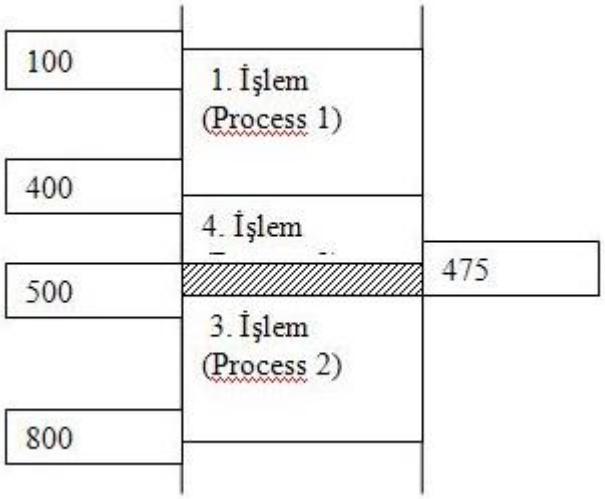
Memory Hierarchy

Hamza Osman İLHAN

hoilhan@yildiz.edu.tr

YTU-CE D037

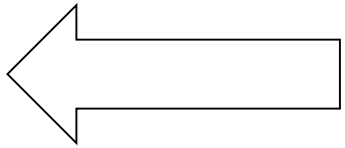
Memory Frames and Pages



İşlem 1 (Process 1)	
Adres	Sayfa
0-100	1
100-200	2
200-275	3

İşlem 2 (Process 2)	
Adres	Sayfa
0-100	1
100-200	2
200-300	3
300-325	4

Sayfa Tablosu	
Sayfa	Çerçeve
İ1-1	1
İ1-2	2
İ1-3	3
İ2-1	4
İ2-2	5
İ2-3	6
İ2-4	7



Hafıza (RAM)	
Adres	Çerçeve
0-100	1
100-200	2
200-300	3
300-400	4
400-500	5
500-600	6
600-700	7
700-800	8
800-900	9
900-1000	10

Hafıza (RAM)		
Adres	Çerçeve	Sayfa
0-100	1	İ1-1
100-200	2	İ1-2
200-300	3	İ1-3
300-400	4	İ2-1
400-500	5	İ2-2
500-600	6	İ2-3
600-700	7	İ2-4
700-800	8	
800-900	9	
900-1000	10	

Paging Example

2. işlem mantıksal adresi 312'ye erişmek istesin

Mantıksal Adres / Sayfa Boyutu = Sayfa Numarası

312 / 100 = 3 3 + 1 = 4

Mantıksal Adres % Sayfa Boyutu = fark miktarı (offset)

312 % 100 = 12

Hafıza (RAM)	
Adres	Çerçeve
0-100	1
100-200	2
200-300	3
300-400	4
400-500	5
500-600	6
600-700	7
700-800	8
800-900	9
900-1000	10

Sayfa Tablosu	
Sayfa	Çerçeve
İ1-1	1
İ1-2	2
İ1-3	3
İ2-1	4
İ2-2	5
İ2-3	6
İ2-4	7

7. çerçeve 600. adresten başlamaktadır bu durumda fiziksel adres:
600 + 12 = 612

Virtual Memory

- Cache memory enhances performance by providing faster memory access speed.
- Virtual memory enhances performance by providing greater memory capacity, without the expense of adding main memory.
- Instead, a portion of a disk drive serves as an extension of main memory.
- If a system uses paging, virtual memory partitions main memory into individually managed *page frames*, that are written *(or paged)* to disk when they are not immediately needed.

