Presshammer

Rowhammer and Rowpress without Physical Address Information

Jonas Juffinger **Sudheendra Raghav Neela** **Martin Heckel** **Lukas Schwarz**
Florian Adamsky **Daniel Gruss**

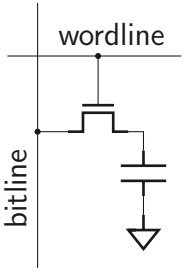
2024-07-19

IAIK, Graz University of Technology, Austria & Hof University of Applied Sciences, Germany

DIMVA 2024 – EPFL, Lausanne, Switzerland

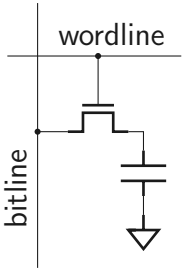
Introduction

Dynamic Random Access Memory (DRAM)

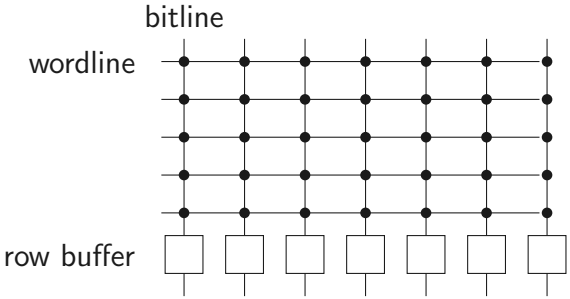


DRAM **Cell**

Dynamic Random Access Memory (DRAM)

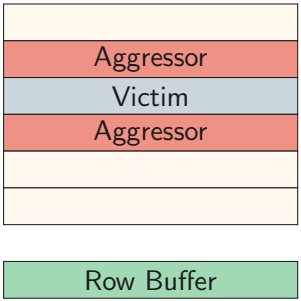


DRAM **Cell**



DRAM **Bank**

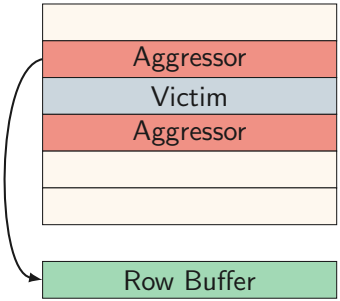
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
for (i = 0; i < N; ++i) {
  *aggressor1;
  *aggressor2;
  flush(aggressor1);
  flush(aggressor2);
}
```

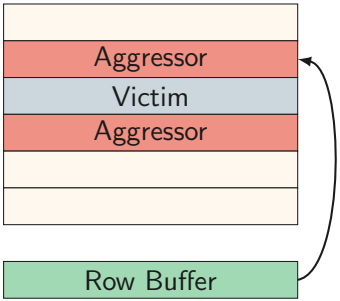
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

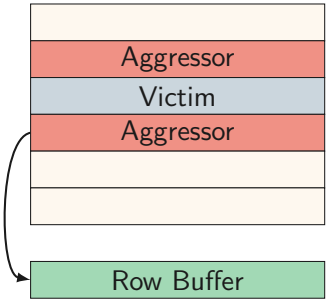
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

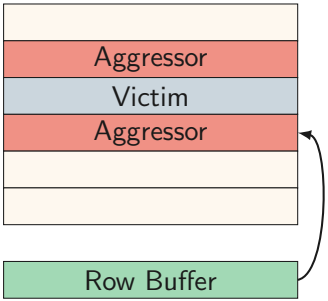
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
for (i = 0; i < N; ++i) {
  *aggressor1;
  *aggressor2;
  flush(aggressor1);
  flush(aggressor2);
}
```

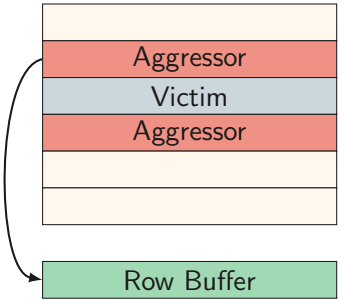
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

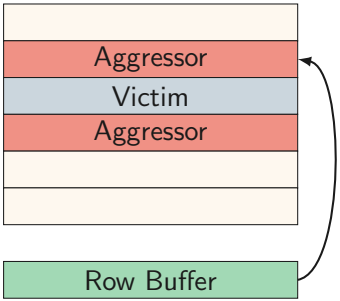
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

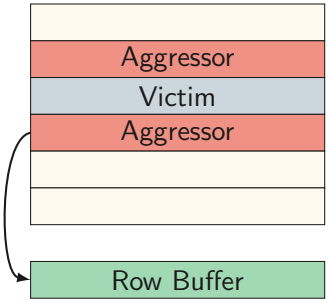
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

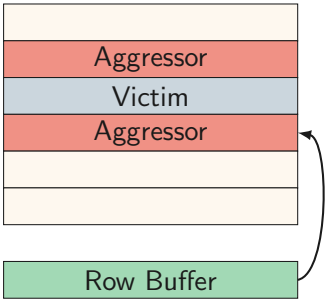
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
for (i = 0; i < N; ++i) {
  *aggressor1;
  *aggressor2;
  flush(aggressor1);
  flush(aggressor2);
}
```

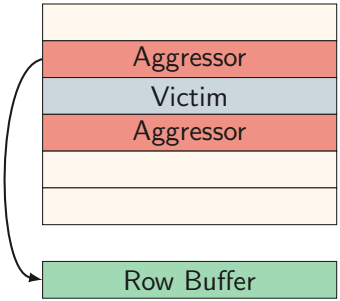
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

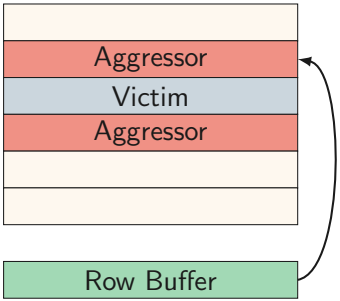
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

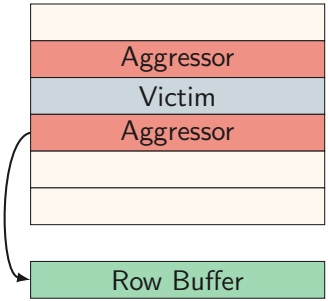
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

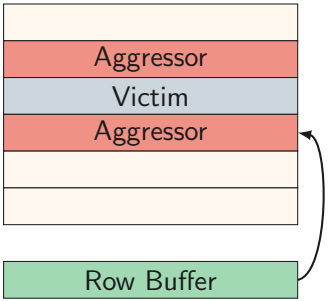
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
for (i = 0; i < N; ++i) {
  *aggressor1;
  *aggressor2;
  flush(aggressor1);
  flush(aggressor2);
}
```

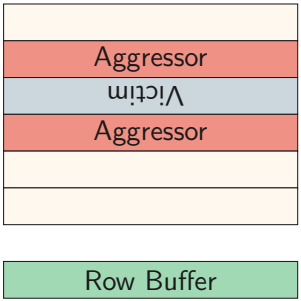
Rowhammer [2]



Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

Rowhammer [2]

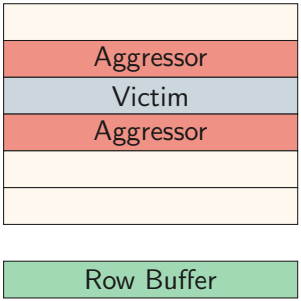


Repeatedly and quickly active multiple rows.

```
1 for (i = 0; i < N; ++i) {
2   *aggressor1;
3   *aggressor2;
4   flush(aggressor1);
5   flush(aggressor2);
6 }
```

RowPress [3]

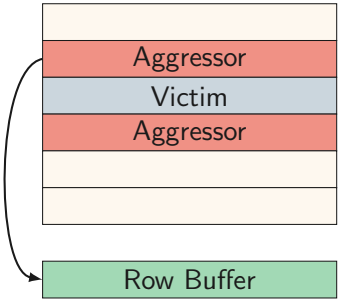
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

