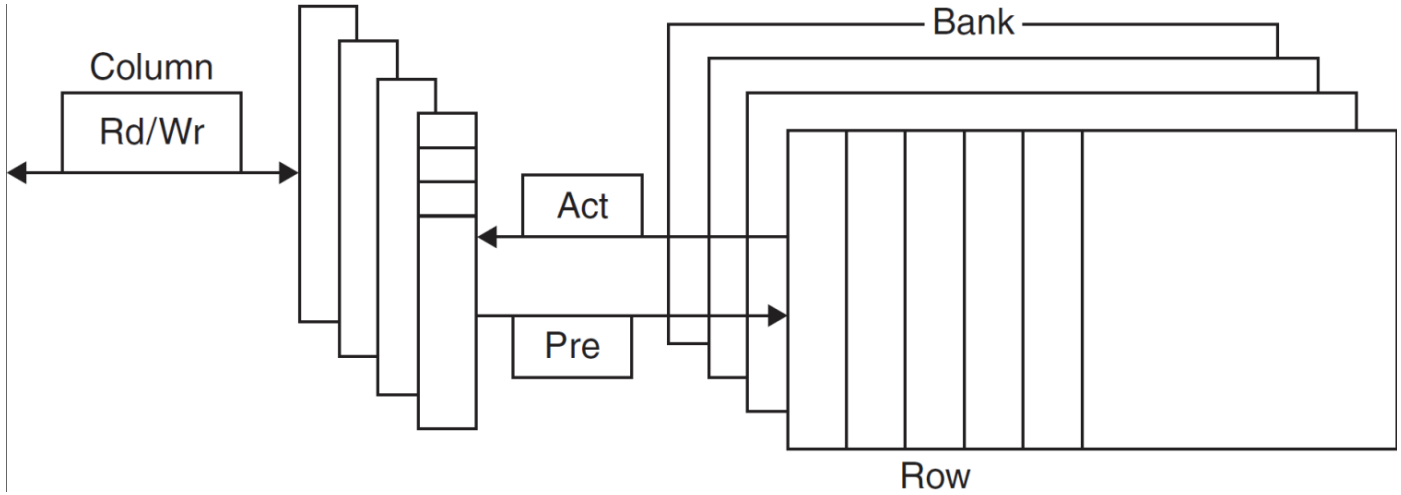
DRAM Technology

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM “row”



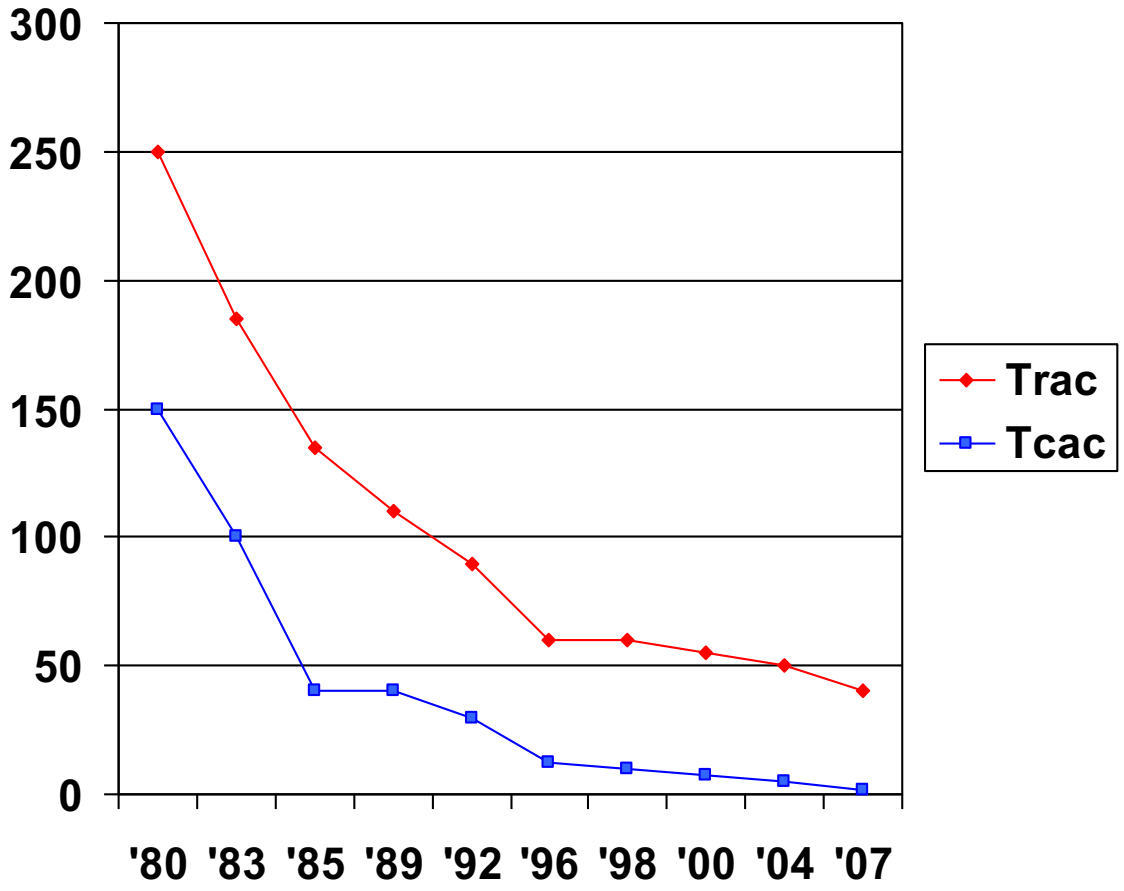
Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges

  Relative to the externally supplied clock (SDRAM)
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



