Big Data Systems

Djellel Difallah

Spring 2023

Lecture 5 (cont.) –

Caching + Memcached + LRU Cache

System Design Requirements (from Facebook)

- Support a very heavy read load
  - Over 1 billion reads / second
  - Insulate backend services (DB) from high read rates

- Geographically Distributed

- System must be flexible enough to support a variety of use cases
  - Support rapid deployment of new features

- **Persistence handled separately**
  - Support mechanisms to refresh content after updates

Caching

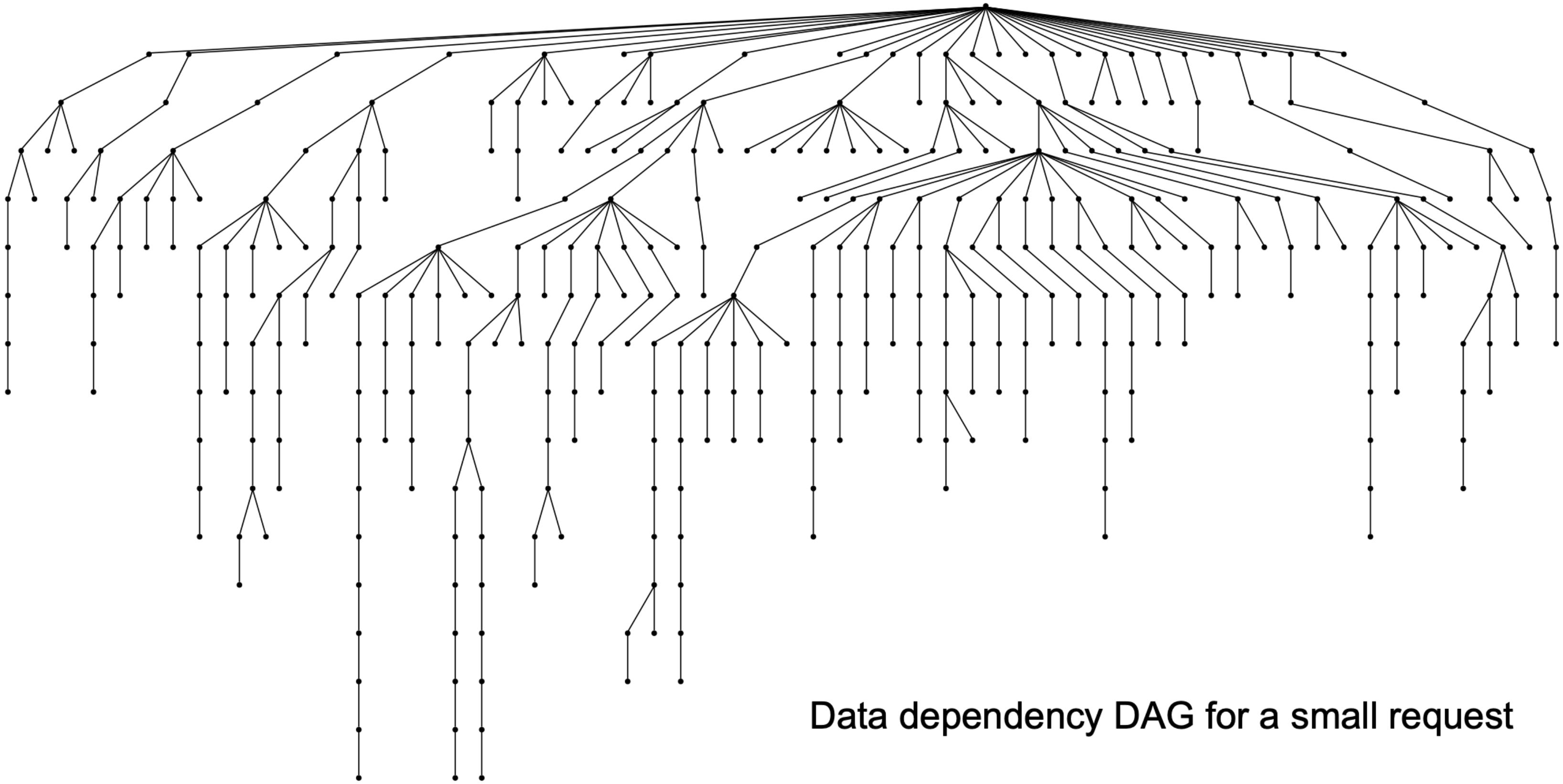
- Memory caching is essential to Web-scale Application Performance.
- Improves:
  - Read latency
  - Availability (When distributed)
  - Reduce the strain on the overall distribute system architecture
- In-memory Key Value Stores are used by virtually every large web platform: Facebook, Netflix, Pinterest, Wikipedia etc

memCached

redis

Amazon
ElastiCache

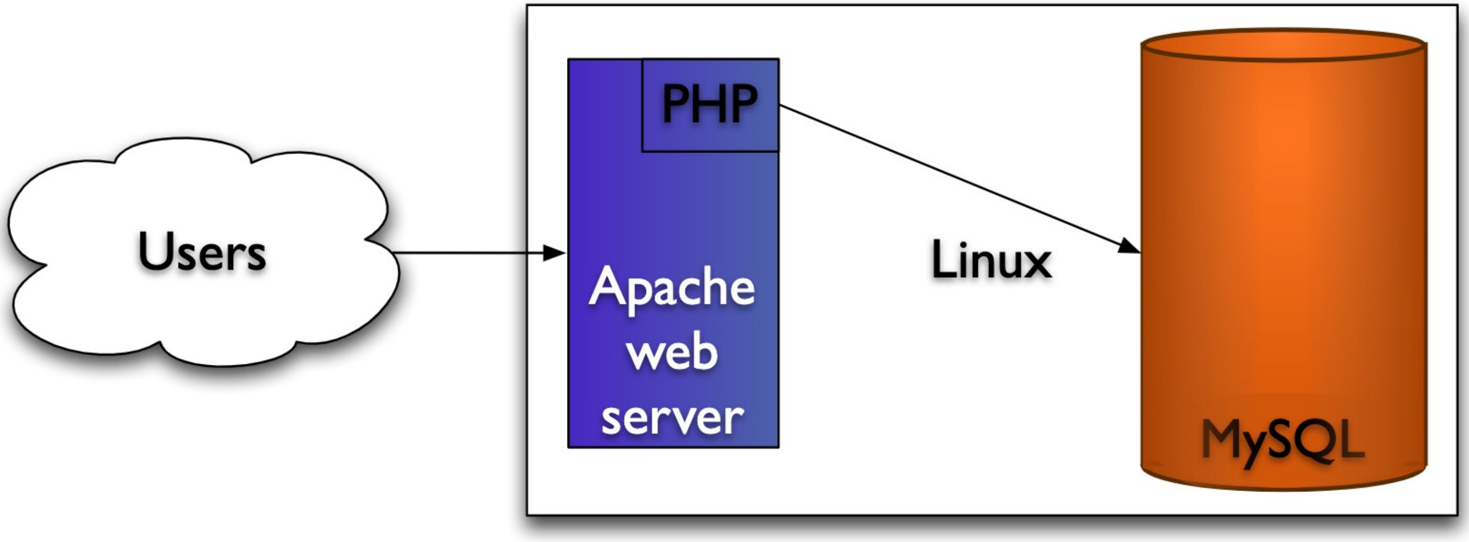
Why Caching Improves Reads?



