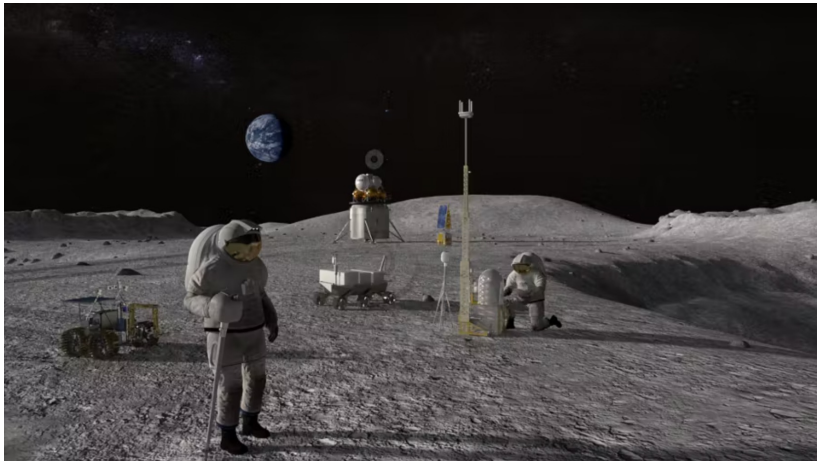
CS 152/252A Computer Architecture and Engineering

Sophia Shao

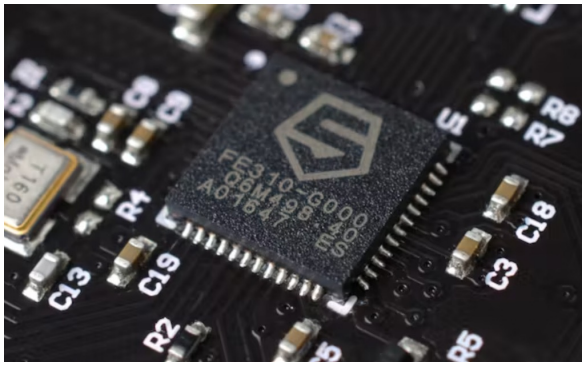
Lecture 6 – Memory

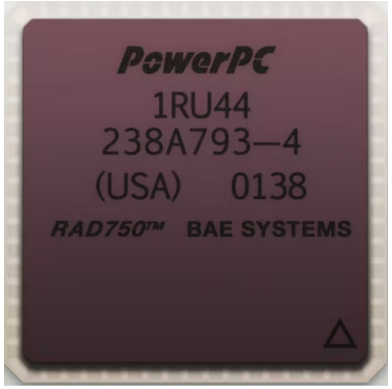


NASA, Microchip, SiFive Announces Partnership for RISC-V Spaceflight Computing Platform

NASA has confirmed a partnership with Microchip and SiFive to create a space-centric processor built around the free and open source RISC-V architecture: the High-Performance Spaceflight Computing (HPSC) chip.

https://www.hackster.io/news/nasa-microchip-sifive-announces-partnership-for-risc-v-spaceflight-computing-platform-f52c55cf14f6

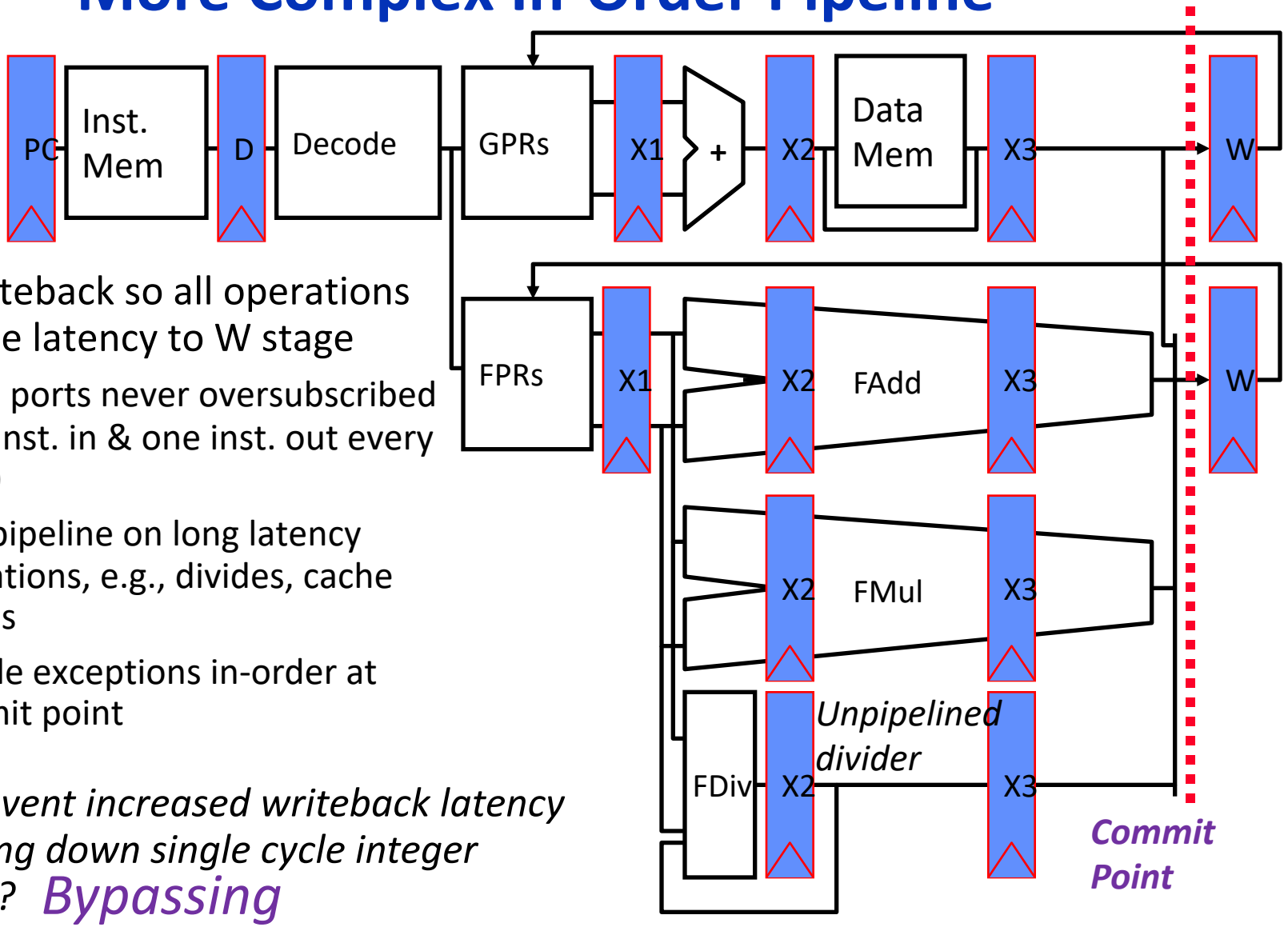




Last time in Lecture 4

- Handling exceptions in pipelined machines by passing exceptions down pipeline until instructions cross commit point in order
- Can use values before commit through bypass network
- Pipeline hazards can be avoided through software techniques: scheduling, loop unrolling
- Decoupled architectures use queues between “access” and “execute” pipelines to tolerate long memory latency
- Regularizing all functional units to have same latency simplifies more complex pipeline design by avoiding structural hazards, can be expanded to in-order superscalar designs

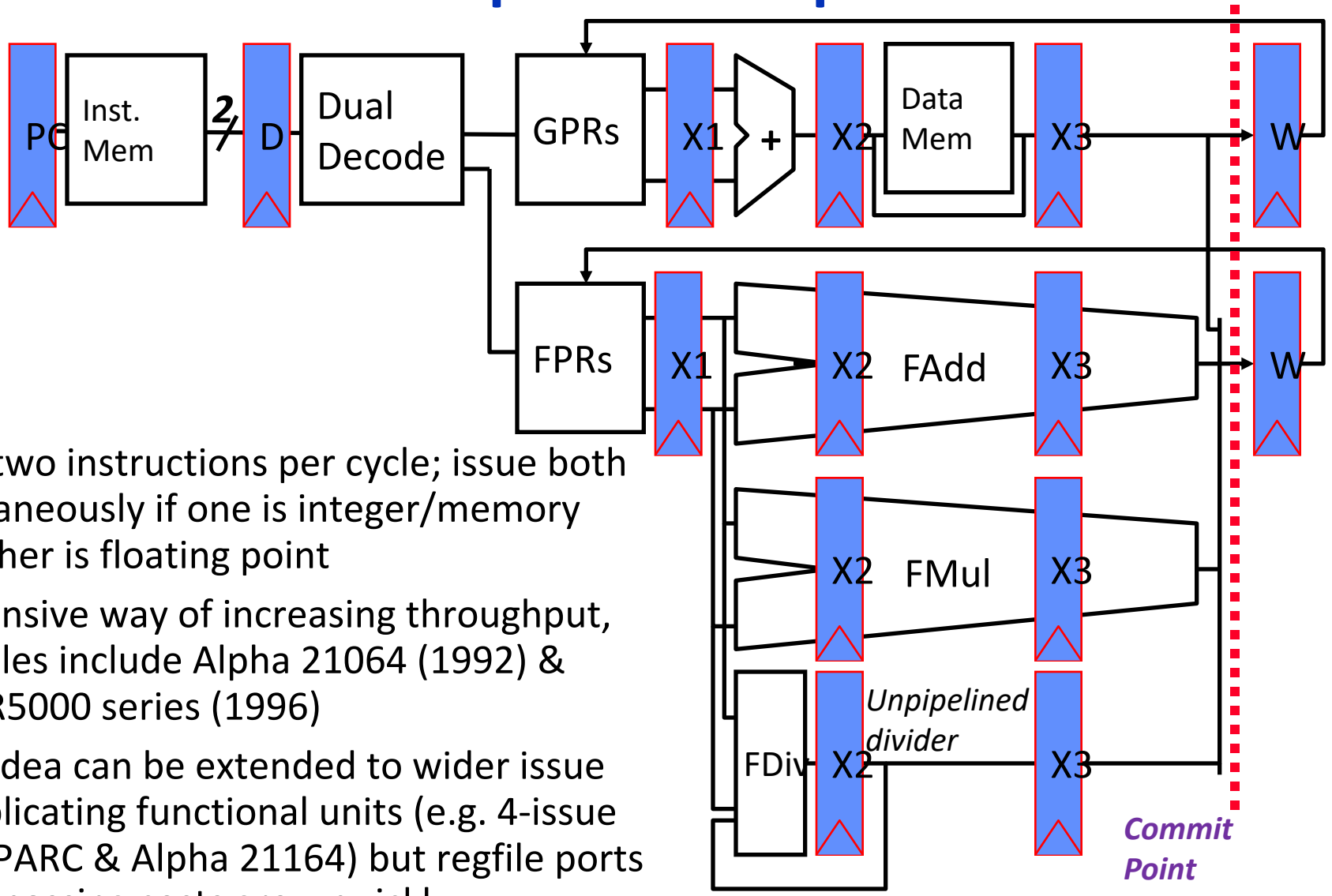
More Complex In-Order Pipeline



- Delay writeback so all operations have same latency to W stage
  - Write ports never oversubscribed (one inst. in & one inst. out every cycle)
  - Stall pipeline on long latency operations, e.g., divides, cache misses
  - Handle exceptions in-order at commit point

How to prevent increased writeback latency from slowing down single cycle integer operations? *Bypassing*

In-Order Superscalar Pipeline



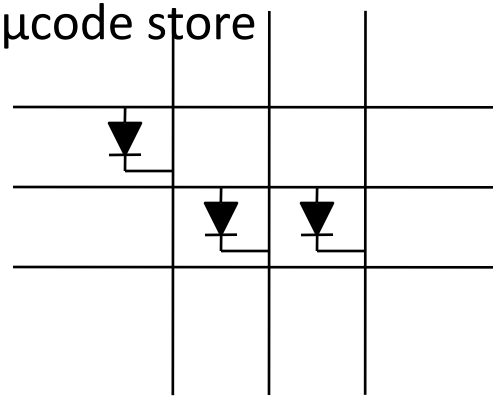
- Fetch two instructions per cycle; issue both simultaneously if one is integer/memory and other is floating point
- Inexpensive way of increasing throughput, examples include Alpha 21064 (1992) & MIPS R5000 series (1996)
- Same idea can be extended to wider issue by duplicating functional units (e.g. 4-issue UltraSPARC & Alpha 21164) but regfile ports and bypassing costs grow quickly