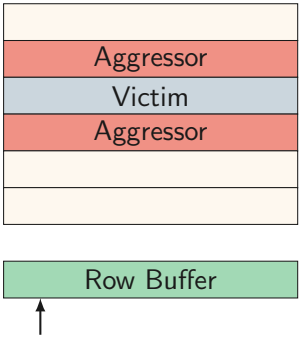
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

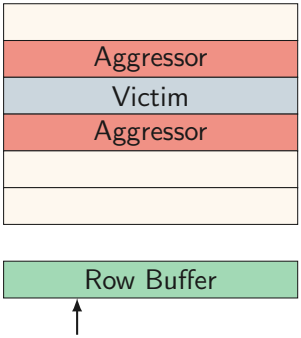
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

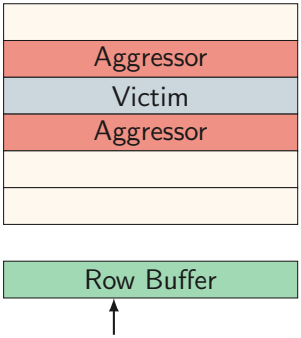
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

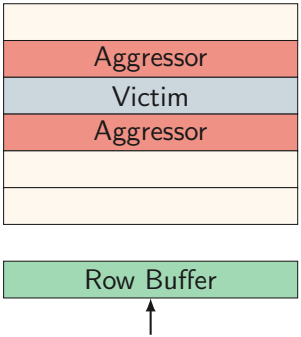
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

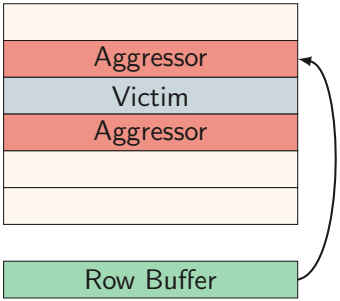
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

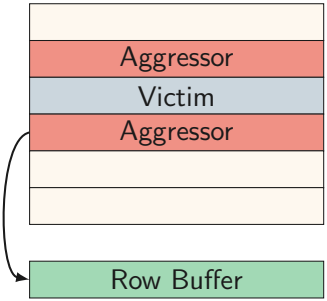
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

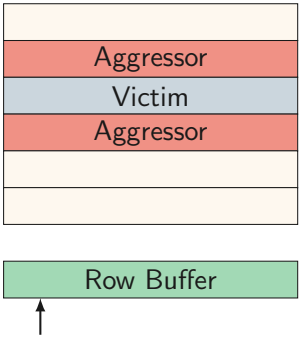
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

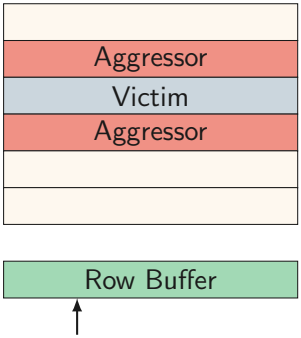
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

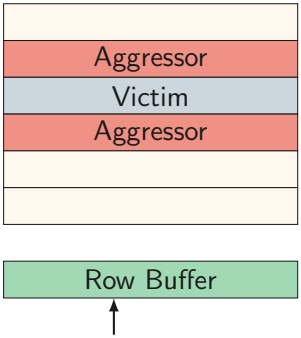
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

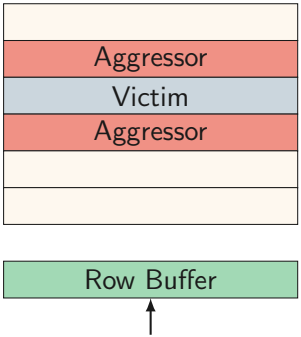
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

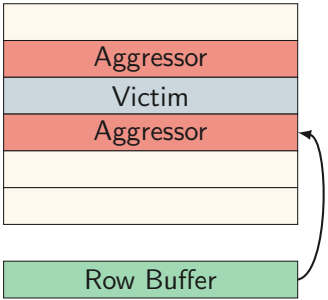
Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

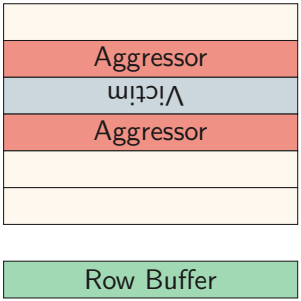
RowPress [3]

Keep rows open as long as possible.




```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

RowPress [3]

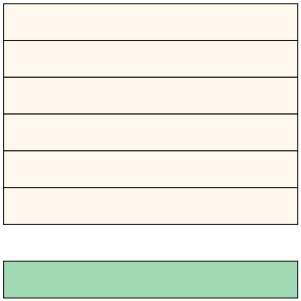


Keep rows open as long as possible.

Flips different bits than Rowhammer

