Lab: Side-Channel Attacks

Systems and Software Security Lab @ SNU
10/4/2019

Today’s Agenda

1. Understand Flush+Reload
2. Leak kernel secret via Flush+Reload

Setup

- Vagrant setup

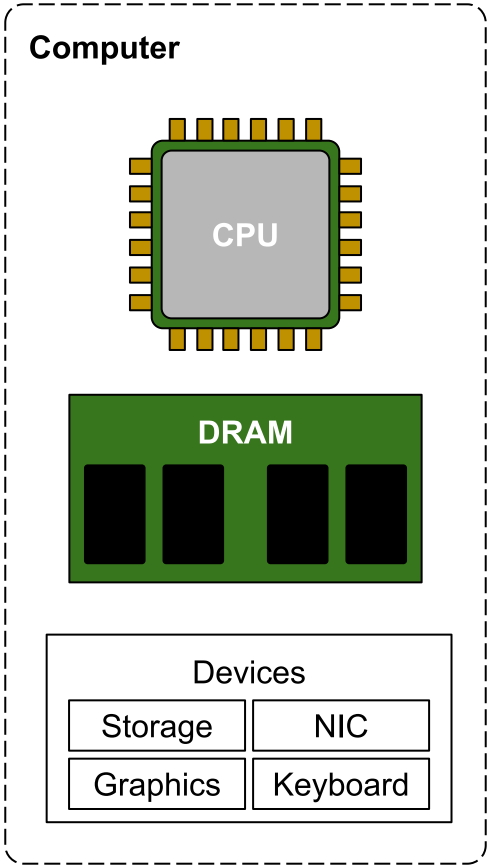
```
user@XXX:~ $ mkdir flush-reload && cd flush-reload
user@XXX:~/flush-reload $ vagrant box add ubuntu/xenial64
user@XXX:~/flush-reload $ vagrant init ubuntu/xenial64
user@XXX:~/flush-reload $ vagrant up && vagrant ssh
```

- Git repo setup

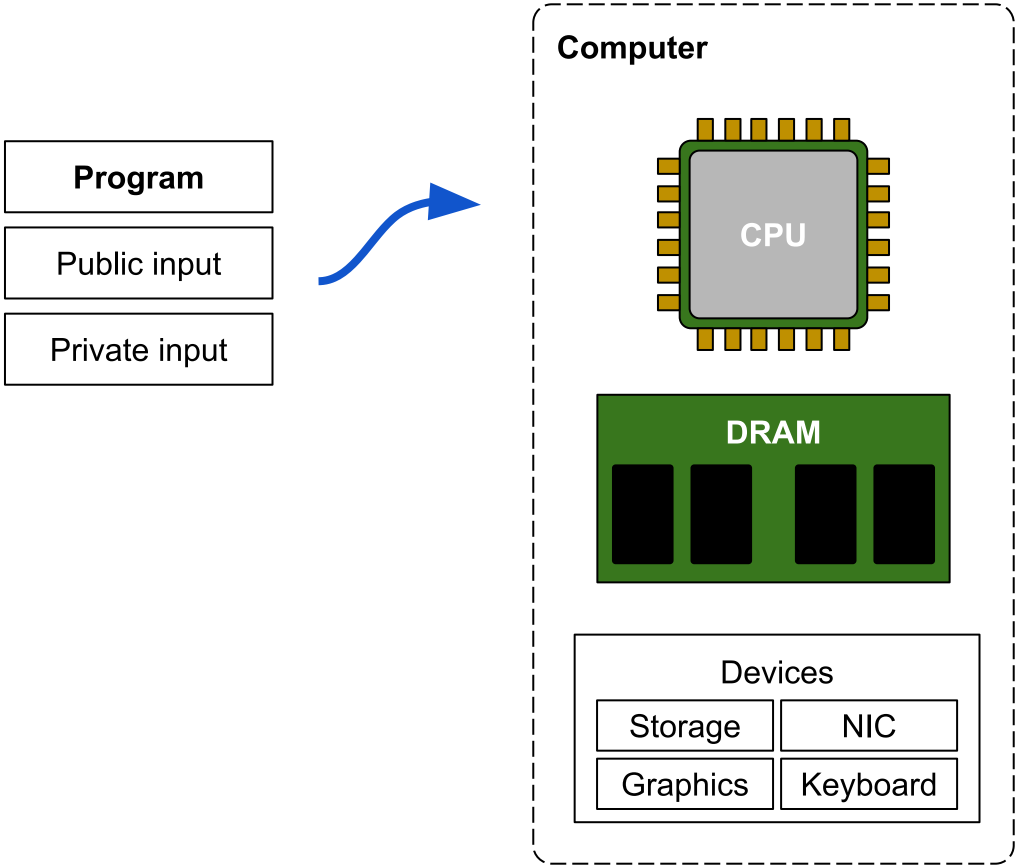
```
vagrant@XXX:~ $ cd /vagrant
vagrant@XXX:/vagrant $ git clone https://github.com/shpark/sca-public
vagrant@XXX:/vagrant $ cd sca-public
vagrant@XXX:/vagrant/csa-public $ # You’re ready!
```

What Are Side-Channel Attacks?

