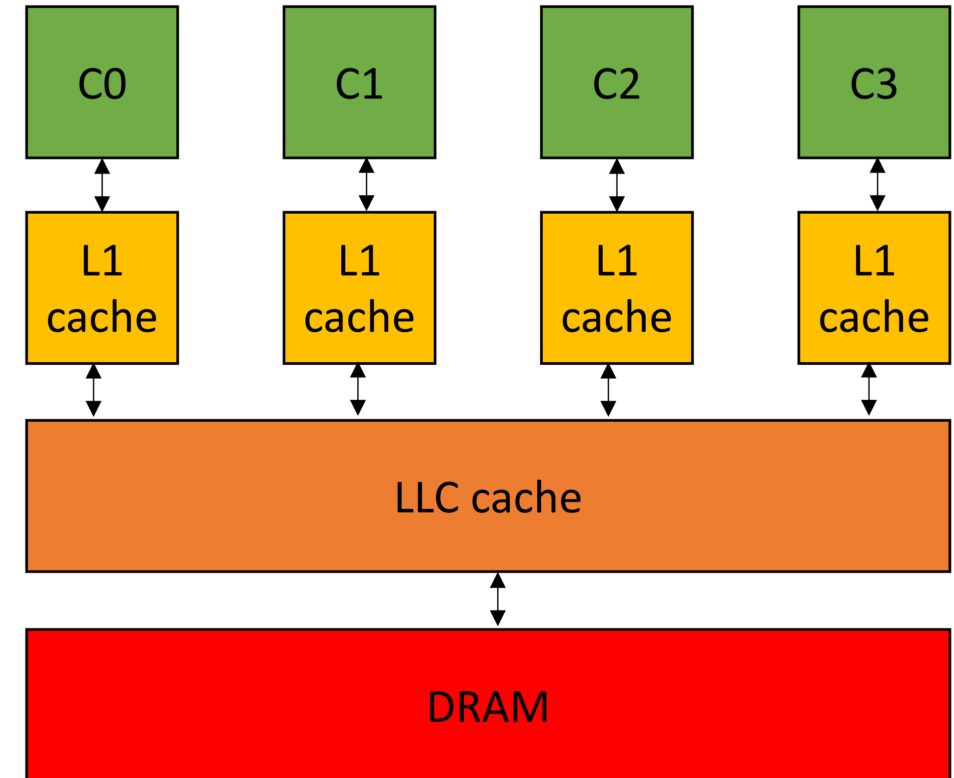
CSE113: Parallel Programming

April 6, 2021

- **Topic**: Architecture and Compiler Overview 2
  - Review of last week's concepts
  - Cache organization
  - Cache coherence
  - Example



Announcements

- Next Thursday
  - Last disruption for the quarter!
  - I will post the recorded lecture by Thursday noon
- Homework 1 will be posted by Thursday midnight
- Reese will post a tool tutorial on what you need
  - Docker container
  - A C++ compiler
  - Python3
  - Bash command line interface

Announcements

- My office hours moved to Friday
  - 3 - 5 pm

Lecture Schedule

- Quiz
- Overview of last week: Compilers, Concurrency, Cache lines
- Cache Organization and Coherence: direct mapped vs. associative, MESI protocal
- Example: false sharing

Lecture Schedule

- **Quiz**
- Overview of last week: Compilers, Concurrency, Cache lines
- Cache Organization and Coherence: direct mapped vs. associative, MESI protocal
- Example: false sharing

Quiz

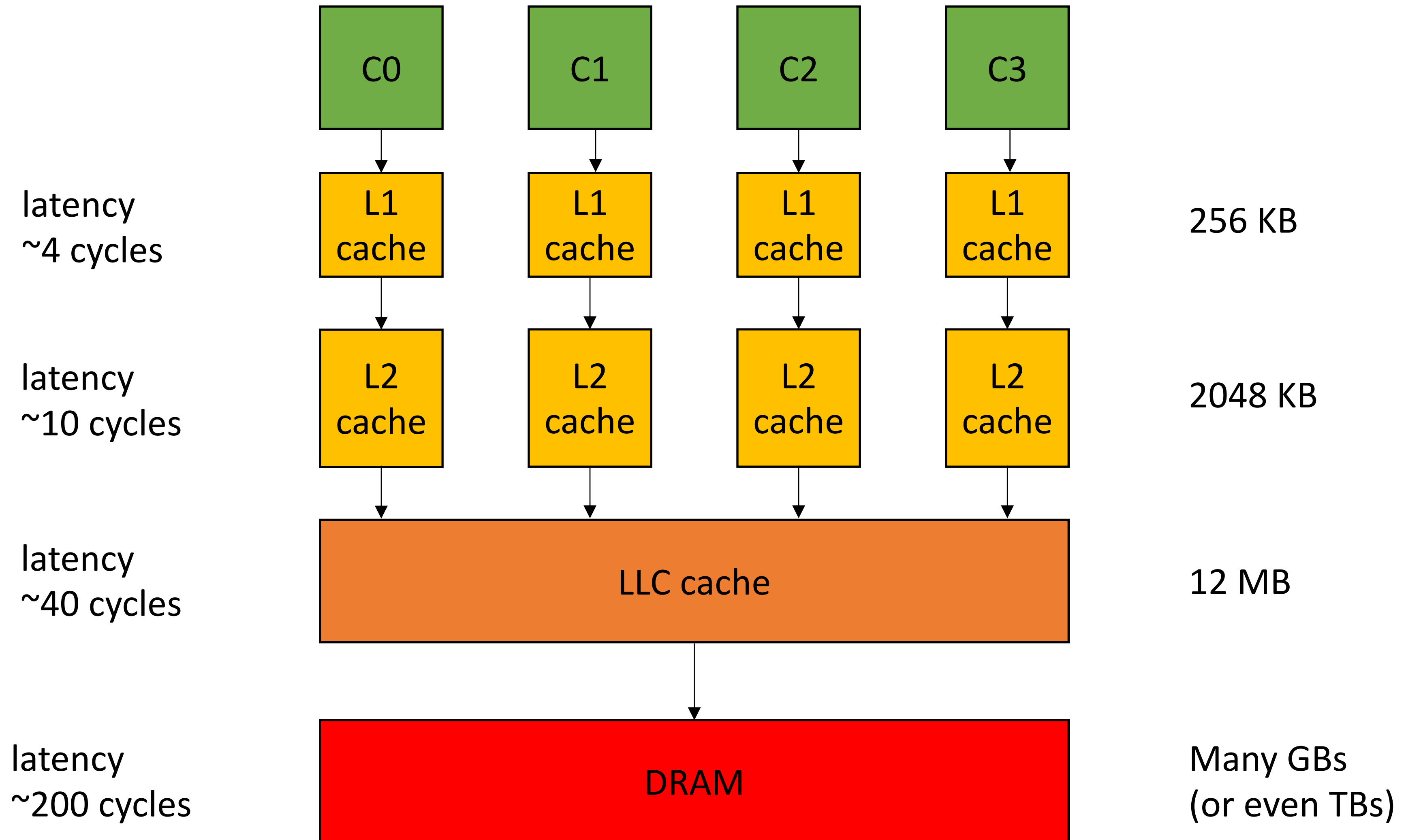
- https://docs.google.com/forms/d/e/1FAIpQLSeohLPi48GyX8l1-xoIMFicFt4ofJi5pfh3gWzARMfl8WTI8Q/viewform?usp=sf_link

- It will be in the chat too

Answers

- Assuming 'a' is a non-empty array of integers, how many 3-address-code instructions would the following C++ expression be compiled into: a[0]++
- How many levels of cache does a typical x86 system have?
- How many doubles can be stored in a cache line?

Answer for #2



Answer for #3

- Cache line size for x86: 64 bytes:
  - 64 chars
  - 32 shorts
  - 16 float or int
  - 8 double or long
  - 4 long long

Lecture Schedule

- Quiz
- Overview of last week: Compilers, Concurrency, Cache lines
- Cache Organization and Coherence: direct mapped vs. associative, MESI protocal
- Example: false sharing

Lecture Schedule

- Quiz
- **Overview of last week**: Compilers, Concurrency, Cache lines
- Cache Organization and Coherence: direct mapped vs. associative, MESI protocal
- Example: false sharing

How are complicated expressions executed?

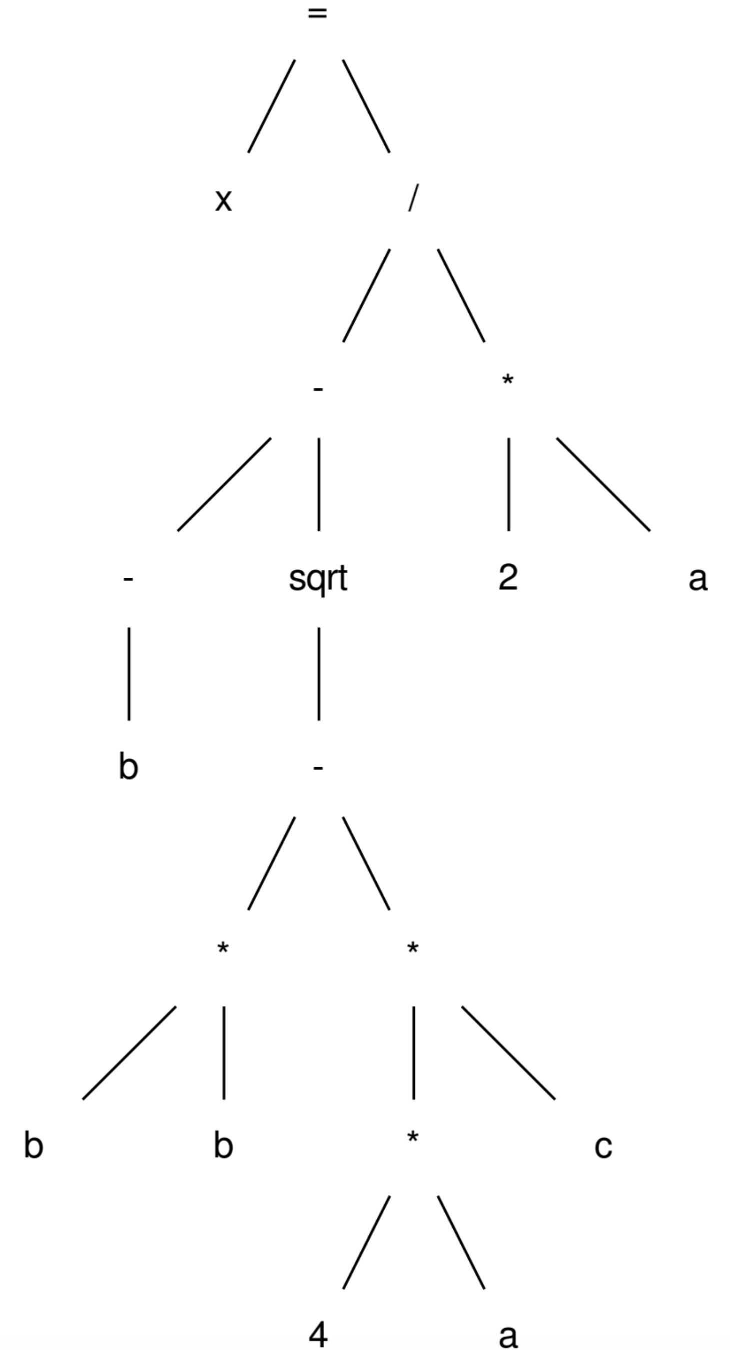
Quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
x = (-b - sqrt(b*b - 4 * a * c)) / (2*a)
```

```
x = (-b - sqrt(b*b - 4 * a * c)) / (2*a)
```

A compiler will turn this into an *abstract syntax tree* (AST)



Simplify this code:

post-order traversal, using temporary variables

```
r0 = neg(b);
r1 = b * b;
r2 = 4 * a;
r3 = r2 * c;
r4 = r1 – r3;
r5 = sqrt(r4);
r6 = r0 – r5;
r7 = 2 * a;
r8 = r6 / r7;
x  = r8;
```

- This is not exactly an ISA
  - unlimited registers
  - not always a 1-1 mapping of instructions.
- but it is much easier to translate to the ISA
- We call this an intermediate representation, or IR
- Examples of IR: LLVM, SPIR-V

```c
// Type your code here, or load an example.
float sqrt(float x);

float add(float a, float b, float c) {
    return (-b - sqrt(b*b - 4 * a * c)) / (2*a);

}
```

```llvm

define dso_local float @_Z3addfff(float %0, float %1, float %2) #0 !dbg !
  %4 = alloca float, align 4
  %5 = alloca float, align 4
  %6 = alloca float, align 4
  store float %0, float* %4, align 4
  call void @llvm.dbg.declare(metadata float* %4, metadata !12, metadata
  store float %1, float* %5, align 4
  call void @llvm.dbg.declare(metadata float* %5, metadata !14, metadata
  store float %2, float* %6, align 4
  call void @llvm.dbg.declare(metadata float* %6, metadata !16, metadata
  %7 = load float, float* %5, align 4, !dbg !18
  %8 = fneg float %7, !dbg !19
  %9 = load float, float* %5, align 4, !dbg !20
  %10 = load float, float* %5, align 4, !dbg !21
  %11 = fmul float %9, %10, !dbg !22
  %12 = load float, float* %4, align 4, !dbg !23
  %13 = fmul float 4.000000e+00, %12, !dbg !24
  %14 = load float, float* %6, align 4, !dbg !25
  %15 = fmul float %13, %14, !dbg !26
  %16 = fsub float %11, %15, !dbg !27
  %17 = call float @_Z4sqrtf(float %16), !dbg !28
  %18 = fsub float %8, %17, !dbg !29
  %19 = load float, float* %4, align 4, !dbg !30
  %20 = fmul float 2.000000e+00, %19, !dbg !31
  %21 = fdiv float %18, %20, !dbg !32
  ret float %21, !dbg !33
}
```

A ▾ Output... ▾ Filter... ▾ Libraries + Add new... ▾ Add tool... ▾

C program **llvm IR**

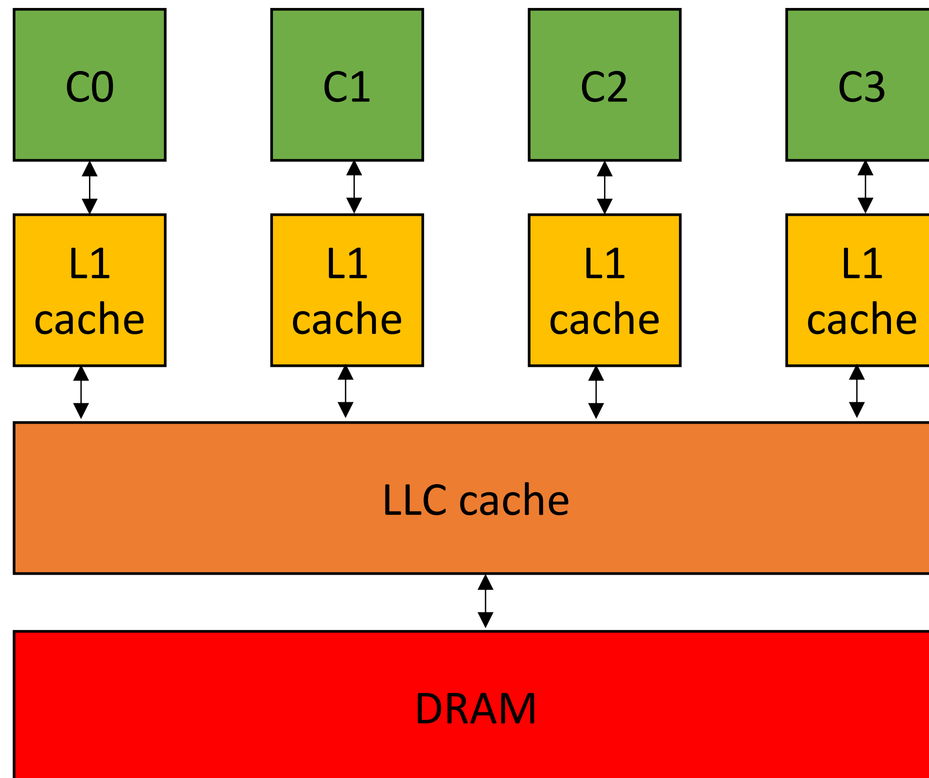
Memory accesses

```
int increment(int *a) {
    a[0]++;
}
```

```
%5 = load i32, i32* %4
%6 = add nsw i32 %5, 1
store i32 %6, i32* %4
```

Unless explicitly expressed in the programming language, loads and stores are split into multiple instructions!

Architecture



Core

A core executes a stream
of sequential ISA instructions

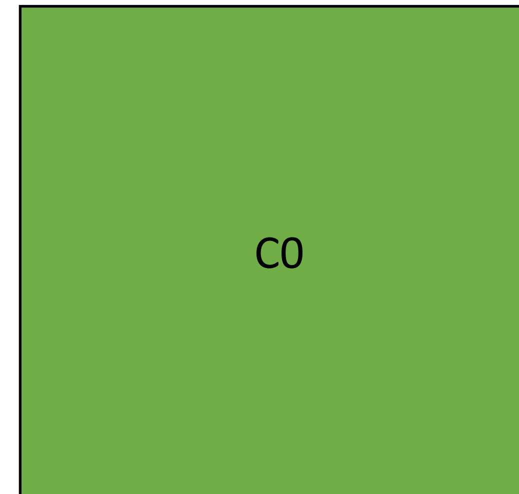
A good mental model executes
1 ISA instruction per cycle

3 Ghz means 3B cycles per second
1 ISA instruction takes .33 ns

Compiled function #0