```
1  for (i = 0; i < N; ++i) {
2    for (j = 0; j < 16; ++j) {
3      *aggressor1[j];
4      flush(aggressor1[j]);
5    }
6    for (j = 0; j < 16; ++j) {
7      *aggressor2[j];
8      flush(aggressor2[j]);
9    }
10 }
```

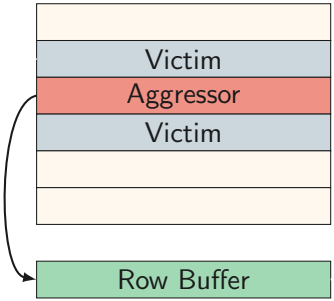
Single-Sided RowPress



Keep one row open as long as possible.

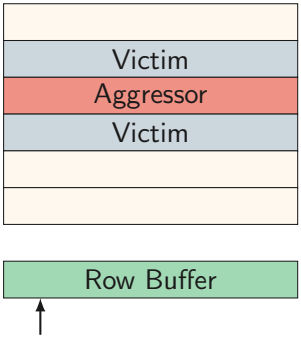
Jonas Juffinger – @NotImaginary_ www.jonasjuffinger.com — IAIK, Graz University of Technology

Single-Sided RowPress



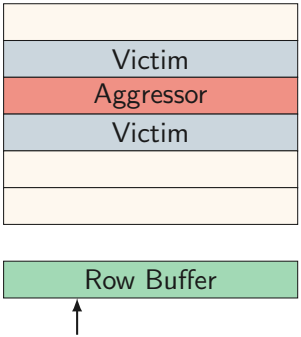
Keep one row open as long as possible.

Single-Sided RowPress



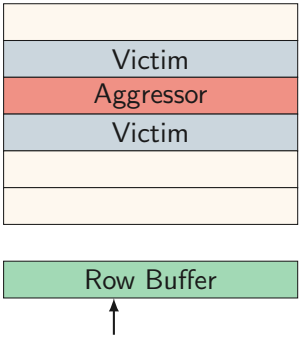
Keep one row open as long as possible.

Single-Sided RowPress



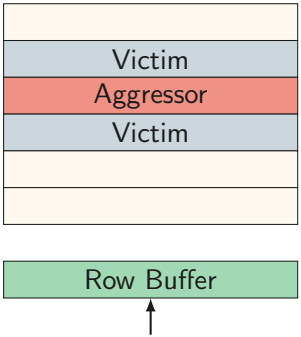
Keep one row open as long as possible.

Single-Sided RowPress



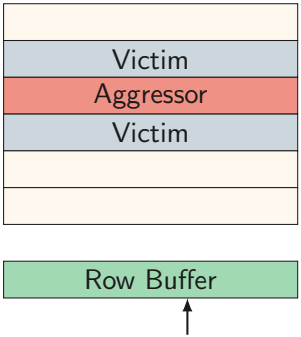
Keep one row open as long as possible.

Single-Sided RowPress



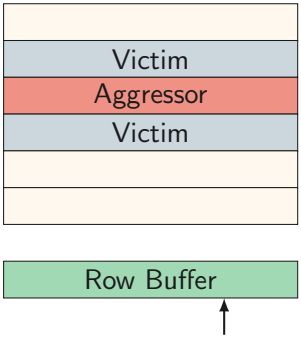
Keep one row open as long as possible.

Single-Sided RowPress



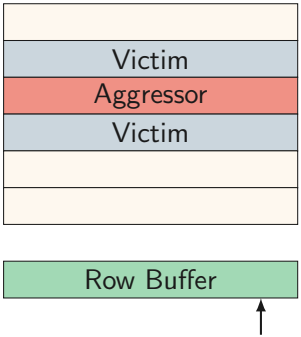
Keep one row open as long as possible.

Single-Sided RowPress



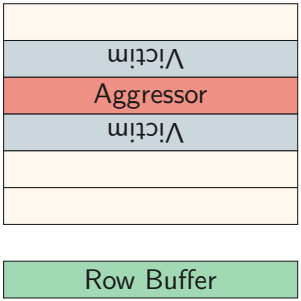
Keep one row open as long as possible.

Single-Sided RowPress



Keep one row open as long as possible.

Single-Sided RowPress



Keep one row open as long as possible.

Another Flip...

But wait...

Another Flip...

There is this paper from 2018.

Another Flip in the Wall of Rowhammer Defenses

Daniel Gruss[1], Moritz Lipp[1], Michael Schwarz[1], Daniel Genkin[2],
Jonas Juffinger[1], Sioli O'Connell[3], Wolfgang Schoechl[1], and Yuval Yarom[3,4]
[1] Graz University of Technology
[2] University of Pennsylvania and University of Maryland
[3] University of Adelaide
[4] Data61

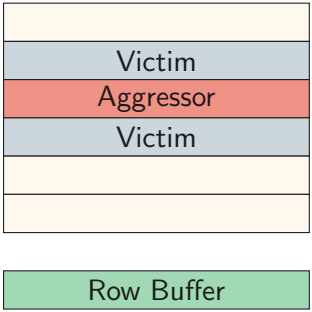
