ECE 252 / CPS 220 Advanced Computer Architecture I

Lecture 12 Memory

Benjamin Lee
Electrical and Computer Engineering
Duke University

www.duke.edu/~bcl15
www.duke.edu/~bcl15/class/class_ece252fall11.html

ECE252 Administrivia

20 October – Homework #3 Due

20 October – Project Proposals Due

One page proposal

1. What question are you asking?
2. How are you going to answer that question?
3. Talk to me if you are looking for project ideas.

25 October – Class Discussion

Roughly one reading per class. Do not wait until the day before!

1. Jouppi. “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers.”
2. Kim et al. “An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches.”
3. Fromm et al. “The energy efficiency of IRAM architectures”
4. Lee et al. “Phase change memory architecture and the quest for scalability”

History of Memory

Core Memory

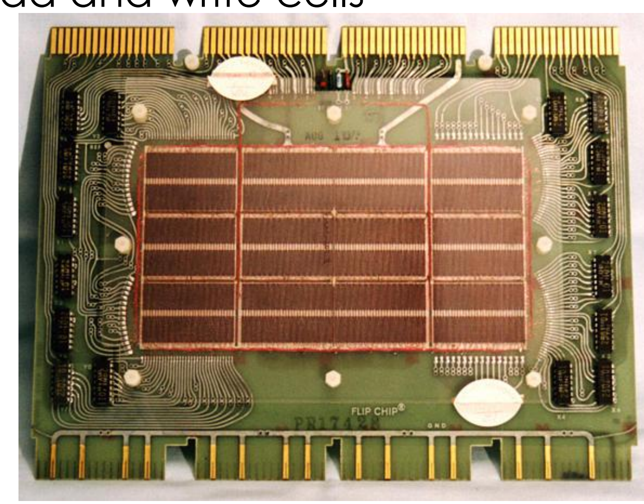
- Williams Tube in Manchester Mark I (1947) unreliable.
- Forrester invented core memory for MIT Whirlwind (1940-50s) in response
- First large-scale, reliable main memory

Magnetic Technology

- Core memory stores bits using magnetic polarity on ferrite cores
- Ferrite cores threaded onto 2D grid of wires
- Current pulses on X- and Y-axis could read and write cells

Performance

- Robust, non-volatile storage
- 1 microsecond core access time



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

Semiconductor Memory

Semiconductor Memory

- Static RAM (SRAM): cross-coupled inverters latch value
- Dynamic RAM (DRAM): charge stored on a capacitor

Advent of Semiconductor Memory

- Technology became competitive in early 1970s
- Intel founded to exploit market for semiconductor memory

Dynamic Random Access Memory (DRAM)

- Charge on a capacitor maps to logical value
- Intel 1103 was first commercial DRAM
- Semiconductor memory quickly replaced core memory in 1970's

Semiconductor Memory

Advent of Semiconductor Memory

- Technology became competitive in early 1970s
- Intel founded to exploit market for semiconductor memory
- Early semiconductor memory was static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters)

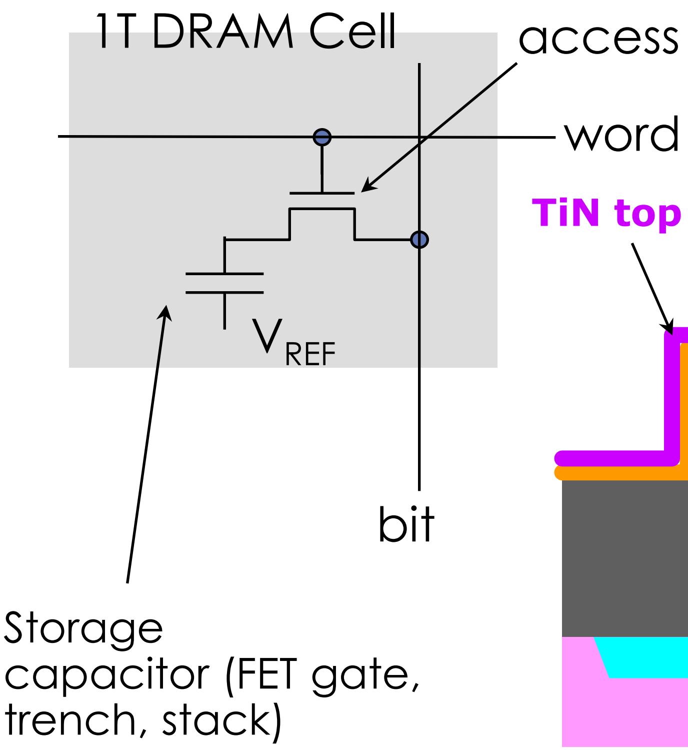
- **Advent of Semiconductor Memory**

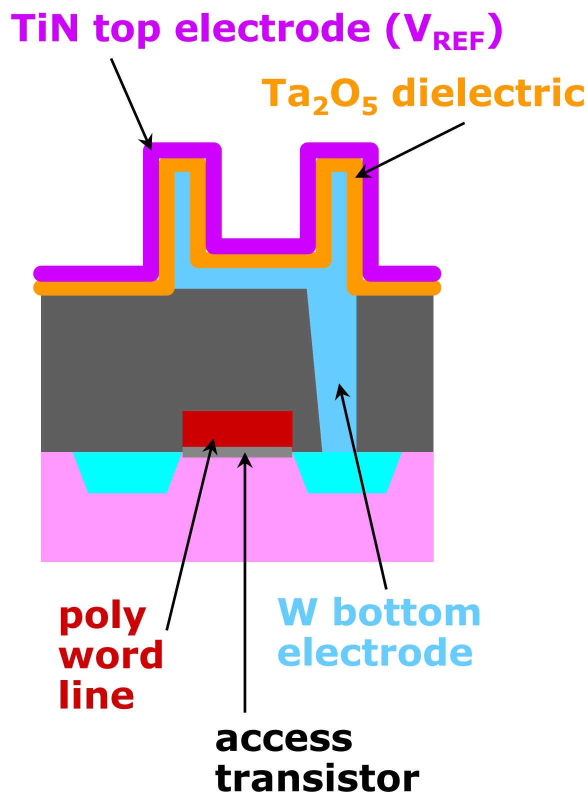
25 October – Class Discussion

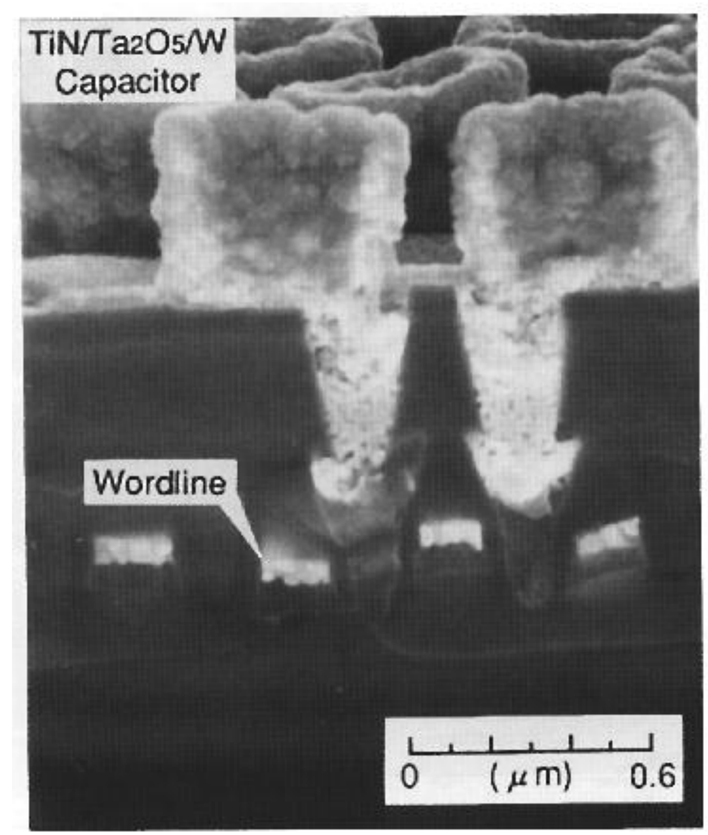
Roughly one reading per class. Do not wait until the day before!

