

Information Architecture

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Defining Information Architecture
- Elements
- Process
- Deliverables
- LBAW: "A3. Information Architecture" artifact

Defining Information Architecture

Context

- We live in information societies, where information services and systems are pervasive and abundant in everyday life, and increasingly essential to the operation of multiple businesses.
- A growing volume of information is available to us, which resulted in a problem of managing attention in the face of a decreasing "signal-to-noise" ratio.
- The proliferation of applications and electronic devices, multiplied the number of channels through which we can access the same information. This resulted in a decoupling of information from its container, no longer 1-1.
- Both problems — information overload and the multiplication of access channels — are tackled by the field of information architecture.

Places Made of Information

- Information products and services are perceived by as "places made of information", where people go for different tasks: learn, shop, connect with other people, etc.
- User experience in these places is defined by a familiar vocabulary consisting of labels, menus, descriptions, buttons, links, visual elements, and content.
- Different uses of this language will make them "distinct" places to the user, and will help them (or not!) be more efficient in accomplishing their tasks and goals.
- Information architecture considers these information-rich spaces and its design for maximizing their effectiveness, whatever the users' goal.

Information Architecture

- Multiple definitions, each highlighting a particular aspect.
- Richard Wurman (1996) emphasizes **organization** and **presentation**:
 - (1) *The individual who organizes the patterns inherent in data, making the complex clear.*
 - (2) *A person who creates the structure or map of information which allows others to find their personal paths to knowledge.*
 - (3) *The emerging 21st century professional occupation addressing the needs of the age focused upon clarity, human understanding, and the science of the organization of information.*
- Rosenfeld and Morville (2015) introduce **multiple perspectives**:
 - *The structural design of shared information environments.*
 - *The synthesis of organization, labeling, search, and navigation systems within digital, physical, and cross-channel ecosystems.*
 - *The art and science of shaping information products and experience to support usability and findability, and understanding.*
 - *An emerging discipline and community of practice focused on bringing principles of design and architecture to the digital landscape.*

Context of Information Architecture

- The evolution of the web and digital tools brought many opportunities and challenges to users and designers.
- The pervasiveness of digital spaces in everyday life increasingly blurs the distinction between physical and digital.
- Information Architecture is needed to organize these spaces, specifically:
 - Connecting information objects and intended users;
 - Identify concepts and pathways for access and navigation;
 - Creating tools and systems for people to organize information;
 - Connect various information spaces, applications, platforms, and channels.
- Tackling these challenges requires skill from multiple fields, e.g. usability, design, information.
- User Experience (UX) Design is an umbrella under which these areas converge.

Related Disciplines

- Usability Engineering — focus on human computer interaction and how user interfaces can allow the users to accomplish their tasks.
- Information Science — a broad interdisciplinary field focused on theories, applications, and technologies related to the creation, organization, retrieval and use of information.
- Human Factors Engineering — ergonomics and physical factors in designing products, processes, and work environments.
- Visual Design — aesthetics and communication of information, using visual language elements such as colors, shapes, spacing, alignment, etc.
- Interaction Design — how a system works in response to user inputs, i.e. the dynamics between the user and the system.

Summary

- Information architecture is a central element in user experience design.
- It deals with the process of planning, designing and building information spaces.
- Its goal is to improve information access, management and use, through the design of meaningful, functional and effective information spaces.
- Is a field of growing importance, as frontiers between physical and information spaces blur.
- Information architecture is relevant for an Informatics Engineer, because:
 - Improves work in the context of multidisciplinary teams, i.e. understand the language and the artefacts;
 - Contributes to the development of better prototypes and products.

Elements of Information Architecture

Elements of Information Architecture

- The main components of an information architecture include:
 - Organization systems - how information is categorized;
 - Labeling systems - how information is represented;
 - Navigation systems - how you can move through the information;
 - Searching Systems - how information can be searched for.

Organization Systems

Organizing Information Spaces

- The way things are organized impacts its meaning, i.e. the way they are perceived.
- Well-organized information is easier for people to find and work with.
- Organizing things is hard, it is necessary to deal with ambiguity, heterogeneity, different perspectives, politics, etc.
- An organization scheme defines the shared characteristics of items and the grouping of those items.
- They can either be exact organization schemes or ambiguous organization schemes.

Exact Organization Schemes

- In exact organization schemes, information elements are divided into well-defined and mutually exclusive sections.
- These scheme are relatively easy to design and maintain.
 - **Alphabetical scheme**, where alphabetical ordering is used to layout information items, examples include: person directory, services directories, libraries.
 - **Chronological scheme**, where time-based factors are central in organizing information items, classic examples include: calendars, tasks lists, news (?).
 - **Geographical scheme**, where place is the key factor in organizing information items, examples include: house or rental listings, transportation.

Ambiguous Organization Schemes

- Language and concepts are ambiguous in nature.
- Ambiguous organization schemes are important because people don't always have an exact definition of what they are looking for, but harder to design and maintain.
 - **Categorical schemes**, where topics, themes are the factors considered in organizing information items.
These properties have subjective interpretations, depending on the knowledge, viewpoints, etc of users.
Examples: newspapers, academic courses, books.
 - **Hierarchical schemes**, where information items are organized according to value, from most to least or vice-versa. The key aspect is that it requires value to be assigned to the information. Examples: social networks, search engines.
 - **Audience-specific schemes**, make sense when multiple audiences exist. This can result in breaking an information space into smaller audience-specific sub-spaces, or including more or less information depending on the audience. Examples: intranets, academic services.

Labeling Systems

Labeling Systems

- Labeling is a form of representation.
- Just as words represent concepts and ideas, labels represent parts of information in information spaces. Example: "About" or "Contact Us" are well-established labels.
- Labels can either be textual or iconic (more ambiguous).
- Textual labels include: contextual links, headings, navigation systems choices.
- Labels can be designed looking at existing information environments (what is used in other contexts or by others in similar contexts) or search logs (what language users use for searching).

Navigation Systems

Embedded Navigation Systems

- The purpose of navigation is to help users move around.
- Navigation systems provide context and a sense of control to users as they explore.
- A good navigation system should help the user answer the following questions:
 - Where am I?
 - What can I do?
 - Where can I go from here? (up, down, parallel)

Types of Navigation Systems

- There are various types of navigation systems, three common ones are:
 - **Global navigation systems**, it is intended to be present in every page, often implemented as a navigation bar at the top, allow direct access to key areas of the system. A central element in the overall usability of a space.
 - **Local navigation systems**, complement the global system and allow users to explore and navigate the immediate subsection.
 - **Contextual navigation systems**, support navigation through the association of concepts that exist in the content being presented. Examples include "see also" or link to related items in different areas.

