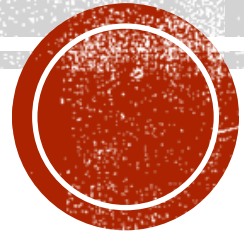
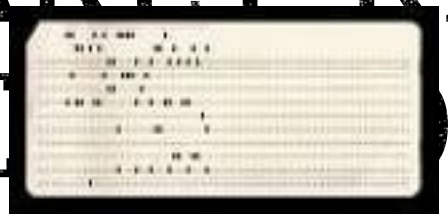
# ARCHITECTURE OF COMPUTER SYSTEMS LECTURE 6 - MEMORY



# LAST TIME IN LECTURE 5

- Control hazards (branches, interrupts) are most difficult to handle as they change which instruction should be executed next
- Branch delay slots make control hazard visible to software, but not portable to more advanced μarchs
- Speculation commonly used to reduce effect of control hazards (predict sequential fetch, predict no exceptions, branch prediction)
- Precise exceptions: stop cleanly on one instruction, all previous instructions completed, no following instructions have changed architectural state
- To implement precise exceptions in pipeline, shift faulting instructions down pipeline to “commit” point, where exceptions are handled in program order

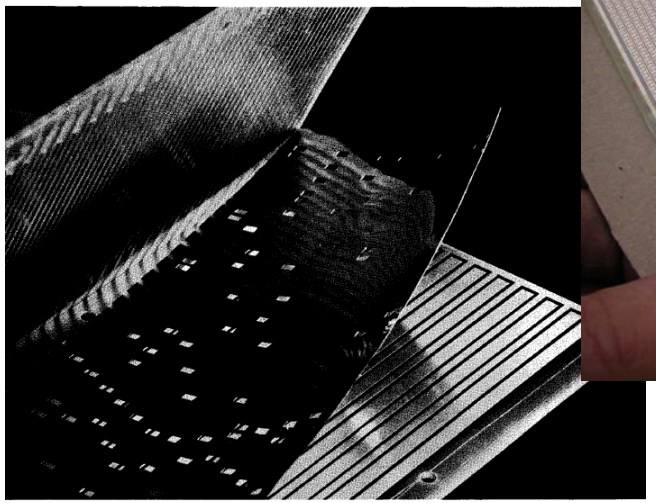
# EARLY READ ONLY MEMORY TE LO



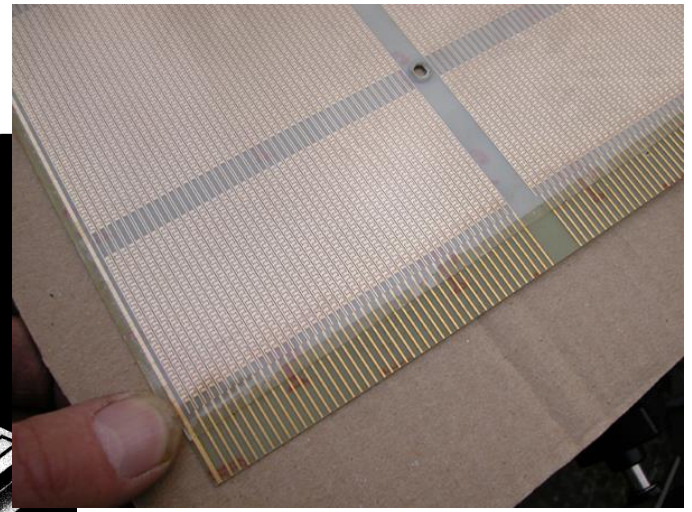
Punched cards, From early 1700s through Jaquard Loom, Babbage, and then IBM

Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 µcode store



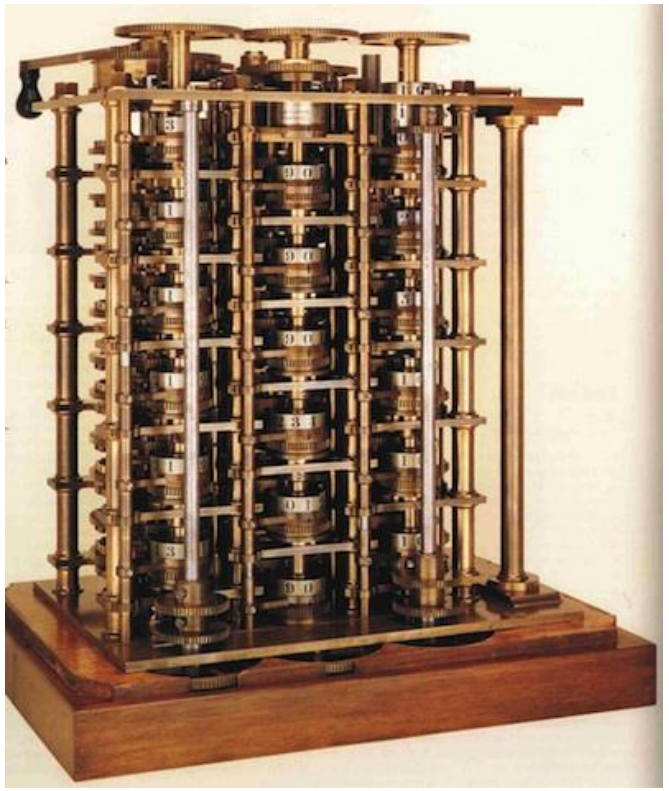
IBM Card Capacitor ROS

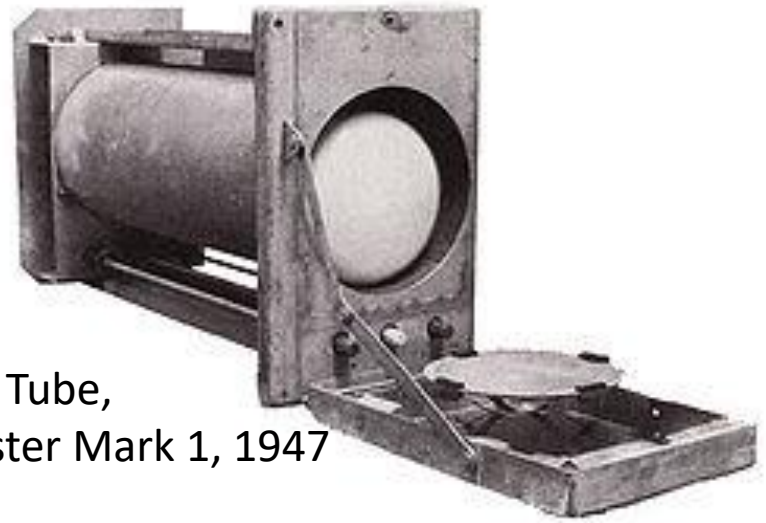


IBM Balanced Capacitor ROS

# MAIN MEMORY TECHNOLOGIES

Babbage, 1800s. Digits stored on mechanical wheels



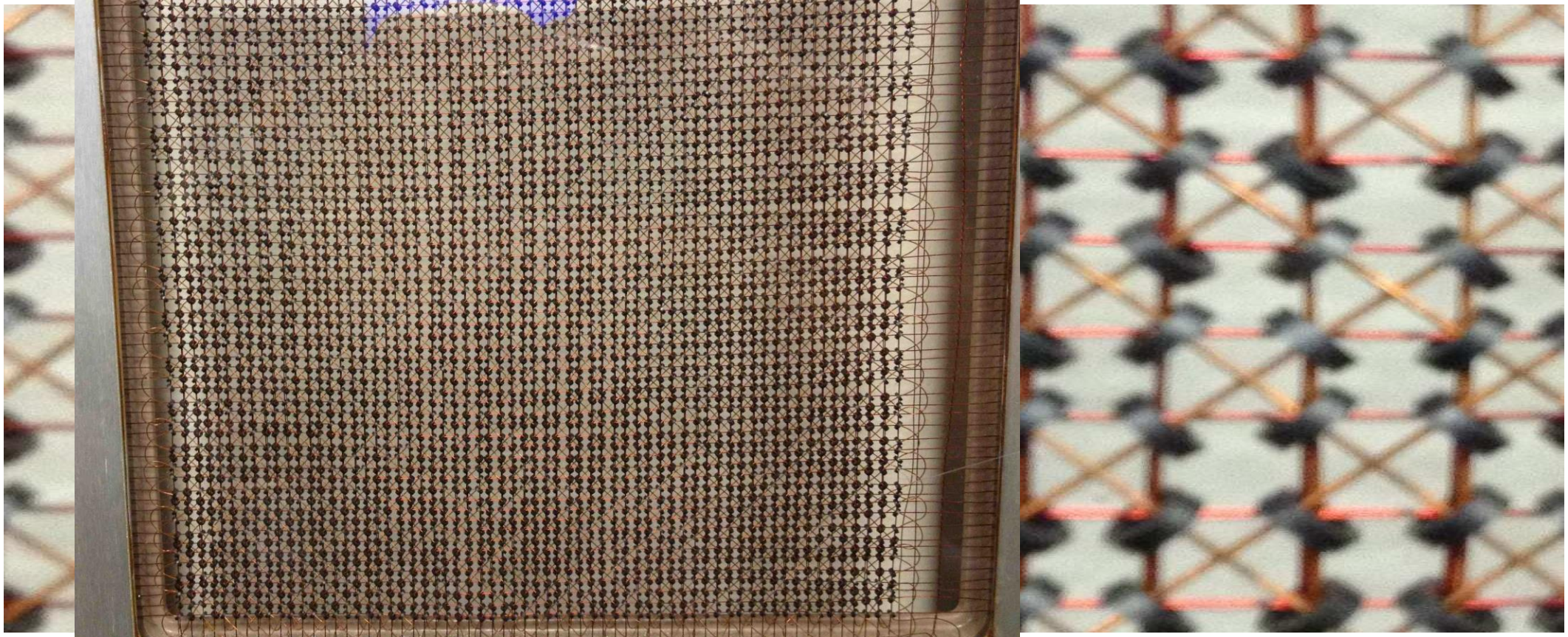


Williams Tube, Manchester Mark 1, 1947

Mercury Delay Line, Univac 1, 1951

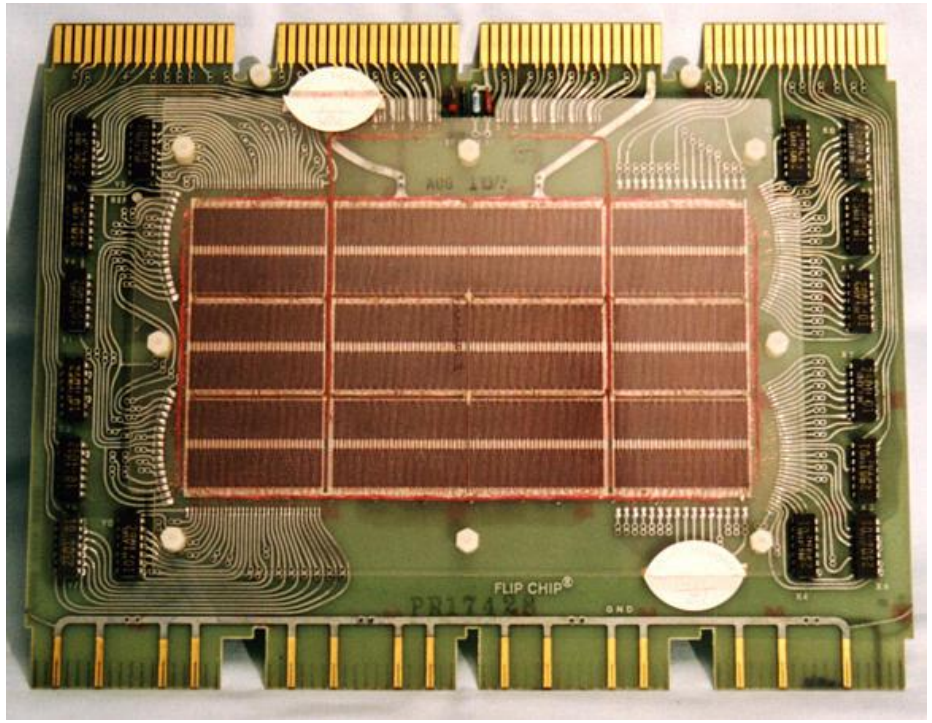

Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

# MIT WHIRLWIND CORE M



# 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)

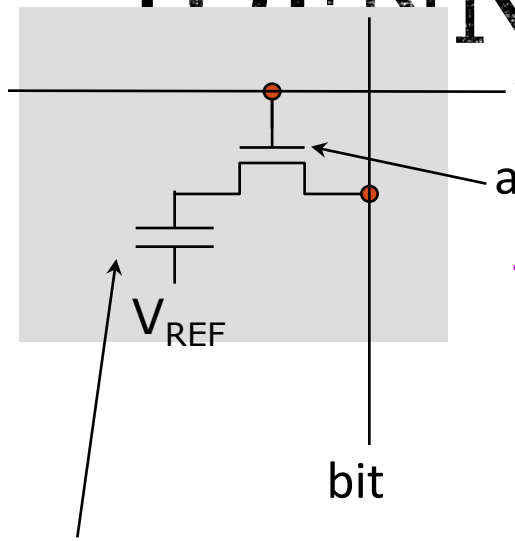
# SEMICONDUCTOR MEMORY

- 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).

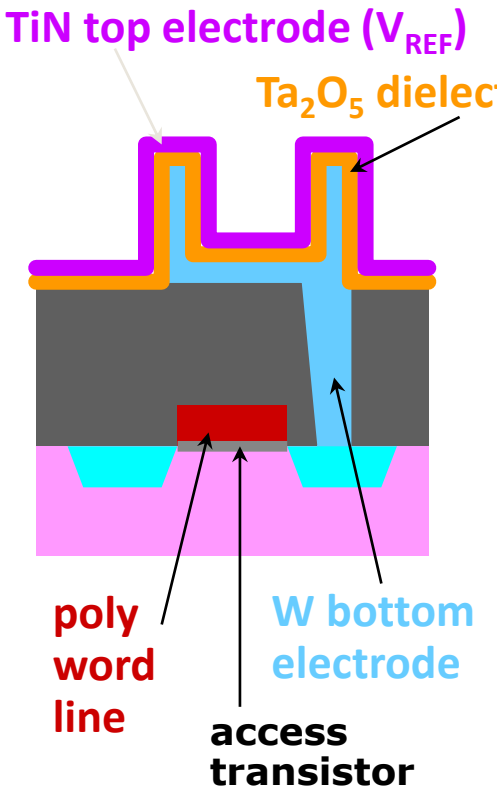
- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

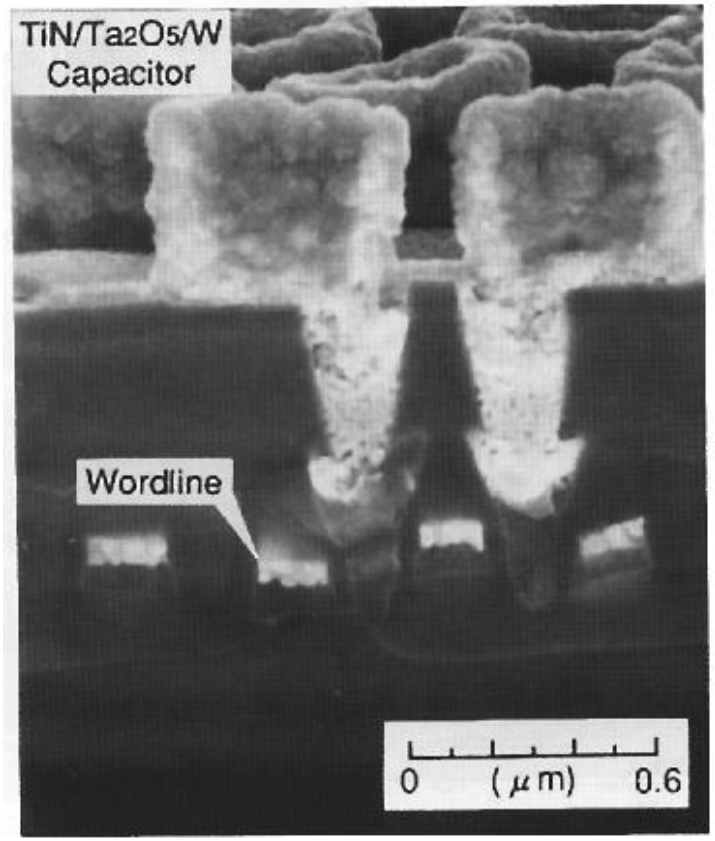
*Semiconductor memory quickly replaced core in '70s*

# ONE-TRANSISTOR DYNAMIC RAM [DENNARD, IBM]

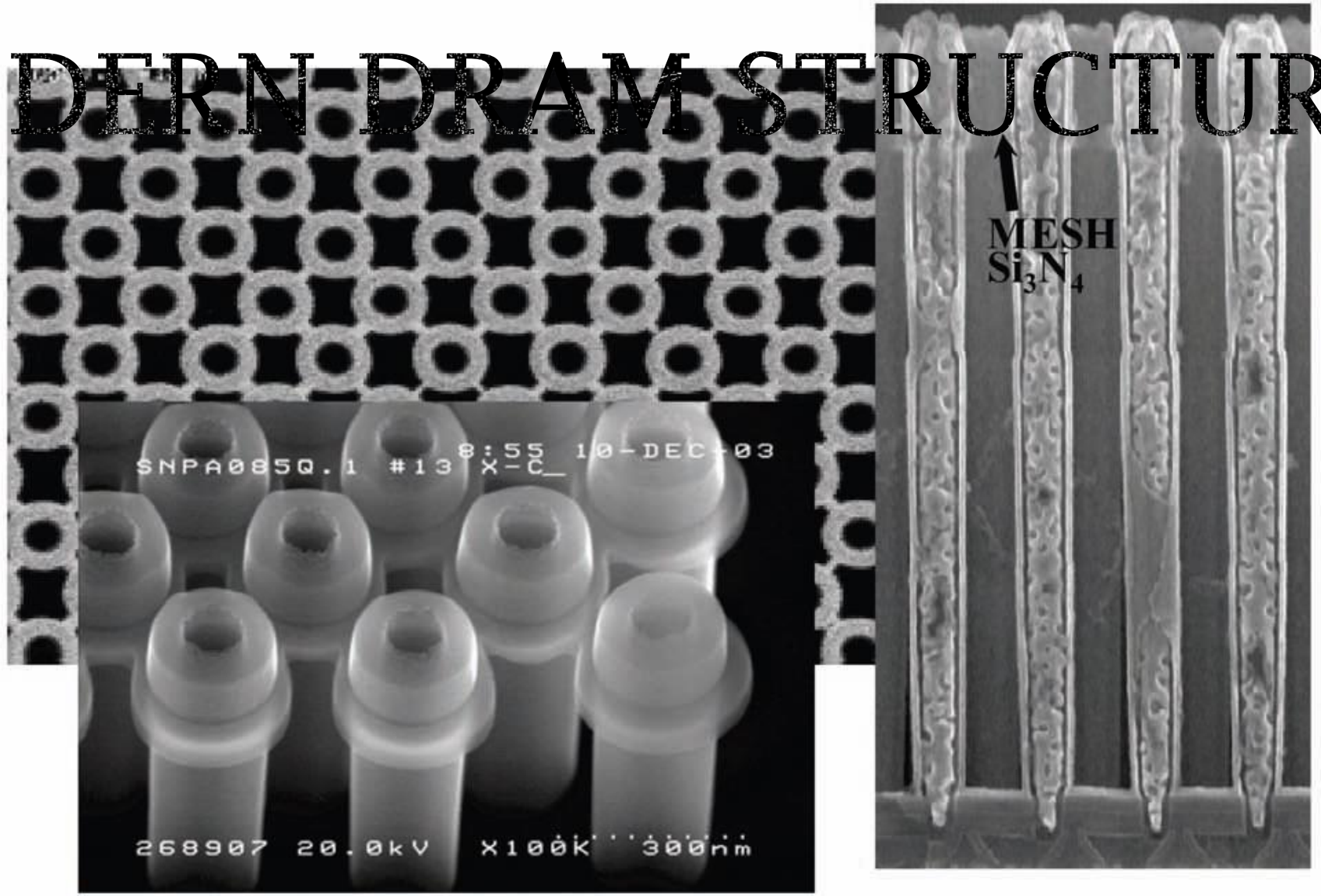



Storage capacitor (FET gate, trench, stack)



