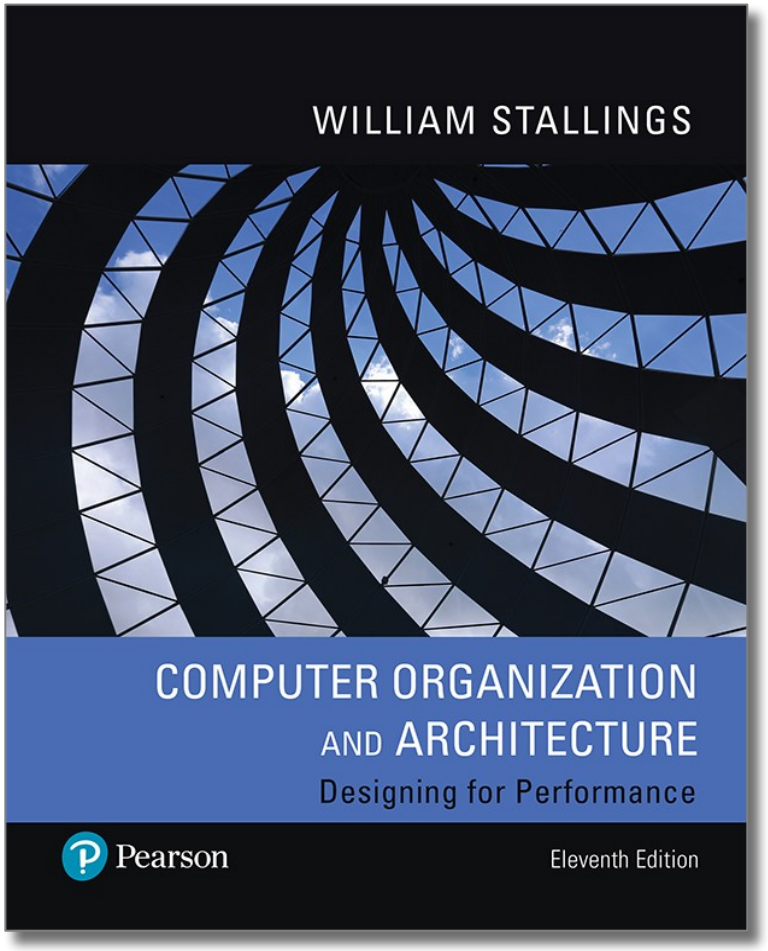
Computer Organization and Architecture

Designing for Performance

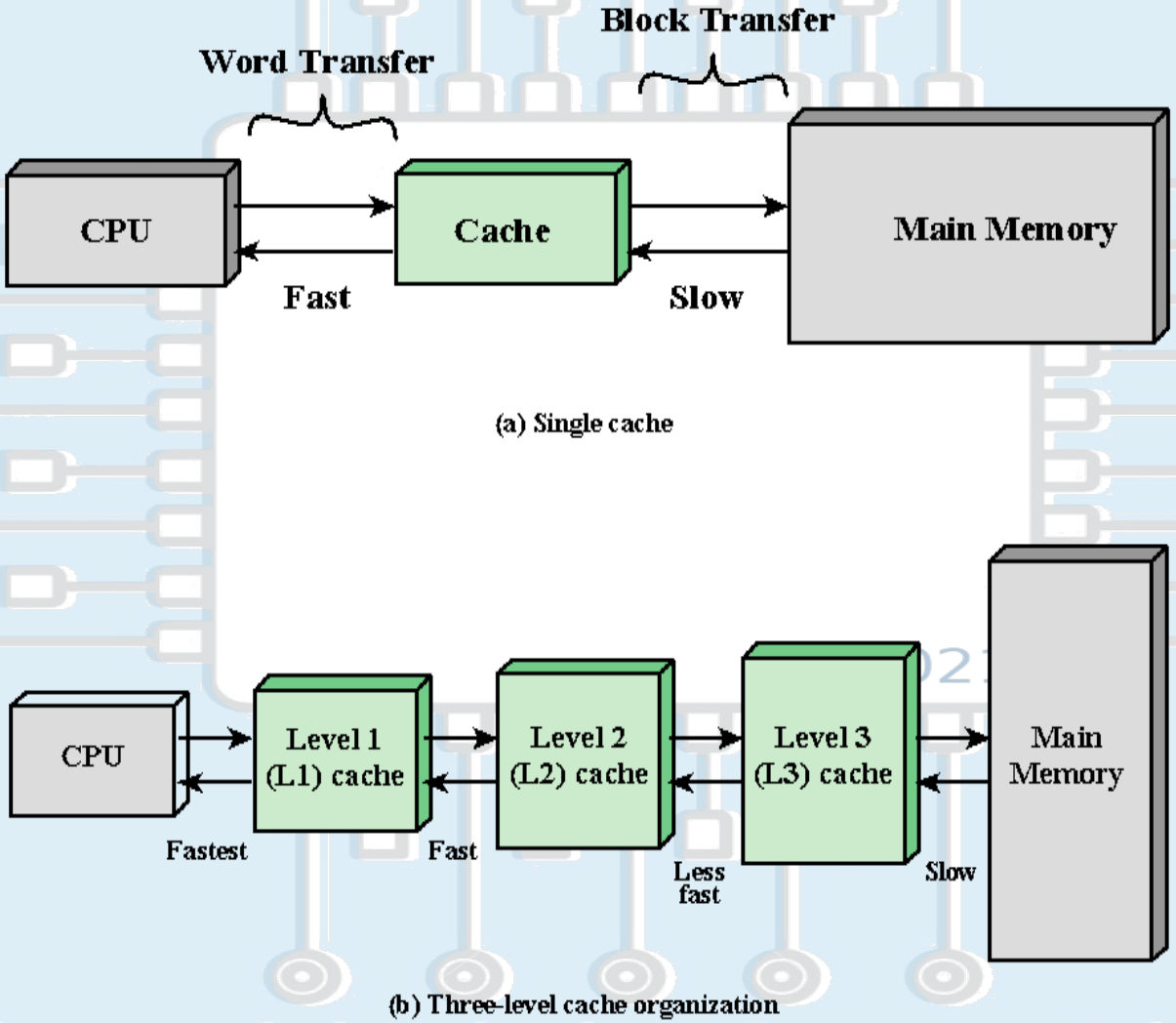
11th Edition



Chapter 5

Cache Memory

Figure 5.1
Cache and Main Memory



Cache Memory Principles

- Block
  - The minimum unit of transfer between cache and main memory
- Frame
  - To distinguish between the data transferred and the chunk of physical memory, the term frame, or block frame, is sometimes used with reference to caches
- Line
  - A portion of cache memory capable of holding one block, so-called because it is usually drawn as a horizontal object
- Tag
  - A portion of a cache line that is used for addressing purposes
- Line size
  - The number of data bytes, or block size, contained in a line