1. Jouppi. "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers."
2. Kim et al. "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches."
3. Fromm et al. "The energy efficiency of IRAM architectures"
4. Lee et al. "Phase change memory architecture and the quest for scalability"

DRAM – Dennard 1968



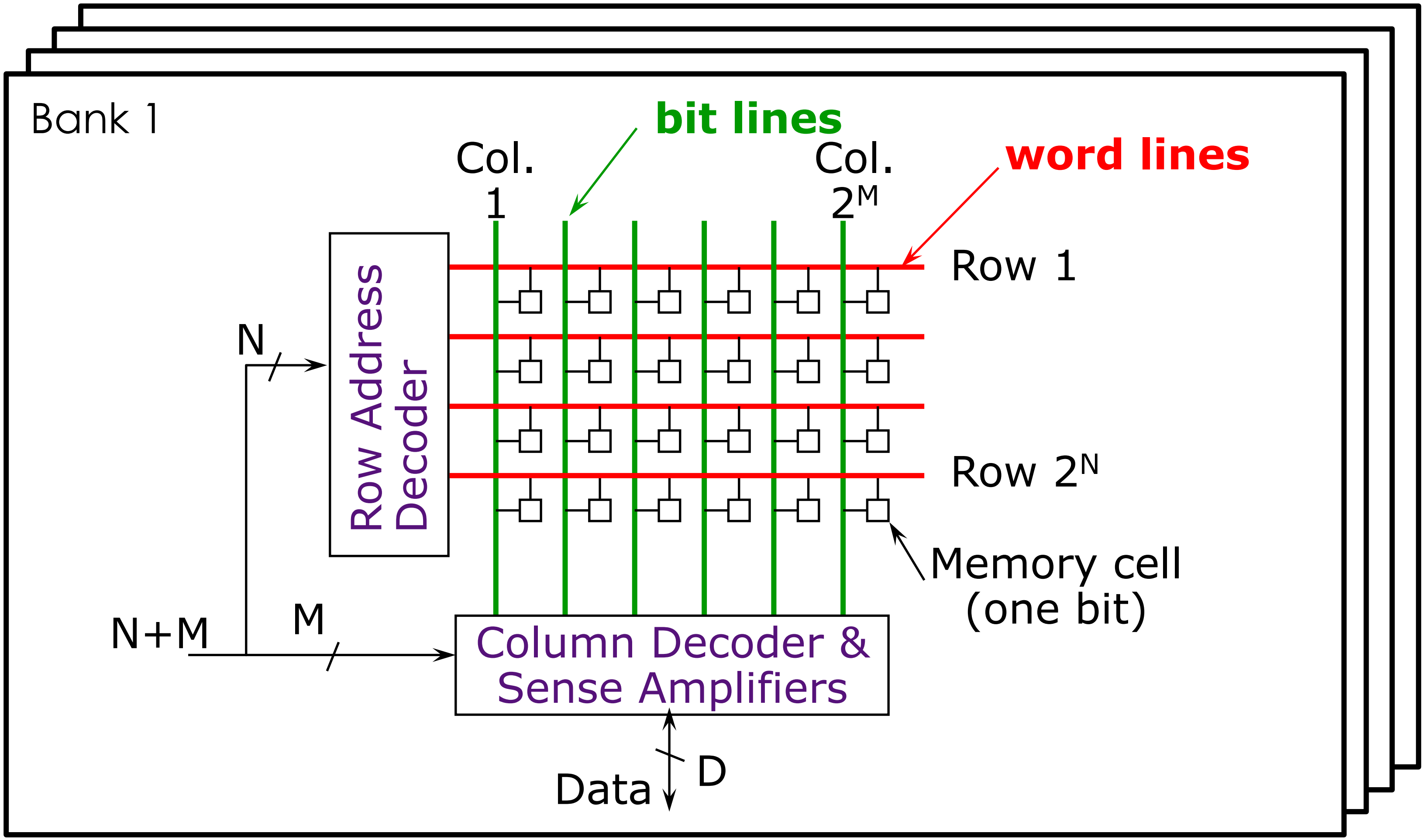




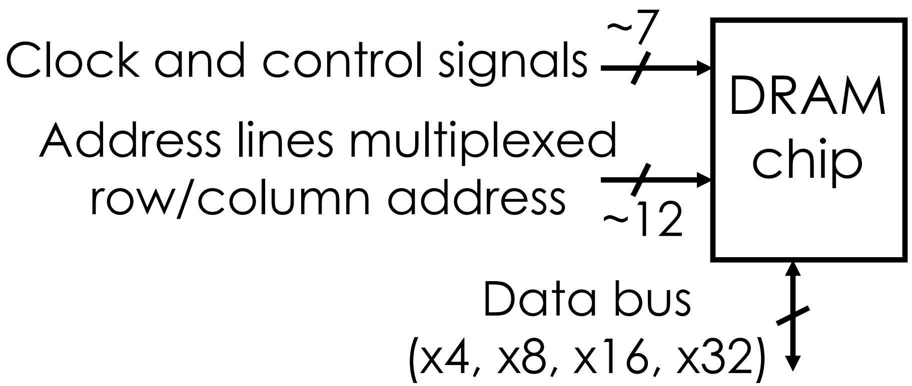
DRAM Chip Architecture

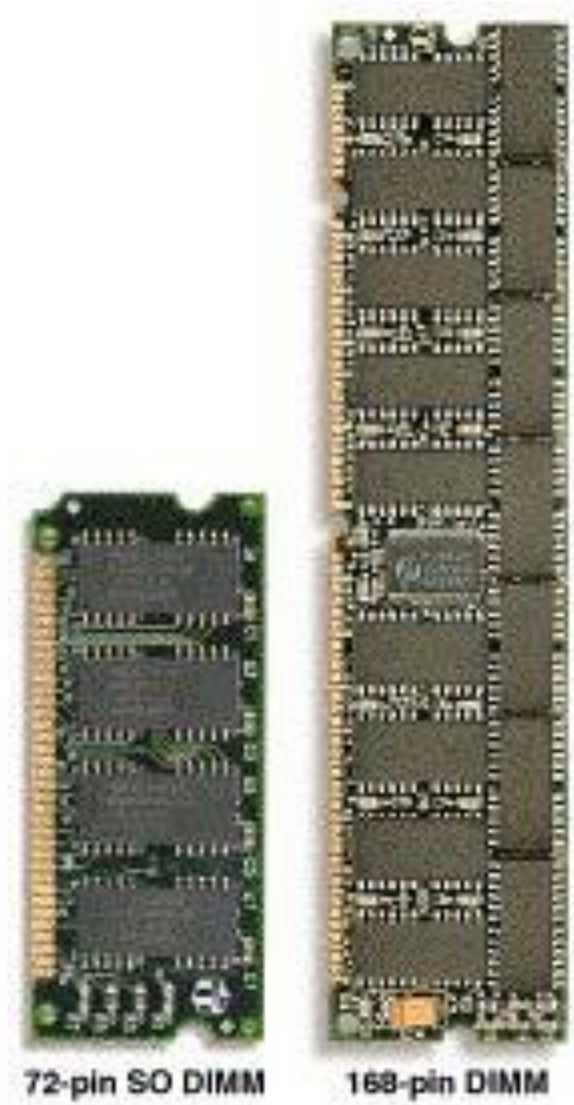
-- Chip organized into 4-8 logical banks, which can be accessed in parallel

