



# Physical and Virtual Memories

---

Alexandre David

1.2.05

[adavid@cs.aau.dk](mailto:adavid@cs.aau.dk)





# Overview

---

- Key characteristics.
- Physical addressing.
- Virtual memory.



# Key Aspects

---

- Technology
  - the faster, the more expensive
  - the denser, the cheaper
  - types: volatile/non-volatile, random/sequential, read-only/read-write
    - map to examples
- Organization
  - how the bits are packed and organized



# Memory Types

---

- We are mostly concerned about RAM.
- But other types
  - in general – storage (SD, SSD, disks...)
  - sequential access memory.
- Technologies:
  - SRAM (static) – very fast & expensive
    - flip-flop maintaining a bit value – power, heat, cost
  - DRAM (dynamic) – fast and cheap
    - capacitor that needs to be refreshed
    - different latencies, cycle times etc... to accommodate the refresh.



# Terminology

---

- Important commonly used terms:
  - primary memory
    - high speed, expensive
    - relatively limited
    - in practice: RAM, volatile
  - secondary memory
    - slow, cheap
    - high capacity
    - in practice: disks/ssd, non-volatile
  - memory hierarchy
    - hierarchy of the different types register-cache-RAM...
    - keys: speed, cost, and capacity



# Different Uses

---

- Instructions – programs
  - very local, much sequential
  - fetch-and-execute cycle
- Data
  - spatial & temporal localities are different
  - fetch-and-store, more randomly accessed
- In practice it makes sense to have L1 caches for each type.



# Characteristics of RAM

---

- Bandwidth
  - bytes/s
- Latency
  - delay to get the 1<sup>st</sup> byte
  - read latency often different from write latency
- Density
  - bit cells/surface, packed in chips of x MBytes.
  - the higher, the hotter



# Memory Organization

---

- Traditional:  
processor ↔ controller ↔ memory
- Modern:  
processor ↔ memory  
(integrated controller, e.g., i5, i7, Athlon).
- The point: The processor sees a given interface that can be implemented differently
  - DDRAM
  - QDRAM
  - dual/triple channel





# Memory Access

---

- Processor ↔ controller:
  - Processor issues read/write requests.
  - Controller translates request into proper signals for the memory chips.
  - Controller replies and prepares for next request.
  - Latencies: read & write cycle times.
- Controller ↔ processor:
  - parallel interface – a bus – one wire per bit
  - front-side bus
  - width of the bus = width of cache line in practice