Figure 5.2
Cache/Main Memory Structure

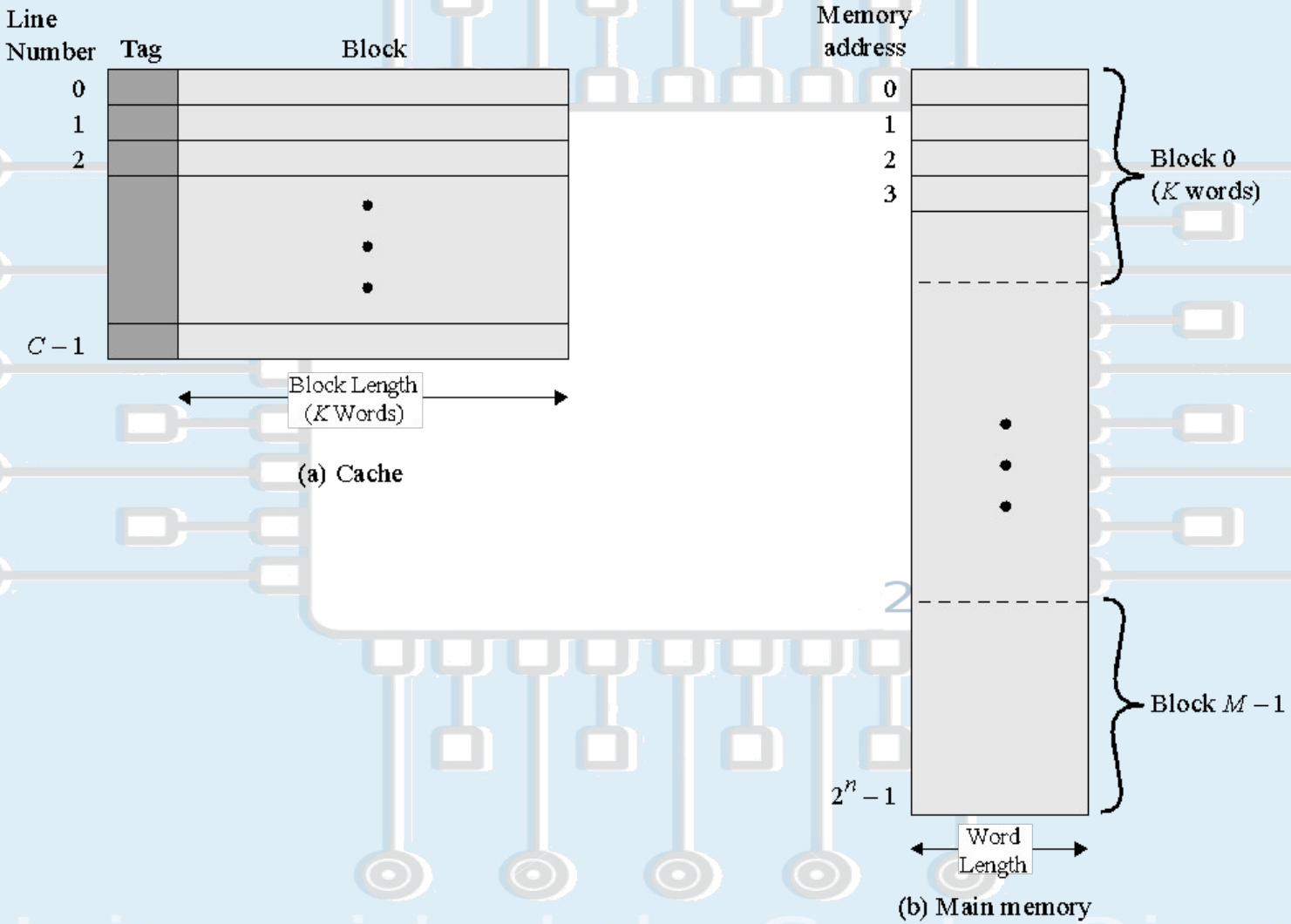


Figure 5.3
Cache Read Operation

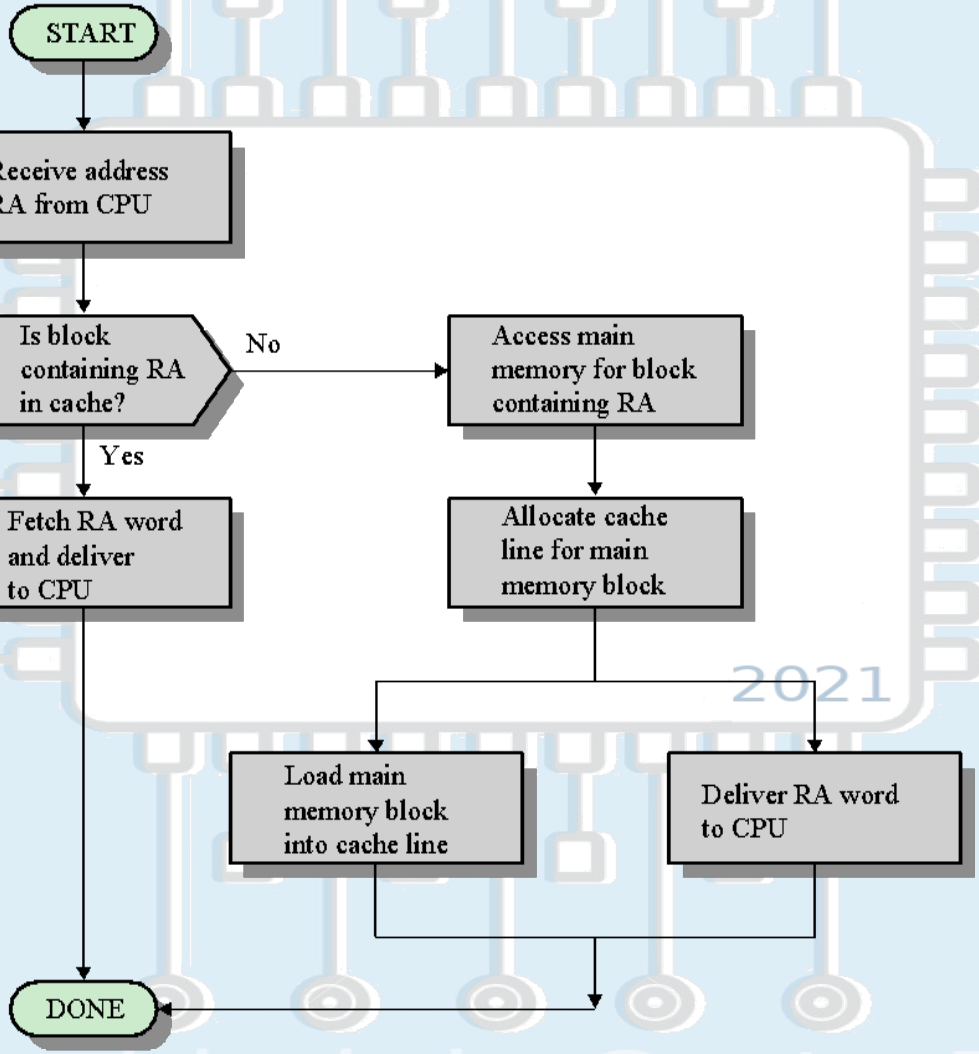


Figure 5.4
Typical Cache Organization

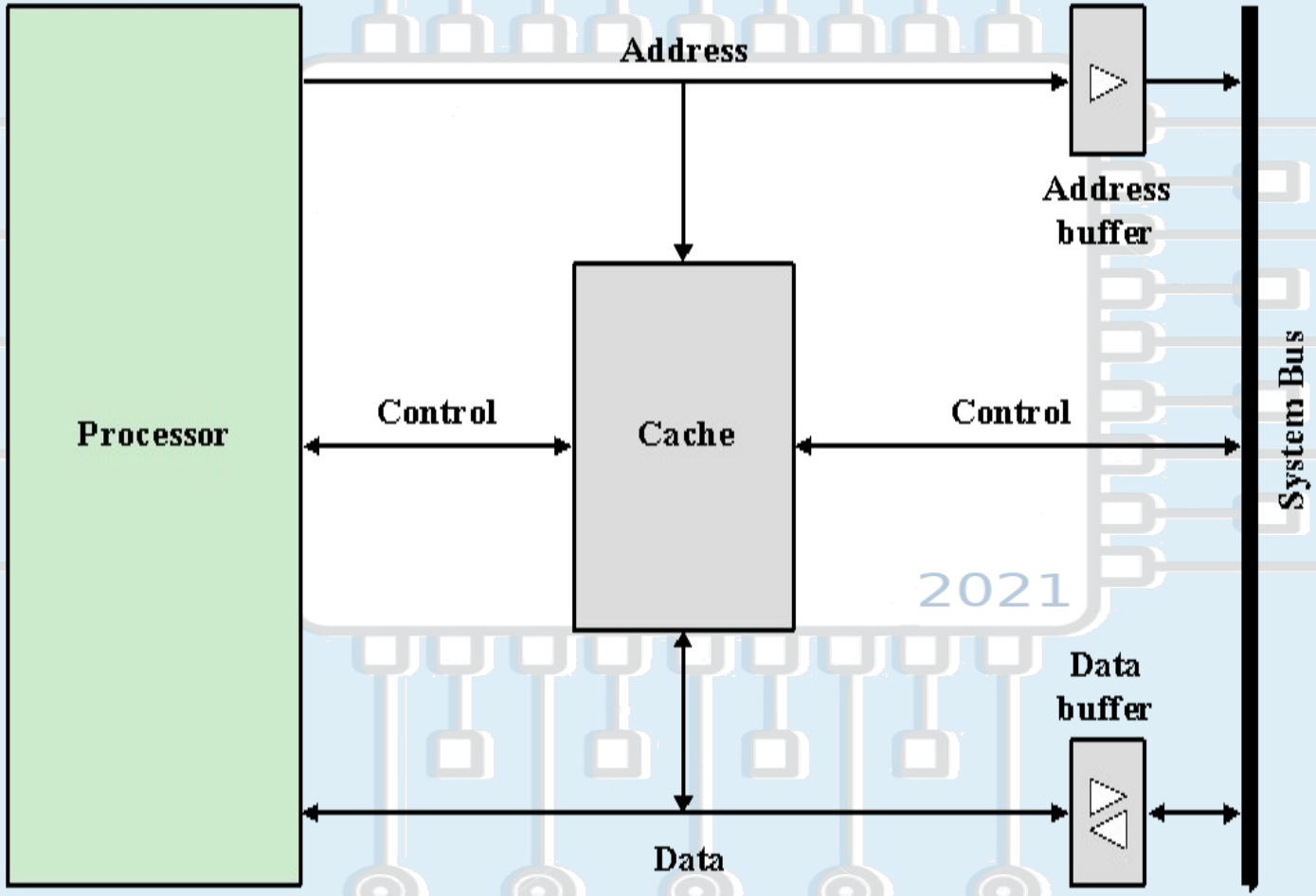


Table 5.1
Elements of Cache Design

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

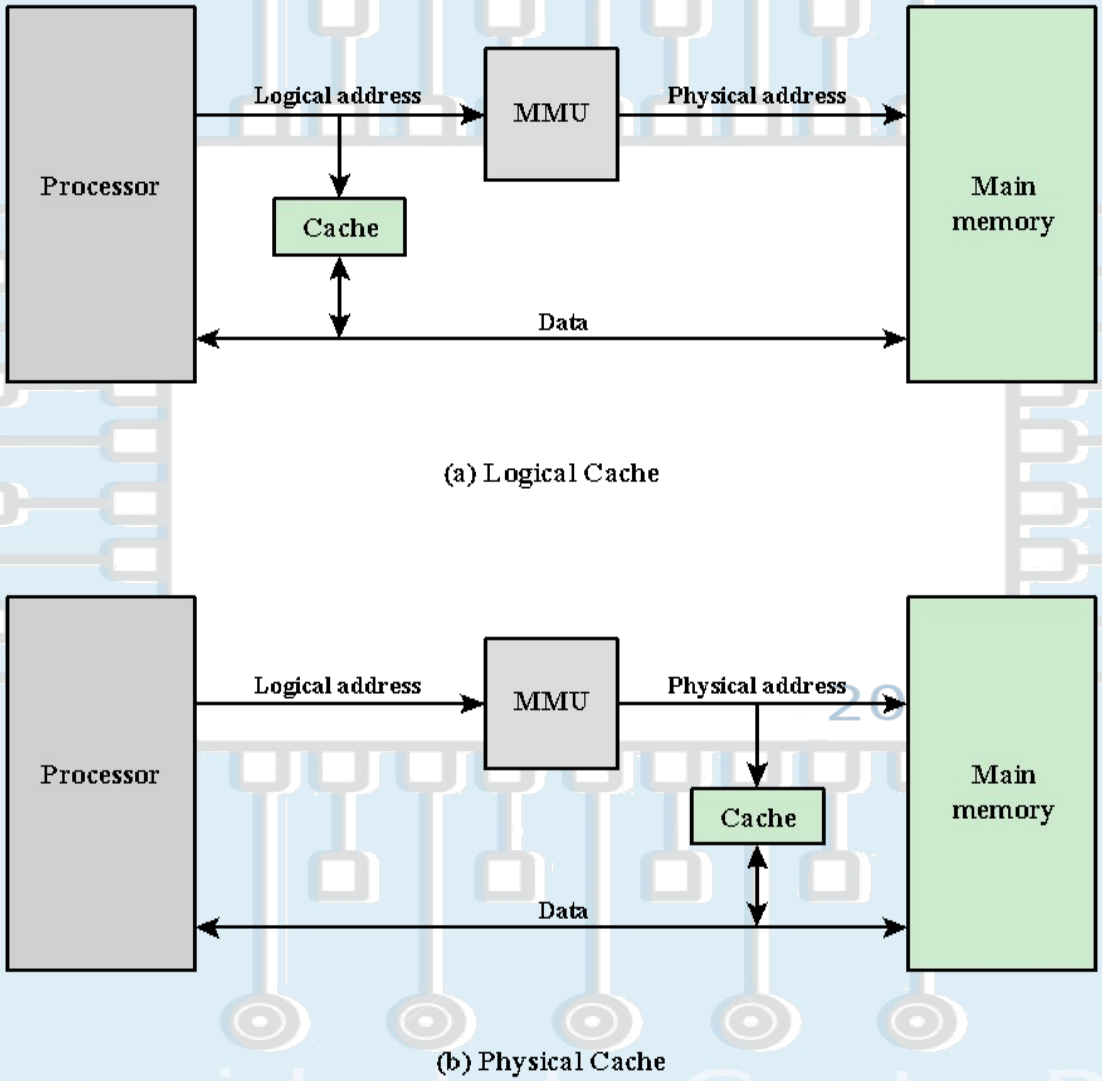
Number of Caches
- Single or two level
- Unified or split

