Caching

Host: ducvm10
Presenter: ducvm10, hoangtl1, huylq33

Content

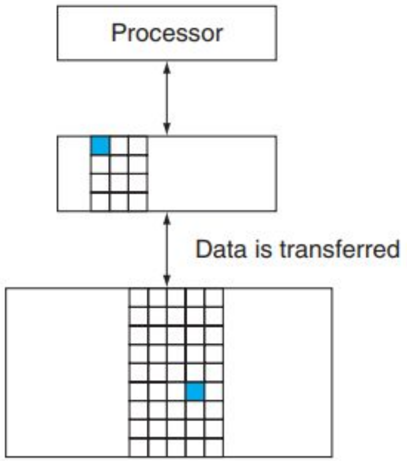
1. Introduction about cache (hoangtl1)
2. Data protection when loss power (ducvm10)
3. Caching in monolithic and microservices (ducvm10)
4. Typical cache libraries (huylq33)
5. Caching in Hibernate (ducvm10)

1. Introduction

- In computing, a cache is a hardware or software component that stores data
  → future requests for that data can be served faster
- A cache hit = requested data can be found in a cache
- A cache miss = requested data cannot be found in a cache
- A block = minimum unit of information represented in cache

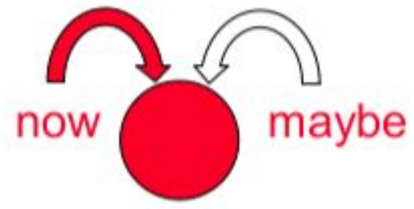
- Caches must be relatively small for efficiency

  → take advantages of high degree of locality of reference.

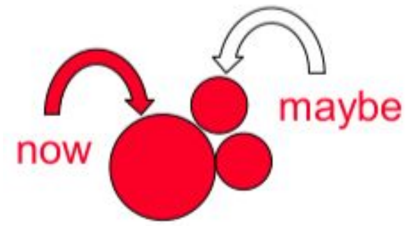


Principle of locality

- Temporal locality
    - If an item is referenced, it will tend to be referenced again soon
    - ex: instructions in a loop, induction variables, …



- Spatial locality
    - if an item is referenced, items whose addresses are close by will tend to be referenced soon
    - ex: sequential instruction access, program code, array data, ...



1. Introduction

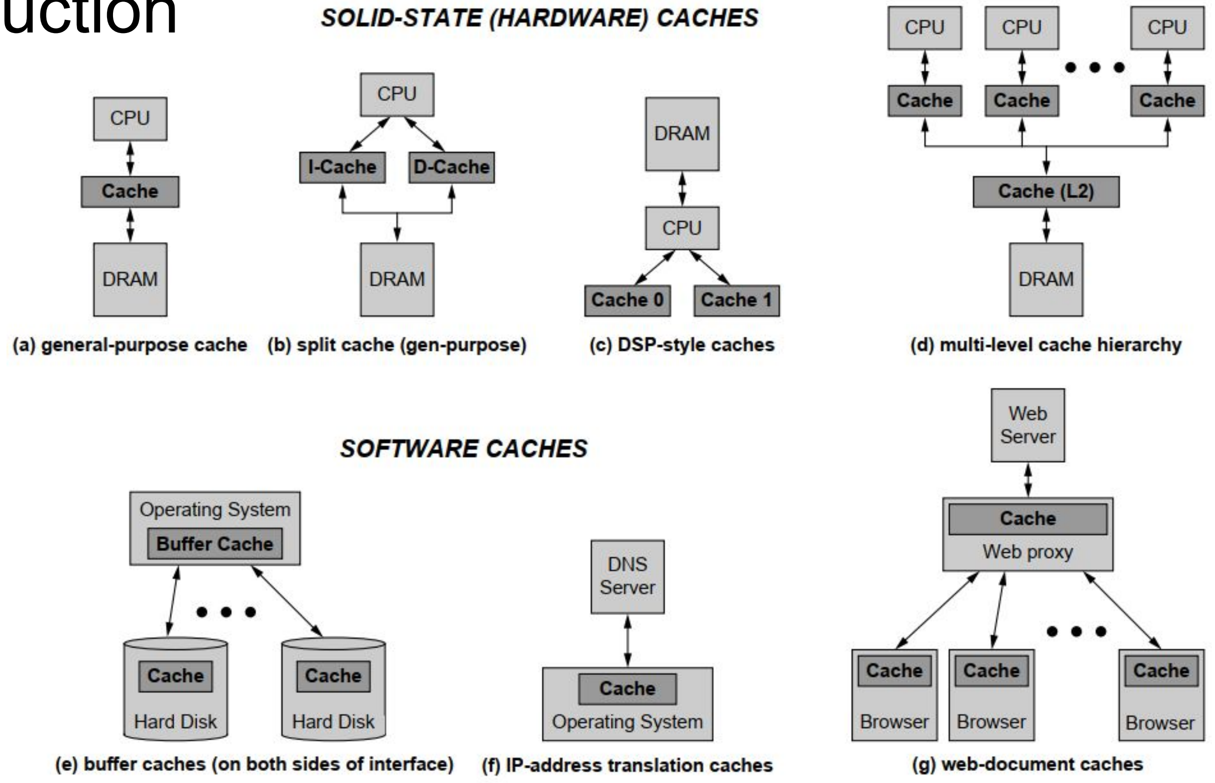
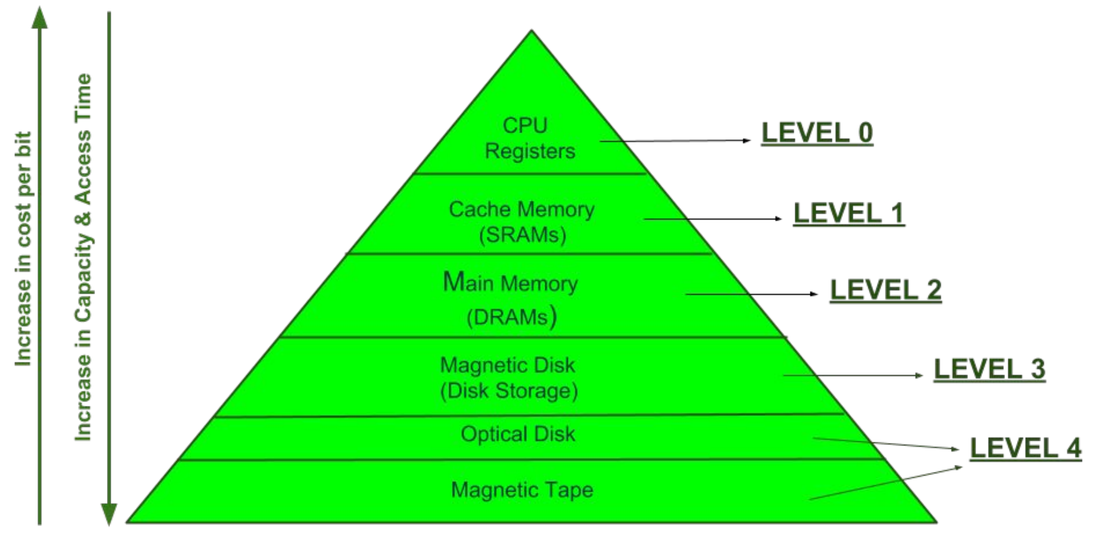


FIGURE 1.1: Examples of caches. The caches are divided into two main groups: solid-state caches (top), and those that are implemented by software mechanisms, typically storing the cached data in main memory (e.g., DRAM) or disk.

