C335
Computer Structures

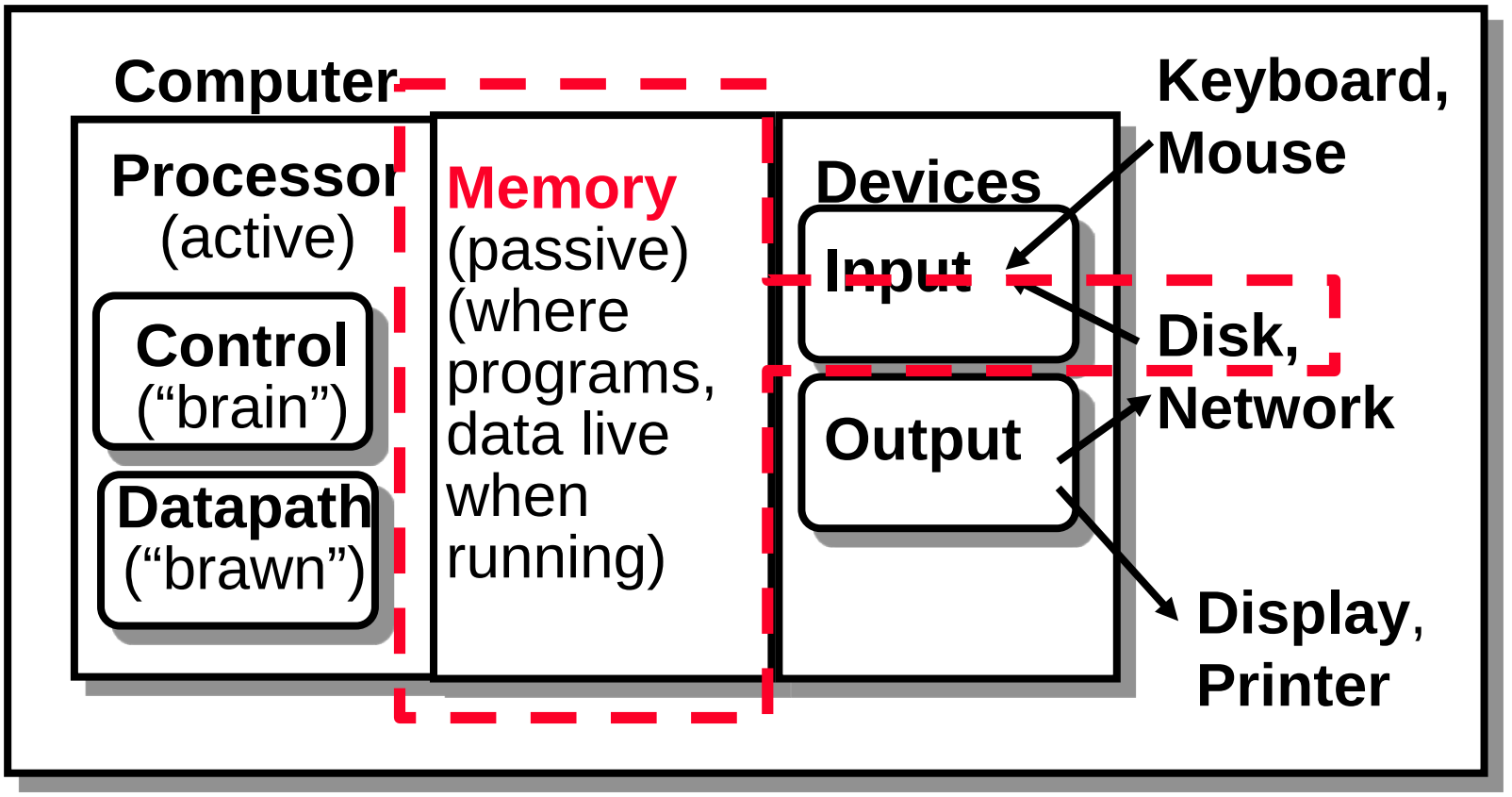
Memory Hierarchies (I)