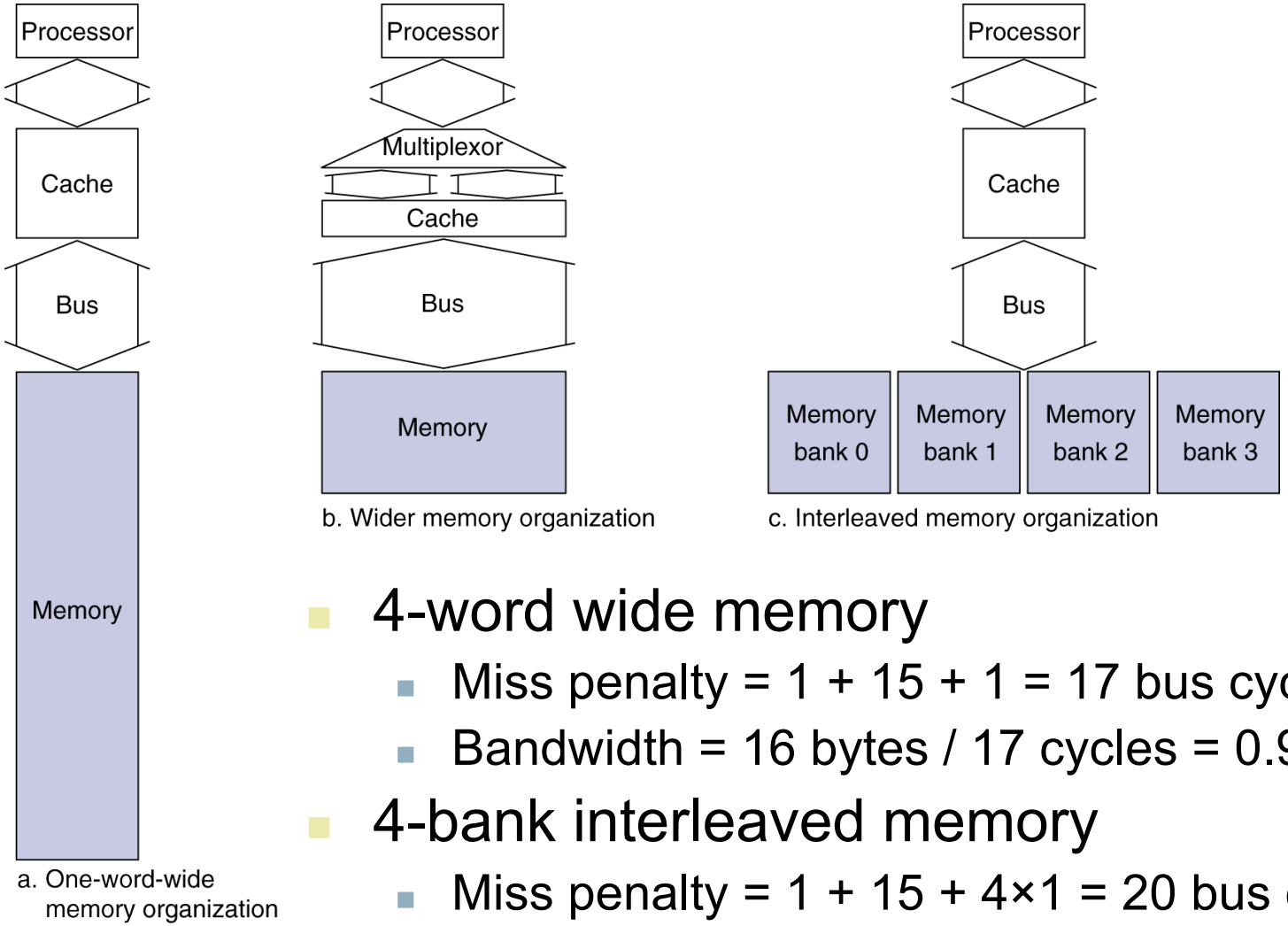
DRAM Performance Factors

- Row buffer
  - Allows several words to be read and refreshed in parallel
- Synchronous DRAM Operates in Synchrony with external clock

SDRAM

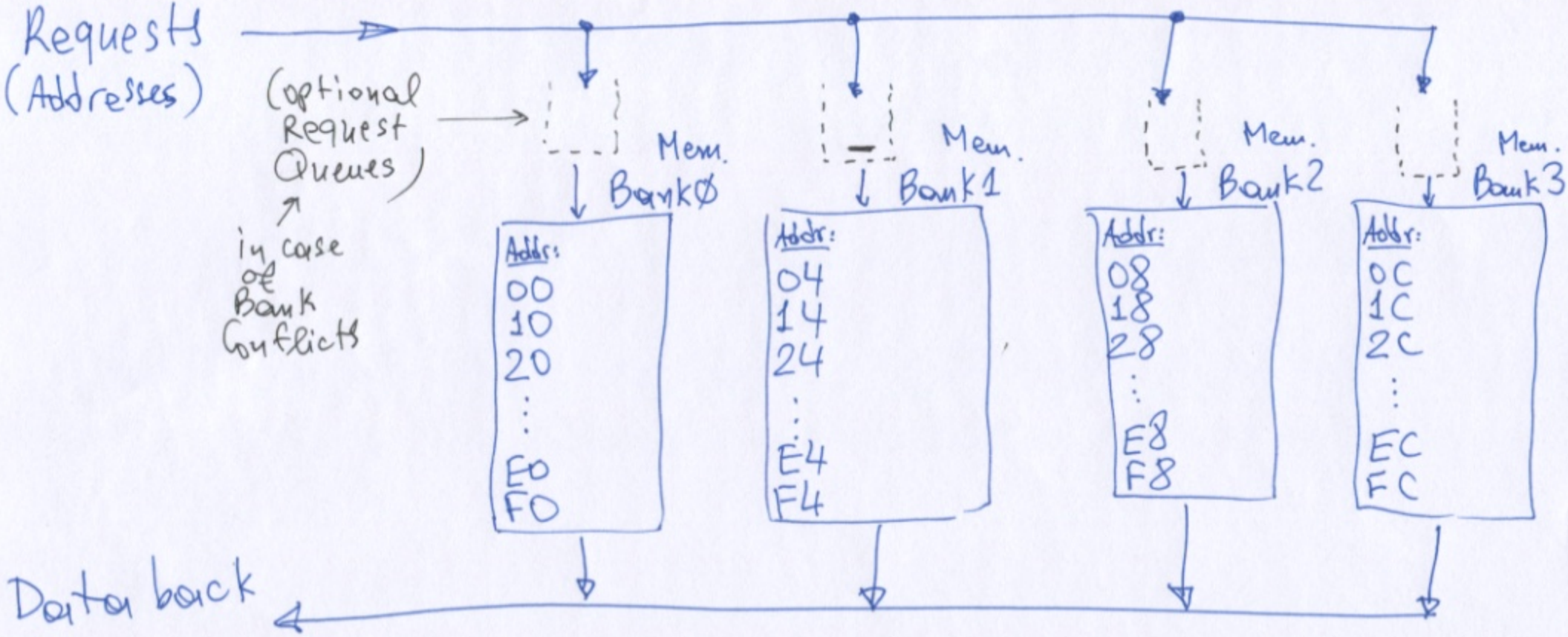
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth
- DRAM banking (Interleaves Memory Banks)
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth

Increasing Memory Bandwidth



- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

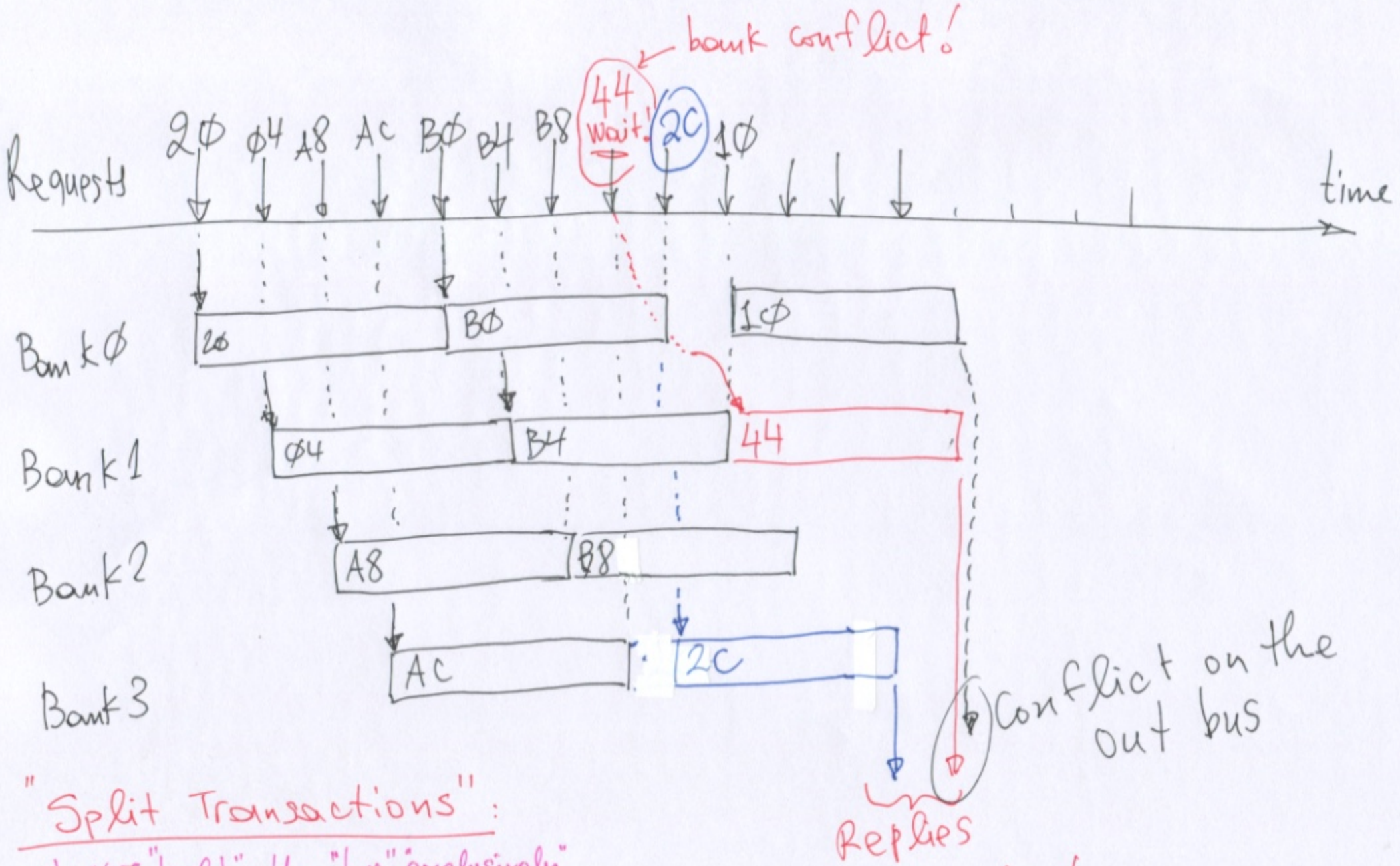
Interleaved Memory Banks (Διαφύλλωση Μνήμης)



e.g.: 100 ns per Bank Access, one new Request every 25 ns

"Split Transactions" on request/reply "bus" ... pipelining

... multiple transactions interleaved in time

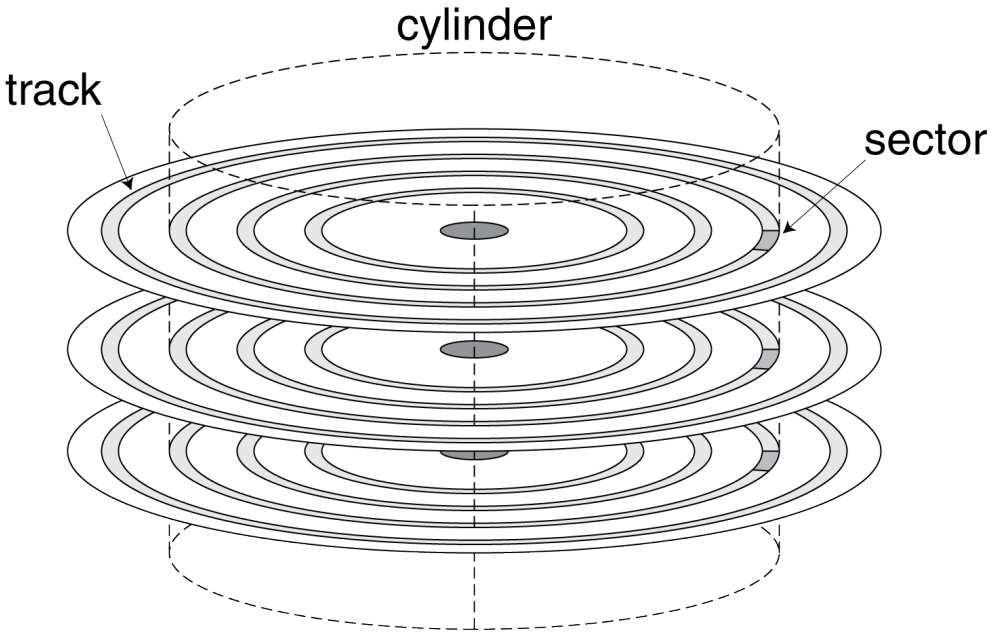


"Split Transactions":

- do NOT "hold" the "bus" "exclusively" between request and reply: release it, for other transactions
- need Transaction ID with every request and its corresponding reply, especially in case of replied out-of-order

Disk Storage

- Nonvolatile, rotating magnetic storage



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Disk Access Example

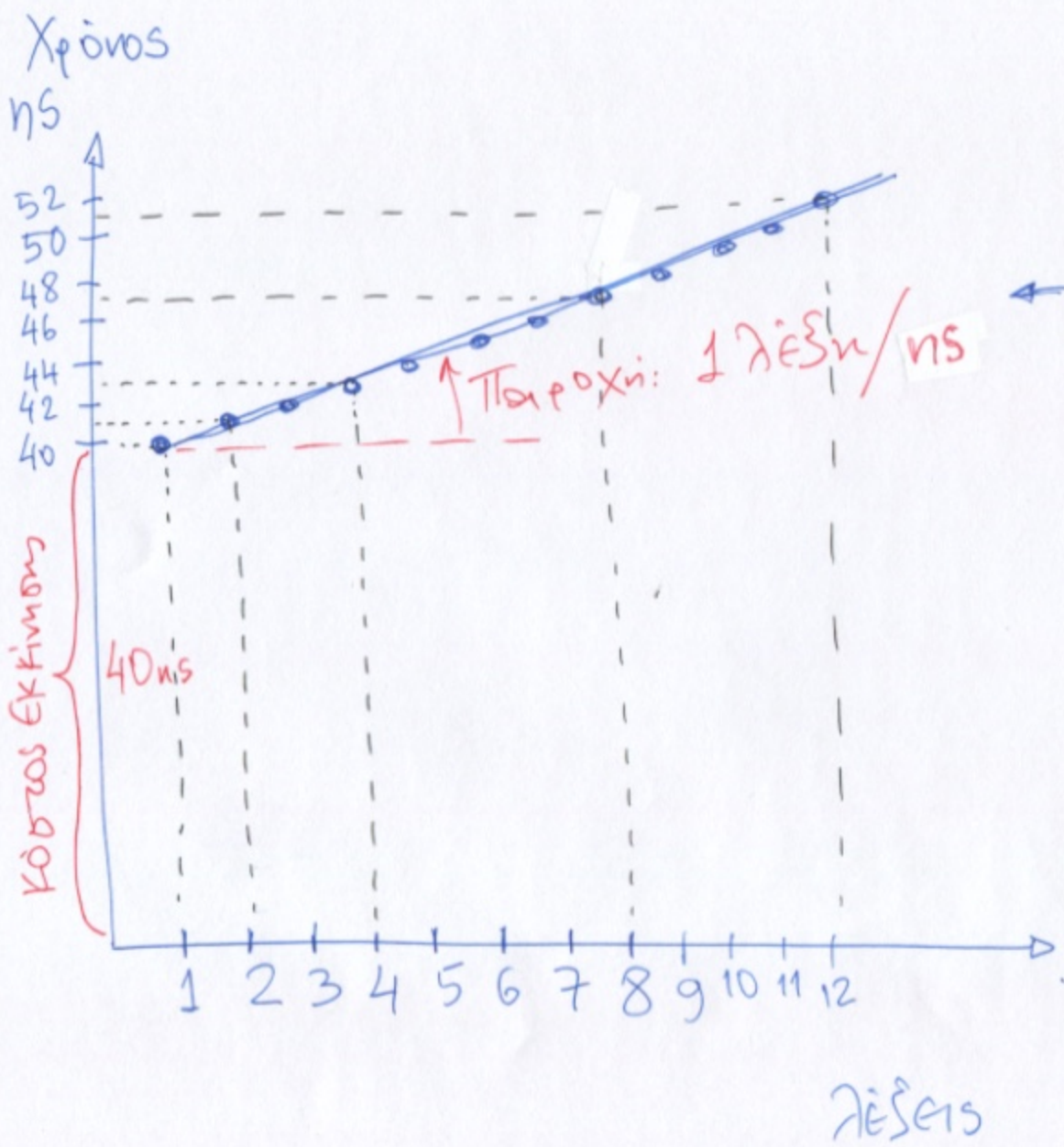
- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time
  - 4ms seek time
    + ½ / (15,000/60) = 2ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 6.2ms
- If actual average seek time is 1ms
  - Average read time = 3.2ms

Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

§13.2 Κόστος Εκκίνησης, Παροχή (startup Cost vs. Throughput)

=>Amortize cost over Large data blocks



Παραδείγματα:

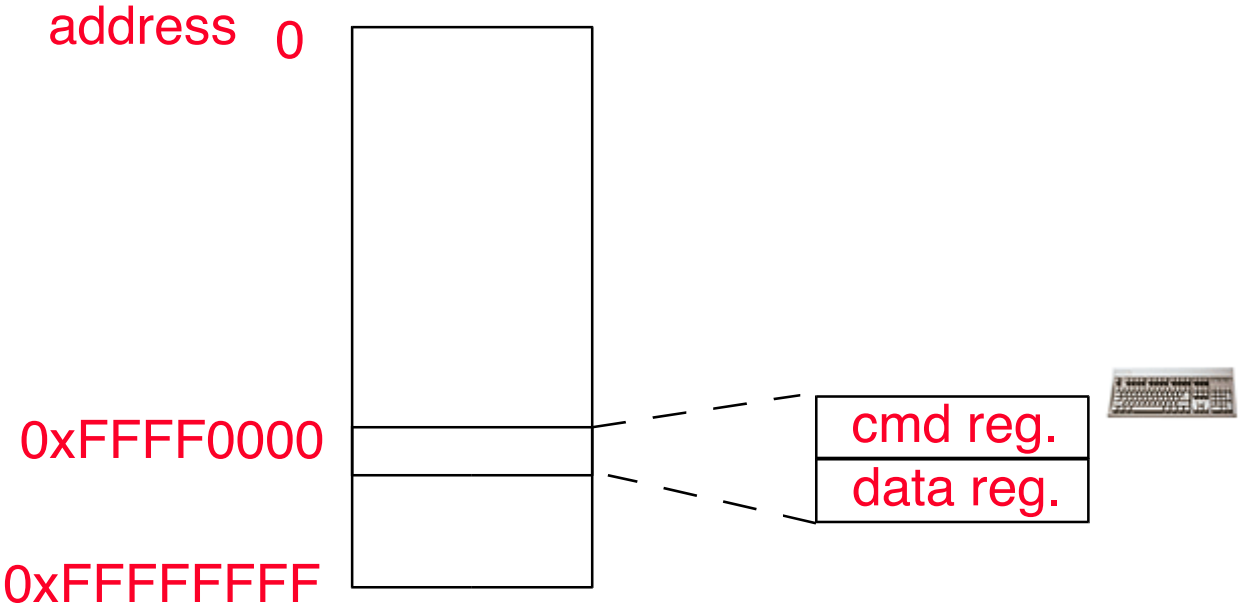
- η pipeline μας:
  - Κόστος εκκίνησης = 5 κύκλοι ρολ.
  - παροχή ≈ 1 εντολή/κύκλο
- DRAM - προσβ. σε συνεχόμενες λέξεις
  - Κόστος εκ. = (row) access time
  - παροχή - π.χ. 500MHz clock DDR timing
    ⇒ μια λέξη ανά $\frac{1}{2}T_{CK} = \frac{2ns}{2} = 1ns$
- Δίσκος π.χ. (milli 10^{-3})
  - Κόστος εκκίνησης ≈ 10 ms !
  - παροχή ≈ 100 MBy/s = 1 Byte ανά 10 ns (nano 10^{-9})
- Δίκτυα π.χ. 20km, 10 Gb/s
  - πρώτο bit: $\frac{20\,km}{200\,\frac{Mm}{s}} \approx 0.1ms = 100\mu s$!
    (ταχ. φωτός σε οπτική ίνα)
  - $10\,\frac{Gb}{s} = 1$ bit ανά 0.1 ns

Instruction Set Architecture for I/O

- Some machines have special input and output instructions
- Alternative model (used by MIPS):
  - Input: ~ reads a sequence of bytes
  - Output: ~ writes a sequence of bytes
- Memory also a sequence of bytes, so use loads for input, stores for output
  - Called "Memory Mapped Input/Output"
  - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)

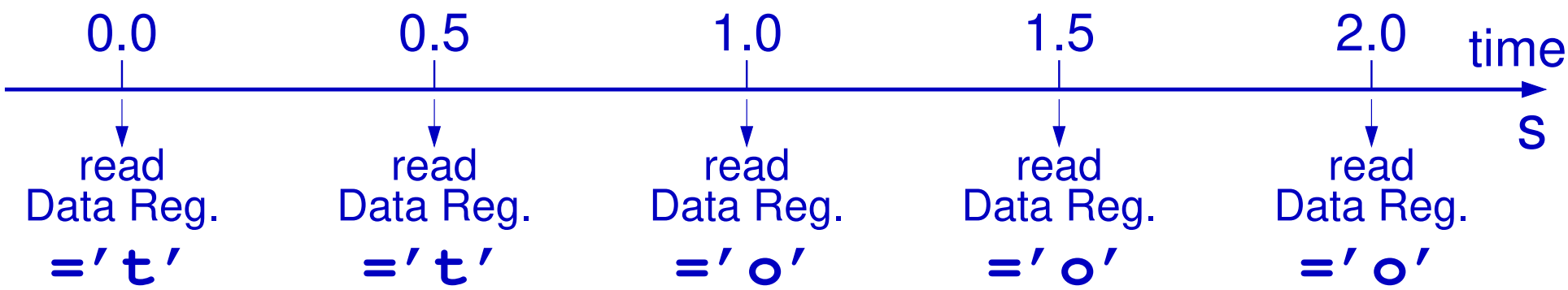
Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices

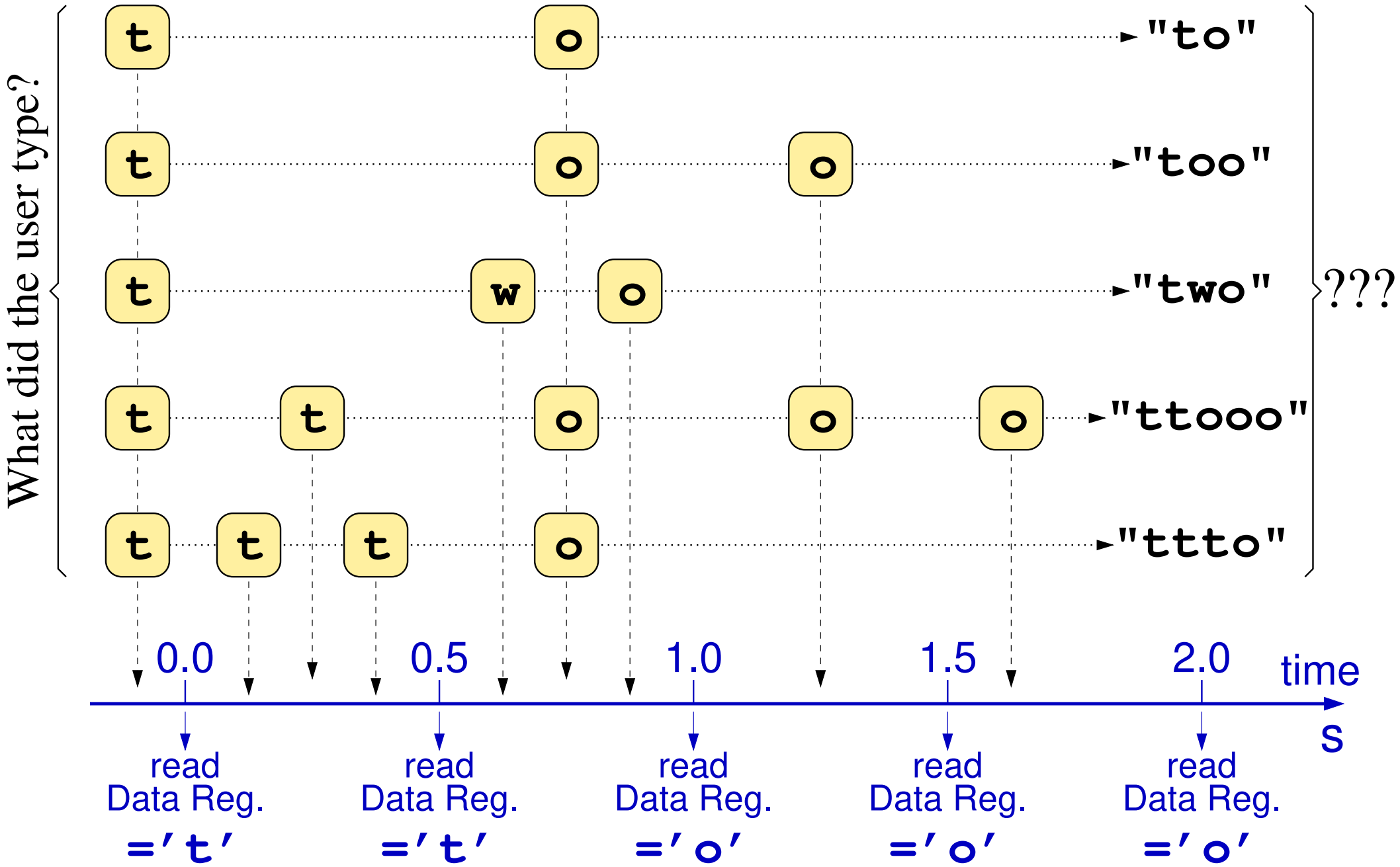


Example: keyboard... if only a Data Register:

What did the user type?



Example: keyboard... if only a Data Register:



Processor Checks Status before Acting

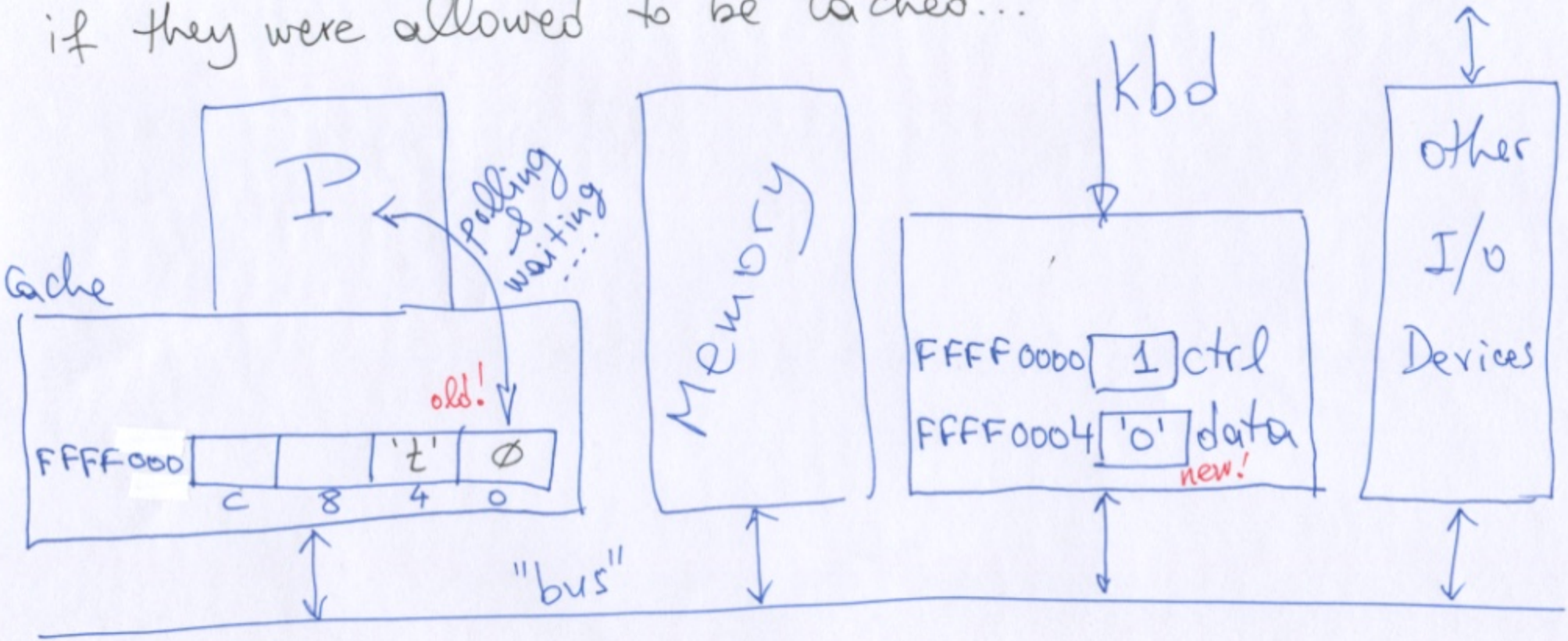
- Path to device generally has 2 registers:
  - 1 register says it's OK to read/write (I/O ready), often called Control Register
  - 1 register that contains data, often called Data Register
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg to say its OK (0 Þ 1)
- Processor then loads from (input) or writes to (output) data register
  - Load from device/Store into Data Register resets Ready bit (1 Þ 0) of Control Register

"Polling"

"Busy wait" if done continuously; else, poll multiple devices on every interrupt from the real-time clock (usu. 50-120 Hz)

I/O Address Pages must be non-cacheable!

