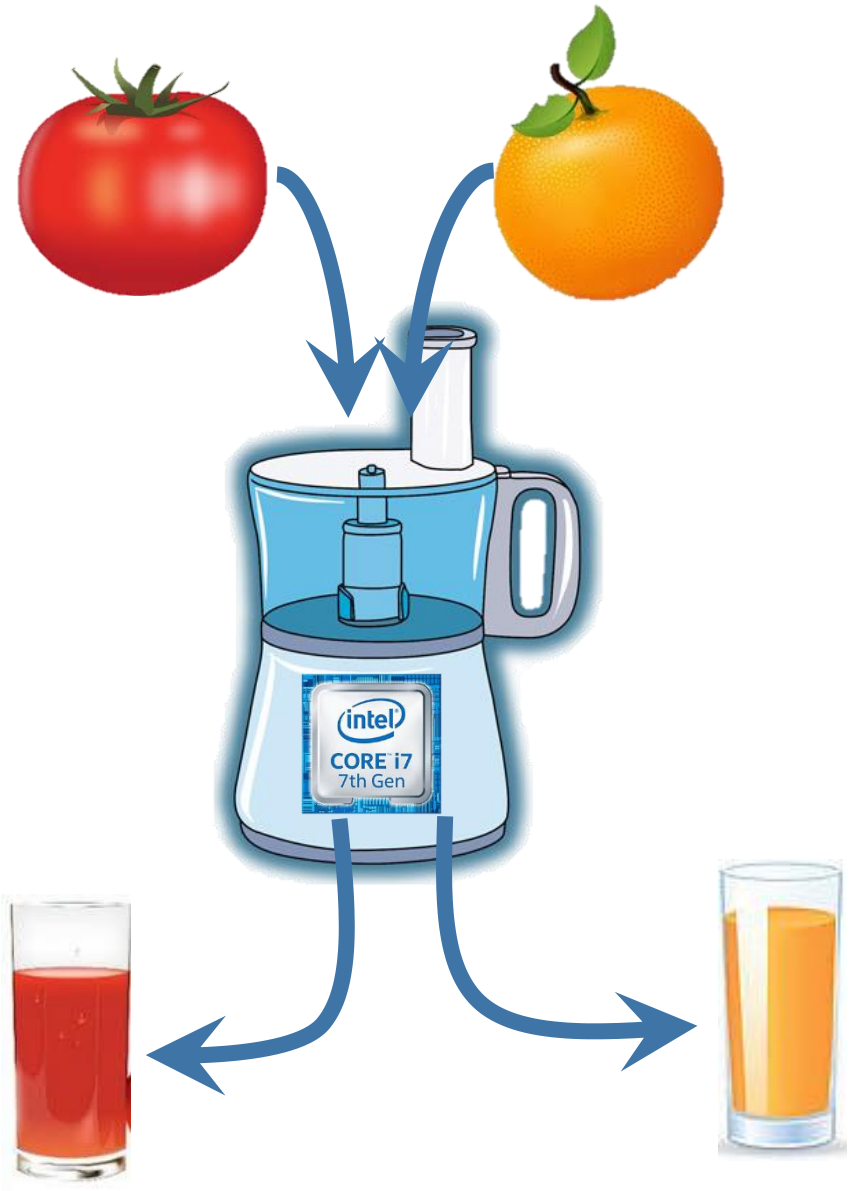
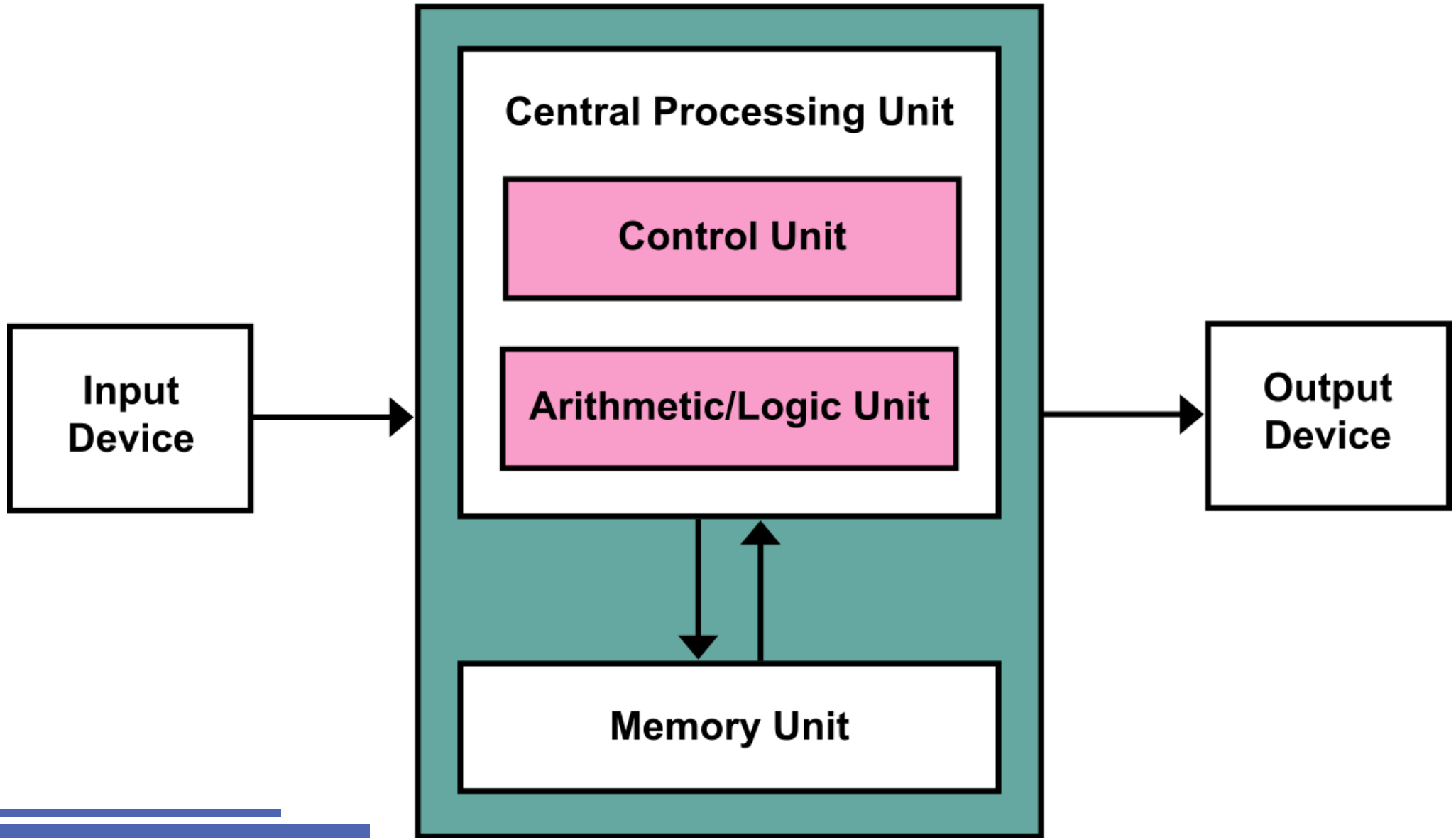
An Introduction to Microarchitectural Attacks

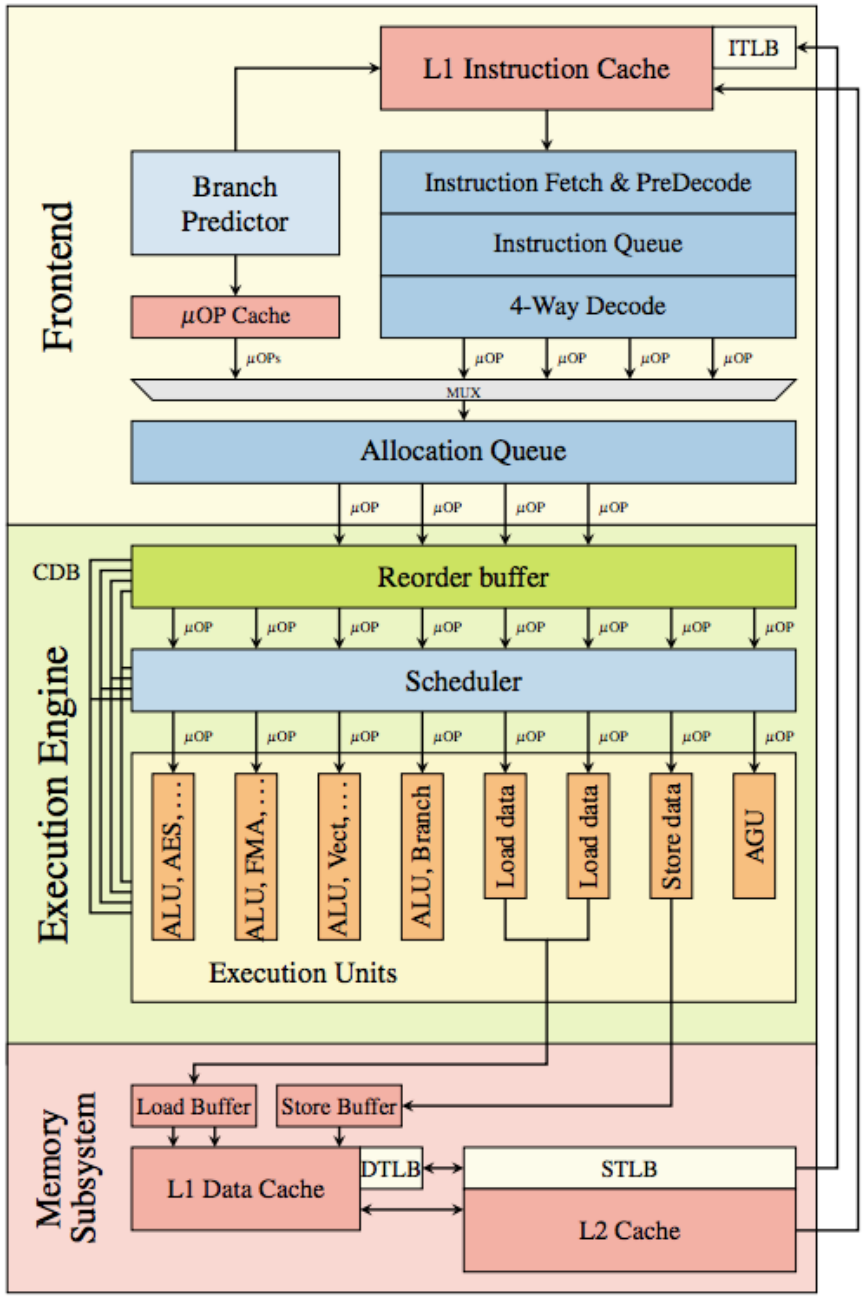
Yuval Yarom, The University of Adelaide and Data61