```
13        movd    eax, xmm0
14        xor     eax, 2147483648
15        movd    xmm0, eax
16        movss   dword ptr [rbp - 16], xmm0
17        movss   xmm0, dword ptr [rbp - 8]
18        mulss   xmm0, dword ptr [rbp - 8]
19        movss   xmm1, dword ptr [rip + .LCPI0_1]
20        mulss   xmm1, dword ptr [rbp - 4]
21        mulss   xmm1, dword ptr [rbp - 12]
22        subss   xmm0, xmm1
23        call    sqrt(float)
24        movaps  xmm1, xmm0
25        movss   xmm0, dword ptr [rbp - 16]
26        subss   xmm0, xmm1
27        movss   xmm1, dword ptr [rip + .LCPI0_0]
28        mulss   xmm1, dword ptr [rbp - 4]
29        divss   xmm0, xmm1
```

Thread 0



Core

Concurrency vs. Parallelism

Concurrency vs. Parallelism

- Abstract tasks:
  - In the abstract: a sequence of computation
  - *Given an input, produces an output*

Concurrency vs. Parallelism

- Abstract tasks:
  - In the abstract: a sequence of computation
  - *Given an input, produces an output*

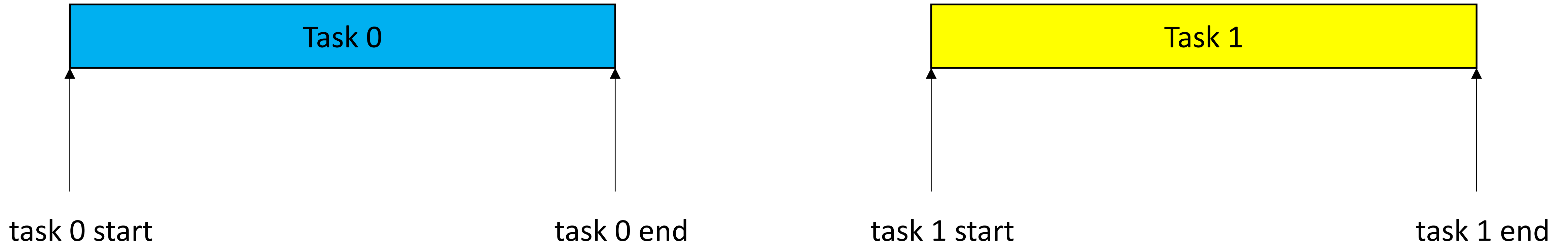
- Concrete tasks:
  - Application (e.g. Spotify and Chrome)
  - Function
  - Loop iterations
  - Individual instructions
  - Circuit level?

coarse

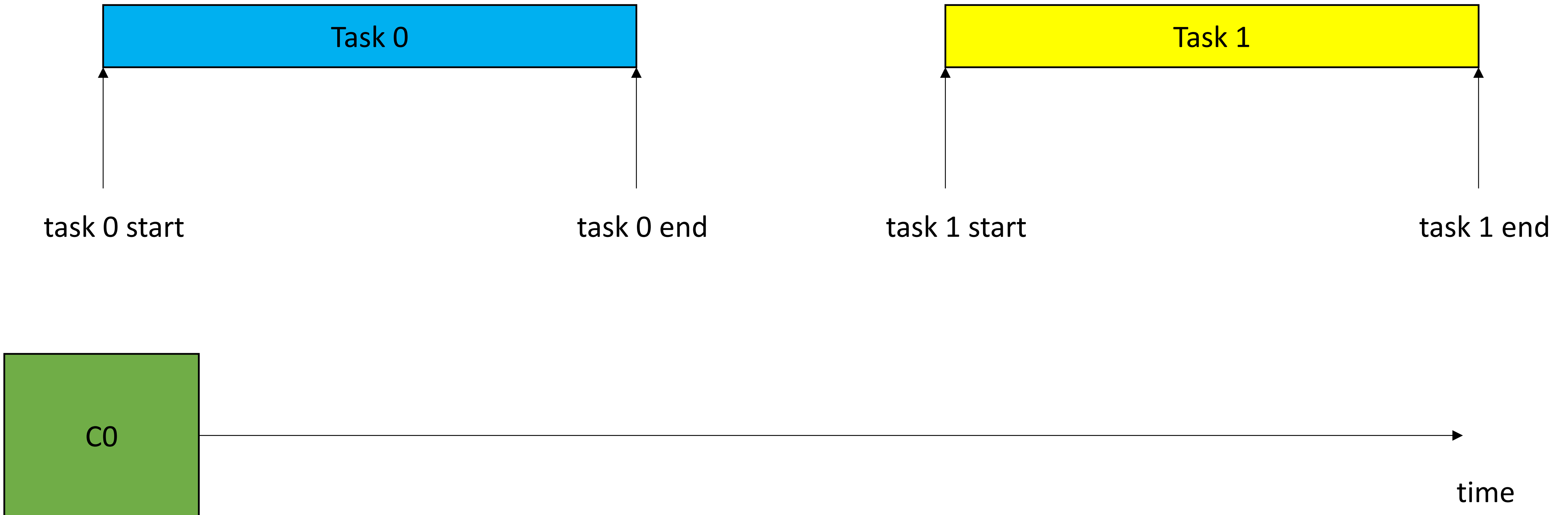
granularity

fine

Concurrency vs. Parallelism

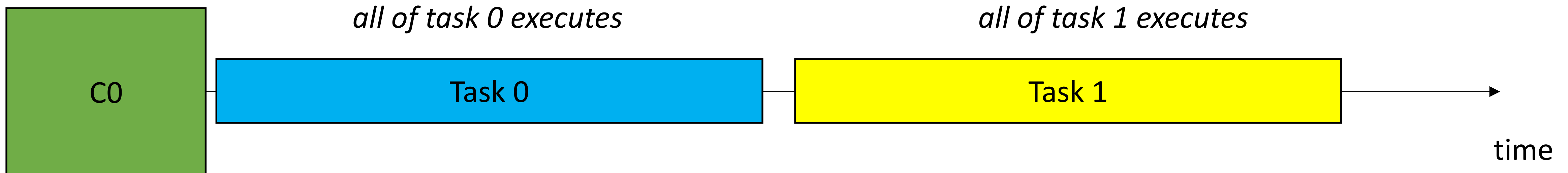


Concurrency vs. Parallelism



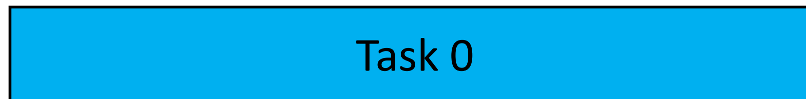
Concurrency vs. Parallelism

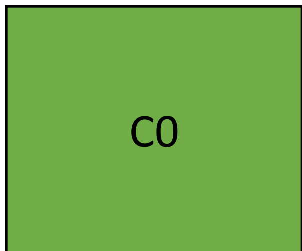
Sequential execution
Not concurrent or parallel

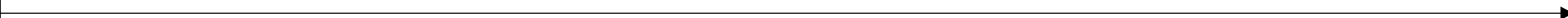


Concurrency vs. Parallelism

The OS can preempt a thread
(remove it from the hardware resource)



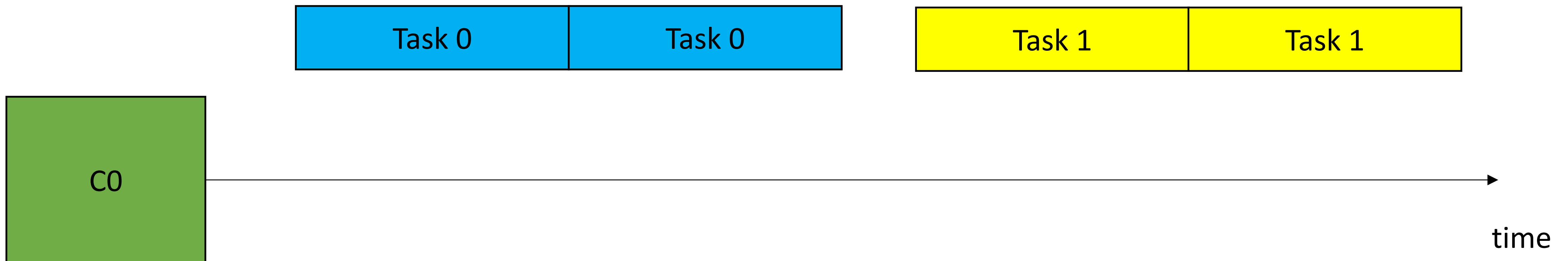




time

Concurrency vs. Parallelism

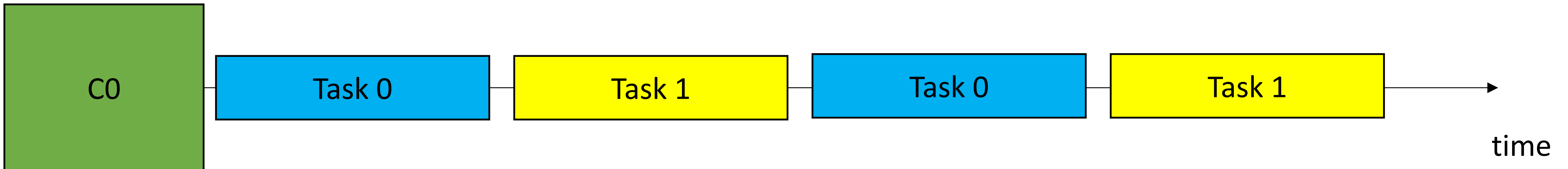
The OS can preempt a thread
(remove it from the hardware resource)



Concurrency vs. Parallelism

The OS can preempt a thread
(remove it from the hardware resource)

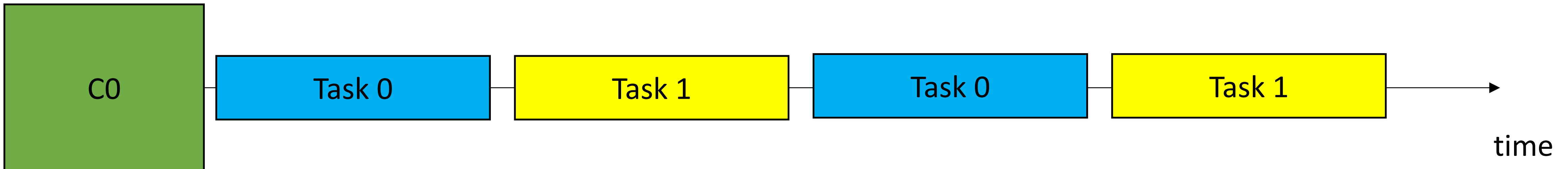
tasks are interleaved on the same processor



Concurrency vs. Parallelism

- Definition:
  - 2 tasks are **concurrent** if there is a point in the execution where both tasks have started and neither has ended.

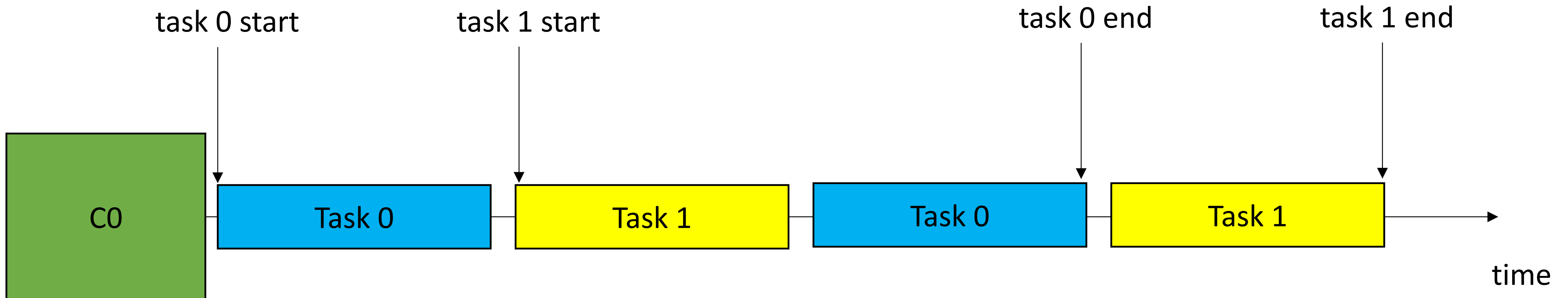
The OS can preempt a thread
(remove it from the hardware resource)



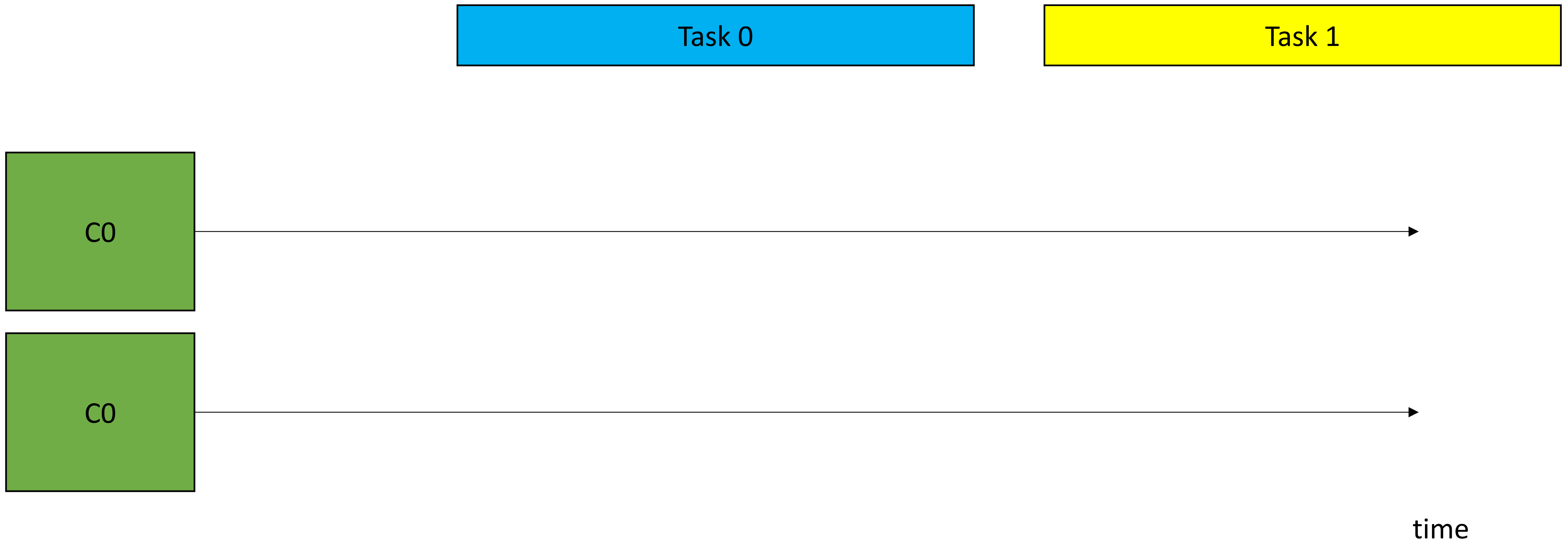
Concurrency vs. Parallelism

- Definition:
  - 2 tasks are **concurrent** if there is a point in the execution where both tasks have started and neither has ended.

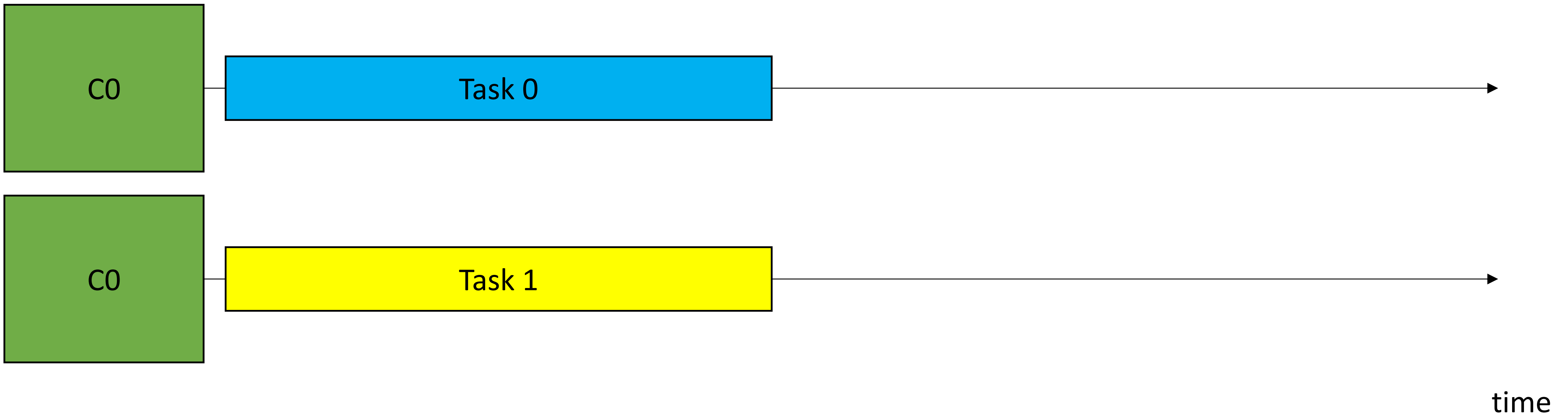
The OS can preempt a thread (remove it from the hardware resource)



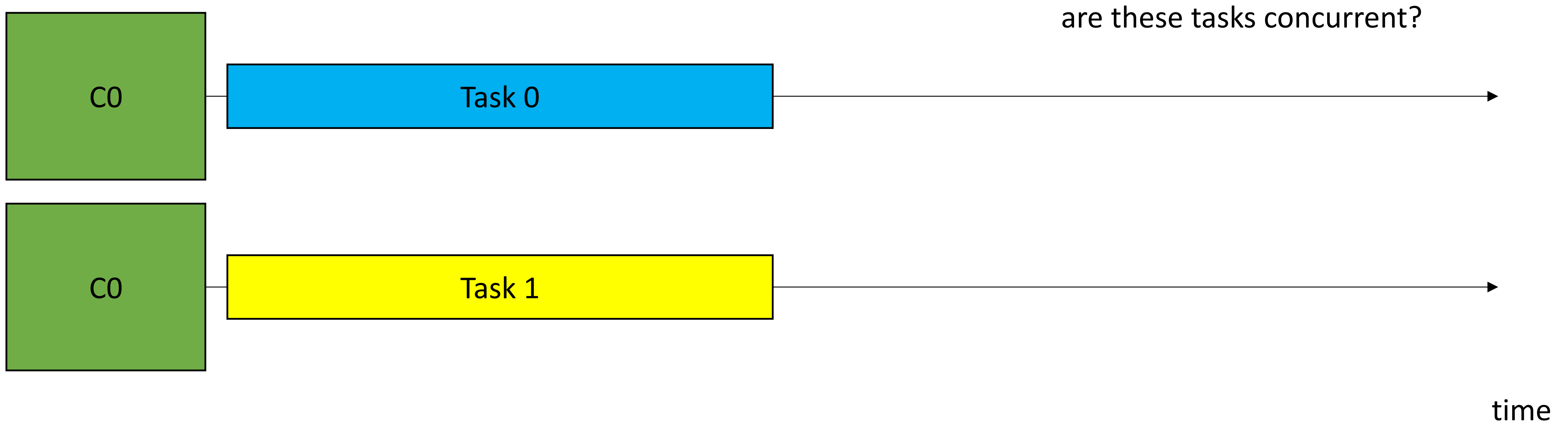
Concurrency vs. Parallelism



Concurrency vs. Parallelism

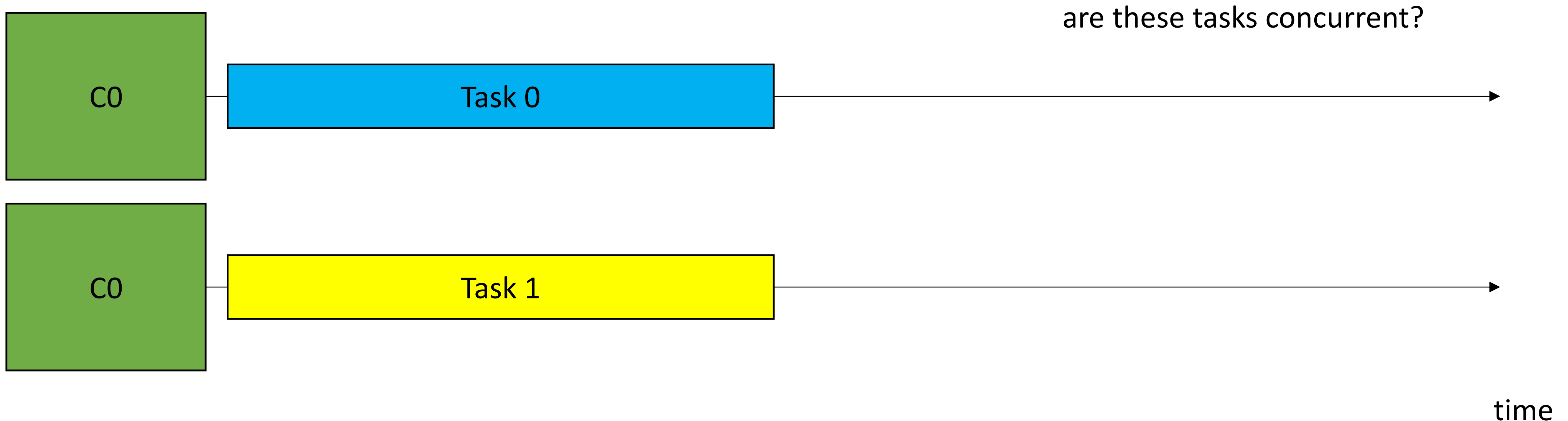


Concurrency vs. Parallelism

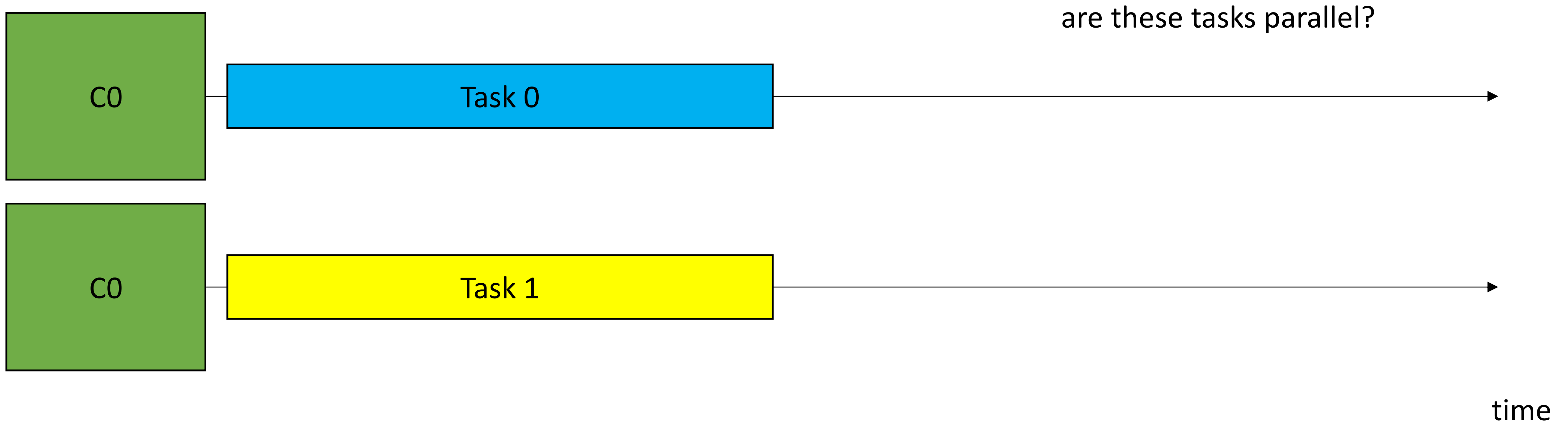


Concurrency vs. Parallelism

- 2 tasks are **concurrent** if there is a point in the execution where both tasks have started and neither has ended.

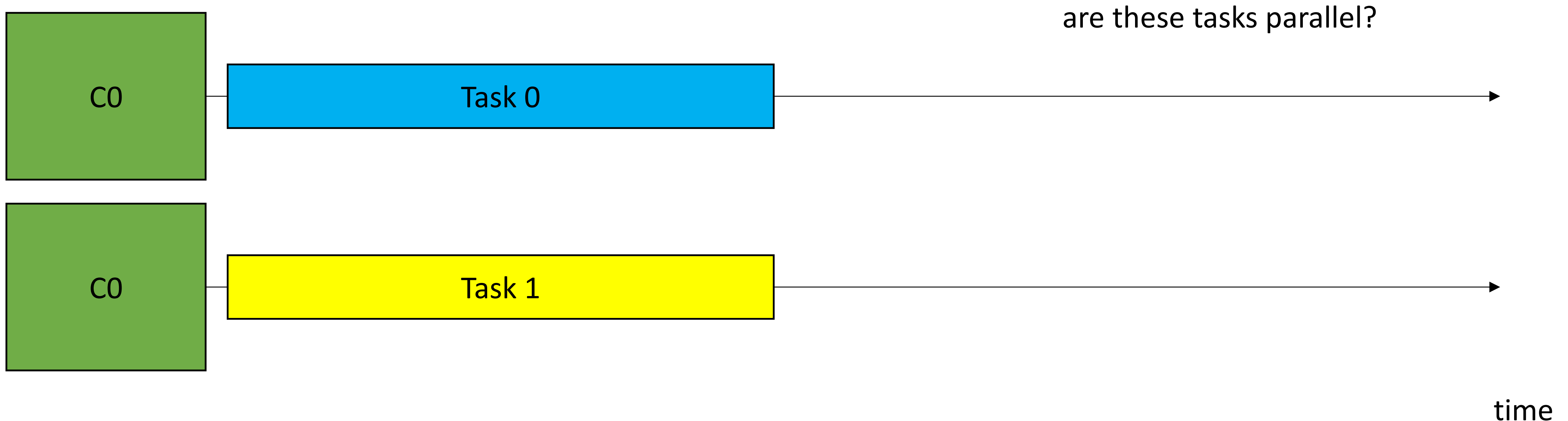


Concurrency vs. Parallelism



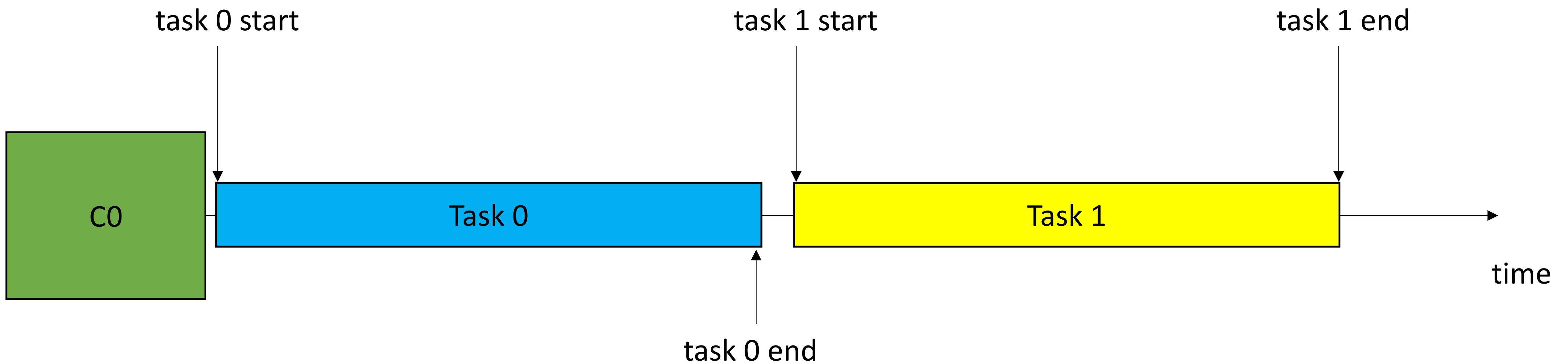
Concurrency vs. Parallelism

- Definition:
  - An execution is **parallel** if there is a point in the execution where computation is happening simultaneously



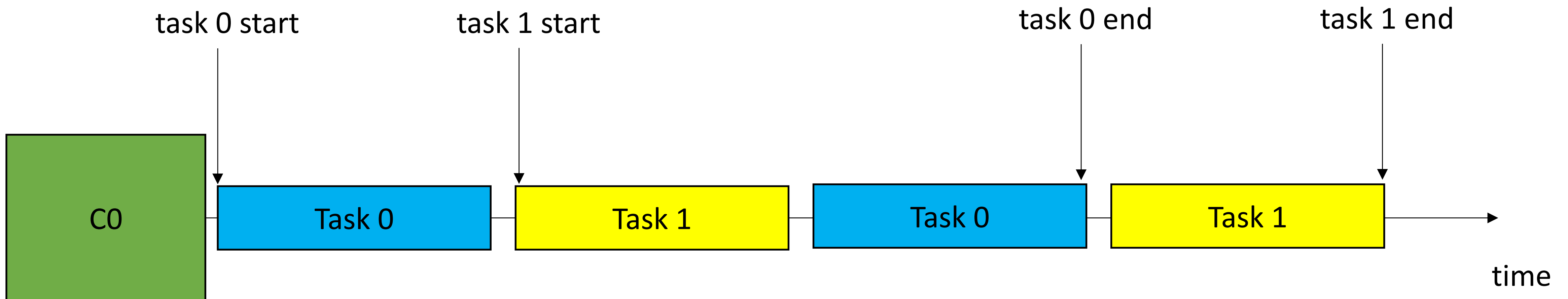
Concurrency vs. Parallelism

- Examples:
  - Neither concurrent or parallel (sequential)



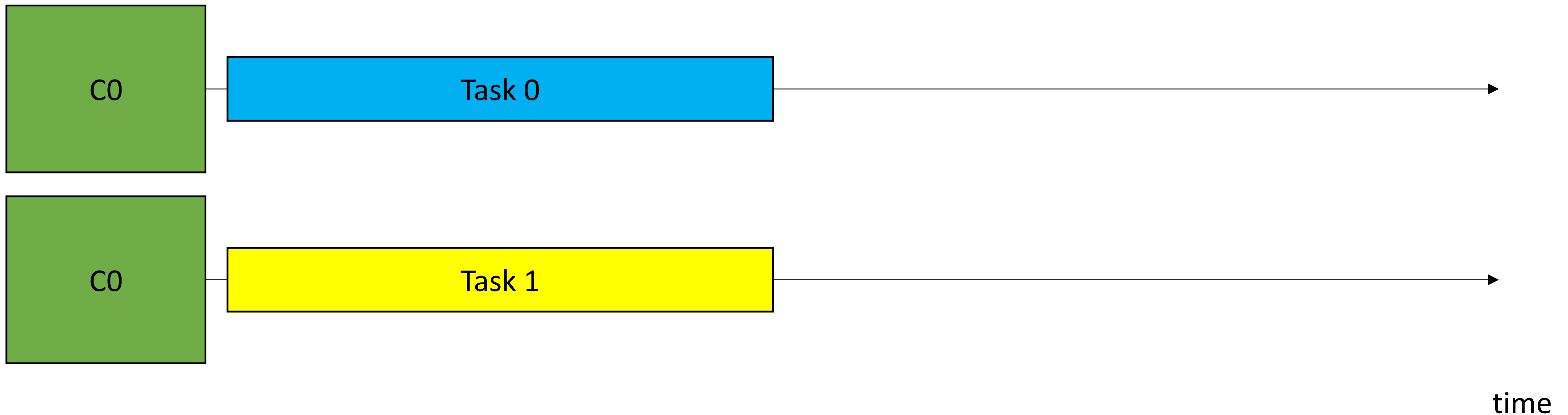
Concurrency vs. Parallelism

- Examples:
  - Concurrent but not parallel



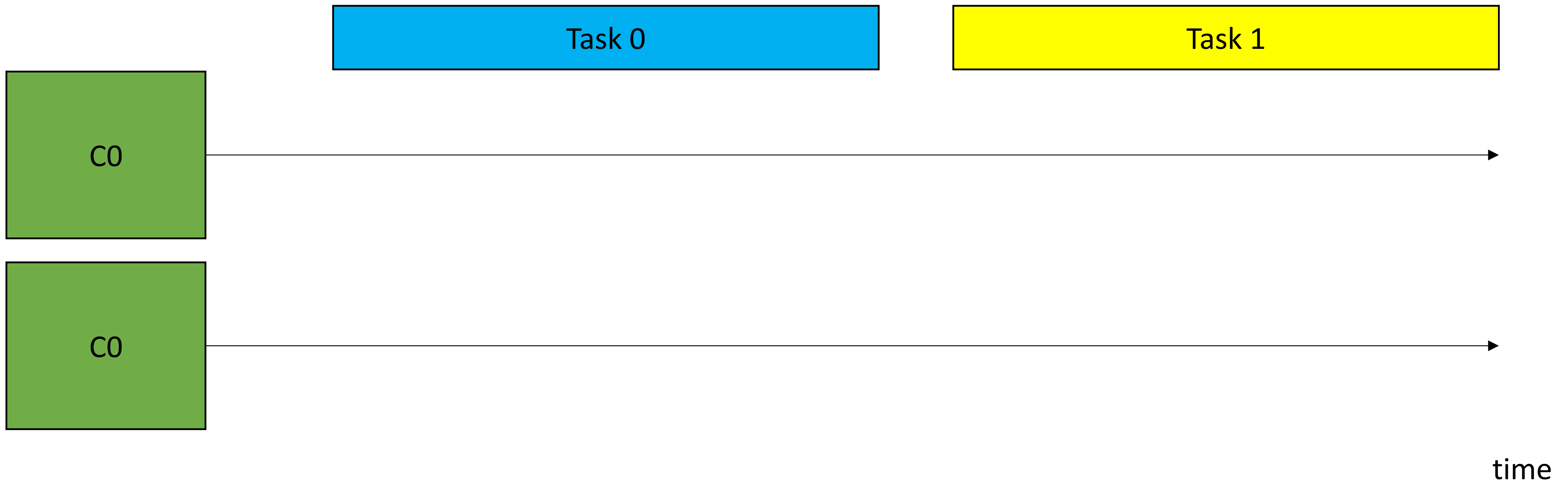
Concurrency vs. Parallelism

- Examples:
  - Parallel and Concurrent



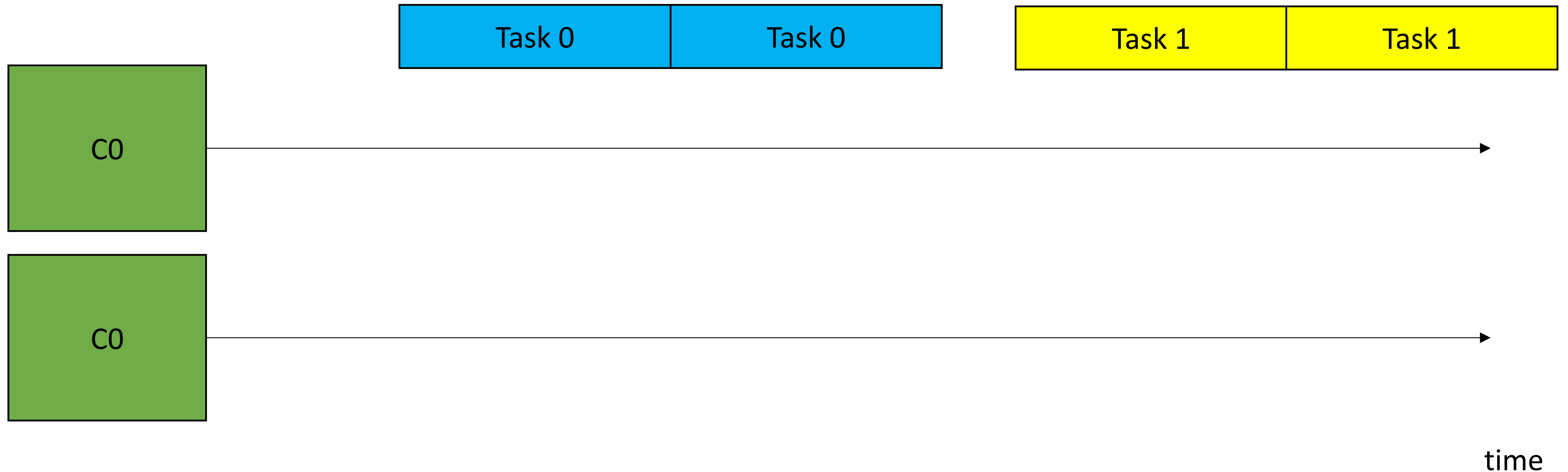
Concurrency vs. Parallelism

- Examples:
  - Parallel but not concurrent?



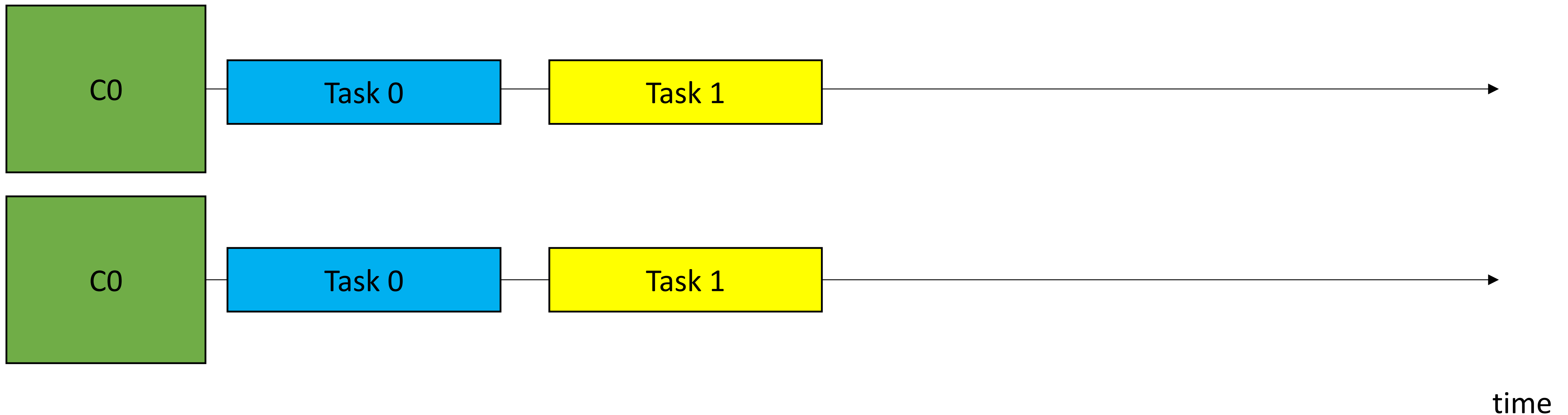
Concurrency vs. Parallelism

- Examples:
  - Parallel but not concurrent?



Concurrency vs. Parallelism

- Examples:
  - Parallel execution but task 0 and task 1 are not concurrent?



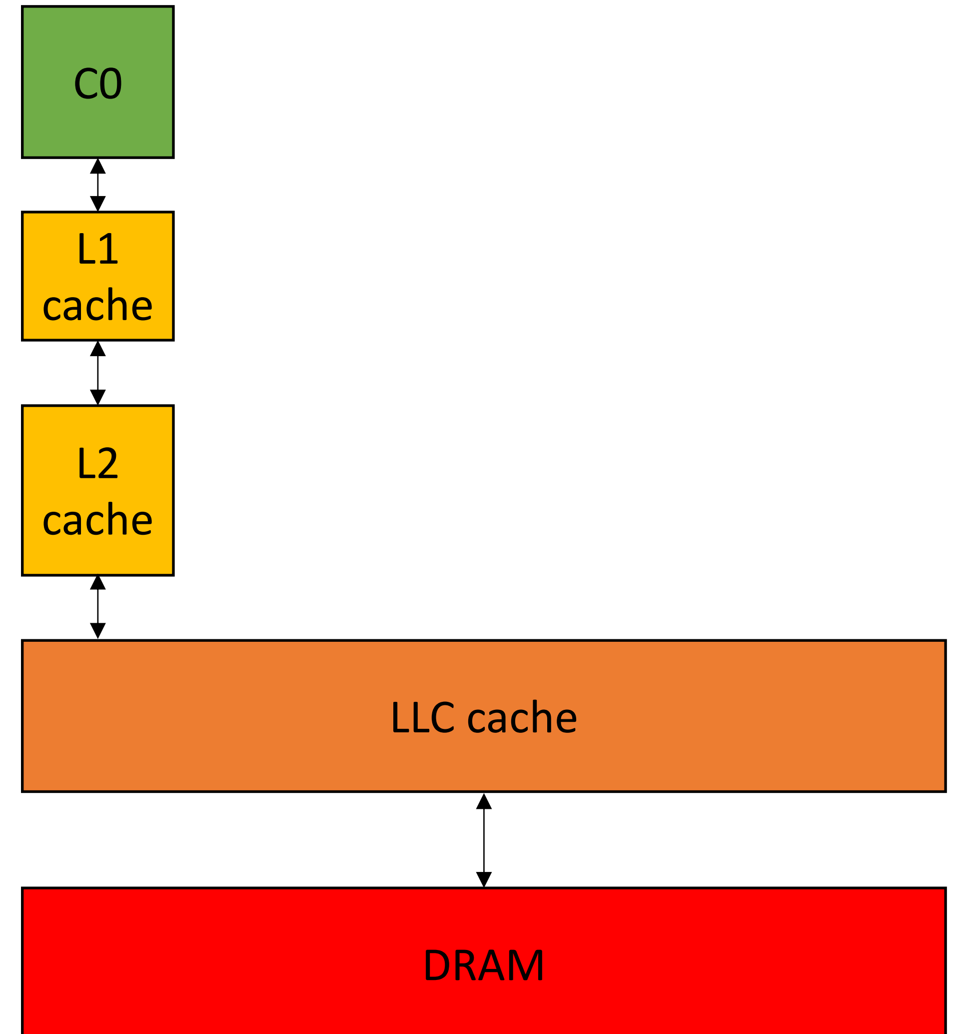
Concurrency vs. Parallelism

- In practice:
  - Terms are often used interchangeably.
  - *Parallel programming* is often used by high performance engineers when discussing using parallelism to accelerate things
  - *Concurrent programming* is used more by interactive applications, e.g. event driven interfaces.

Cache lines

