NOTES: Cache

notes/cache.md

Cache

Motivation

- Cache is needed because of locality
- Temporal locality: items accessed now are likely to be needed again soon
- Spatial locality: if an item is accessed, other items "next to" it are likely to be accessed soon
- Working set theory: The memory you need now (the working set) is smaller than the size of all memory usage

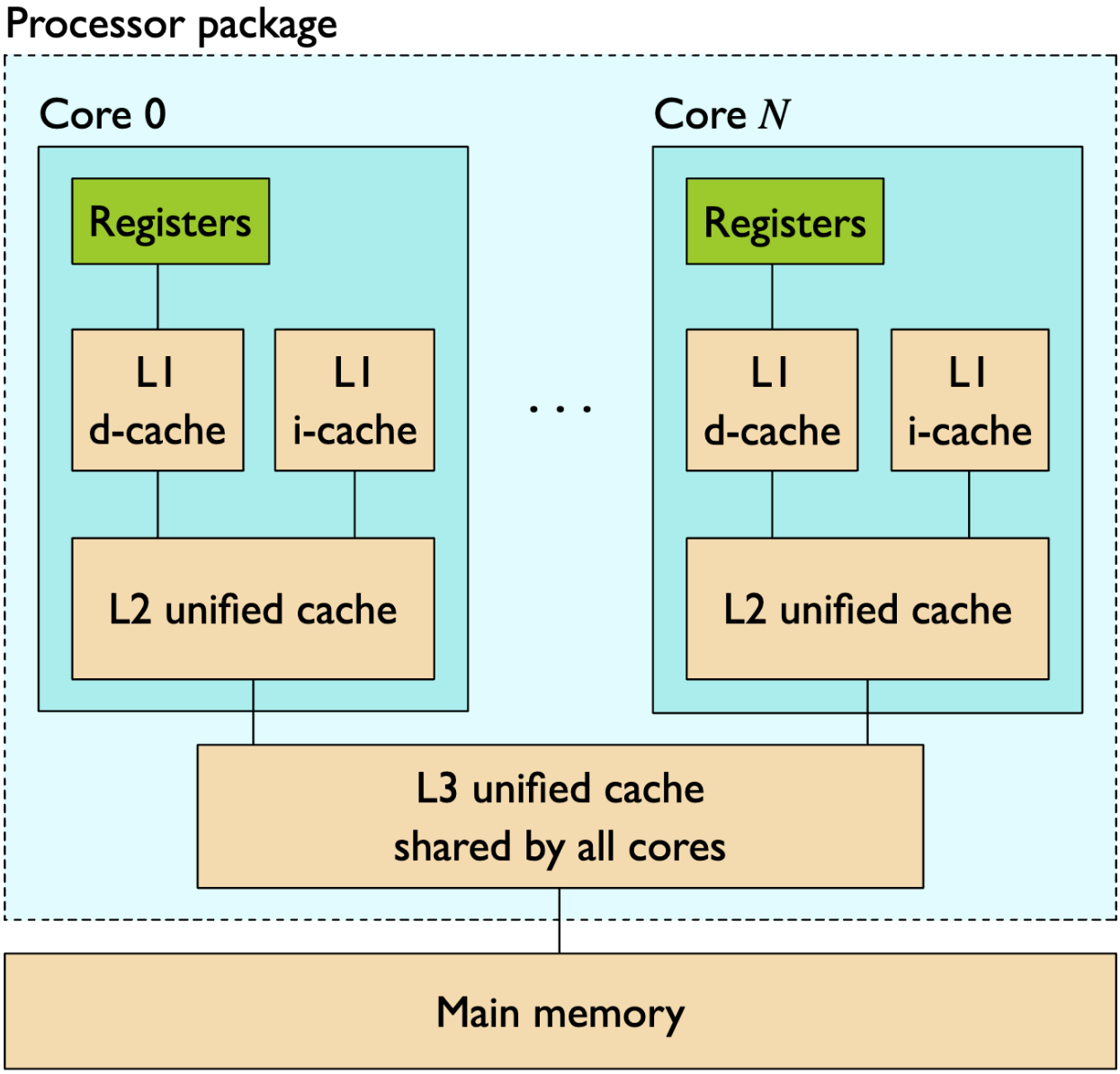
Problems

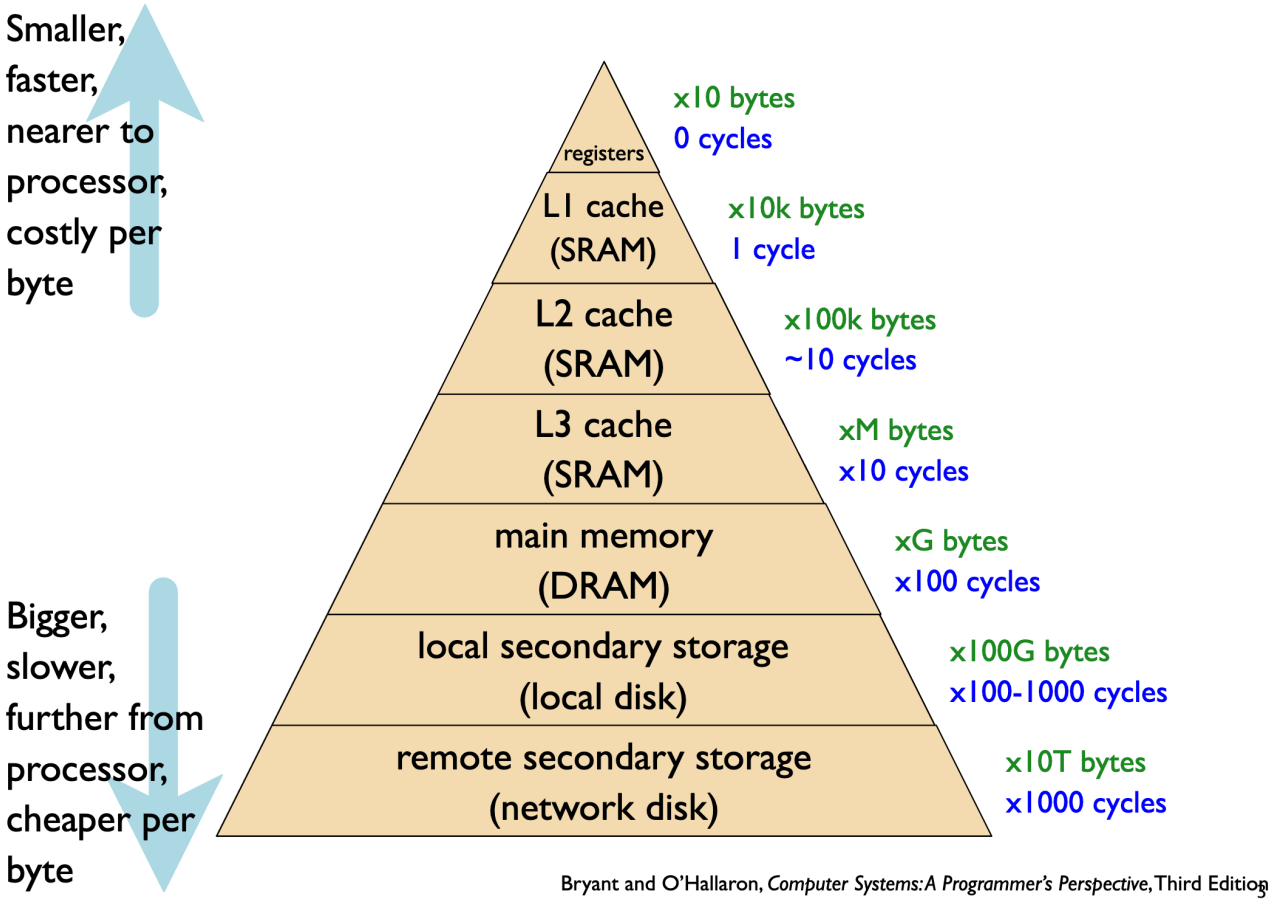
- When to invalidate cache
- Placement policy
- Replacement policy
- Eviction policy
- Cache agreement/coherence with multiple caches
- Cache misses
    - Cache miss rate: `(# of cache misses) / (# of requests)`

Design

- Cache is designed in a heirarchy—L1 is closer to CPU and faster but has less memory than L2, etc