intel
CORE i7
7th Gen

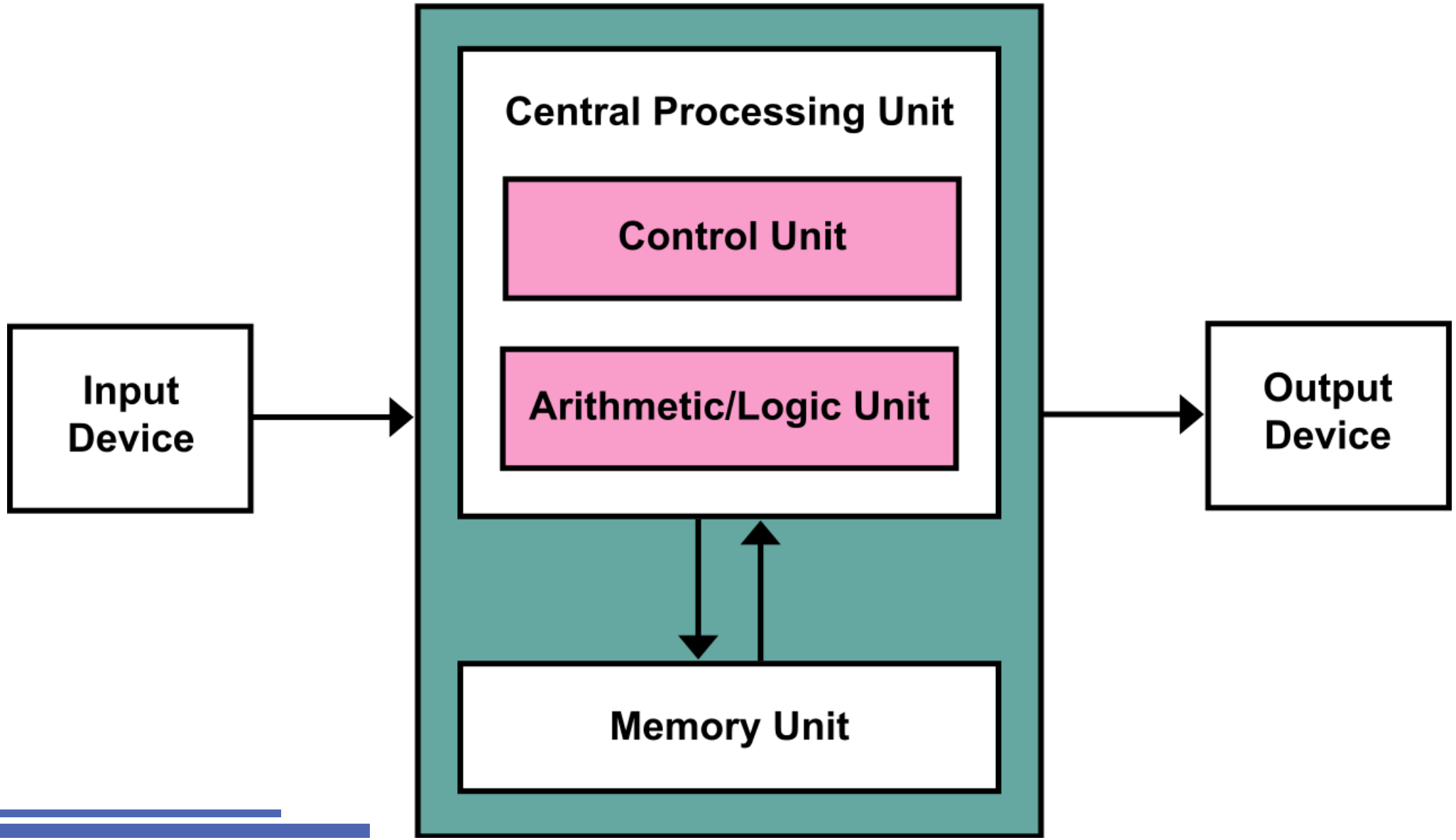
First year computer science: Von Neumann Architecture



Computer Architecture course



The rest of the CS degree

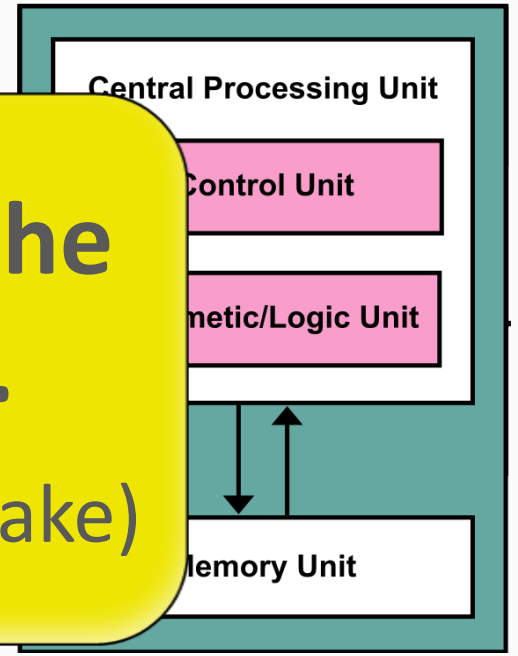


Programmers' Model of Execution

- High level languages
- Asymp
- Archit
- Focus
- CPU V

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

Thomas Dullien (@halvarflake)



	Abstract	Concrete
Hardware	Dedicated	Shared
Memory	Uniform	Non-uniform
Execution	Serial	Superscalar

CPU vs. Memory

	Processor Speed	Memory Latency
	1 MHz	500 ns

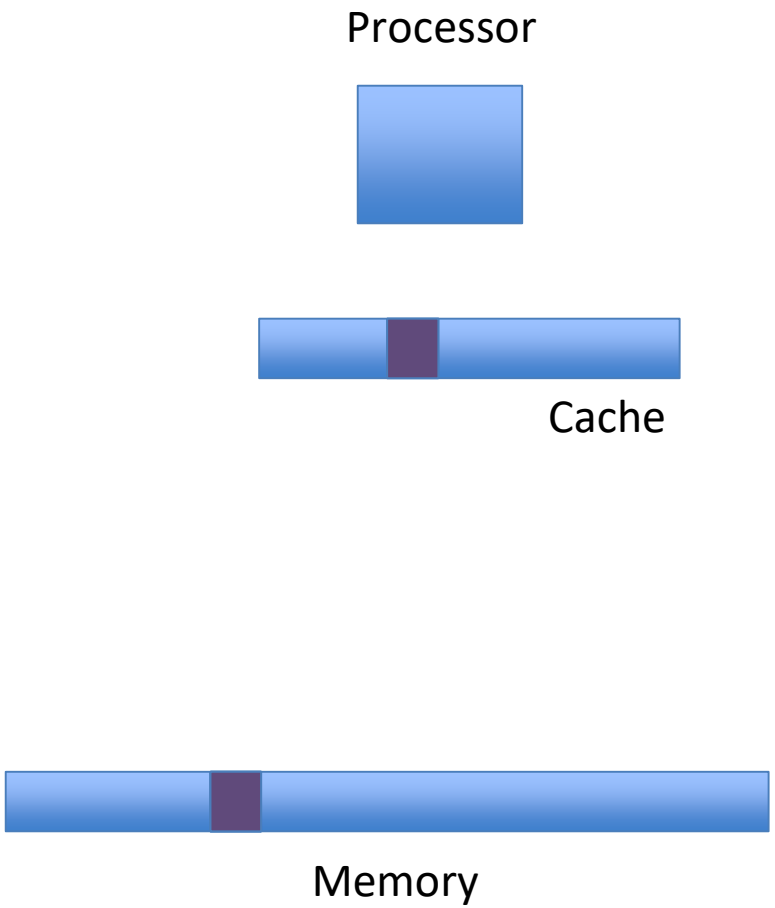
	8*2600 MHz	63 ns

Bridging the gap

Cache utilises locality to bridge the gap

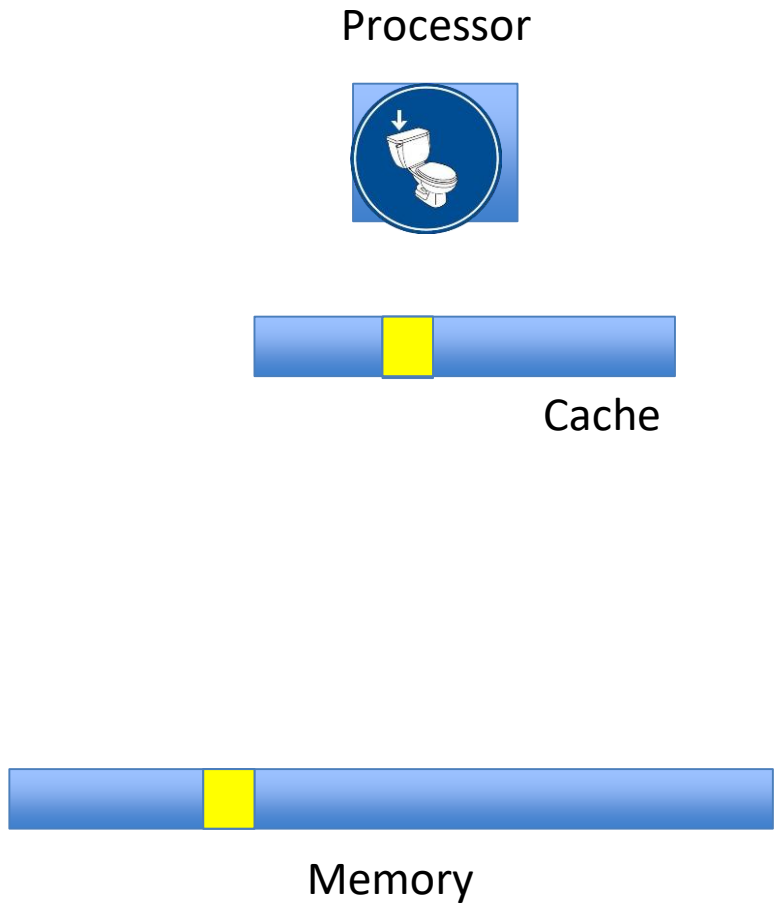
- Divides memory into *lines*
- Stores recently used lines

