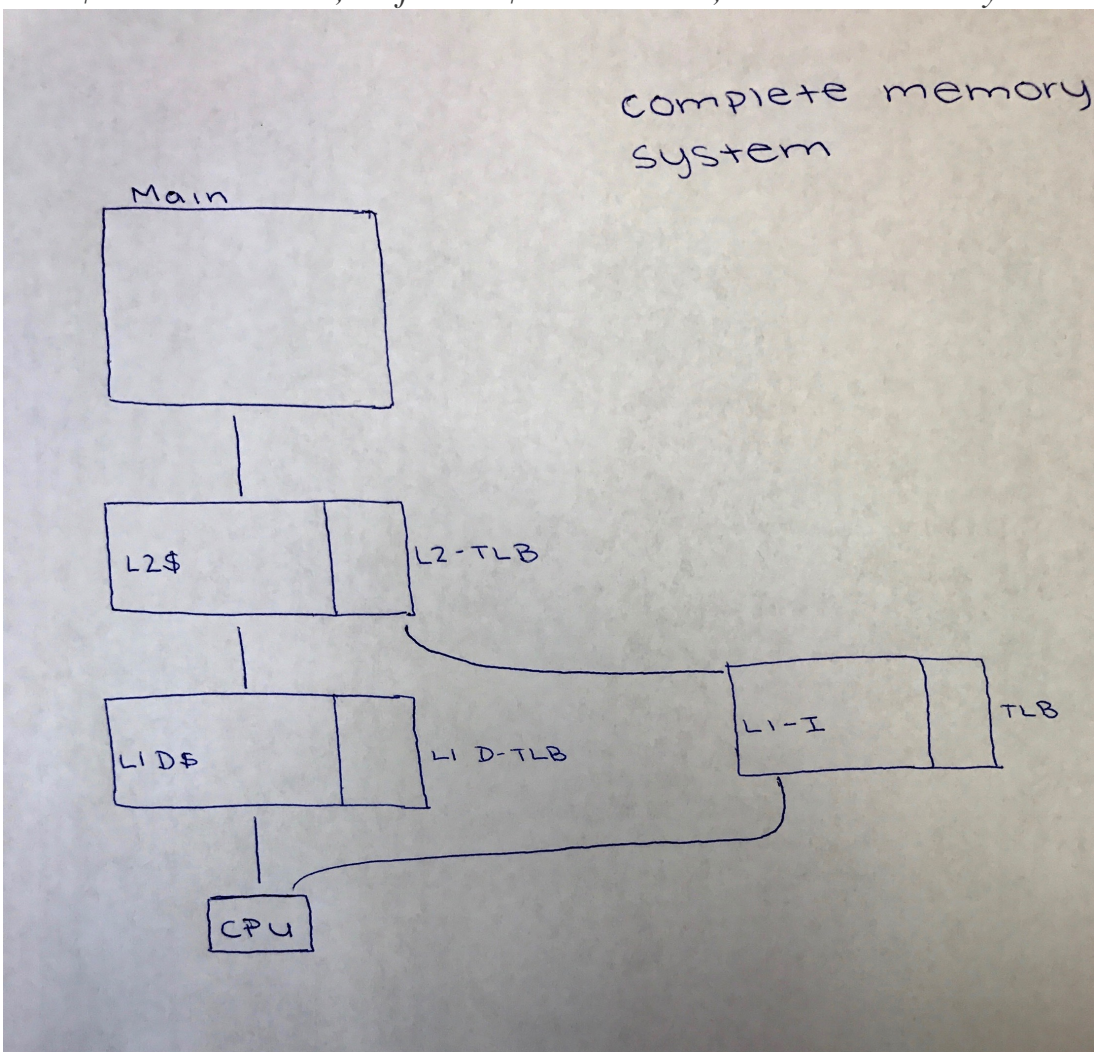
Mariam Sulakian

NUMBER 1:

Draw a complete memory system that includes the following: L1 I$ and L1 I-TLB, L1 D$ and L1 D- TLB, unified L2$ and L2 TLB, and main memory.



