CS773-2025-Spring: Computer Architecture for Performance and Security

Lecture 2: Catch the Cache

https://www.cse.iitb.ac.in/~biswa/


ON SILENT MODE PLEASE

Microarchitecture 101: World with no caches

North pole ☹

Core

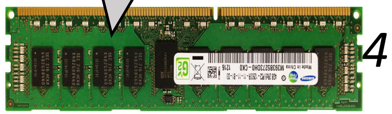
32-bit Address

Data

200 to 300 cycles

Costly DRAM accesses☹

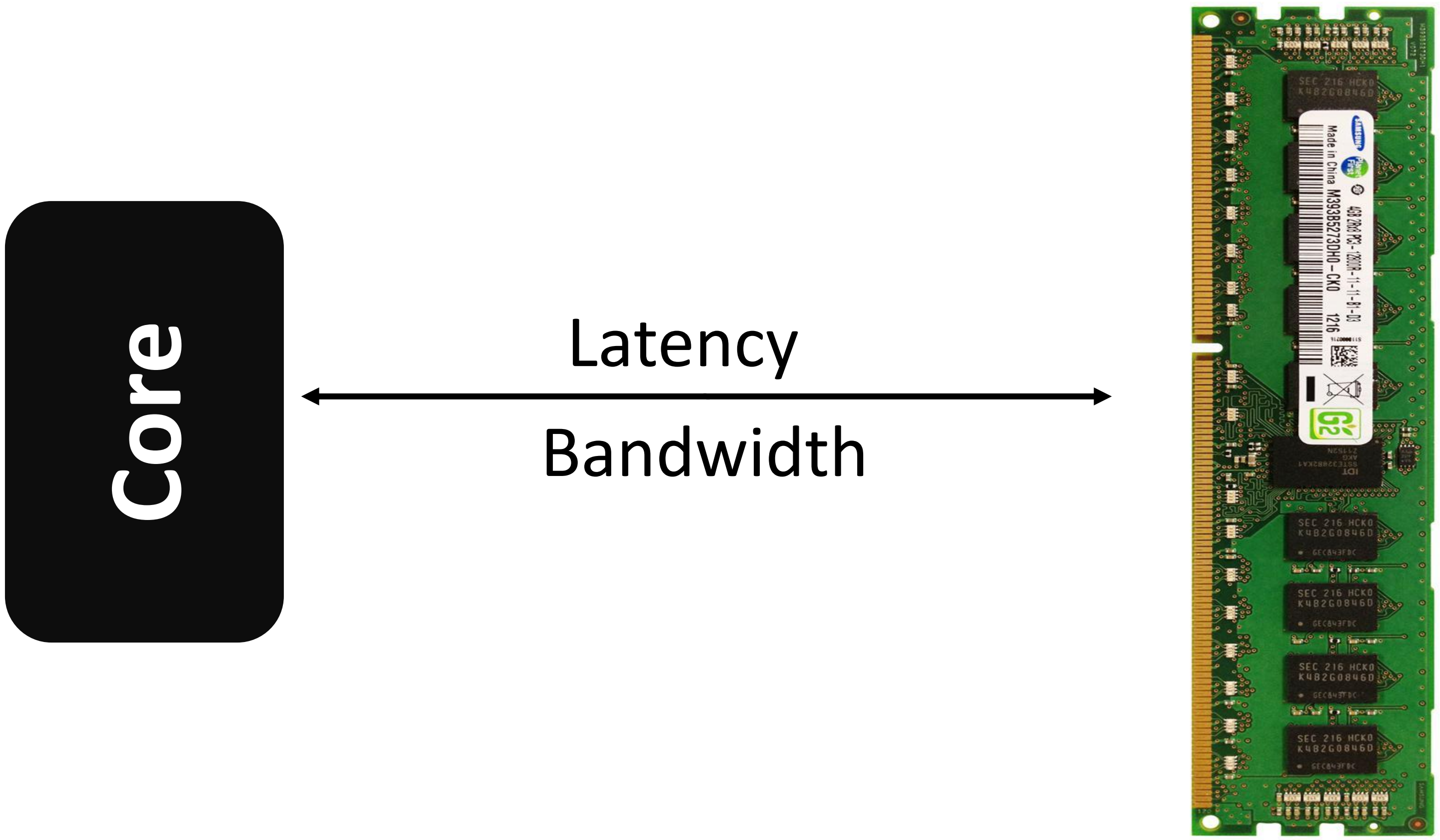
Minimizing costly DRAM accesses is critical for performance



