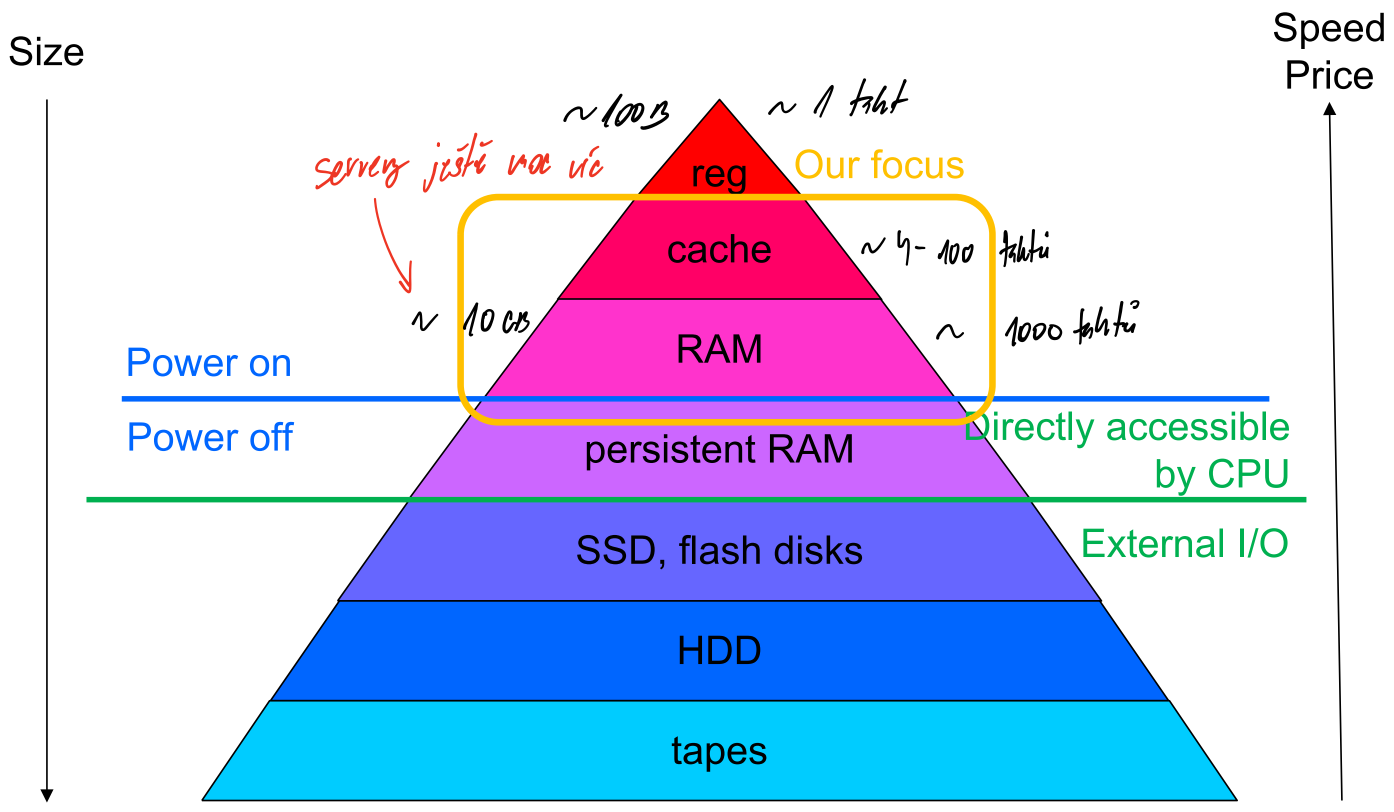
Memory

NSWI170 Computer Systems

Jakub Yaghob, Martin Kruliš

Computer memory hierarchy



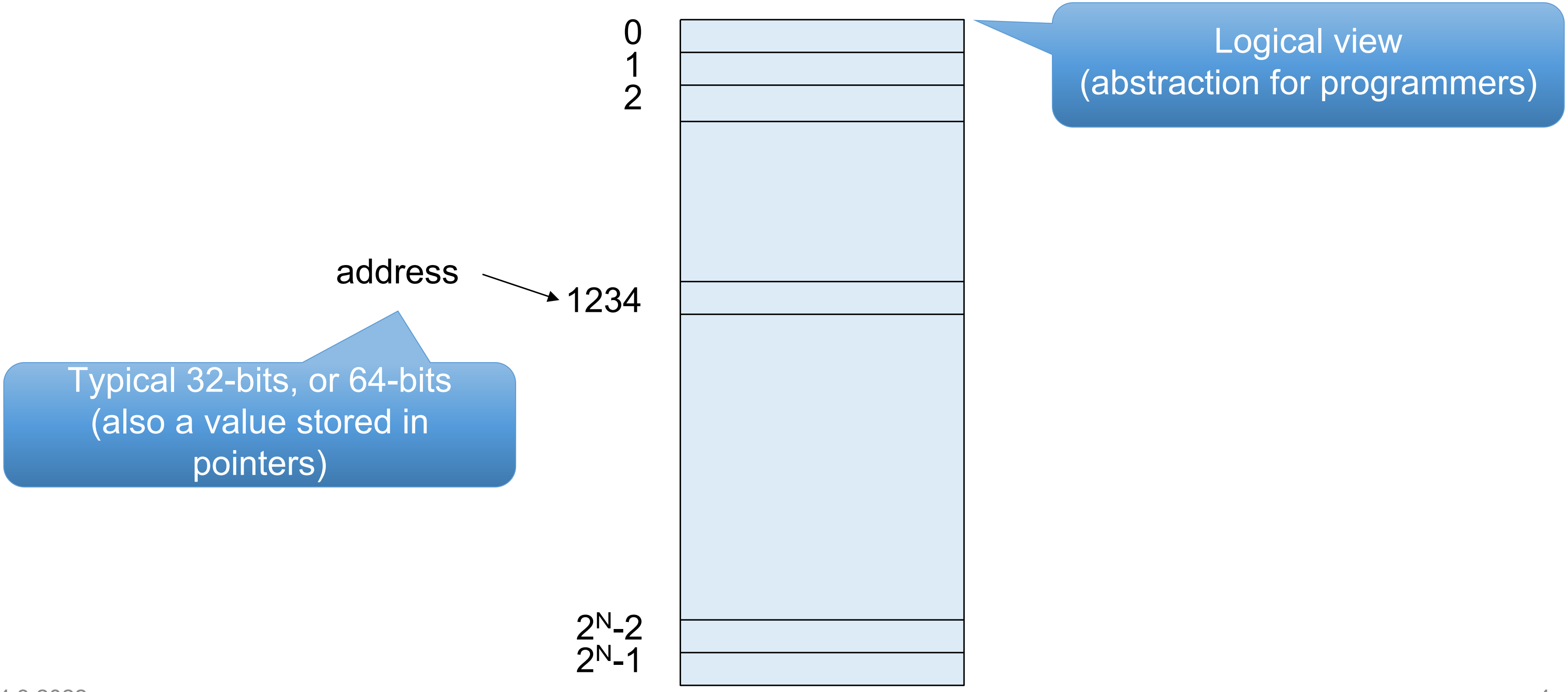
Memory

- Definition
  - Each memory organized into memory cells – bits
  - Bits are grouped into words of fixed length
    - 1, 2, 4, 8, 16, 32, 64, and 128 bits
  - Each word can be accessed by a binary address
    - N bits
    - We can store 2^N words in the memory
  - Today, the 8-bit word is used exclusively
    - Byte

This is the basis for addressing, but many architectures have “native” support only for larger words (e.g., 32 bit)

