CS 152 Computer Architecture and Engineering

Lecture 6 - Memory

Dr. George Michelogiannakis
EECS, University of California at Berkeley
CRD, Lawrence Berkeley National Laboratory

http://inst.eecs.berkeley.edu/~cs152

CS152 Administritivia

- PS 1 due on Wednesday's class
- Lab 1 also due at the same time

- Hand paper reports or email

- PS 2 will be released on Wendesday
- Lab 2 Wednesday or Thursday

- Quiz next week Wednesday (17^{th})

- Discussion section on Thursday to cover lab 2 and PS 1

Question of the Day

- Can a cache worsen performance, latency, bandwidth compared to a system with DRAM and no caches?

Last time in Lecture 5

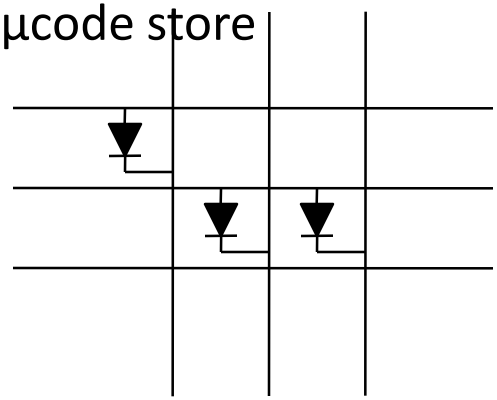
- Control hazards (branches, interrupts) are most difficult to handle as they change which instruction should be executed next
- Branch delay slots make control hazard visible to software, but not portable to more advanced µarchs
- Speculation commonly used to reduce effect of control hazards (predict 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

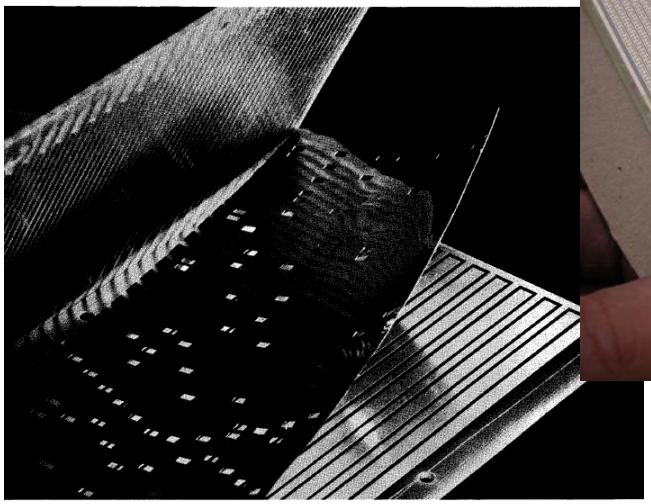
Early Read-Only Memory Technologies

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

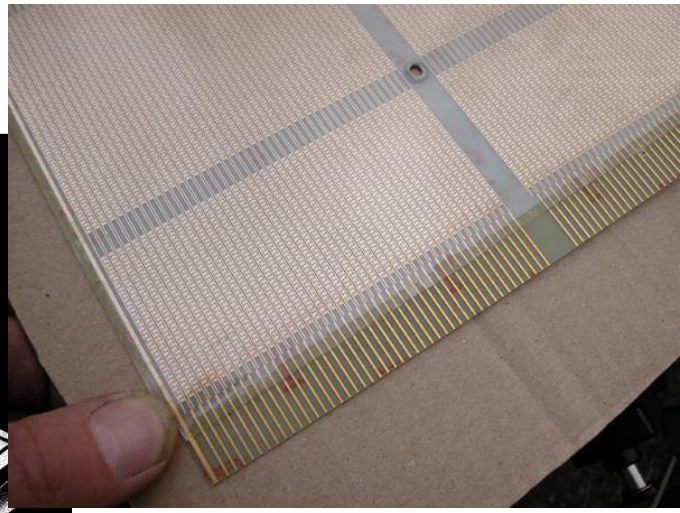
Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 µcode store





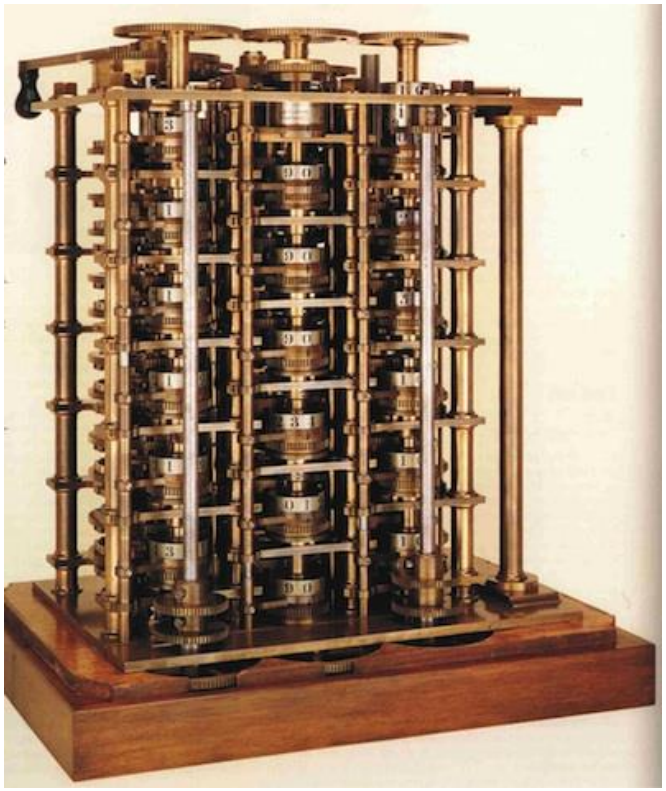
IBM Card Capacitor ROS

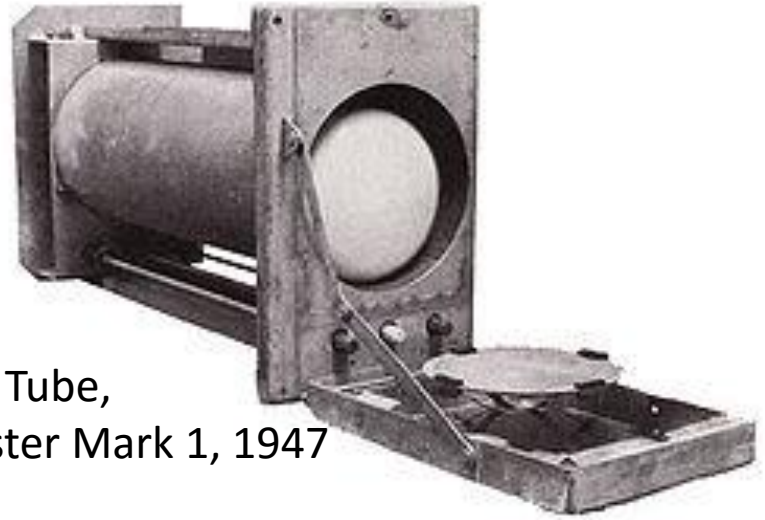


IBM Balanced Capacitor ROS

Early Read/Write Main Memory Technologies

Babbage, 1800s: Digits stored on mechanical wheels





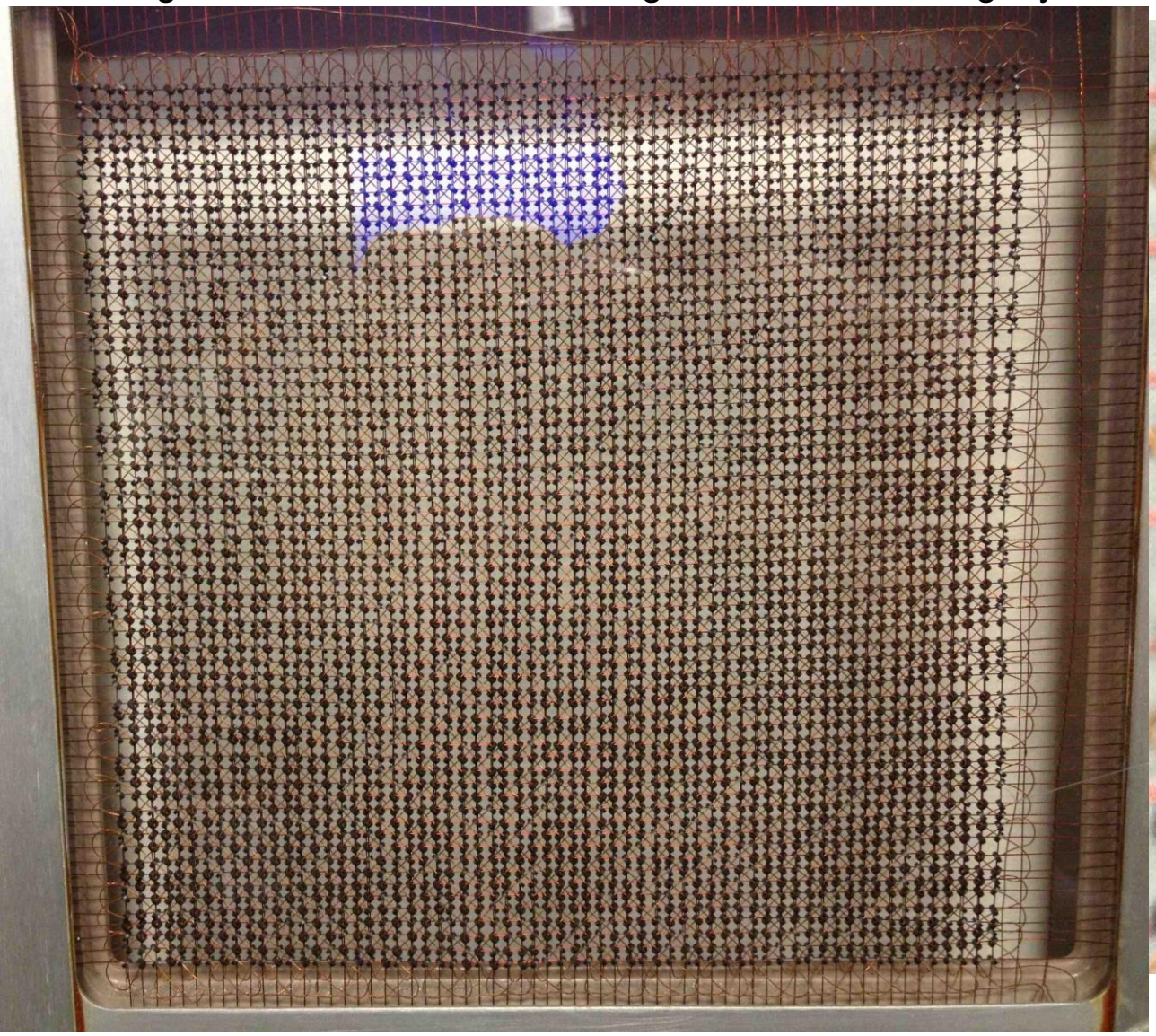
Williams Tube, Manchester Mark 1, 1947

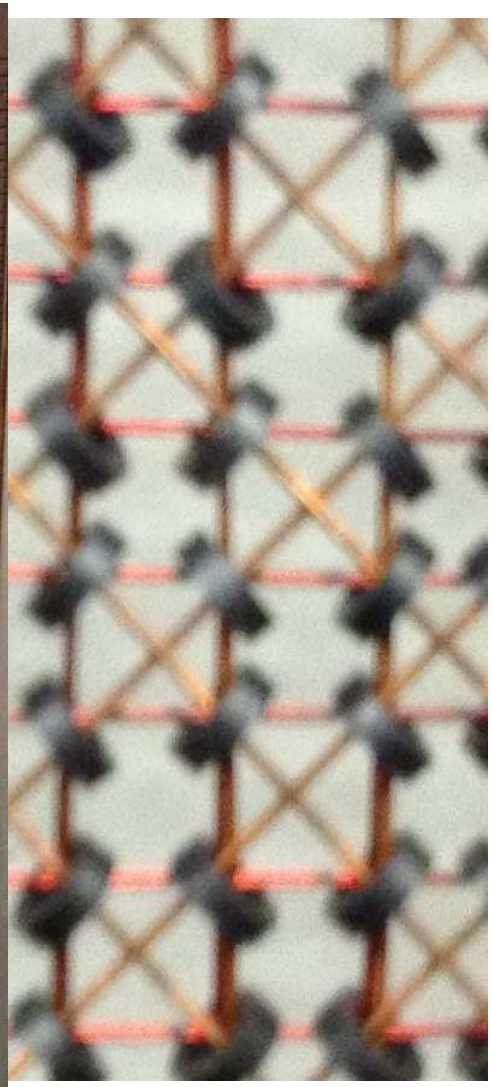
Mercury Delay Line, Univac 1, 1951

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

MIT Whirlwind Core Memory

Magnetic: Each “donut” was magnetized or not to signify zero or 1

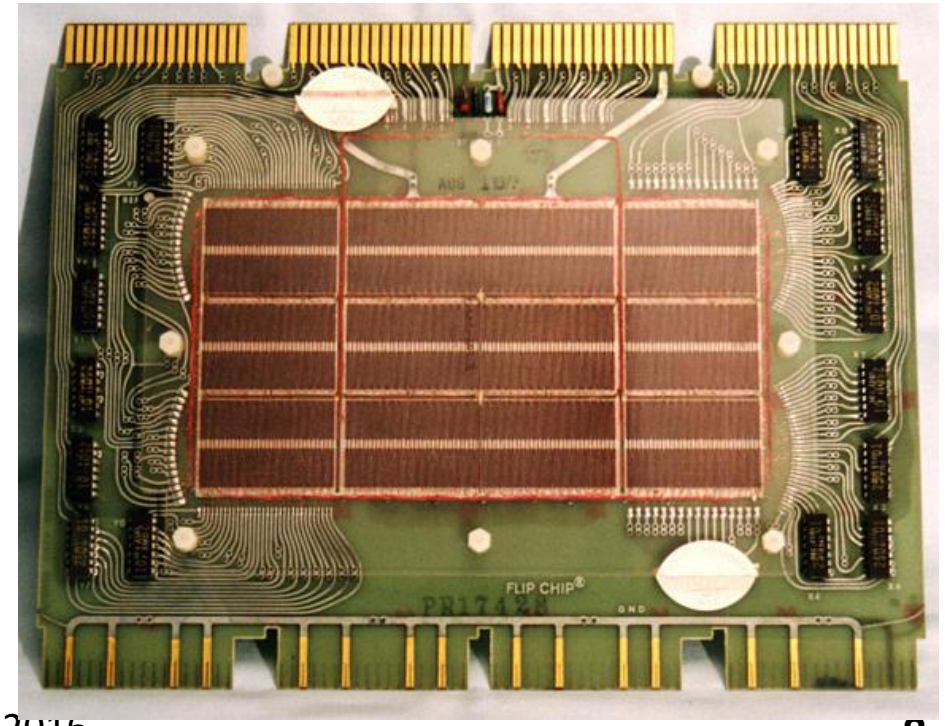




Core Memory

- 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)

