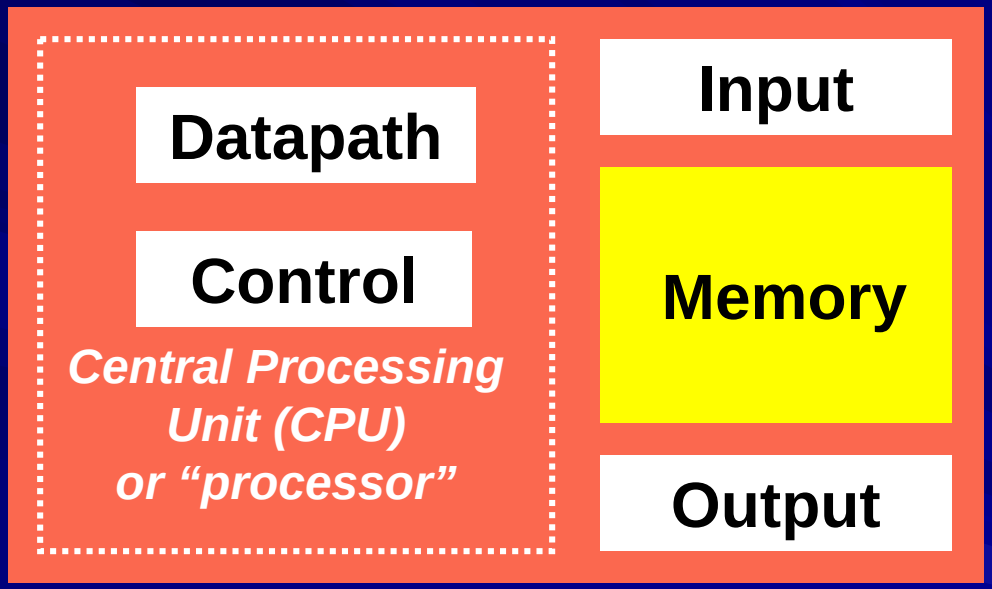
ELEC 5200-001/6200-001
Computer Architecture and Design
Spring 2019

Memory Organization (Chapter 5)

Christopher B. Harris
Assistant Professor
Department of Electrical and Computer Engineering
Auburn University, Auburn, AL 36849

(Adapted from slides by Vishwani D. Agrawal)

Five Parts of a Computer



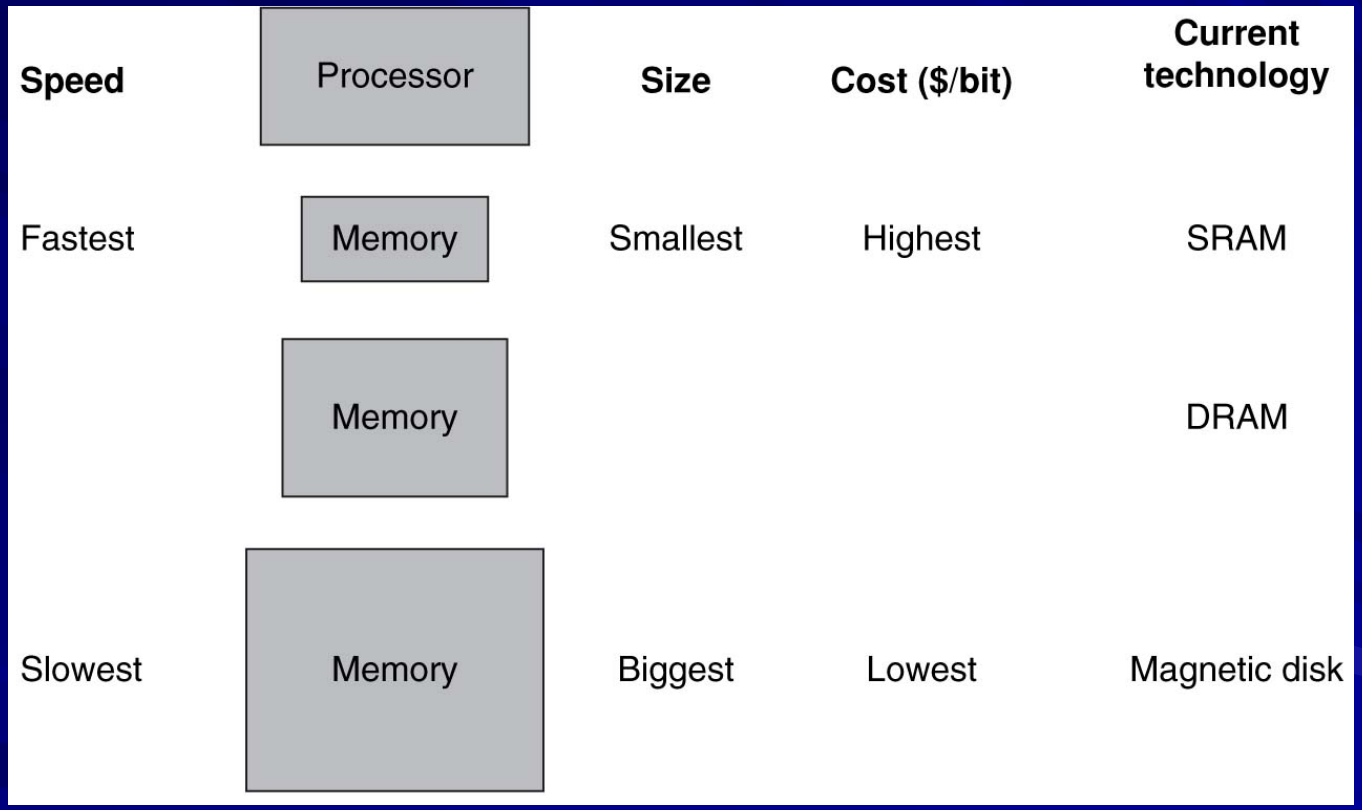
Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

Taking Advantage of Locality

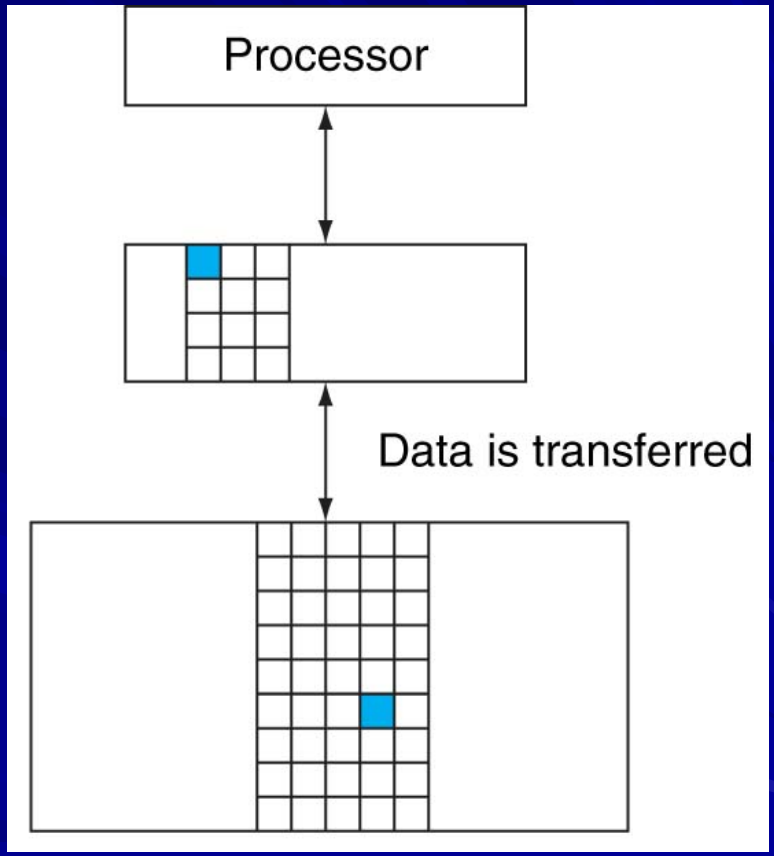
- Store everything on disk
  - Copy recently accessed (and nearby) items from disk to smaller, closer memory
    - Main memory (DRAM)
  - Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
    - Cache memory attached to CPU
  - Items used right now are stored in fastest, closest memory. (CPU register file)
- This is called the “Memory hierarchy”

Memory Hierarchy

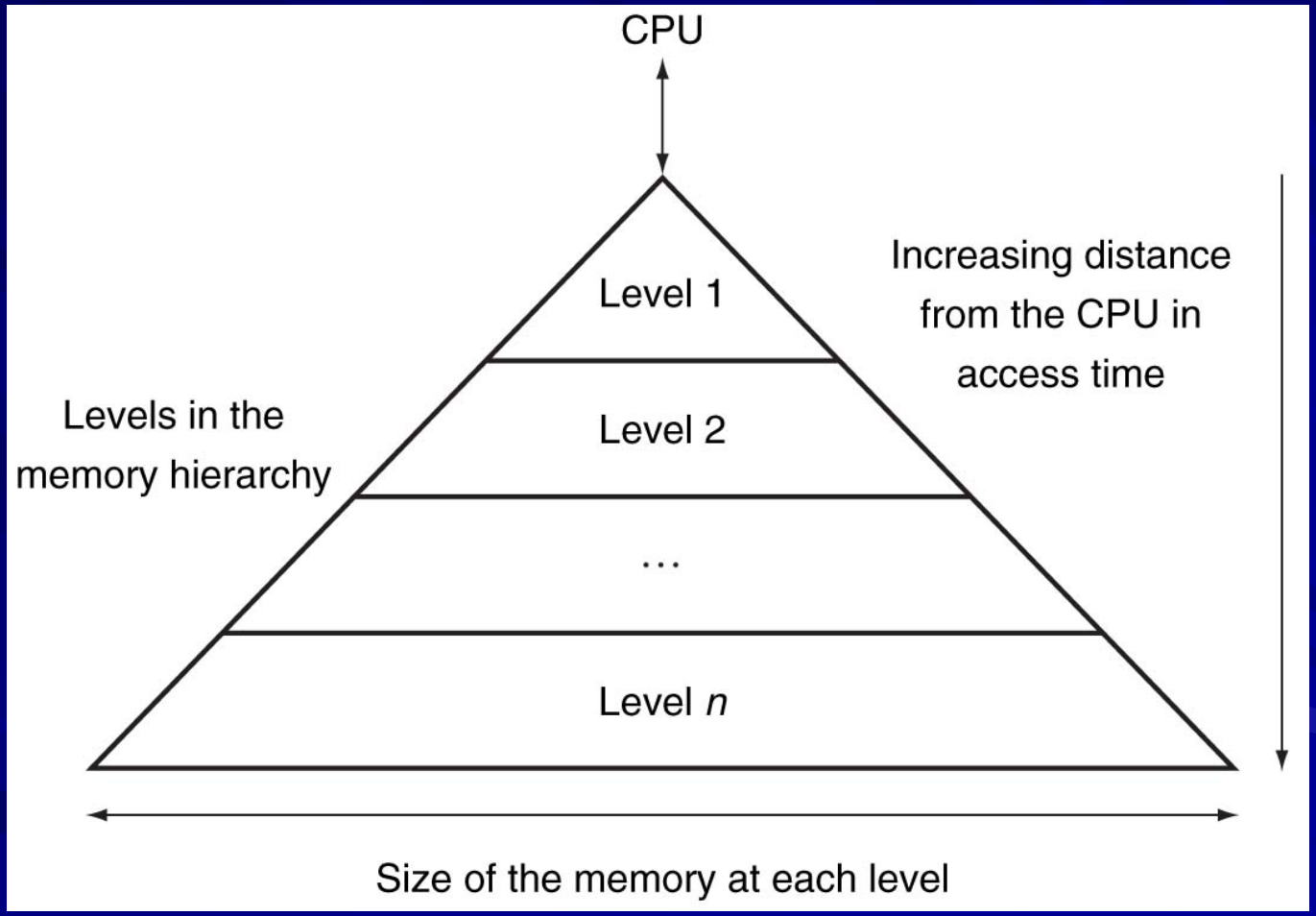


Interaction Between Memory Levels

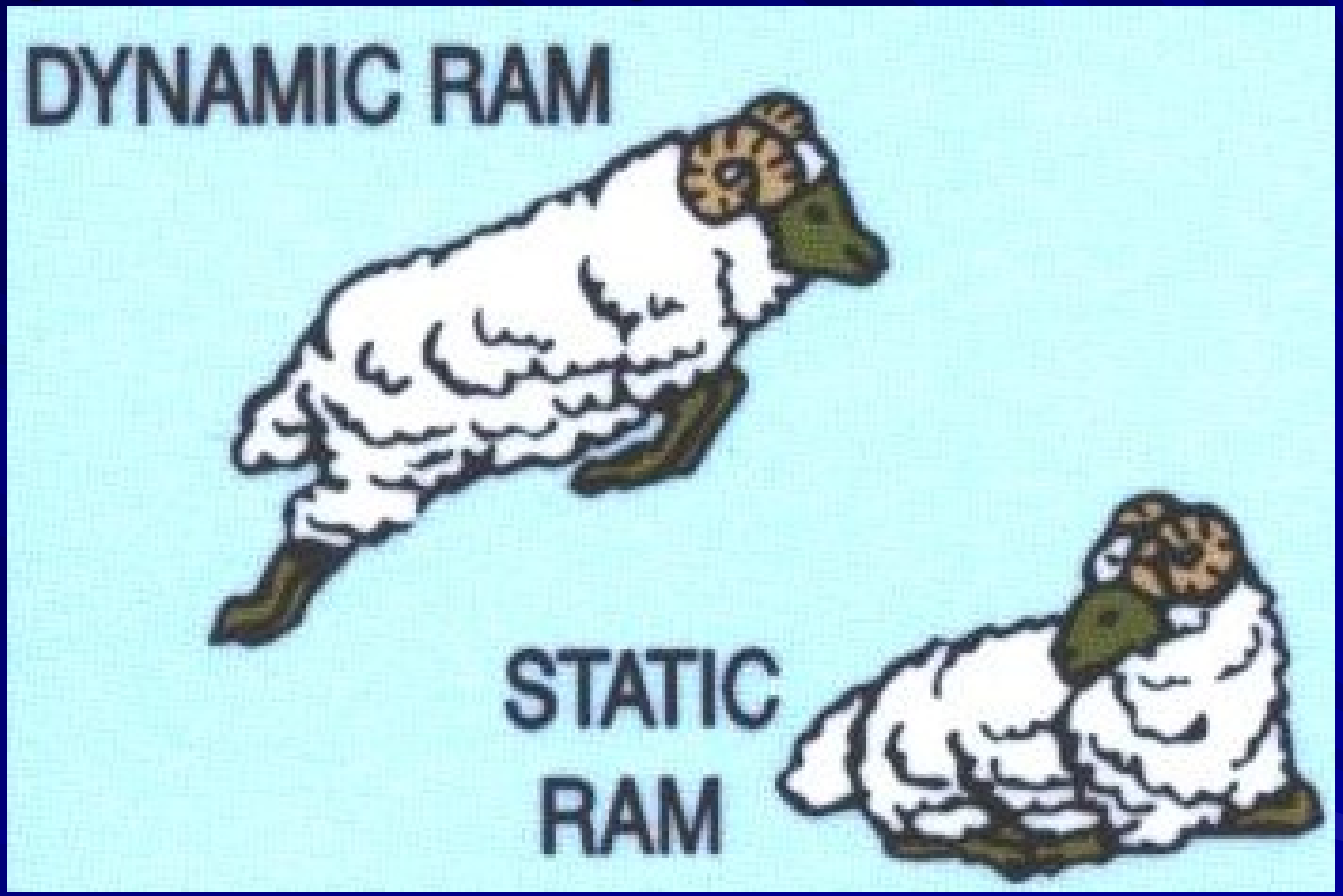
- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses = 1 – hit ratio
  - Then accessed data supplied from upper level



Memory Hierarchy Pyramid



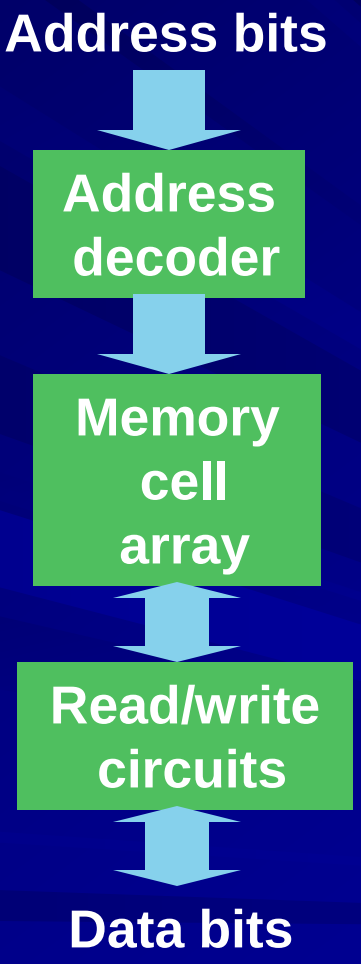
Types of Computer Memories



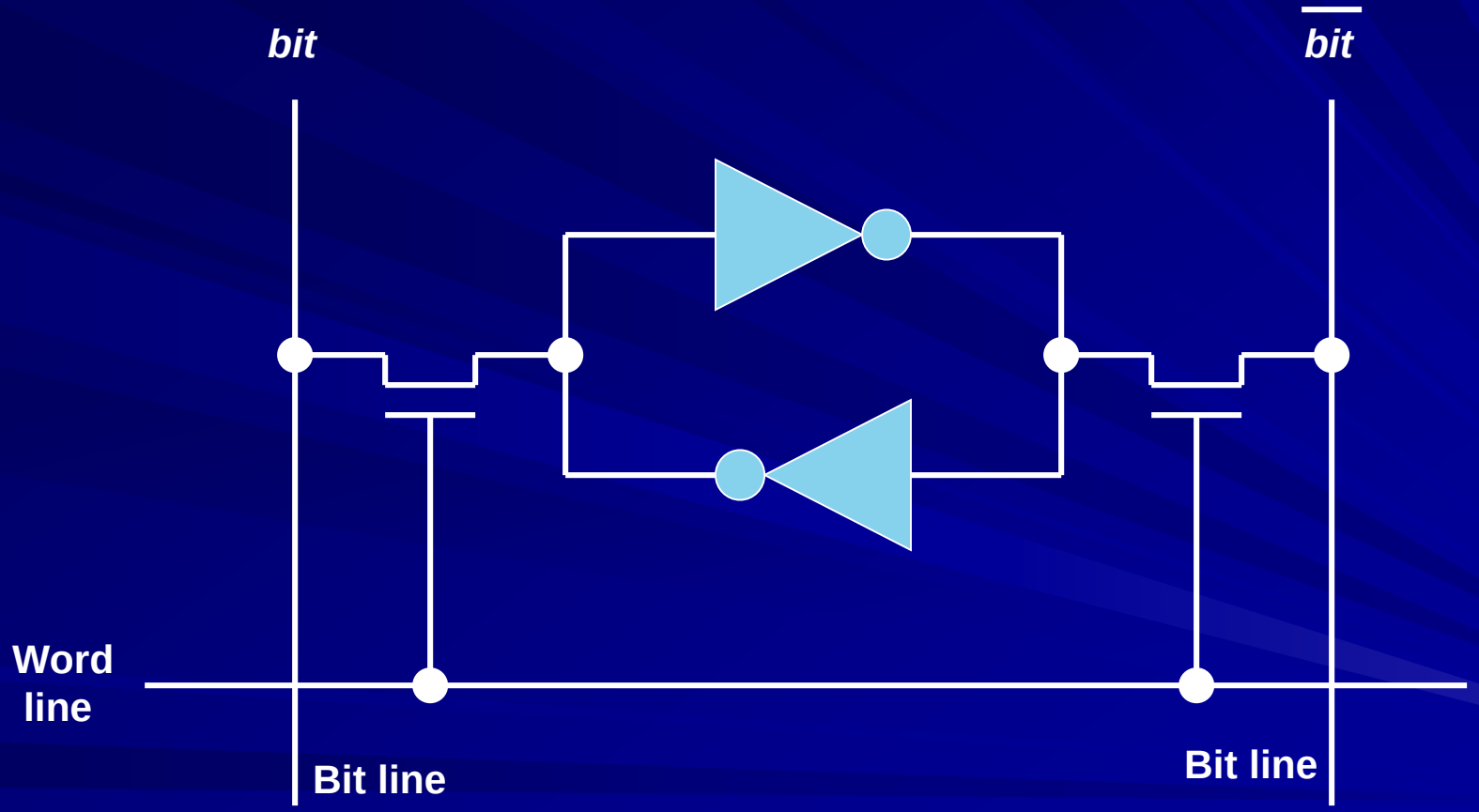
From the cover of:

A. S. Tanenbaum, *Structured Computer Organization, Fifth Edition*, Upper Saddle River, New Jersey: Pearson Prentice Hall, 2006.

