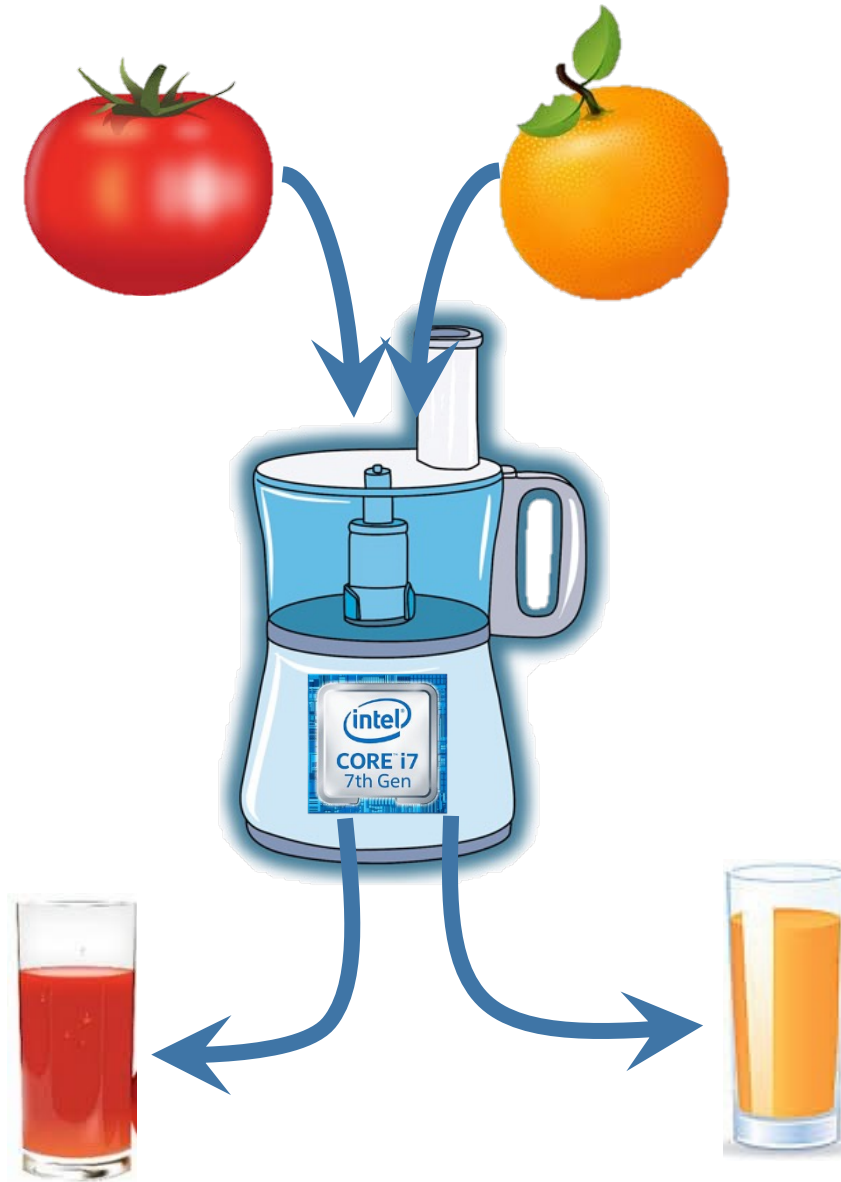


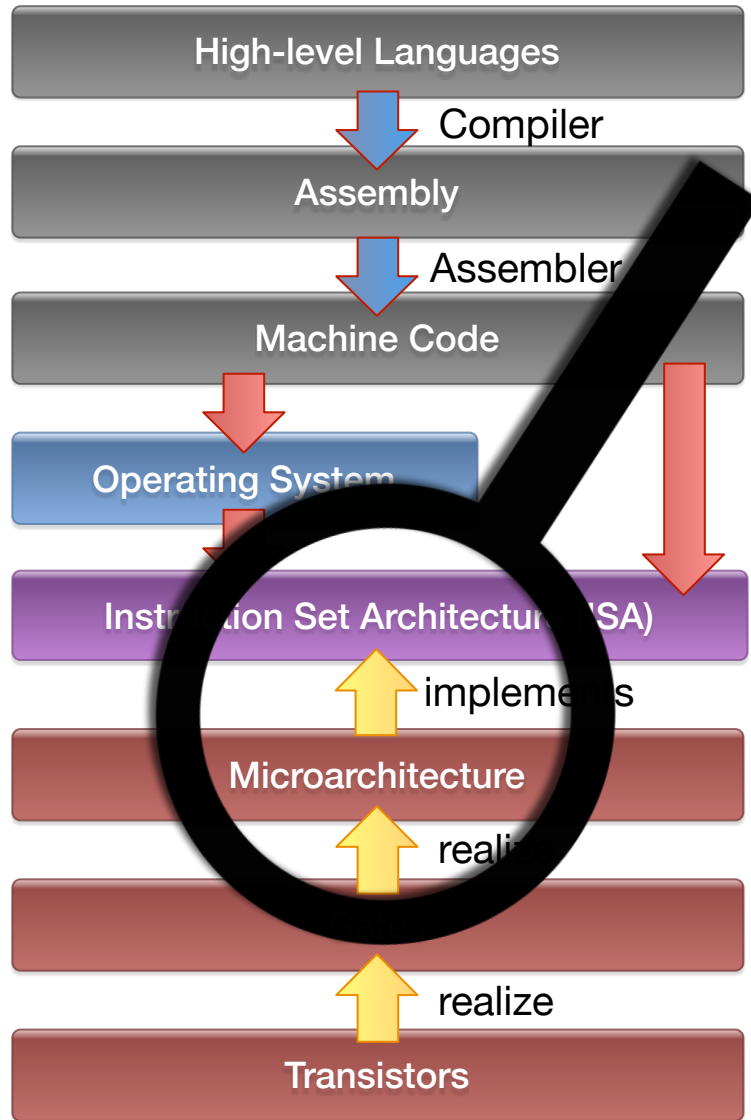
An Introduction to Microarchitectural Attacks

Jan Reineke

Based on slides kindly provided by
Yuval Yarom, ~~The University of Adelaide and Data61~~
Ruhr University Bochum



Abstraction Layers



Hardware
Memory
Execution

Illusion (ISA)

Reality
(mArchitecture)

Dedicated

Shared

Uniform

Non-uniform

Serial

Superscalar

(In)Security lives and breathes in the cracks between abstraction layers.

Thomas Dullien (@halvarflake)

CPU vs. Memory



**Processor
Speed**

1 MHz

**Memory
Latency**

500 ns



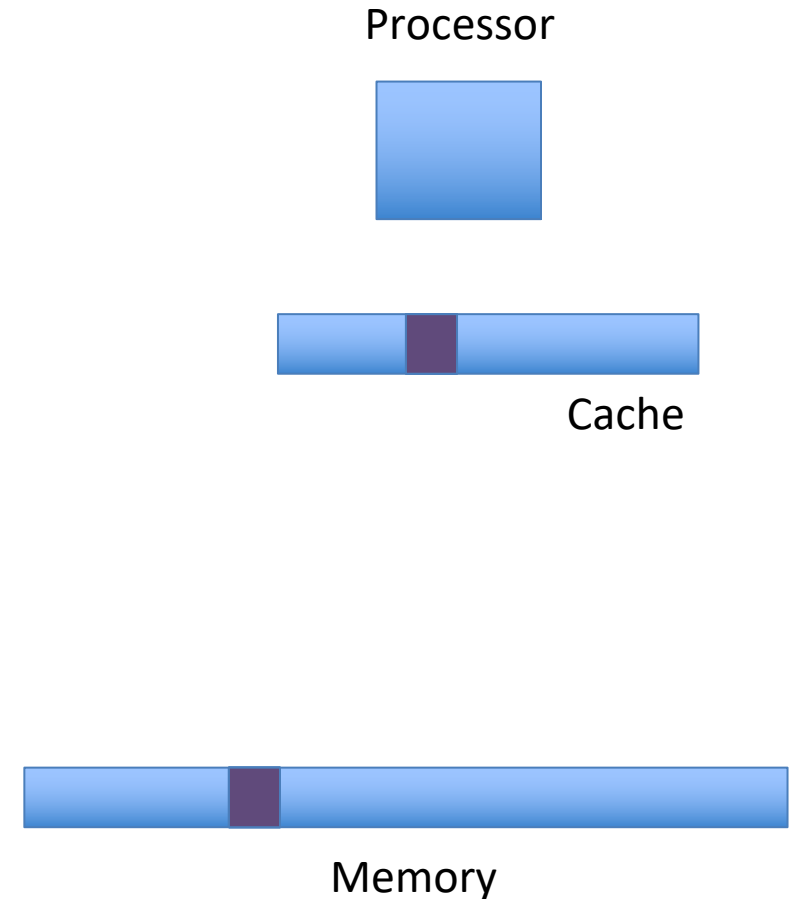
8*2600 MHz

63 ns

Bridging the gap

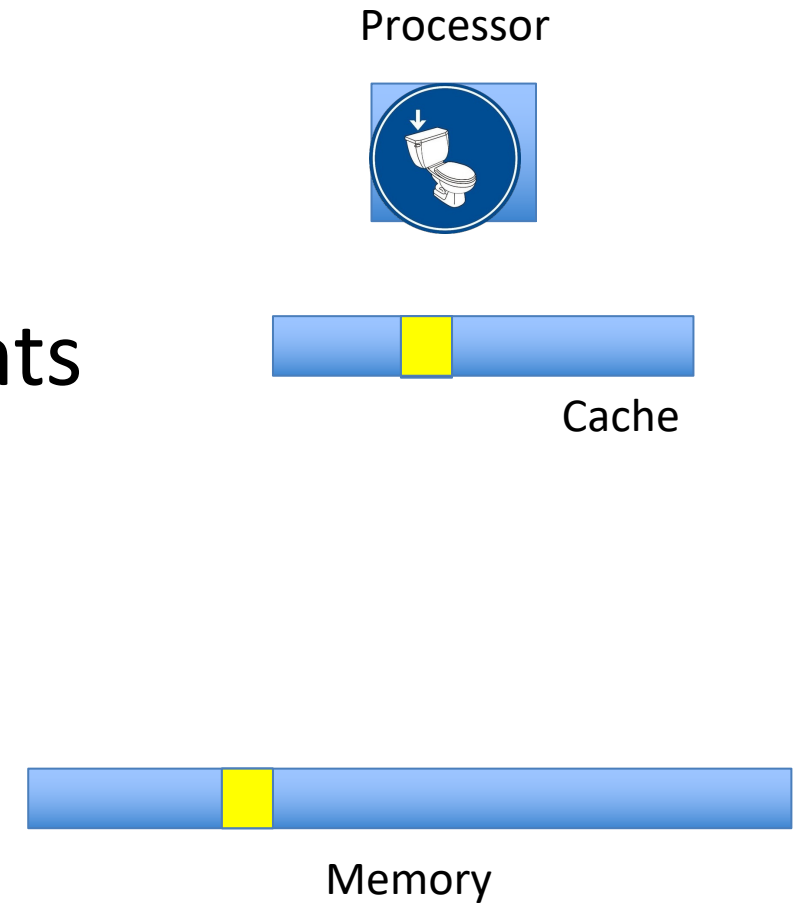
Cache utilises locality to bridge the gap