```
int increment(int *a) {
    a[0]++;
}
```

```
%5 = load i32, i32* %4
%6 = add nsw i32 %5, 1
store i32 %6, i32* %4
```

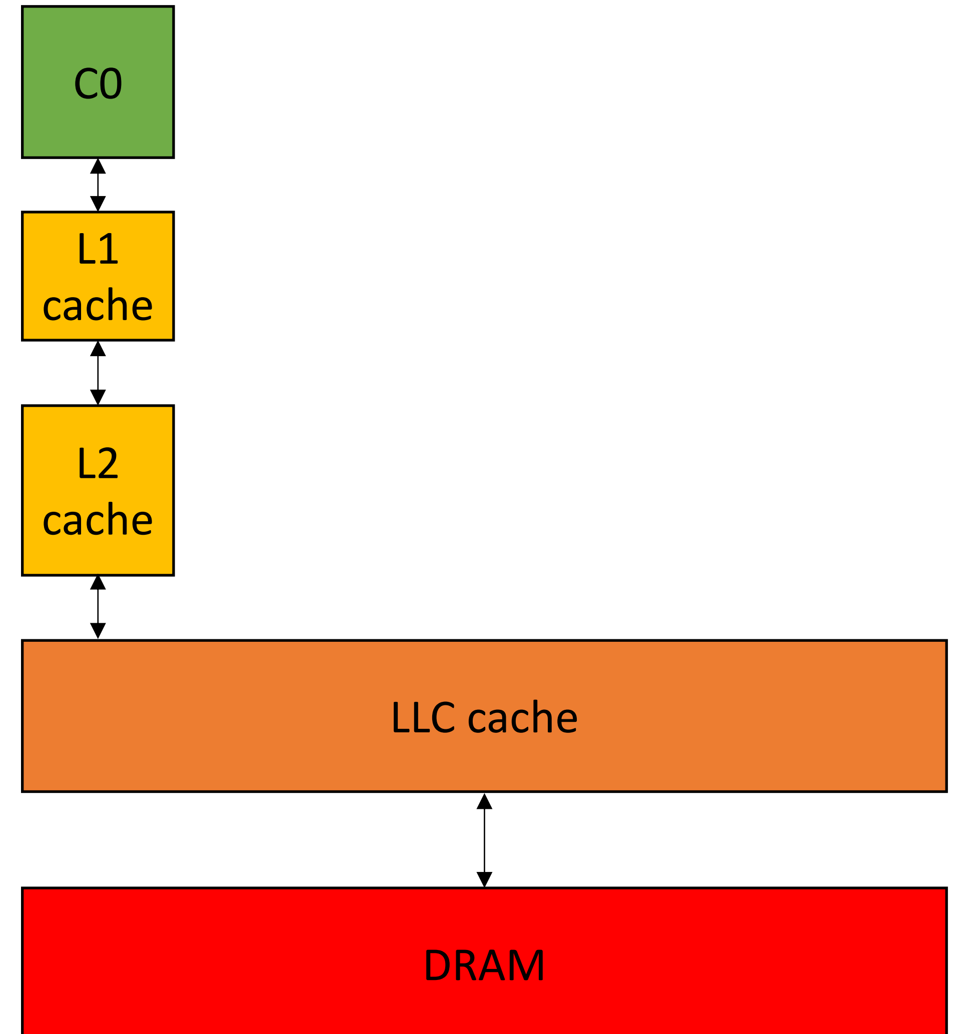


Assume a[0] is not in the cache

Cache lines

```
int increment(int *a) {
    a[0]++;
}
```

```
%5 = load i32, i32* %4
%6 = add nsw i32 %5, 1
store i32 %6, i32* %4
```



Assume a[0] is not in the cache

Passing arrays in C++

```
int increment(int *a) {
    a[0]++;
}
```

```
int increment_alt1(int a[1]) {
    a[0]++;
}
```

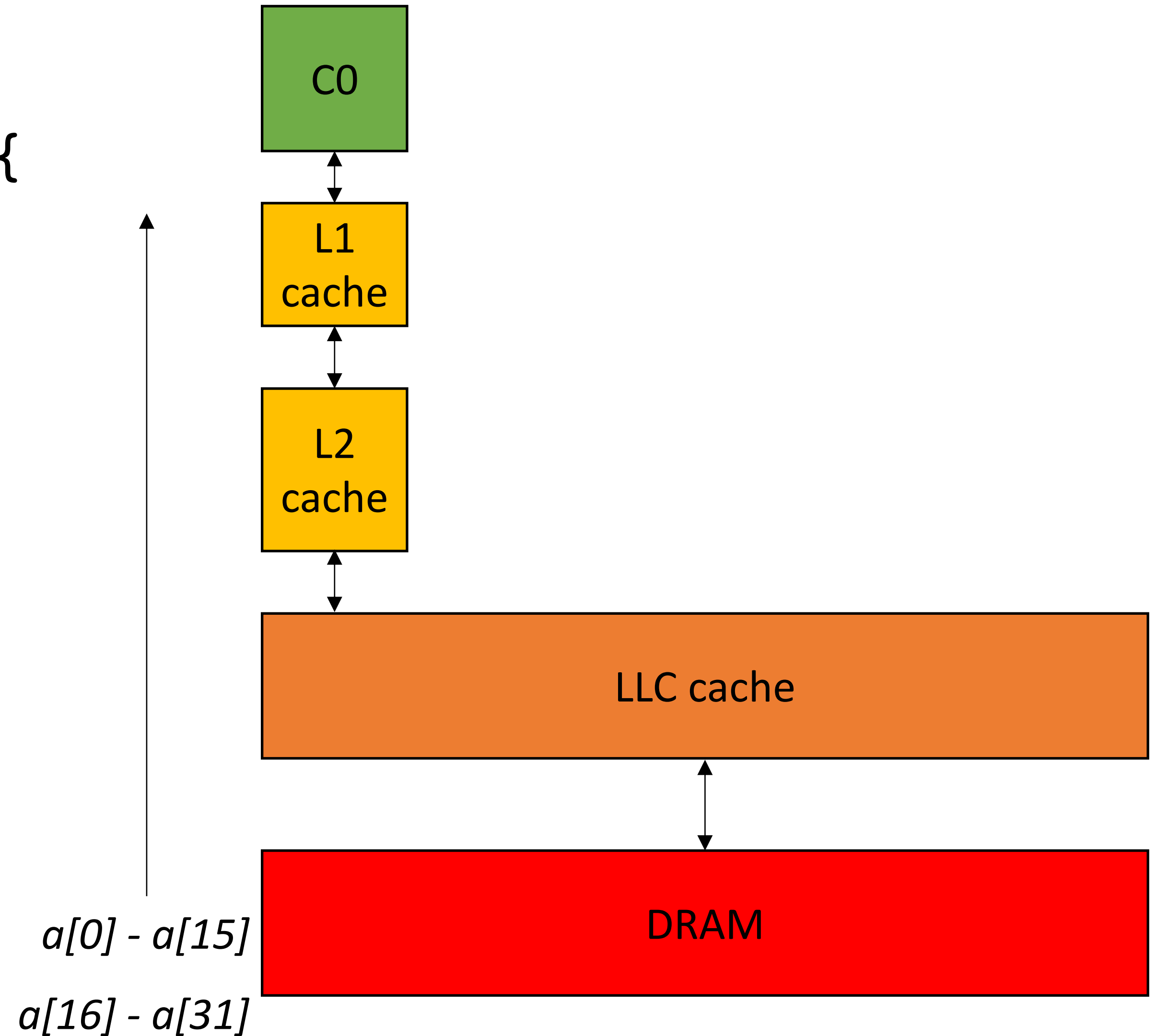
Not checked at compile time! but hints can help with compiler optimizations. Also good self documenting code.

```
int increment_alt2(int a[]) {
    a[0]++;
}
```

Cache alignment

```
int increment_several(int *b) {
    b[0]++;
    b[15]++;
}

int foo(int *a) {
    increment_several(&a[8])
}
```

This loads a[8]
This loads a[23], a miss!



Passing pointers

```
int foo0(int *a) {
   increment_several(&a[8])
}


int foo1(int *a) {
   increment_several(a + 8)
}
```

Memory Allocation

```
int allocate_int_array0() {
  int ar[16];
}
```

stack allocation

```
int allocate_int_array1() {
  int *ar = new int[16];
  delete[] ar;
}
```

C++ style

```
int allocate_int_array2() {
  int *ar = (int*)malloc(sizeof(int)*16);
  free(ar);
}
```

C style

Memory Allocation

```
int allocate_int_array0() {
  int ar[16];
}
```

stack allocation

```
int allocate_int_array1() {
  int *ar = new int[16];
  delete[] ar;
}
```

C++ style

