Conceptual Questions: Why do we cache? What is the end result of our caching, in terms of capability?

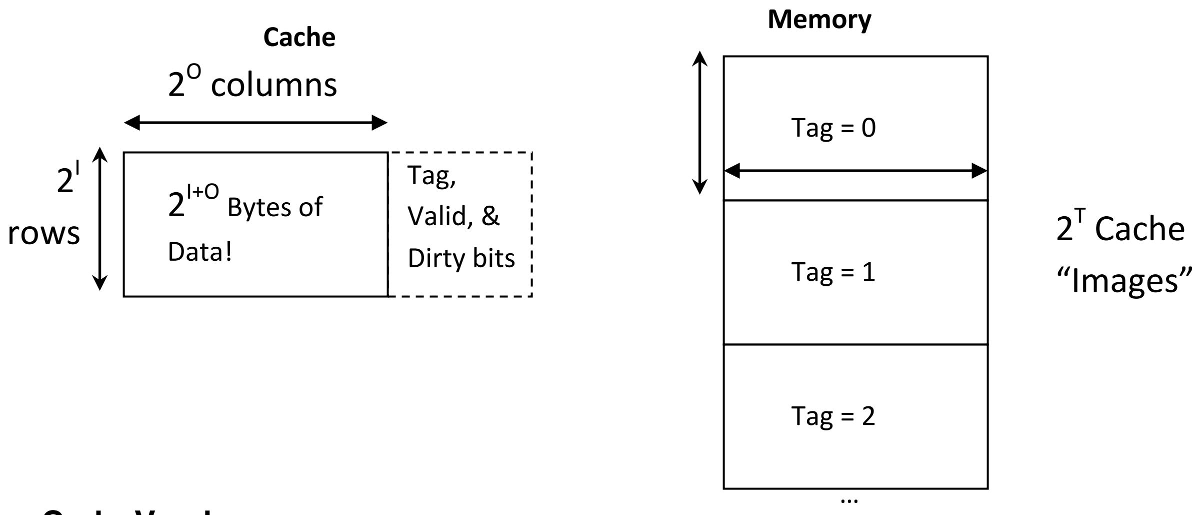
What are temporal and spatial locality? Give high level examples in software of when these occur.

Break up an address:

Tag	**I**ndex	**O**ffset

Offset: "column index", Indexes into a block. (O bits)
Index: "row index," Indexes blocks in the cache. (I bits)
Tag: Where from memory did the block come from? (T bits)

Segmenting the address into TIO implies a geometrical structure (and size) on our cache. Draw memory with that same geometry!



Cache Vocab:

Cache hit – found the right thing in the cache! Booyah!
Cache miss – Nothing in the cache block we checked, so read from memory and write to cache!
Cache miss, block replacement – We found a block, but it had the wrong tag!

1) Fill in the table assuming a direct mapped cache. (B = byte.)

Address Bits	Cache Size	Block Size	Tag Bits	Index Bits	Offset Bits	Bits per Row
16	4KB	4B				
16	16KB	8B				
32	8KB	8B				
32	32KB	16B				
32	64KB		16	12	4	15
32	512KB				5	
64		64B		14		
64	2048			14		1068

2) Assume 16 words of memory and an 8 word direct-mapped cache with 2-word blocks (that starts empty). Classify each of the following WORD memory accesses as hit (H), miss (M), or miss with replacement (R).

a. 4
b. 5
c. 2
d. 6
e. 1
f. 10
g. 7
h. 2

3) You know you have 1 MiB of memory (maxed out for processor address size) and a 16 KiB cache (data size only, not counting extra bits) with 1 KiB blocks.

```
#define NUM_INTS 8192
int A[NUM_INTS]; // lives at 0x100000
int i, total = 0;
for (i = 0; i < NUM_INTS; i += 128) A[i] = i; // Line 1
for (i = 0; i < NUM_INTS; i += 128) total += A[i]; // Line 2
```

a) What is the T:I:O breakup for the cache (assuming byte addressing)?
b) Calculate the hit percentage for the cache for the line marked “Line 1”.
c) Calculate the hit percentage for the cache for the line marked “Line 2”.
d) How could you optimize the computation?