Datorsystem VT 2022

F6

Main memory,
&
Cache memory
&
Virtual memory

Stockholm University

Outline

- Type of computer memory,
- Cache memory
- Virtual memory
- Example

Types of Memory

- *Random access memory, RAM, and read-only-memory, ROM*.
- Dynamic RAM (DRAM) and static RAM (SRAM).
- DRAM consists of capacitors -> slowly leak their charge over time.
- they must be refreshed every few milliseconds to prevent data loss.
- DRAM is “cheap” memory owing to its simple design.

Types of Memory

- SRAM is very fast memory and it doesn't need to be refreshed like DRAM does.
- It is used to build cache memory.
- ROM also does not need to be refreshed.
- ROM is used to store permanent, or semi-permanent data that persists even while the system is turned off.

Types of Computer Memory

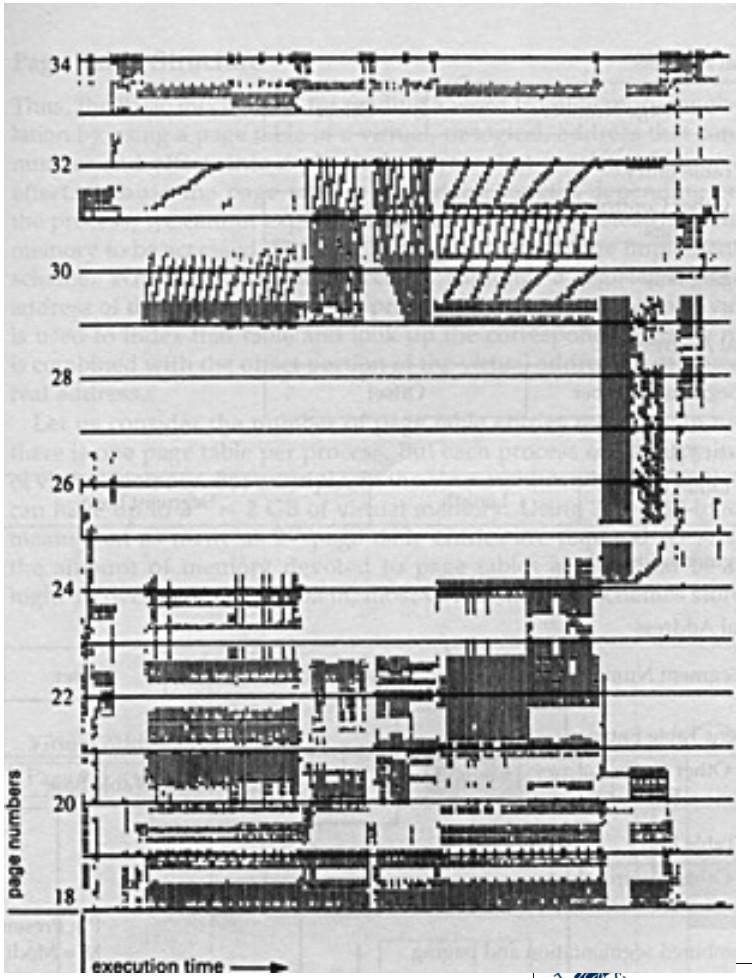
- Characteristics
  - Volatile vs. Nonvolatile
  - Solid State vs. Mechanical
  - Read only vs. Read-write (RW)
  - Expense
  - Speed
  - Storage capacity (a function of size & expense)

Types of Computer Memory

- Registers (made up of flip-flops-like circuits)
  - Volatile, small, expensive, fast, solid state, RW
- DRAM (grid format using capacitance)
  - Connected to the rest of the CPU via a memory bus
  - Volatile, now large, expensive, fast, solid state, RW
- ROM (grid format using diodes)
  - Nonvolatile, Read-only, cheap, fast
- Hard Disk
  - Nonvolatile, mechanical, RW, cheap, slow
- Flash Memory (grid format using fancy electronics)
  - Nonvolatile, solid state, RW, expensive, fast, small
  - Versus hard disk: silent, lighter, faster … but expensive!
- Flash RAM (e.g., car radio presets)
  - Like Flash memory, but needs power
- Other external storage: CD-ROM, Floppy, Zip

Caching

- Caches store items that have been used recently for faster access
- Based on the notion of locality
  - Something used recently is more likely to be used again soon than other items.
- Can be done using
  - Registers
  - RAM
  - Hard Disk, etc

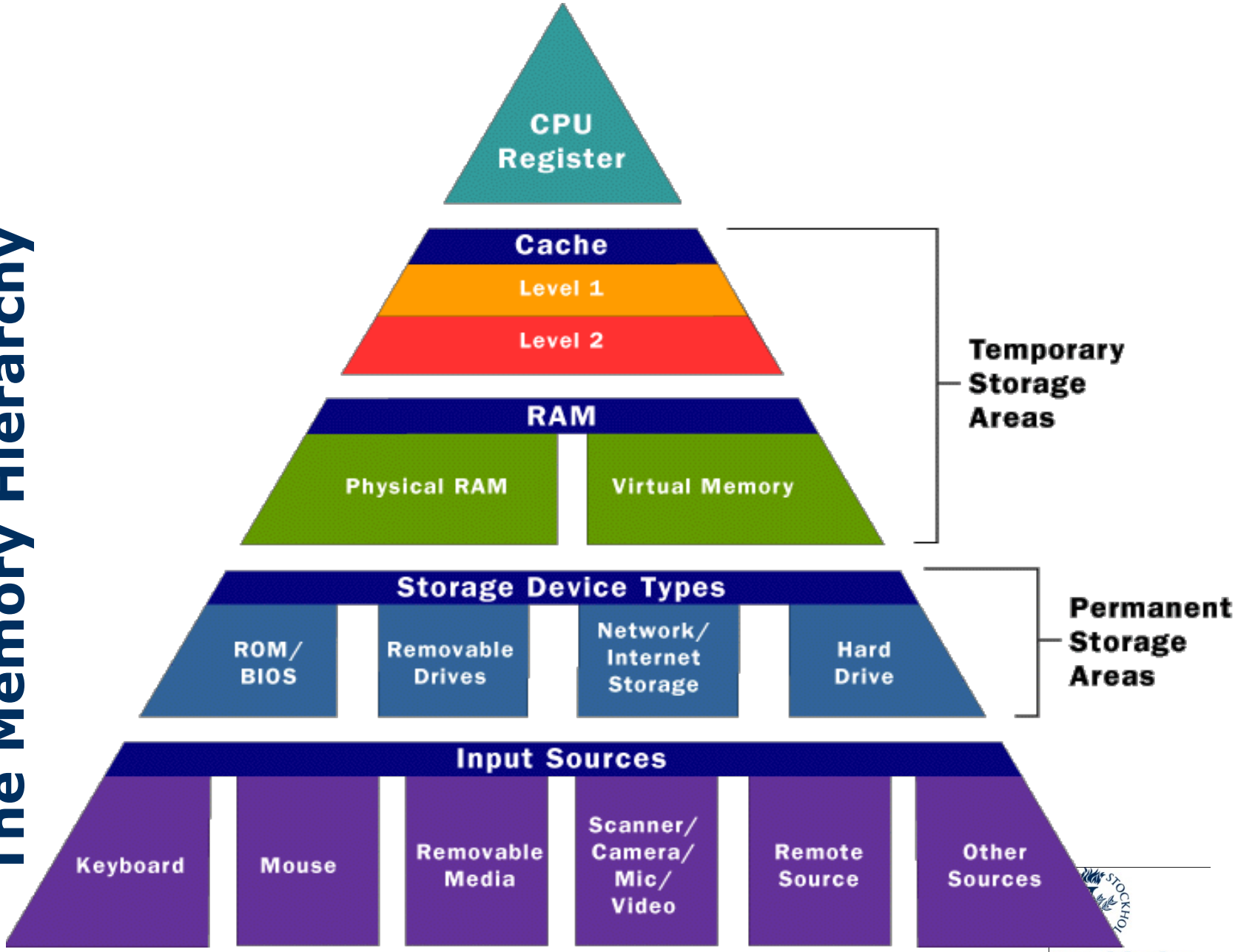


Stockholm University

Types of Computer Memory

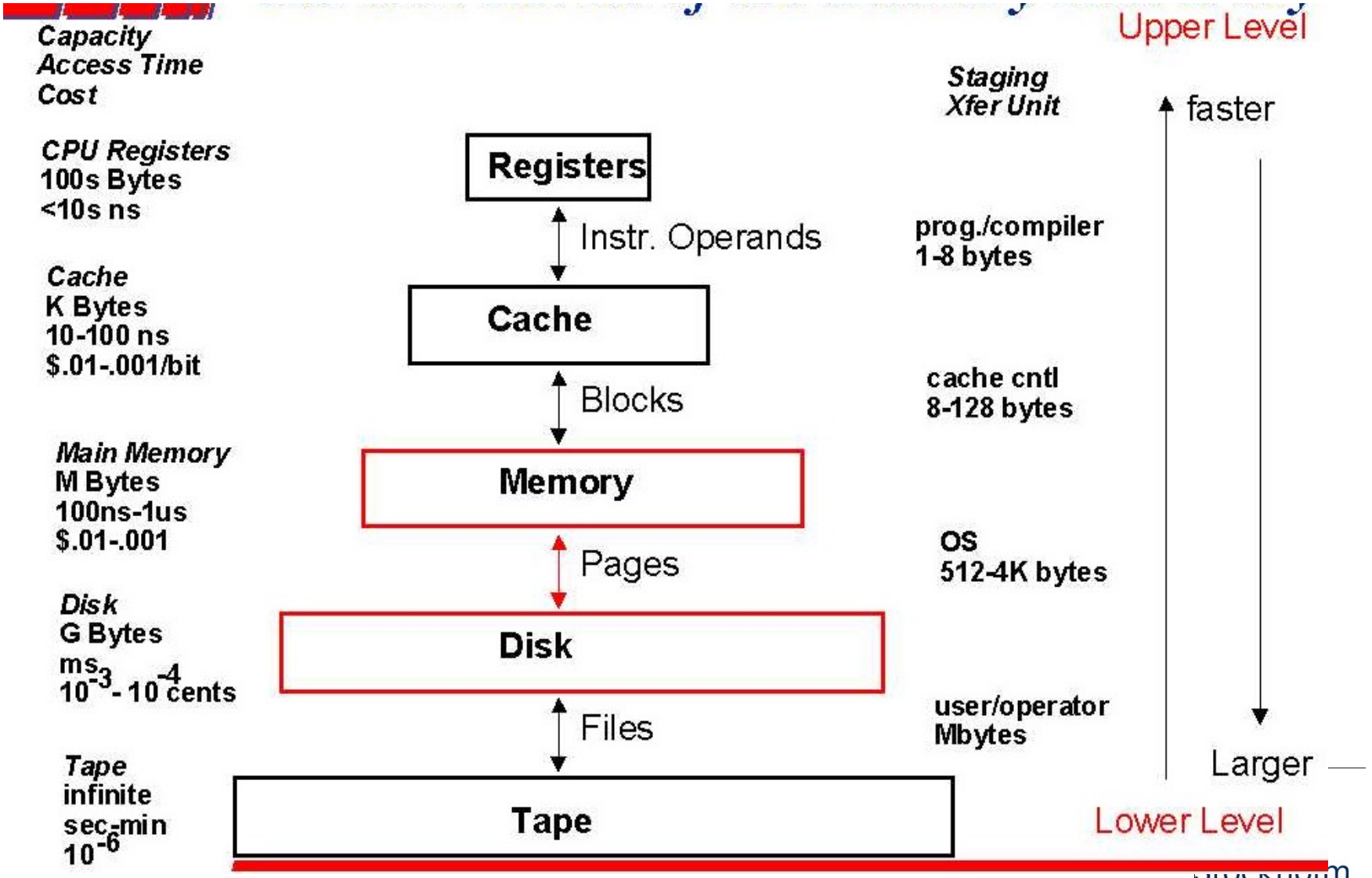
- Cache
  - The term refers to how the memory is used
  - Can be volatile or on disk
    - **L1 cache** - Memory accesses at full microprocessor speed (10 nanoseconds, 4 kilobytes to 16 kilobytes in size)
    - **L2 cache** - Memory access of type SRAM (around 20 to 30 nanoseconds, 128 kilobytes to 512 kilobytes in size)
    - **Main memory** - Memory access of type RAM (around 60 nanoseconds, 2 gigabytes to 64 gigabytes in size)
    - **Hard disk** - Mechanical, slow (around 12 milliseconds, 100 gigabyte to 1 terabytes in size)
    - **Internet** - Incredibly slow (between 1 second and 3 days, unlimited size)

The Memory Hierarchy



Stockholm University

Levels of a Memory Hierarchy

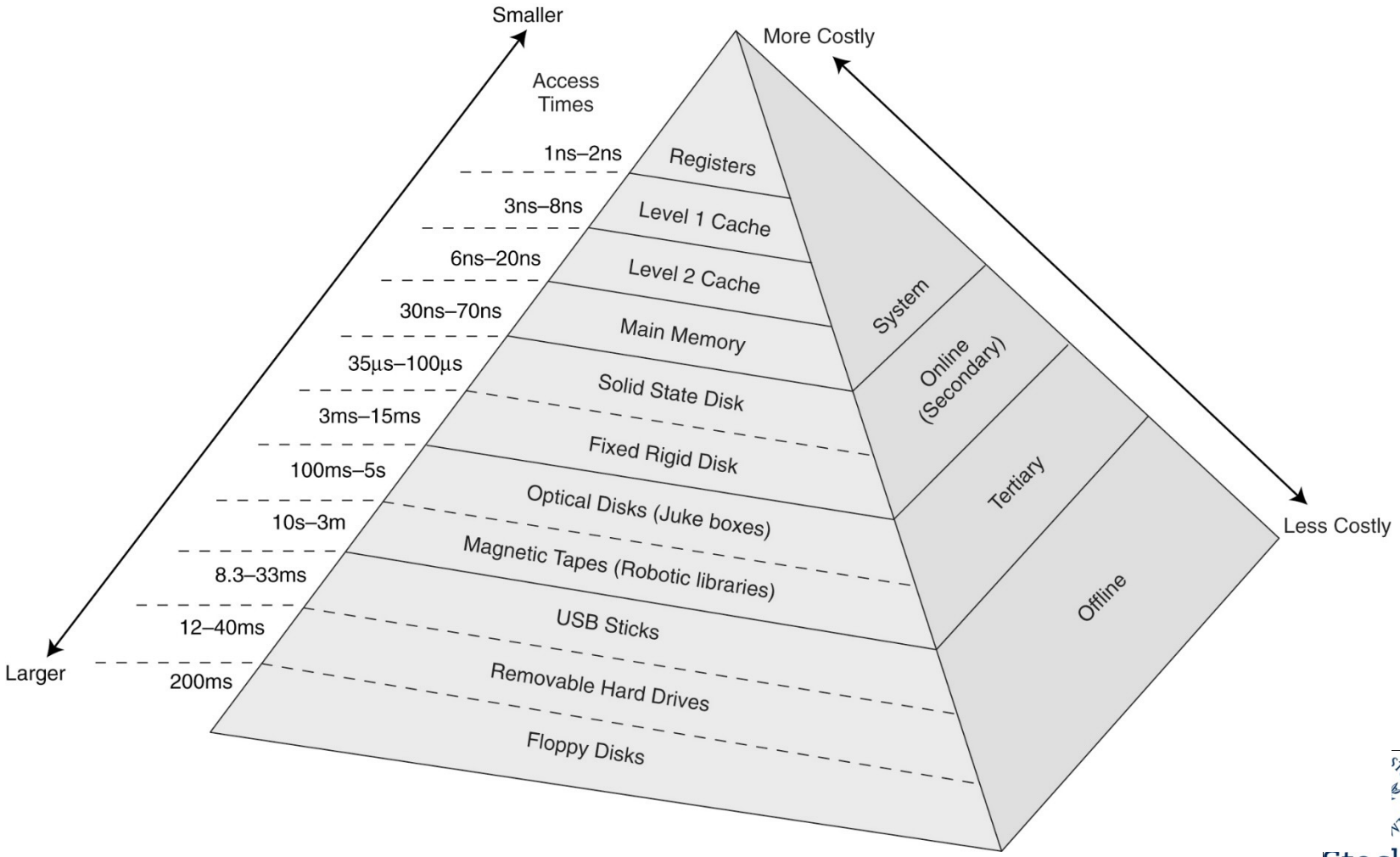


The Memory Hierarchy

- Faster memory is more expensive than slower memory.
- To provide the **best performance** at the lowest cost, memory is organized in a **hierarchical** fashion.
- Small, fast storage elements are kept in the CPU, larger, slower main memory is accessed through the data bus.
- Larger, (almost) permanent storage in the form of disk and tape drives is still further from the CPU.