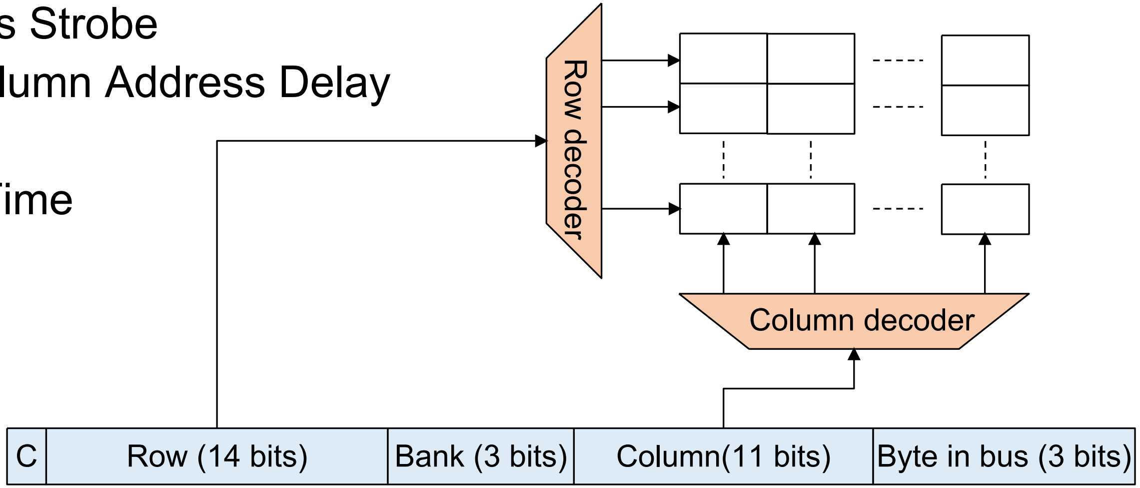
Očíslování všech slov

Memory – address space



Memory – physical view

- 2D array
  - Row x column
  - Select, access, deselect row
  - Timing → jak dlouho trvá najít správný řádek
    - CAS (tCL) – Column Access Strobe
    - tRCD – Row Address to Column Address Delay
    - tRP – Row Precharge
    - RAS (tRAS) – Row Active Time

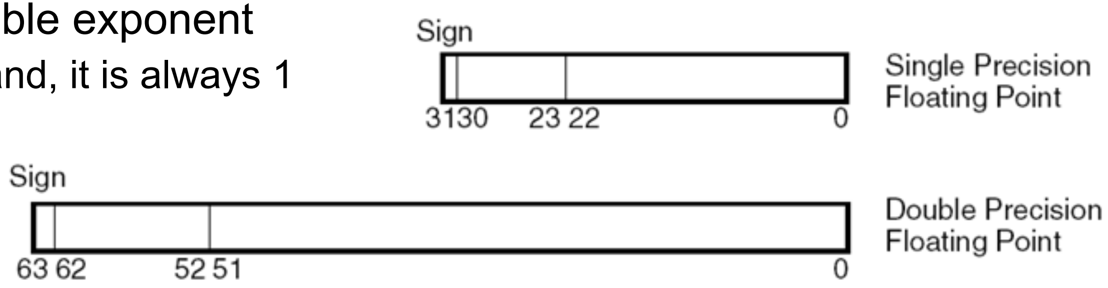


Data representation – integers

- Unsigned numbers
  - Simple binary representation of a number
  - Usual sizes
    - 1, 2, 4, 8 bytes
  - Represented range
    - $[0; 2^{N}-1]$
- Signed numbers
  - Two's complement
    - Bitwise negation + 1
    - One 0
    - Compatible with unsigned arithmetic
  - Asymmetric range
    - $[-2^{N-1};2^{N-1}-1]$

Data representation – floats

- IEEE 754 floating point numbers
  - Hidden bit convention
    - Memory representation for single-precision, double-precision
    - Use the smallest representable exponent
      - Hide leading bit of significand, it is always 1
  - Exponent
    - Bias (SP=127, DP=1023)
    - Special values
  - Value

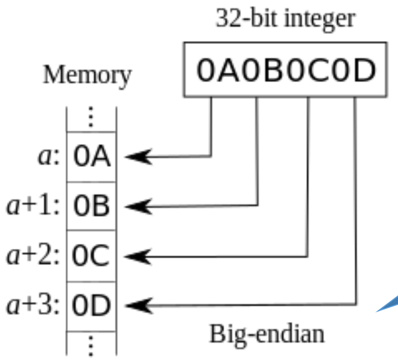


Procesor si to převádí do vlastní reprezentace

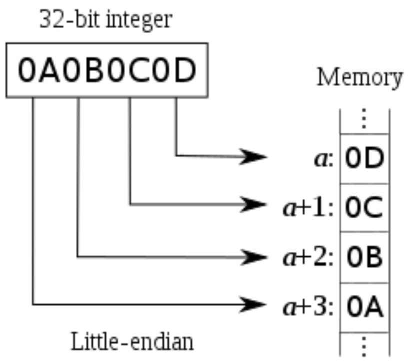
CPUs may internally use different representation (more suitable for circuits, but more redundant)