Cache Addresses

•Virtual Memory

- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory

Figure 5.5
Logical and Physical Caches



Cache Size

- Preferable for the size of the cache to be:
  - Small enough so that the overall average cost per bit is close to that of main memory alone
  - Large enough so that the overall average access time is close to that of the cache alone
- Motivations for minimizing cache size:
  - The larger the cache, the larger the number of gates involved in addressing the cache resulting in large caches being slightly slower than small ones
  - The available chip and board area also limits cache size
- Because the performance of the cache is very sensitive to the nature of the workload, it is impossible to arrive at a single "optimum" cache size

Table 5.2
Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 Cache[a]	L2 cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	–	–
PDP-11/70	Minicomputer	1968	1 kB	–	–
IBM 3033	Mainframe	1968	64 kB	–	–
IBM 3090	Mainframe	1968	128 to 256 kB	–	–
Intel 80486	PC	1968	8 kB	–	–
Pentium	PC	1968	8 kB/8 kB	256 to 512 kB	–
PowerPC 620	PC	1968	32 kB/32 kB	–	–
IBM S/390 G6	Mainframe	1968	256 kB	8 MB	–
Pentium 4	PC/server	1968	8 kB/8 kB	256 kB	–
Itanium	PC/server	1968	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	1968	32 kB	256 kB	6 MB
IBM POWER5	High-end server	1968	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	1968	64 kB/64 kB	1 MB	–
IBM POWER6	PC/server	1968	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	1968	64 kB/128 kB	3 MB	24-48 MB
Intel Core i7 EE 990	Workstaton/ Server	1968	6 × 32 kB/32 kB	6 × 1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ Server	1968	24 × 64 kB/128 kB	24 × 1.5 MB	24 MB L3 192 MB L4
IBM z13	Mainframe/ server	1968	24 × 96 kB/128 kB	24 × 2 MB/2 MB	64 MB L3 480 MB L4
Intel Core i0-7900X	Workstation/ server	1968	8 × 32 kB/32 kB	8 × 1 MB	14 MB

[a] Two values separated by a slash refer to instruction and data caches.

(Table can be found on page 145 in the textbook.)

Table 5.3
Cache Access Methods

Method	Organization	Mapping of Main Memory Blocks to Cache	Access using Main Memory Address
Direct Mapped	Sequence of *m* lines	Each block of main memory maps to one unique line of cache.	*Line* portion of address used to access cache line; *Tag* portion used to check for hit on that line.
Fully Associative	Sequence of *m* lines	Each block of main memory can map to any line of cache.	*Tag* portion of address used to check every line for hit on that line.
Set Associative	Sequence of *m* lines organized as *v* sets of *k* lines each ($m = v \times k$)	Each block of main memory maps to one unique cache set.	*Line* portion of address used to access cache set; *Tag* portion used to check every line in that set for hit on that line.