- In a *cache hit*, data is retrieved from the cache
- In a *cache miss*, data is retrieved from memory and inserted to the cache



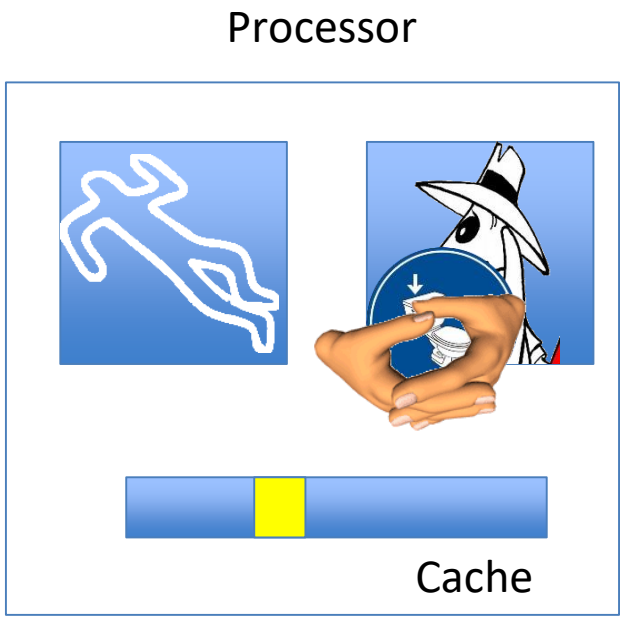
Cache Consistency

- 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

$$M = C^d \bmod N$$

M

$$C = M^e \bmod N$$

Key Generation:

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

GnuPG 1.4.13 Exponentiation

```
x ⟵1
for i ⟵|d|-1 downto 0 do
    x ⟵x^2 mod n
    if (d_i = 1) then
        x = xC mod n
    endif
done
return x
```

Example:

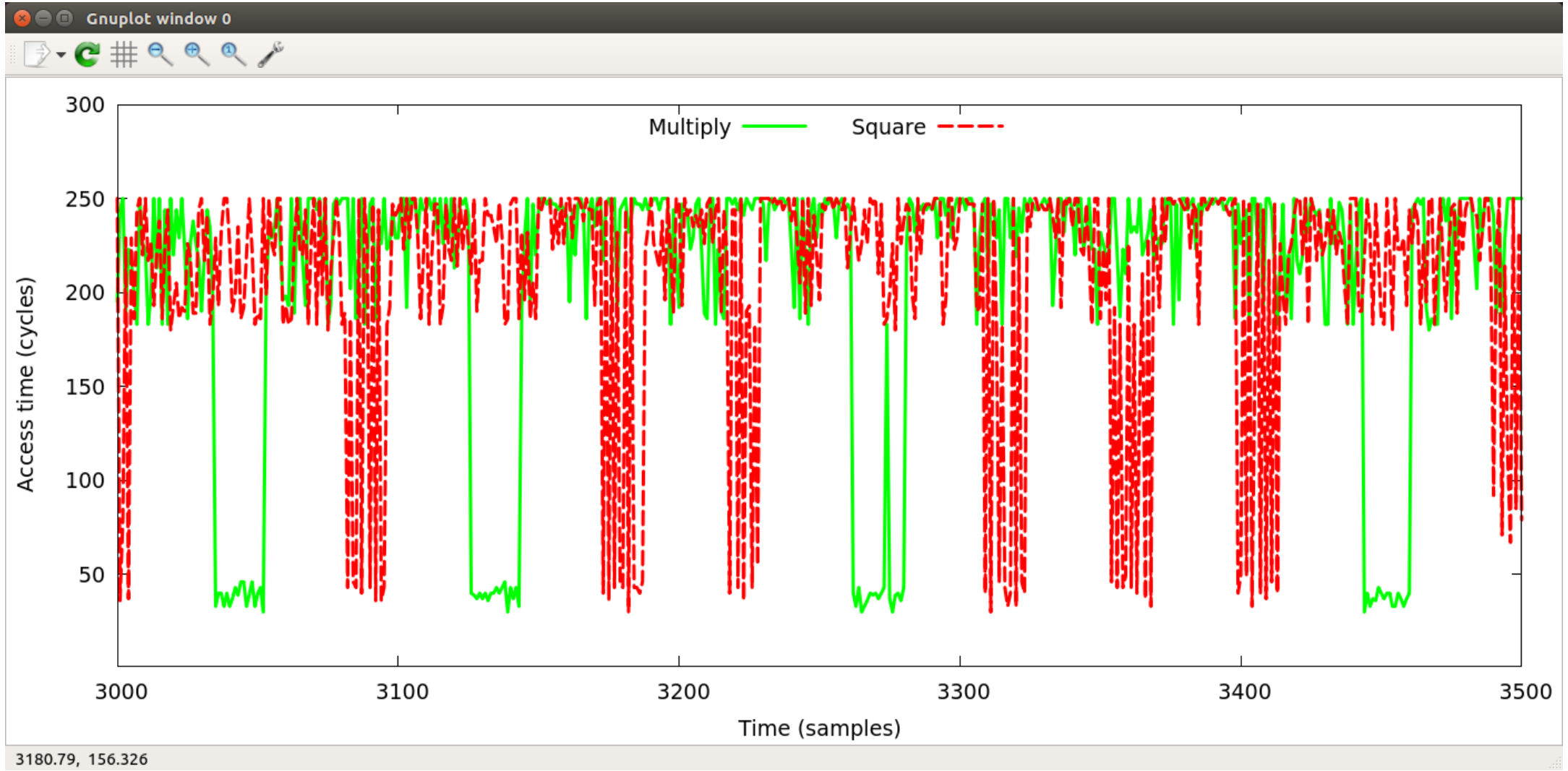
$11^5 \bmod 100 =$

$161{,}051 \bmod 100 = 51$

Operation	x	i	d_i

The private key is encoded in the sequence of operations !!!

Flush+Reload on GnuPG 1.4.13

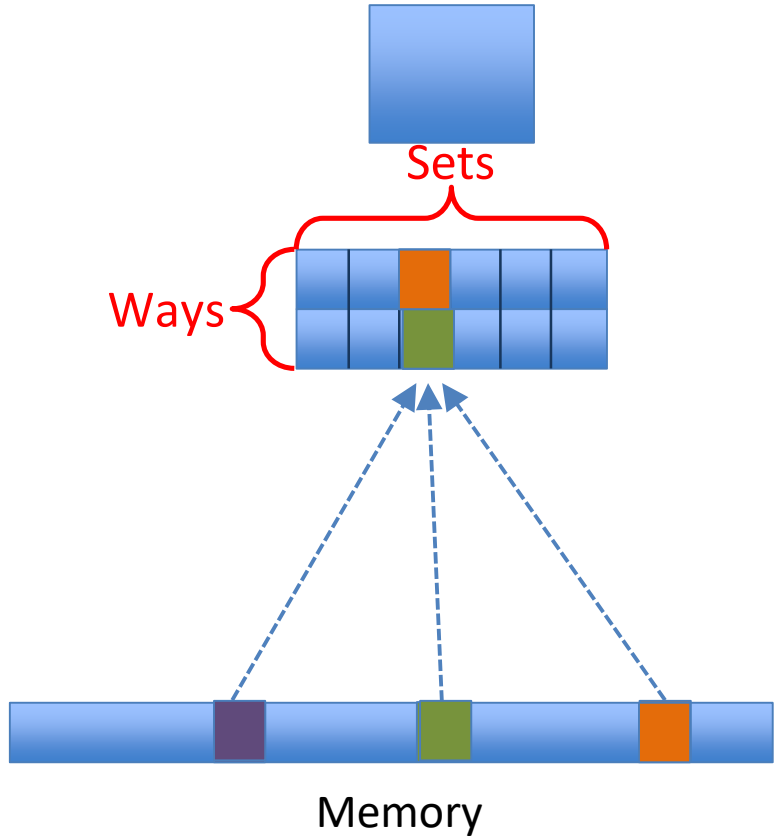


The FLUSH+RELOAD Technique

- 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

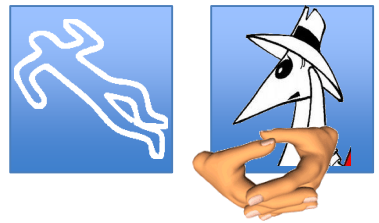
Set Associative Caches

- 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

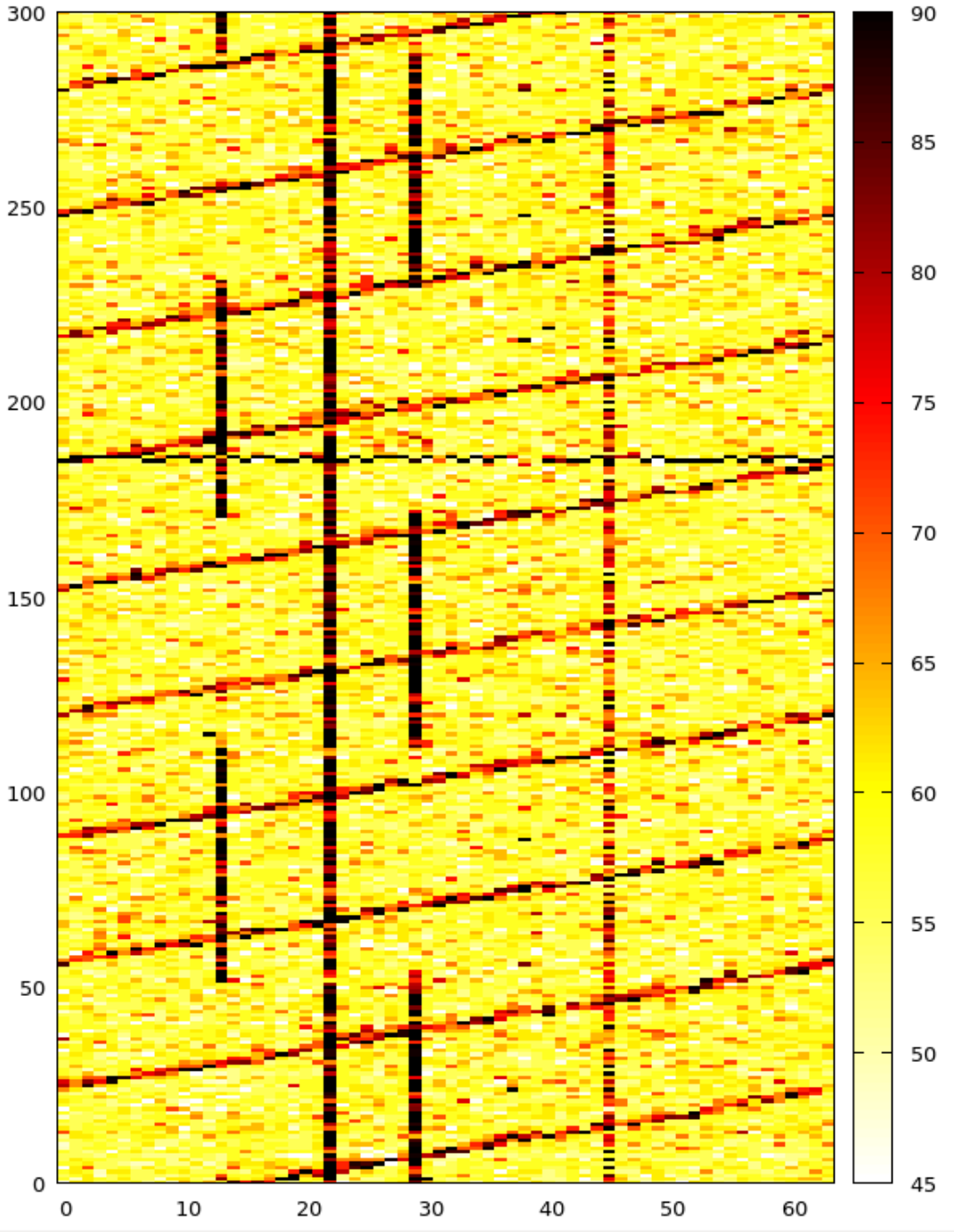
Sample Victim: Data Rattle

```
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     ) ^ 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 >>
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >>
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >>
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >>
    /* 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, 0xfbf0f00bU,
    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,
```