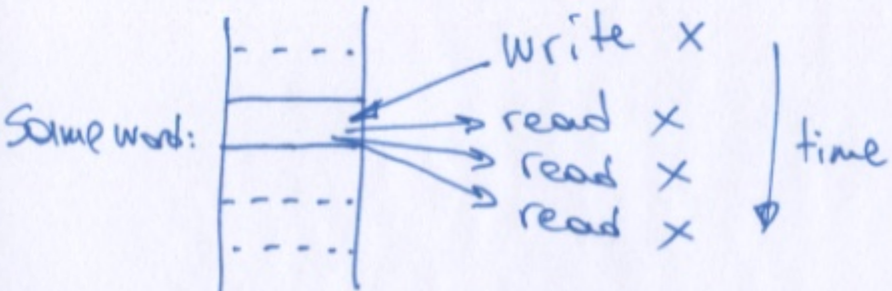
if they were allowed to be cached...



- traditional ("non-coherent"...) caching does NOT work when other devices (I/O, other proc. cores) access memory independently
- note: write-through is a "half-solution": works for output, but not for input...

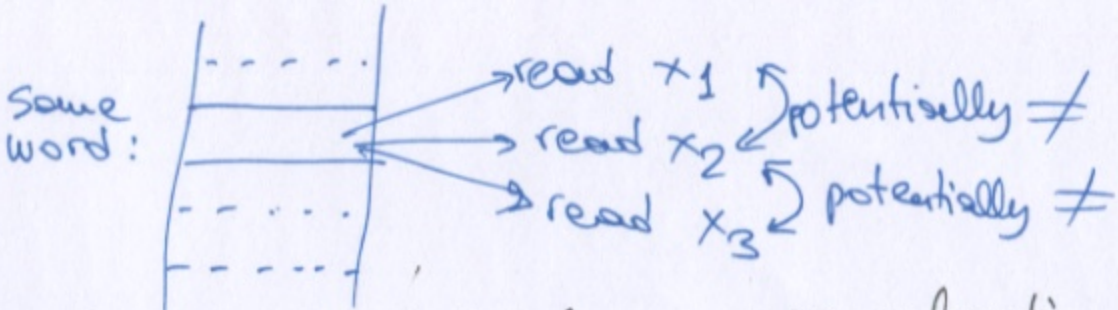
I/O/Communication Registers ≠ Normal Memory Semantics (non-shared)

Normal Memory:

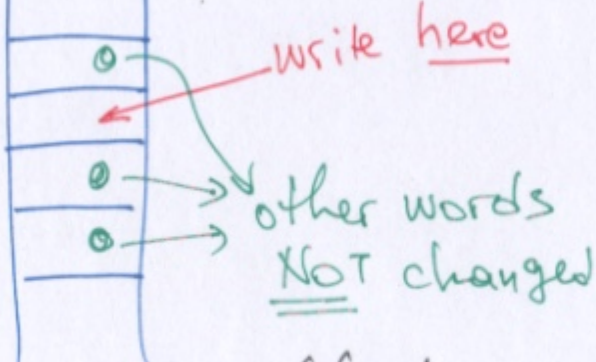


Read **always** yields the last written value

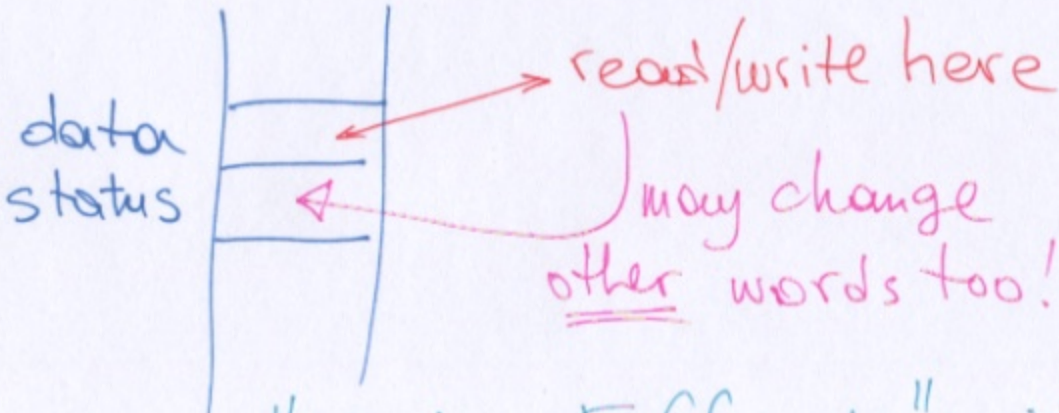
I/O/Communication Registers:



Successive reads from a same location (without any intervening writes from processor) may yield different values!