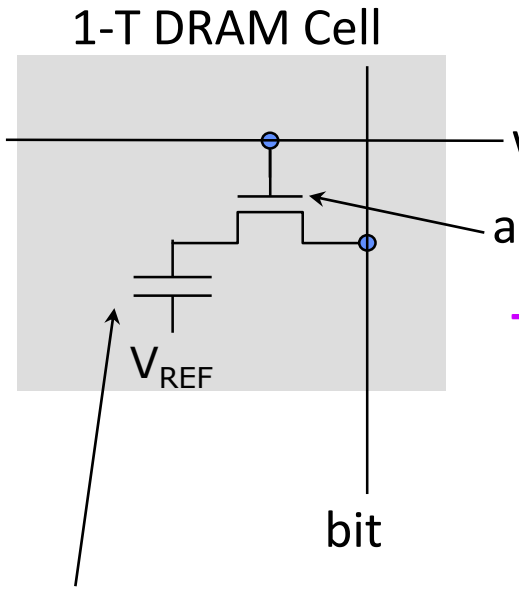
Semiconductor Memory

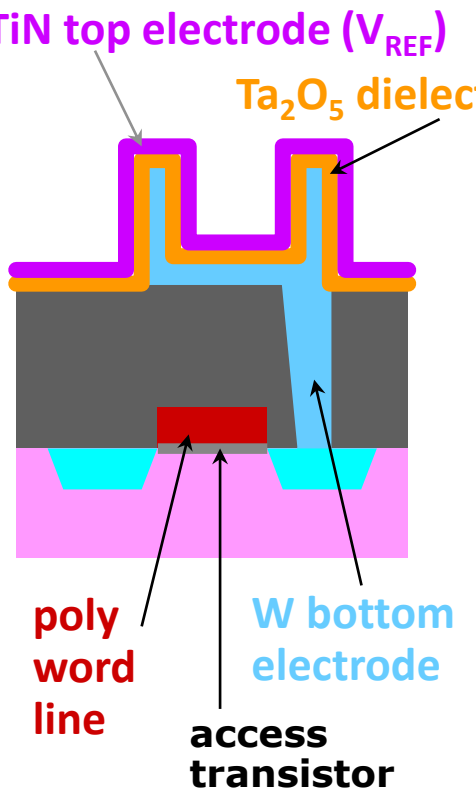
- Semiconductor memory began to be competitive in early 1970s
  - Intel formed to exploit market for semiconductor memory
  - Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).

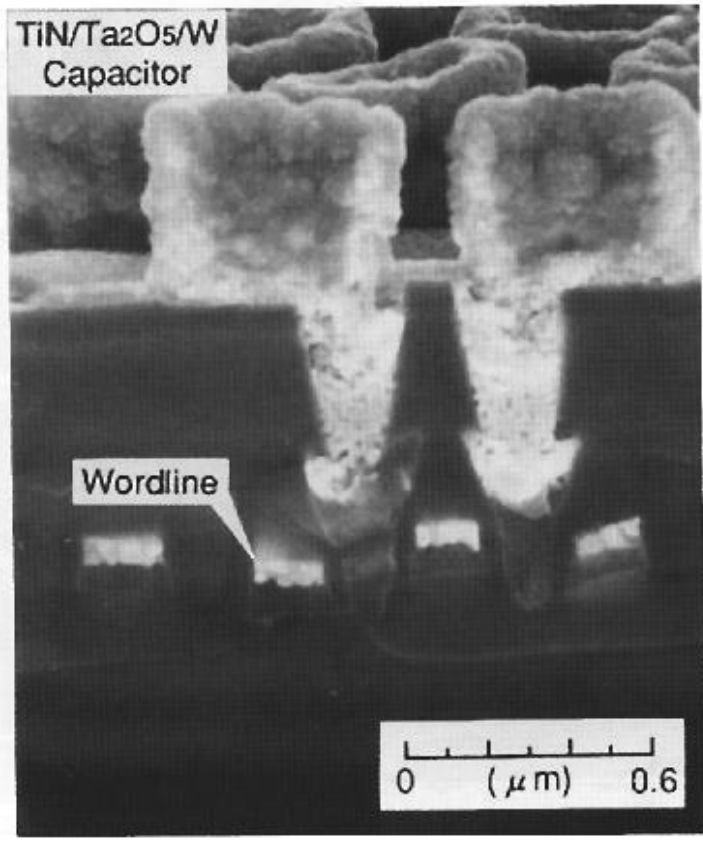
- First commercial Dynamic RAM (DRAM) was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

Semiconductor memory quickly replaced core in '70s

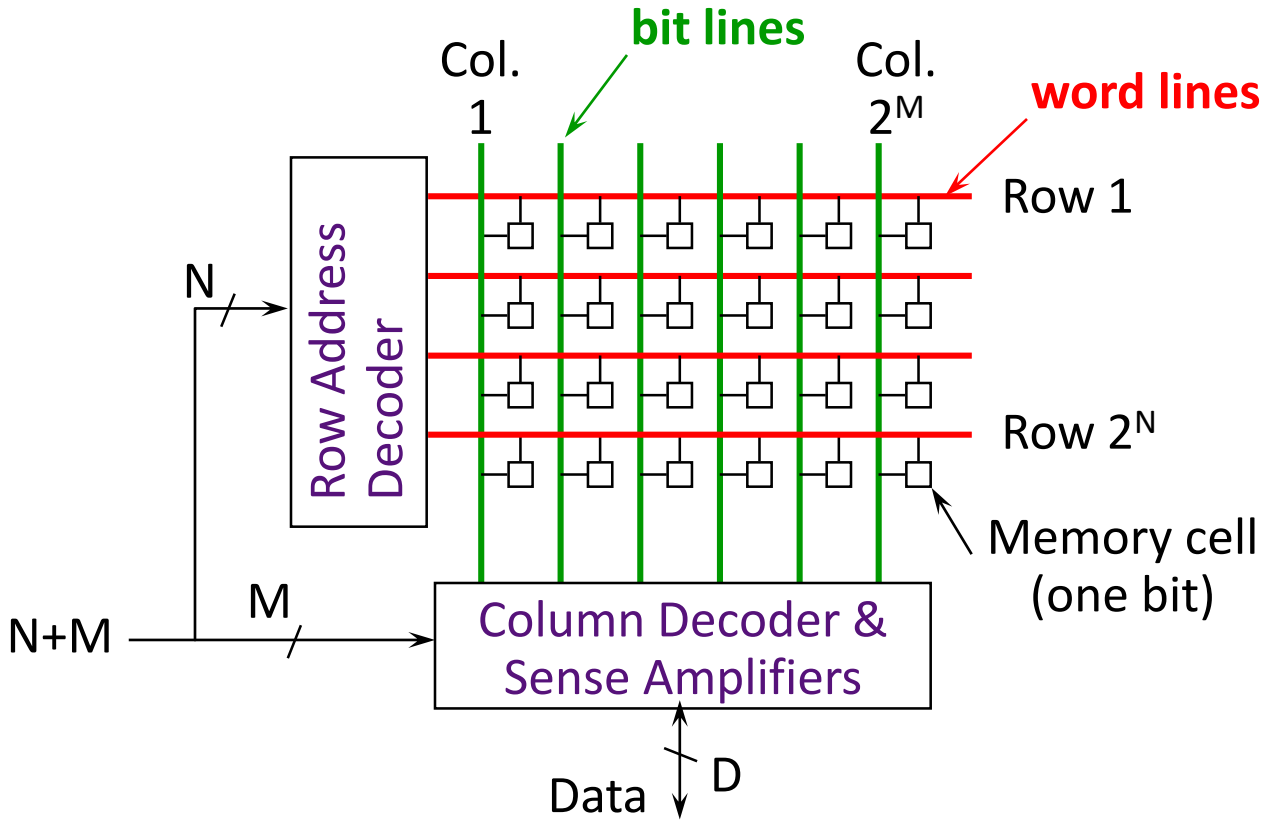
One-Transistor Dynamic RAM [Dennard, IBM]





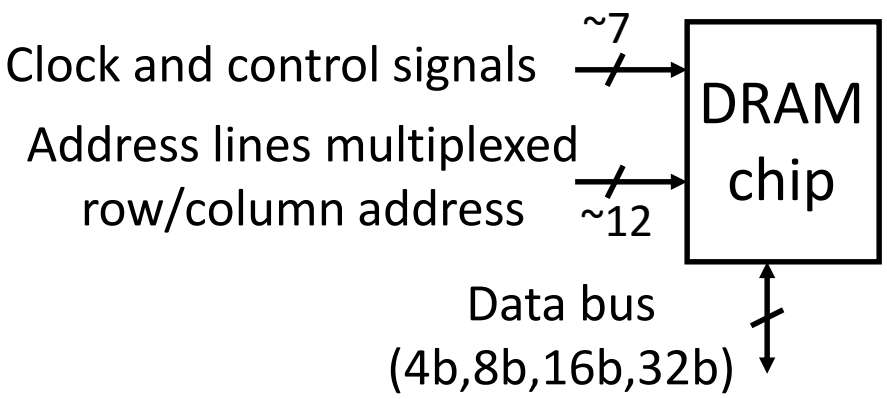


DRAM Architecture

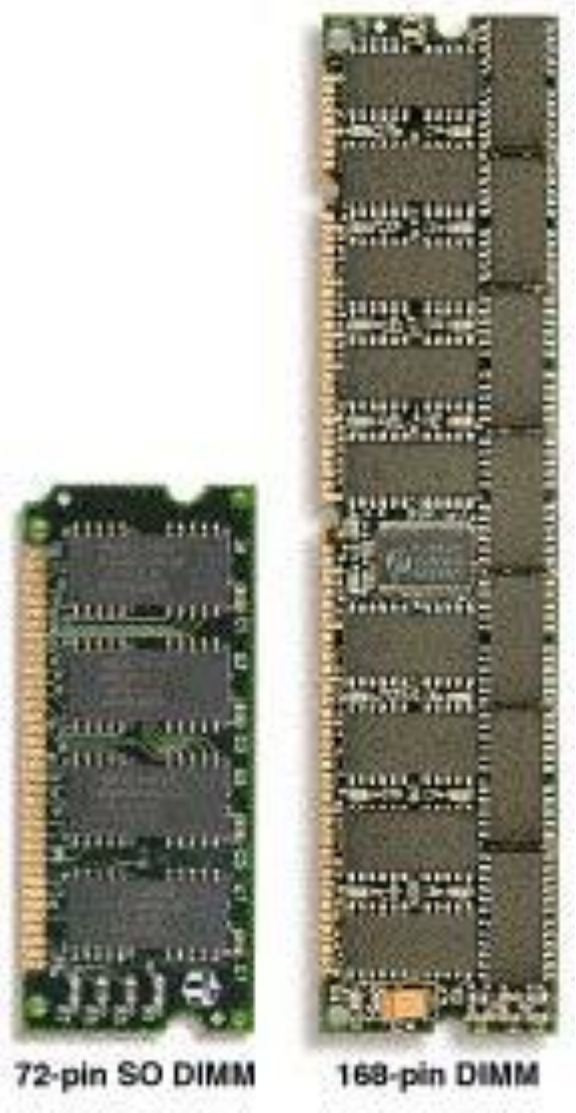


- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4-8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