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

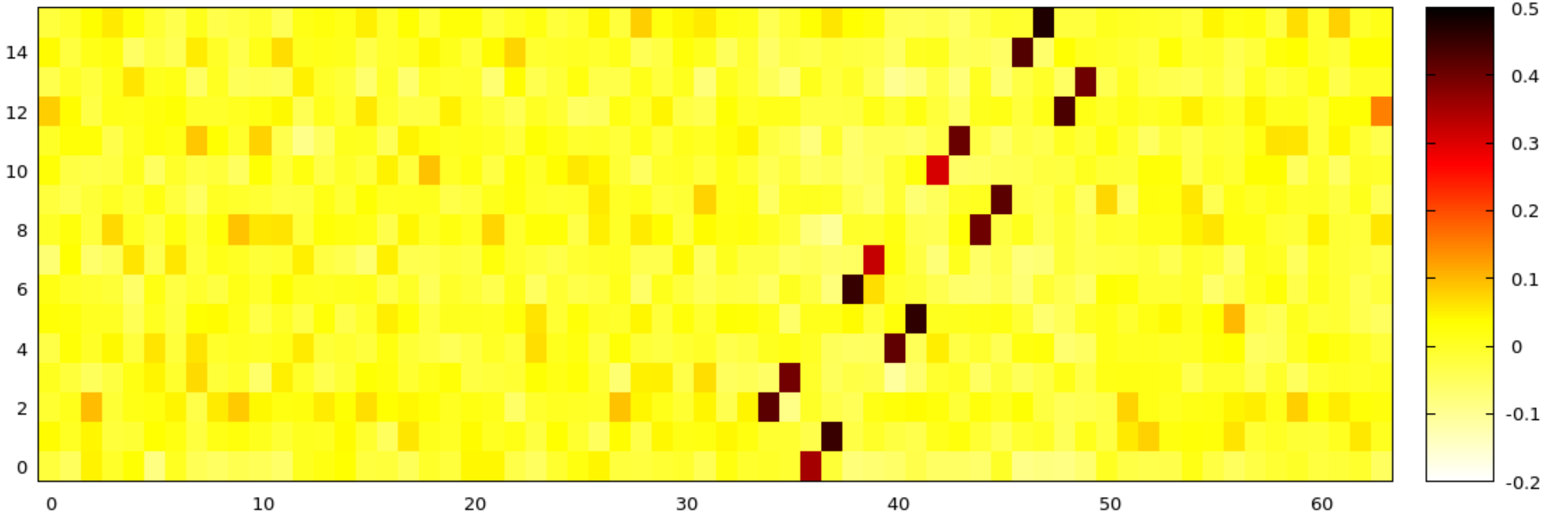
- Assume we know the plaintext and the index (s0>>24)
  - **We can recover the most significant byte of the key**

Prime+Probe Attack on AES

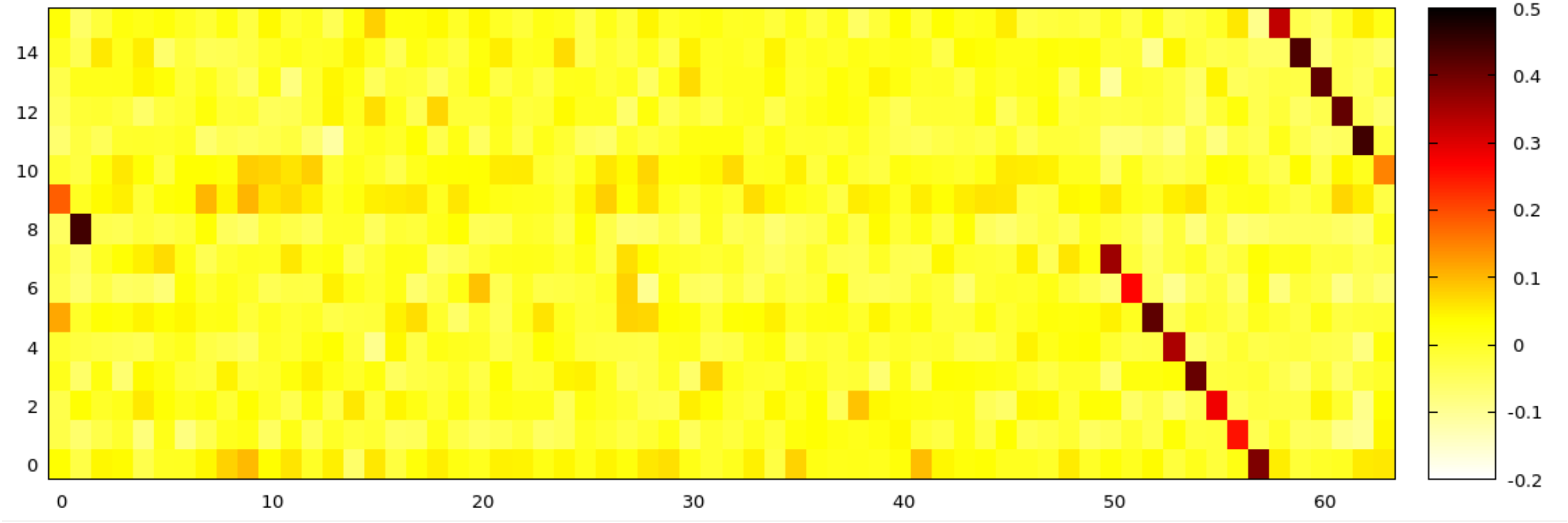
```
s0 = plaintext ^ key
t0 = Te0[s0>>24]
```

- For many plaintexts do: Prime, Encrypt, Probe
- Calculate the average probe time of each cache set as a function of the byte value

PP Attack on AES - Results



PP Attack on AES – More Results



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-**A**rchitectural **S**ide-channel **T**ool**K**it
- 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

Mastik – Setup

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

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

  fr_t fr = fr_prepare(

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

  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

```
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);
}
```

Run attack

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.

```
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 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

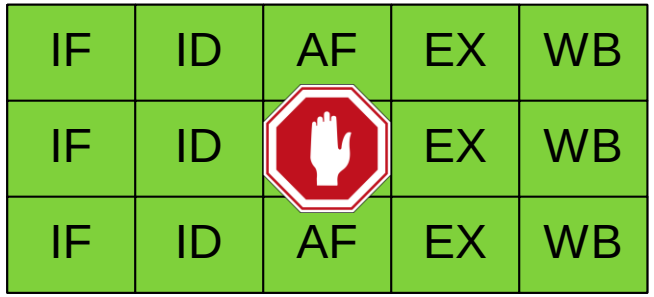
```
c = a / b;
d = c + 5;
```

```
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,%rdD
```

- Problem: dependencies

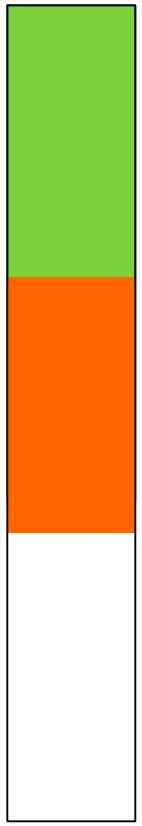
Out-of-order execution

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



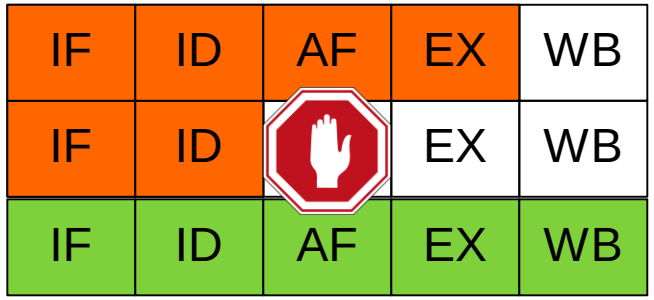

```
c = a / b;
d = c + 5;
e = f + g;
```

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



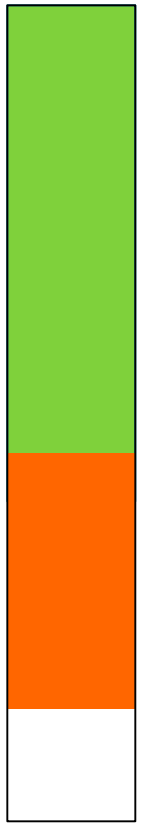
Out-of-order execution

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




```
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!**

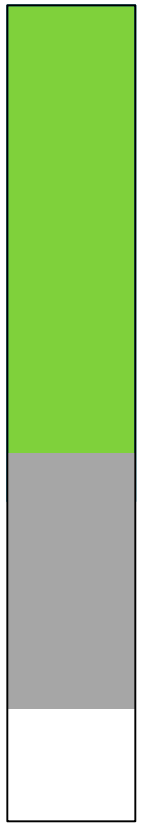
Speculative execution

- **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`!!