Types of Navigation Systems

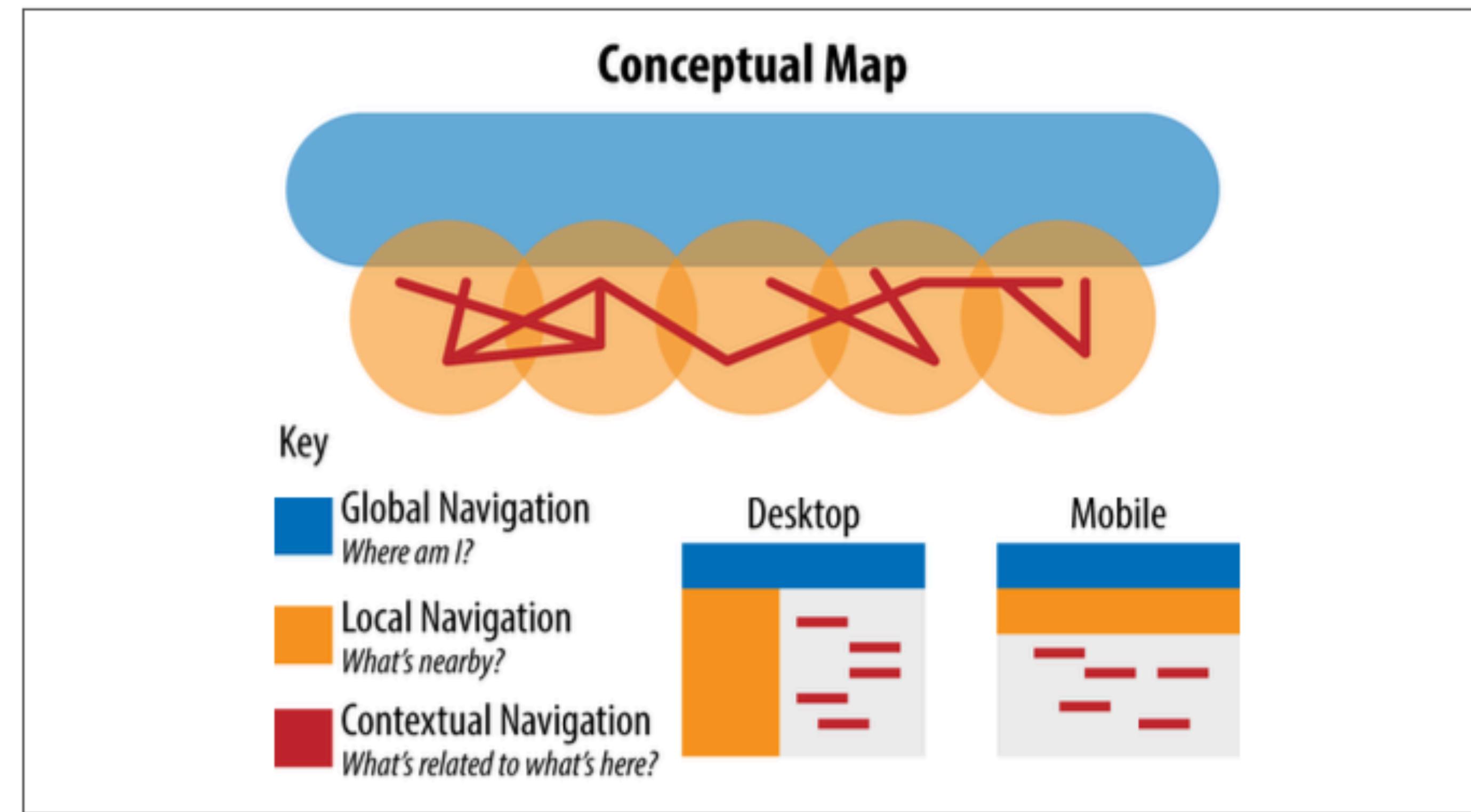


Figure 8-1. Global, local, and contextual embedded navigation systems

Supplemental Navigation Systems

- Supplemental navigation systems are external to the basic hierarchy of a system and provide complementary ways of finding content and completing tasks.
- Examples include: sitemaps, site indexes, breadcrumb trails, FAQs, tutorials, search.
- **Sitemaps** provide a broad view of the content in the system and facilitates random access to individual items.
- **Breadcrumbs** are dynamic and represent the path followed by the user or, alternatively, the location of the content within the hierarchy.
- **Search**, is a central mechanism for navigation, it provides users a direct access to the content they are looking for.

Sitemap

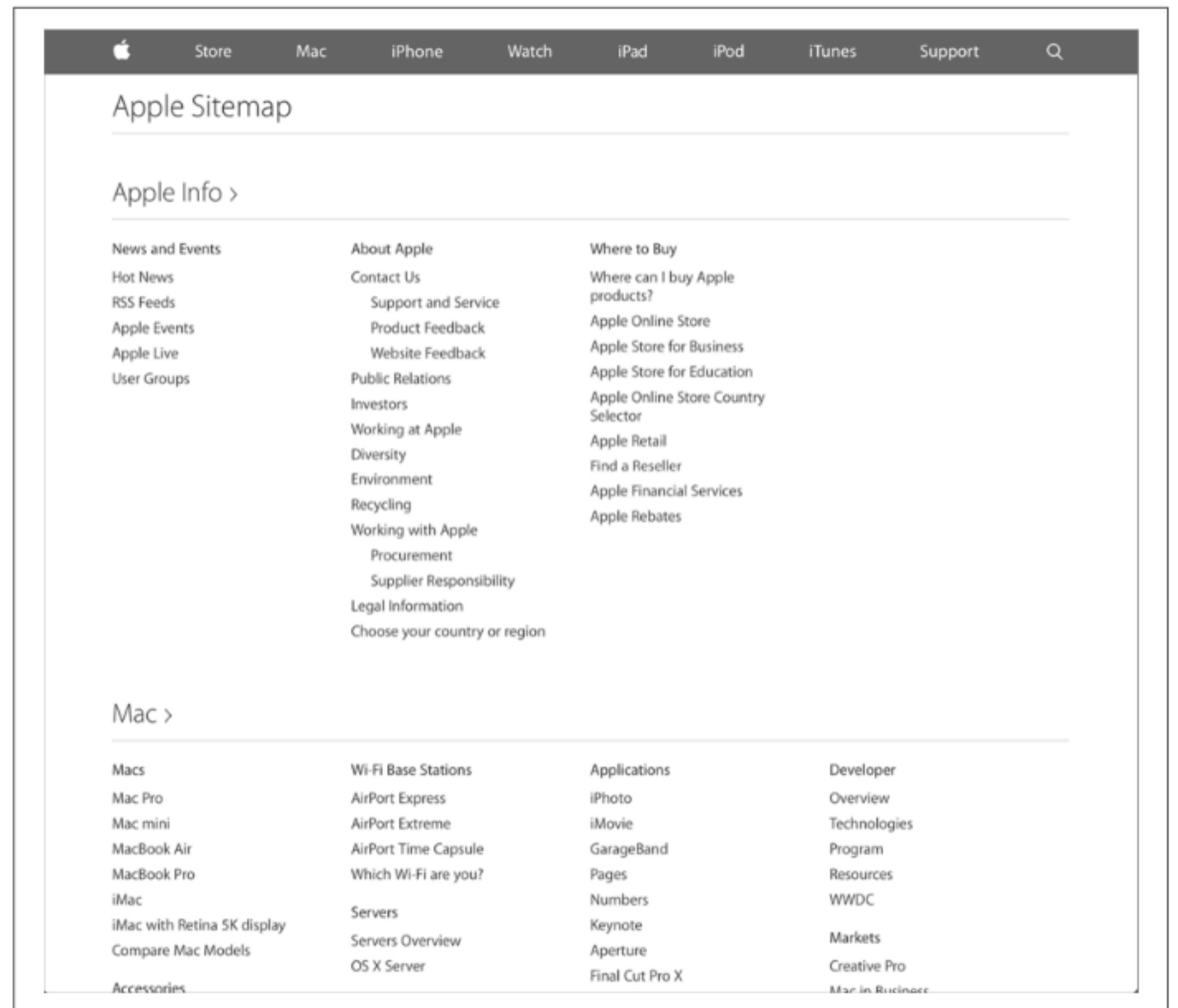


Figure 8-16. Apple's sitemap

Advanced Navigation Solutions

- Advanced solutions for navigation, include:
 - **Personalization**, serve information based on a model of the user, e.g. based on behavior, navigation, profile. Example: personalized recommendations in a shopping website.
 - **Customization**, give the user direct control over content and navigation options, e.g. visible options, preferences. Example: ordering or labels in a webmail application.
 - **Visualization**, provide navigation mechanisms based on visual properties of information items. Example: a visual search engine in a e-commerce website.
 - **Social navigation**, organize content access or navigation based on users' input. Example: ranking content by votes from other users (e.g. Reddit), or using users' tags to provide navigation features (e.g. Stackoverflow).

Search Systems

Search Systems

- Search is an advanced navigation mechanism.
- An apparently simple interface masks a complex system (covered later).
- Opting for a search system requires evaluation considering multiple issues, including:
 - Amount of content in the "information space".
 - Time and know-how to optimize the search system.
- Users expect search: it's a familiar tool, where users control of the vocabulary to find information, and can 'cut across' the existing structure.
- Multiple decisions are required, including which content fields to index, what text processing to perform, which additional filtering criteria to consider.

Search User Interface

- In designing the user interface, it is necessary to decide which information components to include in the search results.
- Complementary sorting options can be provided, e.g: relevance ranking, chronological, alphabetically, popularity, etc.
- Faceted search is an advanced mechanism commonly used in stores and complex information contexts.

