CS-310 Scalable Software Architectures

Lecture 4: Proxies and Caches

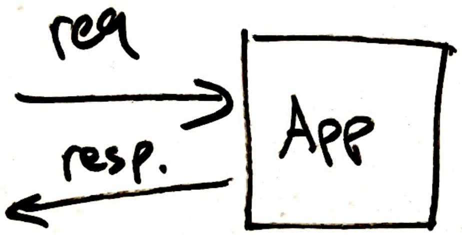
Steve Tarzia

Last time: Stateless Services

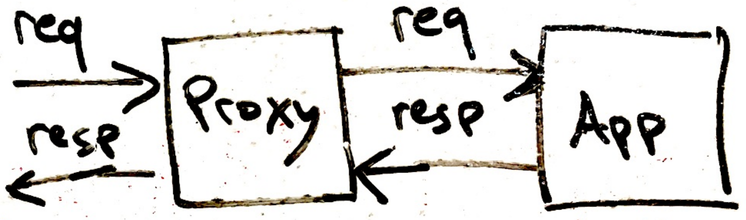
- Defined **stateless** and **stateful** services.
- Showed how databases and cookies make MediaWiki stateless and scalable.
- In other words, we achieved parallelism and distributed execution while avoiding difficult coordination problems. Just push away all shared state. Push state **up** to client and/or **down** to database.
- First lesson of scalability: **Don't share!**

Proxies

- A proxy server is an **intermediary router** for requests.
- The proxy does not know how to answer requests, but it knows who to ask.
- The request is relayed to another server and the response relayed back.
- Proxies can be transparently added to **any stateless service**, like HTTP:



Before



After adding proxy

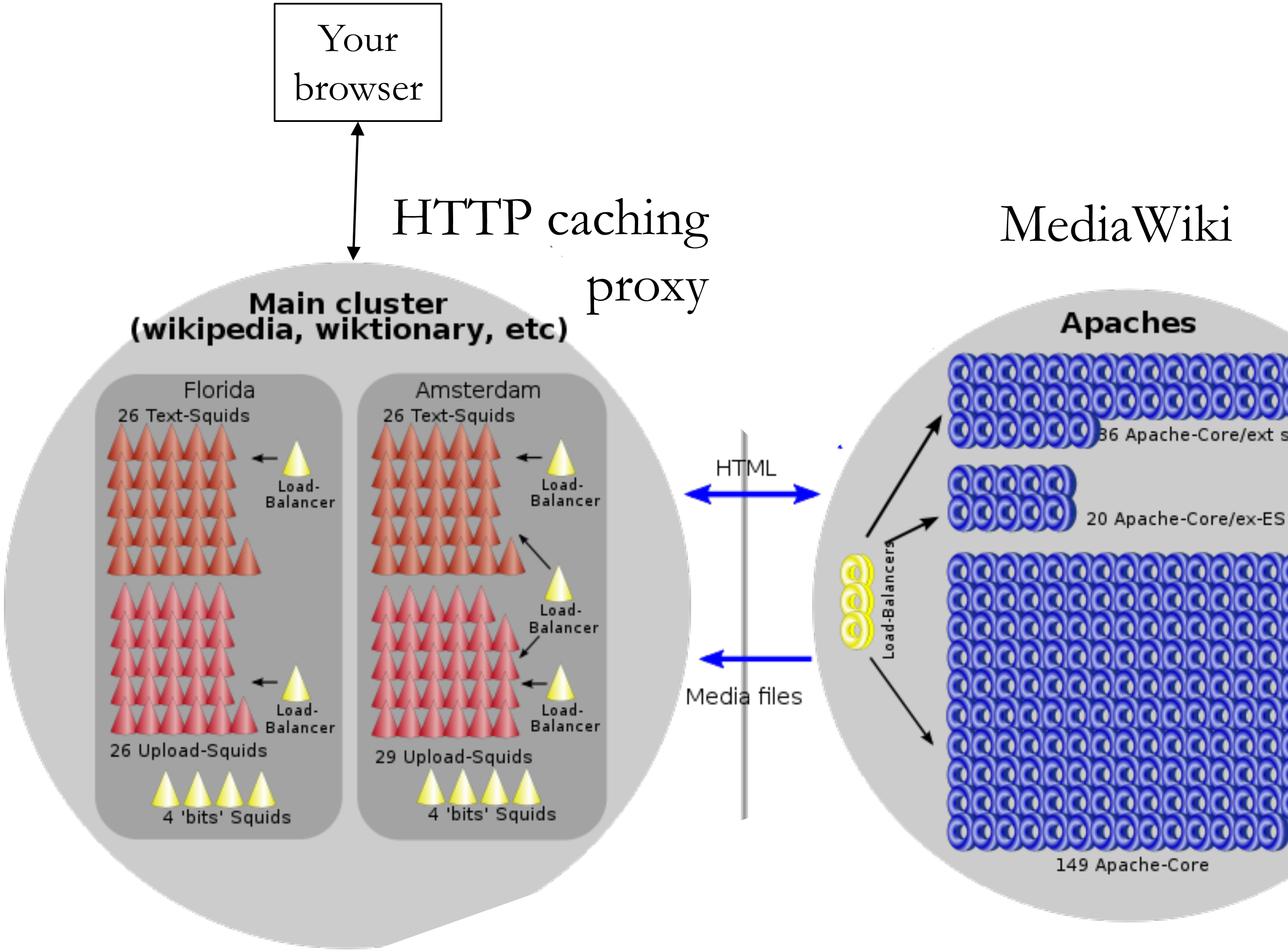
- A **load balancer** is a type of proxy that connects to many app servers.
  - The work done by the load balancer is very simple, so it can handle much more load than an application server.
  - Creates a single point of contact for a large cluster of app servers.

