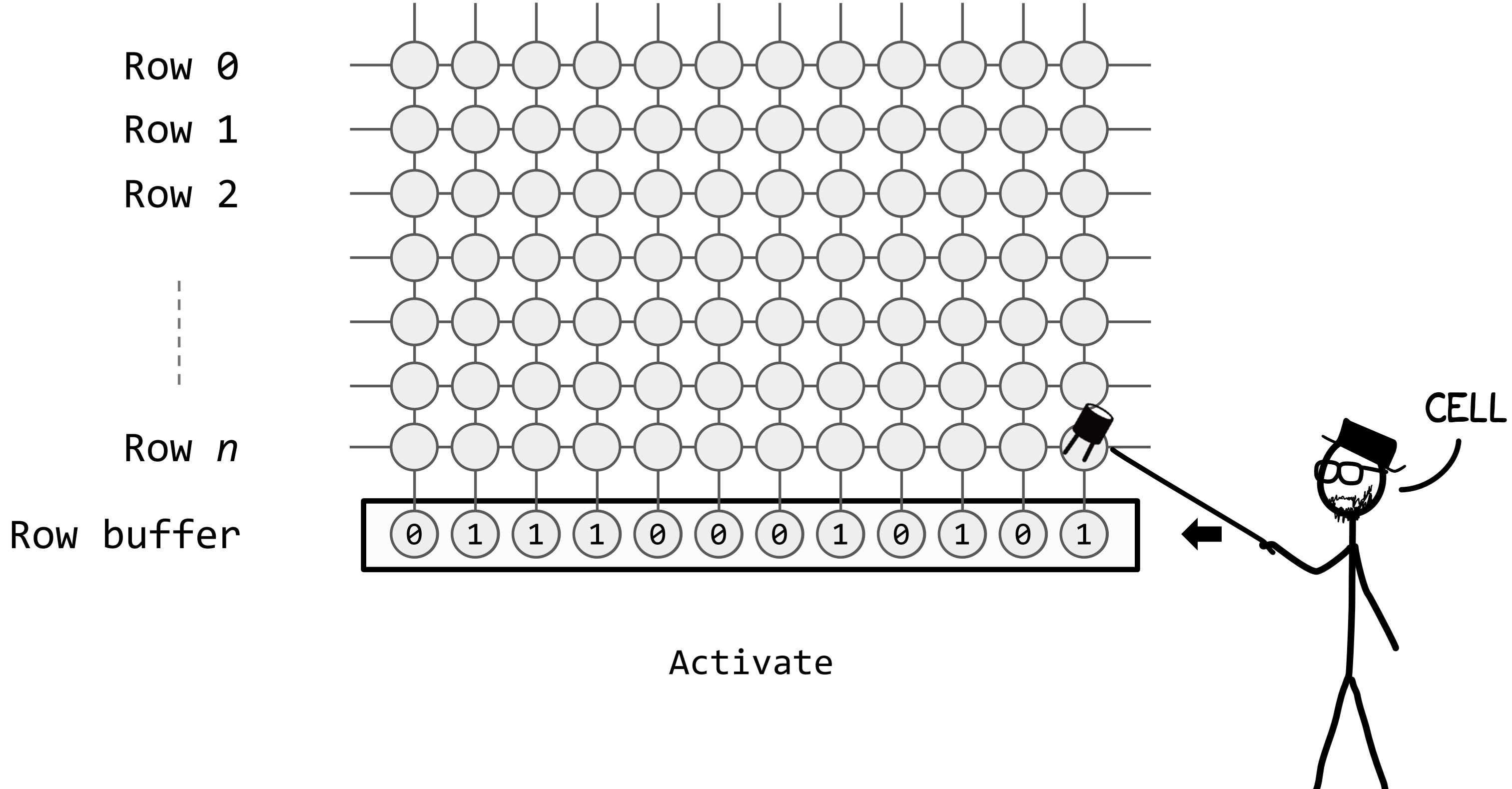


# DRAM

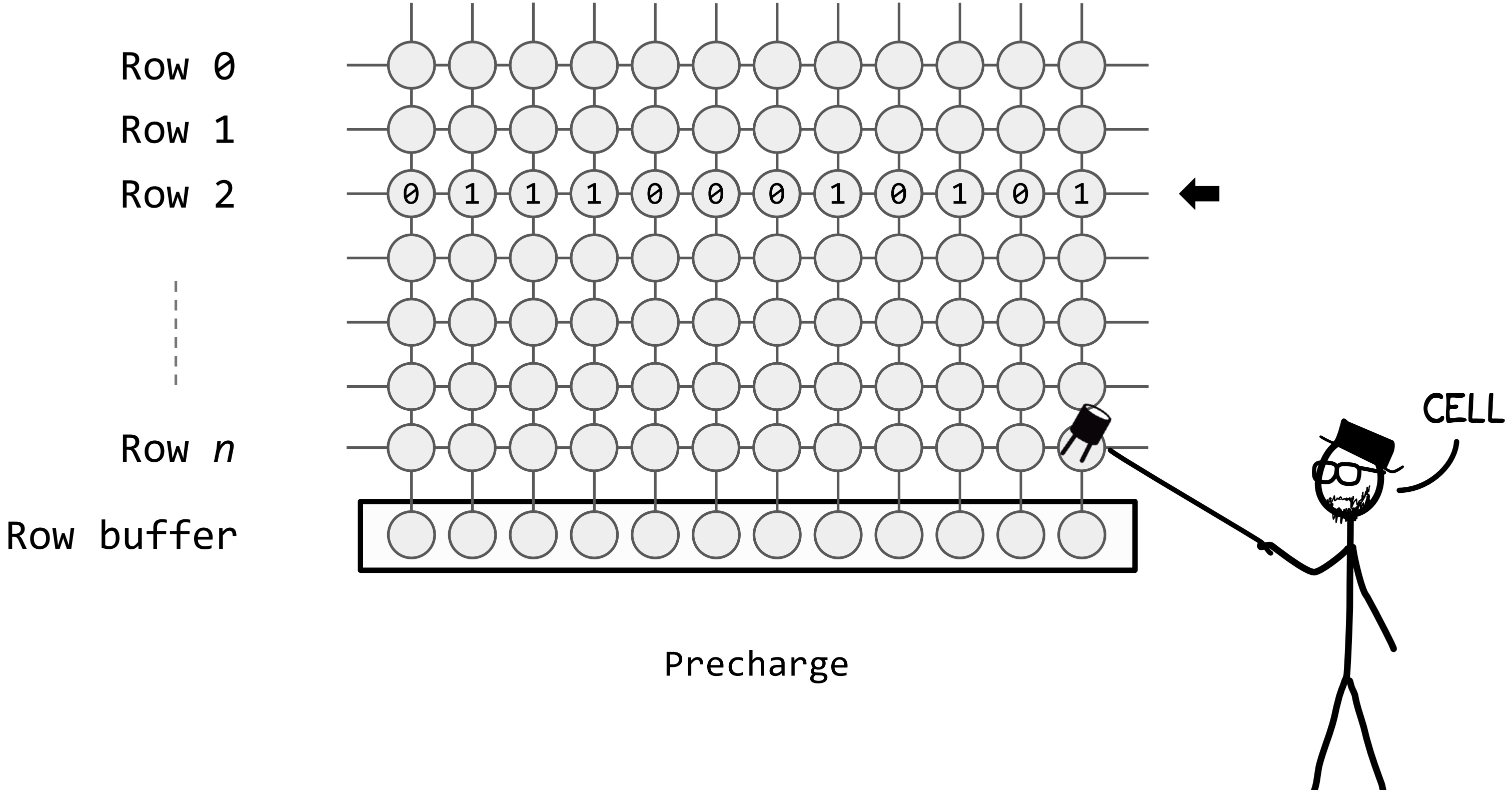



Activate

# DRAM

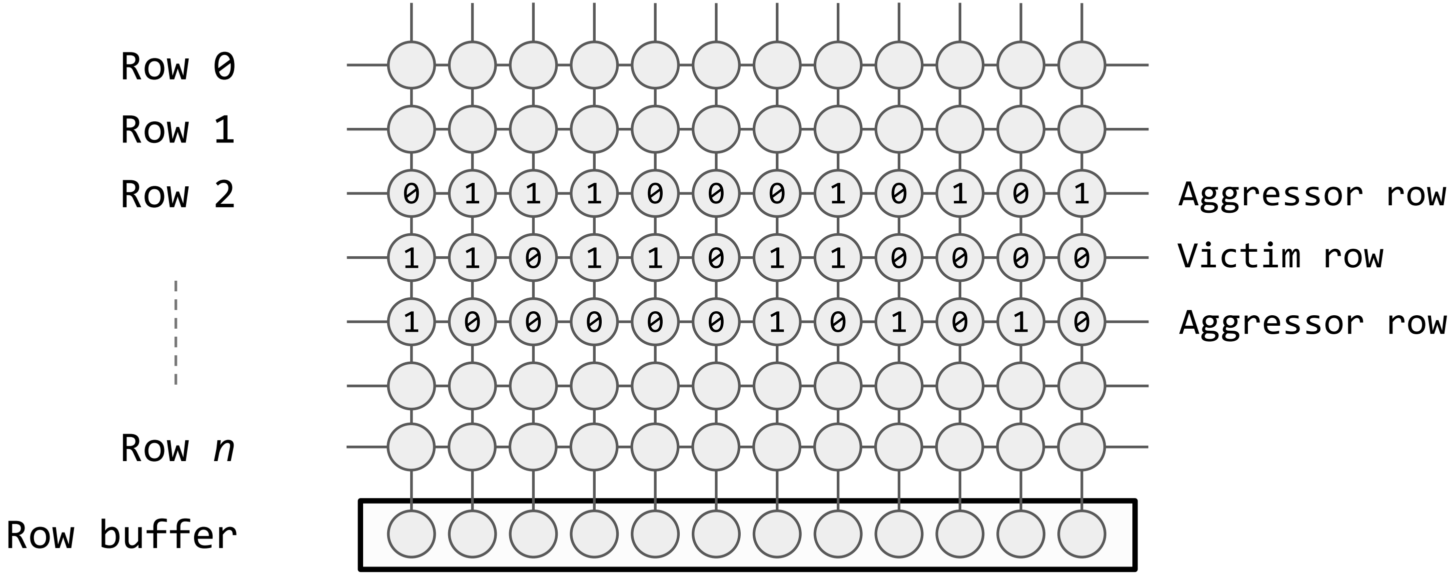


# DRAM



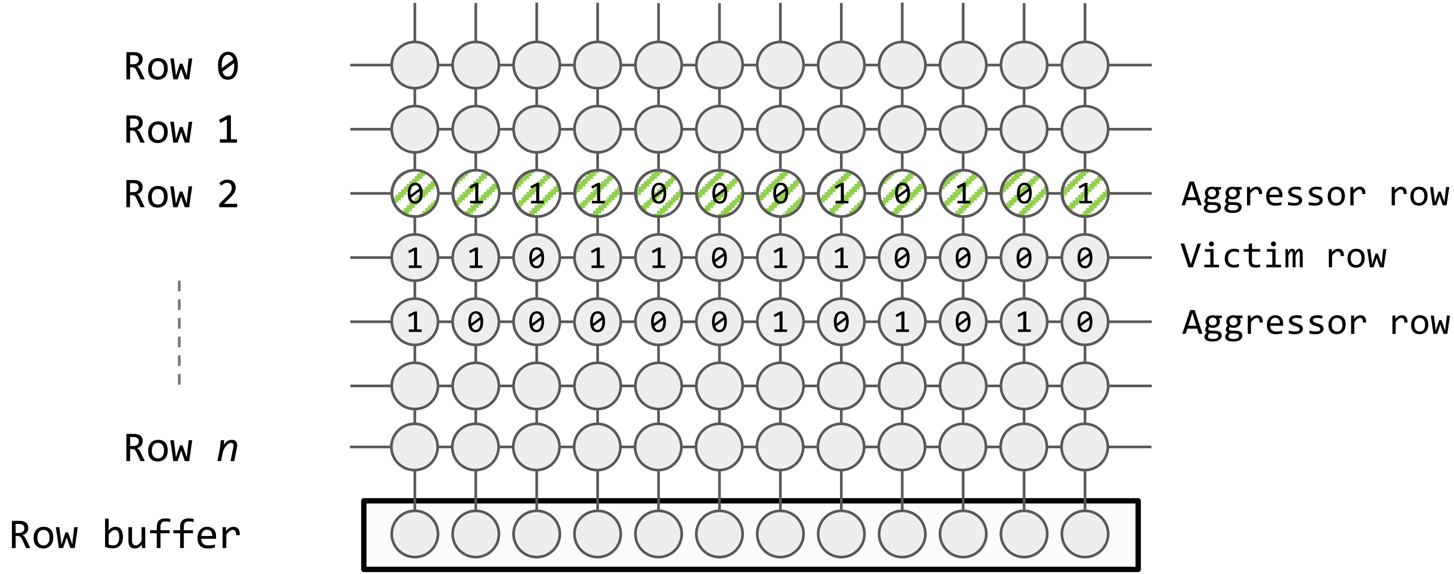
Row 0
Row 1
Row 2
0 1 1 1 0 0 0 1 0 1 0 1
Row n
Row buffer
CELL
Precharge

# Rowhammer



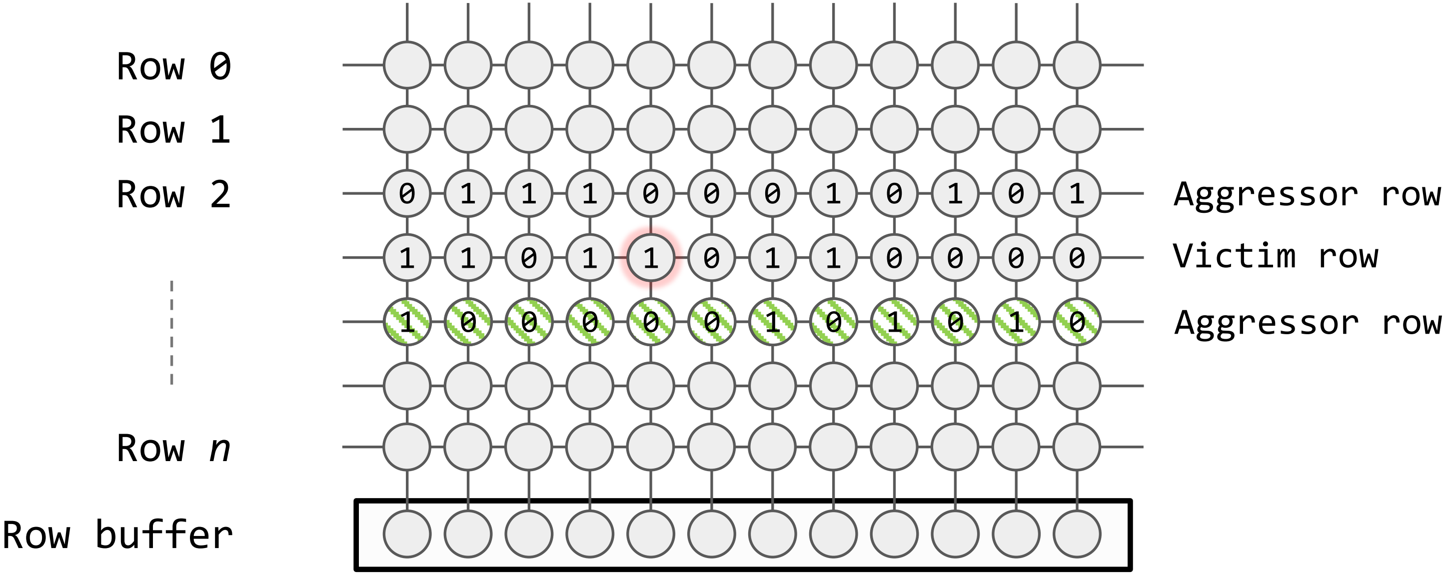
Row 0
Row 1
Row 2
0 1 1 1 0 0 0 1 0 1 0 1
Aggressor row
1 1 0 1 1 0 1 1 0 0 0 0
Victim row
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Row *n*
Row buffer

# Rowhammer

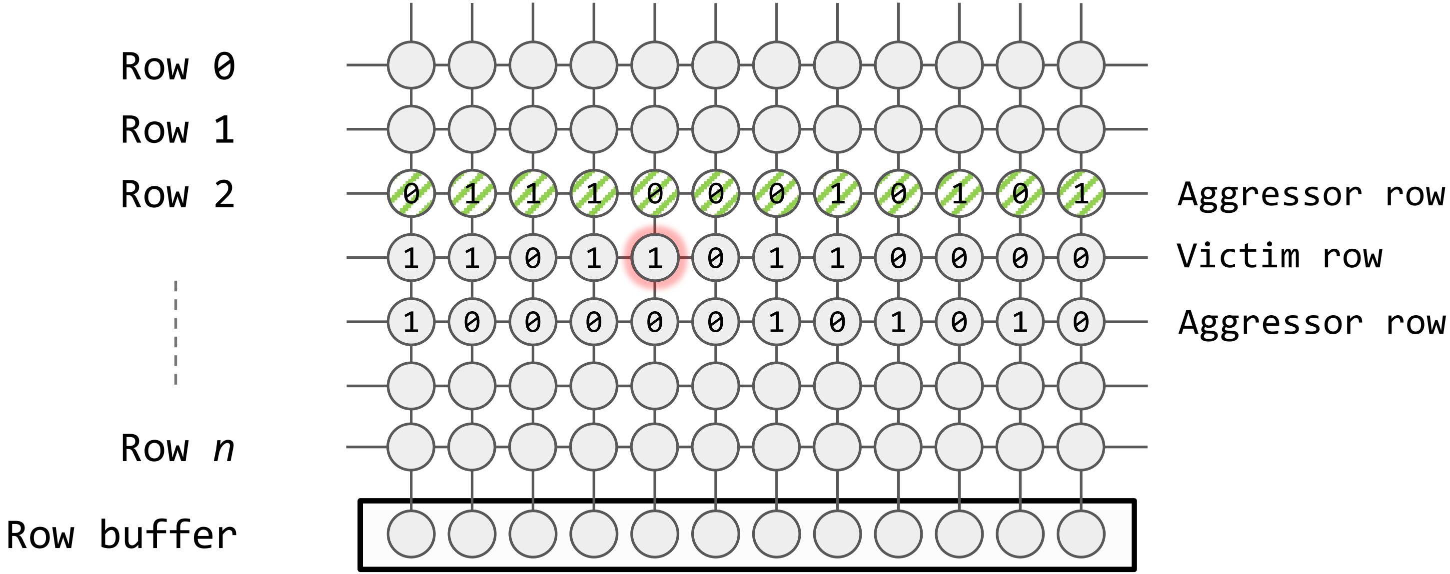


Row 0
Row 1
Row 2
Row n
Row buffer
0 1 1 1 0 0 0 1 0 1 0 1
1 1 0 1 1 0 1 1 0 0 0 0
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Victim row
Aggressor row

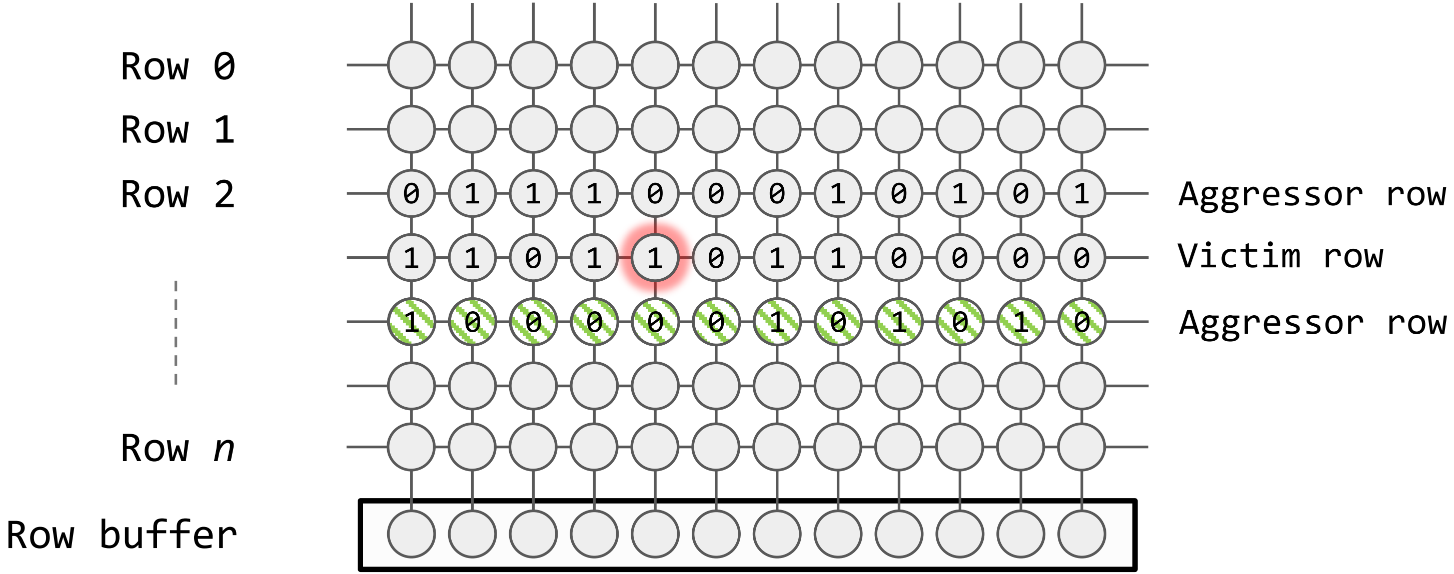
# Rowhammer


Row 0
Row 1
Row 2
0 1 1 1 0 0 0 1 0 1 0 1
Aggressor row
1 1 0 1 1 0 1 1 0 0 0 0
Victim row
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Row n
Row buffer

# Rowhammer

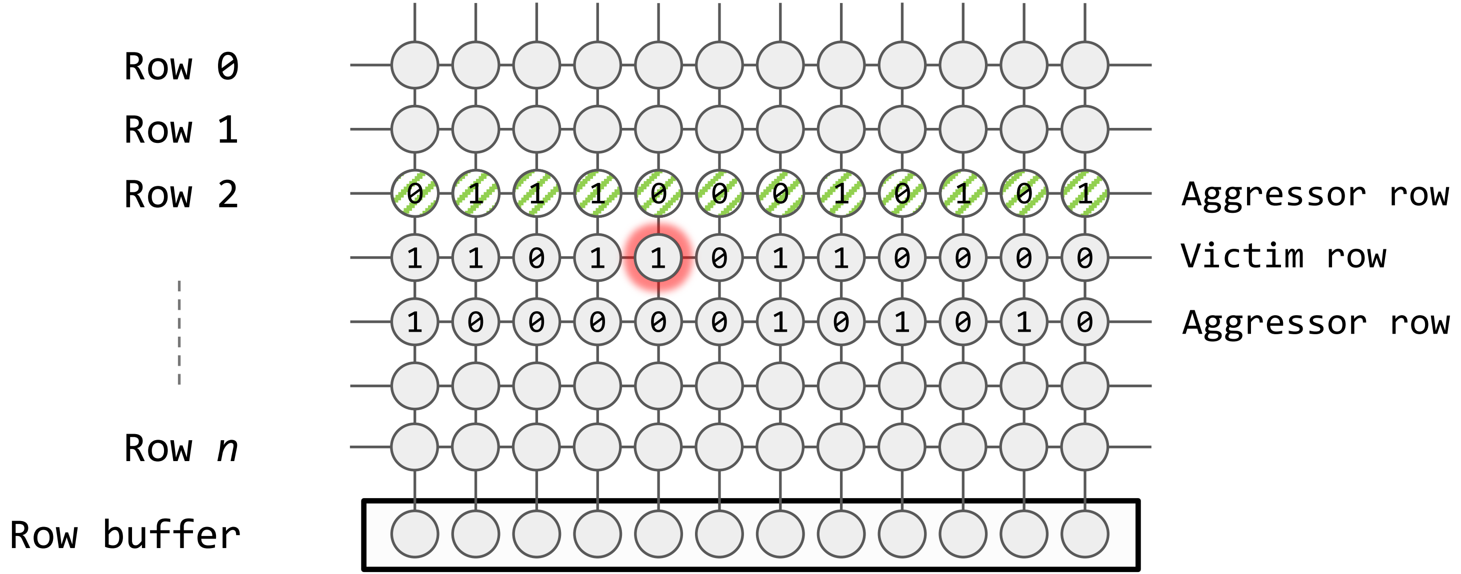

Row 0
Row 1
Row 2
Row n
Row buffer
0 1 1 1 0 0 0 1 0 1 0 1
1 1 0 1 1 0 1 1 0 0 0 0
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Victim row
Aggressor row

# Rowhammer



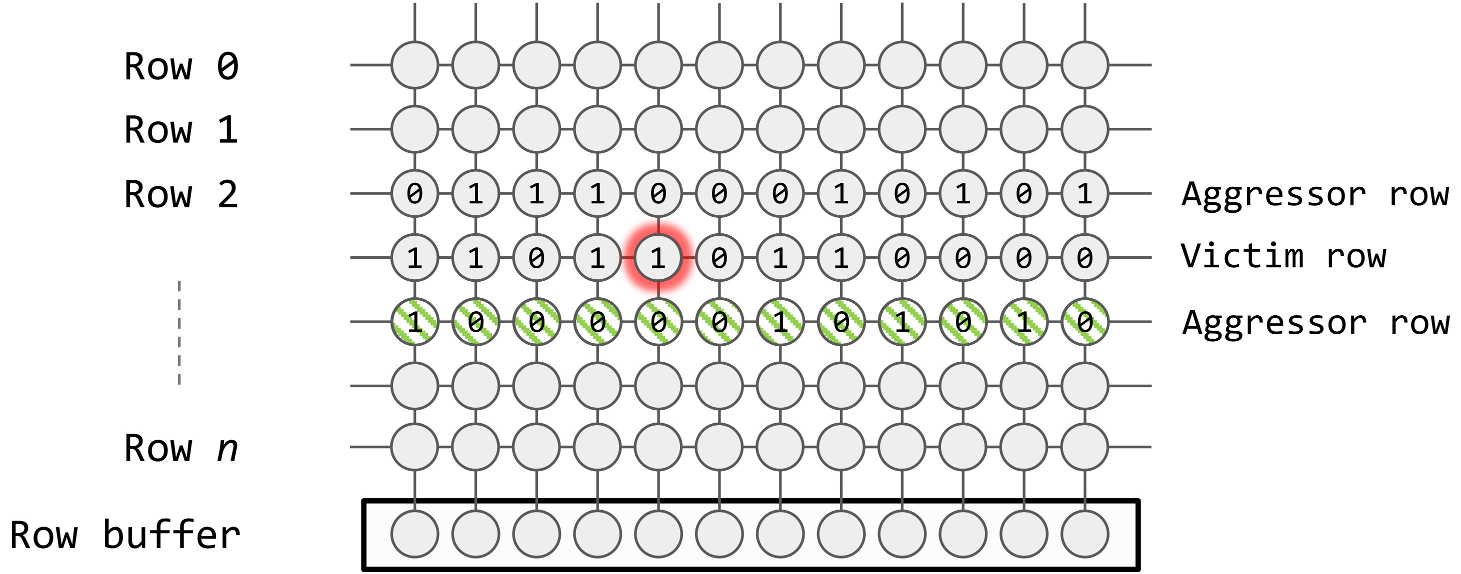
Row 0
Row 1
Row 2
0 1 1 1 0 0 0 1 0 1 0 1
Aggressor row
1 1 0 1 1 0 1 1 0 0 0 0
Victim row
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Row n
Row buffer

# Rowhammer



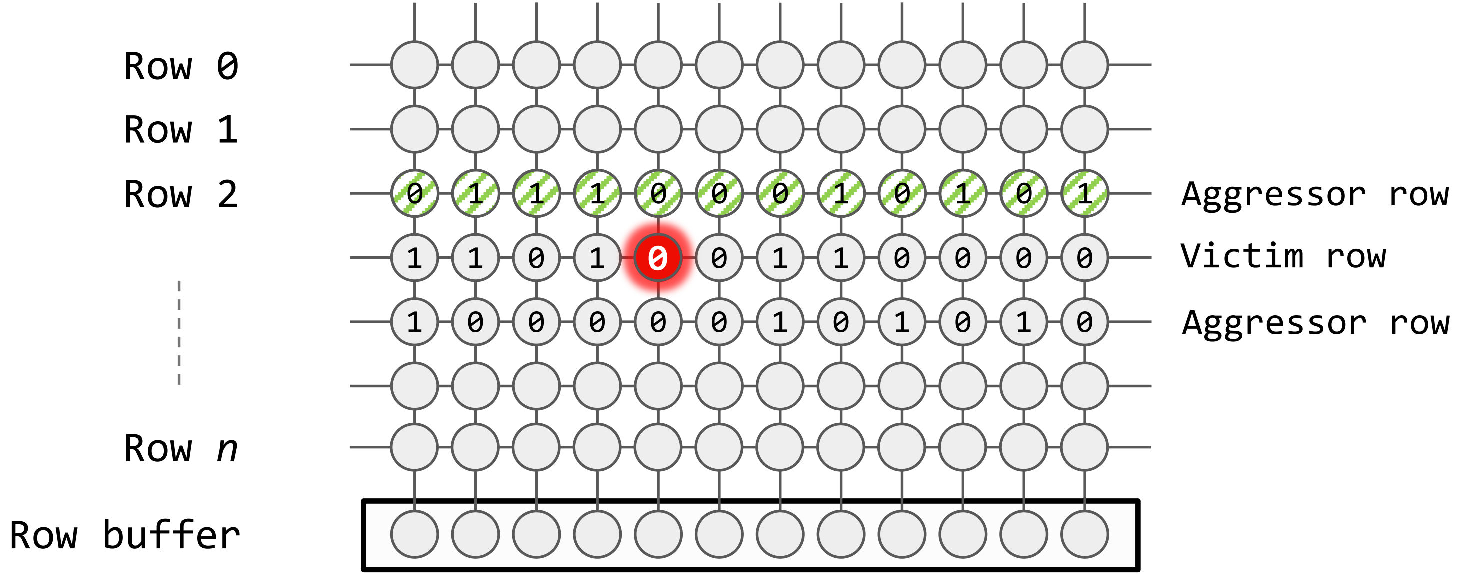
Row 0
Row 1
Row 2
Row n
Row buffer
0 1 1 1 0 0 0 1 0 1 0 1
1 1 0 1 1 0 1 1 0 0 0 0
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Victim row
Aggressor row

# Rowhammer



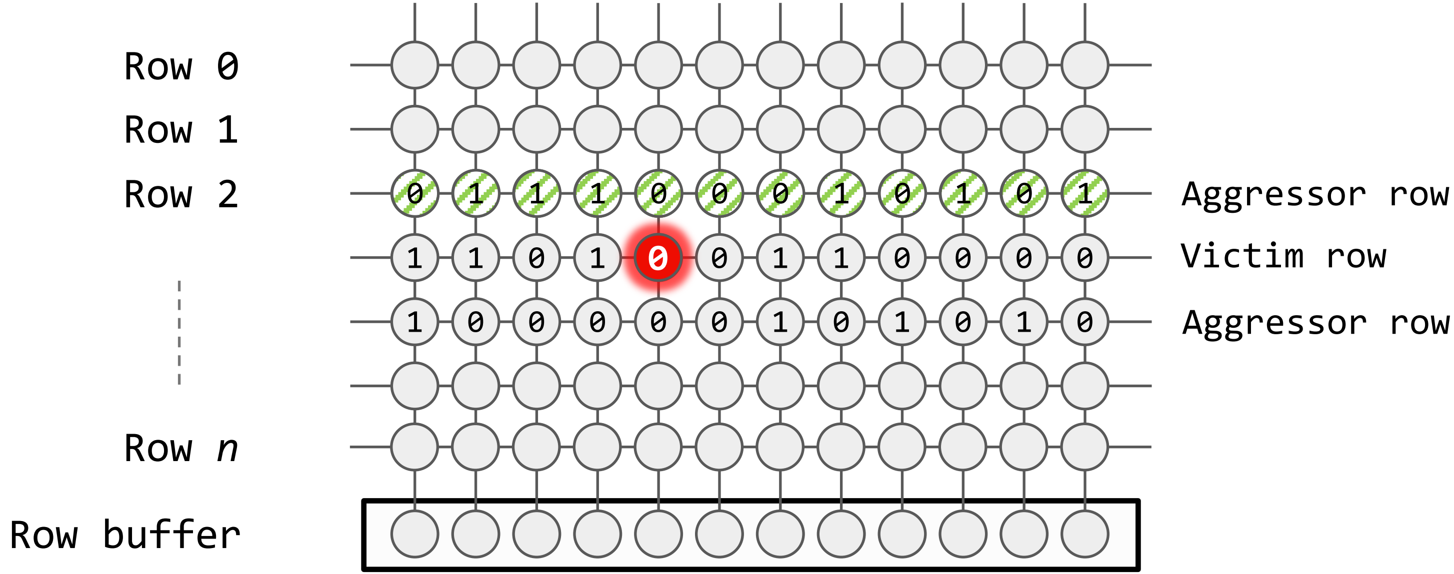
Row 0
Row 1
Row 2
Row n
Row buffer
0 1 1 1 0 0 0 1 0 1 0 1
1 1 0 1 1 0 1 1 0 0 0 0
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Victim row
Aggressor row

# Rowhammer



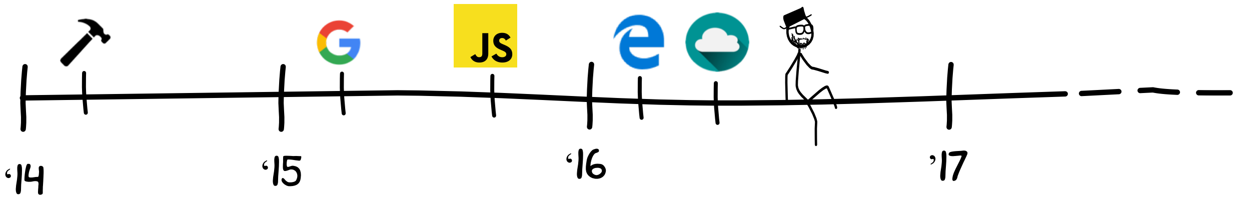
Row 0
Row 1
Row 2
Row n
Row buffer
0 1 1 1 0 0 0 1 0 1 0 1
1 1 0 1 0 0 1 1 0 0 0 0
1 0 0 0 0 0 1 0 1 0 1 0
Aggressor row
Victim row
Aggressor row

# Rowhammer




**Reproducible!**

# GLitch: the chronicles




[1] Flipping bits in memory without accessing them

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

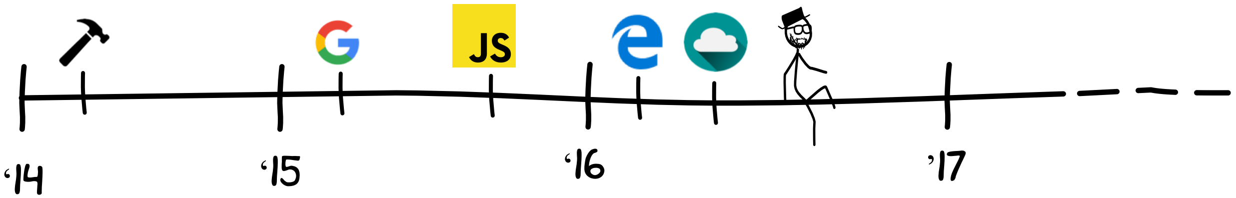
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

# GLitch: the chronicles




[1] Flipping bits in memory without accessing them

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

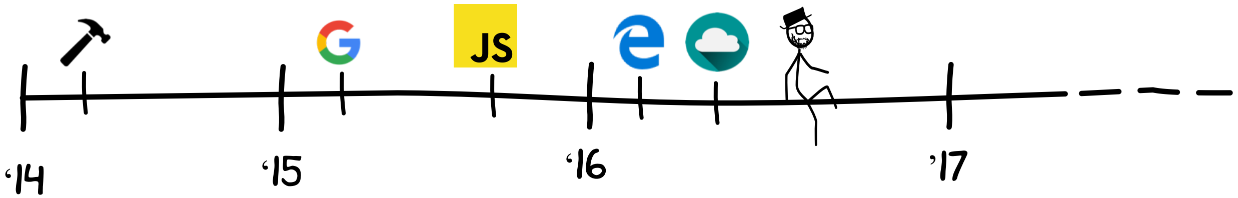
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

# GLitch: the chronicles




[1] Flipping bits in memory without accessing them

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

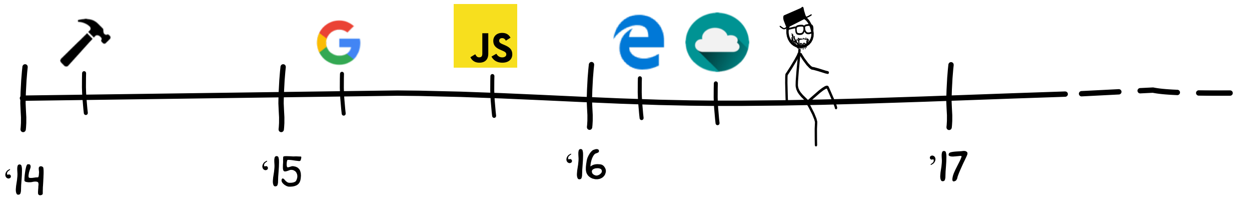
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

# GLitch: the chronicles




[1] Flipping bits in memory without accessing them

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

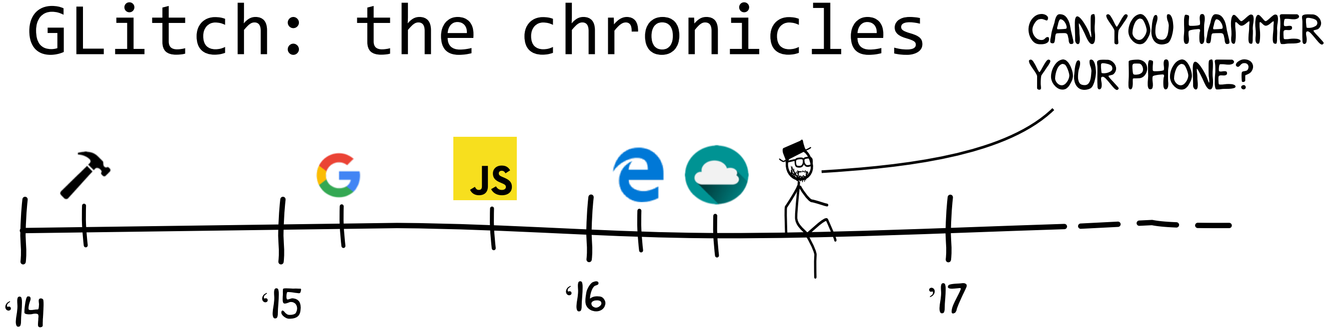
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

# GLitch: the chronicles




[1] Flipping bits in memory without accessing them

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

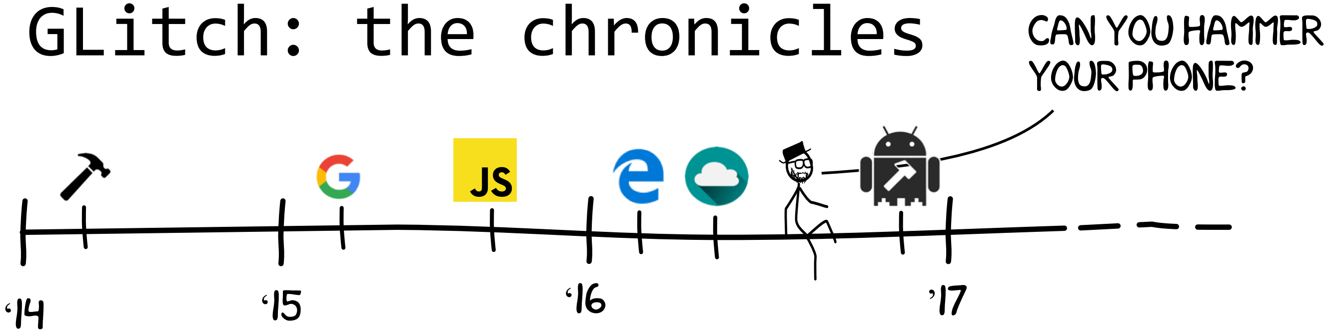
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

# GLitch: the chronicles




[1] Flipping bits in memory without accessing them

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

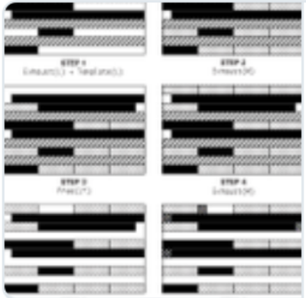
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

@vvdveen +100XP



**Drammer: Flip Feng Shui Goes Mobile - VUSec**

Drammer is the first instance of mobile Rowhammer and comprehends a deterministic Android root exploit that does not rely on any software vulnerability.

vusec.net

2 24 22

**Victor van der Veen** @vvdveen · 25 ott 2016

I wouldn't be surprised if we could pull this one from a browser actually...

