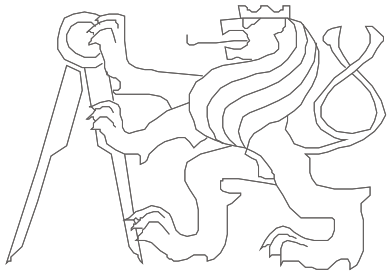
Computer Architectures

Fast and/or Large Memory – Cache and Memory Hierarchy

Pavel Píša, Richard Šusta, Petr Štěpán
Michal Štepanovský, Miroslav Šnorek

Main source of inspiration: Patterson



Czech Technical University in Prague, Faculty of Electrical Engineering

English version partially supported by:
European Social Fund Prague & EU: We invests in your future.

Picture of the Day – Sareni Pasovi Durmitor

Lecture Motivation

Quick Quiz 1.: Is the result of both code fragments a same?

Quick Quiz 2.: Which of the code fragments is processed faster and why?

A:

```
int matrix[M][N];
int i, j, sum = 0;
…
for(i=0; i<M; i++)
  for(j=0; j<N; j++)
    sum += matrix[i][j];
```

B:

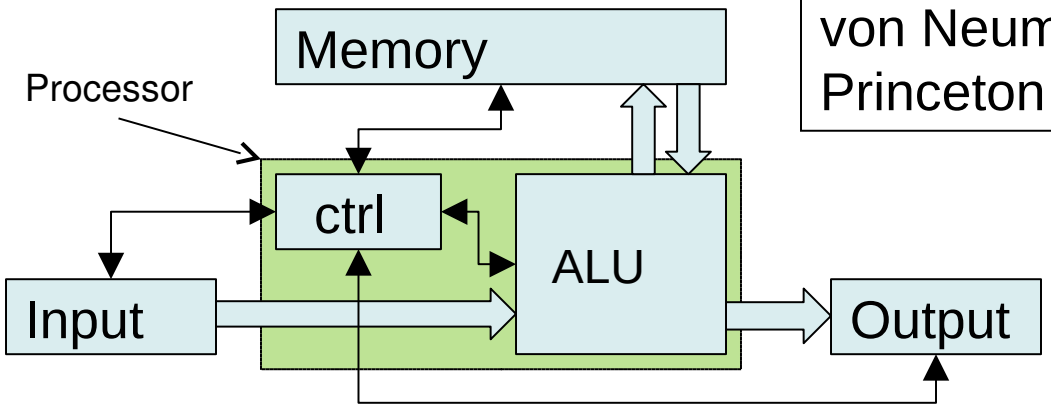
```
int matrix[M][N];
int i, j, sum = 0;
…
for(j=0; j<N; j++)
  for(i=0; i<M; i++)
    sum += matrix[i][j];
```

Is there a rule how to iterate over matrix element efficiently?

Lecture Outline

- Overview of memory related terms and definitions
- Memory hierarchy
  - Management and mapping of data between levels
- Cache memory
  - Basic concept
  - More realistic approach
  - Multi-level cache memory
- Virtual memory
- Memory hierarchy and related problems
- Secondary(+more) storage (mass storage)

John von Neumann, Hungarian physicist



von Neumann's computer architecture
Princeton Institute for Advanced Studies

28. 12. 1903 -
8. 2. 1957

- 5 functional units – control unit, arithmetic logic unit, memory, input (devices), output (devices)
- An computer architecture should be independent of solved problems. It has to provide mechanism to load program into memory. The program controls what the computer does with data, which problem it solves.
- Programs and results/data are stored in the same memory. That memory consists of a cells of same size and these cells are sequentially numbered (address).
- The instruction which should be executed next, is stored in the cell exactly after the cell where preceding instruction is stored (exceptions branching etc.).
- The instruction set consists of arithmetics, logic, data movement, jump/branch and special/control instructions.

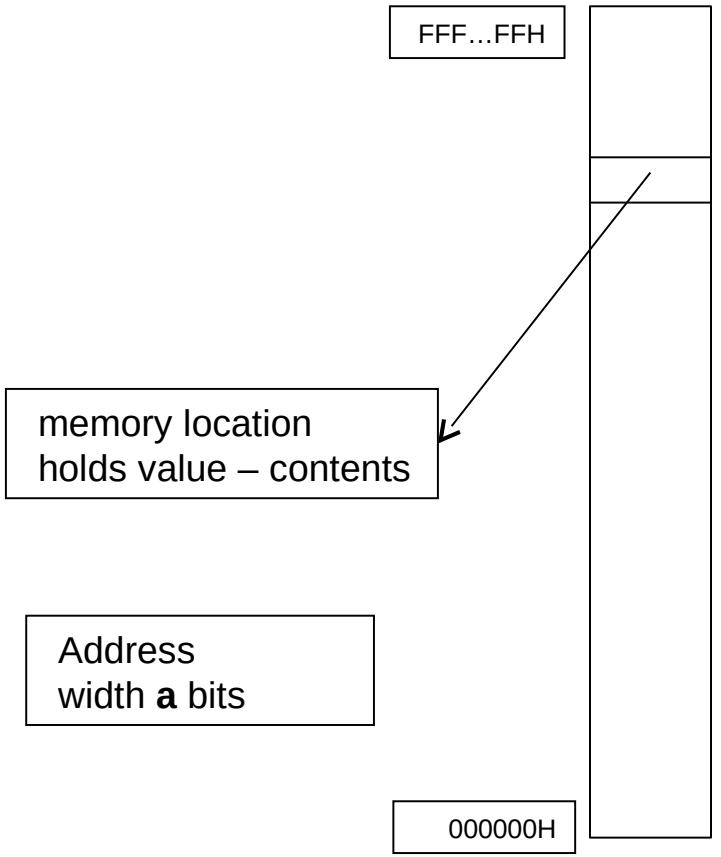
What is Memory?

Memory
- an array of addressable units
  - each unit can hold a data value - byte.
  - maximum size depends of address size

RAM
- Random Access Memory
  - random means any address, not only sequential access

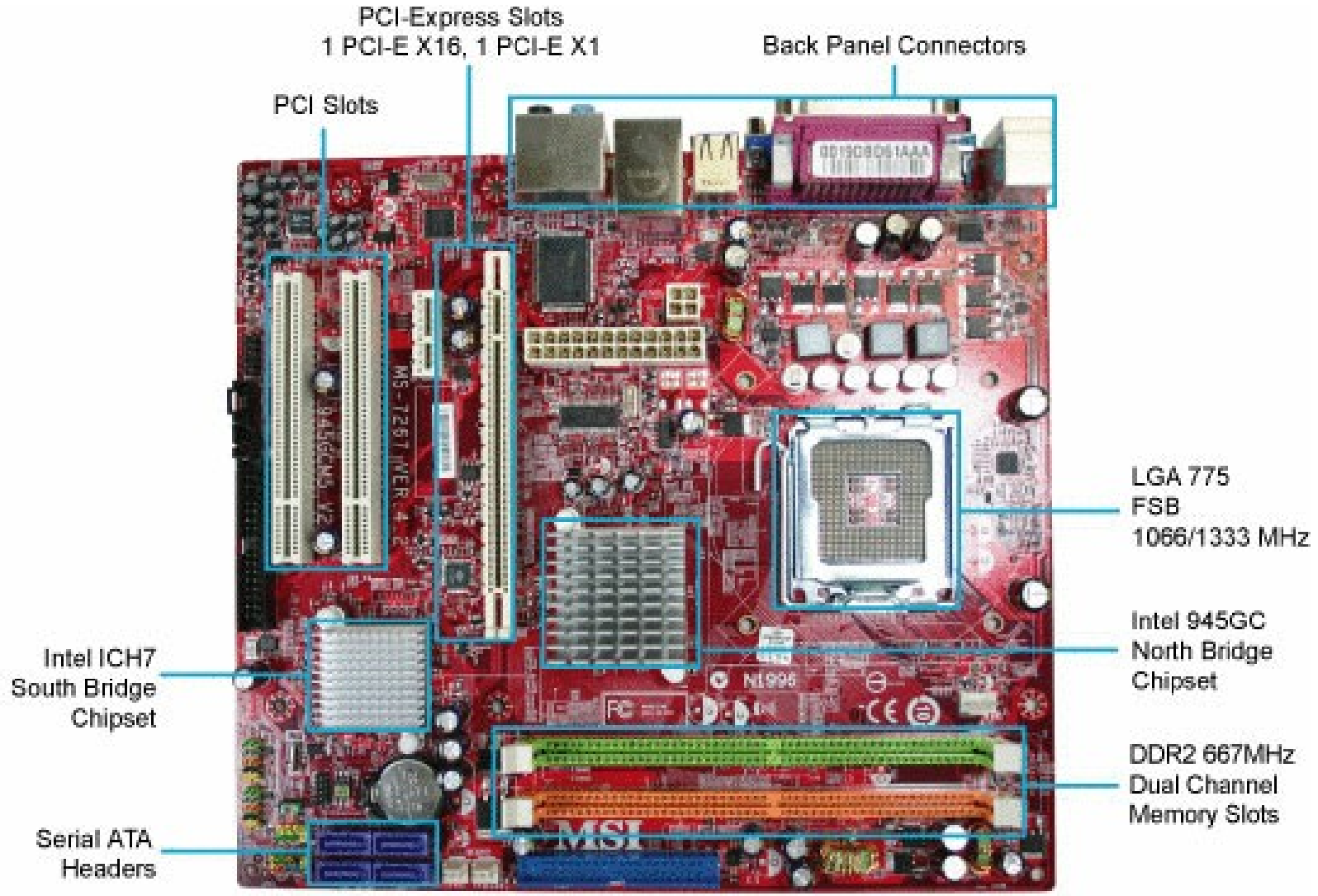
unsigned char RAM[2^a];

a	2↑a
8	256 distinct locations
16	64K (K=1024)
…	……
32	4G (4096M, M=K↑2)



The most common size of addressable memory unit is 1B (8 bits)

PC Computer Motherboard



http://www.pcplanetsystems.com/abc/product_details.php?

Memory Types and Maintenance

- Types: RAM (RWM), ROM, FLASH
- Implementation: SRAM, DRAM

- Data retention time and conditions (volatile/nonvolatile)
- Dynamic memories (DRAM, SDRAM) require specific care
  - Memory refresh – state of each memory cell has to be internally read, amplified and fed back to the cell once every refresh period (usually about 60 ms), even in idle state. Each refresh cycle processes one row of cells.
  - Precharge – necessary phase of access cycle to restore cell state after its partial discharge by read
  - Both contribute to maximal and average access time.

Typical SRAM and DRAM Memory Parameters

Memory kind	Number of transistors	1 bit area on silicon	Data availability	latency
SRAM	about 6	< 0,1 μm^2	instantenous	< 1ns – 5ns
DRAM	1	< 0,001 μm^2	needs refresh usually 64ms	>ten ns

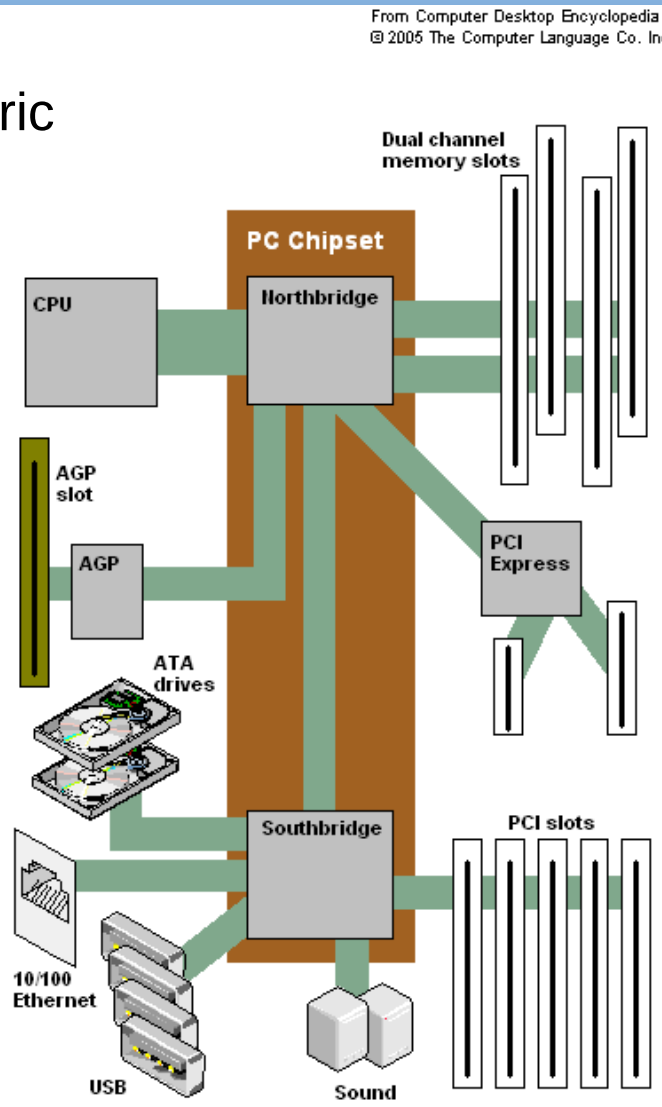
DRAM memory with external clock is called SDRAM Synchronous DRAM

DDR memory is type of SDRAM

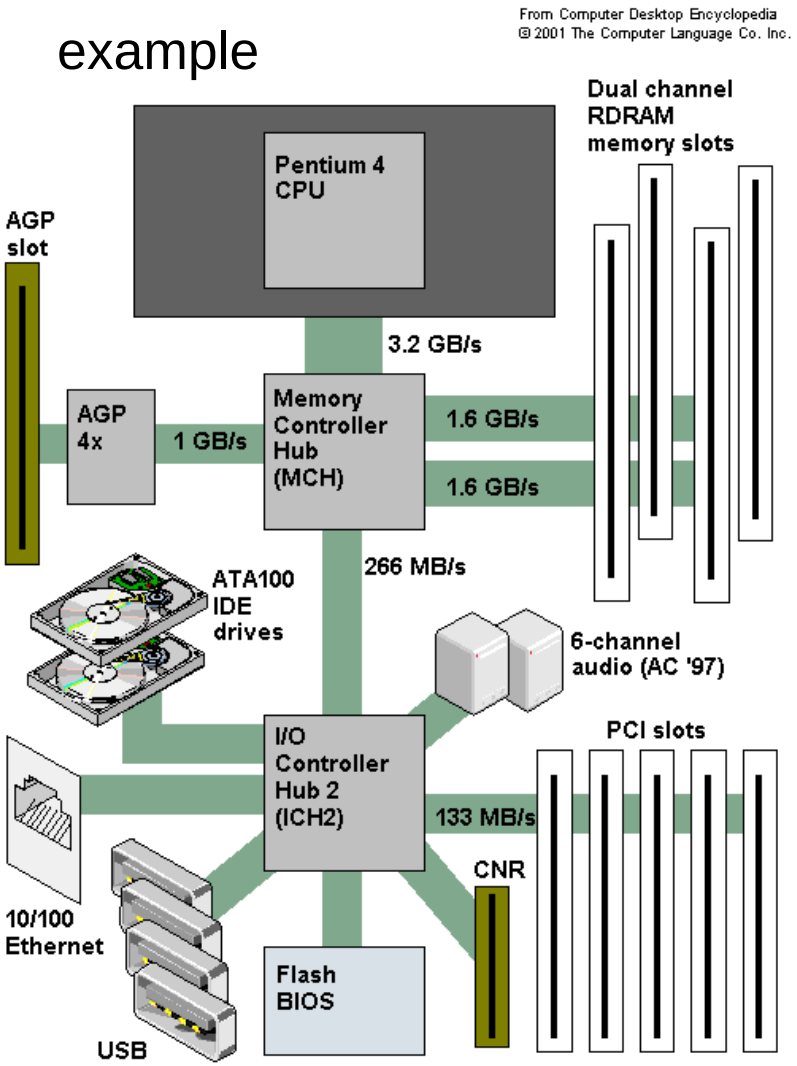
DDR5 throughput 50-75GB/s
latency 25-35ns

Computer architecture (desktop x86 PC)

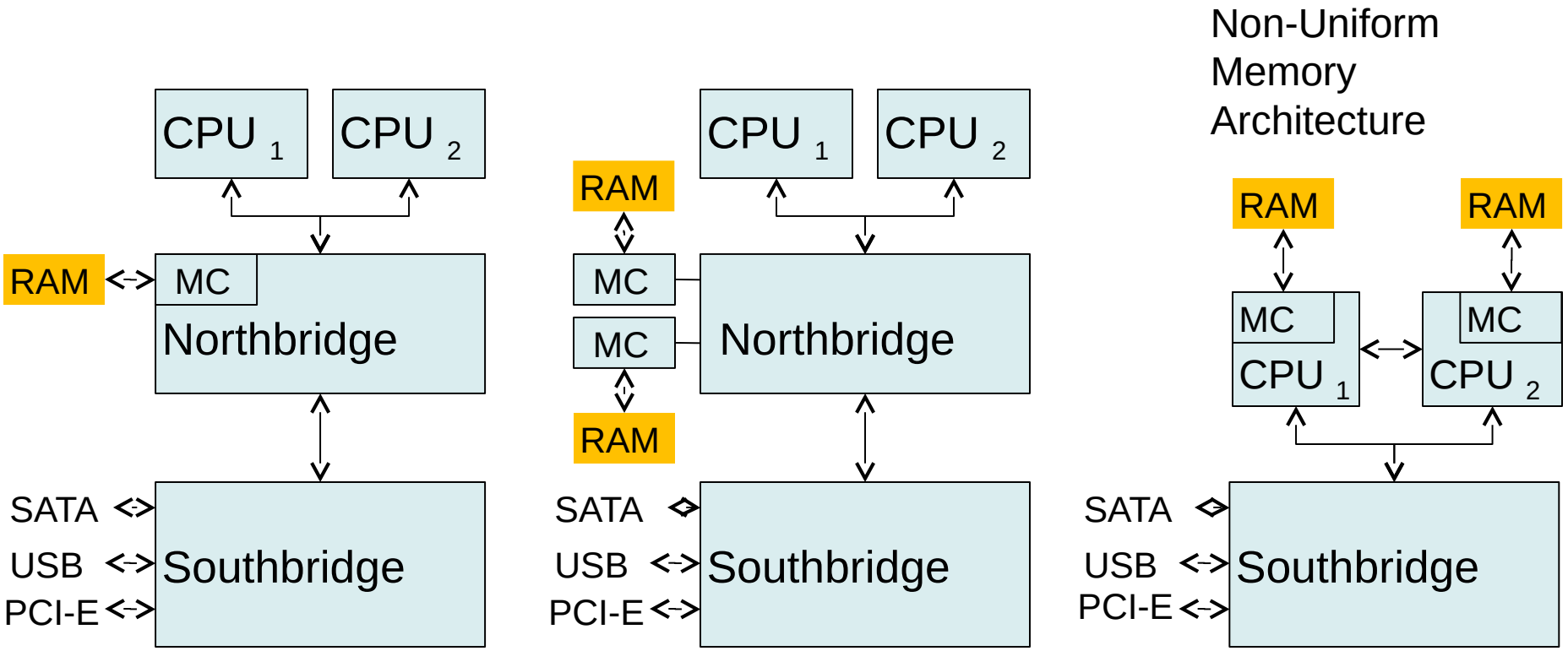
generic



example

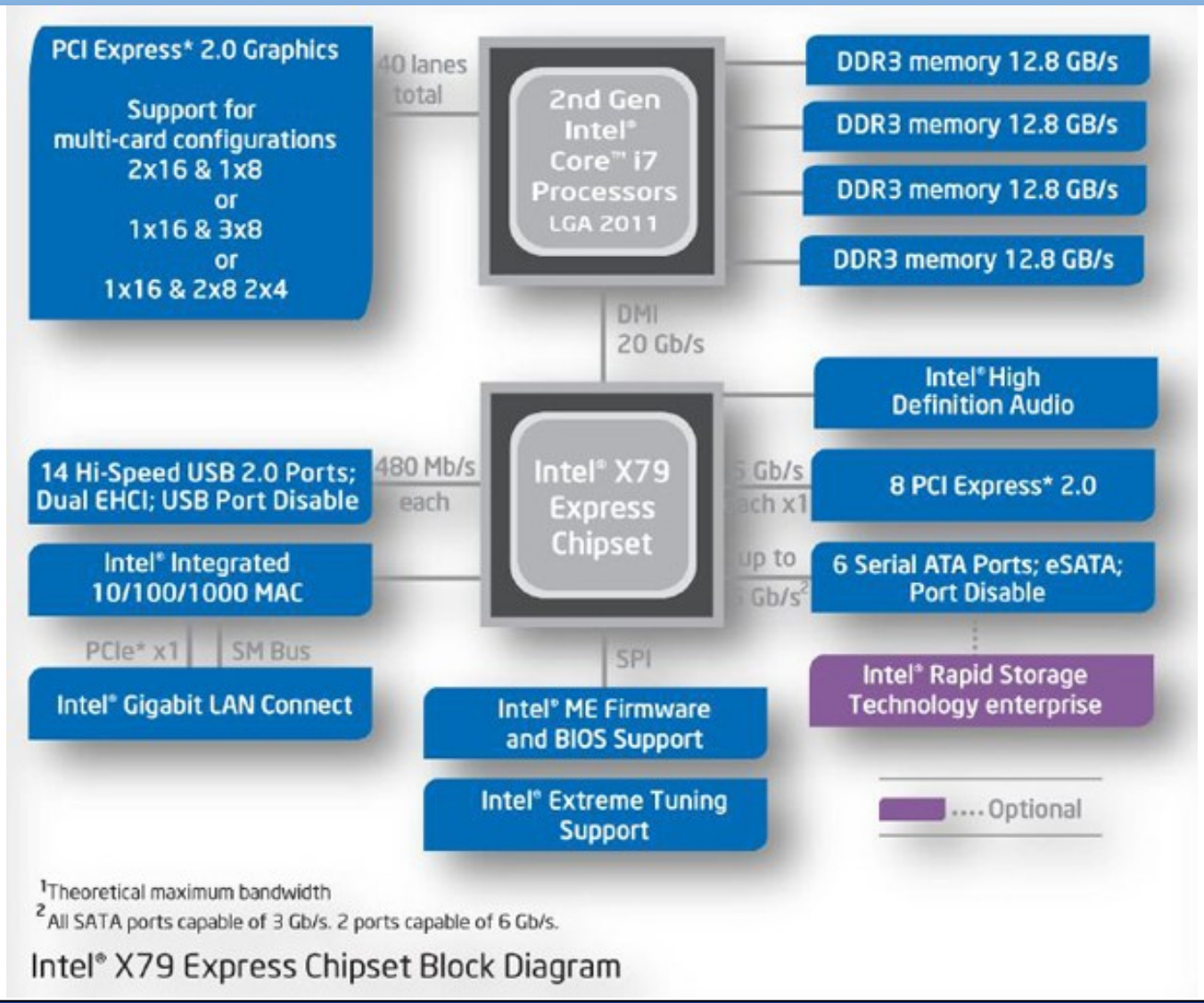


From UMA to NUMA development (even in PC segment)



