Flipping Bits From Software with Rowhammer a Decade Later

Andrea Di Dio

$ whoami

- PhD Candidate at VUSec, Amsterdam
- Interested in Attacks/Defenses in the HW/SW boundary and OS Design (mostly MM)
- 4+ Years on Rowhammer 🔨
- 𝕏: @hammertux
- https://hammertux.github.io

“Sharing is caring.”

-Probably every kindergarten teacher ever-

*“Sharing is **NOT** caring.”*

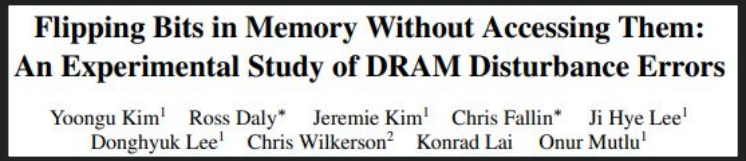
-Anonymous-

Why?

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

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

Why?



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

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

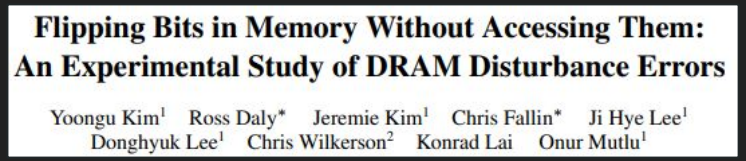
WIRED SUBSCRIBE

ANDY GREENBERG SECURITY AUG 31, 2016 7:00 AM

Forget Software—Now Hackers Are Exploiting Physics

An emerging form of hacking techniques targets the fundamental physical properties of computation.

Why?



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

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

WIRED SUBSCRIBE

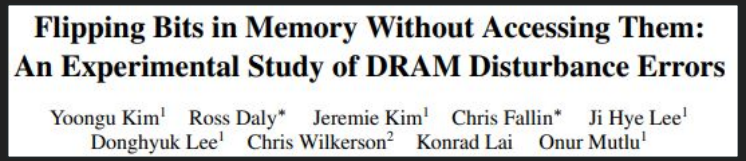
ANDY GREENBERG SECURITY AUG 31, 20

Forget Software—Now Hackers Are Exploiting Physics

An emerging form of hacking techniques targets the fundamental physical properties of computation.

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

Why?



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

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

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY AUG 31, 20

Forget Software—Now Hackers Are Exploiting Physics

h of hacking techniques
mental physical properties

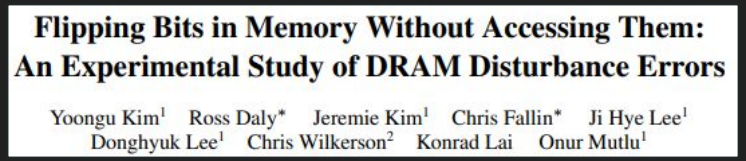
Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

THROWHAMMER —

Packets over a LAN are all it takes to trigger serious Rowhammer bit flips

The bar for exploiting potentially serious DDR weakness keeps getting lower.

Why?



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

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

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY AUG 31, 20

Forget Software—Now Hackers Are Exploiting Physics

h of hacking techniques
mental physical properties

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

THROWHAMMER —

Packets over a LAN are all it takes to trigger serious Rowhammer bit flips

The bar for exploiting potentially serious DDR weakness keeps getting lower.

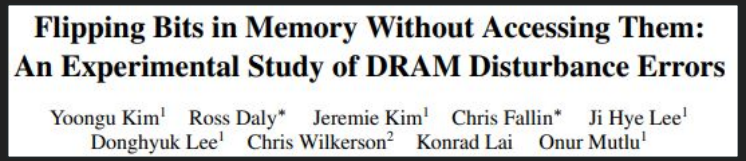
WIRED SUBSCRIBE

LILY HAY NEWMAN SECURITY NOV 21, 2018 3:31 PM

An Ingenious Data Hack Is More Dangerous Than Anyone Feared

Researchers have discovered that the so-called Rowhammer technique works on "error-correcting code" memory, in what amounts to a serious escalation.

Why?



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

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

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY AUG 31, 20

Forget Software—Now Hackers Are Exploiting Physics

h of hacking techniques
mental physical properties

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

THROWHAMMER —

Packets over a LAN are all it takes to trigger serious Rowhammer bit flips

The bar for exploiting potentially serious DDR weakness keeps getting lower.

WIRED SUBSCRIBE

LILY HAY NEWMAN SECURITY NOV 21, 2018 3:31 PM

An Ingenious Data Hack Is More Dangerous Than Anyone Feared

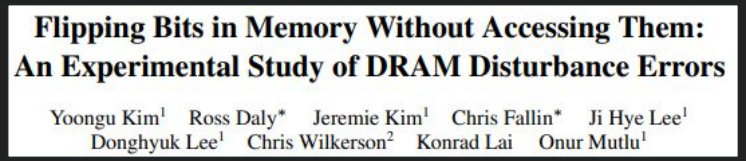
Researchers have discovered that the so-called Rowhammer technique works on "error-correcting code" memory, in what amounts to a serious escalation.

BLACKSMITH —

DDR4 memory protections are broken wide open by new Rowhammer technique

Researchers build "fuzzer" that supercharges potentially serious bitflipping exploits.

Why?



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

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

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY AUG 31, 20

Forget Software—Now Hackers Are Exploiting Physics

h of hacking techniques
mental physical properties

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms

THROWHAMMER —

Packets over a LAN are all it takes to trigger serious Rowhammer bit flips

The bar for exploiting potentially serious DDR weakness keeps getting lower.

WIRED SUBSCRIBE

LILY HAY NEWMAN SECURITY NOV 21, 2018 3:31 PM

An Ingenious Data Hack Is More Dangerous Than Anyone Feared

e so-
on
what

BLACKSMITH —

DDR4 memory protections are broken wide open by new Rowhammer technique

Researchers build "fuzzer" that supercharges potentially serious bitflipping exploits.

WIRED SUBSCRIBE

LILY HAY NEWMAN SECURITY MAY 26, 2021 10:01 AM

As Chips Shrink, Rowhammer Attacks Get Harder to Stop

A full fix for the "Half-Double" technique will require rethinking how memory semiconductors are designed.

Teaser

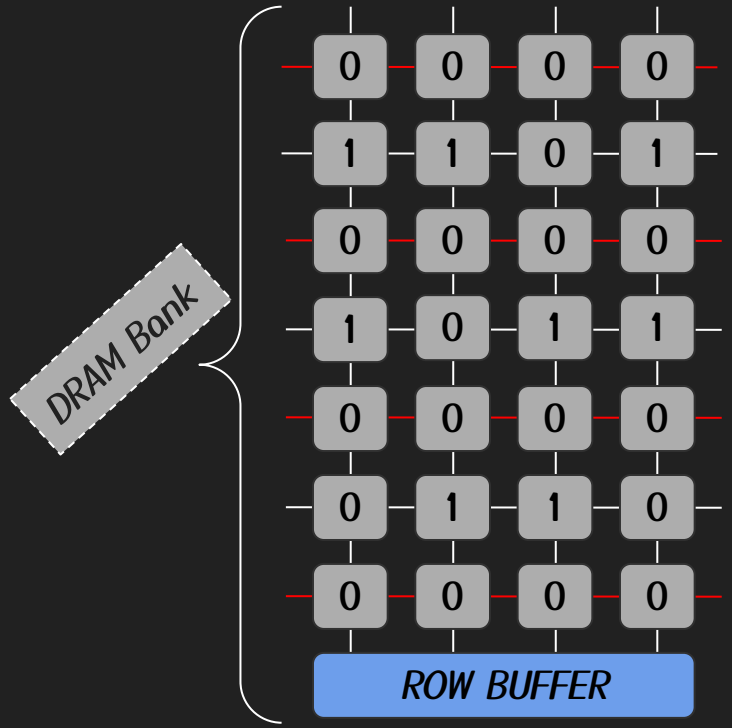
- Rowhammer is still an issue almost a decade later
- Full exploit workchain:
  - Co-hosted VMs
  - Page deduplication for memory massaging
  - Opcode Flipping on the sudo binary

Roadmap

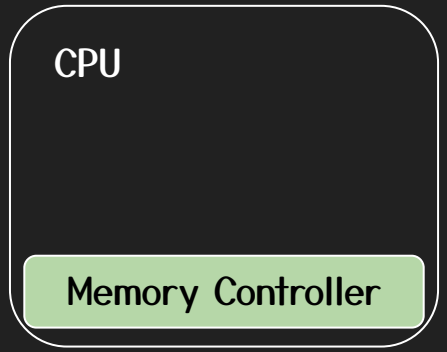
1. DRAM Timing Side Channel
2. Rowhammer & Templating
3. Opcode Flipping
4. Page Deduplication
5. Demo

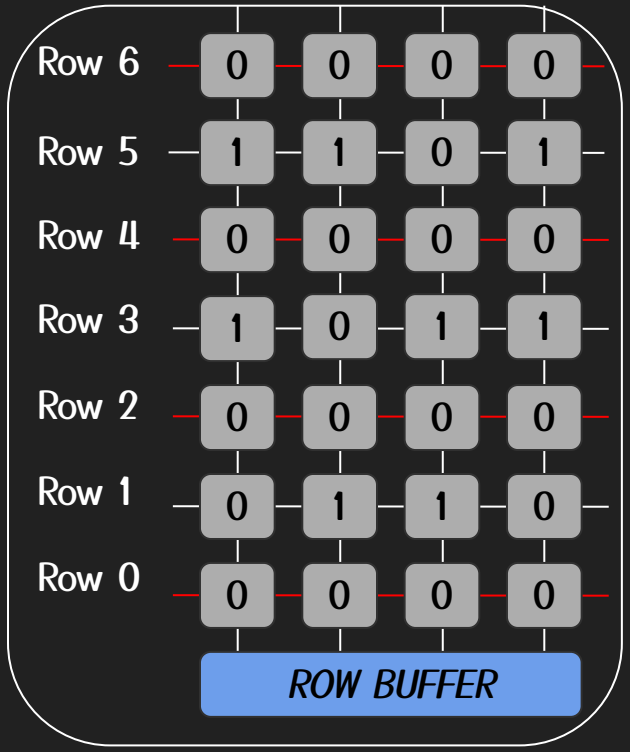
DRAM Primer

- Each DRAM Bank = Logical matrix of cells
- Row buffer: per bank ‘cache’
- Accessing a row causes a row activation
- Periodic refresh issued by the memory controller (64ms)