Traduci dalla lingua originale: inglese

1 4 5

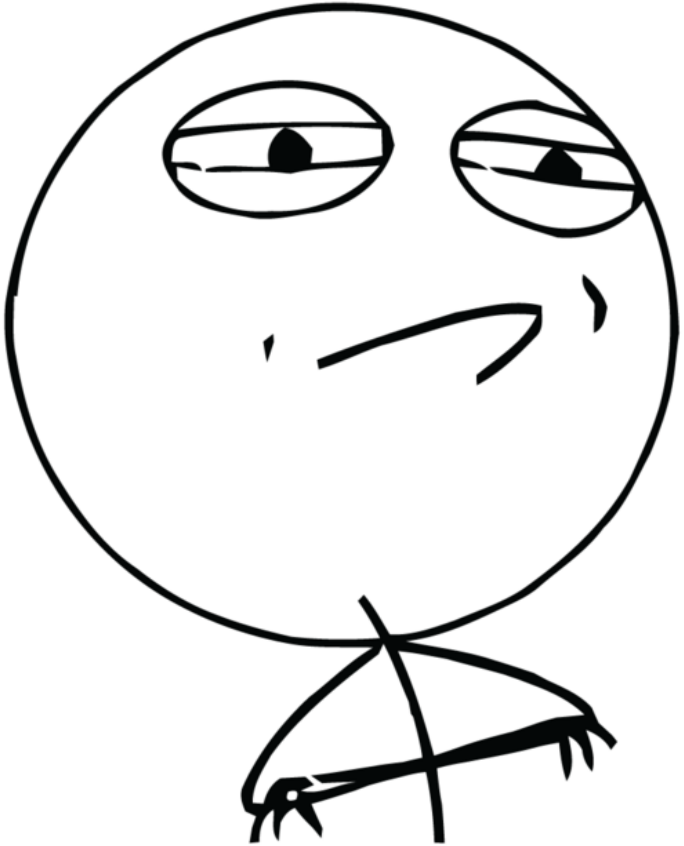
**the grugq**
@thegrugq

Following

In risposta a @vvdveen e @vu5ec

love to see it happen. :)

# CHALLENGE ACCEPTED



# Attacker primitives

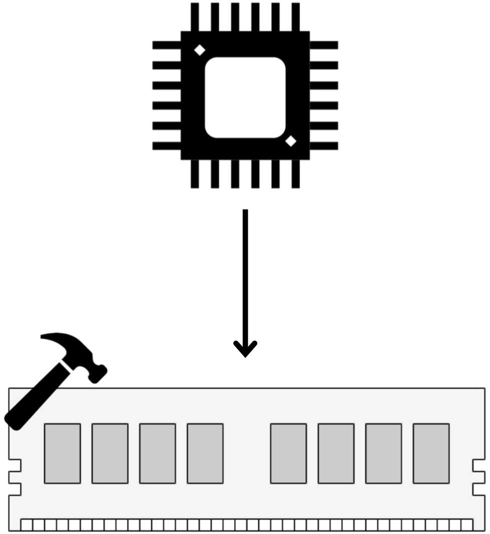
# Attacker primitives

# Attacker primitives

#P1. Fast memory access

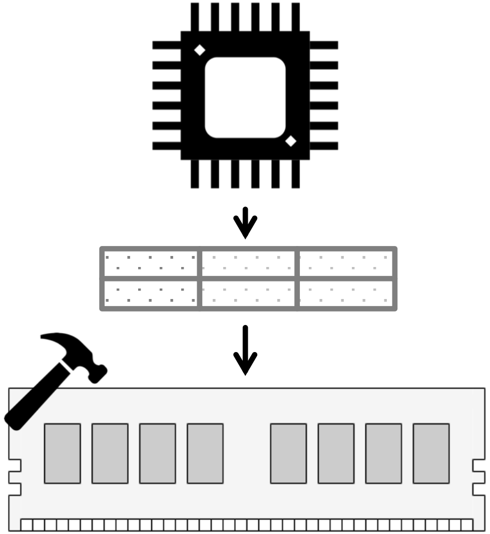
# Attacker primitives

#P1. Fast memory access



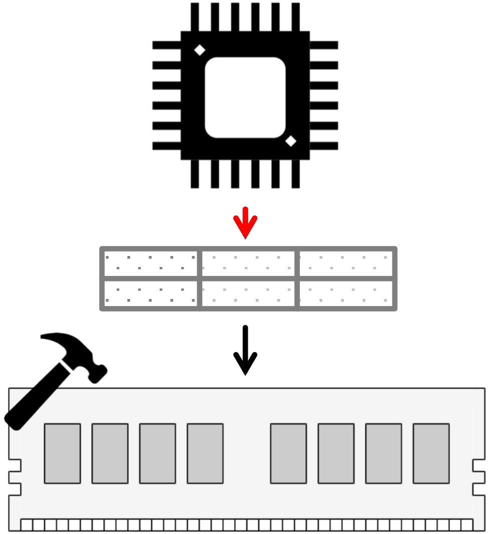
# Attacker primitives

#P1. Fast memory access



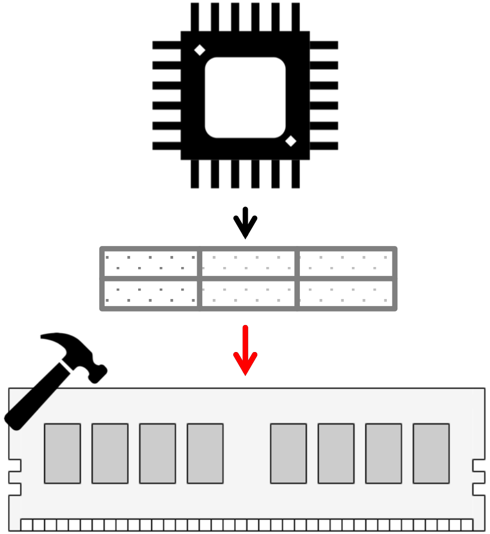
# Attacker primitives

#P1. Fast memory access



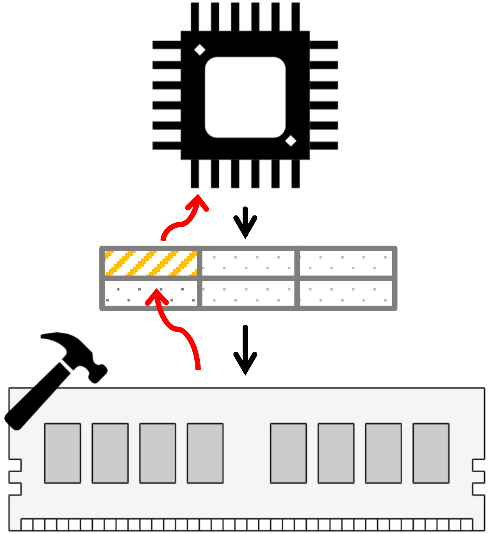
# Attacker primitives

#P1. Fast memory access



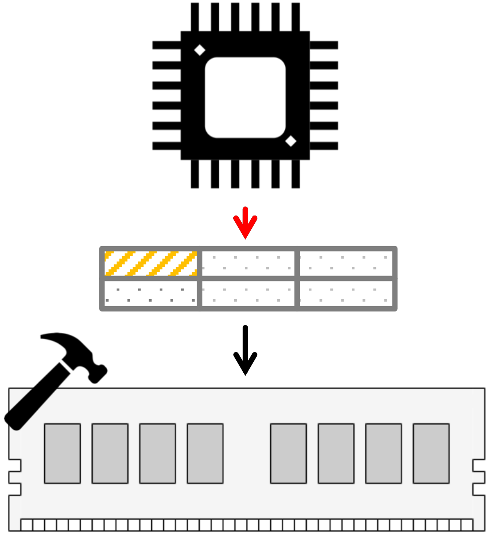
# Attacker primitives

#P1. Fast memory access



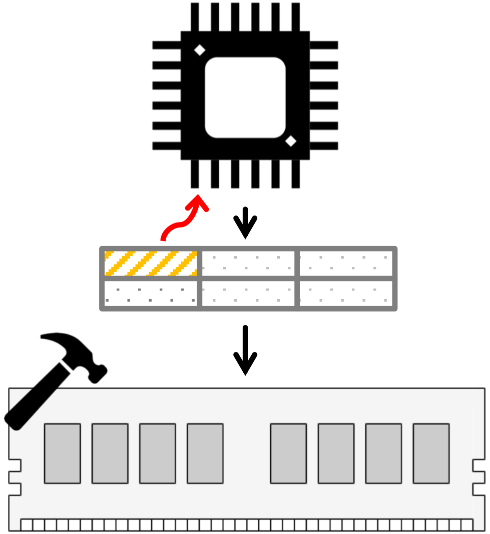
# Attacker primitives

#P1. Fast memory access



# Attacker primitives

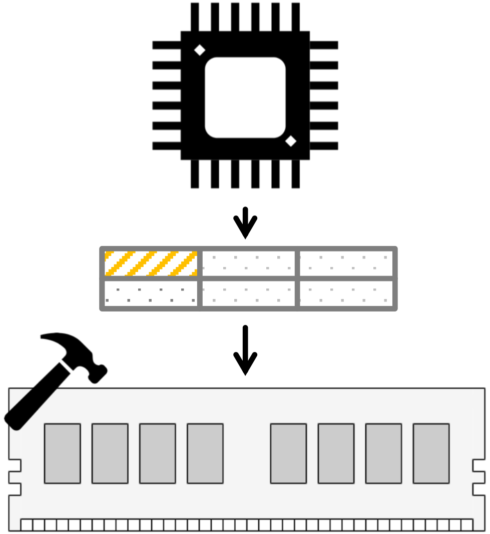
#P1. Fast memory access



# Attacker primitives

#P1. Fast memory access

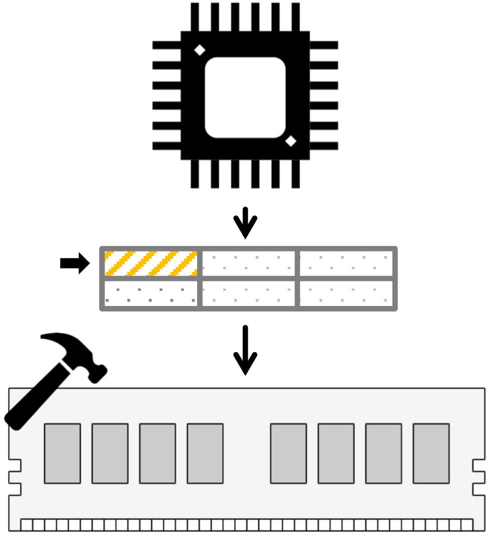
- clflush (native)



# Attacker primitives

#P1. Fast memory access

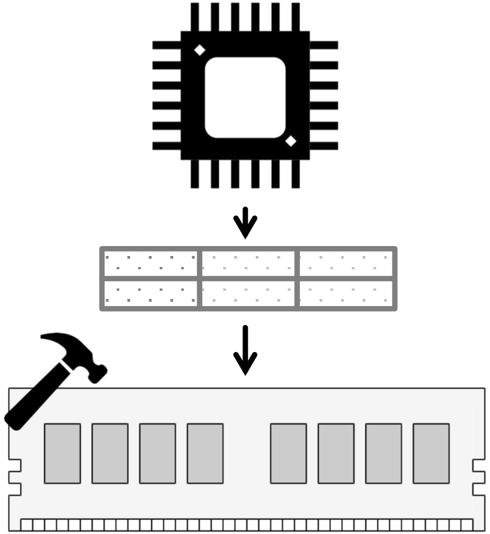
- clflush (native)



# Attacker primitives

#P1. Fast memory access

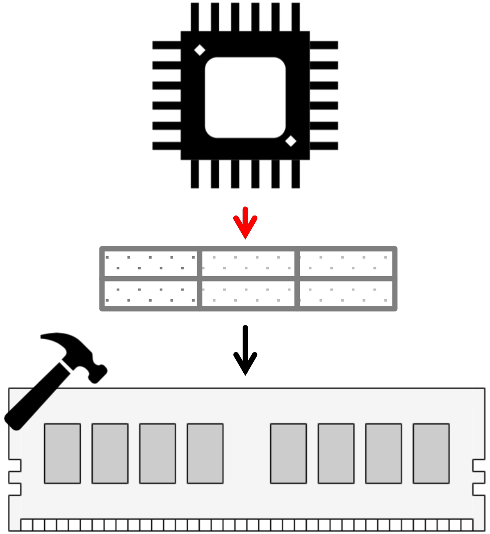
- clflush (native)



# Attacker primitives

#P1. Fast memory access

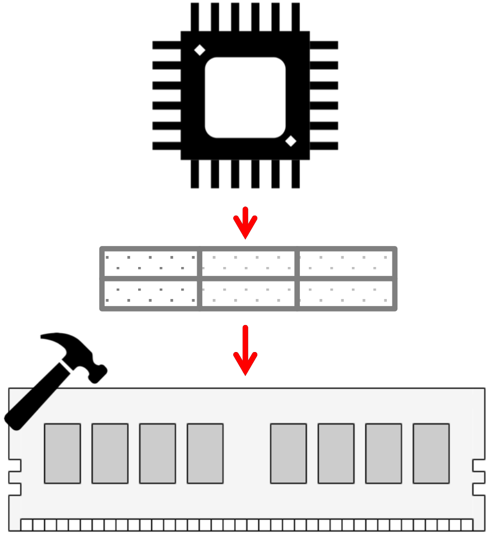
- clflush (native)



# Attacker primitives

#P1. Fast memory access

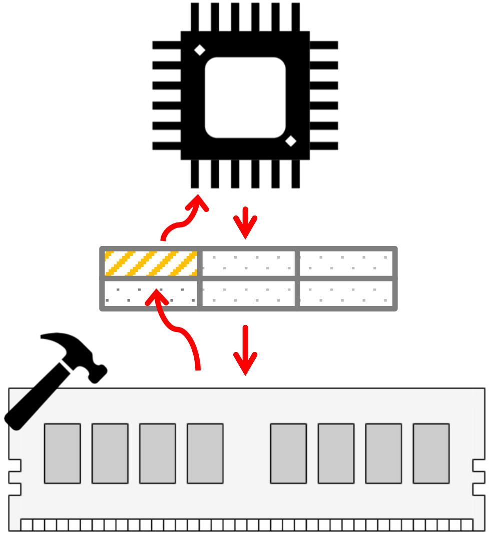
- clflush (native)



# Attacker primitives

#P1. Fast memory access

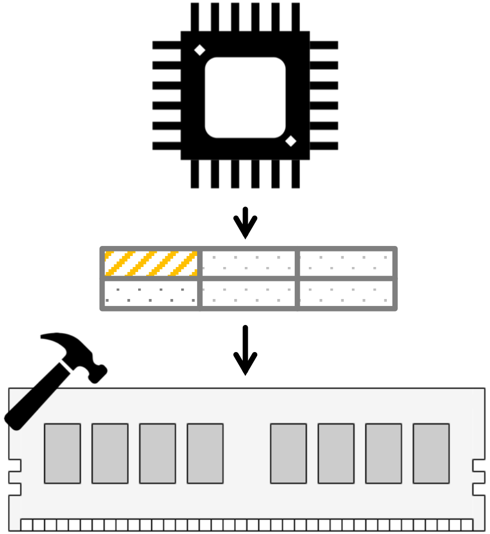
- clflush (native)



# Attacker primitives

#P1. Fast memory access

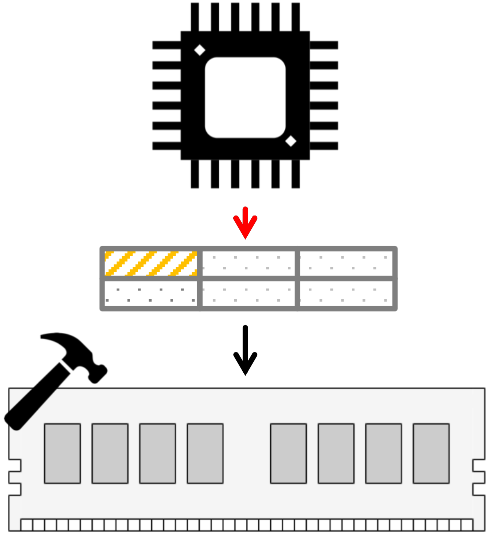
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

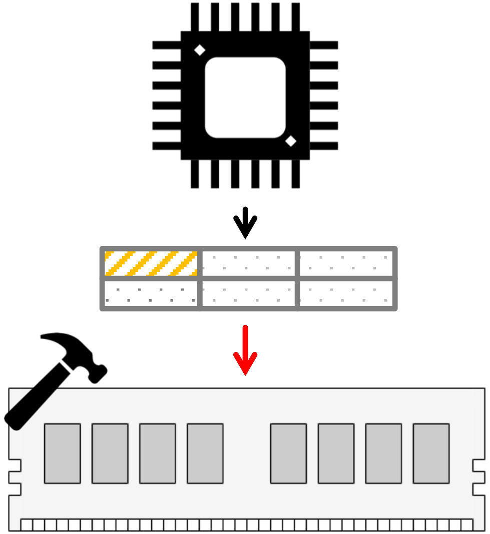
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

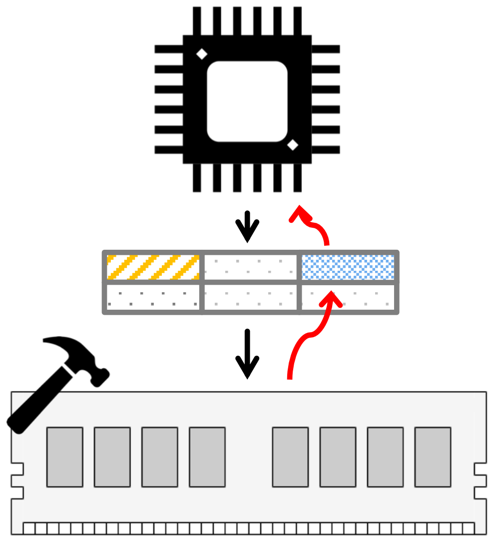
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

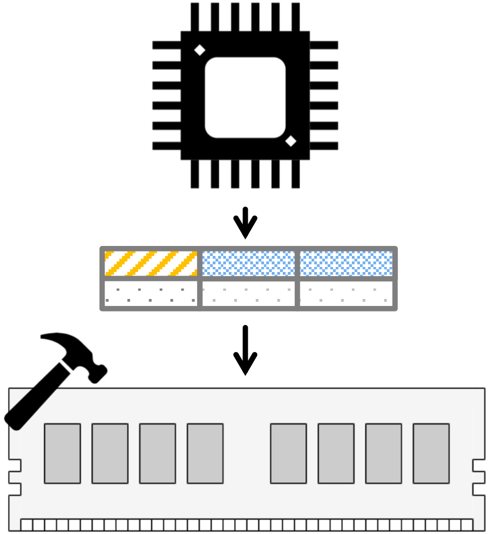
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

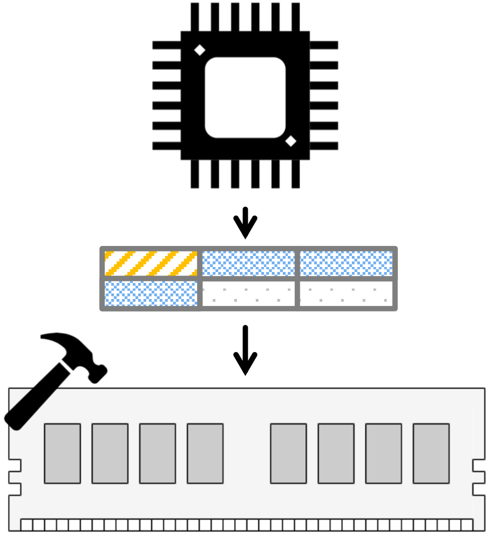
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

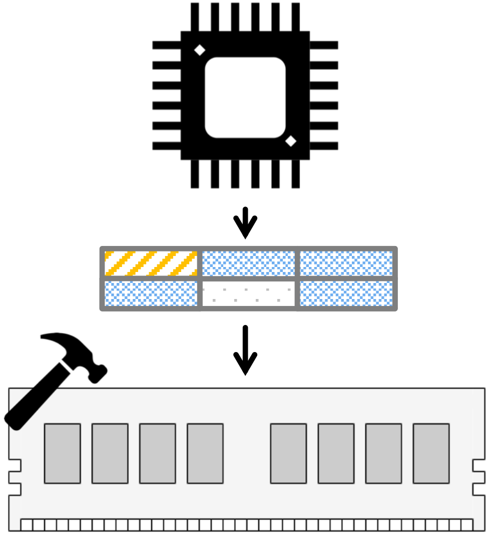
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

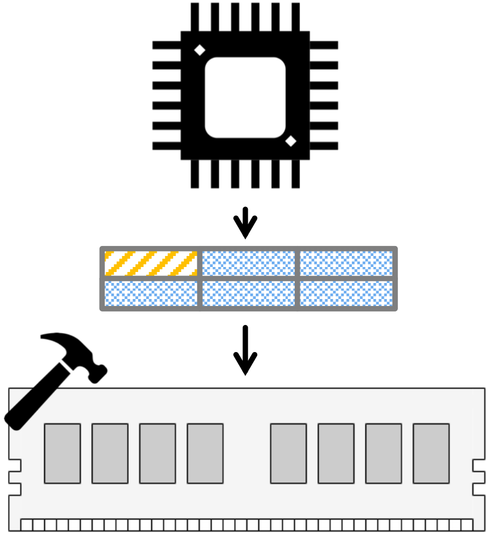
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

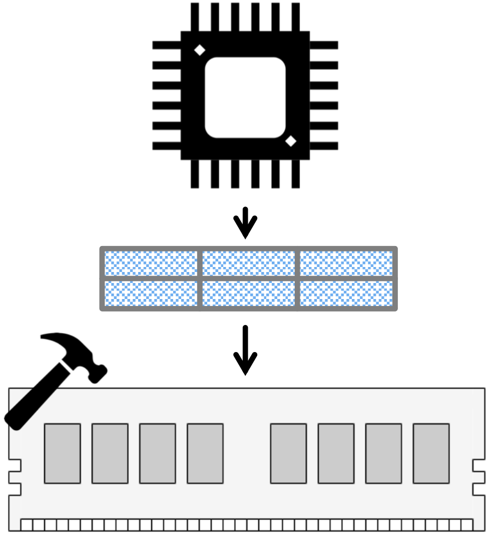
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

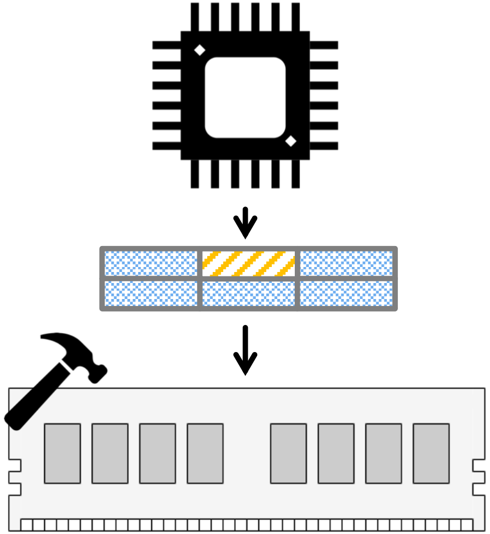
- clflush (native)
- eviction sets (JS)



# Attacker primitives

#P1. Fast memory access

- clflush (native)
- eviction sets (JS)

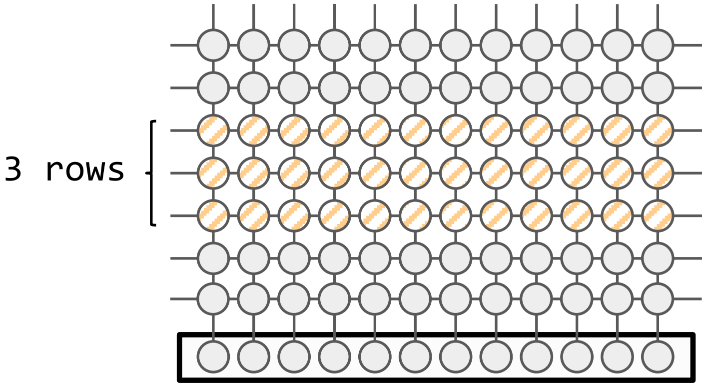


# Attacker primitives

#P1. Fast memory access

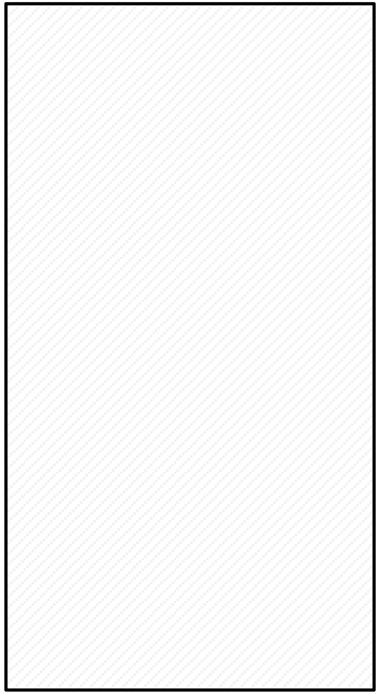
- clflush (native)
- eviction sets (JS)

#P2. Contiguous memory



# Address translation

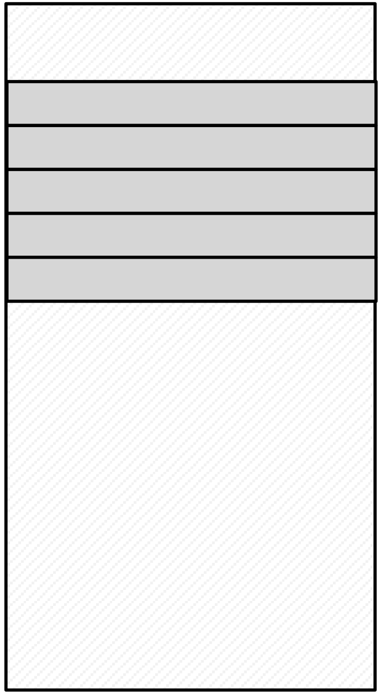
Virtual Memory



```c
char* buffer = malloc(sizeof(char)*KB(20));
```

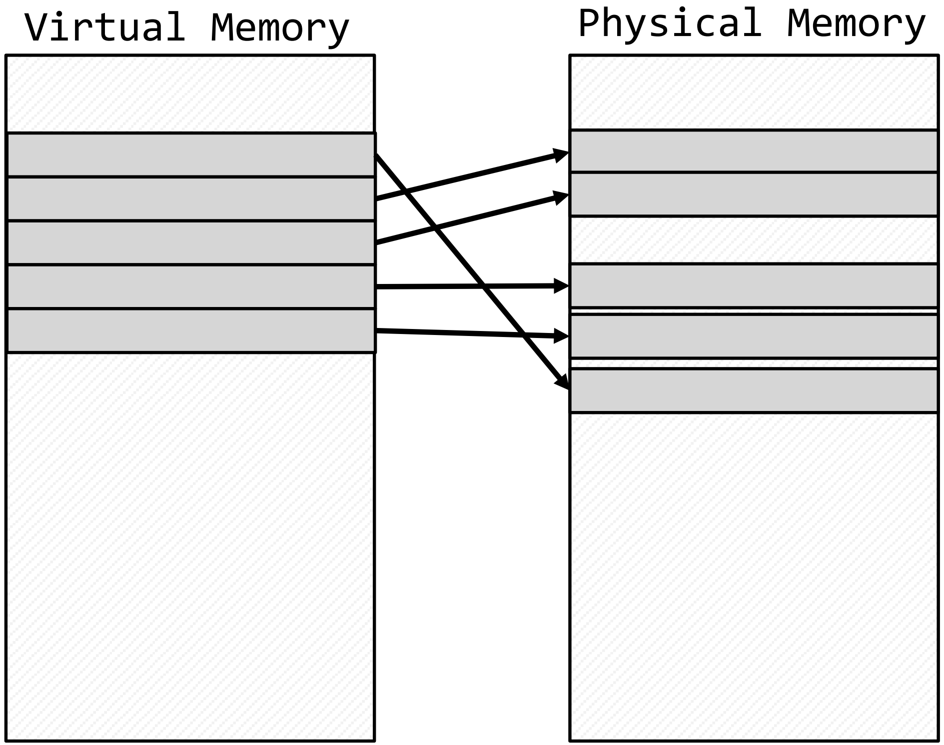
# Address translation

Virtual Memory

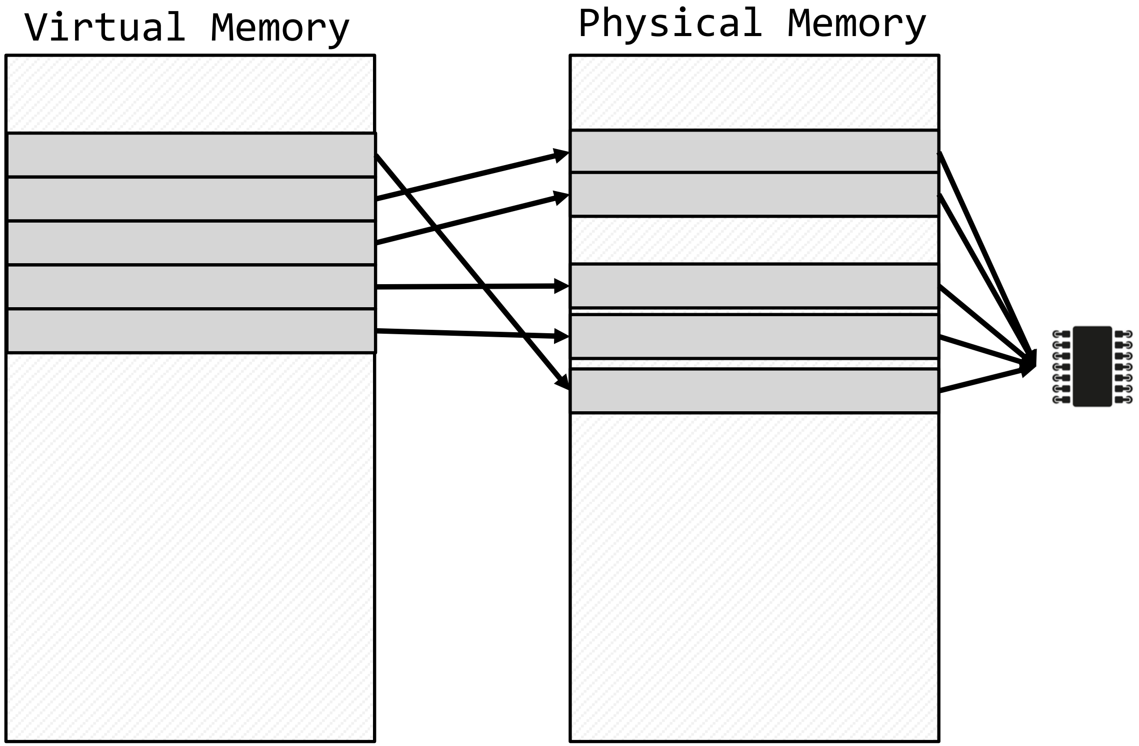


```
char* buffer = malloc(sizeof(char)*KB(20));
```

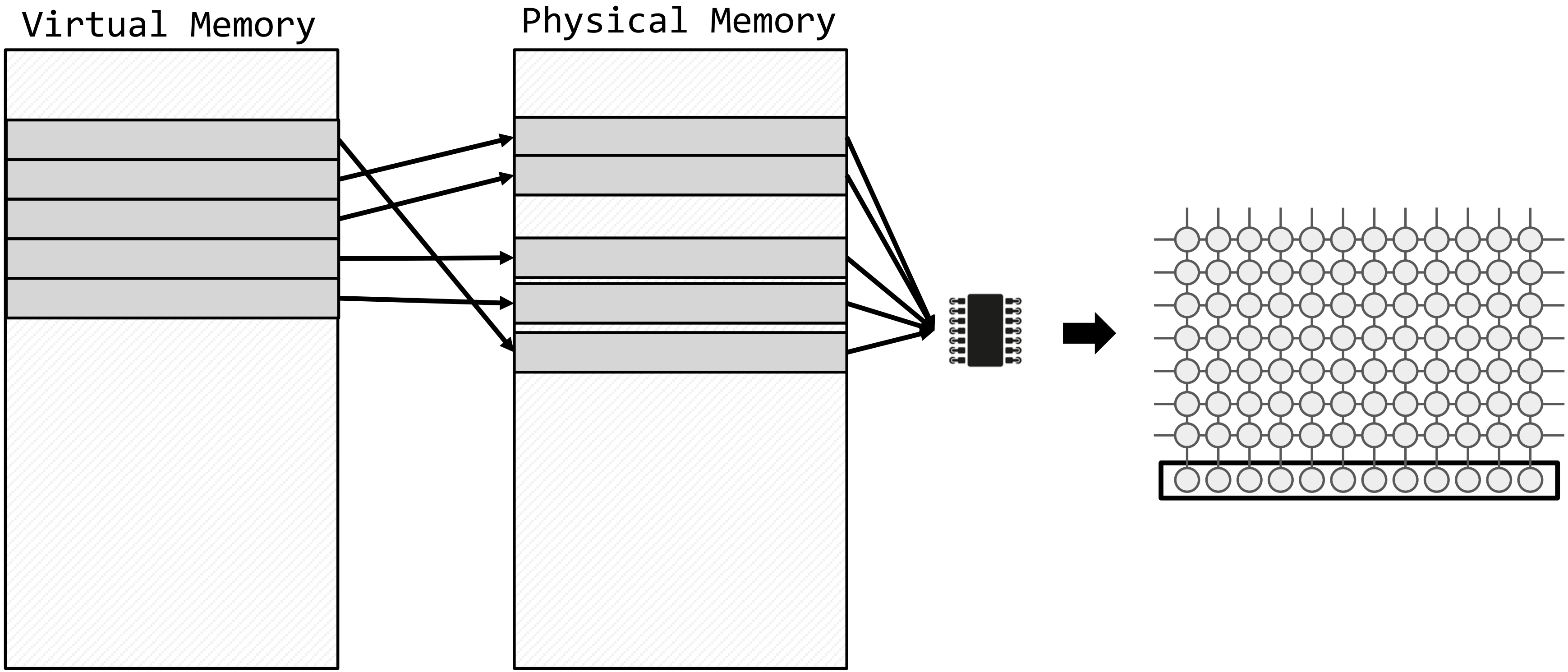
# Address translation



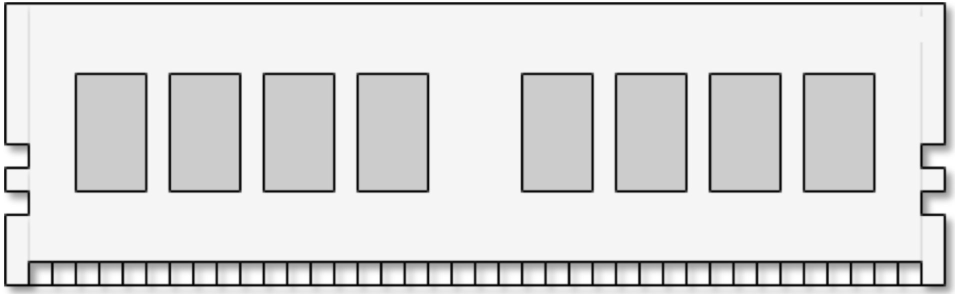
# Address translation



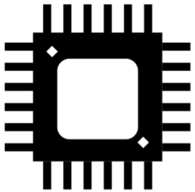
# Address translation



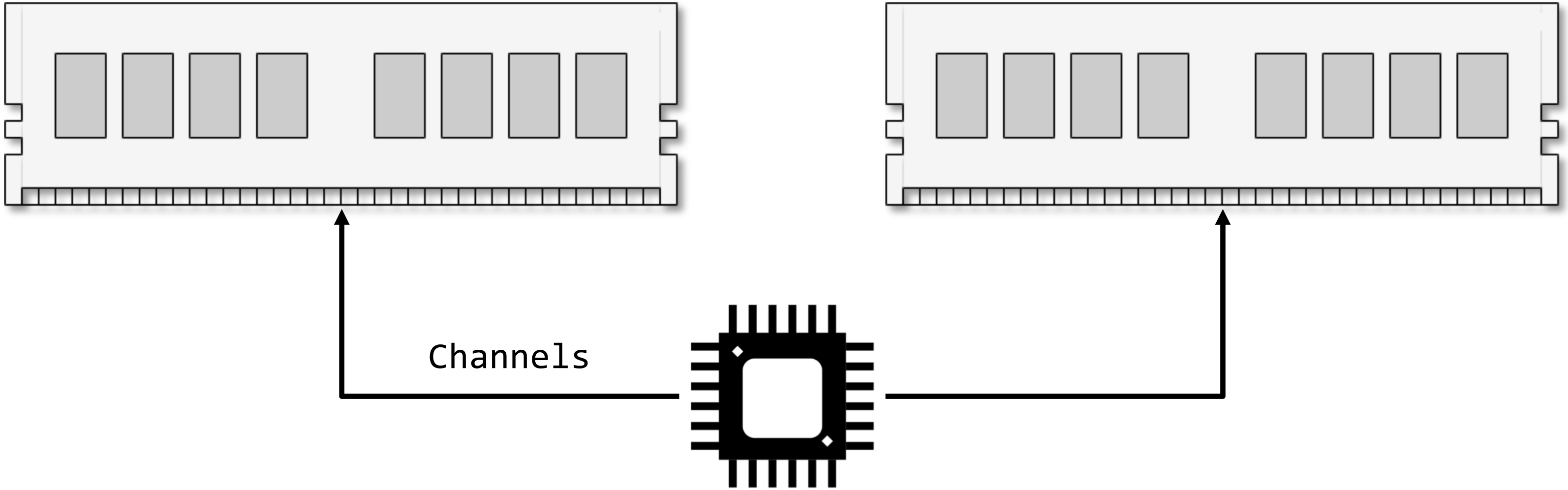
# DRAM



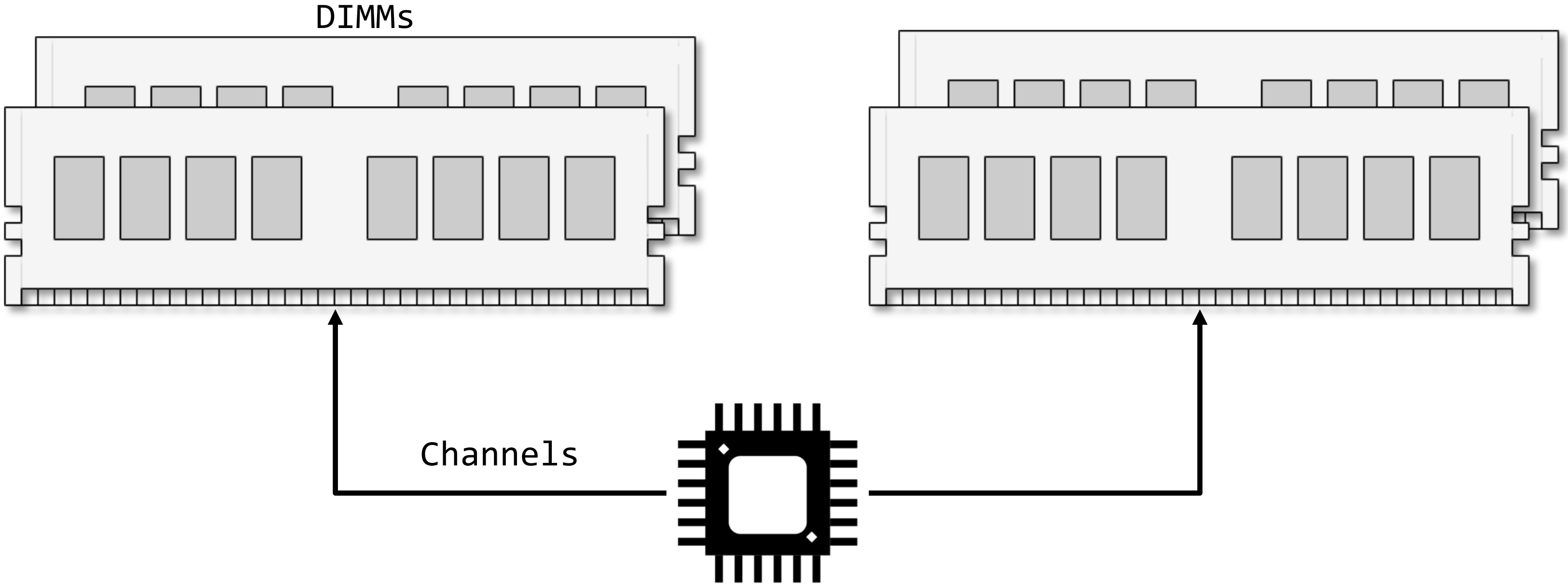
# 🔨DRAM: organization



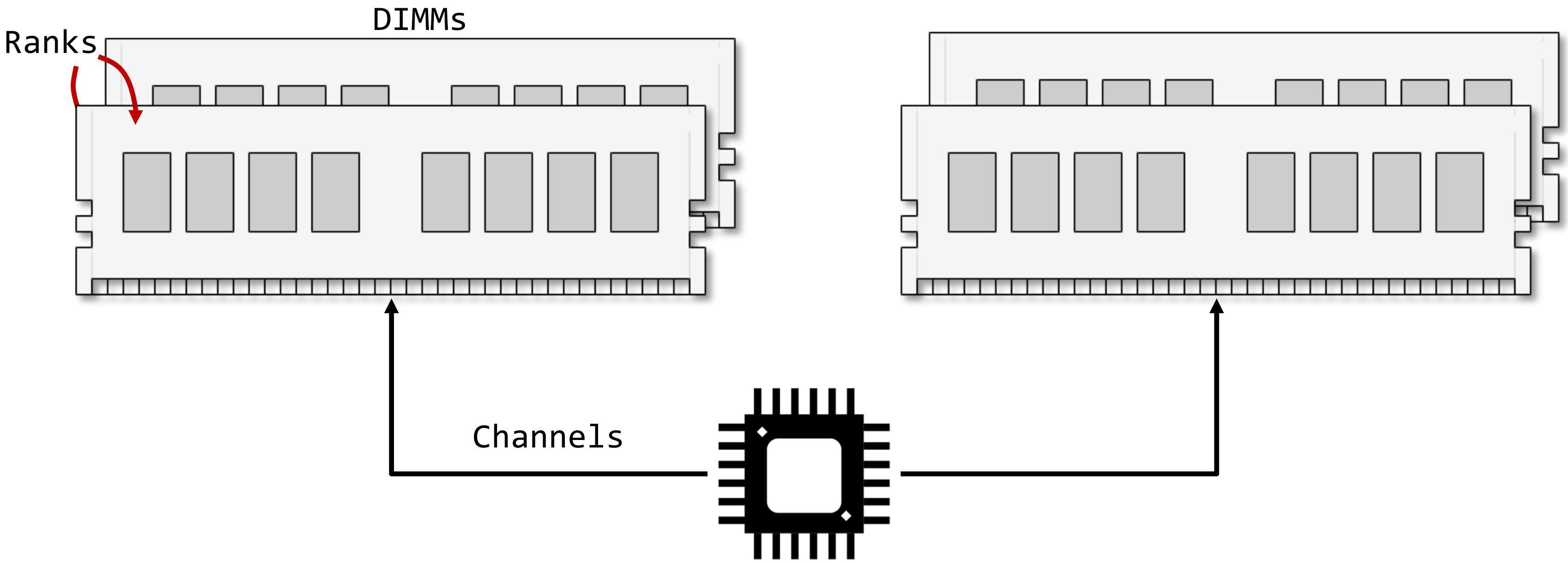
# DRAM: organization



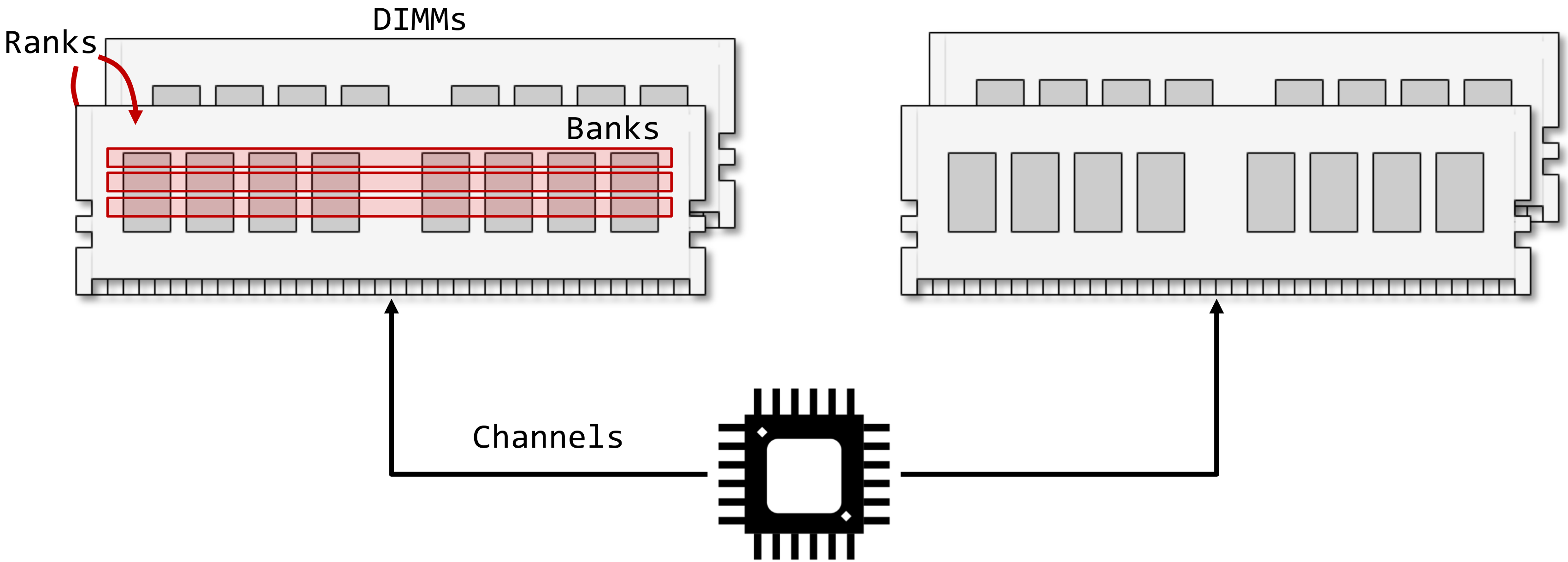
# DRAM: organization



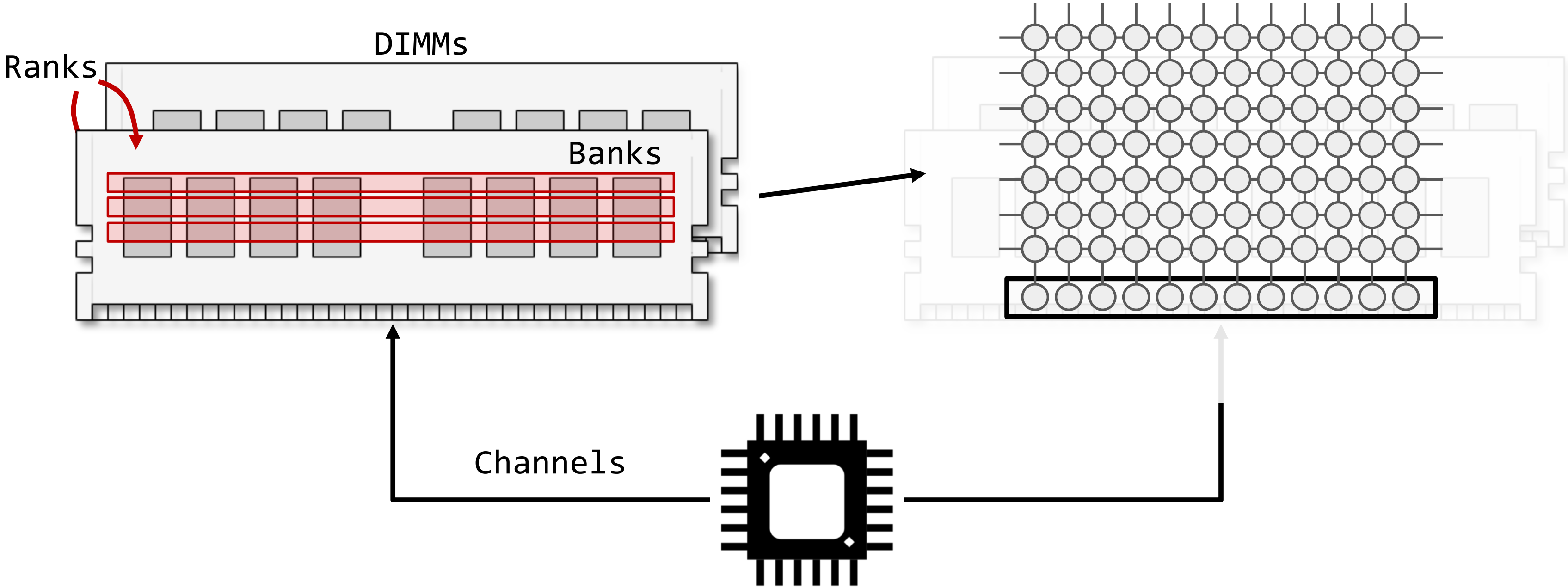
# DRAM: organization



# DRAM: organization

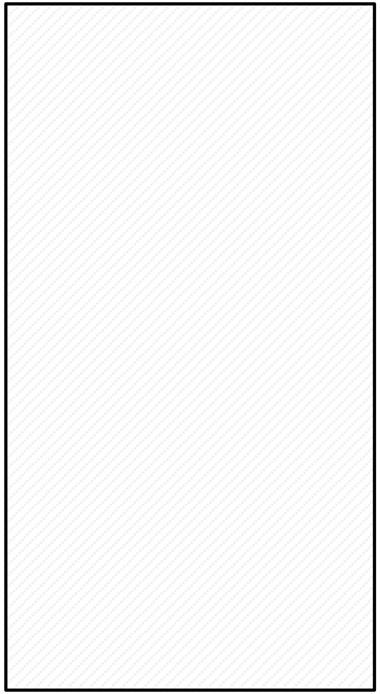


# DRAM: organization



# Address translation: THPs

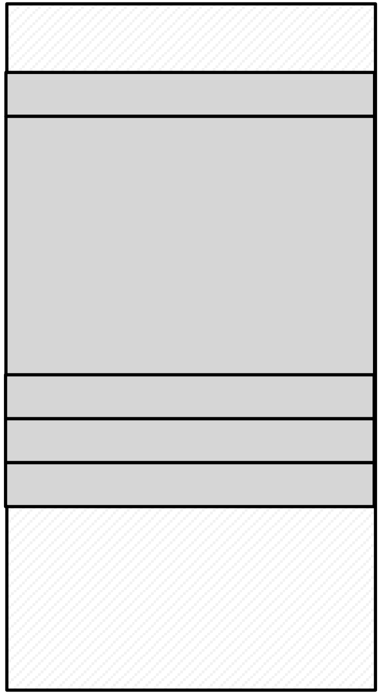
Virtual Memory



```
char* buffer = malloc(sizeof(char)*MB(3));
```