***Abstract*—The Rowhammer bug allows unauthorized modification of bits in DRAM cells from unprivileged software, enabling powerful privilege-escalation attacks. Sophisticated Rowhammer countermeasures have been presented, aiming at mitigating the Rowhammer bug or its exploitation. However, the state of the art provides insufficient insight on the completeness of these defenses.**

In this paper, we present novel Rowhammer attack and exploitation primitives, showing that even a combination of all defenses is ineffective. Our new attack technique, *one-location hammering*, breaks previous assumptions on requirements for triggering the Rowhammer bug, i.e., we do not hammer multiple DRAM rows but only keep one DRAM row constantly open. Our new exploitation technique, *opcode flipping*, bypasses recent isolation mechanisms by flipping bits in a predictable and targeted way in userspace binaries. We replace conspicuous

Software-based mitigations, which can be implemented on commodity systems, have also been proposed. These include ad-hoc defense techniques such as doubling the RAM refresh rates [44], removing unprivileged access to the `pagemap` interface [45, 62, 65], and prohibiting the `clflush` instruction [65]. However, recent works have already bypassed these countermeasures [6, 24, 68]. Other ad-hoc attempts, such as disabling page deduplication by default [52, 60], only prevent specific Rowhammer attacks exploiting these features [11, 59], but not all Rowhammer attacks.

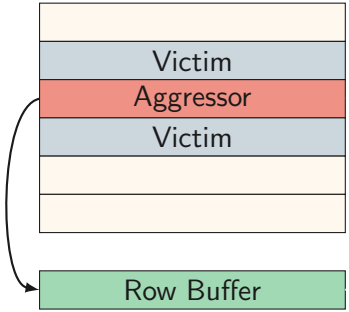
The research community proposed sophisticated defenses which seemingly have solved the Rowhammer problem. Based on the underlying primitives of these defenses, we introduce

One-Location Rowhammer [1]



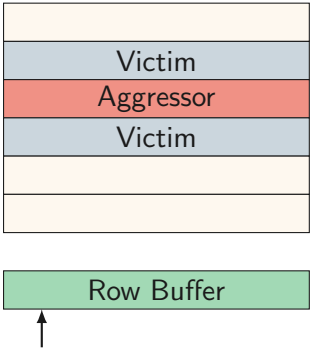
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



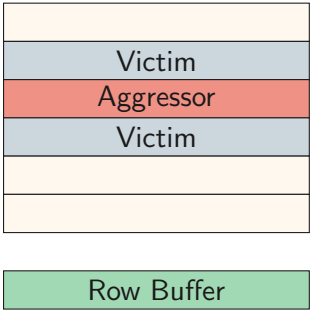
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



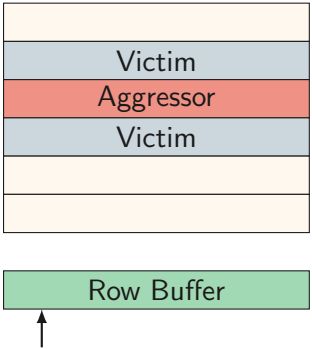
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



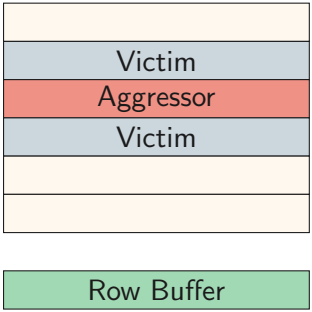
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



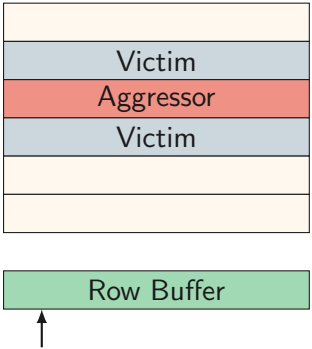
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



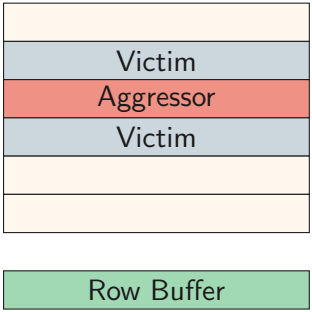
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



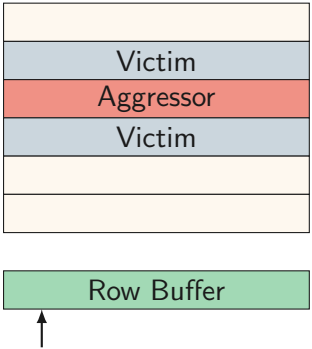