- Divides memory into *lines*
- Stores recently used lines
- In a *cache hit*, data is retrieved from the cache
- In a *cache miss*, data is retrieved from memory and inserted to the cache



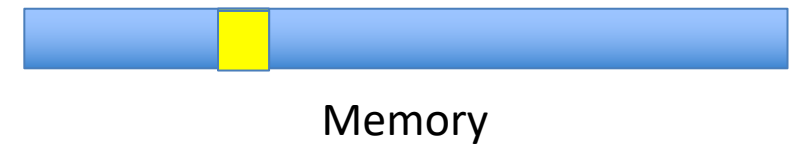
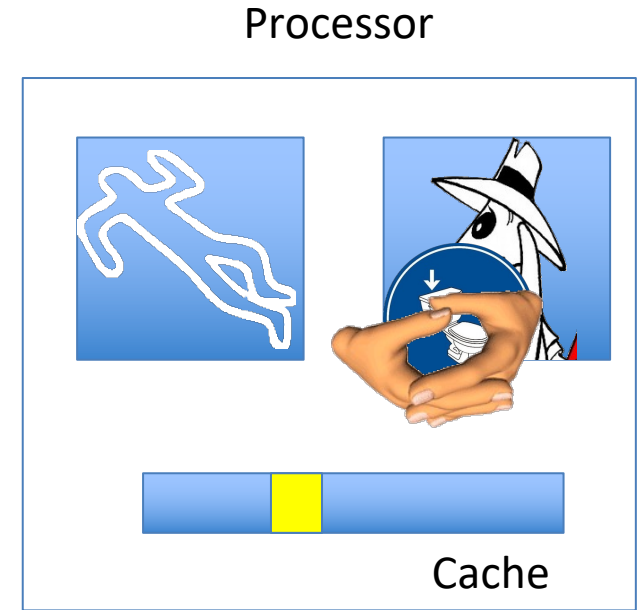
Cache Consistency

- Memory and cache can be in inconsistent states
 - Rare, but possible
- Solution: Flushing the cache contents
 - Ensures that the next load is served from the memory



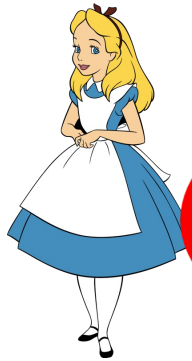
FLUSH+RELOAD [YF14]

- **FLUSH** memory line
- Wait a bit
- Measure time to **RELOAD** line
 - slow-> no access
 - fast-> access
- Repeat



The RSA Encryption System

- The RSA encryption is a public key cryptographic scheme



$$M = C^d \pmod{N}$$

Key Generation:

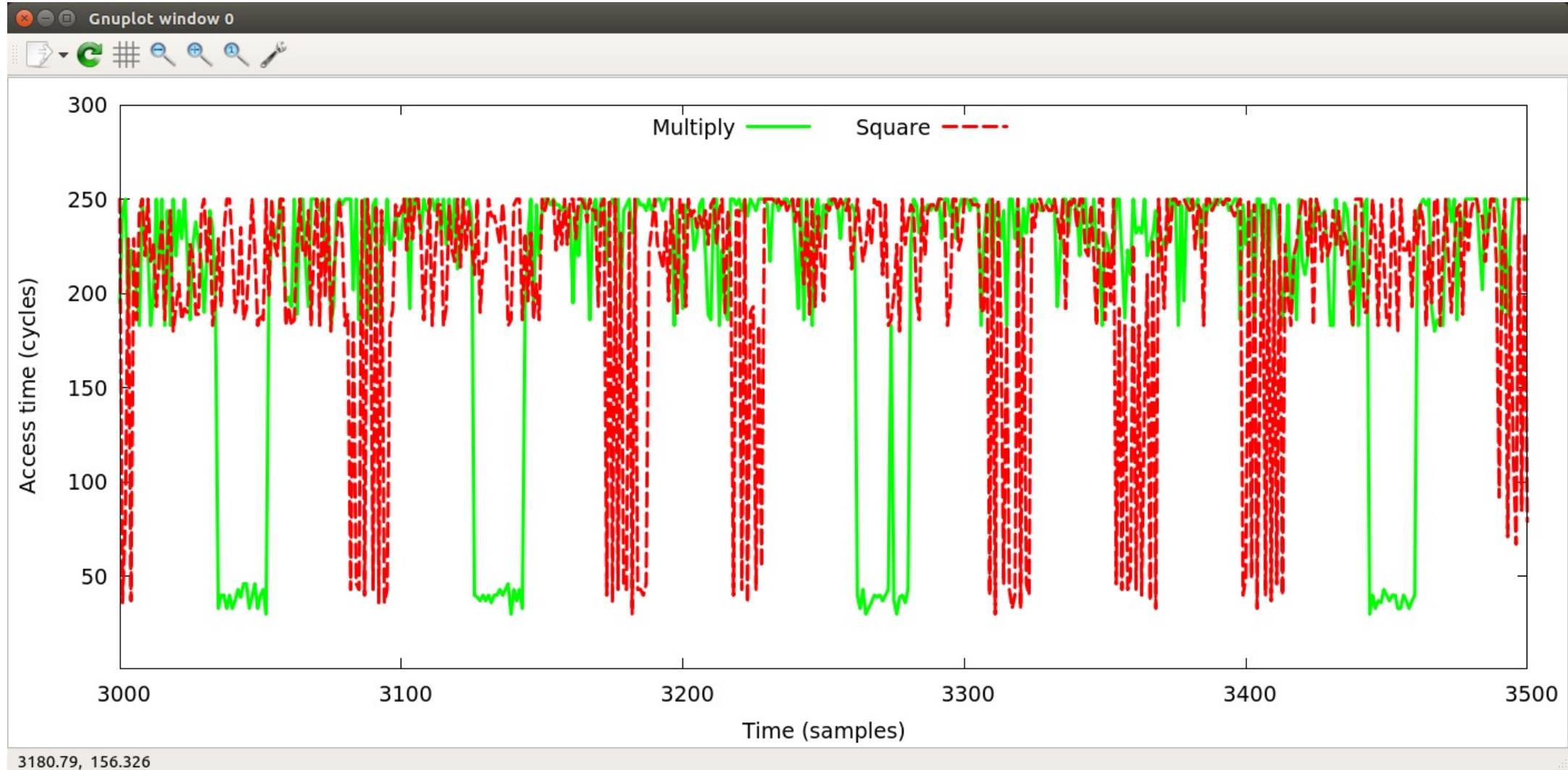
- Select random primes p and q
- Calculate $N = pq$
- Select a public exponent $e (=65537)$
- Compute $d = e^{-1} \pmod{\phi(N)}$
- (N, e) is the public key
- (p, q, d) is the private key

M

$$C = M^e \pmod{N}$$



Flush+Reload on GnuPG 1.4.13

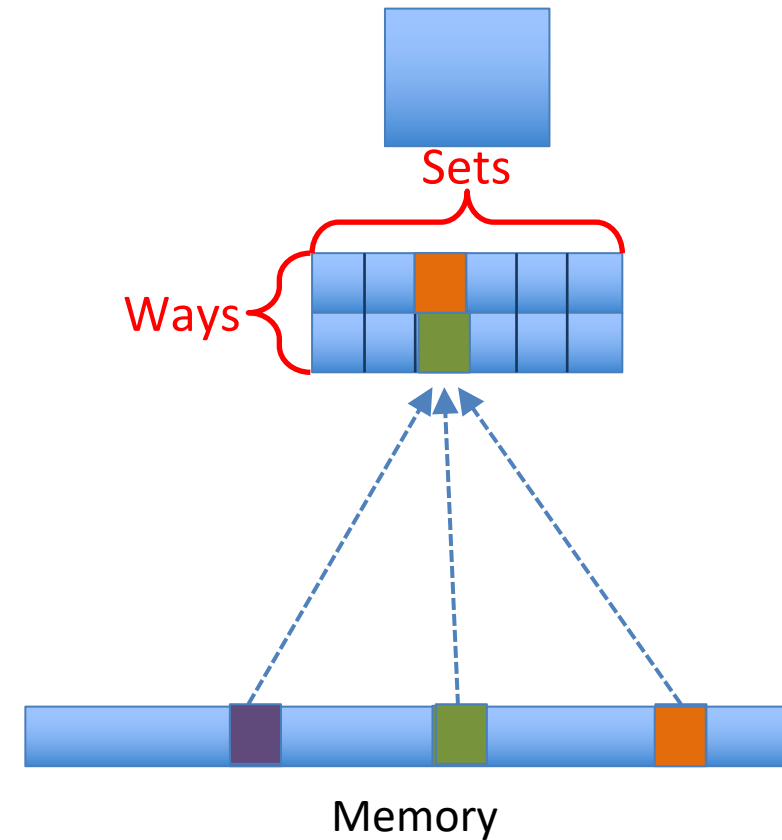


The FLUSH+RELOAD Technique

- Leaks information on victim access to shared memory.
- Spy monitors victim's access to shared code
 - Spy can determine what victim does
 - Spy can infer the data the victim operates on

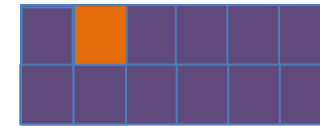
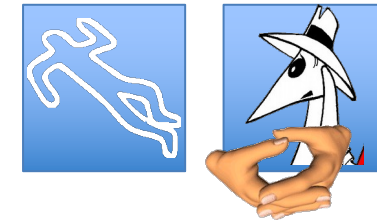
Set Associative Caches

- Memory lines map to *cache sets*. Multiple lines map to the same set.
- Sets consist of *ways*. A memory line can be stored in **any** of the ways of the set it maps to.
- When a cache miss occurs, one of the lines in the set is *evicted*.