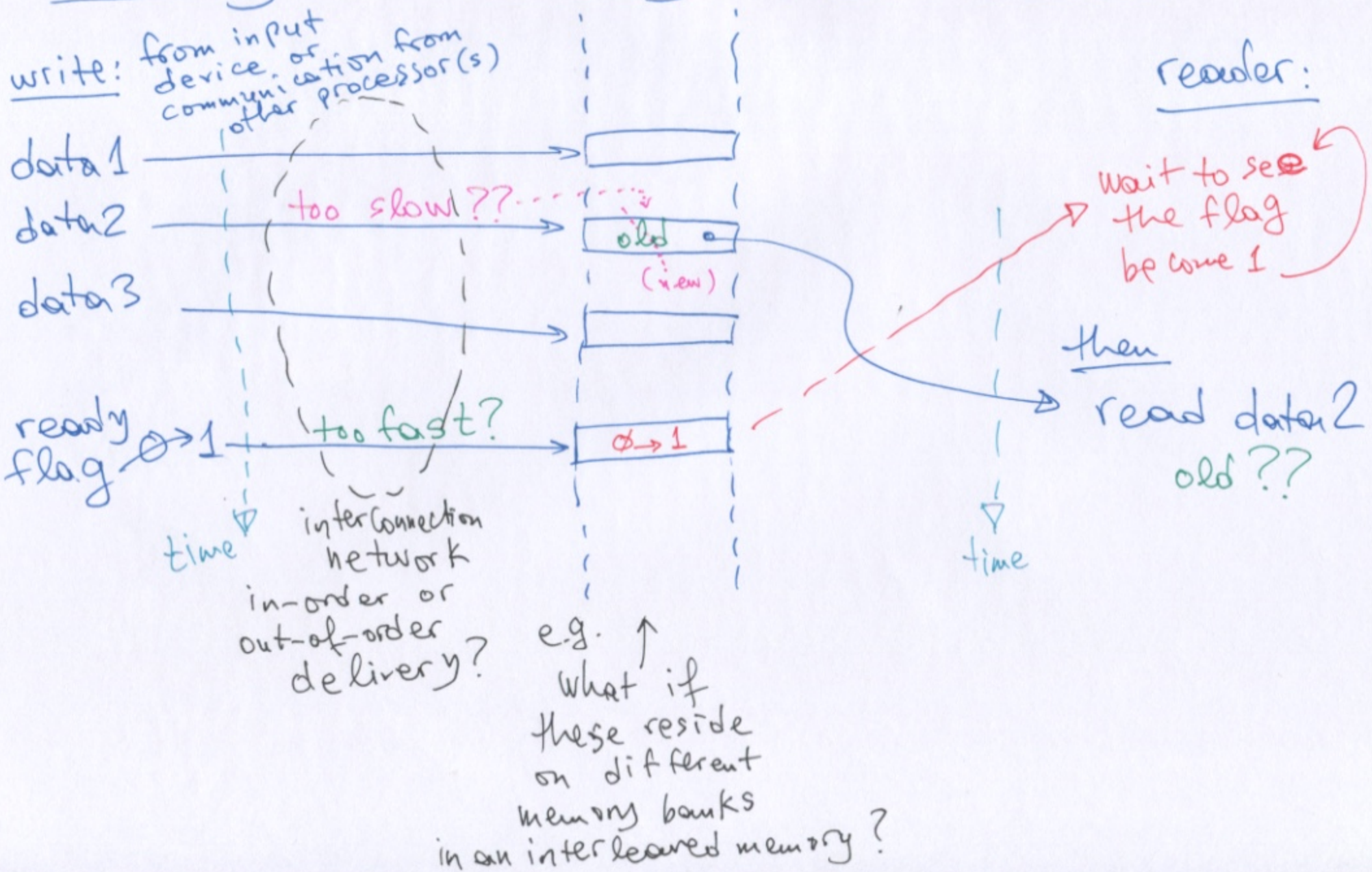
writes **only** affect the word being written



"Side-Effects"

παρενέργειες...

Memory Consistency (Συνέπεια Μνήμης)



What is the alternative to polling?

- Wasteful to have processor spend most of its time “spin–waiting” for I/O to be ready
- Wish we could have an unplanned procedure call that would be invoked only when I/O device is ready
- Solution: use exception mechanism to help I/O. Interrupt program when I/O ready, return when done with data transfer

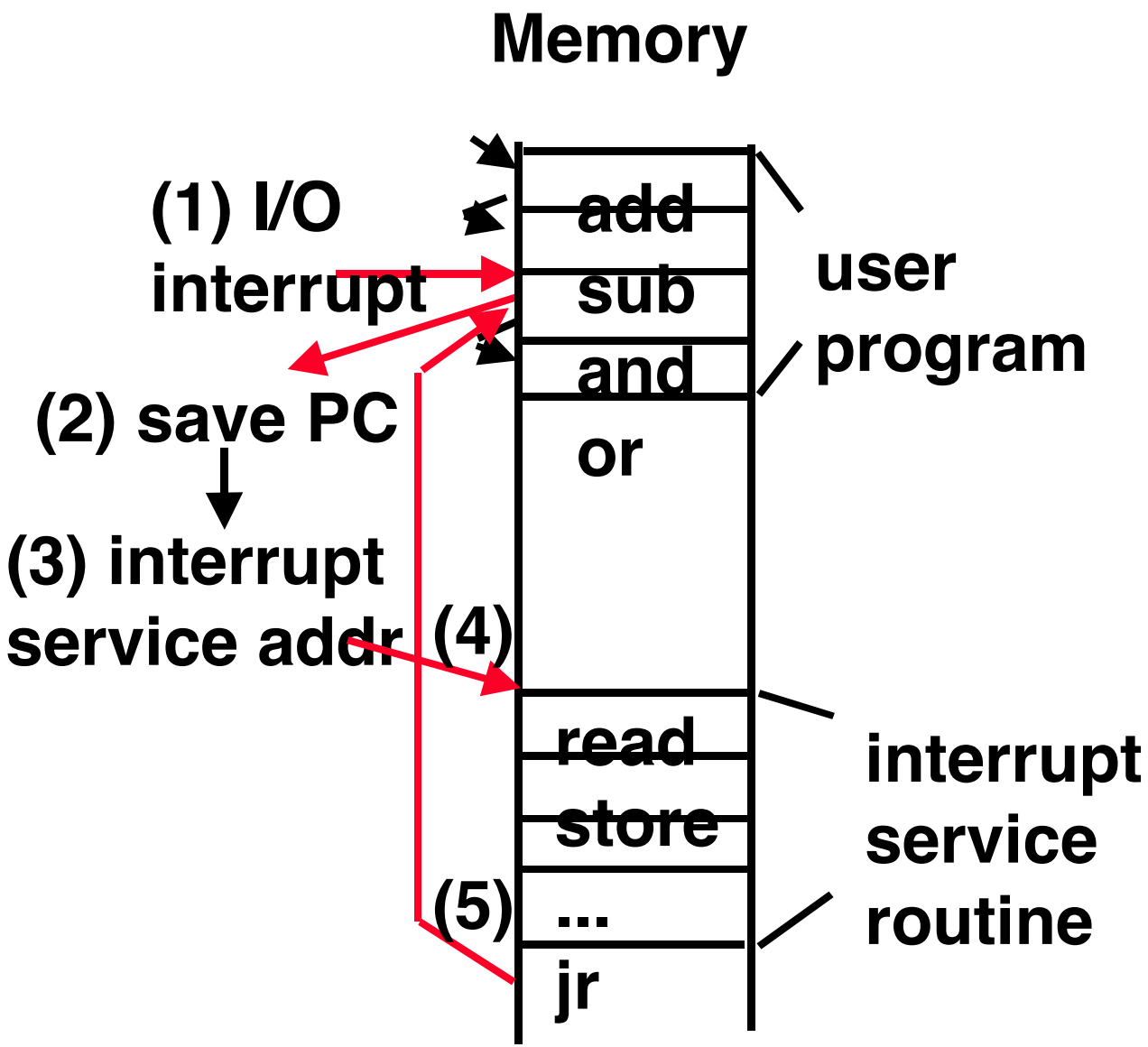
I/O Interrupt

- An I/O interrupt is like an overflow exceptions except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  - I/O interrupt does not prevent any instruction from completion

Definitions for Clarification

- **Exception**: signal marking that something "out of the ordinary" has happened and needs to be handled
- **Interrupt**: asynchronous exception
- **Trap**: synchronous exception
- Note: These are different from the book's definitions.