Front-end Cache

Squid is a **caching** proxy.

(A cache stores recently retrieved items for reuse)

- Frequent requests are found in (*hit*) the cache, without re-asking MediaWiki and accessing the shared database.
- Unusual requests are not in (*miss*) the cache, and are relayed to MediaWiki.



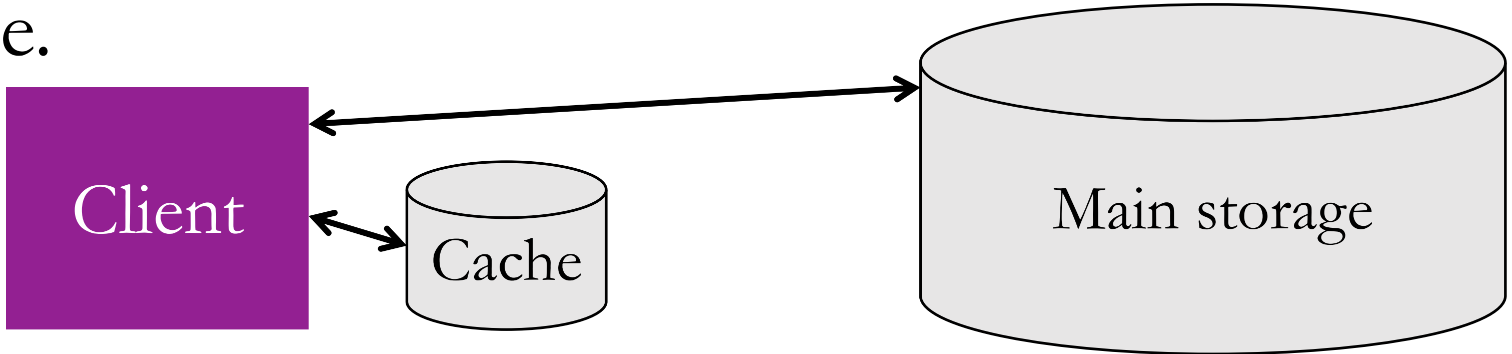
Your browser

HTTP caching proxy

MediaWiki

Cache basics

- Caching is a general concept that applies to web browsers, computer memory, filesystems, databases, etc.
  - any time you wish to improve performance of data access.
- A **cache** is a small data storage structure designed to improve performance when accessing a large data store.
  - For now, think of our data set as a dictionary or map (storing key-value pairs).
- The cache stores the **most recently** or **most frequently** accessed data.
- Because it's small, the cache can be accessed more quickly than the main data store.



Cache **hits** and **misses**

- The cache is small, so it cannot contain every item in the data set!

When reading data:

1. Check cache first, hopefully the data will be there (called a cache **hit**).
   - Record in the cache that this entry was accessed at this time.
2. If the data was not in the cache, it's a cache **miss.**
   - Get the data from the main data store.
   - Make room in the cache by **evicting** another data element.
   - Store the data in the cache and repeat Step 1.