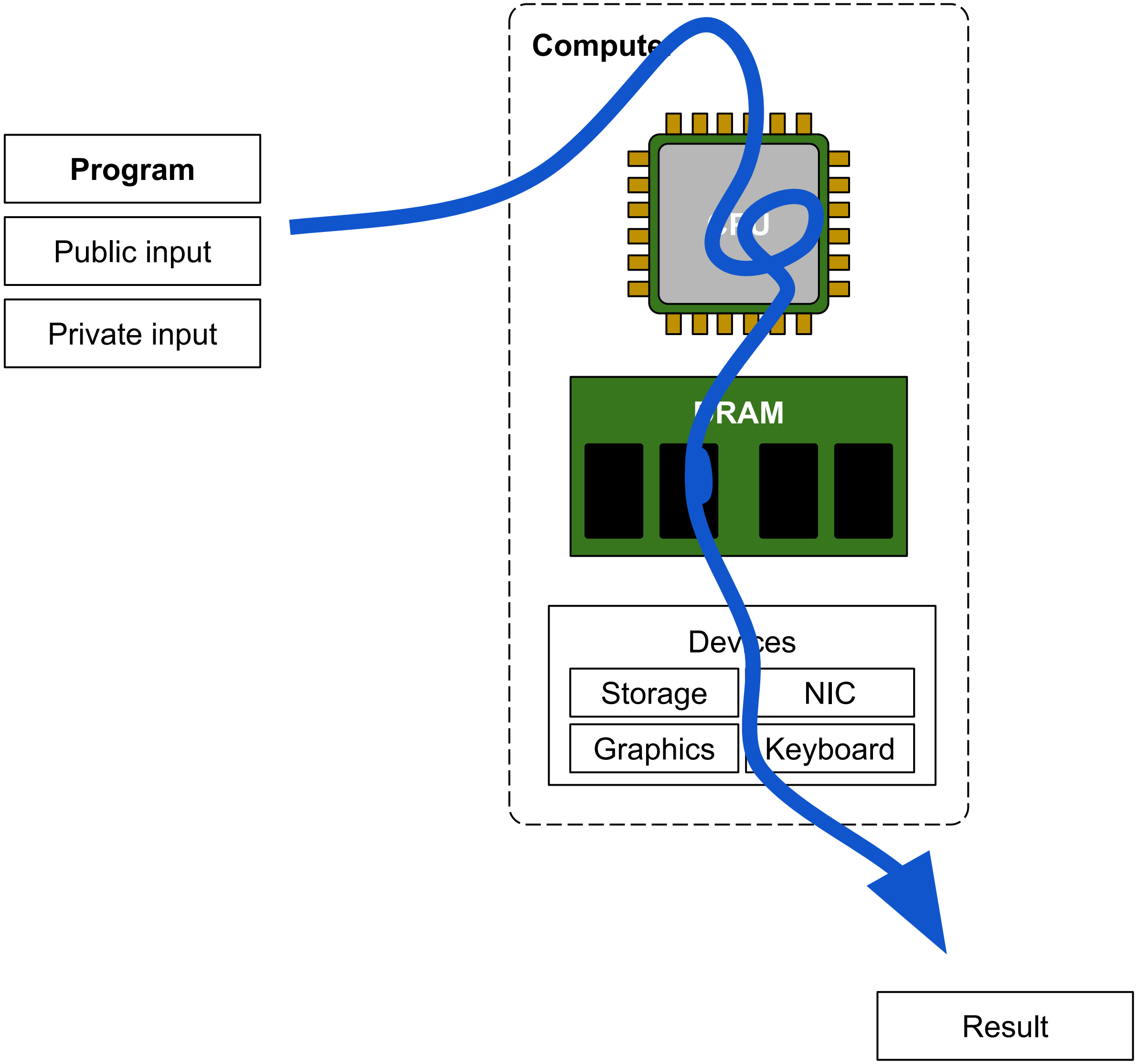
Side-Channels?



Side-Channels?



Side-Channels?



