ECE 550D
Fundamentals of Computer Systems and Engineering

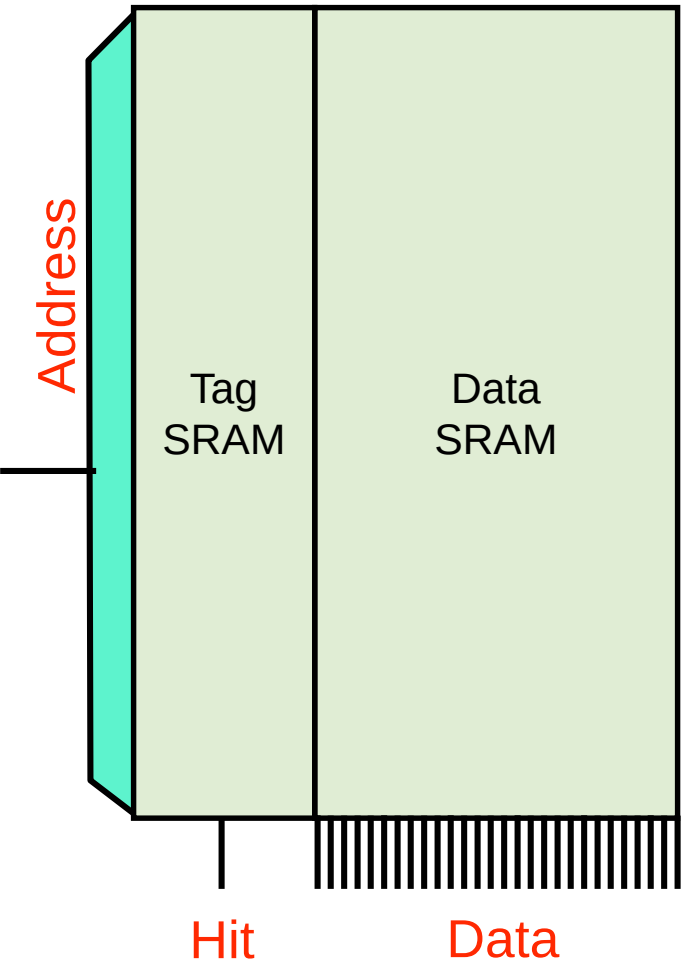
Fall 2023

Memory Hierarchy

Xin Li & Dawei Liu
Duke Kunshan University

Slides are derived from work by
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

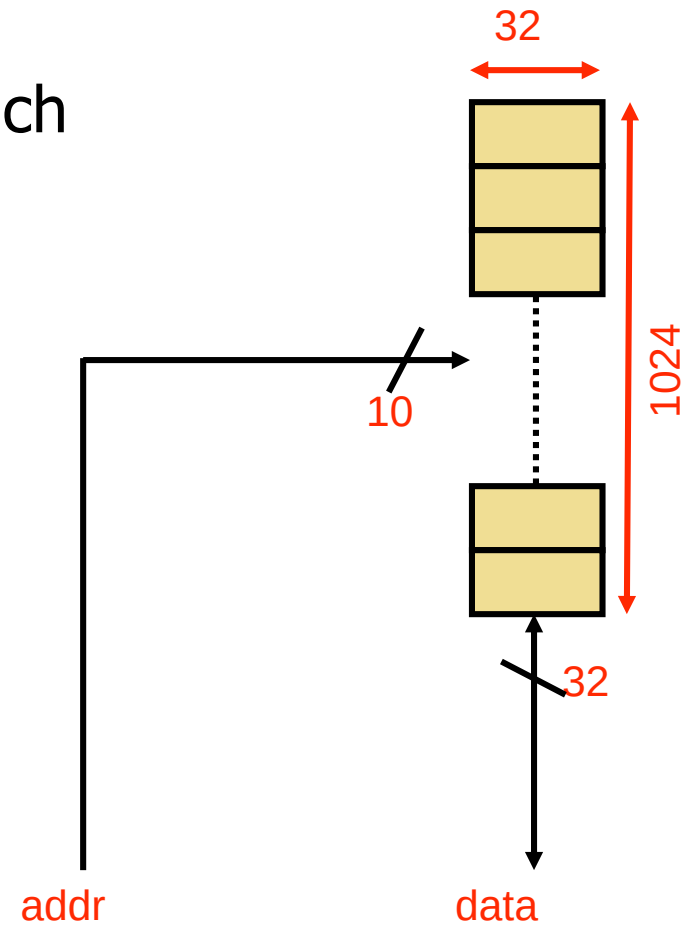
SRAMS -> Caches



- Use SRAMs to make caches
  - Hold a sub-set of memory
- Reading:
  - Input: Address to read (32 or 64 bits)
  - Output:
    - Hit? 1-bit: was it there?
    - Data: if there, requested value