- L1 and L2 are usually inside the core, but L3 is shared between cores





Blocking

- **Blocking**: take advantage of spatial locality
  - Entire memory is partitioned into blocks of fixed size, and blocks are cached together
  - Block size can be changed to optimize this, but in practice it's best at 64 bytes

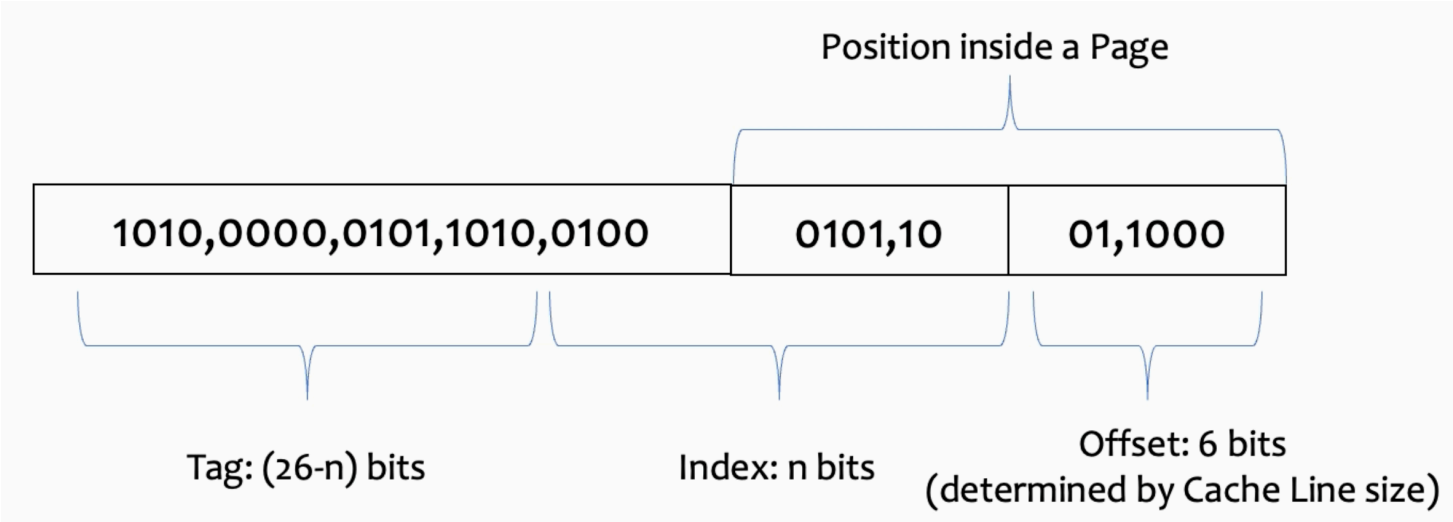
Data Stored in a Cache

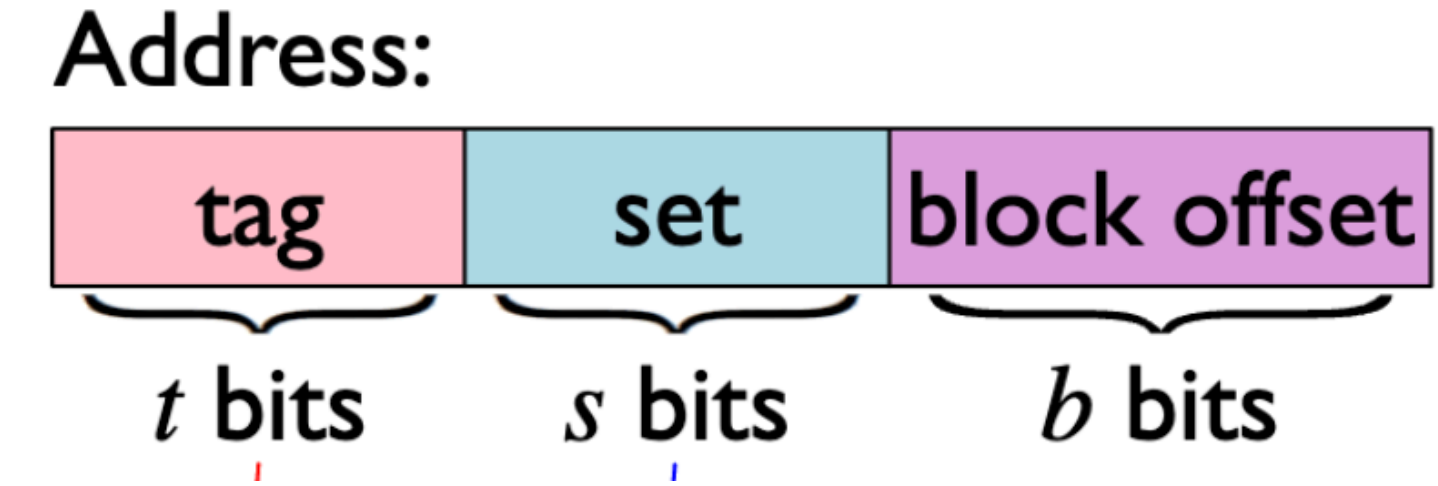
- The data stored in one cache line looks something like this:
  - Valid bit tracks whether the cache line is valid
  - Tag: The upper 26 bits of the address, to disambiguate collisions
  - Cache block: the actual data—32 bytes

```
valid | tag | cache block (data)
```