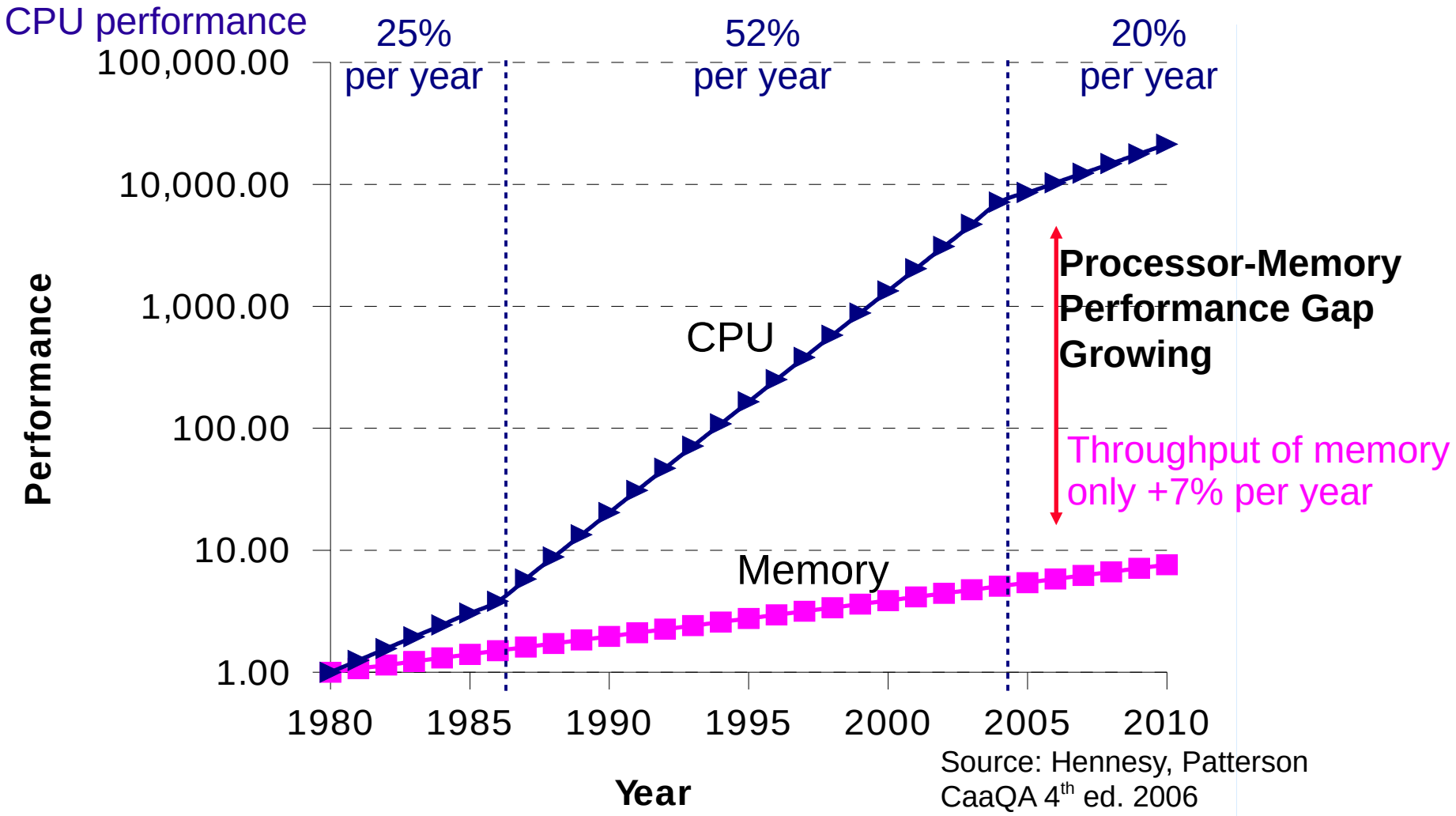
MC - Memory controller – contains circuitry responsible for SDRAM read and writes. It also takes care of refreshing each memory cell every 64 ms.

Intel i3/5/7 generation



Intel® X79 Express Chipset Block Diagram

Memory and CPU speed – Moore's law



Memory Subsystem – Terms and Definitions

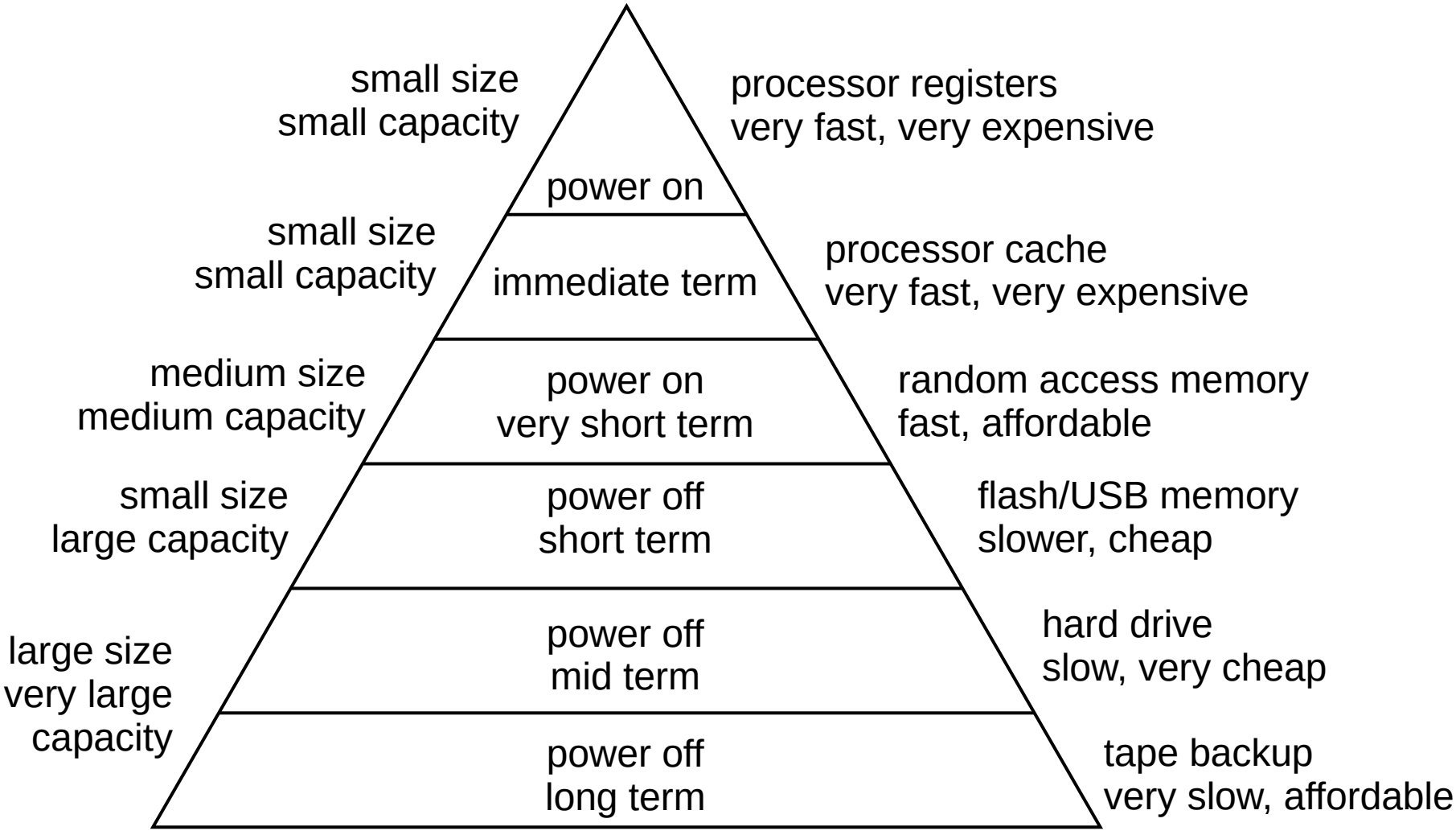
- Memory address – fixed-length sequences of bits or index
- Data value – the visible content of a memory location

  Memory location can hold even more control/bookkeeping information
  - validity flag, parity and ECC bits etc.

- Basic memory parameters:
  - Access time – delay or latency between a request and the access being completed or the requested data returned
  - Memory latency – time between request and data being available (does not include time required for refresh and deactivation)
  - Throughput/bandwidth – main performance indicator. Rate of transferred data units per time.
  - Maximal, average and other latency parameters

Processor-memory Performance Gap Solution – Cache

Memory Hierarchy – Speed, Capacity, Price



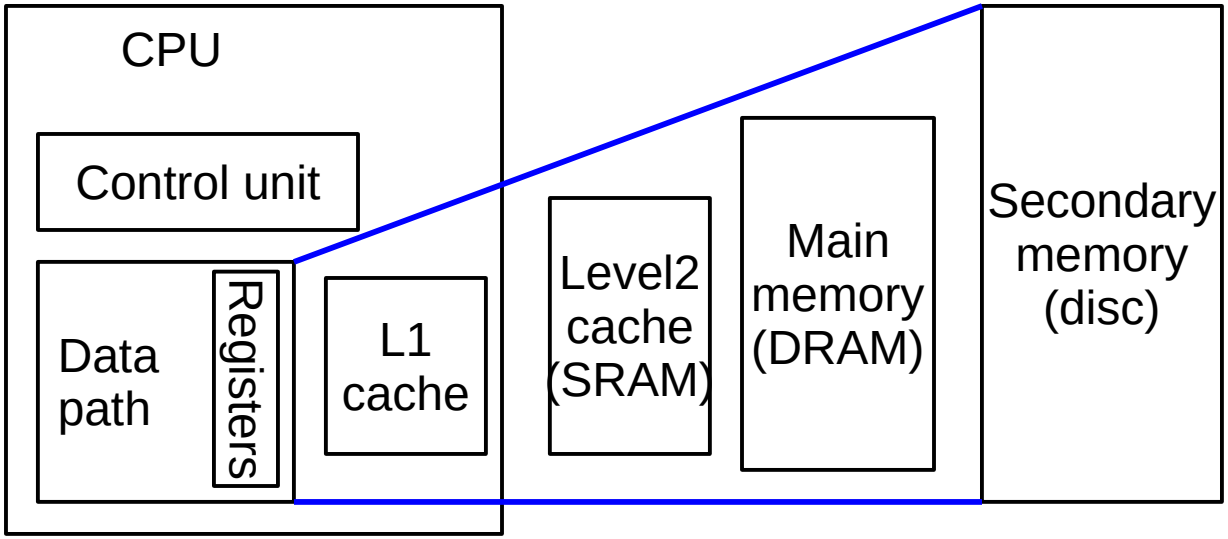
Source: Wikipedia.org

Performance Gap Between CPU and Main Memory

- Solution – **cache** memory
- Cache – component that (transparently) stores data so that future requests for that data can be served faster
- Transparent cache – hidden memory

- Placed between two subsystems with different data throughput. It speeds-up access to (recently) used data.
- This is achieved by maintaining copy of data on memory device faster than the original storage

Contemporary Price/Size Examples



Type/ Size	L1 32kB	Sync SRAM	DDR3 16 GB	HDD 3TB
Price	10 kč/kB	300 kč/MB	123 kč/GB	1 kč/GB
Speed/ throughput	0.2...2ns	0.5...8 ns/word	15 GB/sec	100 MB/sec

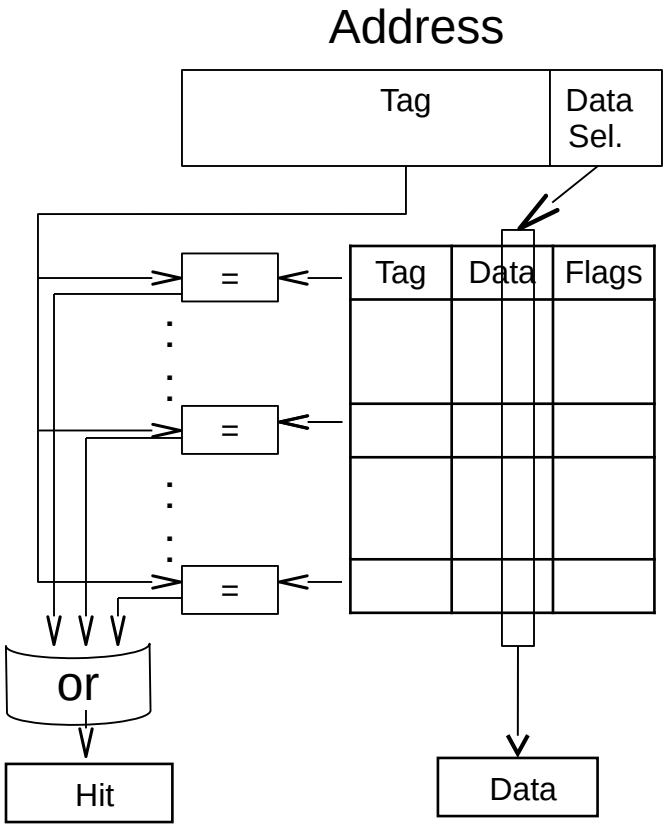
Some data can be available in more copies (consider levels and/or SMP).
Mechanisms to keep consistency required if data are modified.

Mechanism to Lookup Demanded Information?

- According to the address and other management information (data validity flags etc).
- The lookup starts at the most closely located memory level (local CPU L1 cache).
- Requirements:
  - Memory consistency/coherency.
- Used means:
  - Memory management unit to translate virtual address to physical and signal missing data on given level.
  - Mechanisms to free (swap) memory locations and migrate data between hierarchy levels
- Hit (data located in upper level – fast), miss (copy from lower level required)

Initial Idea – Fully Associative Cache

- **Tag** – the key to locate data (value) in the cache. The original address in the main memory for fully associative case.
- **Data** – the stored information, basic unit – word – is usually 4 bytes, can be more words – power of 2
- **Flags** – additional bits to keep service information.

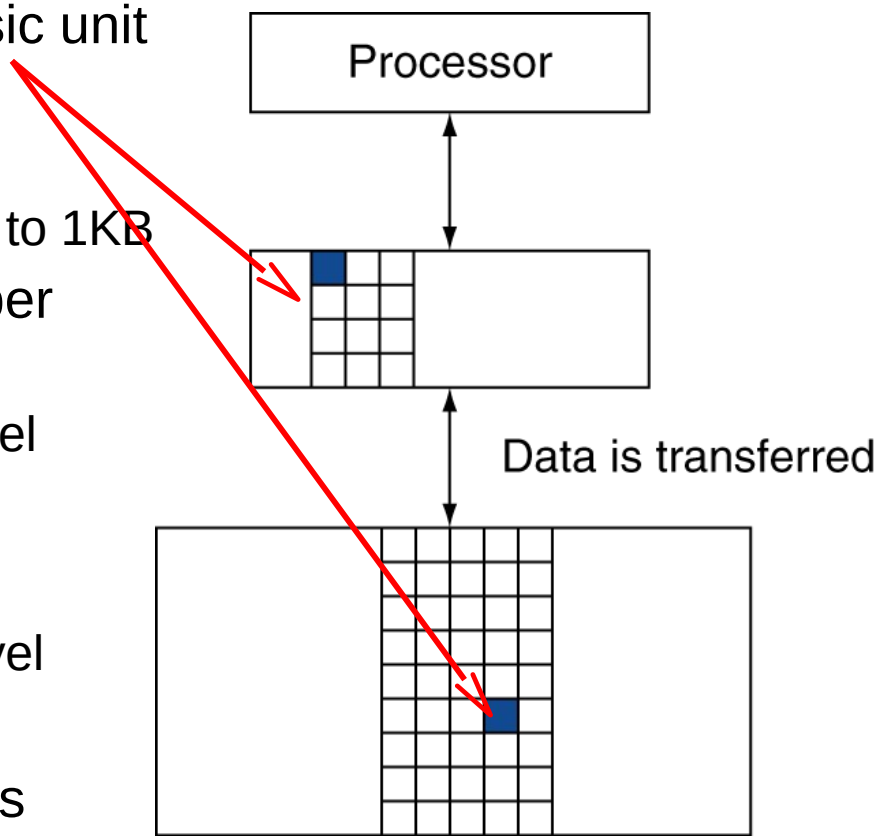


Cache line of fully associative cache

Tag	Data	Flags

Definitions for Cache Memory

- **Cache line** or **cache block** – basic unit copied between levels
  - May be multiple words
  - Usual cache line size from 8B up to 1KB
- If accessed data is present in upper level
  - **Hit**: access satisfied by upper level
    - **Hit rate**: hits/accesses
- If accessed data is absent
  - **Miss**: block copied from lower level
    - Time taken: **miss penalty**
    - **Miss rate**: misses/accesses = 1 – hit rate
  - Then the accessed data is supplied from upper level



Important Cache Access Statistical Parameters

- **Hit Rate** – number of memory accesses satisfied by given level of cache divided by number of all memory accesses
- **Miss Rate** – same, but for requests resulting in access to slower memory = 1 – Hit Rate
- **Miss Penalty** – time required to transfer block (data) from lower/slower memory level
- Average Memory Access Time (AMAT)

$$AMAT = Hit\ Time + Miss\ Rate \times Miss\ Penalty$$

- Miss Penalty for multi-level cache can be computed by recursive application of AMAT formula

Average Memory Access Time (AMAT) Example

$AMAT = HitTime_{L1} + MissRate_{L1} * MissPenalty_{L1}$

L1 access time: 1 cycle

Memory access time: 8 cycles

Program behavior: 2% miss rate

AMAT with cache: 1 + (0.02 * 8) = 1.16

What is the time access without cache?

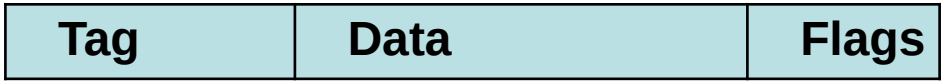
A) 1 cycle

B) 2 cycles

C) 4 cycles

D) 8 cycles

How big is Tag?



