Assembly

How do we write string literals:

C:

```
printf("Hello world\n");
```

Assembly:

```
.text
myString:
        .asciz "Hello World\n"

_foo:
        lea   myString(%rip), %rdi
                        # passes the string as a parameter
        mov $0, %rax #printf requires %rax=0.
        call -printf

        # Suppose the variable x is at -8(%rbp)
        # and you want x = "Hello World\n";
        lea myString(%rip), -8(%rbp)
```

Global Variables

- allocated in the data section of memory.

```
.data
myX:
    .long   0
            (4 bytes (32 bits); 0 = initial value)

Y:
    .quad   0xFF
            (8 bytes (64 bits); 0xFF = initial value)

.text
_bar:
    ...
    mov  myX(%rip), %eax      # %rax = myX
```

For a function, string, or global variable to be visible outside the file, you need:

```
.globl myX
```

On a Mac, if you want the function, string, or variable usable by C, then the name has to begin with "_".

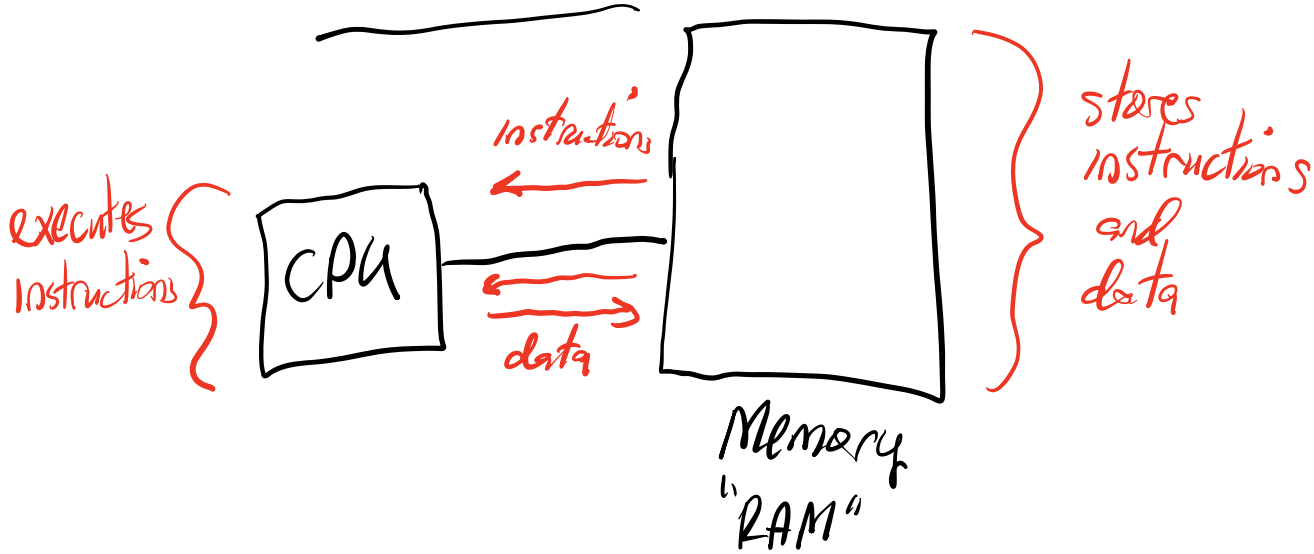
```
.data
.globl _z

_z: .quad 155
```

* See the "Programs discussed in class" tab on Brightspace, under lecture 18.

Done with assembly code!

Caches



The CPU can execute instructions much faster than the memory can provide instructions and data.

3 GHz processor: 3 billion cycles per second
- ideally, the processor can execute a simple instruction (add) in 1 cycle.

↳ 1 cycle $= \frac{1}{3 \text{ billion}}$ sec $= .33$ ns

nanosecond (1-billionth of a sec)