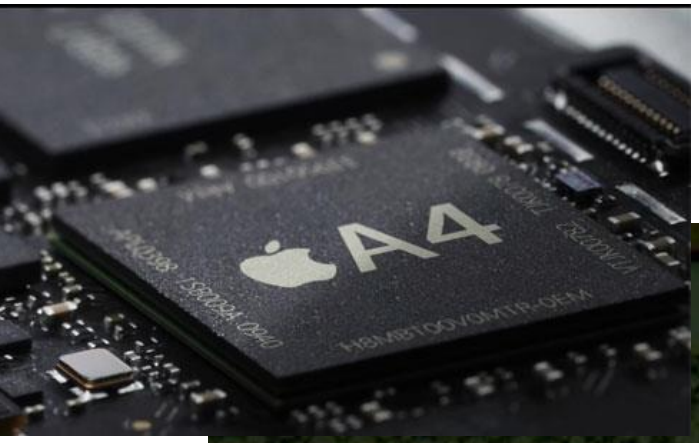
DRAM Packaging
(Laptops/Desktops/Servers)



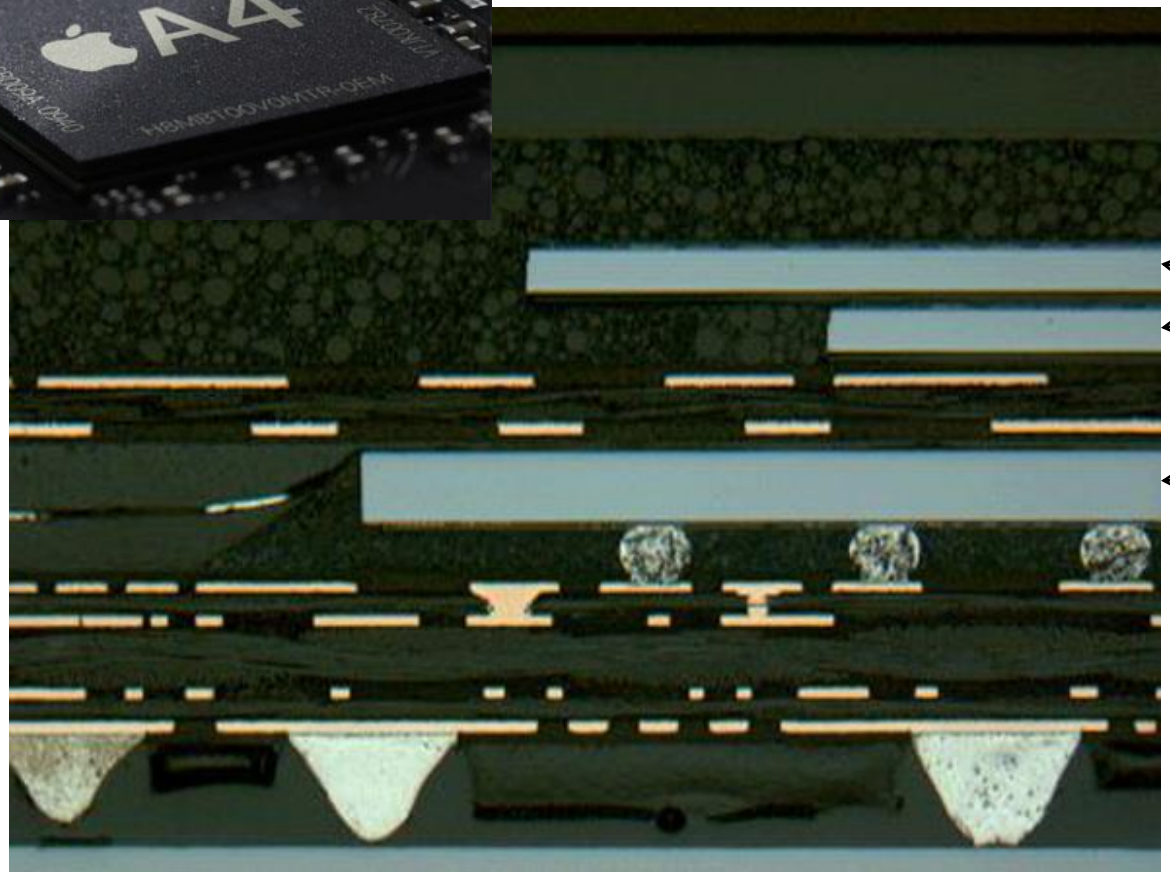
- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips)
- Data pins work together to return wide word (e.g., 64-bit data bus using 16x4-bit parts)



DRAM Packaging, Mobile Devices



[Apple A4 package on circuit board]

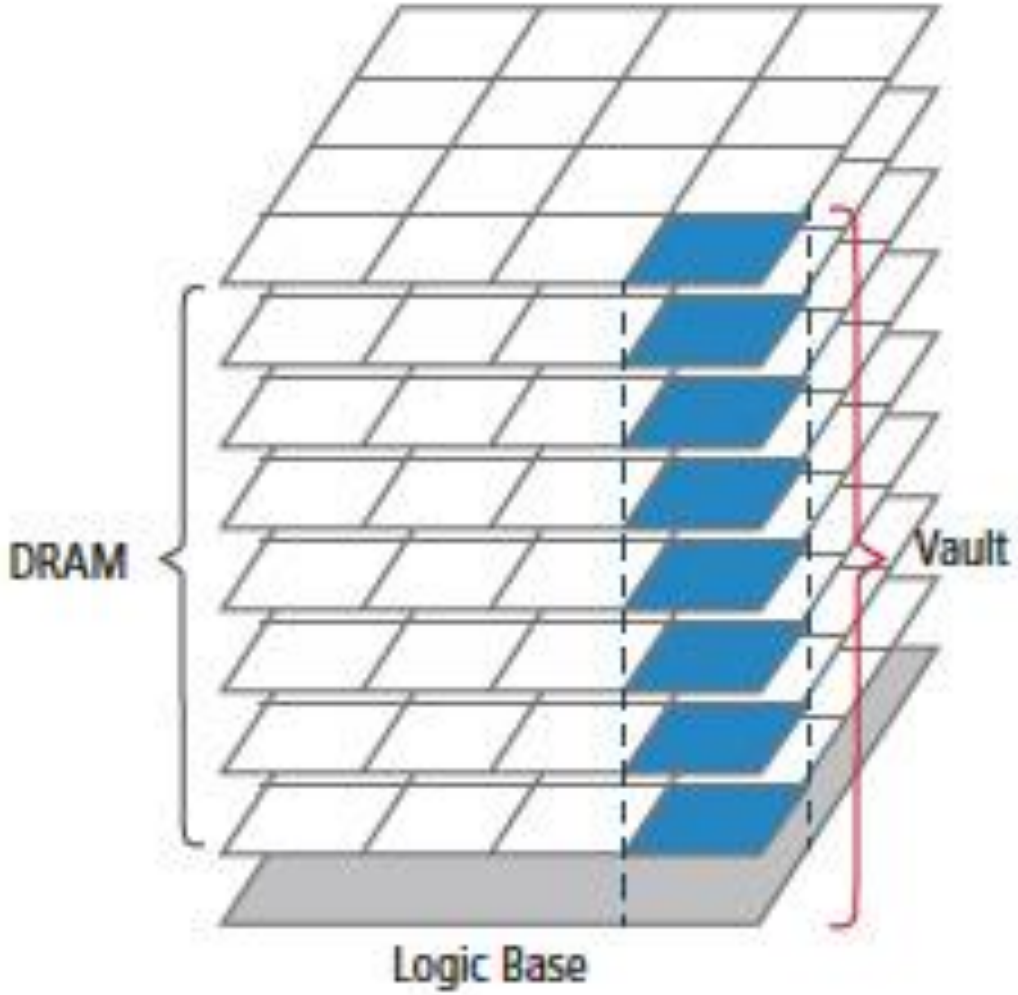


Two stacked DRAM die

Processor plus logic die

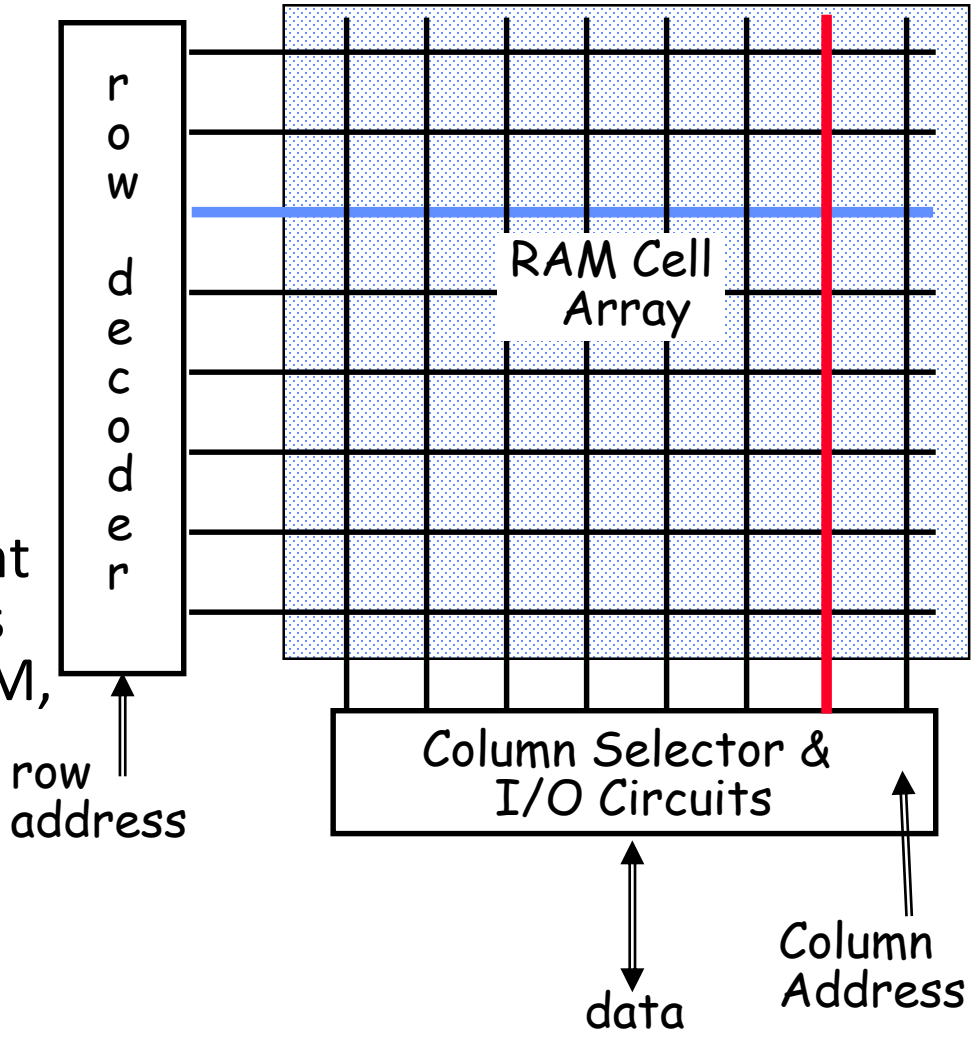
[Apple A4 package cross-section, iFixit 2010]

3D Stacked Memory



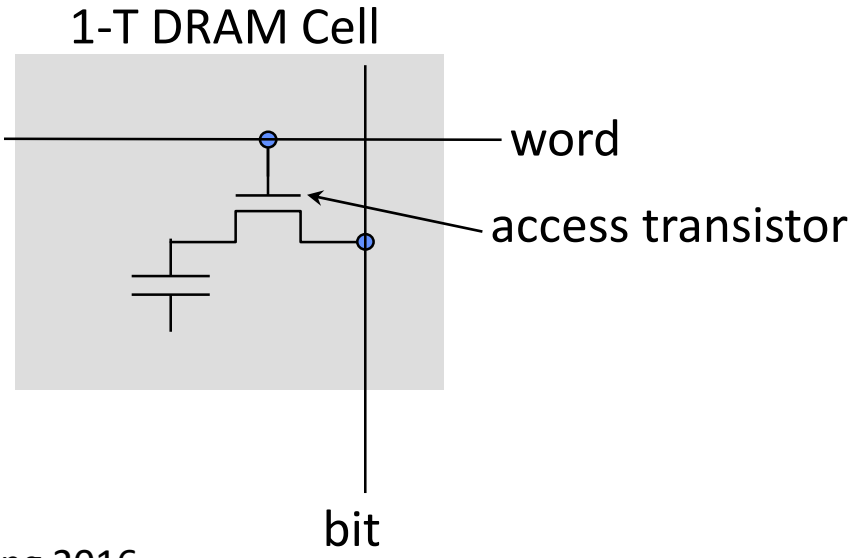
DRAM Operation

- Three steps in read/write access to a given bank
- Row access (RAS)
- Column access (CAS)
- 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

