CSE 490/590 Computer Architecture

Cache I

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time…

- Pipelining hazards
  - Structural hazards
  - Data hazards
  - Control hazards
- Data hazards
  - Stall
  - Bypass
- Control hazards
  - Jump
  - Conditional branch

Branch Delay Slots (expose control hazard to software)

- Change the ISA semantics so that the instruction that follows a jump or branch is always executed
  - gives compiler the flexibility to put in a useful instruction where normally a pipeline bubble would have resulted.

```
I1    096    ADD
I2    100    BEQZ r1 +200
I3    104    ADD  <-------------
I4    304    ADD
```

Delay slot instruction executed regardless of branch outcome

- Other techniques include more advanced branch prediction, which can dramatically reduce the branch penalty... *to come later*

Branch Pipeline Diagrams (branch delay slot)

time	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) 100: BEQZ +200		IF_2	ID_2	EX_2	MA_2	WB_2			
(I_3) 104: ADD			IF_3	ID_3	EX_3	MA_3	WB_3		
(I_4) 304: ADD				IF_4	ID_4	EX_4	MA_4	WB_4	

Resource Usage	*time*	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
	IF	I_1	I_2	I_3	I_4					
	ID		I_1	I_2	I_3	I_4				
	EX			I_1	I_2	I_3	I_4			
	MA				I_1	I_2	I_3	I_4		
	WB					I_1	I_2	I_3	I_4	

Why an Instruction may not be dispatched every cycle (CPI>1)

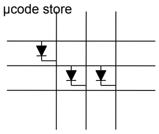
- Full bypassing may be too expensive to implement
  - typically all frequently used paths are provided
  - some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
- Loads have two-cycle latency
  - Instruction after load cannot use load result
  - MIPS-I ISA defined *load delay slots*, a software-visible pipeline hazard (compiler schedules independent instruction or inserts NOP to avoid hazard). Removed in MIPS-II (pipeline interlocks added in hardware)
    - » MIPS:"**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages"
- Conditional branches may cause bubbles
  - kill following instruction(s) if no delay slots

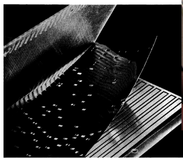
Early Read-Only Memory Technologies

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

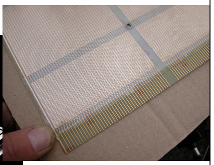
Punched paper tape, instruction stream in Harvard Mk 1

Diode Matrix, EDSAC-2 µcode store





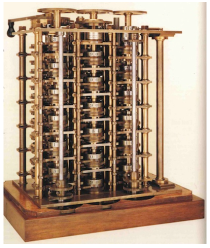
IBM Card Capacitor ROS



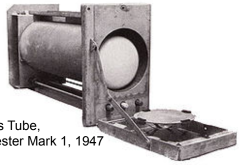
IBM Balanced Capacitor ROS

Early Read/Write Main Memory Technologies

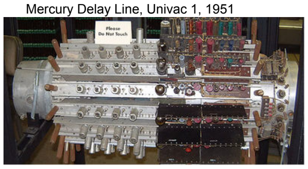
Babbage, 1800s: Digits stored on mechanical wheels