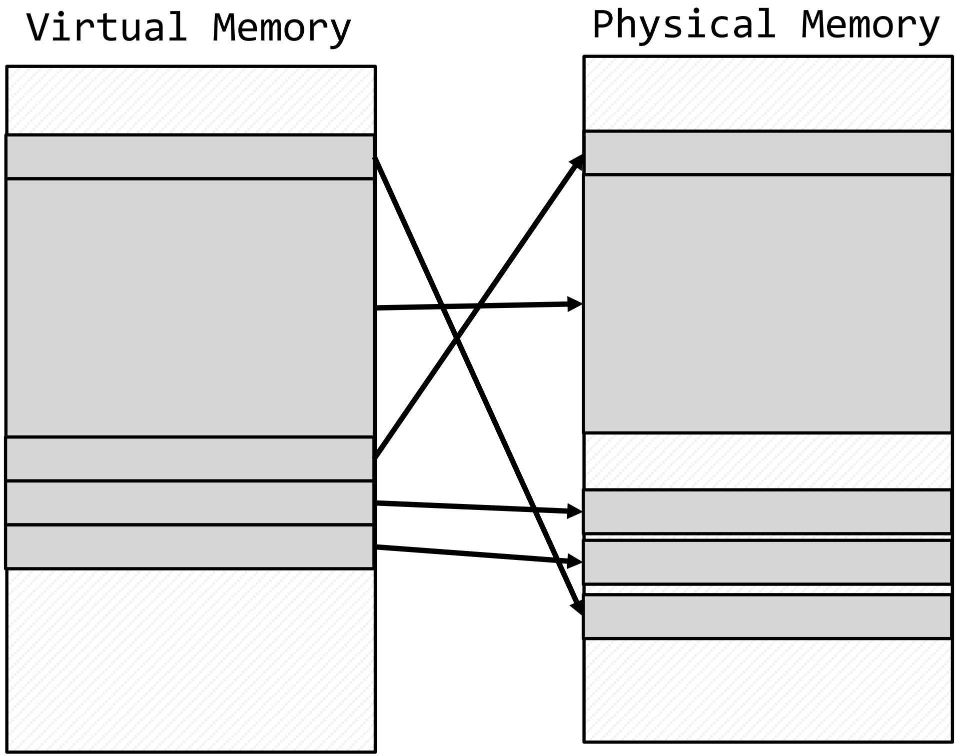
# Address translation: THPs

Virtual Memory

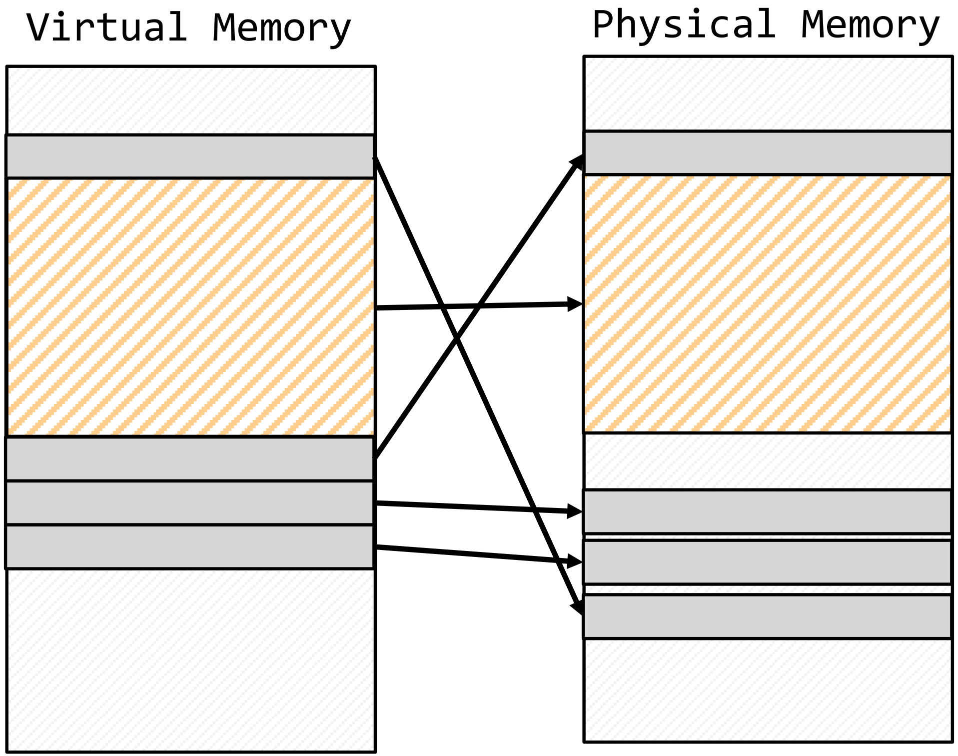


```
char* buffer = malloc(sizeof(char)*MB(3));
```

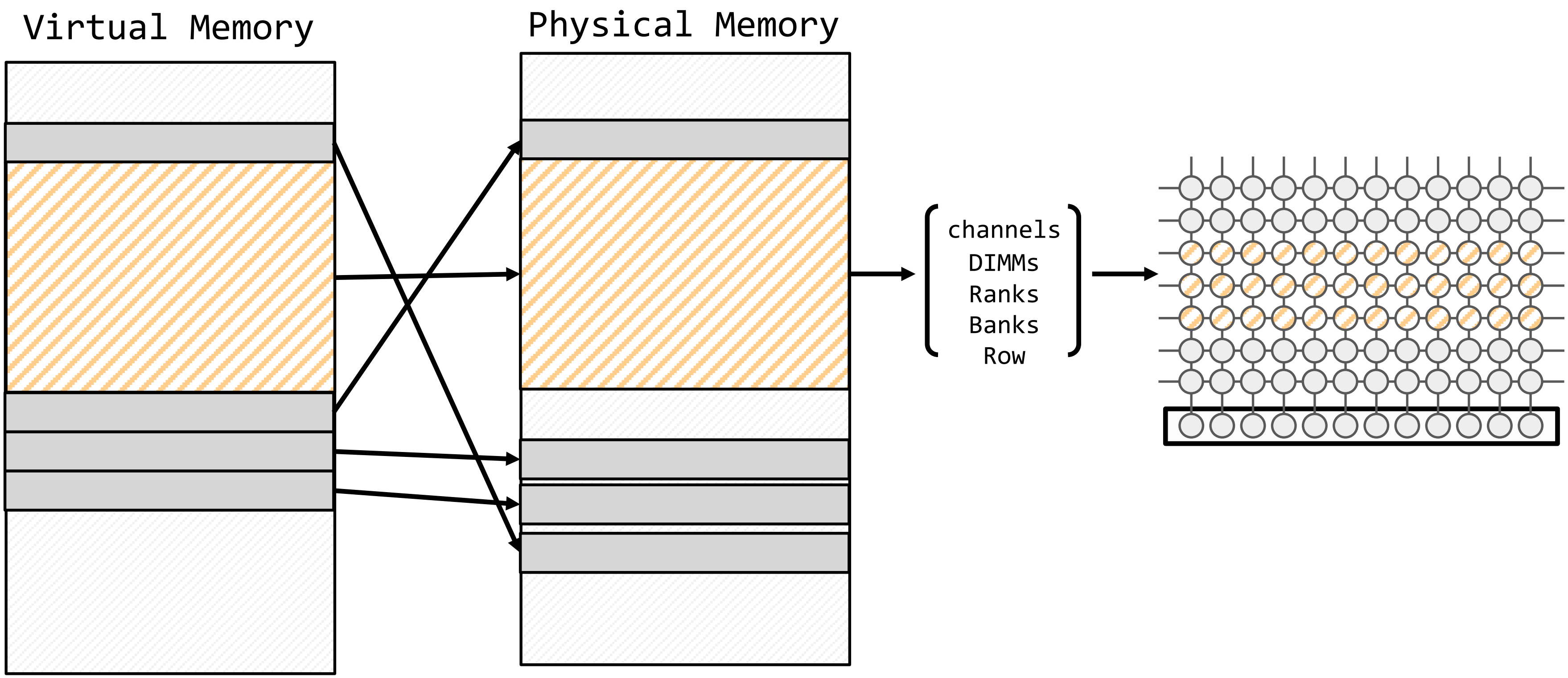
# Address translation: THPs



# Address translation: THPs



# Address translation: THPs



# Attacker primitives

#P1. Fast memory access

- clflush (native)
- eviction sets (JS)

#P2. Contiguous memory

- THPs (native and JS)

arm

# Attacker primitives

#P1. Fast memory access

- ✗ clflush (native)
- eviction sets (JS)

#P2. Contiguous memory

- THPs (native and JS)

# Attacker primitives

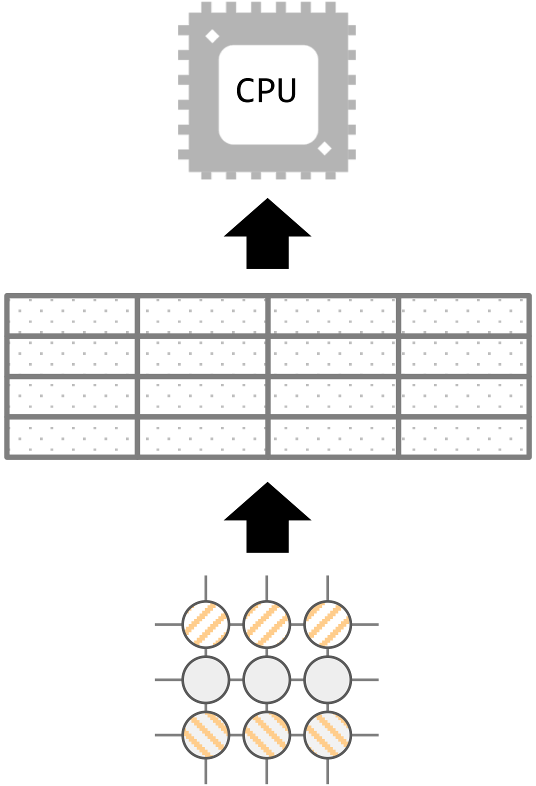
#P1. Fast memory access

- ✗ clflush (native)
- ✗ eviction sets (JS)

#P2. Contiguous memory

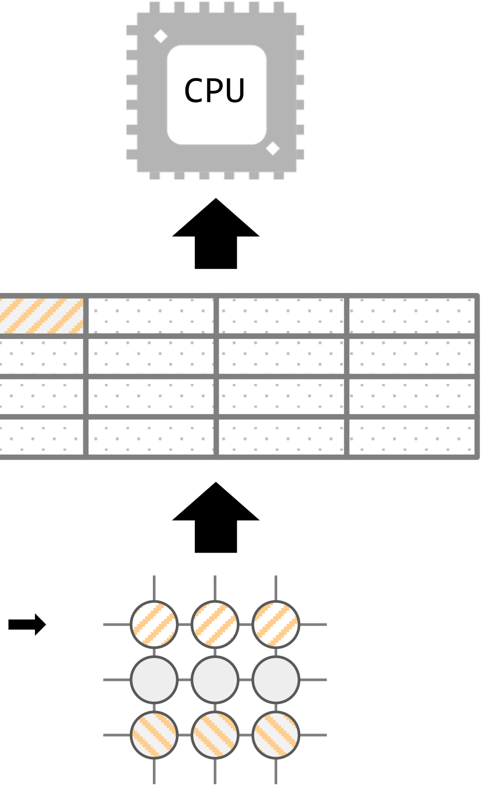
- THPs (native and JS)

# #P2. Eviction-based Rowhammer: arm




Caches: Large & random

# #P2. Eviction-based Rowhammer:

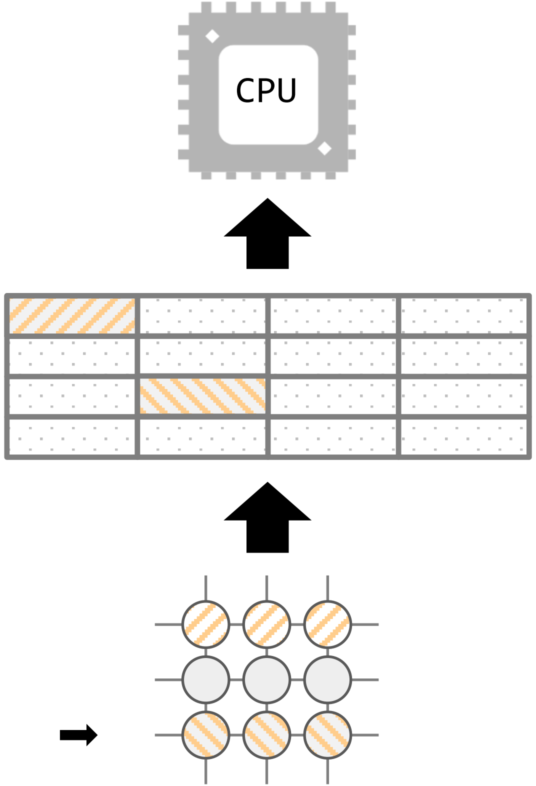



Caches: Large & random

Steps:

1. Read row *n-1*

# #P2. Eviction-based Rowhammer: arm

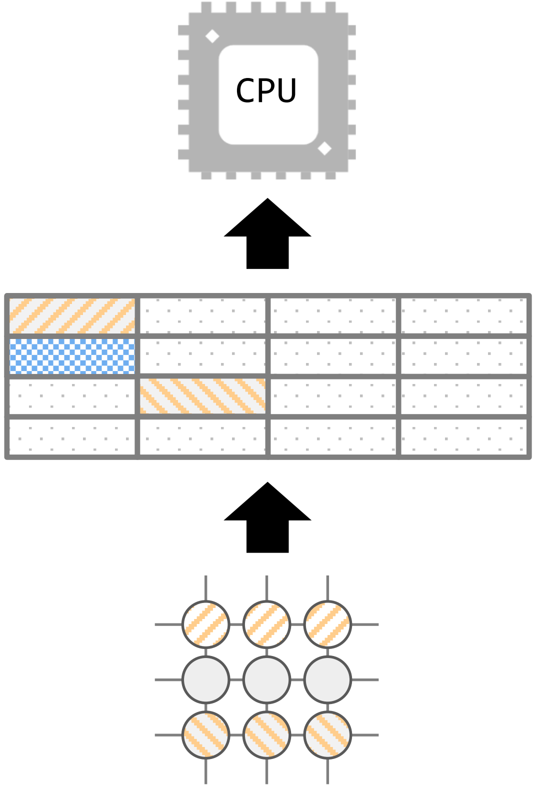



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*

# #P2. Eviction-based Rowhammer: arm

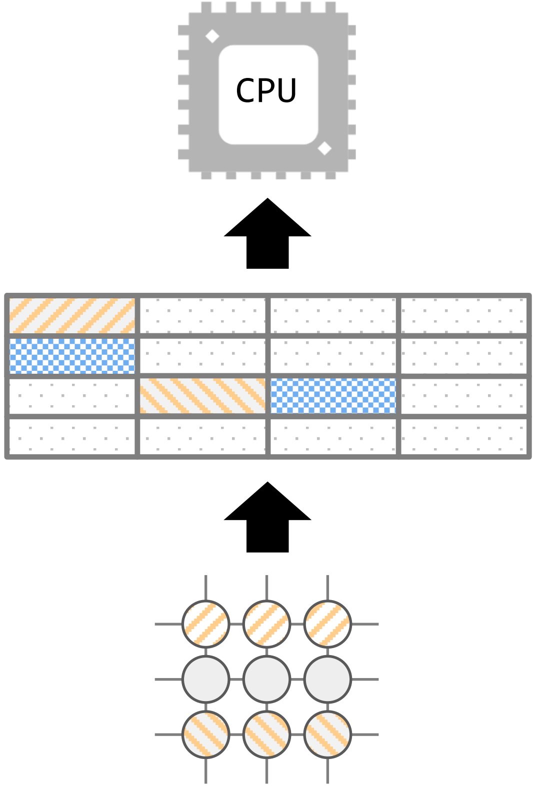



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: arm

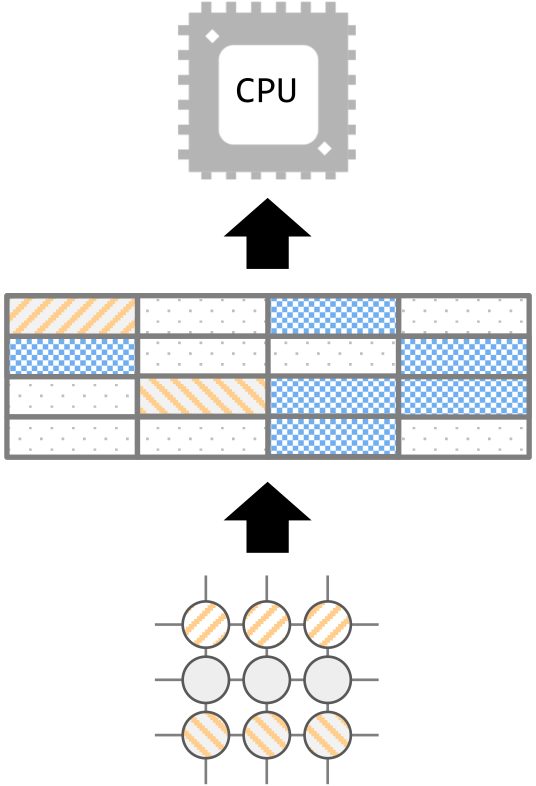



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: arm

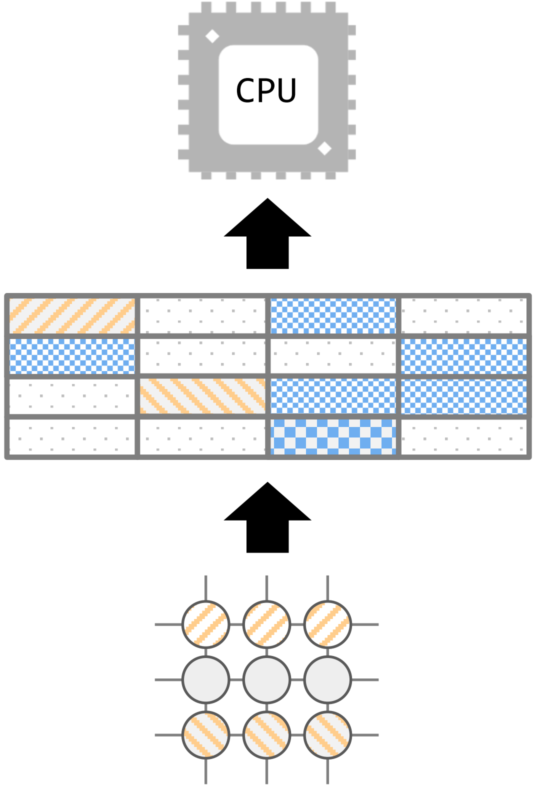



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: arm

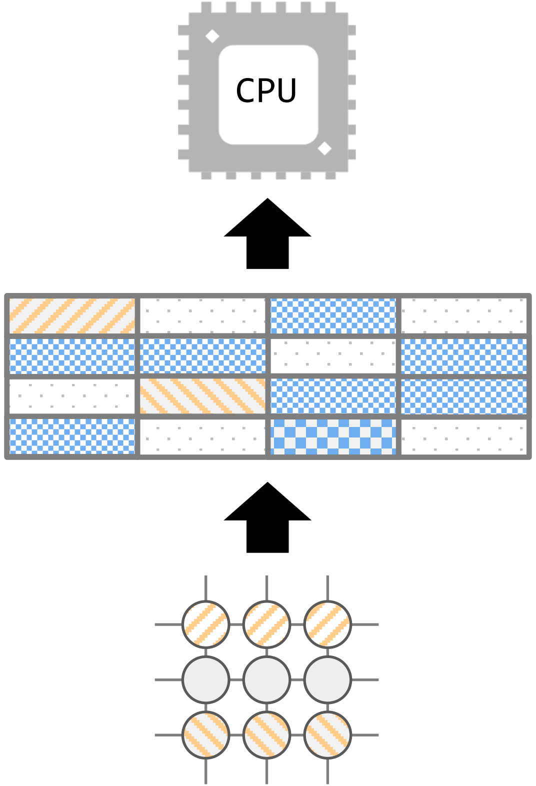



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: arm

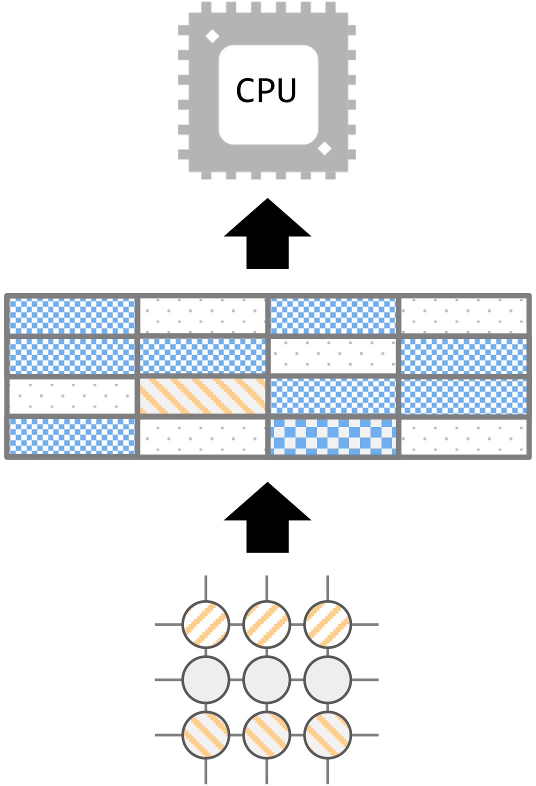



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: arm

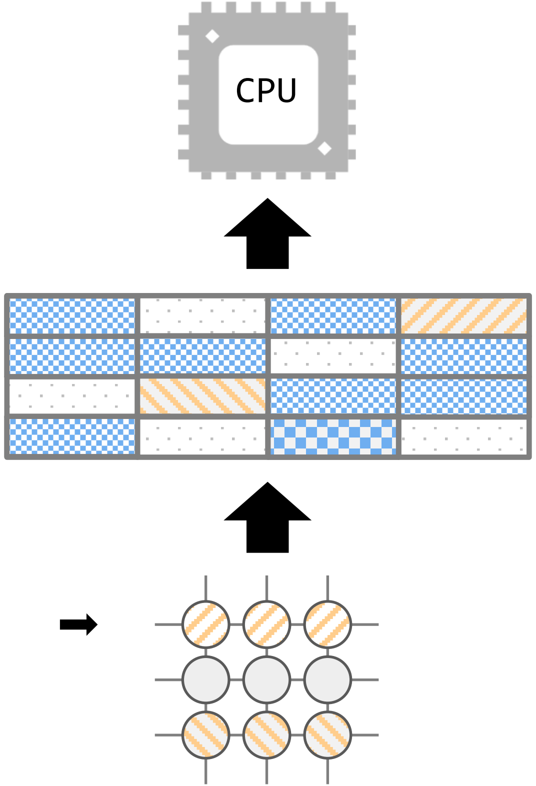



Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: arm




Caches: Large & random

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++
4. Read row n-1

   ...

# Attacker primitives

#P1. Fast memory access

- ✗ clflush (native)
- ✗ eviction sets (JS)

#P2. Contiguous memory

- ✗ THPs (native and JS)

arm

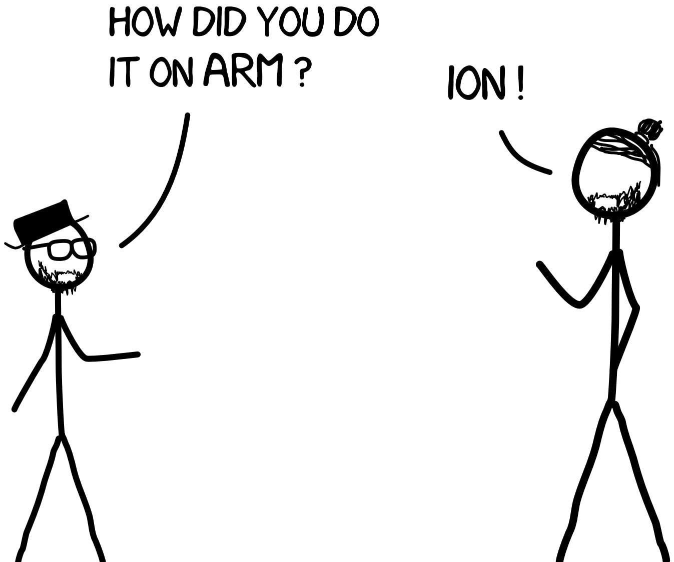
# Attacker primitives

#P1. Fast memory access

- ✗ clflush (native)
- ✗ eviction sets (JS)

#P2. Contiguous memory

- ✗ THPs (native and JS)



# Attacker primitives

#P1. Fast memory access

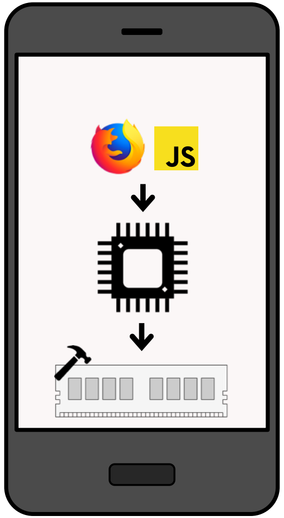
- ✗ c
- ✗ e

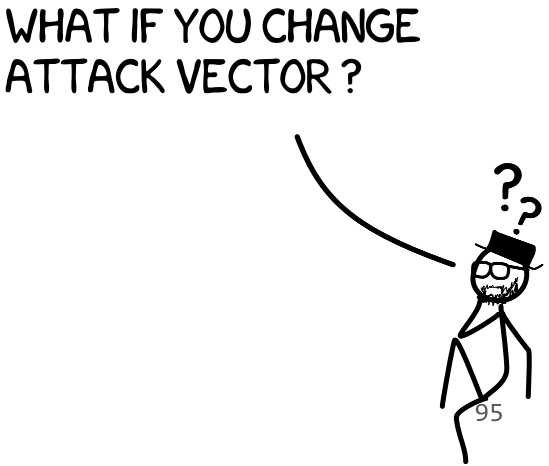
#P2. Conti

- ✗ THPs (native and JS)

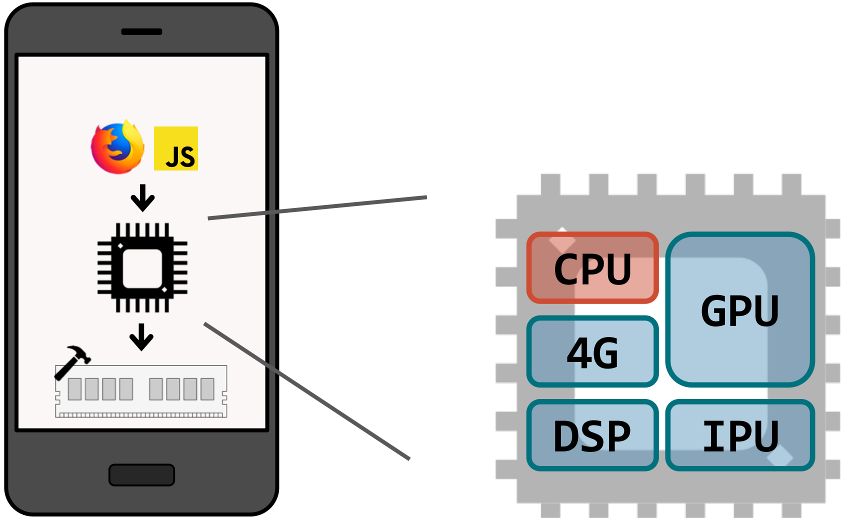

We are screwed.

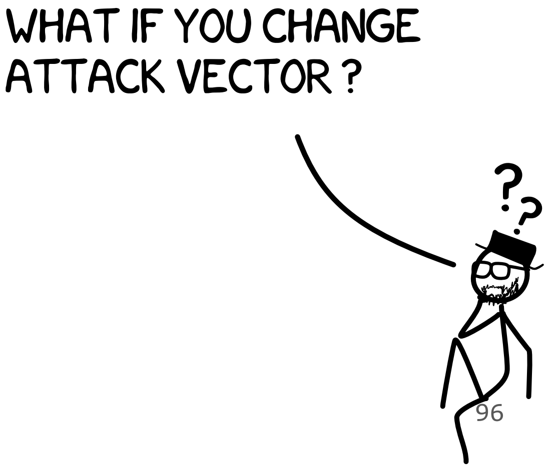
# Attack Vector



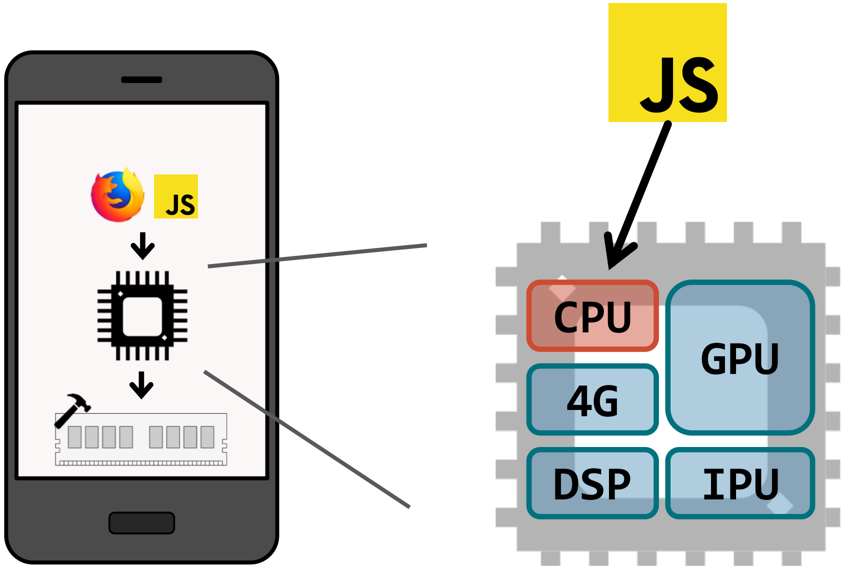



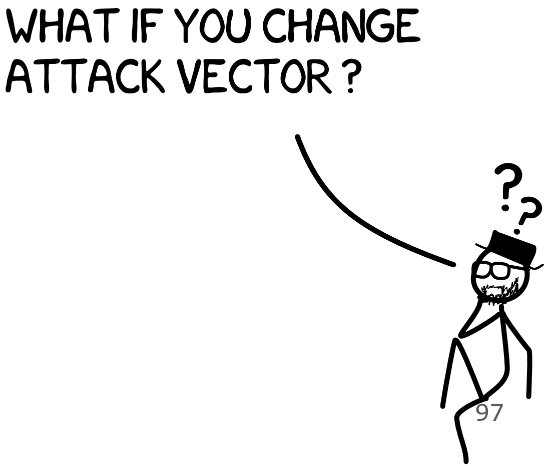
# Attack Vector



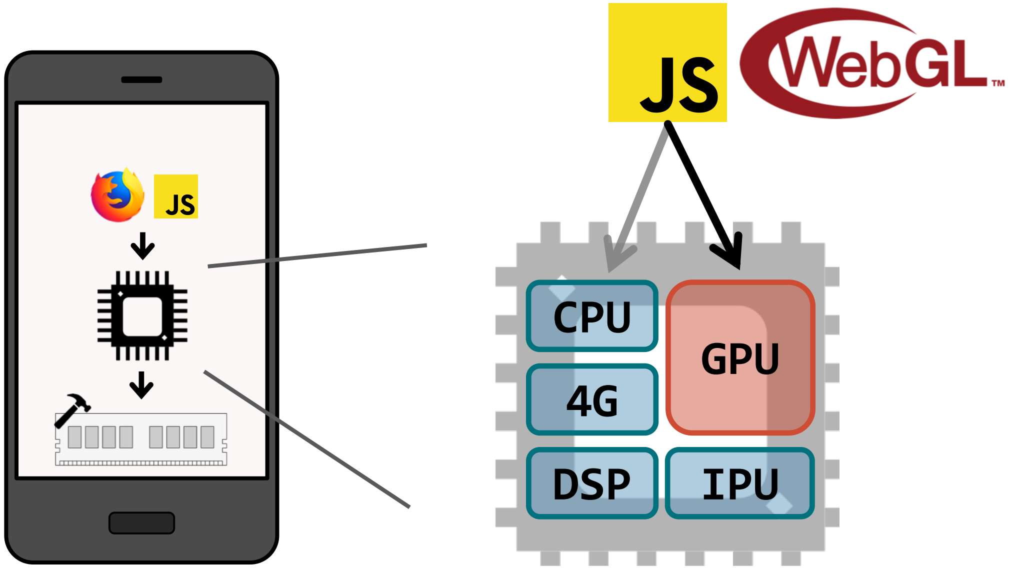



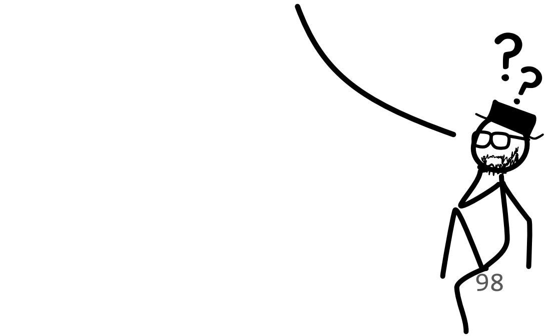
# Attack Vector



# Attack Vector



# Attacker primitives

#P1. Fast memory access

#P2. Contiguous memory

# Attacker primitives

#P1. DRAM Access

#P2. Fast memory access

#P3. Contiguous memory

# Attacker primitives

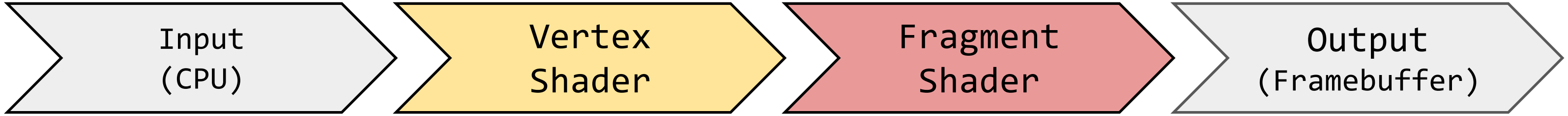
#P1. DRAM Access

#P2. Fast memory access

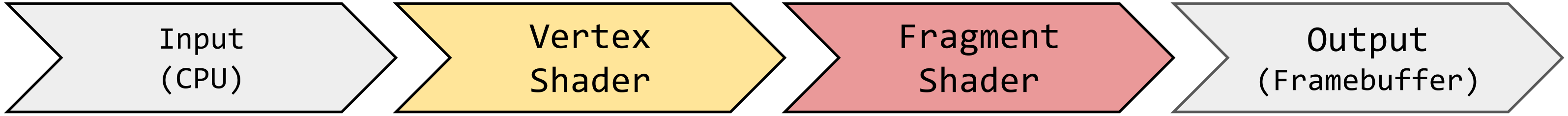
#P3. Contiguous memory

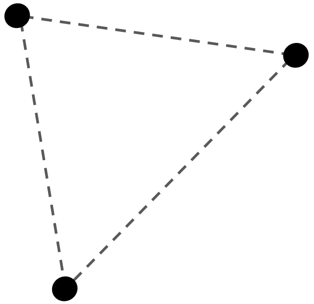
# Understanding the GPU

# #P1. GPU: The rendering pipeline



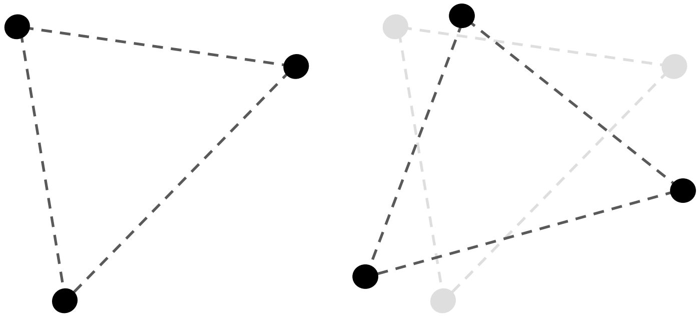
# **#P1.** GPU: The rendering pipeline



# **#P1.** GPU: The rendering pipeline

Input (CPU) → Vertex Shader → Fragment Shader → Output (Framebuffer)



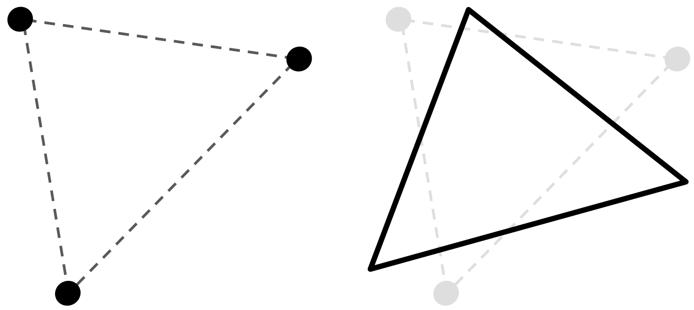
# **#P1.** GPU: The rendering pipeline

Input
(CPU)

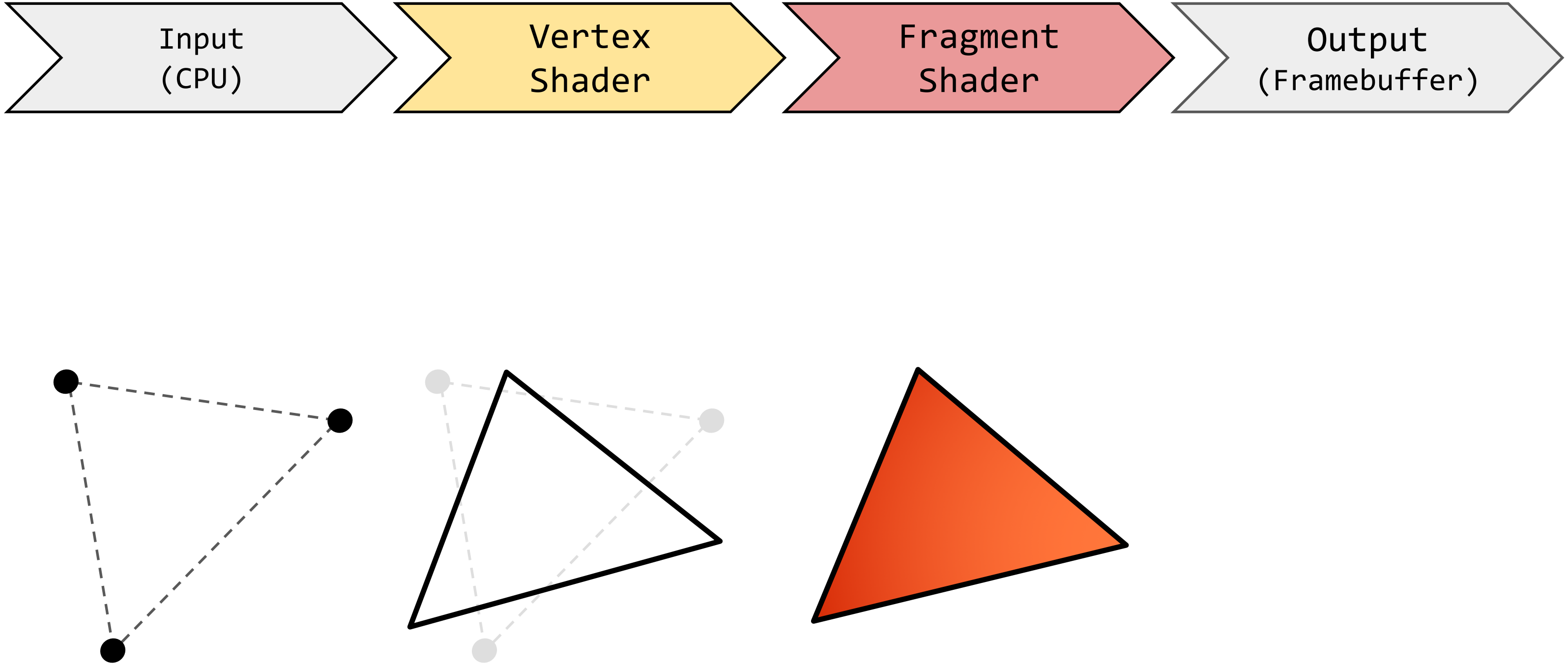
Vertex
Shader

Fragment
Shader

Output
(Framebuffer)