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



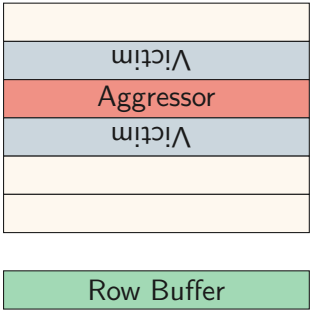
Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



Repeatedly and quickly activate one row.

One-Location Rowhammer [1]



Repeatedly and quickly activate one row.

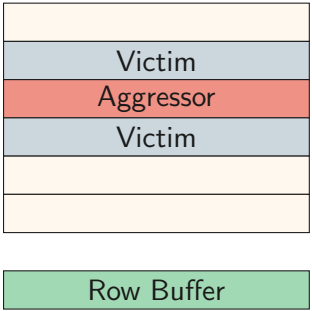

LOOKS A LOT LIKE ROWPRESS
imgflip.com

One-Location Rowhammer - Hypothesis

One-Location Rowhammer - Hypothesis

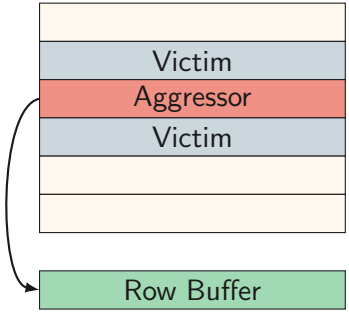
It's the memory controller.

One-Location Rowhammer - Hypothesis



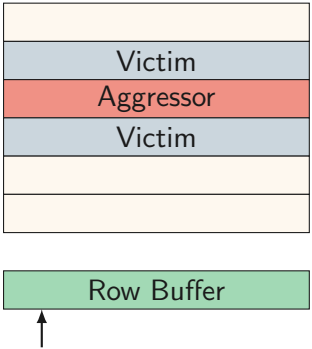
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



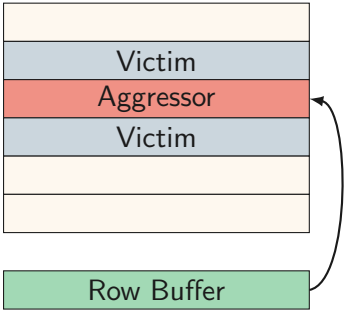
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



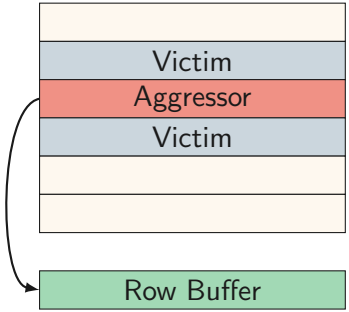
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



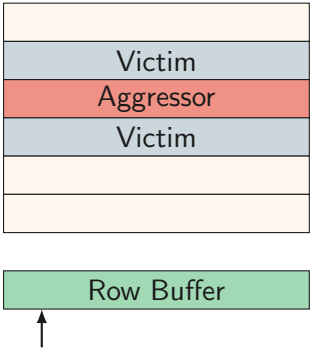
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



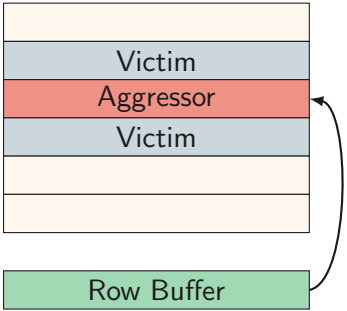
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



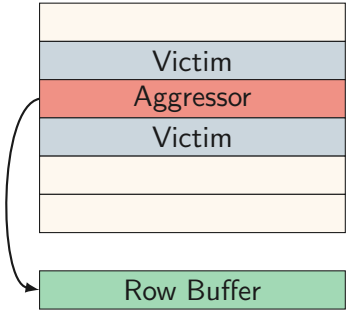
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



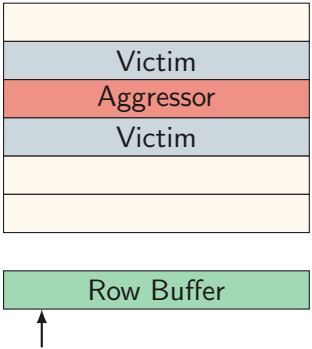
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



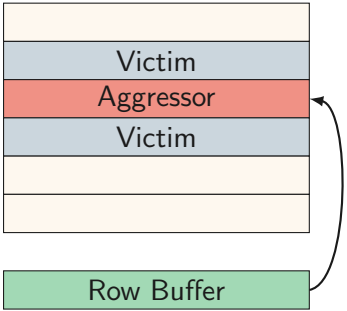
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



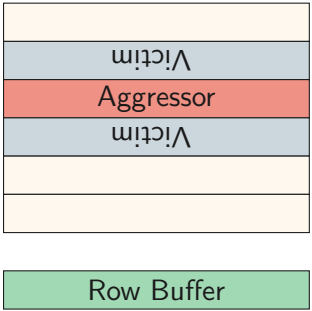
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



Memory controller employs a *closed-row* policy.

Case closed ✓

Is it?

Is Rowpress = OL Rowhammer?

Is Rowpress = OL Rowhammer?

or where Gruss et al. wrong all along?

Experiment

Experiment

RH and RP are caused by different physical effects

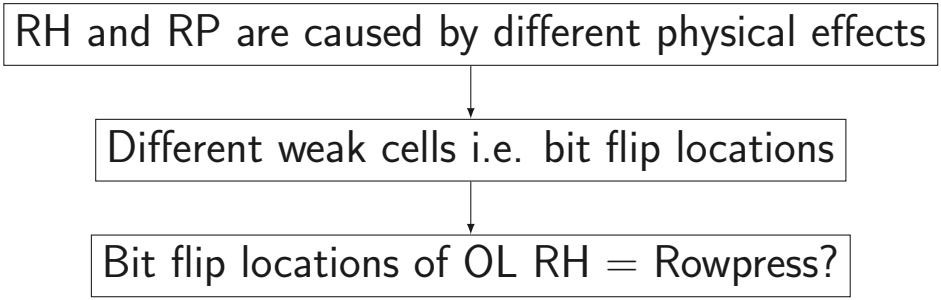
Experiment

RH and RP are caused by different physical effects

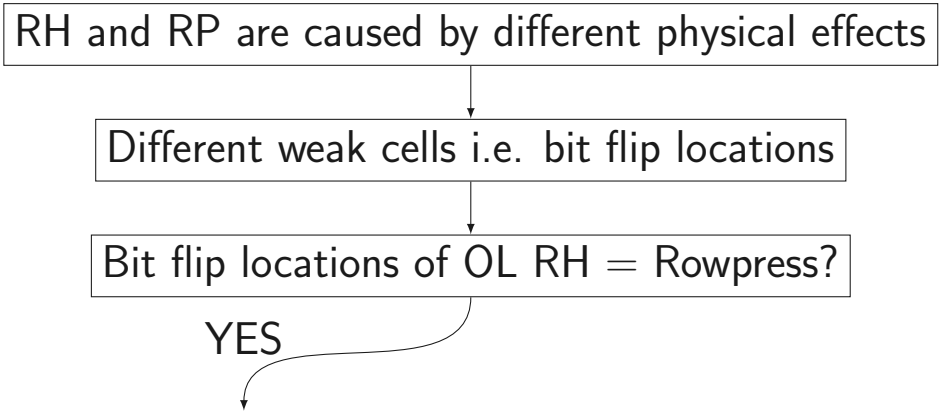
↓

Different weak cells i.e. bit flip locations

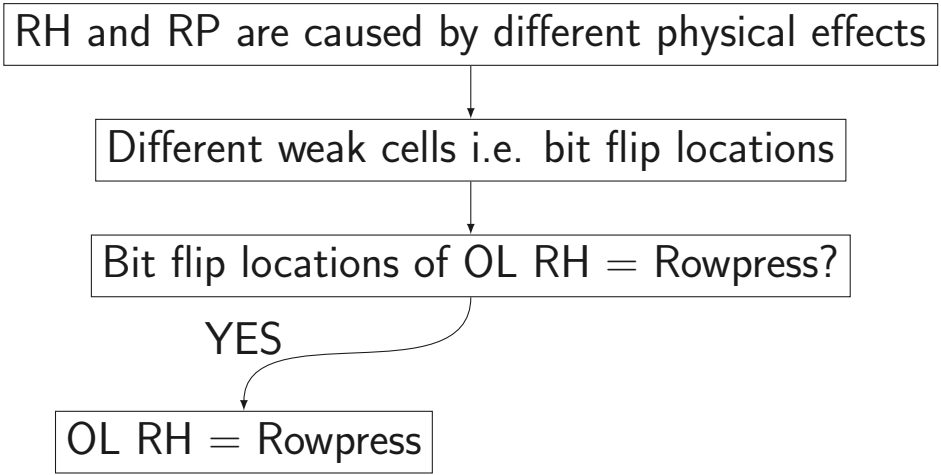
Experiment


RH and RP are caused by different physical effects
Different weak cells i.e. bit flip locations
Bit flip locations of OL RH = Rowpress?