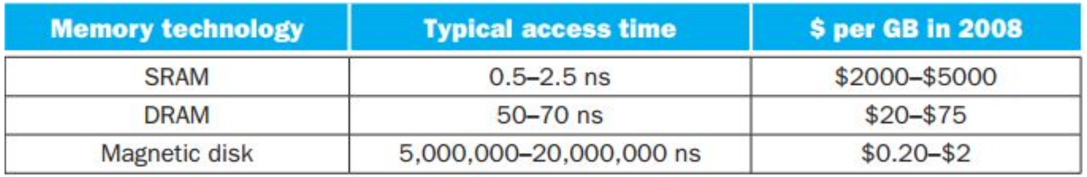
Memory hierarchy

- 3 primary technologies: SRAM, DRAM, Magnetic disk

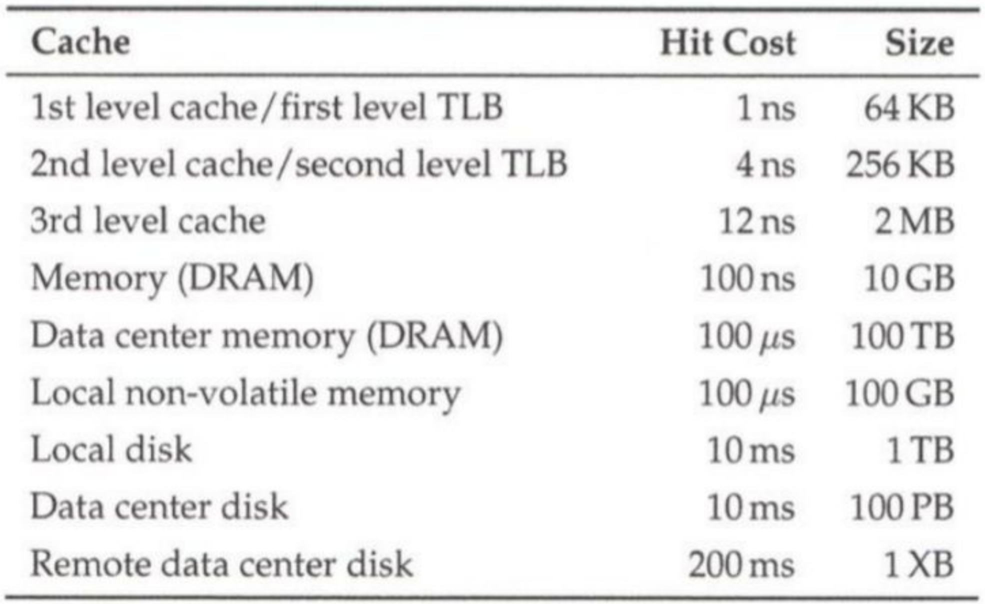


MEMORY HIERARCHY DESIGN

Memory hierarchy



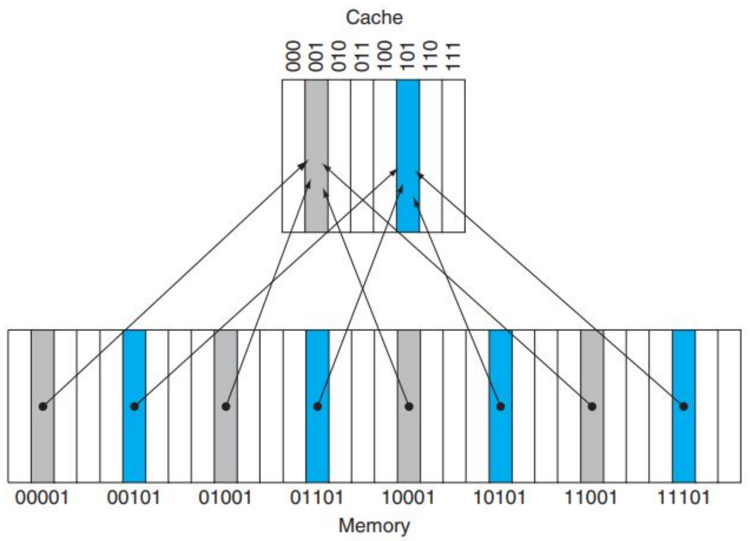
Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Magnetic disk	5,000,000–20,000,000 ns	$0.20–$2

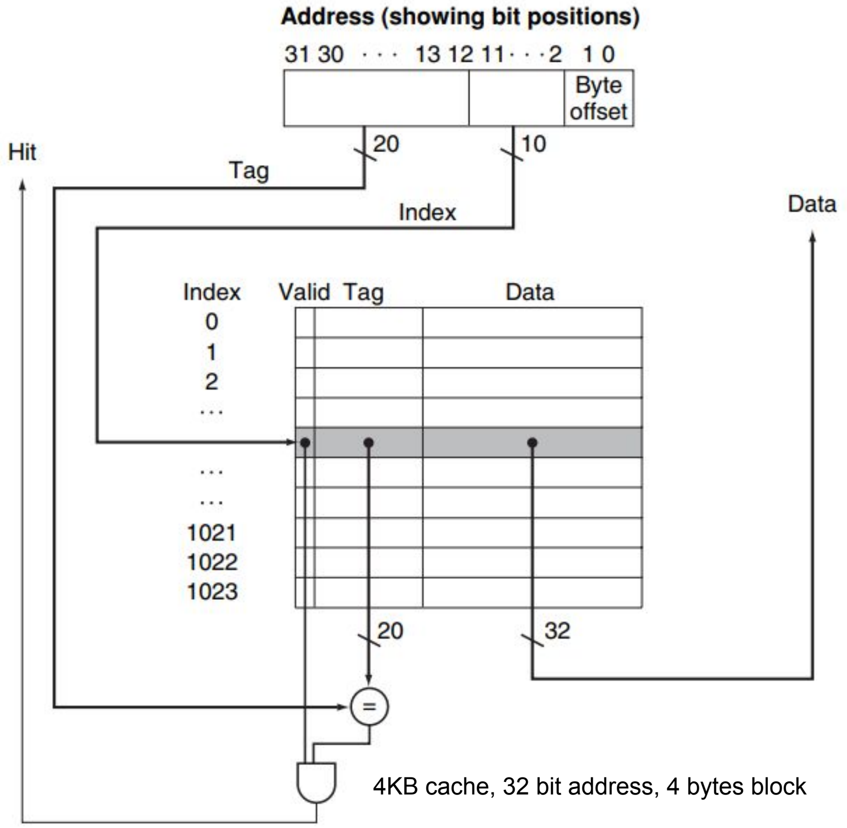


Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

Cache mapping

- Direct-mapped cache
  - a direct mapping from any block address in memory to a single location in the upper level of the hierarchy

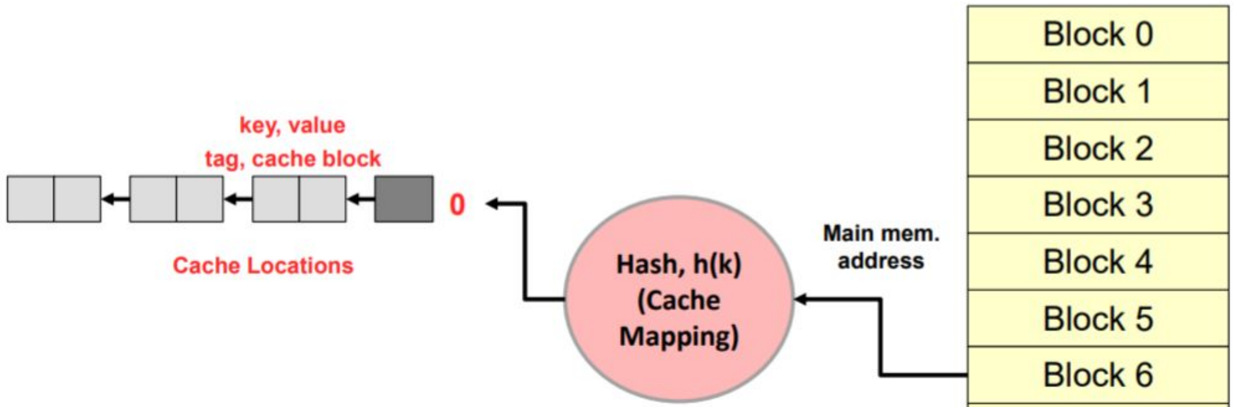




4KB cache, 32 bit address, 4 bytes block

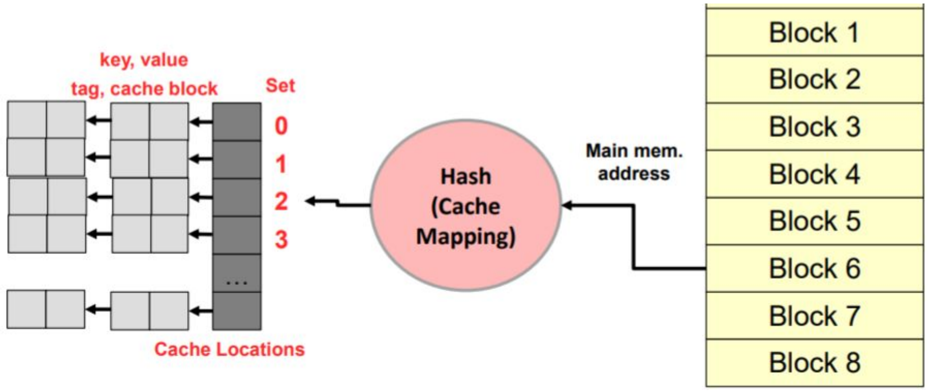
Cache mapping

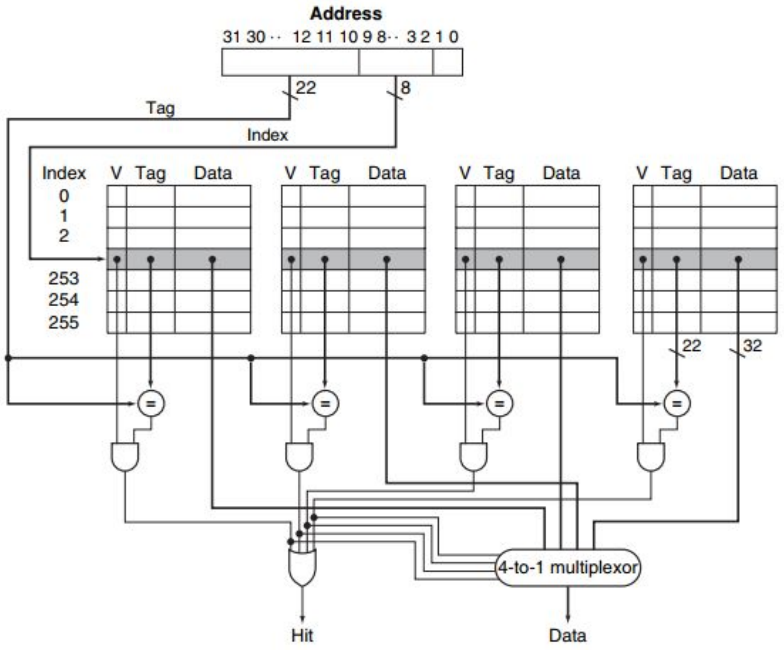
- Fully associative cache
  - A cache structure in which a block can be placed in any location in the cache