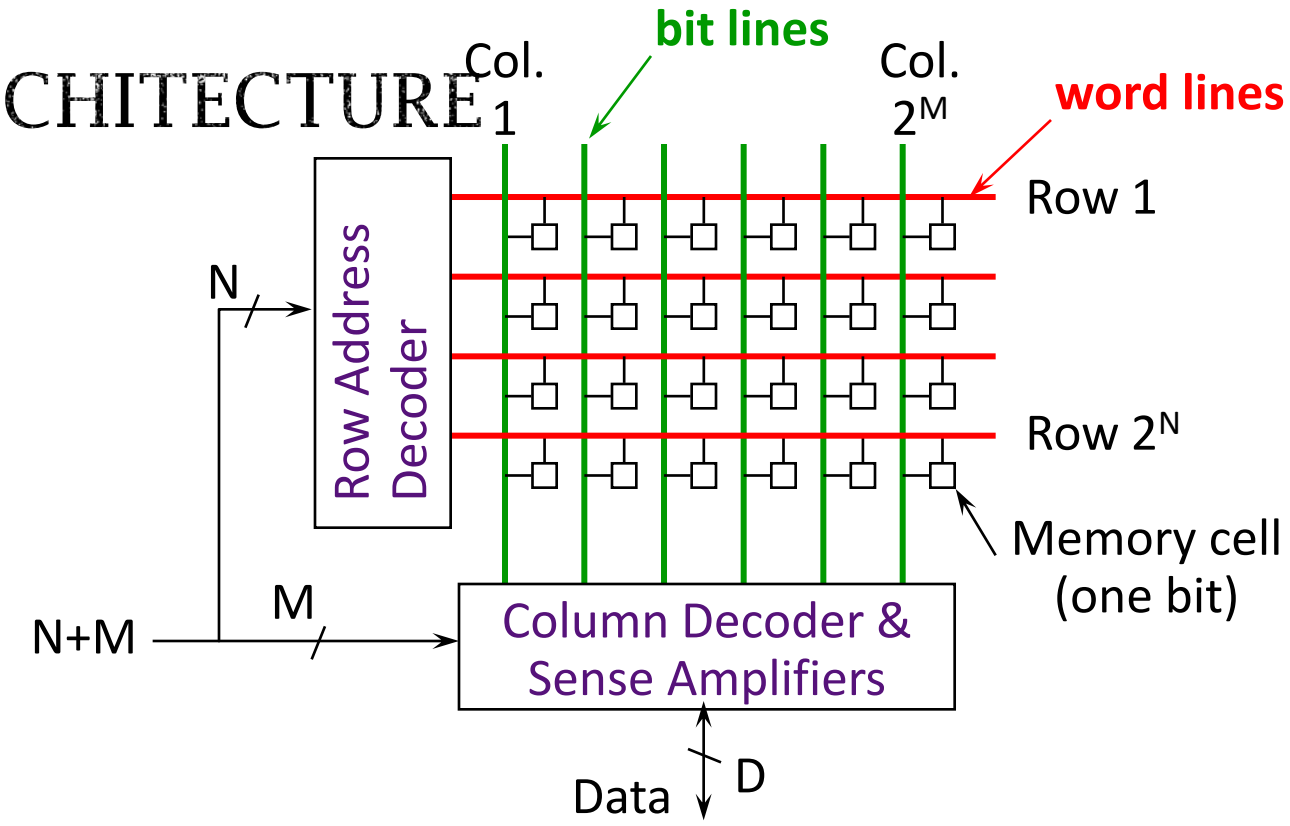



# MODERN DRAM STRUCTURE



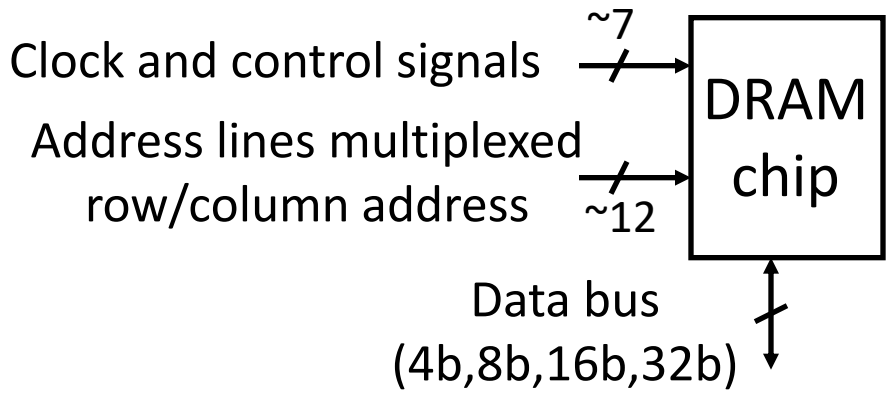

[Samsung, sub-70nm DRAM, 2004]

# DRAM ARCHITECTURE

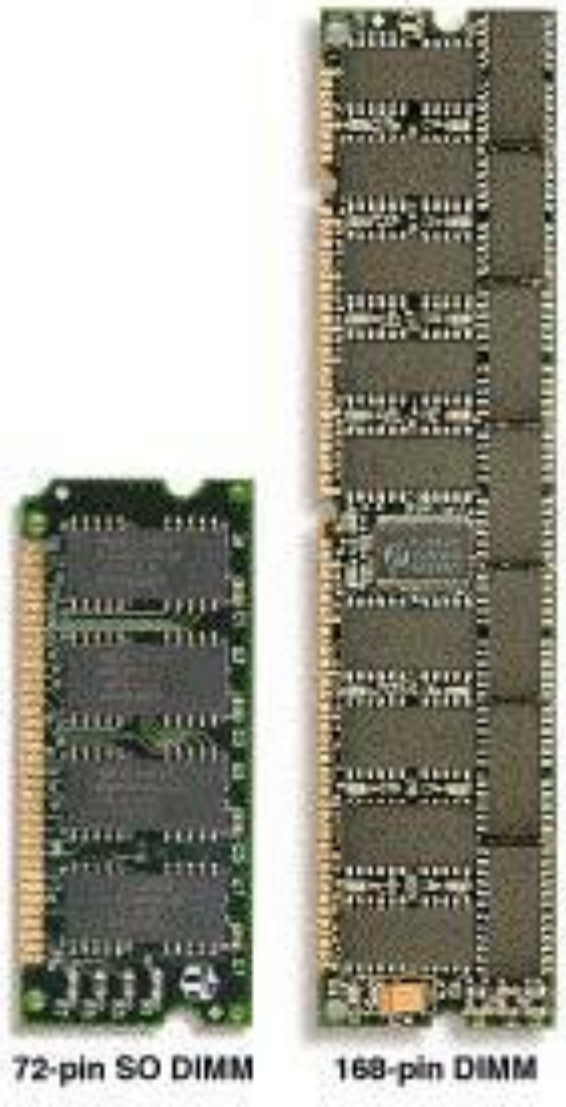



- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

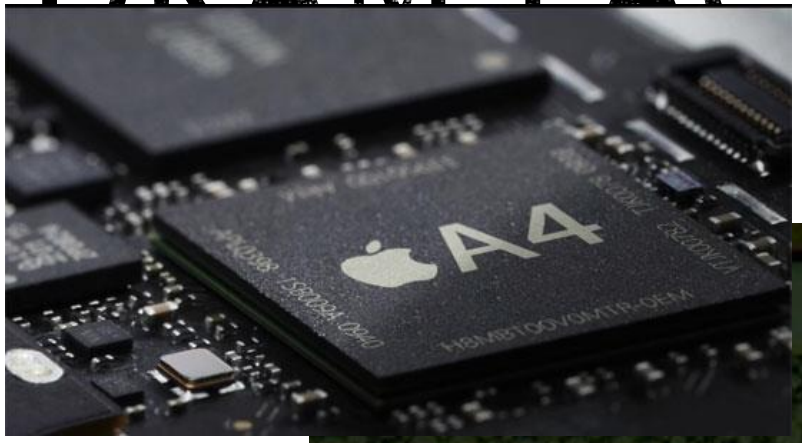
# DRAM PACKAGING
## (LAPTOPS/DESKTOPS/SERVERS)



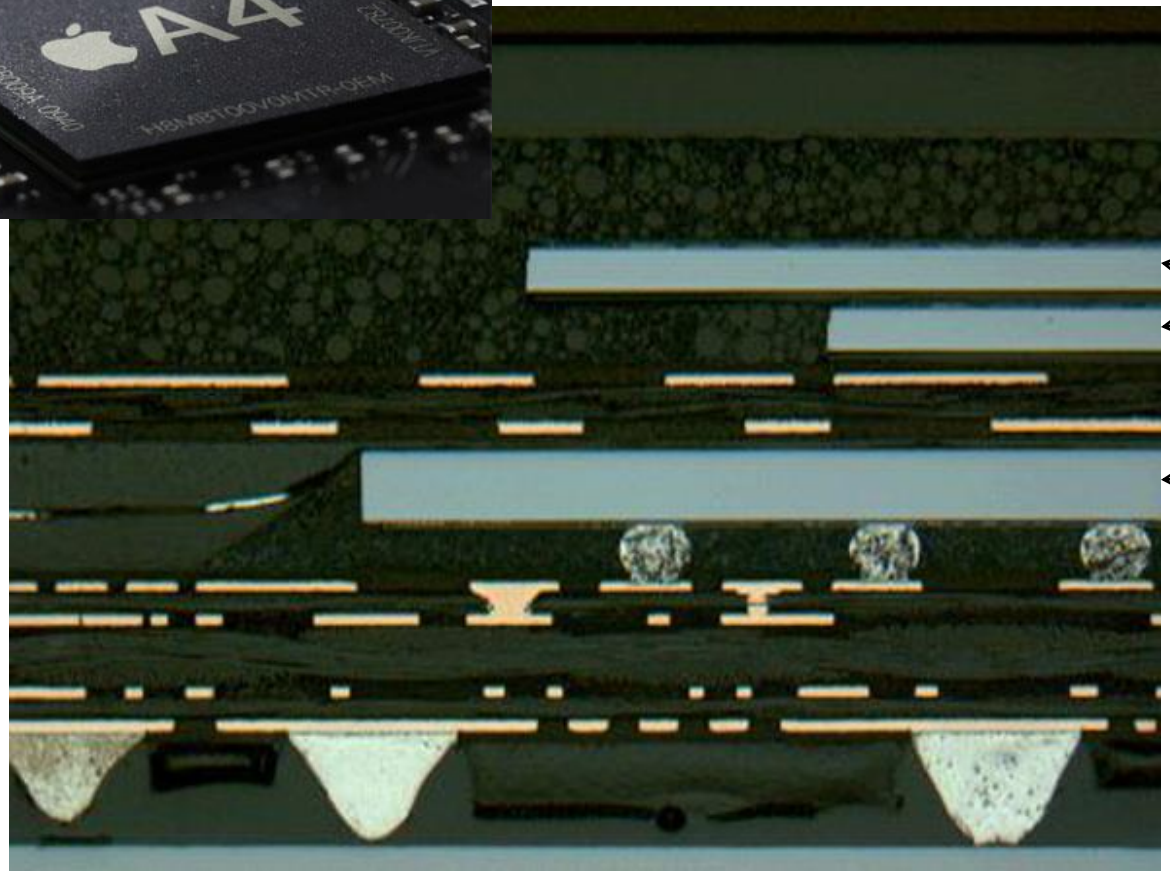

- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)



