GLitch:

when WebGL turns into a hammer

Pietro Frigo

Cristiano Giuffrida, Herbert Bos, Kaveh Razavi

VU VRIJE UNIVERSITEIT AMSTERDAM

VUSec

GLitch: what?

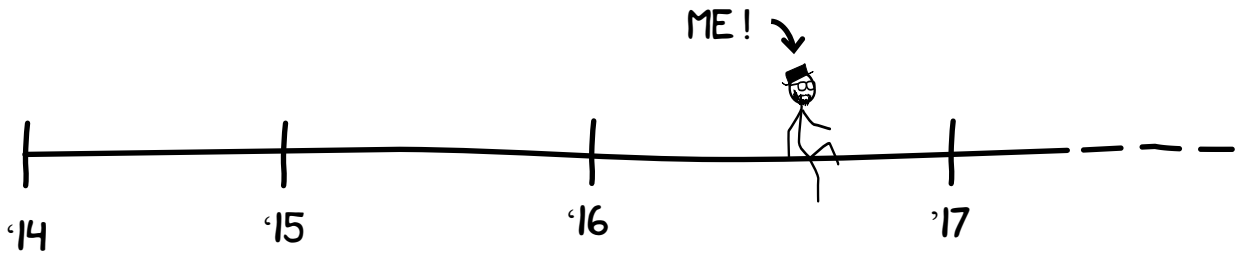
First **Rowhammer** exploit from **JavaScript** on **mobile**

Fastest JS-based Rowhammer exploit

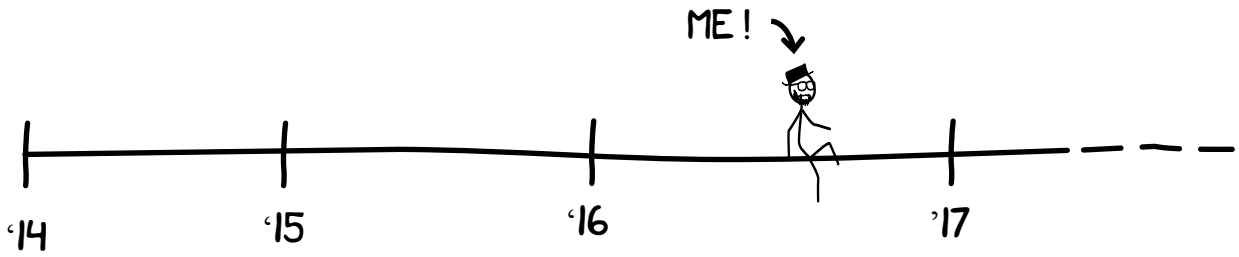
First **GPU-accelerated** bit flip

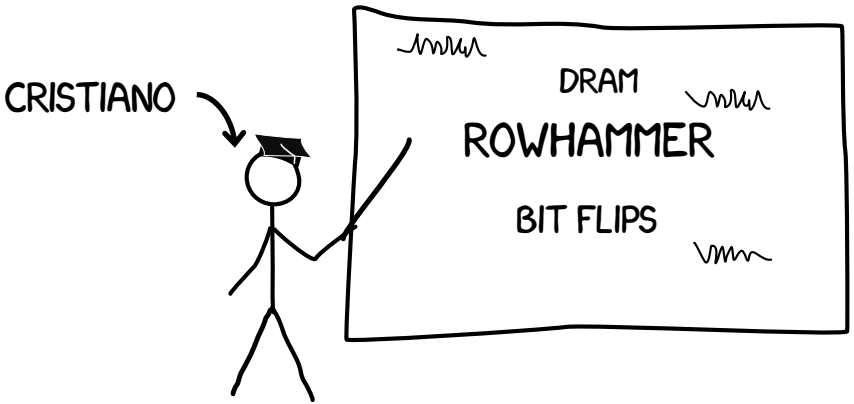
The chronicles of GLitch

GLitch: the chronicles

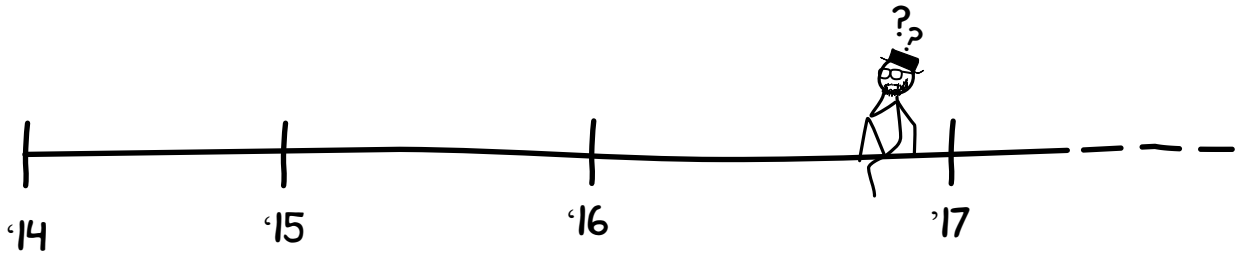


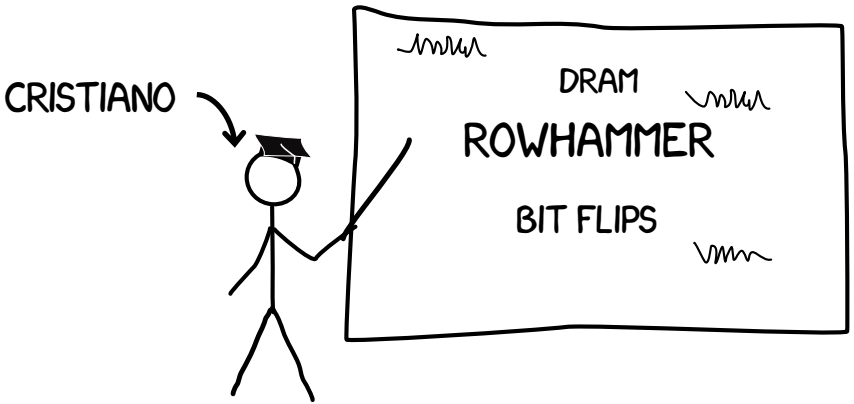
GLitch: the chronicles



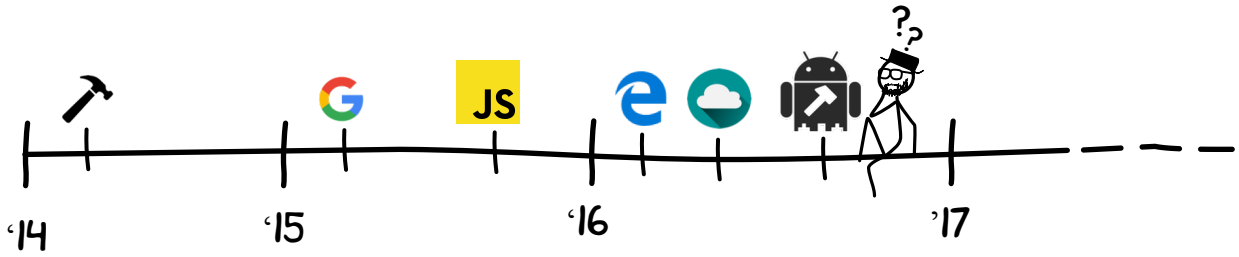


GLitch: the chronicles





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

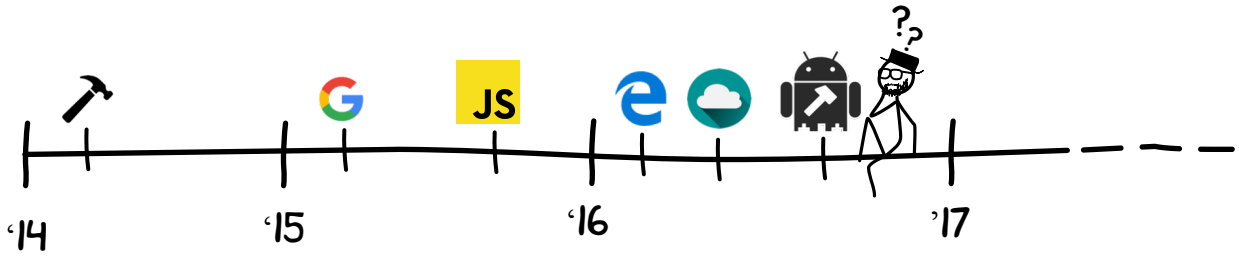
[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

@vvdveen
+100XP

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

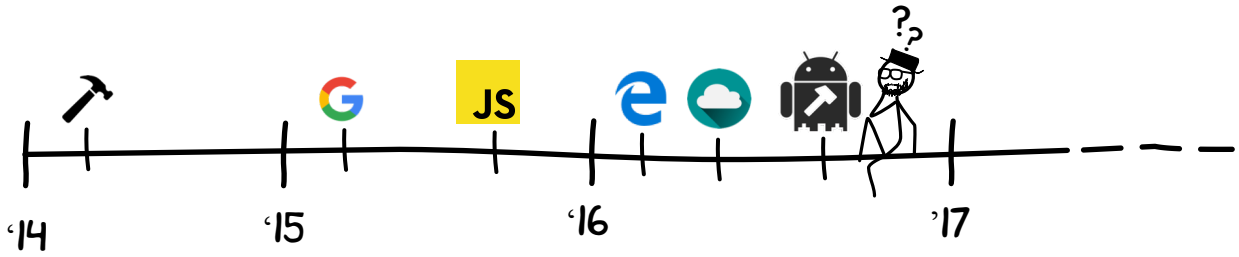
[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

@vvdveen +100XP

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

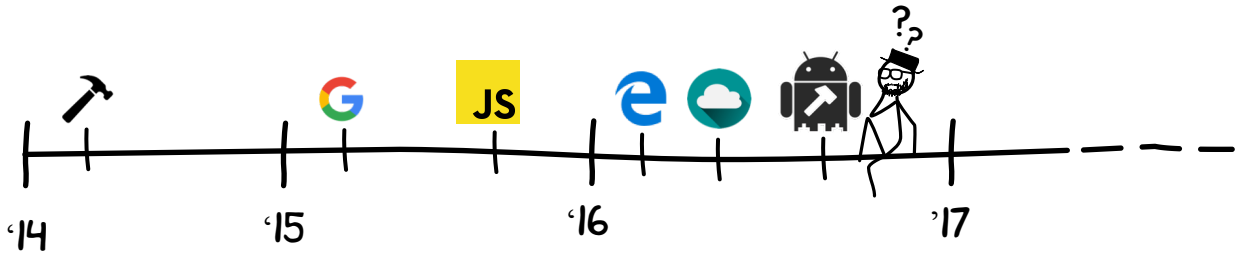
[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

@vvdveen +100XP

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

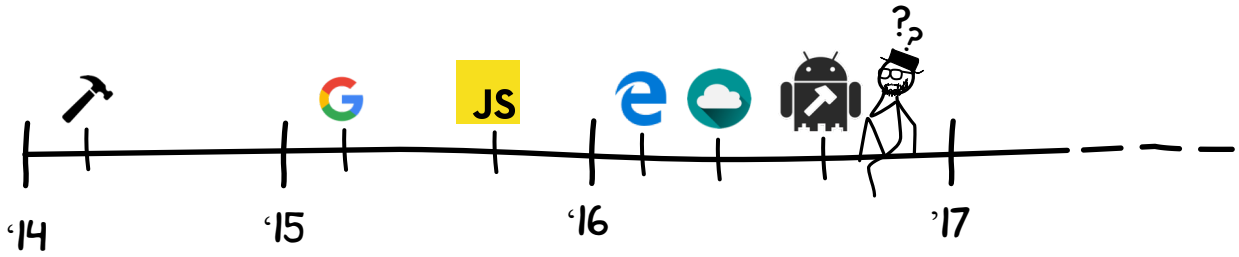
[4] Dedup est Machina: Breaking Microsoft Edge's sandbox

[5] Flip Feng Shui: Breaking the cloud

[6] Drammer: Flip feng shui goes mobile

@vvdveen +100XP

GLitch: the chronicles



[1] Flipping bits in memory without accessing them

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

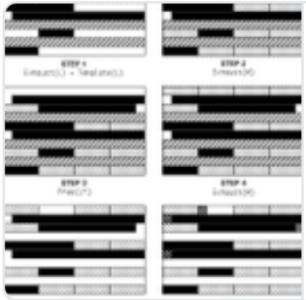
[3] Rowhammer.js: 1st RH bit flip in JavaScript

[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

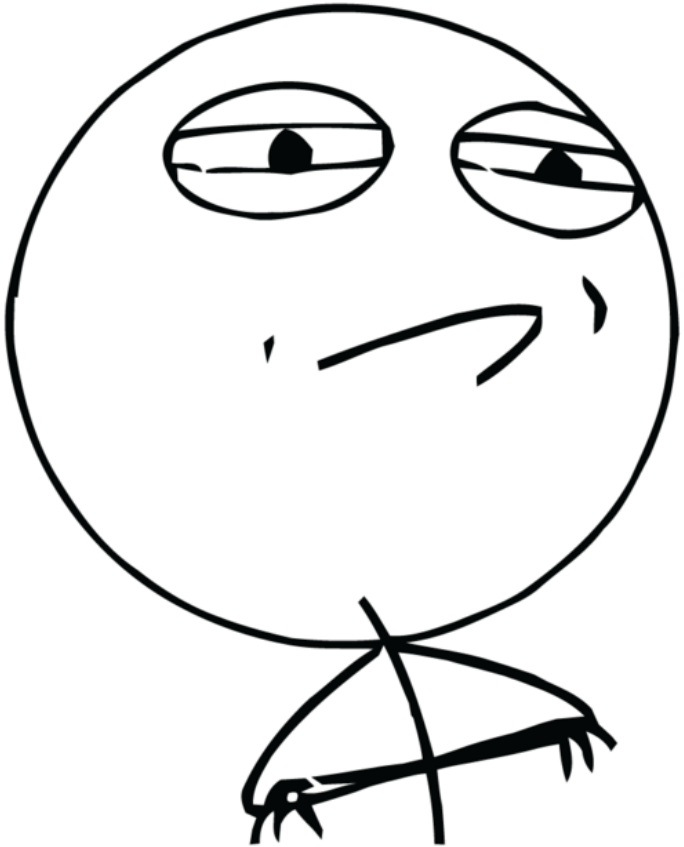
@thegrugq

Following

In risposta a @vvdveen e @vu5ec

love to see it happen. :)

CHALLENGE ACCEPTED



Attacker primitives

Attacker primitives

#P1. Fast cache eviction

- clflush (native)
- eviction sets (JS)

#P2. Contiguous memory

- THPs (native and JS)

Attacker primitives

#P1. Fast cache eviction

- clflush (native)
- eviction sets (JS)

#P2. Contiguous memory

- THPs (native and JS)

Attacker primitives

#P1. Fast cache eviction

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

#P2. Contiguous memory

- ✗ THPs (native and JS)

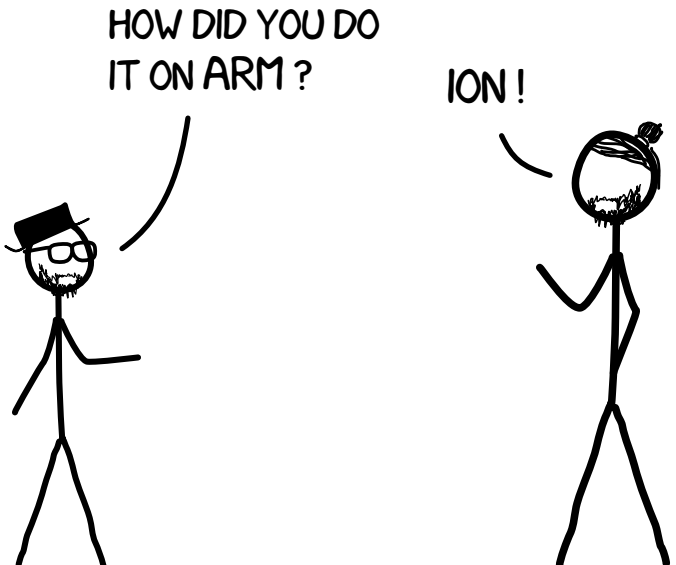
Attacker primitives

#P1. Fast cache eviction

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

#P2. Contiguous memory

- ✗ THPs (native and JS)



Attacker primitives

#P1. Fast cache eviction

- ✗ [illegible]
- ✗ [illegible]

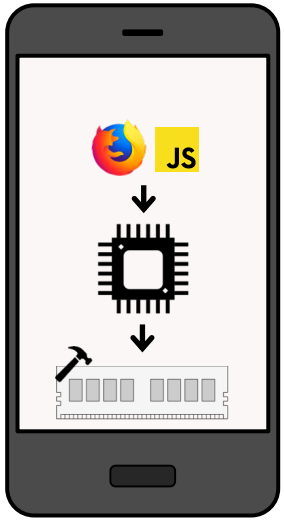
#P2. Conti[illegible]

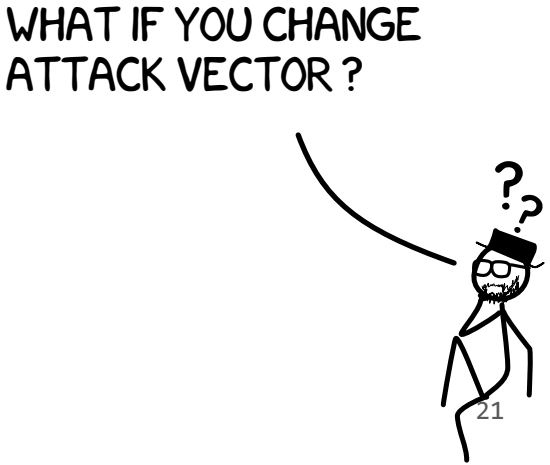
- ✗ THPs (native and JS)

ION == DMA memory
Uncached & Contiguous

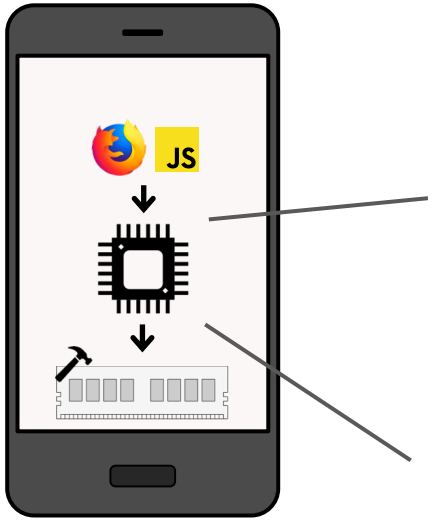

We are screwed.

