Seguridad Informatica - Fall 2024

Module IV: Physical security

Lecture 2

Cache-based side channels

Marco Guarnieri

IMDEA Software Institute

Module IV

- Lecture 1 — Introduction
- Lecture 2 — Cache-based side channel attacks
- Lecture 3 — Speculative execution attacks
- Lecture 4 — Non-interference
- Lecture 5 — Automated detection of speculative leaks

Module IV

- Lecture 1 — Introduction
- Lecture 2 — Cache-based side channel attacks
- Lecture 3 — Speculative execution attacks
- Lecture 4 — Non-interference
- Lecture 5 — Automated detection of speculative leaks

Recommended readings

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES

Available at https://eprint.iacr.org/2005/271.pdf

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack

Available at https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-yarom.pdf

What is a side-channel attack?

What is a side-channel attack?

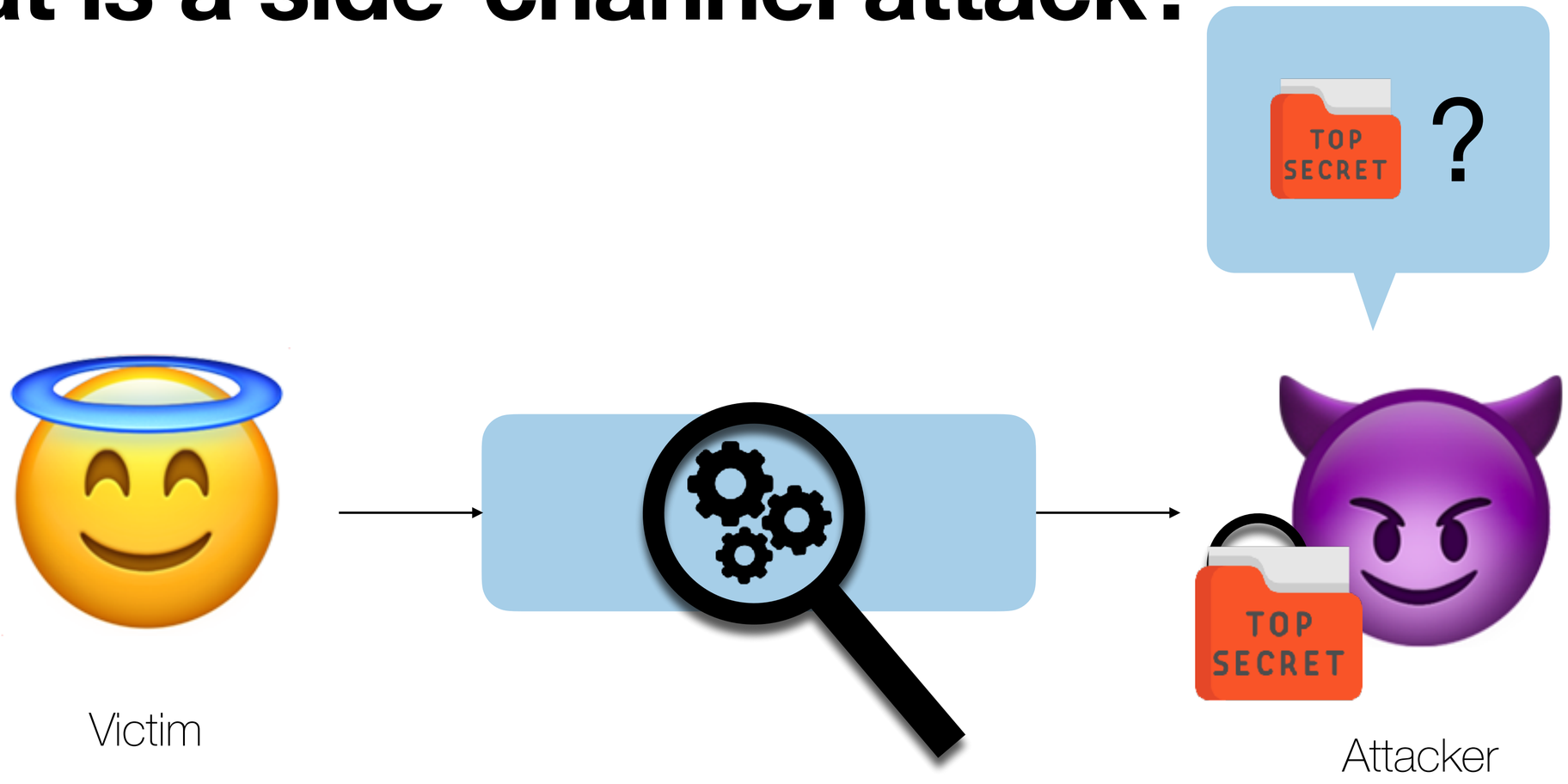
Victim

What is a side-channel attack?

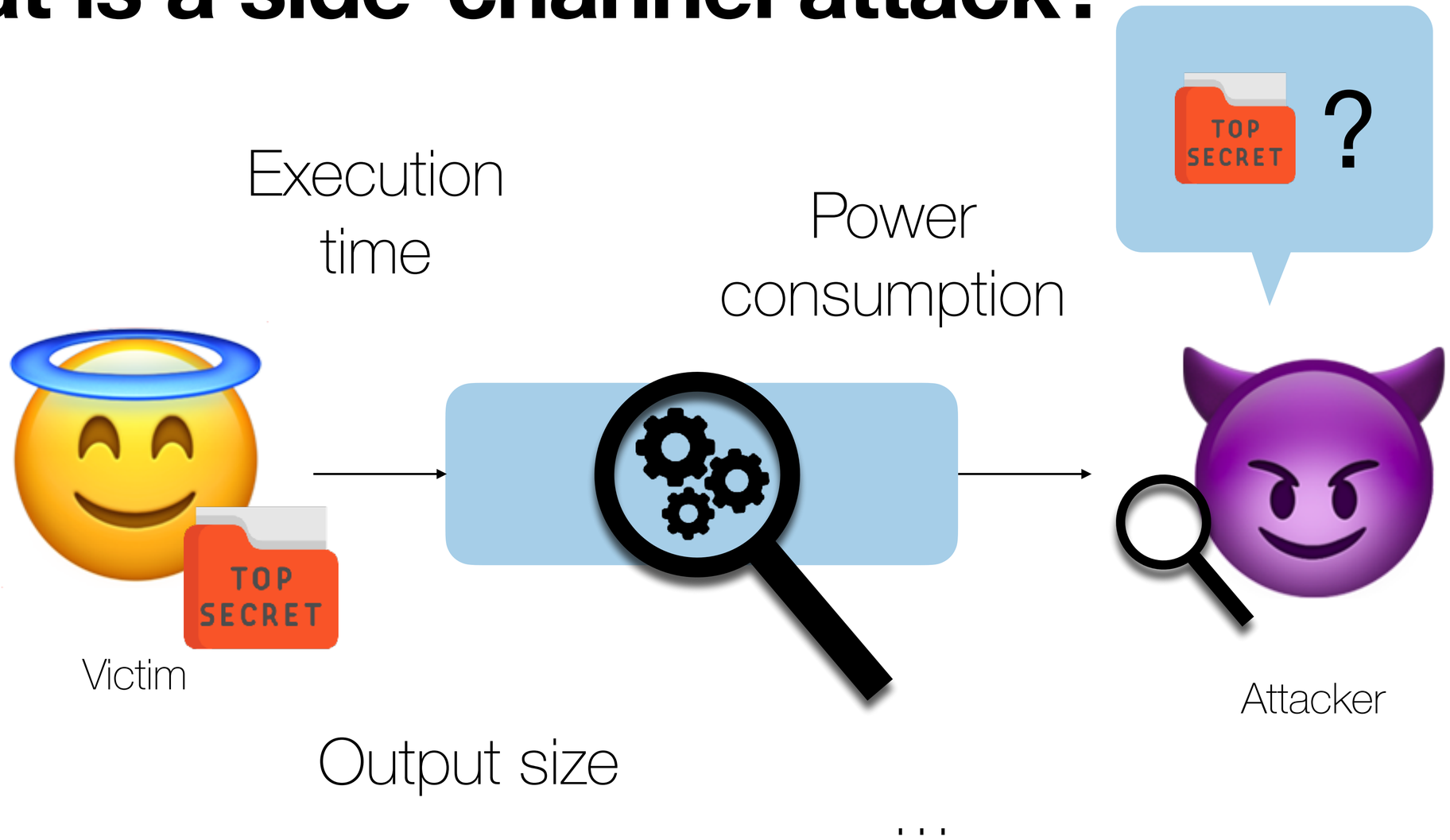
Victim

Attacker

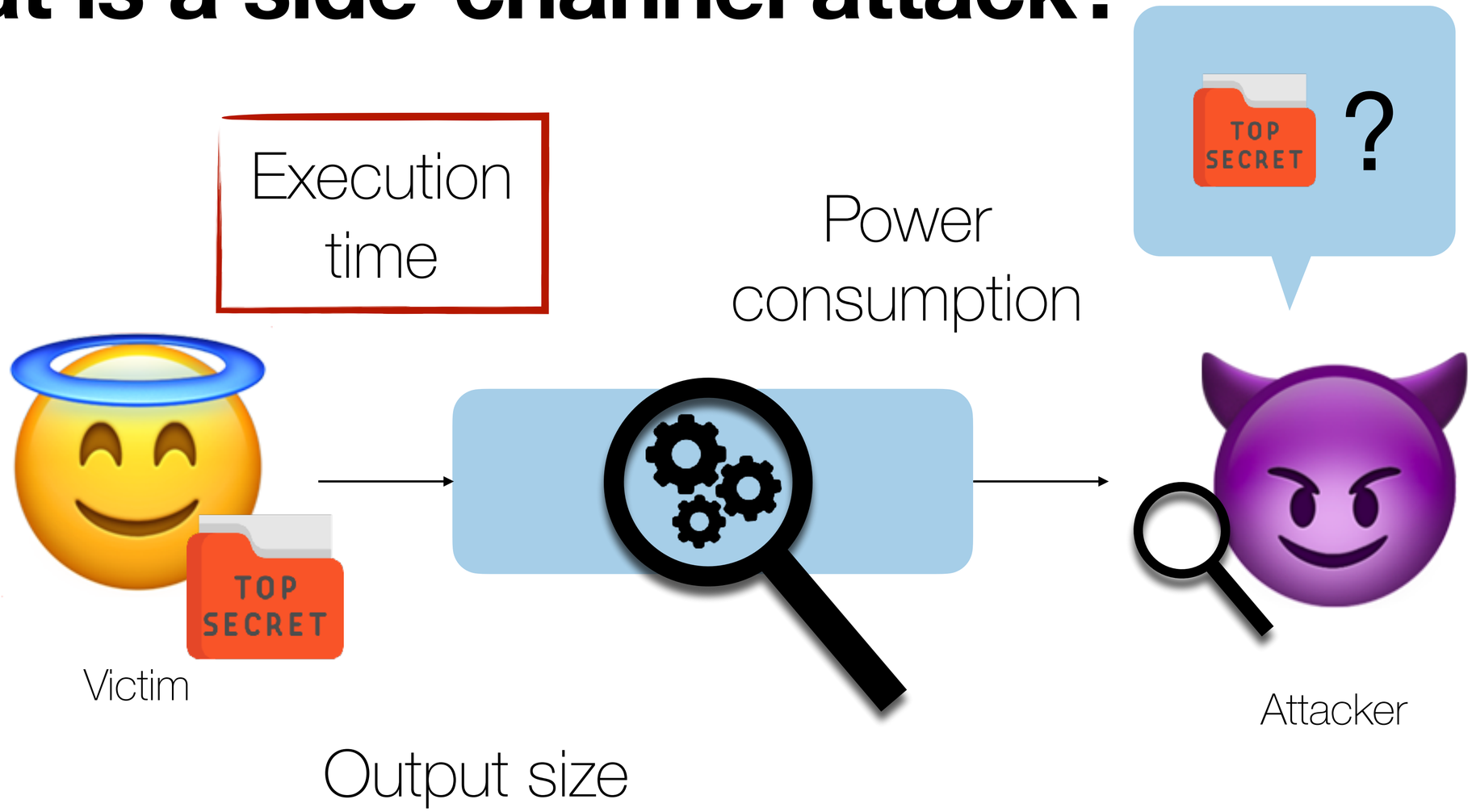
What is a side-channel attack?



What is a side-channel attack?

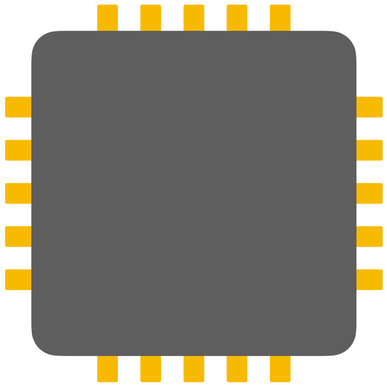


What is a side-channel attack?

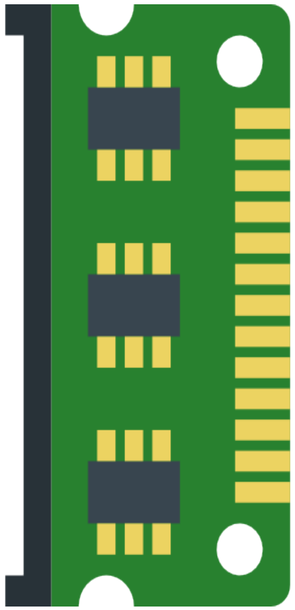


Processors and caches

Processors and memory



Processor

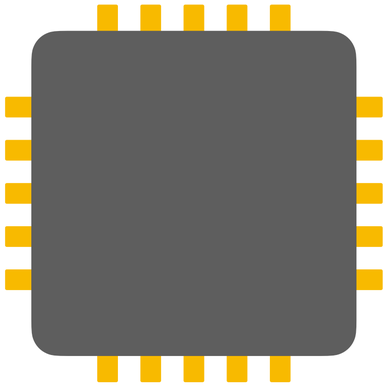


Main memory
(DRAM)

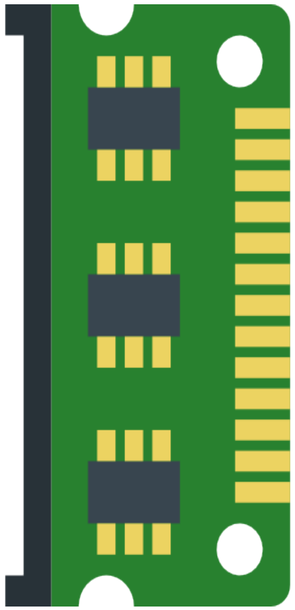
DRAM latency for Intel i7-4770 (Haswell)

Processors and memory

```
x = array[idx];
```



Processor



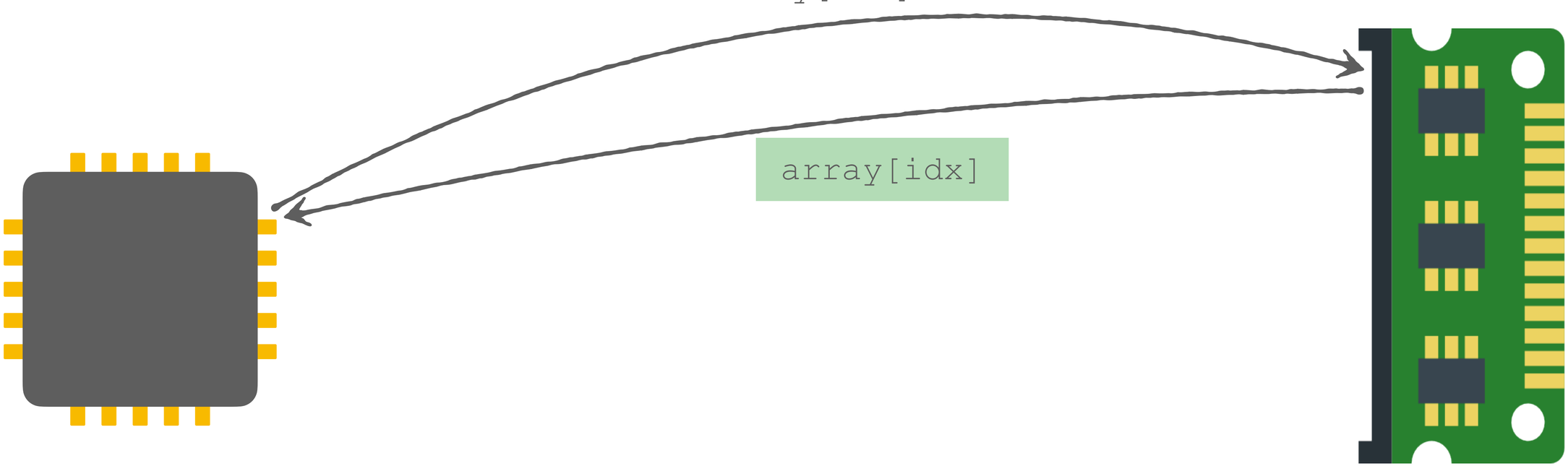
Main memory
(DRAM)

DRAM latency for Intel i7-4770 (Haswell)

Processors and memory

```
x = array[idx];
```

36 cycles + 57 ns / ~67 ns



DRAM latency for Intel i7-4770 (Haswell)

Processors and memory

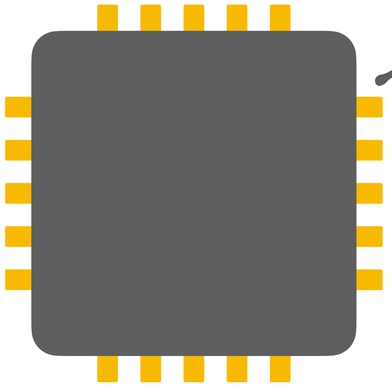
```
x = array[idx];

y = array[idx];
```

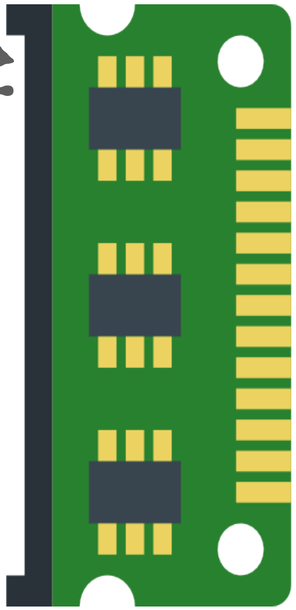
36 cycles + 57 ns / ~67 ns

array[idx]?

array[idx]



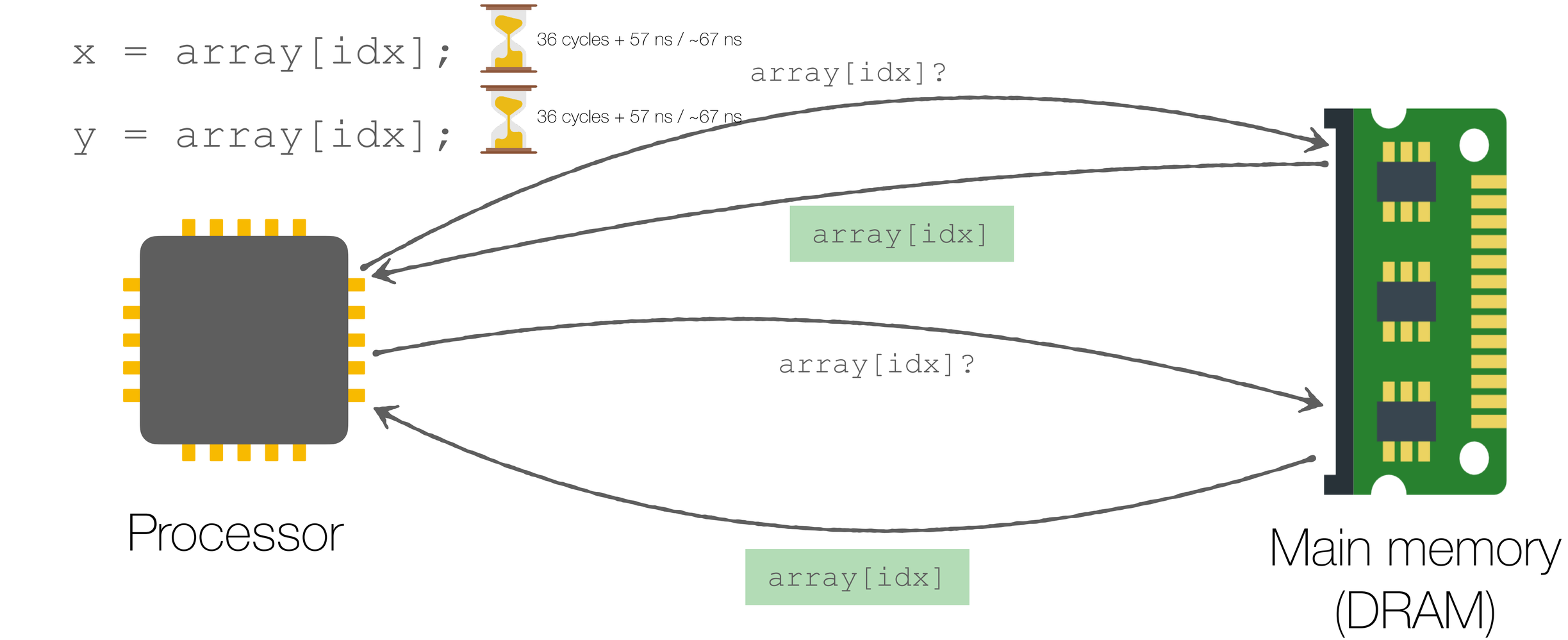
Processor



Main memory
(DRAM)

DRAM latency for Intel i7-4770 (Haswell)

Processors and memory

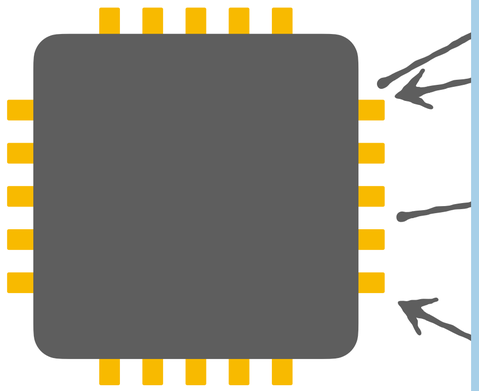


DRAM latency for Intel i7-4770 (Haswell)

Processors and memory

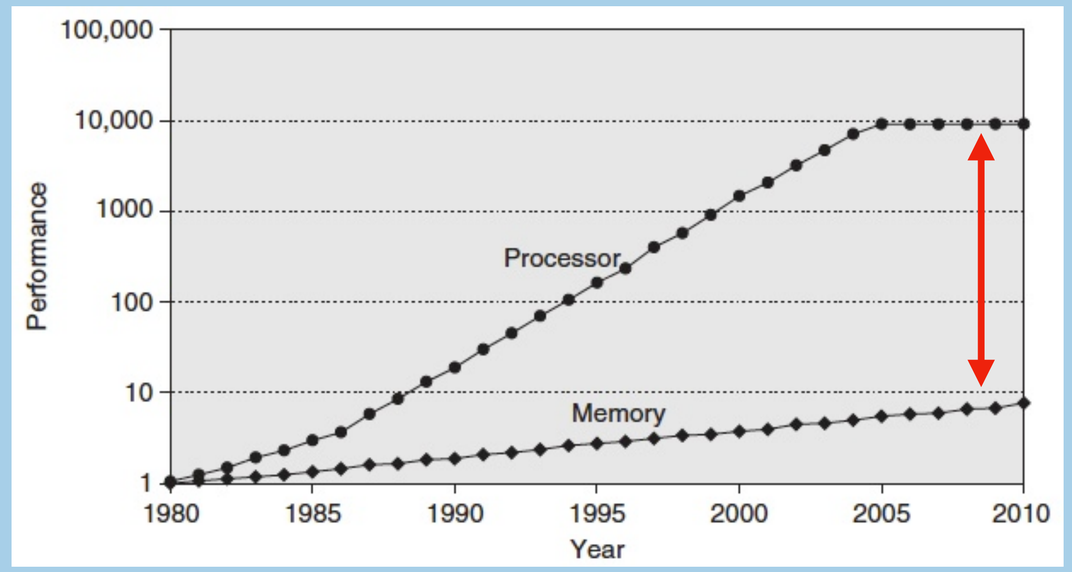
```
x = array[idx]

y = array[idx]
```



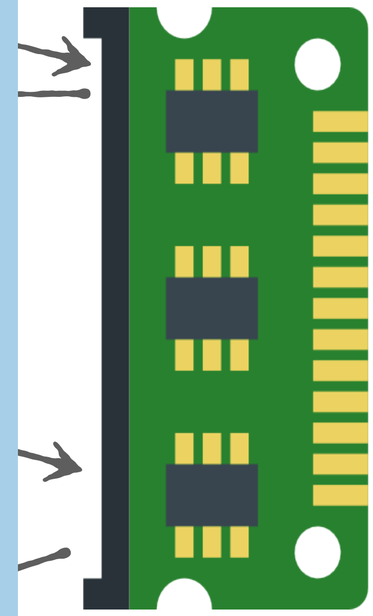
Processor

We have a problem…



Source: Hennesy, Patterson - Computer architectures: a quantitative approach

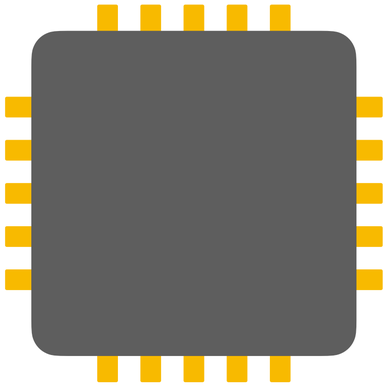
```
array[idx]
```



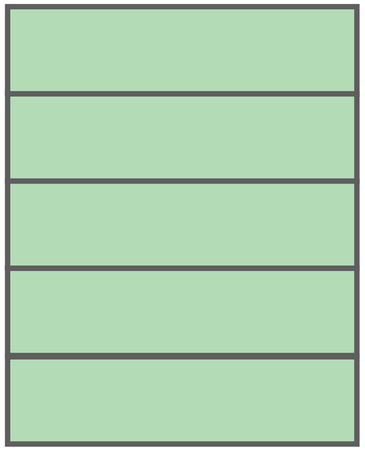
Main memory (DRAM)

DRAM latency for Intel i7-4770 (Haswell)

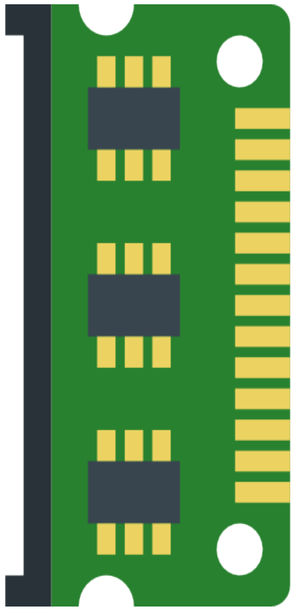
Caches to the rescue!



Processor



Cache

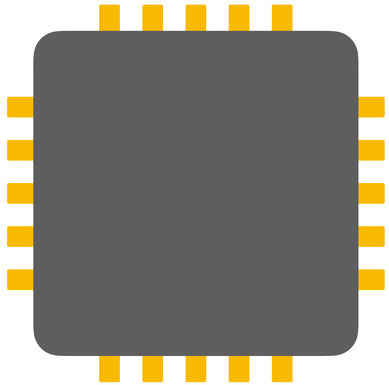


Main memory
(DRAM)

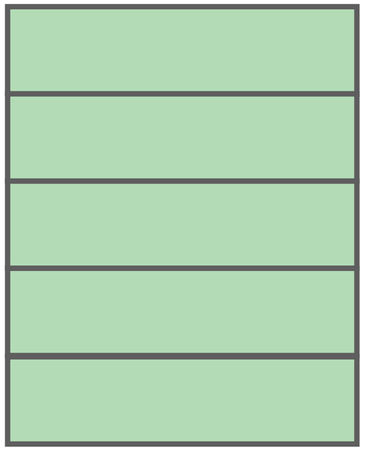
DRAM and L1 latency for Intel i7-4770 (Haswell)

Caches to the rescue!

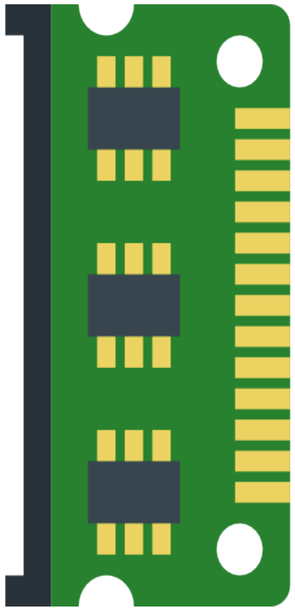
```
x = array[idx];
```



Processor



Cache



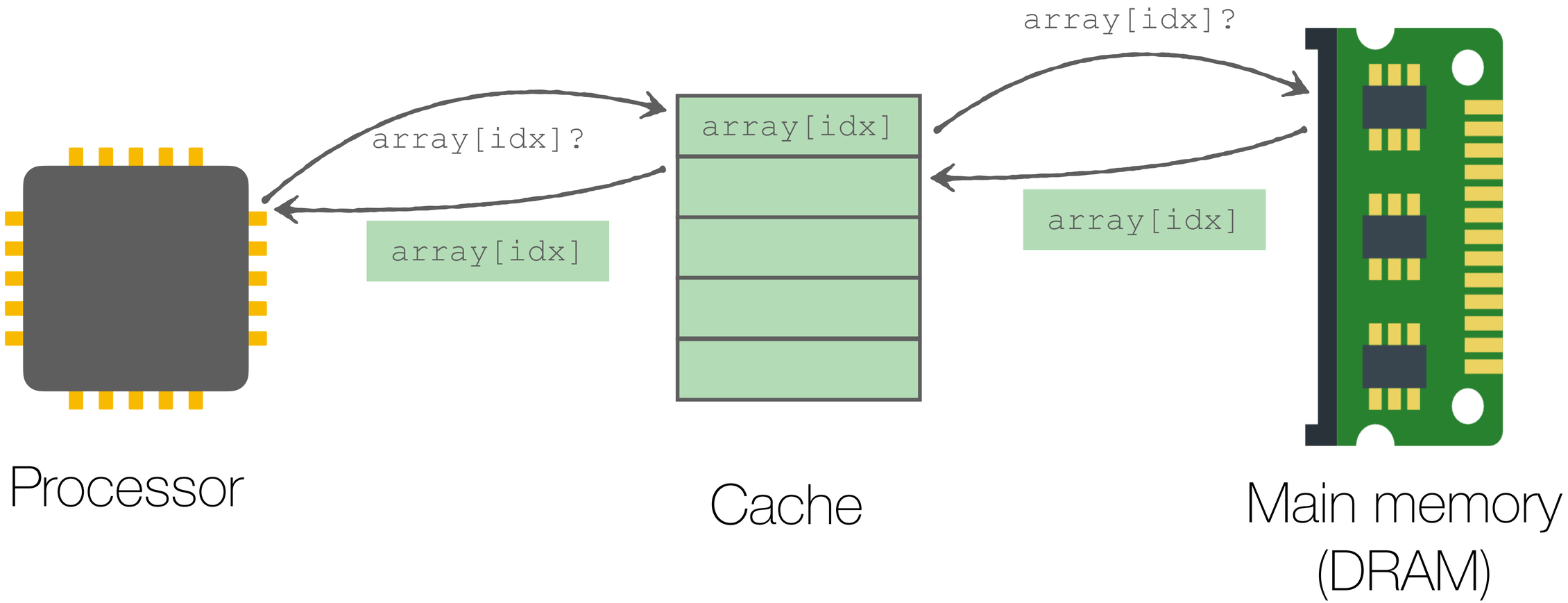
Main memory
(DRAM)

DRAM and L1 latency for Intel i7-4770 (Haswell)

Caches to the rescue!

