Unit 1 – Case Study: Wikipedia

Network Information System Technologies

Goals

- Present a case study that shows why some technologies are needed in networked systems.
- Analyse which is the aim of those technologies.
- Show the architecture of a scalable distributed system.
- Know the evolution of that scalable system.

Index

1. Introduction

- Goal of TSR
  - Describe current technologies in networked information systems
    - Networked information systems → Distributed systems (DSs)
- To this end...
  - A case study showing the problems to be solved in DSs is needed.
  - Different technologies solve different problems.
    - Although there is no one-to-one technology-problem relationship.

1. Introduction

- Case study candidates:
  - Those large enough to rise as many challenges and problems as possible.
  - Examples:
    - Google search service. It uses more than 1 million servers.
    - Facebook social network. More than 1000 million users.
    - Chinese official website for booking train tickets. More than 1000 million requests per day (in January 2012!!).
    - YouTube. More than 1000 million users and more than 7000 million requests per day.
  - But we need reliable and public information about their architecture and their technologies...
    - ...and it is unavailable (and confidential) in most cases.
  - The **Wikipedia** system is a good choice!!

Index

2. Wikipedia nowadays

- Wikipedia is...
  - a digital encyclopaedia created in 2001,
  - written collaboratively by volunteer editors,
  - available in many languages (currently, 323),
  - administered by the Wikimedia Foundation
    - with only 400 employees!!!
- ...and it had in 2015...
  - around 5 million articles in its English version,
    - 35 million articles considering all available languages
  - 25 millions of registered users,
  - 73000 active editors,
  - 500 million visitors per month

2. Wikipedia nowadays

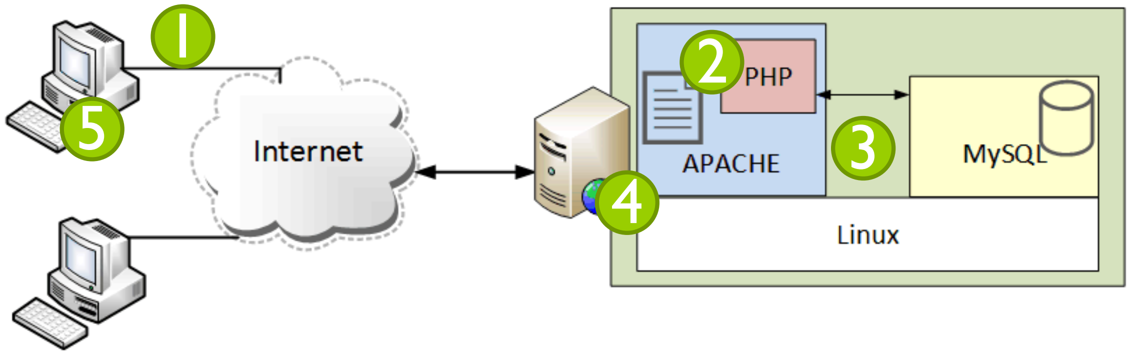
- So Wikipedia is a good example of large distributed service:
  - Based on “wiki” technology:
    - Collaborative edition of contents
    - It maintains a history of the updates applied on each page (encyclopaedia articles, in this case)
    - Quoting the “wiki” entry in the Wikipedia...
      - *“**A wiki enables communities to write documents collaboratively, using a simple markup language and a web browser.** A single page in a wiki website is referred to as a "wiki page", while the entire collection of pages, which are usually well interconnected by hyperlinks, is "the wiki". **A wiki is essentially a database for creating, browsing, and searching through information. A wiki allows non-linear, evolving, complex and networked text, argument and interaction.**”*