Data dependency DAG for a small request

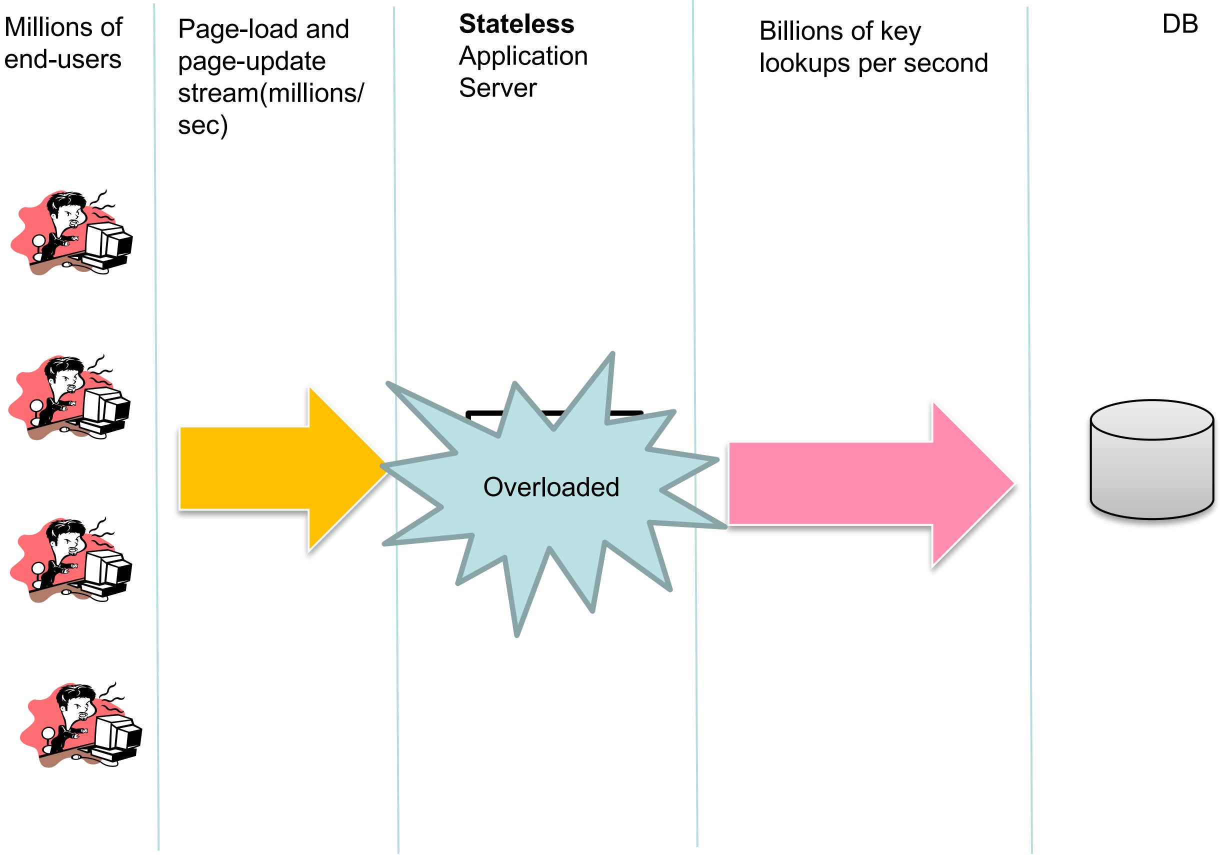
From "Scaling Memcache at Facebook" NSDI'13

Web Stack Architecture (LAMP)



Wikimedia architecture by Mark Bergsma

Why Separate Caching?



Millions of end-users

Page-load and page-update stream(millions/sec)

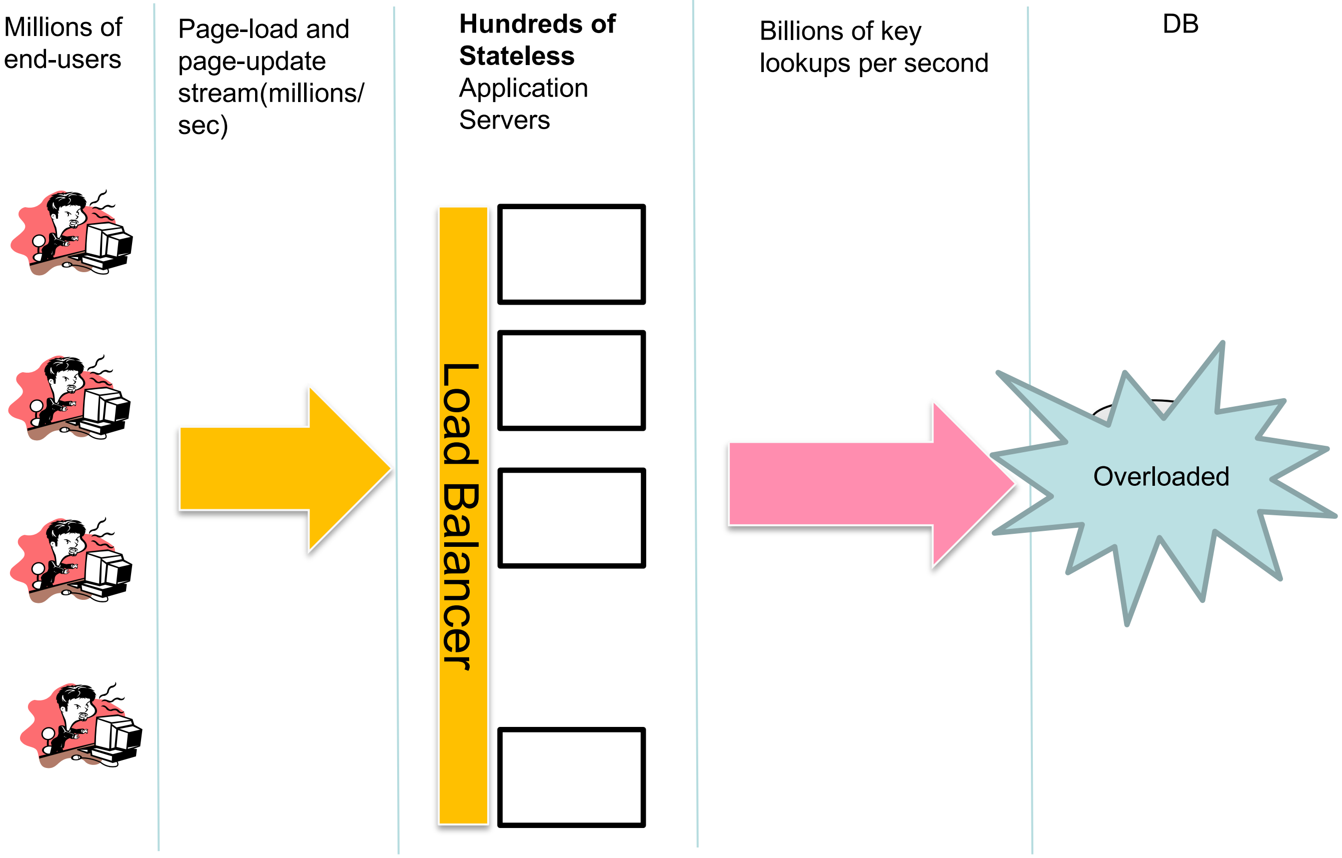
Stateless Application Server

Billions of key lookups per second

DB

Overloaded

Why Separate Caching?