4 GB DRAM

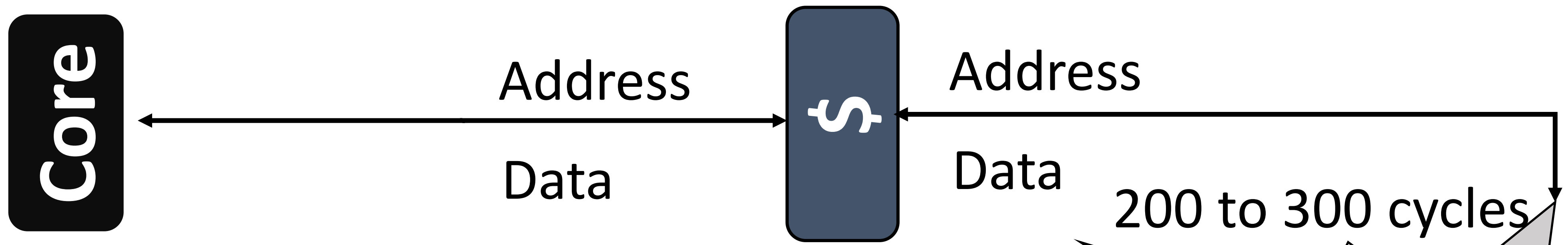
South pole ☹

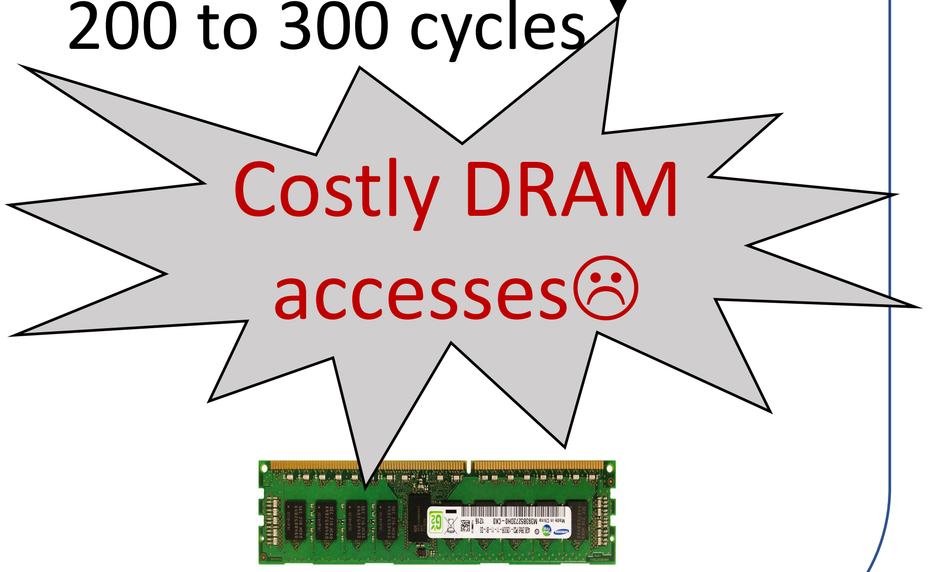
Remember Latency and Bandwidth



Caching: 10K Feet View

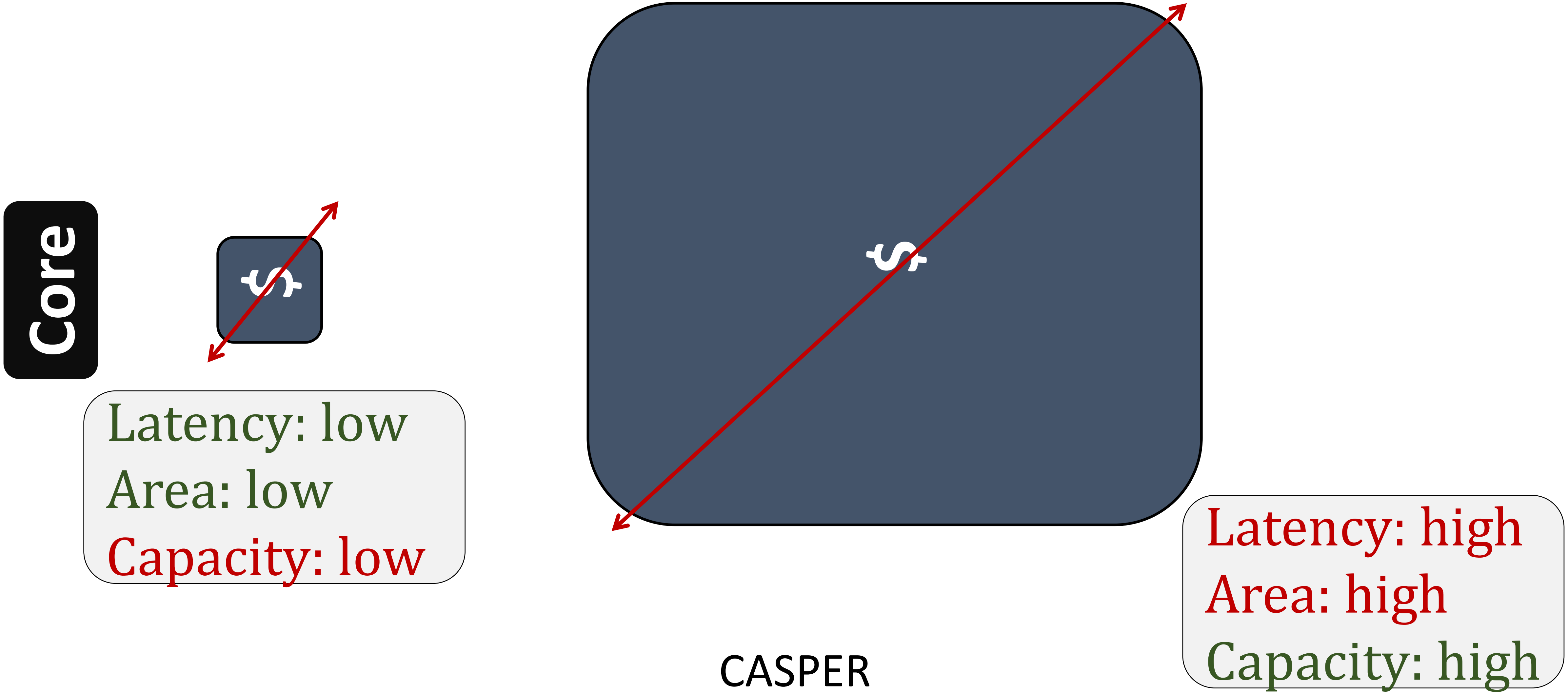
North pole ☺



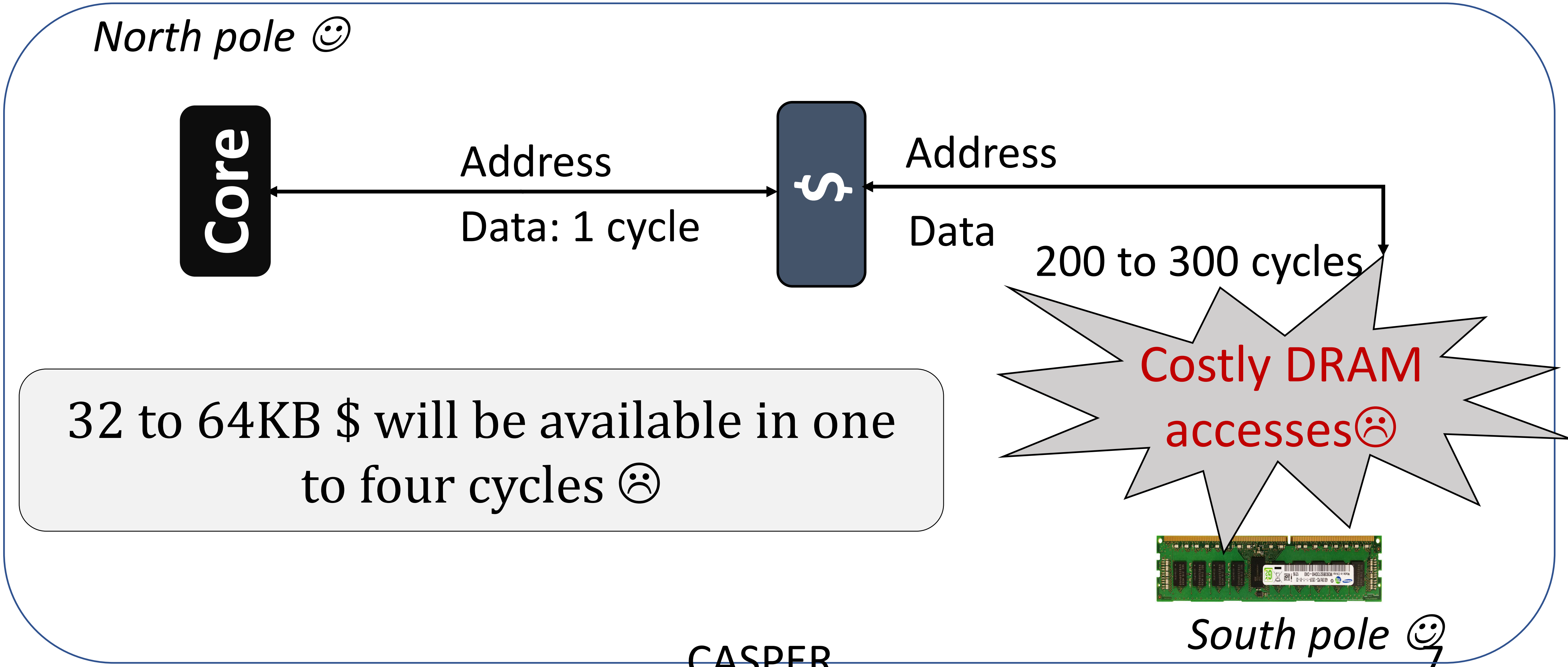


Caching is a *speculation* technique ☺
Works – if locality

How big/small?