Random Access Memory (RAM)

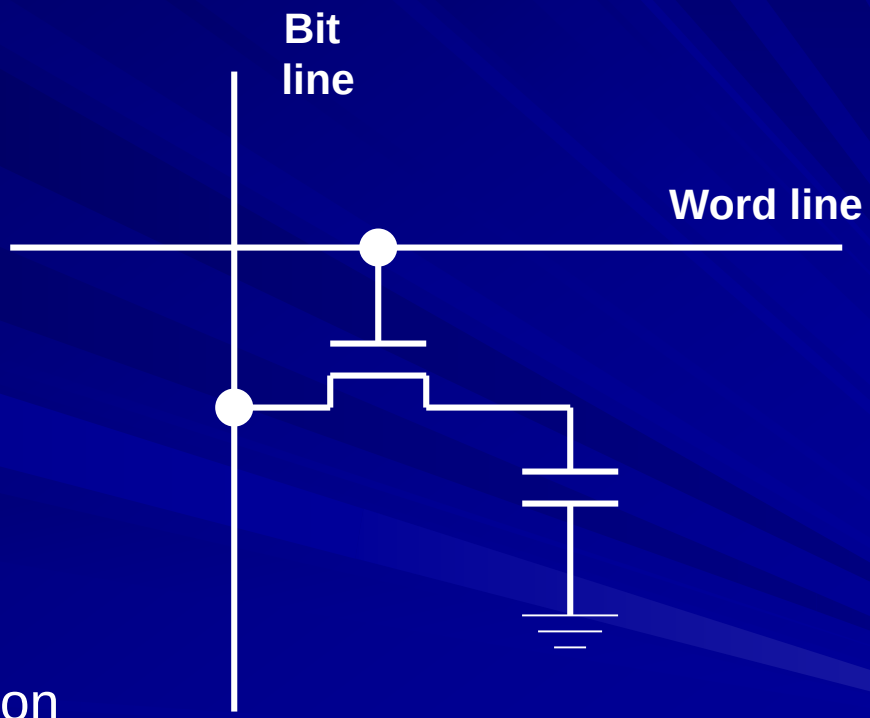


Six-Transistor SRAM Cell



Dynamic RAM (DRAM) Cell

"Single-transistor DRAM cell"
Robert Dennard's 1967 invevention



Electronic Memory Devices

Memory technology	Typical access time	Clock rate GHz	Cost per GB in 2011*
SRAM	~1 ns	1.0 GHz	$500.00
DRAM	~10 ns	100 MHz	$6.00
Flash	~50 us	20 kHz	$1.50
Magnetic disk	~4 ms	250 Hz	$0.10

* from **Hierarchy of Memory and Caches** by Carl Burch, Hendrix College, http://www.toves.org/books/cache/

For more on memories:

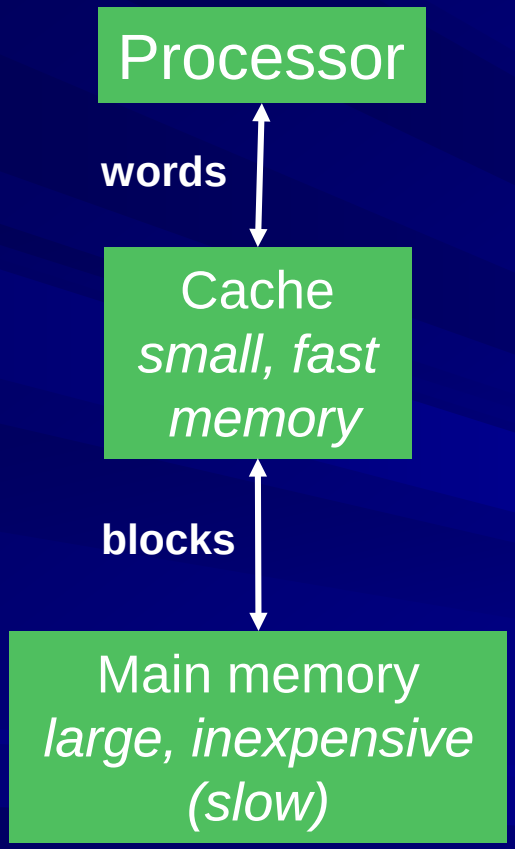
Semiconductor Memories: A Handbook of Design, Manufacture and Application, by Betty Prince, Wiley 1996.

Emerging Memories: Technologies and Trends**,** by Betty Prince, Springer 2002.

Building a Computer with 1GHz Clock and 100 GB Memory

Type of memory	Cost	Clock rate
SRAM	$50,000	1 GHz
DRAM	$600	100 MHz
Flash	$150	20 kHz
Disk	$10	250 Hz

Cache



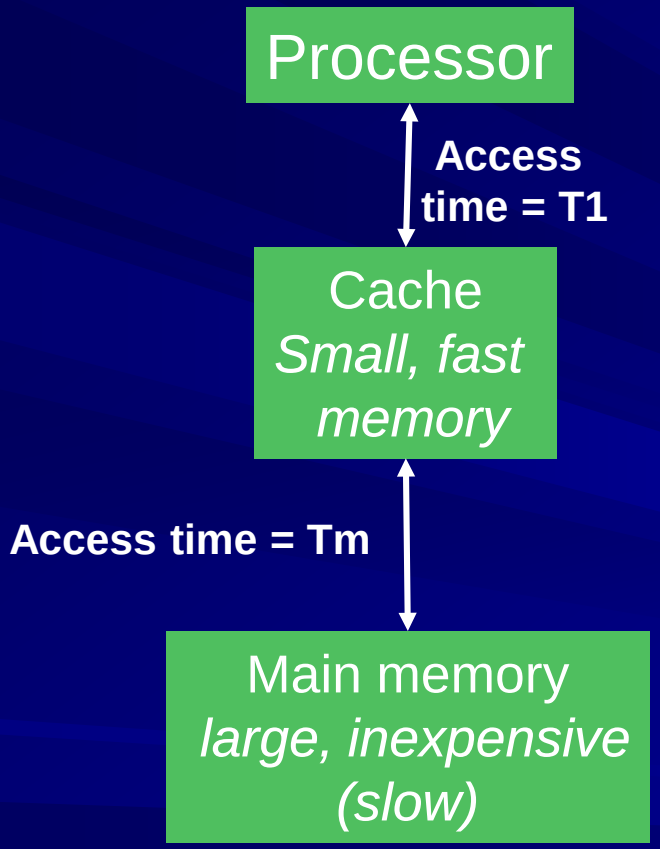
- Processor does all memory operations with cache.
- *Miss* – If requested word is not in cache, a *block* of words containing the requested word is brought to cache, and then the processor request is completed.
- *Hit* – If the requested word is in cache, read or write operation is performed directly in cache, without accessing main memory.
- *Block* – minimum amount of data transferred between cache and main memory.

Inventor of Cache

M. V. Wilkes, "Slave Memories and Dynamic Storage Allocation," *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 2, pp. 270-271, April 1965.

Cache
noun: **cache**; plural noun: **caches**
a hidden or inaccessible storage place for valuables.
(yes, it is also more expensive…)

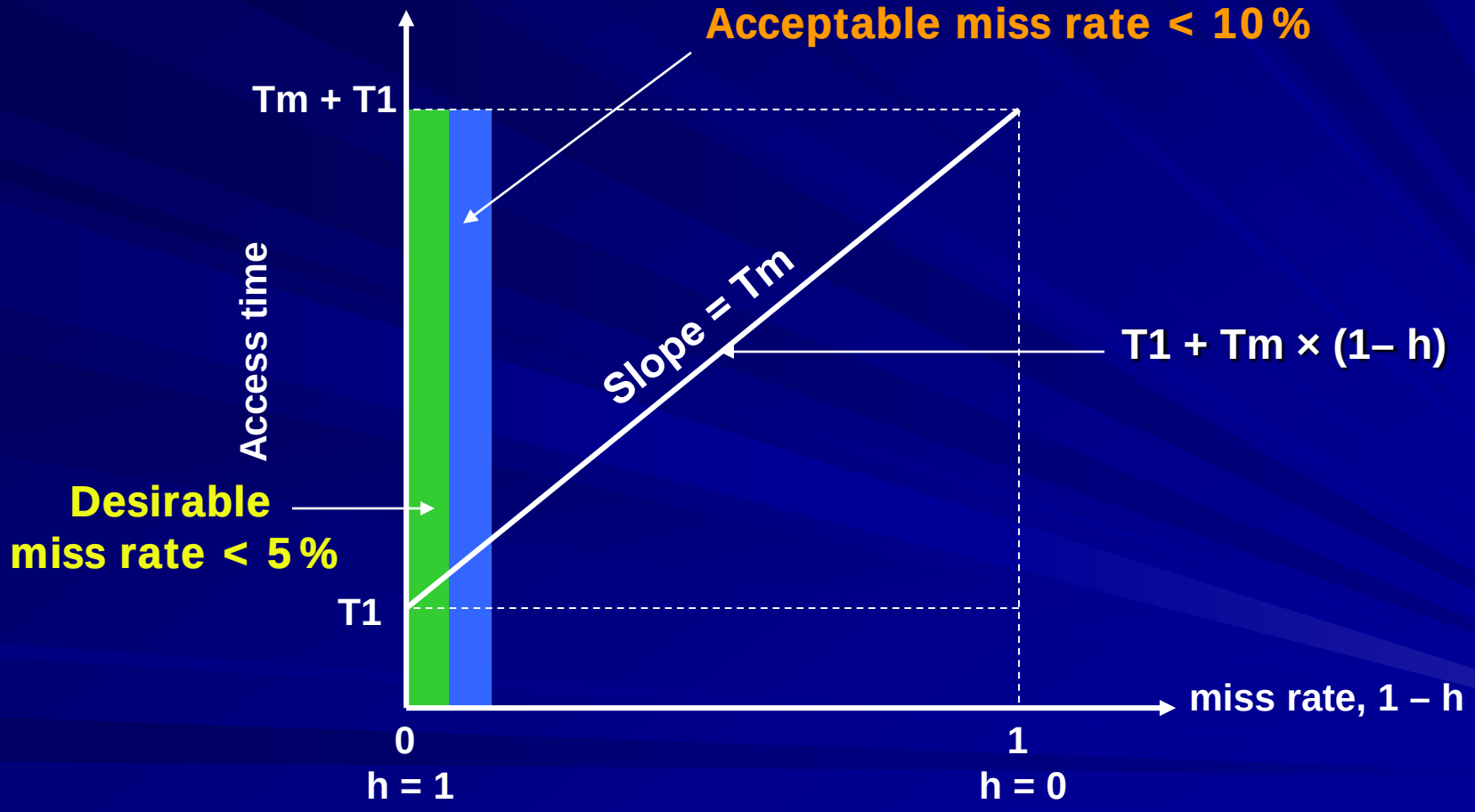
Cache Performance



- Average access time
  $= T1 \times h + (Tm + T1) \times (1 - h)$
  $= T1 + Tm \times (1 - h)$

  where
  - T1 = cache access time (small)
  - Tm = memory access time (large)
  - h = hit rate ($0 \le h \le 1$)
- Hit rate is also known as hit ratio,
  miss rate = 1 – hit rate

Average Access Time



Comparing Performance

- Ideal processor with 1 cycle memory access, CPI = 1
- Processor without cache:
  - Assume main memory access time of 10 cycles
  - Assume 30% instructions require memory data access
- Processor with cache:
  - Assume cache access time of 1 cycle
  - Assume hit rate 0.95 for instructions, 0.90 for data
  - Assume miss penalty (time to read memory into cache and from cache to processor) is 11 cycles

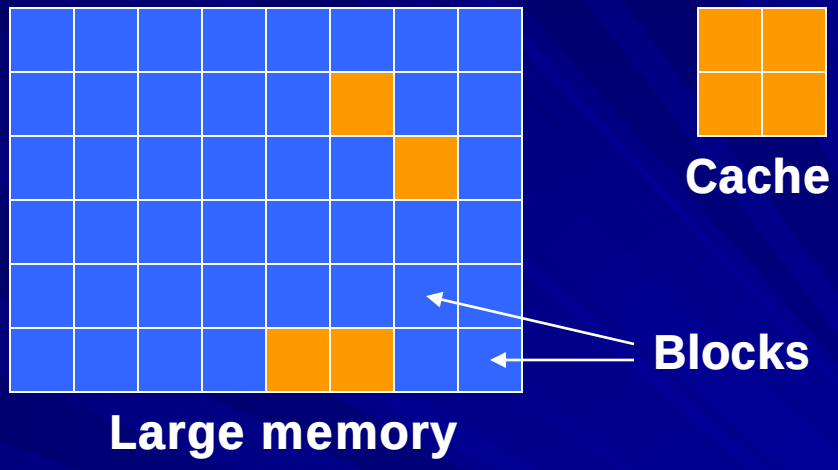
– *Comparing times of* 100 *instructions:*

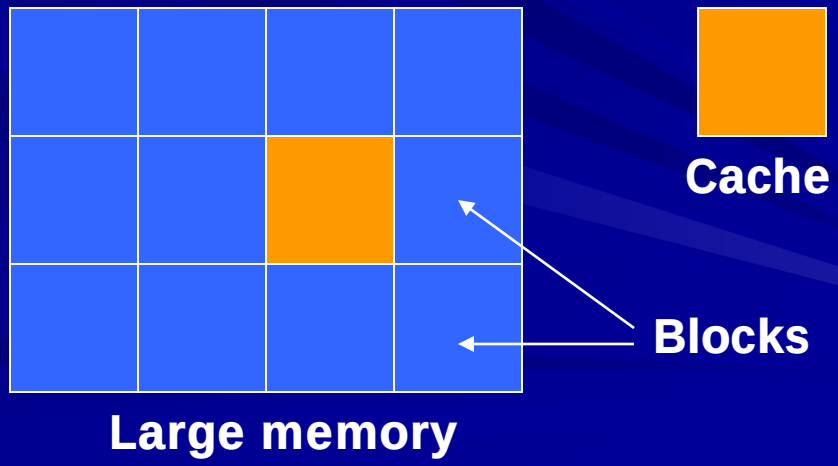
$$\frac{\text{Time without cache}}{\text{Time with cache}} = \frac{100\times10 + 30\times10}{100(0.95\times1+0.05\times11) + 30(0.9\times1+0.1\times11)}$$

$$= 6.19$$

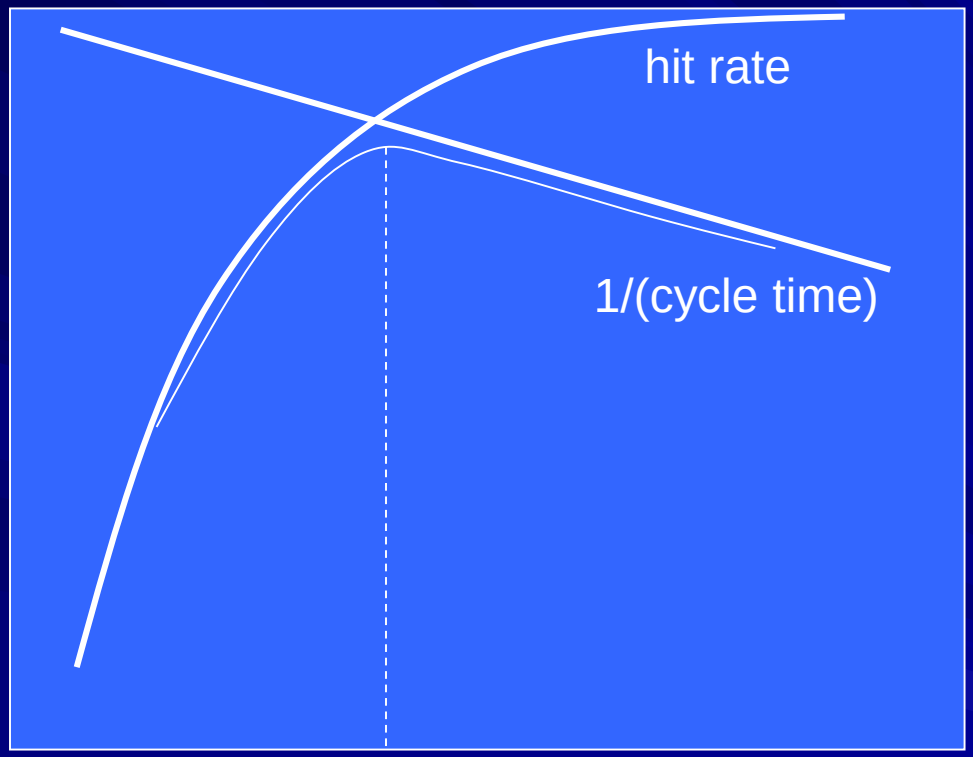
Controlling Miss Rate

- Increase cache size
  - More blocks can be kept in cache; lower miss rate.
  - Larger cache is slower; expensive.
- Increase block size
  - More data available; may lower miss rate.
  - Fewer blocks in cache increase miss rate.
  - Larger blocks need more time to swap.