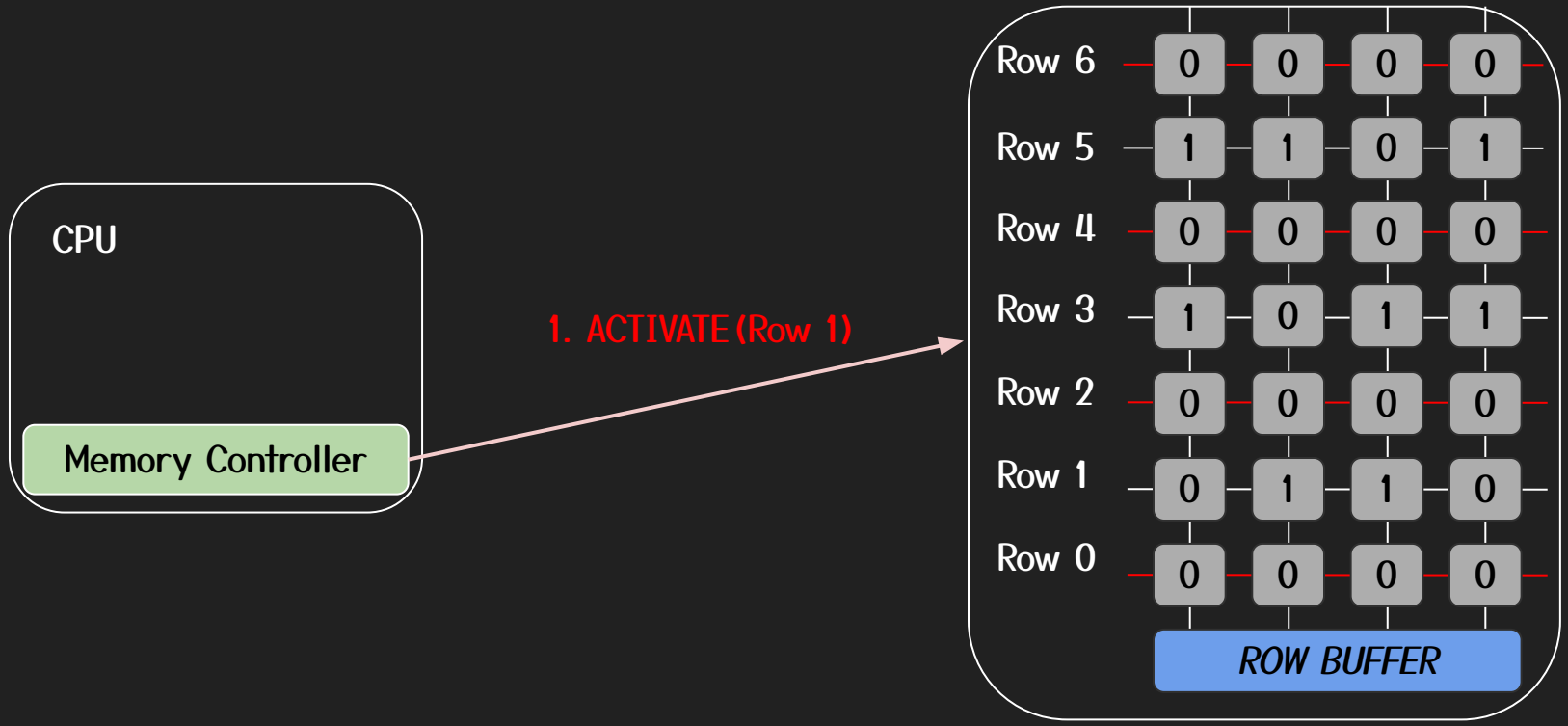


Reading from DRAM

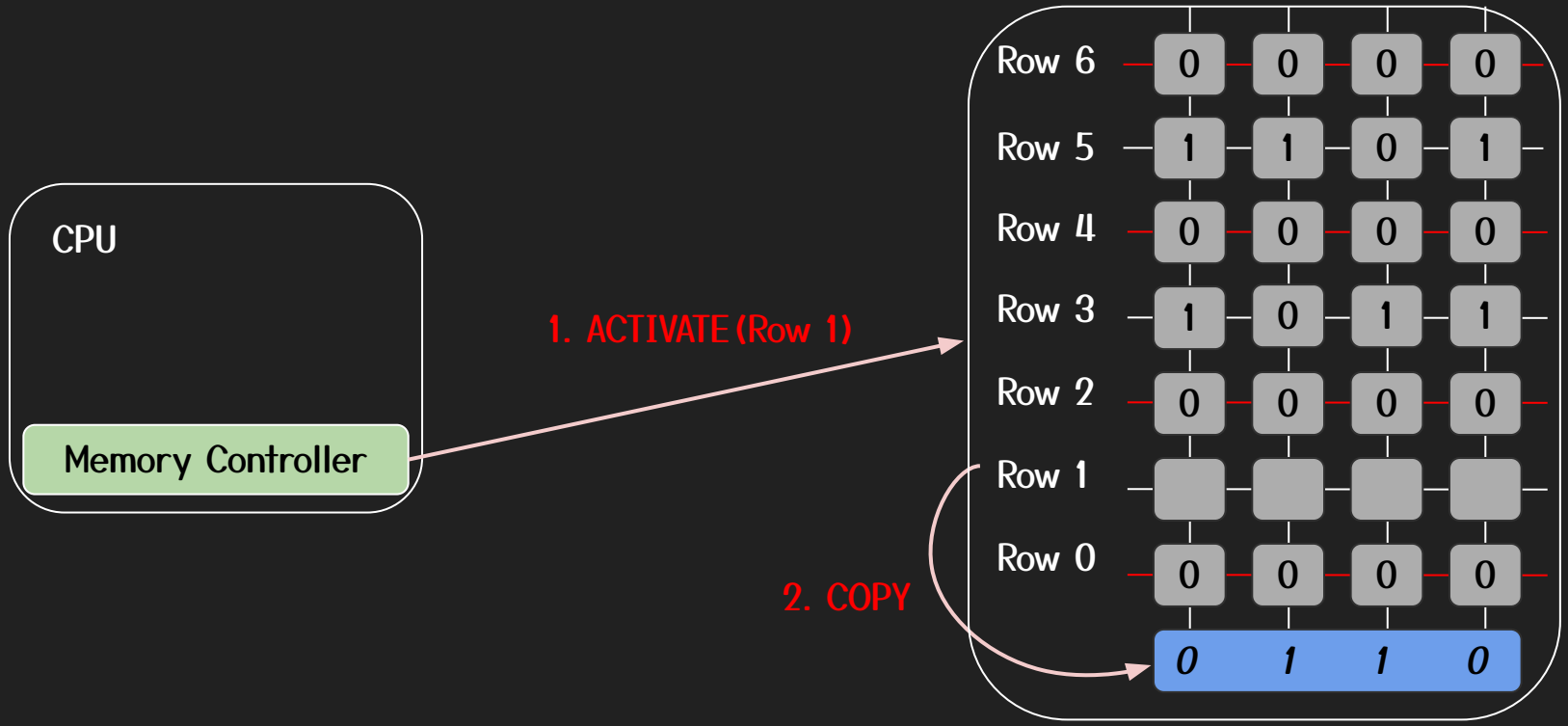




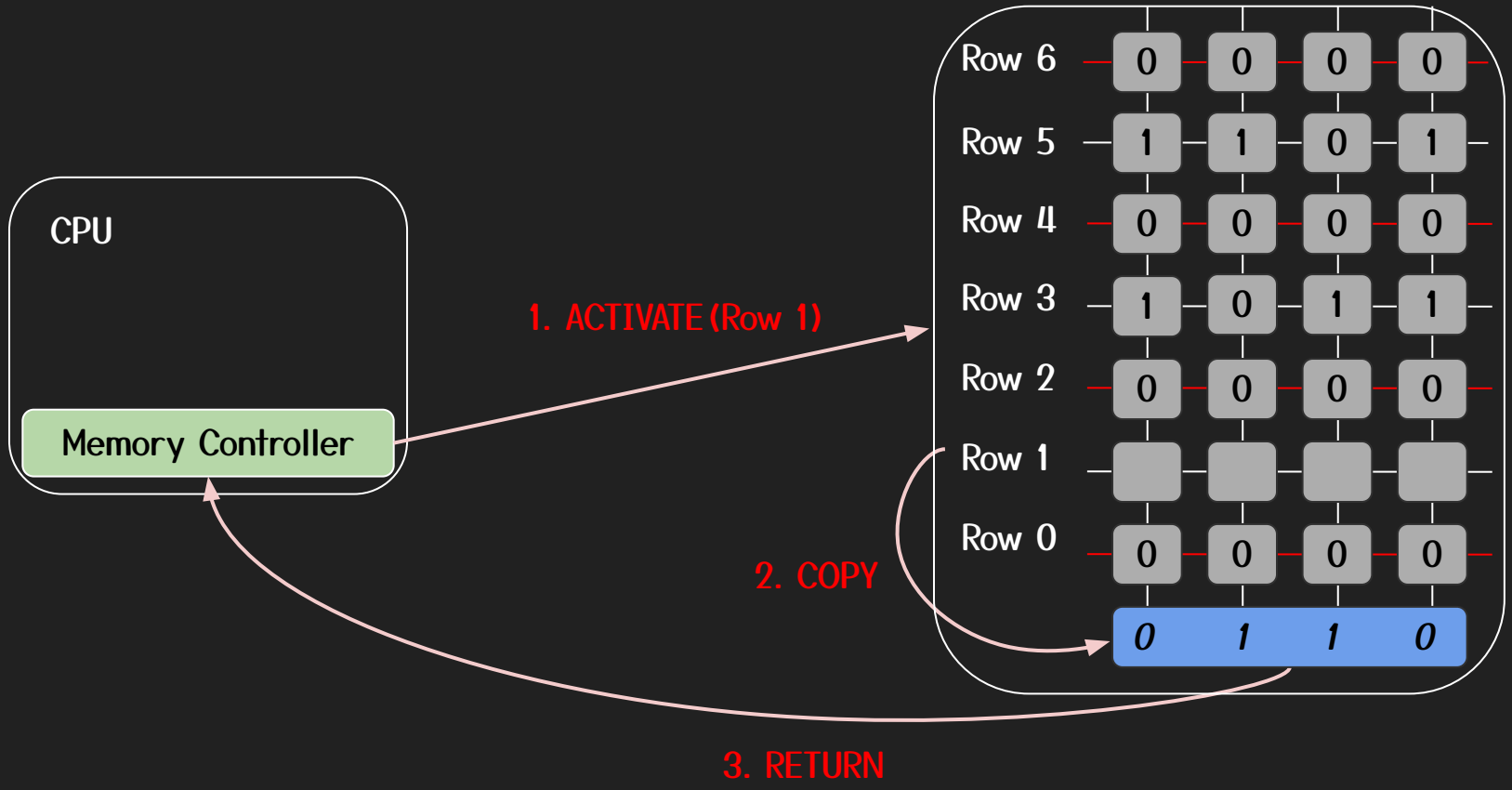
Reading from DRAM



Reading from DRAM



Reading from DRAM



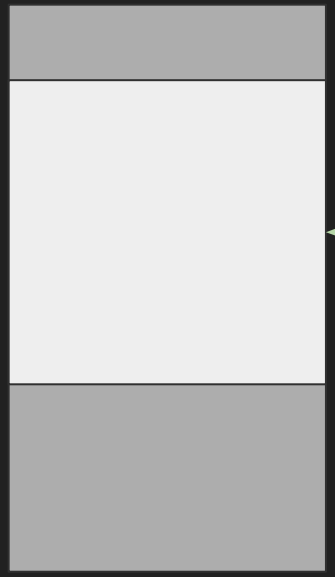
Row Replacement Policy

- When is the data copied back to the original row?
- “closing a row” → precharge
- Open row policy: postpone precharge until next row activation
- Closed row policy: precharge after returning the data to CPU
- Adaptive: Somewhere in between

DRAM Addressing

- Physical addresses are not mapped linearly to DRAM
- Complex XOR functions to ‘translate’ physical addresses to DRAM addresses
- Dram Address: hex tuple <Channel, DIMM, Rank, Bank, Column, Row>
- Functions recovered via Bank conflicts ⏰

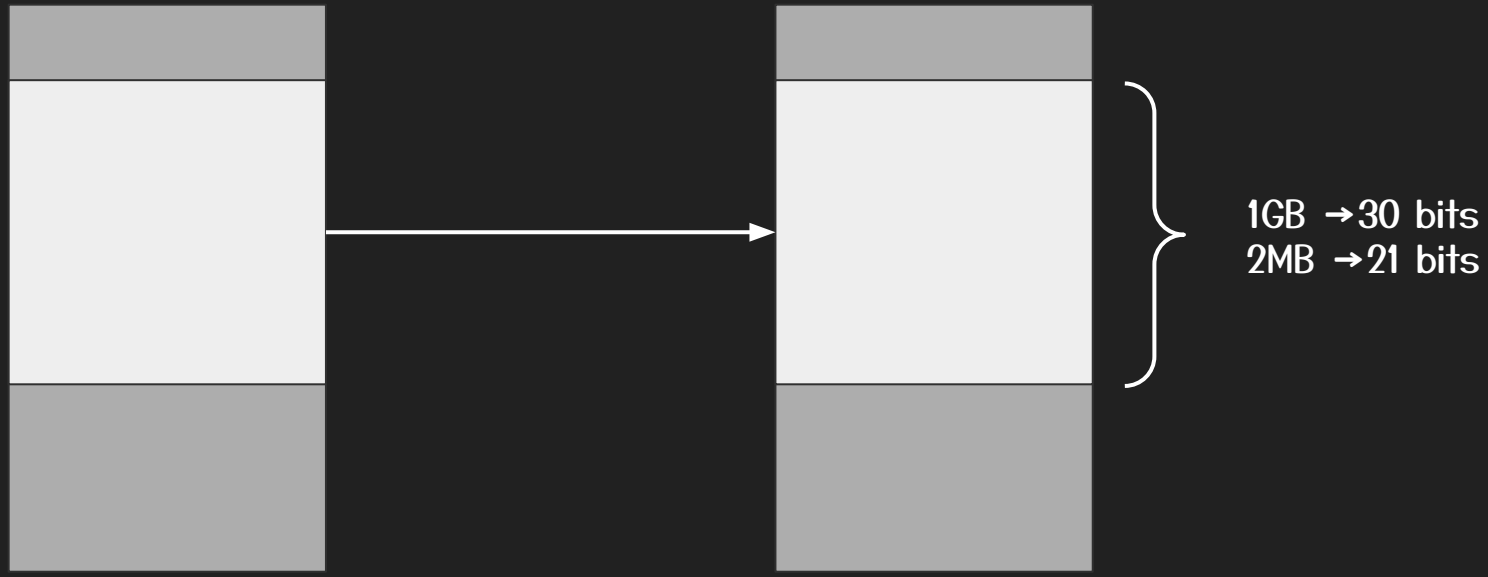
Reversing the DRAM Mapping



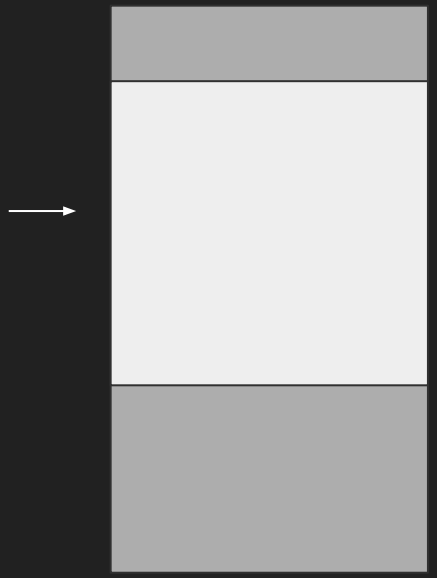
```
uint8_t *buf = mmap(NULL, .., HUGE_FLAGS, ..);
```