Millions of end-users

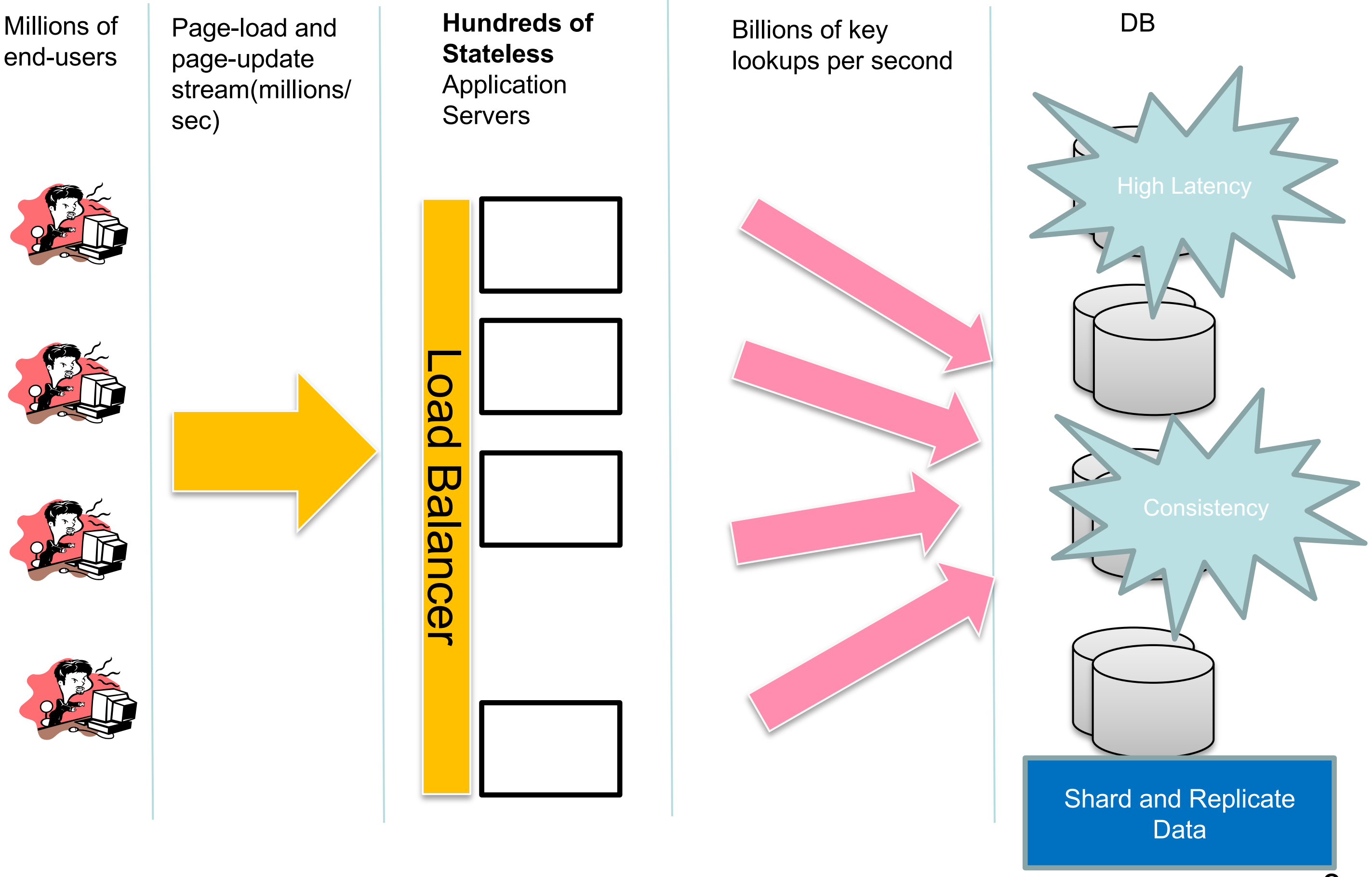
Page-load and page-update stream(millions/sec)