Dr. Liqiang Zhang

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

The Big Picture

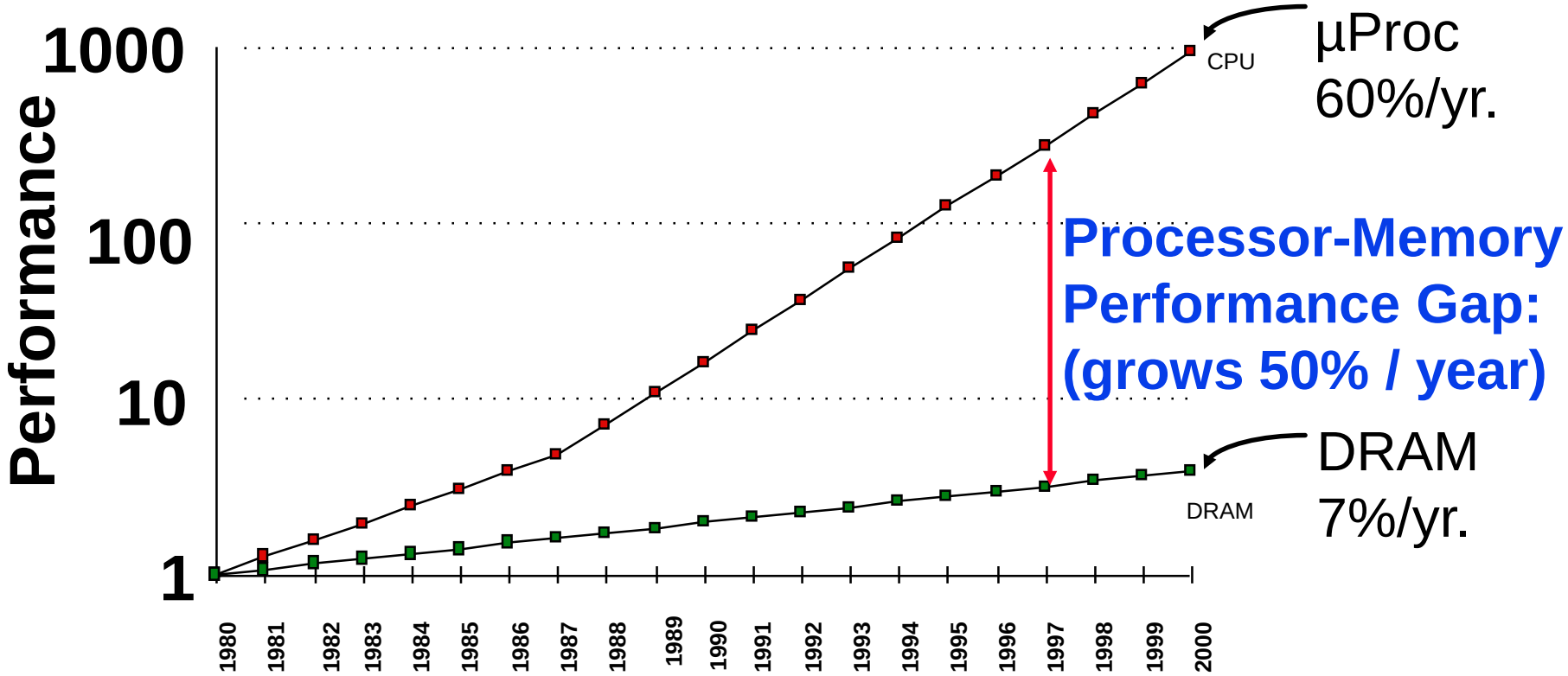


Memory Hierarchy

Storage in computer systems:

- Processor
  - holds data in register file (~100 Bytes)
  - Registers accessed on sub-nanosecond timescale
- Memory (we'll call "main memory")
  - More capacity than registers (~Gbytes)
  - Access time ~10s ns
- Disk
  - HUGE capacity (virtually limitless)
  - VERY slow: runs ~milliseconds

