Security:
Can we afford to have it?
Can we afford not to have it?

Daniel Gruss

2025-10-26

Graz University of Technology


side channel
= obtaining meta-data and
deriving secrets from it
CHANGE MY MIND
imgflip.com

Side Channel or not?

Side Channel or not?

- Profiling cache utilization with performance counters?

Side Channel or not?

- Profiling cache utilization with performance counters? → No

Side Channel or not?

- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key?

Side Channel or not?

- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes

Side Channel or not?

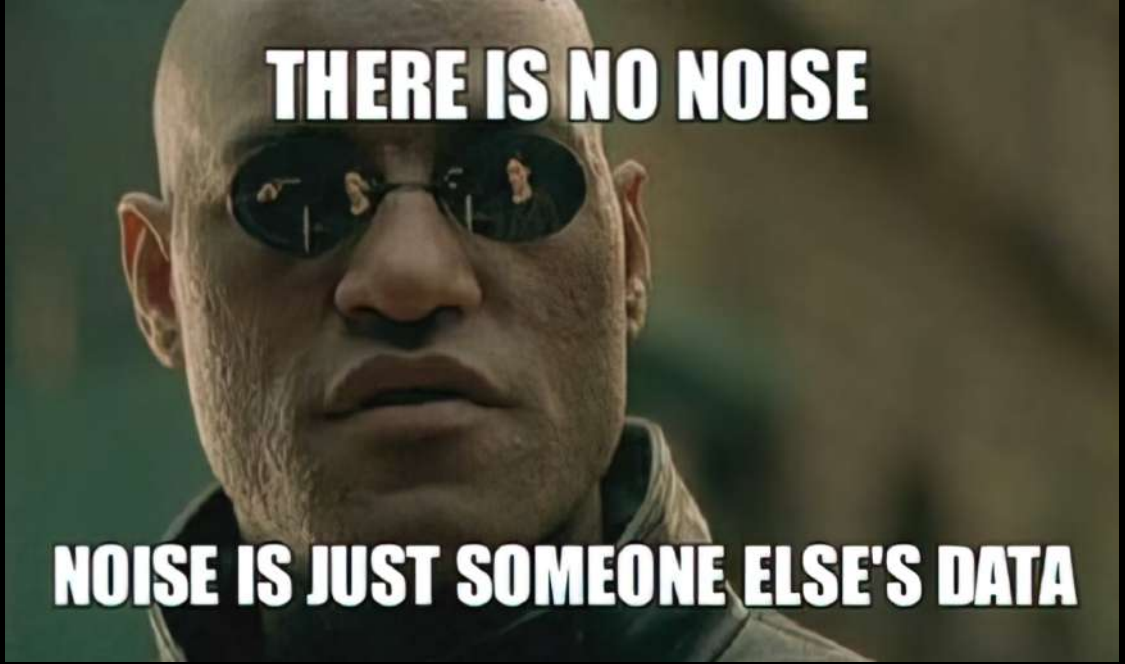
- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload?

Side Channel or not?

- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No

- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload? $\rightarrow$ No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings?

- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload? $\rightarrow$ No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings? $\rightarrow$ Yes


THERE IS NO NOISE
NOISE IS JUST SOMEONE ELSE'S DATA

1337 4242

FOOD CACHE

Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

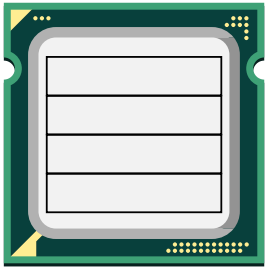
Can store **ALL** kinds of food.

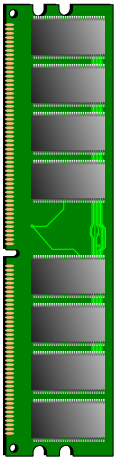
ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

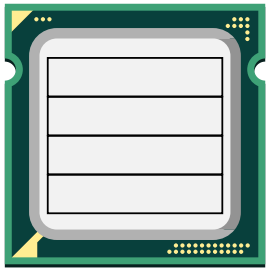
```
printf("%d", i);
printf("%d", i);
```

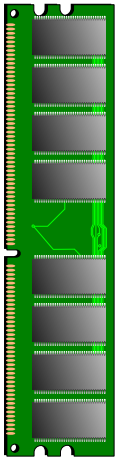




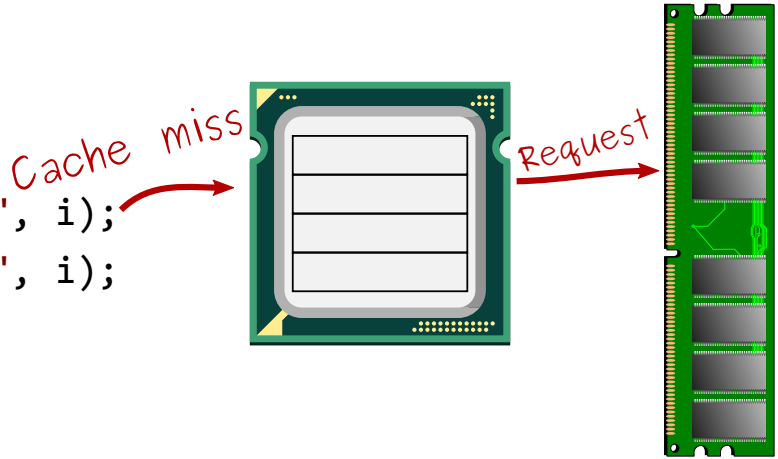
```
printf("%d", i);
printf("%d", i);
```

Cache miss

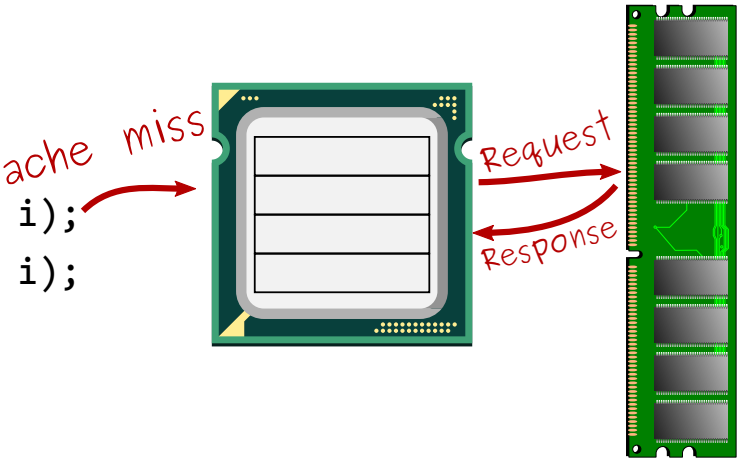


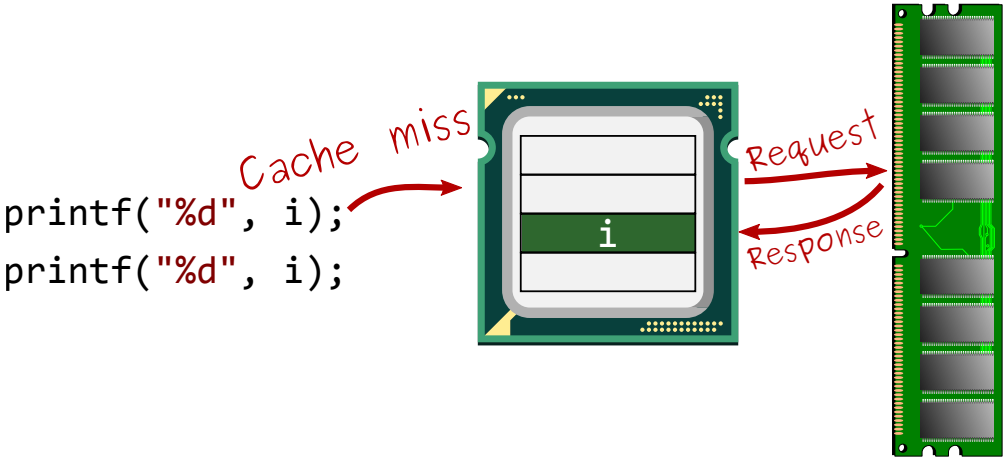


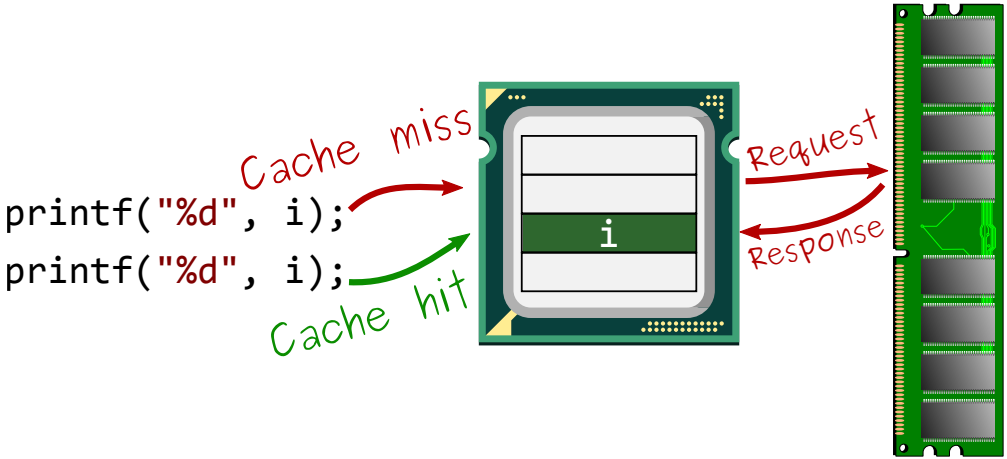
```
printf("%d", i);
printf("%d", i);
```

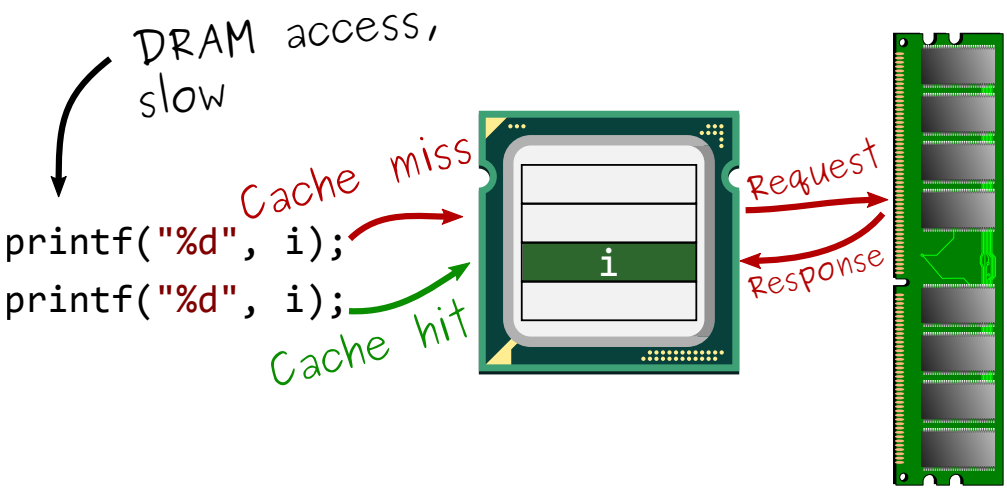


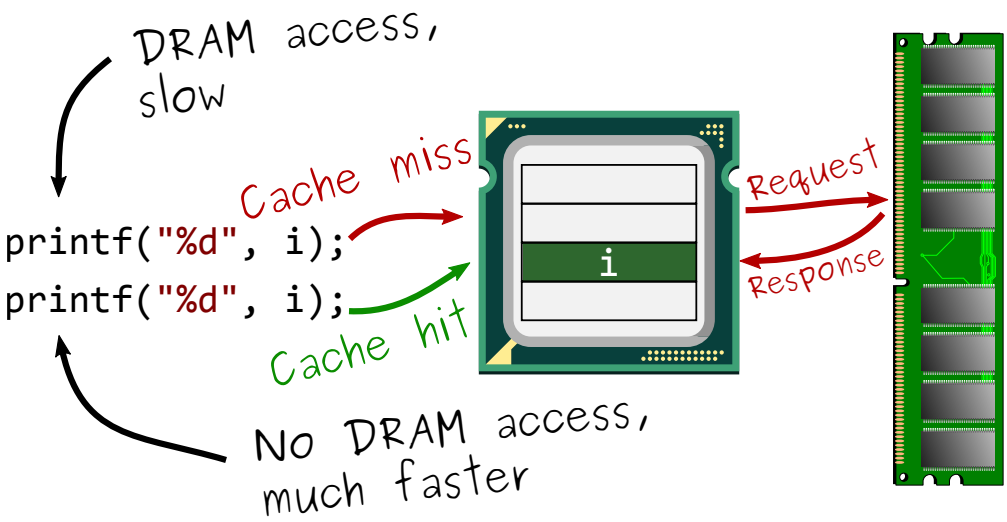
```
printf("%d", i);
printf("%d", i);
```