The Prime+Probe Attack [OST06]

- Allocate a cache-sized memory buffer
- *Prime:*
fills the cache with the contents of the buffer
- *Probe:*
measure the time to access each cache set
 - Slow access indicates victim access to the set
- The probe phase primes the cache for the next round



Memory

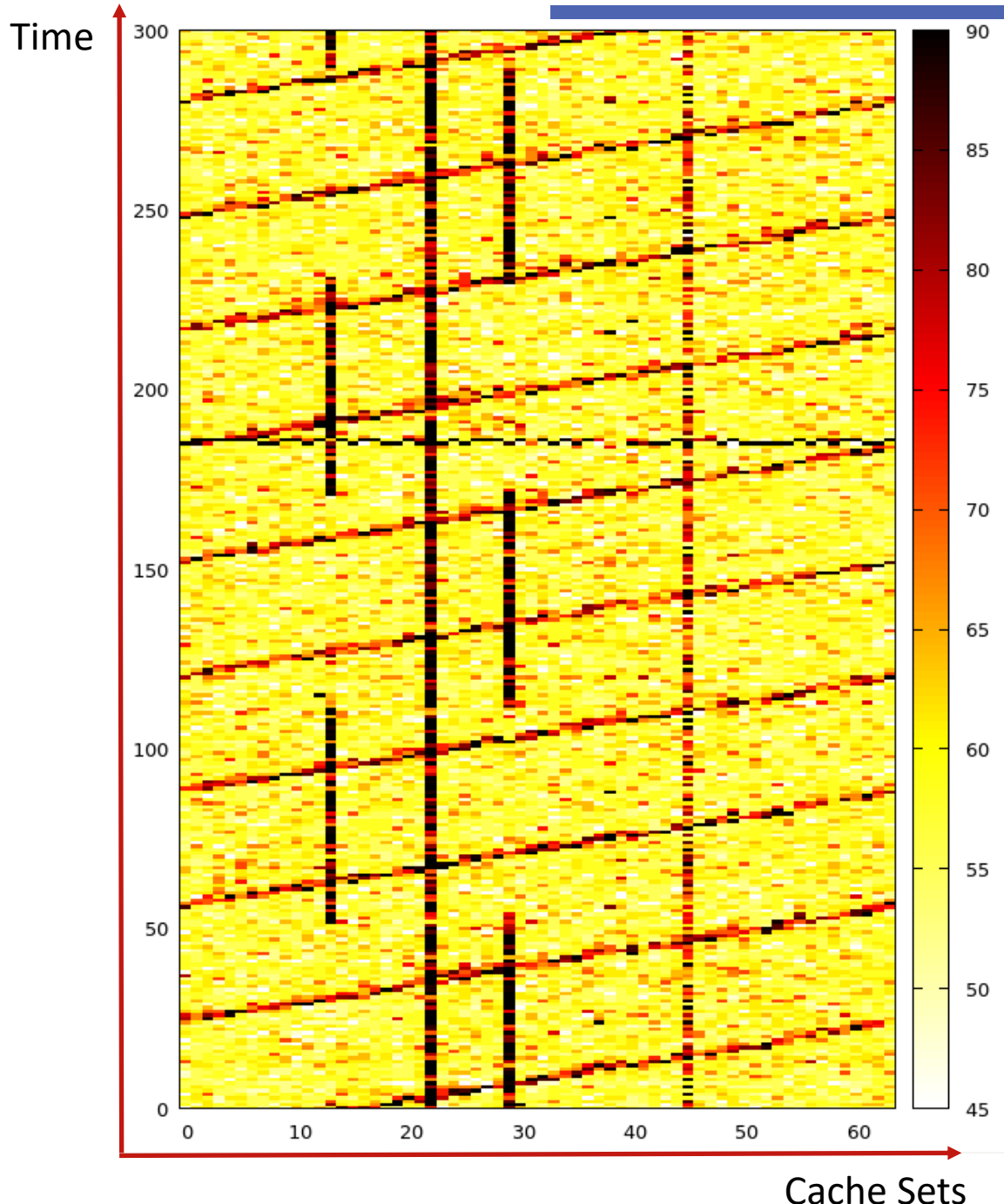
Sample Victim: Data Rattle

```
volatile char buffer[4096];

int main(int ac, char **av) {
    for (;;) {
        for (int i = 0; i < 64000; i++)
            buffer[800] += i;

        for (int i = 0; i < 64000; i++)
            buffer[1800] += i;
    }
}
```

Cache Fingerprint of the Rattle Program



Real Victim – AES

```
s0 = GETU32(in + 0) ^ rk[0];
s1 = GETU32(in + 4) ^ rk[1];
s2 = GETU32(in + 8) ^ rk[2];
s3 = GETU32(in + 12) ^ rk[3];
#ifdef FULL_UNROLL
/* round 1: */
t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[4];
t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[5];
t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[6];
t3 = Te0[s3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[7];
/* round 2: */
s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[8];
s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[9];
s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[10];
s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[11];
/* round 3: */
t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[12];
t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[13];
t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[14];
t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[15];
/* round 4: */
s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[16];
```

```
static const u32 Te0[256] = {
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,
    0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,
    0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfb0f00bU,
    0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,
    0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,
```


AES T-table access

```
static const u32 Te0[256] = {  
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,  
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,  
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,  
    0xe7fefe10U, 0xb5d7d762U, 0x4dabab61U, 0xec76769aU
```

```
s0 = plaintext ^ key  
t0 = Te0[s0>>24]
```

- Assume we know the plaintext and the index (s0>>24)
 - We can recover the most significant byte of the key

AES T-tables and cache lines

```
static const u32 Te0[256] = {  
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,  
    0xffff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U, Cache Line 0  
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,  
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,  
    0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,  
    0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfbfb0f00bU, Cache Line 1  
    0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,  
    0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,  
    0x75b7b7c2U, 0xe1fdfd1cU, 0x3d9393aeU, 0x4c26266aU,  
    0x6c36365aU, 0x7e3f3f41U, 0xf5f7f702U, 0x83cccc4fU, Cache Line 2  
    0x6834345cU, 0x51a5a5f4U, 0xd1e5e534U, 0xf9f1f108U,  
    0xe2717193U, 0xabd8d873U, 0x62313153U, 0x2a15153fU,  
    0x0804040cU, 0x95c7c752U, 0x46232365U, 0x9dc3c35eU,  
    0x30181828U, 0x379696a1U, 0x0a05050fU, 0x2f9a9ab5U, Cache Line 3  
    0x0e070709U, 0x24121236U, 0x1b80809bU, 0xdfe2e23dU,  
    0xcdebeb26U, 0x4e272769U, 0x7fb2b2cdU, 0xea75759fU,  
    0x1209091bU, 0x1d83839eU, 0x582c2c74U, 0x341a1a2eU,  
    0x361b1b2dU, 0xdc6e6eb2U, 0xb45a5aeeU, 0x5ba0a0fbU, Cache Line 4  
    0xa45252f6U, 0x763b3b4dU, 0xb7d6d661U, 0x7db3b3ceU,  
    0x5229297bU, 0xdde3e33eU, 0x5e2f2f71U, 0x13848497U,  
    0xa65353f5U, 0xb9d1d168U, 0x00000000U, 0xc1eded2cU,  
    0x40202060U, 0xe3fcfc1fU, 0x79b1b1c8U, 0xb65b5bedU, Cache Line 5  
    0xd46a6abeU, 0x8dcbc46U, 0x67bebed9U, 0x7239394bU,  
    0x944a4adeU, 0x984c4cd4U, 0xb05858e8U, 0x85cfcf4aU,  
    0xbbd0d06bU, 0xc5efef2aU, 0x4faaaae5U, 0xedfbfb16U,  
    0x864343c5U, 0x9a4d4dd7U, 0x66333355U, 0x11858594U,  
    0x8a4545cfU, 0xe9f9f910U, 0x04020206U, 0xfe7f7f81U,  
    0xa05050f0U, 0x783c3c44U, 0x250f0fb1U, 0x4ba8a8e3U
```

AES T-tables and cache lines

```
static const u32 Te0[256] = {  
0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,  
0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U, Cache Line 0  
0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,  
0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,  
0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,  
0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfb0f0f0bU, Cache Line 1  
0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,  
0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,  
0x75b7b7c2U, 0xe1fdfd1cU, 0x3d9393aeU, 0x4c26266aU,  
0x6c36365aU, 0x7e3f3f41U, 0xf5f7f702U, 0x83cccc4fU, Cache Line 2  
0x6834345cU, 0x51a5a5f4U, 0xd1e5e534U, 0xf9f1f108U,  
0xe2717193U, 0xabd8d873U, 0x62313153U, 0x2a15153fU,  
0x0804040cU, 0x95c7c752U, 0x46232365U, 0x9dc3c35eU,  
0x30181828U, 0x379696a1U, 0x0a05050fU, 0x2f9a9ab5U,
```

- If $0 \leq \text{plaintext}^{\text{key}} < 16$, Cache Line 0 is accessed.
- What if $\text{plaintext}^{\text{key}} \geq 16$?

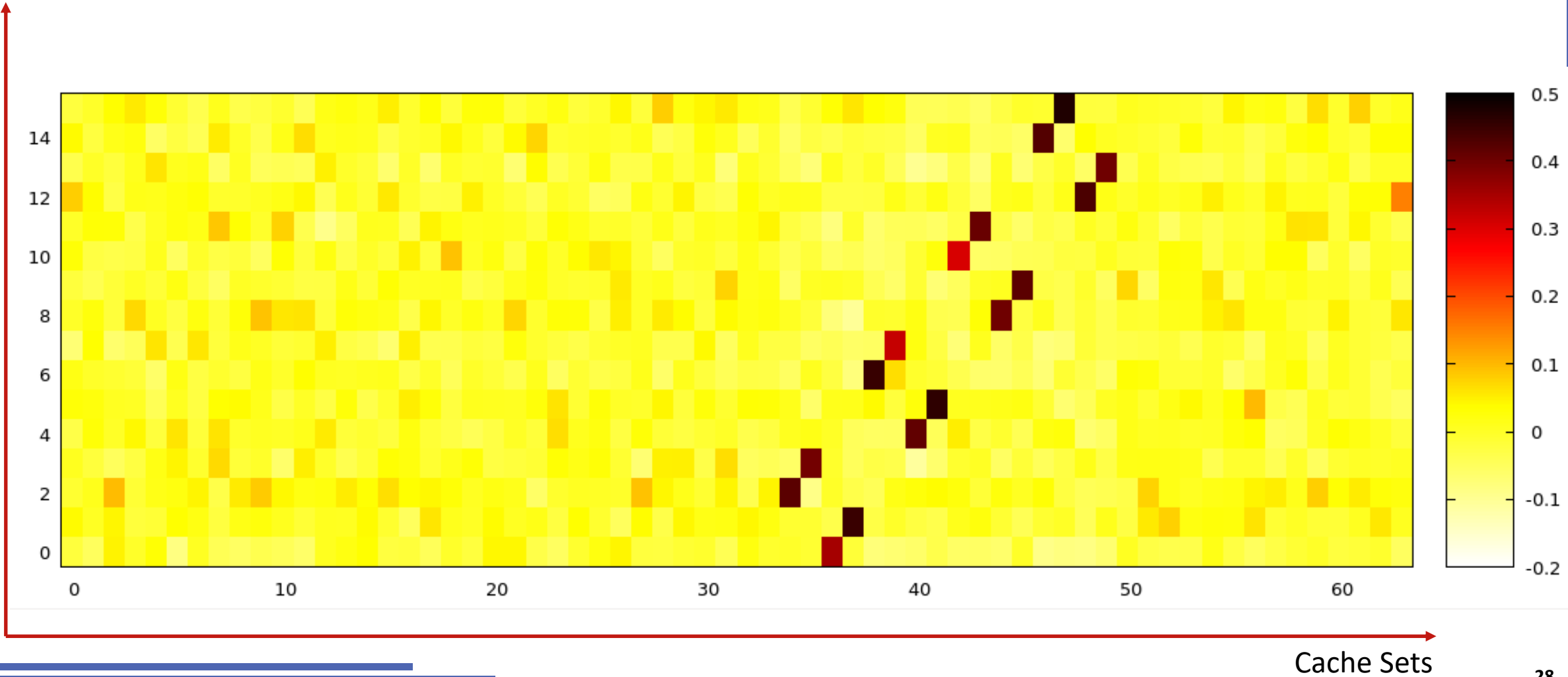
Prime+Probe Attack on AES

```
s0 = plaintext ^ key  
t0 = Te0[s0>>24]
```