- The **Tag** field width depends on address width
- **Tag** must identify, which data are in Data section

Example:

- Data size is 16 bytes = 2^4
- Address size is 64 bits

How many bits has the **Tag?**

A) 64 bits

B) 60 bits

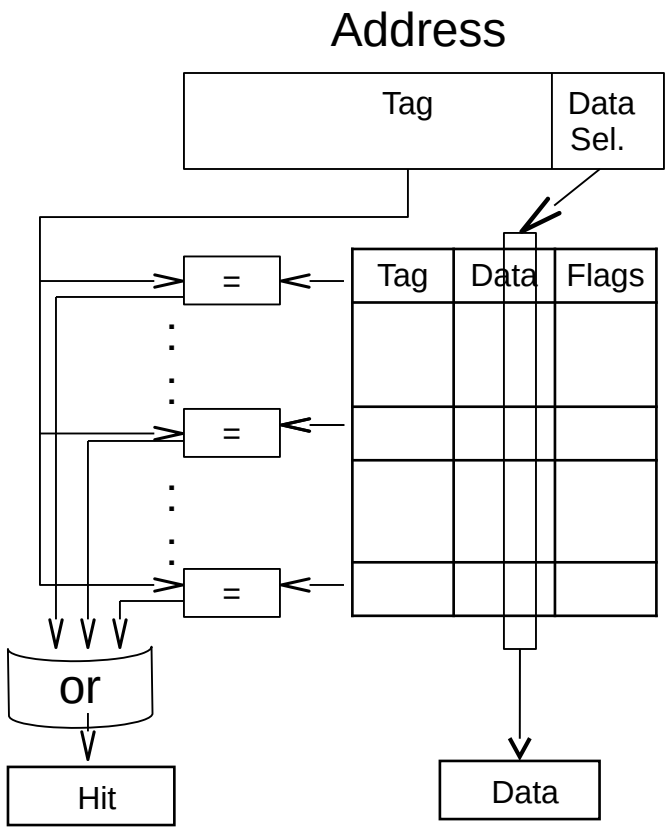
C) 32 bits

D) 4 bits

Fully Associative Cache Implementation

- Each cache line requires its own multiplexer input and same number of one-bit comparators as is size of the tag field.
- Cache line count determines capacity of the cache
- Cache requires complex replacement policy logic to find out which of all lines is the best candidate for new request.

- Such cache implementation is very expensive to implement in HW (ratio of gate count/capacity is high)
- That is why other cache types are used in practice
  - Direct mapped
  - n-way associative – with limited associativity



Direct Mapped Cache

Cache parameters:

Capacity – C

Number of sets – S

Block size – b

Number of blocks – B

Degree of associativity – N

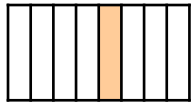
C = 8 (8 words),

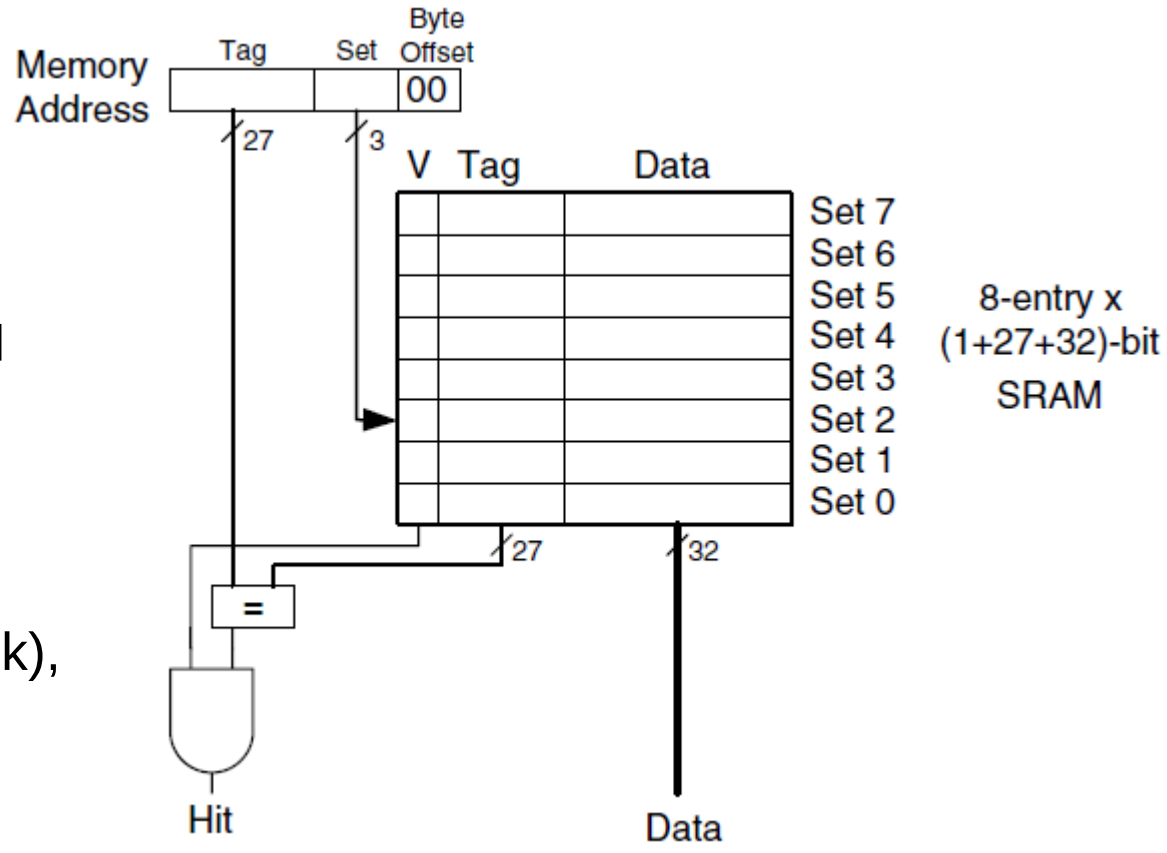
S = B = 8,

b = 1 (one word in the block),

N = 1

direct mapped cache: one block in each set

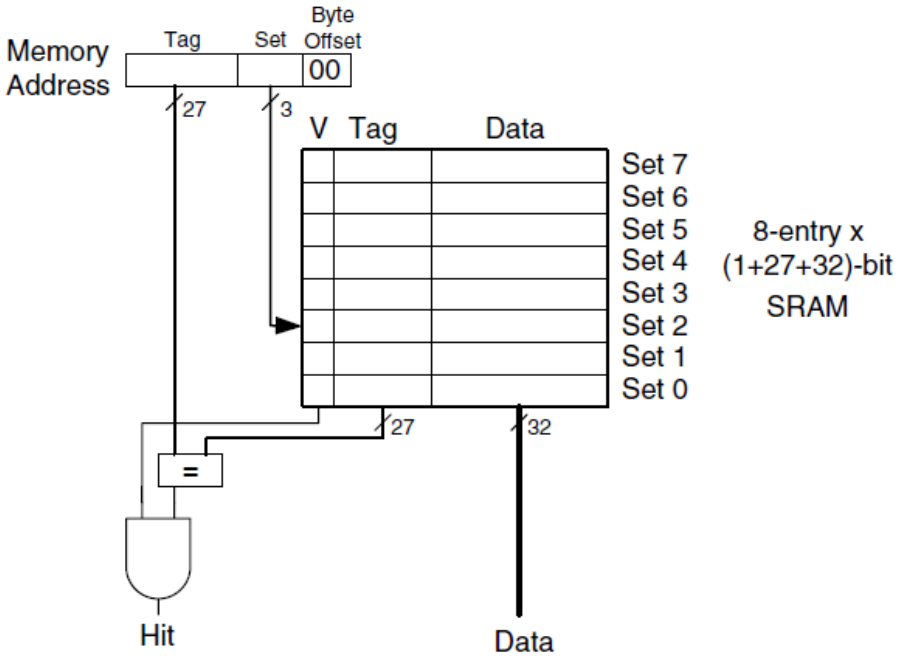




How big is Tag for direct mapped cache?

Example:

- Data size is 16 bytes = 2^4
- Address size is 64 bits
- Number of sets is 4096 = 2^{12}



How many bits has the **Tag?**

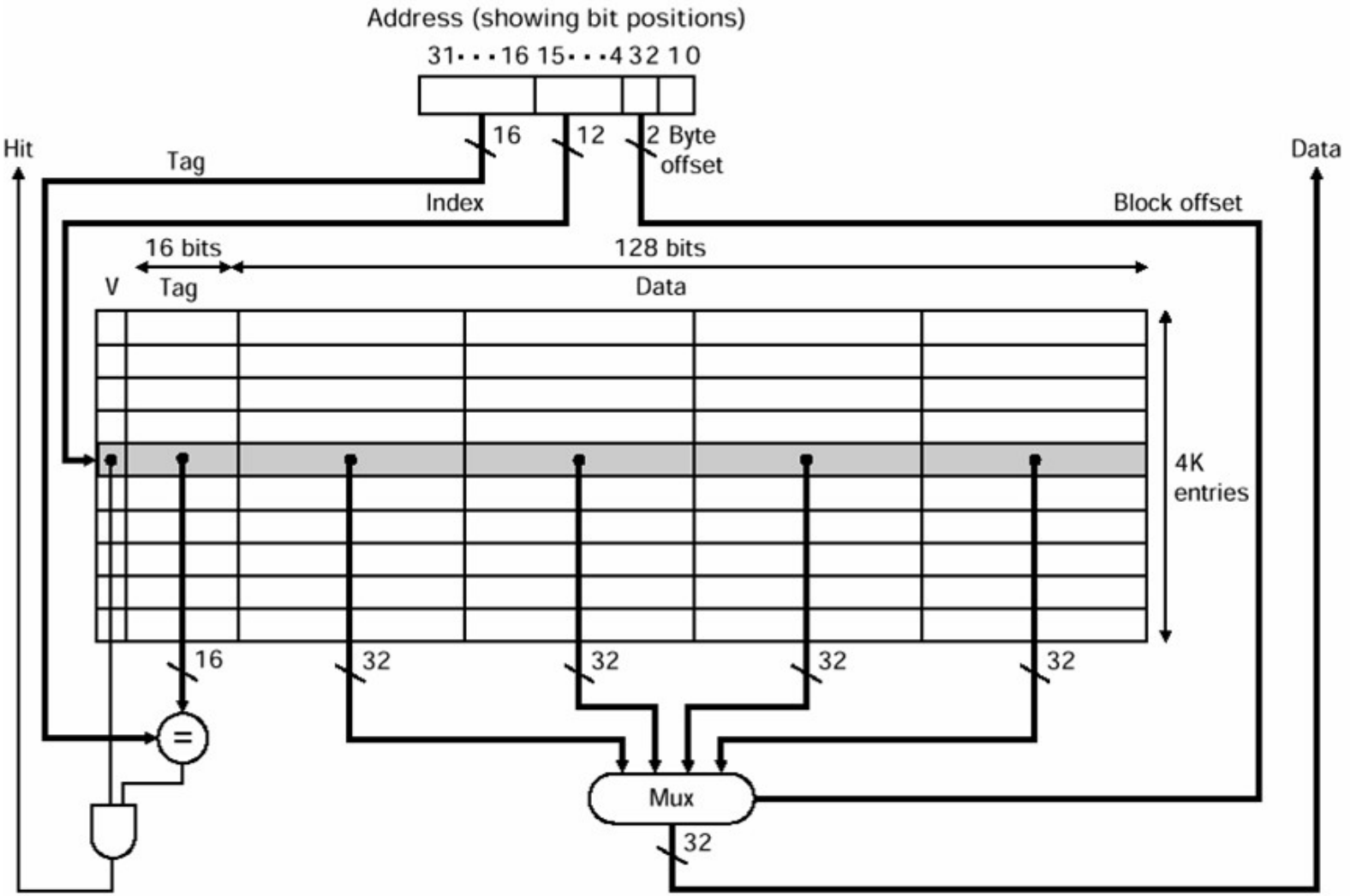
A) 60 bits

B) 55 bits

C) 54 bits

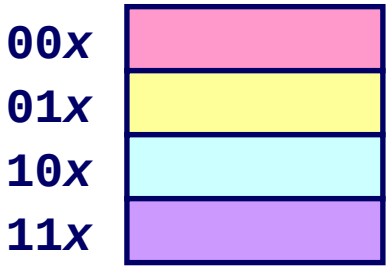
D) 48 bits

Direct Mapped Cache Implementation



Why to Use Middle-Order Bits for Index

4-řádková cache

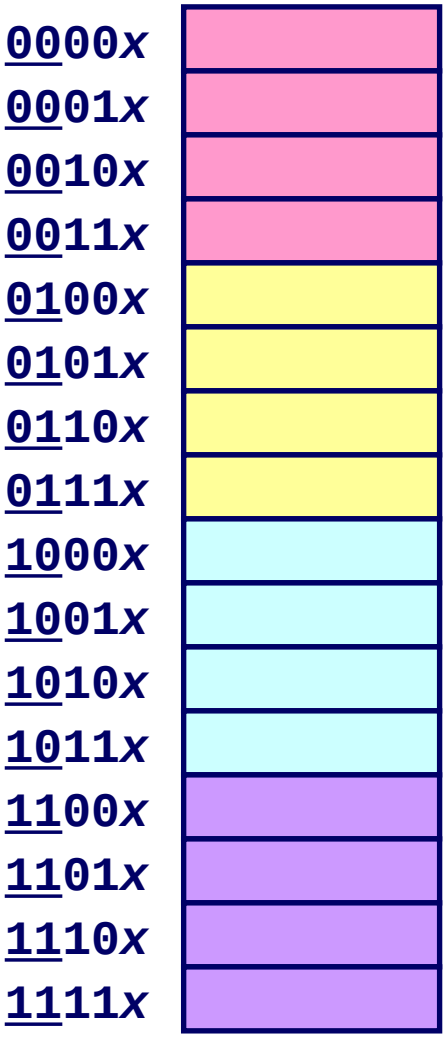


MSB – only part of cache used for continuous variables or code block

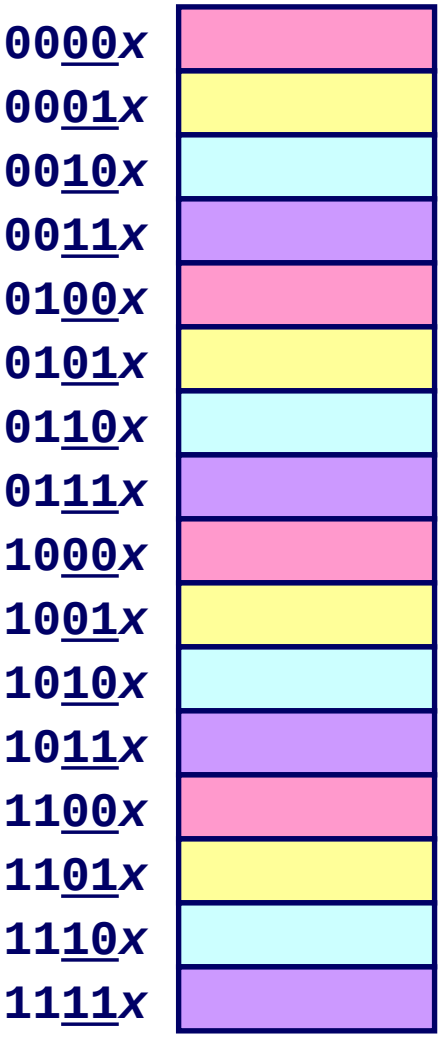
LSB – too small blocks, to much metadata per data byte

Middle-order – the compromise

High-Order Bit Index

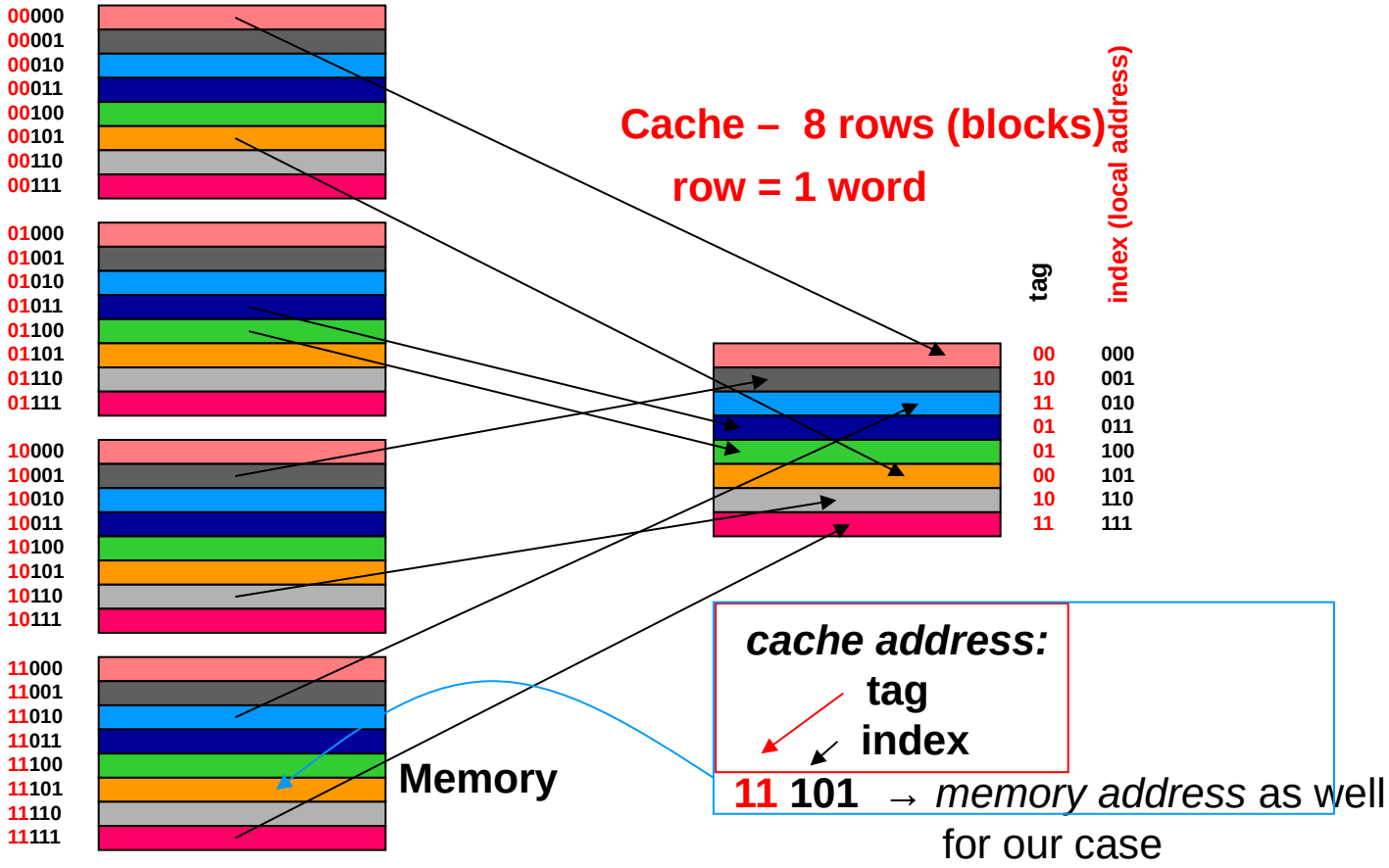


Middle-Order Bit Index



Direct Mapped Cache – 8 bit CPU Example

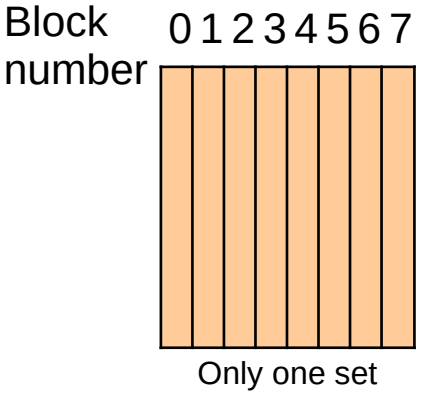
8-bit CPU (1 word = 1 byte)
5-bit address, data memory address range 0-31



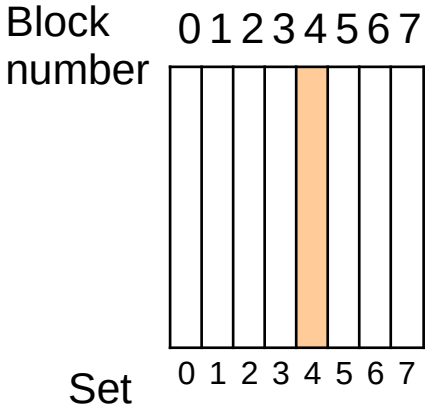
Example to Illustrate Base Cache Types

- The cache capacity 8 blocks. Where can be block/address 12 placed for
  - Fully associative
  - Direct mapped
  - N-way (set) associative – i.e. N=2 (2-way cache)