Program

Public input

Private input

Computer

CPU

DRAM

Devices

Storage

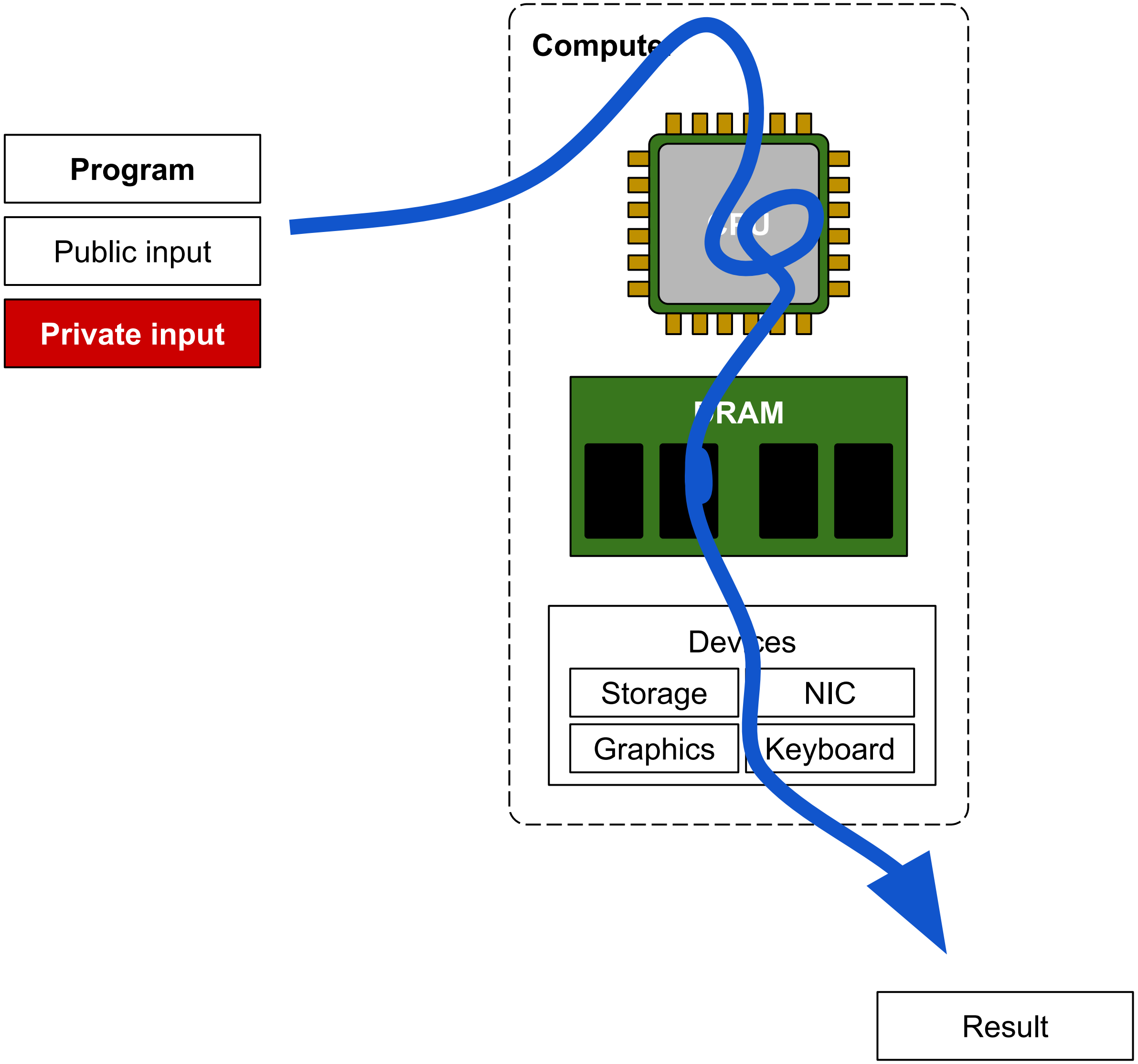
NIC

Graphics

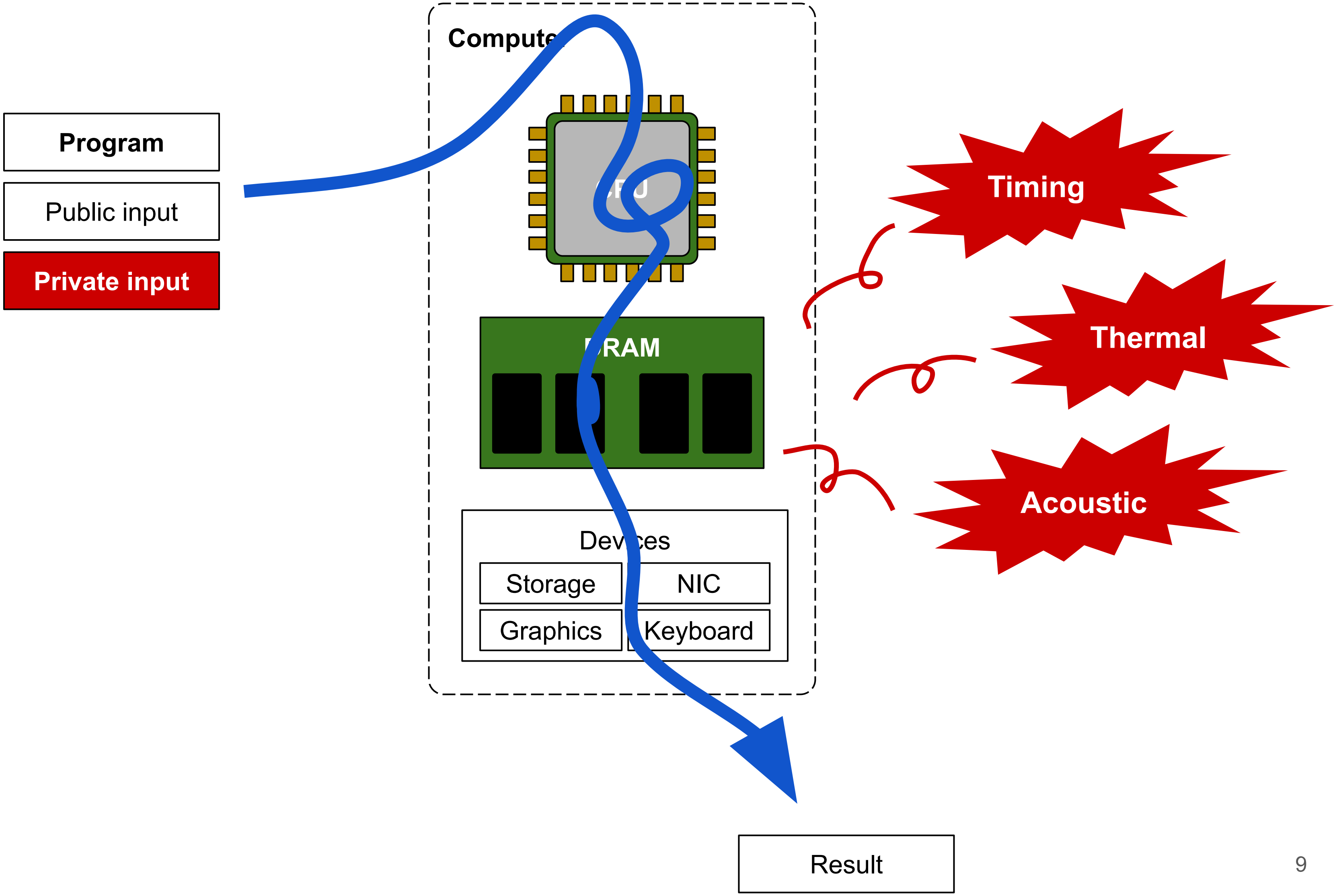
Keyboard

Result

Side-Channels?



Side-Channels?