The Memory Hierarchy

- This storage organization can be thought of as a pyramid:



Stockholm University

The Memory Hierarchy

- Virtual memory -> implemented using a hard drive
  - it extends the address space from RAM to the hard drive.
- Virtual memory provides **more space**:
- Cache memory provides **speed.**

The Memory Hierarchy

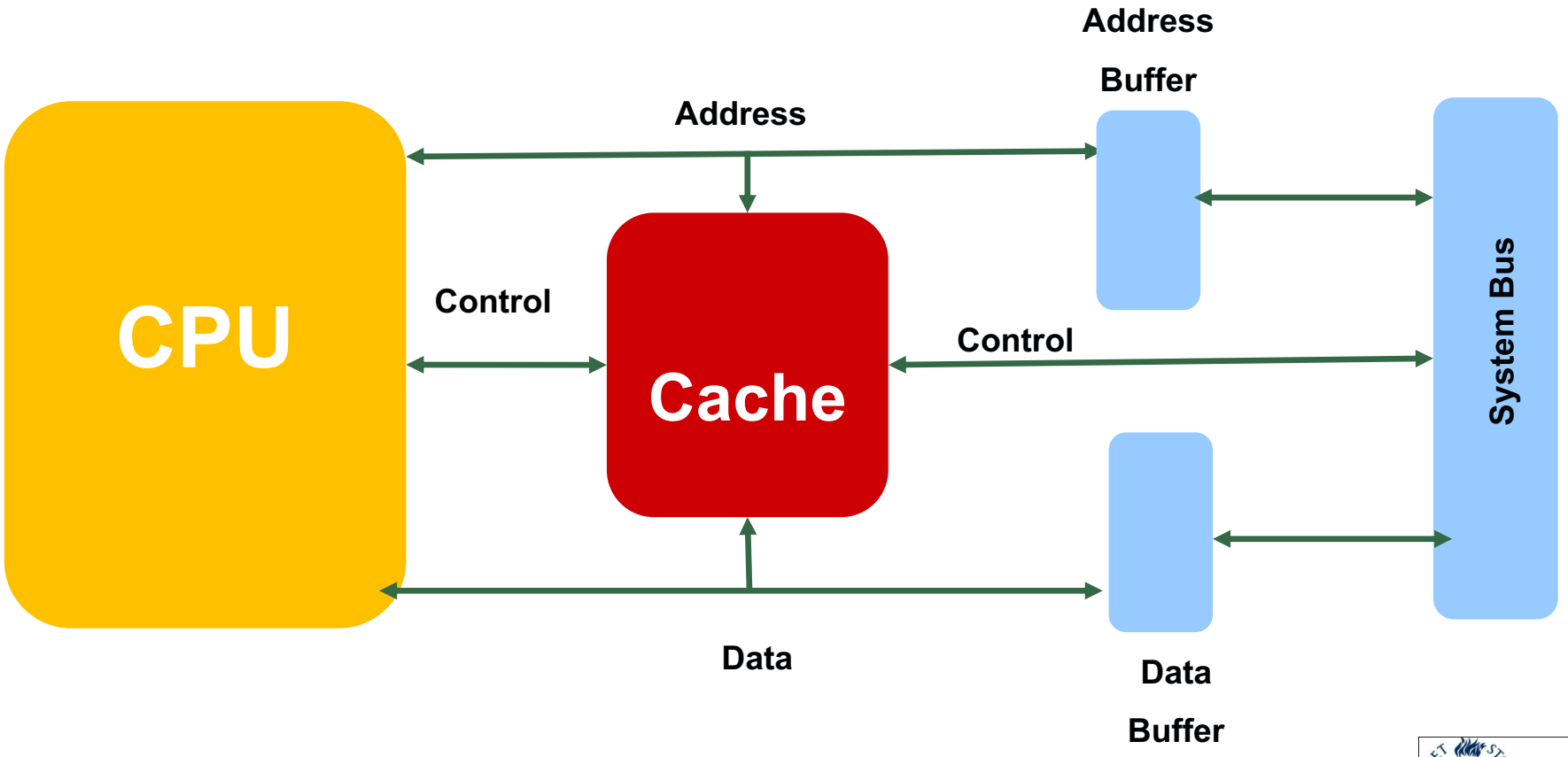
- CPU **->**
  - sends a request to nearest memory->usually cache.
  - If data not in cache-> main memory
  - If data not in main memory-> disk
- Once the data is located, -> the data, and a number of its nearby data elements are fetched into cache memory.
  - Hit -> data is found
  - Miss -> not found

Cache Memory

- Small amount of fast memory
- Sits between normal main memory an CPU
- May be located on CPU chip or module

Stockholm University

Cache Organization



Stockholm University

working of Cache Memory

- The CPU initially looks in the Cache for the data it needs
- If the data is there, it will retrieve it and process it
- If the data is not there then the CPU accesses the system memory and then puts a copy of the new data in the cache before processing it.
- Next time if the CPU needs to access the same data again it will just retrieve the data from the cache instead of going through the loading process again.

Stockholm University

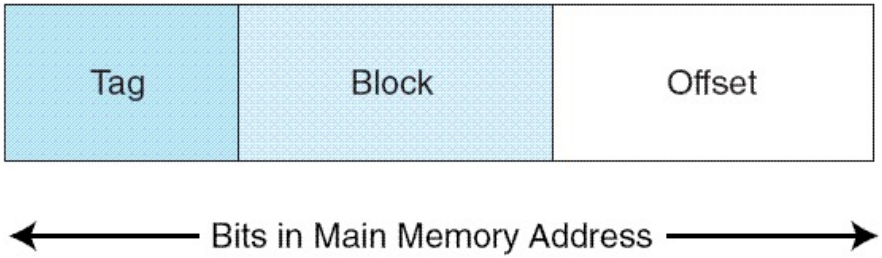
Cache Memory

- **Purpose** of cache memory
  - **to speed up accesses** -> storing recently used data closer to the CPU.
- Cache is **smaller** than main memory.
- Access time is a fraction of main memory.
- Cache is accessed by **content**
  - it is often called ***content addressable memory***.
- A single large cache memory -> it takes longer to search.

Cache Memory

- Content addressable cache memory is
  - **subset of the bits** of a

    main memory address -> *field*.
  - Many blocks of main memory map to a single block of **cache**
  - A *tag* field in the cache block distinguishes one cached memory block from another.
  - A *valid bit* indicates -> the cache block is being used.
  - An *offset field* points to the desired data in the block.



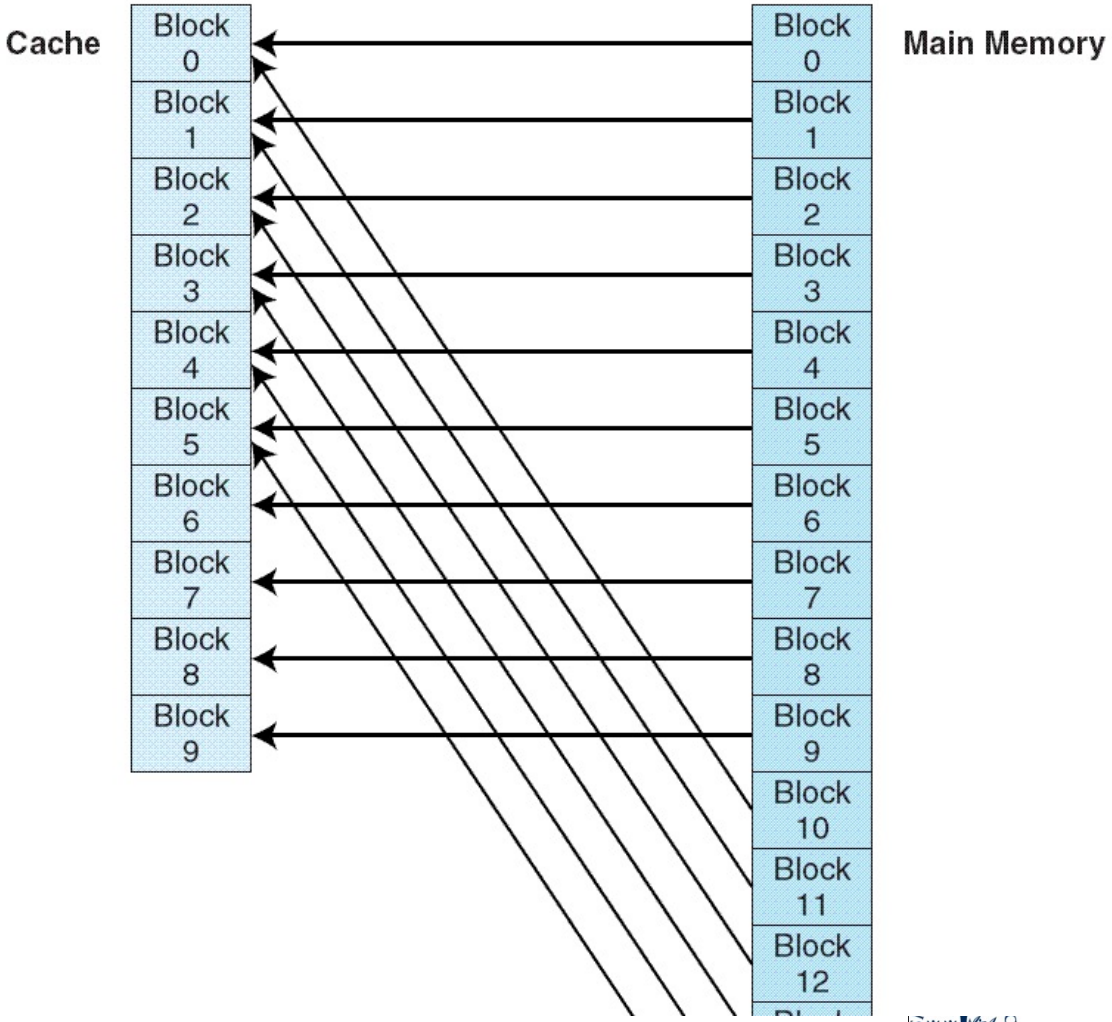
Stockholm University

Cache Memory

- *Direct mapped cache*
- Cache consisting of *N* blocks of cache, block *X* of main memory maps to cache block *Y* = *X* mod *N*.
- Block 7 of cache hold blocks 7, 17, 27, 37, . . . of main memory.

Cache Memory

Direct mapped cache



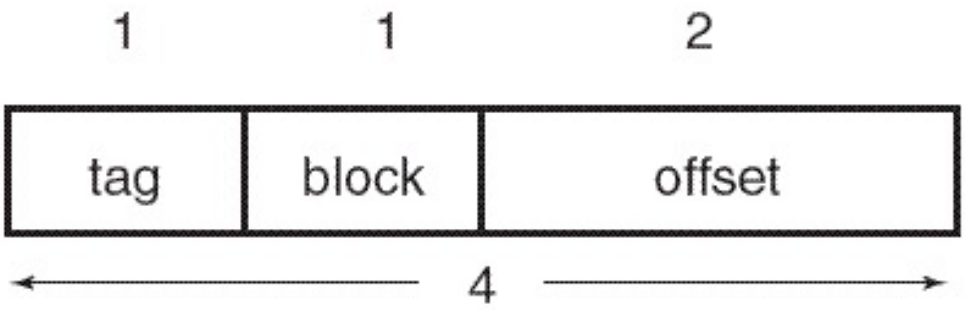
Stockholm University

Cache Memory

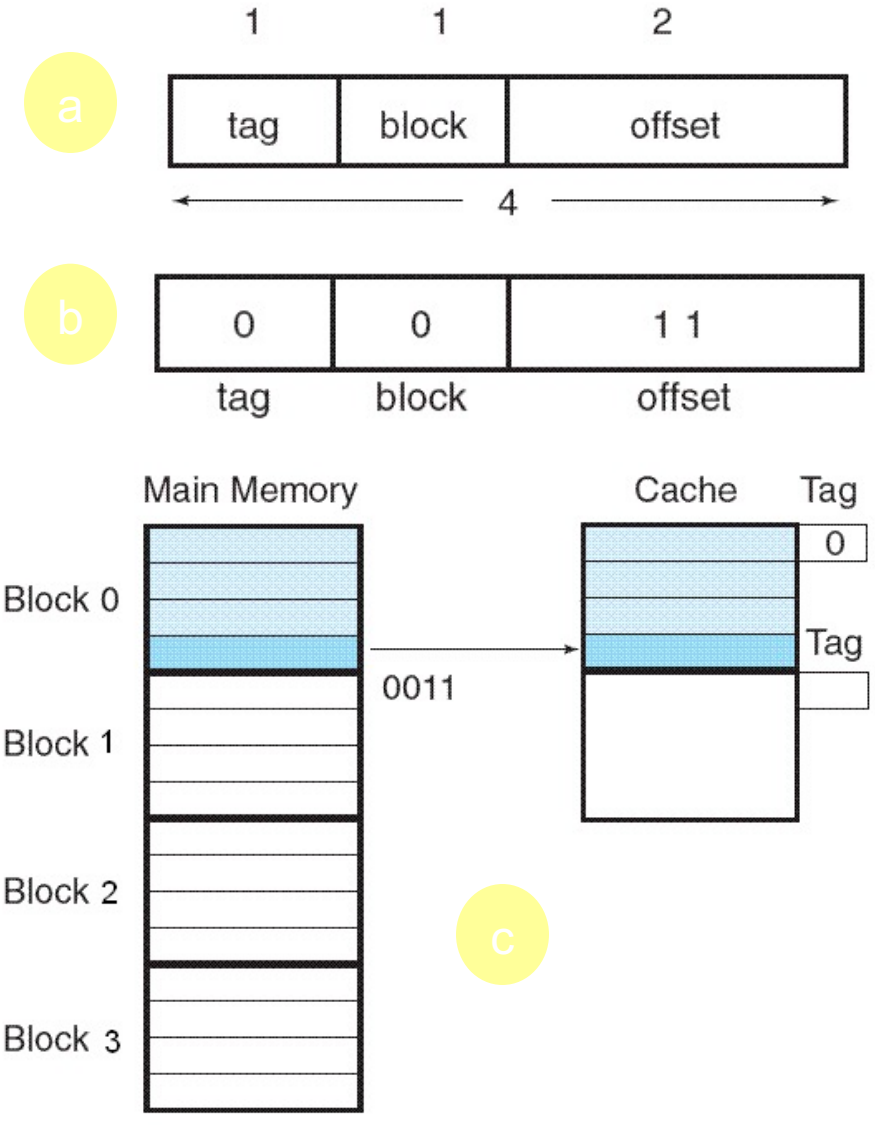
- EX1: main memory consisting of four blocks, and a cache with two blocks.
  - Each block is 4 words.
  - Block 0 and 2 of main memory map to Block 0 of cache.
  - Blocks 1 and 3 of main memory map to Block 1 of cache.