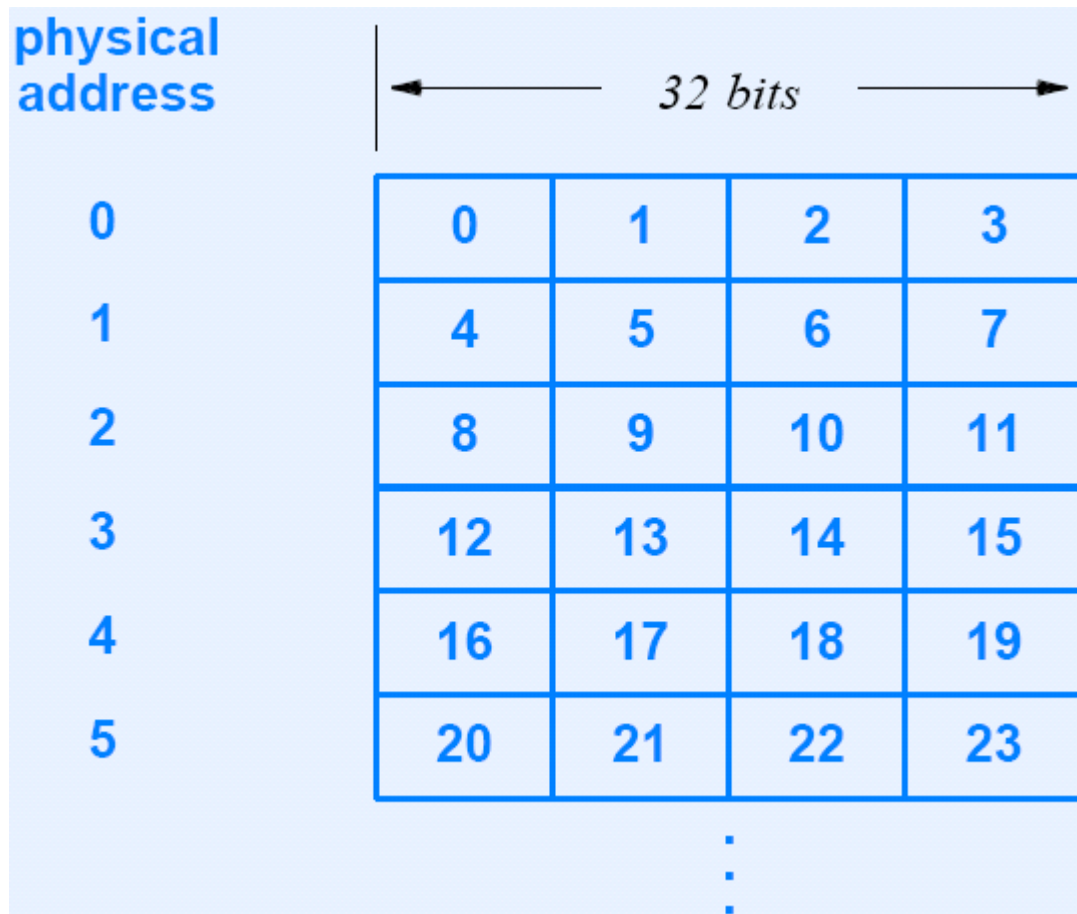
# Physical Memory Addresses

---

- Memory transfers *words*.
- Each word has a unique physical address.
- Trade-off on the size of words
  - larger, higher performance, more expensive
- Byte addressing used, more convenient
  - memory still uses word addressing, transparent
  - translation done by the controller

# Byte-to-Word Addressing

- Assume 32-bit physical words



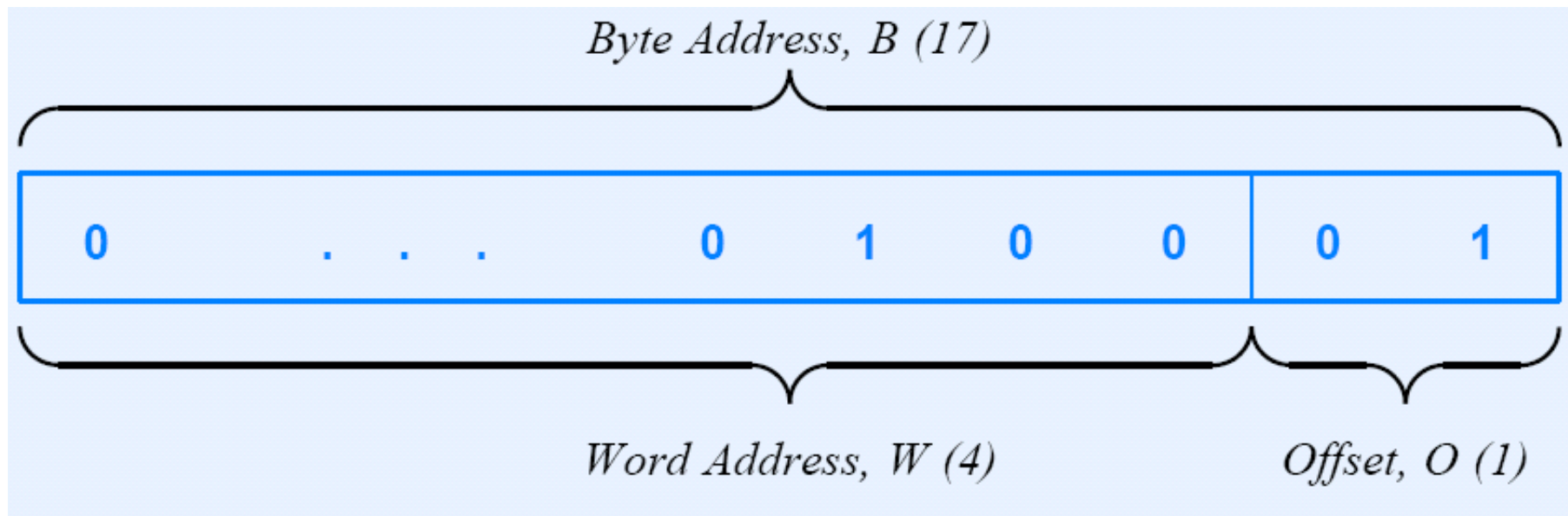
Word address:  
 $W = \text{floor}(B/N)$

Offset:  
 $O = B \bmod N$

Avoid arithmetics:  
choose powers of 2  
→ read only some  
bits instead!

# Byte-to-Word Addressing

- This principle is very important.
- Used for paging.





# Alignment

---

- A data-structure of size  $N$  (generally power of 2) is aligned in memory if its address starts at a multiple of  $N$  (power of 2).
- Ex: int, 4 bytes, generally stored at multiple of 4.
- Why? If an int is not aligned then we need to get 2 words to read it.
  - Performance reason.
  - Some architectures *require* this.



# Memory Size & Address Space

---

- Size of address (registers) limits the maximum amount of addressable memory.
- 32-bit:  $2^{32} = 4,294,967,296$  addressable bytes = address space
- Consequence of powers of 2 (size & addressing): sizes expressed in multiple of  $2^{10}$  and not 1000.