Virtual Address Space

Reversing the DRAM Mapping



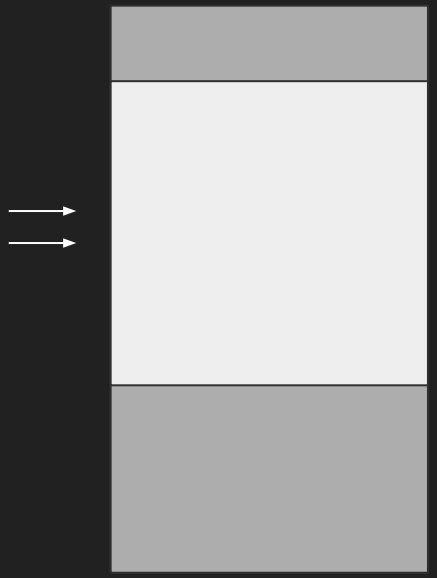
Reversing the DRAM Mapping



Virtual Address Space

```
uint8_t **probes = [*addr1,
```

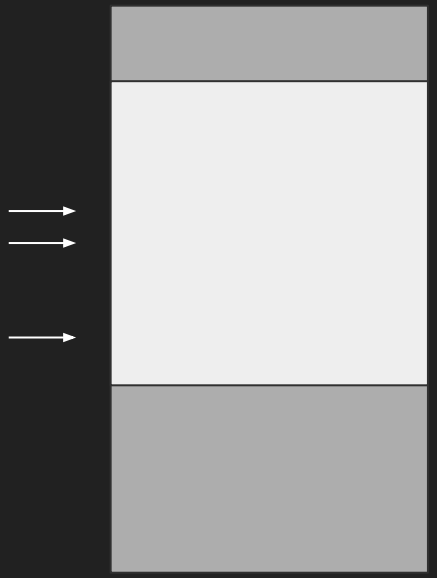
Reversing the DRAM Mapping



Virtual Address Space

```
uint8_t **probes = [*addr1, *addr2,
```

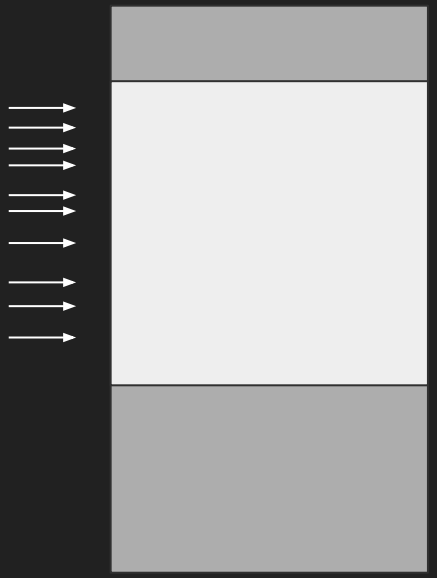
Reversing the DRAM Mapping



Virtual Address Space

```
uint8_t **probes = [*addr1, *addr2, *addr3,
```

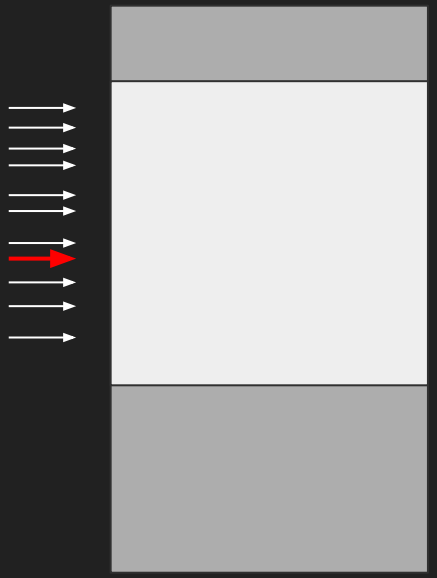
Reversing the DRAM Mapping



Virtual Address Space

```
uint8_t **probes = [*addr1, *addr2, *addr3, ..., *addrN]
```

Reversing the DRAM Mapping

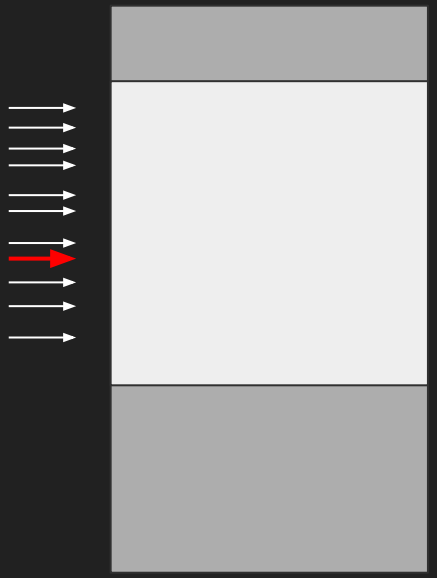


Virtual Address Space

```
uint8_t **probes = [*addr1, *addr2, *addr3, ..., *addrN]

uint8_t *base = buf + (rand() % BUF_SIZE);
```

Reversing the DRAM Mapping



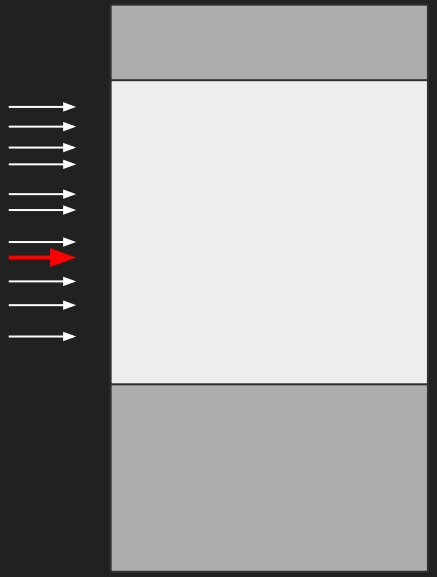
Virtual Address Space

```
uint8_t **probes = [*addr1, *addr2, *addr3, ..., *addrN]

uint8_t *base = buf + (rand() % BUF_SIZE);

for(auto addr : probes) {
    time_access(base, addr);
}
```

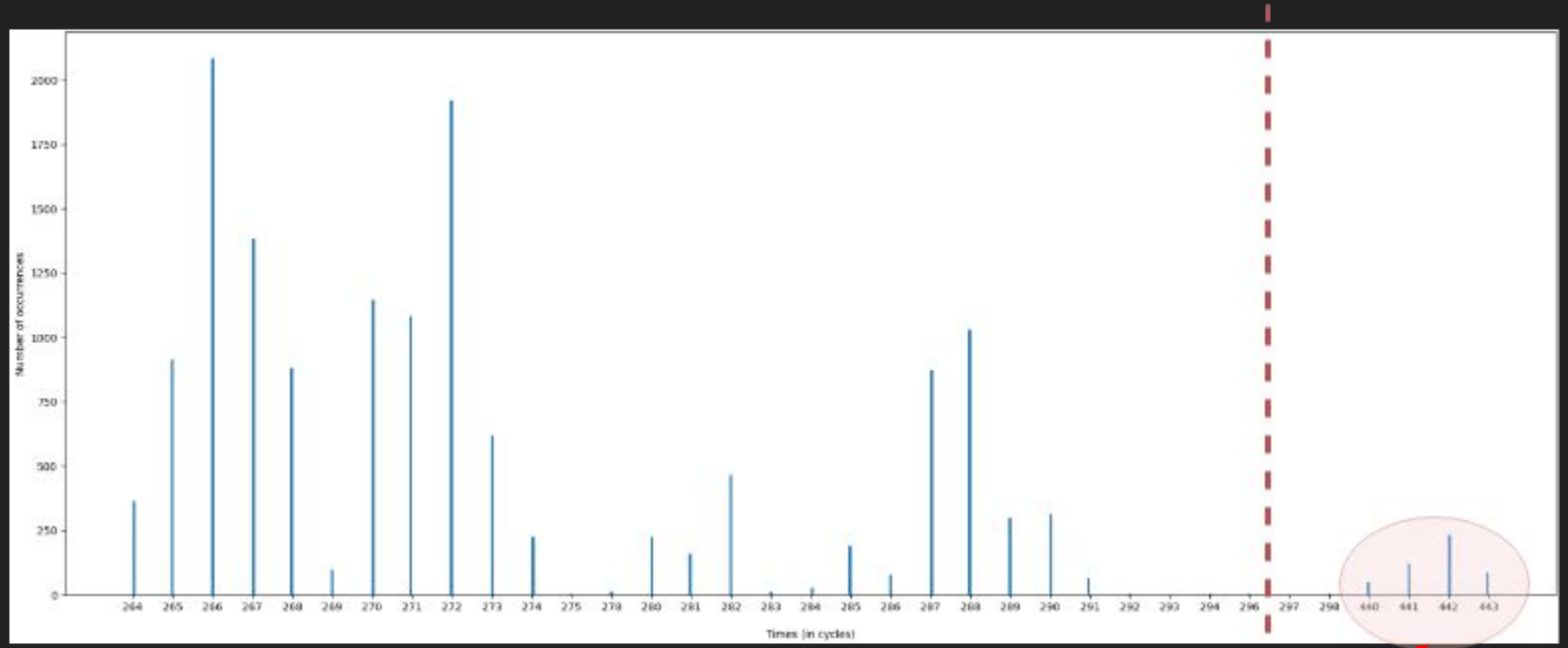
Reversing the DRAM Mapping



Virtual Address Space

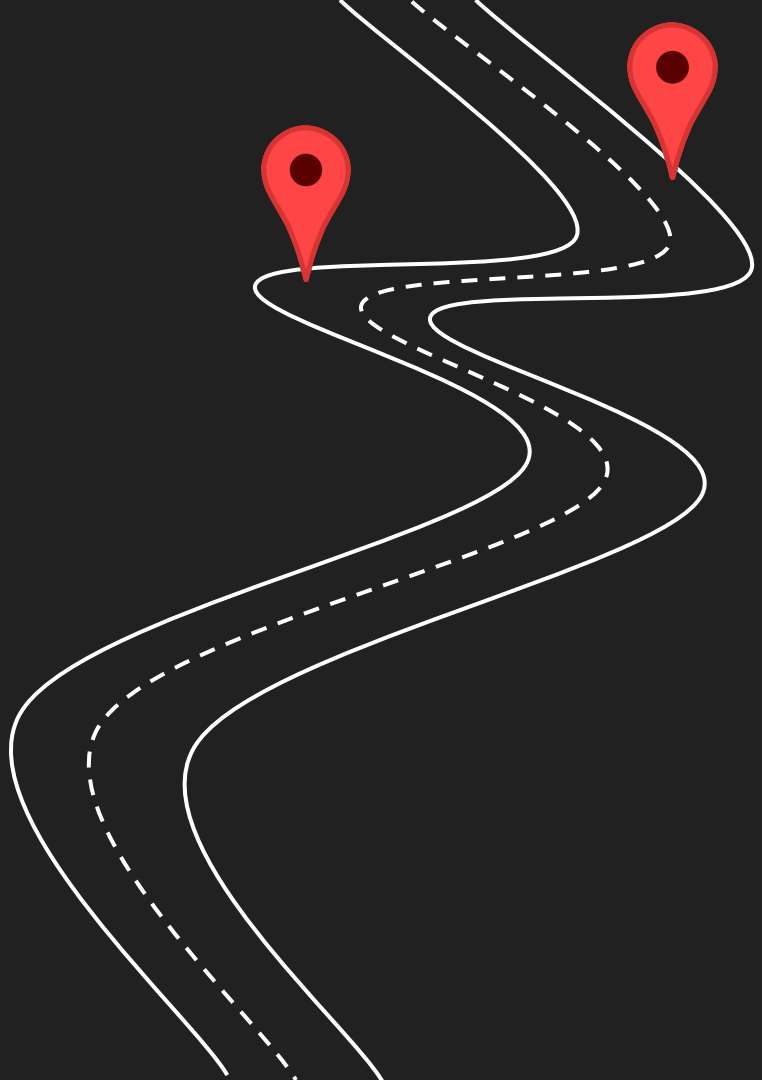
```
uint64_t time_access(volatile uint8_t *a, volatile uint8_t *b) {
    while(rounds--) {
        t_start = rdtscp();
        *a;
        *b;
        t_delta = rdtscp() - t_start;
        time_measurements[rounds] = t_delta;
        lfence();
        clflush(a);
        clflush(b);
        mfence();
    }
    return median(time_measurements);
}
```

Bank Conflicts



Roadmap

1. DRAM Timing Side Channel
2. Rowhammer & Templating
3. Opcode Flipping
4. Page Deduplication
5. Demo



Rowhammer Recipe

Three main steps to exploit Rowhammer:

1. Memory Templating
2. Memory Massaging
3. Rapid Hammering

Rowhammer 101

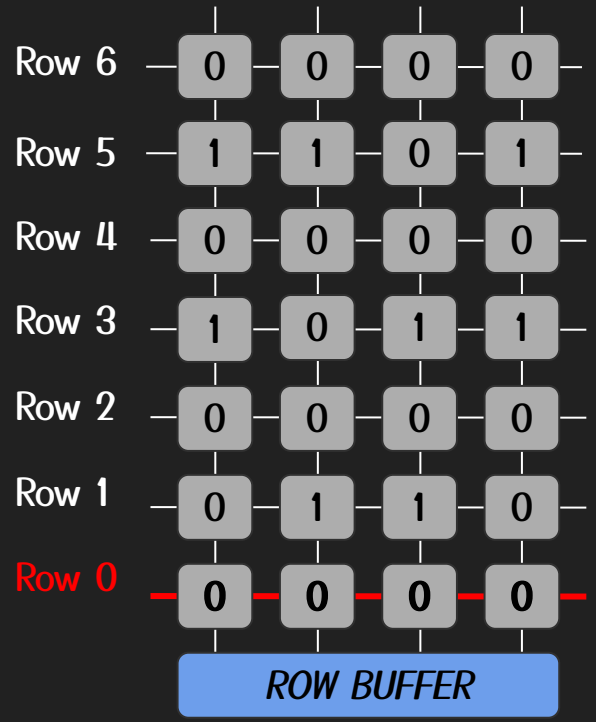
- Repeated row activations →bit flips in nearby rows
- Mostly deterministic and reproducible
- Both directions possible (1→0, 0→1)
- Most effective hammering pattern depends on DDR version

Rowhammer 101

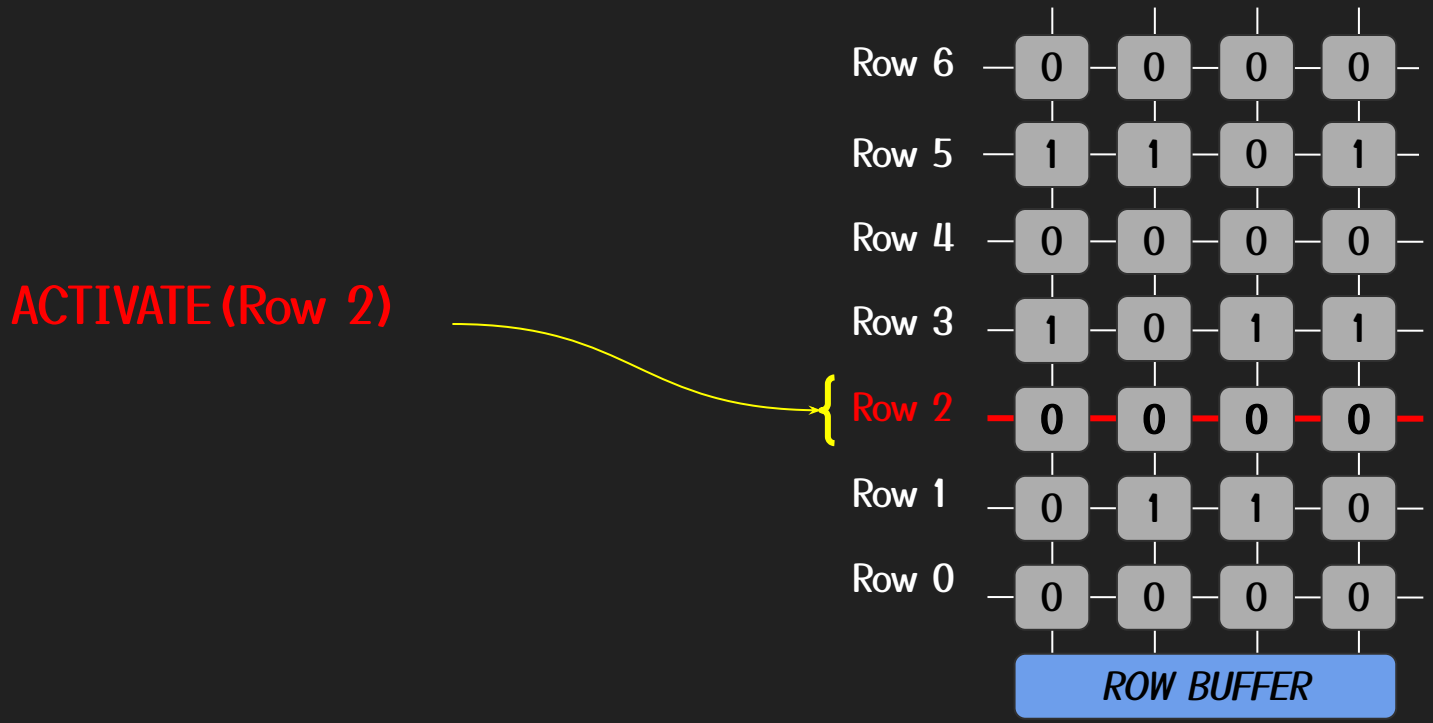
```
while(--n_activations) {
    for(auto row : aggressor_rows) {
        *row;
    }
    for(auto row : aggressor_rows) {
        asm volatile("clflush (%0)" :: "r" (row) : "memory");
    }
}
```

Rowhammer 101

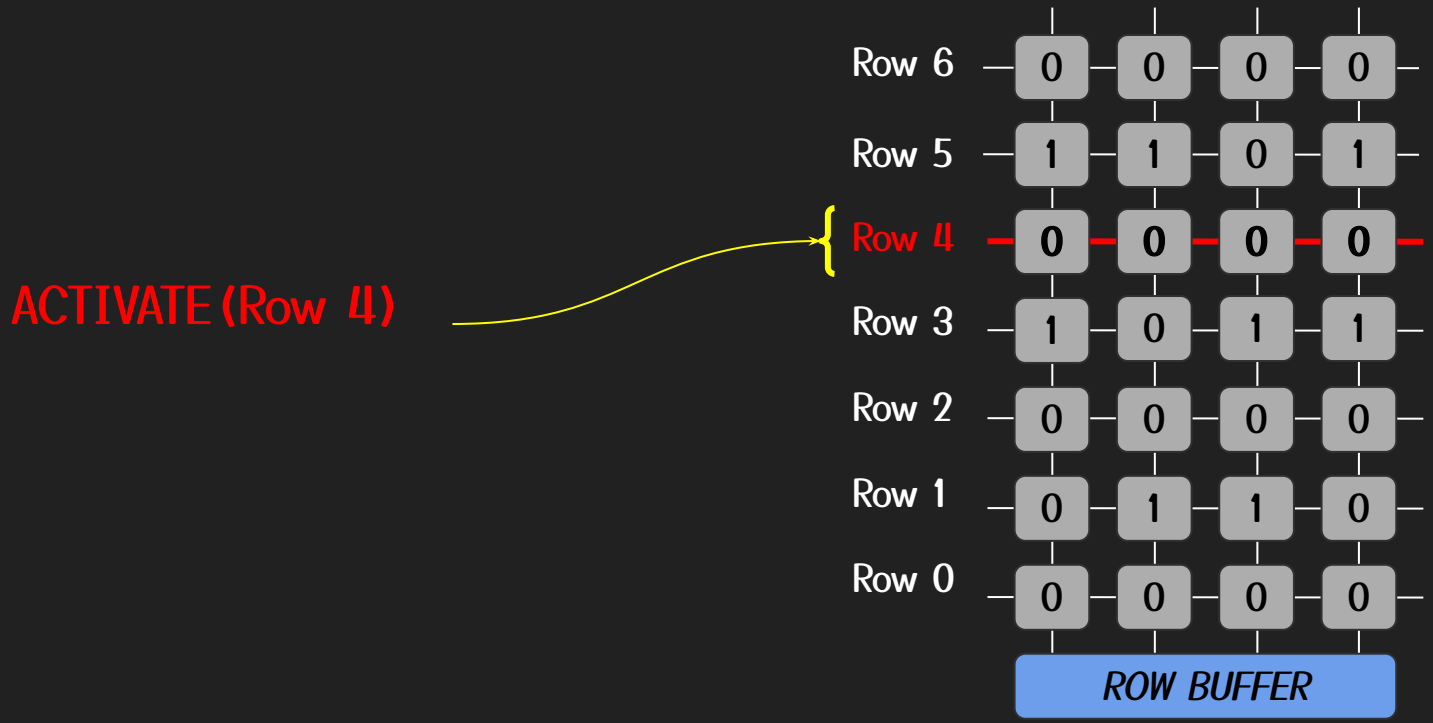
ACTIVATE (Row 0)