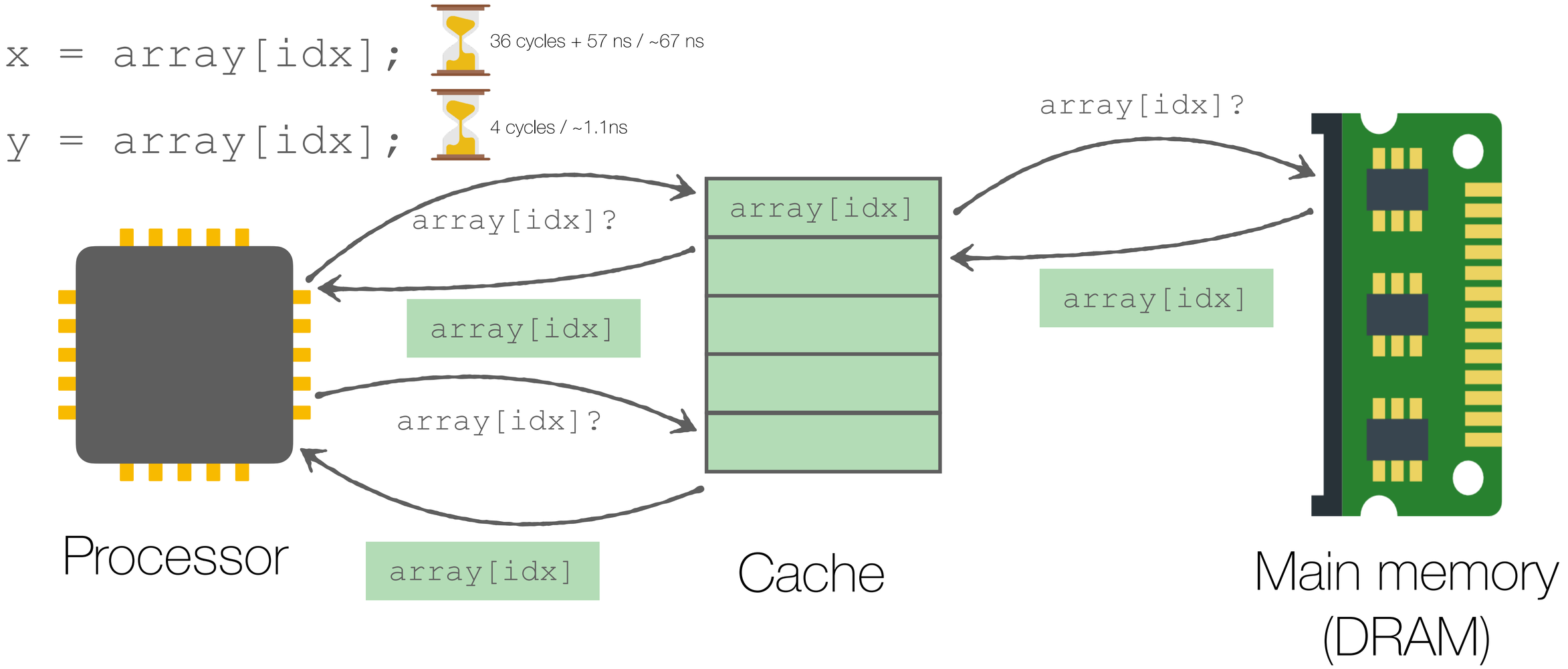
```
x = array[idx];
```

36 cycles + 57 ns / ~67 ns



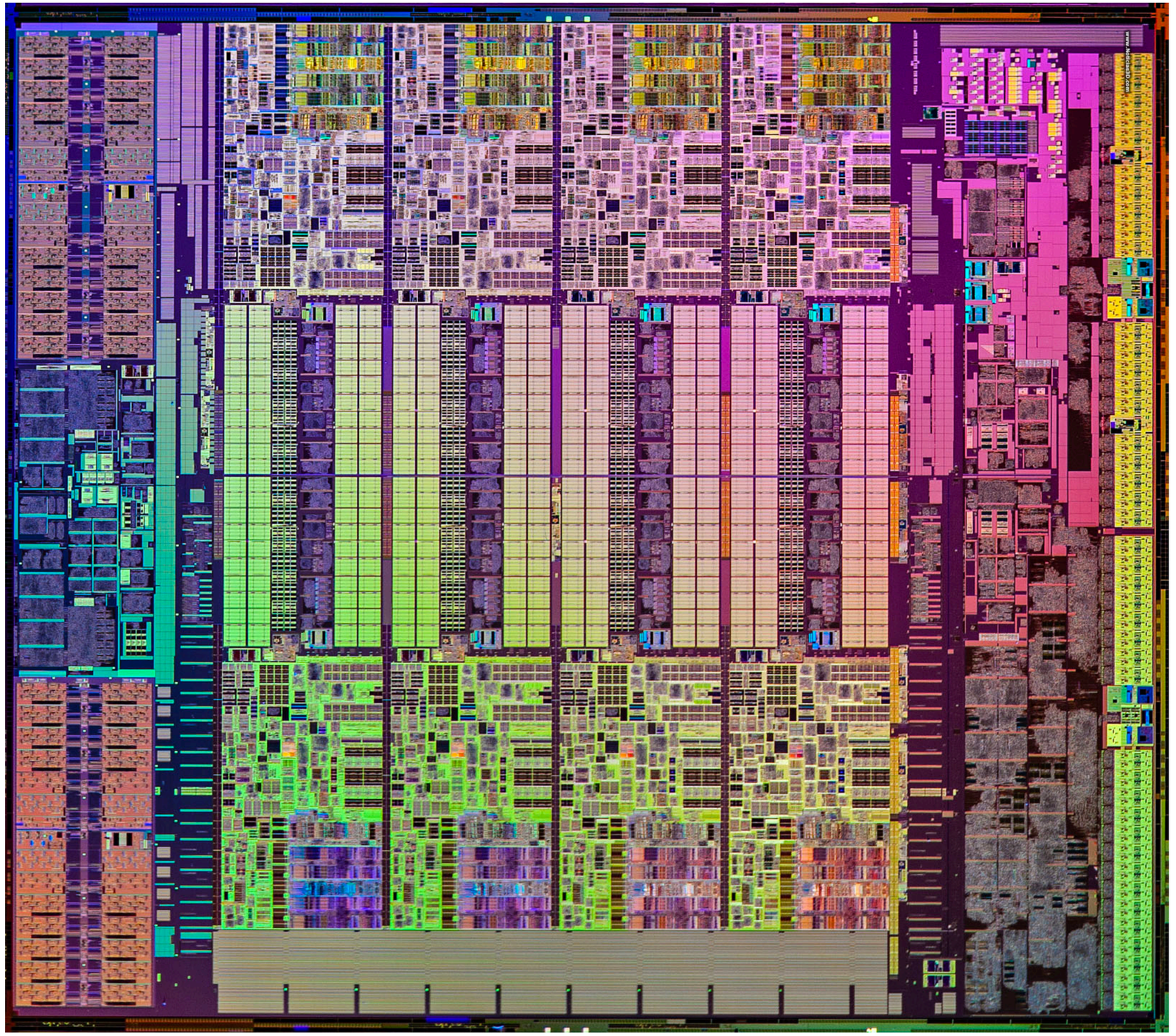
DRAM and L1 latency for Intel i7-4770 (Haswell)

Caches to the rescue!



DRAM and L1 latency for Intel i7-4770 (Haswell)

Processors 101

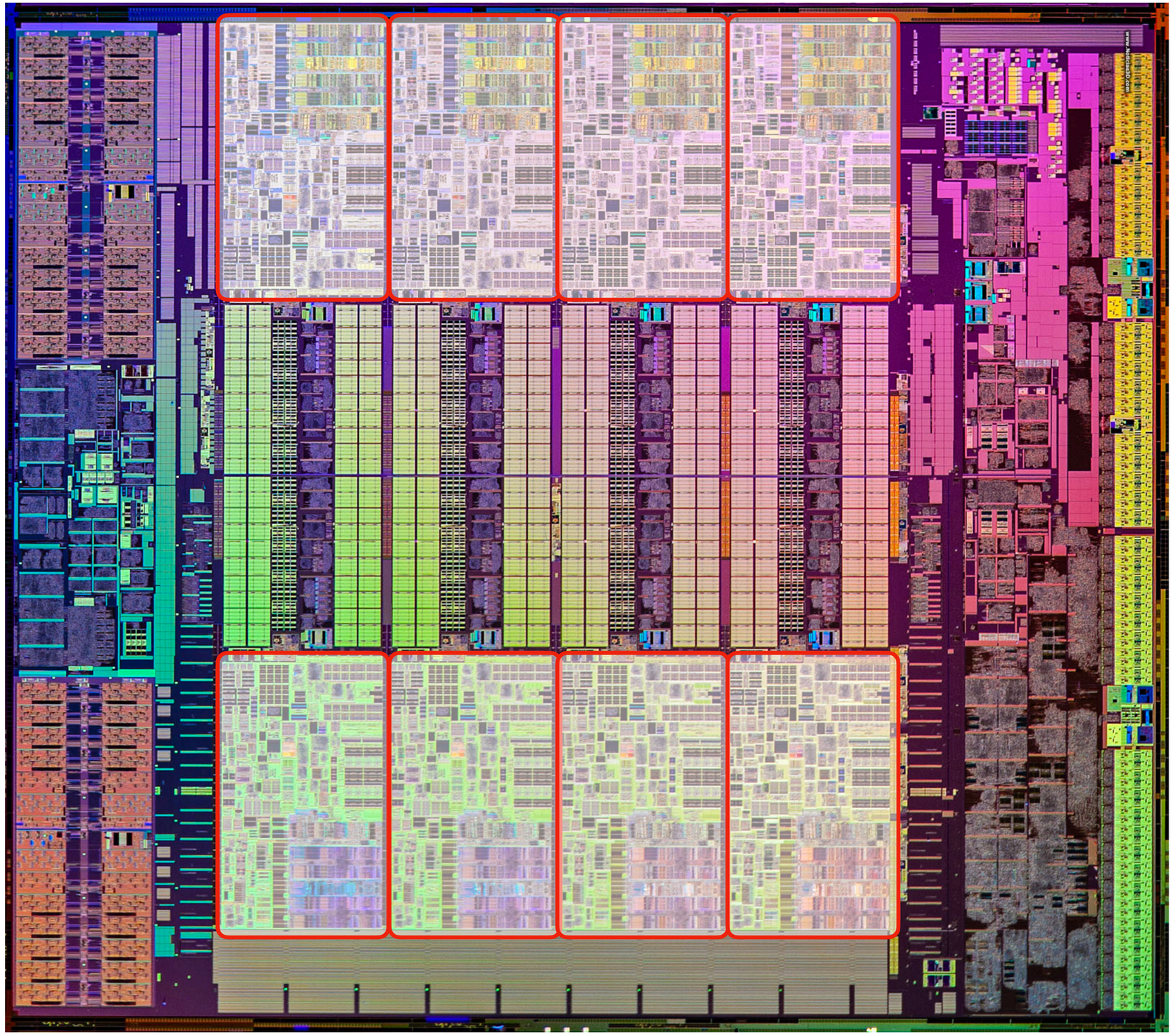


Picture of Intel “Haswell-E” Eight Core CPU

Processors 101

CORE

Execute instructions



Picture of Intel “Haswell-E” Eight Core CPU

Processors 101

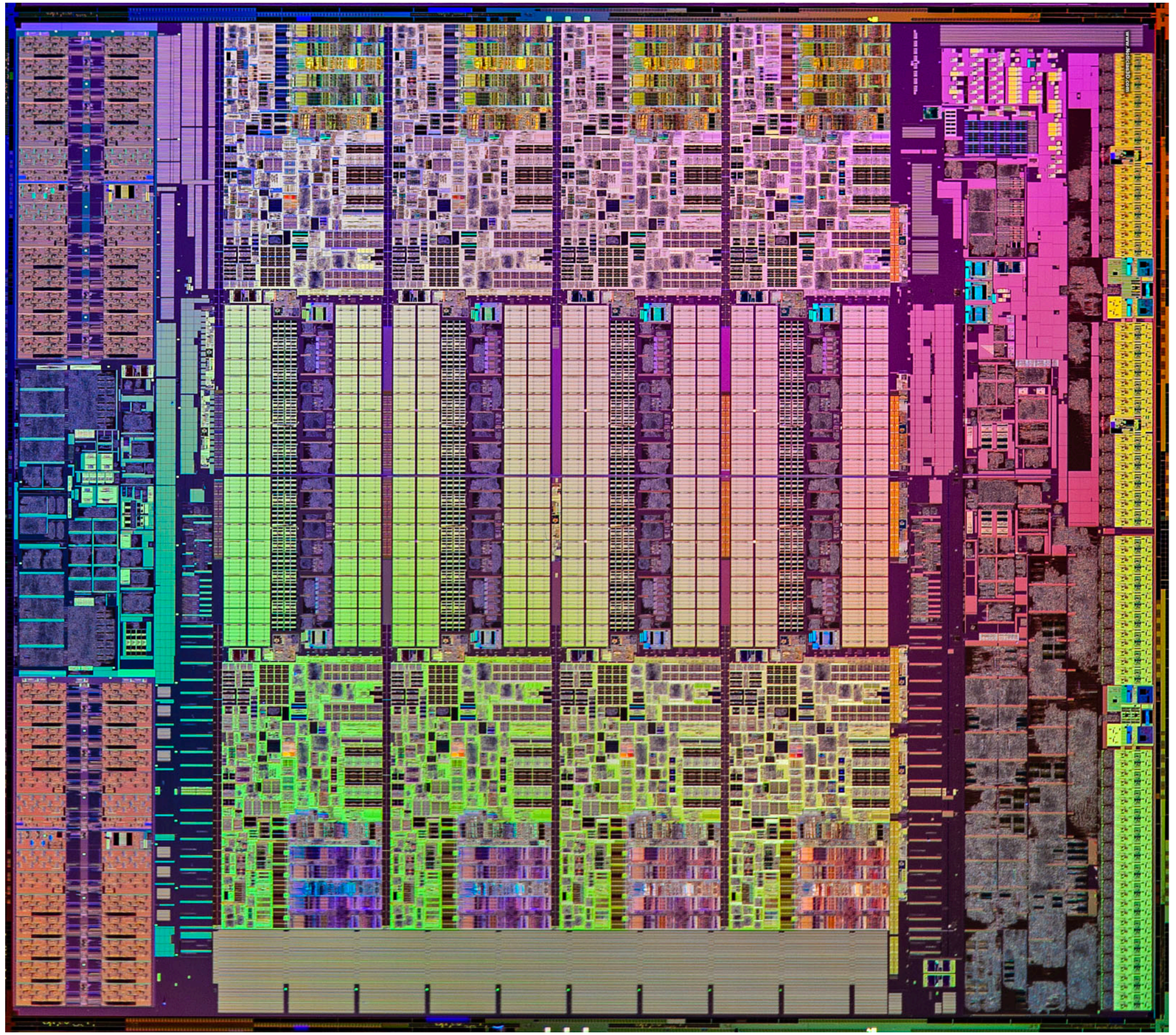
CORE

Execute instructions

CACHE

Memory storing recently accessed data

Speed up computation by reducing interaction with DRAM



Picture of Intel “Haswell-E” Eight Core CPU

Processors 101

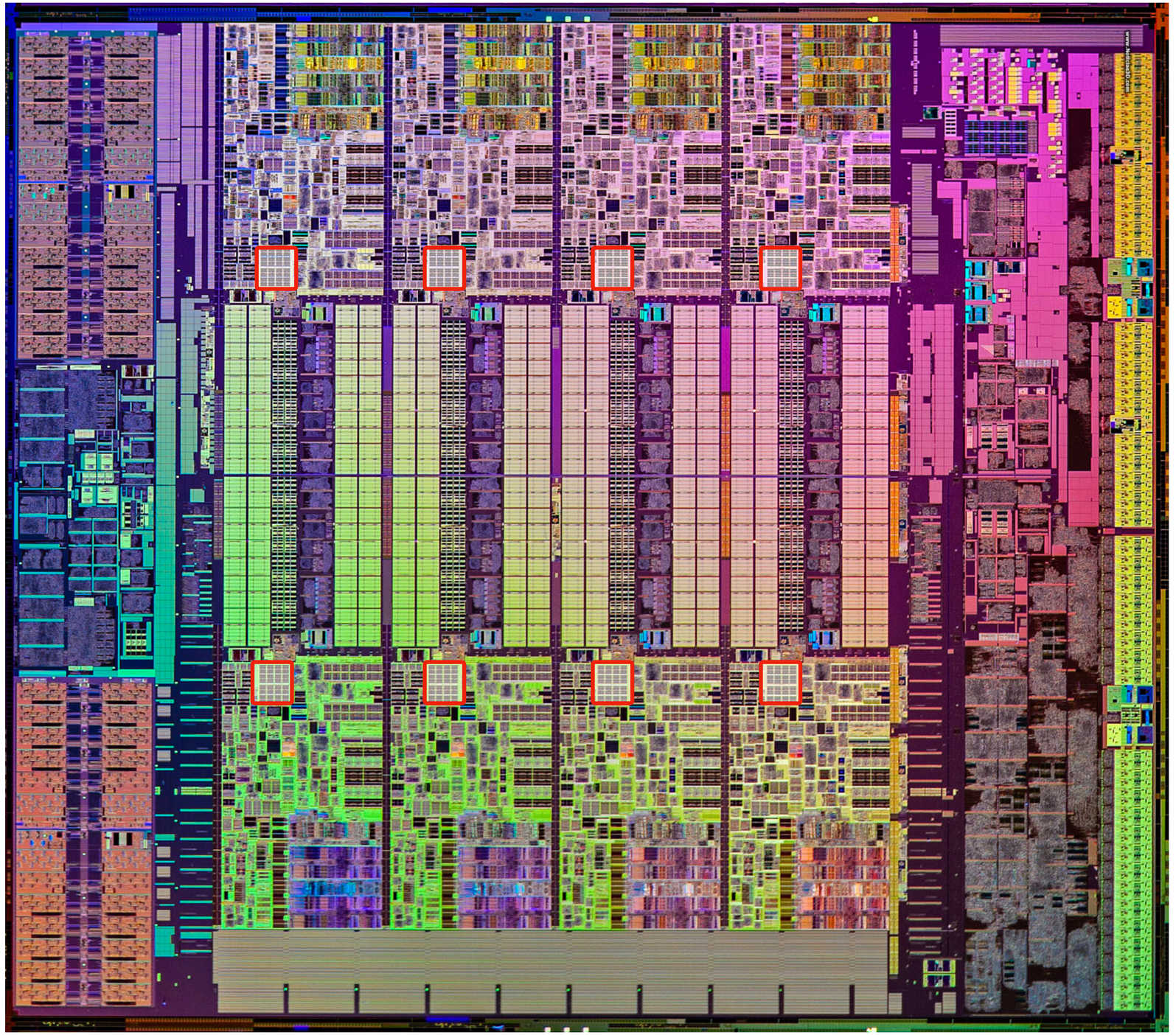
CORE

Execute instructions

CACHE

Memory storing recently accessed data

Speed up computation by reducing interaction with DRAM



Picture of Intel “Haswell-E” Eight Core CPU

Processors 101

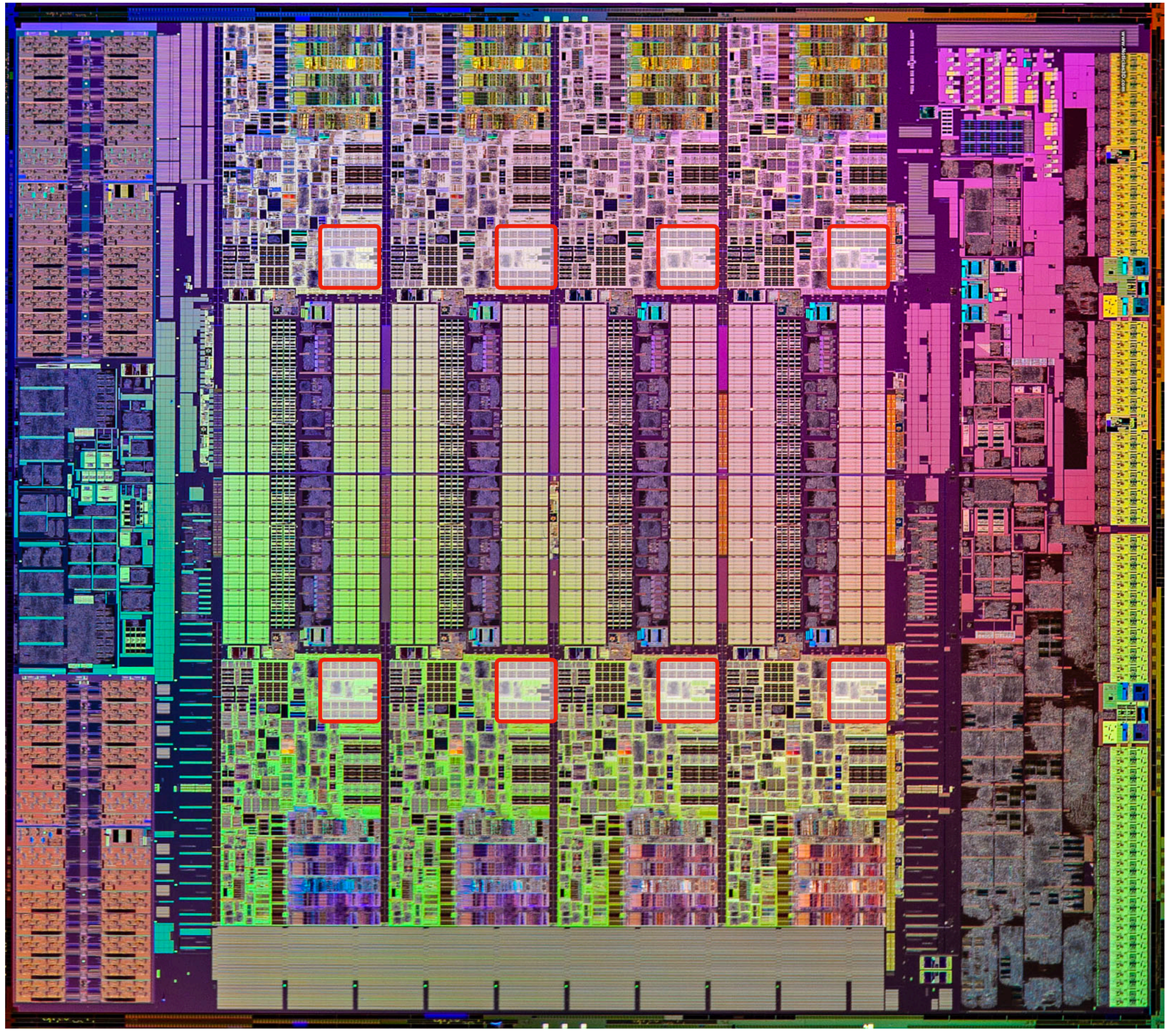
CORE

Execute instructions

CACHE

Memory storing recently accessed data

Speed up computation by reducing interaction with DRAM



Picture of Intel “Haswell-E” Eight Core CPU

Processors 101

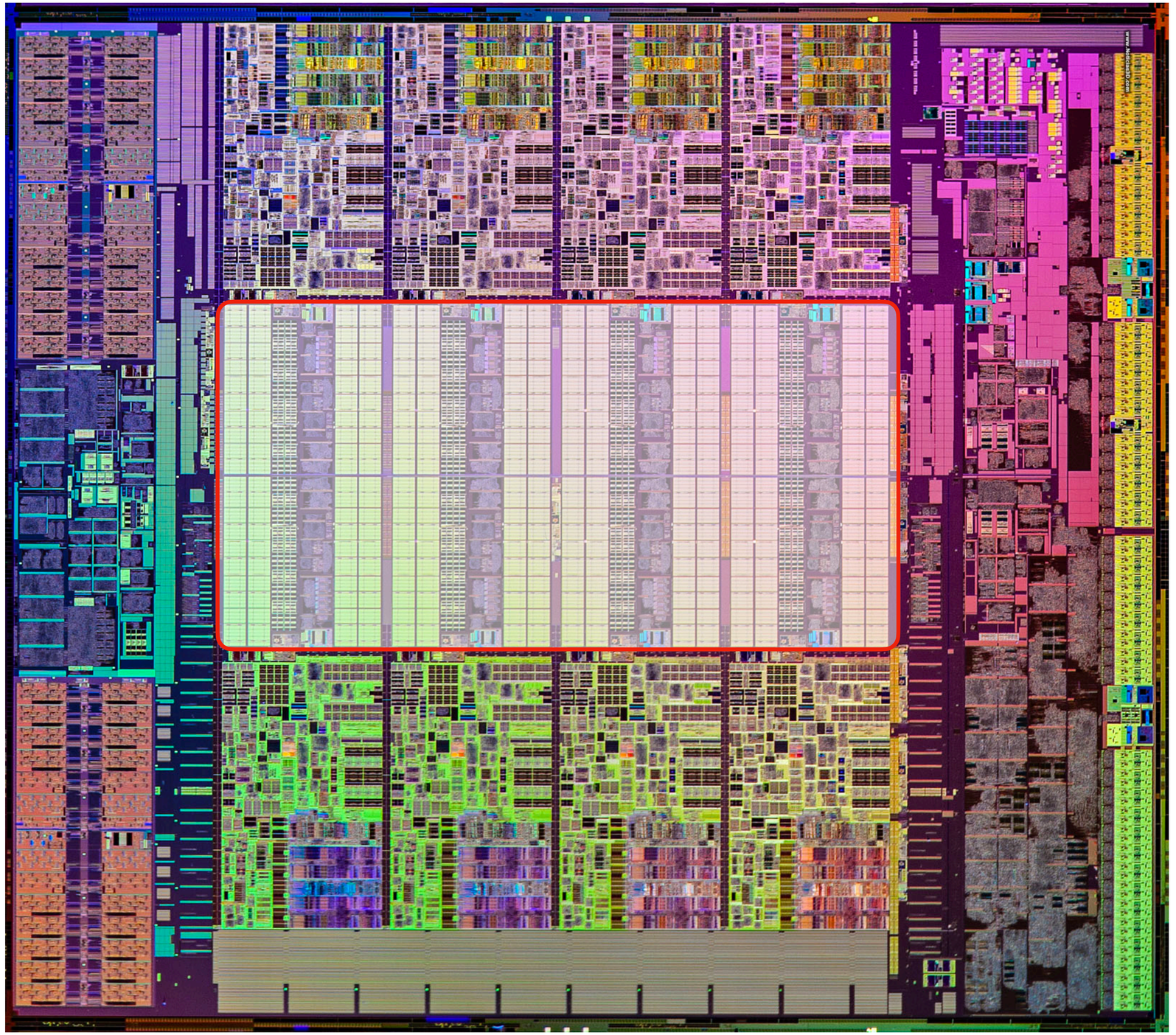
CORE

Execute instructions

CACHE

Memory storing recently accessed data

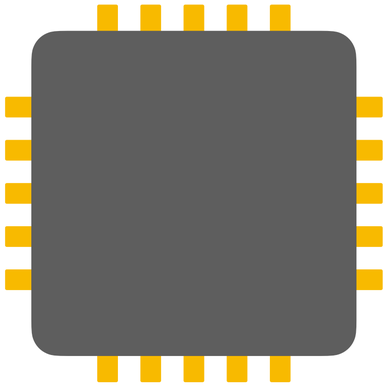
Speed up computation by reducing interaction with DRAM



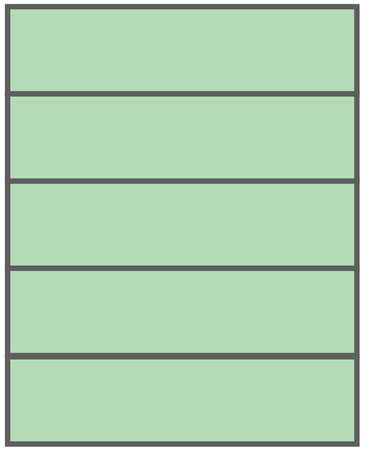
Picture of Intel “Haswell-E” Eight Core CPU

Caches 101

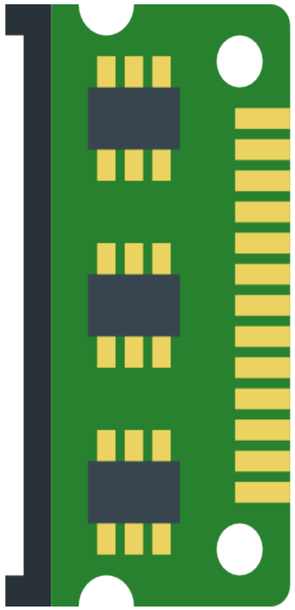
```
x = array[idx];
```



Processor



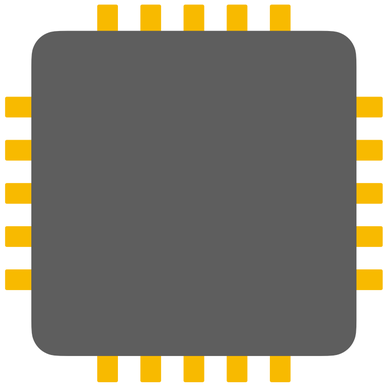
Cache



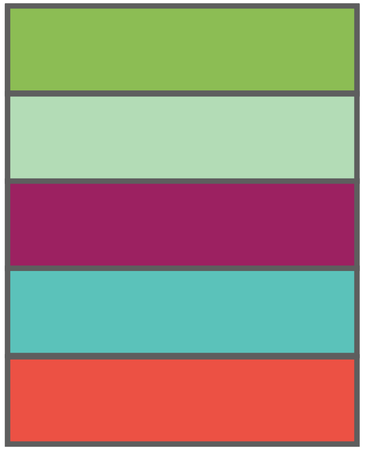
Main memory
(DRAM)

Caches 101

```
x = array[idx];
```

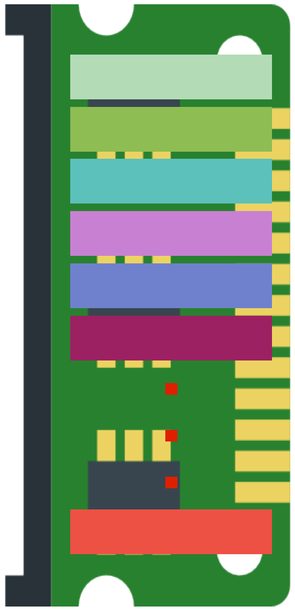


Processor



Cache

Divided in ***blocks***



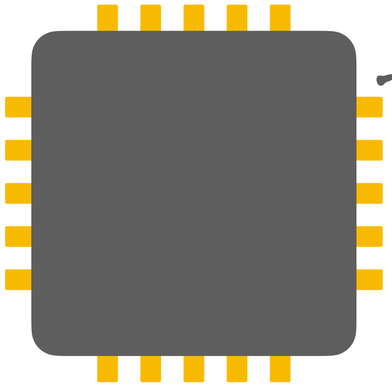
Main memory
(DRAM)

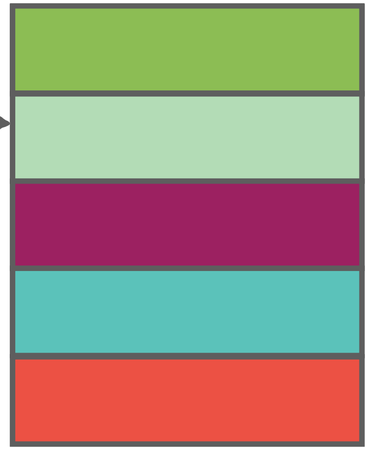
Caches 101

```
x = array[idx];
```

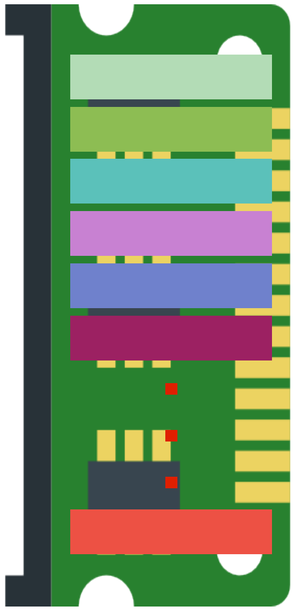
1. Address translation

0x401000





Divided in ***blocks***



Processor

Cache

Main memory (DRAM)

Caches 101

Results in a ***cache hit*** or ***miss***

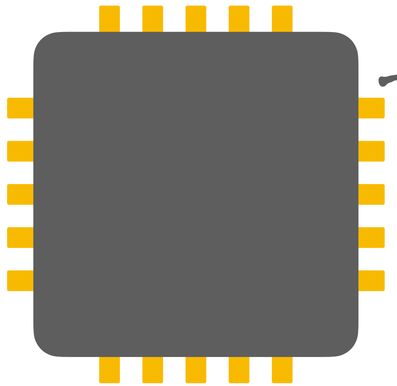
Divided in ***blocks***

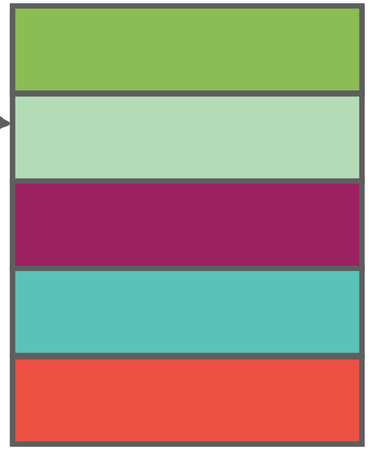
```
x = array[idx];
```

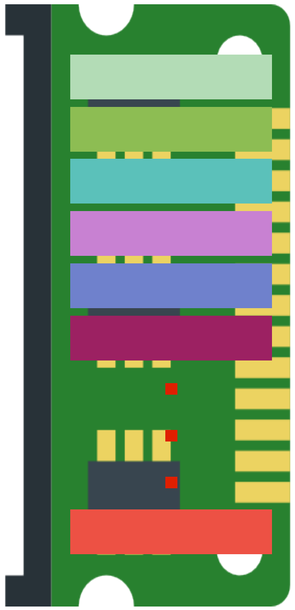
1. Address translation

2. Cache lookup

0x401000





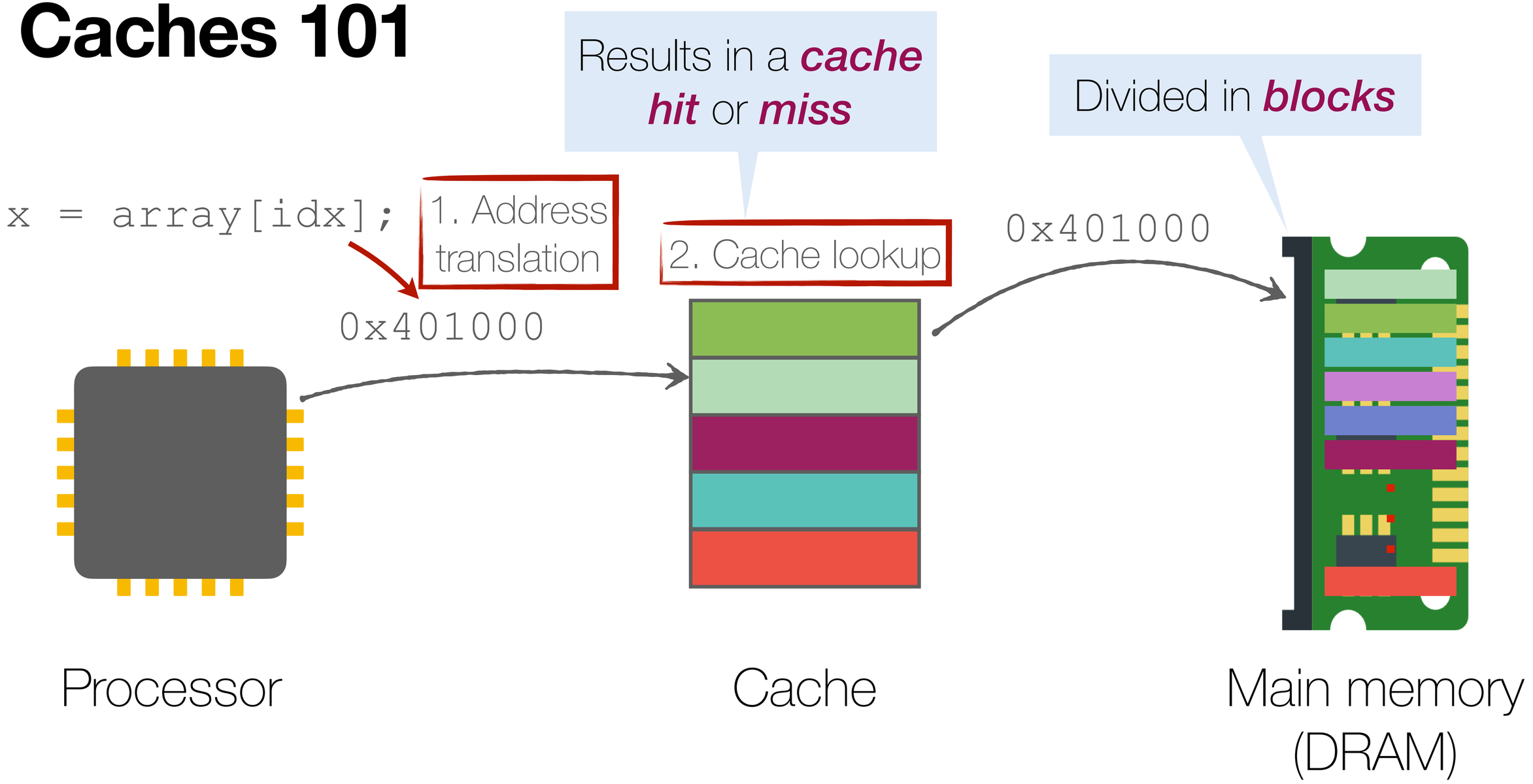


Processor

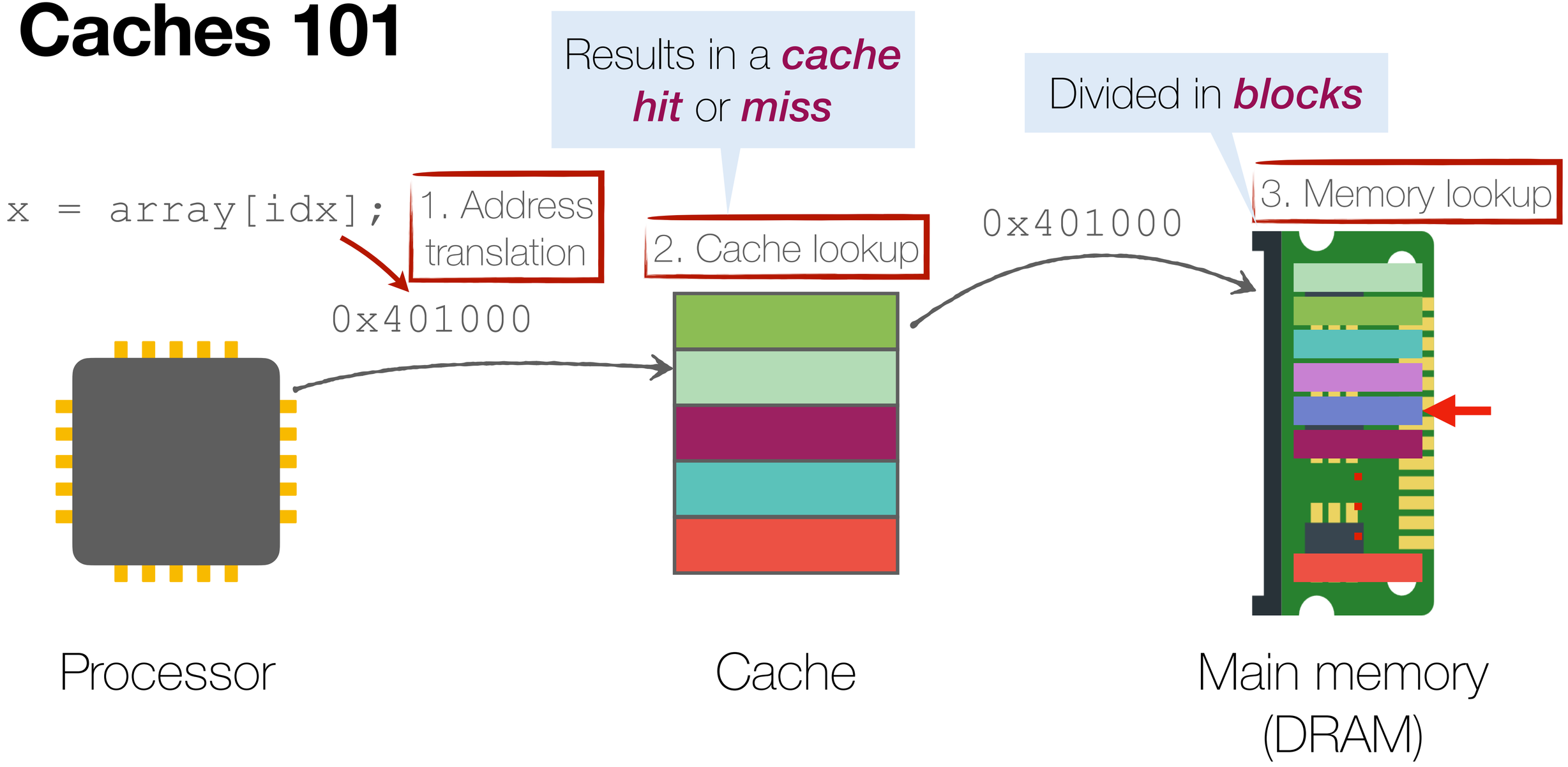
Cache

Main memory (DRAM)