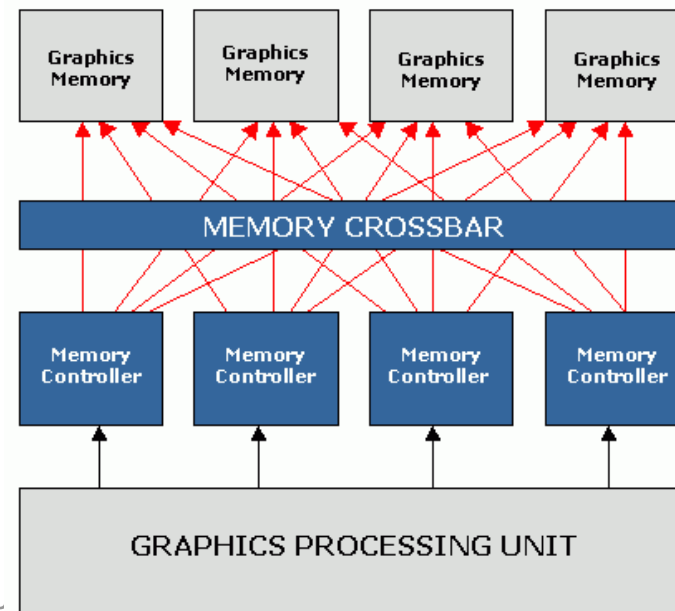
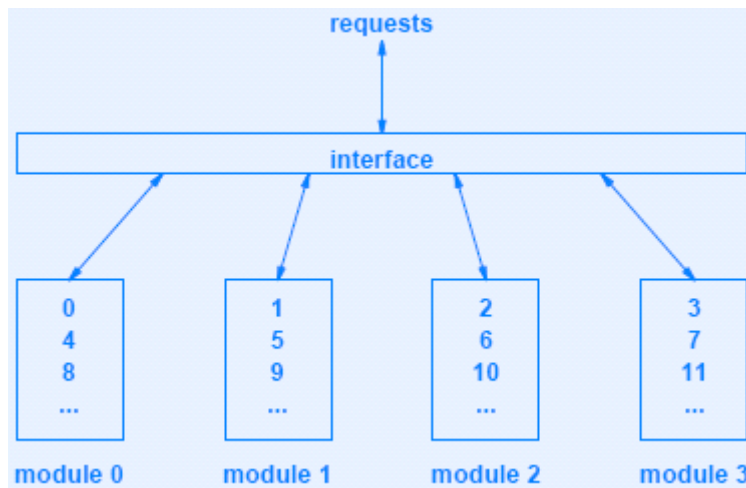
# Memory Dumps

---

- Sequence of address: content (in hexa).
  - Usually packed in 4 bytes.
  - Ex: struct node { int count; struct node\* next; } will occupy an int *followed by* a pointer.
    - 32-bit: 2x 4 bytes.
    - 64-bit: compilers will reserve 4 bytes (int) + 4 bytes (padding) + 8 bytes (pointer).
    - nodes not necessarily consecutive → more expensive than arrays (+ use indirect addressing).
- Example in exercise: disassembled program.
  - Relative address (from beginning): opcodes.
    - And assembly (text) corresponding to the opcode.

# Memory Banks & Interleaving

- Hidden by hardware on PCs.
  - Dual/triple channel technology.
  - Interleave bits on different banks.
- Can be visible on special hardware.
  - Special: Crossbar memory controllers on GPUs.







# Content Addressable Memory

---

- Concept very important – used as TLB.
- Expensive & fast:
  - parallel search
  - memory organized as a number of slots
  - implements dictionary structure: stores (key, value).



# Virtual Memory

---

- General concept:
  - mechanism hiding details of physical memory to provide a more convenient addressing.
- VM system in practice:
  - Illusion of continuous memory of a given size.
  - Hides real hardware that may be missing a bank (not all slots populated).
  - Maps virtual address to physical address.
  - Protection between processes controlled by processor modes.