Data representation - Endianness

- How to store multi-byte numbers?
- Big endian
  - MSB first, LSB last
  - PowerPC, …
- Little endian
  - LSB first, MSB last
  - Intel (x86)
- Example
  - Store 32-bit number 0x0A0B0C0D



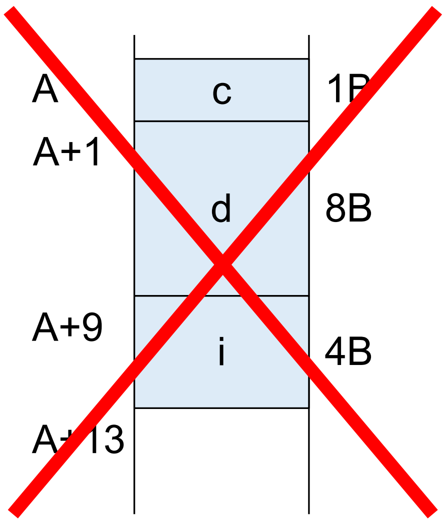
Btw. big endian is used in TCP/IP protocols

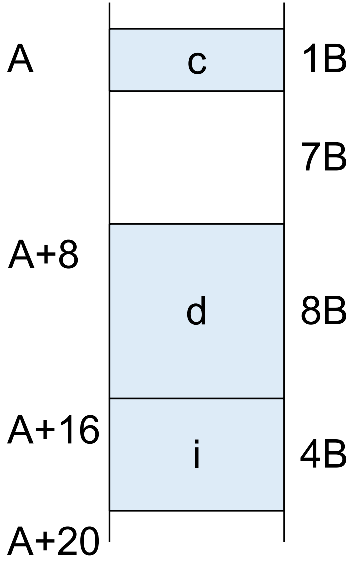


Data alignment – inner padding

- Modern CPUs require data in memory aligned to their size
  - E.g. integer (4B) must have address aligned to 4
  - Structure is aligned to largest data type available on CPU (e.g., 16B)

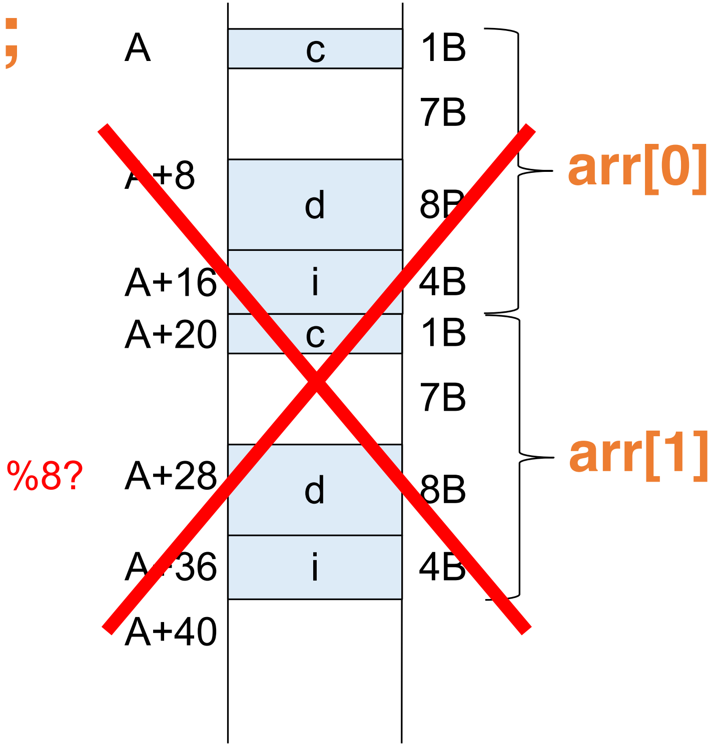
```
struct dem {
    char c;
    double d;
    int i;
};
```