Virtual Memory

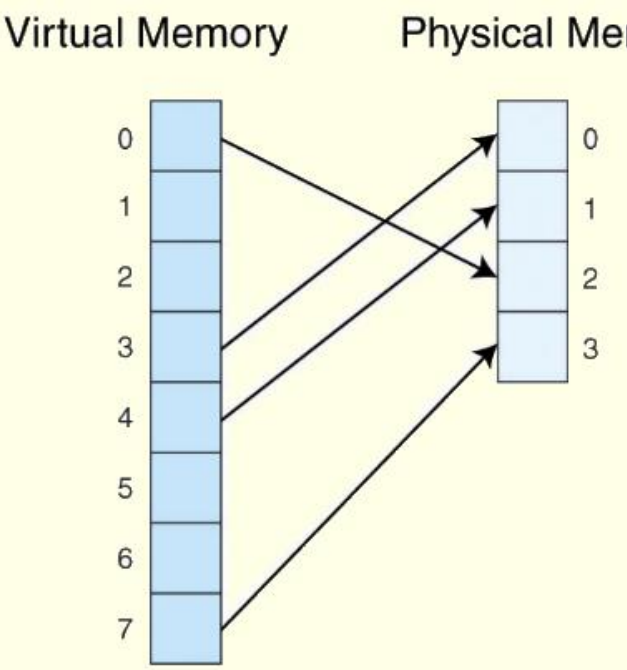
- A *physical address* is the actual memory address of physical memory.
- Programs create *virtual addresses* that are *mapped* to physical addresses by the memory manager.
- *Page faults* occur when a logical address requires that a page be brought in from disk.
- *Memory fragmentation* occurs when the paging process results in the creation of small, unusable clusters of memory addresses.

Virtual Memory

- Main memory and virtual memory are divided into equal sized pages.
- The entire address space required by a process need not be in memory at once. Some parts can be on disk, while others are in main memory.
- Further, the pages allocated to a process do not need to be stored contiguously-- either on disk or in memory.
- In this way, only the needed pages are in memory at any time, the unnecessary pages are in slower disk storage.

Virtual Memory

- Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a *page table* (shown below).
- There is one page table for each active process.



Page Table

Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Virtual Memory

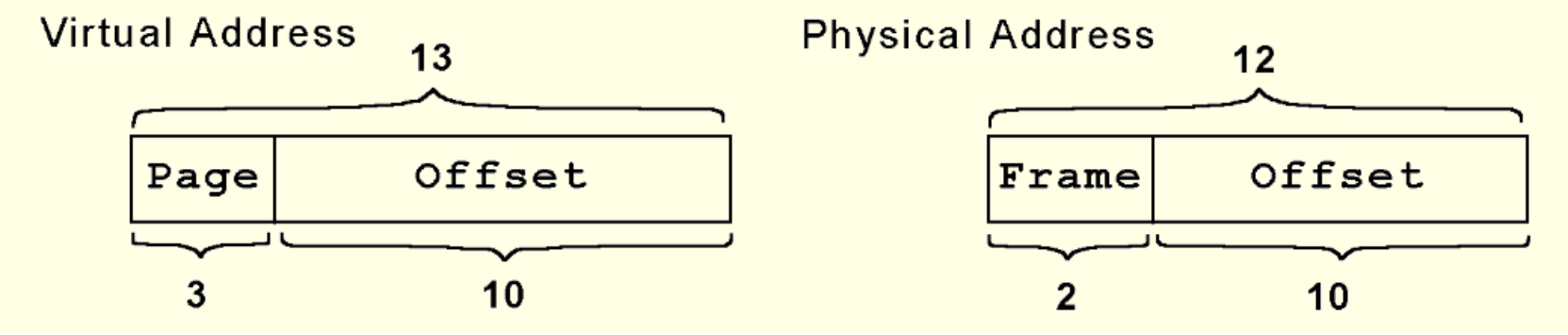
- When a process generates a virtual address, the operating system translates it into a physical memory address.
- To accomplish this, the virtual address is divided into two fields: A *page* field, and an *offset* field.
- The page field determines the page location of the address, and the offset indicates the location of the address within the page.
- The logical page number is translated into a physical page frame through a lookup in the page table.

Virtual Memory

- If the valid bit is zero in the page table entry for the logical address, this means that the page is not in memory and must be fetched from disk.
  - This is a page fault.
  - If necessary, a page is evicted from memory and is replaced by the page retrieved from disk, and the valid bit is set to 1.
- If the valid bit is 1, the virtual page number is replaced by the physical frame number.
- The data is then accessed by adding the offset to the physical frame number.

Virtual Memory

- As an example, suppose a system has a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing. The page size is 1024.
  - We have $2^{13}/2^{10} = 2^3$ virtual pages.
- A virtual address has 13 bits ($8K = 2^{13}$) with 3 bits for the page field and 10 for the offset, because the page size is 1024.
- A physical memory address requires 12 bits ($4K = 2^{12}$), the first two bits for the page frame and the trailing 10 bits the offset.



Virtual Memory

- Suppose we have the page table shown below.
- What happens when CPU generates address 5459_{10} = 1010101010011_2?