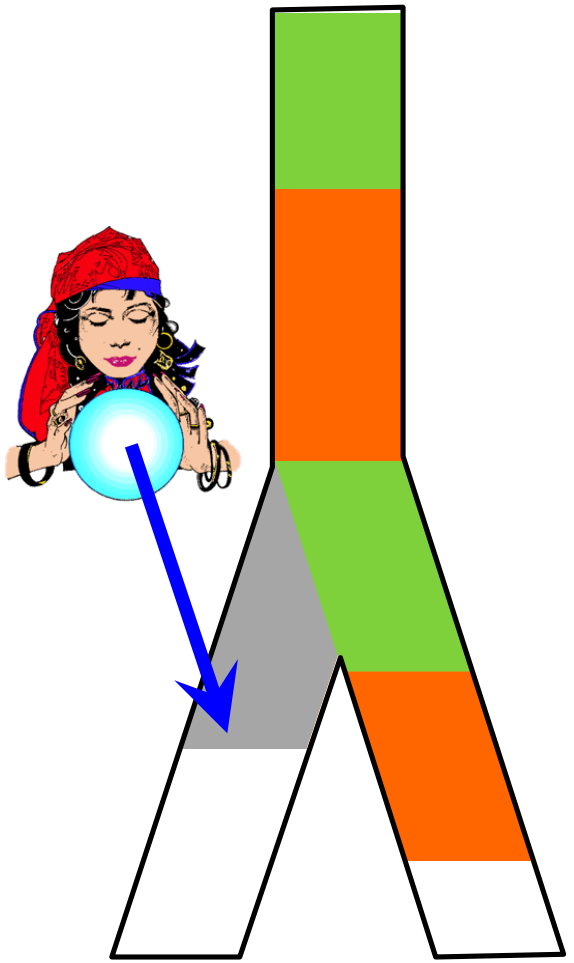


- Also useful for handling branches

Speculative Execution and Branches

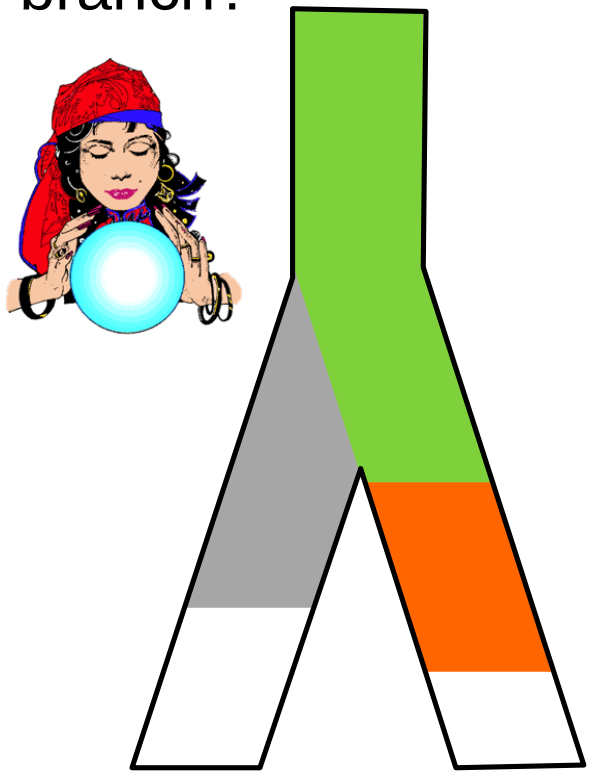
- When execution reaches a branch
- The processor predicts the outcome of the branch
- Execution proceeds (speculatively) along predicted branch

- **Correct prediction** → all is well
- **Misprediction** → abandon and resume

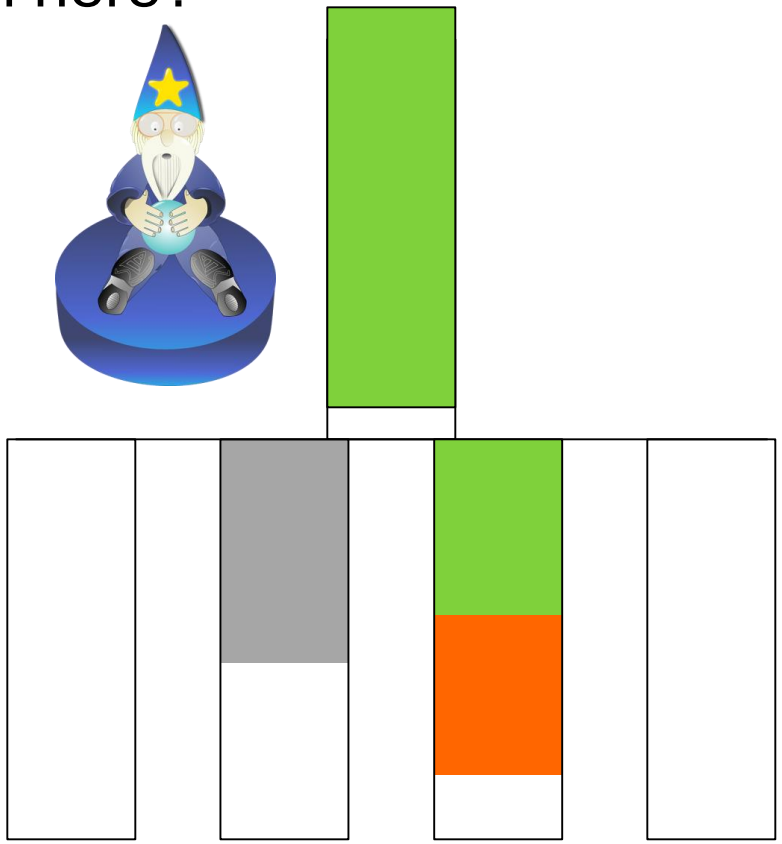


Branch Prediction

- 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?