Williams Tube, Manchester Mark 1, 1947



Mercury Delay Line, Univac 1, 1951

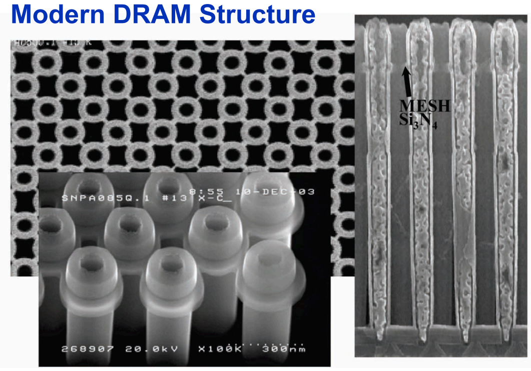


Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

Semiconductor Memory

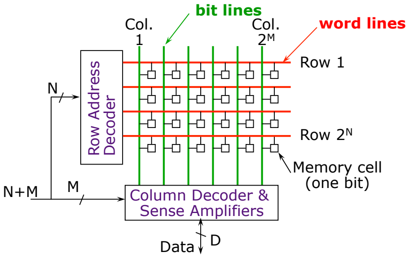
- Semiconductor memory began to be competitive in early 1970s
  – Intel formed to exploit market for semiconductor memory
  – Early semiconductor memory was Static RAM (SRAM). SRAM cell internals similar to a latch (cross-coupled inverters).
- First commercial Dynamic RAM (DRAM) was Intel 1103
  – 1Kbit of storage on single chip
  – charge on a capacitor used to hold value
- Semiconductor memory quickly replaced core in '70s

Modern DRAM Structure



[Samsung, sub-70nm DRAM, 2004]

DRAM Architecture



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

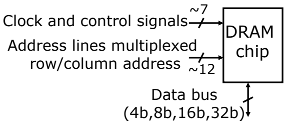
DRAM Operation

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

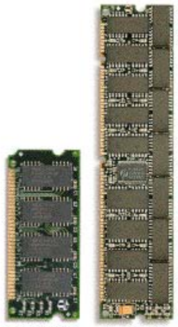
Each step has a latency of around 15-20ns in modern DRAMs
Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture

DRAM Packaging



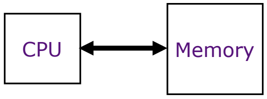
Clock and control signals
~7
Address lines multiplexed row/column address
~12
DRAM chip
Data bus (4b,8b,16b,32b)

- 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)



72-pin SO DIMM 168-pin DIMM

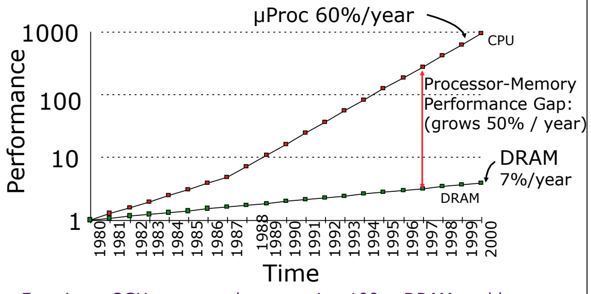
CPU-Memory Bottleneck



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

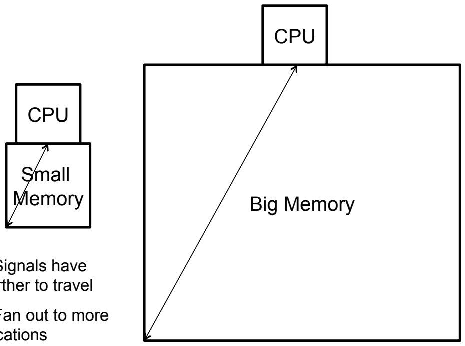
- Latency (time for a single access)
  Memory access time >> Processor cycle time
  Problematic
- Bandwidth (number of accesses per unit time)
  Increase the bus size, etc.
  Usually OK

Processor-DRAM Gap (latency)



Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

Physical Size Affects Latency

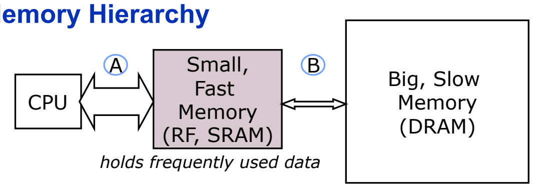


- Signals have further to travel
- Fan out to more locations

CSE 490/590 Administrivia

- Very important to attend
  - Recitations next week & the week after
- Guest lectures
  - There will be a couple guest lectures late Feb/early Mar.
- Quiz 1
  - Rescheduled
  - Fri, 2/11
  - Closed book, in-class
  - Includes lectures until last Monday (1/31)
  - Review: next Wed (2/9)