Page Table

	Frame	Valid Bit
Page 0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

	Addresses
Page 0 :	0 - 1023
1 :	1024 - 2047
2 :	2048 - 3071
3 :	3072 - 4095
4 :	4096 - 5119
5 :	5120 - 6143
6 :	6144 - 7167
7 :	7168 - 8191

Virtual Memory

- The address 1010101010011_2 is converted to physical address 010101010011 because the page field 101 is replaced by frame number 01 through a lookup in the page table.

Page Table

	Frame	Valid Bit
Page 0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

	Addresses
Page 0 :	0 - 1023
1 :	1024 - 2047
2 :	2048 - 3071
3 :	3072 - 4095
4 :	4096 - 5119
5 :	5120 - 6143
6 :	6144 - 7167
7 :	7168 - 8191

Virtual Memory

- What happens when the CPU generates address 1000000000100_2?

Page Table

	Frame	Valid Bit
Page 0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

	Addresses
Page 0 :	0 - 1023
1 :	1024 - 2047
2 :	2048 - 3071
3 :	3072 - 4095
4 :	4096 - 5119
5 :	5120 - 6143
6 :	6144 - 7167
7 :	7168 - 8191

Virtual Memory

- Effective access time (EAT) takes all levels of memory into consideration.
- Thus, virtual memory is also a factor in the calculation, and we also have to consider page table access time.
- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk. We have:

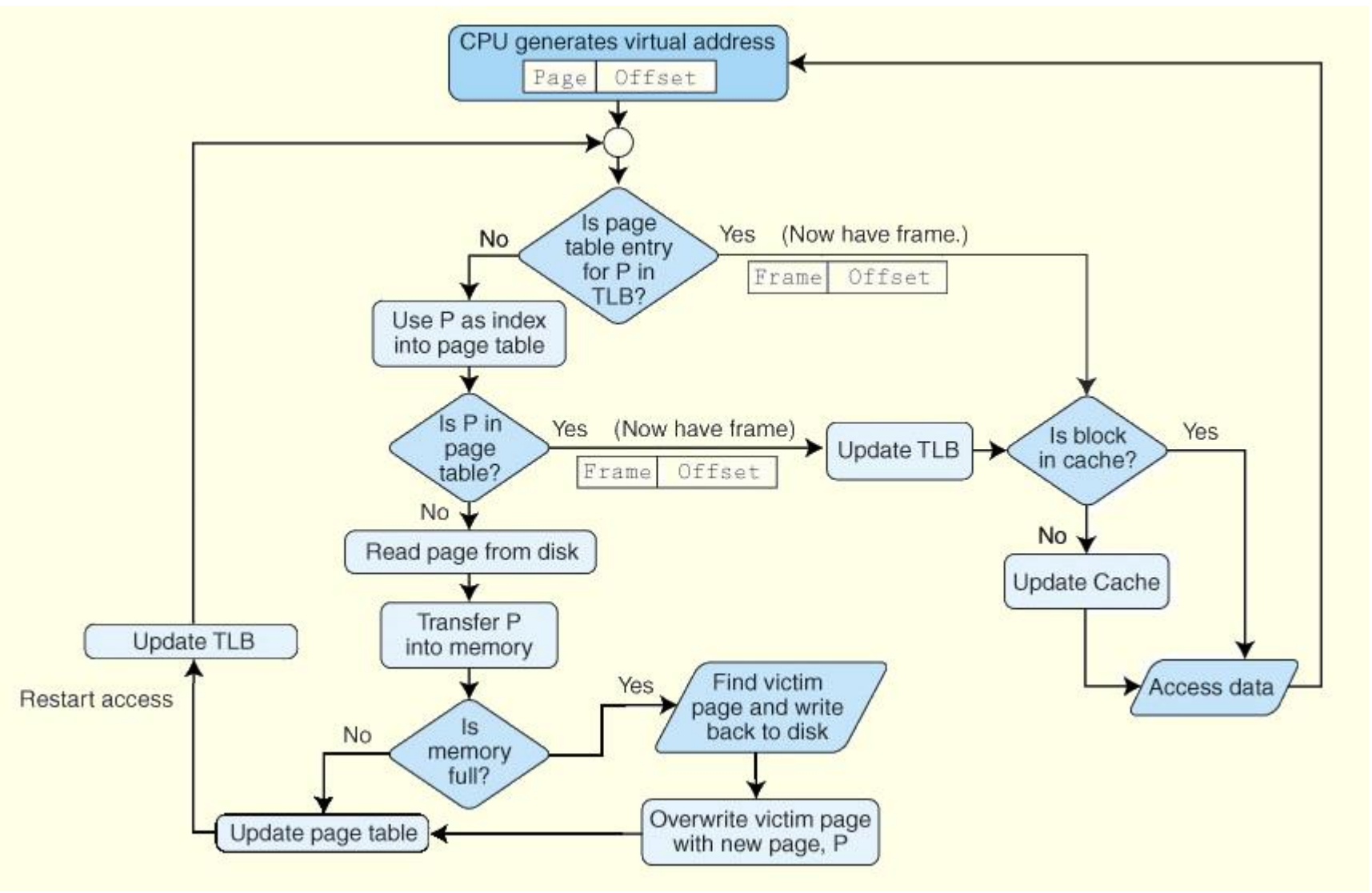
EAT = 0.99(200ns + 200ns) 0.01(10ms) = 100, 396ns.