Which data should be evicted?

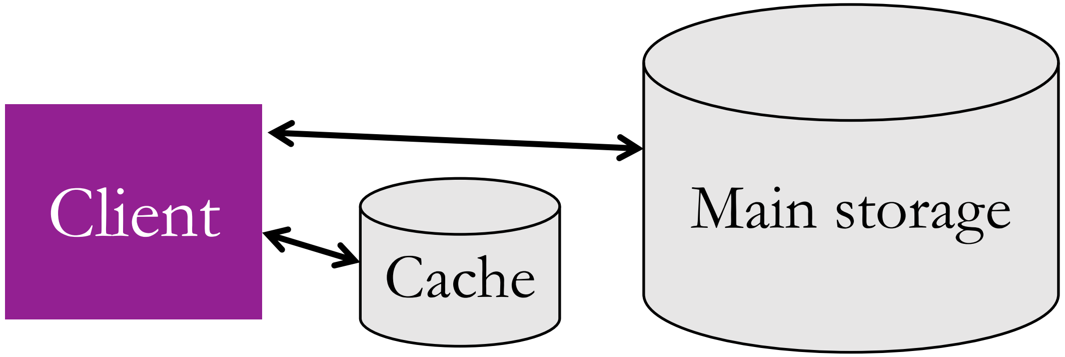
STOP and THINK

- The most common eviction policy is LRU: least recently used

Types of Caches

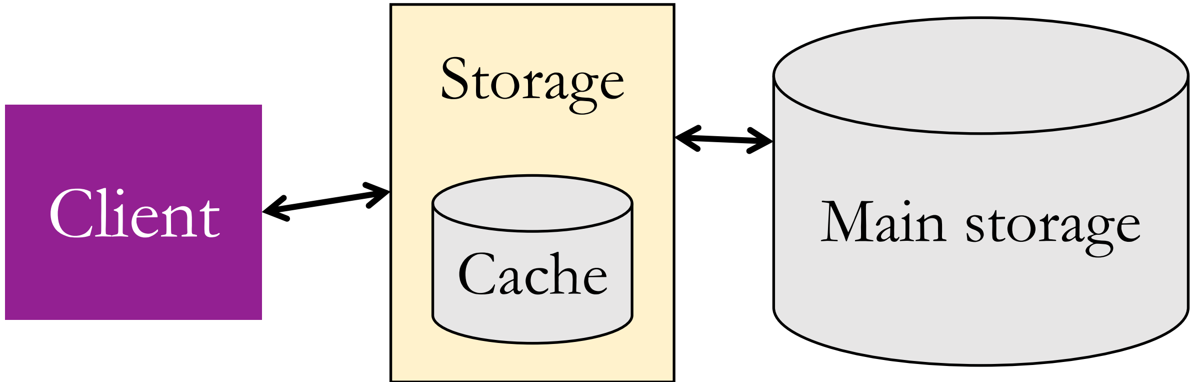
Managed Cache

- Client has direct access to both the small and large data store.
  - Client is responsible for implementing the caching logic.
- Eg.: Redis, Memcached