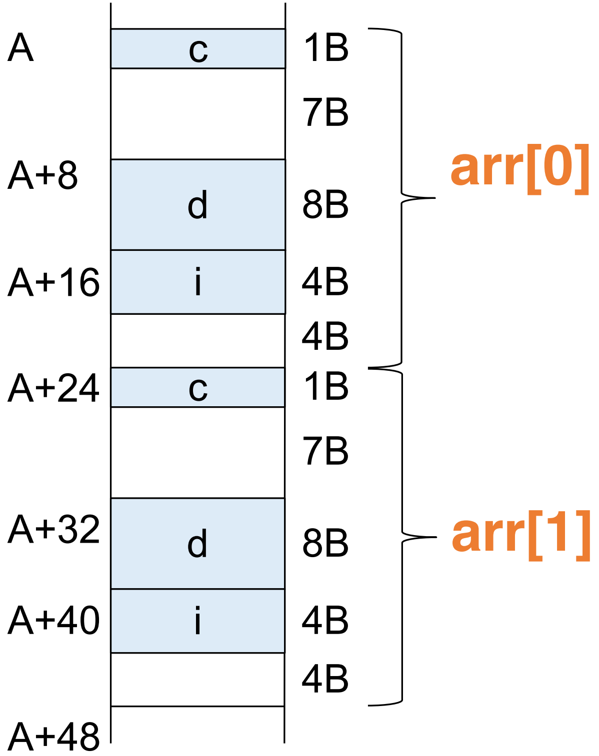




Data alignment – outer padding

dem arr[2];





Memory management

- Global variables
  - Allocated at the beginning, fixed place, released when application terminates
- Local variables, function arguments
  - Allocated on stack by shifting stack pointer
  - Stack growth, reallocation
- Dynamically allocated variables
  - Allocated explicitly by programmer (**malloc**, **new**, …)
  - Special dedicated memory block for these allocations
    - Requires special management provided by runtime in cooperation with OS

Memory allocation

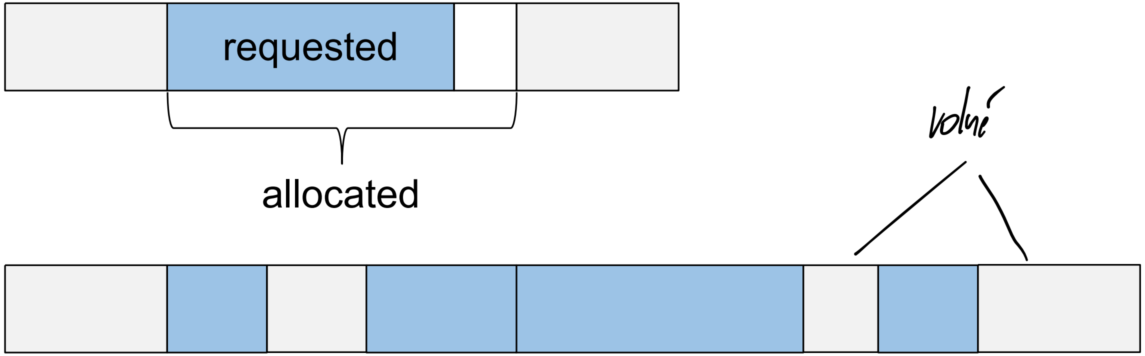
- Task
  - Locate a block of unused memory of sufficient size
  - Allocate portions from a large pool of memory
    - Heap, memory arena/pool
- Lifecycle
  - Allocate a block → „dej mi nějakou paměť!"
    - Different strategies, allocators
  - Use the block
  - Free the block → python, C#, ať systém sám usoudí
    - Explicitly, garbage collector

Fragmentation

-> S tou se nedá bojovat

- Internal -> což je zde plýtvání místem
  - Allocated more memory than needed in a block -> abych si nepamatoval využitý každý byte
- External
  - Free memory separated into small blocks and interspersed by allocated memory

s tím se bojovat dá



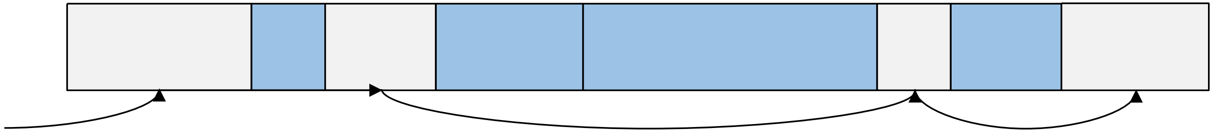
sice mám spoustu místa, ale nemusí být souvislé za sebou.