Interrupt Driven Data Transfer



Questions Raised about Interrupts

- **Which I/O device caused exception?**
  - **Needs to convey the identity of the device generating the interrupt** Cause register, or Vectored Interrupts
- **Can avoid interrupts during the interrupt routine?**
  - **What if more important interrupt occurs while servicing this interrupt?**
  - **Allow interrupt routine to be entered again?**
- **Who keeps track of status of all the devices, handle errors, know where to put/supply the I/O data?**

Fast Devices need I/O **Buffer** – not just a Register

... Amortize the cost of Interrupt over many data

example:

(just) 1 Gbit/s

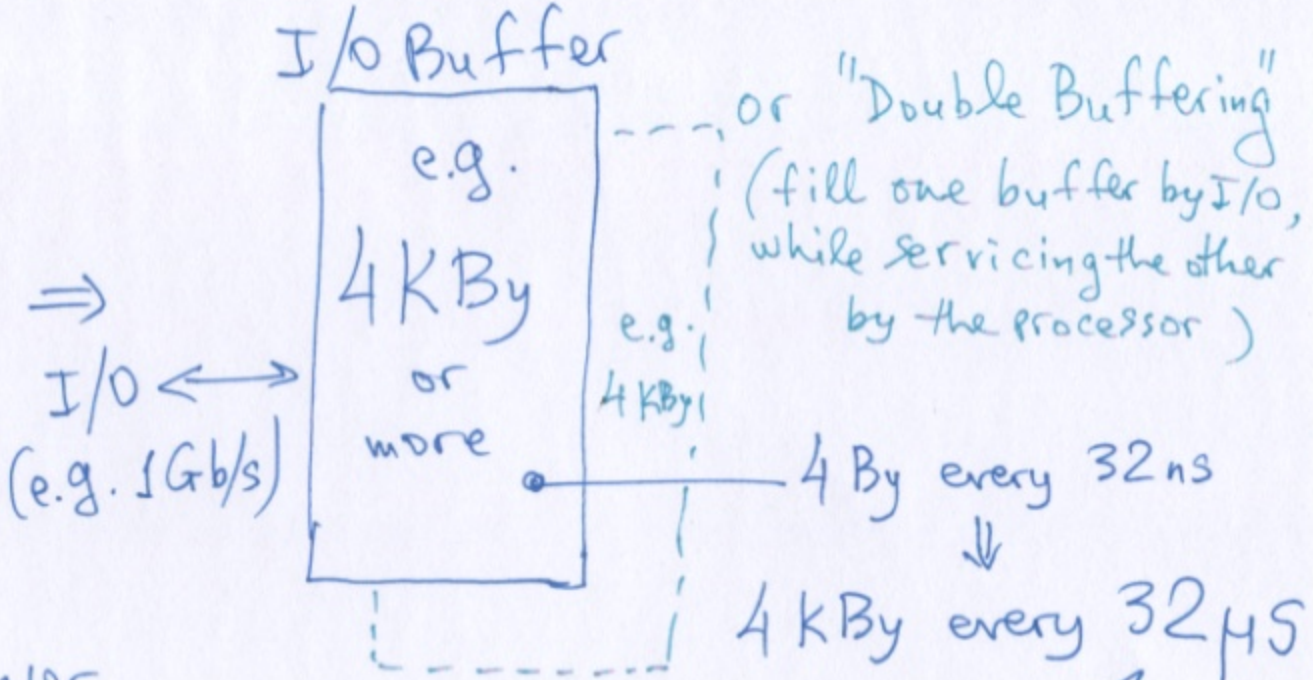
⇓

1 bit every 1 ns

⇓

32 bits every 32 ns

one "Register"



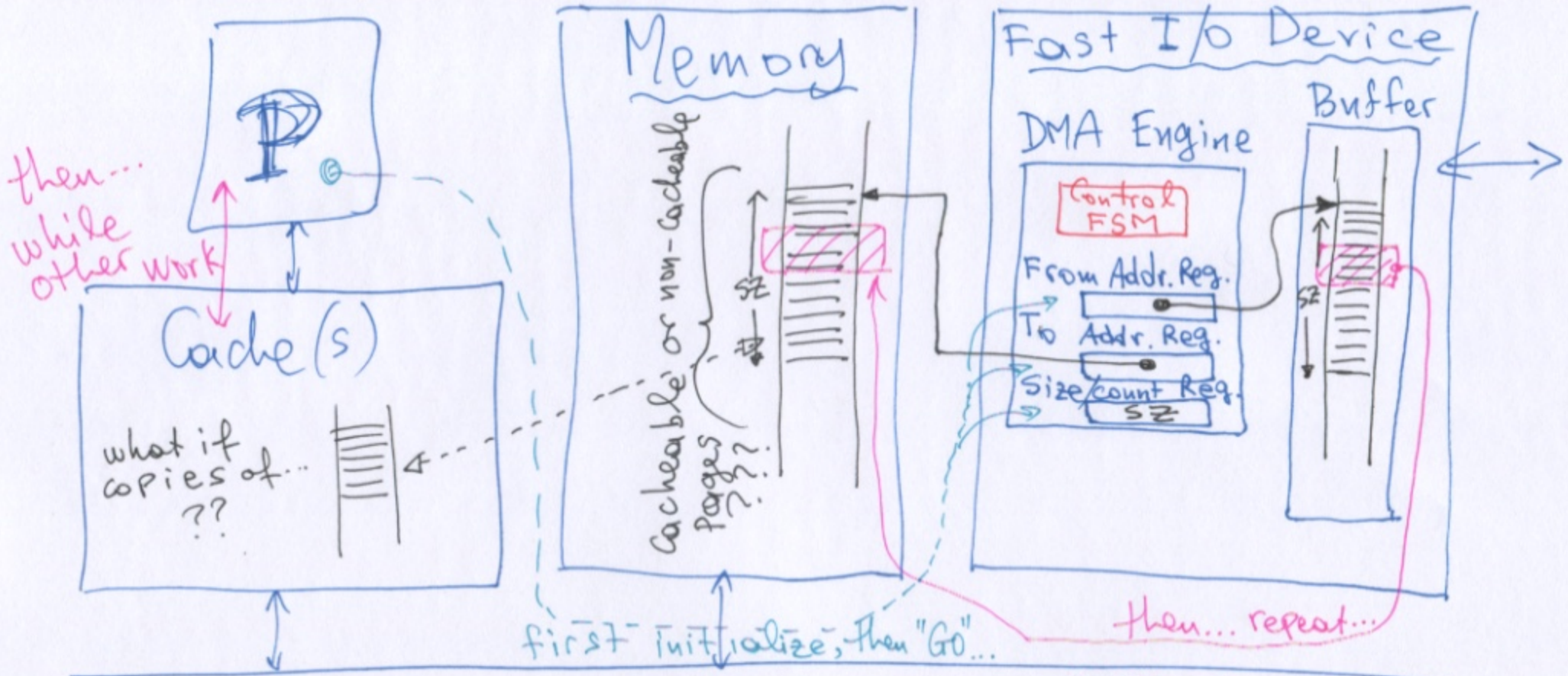
- Cost to poll status register
  (non-cacheable, off-chip)
  usually ≥ DRAM access
  usually ~ 100 ns (or more)
- then read the data register
  similarly ~ 100 ns

but this may still be a problem if transfer done word-by-word by the processor! (load-store loop)

OK ratio

- Cost of Interrupt + kernel interrupt handler usually ~ 1 µs (1000 ns)!

Direct Memory Access (DMA)



Alternatives for cacheability: (write-through only solves half the problem)

- DMA onto non-cacheable memory pages ... too slow when processor processes the I/O data
- Flush the cache before/after I/O DMA ... quite expensive operation (total flush? selective flush? (scan entire cache))
- Cache-Coherent DMA ← good! → next chapter...