```
int allocate_int_array2() {
  int *ar = (int*)malloc(sizeof(int)*16);
  free(ar);
}
```

C style

End of review

Lecture Schedule

- Quiz
- Overview of last week: Compilers, Concurrency, Cache lines
- Cache Organization and Coherence: direct mapped vs. associative, MESI protocal
- Example: false sharing

Lecture Schedule

- Quiz
- Overview of last week: Compilers, Concurrency, Cache lines
- **Cache Organization and Coherence**: direct mapped vs. associative, MESI protocal
- Example: false sharing

Cache organization

In this illustration, box is a cache line.

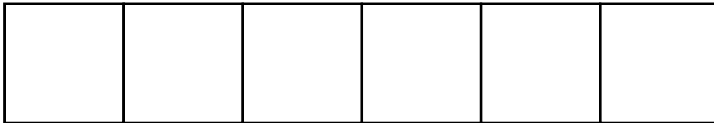
Assume we read only addresses that start a cache line

Cache is size 6 * 64 bytes

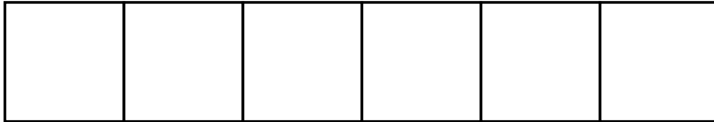
Memory is size 18 * 64 bytes

Cache

value

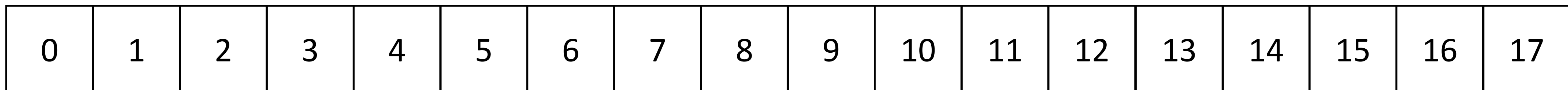


address

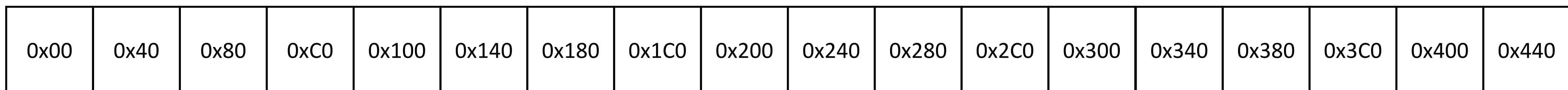


Memory

value



address

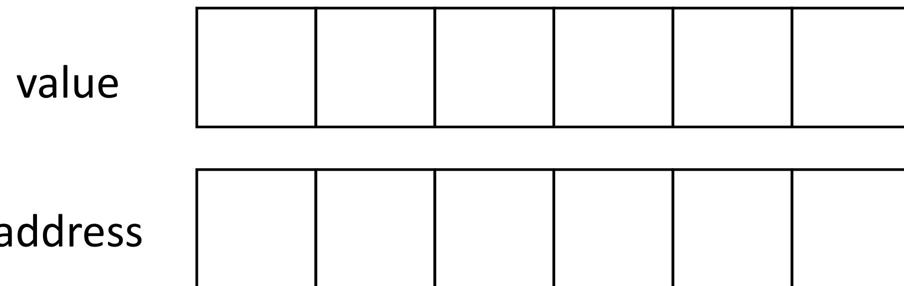


Cache organization

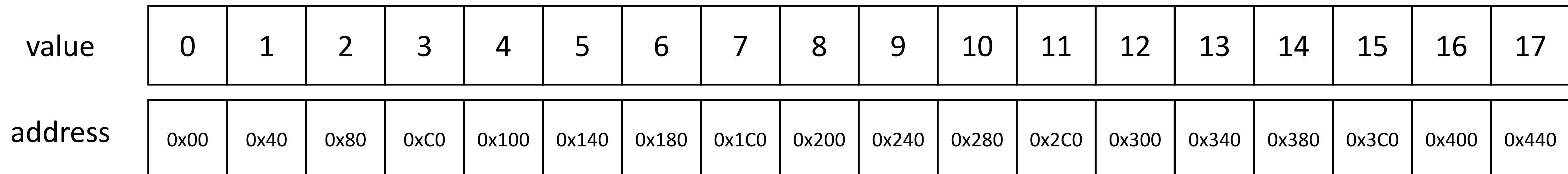
Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Cache



Memory

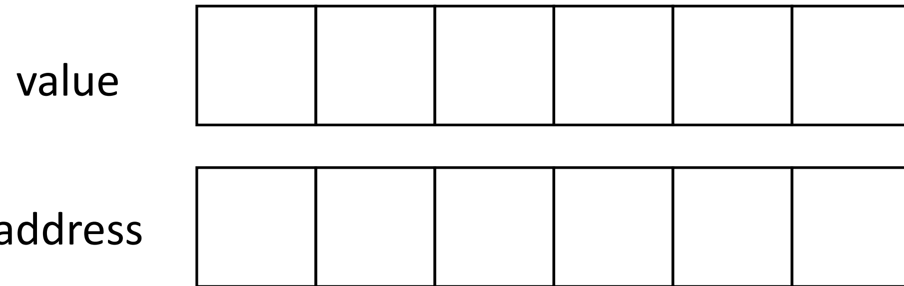


Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

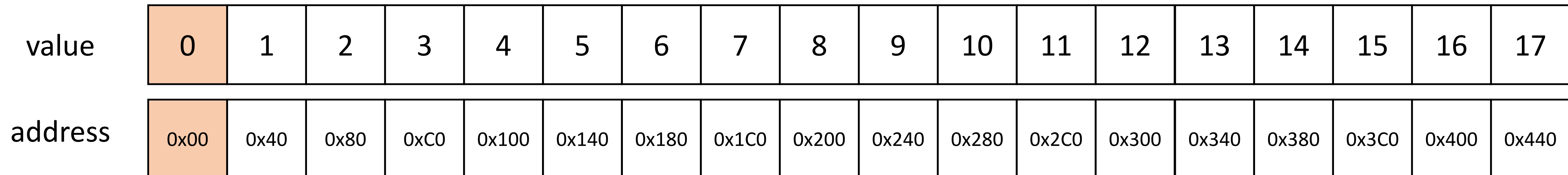
cache block location = (address/64) % (cache size)

Cache



Example: Read address 0x00

Memory



Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x00

Cache

value	0					
address	0x00					

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Cache

value	0					
address	0x00					

Example: Read address 0x1C0

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x80

Cache

value	0	7				
address	0x00	0x1C0				

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Cache

value	0	7				
address	0x00	0x1C0				

Example: Read address 0x80

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x80

Cache

value	0	7	2			
address	0x00	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x1C0

Cache

value	0	7	2			
address	0x00	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x1C0

Cache

value	0	7	2			
address	0x00	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x1C0

Cache

value	0	7	2			
address	0x00	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x180

Cache

value	0	7	2			
address	0x00	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x180

Cache