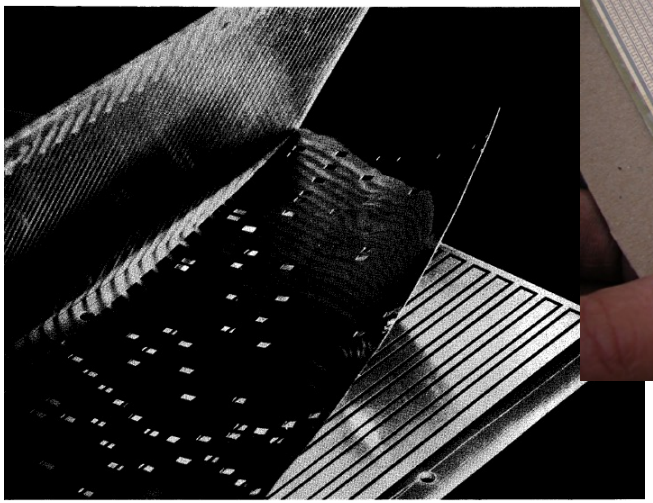
Early Read-Only Memory Technologies

Punched cards, From early 1700s through Jaquard Loom, Babbage, and then IBM

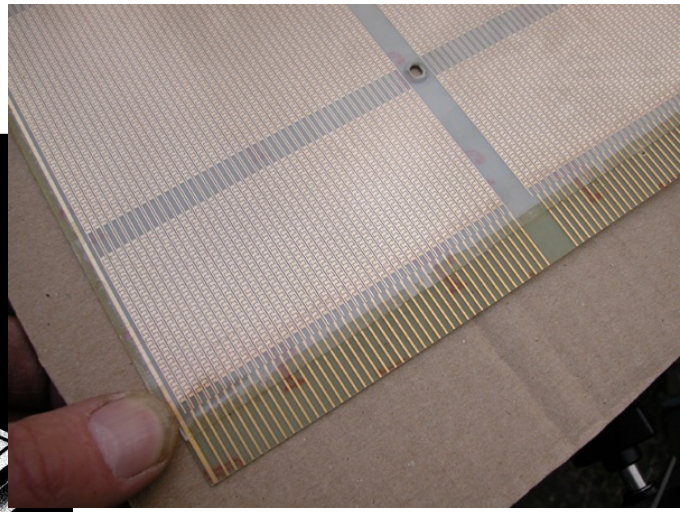
Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 µcode store





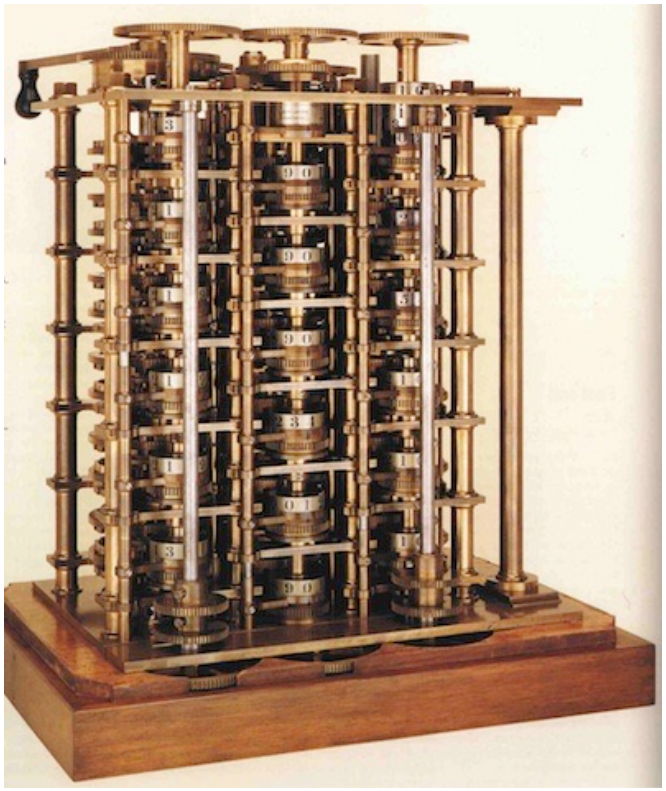
IBM Card Capacitor ROS

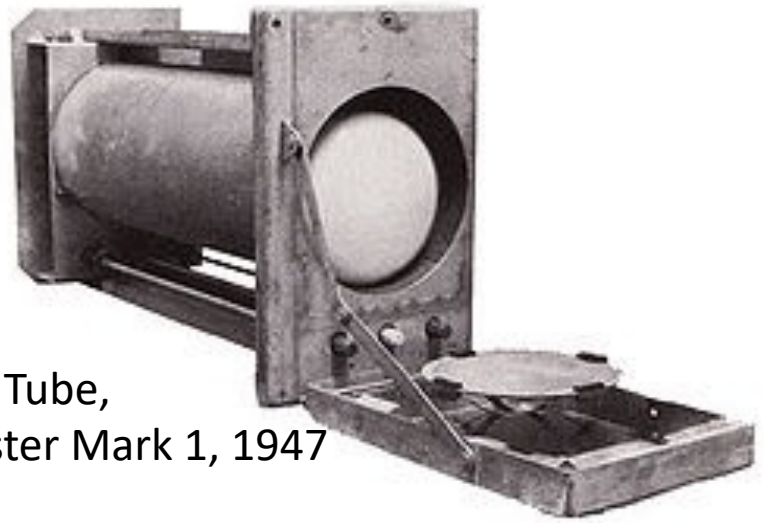


IBM Balanced Capacitor ROS

Early Read/Write Main Memory Technologies

Babbage, 1800s: Digits stored on mechanical wheels

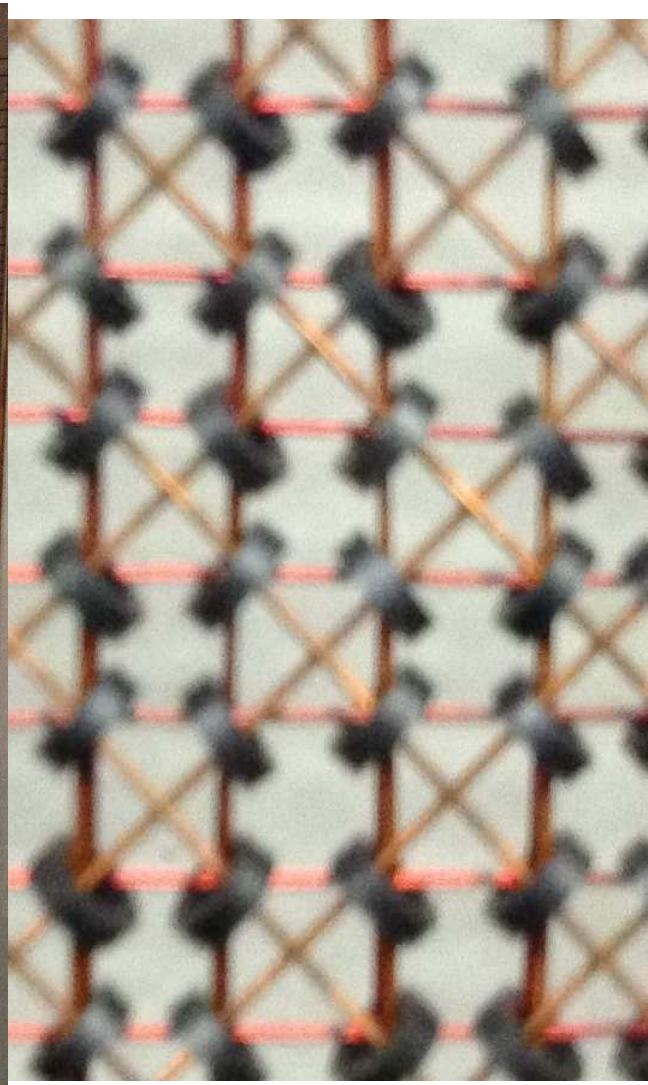




Williams Tube, Manchester Mark 1, 1947

Rotating magnetic drum memory on IBM 650, 1954

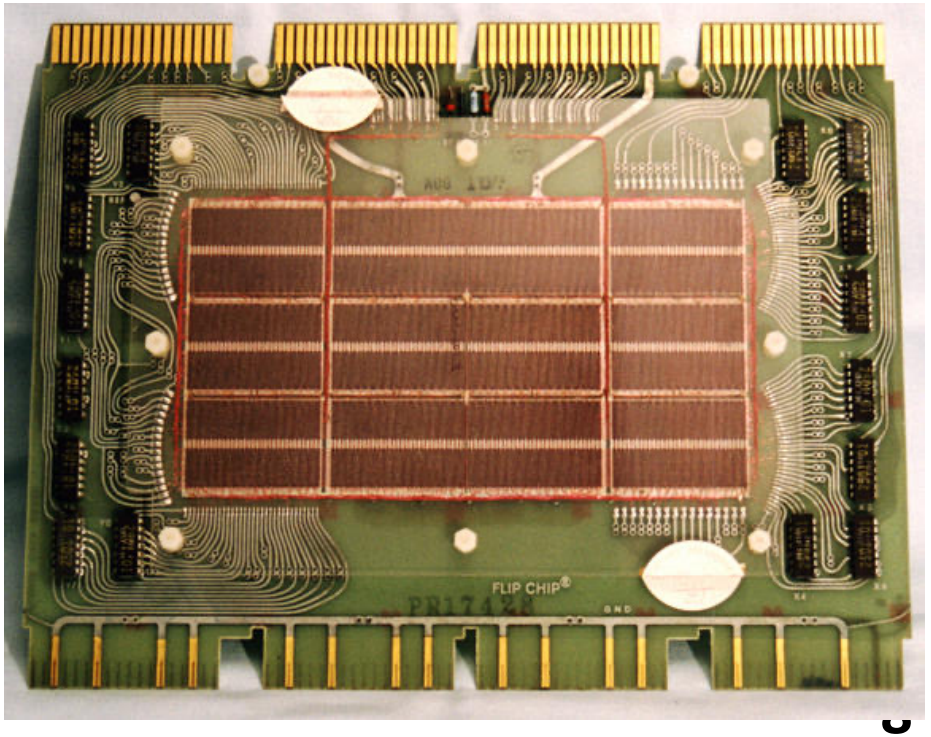
MIT Whirlwind Core Memory, 1950



Core Memory

- Core memory was first large scale reliable main memory
  - invented by Forrester in late 40s/early 50s at MIT for Whirlwind project
- Bits stored as magnetization polarity on small ferrite cores threaded onto two-dimensional grid of wires
- Coincident current pulses on X and Y wires would write cell and also sense original state (destructive reads)

- Robust, non-volatile storage
- Used on space shuttle computers
- Cores threaded onto wires by hand (25 billion a year at peak production)
- Core access time ~ 1µs

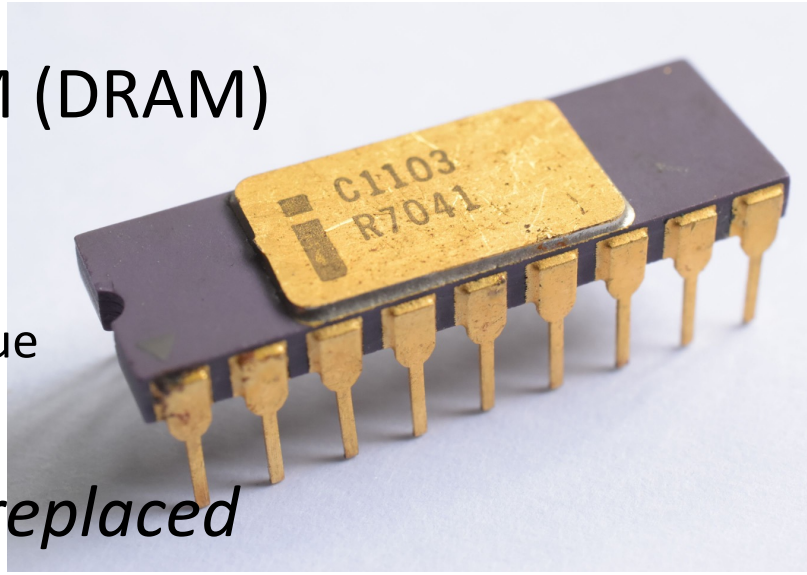


DEC PDP-8/E Board,
4K words x 12 bits, (1968)

Semiconductor Memory

- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
  - Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).