Virtual Memory

- Even if we had no page faults, the EAT would be 400ns because memory is always read twice: First to access the page table, and second to load the page from memory.
- Because page tables are read constantly, it makes sense to keep them in a special cache called a *translation look-aside buffer* (TLB).
- TLBs are a special associative cache that stores the mapping of virtual pages to physical pages.

The next slide shows how all the pieces fit together.

Virtual Memory



Virtual Memory

- Another approach to virtual memory is the use of *segmentation*.
- Instead of dividing memory into equal-sized pages, virtual address space is divided into variable-length segments, often under the control of the programmer.
- A segment is located through its entry in a segment table, which contains the segment's memory location and a bounds limit that indicates its size.
- After a page fault, the operating system searches for a location in memory large enough to hold the segment that is retrieved from disk.

Virtual Memory

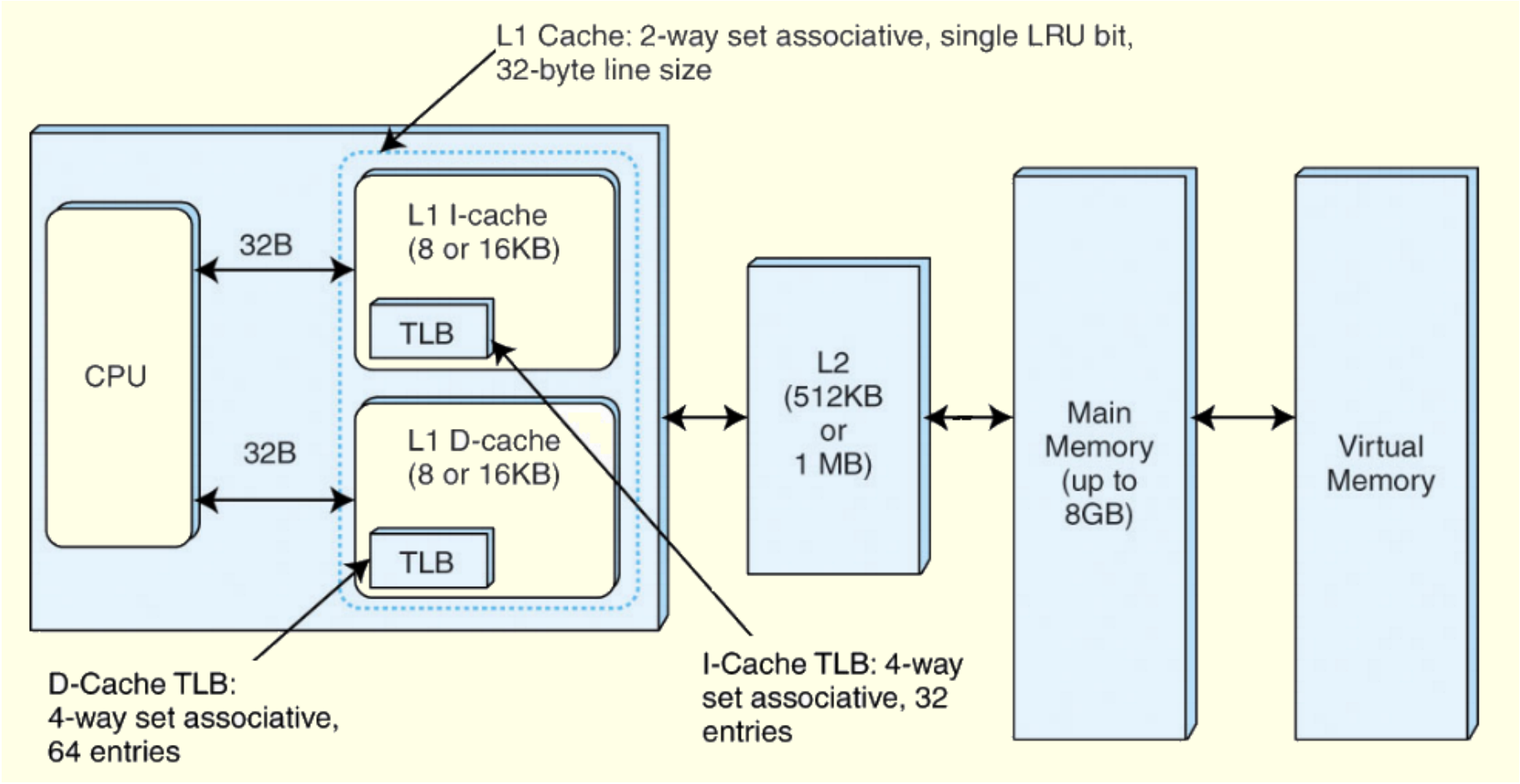
- Both paging and segmentation can cause fragmentation.
- Paging is subject to *internal* fragmentation because a process may not need the entire range of addresses contained within the page. Thus, there may be many pages containing unused fragments of memory.
- Segmentation is subject to *external* fragmentation, which occurs when contiguous chunks of memory become broken up as segments are allocated and deallocated over time.