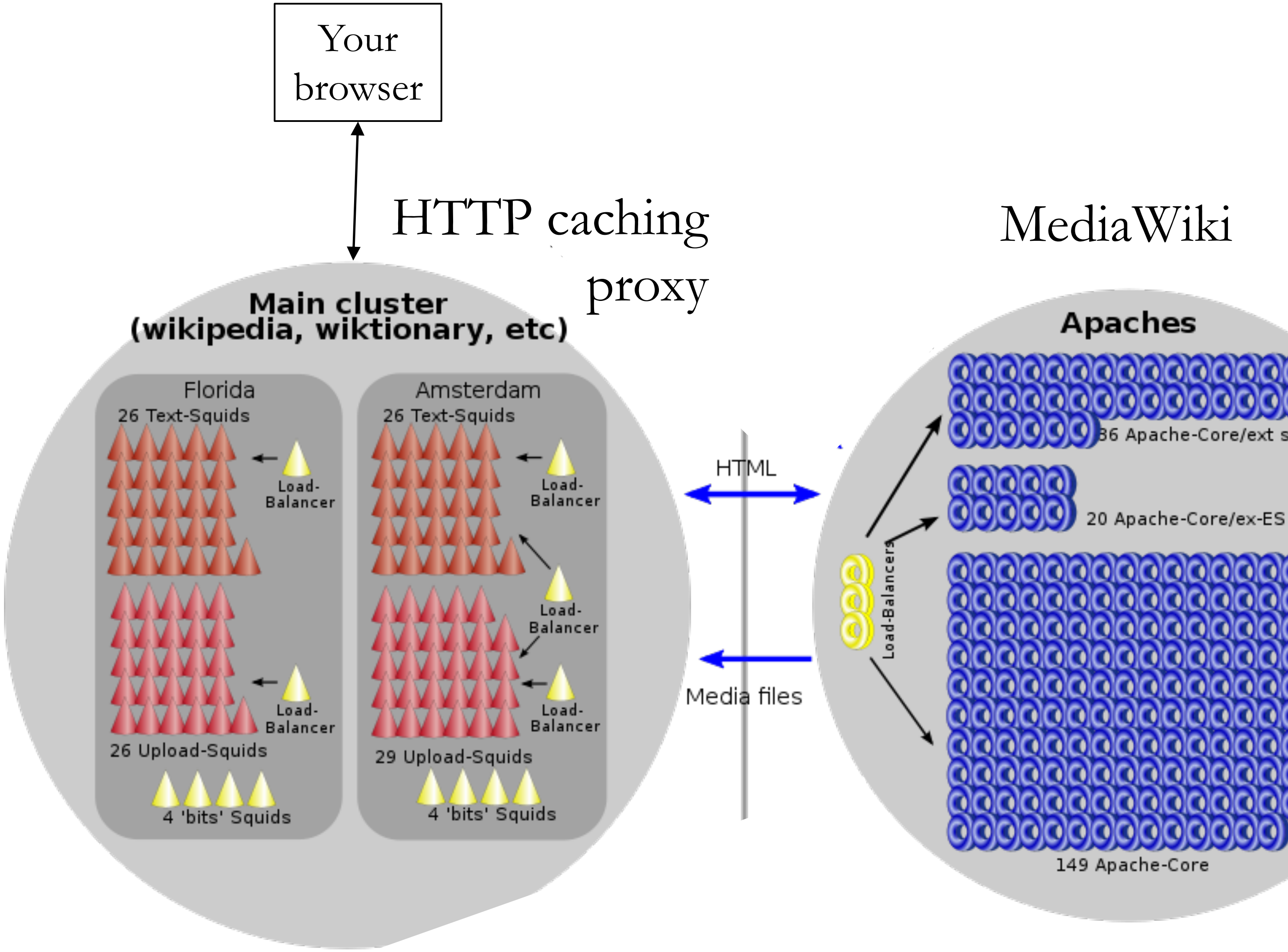
Transparent Cache

- Client connects to one data store.
- Caching is implemented inside storage "black box."
- Eg.:
  - Squid caching proxy, CDNs
  - Database server.