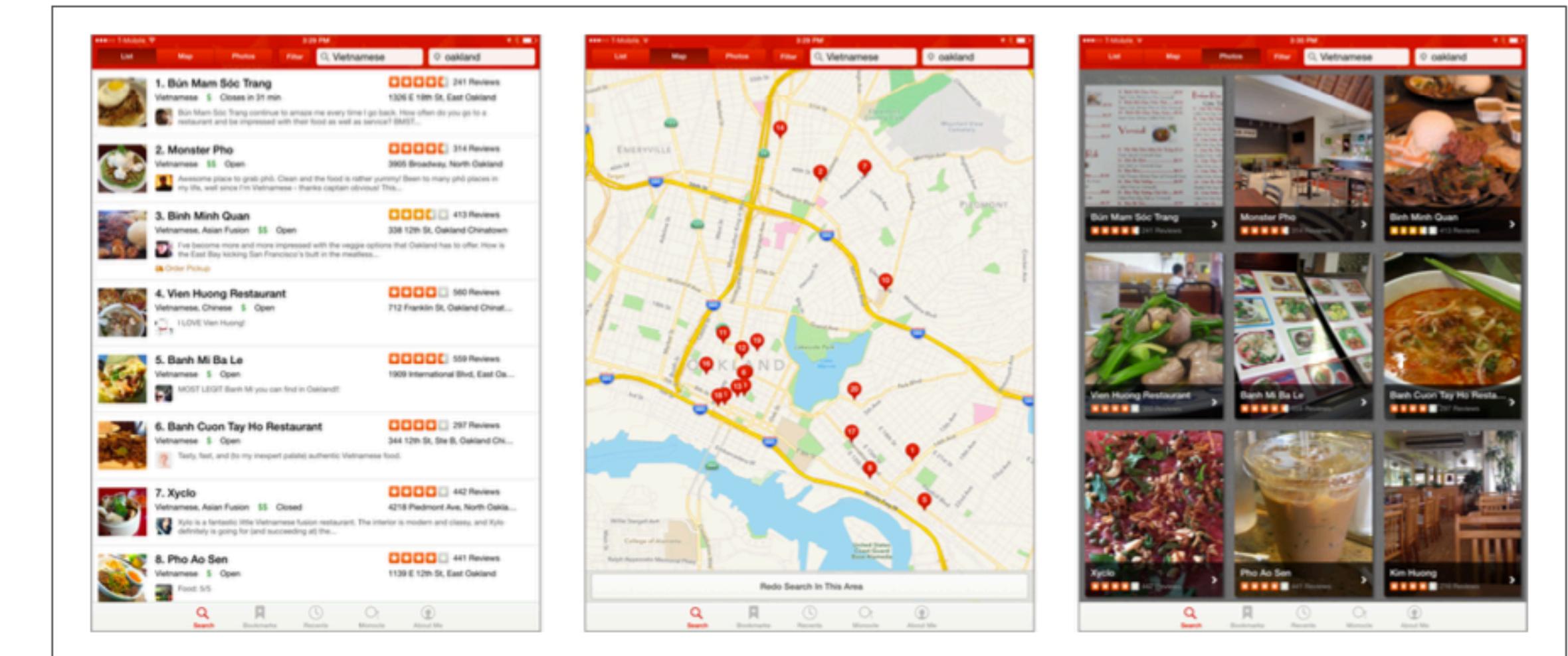


Figure 9-11. The Yelp iPad app allows users to select three different ways of viewing search results: as listings, as locations on a map, or as images

Faceted Search

The screenshot shows the Forrester website's search results page for the query "user experience". The top navigation bar includes links for research & data, analysts, events, consulting, leadership boards, help, log in, and register. The search bar shows the query "user experience". Below the search bar, there are buttons for "Reports" and "Advanced Search". A link to "Turn Highlighting On" is also present. The search results are categorized under "Reports" (2083 results). The results are sorted by Most Relevant. The first result is a report titled "REPORT: Best Practices For Chinese Mobile User Experience" by Samantha Jaddou, dated December 18, 2014. The second result is a report titled "REPORT: Case Study: Cornerstone OnDemand Makes End User Experience The Priority For IT Operations" by Eveline Oehrlich, dated September 26, 2012. The third result is a report titled "REPORT: Reviewer's Guide: Website Top 10 User Experience Review 8.0" by Adele Budovsky, dated November 14, 2011. The fourth result is a report titled "REPORT: Best And Worst Of Website User Experience, 2011: Canadian Banks" by Ron Rogowski, dated June 15, 2011. The sidebar on the left provides facets for refining the search results, including categories like Reports, Playbooks, Blogs, Data Surveys, Events, Charts & Figures, Everything, Date Range, Primary Role, Methodology, and more.

Figure 9-21. Forrester contextualizes search results for the query “user experience”

Best Practices in Search

- Support autocomplete
- Support query operators
- Provide context
- Allow query reformulation
- Support additional filters
- Support alternative rankings
- Customize snippets
- Highlight matched search terms

The screenshot shows the search results for 'edward snowden' on The New York Times website. The search bar at the top contains the query 'edward snowden'. To the right of the search bar are buttons for 'Go' and 'Most Popular Searches'. On the left, there are filter options: 'Date Range' (set to 'All Since 1851'), 'Sort by' (set to 'Relevance'), and 'Result Type' (set to 'All Types'). Below these filters, the search results are listed:

- Chip Maker to Investigate Claims of Hacking by N.S.A. and British Spy Agencies**
encryption codes. The claims — reported on a website called The Intercept — were based on documents from 2010 provided by Edward J. Snowden, the former N.S.A. contractor. The American and British intelligence
February 21, 2015 - By MARK SCOTT - World - Print Headline: "Chip Maker to Investigate Claims of Hacking by N.S.A. and British Spy Agencies"
- Canada Agency Monitors File-Sharing, Reports Say**
downloaded them — as part of an antiterrorism effort involving the United States and other allies, a document leaked by Edward J. Snowden indicates. The project, to detect possible extremists by monitoring visits
January 29, 2015 - By IAN AUSTEN - World - Print Headline: "Canada Agency Monitors File-Sharing, Reports Say"
- British Court Says Spying on Data Was Illegal**
and American intelligence agencies have used a program known as Prism — first revealed by the former N.S.A. contractor Edward J. Snowden — and others like it to gain access to individuals' Internet
February 07, 2015 - By MARK SCOTT - World - Print Headline: "British Court Says Spying on Data Was Illegal"
- British Spies Seized Emails to Reporters**
from a single intercept — is contained in a cache of British documents that are among the classified trove leaked by Edward J. Snowden, the former contractor for the National Security Agency. There is no
January 20, 2015 - By JAMES GLANZ - World - Print Headline: "British Spies Seized Emails to Reporters"
- F.B.I. Is Broadening Surveillance Role, Report Shows**
declassifications about government surveillance activities in response to leaks by the former intelligence contractor Edward J. Snowden. The report

Figure 9-32. All aspects of the search are restated as part of these search results

Information Architecture Process

Information Architecture Process

- The design of information environments requires interdisciplinary teams, including interaction designers, software developers, content strategists, usability engineers, and other experts.
- An information architecture process is structured in four general activities:
 - **Research**, understands users, content, and context;
 - **Design**, specify the information architecture, creating detailed sitemaps, wireframes, etc;
 - **Implement**, solutions that adhere to the design and specifications produced;
 - **Evaluate**, and improve the system throughout its life cycle.