Cache mapping

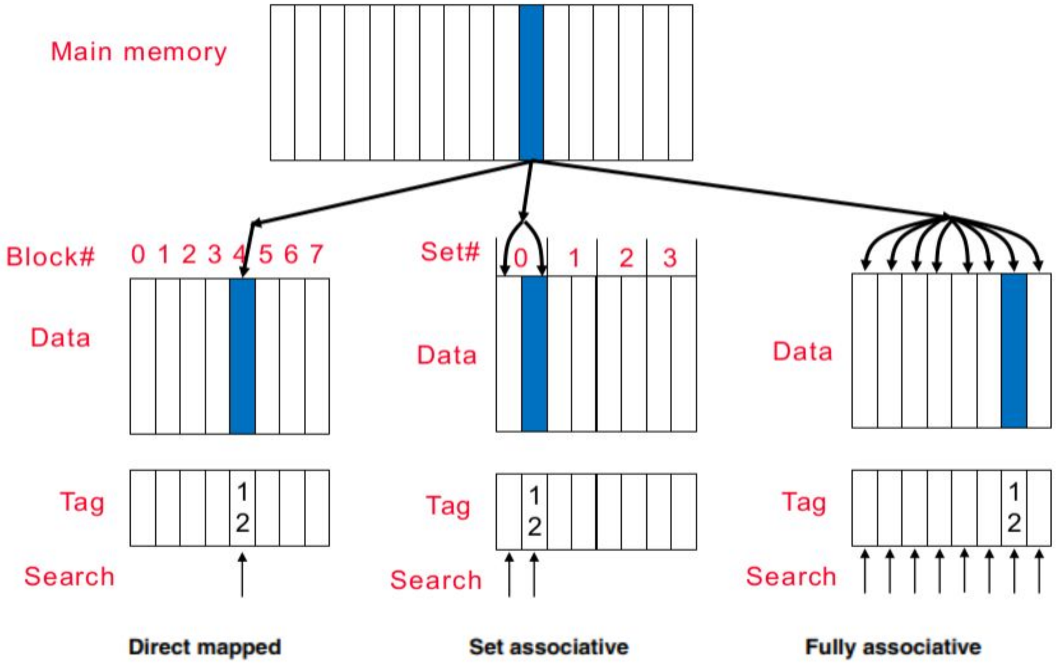
- Set-associative cache
    - A cache that has a fixed number of locations (at least two) where each block can be placed





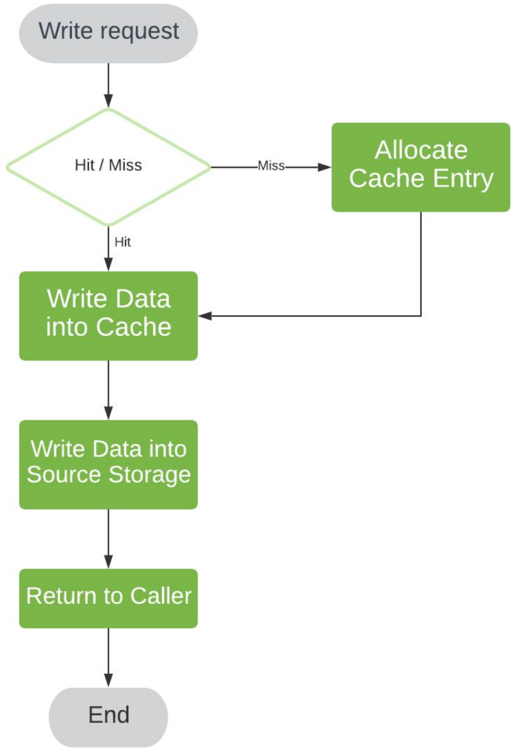
Cache mapping

- Increasing the degree of associativity
  - (+) decreases cache misses
  - (-) increase in hit time



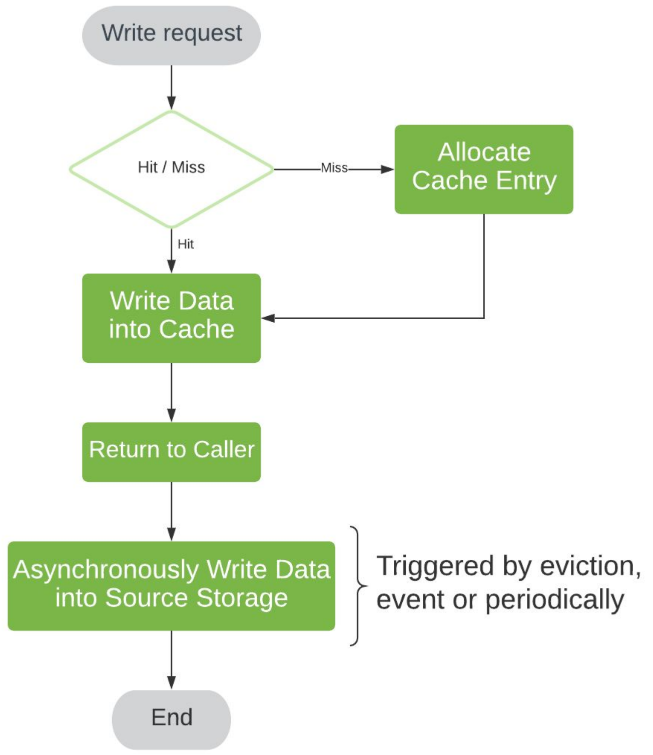
Writing policies

- Write-through
  - Write is done synchronously both to the cache and to the main memory.
  - (+) Pros: Simple, ensuring data consistency
  - (-) Cons: every write causes the data to be written to main memory → inefficiency
    - Solution: Write buffer
    - A queue that holds data while the data is waiting to be written to memory



Writing policies

- Write-back
  - The information is written only to the block in the cache. The modified block is written to main memory only when it is replaced (using dirty bit)
  - (+) Pros: reduce number of main memory references by taking advantages of temporal locality of reference.
  - (-) Cons: Potential inconsistent data



Cache replacement policies

- Which block is replaced on a miss?
- An algorithm to determine which item(s) to discard to make room for new ones when the cache is full
    - Least recently used (LRU)
    - Random
    - First in first out (FIFO)
    - Least frequently used (LFU)
    - …

2. Data protection when loss power

a, Battery backup cache.

- Historically, caches have been protected by the use of batteries to provide backup power to the memory in the event of an unexpected power outage.

IBM SAN Volume Controller 2145-DH8 Raid Cache BackUp Battery 00AR260 00AR056

2. Bảo toàn dữ liệu trong cache khi mất nguồn điện

a, Battery backup cache.

- Some limitations :
    + The cached data is only maintained when the battery is capable of supplying power. Traditionally, the goal is to maintain cached data for at least 72 hours.
    + The other challenge is that if the size of the memory is increased, the battery capacity must increase. As the battery capacity increases, the battery size increases, and in some cases, it may become very large.

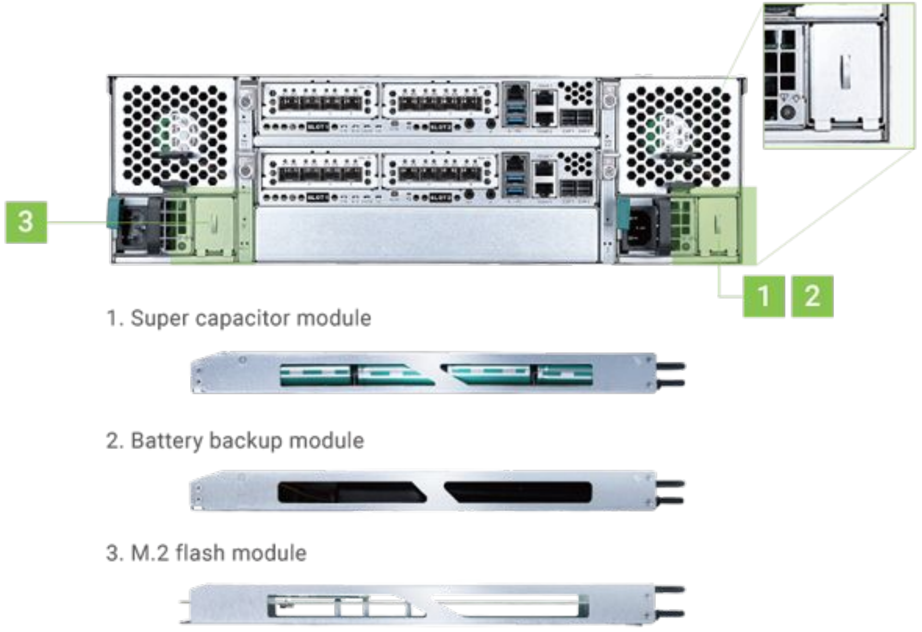
2. Bảo toàn dữ liệu trong cache khi mất nguồn điện

b, Cache-to-flash

- Cache-to-Flash memory protection function will safely transfer the memory cache data to a non-volatile flash device for permanent preservation.
- Typically, the Cache-to-Flash module comes with a flash module and a battery module which provides power to copy data from the cache memory to flash module in the event of a power failure.
- Cache-to-Flash technology will first flush CPU cache to memory, then flush memory to the flash module to maintain the utmost data consistency. It leverages the strength of both the BIOS and CPU to quickly backup memory data to the flash module.