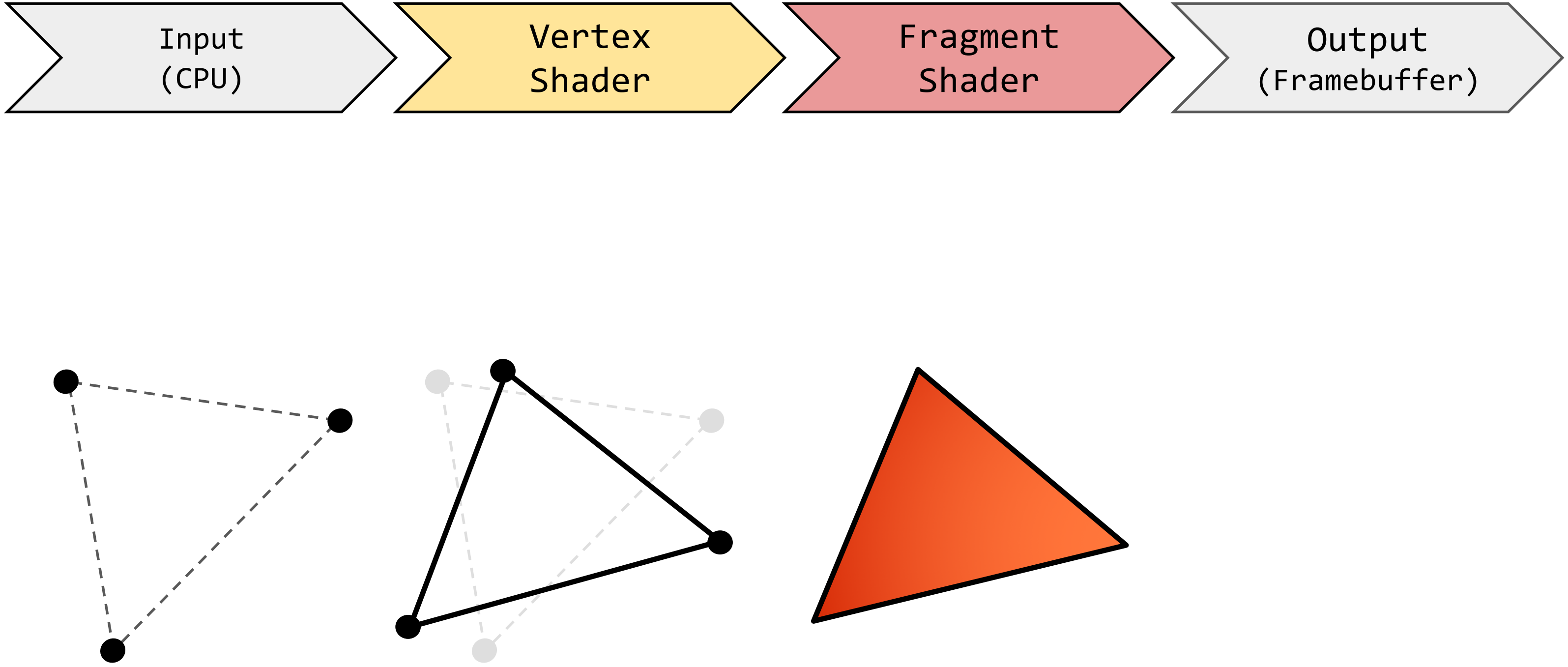


# #P1. GPU: The rendering pipeline

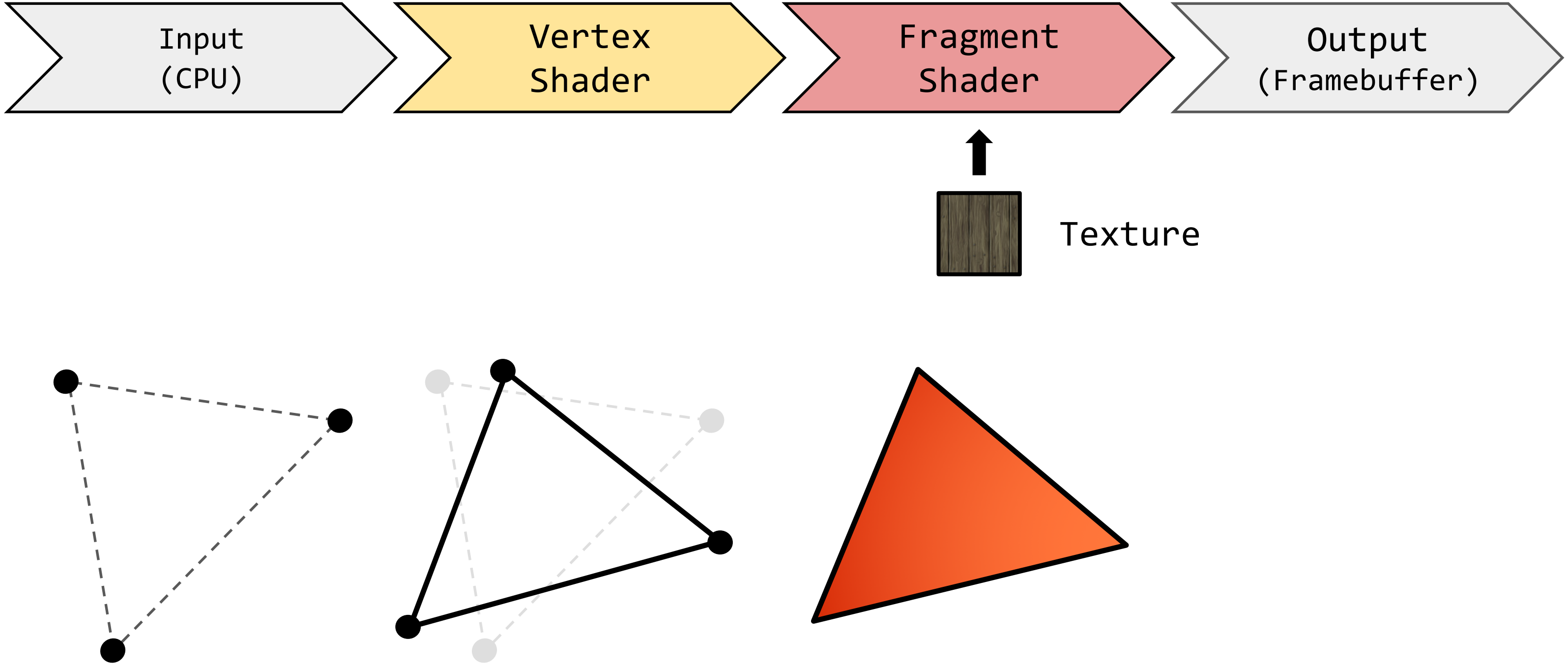



MINECRAFT
EDUCATION EDITION

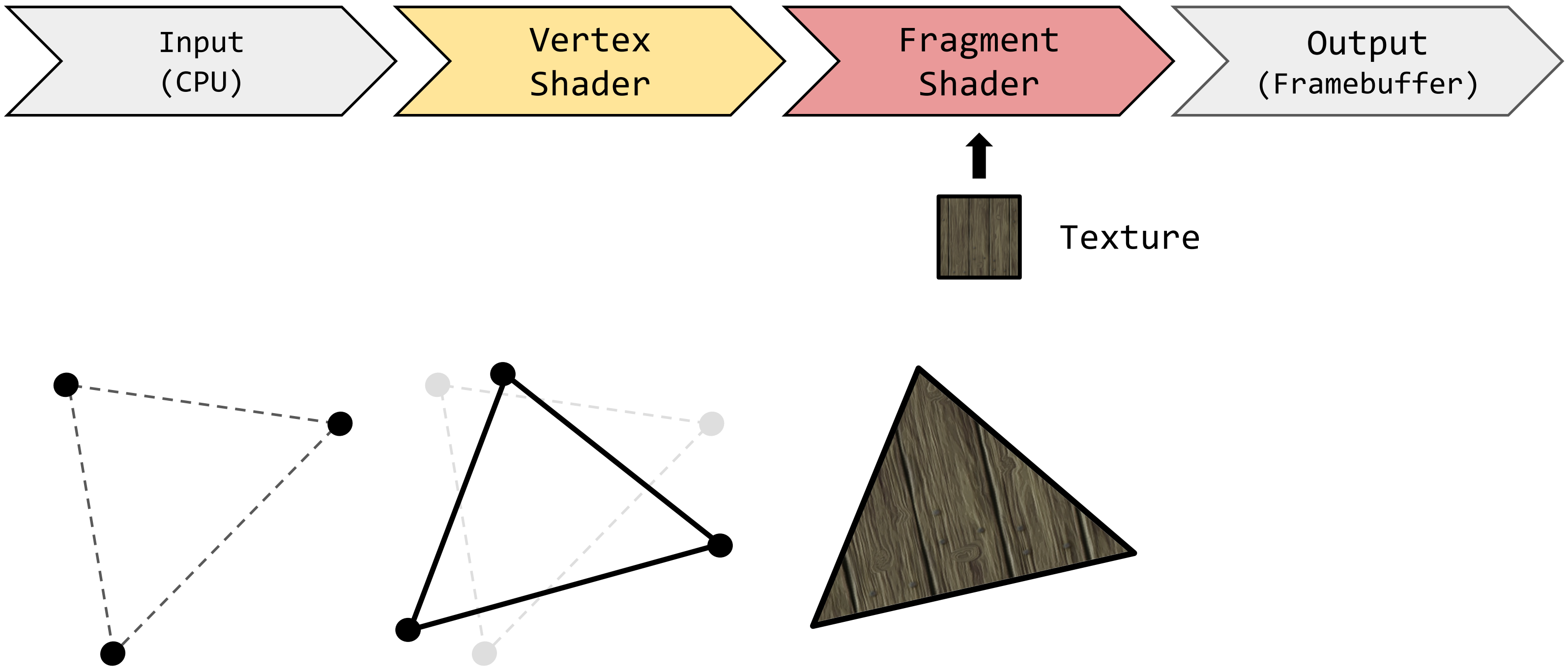
# #P1. GPU: The rendering pipeline



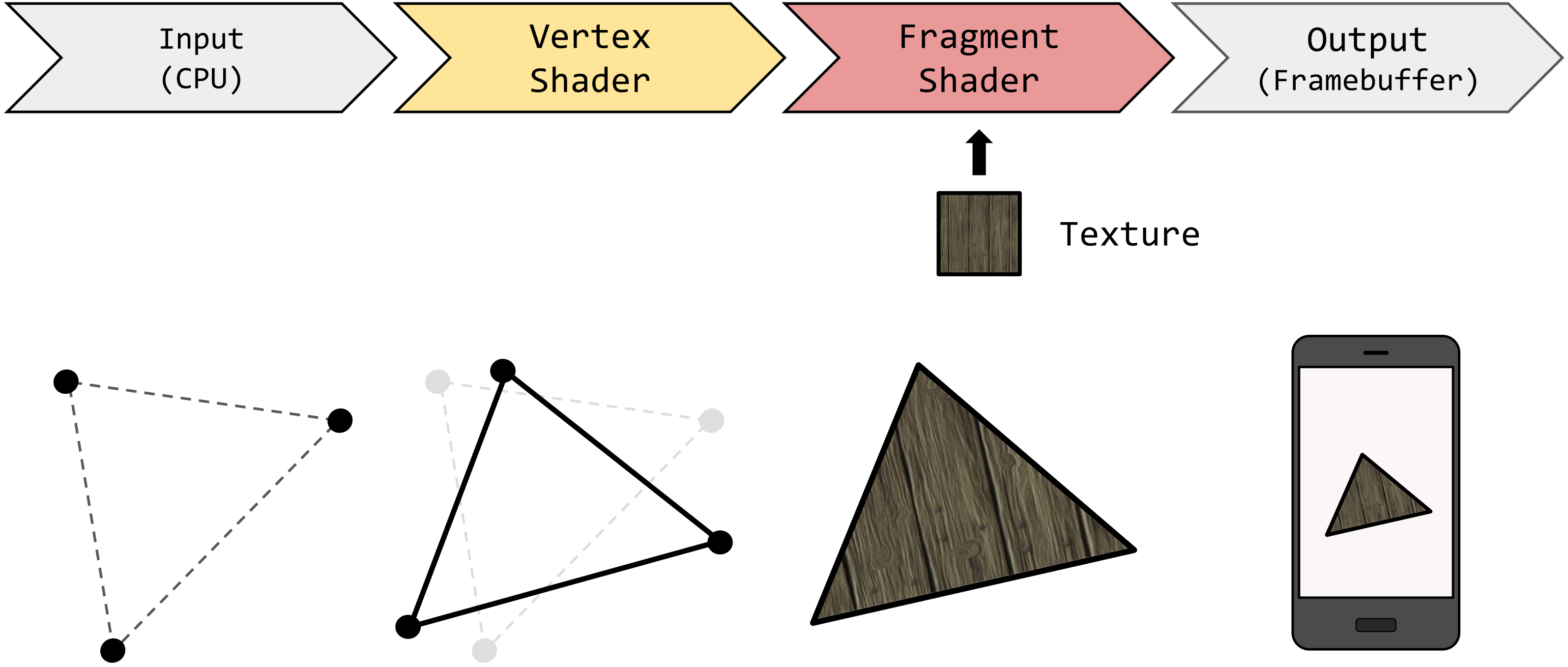
# #P1. GPU: The rendering pipeline



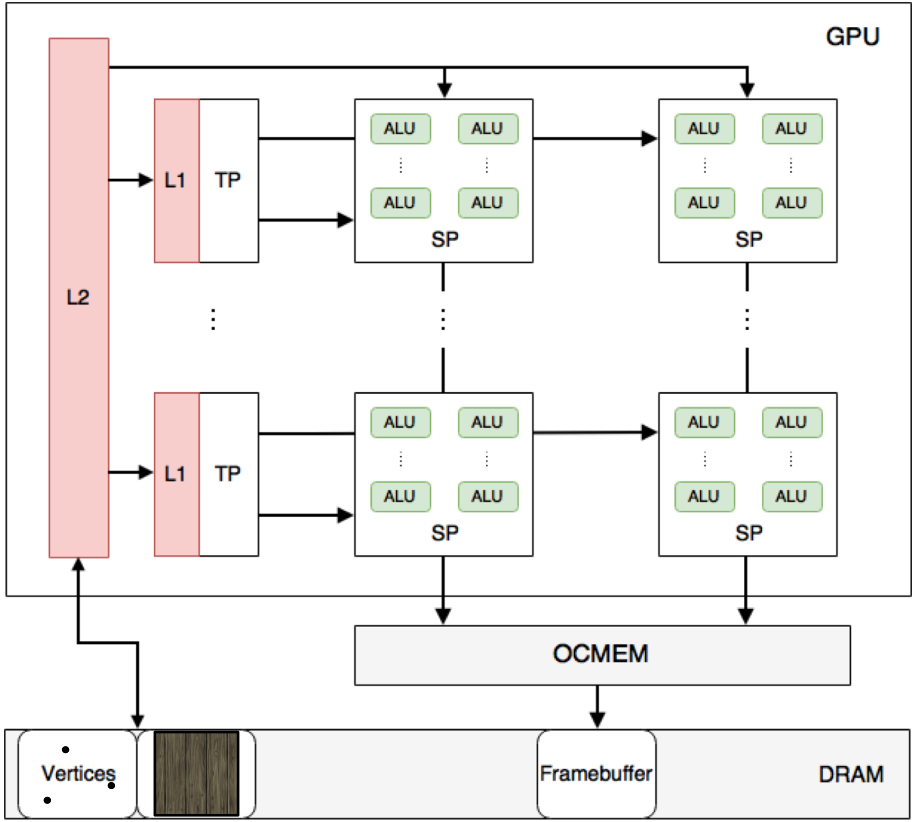
# #P1. GPU: The rendering pipeline



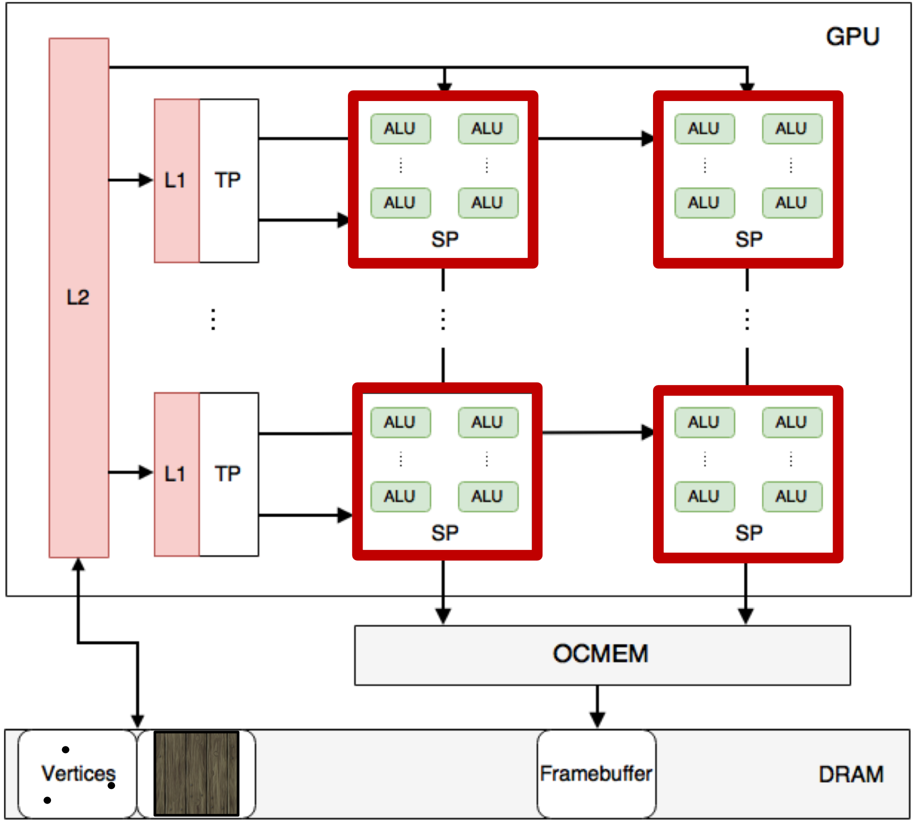
# #P1. GPU: The rendering pipeline



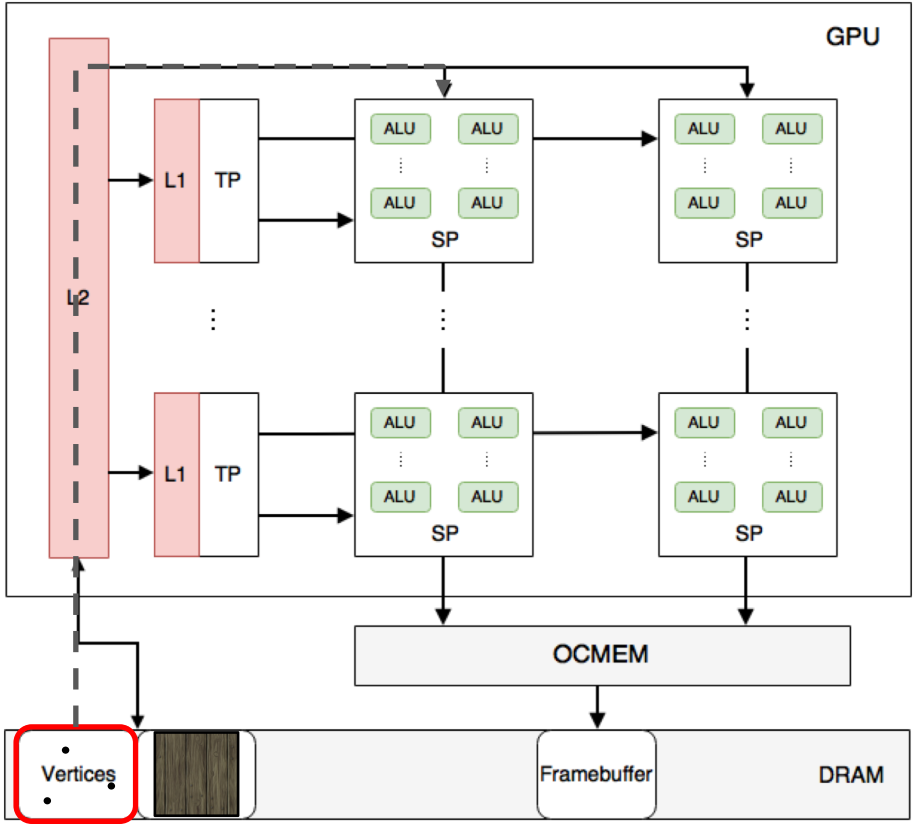
# #P1. GPU: The architecture



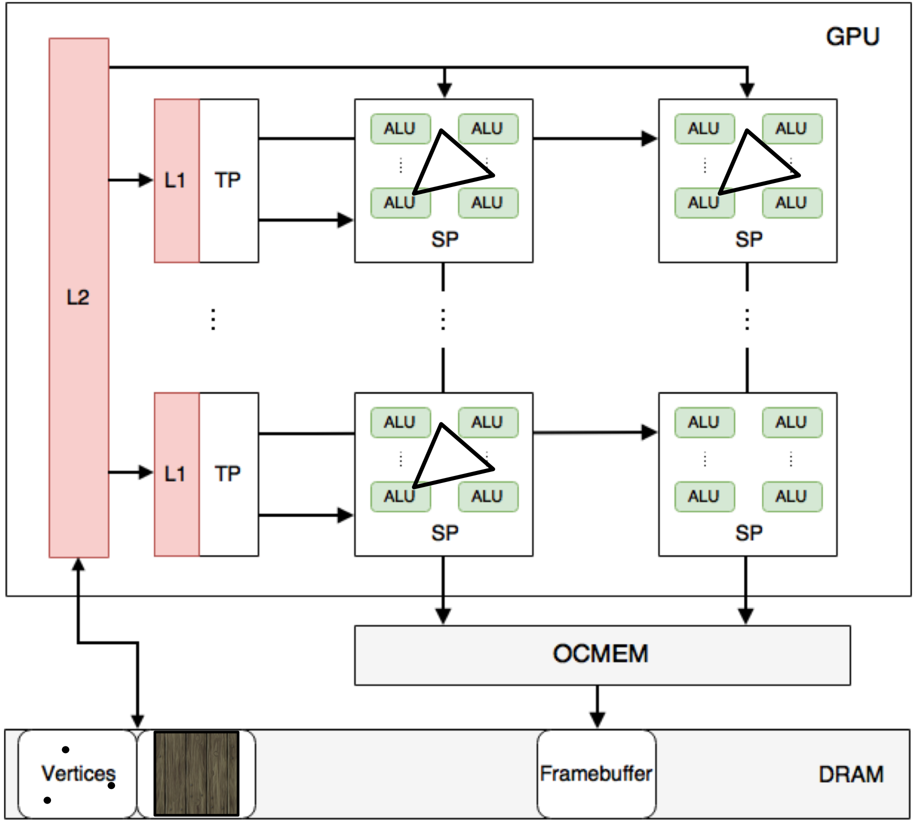
# #P1. GPU: The architecture



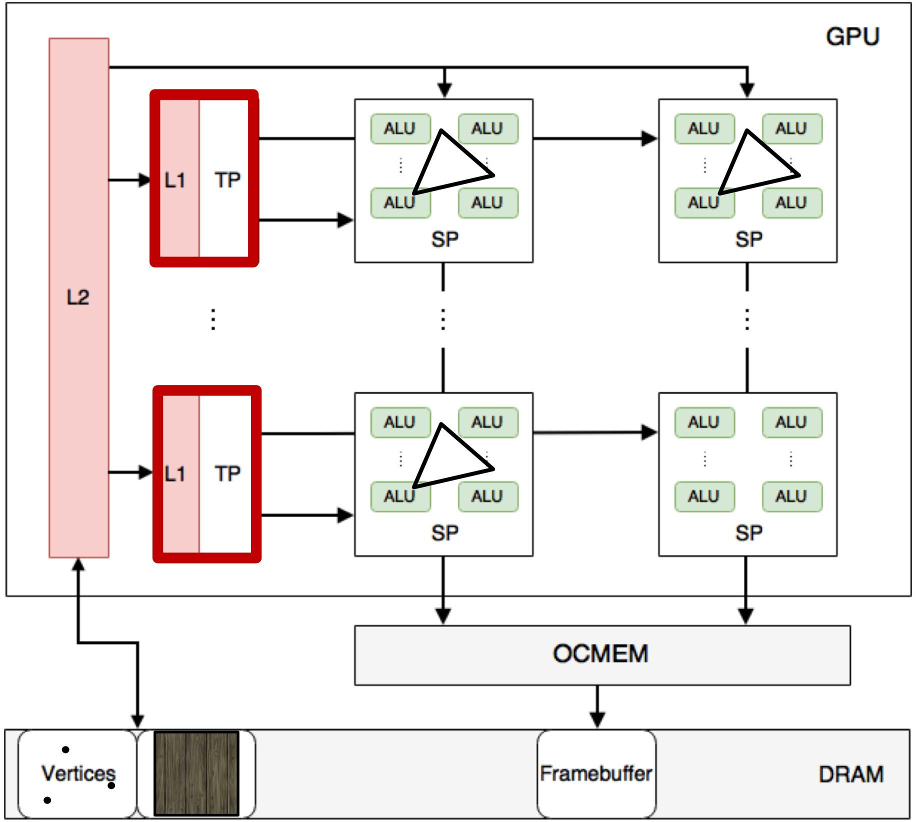
# #P1. GPU: The architecture



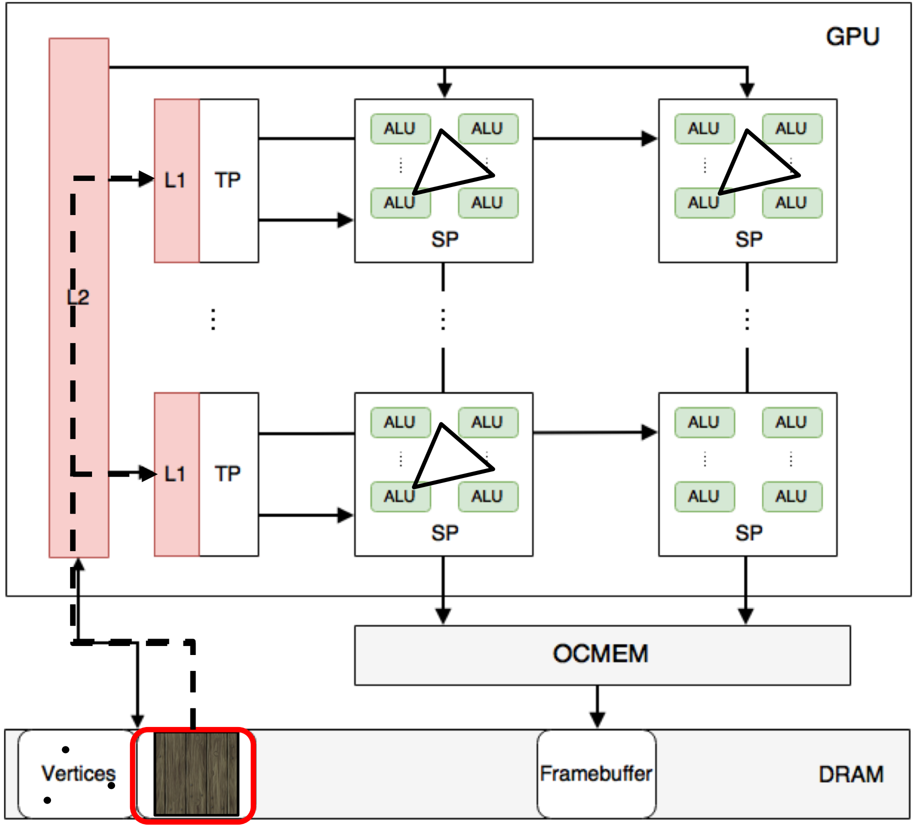
# #P1. GPU: The architecture



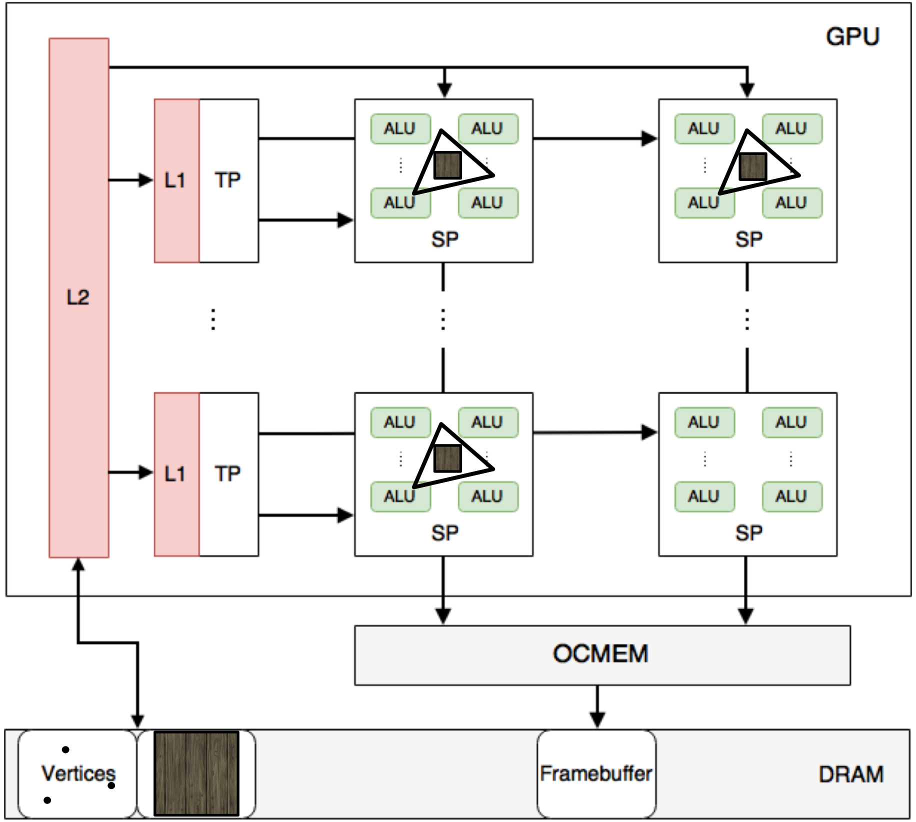
# #P1. GPU: The architecture



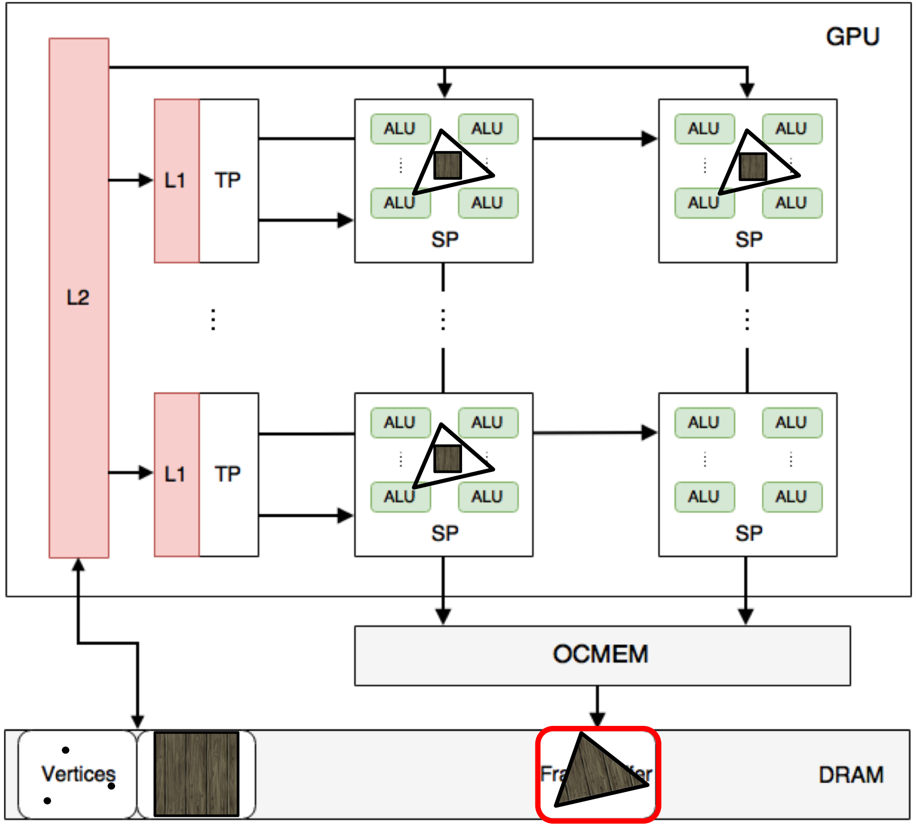
# #P1. GPU: The architecture



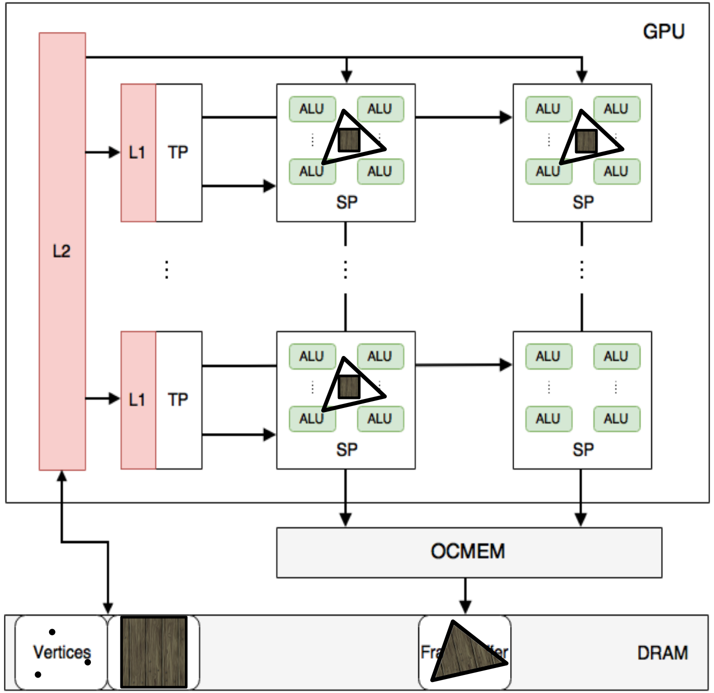
# #P1. GPU: The architecture



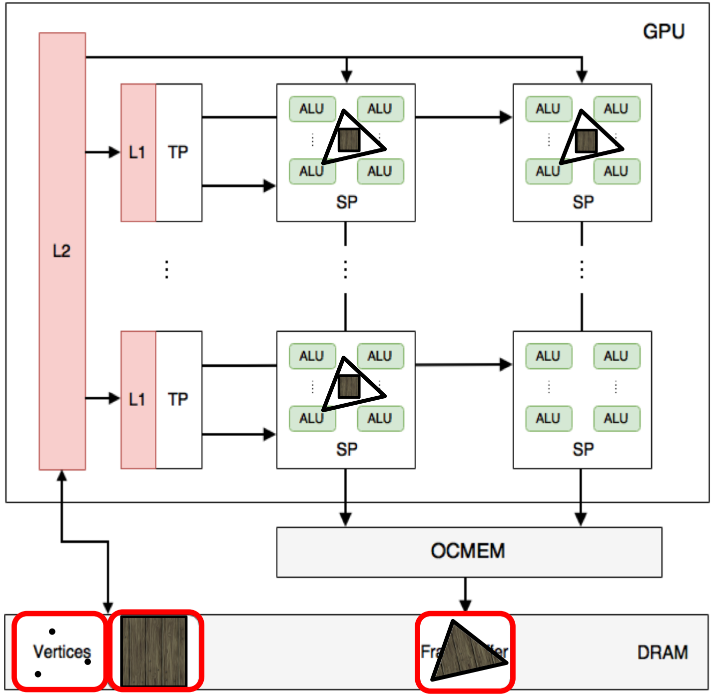
# #P1. GPU: The architecture



# #P1. DRAM access

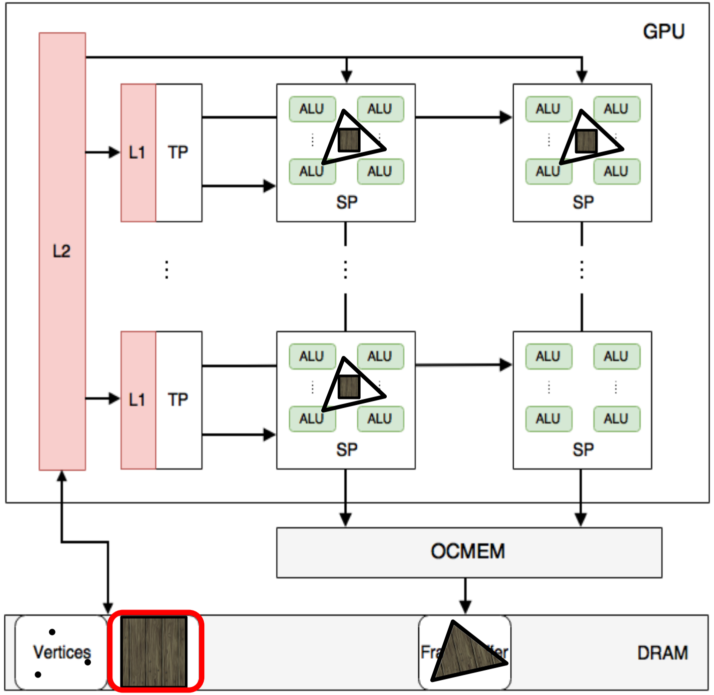


# #P1. DRAM access



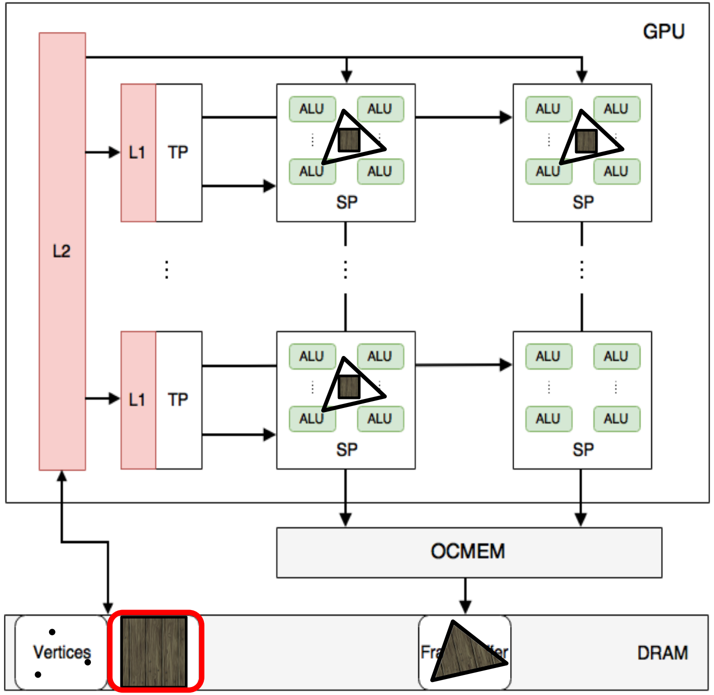

1. Read Vertices
2. Read Textures
3. Write to Framebuffer

# #P1. DRAM access




1. Read Vertices
2. Read Textures ==> most predictable
3. Write to Framebuffer

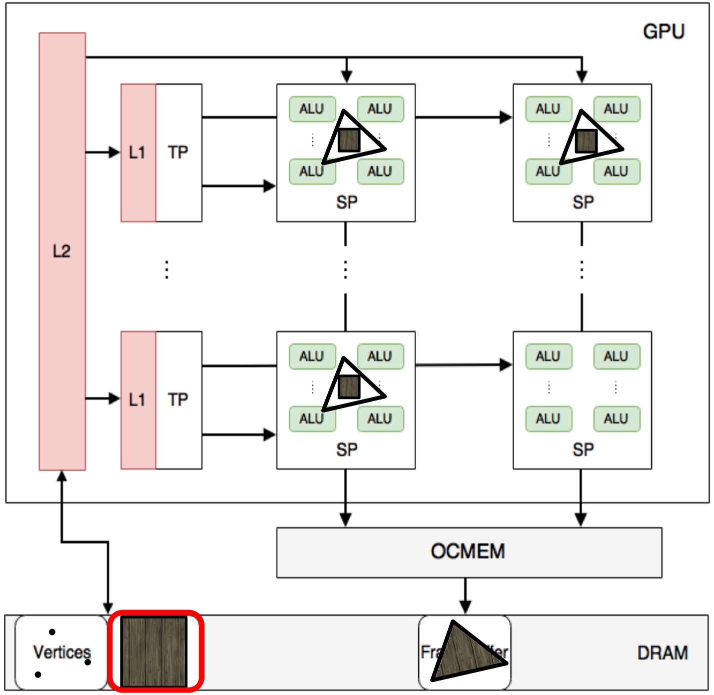
# #P1. DRAM access: texture sampling




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  gl_FragColor = texture2D(tex, coord);
}
```

# #P1. DRAM access: texture sampling




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  gl_FragColor = texture2D(tex, coord);
}
                tex[coord]
```