# DRAM PACKAGING, MOBILE




*[ Apple A4 package on circuit board]*



Two stacked DRAM die

Processor plus logic die

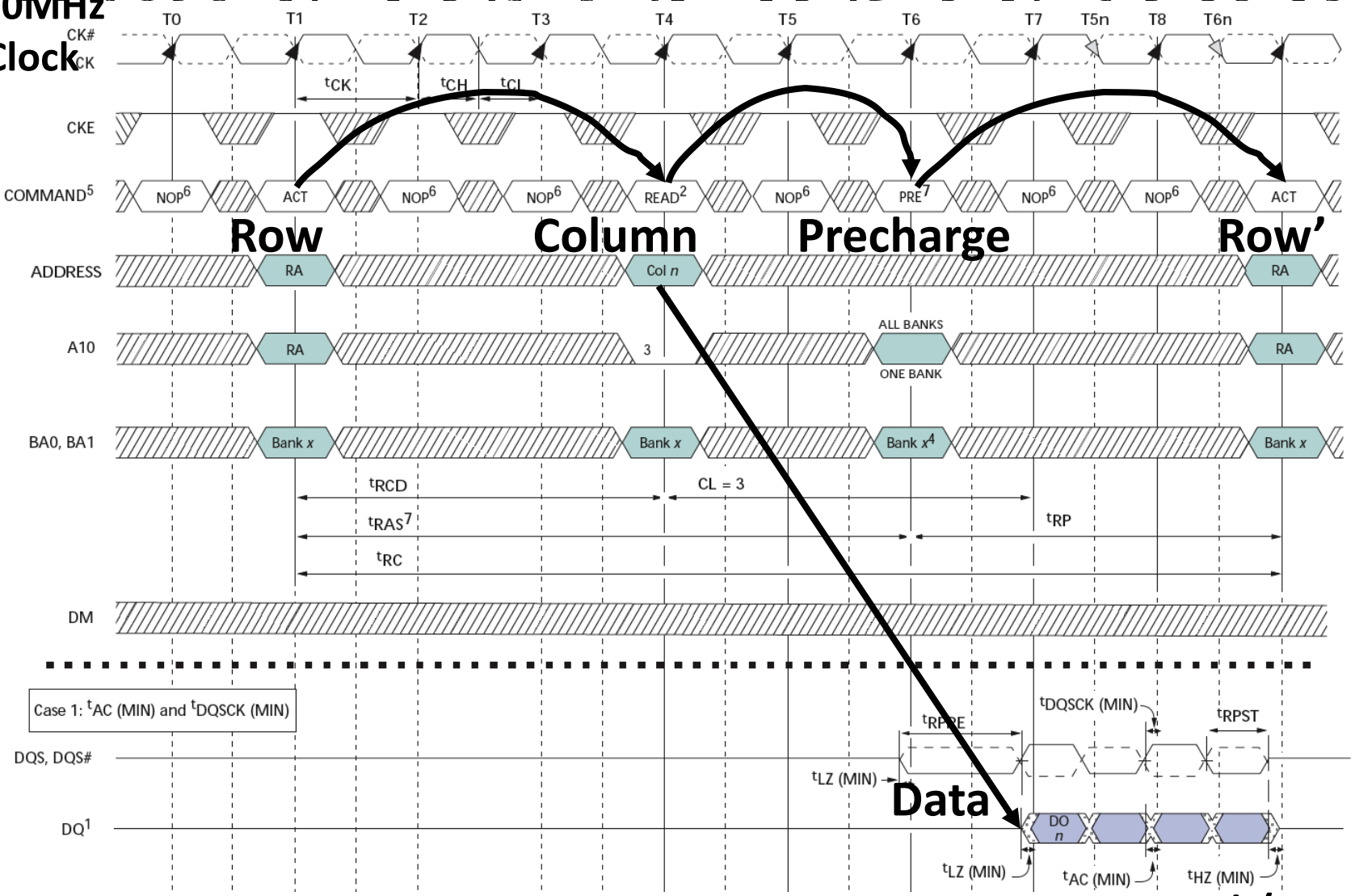
*[ Apple A4 package cross-section, iFixit 2010 ]*

# DRAM OPERATION

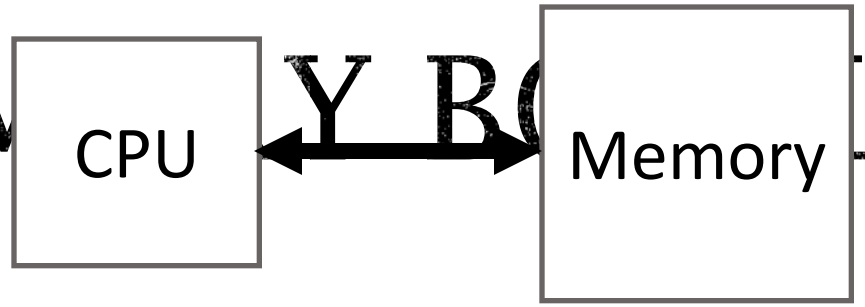
- Three steps in read/write access to a given bank
- Row access (RAS)
  - decode row address, enable addressed row (often multiple Kb in row)
  - bitlines share charge with storage cell
  - small change in voltage detected by sense amplifiers which latch whole row of bits
  - sense amplifiers drive bitlines 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) DI

- [ M



# CPU-MEMORY BOTTLENECK




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

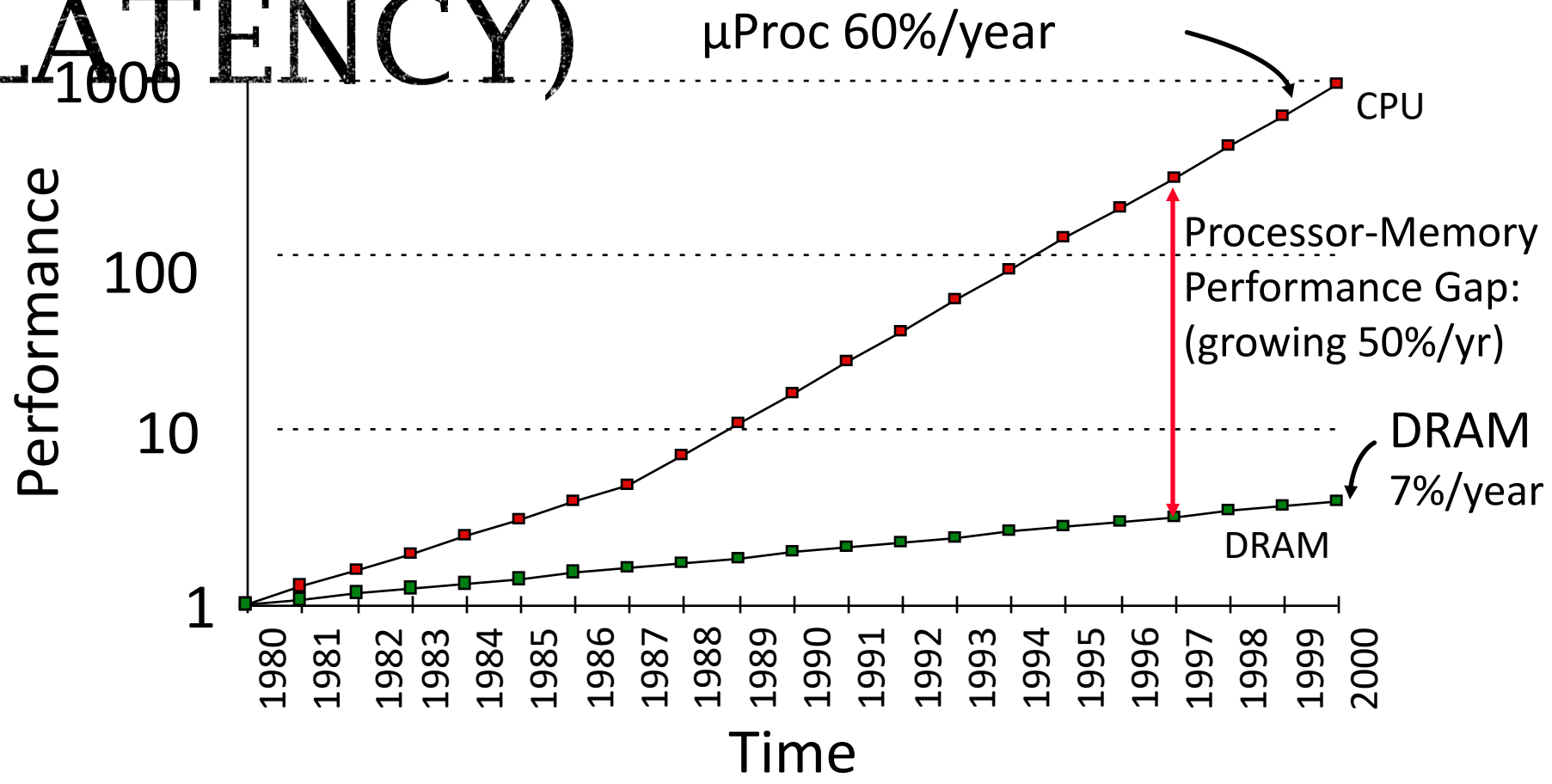
- Latency (time for a single access)
  - Memory access time >> Processor cycle time
- Bandwidth (number of accesses per unit time)

  if fraction m of instructions access memory

  ⇒ 1+m memory references / instruction

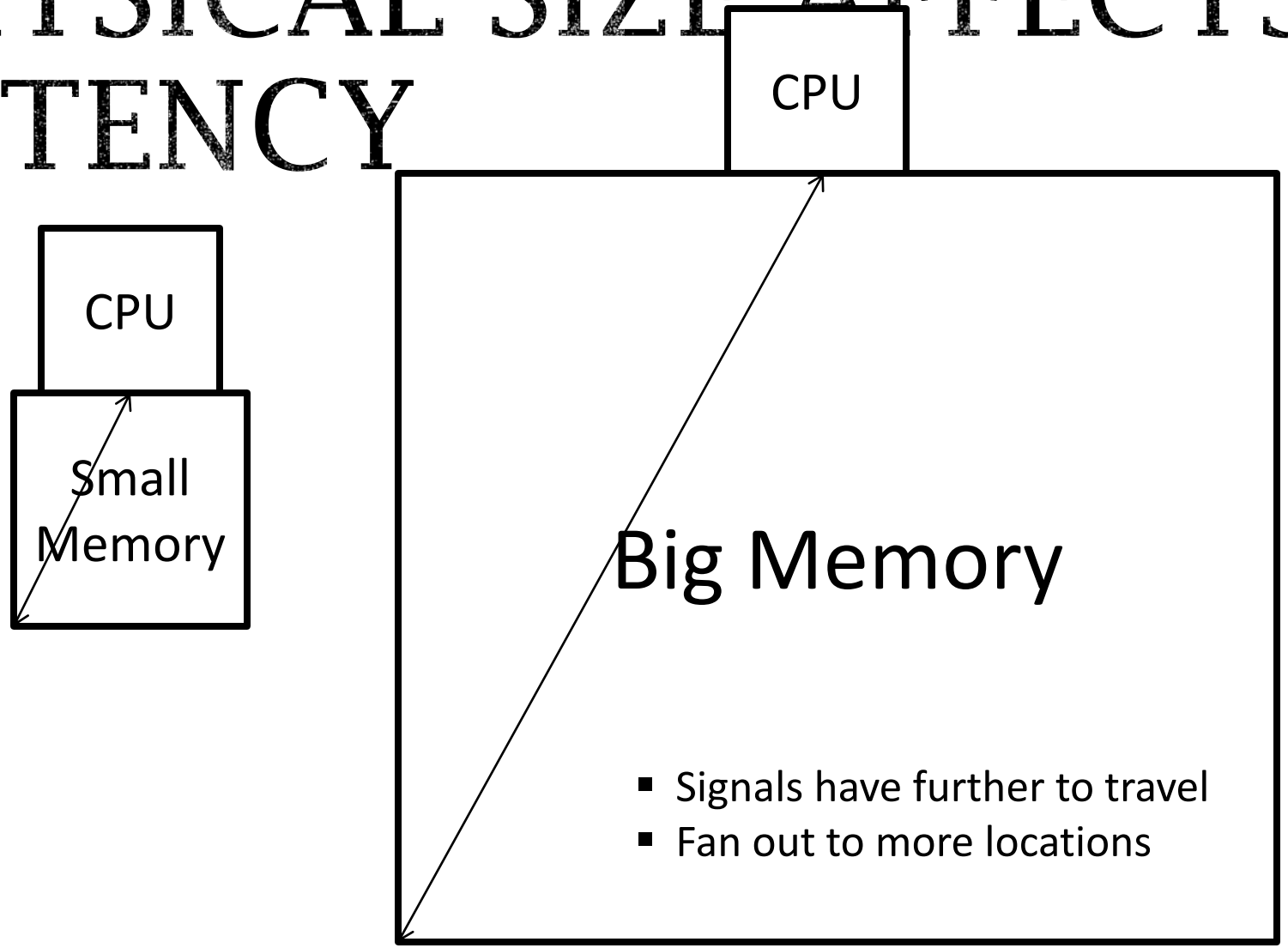
  ⇒ CPI = 1 requires 1+m memory refs / cycle (assuming RISC-V ISA)