Motivation: Why We Use Caches



Motivation: Why We Use Caches

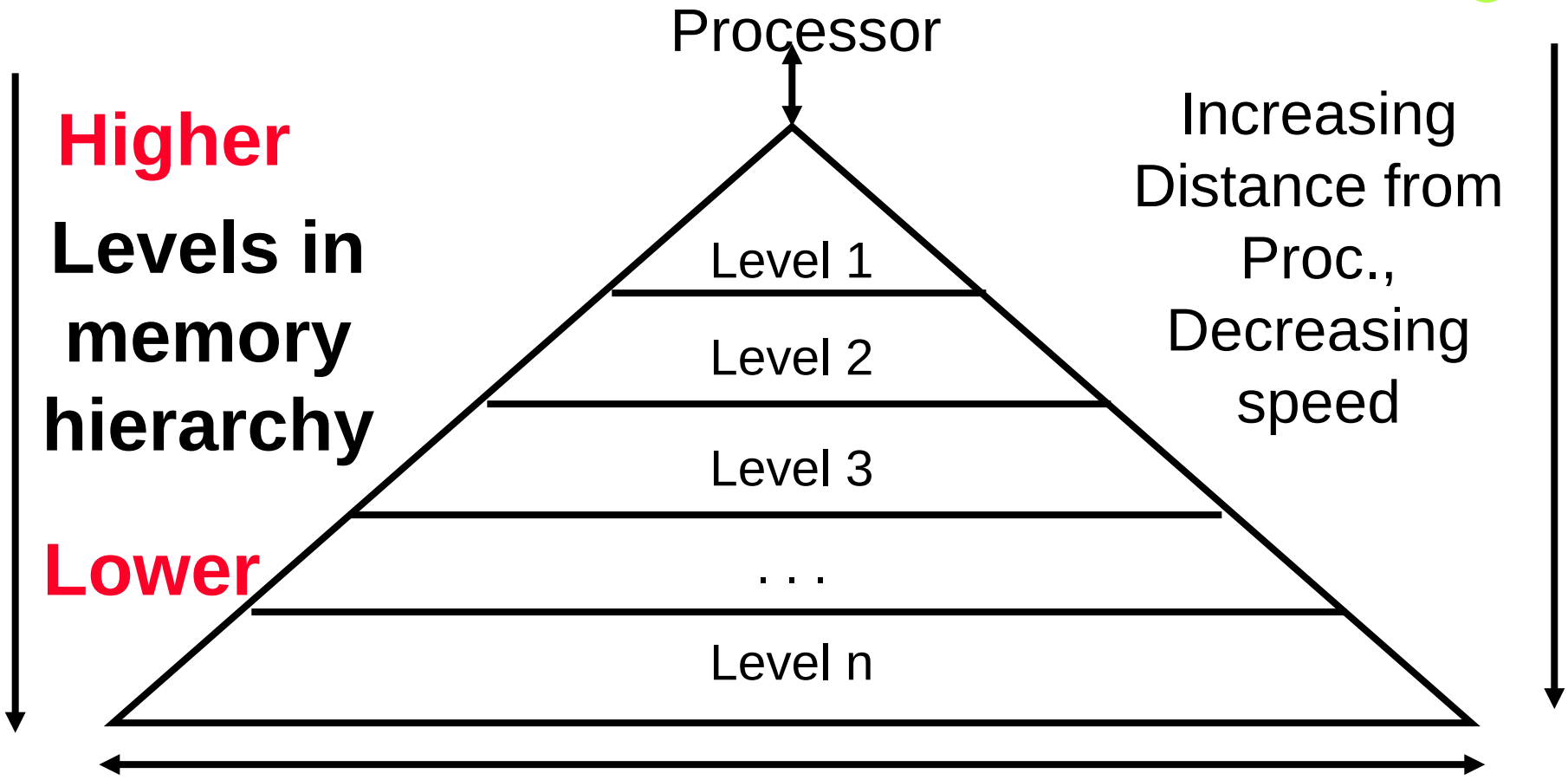
Year introduced	Chip size	$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 Kibibit	$1,500,000	250 ns	150 ns
1983	256 Kibibit	$500,000	185 ns	100 ns
1985	1 Mebibit	$200,000	135 ns	40 ns
1989	4 Mebibit	$50,000	110 ns	40 ns
1992	16 Mebibit	$15,000	90 ns	30 ns
1996	64 Mebibit	$10,000	60 ns	12 ns
1998	128 Mebibit	$4,000	60 ns	10 ns
2000	256 Mebibit	$1,000	55 ns	7 ns
2004	512 Mebibit	$250	50 ns	5 ns
2007	1 Gibibit	$50	45 ns	1.25 ns
2010	2 Gibibit	$30	40 ns	1 ns
2012	4 Gibibit	$1	35 ns	0.8 ns

DRAM size increased by multiples of four approximately once every three years until 1996, and thereafter considerably slower. The improvements in access time have been slower but continuous, and cost roughly tracks density improvements, although cost is often affected by other issues, such as availability and demand. The cost per gigabyte is not adjusted for inflation.