- For many plaintexts do: Prime, Encrypt, Probe
- Calculate the average probe time of each cache set as a function of the plaintext value
- Extract key from results

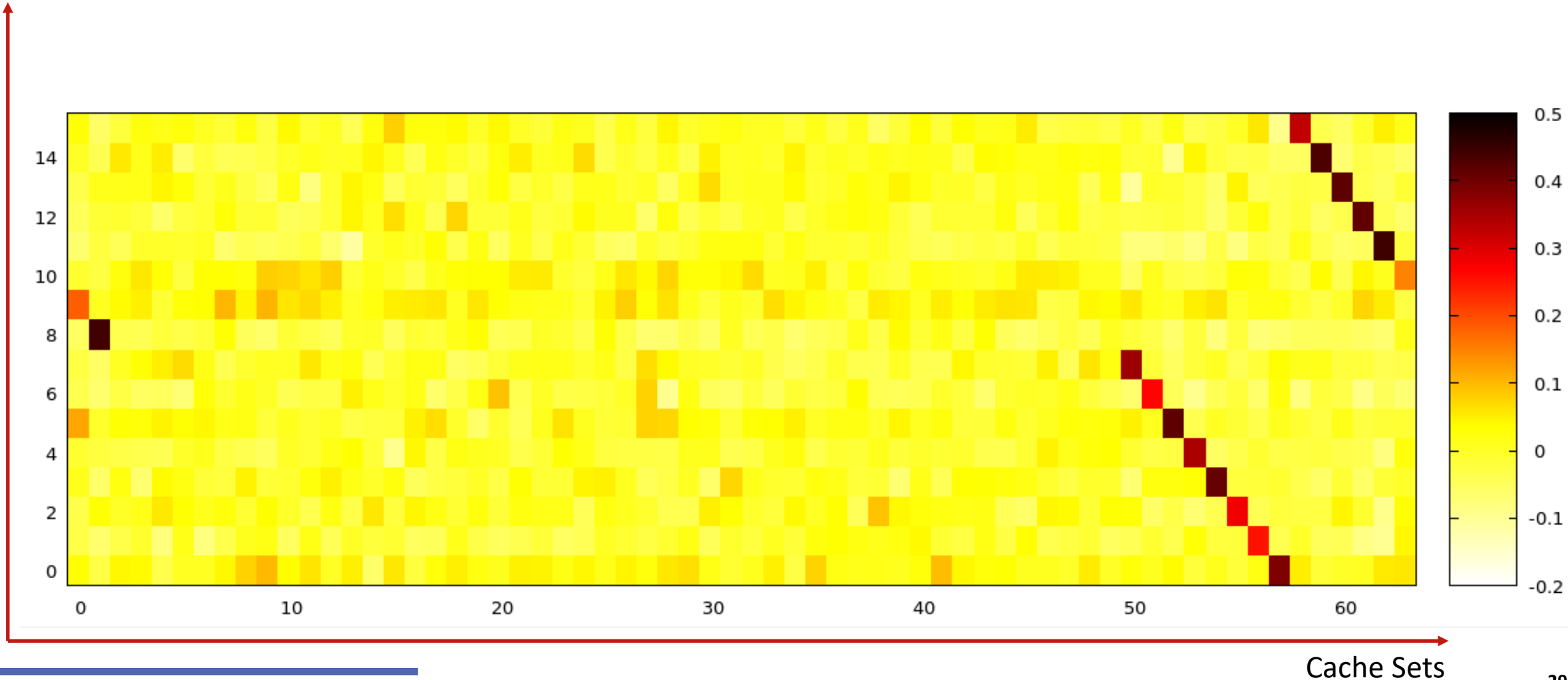
PP Attack on AES - Results

plaintext[5..8]



PP Attack on AES – More Results

plaintext[5..8]



Other Techniques (a very partial list)

- Evict+Time [OST06]
- Branch prediction [AKS06,ERAP18,...]
- L1-I Prime+Probe [Aci07]
- LLC Prime+Probe [LYG+15,IES15]
- Flush+Flush [GMWM15]
- CacheBleed [YGH17]
- TLBleed [GRBG18]
- PortSmash [CBH+18]
- SPOILER [IMB+19]

• OpenSSL

LOW Severity. This includes issues such as those that ... or hard to exploit timing (side channel) attacks.

<https://www.openssl.org/policies/secpolicy.html>

• Attacks are easy, but at the same time

- Publications are terse – technical details are often omitted
- Generic tools do not exist

Mastik

- Extremely bad acronym for
Micro-Architectural Side-channel ToolKit
- Original Aims
 - Collate information on SC attacks
 - Improve our understanding of the domain
 - Provide somewhat-robust implementations of all known SC attack techniques for every architecture
 - Implementation of generic analysis techniques
 - Reduce barriers to entry into the area
 - Shift focus to cryptanalysis

Current Status

- Reasonably robust implementation of six attacks
 - Prime+Probe on L1-D, L1-I and L3
 - Flush+Reload
 - Flush+Flush
 - Performance degradation
- Only Intel x86-64, on Linux and Mac (limited)
 - x86-32 and limited ARM currently working in the lab
- Zero documentation, little testing
- Little user feedback

<https://cs.adelaide.edu.au/~yval/Mastik/>

Mastik – Setup

```
#define NMONITOR 3
char *monitor[] = {
    "mpih-mul.c:85",
    "mpih-mul.c:271",
    "mpih-div.c:356"
};
```

```
int main(int ac, char *av[],
char *binary = av[1])
```

```
fr_t fr = fr_prepare(binary);
```

```
for (int i = 0; i < NMONITOR; i++) {
    uint64_t offset = sym_getsymboloffset(binary, monitor[i]);
    fr_monitor(fr, map_offset(binary, offset));
}
```

```
uint16_t *res = malloc(SAMPLES * NMONITOR * sizeof(uint16_t));
for (int i = 0; i < SAMPLES * NMONITOR ; i+= 4096/sizeof(uint16_t))
    res[i] = 1;
fr_probe(fr, res);
```

Allocate a handler of a
Flush+Reload attack

Tell handler what to
mo

Prepare space for results

Mastik – Attack

Run attack

```
int l = fr_trace(fr, SAMPLES, res, SLOT, THRESHOLD, 500);

for (int i = 0; i < l; i++) {
    for (int j = 0; j < NMONITOR; j++)
        printf("%d ", res[i * NMONITOR + j]);
    putchar('\n');
}

free(res);
fr_release(fr);
}
```

Output results

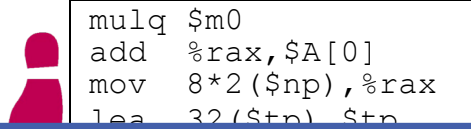
No need to program

```
FR-trace -s 2000 -c 100000 -f ./gpg \  
-m mpih-mul.c:85 \  
-m mpih-mul.c:271 \  
-m mpih-div.c:356
```

Speculative Execution Attacks

Microarchitectural channels

- Program execution leaves traces inside the processor



```
mulq $m0
add  %rax,$A[0]
mov  8*2($np),%rax
lea  32($tp), $tp
```

- We believe that hardware is working correctly. It is therefore the responsibility of software that processes sensitive material to introduce the appropriate countermeasures. (Intel)

```
adc  \0,%rdx
add  $A[0],$N[0]
adc  \0,%rdx
mov  $N[0],24($tp)
mov  %rdx,$N[1]
```

Instruction Pipelining

- Nominally, the processor executes instructions one after the other
- Instruction execution consists of multiple steps
 - Each uses a different unit

```
mulq $m0
add %rax,$A[0]
mov 8*2($np),%rax
lea 32($tp),$tp
adc \0,%rdx
mov %rdx,$A[1]
mulq $m1
add %rax,$N[0]
mov 8($a,$j),%rax
adc \0,%rdx
add $A[0],$N[0]
adc \0,%rdx
mov $N[0],-
24($tp)
mov %rdx,$N[1]
mulq $m0
add %rax,$A[1]
mov 8*1($np),%rax
adc \0,%rdx
mov %rdx,$A[0]
mulq $m1
add %rax,$N[1]
mov ($a,$j),%rax
mov 8($a,$j),%rax
adc \0,%rdx
```

Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
-------------------	--------------------	----------------	---------	------------

Instruction Pipelining

- Nominally, the processor executes instructions one after the other
- Instruction execution consists of multiple steps
 - Each uses a different unit
- Pipelining increases utilisation by executing steps of multiple instructions

Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back
Instruction Fetch	Instruction Decode	Argument Fetch	Execute	Write Back

$c = a / b;$

$d = c + 5;$


```

mulq $m0
add %rax,$A[0]
mov 8*2($np),%rax
lea 32($tp),$tp
adc \0,%rdx
mov %rdx,$A[1]
mulq $m1
add %rax,$N[0]
mov 8($a,$j),%rax
adc \0,%rdx
add $A[0],$N[0]
adc \0,%rdx
mov $N[0],-24($tp)
mov %rdx,$N[1]
mulq $m0
add %rax,$A[1]
mov 8*1($np),%rax
adc \0,%rdx
mov %rdx,$A[0]
mulq $m1
add %rax,$N[1]
mov ($a,$j),%rax
mov 8($a,$j),%rax
adc \0,%rdx
    
```

- Problem: dependencies

Out-of-order execution

- Execute instructions when data is available rather than by program order

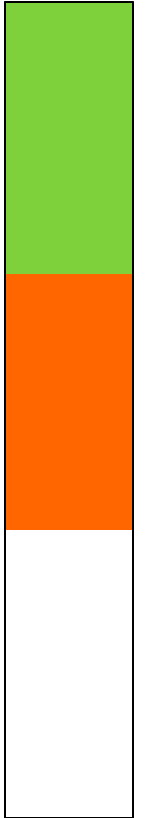
IF	ID	AF	EX	WB
IF	ID		EX	WB
IF	ID	AF	EX	WB

$c = a / b;$

$d = c + 5;$


$e = f + g;$

- Completed instructions wait in the **reorder buffer** until all previous instructions are **retired**
- Why not retire immediately?