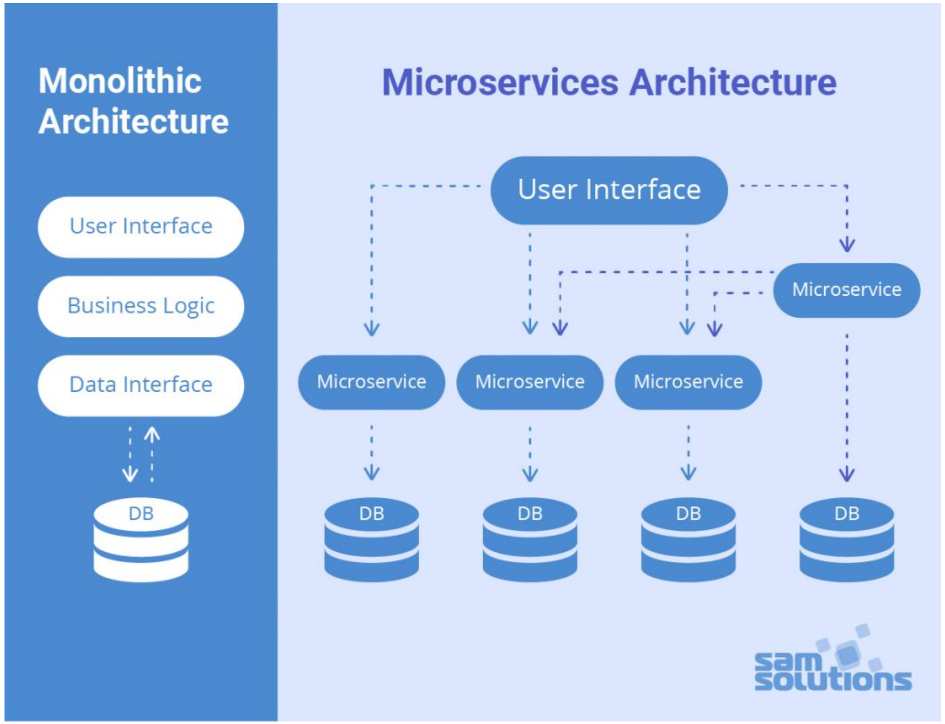
2. Bảo toàn dữ liệu trong cache khi mất nguồn điện

When the power is restored, the BIOS will check the status of the C2F flag during the Cache-to-Flash recovery phase. If the C2F flag is ON, the I/O cache data will be restored from the flash module and then resume normal startup. If the C2F flag is OFF, the normal startup process continues.



3. Caching in monolithic and microservices

a, Caching in monolithic is less effective ?



3. Caching between monolithic and microservices

b, Some architectural patterns for caching microservices

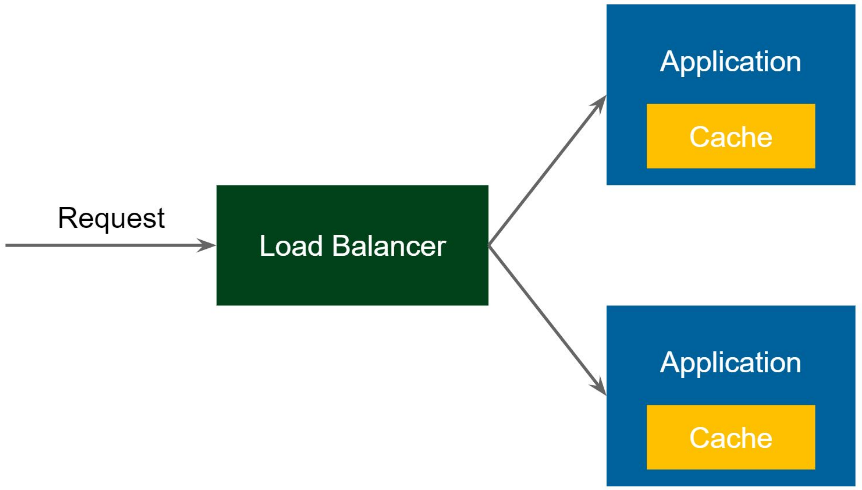
- Request comes in to the Load Balancer
- Load Balancer forwards the request to

one of the Application services

- Application receives the request and

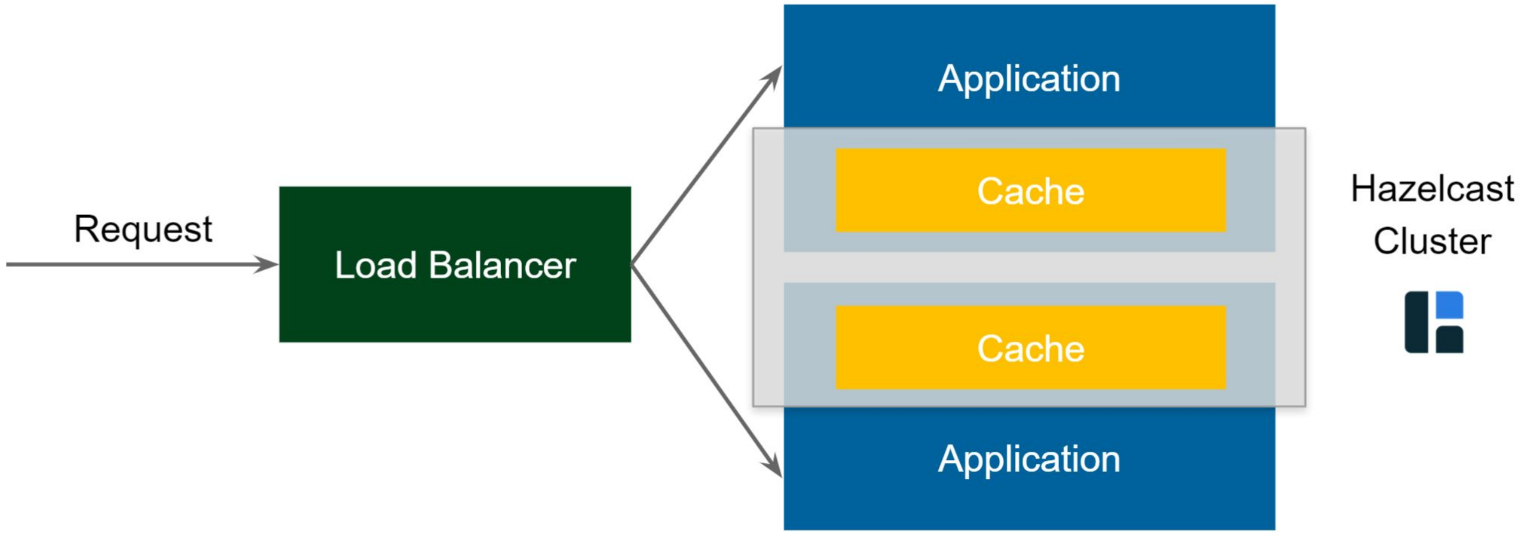
checks if the same request was already

executed (and stored in cache)



Embedded Cache

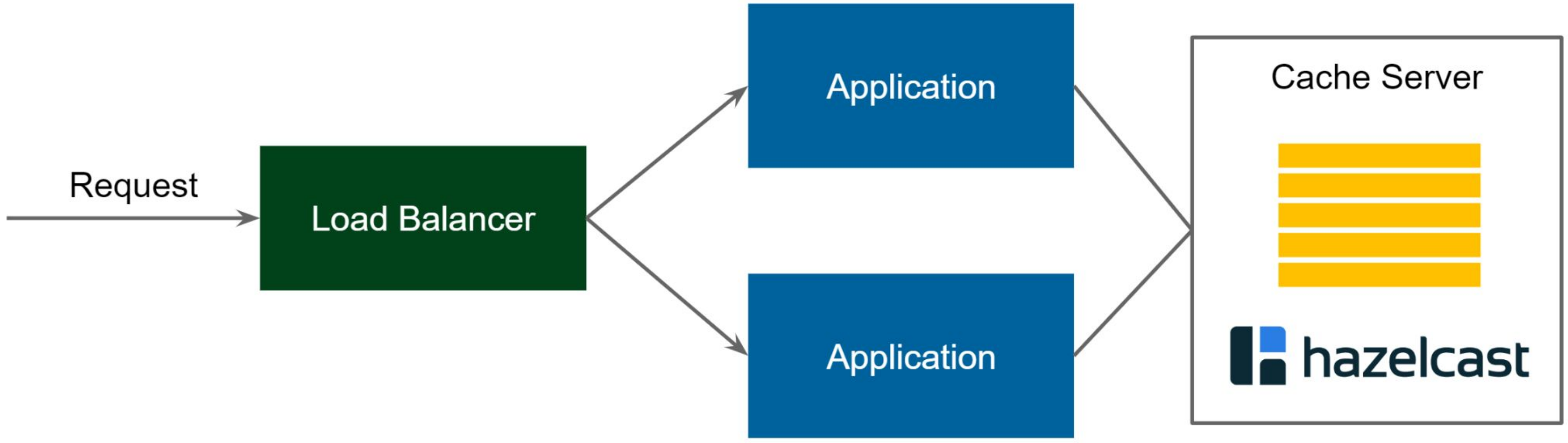
b, Some architectural patterns for caching microservices