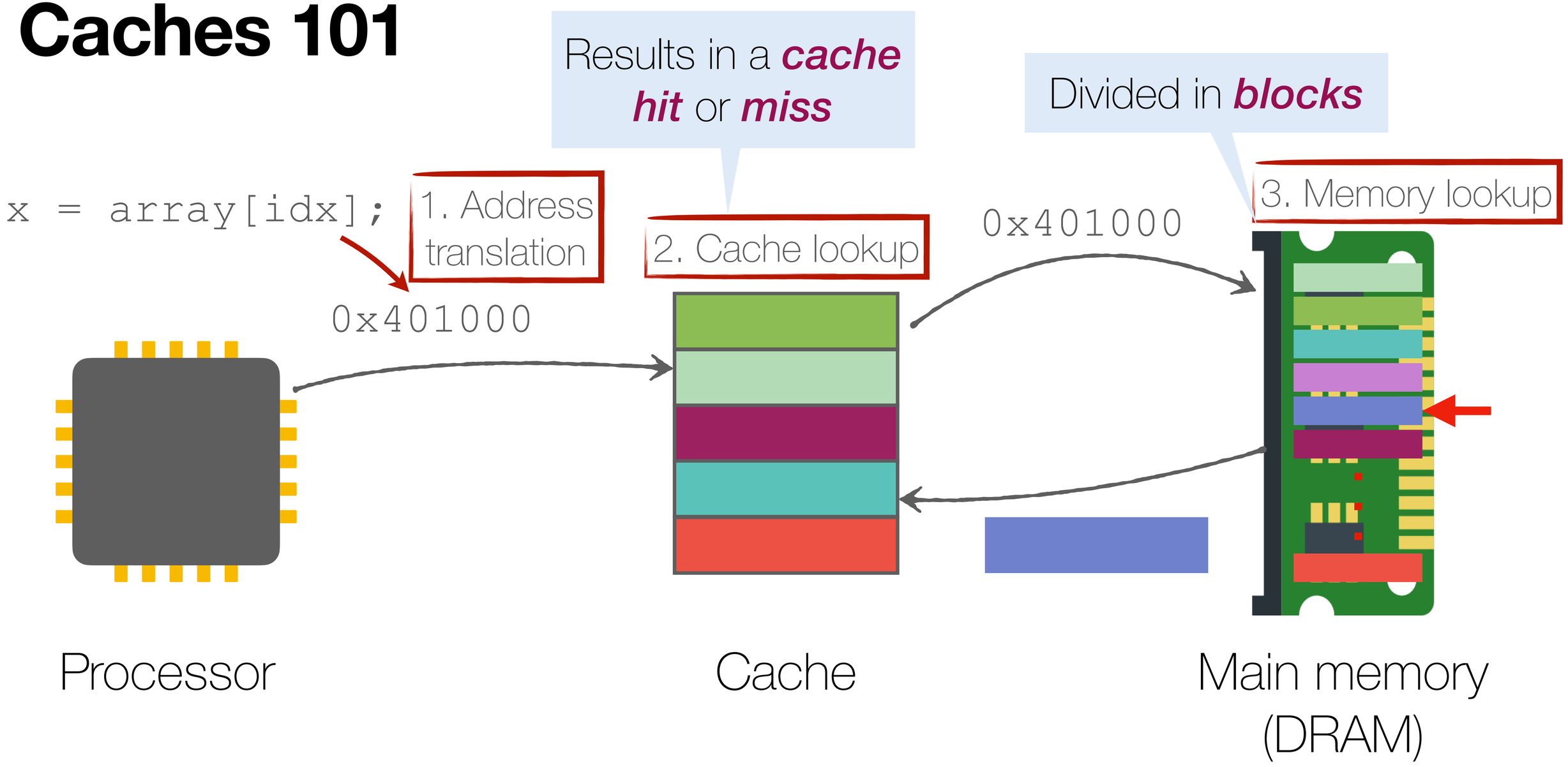
Caches 101


Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
2. Cache lookup
0x401000
0x401000
Processor
Cache
Main memory (DRAM)

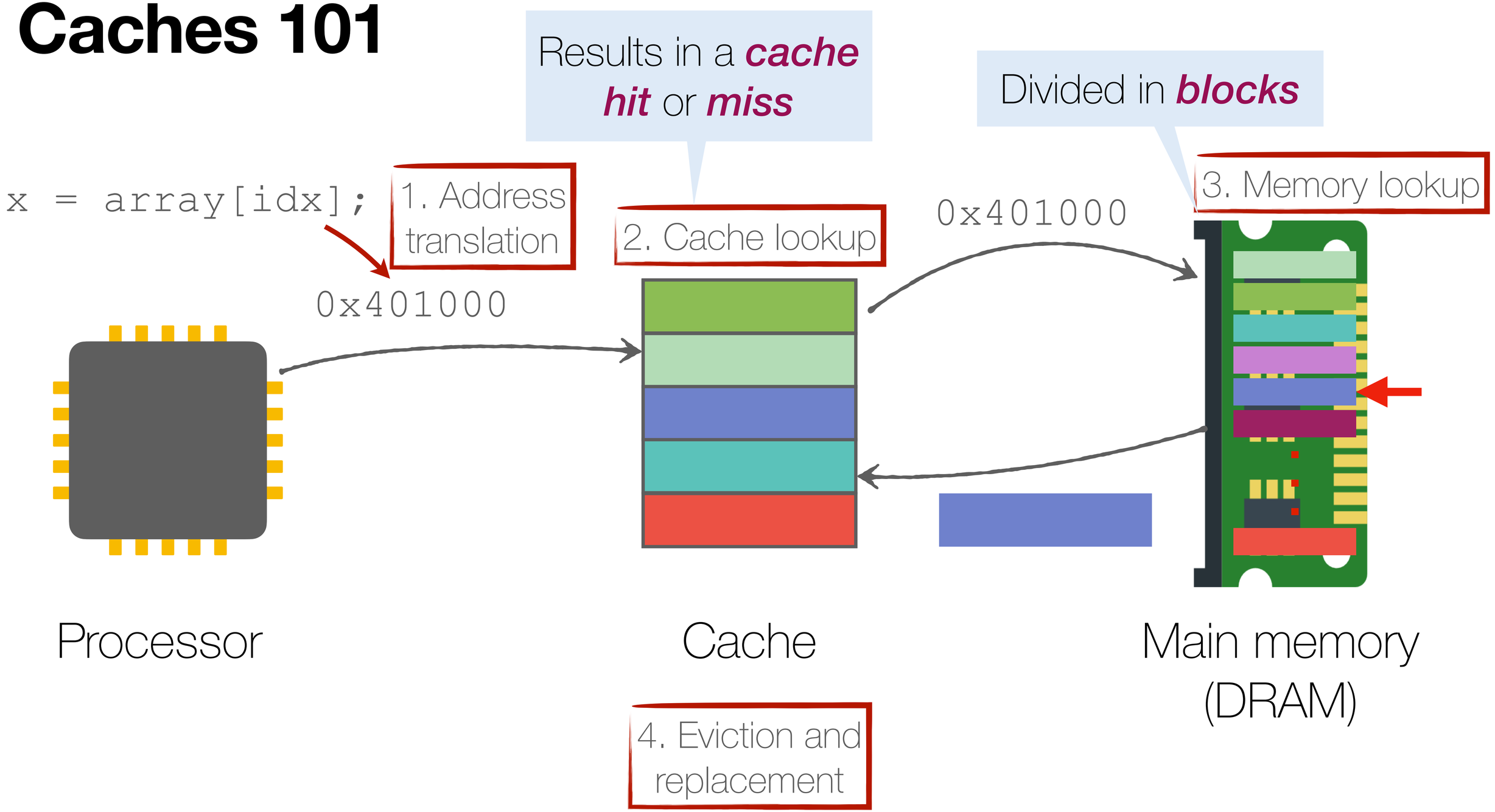
Caches 101


Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
0x401000
2. Cache lookup
0x401000
3. Memory lookup
Processor
Cache
Main memory (DRAM)

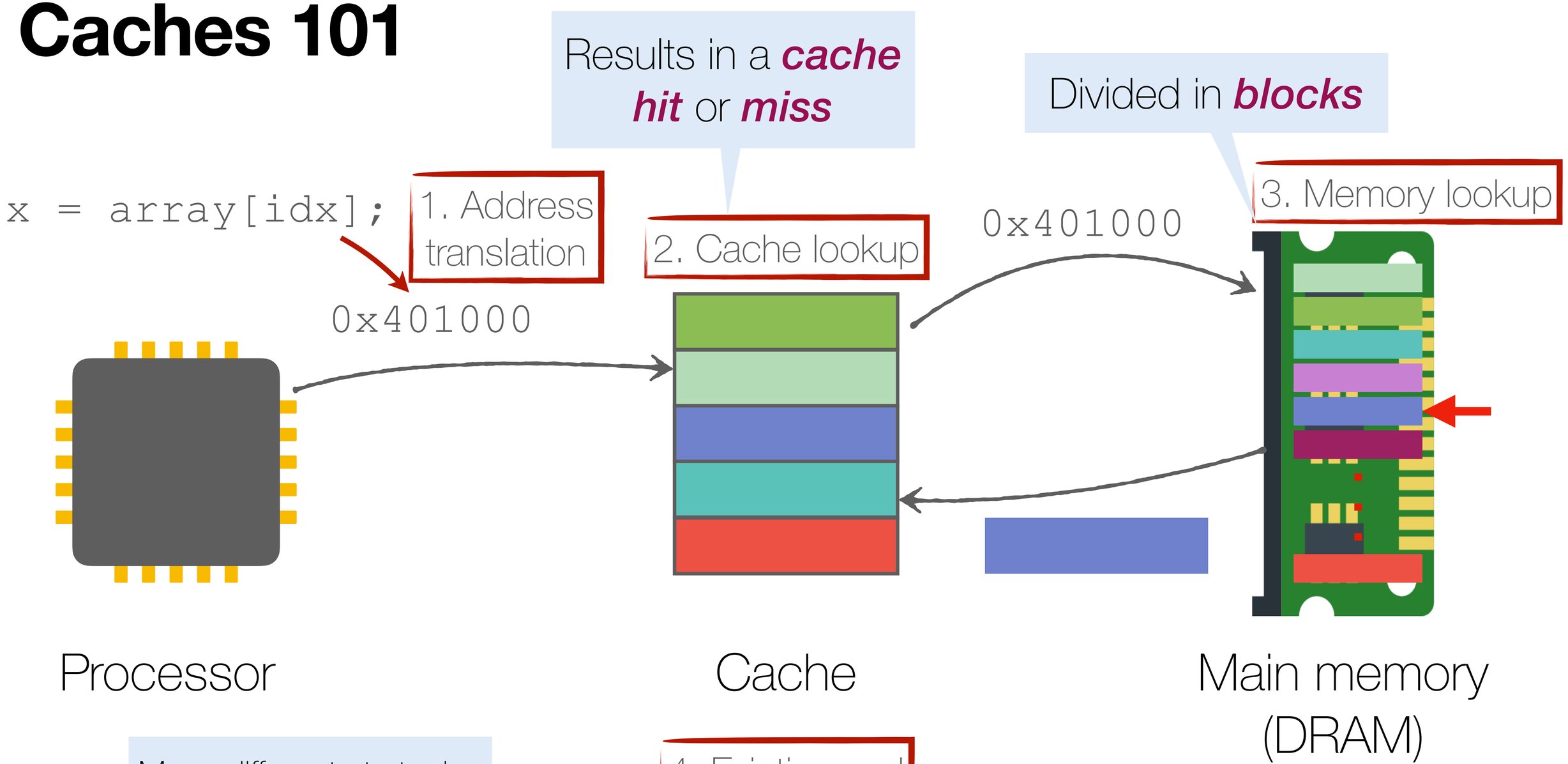
Caches 101


Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
0x401000
2. Cache lookup
0x401000
3. Memory lookup
Processor
Cache
Main memory (DRAM)

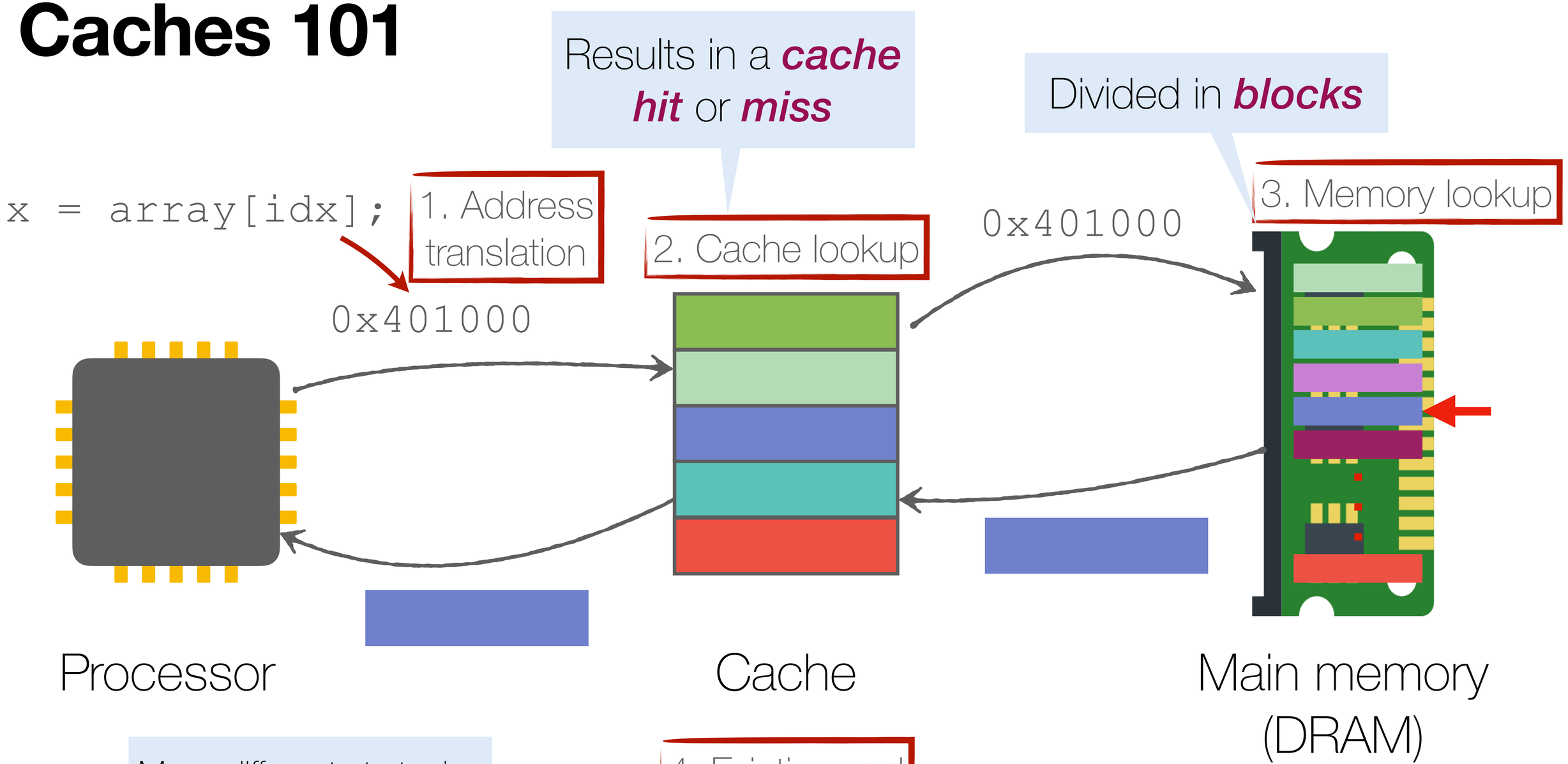
Caches 101


Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
0x401000
2. Cache lookup
0x401000
3. Memory lookup
Processor
Cache
Main memory (DRAM)
4. Eviction and replacement

Caches 101

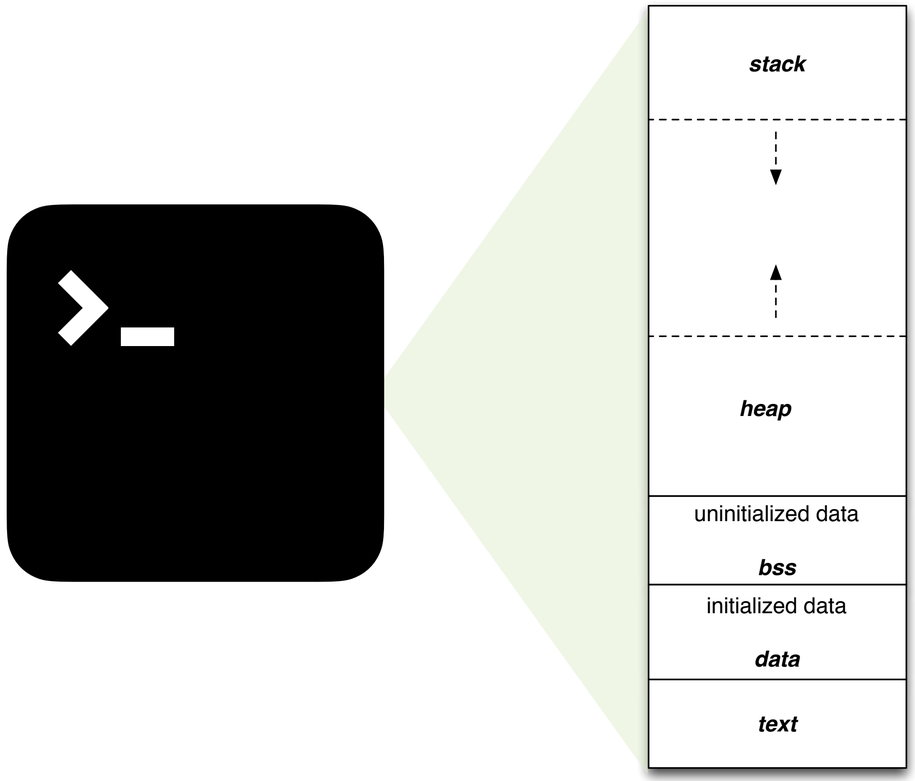

Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
0x401000
2. Cache lookup
0x401000
3. Memory lookup
Processor
Cache
Main memory (DRAM)
Many different strategies (replacement policies)
4. Eviction and replacement

Caches 101


Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
2. Cache lookup
3. Memory lookup
0x401000
0x401000
Processor
Cache
Main memory (DRAM)
Many different strategies (replacement policies)
4. Eviction and replacement

Virtual and physical addresses

Virtual and physical addresses

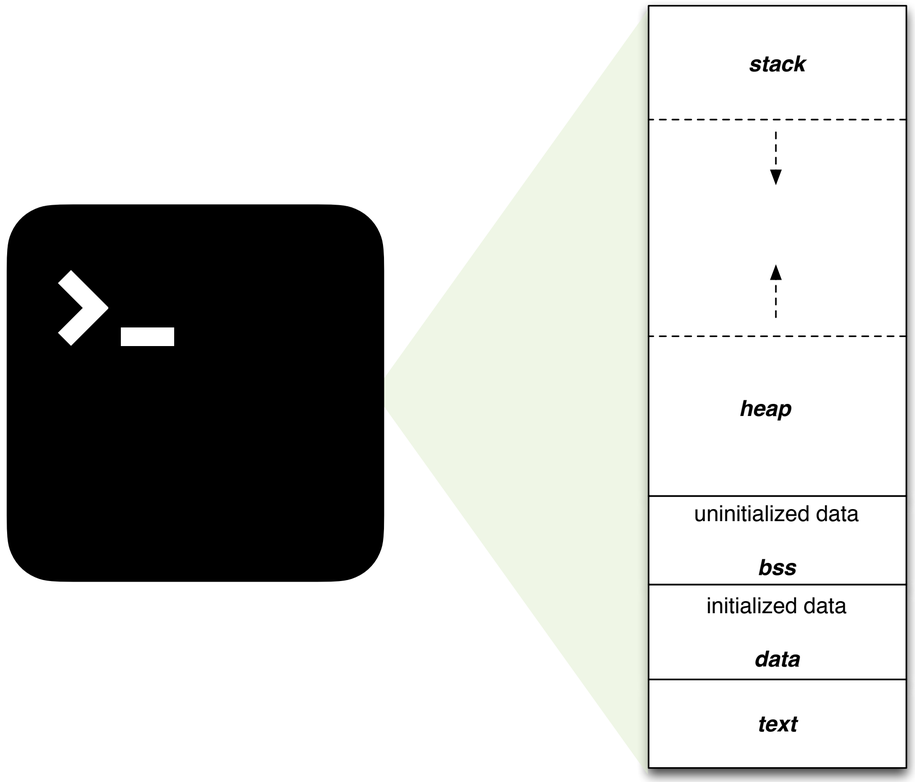


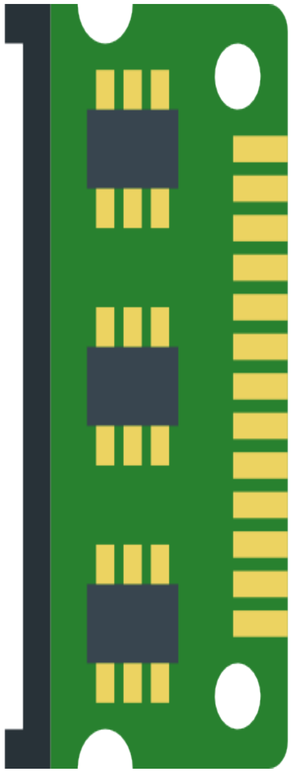
Virtual memory

Abstraction allowing processes to treat ***memory*** as ***linear*** and ***isolated***

Virtual memory layout from https://en.wikipedia.org/wiki/File:Program_memory_layout.pdf

Virtual and physical addresses



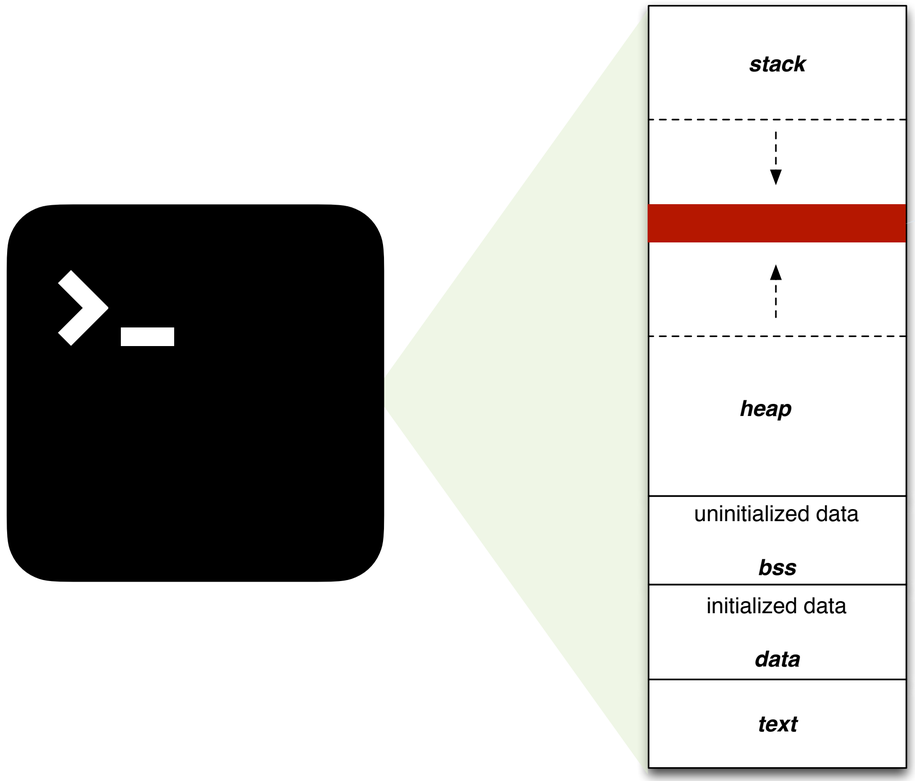


Virtual memory
Abstraction allowing processes to treat ***memory*** as ***linear*** and ***isolated***

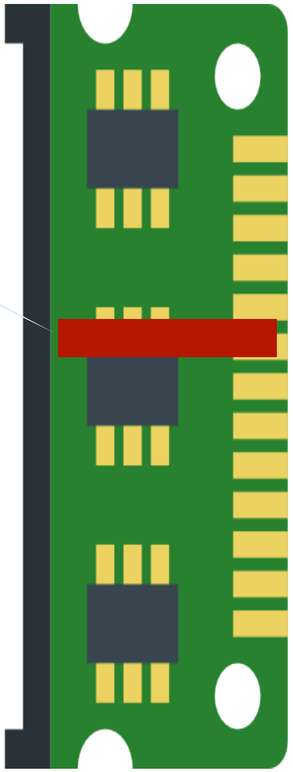
OS + ***memory management unit***
map ***virtual*** addresses to ***physical*** addresses

Virtual memory layout from https://en.wikipedia.org/wiki/File:Program_memory_layout.pdf

Virtual and physical addresses



Virtual and physical memory divided in ***pages*** of 2^p bytes

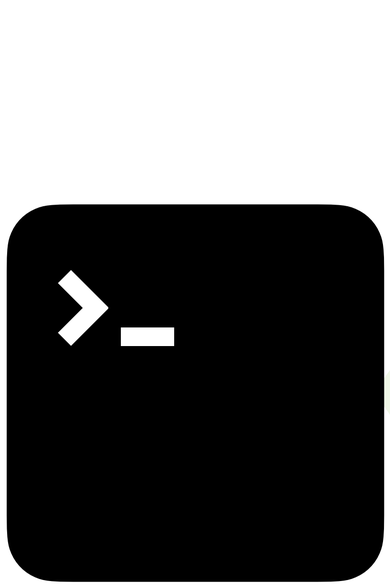


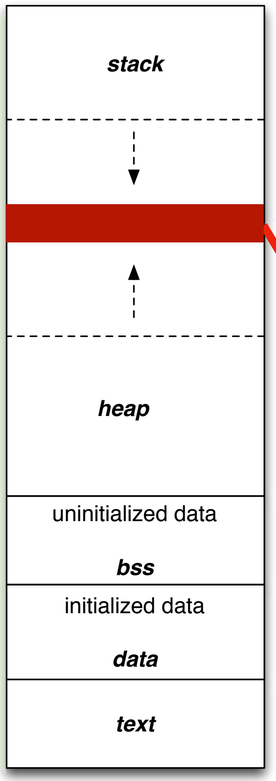
Virtual memory
Abstraction allowing processes to treat ***memory*** as ***linear*** and ***isolated***

OS + ***memory management unit***
map ***virtual*** addresses to ***physical*** addresses

Virtual memory layout from https://en.wikipedia.org/wiki/File:Program_memory_layout.pdf

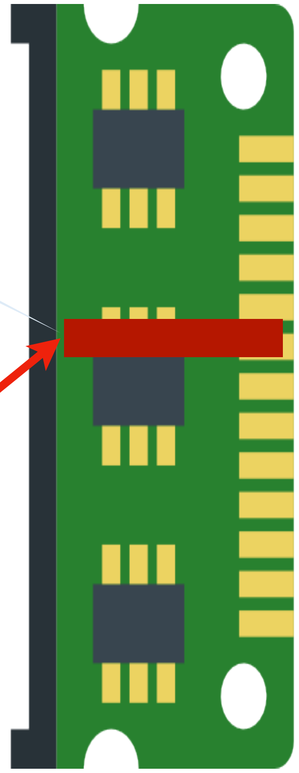
Virtual and physical addresses





Virtual and physical memory divided in ***pages*** of 2^p bytes

Mapping is stored in ***page table***
Last p bits are the same

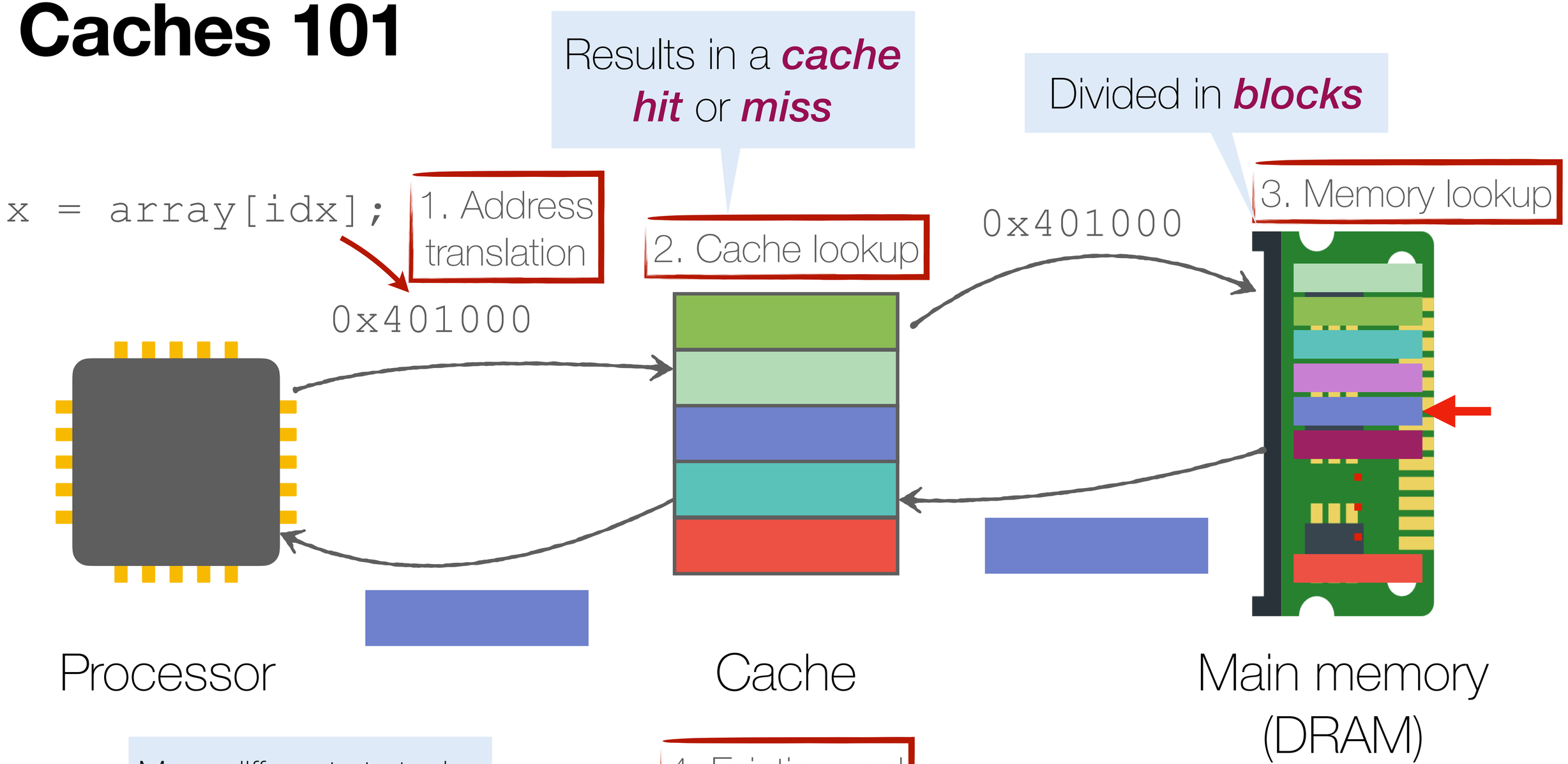


Virtual memory
Abstraction allowing processes to treat ***memory*** as ***linear*** and ***isolated***