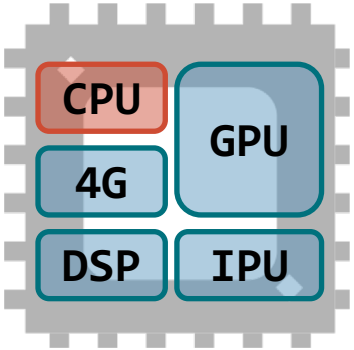
Attack Vector

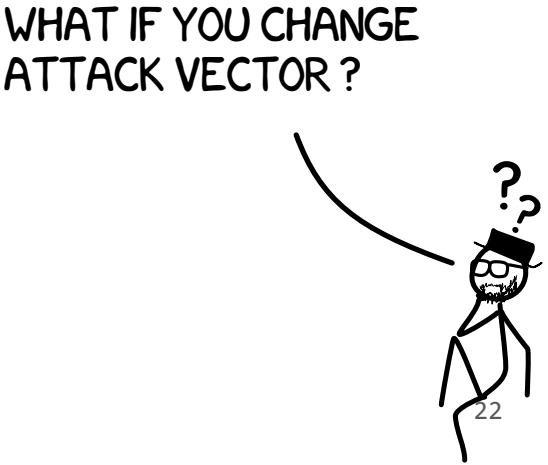




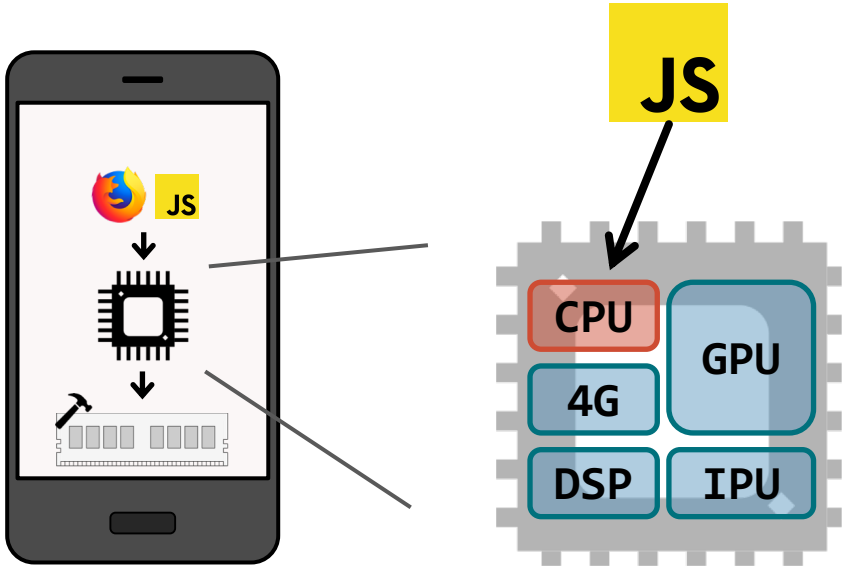
Attack Vector

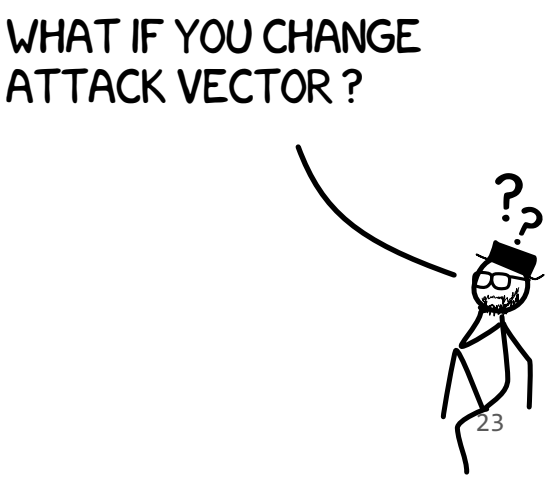




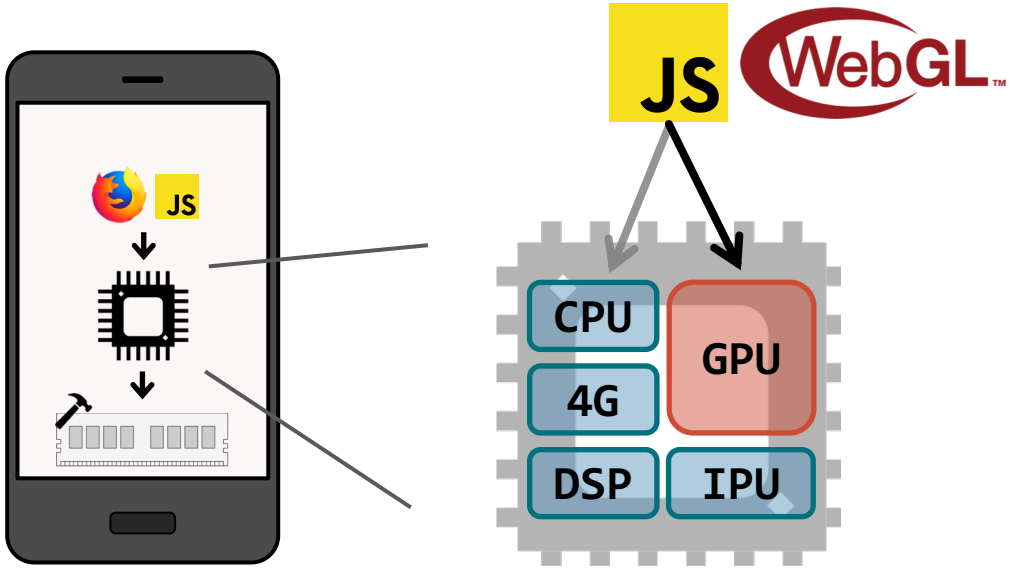


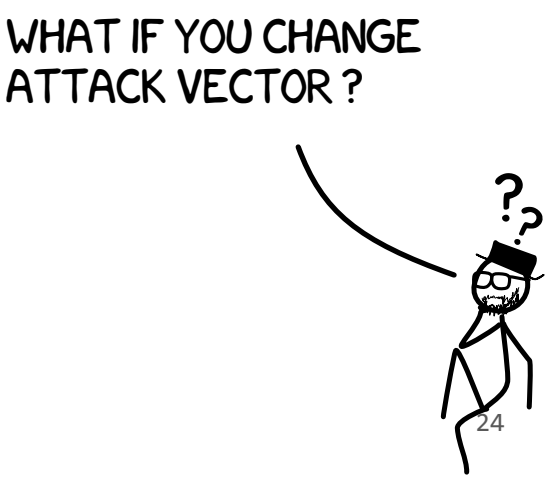
Attack Vector





Attack Vector





Attacker primitives

#P1. DRAM Access

#P2. Fast cache eviction

#P3. Contiguous memory

Attacker primitives

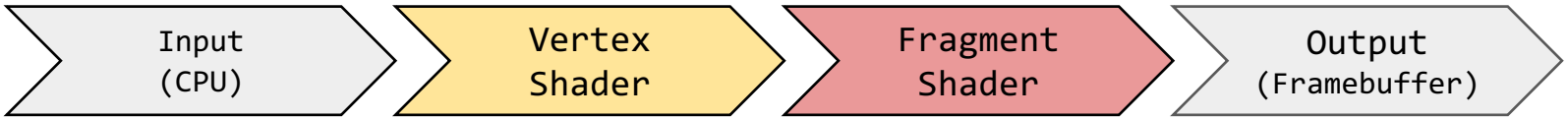
#P1. DRAM Access

#P2. Fast cache eviction

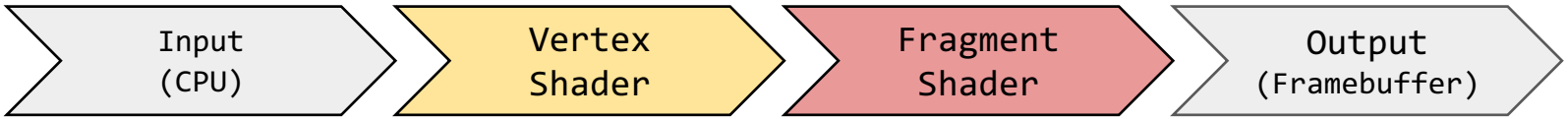
#P3. Contiguous memory

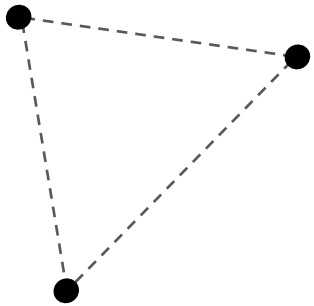
Understanding the GPU

#P1. GPU: The rendering pipeline

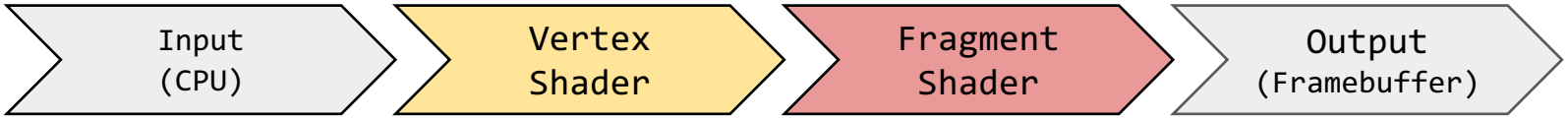


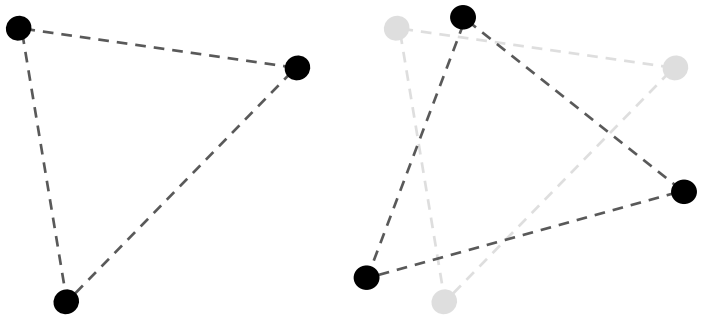
#P1. GPU: The rendering pipeline



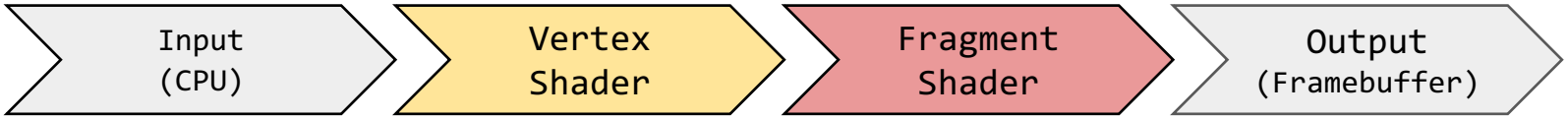


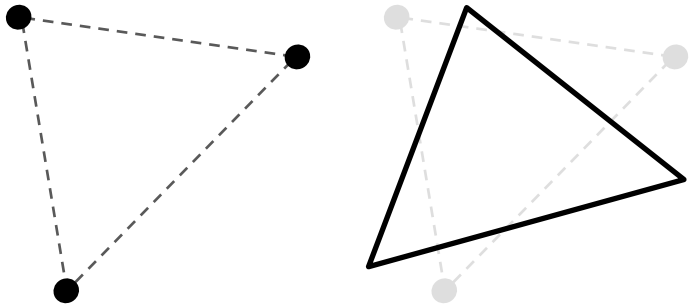
#P1. GPU: The rendering pipeline



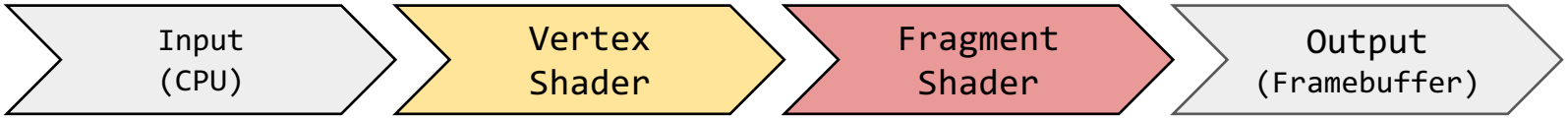


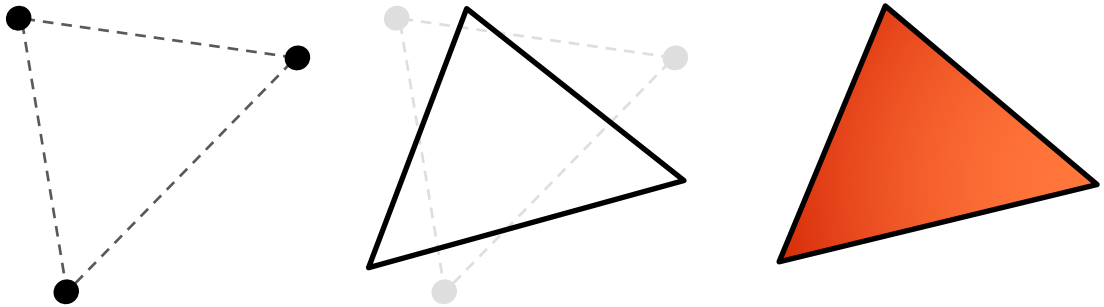
#P1. GPU: The rendering pipeline





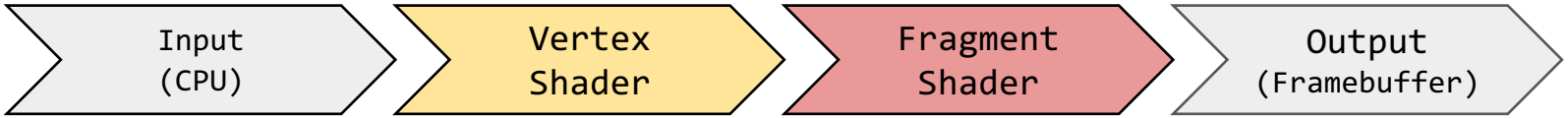
#P1. GPU: The rendering pipeline

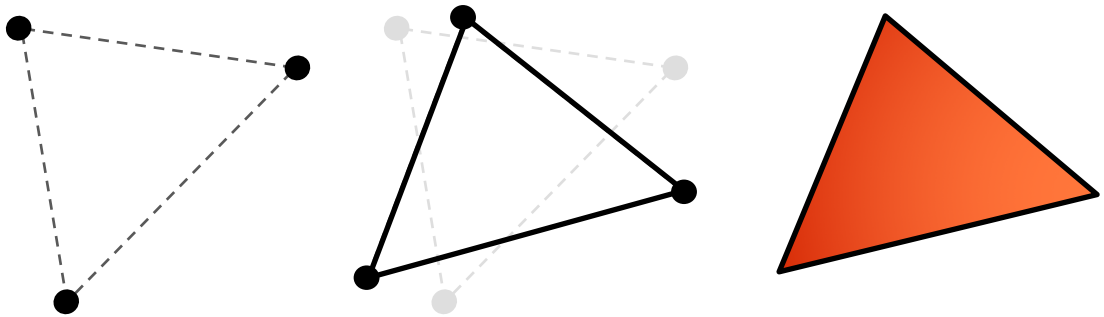




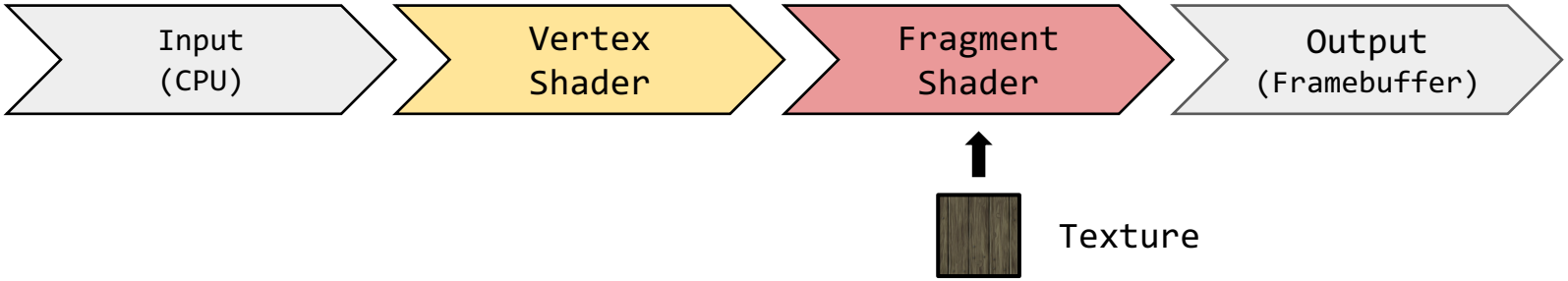

MINECRAFT
EDUCATION EDITION

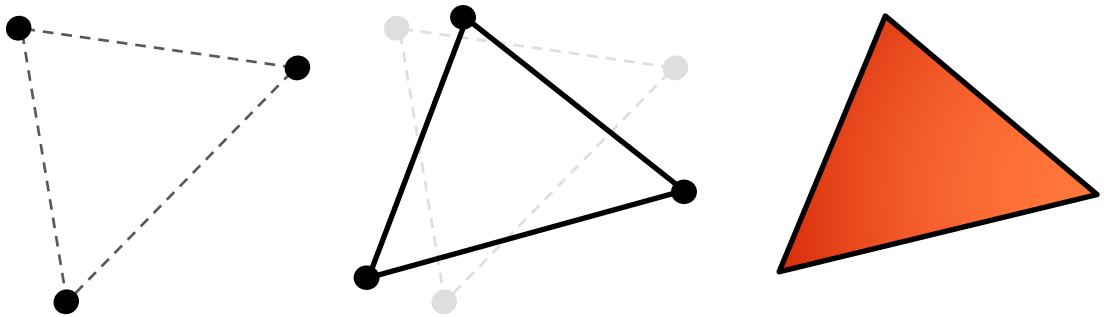
#P1. GPU: The rendering pipeline



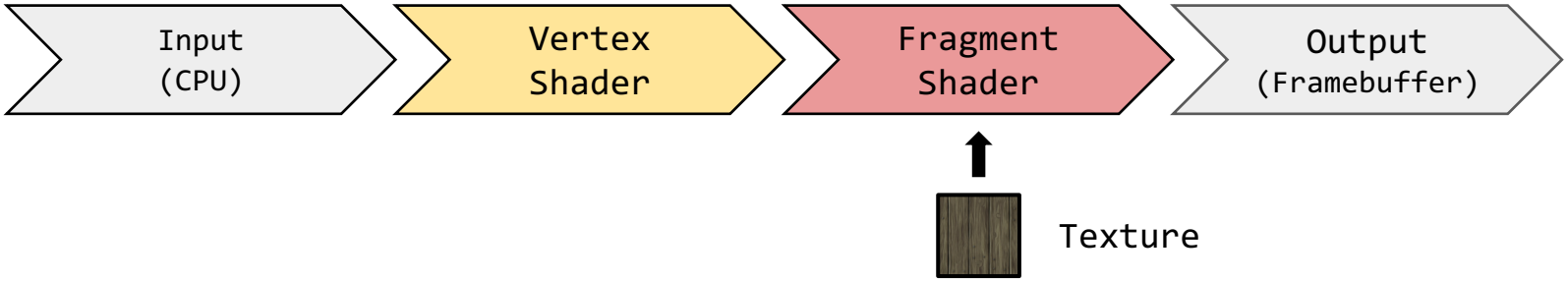


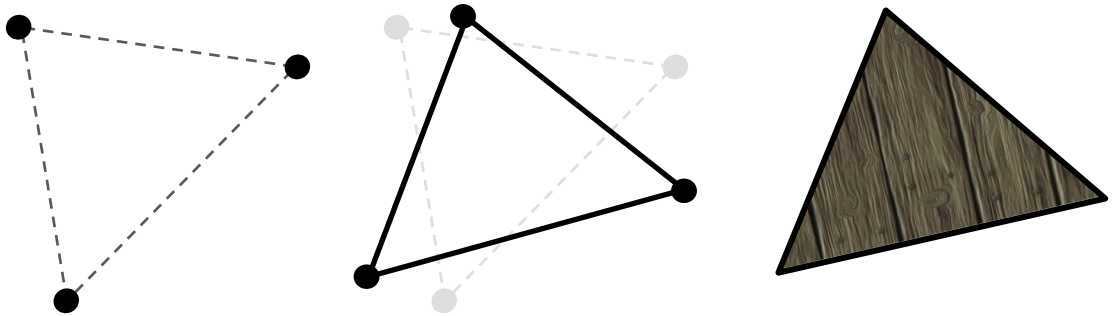
#P1. GPU: The rendering pipeline