Out-of-order execution

- Execute instructions when data is available rather than by program order

IF	ID	AF	EX	WB
IF	ID		EX	WB
IF	ID	AF	EX	WB

$c = a / b;$

$d = c + 5;$

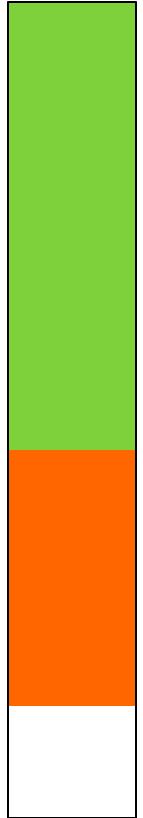
$e = f + g;$

What if $b=0$?

- Completed instructions wait in the **reorder buffer** until all previous instructions are **retired**

- Why not retire immediately?

- Out-of-order execution is speculative!**



Speculative execution

- **Abandon** instructions in the reorder buffer if never executed in program order

IF	ID	AF	EX	WB
IF	ID	AF	EX	WB
IF	ID	AF	EX	WB

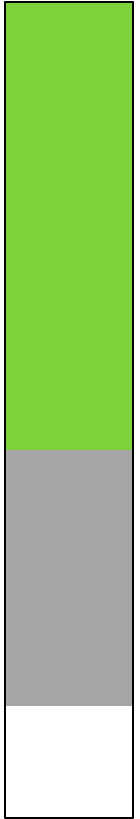
`c = a / b;`

`d = c + 5;`

`e = f + g;`

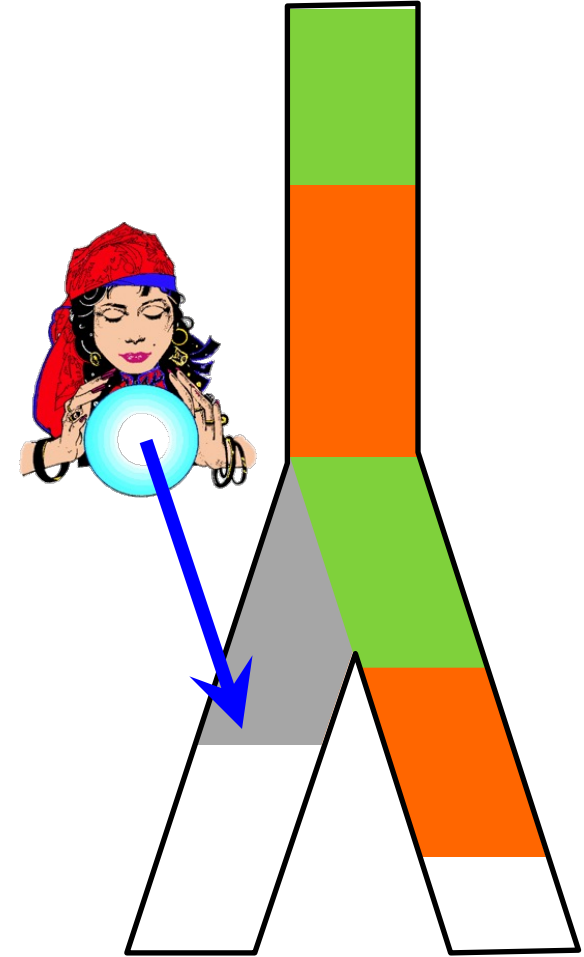
With **b=0!!**

- Also useful for handling branches



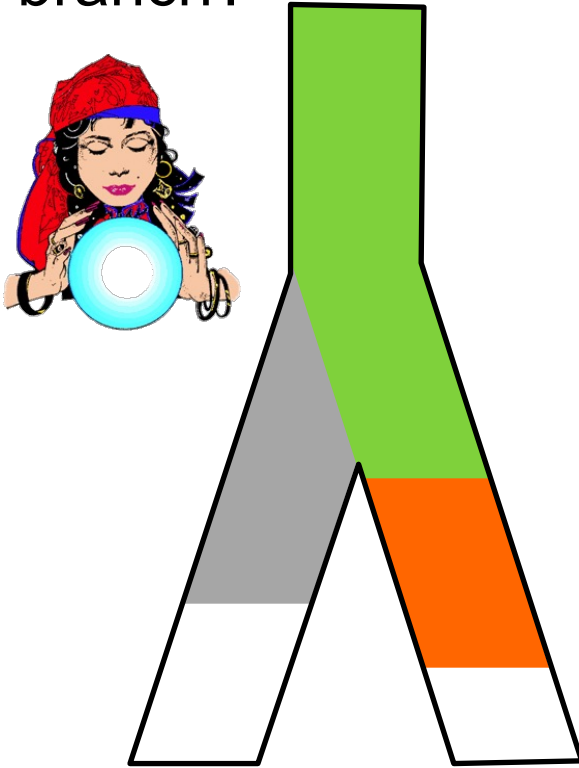
Speculative Execution and Branches

- When execution reaches a branch
- The processor predicts the outcome of the branch
- Execution proceeds (speculatively) along predicted branch
- **Correct prediction** → all is well
- **Misprediction** → abandon and resume

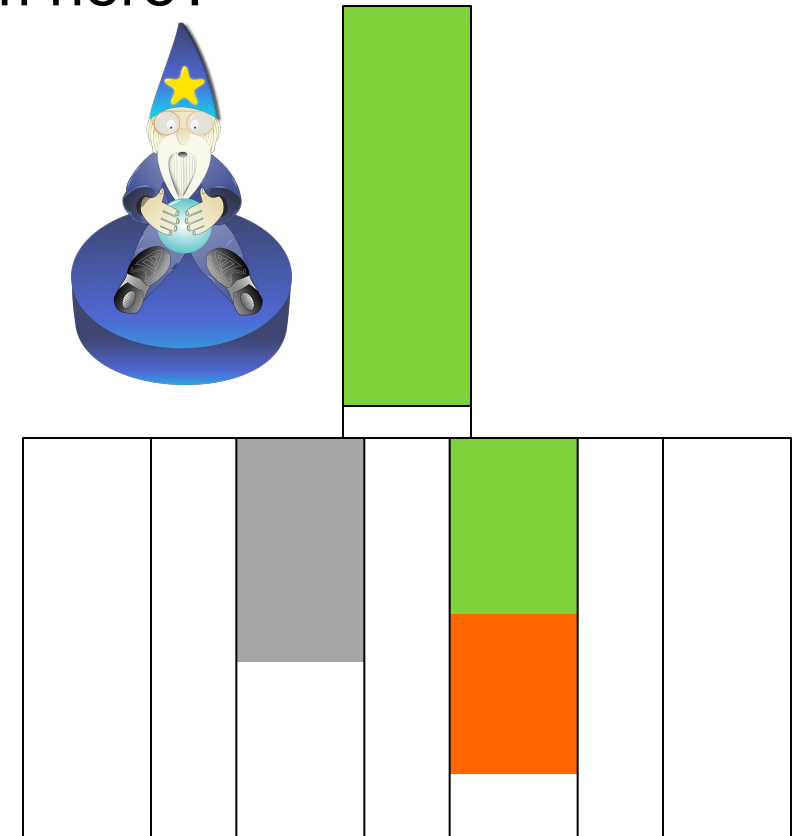


Branch Prediction

- Branch History Buffer (BHB)
 - Outcome of conditional branches
 - Does the program tend to take this branch?

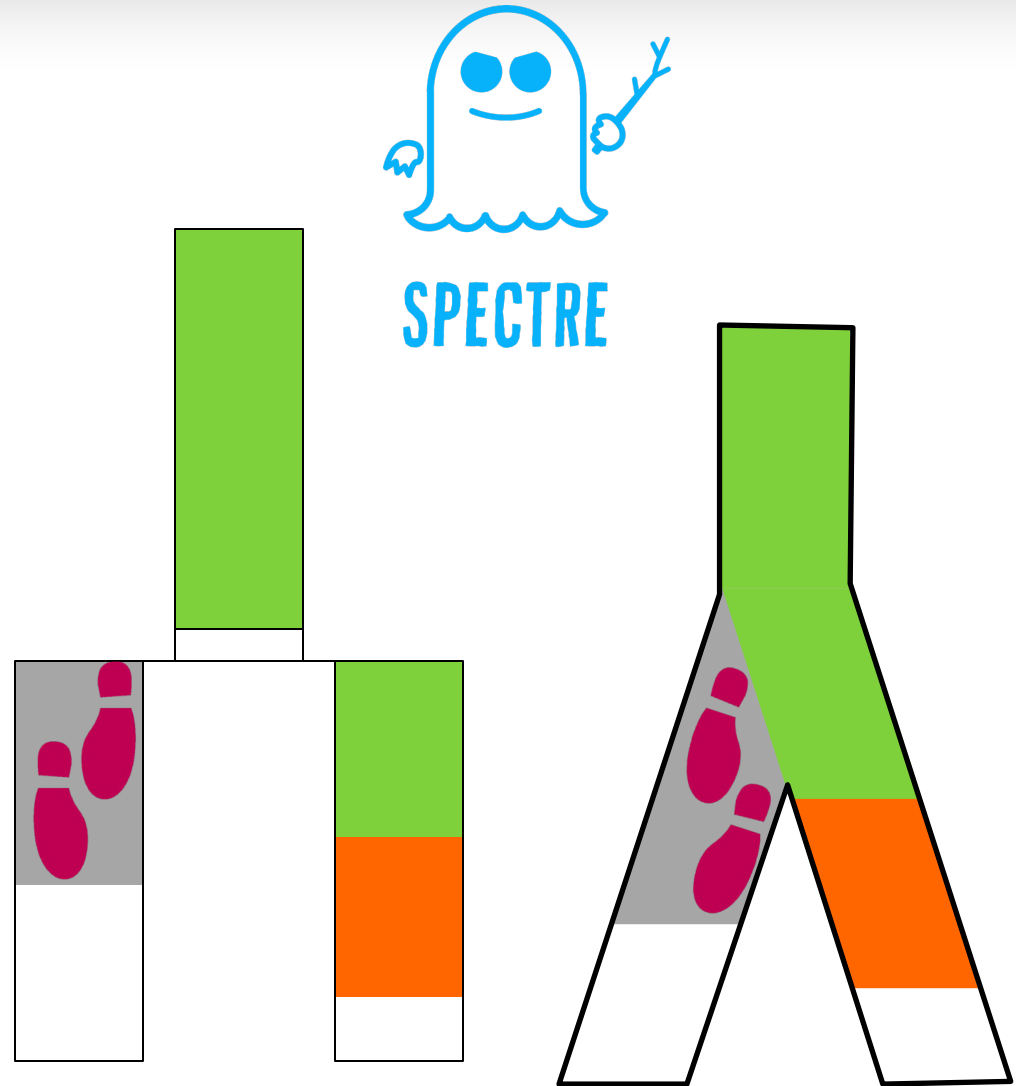


- Branch Target Buffer (BTB)
 - Target of indirect branches
 - Where does the program usually go from here?