NUMBER 2:

Here are many of the possible outcomes for a given load. For each one, explain how it can occur and what happens in the given situation. Why is the shaded situation impossible?

situation	L1 D-TLB	L1 D$	L2 TLB	L2$	mem
1	hit	hit			
2	hit	miss		hit	
3	miss	hit			
4	miss	miss	hit	hit	
5	miss	miss	miss	hit	
6	miss	miss	miss	miss	hit
7	miss	miss	hit	miss	hit
8	miss	miss	hit	miss	miss
9	miss	miss	miss	miss	miss

Situation 1: Hit in L1 TLB and data is in L1.
Situation 2: Hit in L1 TLB but data is not in L1 cache so miss. Data in L2 cache.
Situation 3: Miss in L1 TLB but hit in cache.
Situation 4: Data is not in L1, most recent version kicked out from L1 or never written to L1 in first place.
Situation 5: Miss in L1 completely, Miss in L2 cache and translation
Situation 6: Not in cache. Miss everywhere except in memory.
Situation 7: You have translation from L2 TLB but you check L2 and data is still not there so miss. Hit in memory.
Situation 8: If data is not in main memory then the data is not there. Any translation (L2 TLB) would be invalid.
Situation 9: Data been paged out of main memory and sitting on hard drive. Main would have to ask hard drive for the data.

Translations only exist if data is in main memory. Data can be in L1, L2, or mem. Anything in L1 will usually also be in L2.

NUMBER 3:
What are the trade-offs between handling a TLB miss in software versus handling a TLB miss with hardware? Consider issues like latency, hardware cost, backward compatibility, etc.

Latency: TLB miss in software is slower in software than hardware. You have a FSM in hardware vs software you would have to iterate over some data structure in memory to find.

Hardware cost: TLB miss in hardware would be more expensive.
Backward compatibility: TLB miss in hardware harder more expensive.

NUMBER 4:
Let's say you want to implement a virtual/physical cache (virtually indexed and physically tagged). Assume: 32-bit architecture, 16B cache blocks, 32KB pages. The cache is 128KB. How set-associative must the cache be to permit the use of virtual indexing with physical tagging?

Set-associativity = 4

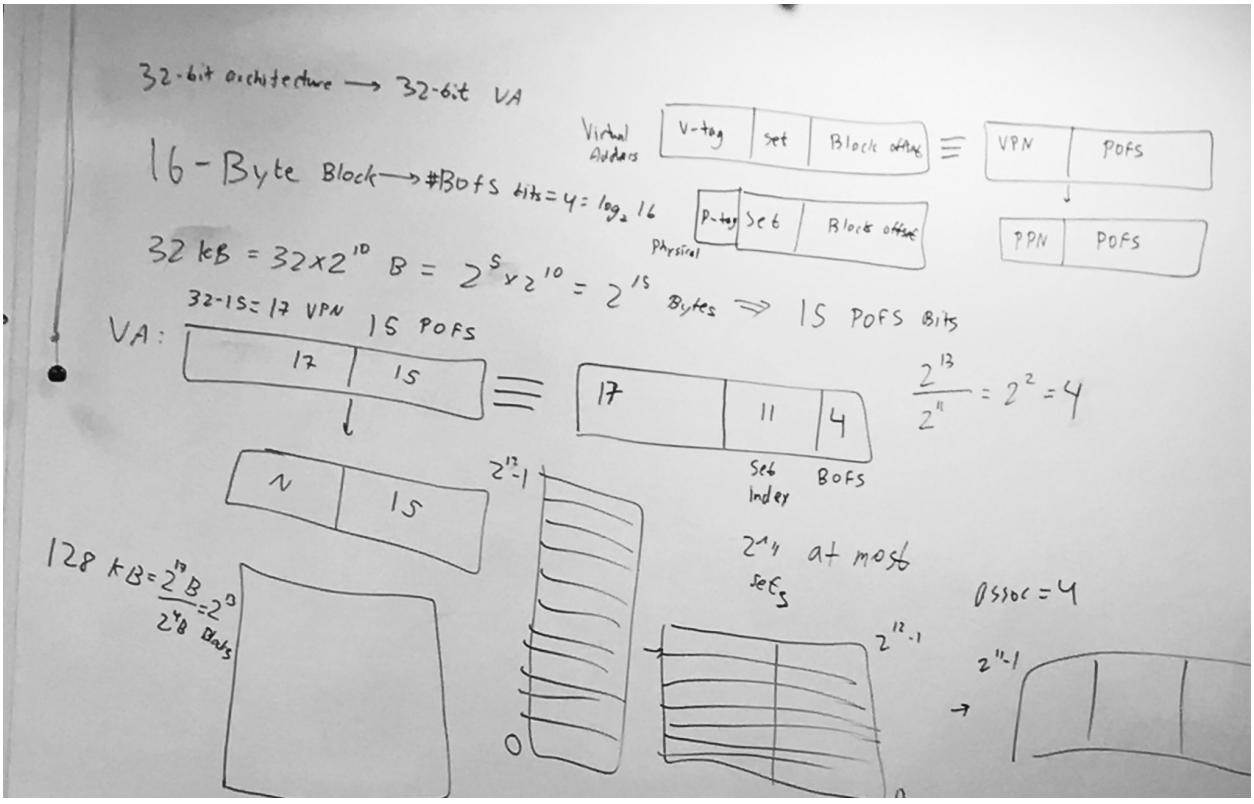
32-bit architecture → 32 bit virtual address
16B block → number of block offset bits is 4 = log(2) 16
32KB page = 32 x 2^10B = 2^5 * 2^10 = 2^15 bytes → 15 page offset bits