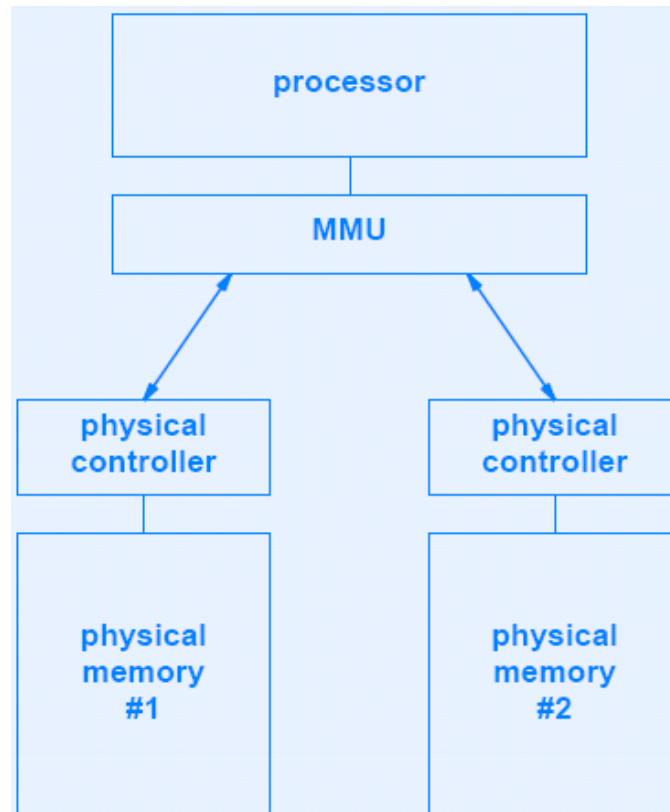
# Terminology

---

- Memory management unit (MMU)
  - hardware unit supporting VM
  - translates VM addresses to physical addresses,
  - protects accesses.
- Virtual address
  - used by processor to access data
  - translated.
- Virtual address space
  - set of all virtual addresses – usually same as physical address space,
- Virtual memory system
  - system with support for VM, including the OS

# Multiple Physical Memory Systems

- More than one memory systems, e.g., DRAM + disk (swap) in the same system.
- Mapping decided by OS, supported by MMU.





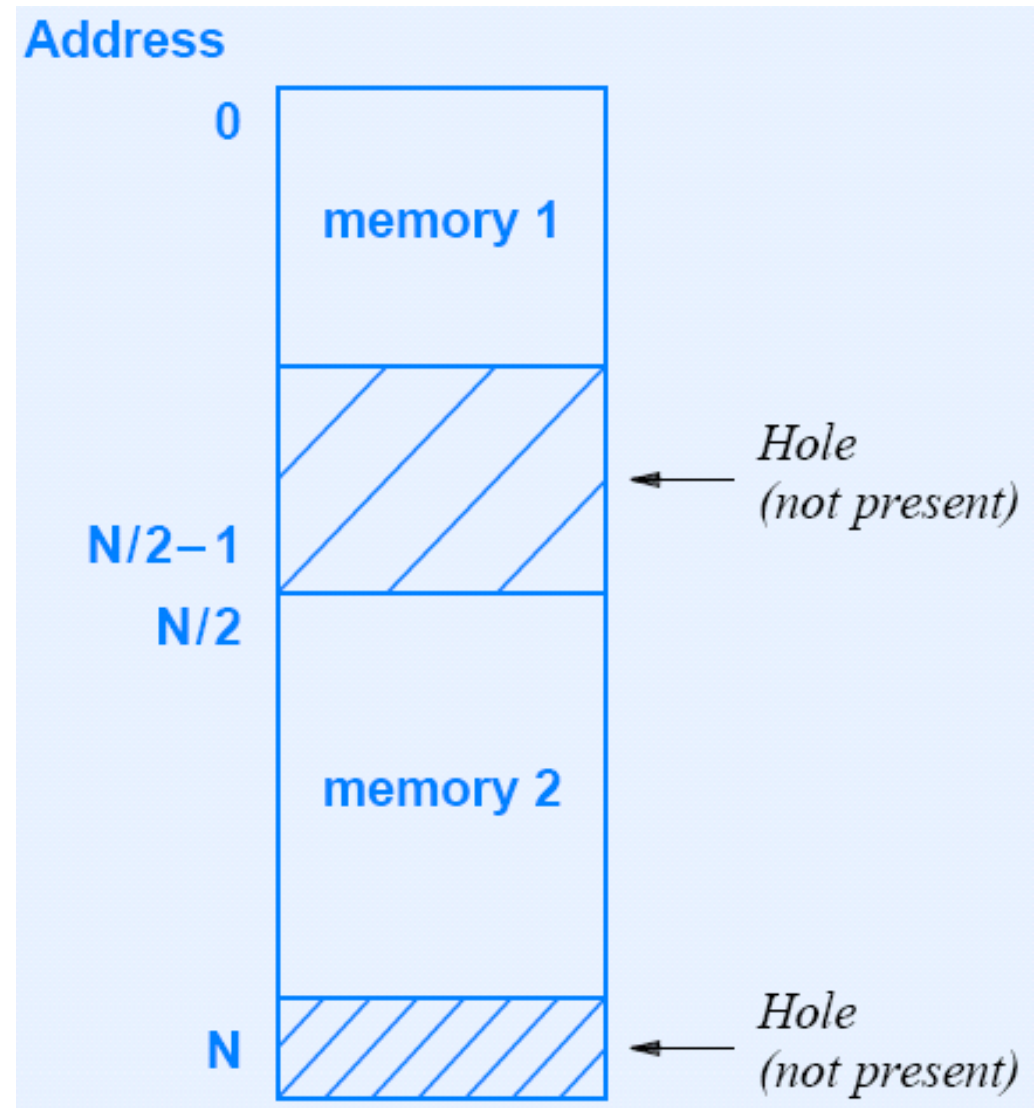
# Virtual Addressing

---

- Unique address for each location in memory.
- MMU translates from virtual space to real physical memory.
  - Mapping may be dynamic to use the hardware efficiently.
  - Illusion of continuous memory.
- Avoid arithmetic – use powers of 2.
  - $V_{\text{adr}} / N$  = read high bits of  $V_{\text{adr}}$ 
    - bank/page
  - $V_{\text{adr}} \% N$  = read low bits of  $V_{\text{adr}}$ 
    - offset

# Illusion of Continuous Memory

- Memory slots may be unoccupied.
- CPU does not know which ones until it boots.
- Visible address space (controlled by OS) may be continuous.