Program

Public input

Private input

Computer

CPU

DRAM

Devices

Storage

NIC

Graphics

Keyboard

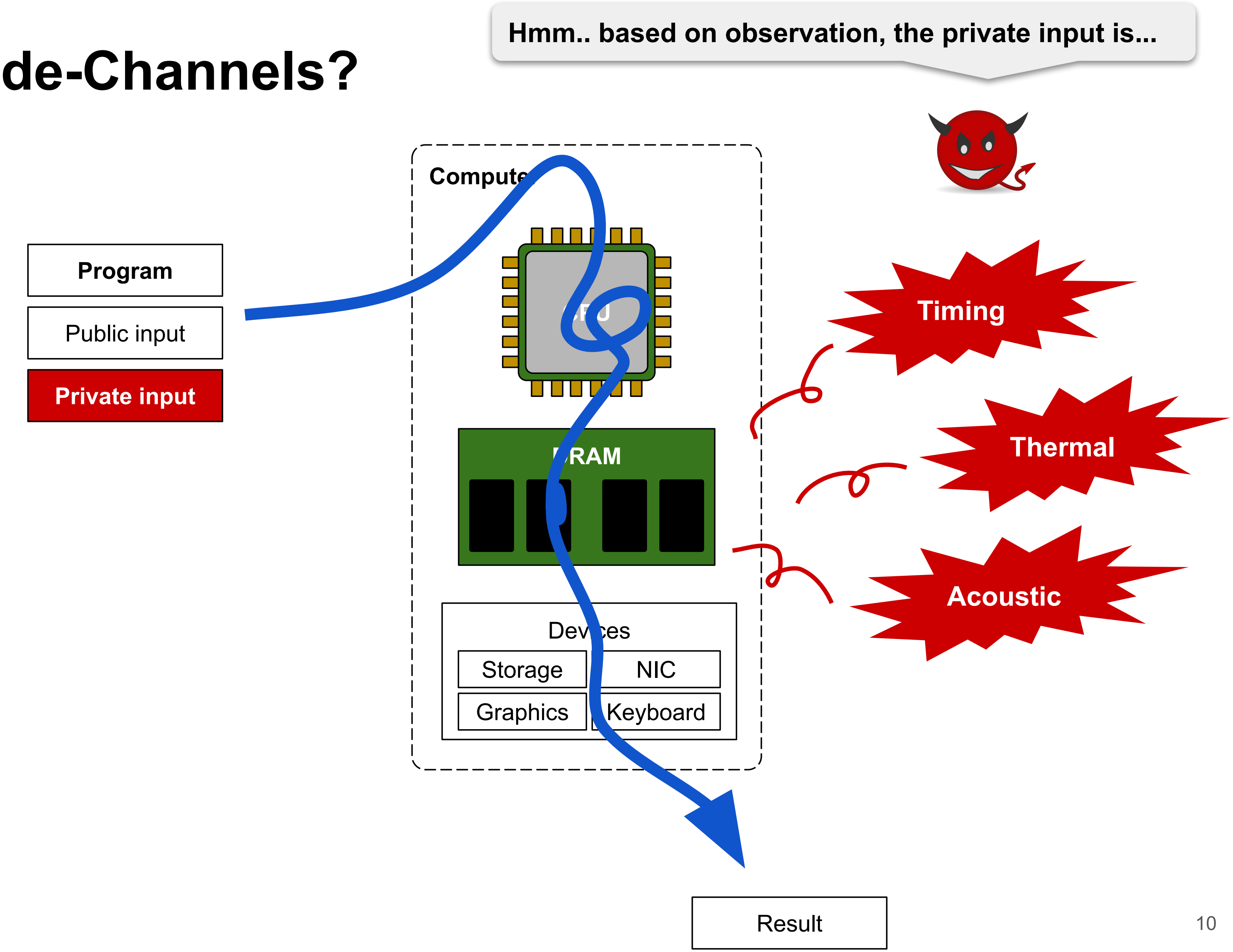
Timing

Thermal

Acoustic

Result

Side-Channels?



Hmm.. based on observation, the private input is...