# PROCESSOR-DRAM GAP (LATENCY)




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

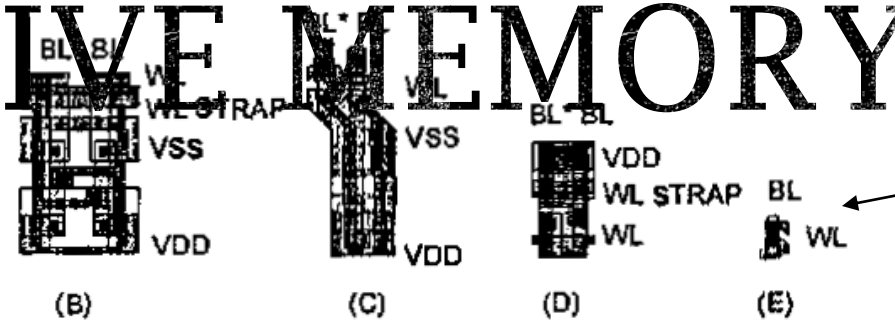
# PHYSICAL SIZE AFFECTS LATENCY

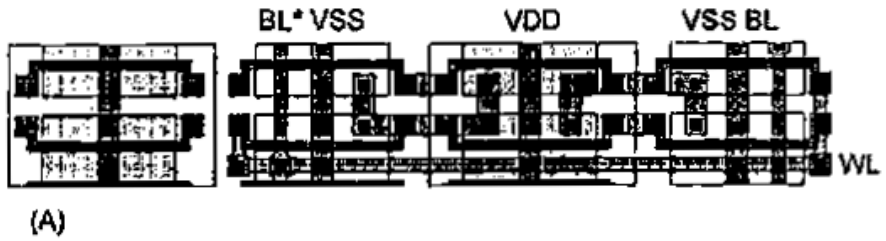


# RELATIVE MEMORY CELL SIZES

On-Chip SRAM in logic chip

DRAM on memory chip




1 Memory cell in 0.5µm processes

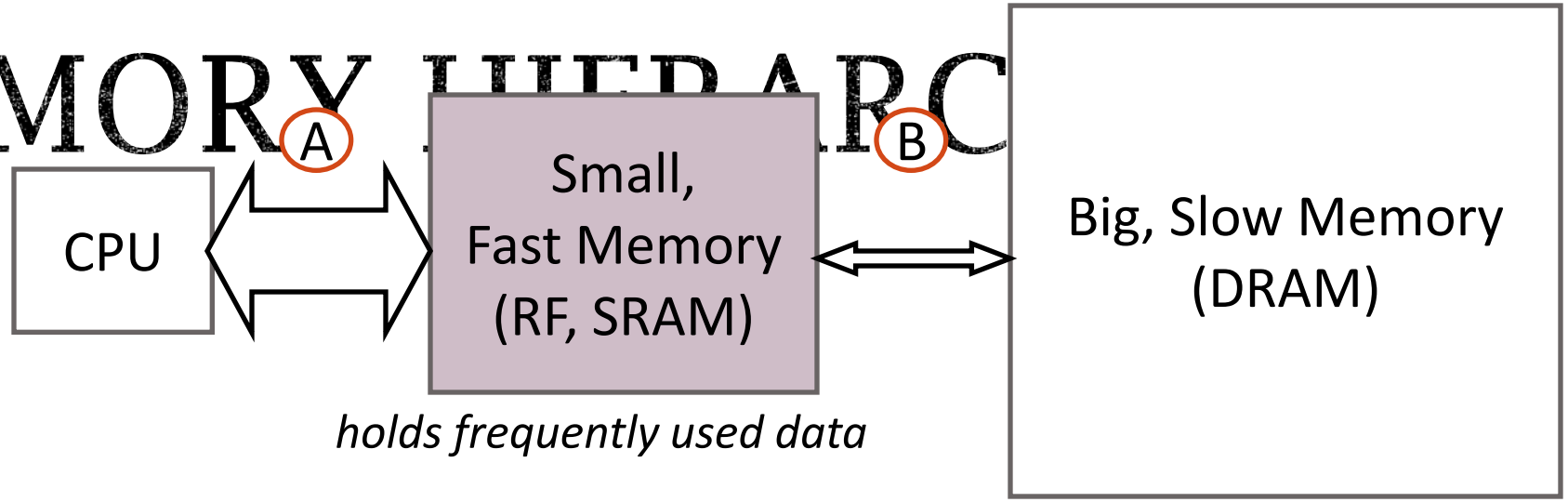
a) Gate Array SRAM
b) Embedded SRAM
c) Standard SRAM (6T cell with local interconnect)
d) ASIC DRAM
e) Standard DRAM (stacked cell)

*[ Foss, "Implementing Application-Specific Memory", ISSCC 1996 ]*

| Memory | Process | Cell size ($\mu m^2$) | Cell efficiency | Bits in $100mm^2$ ($10^6$) | Gate size ($\mu m^2$) | Gate utilization | Gates in $100mm^2$ ($10^6$) |
|---|---|---|---|---|---|---|---|
| Gate array SRAM | 3-metal ASIC | 370 | 80% | 216 | 185 | 70% | 378 |
| Embedded SRAM | 3-metal ASIC | 67 | 70% | 1045 | 185 | 70% | 378 |
| Standard SRAM | 2-metal 6T local int. | 43 | 65% | 1512 | 245 | 40% | 163 |
| Embedded ASIC-DRAM | 3-metal ASIC | 23 | 60% | 2609 | 185 | 70% | 378 |
| Standard DRAM | 2-metal stacked cell | 3.2 | 50% | 15625 | 411 | 40% | 97 |

**Table 1: Memory and logic density for a variety of 0.5µm implementations.**

# MEMORY HIERARC




- *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)*

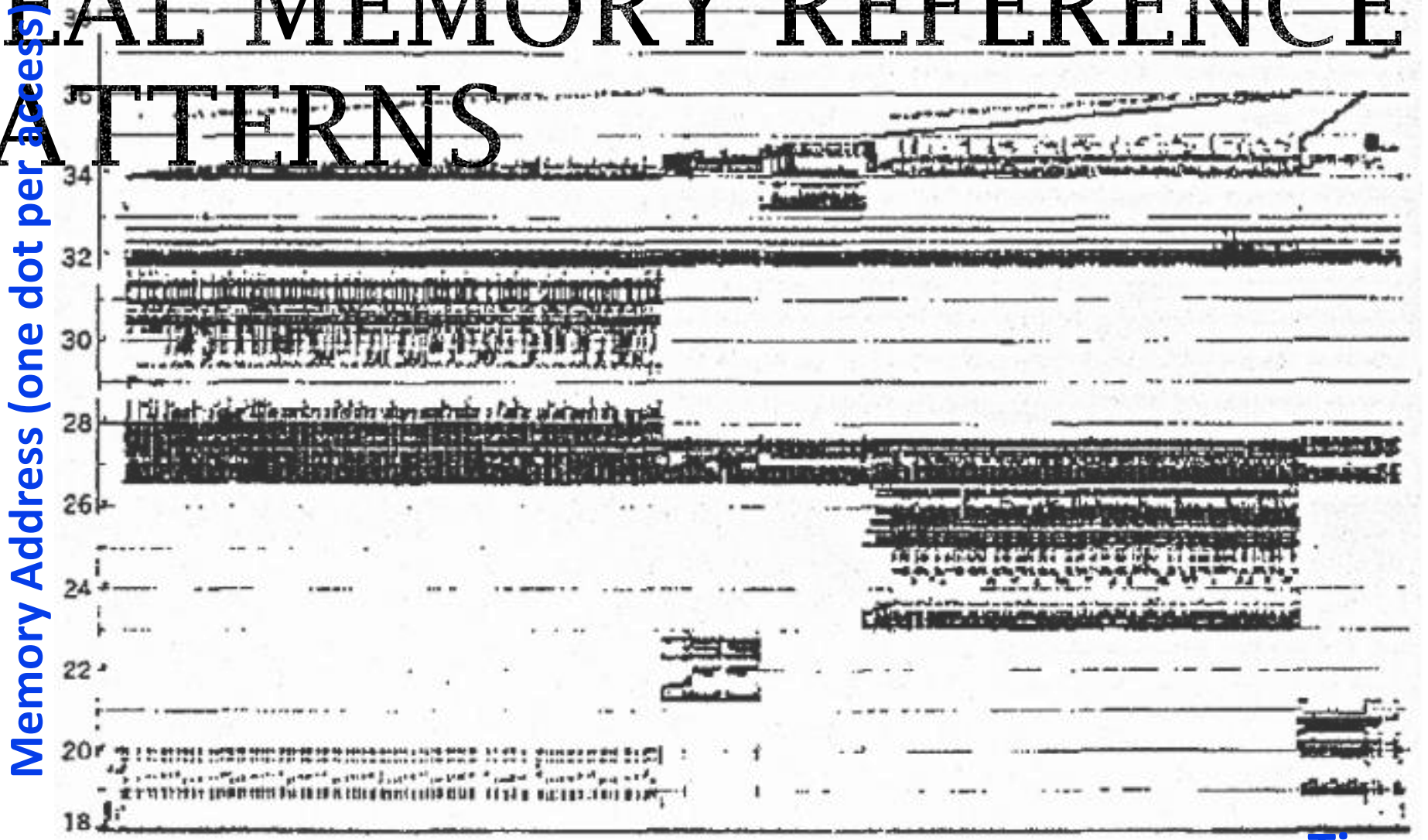
*if* data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