Dynamic memory allocation

- Contiguous allocation of variable size
- Free blocks evidence
  - Linked list
  - Bitmap
    - Each bit represents a block of a fixed size

Problém je režie

kdybych si pamatoval každý byte, je to 1:8, takže si radši zase pamatuji celý blok paměti



0	0	0	0	0	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0

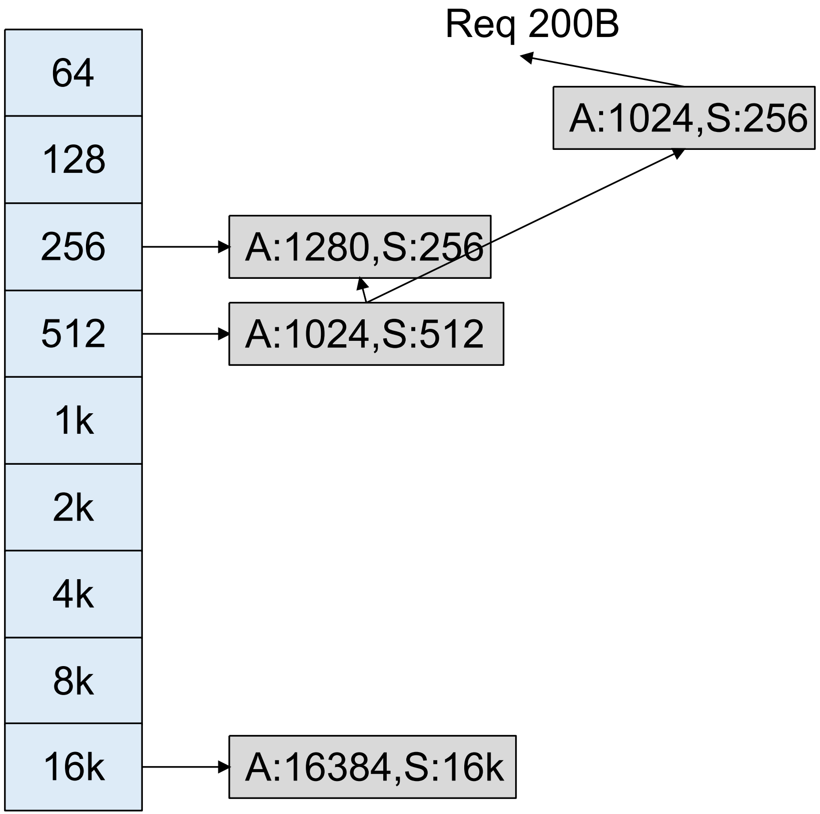
Allocation algorithms

- First fit → vyrábí to silnou externí fragmentaci
  - Start from the beginning, find the first free space big enough to accommodate required block size
  - Pros: fast, simple
    Cons: can divide larger blocks
- Next fit
  - Like the first fit, but starts from the last position
  - Pros: fast, doesn't make fragmen-tation on the start of the heap
- Best fit
  - Start from the beginning, find the smallest space big enough
  - Pros: keeps large blocks
    Cons: slower, creates many tiny blocks → jsou opravdu maličké
- Worst fit
  - Start from the beginning, find the largest space
  - Cons: divides large blocks

Buddy memory allocation

- Blocks of 2^N size
  - Address aligned to 2^N
- Find the smallest 2^N block fitting the required size
  - “List” of free blocks lists with fixed sizes 2^N
- If there are no small blocks, create them dividing larger blocks
  - Buddies
    - Find the buddy address by XORing my address with the block size
- Merge blocks back when both buddies are free
- Significant internal fragmentation