Index

LAMP systems

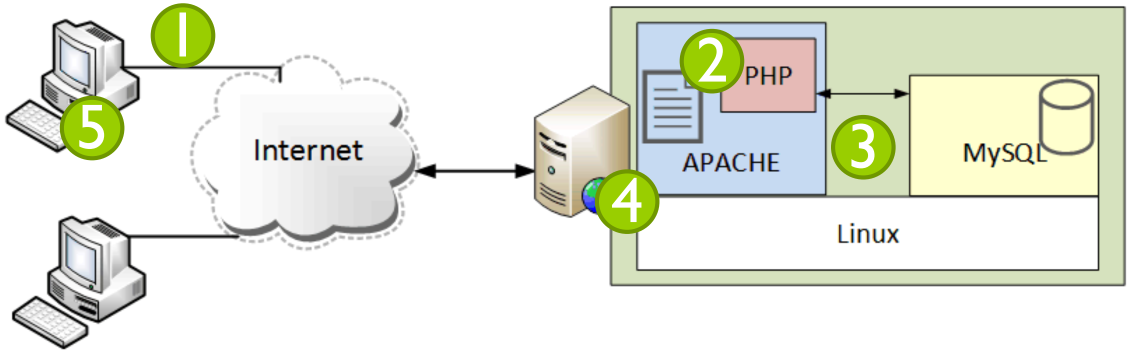
- The Wikipedia engine is called MediaWiki.
- MediaWiki is a LAMP system.
  - LAMP = Linux + Apache + MySQL + PHP
    - Linux is an example of UNIX operating system.
    - Apache is a web server.
    - MySQL is a database management system (DBMS).
    - PHP is a scripting language.
  - A LAMP system usually follows a 3-layered architecture
    - The **user interface** layer is implemented by the hypertext documents returned by the Apache web server and shown by the client web browser.
    - The **application** layer consists of the application rules and is implemented in PHP.
    - The **data** layer consists of the persistent data and is implemented by the MySQL database.
    - All layers are deployed onto Linux nodes.

LAMP systems



- A request is served following this sequence:
  1. A client sends the request to the Apache server.
  2. Apache forwards that request to its internal PHP module.
  3. The PHP program sends multiple requests to the MySQL process.
  4. With the results of those requests a web document is built and returned to the client by the Apache server.
  5. The client shows the resulting page in its browser.

LAMP systems



- Note that Apache and MySQL are two different processes:
  - They may be deployed in two different computers in order to improve performance.
  - Apache behaves as a client of MySQL.

Index

4. MediaWiki

- Let us compute approximately which workload is supported by MediaWiki in its Wikipedia implementation...
  - On peak workloads, 27774 accesses per second are reached.
  - Those accesses, multiplied by the average wiki page length, will provide the peak bandwidth needed.
    - But... which is the average wiki page length?
      - There is no official data regarding that parameter.
      - A small page, as the *albufera* entry, needs 1130 KB.
      - A large page, as the *United States* entry, needs 7210 KB.
        - These are not the smallest and largest pages!!
    - This means that the peak bandwidth being needed is larger than 251 Gbps and smaller than 1602 Gbps.
      - 251 Gbps = 27774*1130*8 / 1000000 [acc/s*page size*bits/byte / KB per GB]
      - 1602 Gbps = 27774*7210*8 / 1000000

4. MediaWiki

- Could this bandwidth be provided by a single link to a single server?
  - No. It is currently impossible!!
  - The current best bandwidth for regular Internet providers is 2000 Mbps (*Google Fiber* in USA, August 2021).
  - In the worst case we need at least 801 channels of that bandwidth!!
    - 1602000 / 2000 = 801
    - So, at least 801 servers seem to be needed for supporting a peak Wikipedia workload!!

4. MediaWiki

- Therefore, we clearly need multiple servers in our MediaWiki system
  - Goal: To boost performance
  - Solution: Replication
    - To be analysed in Unit 5
    - But we need a lot of replicas (at least 800)!!! Many other problems arise in that case!!
      - How many usable endpoints exist to access that service?
      - How do we forward the incoming traffic to the other server nodes?
      - If a client request modifies a wiki page... how do we propagate that update to all other wiki page replicas?
        - Warning!!! This seriously compromises performance and consistency!!!
      - What happens if a server node fails while a request was being processed by that server?
      - With so many replicas, which is the probability that none of them fails in a given interval?

