VIETNAM WEB SUMMIT

Caching – Methodology & Strategies

By Vu Van Tiep

Manager, Head Of Web Platform - VNG

Skype: tiepvv

Agenda

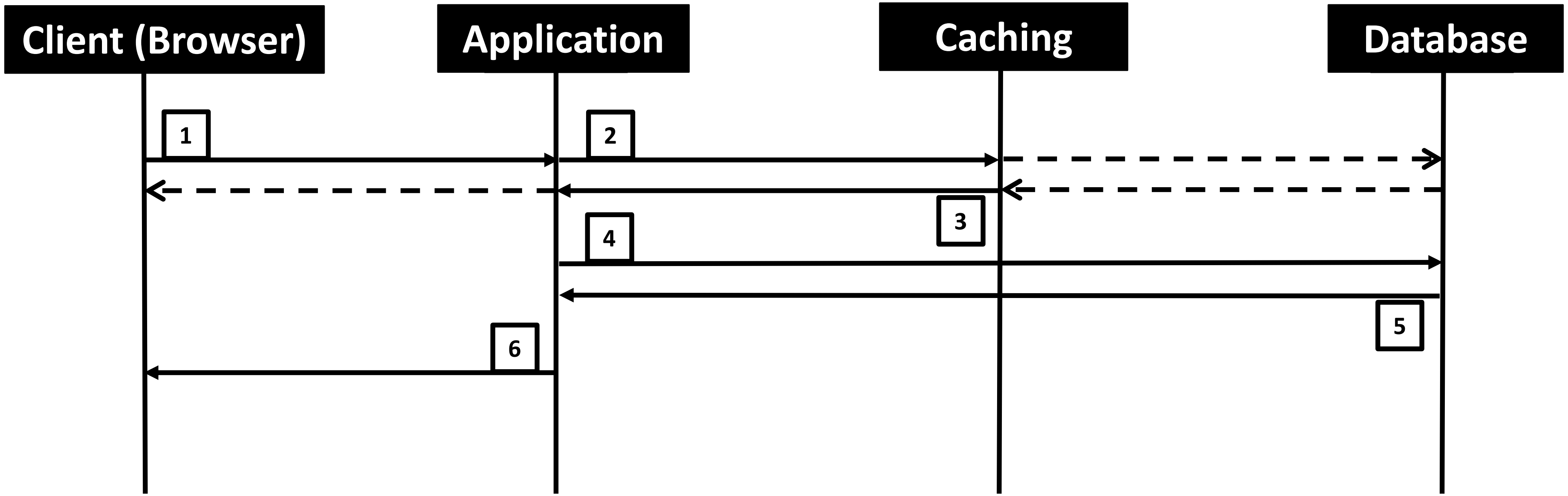
- What
- Cache Layer
- Cache Application/Service
  - Methodology
  - Strategies
  - Performance?
- Q&A

Cache – What?

- Is a hardware or software component that stores data so future requests for that data can be **served faster**;
- The data stored in a cache might be the result of an **earlier computation**, or the **duplicate** of data stored elsewhere
- A *cache* ***hit*** occurs when the requested data can be found in a cache, while a *cache* ***miss*** occurs when it cannot.
- Cache **hits** are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests can be served from the cache, the faster the system performs.

https://en.wikipedia.org/wiki/Cache_(computing)

Cache Layer



- Browser
- Proxy
- CDN

- in-memory
- Server Caching

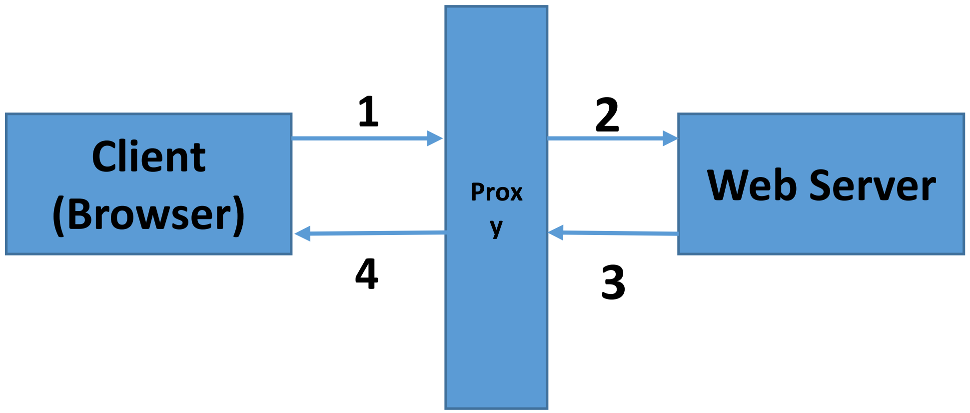
- Distributed in memory key/value
- Service:
    • Memcache
    • Membase
    • Redis
    • …