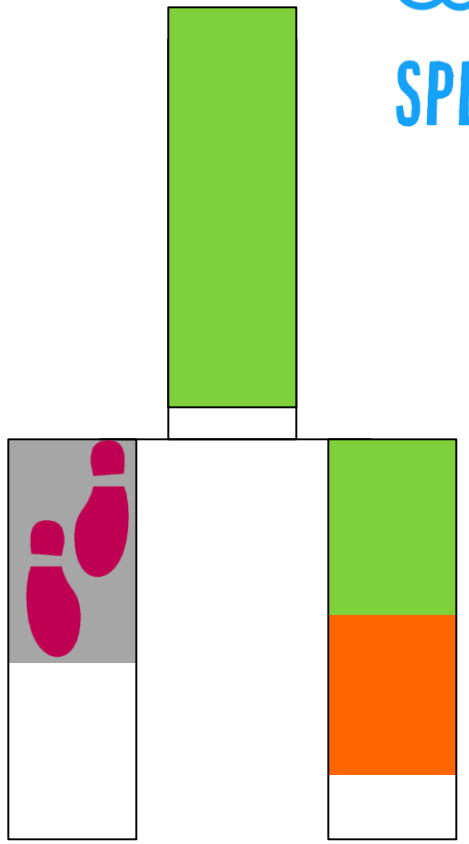


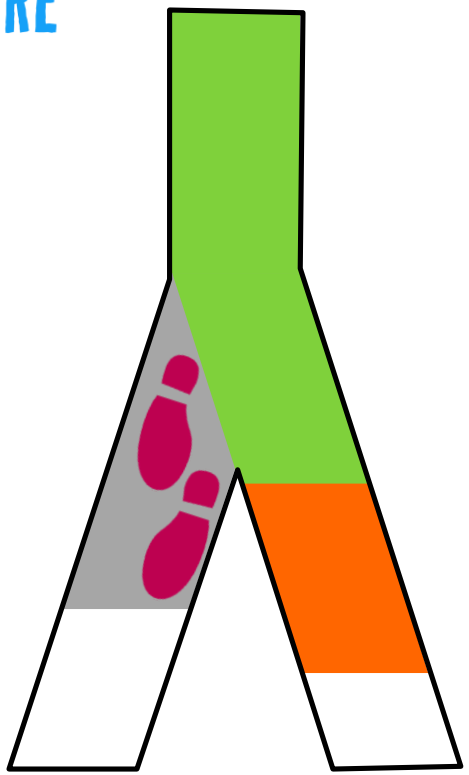
Main Discovery

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

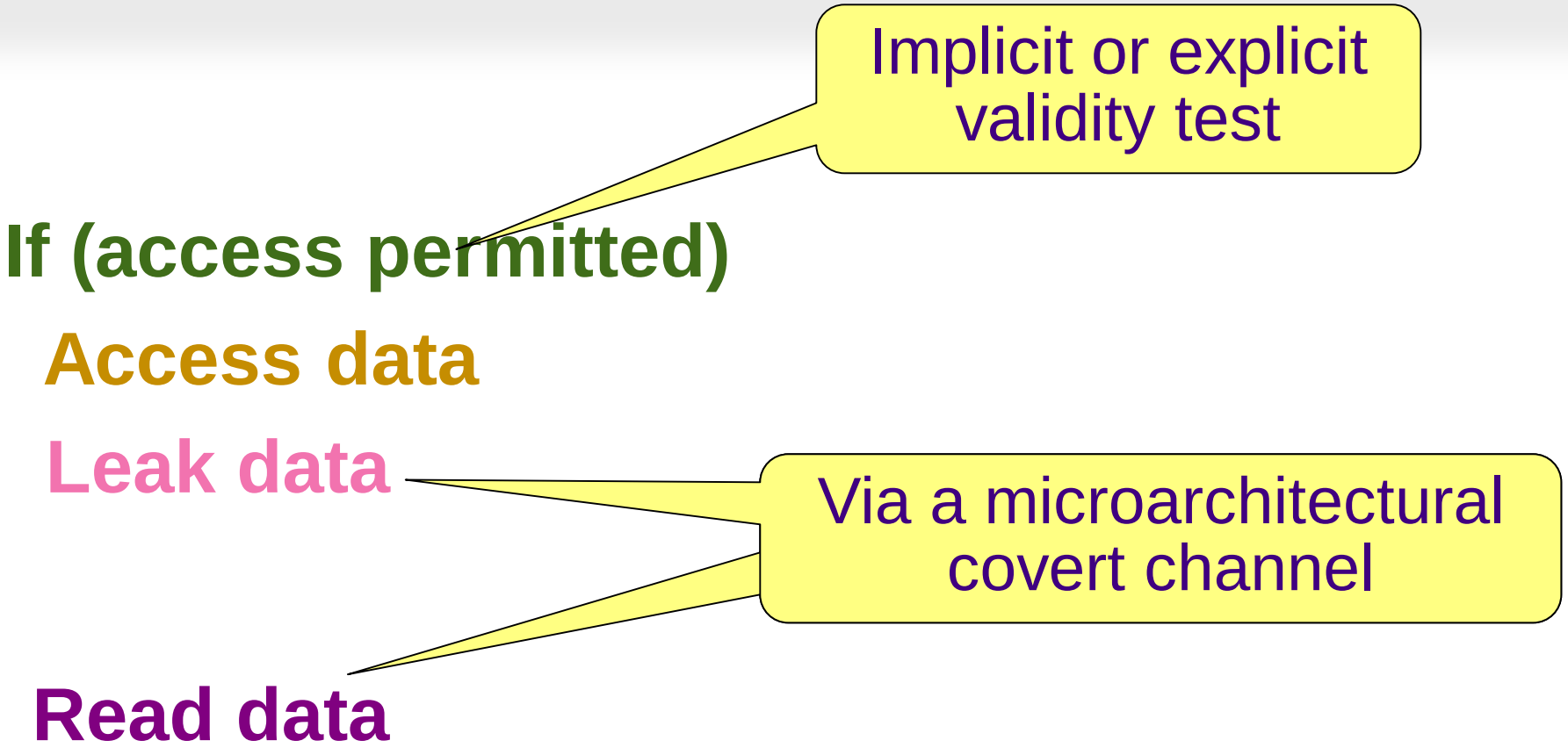
MELTDOWN

SPECTRE





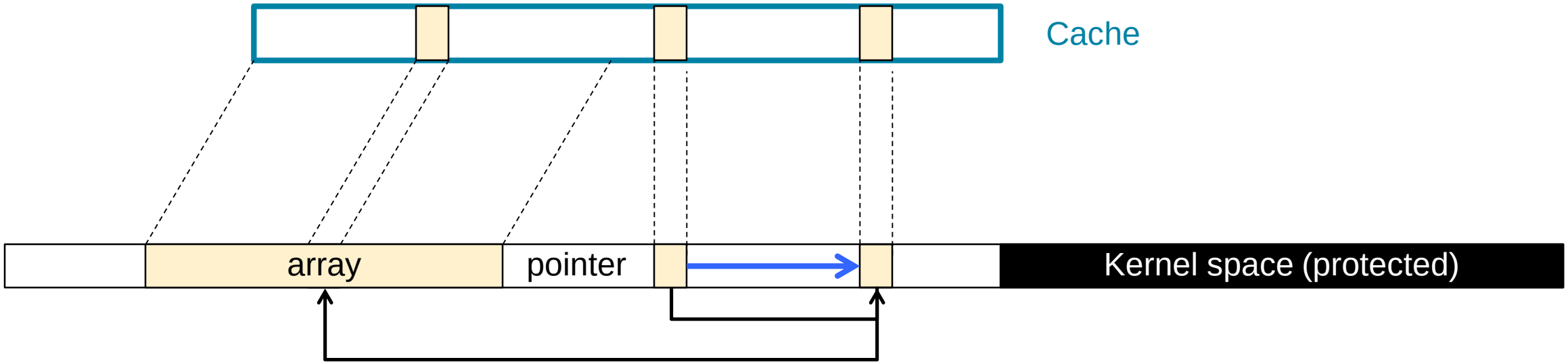
Attack overview



Meltdown

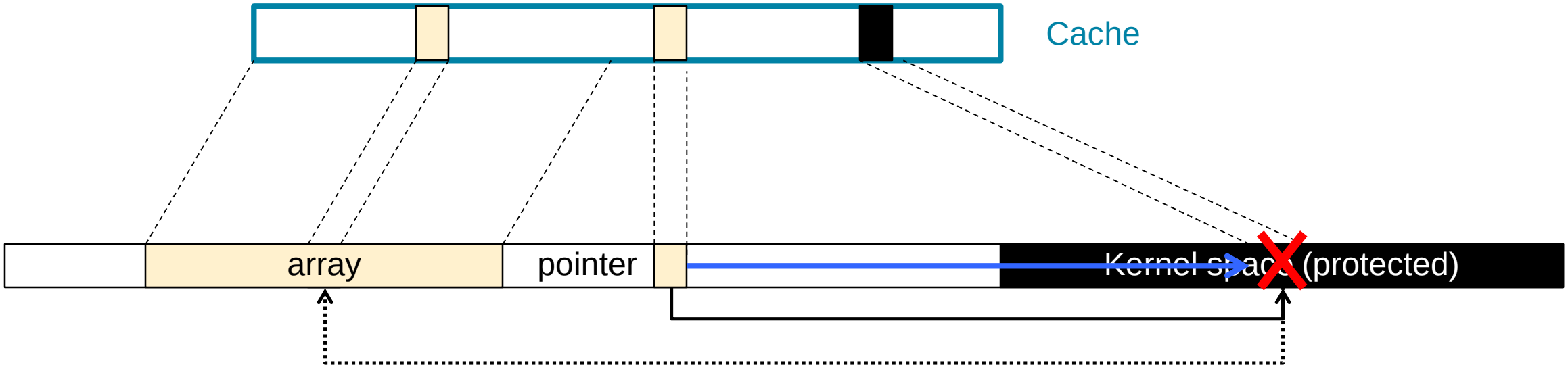
Meltdown

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

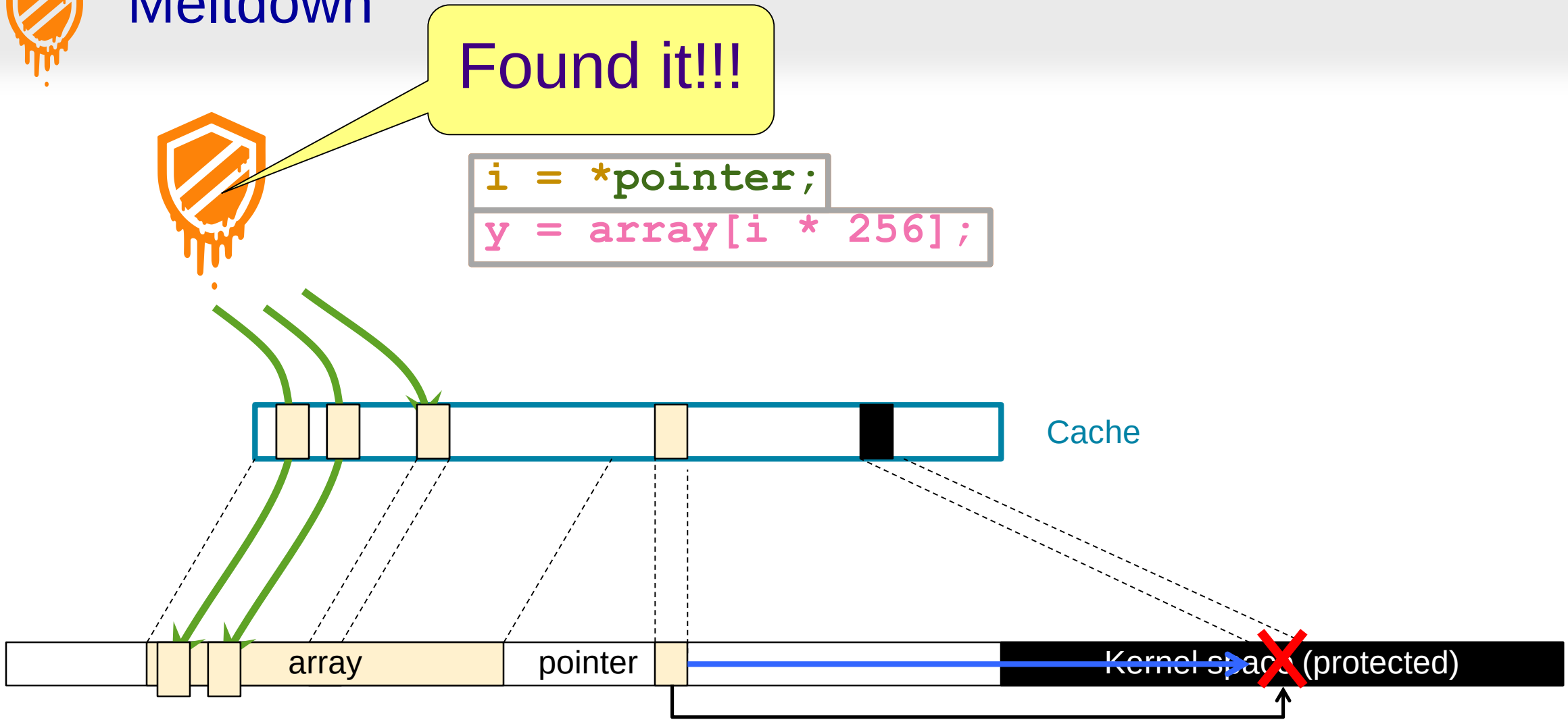


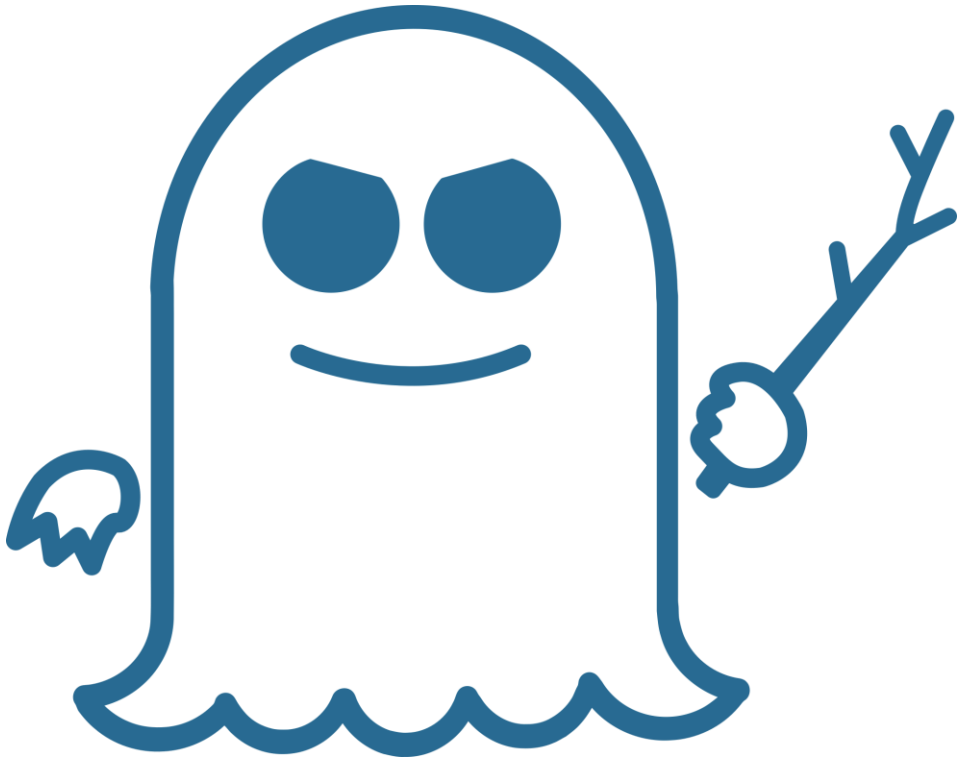
Meltdown

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



Meltdown


Found it!!!
i = *pointer;
y = array[i * 256];
Cache
array
pointer
Kernel space (protected)