Buddy memory allocation



Cache

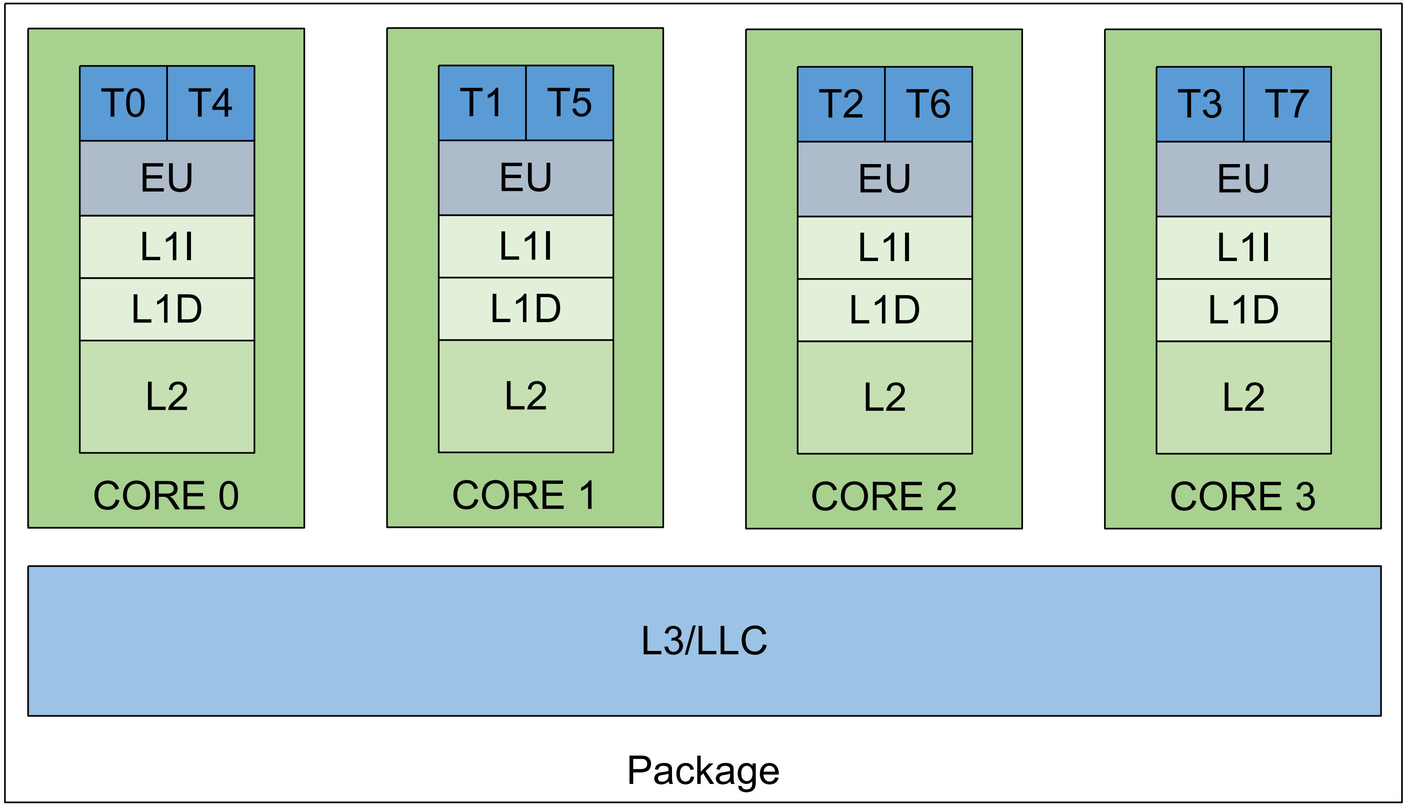
- HW or even SW
  - A structure holding data
    - Data loads/computations (for the first time) is slow/expensive
    - Future requests for that data can be served faster
    - Limited (fixed) size
  - Generic cache operation
    - Make a request for data
    - Are data placed in the cache?
    - If they are, return them, otherwise do a slow calculation/access

- Cache in CPU
  - Hides memory latency
  - Based on locality of reference
  - CPU cache operation
    - Make a request for data in the memory
    - Are data placed in the cache? Look in all levels of cache in the CPU from the fastest L1 to the slowest LLC
    - If they are, return them to the execution unit in a CPU core, otherwise do a full memory access

Cache terminology

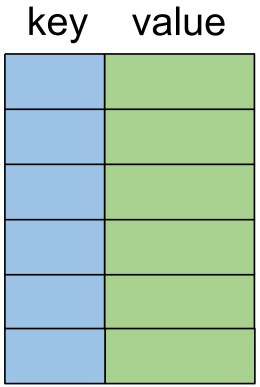
- Cache line/entry
  - Caches are organized in lines
    - Usual size is about 64B
    - Aligned
- Cache hit
  - Request served from the cache
  - Success rate around 97%
    - Common problems/algorithms
- Cache miss
  - Data not found in a cache hierarchy, do a full memory access
  - Load data from the memory to a cache line
    - Select either a free cache line or select a victim cache line
    - Store modified cache lines back to the memory
- Cache line state
  - MESI protocol