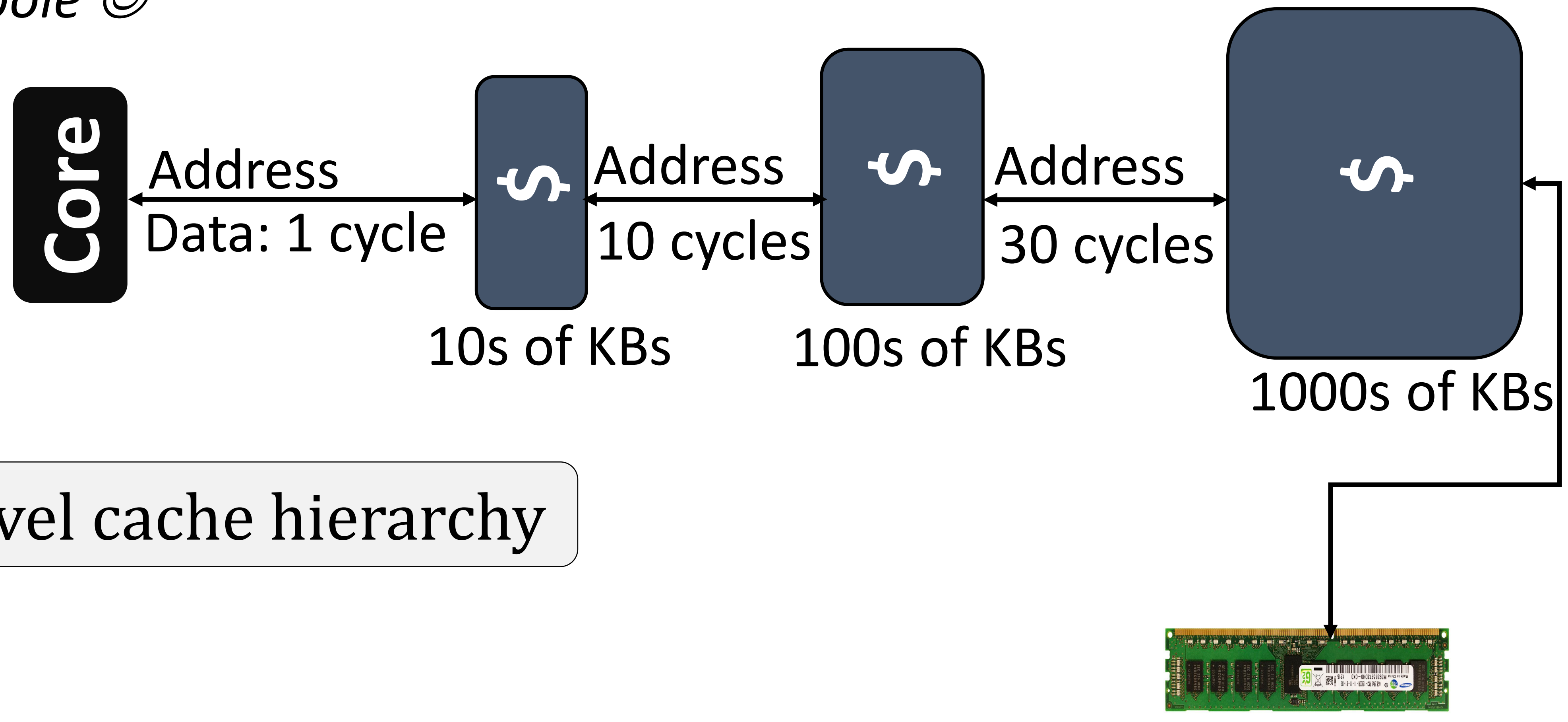


Cache with latency



Cache hierarchy with latency

North pole ☺

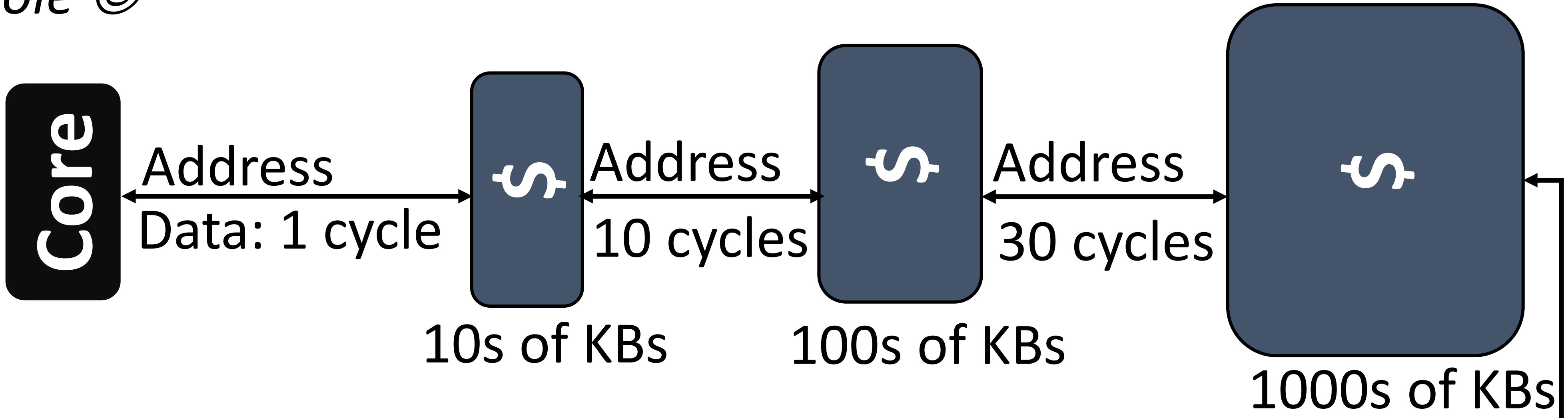


South pole ☺

Multi-level cache hierarchy

Cache hierarchy with latency

North pole ☺



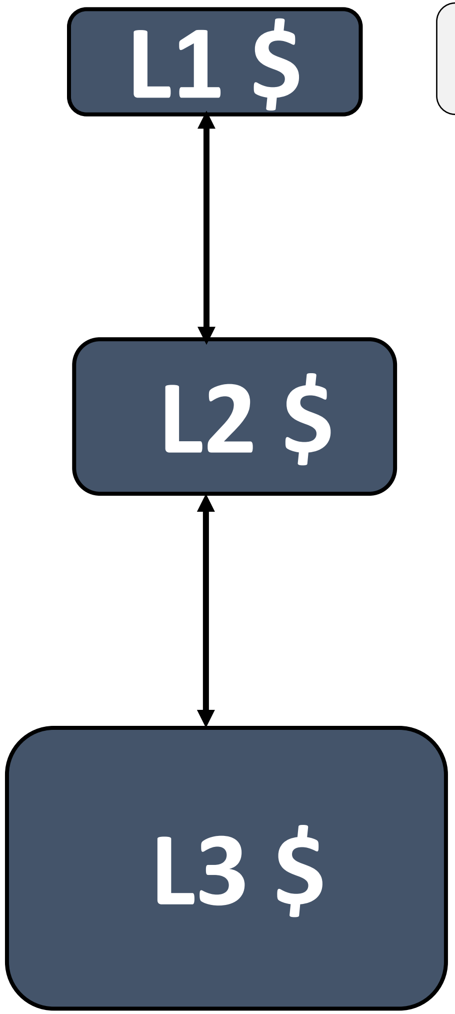
Multi-level cache hierarchy

How many levels ?