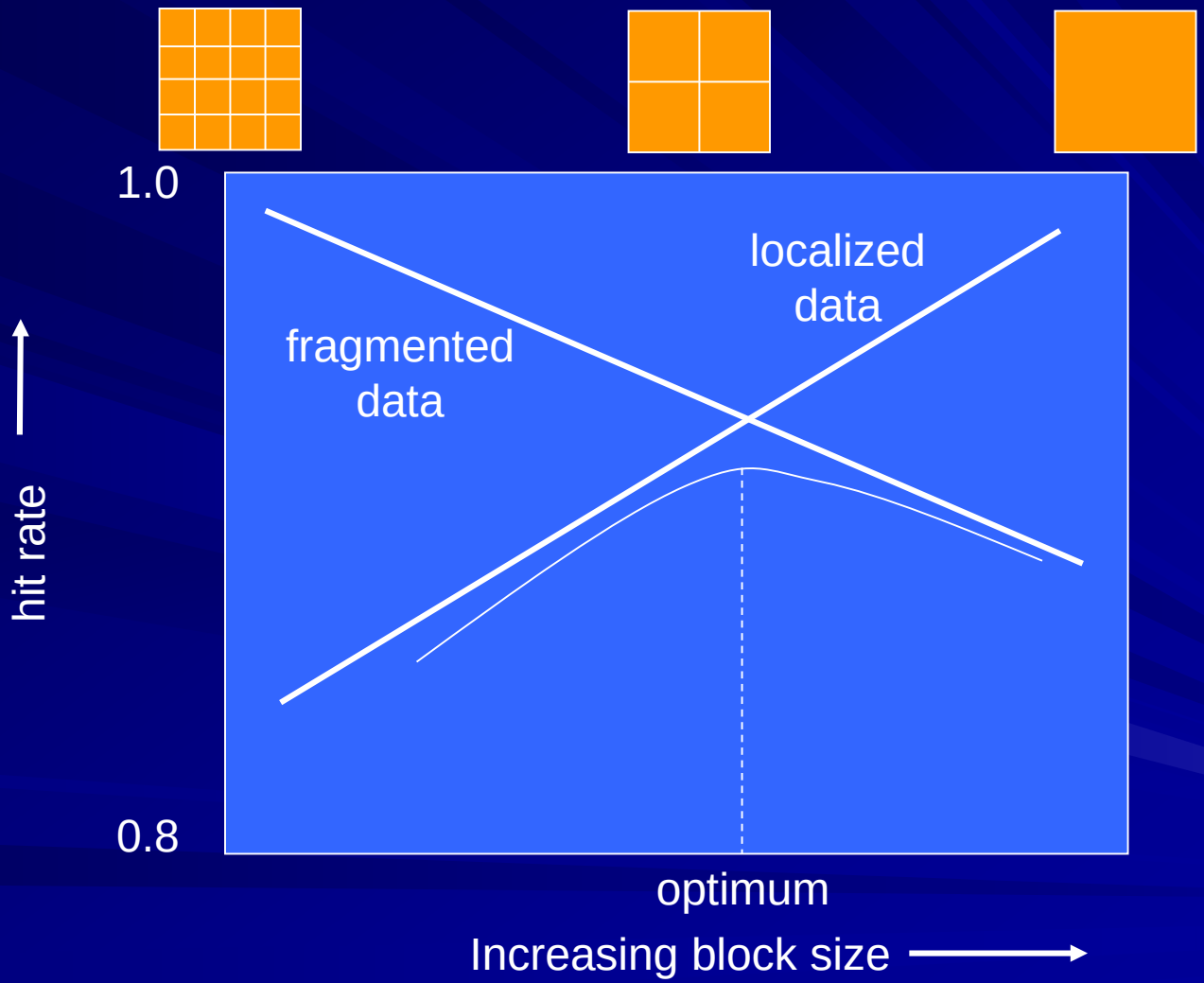




Cache Size

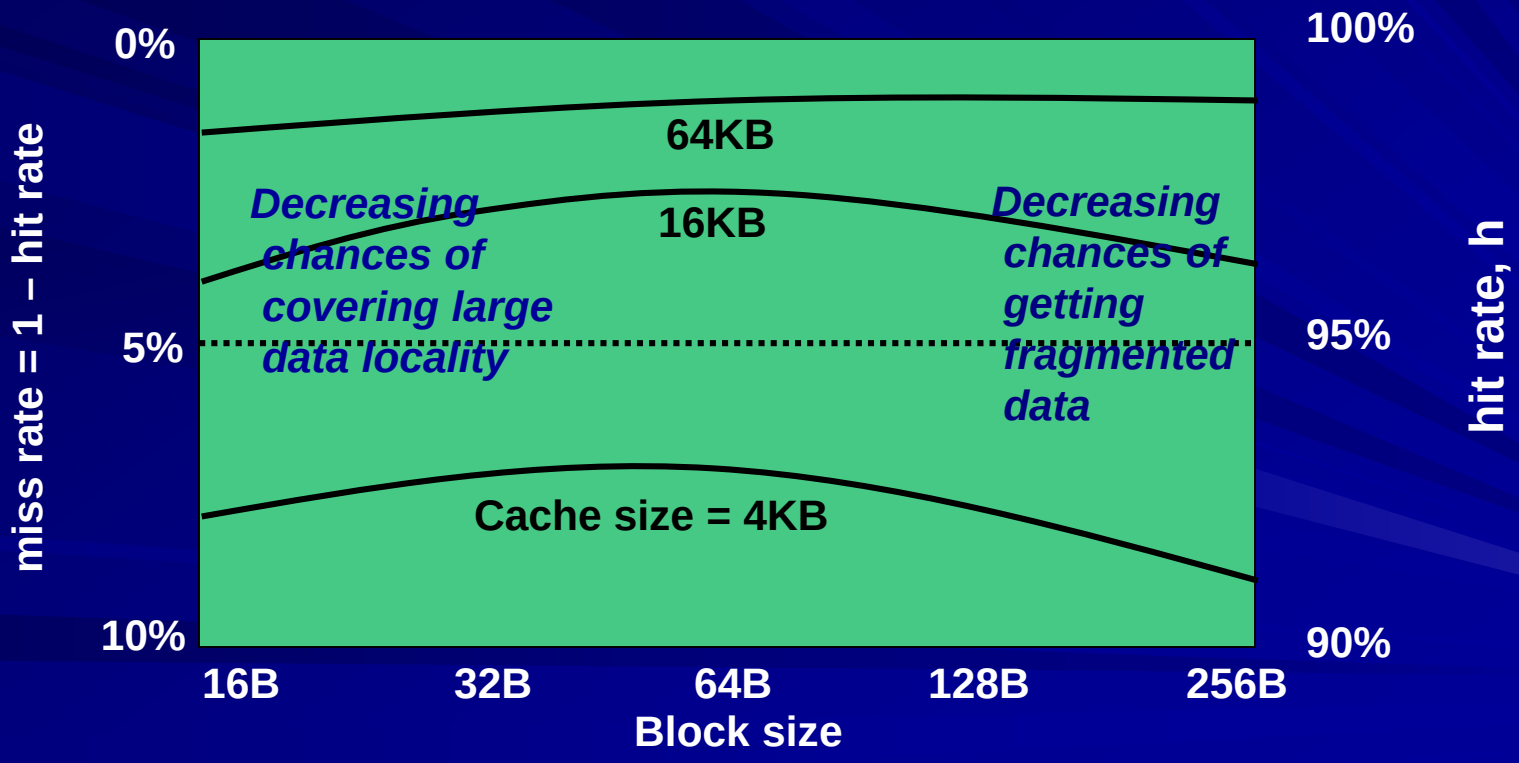


Block Size


1.0
0.8
hit rate
fragmented data
localized data
optimum
Increasing block size

Increasing Hit Rate

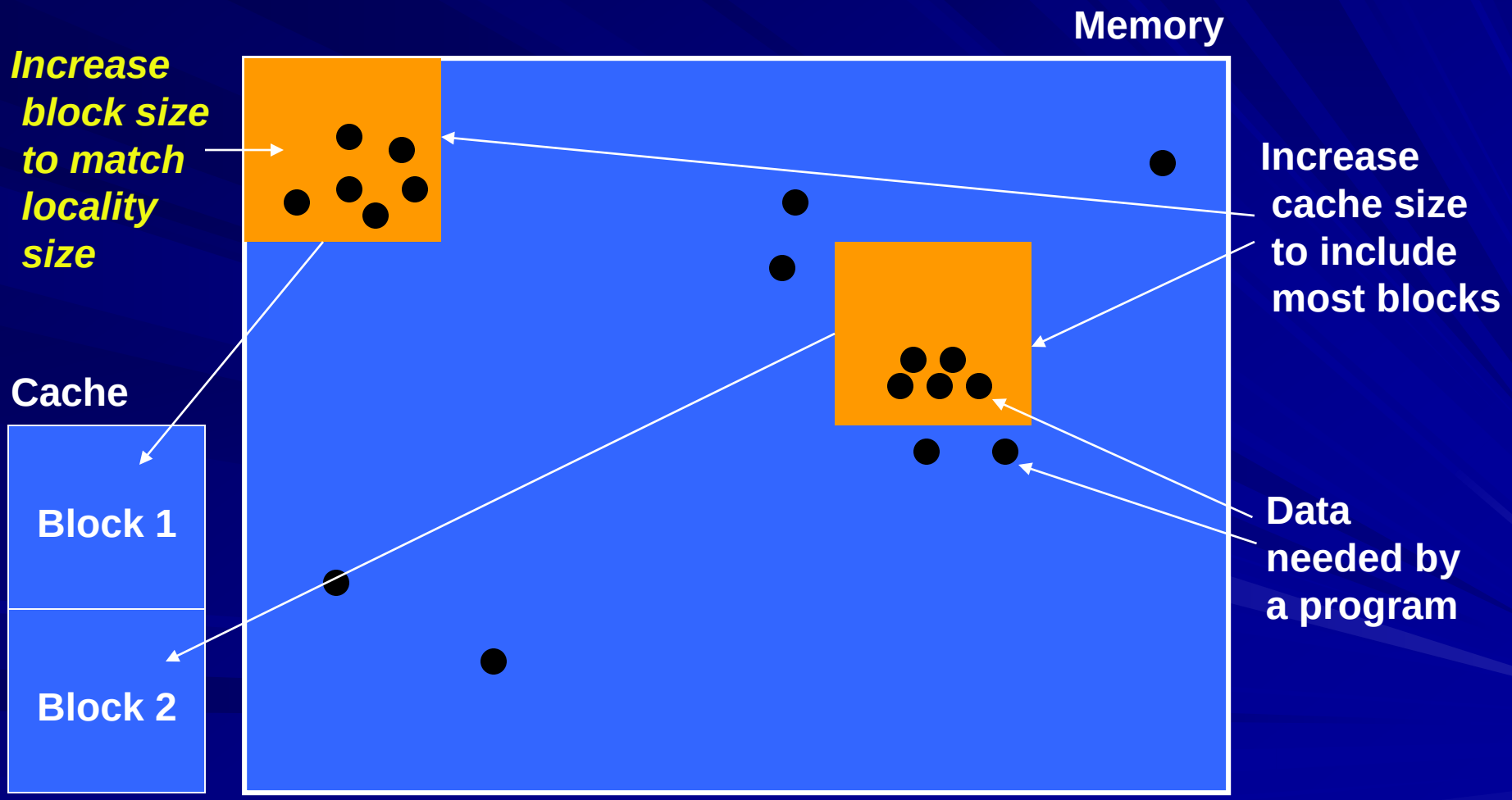
- Hit rate increases with cache size.
- Hit rate mildly depends on block size.



Recall The Locality Principle

- A program tends to access data that form a physical cluster in the memory – multiple accesses may be made within the same block.
- Physical localities are temporal and may shift over longer periods of time – data not used for some time is less likely to be used in the future. Upon miss, the *least recently used* (LRU) block can be overwritten by a new block.
- P. J. Denning, "The Locality Principle," *Communications of the ACM*, vol. 48, no. 7, pp. 19-24, July 2005.

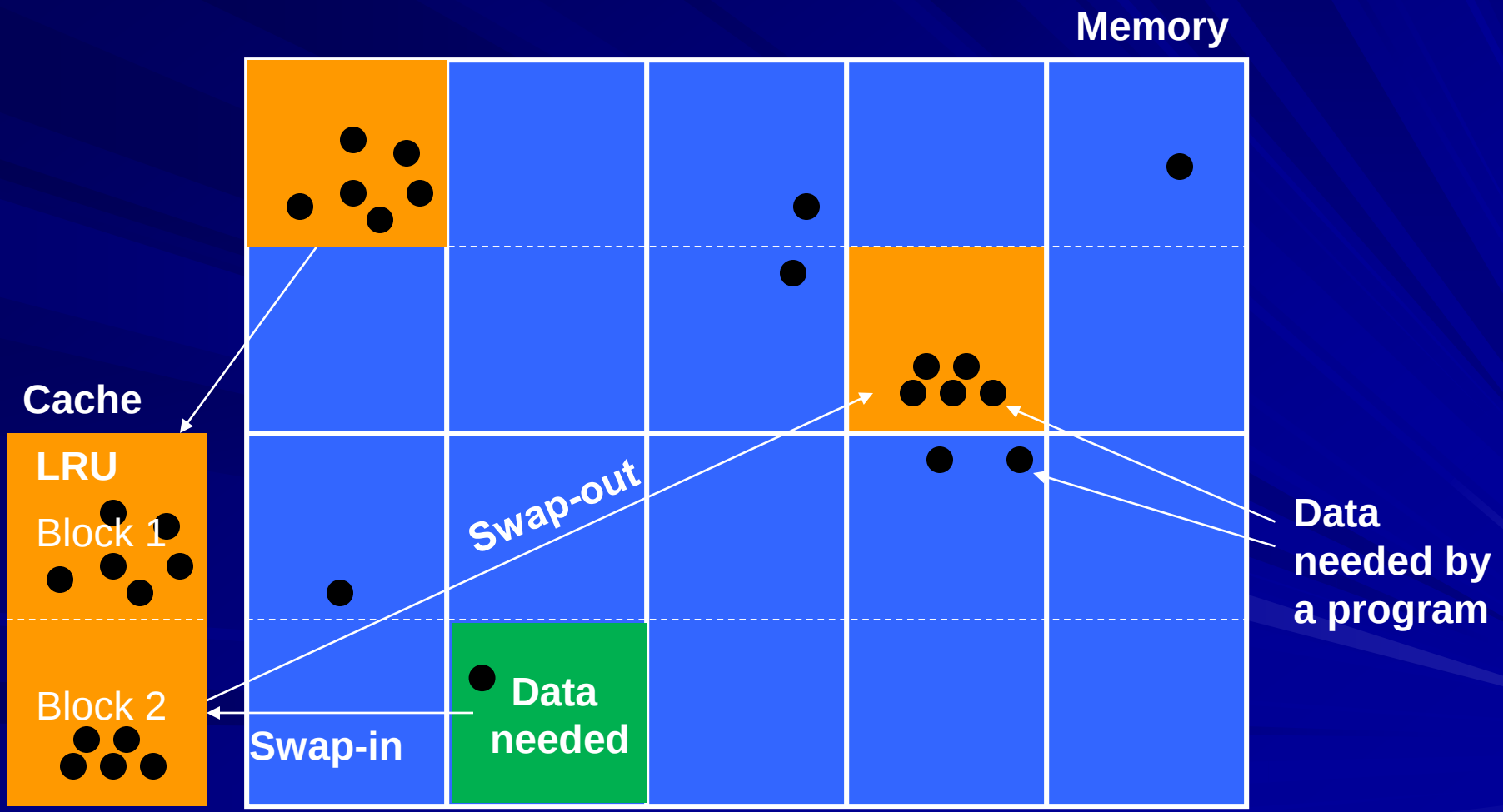
Data Locality, Cache, Blocks



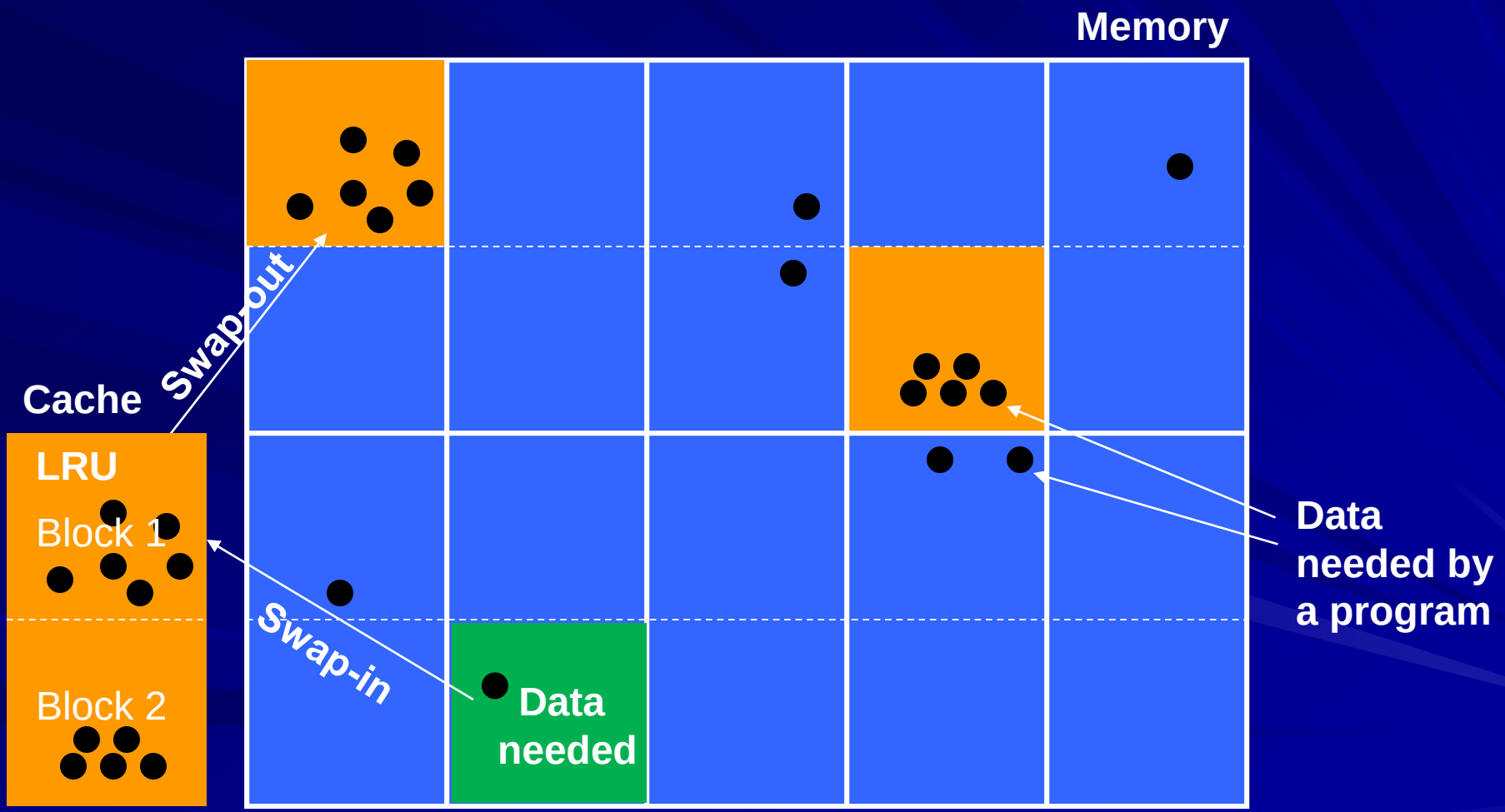
Types of Caches

- Direct-mapped cache
  - Partitions of size of cache in the memory
  - Each partition subdivided into blocks
- Set-associative cache
  - s