- **IMPORTANT**: When talking about cache size, we're talking about the amount of data able to be stored, without including the headers (valid bit, tag, etc) in our calculations.
  - So, the number of cache lines is the cache size divided by the size of the cache block (data part of the cache line)

Cache Addressing





- An address looks like this:
  - Tag: used to disambiguate collisions
    - When you find a supposedly-matching cache line, check the valid bit, and compare the tags
  - Set index: used to locate the set (associativity) in the cache
  - Offset: used to locate the data within the cache block
    - If the cache block is 32 bytes, the offset is 5 bits long

```
tag | set index | offset
```

Placement Policy

Option 1: Directly Mapped

- Since a cache line is usually larger than a page, you could map the page offset to the position in the cache

- The index is used to locate the cache line
    - The index usually goes past the page offset because the cache is larger than the size of a page
- Since a cache block is $2^6$26 bytes, the last 6 bits of the address are the offset (position within the block)
- Since only the *n* bits in the index are used, there can be collisions, requiring a tag to differentiate between the blocks
    - If two blocks are mapped to the same cache line, one must be evicted

Example Lookup

1. Use Index to locate the Cache Line
2. If the tag matches (hit), use the offset to find the data
3. If the tag doesn't match (miss), evict the cache line to memory, load the correct memory block, and use the offset to find the data

Option 2: Fully Associative

- A block in memory can be placed anywhere within a given cache line
- A linear search is required to find the block in the cache (search until tags match)
- Reduces the change of index collisions

Option 3: Set Associative

- Blocks in memory can be placed anywhere within a set of size *n*
- The index tells you which set you're in
- A linear search is required to find the block, but you only search the set
    - You search until you the tags match, or get to the end of the set

Virtually vs. Phyiscally Addressed Cache

- The cache can be put either before or after the MMU, which determines if it's addressed by virtual or physical addresses
    - Before MMU: virtually-addressed cache
        - However, the virtual addresses are per-process, so the OS has to tell the cache about PIDs
        - During a context switch, the cache would need to be erased
    - After MMU: physically-addressed cache
        - You have to go through the MMU before you can access the cache
    - Now that TLBs are so fast, physical-addressed cache is preferred

Virtually Indexed, Physically Tagged (VIPT)

- **You can use a VIPT scheme to both locate the correct set and translate the page to physical address space in parallel**
- The cache is indexed by the virtual address (correct set is found)
- Tags are compared when searching the set
    - The tag is the entire page number (not just the bits not used by the index+offset)
        - This is to make it so that the found cached memory is for the correct page

Replacement/Eviction Policy

- With set- and fully- associative placement policy, which cache line should we evict when there's a cache miss?
- Option 1: choose a random cache line
    - Con: it's hard to implement an RNG
- Option 2: Least Recently Used (LRU)
    - Con: it's hard to keep track of time of last access

Write Policy

- What to do on a write hit?
    - *Write-through*: write immediately to memory
    - *Write-back*: defer write to memory until replaced/evicted (lazy)
- What to do on a write miss?
    - *Write-allocate*: load into cache, update in cache
    - *No-write-allocate*: write straight to memory
- Typical implementations
    - Write-through + no-write-allocate
    - Write-back + write-allocate