Total latency < DRAM latency

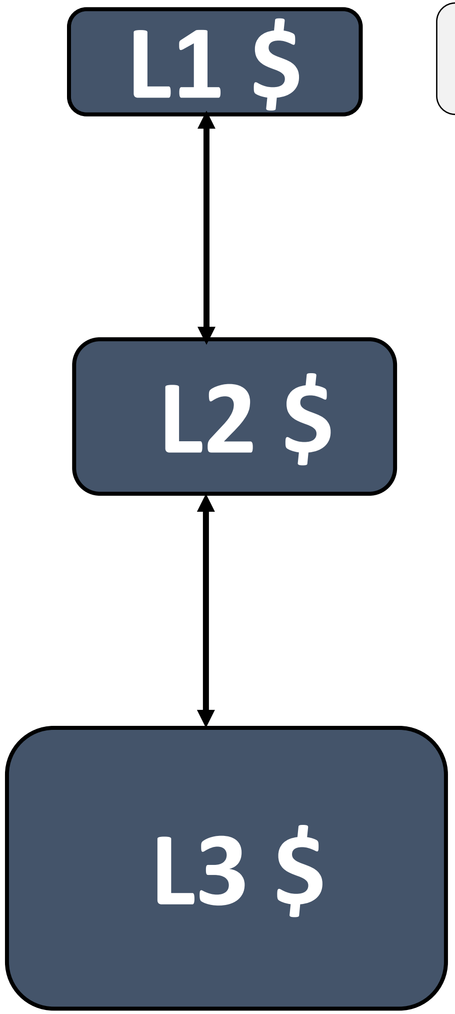
South pole ☺

Takeaway



Latency and bandwidth (multiple ports)

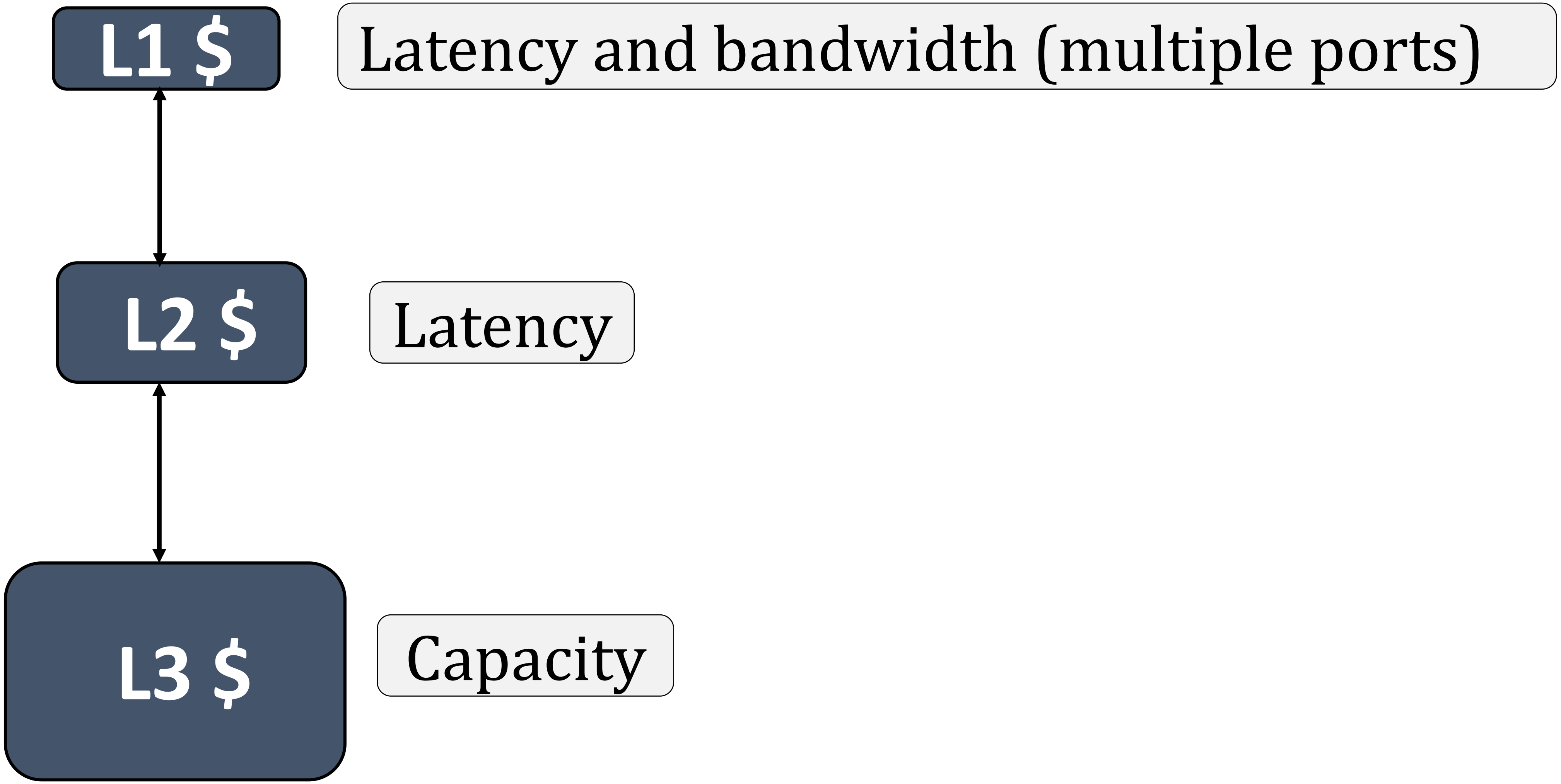
Takeaway



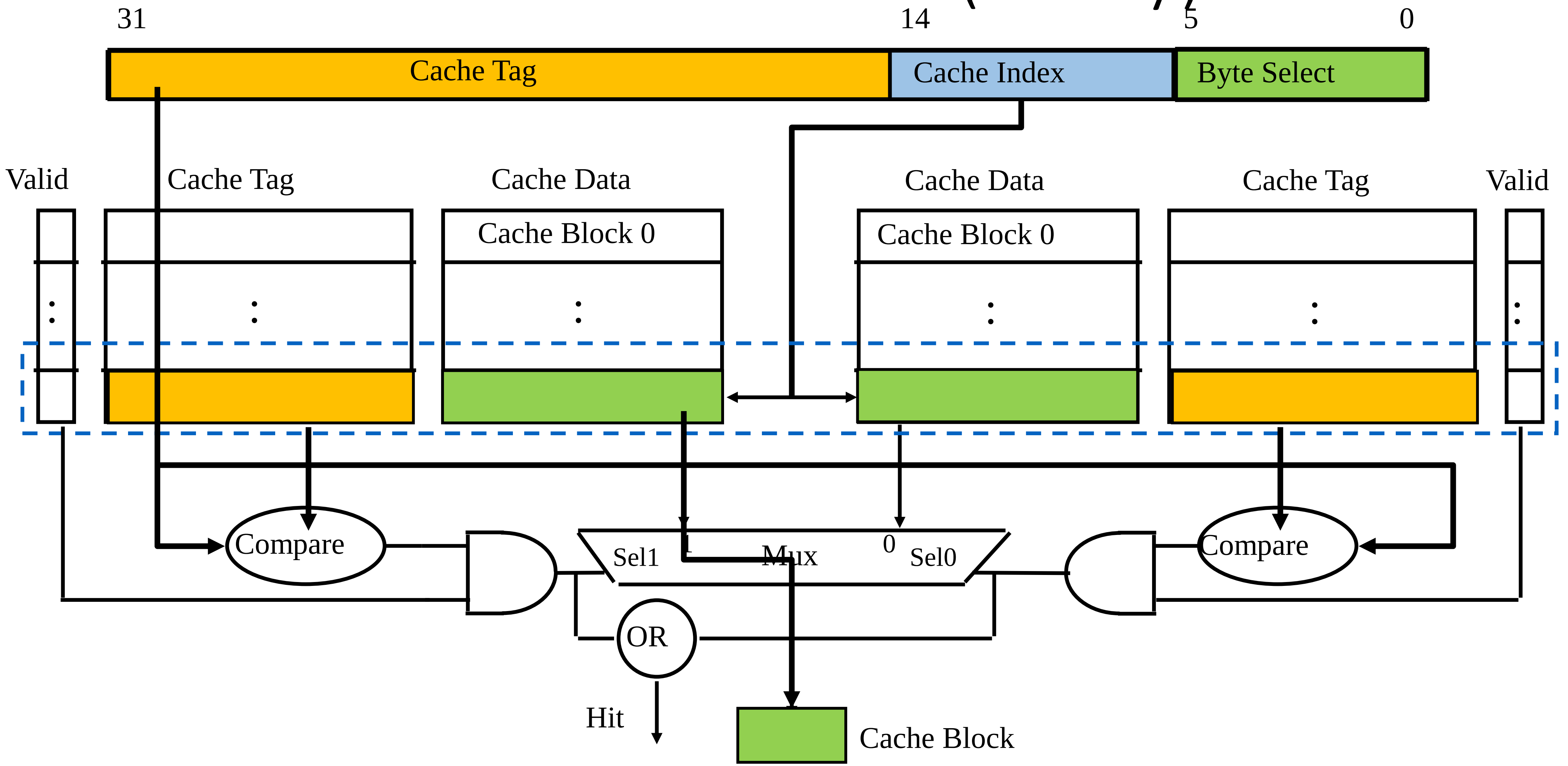
Latency and bandwidth (multiple ports)