Cache Memory

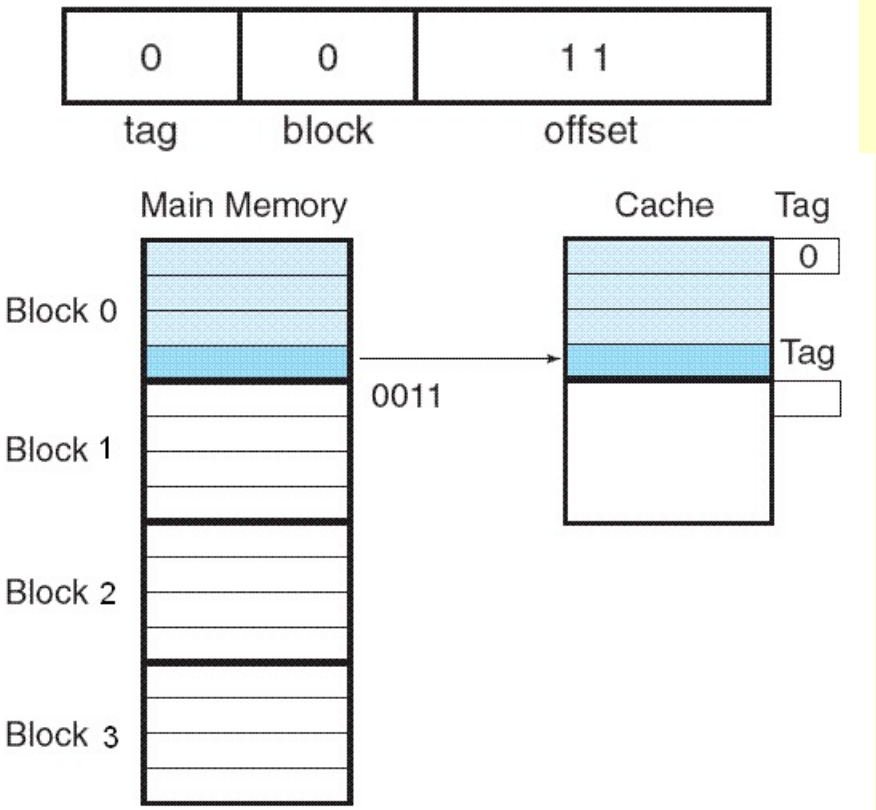
- EXAMPLE Consider a word-addressable main memory consisting of 4 blocks, and a cache with 2 blocks, where each block is 4 words.
  - First, we need to determine the address format for mapping. Each block is 4 words, so the offset field must contain 2 bits; there are 2 blocks in cache, so the block field must contain 1 bit; this leaves 1 bit for the tag (as a main memory address has 4 bits because there are a total of 2^4=16 words).

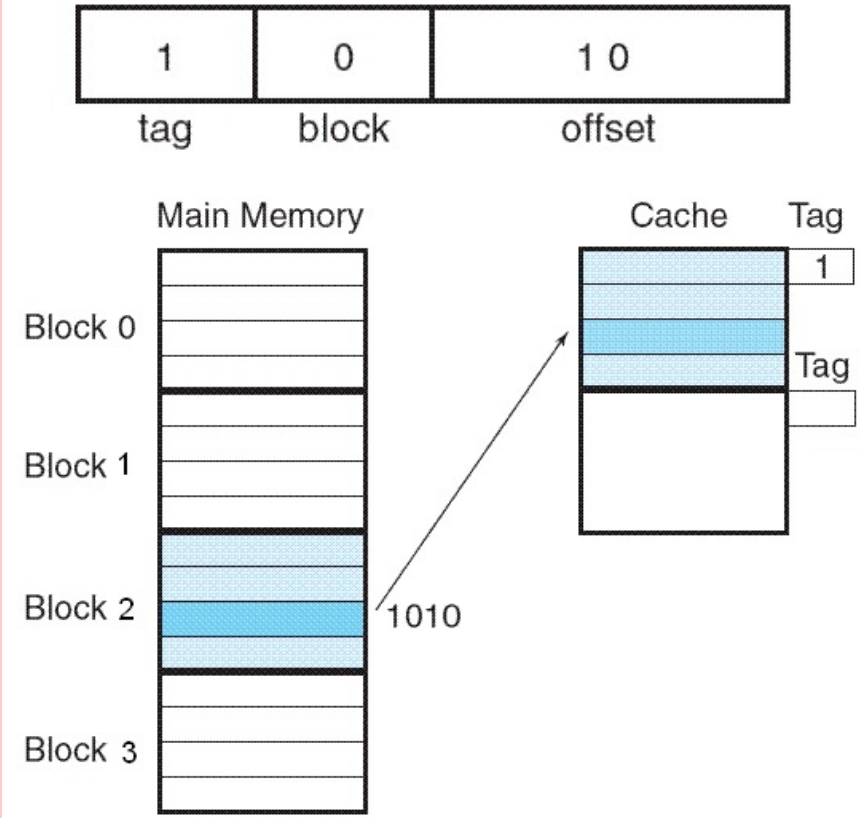


- EX. Cont'd
  - we need to access main memory address 3_{16} (0011 in binary).
  - If we partition 0011 using the address format from Figure a, we get Figure b.
  - the main memory address 0011 maps to cache block 0.



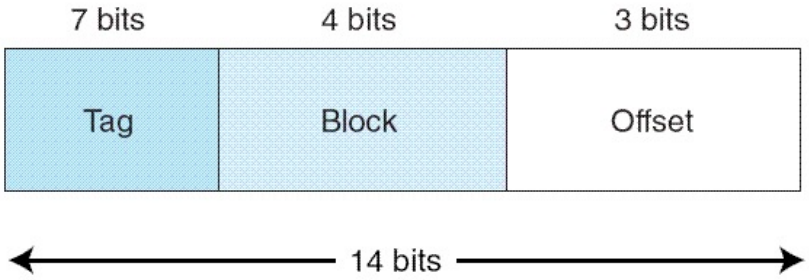
Cache Memory





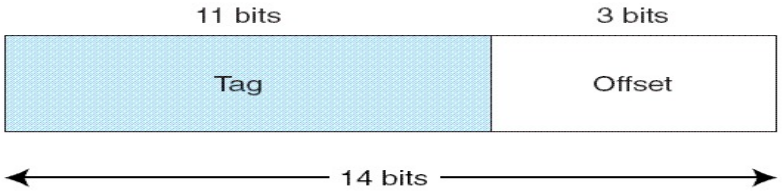
Stockholm University

- EX.2. Assume a byte-addressable memory of 2^{14} bytes, cache has 16 blocks, and each block has 8 bytes.
  - The number of memory blocks are: $\frac{2^{14}}{2^3} = 2^{11}$
  - Each main memory address requires14 bits.
  - Of this 14-bit address field, the rightmost 3 bits reflect the offset field
  - 4 bits to select a specific block in cache, so the block field consists of the middle 4 bits(2^4 =16)
  - The remaining 7 bits make up the tag field.



Stockholm University

- 14-bit memory addresses and a cache with 16 blocks, each block of size 8.
- The field format of a memory reference is:

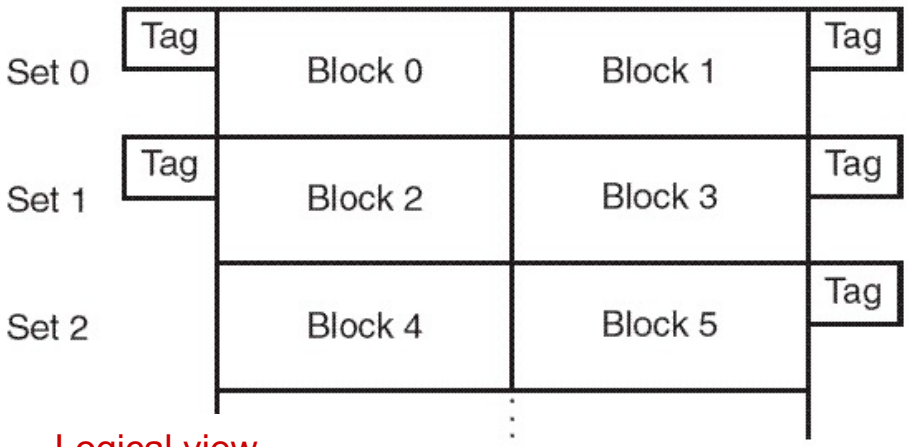


- When the cache is searched -> all tags are searched in parallel
  - to retrieve the data quickly.
- This requires special, costly hardware.

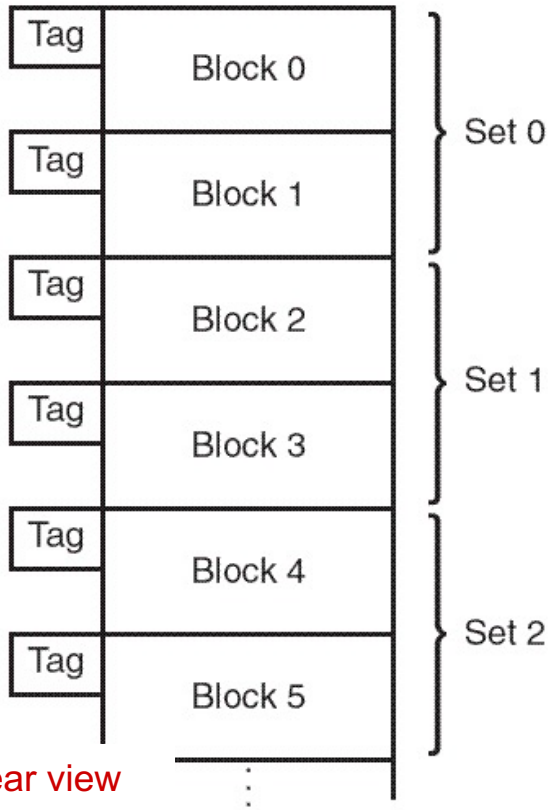
Associative Cache Memory

- Associative cache combines the ideas of direct mapped cache and fully associative cache.
- An *N*-way set associative cache mapping is like direct mapped cache in that a memory reference maps to a particular location in cache.
- Memory reference maps to a set of several cache blocks,
- Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.

- **EX, a 2-way set associative cache can be conceptualized as shown in the schematic below.**
  - **Each set contains two different memory blocks.**



Logical view



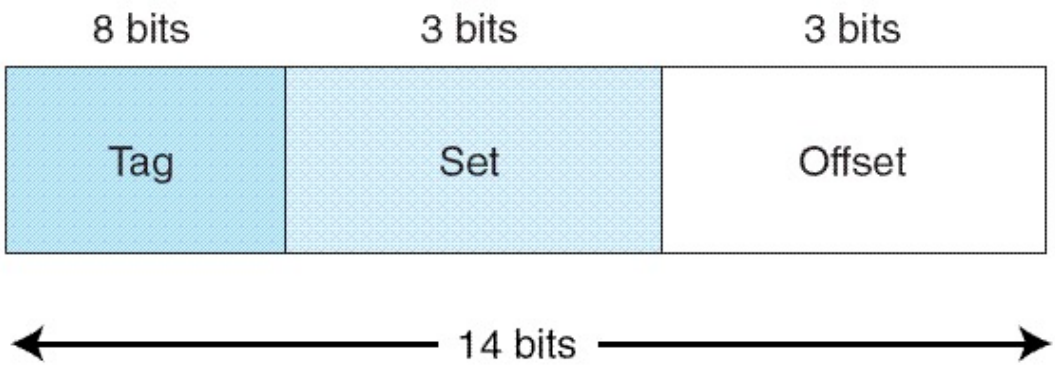
Linear view

Stockholm University

Cache Memory

- In set associative cache mapping, a memory reference is divided into three fields: **tag**, **set**, and **offset**.
- As with direct-mapped cache, the **offset** field chooses the word within the **cache block**, and the **tag** field **uniquely identifies** the memory address.
- The **set** field determines the **set to** which the memory block maps.

- EX, 2-way set associative mapping with a word-addressable main memory of 2^{14} words and a cache with 16 blocks, where each block contains 8 words(2^3).
  - Cache has a total of 16 blocks, and each set has 2 blocks, then there are 8 (2^3) sets in cache.
  - Thus, the set field is 3 bits, the offset field is 3 bits, and the tag field is 8 bits.



Stockholm University

Replacement policy

- ***First-in, first-out* (FIFO)** is a popular cache **replacement policy**.
- In **FIFO,** the block that has been in the cache the longest, regardless of when it was last used.
- A ***random* replacement policy** = implies: It picks a block at random and replaces it with a new block.
- Random replacement can certainly evict a block that will be needed often or needed soon, but it never thrashes.