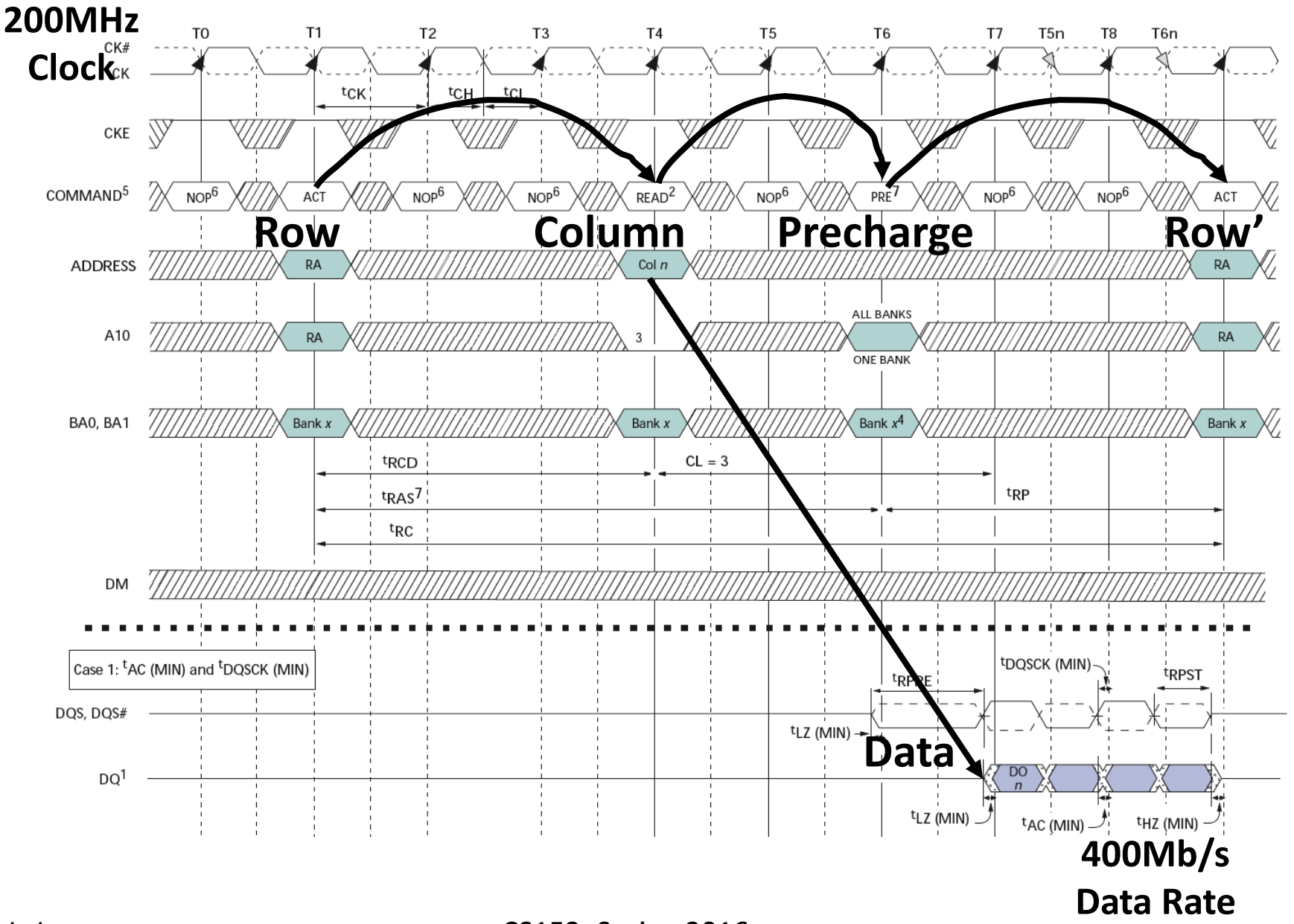


DRAM Operation (Verbose)

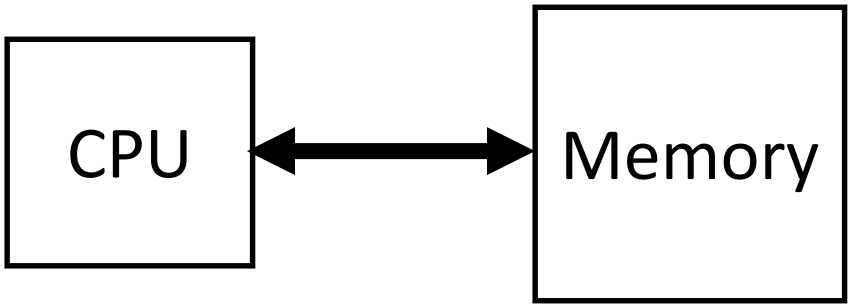
- Row access (RAS)
  - decode row address, enable addressed row (often multiple Kb in row)
  - bitlines share charge with storage cell
  - small change in voltage detected by sense amplifiers which latch whole row of bits
  - sense amplifiers drive bitlines full rail to recharge storage cells
- Column 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
  - reads are destructive!



Double-Data Rate (DDR2) DRAM



CPU-Memory Bottleneck



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

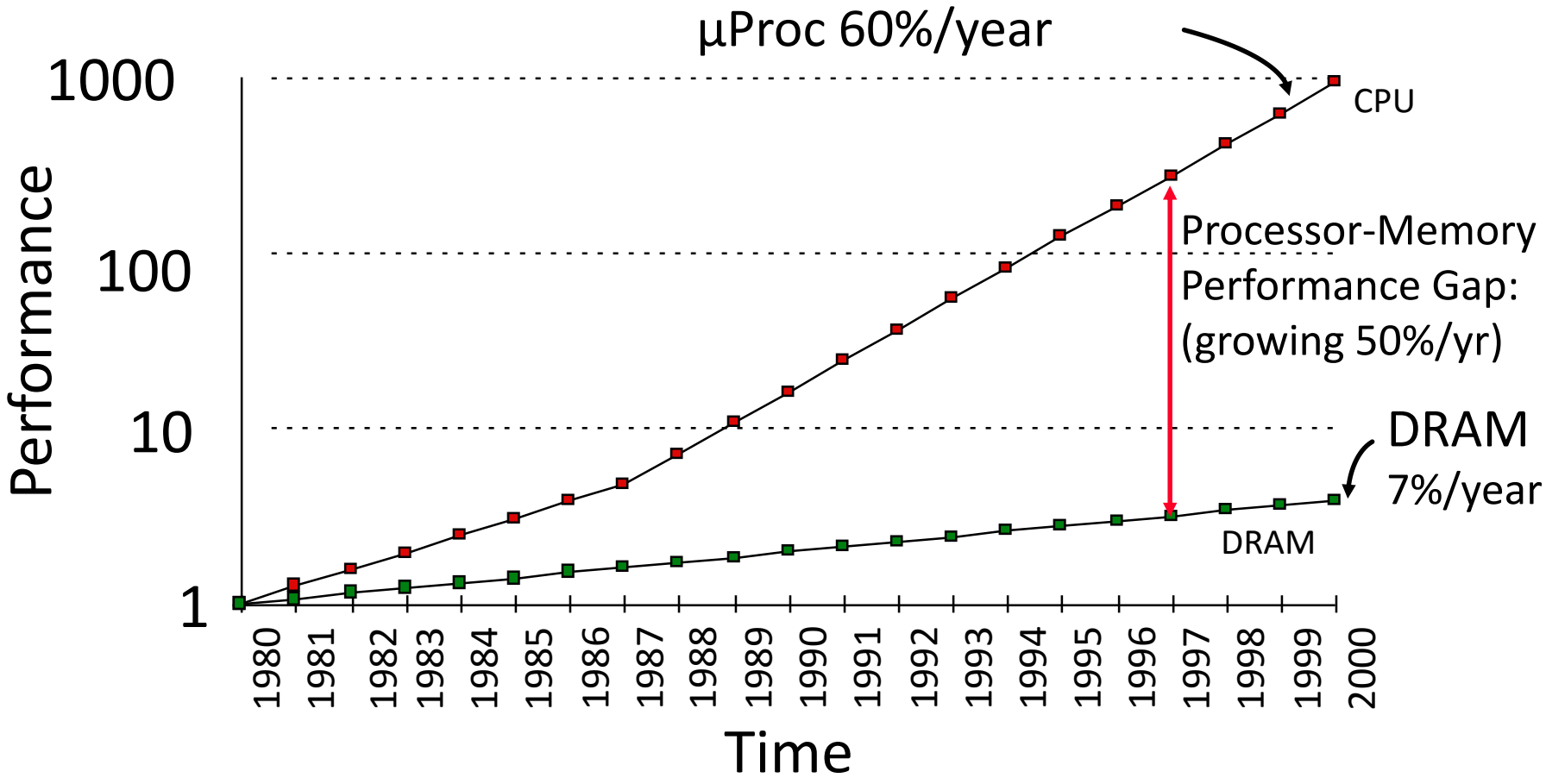
- Latency (time for a single access)
  - Memory access time >> Processor cycle time
- Bandwidth (number of accesses per unit time)

  if fraction m of instructions access memory

  => 1+m memory references / instruction

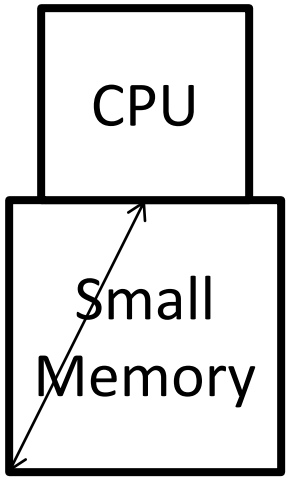
  => CPI = 1 requires 1+m memory refs / cycle (assuming RISC-V ISA)

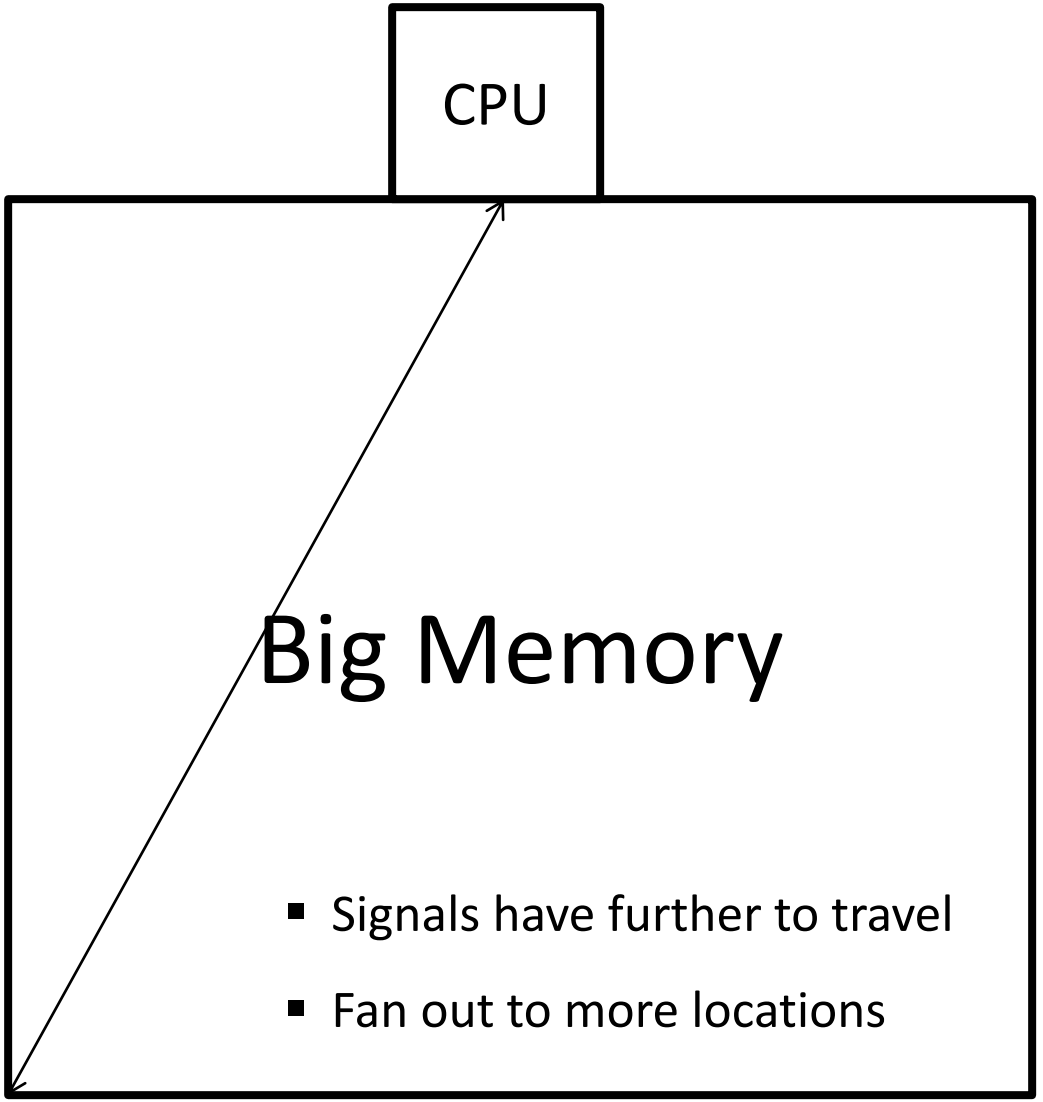
Processor-DRAM Gap (latency)