Figure 5.6
Mapping from Main Memory to Cache: Direct and Associative

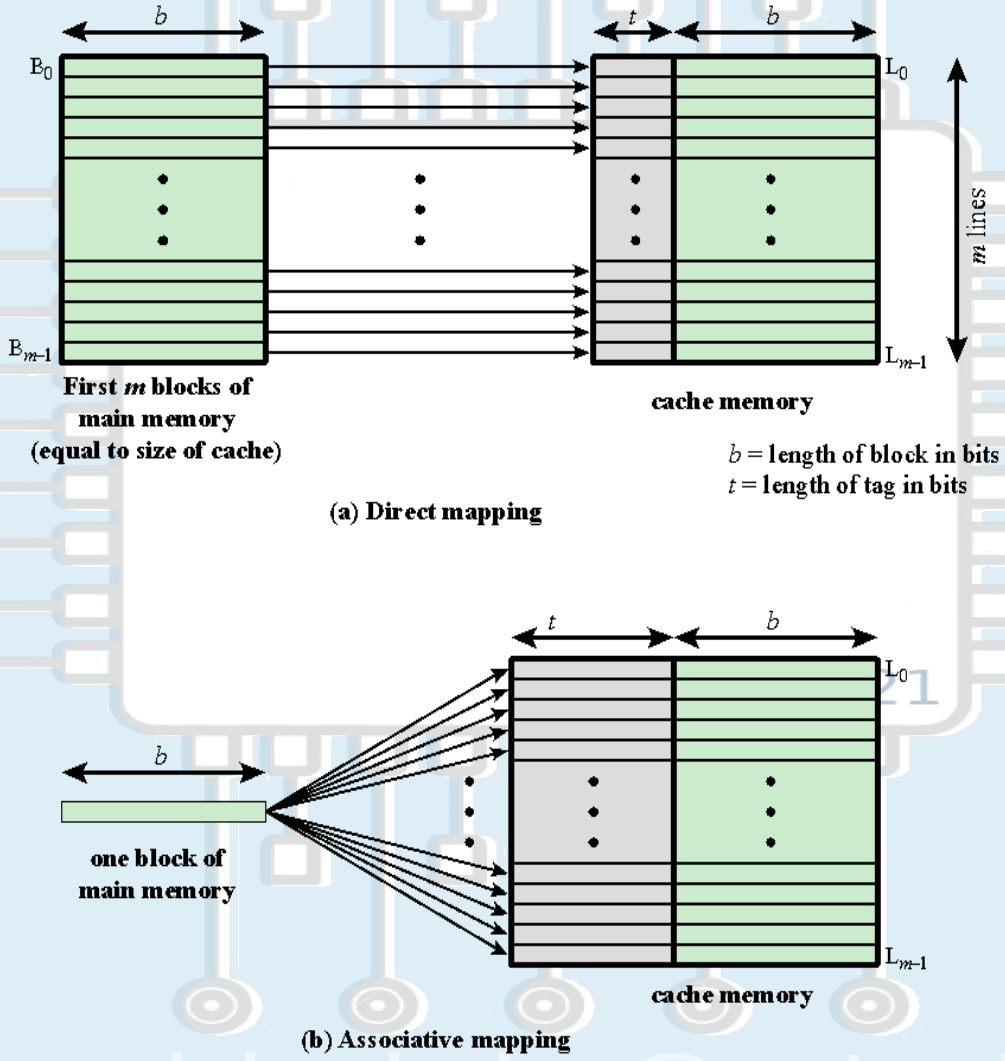


Figure 5.7
Direct-Mapping Cache Organization

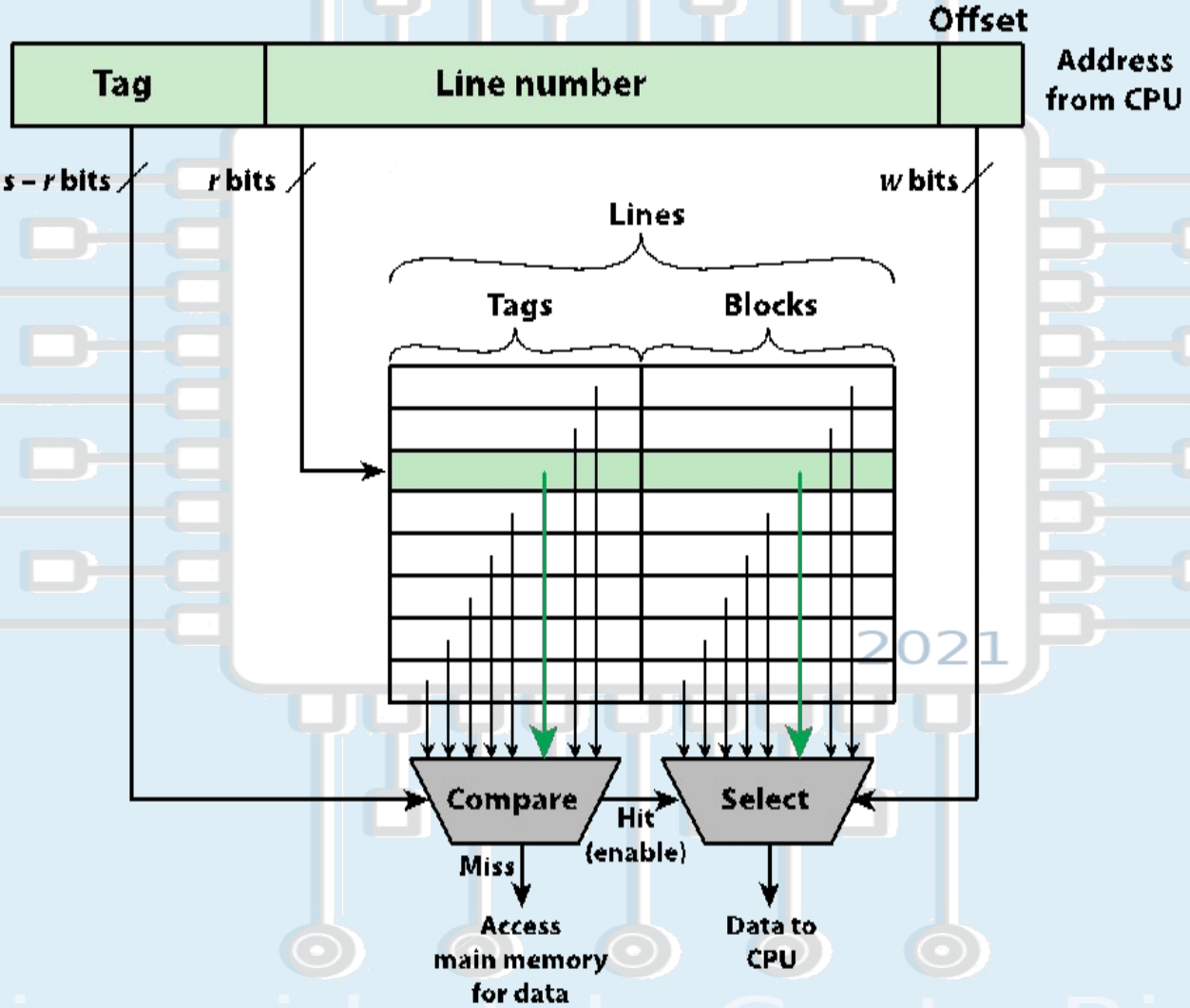
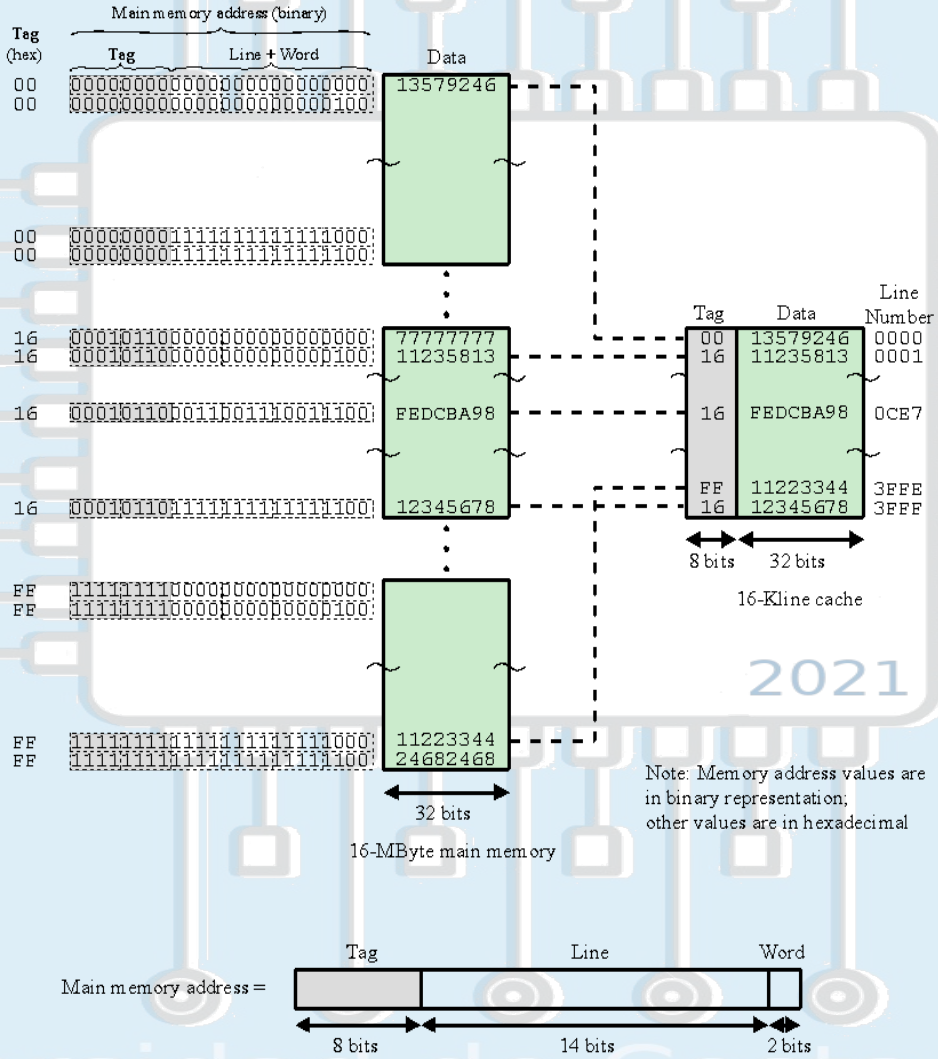


Figure 5.8
Direct Mapping Example



Content-Addressable Memory (CAM)

- Also known as associative storage
- Content-addressable memory is constructed of static RAM (SRAM) cells but is considerably more expensive and holds much less data than regular SRAM chips
- A CAM with the same data capacity as a regular SRAM is about 60% larger
- A CAM is designed such that when a bit string is supplied, the CAM searches its entire memory in parallel for a match
  - If the content is found, the CAM returns the address where the match is found and, in some architectures, also returns the associated data word
  - This process takes only one clock cycle