-- Each bank implements 2-D array of bits



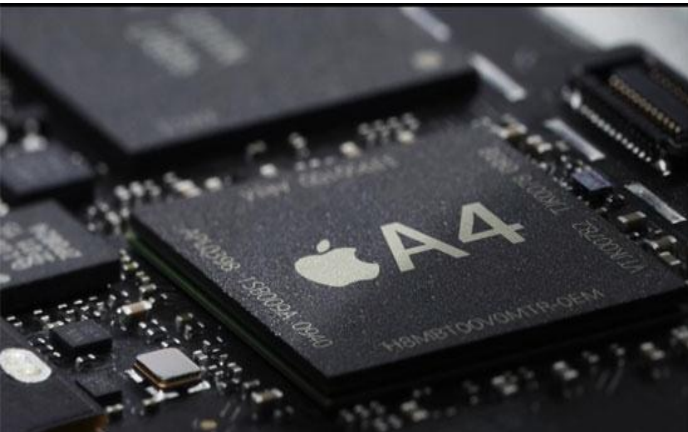
Packaging & Memory Channel



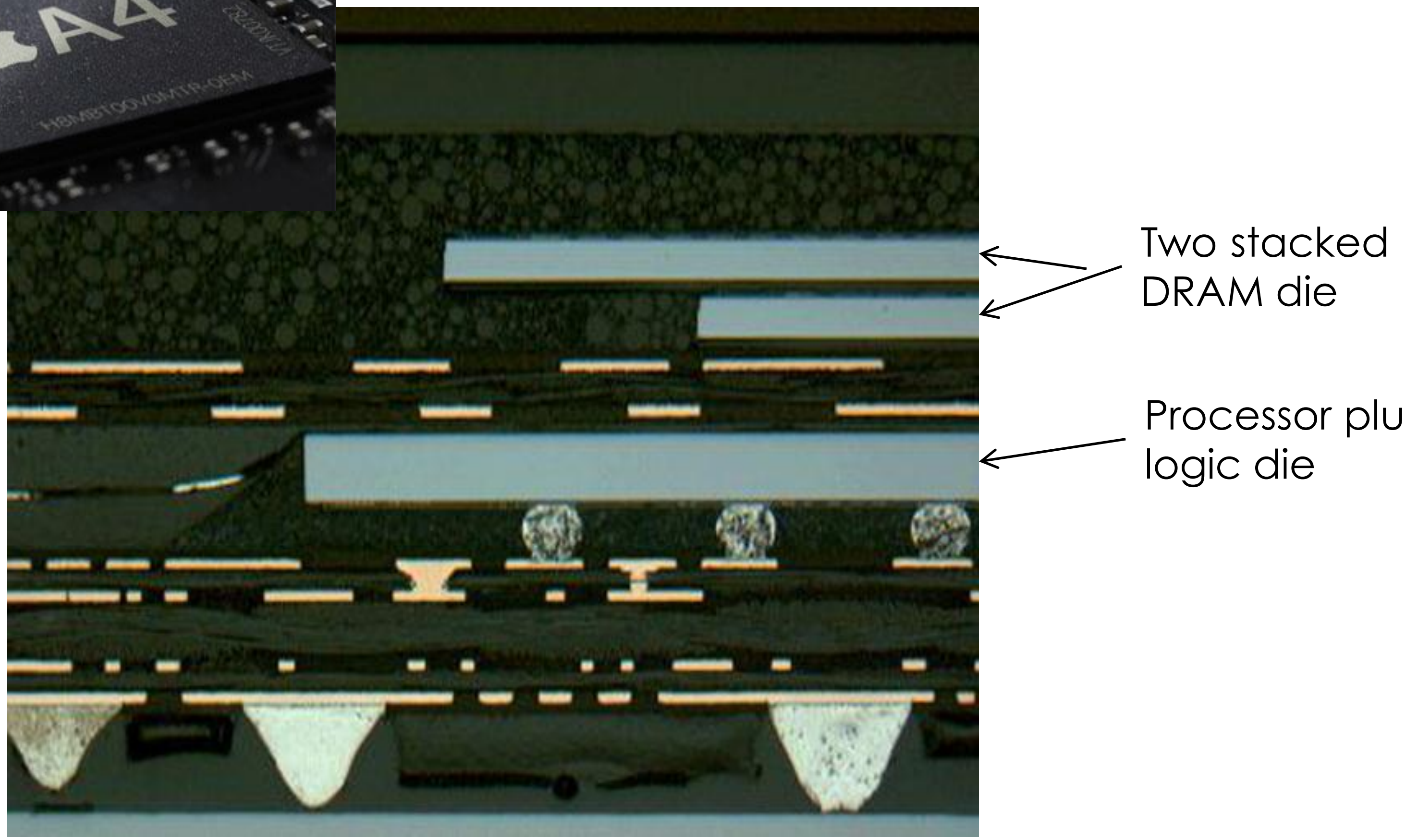


- DIMM (Dual Inline Memory Module): Multiple chips sharing the same clock, control, and address signals.
- Data pins collectively supply wide data bus. For example, four x16 chips supply 64b data bus.

Packaging & 3D Stacking



[Apple A4 package on circuit board]



[Apple A4 package cross-section, iFixit 2010]

DRAM Operation

1. Activate (ACT)
   - Decode row address (RAS). Enable the addressed row (e.g., 4Kb)
   - Bitline and capacitor share charge
   - Sense amplifiers detect small change in voltage.
   - Latch row contents (a.k.a. row buffer)

2. Read or Write
   - Decode column address (CAS). Select subset of row (e.g., 16b)
   - If read, send latched bits to chip pins
   - If write, modify latched bits and charge capacitor
   - Can perform multiple CAS on same row without RAS (i.e., buffer hit)

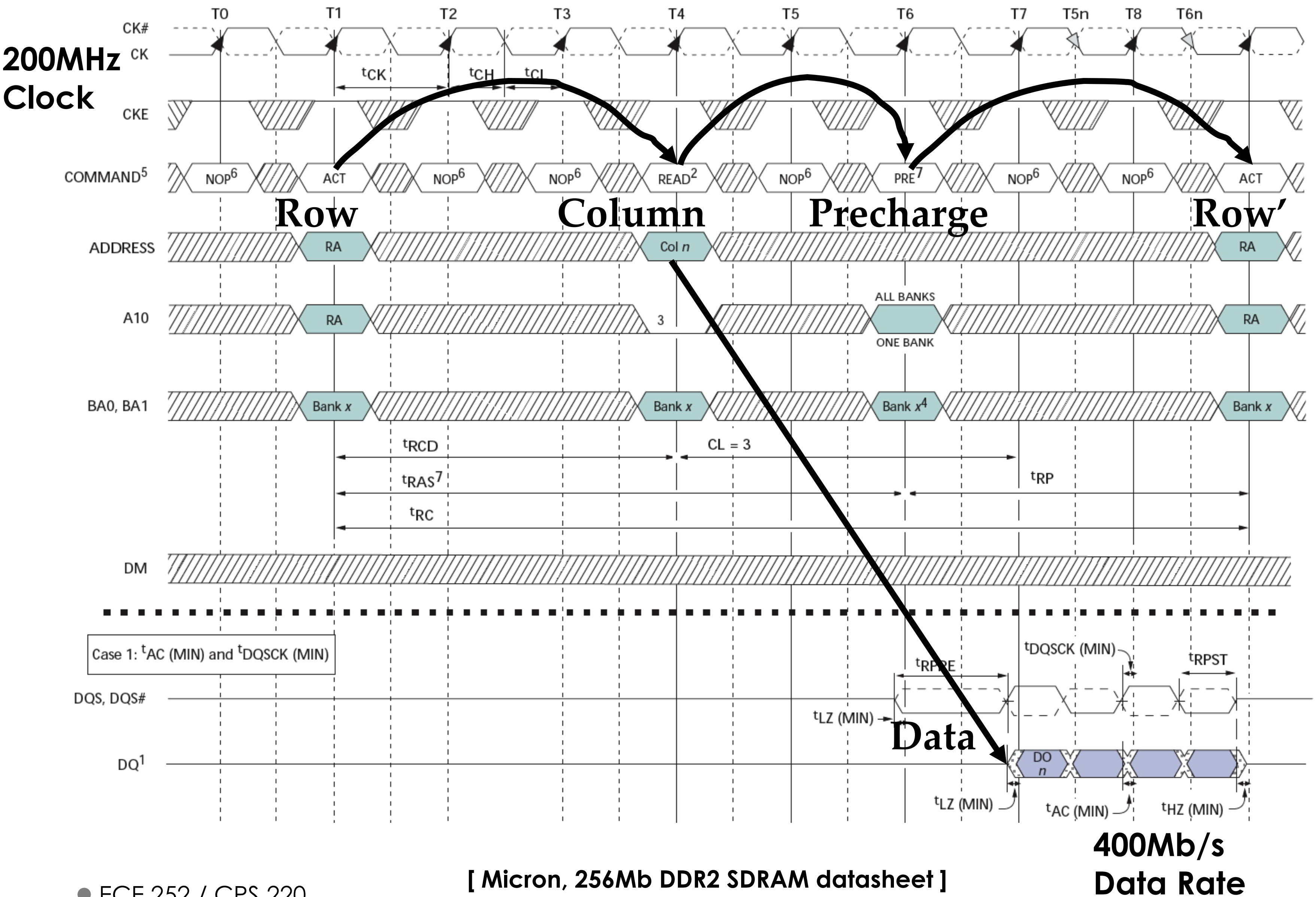
3. Precharge
   - Charge bit lines to buffer to prepare for next row access