Latency

Takeaway (Do not forget the word microarch.)



Associative Cache in action (2-way)



CASPER

Knobs of interest

Line size, associativity, cache size

Tradeoff: latency, complexity, energy/power

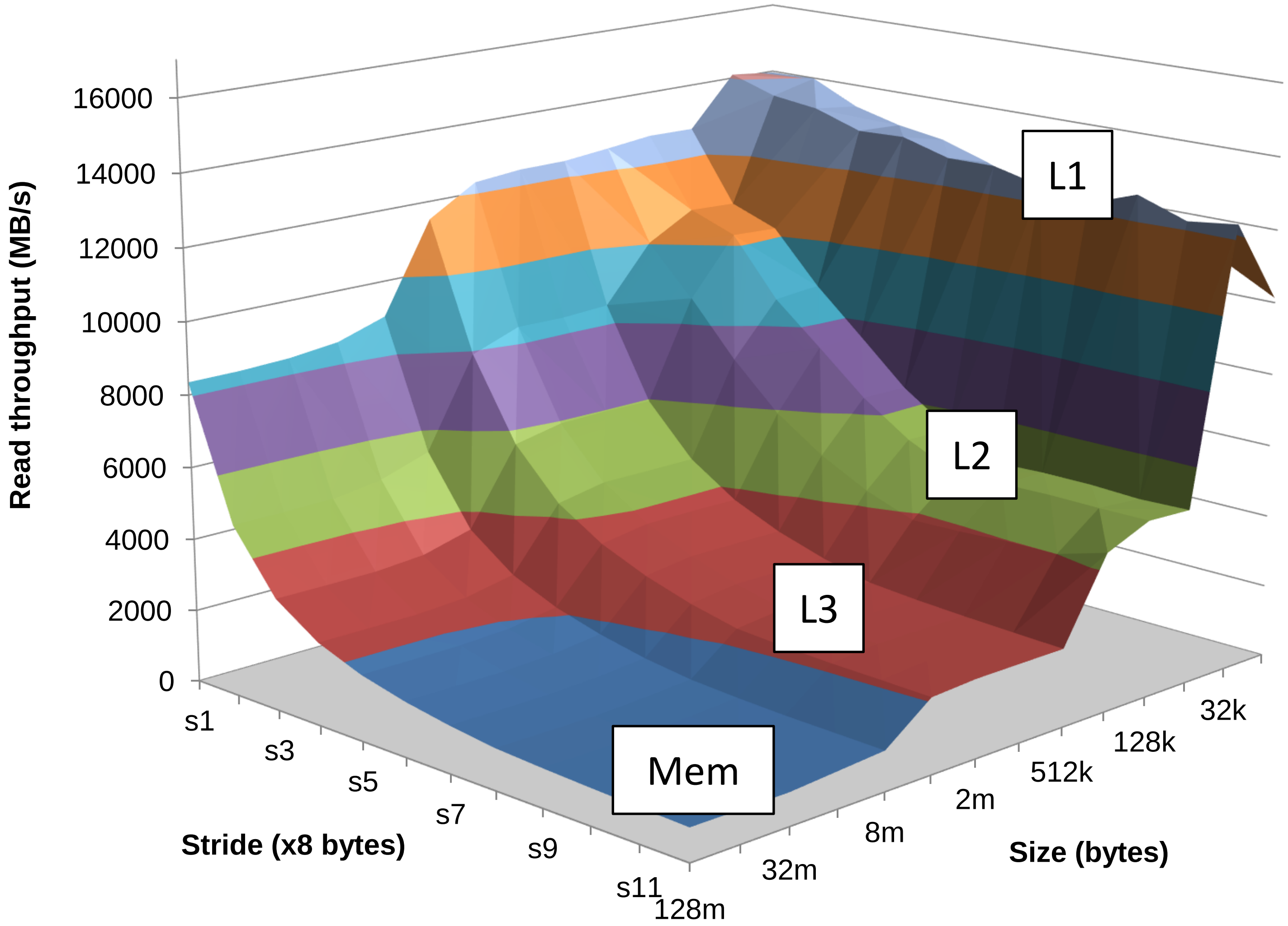
Tips: Think about the extremes:

Line size = one byte or cache size

Associativity = one or #lines

Cache size = Goal oriented: latency/bandwidth or capacity

Memory Mountain



Into the Real World

sudo dmidecode -t cache

cat /proc/cpuinfo

getconf -a | grep CACHE

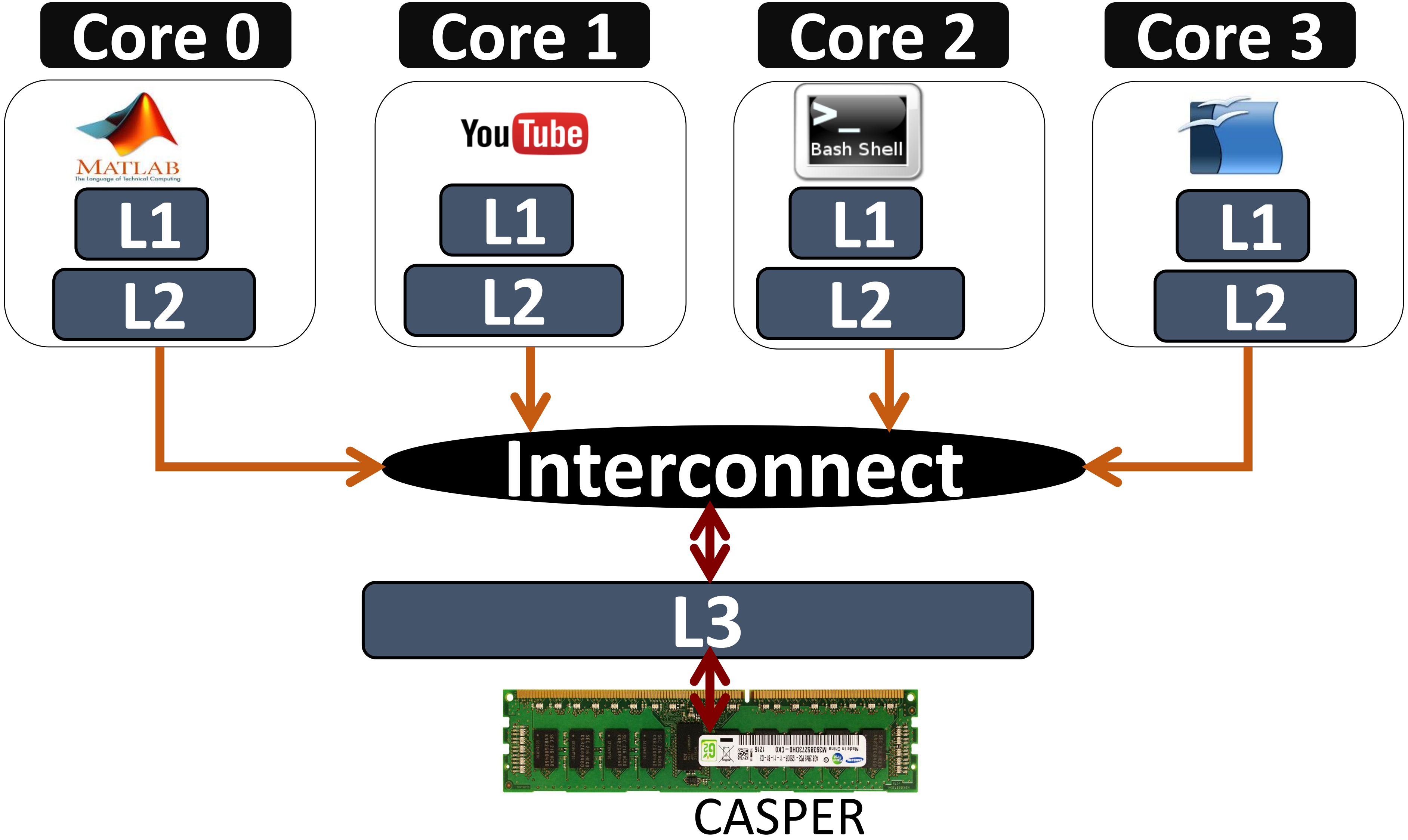
lscpu

Wiki chip: https://en.wikichip.org/wiki/WikiChip

Perf tool: https://perf.wiki.kernel.org/index.php/Main_Page

sudo perf stat -e cache-misses ….

Multicore


Core 0
Core 1
Core 2
Core 3
MATLAB
The Language of Technical Computing
YouTube
Bash Shell
L1
L2
L1
L2
L1
L2
L1
L2
Interconnect
L3
CASPER

PAUSE

Information leakage

CIA: Confidentiality, Integrity, Availability

Information leakage

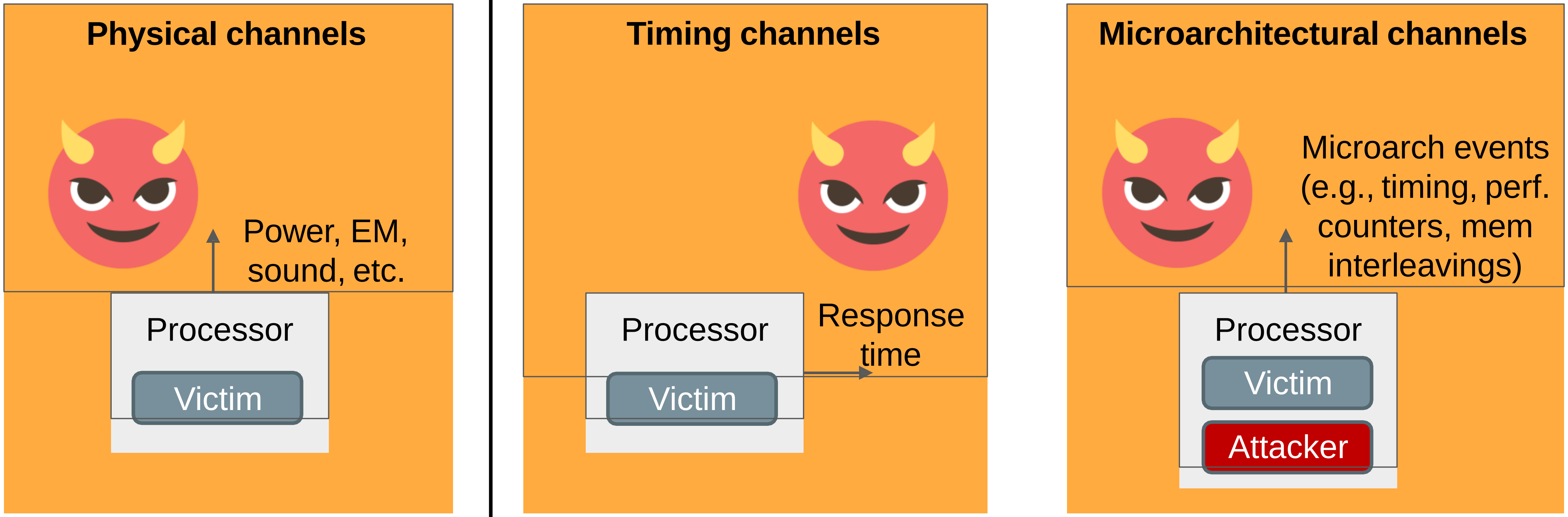
CIA: Confidentiality, Integrity, Availability

Confidentiality: was data being computed upon not revealed to an un-permitted party?

Integrity: was the computation performed correctly, returning the correct result?

Availability: did the computational resource carry out the task at all?

Channels of Interest



Attacker requires measurement equipment → physical access

Attacker may be remote (e.g., over an internet connection)

Attacker may be remote, or be co-located

Side/Covert Channel

- A **side channel** is an *unintended communication* between two or more parties
- A **covert channel** is an *intended communication* between two or more parties

(you upload a video on YouTube to communicate some information to your friends, if Gmail, whatsapp, call is not allowed)

In both cases:

- Communication should not be possible, following computing systems semantics
- The physical channel used for the communication can be the same

Side channels → unintended → need de-noising
Covert channels can show “best case” leakage

Scope of these channels

- Inter-process(application) communication that can violate privilege boundaries
- Infer information from application's data-dependent HW resource usage

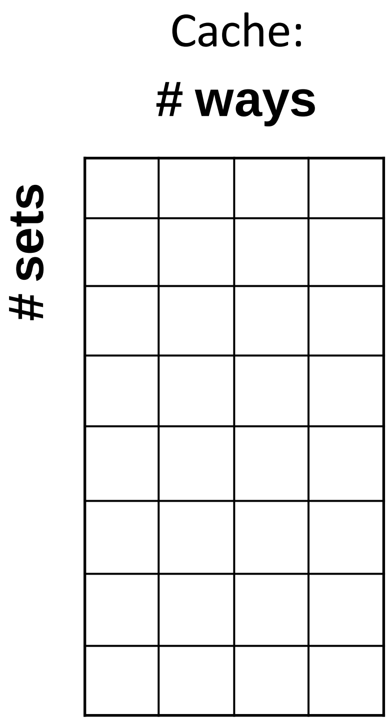
Side/covert channels not in any interface specification (e.g. ISA).