Memory Caching

- Mismatch between processor and memory speeds leads us to add a new level: a memory cache
- Implemented with same IC processing technology as the CPU (usually integrated on same chip): faster but more expensive than DRAM memory.
  - 1989 first Intel CPU with cache on chip
  - 1998 Pentium III has two levels of cache on chip
  - 2003 Itanium 2 has three levels of cache on chip
- Cache is a copy of a subset of main memory.
- Most processors have separate caches for instructions and data.

Memory Hierarchy



As we move to deeper levels the latency goes up and price per bit goes down.

Memory Hierarchy

Memory technology	Typical access time
SRAM	0.5 – 2.5 ns
DRAM	40- 70 ns
Magnetic disk	5,000,000 – 20,000,000 ns

Memory Hierarchy

- If level closer to Processor, it is:
  - smaller
  - faster
  - subset of lower levels (contains most recently used data)*
- Lowest Level (usually disk) contains all available data
- Memory Hierarchy presents the processor with the illusion of a very large very fast memory.

Memory Hierarchy Basis

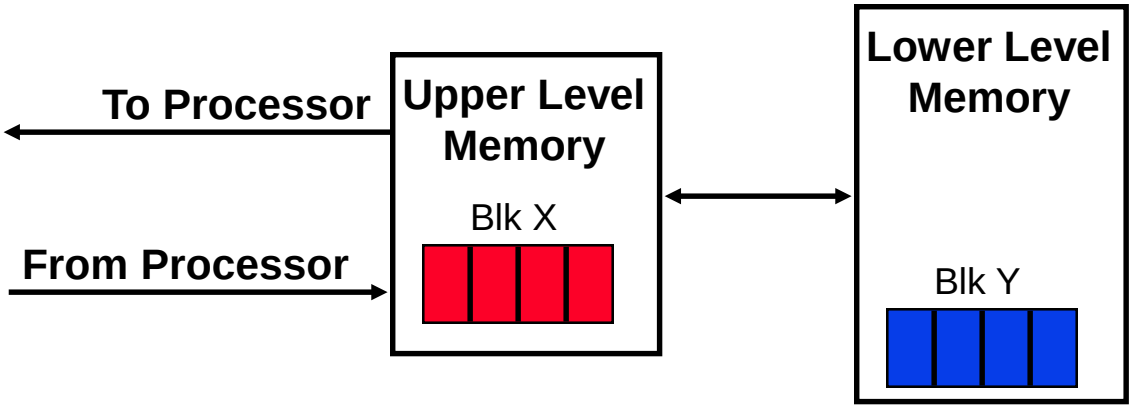
- Cache contains copies of data in memory that are being used.
- Memory contains copies of data on disk that are being used.
- Caches work on the principles of temporal and spatial locality.
  - Temporal Locality: if we use it now, chances are we'll want to use it again soon.
  - Spatial Locality: if we use a piece of memory, chances are we'll use the neighboring pieces soon.

More about Locality

- Temporal Locality (Locality in Time):
  - Low temporal locality for data means accessing variables only once.
  - High temporal locality for data means accessing variables over and over again.
- Spatial Locality (Locality in Space):
  - Low spatial locality for data means no marching through arrays; data is scattered.
  - High spatial locality for data implies marching through arrays.

The Memory Hierarchy: Why Does it Work?

- Temporal Locality (Locality in Time):
  - ⇒ Keep most recently accessed instructions/datum closer to the processor
- Spatial Locality (Locality in Space):
  - ⇒ Move blocks consisting of contiguous words to the upper levels