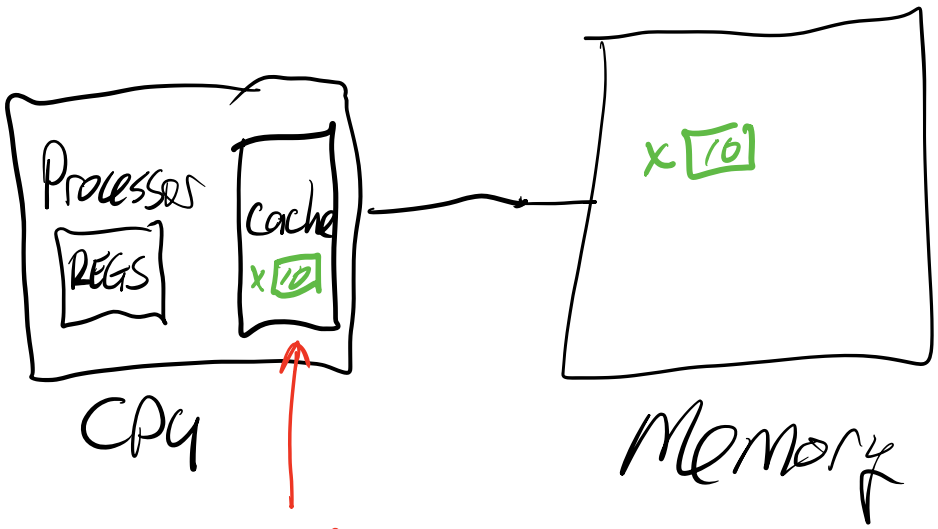
RAM: 60-100 ns
= 180 - 300 cycles

Registers are on the processor, so data in registers can be accessed immediately.
- but lots of data is in memory
- all instructions are in memory.

How to avoid constantly waiting for memory?

- use caches

Cache: very small but very fast memory that is generally on the CPU chip.