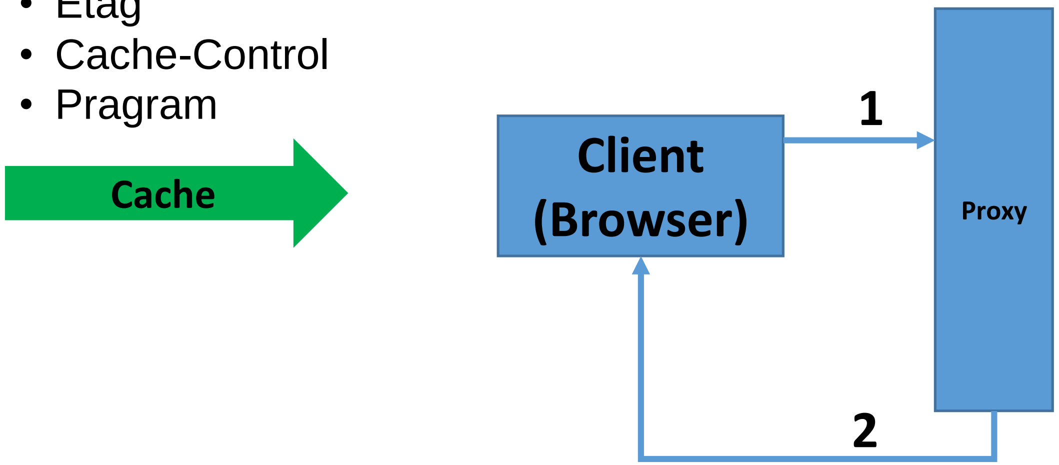
- Database caching
    • Entity In memory
- DB:
    • MySQL
    • MongoDB
    • Casandra, LevelDB
    • …

Http Layer Caching – Browser & Proxy

Response Header

- Expires
- Last-Modified
- Etag
- Cache-Control
- Pragram





Cache Application/Service

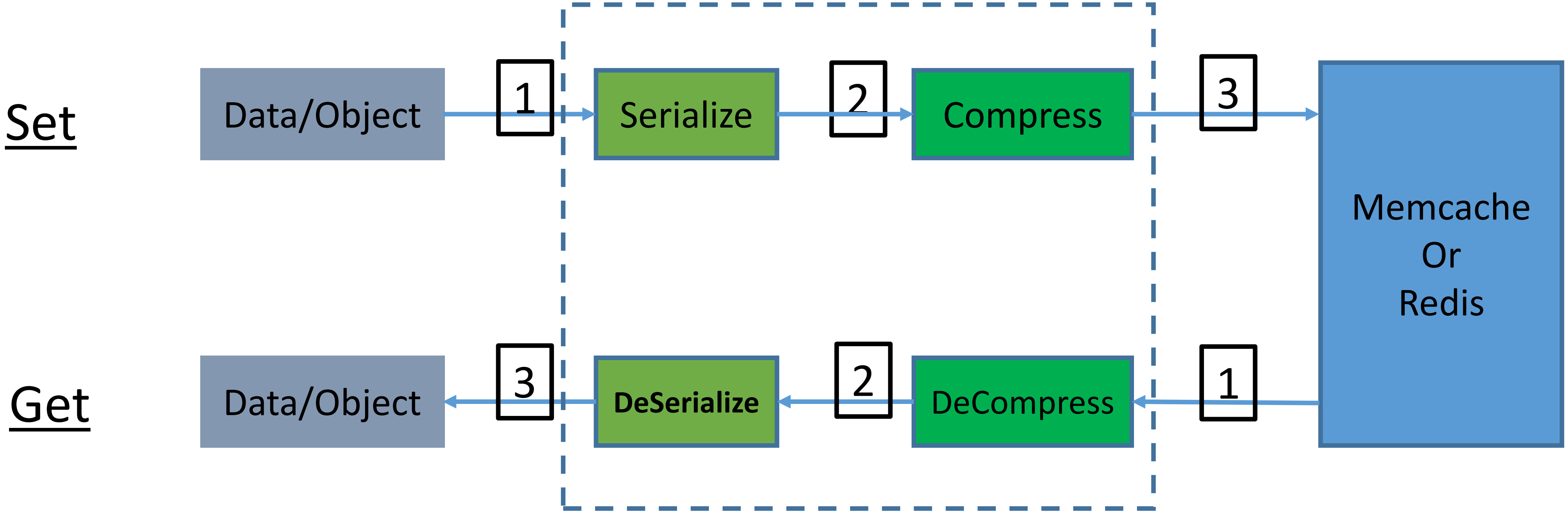
Latency Numbers

L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	
Mutex lock/unlock	100 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	10,000 ns	
Send 2K bytes over 1 Gbps network	**20,000 ns**	**(1)**
Read 1 MB sequentially from memory	**250,000 ns**	**(2)**
Round trip within same datacenter	500,000 ns	
Disk seek	**10,000,000 ns**	**1 + 2 < (3)**
Read 1 MB sequentially from network	10,000,000 ns	
Read 1 MB sequentially from disk	30,000,000 ns	
Send packet CA->Netherlands->CA	150,000,000 ns	