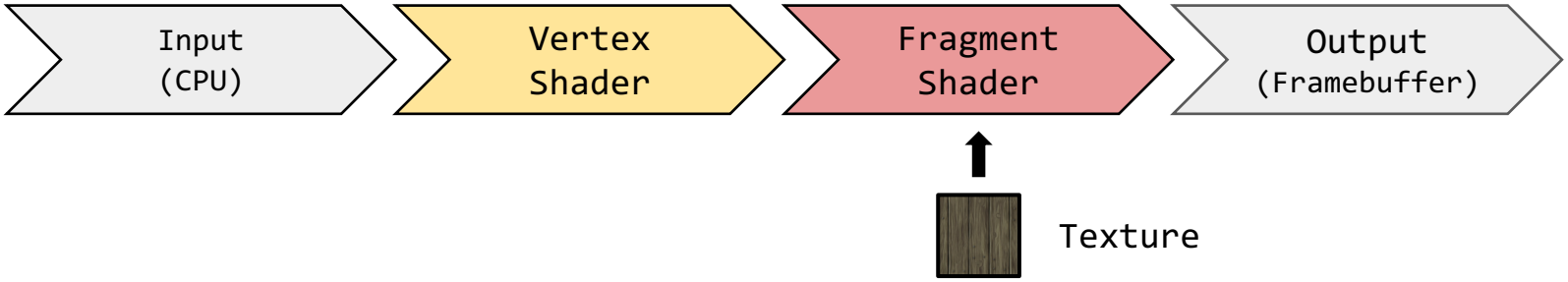


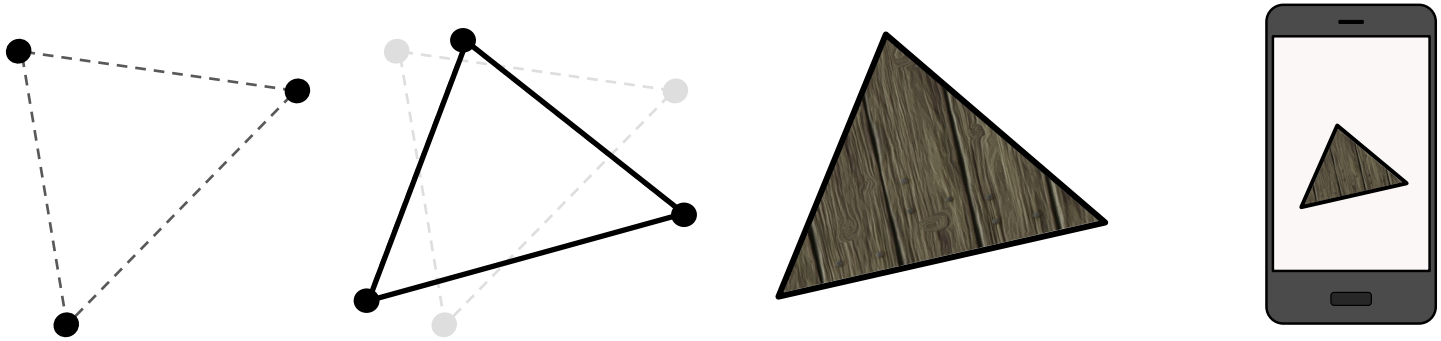
#P1. GPU: The rendering pipeline



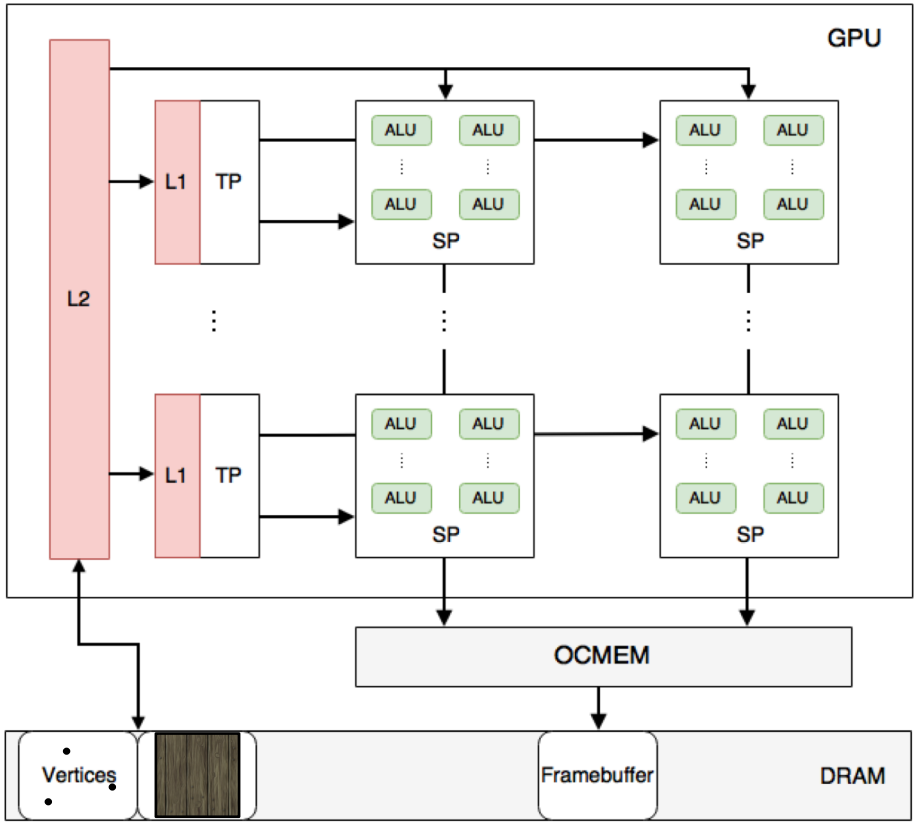


#P1. GPU: The rendering pipeline

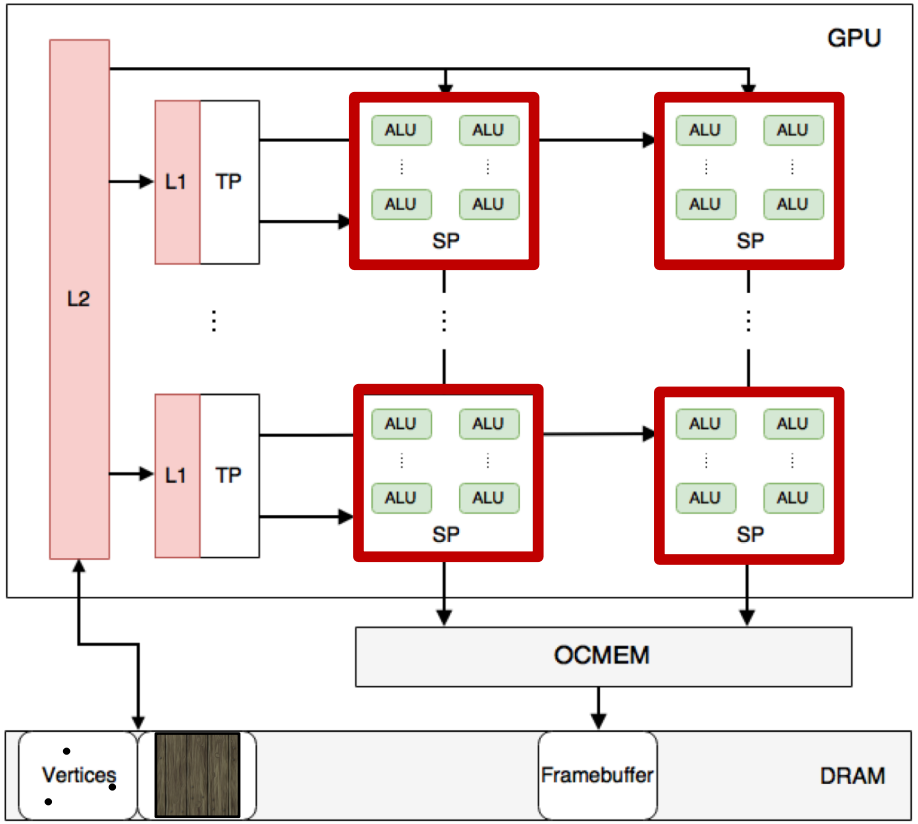




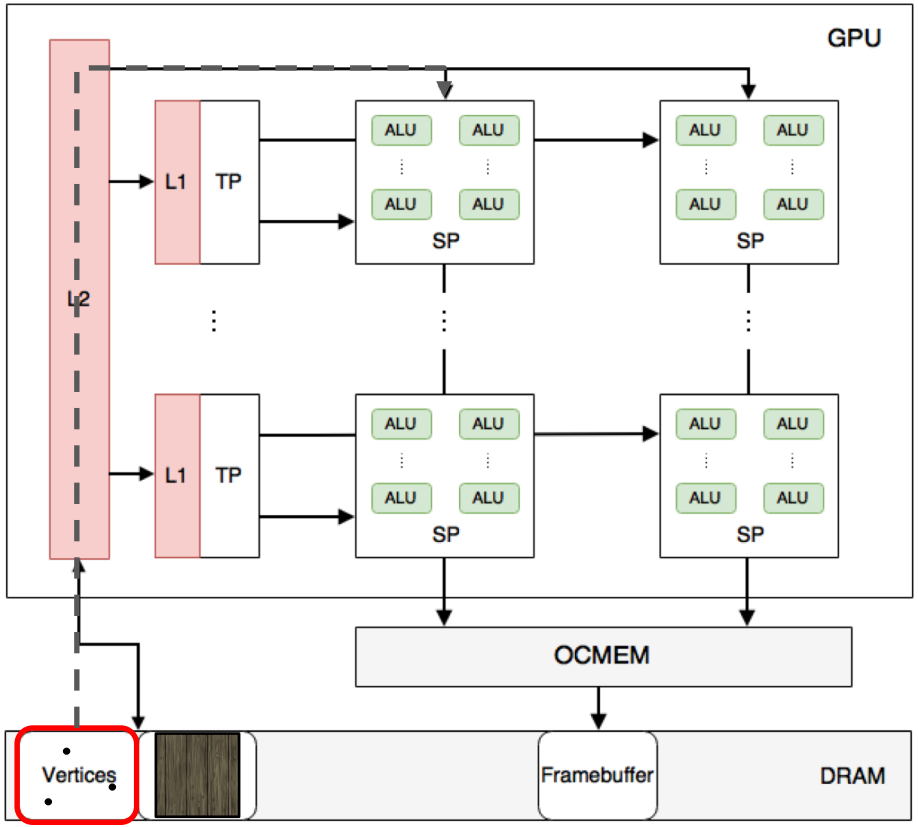
#P1. GPU: The architecture



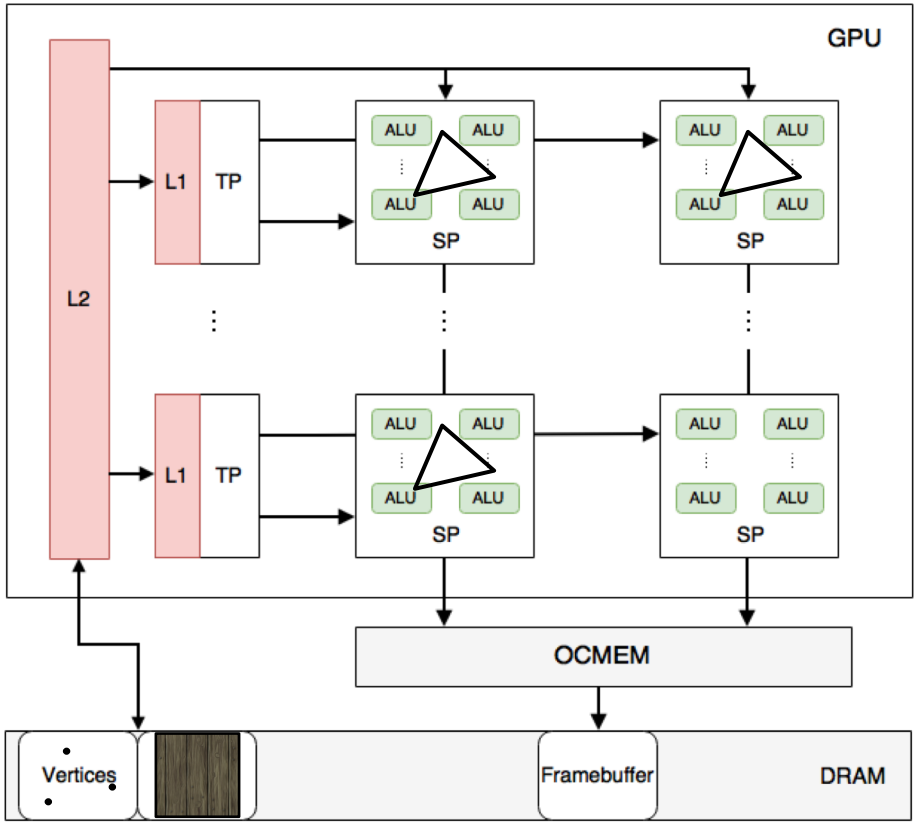
#P1. GPU: The architecture



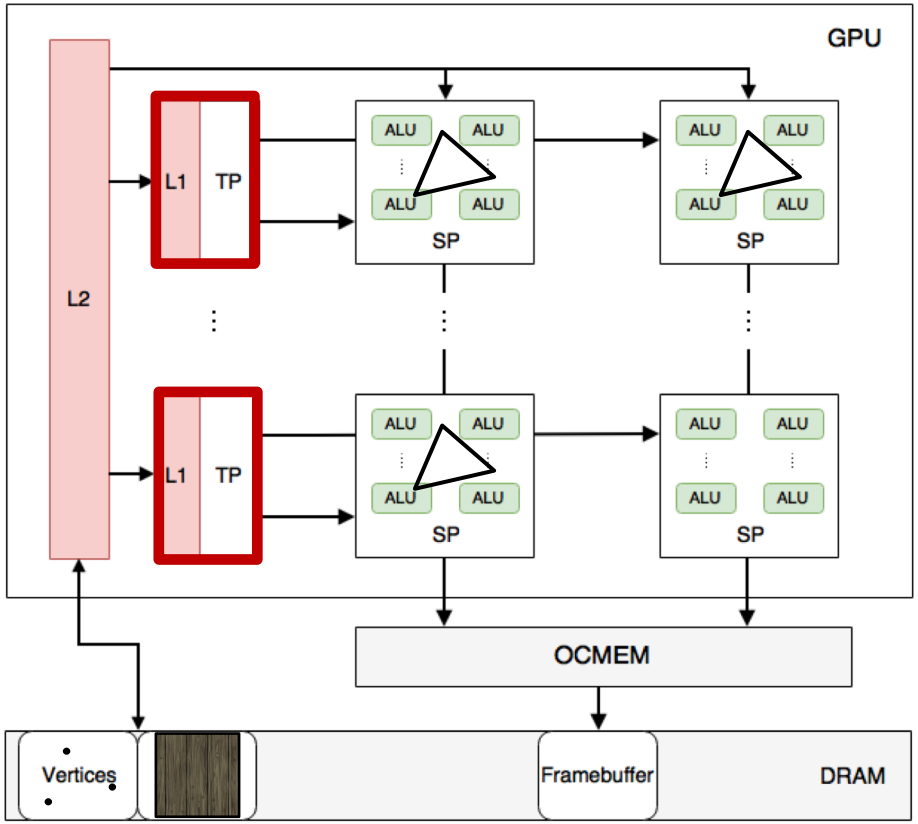
#P1. GPU: The architecture



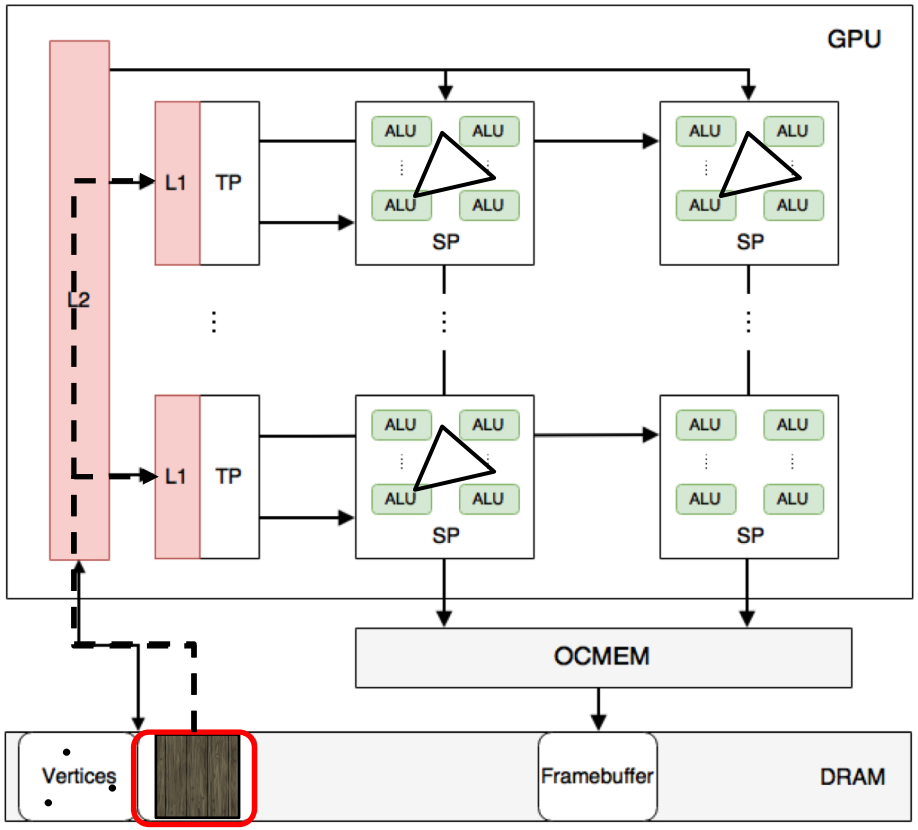
#P1. GPU: The architecture



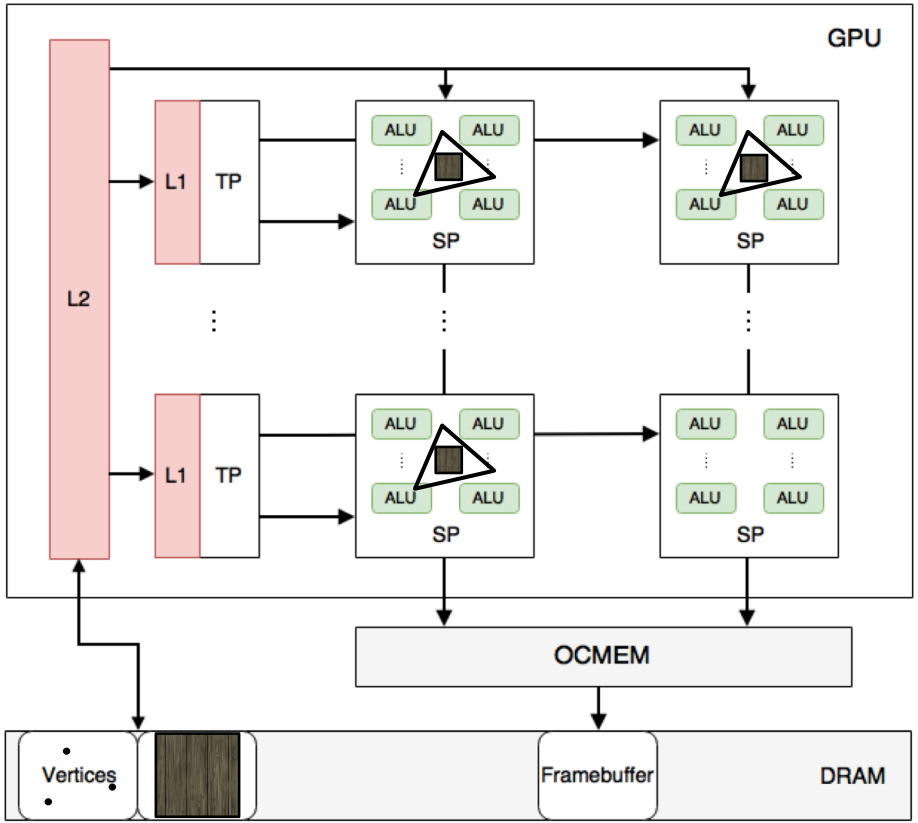
#P1. GPU: The architecture



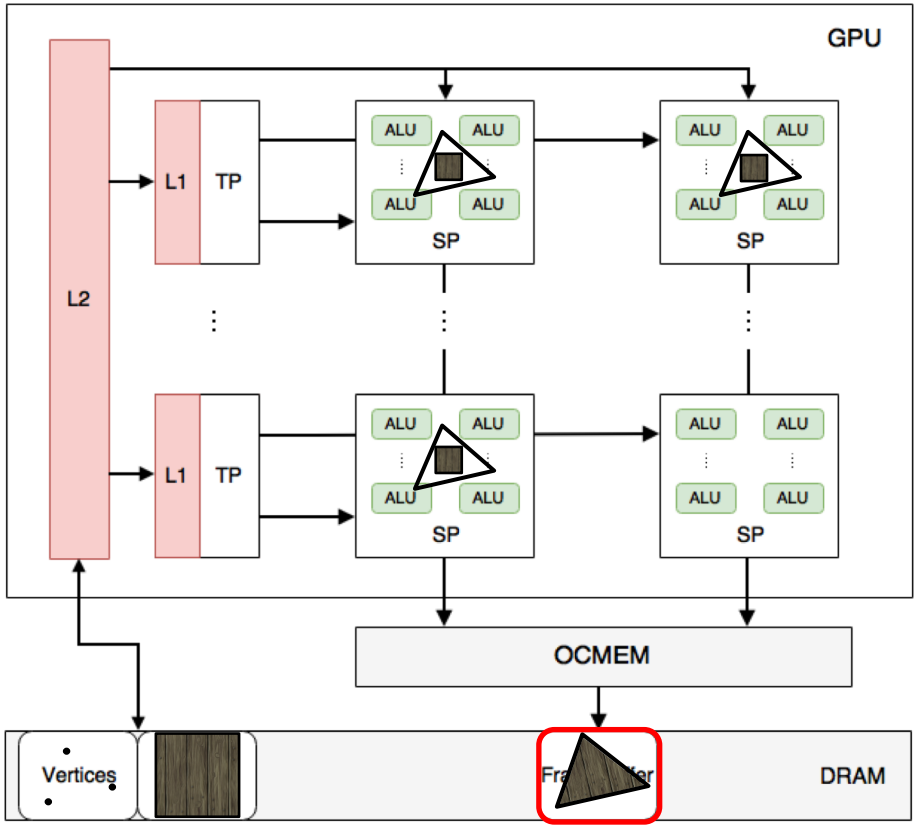
#P1. GPU: The architecture



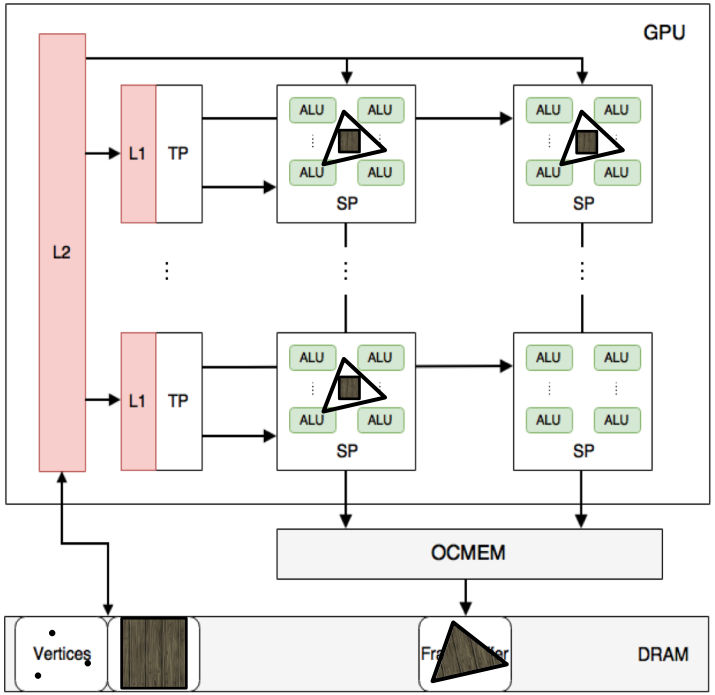
#P1. GPU: The architecture



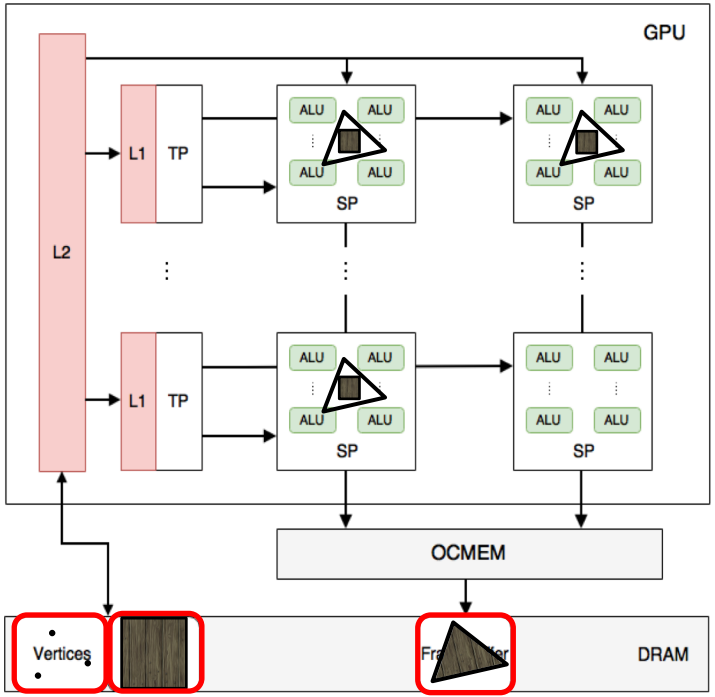
#P1. GPU: The architecture



#P1. DRAM access

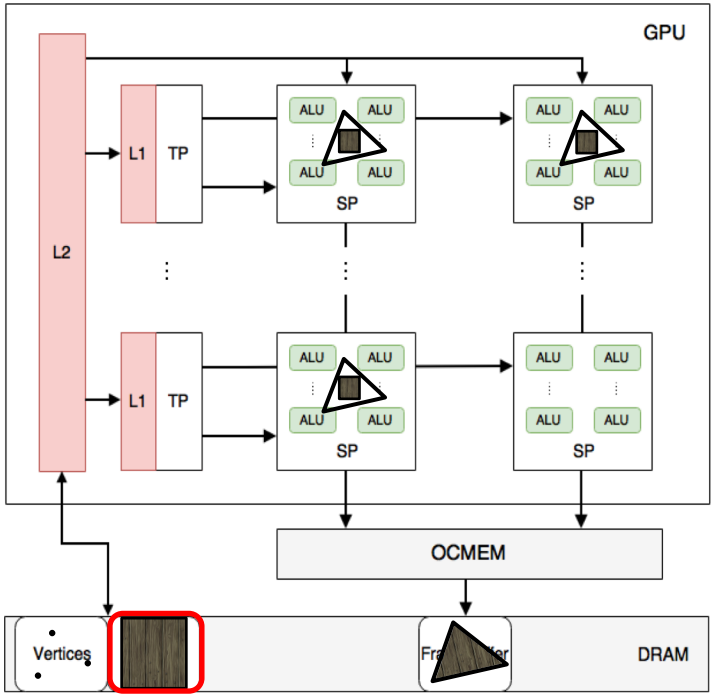


#P1. DRAM access



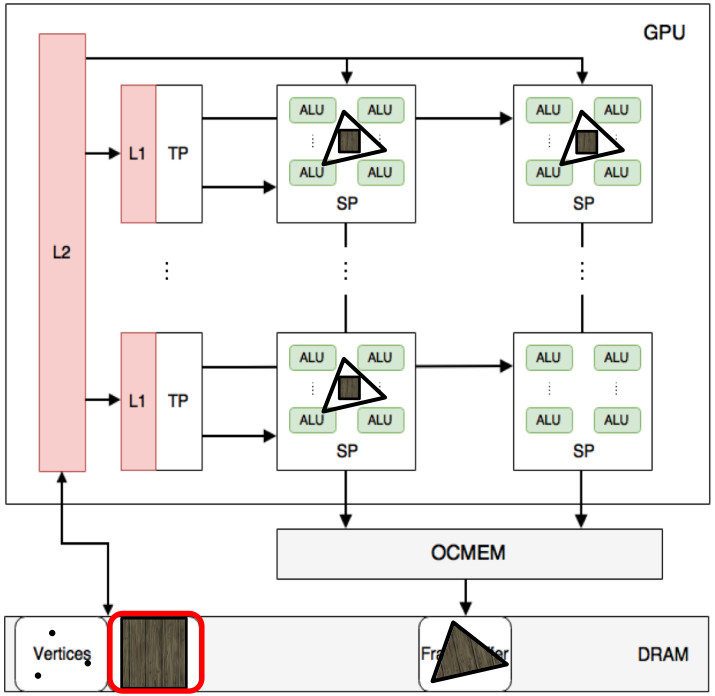
1. Read Vertices
2. Read Textures
3. Write to Framebuffer

#P1. DRAM access



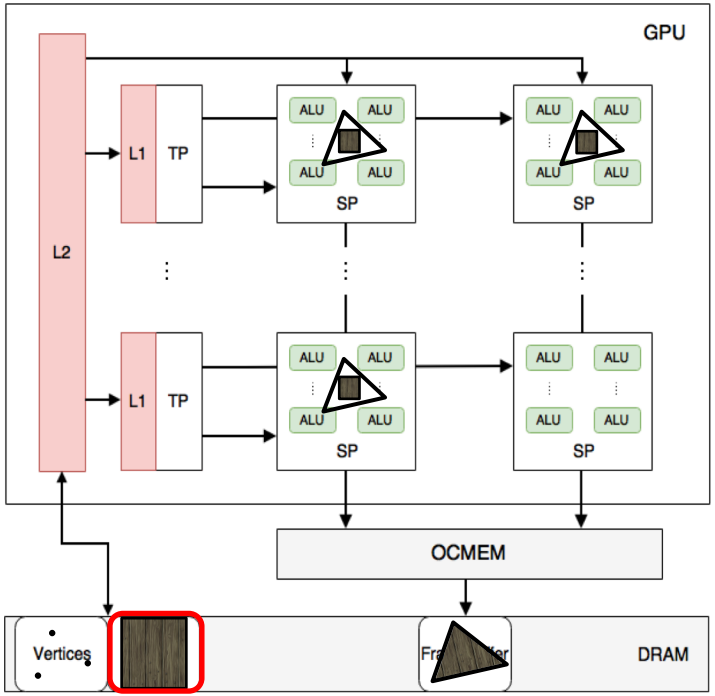
1. Read Vertices
2. Read Textures ==> most predictable
3. Write to Framebuffer

#P1. DRAM access: texture sampling




```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   gl_FragColor = texture2D(tex, coord);
6 }
```

#P1. DRAM access: texture sampling




```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   gl_FragColor = texture2D(tex, coord);
6 }
```

tex[coord]