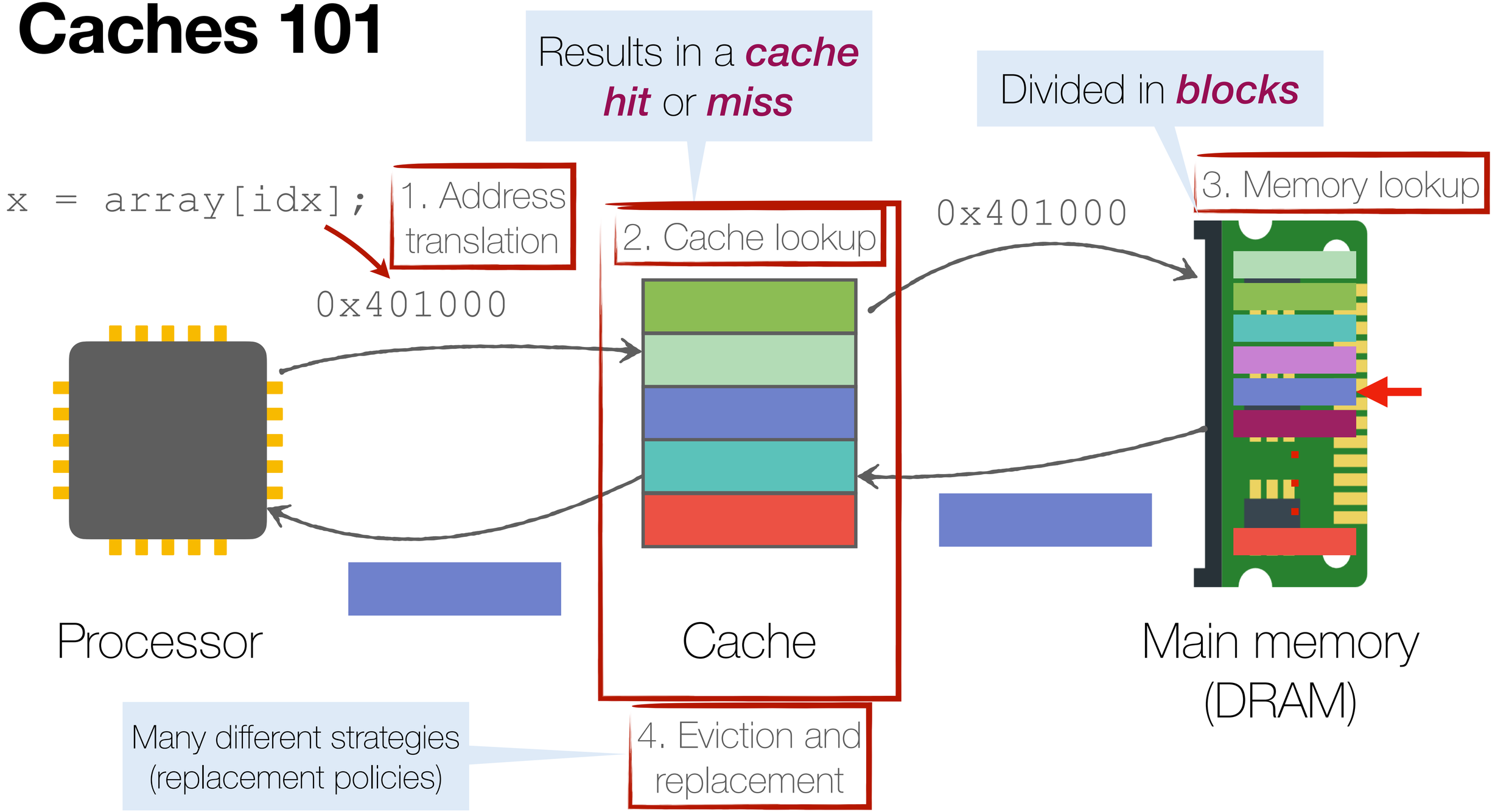
OS + ***memory management unit***
map ***virtual*** addresses to ***physical*** addresses

Virtual memory layout from https://en.wikipedia.org/wiki/File:Program_memory_layout.pdf

Caches 101

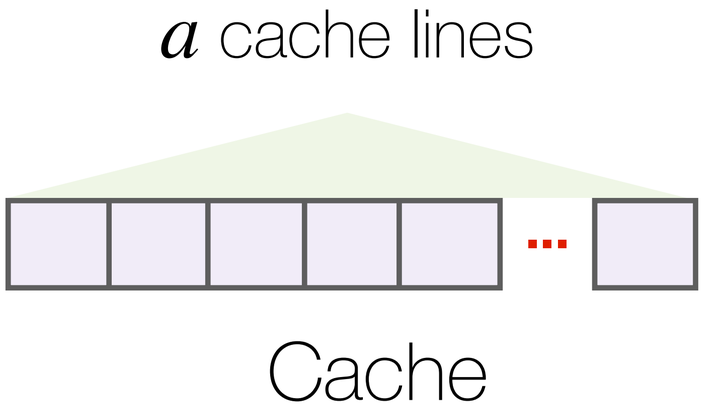

Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
2. Cache lookup
3. Memory lookup
0x401000
0x401000
Processor
Cache
Main memory (DRAM)
Many different strategies (replacement policies)
4. Eviction and replacement

Caches 101


Results in a *cache hit* or *miss*
Divided in *blocks*
x = array[idx];
1. Address translation
0x401000
2. Cache lookup
0x401000
3. Memory lookup
Processor
Cache
Main memory (DRAM)
Many different strategies (replacement policies)
4. Eviction and replacement

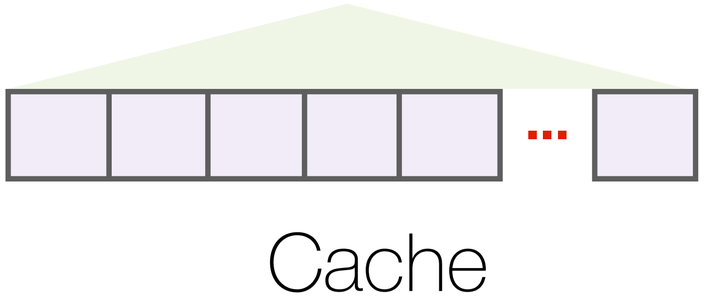
Fully associative caches

Fully associative caches



Divided in ***cache lines***
each one storing a ***block***

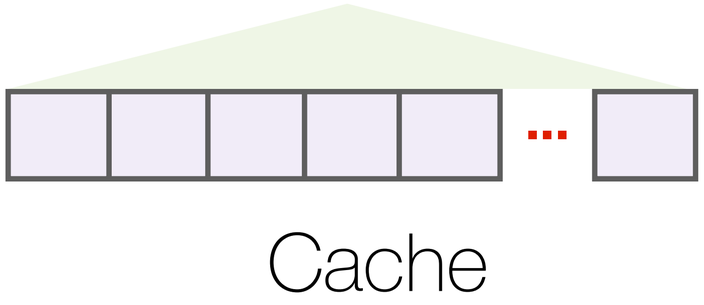
Fully associative caches



Memory divided in ***blocks*** of 2^l bits

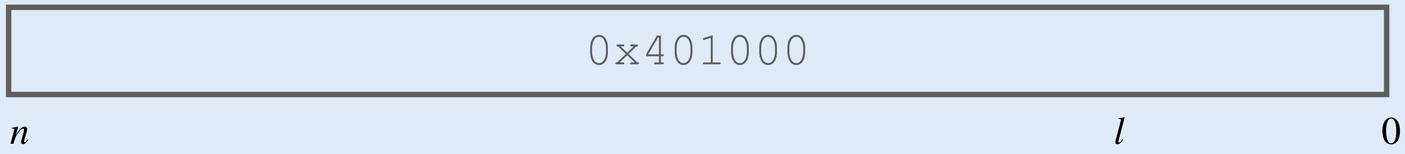
Divided in ***cache lines***
each one storing a ***block***

Fully associative caches



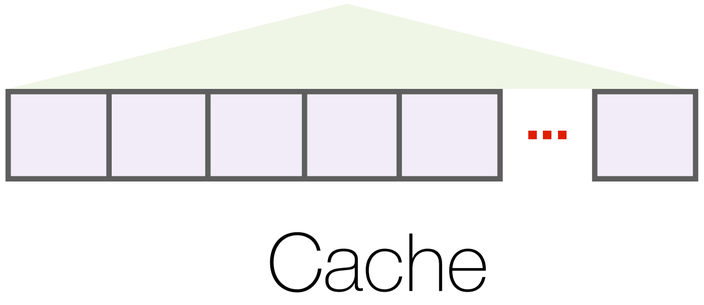
Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache



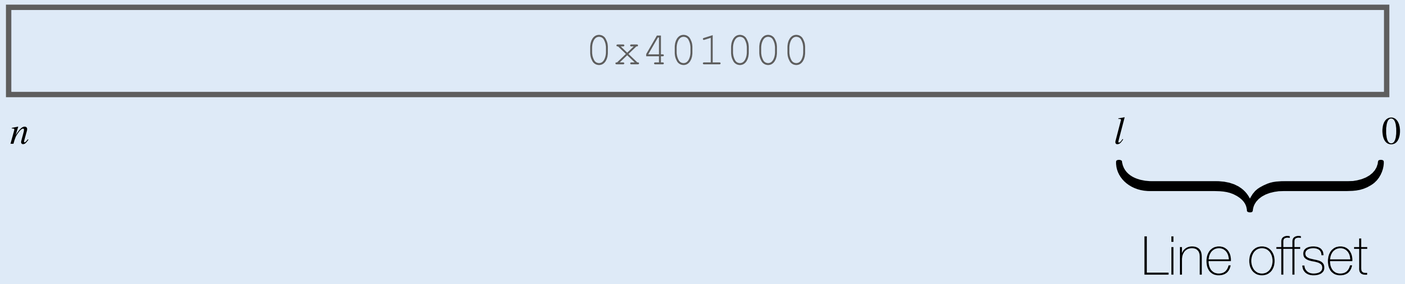
Divided in ***cache lines***
each one storing a ***block***

Fully associative caches



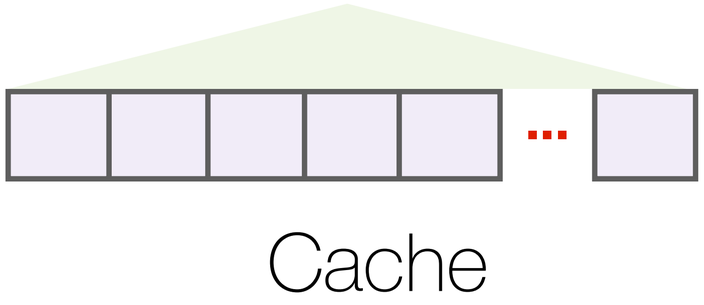
Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache



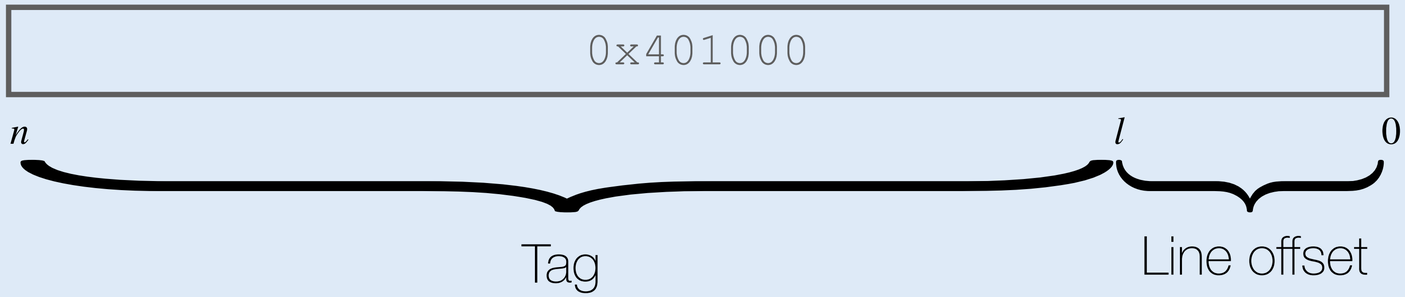
Divided in ***cache lines***
each one storing a ***block***

Fully associative caches



Memory divided in ***blocks*** of 2^l bits

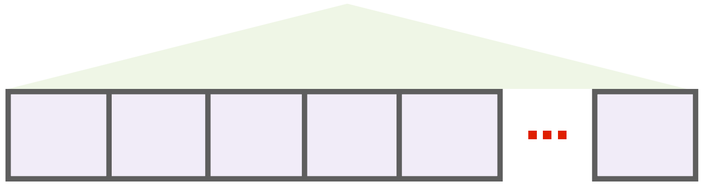
Mapping ***addresses*** to cache



Divided in ***cache lines***
each one storing a ***block***

Fully associative caches

a cache lines



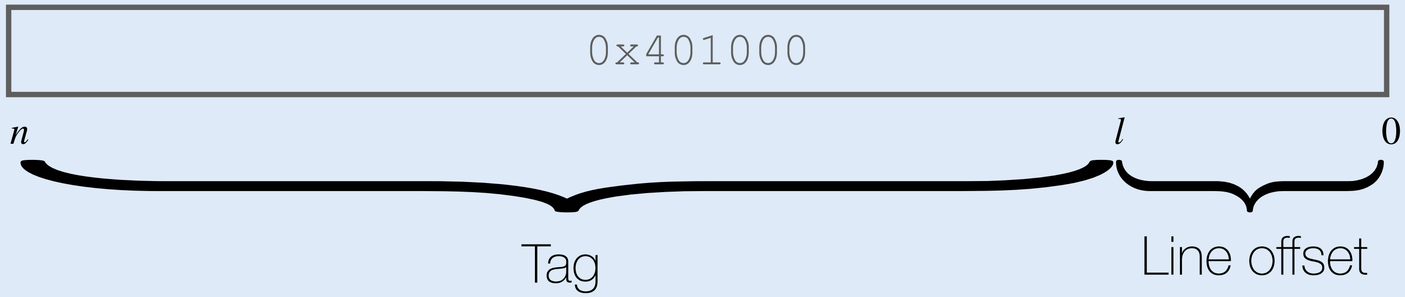
Cache

Divided in ***cache lines***
each one storing a ***block***

Memory divided in ***blocks*** of 2^l bits

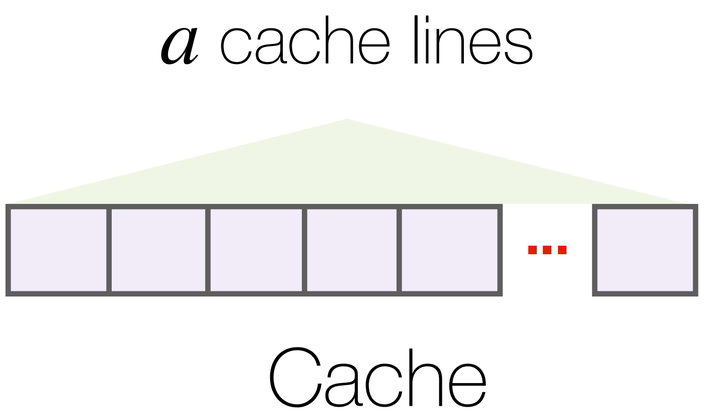
Mapping ***addresses*** to cache

Physical address



A ***memory block*** can fit in ***any*** of the cache lines

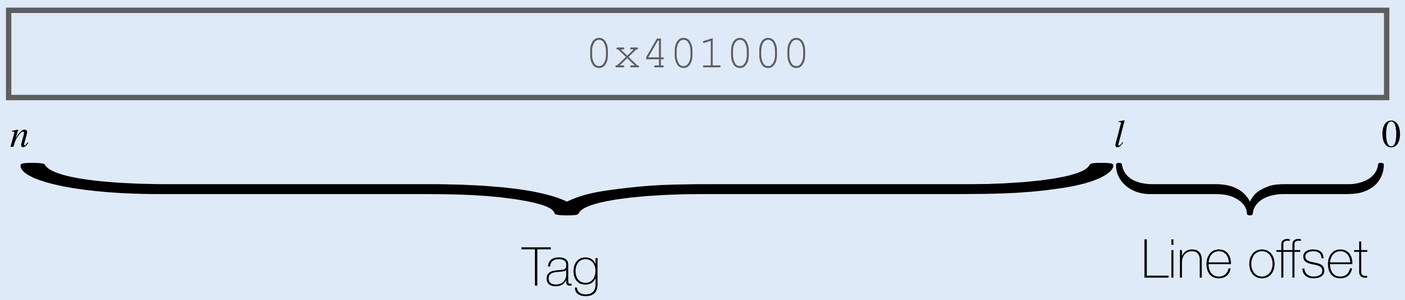
Fully associative caches



Divided in ***cache lines*** each one storing a ***block***

Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache

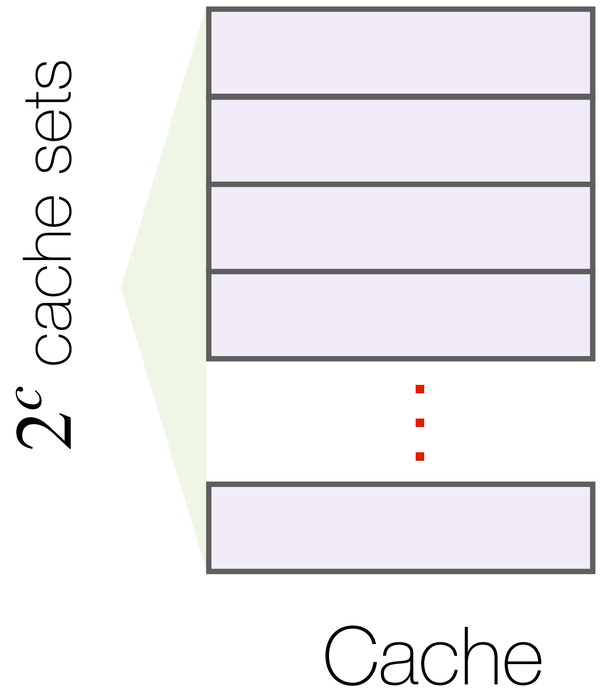


A ***memory block*** can fit in ***any*** of the cache lines

Expensive to implement

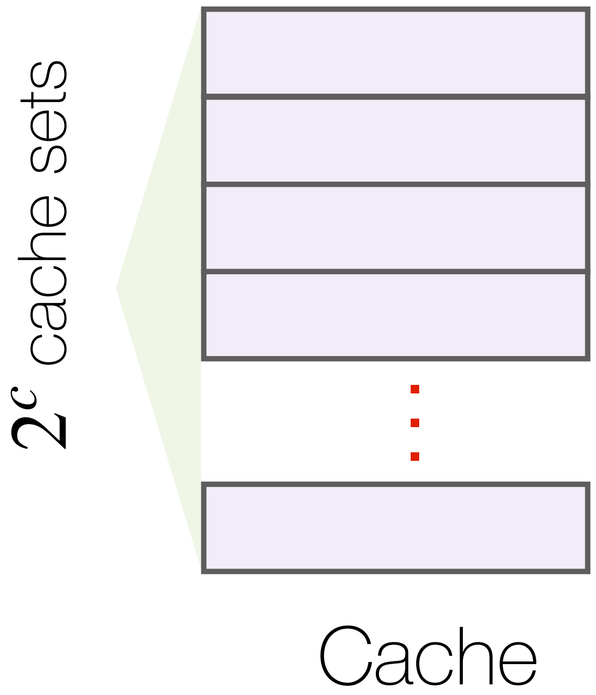
Directly mapped cache

Directly mapped cache



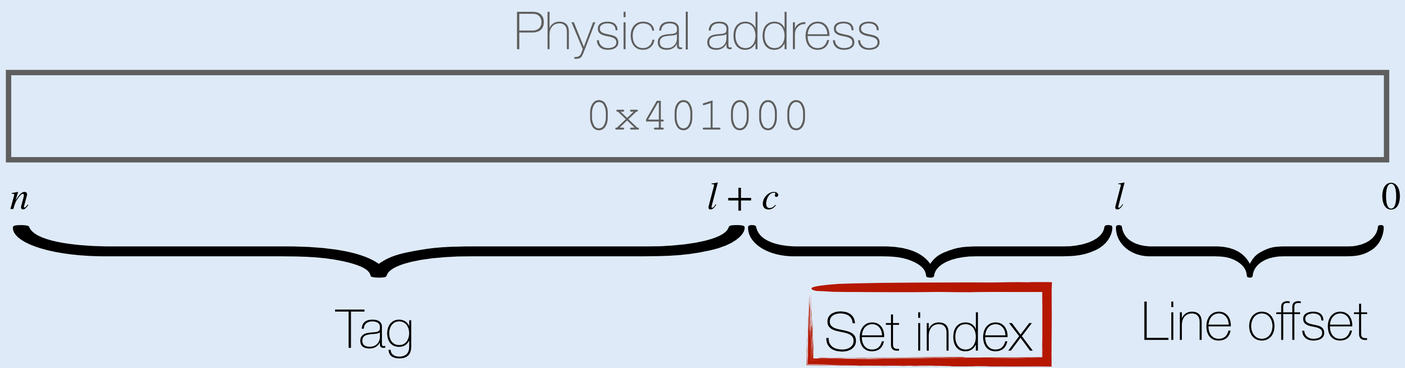
Divided in ***cache sets*** each one with 1 ***cache line***

Directly mapped cache



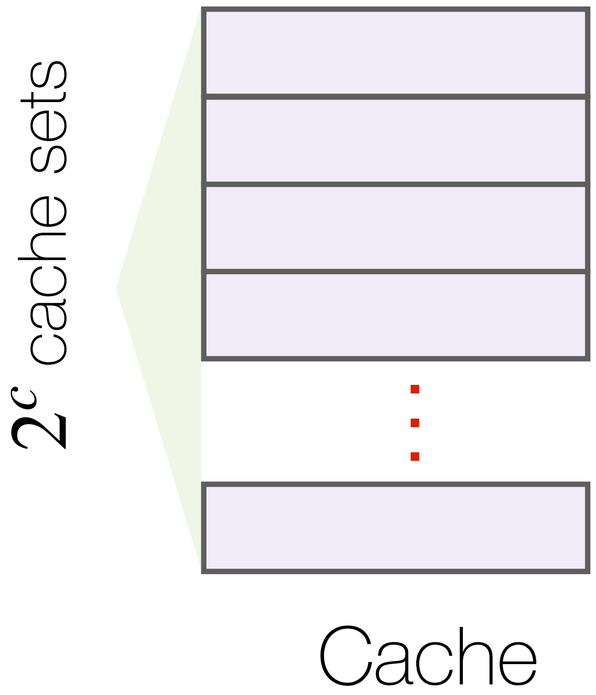
Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache



Divided in ***cache sets*** each one with 1 ***cache line***

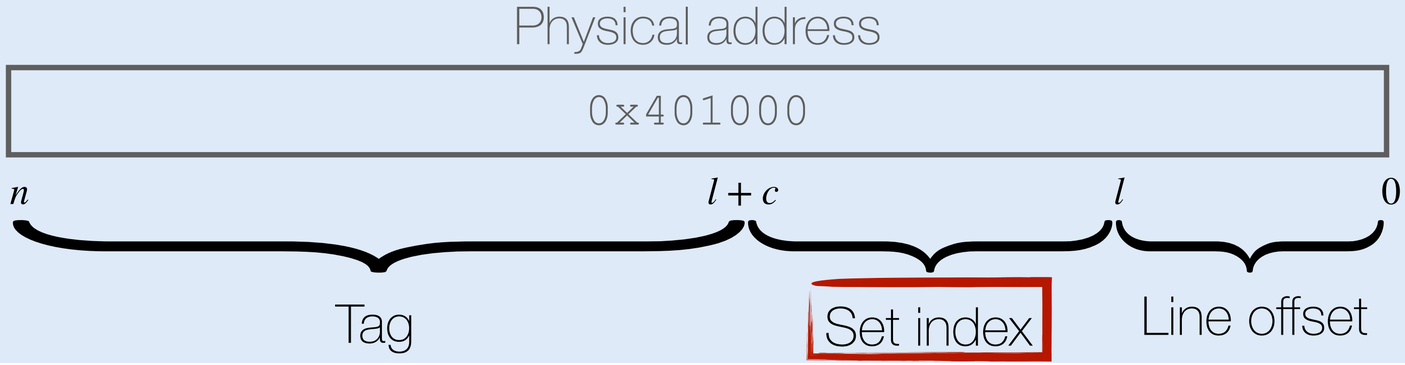
Directly mapped cache



Divided in ***cache sets*** each one with 1 ***cache line***

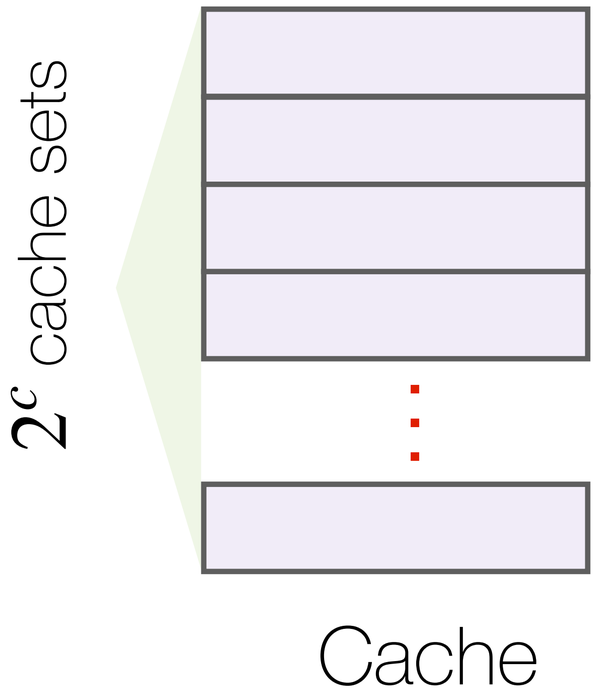
Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache



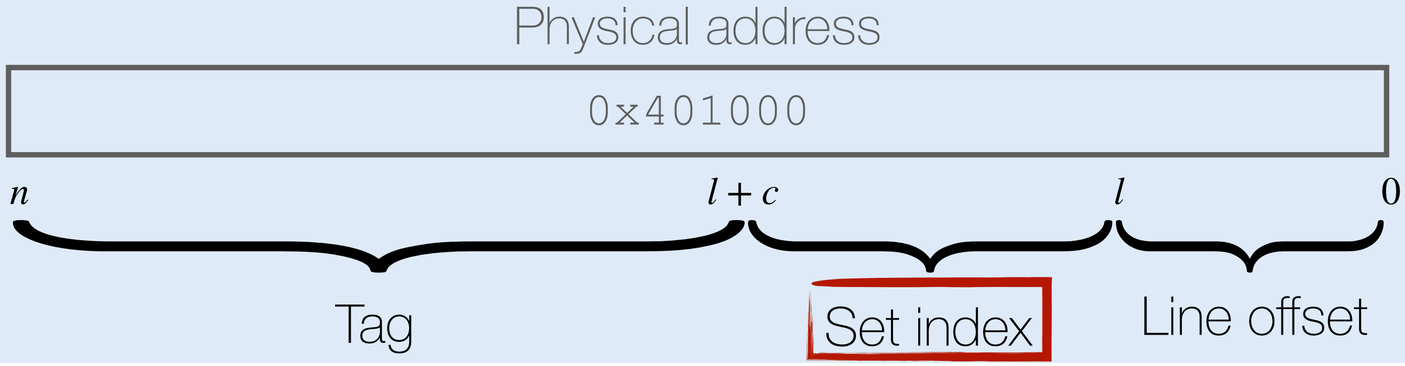
Addresses with the ***same set index*** map to the ***same cache set***

Directly mapped cache



Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache



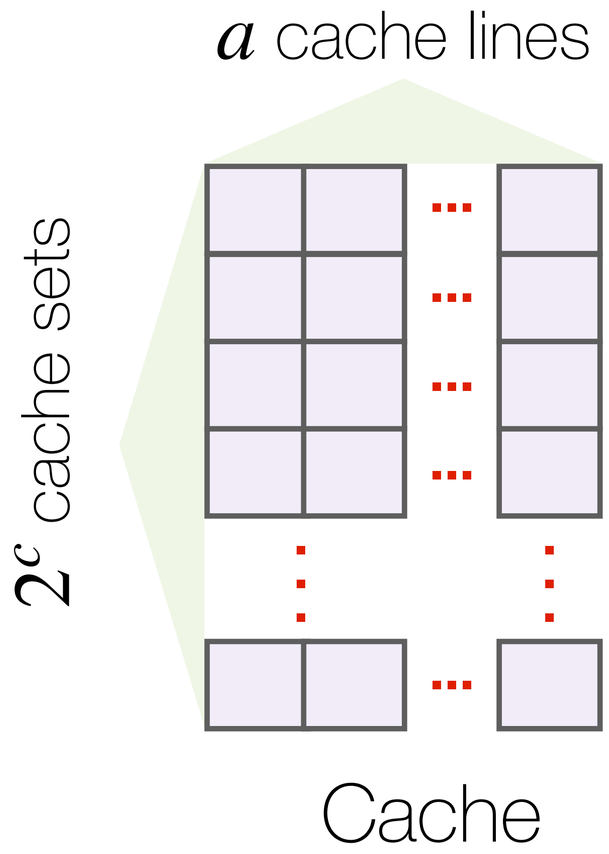
Addresses with the ***same set index*** map to the ***same cache set***

Divided in ***cache sets*** each one with 1 ***cache line***

Only ***1 block*** per set

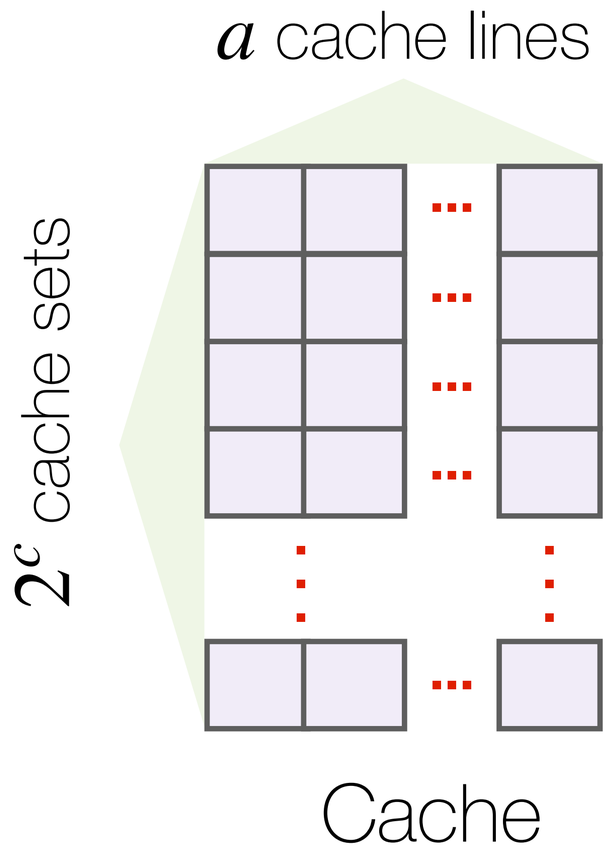
Set-associative caches

Set-associative caches



Divided in ***cache sets*** each one split into ***cache lines***

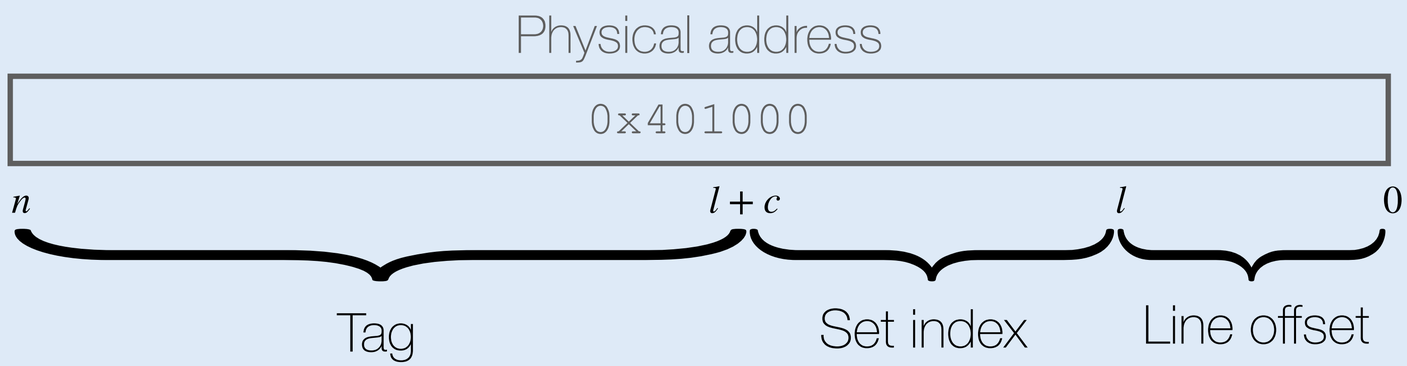
Set-associative caches



Divided in ***cache sets*** each one split into ***cache lines***

Memory divided in ***blocks*** of 2^l bits

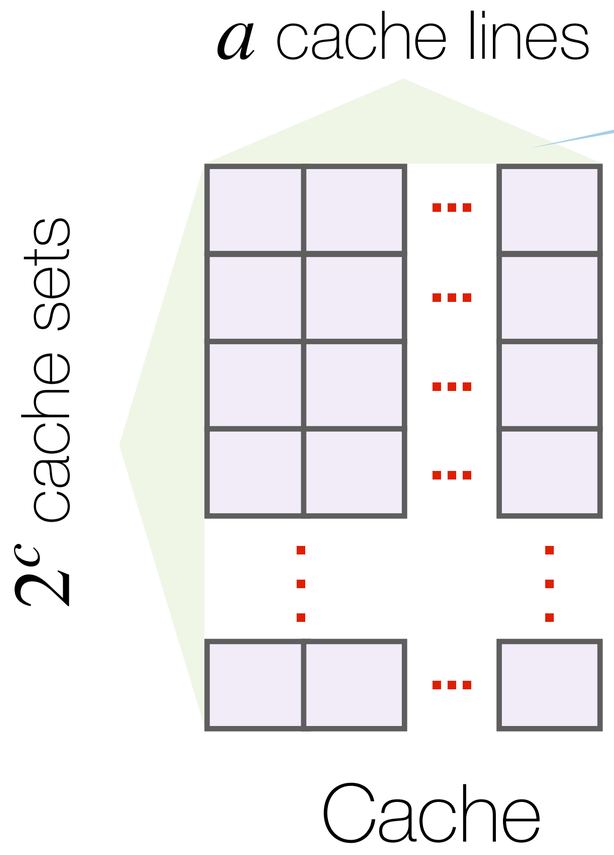
Mapping ***addresses*** to cache



Addresses with the ***same set index*** map to the ***same cache set***

Set-associative caches

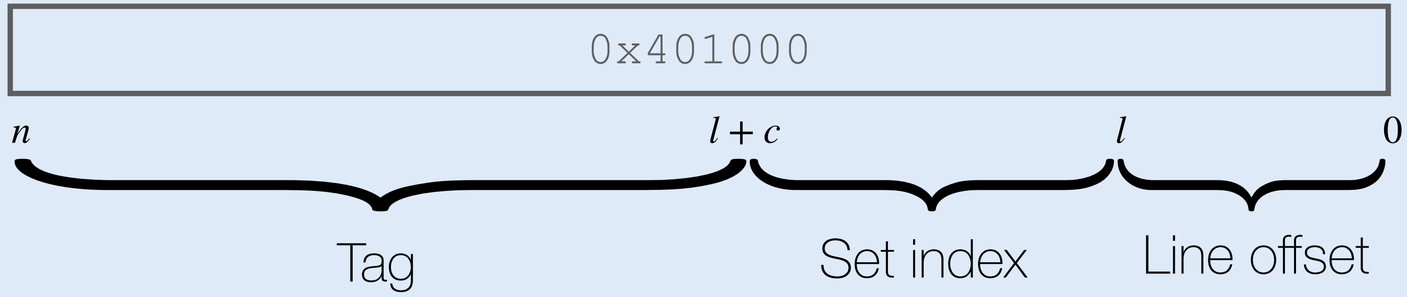
Intel Haswell
L1: 64 cache sets, 8 lines each
L3: 8192 cache sets, 16 lines each