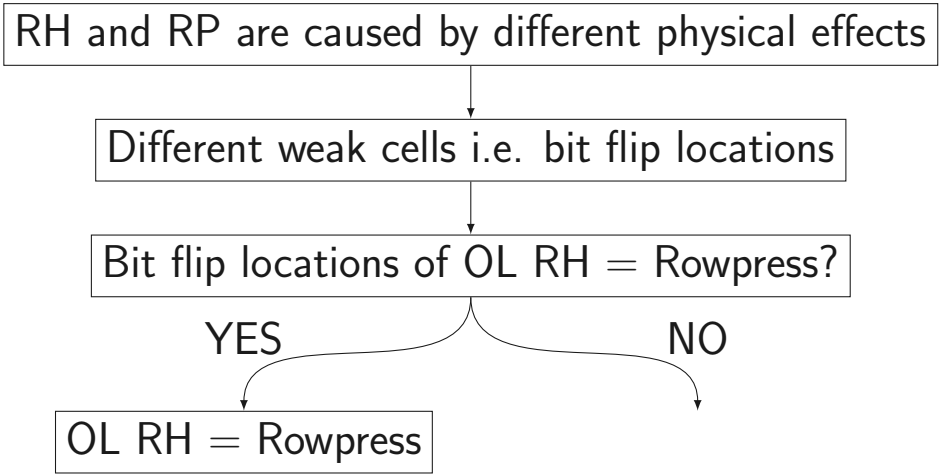
Experiment


RH and RP are caused by different physical effects
Different weak cells i.e. bit flip locations
Bit flip locations of OL RH = Rowpress?
YES

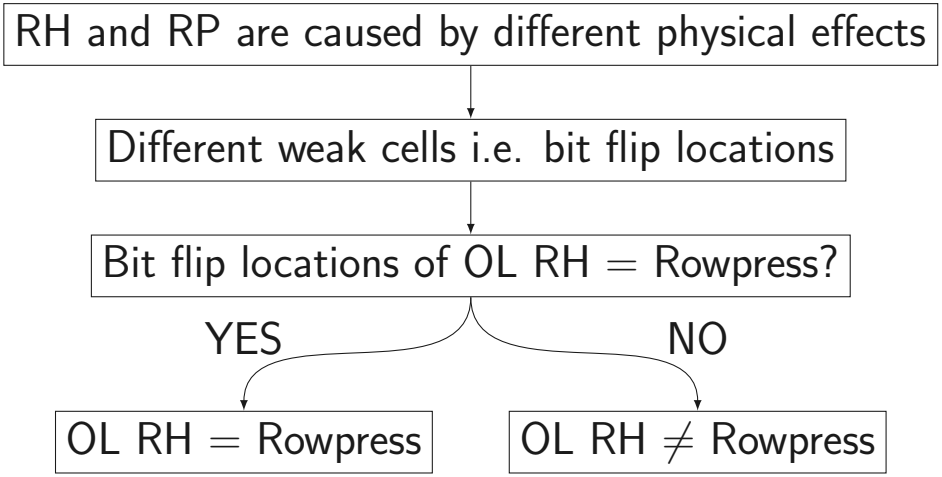
Experiment


RH and RP are caused by different physical effects
Different weak cells i.e. bit flip locations
Bit flip locations of OL RH = Rowpress?
YES
OL RH = Rowpress

Experiment


RH and RP are caused by different physical effects
Different weak cells i.e. bit flip locations
Bit flip locations of OL RH = Rowpress?
YES
NO
OL RH = Rowpress

Experiment


RH and RP are caused by different physical effects
Different weak cells i.e. bit flip locations
Bit flip locations of OL RH = Rowpress?
YES
NO
OL RH = Rowpress
OL RH ≠ Rowpress

The Matrix

		RH		RP	
		DS	OL	DS	OL
RH	DS				
	OL				
RP	DS				
	OL				

Results

	DIMM	Size	MT/s	Double Sided RH*	Double Sided RP	One Location RH	One Location RP
DDR4	D1	8 GB	2133	✓	✓	✓	✓
	D2	8 GB	2133	✓	✓	✓	✓
	D3	8 GB	2400	✓	✗	∼	∼
	D4	8 GB	2400	✓	✗	∼	∼
	D5	8 GB	2133	✗	✗	∼	∼
	D6	8 GB	2666	✓	✗	∼	∼
	D7	8 GB	2666	✗	✗	∼	∼
	D8	8 GB	2666	✗	✗	∼	∼
	D9	8 GB	2666	✓	✗	∼	∼
	D10	8 GB	2666	✗	✗	∼	∼
	D11	8 GB	3000	✗	✗	∼	∼
	D12	8 GB	2133	✗	✗	∼	∼

Inconclusive...

		RH		RP	
		DS	OL	DS	OL
RH	DS	100	56.5	13.7	1.0
	OL	61.8	100	14.5	1.1
RP	DS	98.1	95.3	100	1.0
	OL	96.3	98.8	81.4	100

PressHammer Exploit

PressHammer Exploit

Exploit Steps

- Find complete DRAM row

PressHammer Exploit

Exploit Steps

- Find complete DRAM row
- Template memory for bit flips

PressHammer Exploit

Exploit Steps

- Find complete DRAM row
- Template memory for bit flips
- Spray page tables

PressHammer Exploit

Exploit Steps

- Find complete DRAM row
- Template memory for bit flips
- Spray page tables
- Flip page tables

PressHammer Exploit

Exploit Steps

- Find complete DRAM row
- Template memory for bit flips
- Spray page tables
- Flip page tables
- $\rightarrow$ Profit

Find Complete DRAM Row

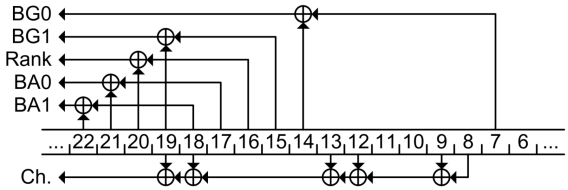
- DRAM row typically 4 kB

Find Complete DRAM Row

- DRAM row typically 4 kB
- Memory Page typically 4 kB