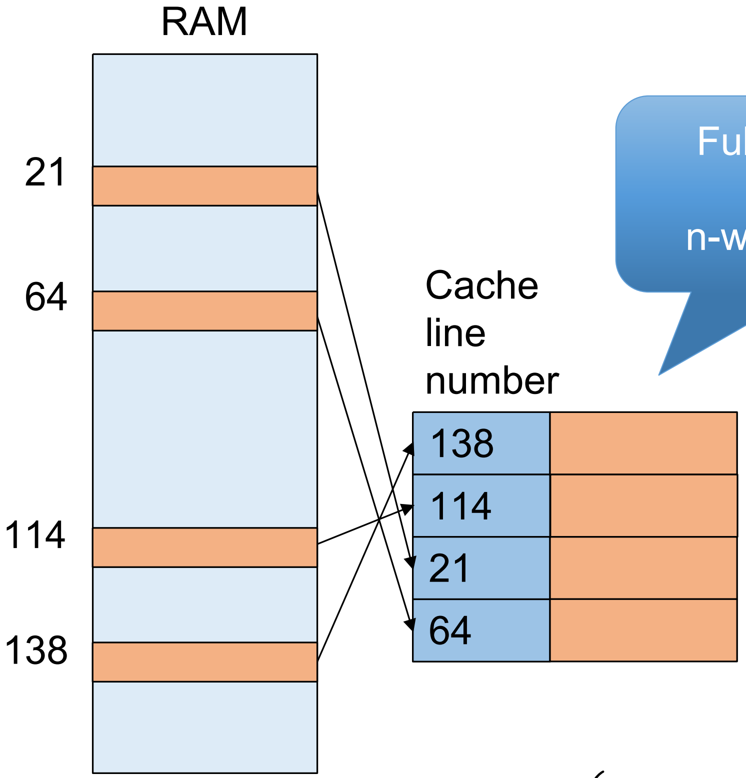
CPU caches (reminder)



Associative memory *HW implementace!*

- Associative memory
  - Very fast
  - Content based addressing
  - Used in CPU caches





Fully associative
vs.
n-way associative

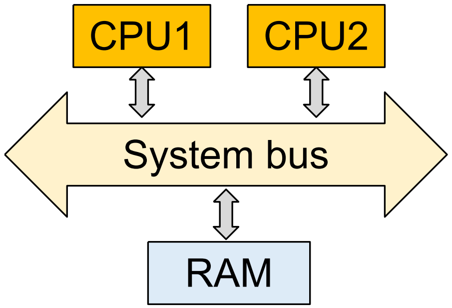
Hledám podle adresy v RAMce, jen případně dostanu adresu do cache. (na cache line)

NUMA

- Multiprocessors
  - SMP – Symmetric multiprocessing
  - NUMA – Non-uniform memory access

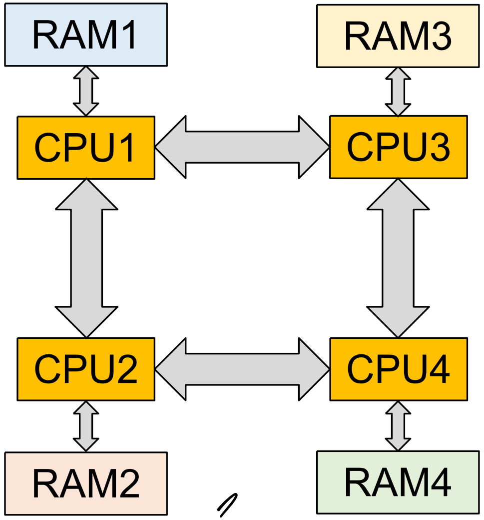
NUMA dvojice – CPU + RAM

„lokální" paměť

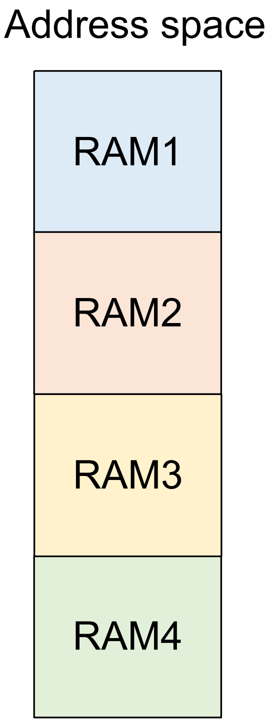


Von Neumann

– to bylo pomalejší! (>= 8 CPU se sběrnice ucpává)



Všechny další procesory



lokální sítě do jednoho

Discussion