Step 1: Data Basics

- 32-bit addresses
  - 4 Byte words only (to start)
- Start with **blocks** that are 1 word each
  - 4KB, organized as 1K 4B blocks
  - Block: basic unit of data in cache
- Physical cache implementation
  - 1K (1024) by 4B (32) **SRAM**
  - Called **data array**
  - 10-bit address input
  - 32-bit data input/output

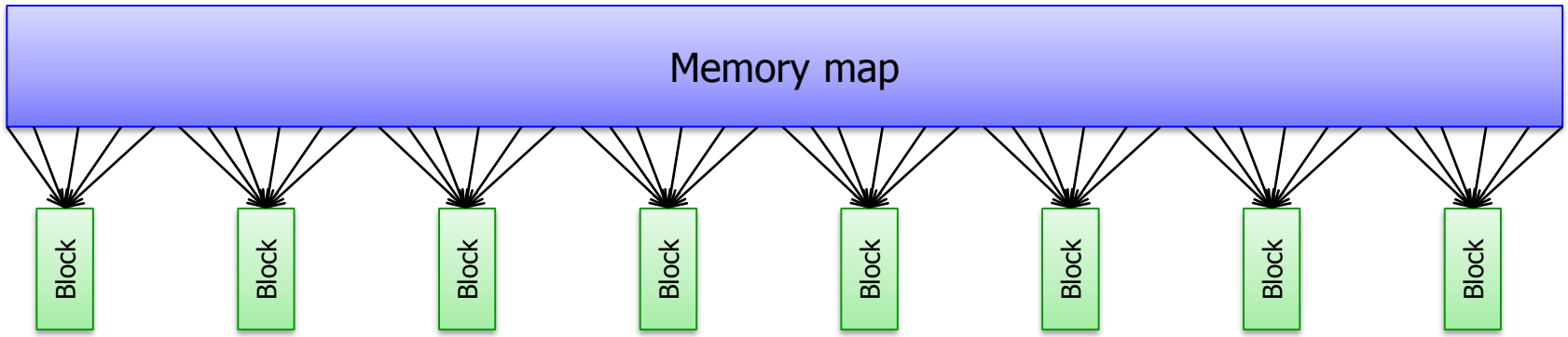


Which bits to use for index?

- Can skip the lowest $\log_2$(block_size) bits: those tell us which byte in the block we're looking for.
- Of the remaining bits, do we pick the lowest ones or the highest ones?

- If we pick **highest** bits for index: 

  - Two addresses that are numerically close will both map to the same block



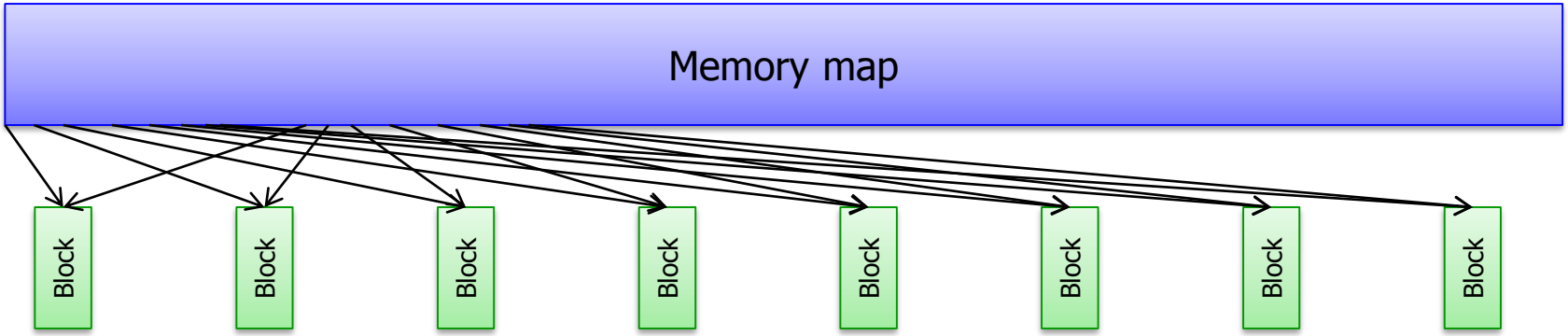
  - Neighbors in memory are likely to collide; fight over the same block
  - Opposite of what we want – this penalizes spatial locality
  - Bad!

Which bits to use for index?

- Can skip the lowest $\log_2$(block_size) bits: those tell us which byte in the block we're looking for.
- Of the remaining bits, do we pick the lowest ones or the highest ones?