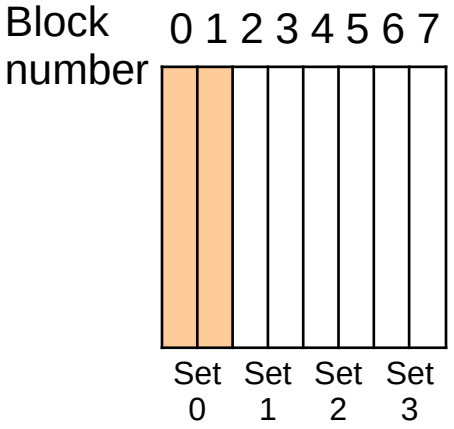
Fully associative:
Address 12 can be placed anywhere



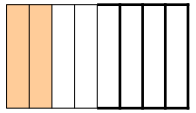
Direct mapped:
Address 12 placed only to block 4 (12 mod 8)



2-way associative:
Address 12 is placed into set 0 (12 mod 4)



2-way Set Associative Cache



Capacity – C

Number of sets – S

Block size – b

Number of blocks – B

Degree of associativity – N

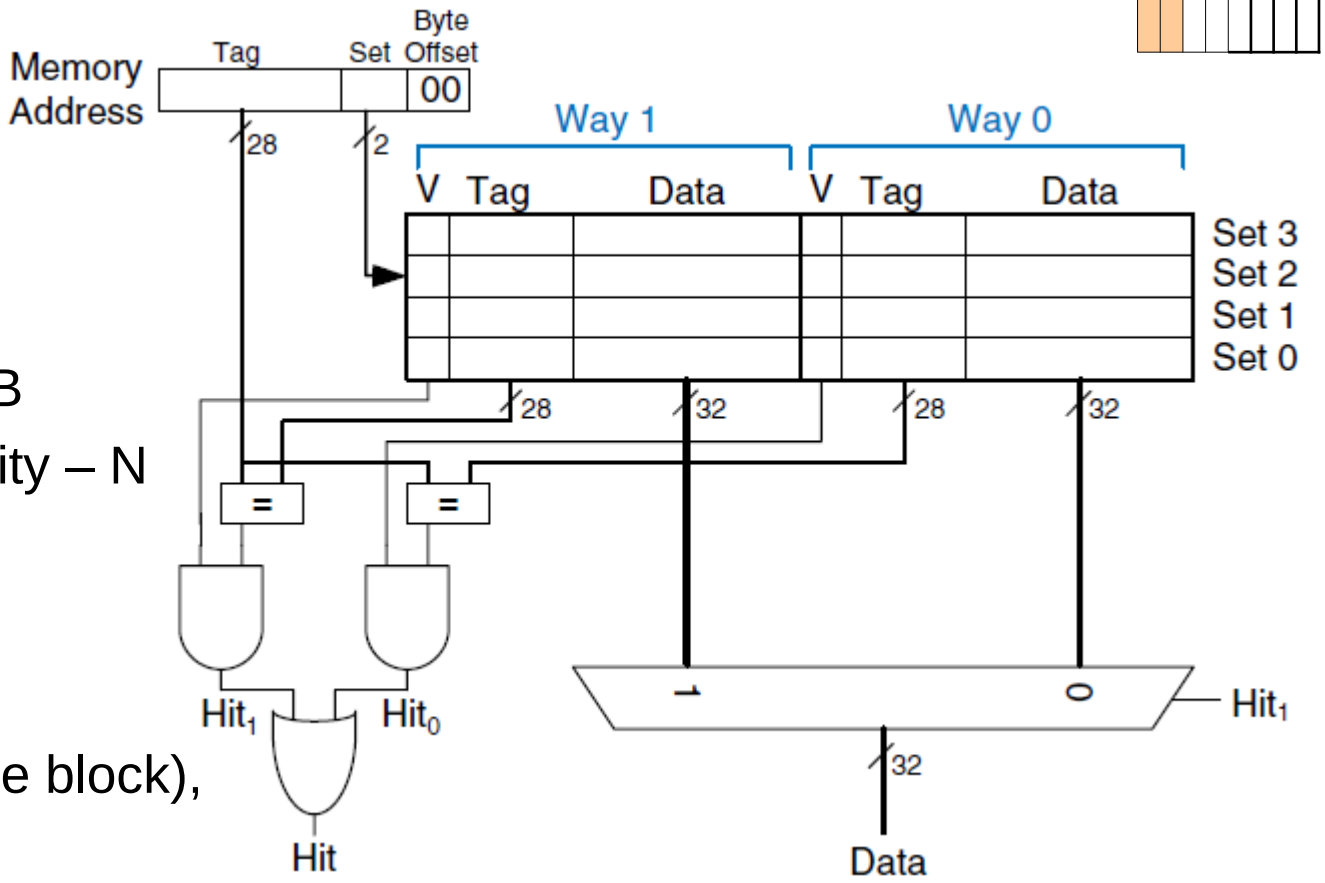
C = 8 (8 words),

S = 4,

b = 1 (one word in the block),

B = 8

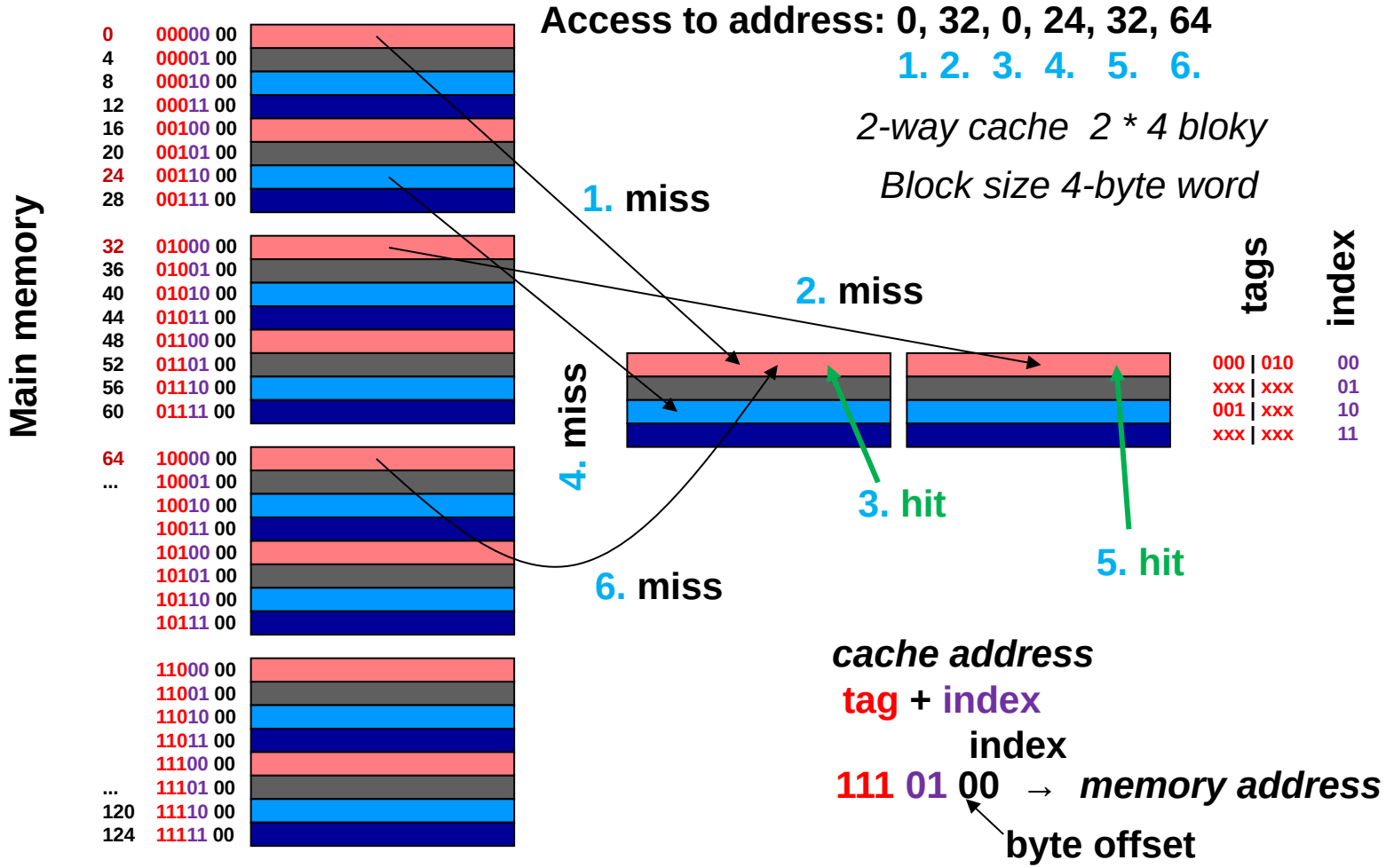
N = 2



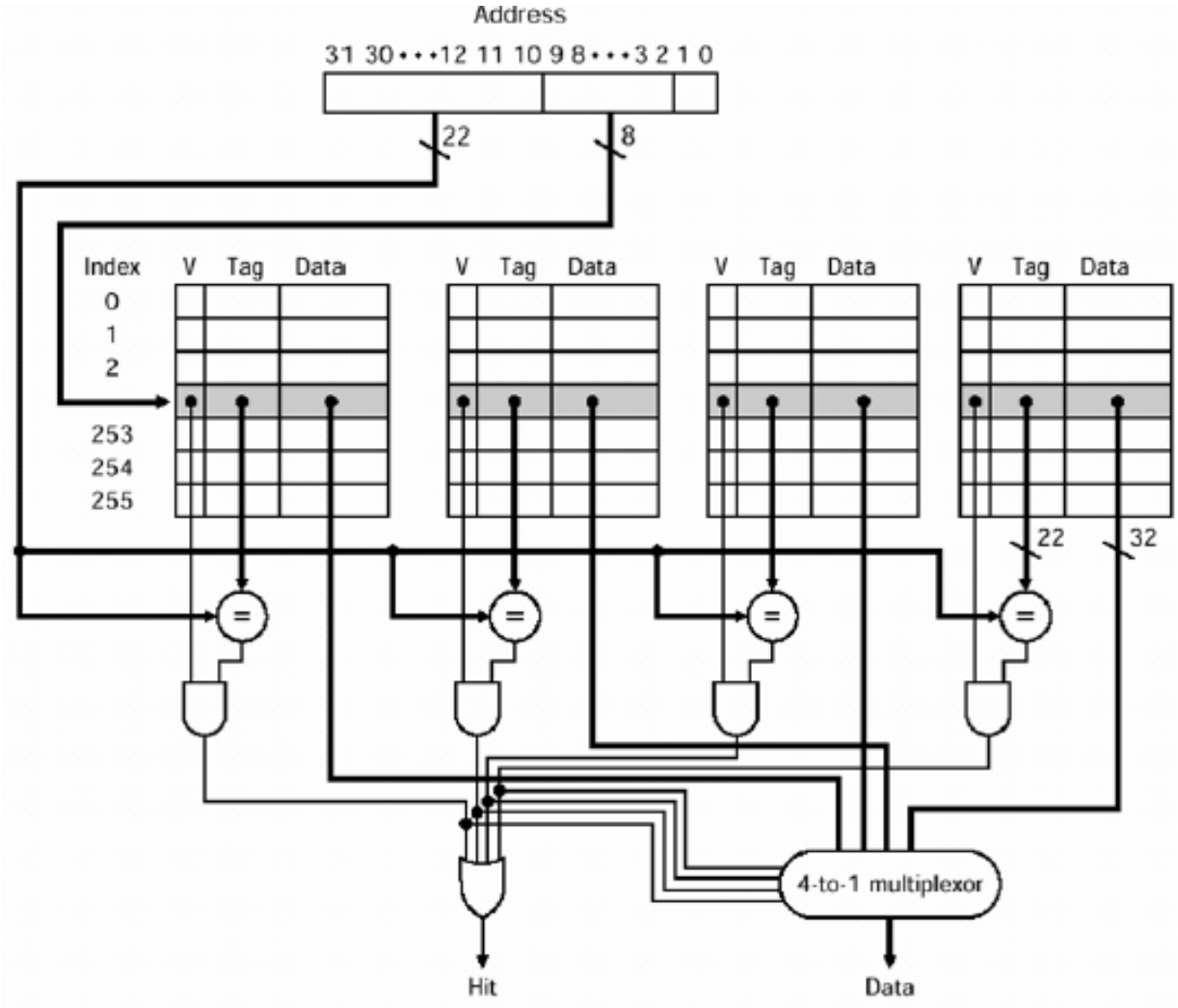
What is main advantage of higher associativity?

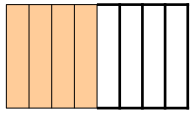
Two-way Set Associative Cache

128 bytes memory read and write as 4-byte words

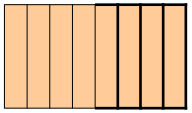


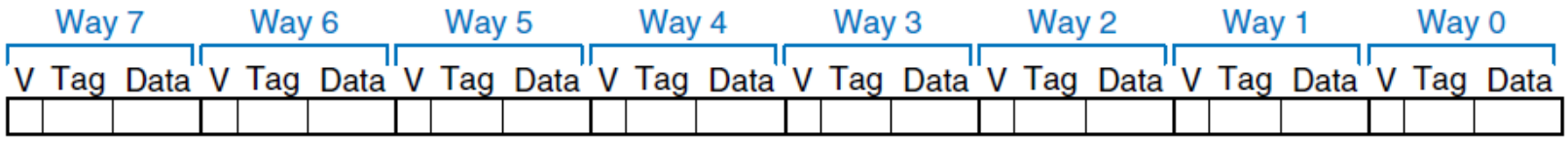
4-way Set Associative Cache





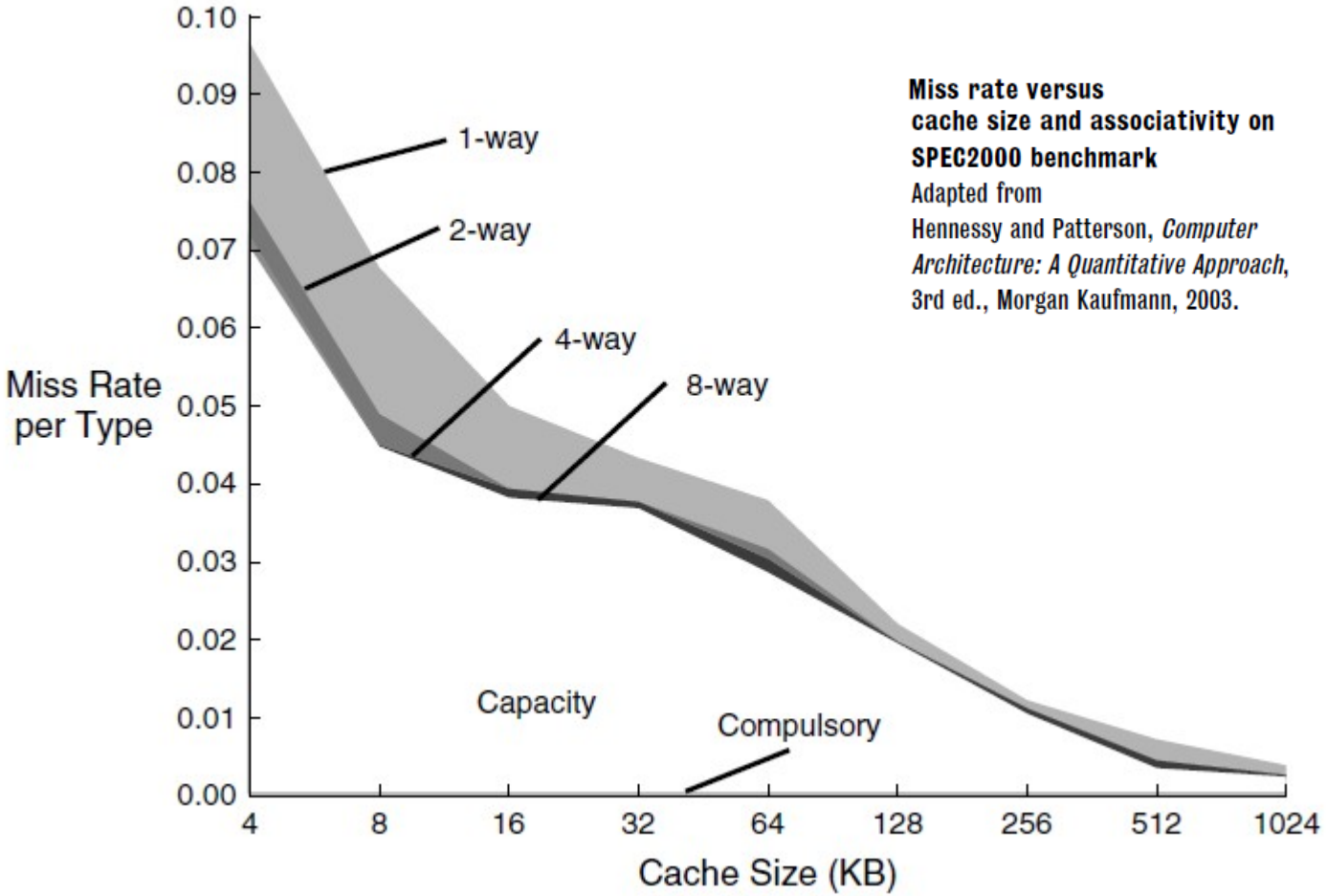
Fully Associative Cache as Special N-way Case





- From the above, a fully associative cache can be considered as N-way with only one set. N=B=C/(b·4)
- The same way we can define direct mapped cache as a special case where the number of ways is one.

Comparison of Different Cache Sizes and Organizations



Miss rate versus cache size and associativity on SPEC2000 benchmark
Adapted from Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed., Morgan Kaufmann, 2003.

Remember: 1. miss rate is not cache parameter/feature!
2. miss rate is not parameter/feature of the program!

Bubble sort – algorithm Example from Seminaries

```
int array[5]={5,3,4,1,2};
int main()
{
    int N = 5,i,j,tmp;
    for(i=0; i<N; i++)
        for(j=0; j<N-1-i; j++)
            if(array[j+1]<array[j])
            {
                tmp = array[j+1];
                array[j+1] = array[j];
                array[j] = tmp;
            }
    return 0;
}
```

What we can consider and expect from our programs?

Think about some typical data access patterns and execution flow.

Memory Hierarchy – 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

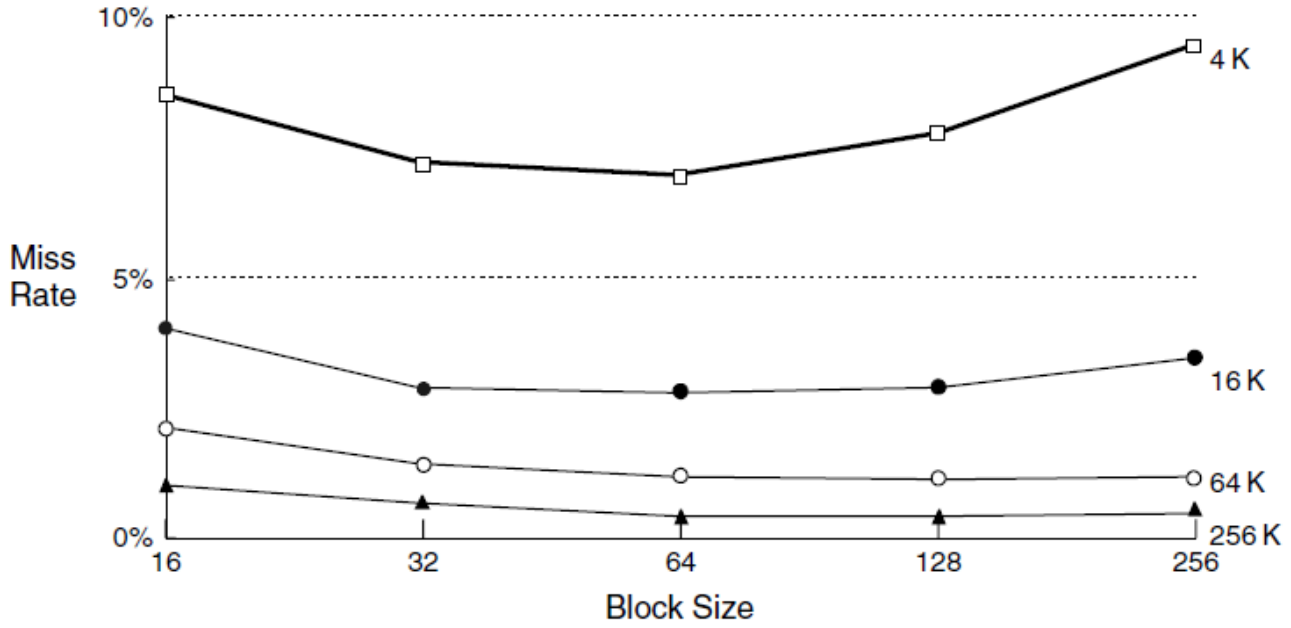
Source: Hennesy, Patterson

Memory Hierarchy Introduced Based on Locality

- The solution to resolve capacity and speed requirements is to build address space (data storage in general) as hierarchy of different technologies.
- Store input/output data, program code and its runtime data on large and cheaper secondary storage (hard disk)
- Copy recently accessed (and nearby) items from disk to smaller DRAM based main memory (usually under operating system control)
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory (cache) attached to CPU (hidden memory, transactions under HW control), optionally, tightly coupled memory under program's control
- Move currently processed variables to CPU registers (under machine program/compiler control)

What Can Be Gained from Spatial Locality?