Program

Public input

Private input

Compute

CPU

DRAM

Devices

Storage

NIC

Graphics

Keyboard

Timing

Thermal

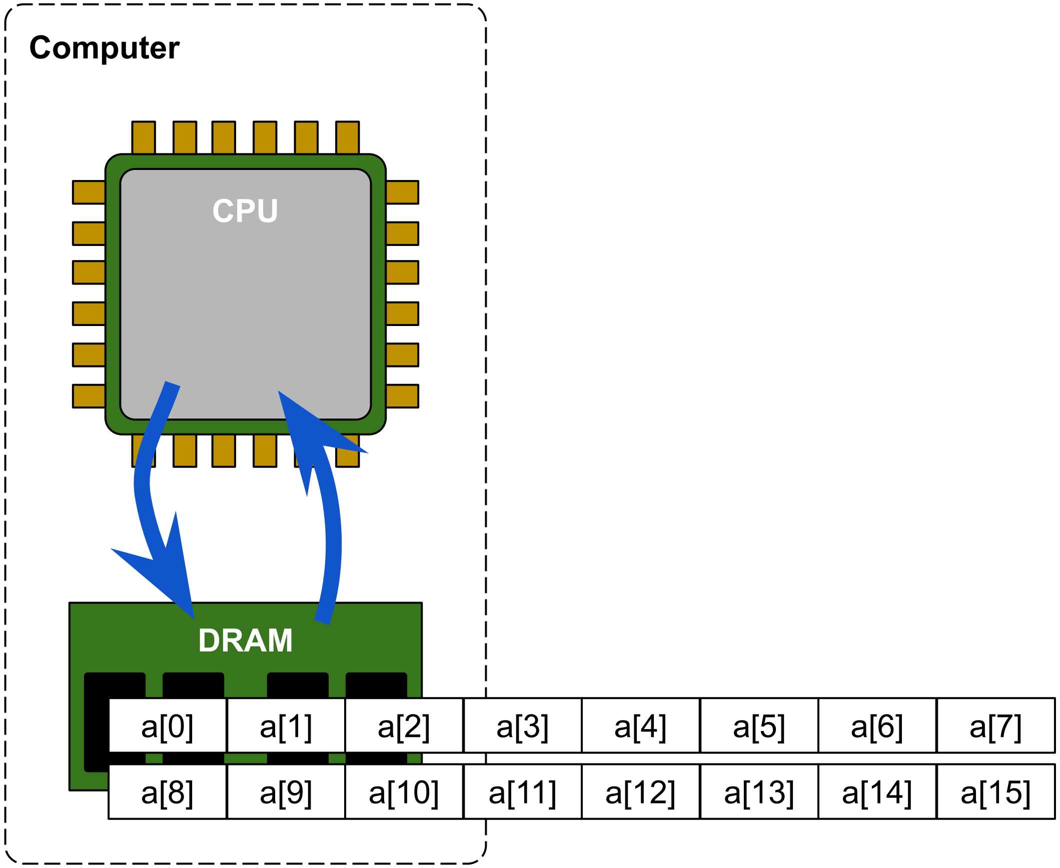
Acoustic

Result

Closer Look at Cache Side-Channels

Primer: Cache

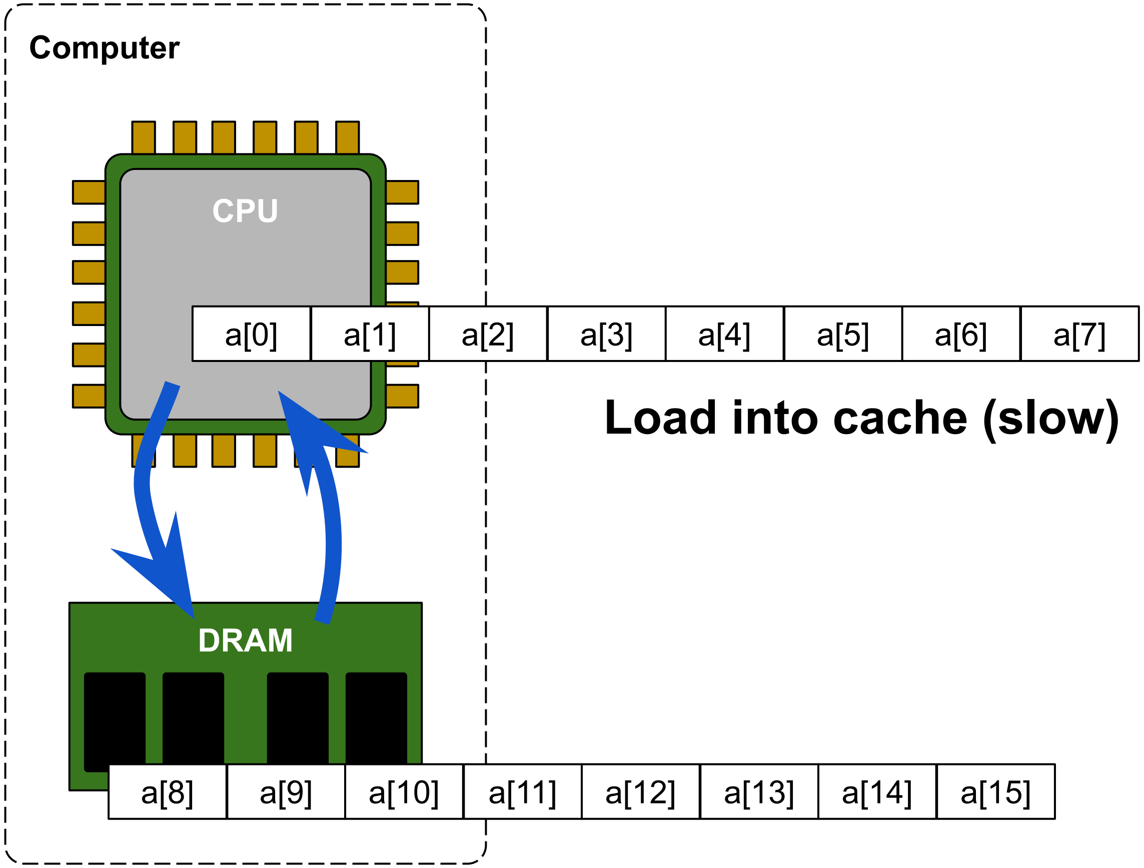
```
int a[32] = {0};
int y;

printf(“%d\n”, a[0]);
a[0] = 0x1337;
a[1] = 0xdeadbeef;
```



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]

Primer: Cache

```
int a[32] = {0};
int y;

printf(“%d\n”, a[0]); 🐢
a[0] = 0x1337;
a[1] = 0xdeadbeef;
```

