CLOUD COMPUTING APPLICATIONS

Caching as a Universal Concept: Overview

Prof. Reza Farivar

The need for caching

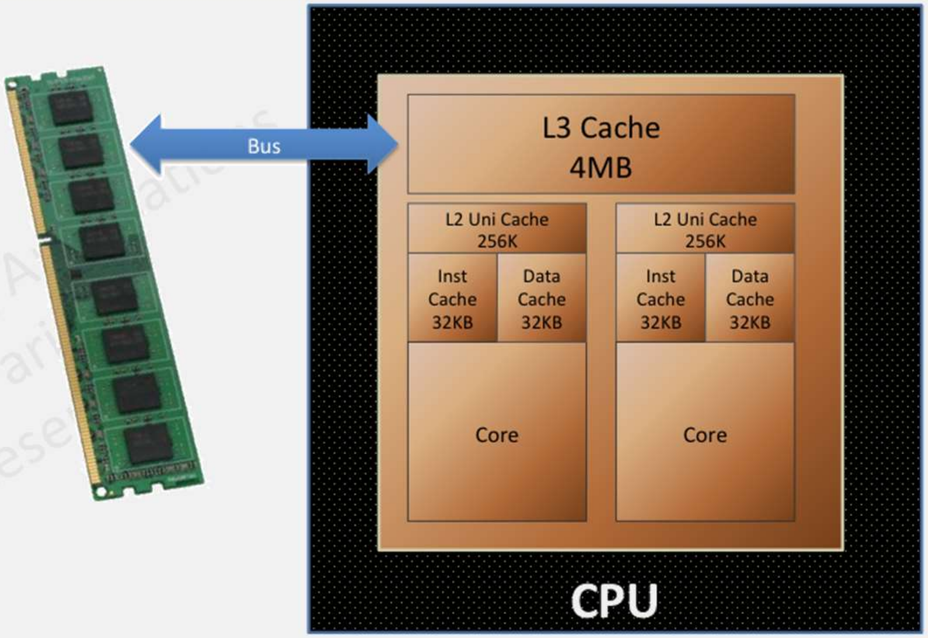
- Success for many websites and web applications relies on speed
  - Users can register a 250-millisecond (1/4 second) difference between competing sites
    - "*For Impatient Web Users, an Eye Blink Is Just Too Long to Wait*" NYT, 2012
  - For every 100-ms (1/10 second) increase in load time, sales decrease 1 percent
  - Data that is cached can be delivered much faster
- In-memory Key-value stores can provide sub-millisecond latency
  - querying a database is always slower and more expensive than locating a key in a key-value pair cache

Caching

- Caching is a universal concept
- Based on the principle of locality (aka. locality of reference)
  - Tendency of the "processor" to access the same set of memory locations repetitively over a short period of time
  - Temporal locality vs spatial locality
- Whenever you have "large + slow" source of information and "small + fast" storage technology, you can use the latter to cache the former
- You can see this concept anywhere from CPUs and processors, to operating systems, to large web applications on the cloud

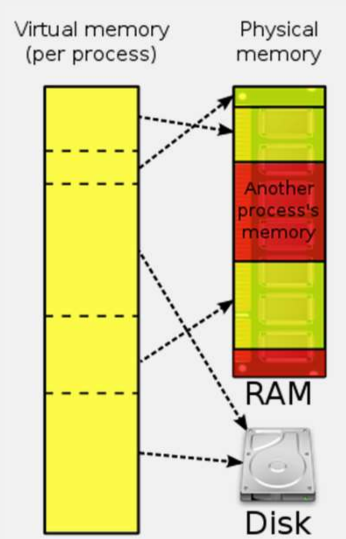
Caching in Processors: Data & Instructions

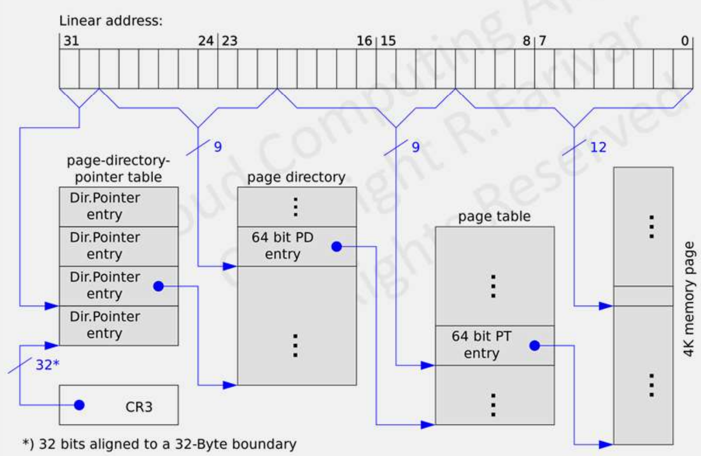
- Big/Slow RAM memory
- Multiple layers of caching
- Access to data exhibits temporal and spatial locality
  - L1 Data Cache, L2 and L3 Caches
- The program instructions have spatial and temporal locality
    - One instruction after another
    - Loops
  - Also branch locality
    - If … else