Research and Research Methods

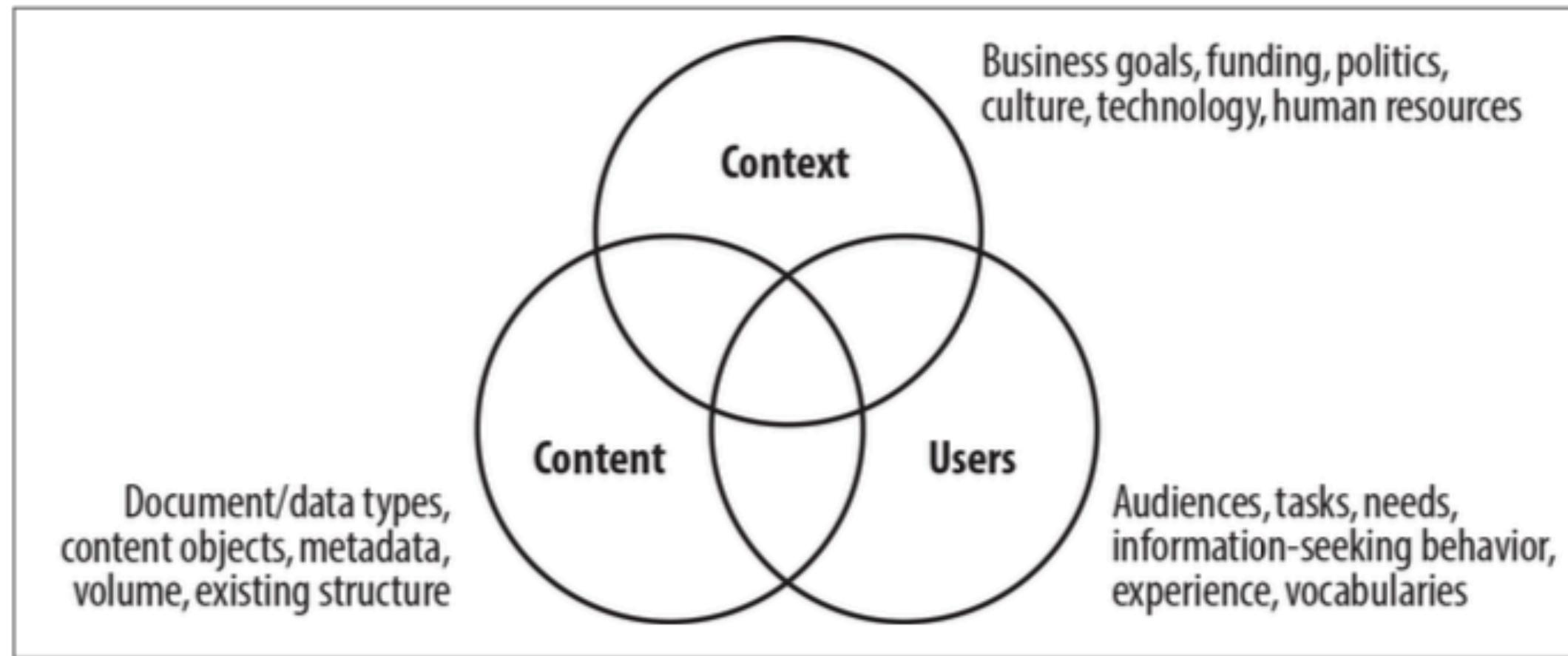


Figure 11-2. A balanced approach to research

Context	Background research	Presentations and meetings	Stakeholder interviews	Technology assessment
Content	Heuristic evaluation	Metadata and content analysis	Content mapping	Benchmarking
Users	Search log and clickstream analysis	Use cases and personas	Contextual inquiry	User interviews and user testing

Figure 11-3. Tools and methods for research

Information Architecture Deliverables

Information Architecture Deliverables

- Deliverables are essential for any well-structured process.
- In multidisciplinary contexts, they work as "anchors" between teams and different project phases.
- Visual diagrams contribute to define:
 - Content components, i.e. what constitutes a unit of content, and how these are grouped.
 - Connections between components, i.e. how components are linked.
- Deliverables can provide multiple views, at different levels and for different audiences, over the information architecture of a system.
- Commonly used deliverables are sitemaps and wireframes. Others include wireflows, content inventories, prototypes, design mockups, etc.

Sitemaps

Sitemaps

- Sitemaps show the relationship between information elements, such as pages, and can be used to portray organization, navigation, and labeling systems.
- It can be used to portray the content organization, the navigation system, and the labeling systems. It provides a condensed view for both developers and users.
- High-level sitemaps are usually the result of a top-down design process.
- Sitemaps can be used to map specific areas or parts of a complex information environment.
- Adopt vector based tools that support collaboration.

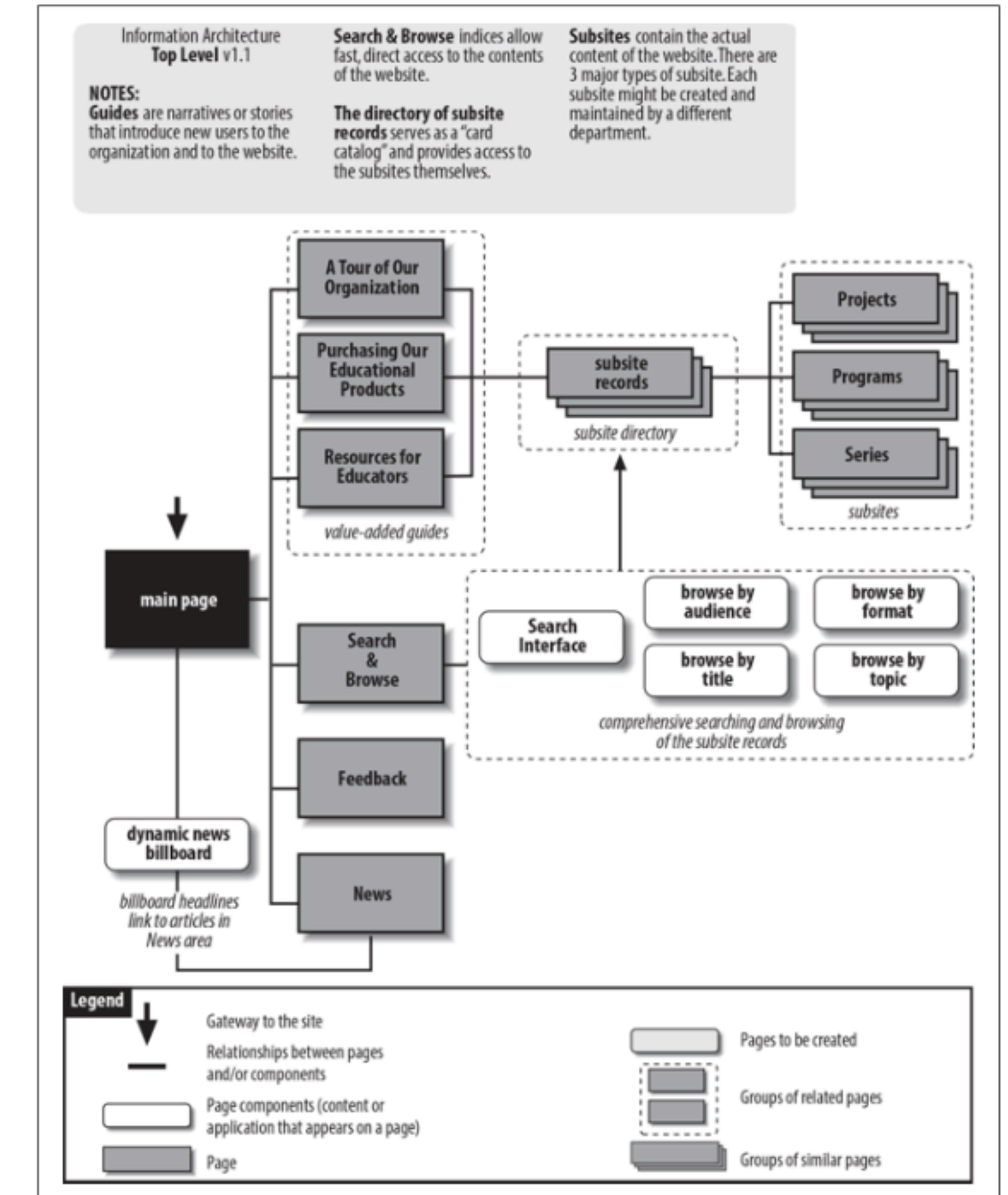


Image from: Rosenfeld et al.
*Information Architecture -
 For the Web and Beyond,*
 O'Reilly (2015)

Figure 13-1. A high-level sitemap

Wireframes

Wireframes

- Wireframes depict how an individual page or template should look from a structural and architectural perspective.
- Wireframes forces us to consider issues such as the location of navigation systems, the content hierarchy, content grouping, what to include and what to discard in each view of the system, etc.
- Wireframes are typically created for the system's most important pages or screens.
- They are also used to describe templates that are consistently applied to many pages.
- They are also a good way to explore the impact of different screen sizes on content layout.
- Wireframes can vary in their "level of fidelity", from low-level to high-level, depending on the stage of the development lifecycle.