# Attacker primitives

#P1. DRAM access ✓

#P2. Fast memory access

#P3. Contiguous memory

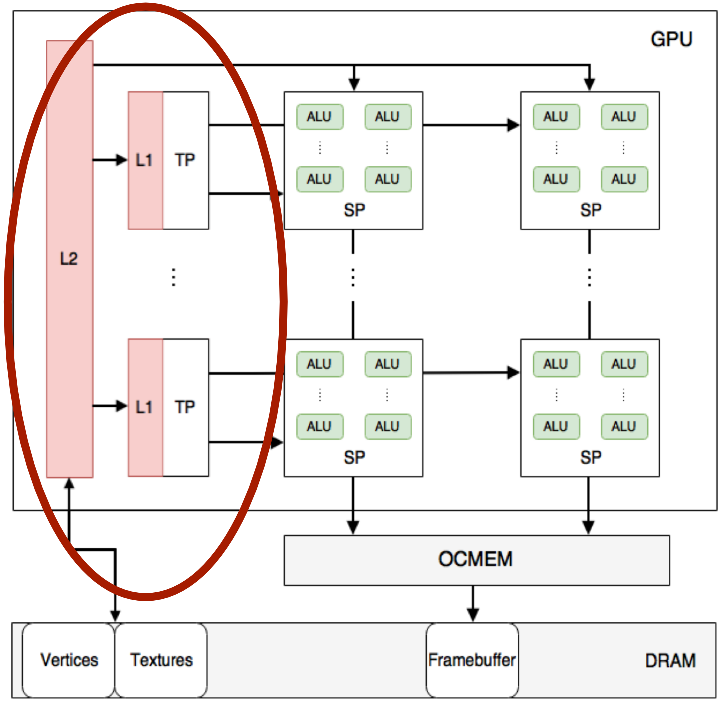
# Attacker primitives

#P1. DRAM access ✓

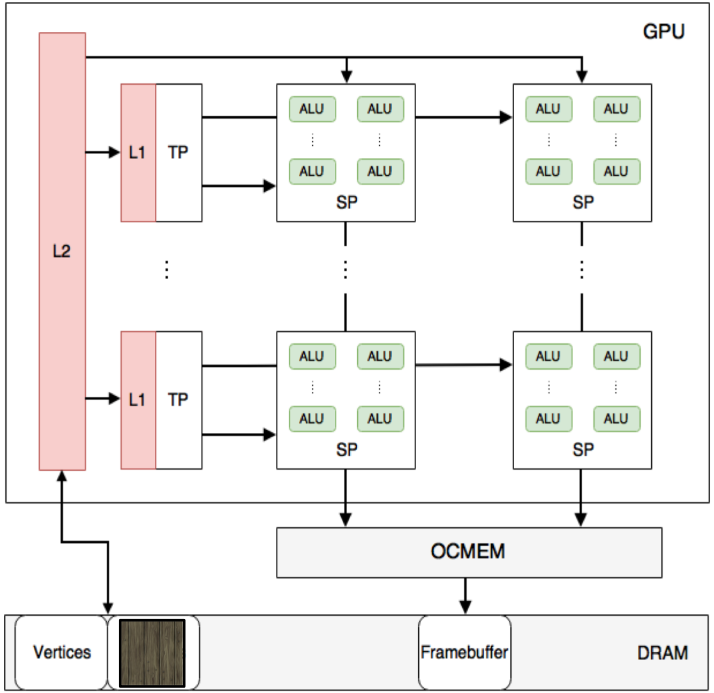
#P2. Fast memory access

#P3. Contiguous memory

# #P2. Fast memory access



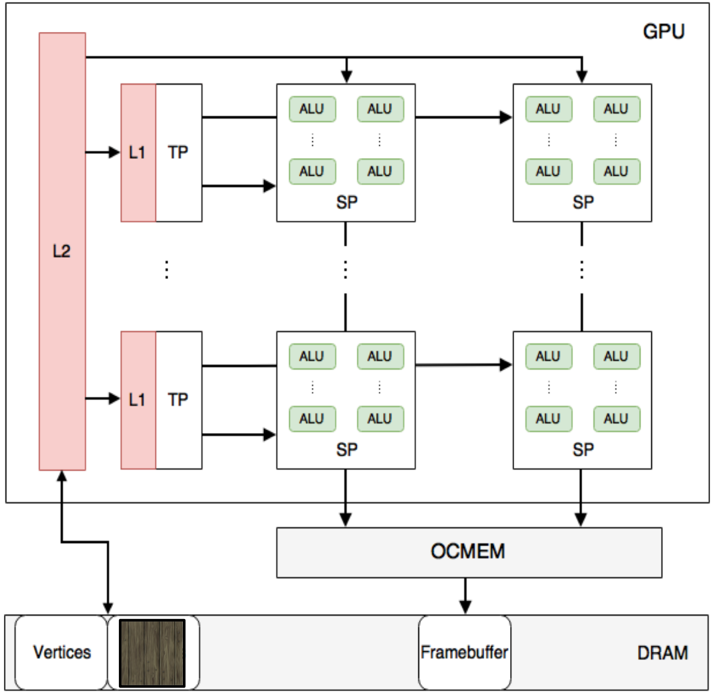
# #P2. Fast memory access




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

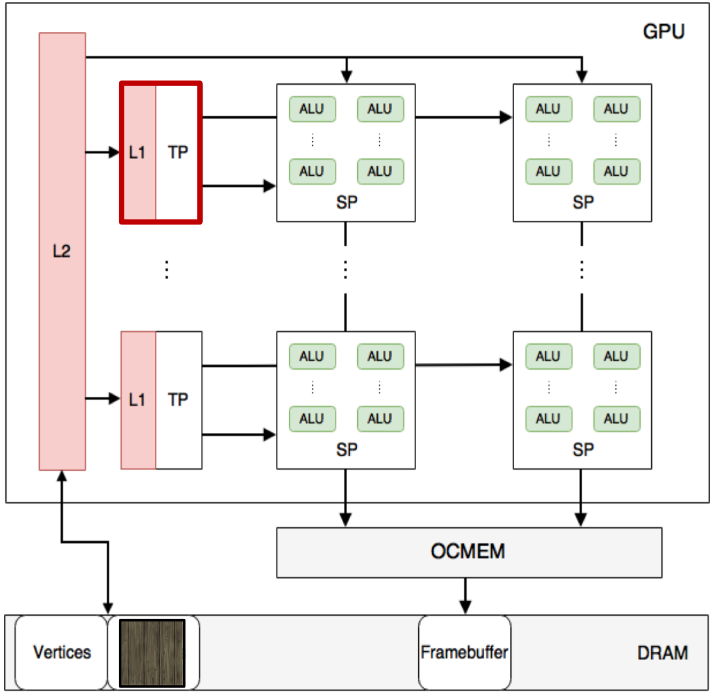
# #P2. Fast memory access




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

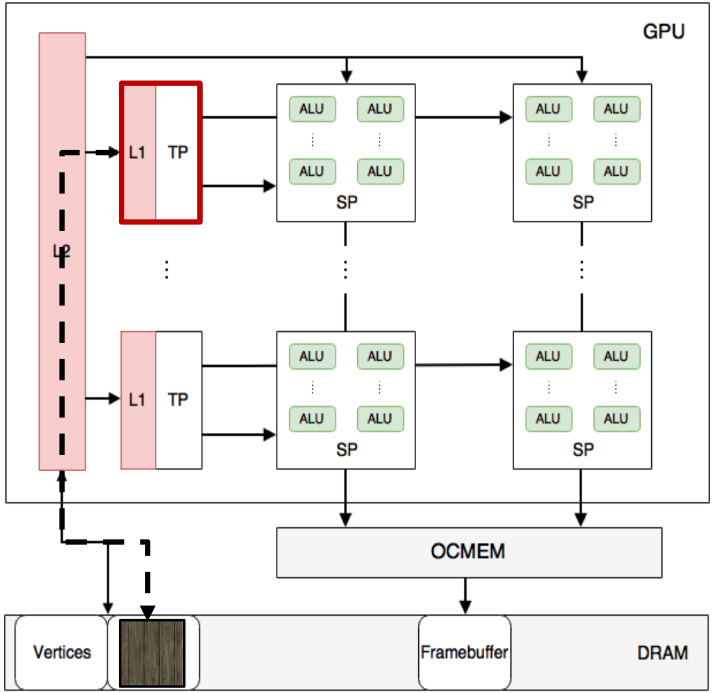
# #P2. Fast memory access




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

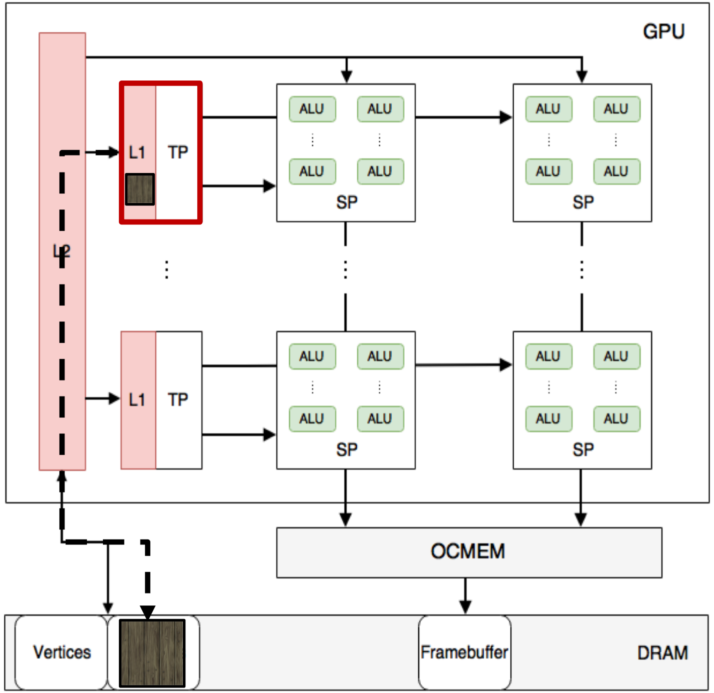
# #P2. Fast memory access




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

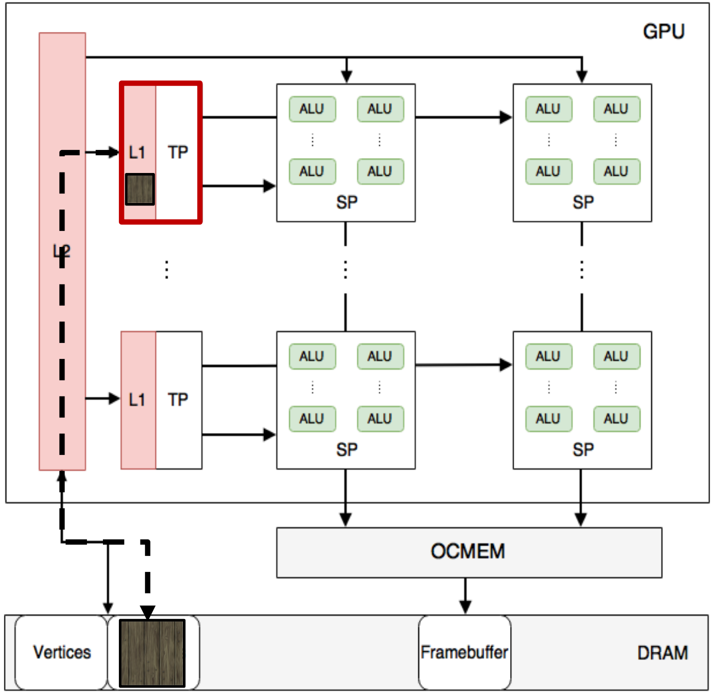
# #P2. Fast memory access




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

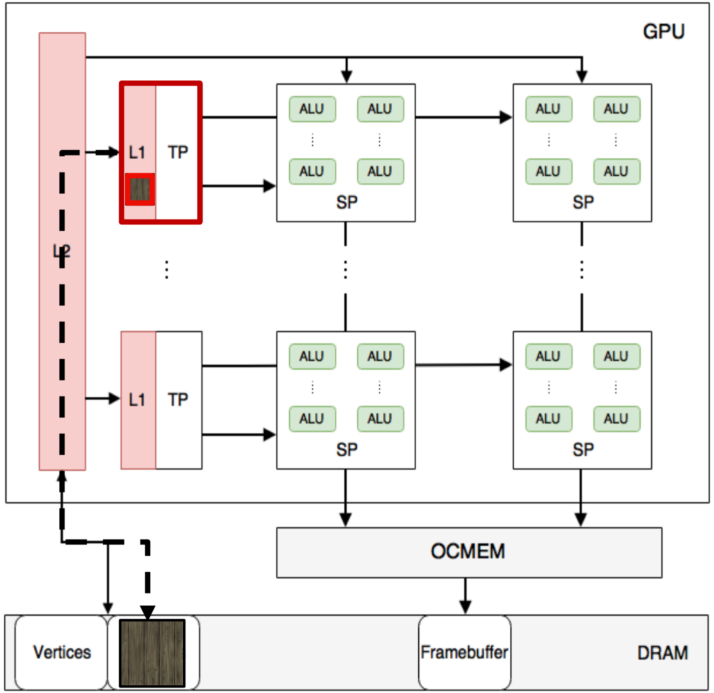
# #P2. Fast memory access




```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

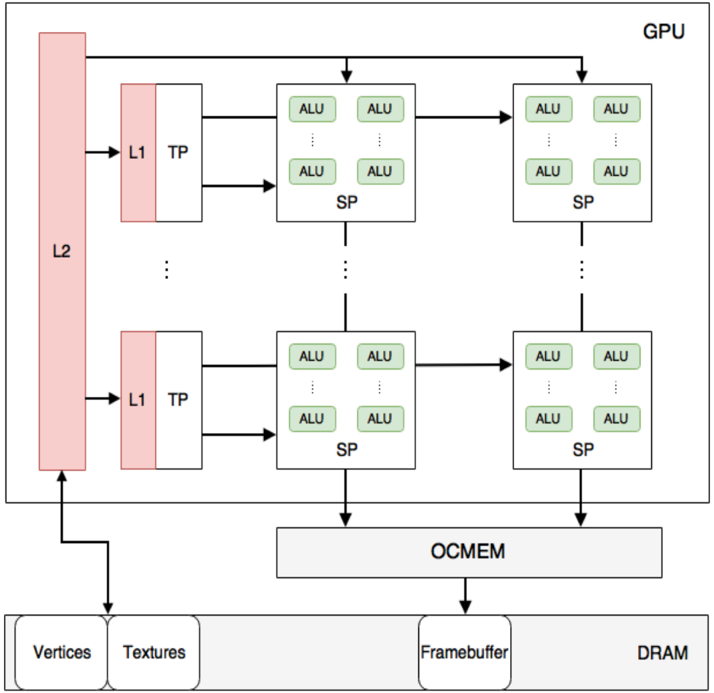
# #P2. Fast memory access



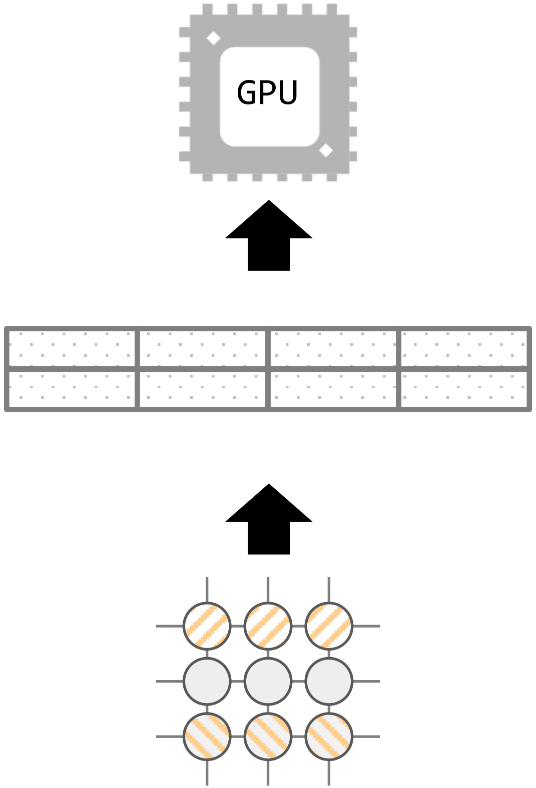

```
uniform sampler2D tex;

void main() {
  vec2 coord = vec2(0,0);
  vec4 a = texture2D(tex, coord);
  vec4 b = texture2D(tex, coord);
}
```

# #P2. Fast cache eviction

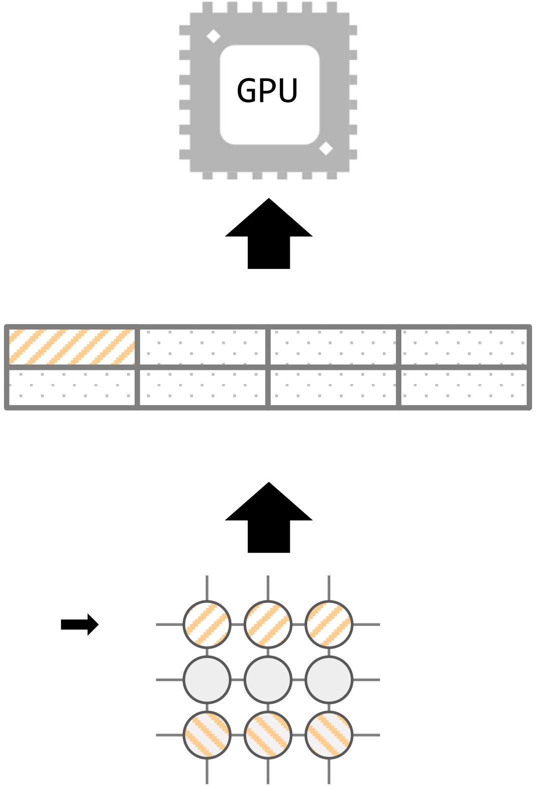


# #P2. Eviction-based Rowhammer: GPU




Caches: Small & Deterministic

# #P2. Eviction-based Rowhammer: GPU

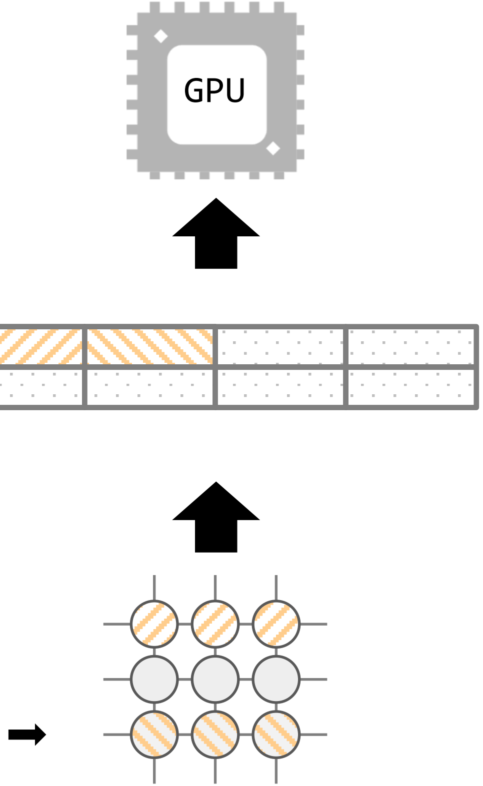



Caches: Small & Deterministic

Steps:

1. Read row *n-1*

# #P2. Eviction-based Rowhammer: GPU

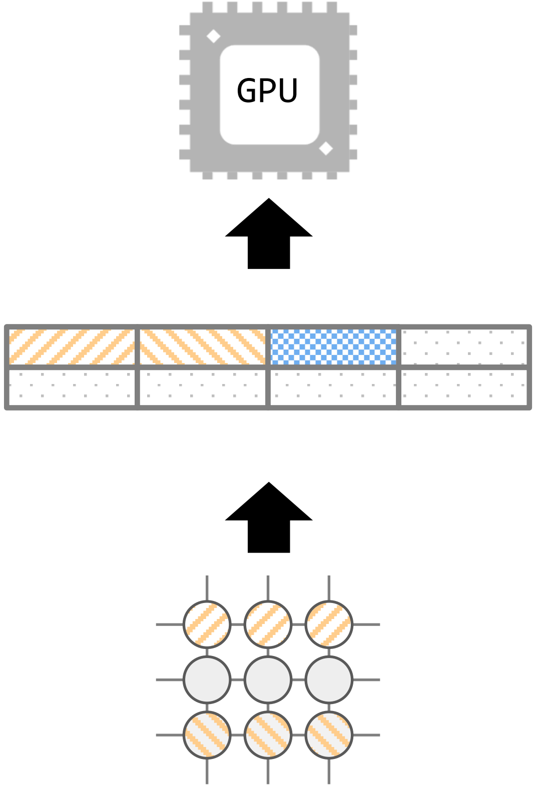



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*

# #P2. Eviction-based Rowhammer: GPU

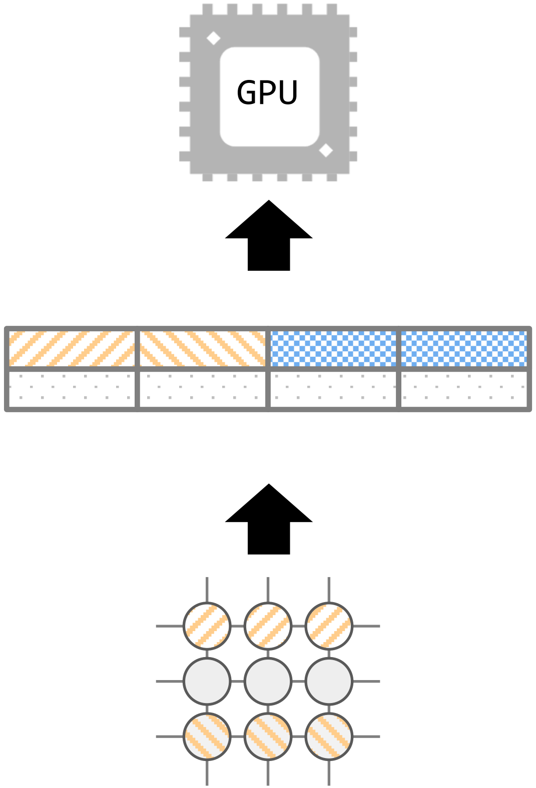



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

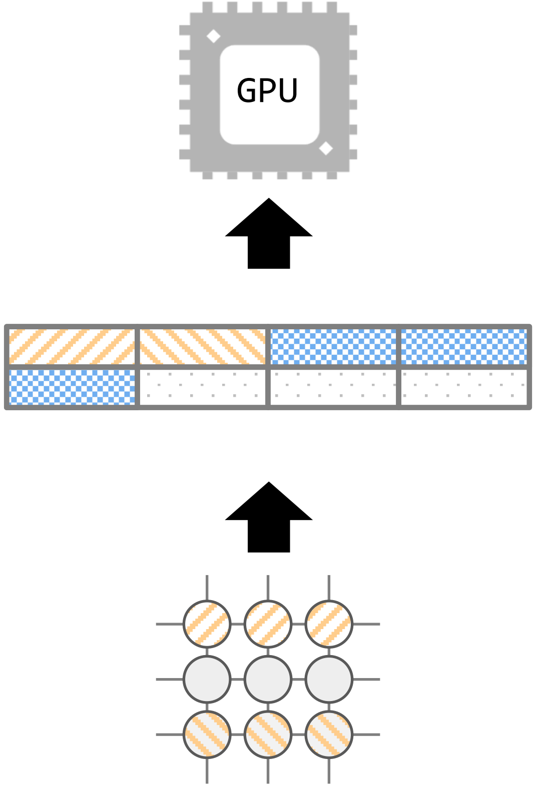



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

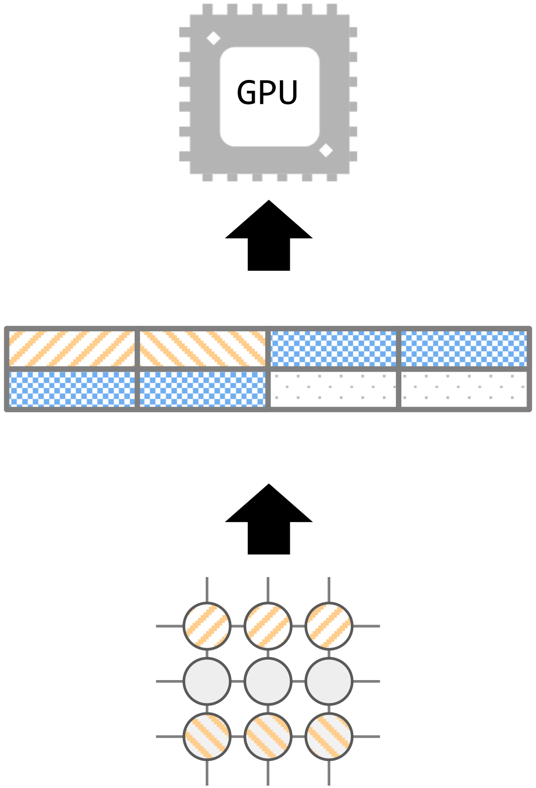



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

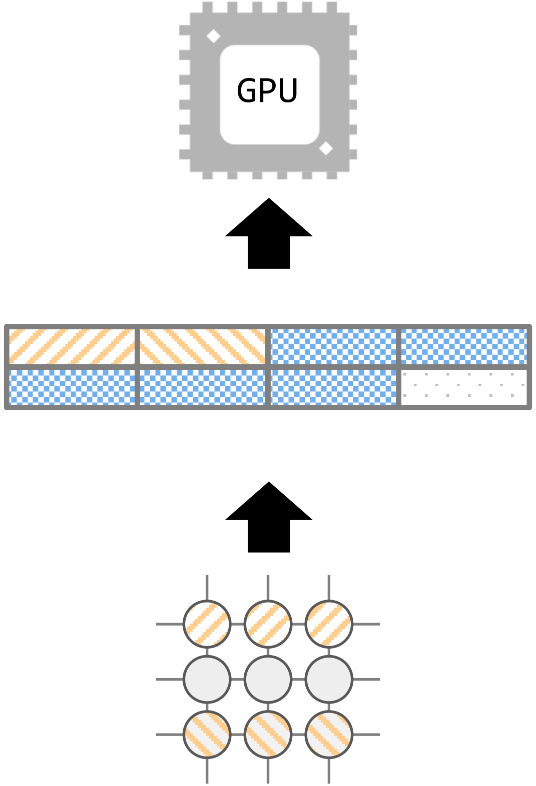



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

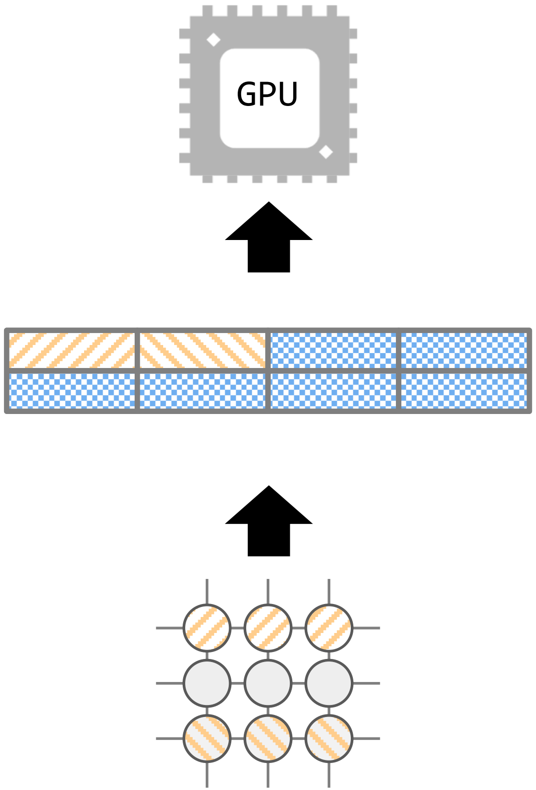



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

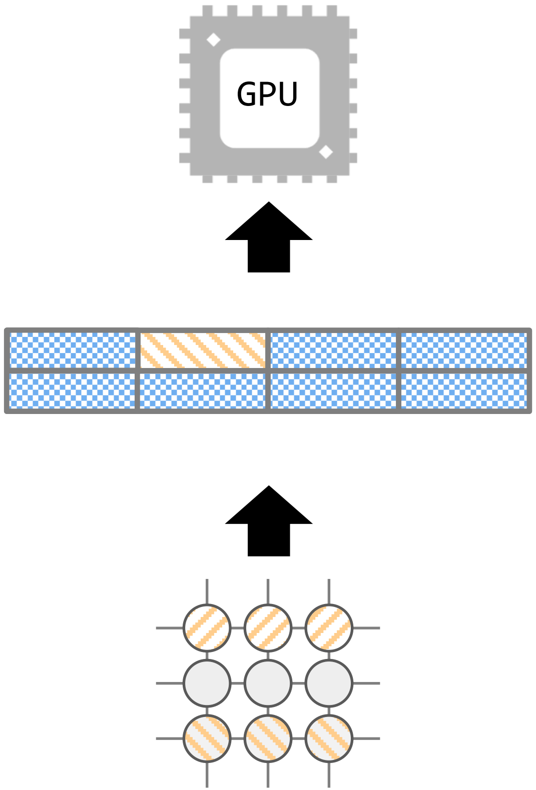



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

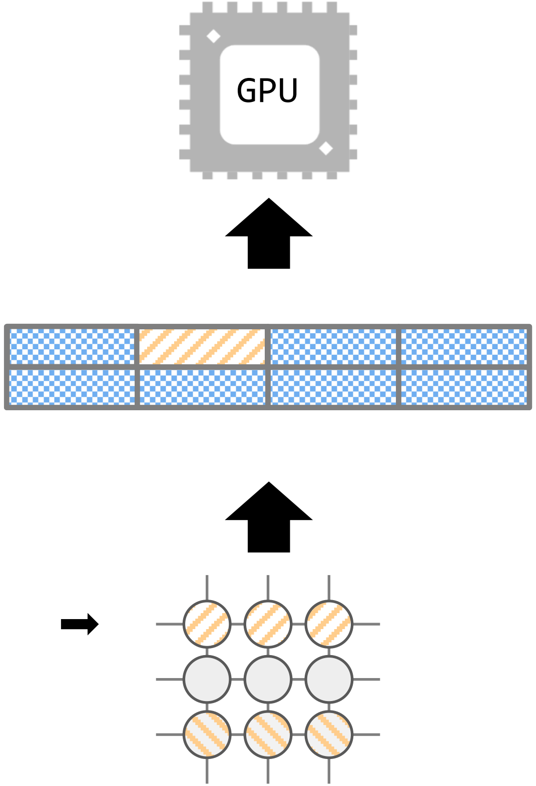



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++

# #P2. Eviction-based Rowhammer: GPU

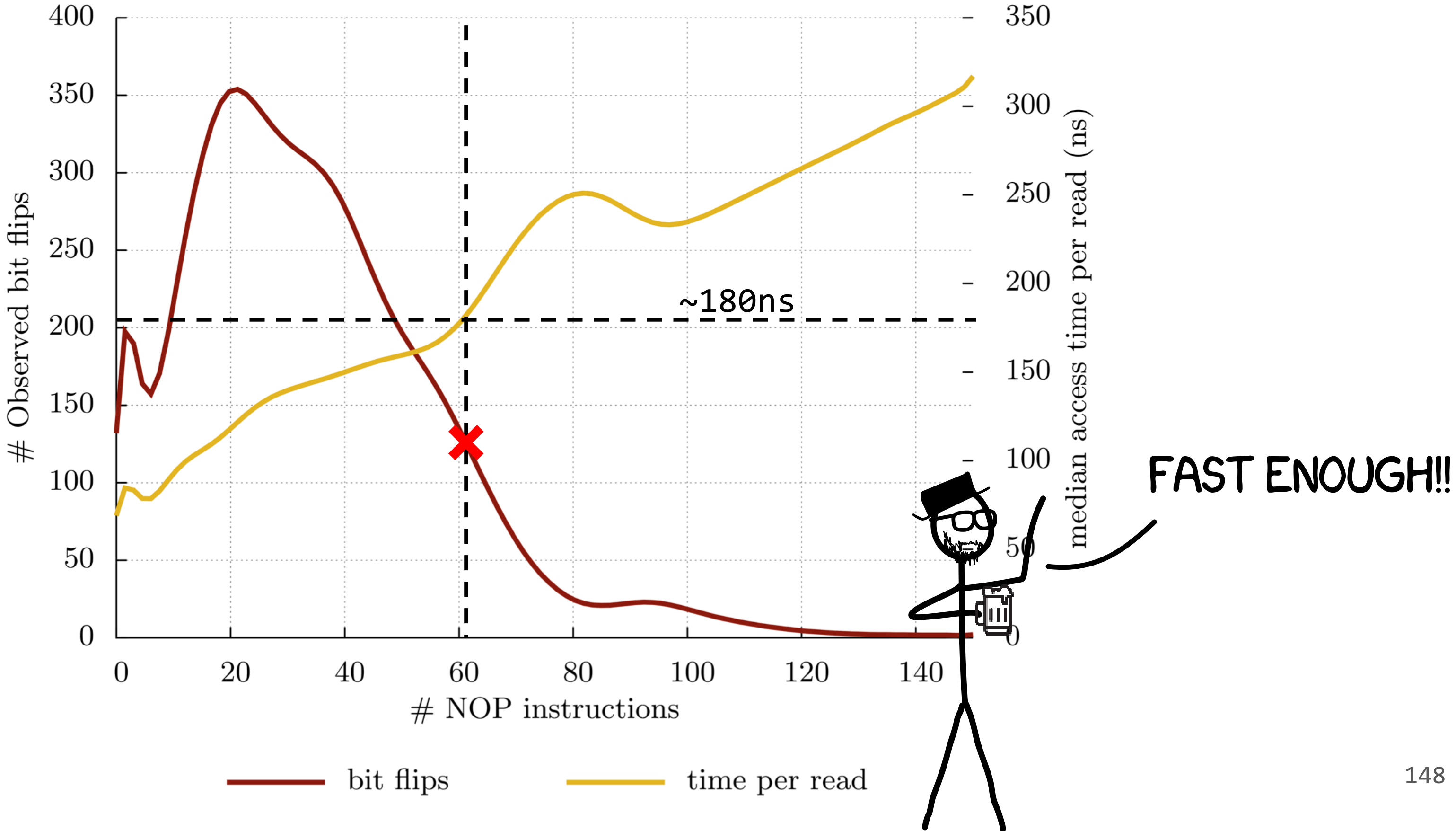



Caches: Small & Deterministic

Steps:

1. Read row *n-1*
2. Read row *n+1*
3. Evict++
4. Read row n-1

# #P2. Eviction-based Rowhammer: GPU



# Attacker primitives

#P1. DRAM access ✓

#P2. Fast memory access

#P3. Contiguous memory

# Attacker primitives

#P1. DRAM access ✓

#P2. Fast memory access ✓

#P3. Contiguous memory

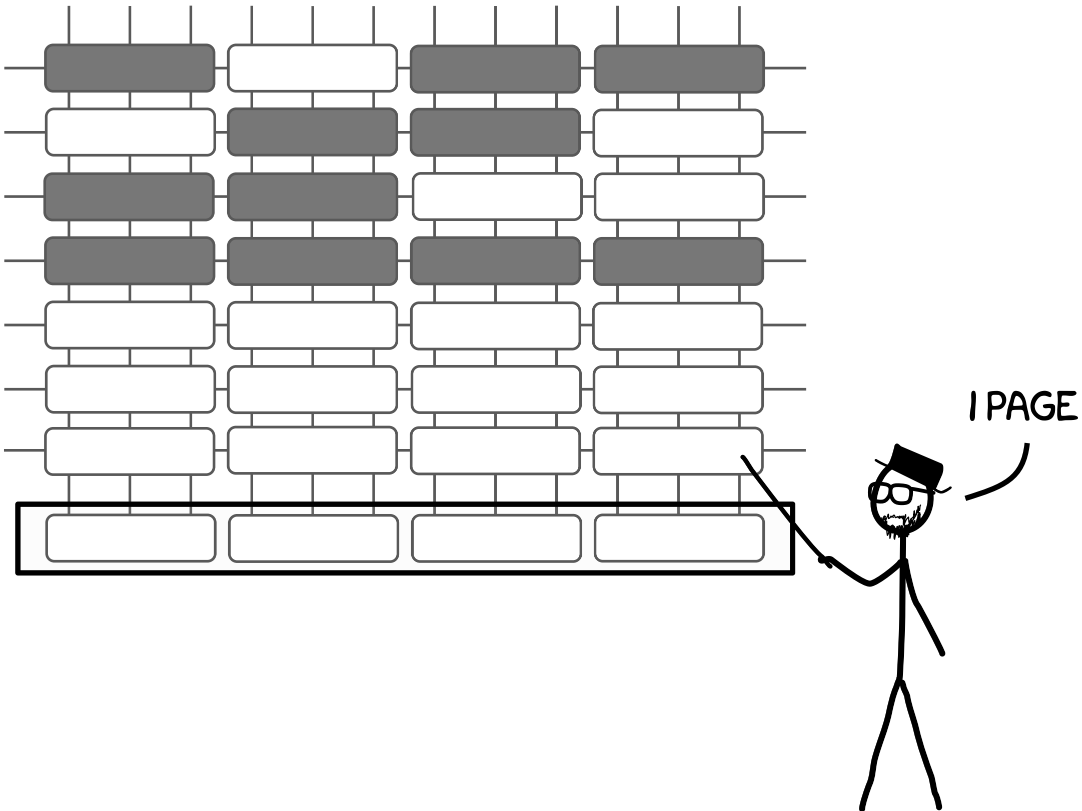
# Attacker primitives

#P1. DRAM access ✓

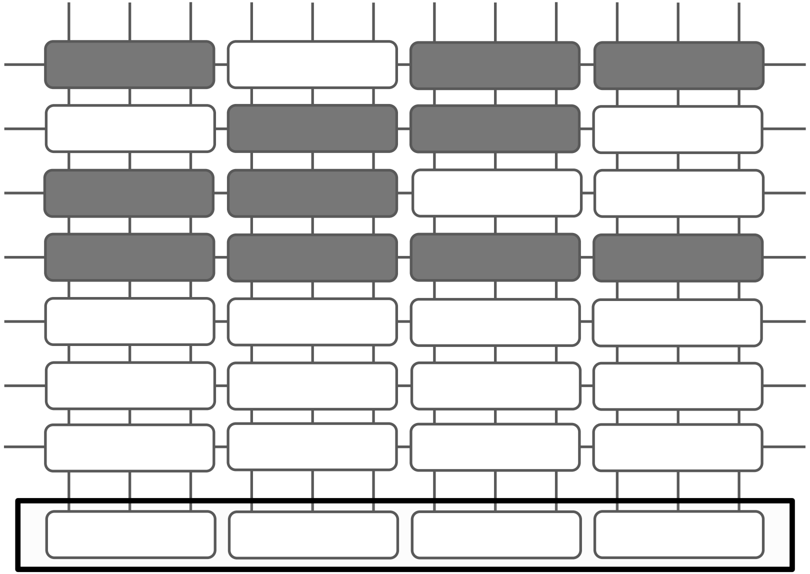
#P2. Fast memory access ✓

#P3. Contiguous memory

# #P3. Memory Allocation

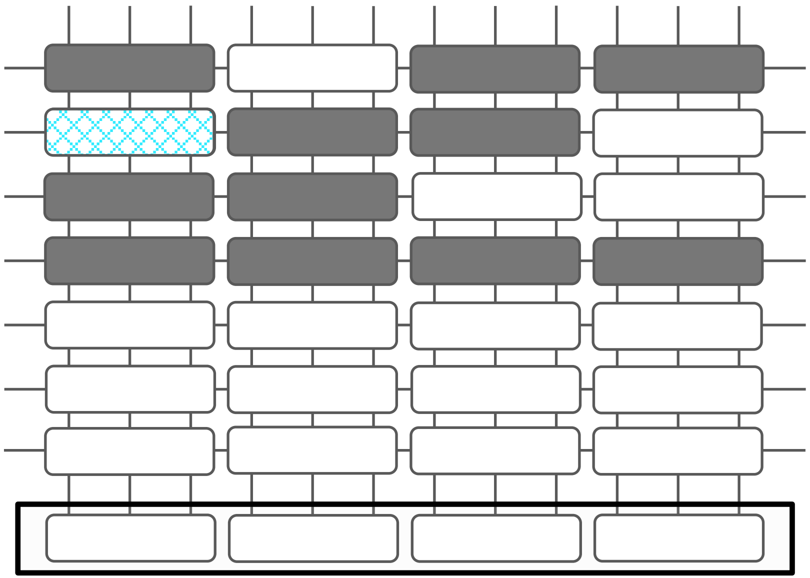


# #P3. Memory Allocation



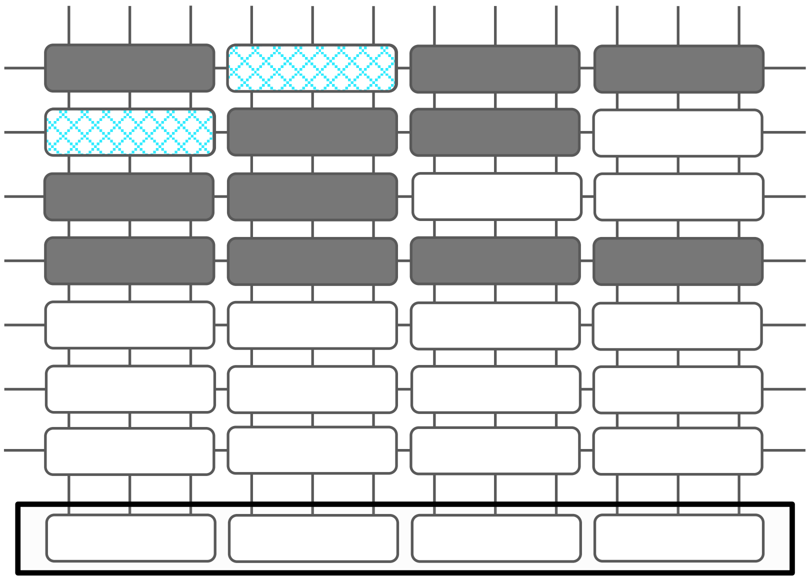
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



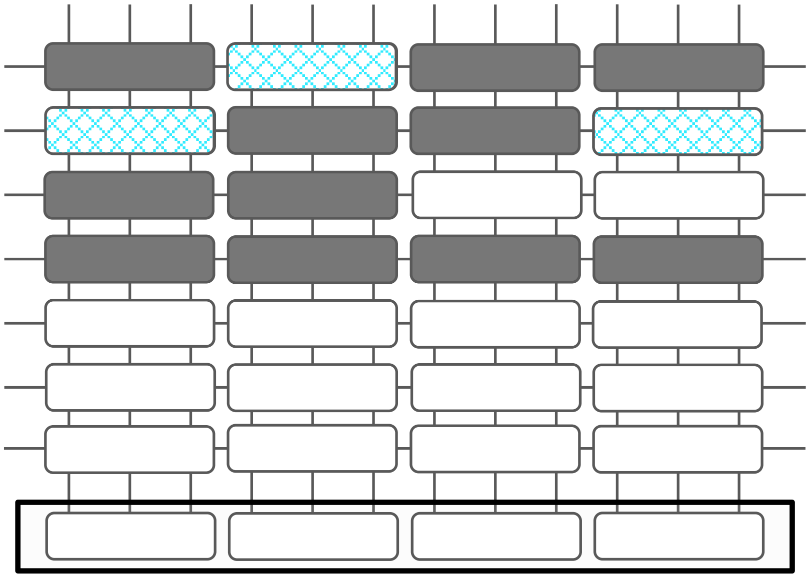
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



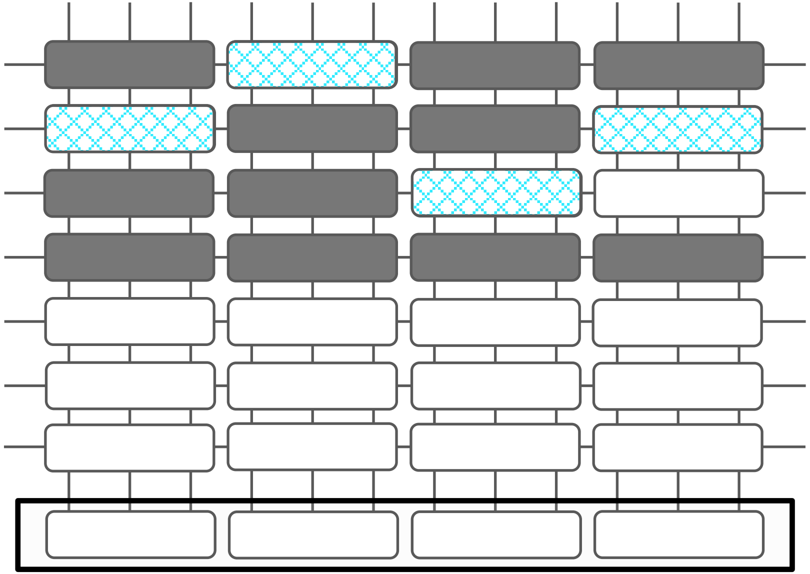
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



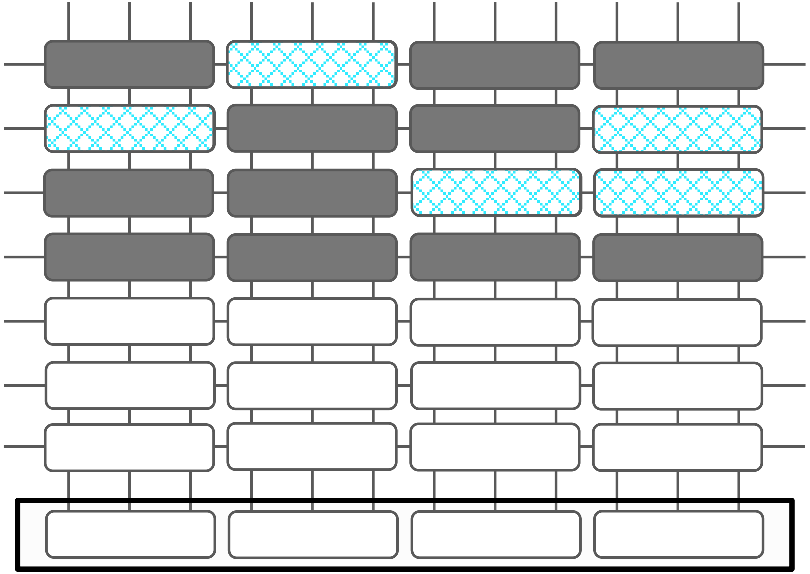
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



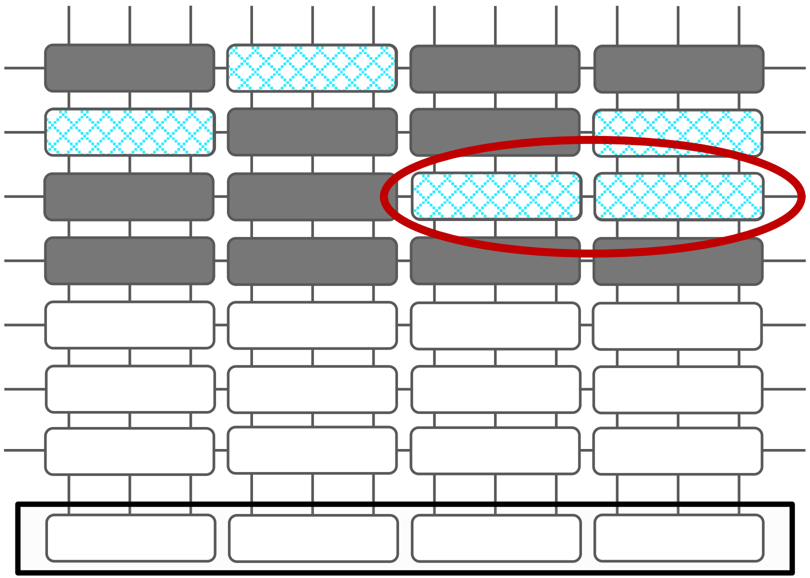
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



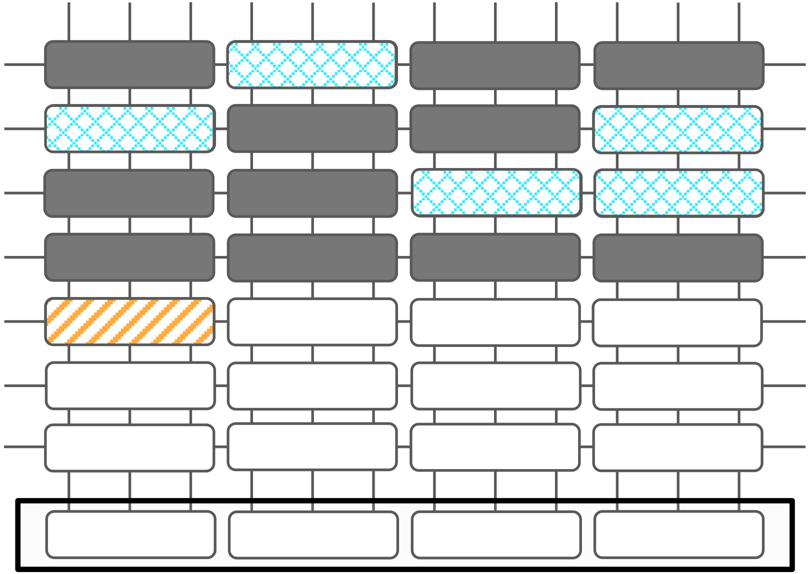
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



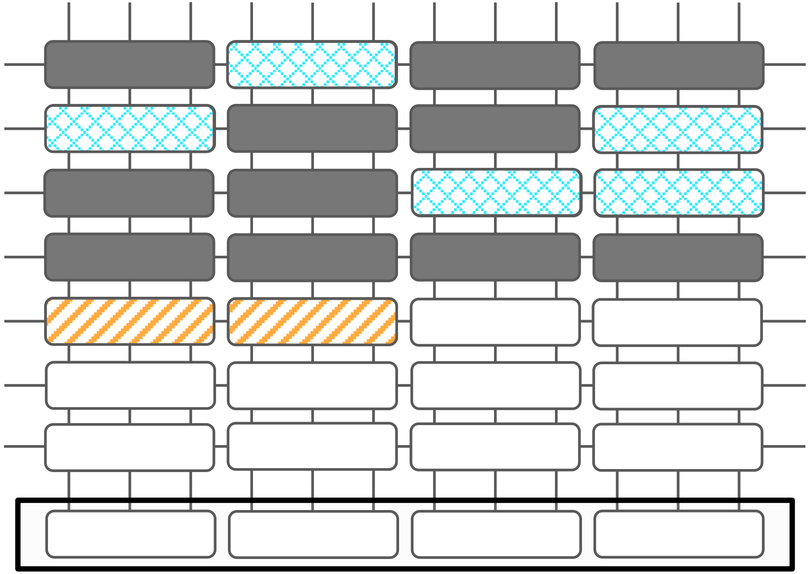
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



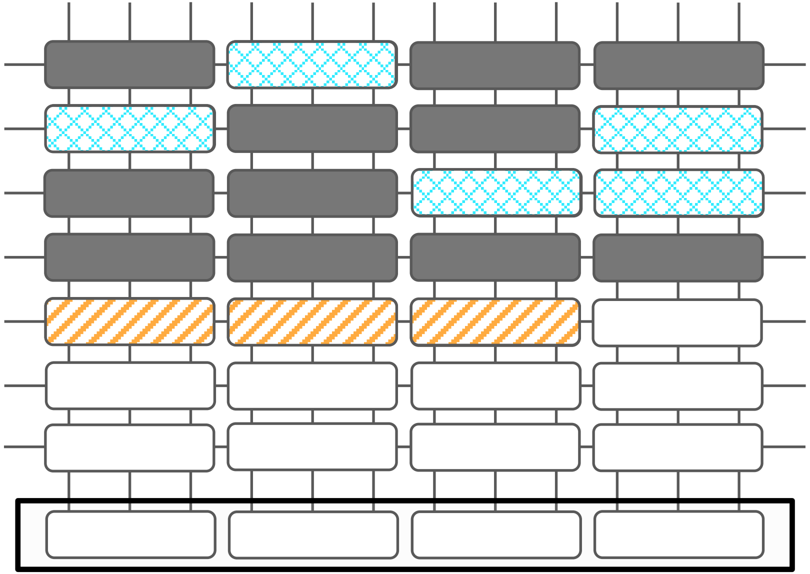
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



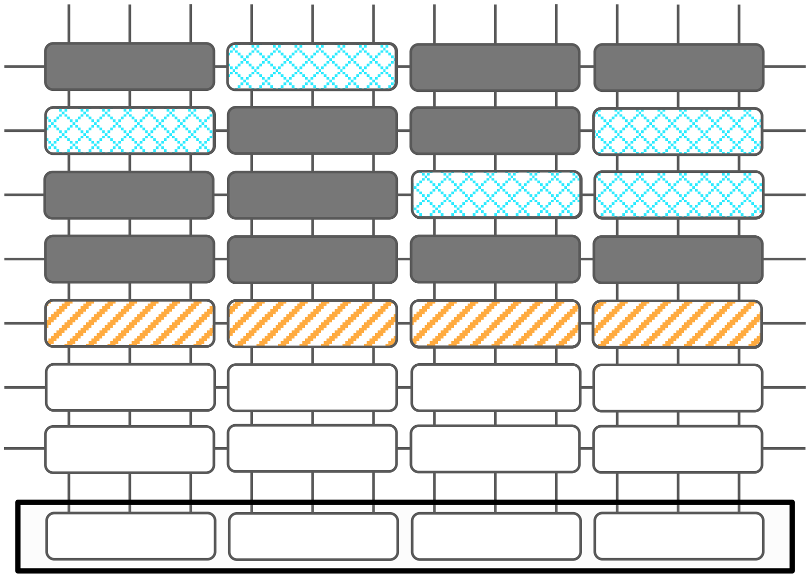
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



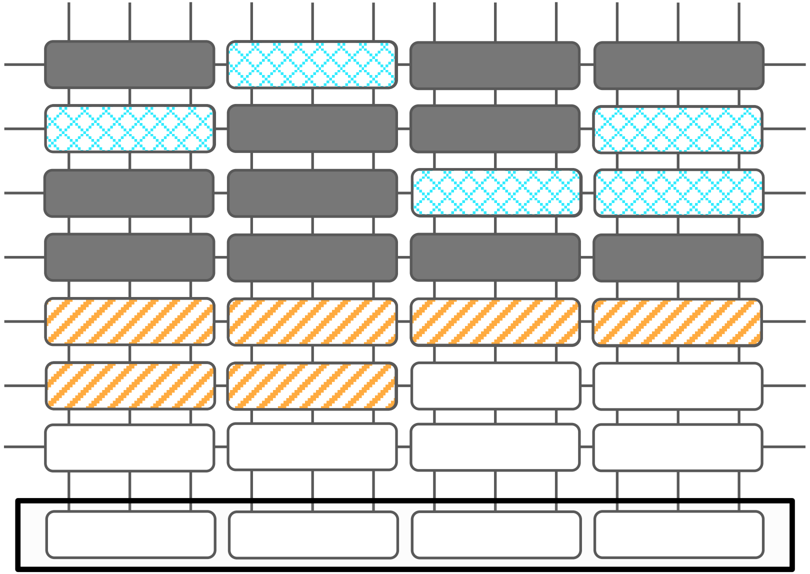
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



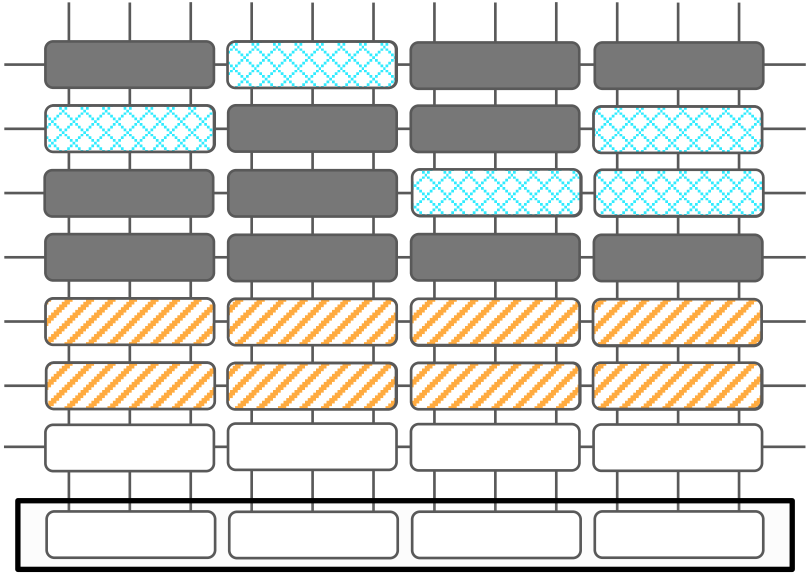
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



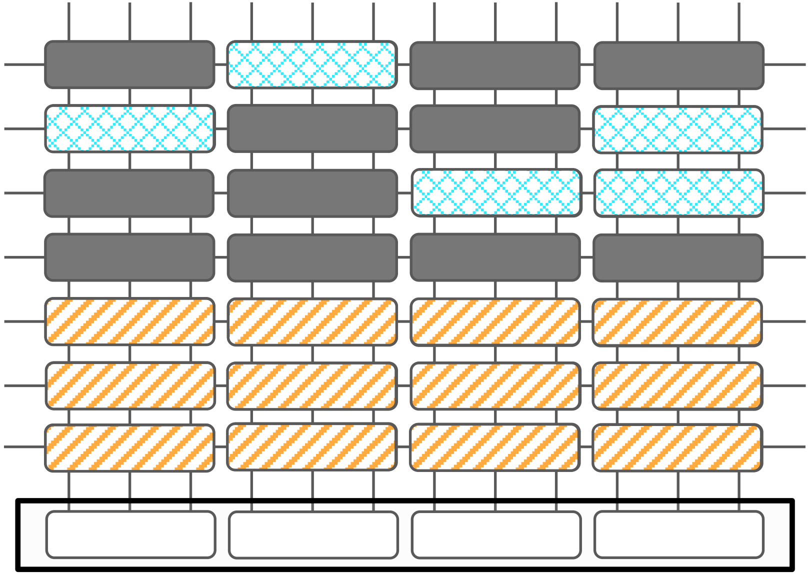
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



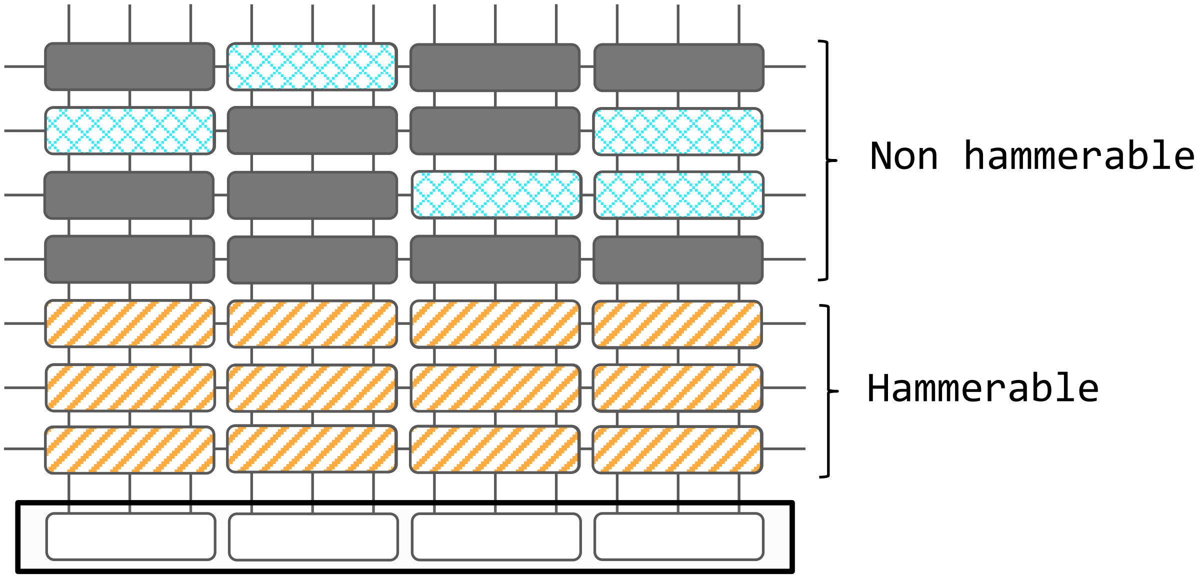
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

# #P3. Memory Allocation