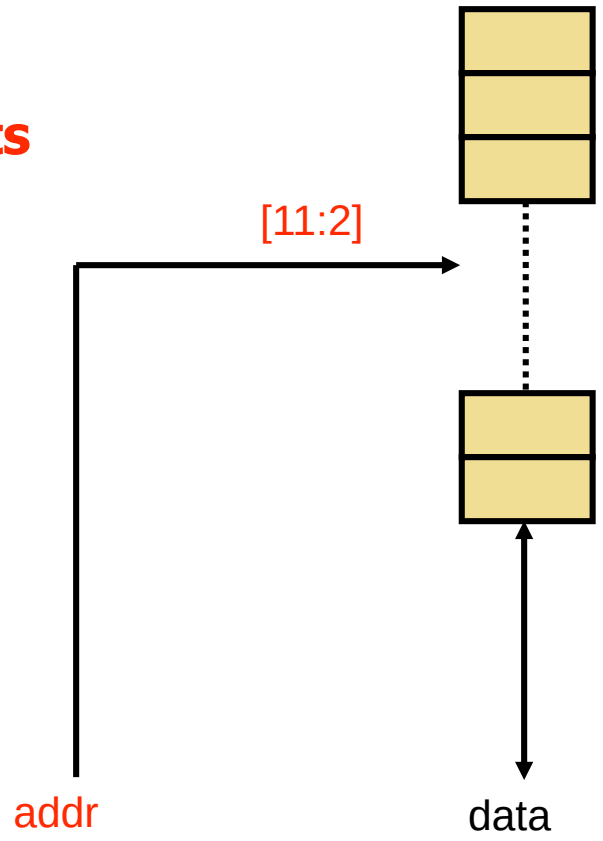
- If we pick **lowest** bits for index:

  - Two addresses that are numerically close will both map to the same block



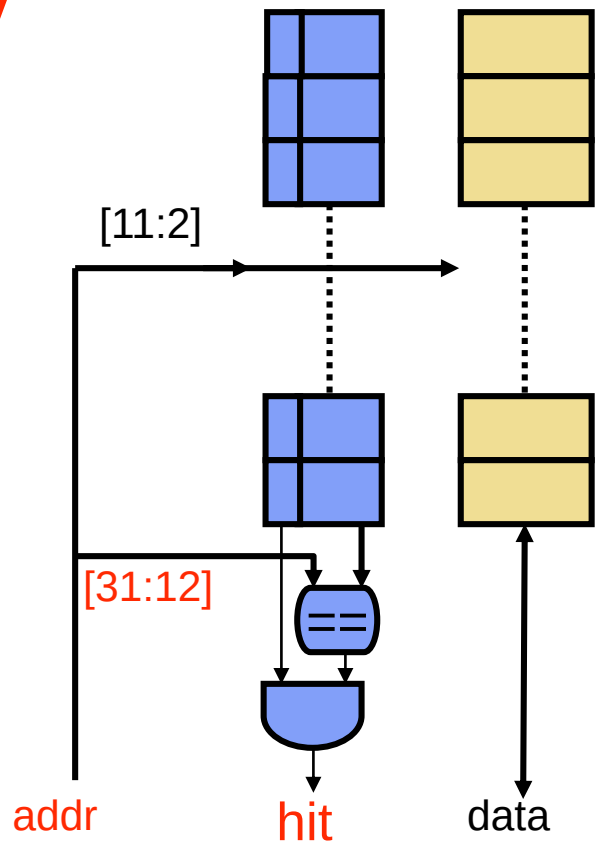
  - Neighbors in memory get neighboring blocks
  - Spatial locality leads to broad use of cache capacity
  - Good!

- Q: which 10 of the 32 address bits to use?
- A: bits [11:2]
  - 2 LS bits [1:0] are the **offset bits**
    - Locate byte within word
    - Don't need these to locate word
  - Next 10 LS bits [11:2] are the **index bits**
    - These locate the word



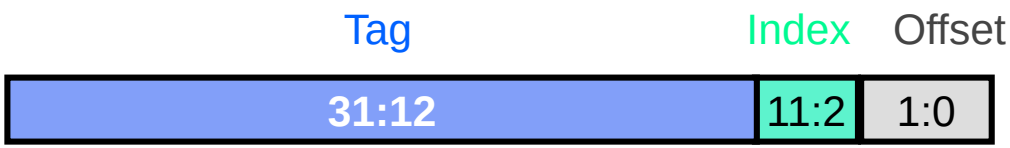
Knowing that You Found It

- Hold a subset of memory
  - How do we know if we have what we need?
  - 2^{20} different addresses map to one particular block
- Build separate and parallel **tag array**
  - 1K by 21-bit SRAM
  - 20-bit (next slide) **tag** + 1 **valid bit**
- Lookup algorithm
  - Read tag indicated by index bits
  - (Tag matches & valid bit set)
    ? Hit → data is good
    : Miss → data is garbage, wait…



Cache Use of Addresses

- Split address into three parts:
  - Offset: least-significant $\log_2$(block-size)
  - Index: next $\log_2$(number-of-sets)
  - Tag: everything else