printf("%d", i);
printf("%d", i);
Cache miss
i
Request
Response


printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response


DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

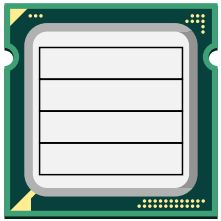

DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access, much faster

Shared Memory

ATTACKER

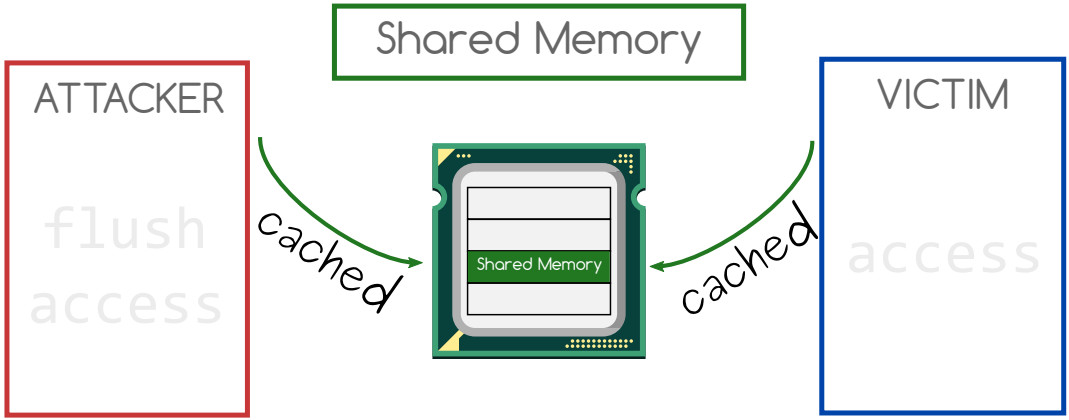
flush

access



VICTIM

access

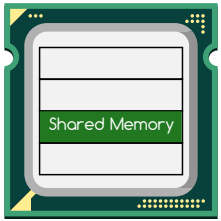

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

Shared Memory

ATTACKER

flush

access



VICTIM

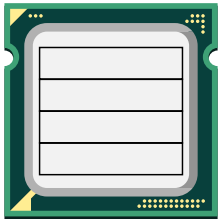
access

Shared Memory

ATTACKER

flush

access



VICTIM

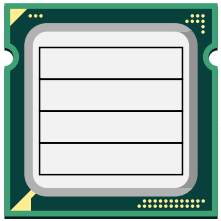
access

Shared Memory

ATTACKER

flush

access



VICTIM

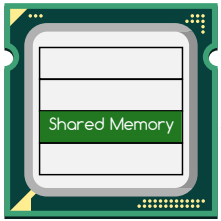
access

Shared Memory

ATTACKER

flush

access



VICTIM

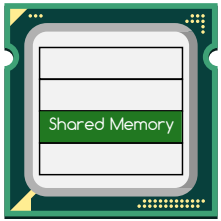
access

Shared Memory

ATTACKER

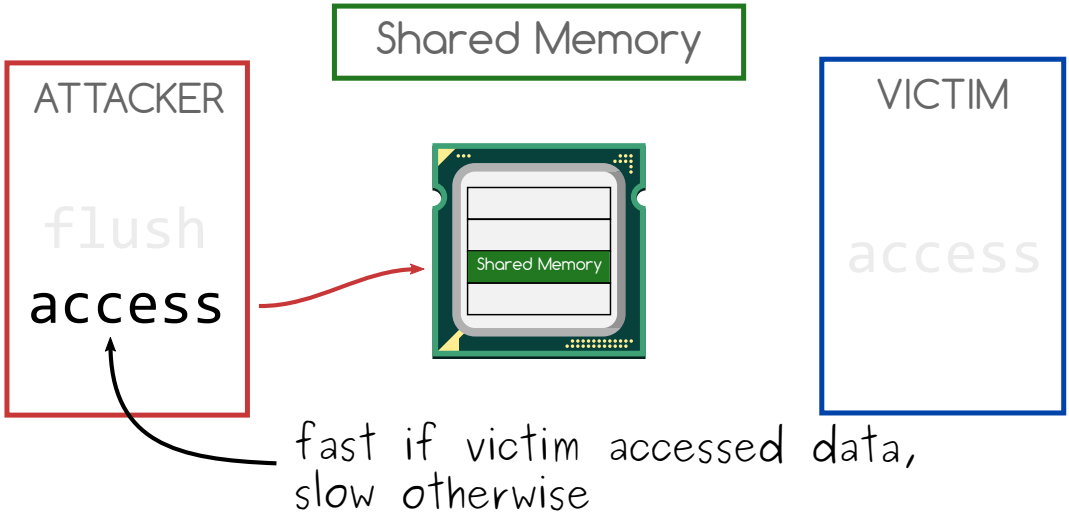
flush

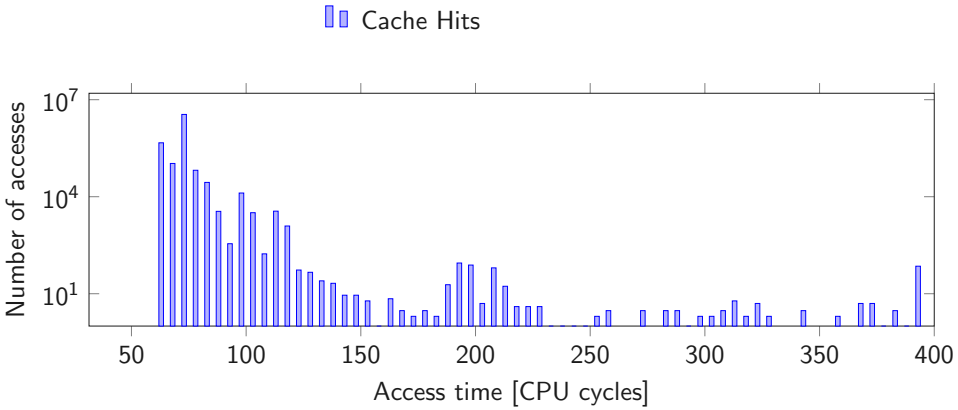
access



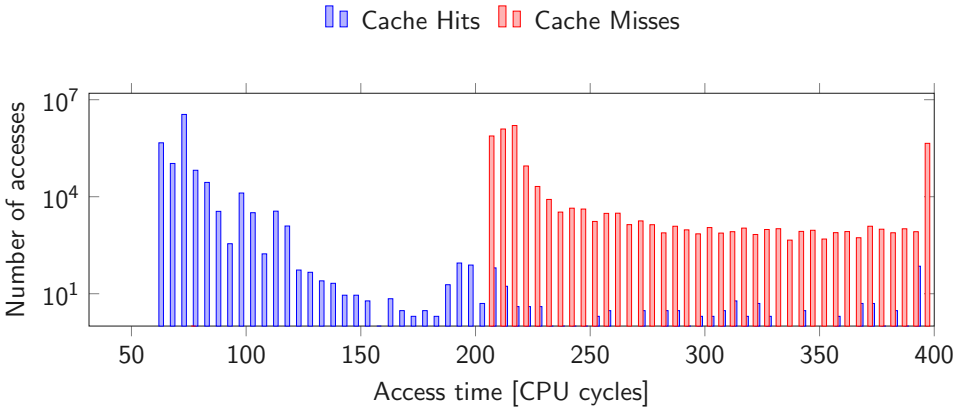
VICTIM

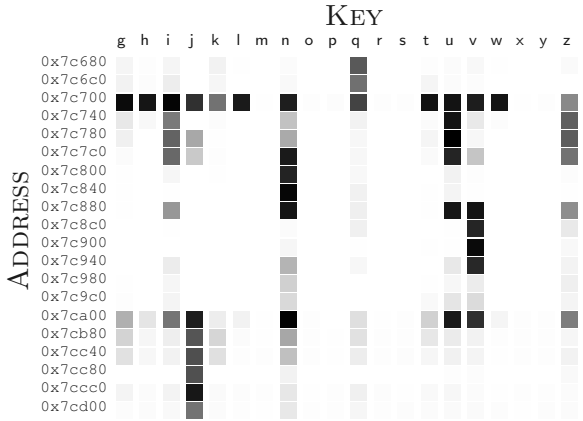
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
fast if victim accessed data,
slow otherwise


Cache Hits
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]

Memory Access Latency




Key
g h i j k l m n o p q r s t u v w x y z
Address
0x7c680
0x7c6c0
0x7c700
0x7c740
0x7c780
0x7c7c0
0x7c800
0x7c840
0x7c880
0x7c8c0
0x7c900
0x7c940
0x7c980
0x7c9c0
0x7ca00
0x7cb80
0x7cc40
0x7cc80
0x7ccc0
0x7cd00

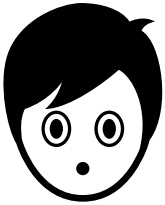
- Add a layer of indirection to test

```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Add a layer of indirection to test

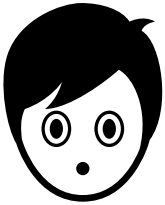
```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Then check whether any part of `array` is cached



- Flush+Reload over all pages of the array

- Index of cache hit reveals data



- Flush+Reload over all pages of the array

- Index of cache hit reveals data
- Permission check is in some cases not fast enough

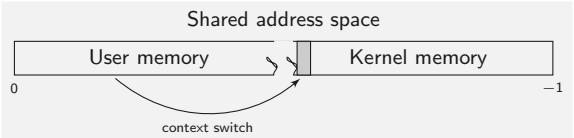
Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

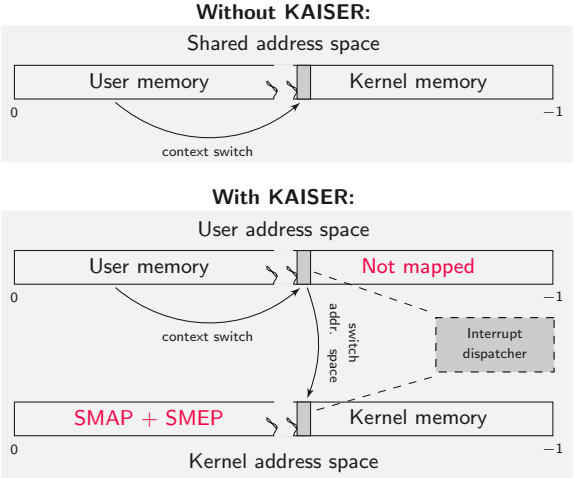
KAISER /ˈkʌɪzə/

1. [german] Emperor, ruler of an empire
2. largest penguin, emperor penguin

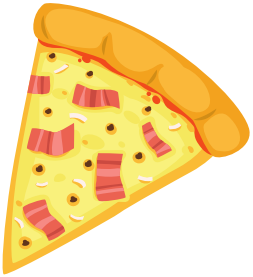
Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

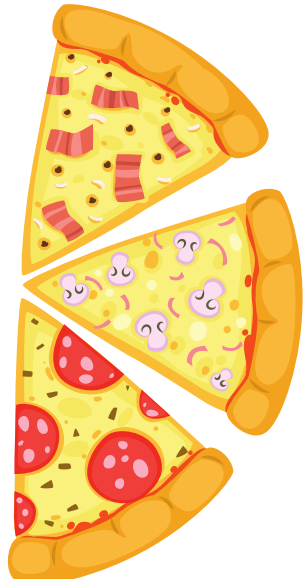
Without KAISER:

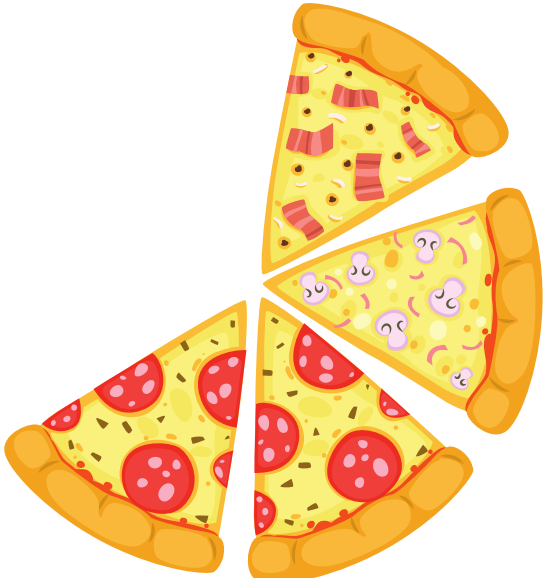


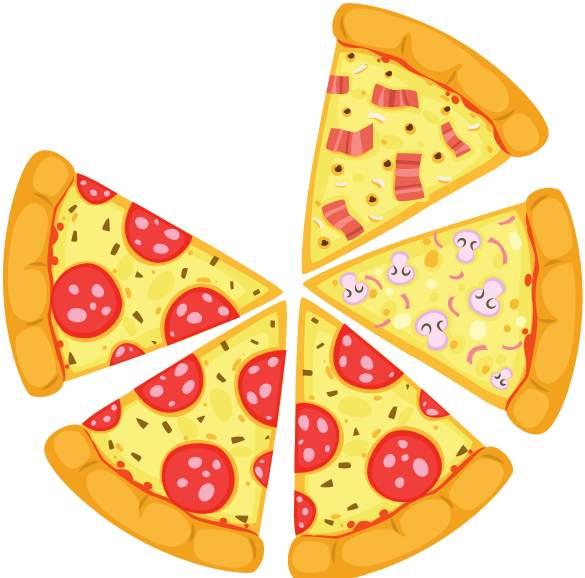

Without KAISER:
Shared address space
User memory
Kernel memory
0
−1
context switch
With KAISER:
User address space
User memory
Not mapped
0
−1
context switch
addr. space switch
Interrupt dispatcher
SMAP + SMEP
Kernel memory
0
−1
Kernel address space

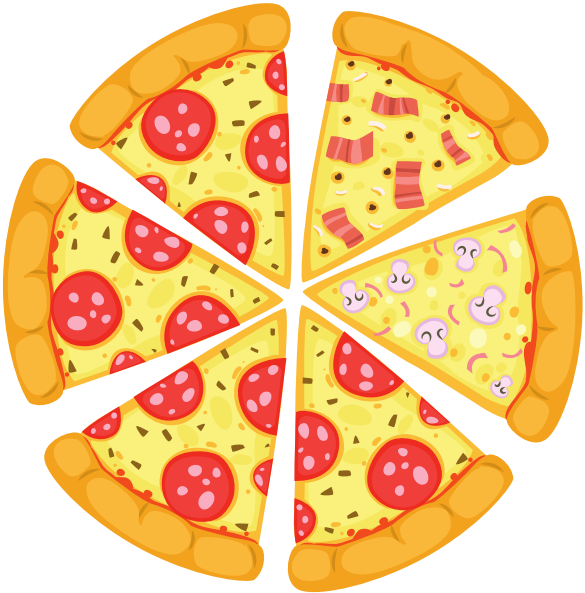

PIZZA
SPECIAL RECIPES












»A table for 6 please«

Speculative Cooking


»A table for 6 please«


PIZZA
SPECIAL RECIPES


PIZZA
SPECIAL RECIPES

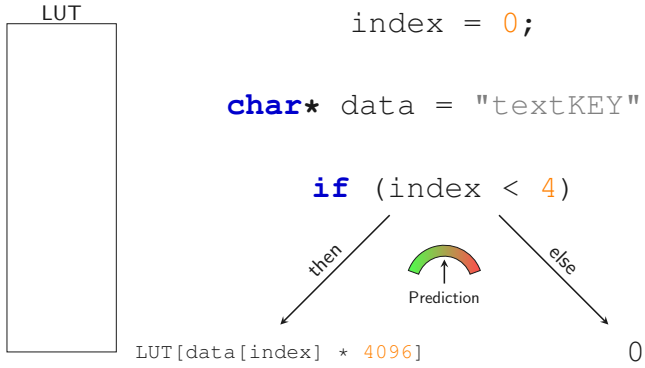

Pizza

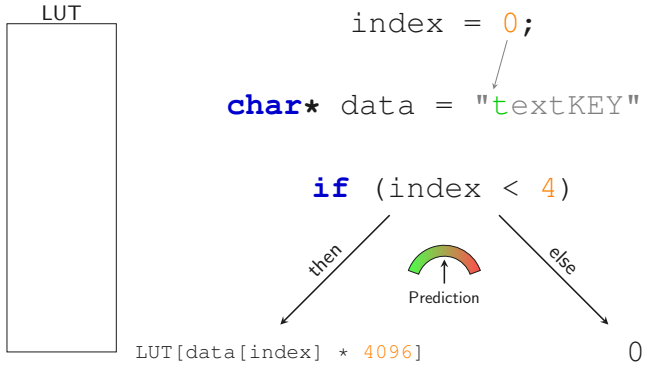

PIZZA
SPECIAL RECIPES

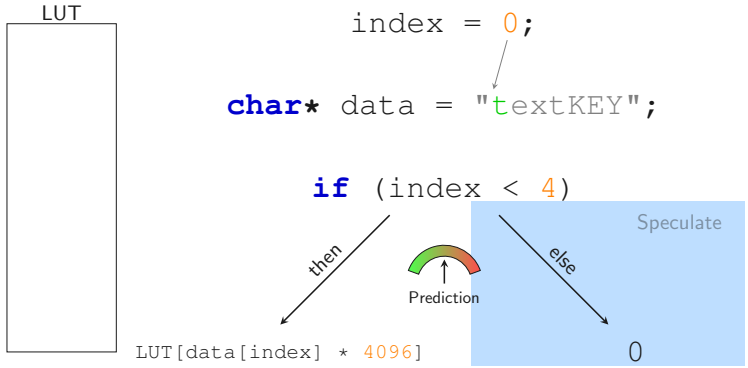

Pizza

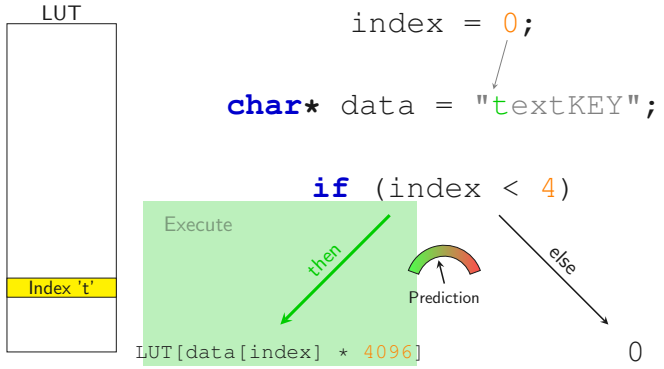

PIZZA
SPECIAL RECIPES

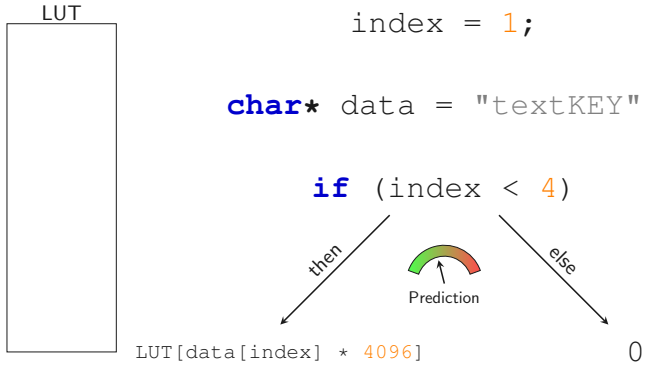

Pizza

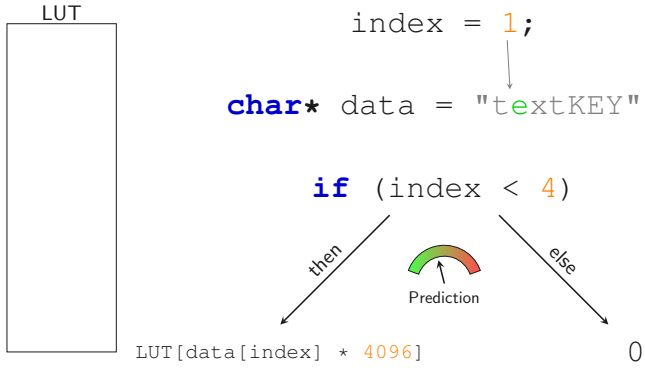

LUT
index = 0;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

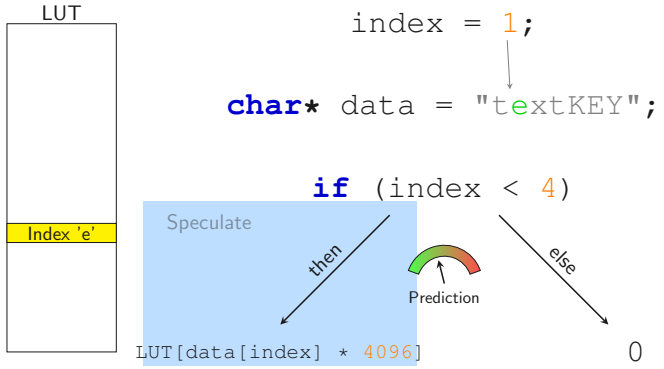

LUT
index = 0;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

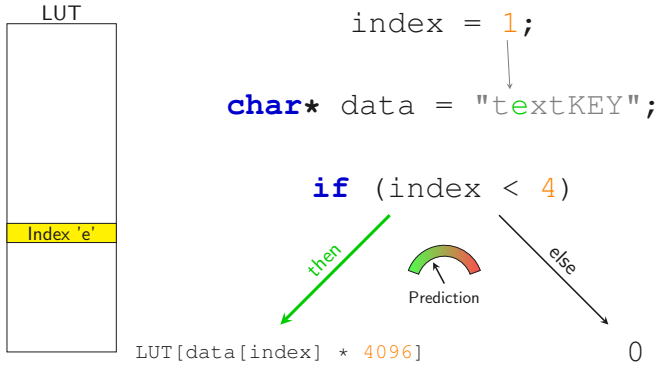

LUT
index = 0;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
Speculate
LUT[data[index] * 4096]
0