4. MediaWiki

- Let us see in the following pages how the LAMP server components of MediaWiki deal with high workloads.
- Those components are:
  - Apache server
  - MySQL server

4.1. Apache Server

- The Apache server waits for client HTTP requests.
- Those requests could be:
  - Static, reading and returning an existing document from a file.
  - Dynamic, leading to the execution of a PHP program.
- Service of static requests may be improved using...
  - More memory in servers, placing the requested documents in RAM.
    - Too expensive!!
  - Caching.
  - Reverse proxies.
    - A kind of server-side caching.
    - A set of intermediate processes (that play a server role for the clients) cache the contents of the most requested pages.
      - Those processes are indexed in some way, thus distributing client requests to the appropriate reverse proxy instance.
    - **Warning**: When we add other components, we should take care about their failures!!

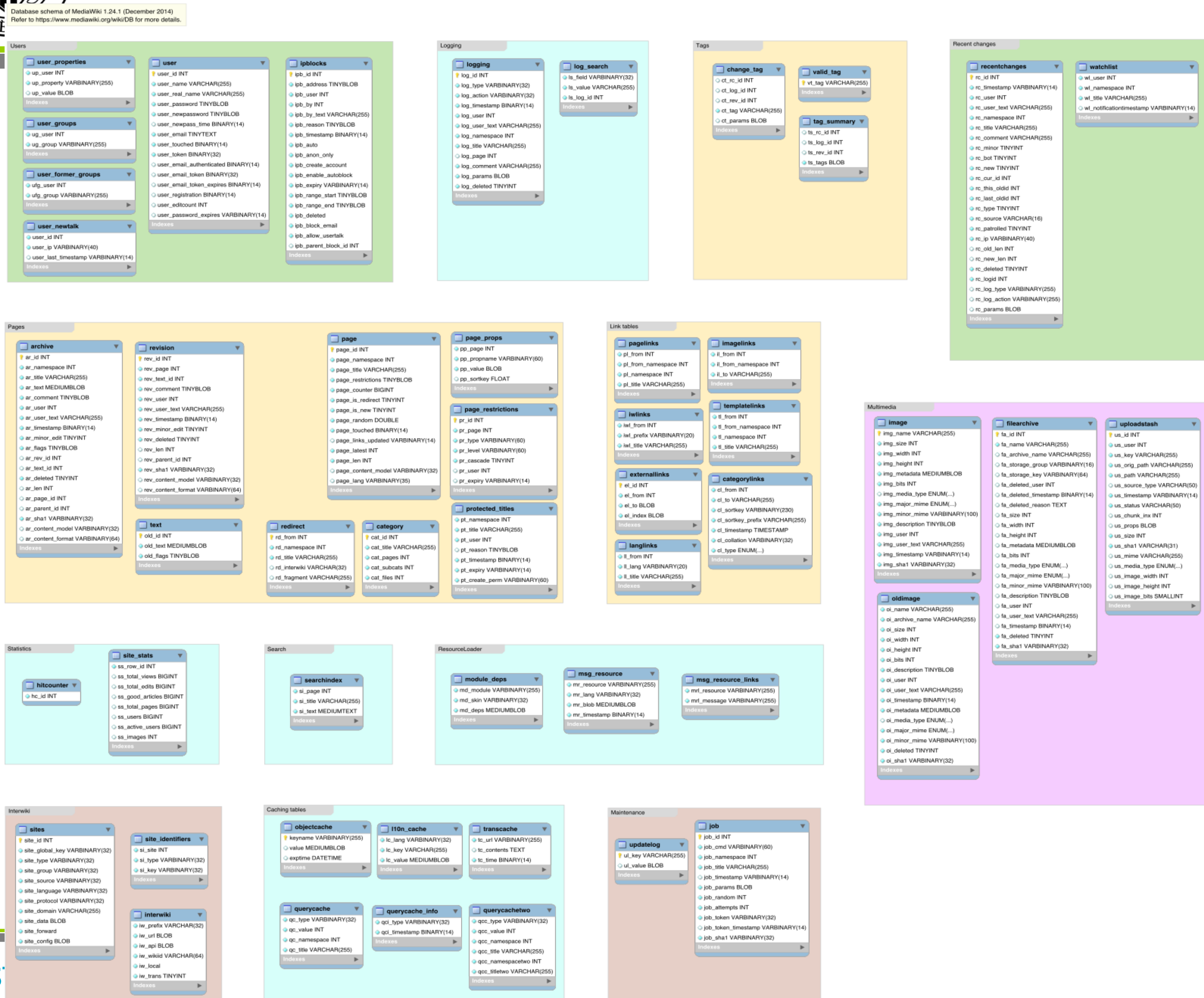
4.1. Apache Server

- Service of dynamic requests depends on the operation type:
  - Wiki page updates:
    - In the common case, a single copy of the page is firstly updated.
    - The update should be propagated to the remaining replicas and to the cached copies of the reverse proxy.
      - This is a non-trivial task!!!
      - Solutions: invalidation vs propagation.
  - Pages that depend on the requesting user:
    - Some kind of session management is needed.
    - Requests on the same session are forwarded to the same server replica.
      - Problems in case of failure of that server instance.
  - Keyword-based queries:
    - Can be managed using server-side caches.
    - Difficult to manage if the dynamic requests add other parameters to the query.
  - Queries for page-related information:
    - When a wiki page maintains some related content and that content can be searched in subsequent accesses.
    - Again, server-side caches may be useful.

4.2. MySQL

- MySQL maintains the persistent data of Wikipedia; i.e., all wiki page contents beside many other information related to users, page links, statistics of usage, recent changes,...
- In the first generation of MediaWiki, both servers (Apache and MySQL) were placed in the same computer.
- Other architectures were used afterwards...
  - Each server in a different node.
  - Soon, Apache servers were replicated. This demanded a distribution of the MySQL tables in multiple computers.
  - In 2014, the MediaWiki database consisted of 13 different databases maintaining around 50 tables.
    - Each database may be placed in a different computer.
    - Databases with high read workload are also replicated in multiple nodes.

4.2. MySQL (Schema in 2014)



Index

1. Introduction
2. Wikipedia nowadays
3. LAMP systems
4. MediaWiki
5. Wikipedia architecture
6. Conclusions
7. Learning results