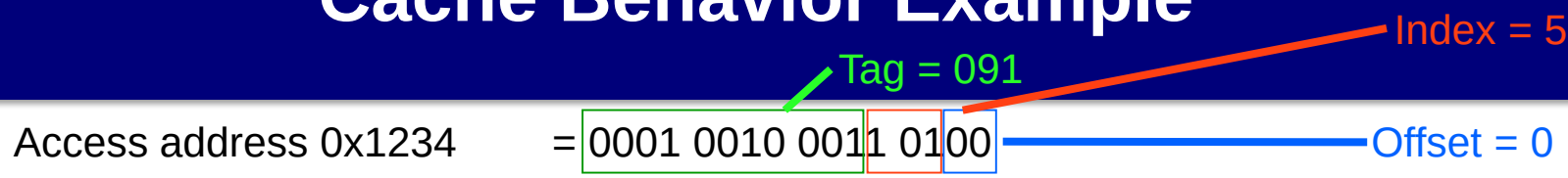


Cache Behavior Example

Cache starts empty (valid = 0). 8 sets, 16 bit address for example

Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	0	000	00 00 00 00
6	0	000	00 00 00 00
7	0	000	00 00 00 00

Cache Behavior Example

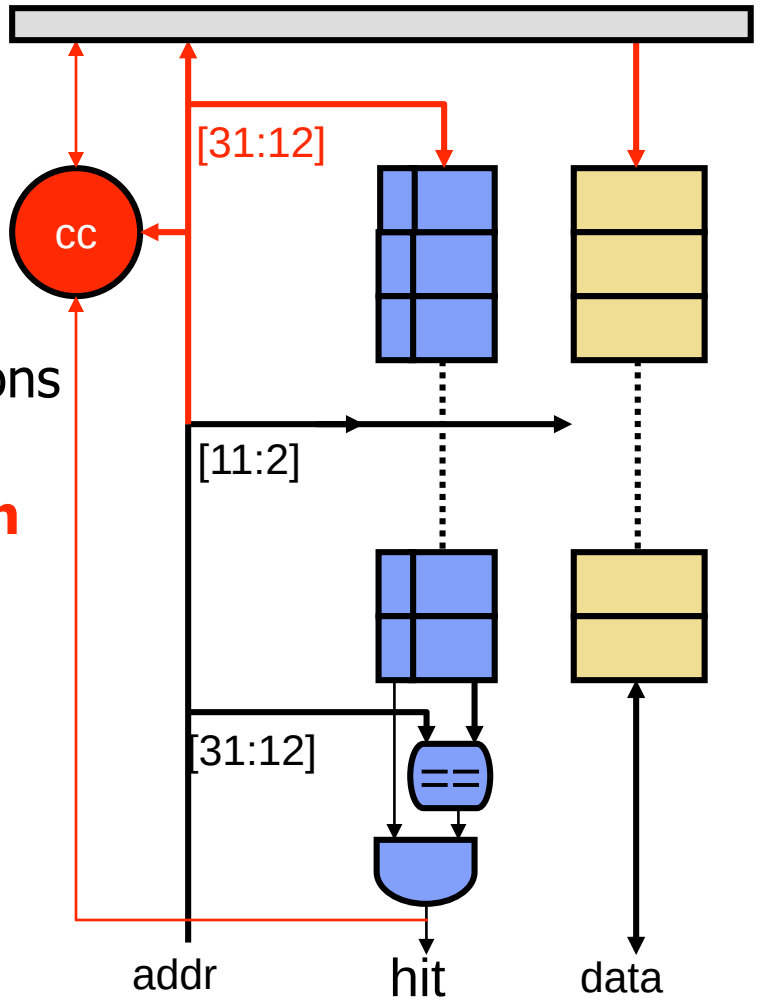


Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	0	000	00 00 00 00
6	0	000	00 00 00 00
7	0	000	00 00 00 00

Not valid: miss

Handling a Cache Miss

- What if requested word isn't in the cache?
  - How does it get in there?
- **Cache controller**: FSM
  - Remembers miss address
  - Asks next level of memory
  - Waits for response
  - Writes data/tag into proper locations
  - All of this happens on the **fill path**
  - Sometimes called **backside**