Attacker primitives

#P1. DRAM access ✓

#P2. Fast cache eviction

#P3. Contiguous memory

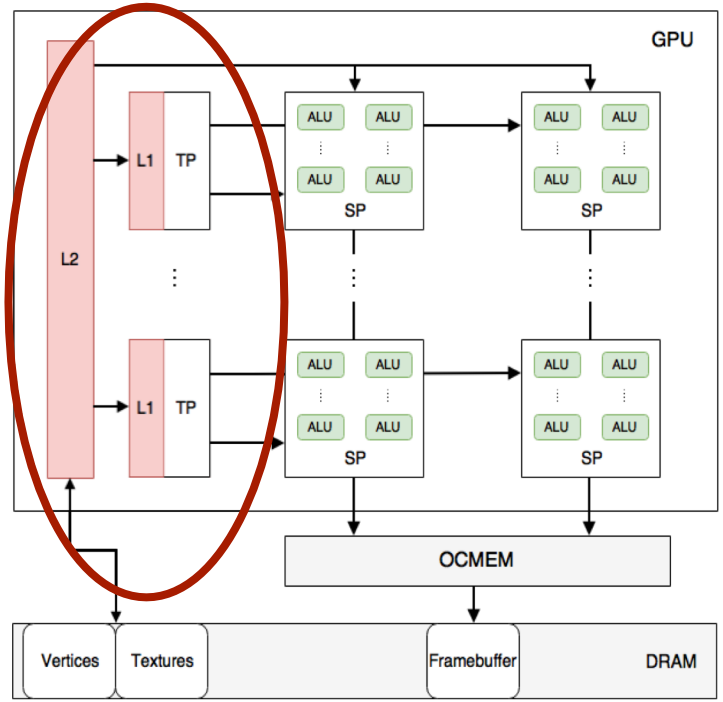
Attacker primitives

#P1. DRAM access ✓

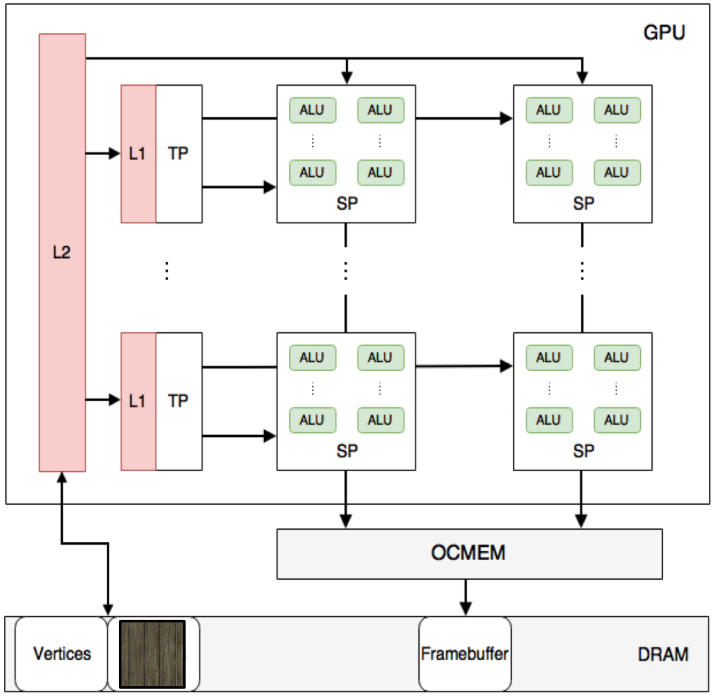
#P2. Fast cache eviction

#P3. Contiguous memory

#P2. Fast cache eviction

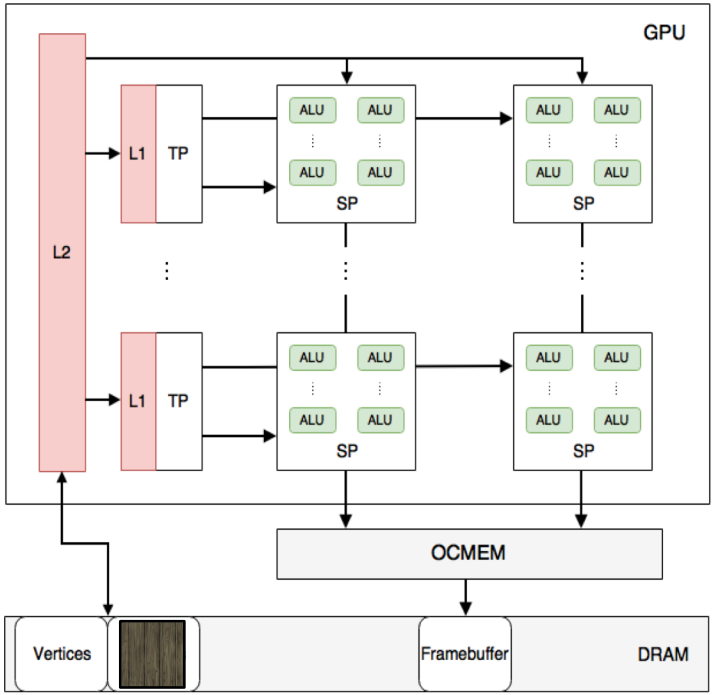


#P2. Fast cache eviction



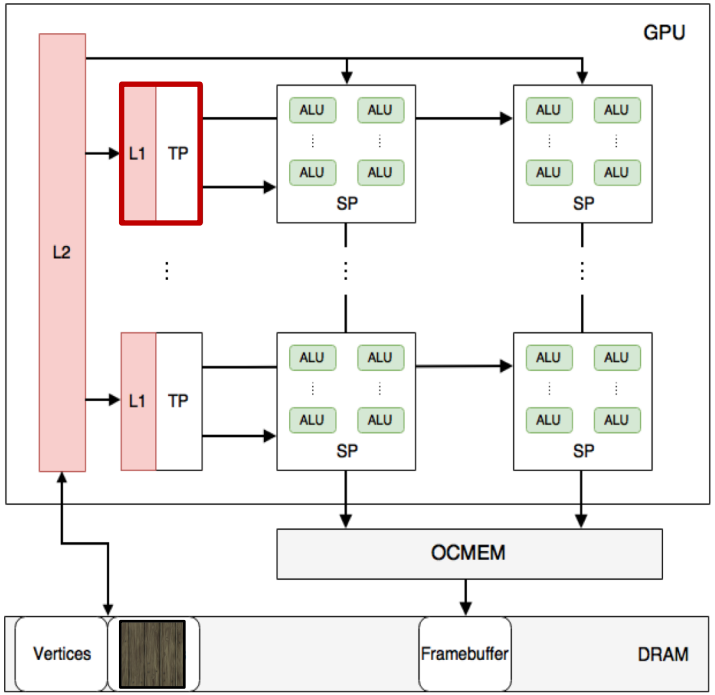

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Fast cache eviction



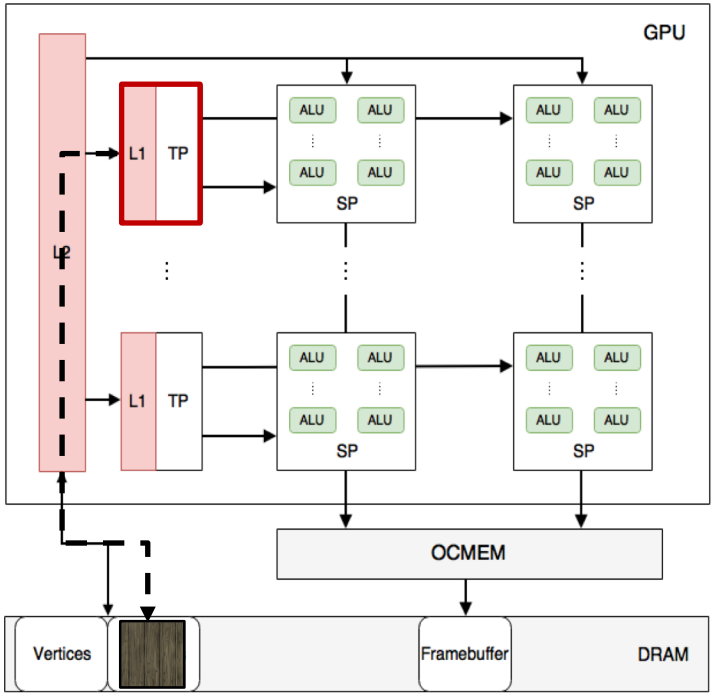

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Fast cache eviction



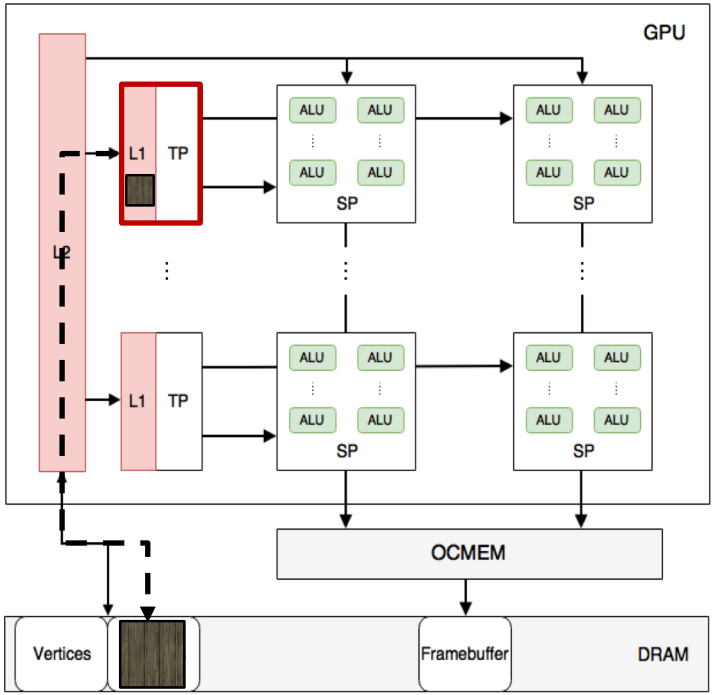

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Fast cache eviction



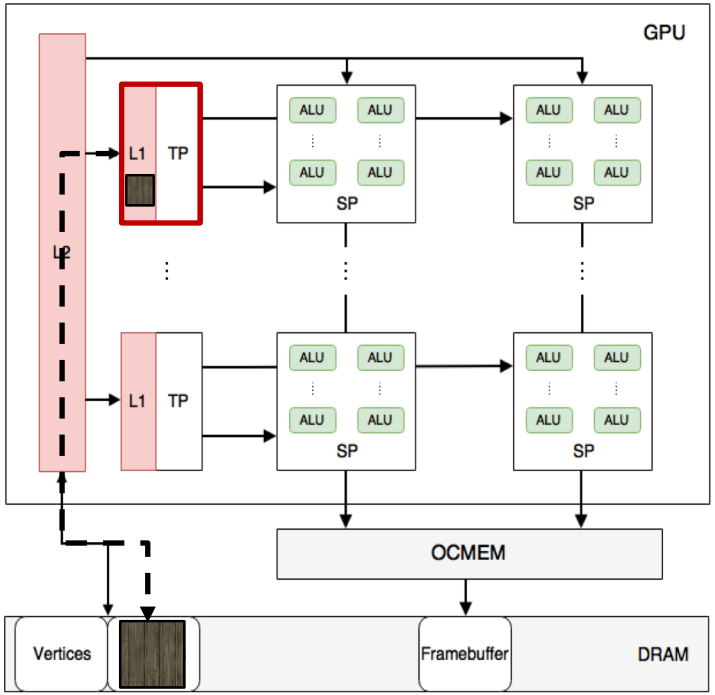

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Fast cache eviction



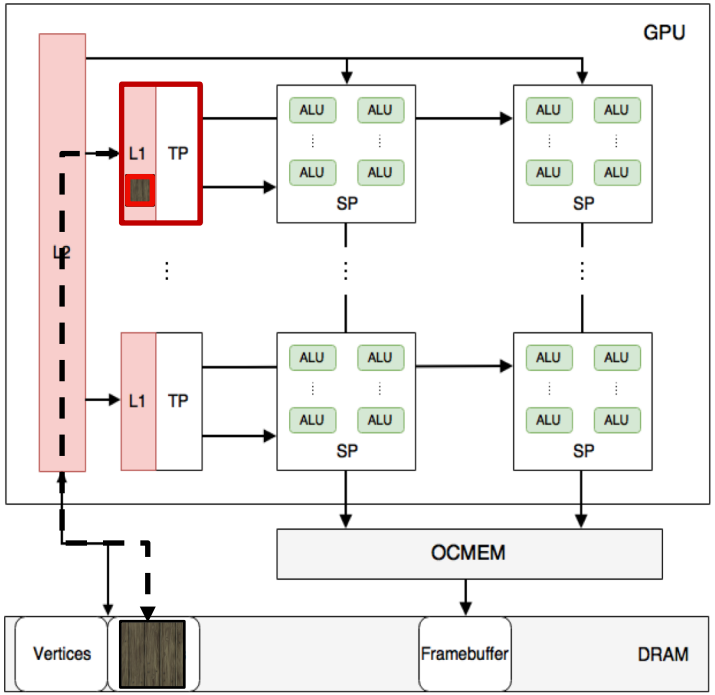

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Fast cache eviction



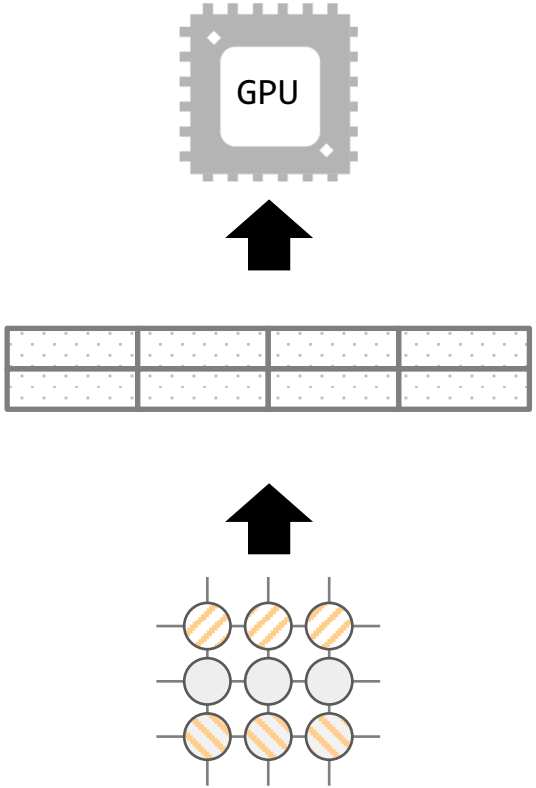

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Fast cache eviction



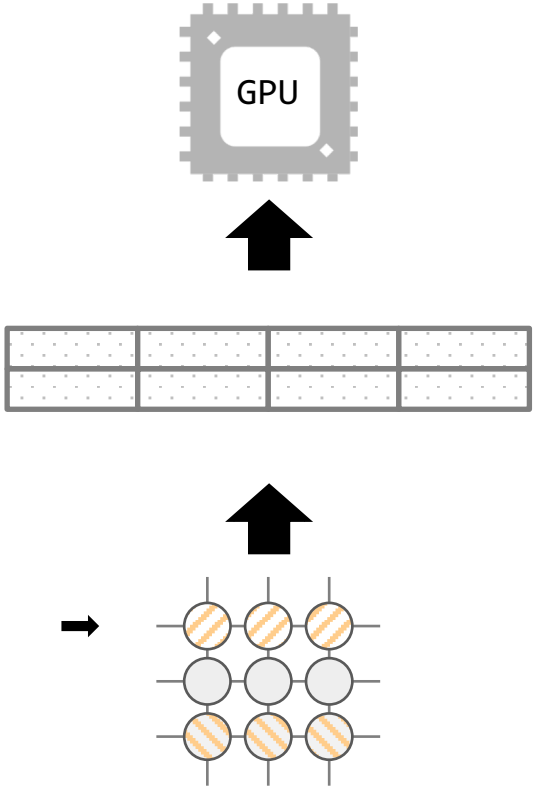

```
1 uniform sampler2D tex;
2
3 void main() {
4   vec2 coord = vec2(0,0);
5   vec4 a = texture2D(tex, coord);
6   vec4 b = texture2D(tex, coord);
7 }
```

#P2. Eviction-based Rowhammer: GPU



Caches: Small & Deterministic