DRAM Controller

1. Interfaces to Processor Datapath
   - Processor issues a load/store instruction
   - Memory address maps to particular chips, rows, columns

2. Implements Control Protocol
   - (1) Activate a row, (2) Read/write the row, (3) Precharge
   - Enforces timing parameters between commands
   - Latency of each step is approximately 15-20ns
   - Various DRAM standards (DDR, RDRAM) have different signals

Double Data Rate (DDR*) DRAM



[Micron, 256Mb DDR2 SDRAM datasheet]

Processor-Memory Bottleneck

Memory is usually a performance bottleneck

- Processor limited by memory bandwidth and latency

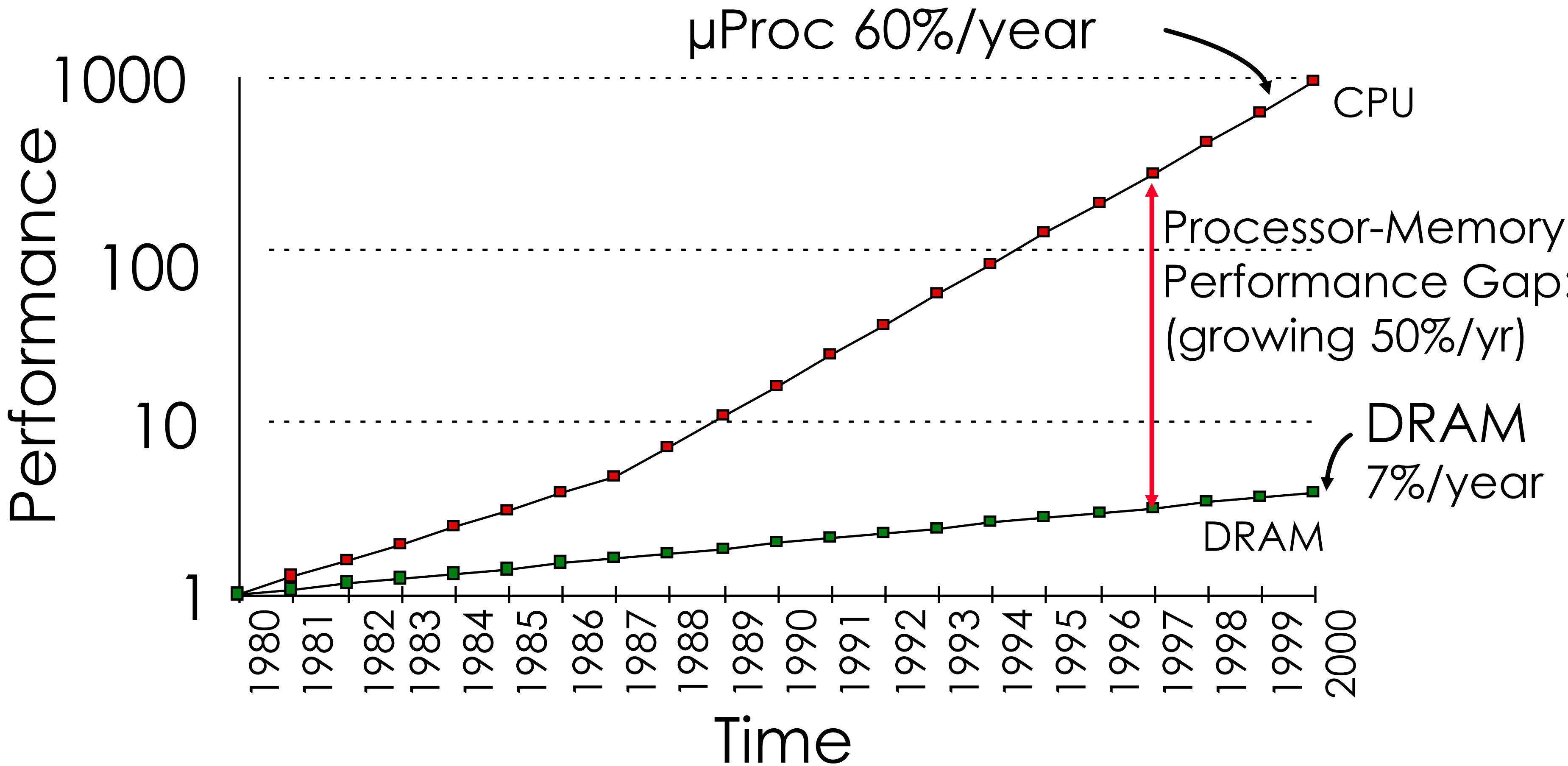
Latency (time for single transfer)

- Memory access time >> Processor cycle time
- Example: 60ns latency translates into 60 cycles for 1GHz processor

Bandwidth (number of transfers per unit time)

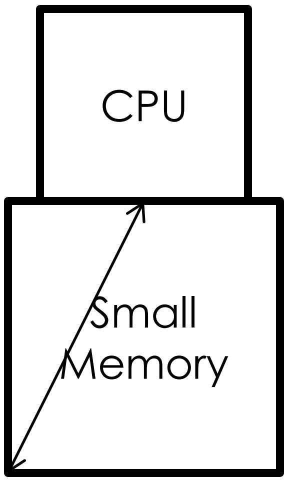
- Every instruction is fetched from memory
- Suppose M is fraction of loads/stores in a program
- On average,1+M memory references per instruction
- For CPI = 1, system must supply 1+M memory transfers per cycle.

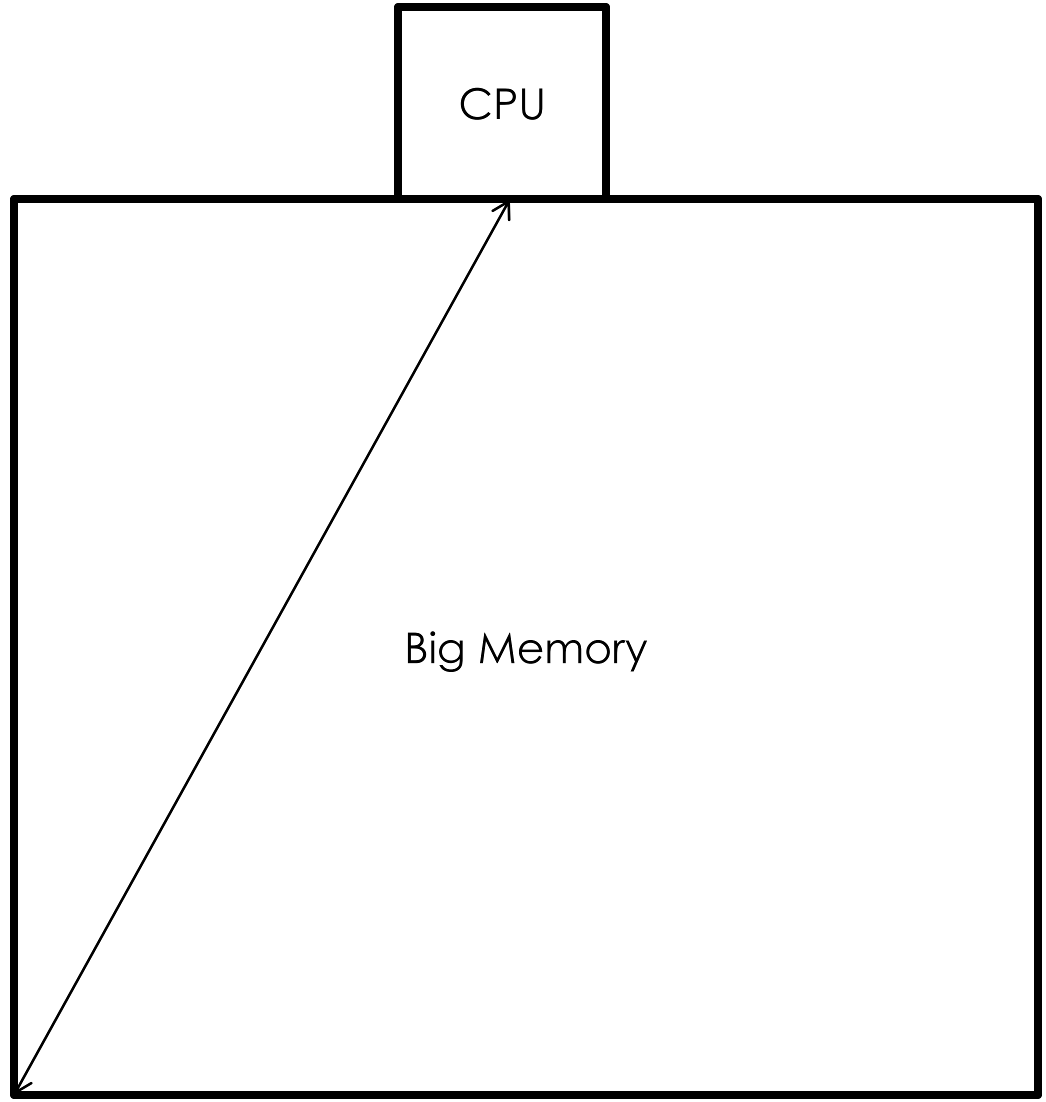
Processor-Memory Latency



Consider processor. Four-way superscalar. 3GHz clock. In 100ns required to access DRAM once, processor could execute 1,200 instructions