# Why VM?

---

- Flexibility.
- Homogeneous integration of hardware (same interface).
- Programming convenience (continuity, abstract from where the chip is connected).
- Support for multi-tasking
  - multi-programming
  - data + program protection through modes of execution

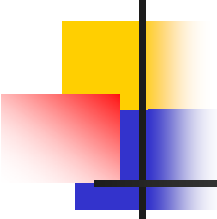


# Multiple Virtual Spaces

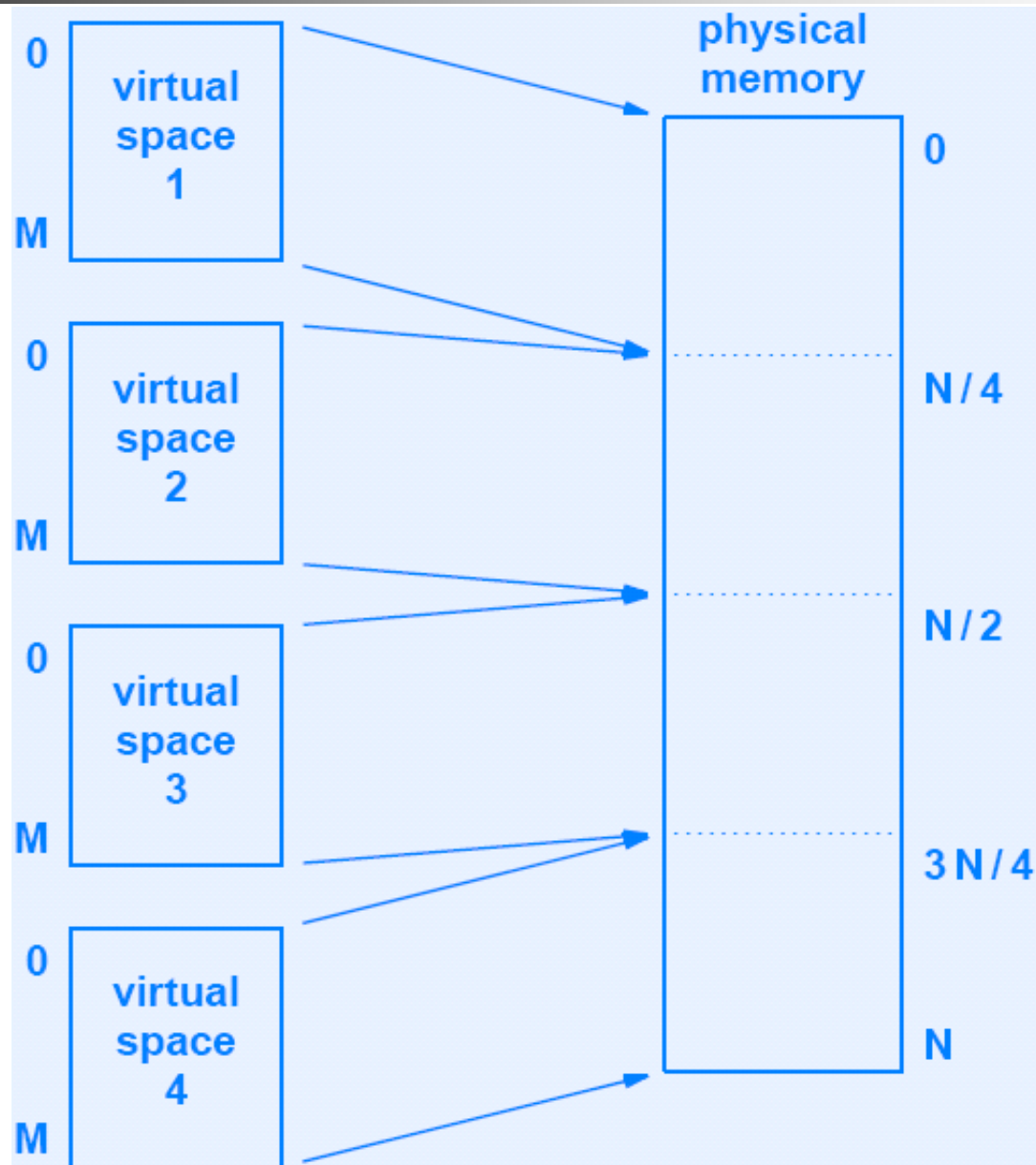
---

- Programs do not interfere with each other.
- Programs share memory resources.
  - Programs run concurrently.
- Separate virtual space to each program.
  - Programs see a subset of the (total) virtual space and has access only to that subset.