Figure 5.9
Content-Addressable Memory

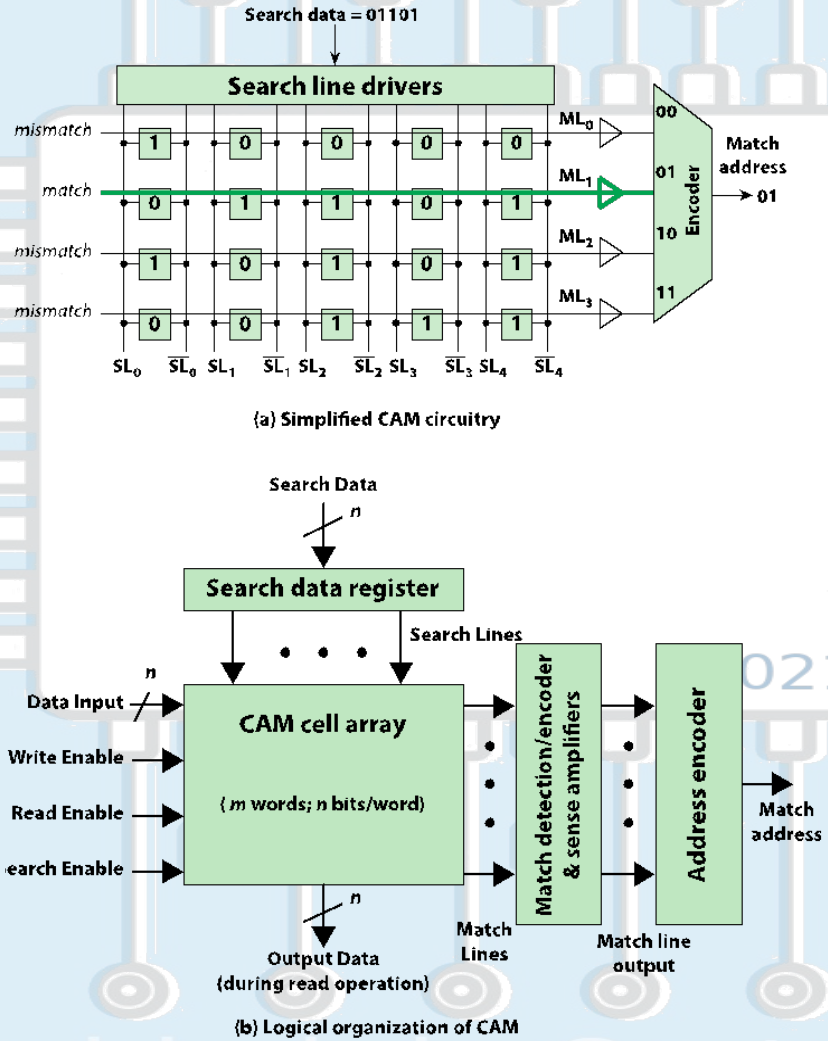


Figure 5.10
Fully Associative Cache Organization

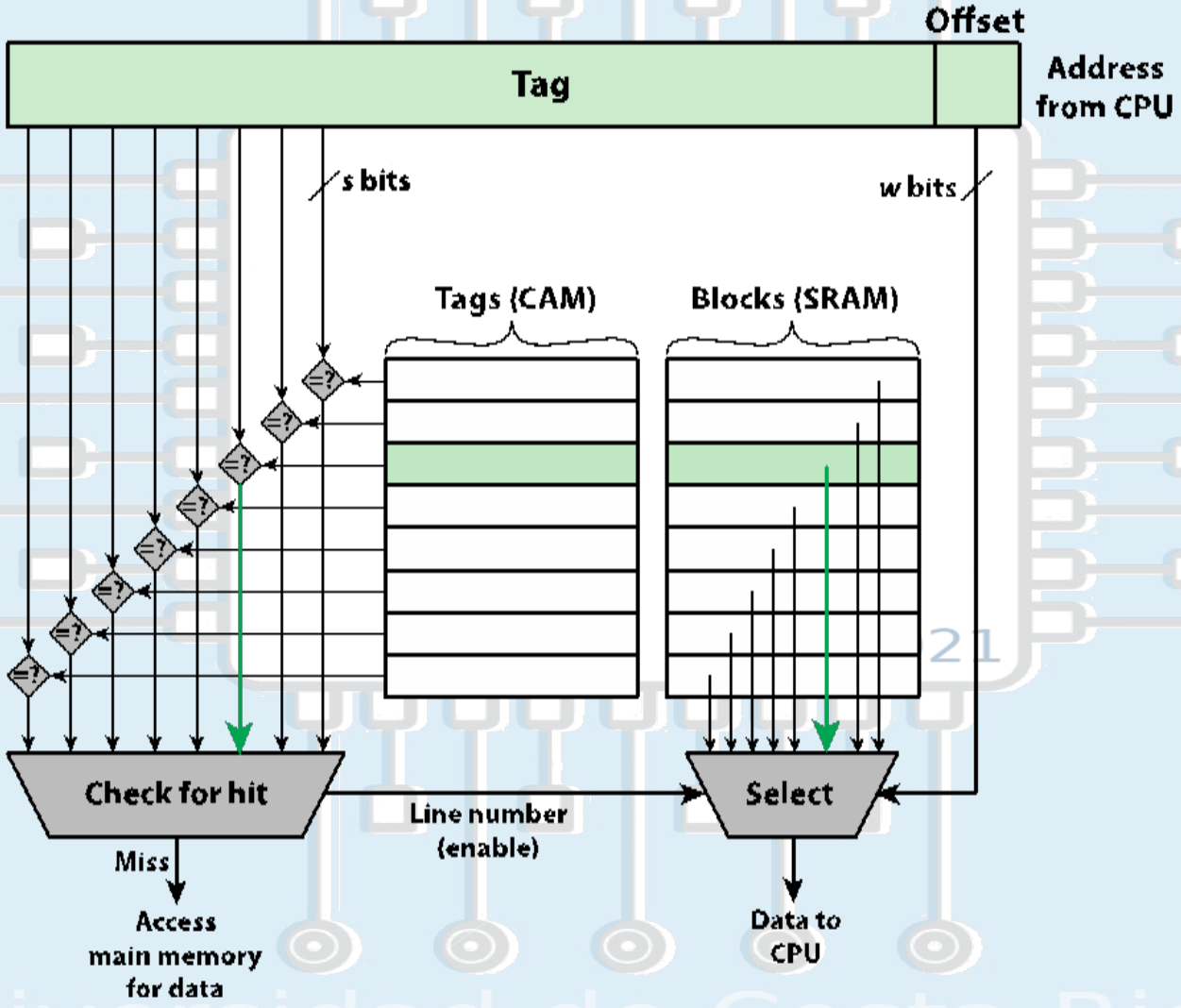
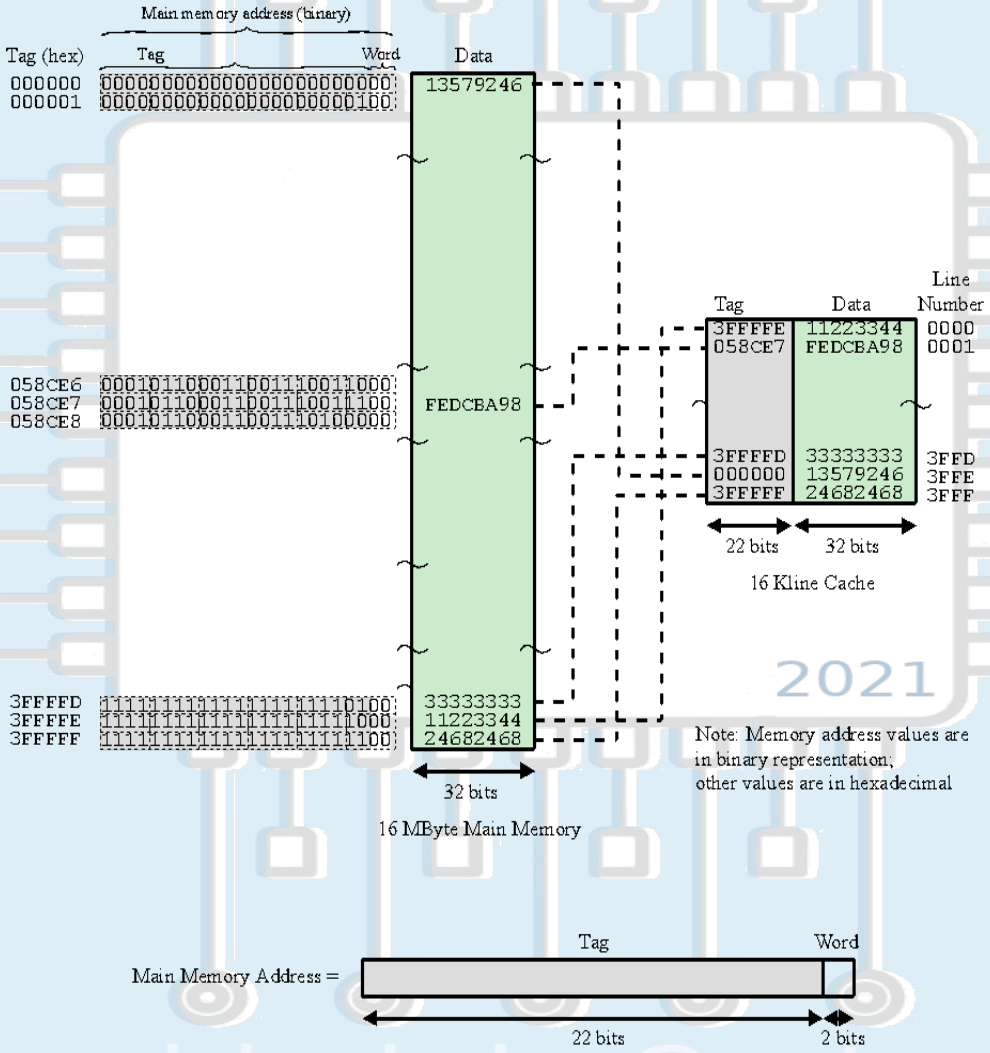


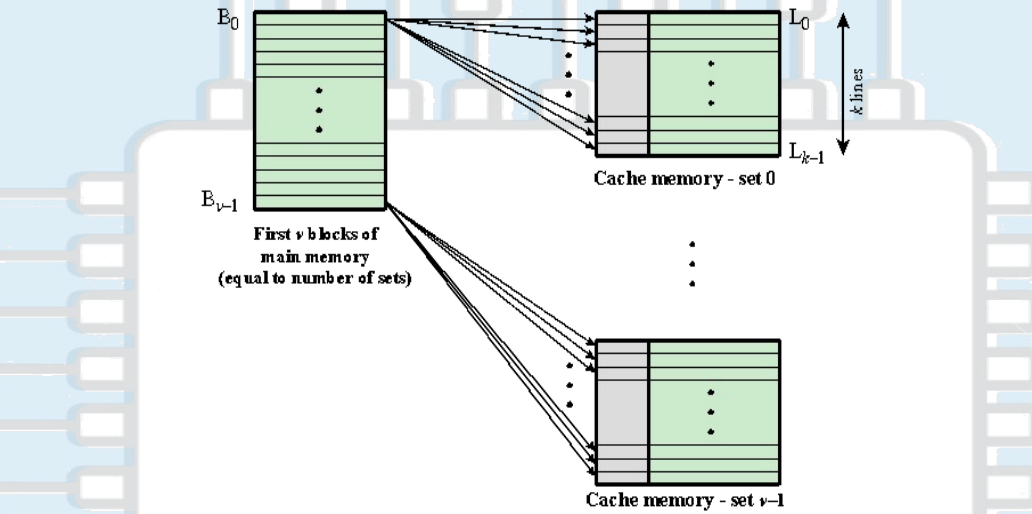
Figure 5.11
Associative Mapping Example



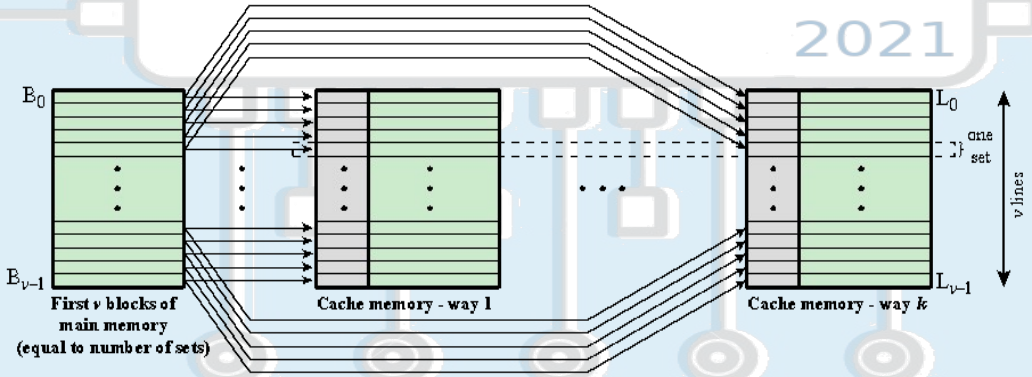
Set Associative Mapping

- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Figure 5.12
Mapping from Main Memory to Cache: k-Way Set Associative