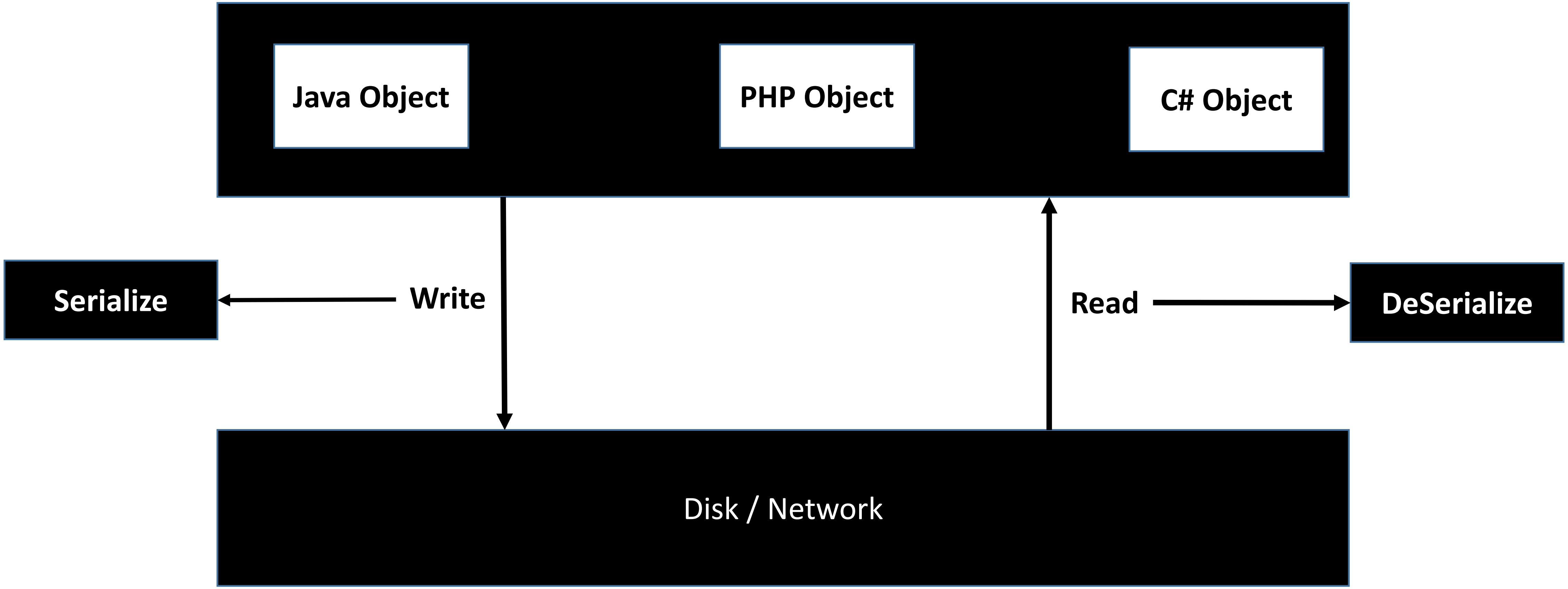
Note: 1 ns = 10^-9 seconds

From Jeff Dean - google.com

Cache: Methodology



Serialize


Java Object
PHP Object
C# Object
Serialize
Write
Read
DeSerialize
Disk / Network

Performance Comparison

Binary serialize take less space

Text Serialize	Framework	Binary Serialize

Json – size: 167 bytes, se: 6ms, de: 3ms
`{ "ProfileInfo": { "userId": 1, "registerDate": 1476200214431, "userName": "tiepvv", "birthDate": 1476200214431, "displayName": "Tiep Vu" } }`
 	
XML – 235 bytes
`<?xml version="1.0" encoding="UTF-8" ?> <root> <username>tiepvv</username> <registerDate>1476198686</registerDate> <userId>1</userId> <birthDate>1476198686</birthDate> <displayName>Tiep Vu</displayName> </root>`
	XML, Json
	ProtoBuf, Thrift
	Message Pack
	Apache Arvo
	…
		Thrift – size: 64 bytes, se: 53ms, de:1ms
		`namespace java vietnam.websummit.thrift`
		`struct ProfileInfo { 1:i32 userId, 2:string username, 3:string displayName, 4:i32 birthDate, 5:i32 registerDate }`
		