Miss rate of consecutive accesses can be reduced by increasing block size. On the other hand, increased block size for same cache capacity results in smaller number of sets and higher probability of conflicts (set number aliases) and then to increase of miss rate.



Miss rate versus block size and cache size on SPEC92 benchmark Adapted from Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed., Morgan Kaufmann, 2003.

Six Basic Cache Optimizations

1) Larger block size

   Reduces compulsory misses; increases other misses, miss penalty

2) Larger cache size

   Reduces capacity/conflict misses; increases hit time, power, cost

3) Greater associativity

   Reduces conflict misses; increases hit time, power

4) Multiple cache levels

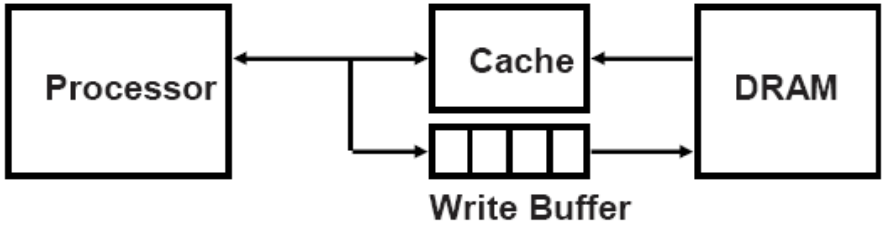
   Reduces Miss Penalty, allows for optimizations at each level

5) Prioritize read misses over writes

6) Avoid address translation of cache index

CPU Writes to Main Memory

- There is cache in the way
- **Data consistency** – requirement for data coherency for same address accessed through different paths
- **Write through** – data are written to the cache and write buffer/queue simultaneously
- **Write back** – data are written to the cache only and dirty bit is set. Write to the other level is delayed until cache line replacement time or to cache flush event
- **Dirty bit** – an additional flag for cache line. It Indicates that cached value is updated and does not correspond with the main memory.



V	Other bits, i.e. D	Tag	Data

The Process to Resolve **Cache Miss**

- Data has to be filled from main memory, but quite often all available cache locations which address can be mapped to are allocated

- **Cache content replacement policy** (offending cache line is invalidated either immediately or after data are placed in the write queue/buffer)
- **Random** – random cache line is evicted
- **LRU** (Least Recently Used) – additional information is required to find cache line that has not been used for the longest time
- **LFU** (Least Frequently Used) – additional information is required to find cache line that is used least frequently – requires some kind of forgetting
- **ARC** (Adaptive Replacement Cache) – combination of LRU and LFU concepts
- **Write-back** – content of the modified (dirty) cache line is moved to the write queue

Real Cache Organization

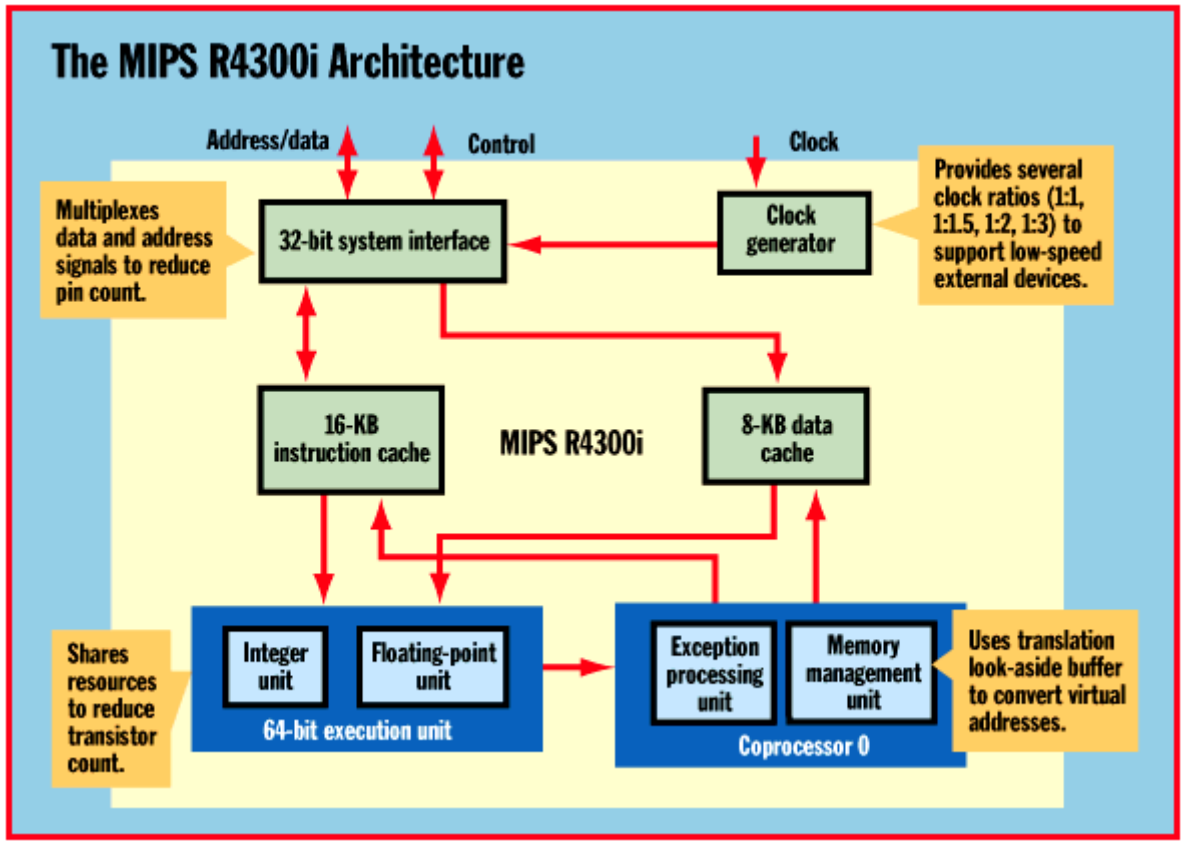
- **Tag** is index of the block corresponding to the cache set size in the main memory (that is address divided by block length and number or the cache lines in the set)
- **Data** are organized in cache line blocks, multiple words.
- **Valid bit** – marks line contents (or sometimes individual words) as valid.
- **Dirty bit** – corresponding cache line (word) was modified and write back will be required later
- Possible cache line states (for coherence protocols) – Invalid, Owned, Shared, Modified, Exclusive – out of the scope for this lecture

V	Flags, i.e. dirty bit D	Tag	Data

Multi-level Cache Organization

CPU Including Cache – Harvard Cache Architecture

Separated instruction and data cache
The concept of Von Neumann's CPU with Harvard cache is illustrated on a MIPS CPU family member, i.e. real CPU which is superset of the design introduced during lectures 2 and 4.

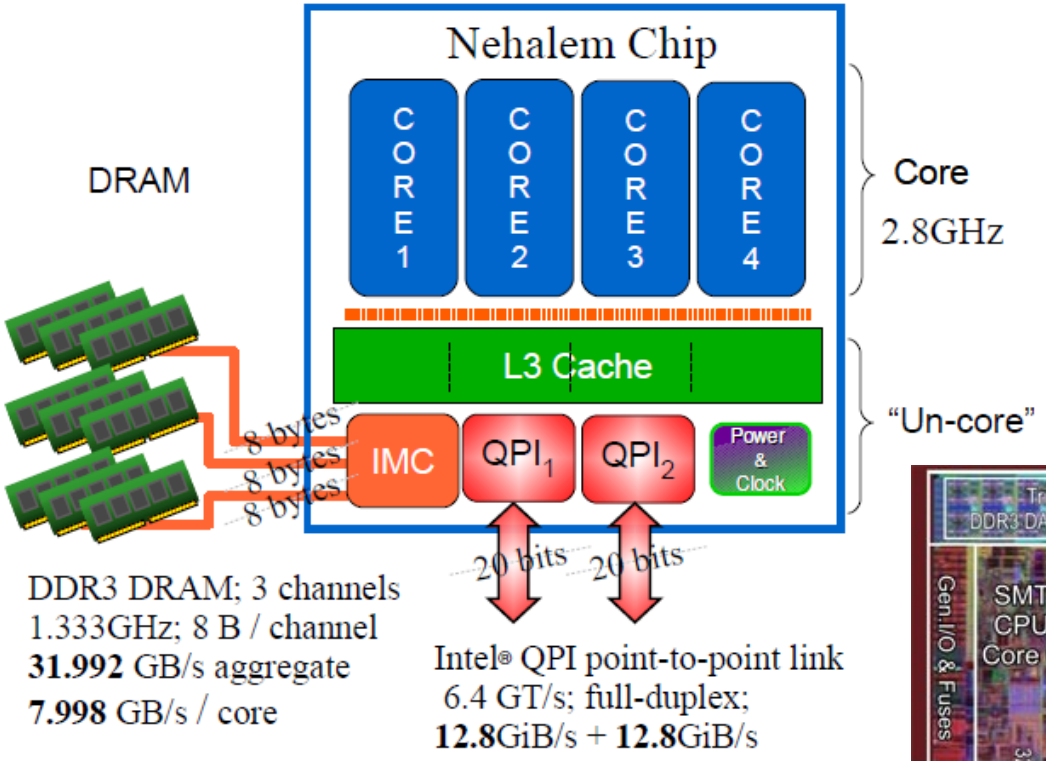


Multiple Cache Levels – Development Directions

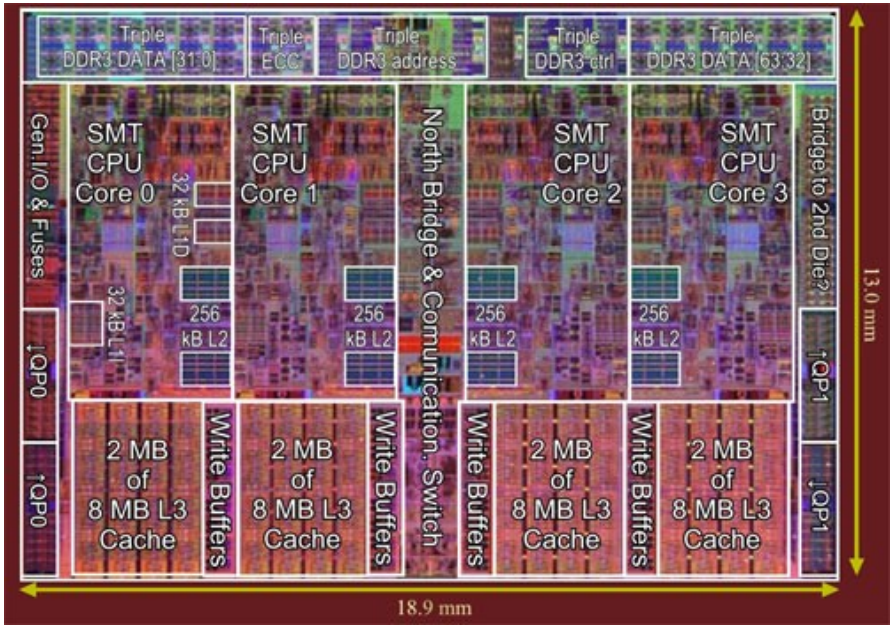
- Primary/L1 cache – tightly coupled to the CPU
  - Fast but small. Main objective: minimal Hit Time/latency
  - Usually separated caches for instruction and for data
  - Size usually selected so that cache lines can be virtually tagged without aliasing. (set/way size is smaller than page size)
- L2 cache resolves cache misses of the primary cache
  - Much bigger and slower but still faster than main memory. Main goal: low Miss Rate
- L2 cache misses are resolved by main memory
- Trend to introduce L3 caches, inclusive versus exclusive cache

	Usual for L1	**Usual for L2**
Block count	250-2000	15 000-250 000
KB	16-64	2 000-3 000
Block size in bytes	16-64	64-128
Miss penalty (cycles)	10-25	100-1 000
Miss rates	2-5%	0,1-2%

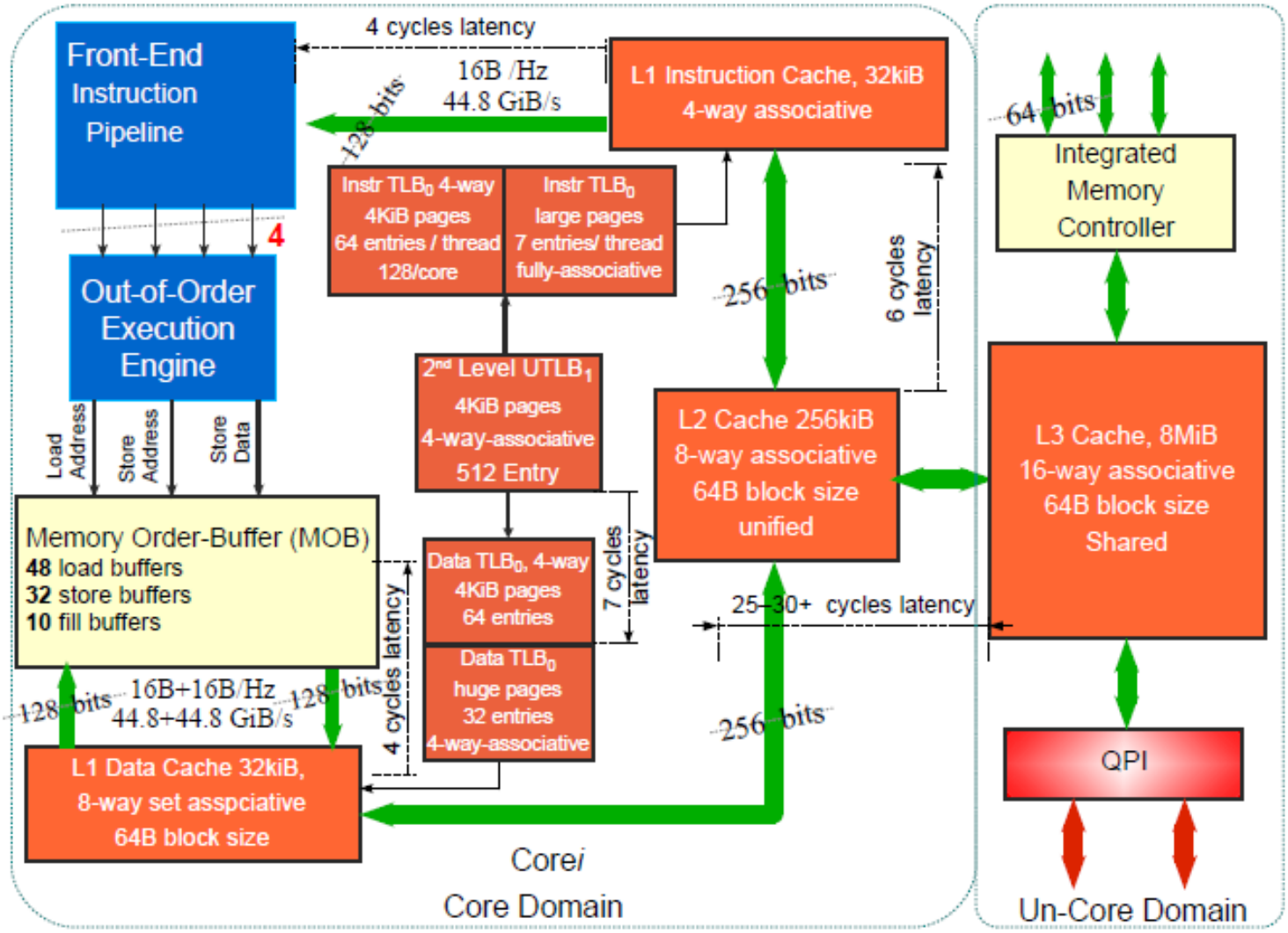
Intel Nehalem – Example of Harvard Three-level Cache



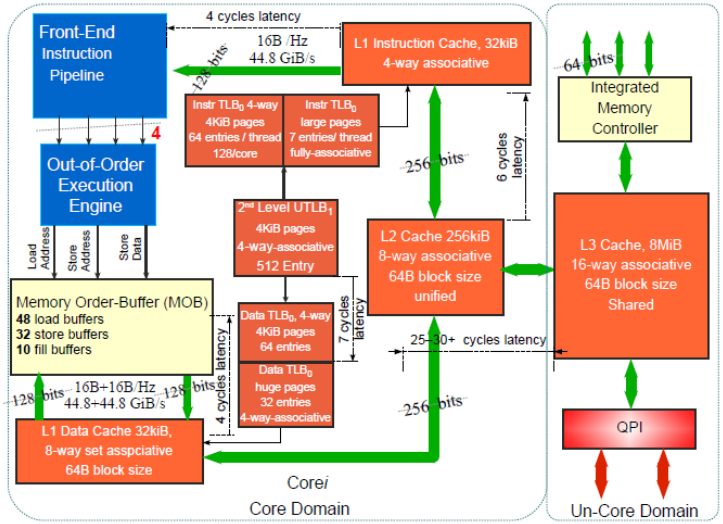
- IMC: integrated memory controller with 3 DDR3 memory channels,
- QPI: Quick-Path Interconnect ports



Intel Nehalem – Memory Subsystem Structure



Notes for Intel Nehalem Example



- Block size: 64B
- CPU reads whole cache line/block from main memory and each is 64B aligned
- (6 LS bits are zeros), partial line fills allowed
- L1 – Harvard. Shared by two (H)threads

  instruction – 4-way 32kB, data 8-way 32kB
- L2 – unified, 8-way, non-inclusive, WB
- L3 – unified, 16-way, inclusive (each line stored in L1 or L2 has copy in L3), WB
- Store Buffers – temporal data store for each write to eliminate wait for write to the cache or main memory. Ensure that final stores are in original order and solve “transaction” rollback or forced store for:

  - exceptions, interrupts, serialization/barrier instructions, lock prefix,..
- TLBs (Translation Lookaside Buffers) are separated for the first level

  Data L1 32kB/8-ways results in 4kB range (same as page) which allows to use 12 LSBs of virtual address to select L1 set in parallel with MMU/TLB

Two-level Cache (Pentium 4) Example

$$AMAT = HitTime_{L1} + MissRate_{L1} * (HitTime_{L2} + MissRate_{L2} * MissPenalty_{L2})$$

L1: 2 cycles access time

L2: 19 cycles access time

Memory access time: 240 cycles

Program behavior: 5% L1 and 25% L2 miss rates

$$AMAT_{L1+L2} = 2 + 0.05 * (19 + 0.25 * 240) = 5.95$$

$$AMAT_{L1} = 2 + 0.05 * 240 = 14$$

Source: Gedare Bloom https://www.uccs.edu/cs/about/faculty/gedare-bloom

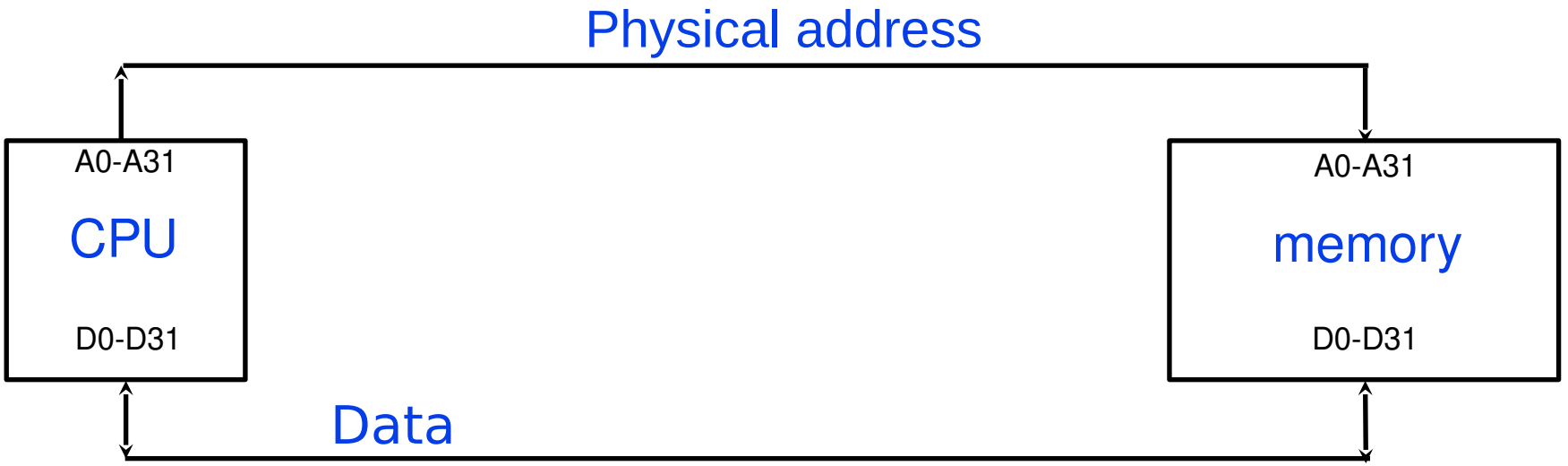
Advanced Cache Optimizations

Hennessy & Patterson, 2.3

1) Small and simple L1 Cache
2) Way Prediction
3) Pipelined and Banked Caches
4) Non-blocking Caches
5) Critical Word First and Early Restart
6) Merging Write Buffers
7) Compiler Techniques: Loop interchange/blocking
8) Prefetching: Hardware / Software

Virtual Memory

Physical Address to Memory?