Divided in ***cache sets*** each one split into ***cache lines***

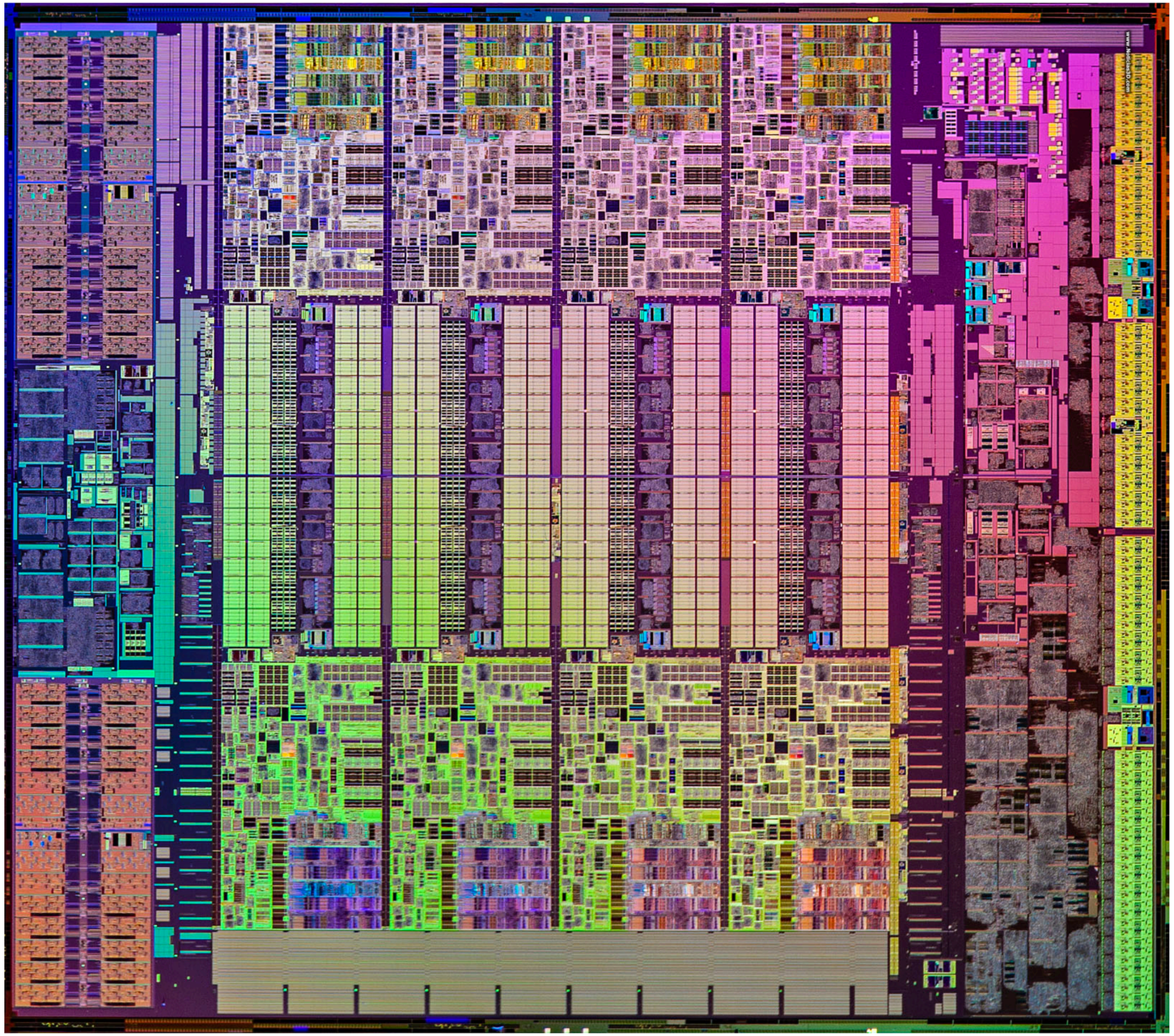
Memory divided in ***blocks*** of 2^l bits

Mapping ***addresses*** to cache



Addresses with the ***same set index*** map to the ***same cache set***

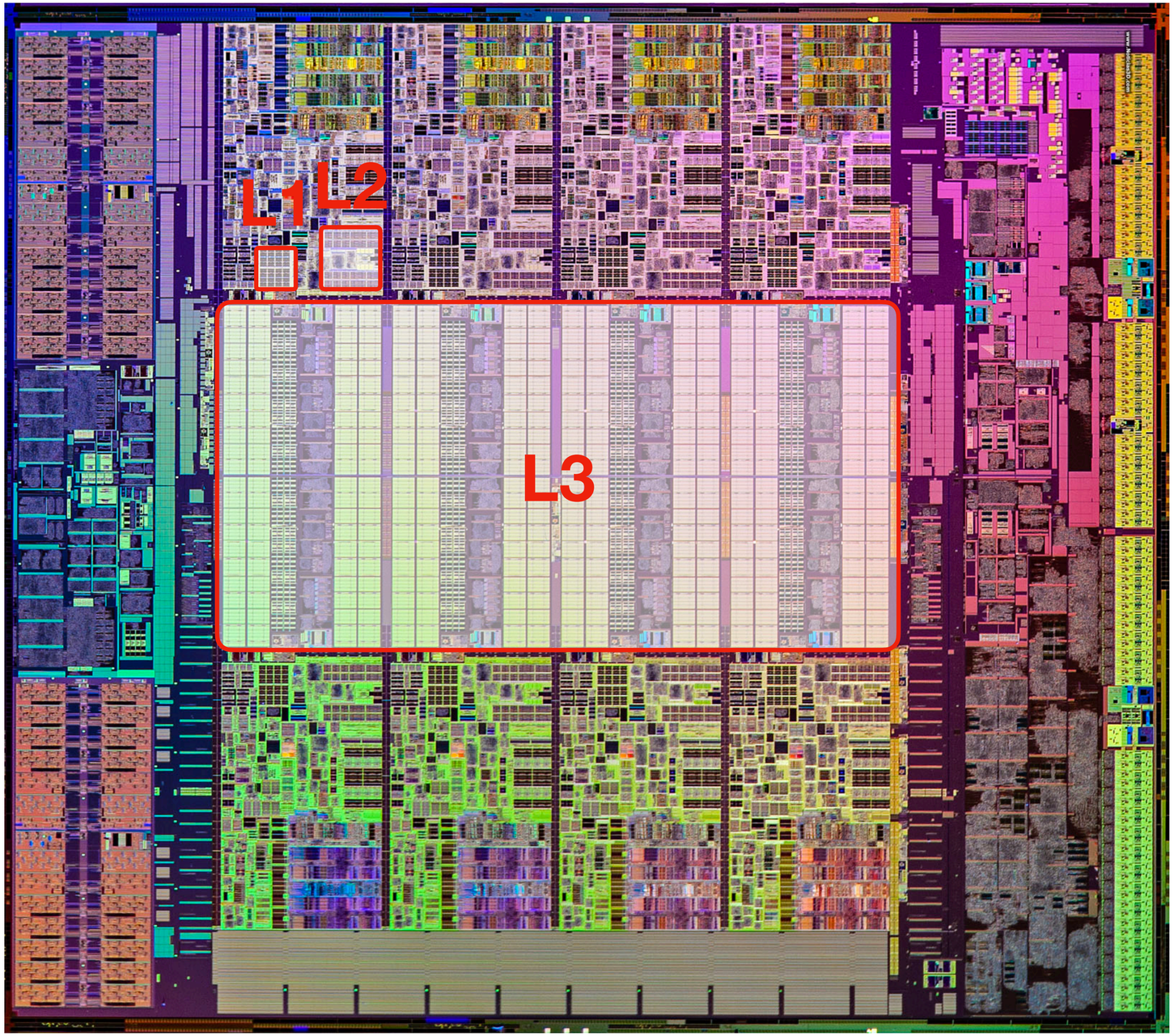
Cache hierarchy



Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

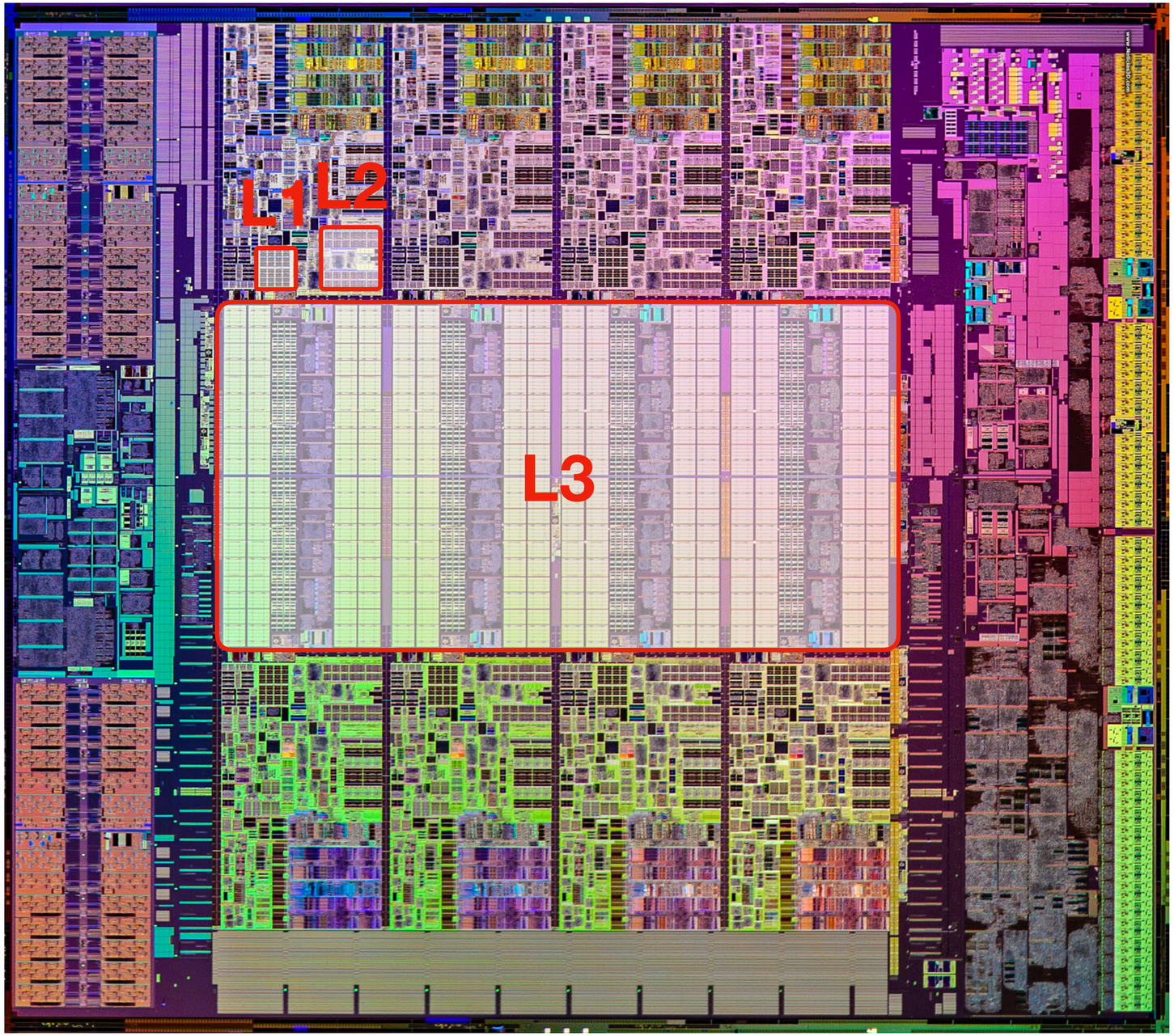


Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

Memory ***requests*** travel through the ***hierarchy***

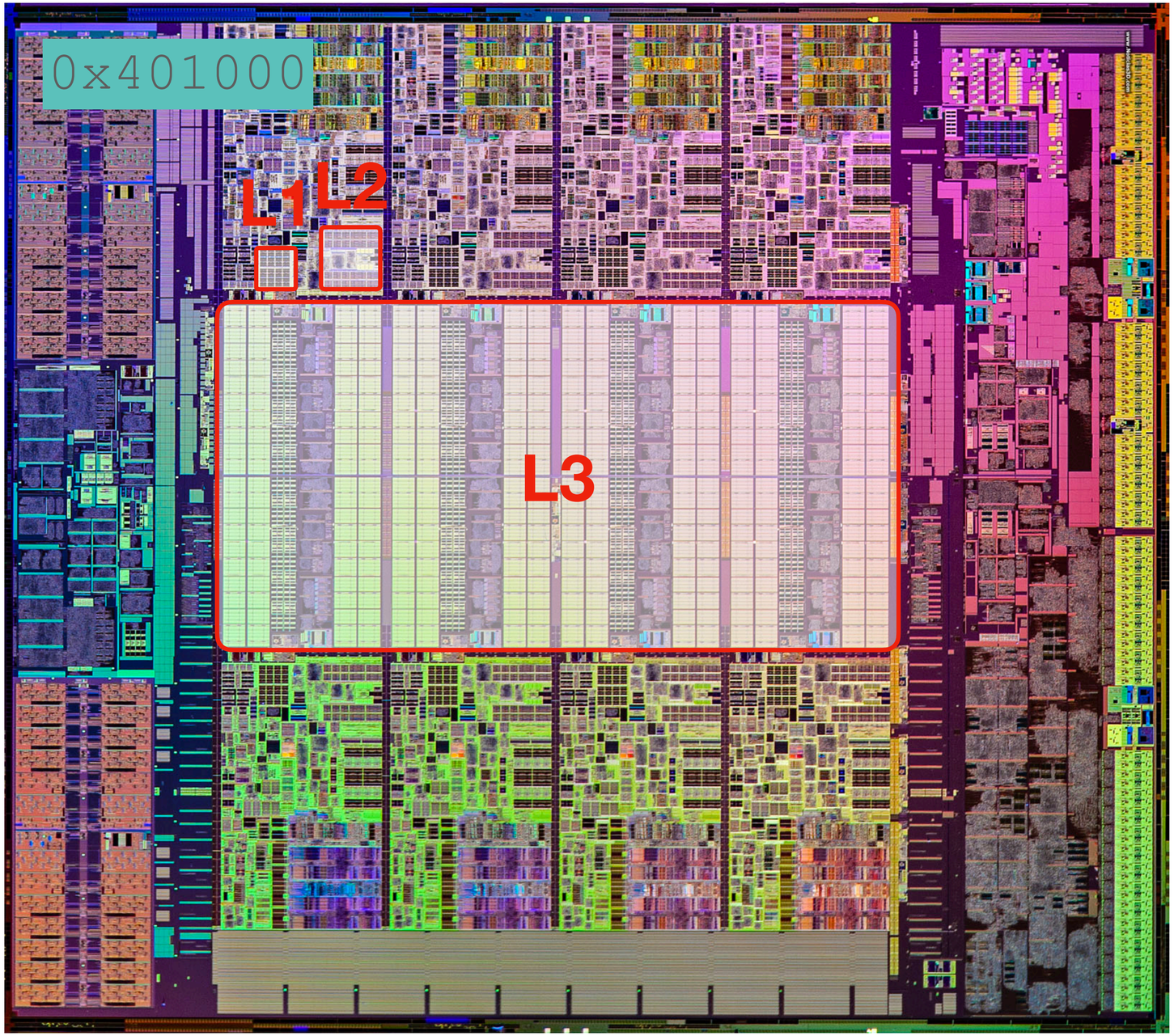


Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

Memory ***requests*** travel through the ***hierarchy***

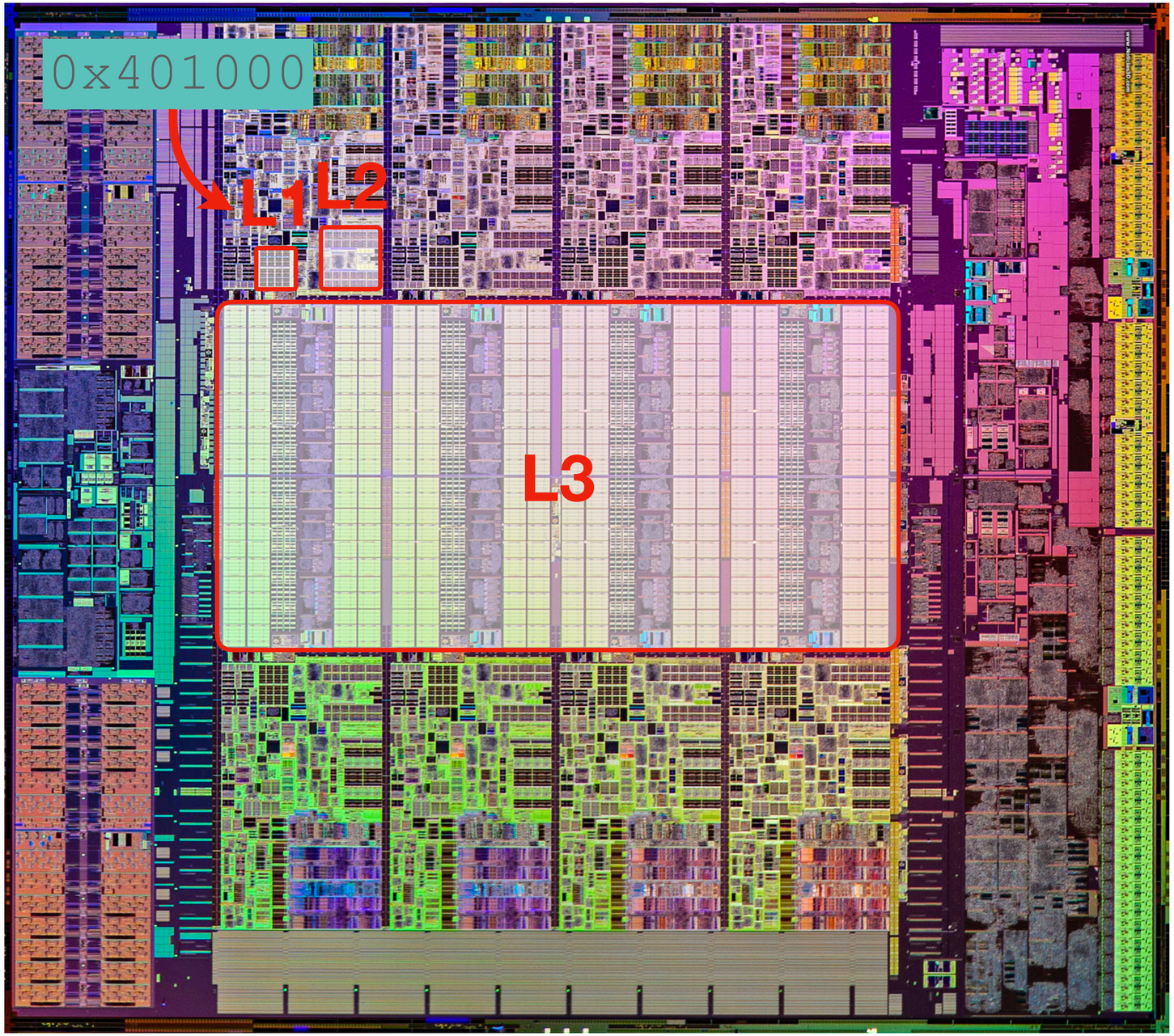


Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

Memory ***requests*** travel through the ***hierarchy***

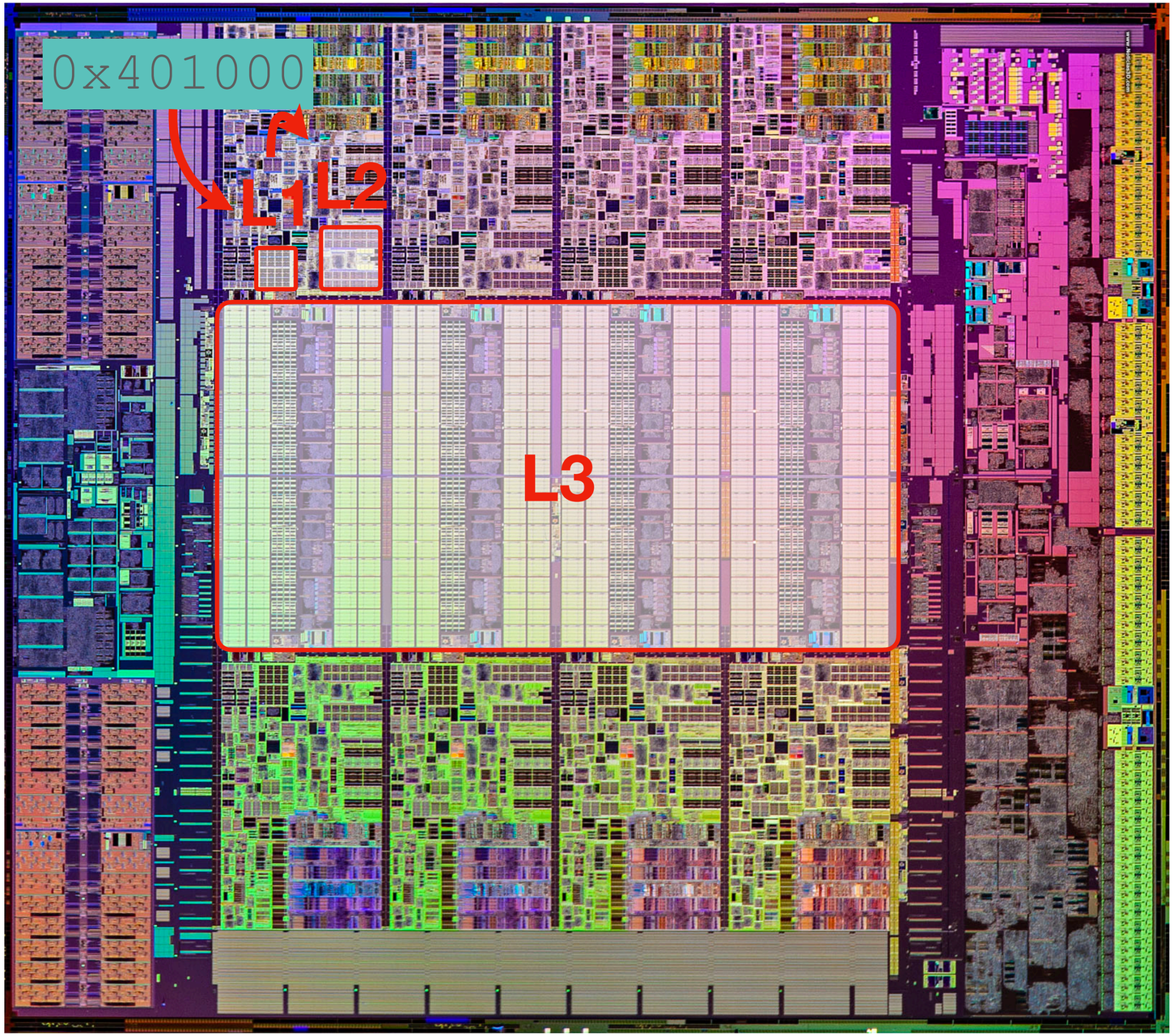


Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

Memory ***requests*** travel through the ***hierarchy***

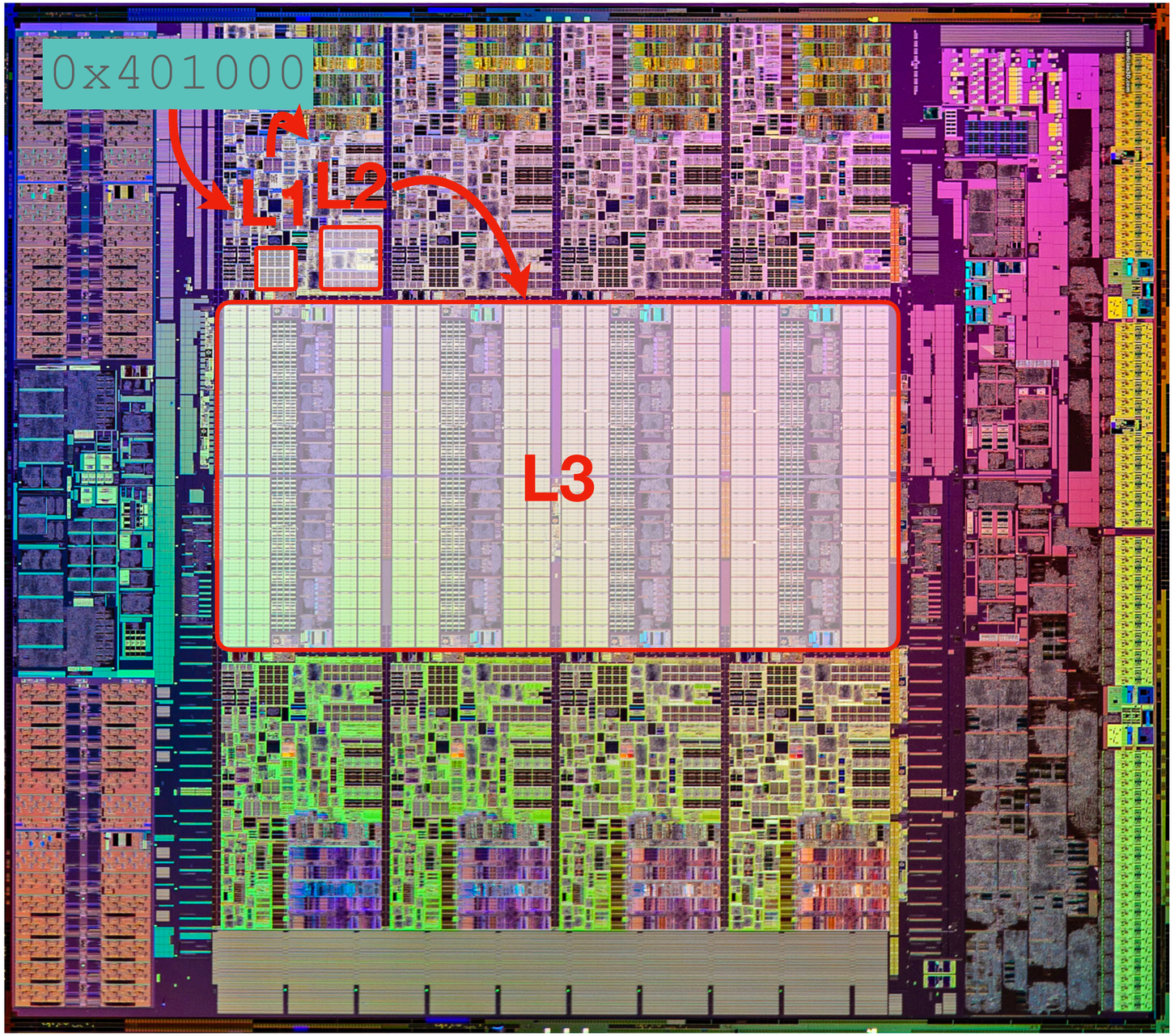


Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

Memory ***requests*** travel through the ***hierarchy***



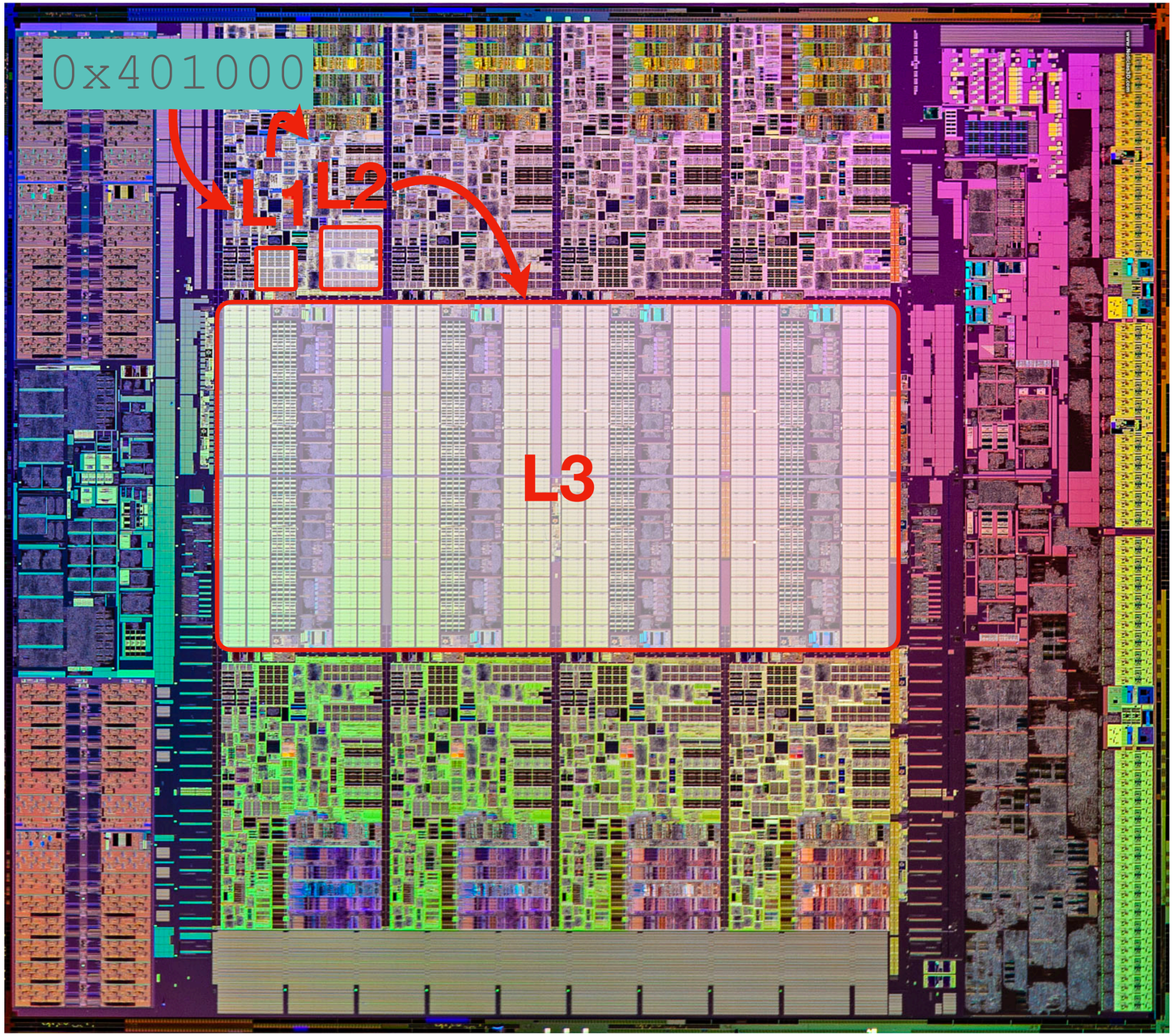
Picture of Intel “Haswell-E” Eight Core CPU

Cache hierarchy

Processors have ***multiple levels*** of caches

Memory ***requests*** travel through the ***hierarchy***

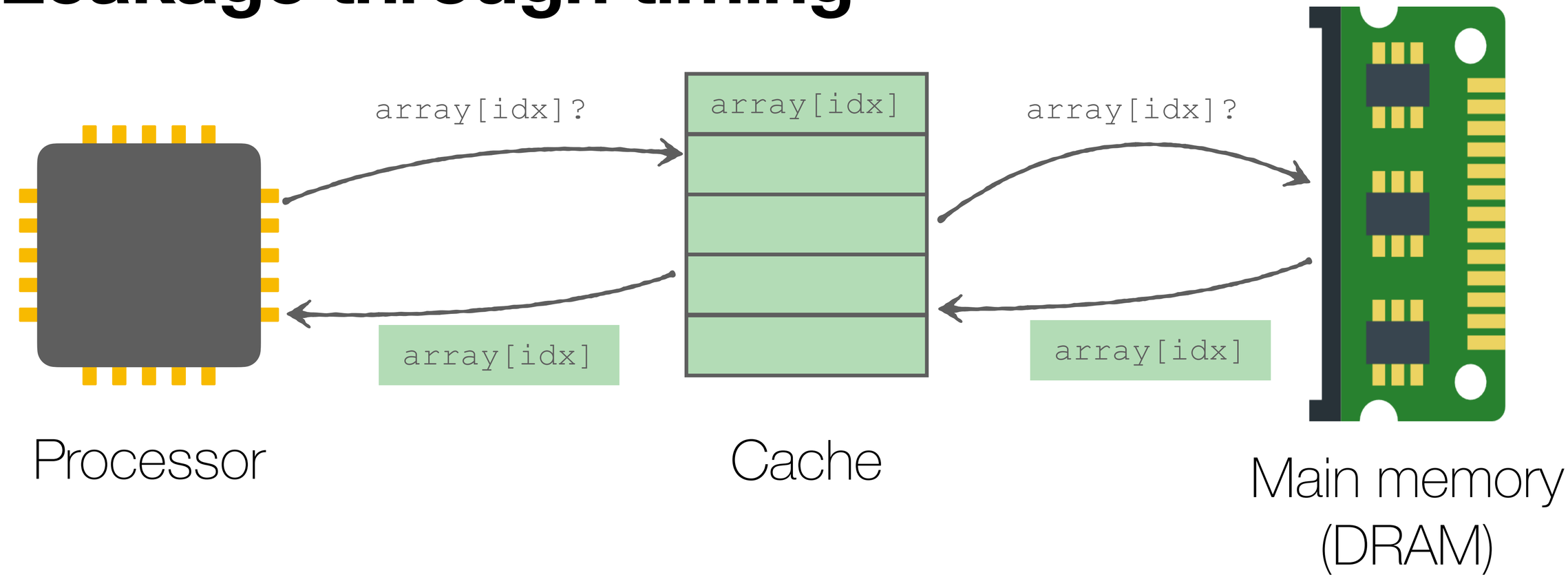
Cache hierarchy can be ***inclusive***, ***exclusive***, or ***neither***