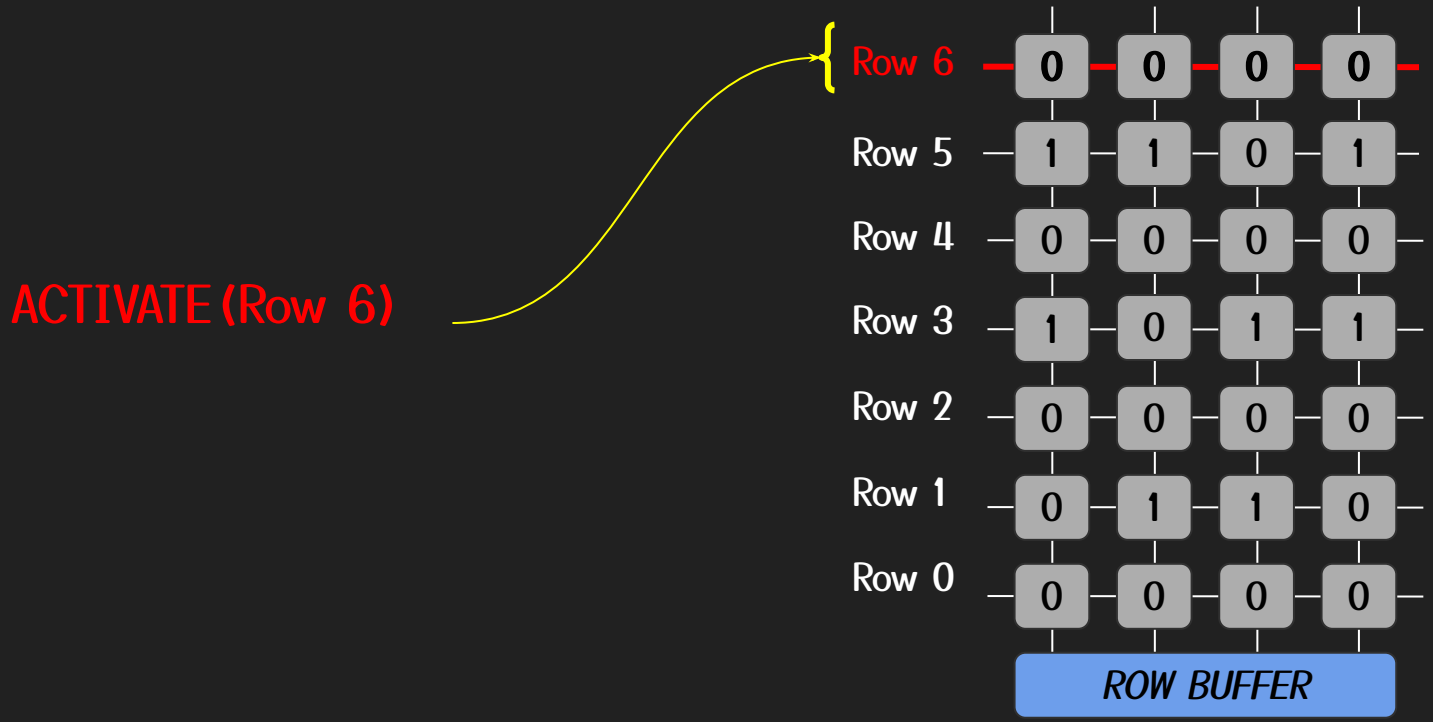
Rowhammer 101



Rowhammer 101

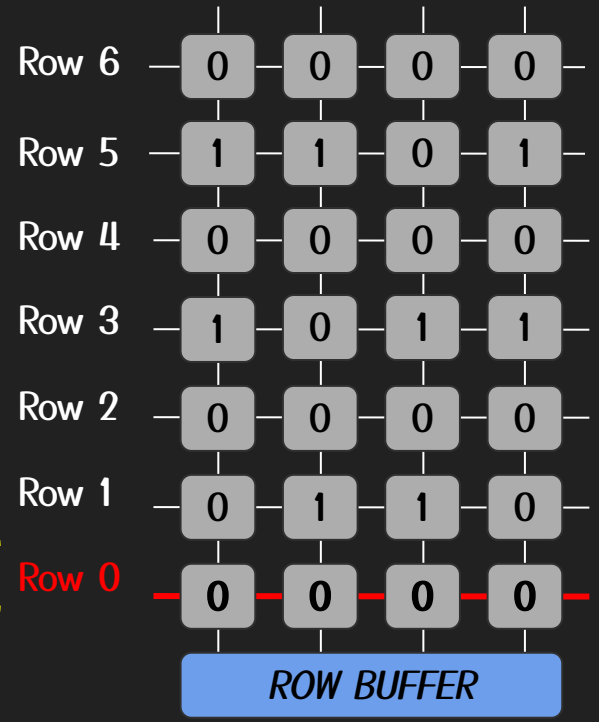


Rowhammer 101

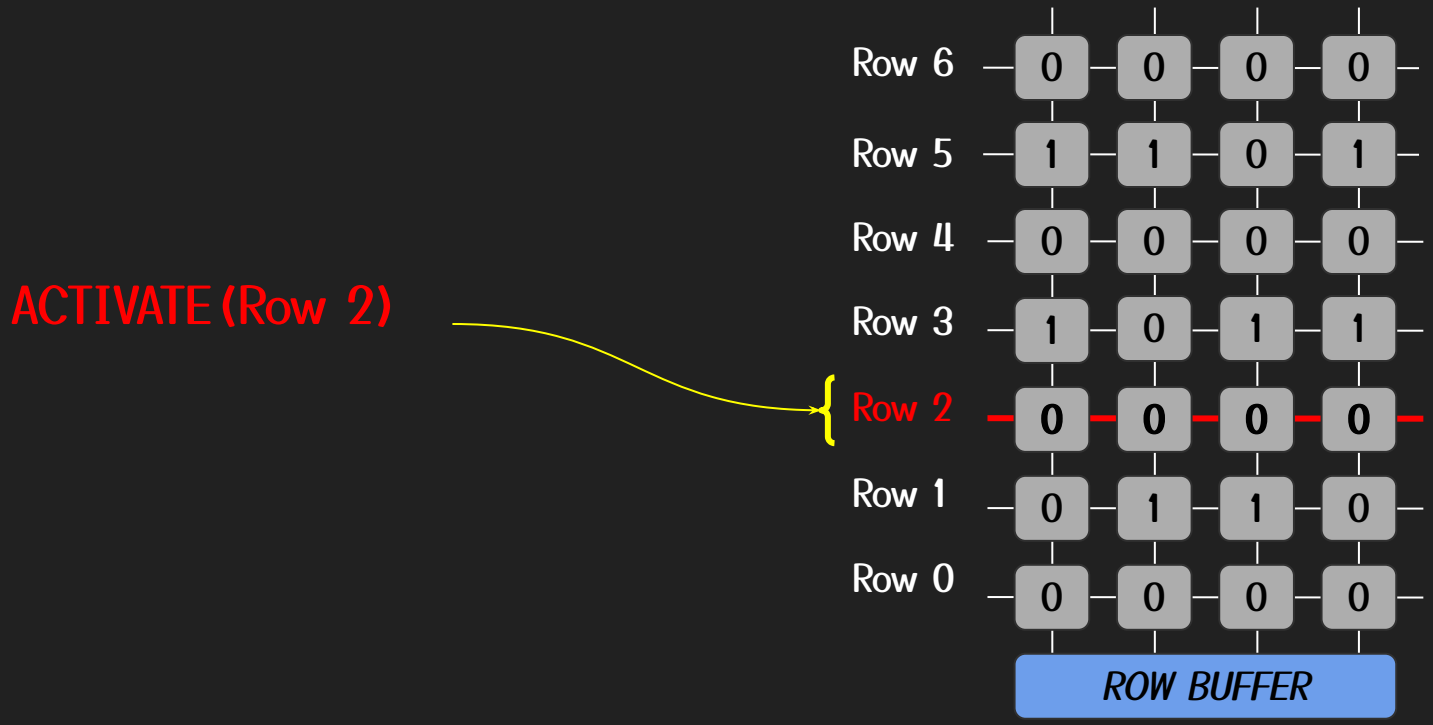


Rowhammer 101

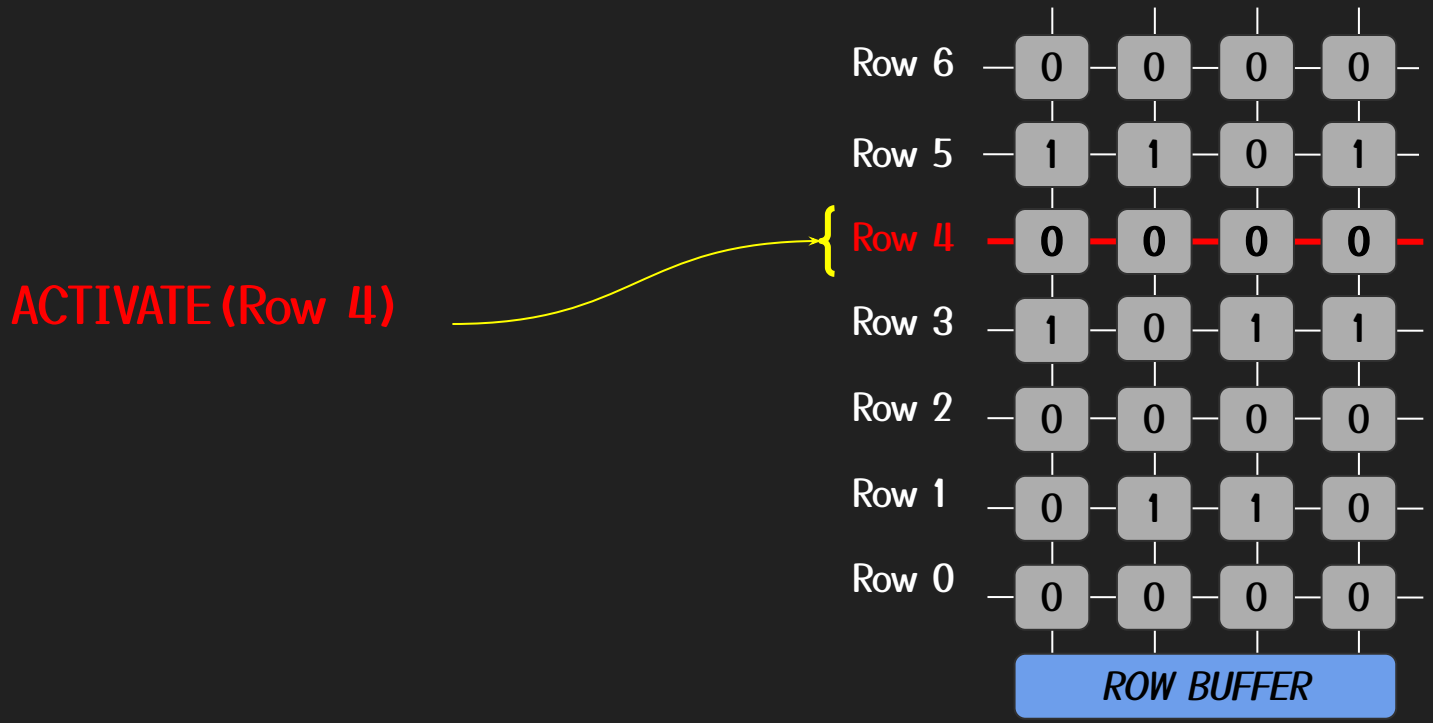
ACTIVATE (Row 0)



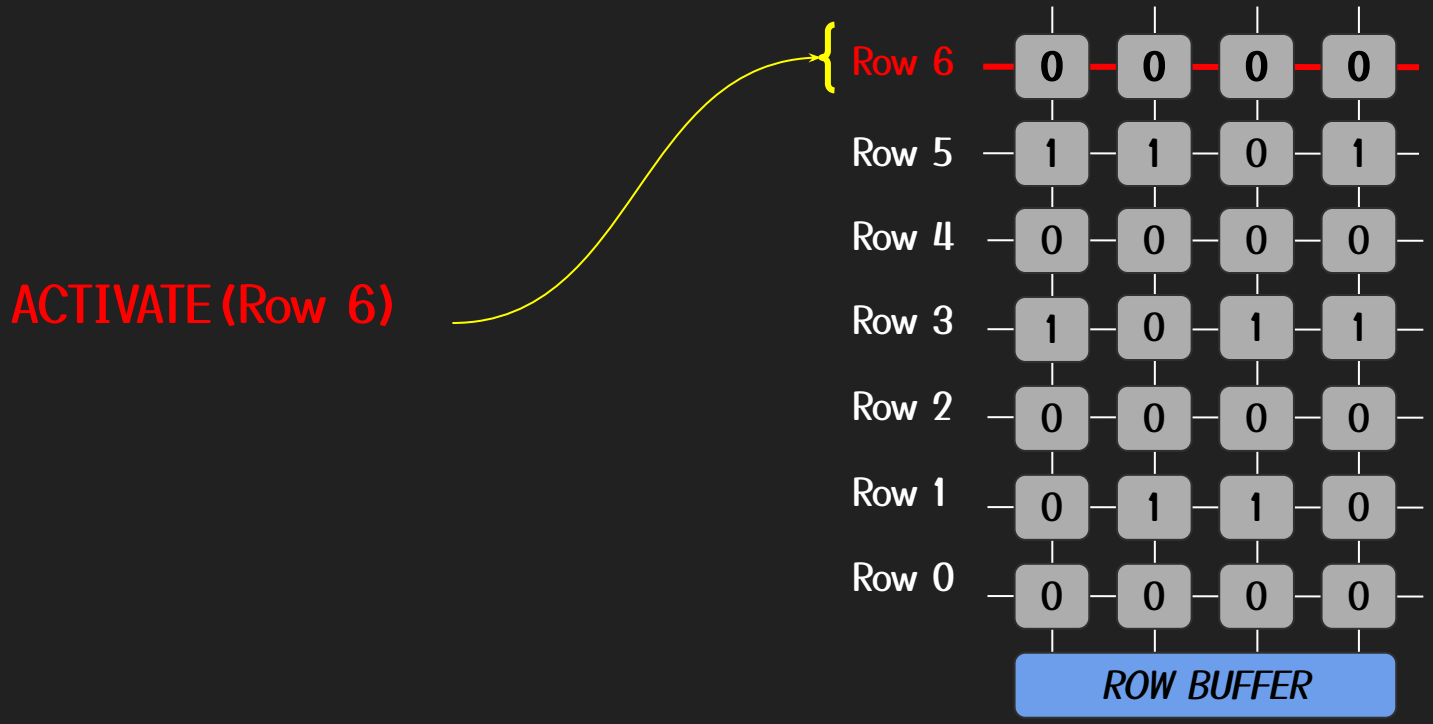
Rowhammer 101



Rowhammer 101

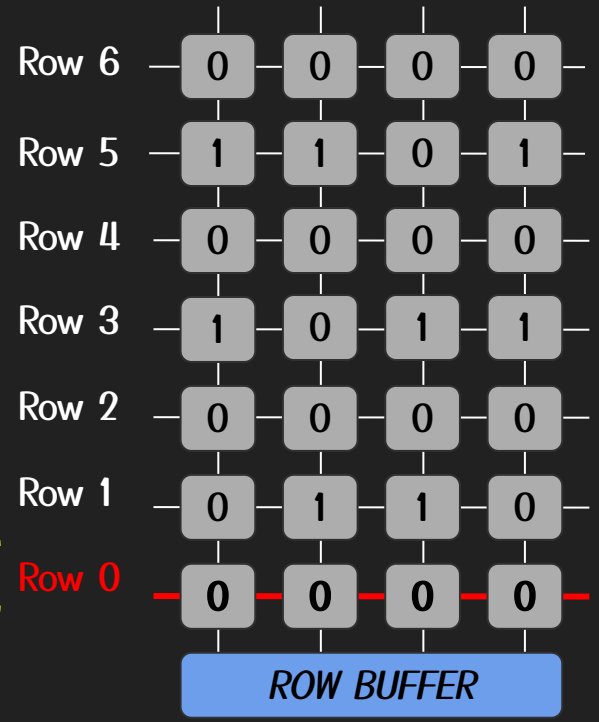


Rowhammer 101



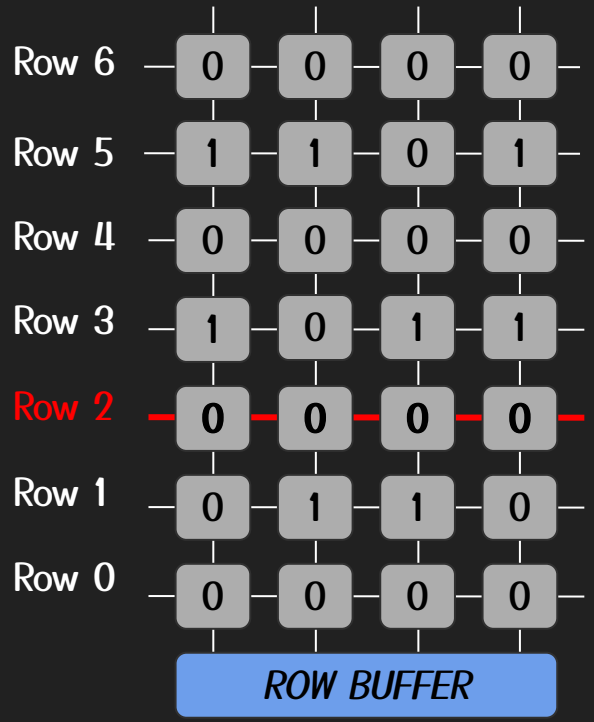
Rowhammer 101

ACTIVATE (Row 0)