Cache Coherence: lösningar

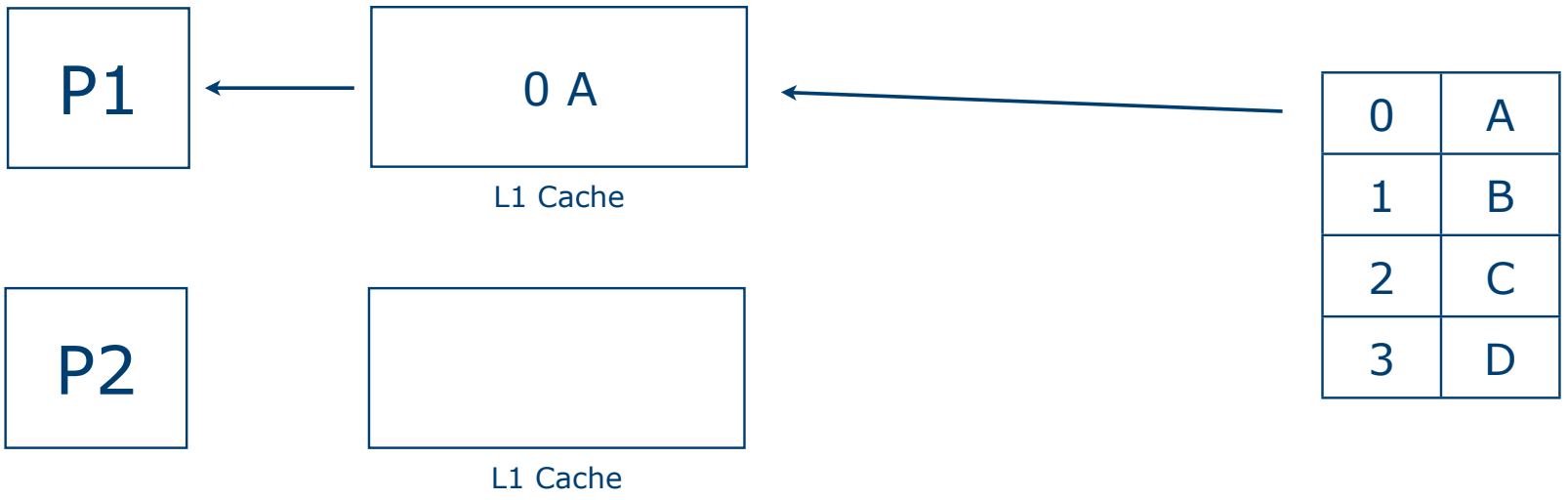
Write back

-Minnesskrivningar skrivs till cacheminnet, överförs till primärminnet vid rensning ur cacheminnet.

Write through

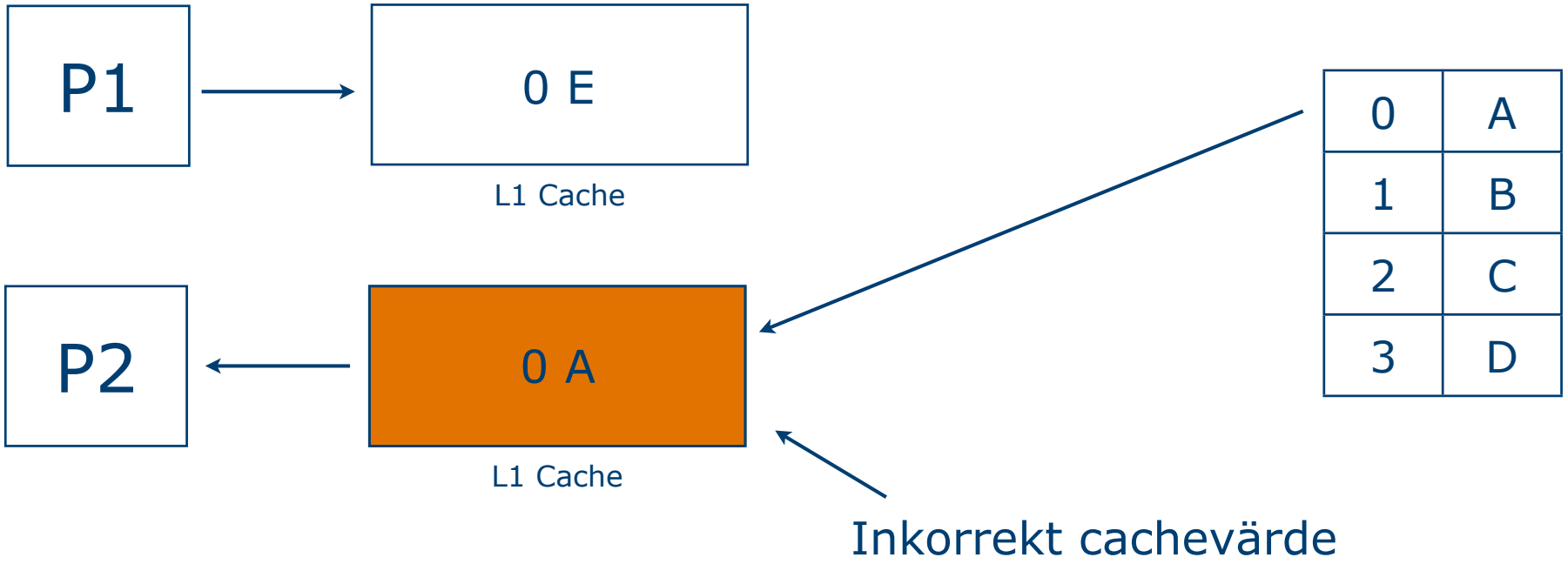
-Minnesskrivningar skrivs alltid till både cacheminne och primärminne.