		ProtoBuf – size: 31 bytes, se: 34ms, de: 4ms
		`syntax = "proto3";`
		`package vietnam.websummit.proto;`
		`option java_package = "vietnam.websummit.proto";`
		`message ProfileInfo { int32 userId = 1; string username = 2; string displayName = 3; int32 birthDate = 4; int32 registerDate = 5; }`

Compress/Decompress

Name	Ratio	C.speed	D.speed
		MB/s	MB/s
zstd 0.8.2 -1	**2.877**	**330**	**940**
zlib 1.2.8 -1	**2.73**	**95**	**360**
brotli 0.4 -0	2.708	320	375
QuickLZ 1.5	2.237	510	605
LZO 2.09	2.106	610	870
LZ4 r131	2.101	620	3100
Snappy 1.1.3	2.091	480	1600
LZF 3.6	2.077	375	790

As a reference, several fast compression algorithms were tested and compared on a Core i7-3930K CPU @ 4.5GHz, using lzbench, an open-source in-memory benchmark by @inikep compiled with GCC 5.4.0, with the Silesia compression corpus.

https://github.com/facebook/zstd

Caching Strategies

Cache Strategies

1. Optimize Data Size
2. Hot Object
3. In-Process
4. Lazy load
5. Distributed

Optimize Data Size

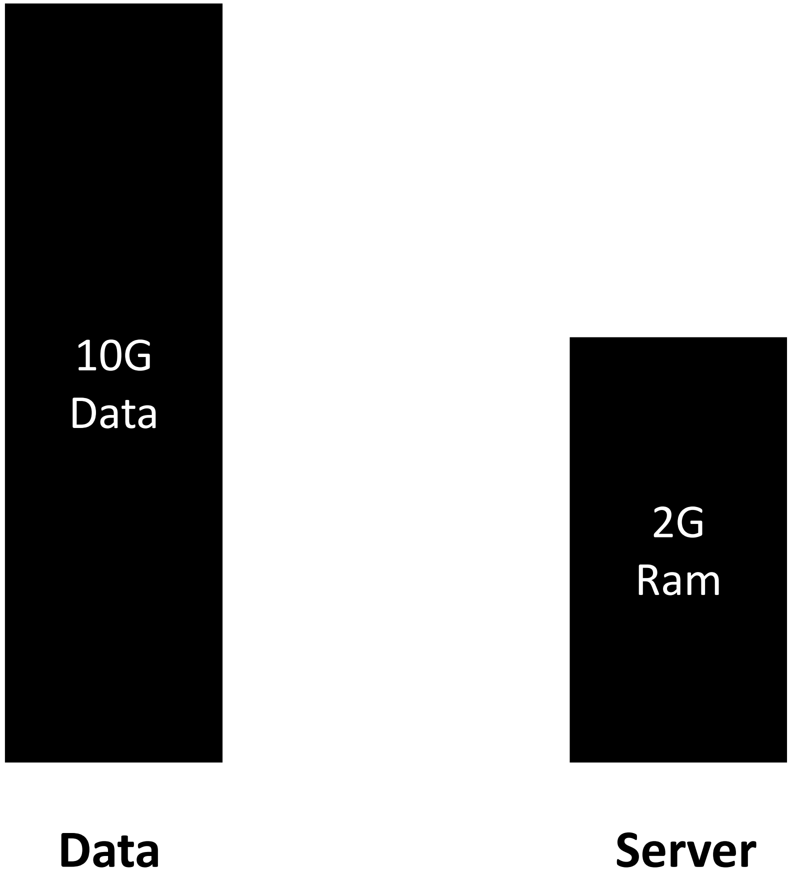
Serialize	
Method	**Size (smaller is better)**
Thrift — TCompactProtocol	278 (not bad)
Thrift — TBinaryProtocol	460
Protocol Buffers	250 (winner!)
RMI	905
REST — JSON	559
REST — XML	836

High Level Goals:

- Transparent between multiple programming languages (PHP/Java/Cpp/…)
- Maintain Right balance between:
  - Efficiency (how much time/space?)
  - Availability of existing libraries

