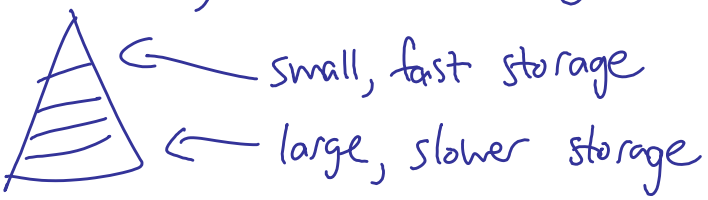
Cache mapping schemes

Recall from last time, have heirarchy of memory and/or storage types:



For simplicity, consider just 2 levels. Call them "upper" and "lower"

The storage is divided into fixed-size blocks. (e.g. might have 1KB or 4KB blocks).

Every time a block on the lower level is accessed (i.e. read or written), a copy of that block is stored somewhere in the upper level. (When access to a block is requested, the upper level is searched first.) — we say it is "cached"

Why do this? Because of locality: after a given byte is accessed, it's likely that other nearby bytes will also be accessed soon - so store them on the upper level where we can get them quickly.

More specifically, there are three types of locality.

spatial: nearby bytes are often accessed together

temporal: a given byte is often accessed repeatedly in a short time

sequential: bytes are often accessed in sequence.

To implement caching, we need to answer two questions:

(A) When copying a block from lower to upper level, where do we put it?

(B) When looking for a copy of a lower level block in the upper level, how do we know if it is there?

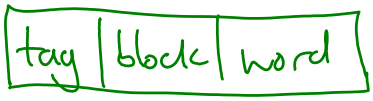
exception: fully associative does uniquely identify blocks

but not uniquely identify; tag is first few bits of block address

We examine 3 possible answers: direct mapped, fully associative and N-way set associative.

In all schemes, the system uses a tag to identify each lower-level block, and the upper level stores a valid bit for each block to tell us if data is valid.

Summary of the 3 strategies:

	where do we put it?	how do we know if it's there?
direct mapped 	in the block given by the block field	check tag in relevant block
fully associative 	anywhere*	special hardware finds the block if it's there
N-way set associative 	anywhere* in the set given by the set field	set field determines the set; special hardware finds tag if it's in that set

* If no space left, need to evict a block from the cache. Use an eviction strategy to choose a victim for eviction. Least recently used (LRU) is one possible eviction

strategy — but LRU is too expensive to implement in practice, so approximate LRU algorithms are used instead.

- See handout for worked examples of each scheme.

- There are two different strategies for handling writes:

  - write through: the change is written in both upper and lower levels.

  - write back: the change is written in upper level only; block is marked dirty. When a dirty block is evicted, it is first copied back to the lower level.

- Modern processors often have 2 separate caches: one for data and one for instructions.

- Note: we can now explain the 2D array paradox from the first lecture in this course!
  Sequential access to memory is faster than staggered access, because sequential has a higher hit rate.