value	0	7	2			
address	0x00	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x180

Cache

evict!

value		7	2			
address		0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x180

Cache

value		7	2			
address		0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Direct mapped: every memory location can go exactly one place in the cache.

cache block location = (address/64) % (cache size)

Example: Read address 0x180

We had to evict even though there was room in the cache!

Cache

value	6	7	2			
address	0x180	0x1C0	0x80			

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

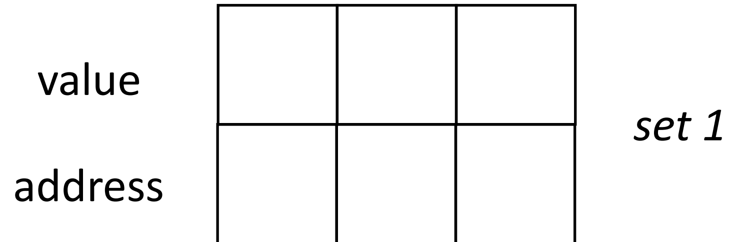
Cache organization

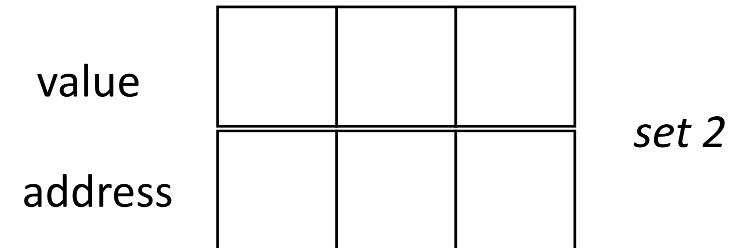
N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Cache

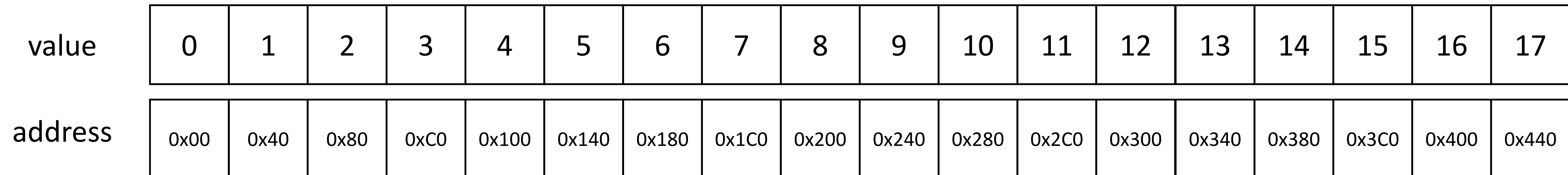




Read 0x00
Read 0x1C0
Read 0x40

example 2-way associative

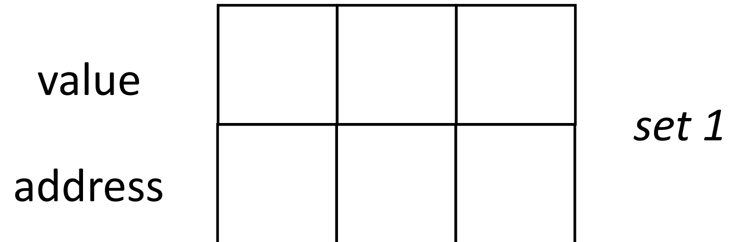
Memory

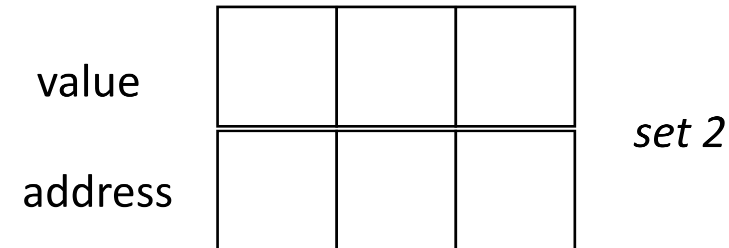


value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

Cache





N-way Associative: every memory location can go N places in the cache.

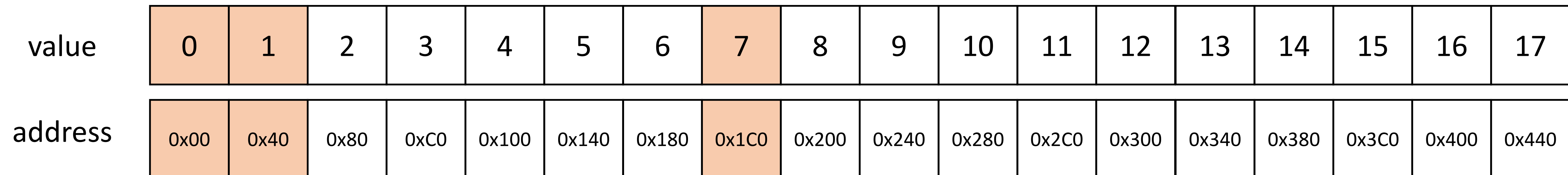
cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Read 0x00
Read 0x1C0
Read 0x40

example 2-way associative

Memory



Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an "intelligent" decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value			
address			

Read 0x00
Read 0x1C0
Read 0x40

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value			
address			

Read 0x180

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an "intelligent" decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value			
address			

Read 0x180

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value			
address			

Read 0x180

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an "intelligent" decision on which value to evict

Cache

value	0	1	7	*set 1*
address	0x00	0x40	0x1C0	
value	6			*set 2*
address	0x180			

Read 0x180

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value	6		
address	0x180		

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value	6		
address	0x180		

Read 0x300

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an “intelligent” decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value	6		
address	0x180		