Find Complete DRAM Row

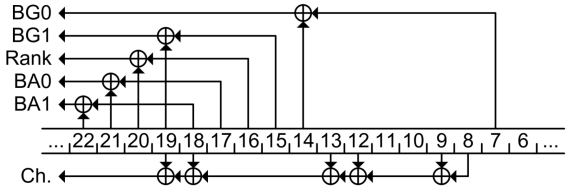
- DRAM row typically 4 kB
- Memory Page typically 4 kB
- No 1:1 mapping



Pessl et al. [4]

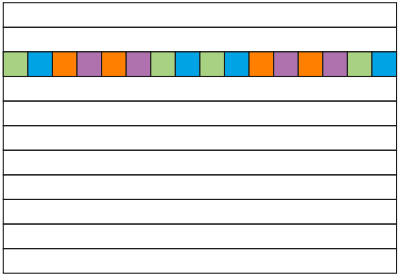
Find Complete DRAM Row

- DRAM row typically 4 kB
- Memory Page typically 4 kB
- No 1:1 mapping



Pessl et al. [4]

DRAM Row Finding

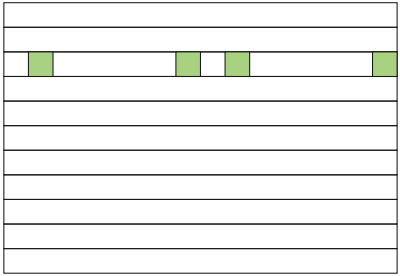


Bank 0

Bank 1

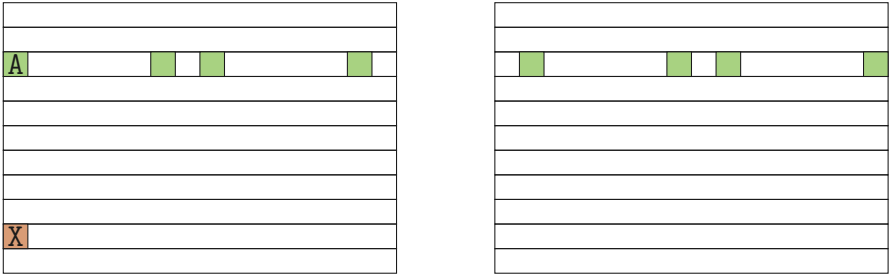
DRAM Row Finding

Bank 0



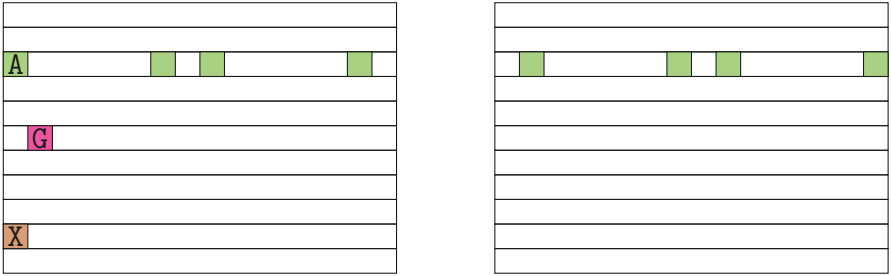
Bank 1

DRAM Row Finding



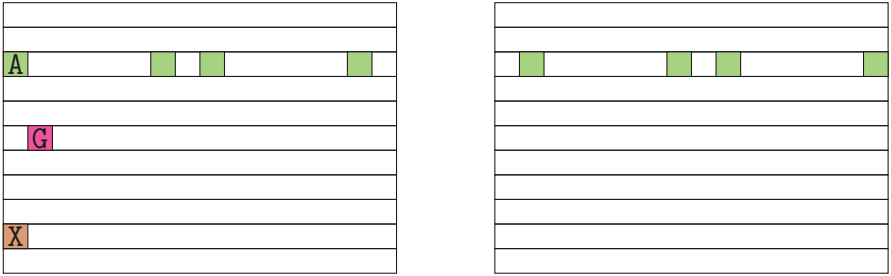
R1 Bank conflict with `X`

DRAM Row Finding



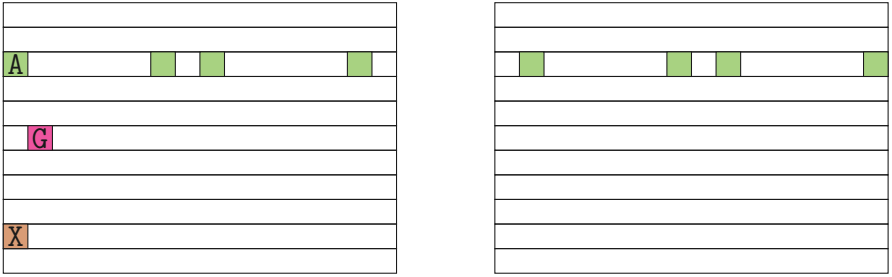
R1 Bank conflict with X

DRAM Row Finding



R1 Bank conflict with X ✓

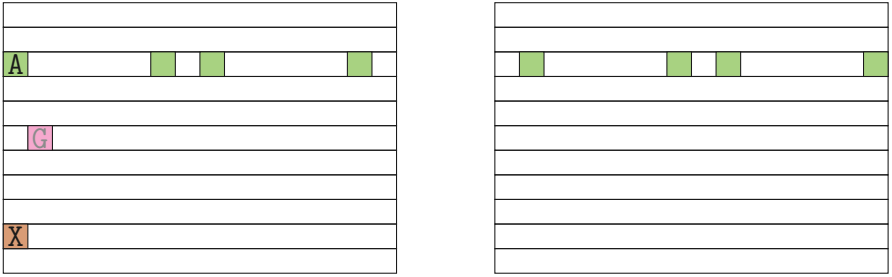
DRAM Row Finding



R1 Bank conflict with `X` ✓

R2 No bank conflict with `A`

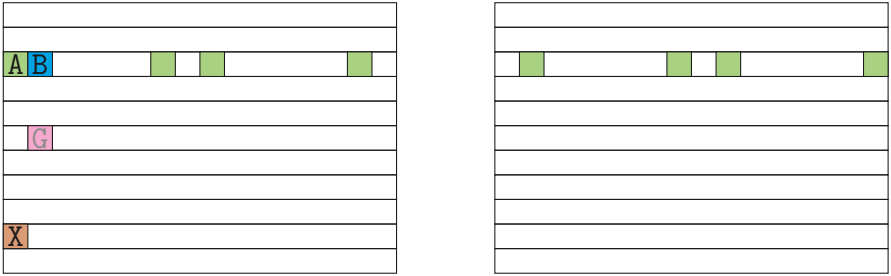
DRAM Row Finding



R1 Bank conflict with `X` ✓

R2 No bank conflict with `A` ✗

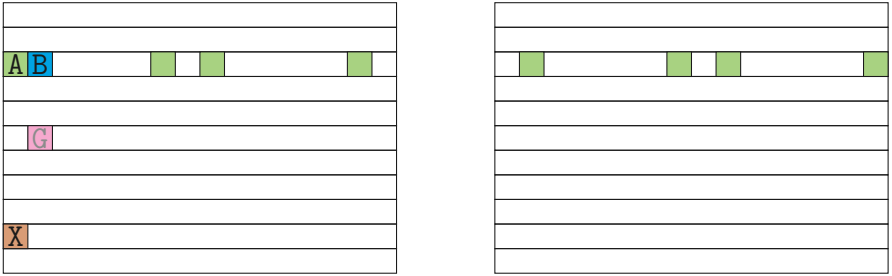
DRAM Row Finding



R1 Bank conflict with `X`

R2 No bank conflict with `A`

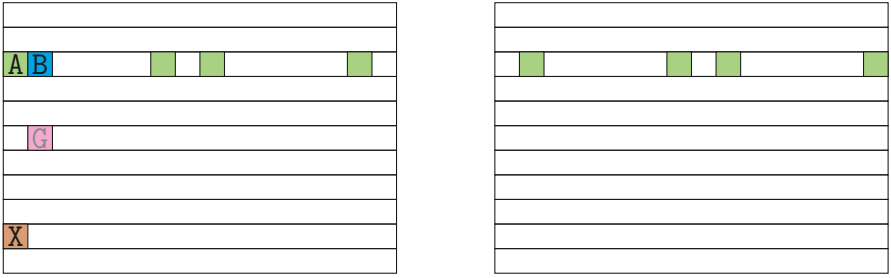
DRAM Row Finding



R1 Bank conflict with `X` ✓

R2 No bank conflict with `A`

DRAM Row Finding

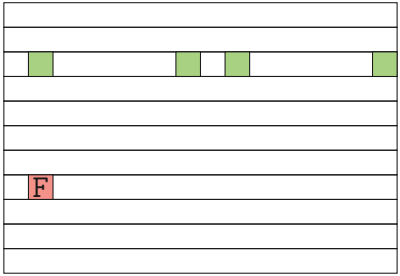


R1 Bank conflict with `X` ✓

R2 No bank conflict with `A` ✓

DRAM Row Finding

Bank 0

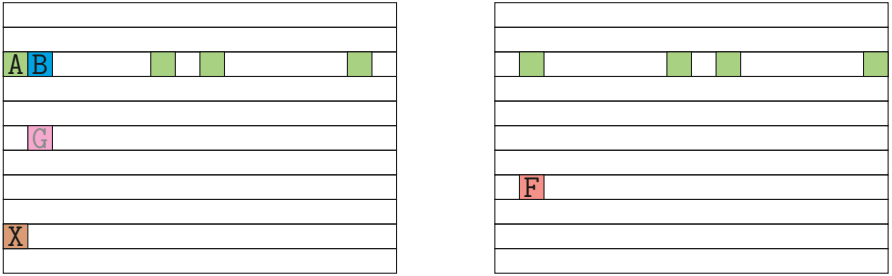


Bank 1

R1 Bank conflict with `X`

R2 No bank conflict with `A`

DRAM Row Finding

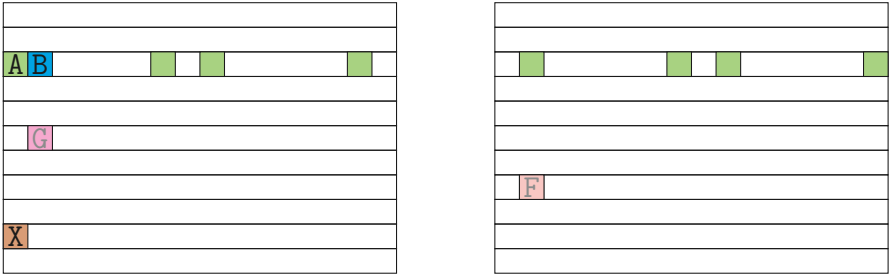


Bank 0 Bank 1

R1 Bank conflict with X ✗

R2 No bank conflict with A

DRAM Row Finding

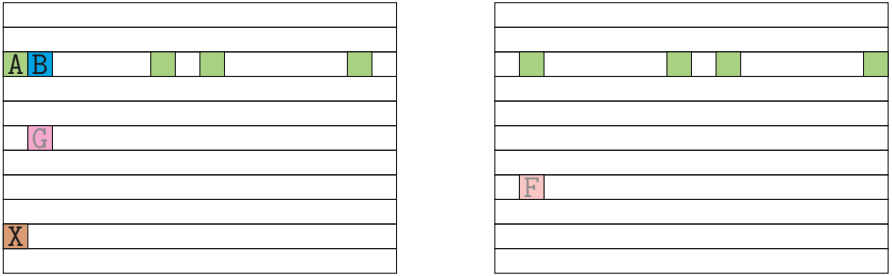


Bank 0 Bank 1

R1 Bank conflict with X ✗

R2 No bank conflict with A

DRAM Row Finding

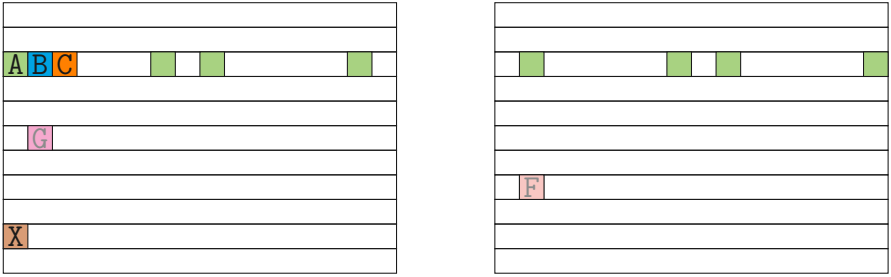


Bank 0 Bank 1

R1 Bank conflict with X ✗

R2 No bank conflict with A ✓

DRAM Row Finding

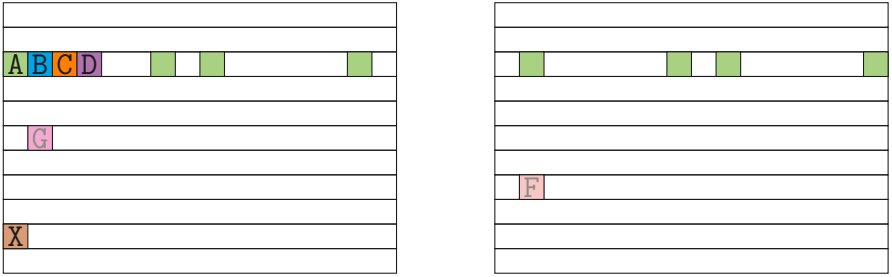


Bank 0 Bank 1

R1 Bank conflict with `X` ✓

R2 No bank conflict with `A` ✓

DRAM Row Finding



Bank 0 Bank 1

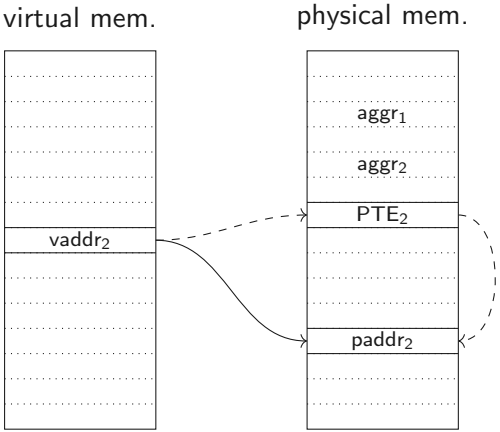