Hundreds of Stateless Application Servers

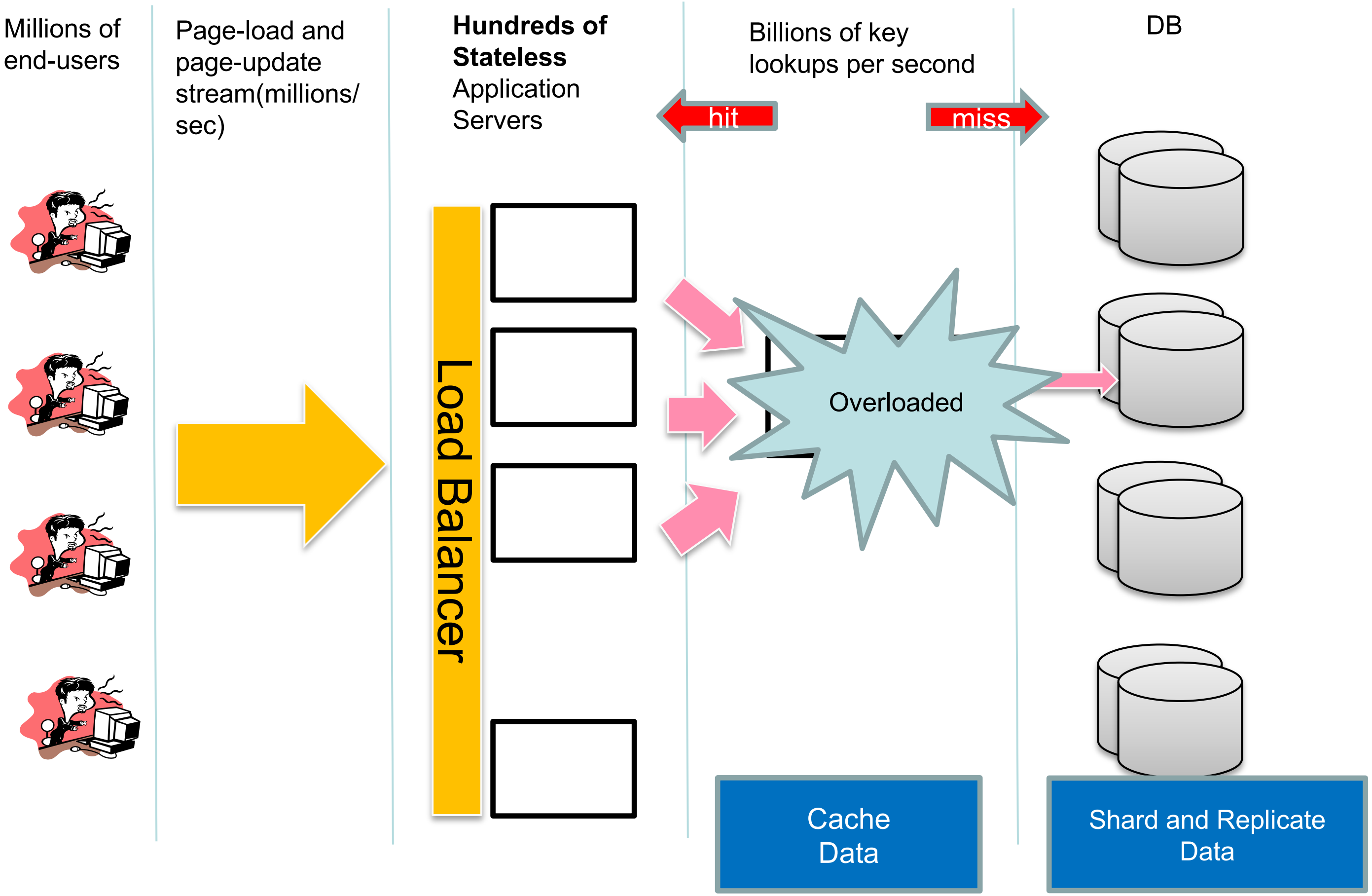
Billions of key lookups per second

DB

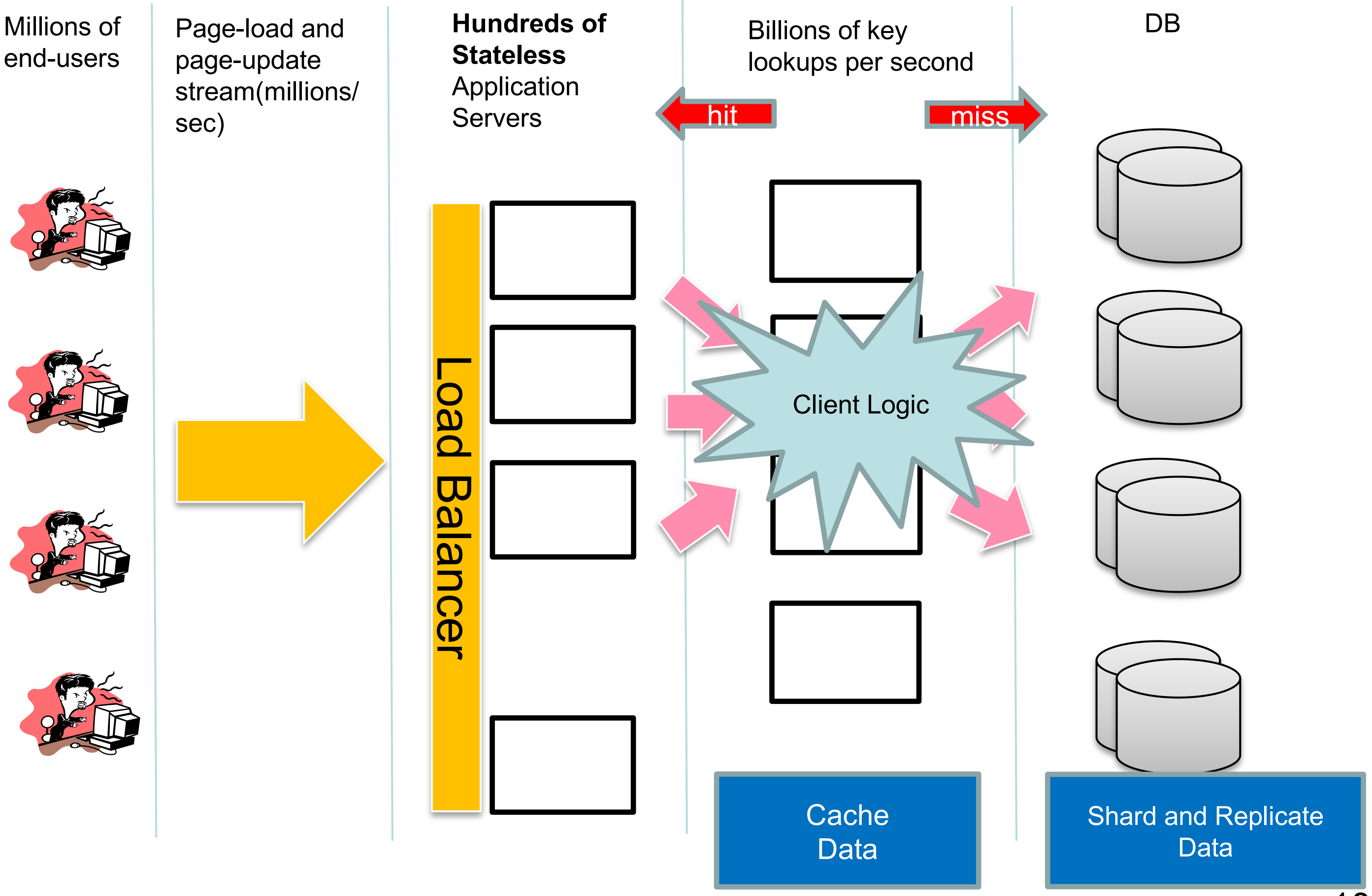
Why Separate Caching?


Millions of end-users
Page-load and page-update stream(millions/sec)
Hundreds of Stateless Application Servers
Billions of key lookups per second
DB
Load Balancer
High Latency
Consistency
Shard and Replicate Data

Why Separate Caching?


Millions of end-users
Page-load and page-update stream(millions/ sec)
Hundreds of Stateless Application Servers
Billions of key lookups per second
DB
hit
miss
Load Balancer
Overloaded
Cache Data
Shard and Replicate Data

Why Separate Caching?



Millions of end-users

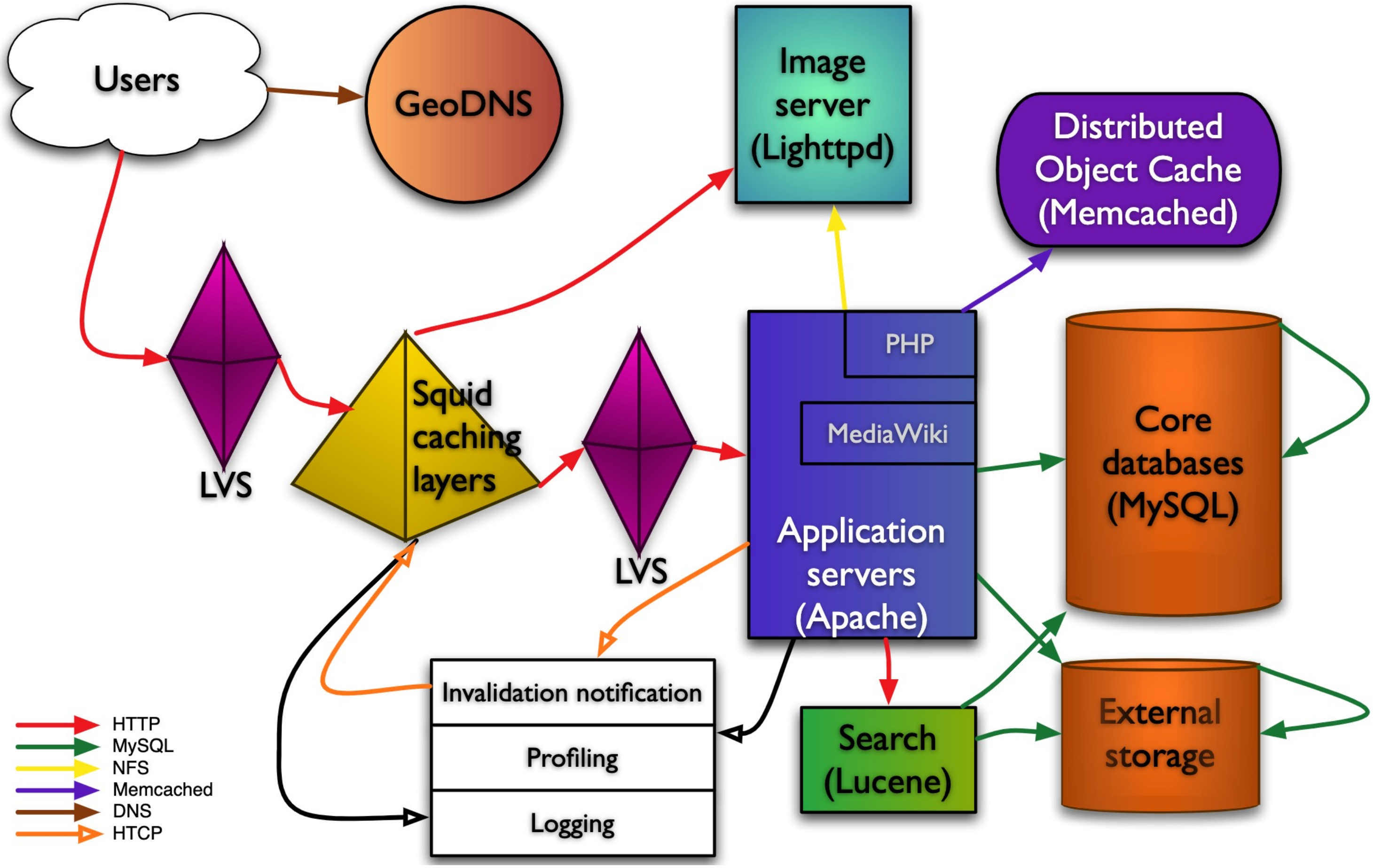
Page-load and page-update stream(millions/sec)