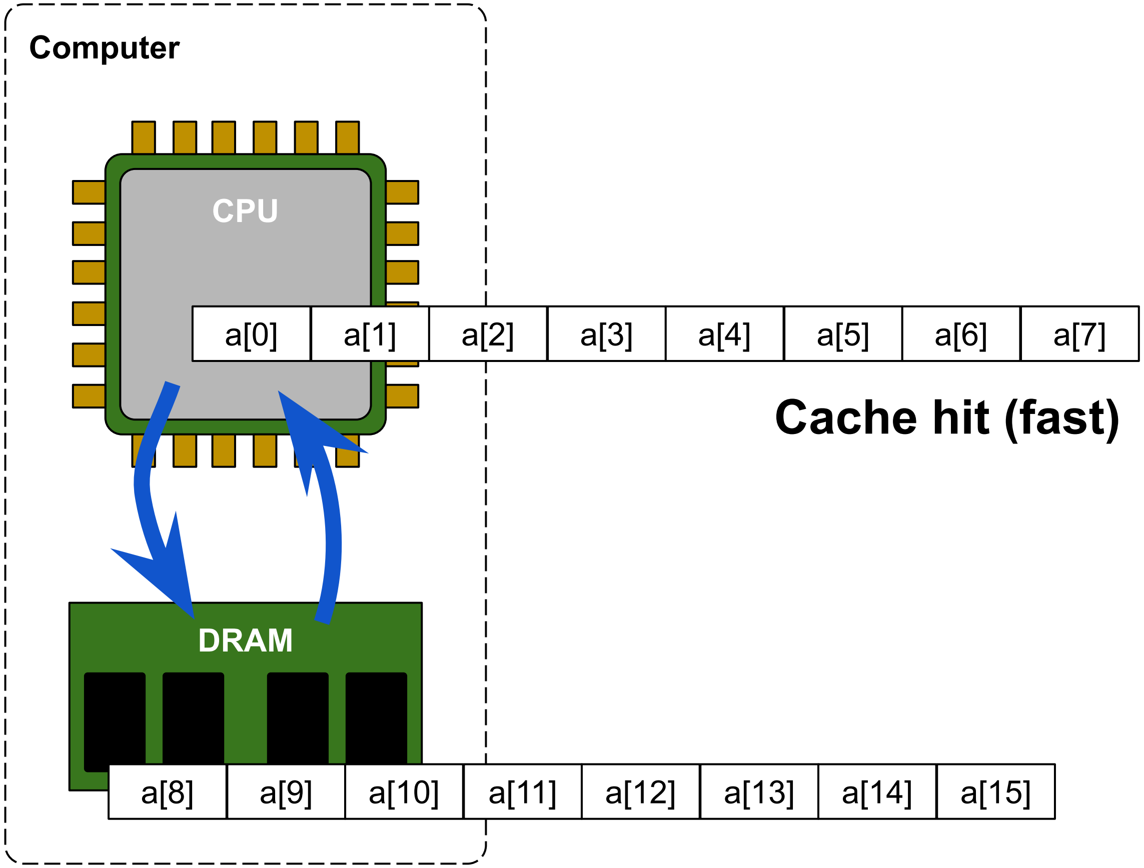


a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]

Load into cache (slow)

Primer: Cache

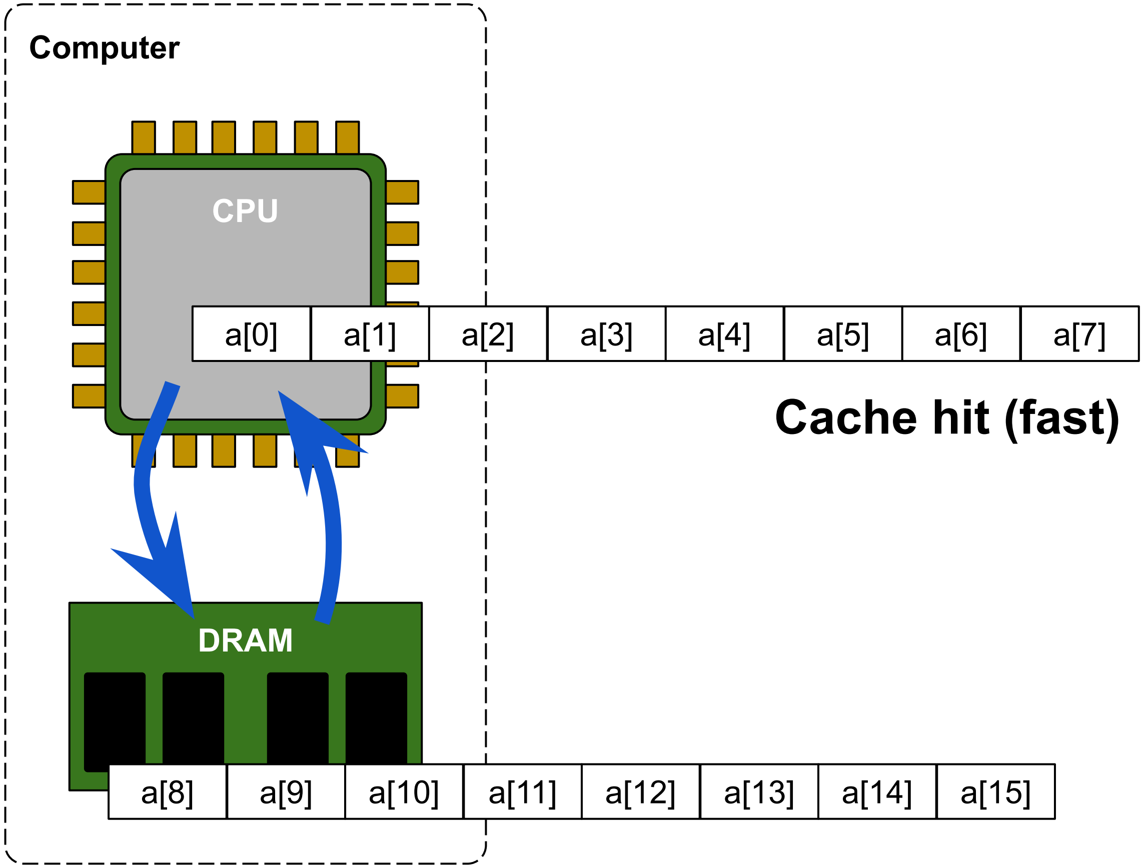
```
int a[32] = {0};
int y;

printf("%d\n", a[0]); 🐇
a[0] = 0x1337;
a[1] = 0xdeadbeef;
```



Cache hit (fast)