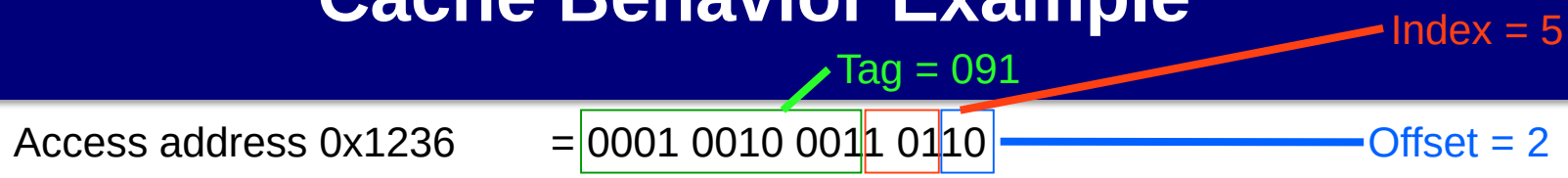
Cache Behavior Example

Access address 0x1234 = 0001 0010 0011 0100

Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	091	0F 1E 39 EC
6	0	000	00 00 00 00
7	0	000	00 00 00 00

```
lb: 00 00 00 EC
lh: 00 00 39 EC
lw: 0F 1E 39 EC
```

Cache Behavior Example

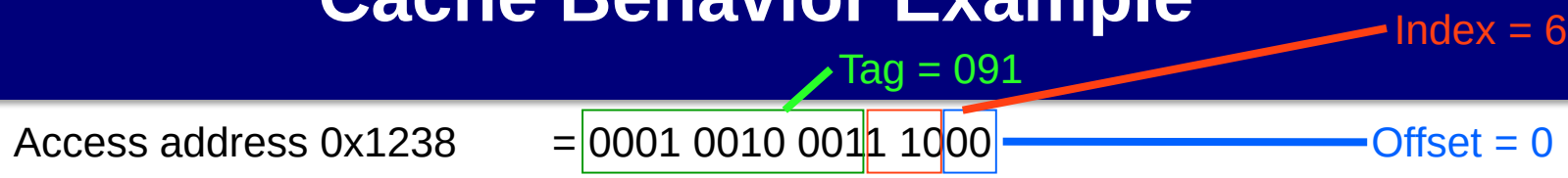


Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	091	0F 1E 39 EC
6	0	000	00 00 00 00
7	0	000	00 00 00 00

Valid && Tag match -> hit

```
lb: 00 00 00 1E
lh: 00 00 0F 1E
lw: (unaligned)
```

Cache Behavior Example



Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	091	0F 1E 39 EC
6	0	000	00 00 00 00
7	0	000	00 00 00 00

Not valid: miss

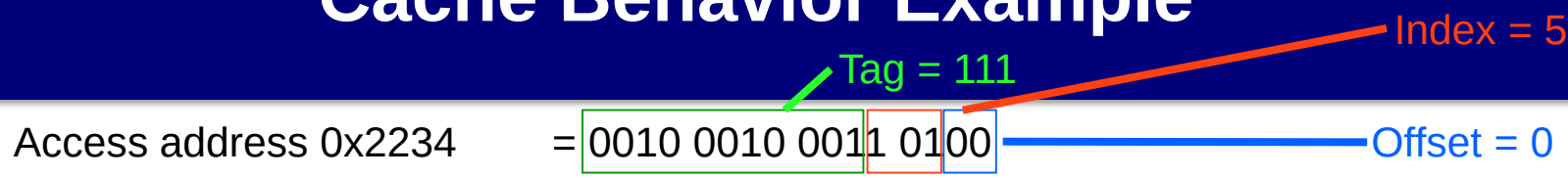
Cache Behavior Example

Access address 0x1238 = 0001 0010 0011 1000

Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	091	0F 1E 39 EC
6	**1**	**091**	3C 99 11 12
7	0	000	00 00 00 00

Make request to next level...
wait for it....

Cache Behavior Example



Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	091	0F 1E 39 EC
6	1	091	3C 99 11 12
7	0	000	00 00 00 00

Valid, but tag does not match: miss

Cache Behavior Example

Access address 0x2234 = 0010 0010 0011 0100

Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	**111**	0F 1E 39 EC
6	1	091	3C 99 11 12
7	0	000	00 00 00 00

Make request to next level...
wait for it....