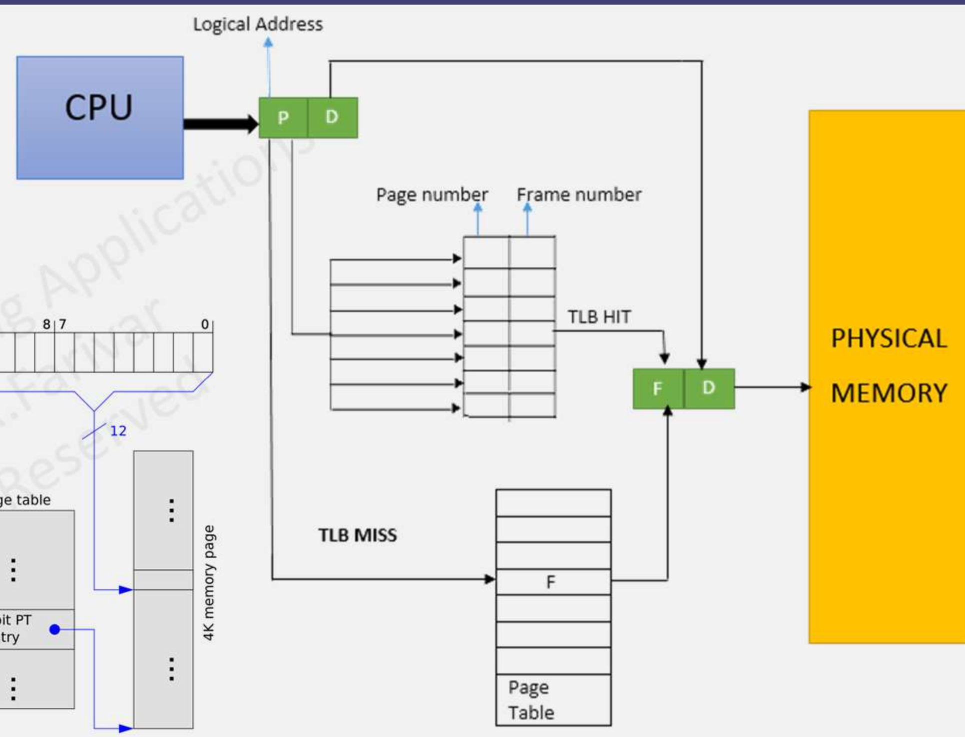


Caching in Processors: Virtual Memory

- Virtual memory address translation
- Every memory access needs a translation
  - Needs a page walk
  - Slow/big source of data
- Translation Lookaside Buffer (TLB)
  - iTLB and dTLB







Virtual memory and OS-level Page Caching

- Virtual memory:
  - Each process thinks it has 2^{48} = 256 TB of memory
- Paging
  - computer stores and retrieves data from secondary storage (HDD/SSD) for use in main memory
  - RAM acts as the "cache" for the SSD
  - When a process tries to reference a page not currently present in RAM, the processor treats this invalid memory reference as a ***page fault*** and transfers control from the program to the operating system
- OS does the following
  - Determine the location of the data on disk
  - Obtain an empty page frame in RAM to use as a container for the data
  - Load the requested data into the available page frame
  - Update the page table to refer to the new page frame
  - Return control to the program, transparently retrying the instruction that caused the page fault

Linux Page Cache

- Linux kernels up to version 2.2 had both a Page Cache as well as a Buffer Cache. As of the 2.4 kernel, these two caches have been combined. Today, there is only one cache, the Page Cache.
- This mechanism also caches files.
- Usually, all physical memory not directly allocated to applications is used by the operating system for the page cache
- So the OS keeps other pages that it may think may be needed in the page cache.
- If Linux needs more memory for normal applications than is currently available, areas of the Page Cache that are no longer in use will be automatically deleted.

```
wfischer@pc:~$ dd if=/dev/zero of=testfile.txt bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0,0121043 s, 866 MB/s
wfischer@pc:~$ cat /proc/meminfo | grep Dirty
Dirty:              10260 kB
wfischer@pc:~$ sync
wfischer@pc:~$ cat /proc/meminfo | grep Dirty
Dirty:                  0 kB
```

https://www.thomas-krenn.com/en/wiki/Linux_Page_Cache_Basics

Linux VFS Cache

- Dentry Cahce
  - A "dentry" in the Linux kernel is the in-memory representation of a directory entry
  - A way of remembering the resolution of a given file or directory name without having to search through the filesystem to find it
  - The dentry cache speeds lookups considerably; keeping dentries for frequently accessed names like `/tmp`, `/dev/null`, or `/usr/bin/tetris` saves a lot of filesystem I/O.
- Inode Cache
  - As the mounted file systems are navigated, their VFS inodes are being continually read and, in some cases, written
  - he Virtual File System maintains an inode cache to speed up accesses to all of the mounted file systems
  - Every time a VFS inode is read from the inode cache the system saves an access to a physical device.

Caching in Distributed Systems

- CDN Caching
- Web Server Caching
  - Reverse Proxies
  - Varnish
  - Web servers can also cache requests, returning responses without having to contact application servers
    - NGINX
- Database Caching
- Application Caching
  - In-memory caches such as Memcached and Redis are key-value stores between your application and your data storage

What to Cache

- There are multiple levels you can cache that fall into two general categories: **database queries** and **objects**:
  - Row level
  - Query-level
  - Fully-formed serializable objects
  - Fully-rendered HTML

Caching at the database query level

- Whenever you query the database, hash the query as a key and store the result to the cache
- Suffers from expiration issues:
  - Hard to delete a cached result with complex queries
  - If one piece of data changes such as a table cell, you need to delete all cached queries that might include the changed cell

Caching at the object level

- See your data as an object, similar to what you do with your application code. Have your application assemble the dataset from the database into a class instance or a data structure(s):
  - Remove the object from cache if its underlying data has changed
  - Allows for asynchronous processing: workers assemble objects by consuming the latest cached object
- Suggestions of what to cache:
  - User sessions
  - Fully rendered web pages
  - Activity streams
  - User graph data