Read 0x300

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an "intelligent" decision on which value to evict

Cache

set 1

value	0	1	7
address	0x00	0x40	0x1C0

set 2

value	6		
address	0x180		

example 2-way associative

Read 0x300

Evict the "least recently used" value

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an "intelligent" decision on which value to evict

Cache

value		1	7	*set 1*
address		0x40	0x1C0	
value	6			*set 2*
address	0x180			

Read 0x300

Evict the "least recently used" value

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

Cache organization

N-way Associative: every memory location can go N places in the cache.

cache block location (address/64) % (cache size / N)

Cache will make an "intelligent" decision on which value to evict

Cache

set 1

value	12	1	7
address	0x300	0x40	0x1C0

set 2

value	6		
address	0x180		

Read 0x300

Evict the "least recently used" value

example 2-way associative

Memory

value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
address	0x00	0x40	0x80	0xC0	0x100	0x140	0x180	0x1C0	0x200	0x240	0x280	0x2C0	0x300	0x340	0x380	0x3C0	0x400	0x440

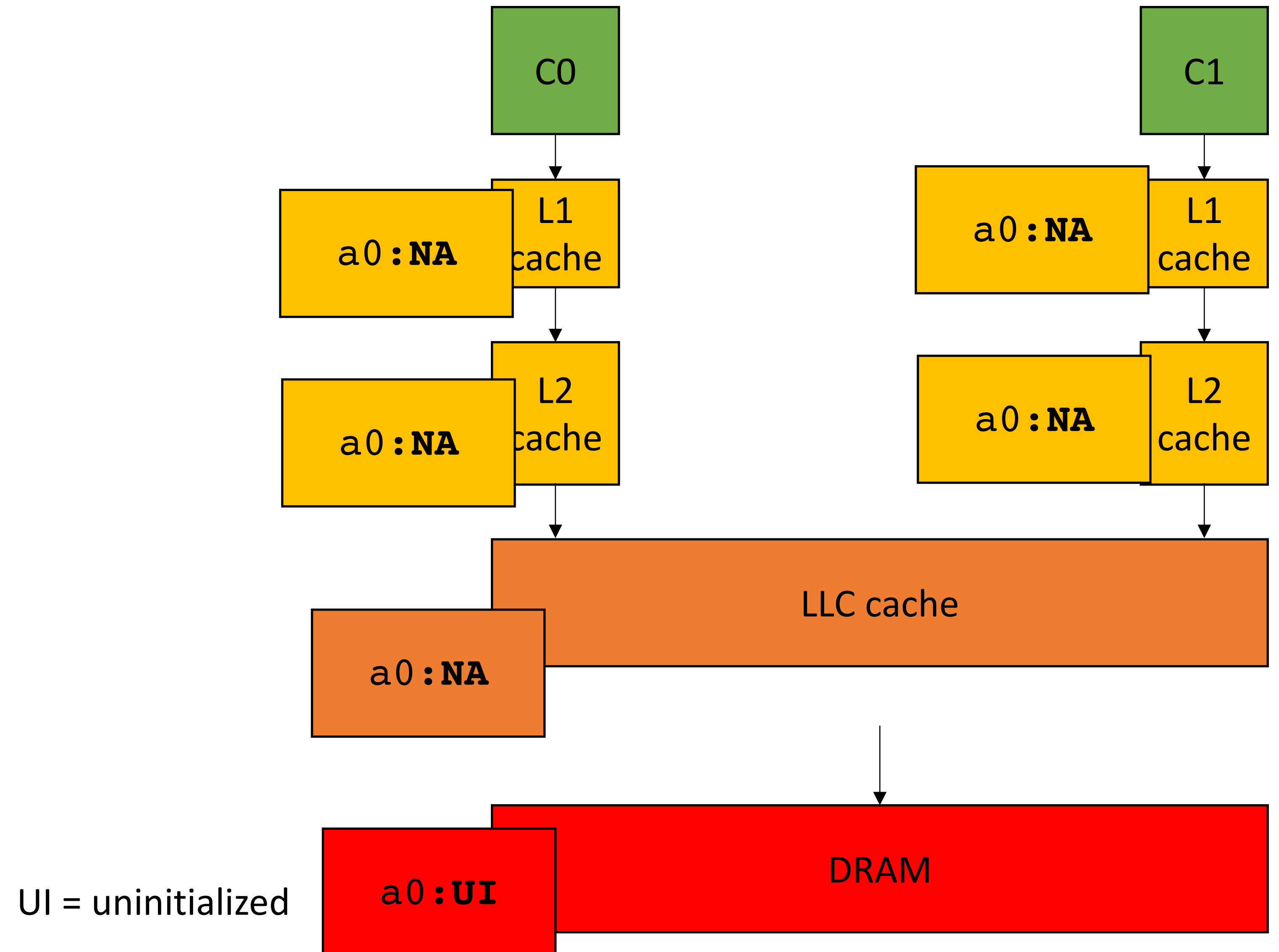
Cache organization

- Why aren’t caches fully associative?

Cache organization

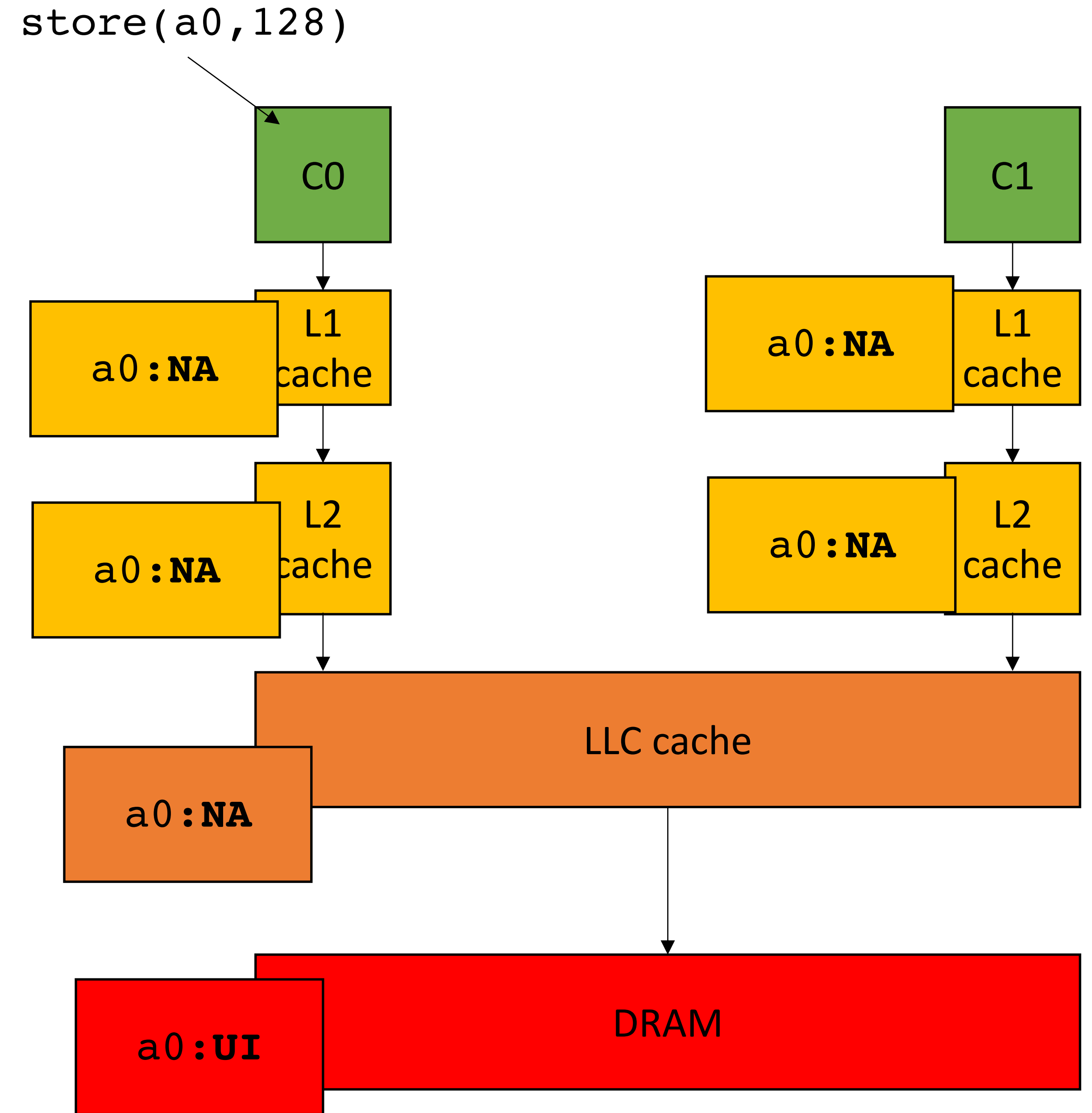
- For Intel Processors:
  - **L1** 8-way associative
  - **L2** 4-way associative
  - **L3** 12-way associative

Multicore caches

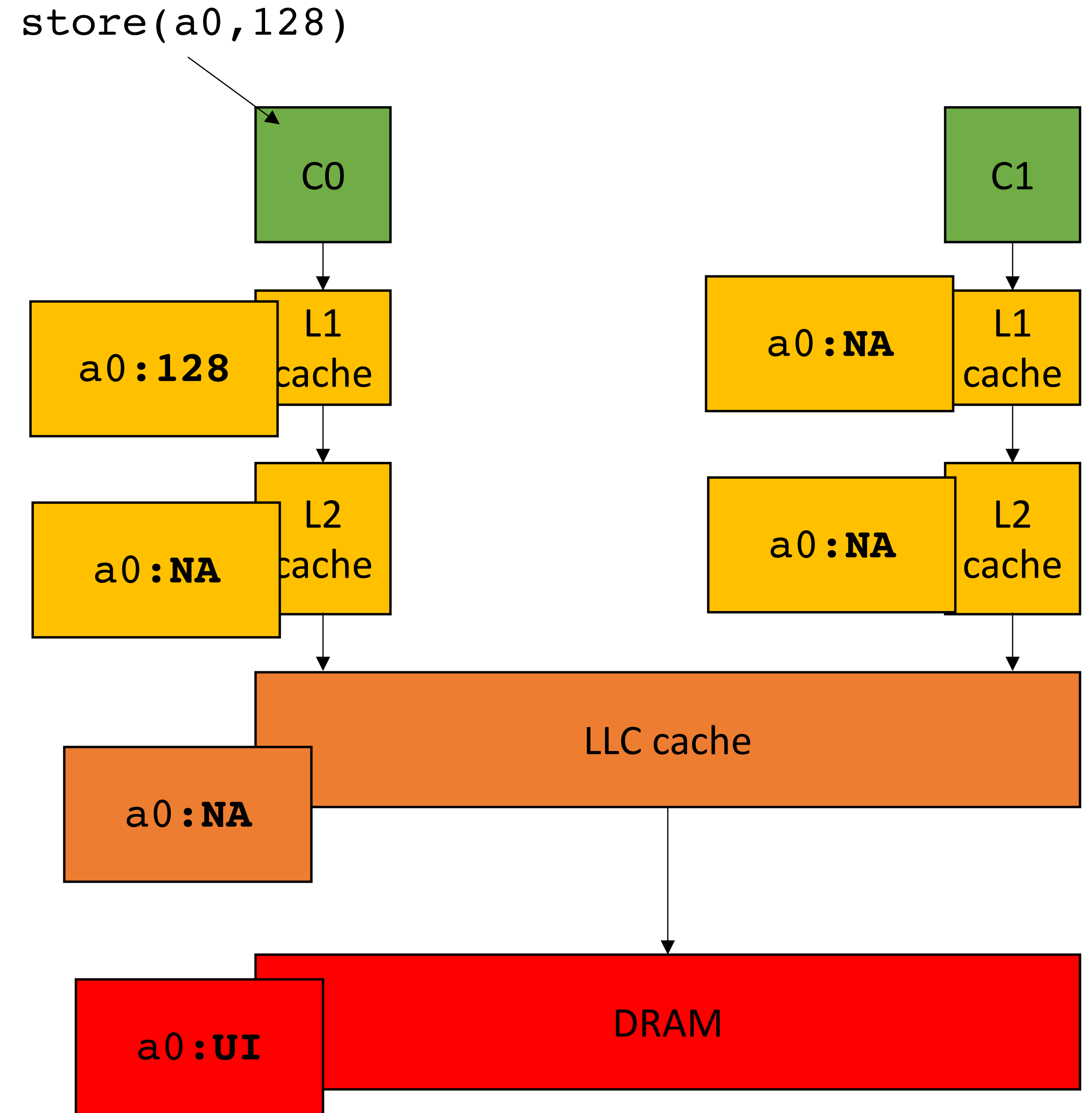


UI = uninitialized

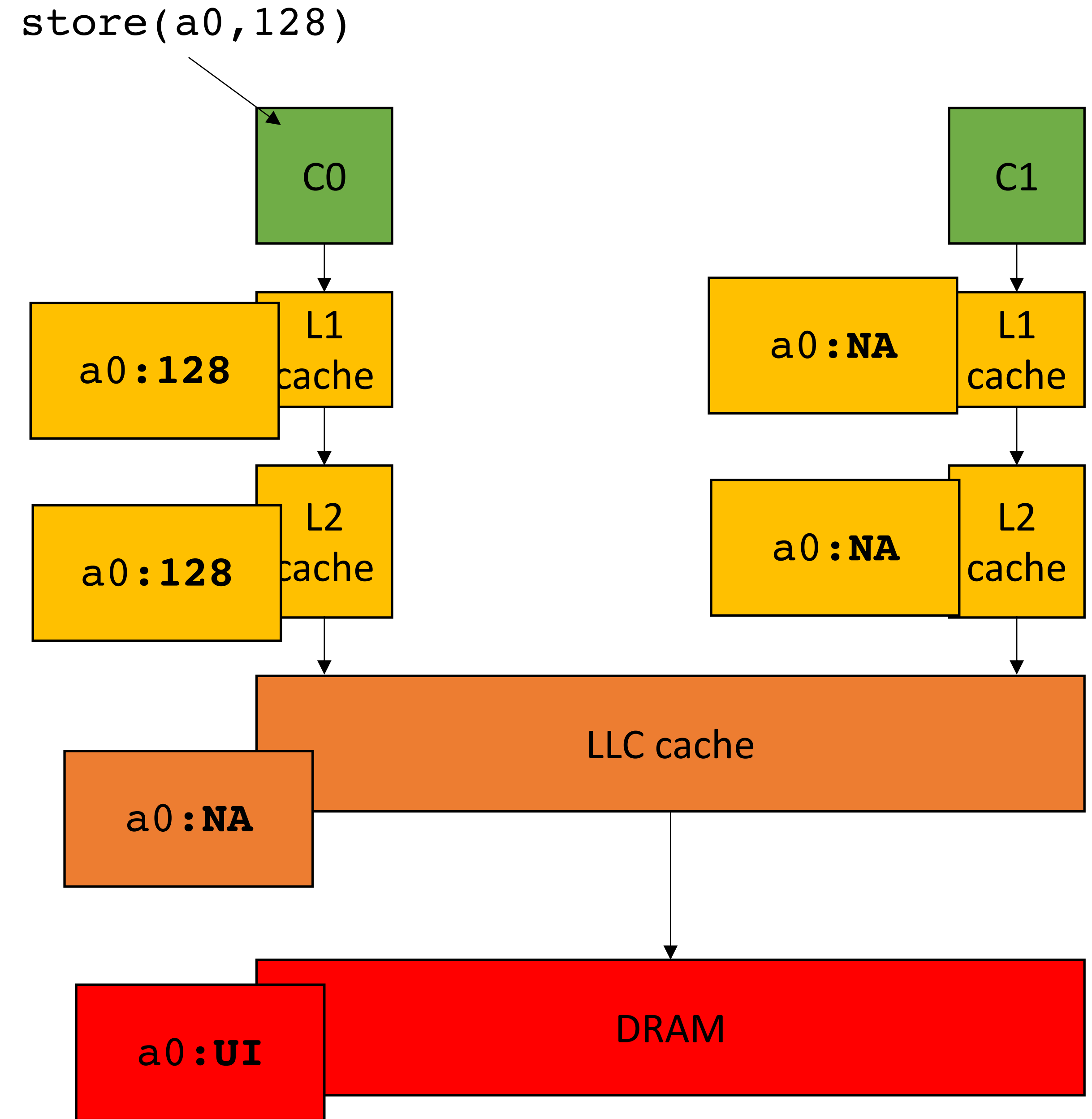
Multicore caches



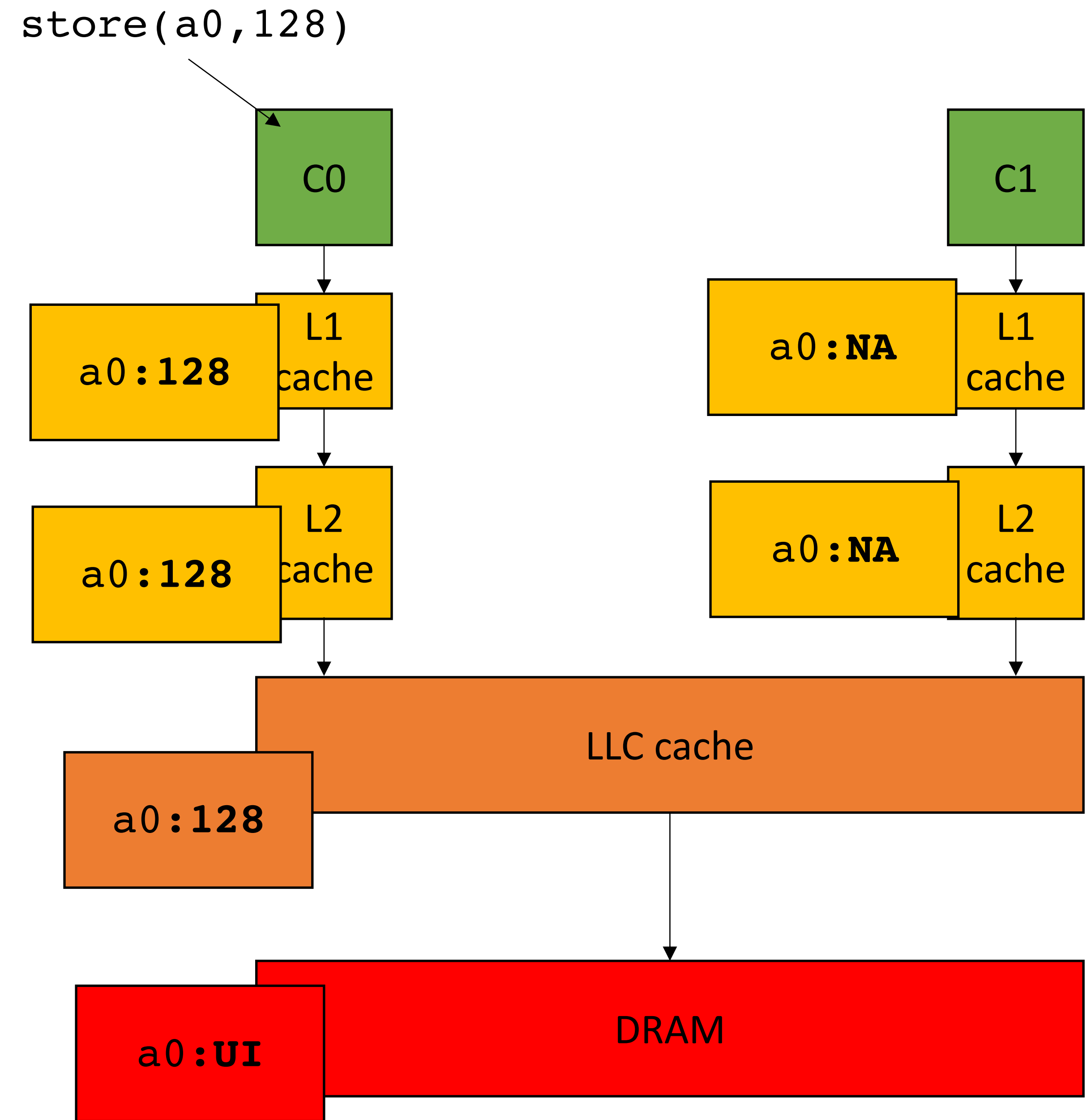
Multicore caches



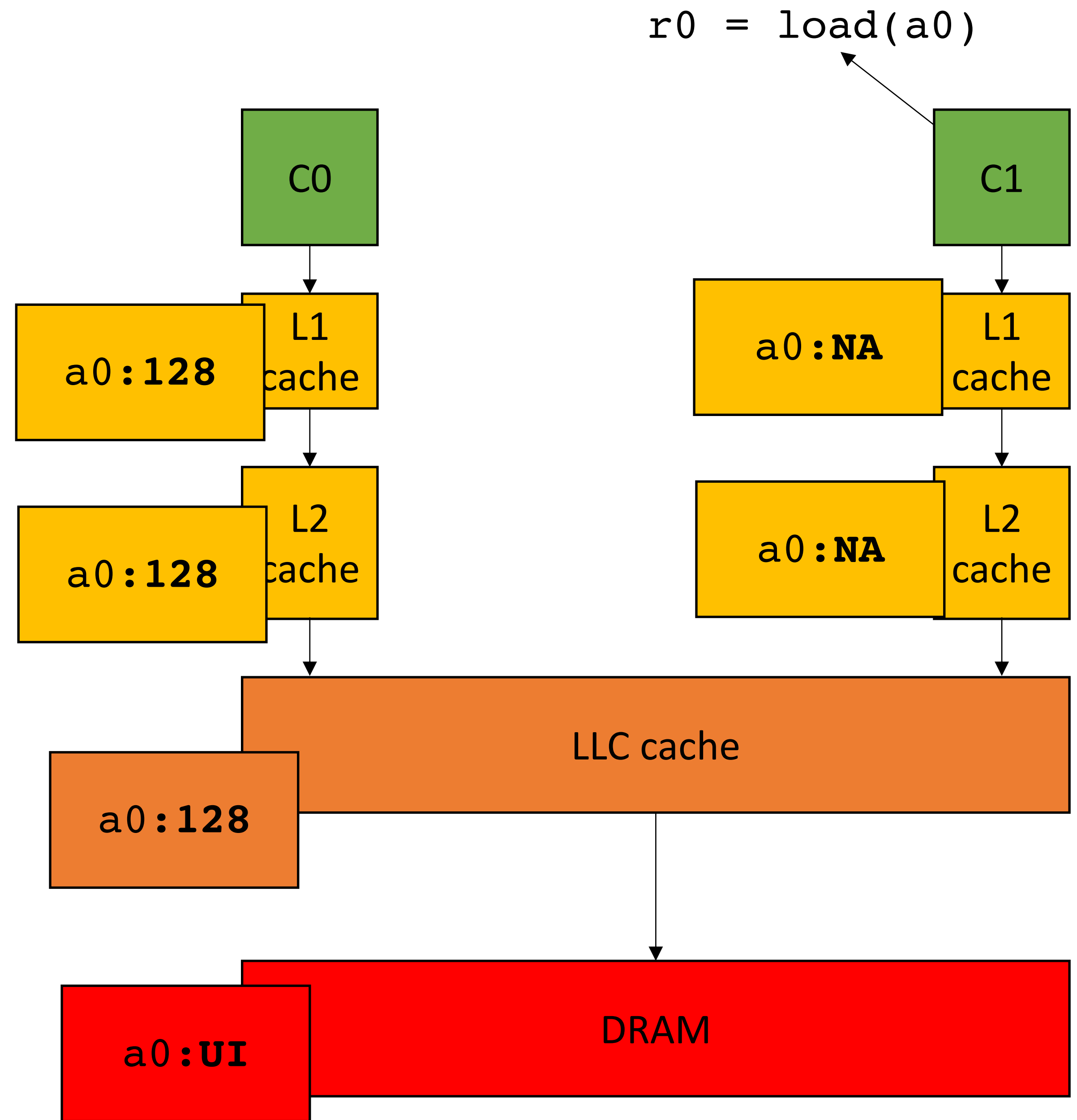
Multicore caches



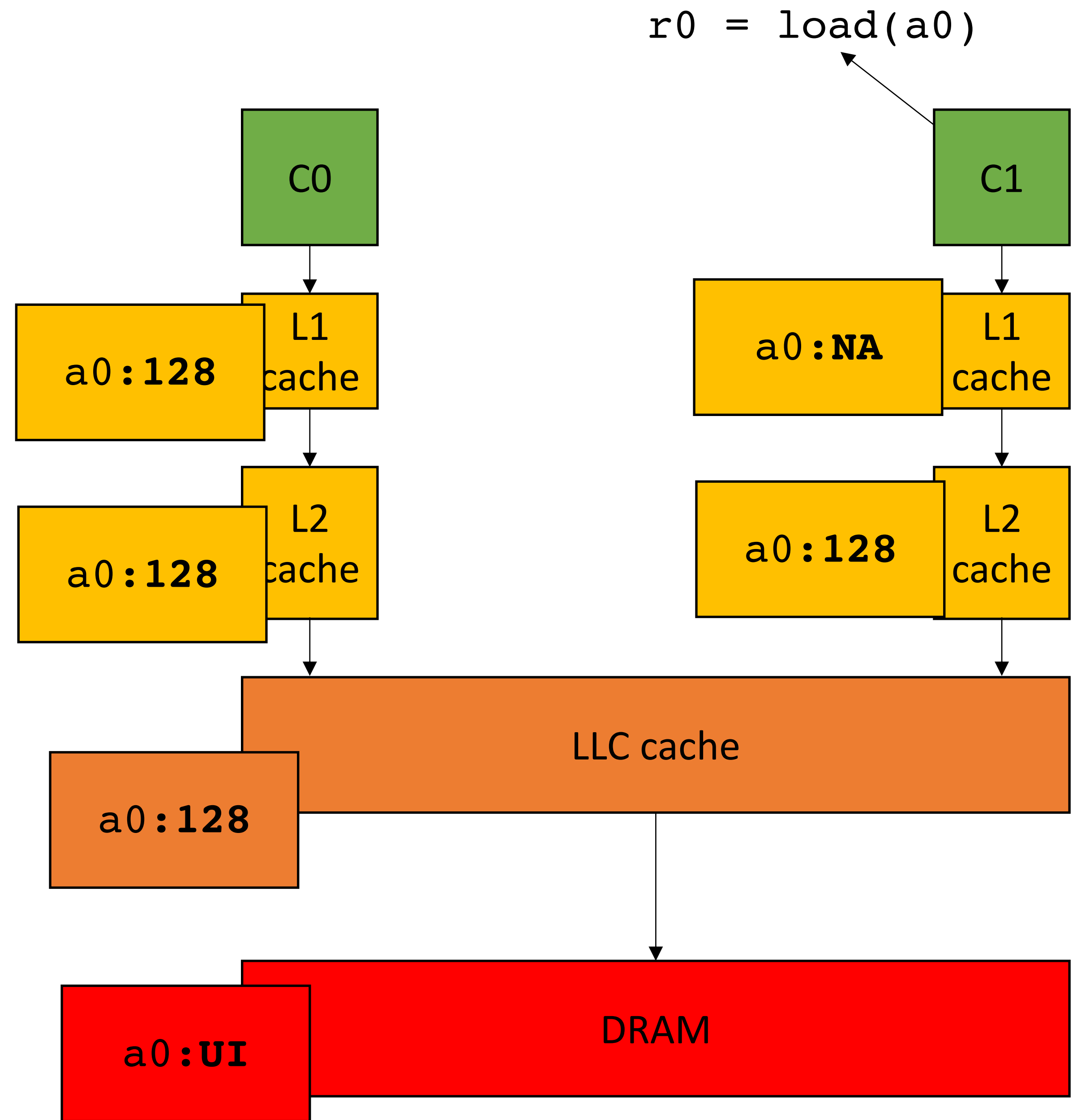
Multicore caches



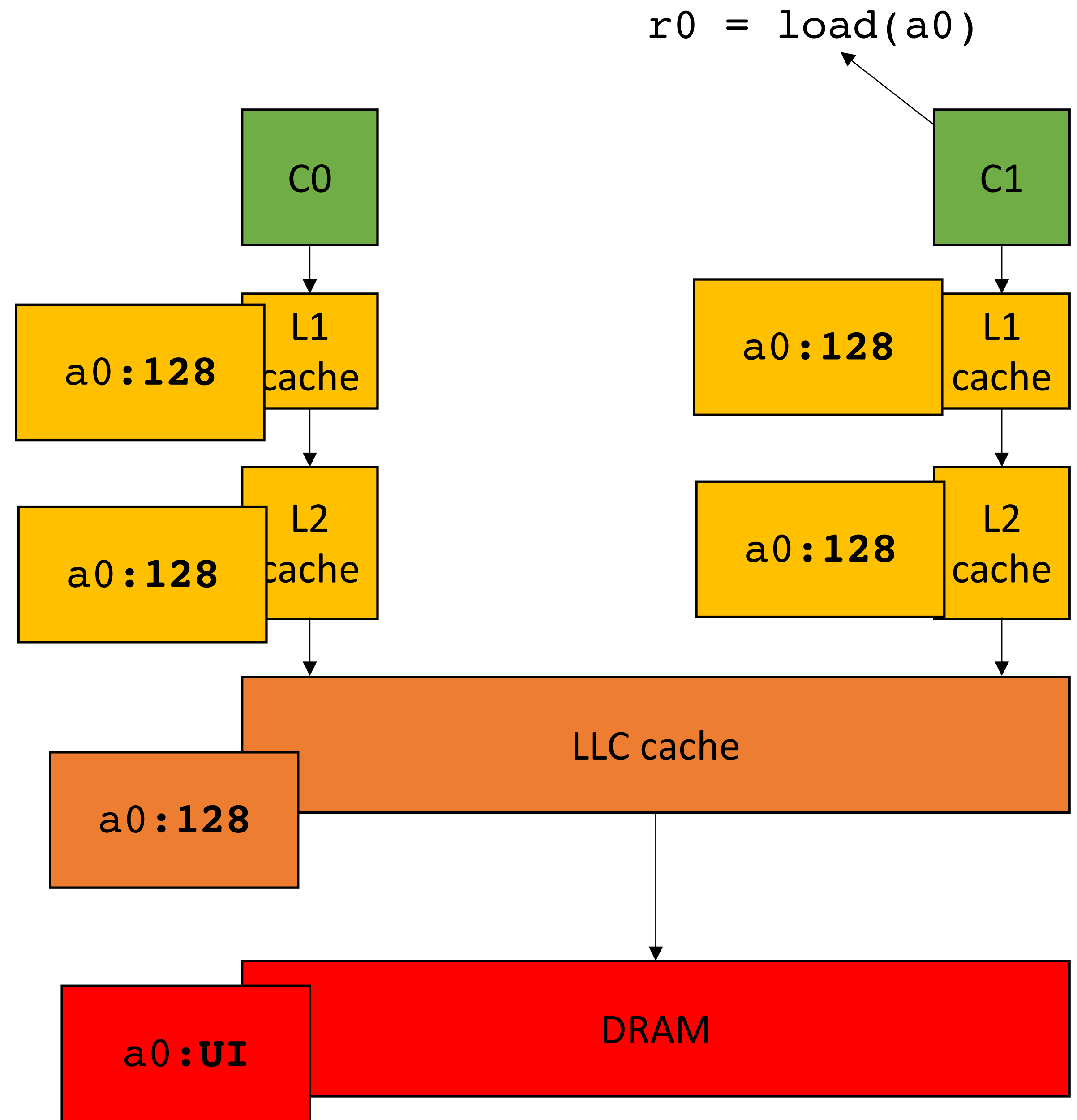
Multicore caches



Multicore caches

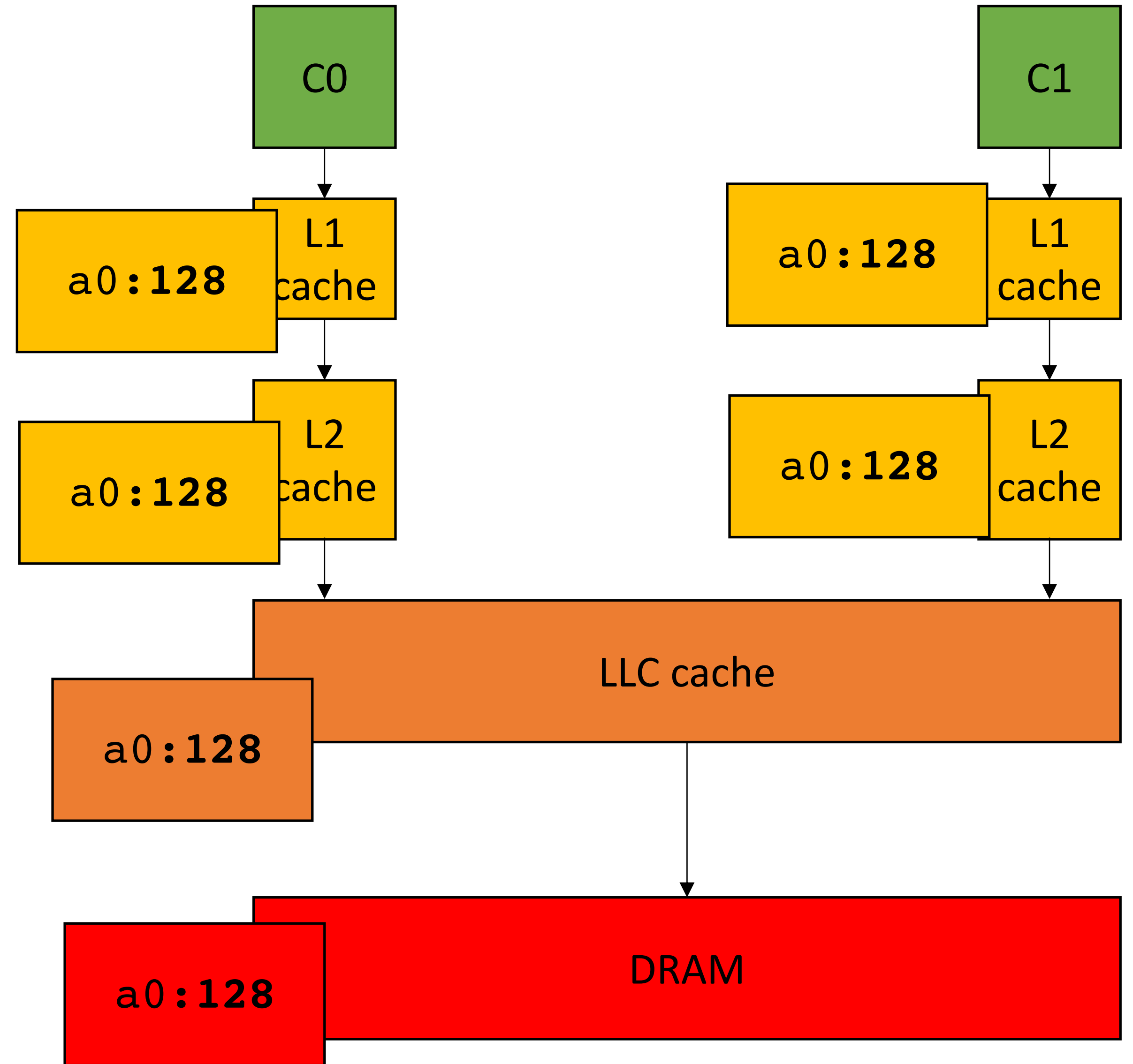


Multicore caches



C1 can read values before they are propagated into DRAM

Multicore caches

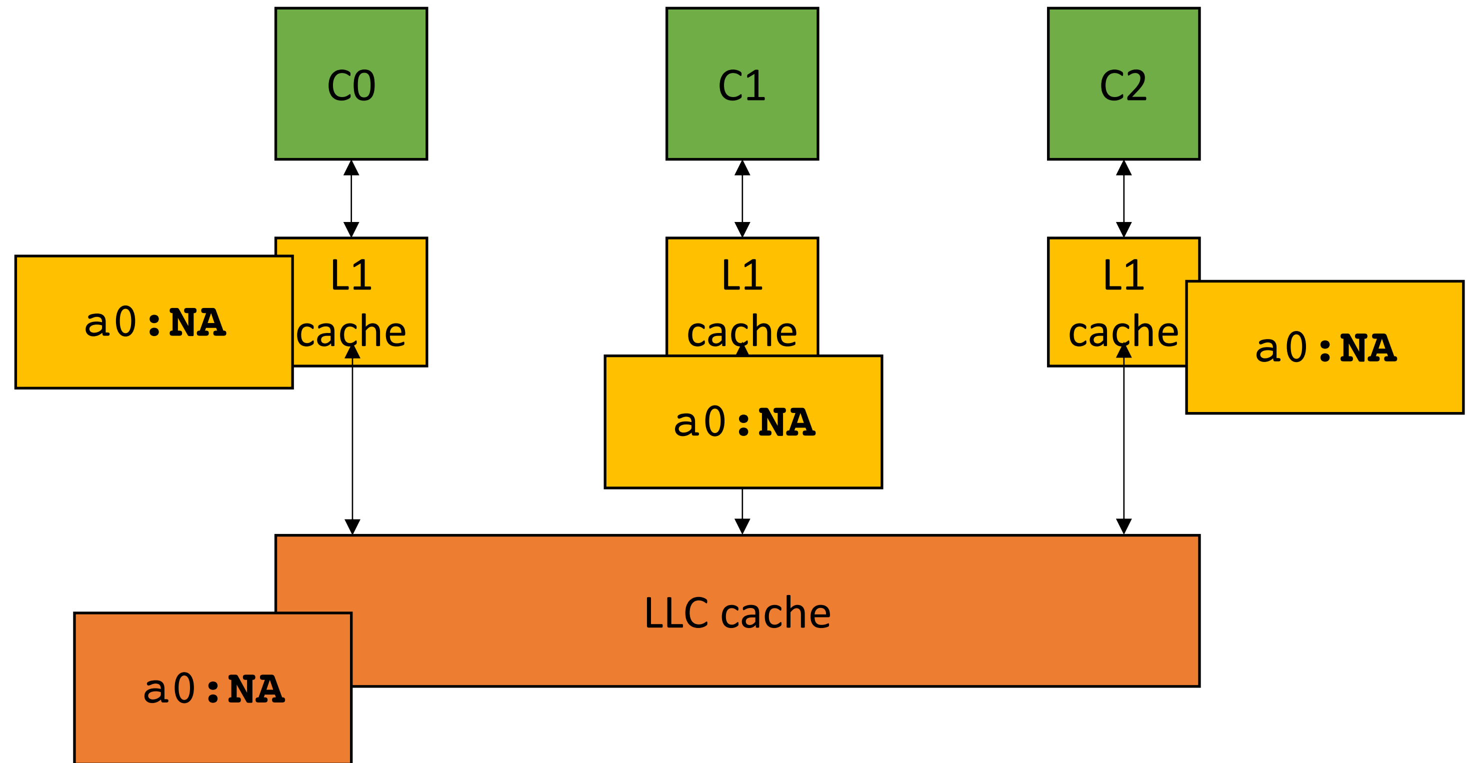


Cache coherence

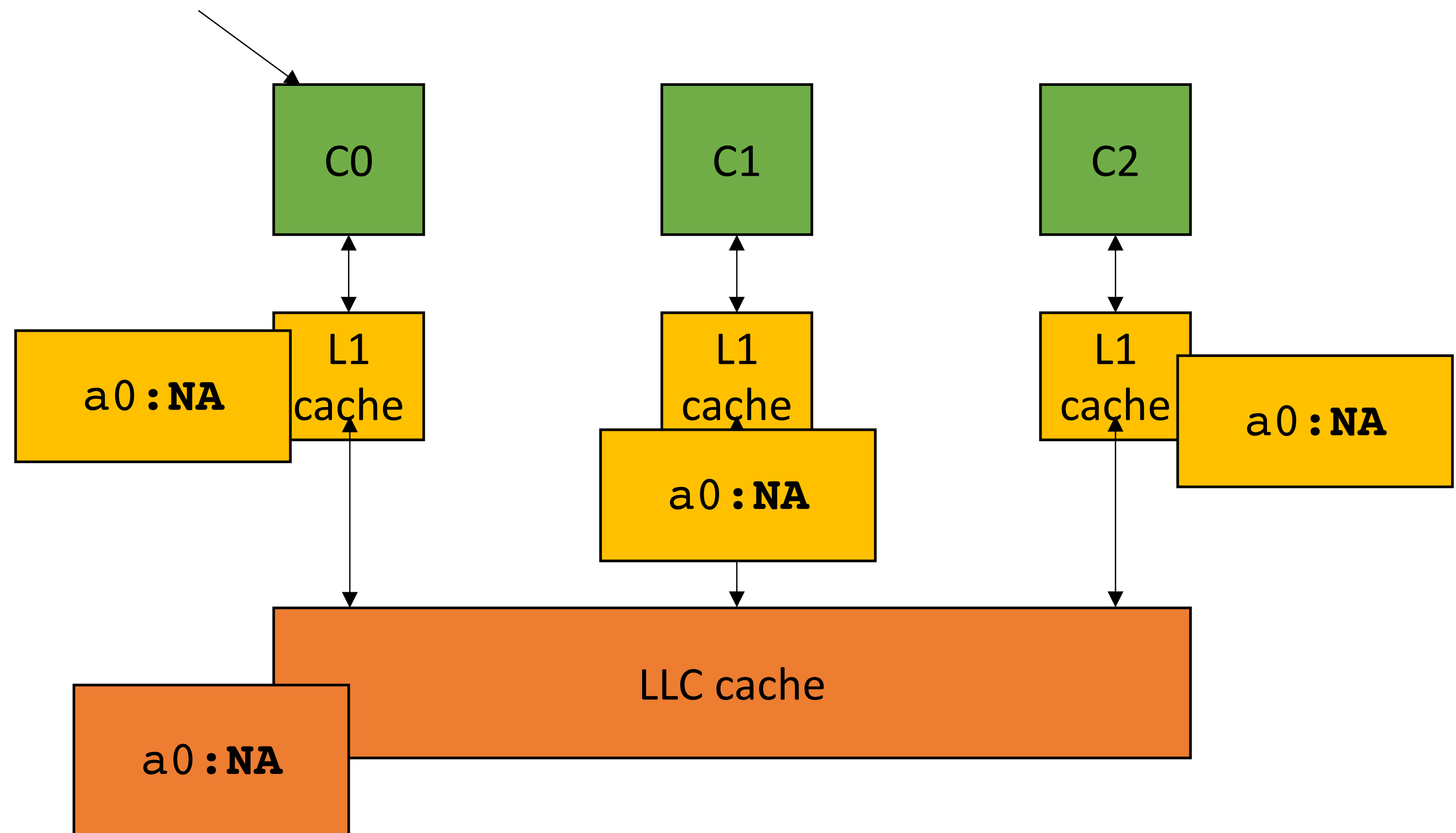
How to manage multiple values for the same address in the system?

simplified view for illustration: L1 cache and LLC

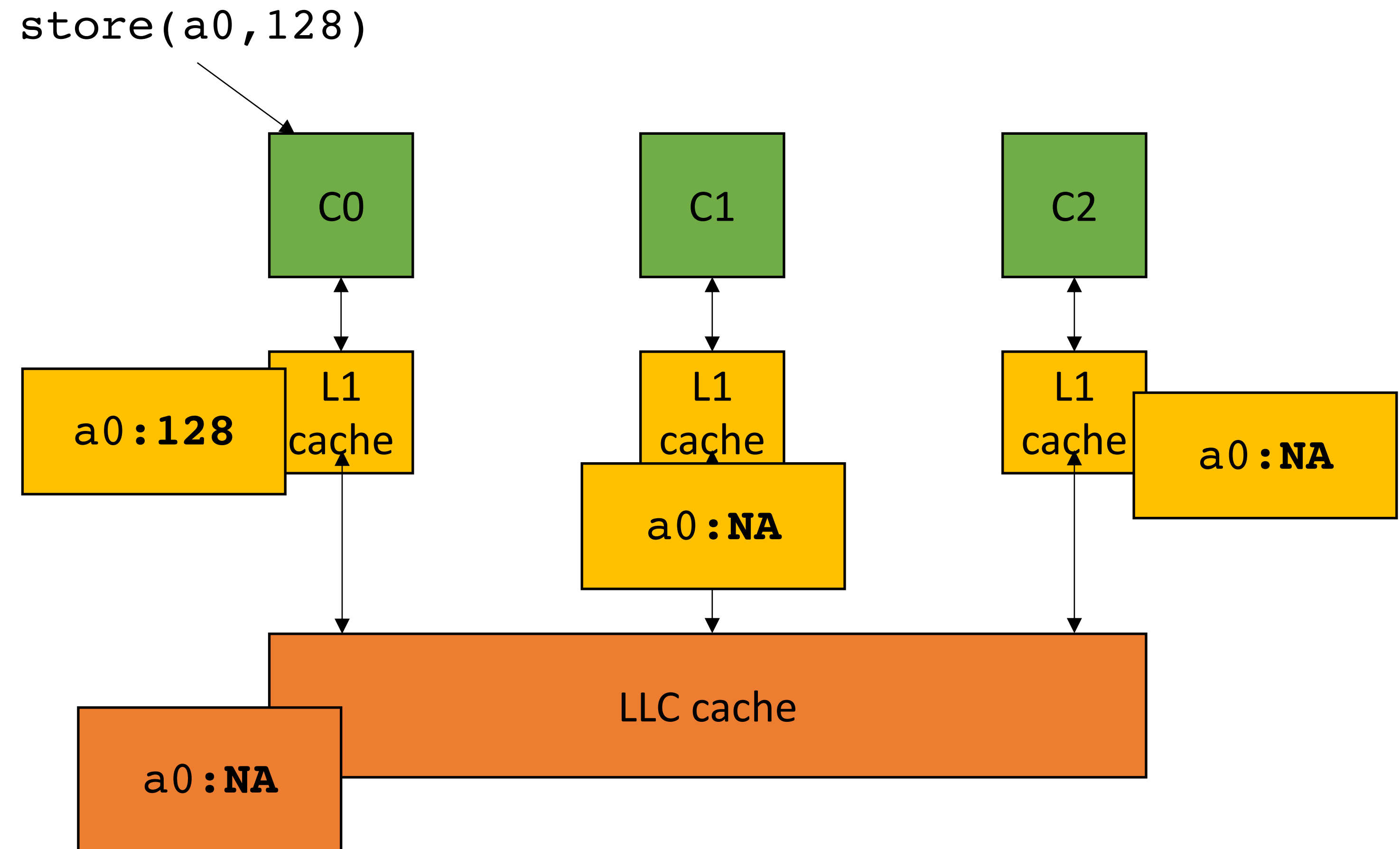
Consider 3 cores accessing the same memory location



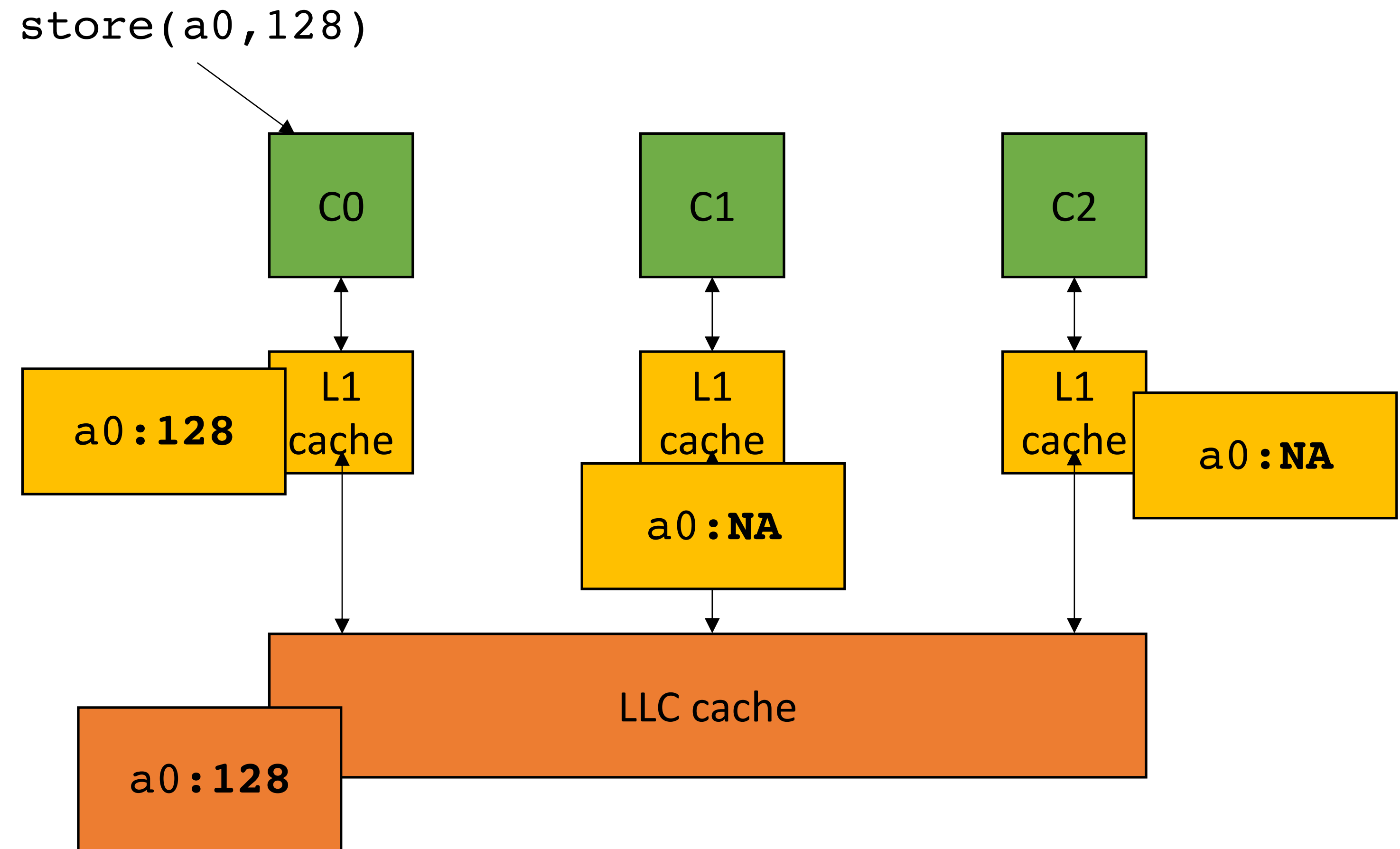