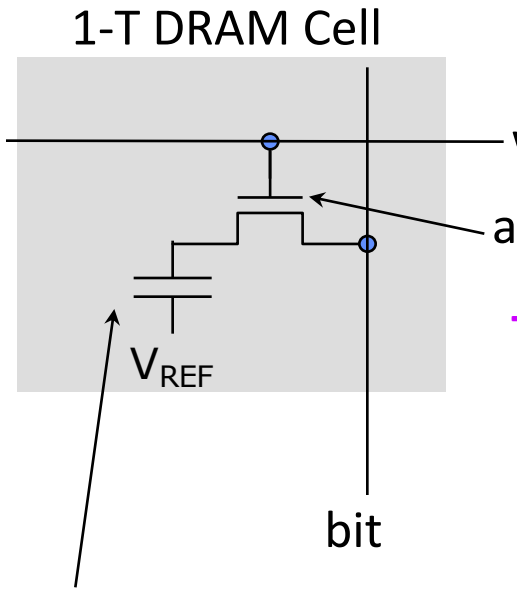
- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

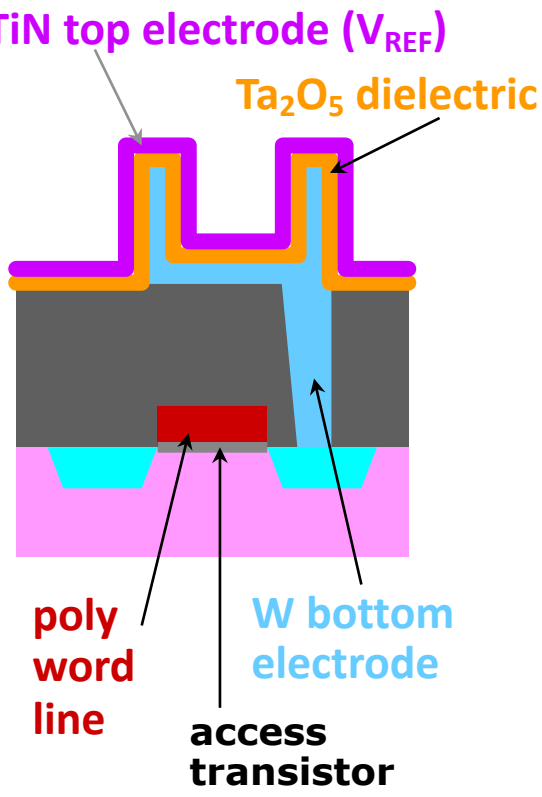


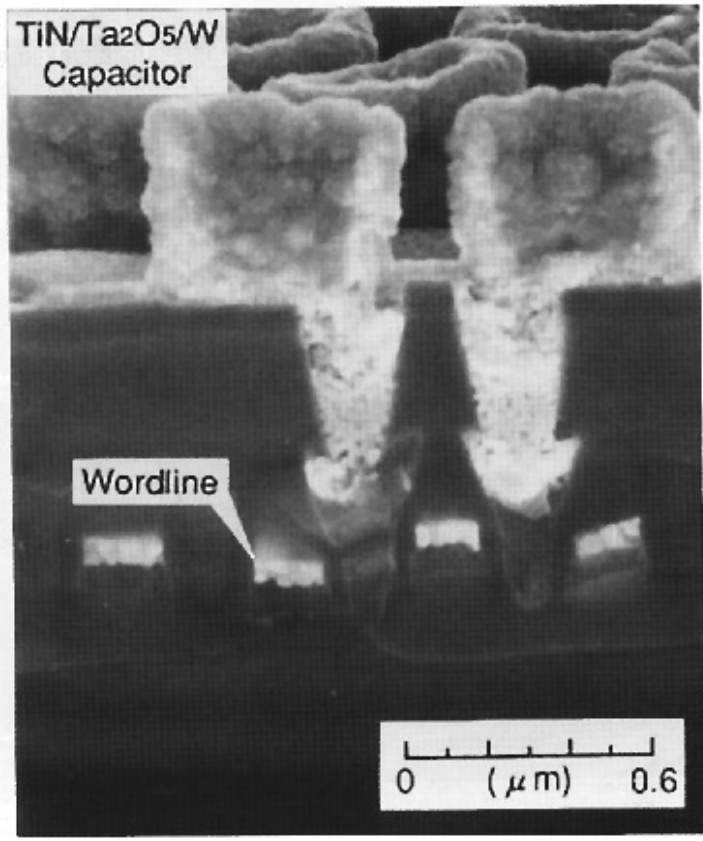
[Thomas Nguyen CC-BY-SA]

Semiconductor memory quickly replaced core in '70s

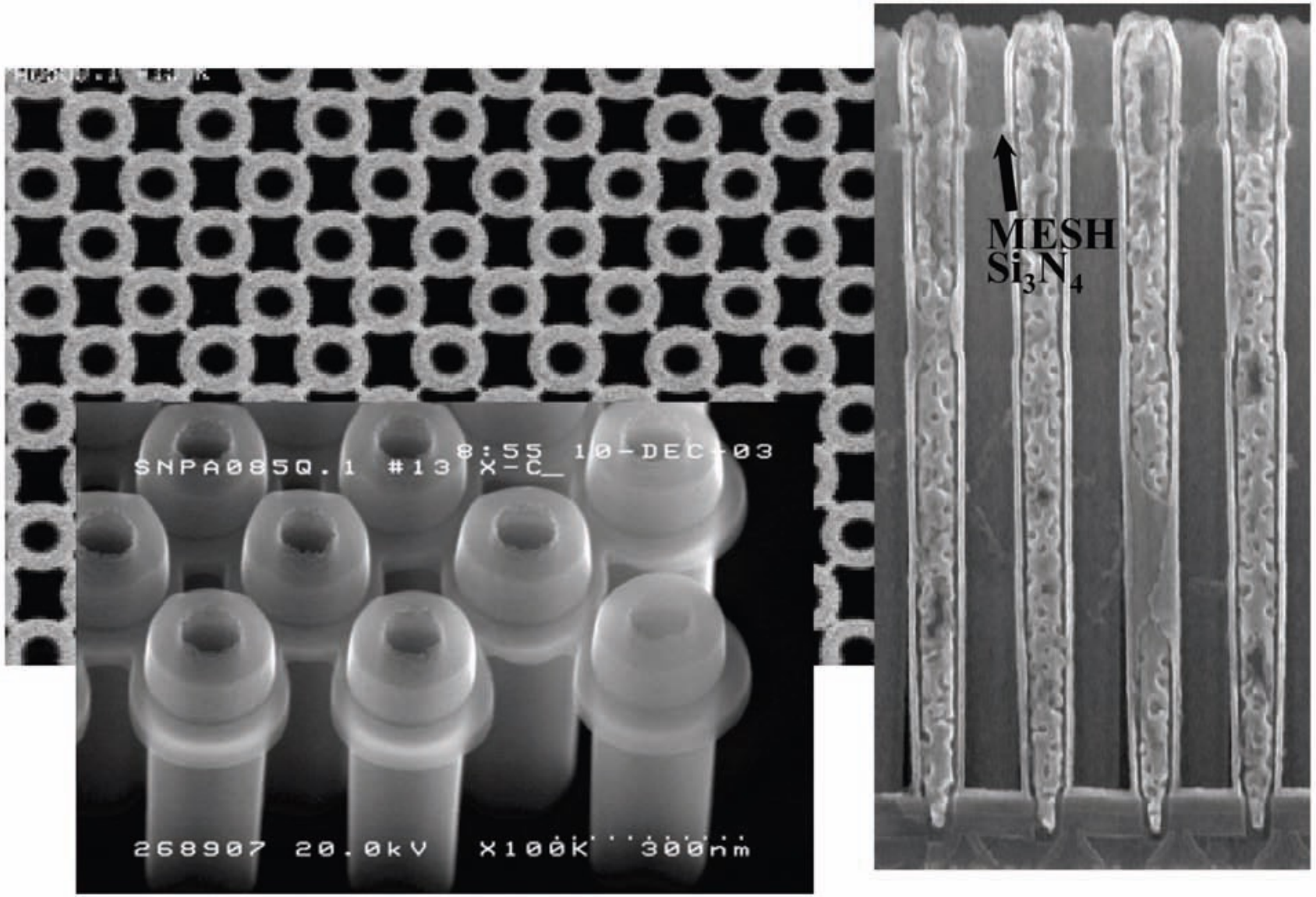
One-Transistor Dynamic RAM [Dennard, IBM]





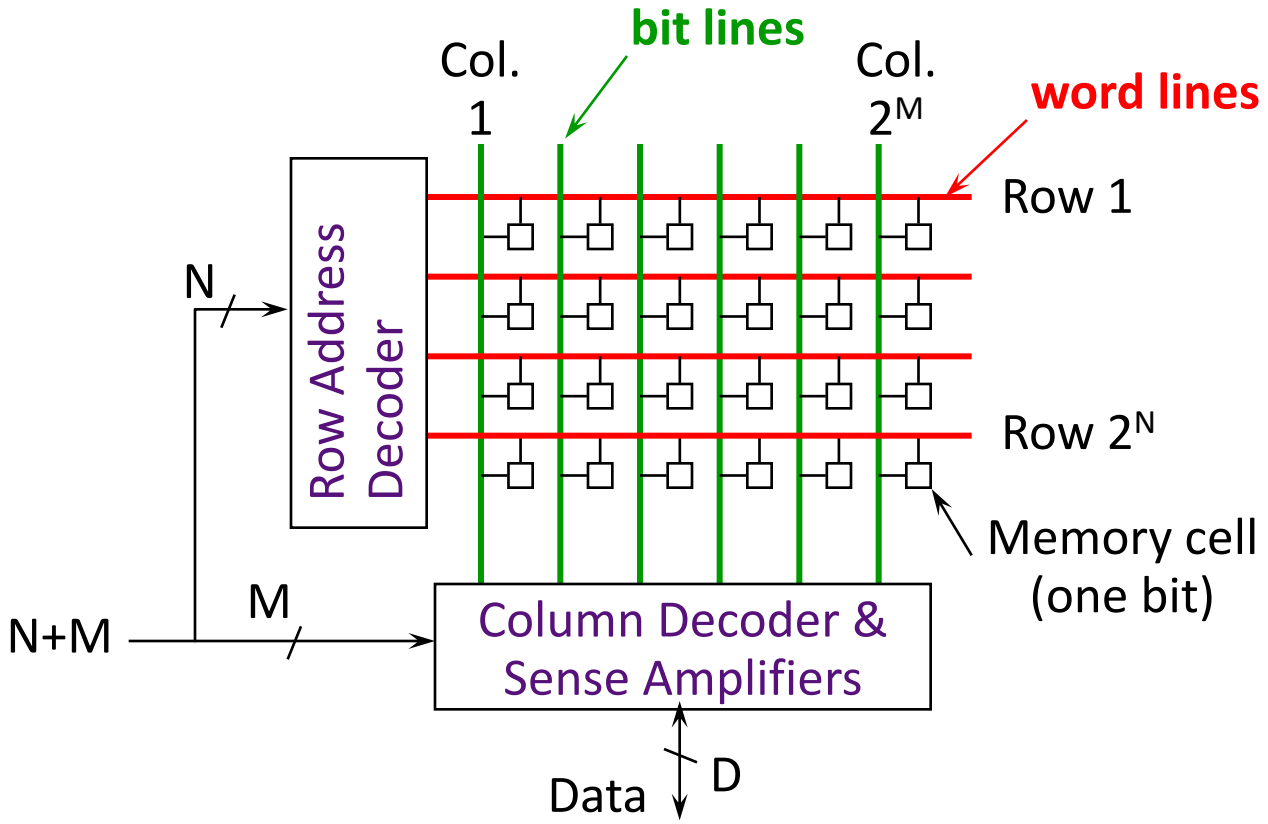


Modern DRAM Structure



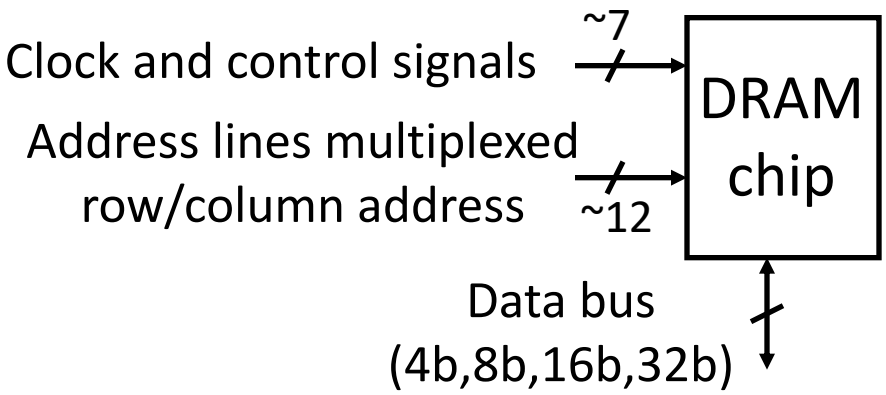
[Samsung, sub-70nm DRAM, 2004]

