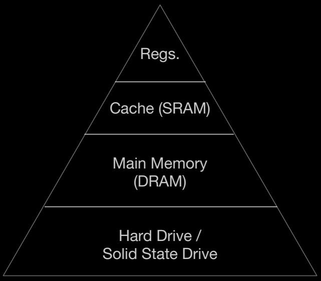
Review of Caching — Store a Subset of Commonly Used Data from Memory

It is possible to create small amounts of fast memory (SRAM)

Create a small working space of SRAM

Only have to read from slow memory (DRAM) when transferring data to our cache (SRAM)

Anytime we need data, go down a level in our memory hierarchy

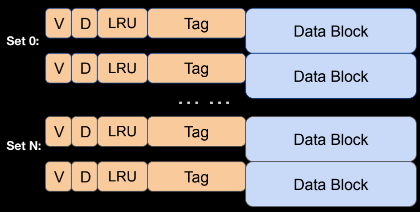


From memory to cache visual

Note: sets are generalization of lines - will be discussed in detail in lecture next week

Address breakdown: → Cache:

Tag Bits	Set Bits	Block Offset Bits
• Will use this value to check if an address is in the cache by checking if the value of these tag bits matches any existing tag in the cache ◦ Match = Hit ◦ No Match = Miss	• Will use this value to narrow our search to the set which is indicated by the set bits- we check for a cache hit or miss within this set	• Will use this value to index the data block to retrieve the actual value contained at this memory address



Note: the graphic on this page is supposed to give a high level overview of how we go from an address to the cache- the structure of the cache (number of ways per set, number of sets, etc.) and memory address break down will depend on the parameters given to you for a cache implementation

Example: The Size and Number of Blocks

Sample 16-bit Address	1	0	0	1	1	1	0	1	0	1	1	0	0	1	0	1

For this problem, our byte-addressable processor uses 16-bit addresses with 4 bits for the block offset and the remaining 12 bits for the tag.

How big are each of our blocks in the cache?

*4 block offset bits → 2^4 B per block = **16B per block***

How many blocks can our cache fit if it has 128B of data storage?

*128B cache / 16B per block = **8 blocks***

Example: Where Do the Blocks End Up?

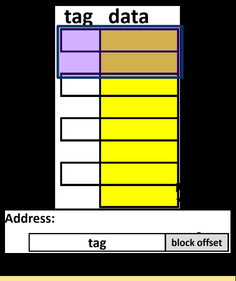
Imagine an 8-bit, byte-addressable architecture.

We are testing 3 different 8B caches with the following specification:

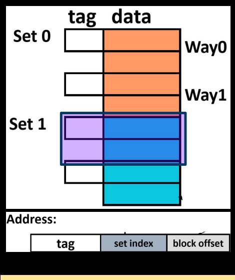
Cache #1:	Fully-Associative,	2B blocks
Cache #2:	2-way Set-Associative,	2B blocks
Cache #3:	Direct-Mapped,	2B blocks

Where does the read to `0b1001 0111` end up in each of our caches? Assume the cache is initially empty.

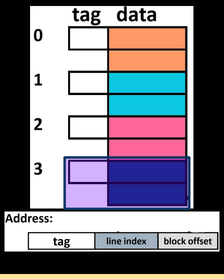
Example: Where Do the Blocks End Up?



Fully-Associative Cache

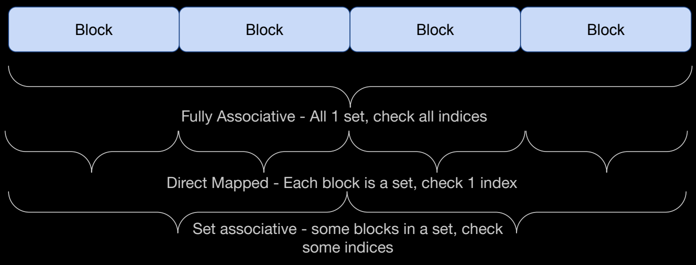


Set-Associative Cache



Direct-Mapped Cache

Project 4: Partition the Blocks Array into Sets

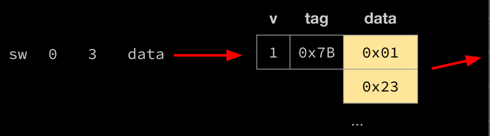

Block
Block
Block
Block
Fully Associative - All 1 set, check all indices
Direct Mapped - Each block is a set, check 1 index
Set associative - some blocks in a set, check some indices

Recommendation: Use multiplication/left shift with your cache parameters (Blocks per set) and set bits to figure out start and end indices for each set.

Write-Through Caches Ensure Memory Coherence

In a write-through cache, any write is also sent off to memory

This ensures memory always holds the correct values, helpful if multiple processes are reading from memory

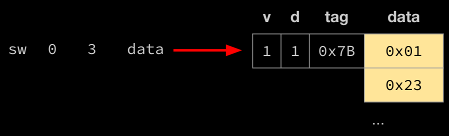


address	data
0xF5	0xB6
0xF6	0x01
0xF7	0x23
0xF8	0xDE
0xF9	0xAD

Write-Back Caches Reduce Memory Writes

In a write-back cache, we only write to memory when evicting dirty cache lines

This minimizes the number of writes we perform (but maybe not the number of *bytes written*)



address	data
0xF5	0xB6
0xF6	0x56
0xF7	0x78
0xF8	0xDE
0xF9	0xAD

Review: Writes

Store w/ No Allocate		Write-Back	Write-Through
	Hit?	Write Cache	Write to Cache **+ Memory**
	Miss?	**Write to Memory**	**Write to Memory**
	Replace block?	If evicted block is dirty, write to Memory	Do Nothing

Store w/ Allocate		Write-Back	Write-Through
	Hit?	Write Cache	Write to Cache **+ Memory**
	Miss?	**Read from Memory to Cache, Allocate to LRU block Write to Cache**	**Read from Memory to Cache, Allocate to LRU block Write to Cache + Memory**
	Replace block?	If evicted block is dirty, write to Memory	Do Nothing

Important Formulas

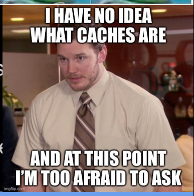
cheat sheet!

- **Block Offset Bits** = $\log_2$(Block Size)
- **Set Index Bits** = $\log_2$(#Sets)
- **Tag Bits** = Address Size - (Set Index Bits + Block Offset Bits)
- **Address Size** = $\log_2$(Size of Memory)
- **Cache Size** = #Cache Blocks * Block Size
- **#Cache Blocks** = #Cache Lines = #Sets * #BlocksPerSet
- **#BlocksPerSet** = #Ways = Associativity
- **LRU Bits** = $\log_2$(#BlocksPerSet)

Recommendation: Understand how to derive the formulas
Don't just memorize!

Don't be like Andy - ask any questions about caches since they will be everywhere for the rest of the semester.

Right when you thought you could move on from caches and you realize they're everywhere


I HAVE NO IDEA WHAT CACHES ARE
AND AT THIS POINT I'M TOO AFRAID TO ASK