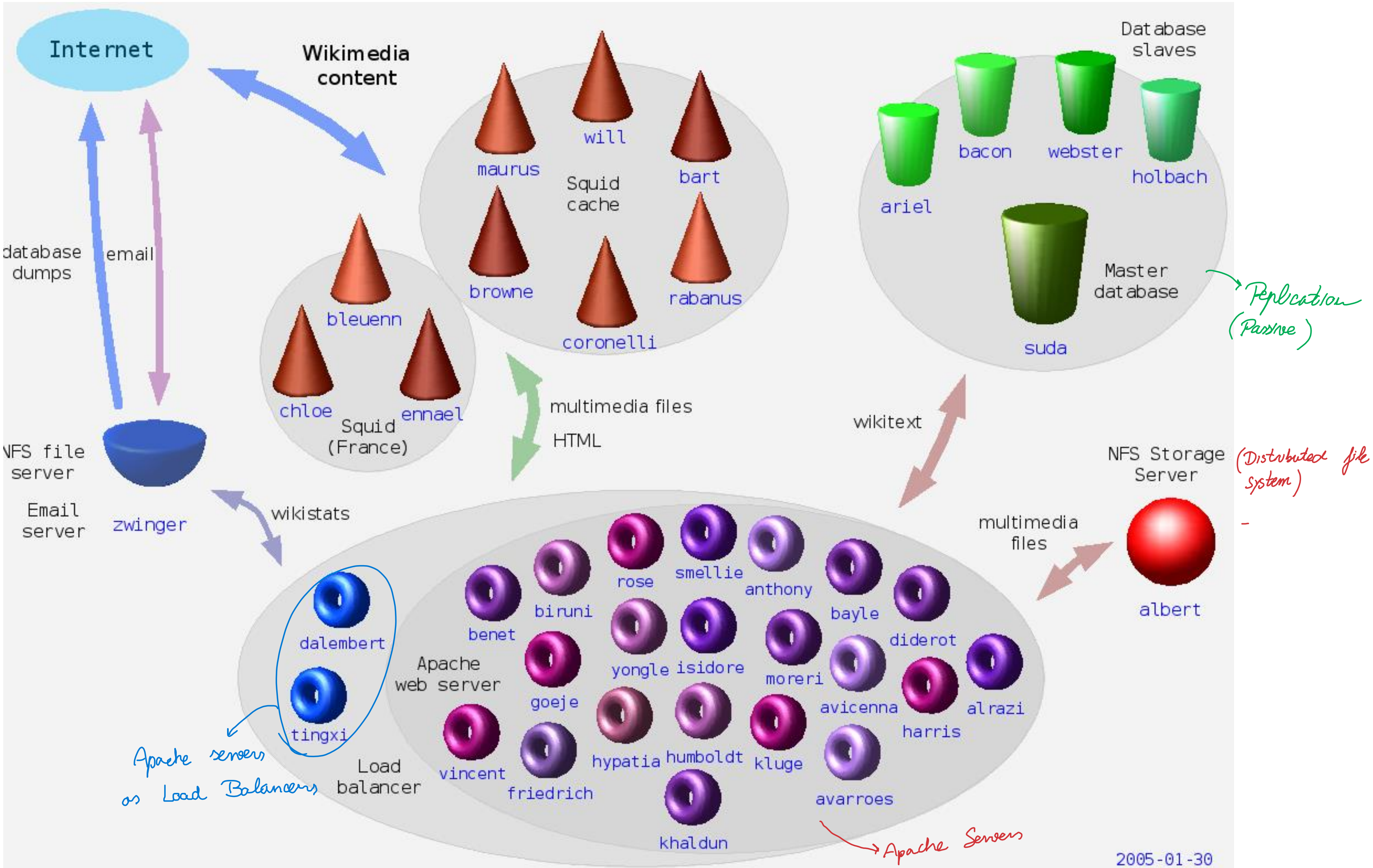
5. Wikipedia architecture

- As we have already said, in its first two stages, this system architecture consisted of:
  1. A single computer holding all MediaWiki components.
     - Unable to scale out.
  2. Two computers, each one holds a different server.
     - Apache + PHP modules
     - MySQL
- The Wikipedia success forced the Wikimedia Foundation to develop more complex architectures, replicating and distributing the main components.
  - Three examples are shown in the next slides:
    - Architecture in 2005
    - Architecture in 2010
    - Component architecture

5.1. Architecture in 2005

- It consists of:
  - Two reverse proxy services (Squid), with multiple nodes in each one.
  - Five nodes in a distributed MySQL service following a primary-backup replication model.
    - The primary replica directly manages read and write operations while backup nodes also serve read requests and receive the updates from the primary.
  - Many Apache servers, with two associated load balancer nodes.
  - Multimedia resources are stored as regular files in a single NFS server.
  - Summary: 39 computers are used in this deployment.