Cache Coherence: Write back



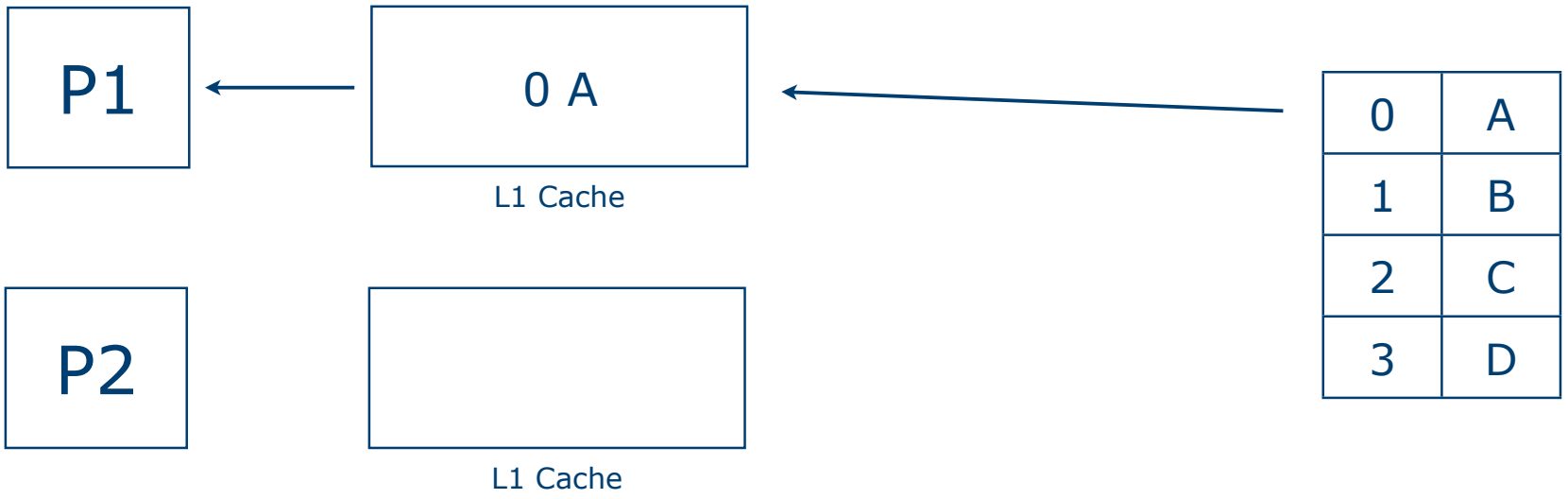
Stockholm University

Cache Coherence: Write back



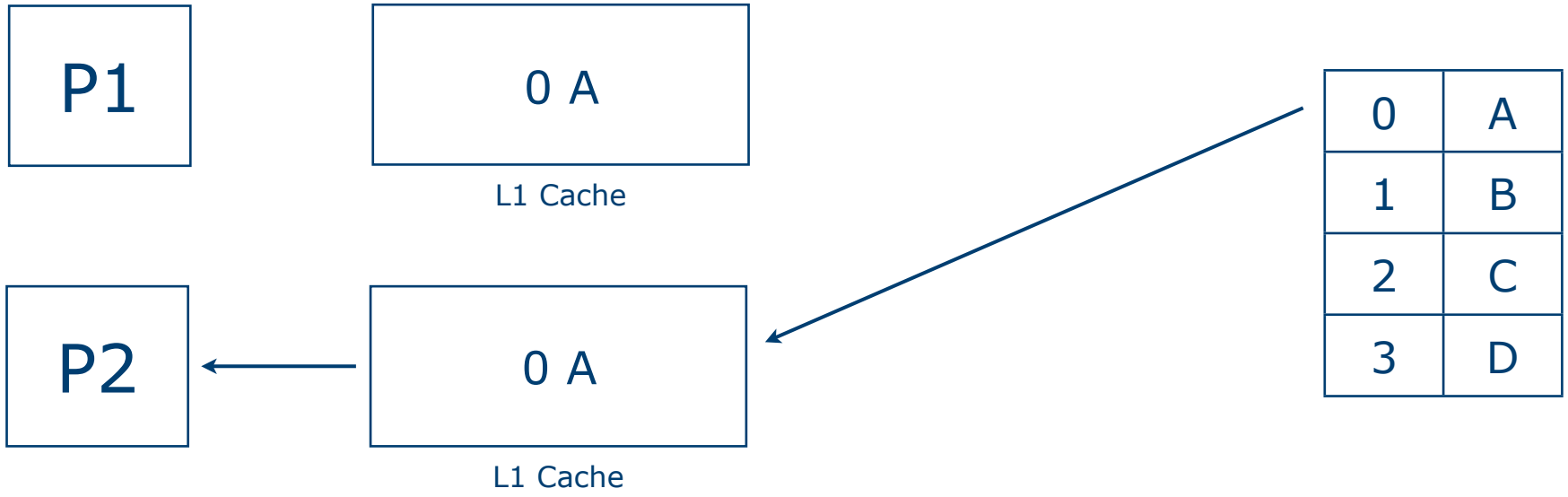
Stockholm University

Cache Coherence: Write Through



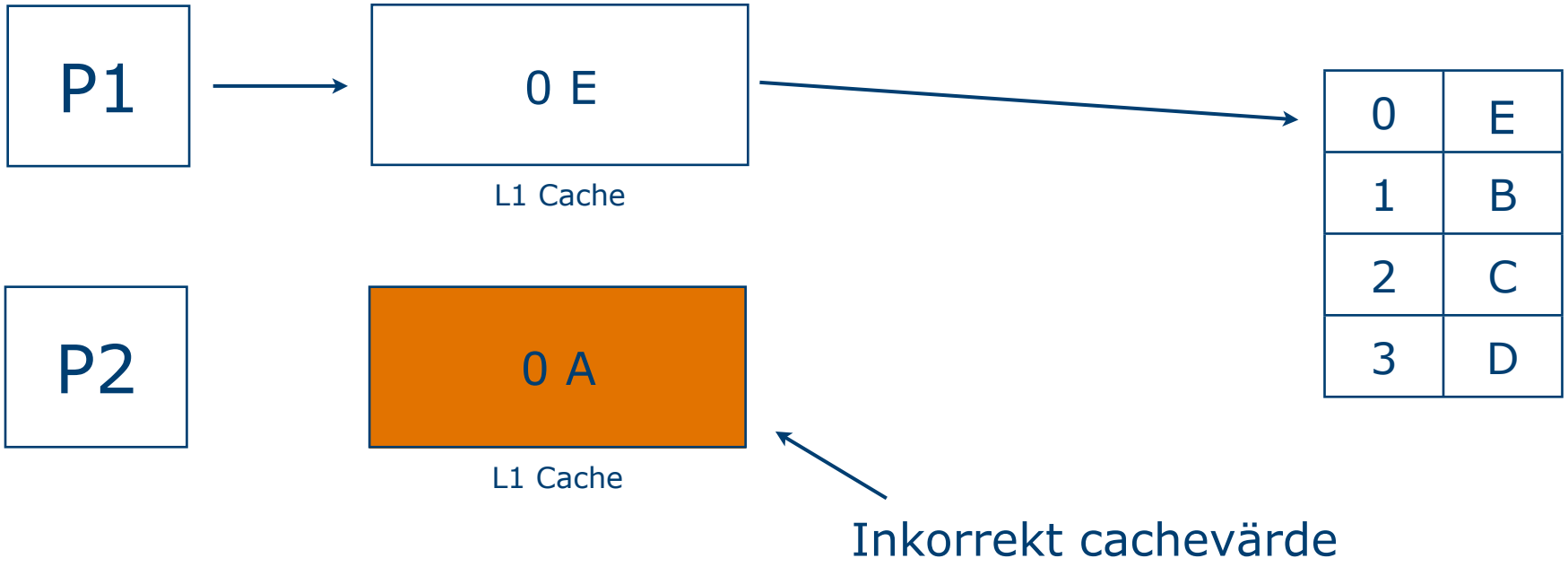
Stockholm University

Cache Coherence: Write Through



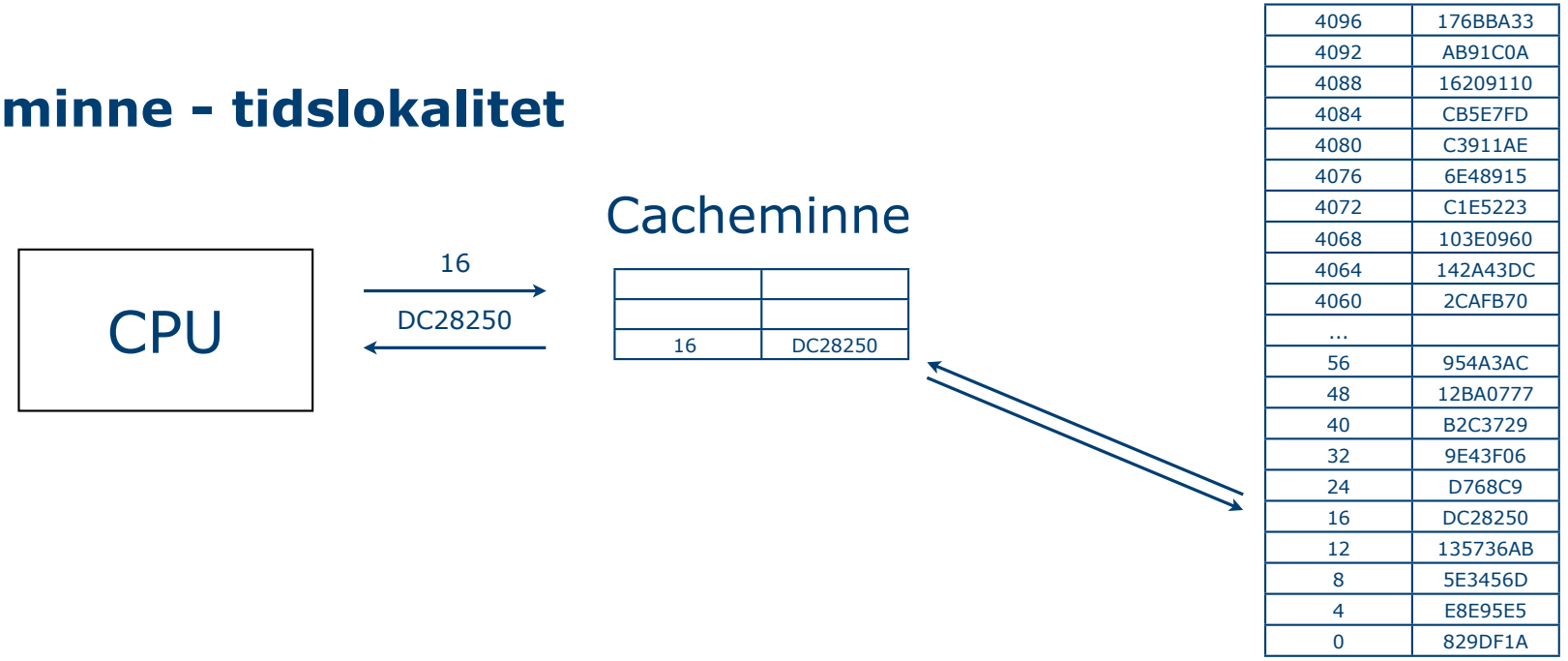
Stockholm University

Cache Coherence: Write Through



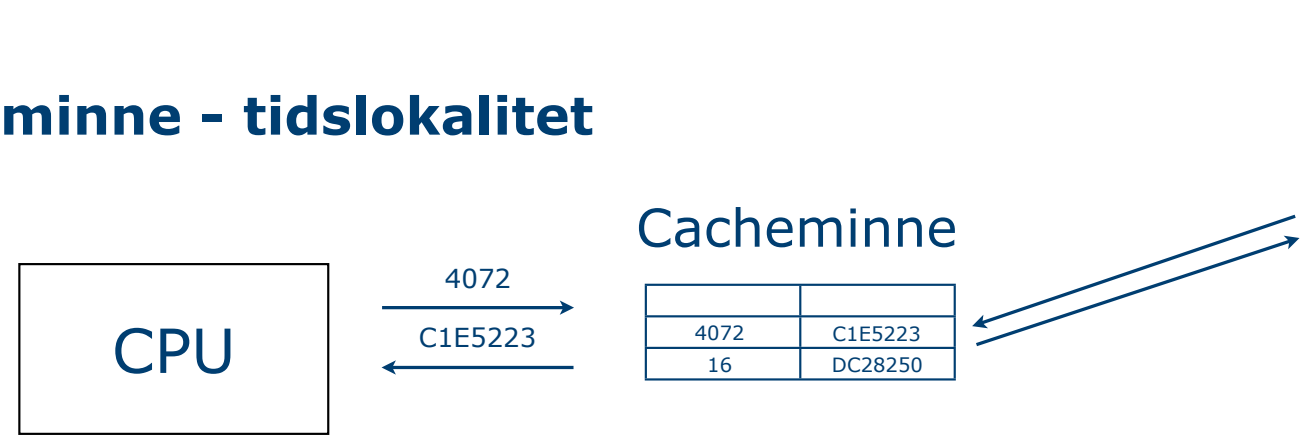
Stockholm University

Cacheminne - tidslokalitet



Stockholm University

Cacheminne - tidslokalitet



4096	176BBA33
4092	AB91C0A
4088	16209110
4084	CB5E7FD
4080	C3911AE
4076	6E48915
4072	C1E5223
4068	103E0960
4064	142A43DC
4060	2CAFB70
...	
56	954A3AC
48	12BA0777
40	B2C3729
32	9E43F06
24	D768C9
16	DC28250
12	135736AB
8	5E3456D
4	E8E95E5
0	829DF1A

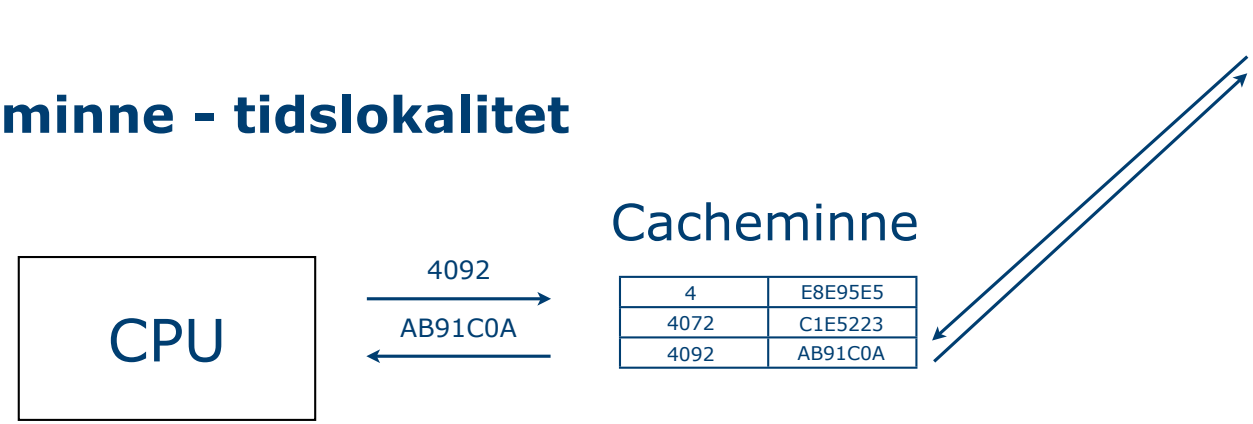
Stockholm University

Cacheminne - tidslokalitet

4096	176BBA33
4092	AB91C0A
4088	16209110
4084	CB5E7FD
4080	C3911AE
4076	6E48915
4072	C1E5223
4068	103E0960
4064	142A43DC
4060	2CAFB70
...	
36	954A3AC
32	12BA0777
28	B2C3729
24	9E43F06
20	D768C9
16	DC28250
12	135736AB
8	5E3456D
4	E8E95E5
0	829DF1A

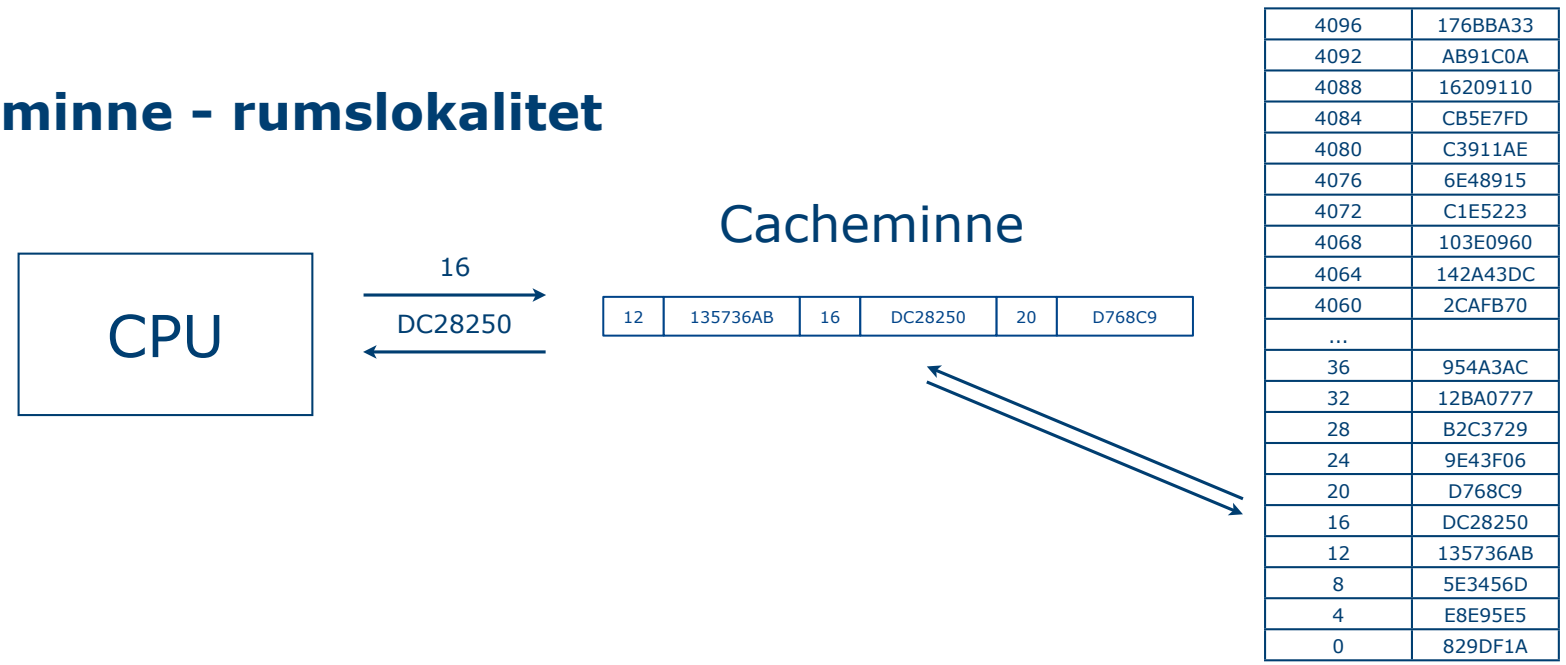
Stockholm University

Cacheminne - tidslokalitet



Stockholm University

Cacheminne - rumslokalitet



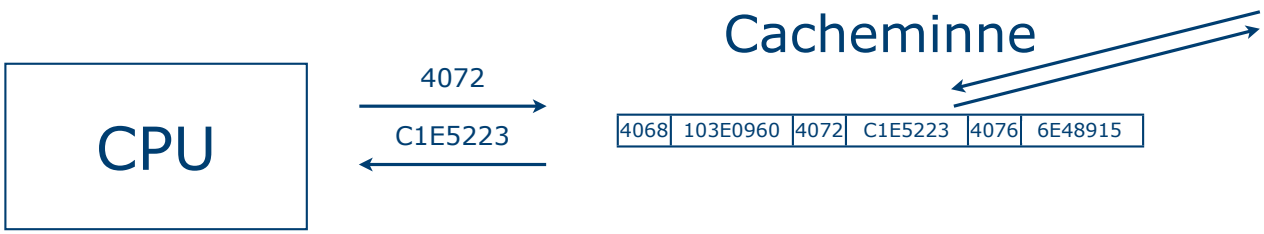
Stockholm University

Cacheminne - rumslokalitet

4096	176BBA33
4092	AB91C0A
4088	16209110
4084	CB5E7FD
4080	C3911AE
4076	6E48915
4072	C1E5223
4068	103E0960
4064	142A43DC
4060	2CAFB70
...	
36	954A3AC
32	12BA0777
28	B2C3729
24	9E43F06
20	D768C9
16	DC28250
12	135736AB
8	5E3456D
4	E8E95E5
0	829DF1A

Stockholm University

Cacheminne - rumslokalitet



4096	176BBA33
4092	AB91C0A
4088	16209110
4084	CB5E7FD
4080	C3911AE
4076	6E48915
4072	C1E5223
4068	103E0960
4064	142A43DC
4060	2CAFB70
...	
36	954A3AC
32	12BA0777
28	B2C3729
24	9E43F06
20	D768C9
16	DC28250
12	135736AB
8	5E3456D
4	E8E95E5
0	829DF1A

Stockholm University

Mjukvaru & Hårdvaru lösningar för Cache Coherence

Mjukvarulöningar

-Kompilatorn bedömer vilka minnesreferenser som kan bli riskabla och låter dem inte placeras i cachen

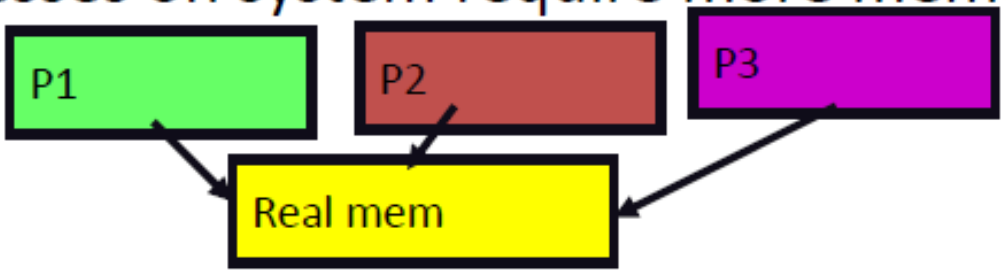
Hårdvarulösningar

-Directory

- Snooping

Thrashing: exposing the lie of VM

- Thrashing: processes on system require more memory than it has.



  - Each time one page is brought in, another page, whose contents will soon be referenced, is thrown out.
  - Processes will spend all of their time blocked, waiting for pages to be fetched from disk
  - I/O devs at 100% utilization but system not getting much useful work done

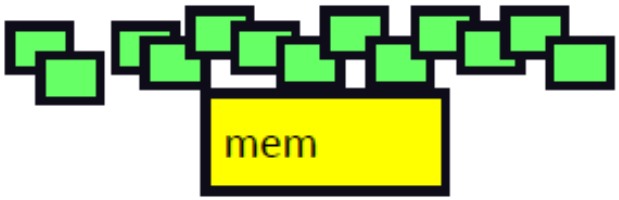
- What we wanted: virtual memory the size of disk with access time of of physical memory
- What we have: memory with access time = disk access

Thrashing

- Process(es) “frequently”reference page not in mem
  - Spend more time waiting for I/O then getting work done
- Three different reasons
  - process doesn’t reuse memory, so caching doesn’t work (past != future)

  - process does reuse memory, but it does not “fit”

  - individually, all processes fit and reuse memory, but too many for system.



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

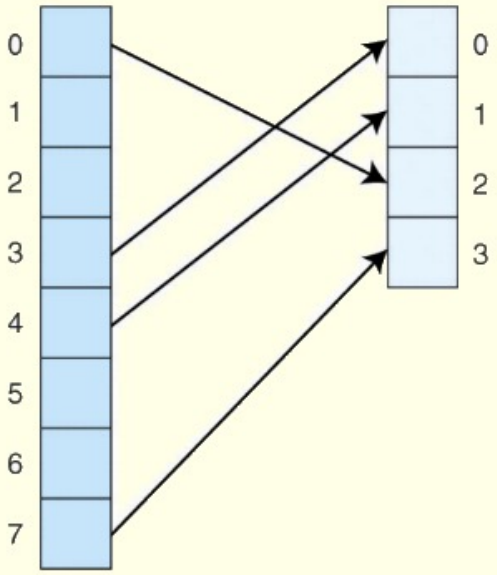
- A ***physical address*** is the actual memory address of physical memory.
- Programs **create *virtual addresses* -> *mapped* to physical addresses** by the memory manager.
- *Page faults* occur when a logical address requires that a page be brought in from disk.
- ***Memory fragmentation*** occurs 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.
- the pages allocated to a process do not need to be stored contiguously-- either on disk or in memory.
- Then 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.