Stop and think

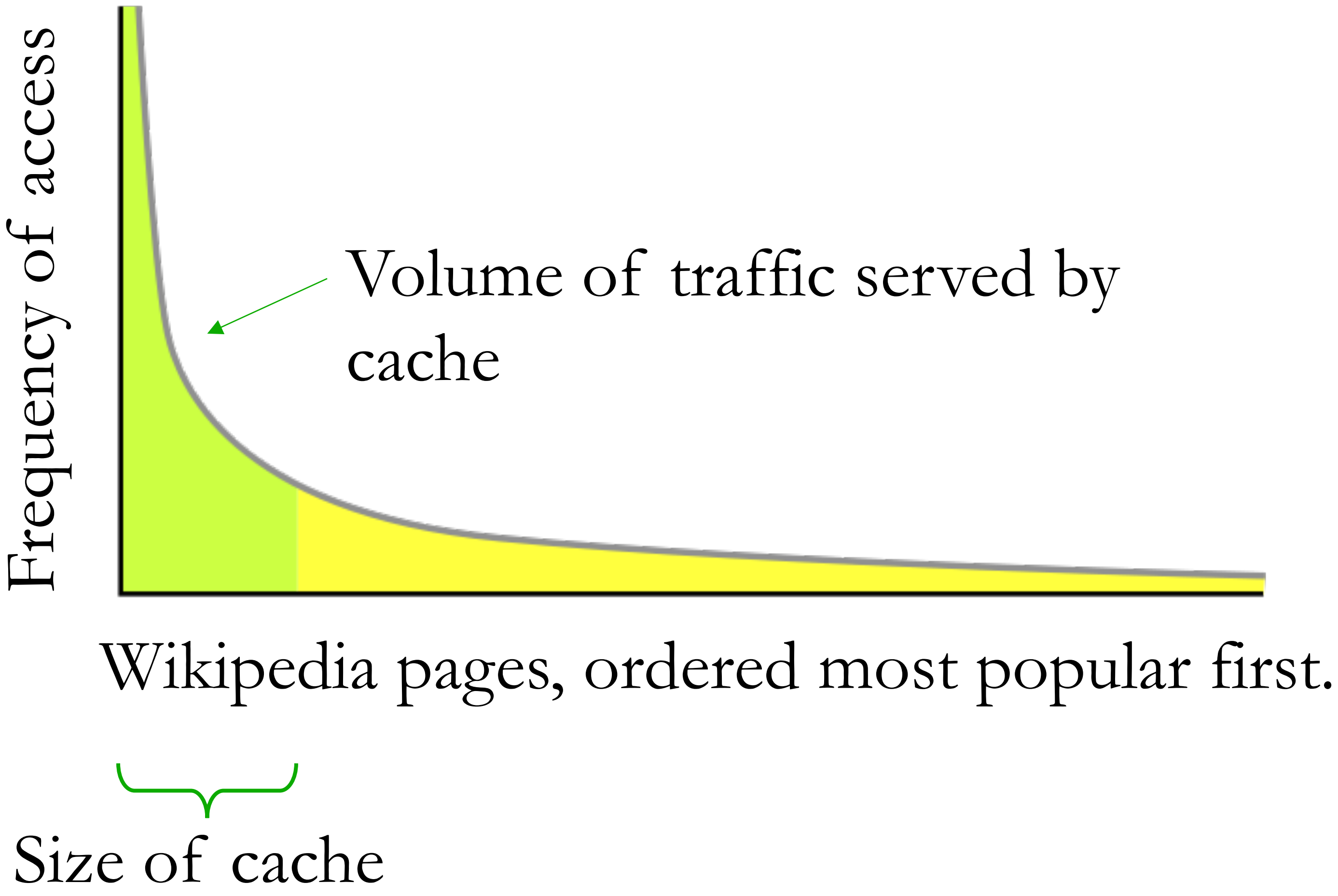
- A small frontend cache might serve 90% of the requests without touching the shared database.
- Why is Wikipedia able to handle so many of its requests from a cache?
- What prices do we pay for this efficiency?


Your browser

HTTP caching proxy

MediaWiki

"Long tail" of Wikipedia



- A small fraction of Wikipedia pages account for a large fraction of traffic.
  - Optimize performance for these pages.
  - These will naturally be stored in the frontend cache.
- The "long tail" is the large number of rarely-accessed pages.
  - Most accesses to these rare pages involve a database access

Data **writes** cause cache to be out of date!

- Remember that we can have many clients, each with its own cache.
- When data changes, out-of-date copies of data may be cached and returned to clients. Eg., a Wiki article is edited. What to do?

STOP
and
THINK

Three basic solutions:

- **Expire** cache entries after a certain TTL (time to live)
- After writes, send new data or an invalidation message to all caches. This creates a **coherent cache**. But it adds performance overhead.
- Don't every change your data! For example, create a new filename every time you add new data. This is called **versioned data**.

HTTP support caching well

- HTTP is **stateless**, so the same response can be saved and reused for repeats of the same request.
- HTTP has different **methods** GET/PUT/POST/DELETE.
  - GET requests can be cached, others may not because they modify data.
- HTTP has **Cache-Control headers** for both client and server to enable/disable caching and control expiration time.