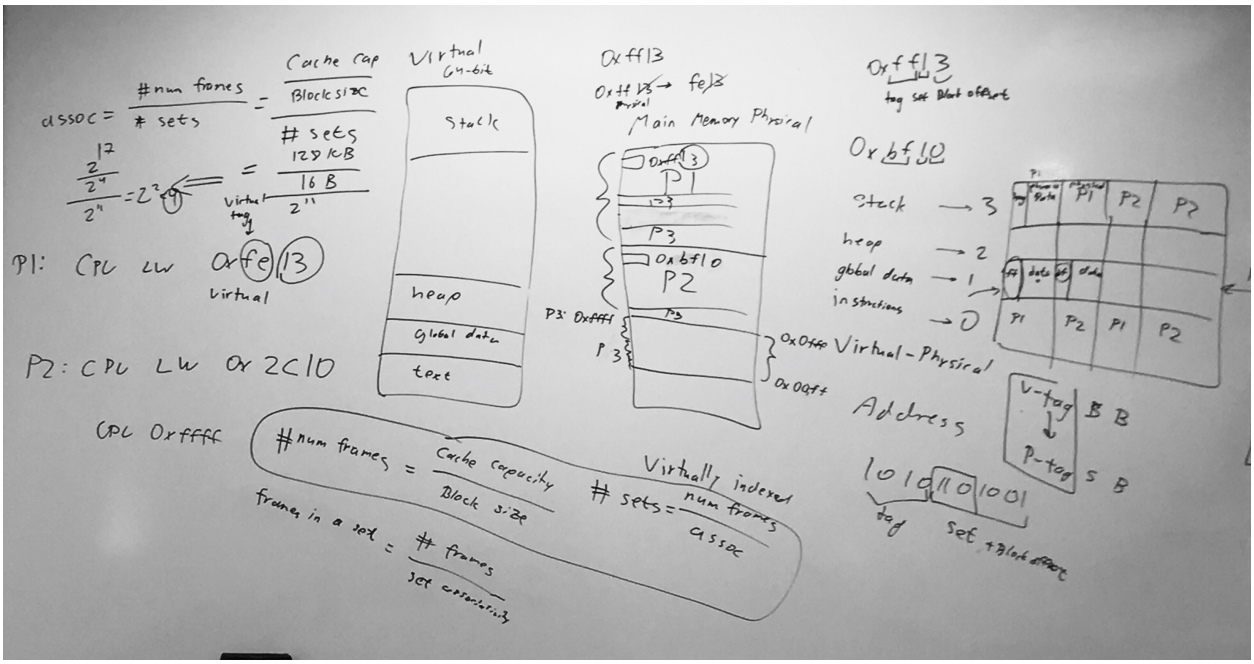
32 – 15 = 17 virtual page number bits
15 page offset bits

We need *at most* 2^11 sets. But we can also have less.
128KB cache = 2^17B
2^17 / 16B blocks = 2^17 / 2^4 = 2^13 blocks in this cache.

128KB / 16B / 2^11 (max sets) = 2^17 / 2^4 / 2^11 = 2^2

num frames = cache capacity / block size
sets = num frames / associativity





16B cache blocks; 16B block means in a block there are 16 bytes and then there are 4 offset bits

Question: how bit does set have to be so that v-tag and p-tag don't collide? more associative → more set bits;

Mariam Sulakian

Perks of virtual/physical cache:

1. Allows processes to share same set in cache.
2. Only thing you have to translate are tags. TLB can thus be smaller.

Virtually indexed then sets are virtual sets and map to space in virtual memory of a program. Virtual/Physical system that allows processors to share sets in same physical caches simultaneously without them bothering one another. Virtual specifies function of each set. Physical allows programs to share spaces in each set.