DRAM Architecture

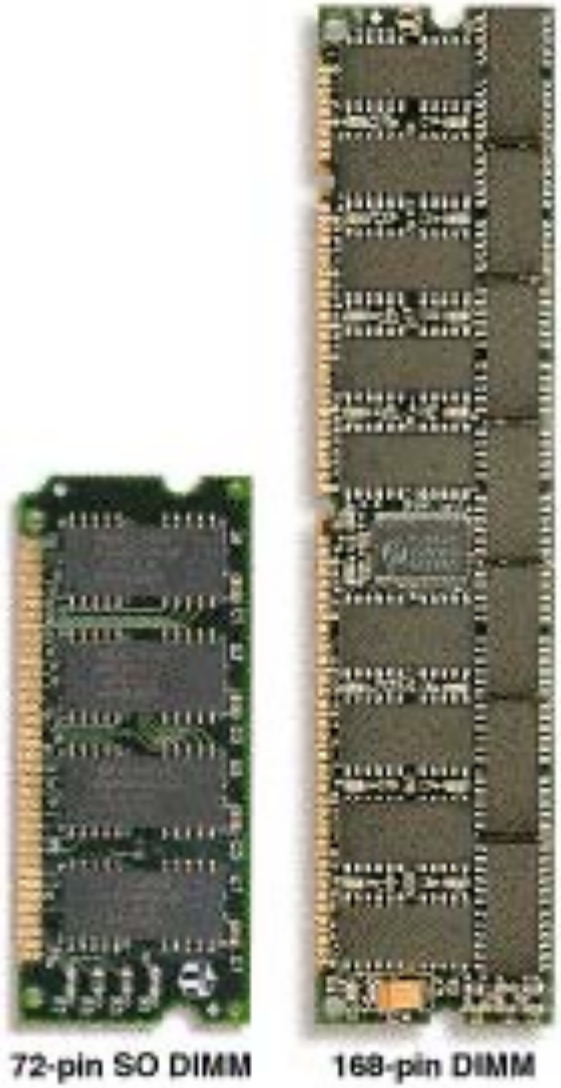


- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

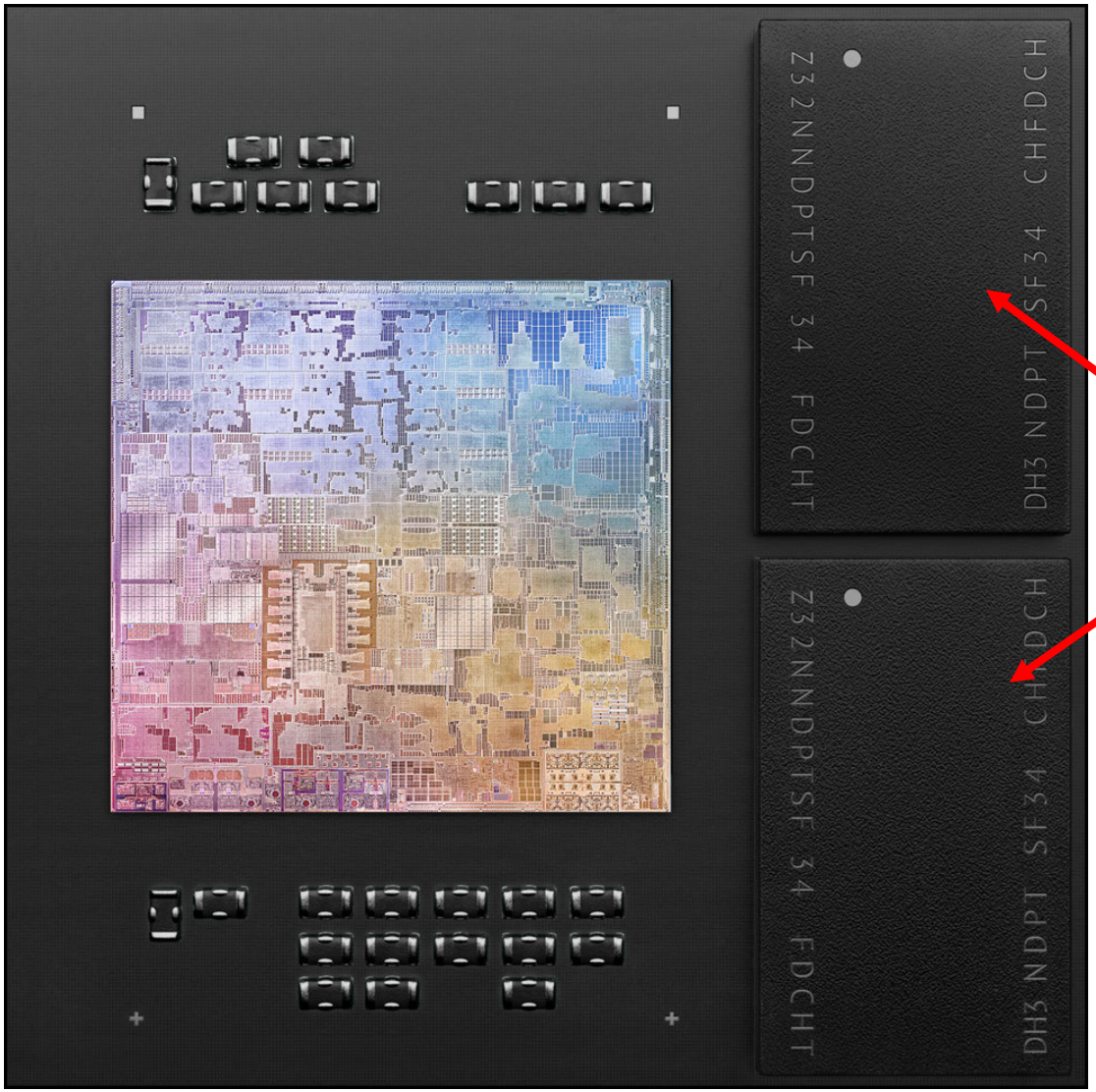
DRAM Packaging
(Laptops/Desktops/Servers)



- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)



DRAM Packaging, Apple M1



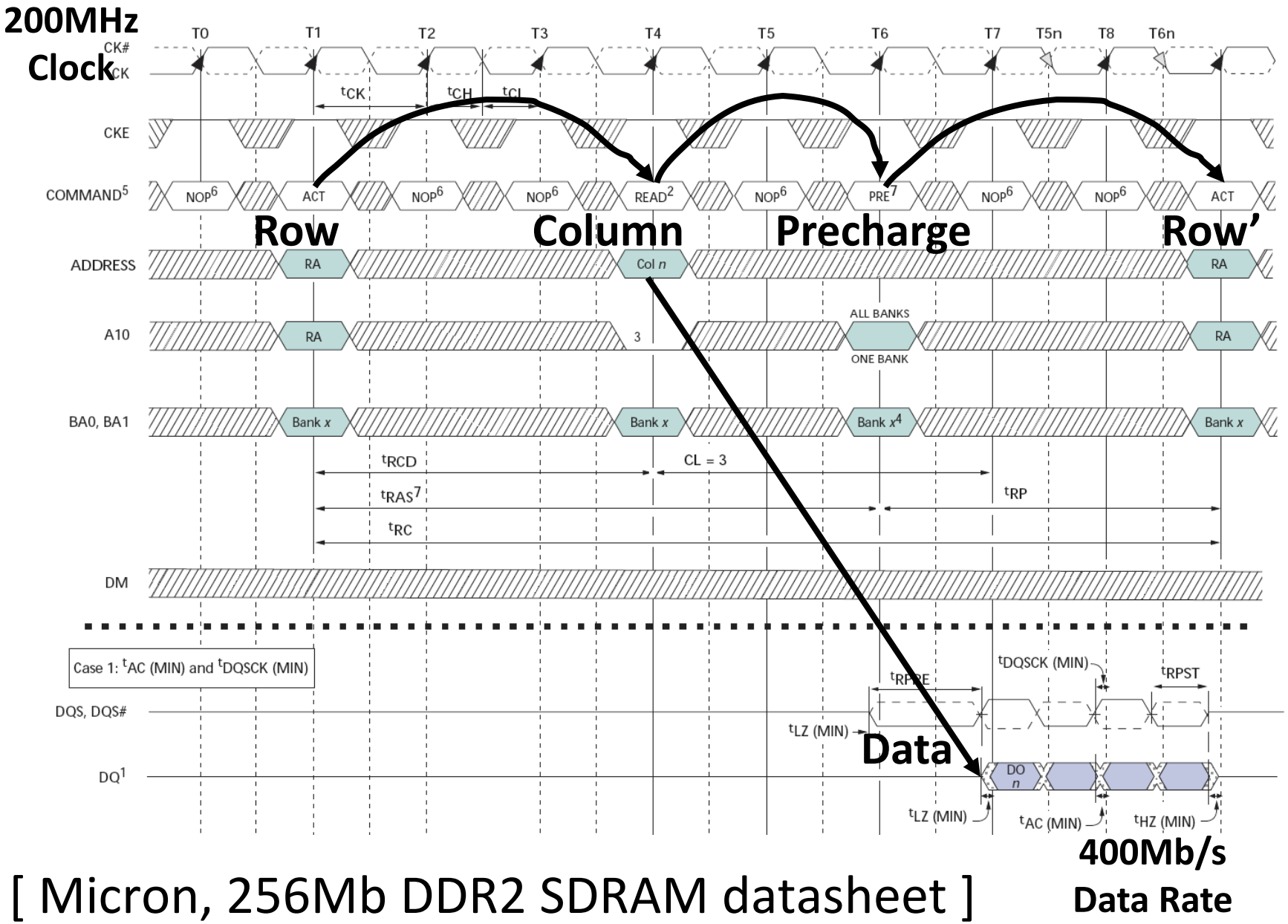
Two DRAM chips on same package as system SoC

- 128b databus, running at 4.2Gb/s
- 68GB/s bandwidth

DRAM Operation

- Three steps in read/write access to a given bank
- Row access (RAS)
  - decode row address, enable addressed row (often multiple Kb in row)
  - bitlines share charge with storage cell
  - small change in voltage detected by sense amplifiers which latch whole row of bits
  - sense amplifiers drive bitlines full rail to recharge storage cells
- Column access (CAS)
  - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
  - on read, send latched bits out to chip pins
  - on write, change sense amplifier latches which then charge storage cells to required value
  - can perform multiple column accesses on same row without another row access (burst mode)
- Precharge
  - charges bit lines to known value, required before next row access
- Each step has a latency of around 15-20ns in modern DRAMs
- Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture

Double-Data Rate (DDR2) DRAM



[Micron, 256Mb DDR2 SDRAM datasheet]

Computer Architecture Terminology

Latency (in seconds or cycles): Time taken for a single operation from start to finish (initiation to useable result)

Bandwidth (in operations/second or operations/cycle): Rate of which operations can be performed

Occupancy (in seconds or cycles): Time during which the unit is blocked on an operation (structural hazard)

Note, for a single functional unit:

- Occupancy can be much less than latency (how?)
- Occupancy can be greater than latency (how?)
- Bandwidth can be greater than 1/latency (how?)
- Bandwidth can be less than 1/latency (how?)