# 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*

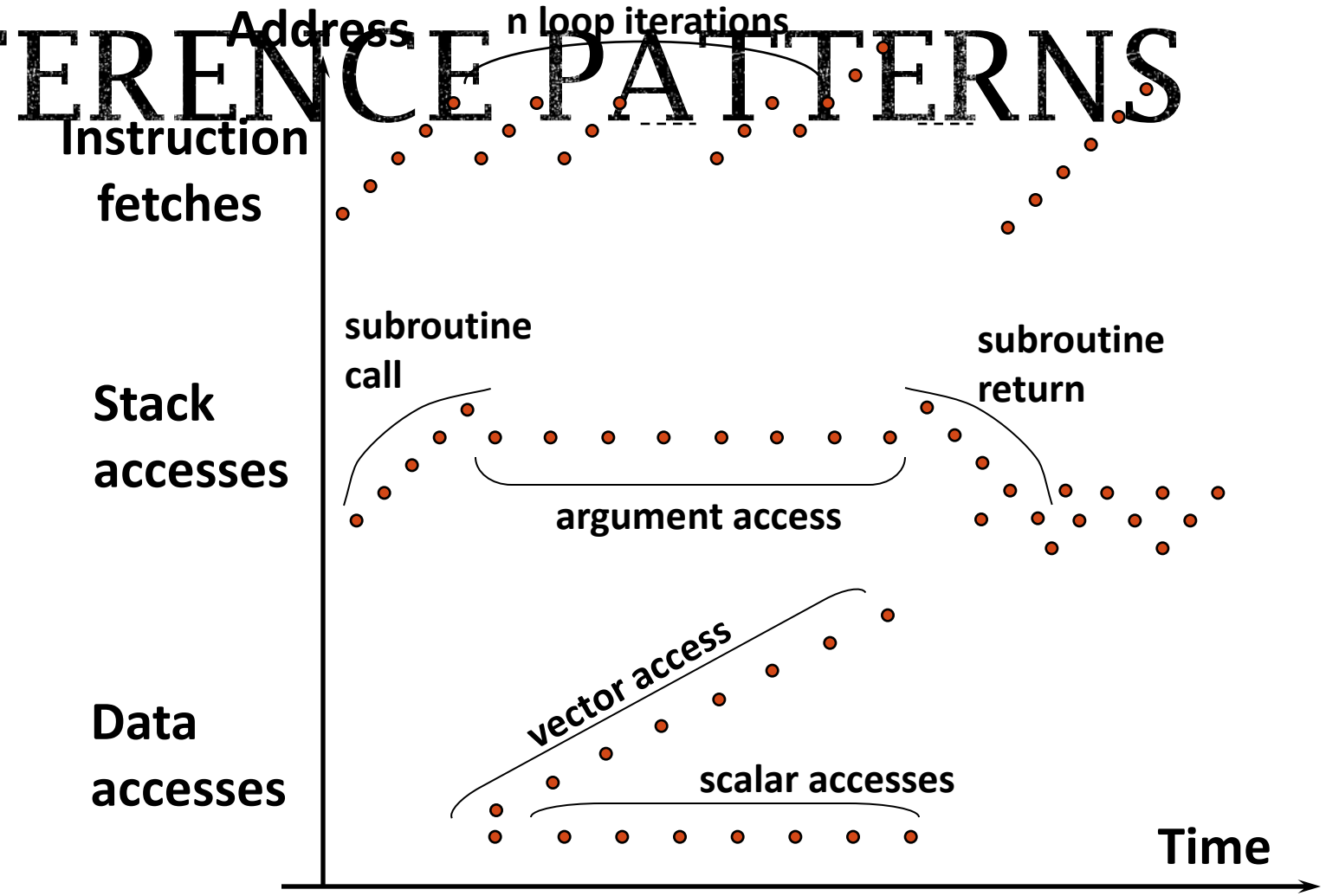
- Larger/slower storage, e.g., main memory
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
    - *but software may provide "hints", e.g., don't cache or prefetch*