Cache Design

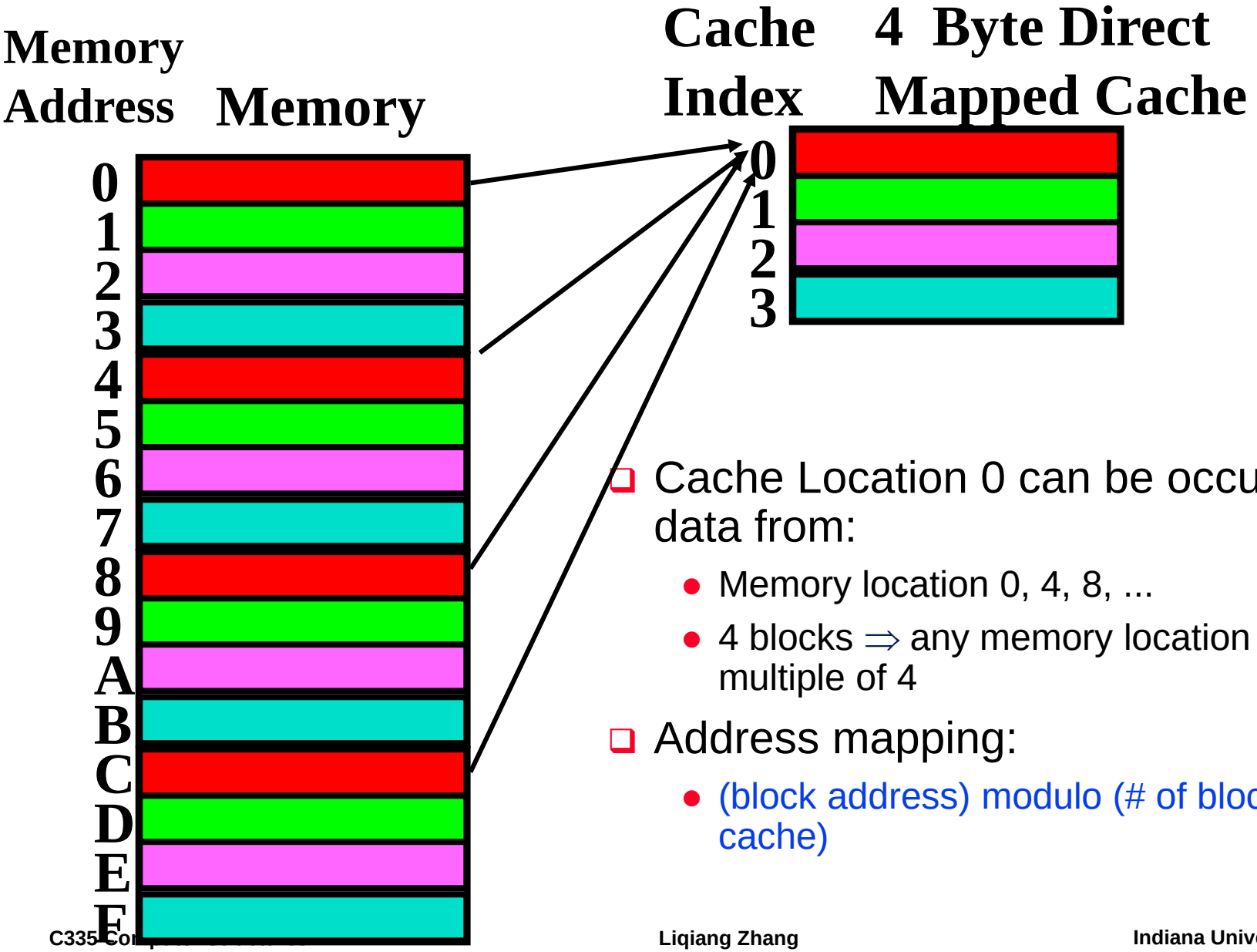
- How do we organize cache?
- Where does each memory address map to?

  (Remember that cache is subset of memory, so multiple memory addresses map to the same cache location.)
- How do we know which elements are in cache?
- How do we quickly locate them?

Direct-Mapped Cache (1/2)

- In a direct-mapped cache, each memory address is associated with one possible block within the cache
  - Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
  - Block is the unit of transfer between cache and memory

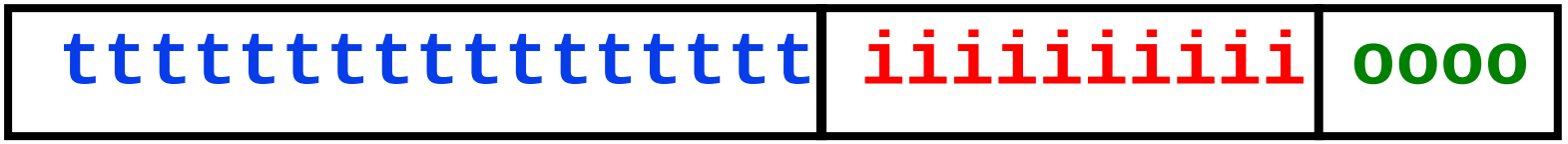
Direct-Mapped Cache (2/2)



- Cache Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ...
  - 4 blocks $\Rightarrow$ any memory location that is multiple of 4
- Address mapping:
  - (block address) modulo (# of blocks in the cache)

Issues with Direct-Mapped

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- What if we have a block size > 1 byte?
- Answer: divide memory address into three fields



tag	index	byte
to check	to	offset
if have	select	within
correct block	block	block