Primer: Cache

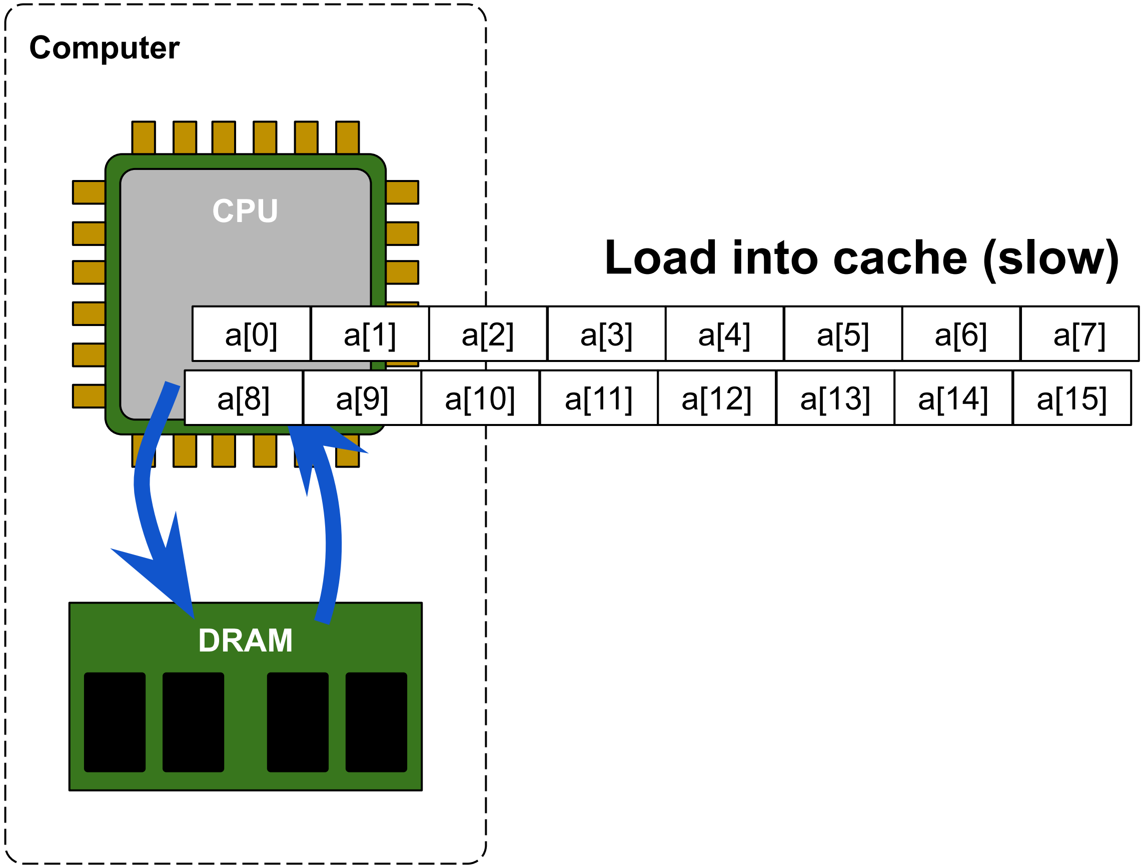
```
int a[32] = {0};
int y;

printf(“%d\n”, a[0]);
a[0] = 0x1337;
a[1] = 0xdeadbeef;  🐇
```



Cache hit (fast)

Primer: Cache

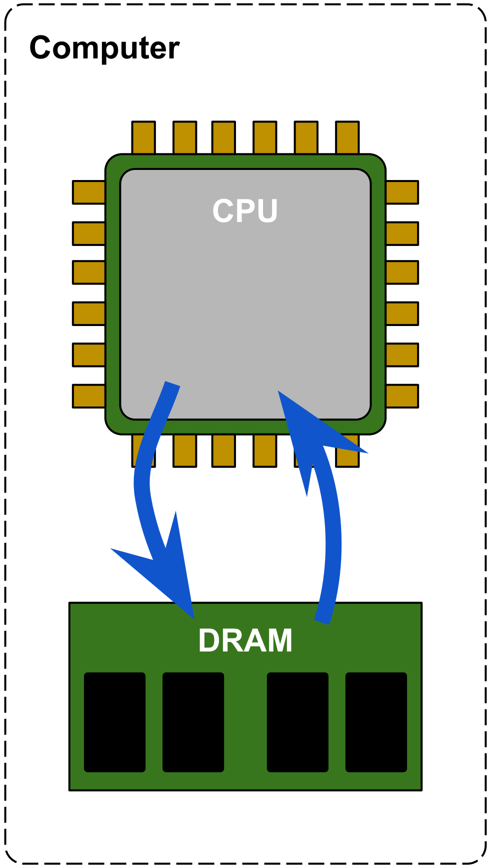
```
int a[32] = {0};
int y;

printf("%d\n", a[0]);
a[0] = 0x1337;
a[1] = 0xdeadbeef;
a[10] = 42;
```



Load into cache (slow)

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]