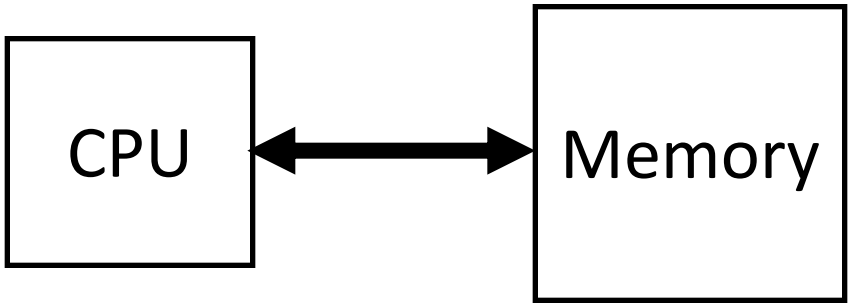
CS152 Administrivia

- HW1 released
  - Due Today
- Lab1 released
  - Due Feb 09
- Lab reports must be readable English summaries – **not dumps of log files!!!!!!**
  - We will reward good reports, and penalize undecipherable reports
  - Page limit (check lab spec/Ed)
- Lecture Ed thread
  - One thread per lecture
  - Post your questions following the format:
    - [Slide #] Your question
  - The staff team will address and clarify the questions asynchronously.
- Tell us what you think
  - http://tinyurl.com/cs152feedback

CS252 Administrivia

- CS252 Readings on
  - https://ucb-cs252-sp23.hotcrp.com/u/0/
  - Use hotcrp to upload reviews before Wednesday:
    - Write one paragraph on main content of paper including good/bad points of paper
    - Also, answer/ask 1-3 questions about paper for discussion
    - First two “360 Architecture”, “VAX11-780”
  - 2-3pm Wednesday, Soda 606/Zoom
- CS252 Project Timeline
  - Proposal Wed Feb 22
  - Use 252A GSIs (Abe and Prashanth) and my OHs to get feedback.

CPU-Memory Bottleneck



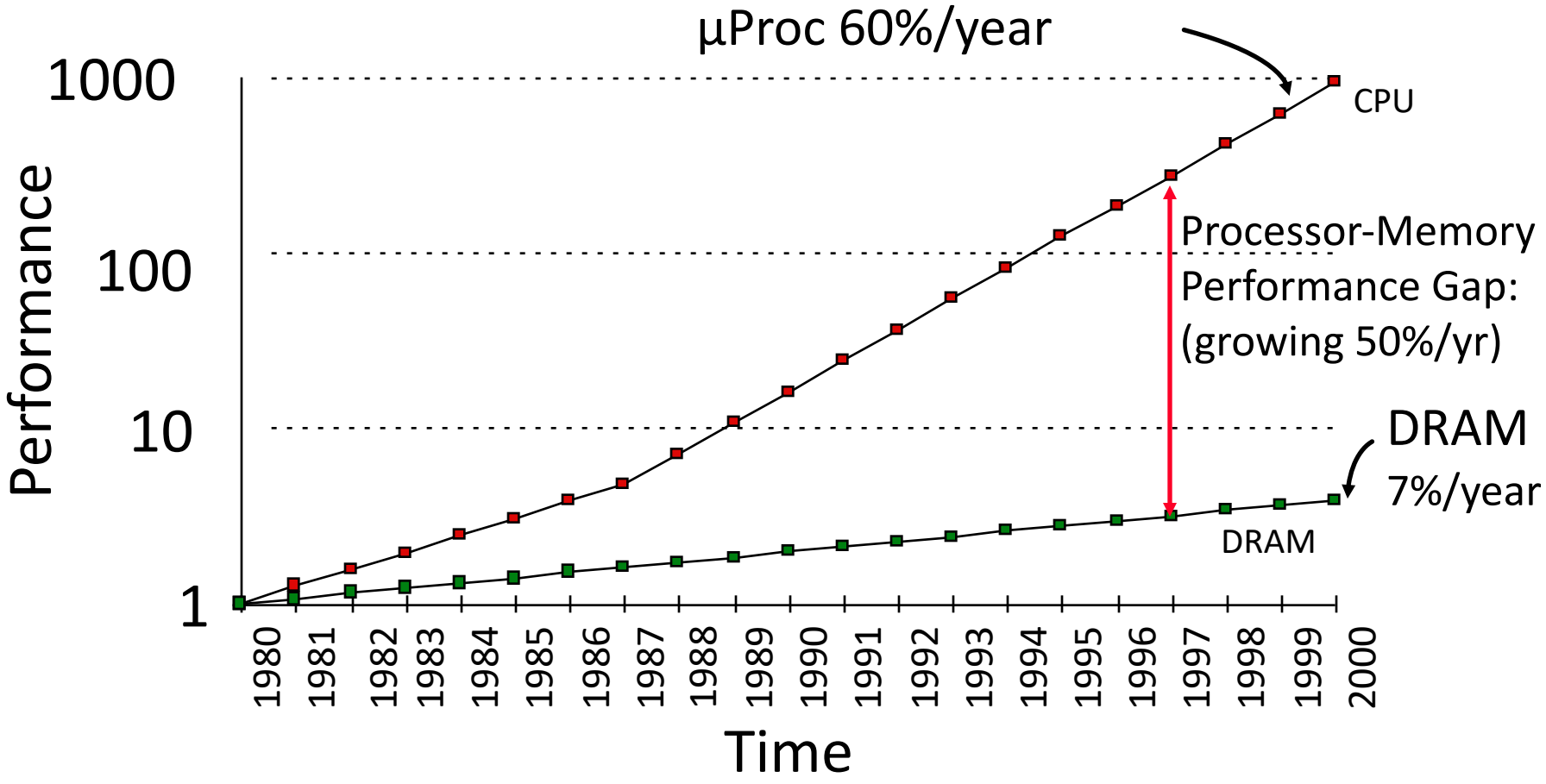
Performance of high-speed computers is usually limited by memory bandwidth & latency

- Latency (time for a single access)
  - Memory access time >> Processor cycle time
- Bandwidth (number of accesses per unit time)

  if fraction m of instructions access memory

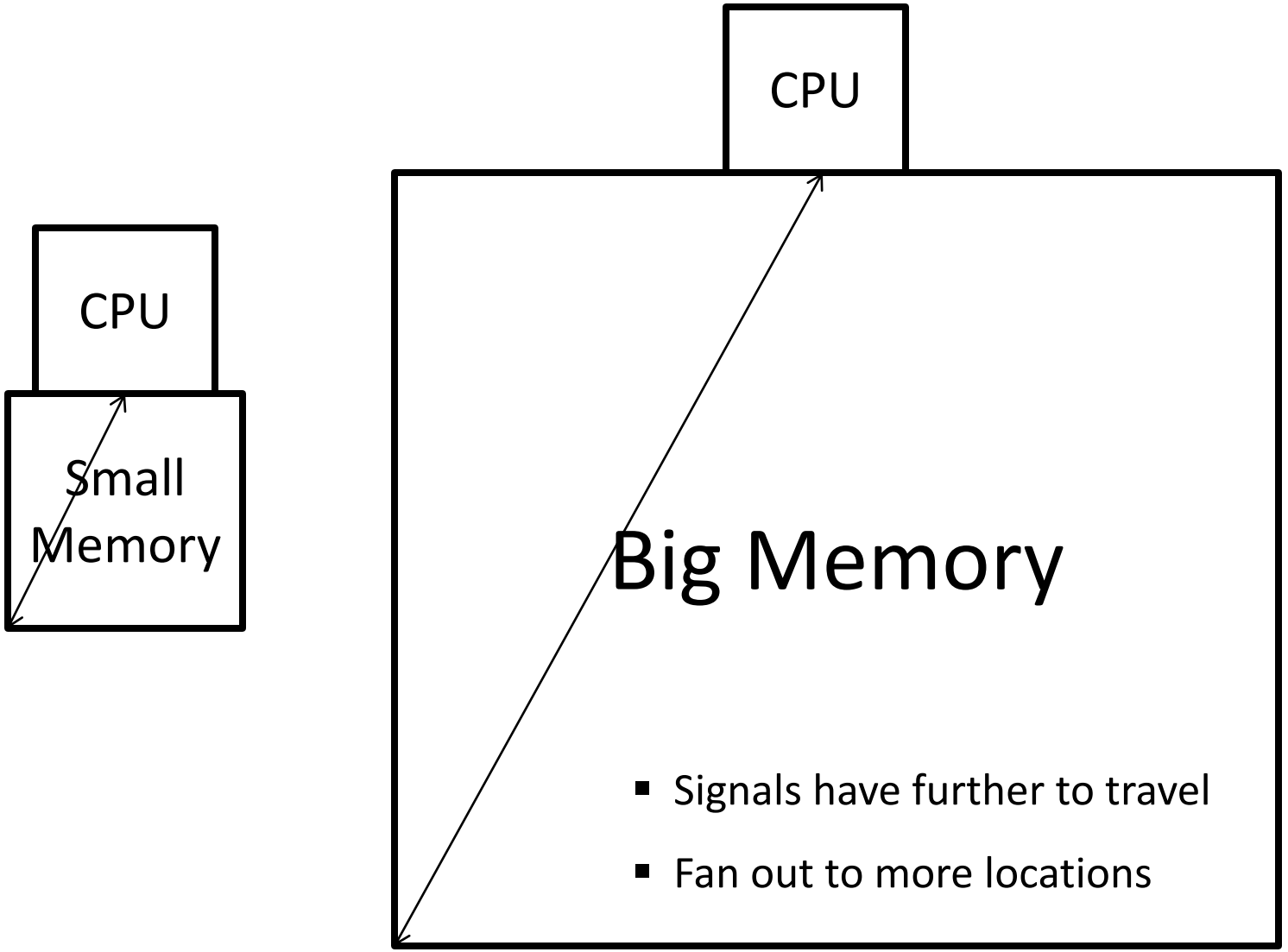
  ⇒ 1+m memory references / instruction
- *Also, Occupancy (time a memory bank is busy with one request)*

Processor-DRAM Gap (latency)

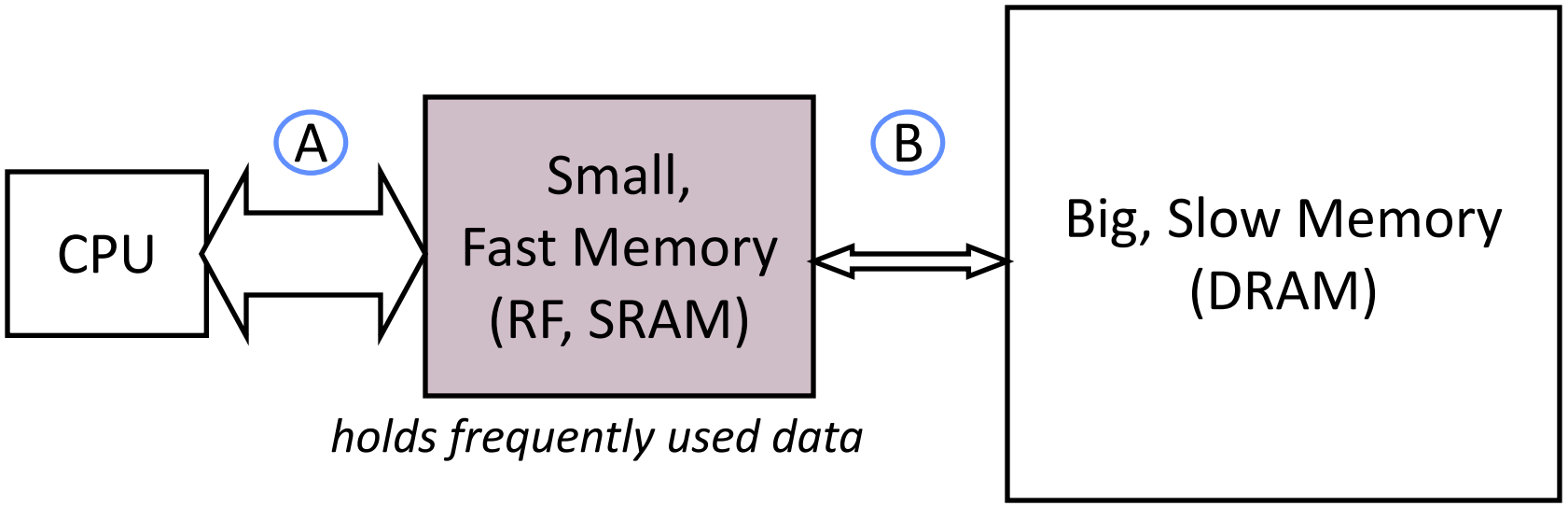


Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!