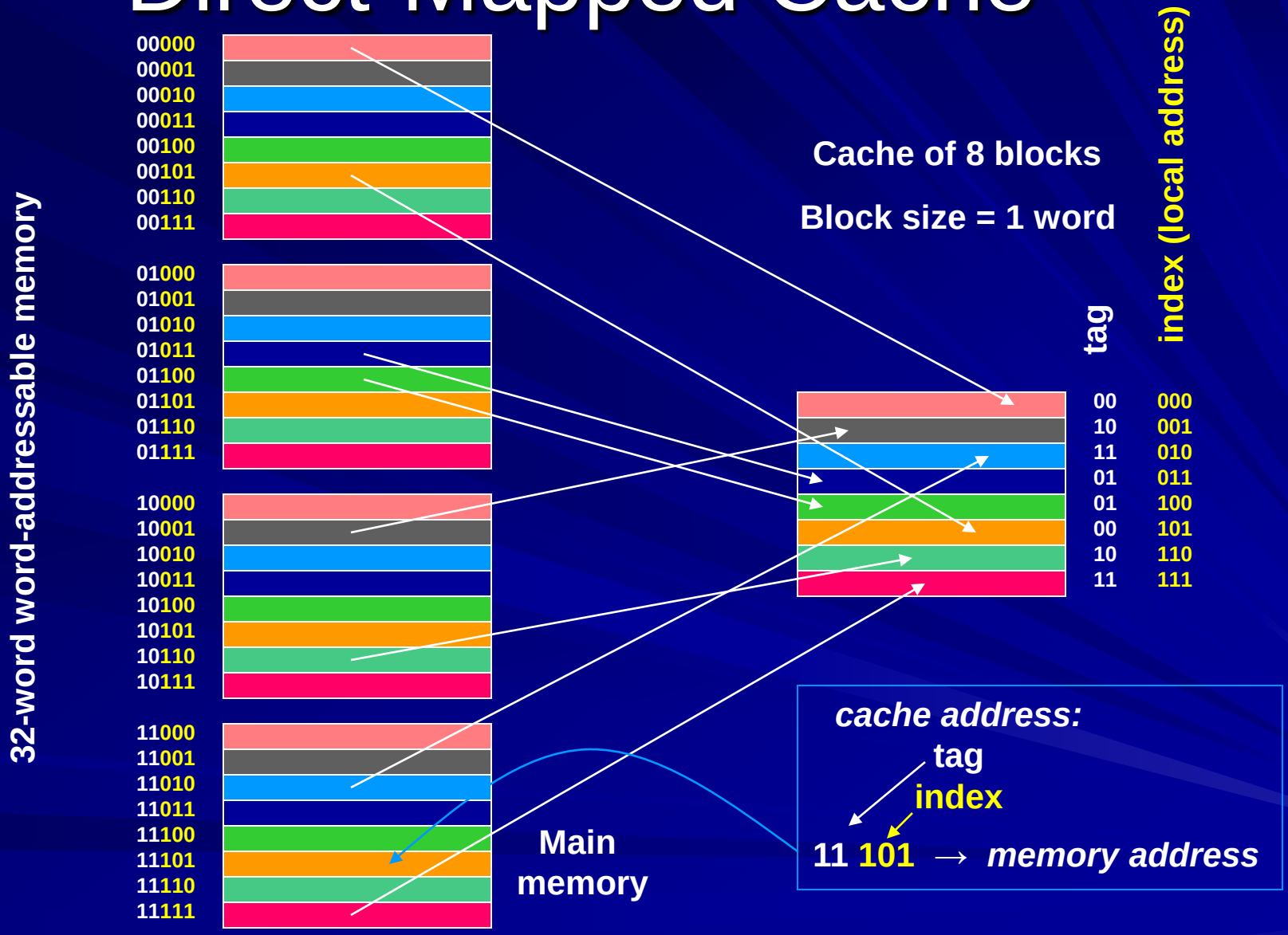
Direct-Mapped Cache



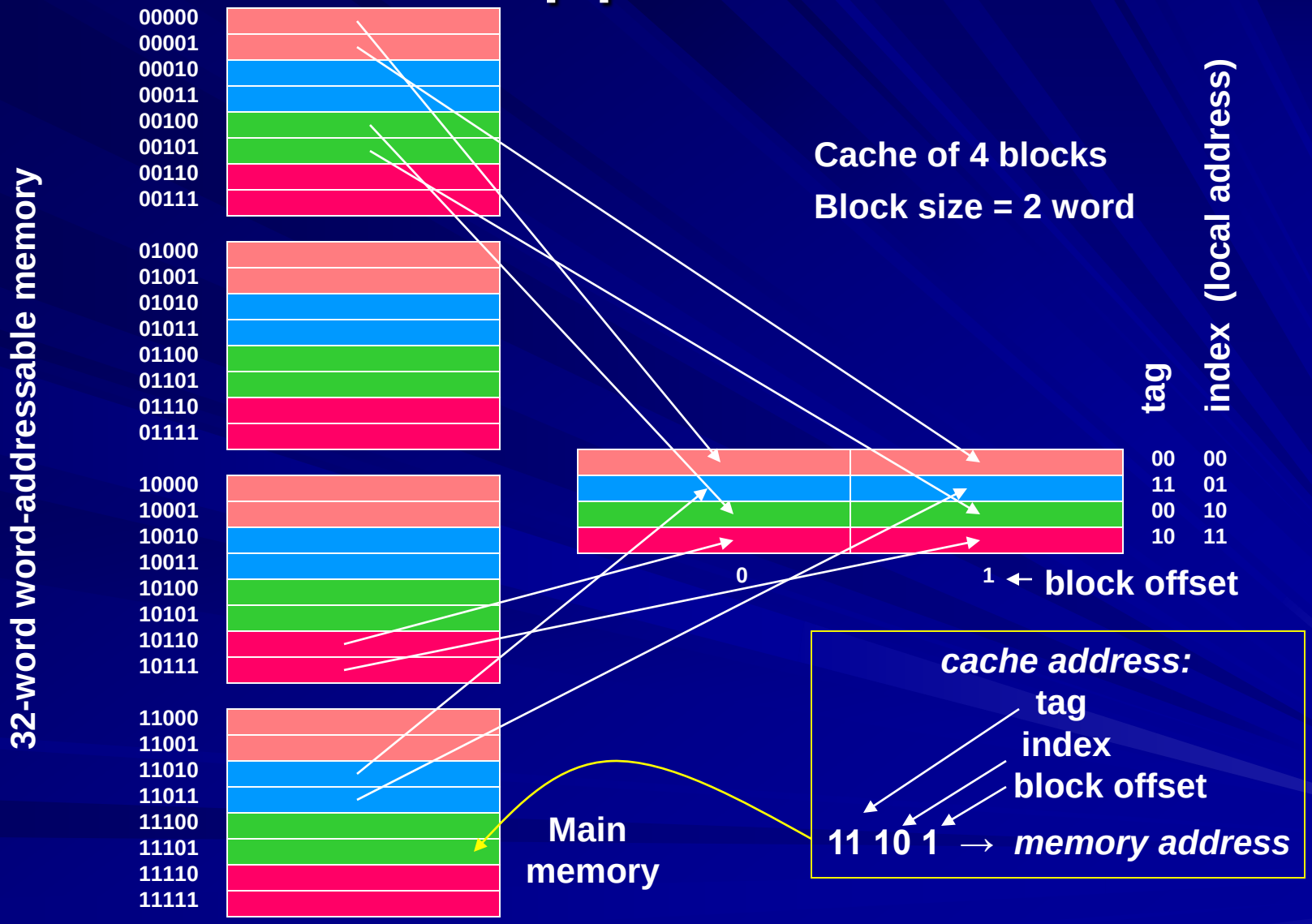
Set-Associative Cache



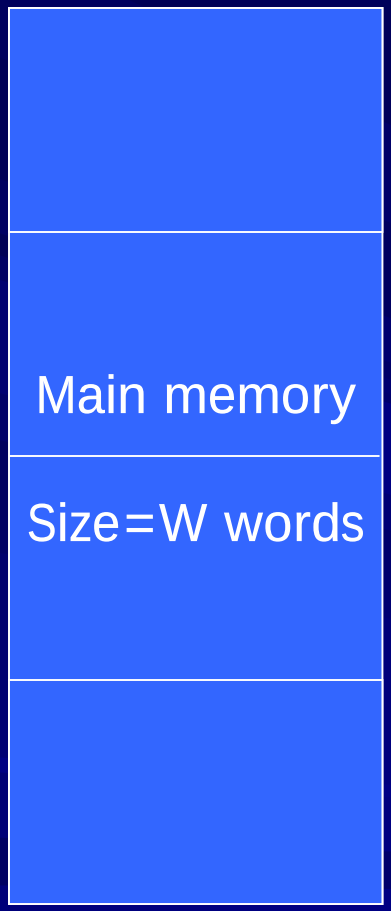
Direct-Mapped Cache



Direct-Mapped Cache



Number of Tag and Index Bits



Cache Size
= w words

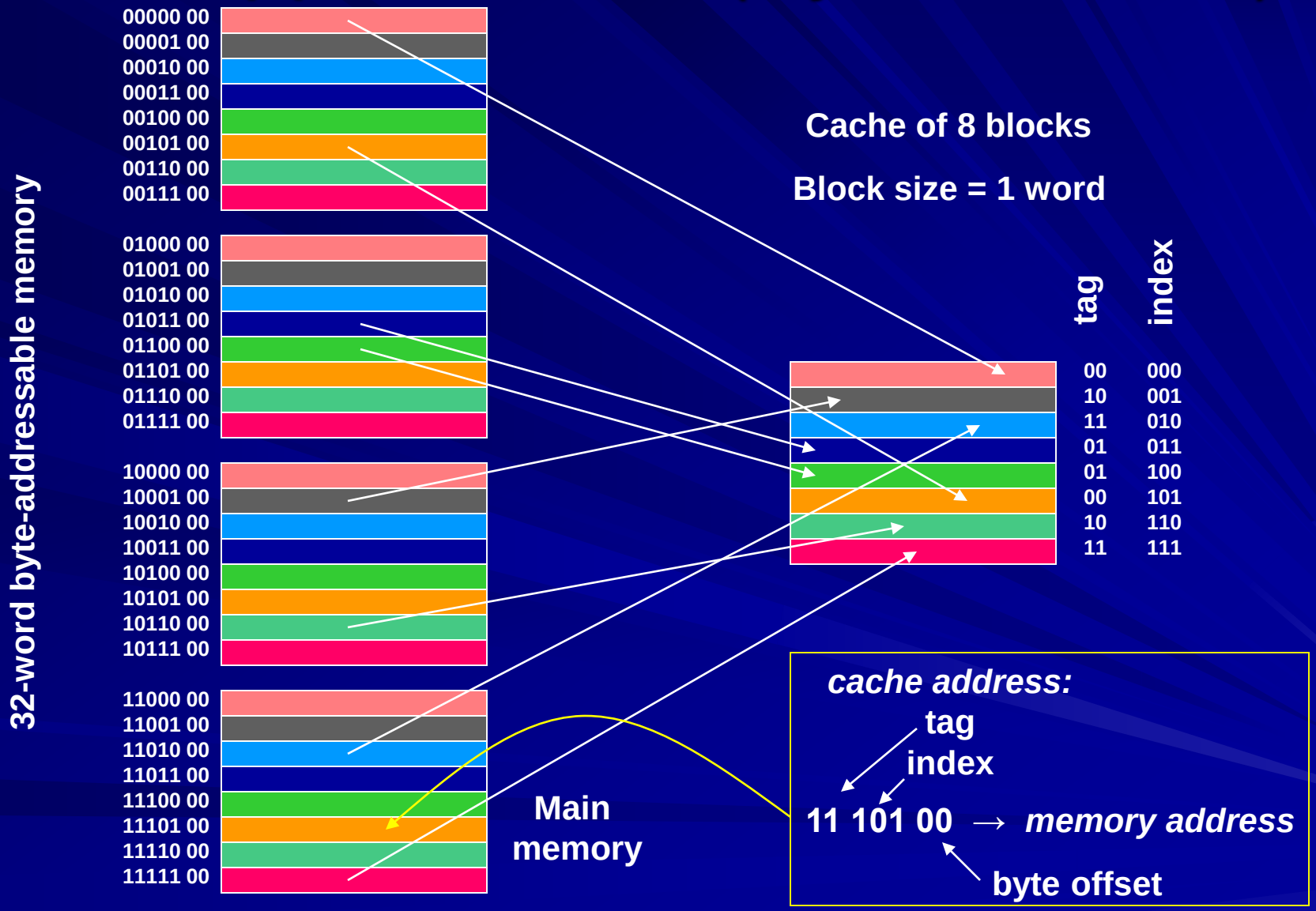
Each word in cache has unique index (local addr.)
Number of index bits = $\log_2 w$

Index bits are shared with block offset when a block contains more words than 1

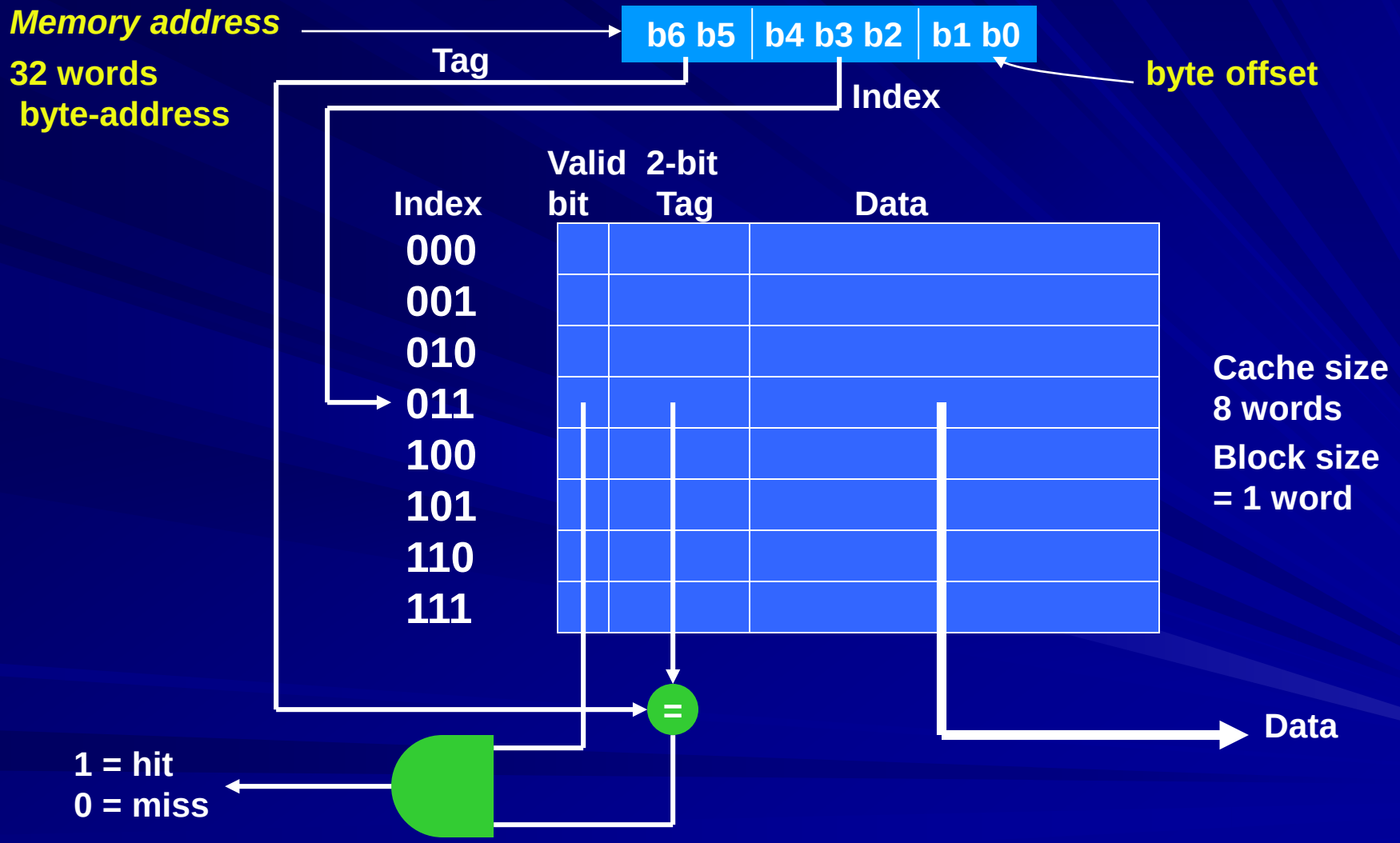
Assume partitions of w words each in the main memory.

W/w such partitions, each identified by a tag
Number of tag bits = $\log_2(W/w)$

Direct-Mapped Cache (Byte Address)



Finding a Word in Cache



How Many Bits In A Cache?

- Consider a main memory:
  - 32 words; byte address is 7 bits wide: b6 b5 b4 b3 b2 b1 b0
  - Each word is 32 bits wide
- Assume that cache block size is 1 word (32 bits data) and it contains 8 blocks.
- Cache requires, for each word:
  - 2 bit *tag*, and one *valid bit*
  - Total storage needed in cache

    = #blocks in cache × (data bits/block + tag bits + valid bit)

    = 8 (32+2+1) = 280 bits

Physical storage/Data storage = 280/256 = 1.094

A More Realistic Cache

- Consider 4 GB, byte-addressable main memory:
  - 1Gwords; byte address is 32 bits wide: b31…b16 b15…b2 b1 b0
  - Each word is 32 bits wide
- Assume that cache block size is 1 word (32 bits data) and it contains 64 KB data, or 16K words, i.e., 16K blocks.
- Number of cache index bits = 14, because 16K = 2^{14}
  - Tag size = 32 – byte offset – #index bits = 32 – 2 – 14 = 16 bits
- Cache requires, for each word:
  - 16 bit *tag*, and one *valid bit*
  - Total storage needed in cache

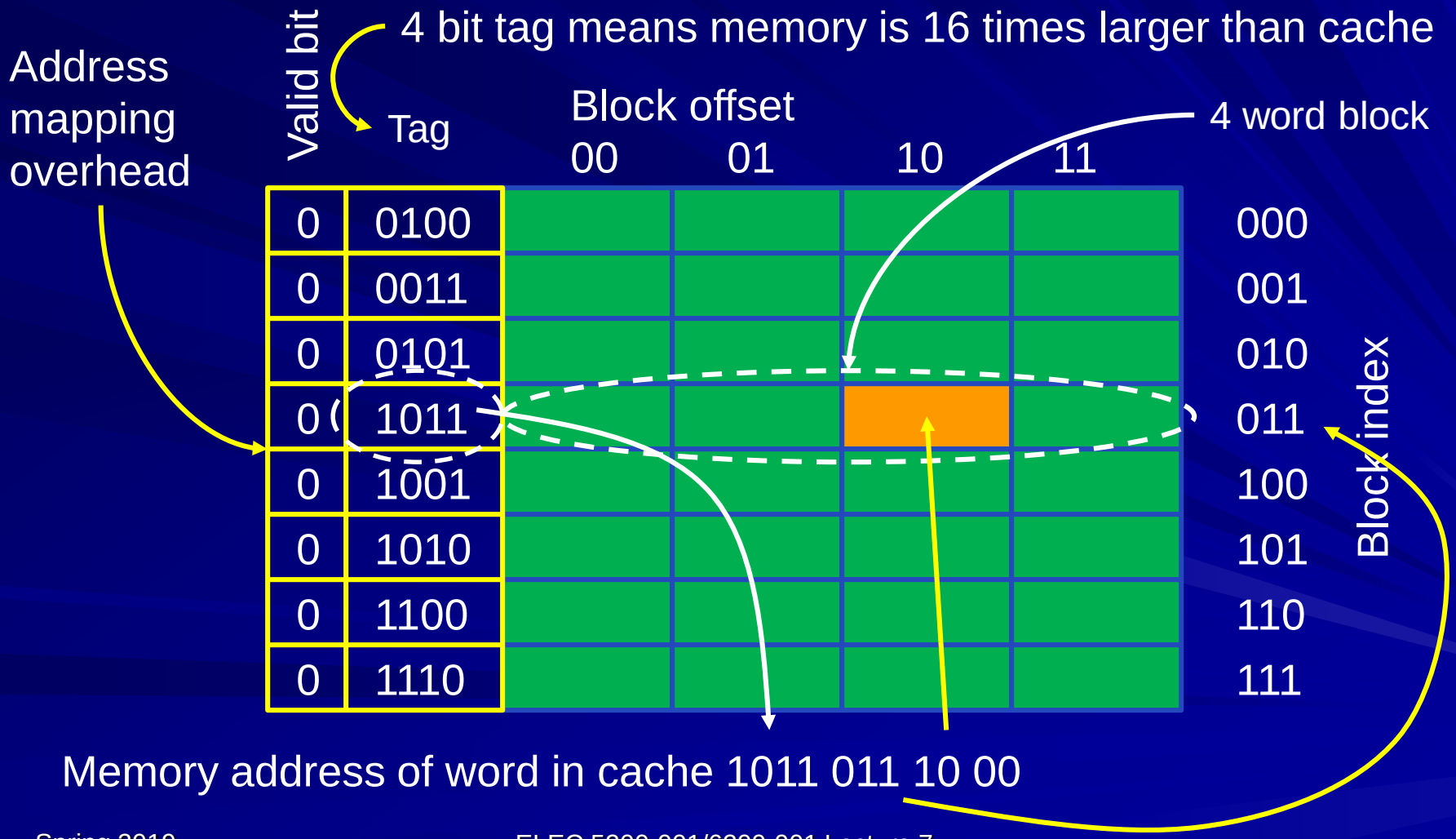
  = #blocks in cache × (data bits/block + tag size + valid bits)

  = 2^{14}(32+16+1) = 16×2^{10}×49 = 784×2^{10} bits = 784 Kb = 98 KB

Physical storage/Data storage = 98/64 = 1.53

But, need to increase the block size to match the size of locality.