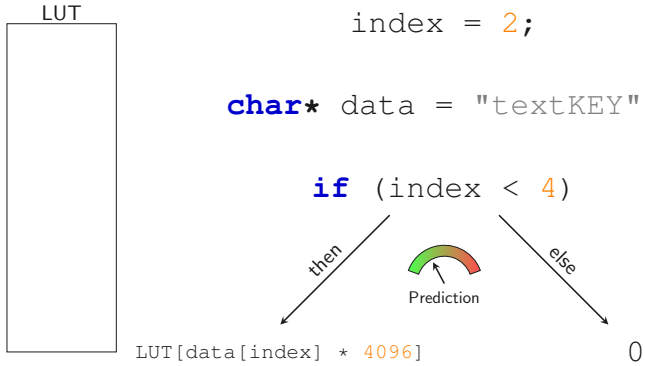

LUT
Index 't'
index = 0;
char* data = "textKEY";
if (index < 4)
Execute
then
else
Prediction
LUT[data[index] * 4096]
0

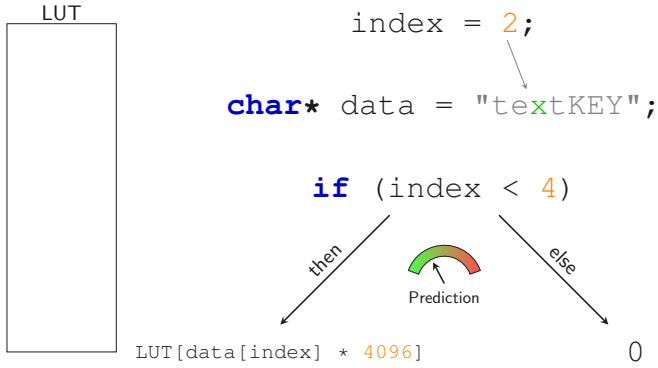

LUT
index = 1;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

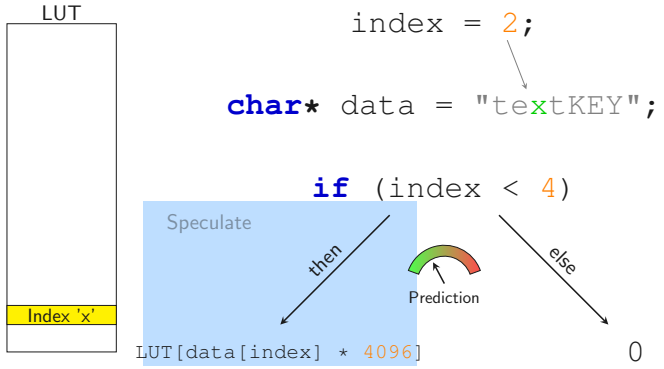

LUT
index = 1;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

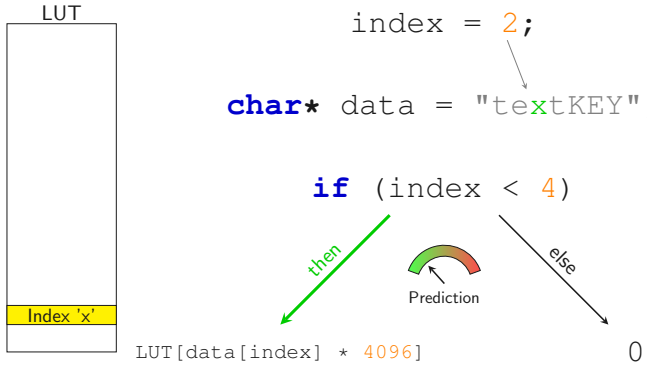

LUT
Index 'e'
index = 1;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

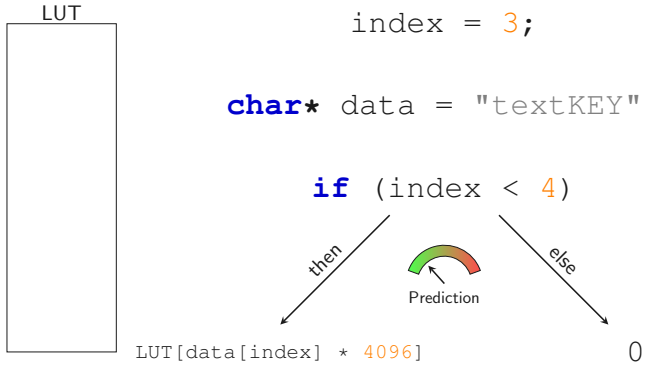

LUT
Index 'e'
index = 1;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

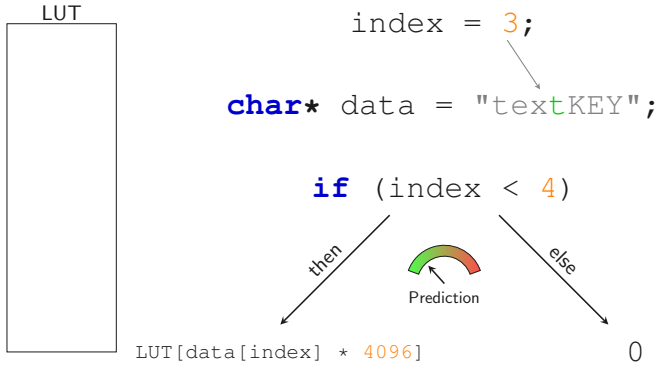

LUT
index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

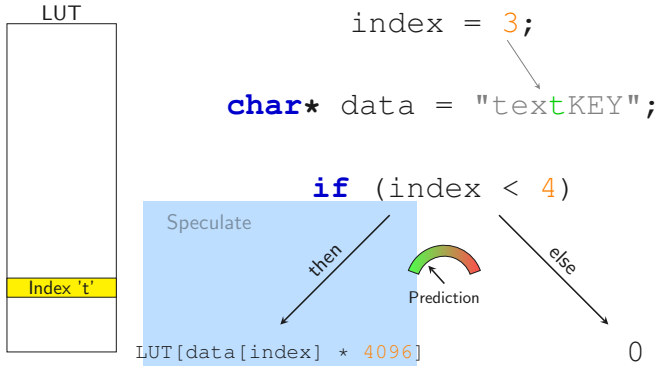

LUT
index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

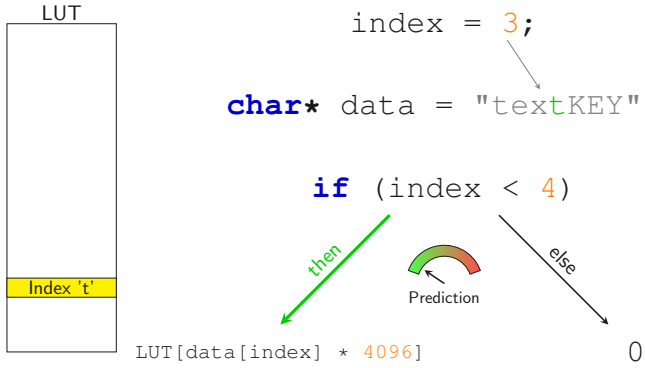

LUT
Index 'x'
index = 2;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

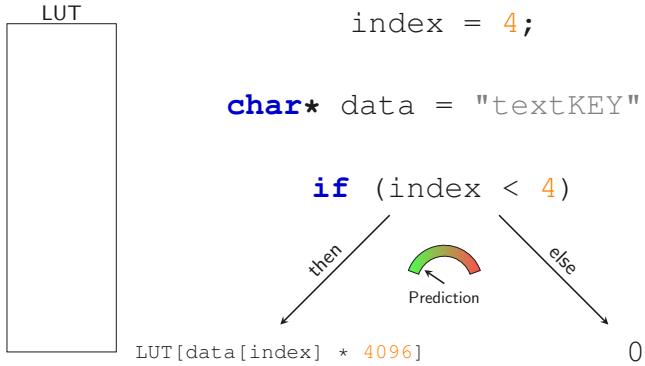

LUT
Index 'x'
index = 2;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

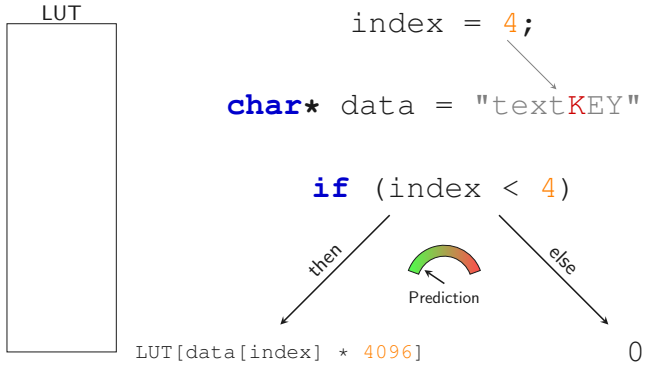

LUT
index = 3;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

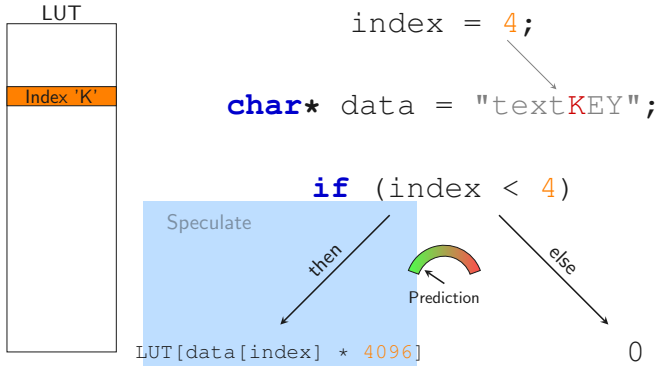

LUT
index = 3;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

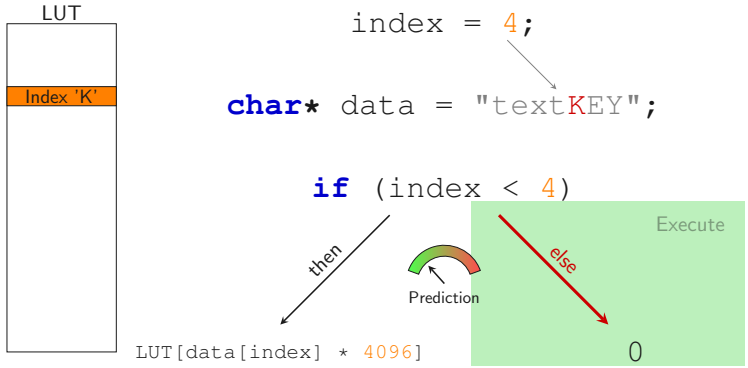

LUT
Index 't'
index = 3;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

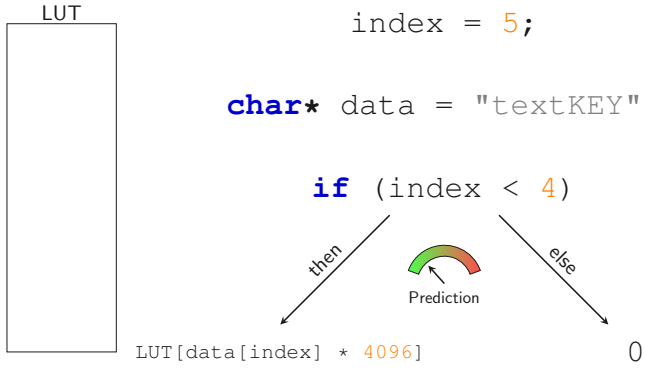

LUT
Index 't'
index = 3;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

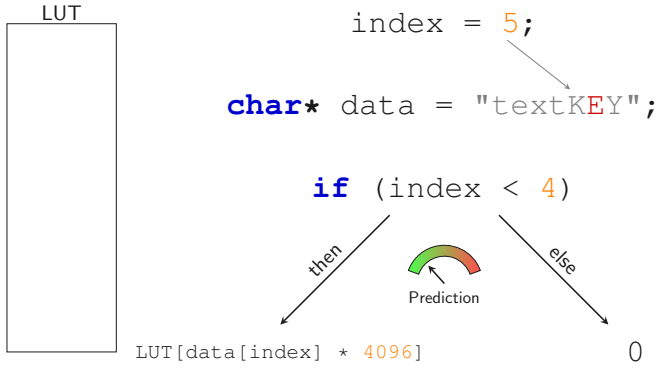

LUT
index = 4;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

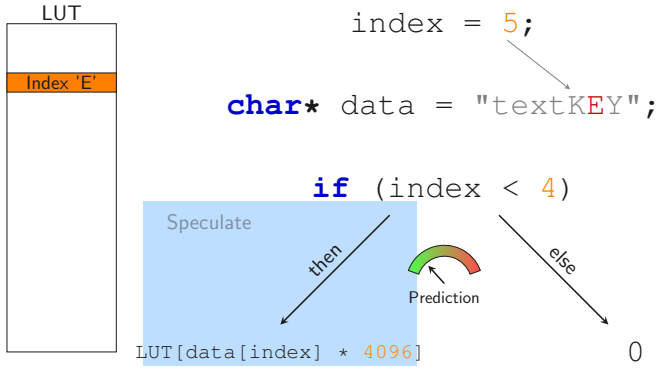

LUT
index = 4;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

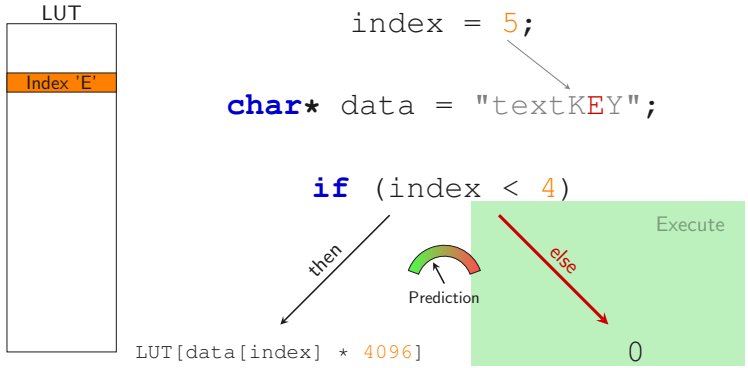

LUT
Index 'K'
index = 4;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

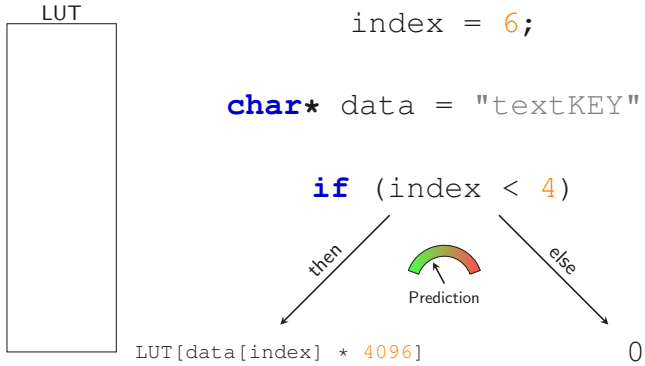

LUT
Index 'K'
index = 4;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
Execute
LUT[data[index] * 4096]
0