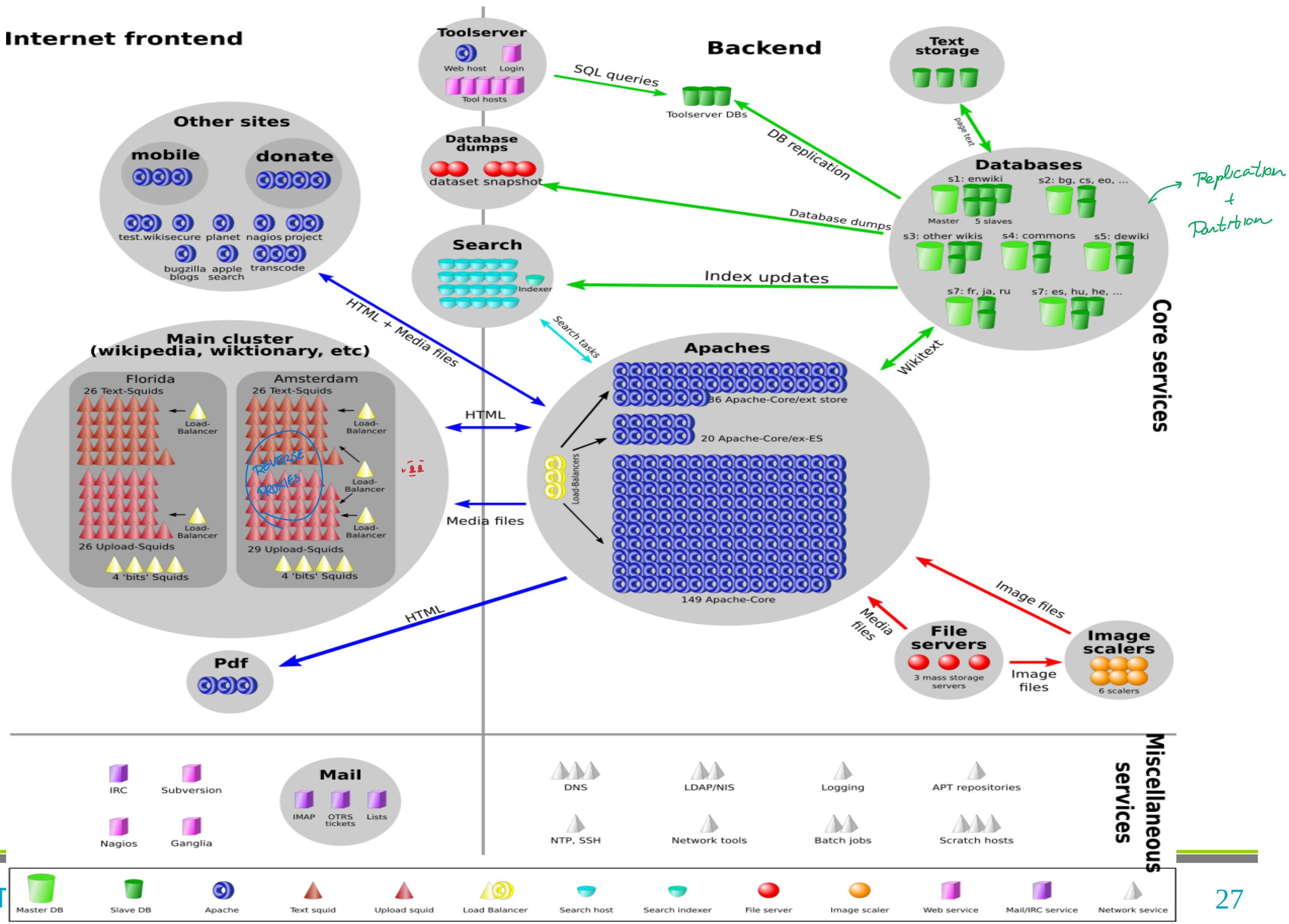
5.1. Architecture in 2005



5.2. Architecture in 2010

- The system is much larger than in 2005.
- Multiple complementary services have been needed.
- There are many more instances in each component (reverse proxies, load balancers, Apache servers, MySQL nodes...).
- There are multiple kinds of Apache servers, depending on the resources being requested (wiki pages in HTML, PDF files, versions of wiki pages for mobile devices, etc.).