Distance Affects Latency



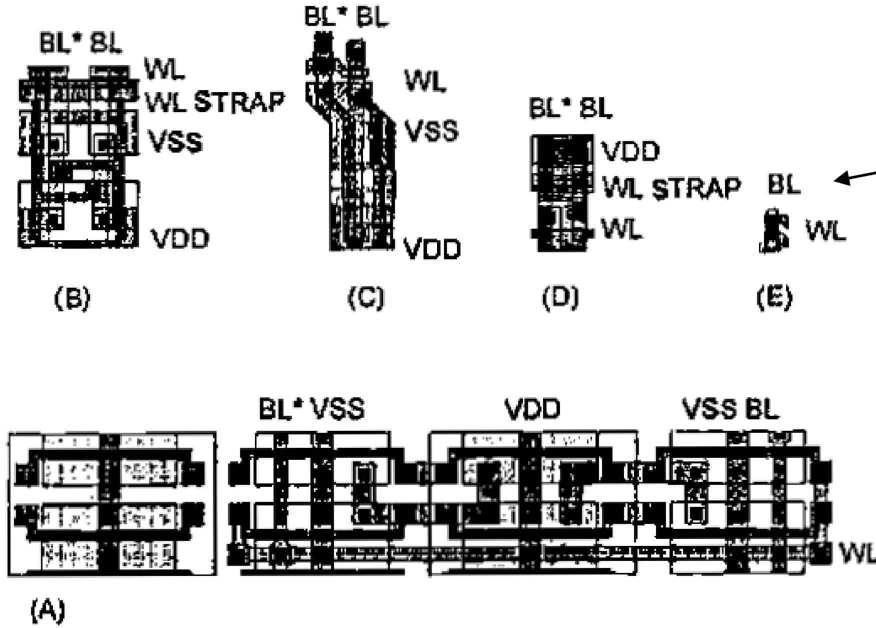


- Signals have further to travel
- Fan out to more locations

Memory Cell Size

On-Chip SRAM in logic chip

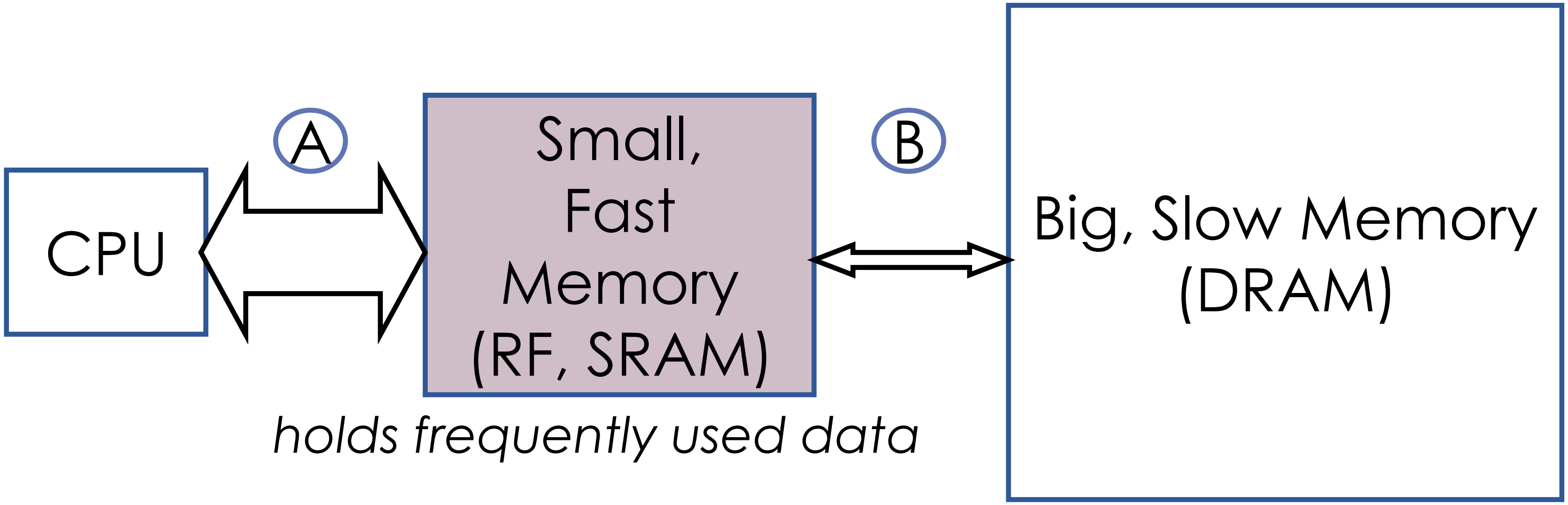
DRAM on memory chip



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)

Off-chip DRAM has higher density than on-chip SRAM.
[Foss, "Implementing Application-Specific Memory", ISSCC 1996]

Memory Hierarchy



Capacity	Register (RF) << SRAM << DRAM
Latency	Register (RF) << SRAM << DRAM
Bandwidth	on-chip >> off-chip

Consider a data access.

If data is located in fast memory, latency is low (e.g., SRAM).

If data is not located in fast memory, latency is high (e.g., DRAM).

Memory Hierarchy Management

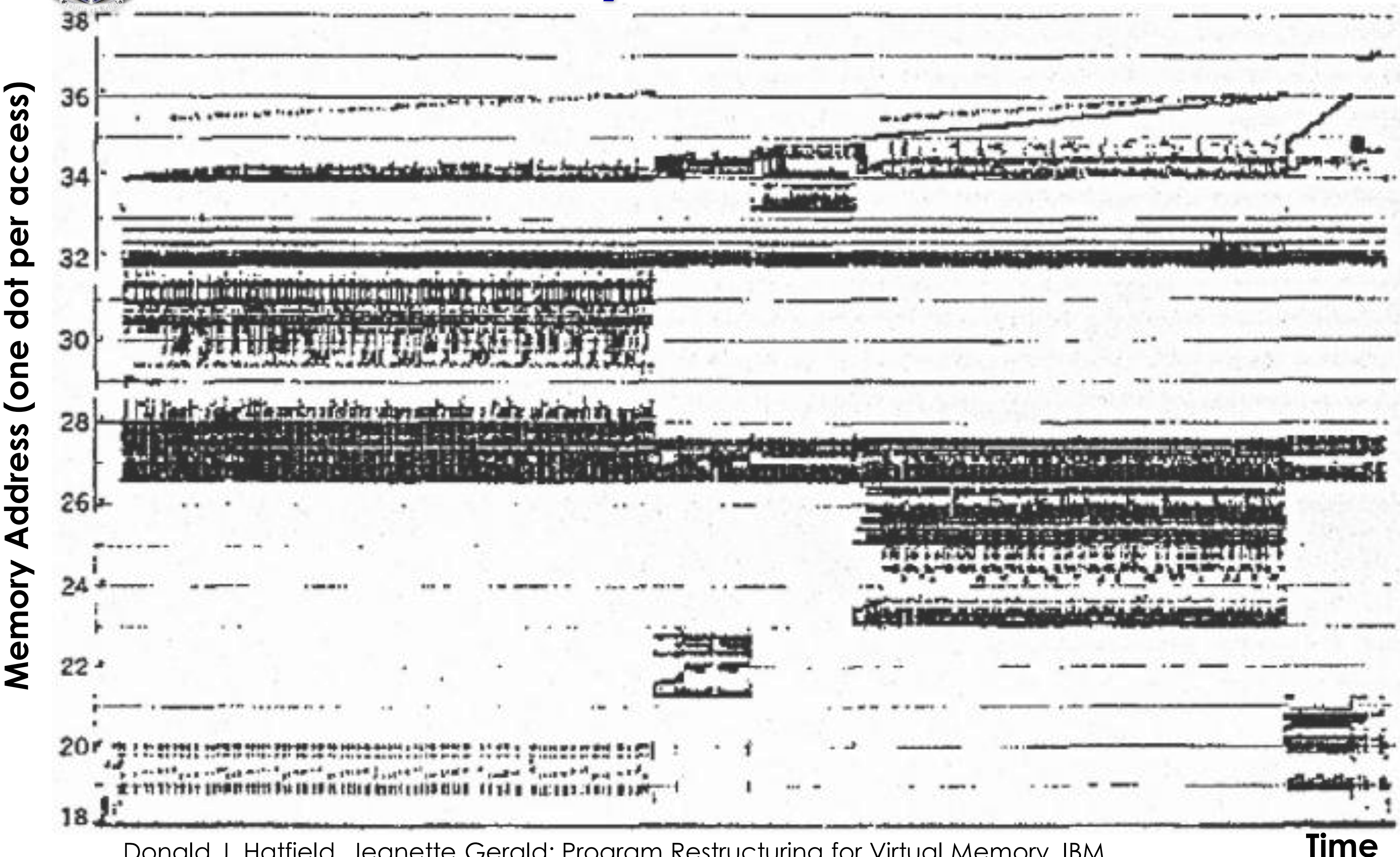
Small & Fast (Registers)