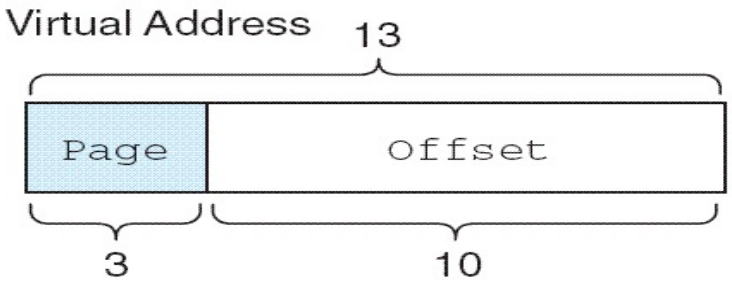
Page Table

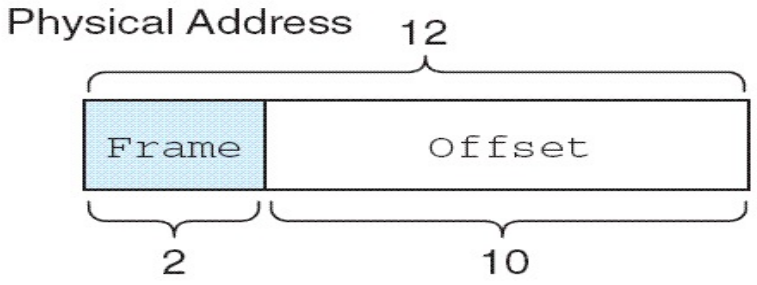
Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Stockholm University

Virtual Memory

- As an example, suppose a system has a **virtual address space of 8K** (8000 bytes = 2^{13}) and **a physical address space of 4K** (4000 bytes = 2^{12}), and the system uses byte addressing.
  - 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 (2^{10}).
- A physical memory address requires **12 bits**, the first two bits for the page frame and the trailing 10 bits the offset.





Stockholm University

Virtual Memory

- Suppose we have the page table shown below.
- What happens when CPU generates address $\mathbf{5459_{10}}$ = 1010101010011_2 = 1553_{16}?

Page Table

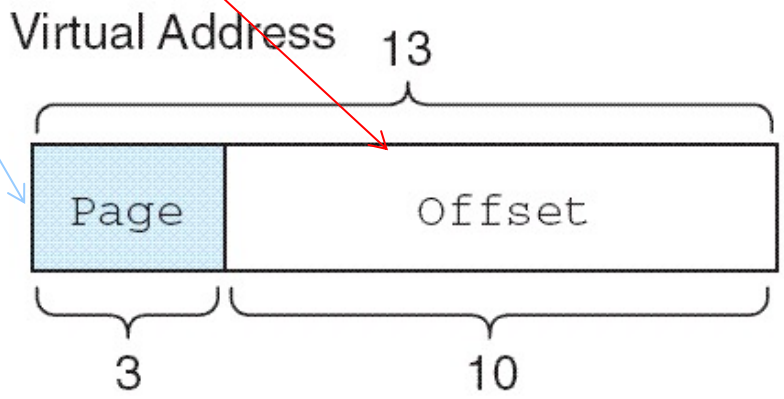
Page	Frame	Valid Bit
0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

Addresses

Page	Base 10	Base 16
0 :	0 - 1023	0 - 3FF
1 :	1024 - 2047	400 - 7FF
2 :	2048 - 3071	800 - BFF
3 :	3072 - 4095	C00 - FFF
4 :	4096 - 5119	1000 - 13FF
5 :	5120 - 6143	1400 - 17FF
6 :	6144 - 7167	1800 - 1BFF
7 :	7168 - 8191	1C00 - 1FFF

Virtual Memory

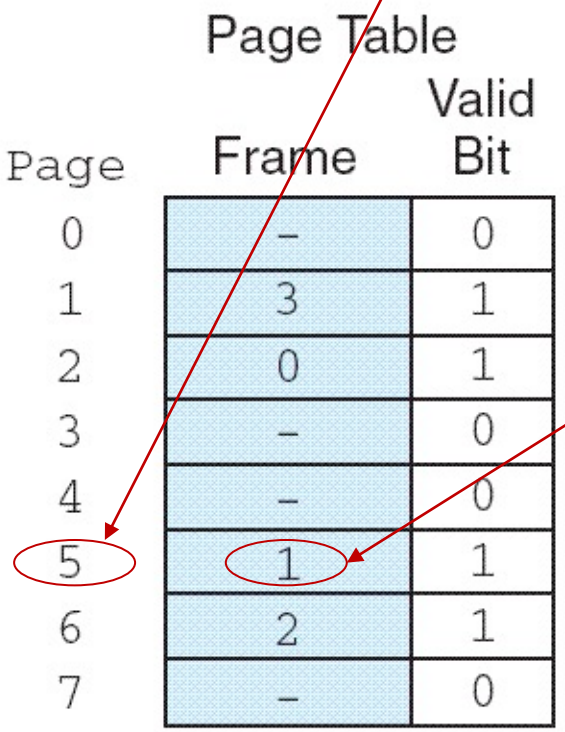
- What happens when CPU generates address 5459_{10} = 1010101010011_2 = 1553_{16}?

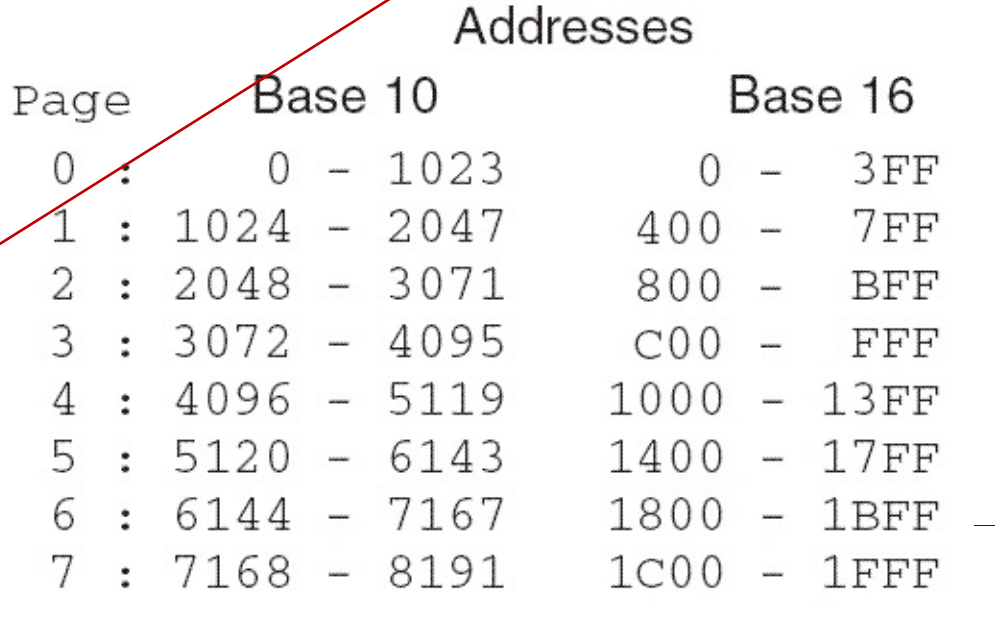


The high-order 3 bits of the virtual address, 101 (5_{10}), provide the page number in the page table.

Virtual Memory

- The address $101010101001 1_2$ is converted to physical address $010101010011_2 = 1363_{10}$ because the page field **101** is replaced by frame number **01** through a lookup in the page table.





Stockholm University

Virtual Memory

- What happens when the CPU generates address 1000000000100_2?

Page Table

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

Addresses

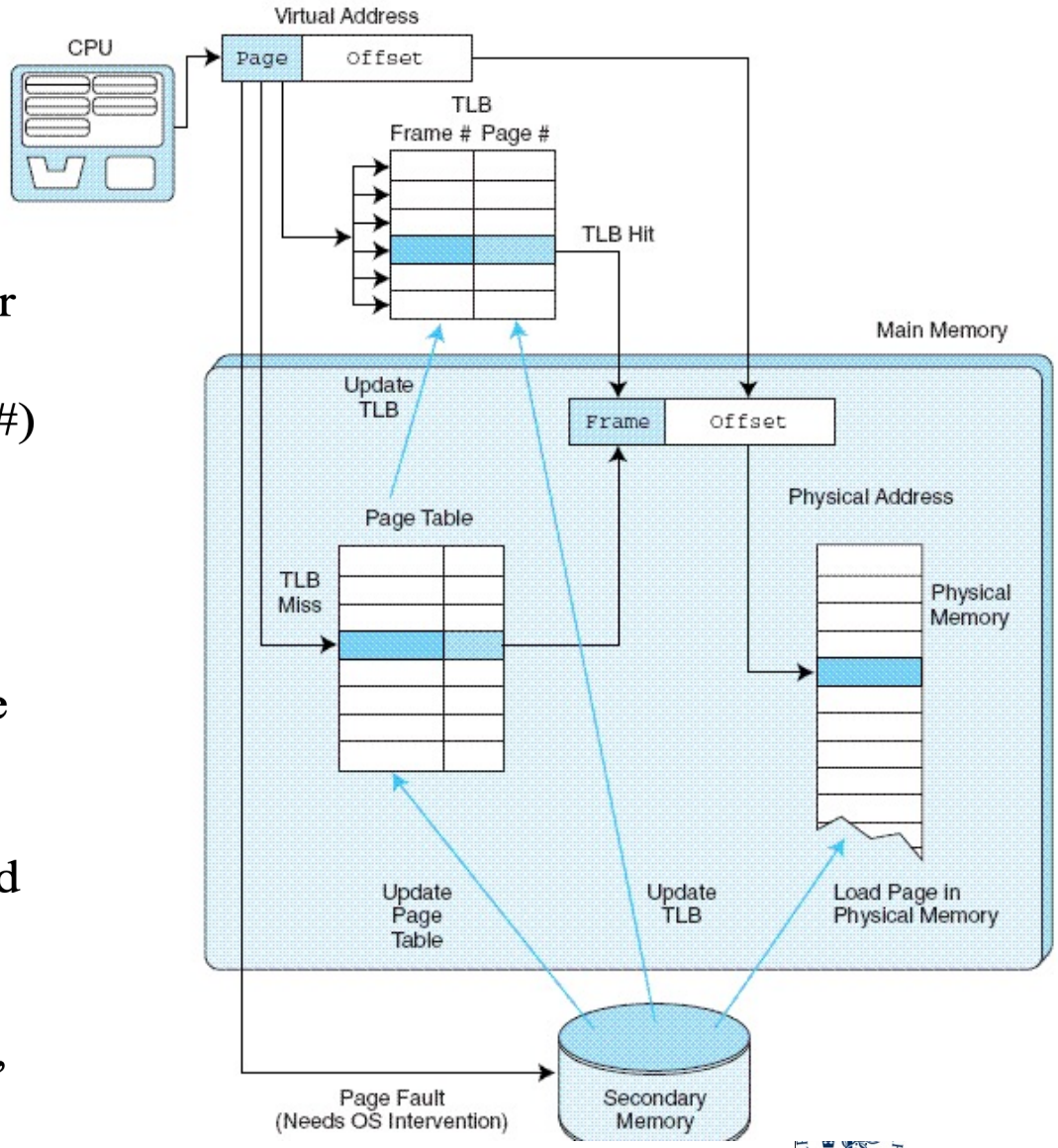
Page	Base 10	Base 16
0 :	0 - 1023	0 - 3FF
1 :	1024 - 2047	400 - 7FF
2 :	2048 - 3071	800 - BFF
3 :	3072 - 4095	C00 - FFF
4 :	4096 - 5119	1000 - 13FF
5 :	5120 - 6143	1400 - 17FF
6 :	6144 - 7167	1800 - 1BFF
7 :	7168 - 8191	1C00 - 1FFF

Translation look-aside buffer (TLB) lookup process

1. Extract the page number from the virtual address.
2. Extract the offset from the virtual address.
3. Search for the virtual page number in the TLB.
4. If the (virtual page #, page frame #) pair is found in the TLB, add the offset to the physical frame number and access the memory location.
5. If there is a TLB miss, go to the page table to get the necessary frame number.

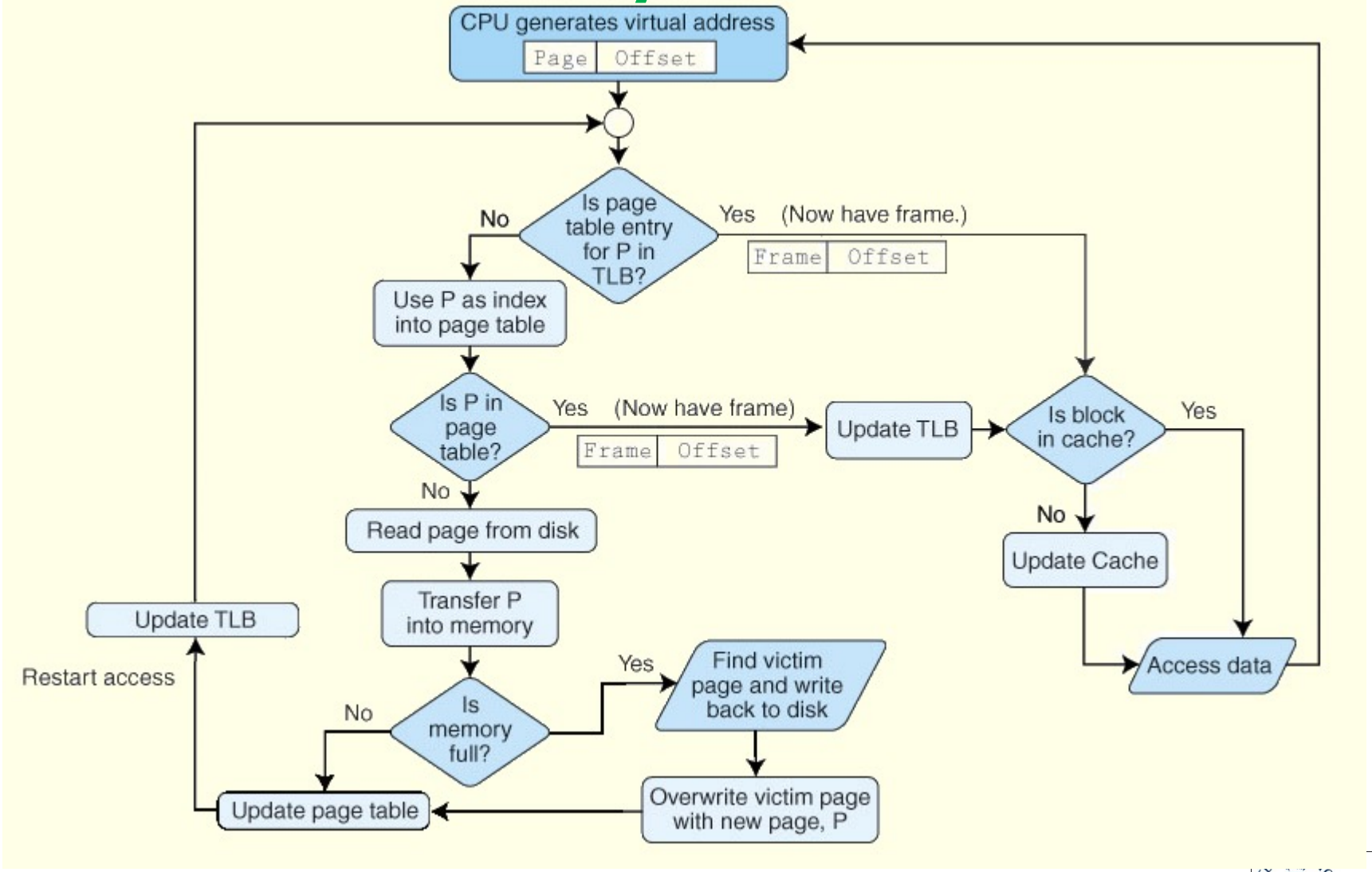
If the page is in memory, use the corresponding frame number and add the offset to yield the physical address.