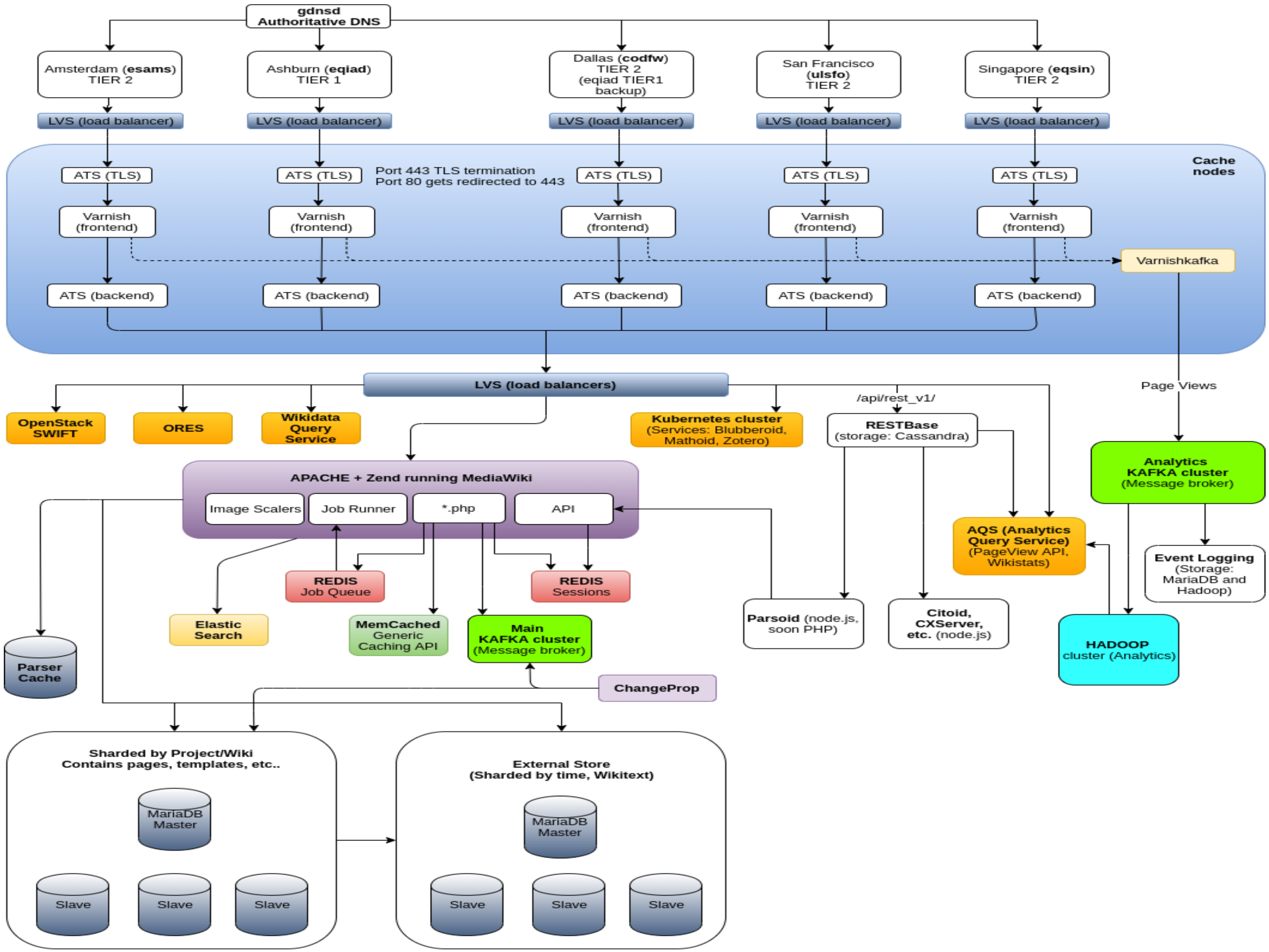
5.2. Architecture in 2010



5.3. Component architecture

- Both previous examples are focused on how many nodes have been used in the system deployment.
- The next slide is centred in inter-component dependences.
  - That defines the distributed system architecture.
    - Centred in system functionality.
    - More recent → Additional functionality.
  - The two previous "architectures" are deployment cases.
    - With a simpler architecture, since Wikipedia has added more functionality in each of its evolution stages.
- Please, revise the student guide in order to understand the functionality of each component.