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

Physical Size Affects Latency





Motivates 3D stacking

Memory Bandwidth Growth

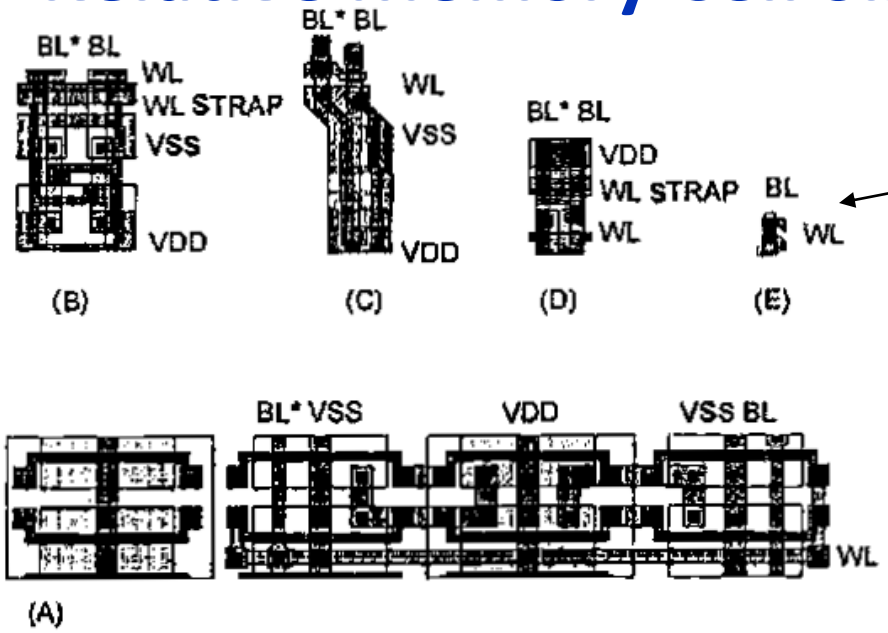
How to take advantage of all this bandwidth?

- Simple in-order cores
- Complex out of order cores
- ?

Relative Memory Cell Sizes

On-Chip SRAM in logic chip

DRAM on memory chip



1 Memory cell in 0.5µm processes

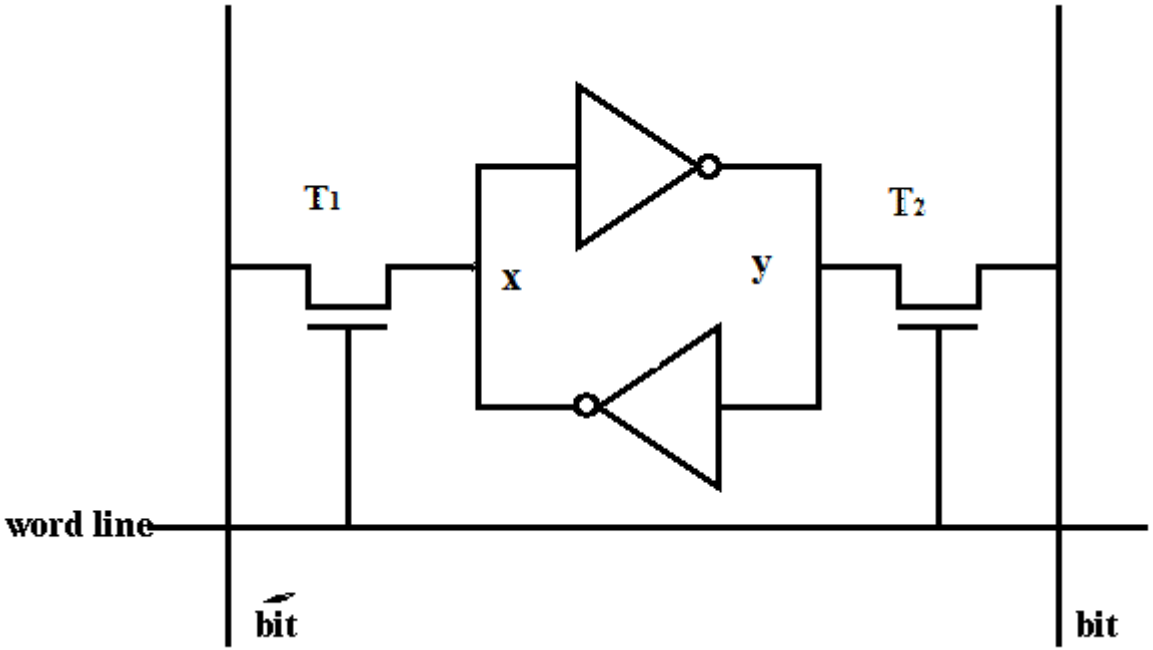
a) Gate Array SRAM
b) Embedded SRAM
c) Standard SRAM (6T cell with local interconnect)
d) ASIC DRAM
e) Standard DRAM (stacked cell)

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

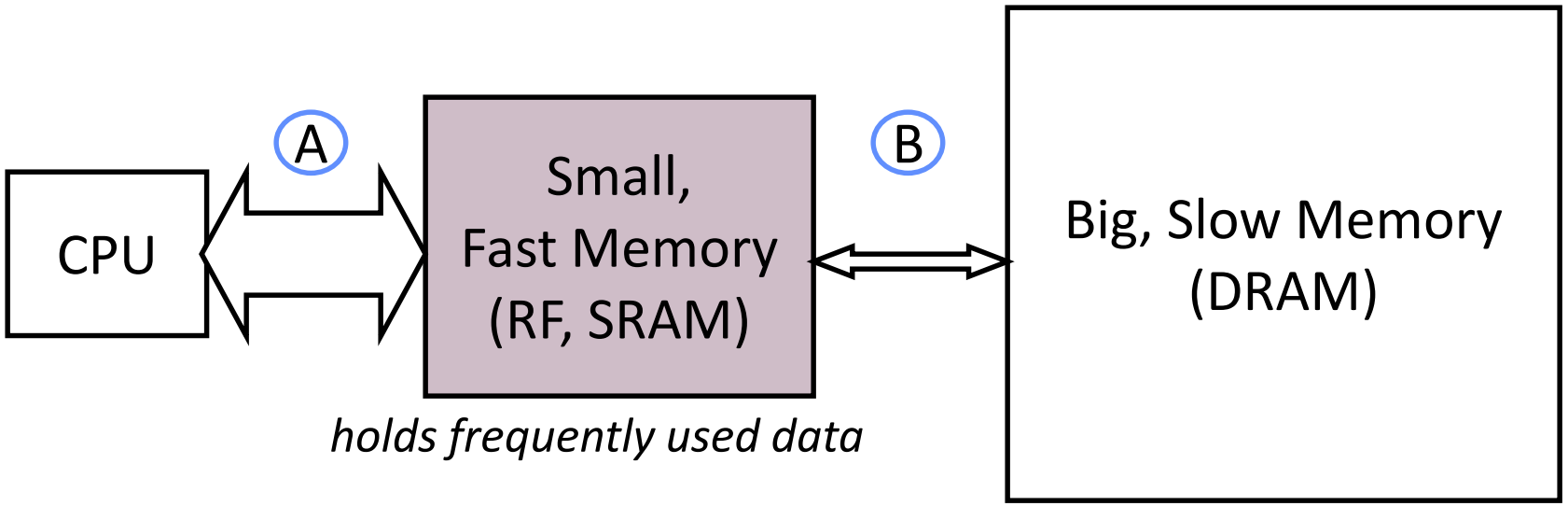
Memory	Process	Cell size (μm^2)	Cell efficiency	Bits in $100mm^2 (10^6)$	Gate size (μm^2)	Gate utilization	Gates in $100mm^2 (10^6)$
Gate array SRAM	3-metal ASIC	370	80%	216	185	70%	378
Embedded SRAM	3-metal ASIC	67	70%	1045	185	70%	378
Standard SRAM	2-metal 6T local int.	43	65%	1512	245	40%	163
Embedded 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.

SRAM Cell



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)*

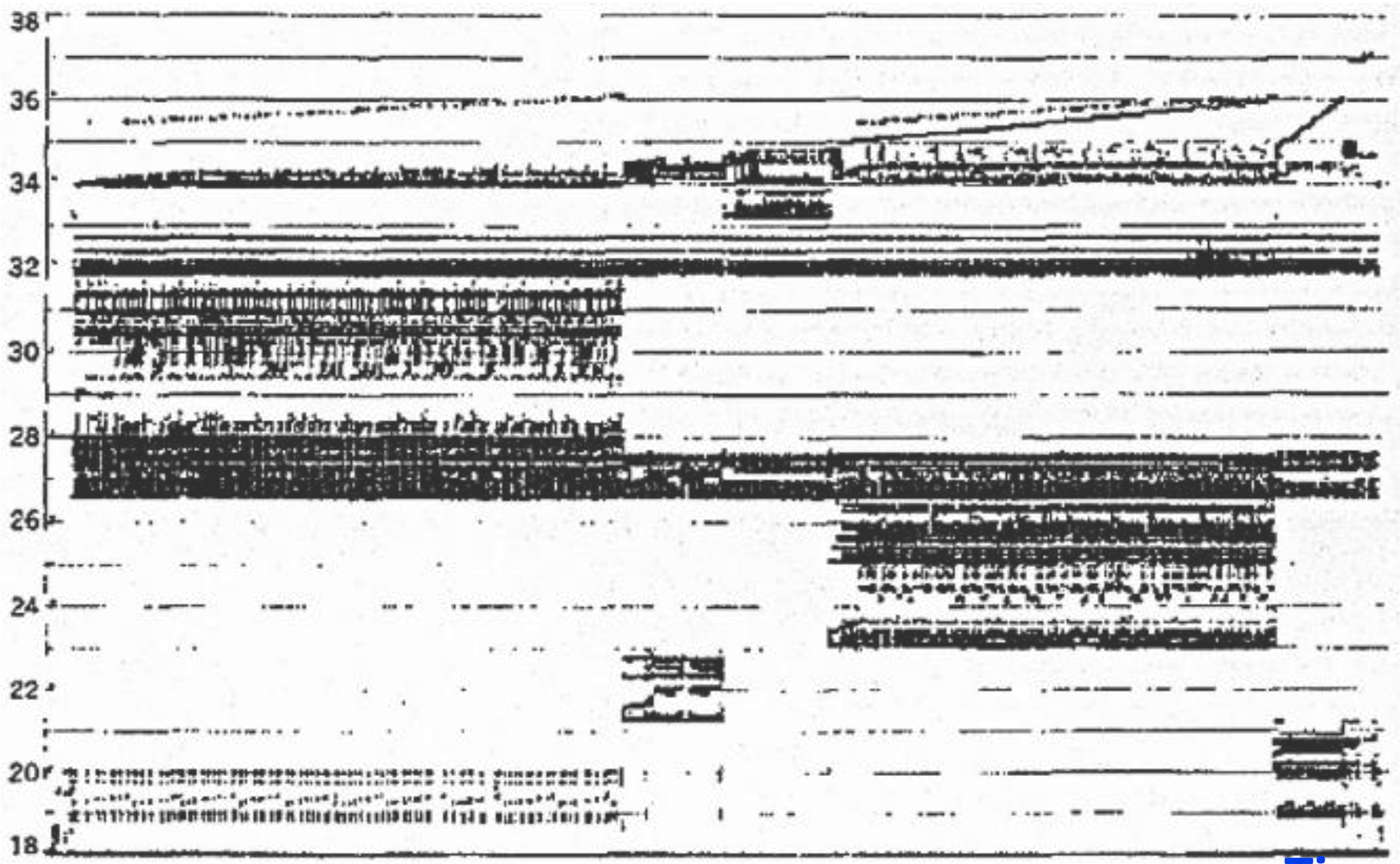
if data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