# Example





# Dynamic Address Spaces

---

- Processor configures the MMU.
  - Controlled by OS.
- Address space mapping changes.
- In practice
  - access to MMU restricted to OS
  - OS runs in real mode (sees physical address)
  - applications see VM
  - each memory access is checked by MMU
    - outside references result in errors (segfault)



# Segmentation

---

- Fine granularity mapping
  - program (and data) divided into segments – whole procedures fit in them – whole data-structures too.
  - map each segment to memory
  - give access to segments
- Key idea
  - keep segment in primary memory only when needed (resident)
  - move it to disk when not needed (swap out)
- Problems
  - not flexible, fixed sized may be too large/small
  - variable sized segments cause fragmentation.



# Demand Paging

---

- Similar to segments:
  - Divide space into pieces.
  - Keep in memory only when needed.
- Key differences:
  - Fixed size pages (4kB usually).
  - Not assigned to procedures/data-structures.
    - Ex with big array.
  - Access on-demand.

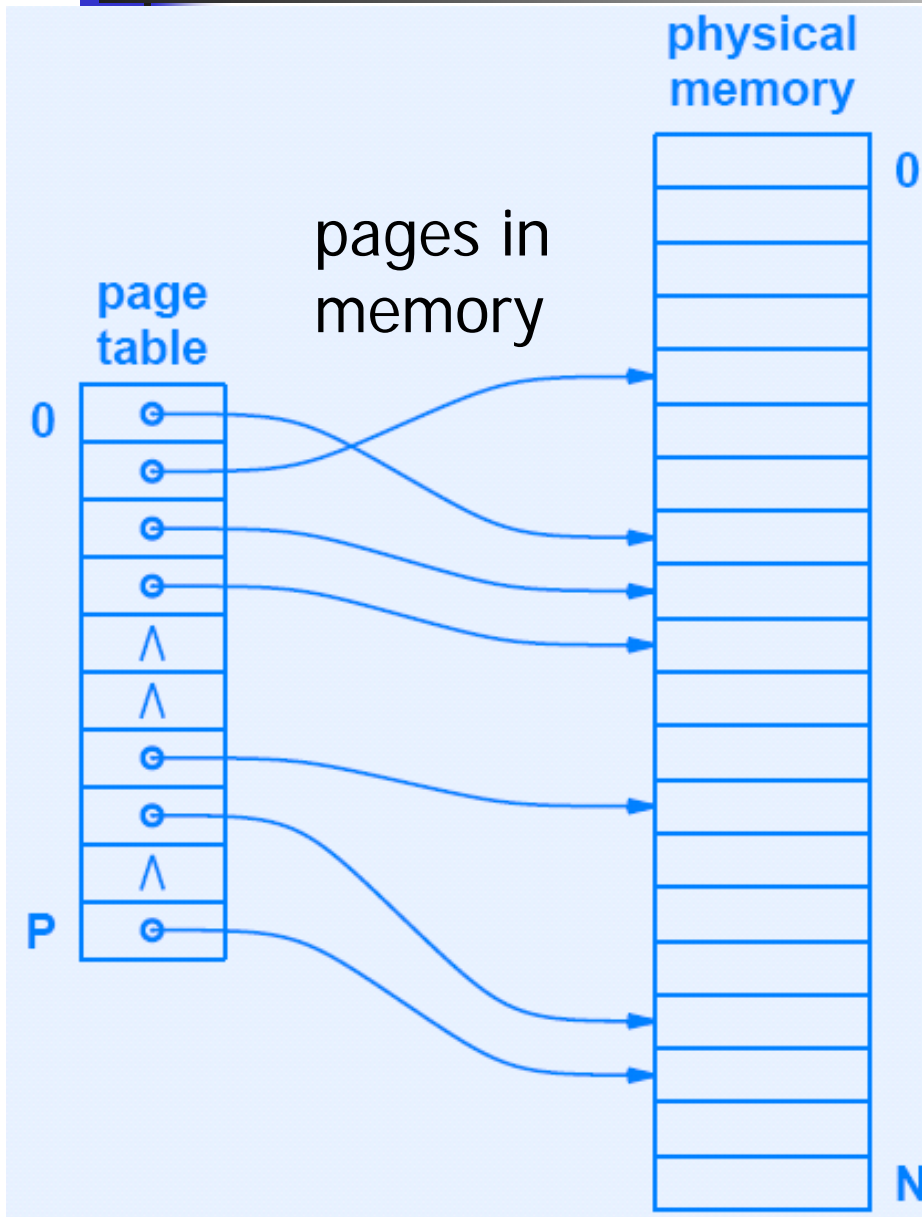


# Paging

---

- MMU
  - intercepts each memory reference,
  - translates address if the page is in memory,
  - generates a page fault otherwise and let the OS handle it.
- OS
  - manages the pages and moves them between primary & secondary memories.
- The catch:
  - Reserve some memory for the page table.
  - Very fast access to the page table.