Therefore stealthy

- Sophisticated mechanisms needed to detect channel
- No permanent indication one has been exploited

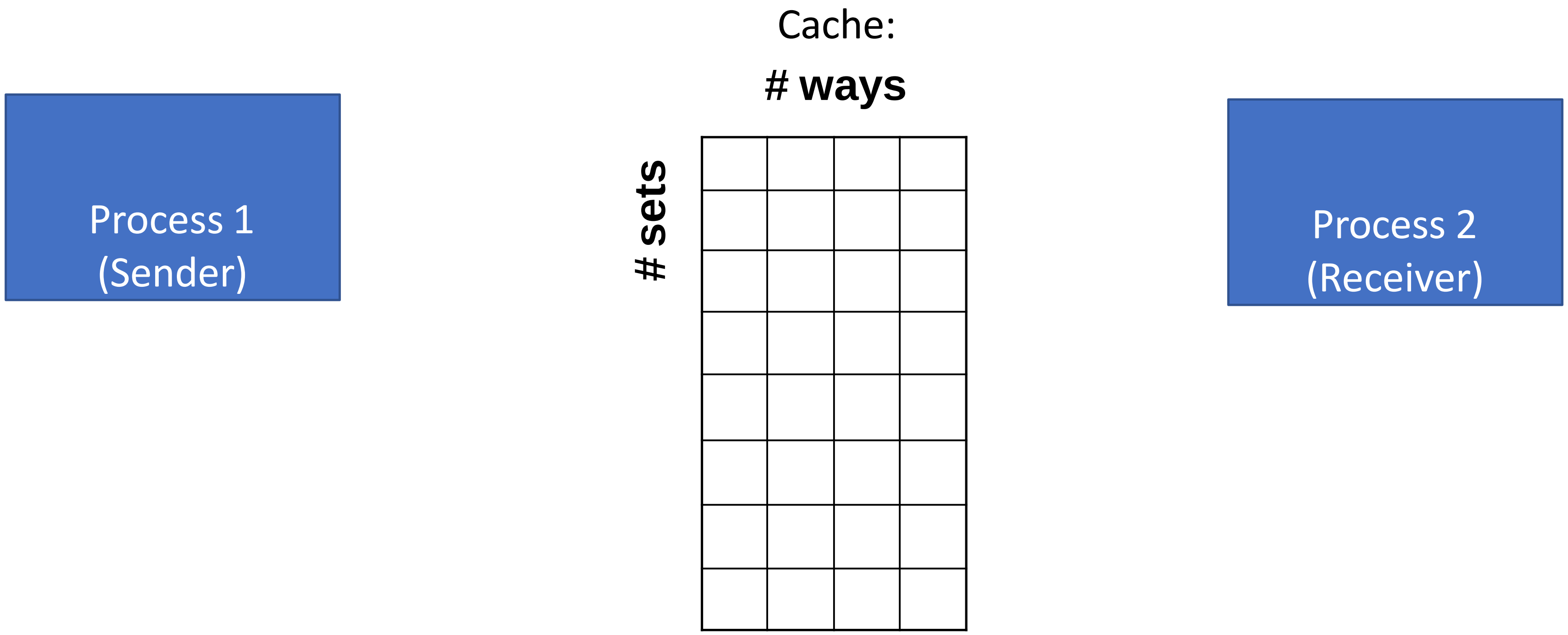
Let’s try to send a bit

Two processes can agree on “dead drops”



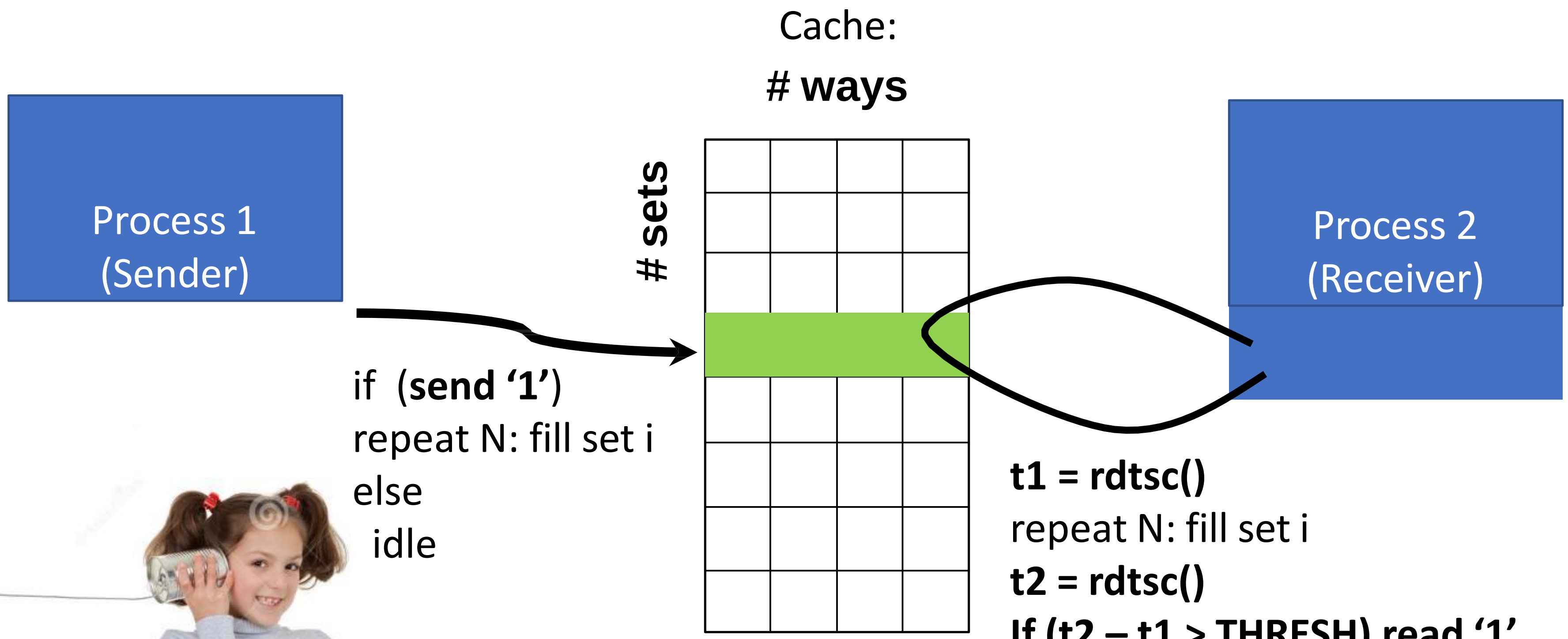
Let’s try to send a bit

Two processes can agree on “dead drops”



Let’s try to send a bit

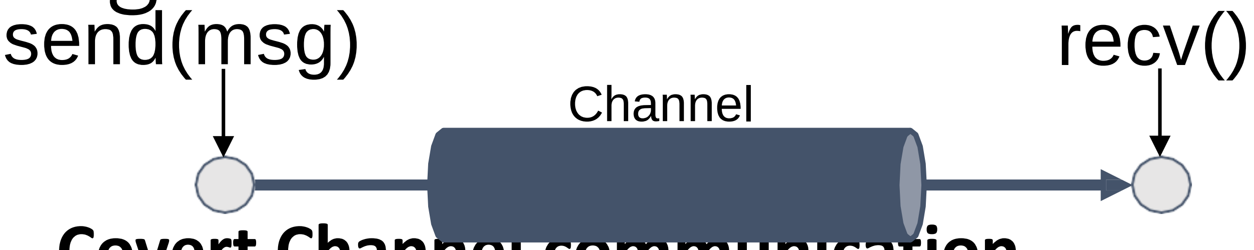
Two processes can agree on “dead drops”



if (**send ‘1’**)
repeat N: fill set i
else
idle

t1 = rdtsc()
repeat N: fill set i
t2 = rdtsc()
If (t2 – t1 > THRESH) read ‘1’
Else read ‘0’

How is it different from legitimate communication



Normal communication

```
include <socket.h>


void send(bit msg) {
  socket.send(msg);
}


bit recv() {
  return socket.recv(msg);
}
```