Physical Size Affects Latency



Memory Hierarchy



- *capacity*: Register << SRAM << DRAM
- *latency*: Register << SRAM << DRAM
- *bandwidth:* on-chip >> off-chip

On a data access:

if data $\in$ fast memory $\Rightarrow$ low latency access *(SRAM)*

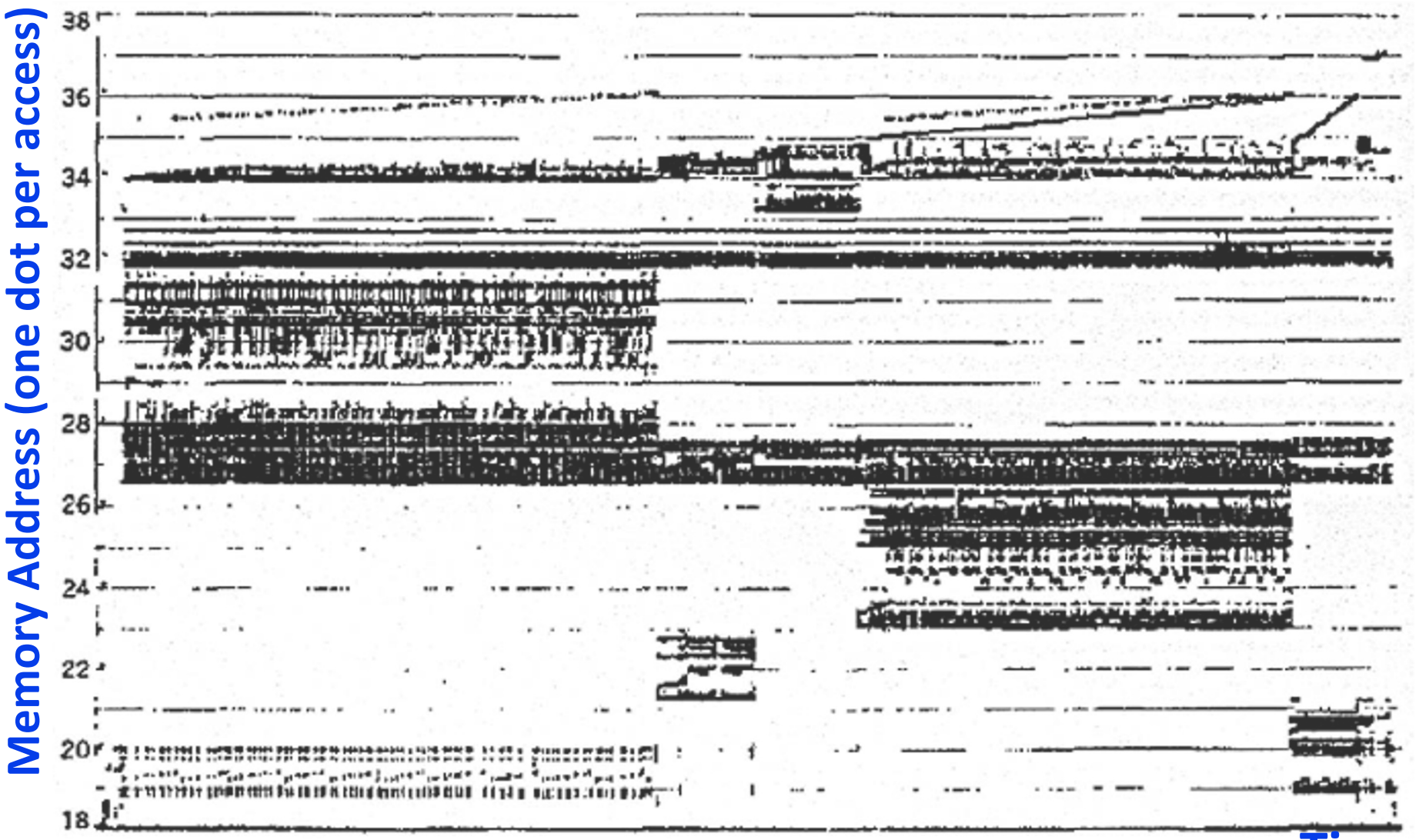
if data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

Management of Memory Hierarchy

- Small/fast storage, e.g., registers
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - *but hardware might do things behind software's back, e.g., stack management, register renaming*

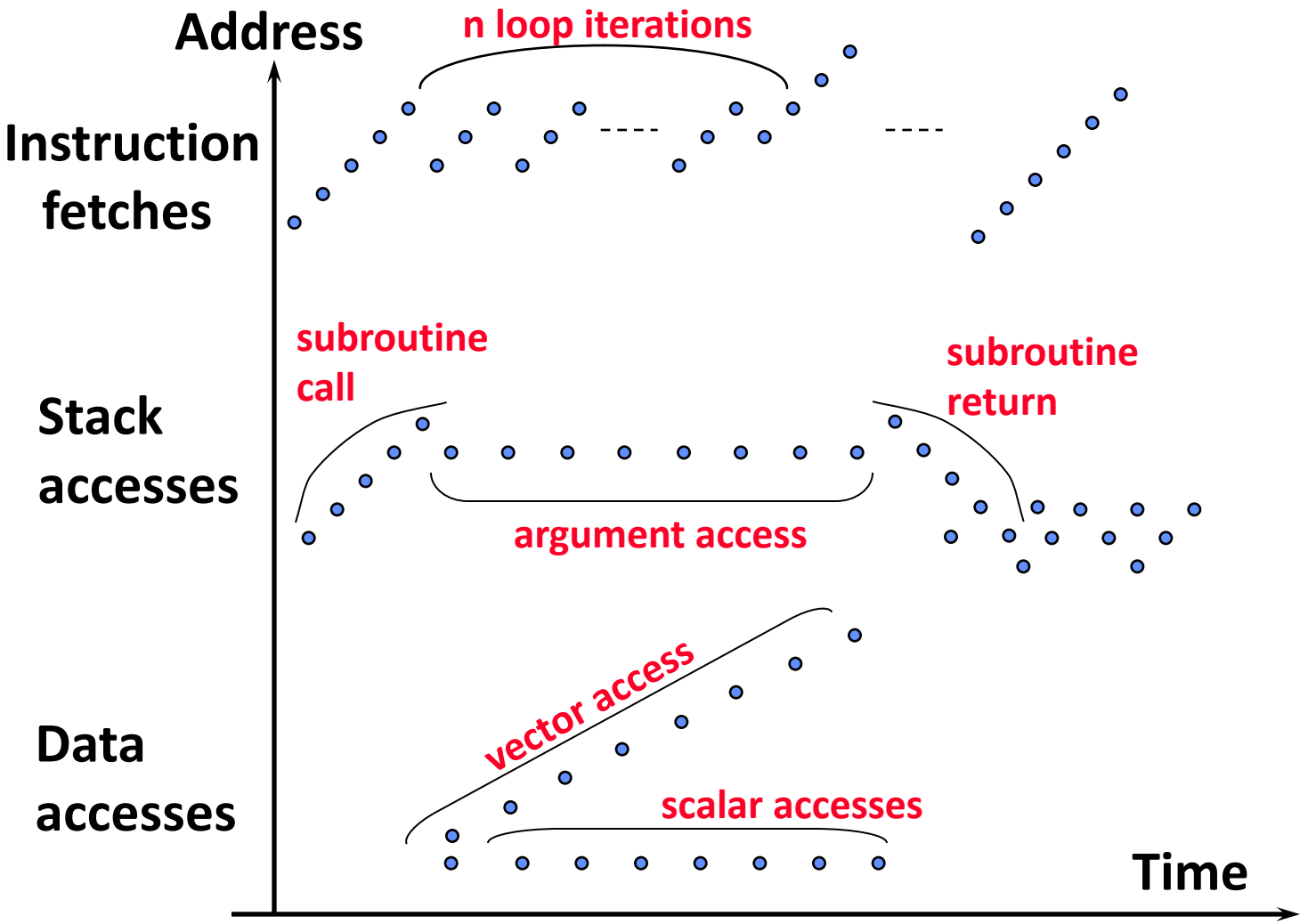
- Larger/slower storage, e.g., main memory
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
    - *but software may provide "hints", e.g., prefetch*

Real Memory Reference Patterns



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Typical Memory Reference Patterns

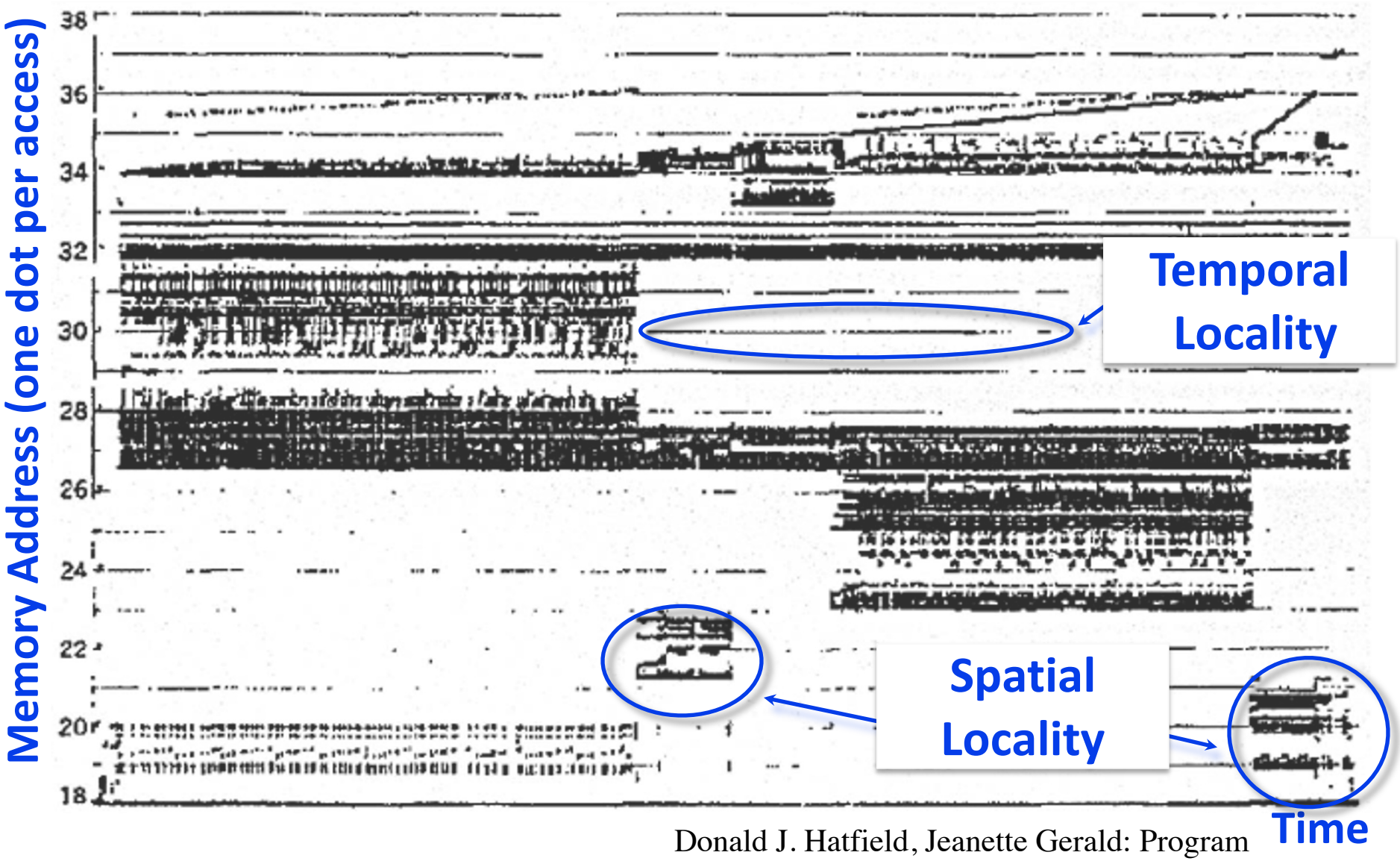


Two predictable properties of memory references:

- **Temporal Locality**: If a location is referenced it is likely to be referenced again in the near future.