Data Organization in Cache

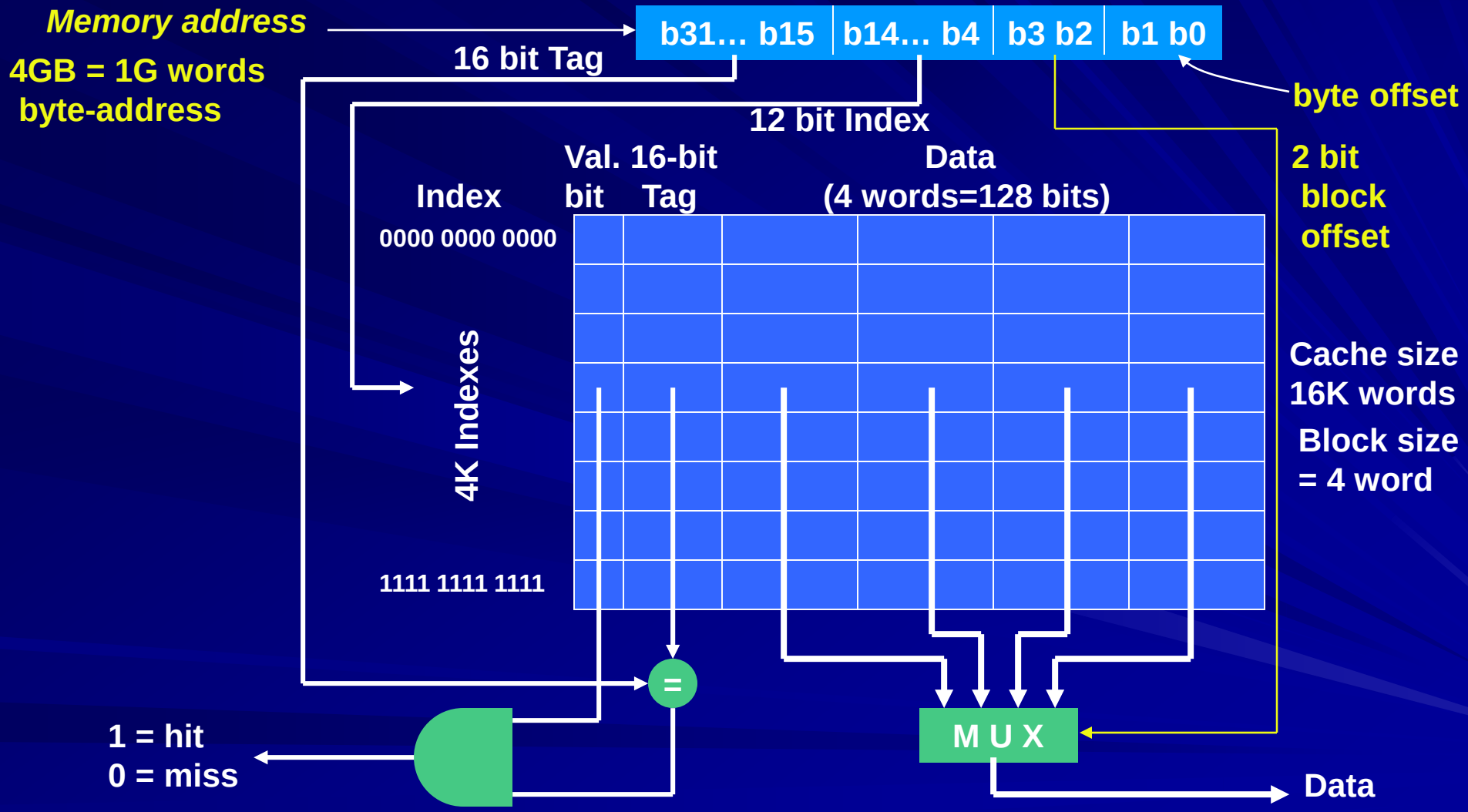


Cache Bits for 4-Word Block

- Consider 4 GB, byte-addressable main memory:
  - 1Gwords; byte address is 32 bits wide: b31…b16 b15…b2 b1 b0
  - Each word is 32 bits wide
- Assume that cache block size is 4 words (128 bits data) and it contains 64 KB data, or 16K words, i.e., 4K blocks.
- Number of cache index bits = 12, because 4K = 2^{12}
  - Tag size = 32 – byte offset – #block offset bits – #index bits
    = 32 – 2 – 2 – 12 = 16 bits
- Cache requires, for each word:
  - 16 bit *tag*, and one *valid bit*
  - Total storage needed in cache
    = #blocks in cache × (data bits/block + tag size + valid bit)
    = $2^{12}(4\times32+16+1) = 4\times2^{10}\times145 = 580\times2^{10}$ bits =580 Kb = 72.5 KB

Physical storage/Data storage = 72.5/64 = 1.13

Using Larger Cache Block (4 Words)



Handling a Miss

- Miss occurs when data at the required memory address is not found in cache.
- Controller actions:
  - Stall pipeline
  - Freeze contents of all registers
  - Activate a separate cache controller
    - If cache is full
      - **select the least recently used (LRU) block in cache for over-writing**
      - **If selected block has inconsistent data, take proper action**
    - Copy the block containing the requested address from memory
  - Restart Instruction

Miss During Instruction Fetch

- Send original PC value (PC – 4) to the memory.
- Instruct main memory to perform a read and wait for the memory to complete the access.
- Write cache entry.
- Restart the instruction whose fetch failed.

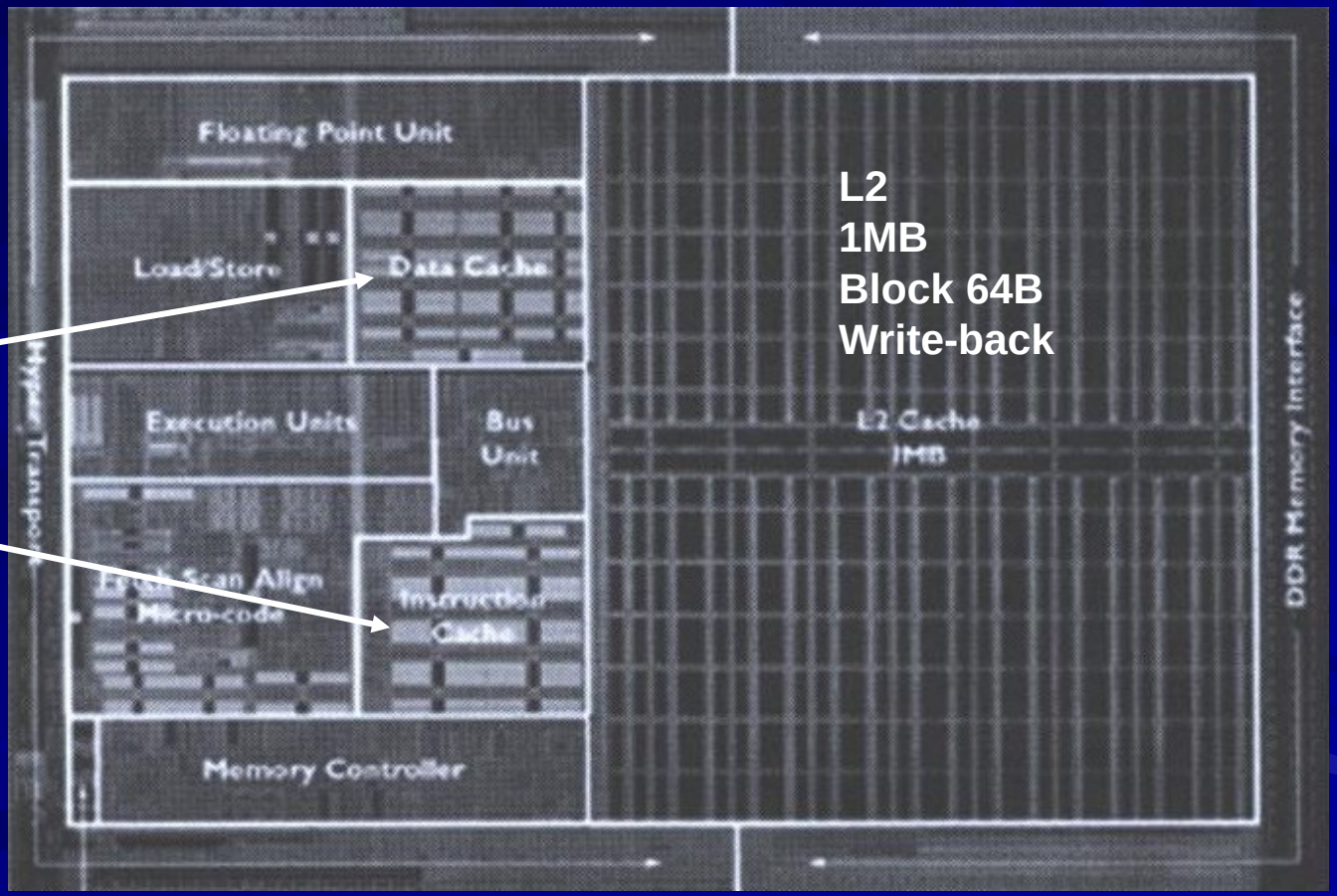
Writing to Memory

- Cache and memory become *inconsistent* when data is written into cache, but not to memory – *the cache coherence problem.*
- Strategies to handle inconsistent data:
  - Write-through
    - Write to memory and cache simultaneously always.
    - Write to memory is ~100 times slower than to cache.
  - Write buffer
    - Write to cache and to buffer for writing to memory.
    - If buffer is full, the processor must wait.

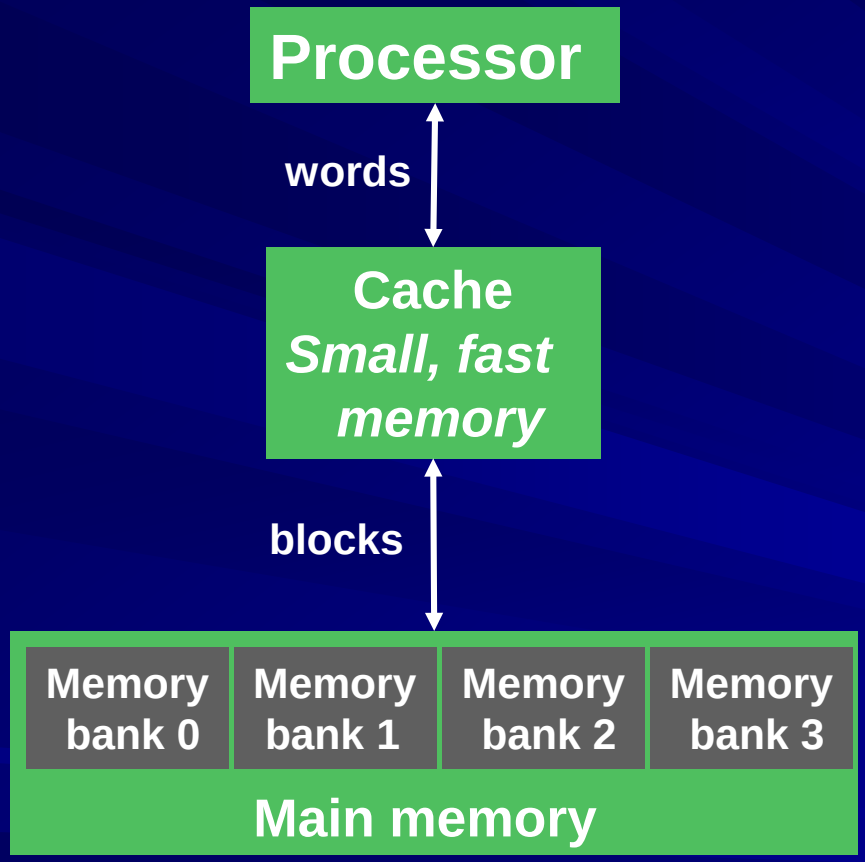
Writing to Memory: Write-Back

- Write-back (or copy back) writes only to cache but sets a “dirty bit” in the block where write is performed.
- When a block with dirty bit “on” is to be overwritten in the cache, it is first written to the memory.

AMD Opteron Microprocessor

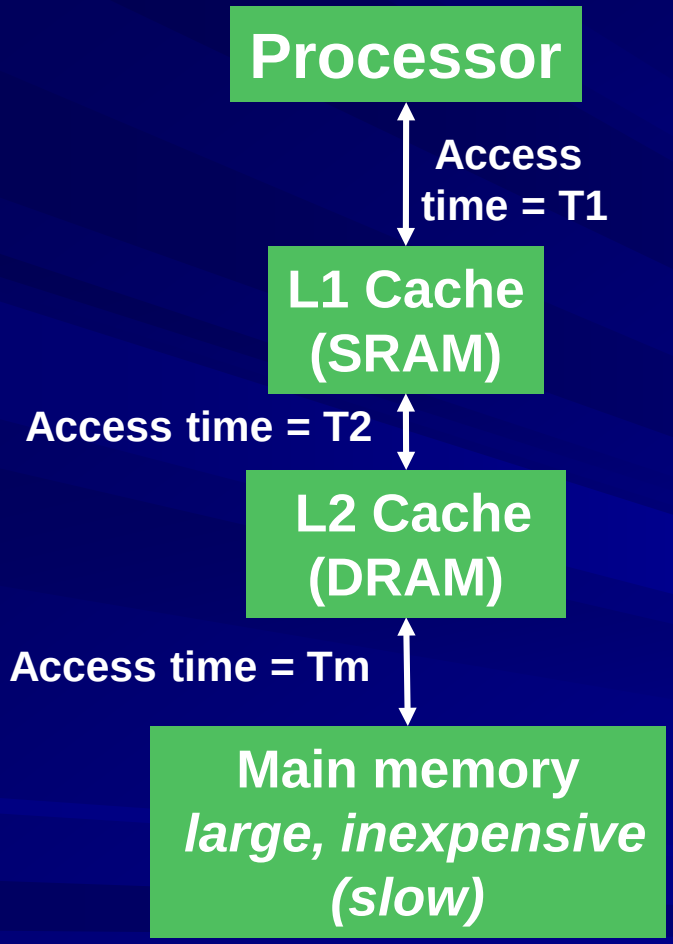


Interleaved Memory



- Reduces miss penalty.
- Memory designed to read words of a block simultaneously in one read operation.
- Example:
  - Cache block size = 4 words
  - Interleaved memory with 4 banks
  - Suppose memory access ~15 cycles
  - Miss penalty = 1 cycle to send address + 15 cycles to read a block + 4 cycles to send data to cache
    = 20 cycles
  - Without interleaving,
    Miss penalty = 65 cycles

Cache Hierarchy



- Average access time

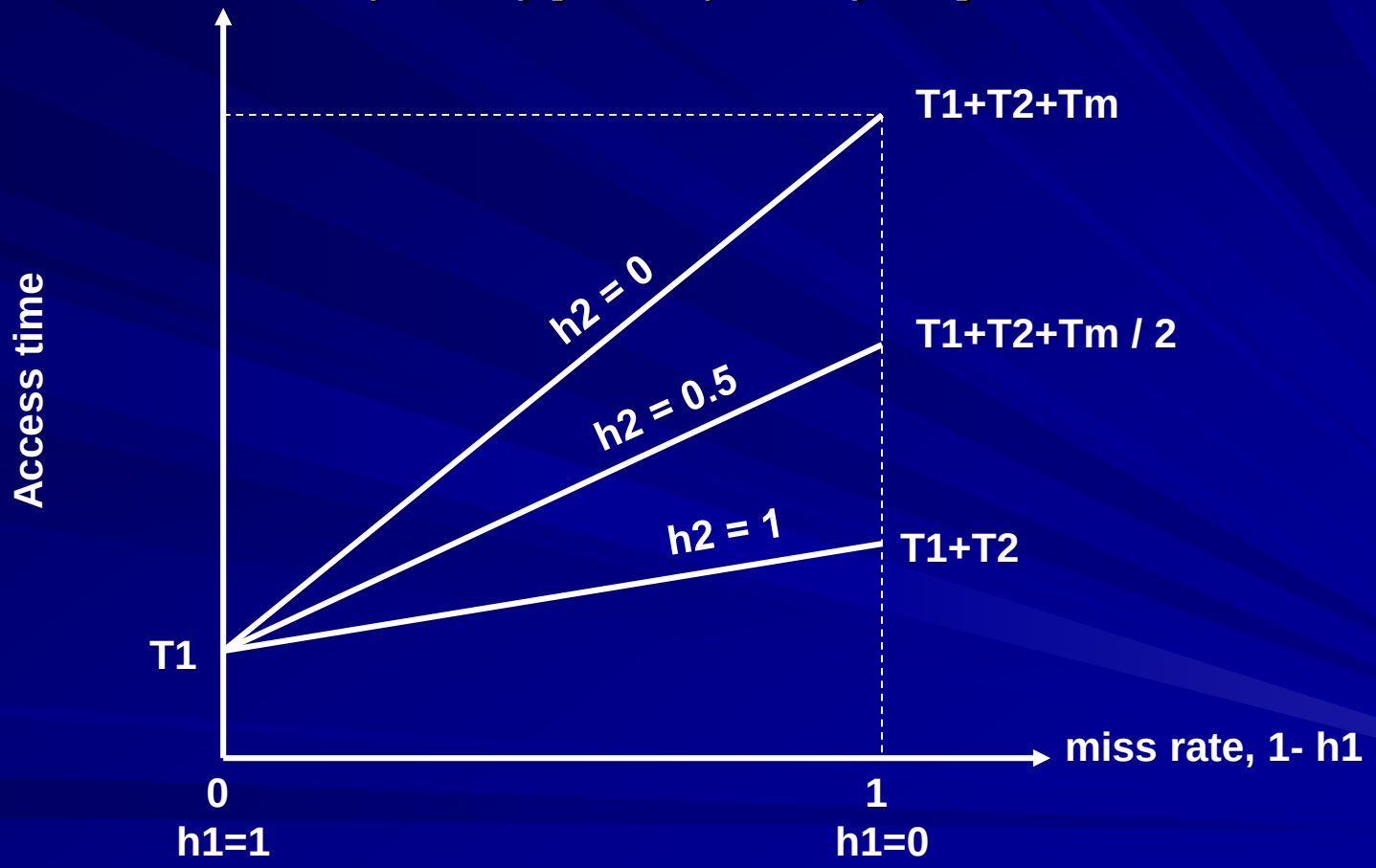
  = T1 + (1 – h1) [T2 + (1 – h2)Tm]

- Where
  - T1 = L1 cache access time (smallest)
  - T2 = L2 cache access time (small)
  - Tm = memory access time (large)
  - h1, h2 = hit rates ($0 \le h1, h2 \le 1$)
- Average access time reduces by adding a cache.