Embedded Distributed Cache

b, Some architectural patterns for caching microservices

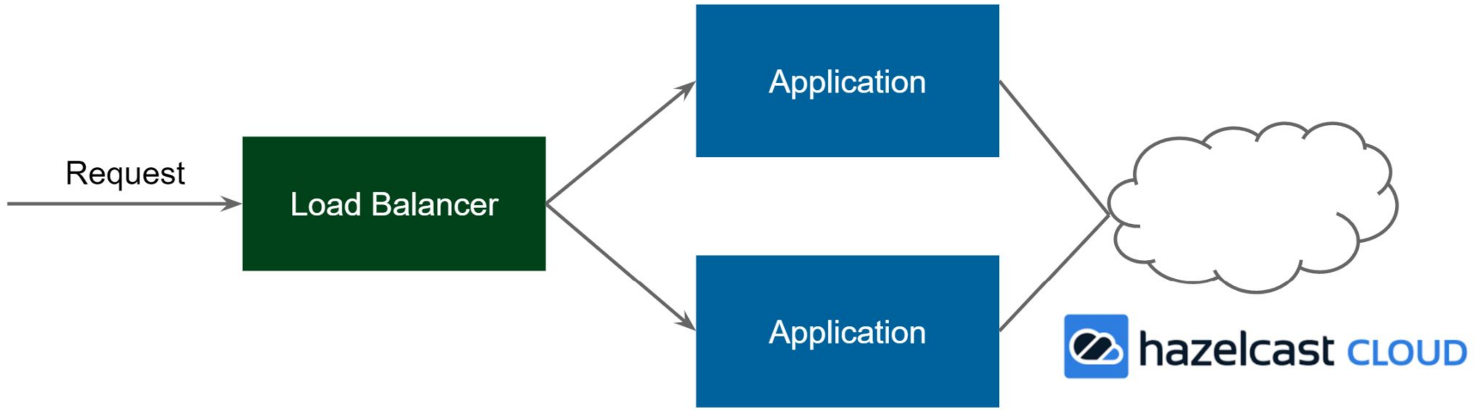
- Request comes into the Load Balancer and is forwarded to one of the Application services
- Application uses cache client to connect to Cache Server
- If there is no value found, then perform the usual business logic, cache the value, and return the response



Client-Server Cache

b, Some architectural patterns for caching microservices

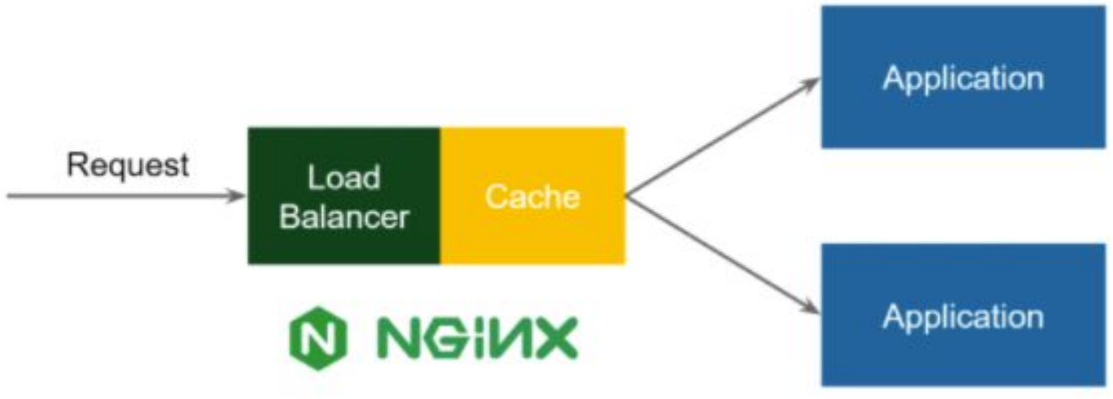
- Cloud is like Client-Server, with the difference being that the server part is moved outside of your organization and is managed by your cloud provider.



Cloud cache

b, Some architectural patterns for caching microservices

- Request comes in to the Load Balancer
- Load Balancer checks if such a request is already cached
- If yes, then the response is sent back and the Request is not forwarded to the Application



Reverse Proxy Cache

4. Typical Cache Libraries

- Introduce cache libraries:
  - Guava
  - JCS (Java Caching System)
  - Memcached
  - Hazelcast
  - Ehcache
- Demo Hazelcast in Spring

Guava

Guava is Google core libraries for Java, first release in 2009

- Simple local cache
- Eviction: size-based, timed, reference-based
- Statistics (hitRate, averageLoadPenalty, evictionCount)
- Used by Caffeine

Guava

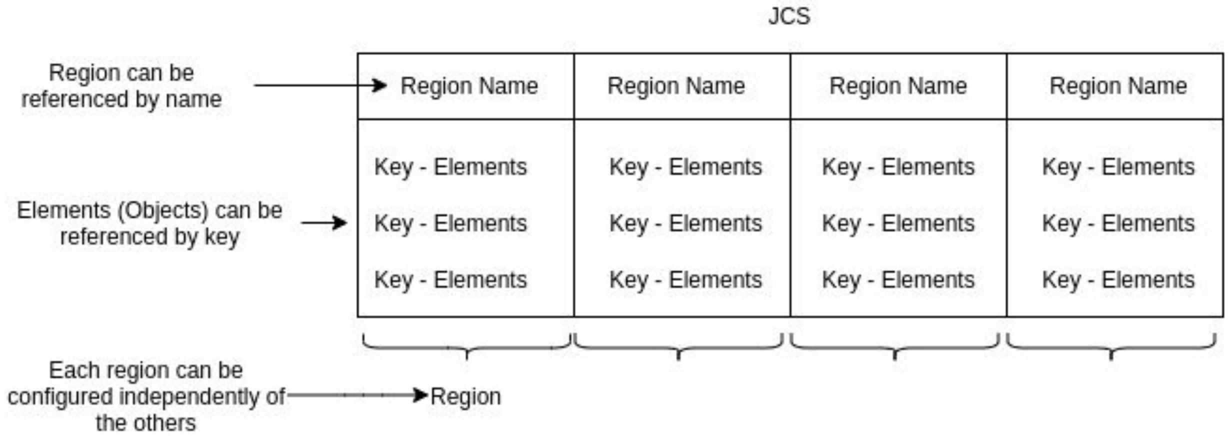
Guava Cache vs Caffeine:

- Guava has worse performance
- Guava has race condition problem

→ Recommend using Caffeine instead

JCS (Java Caching System)

- Apache open-source distributed caching system written in Java
- Core concepts: **elements**, **regions** and **auxiliaries**
- Composite cache


JCS
Region can be referenced by name
Region Name
Region Name
Region Name
Region Name
Key - Elements
Elements (Objects) can be referenced by key
Each region can be configured independently of the others
Region

JCS: Auxiliaries

Each region must have one (and only one) **MEMORY** auxiliary and can have any number of **OTHER** auxiliaries

- **Memory** auxiliary: cache policies: LRU (default), LHM LRU, MRU, FIFO, ARC
- **Disk** auxiliary: store data on disk when memory threshold is reached
- **Lateral** auxiliary: distribute cached data to multiple servers
- **Remote** auxiliary: rather than having each node connect to every other node, you can use the remote cache server as the connection point

Memcached

- Open-source **distributed** memory object caching system
- Fast, easy to use
- Developed by Brad Fitzpatrick for LiveJournal in 2003
- Used by Facebook, Youtube and Twitter [Ref]

Memcached: Pros and Cons

Pros:

- Use allocation mechanism called Slab → Reduce data fragmentation
- **Multithreaded** → Easily to scale vertically
- Support most popular languages

Cons:

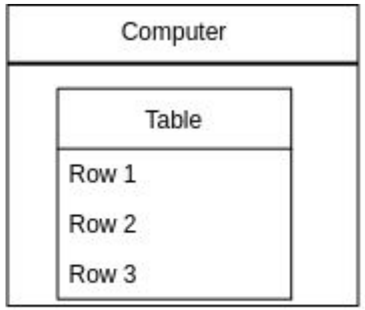
- Store objects indexed by a string key
- Limited to LRU eviction policy
- Backup and restore not supported

Memcached: Scaling at Facebook

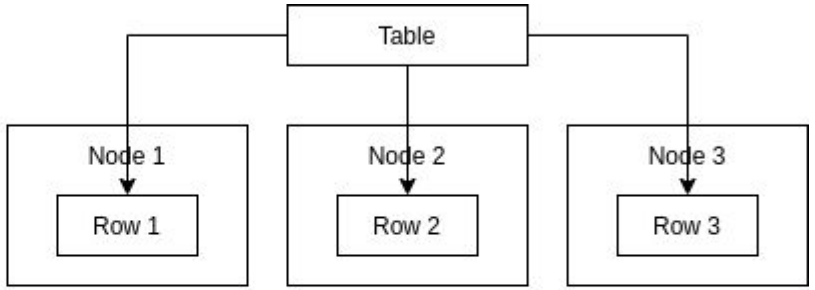
- Write: SQL update in DB and delete in cache
- Handle failures of node: Facebook dedicates a small set of machines, named Gutter, to take over the responsibilities of a few failed servers
- Replication: rely on MySQL’s replication mechanism

Hazelcast

Open source **in-memory data grid** written in Java, first release in 2009



Traditional DB



Data Grid

Hazelcast

Hazelcast: Features

- Management Center
- SQL querying, data structures
- Cache Hot Restart Store
- Security Suite
- ...

