Design and Engineering of Computer Systems

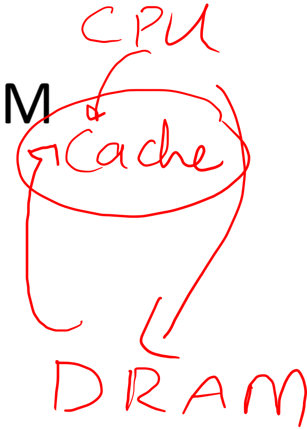
Lecture 34: Caching

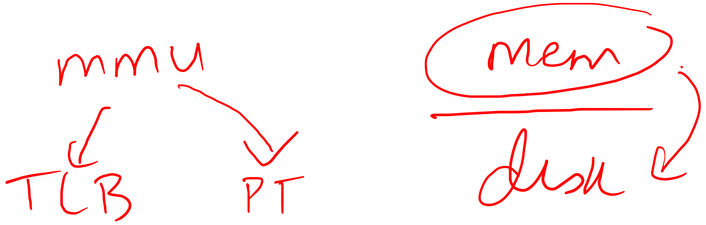
Mythili Vutukuru

IIT Bombay

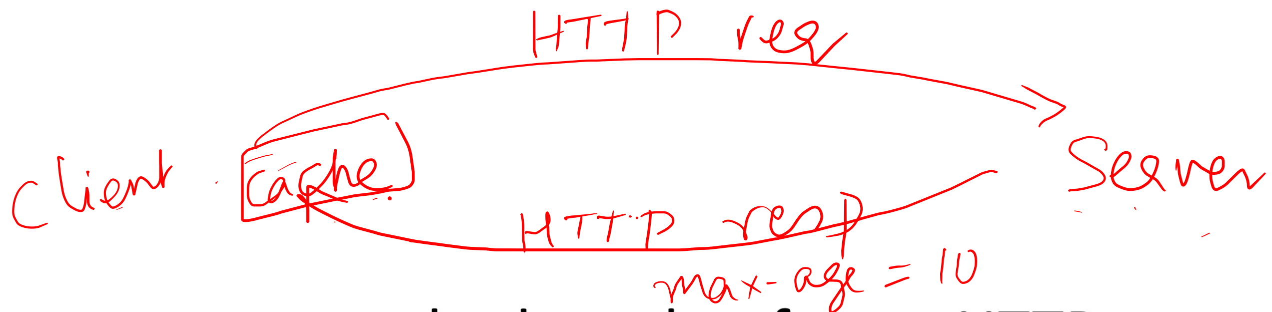
Caching in computer systems

- Caching: when data has been fetched from a “far away” location, keep a copy in a “near by” location, in case it is needed again in future
  - Cache has limited capacity and cannot store all the original data
- Why caching? If fetching from “far away” component is the performance bottleneck, caching will improve capacity
- Examples of caching in computer systems seen so far
  - CPU caches keep a copy of data fetched from main memory (DRAM) in SRAM (more expensive but faster access time than DRAM)
  - Disk buffer cache keeps a copy of recently accessed disk blocks in memory
  - TLB caches recently accessed virtual-to-physical address translations
- More examples of caching in this lecture

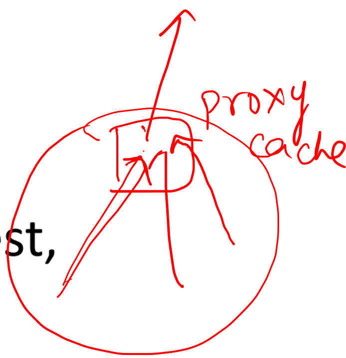




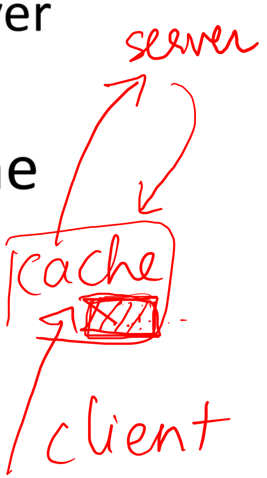
HTTP caching



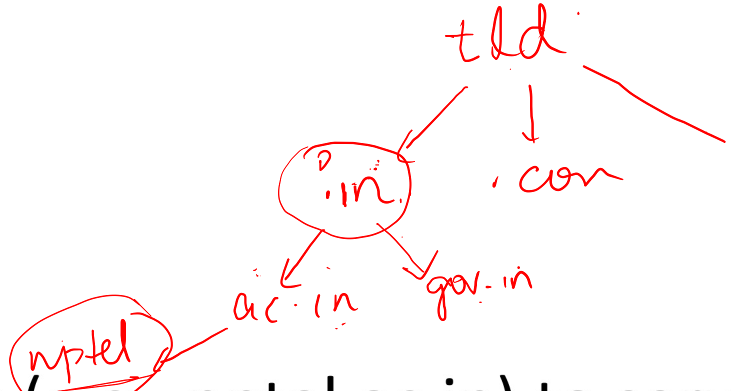
- HTTP cache: received HTTP responses are cached, so that future HTTP requests for same URL can be satisfied without fetching over network
  - Shared HTTP caches: proxy HTTP server close to the client, intercepts HTTP request, checks if response is locally cached, returns if cache hit, fetch from server if miss
  - Private HTTP caches within user browsers also
  - Cannot cache encrypted HTTPS content, only plain HTTP