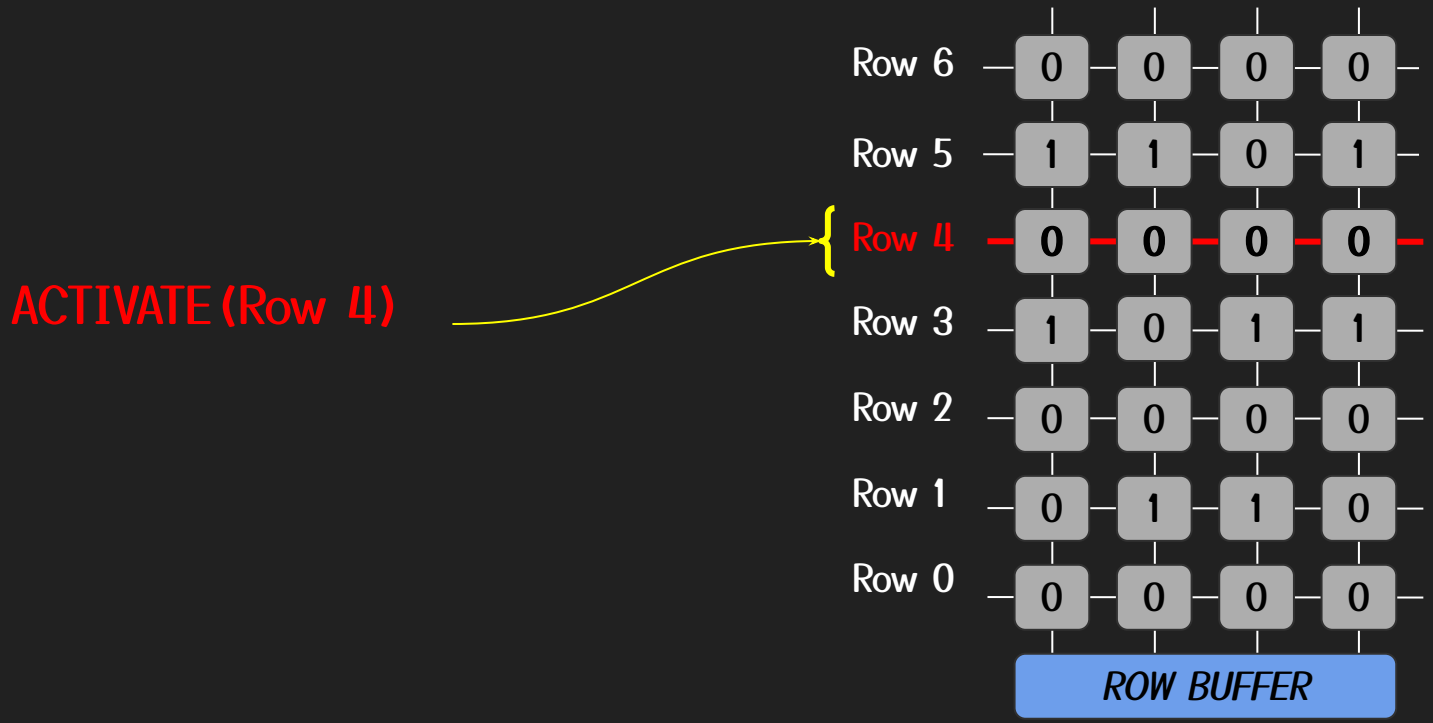


Rowhammer 101

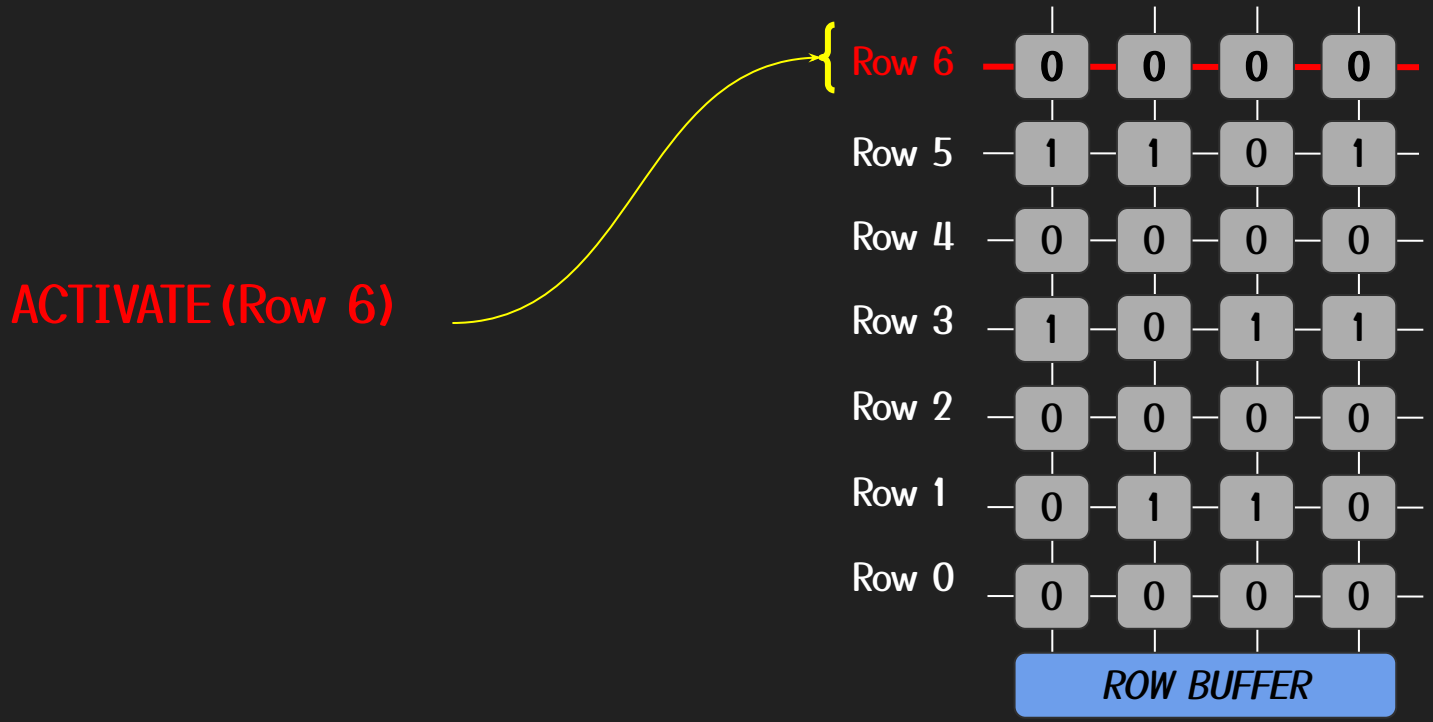
ACTIVATE (Row 2)



Rowhammer 101

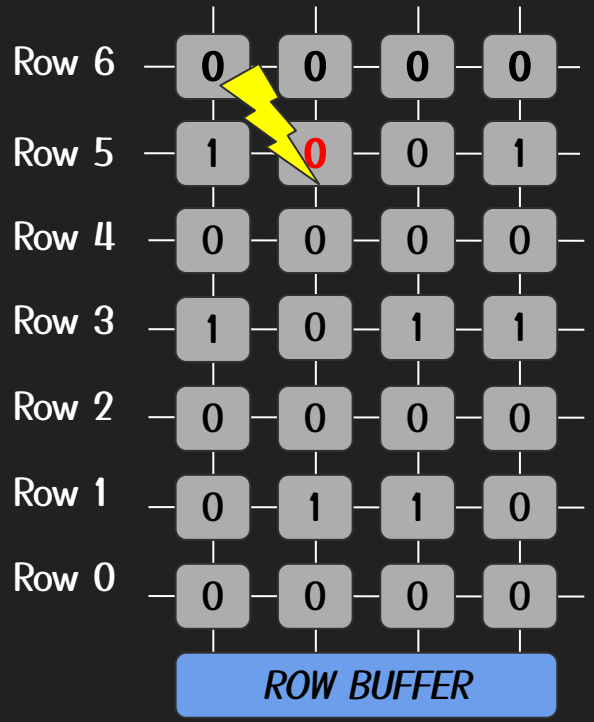


Rowhammer 101



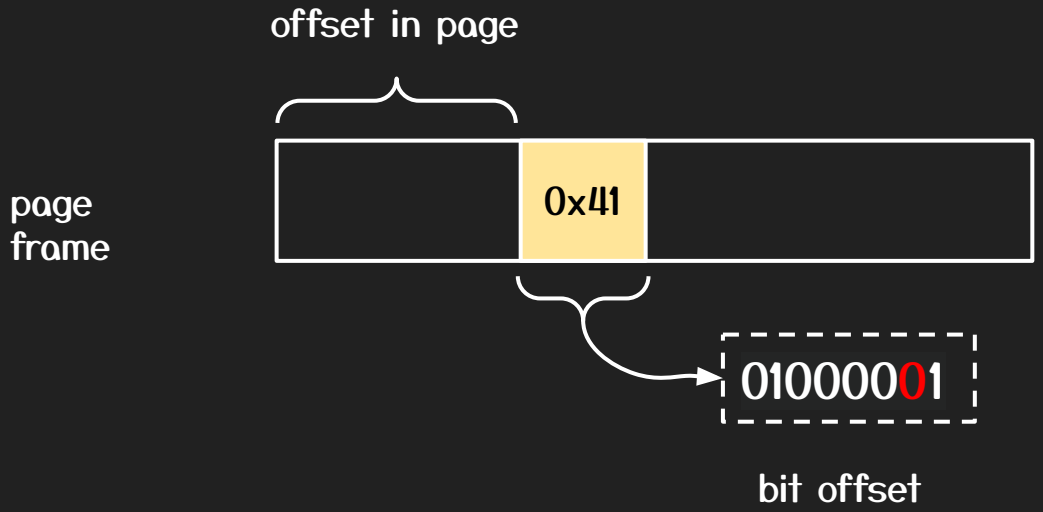
Rowhammer 101

Bit Flip in Row 5 (1 → 0)



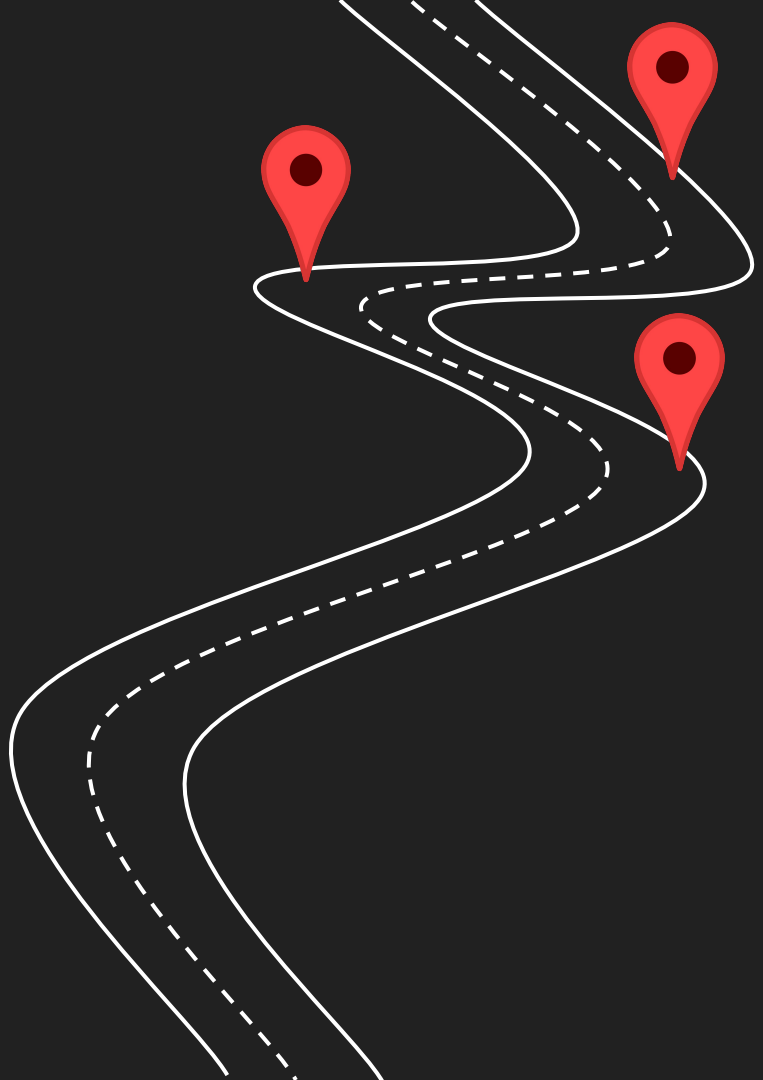
Templating

- Fingerprinting memory for ‘useful’ bit flips (a.k.a. templates)
- Template: combination of a vulnerable page and the offset of the bit flip



Roadmap

1. DRAM Timing Side Channel
2. Rowhammer & Templating
3. Opcode Flipping
4. Page Deduplication
5. Demo



x86 ASM

- x86 has a very ‘dense’ instruction set
- Flipping a bit in an opcode gives you another *valid* opcode with high probability

Toy Example

```
int check(char *passwd) {
    char *true_passwd = "secret";
    return strcmp(passwd, true_passwd) != 0;
}

int main(int argc, char **argv) {
    if(check(argv[1]) == 0) {
            printf("Success\n");
    }
    else {
            printf("Login Failed\n");
            return -1;
    }
    return 0;
}
```

Program Behaviour

```
$ ./login "secret"
$ Success

$ ./login "garbage"
$ Login Failed
```