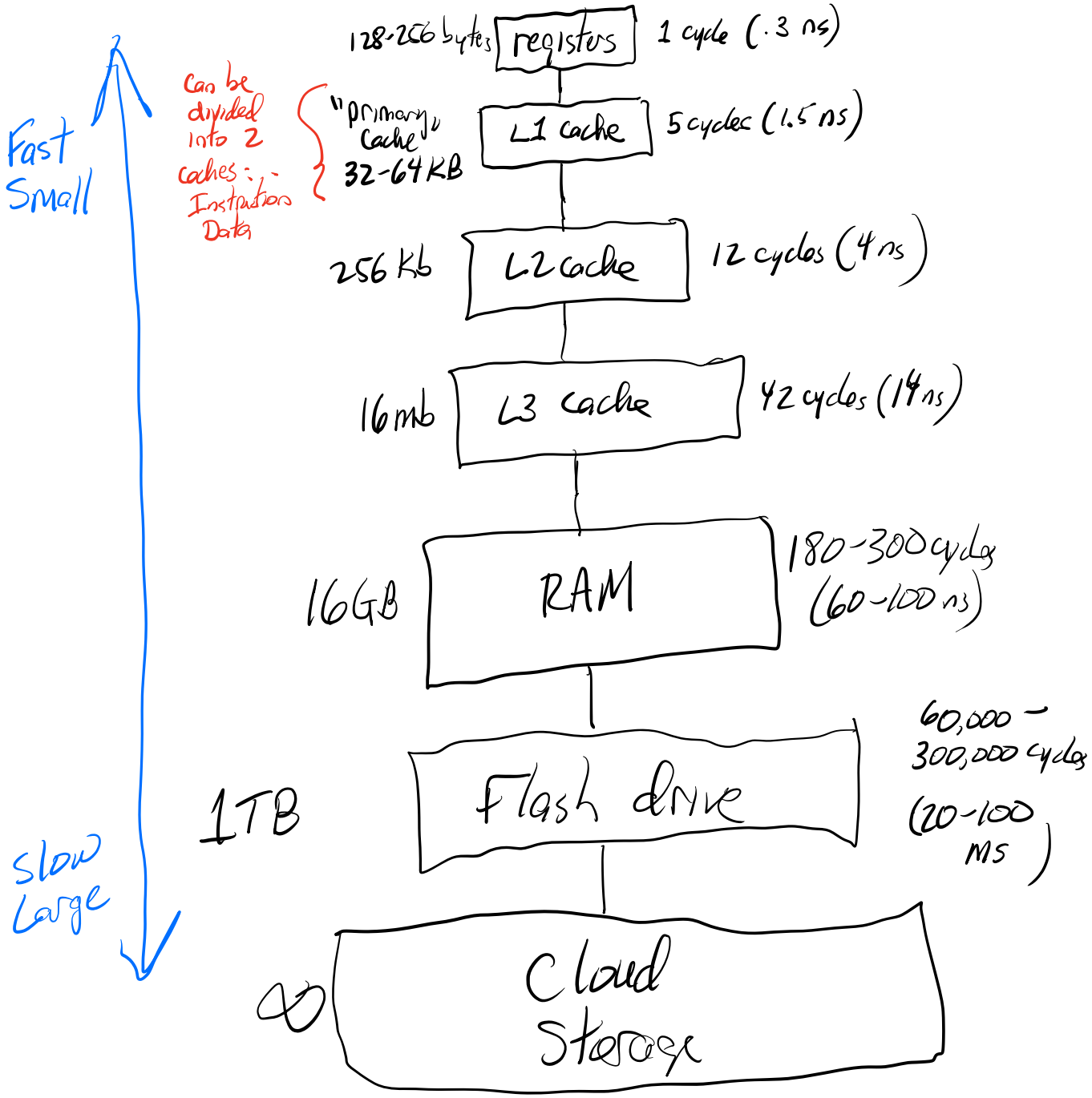


Cache
fast access

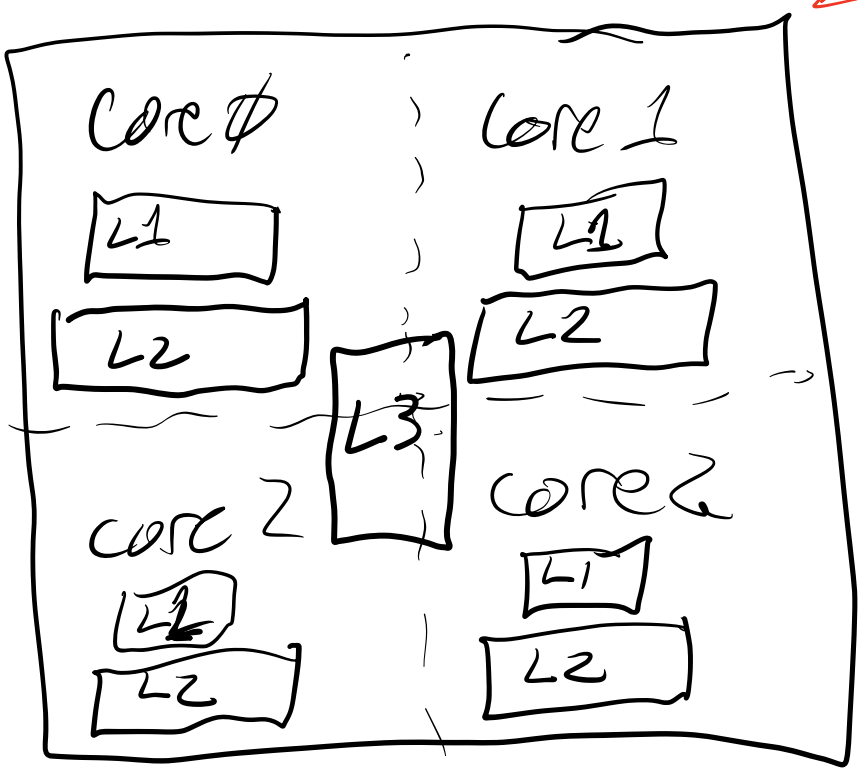
The cache contains copies of the data and instructions currently being used by the program

- when the processor asks for an instruction or data in memory, if that instruction or data is in the cache, then it can be retrieved quickly from the cache.

There are typically multiple levels of cache.
- part of the "memory hierarchy"



On a modern multi-core processor: e.g. 4 core



Each core has it's own L1 and L2 caches, and there is one shared L3 cache.

Some terminology:

"Cache Hit": When the CPU issues a request for data or an instruction in memory, and it is found in the cache.

"Cache Miss" - when the requested instruction or data is not in the cache.
- processor has to wait for RAM to provide it.
slow!

"I-cache" - instruction cache
- holds instructions

"D-cache" - data cache
- holds data

How can we maximize the odds that the instructions or data the processor needs will be in the cache?

- take advantage of "locality"
  - temporal locality
  - spatial locality