Caching and TLBs

0 Main Points:

Cache concept, in general and as applied to translation
Ways of organizing caches -- associativity
Problems with caching

1 Cache concept

Cache: copy that can be accessed more quickly than original.

Idea is: make frequent case efficient, infrequent path doesn't matter as much. Caching is a fundamental concept used in lots of places in computer systems. It underlies many of the techniques that are used today to make computers go fast: can cache translations, memory locations, pages, file blocks, file names, network routes, authorizations for security systems, etc.

2 Generic Issues in Caching

Cache hit: item is in the cache
Cache miss: item is not in the cache, have to do full operation

Effective access time =
P(hit) * cost of hit + P(miss) * cost of miss

1. How do you find whether item is in the cache (whether there is a cache hit)?

2. If it is not in cache (cache miss), how do you choose what to replace from cache to make room?

3. Consistency -- how do you keep cache copy consistent with real version?

3 Memory Hierarchy

Two principles:

1. The smaller amount of memory needed, the faster that memory can be accessed.

2. The larger amount of memory, the cheaper per byte.

Thus, put frequently accessed stuff in small, fast, expensive memory; use large, slow, cheap memory for everything else.

	Latency	Size	Cost
registers	5ns	32-128 bytes	on chip
on-chip cache	10ns	4KB	on chip
off-chip cache	25ns	64KB	$5000/MB
main memory	200ns	64MB	$50/MB
disk	10ms (10M ns)	2 GB	$0.50/MB
robotic tape	10s (10B ns)	1 TB	$0.01/MB

Use caching at each level, to provide illusion of a terabyte, with register access times.

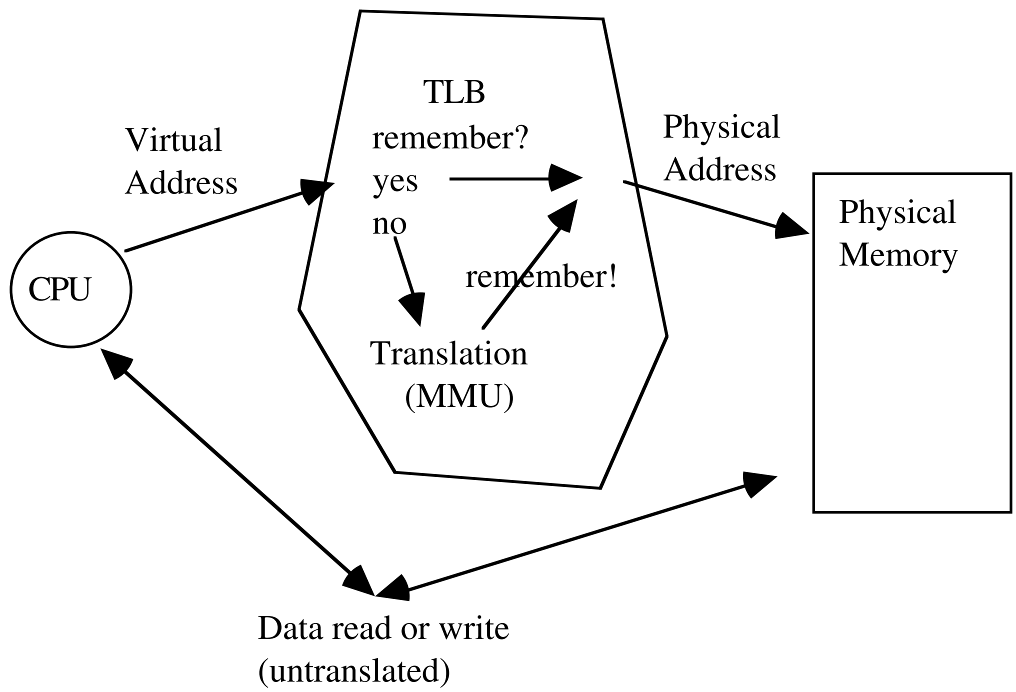
Works because programs aren't random. Exploit **locality**: that computers behave in future like they have in the past.

Temporal locality: will reference same locations as accessed in the recent past

Spatial locality: will reference locations near those accessed in the recent past

4 Caching applied to address translation

Often reference same page repeatedly, why go through entire translation each time?



Translation Buffer, Translation Lookaside Buffer: hardware table of frequently used translations, to avoid having to go through page table lookup in common case. Typically, on chip, so access time of 5-10ns, instead of several hundred for main memory.

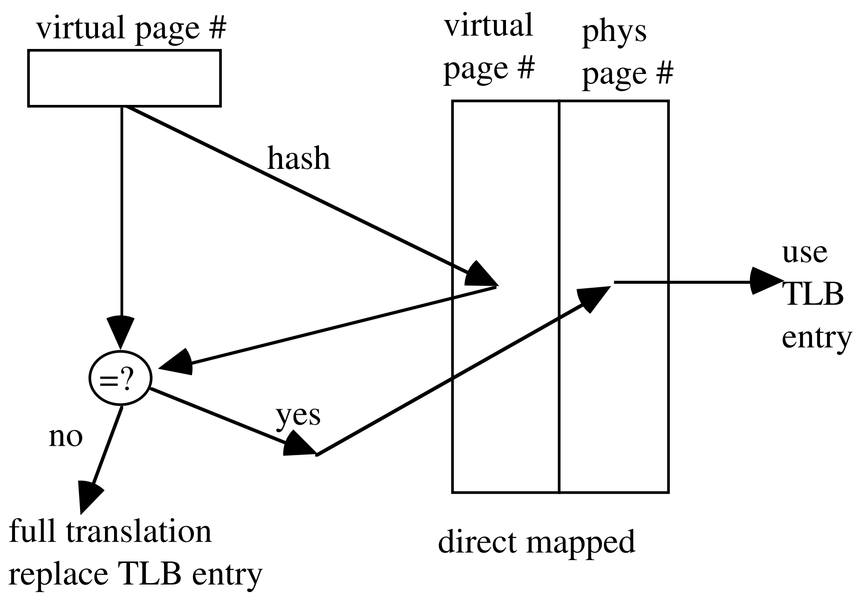
TLB for example from previous lecture (simple paging)

virtual page #	physical page #	control bits
2	1	valid, rw
-	-	invalid
0	4	valid, rw

14.4.1 How do we tell if needed translation is in TLB?

1. Search table in sequential order

2. **Direct mapped:** restrict each virtual page to use specific slot in TLB



For example, use upper bits of virtual page number to index TLB. Compare against lower bits of virtual page number to check for match.

Consider a 256 entry TLB, and the following reference stream of virtual page numbers (in hex):

621
2145
621
2145
...
321
2145
321
621

What happens to the TLB?

Do we need to store the entire virtual page number in the TLB? No, only the upper bits.

What if two pages conflict for the same TLB slot? Ex: program counter and stack.

Thrashing: cache contents tossed out even if still needed

One approach: pick hash function to minimize conflicts

What if use low order bits as index into TLB?

What if use high order bits as index into TLB?

Thus, use selection of high order and low order bits as index.

3. **Set associativity**: arrange TLB (or cache) as N separate banks. Do simultaneous lookup in each bank. In this case, called "N-way set associative cache".