Disassembly

```
0000000000001169 <check>:
    1169:       f3 0f 1e fa             endbr64
    116d:       55                      push   %rbp
    116e:       48 89 e5                mov    %rsp,%rbp
    1171:       48 83 ec 20             sub    $0x20,%rsp
    1175:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
    1179:       48 8d 05 84 0e 00 00    lea    0xe84(%rip),%rax
# 2004 <_IO_stdin_used+0x4>
    1180:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    1184:       48 8b 55 f8             mov    -0x8(%rbp),%rdx
    1188:       48 8b 45 e8             mov    -0x18(%rbp),%rax
    118c:       48 89 d6                mov    %rdx,%rsi
    118f:       48 89 c7                mov    %rax,%rdi
    1192:       e8 d9 fe ff ff          call   1070 <strcmp@plt>
    1197:       85 c0                   test   %eax,%eax
    1199:       75 07                   jne    11a2 <check+0x39>
    119b:       b8 00 00 00 00          mov    $0x0,%eax
    11a0:       eb 05                   jmp    11a7 <check+0x3e>
    11a2:       b8 01 00 00 00          mov    $0x1,%eax
    11a7:       c9                      leave
    11a8:       c3                      ret
```

Disassembly

```
0000000000001169 <check>:
    1169:       f3 0f 1e fa             endbr64
    116d:       55                      push   %rbp
    116e:       48 89 e5                mov    %rsp,%rbp
    1171:       48 83 ec 20             sub    $0x20,%rsp
    1175:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
    1179:       48 8d 05 84 0e 00 00    lea    0xe84(%rip),%rax
# 2004 <_IO_stdin_used+0x4>
    1180:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    1184:       48 8b 55 f8             mov    -0x8(%rbp),%rdx
    1188:       48 8b 45 e8             mov    -0x18(%rbp),%rax
    118c:       48 89 d6                mov    %rdx,%rsi
    118f:       48 89 c7                mov    %rax,%rdi
    1192:       e8 d9 fe ff ff          call   1070 <strcmp@plt>
    1197:       85 c0                   test   %eax,%eax
    1199:       75 07                   jne    11a2 <check+0x39>
    119b:       b8 00 00 00 00          mov    $0x0,%eax
    11a0:       eb 05                   jmp    11a7 <check+0x3e>
    11a2:       b8 01 00 00 00          mov    $0x1,%eax
    11a7:       c9                      leave
    11a8:       c3                      ret
```

Flipped Opcode

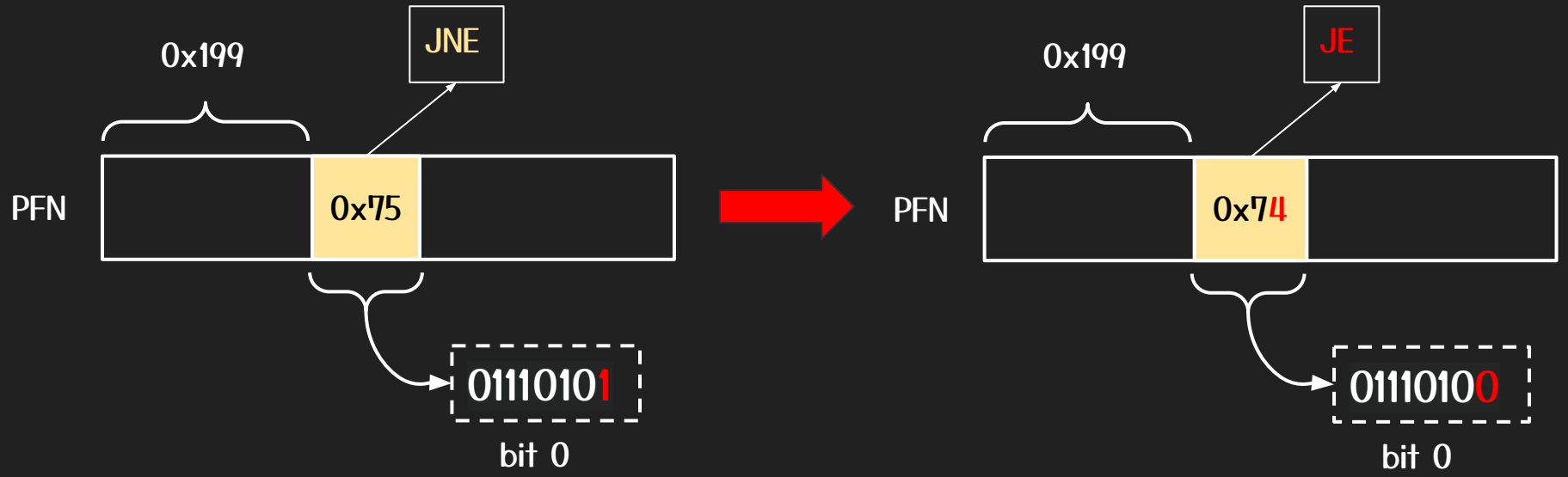
```
0000000000001169 <check>:
    1169:       f3 0f 1e fa             endbr64
    116d:       55                      push   %rbp
    116e:       48 89 e5                mov    %rsp,%rbp
    1171:       48 83 ec 20             sub    $0x20,%rsp
    1175:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
    1179:       48 8d 05 84 0e 00 00    lea    0xe84(%rip),%rax
# 2004 <_IO_stdin_used+0x4>
    1180:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    1184:       48 8b 55 f8             mov    -0x8(%rbp),%rdx
    1188:       48 8b 45 e8             mov    -0x18(%rbp),%rax
    118c:       48 89 d6                mov    %rdx,%rsi
    118f:       48 89 c7                mov    %rax,%rdi
    1192:       e8 d9 fe ff ff          call   1070 <strcmp@plt>
    1197:       85 c0                   test   %eax,%eax
    1199:       74 07                   je     11a2 <check+0x39>
    119b:       b8 00 00 00 00          mov    $0x0,%eax
    11a0:       eb 05                   jmp    11a7 <check+0x3e>
    11a2:       b8 01 00 00 00          mov    $0x1,%eax
    11a7:       c9                      leave
    11a8:       c3                      ret
```

New Behaviour

```
$ ./login "secret"
$ Login Failed

$ ./login "garbage"
$ Success
```

Template

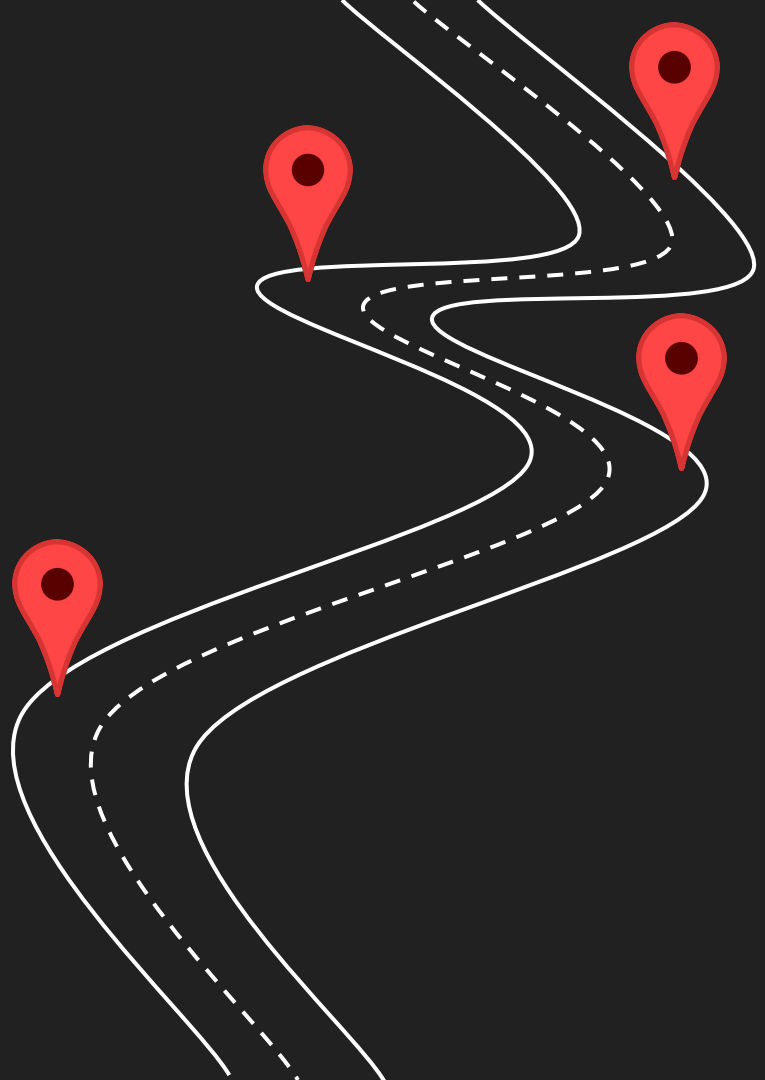

0x199
JNE
PFN
0x75
01110101
bit 0
0x199
JE
PFN
0x74
01110100
bit 0

OK, Let’s Get Real

- Good real world targets: setuid binaries
- sudoers.so library mapped in by e.g., sudo
- Binary analysis to discover ‘interesting’ opcodes
- Invert program behaviour for binaries using sudoers.so

Roadmap

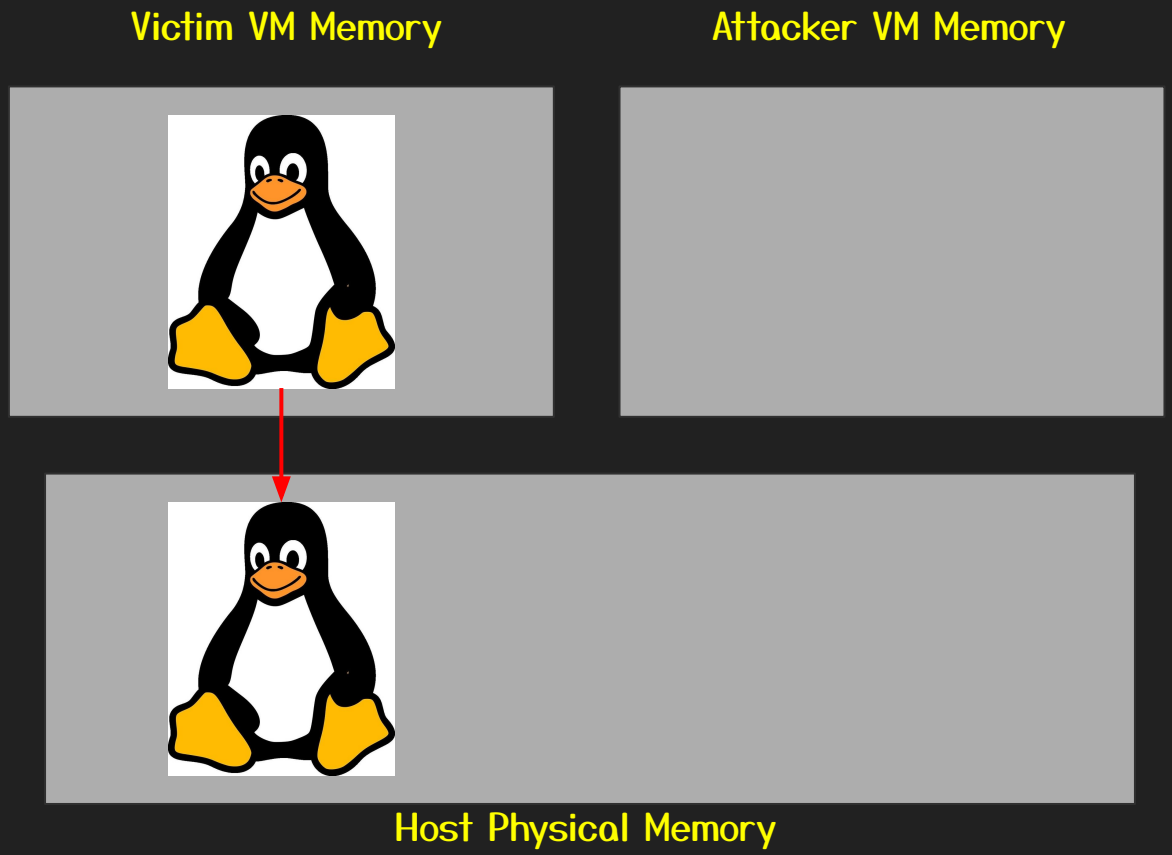
1. DRAM Timing Side Channel
2. Rowhammer & Templating
3. Opcode Flipping
4. Page Deduplication
5. Demo