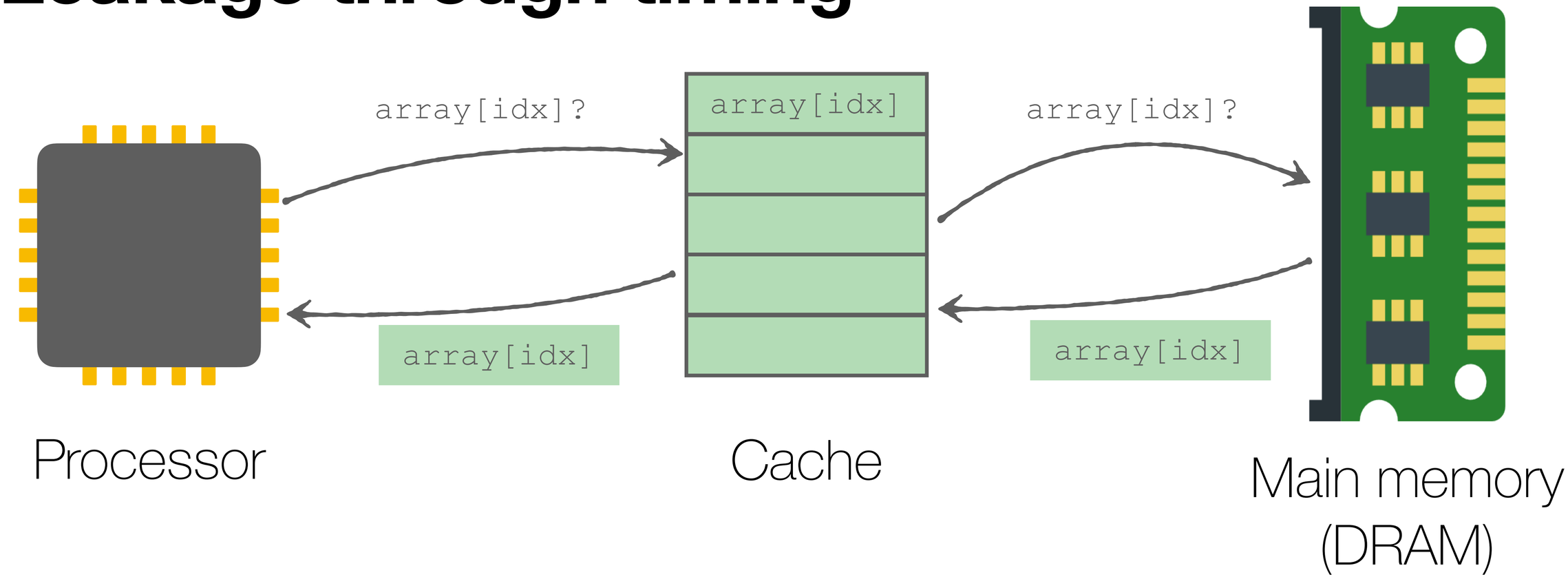
Picture of Intel “Haswell-E” Eight Core CPU

Caches and side channels

Leakage through timing

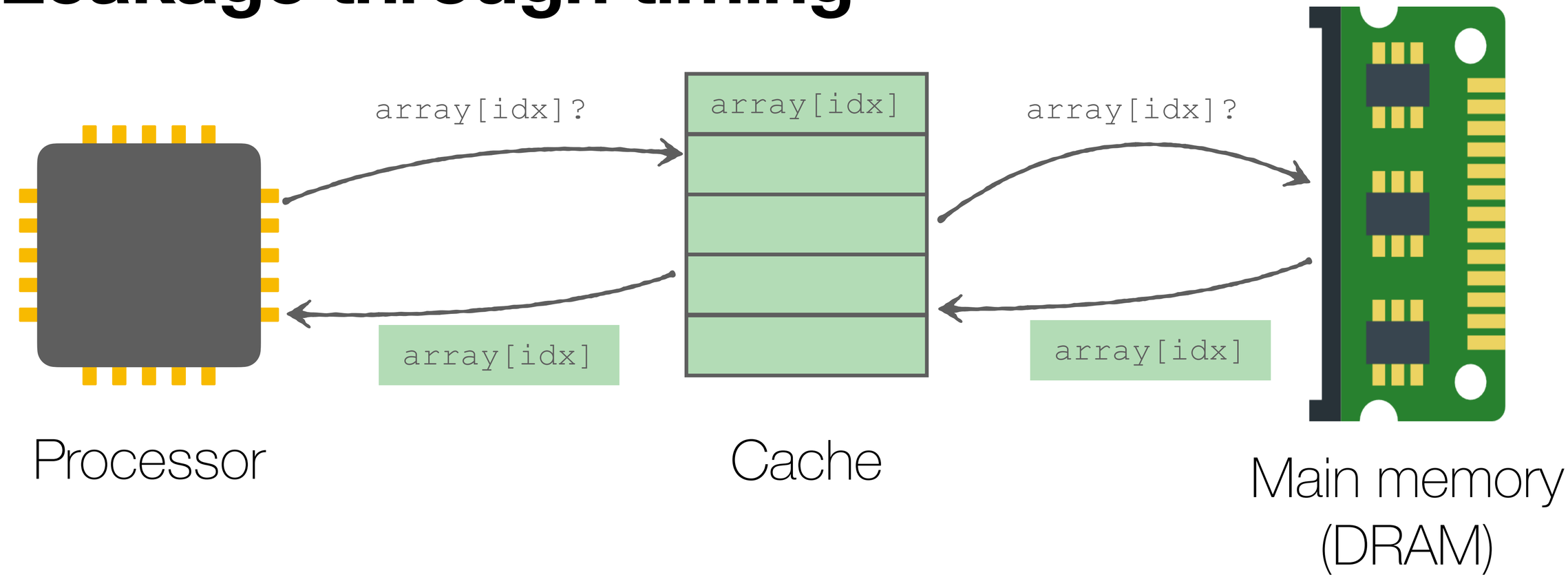


Leakage through timing



Latency of memory operations ***leaks*** information about ***cache state***

Leakage through timing

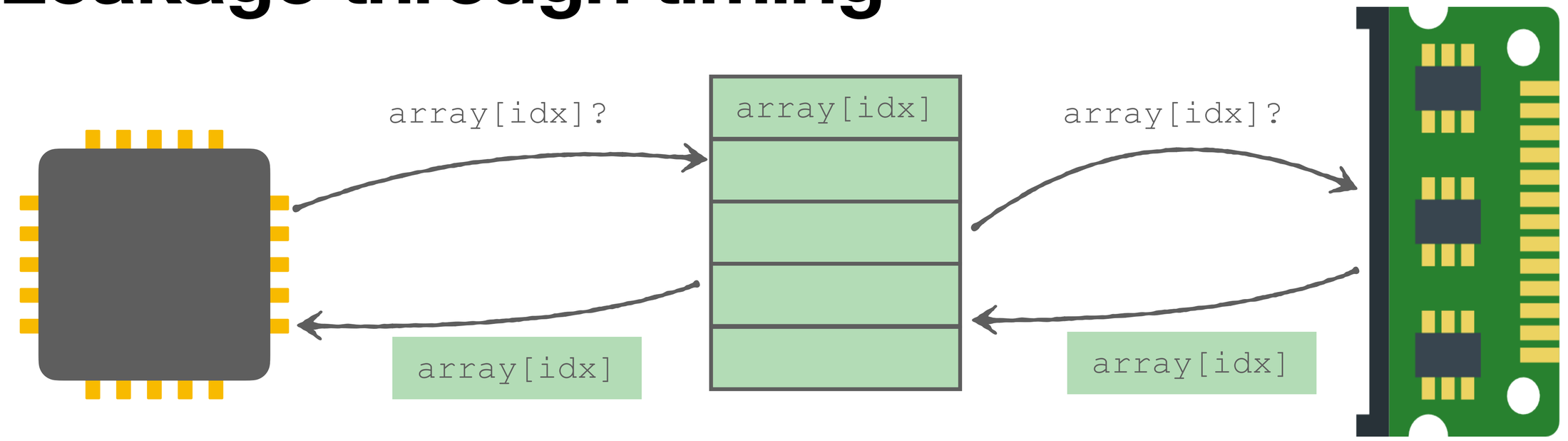


Latency of memory operations ***leaks*** information about ***cache state***

Cache hit —> block already ***in*** cache

Cache miss —> block ***not in*** cache

Leakage through timing



Latency of memory operations ***leaks*** information about ***cache state***

Cache hit —> block already ***in*** cache
Cache miss —> block ***not in*** cache

Cache state depends on ***prior*** memory ***accesses***

Leakage through timing

Leakage through timing

Goal:
Building primitive for
observing victim's
memory accesses

Leakage through timing

Goal:

Building primitive for ***observing*** victim's ***memory accesses***

Why?

In many cryptographic implementations, ***memory accesses*** leak information about the ***key***

Leakage through timing

Goal:
Building primitive for ***observing*** victim's ***memory accesses***

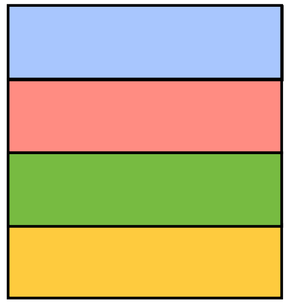
Why?

In many cryptographic implementations, ***memory accesses*** leak information about the ***key***

- Key-dependent memory accesses (e.g., lookup tables in AES or DES)
- Key-dependent control-flow (e.g., Square-and-Multiply in RSA)

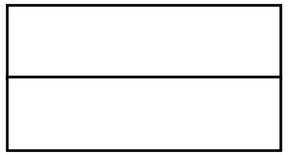
Ex: Cache-based Side Channel

Main memory



...

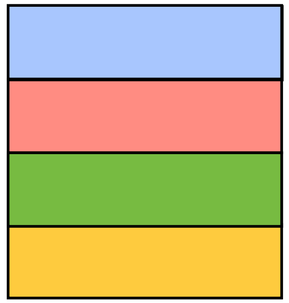
Cache



CPU

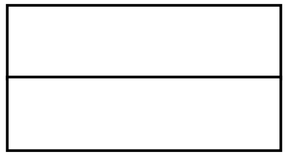
Ex: Cache-based Side Channel

Main memory



...

Cache

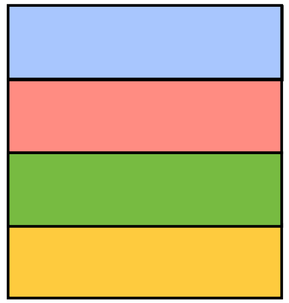


CPU

```
b = A[i];
c = A[j];
```

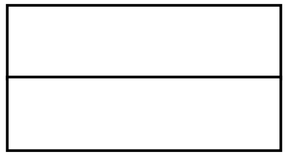
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

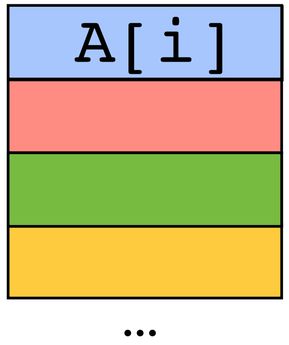
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

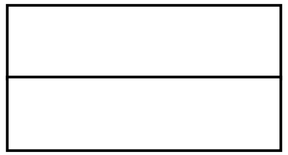
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

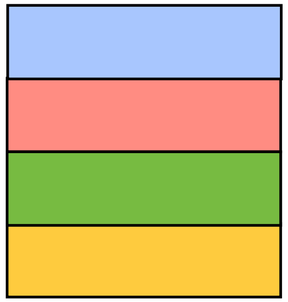
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

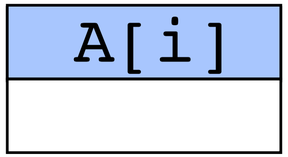
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

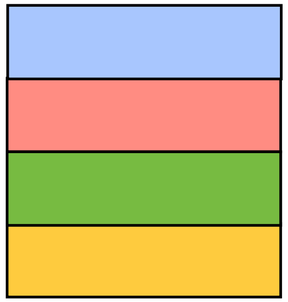
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

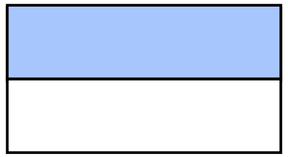
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

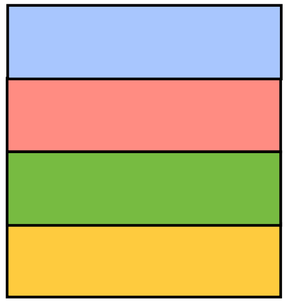
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

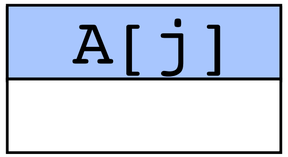
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

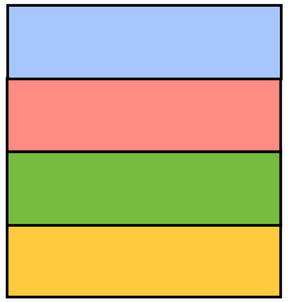
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

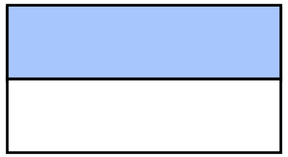
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

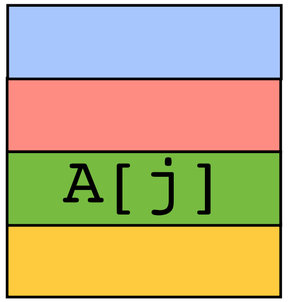
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

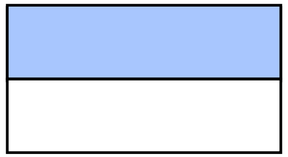
Ex: Cache-based Side Channel

Main memory



...

Cache



CPU

```
b = A[i];
c = A[j];
```

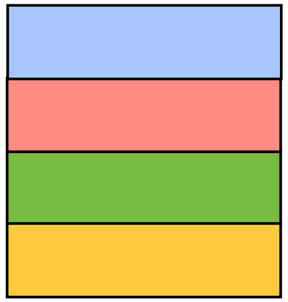
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

Ex: Cache-based Side Channel

Main memory



...

Cache

CPU

```
b = A[i];
c = A[j];
```

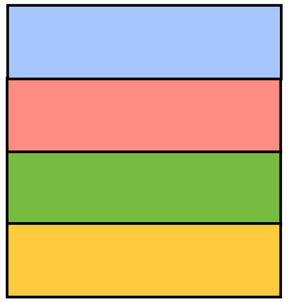
Accesses:

baseA + elementSize * ***i***

baseA + elementSize * ***j***

Ex: Cache-based Side Channel

Main memory



...

Cache

CPU

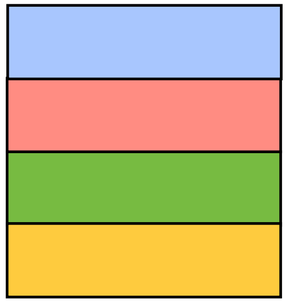
Accesses:
baseA + elementSize * ***i***
baseA + elementSize * ***j***

```
b = A[i];
c = A[j];
```

Latency of lookups ***leaks relation*** between indices

Ex: Cache-based Side Channel

Main memory



...

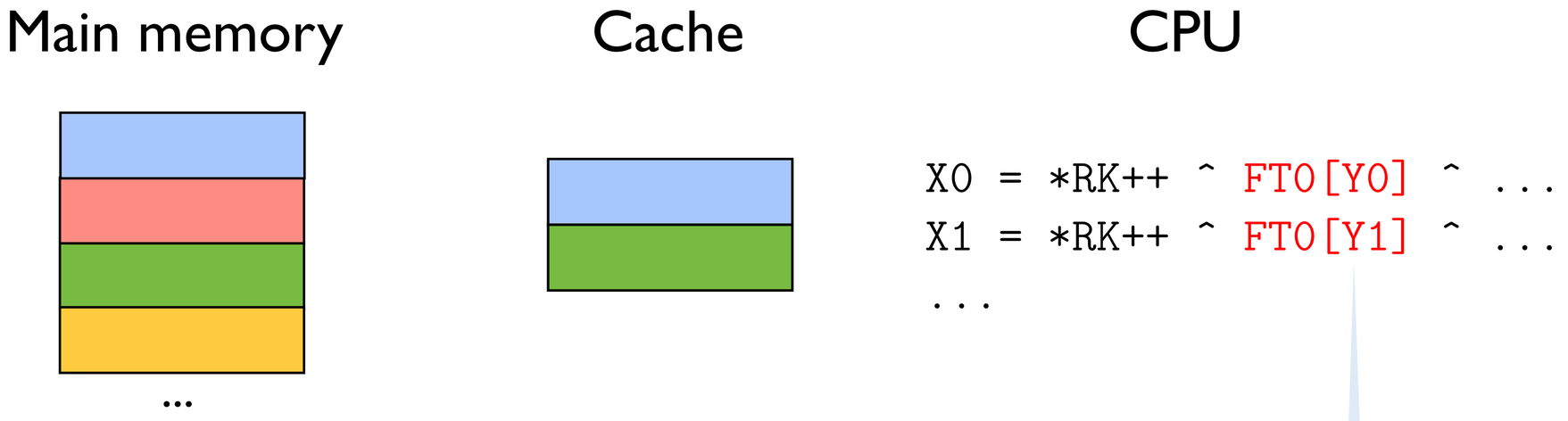
Cache

CPU

```
X0 = *RK++ ^ FT0[Y0] ^ ...
X1 = *RK++ ^ FT0[Y1] ^ ...
...
```

Code from PolarSSL AES implementation using T-tables

Ex: Cache-based Side Channel



- First attacks: 2005 (Bernstein/Shamir)
- Increasingly effective attacks: 2013… (Yarom/…)

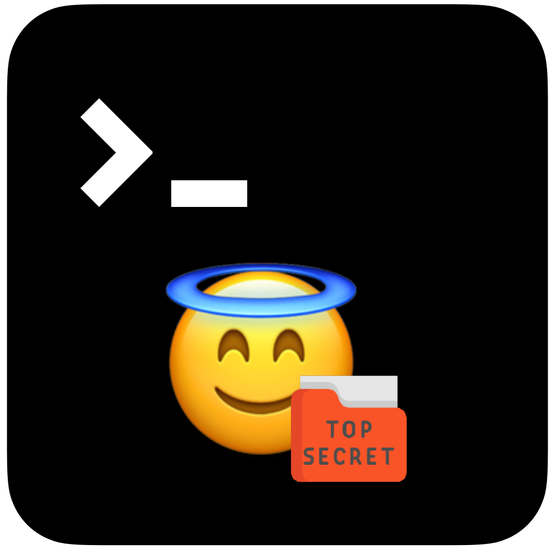
Two classes of attacks

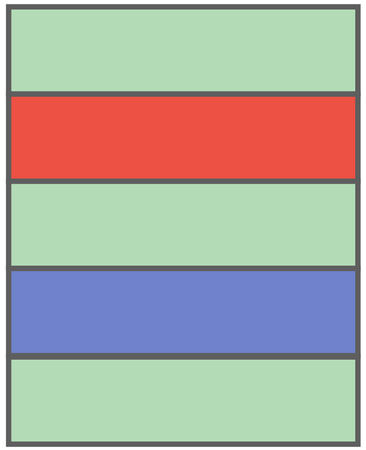
Two classes of attacks

Attacks that ***infer memory accesses*** from ***victim's execution time***

Attacks that ***infer memory accesses*** from ***cache state***

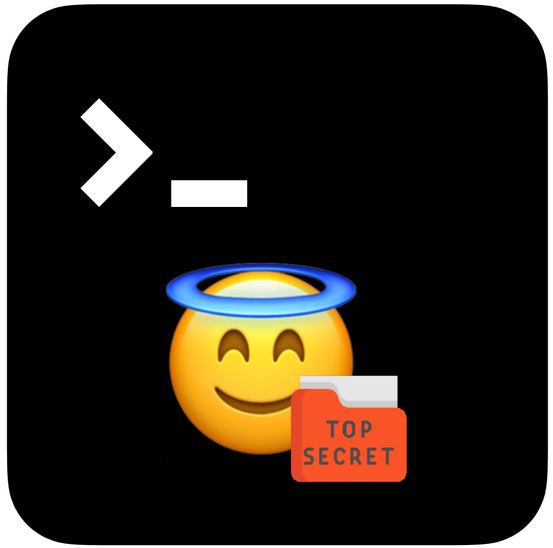
Inferring memory accesses from exec. time

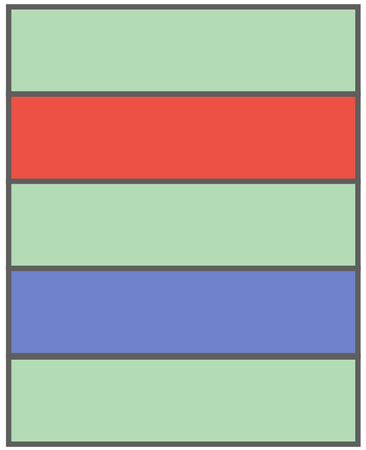




Cache

Inferring memory accesses from exec. time



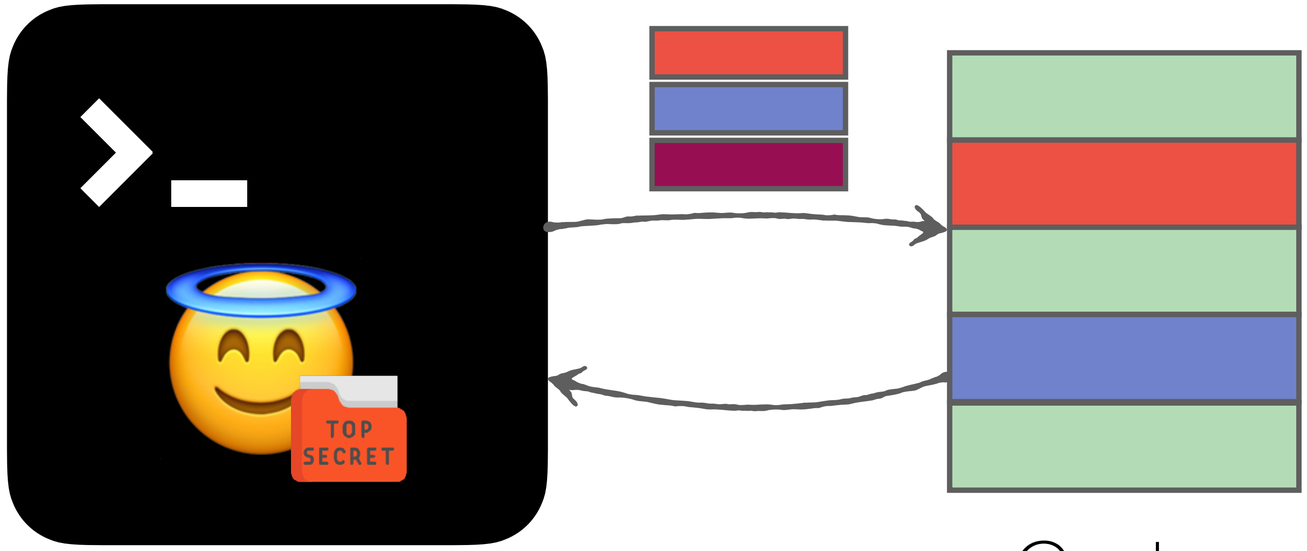


Cache

Victim's ***execution time*** depends on ***initial cache state*** and performed ***memory accesses***

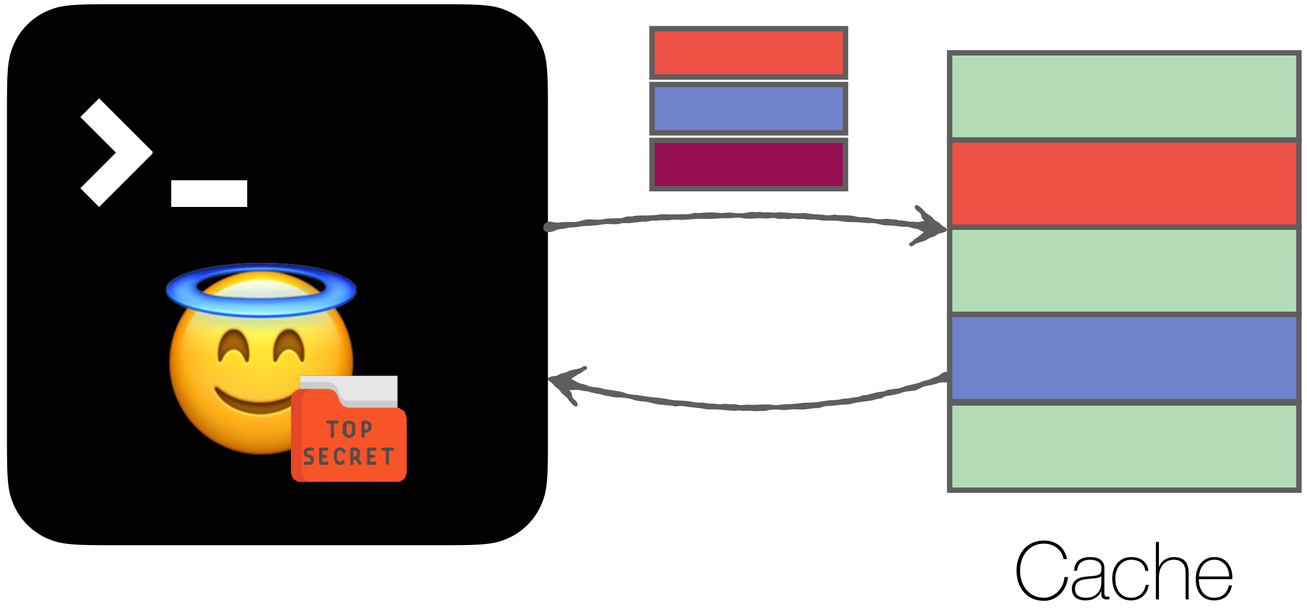
Inferring memory accesses from exec. time

Victim's ***execution time*** depends on ***initial cache state*** and performed ***memory accesses***



Cache

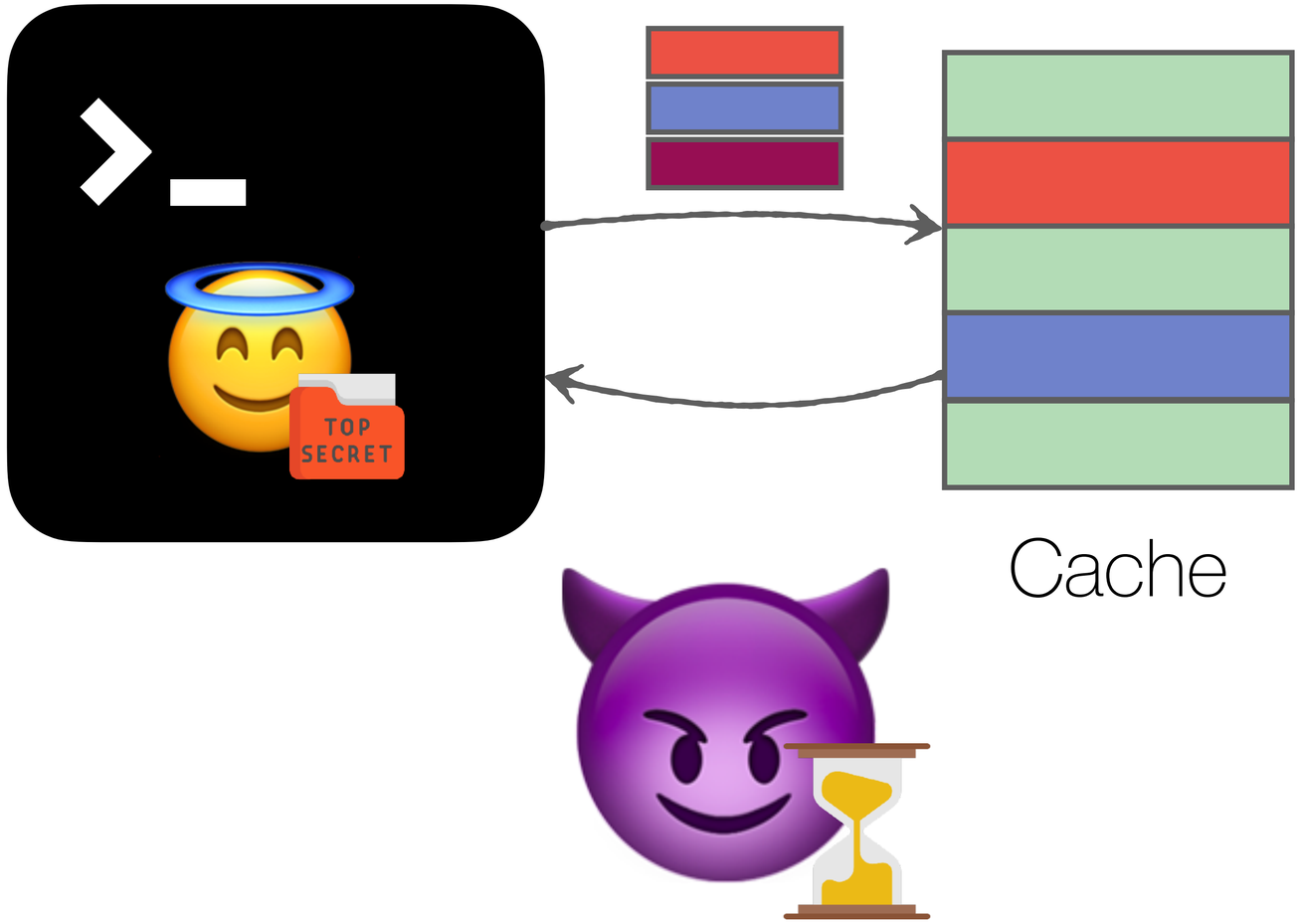
Inferring memory accesses from exec. time



Victim's ***execution time*** depends on ***initial cache state*** and performed ***memory accesses***

Attacker measures ***victim's execution time*** and infers information about ***memory accesses***

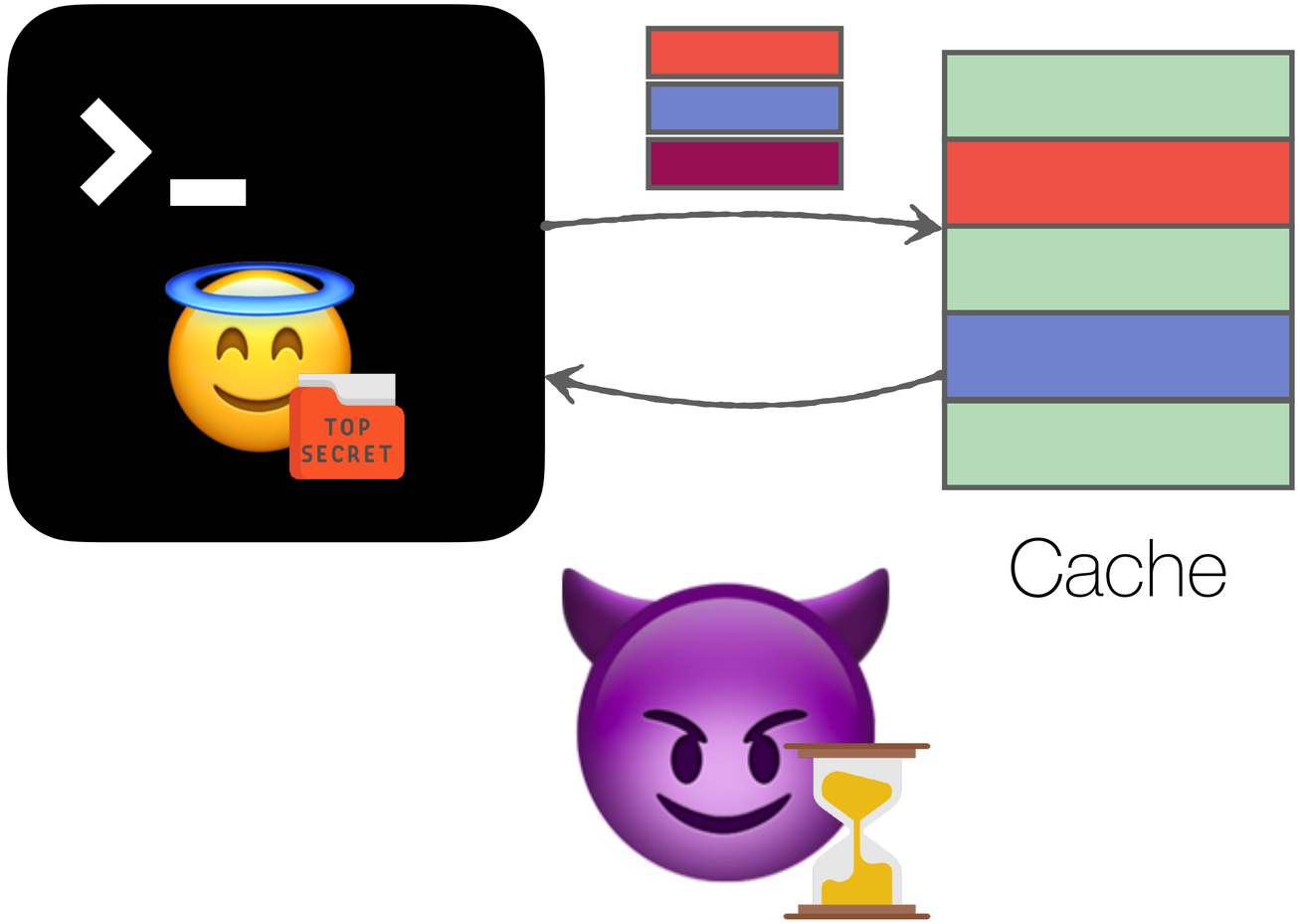
Inferring memory accesses from exec. time



Victim's ***execution time*** depends on ***initial cache state*** and performed ***memory accesses***

Attacker measures ***victim's execution time*** and infers information about ***memory accesses***

Inferring memory accesses from exec. time

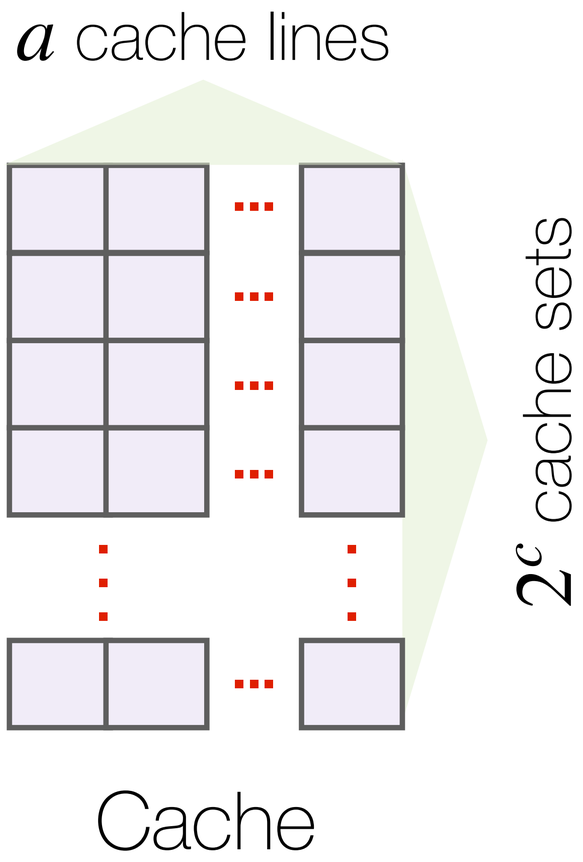


Victim's ***execution time*** depends on ***initial cache state*** and performed ***memory accesses***

Attacker measures ***victim's execution time*** and infers information about ***memory accesses***