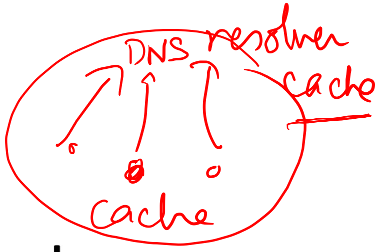
- What if server changes web page? How will cache know?
  - Server indicates how long the response can be cached (or even say that it should never be cached) in HTTP response headers (e.g., cache-control header, max-age header)
- Conditional HTTP GET: cached content has expired, user has requested same URL, cache performs conditional fetch (fetch only if needed)
  - Cache sends HTTP GET to server, indicating last modified time of its cached copy
  - Server indicates whether previous content is still valid or sends updated response



DNS caching



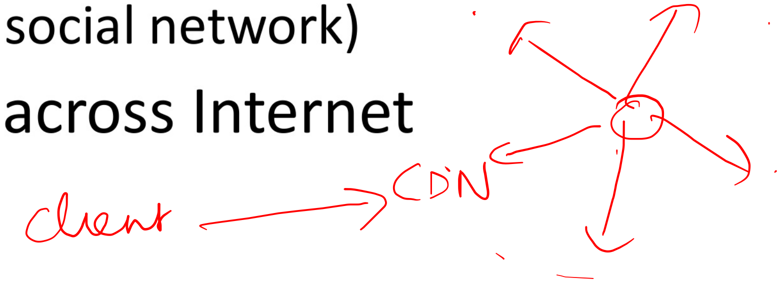
- DNS: resolves domain names (e.g., nptel.ac.in) to server IP address
  - DNS records stored at authoritative name servers hierarchically
  - DNS resolvers contact name servers recursively to resolve a name



- DNS records mapping domain names (of final host as well as intermediate authoritative name servers) to IP addresses are cached
  - Locally within machine, or shared across clients at DNS resolver
- DNS resolution involves multiple network communications and can take up to few tens of milliseconds
  - Caching of popular DNS records is critical for good performance
- Time to live (TTL) in DNS response indicates how long it can be cached
  - Allows server to update IP address and change DNS records after some time

Application-layer caching

- Application-level data objects, database queries, and many other app-level information can also be cached
  - Front-end caches responses from app servers
  - App server caches responses from database
  - Reduce communication between app components, improve performance
- App-layer caches are separate software components that sit between various other components, e.g., between app server and database
- In-memory key-value stores (e.g. Redis, memcached) often used as app-layer caches
  - Example: key = database query, value = result of query (app database queries)
  - Example: key = image name, value = image (popular images in social network)
- CDNs also cache some app-layer objects and web pages across Internet

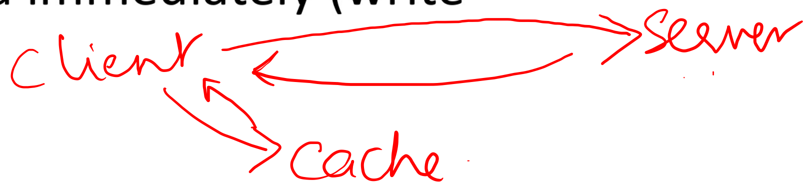


When to cache?

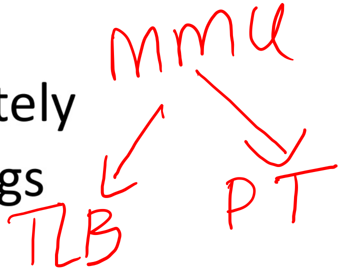
- Workloads which lead to high cache hit rates
  - High locality of reference (i.e., past data is needed in future again)
  - Skewed distribution, some items are very popular and accessed very often (heavy hitters)
- Cache has to be faster access technology and/or closer geographically than original copy of data
  - In-memory caches of database queries vs original database on disk
  - SRAM (CPU caches) is faster access than DRAM (main memory)
- Caches cannot accommodate all the original data, so need a good eviction policy to decide which data is cleared when cache is full, e.g., least recently used (LRU)
  - Eviction policy must ensure less probability of evicting a useful item needed in future
  - Eviction policy should be implementable easily with low overhead
- Cache has to be large enough size to accommodate the working set size, i.e., most frequently used data in a given interval of time
  - Otherwise, very poor hit rates, no benefit of using cache