Memory Hierarchy

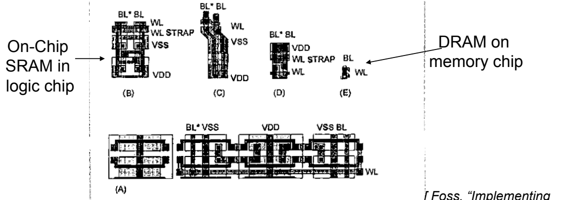


- *capacity*: Register << SRAM << DRAM *why?*
- *latency:* Register << SRAM << DRAM *why?*
- *bandwidth:* on-chip >> off-chip *why?*

On a data access:

if data ∈ fast memory ⇒ low latency access *(SRAM)*
If data ∉ fast memory ⇒ long latency access *(DRAM)*

Relative Memory Cell Sizes

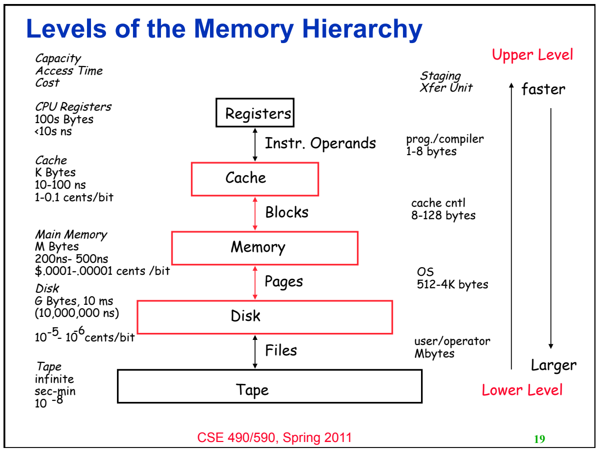


1 Memory cell in 0.5μm processes
a) Gate Array SRAM
b) Embedded SRAM
c) Standard SRAM (6T cell with local interconnect)
d) ASIC DRAM
e) Standard DRAM (stacked cell)

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

Memory	Process	Cell size (μm^2)	Cell efficiency	Bits in $100mm^2 (10^3)$	Gate size (μm^2)	Gate utilization	Gates in $100mm^2 (10^3)$
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.



Levels of the Memory Hierarchy

Capacity
Access Time
Cost

CPU Registers
100s Bytes
<10s ns

Cache
K Bytes
10-100 ns
1-0.1 cents/bit

Main Memory
M Bytes
200ns- 500ns
$.0001-.00001 cents /bit

Disk
G Bytes, 10 ms
(10,000,000 ns)