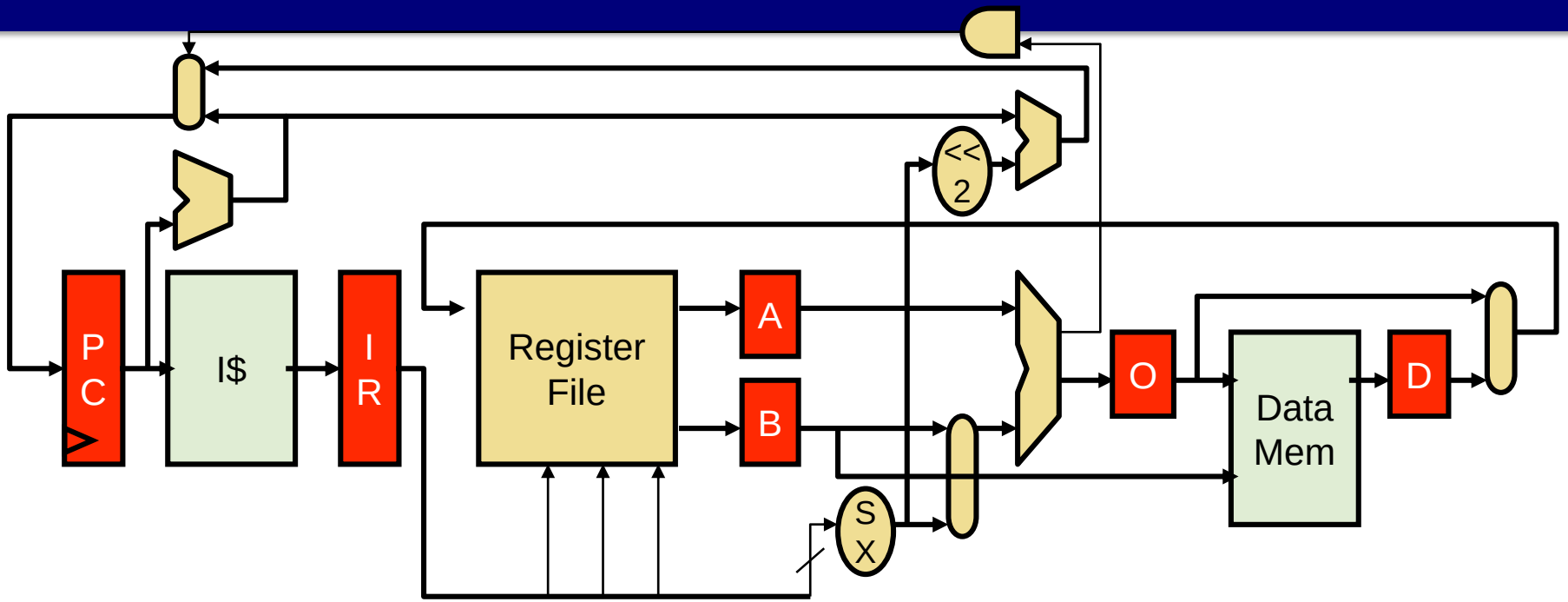
Cache Behavior Example

Access address 0x2234 = 0010 0010 0011 0100

Set #	Valid	Tag	Data
0	0	000	00 00 00 00
1	0	000	00 00 00 00
2	0	000	00 00 00 00
3	0	000	00 00 00 00
4	0	000	00 00 00 00
5	1	111	01 CF D0 87
6	1	091	3C 99 11 12
7	0	000	00 00 00 00

Note that now, 0x1234 is gone replaced by 0x2234

Cache Misses and CPI



- I$ and D$ misses stall datapath (multi-cycle or pipeline)
  - Increase CPI
  - Cache hits built into "base" CPI
    - E.g., Loads = 5 cycles in multi-cycle includes t_{hit}
    - Some loads may take more cycles...
      - **Need to know latency of "average" load (t_{avg})**

Measuring Cache Performance

- Ultimate metric is t_{avg}
  - Cache capacity roughly determines t_{hit}
  - Lower-level memory structures determine t_{miss}
  - Measure $\%_{miss}$
    - Hardware performance counters (since Pentium)
    - Performance Simulator
    - Paper simulation (like we just did)
      - Only works for small caches
      - Small number of requests (would not do for 1M accesses)

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C				invalid	0	0	1	
E								
8								
3								
8								
0								
8								
4								
6								

- Tag, index, offset?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C				invalid	0	0	1	
E								
8								
3								
8								
0								
8								
4								
6								

- Tag: 1 bit, Index: 2 bits, Offset: 1 bit

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C				invalid	0	0	1	
E								
8								
3								
8								
0								
8								
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E				invalid	0	1	1	
8								
3								
8								
0								
8								
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8				invalid	0	1	1	
3								
8								
0								
8								
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3				1	0	1	1	
8								
0								
8								
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3	0	1	1	1	0	1	1	Hit
8				1	0	1	1	
0								
8								
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3	0	1	1	1	0	1	1	Hit
8	1	0	0	1	0	1	1	Hit
0				1	0	1	1	
8								
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3	0	1	1	1	0	1	1	Hit
8	1	0	0	1	0	1	1	Hit
0	0	0	0	1	0	1	1	Miss
8				0	0	1	1	
4								
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3	0	1	1	1	0	1	1	Hit
8	1	0	0	1	0	1	1	Hit
0	0	0	0	1	0	1	1	Miss
8	1	0	0	0	0	1	1	Miss
4				1	0	1	1	
6								

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3	0	1	1	1	0	1	1	Hit
8	1	0	0	1	0	1	1	Hit
0	0	0	0	1	0	1	1	Miss
8	1	0	0	0	0	1	1	Miss
4	0	2	0	1	0	1	1	Miss
6				1	0	0	1	

- What happens for each request?