Main Discovery

- Abandoned speculative execution leaves traces in the microarchitecture
- Developed techniques to implement a covert channel from the abandoned code to the attacker



Attack overview

If (access permitted)

Access data

Leak data

Read data

Implicit or explicit validity test

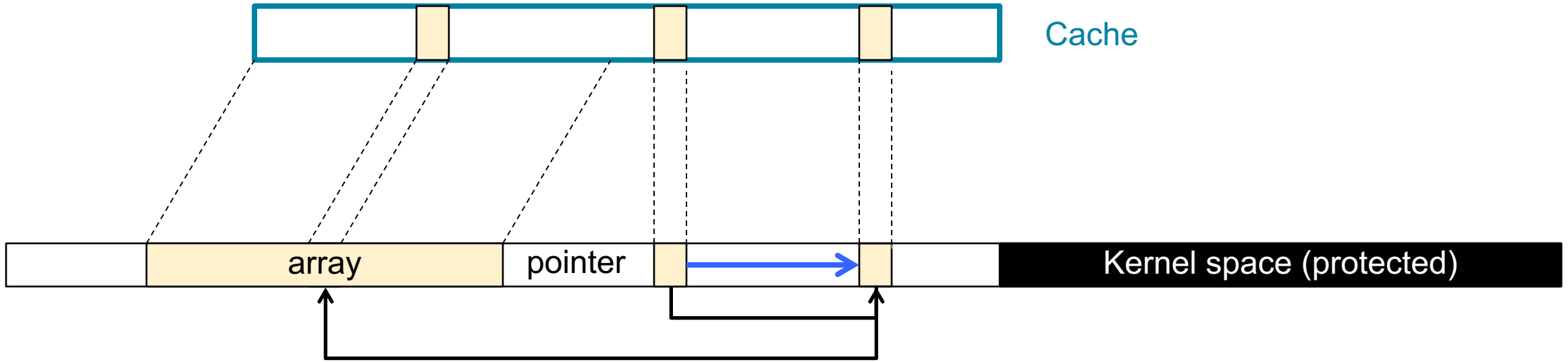
Via a microarchitectural covert channel



Meltdown

Meltdown

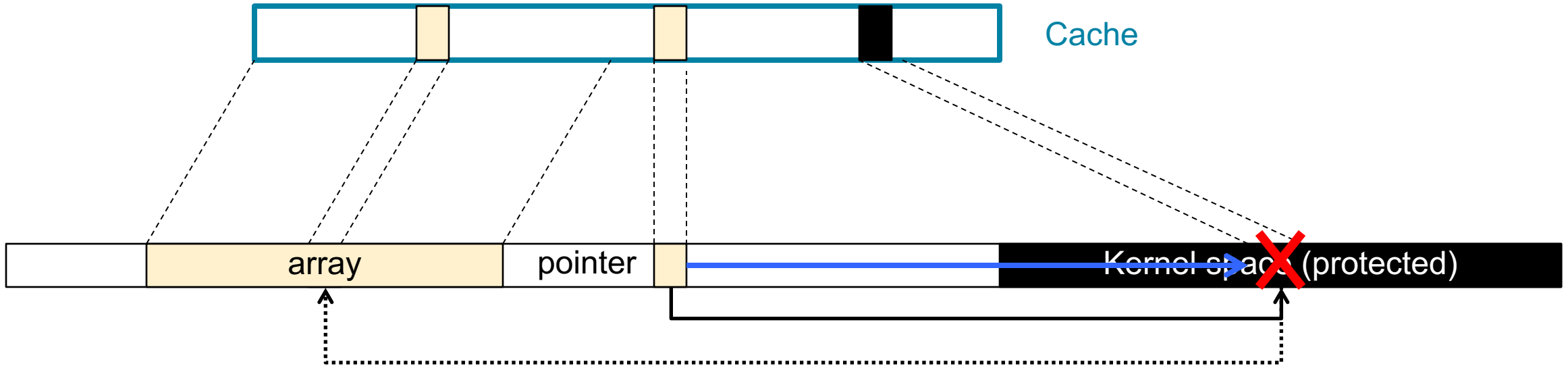
```
i = *pointer;  
y = array[i * 256];
```



Meltdown



```
i = *pointer;  
y = array[i * 256];
```

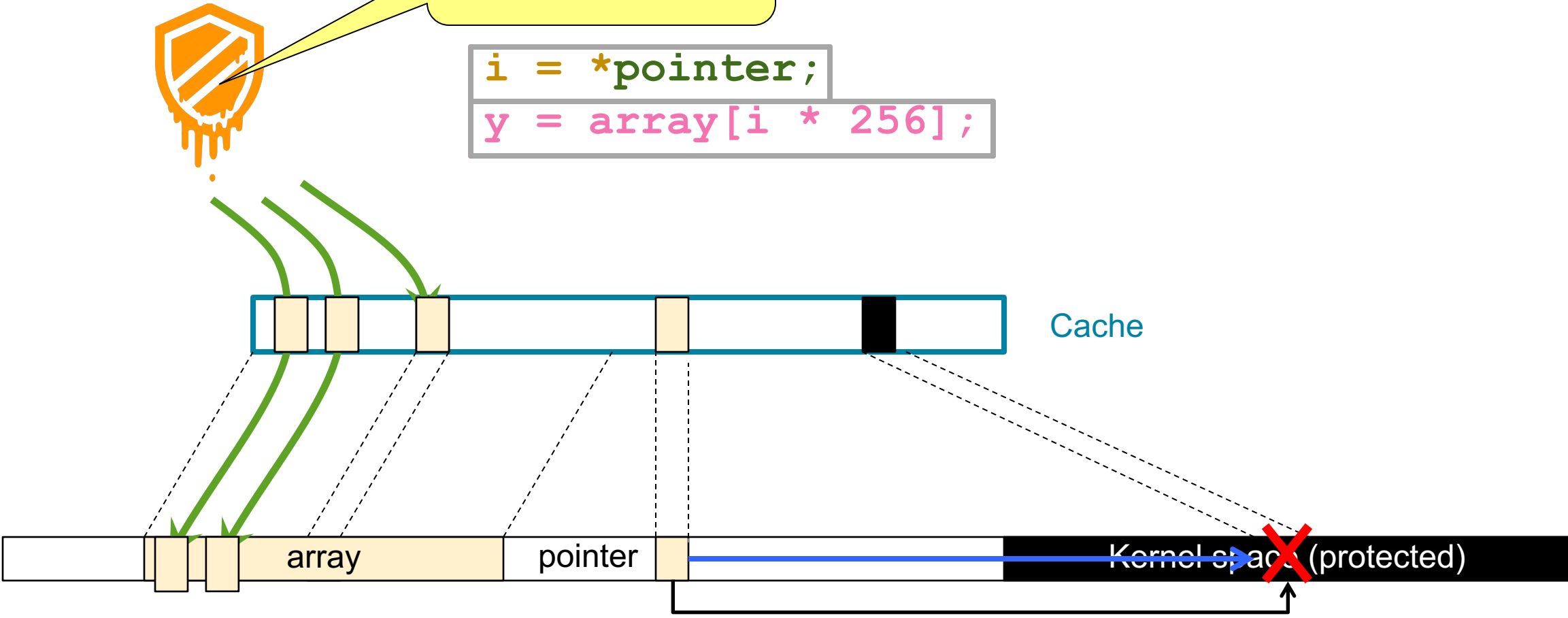


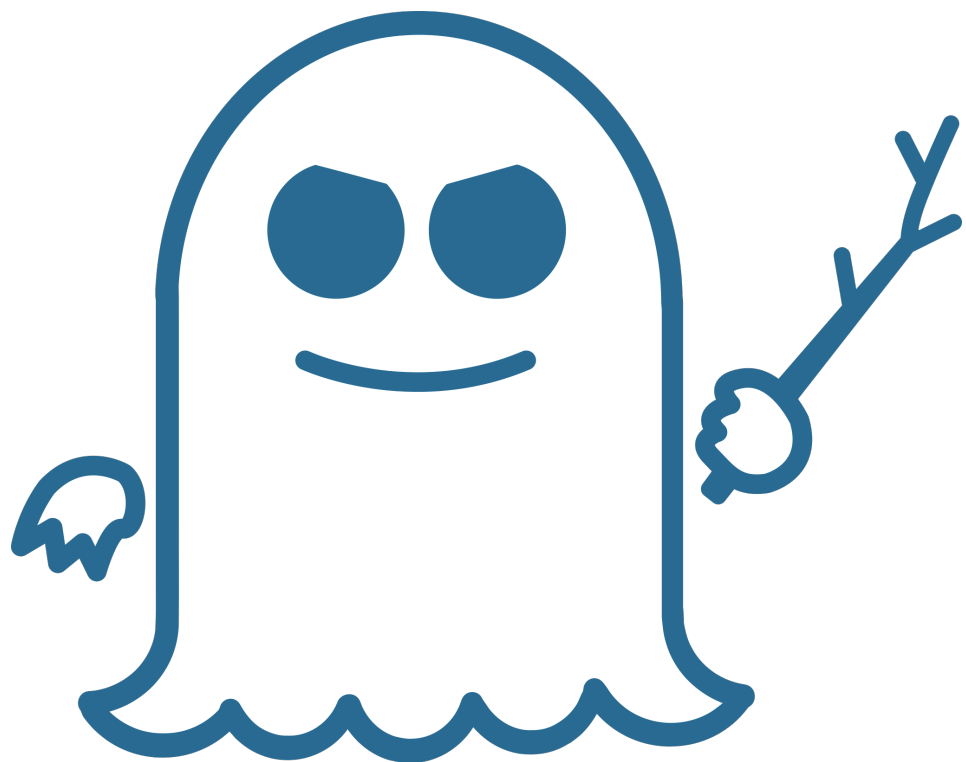
Meltdown



Found it!!!

```
i = *pointer;  
y = array[i * 256];
```



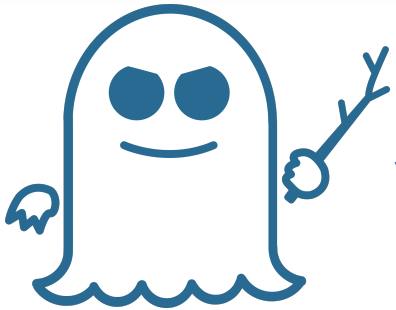


Spectre

Variant 1

Spectre (Variant 1)

Victim

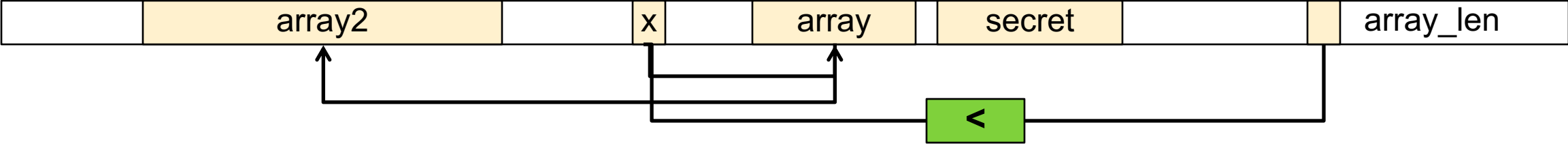


Attacker

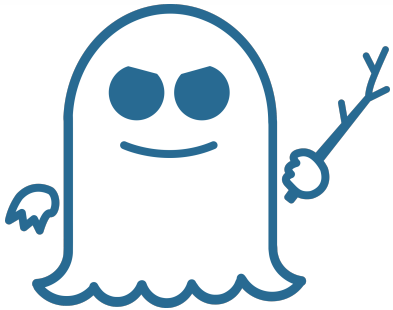


Branch not taken!