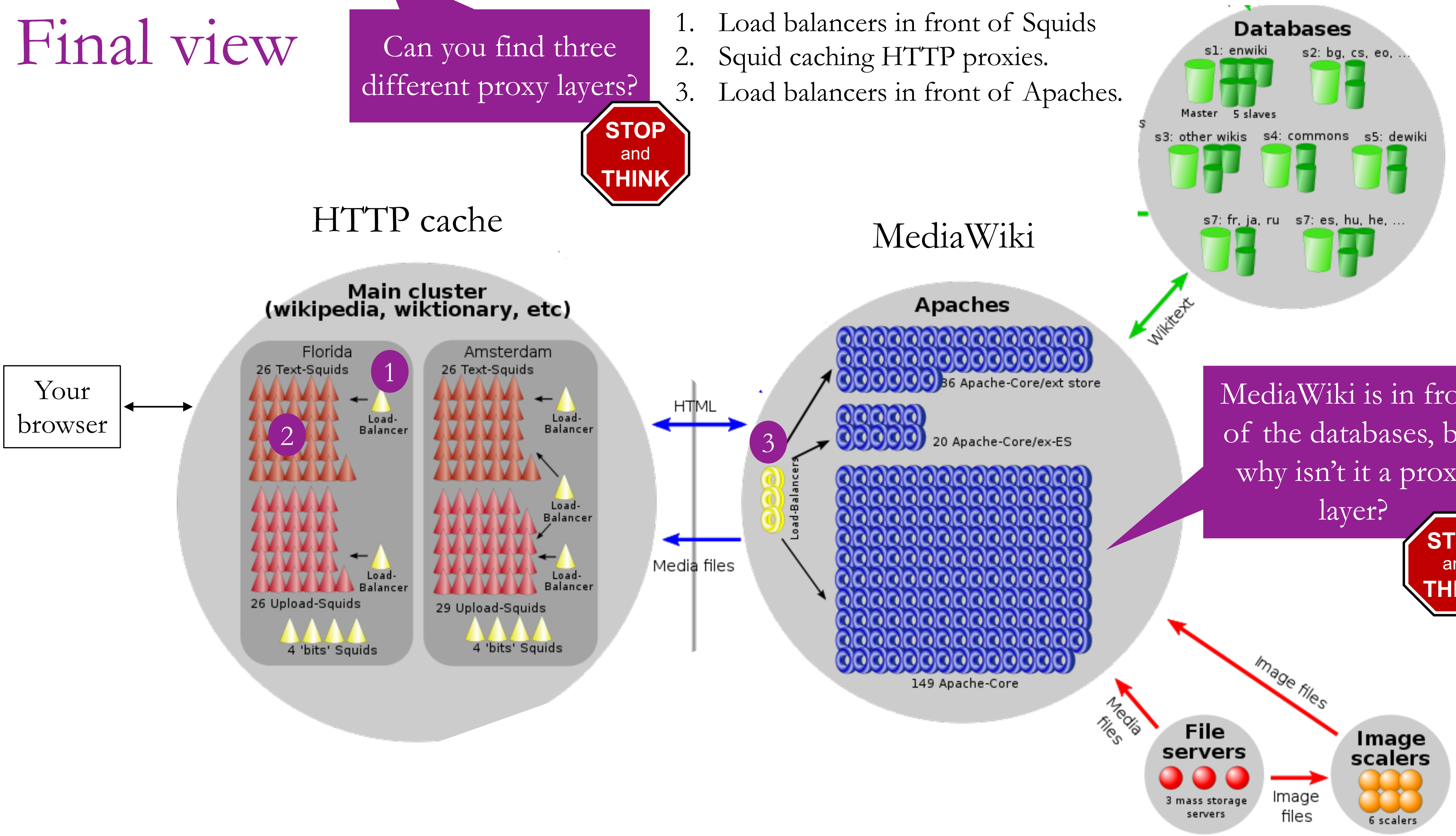
- These features allow a web browser to skip repeated requests.
- Also, an HTTP caching proxy, like Squid, is compatible with any web server and can be *transparently* added.

Final view

Can you find three different proxy layers?

STOP and THINK

1. Load balancers in front of Squids
2. Squid caching HTTP proxies.
3. Load balancers in front of Apaches.



MediaWiki is in front of the databases, but why isn't it a proxy layer?

STOP and THINK

Review

- Introduced **proxies** and **caching**.
- A proxy is an intermediary for handling requests.
  - Useful both for **caching** and **load balancing** (discussed later).
- Often, many of a service's requests are for a few popular documents.
  - Caching allows responses to be saved and repeated for duplicate requests.
- HTTP has built-in support for caching.