LUT
index = 5;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

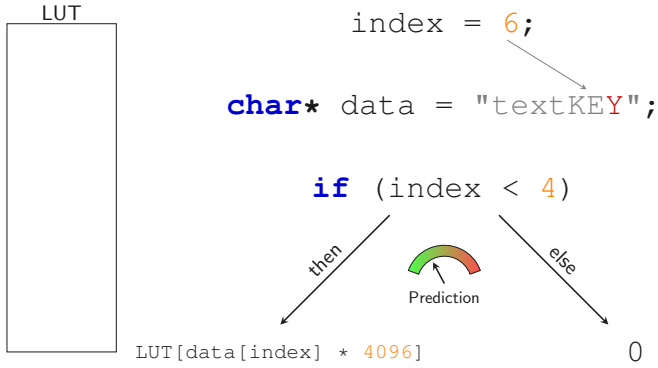

LUT
index = 5;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

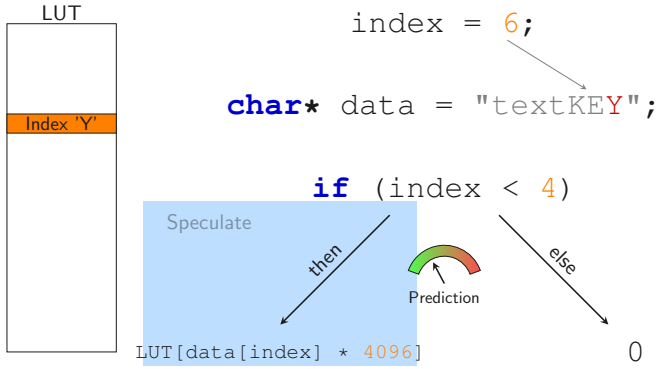

LUT
Index 'E'
index = 5;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

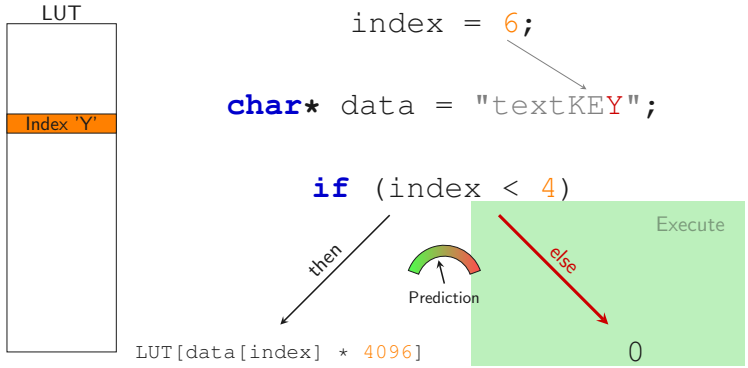

LUT
Index 'E'
index = 5;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
Execute
LUT[data[index] * 4096]
0

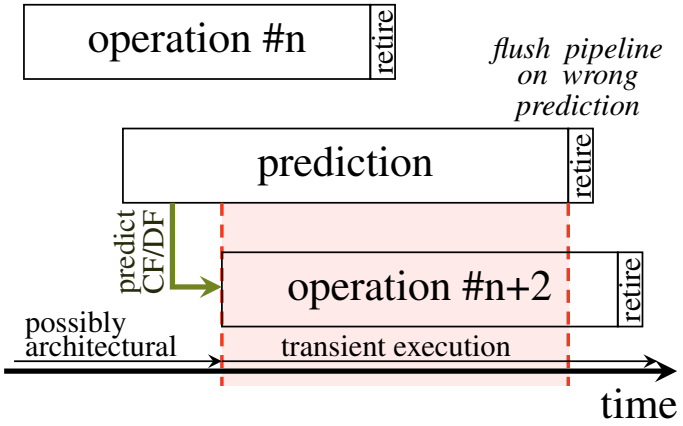

LUT
index = 6;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

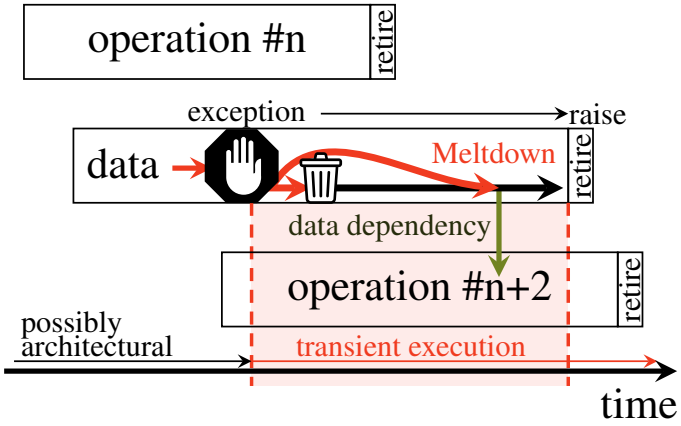
Spectre-PHT (v1)




LUT
Index 'Y'
index = 6;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0


LUT
Index 'Y'
index = 6;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
Execute
LUT[data[index] * 4096]
0


operation #n
retire
flush pipeline on wrong prediction
prediction
retire
predict CF/DF
operation #n+2
retire
possibly architectural
transient execution
time


operation #n
retire
exception
raise
data
Meltdown
retire
data dependency
operation #n+2
retire
possibly
architectural
transient execution
time

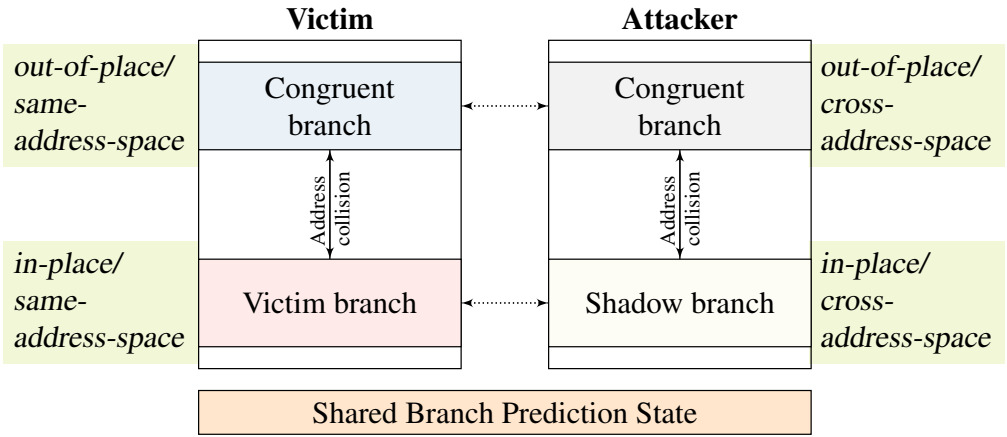
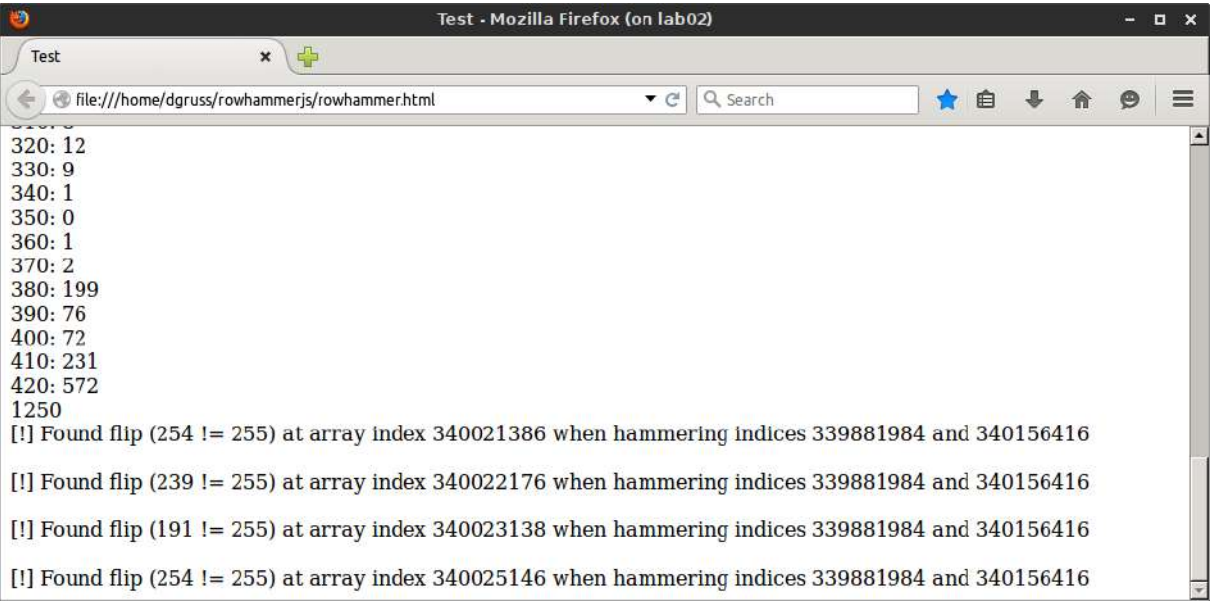
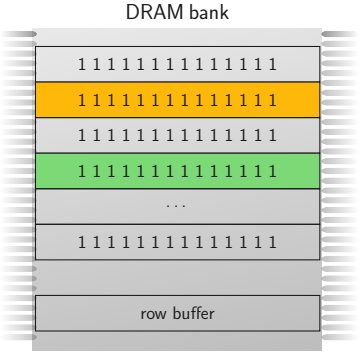
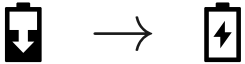

Victim
Attacker
out-of-place/ same-address-space
Congruent branch
Congruent branch
out-of-place/ cross-address-space
Address collision
Address collision
in-place/ same-address-space
Victim branch
Shadow branch
in-place/ cross-address-space
Shared Branch Prediction State

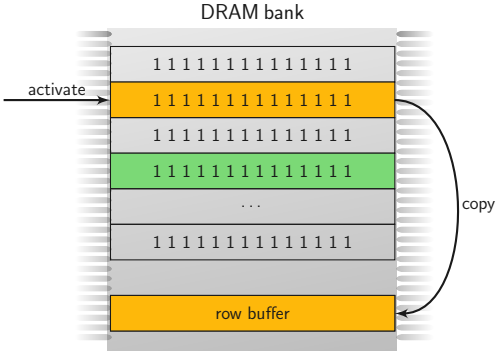
Table 1: Reported performance impacts of countermeasures

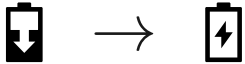
Defense \ Impact	Performance Loss	Benchmark
InvisiSpec	22%	SPEC
SafeSpec	3% (improvement)	SPEC2017 on MARSSx86
DAWG	2–12%, 1–15%	PARSEC, GAPBS
RSB Stuffing	no reports	
Retpoline	5–10%	real-world workload servers
Site Isolation	only memory overhead	
SLH	36.4%, 29%	Google microbenchmark suite
YSNB	60%	Phoenix
IBRS	20–30%	two sysbench 1.0.11 benchmarks
STIPB	30– 50%	Rodinia OpenMP, DaCapo
IBPB	no individual reports	
Serialization	62%, 74.8%	Google microbenchmark suite
SSBD/SSBB	2–8%	SYSmark®2014 SE & SPEC integer
KAISER/KPTI	0–2.6%	system call rates
L1TF mitigations	-3–31%	various SPEC

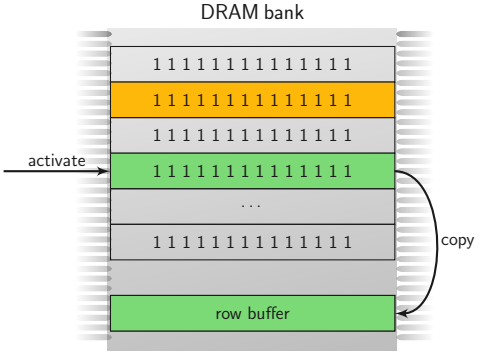

Test - Mozilla Firefox (on lab02)
Test
file:///home/dgruss/rowhammerjs/rowhammer.html
Search
320: 12
330: 9
340: 1
350: 0
360: 1
370: 2
380: 199
390: 76
400: 72
410: 231
420: 572
1250
[!] Found flip (254 != 255) at array index 340021386 when hammering indices 339881984 and 340156416
[!] Found flip (239 != 255) at array index 340022176 when hammering indices 339881984 and 340156416
[!] Found flip (191 != 255) at array index 340023138 when hammering indices 339881984 and 340156416
[!] Found flip (254 != 255) at array index 340025146 when hammering indices 339881984 and 340156416

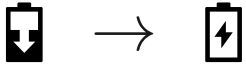

DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer



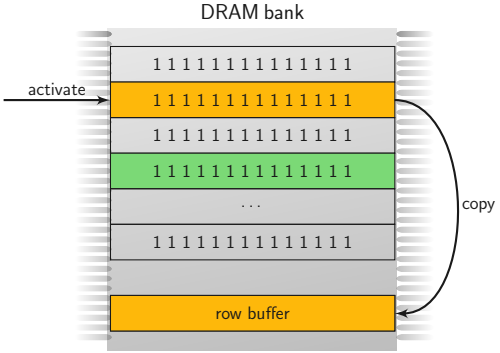

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
copy
row buffer

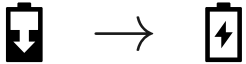




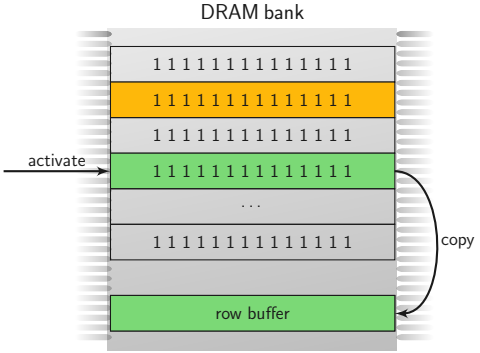


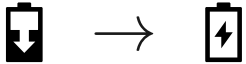
Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer



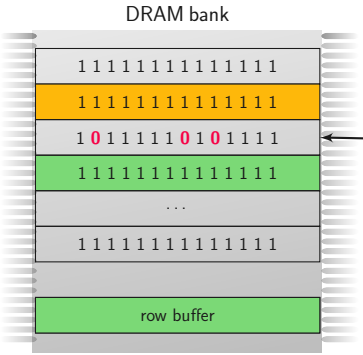


Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer

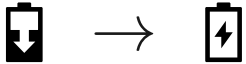




Cells leak faster upon proximate accesses $\rightarrow$ Rowhammer

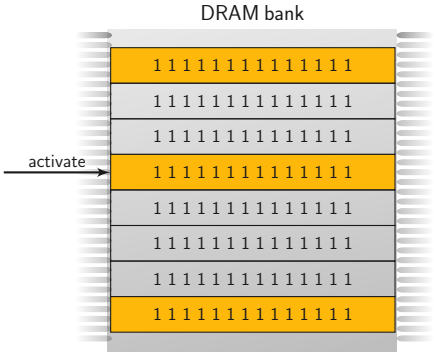


bit flips in row 2!