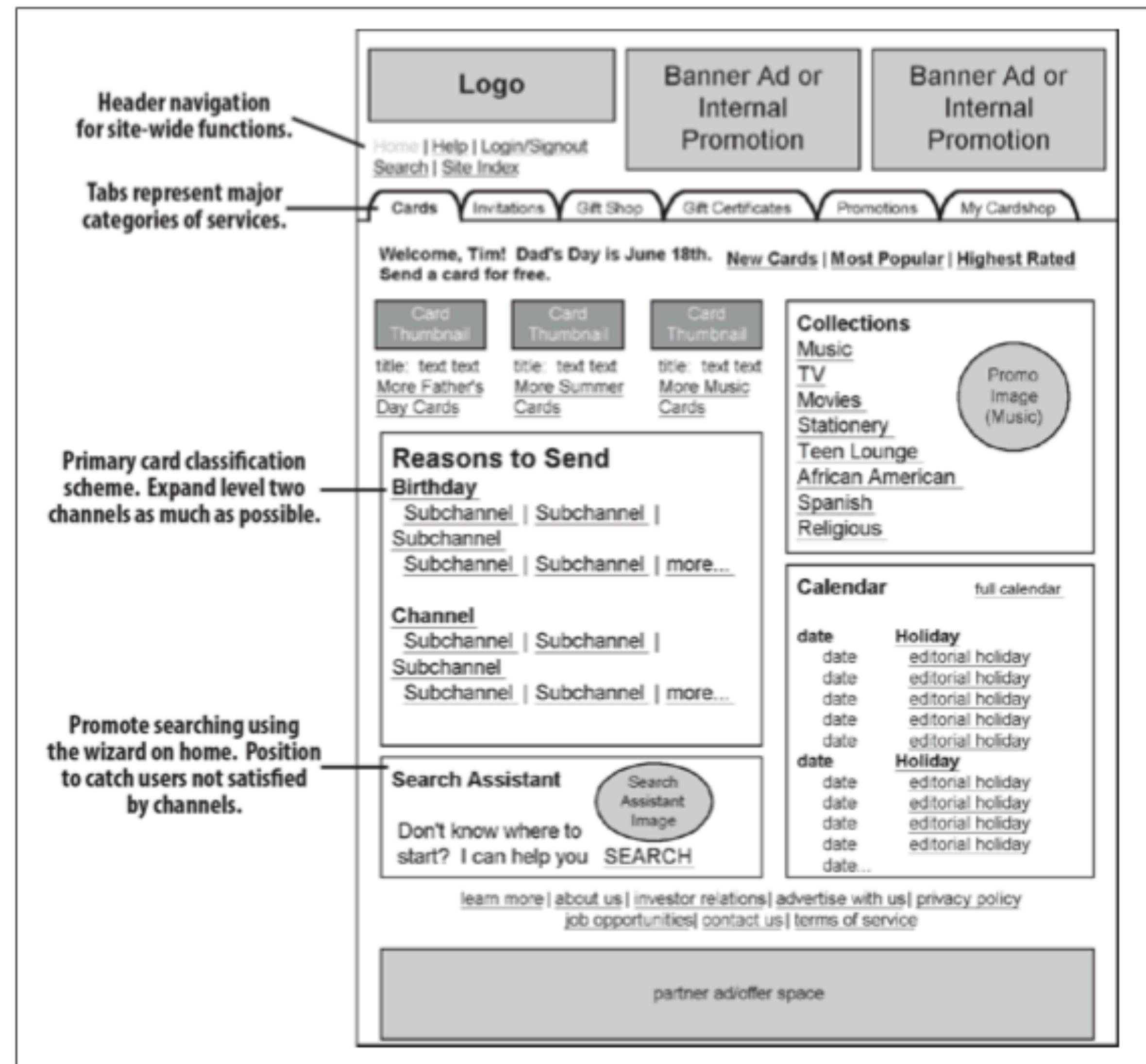


Image from: Rosenfeld et al.
Information Architecture -
For the Web and Beyond,
O'Reilly (2015)

Figure 13-11. A wireframe of the main page of a greeting card site

The Looks

Defaults to look #1. Product thumbnails are shown for products related to the look shown. When user rolls over a product image, the name of the product is revealed. (Either a tooltip or the name is highlighted in the text above.) When user clicks the product thumbnails or the product name in the text, they are taken to the product detail page. Previous and Next arrows allow the user to go to the next/previous look. These cycle in a continuous loop. The user may also click on the partial preview above or below to go to the next look.

Wireframe

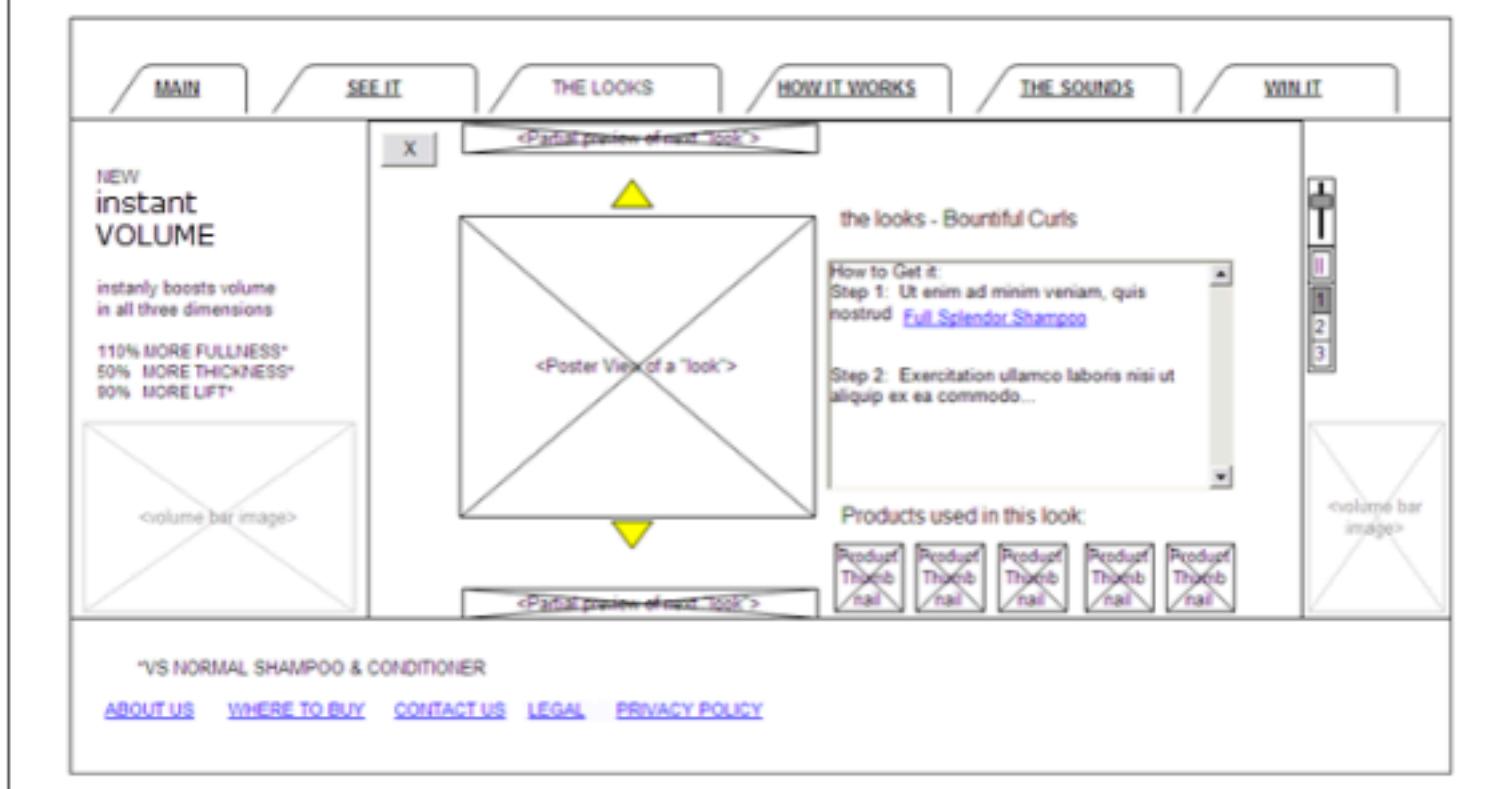


Figure 13-13. A low-fidelity wireframe; note that the focus is on layout of content and visual elements over content accuracy (wireframe developed for ProQuest LLC; reproduced with permission of ProQuest LLC—further reproduction is prohibited without permission)

