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

To make memory seem faster.

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

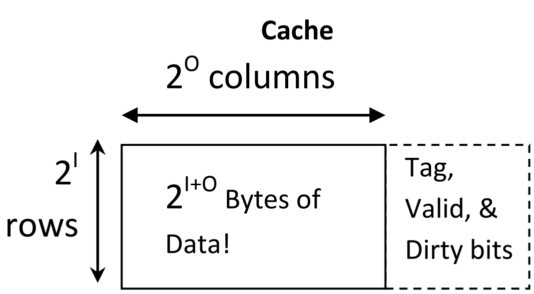
Temporal locality — if a value is accessed; it is likely to be accessed again soon
Examples: loop indices, accumulators, local variables in functions
Spatial locality — if a value is accessed; values near to it are likely to be accessed again soon
Examples: iterating through an array

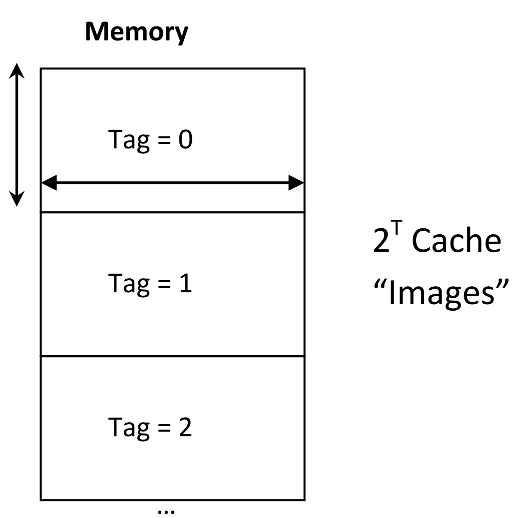
Break up an address:

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	4	10	2	37
16	16KB	8B	2	11	3	67
32	8KB	8B	19	10	3	84
32	32KB	16B	17	11	4	146
32	64KB	16B	16	12	4	15
32	512KB	32B	13	14	5	270
64	1024KB	64B	44	14	6	557
64	2048	128B	43	14	7	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 M
b. 5 H
c. 2 M
d. 6 M
e. 1 M
f. 10 R
g. 7 H
h. 2 R

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)?
6:4:10

b) Calculate the hit percentage for the cache for the line marked “Line 1”.

On each step, we traverse 512 bytes. But there are 1024 bytes in the cache block. So we access each cache block twice, missing on the first and hitting on the second. So the hit rate is 50%

c) Calculate the hit percentage for the cache for the line marked “Line 2”.

The upper half of the array is in cache at this point, so we get the exact same sequence of hits and misses. Therefore the hit rate is again %50

d) How could you optimize the computation?

You could do the second loop in the opposite direction, or you could collapse the two loops into one.