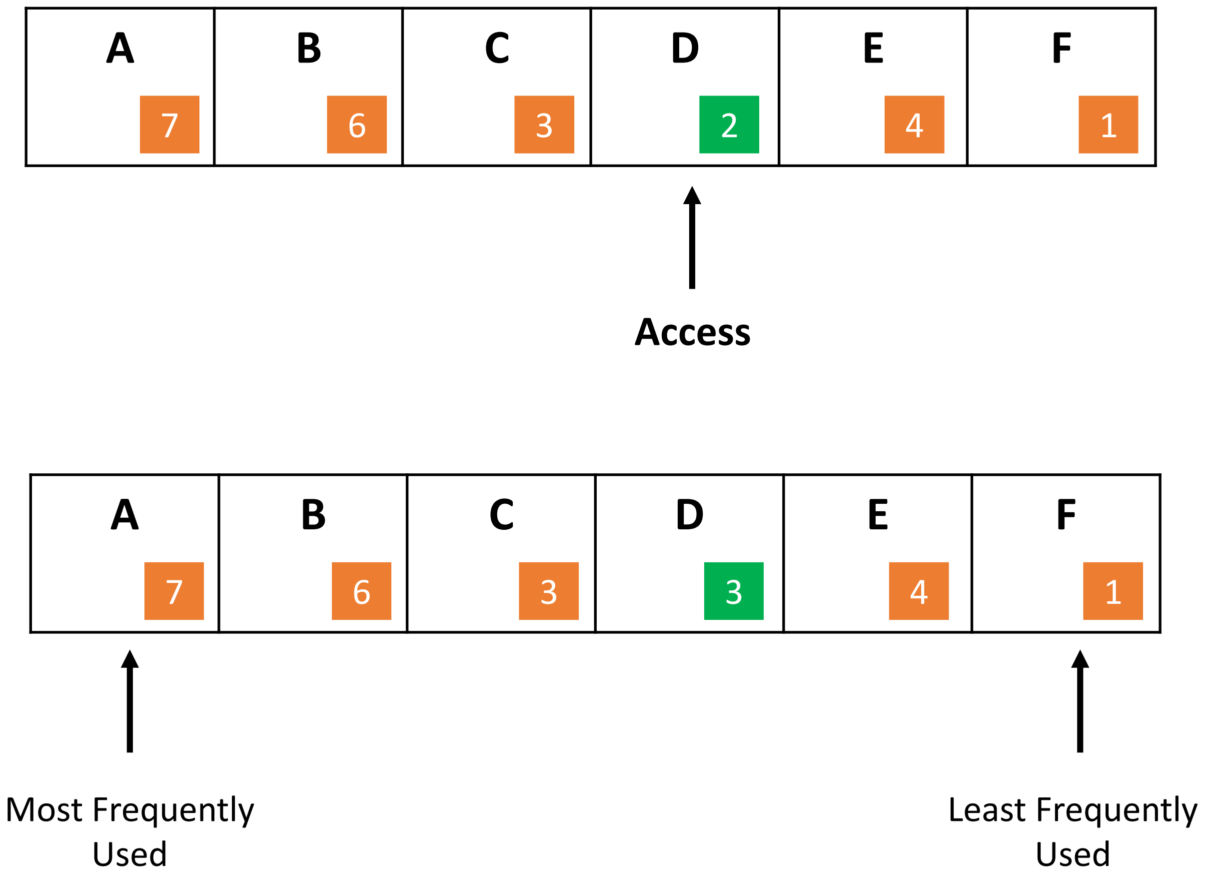
Compress			
Name	**Ratio**	**C.speed MB/s**	**D.speed MB/s**
zstd 0.8.2 -1	**2.877**	**330**	**940**
zlib 1.2.8 -1	**2.73**	**95**	**360**
brotli 0.4 -0	2.708	320	375
QuickLZ 1.5	2.237	510	605
LZO 2.09	2.106	610	870
LZ4 r131	2.101	620	3100
Snappy 1.1.3	2.091	480	1600
LZF 3.6	2.077	375	790