(a) v associative-mapped caches



(b) k direct-mapped caches

Figure 5.13 *k*-Way Set Associative Cache Organization

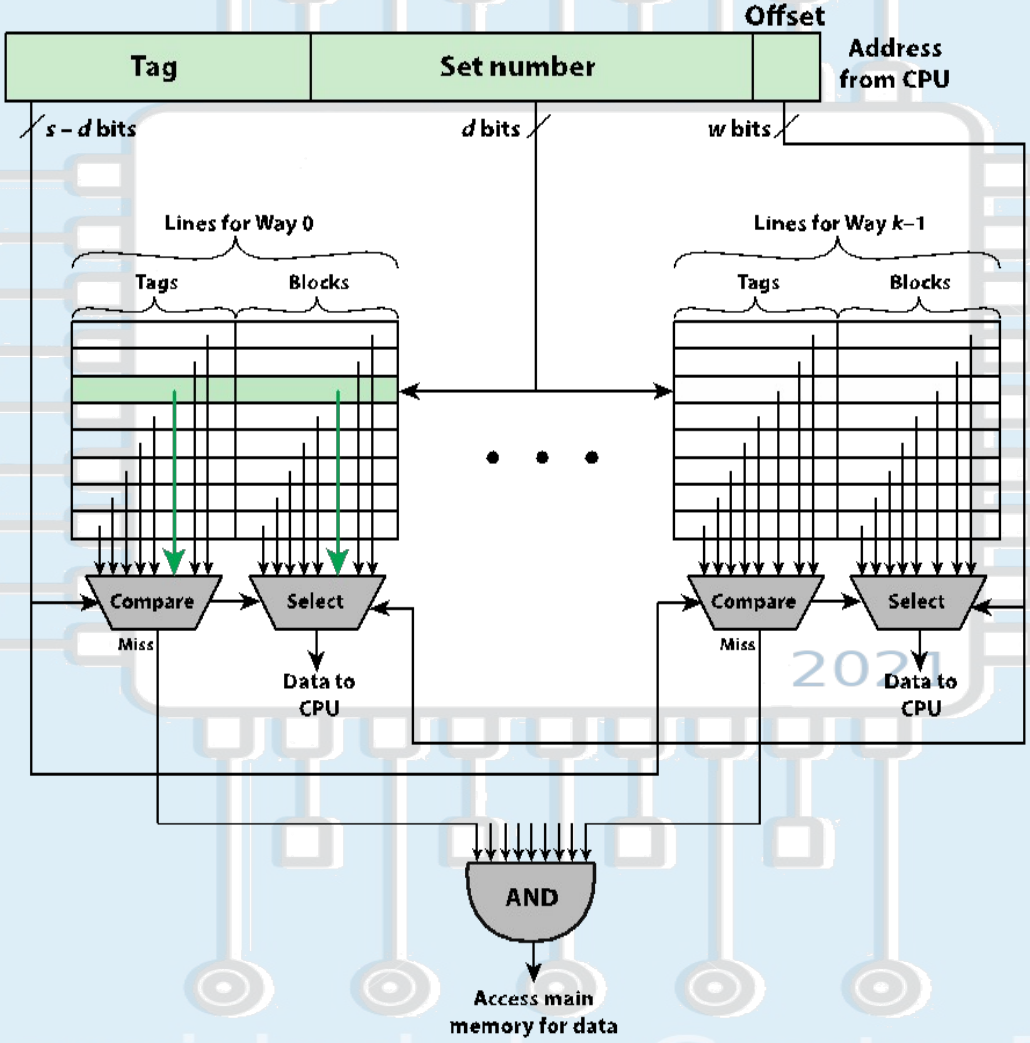


Figure 5.14
Two-Way Set-Associative Mapping Example

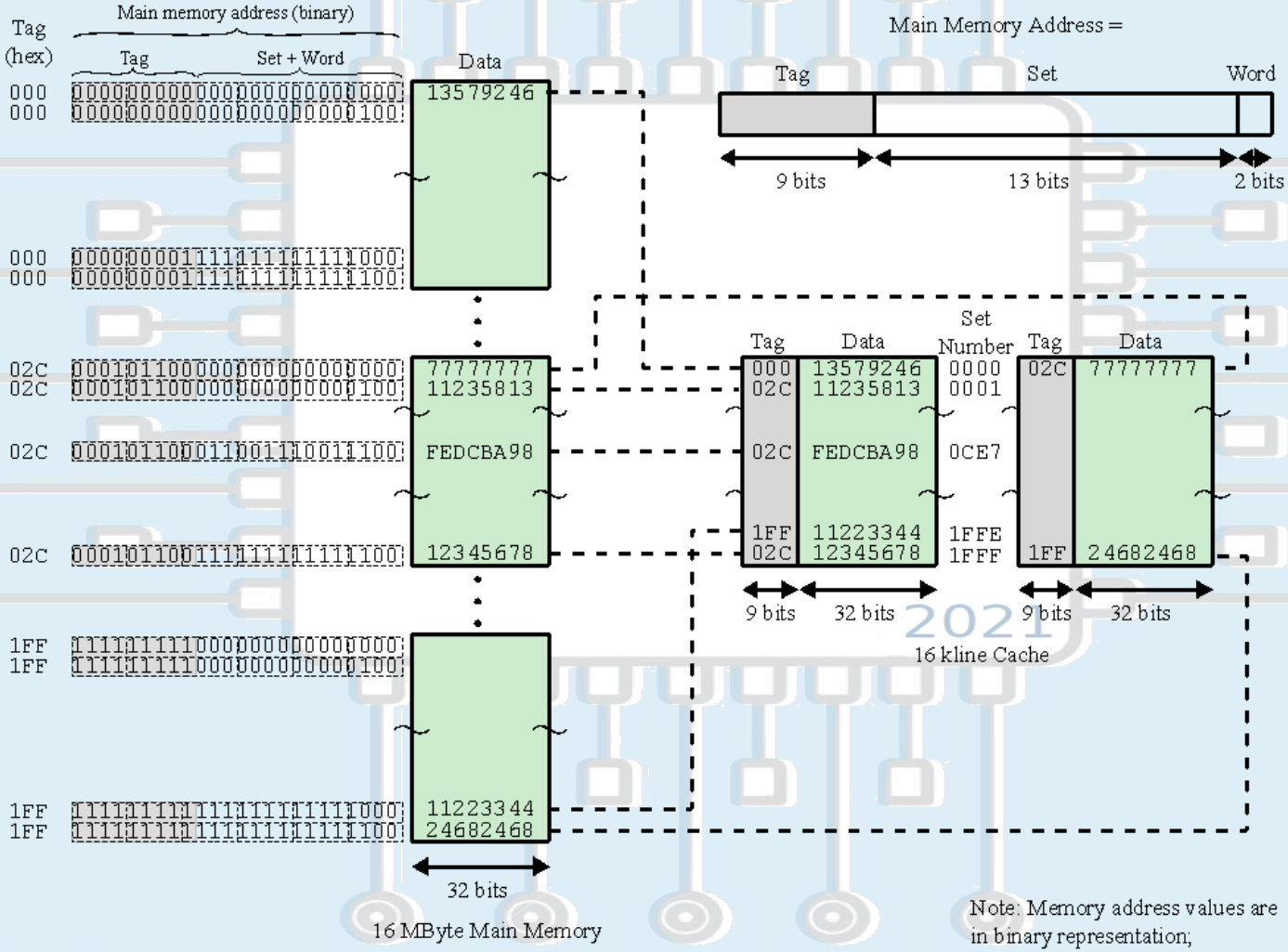
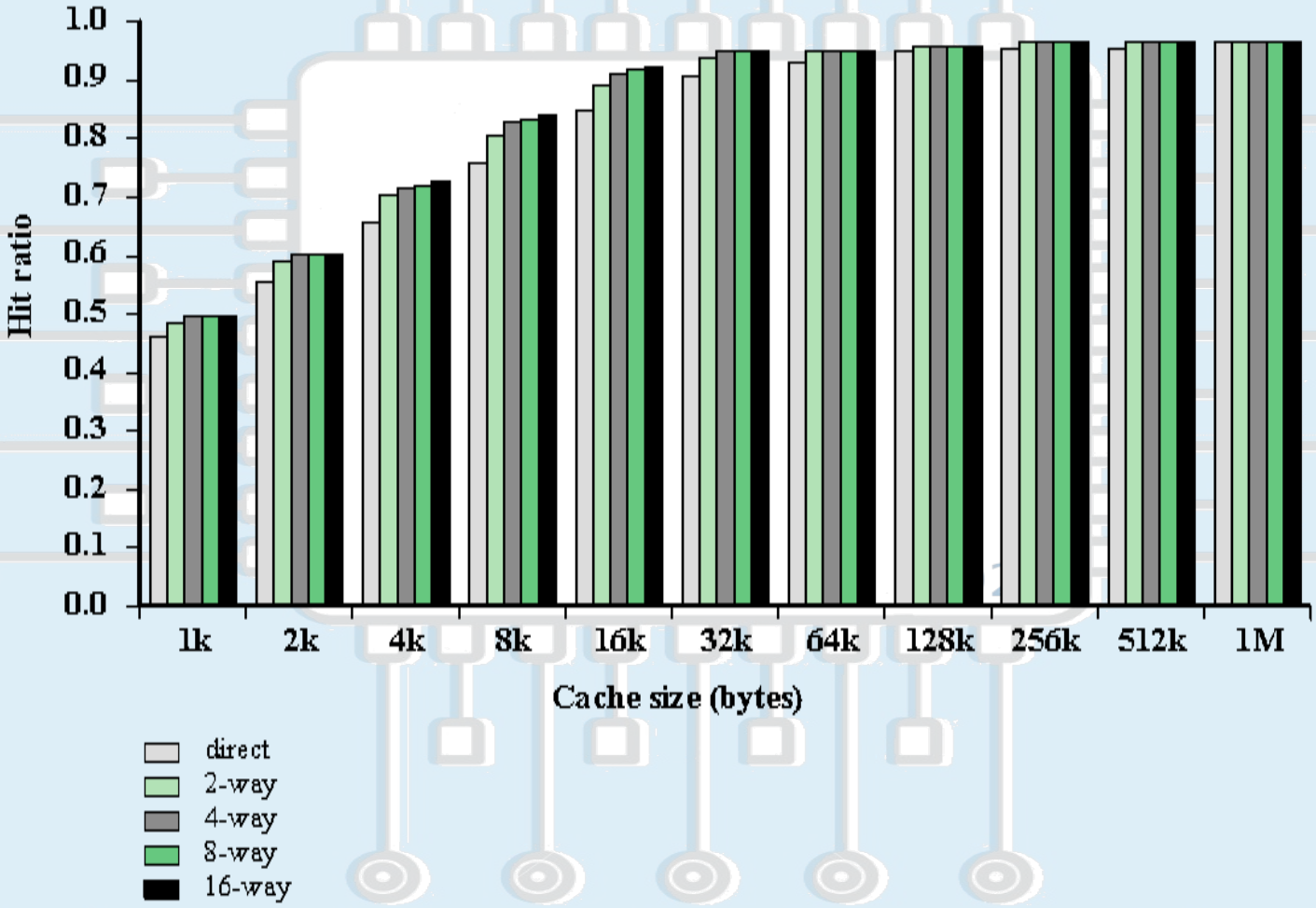