Average Access Time

$$T1 + (1 - h1)\,[\,T2 + (1 - h2)Tm\,]$$

T1 < T2 < Tm



Processor Pipeline Performance Without Cache

- 5GHz processor, cycle time = 0.2ns
- Memory access time = 100ns = 500 cycles
- Ignoring memory access, CPI = 1
- Assuming no memory data access:

  CPI = 1 + # stall cycles

  = 1 + 500 = 501

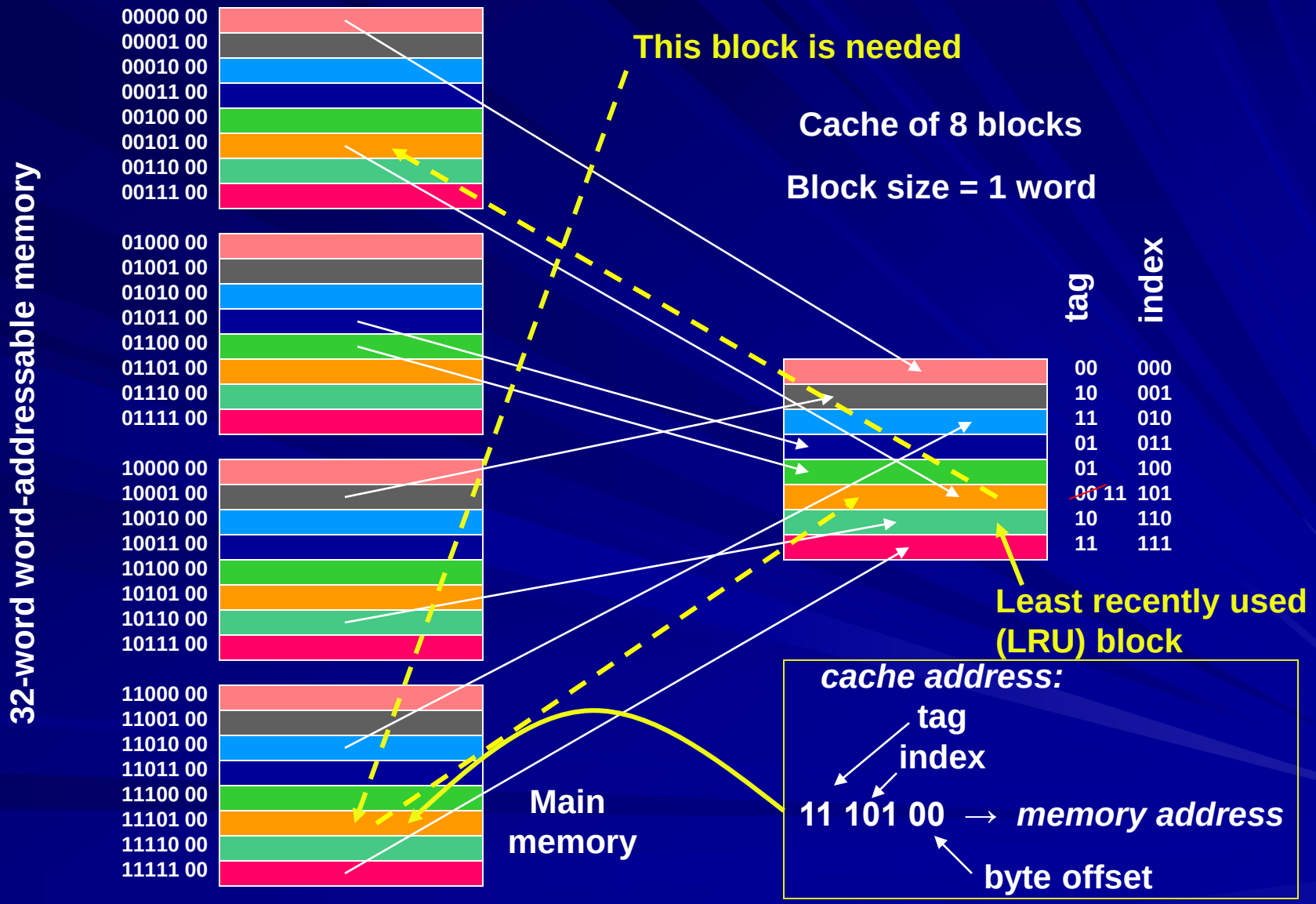
Performance with 1 Level Cache

- Assume hit rate, h1 = 0.95
- L1 access time = 0.2ns = 1 cycle
- CPI = 1 + # stall cycles
  = 1 + 0.05×500
  = 26
- Processor speed increase due to cache
  = 501/26 = 19.3

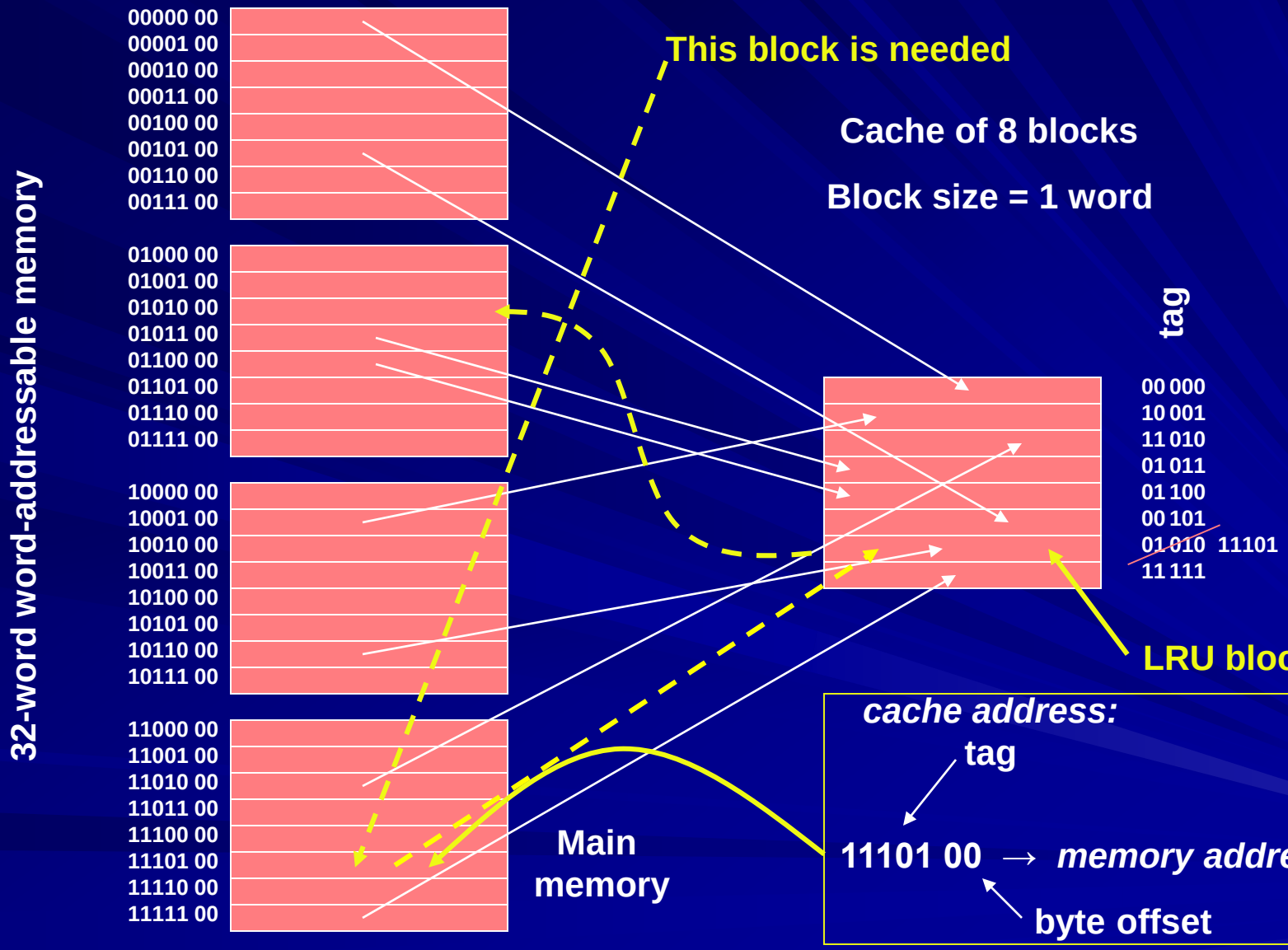
Performance with 2 Level Caches

- Assume:
  - L1 hit rate, h1 = 0.95
  - L2 hit rate, h2 = 0.90
  - L2 access time = 5ns = 25 cycles
- CPI = 1 + # stall cycles
  = 1 + 0.05 (25 + 0.10×500)
  = 1 + 3.75 = 4.75
- Processor speed increase due to 2 level caches
  = 501/4.75 = 105.5
- Speed increase due to L2 cache
  = 26/4.75 = 5.47

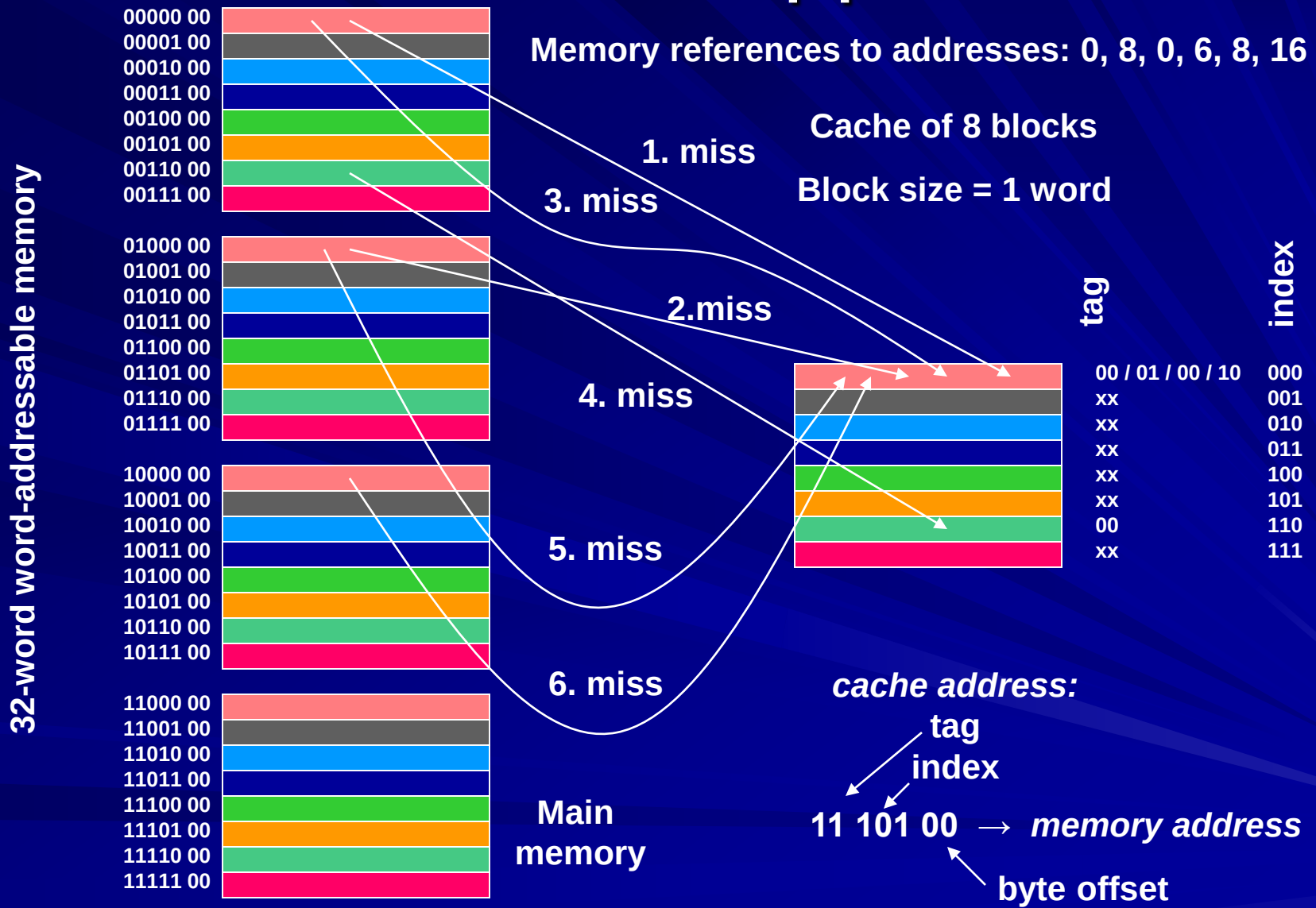
Miss Rate of Direct-Mapped Cache



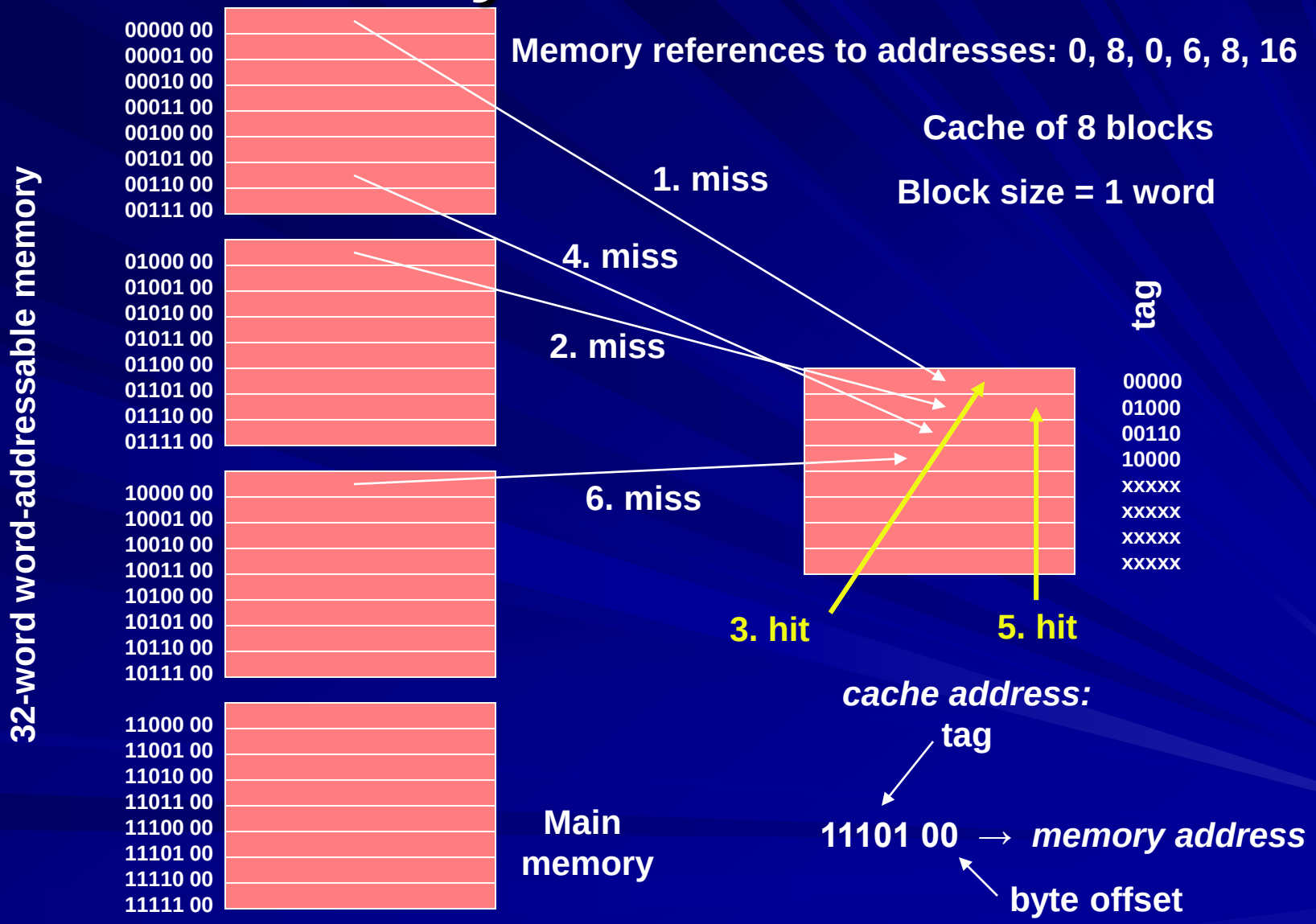
Fully-Associative Cache (8-Way Set Associative)