#P2. Eviction-based Rowhammer: GPU



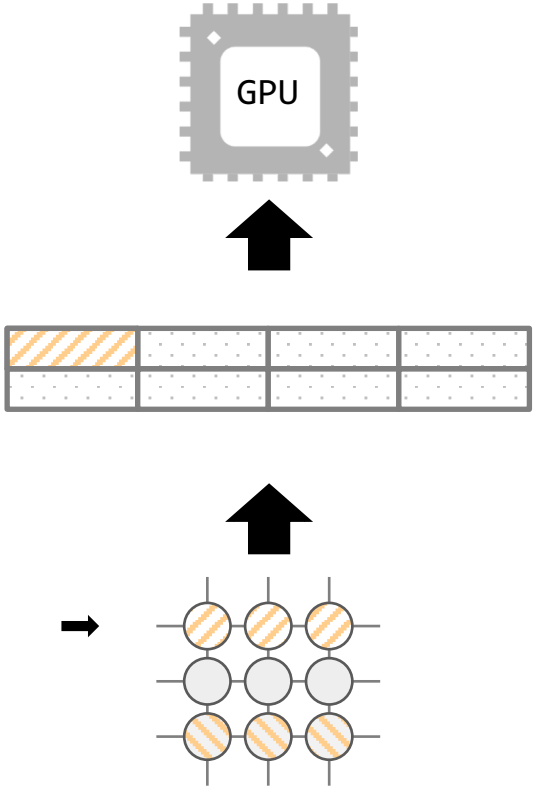
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



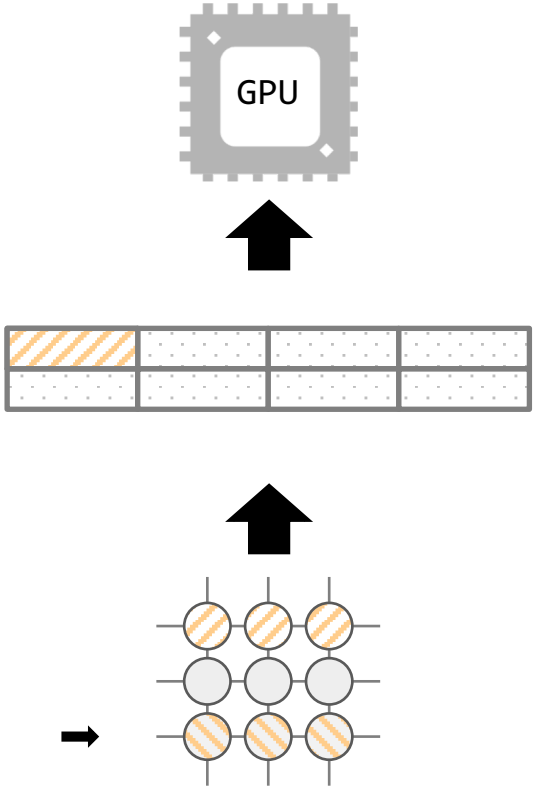
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



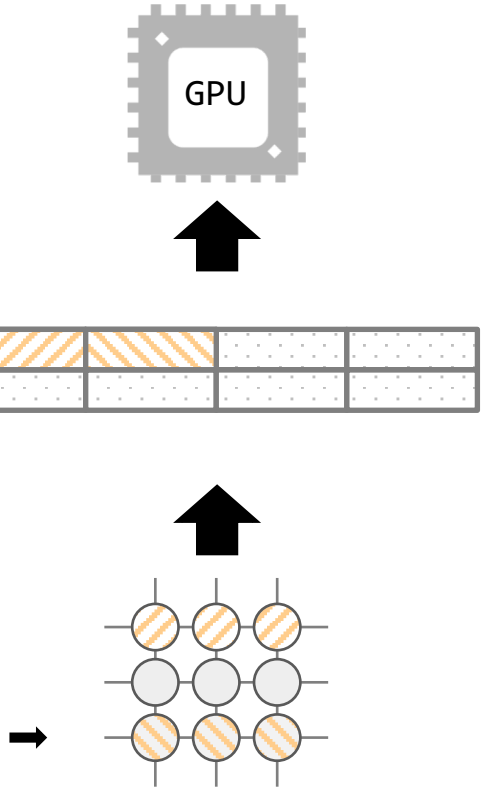
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



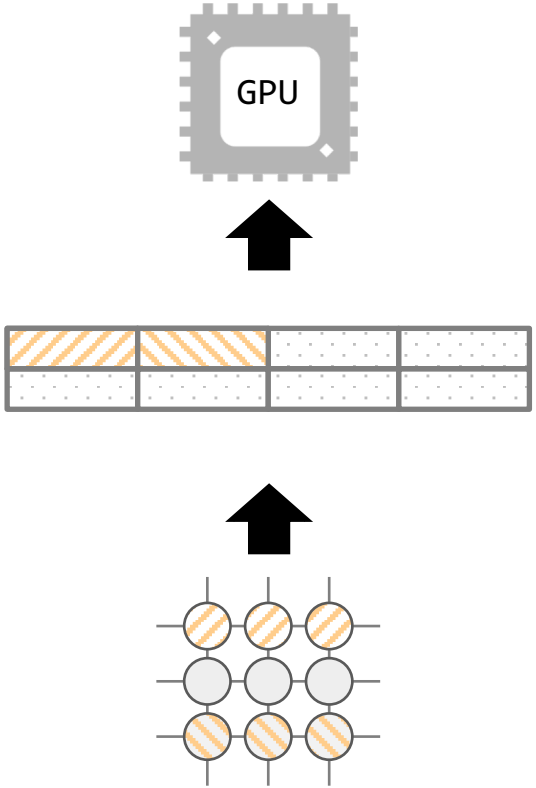
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



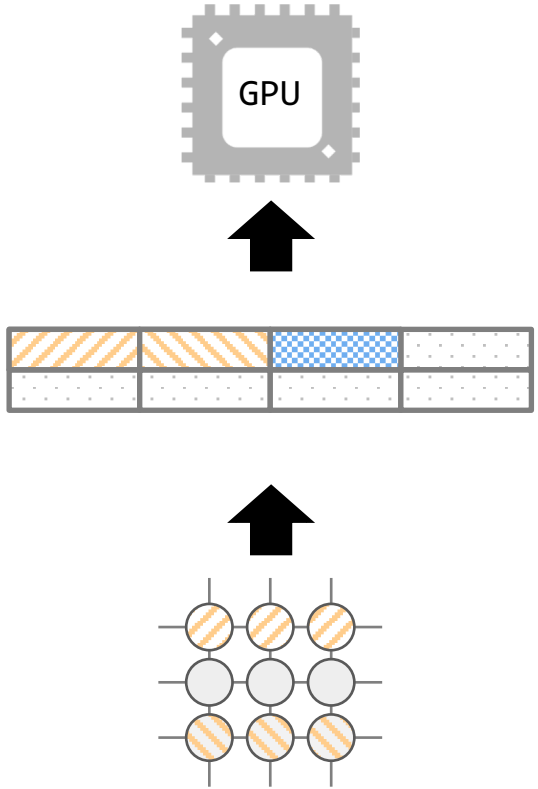
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



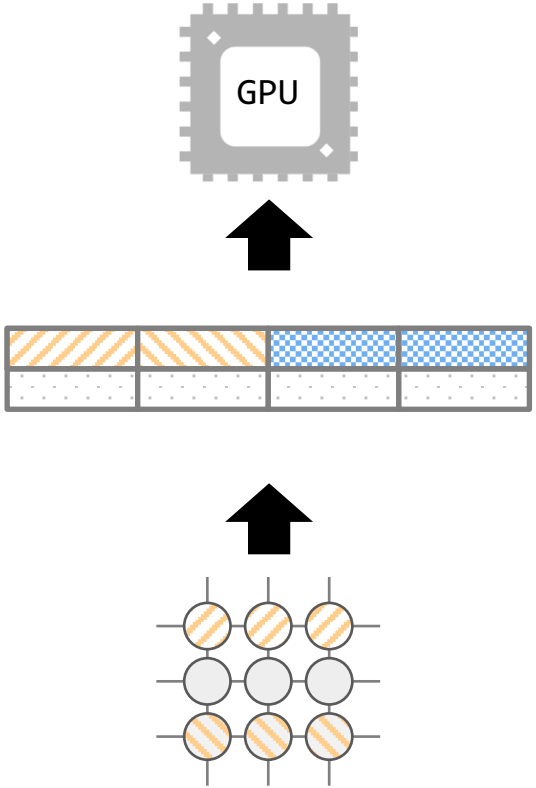
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



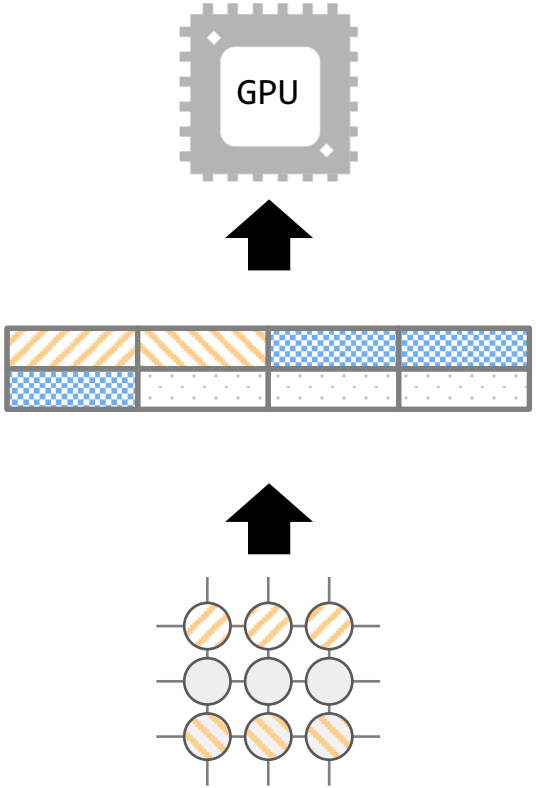
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



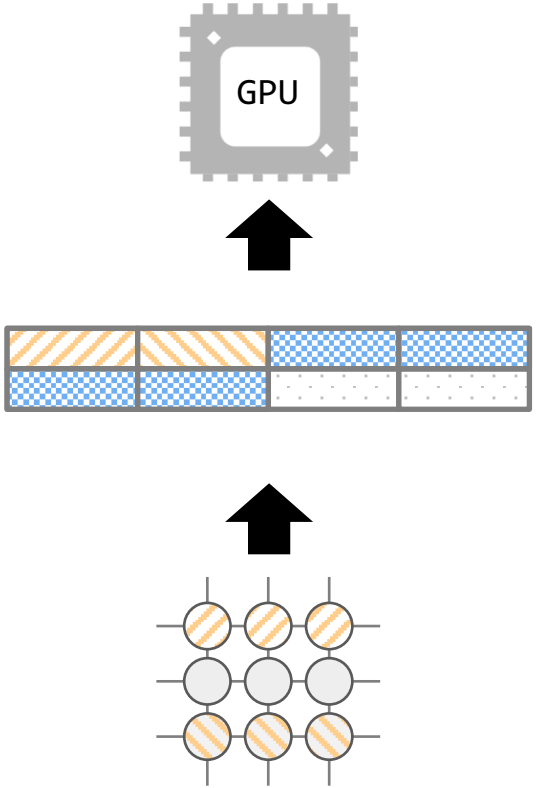
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



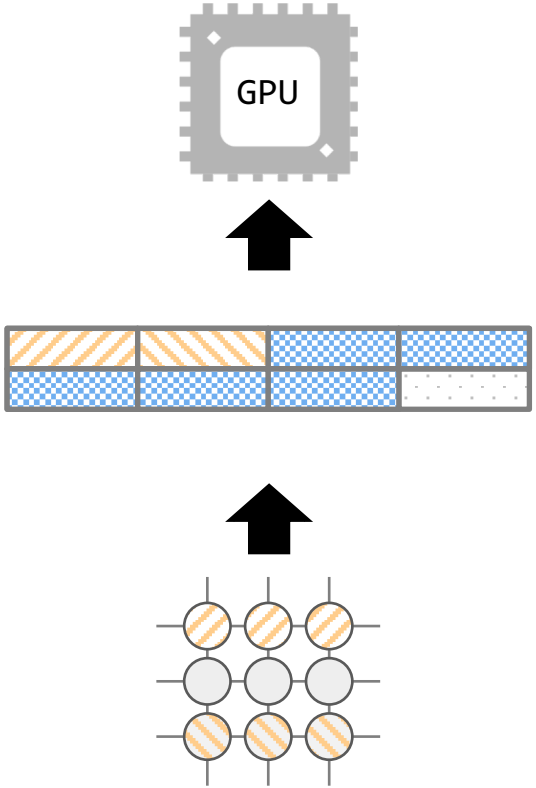
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



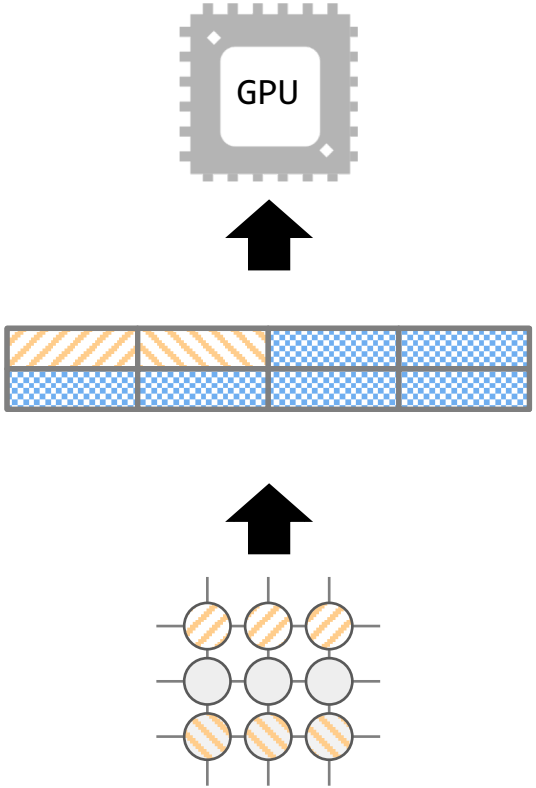
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



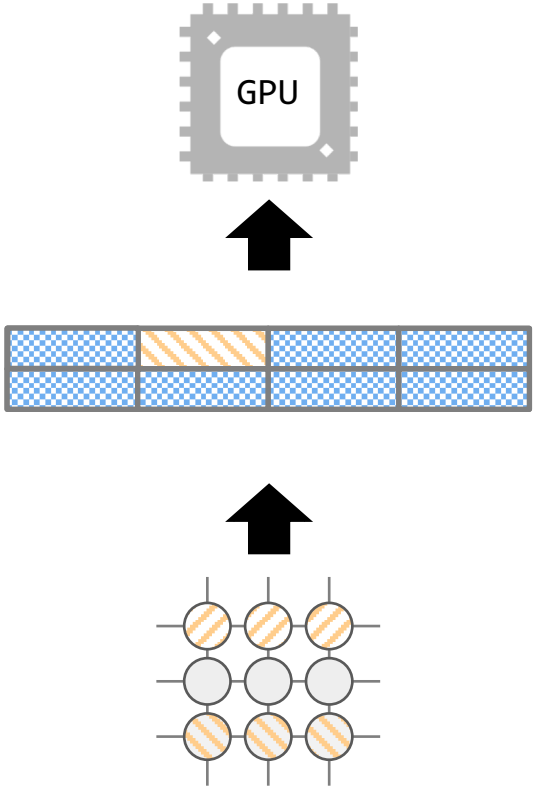
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



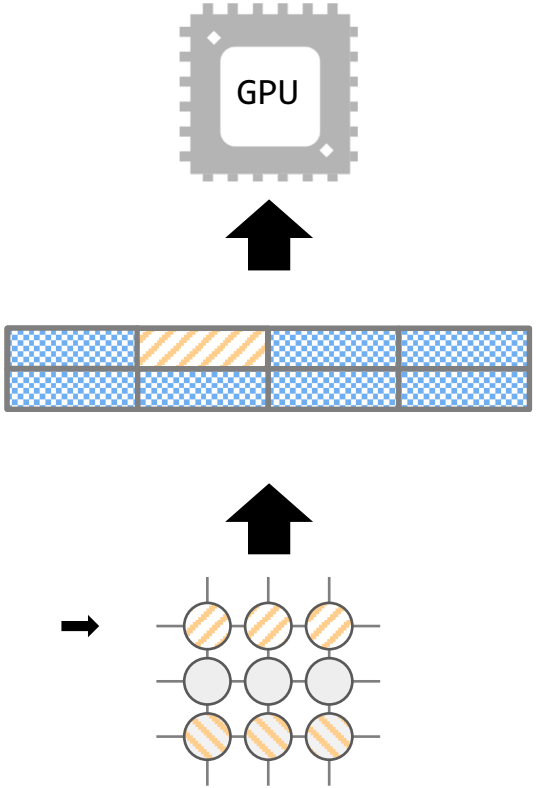
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



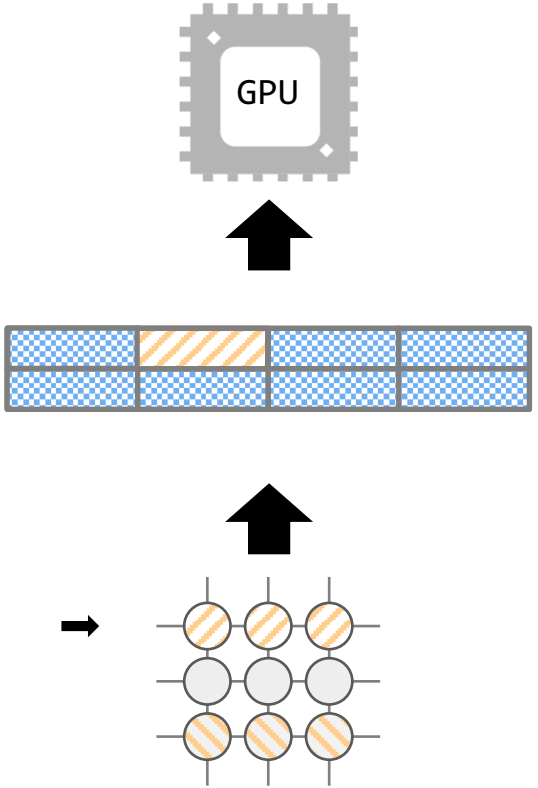
Caches: Small & Deterministic

Steps:

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

   ...