```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

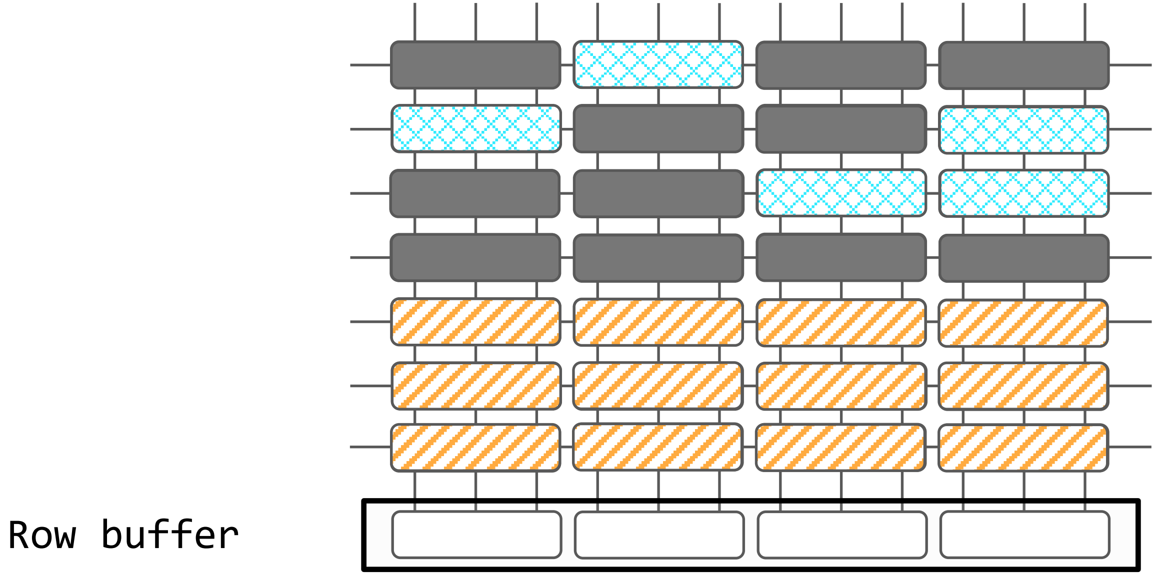
# #P3. Memory Allocation





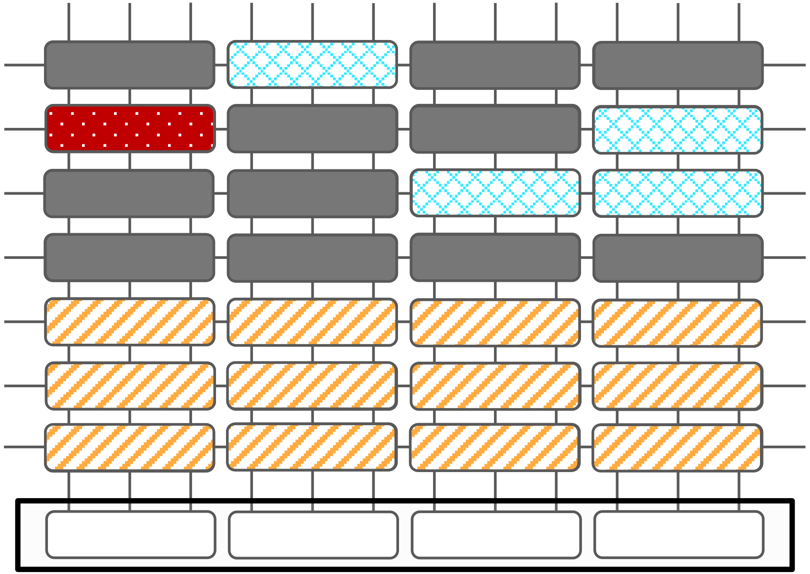
SIDE CHANNELS
SIDE CHANNELS EVERYWHERE
imgflip.com