10^{-5}- 10^{-6}cents/bit

Tape
infinite
sec-min
10^{-8}

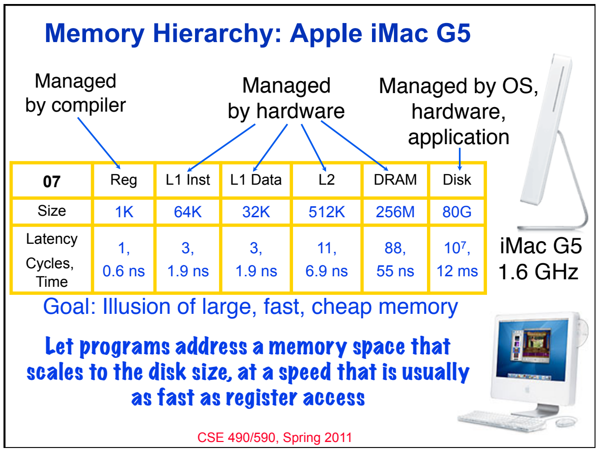
Staging
Xfer Unit

prog./compiler
1-8 bytes

cache cntl
8-128 bytes

OS
512-4K bytes

user/operator
Mbytes



Memory Hierarchy: Apple iMac G5

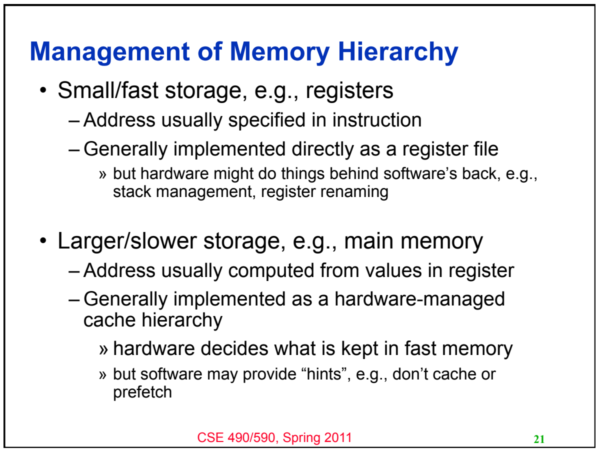
Managed
by compiler
Managed
by hardware
Managed by OS,
hardware,
application

07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

iMac G5
1.6 GHz

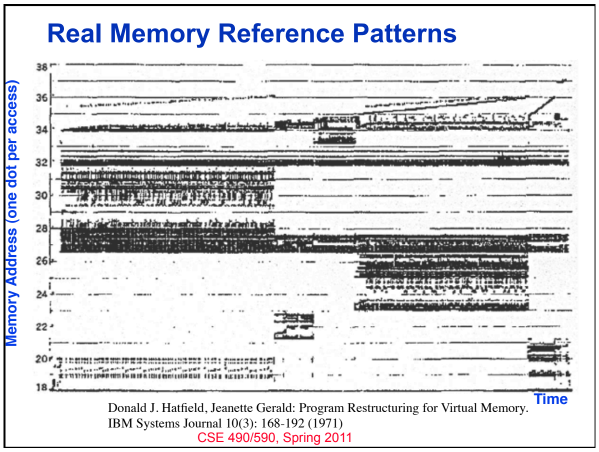
Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access