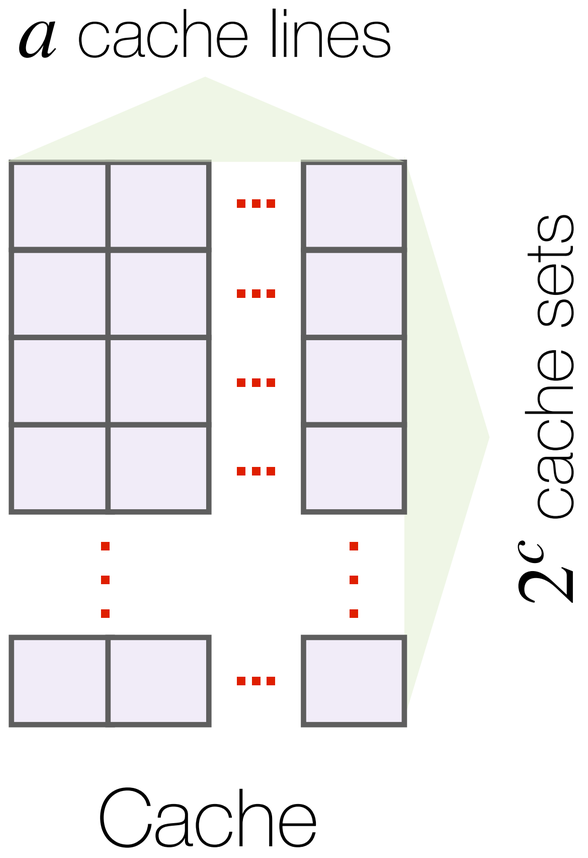
Noisy measurements

Evict+Time



D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

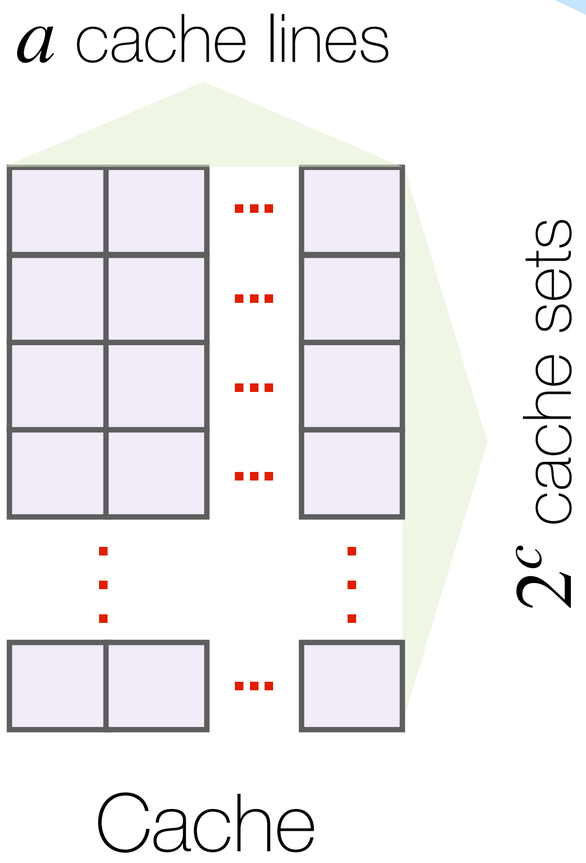


1. ***Victim*** runs

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

Attacker controls when victim starts

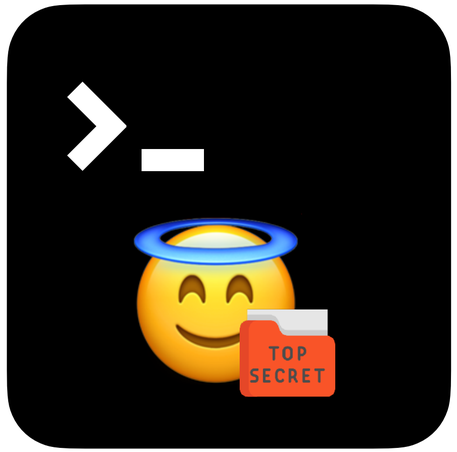


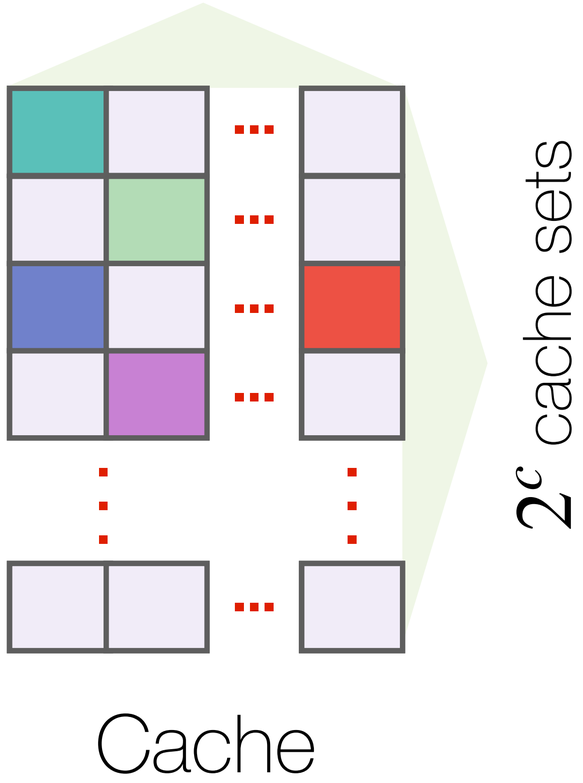
1. ***Victim*** runs

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

Attacker controls when victim starts



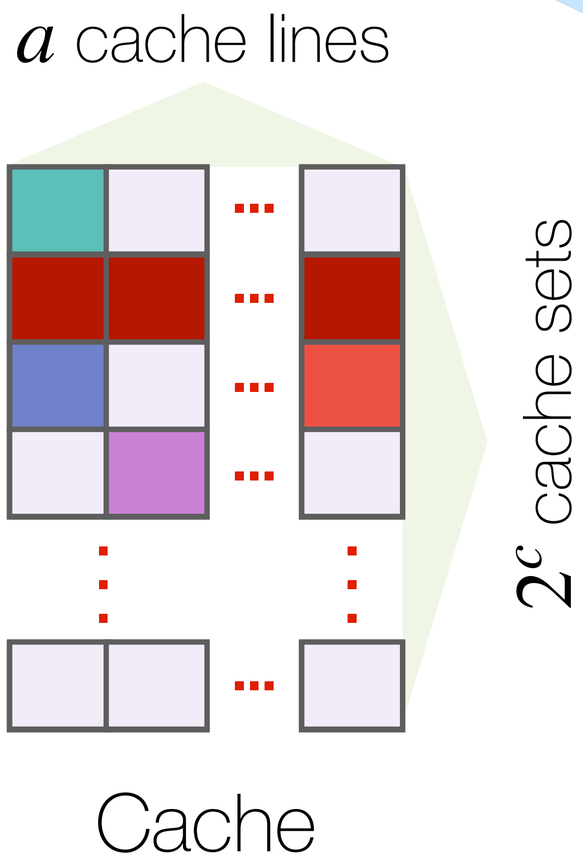


1. ***Victim*** runs

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

Attacker controls when victim starts



1. ***Victim*** runs

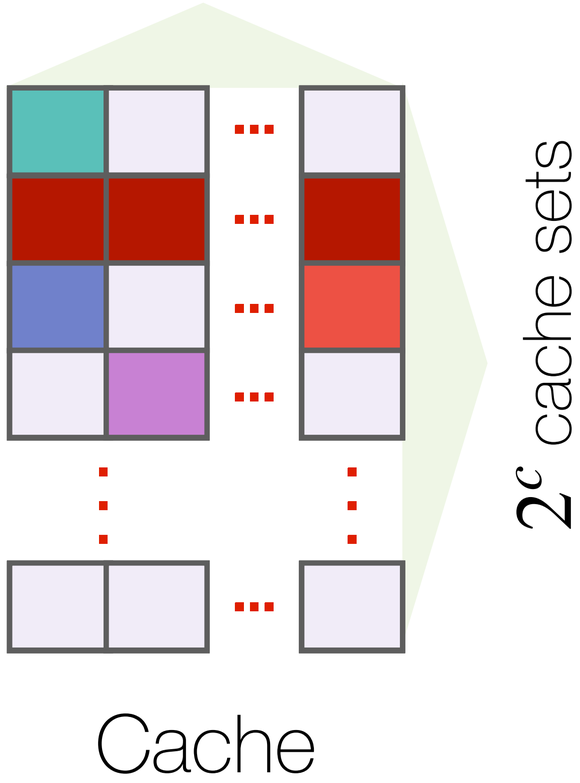
2. ***Evict:*** Attacker ***removes*** victim's data from cache set

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

Attacker controls when victim starts

Eviction set needed: more than a blocks targeting the cache set



1. ***Victim*** runs

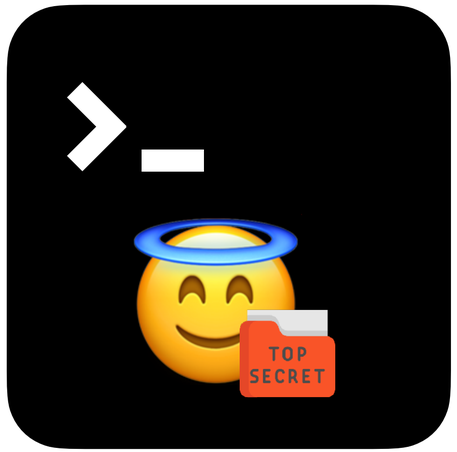
2. ***Evict:*** Attacker ***removes*** victim's data from cache set

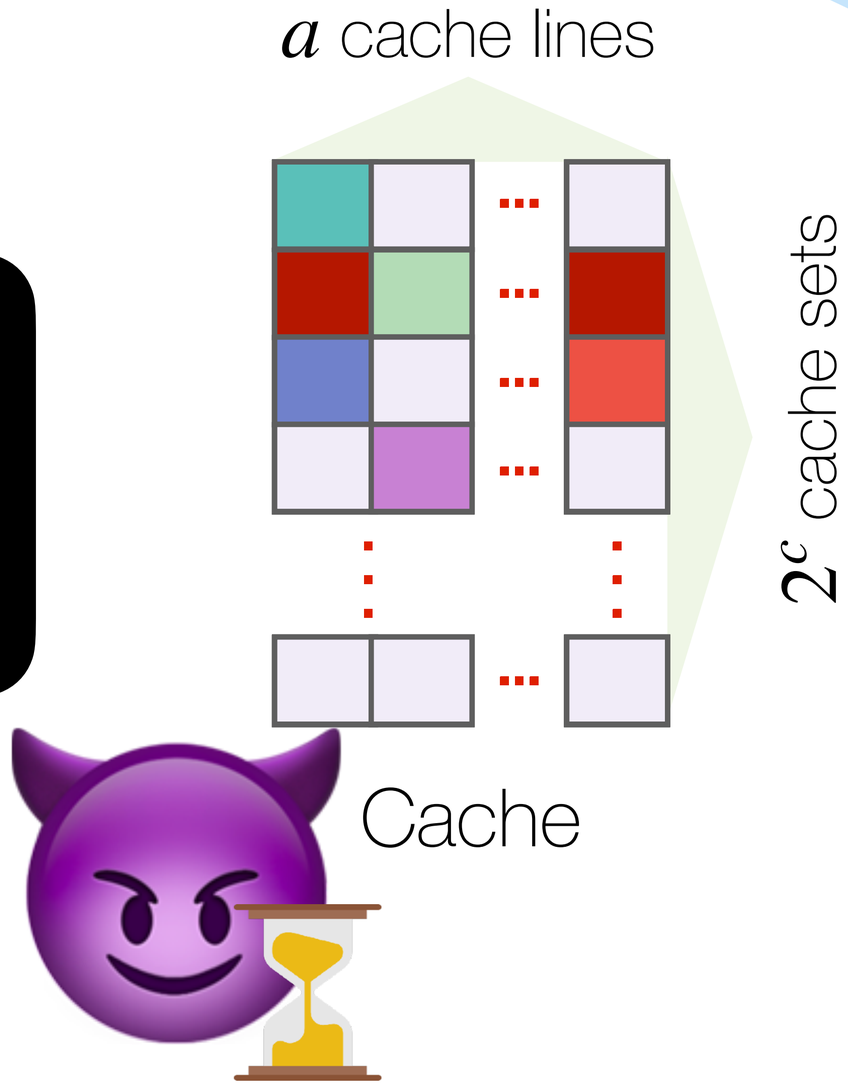
D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

Attacker controls when victim starts

Eviction set needed: more than a blocks targeting the cache set





1. ***Victim*** runs

2. ***Evict:*** Attacker ***removes*** victim's data from cache set

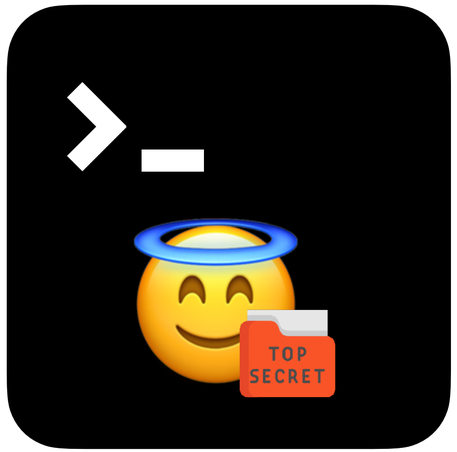
3. ***Time: Victim*** runs and ***attacker*** measures execution time

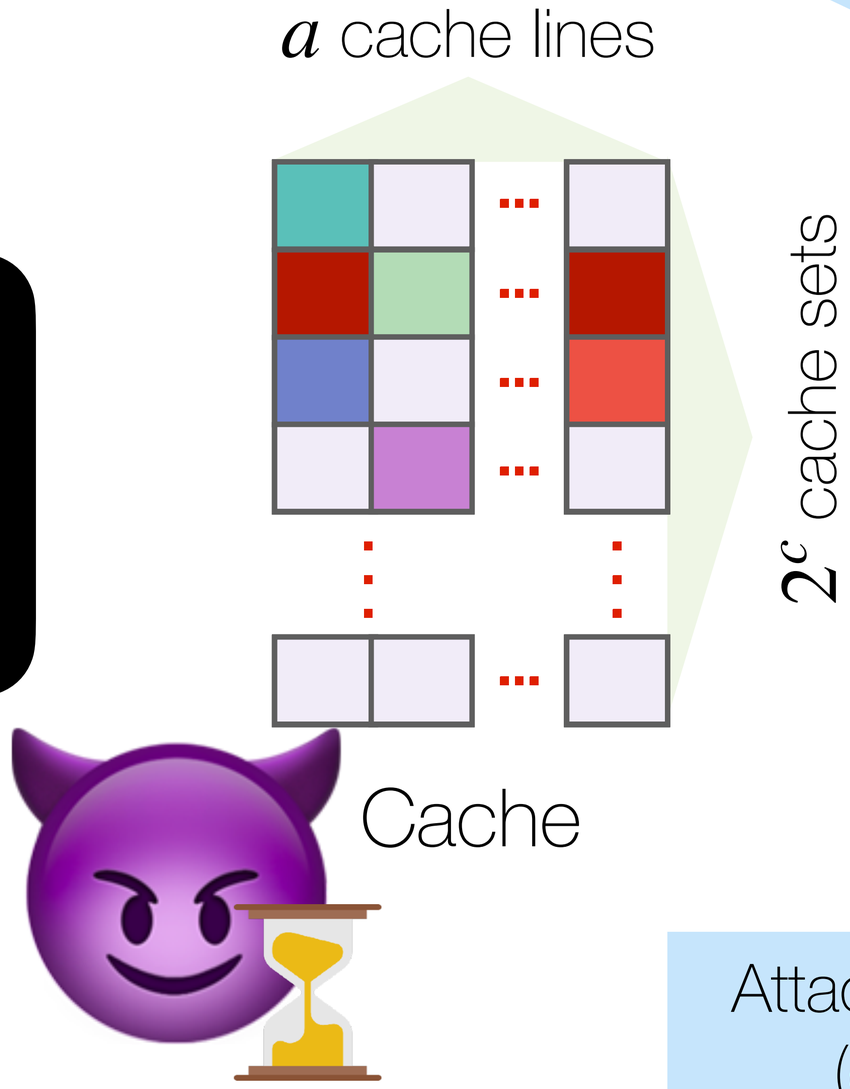
D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Evict+Time

Attacker controls when victim starts

Eviction set needed: more than a blocks targeting the cache set





1. ***Victim*** runs

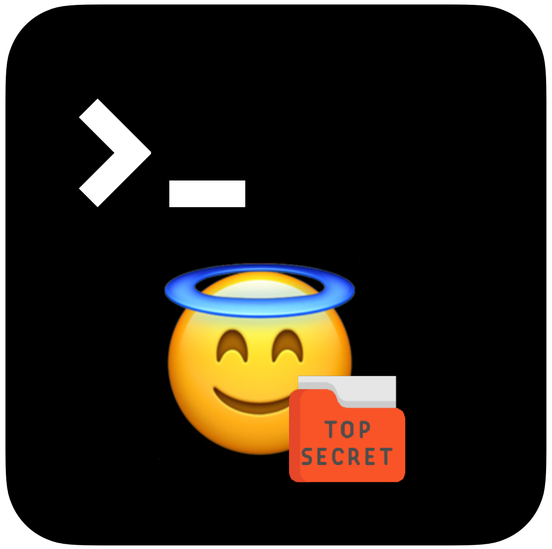
2. ***Evict:*** Attacker ***removes*** victim's data from cache set

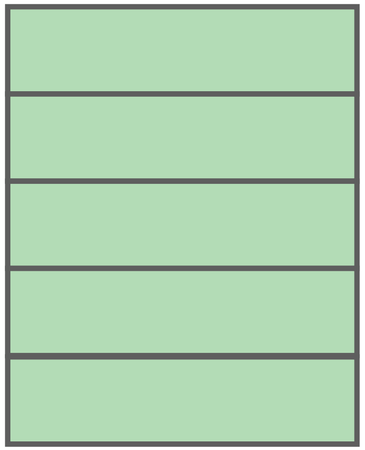
3. ***Time: Victim*** runs and ***attacker*** measures execution time

Attacker learns if ***cache miss*** happened (***cache set*** has been ***accessed***)

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

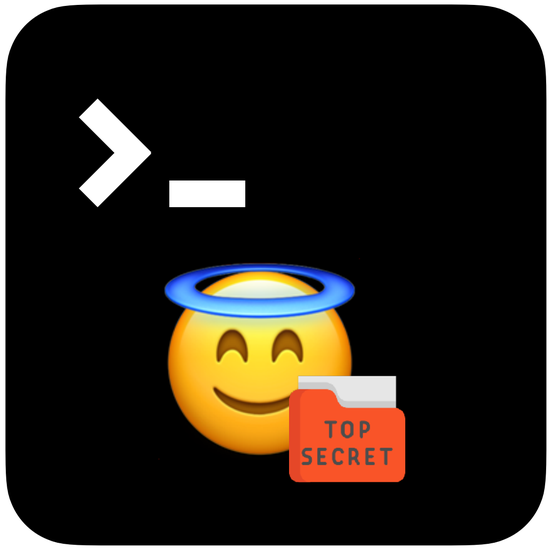
Inferring memory accesses from cache state