Ehcache

- Most widely-used open-source Java cache
- Robust, proven, full-featured, integrates with other popular libraries and frameworks
- Popularly used as Hibernate second-level cache
- Currently supported by Terracotta
- Used by LinkedIn

Ehcache: Features

- Eviction policy: LRU, LFU, FIFO
- Ehcache default is local cache
- Deploy Terracotta Server for clustering capabilities
- RestartStore: fast restartability and options for cache persistence
- Replication using RMI

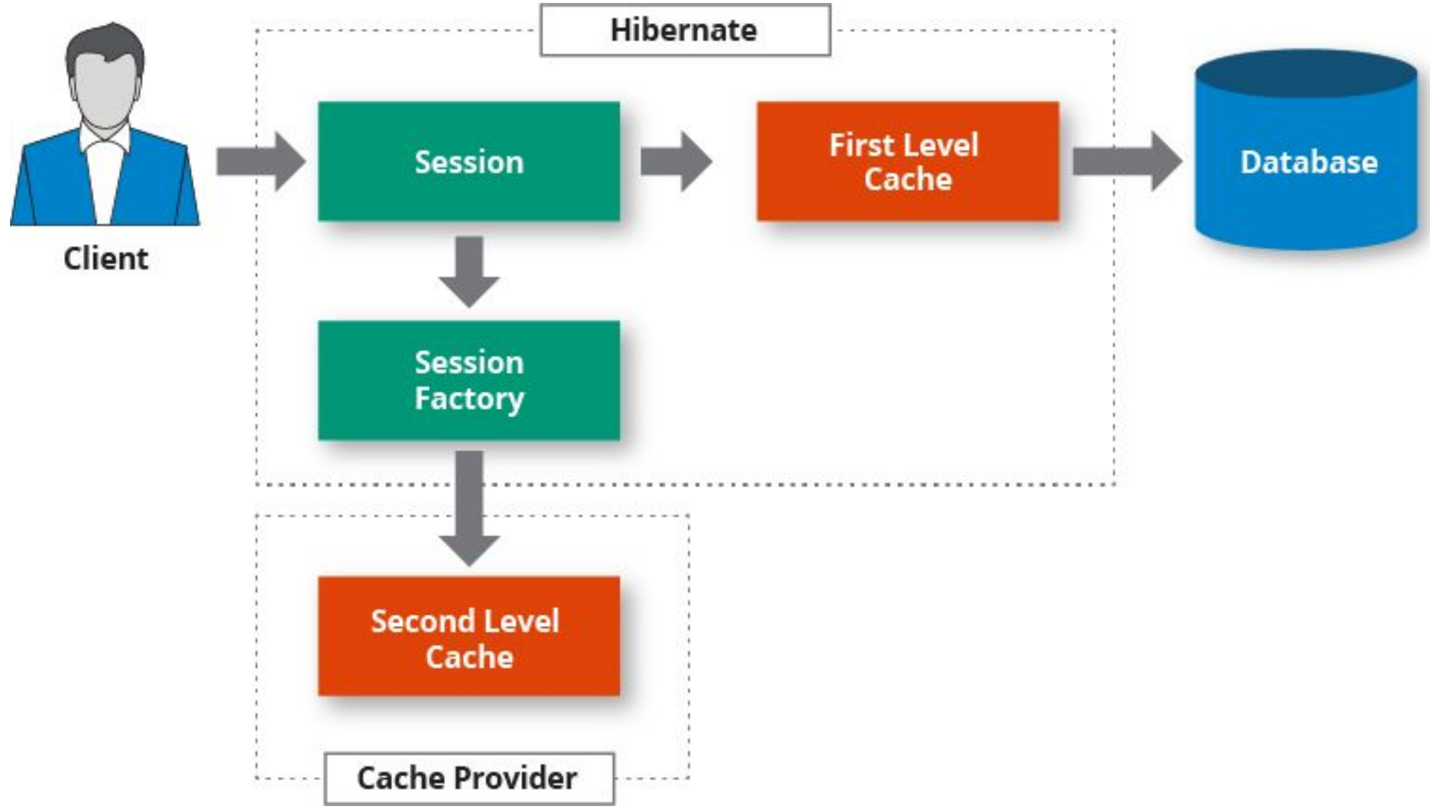
Summary

	JCS	Guava	Memcached	Hazelcast	Ehcache
Support data structures	❌	❌	❌	✅	❌
Multi-key operation	❌	✅	✅ (Only get)	✅	✅
Distributed servers	✅	❌	✅	✅	✅
High availability	✅	❌	❌	✅	✅
Data persistence	✅	❌	❌	✅	✅

Demo cache libraries in Spring

- Hazelcast

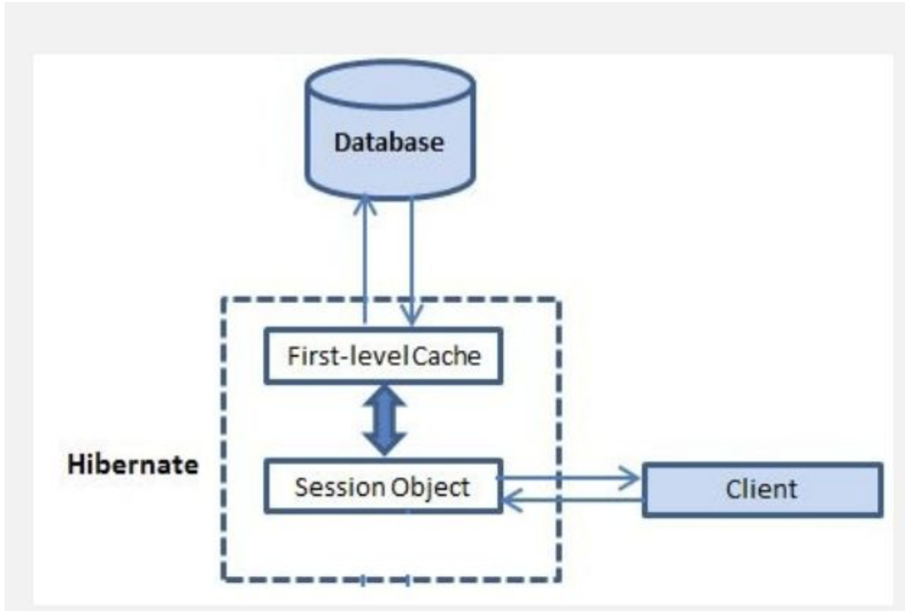
5. Caching in Hibernate



5. Caching in hibernate

First level cache

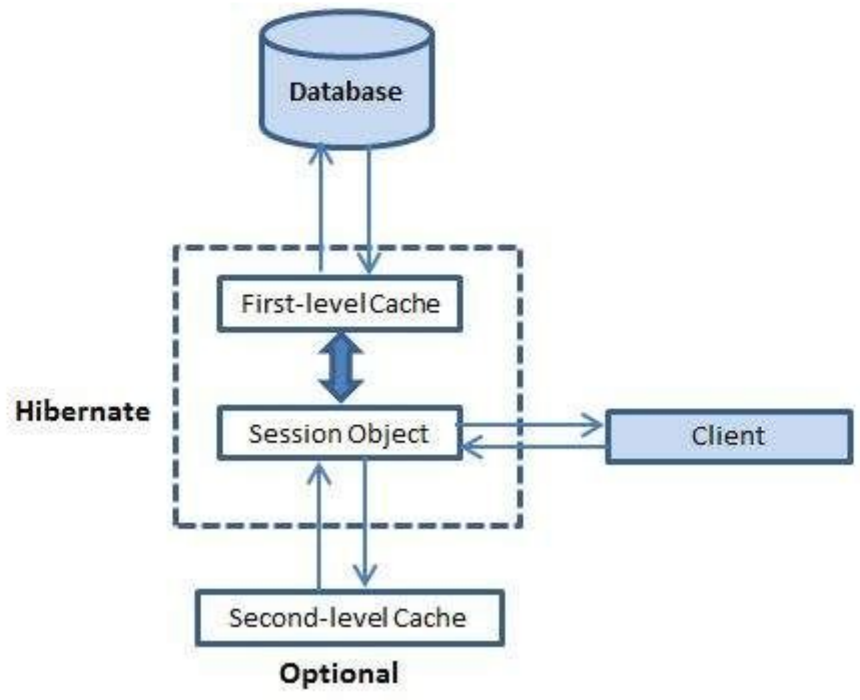
- First level cache is associated with "session" object and other session objects in application can not see it.
- The scope of cache objects is of session. Once session is closed,cached objects are gone forever.
- First level cache is enabled by default and you can not disable it.



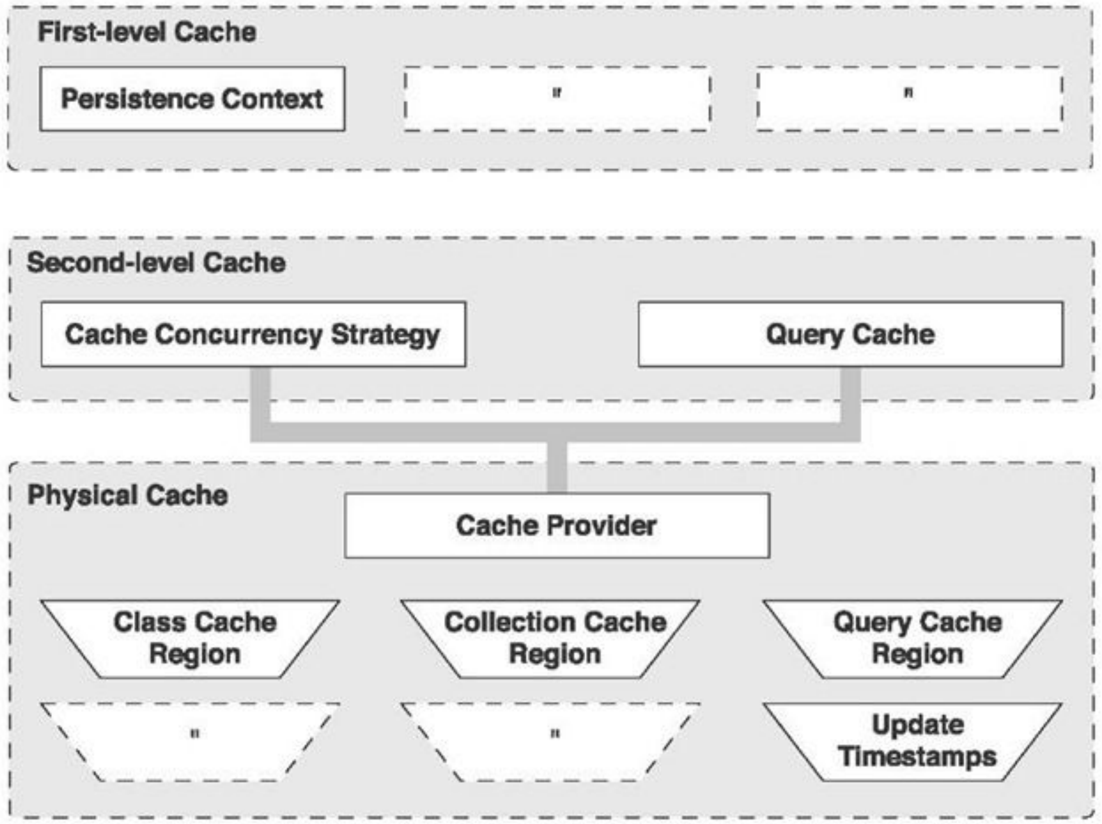
5. Caching in hibernate

Second level cache

- Hibernate second level cache uses a common cache for all the session object of a session factory.