5.3. Component architecture (2020)



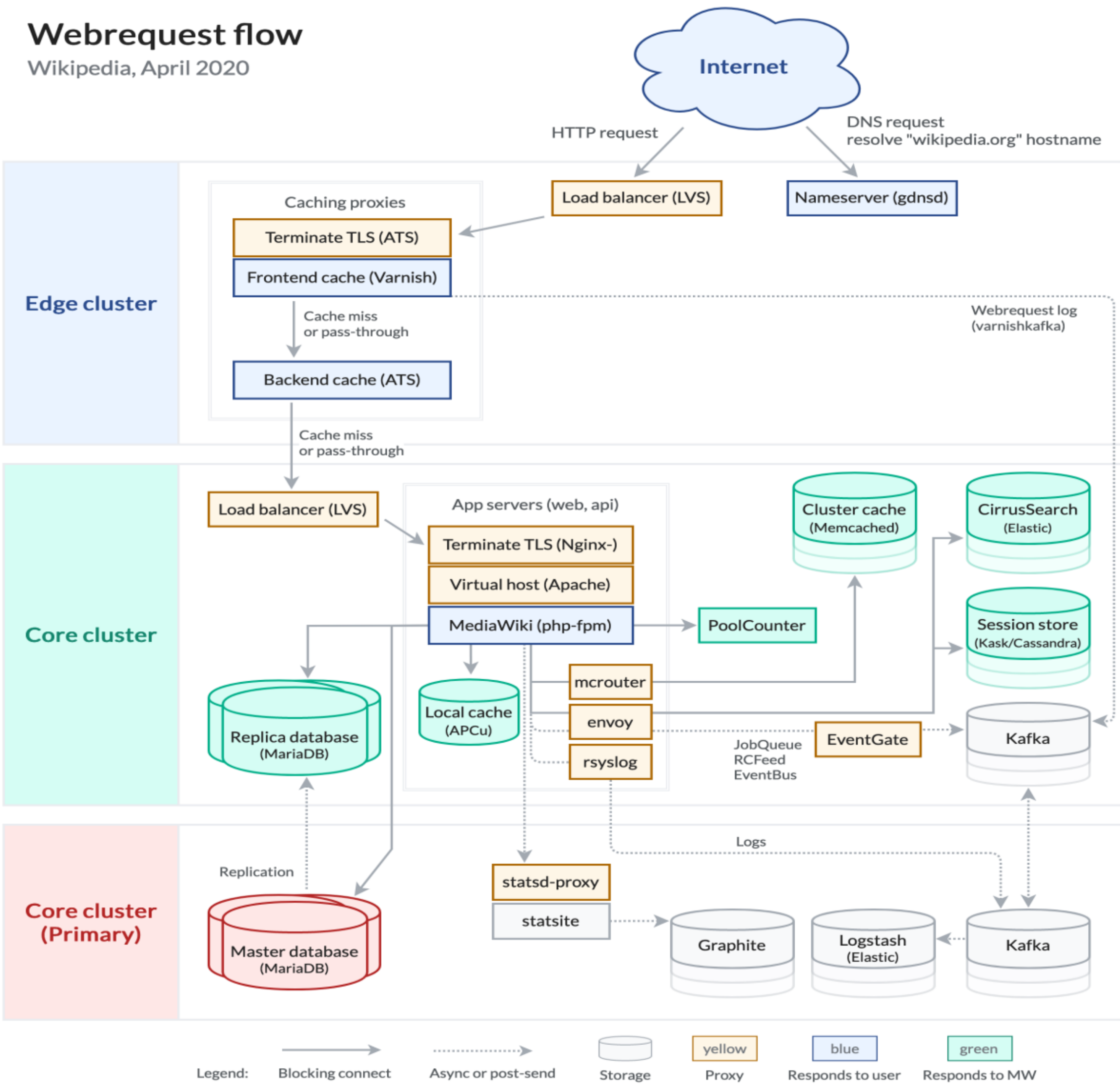
Text

5.3. Component architecture (2020)

- But that previous page still shows some deployment aspects…
  - Its two upper layers…
    - Load balancers
    - Reverse proxies + caches
  - …are deployed in five different datacentres.

- The next page shows a schematic view of that same architecture, without those details.

5.3. Component architecture (2020)



Index

1. Introduction
2. Wikipedia nowadays
3. LAMP systems
4. MediaWiki
5. Wikipedia architecture
6. Conclusions
7. Learning results

6. Conclusions

- There are multiple examples of successful distributed services that are currently used by millions of people.
- Their architecture demands a careful design that should deal with many challenges.
- The workload being supported by these services should be distributed among as many nodes as possible.
  - Thus, each server node is responsible of only a small part of the load.
- To ensure continuity of service, server nodes should be replicated.
  - In case of failure of a replica, other replicas share its work and complete its already started service requests.
- Caching and reverse proxies are other approaches to improve service capacity.

Index

7. Learning results

- At the end of this unit, the student would be able to:
  - Identify some general examples of large (and, as such, highly scalable) distributed services.
  - Identify some of the problems and challenges to be managed in those services: caching strategies, request forwarding, data persistence, data distribution, data consistency, server failures, service continuity...
  - Identify several approaches to deal with those challenges: workload distribution, service replication...