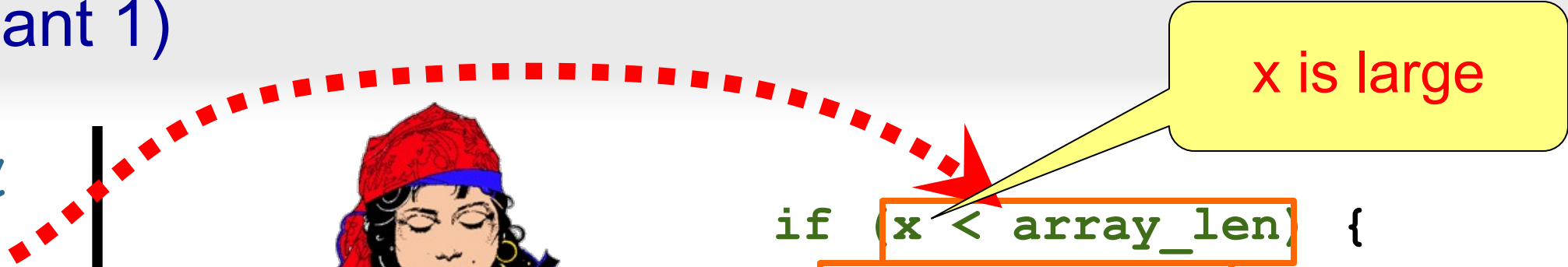
```
if (x < array_len) {  
    i = array[x];  
    y = array2[i * 256];  
}
```



Spectre (Variant 1)



Attacker

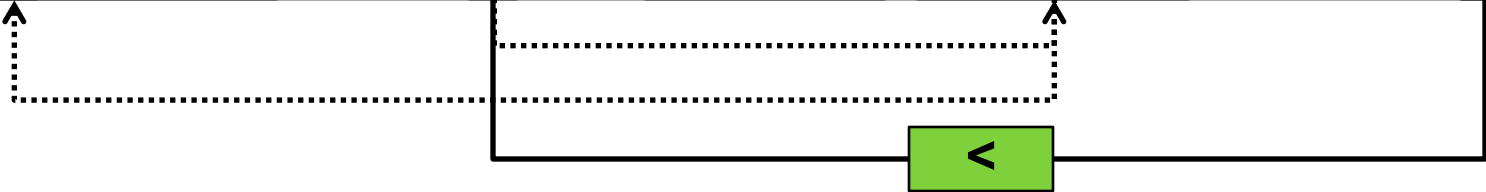
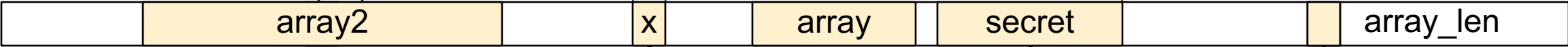


Branch not taken!

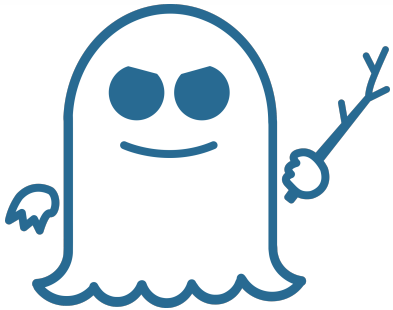
```
if (x < array_len) {  
    i = array[x];  
    y = array2[i * 256];  
}
```



Cache



Spectre (Variant 1)



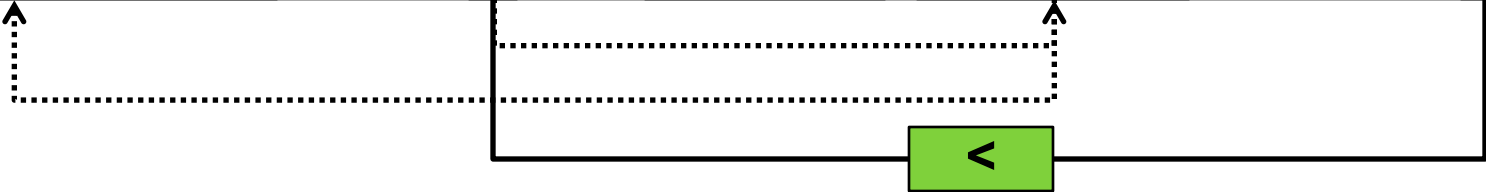
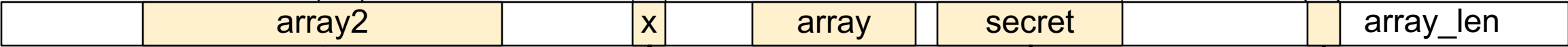
Attacker

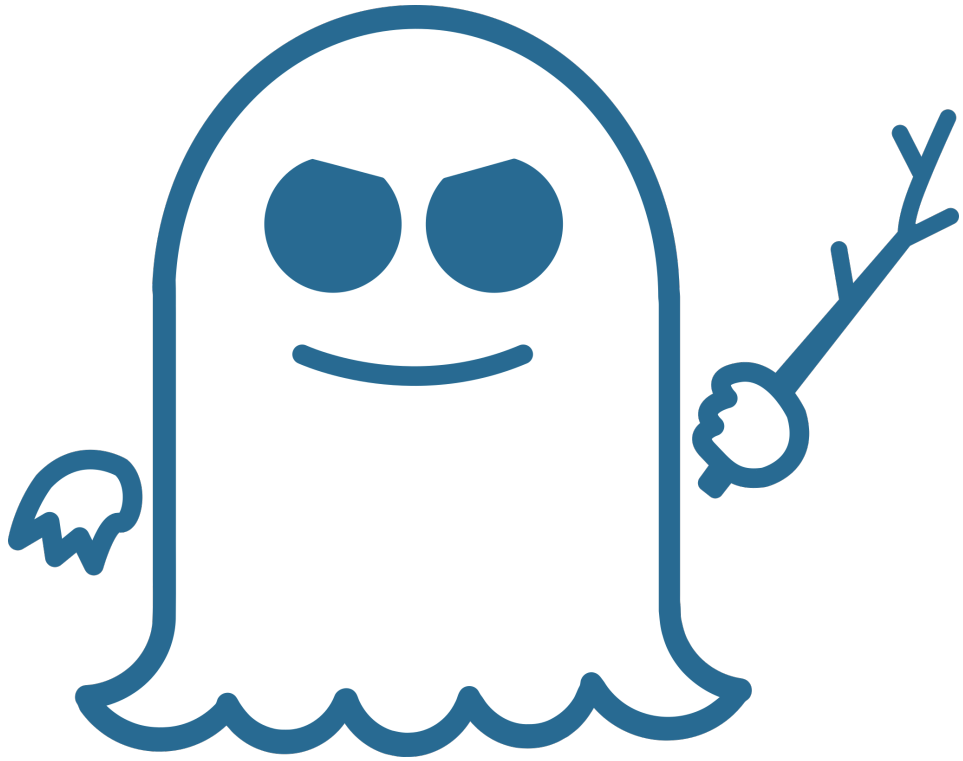


~~Branch not taken~~

Mispredict

```
if (x < array_len) {  
    i = array[x];  
    y = array2[i * 256];  
}
```



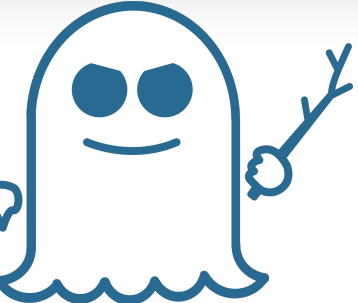


Spectre

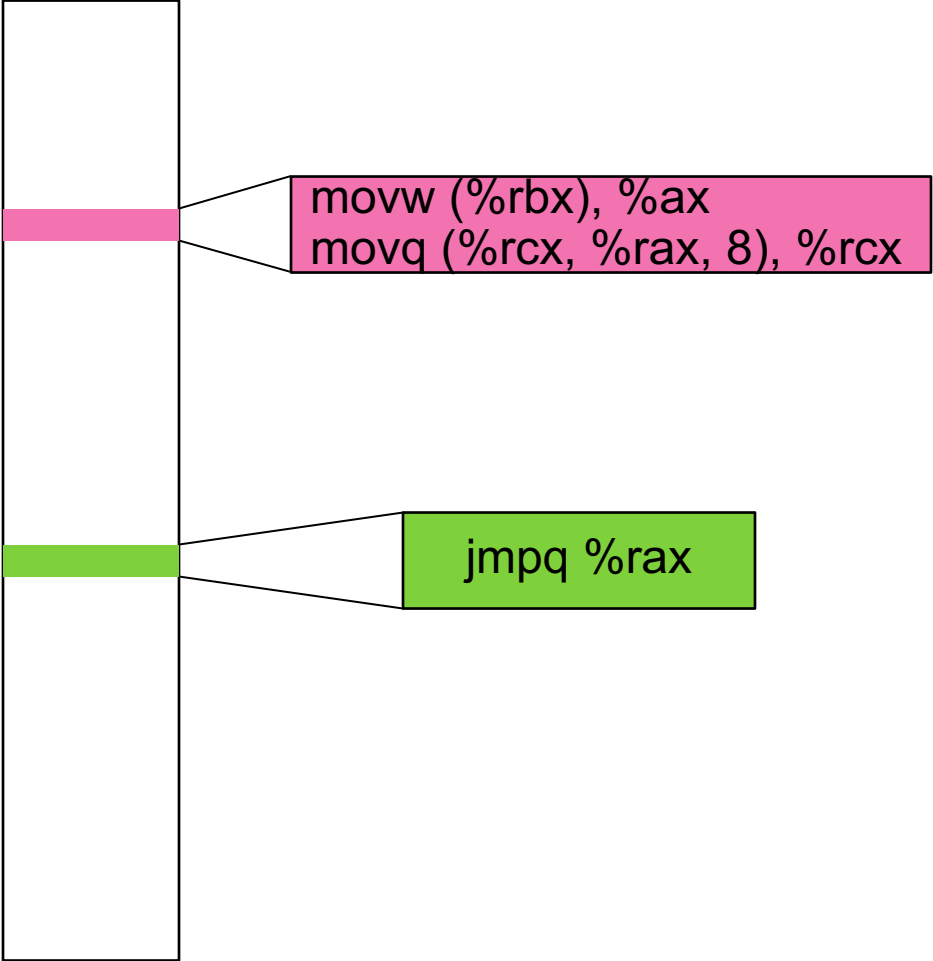
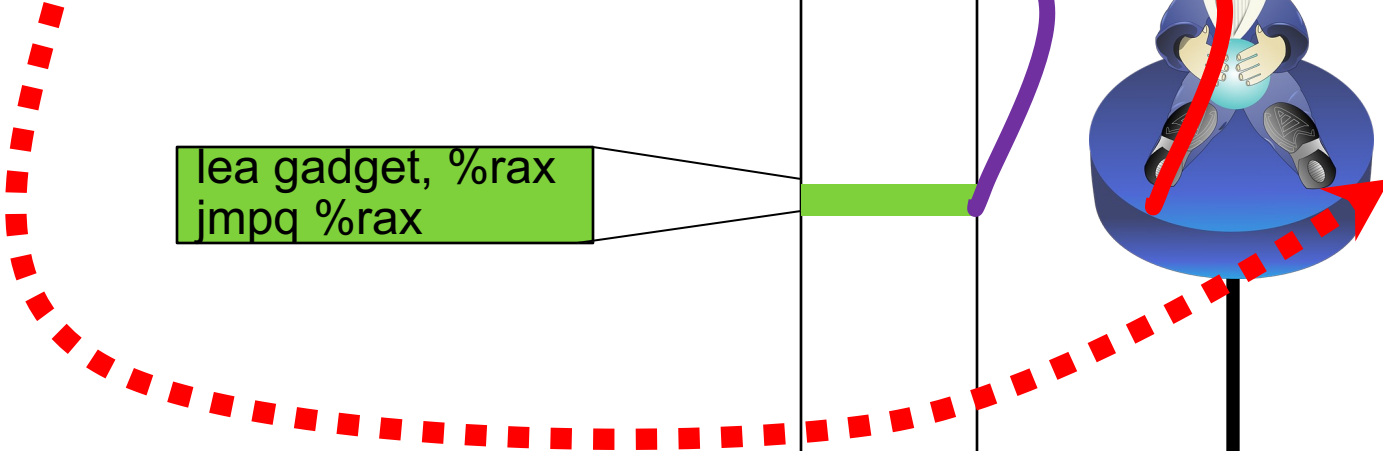
Variant 2