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*

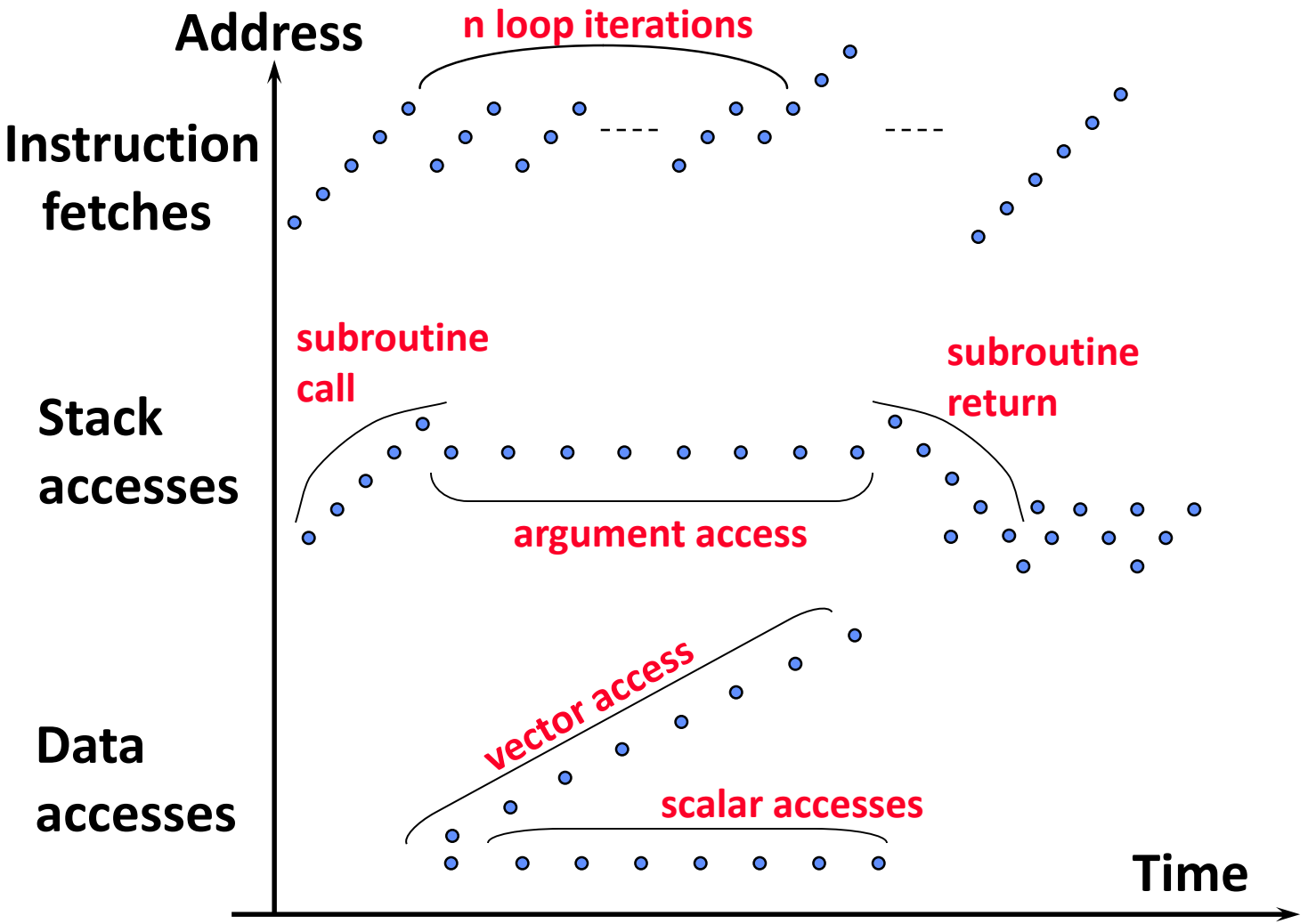
- Larger/slower storage, e.g., main memory
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
    - *but software may provide "hints", e.g., don't cache or prefetch*

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

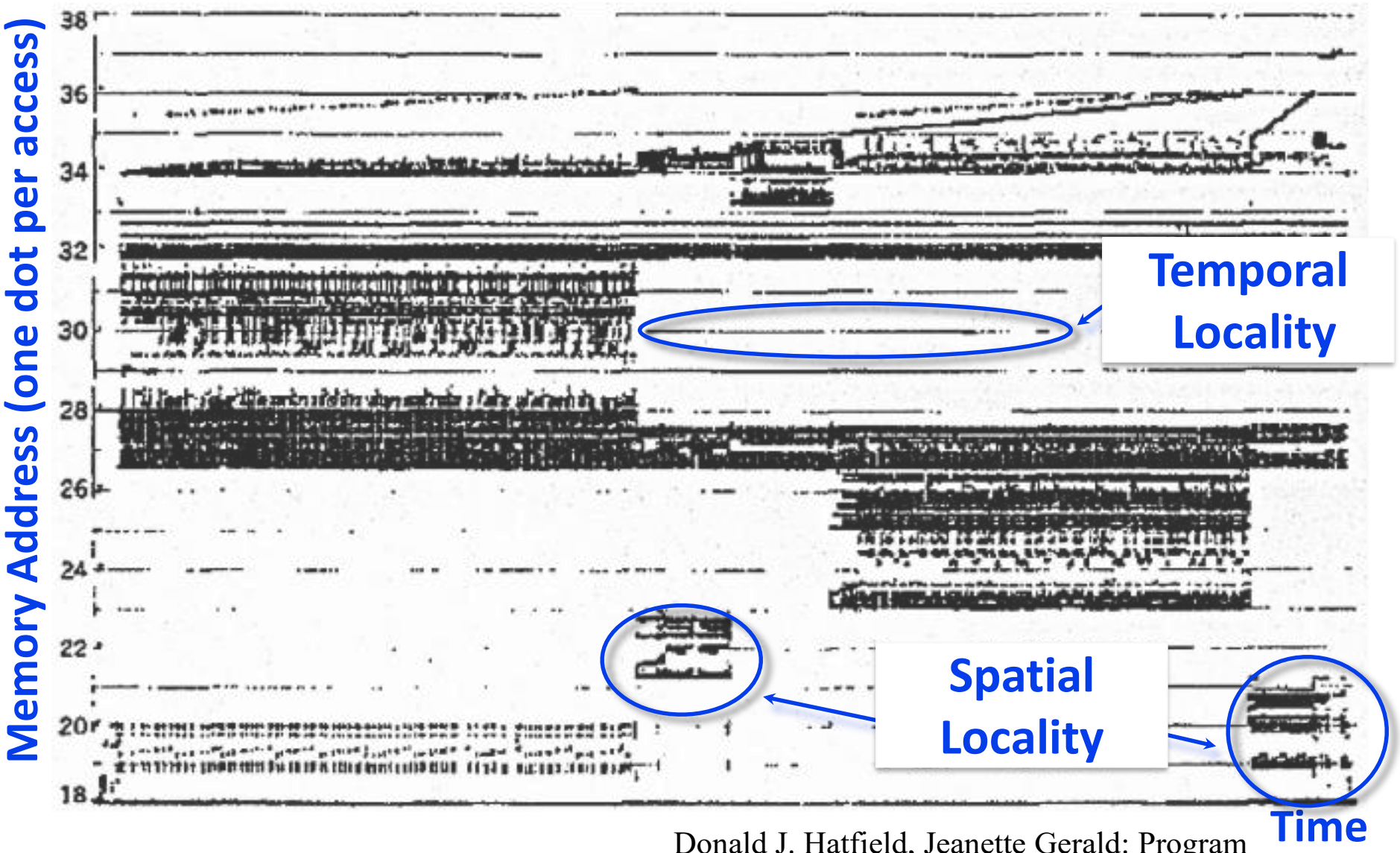


Two predictable properties of memory references:

- **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

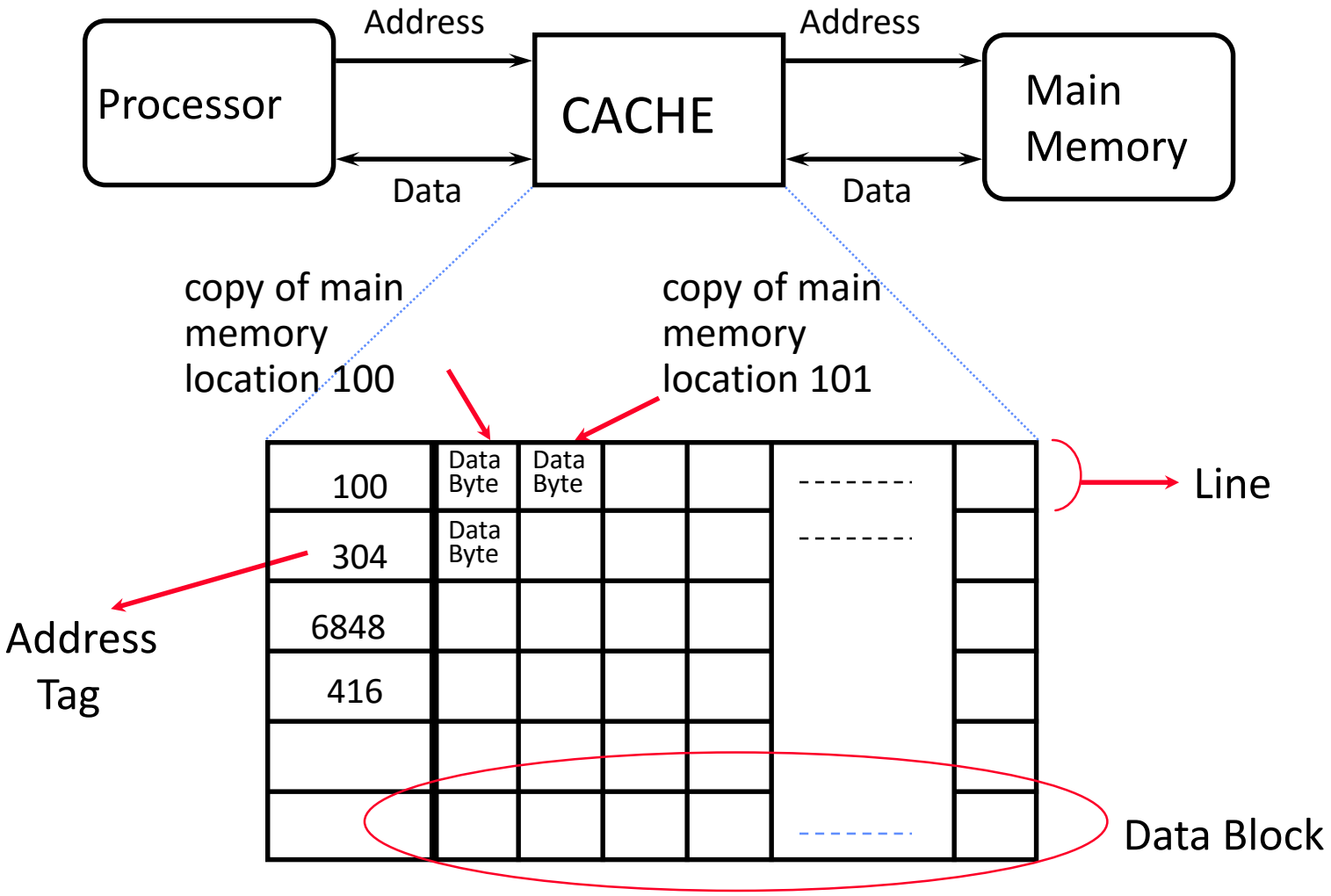


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

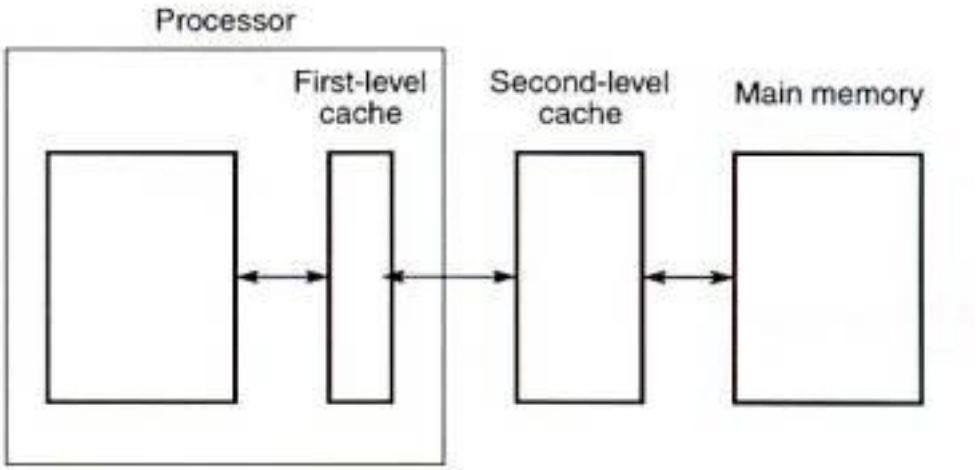
Caches exploit both types of predictability:

- 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

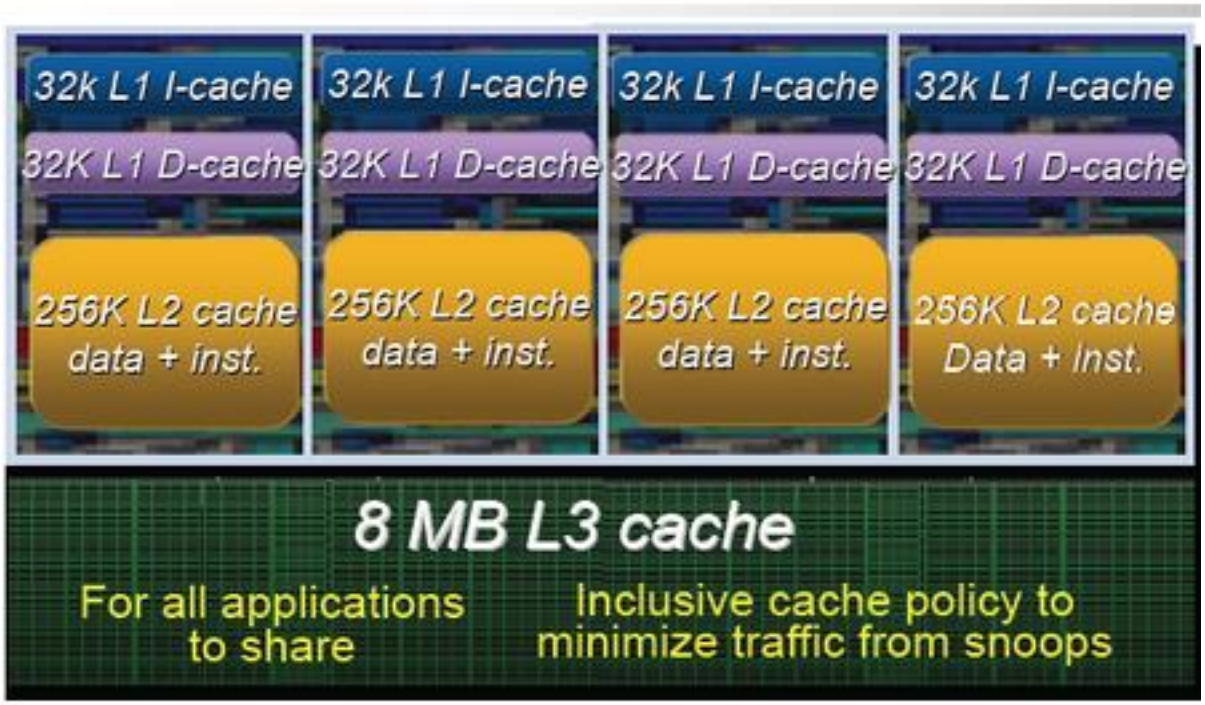


Multiple Cache Levels

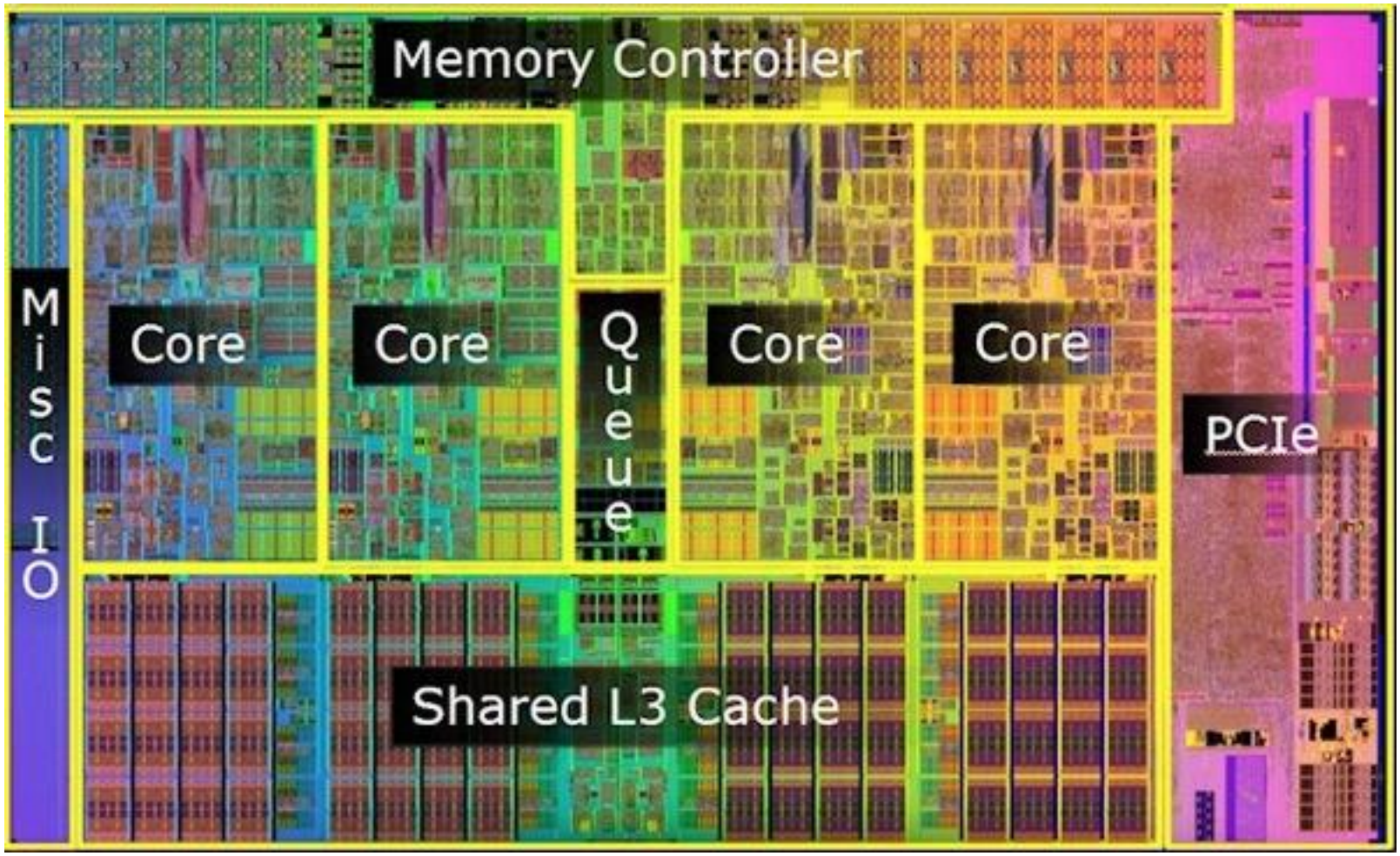


Intel i7 (Nahelem)

- Private L1 and L2
  - L2 is 256KB each. 10 cycle latency
- 8MB shared L3. ~40 cycles latency

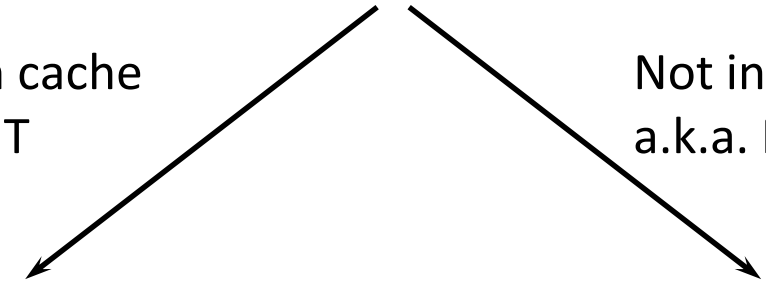


Area


Memory Controller
Misc IO
Core
Core
Queue
Core
Core
PCIe
Shared L3 Cache

Cache Algorithm (Read)

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



Found in cache
a.k.a. HIT

Not in cache
a.k.a. MISS

Return copy of data from cache

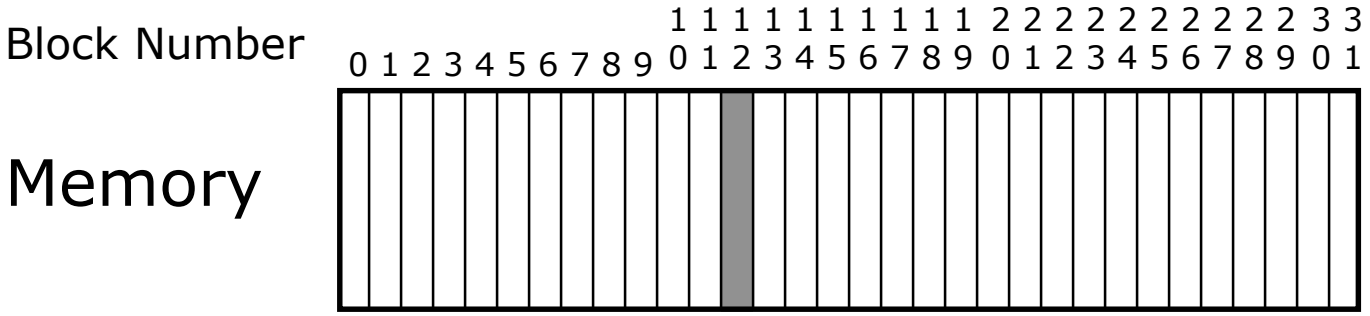
Read block of data from Main Memory

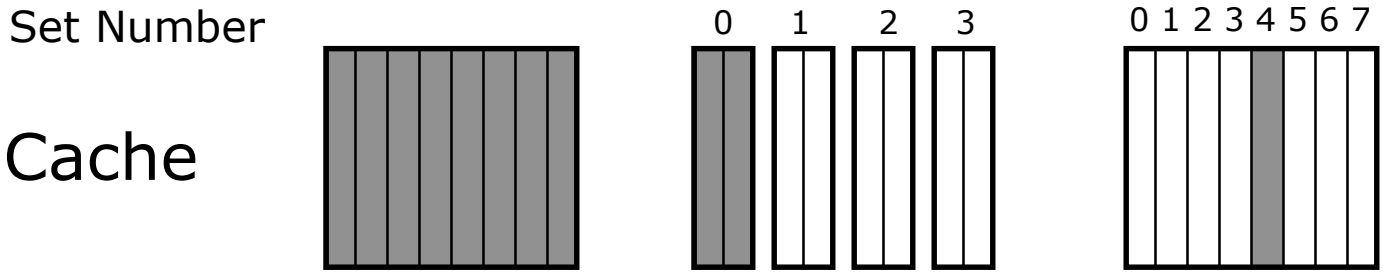
Wait …

Return data to processor and update cache

Q: Which line do we replace?

Placement Policy





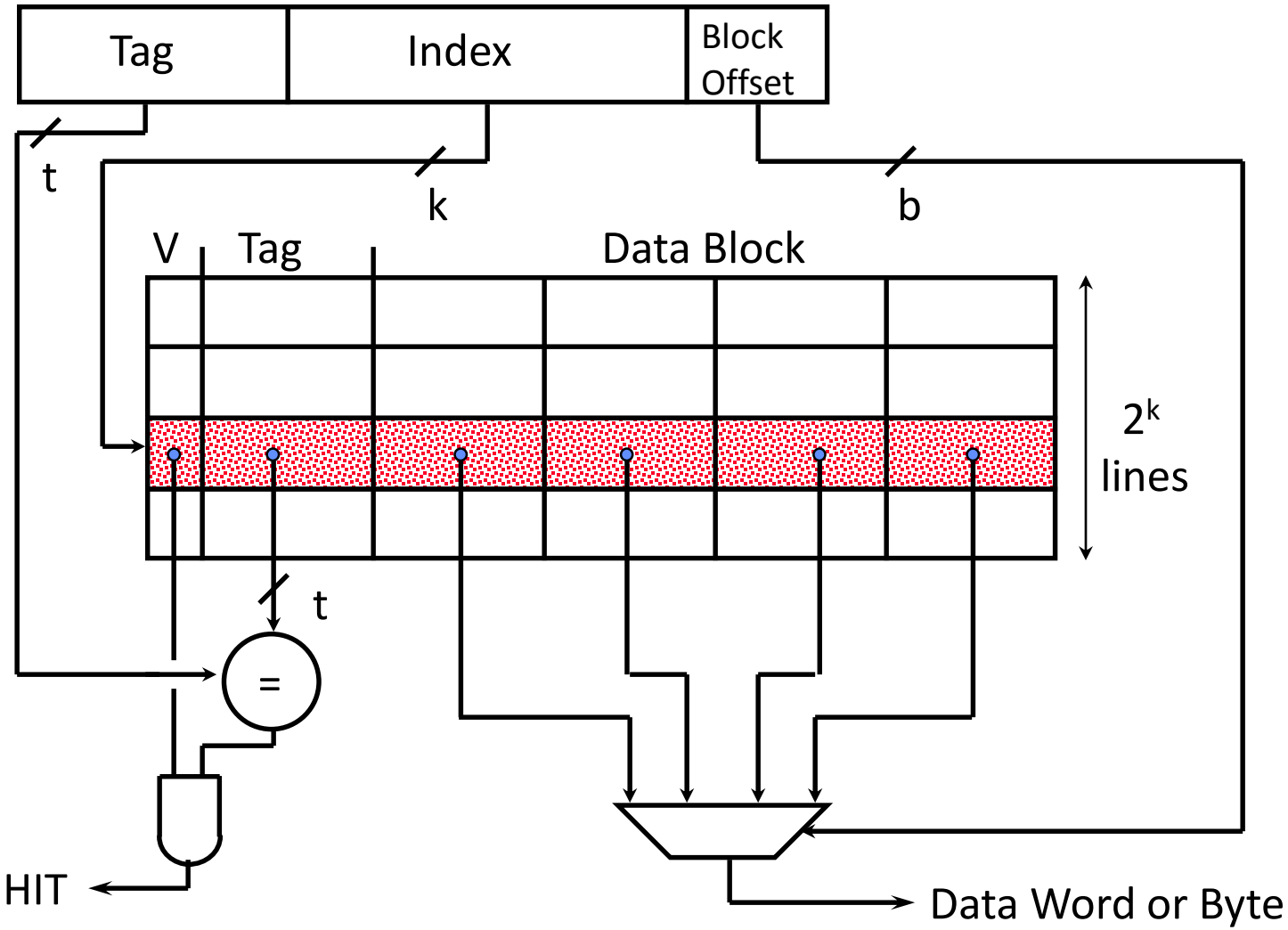
Fully Associative
anywhere

(2-way) Set Associative
anywhere in set 0
(12 mod 4)