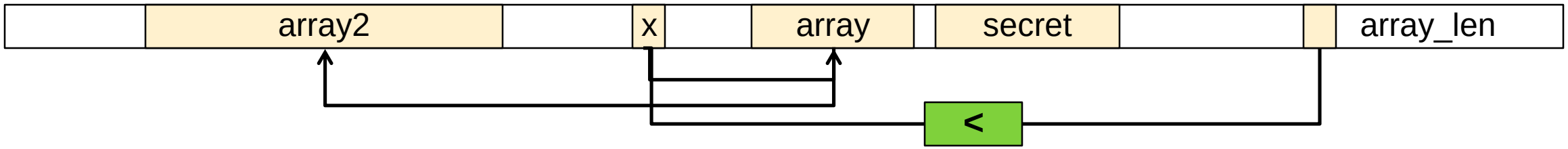
Spectre

Variant 1

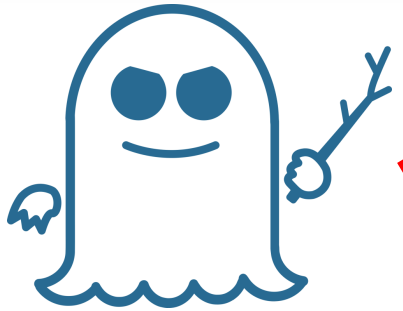
Spectre (Variant 1)

Victim


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



Spectre (Variant 1)

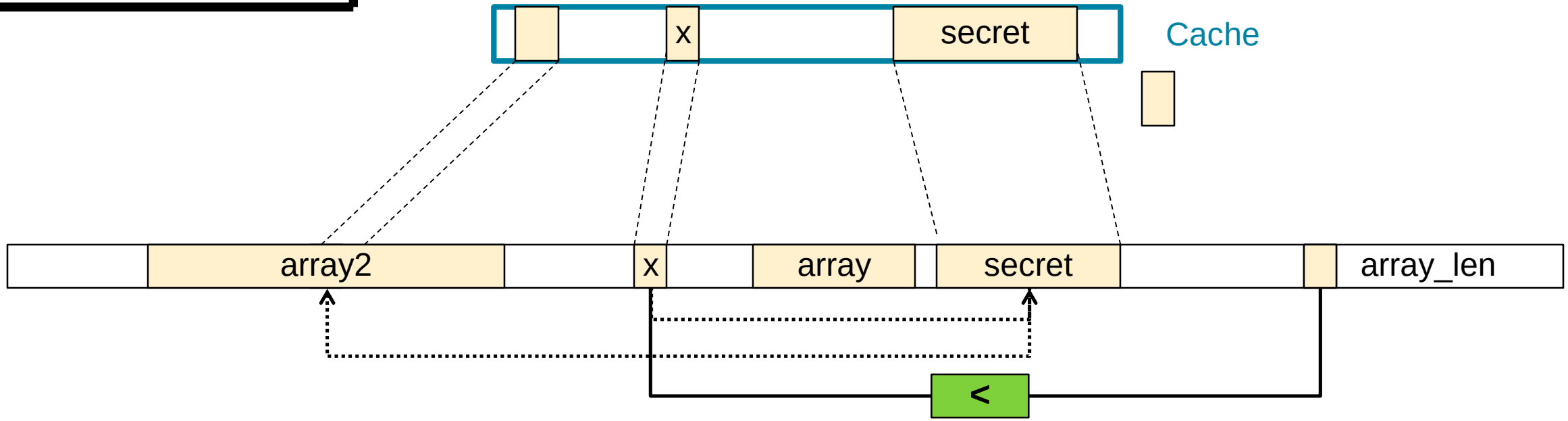


Attacker

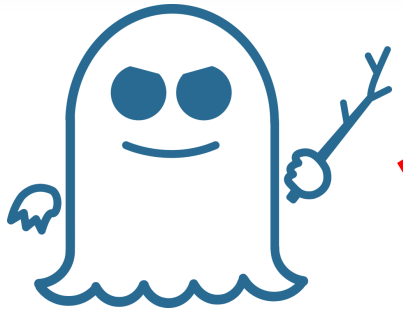
Branch not taken!

X is large

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



Spectre (Variant 1)