# 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

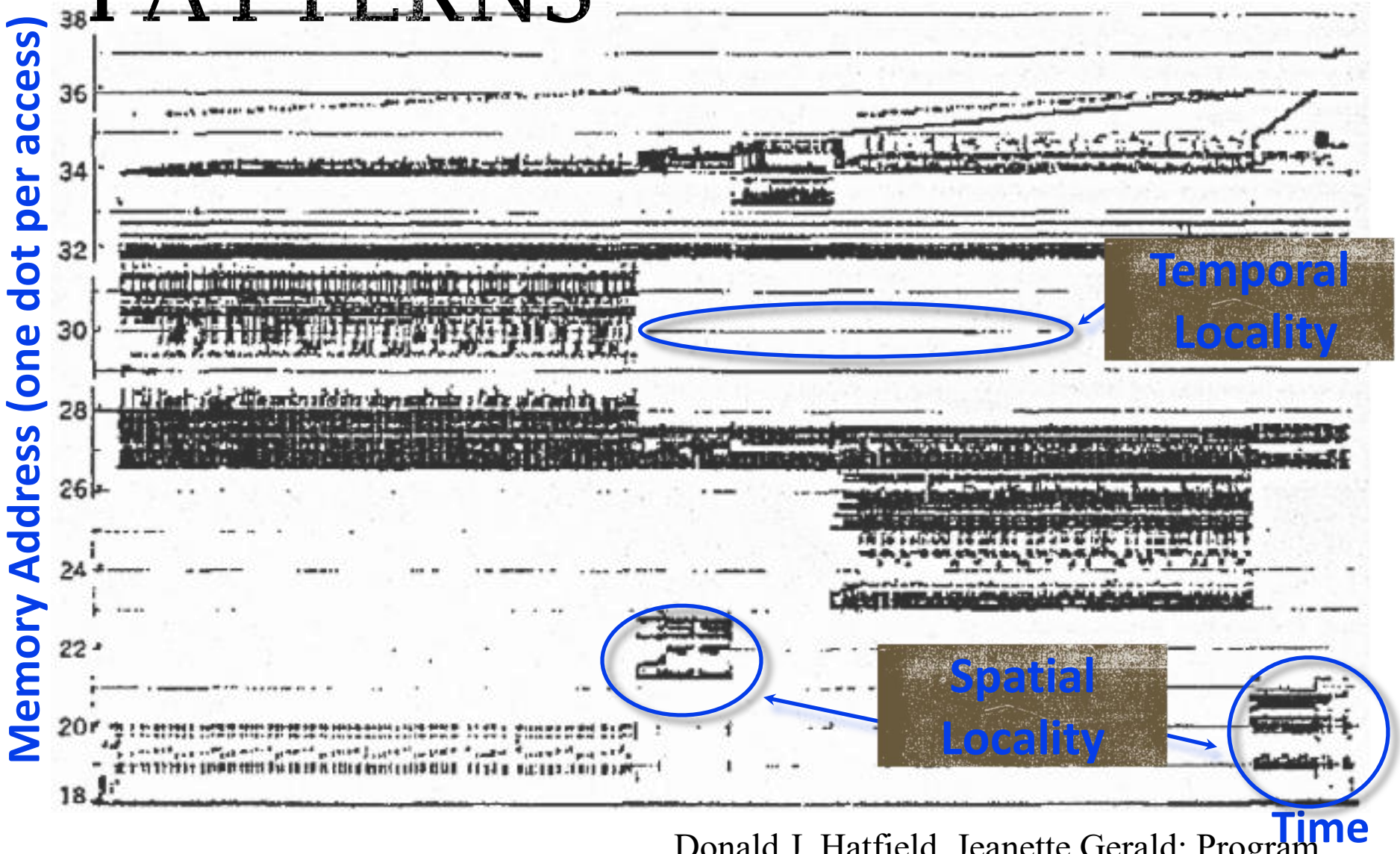


# TWO PREDICTABLE PROPERTIES OF MEMORY REFERENCES:

- **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

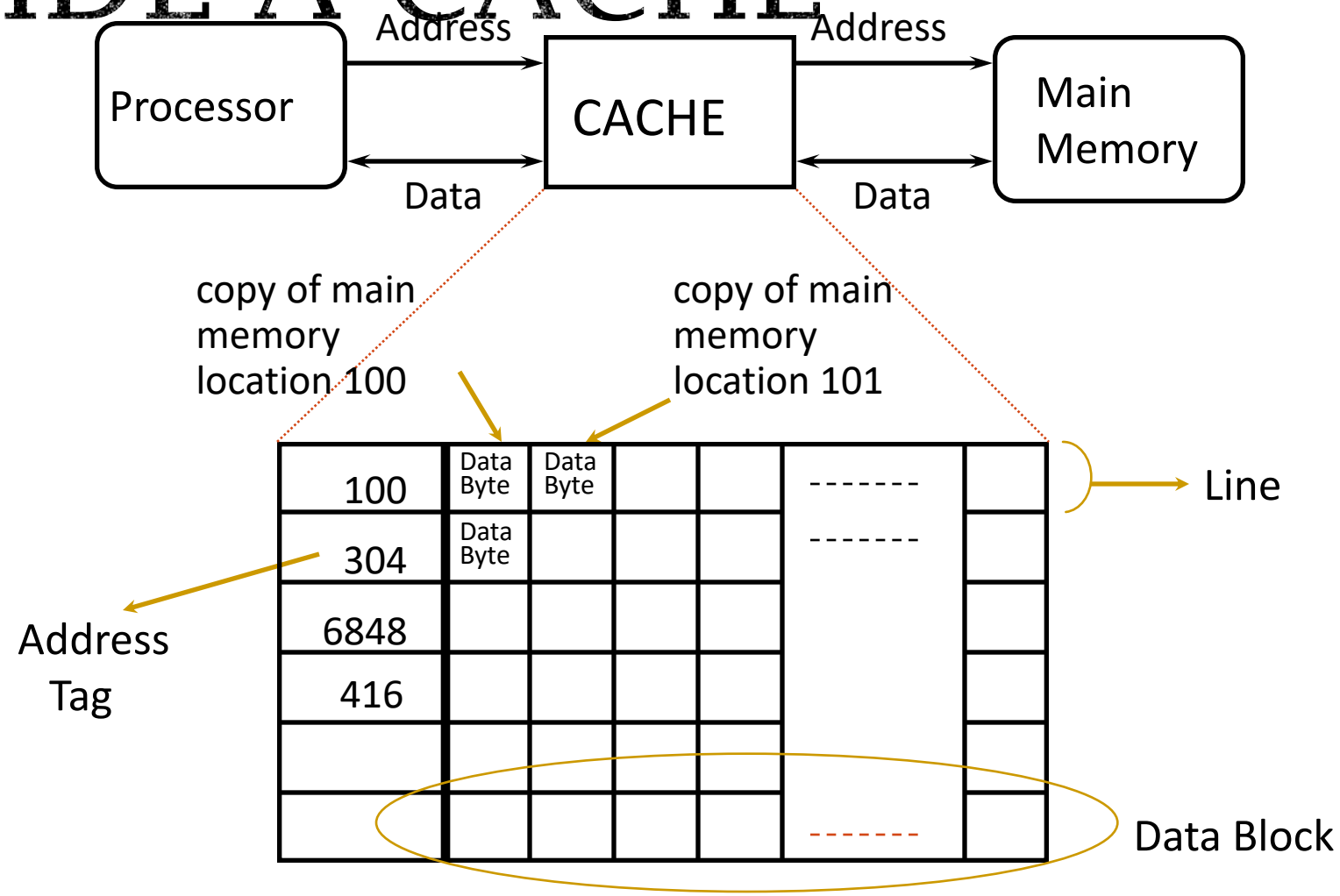



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

# 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

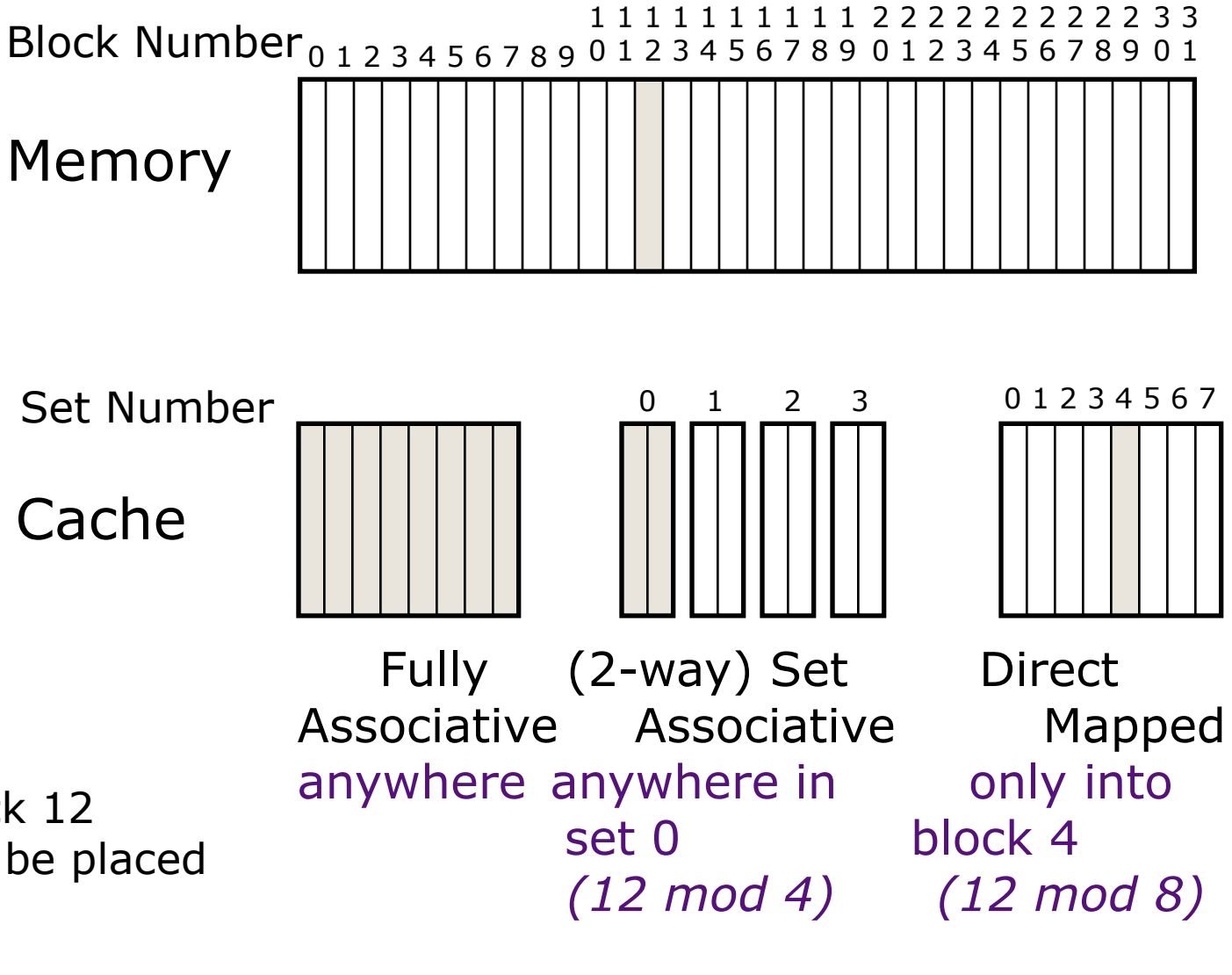
Read block of data from
Main Memory

Wait …

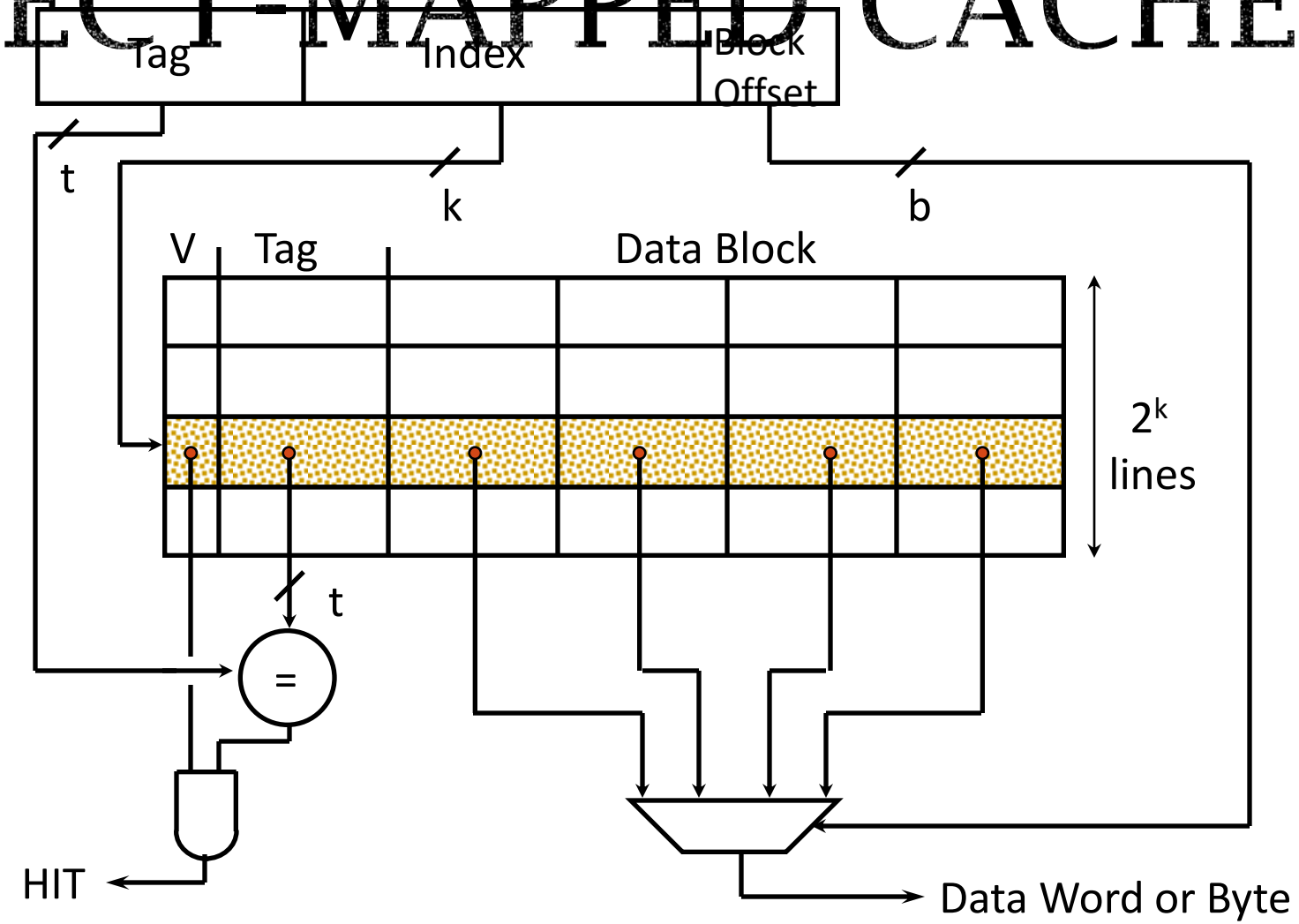
Return data to processor
and update cache

Q: Which line do we replace?