Hundreds of Stateless Application Servers

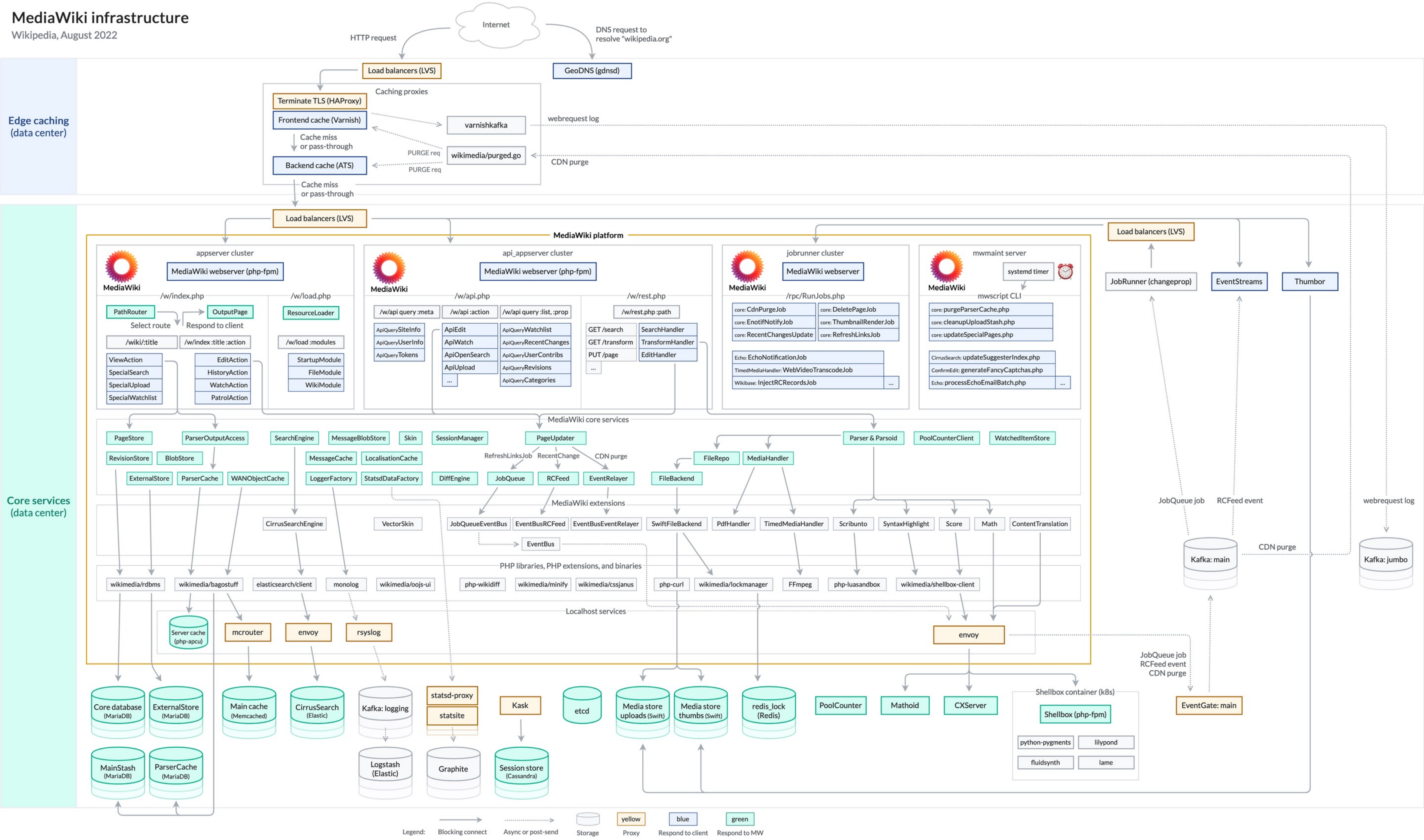
Billions of key lookups per second

DB

Wikipedia Architecture



Wikimedia architecture by Mark Bergsma (2007)