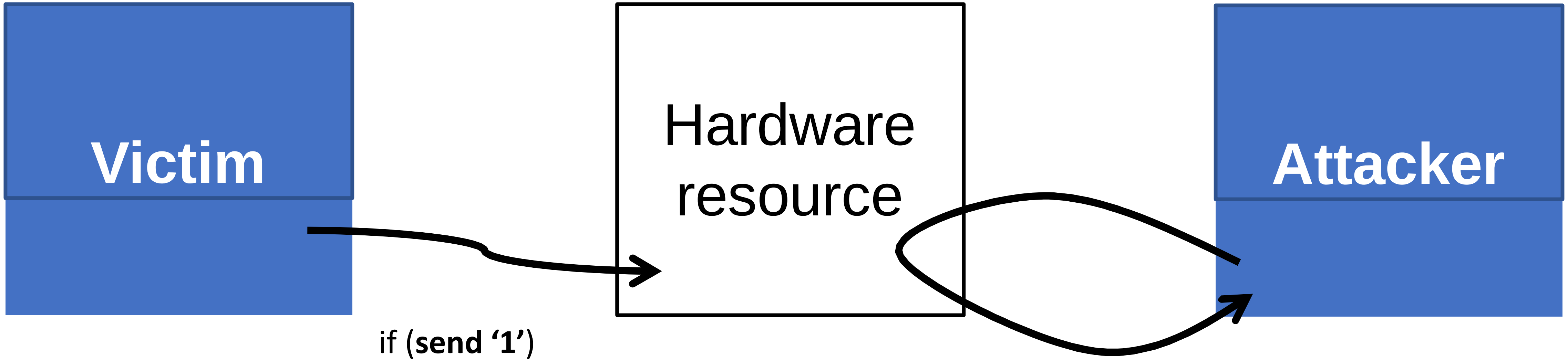
Covert Channel communication

```

void send(bit msg) {
  // pressure on cache
}


bit recv() {
  st = time();
  // pressure on cache
  return time() - st > THRESH;
}
```

From Covert to Side Channel



Covert channel:

if (**send ‘1’**)
Use resource
else
idle

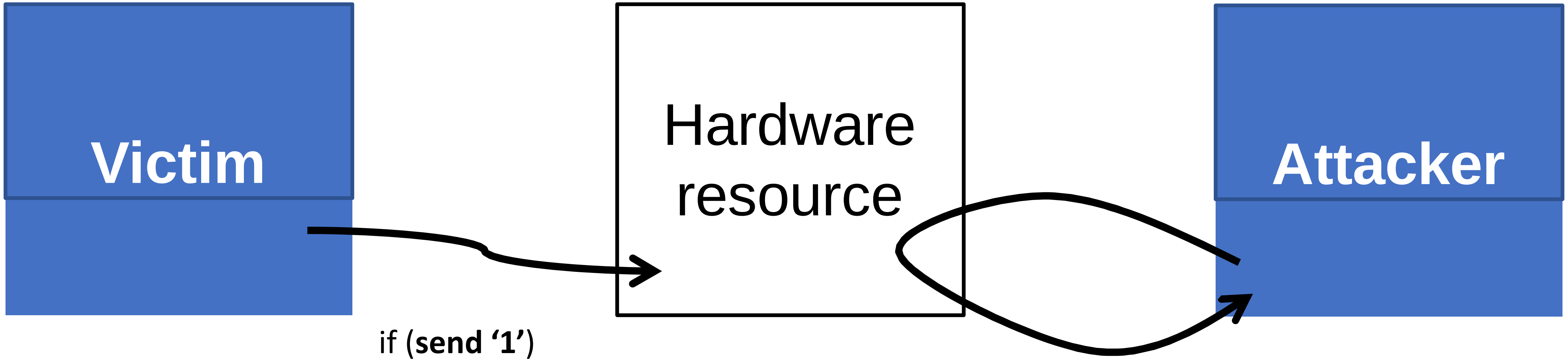
Side channel:

if (**secret**)
Use resource
else
idle

t1 = rdtsc()
Use resource
t2 = rdtsc()

if (t2 – t1 > THRESH) read ‘1’
else read ‘0’

From Covert to Side Channel



Covert channel:

if (**send ‘1’**)
Use resource
else
idle

Side channel:

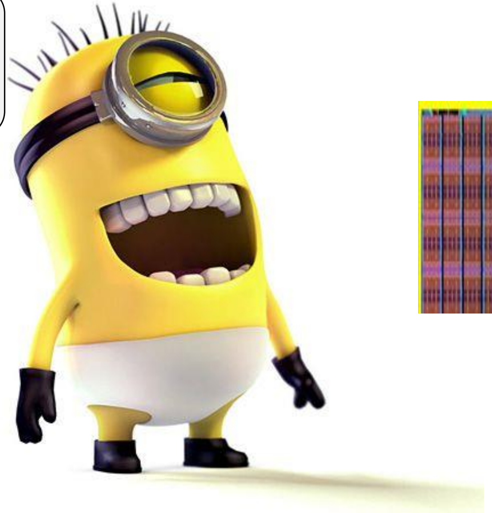
if (**secret**)
Use resource
else
idle

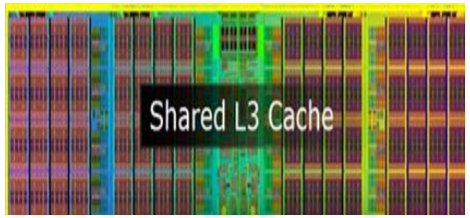
t1 = rdtsc()
Use resource
t2 = rdtsc()

if (t2 – t1 > THRESH) read ‘1’
else read ‘0’

Side/Covert Channel: Summary

Spy

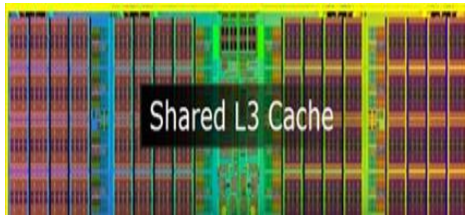


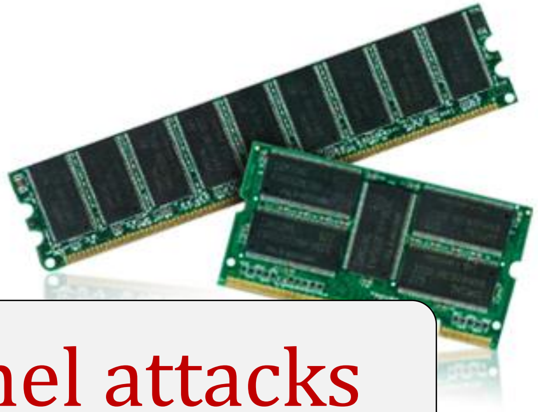


Victim

Side-channel attacks

Let's play





Oh Yes!!

Covert-channel attacks

Information leakage in the real world

$x \leftarrow 1$

for $i \leftarrow |e|$-1 **downto** 0 do

$\quad \boxed{x \leftarrow x^2 \bmod n}$

if $(e_i = 1)$ **then**

$\quad\quad \boxed{x = xb \bmod n}$

endif

done

return x

square

reduce

multiply

Modular exponentiation, b^e mod n

Exponent e is used for decryption

Attacker tries to get the e

Information leakage

$x \leftarrow 1$

for $i \leftarrow |e|$-1 **downto** 0 **do**

$x \leftarrow x^2 \bmod n$

if $(e_i = 1)$ **then**

$x = xb \bmod n$

endif

done

return x

square

reduce

multiply

Modular exponentiation, $b^e \bmod n$

Exponent e is used for decryption

e_i = 0, Square Reduce (SR)
e_i = 1, SRMR

Attacker tries to get the e

Summary

- Latency
- Bandwidth
- Side/Covert Channels
- Bandwidth, Accuracy, Agility
- Attack detector
- More details next lecture