Cache coherence



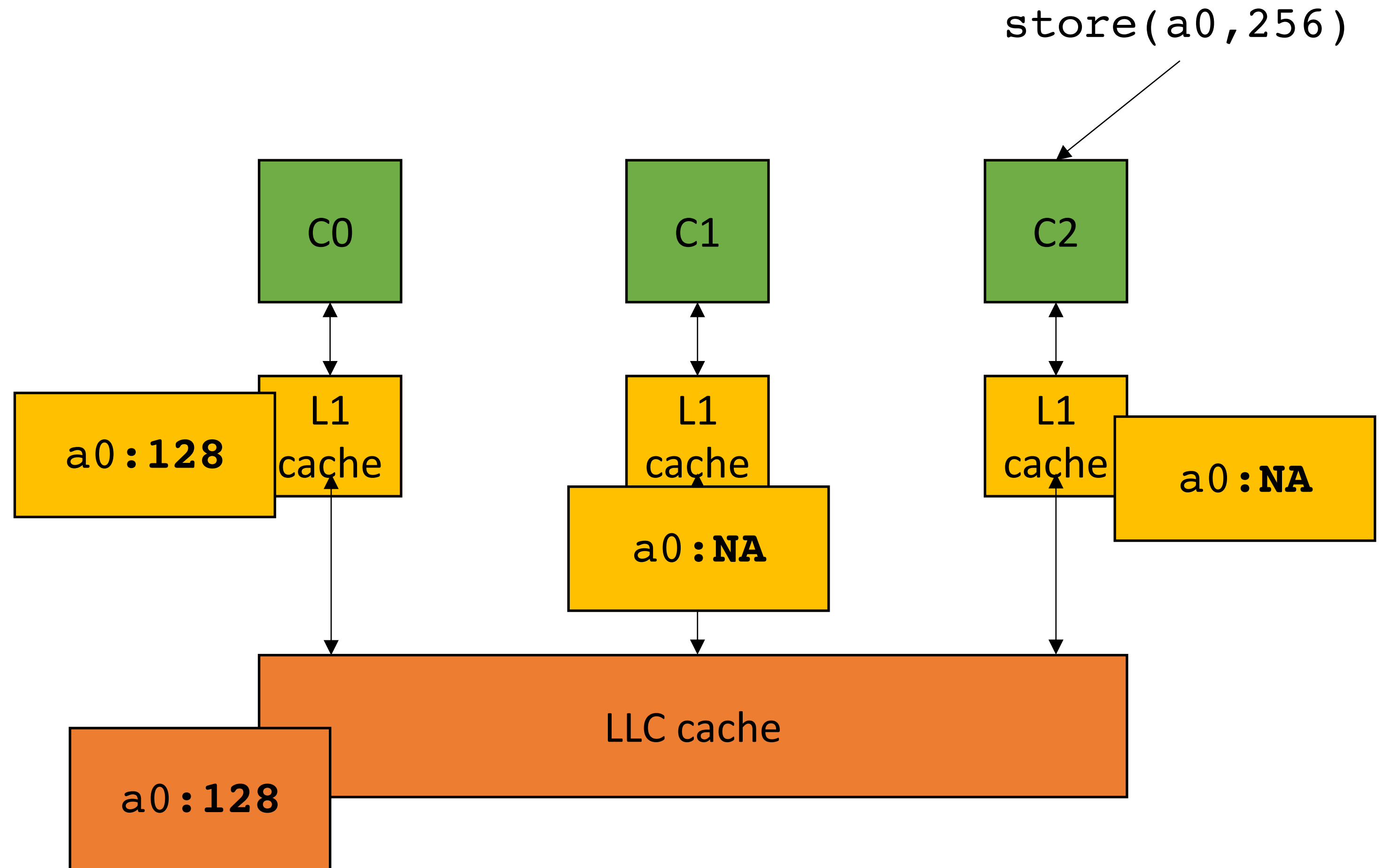
Cache coherence



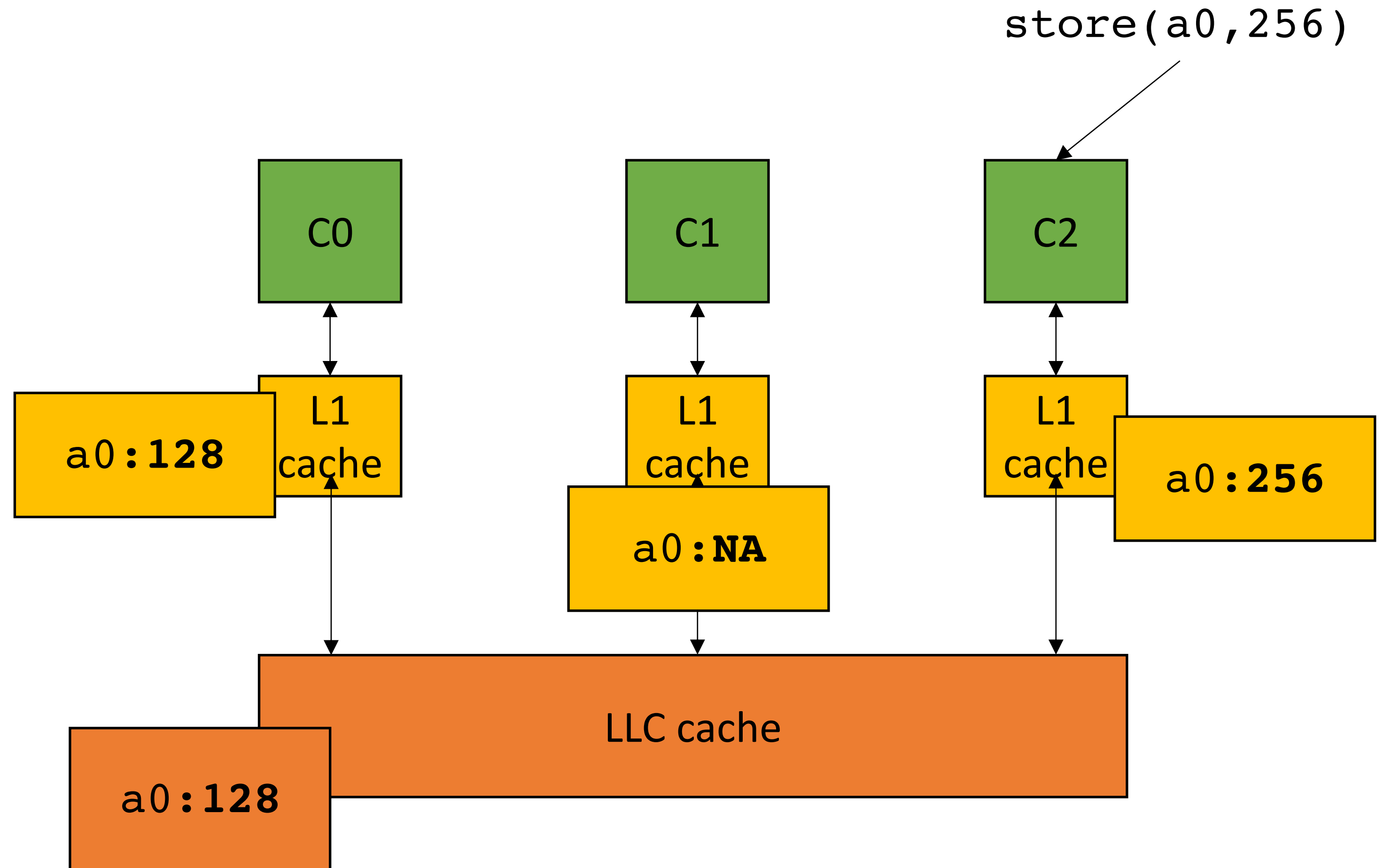
Cache coherence



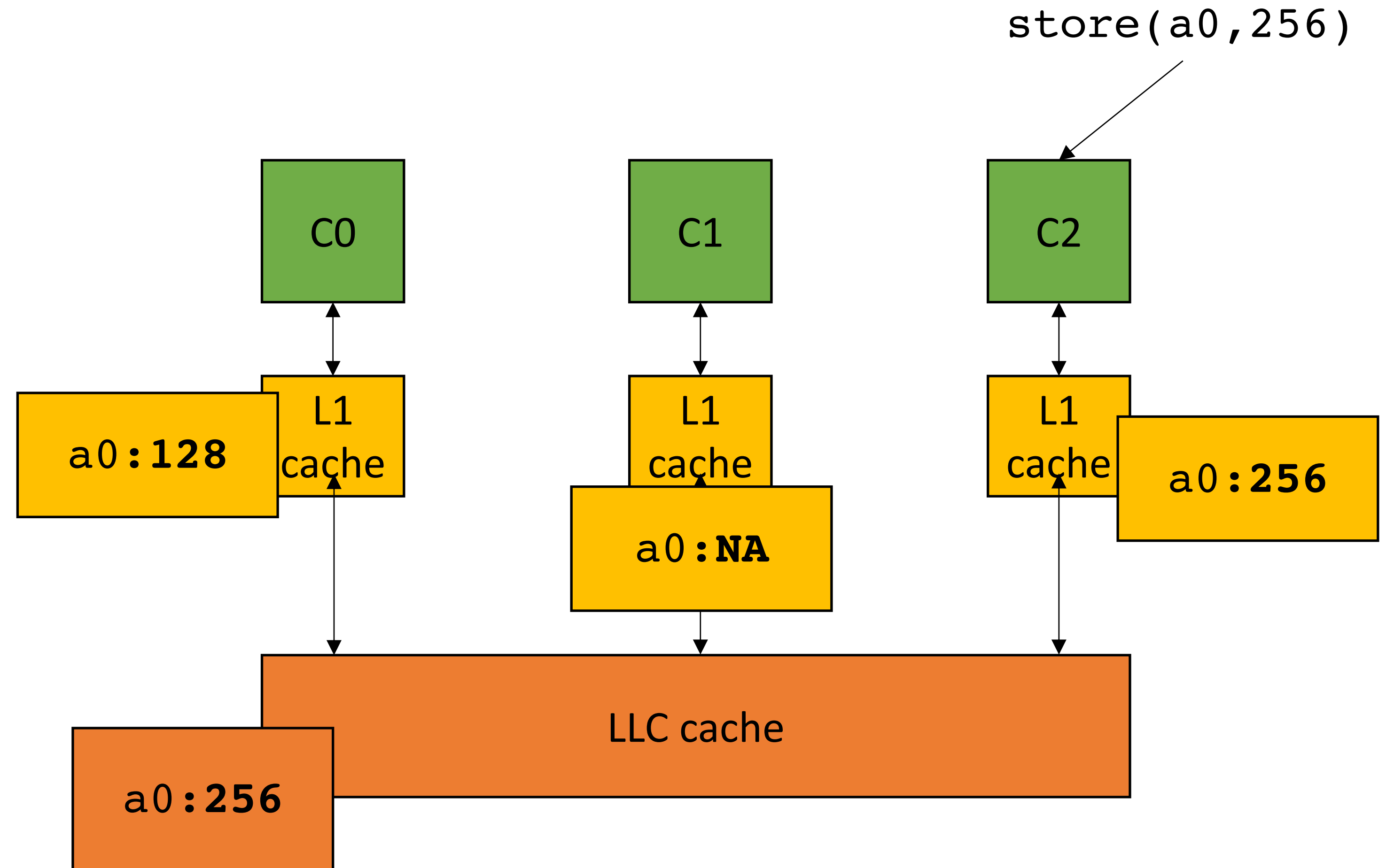
Cache coherence



Cache coherence

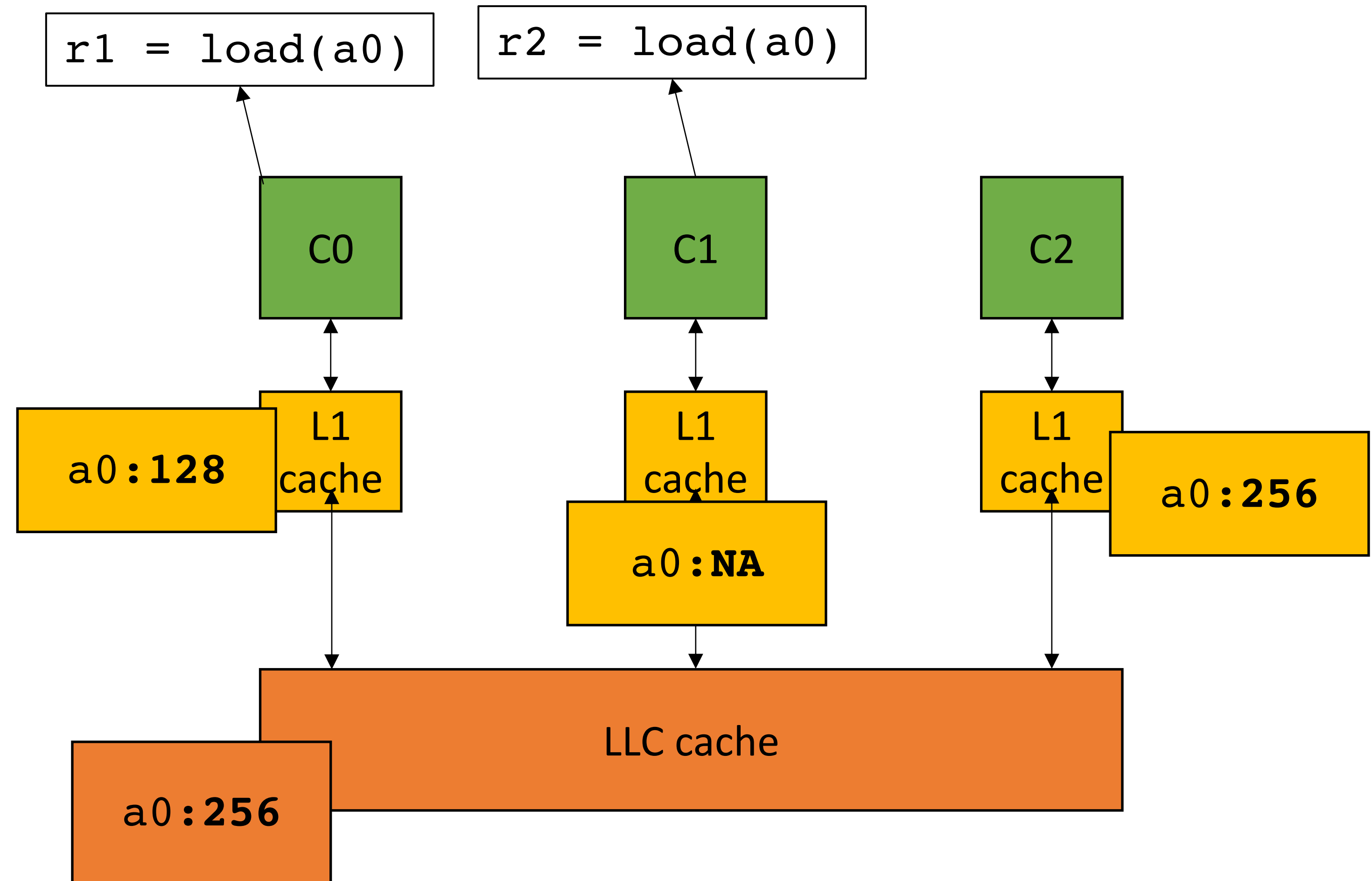


Cache coherence

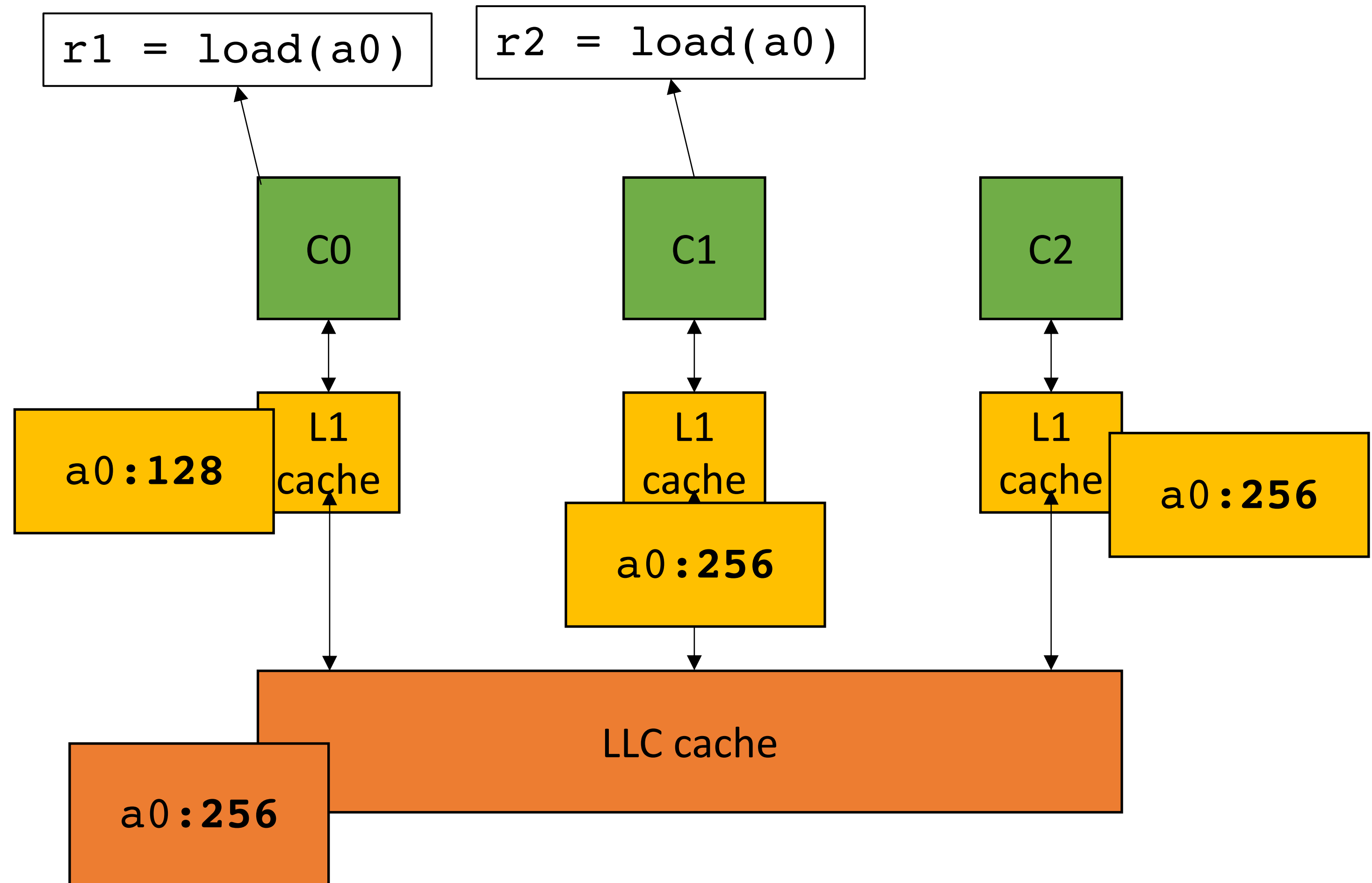


Cache coherence

in parallel

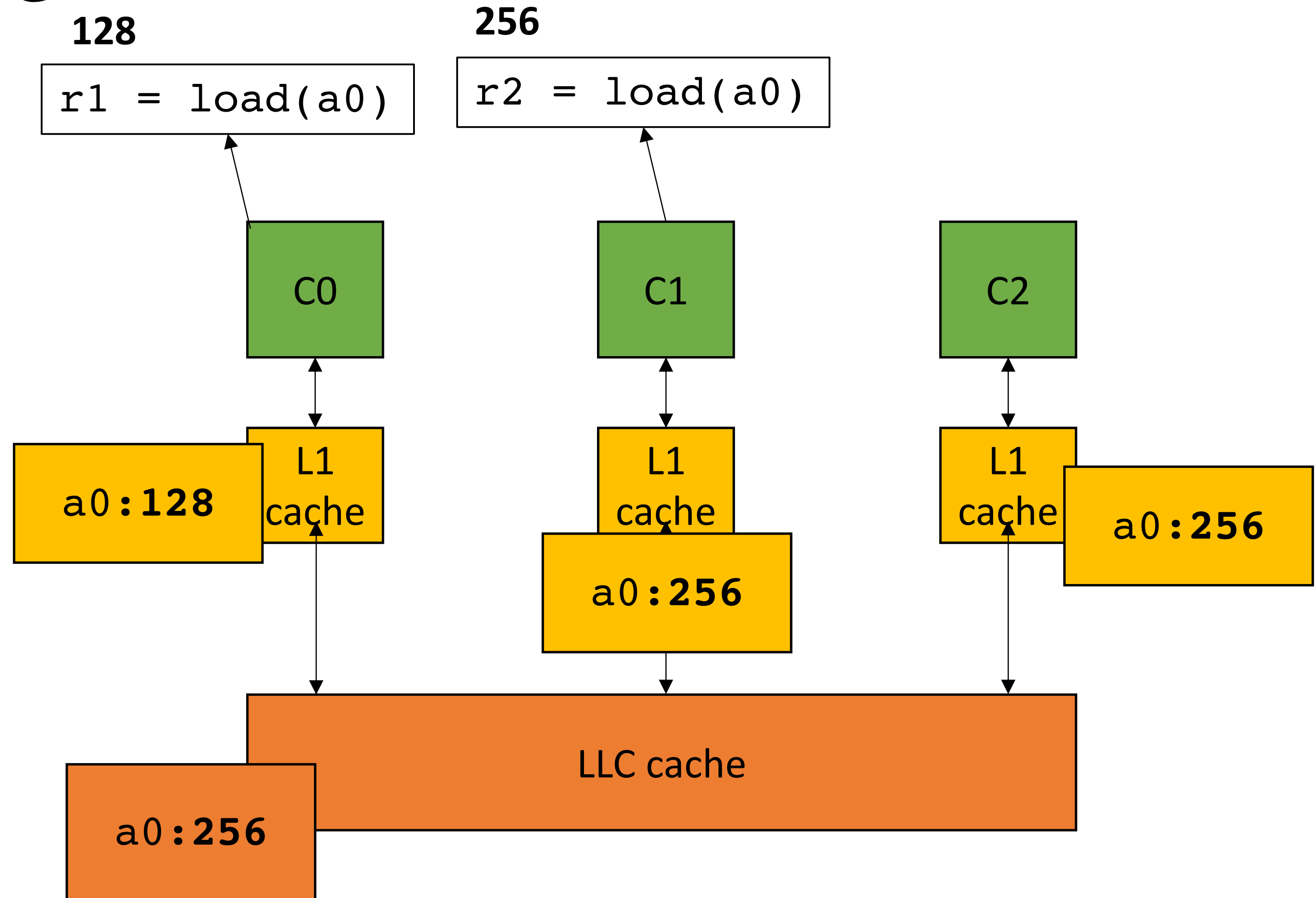


Cache coherence



Cache coherence

Incoherent view of values!



Cache coherence

- MESI protocol

- Cache line can be in 1 of 4 states:

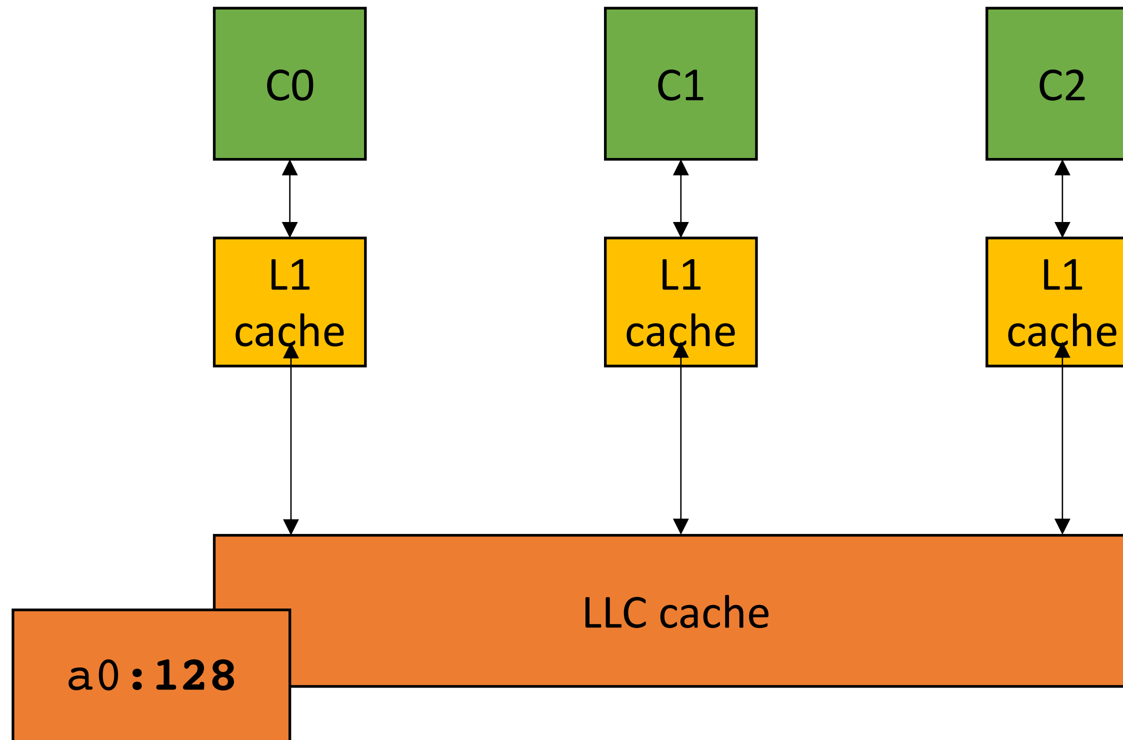
  - **Modified** - the cache contains a modified value and it must be written back to the lower level cache

  - **Exclusive** - only 1 cache has a copy of the value

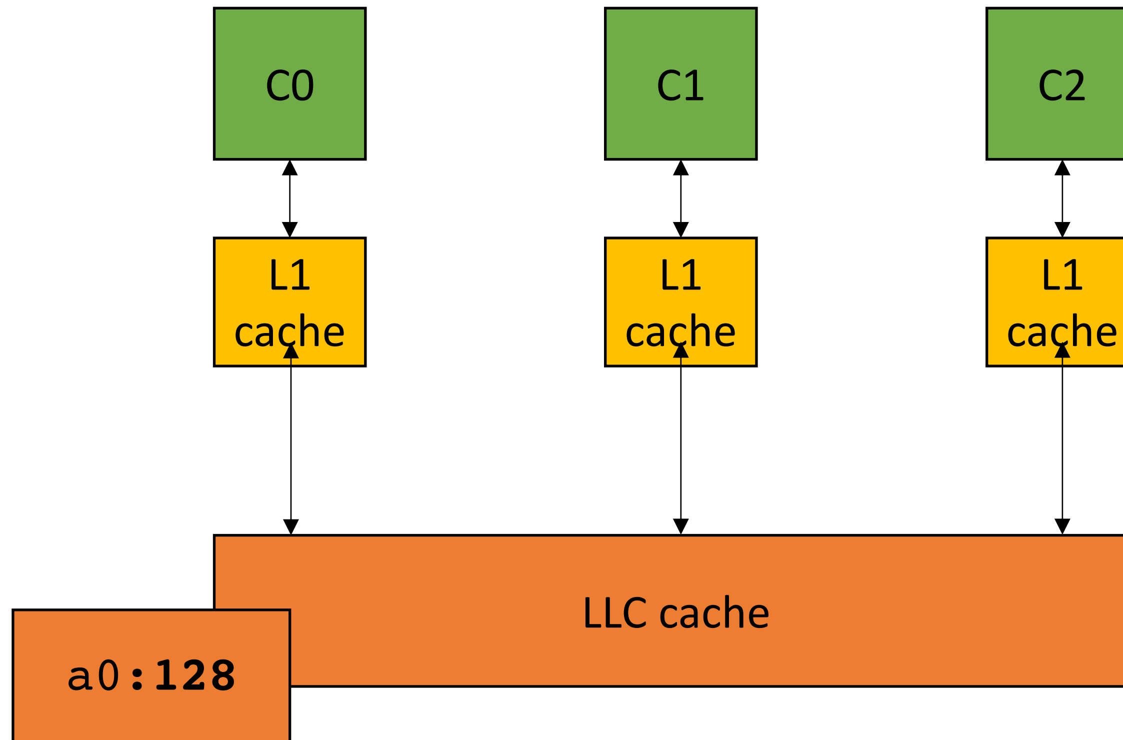
  - **Shared** - more than 1 cache contains the value, they must all agree on the value

  - **Invalid** - the data is stale and a new value must be fetched from a lower level cache

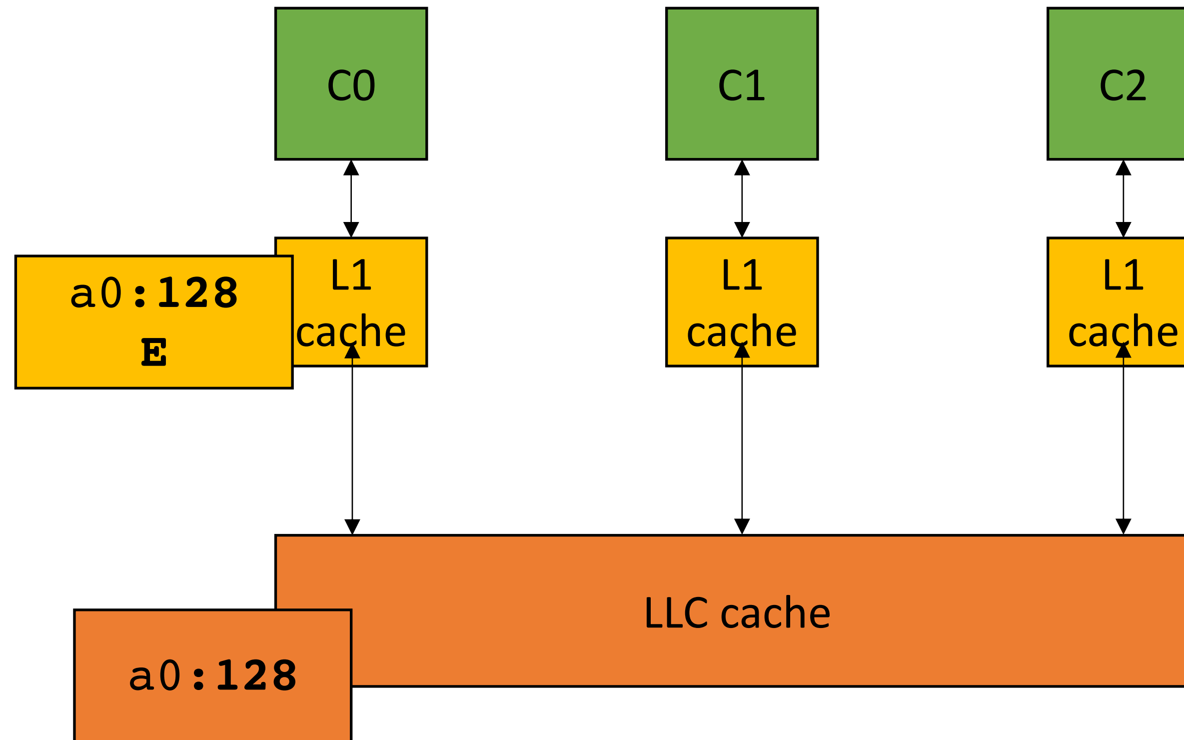
Cache coherence



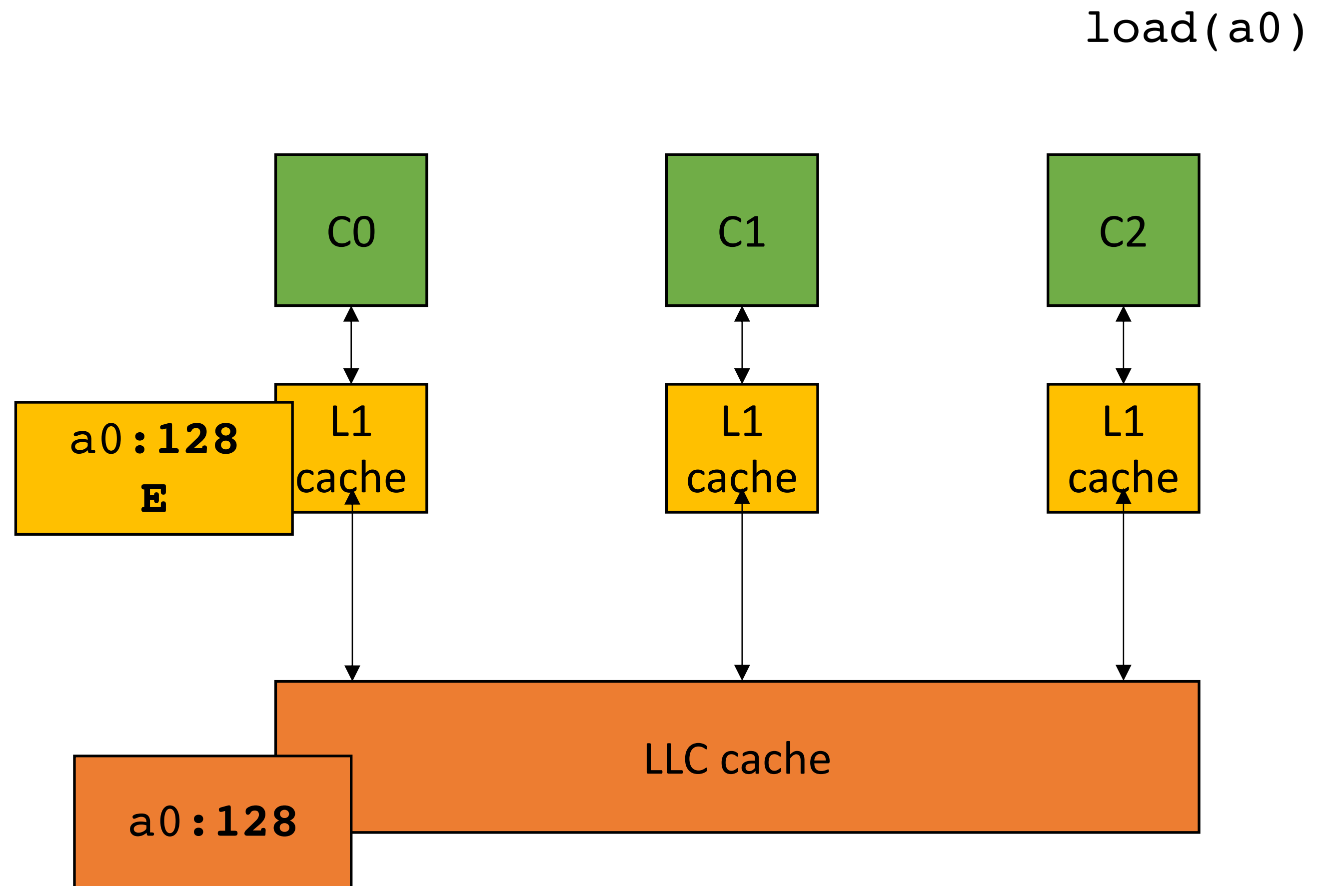
Cache coherence



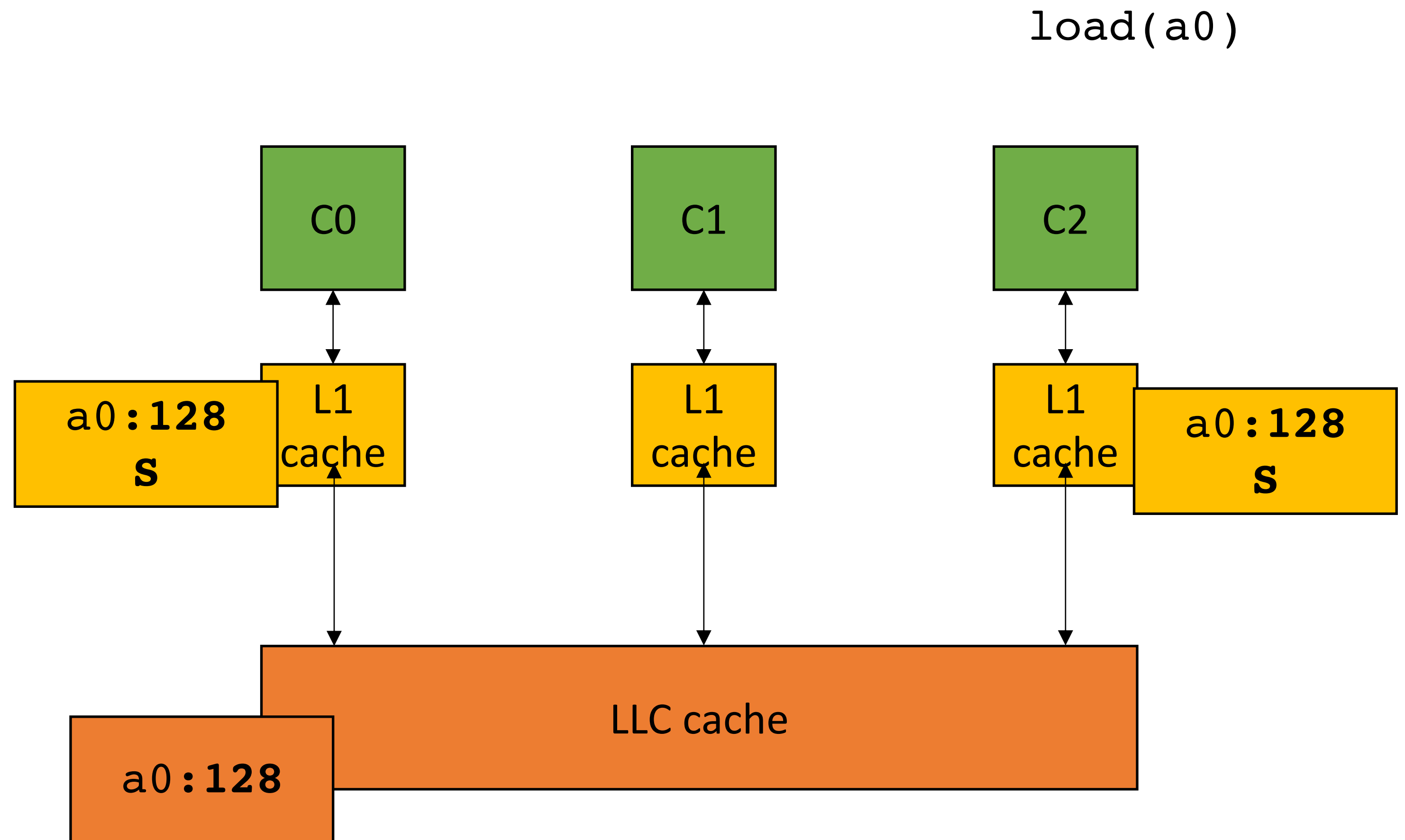
Cache coherence



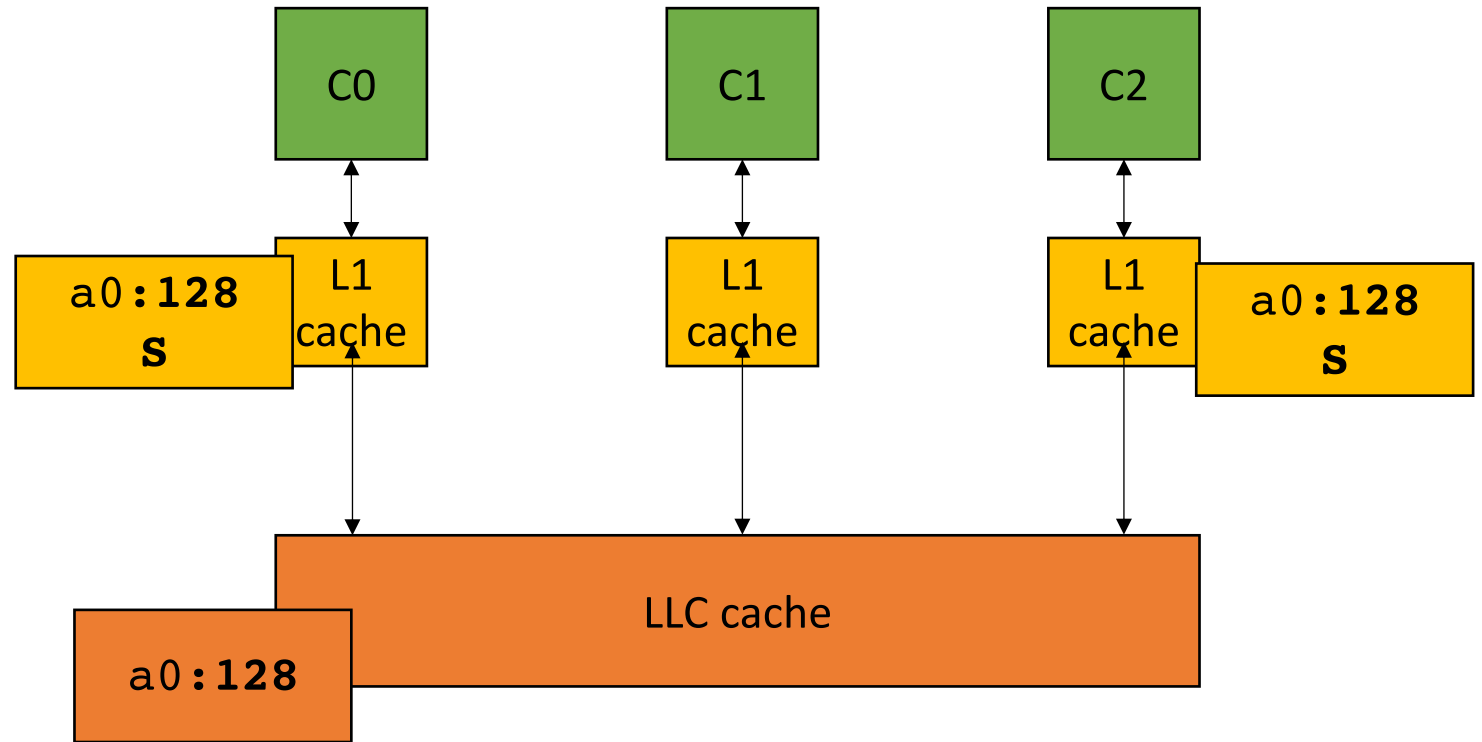
Cache coherence



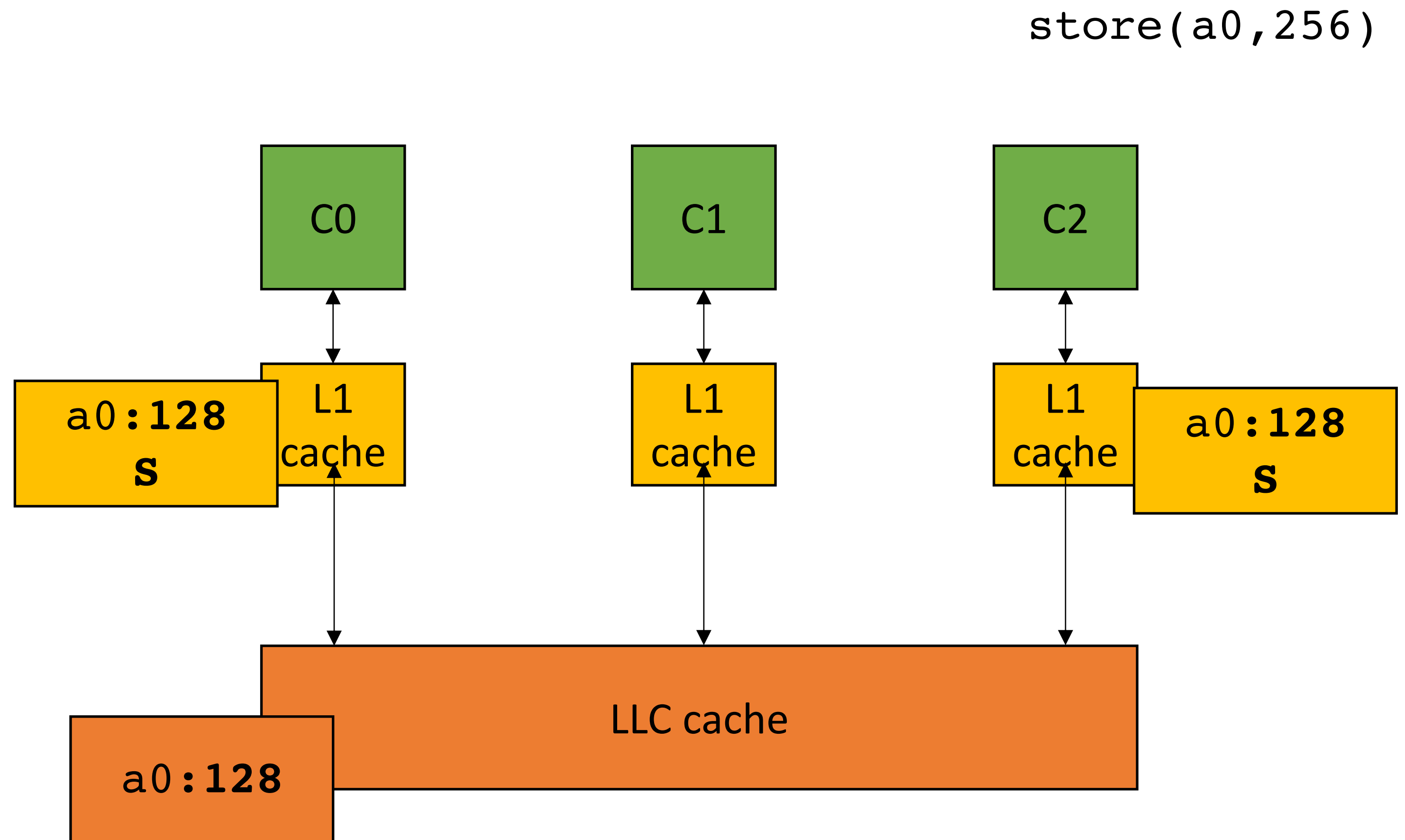
Cache coherence



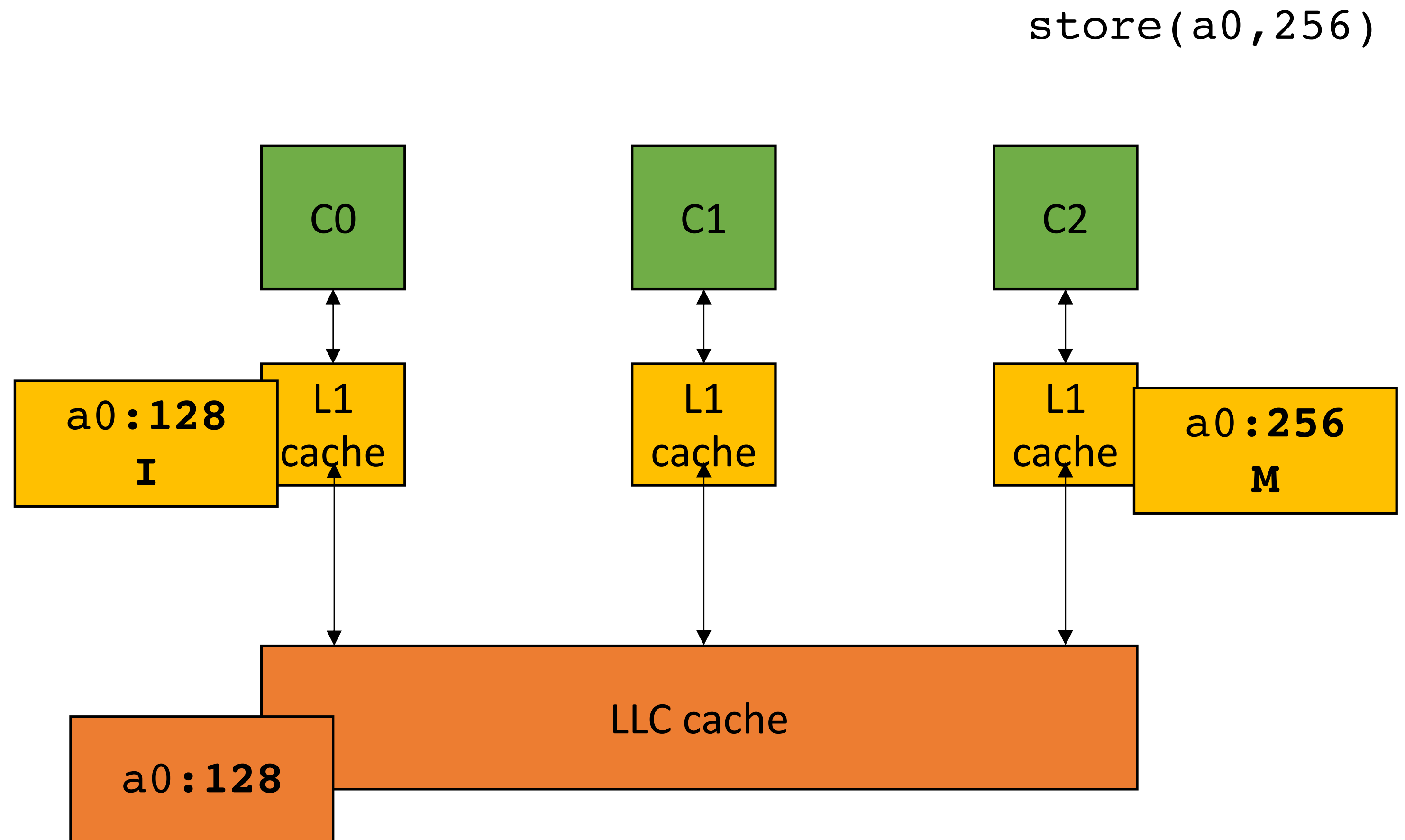
Cache coherence



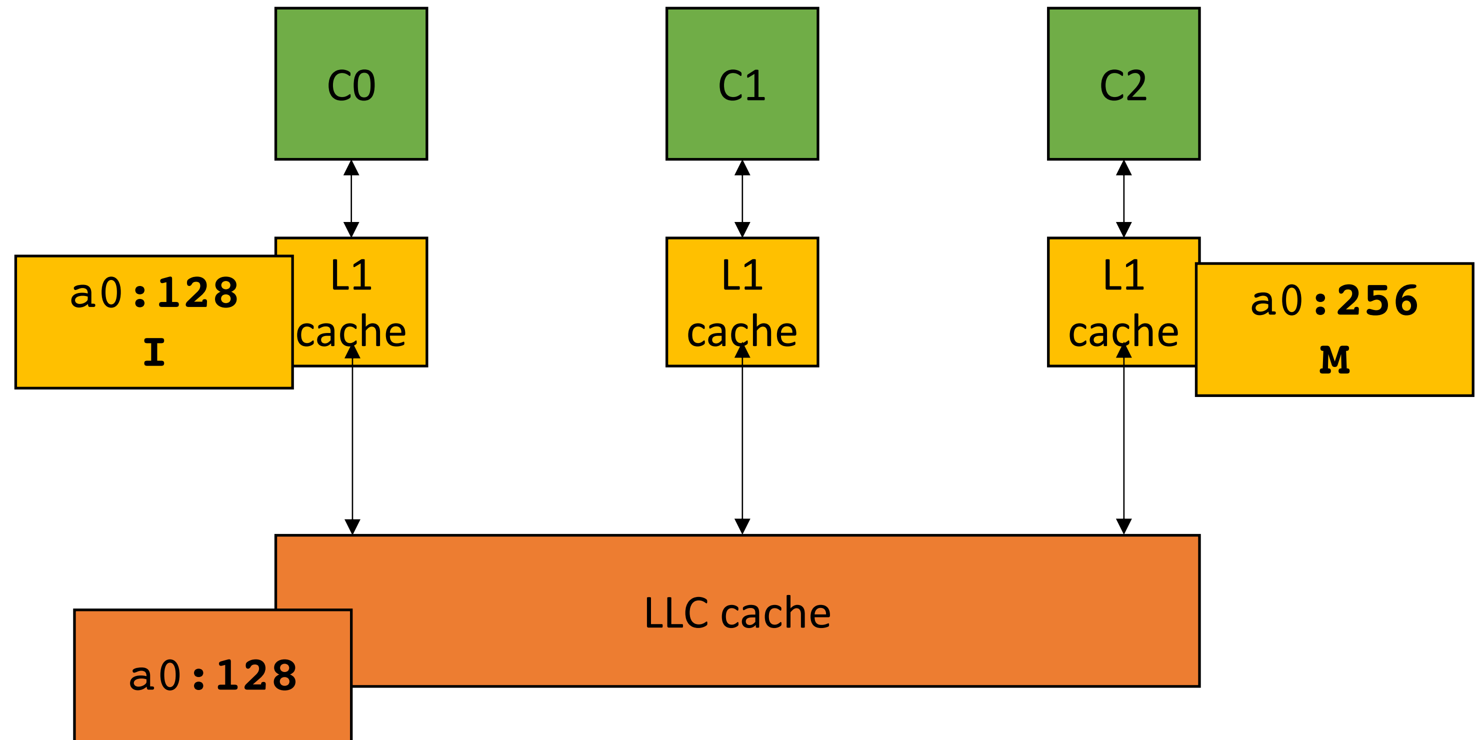
Cache coherence



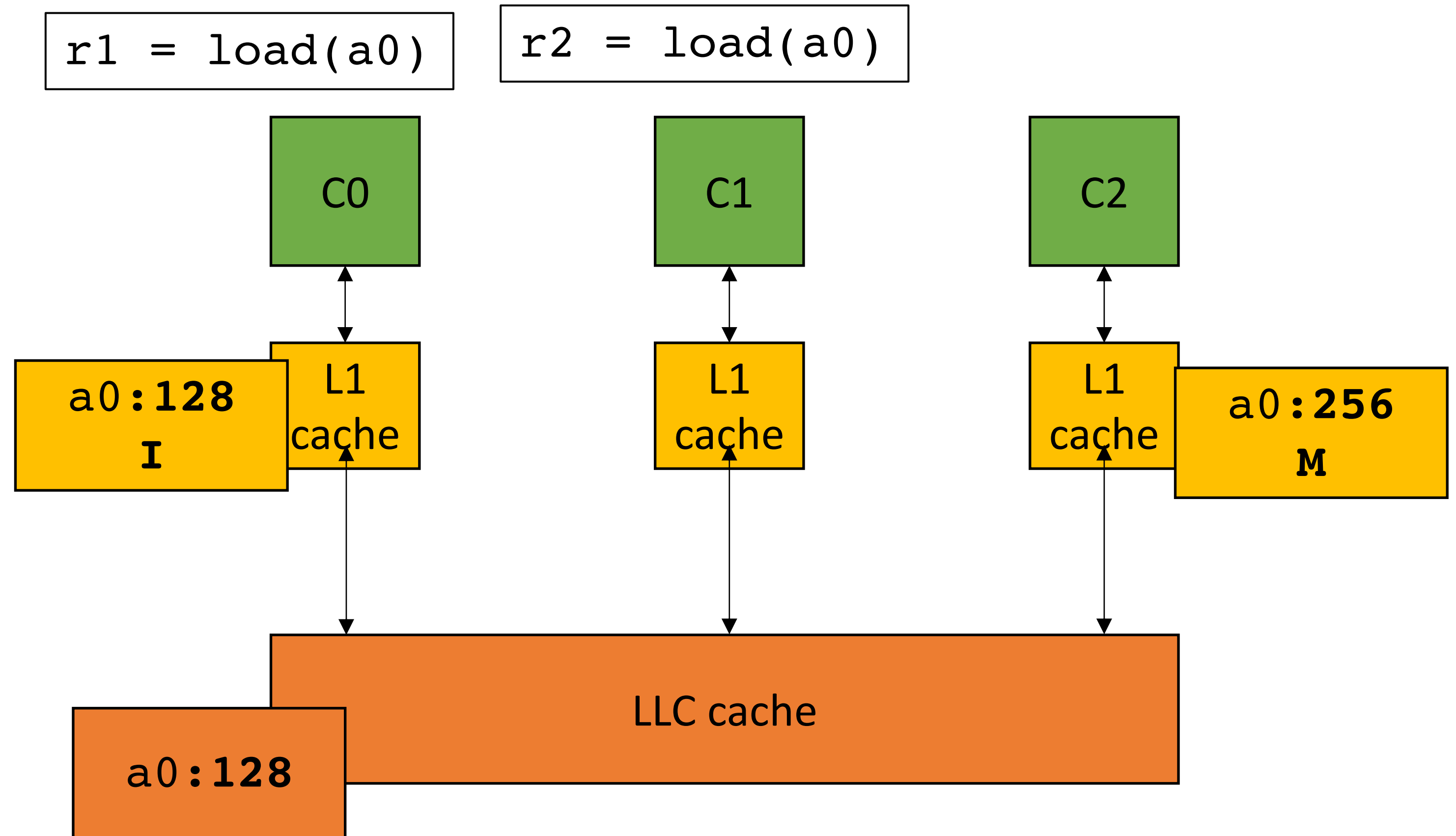
Cache coherence



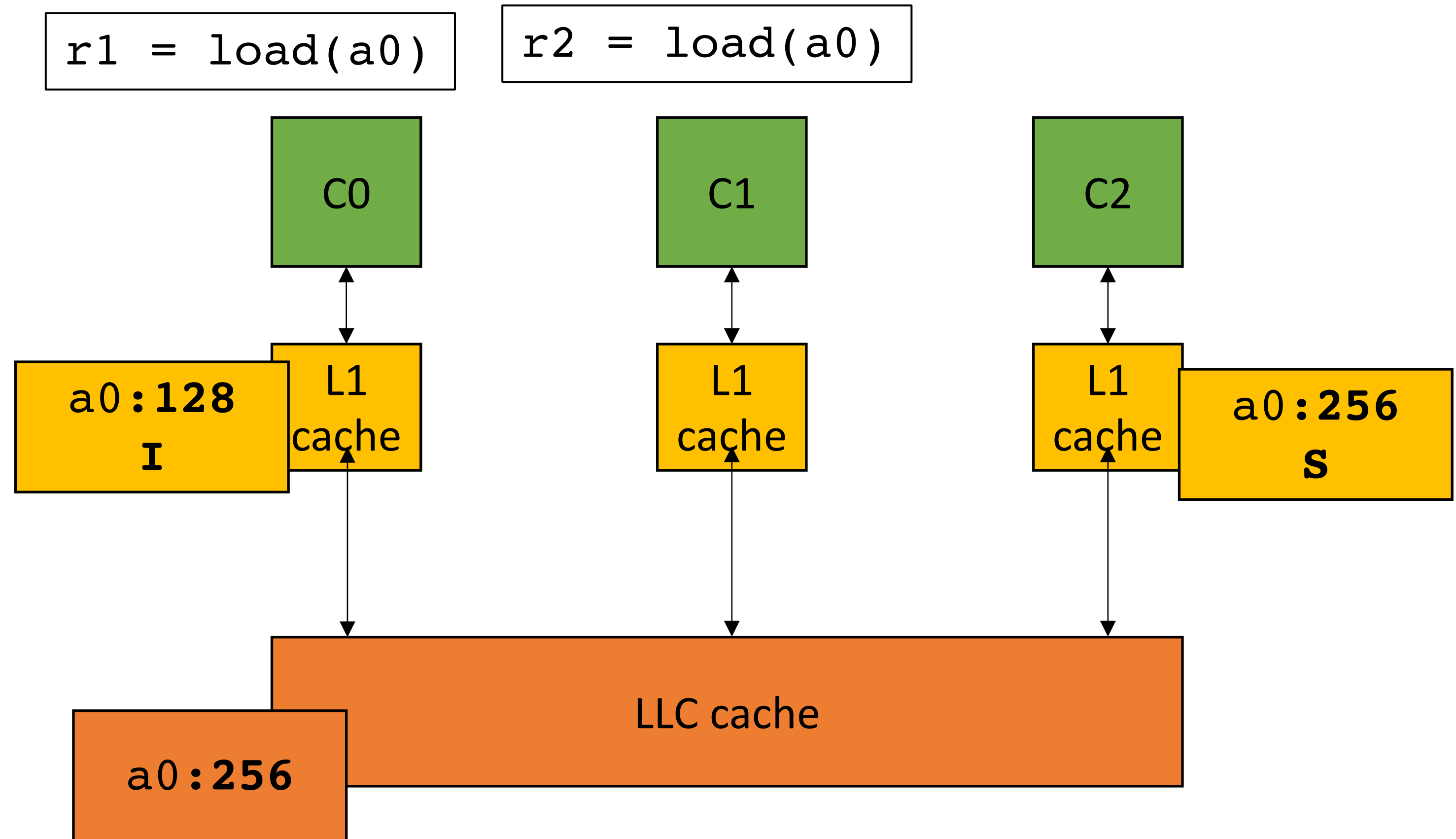
Cache coherence



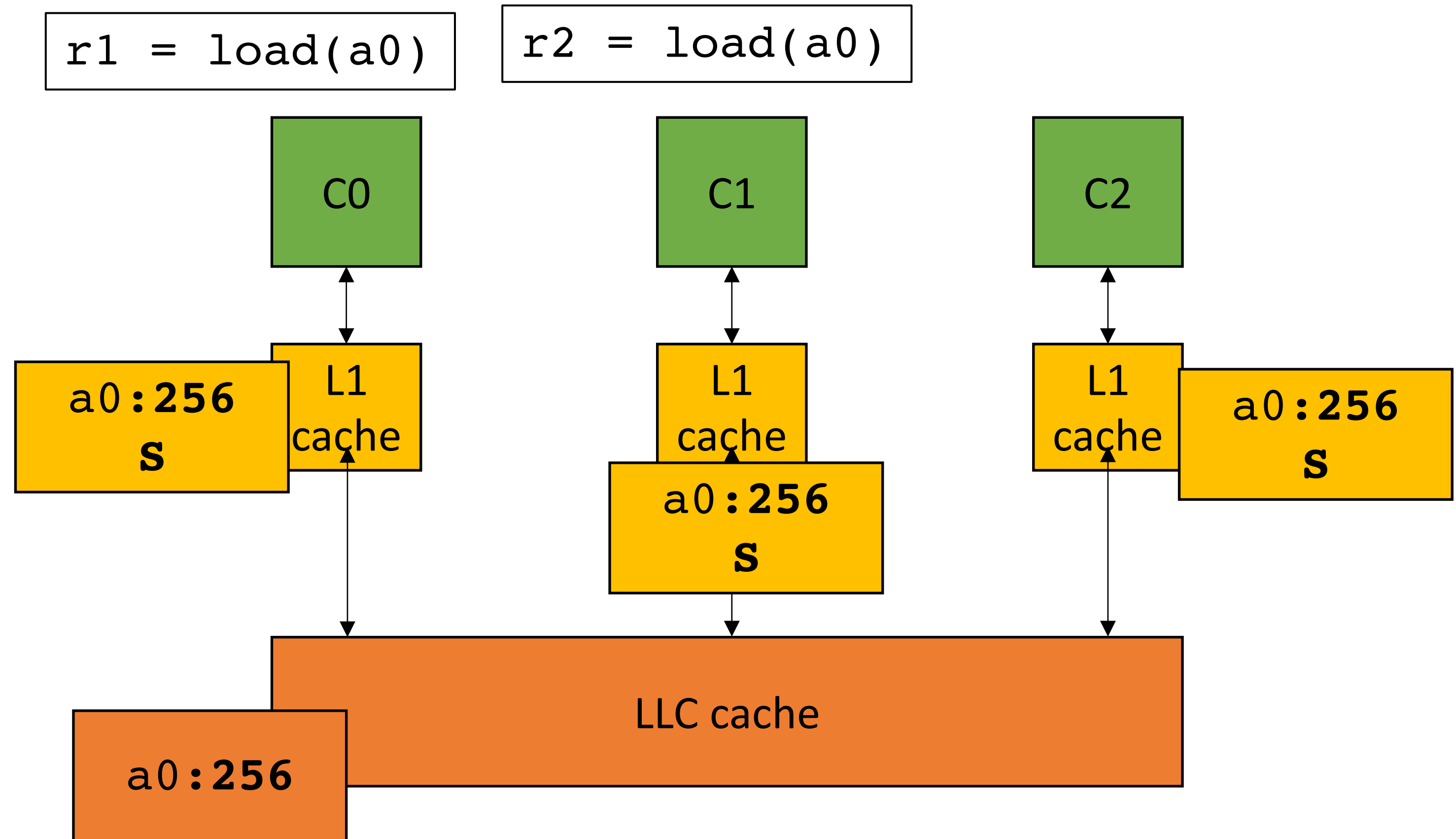
Cache coherence



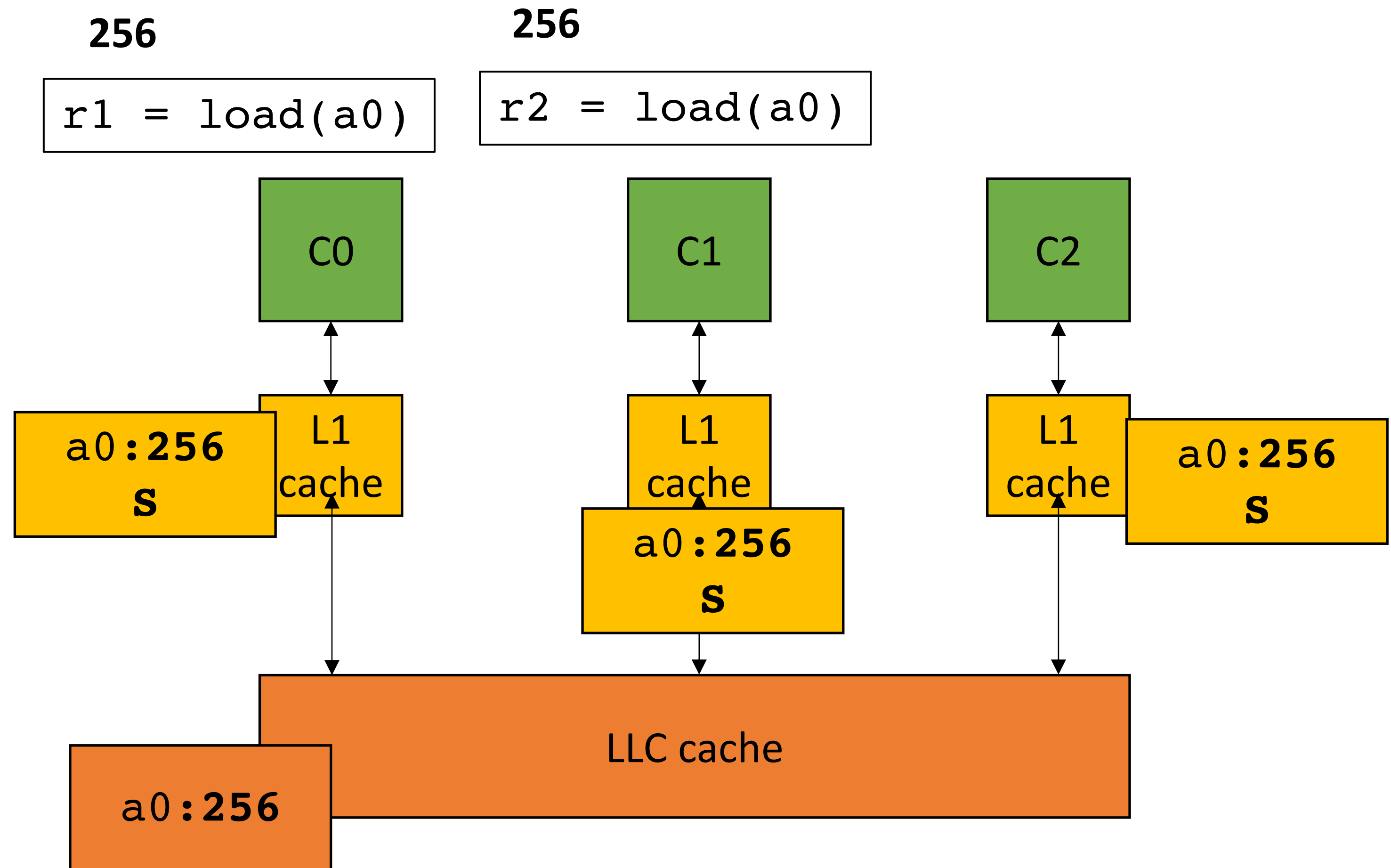
Cache coherence



Cache coherence



Cache coherence



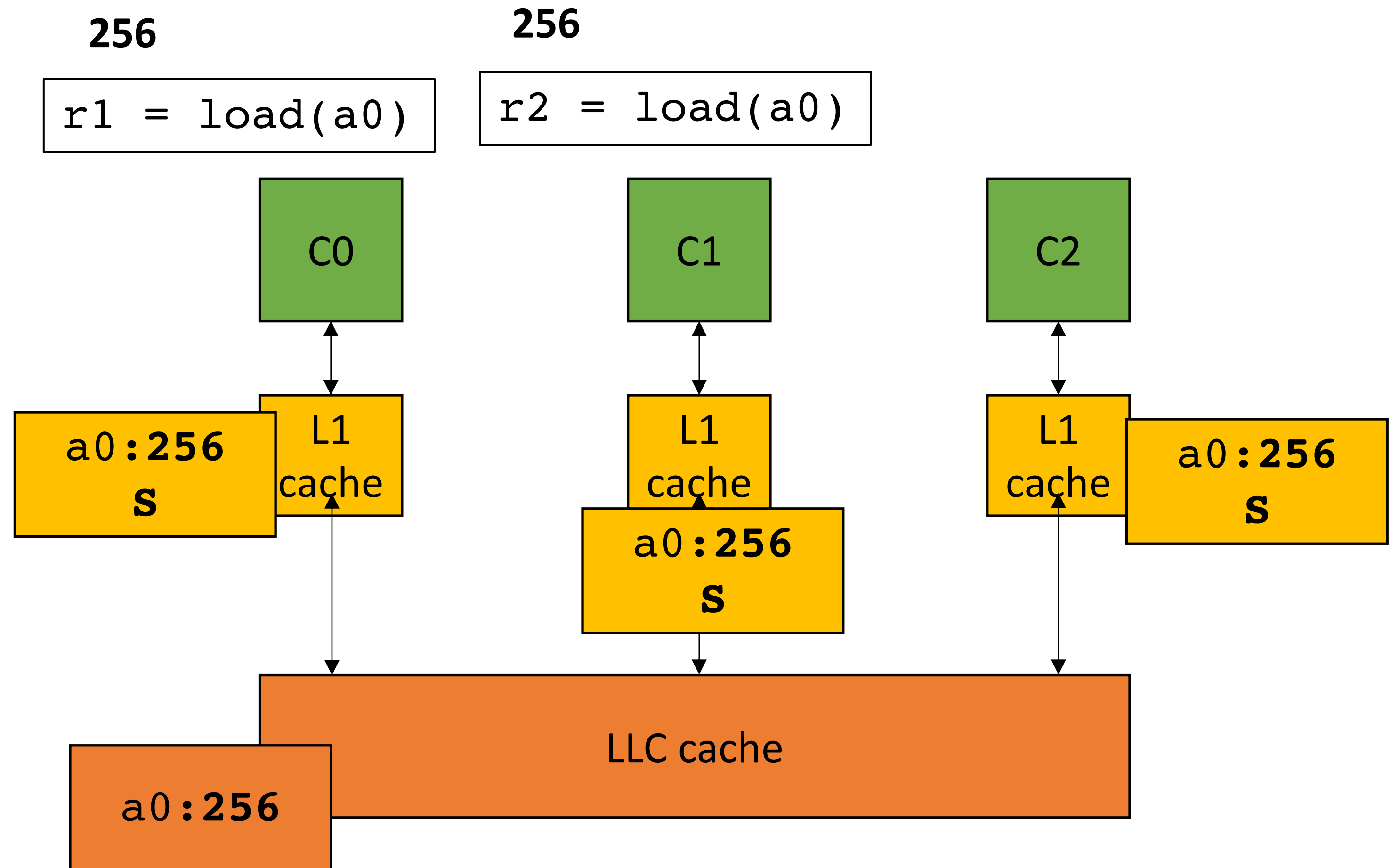
Cache coherence

Takeaways:

Caches must agree on values across cores.

Caches are functionally invisible! Cannot tell with raw input and output

But performance measurements can expose caches, especially if they share the same cache line



Lecture Schedule

- Overview - why do we need a lecture on compilation and architecture?
- Compilation - How do we translate a program from a human-accessible language to a language that the processor understands
- Architecture - How do processors execute programs?
- **Example**

Example

- A function that increments a memory location ITERATION times