Figure 13-14. A medium-fidelity wireframe by Chris Farnum and Katherine Root; more detail, more explanation, and more unique content (wireframe developed for ProQuest LLC; reproduced with permission of ProQuest LLC—further reproduction is prohibited without permission)

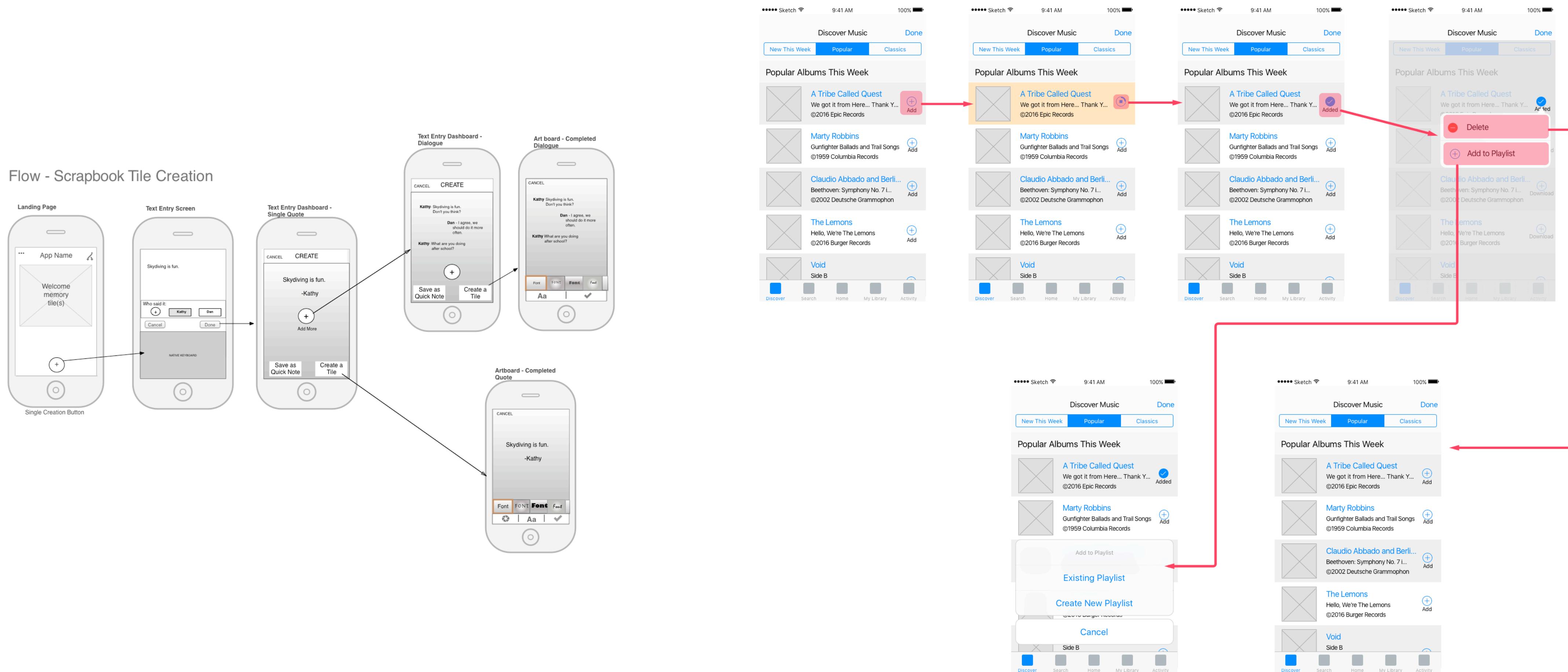
Figure 13-15. A high-fidelity wireframe (wireframe developed for ProQuest LLC by Chris Farnum; reproduced with permission of ProQuest LLC—further reproduction is prohibited without permission)

Wireflows

Wireflows

- Wireflows combine wireframes and flowcharts to provide both a view of page-level layout ideas (structure), and document complex workflows and user tasks (flux).
- Useful when documenting systems with few pages that dynamically change its content and layout based on user interaction. This is an increasingly common pattern, particularly in mobile applications.
- Wireflows document interaction.

Wireflows



Images from: Laubheimer P. Wireflows: A UX Deliverable for Workflow and Apps. Nielsen Norman Group (2015)

A3. Information Architecture

A3. Information Architecture

- The Information Architecture artifact presents a brief overview of the information architecture of the system to be developed, and has the following goals:
 - Help to identify and describe the user requirements, and raise new ones;
 - Preview and empirically test the user interface of the product to be developed;
 - Enable quick and multiple iterations on the design of the user interface.
- This artifact enables a brief exploration of the information architecture of the system to be developed, in particular the identification of the content, how it is organized and made available, and how it is presented.
- Includes two elements: sitemap and wireframes.

A3. Sitemap

→ Overview of the information architecture from the viewpoint of the users.

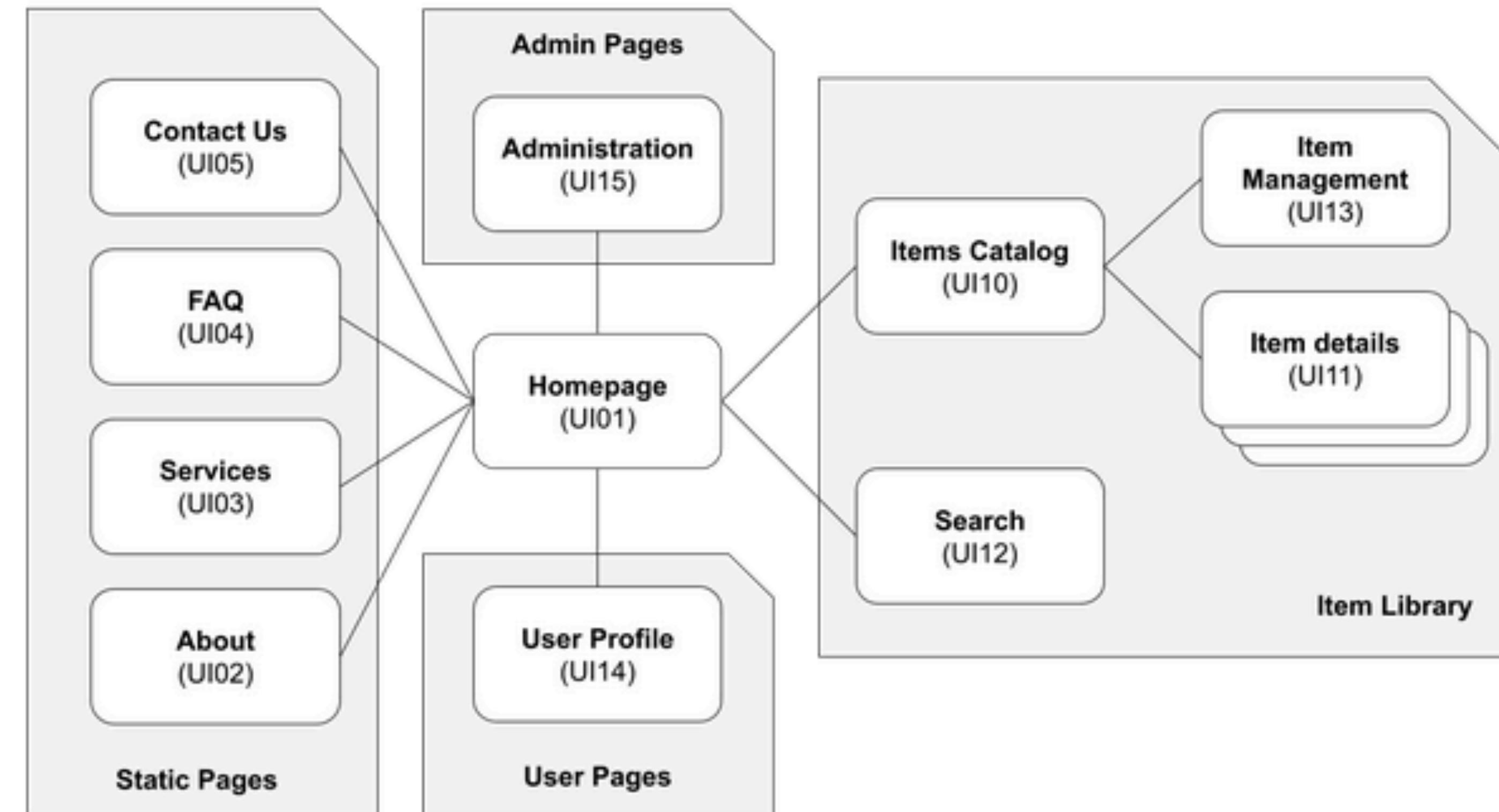


Figure 1: MediaLibrary sitemap.