#P2. Eviction-based Rowhammer: GPU



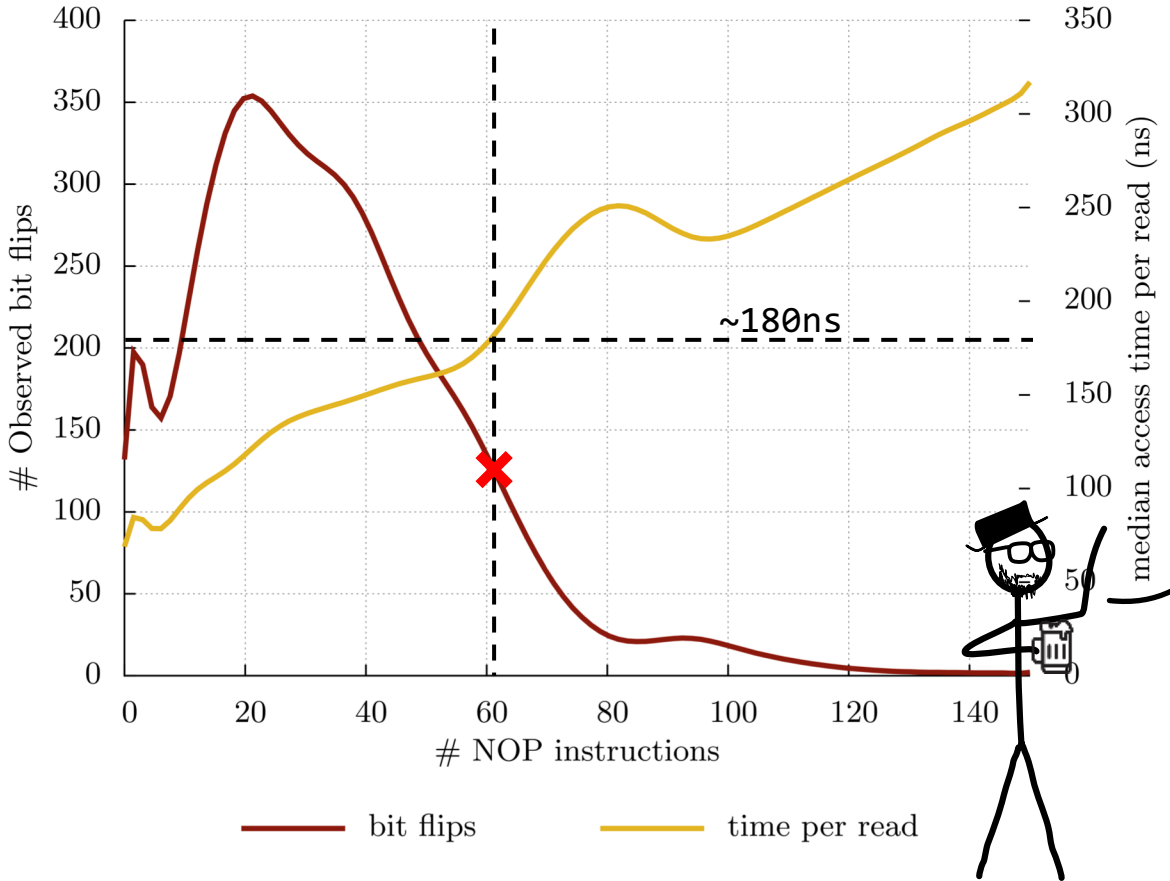
Caches: Small & Deterministic

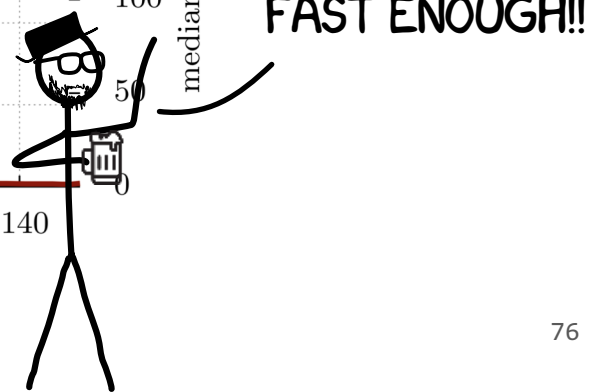
Steps:

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 cache eviction ✓

#P3. Contiguous memory

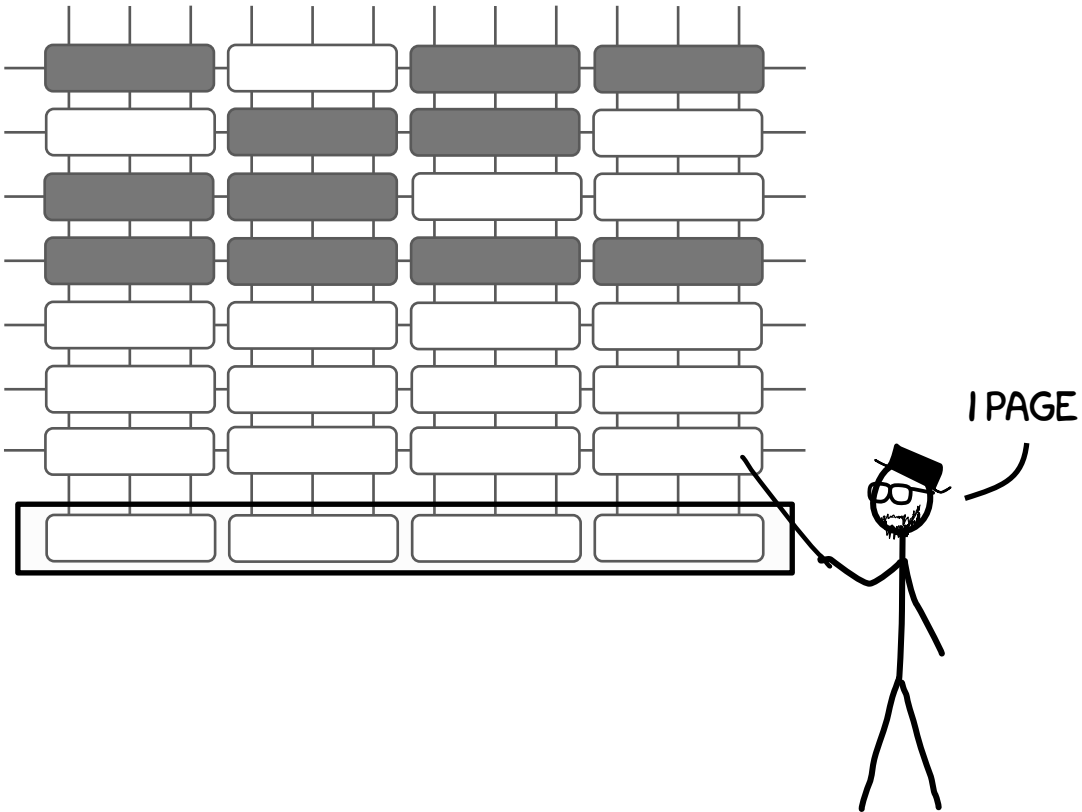
Attacker primitives

#P1. DRAM access ✓

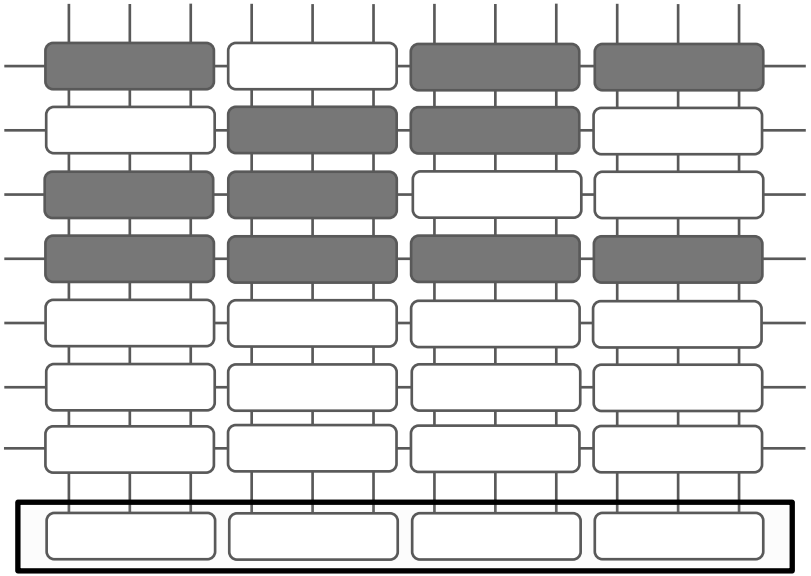
#P2. Fast cache eviction ✓

#P3. Contiguous memory

#P3. Memory Allocation

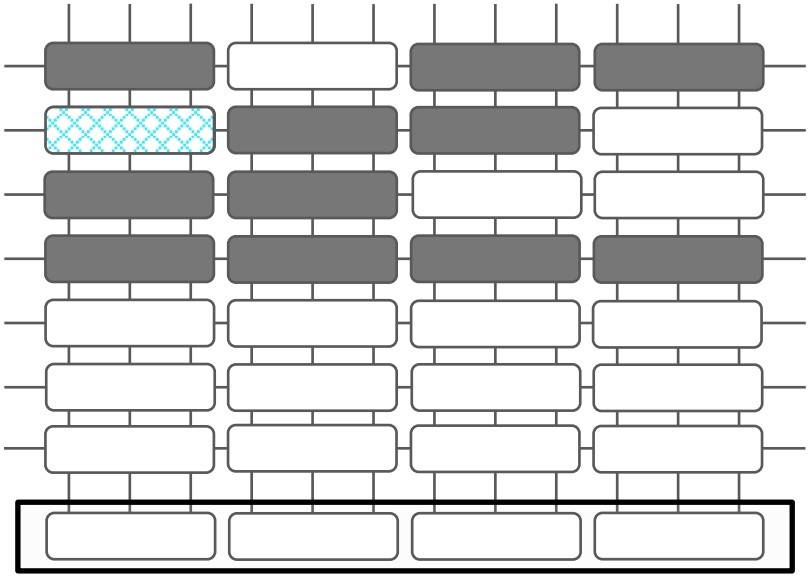


#P3. Memory Allocation



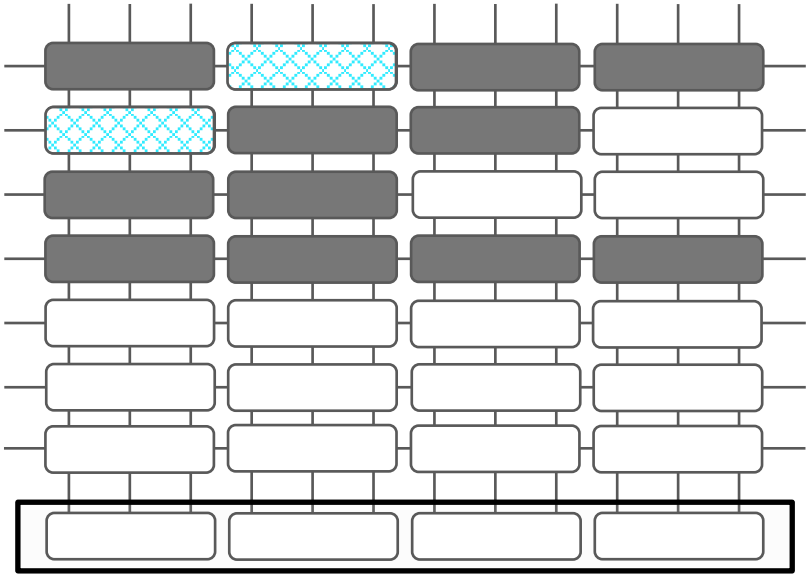
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



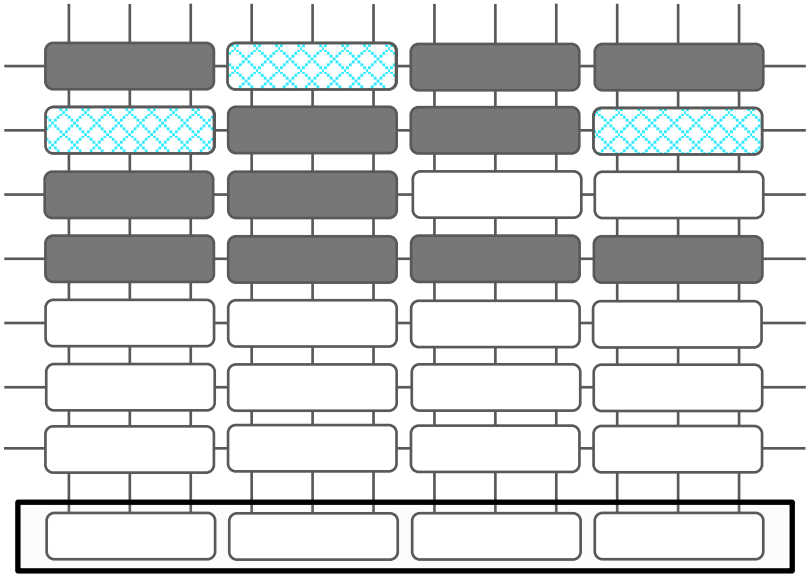
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



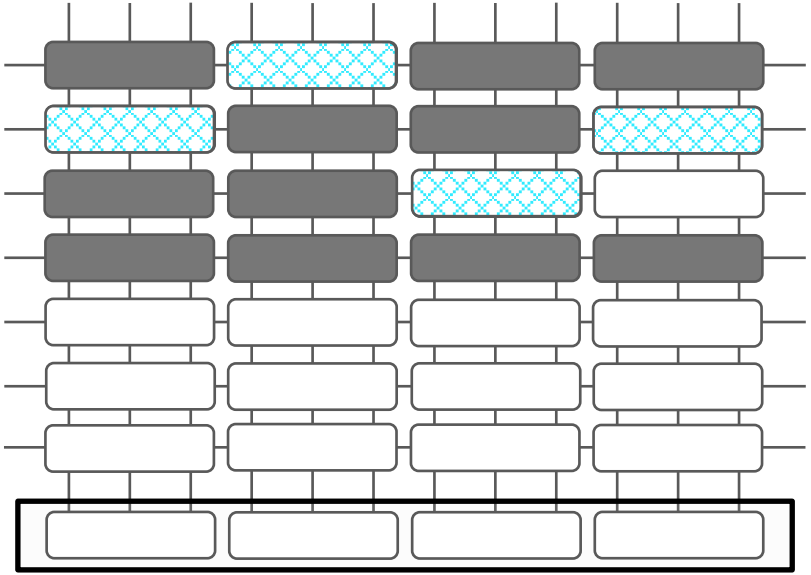
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



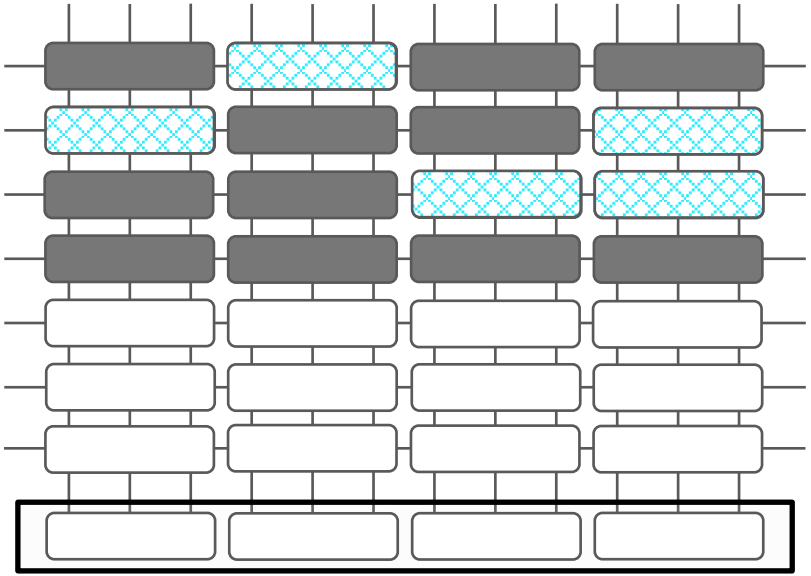
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



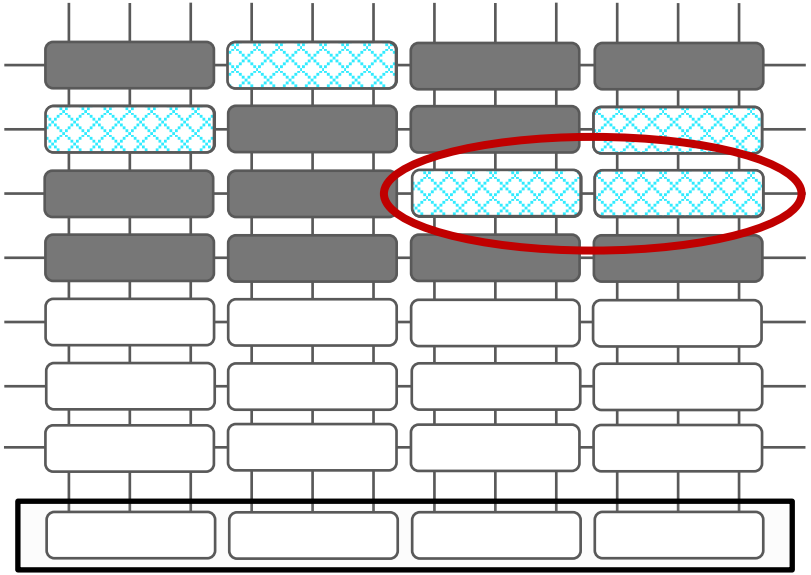
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