6. If the page is not in main memory, generate a page fault and restart the access when the page fault is complete.



Stockholm University

Virtual Memory



Putting it all together: The TLB, Page Table, and Main Memory

Virtual Memory

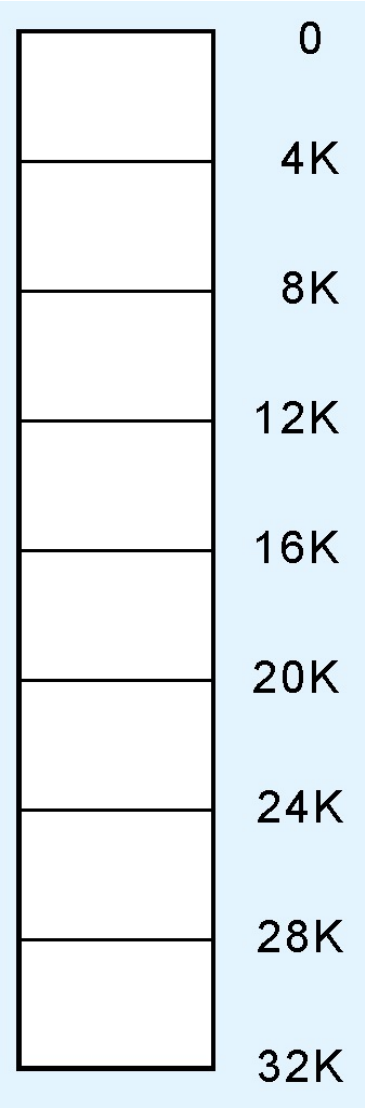
- Another approach to virtual memory is the use of ***segmentation***.
- Instead of dividing memory into equal-sized pages, virtual address space is divided into variable-length segments, often under the control of the **programmer**.
- A segment is located through its entry in a segment table, which 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 de-allocated over time.

Virtual Memory

- Consider a small computer having 32K of memory.
- The 32K memory is divided into 8 page frames of 4K each (32/8=4).
- A schematic of this configuration is shown at the right.
- The numbers at the right are memory frame addresses.



Stockholm University

Virtual Memory

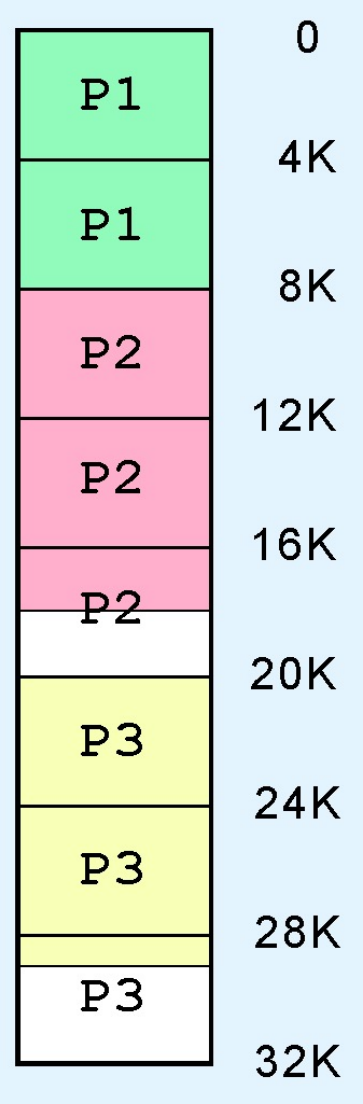
- Suppose there are four processes waiting to be loaded into the system with memory requirements as shown in the table.
- We observe that these processes require 31K of memory.

Process Name	Memory Needed
P1	8K
P2	10K
P3	9K
P4	4K

Virtual Memory

- When the first three processes are loaded, memory looks like this:
- All of the frames are occupied by three of the processes.

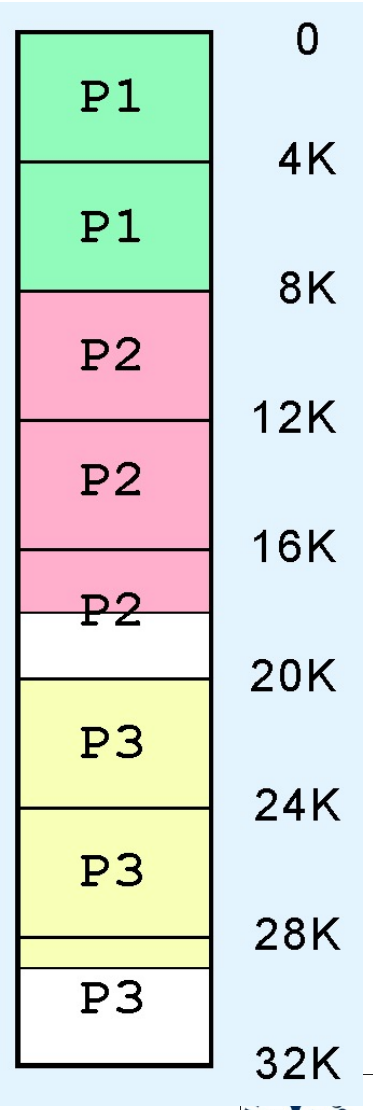
P1	8K
P2	10K
P3	9K
P4	4K



Virtual Memory

- Despite the fact that there are enough free bytes in memory to load the fourth process, P4 has to wait for one of the other three to terminate, because there are no unallocated frames.
- This is an example of ***internal fragmentation***.

P1	8K
P2	10K
P3	9K
P4	4K



Stockholm University

Virtual Memory

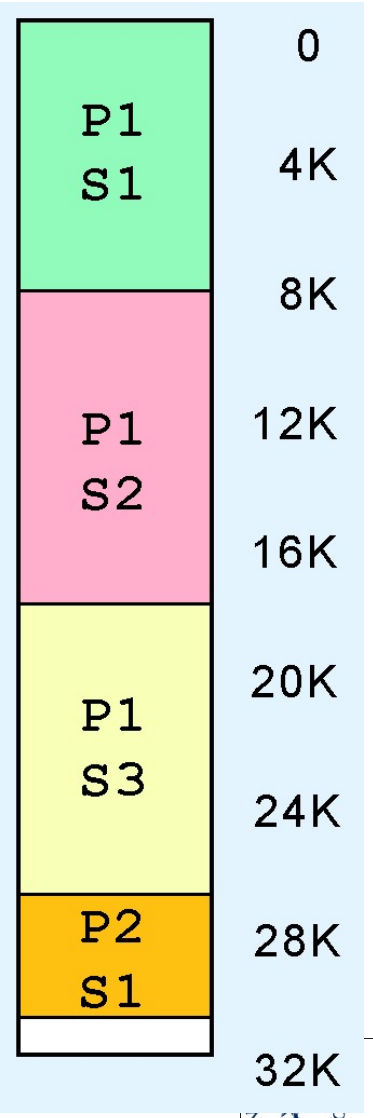
- Suppose that instead of frames, our 32K system uses segmentation.
- The memory segments of two processes is shown in the table at the right.
- The segments can be allocated anywhere in memory.

Process Name	Segment	Memory Needed
P1	S1	8K
	S2	10K
	S3	9K
P2	S1	4K
	S2	11K

Virtual Memory

- All of the segments of P1 and one of the segments of P2 are loaded as shown at the right.
- Segment S2 of process P2 requires 11K of memory, and there is only 1K free, so it waits.

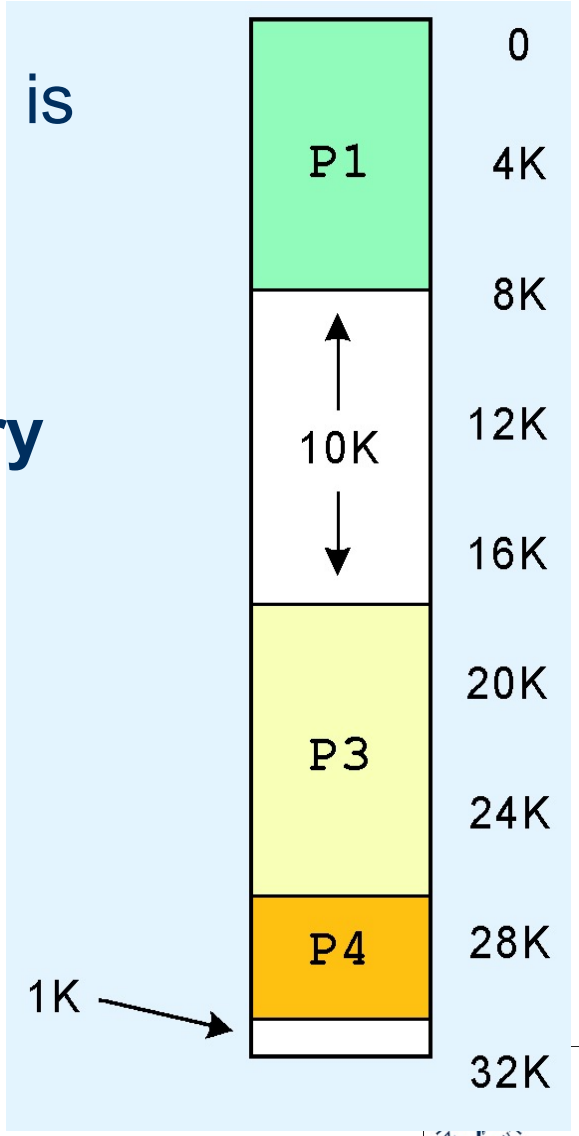
P1	S1	8K
	S2	10K
	S3	9K
P2	S1	4K
	S2	11K



Virtual Memory

- Eventually, Segment 2 of Process 1 is no longer needed, so it is unloaded giving 11K of free memory.
- But Segment 2 of Process 2 cannot be loaded because the **free memory** is **not** contiguous.

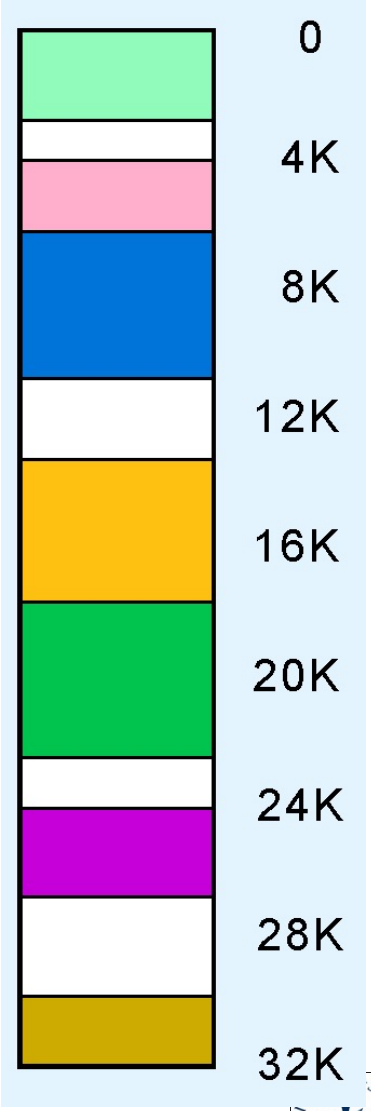
P1	S1	8K
	S2	10K
	S3	9K
P2	S1	4K
	S2	11K



Stockholm University

Virtual Memory

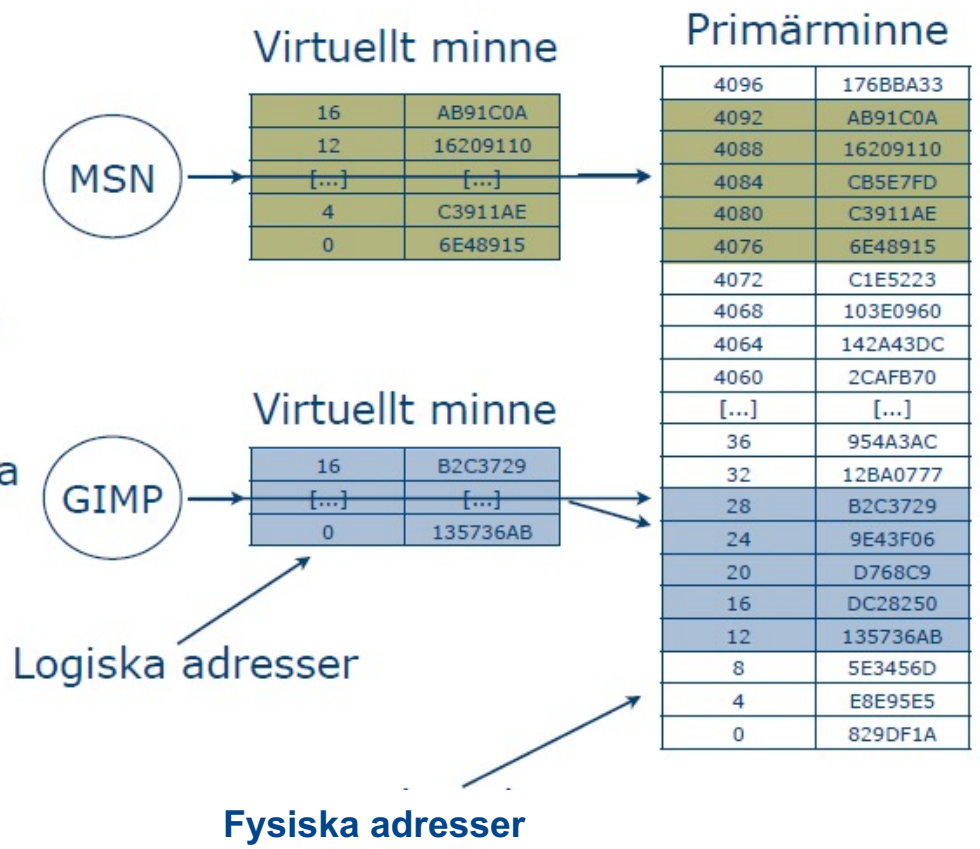
- Over time, the problem gets worse, resulting in small unusable blocks scattered throughout physical memory.
- This is an example of *external fragmentation*.
- Eventually, this memory is recovered through **compaction**, and the process starts over.