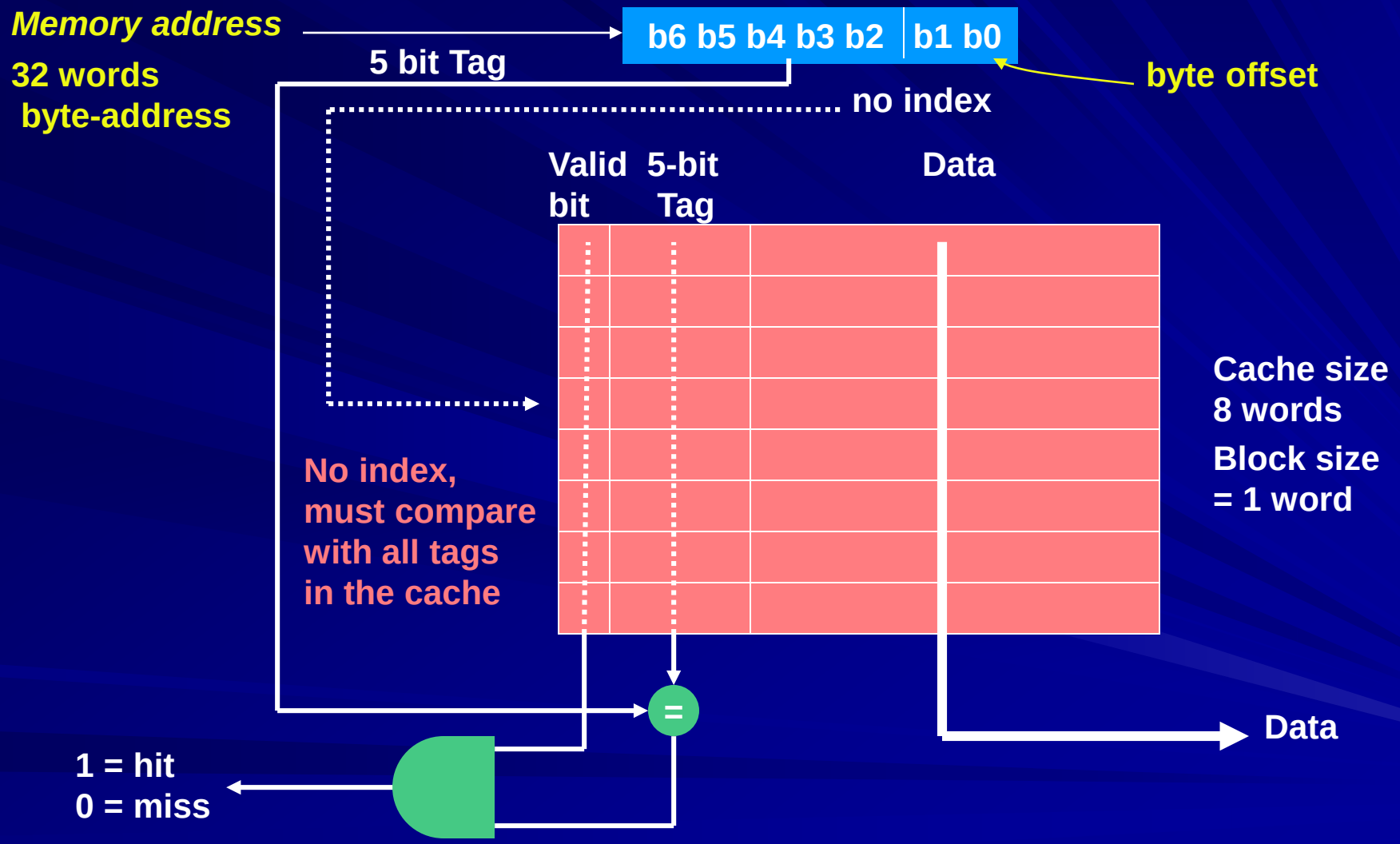
Miss Rate of Direct-Mapped Cache



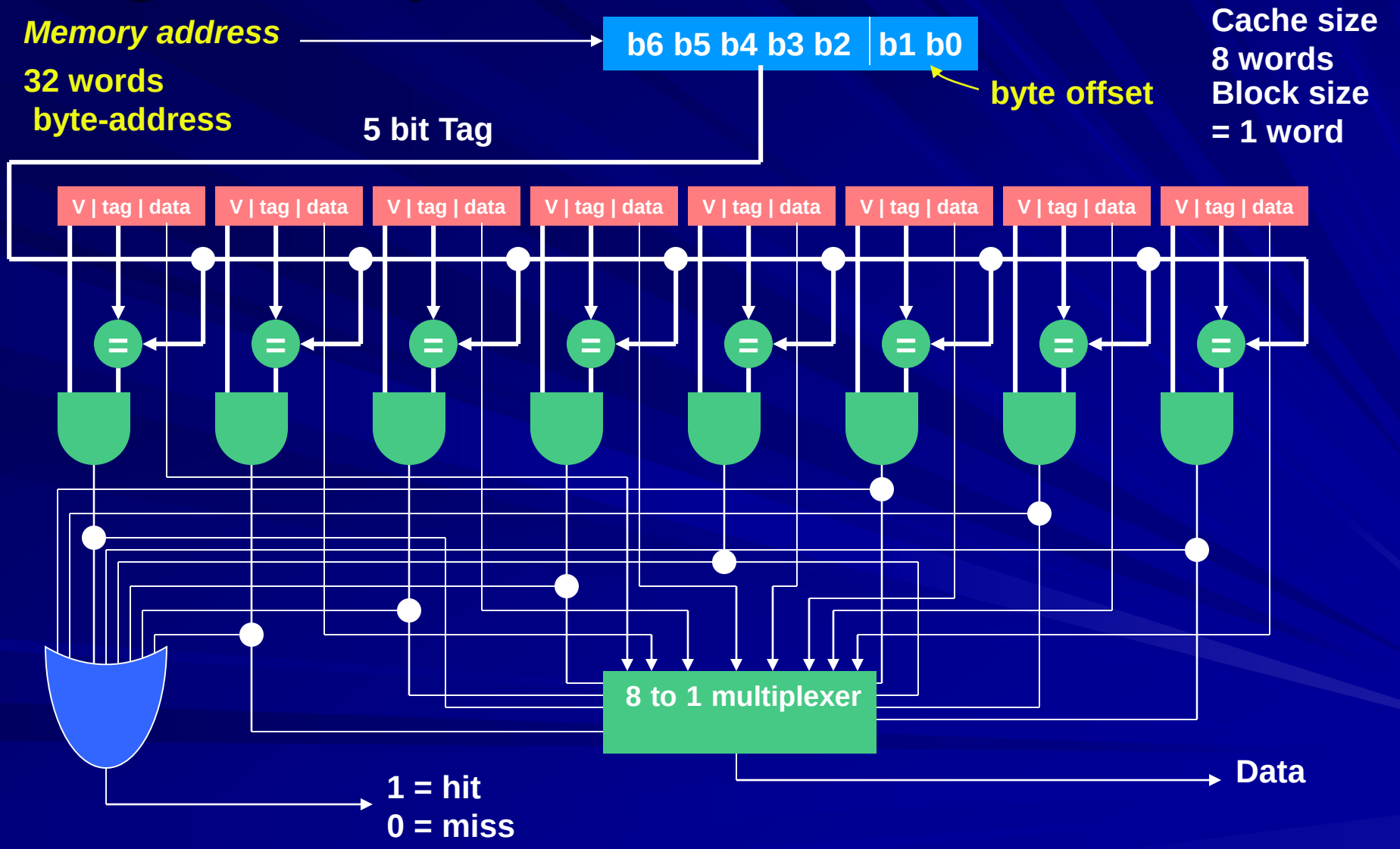
Miss Rate: Fully-Associative Cache



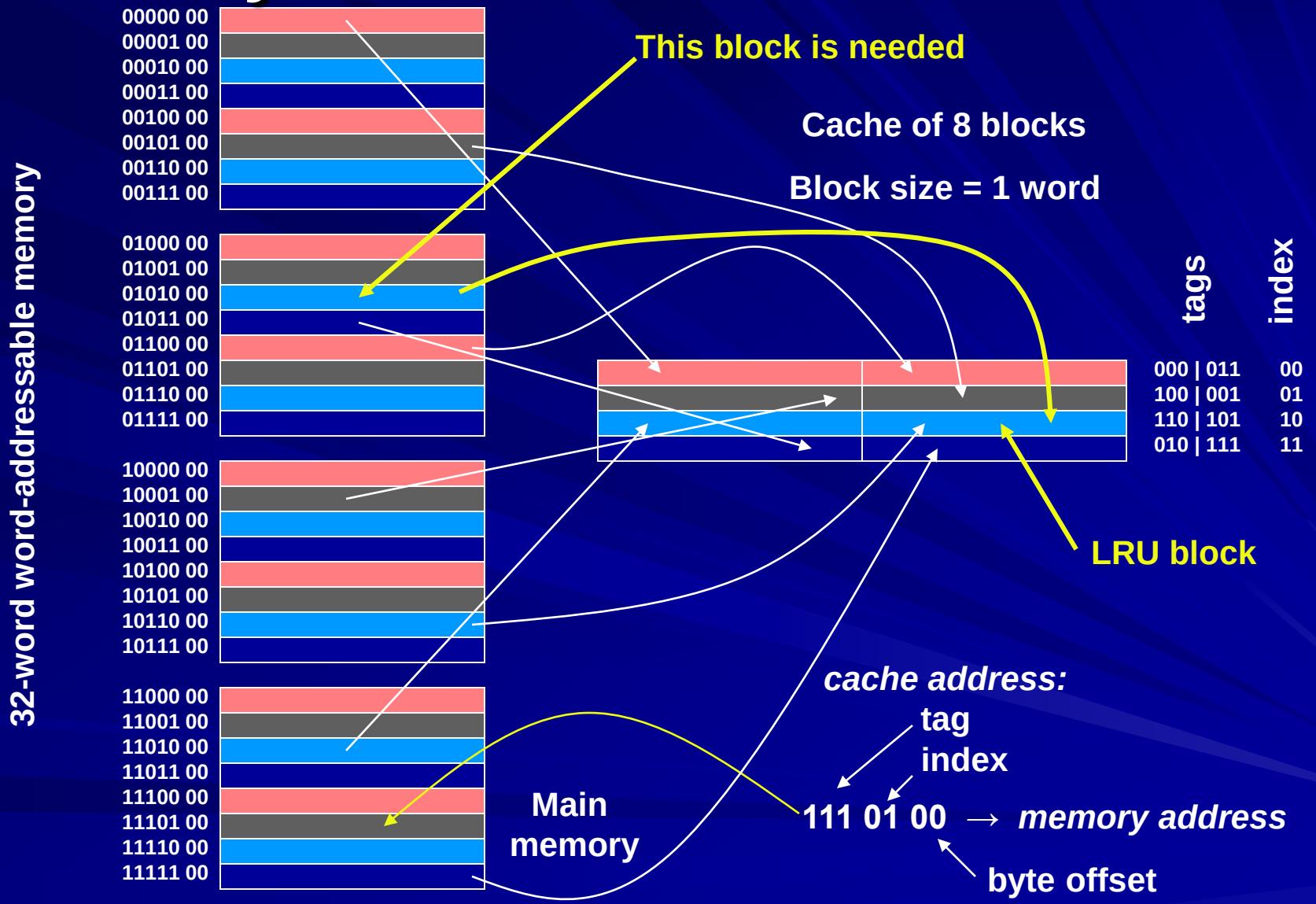
Finding a Word in Associative Cache



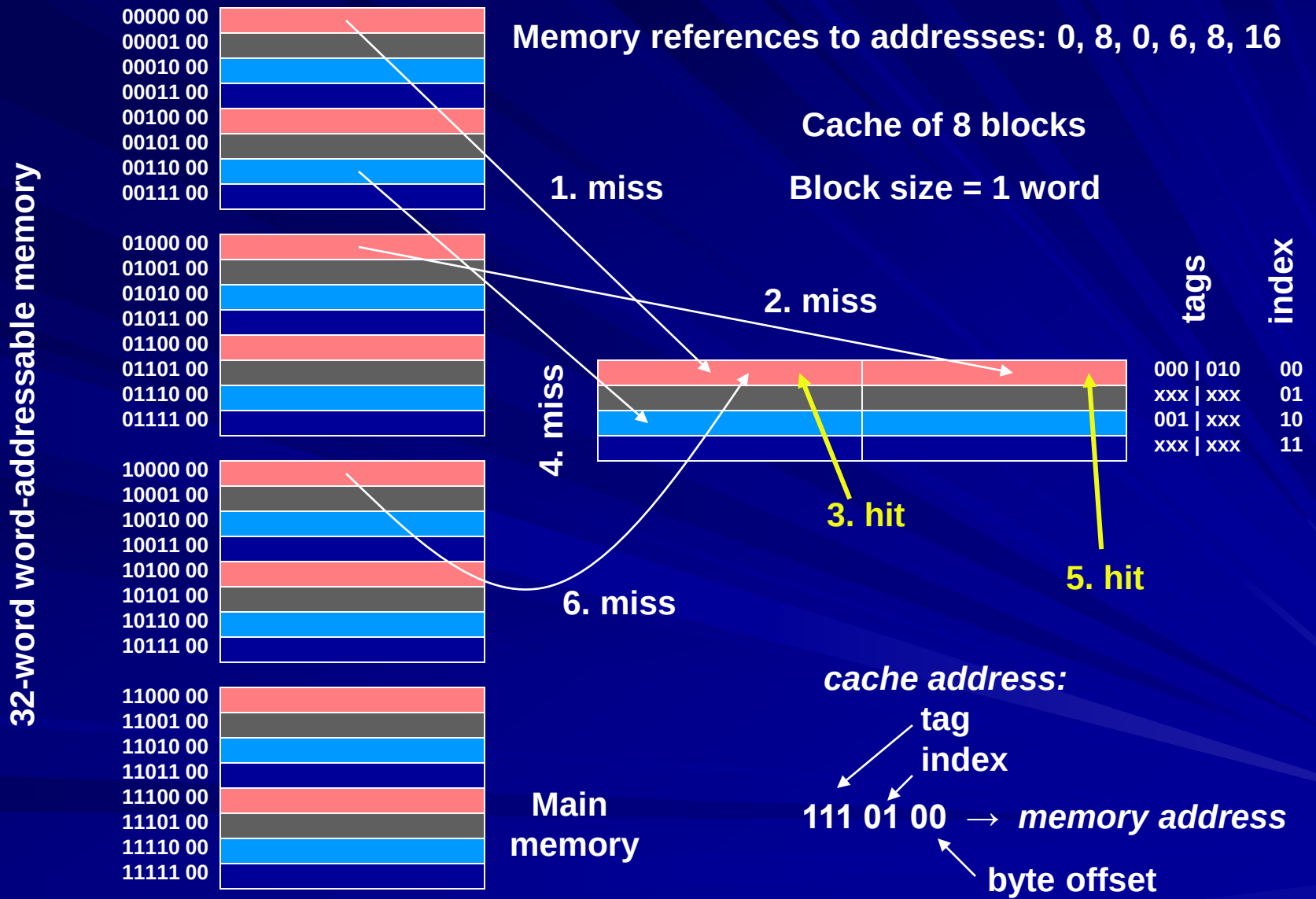
Eight-Way Set-Associative Cache



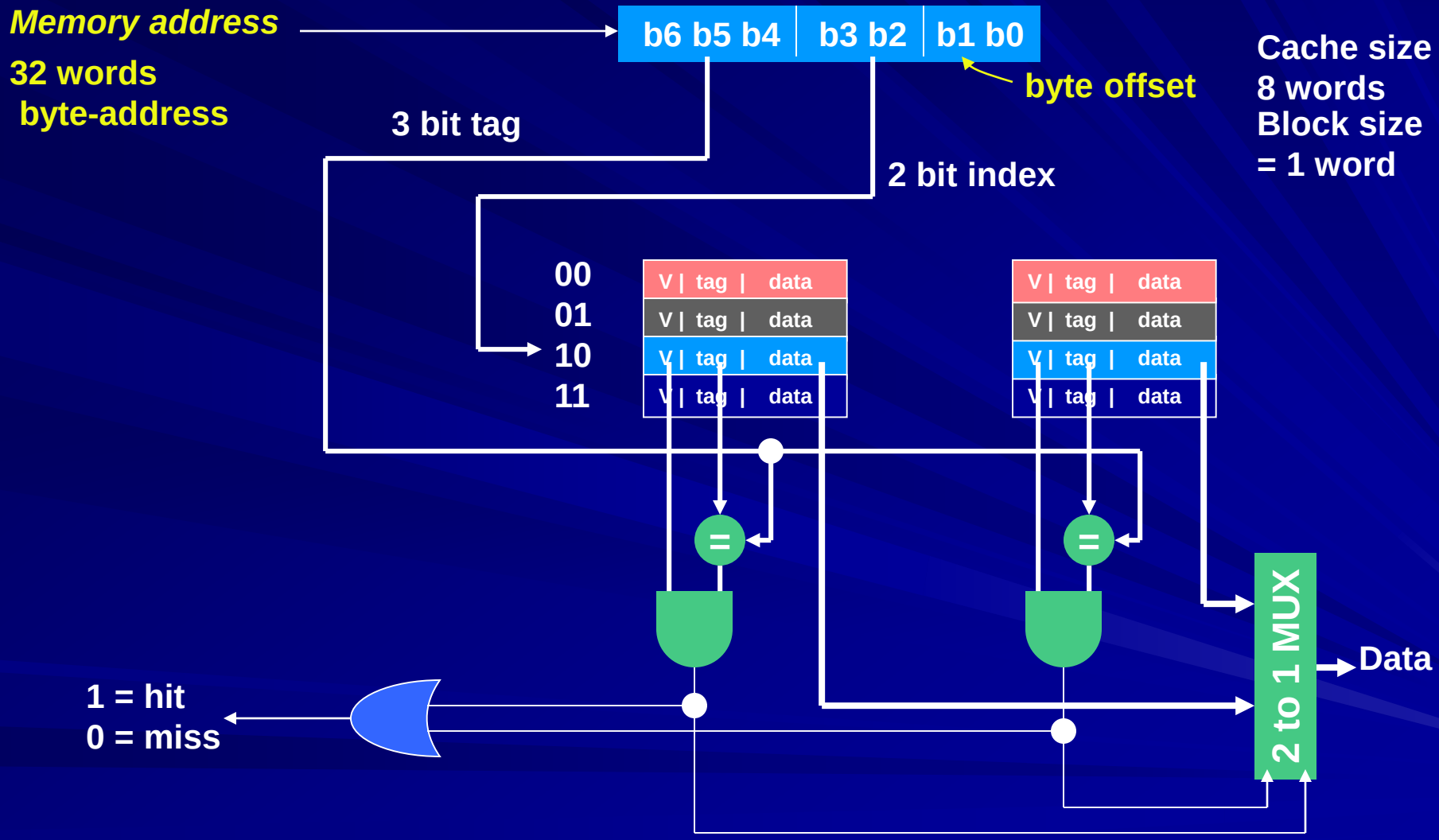
Two-Way Set-Associative Cache



Miss Rate: Two-Way Set-Associative Cache



Two-Way Set-Associative Cache



Cache Miss and Page Fault

Pages

(Write-back, same as in cache)

Main Memory

Processor Cache MMU

All data, organized in Pages (~4KB), accessed by Physical addresses

Cached pages, Page table

Page fault: a required page is not found in main memory

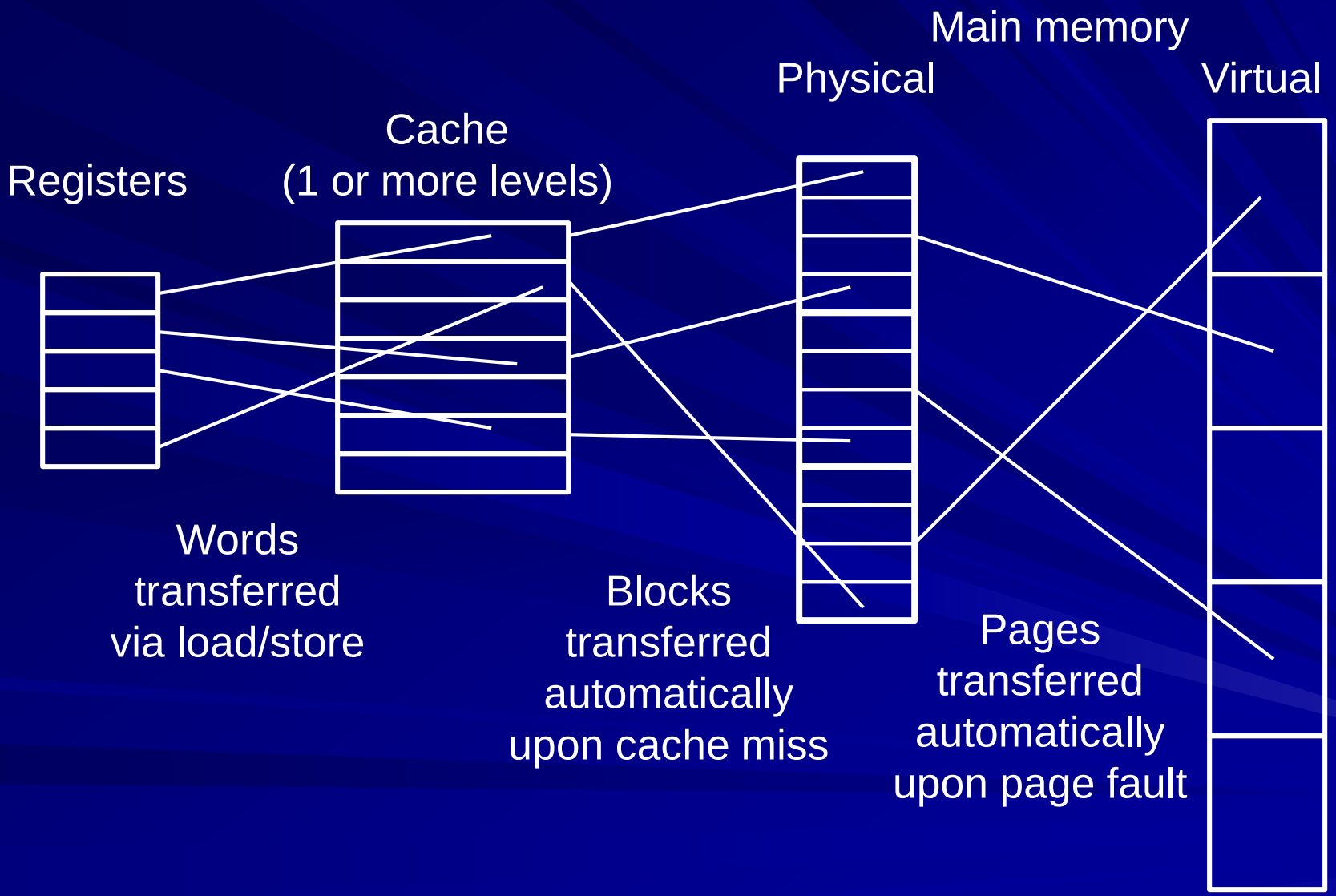
Cache miss: a required block is not found in cache

"Page fault" in virtual memory is similar to "miss" in cache.

Virtual vs. Physical Address

- Processor assumes a virtual memory addressing scheme:
  - Disk is a virtual memory (large, slow)
  - A block of data is called a virtual page
  - An address is called virtual (or logical) address (VA)
- Main memory may have a different addressing scheme:
  - Physical memory consists of caches
  - Memory address is called physical address
  - MMU translates virtual address to physical address
  - Complete address translation table is large and is kept in main memory
  - MMU contains TLB (translation-lookaside buffer), which keeps record of recent address translations.