Real-World Example

- The Pentium architecture supports both paging and segmentation, and they can be used in various combinations including unpaged unsegmented, segmented unpaged, and unsegmented paged.
- The processor supports two levels of cache (L1 and L2), both having a block size of 32 bytes.
- The L1 cache is next to the processor, and the L2 cache sits between the processor and memory.
- The L1 cache is in two parts: and instruction cache (I-cache) and a data cache (D-cache).

The next slide shows this organization schematically.

Real-World Example



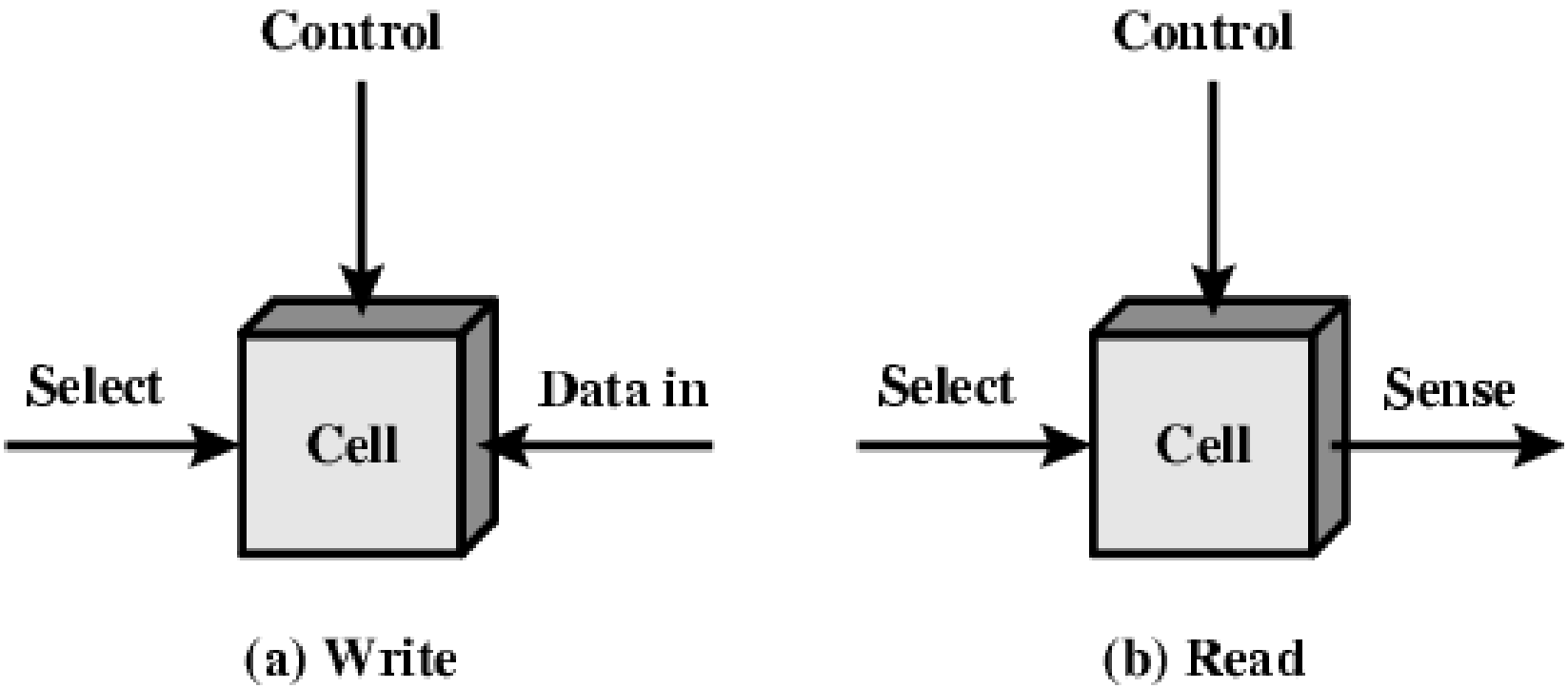
Semiconductor Memory Types

<table>
<tr><th>Memory Type</th><th>Category</th><th>Erasure</th><th>Write Mechanism</th><th>Volatility</th></tr>
<tr><td>Random-access memory (RAM)</td><td>Read-write memory</td><td>Electrically, byte-level</td><td>Electrically</td><td>Volatile</td></tr>
<tr><td>Read-only memory (ROM)</td><td rowspan="2">Read-only memory</td><td rowspan="2">Not possible</td><td>Masks</td><td rowspan="5">Nonvolatile</td></tr>
<tr><td>Programmable ROM (PROM)</td><td rowspan="4">Electrically</td></tr>
<tr><td>Erasable PROM (EPROM)</td><td rowspan="3">Read-mostly memory</td><td>UV light, chip-level</td></tr>
<tr><td>Electrically Erasable PROM (EEPROM)</td><td>Electrically, byte-level</td></tr>
<tr><td>Flash memory</td><td>Electrically, block-level</td></tr>
</table>

Semiconductor Memory

- RAM
  - Misnamed as all semiconductor memory is random access
  - Read/Write
  - Volatile
  - Temporary storage
  - Static or dynamic

Memory Cell Operation



Dynamic RAM

- Bits stored as charge in capacitors