Stockholm University

Virtuellt minne

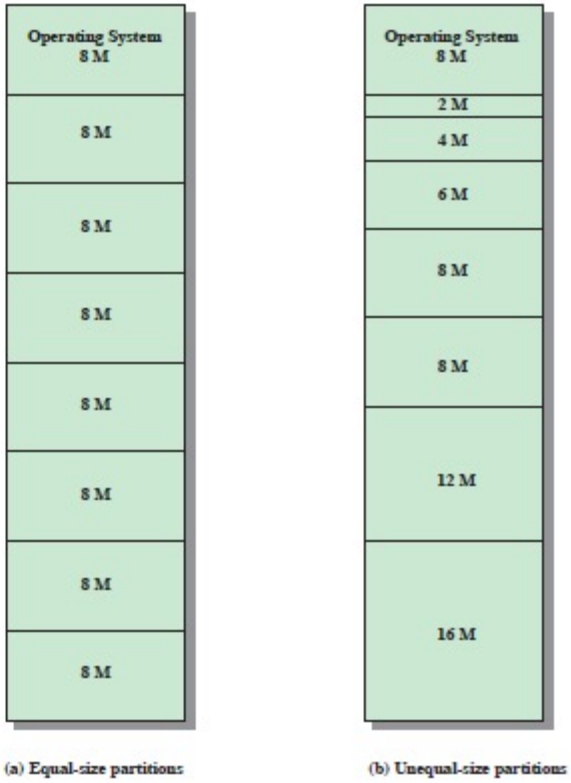
- En process kan swappas in var som helst i primärminnet. Processen måste referera relativt vart i minnet processen är inswappad.
- En fysisk adress beräknas genom att ta den logiska adressen plus processens *basadress*.



Stockholm University

Fixed-size partitioning

- Varje process tilldelas en minnespartition.
- Varje minnespartition har en förutbestämd storlek.
- Leder till att minnet utnyttjas suboptimalt.



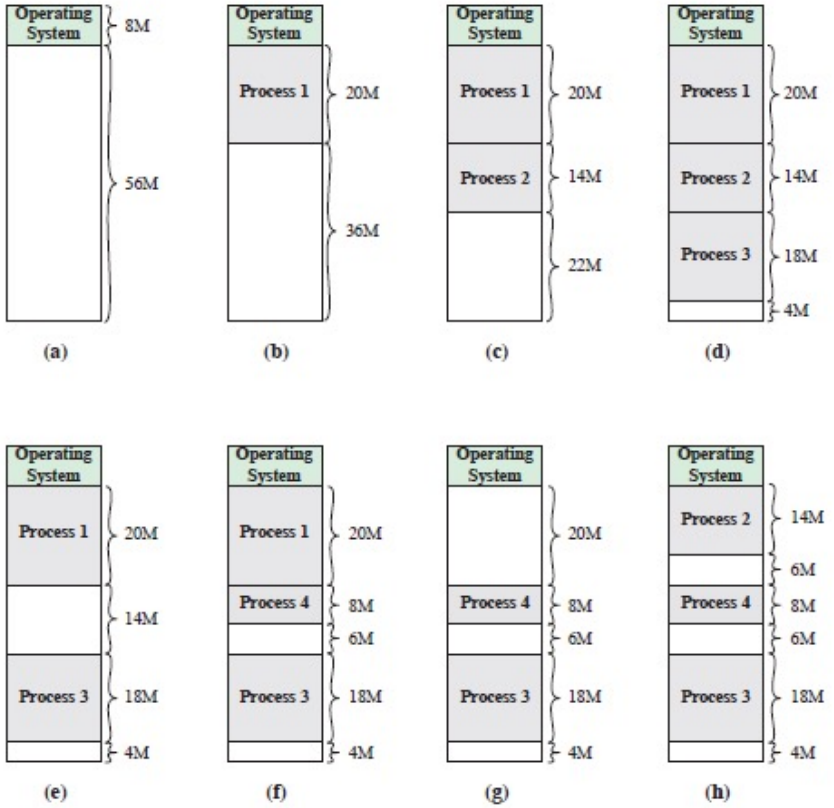
(a) Equal-size partitions

(b) Unequal-size partitions

Stockholm University

Variable size partitioning

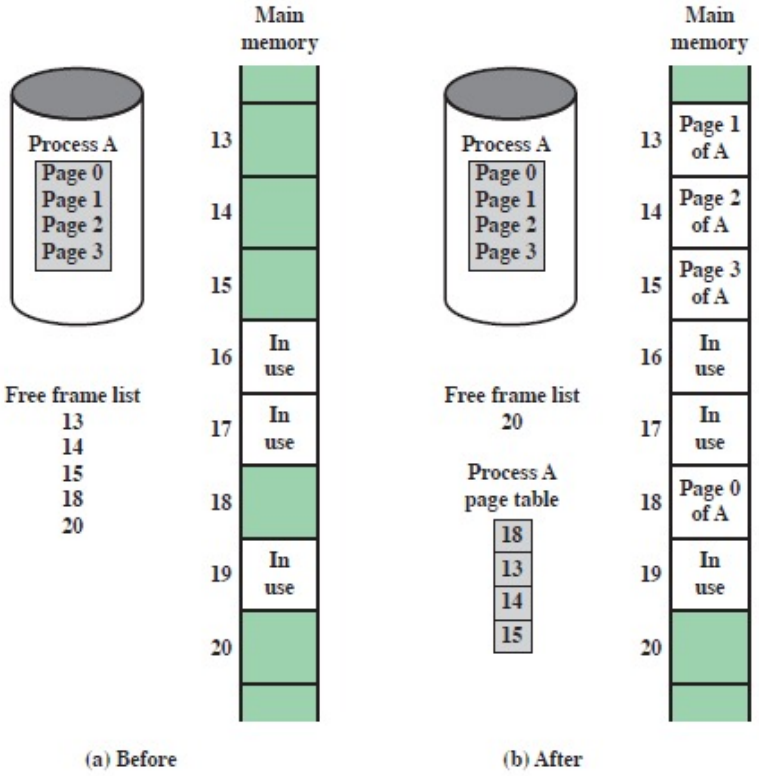
- Varje process tilldelas den mängd minne som processen behöver.
- Leder i längden till att minnet utnyttas suboptimalt, på grund av (extern) fragmentering.
- Ett fragmenterat minne kan jämnas ut av compaction.



Stockholm University

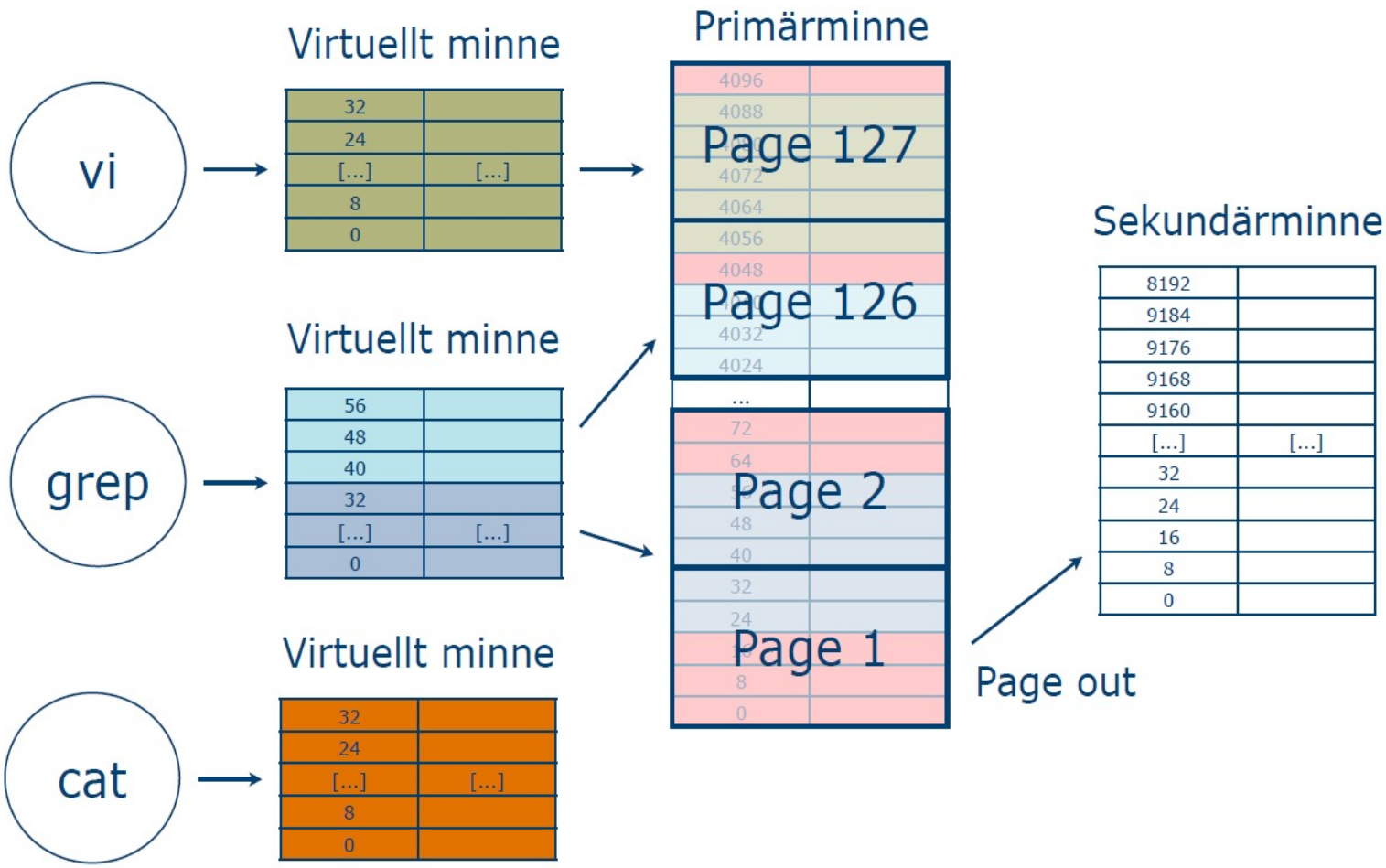
Paging

- Minnet delas in i lika stora delar, frames.
- Varje program delas upp i lika delar lika stora som framestorleken, kallade pages.
- Detta minimerar spillet till intern fragmentering.



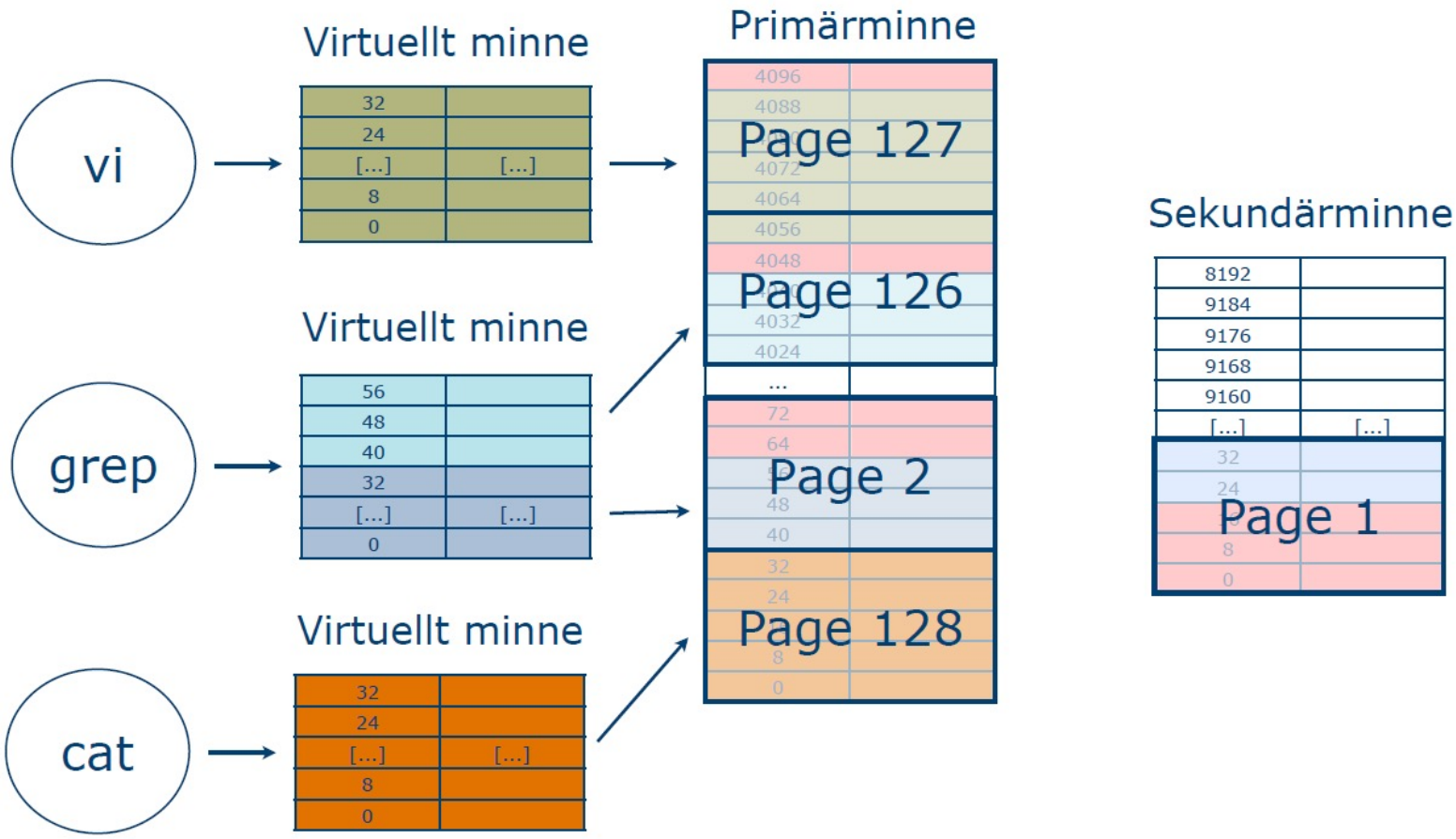
Stockholm University

Paging



Stockholm University

Paging



Stockholm University

Summary

- An overview of the:
  - Main memory, Cache memory & Virtual memory
- Types of Memory: ROM, RAM, SRAM, DRAM
- Cache Memory:
  - Direct mapped cache
  - Set Associative cache mapping
  - Replacement policy
- Virtual Memory
  - virtual address
  - TLB
  - Paging
  - Segmentation