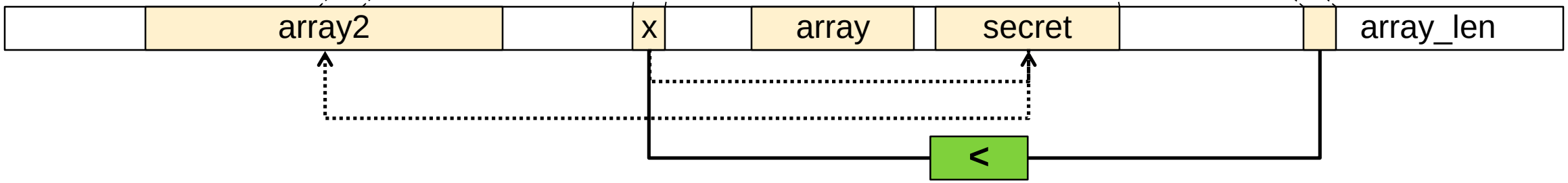


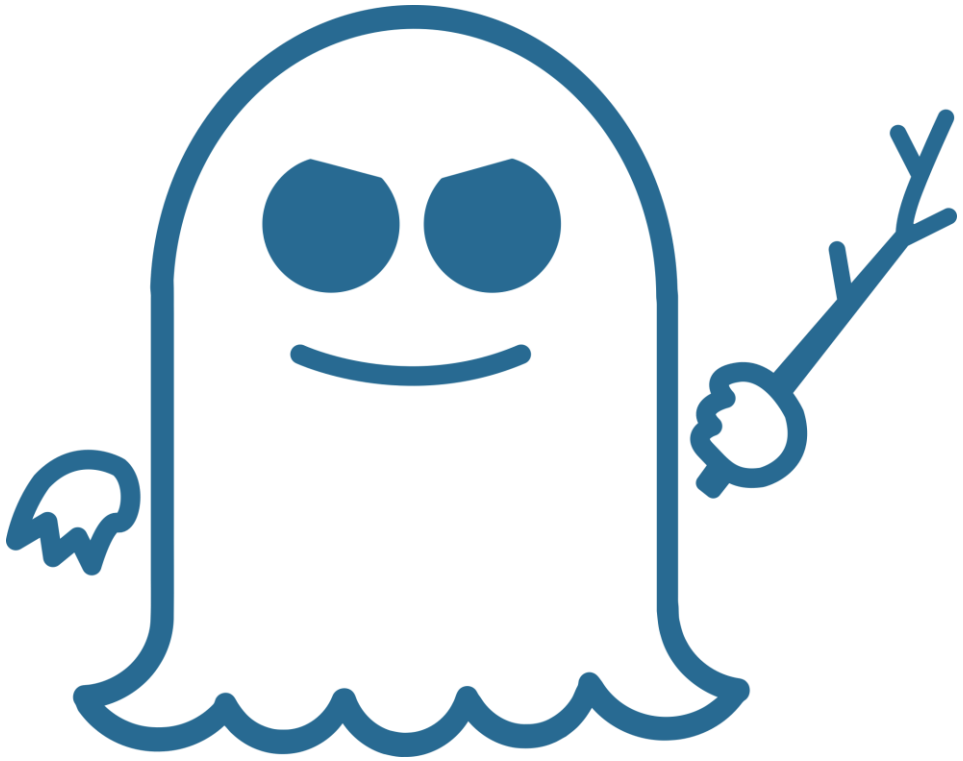
Attacker

Mispredict

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

Cache



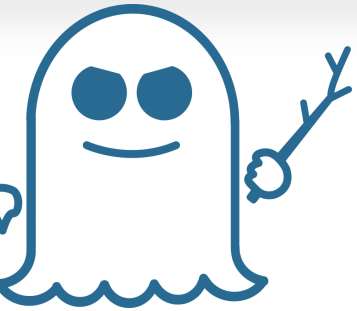


Spectre

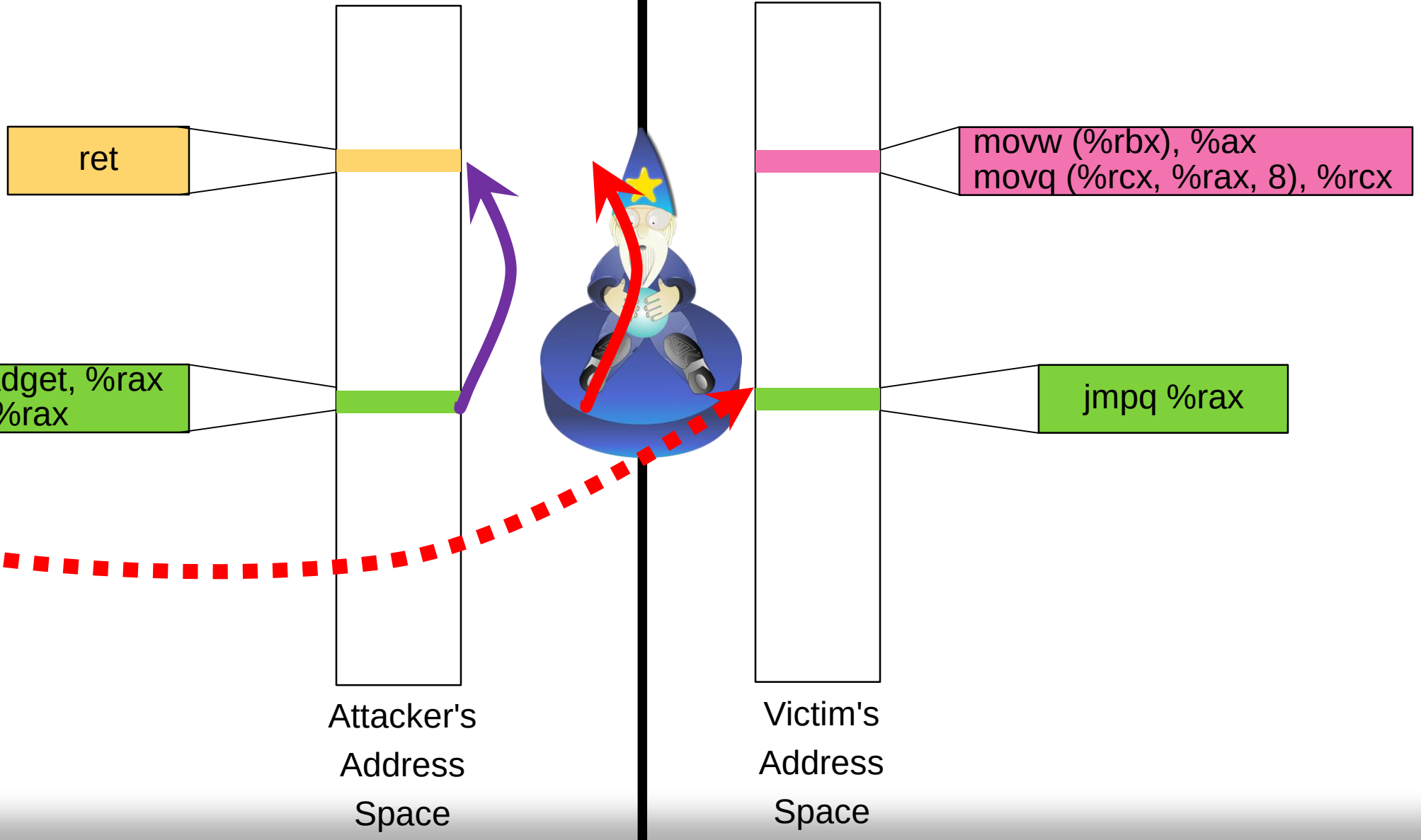
Variant 2

Spectre (Variant 2)

Victim



Attacker

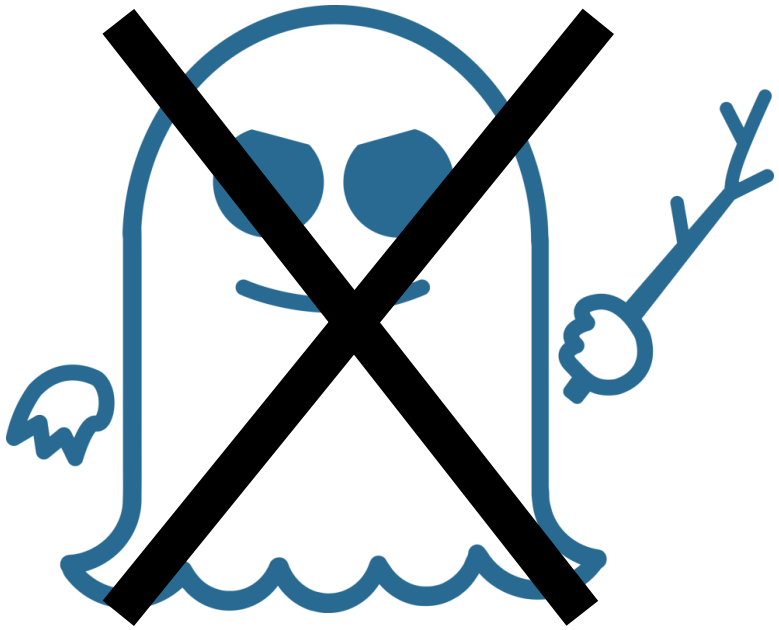


How deep does the rabbit hole go?

- Variant 3a: leak model-specific registers

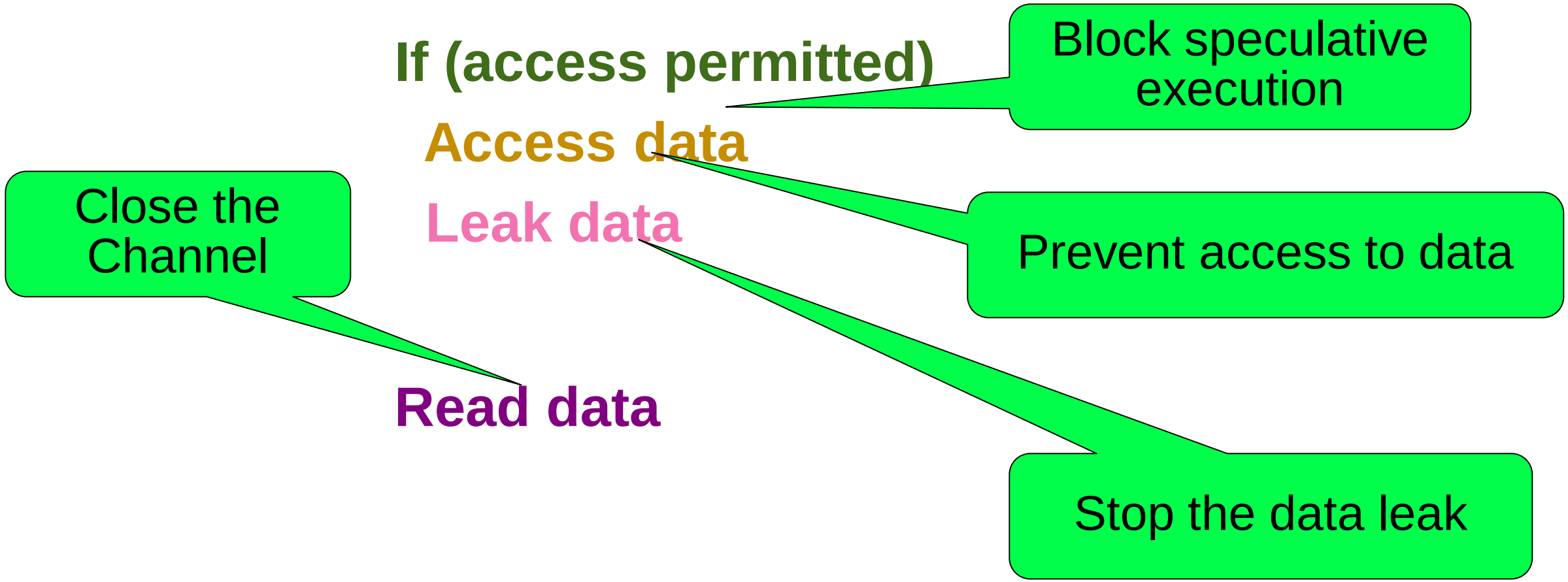
> "The processor is, in fact, operating as it is designed," Smith said. "And in every case, it's been this side-channel approach that the researchers used to gain information even while the processor is executing normally its intended functions."

- Fallout: read data from the store buffer



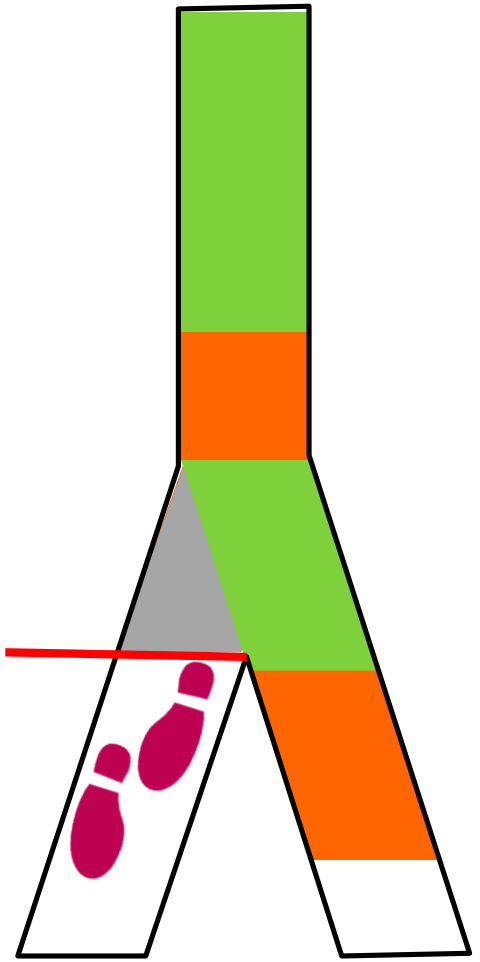
Countermeasures

Countermeasures



Spectre – Variant 1

- Compiler patches to block speculative execution
- Barriers at eve
  - Use static ana



Spectre – Variant 2

- New MSRs to prevent BTB
  - No
  - Hi
    lea
  - BT

PCWorld FROM IDG

SUBSCRIBE

NEWS | REVIEWS | HOW-TO | VIDEO

BUSINESS | LAPTOPS | TABLETS | PHONES | HARDWARE | SECURITY | SOFTWARE | GADGETS

Privacy | Encryption | Antivirus

NEWS

Microsoft issues emergency Windows patch to disable Intel's buggy Spectre fix

If your Windows PC seems buggier than usual after the recent round of Spectre patches, you might want to download this.

By Mark Hachman

Senior Editor, PCWorld | JAN 28, 2018 11:40 PM PT

Meltdown

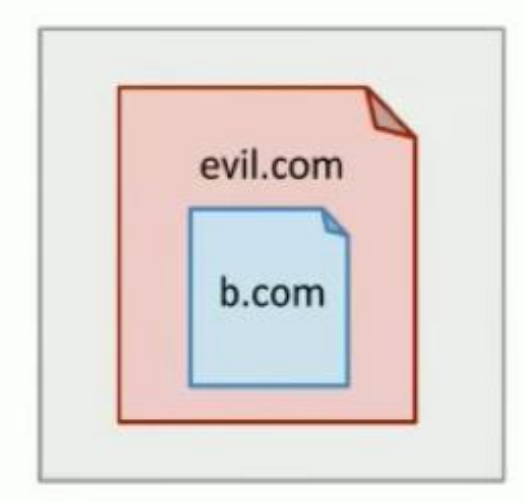
- Use a sep e kernel
  - Overh e

ted)

Kernel

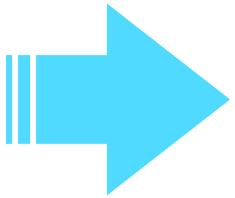
Spectre – Variant 1

- 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