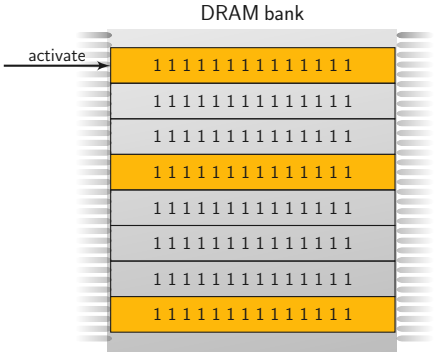


Cells leak faster upon proximate accesses → Rowhammer

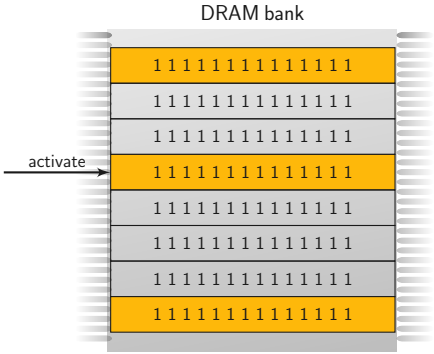
#1 - Single-sided hammering



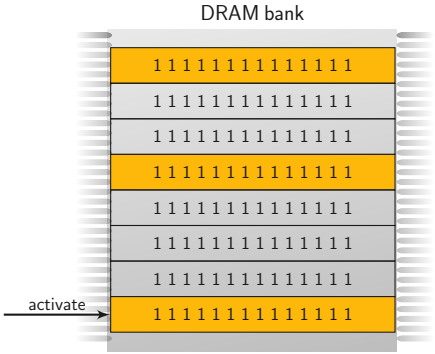
#1 - Single-sided hammering



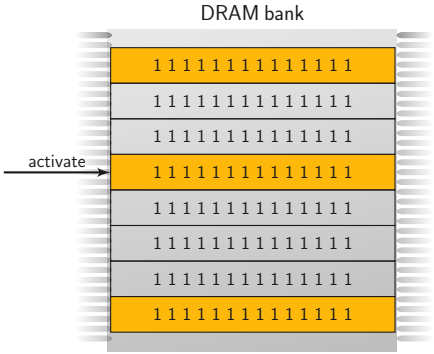
#1 - Single-sided hammering



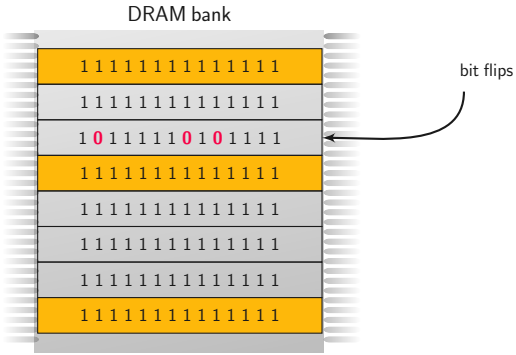
#1 - Single-sided hammering



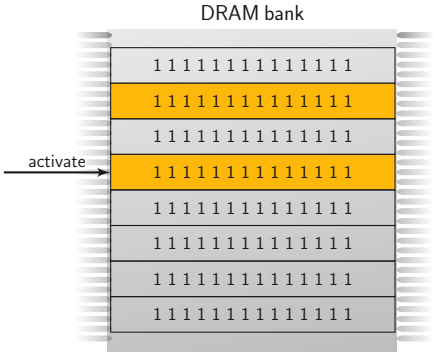
#1 - Single-sided hammering



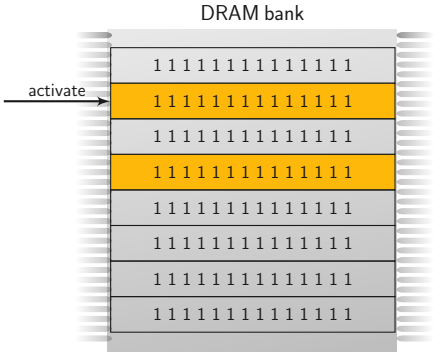
#1 - Single-sided hammering



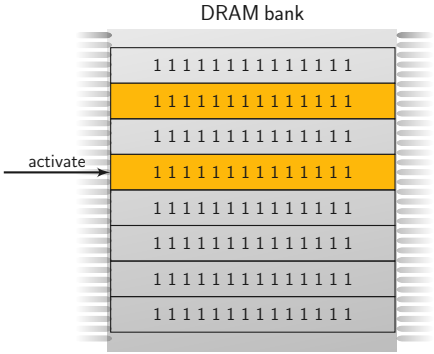
#2 - Double-sided hammering



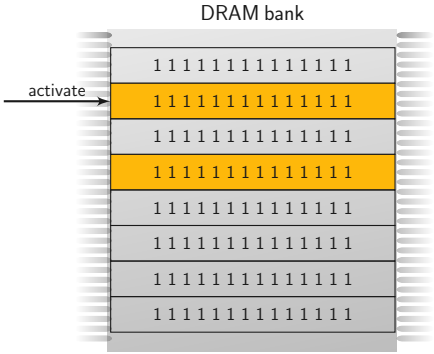
#2 - Double-sided hammering



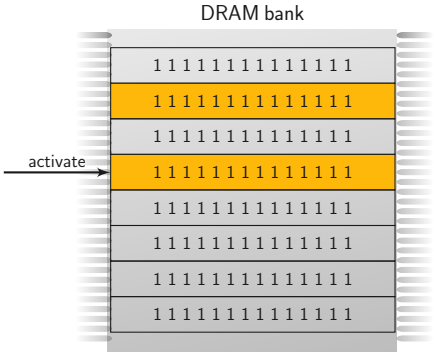
#2 - Double-sided hammering



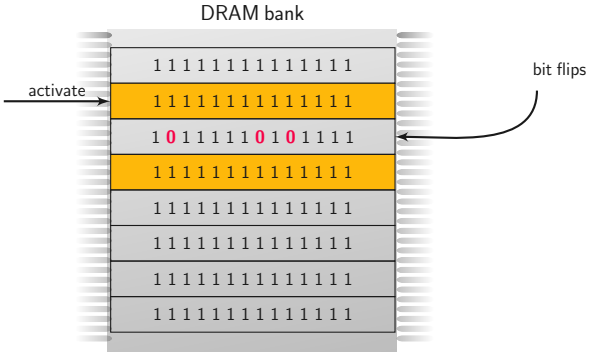
#2 - Double-sided hammering



#2 - Double-sided hammering



#2 - Double-sided hammering

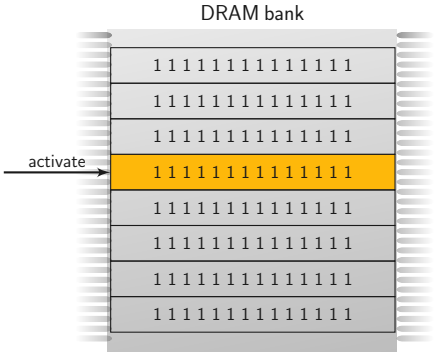


HAMMERING
TWO ROWS

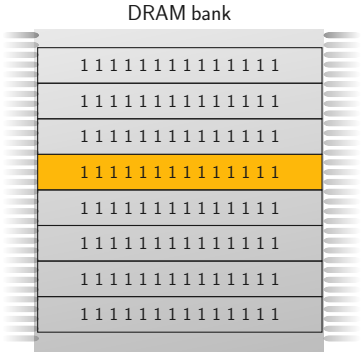
HAMMERING TWO ROWS

HAMMERING A SINGLE ROW

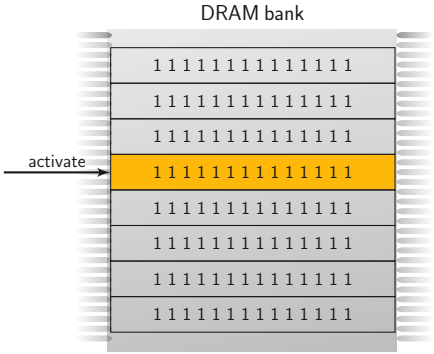
#3 - One-location hammering



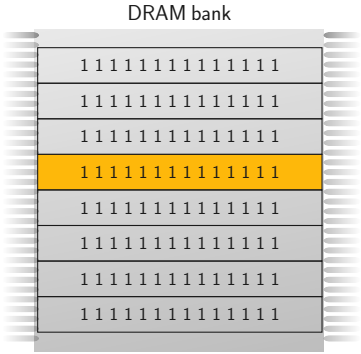
#3 - One-location hammering



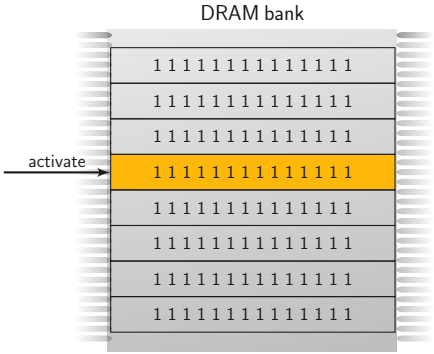
#3 - One-location hammering



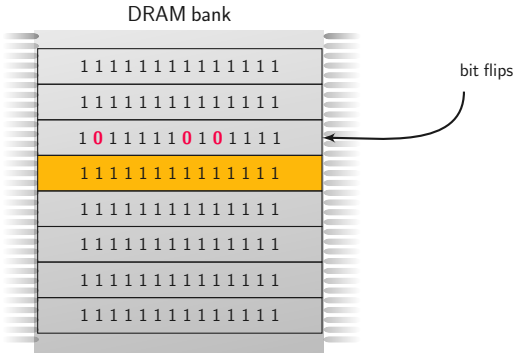
#3 - One-location hammering

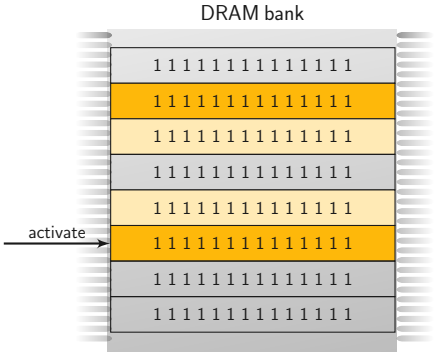


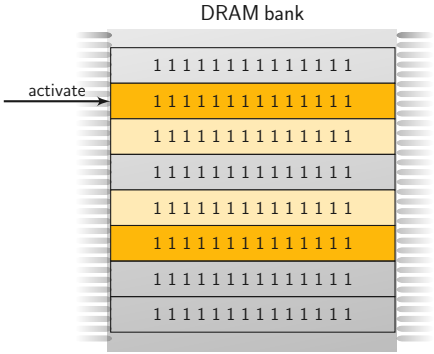
#3 - One-location hammering

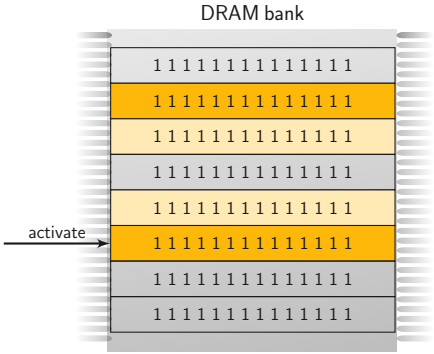


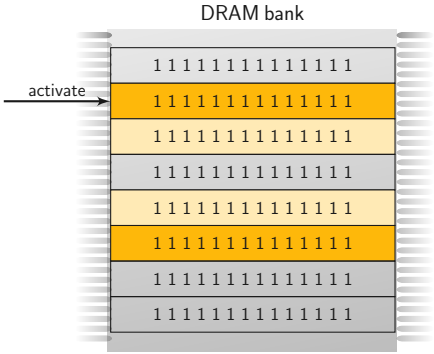
#3 - One-location hammering



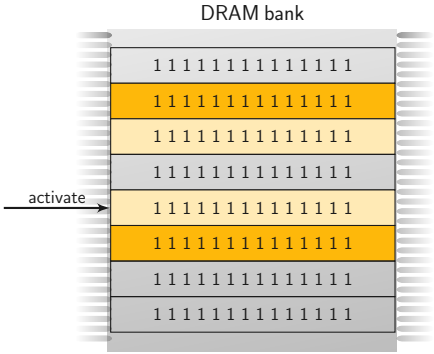

DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

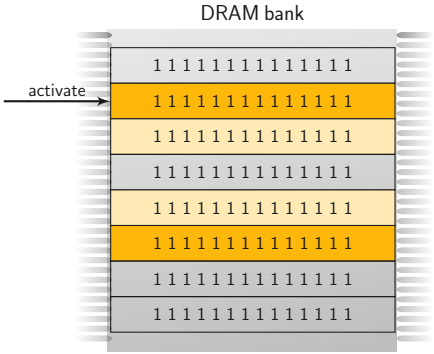

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

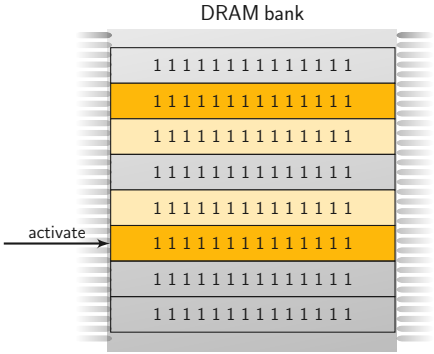

DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

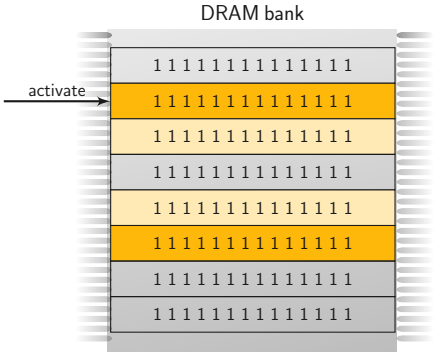

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

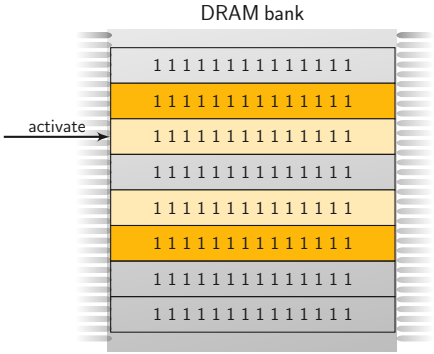
Half-Double


DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

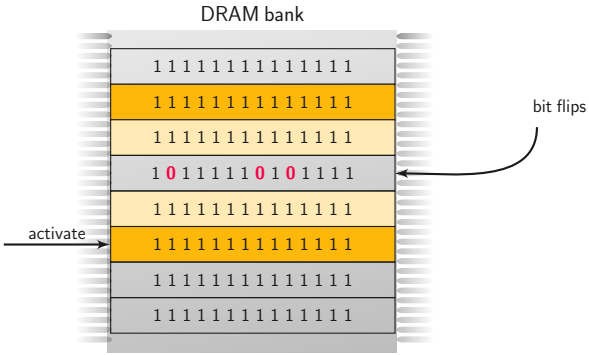

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

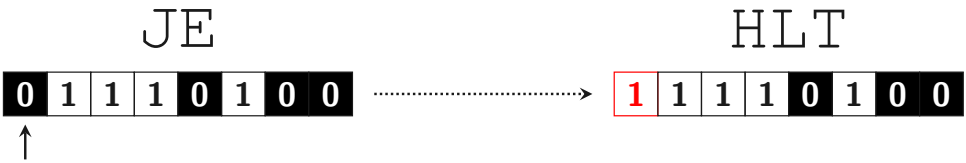

DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

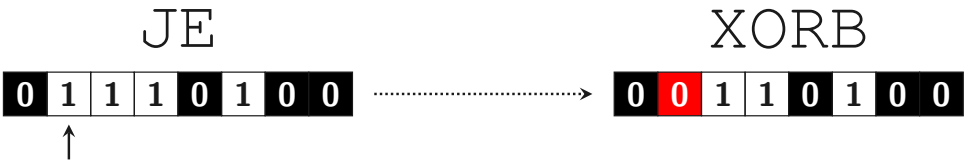

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1


DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Half-Double

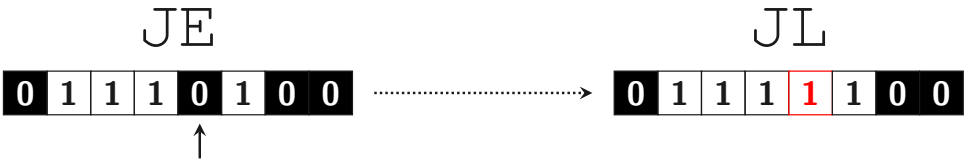

DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
bit flips
activate

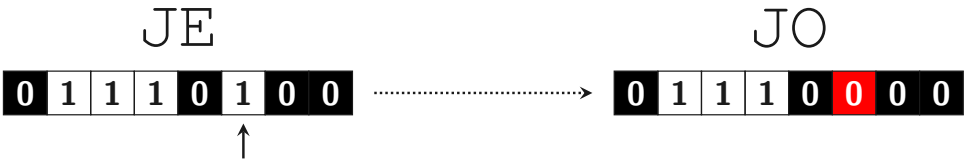

JE
0 1 1 1 0 1 0 0
HLT
1 1 1 1 0 1 0 0

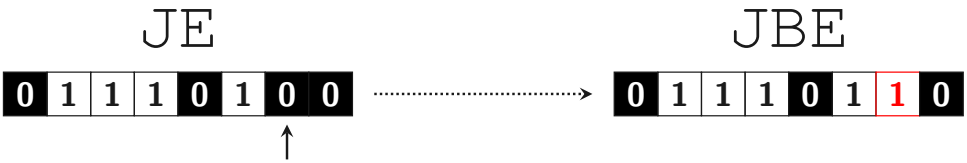

JE
0 1 1 1 0 1 0 0
XORB
0 0 1 1 0 1 0 0

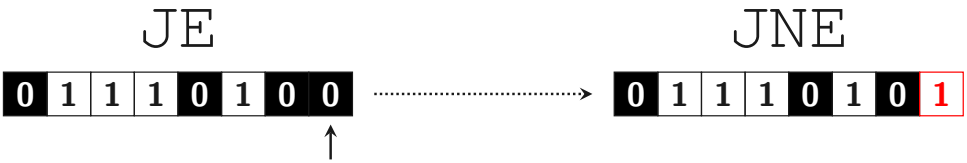

JE
0 1 1 1 0 1 0 0
PUSHQ
0 1 0 1 0 1 0 0


JE
0 1 1 1 0 1 0 0
<prefix>
0 1 1 0 0 1 0 0


JE
0 1 1 1 0 1 0 0
JL
0 1 1 1 1 1 0 0


JE
0 1 1 1 0 1 0 0
JO
0 1 1 1 0 0 0 0


JE
0 1 1 1 0 1 0 0
JBE
0 1 1 1 0 1 1 0

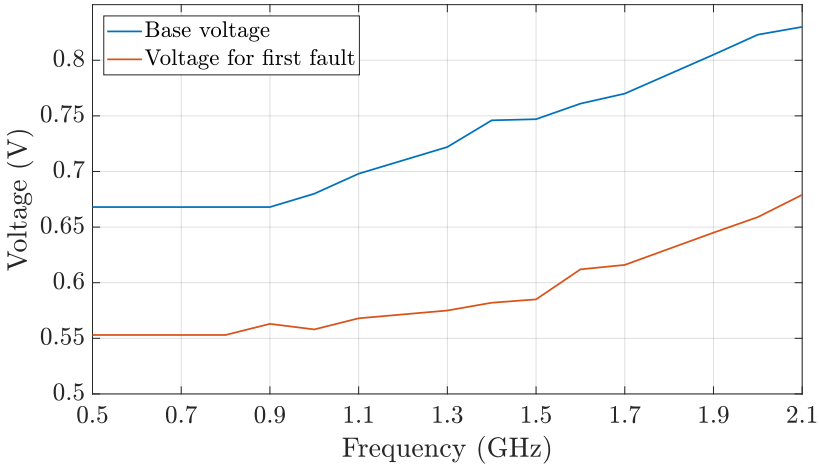

JE
0 1 1 1 0 1 0 0
JNE
0 1 1 1 0 1 0 1

Will it fault?

```
uint64_t multiplier = 0x1122334455667788;
uint64_t correct    = 0xdeadbeef * multiplier;
uint64_t var        = 0xdeadbeef * multiplier;

while (var == correct)
{
    var = 0xdeadbeef * multiplier;
}
uint64_t flipped_bits = var ^ correct;
```