- **Spatial Locality**: If a location is referenced it is likely that locations near it will be referenced in the near future.

Memory Reference Patterns



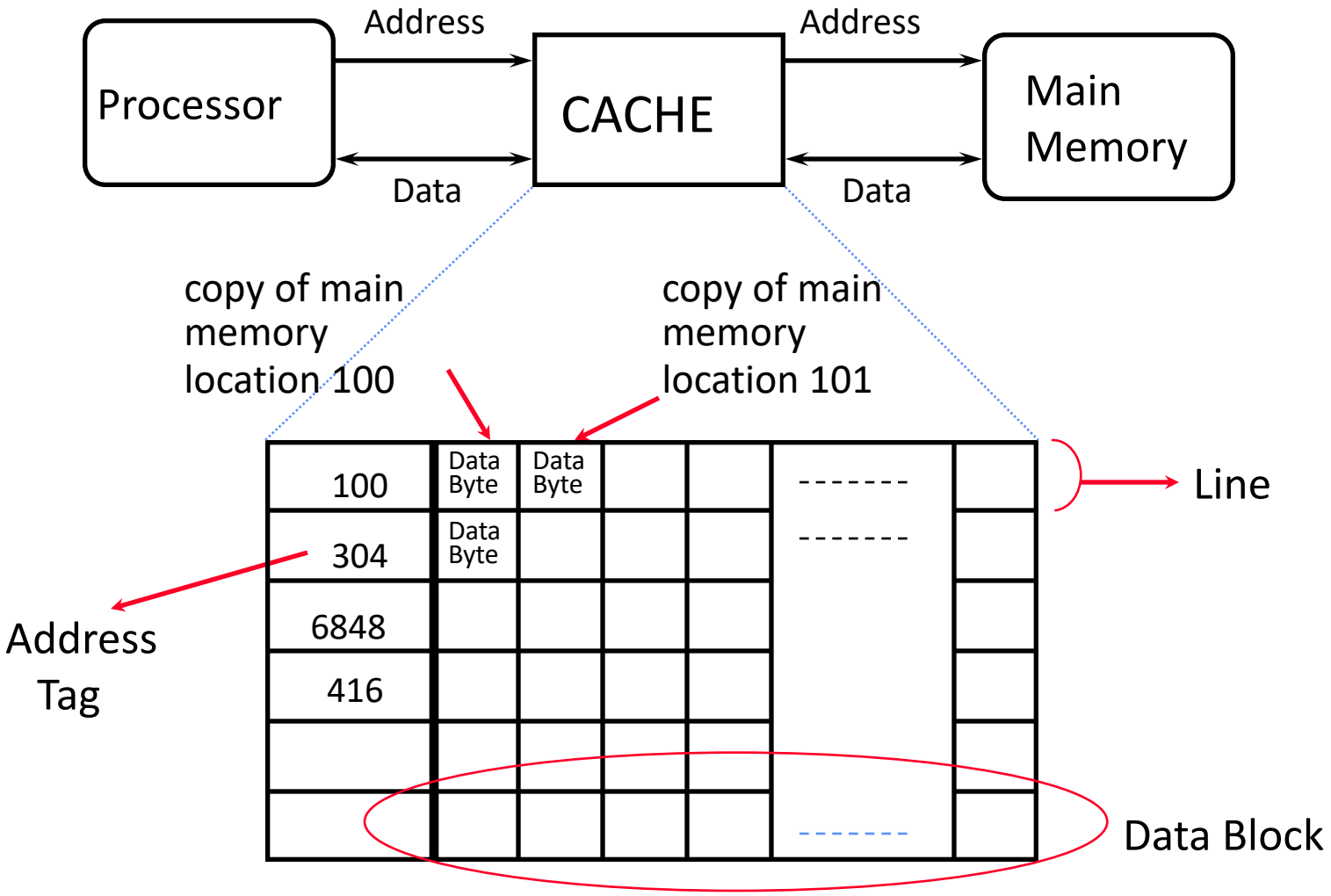
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Caches exploit both types of predictability:

- Exploit temporal locality by remembering the contents of recently accessed locations.

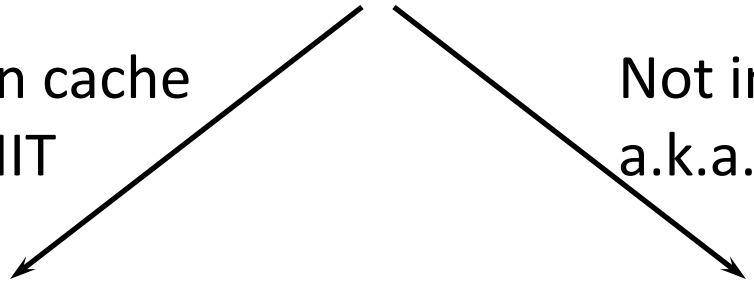
- Exploit spatial locality by fetching blocks of data around recently accessed locations.

Inside a Cache



Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either



Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

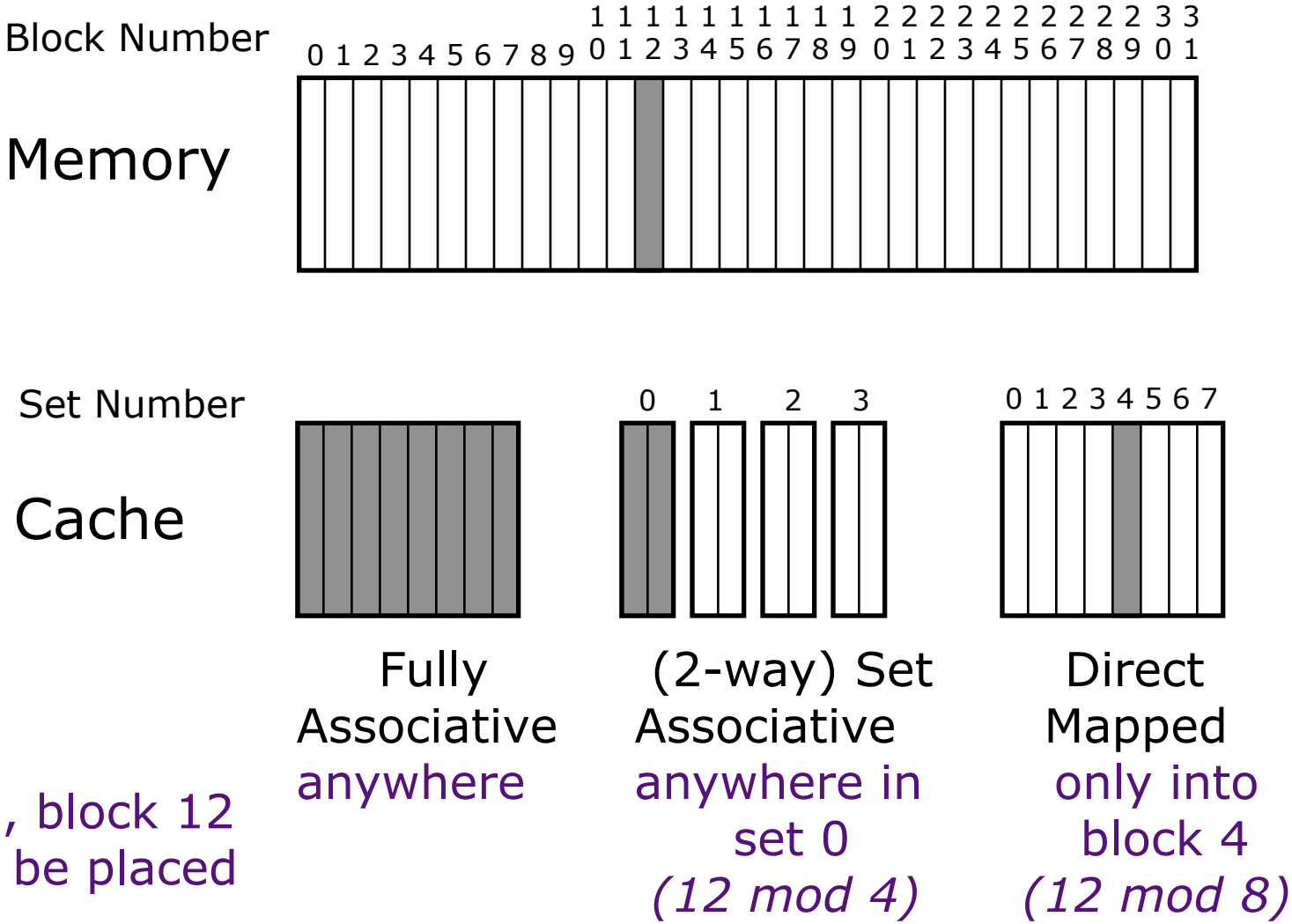
Read block of data from Main Memory

Wait …

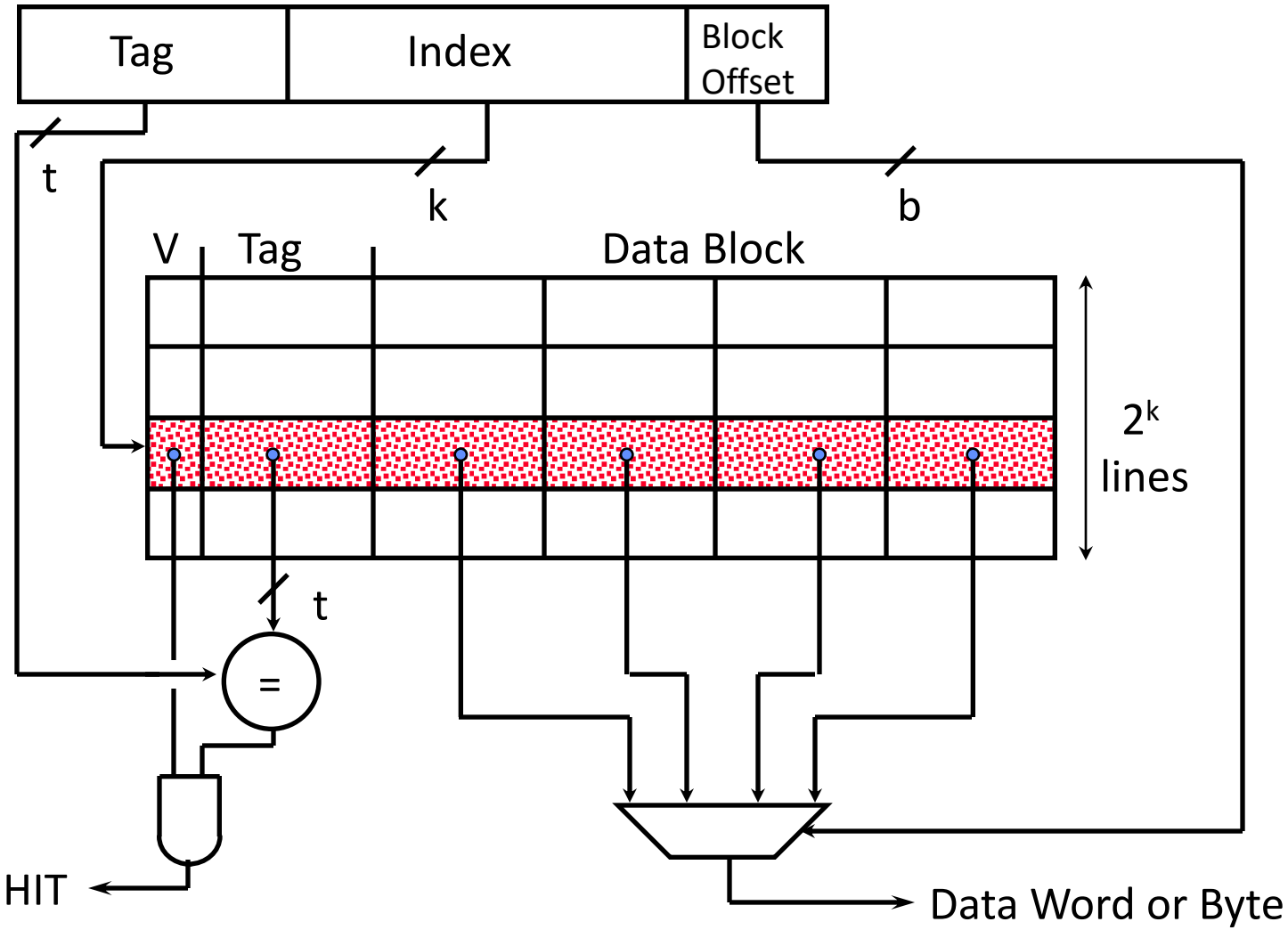
Return data to processor and update cache

Q: Which line do we replace?