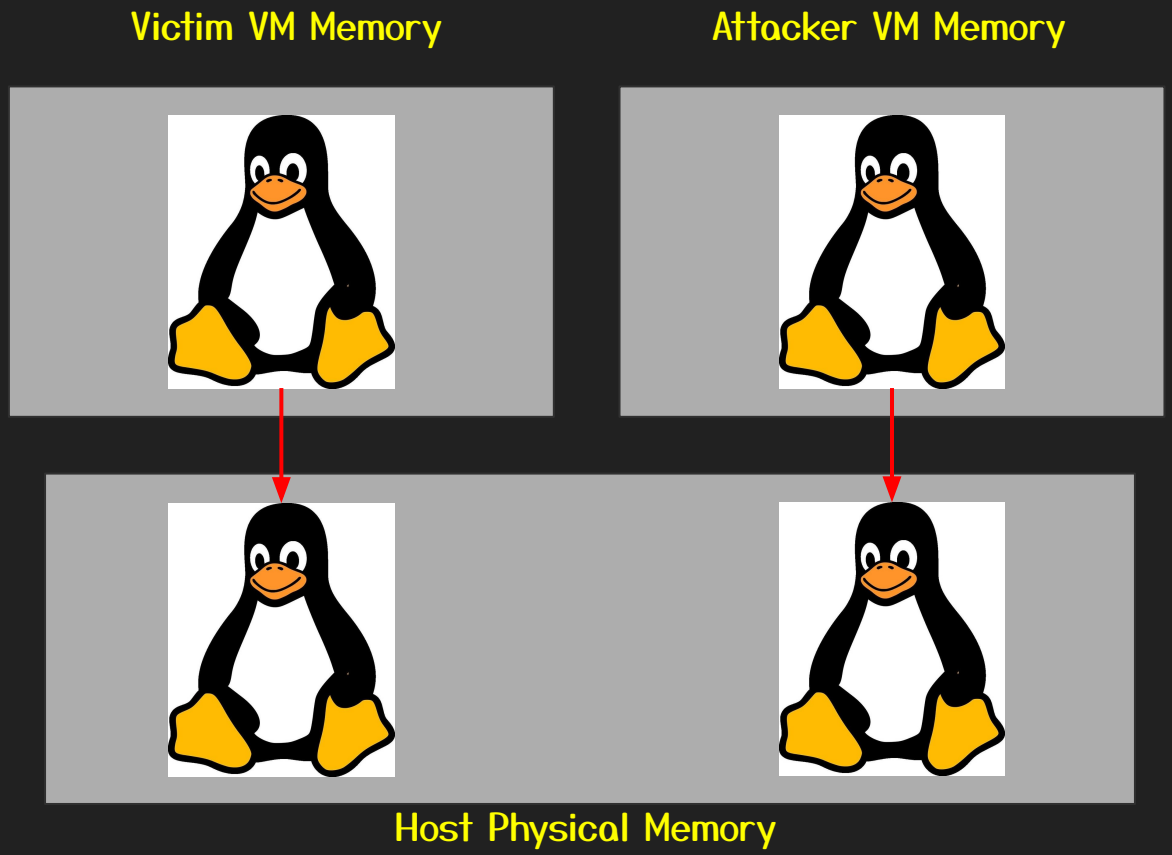
Page Deduplication

- Available on most modern OSes to improve performance
- On Linux known as Kernel Samepage Merging (KSM)
- Pages with the same content get merged to avoid multiple copies of the same data
- Mapped as read-only with COW semantics

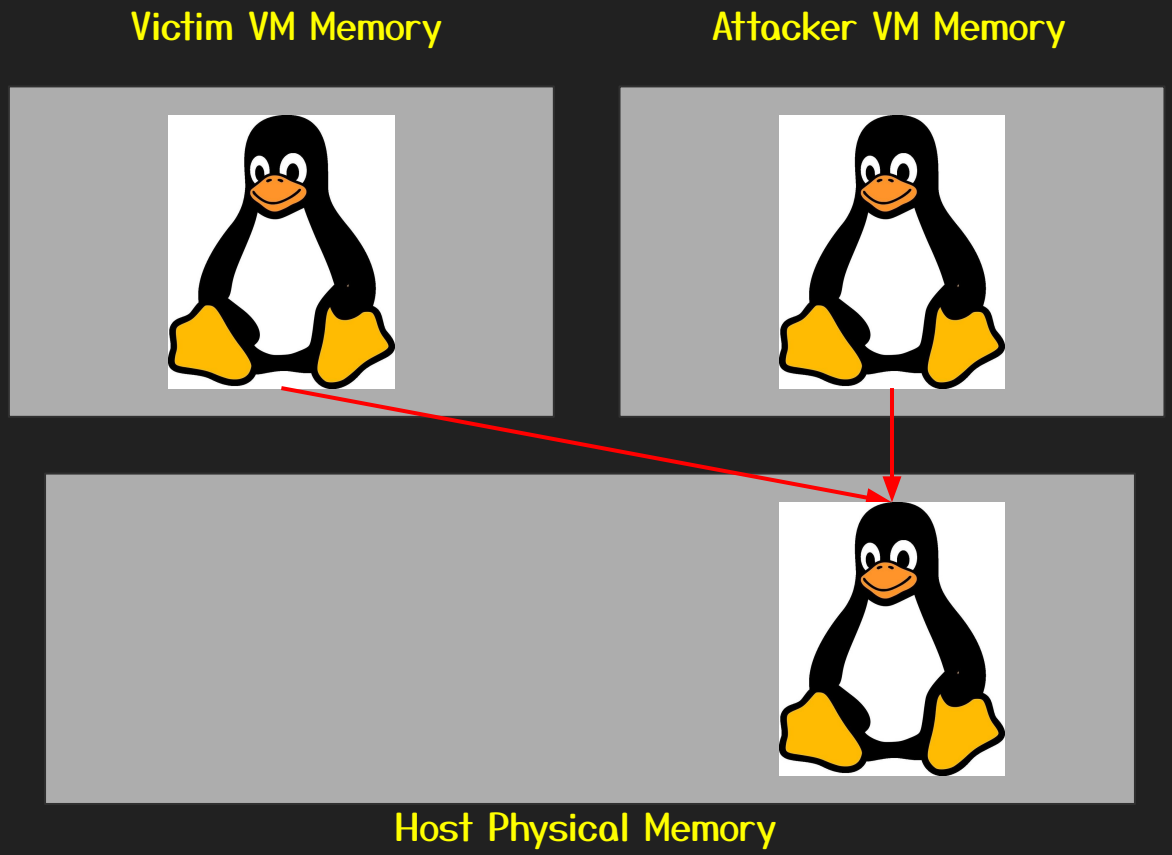
KSM on the Host



KSM on the Host



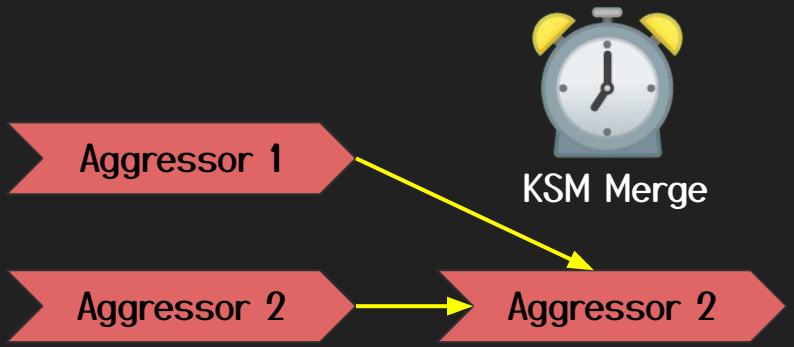
KSM on the Host



Kernel Daemons

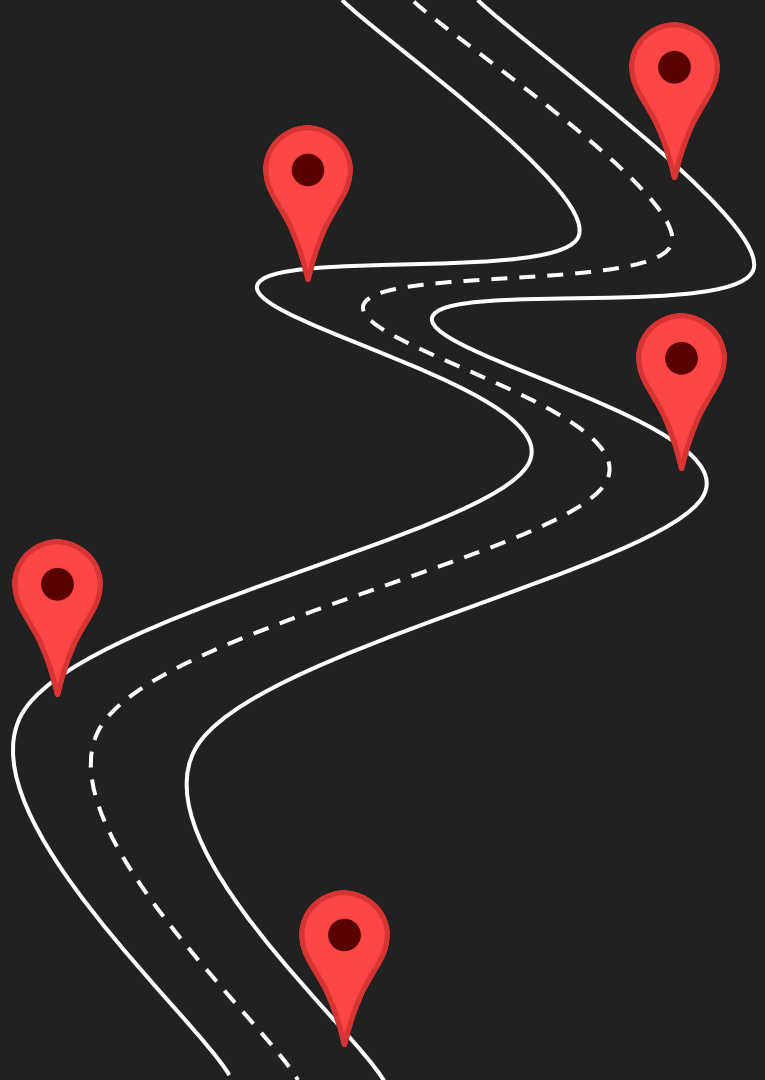
- ksmd splits up huge pages if they are made up of pages with the same data
- This will merge the aggressor rows, nullifying the exploit

Defeating the Daemons

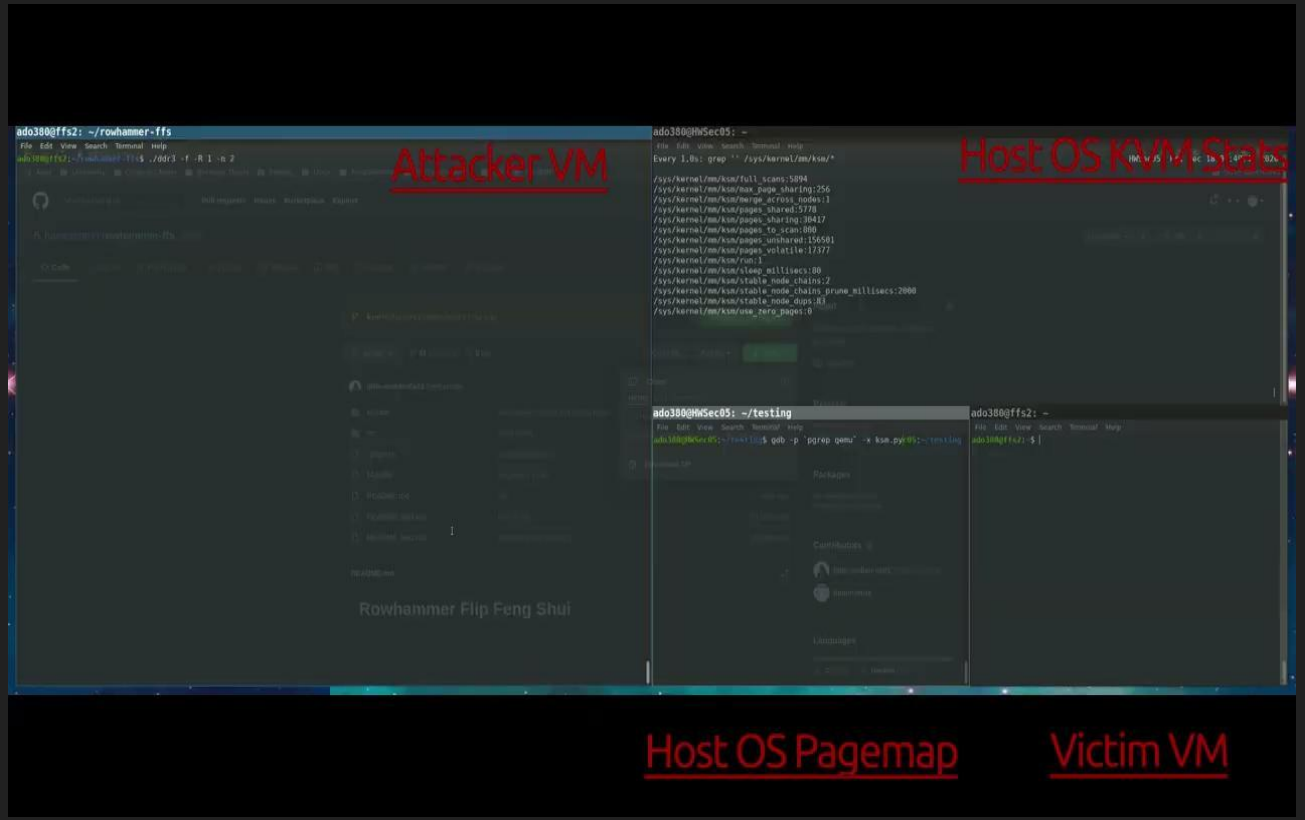


Roadmap

1. DRAM Timing Side Channel
2. Rowhammer & Templating
3. Opcode Flipping
4. Page Deduplication
5. Demo



Demo Video



Thank You for Listening 😊