Hot Object => Eviction Algorithms

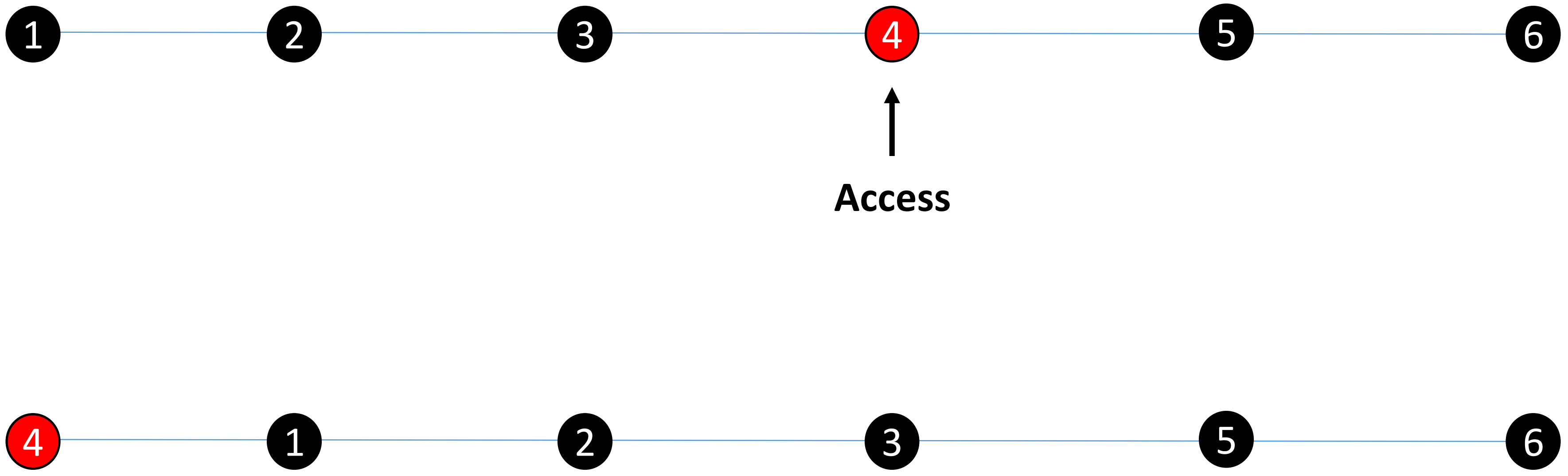


How to cache?

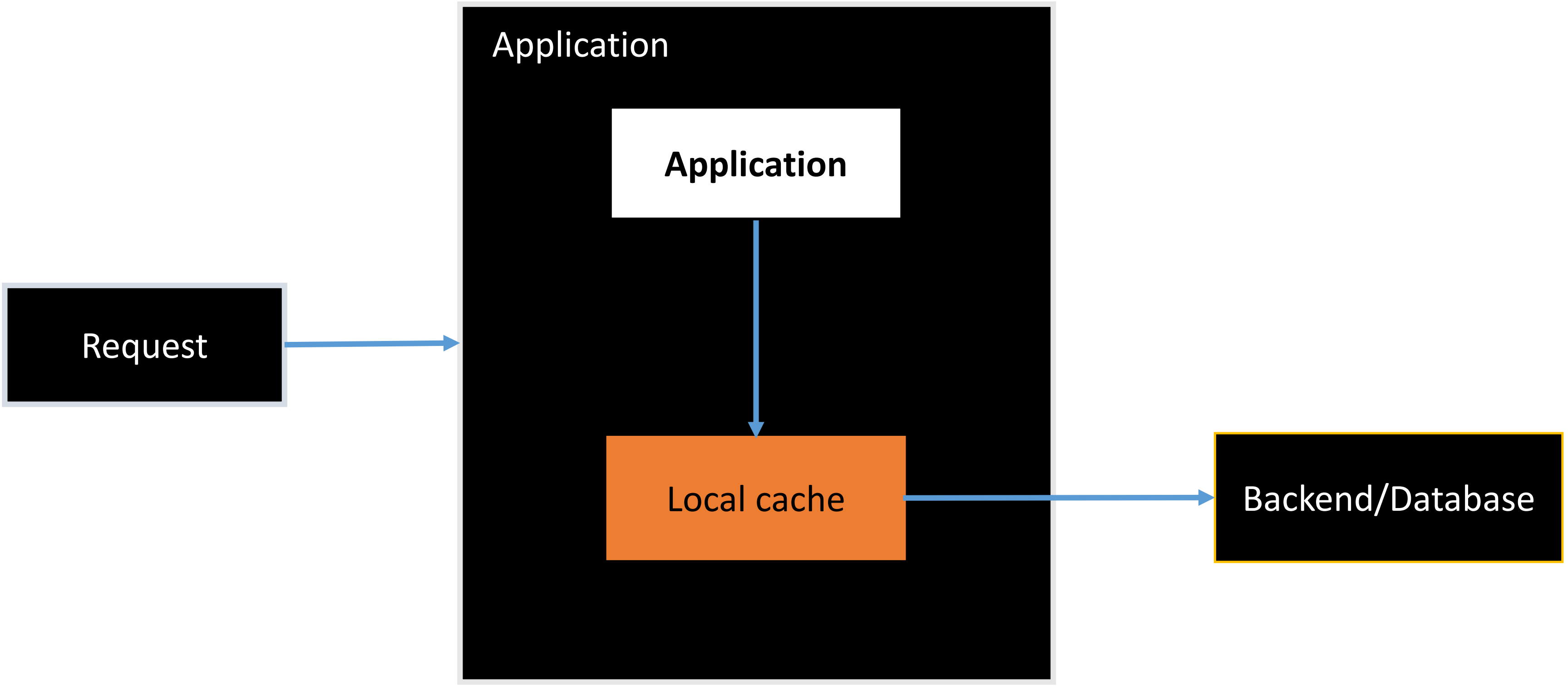
Eviction Algorithms: Least Frequently Used (LFU)