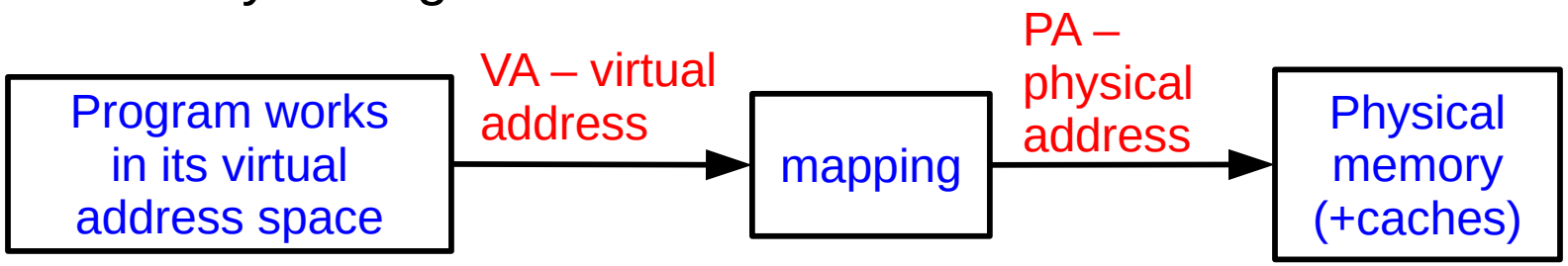


Virtual Memory Motivation …

- Normally we have several tens / hundreds of processes running on your computer…
- Can you imagine a situation where we would divide physics memory (for example, 1 GB) between these processes? How big a piece of memory would belong to one process? How would we deal with collisions - when would a program intentionally (for example, a virus) or inadvertently (by a programmer's error - working with pointers) want to write to a piece of memory that we reserved for another process?
- The solution is just virtual memory…
- We create an illusion to every process that the entire memory is just its and can move freely within it.
- We will even create the illusion of having, for example, 4GB of memory even though the physical memory is much smaller. The process does not distinguish between physical memory and disk (the disk appears to be memory).
- **The basic idea: The process addresses the virtual memory using virtual addresses.** We then have to translate them somehow into physical addresses.

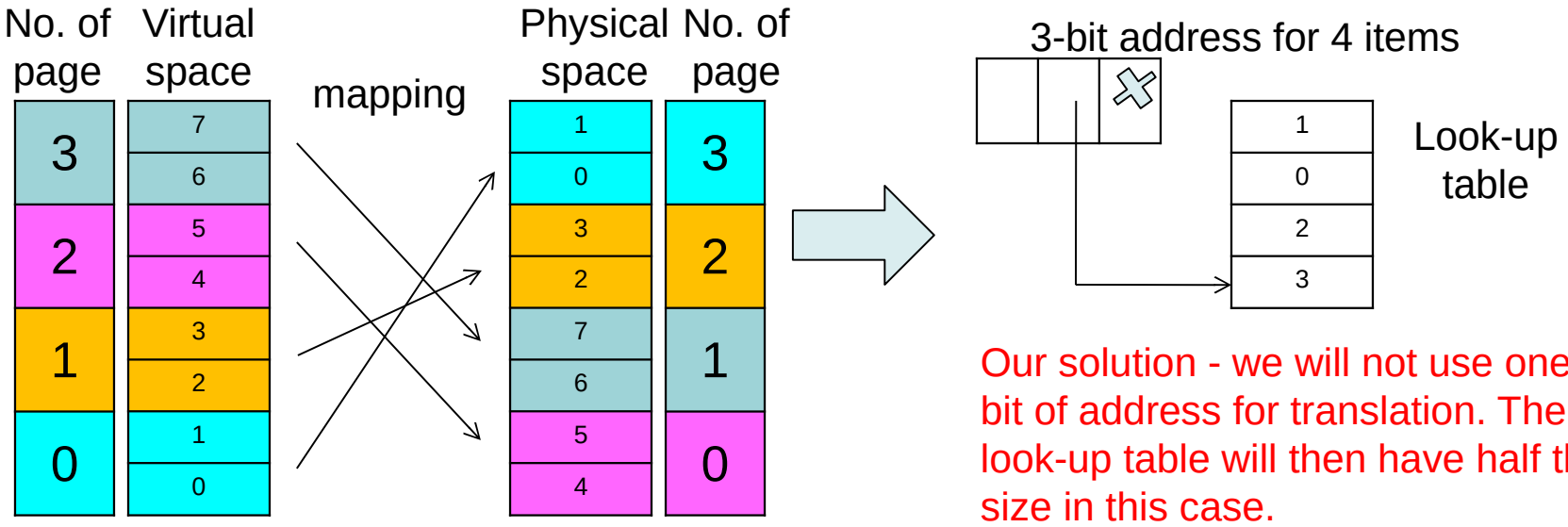
Virtual Memory

- **Virtual memory (VM)** – a separate address space is provided to each process, it is (can be) organized independently on the physical memory ranges and can be even bigger than the whole physical memory
- Programs/instructions running on the CPU operate with data only through virtual addresses
- Translation from virtual address (VA) to physical address (PA) is implemented in HW (MMU, TLB).
- Common OSes implement virtual memory through paging which extends concept even to swapping memory content onto secondary storage



Virtual Memory – Translation by Pages

- **Mapping from any virtual address to any physical address is a virtually unrealistic requirement!**
- **Solution:** Divide the virtual space into equal parts - virtual pages, and physical memory on physical pages=frames. Size of the virtual and physical pages are the same. In our example, we have a 2B page.

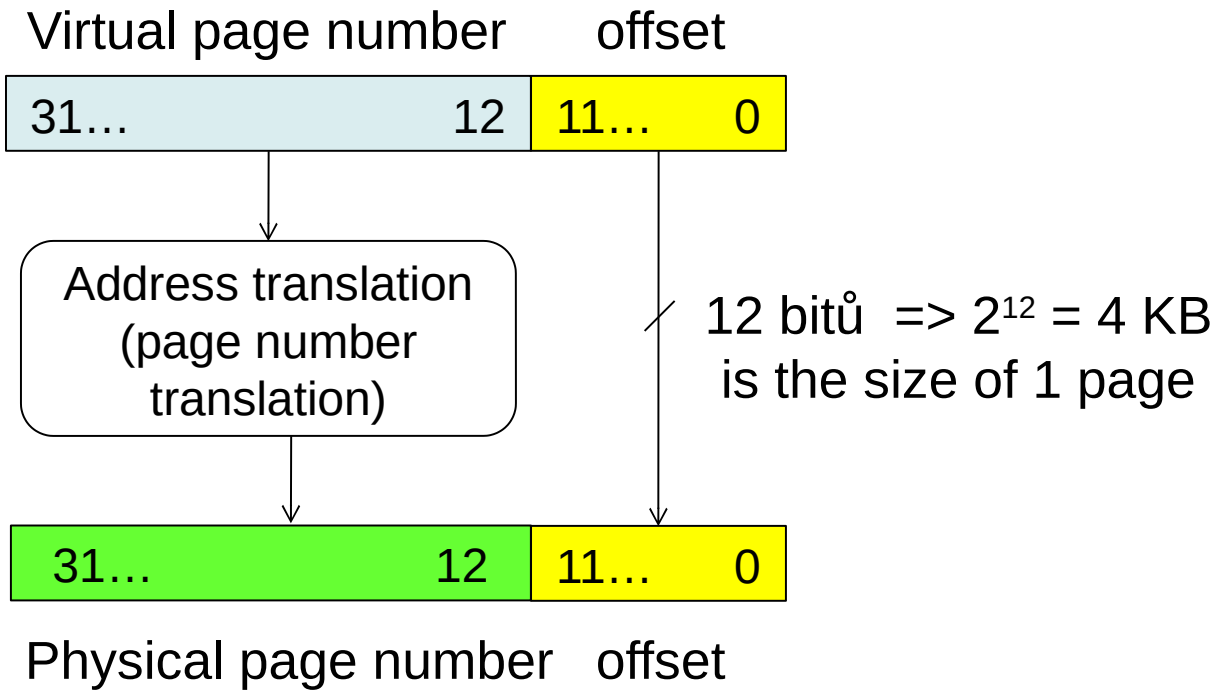


Our solution - we will not use one bit of address for translation. The look-up table will then have half the size in this case.

- So our solution translates virtual addresses in groups… We move inside the page using the bit we ignored during the translation. We are able to use the entire address space.

Virtual and Physical Addressing - in More Detail

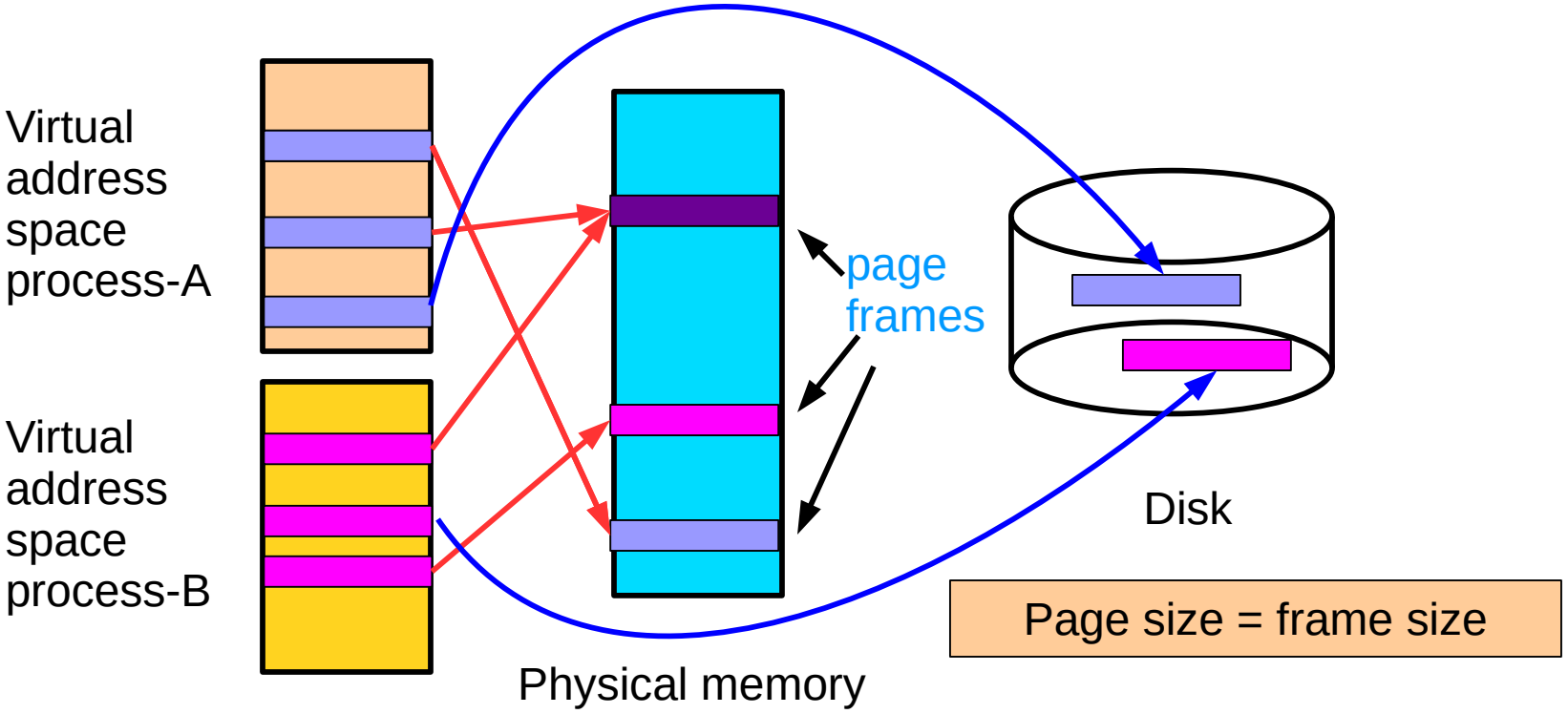
- Assume a 32-bit virtual address, **1GB of physical memory**, and a 4-KB page size



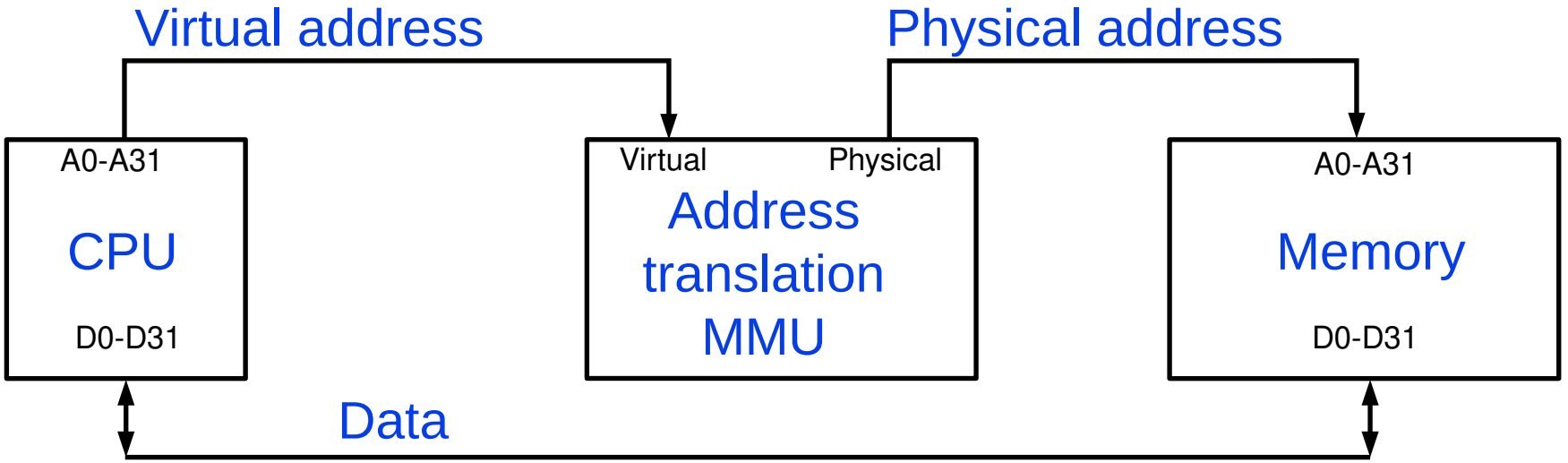
The arrangement of the translation, where the lowest bits of the address remain, has a very important practical consequence, as seen later.

Virtual Memory – Paging

- Process virtual memory content is divided into aligned pages of same size (power of 2, usually 4 kB)
- Physical memory consists of page frames of the same size



Virtual/Physical Address and Data

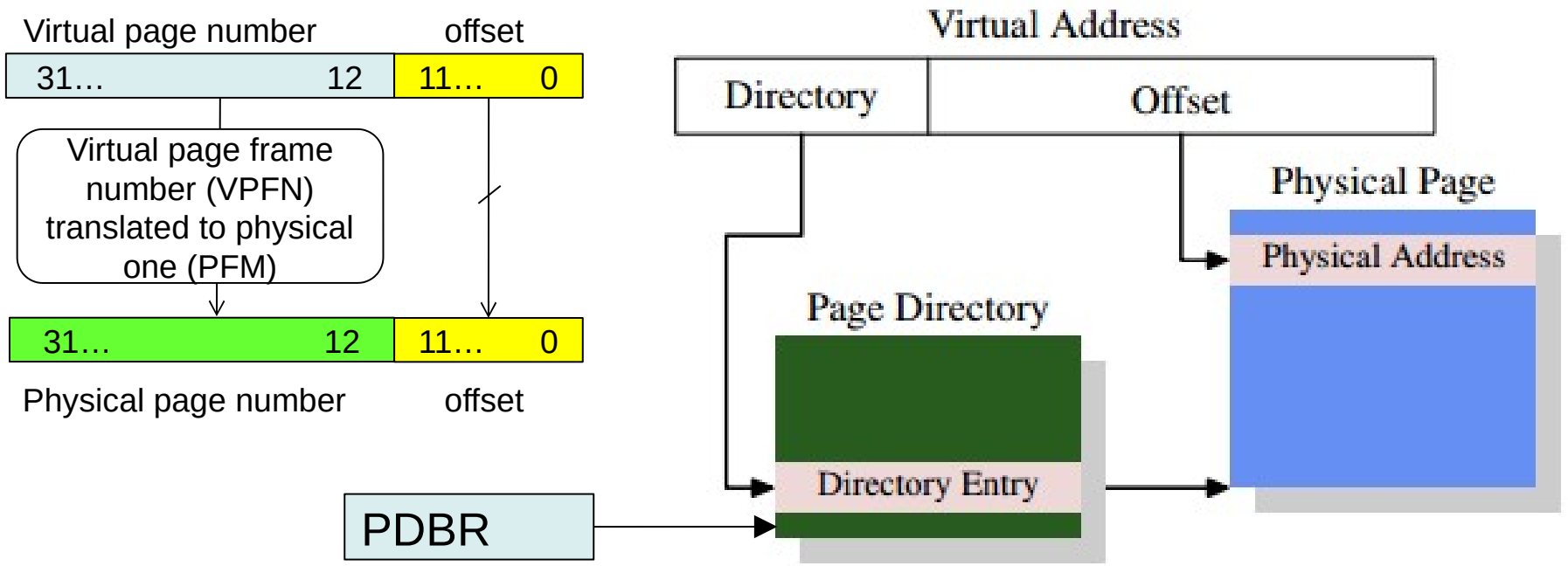


- Paging (realization of virtual memory) does not interfere with the principle of spatial location => important for cache.
- Data on adjacent virtual addresses will be stored in physical memory side by side (of course if they do not cross the page boundary). Locality even in Physically Indexed and Physically Tagged (PIPT) cache is preserved.

Address Translation

- Page Table
  - Root pointer/page directory base register (x86 CR3=PDBR)
  - Page table directory PTD
  - Page table entries PTE
- Basic mapping unit is a page (page frame)
- Page is basic unit of data transfers between main memory and secondary storage
- Mapping is implemented as look-up table in most cases
- Address translation is realized by **Memory Management Unit** (MMU)
- Example follows on the next slide:

Single-level Page Table (MMU)



- Page directory is represented as data structure stored in main memory. OS task is to allocate physically continuous block of memory (for each process/memory context) and assign its start address to special CPU/MMU register.
- PDBR - page directory base register – for x86 register CR3 – holds physical address of page directory start, alternate names PTBR - page table base register – the same thing, page table root pointer URP, SRP on m68k

How to translate virtual address to physical?