# Paging



- If physical memory is full, *replace a page*.
  - Choose a good one – support in HW.
- Page: unit of address space.
- Frame: slot in memory corresponding to a page.
- Resident: page in memory.



# Page Table

---

- One per application.
- Frames addressed by page number.
  - Stores where the page is (RAM/disk).
- Address translation
  - given  $V_{adr}$ , find  $P_{adr}$
  - find the right page number
  - use the page number to read the page table
  - convert to address within the page
  - access the frame in memory



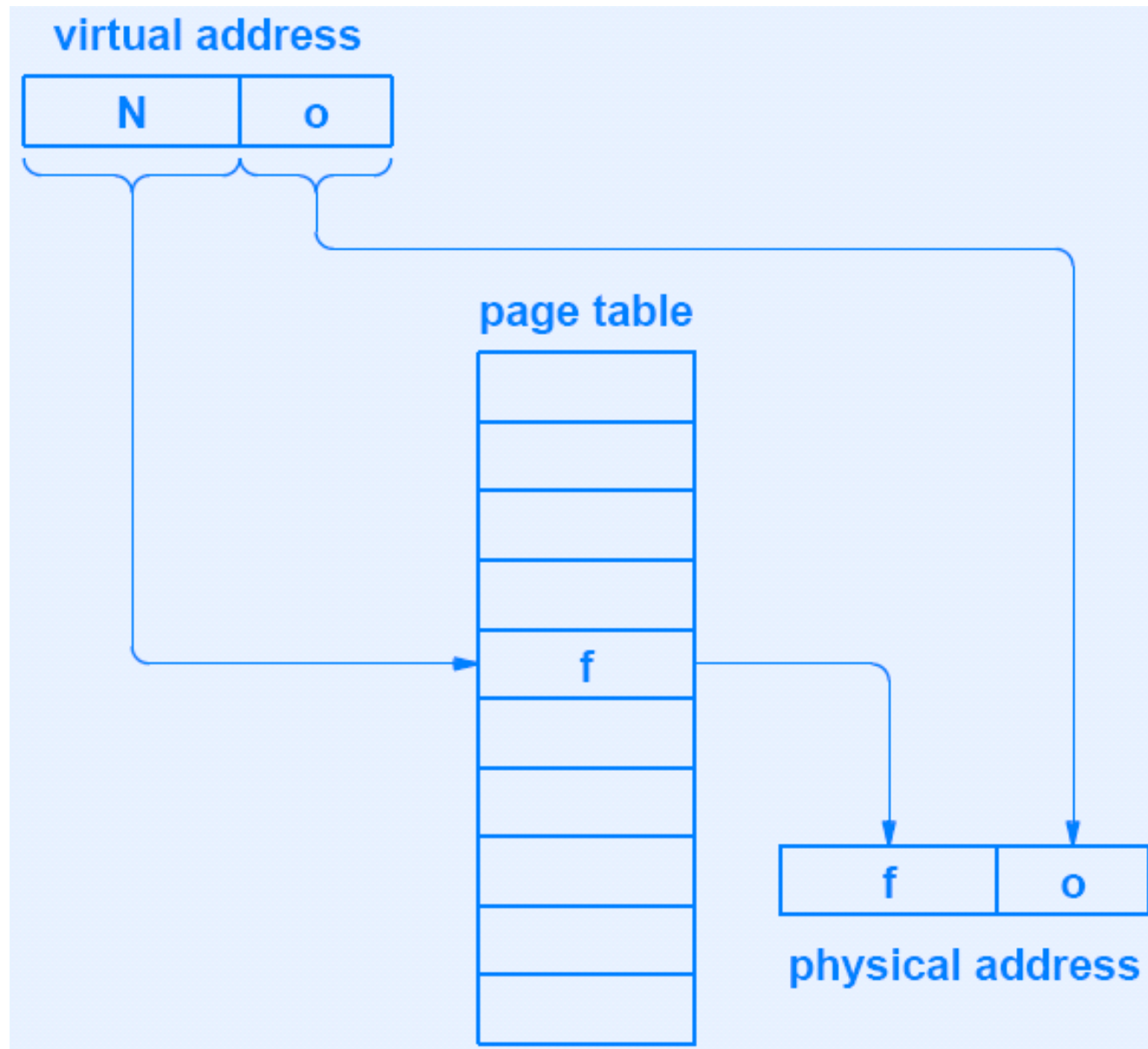
# Address Translation

---

- Translation:
  - K: page size (4kBytes in practice).
  - V: virtual address.
  - N: page number.
  - O: offset within the page.
  - $N = \text{floor}(V/K)$   
 $O = V \% K$
  - Physical address = PageTable[N] + O
  - Avoid arithmetic:  
use powers of 2 → read high/low bits only