# #P3. DRAM Reads: recap

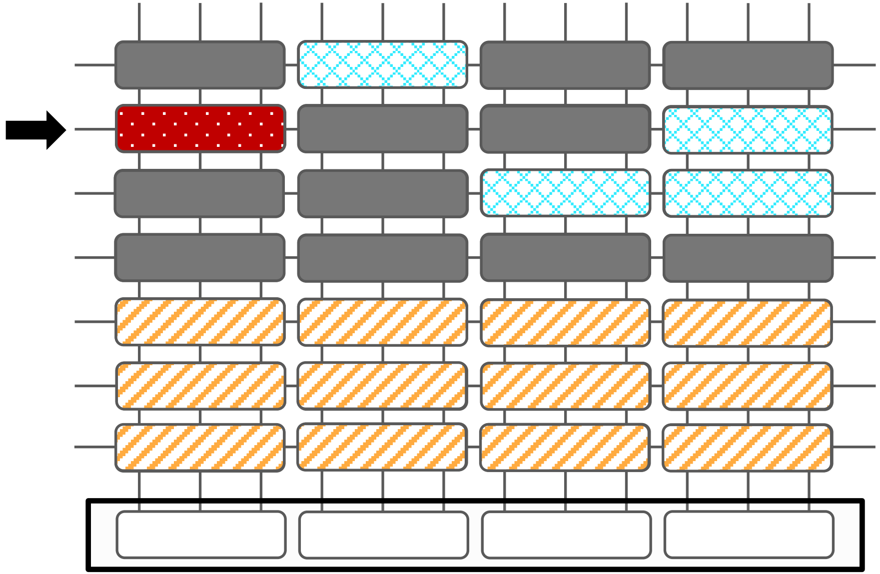


# #P3. DRAM Reads: recap

Row buffer

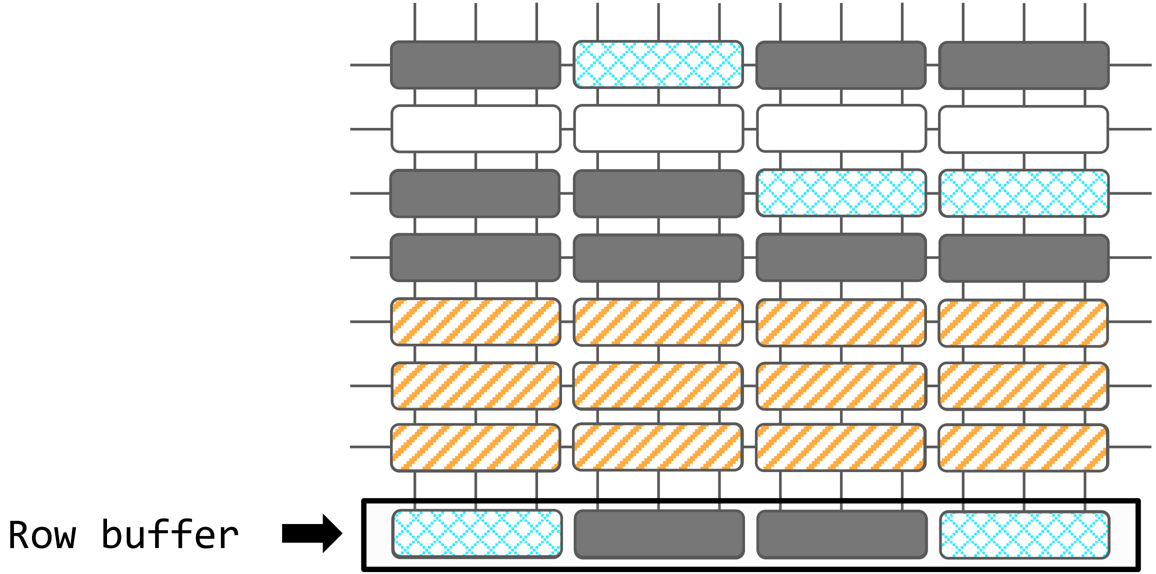


# #P3. DRAM Reads: recap



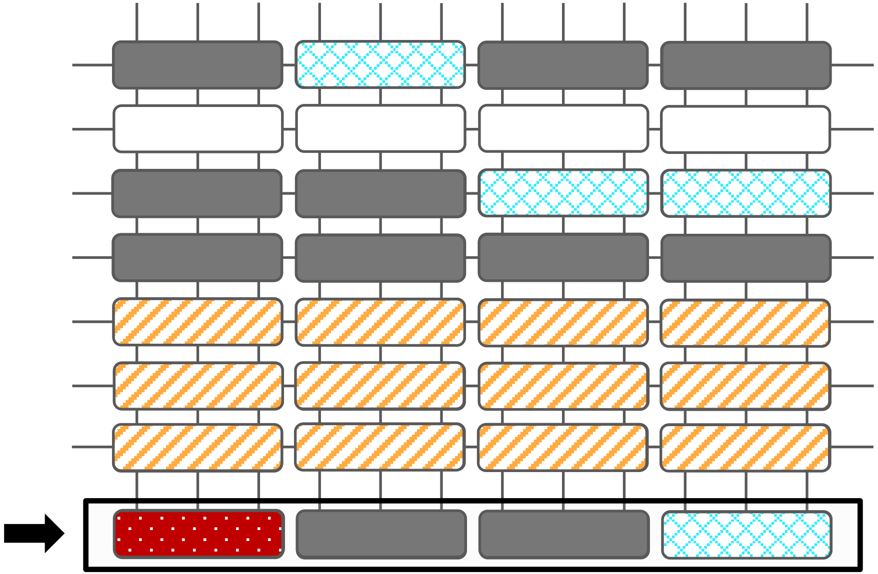
Row buffer

# #P3. DRAM Reads: recap

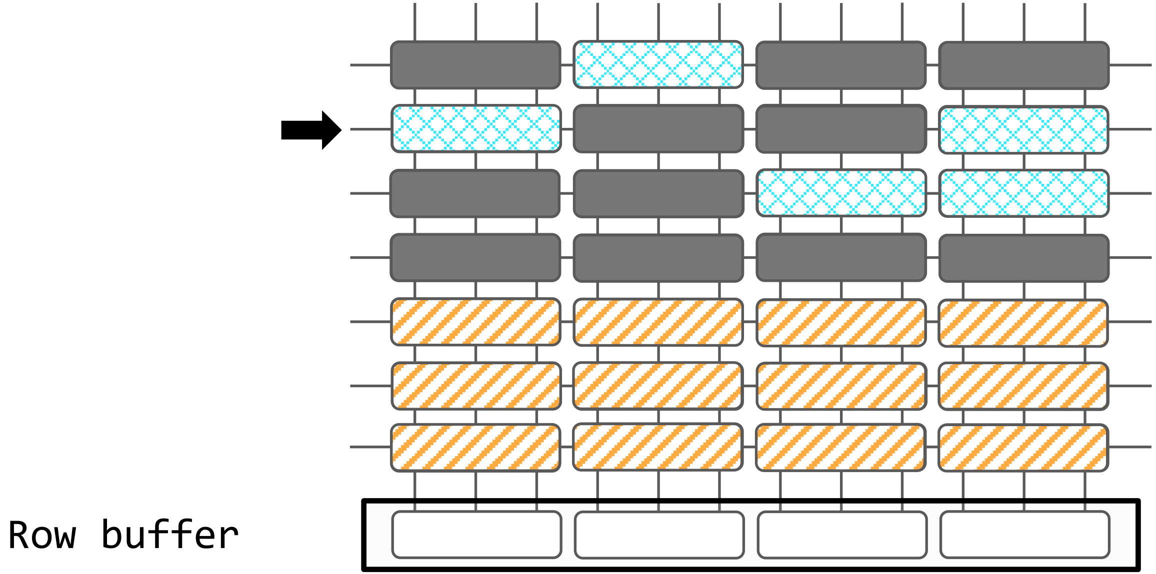


# #P3. DRAM Reads: recap

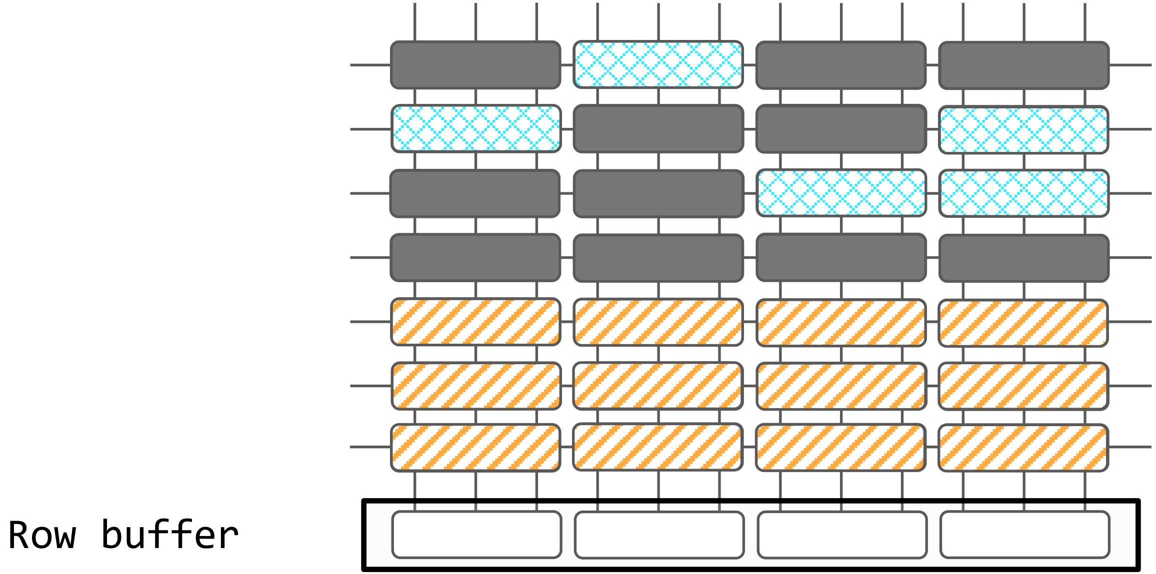
Row buffer



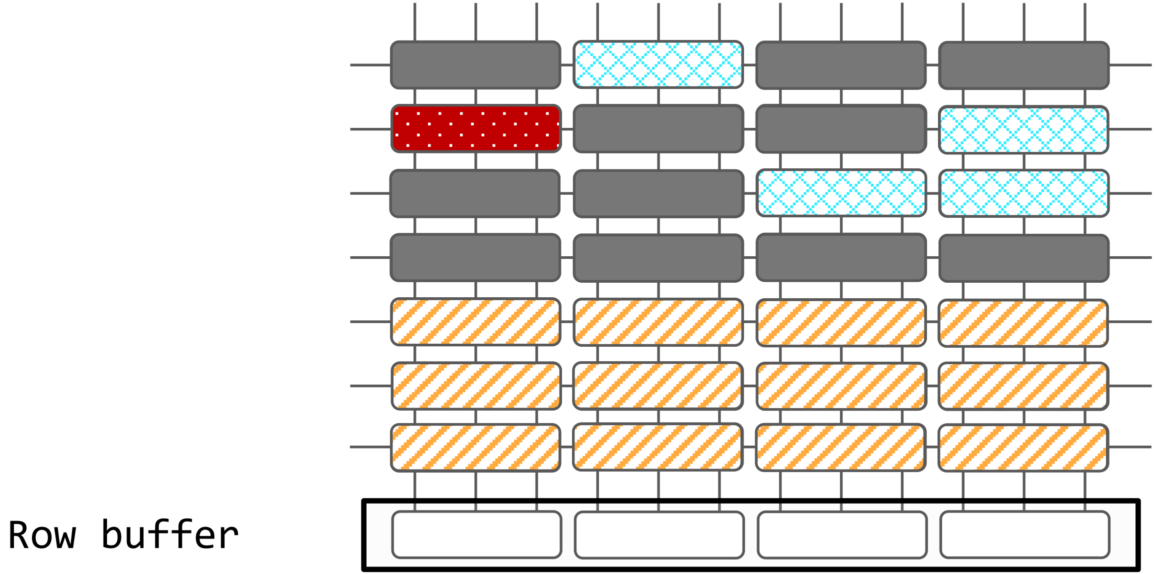
# #P3. DRAM Reads: recap



# #P3. Contiguous Memory: Detection

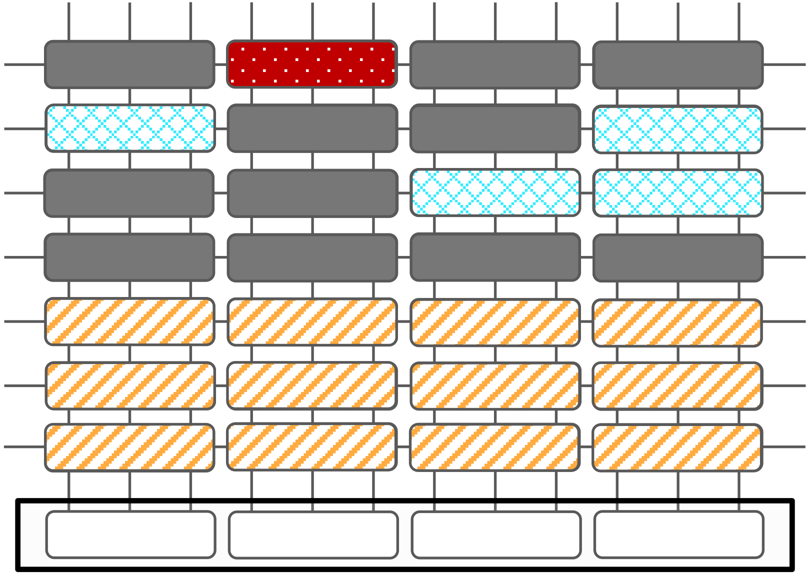


# #P3. Contiguous Memory: Detection



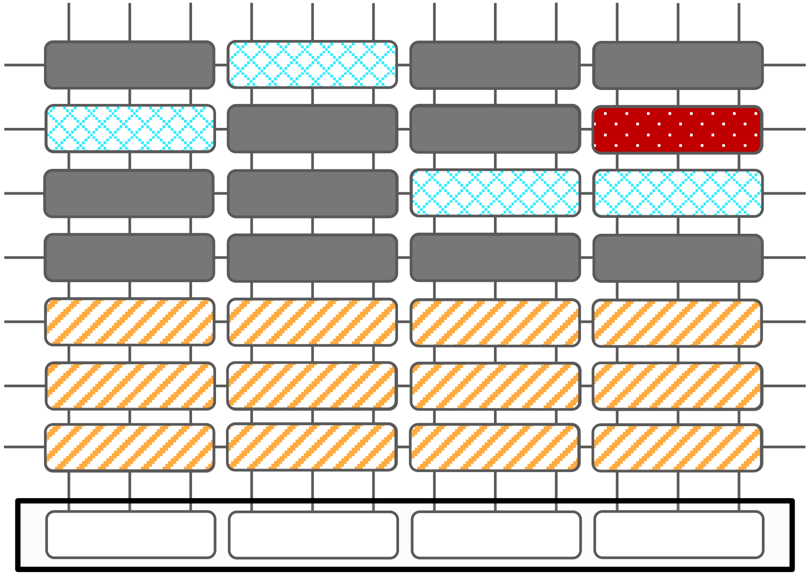
# #P3. Contiguous Memory: Detection

Row buffer

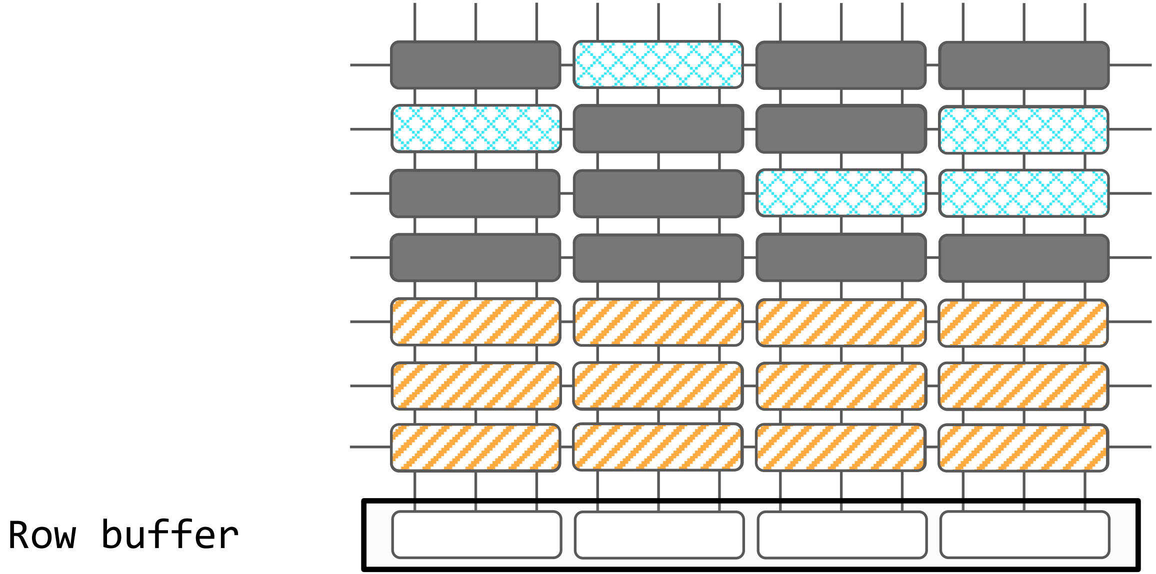


# #P3. Contiguous Memory: Detection

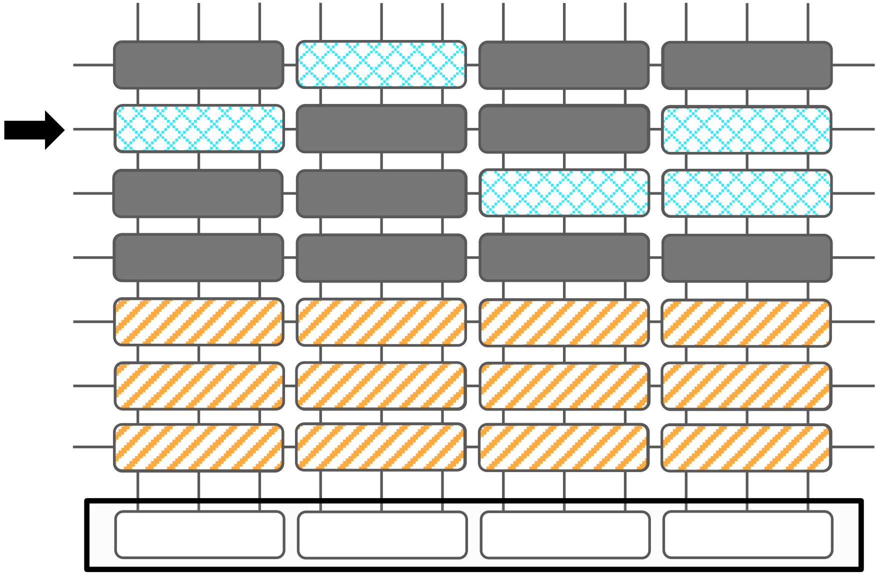
Row buffer



# #P3. Contiguous Memory: Detection

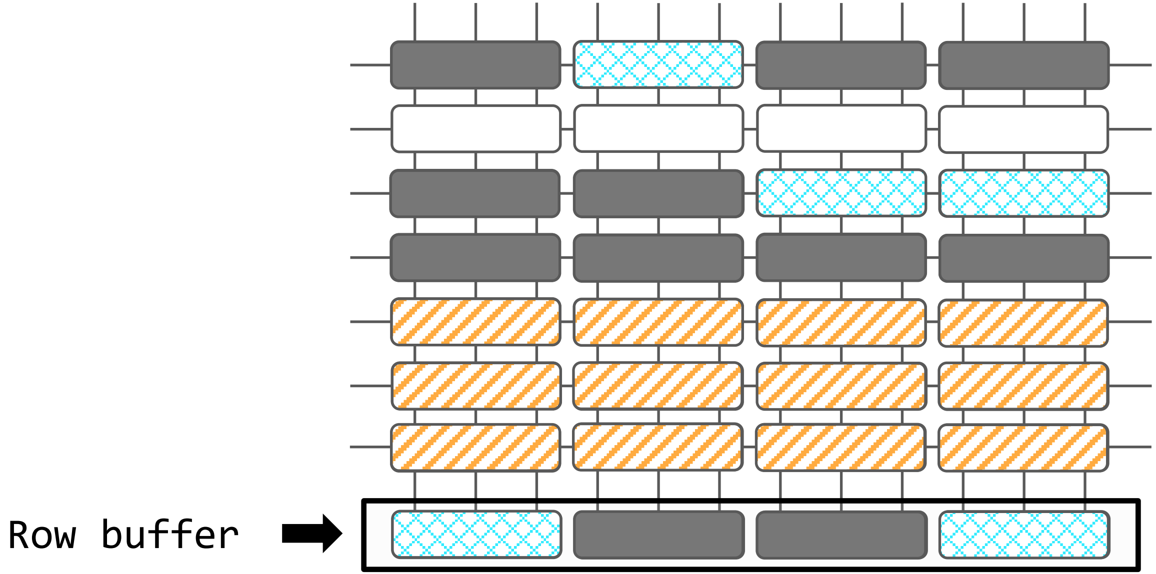


# #P3. Contiguous Memory: Detection

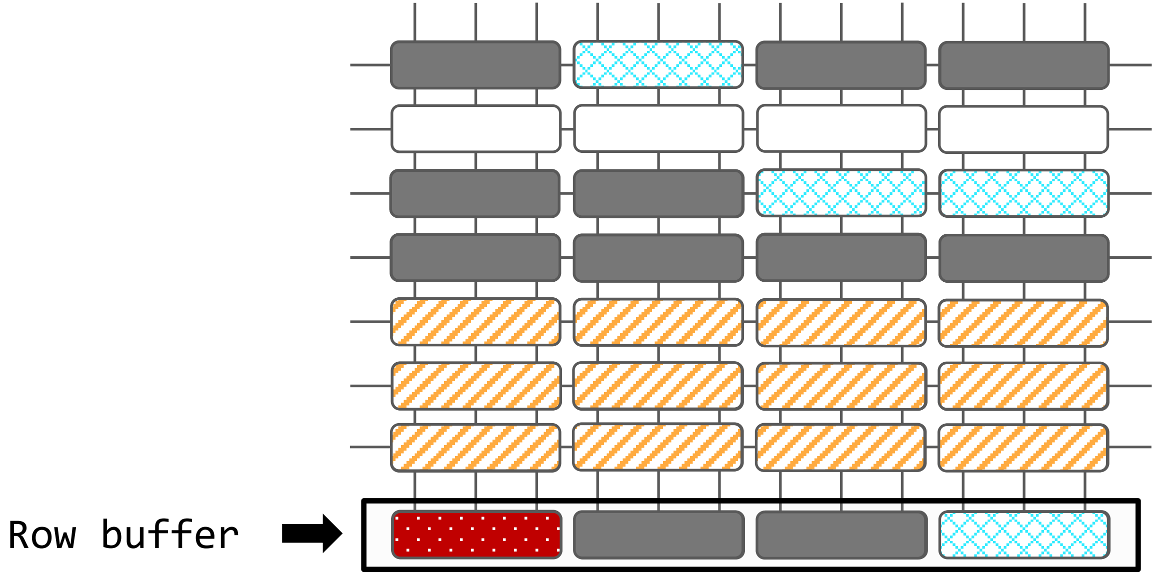


Row buffer

# #P3. Contiguous Memory: Detection

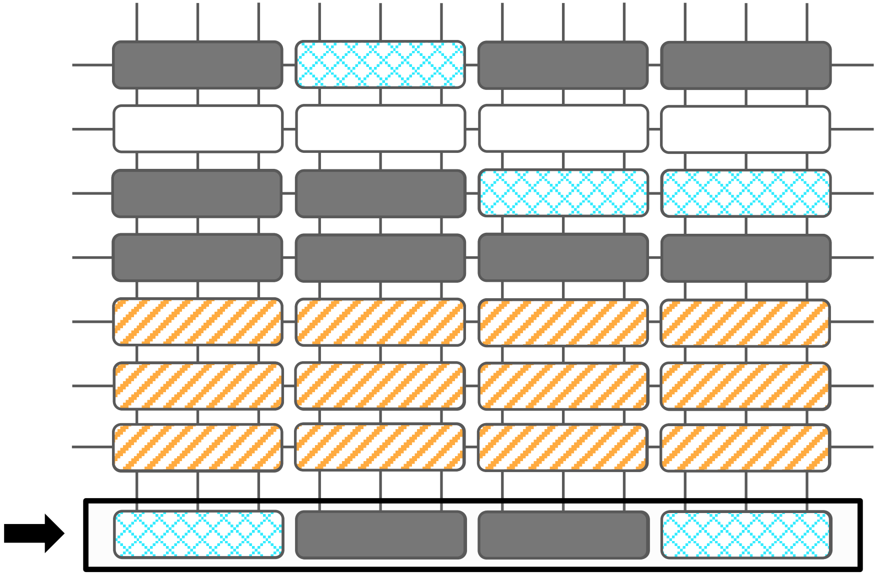


# #P3. Contiguous Memory: Detection

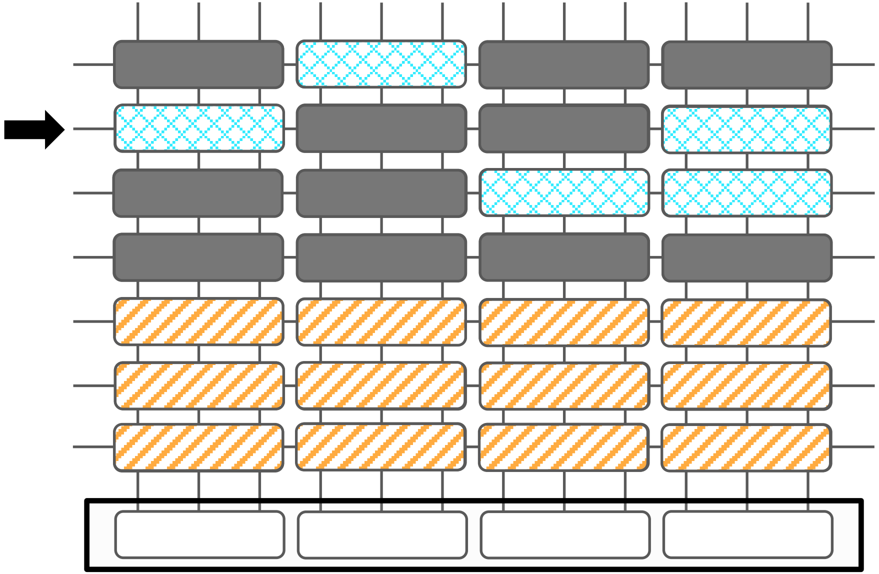


# #P3. Contiguous Memory: Detection

Row buffer

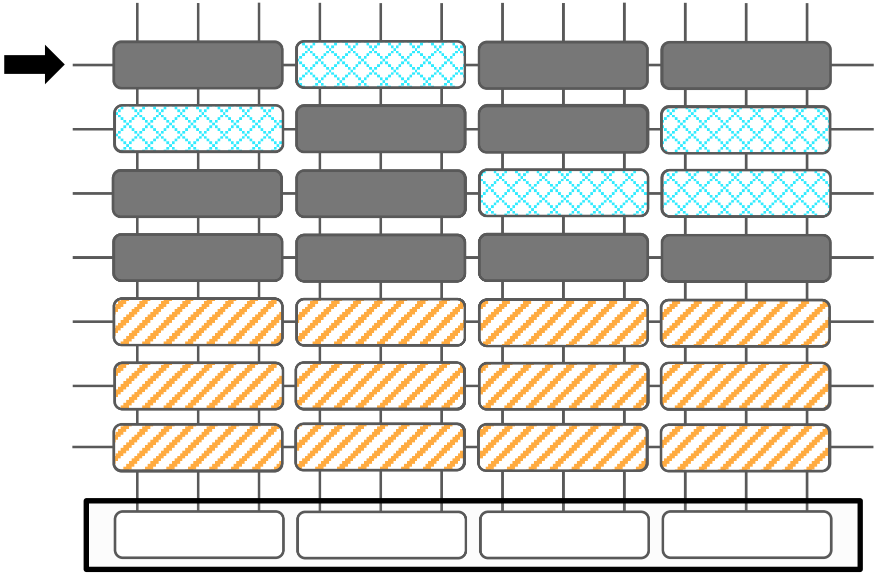


# #P3. Contiguous Memory: Detection



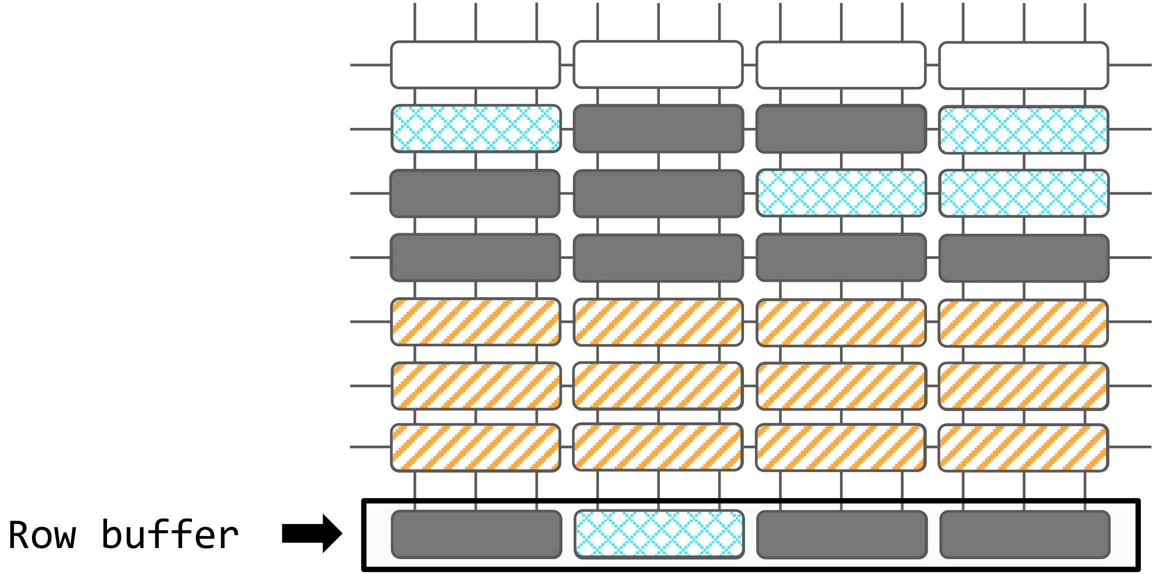
Row buffer

# #P3. Contiguous Memory: Detection

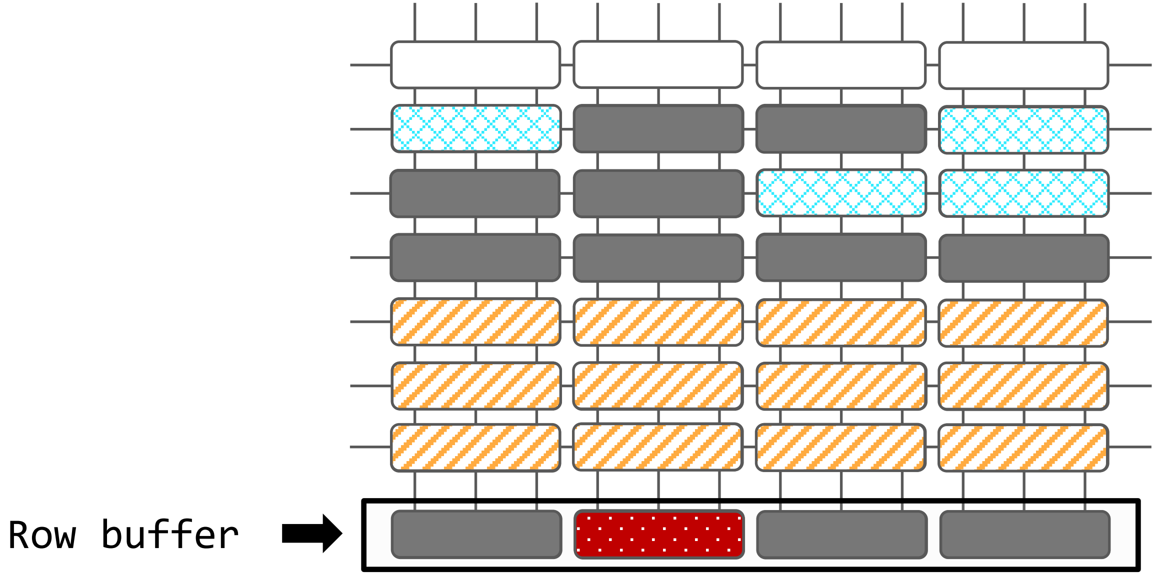


Row buffer

# #P3. Contiguous Memory: Detection

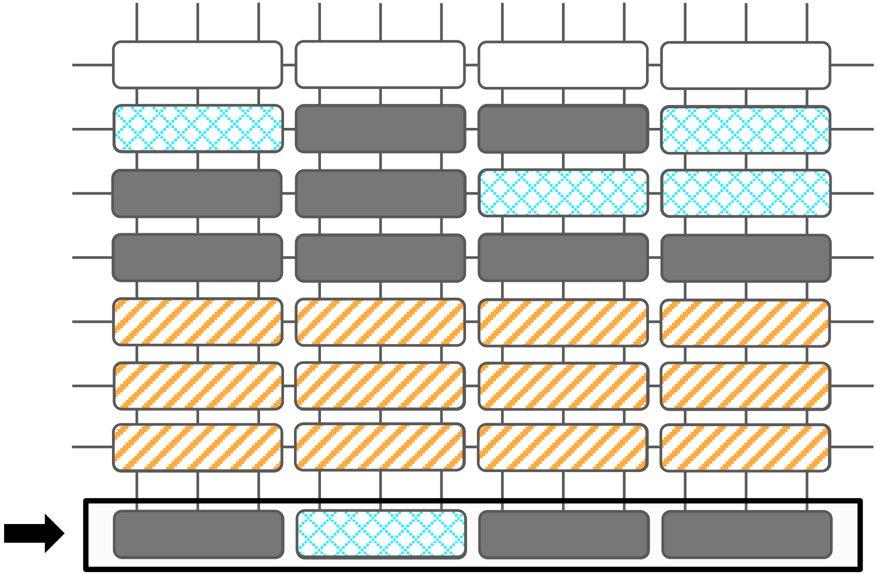


# #P3. Contiguous Memory: Detection

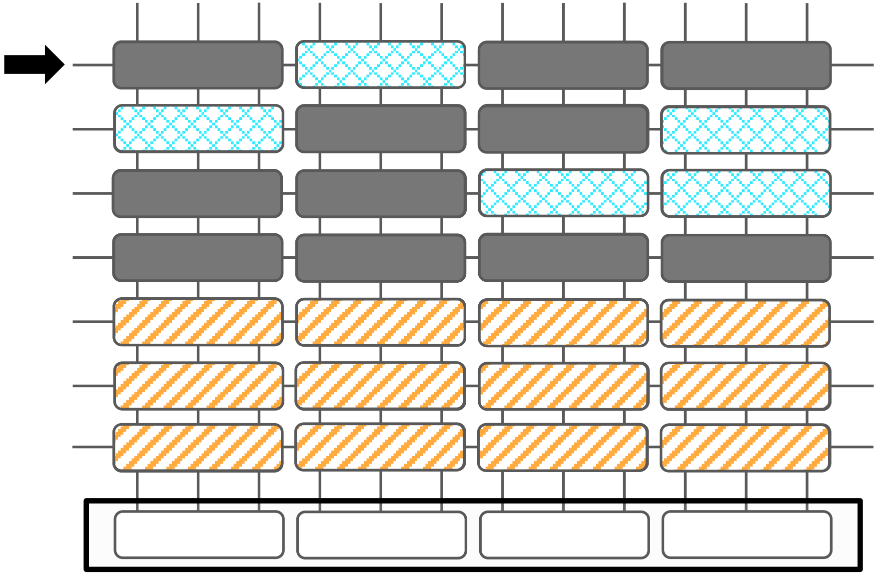


# #P3. Contiguous Memory: Detection

Row buffer

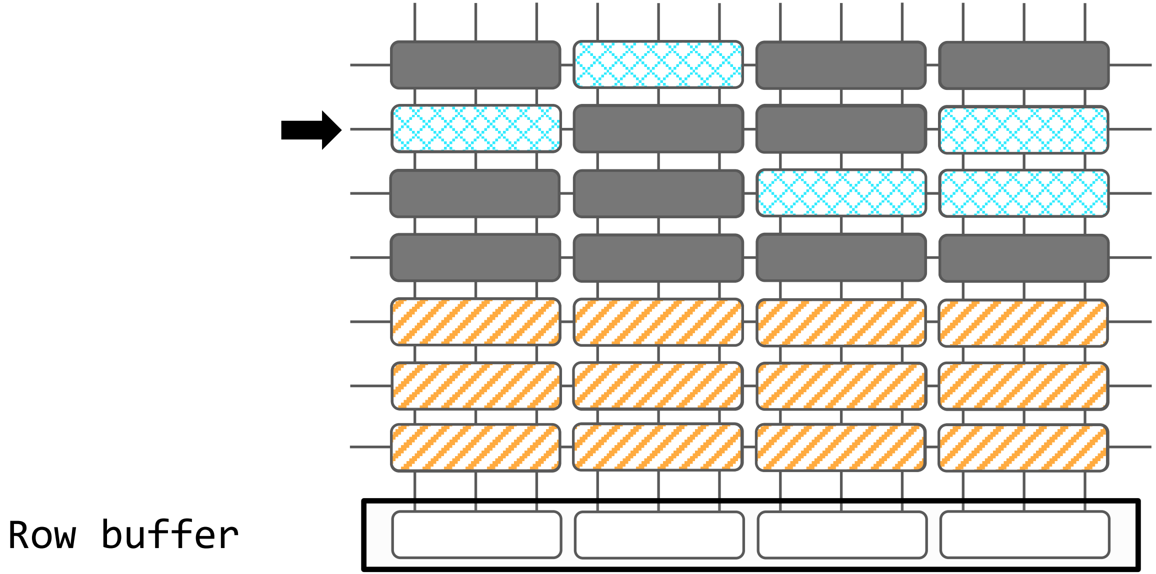


# #P3. Contiguous Memory: Detection

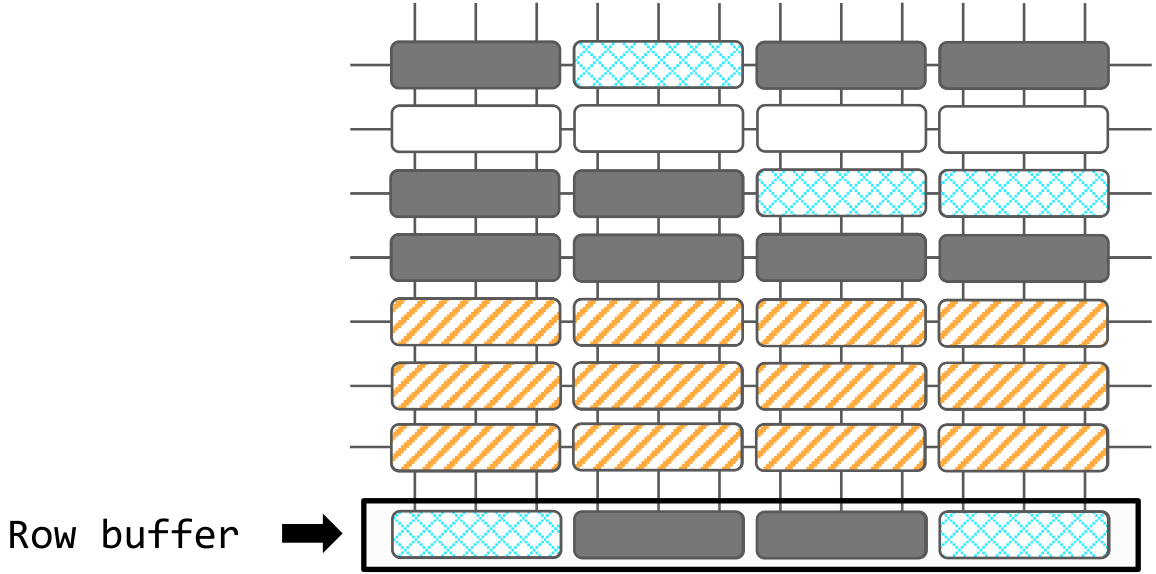


Row buffer

# #P3. Contiguous Memory: Detection

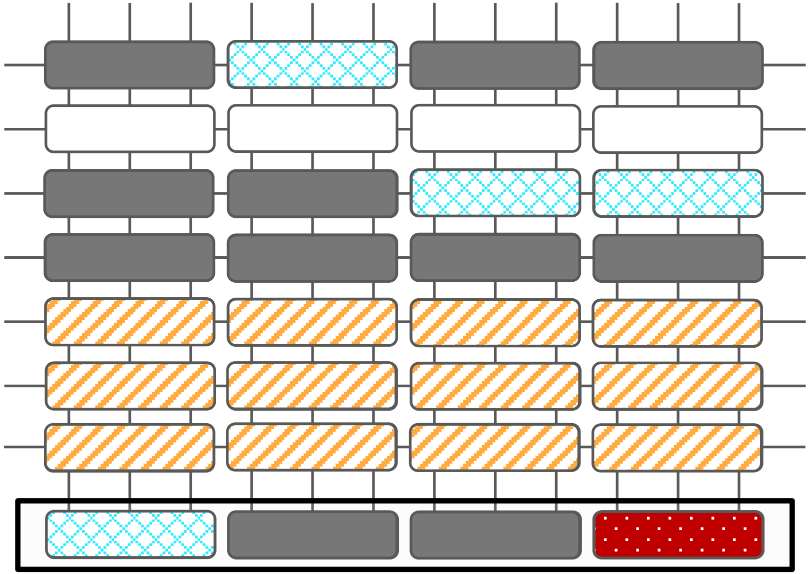


# #P3. Contiguous Memory: Detection

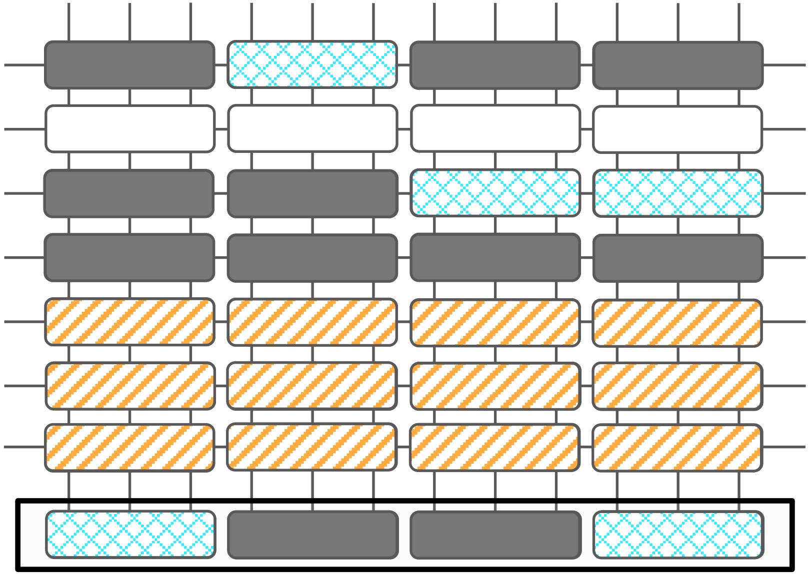


# #P3. Contiguous Memory: Detection

Row buffer

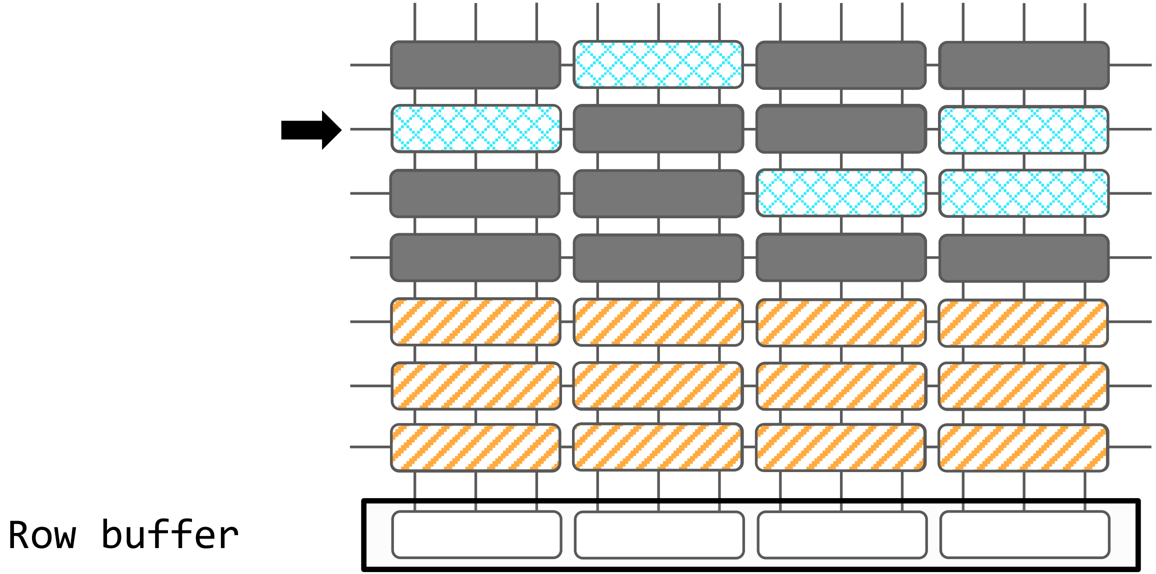


# #P3. Contiguous Memory: Detection

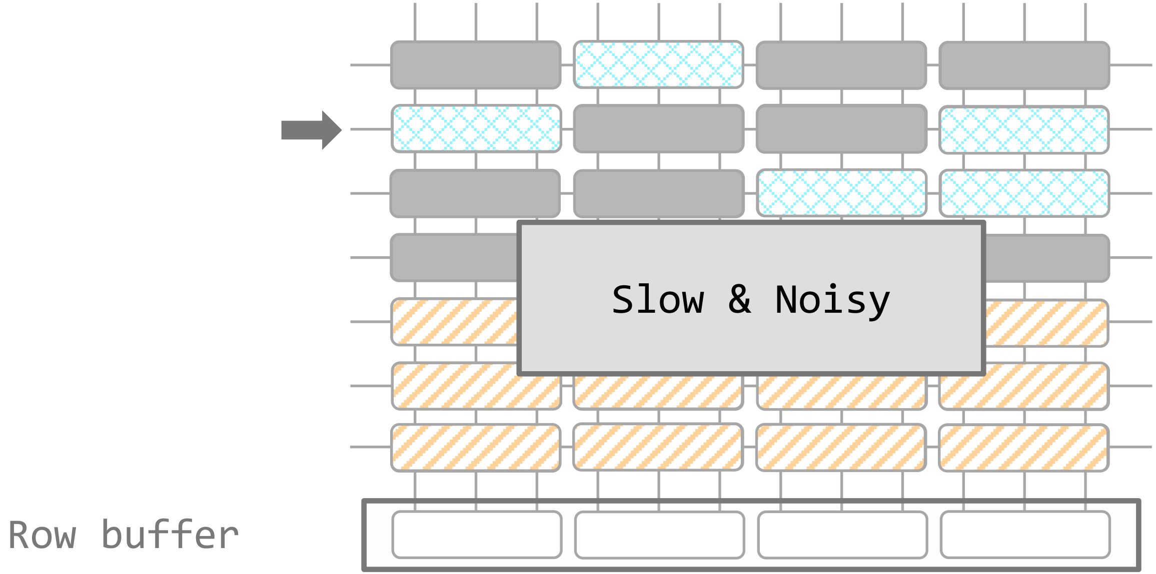


Row buffer

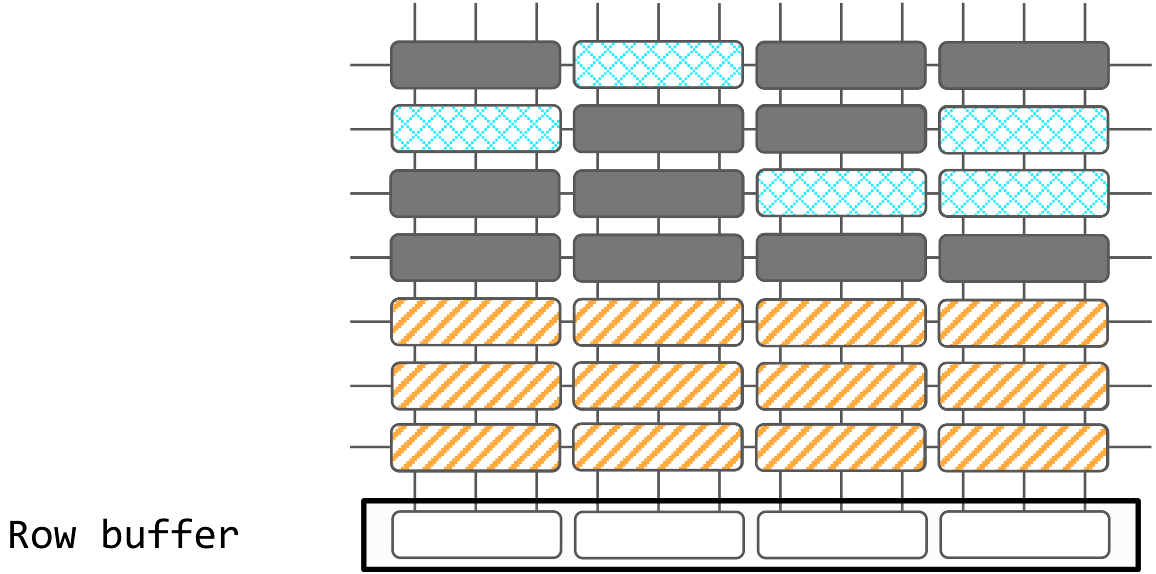
# #P3. Contiguous Memory: Detection



# #P3. Contiguous Memory: Detection

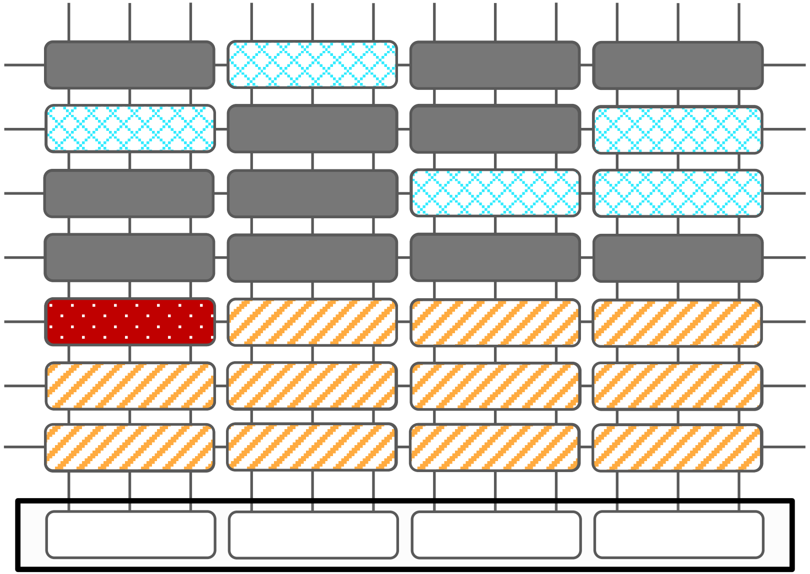


# #P3. Contiguous Memory: Detection



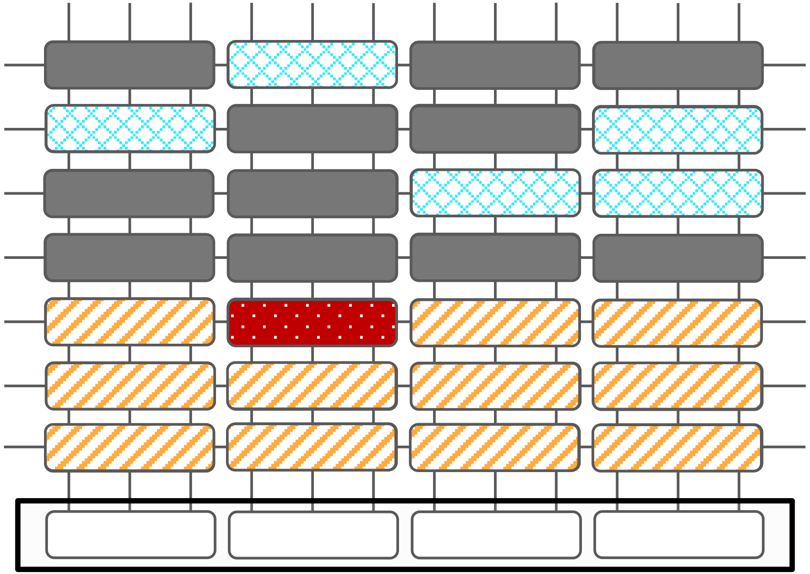
# #P3. Contiguous Memory: Detection

Row buffer

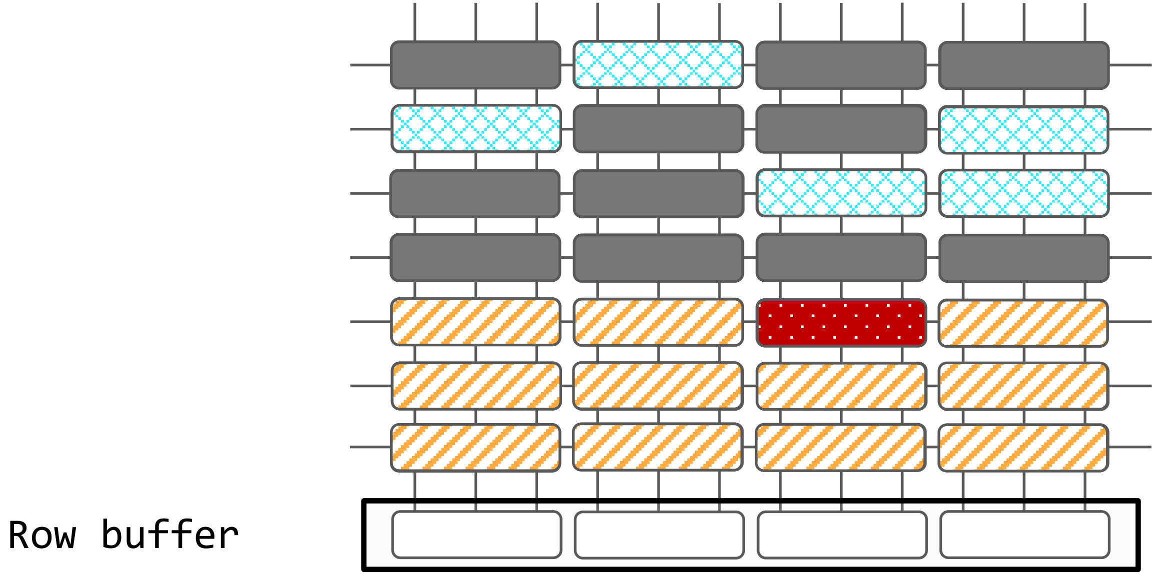


# #P3. Contiguous Memory: Detection

Row buffer

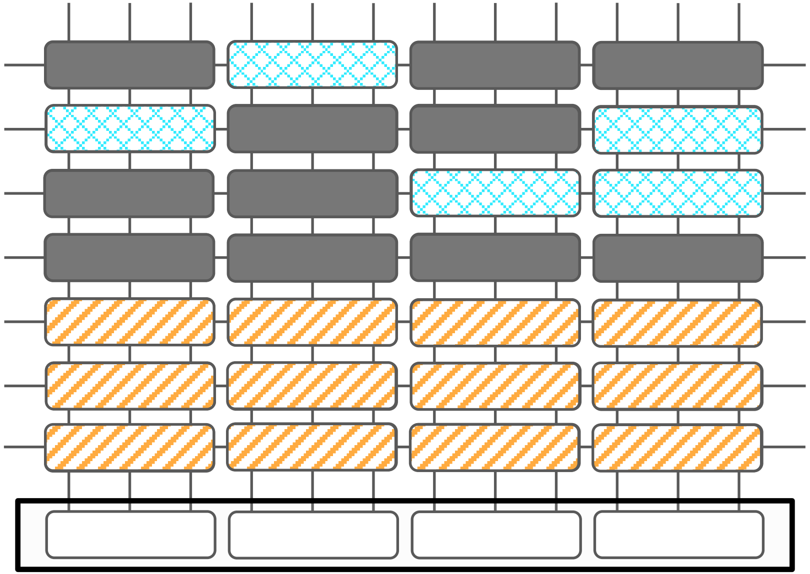


# #P3. Contiguous Memory: Detection

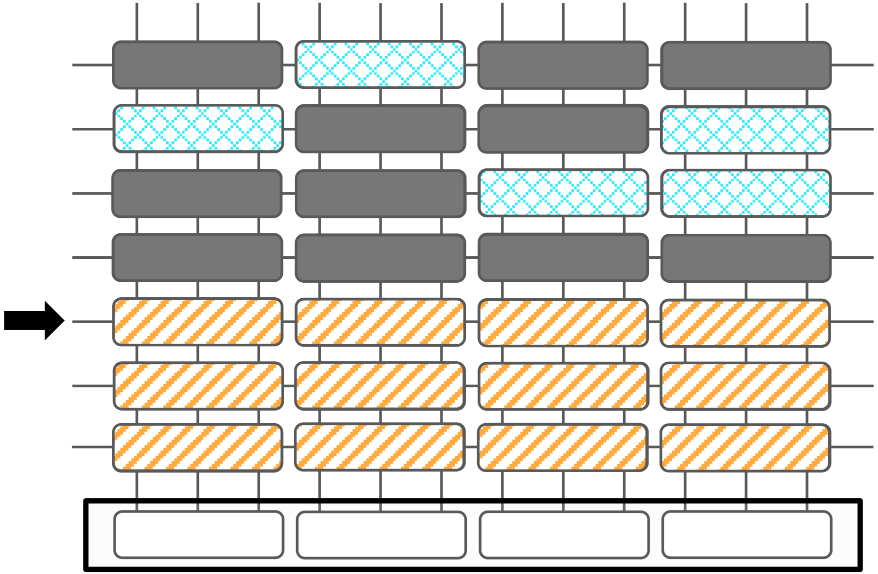


# #P3. Contiguous Memory: Detection

Row buffer

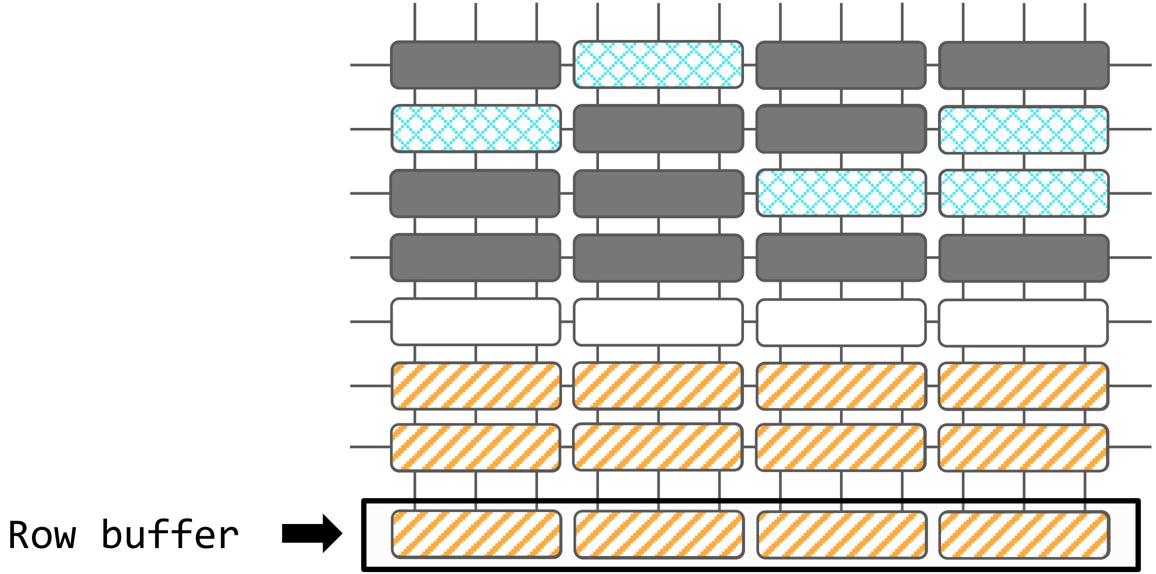


# #P3. Contiguous Memory: Detection



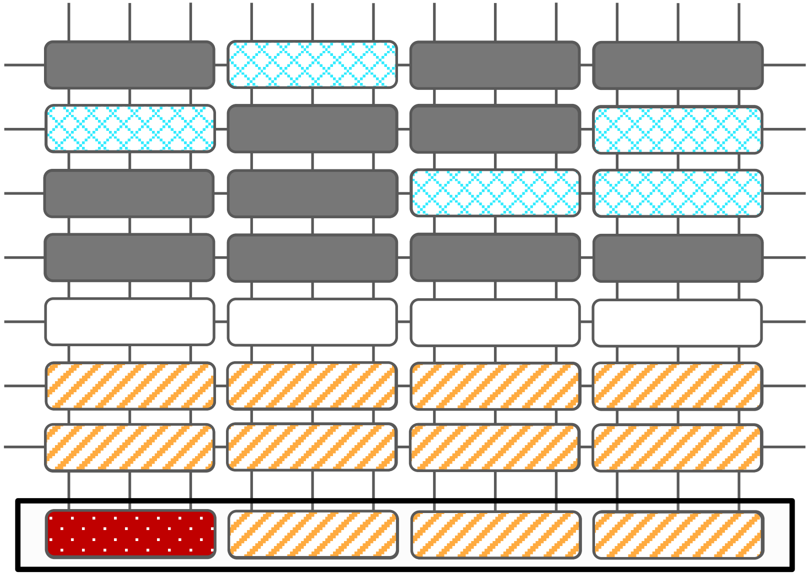
Row buffer

# #P3. Contiguous Memory: Detection

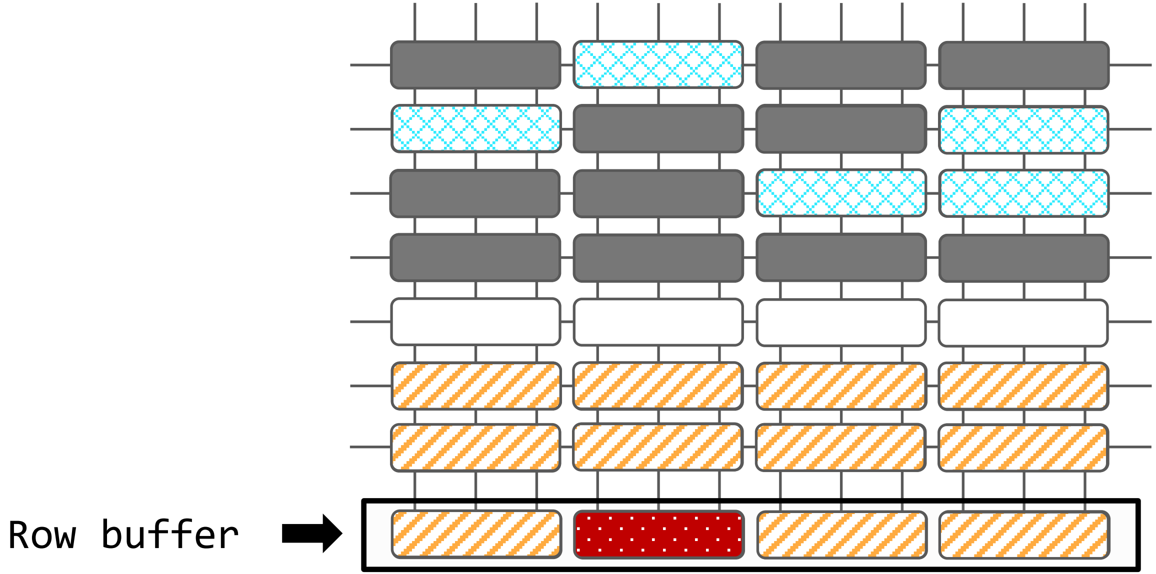


# #P3. Contiguous Memory: Detection

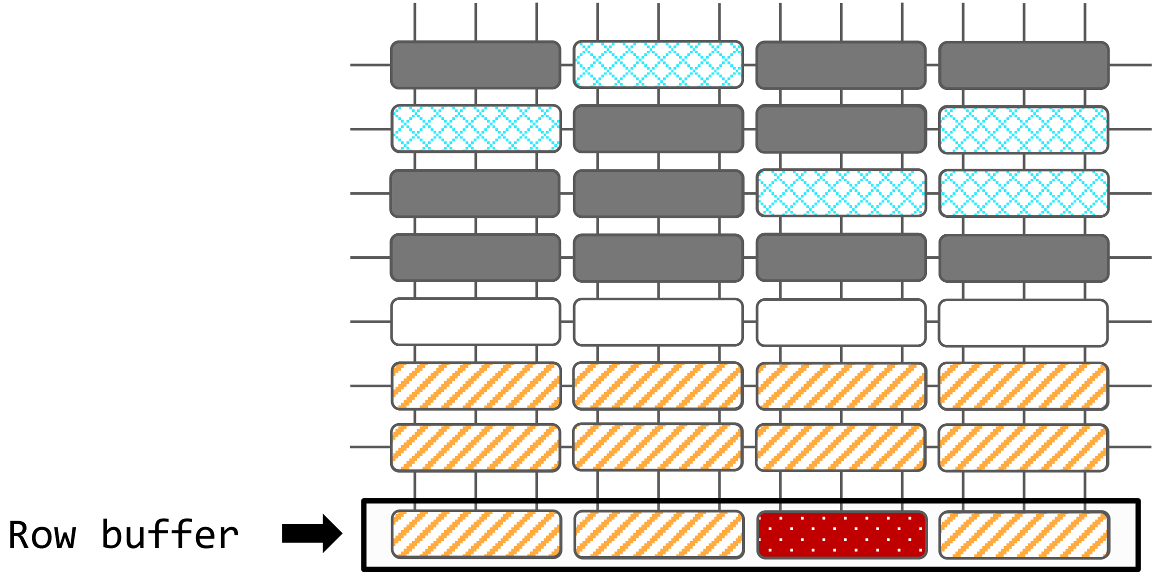
Row buffer



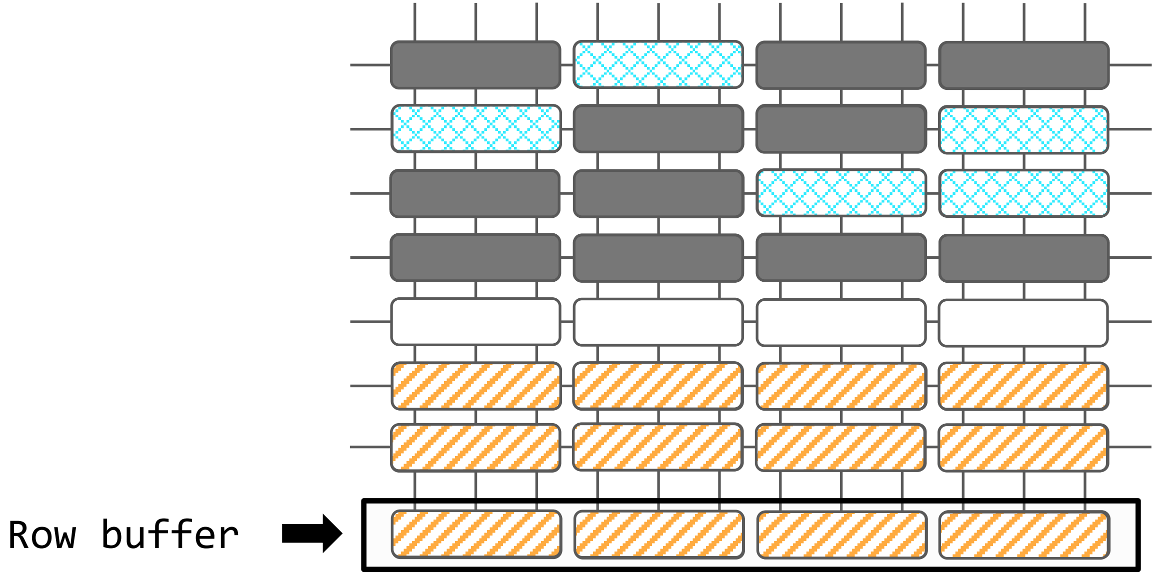
# #P3. Contiguous Memory: Detection



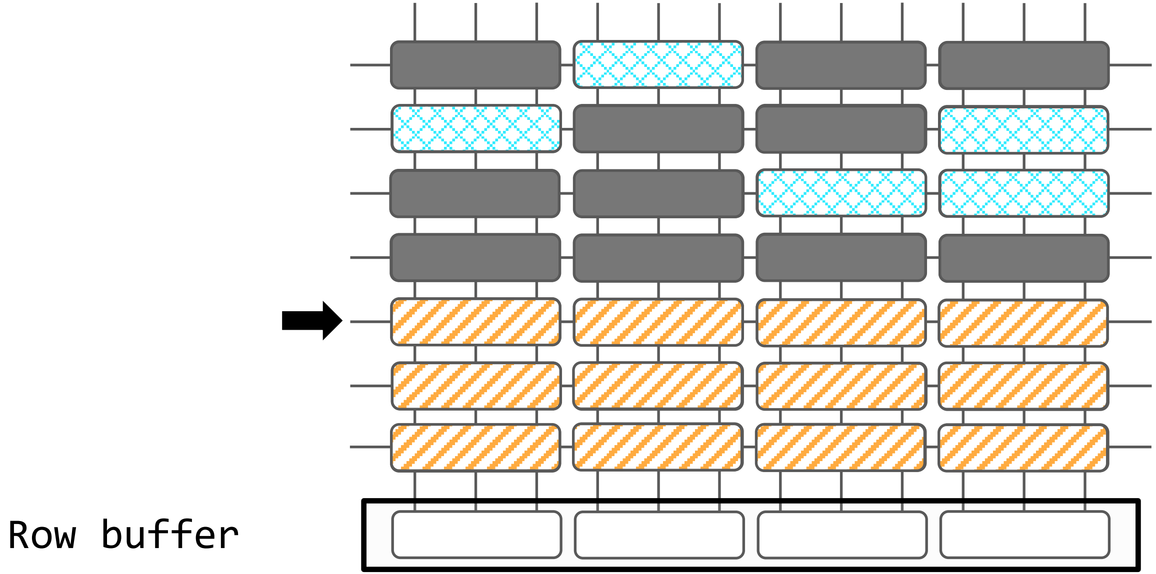
# #P3. Contiguous Memory: Detection



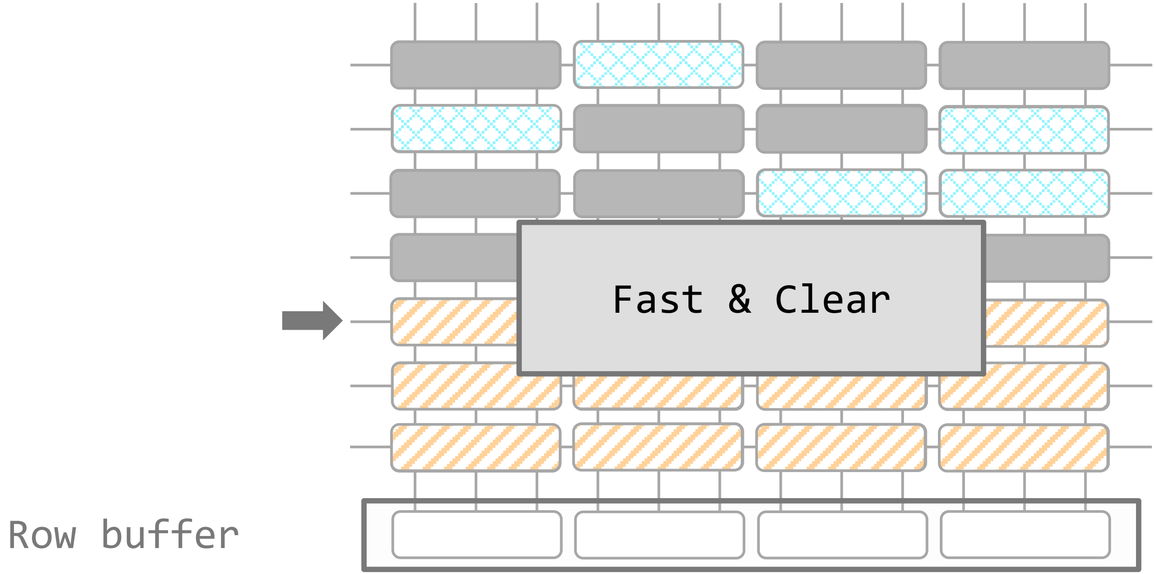
# #P3. Contiguous Memory: Detection



# #P3. Contiguous Memory: Detection



# #P3. Contiguous Memory: Detection



# #P3. WebGL-based timers

**EXT_DISJOINT_TIMER_QUERY**
(Extension - Explicit)

- Similar to clock_gettime()
- High resolution

**WebGLSync**
(WebGL2 - Implicit)

- Sync CPU and GPU
- More coarse-grained

# Attacker primitives

#P1. DRAM access ✓

#P2. Fast memory access ✓

#P3. Contiguous memory ✓

# What do we do with these primitives?

# GLitch

# GLitch: in a nutshell

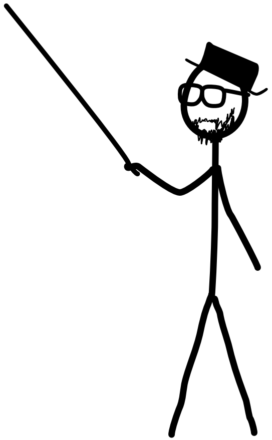
Flip feng shui in JS:

1. Memory templating
2. Memory massaging
3. Exploitation

# GLitch: in a nutshell

Flip feng shui in JS:

1. Memory templating
2. Memory massaging
3. Exploitation



# Exploitation: JS Arrays

| arr[] | |
|---|---|
| 1 | int |
| 1.878e+65 | double |
| *obj | object |
| 0.3 | double |
| *str | string |
| ⋮ | |
| *func | function |
| False | boolean |

```
var arr = new Array(100);
arr[0] = 1 // int
arr[1] = 1.878e+65 // double
arr[3] = new Array(0x12) // object
...
```

**NaN-boxing**

# IEEE-754 floating point (double)

```
1.125  = 1125 * 10^(-3)
           ==      !=
11.25  = 1125 * 10^(-2)
          ↑        ↑
     Significand  exp
```

# IEEE-754 floating point (double)

```
1.125 = 1125 * 10^(-3)
          ==       !=
11.25 = 1125 * 10^(-2)
```

| 1 | 11 | 52 | |
|---|---|---|---|
| 0 | 01111111111 | 0010000000000000000000000000000000000000000000000000 | 1.125 |
| sign | exp | mantissa | |

# IEEE-754 floating point (double)

```
1.125 = 1125 * 10^(-3)
        ==      !=
11.25 = 1125 * 10^(-2)
```

| sign (1) | exp (11) | mantissa (52) | |
|---|---|---|---|
| 0 | 11111111111 | 0010000000000000000000000000000000000000000000000000 | **NaN** |

# IEEE-754 floating point (double)

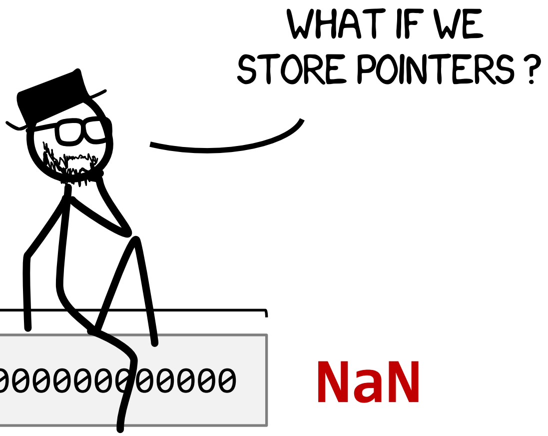
1.125 = 1125 * 10^(-3)

== != 

11.25 = 1125 * 10^(-2)

| 1 | 11 | 52 |
|---|---|---|
| 0 | 11111111111 | 0010000000000000000000000000000000000000000000000000 |
| sign | exp | mantissa |

**NaN**




**2^53-1 unused values**

# Exploitation: NuN-boxing (32bit)

```
                 tag        payload
a = (double) 0x7fffff8c 9a8b7c4d
b = (double) 0xffffff8c 9a8b7c4d
                    64 bits
```

# Exploitation: NuN-boxing (32bit)

tag payload

a = (double) 0x7ffffff8c 9a8b7c4d

b = (double) 0xffffff8c 9a8b7c4d

64 bits

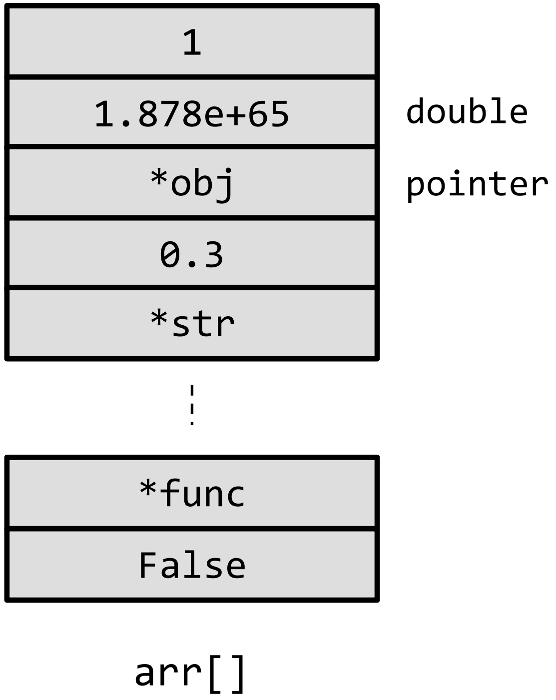
(tag < 0xffffff80)

true false

double *object

# Exploitation: NuN-boxing (32bit)

```
arr[1] = 0x7fffff8c 9a8b7c4d
arr[2] = 0xffffff8c 9a8b7c4d
```



# Exploitation: NuN-boxing (32bit)

```
arr[1] = 0x7ffff8c9a8b7c4d
arr[2] = 0xfffff8c9a8b7c4d
```

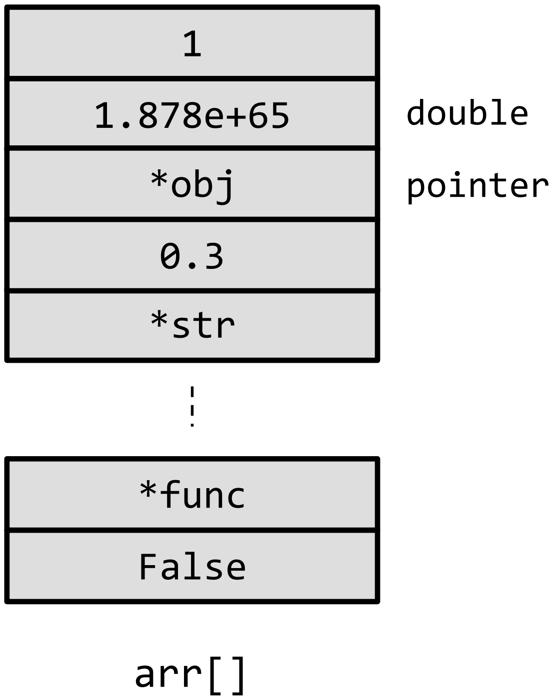
SAME PAYLOD
1-BIT DIFFERENCE IN TAG

| arr[] | |
|---|---|
| 1 | |
| 1.878e+65 | double |
| *obj | pointer |
| 0.3 | |
| *str | |
| ... | |
| *func | |
| False | |