- Typical page size is 4 kB = 2^12
- 12 bits (offset) are enough to address data in page (frame). There are 20 bits left for address translation on 32-bit address/architecture.
- The fastest map/table look-up is indexing ⇒ use array structure
- Test whether page is in memory:

```
#define PAGE_BITS 12
if ((pdir[virt >> PAGE_BITS] & PRESENT) != 0)
```

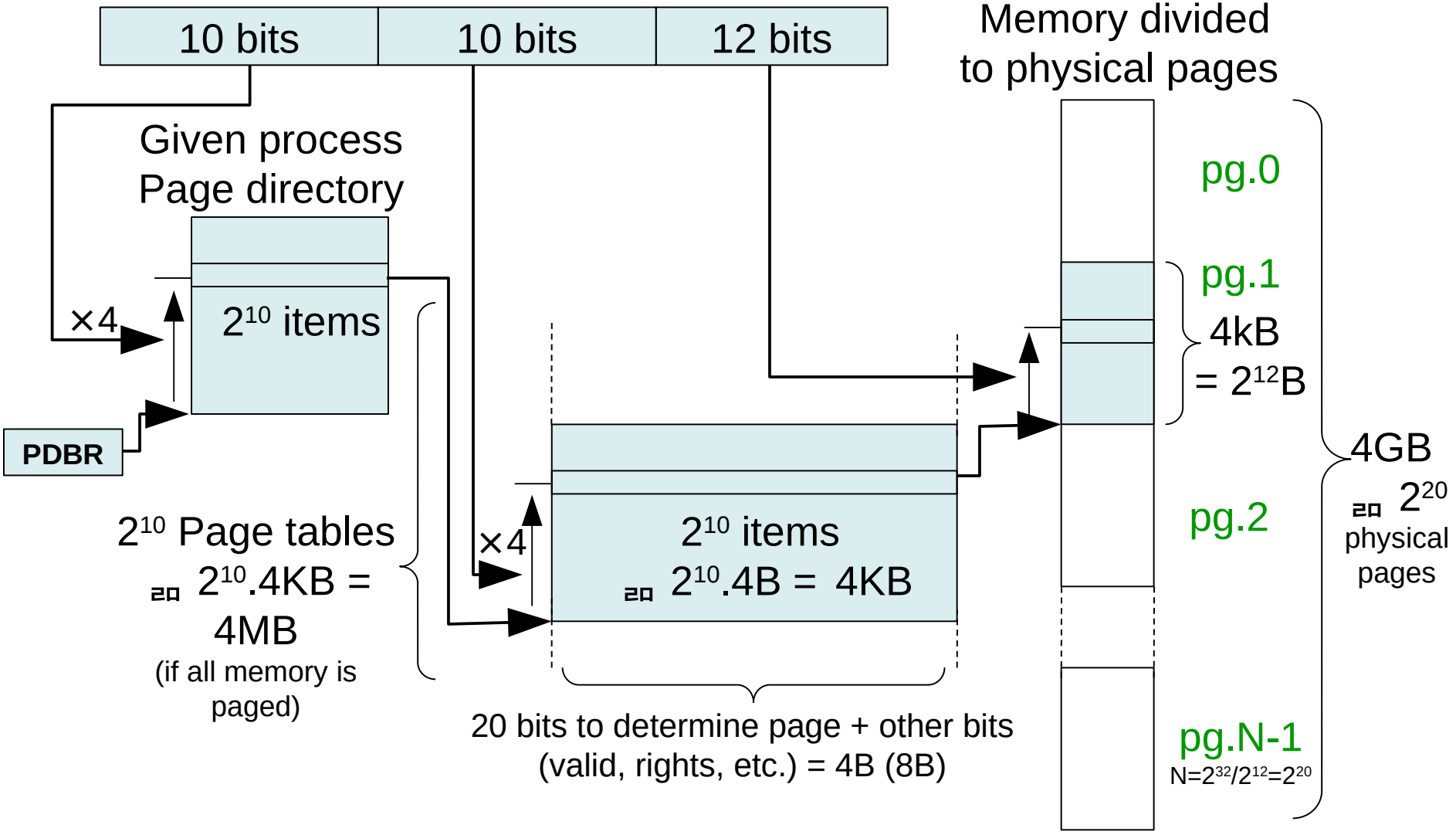
- Memory address:

```
#define PAGE_SIZE (1 << PAGE_BITS)
#define PAGE_MASK (PAGE_SIZE - 1)
addr = (pdir[virt >> PAGE_BITS] & ~PAGE_MASK) |
            (virt & PAGE_MASK);
```

But Consider Memory Consumed by Page Table …

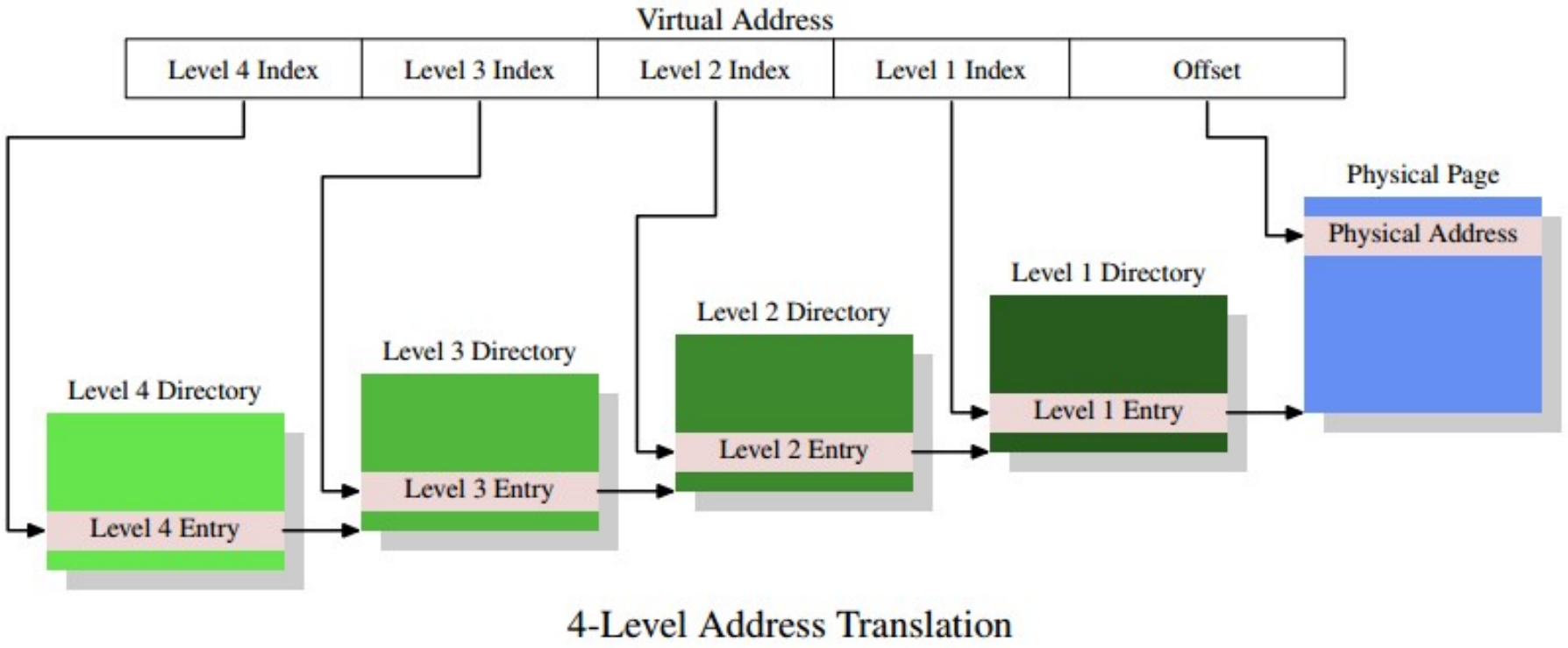
- The page directory must be in memory
- Each process has it's own table
- The page directory is an array of 2^{20} entries (PTE).
  - 1 entry has 4 bytes = 32 bits
  - Whole page table has 4MiB
  - It is too much memory
- 64 bit system has 2^{52} entries (PTE).
  - 1 entry has 8 bytes = 64 bits
  - Whole page table has 32 PiB
  - This is impossible
- Solution: multi-level page table – lower levels populated only for used address ranges

Two Level Paging on Intel x86 in Plain 32-bit Mode



Remark: for 10,10,12-bits each 1024 entries (32-bits each) page table fits exactly in 4k page

Multi-Levels Page Table



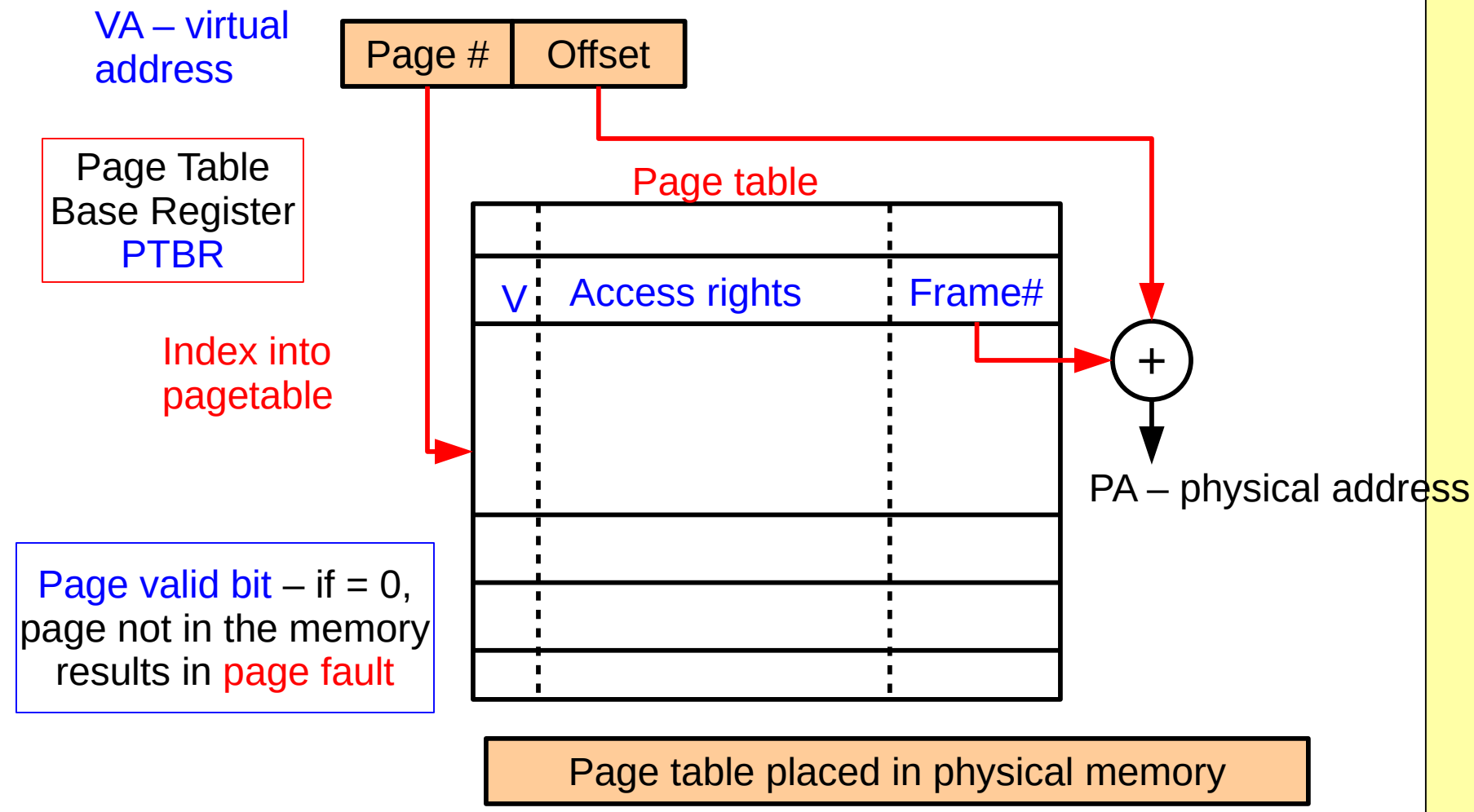
4-Level Address Translation

Multi-level Paging on x86 System in 64-bit Mode

In the case of a 64-bit virtual address, it is customary to use fewer bits for a physical address - for example, 48, or 40. Even virtual address is divided to top part for system and low part for user and gap I left to lower number of levels of pagetable

- Intel Core i7 uses 4-level paging and 48 bit address space
  - Page Table level 1: Page global directory (9 bits)
  - Page Table level 2: Page upper directory (9 bits)
  - Page Table level 3: Page middle directory (9 bits)
  - Page Table level 4: Page table (9 bits)
- Only the first 3-levels are used in case of 2 MB pages
- 5-level page table mode has been introduced as option in Ice Lake microarchitecture to aid with virtualization and shadow pagetables

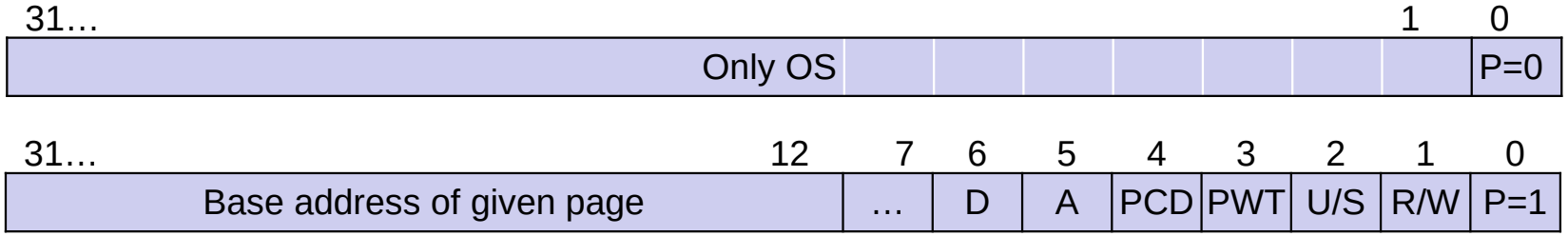
What Is in Page Table Entries?



Page Table - Meaning of Bits in Page Table Entry (PTE)

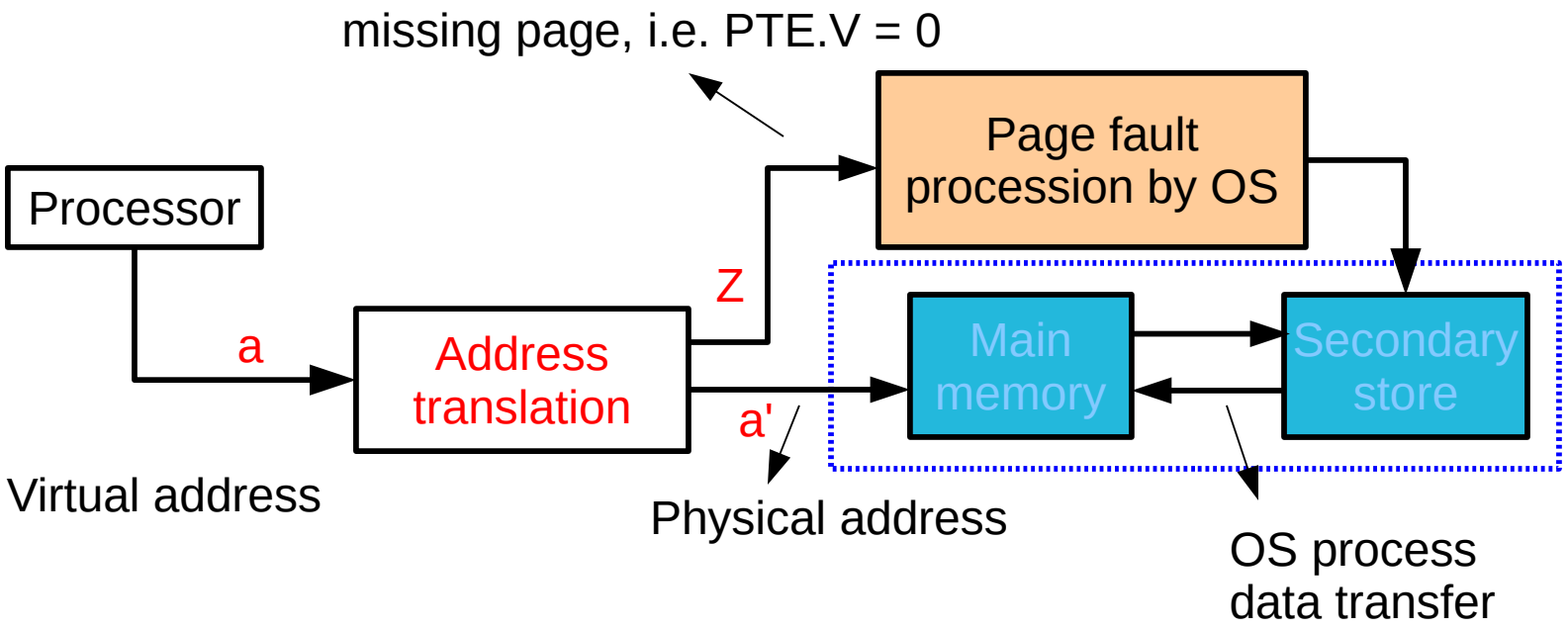
Based on simplified x86 model in 32-bits mode, other architectures similar

Let's look at the item Page Table



- **P** -bit 0: Present bit – *determines whether the page is in memory (1) or on disk (0) Sometimes this bit is called V - valid*.
- **R/W** -bit 1: Read/Write: *if 1 then R/W; otherwise only read*
- **U/S** -bit 2: User/Supervisor: *1 – user access; 0 – only OS*
- **PWT** -bit 3: Write-through/Write-back – *writing strategy for the page*
- **PCD** -bit 4: Cache disabled/enabled – *some peripherals are mapped directly into memory (memory mapped I/O), allowing write / read to / from the periphery. These memory addresses are then I/O ports and they are not written into cache.*
- **A** -bit 5: Accessed –*Whether we have read / written - helps decide which pages to remove when we need to free up memory space*
- **D** bit 6: Dirty bit – *it is set if we wrote into page*.
- 11..7 bit - special use, as memory type, or when to update cache, etc.
- 31-12 bit - Physical Page Address

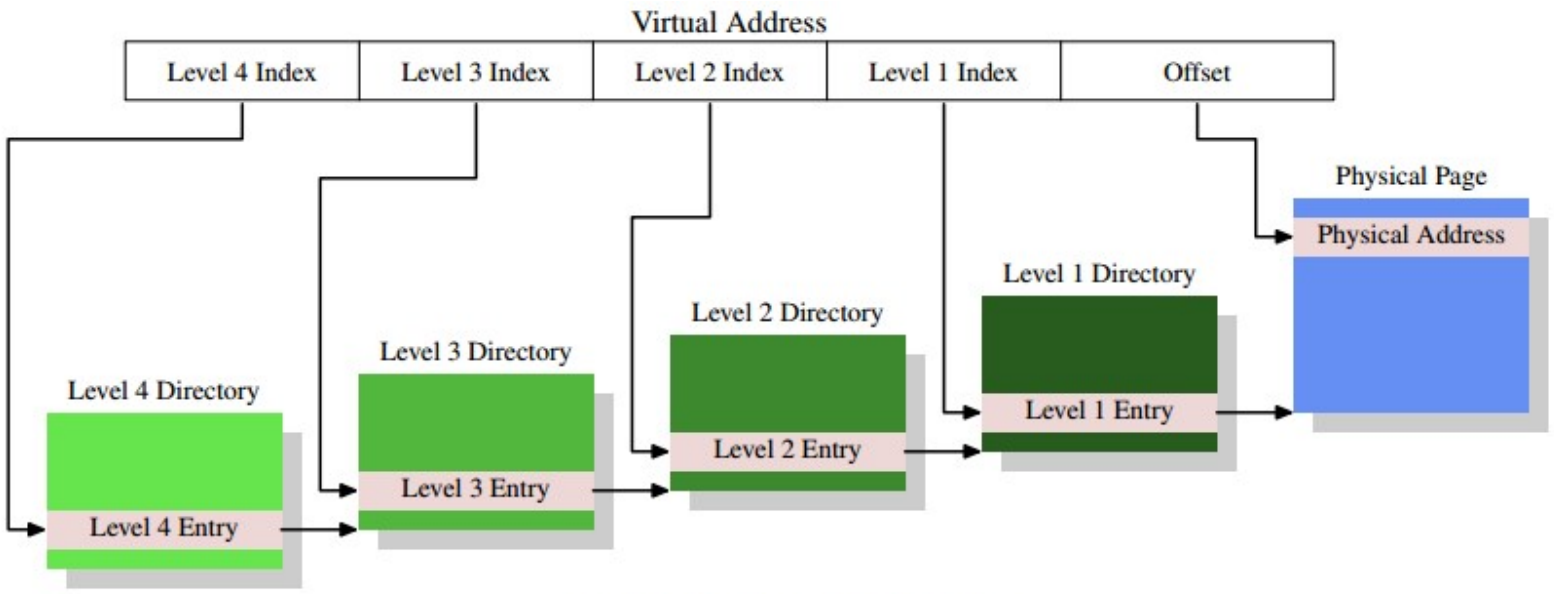
Virtual Memory – Hardware and Software Interaction



How to Resolve Page-fault

- Check first that fault address belongs to process mapped areas
- If free physical frame is available
  - The missing data are found in the backing store (usually swap or file on disk)
  - Page content is read (usually through DMA, Direct Memory Access, part of some future lesson) to the allocated free frame. If read blocks, the OS scheduler switches to another process.
  - End of the DMA transfer raises interrupt, OS updates page table of original process.
  - Scheduler switches to (resumes) original process.
- If no free frame is available, some frame has to be released
  - The LRU algorithm finds (unpinned – not locked in physical memory by OS) frame, which can be released.
  - If the Dirty bit is set, frame content is written to the backing store (disc). If store is a swap – store to the PTE or other place block nr.
  - Then continue with gained free physical frame.