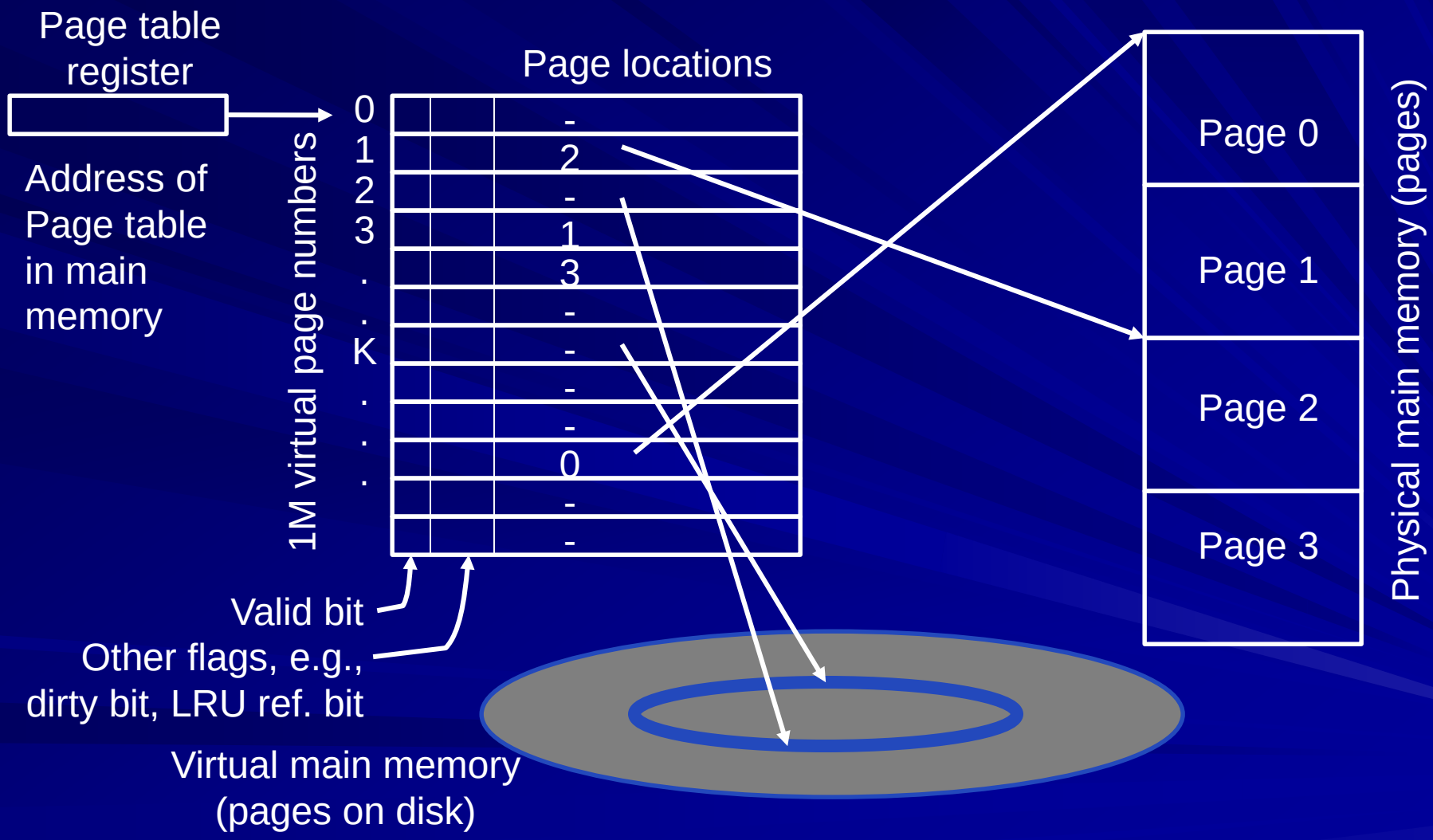
Memory Hierarchy



Memory Hierarchy Example

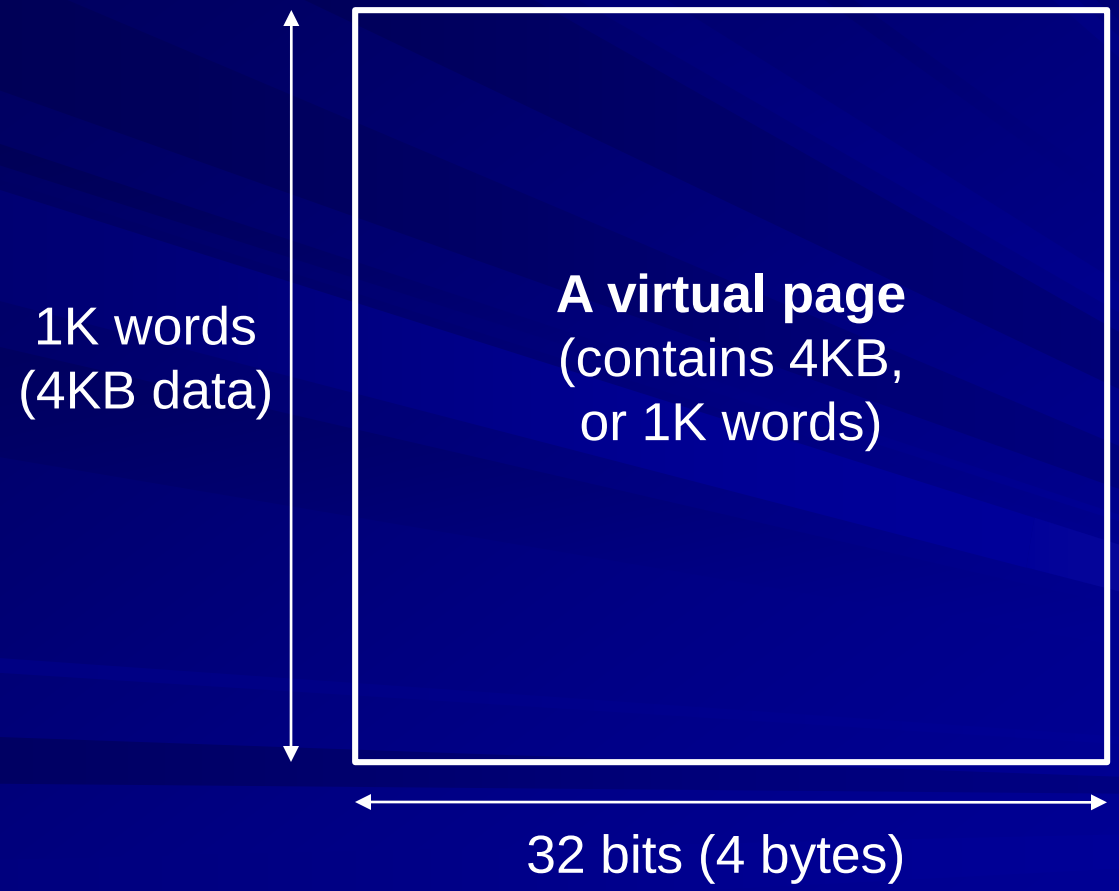
- 32-bit address (byte addressing)
- 4 GB virtual main memory (disk space)
  - Page size = 4 KB
  - Number of virtual pages = $4 \times 2^{30} / (4 \times 2^{10}) = 1M$
  - Bits for virtual page number = $\log_2(1M) = 20$
- 128 MB physical main memory
  - Page size 4 KB
  - Number of physical pages = $128 \times 2^{20} / (4 \times 2^{10}) = 32K$
  - Bits for physical page number = $\log_2(32K) = 15$
  - Page table contains 1M records specifying where each virtual page is located.

Page Table

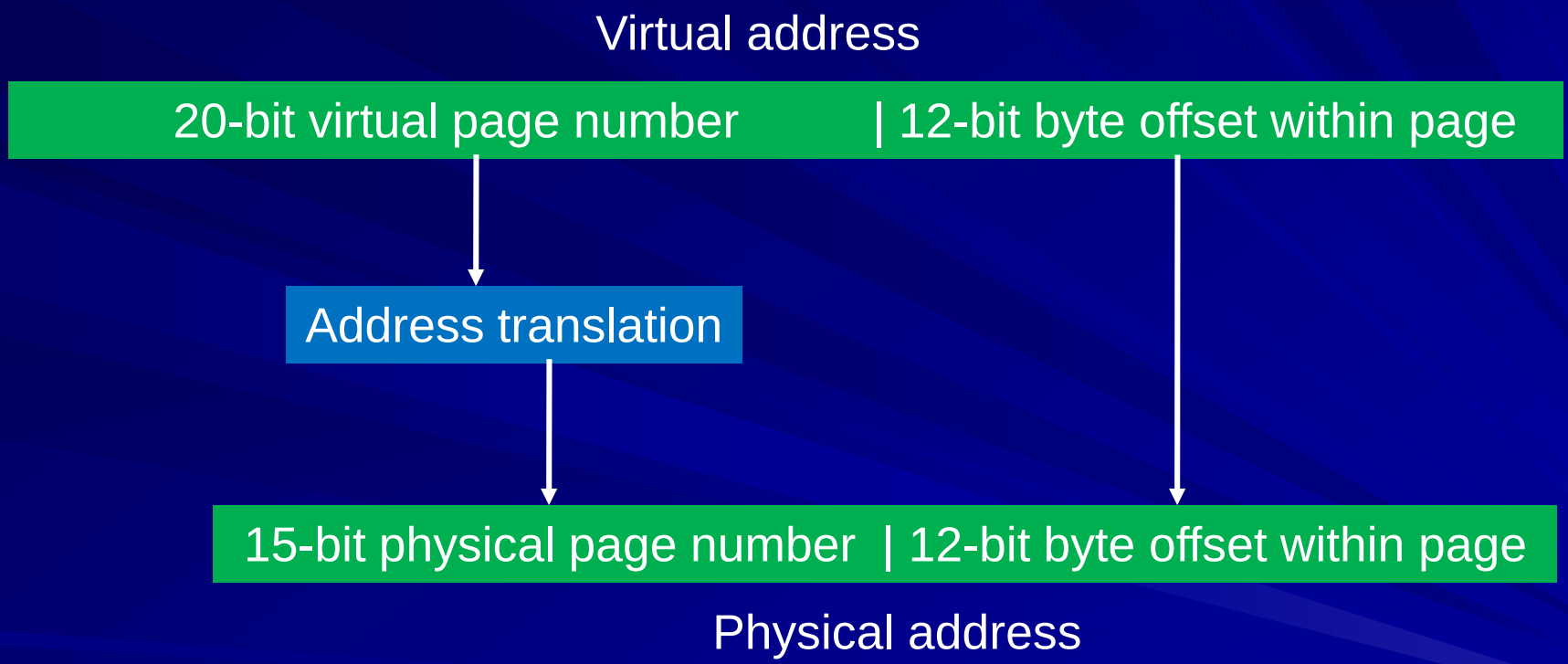

Page table register
Address of Page table in main memory
Page locations
1M virtual page numbers
0
1
2
3
.
.
K
.
.
.
-
2
-
1
3
-
-
-
0
-
-
Valid bit
Other flags, e.g., dirty bit, LRU ref. bit
Virtual main memory (pages on disk)
Page 0
Page 1
Page 2
Page 3
Physical main memory (pages)

32-bit Virtual Address (4 KB Page)

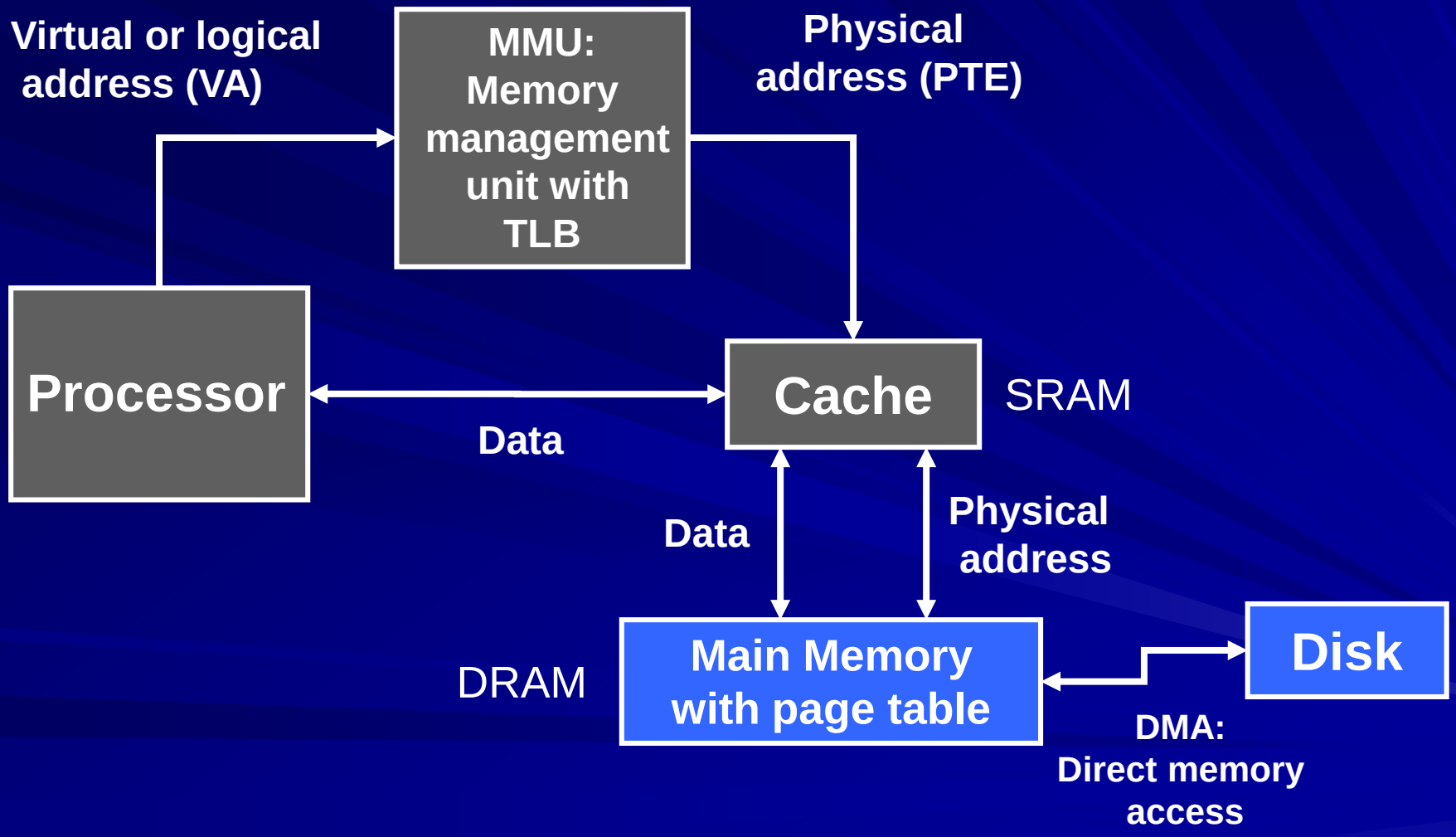
20-bit virtual page number |10-b word number within page|2-b byte offset



Virtual to Physical Address Translation



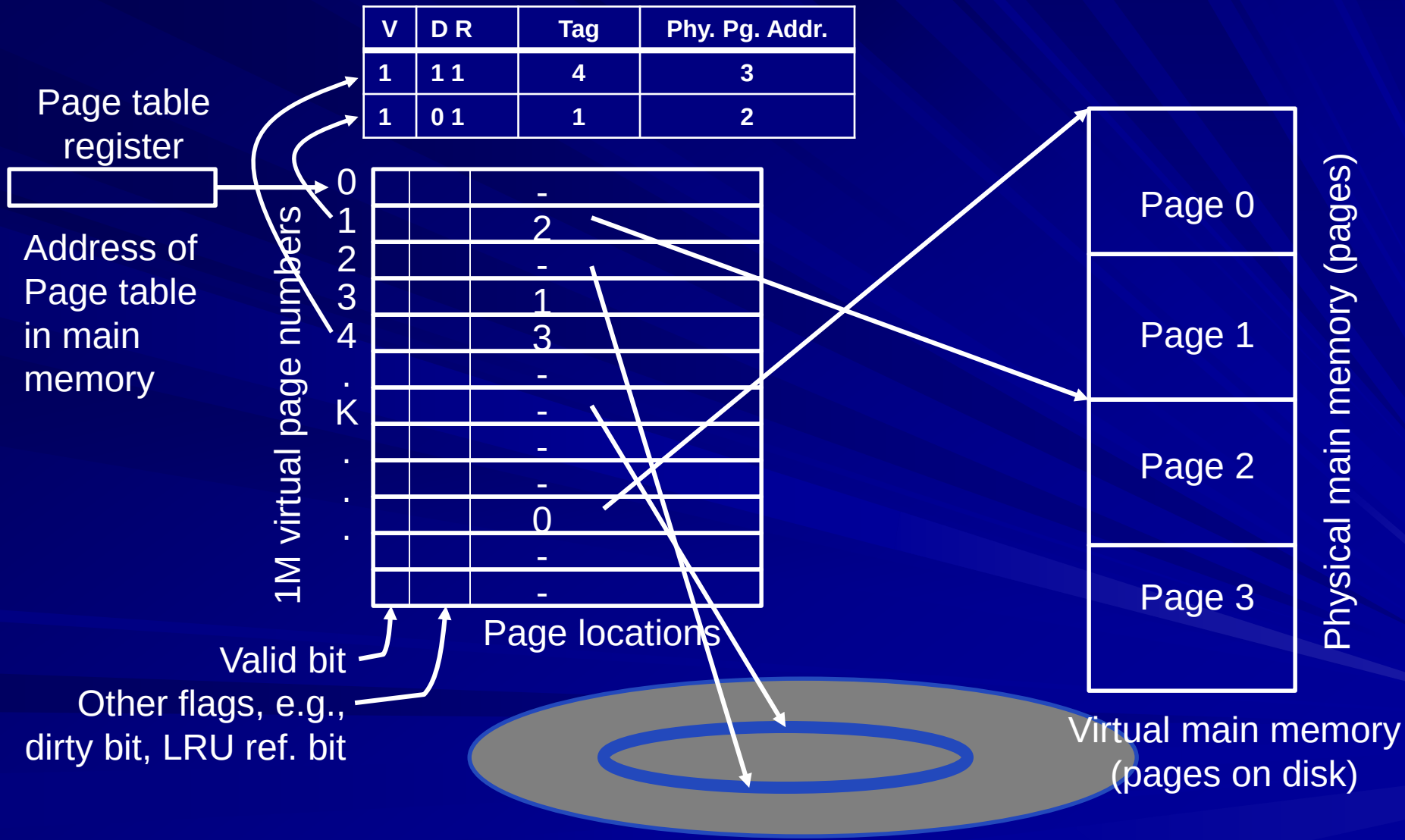
Virtual Memory System



TLB: Translation-Lookaside Buffer

- A processor request requires two accesses to main memory:
  - Access page table to get physical address
  - Access physical address
- TLB acts as a cache of page table
  - Holds recent virtual to physical page translations
  - Eliminates one main memory access if requested virtual page address is found in TLB (hit)

TLB Organization



TLB Data

- V: Valid bit
- D: Dirty bit
- R: Reference bit (LRU)
- Tag: Index in page table
- Physical page address

Typical TLB Characteristics

- TLB size: 16 – 512 entries
- Block size: 1 – 2 page table entries of 4 – 8 bytes each
- Hit time: 0.5 – 1 clock cycle
- Miss penalty: 10 – 100 clock cycles
- Miss rate: 0.01% – 1%

Up Next:

System Level Performance