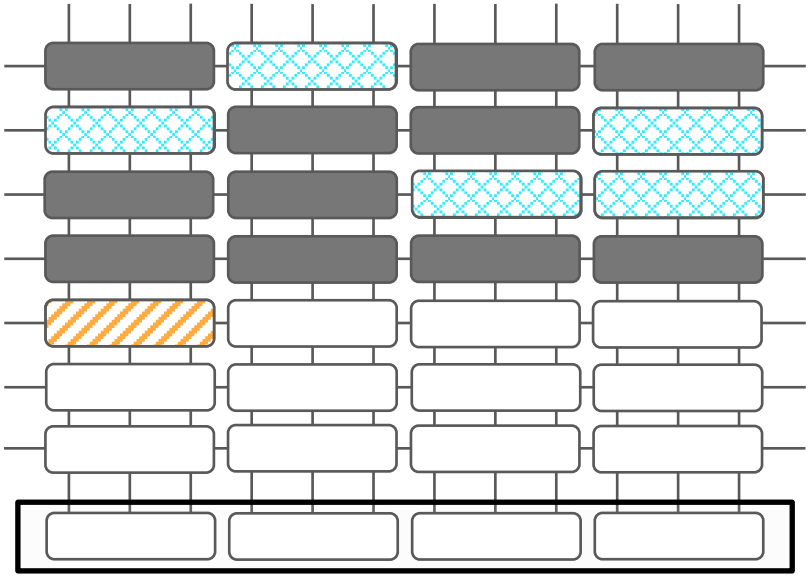
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



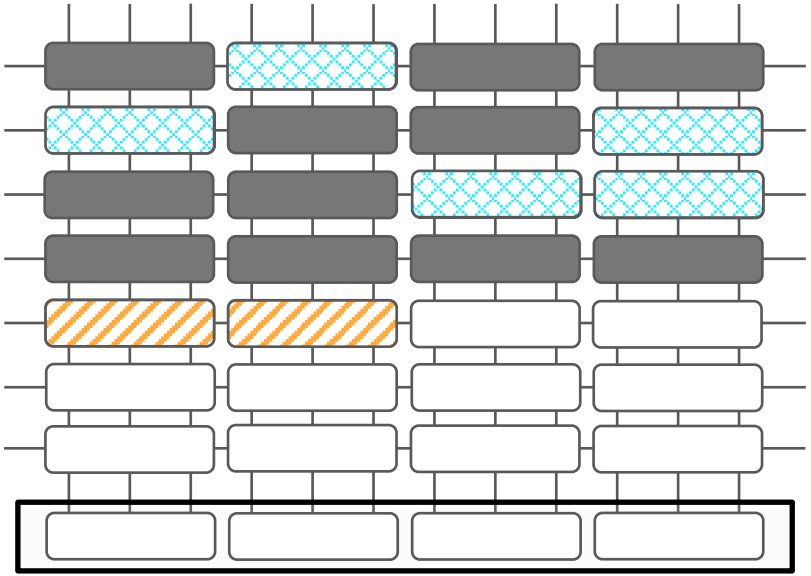
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



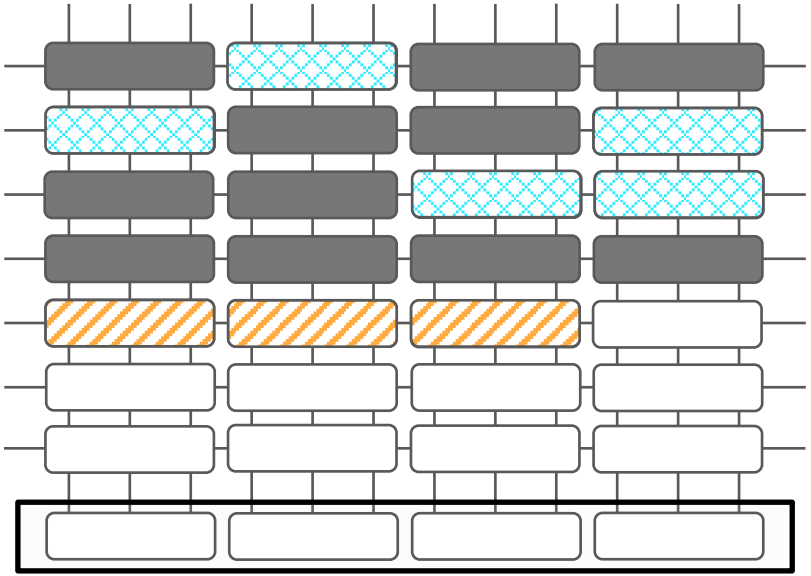
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



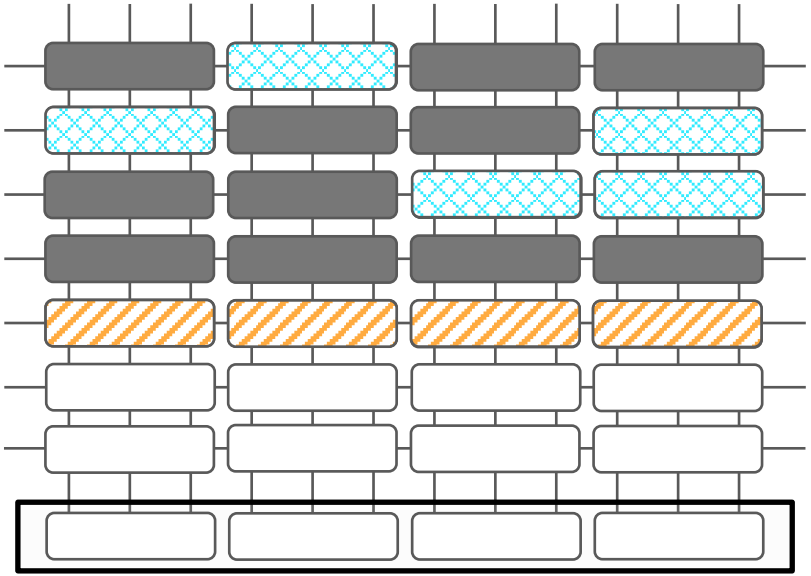
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



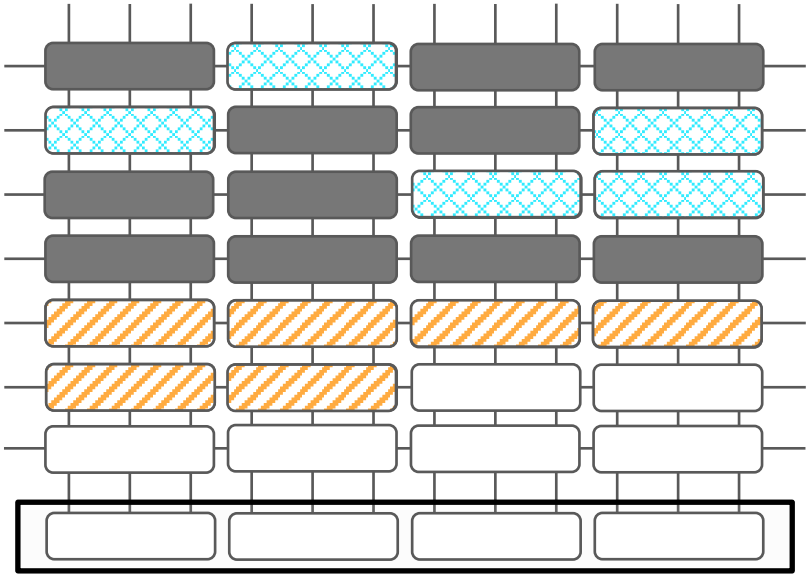
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



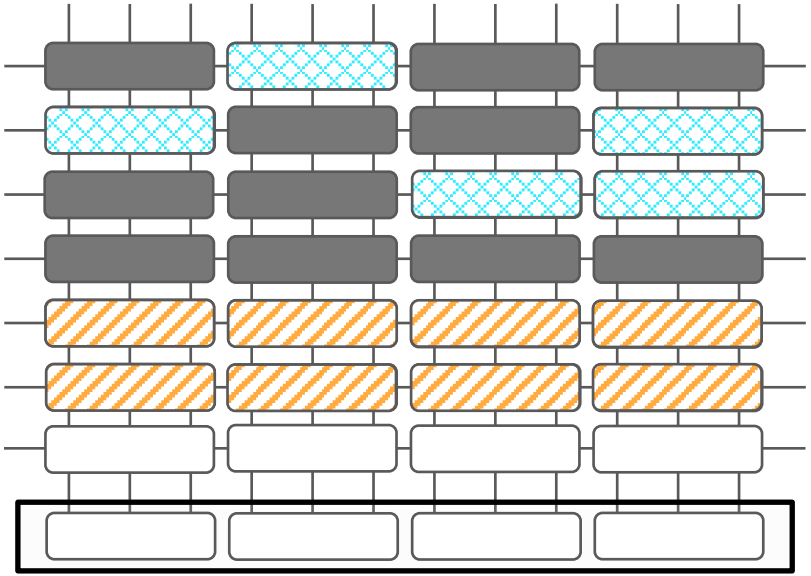
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



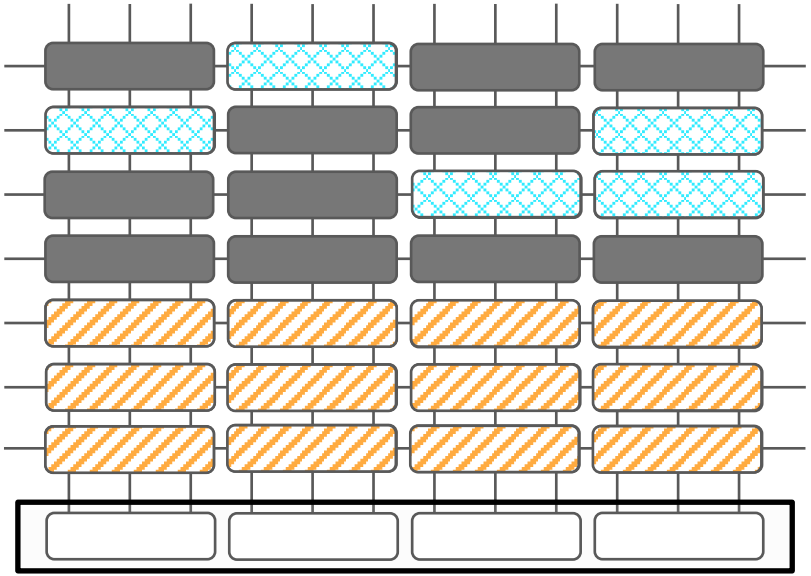
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



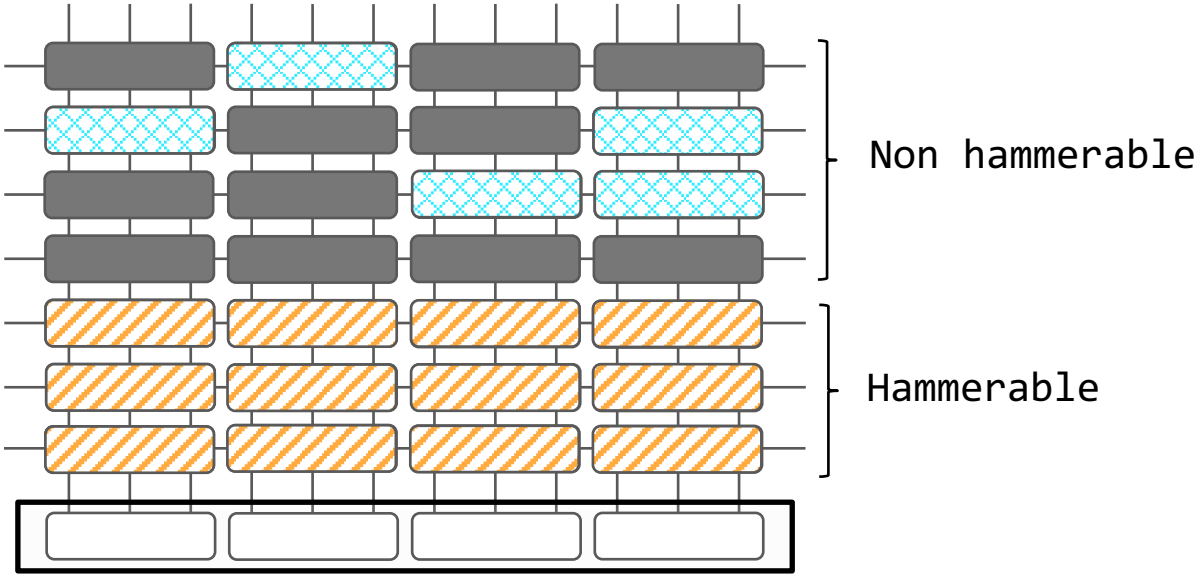
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation

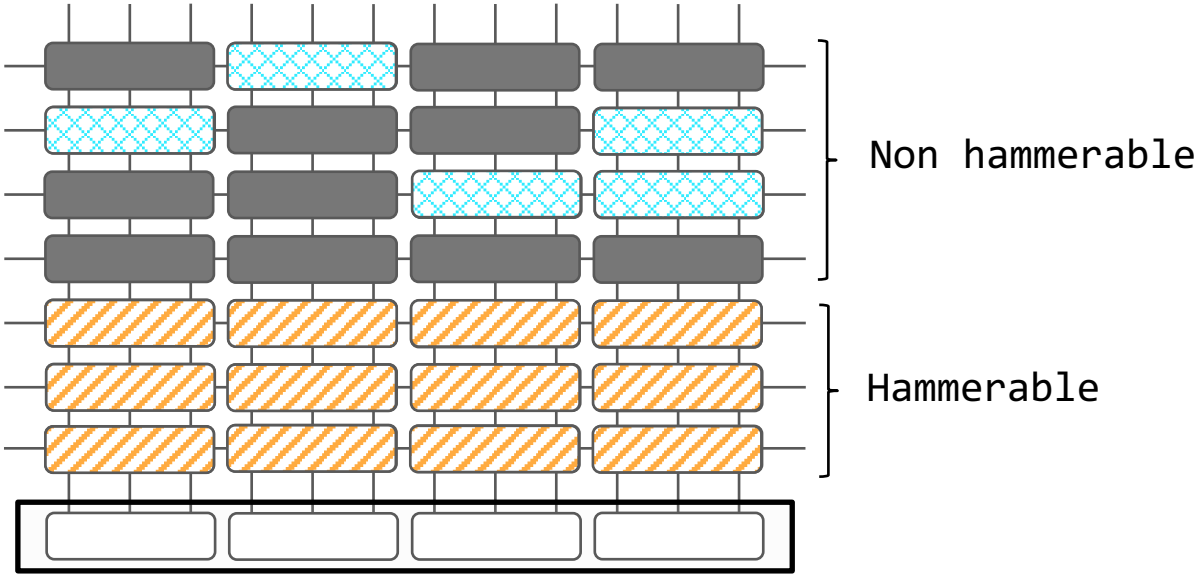


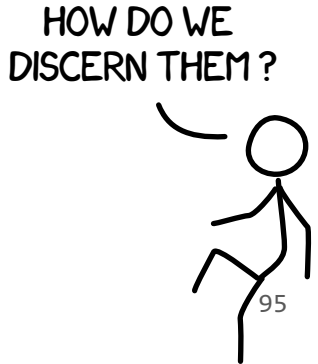
```
while (num_tex--) {
  // size 4KB
  tex[num_tex]= gl.createTexture();
    fill_tex(tex[num_tex])
}
```

#P3. Memory Allocation



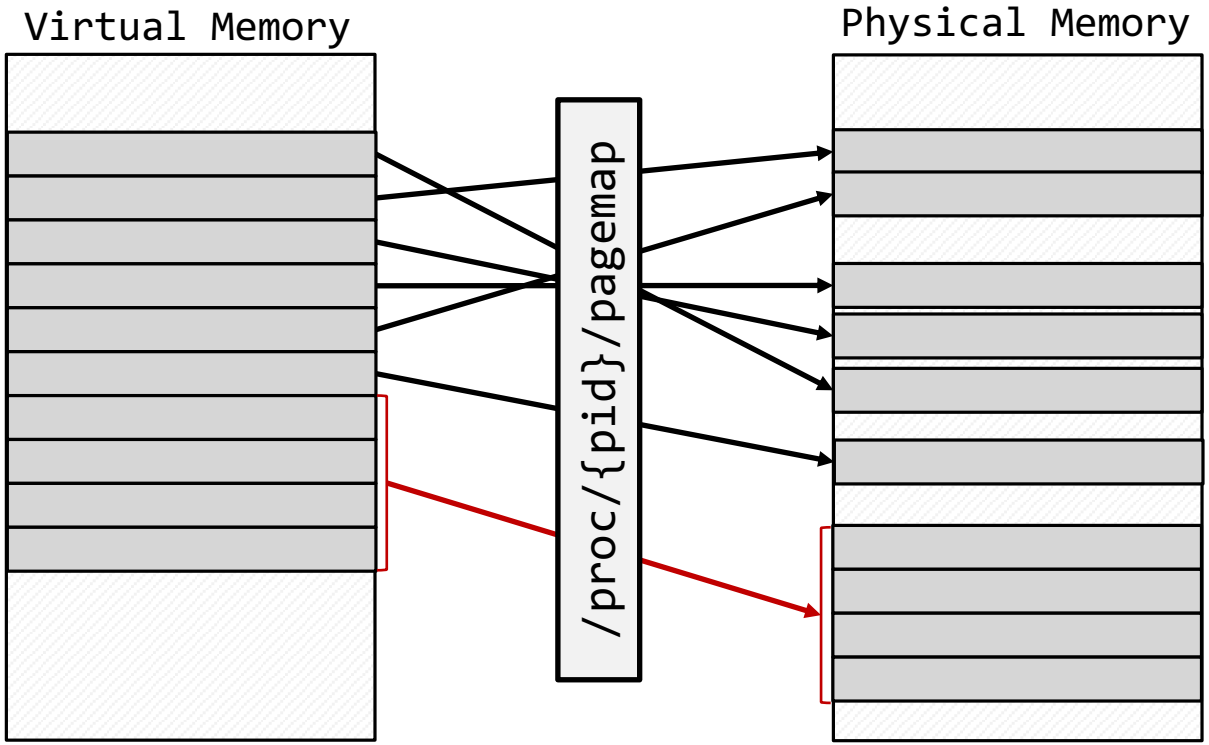
#P3. Memory Allocation






SIDE CHANNELS
SIDE CHANNELS EVERYWHERE
imgflip.com

#P3. Contig Memory detection



Attacker primitives

#P1. DRAM access ✓

#P2. Fast cache eviction ✓

#P3. Contiguous memory ✓

What do we do with these primitives?