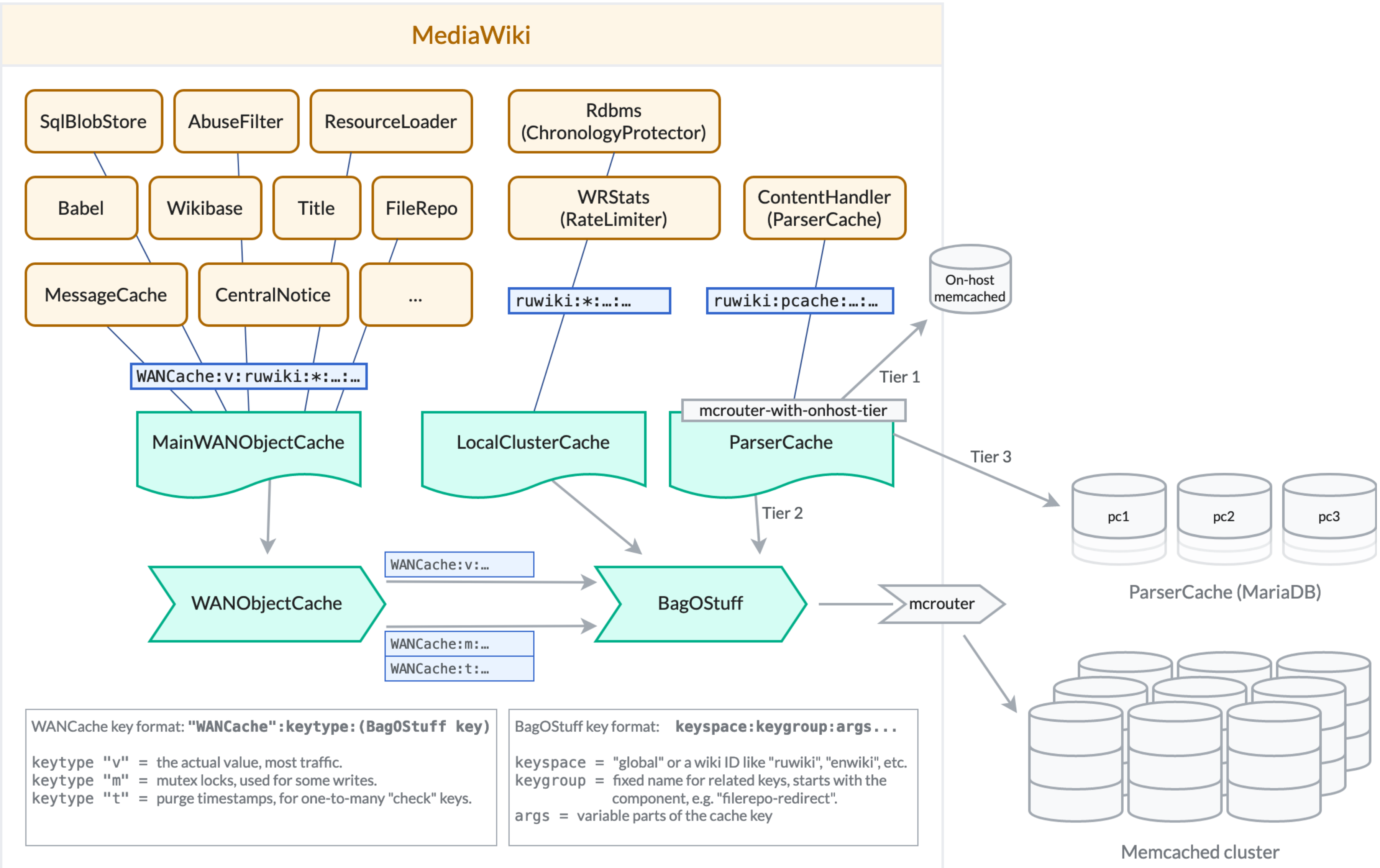
MediaWiki infrastructure

Wikipedia, August 2022

Fast Forward.. MediaWiki 2022

Memcached flow from MediaWiki

Wikipedia, August 2022



WANCache key format: **"WANCache":keytype:(BagOStuff key)**

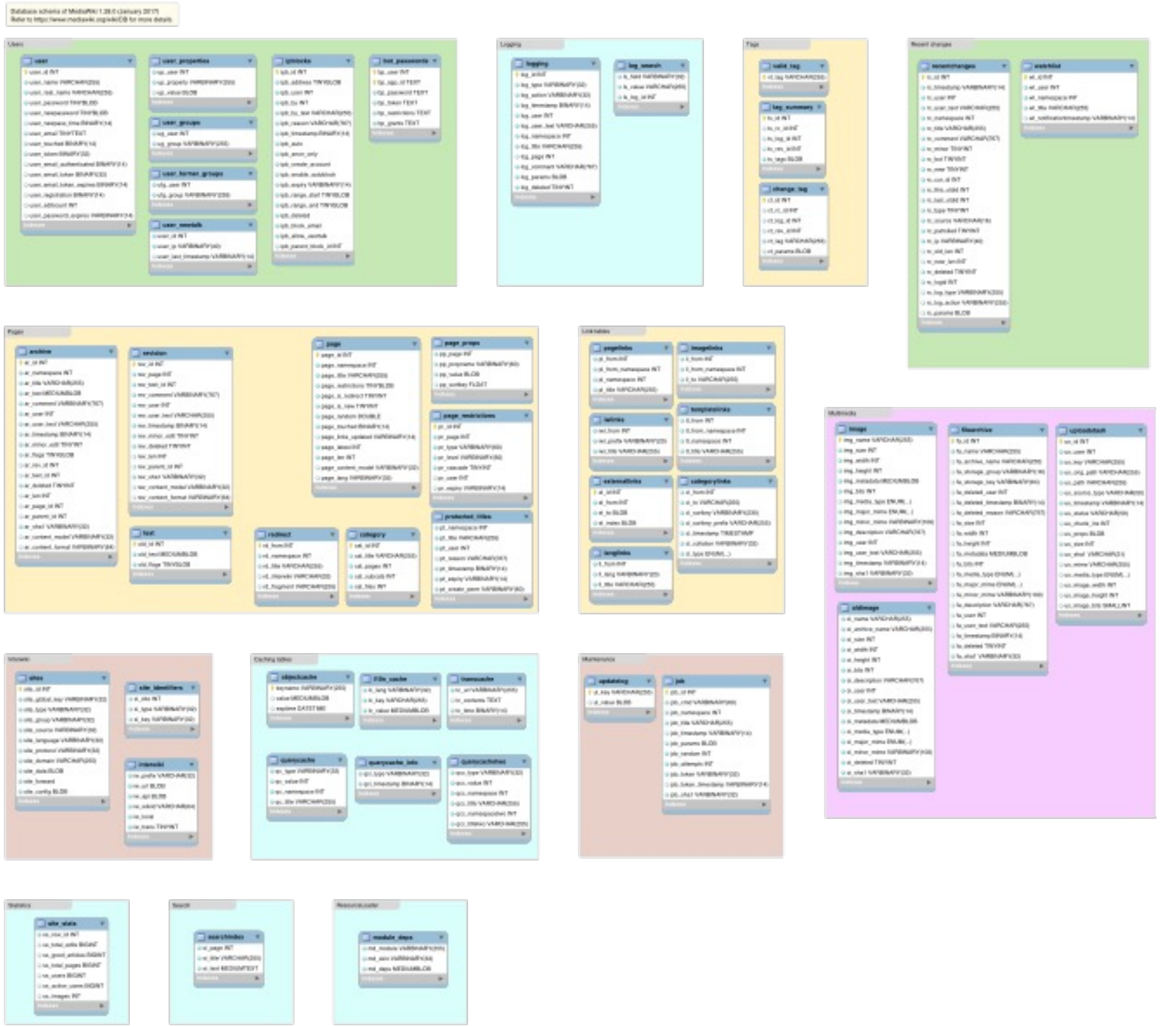
```
keytype "v"  =  the actual value, most traffic.
keytype "m"  =  mutex locks, used for some writes.
keytype "t"  =  purge timestamps, for one-to-many "check" keys.
```

BagOStuff key format: **keyspace:keygroup:args...**

```
keyspace  =  "global" or a wiki ID like "ruwiki", "enwiki", etc.
keygroup  =  fixed name for related keys, starts with the
             component, e.g. "filerepo-redirect".
args  =  variable parts of the cache key
```