Safe voltages



```
do
{
        i++;
        plaintext = <randomly generated>

        result1 = aes128_enc(plaintext);
        result2 = aes128_enc(plaintext);
} while (vec_equal_128(result1,result2) && i<iterations);
```

- Should be related to undervolting

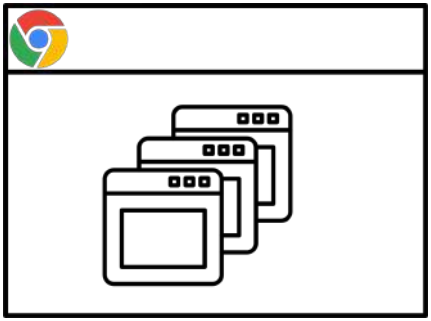
- Should be related to undervolting
- From protected TEE vaults

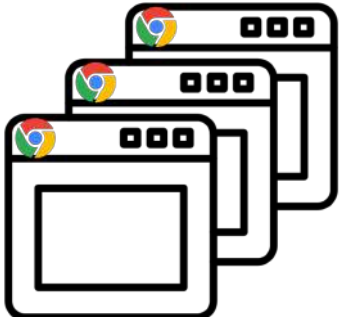
- Should be related to undervolting
- From protected TEE vaults
- Steal

- Should be related to undervolting
- From protected TEE vaults
- Steal, corrupt

- Should be related to undervolting
- From protected TEE vaults
- Steal, corrupt, plunder, ...


PLUNDER
VOLT





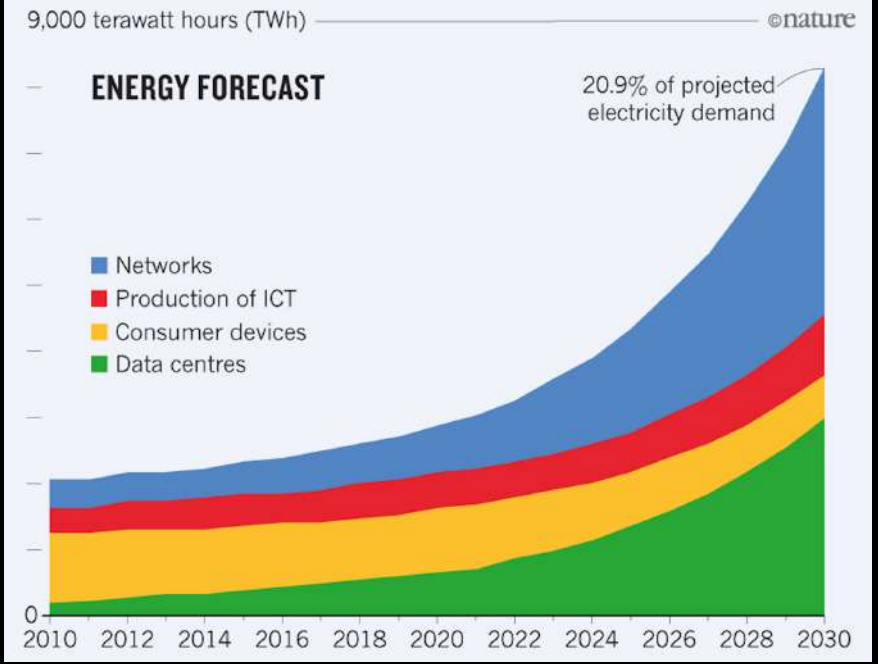

+8.2%


+8.2%


+8.2%


+8.2%


+5%


9,000 terawatt hours (TWh)
©nature
ENERGY FORECAST
20.9% of projected electricity demand
Networks
Production of ICT
Consumer devices
Data centres
0
2010
2012
2014
2016
2018
2020
2022
2024
2026
2028
2030

0.09%

0.40%

There are alternatives

There are alternatives to security!

How expensive is security?

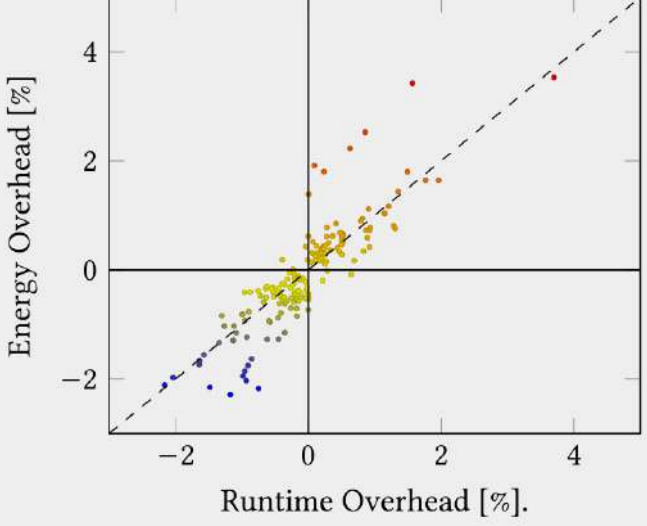


Figure 3: Energy overhead and runtime overhead scatter plot of benchmark runs from 108 Linux kernel CVE fixes.

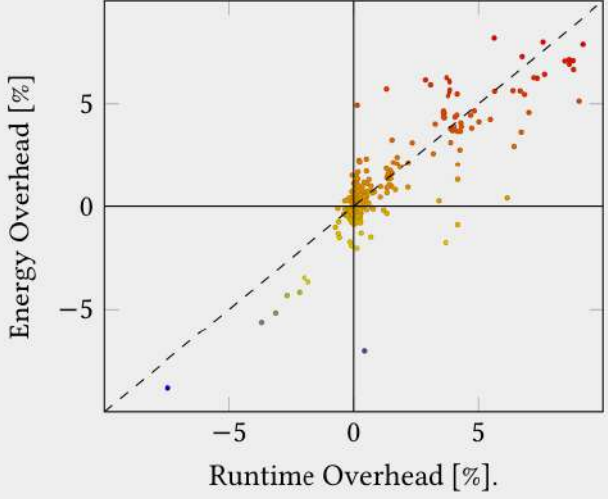


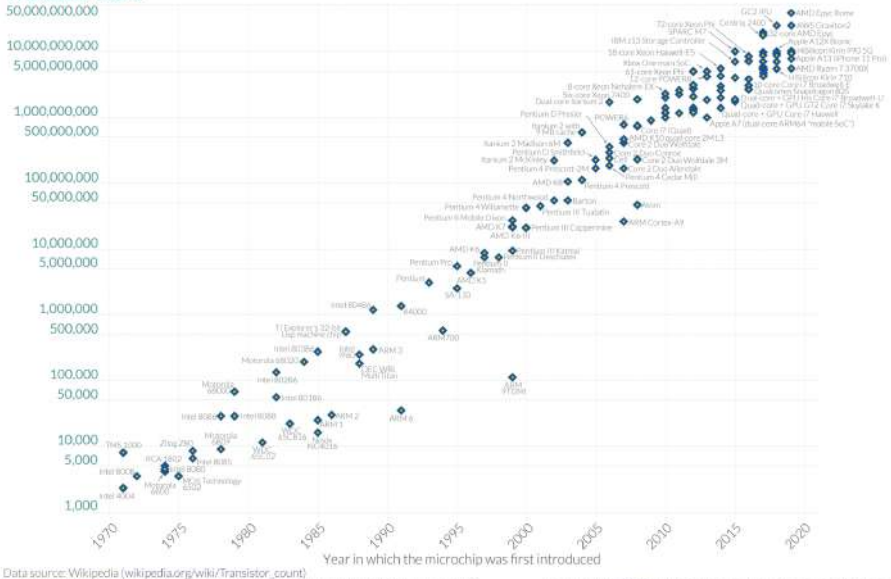
Figure 5: Linux kernel hardware mitigation energy and runtime overhead zoomed-in with an x-range of $[-10, 10]$ and a y-range of $[-10, 10]$.


RACE FOR A VACCINE
WE WANT TO BE FIRST
WE WANT TO BE FIRST!
WE WANT TO BE FIRST!
WE WANT TO BE FIRST!
RACE TO ACT ON CLIMATE
YOU GO FIRST!
WHY SHOULD WE BE FIRST
NO, YOU GO FIRST
THEY SHOULD GO FIRST!


FUNCTIONALITY
SECURITY
imgflip.com

Moore's Law: The number of transistors on microchips has doubled every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)


THE END IS NEAR
OF MOORE'S LAW

Can we make things more efficient by adding security?

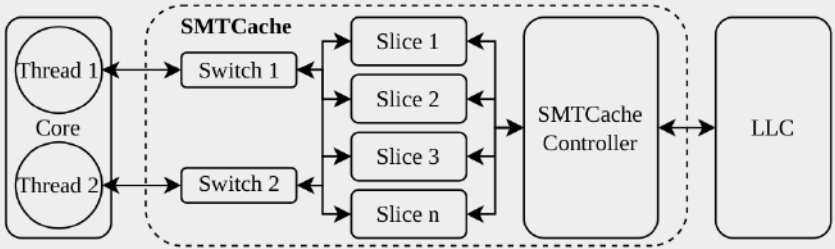
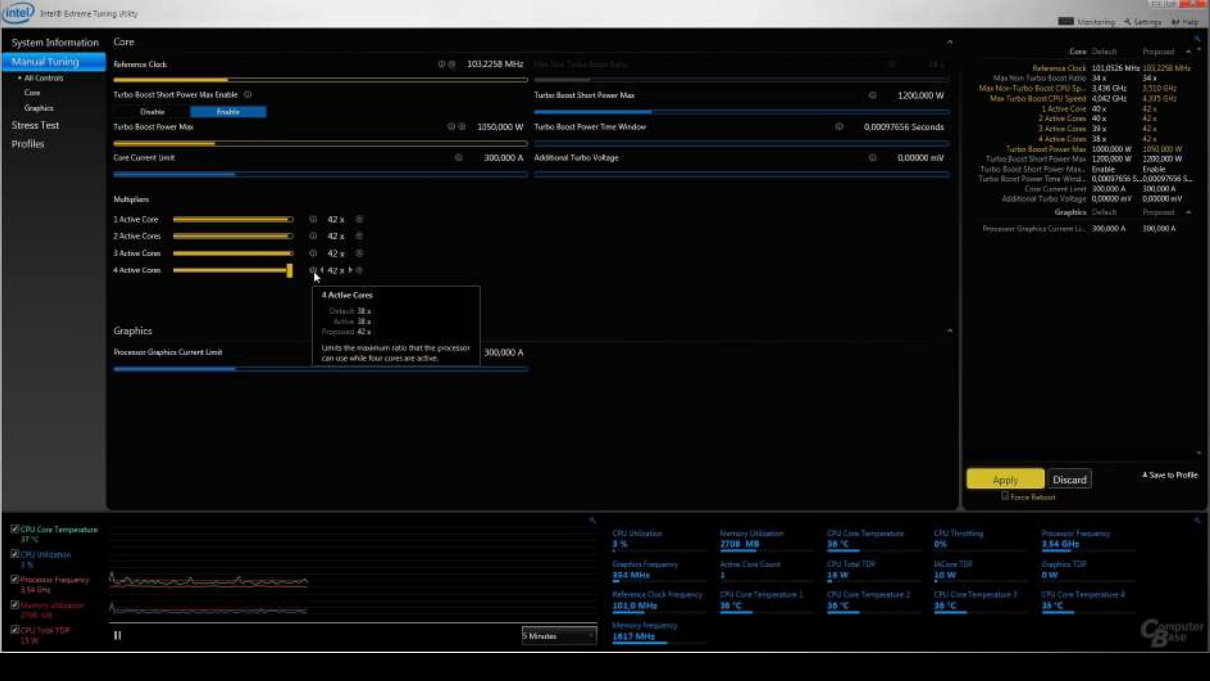


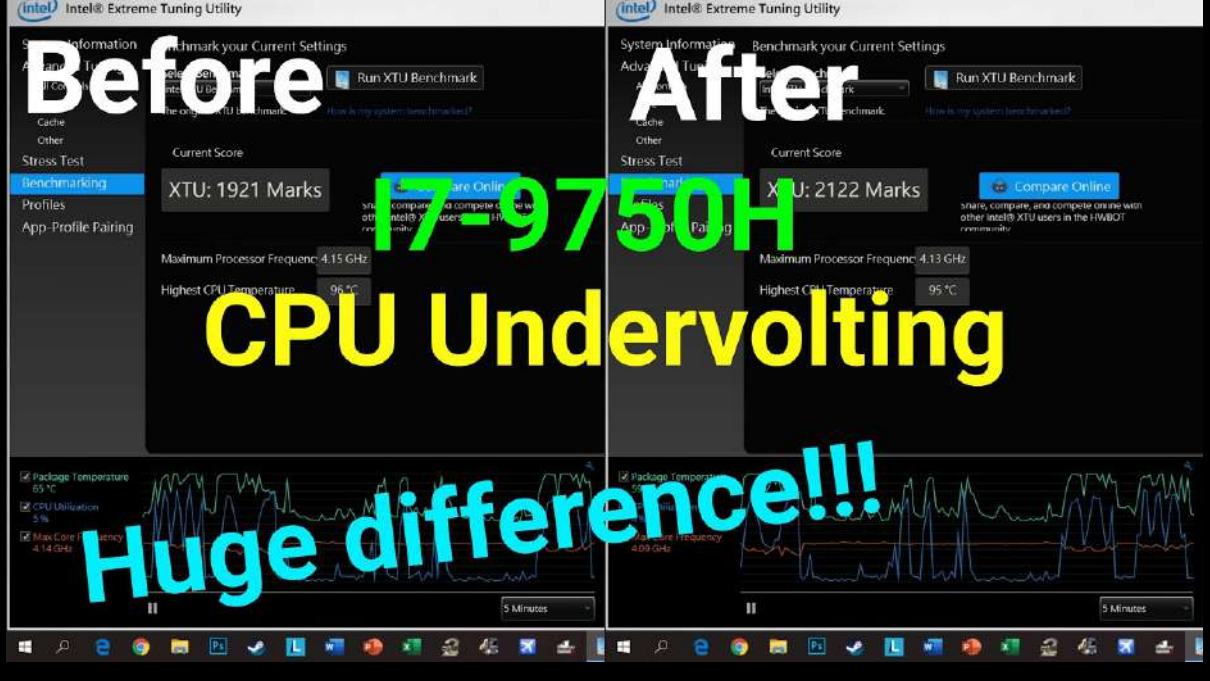
Fig. 1: SMTCache abstract design for n slices. At most 2 slices are active at the same time, one per SMT thread. The SMTCache controller ensures coherence between slices and that SMTCache appears like a normal cache to higher cache levels.