Multi-level Page Table – Translation Overhead

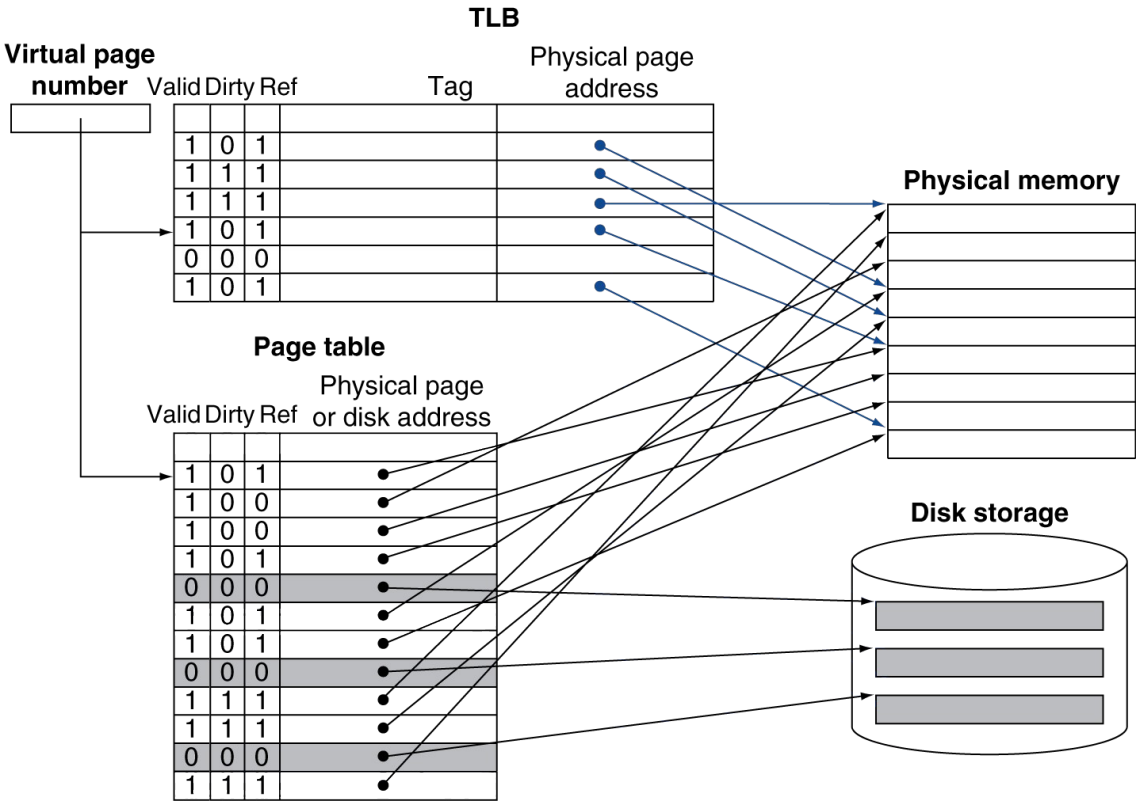


4-Level Address Translation

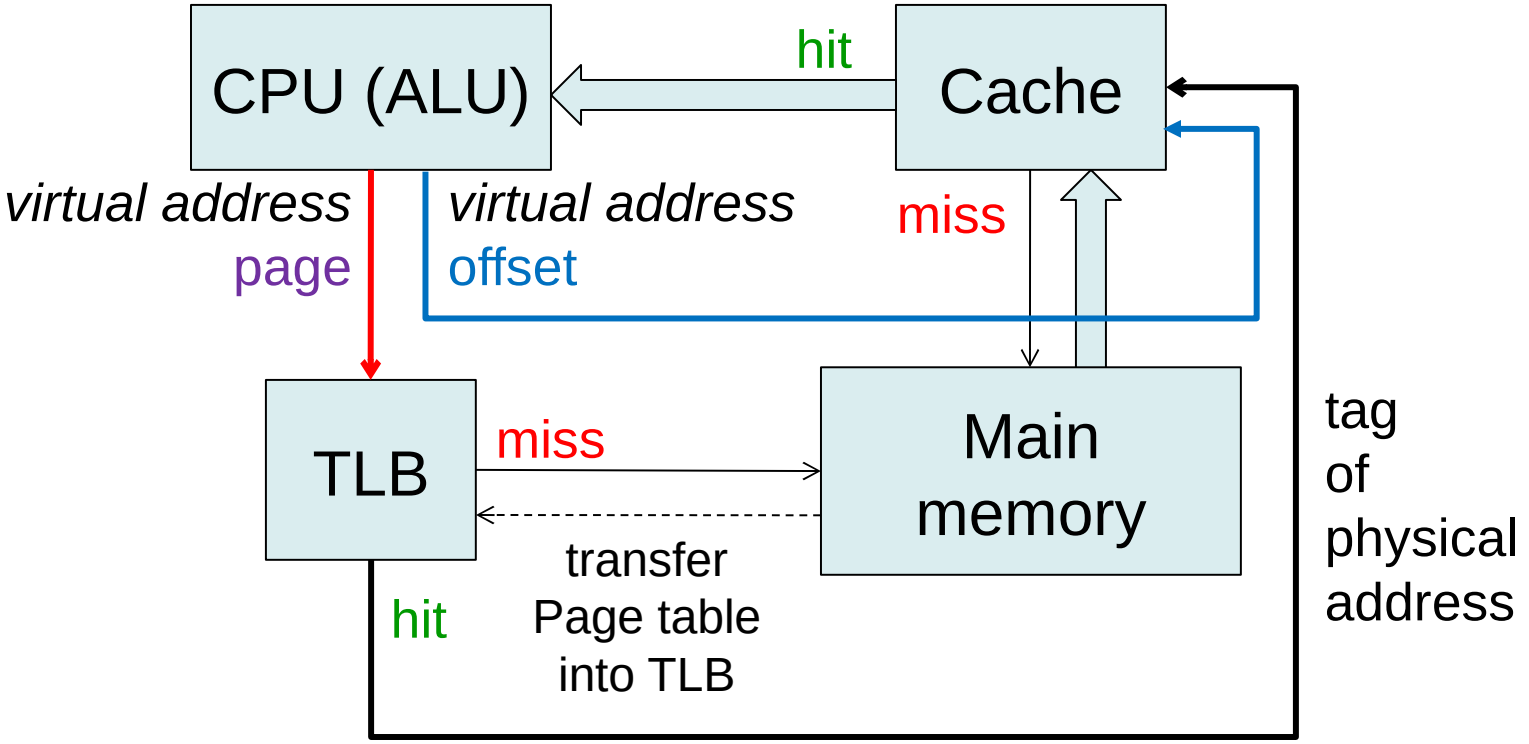
- Translation would take long time, even if entries for all levels were present in cache. (One access per level, they cannot be done in parallel.)
- The solution is to cache found/computed physical addresses
- Such cache is labeled as Translation Look-Aside Buffer
- Even multi-level translation caching are in use today

Fast MMU/Address Translation Using TLB

- Translation-Lookaside Buffer, or may it be, more descriptive name – Translation-Cache
- Cache of frame numbers where key is page virtual addresses

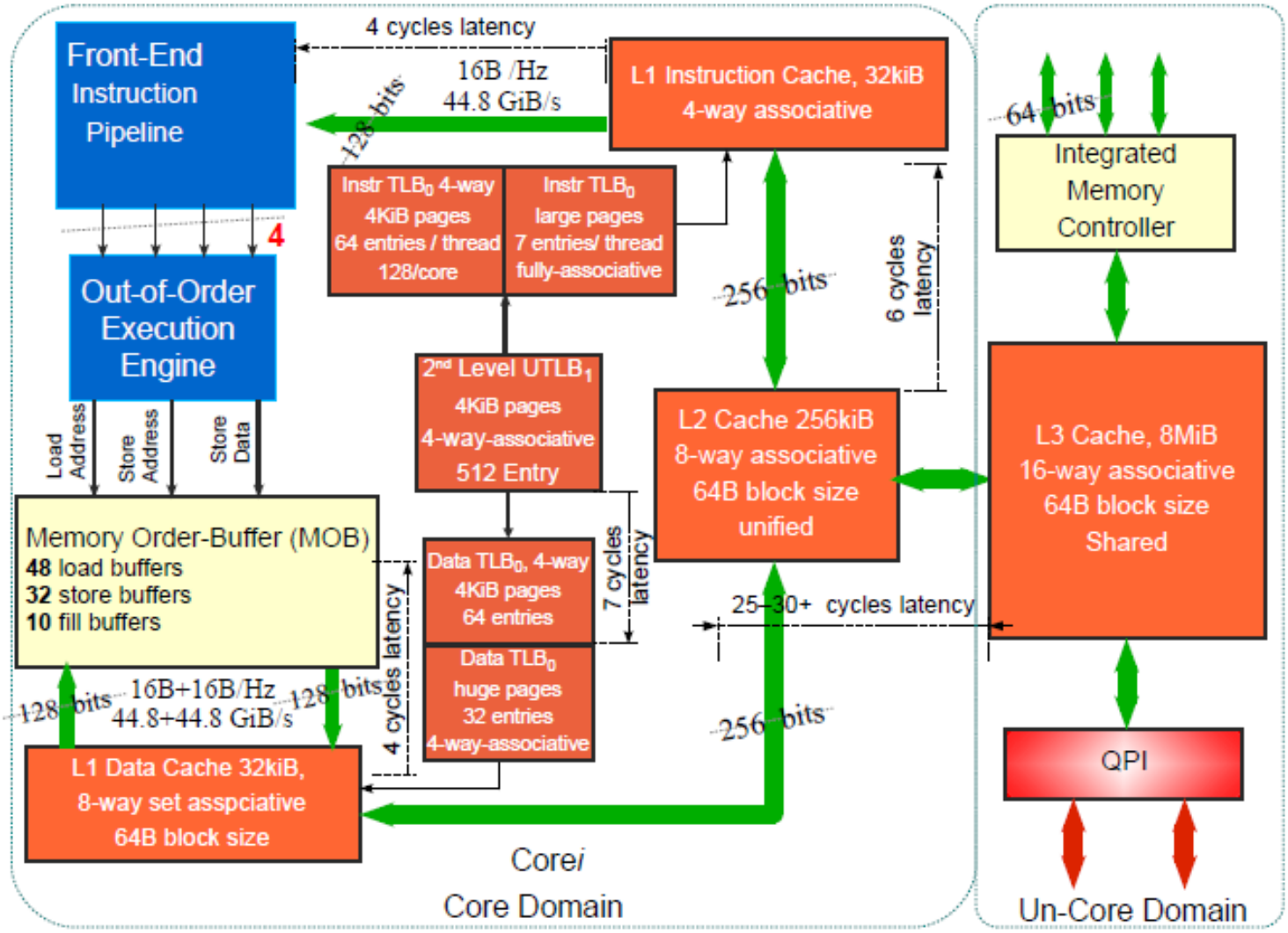


TLB – Idealized Address Translation for Data Read



- Note that there may occur miss in cache for actual data read and in TLB during preceding address translation
- If a TLB miss occurs, we must execute a *page walk*, it uses cache indexed by physical address to access tables in memory usually and that cach result in yet another one or more misses in cache memory

Intel Nehalem – Memory Subsystem Structure



Canonical Address on x86 System in 64-bit Mode

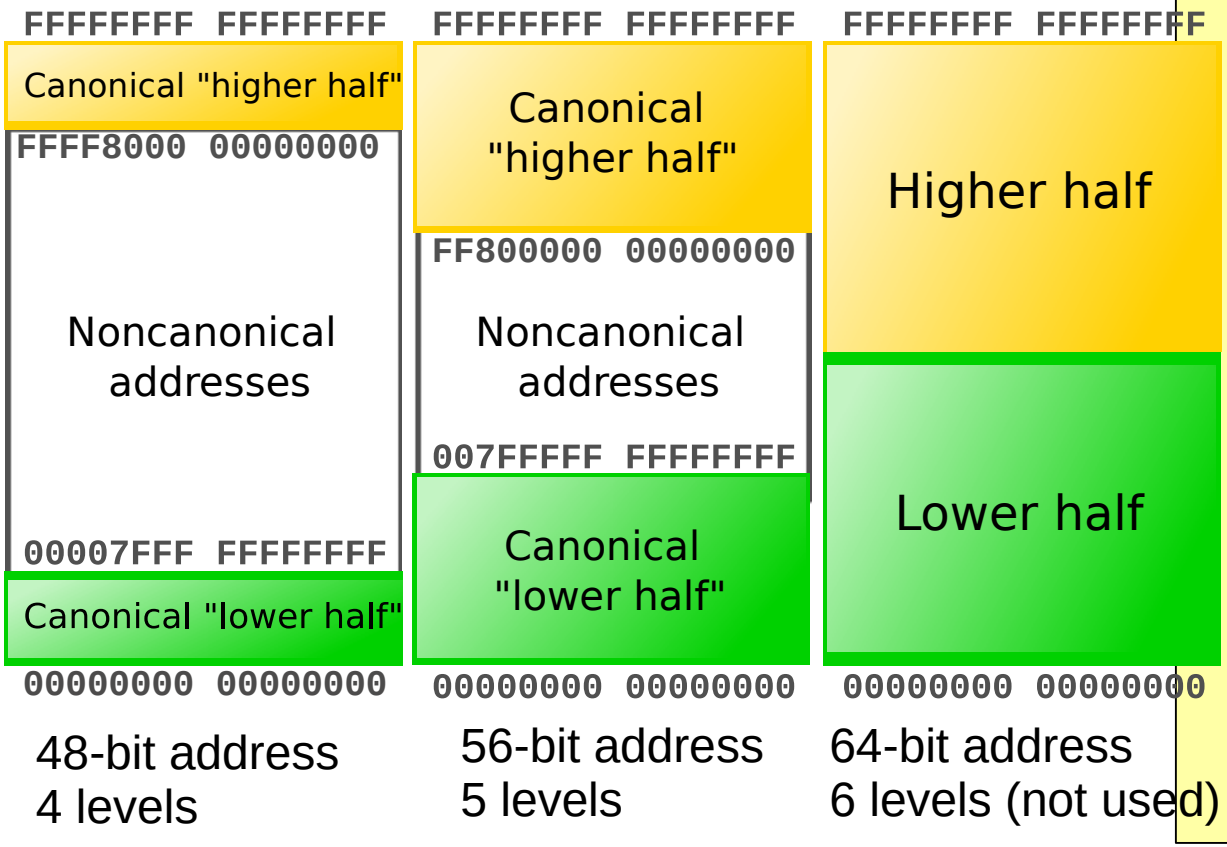
The full physical address range of 64-bits has no use today still.

Even full 64-bit virtual address is not required and causes to many levels traversal. Less bits/levels are used and space is divided to top for kernel and low for user

Bit 47 or 55 is copied to all higher (MSB) Bits

The upper space is used for operating system

Low for user applications



Page Aligned Memory Allocation – Linux

Your program may consider page size and use memory more efficiently - by aligning allocations to multiple page sizes and then reducing internal and external page fragmentation .. (allocation order, etc. See also memory pool)

```
#include <stdio.h>
#include <unistd.h>
int main(void) {
   printf(„Page Size is: %ld B.\n",
          sysconf(_SC_PAGESIZE));
   return 0;
}
```

Allocation of block aligned in memory:

```
void * memalign(size_t size, int boundary)
void * valloc(size_t size)
```

Page Aligned Memory Allocation – Windows

```
#include <stdio.h>
#include <windows.h>

int main(void) {
    SYSTEM_INFO s;
    GetSystemInfo(&s);
    printf("Size of page is: %ld B.\n",
       ns.dwPageSize);
    printf("Address space for application:
       0x%lx – 0x%lx\n",
       s.lpMinimumApplicationAddress,
       s.lpMaximumApplicationAddress);
    return 0;
}
```

Typical Sizes of Today I/D and TLB Caches Comparison

	Typical paged memory parameters	Typical TLB
Size in blocks	16 000-250 000	40-1024
Size	500-1 000 MB	0,25-16 KB
Block sizes in B	4 000-64 000	4-32
Miss penalty (clock cycles)	10 000 000 – 100 000 000	10-1 000
Miss rates	0,00001-0,0001%	0,01-2
Backing store	Pages on the disk	Page table in the main memory
Fast access location	Main memory frames	TLB

Hierarchical Memory Caveats

Some Problems to Be Aware of

- Memory coherence – definition on next slide
- Single processor (single core) systems
  - Solution: D-bit and Write-back based data transactions
  - Even in this case, consistency with DMA requited (SW or HW)
- Multiprocessing (symmetric) SMP with common and shared memory – more complicated. Solutions:
  - Common memory bus: Snooping, MESI, MOESI protocol
  - Broadcast
  - Directories
- More about these advanced topics in A4M36PAP

Coherency Definition

- Memory coherence is an issue that affects the design of computer systems in which two or more processors, cores or bus master controllers share a common area of memory.
- Intuitive definition: The memory subsystem is coherent if the value returned by each read operation is always the same as the value written by the most recent write operation to the same address.
- More formal: P – set of CPU's. $\mathbf{x_m} \in \mathbf{X}$ locations. $\forall \mathbf{p_i}, \mathbf{p_k} \in \mathbf{P}$: $\mathbf{p_i} \neq \mathbf{p_k}$. Memory system is coherent if
  1. $\mathbf{p_i}$ read after $\mathbf{p_i}$ write value **a** to $\mathbf{x_m}$ returns **a** if there is no $\mathbf{p_i}$ or $\mathbf{p_k}$ write between these read and write operations
  2. if $\mathbf{p_i}$ reads $\mathbf{x_m}$ after $\mathbf{p_k}$ write **b** to $\mathbf{x_m}$ and there is no other $\mathbf{p_i}$ or $\mathbf{p_k}$ write to $\mathbf{x_m}$ then $\mathbf{p_i}$ reads **b** if operations are separated by enough time (in other case previous value of $\mathbf{x_m}$ can be read) or architecture specified operations are inserted after write and before read.
  3. writes by multiple CPU's to the given location are serialized such than no CPU reads older value when it already read recent one

Comparison of Virtual Memory and Cache Memory

Virtual memory	Cache memory
Page	Block/cache line
Page Fault	Read/Write Miss
Page size: 512 B – 8 KB	Block size: 8 – 128 B
Fully associative	DM, N-way set associative
Victim selection: LRU	LRU/Random
Write Back	Write Thru/Write Back

- Remarks.: TLB for address translation can be fully associative, but for bigger sizes is 4-way.
- Do you understand the terms?
  - What does victim represent?
- Important: adjectives cache and virtual mean different things.

Inclusive Versus Exclusive Cache/Data Backing Store

- Mapping of contents of the main memory to the cache memory is **inclusive**, i.e. main memory location cannot be reused for other data when corresponding or updated contents is held in the cache
- If there are more cache levels it can be waste of the space to keep stale/old data in the previous cache level. Snoop cycle is required anyway. The **exclusive** mechanism is sometimes used in such situation.
- **Inclusive** mapping is the rule for secondary storage files mapped into main memory.
- But for swapping of physical contents to swap device/file exclusive or mixed approach is quite common.