R1 Bank conflict with `X` ✓

R2 No bank conflict with `A` ✓

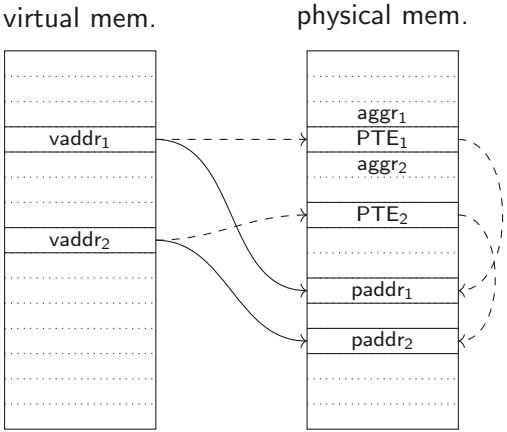
DRAM Row Finding Accuracy and Speed

# Meas.	Duration per Row	Found CLs	Correct CLs
250	40 ms, $\sigma =$ 17	124, $\sigma = 4.9$	119, $\sigma = 14$
500	45 ms, $\sigma =$ 15	128, $\sigma = 0.8$	125, $\sigma = 16$
1000	116 ms, $\sigma =$ 40	125, $\sigma = 4.2$	123, $\sigma = 4.3$
2500	260 ms, $\sigma = 109$	125, $\sigma = 4.9$	124, $\sigma = 5.2$

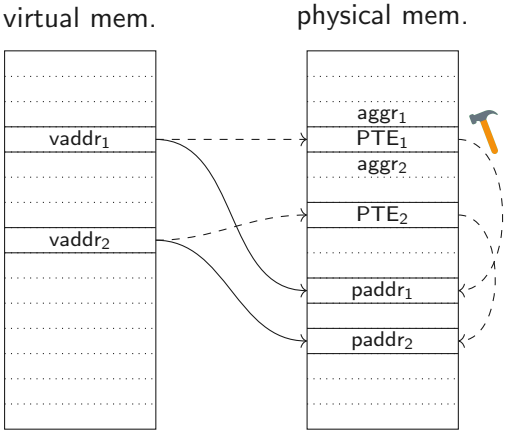
Page Table Entry Flipping



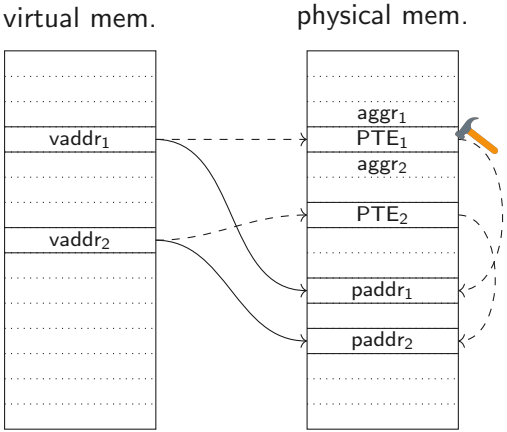
Page Table Entry Flipping



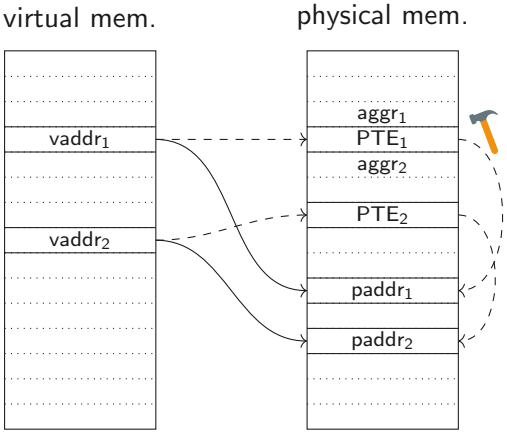
Page Table Entry Flipping



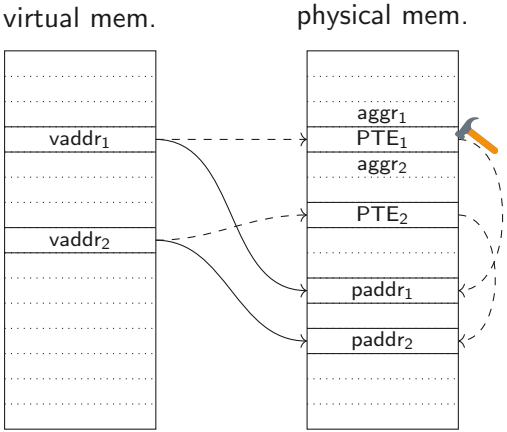
Page Table Entry Flipping



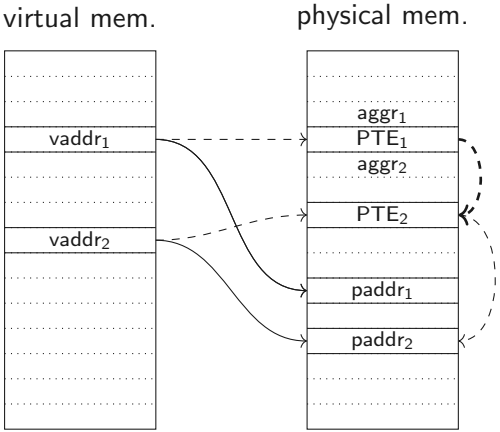
Page Table Entry Flipping



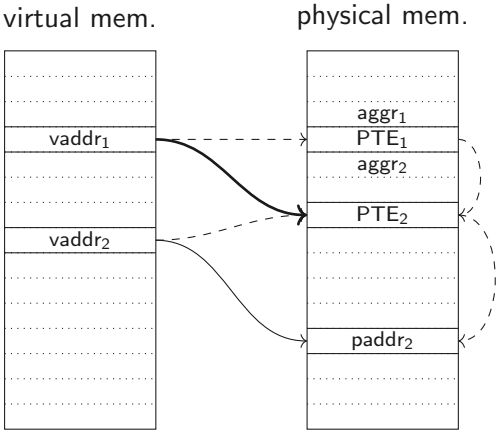
Page Table Entry Flipping



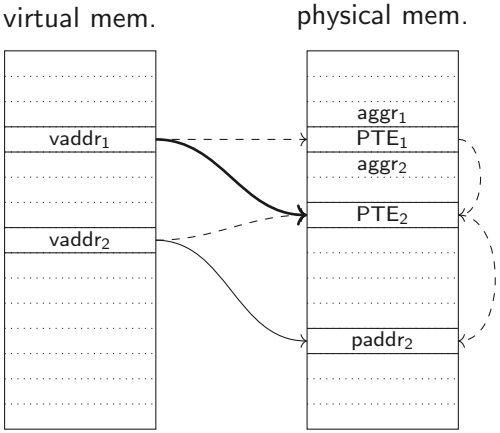
Page Table Entry Flipping



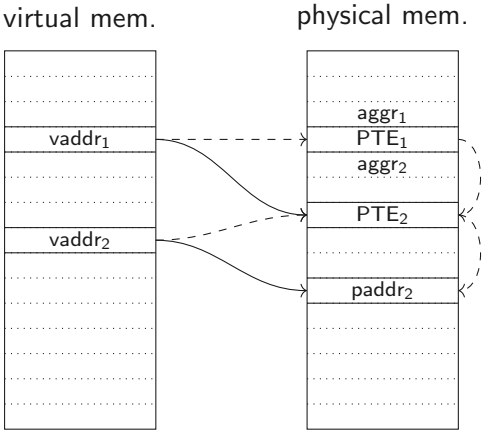
Page Table Entry Flipping



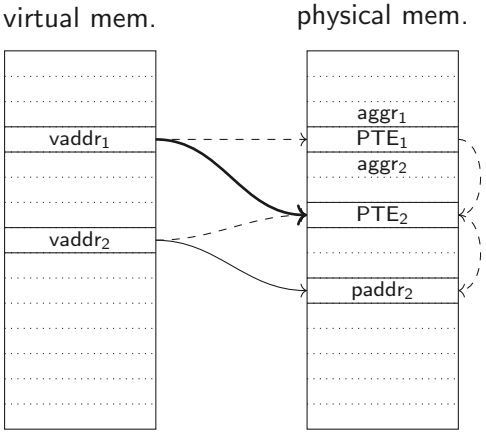
Page Table Entry Flipping



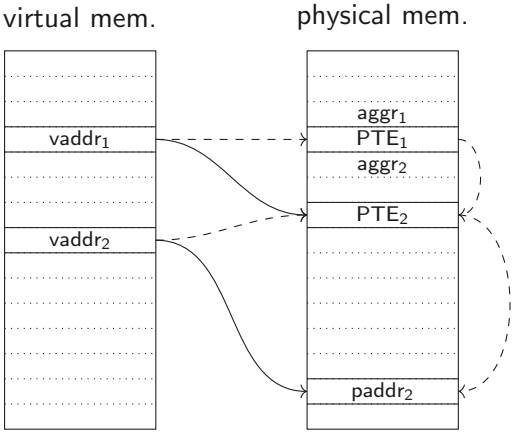
Page Table Entry Flipping



Page Table Entry Flipping



Page Table Entry Flipping

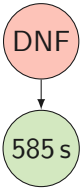


Exploit Results

Exploit Results

Did not successfully hammer page table.

Exploit Results



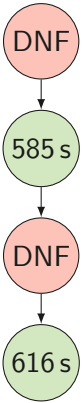
Did not successfully hammer page table.

Exploit Results

Did not successfully hammer page table.

Hammered page table, but didn't point to other.

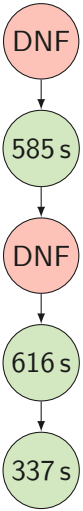
Exploit Results



Did not successfully hammer page table.

Hammered page table, but didn't point to other.

Exploit Results



Did not successfully hammer page table.

Hammered page table, but didn't point to other.

TU Graz

Presshammer

Rowhammer and Rowpress without Physical Address Information

Jonas Juffinger **Sudheendra Raghav Neela**
Martin Heckel **Lukas Schwarz**
Florian Adamsky **Daniel Gruss**

2024-07-19

jonas.juffinger@iaik.tugraz.at
NotImaginary_
www.jonasjuffinger.com