5. Caching in hibernate



5. Caching in hibernate

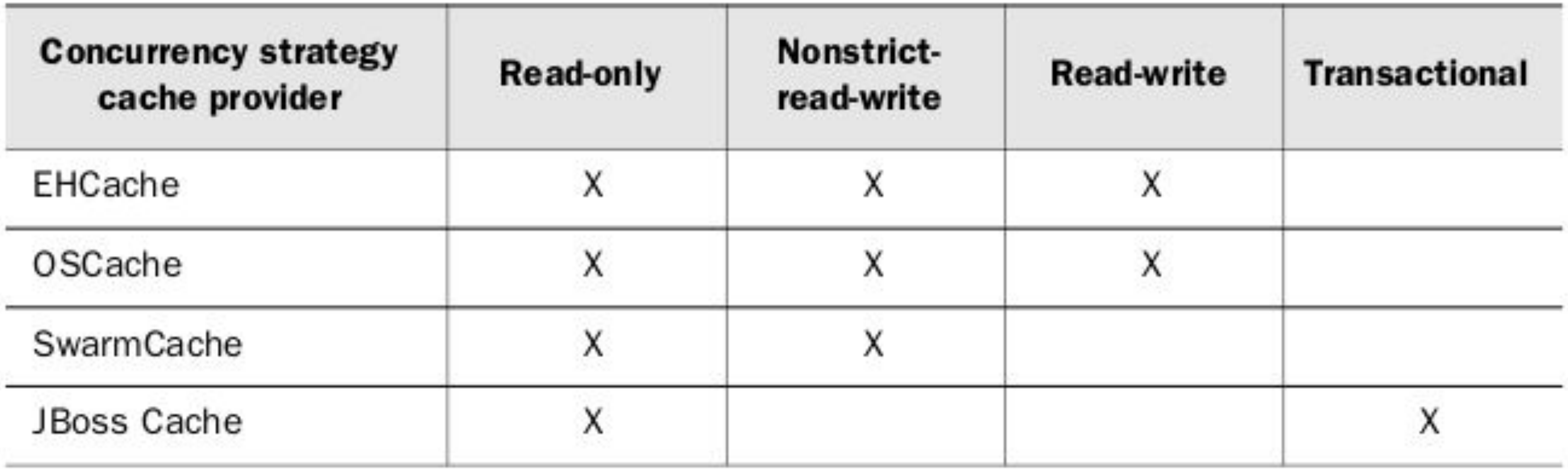
- 4 built-in concurrency strategies
  - **Read-only**
    - Simple, performs well, and is safe to use in a clustered environment.
    - Useful for data that is read frequently but never updated.
  - **Read-write**
    - Suitable for regular update data, more overhead than read-only
    - Not provide serializable transaction isolation (phantom reads can occur), but guarantees strong consistency by using ‘soft’ locks
  - **Nonstrict read-write**
    - Suitable for rarely update data, less overhead than read-write
    - do not need strict transaction isolation
  - **Transactional**
    - It supports transactional cache providers such as JBoss TreeCache and Ehcache
    - only used in the JTA environment.

5. Caching in hibernate

4 Cache providers

- **EHCache**
  - Fast, lightweight, easy to use, supports memory-based and disk-based caching
  - Not support clustering
- **OSCache**
  - Powerful, flexible, supports memory- based and disk-based caching.
  - Provides basic support for clustering via either JavaGroups or JMS.
- **SwarmCache**
  - is a cluster-based caching based on JGroups.
  - Appropriate for more read than write
- **JBoss TreeCache**
  - is a powerful replicated and transactional cache.
  - useful when we need a true transaction-capable caching architecture .

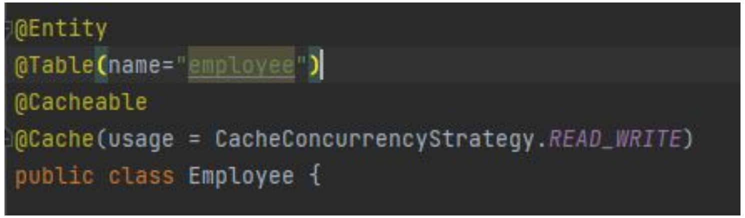
5. Caching in hibernate



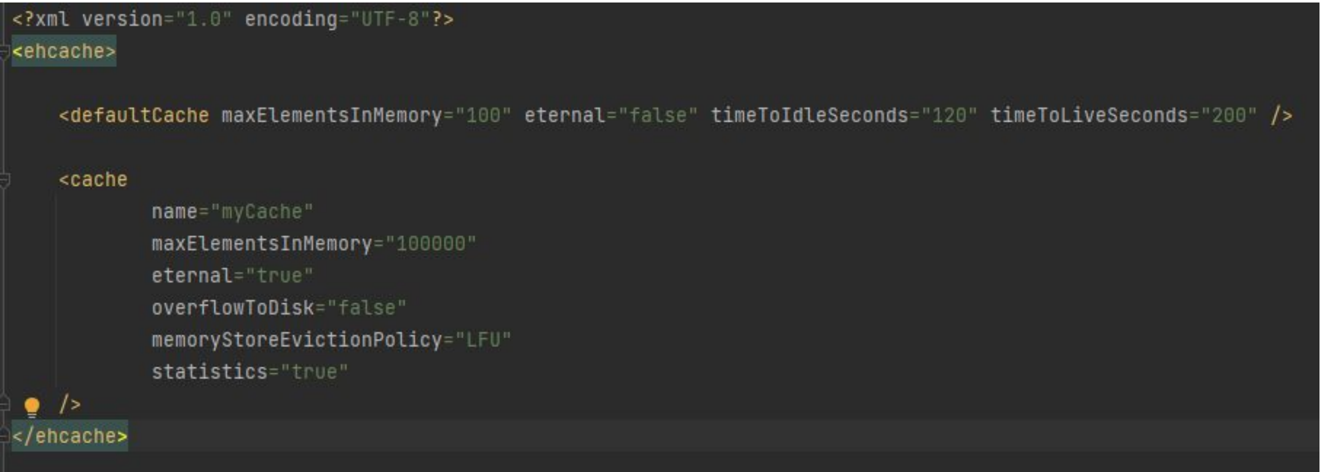
Concurrency strategy cache provider	Read-only	Nonstrict-read-write	Read-write	Transactional
EHCache	X	X	X	
OSCache	X	X	X	
SwarmCache	X	X		
JBoss Cache	X			X

5. Caching in hibernate

- How to set up Hibernate second level cache:
  - Choose cache provider
  - Enable second level cache and configure it
  - Make entities cacheable



```java
@Entity
@Table(name="employee")
@Cacheable
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Employee {
```



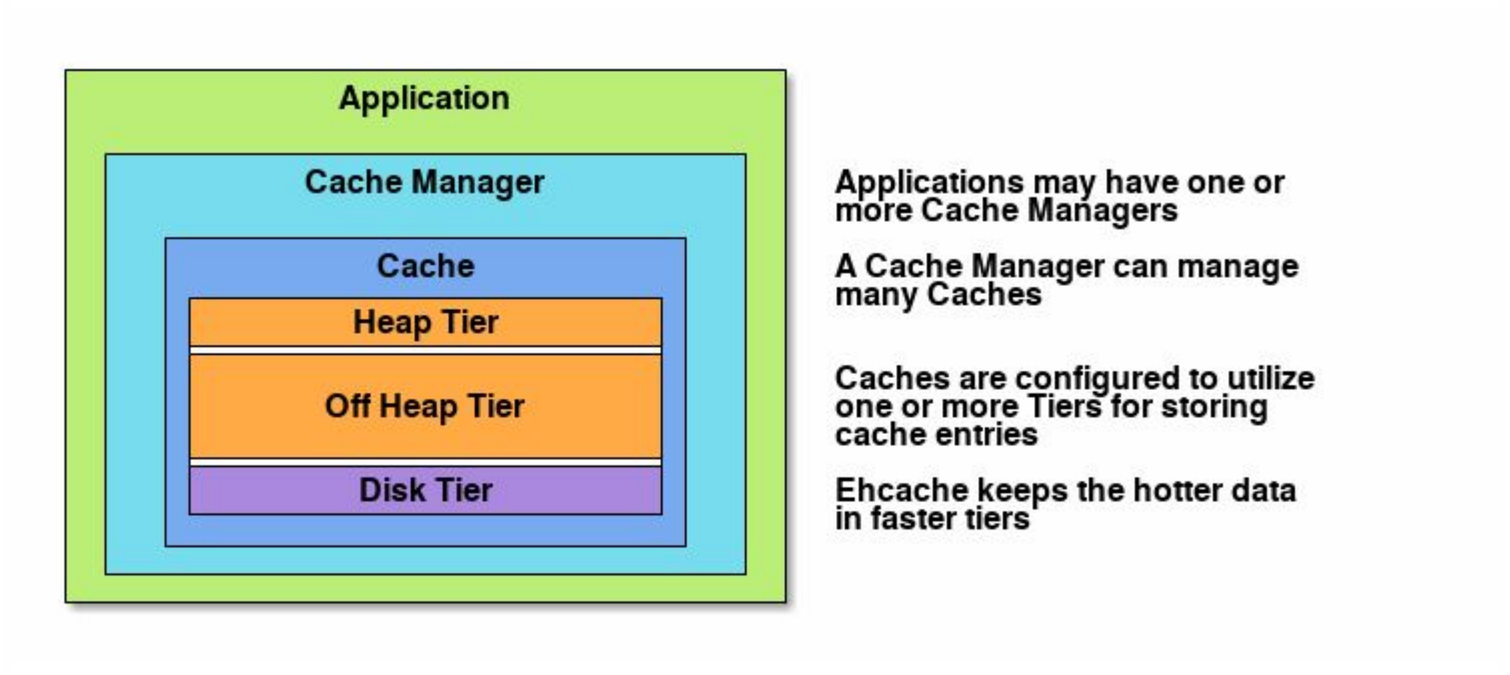
```xml
<?xml version="1.0" encoding="UTF-8"?>
<ehcache>

    <defaultCache maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120" timeToLiveSeconds="200" />

    <cache
            name="myCache"
            maxElementsInMemory="100000"
            eternal="true"
            overflowToDisk="false"
            memoryStoreEvictionPolicy="LFU"
            statistics="true"
    />
</ehcache>
```