Direct Mapped
only into block 4
(12 mod 8)

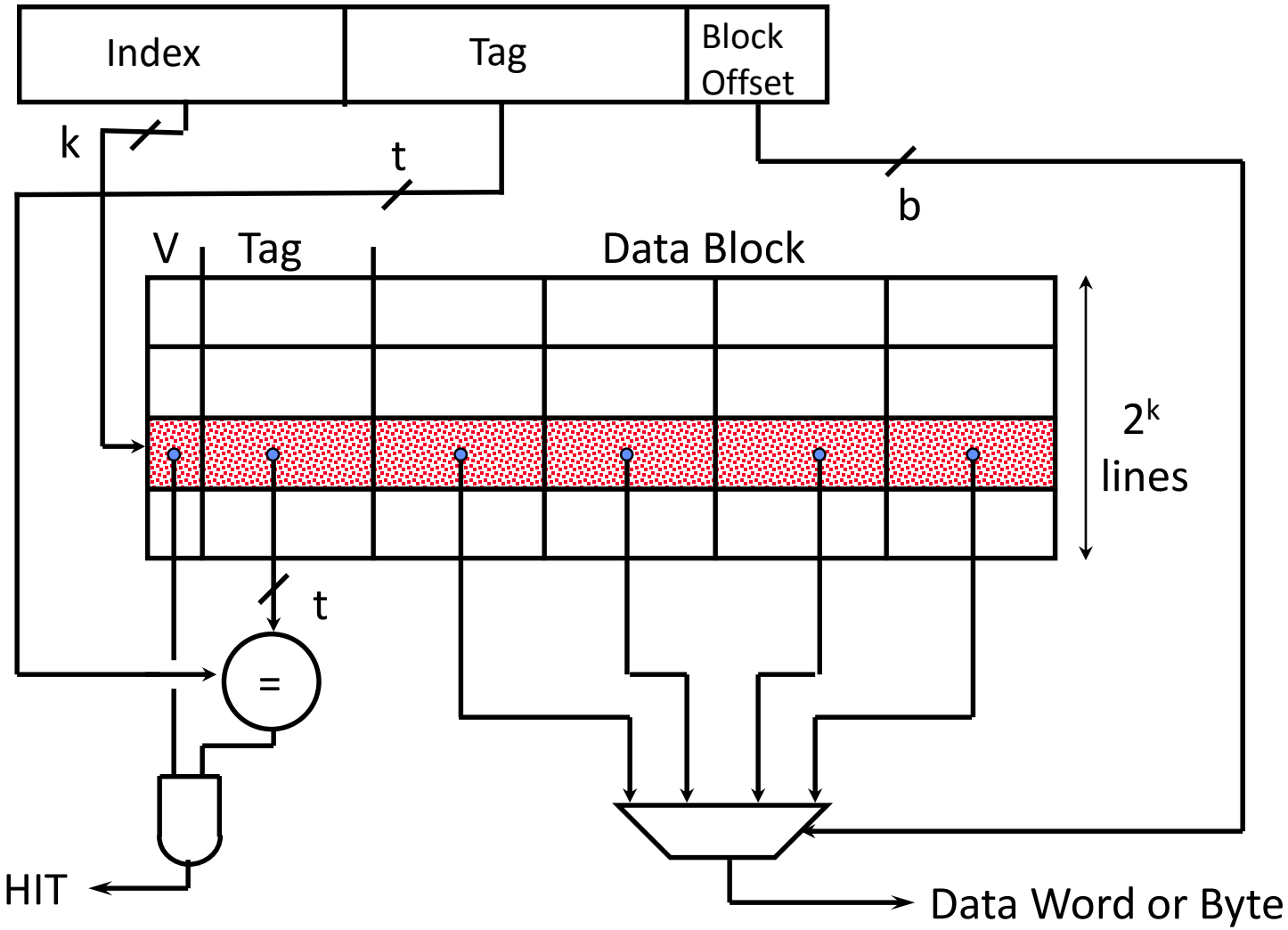
block 12
can be placed

Direct-Mapped Cache

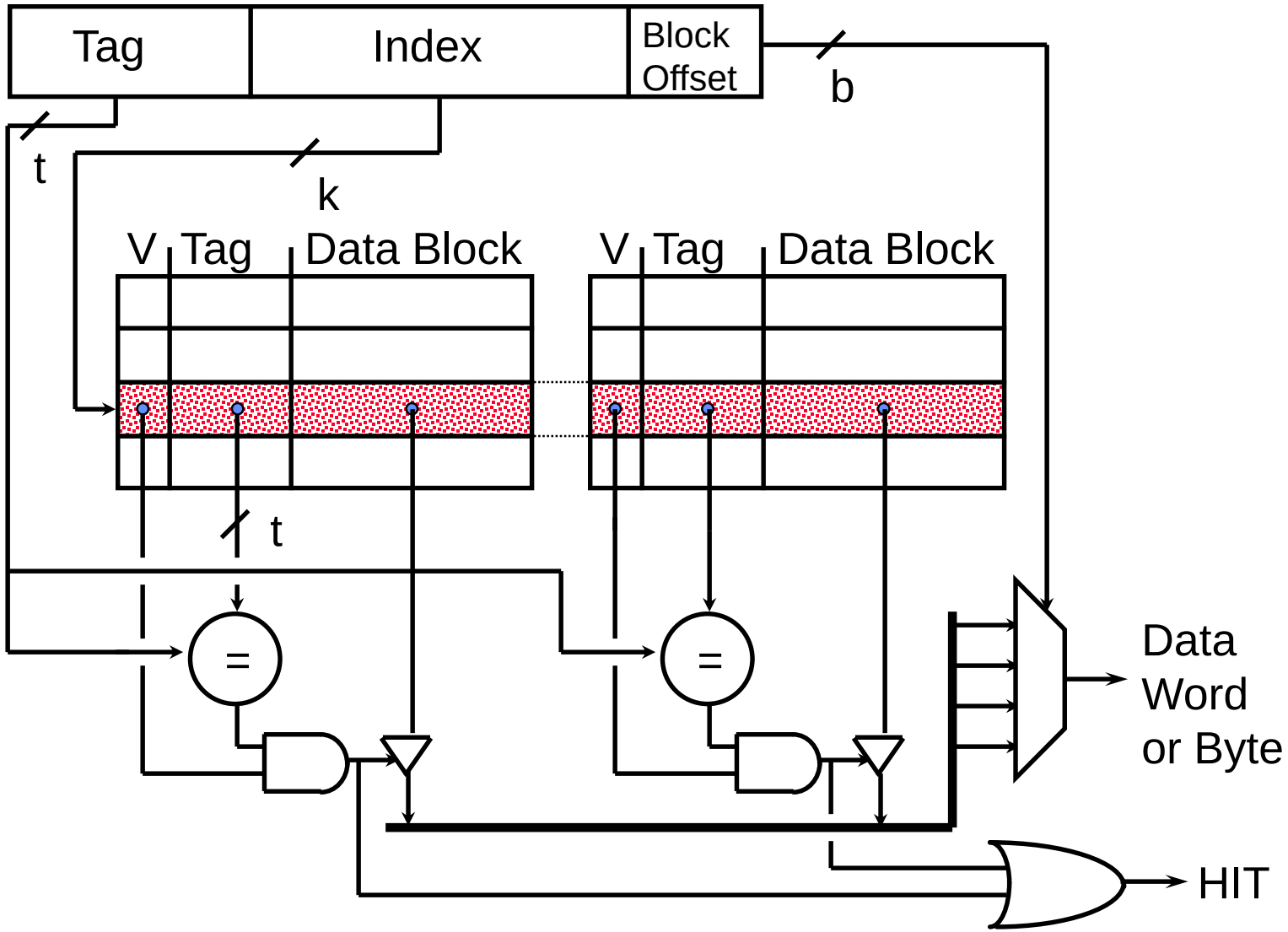


Direct Map Address Selection

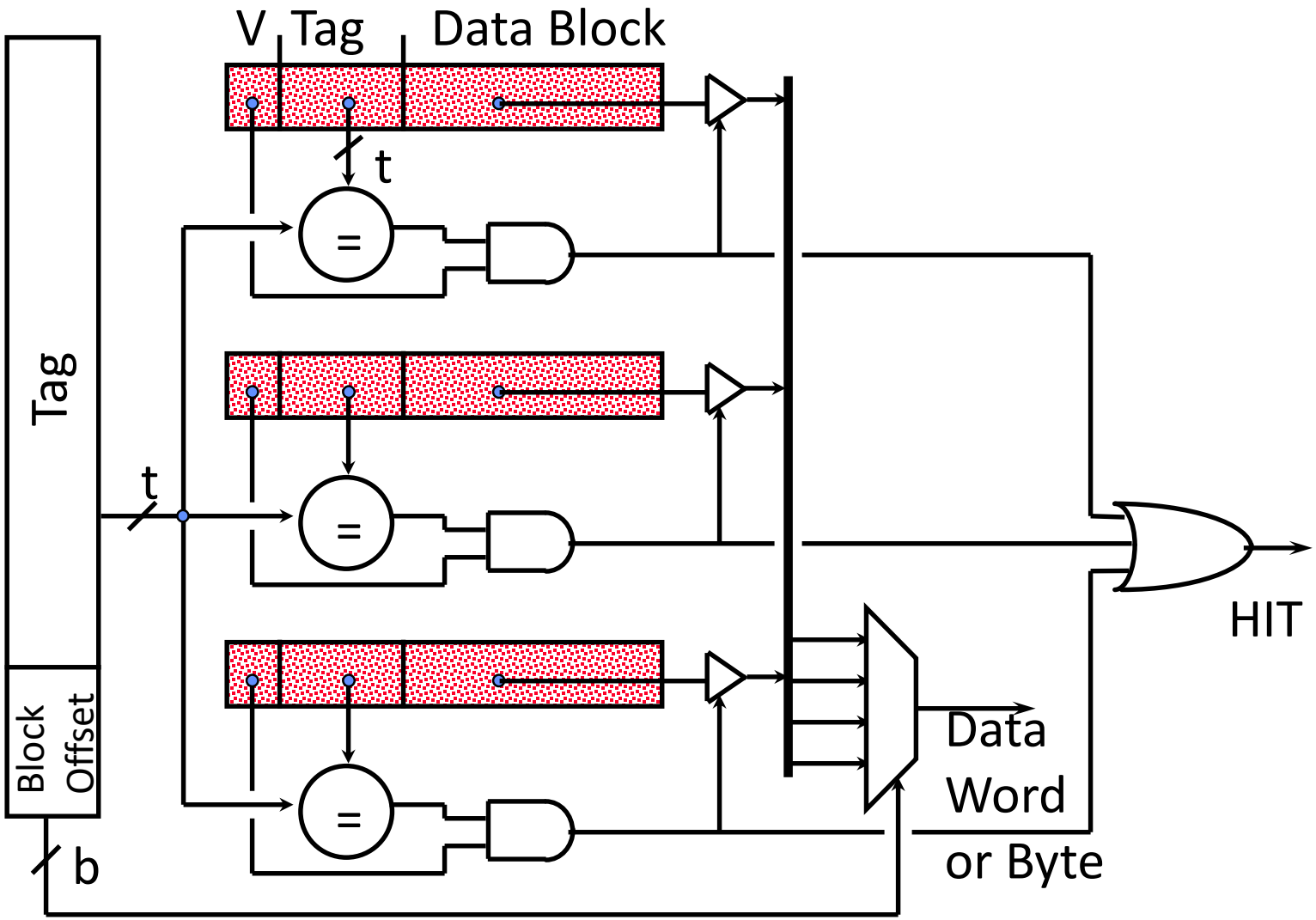
higher-order vs. lower-order address bits



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?

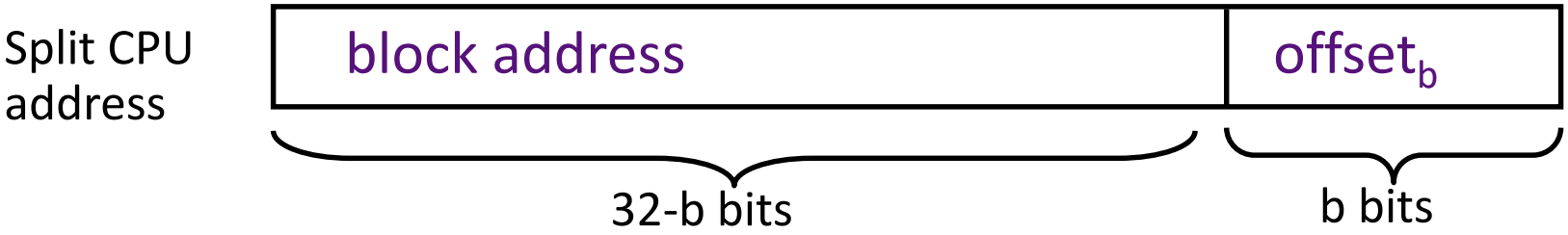
Replacement only happens on misses

Block Size and Spatial Locality

E.g., define how many bytes a memory address references

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.

Question of the Day

- Can a cache worsen performance, latency, bandwidth compared to a system with DRAM and no caches?

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