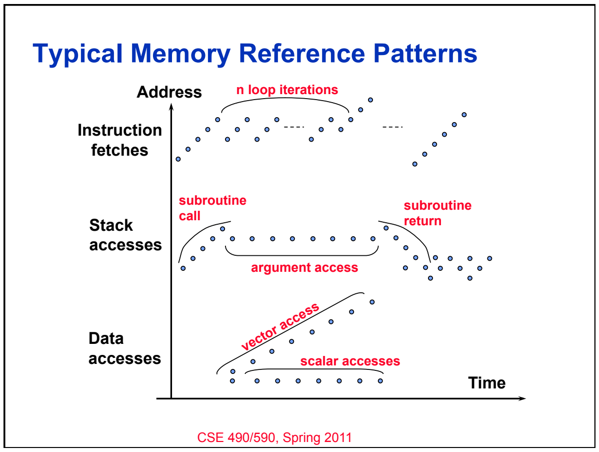
Management of Memory Hierarchy

- 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

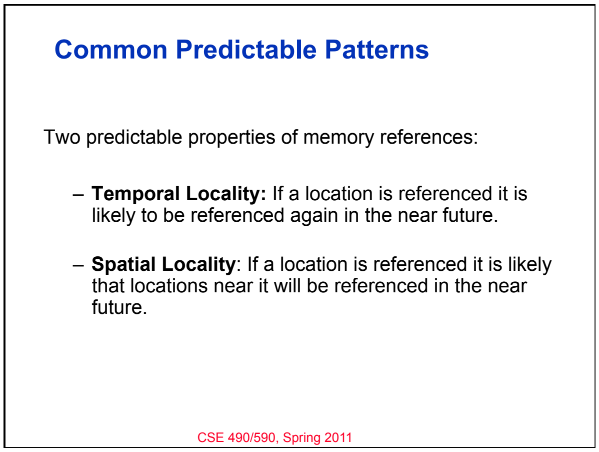


Real Memory Reference Patterns

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



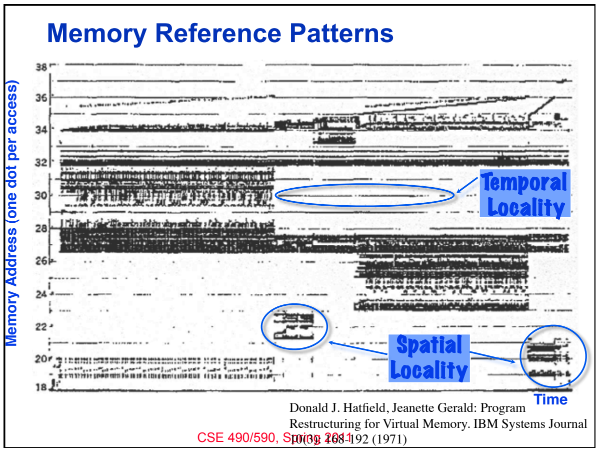
Typical Memory Reference Patterns



Common Predictable Patterns

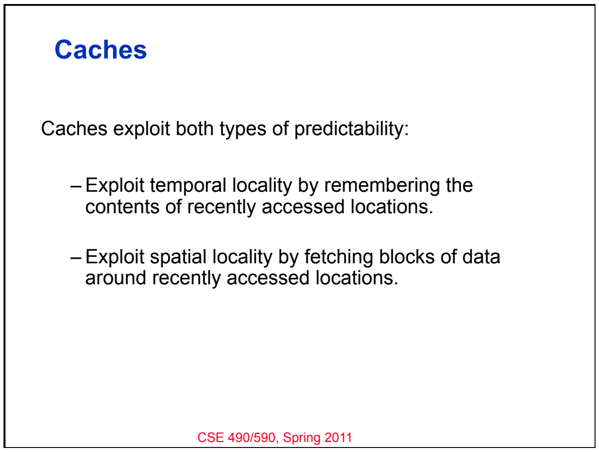
Two predictable properties of memory references:

- **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

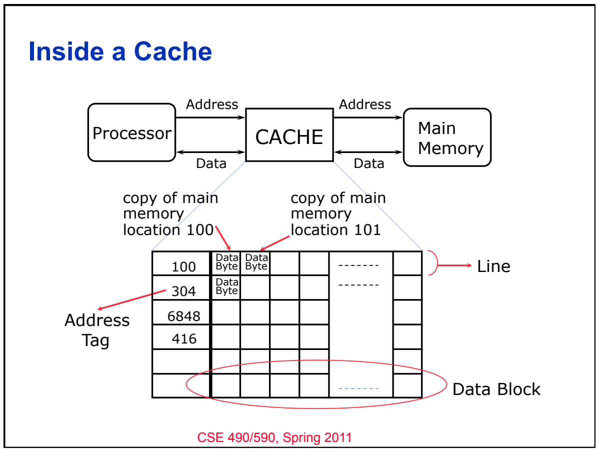
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)