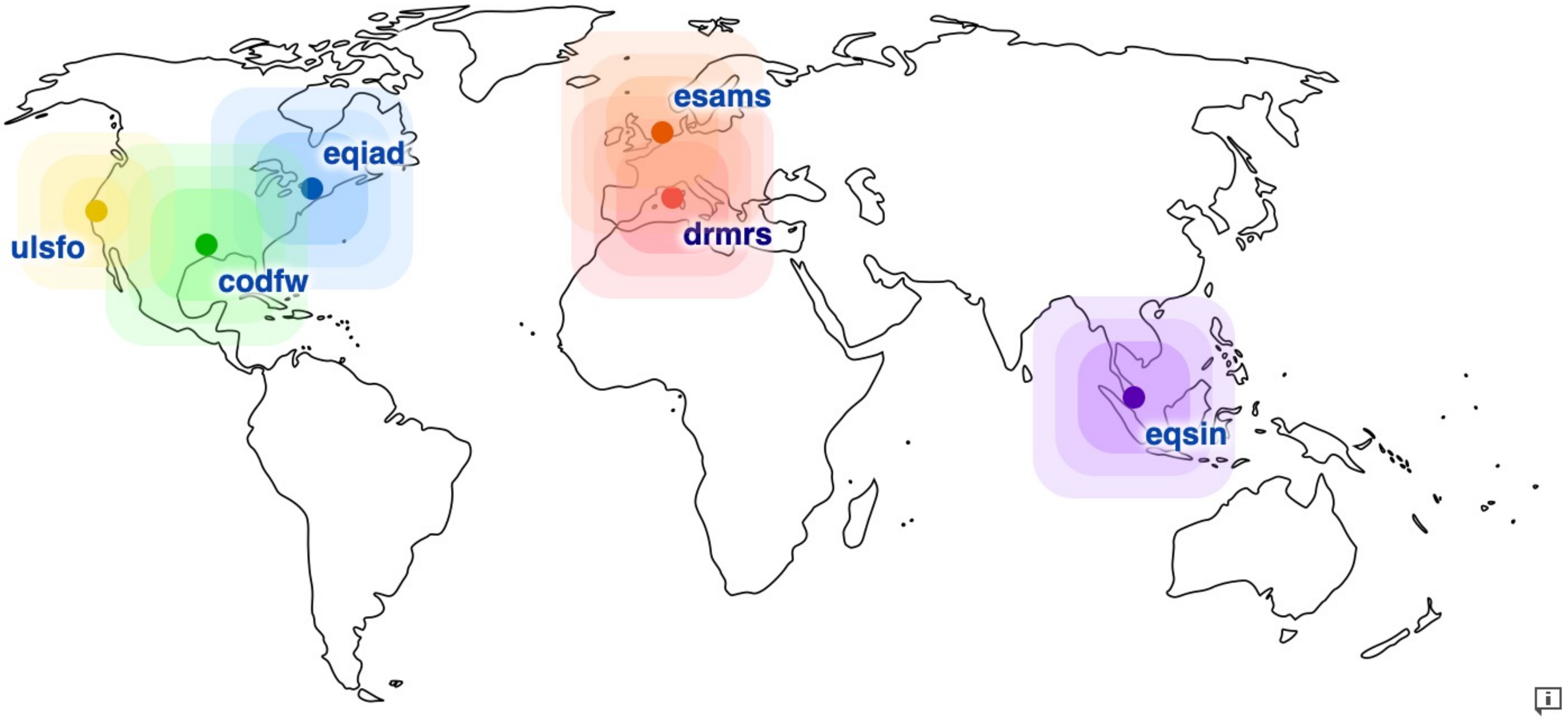
Fast Forward.. MediaWiki 2022

MediaWiki Database Schema



https://www.mediawiki.org/w/index.php?title=Manual:Database_layout/diagram&action=render

Map of Wikmiedia Data Centers



https://wikitech.wikimedia.org/wiki/Data_centers

Introduction to Memcached

- What is Memcached?
  - High-performance, distributed memory object caching system.

- When can we us it?
  - Anywhere where there is RAM
  - Used to cache *objects*

- Why should we us it?
  - If we have a high-traffic site that is dynamically generated with a high database load that contains mostly read threads

Introduction (cont.)

- Memcached is not:
  - A persistent data store
  - A database
  - Application-specific
  - A large object cache
  - Fault-tolerant or highly available

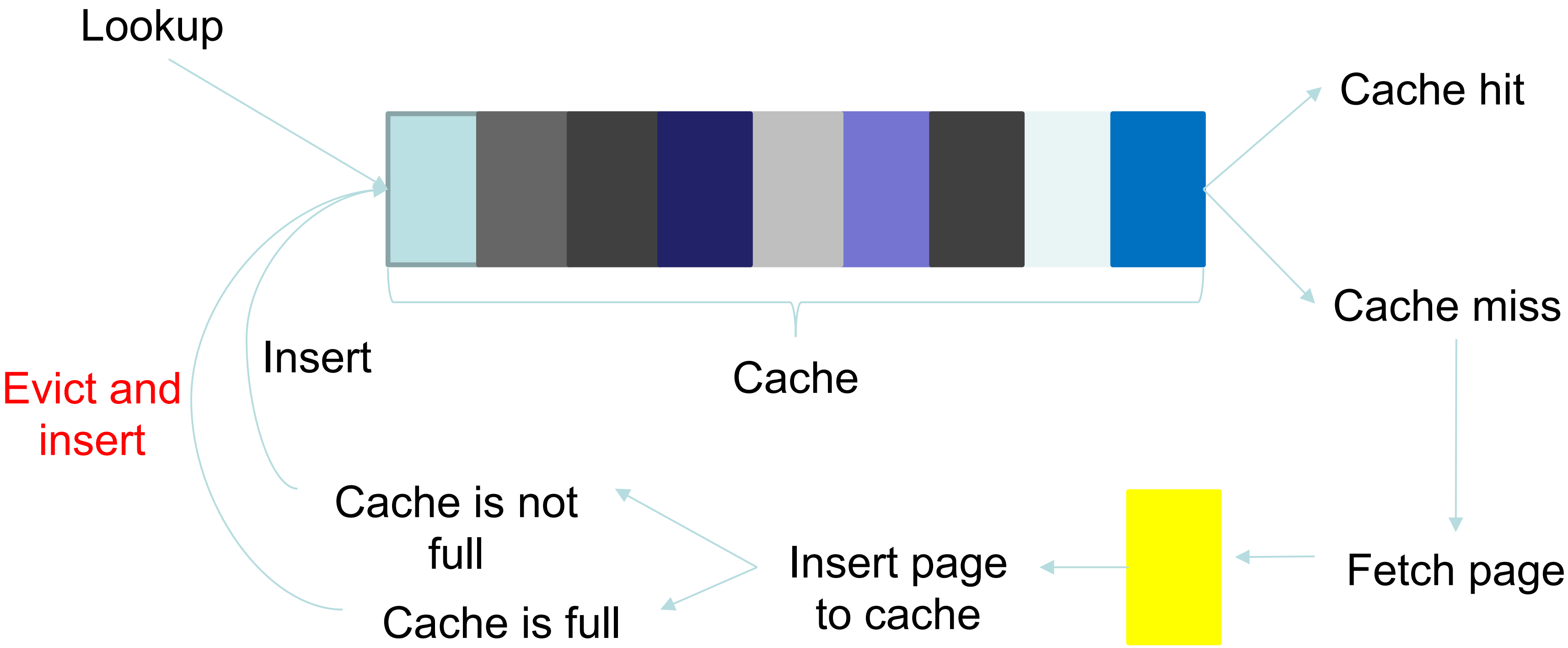
Introduction (cont.)

- Memcached is
  - Pure single-node
  - Key-value store (simple set/get operations)
  - Optimized for multithreading (compared to Redis)
- Limits
  - Key size = 250 bytes
  - Item size: The default value is 1 MB

Use cases for Caching

Site Type	Repeatable Use
Social Networking	Profile caching
Content Aggregation	HTML/page caching
Ad Targeting	Cookie/Profile tracking
Location-based services	DB Query scaling
E-Commerce	Session caching