# Address Translation by MMU





# Hardware Support

---

- Special bits:
  - presence bit: is the page resident
  - use bit: set when the page is referenced
  - modified bit: set when the content is changed
- Use:
  - Presence bit tells where the page is.
  - OS makes passes
    - can replace pages that are not used
    - reset use bit for next pass
    - needs to write to disk modified pages

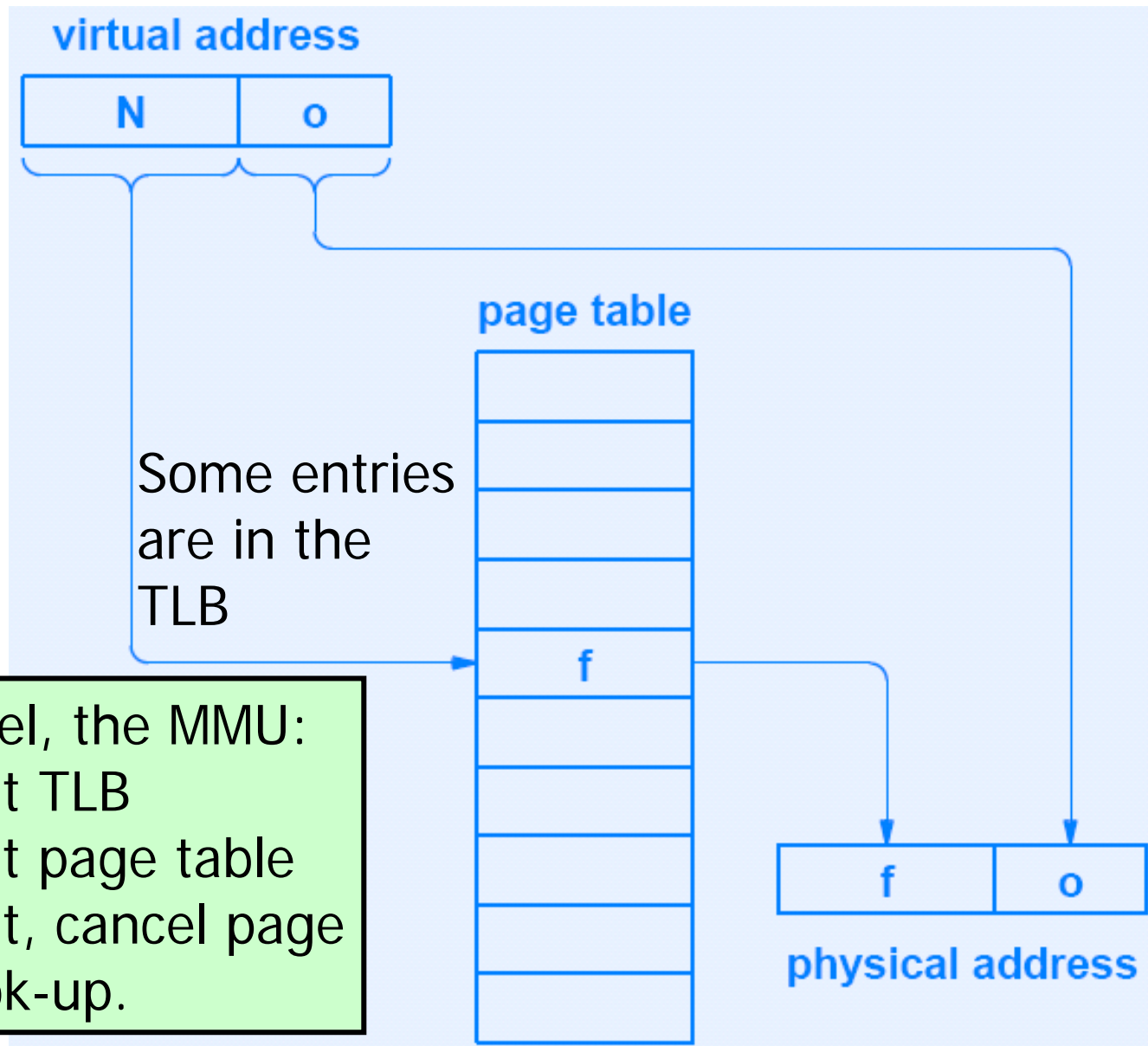


# Efficiency Issue

---

- Von Neuman architecture.
  - Memory critical for data & instruction access.
  - Fetch-and-execute all the time.
- **MUST** be optimized.
  - Use a cache!
  - TLB: translation look-aside buffer.
  - CAM based cache storing pairs (Vadr, Paddr).
  - **TLB = page table cache.**

# Address Translation by MMU



In parallel, the MMU:  
- looks at TLB  
- looks at page table  
If TLB hit, cancel page table look-up.



# Why does it work?

---

- Locality!!
  - spatial & temporal
  - data & instruction
  - within a page
  
- Notes:
  - TLB is above the caches (L1 & L2).
  - The caches are referenced by physical addresses (after TLB translation).
  - L1 & L2 caches do not cache TLB entries.