# Exploitation: Type Flipping

arr[1] = 0x7fffff8c 9a8b7c4d

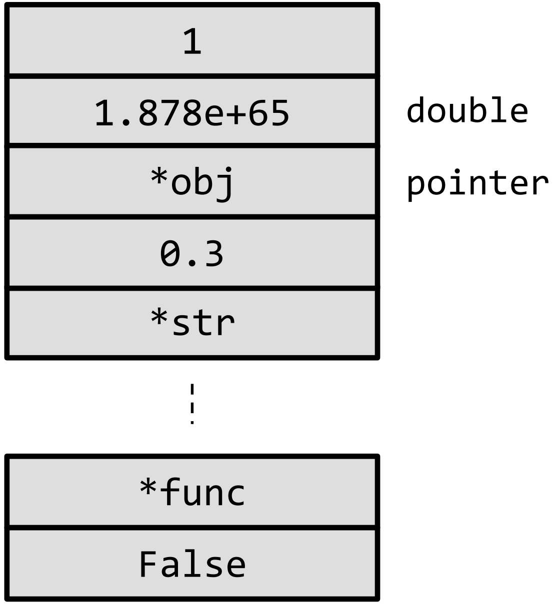
arr[2] = 0xffffff8c 9a8b7c4d



# Exploitation: Type Flipping

arr[1] = 0x7fffff8c9a8b7c4d

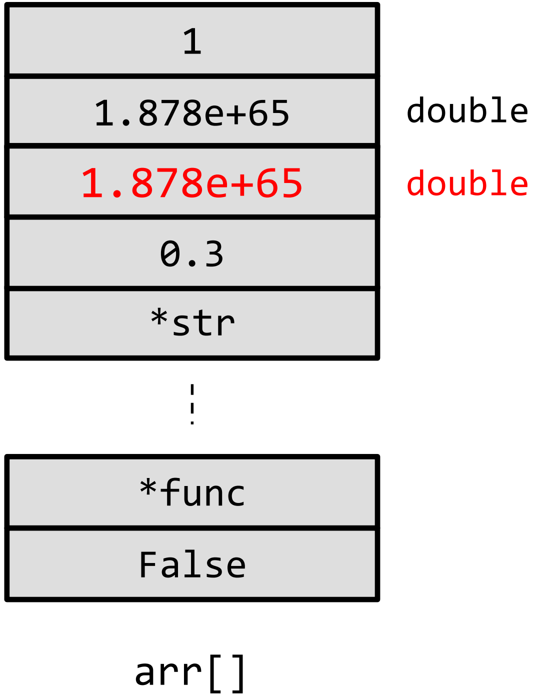
arr[2] = 0x7fffff8c9a8b7c4d




arr[]

# Exploitation: Type Flipping

```
arr[1] = 0x7fffff8c 9a8b7c4d
arr[2] = 0x7fffff8c 9a8b7c4d
```



# Exploitation: Type Flipping

```
arr[1] = 0xffffff8c9a8b7c4d
arr[2] = 0x7fffff8c9a8b7c4d
```

| | |
|---|---|
| 1 | |
| 1.878e+65 | double |
| 1.878e+65 | double |
| 0.3 | |
| *str | |
| ⋮ | |
| *func | |
| False | |

arr[]

# Exploitation: Type Flipping

```
arr[1] = 0xffffff8c9a8b7c4d
arr[2] = 0x7fffff8c9a8b7c4d
```

| arr[] | |
|---|---|
| 1 | |
| *obj | pointer |
| 1.878e+65 | double |
| 0.3 | |
| *str | |
| ⋮ | |
| *func | |
| False | |

# Exploitation: Type Flipping

```
arr[1] = 0xffffff8c 9a8b7c4d
arr[2] = 0x7fffff8c 9a8b7c4d
```

2 Primitives:

#1 Arbitrary Leak [1-to-0]

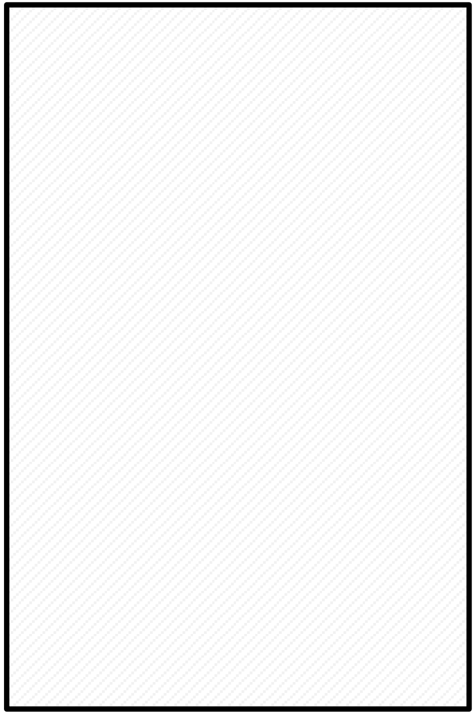
#2 Arbitrary Craft [0-to-1]

| |
|---|
| 1 |
| *obj (pointer) |
| 1.878e+65 (double) |
| 0.3 |
| *str |
| ⋮ |
| *func |
| False |

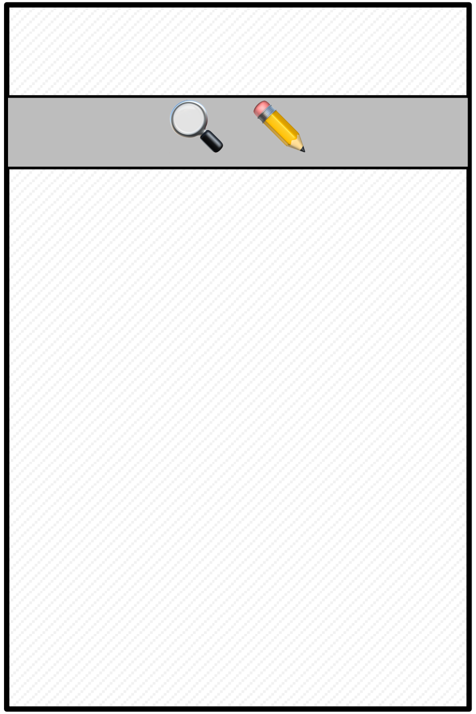
arr[]

# Exploitation: Arbitrary R/W

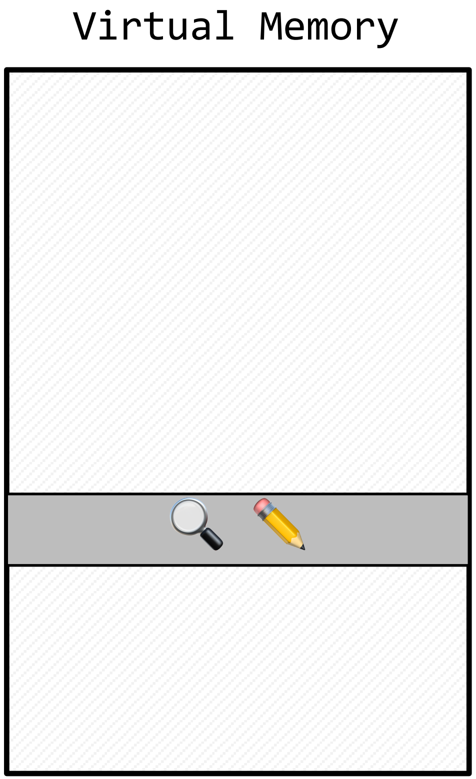
Virtual Memory



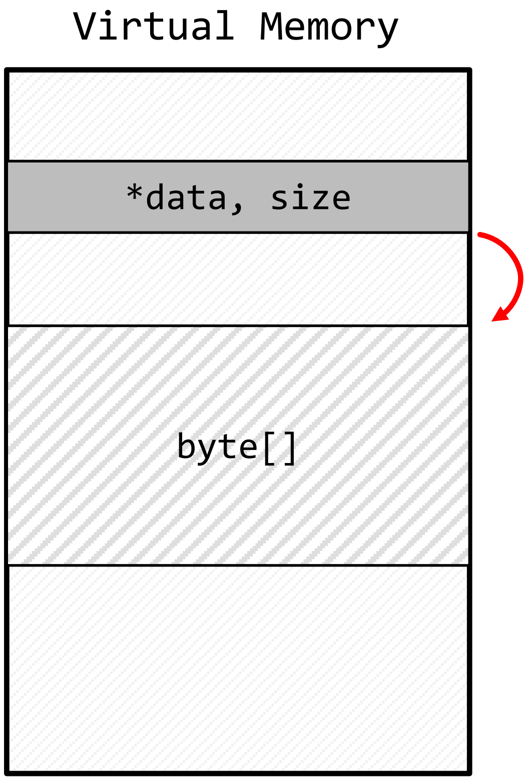
# Exploitation: Arbitrary R/W



# Exploitation: Arbitrary R/W

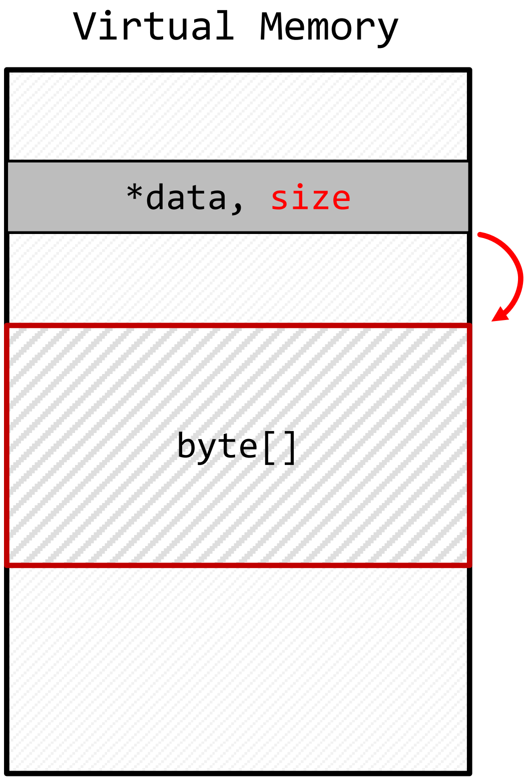


# Exploitation: Arbitrary R/W



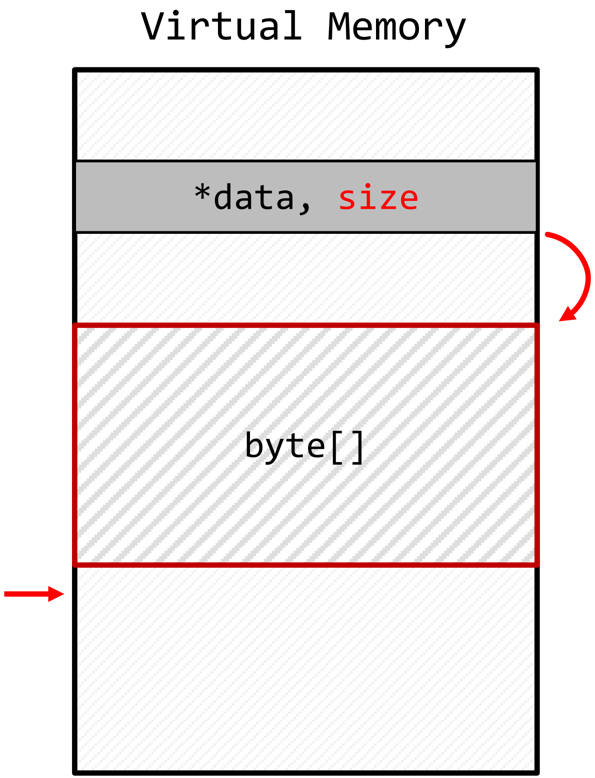

```
var buff = new ArrayBuffer(100);
```

# Exploitation: Arbitrary R/W



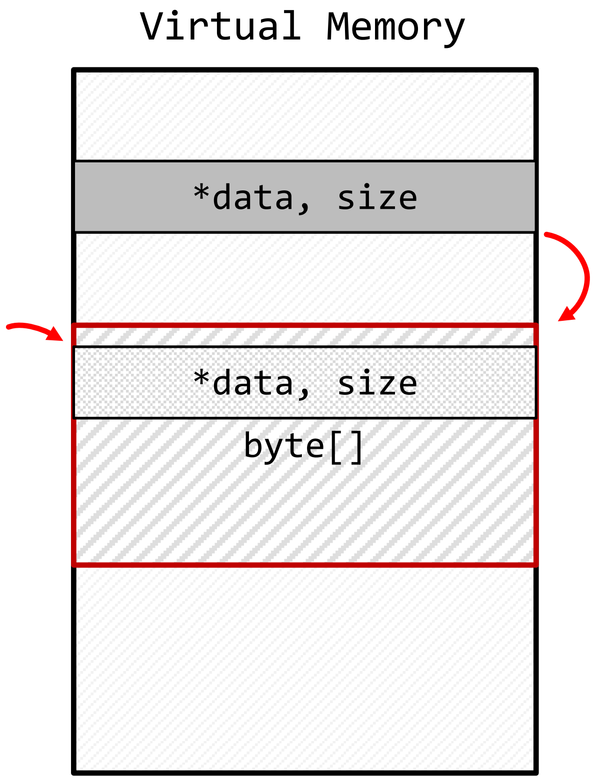

```
var buff = new ArrayBuffer(100);
```

# Exploitation: Arbitrary R/W




```
var buff = new ArrayBuffer(100);
val = buff[108]; // ERROR OoB!
```

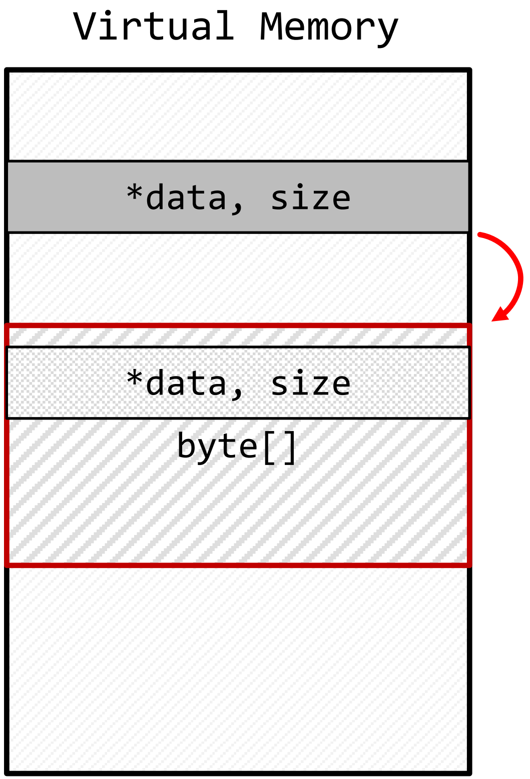
# Exploitation: Arbitrary R/W




```
var buff = new ArrayBuffer(100);
val = buff[108]; // ERROR OoB!
```

```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

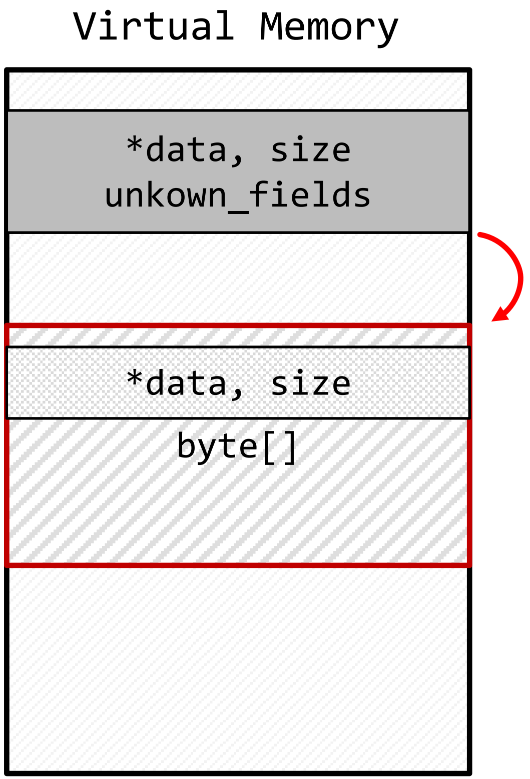
# Exploitation: Arbitrary R/W




```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

Challenges:

# Exploitation: Arbitrary R/W

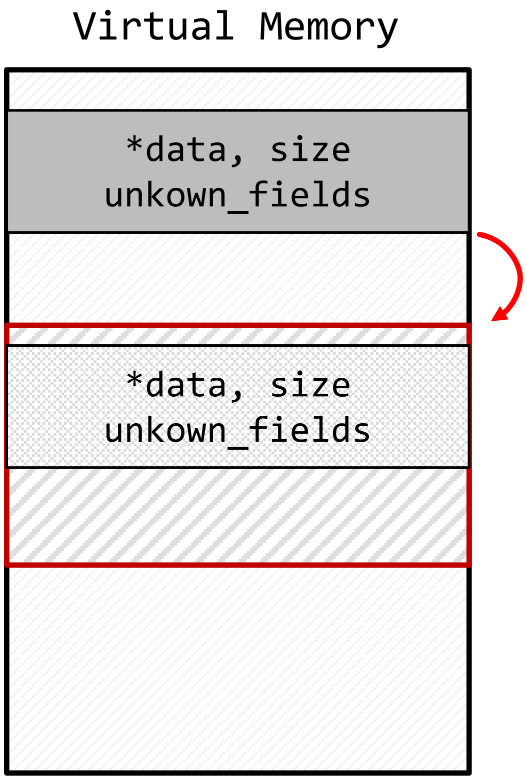



```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

Challenges:

- unknown header fields (e.g., GC root)

# Exploitation: Arbitrary R/W

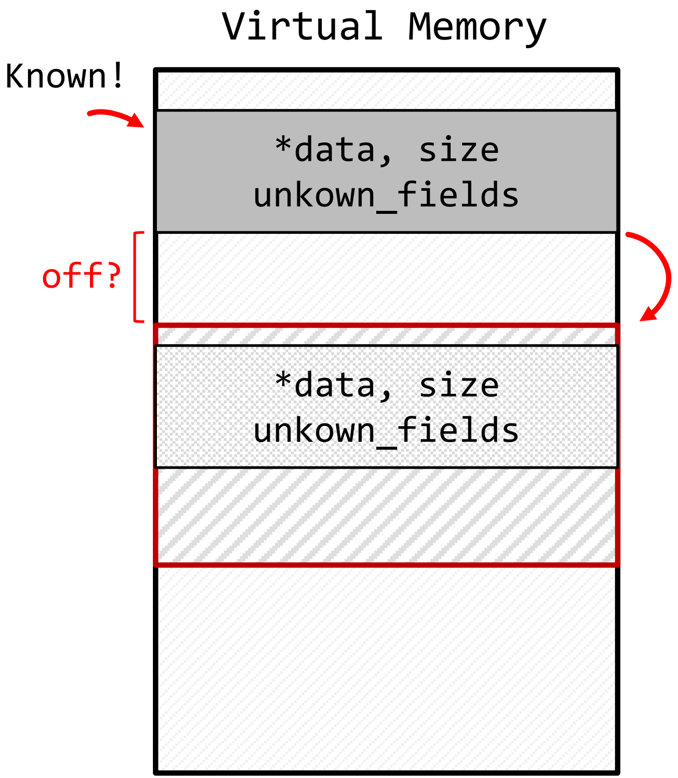



```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

Challenges:

- unknown header fields (e.g., GC root)

# Exploitation: Arbitrary R/W

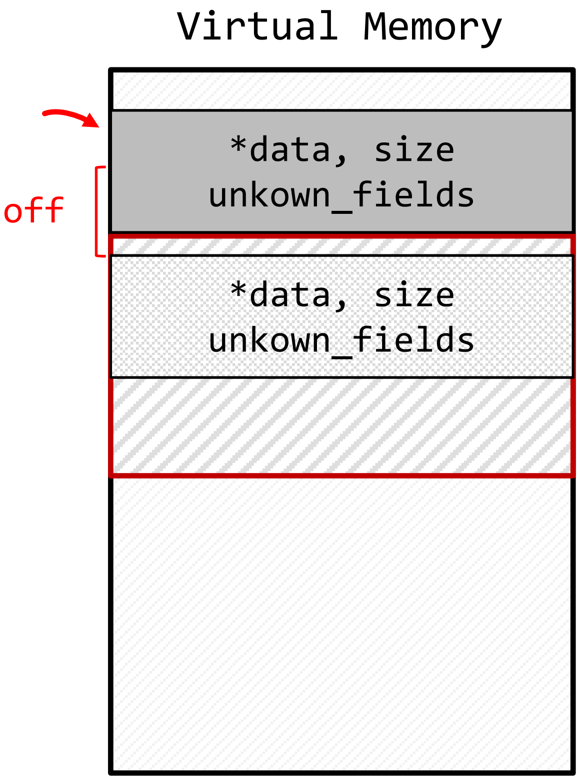



```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

Challenges:

- unknown header fields (e.g., GC root)
- unknown data location

# Exploitation: Arbitrary R/W




```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

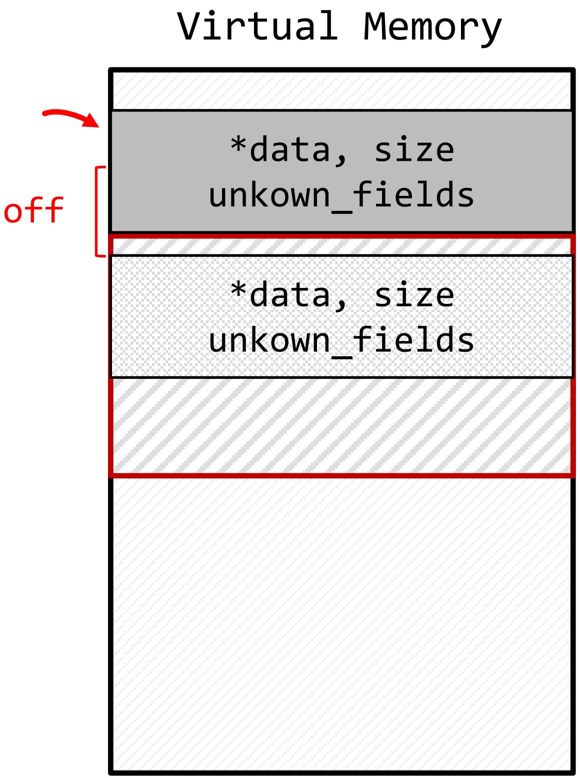
Challenges:

- unknown header fields (e.g., GC root)
- ✓ unknown data location

↓

Inline ArrayBuffers

# Exploitation: Arbitrary R/W




```
buff[K] = create_fake_buff();
var fake_buff = *buff[K];
```

Challenges:

- unknown header fields (e.g., GC root)

✓ unknown data location

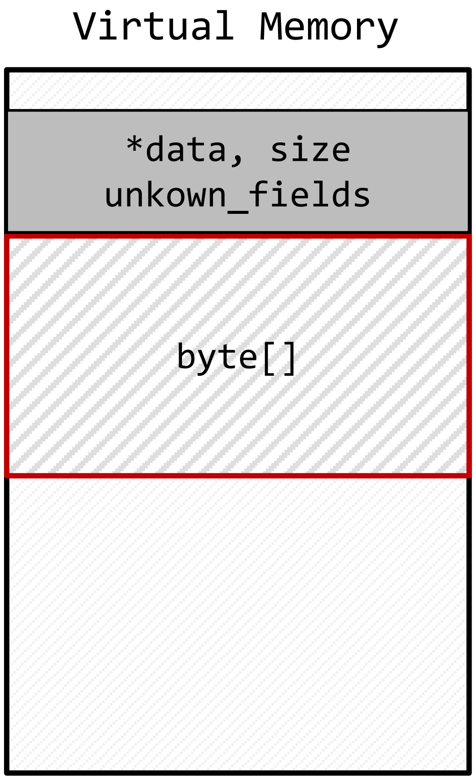
↓

Inline ArrayBuffers

# Exploitation: Arbitrary read

```
class JSString ==> UTF-16 strings       [0x0000 – 0xffff]
  {
    ✓uint32_t flags; // type of string
    ✓uint32_t length; // sizeof(buff_header) 0x30
    ✓char16_t* string;
                                 *buff_header
  }
```

# Exploitation: Arbitrary R/W

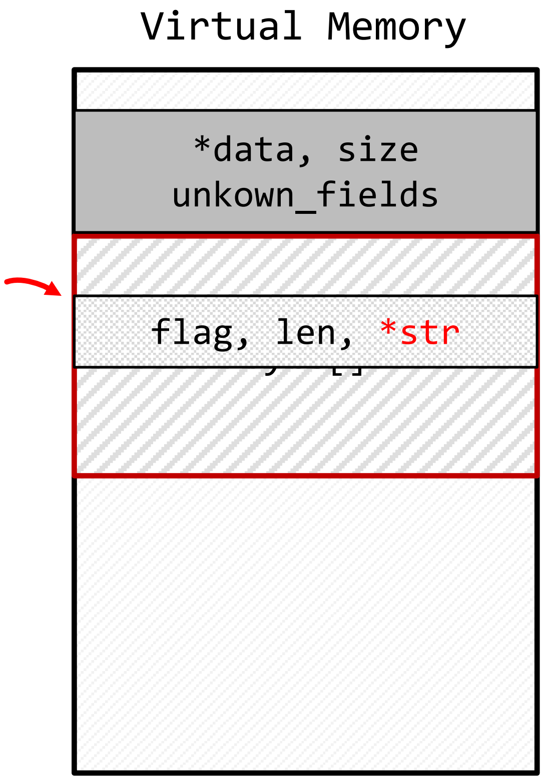



```
buff[0] = create_fake_buff();
var fake_buff = *buff[0];
```

Challenges:

- unknown header fields (e.g., GC root)
- ✓ unknown data location

# Exploitation: Arbitrary R/W



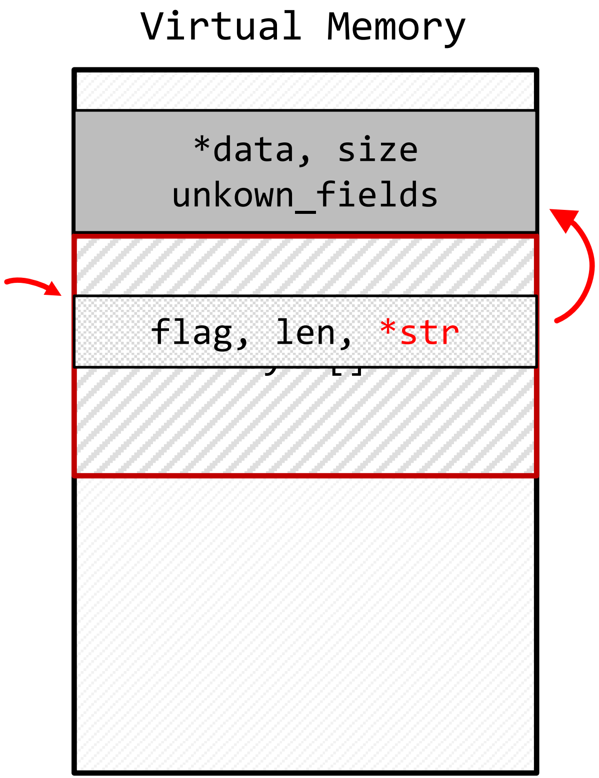

```
buff[0] = create_fake_buff();
var fake_buff = *buff[0];
```

Challenges:

- unknown header fields (e.g., GC root)

✓ unknown data location

# Exploitation: Arbitrary R/W

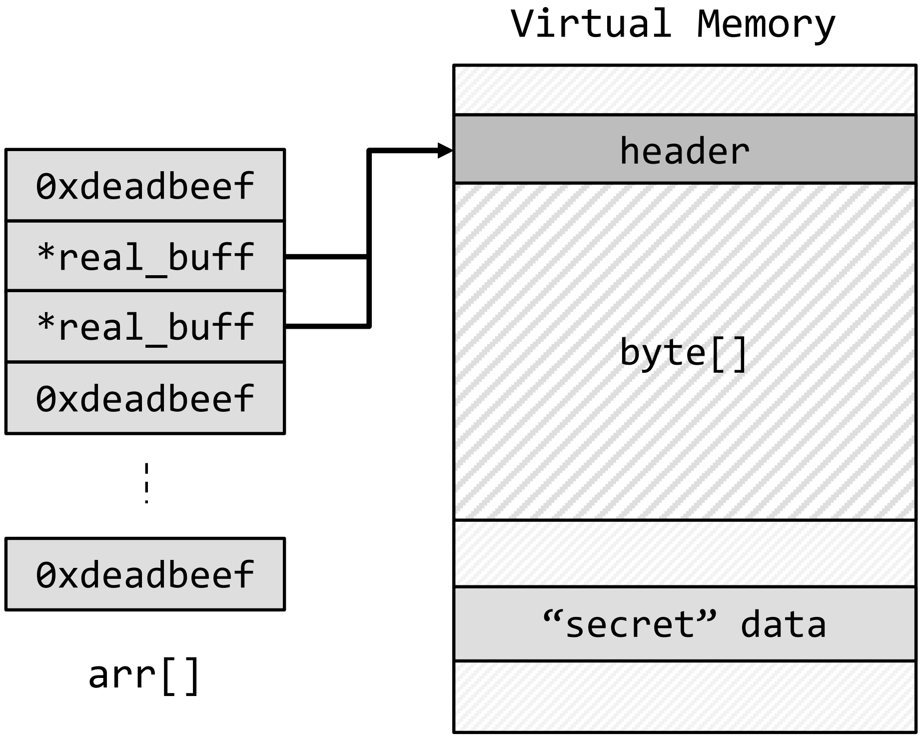



```
buff[0] = create_fake_buff();
var fake_buff = *buff[0];
```

Challenges:

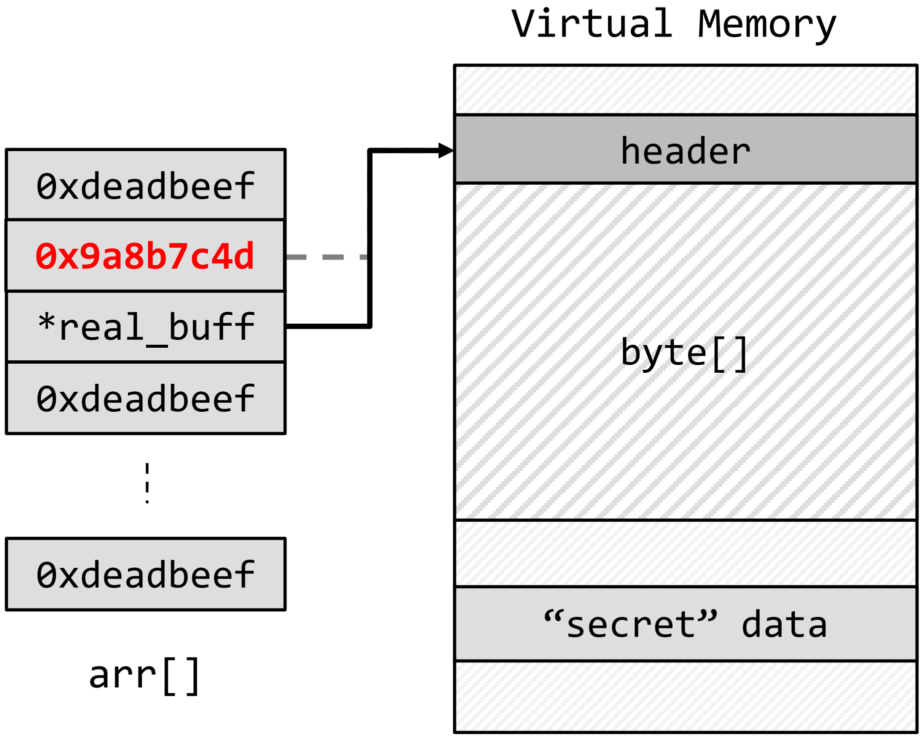
- ✓ unknown header fields (e.g., GC root)
- ✓ unknown data location

# Exploitation: Recap




Exploit in **3 stages:**

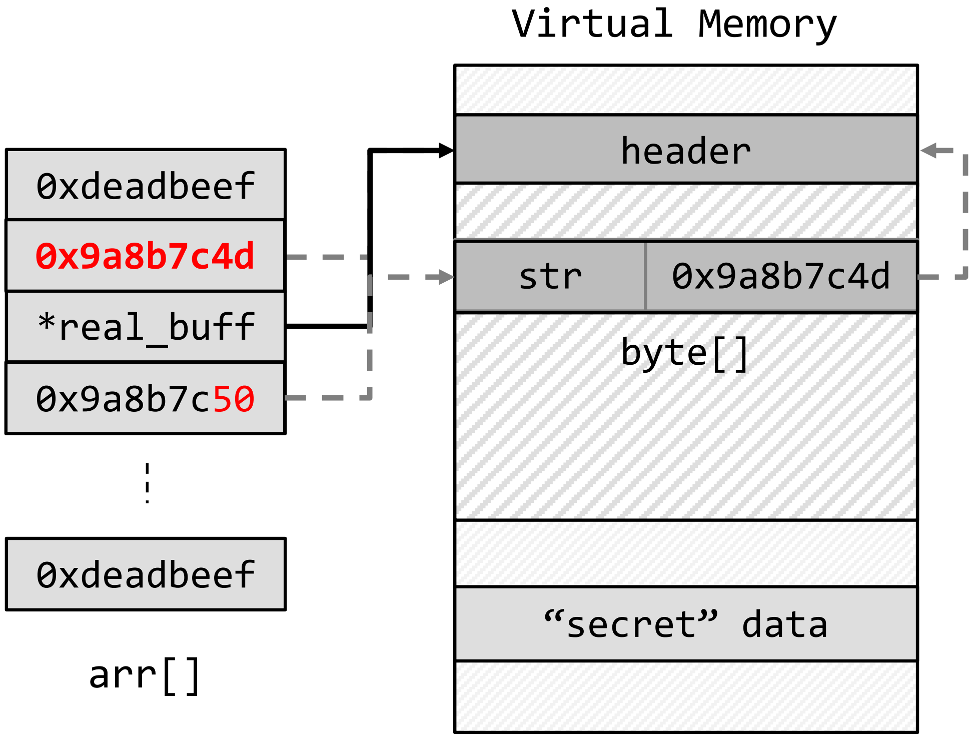
# Exploitation: Recap




Exploit in **3 stages**:

1. Break ASLR

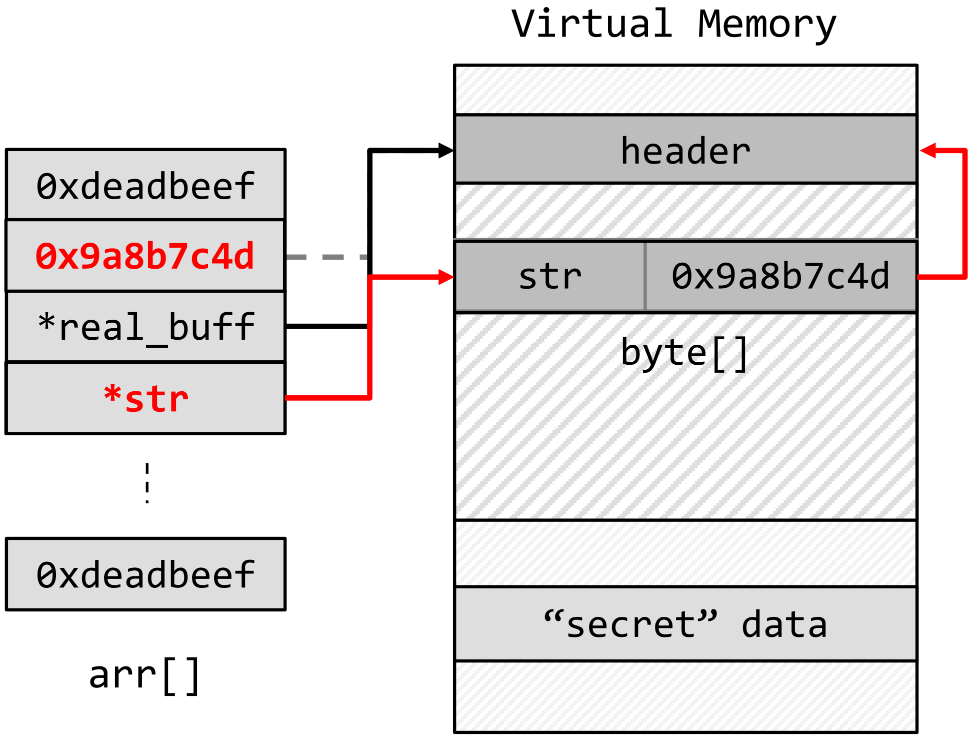
# Exploitation: Recap




Exploit in **3 stages**:

1. Break ASLR
2. Arbitrary read

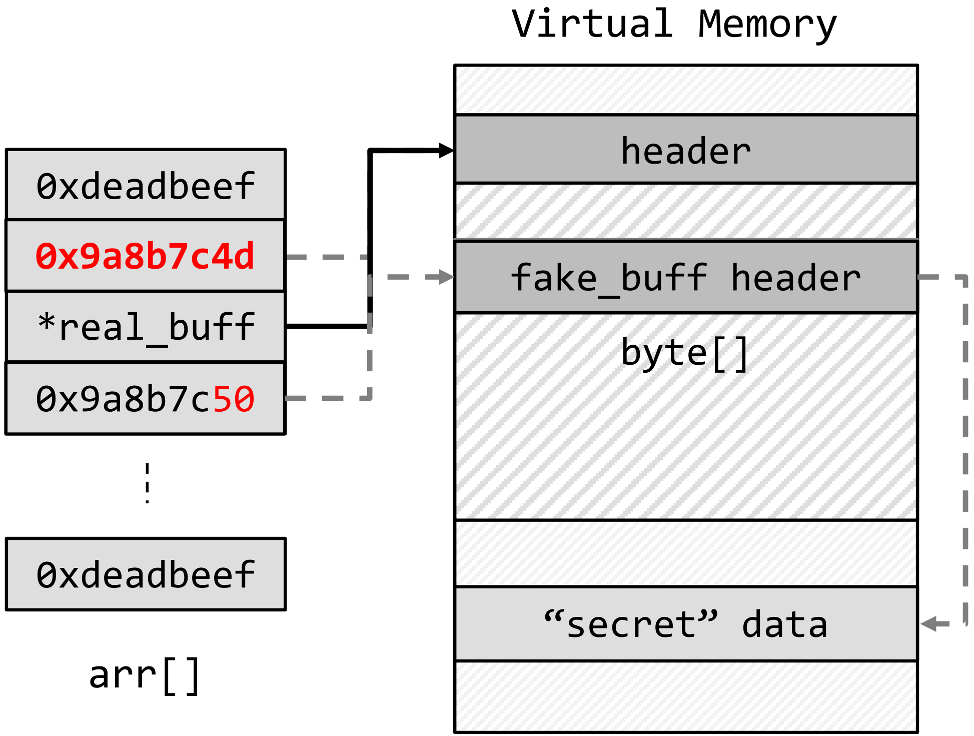
# Exploitation: Recap




Exploit in **3 stages**:

1. Break ASLR
2. Arbitrary read

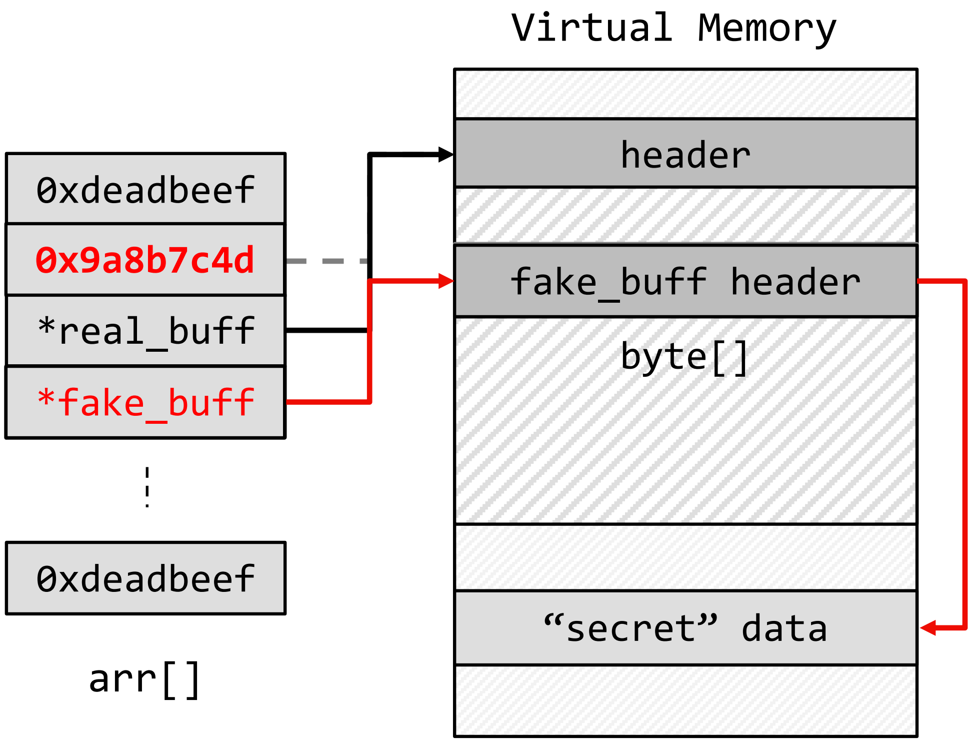
# Exploitation: Recap




Exploit in **3 stages**:

1. Break ASLR
2. Arbitrary read
3. Arbitrary write

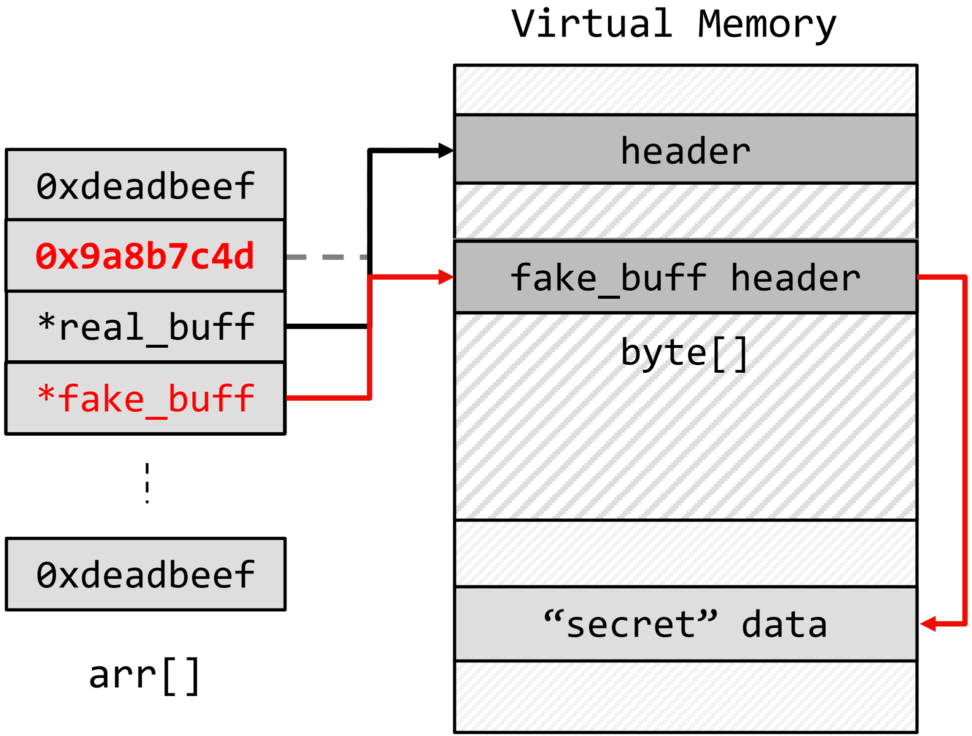
# Exploitation: Recap




Exploit in **3 stages**:

1. Break ASLR
2. Arbitrary read
3. Arbitrary write

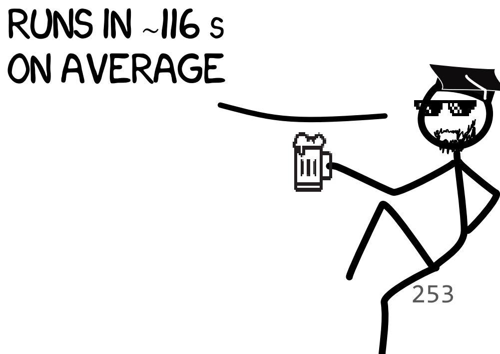
# Exploitation: Recap




Exploit in **3 stages**:

1. Break ASLR ==> 1-to-0
2. Arbitrary read
3. Arbitrary write

$\left.\begin{array}{l}2.\\3.\end{array}\right\}$ 0-to-1



# Demo

# Disclosure & Mitigations

- Disclosure process with the help of Dutch NCSC (CVE-2018-10229)
- Chrome & Firefox released partial mitigations **against timers**. (Will be reenabled soon ¯_(ツ)_/¯)

# Conclusions

- **First** Rowhammer attack from JS on mobile
- **GPU** as new attack vector
- Takeaway: Redefine the **threat model** (DSP, FPGAs, …)

https://www.vusec.net/projects/glitch/

@vu5ec / @pit_frg