A3. Wireframes

→ Description and prioritization of the functionality and content of, at least two, main user interfaces.

The wireframe illustrates a user interface for viewing item details. It includes a header with a logo and navigation links, a breadcrumb trail, a search bar, and a title with an edit link. The main content area displays item information, a placeholder for an image, and sections for description and loan history. A sidebar on the left provides numbered annotations for specific features:

- 1. Breadcrumbs to help the user navigate
- 2. Direct access to the search feature
- 3. Edit is available to the item owner
- 4. Request for loan is available except to the item owner

MediaLibrary Logo

About Services FAQ Contact John Doe ▾

Home ▶ Items ▶ Pulp Fiction ①

Keyword-based search... ②

Pulp Fiction [edit] ③

[favorite | request for loan] ④

Item information

Name: Pulp Fiction
Year: 1994
Owner: John Doe
Type: DVD
Average Rating: ★★★★ (4.2)
Notes: —

Description Loan History

>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed vel elit laoreet ligula dignissim ultricies. Nunc ac aliquet erat. Duis ultricies nibh vitae quam ultrices, eu porta nisi volutpat. Maecenas tristique porttitor facilisis. Suspendisse hendrerit semper laoreet. Nulla mauris felis, blandit auctor nisi ut, molestie luctus augue.

© MediaLibrary

Figure 2: Item Details (UI11) wireframe.

A3. Checklist

A3. Information Architecture		
Artefact	1.1	The artefact reference and name are clear
	1.2	The goal of the artefact is briefly presented (1, 2 sentences)
Sitemap	3.1	The sitemap is included
	3.2	Standard notation is used (lines, boxes and stacks of boxes)
	3.3	The sitemap identifies pages, not functions or features
	3.4	Only main links between pages are included
	3.5	Home page is at the top/center
	3.6	Each page has a unique reference
	3.7	Login is presented as a page (may be a page element)
	3.8	Search results page is included
	3.9	About page is included
	3.10	View/Edit own profile is included
	3.11	Administration area and pages are included
	3.12	View project / View question / View post / etc is included
	3.13	View category / View tag / etc is included
Wireframes	4.1	Wireframes are included
	4.2	Basic graphical elements are used (i.e. simple lines, few colors)
	4.3	Wireframes are presented for at least two main screens
	4.4	For each wireframe, reference zones are identified
	4.5	Headers and footers are included
	4.6	Navigation structures are included
	4.7	Page titles and headings are included

References

Online Resources

- A List Apart, <https://alistapart.com>
- World IA Days, <https://www.worldiday.org>
- Nielsen Normal Group, <https://www.nngroup.com>
- Boxes and Arrows, <https://boxesandarrows.com>
- Smashing Magazine, <https://www.smashingmagazine.com>

Bibliography and Further Reading

- Information Architecture: The Design and Integration of Information Spaces. 2nd Edition. Wei Ding, Xia Lin, and Michael Zarro. Morgan & Claypool, 2017
- Information Architecture: For the Web and Beyond (aka "*Polar Bear*" book) . 4th Edition. Louis Rosenfeld, Peter Morville, and Jorge Arango. O'Reilly Media, 2015
- Everyday Information Architecture. Lisa Maria Martin. A Book Apart, 2019

Database Specification

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Requirements Specification (ER) delivery
- Database Specification (EBD) development
 - Conceptual Data Modeling
 - Relational Schema
 - Schema Validation and Refinement
- PostgreSQL

LBAW Plan

- Plan: Moodle > Week #3 > Lecture #3
 - 3rd week of classes;
 - Lecture: start EBD discussion with conceptual data model (A4);
 - Lab classes: work on the ER component (A1 + A2 + A3); Delivery next week!
- Groups: <https://moodle.up.pt/mod/choicegroup/view.php?id=28565>
 - Many groups still not registered.
 - Critical for work submission.
- Monitor sessions: start in October

Requirement Specification (ER) Delivery

Requirements Specification (ER) Delivery

- First component delivery next week – 2nd (Sunday!) to 6th Oct.
- Deadline is on the day before the lab class, before 12h00 (*midday*). Submission steps:
- 1. **Fill the group spreadsheet checklist:**
 - update the “Group Self-Evaluation” tab;
 - fill the ER, A1, A2, and A3 tabs.
- 2. Verify that the component on the group’s **GitLab wiki is updated with the ER** component.
- 3. **Export the component wiki page to PDF and submit it on Moodle:**
 - Only one submission per group is necessary;
 - Ensure all images were correctly exported;
 - Only the information included in the PDF will be considered for evaluation.

Questions about ER component submission?

Database Specification (EBD) Development

Database Specification (EBD) Component

- The EBD component groups the artifacts to be made by the development team in order to support the storage and retrieval requirements identified in the requirements specification.
- It consists of three artifacts:
 - A4: Conceptual Data Model
 - A5: Relational Schema, Validation and Schema Refinement
 - A6: Indexes, Triggers, Transactions and Database Population
- LBAW Artifacts

A4. Conceptual Data Model

- In this artifact the data requirements of the system are detailed.
- The Conceptual **Domain** Model contains the identification and description of the entities of the domain and the relationships between them.
- The Conceptual Domain Model is simplified to include only concepts (entities and relationships) of the domain that are stored in the database.
- The Conceptual **Data** Model is obtained by using a UML class diagram containing the classes, associations, multiplicity and roles.
- For each class, the attributes, associations and constraints are included in the class diagram.
- Business rules not included in the class diagram are described by words or using OCL (Object Constraint Language) included as UML notes.