Figure 5.15
Varying Associativity over Cache Size



Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping there is only one possible line for any particular block and no choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

The most common replacement algorithms are:

- Least recently used (LRU)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- First-in-first-out (FIFO)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Least frequently used (LFU)
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend with:

More than one device may have access to main memory

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Write Through and Write Back

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

Write Miss Alternatives

- There are two alternatives in the event of a write miss at a cache level:
  - Write allocate
    - The block containing the word to be written is fetched from main memory (or next level cache) into the cache and the processor proceeds with the write cycle
  - No write allocate
    - The block containing the word to be written is modified in the main memory and not loaded into the cache
- Either of these policies can be used with either write through or write back
- No write allocate is most commonly used with write through
- Write allocate is most commonly used with write back

Cache Coherency

- A new problem is introduced in a bus organization in which more than one device has a cache and main memory is shared
- If data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches
- Even if a write-through policy is used, the other caches may contain invalid data
- Possible approaches to cache coherency include:
  - **Bus watching with write through**
    - Each cache controller monitors the address lines to detect write operations to memory by other bus masters
    - If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry
    - This strategy depends on the use of a write-through policy by all cache controllers
  - **Hardware transparency**
    - Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches
    - If one processor modifies a word in its cache, this update is written to main memory
  - **Noncacheable memory**
    - Only a portion of main memory is shared by more than one processor, and this is designated as noncacheable
    - All accesses to shared memory are cache misses, because the shared memory is never copied into the cache
    - The noncacheable memory can be identified using chip-select logic or high-address bits

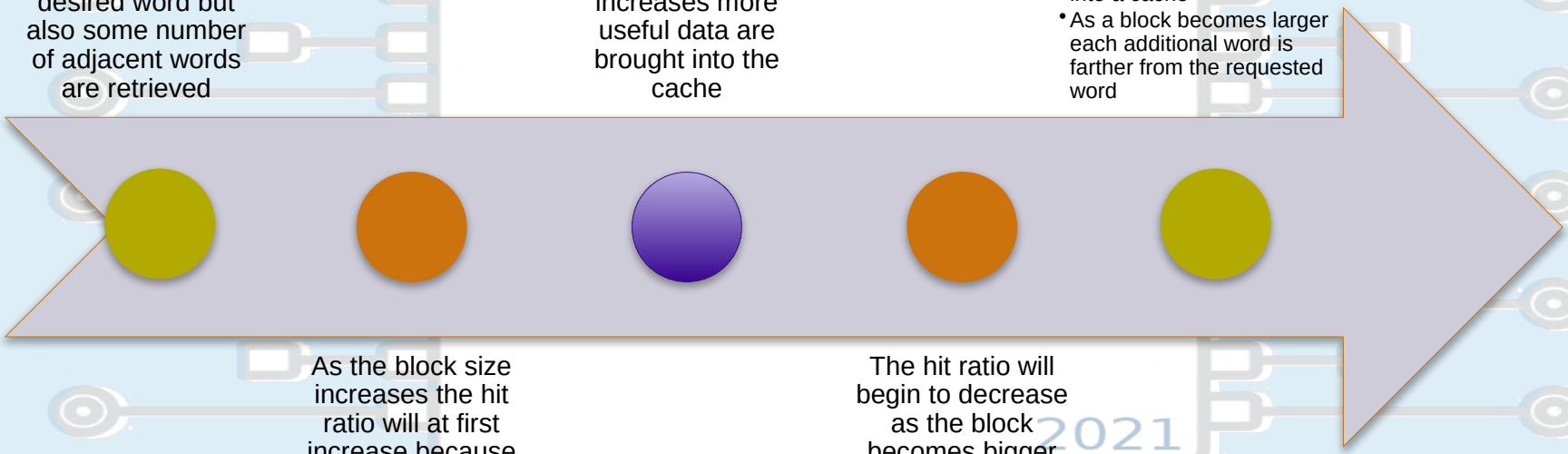
Line Size

When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved

As the block size increases more useful data are brought into the cache

Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache
- As a block becomes larger each additional word is farther from the requested word



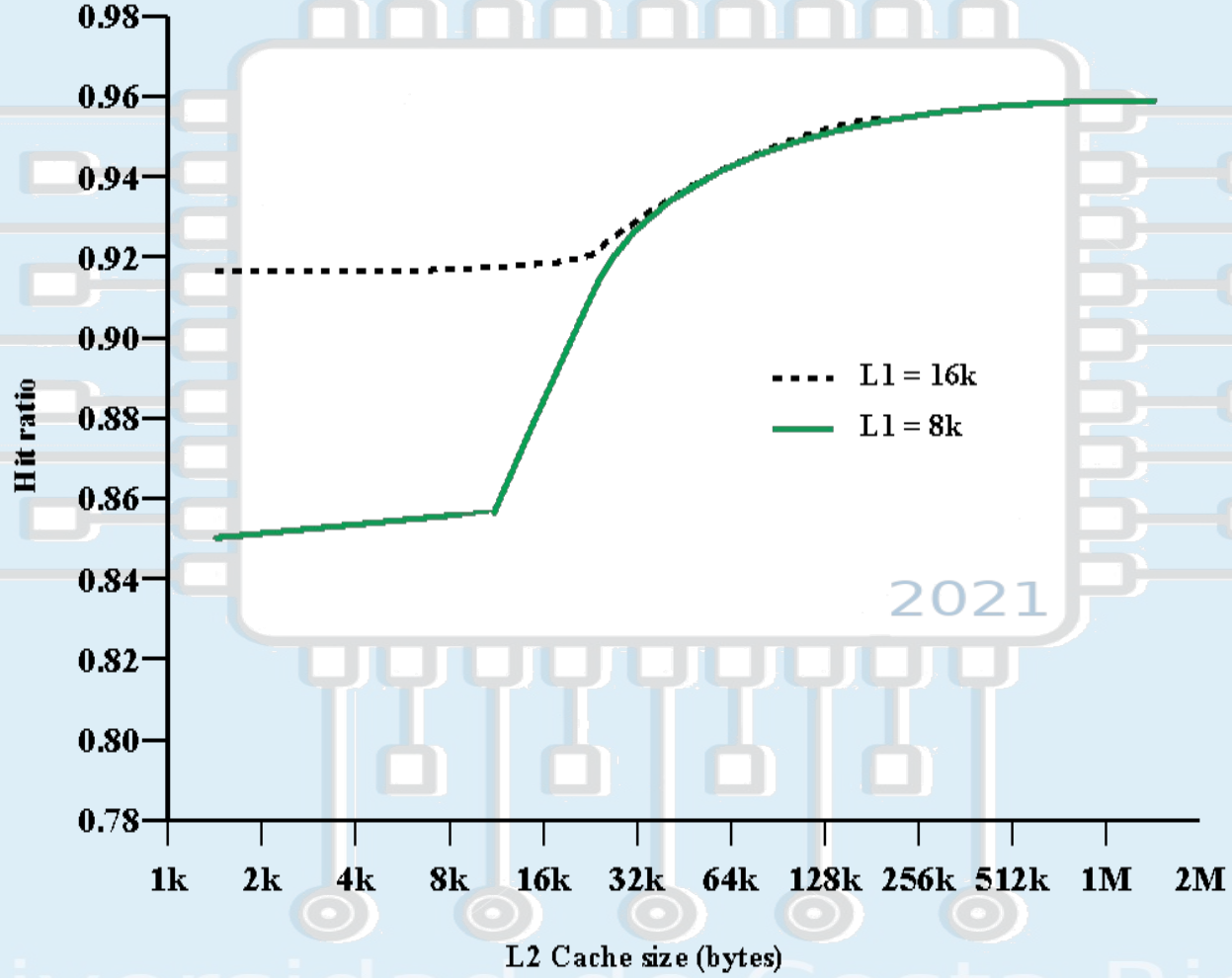
As the block size increases the hit ratio will at first increase because of the principle of locality

The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced

Multilevel Caches

- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
  - During this period the bus is free to support other transfers
- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches
- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

Figure 5.16
Total Hit Ratio (L1 and L2) for 8-kB and 16-kB L1



Unified Versus Split Caches

- Has become common to split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load of instruction and data fetches automatically
    - Only one cache needs to be designed and implemented
- Trend is toward split caches at the L1 and unified caches for higher levels
- Advantages of split cache:
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

Inclusion Policy

- **Inclusive policy**
  - Dictates that a piece of data in one cache is guaranteed to be also found in all lower levels of caches
  - Advantage is that it simplifies searching for data when there are multiple processors in the computing system
  - This property is useful in enforcing cache coherence
- **Exclusive policy**
  - Dictates that a piece of data in one cache is guaranteed not to be found in all lower levels of caches
  - The advantage is that it does not waste cache capacity since it does not store multiple copies of the same data in all of the caches
  - The disadvantage is the need to search multiple cache levels when invalidating or updating a block
  - To minimize the search time, the highest-level tag sets are typically duplicated at the lowest cache level to centralize searching
- **Noninclusive policy**
  - With the noninclusive policy a piece of data in one cache may or may not be found in lower levels of caches
  - As with the exclusive policy, this policy will generally maintain all higher-level cache sets at the lowest cache level

Table 5.4
Intel Cache Evolution

Problem	Solution	Processor on Which Feature First Appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip.	Add external L2 cache using faster technology than main memory.	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-Chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Figure 5.17
Pentium 4 Block Diagram