- Instruction specifies address (e.g., R5)
- Implemented directly as register file
- Hardware might dynamically manage register usage
- Examples: stack management, register renaming

Large & Slow (SRAM and DRAM)

- Address usually computed from values in registers (e.g., ld R1, x(R2))
- Implemented directly as hardware-managed cache hierarchy
- Hardware decides what data is kept in faster memory
- Software may provide hints

Real Memory Reference Patterns



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

Predictable Patterns

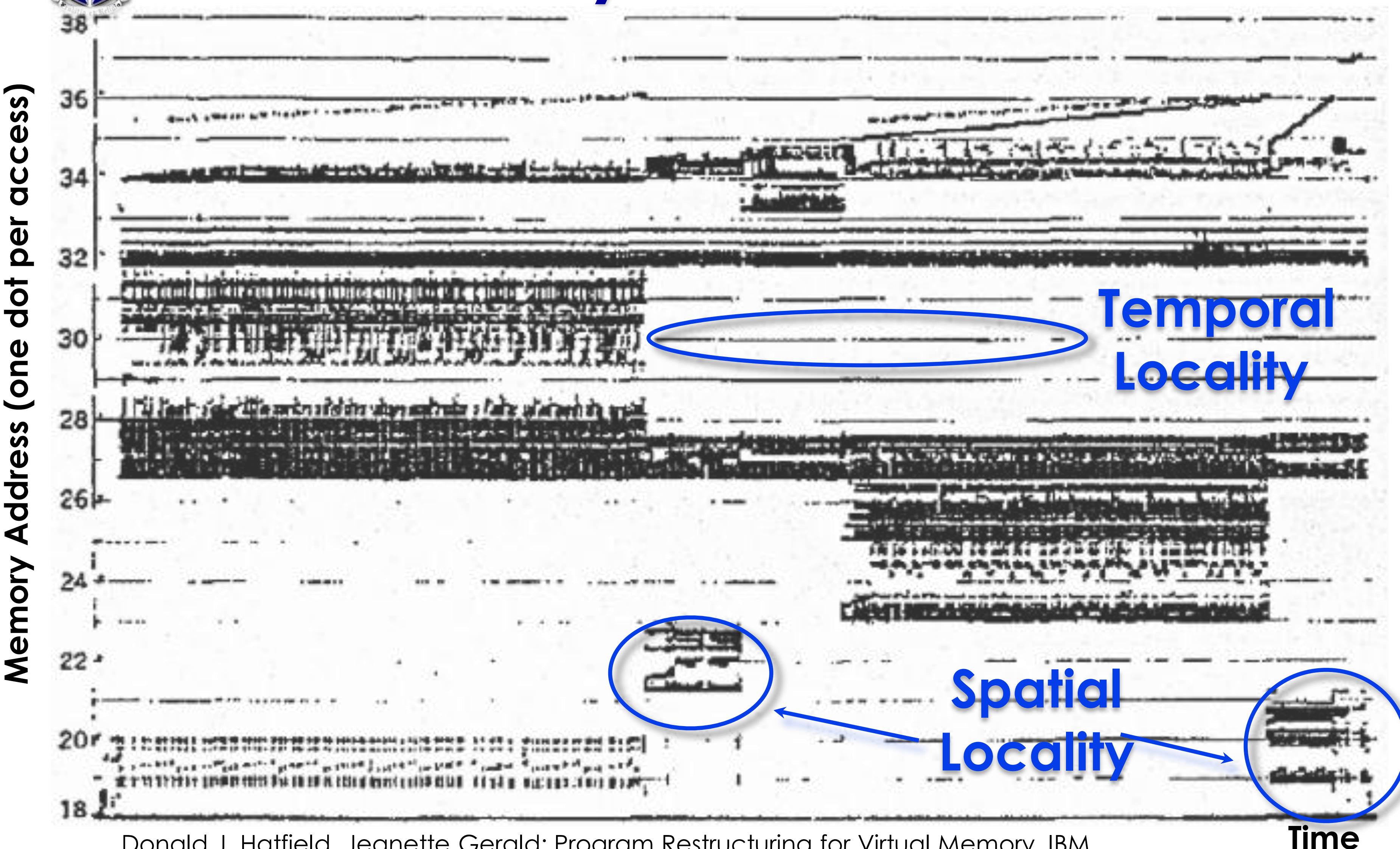
Temporal Locality

If a location is referenced once,
the same location is likely to referenced again in the near future.

Spatial Locality

If a location is referenced once,
nearby locations are likely to be referenced in the near future.