5. Caching in hibernate.

- Ehcache



Applications may have one or more Cache Managers

A Cache Manager can manage many Caches

Caches are configured to utilize one or more Tiers for storing cache entries

Ehcache keeps the hotter data in faster tiers

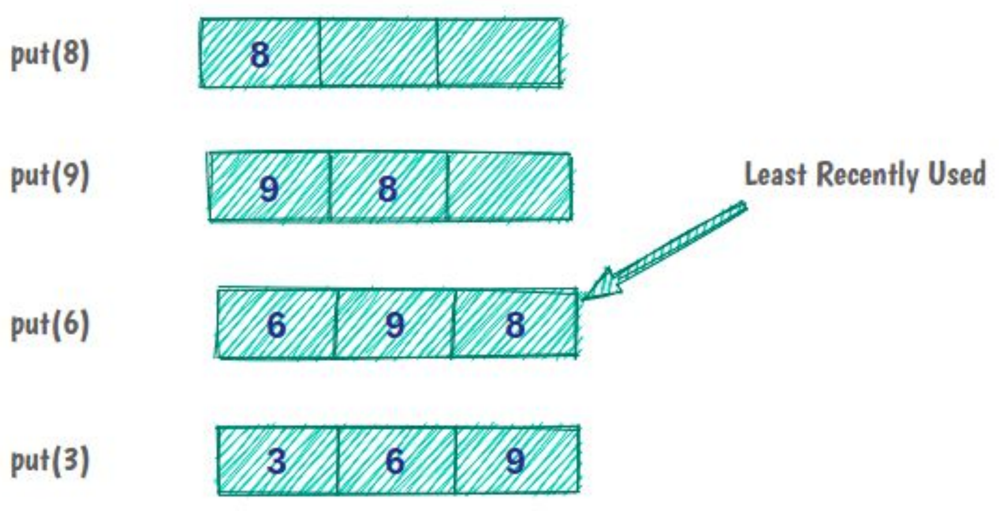
5. Caching in hibernate.

- What happens when maxEntriesLocalHeap is reached? Are the oldest items expired when new ones come in?

  When the maximum number of elements in memory is reached, the Least Recently Used (LRU) element is removed. “Used” in this case means inserted with a put or accessed with a get. If the cache is not configured with a persistence strategy, the LRU element is evicted. If the cache is configured for “localTempSwap”, the LRU element is flushed asynchronously to the DiskStore.

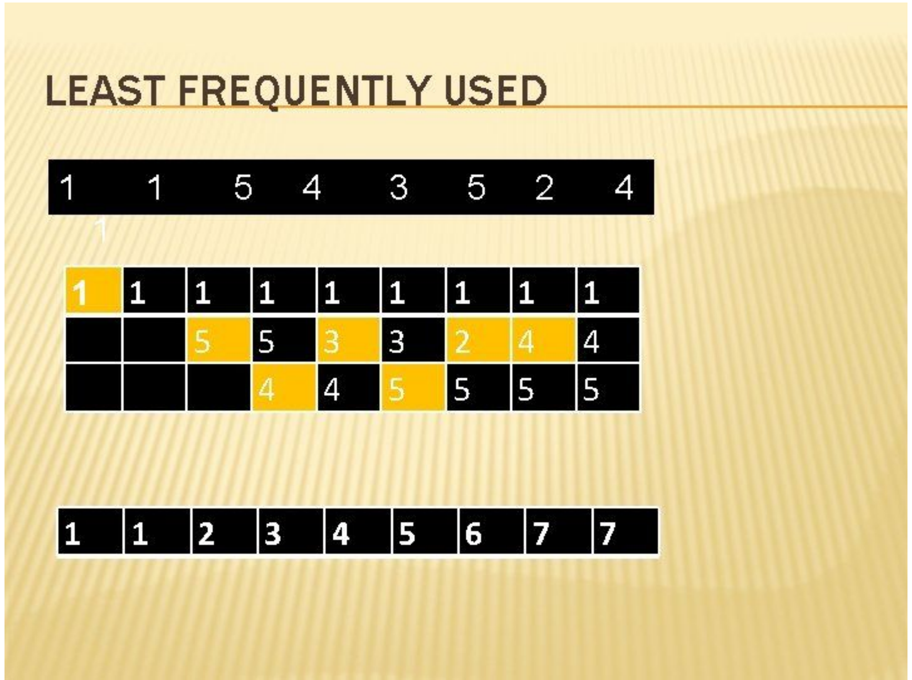
5. Caching in hibernate

Least Recently Used (LRU) *Default**—The eldest element, is the Least Recently Used (LRU). The last used timestamp is updated when an element is put into the cache or an element is retrieved from the cache with a get call.



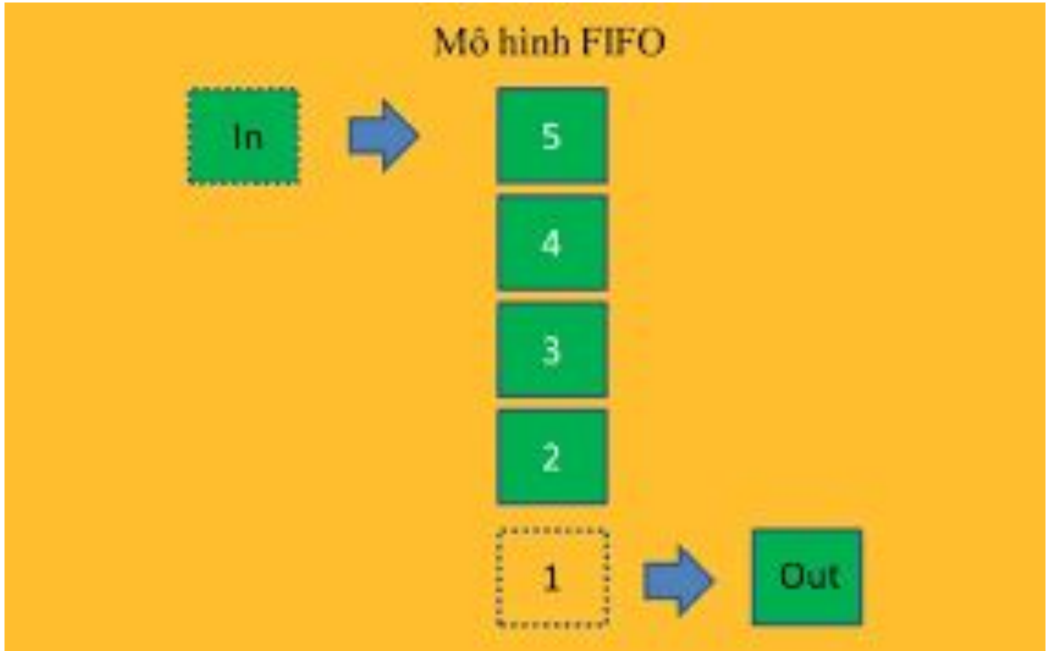
5. Caching in hibernate

Least Frequently Used (LFU)—For each get call on the element the number of hits is updated. When a put call is made for a new element (and assuming that the max limit is reached for the MemoryStore) the element with least number of hits, the Less Frequently Used element, is evicted.



5. Caching in hibernate

First In First Out (FIFO)—Elements are evicted in the same order as they come in. When a put call is made for a new element (and assuming that the max limit is reached for the MemoryStore) the element that was placed first (First-In) in the store is the candidate for eviction (First-Out).



5. Caching in hibernate

DEMO

References

- 7 Reasons Why Hazelcast Is The Best Caching Technology For You
- Memcached Challenges
- https://hibernate.atlassian.net/browse/HHH-12441
- https://www.ehcache.org/documentation/2.8/faq.html
- https://www.ehcache.org/documentation/2.8/get-started/storage-options.html
- https://www.ehcache.org/ehcache.xml
- https://www.ehcache.org/documentation/2.8/apis/explicitlocking.html
- https://www.ehcache.org/documentation/2.8/apis/transactions.html