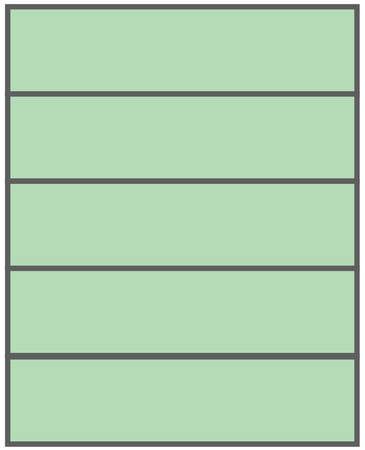




Cache

Inferring memory accesses from cache state

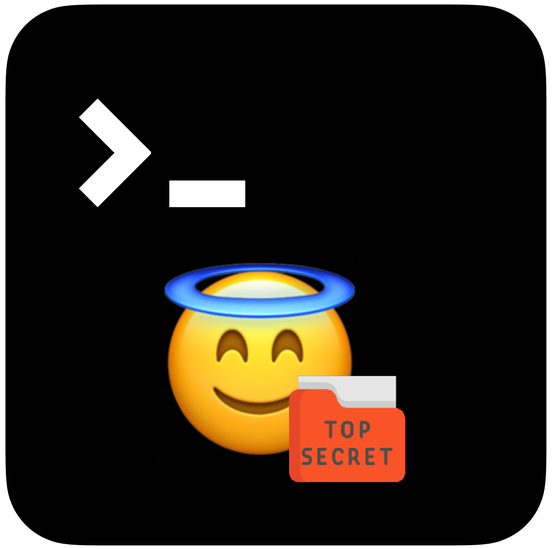


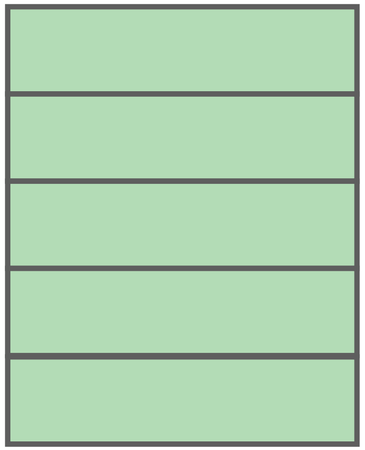


Cache

Attacker and victim ***share*** cache

Inferring memory accesses from cache state



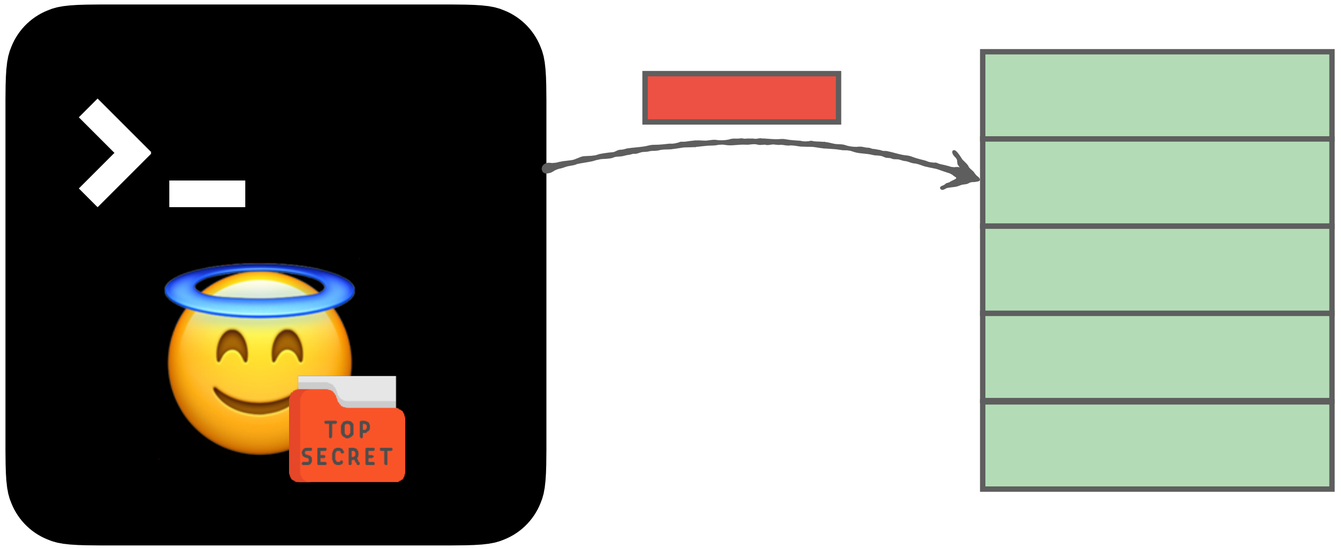


Cache

Executing a program results in ***loading memory blocks*** to cache

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

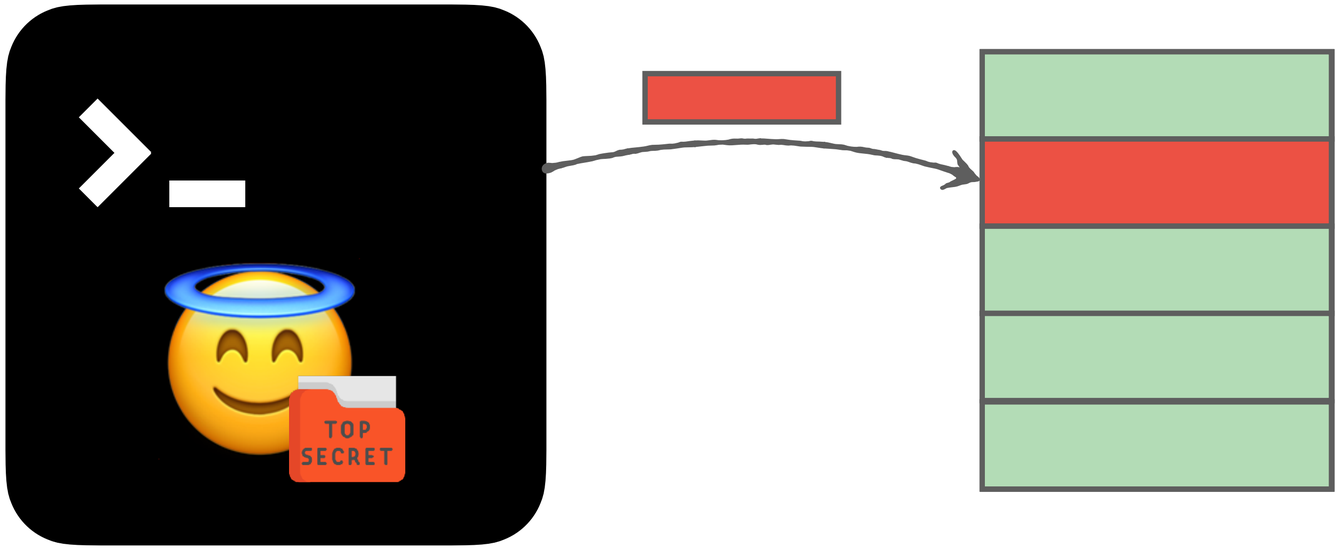


Cache

Executing a program results in ***loading memory blocks*** to cache

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

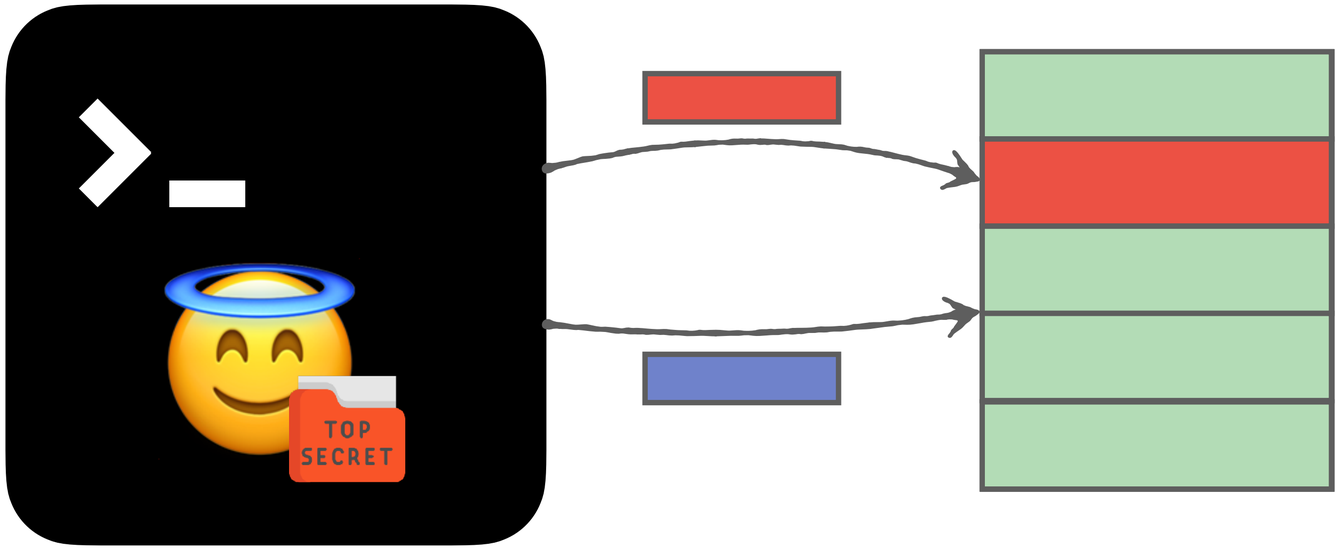


Cache

Executing a program results in ***loading memory blocks*** to cache

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

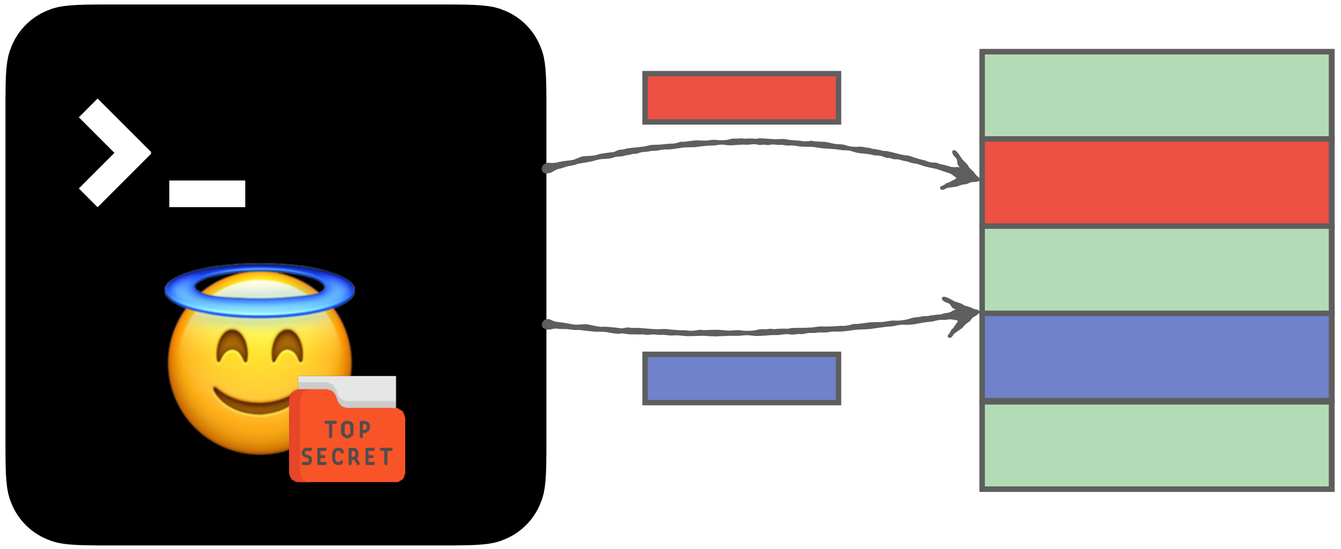


Cache

Executing a program results in ***loading memory blocks*** to cache

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

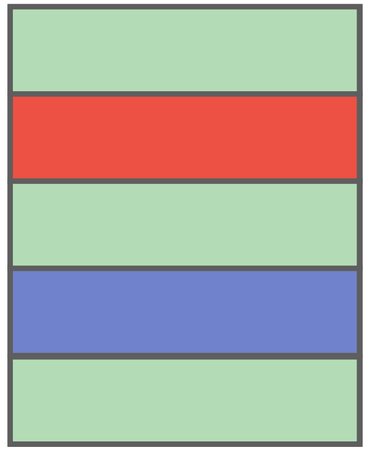


Cache

Executing a program results in ***loading memory blocks*** to cache

Attacker and victim ***share*** cache

Inferring memory accesses from cache state



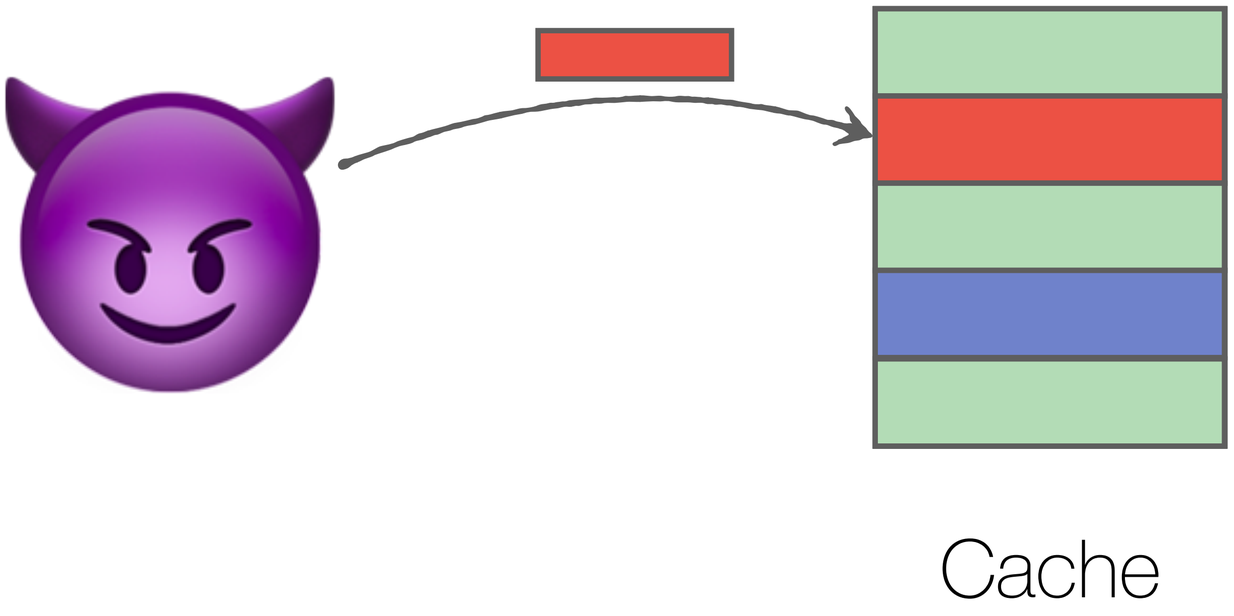
Cache

Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

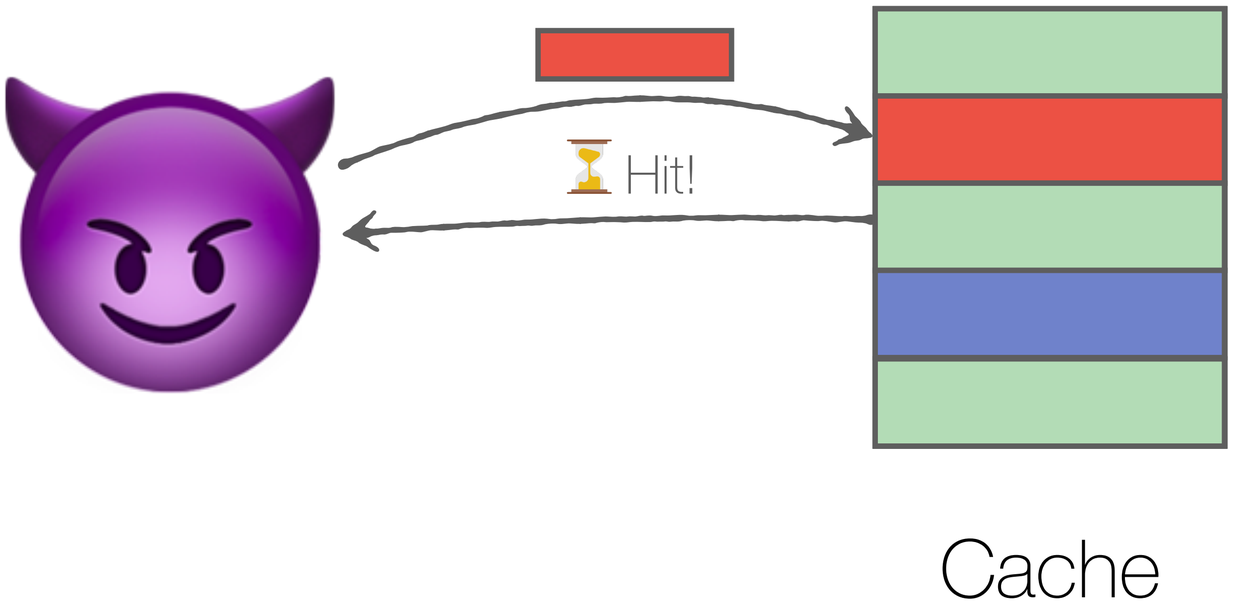


Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

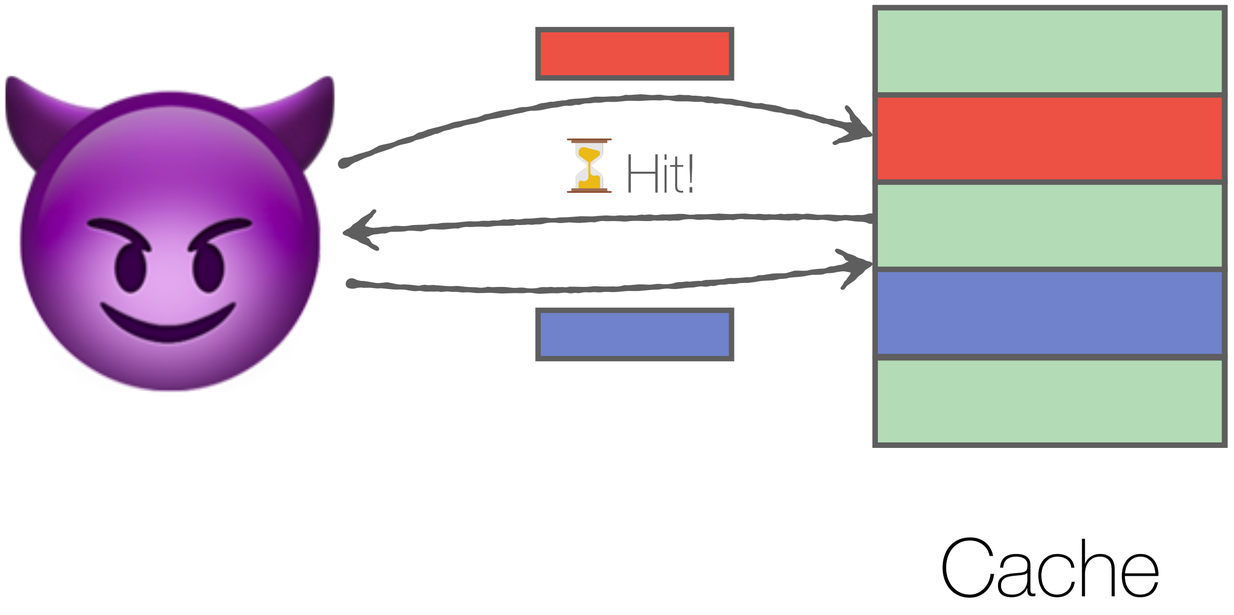


Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

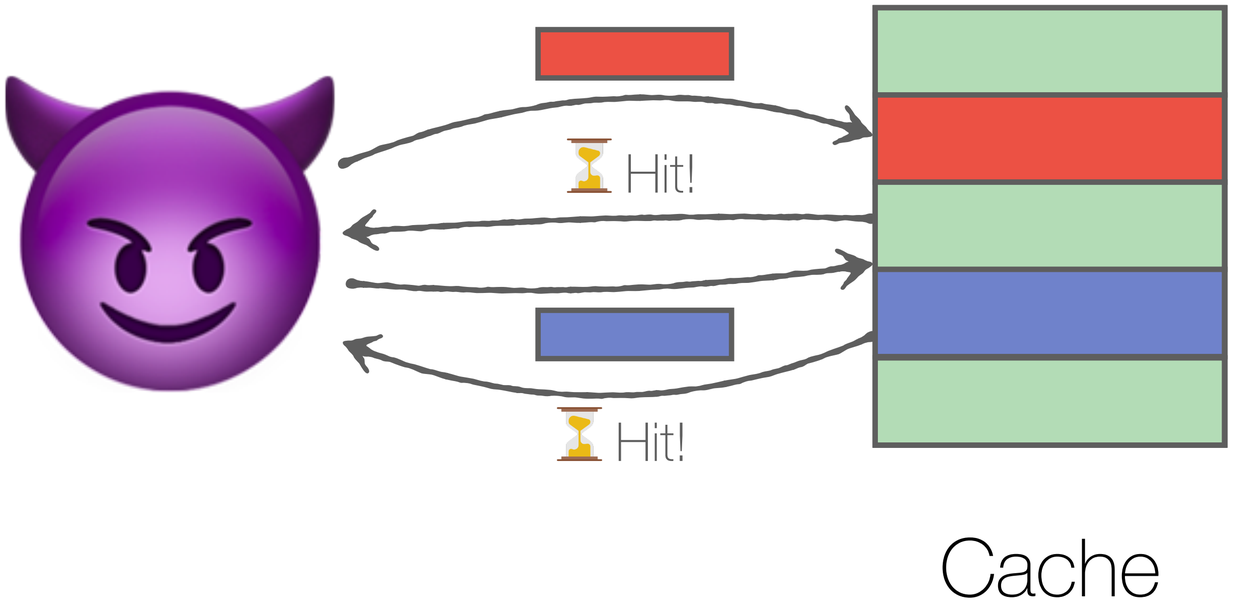


Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

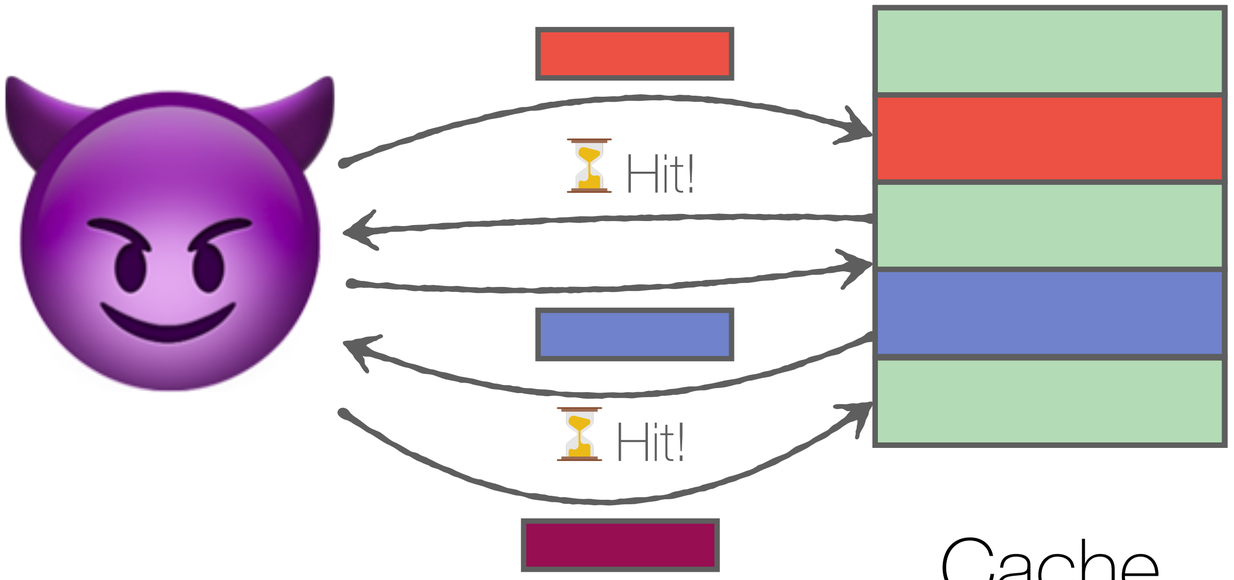


Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

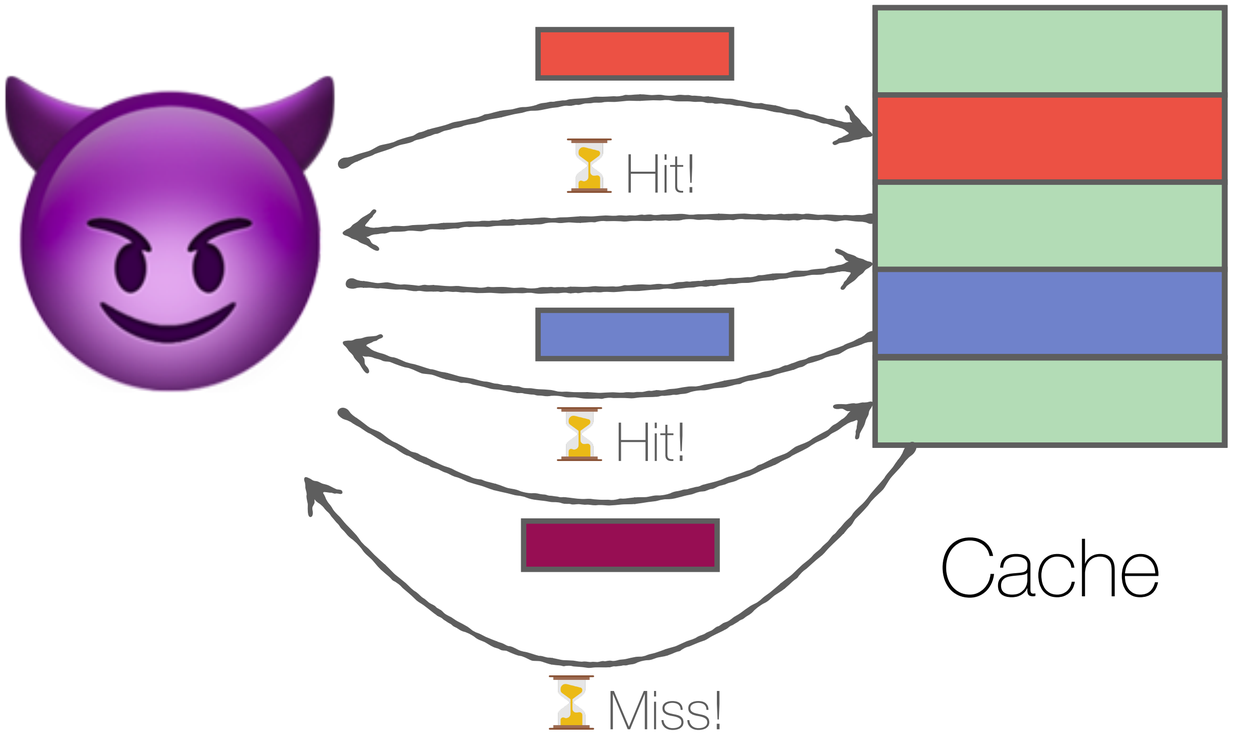


Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

Attacker and victim ***share*** cache

Inferring memory accesses from cache state

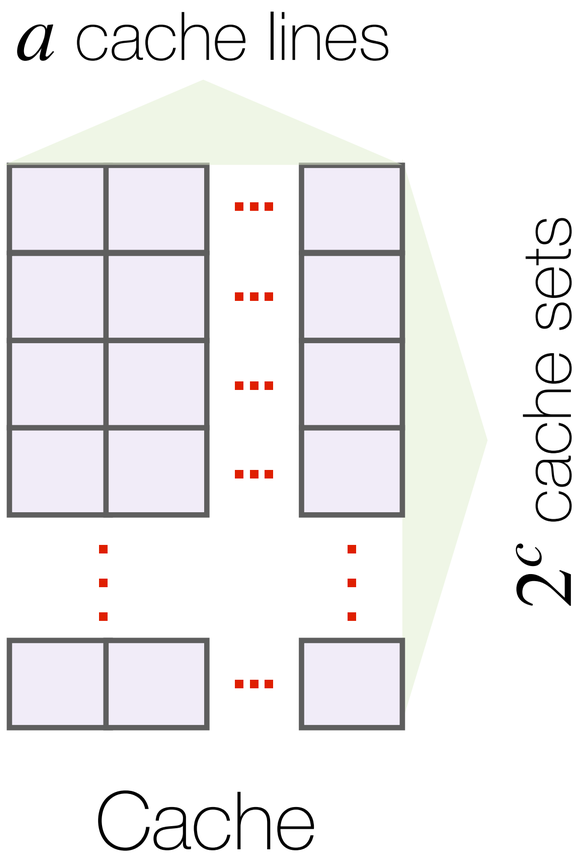


Executing a program results in ***loading memory blocks*** to cache

Attacker can probe the cache to infer ***which blocks*** are ***in*** the ***cache***

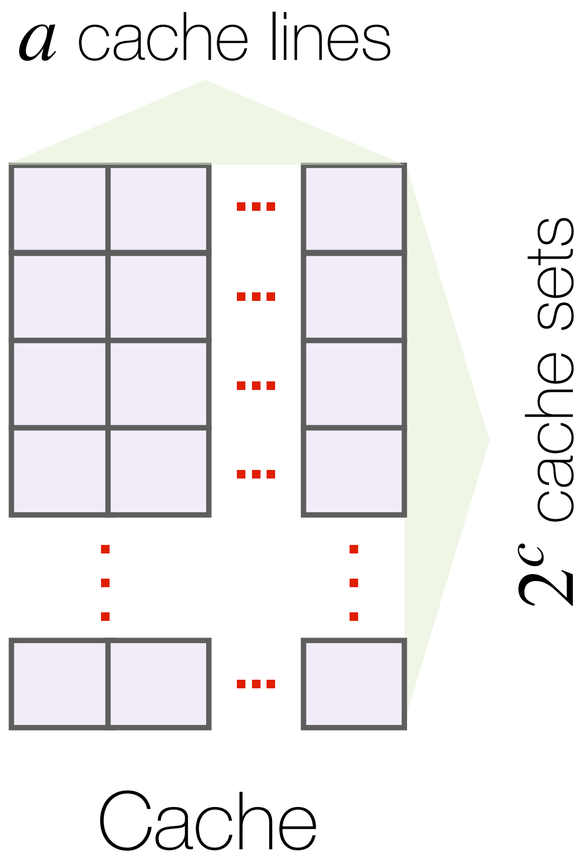
Attacker and victim ***share*** cache

Attack 1: Prime + Probe



D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

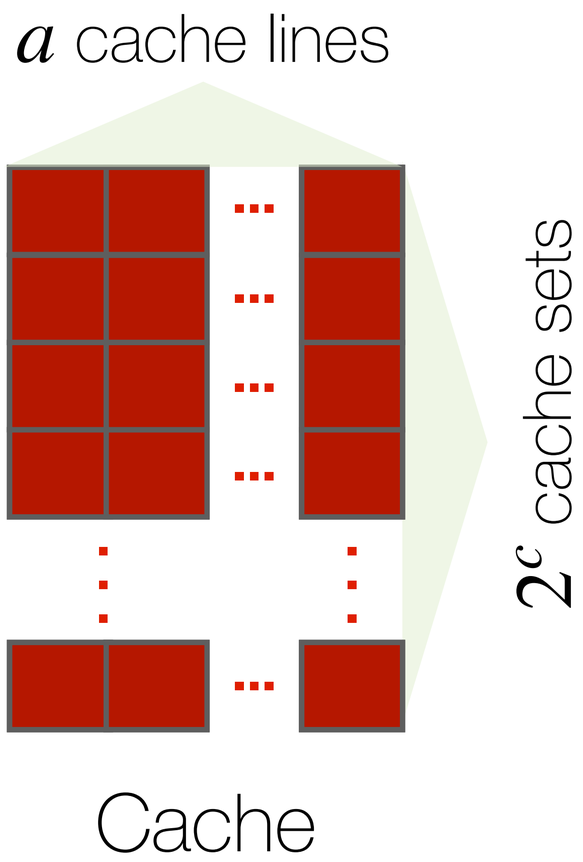
Attack 1: Prime + Probe



1. ***Prime:*** Fill cache with ***attacker's data***

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 1: Prime + Probe

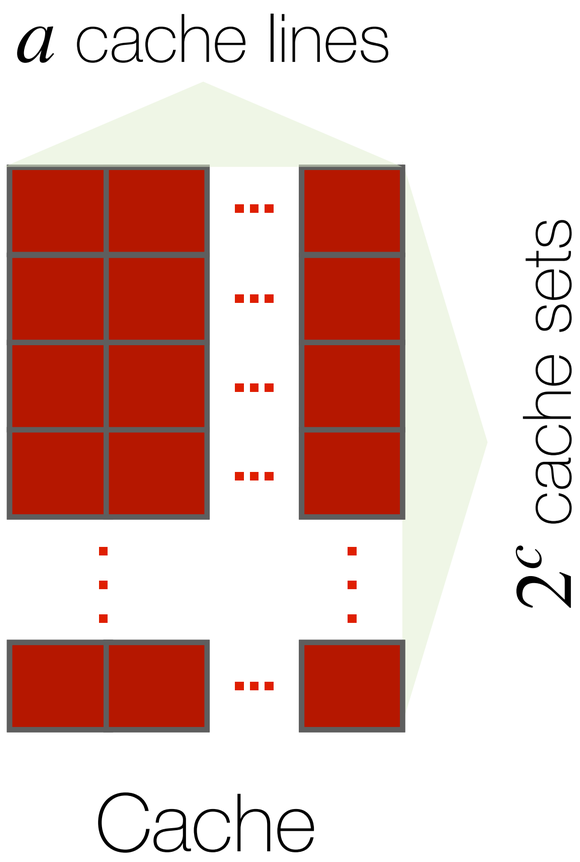


1. ***Prime:*** Fill cache with ***attacker's data***

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 1: Prime + Probe

Eviction sets needed:
more than a blocks for each cache set

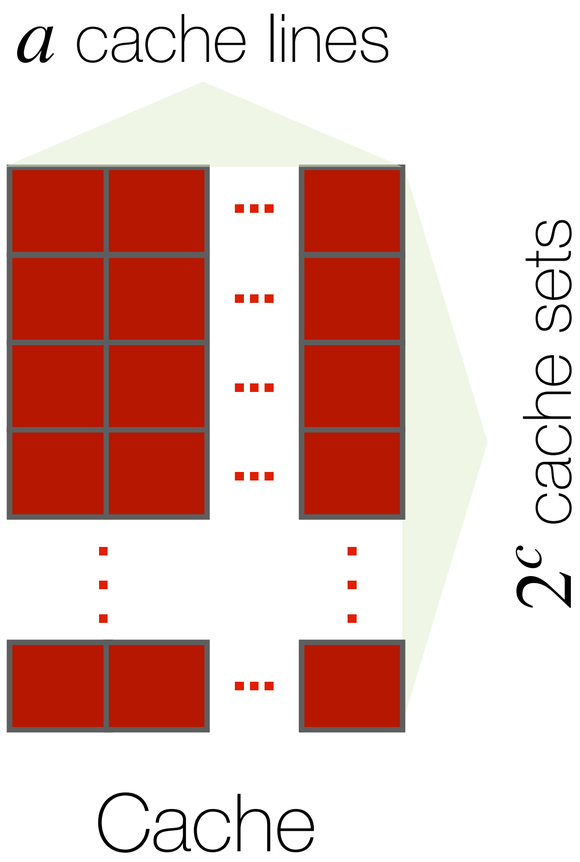


1. ***Prime:*** Fill cache with ***attacker's data***

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 1: Prime + Probe

Eviction sets needed:
more than a blocks for each cache set



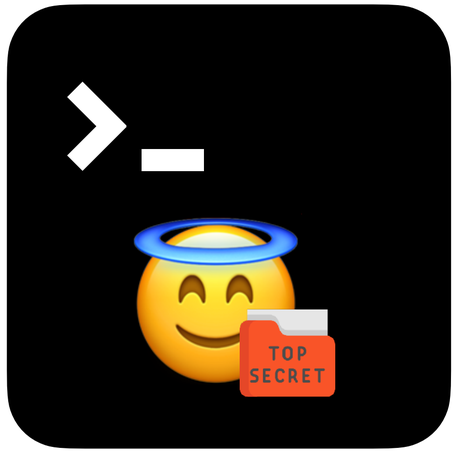
1. ***Prime:*** Fill cache with ***attacker's data***

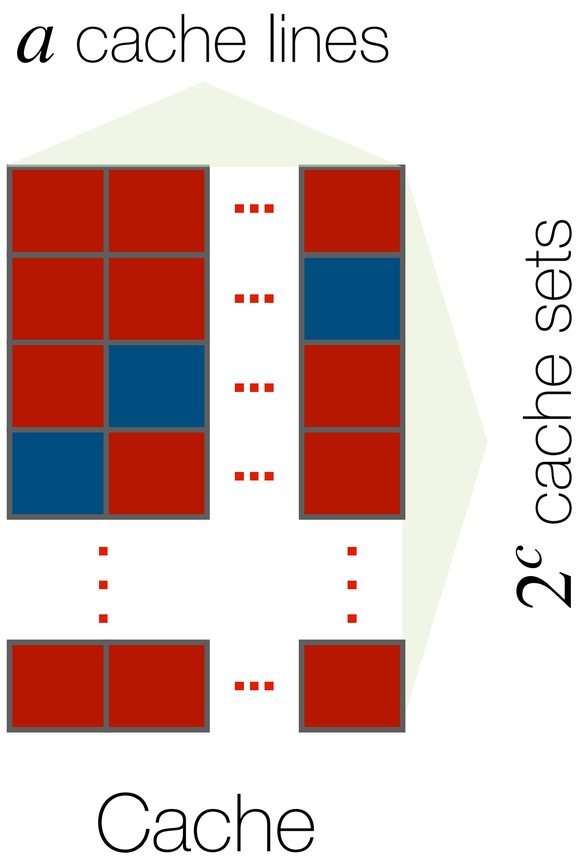
2. ***Victim*** runs

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 1: Prime + Probe

Eviction sets needed:
more than a blocks for each cache set





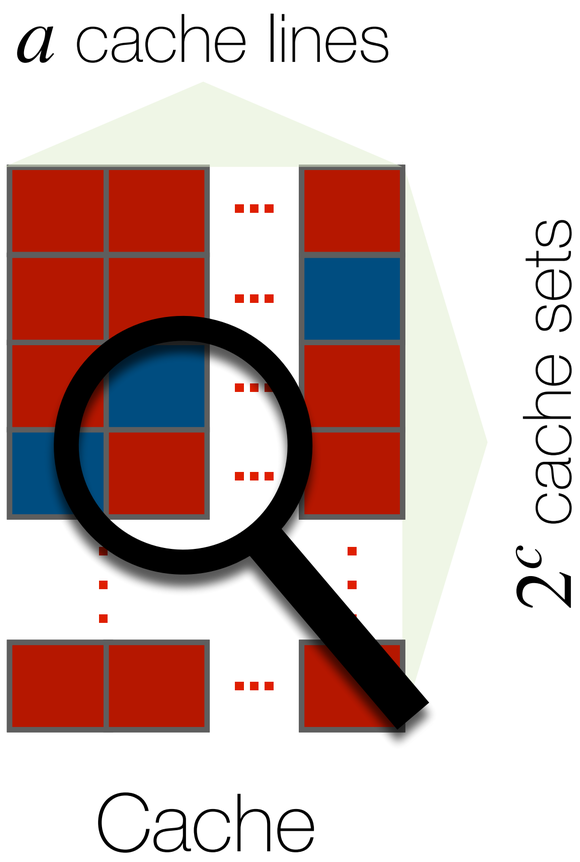
1. ***Prime:*** Fill cache with ***attacker's data***

2. ***Victim*** runs

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 1: Prime + Probe

Eviction sets needed:
more than a blocks for each cache set



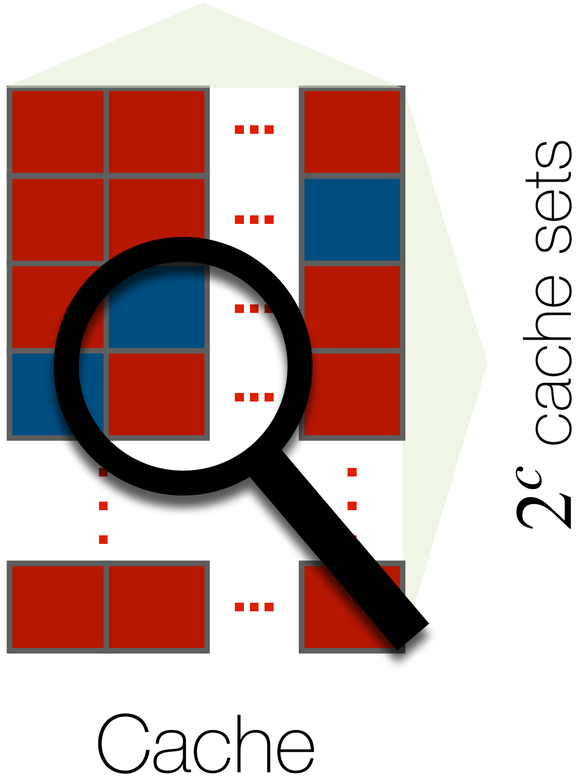
1. ***Prime:*** Fill cache with ***attacker's data***

2. ***Victim*** runs

3. ***Probe: Access again*** data and ***measure*** time

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 1: Prime + Probe



Eviction sets needed: more than a blocks for each cache set

1. ***Prime:*** Fill cache with ***attacker's data***

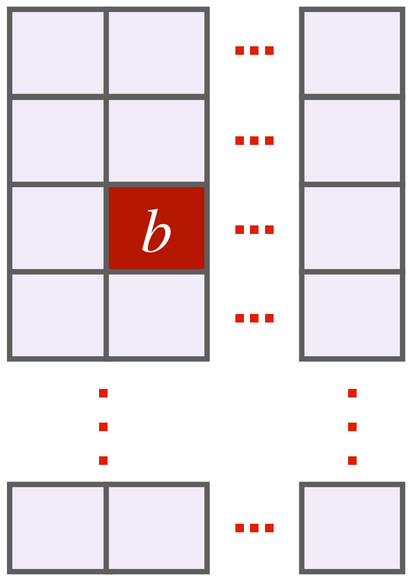
2. ***Victim*** runs

3. ***Probe: Access again*** data and ***measure*** time

Attacker learns which ***cache sets*** have been ***accessed by victim***

D. Osvik, A. Shamir, E. Tromer — Cache attacks and countermeasures: the case of AES— 2005

Attack 2: Flush + Reload

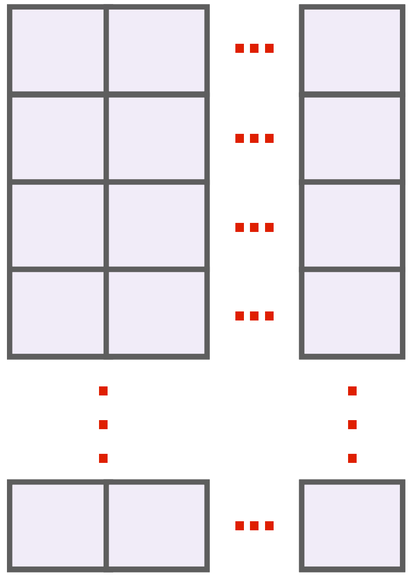


Cache

Attacker and ***victim*** share some memory block b

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack — Usenix Security 2014

Attack 2: Flush + Reload



Cache

1. ***Flush:*** Remove b from cache

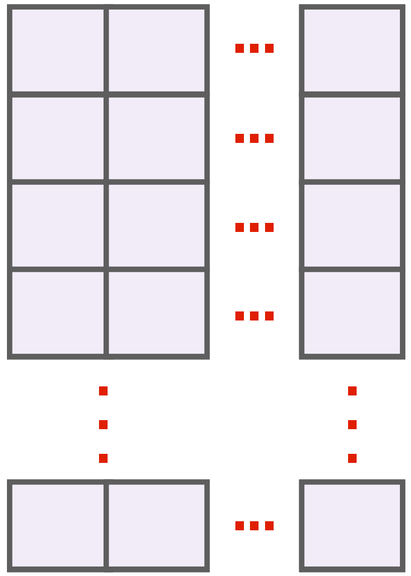
Attacker and ***victim*** share some memory block b

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack — Usenix Security 2014

Attack 2: Flush + Reload

Clflush on x86

1. ***Flush:*** Remove b from cache

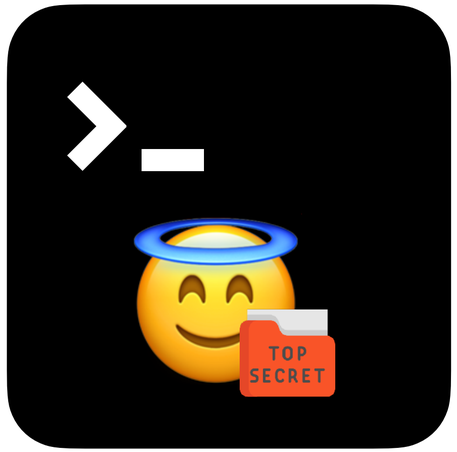


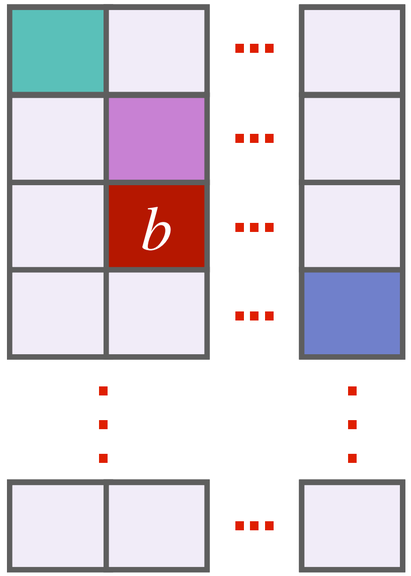
Cache

Attacker and ***victim*** share some memory block b

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack — Usenix Security 2014

Attack 2: Flush + Reload





Cache

Clflush on x86

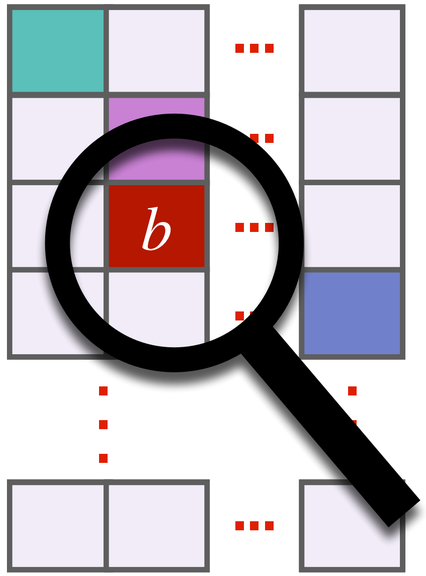
1. ***Flush:*** Remove b from cache

2. ***Victim*** runs

Attacker and ***victim*** share some memory block b

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack — Usenix Security 2014

Attack 2: Flush + Reload



Cache

Attacker and ***victim*** share some memory block b

Clflush on x86

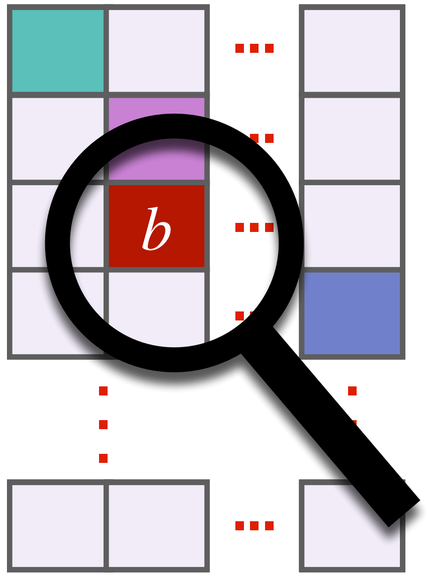
1. ***Flush:*** Remove b from cache

2. ***Victim*** runs

3. ***Reload: Access again*** b and ***measure*** time

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack — Usenix Security 2014

Attack 2: Flush + Reload



Cache

Attacker and ***victim*** share some memory block b

Clflush on x86

1. ***Flush:*** Remove b from cache

2. ***Victim*** runs

3. ***Reload: Access again*** b and ***measure*** time

Attacker learns if b has been accessed

Y. Yarom, K. Falkner — Flush+Reload: a high resolution, low noise L3 cache side-channel attack — Usenix Security 2014

Tidbits

Tidbits

Many more attacks: Flush+Flush, Evict + Flush, Reload + Refresh…

Tidbits

Many more attacks: Flush+Flush, Evict + Flush, Reload + Refresh…

Bandwidth: up to ~50 KB/s (Amazon EC2), up to ~500 KB/s (native)

Tidbits

Many more attacks: Flush+Flush, Evict + Flush, Reload + Refresh…

Bandwidth: up to ~50 KB/s (Amazon EC2), up to ~500 KB/s (native)

What can we do with such attacks?

- Prime + Probe and Evict+Time used to recover AES keys
- Flush + Reload used to recover keys from RSA implementation of GnuPG
- They can be run even from Javascript
- Stream videos across Amazon EC2 instances (see https://youtu.be/yPZmiRi_c-o)

Countermeasures

Countermeasures

Software countermeasures

Countermeasures

Software countermeasures

- Make ***timing measurements*** difficult
  - Not too effective

Countermeasures

Software countermeasures

- Make ***timing measurements*** difficult
  - Not too effective
- ***Constant-time programming***
  - Make memory accesses and control-flow independent of secrets!

Countermeasures

Software countermeasures

- Make ***timing measurements*** difficult
  - Not too effective
- ***Constant-time programming***
  - Make memory accesses and control-flow independent of secrets!

Hardware countermeasures

Countermeasures

Software countermeasures

- Make ***timing measurements*** difficult
  - Not too effective
- ***Constant-time programming***
  - Make memory accesses and control-flow independent of secrets!

Hardware countermeasures

- ***Isolation*** between different processes/security domains
  - Cache coloring/slicing, …

Countermeasures

Software countermeasures

- Make ***timing measurements*** difficult
  - Not too effective
- ***Constant-time programming***
  - Make memory accesses and control-flow independent of secrets!

Hardware countermeasures

- ***Isolation*** between different processes/security domains
  - Cache coloring/slicing, …
- Secure ***randomized caches***
  - Makes targeting ***specific cache sets*** more challenging

Conclusions

Conclusions

Conclusions

Caches ***speed up*** computation but can ***leak*** information

Conclusions

Caches ***speed up*** computation but can ***leak*** information

There are ***many different types*** of attacks

Conclusions

Caches ***speed up*** computation but can ***leak*** information

There are ***many different types*** of attacks

Preventing leaks is difficult but there are some ***promising countermeasures***

Conclusions

Caches ***speed up*** computation but can ***leak*** information

There are ***many different types*** of attacks

Preventing leaks is difficult but there are some ***promising countermeasures***

Cache-based attacks can be used as ***building blocks*** in other attacks