Spectre (Variant 2)

Victim



Attacker

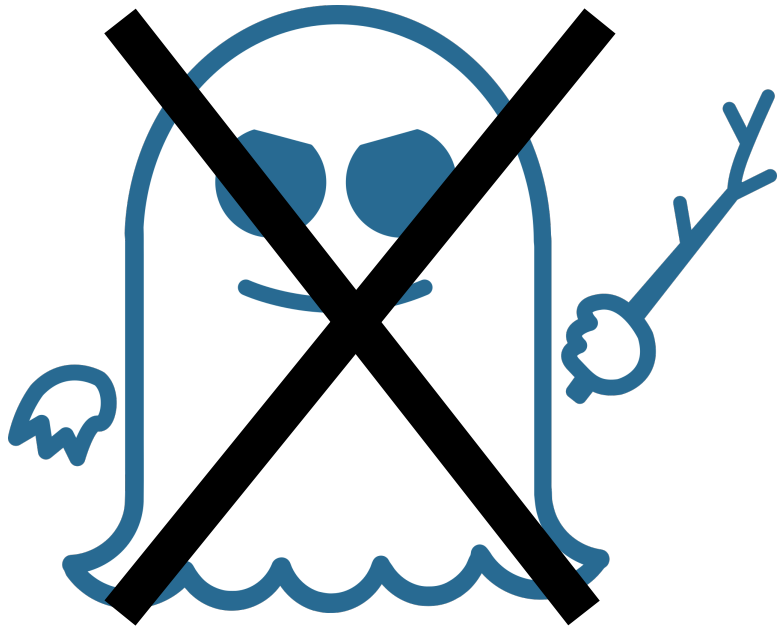


How deep does the rabbit hole go?

- Variant 3a: leak model-specific registers

- "The processor is, in fact, operating as it is designed," Smith said. "And in every case, it's been this side-channel approach that the researchers used to gain information even while the processor is executing normally its intended functions." (Intel)

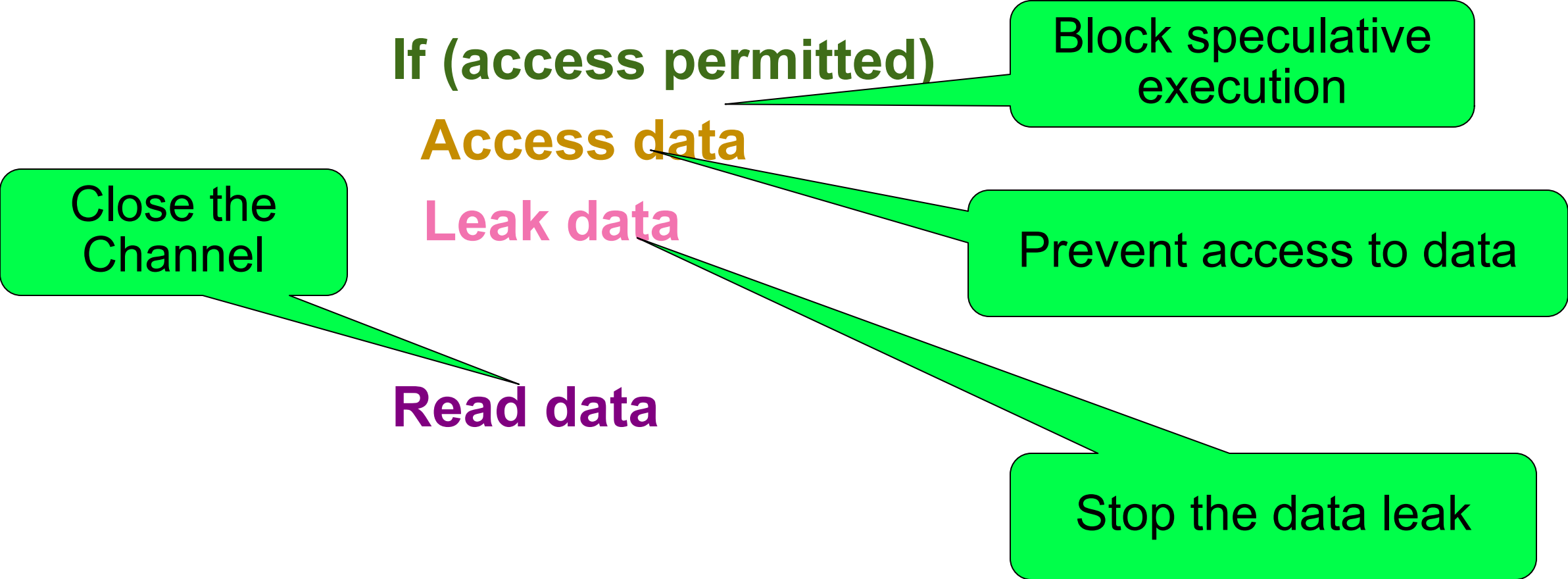
- Fallout: read data from the store buffer



Countermeasures

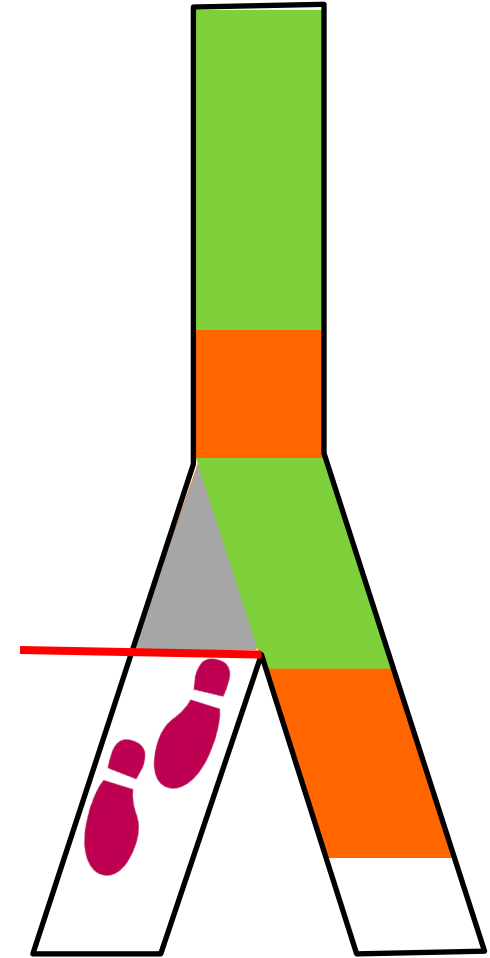


Countermeasures



Spectre – Variant 1

- Compiler patches to block speculative execution
- **Barriers at event**
 - Use static analysis



Spectre – Variant 2

- New MSRs to prevent BTB

preco
dom

– No

– Hi

lea

– BT



The image shows a screenshot of the PCWorld website. The header features the PCWorld logo with 'FROM IDG' underneath. A navigation bar includes categories like NEWS, REVIEWS, HOW-TO, and VIDEO. Below this, there are sub-categories such as BUSINESS, LAPTOPS, TABLETS, PHONES, HARDWARE, SECURITY, SOFTWARE, and GADGETS. A search bar and a 'SUBSCRIBE' button are also visible. The main content area displays a news article titled 'Microsoft issues emergency Windows patch to disable Intel's buggy Spectre fix'. The article is attributed to Mark Hachman, Senior Editor at PCWorld, and is dated January 28, 2018, at 11:40 PM PT. The article text begins with 'If your Windows PC seems buggier than usual after the recent round of Spectre patches, you might want to download this.'



Meltdown

- Use a sensor to detect the kernel
– Overhead

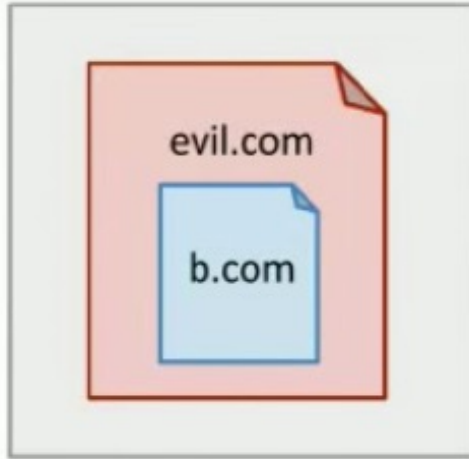


Kernel space (protected)

Kernel

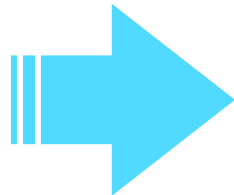
Spectre – Variant 1

- Strict site isolation



- Limit memory access or use of data

```
if (x < array_len) {  
    i = array[x];  
    y = array2[i * 256];  
}
```



```
i = array[x % array_len];  
y = array2[i * 256];
```

Spectre – Variant 1

- Reduce timer frequency
 - Also disable features such as SharedArrayBuffers

Conclusions

- Decades of focus on performance with little regard to security bring us Spectre and Meltdown
 - This is not much different to software development
 - ... but it's harder to fix
- Likely to affect computer security for a long time
 - We do not understand the full implications yet
- Microarchitectural channels matter