# PLACEMENT POLICY

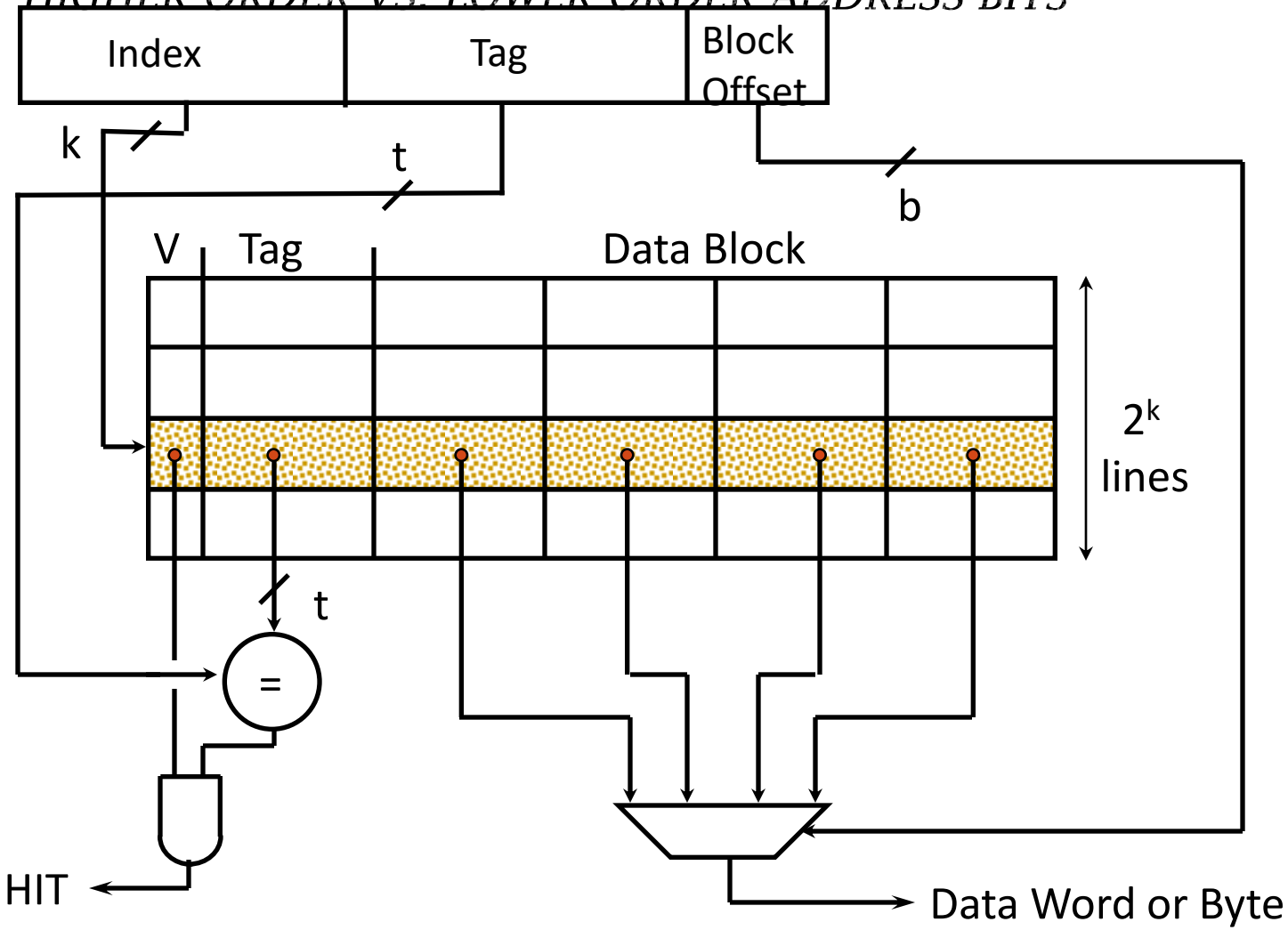


# DIRECT-MAPPED CACHE

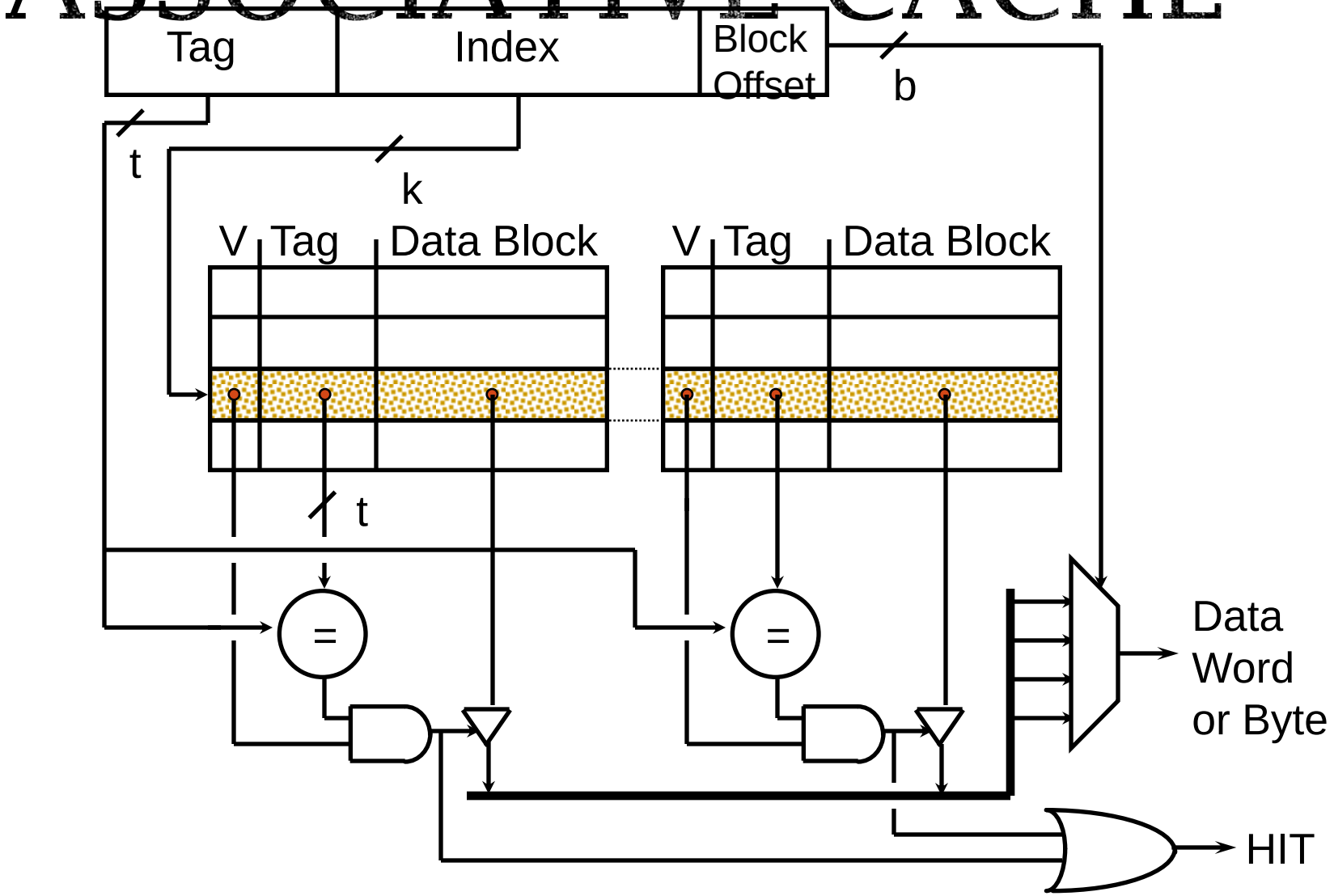


# DIRECT MAP ADDRESS SELECTION

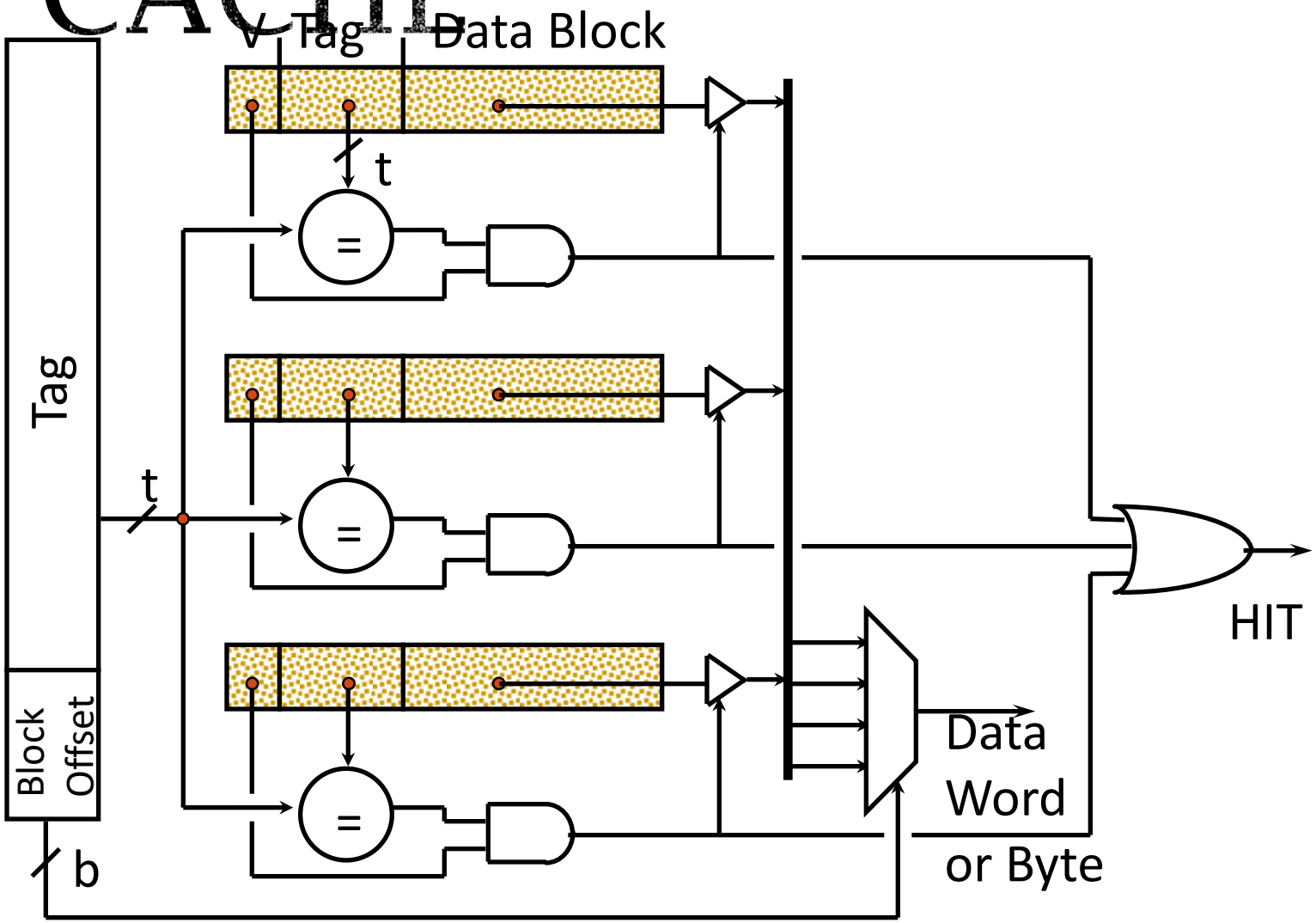
*HIGHER-ORDER VS. LOWER-ORDER ADDRESS BITS*



# 2-WAY SET-ASSOCIATIVE CACHE



# FULLY ASSOCIATIVE CACHE



# REPLACEMENT POLICY

In an associative cache, which block from a set should be evicted when the set becomes full?

- Random
- Least-Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree often used for 4-8 way
- First-In, First-Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches
- Not-Most-Recently Used (NMRU)
  - FIFO with exception for most-recently used block or blocks

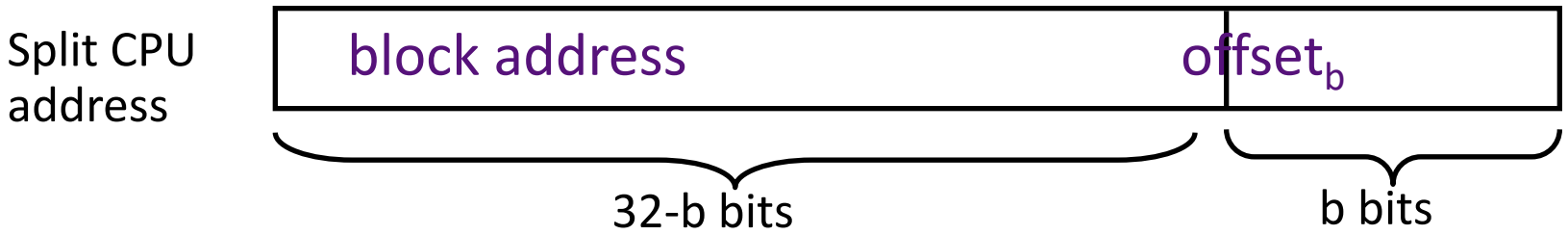
*This is a second-order effect. Why?*

*Replacement only happens on misses*

# BLOCK SIZE AND SPATIAL LOCALITY

Block is unit of transfer between the cache and memory


4 word block, b=2




$2^b$ = block size *a.k.a* line size (in bytes)

Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

*What are the disadvantages of increasing block size?*

*Fewer blocks => more conflicts. Can waste bandwidth.*

# ACKNOWLEDGEMENTS


- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252