Real Memory Reference Patterns



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

Caches

Caches exploit predictable patterns

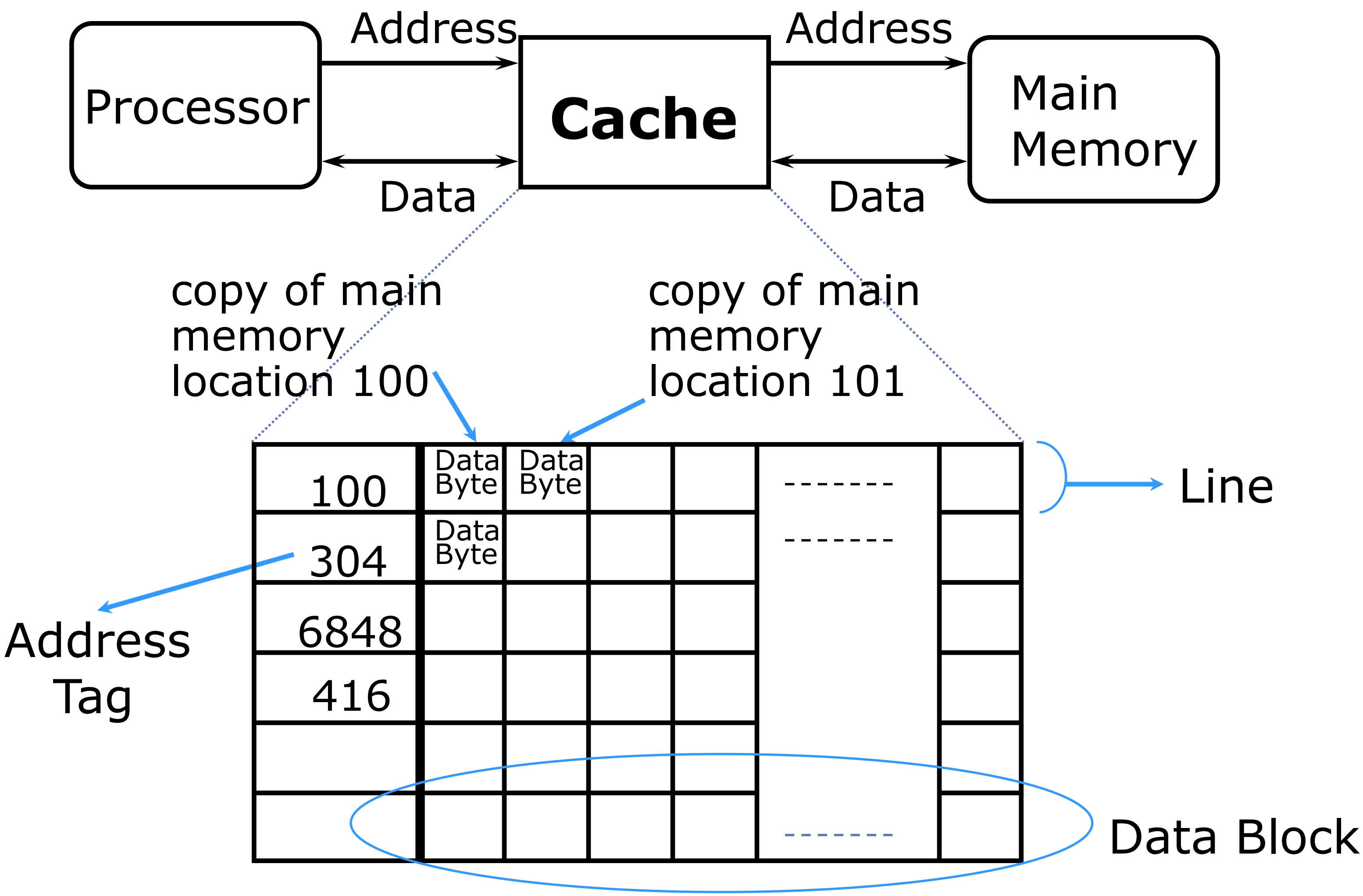
Temporal Locality

Caches remember the contents of recently accessed locations

Spatial Locality

Caches fetch blocks of data nearby recently accessed locations

Caches


Processor
Address
Data
Cache
Address
Data
Main Memory
copy of main memory location 100
copy of main memory location 101
100
304
6848
416
Data Byte
Data Byte
Data Byte

Line
Address Tag
Data Block

Cache Controller

Controller examines address from datapath and searches cache for matching tags.

Cache Hit – address found in cache

- Return copy of data from cache

Cache Miss – address not found in cache

- Read block of data from main memory.
- Wait for main memory
- Return data to processor and update cache
- What is the update policy?

Cache Controller

Controller examines address from datapath and searches cache for matching tags.

Cache Hit – address found in cache

- Return copy of data from cache

Cache Miss – address not found in cache

- Read block of data from main memory.
- Wait for main memory
- Return data to processor and update cache
- What is the update policy?

Data Placement Policy

Fully Associative

- Update – place data in any cache line (a.k.a. block)
- Access – search entire cache for matching tag

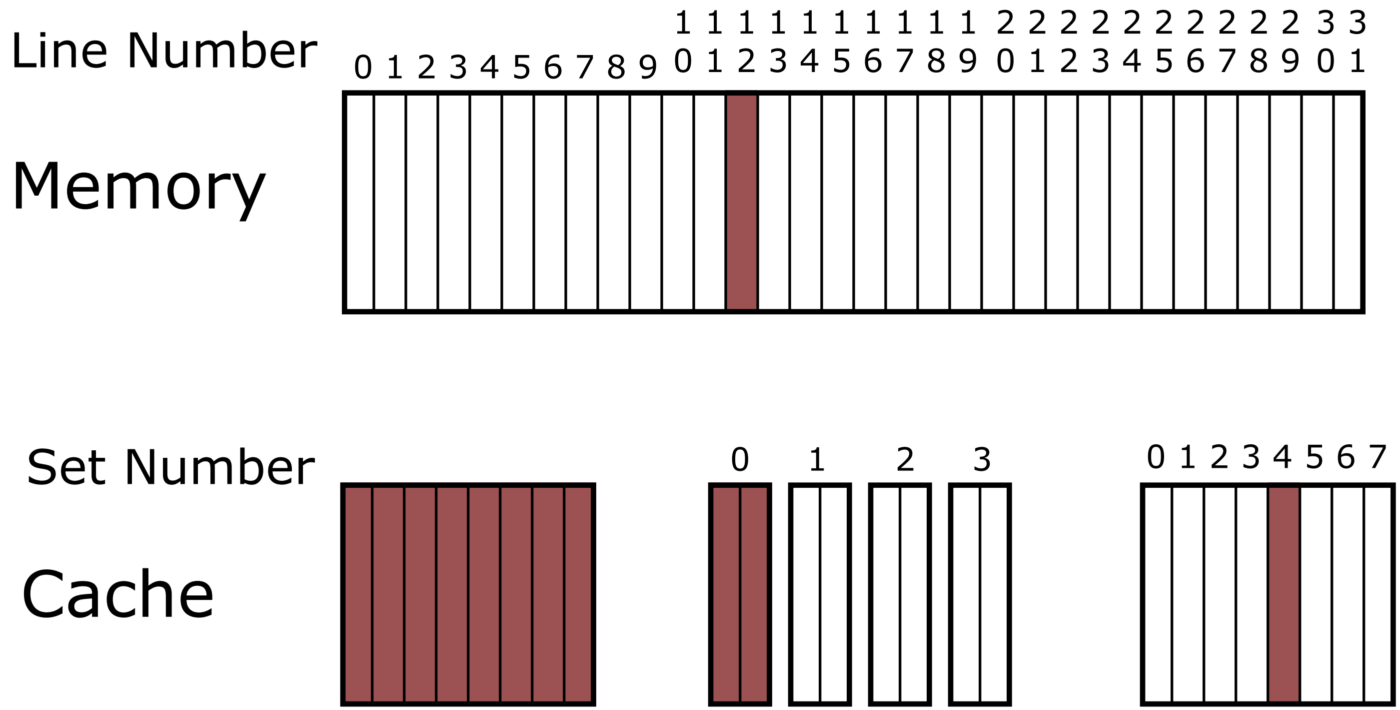
Set Associative

- Update – place data within set of lines determined by address
- Access – identify set from address, search set for matching tag

Direct Mapped

- Update – place data in specific line determined by address
- Access – identify line from address, check for matching tag

