

## Design Tradeoffs

---

### Virtual or physical addressing

#### Virtually-addressed caches:

- access with a virtual address (index & tag)
  - do address translation *only* on a cache miss
- + faster for hits because no address translation
- need to flush the cache on a context switch or process identification (PID) as part of the tag
  - synonyms
    - “the synonym problem”
      - if 2 processes are sharing data, two (different) virtual addresses map to the same physical address
      - 2 copies of the same data in the cache
      - on a write, only one will be updated; so the other has old data
- + there are a few solutions (which we won't study)

## Design Tradeoffs

---

### Virtual or physical addressing

#### Physically-addressed caches

- access with a physical address (index & tag)
  - do address translation on *every* access
- increase in hit time because must translate the virtual address before access the cache
- + increase in hit time can be avoided if address translation is done in parallel with the cache access
- access the cache with a virtual index:
    - restrict cache size so that cache index bits are in the page offset (virtual index = physical index)
  - then can access the TLB with the virtual address bits
  - compare the physical tag from the cache to the physical address (page frame #) from the TLB
  - can increase cache size, but still use page offset bits for the index, by increasing associativity
- + no cache flushing on a context switch
- + no synonym problem

## Cache Hierarchy

---

### Cache hierarchy

- different caches with different sizes & access times & purposes
- + decrease effective memory access time:
  - many misses in the L1 cache will be satisfied by the L2 cache
  - can avoid going all the way to memory

## Cache Hierarchy

---

### Level-1 cache

- **goal: fast access**
  - so minimize hit time (the common case)

•

•

•

•

•

## Cache Hierarchy

---

### Level-1 cache

- **goal: fast access**
  - so minimize hit time (the common case)
- small
  - so can access it in one CPU cycle  
(also there used to be chip real estate constraints)
- virtually-addressed
  - so cache accesses can be fast without constraints on cache size
- direct mapped or set associative?
  - direct mapped: faster access
  - set associative: better hit ratio
- separate caches for instructions & data
  - each is smaller than a unified cache, so the access time is lower  
(can be configured differently)
- write-through for the data cache
  - don't have to check the tag before writing data  
(also less contention for the bus between L1 and L2 caches than the system bus)

## Cache Hierarchy

---

### Level-2 cache

- **goal: keep traffic off the system bus**
  - to alleviate the processor-memory bottleneck

•

•

•

•

•

## Cache Hierarchy

---

### Level-2 cache

- **goal: keep traffic off the system bus**
  - to alleviate the processor-memory bottleneck
- big cache
  - so it will have a high hit ratio
- physically-addressed
  - no flushing on a context switch  
(also there is enough time to do address translation)
- direct-mapped
  - big direct-mapped caches have almost the same hit ratio as big set associative caches  
(also slightly less hardware cost)
- unified
  - its hit ratio is higher than that of two separate caches (I&D)  
half the size
- write-back
  - fewer updates to memory

## Alpha 21264 Memory Hierarchy

---

### L1 on-chip instruction cache

- 64KB
- 64B block
- variation of 2-way set associative
- virtually-addressed cache: virtual index, virtual tags
  - TLB lookup in parallel

### L1 on-chip data cache

- 64KB
- 64B block
- 2-way set associative
- virtually-addressed cache: virtual index, physical tags
  - 2-bits taken from outside the page offset
    - a virtual address can reside in one of four cache locations, depending on the virtual-to-physical translation for these bits
    - HW guarantees that only one will reside in the cache at a time
  - TLB lookup in parallel
- write-back

## Alpha 21264 Memory Hierarchy

---

### L2 on-board cache

- 1MB - 16MB
- low set associative
- 64B block
- physically indexed
- 12-cycle load-to-use latency

### TLBs

- separate instruction & data TLBs
- fully-associative
- 128 entries (instruction); 128 entries (data)
- maps 1, 8, 64 or 512 contiguous 8KB pages
- round-robin allocation
- 8-bit PID
- TLB misses handled in software with hardware assists (special instructions for invalidating TLB entries)

## Pentium Pro Memory Hierarchy

---

### L1 on-chip instruction cache

- 8KB
- 32B block
- 4-way set associative
- physically-addressed cache: physical index, physical tags
  - TLB lookup in parallel

### L1 on-chip data cache

- 8KB
- 32B block
- 2-way set associative
- physically-addressed cache: physical index, physical tags
  - TLB lookup in parallel
- write-back

## Pentium Pro Memory Hierarchy

---

### L2 on-chip cache

- 256KB
- 4-way set associative
- 32B block
- physically indexed

### TLBs

- separate instruction & data TLBs
- 4-way set associative
- 32 entries (instruction); 64 entries (data)
- TLB misses handled in hardware

## Measuring Cache Hierarchy Performance

---

### Effective Access Time:

$$\text{hit time}_{L1} + \text{miss ratio}_{L1} \cdot \text{misspenalty}_{L1}$$
$$\text{hit time}_{L2} + \text{miss ratio}_{L2} \cdot \text{misspenalty}_{L2}$$

## Comparing Caches & Paging

---

### Timing aspects

- cache miss takes about 6 (L1) to 60 (L2) cycles
- TLB miss takes 100s of cycles
- page fault takes milliseconds (millions of cycles)

### How a miss/fault is handled

- cache miss: in hardware
- TLB miss: either in hardware or software  
if software, often there is no trap
- page fault: in software  
trap to the operating system

### Mapping

- caches: direct-mapped or set associative
- TLBs: usually fully associative
- paging: fully associative

## Comparing Caches & Paging

---

### Page/block size

- cache block: 8 to 128 bytes
- TLB entry: size of a PTE (typically 4 to 8 bytes)
- page: 4KB - 4MB

### Memory update policy

- caches: write-through or write-back to memory
- TLBs: write-back to memory
- pages: write-back to disk

### Replacement policy

- TLBs & caches: LRU if 2-way set-associative, but not as important
- paging: important to be LRU (why?)

### All are demand-driven

Be sure you know why all these choices were made!