- Charges leak
- Need refreshing even when powered
- Simpler construction
- Smaller per bit
- Less expensive
- Need refresh circuits
- Slower
- Main memory
- Essentially analogue
  - Level of charge determines value

Static RAM

- Bits stored as on/off switches
- No charges to leak
- No refreshing needed when powered
- More complex construction
- Larger per bit
- More expensive
- Does not need refresh circuits
- Faster
- Cache
- Digital
  - Uses flip-flops

SRAM v DRAM

- Both volatile
  - Power needed to preserve data
- Dynamic cell
  - Simpler to build, smaller
  - More dense
  - Less expensive
  - Needs refresh
  - Larger memory units
- Static
  - Faster
  - Cache

Read Only Memory (ROM)

- Permanent storage
  - Nonvolatile
- Used in...
  - Microprogramming
  - Library subroutines
  - Systems programs (BIOS)
  - Function tables

Types of ROM

- Written during manufacture
  - Very expensive for small runs
- Programmable (once)
  - PROM
  - Needs special equipment to program
- Read "mostly"
  - Erasable Programmable (EPROM)
    - Erased by UV
  - Electrically Erasable (EEPROM)
    - Takes much longer to write than read
  - Flash memory
    - Erase whole memory electrically

Organisation in detail

- A 16Mbit chip can be organised as 1M of 16 bit words
- A bit per chip system has 16 lots of 1Mbit chip with bit 1 of each word in chip 1 and so on
- A 16Mbit chip can be organised as a 2048 x 2048 x 4bit array
  - Reduces number of address pins
    - Multiplex row address and column address
    - 11 pins to address (2^{11}=2048)
    - Adding one more pin doubles range of values so x4 capacity