Direct-Mapped Cache Terminology

- All fields are read as unsigned integers.
- Index: specifies the cache index (which “row”/block of the cache we should look in)
- Offset: once we’ve found correct block, specifies which byte within the block we want
- Tag: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location

Memory address: TIO

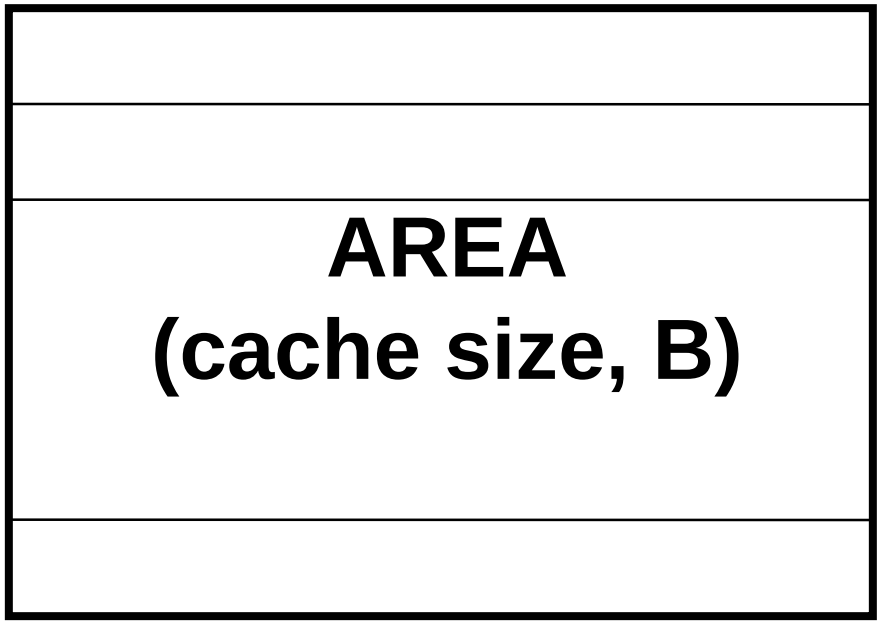
AREA (cache size, B)
= HEIGHT (# of blocks)
* WIDTH (size of one block, B/block)

$2^{(I+O)} = 2^I * 2^O$

Tag	Index	Offset

WIDTH
(size of one block, Bytes/block, 2^O)

HEIGHT
(# of blocks, 2^I)



Direct-Mapped Cache Example (1/3)

- Suppose we have a 16KB of data in a direct-mapped cache with 4-word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture
- Offset
  - need to specify correct byte within a block
  - block contains 4 words

    = 16 bytes

    = 2^4 bytes
  - need 4 bits to specify correct byte

Direct-Mapped Cache Example (2/3)

- Index: (~index into an “array of blocks”)
  - need to specify correct block in cache
  - cache contains 16 KB = 2^{14} bytes
  - block contains 2^4 bytes (4 words)
  - # blocks/cache

$$= \frac{\text{bytes/cache}}{\text{bytes/block}}$$

$$= \frac{2^{14}\ \text{bytes/cache}}{2^4\ \text{bytes/block}}$$

$$= 2^{10}\ \text{blocks/cache}$$

  - need 10 bits to specify this many blocks

Direct-Mapped Cache Example (3/3)

- Tag: use remaining bits as tag
  - tag length = addr length – offset - index
    = 32 - 4 - 10 bits
    = 18 bits
  - so tag is leftmost 18 bits of memory address

- Why not full 32 bit address as tag?
  - All bytes within block share the same tag (saves 4bits)
  - The row # of the cache block is the index itself (saves 10bits)

Exercise

- Suppose we have a 256KB of data in a direct-mapped cache with 16-word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture

Exercise

- Suppose we have a 128KB of data in a direct-mapped cache with 32-word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture

Caching Terminology

- When we try to read memory,
  3 things can happen:

1. cache hit:
   cache block is valid and contains proper address, so read desired word
2. cache miss:
   nothing in cache in appropriate block, so fetch from memory
3. cache miss, block replacement:
   wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)

And in Conclusion…

- We would like to have the capacity of disk at the speed of the processor: unfortunately this is not feasible.
- So we create a memory hierarchy:
  - each successively closer level contains “most frequently used” data from next further level
  - exploits temporal & spatial locality
  - do the common case fast, worry less about the exceptions (design principle of MIPS)
- Locality of reference is a Big Idea