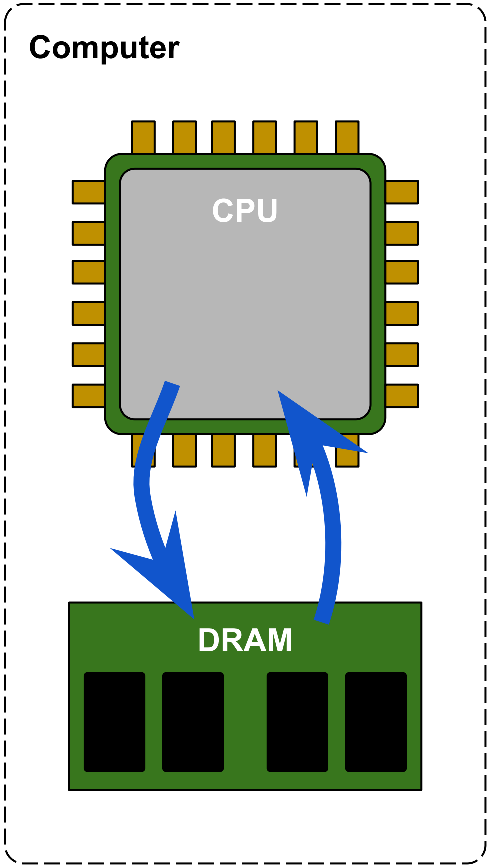
Cache Attack



So far so good.
But what is the attack? 🤔

Cache Attack: Flush+Reload

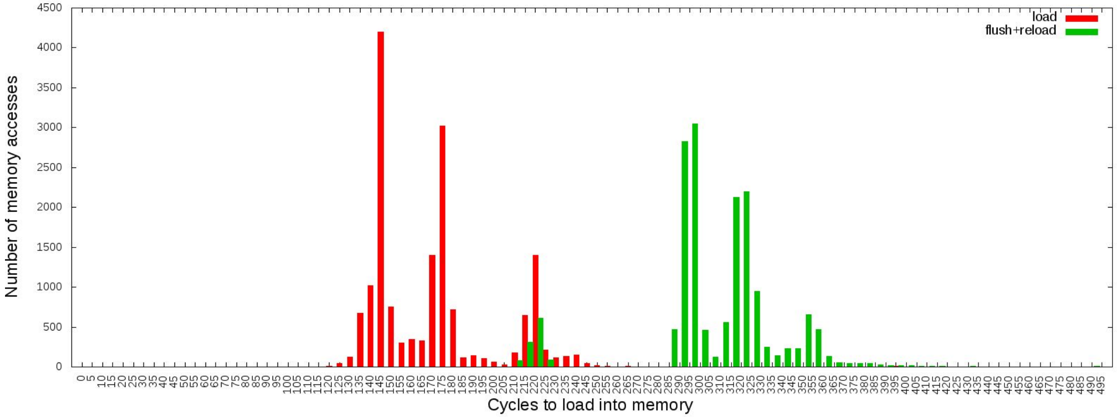
```
// victim.c
int a[32];
int secret;

// get secret

if (secret % 2 == 0) {
  a[0] = 0x1337;
} else {
  a[8] = 42;
}
```



```
// spy.c
int *a_ptr;

// set up a_ptr

// Flush cache lines
// in advance
for line in cachelines:
  clflush(line);

// Wait for victim
// execution.
for line in cachelines:
  measure(line);

if (time < threshold)
  line is accessed!
```

Goals

- **Observation (task 1)**: Measure the timing differences between **Cache Hits** and **Cache misses (Flush+Reload)**



- **Attack (task 2)**: Perform Flush+Reload attack to retrieve the secret

What Are In Your Toolbox?

```
// flush the cache line
void clflush(const void* addr);

// access the cach line (fetch the cache line)
void load(const void* addr);

// measure the time to access the cache line
Uint64_t measure(unit8* addr);
```

Task 1: Hint

```
unsigned char oracle[];
unsigned char *addr = oracle;

int main() {
  // flush+reload
  for i in 0..num_trials {
    clflush(addr);
    time_elapsed1 = measure(addr);
    sched_yield();
  }

  // cache hit
  for i in 0..num_trials {
    time_elapsed2 = measure(addr);
  }
}
```

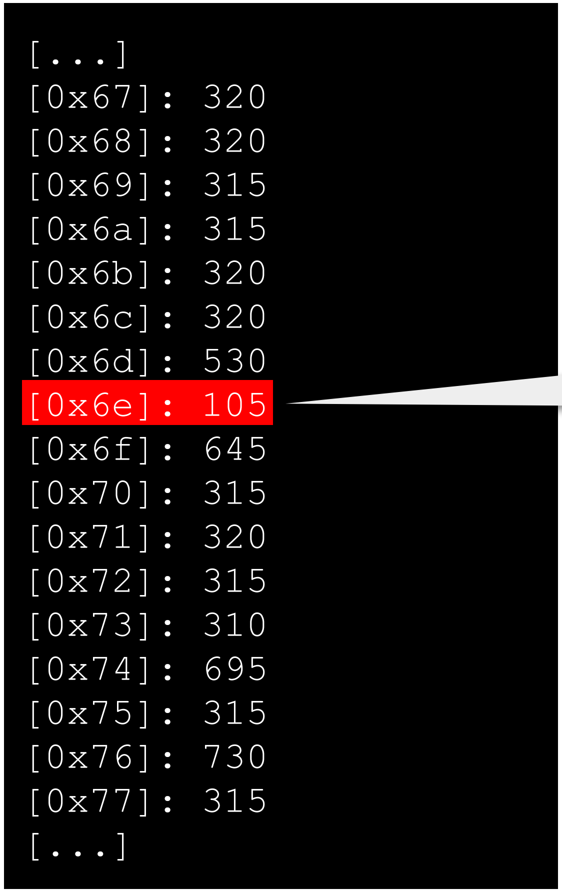
Task 2: Hint

```
unsigned char oracle[256 * 64];

int main() {
  [...]
  int index
  for c in 0..256 {
    score[c] = flush_reload(index,  c);
  }

  // Choose c with maximum (minimum) score
}
```

```
uint64_t flush_reload(int* index,  address
addr) {
  [...]
  for i in 0..num trials {
    flush(oracle + (c << 6));
    run(fd, index);
    [...] = measure(oracle + (c << 6));
  }

  return calculateScore([...]);
}
```

Can you explain why we are performing Flush+Reload at this address?

Task 2: Demo



```
[...]
[0x67]: 320
[0x68]: 320
[0x69]: 315
[0x6a]: 315
[0x6b]: 320
[0x6c]: 320
[0x6d]: 530
[0x6e]: 105
[0x6f]: 645
[0x70]: 315
[0x71]: 320
[0x72]: 315
[0x73]: 310
[0x74]: 695
[0x75]: 315
[0x76]: 730
[0x77]: 315
[...]
```

If you choose proper score (e.g. 3rd quartile?), then you will see at some point you get very small latency