Position of cache

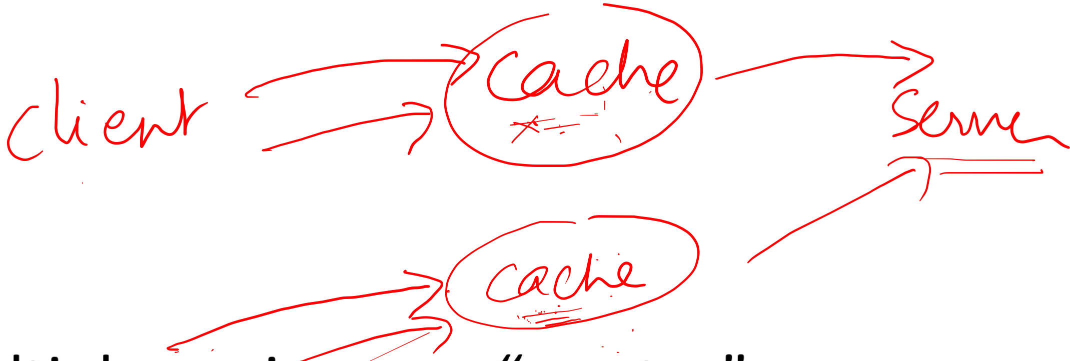
- Suppose client accesses data from server (which has original copy), keeps a copy in cache. Where is cache located relative to client and server?
- Inline cache: cache is on communication path between client and server
  - Client checks cache. If cache miss, cache fetches data and gives it to client, stores copy
  - Client writes in cache first, cache updates original copy of data immediately (write through) or later on (write back)
  - Example: CPU caches, disk buffer cache



- Write-aside/look-aside cache: cache not on direct path, checked on the side
  - Client checks cache. If cache miss, client fetches data from server directly, updates the cache afterwards
  - When client or server update data, cached copy is invalidated or updated separately
  - Example: MMU uses TLB (Translation Lookaside Buffer) to cache address mappings



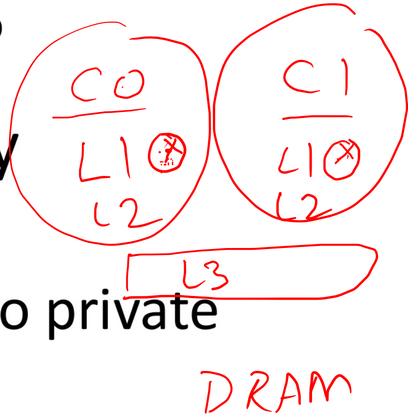
Populating cache contents



- With caching, one piece of data may have multiple copies: one “master” copy and multiple cached copies
  - Contents of memory at some address is stored in main memory itself (master copy) and in one or more CPU caches (private to cores, shared across cores)
- Demand filled cache: content populated in cache only when needed, when requested by clients
  - Example: CPU caches, HTTP caches, DNS caches
  - Different cached copies may diverge from master copy based on access pattern
- Proactive cache: server proactively updates all cached copies whenever it knows content has changed
  - Example: In some CDNs, server pushes updated content to CDN replicas
  - Easier to maintain consistency of cached content across replicas

Cache consistency: tracking replicas of data

- How do we keep track of multiple copies of data, to keep in sync?
- In CPU caches, information about which cache has which memory locations is known across all CPU cores. How?
    - Snooping: when one CPU core accesses memory location and fetches into private cache, all other cores snoop on the access and remember it
    - Directory: all cores update information about their cached memory locations in a directory that is accessible to all CPU cores
- In some systems, e.g., CDNs, server has master copy of content and maintains information on which all CDN replicas it has pushed content to
- In some systems, e.g., HTTP caches within browsers, it is not possible to keep track of all copies of data across all caches of users



Cache consistency: updating replicas

- How to ensure all replicas of cached data are kept in sync?
- Cache coherence protocols: when one CPU core updates the value in its private cache, all copies of the item in other caches are synchronized using cache coherence mechanisms
  - Other CPU cores which have older value in private cache update their value
  - Or, other CPU cores with older value mark their copy as invalid, fetch again in future
- What if server changes value and doesn't keep track of who all have cached it, e.g., HTTP caches?
  - Use some way to identify what is latest copy of data: sequence number, version number, last modified time, …
  - Whenever data is modified in one of the caches or master copy, version number or timestamp is updated
  - When accessing cached copy, check that version number / timestamp is latest (update to latest copy or invalidate if value is stale)

Summary

- In this lecture:
  - Examples of caching in computer systems
  - Principles of designing caches in computer systems
- Using Wireshark, examine the various HTTP headers that are responsible for controlling cache behavior. Check what all web pages are cached in your browser's cache.