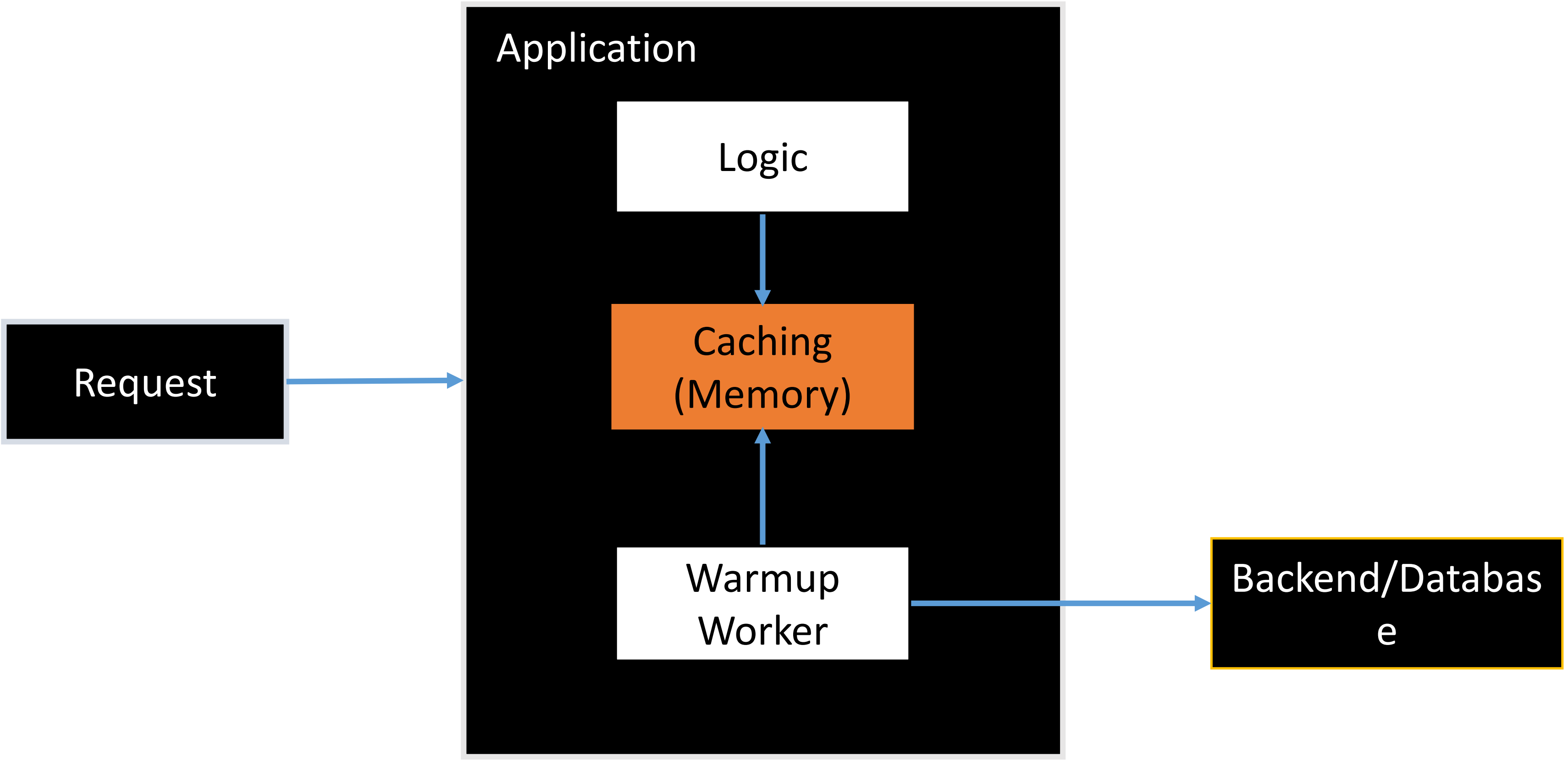
Eviction Algorithms: Least Recently Used (LRU)