Cache Miss Paper Simulation

- 4-bit address, 8B cache, 2B blocks -> 4 sets

Address	Tag	Index	Offset	Set 0	Set 1	Set 2	Set3	Result
C	1	2	0	invalid	0	0	1	Miss
E	1	3	0	invalid	0	1	1	Hit
8	1	0	0	invalid	0	1	1	Miss
3	0	1	1	1	0	1	1	Hit
8	1	0	0	1	0	1	1	Hit
0	0	0	0	1	0	1	1	Miss
8	1	0	0	0	0	1	1	Miss
4	0	2	0	1	0	1	1	Miss
6	0	3	0	1	0	0	1	Miss

- What happens for each request?

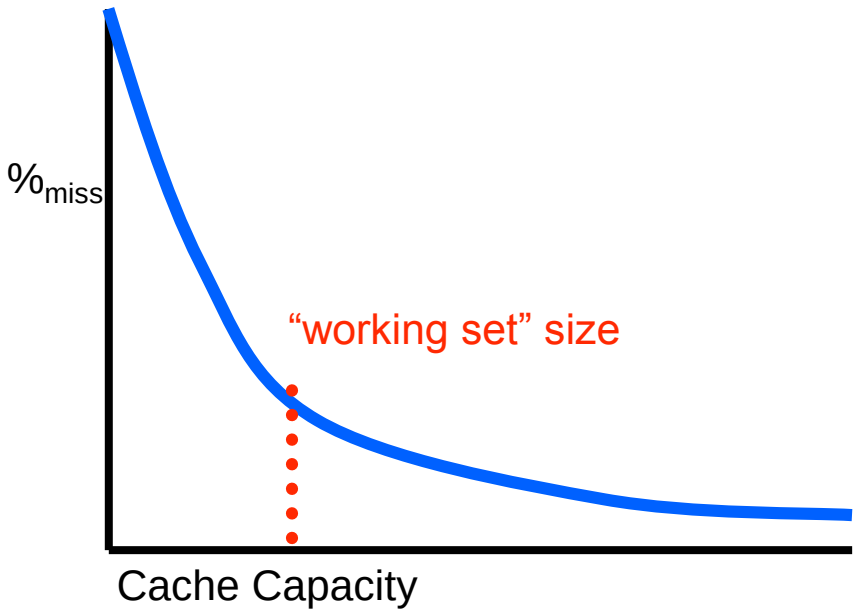
Cache Miss Paper Simulation

- %miss: 6 / 9 = 66%
  - Not good...
  - How could we improve it?

Result
Miss
Hit
Miss
Hit
Hit
Miss
Miss
Miss
Miss

Capacity and Performance

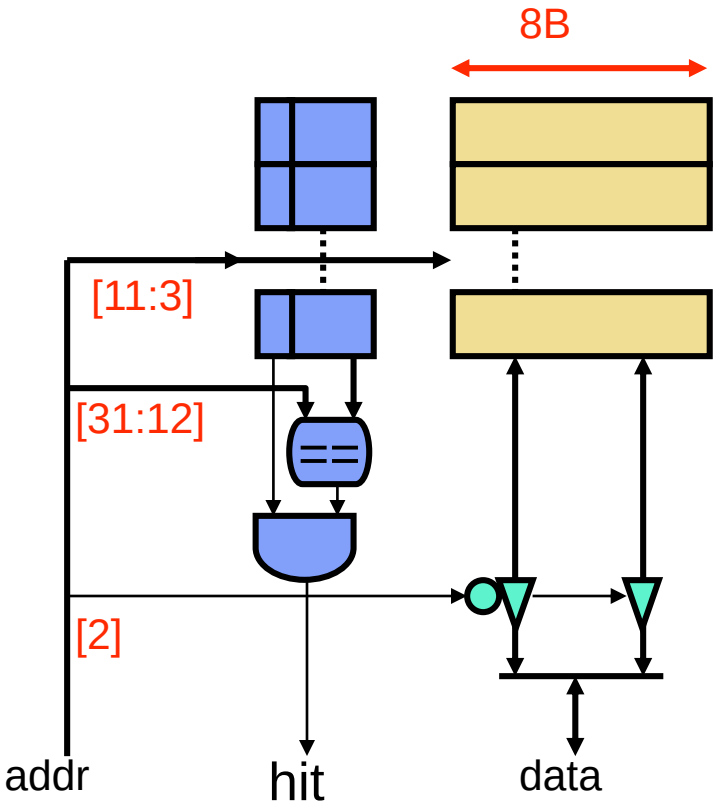
- Simplest way to reduce $\%_{miss}$: increase capacity
  - + Miss rate decreases monotonically
    - **"Working set"**: insns/data program is actively using
  - – t_{hit} increases
  - t_{avg} ?