Placement Policy

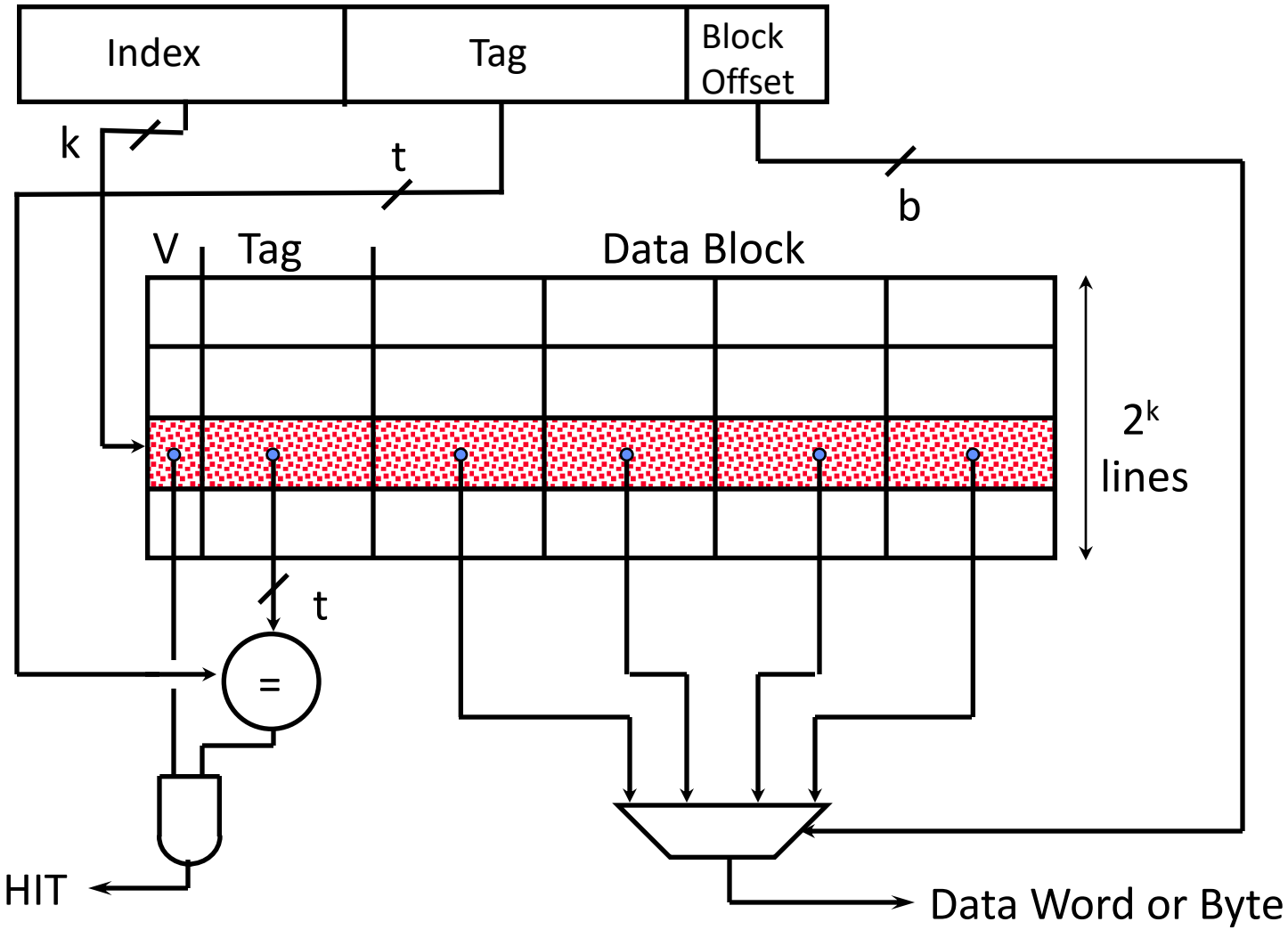


Direct-Mapped Cache

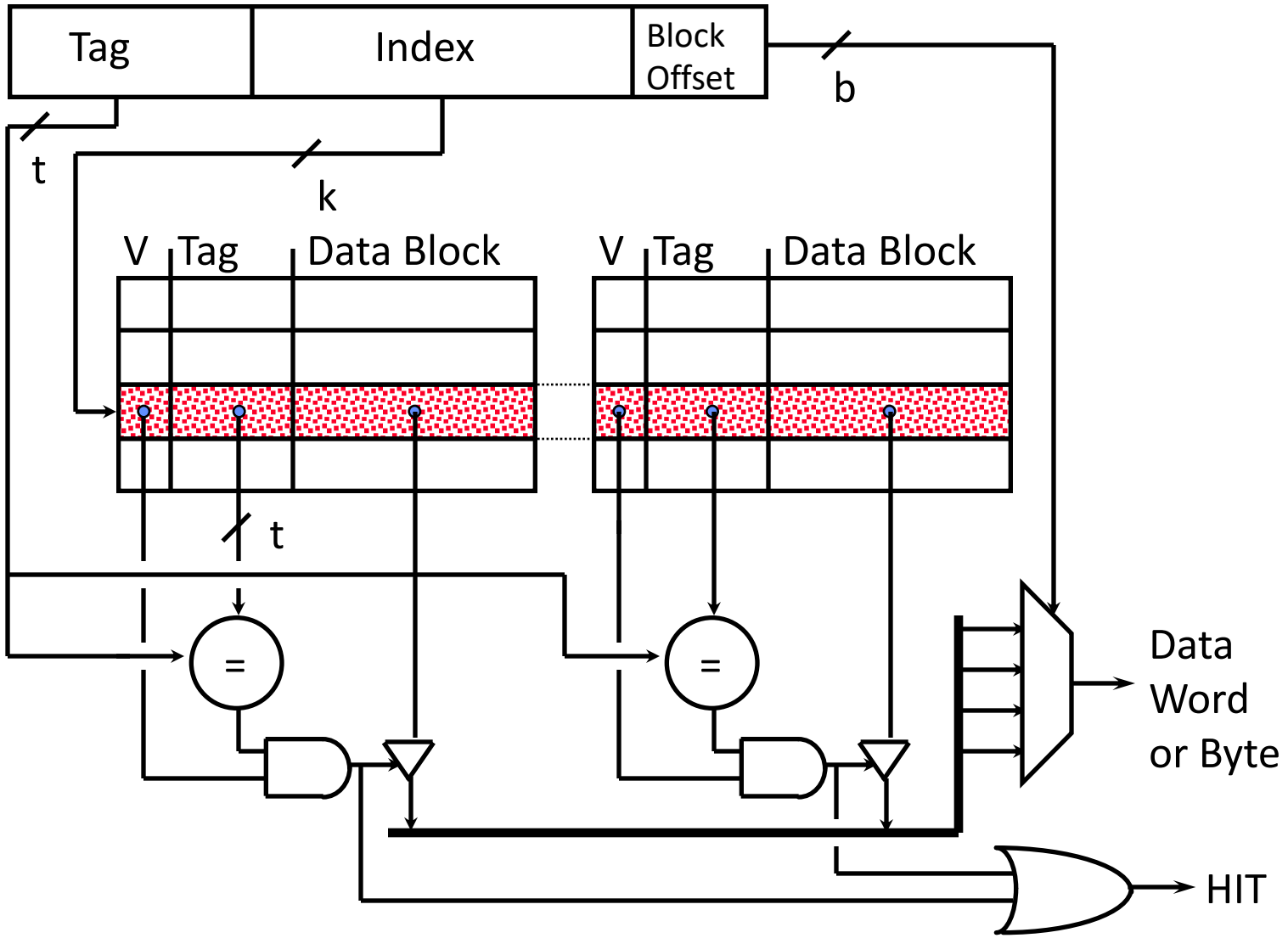


Direct Map Address Selection

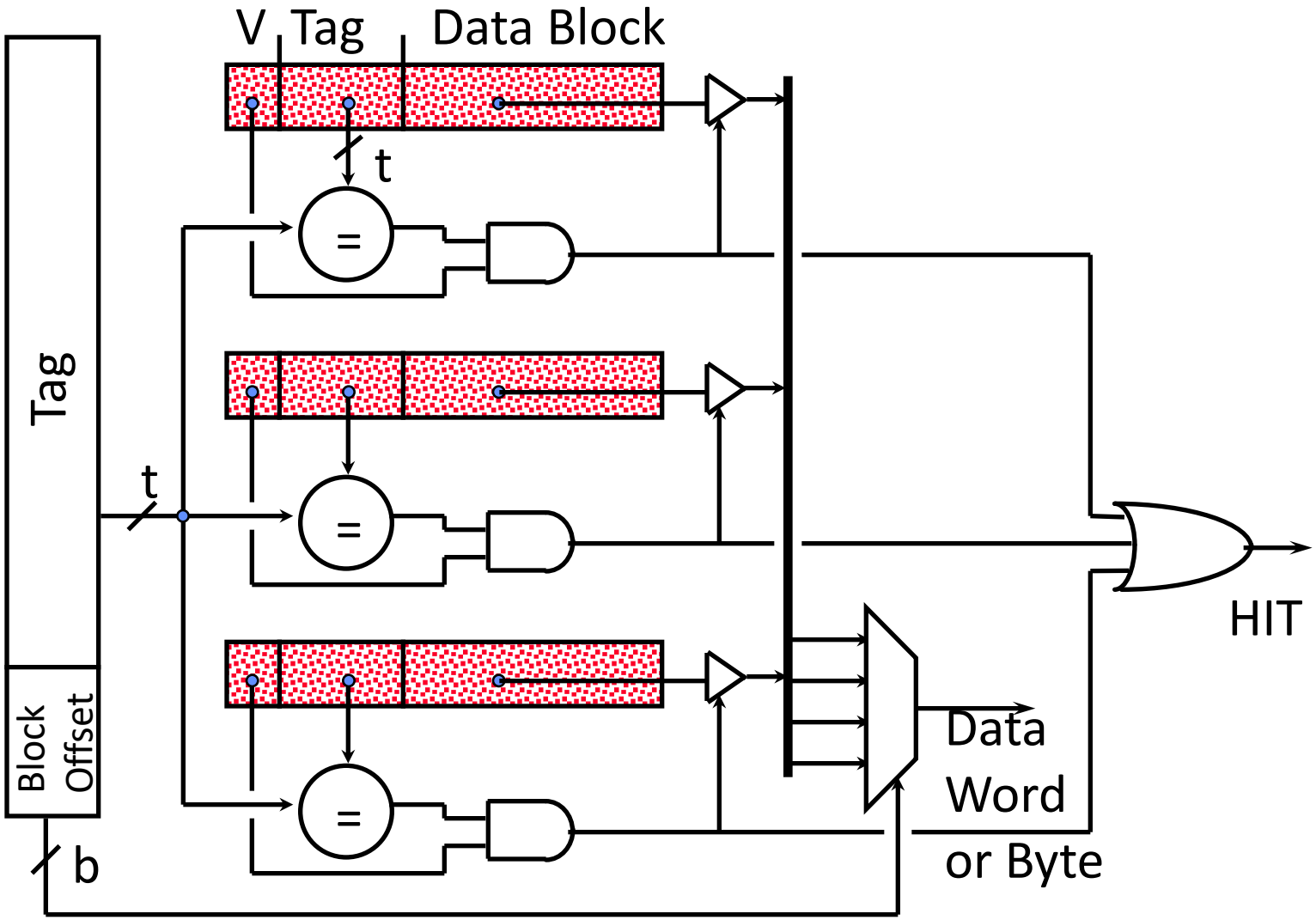
higher-order vs. lower-order address bits



2-Way Set-Associative Cache



Fully Associative Cache



Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)