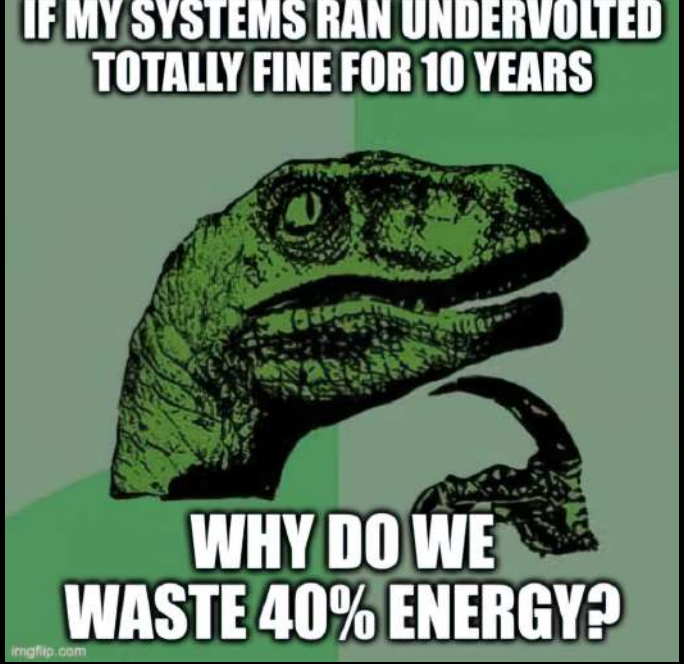
Table 1: Area and power overheads estimated with McPAT [26] and CACTI [37].

Number of Ways	8-way	12-way	16-way	2 slices (8-way)	24-way	3 slices (8-way)	32-way	4 slices (8-way)	40-way	5 slices (8-way)
Total L1 Cache Size	32 KiB	48 KiB	64 KiB	64 KiB	96 KiB	96 KiB	128 KiB	128 KiB	160 KiB	160 KiB
Number of SMT Cores[†]	1	1	2	2	2	2	4	4	4	4
Bus Area [mm^2]	0.37	0.37	0.38	0.38	0.39	0.40	0.41	0.42	0.53	0.54
Bus Peak Dynamic [W]	2.04	2.08	2.11	2.13	2.15	2.13	2.27	2.14	2.41	2.45
Bus Subthreshold Leakage [W]	0.05	0.05	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
Bus Runtime Dynamic [W]	2.04	2.08	2.11	2.13	2.15	2.13	2.27	2.14	2.41	2.45
L1 Dynamic read energy [nJ]	2.53	6.87	11.22	2.53	29.16	2.53	47.09	2.53	81.55	2.53
L1 Dynamic write energy [nJ]	2.58	7.01	11.45	2.58	29.73	2.58	48.01	2.58	82.72	2.58
L1 Standby Leakage [mW]	42.50	64.41	86.33	85.83	132.47	127.49	178.61	169.98	276.95	212.48
L1 Area [mm^2]	3.77	9.26	14.75	7.63	36.52	11.32	58.30	15.10	100.54	18.87
L1 Max. Total Leak. (2 loads+stores/cycle) [W]	10.26	27.84	45.42	10.30	117.90	10.34	190.38	10.38	328.83	10.43
L1 Max. Total Leak. (4 loads+stores/cycle) [W]	-	-	90.77	20.52	235.70	20.56	380.63	20.60	657.47	20.64
L1 Max. Total Leak. (8 loads+stores/cycle) [W]	-	-	-	-	-	-	761.04	41.02	1314.59	41.07

[†] For a fair comparison, we adjusted the number of SMT cores to reflect the L1 cache sizes: 1 SMT core below 64 KiB, 2 SMT cores for the 64 KiB to 96 KiB range, and 4 SMT cores above. We simulate the results for 3 different configurations for the load and store ports from 2 to 8 loads and store per cycle. The maximum total leakage significantly changes with the number of SMT cores and the number of loads and stores per cycle. As the slices of SMTCache act as independent caches, they scale almost linearly in the maximum total leakage.


Intel® Extreme Tuning Utility
Monitoring
Settings
Help
System Information
Manual Tuning
• All Controls
Core
Graphics
Stress Test
Profiles
Core
Reference Clock
103.2258 MHz
Turbo Boost Short Power Max Enable
Disable
Enable
Turbo Boost Short Power Max
1200,000 W
Turbo Boost Power Max
1050,000 W
Turbo Boost Power Time Window
0,00097656 Seconds
Core Current Limit
300,000 A
Additional Turbo Voltage
0,00000 mV
Multipliers
1 Active Core
42 x
2 Active Cores
42 x
3 Active Cores
42 x
4 Active Cores
42 x
4 Active Cores
Default 38 x
Active 38 x
Proposed 42 x
Limits the maximum ratio that the processor can use while four cores are active.
Graphics
Processor Graphics Current Limit
300,000 A
Core Default Proposed
Reference Clock 101,0526 MHz 103,2258 MHz
Max Non-Turbo Boost Ratio 34 x 34 x
Max Non-Turbo Boost CPU Sp... 3,436 GHz 3,510 GHz
Max Turbo Boost CPU Speed 4,042 GHz 4,935 GHz
1 Active Core 40 x 42 x
2 Active Cores 40 x 42 x
3 Active Cores 39 x 42 x
4 Active Cores 38 x 42 x
Turbo Boost Power Max 1000,000 W 1050,000 W
Turbo Boost Short Power Max 1200,000 W 1200,000 W
Turbo Boost Short Power Max... Enable Enable
Turbo Boost Power Time Wind... 0,00097656 S... 0,00097656 S...
Core Current Limit 300,000 A 300,000 A
Additional Turbo Voltage 0,00000 mV 0,00000 mV
Graphics Default Proposed
Processor Graphics Current Li... 300,000 A 300,000 A
Apply
Discard
Force Reboot
Save to Profile
CPU Core Temperature
37 °C
CPU Utilization
3 %
Processor Frequency
3,54 GHz
Memory Utilization
2708 MB
Total CPU Power
16 W
5 Minutes
CPU Utilization
3 %
Memory Utilization
2708 MB
CPU Core Temperature
36 °C
CPU Throttling
0%
Processor Frequency
3,54 GHz
Graphics Frequency
354 MHz
Active Core Count
1
CPU Total TDP
16 W
IACore TDP
10 W
Graphics TDP
0 W
Reference Clock Frequency
101,0 MHz
CPU Core Temperature 1
36 °C
CPU Core Temperature 2
36 °C
CPU Core Temperature 3
36 °C
CPU Core Temperature 4
36 °C
Memory Frequency
1617 MHz


Before
After
Intel® Extreme Tuning Utility
Run XTU Benchmark
Current Score
XTU: 1921 Marks
XTU: 2122 Marks
Compare Online
Maximum Processor Frequency 4.15 GHz
Highest CPU Temperature 96 °C
Maximum Processor Frequency 4.13 GHz
Highest CPU Temperature 95 °C
I7-9750H
CPU Undervolting
Huge difference!!!
5 Minutes

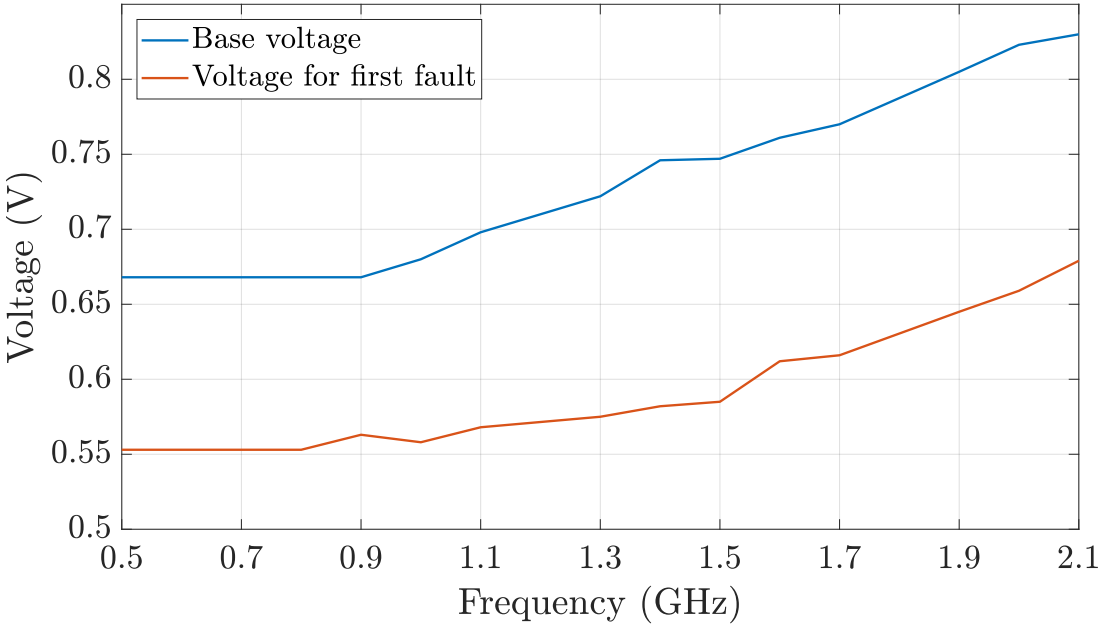

IF MY SYSTEMS RAN UNDERVOLTED
TOTALLY FINE FOR 10 YEARS
WHY DO WE
WASTE 40% ENERGY?
imgflip.com


PLUNDER
VOLT

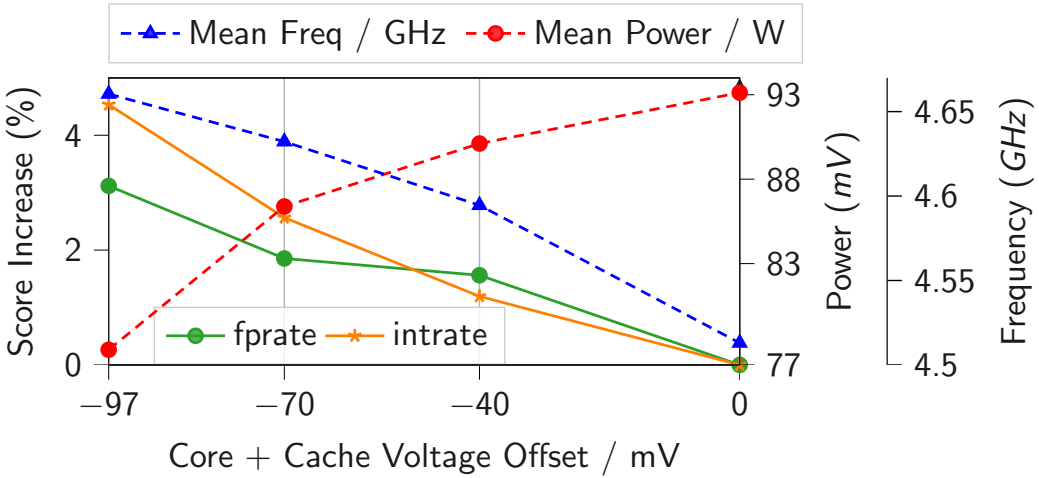
Will it fault?

```
uint64_t multiplier = 0x1122334455667788;
uint64_t correct    = 0xdeadbeef * multiplier;
uint64_t var        = 0xdeadbeef * multiplier;

while (var == correct)
{
    var = 0xdeadbeef * multiplier;
}
uint64_t flipped_bits = var ^ correct;
```

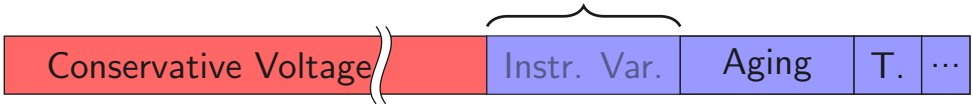

Base voltage
Voltage for first fault
Voltage (V)
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.5
0.7
0.9
1.1
1.3
1.5
1.7
1.9
2.1
Frequency (GHz)

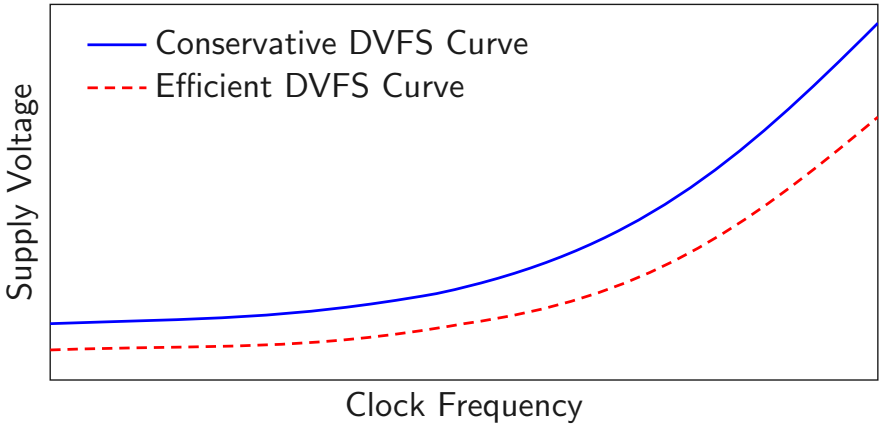
Can we make this secure?

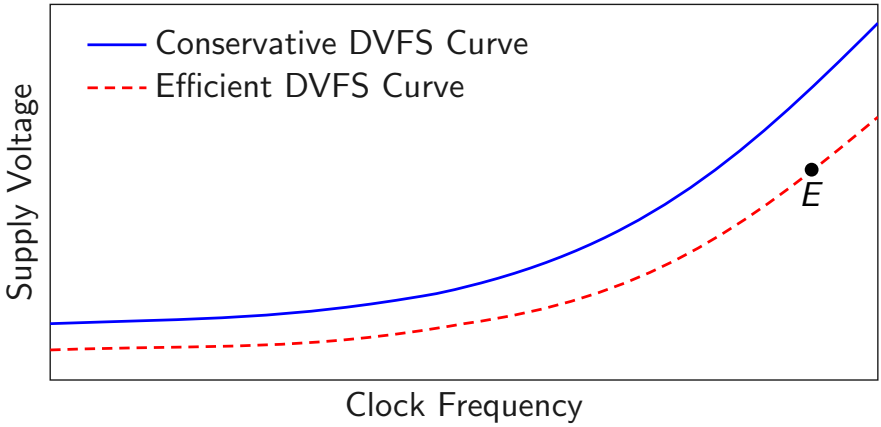

Mean Freq / GHz
Mean Power / W
Score Increase (%)
4
2
0
93
88
83
77
Power (mV)
4.65
4.6
4.55
4.5
Frequency (GHz)
fprate
intrate
−97
−70
−40
0
Core + Cache Voltage Offset / mV

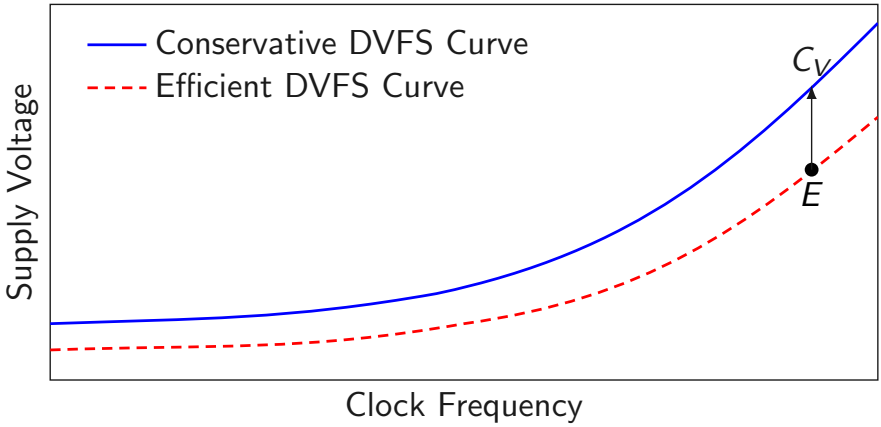

Conservative Voltage
Instr. Var.
Aging
T.
...

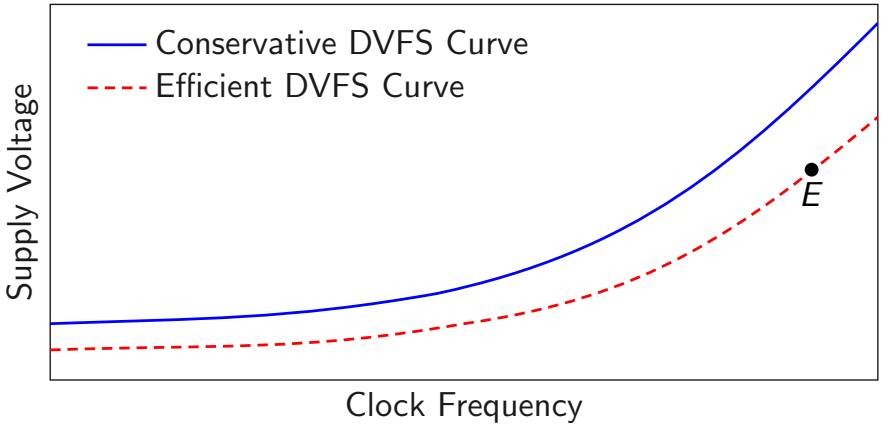
Up to a 150 mV variation in instruction voltage requirement.