```
void repeat_increment(volatile int *a) {
    for (int i = 0; i < ITERATIONS; i++) {
        int tmp = *a;
        tmp +=1;
        *a = tmp;
    }
}
```

Example

- A function that increments a memory location ITERATION times

```
void repeat_increment(volatile int *a) {
    for (int i = 0; i < ITERATIONS; i++) {
        int tmp = *a;
        tmp +=1;
        *a = tmp;
    }
}
```

Example

- A function that increments a memory location ITERATION times
- Do this for 8 elements:
  - Allocate a contiguous array

Example

- A function that increments a memory location ITERATION times

- Do this for 8 elements:
  - Allocate a contiguous array

- Loop through the 8 elements and increment each one:

```
for (int i = 0; i < NUM_ELEMENTS; i++) {
   repeat_increment(a+i);
}
```

Example

- A function that increments a memory location ITERATION times

- Do this for 8 elements:
  - Allocate a contiguous array

- Loop through the 8 elements and increment each one:

```
for (int i = 0; i < NUM_ELEMENTS; i++) {
   repeat_increment(a+i);
}
```

Example

- We can also do each array element in parallel!

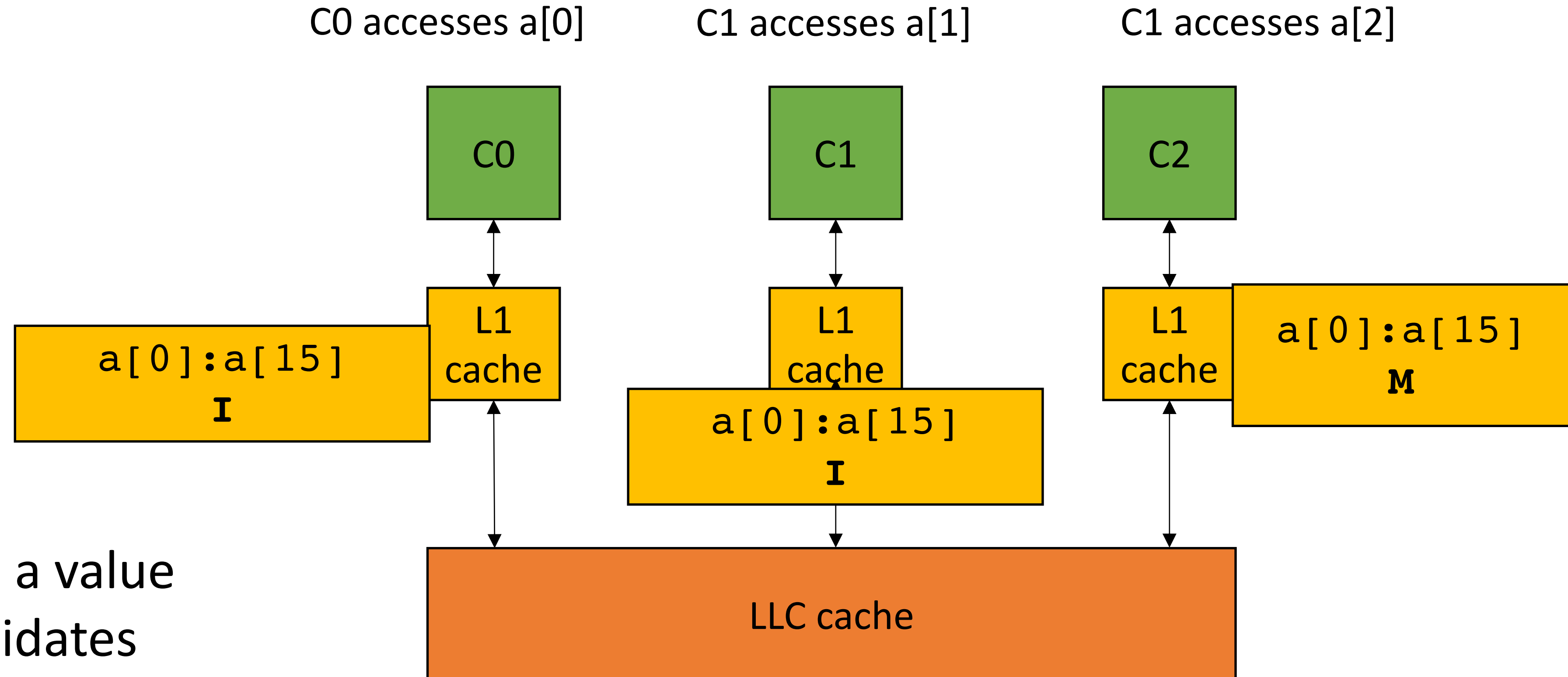
```
for (int i = 0; i < NUM_ELEMENTS; i++) {
   repeat_increment(a+i);
}


for (int i = 0; i < NUM_ELEMENTS; i++) {
   thread(repeat_increment, a+i);
}
```

Don’t worry, we will go over C++ thread in more detail on Thursday

Example

- Run example

What's going on?



when one core modifies a value in the cache line, it invalidates everyone else's cache line.

This is called ***False Sharing***

Fix?

Fix?

- **Padding**: give each element its own cache line:
  - Recall cache line is size 16 ints, so we will use 16x more memory

```
int a[NUM_ELEMENTS * 16];


for (int i = 0; i < NUM_ELEMENTS; i++) {
   thread(repeat_increment, a+(i*16));
}
```

Thank you!

- Remember: Thursday’s lecture is asynchronous
- Homework will be released on Thursday
- We will discuss ILP and C++ threads
- My office hours are on Friday: 3 - 5 pm