- Given capacity, manipulate $\%_{miss}$ by changing **organization**

Block Size

- One possible re-organization: increase **block size**
  - \+ Exploit **spatial locality**
  - – Caveat: increase conflicts too
  - – Increases $t_{hit:}$ need word select mux
    - By a little, not too bad
  - \+ Reduce tag overhead



Tag Overhead

- "4KB cache" means cache holds 4KB of data (**capacity**)
  - Tag storage is considered overhead
    - Valid bit usually not counted
  - Tag overhead = tag size / data size

- 4KB cache with 4B blocks?
  - 4B blocks → 2-bit offset
  - 4KB cache / 4B blocks → 1024 blocks → 10-bit index
  - 32-bit address – 2-bit offset – 10-bit index = 20-bit tag
  - 20-bit tag / 32-bit block = **63% overhead**

Block Size and Tag Overhead

- 4KB cache with 1024 4B blocks?
  - 4B blocks → 2-bit offset, 1024 frames → 10-bit index
  - 32-bit address – 2-bit offset – 10-bit index = 20-bit tag
  - 20-bit tag / 32-bit block = 63% overhead

- 4KB cache with 512 8B blocks
  - 8B blocks → 3-bit offset, 512 frames → 9-bit index
  - 32-bit address – 3-bit offset – 9-bit index = 20-bit tag
  - **20-bit tag** / **64-bit block** = **32% overhead**
  - Notice: tag size is same, but data size is twice as big

- A realistic example: 64KB cache with 64B blocks
  - 16-bit tag / 512-bit block = **~ 2% overhead**

Cache Miss Paper Simulation

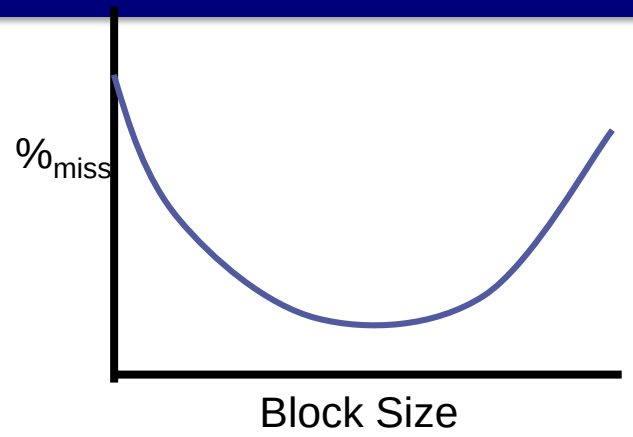
- 8B cache, 4B blocks -> 2 sets

Address	Tag	Index	Offset	Set 0	Set 1	Result
C	1	1	0	invalid	0	Miss
E	1	1	2	invalid	1	Hit
8	1	0	0	invalid	1	Miss
3	0	0	3	1	1	**Miss**
8	1	0	0	0	1	**Miss**
0	0	0	0	1	1	Miss
8	1	0	0	0	1	Miss
4	0	1	0	1	1	Miss
6	0	1	2	1	0	**Hit**

- 8,3: new conflicts (fewer sets)
- 4,6: spatial locality

Block Size and Miss Rate Redux

+ **Spatial prefetching**
  - For blocks with adjacent addresses
  - Turns miss/miss pairs into miss/hit pairs
  - Example: `4, 6`



– **Conflicts**
  - For blocks with non-adjacent addresses (but sharing the same index)
  - Turns hits into misses by disallowing simultaneous residence
  - Example: `8, 3`

- Both effects always present to some degree
  - Spatial prefetching dominates initially (until 64–128B)
  - Conflicts dominate afterwards
  - Optimal block size is 32–256B (varies across programs)
  - Typical: 64B

Block Size and Miss Penalty

- Does increasing block size increase t_{miss}?
  - Don't larger blocks take longer to read, transfer, and fill?
  - They do, but…
- t_{miss} of an isolated miss is not affected
  - **Critical Word First / Early Restart (CRF/ER)**
  - Requested word fetched first, pipeline restarts immediately
  - Remaining words in block transferred/filled in the background
- t_{miss}'es of a cluster of misses will suffer
  - Reads/transfers/fills of two misses cannot be overlapped
  - Latencies start to pile up
  - This is technically a bandwidth problem (more later)

Cache Miss Paper Simulation

- 8B cache, 4B blocks -> 2 sets

Address	Tag	Index	Offset	Set 0	Set 1	Result
C	1	1	0	invalid	0	Miss
E	1	1	2	invalid	1	Hit
8	1	0	0	invalid	1	Miss
3	0	0	3	1	1	**Miss**
8	1	0	0	0	1	**Miss**
0	0	0	0	1	1	Miss
8	1	0	0	0	1	Miss
4	0	1	0	1	1	Miss
6	0	1	2	1	0	**Hit**

- 8 (1000) and 3 (0011): same set
- Can we do anything about this? Next lecture…