Cache Management



Lookup

Cache hit

Cache miss

Cache

Insert

Evict and insert

Cache is not full

Cache is full

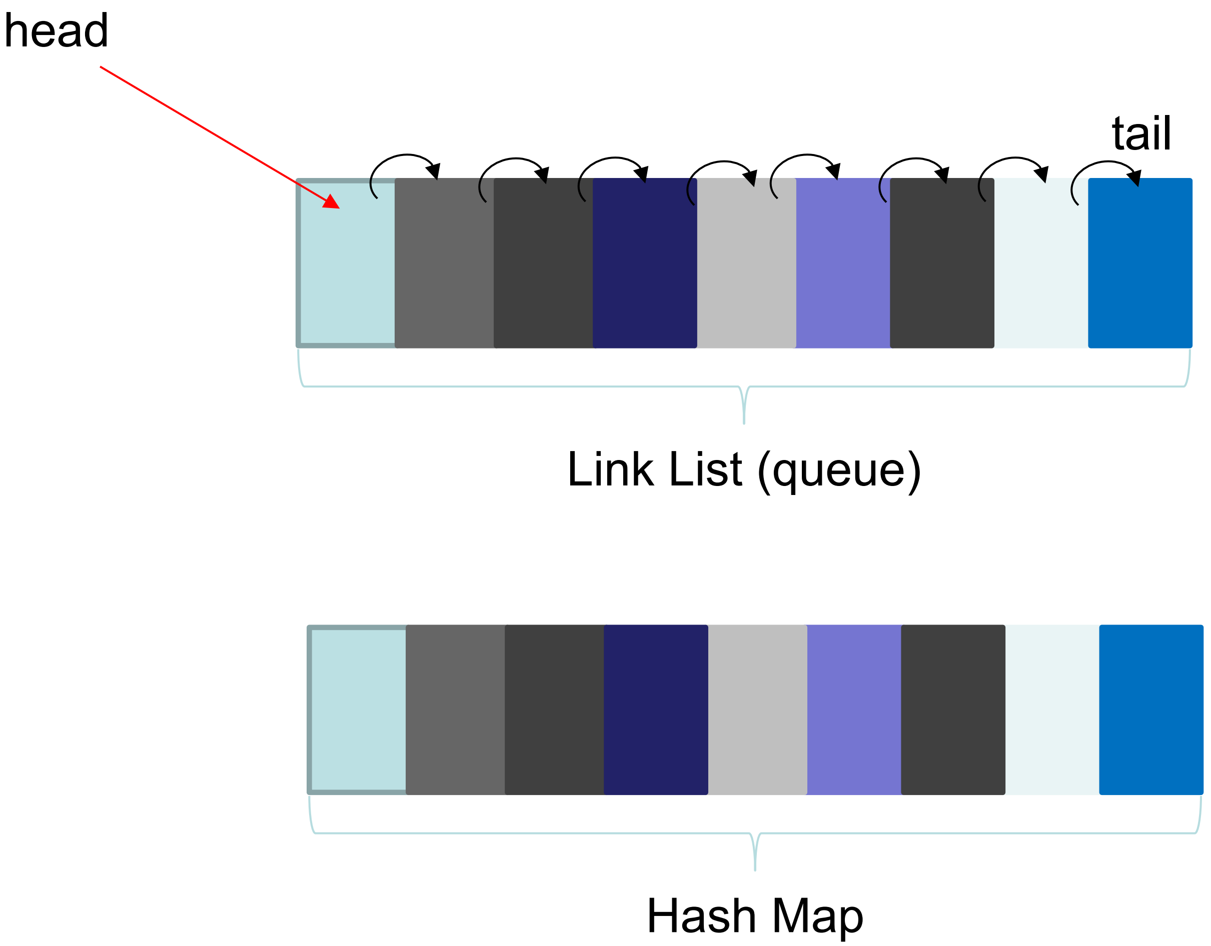
Insert page to cache

Fetch page

Cache Eviction
LRU Cache

- Least recently accessed items are cycled out
- Memcached deals with memory fragmentation using “slab” memory allocation. i.e., different categories of object size are grouped in similar memory allocated areas.
- One LRU exists per “slab class”
- LRU “evictions” need not be common
- Keys expiration time (exptime), however the LRU algorithm may remove expired keys before they are accessed.

Cache Eviction
LRU Cache



Install Memcached

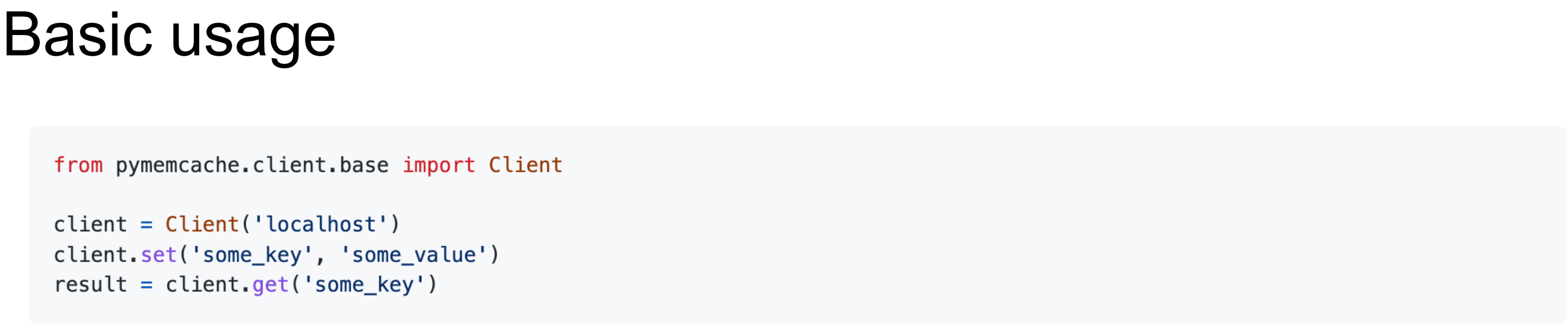
- MacOS
  - brew install Memcached
- Linux
  - apt-get install memcached
- Windows:

  https://github.com/jefyt/memcached-windows

- libMemcache for C++
  - Adds many funcationalities, including consistent hashing
  - Recall, memcached is single-node
- Most libs are wrappers.

Using Memcached in Python

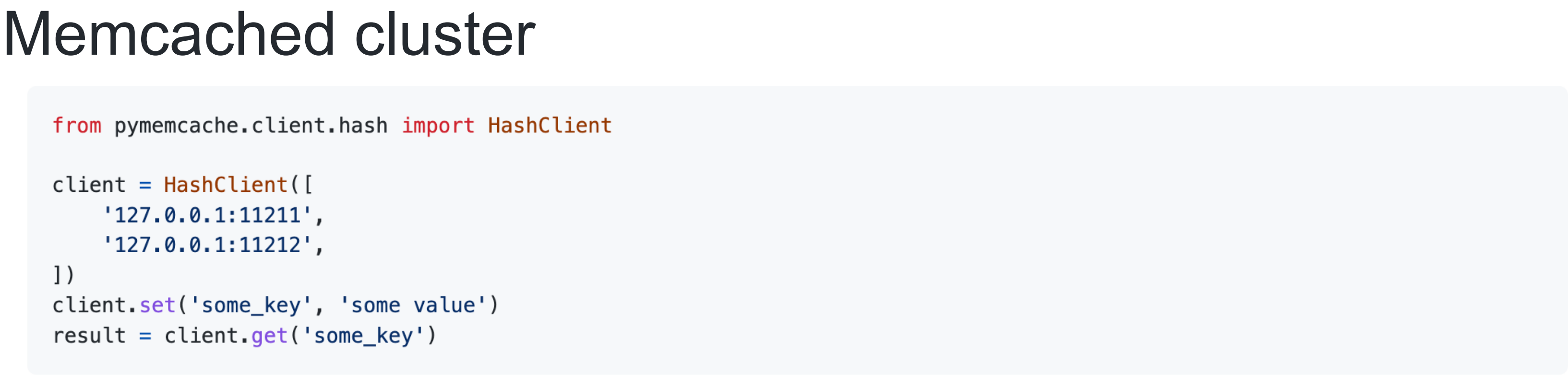
- https://github.com/pinterest/pymemcache
- Quick Start



Basic usage

```
from pymemcache.client.base import Client

client = Client('localhost')
client.set('some_key', 'some_value')
result = client.get('some_key')
```



Memcached cluster

```
from pymemcache.client.hash import HashClient

client = HashClient([
    '127.0.0.1:11211',
    '127.0.0.1:11212',
])
client.set('some_key', 'some value')
result = client.get('some_key')
```

Demo on telnet ☺

```
telnet 127.0.0.1 11211
set v0 0 10 1 h
set v1 0 0 10 TestValue1
get v1
get v0
delete v1
stats items
stats cachedump 1 10
```