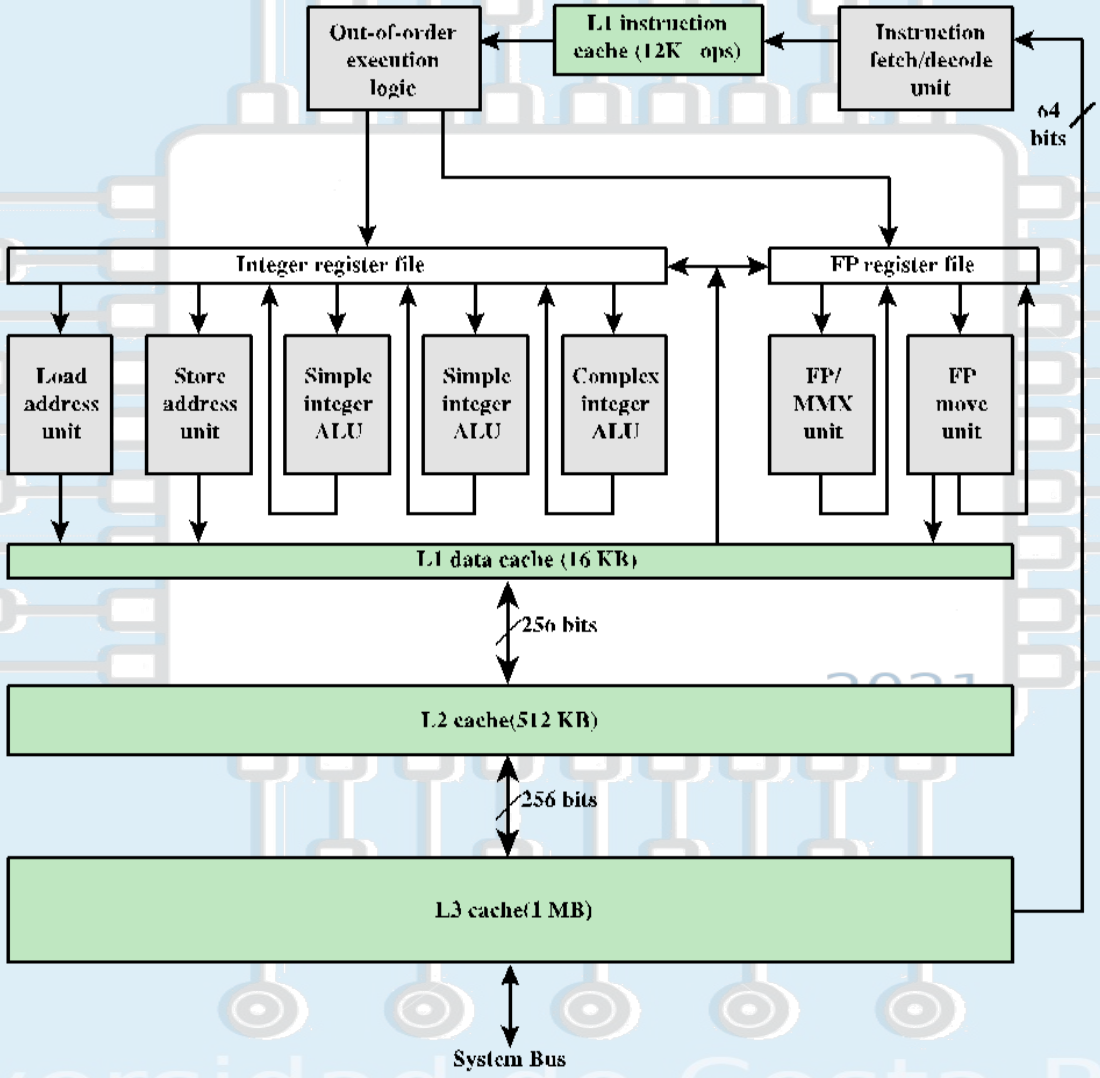
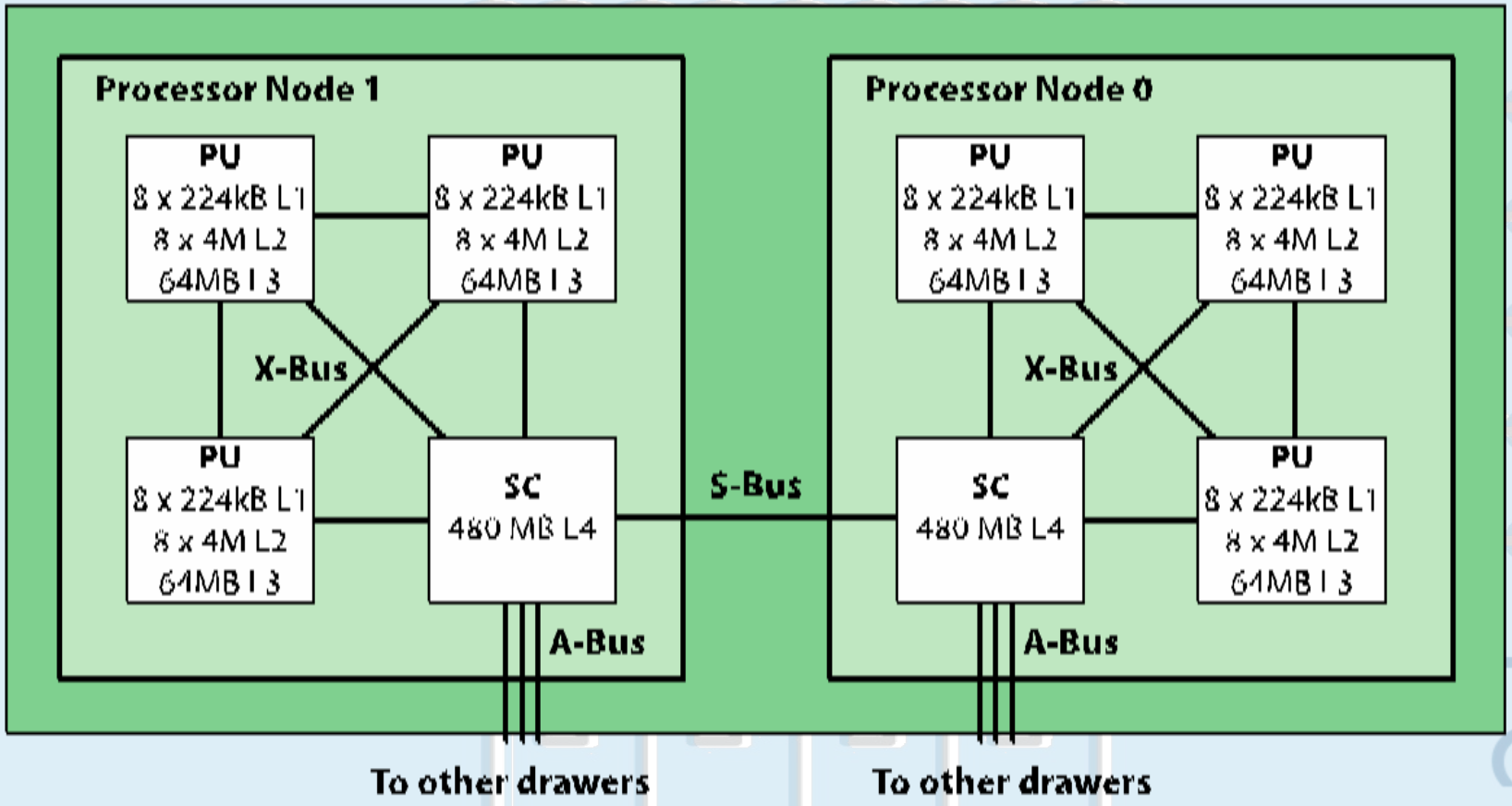


Table 5.5
Pentium 4 Cache Operating Modes

Control Bits		Operating Mode		
CD	**NW**	**Cache Fills**	**Write Throughs**	**Invalidates**
0	0	Enabled	Enabled	Enabled
1	0	Disabled	Enabled	Enabled
1	1	Disabled	Disabled	Disabled

Note: CD = 0; NW = 1 is an invalid combination.

Figure 5.18
IBM z13 CPC Drawer Logical Structure



Cache Timing Model

- Direct-mapped cache access
  - The first operation is checking the Tag field of an address against the tag value in the line designated by the Line field
  - If there is not a match (miss), the operation is complete
  - If there is a match (hit), the cache hardware reads the data block from the line in the cache and then fetches the byte or word indicated by the Offset field of the address
  - An advantage is that it allows simple and fast speculation
- Fully associative cache
  - The line number is not known until the tag comparison is competed
  - The hit time is the same as for direct-mapped
  - Because this is a content-addressable memory, the miss time is simply the tag comparison time
- Set associative
  - It is not possible to transmit bytes and compare tags in parallel as can be done with direct-mapped with speculative access
  - However, the circuitry can be designed so that the data block from each line in a set can be loaded and then transmitted once the tag check is made

Table 5.6
Cache Timing Equations

	Time for hit	Time for miss
Direct-Mapped	$t_{hit} = t_{rl} + t_{xb} + t_{ct}$	$t_{miss} = t_{rl} + t_{ct}$
Direct-Mapped with Speculation	$t_{hit} = t_{rl} + t_{xb}$	$t_{miss} = t_{rl} + t_{ct}$
Fully Associative	$t_{hit} = t_{rl} + t_{xb} + t_{ct}$	$t_{miss} = t_{ct}$
Set-Associative	$t_{hit} = t_{rl} + t_{xb} + t_{ct}$	$t_{miss} = t_{rl} + t_{ct}$
Set-Associative with Way Prediction	$t_{hit} = t_{rl} + t_{xb} + (1 - F_p)\, t_{ct}$	$T= = t_{rl} + t_{ct}$

Table 5.7
Cache Performance Improvement Techniques

Technique	Reduce t_1	Reduce $(1 - h_1)$	Reduce $t_{penalty}$
Way Prediction	□□		
Cache Capacity	Small	Large	
Line Size	Small	Large	
Degree of Associativity	Decrease	Increase	
More Flexible Replacement Policies		□□	
Cache Unity	Split I-cache and D-cache	Unified cache	
Prefetching		□□	
Write Through		Write allocate	No write allocate
Critical Word First			□□
Victim Cache			□□
Wider Busses			□□

Summary

Chapter 5

- Cache memory principles
- Intel x86 cache organization
- The IBM z13 cache organization
- Cache performance modules
  - Cache timing model
  - Design option for improving performance

•Cache

•Memory

- Elements of cache design
  - Cache addresses
  - Cache size
  - Logical cache organization
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches
  - Inclusion policy