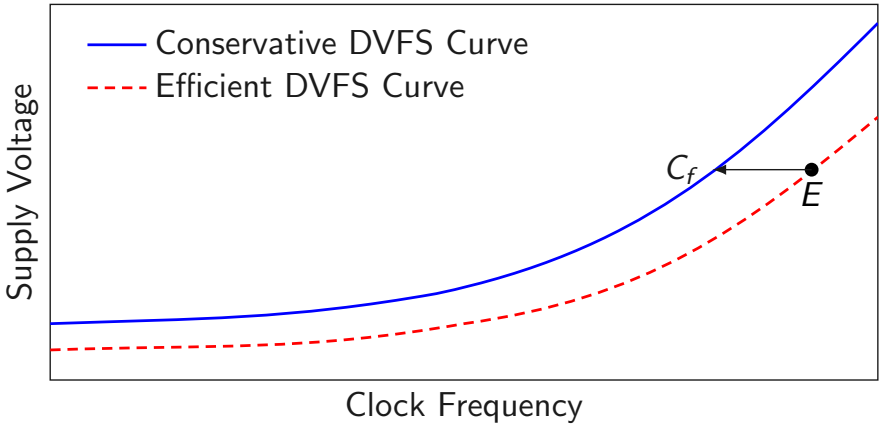


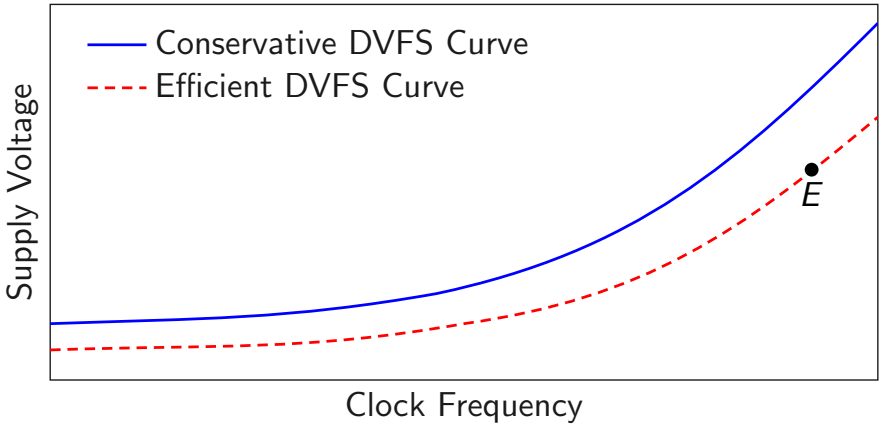

Conservative DVFS Curve
Efficient DVFS Curve
Supply Voltage
Clock Frequency

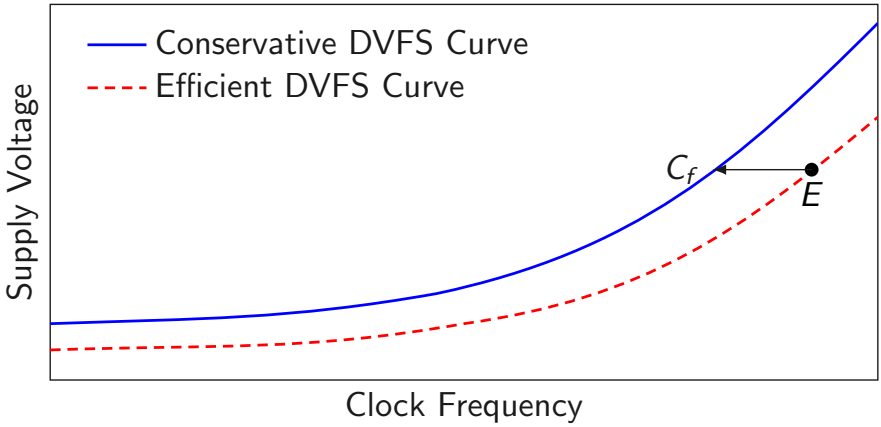

Conservative DVFS Curve
Efficient DVFS Curve
Supply Voltage
Clock Frequency
E

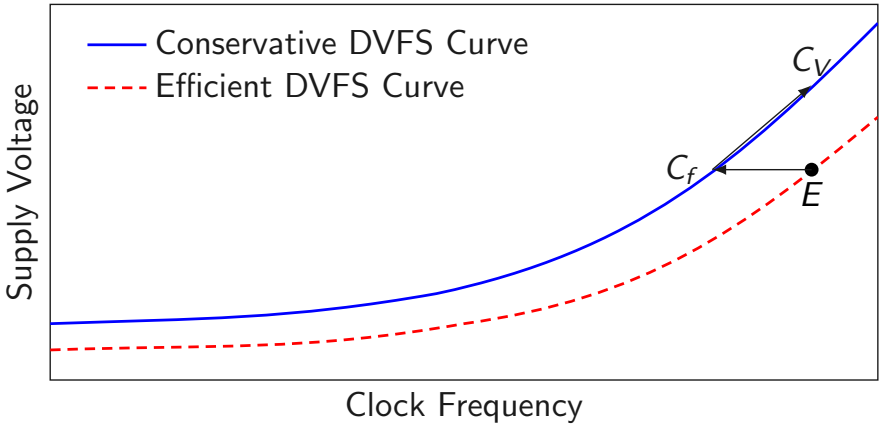

Conservative DVFS Curve
Efficient DVFS Curve
C_V
E
Supply Voltage
Clock Frequency


Conservative DVFS Curve
Efficient DVFS Curve
Supply Voltage
Clock Frequency
E


Conservative DVFS Curve
Efficient DVFS Curve
Supply Voltage
Clock Frequency
C_f
E


Conservative DVFS Curve
Efficient DVFS Curve
Supply Voltage
Clock Frequency
E

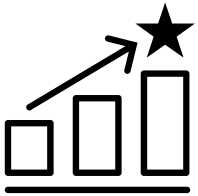

Conservative DVFS Curve
Efficient DVFS Curve
Supply Voltage
Clock Frequency
C_f
E


Conservative DVFS Curve
Efficient DVFS Curve
C_V
C_f
E
Supply Voltage
Clock Frequency

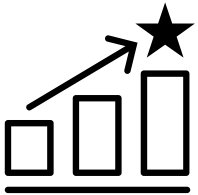
CPU	V_{off}	Score	Power	Freq.	Energy Eff.
i5-1035G1	−70 mV	+6.0 %	−0.1 %	+8.5 %	+6.1 %
	−97 mV	+7.9 %	−0.5 %	+12 %	+8.4 %
i9-9900K	−70 mV	+2.2 %	−7.2 %	+2.6 %	+10 %
	−97 mV	+3.8 %	−16 %	+3.3 %	+23 %
7700X*	−70 mV	+1.4 %	−9.8 %	+1.8 %	+12 %
	−97 mV	+1.9 %	−15 %	+1.8 %	+20 %

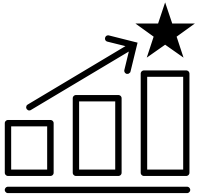
Why are problems like Rowhammer not solved already?

Common misunderstandings...



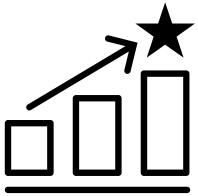
... create bad incentives.





... create bad incentives.

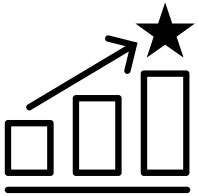
- A “bit” more reliability



... create bad incentives.

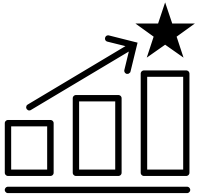
- A “bit” more reliability
- Why not higher or dynamic refresh rates everywhere (e.g. TRR, PARA, ...)?

Common misunderstandings...



... create bad incentives.

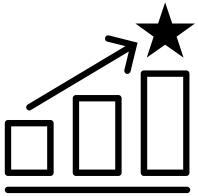
- A "bit" more reliability
- Why not higher or dynamic refresh rates everywhere (e.g. TRR, PARA, ...)?
  - "just a few more targeted refreshes"



... create bad incentives.

- A "bit" more reliability
- Why not higher or dynamic refresh rates everywhere (e.g. TRR, PARA, ...)?
  - "just a few more targeted refreshes"
- Why not ECC everywhere?

Common misunderstandings...

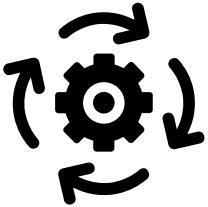


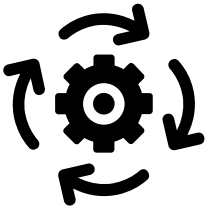
... create bad incentives.

- A "bit" more reliability
- Why not higher or dynamic refresh rates everywhere (e.g. TRR, PARA, ...)?
  - "just a few more targeted refreshes"
- Why not ECC everywhere?

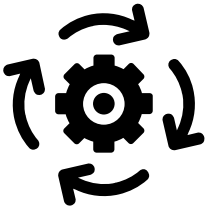
$\rightarrow$ What incentives does it create?

Optimizing systems...



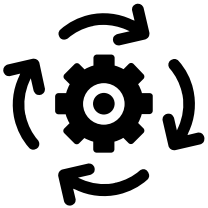


Fundamental problem: we assume what is still reliable



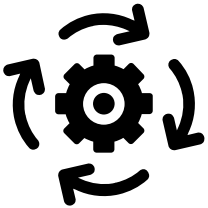
Fundamental problem: we assume what is still reliable

- Refreshing x times per second is fine



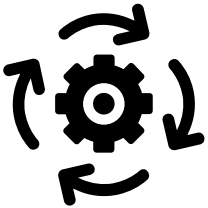
Fundamental problem: we assume what is still reliable

- Refreshing x times per second is fine
- Normal usage, no adversary



Fundamental problem: we assume what is still reliable

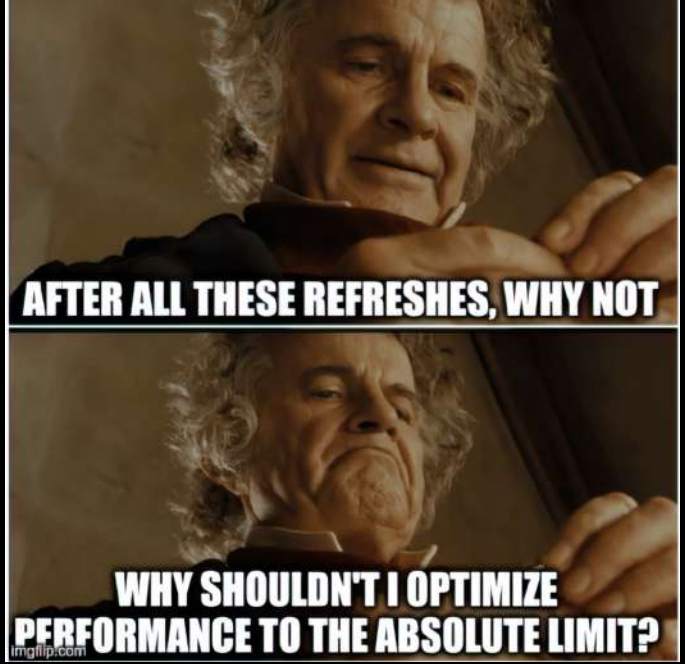
- Refreshing x times per second is fine
- Normal usage, no adversary
- Assume there won't be more than n bit errors



Fundamental problem: we assume what is still reliable

- Refreshing x times per second is fine
- Normal usage, no adversary
- Assume there won't be more than n bit errors

$\rightarrow$ How far can we go with x while staying below n bit errors?


AFTER ALL THESE REFRESHES, WHY NOT
WHY SHOULDN'T I OPTIMIZE PERFORMANCE TO THE ABSOLUTE LIMIT?
imgflip.com

Mobile vendors since 2018: let's add ECC by default

Mobile vendors since 2018: let's add ECC by default

- ECC memory $\rightarrow$ fewer bit flips + more security

Mobile vendors since 2018: let's add ECC by default

- ECC memory $\rightarrow$ fewer bit flips + more security

Also vendors:

Mobile vendors since 2018: let's add ECC by default

- ECC memory $\rightarrow$ fewer bit flips + more security

Also vendors:

- Let's squeeze out the last bit of efficiency for battery runtime until just before bit flips occur

It's an optimization problem

It's an optimization problem

It's an optimization problem

- You never know how far is still safe

It's an optimization problem

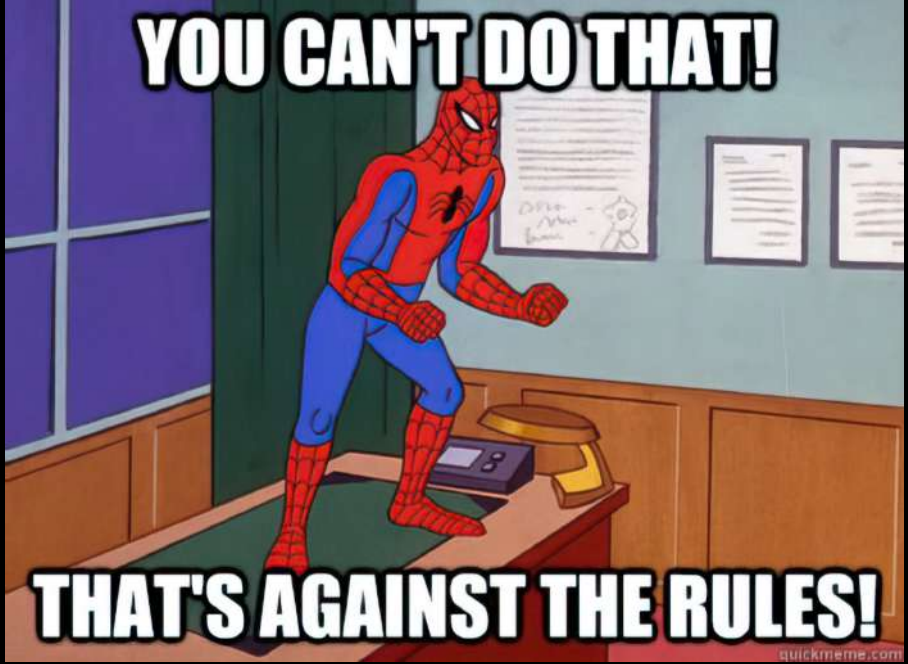
- You never know how far is still safe
- "safe" / "reliable" changes over time

It's an optimization problem

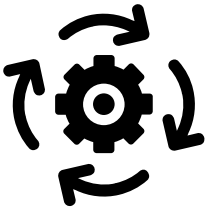
- You never know how far is still safe
- "safe" / "reliable" changes over time
- Adversary is intelligent and improves attacks over time

Security vs Reliability

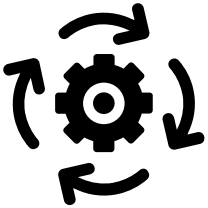
Security ~~vs Reliability~~


YOU CAN'T DO THAT!
THAT'S AGAINST THE RULES!
quickmeme.com

Security for Efficiency?

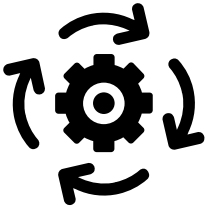


Make bit flips degrade performance **without** impacting security



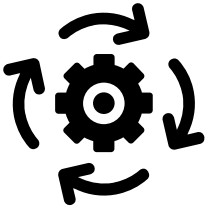
Make bit flips degrade performance **without** impacting security

- Cryptographic MAC



Make bit flips degrade performance **without** impacting security

- Cryptographic MAC
- Detect **any** number of bit flips



Make bit flips degrade performance **without** impacting security

- Cryptographic MAC
- Detect **any** number of bit flips
- Correction by **brute-force** search for correct data

# Errors	# MAC Comp.	Avg Duration
1	17	11 ns
2	771	3.68 µs
3	33 800	124 µs
4	1.51×10^6	6.65 ms
5	6.91×10^7	261 ms
6	3.07×10^9	12.8 s
7	1.21×10^{11}	9.11 min
8	5.72×10^{12}	6.11 h

- Silent data corruption less than once per 10^9 billion years

- Silent data corruption less than once per 10^9 billion years
- Second preimage after hammering for one year: $9.75 \cdot 10^{-5}$ %

- Silent data corruption less than once per 10^9 billion years
- Second preimage after hammering for one year: $9.75 \cdot 10^{-5}\,\%$
- Erroneous correction of 8-bit errors: $0.0161\,\%$

Security:
Can we afford to have it?
Can we afford not to have it?

Daniel Gruss

2025-10-26

Graz University of Technology