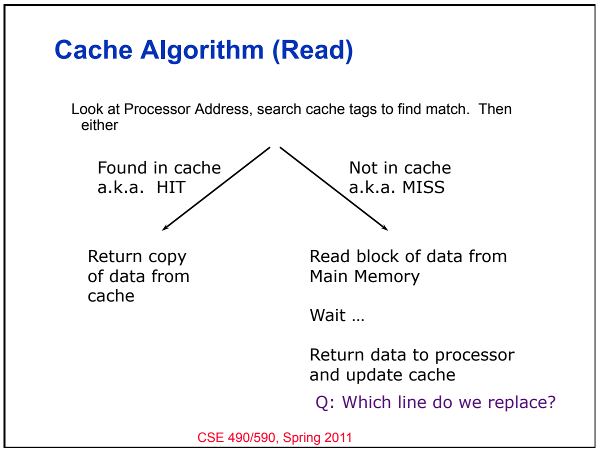
Caches

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

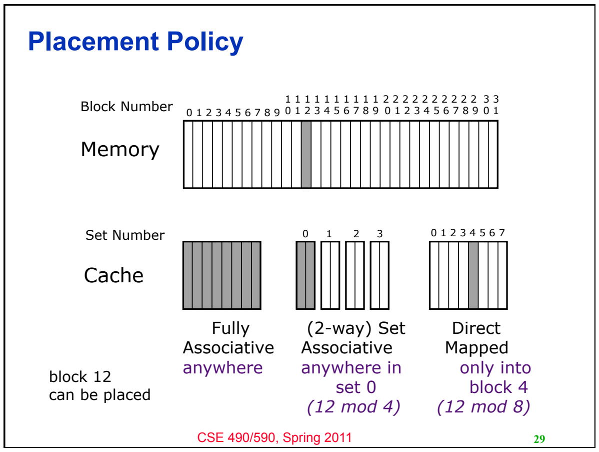
Not in cache
a.k.a. MISS

Read block of data from Main Memory

Wait …

Return data to processor and update cache

Q: Which line do we replace?



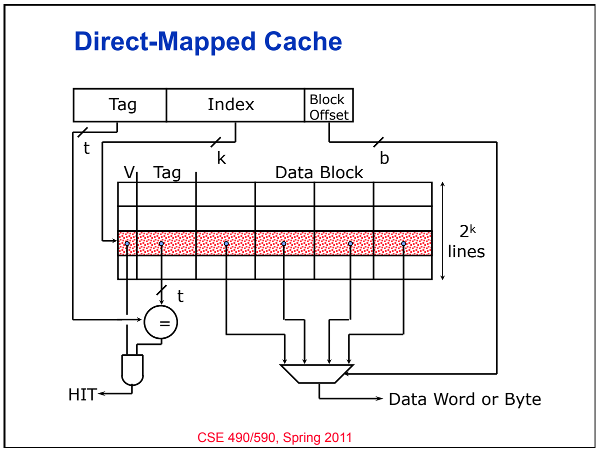
Placement Policy

Fully Associative
anywhere

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

Direct Mapped
only into block 4
(12 mod 8)

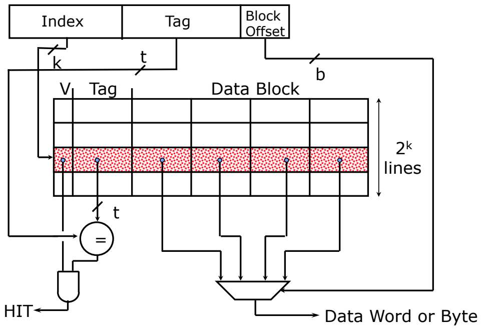
block 12 can be placed



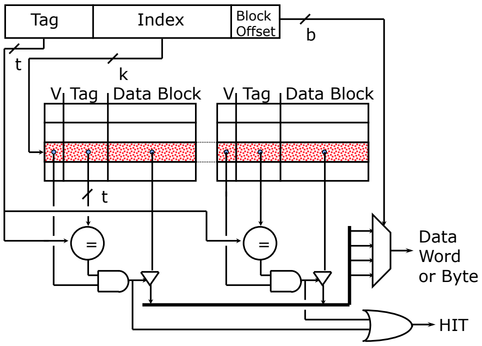
Direct-Mapped Cache

Direct Map Address Selection

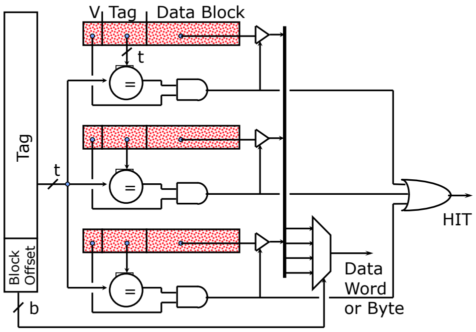
higher-order vs. lower-order address bits



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 Least Recently Used (NLRU)
  - FIFO with exception for most recently used block or blocks

This is a second-order effect. Why?

Replacement only happens on misses

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252