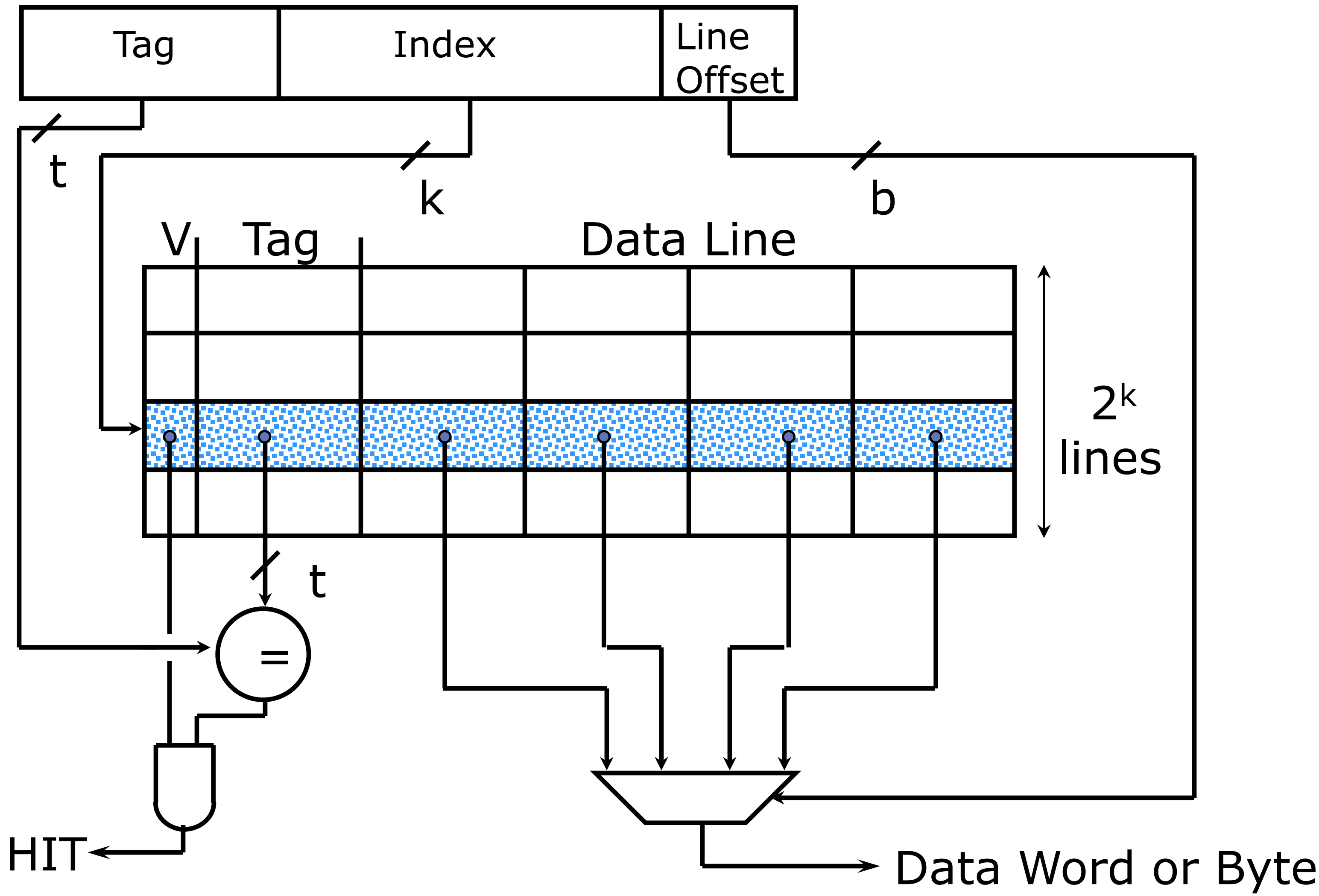
Placement Policy



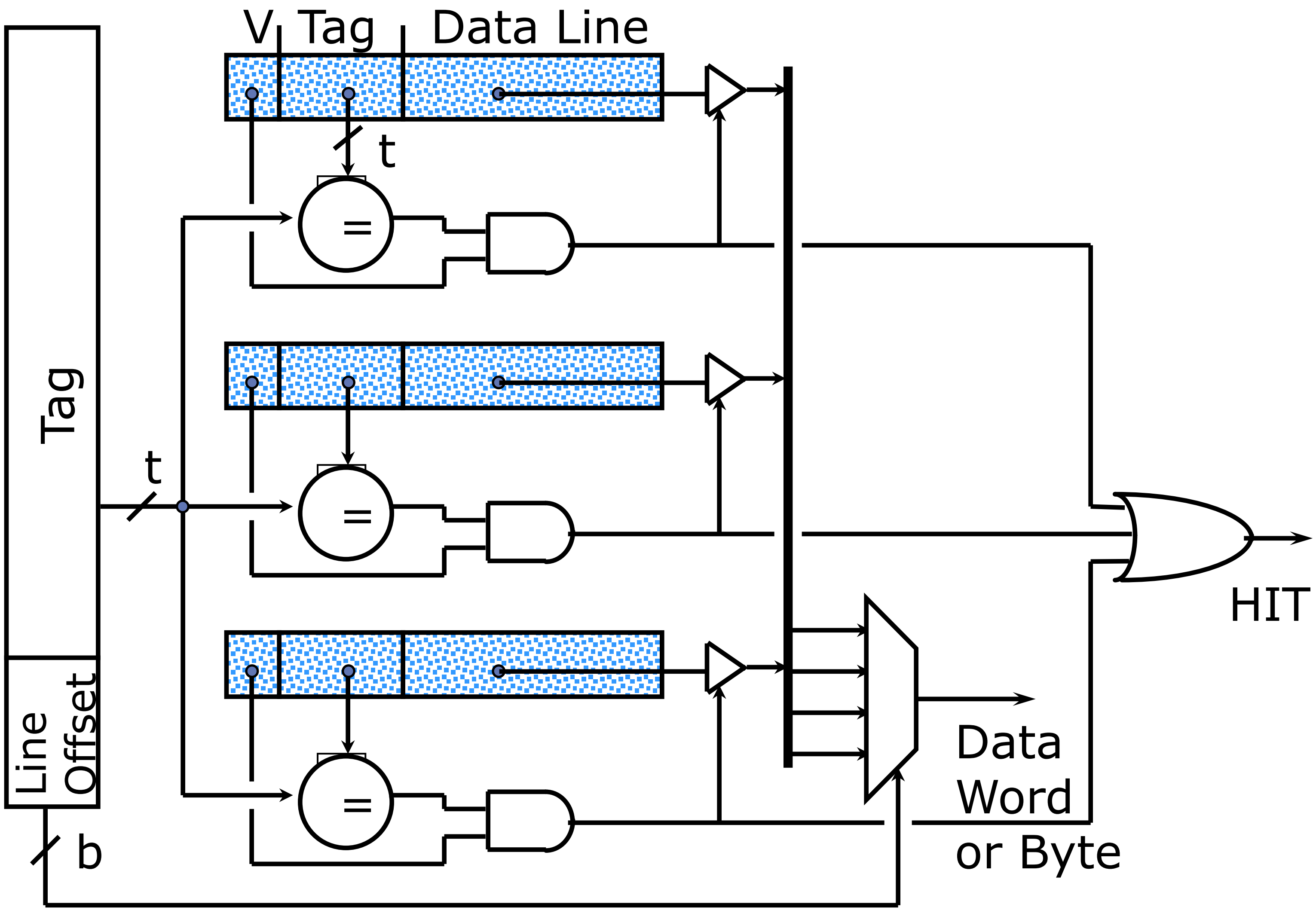
Line 12 can be placed

Fully Associative	(2-way) Set Associative	Direct Mapped
anywhere	anywhere in set 0 *(12 mod 4)*	only into block 4 *(12 mod 8)*

Direct-Mapped Cache



Fully Associative Cache



Update/Replacement Policy

In an associative cache, which cache line in 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 (e.g., 2-way)
- Approximation algorithms exist for larger sets

First-In, First-Out (FIFO)

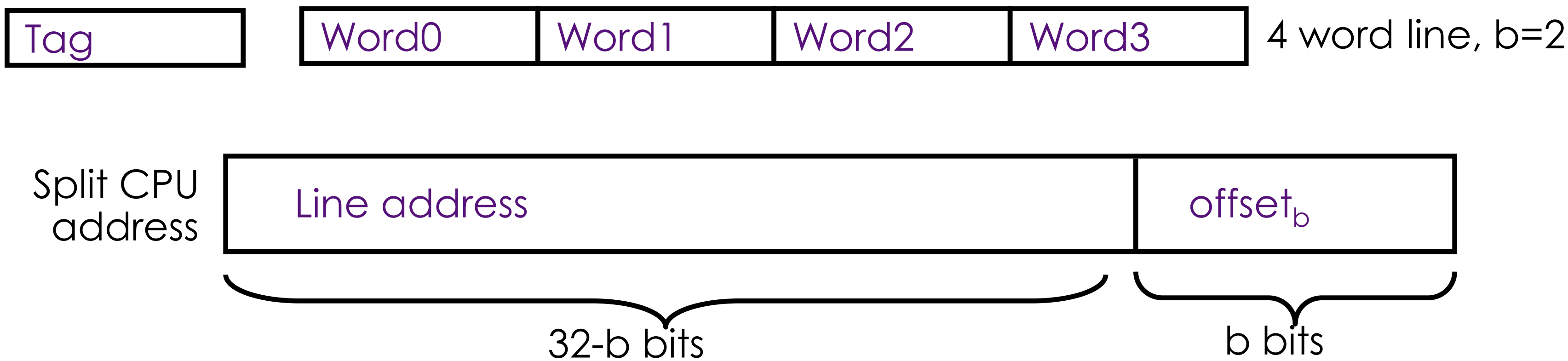
- Used in highly associative caches

Not Most Recently Used (NMRU)

- Implements FIFO with an exception for most recently used blocks

Line Size and Spatial Locality

Line is unit of transfer between the cache and memory



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

Larger line size has distinct hardware advantages
- -- less tag overhead
- -- exploit fast burst transfers from DRAM
- -- exploit fast burst transfers over wide bus

What are the disadvantages of increasing block size?
- -- fewer lines, more line conflicts
- -- can waste bandwidth depending on application's spatial locality

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)
- Alvin Lebeck (Duke)
- David Patterson (UCB)
- Daniel Sorin (Duke)