GLitch

GLitch: in a nutshell

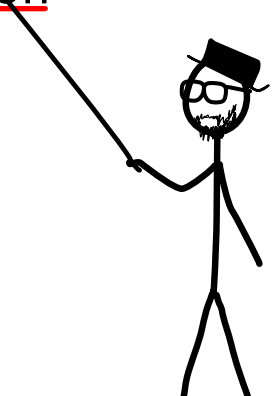
Flip feng shui in JS:

0. Allocate/detect contiguous memory (side channel)
1. Memory templating
2. Memory massaging
3. Exploitation

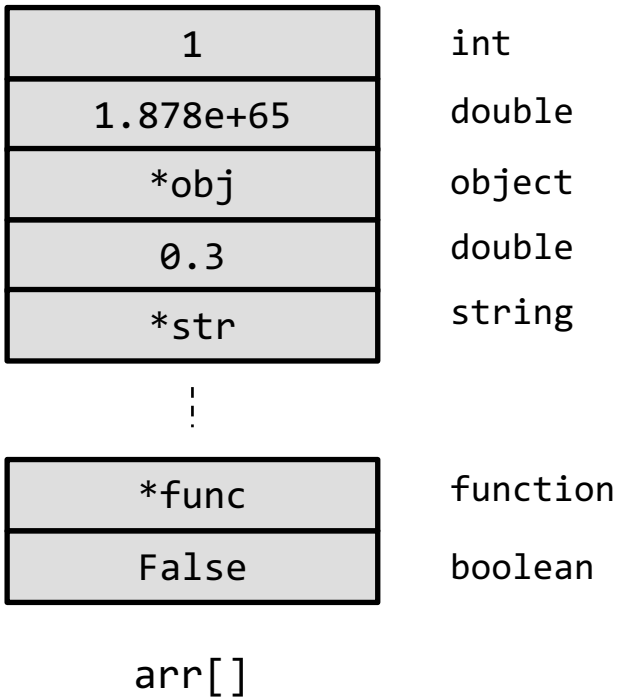
GLitch: in a nutshell

Flip feng shui in JS:

0. Allocate/detect contiguous memory (side channel)
1. Memory templating
2. Memory massaging
3. Exploitation

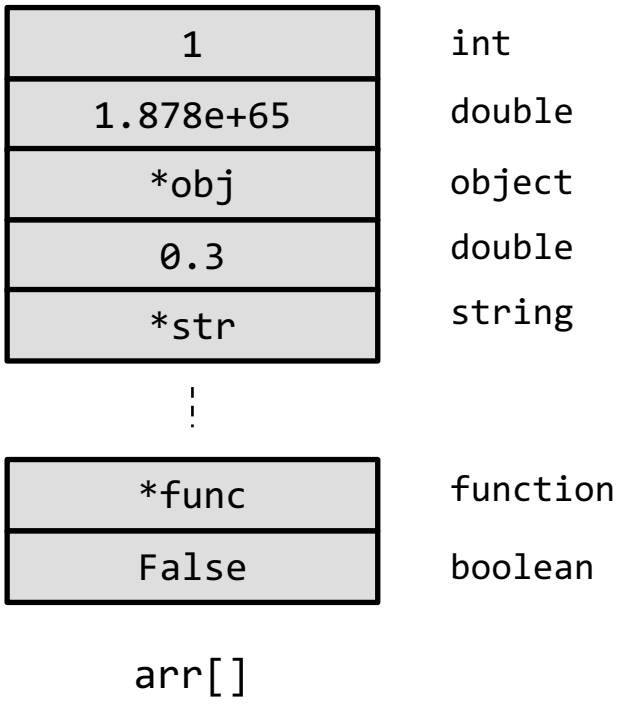


Exploitation: JS Arrays




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

Exploitation: JS Arrays



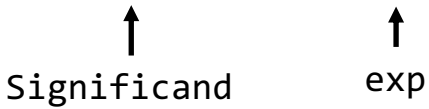

```
1 var arr = new Array(100);
2 arr[0] = 1 // int
3 arr[1] = 1.878e+65 // double
4 arr[3] = new Array(0x12) // object
...
```

NaN-boxing

IEEE-754 floating point (double)

1.125 = 1125 * 10^(-3)

11.25 = 1125 * 10^(-2)



IEEE-754 floating point (double)

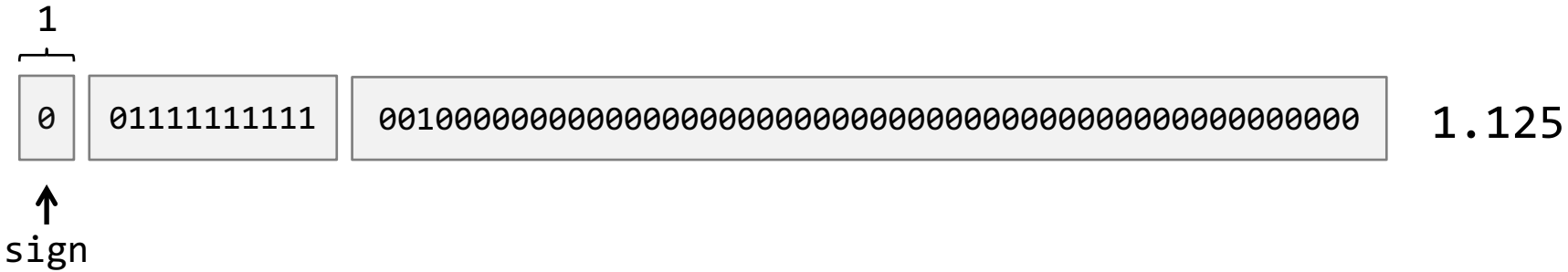
```
1.125  = 1125 * 10^(-3)
           ==     !=
11.25  = 1125 * 10^(-2)
```

IEEE-754 floating point (double)

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

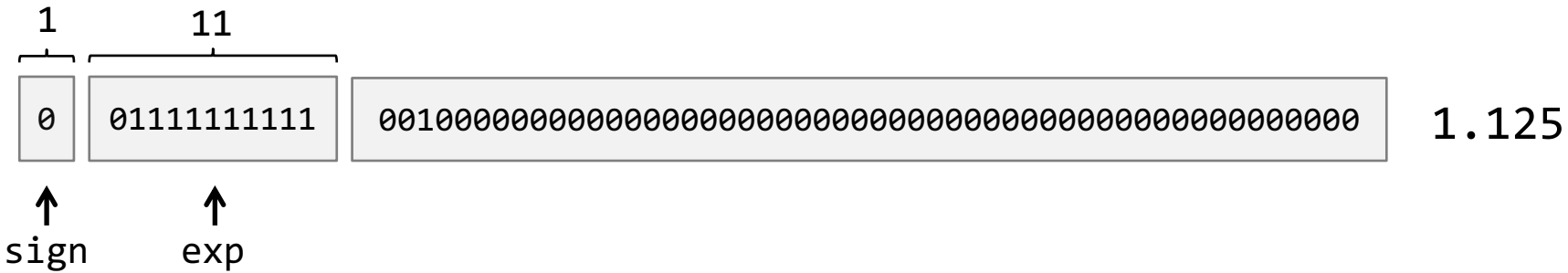
IEEE-754 floating point (double)

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



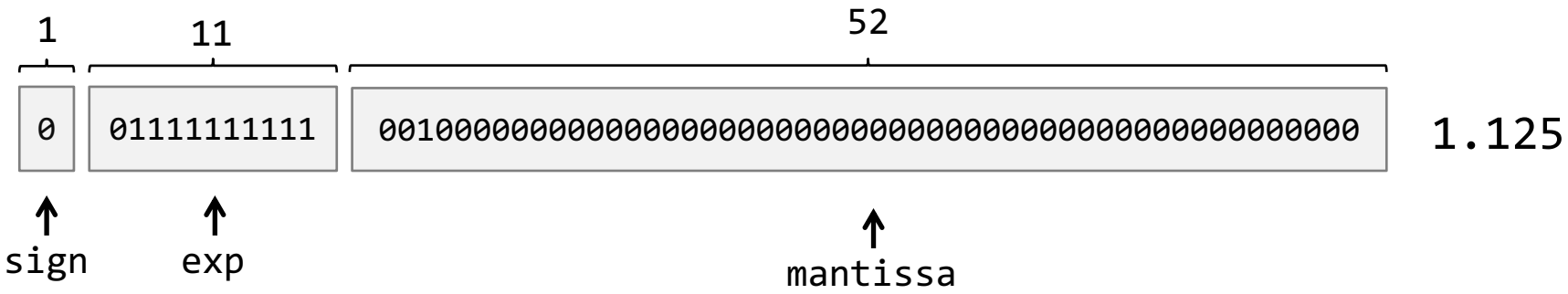
IEEE-754 floating point (double)

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



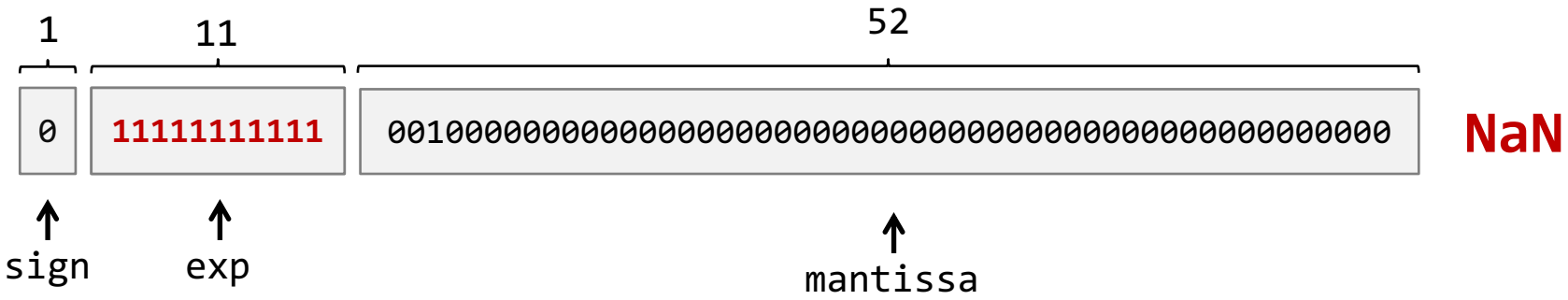
IEEE-754 floating point (double)

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



IEEE-754 floating point (double)

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

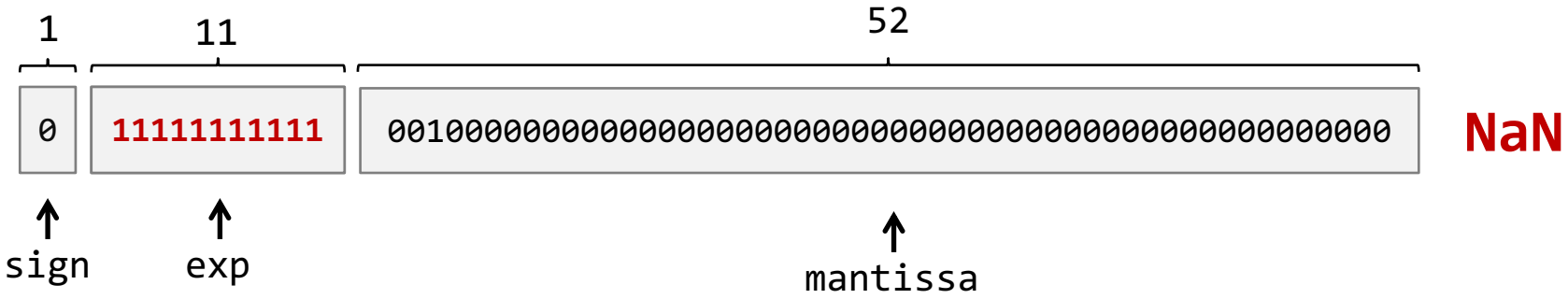


IEEE-754 floating point (double)

1.125 = 1125 * 10^(-3)

== !=

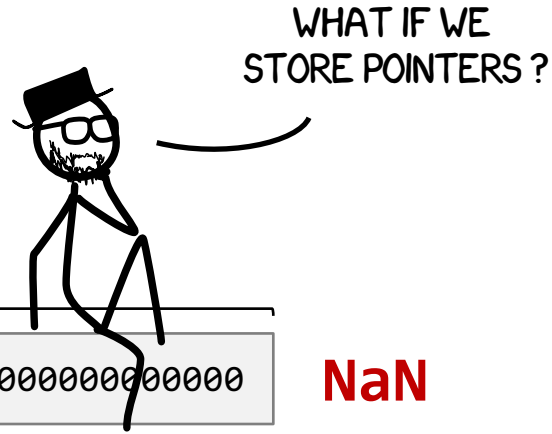
11.25 = 1125 * 10^(-2)



2^53-1 unused values

IEEE-754 floating point (double)

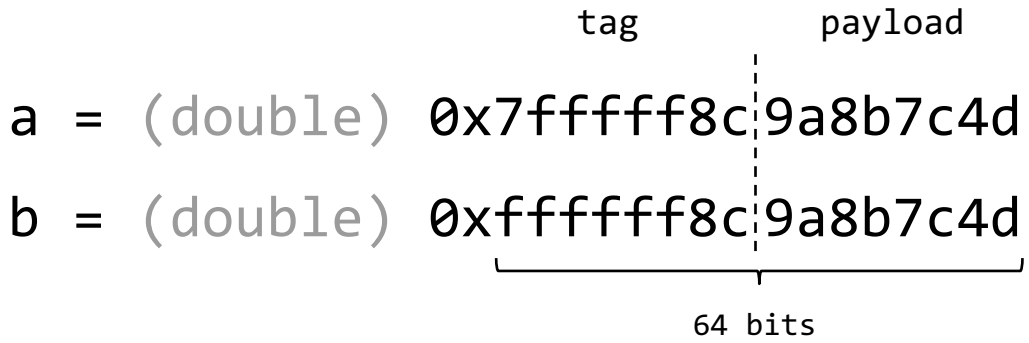
```
1.125  = 1125 * 10^(-3)
       ==     !=
11.25  = 1125 * 10^(-2)
```

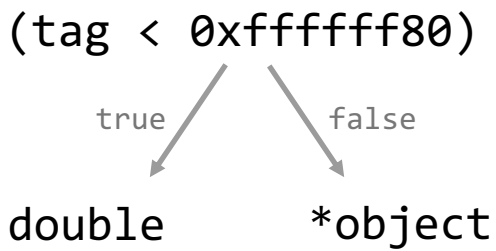


NaN

2^53-1 unused values

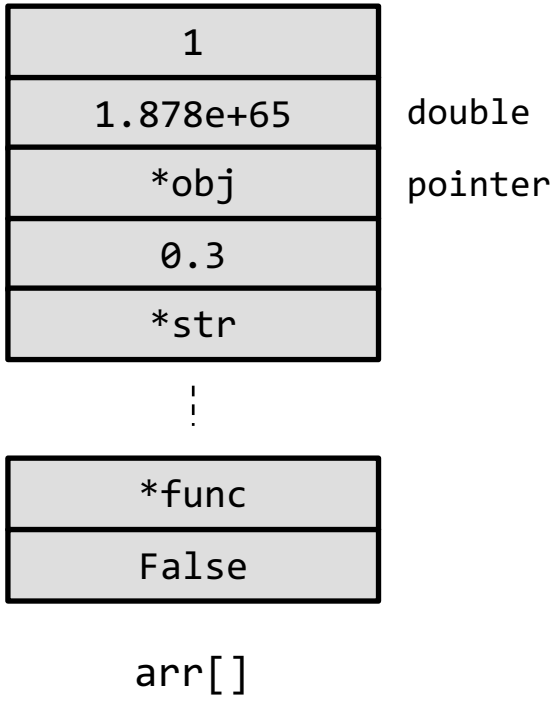
Exploitation: NuN-boxing (32bit)





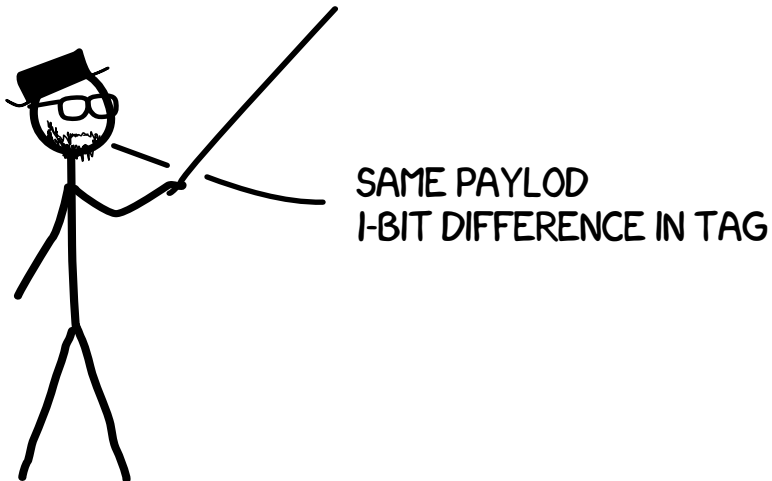
Exploitation: NuN-boxing (32bit)

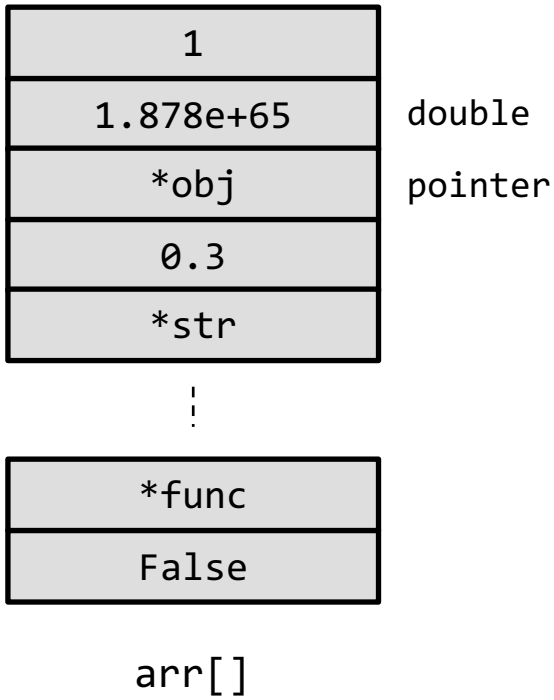
```
arr[1] = 0x7fffff8c 9a8b7c4d
arr[2] = 0xffffff8c 9a8b7c4d
```



Exploitation: NuN-boxing (32bit)

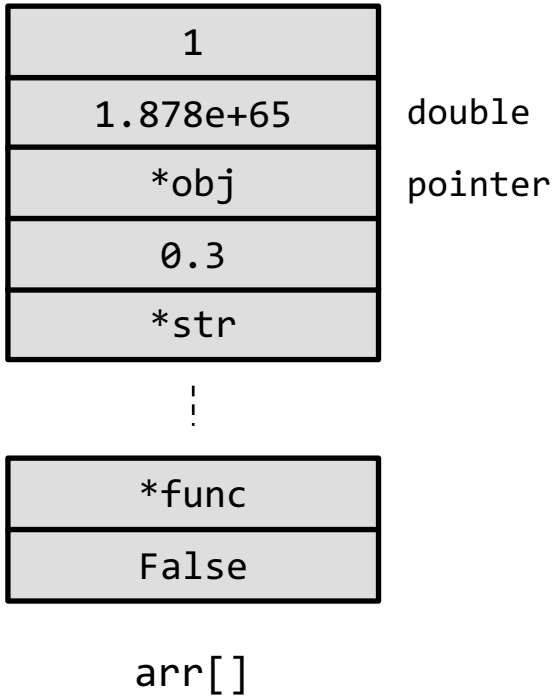

```
arr[1] = 0x7fffff8c 9a8b7c4d
arr[2] = 0xfffffff8c 9a8b7c4d
```





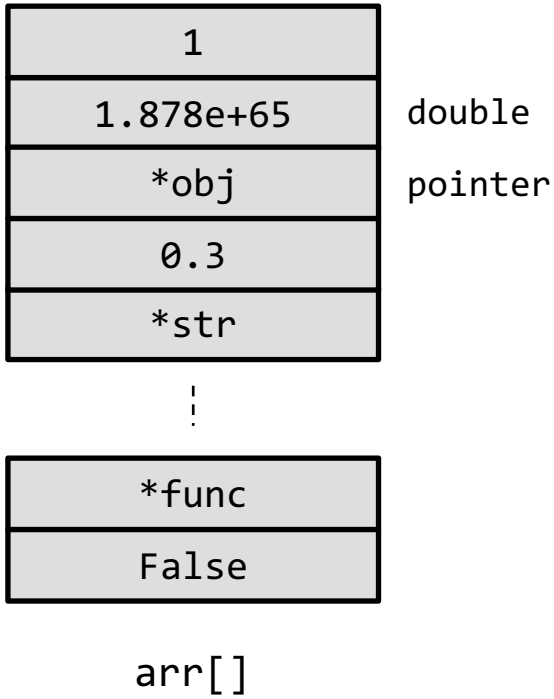
Exploitation: Type Flipping

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



Exploitation: Type Flipping

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



Exploitation: Type Flipping

Exploitation: Type Flipping

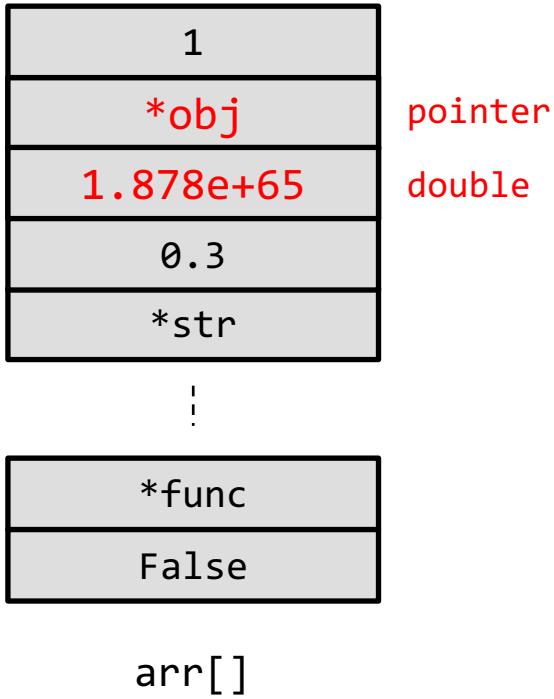
Exploitation: Type Flipping

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]



GLitch: in a nutshell

Flip feng shui in JS:

0. Allocate/detect contiguous memory (side channel)
1. Memory templating
2. Memory massaging
3. Exploitation (Arbitrary R/W)
    1. Break ASLR (1-to-0 flip)
    2. Reference fake object (0-to-1)


WHAT ABOUT YOU?

Assignment

Steps:

- Learn about OpenGL
- Learn about GPU performance counters
- Trigger bit flips natively
- Port bit flips to JavaScript
- Exploit (Break ASLR)

One week