In-Process

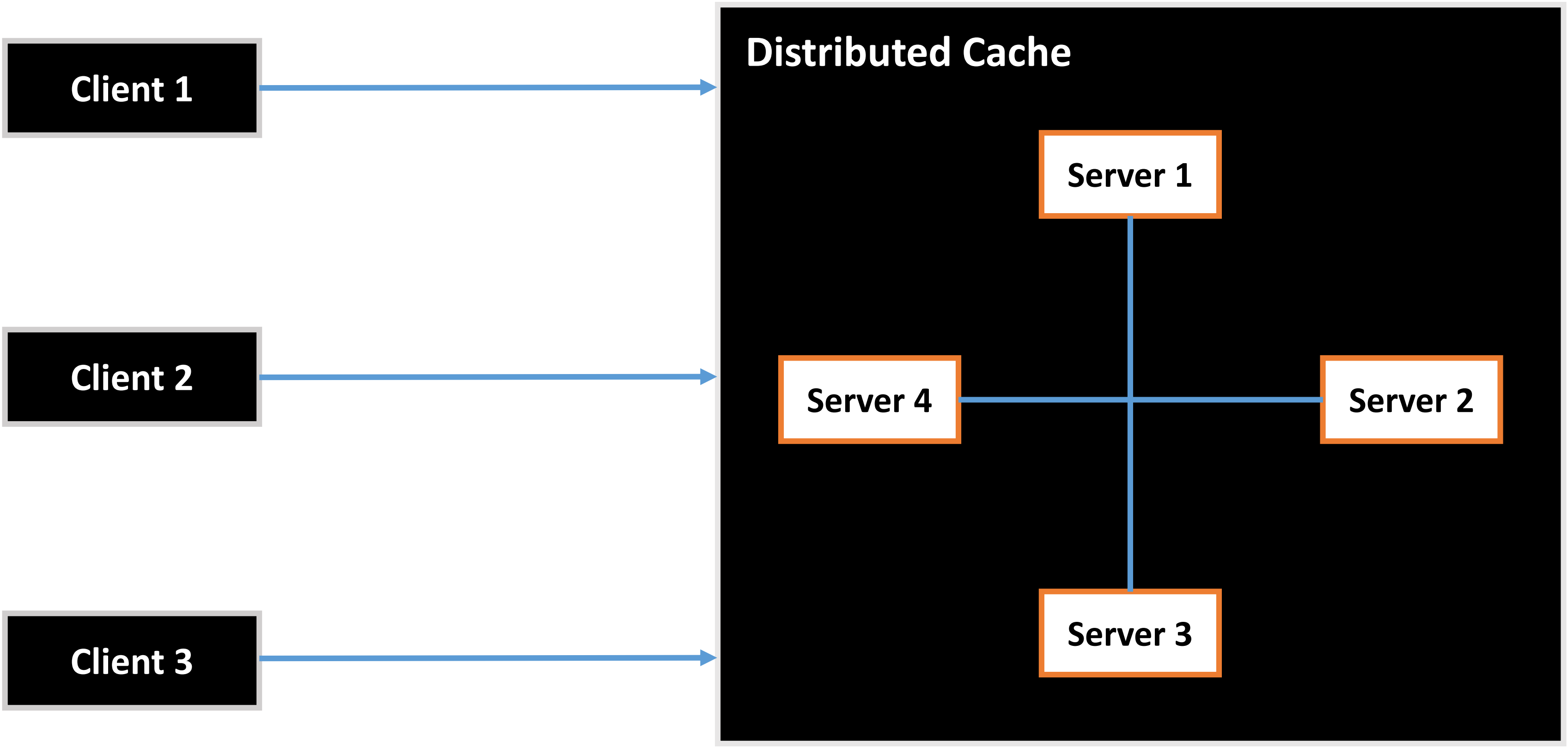

Application
Application
Request
Local cache
Backend/Database

Lazy Load



Request
Application
Logic
Caching
(Memory)
Warmup
Worker
Backend/Databas
e

Distributed Cache



Performance: Profiler is the key

Method	Total Request	Last RPS/ Last Exe Time	Max RPS	Min RPS	Avg RPS	Max ExeT	Min ExeT	Avg ExeT
get	2929261981	21,7,14,18,20 0,0,0,0,0	61	2	21.3999996	0	0	0.01778333
multiGet	0	0,0,0,0,0 0,0,0,0,0	0	0	0	0	0	0
enable	7844039	143,132,132,209,165 0,0,0,0,0	231	44	140.983337	0	0	0.19936667
disable	0	0,0,0,0,0 0,0,0,0,0	0	0	0	0	0	0
remove	7844039	143,132,132,209,165 0,0,0,0,0	231	44	140.983337	0	0	0.21293333
getStat	6690	1,1,1,1,0 0,0,0,22,0	1	0	0.0666666701	22	0	0.41006669

Cache: item	Cache hit(%)	Cache miss(%)
23982034	99.1956482	0.804351926

Q&A

Thank You!

Contact info:

- Name: Vu Van Tiep
- Email: tiepvv@vng.com.vn / vuvantiep@gmail.com
- Skype: tiepvv

VIETNAM WEB SUMMIT

Reference

- *https://en.wikipedia.org/wiki/Cache_(computing)*
- *http://www.slideshare.net/IgorAnishchenko/pb-vs-thrift-vs-avro*
- *https://github.com/facebook/zstd*
- *http://www.codeproject.com/Articles/56138/Consistent-hashing*