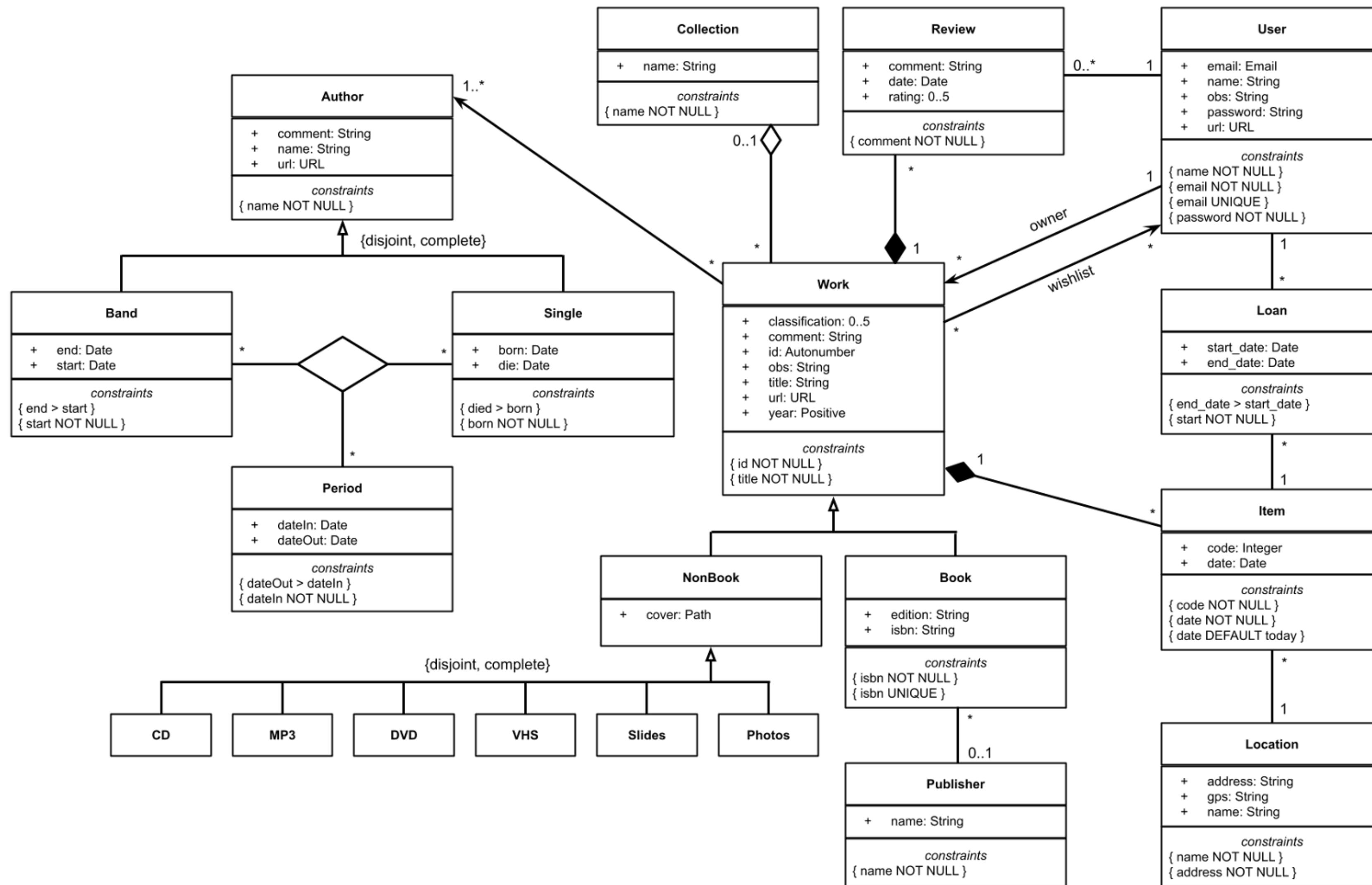
A4. Data Modeling

- To obtain a conceptual model, **iteratively** go through these steps:
 - 1. Identify entity types (a collection of people, places, things, events, or concepts);
 - 2. Identify relationships (entities have relationships with other entities);
 - 3. Identify attributes (each entity type will have one or more data attributes);
 - 4. Apply naming conventions (team standards and guidelines applicable to data modeling).
- *Scott Ambler. The Object Primer. Cambridge University Press, 3rd Edition, 2004. Section. 8.5*

A4. Data Modeling

- Data models are used for a variety of purposes, from conceptual models to physical design models
- Logical data models are used to **explore domain concepts and their relationships**
- With data modeling, you **identify data entities and assign data attributes** to them
 - whereas with class modeling you *identify classes and assign responsibilities to them*
- Then, you identify the **associations** between entities
 - relationships, inheritance, and composition
 - similar to the associations between classes
- Scott Ambler. *The Object Primer*. Cambridge University Press, 3rd Edition, 2004. Section. 8.5

A4. MediaLibrary Class Diagram



A4. MediaLibrary Business Rules

- BR01. A user cannot loan its own items.
- BR02. An item can only be lended by its owner.
- BR03. An item can only be loaned to one user at a time.

A4. Checklist

A4. Conceptual Data Model		
Artefact	1.1	The artefact reference and name are clear
	1.2	The goal of the artefact is briefly presented (1, 2 sentences)
UML	2.1	UML notation is consistently used
	2.2	Diagram layout is clear (visual organization)
	2.3	Classes are correctly represented
	2.4	Generalizations are correctly represented
	2.5	Associations are correctly represented
	2.6	Restrictions and business rules are correctly represented
Classes	3.1	Classes are presented with areas for name and attributes
	3.2	Classes do not have methods associated
	3.3	The classes support all high priority user stories defined in A2
	3.4	Class names are consistent (e.g. always singular, always in English)
Associations, multiplicity, roles	4.1	Automatic primary keys are not presented
	4.2	Natural keys are not used as primary keys (e.g. NIF)
	4.3	Multiplicity is defined for all associations
	4.4	Roles are used to explain how an object participates in the relationship
	4.5	Mandatory associations (not null) are indicated in the multiplicity
	4.6	In 1-1 associations, directionality is defined
	4.7	The associations support all high priority user stories defined in A2
	4.8	There is an "Authorship" association

Attributes, domains and restrictions	5.1	All attributes have a generic type (text, number, date, boolean)
	5.2	Attribute visibility is omitted (e.g. '+' prefix not included)
	5.3	Domains are defined for attributes that have predefined fixed values
	5.4	Not null attributes are indicated in the restrictions
	5.5	Unique attributes are indicated in the restrictions
	5.6	Restrictions related to numerical attributes are indicated (e.g. > 0)
	5.7	Restrictions related to date types are indicated (e.g. > today)
	5.8	Attributes with default values are indicated
	5.9	All generalizations have constraints defined
	5.10	All restrictions and business rules defined in A2 are included

A4. Discussion

- Example:
 - Social network, modeling friends and invitations.
- Modeling issues to consider:
 - Notifications.
 - Deal with user account deletion while keeping user data.

A5. Relational Schema, Validation and Schema Refinement

- The A5 artifact contains the Relational Schema obtained by mapping from the Conceptual Data Model.
- The Relational Schema includes each relation schema, attributes, domains, primary keys, foreign keys and other integrity rules: UNIQUE, DEFAULT, NOT NULL, CHECK.
- Relation schemas are specified in the compact notation.
- In addition to this representation, the relational schema is also presented in SQL as an annex.
- To validate the Relational Schema obtained from the Conceptual Model, all functional dependencies are identified and the normalization of all relation schemas is accomplished.
- Should it be necessary, in case the scheme is not in the Boyce–Codd Normal Form (BCNF), the relational schema is refined using normalization.

A5. Relational Schema Compact Notation

- Relation schemas are specified in the compact notation:
- $\text{table1}(\underline{\text{id}}, \text{attribute NN})$
- $\text{table2}(\underline{\text{id}}, \text{attribute} \rightarrow \text{Table1 NN})$
- $\text{table3}(\underline{\text{id1}}, \underline{\text{id2}} \rightarrow \text{Table2}, \text{attribute UK NN})$
- $\text{table4}(\underline{\text{id1}}, \underline{\text{id2}}) \rightarrow \text{Table3}, \text{id3, attribute})$
- Primary keys are underlined. UK means UNIQUE and NN means NOT NULL.
- The specification of additional domains can also be made in a compact form, using the notation:
- Today DATE DEFAULT CURRENT_DATE
- Priority ENUM ('High', 'Medium', 'Low')
- **In PostgreSQL use lower case and the "snake_case" convention.**

A5. Relational Schema Mapping

Summary of Mapping Rules from Logical UML Models to Relational Schemas

Translated from:

UML – Metodologias e Ferramentas CASE, Vol. 1, 2^a Edição, pp. 314-315

Alberto Silva e Carlos Videira, Centro Atlântico (2005)

Rule 1	Classes are mapped into relation schemas
Rule 2	Class attributes are mapped to attributes of relations.
Rule 3	Operations of classes are generally not mapped. They can nevertheless be mapped to <i>stored procedures</i> , stored and executed in the global context of the database involved.
Rule 4	Objects are mapped into tuples of one or more relations.
Rule 5	Each object is uniquely identified. If the identification of an object is defined explicitly by the <i>OID (object identifier)</i> stereotype, associated with one or more attributes, this attribute is mapped to primary key in the relation schema. Otherwise, we assume implicitly that the corresponding primary key is derived from a new attribute with the name of the relation and common suffix (e.g. "PK", "ID").
Rule 6:	The mapping of many-to-many associations involves the creation of a new relation schema, with attributes acting together as primary key, and individually as foreign key for each of the schemas derived from the classes involved.
Rule 7:	The mapping of one-to-many associations involves the introduction, in the relation schema corresponding to the class that has the constraint "many", of a foreign key attribute for the other schema.

A5. Relational Schema Mapping

Rule 8:	The mapping of one-to-one associations has in general two solutions. The first corresponds to the fusion of the attributes of the classes involved in one common schema. The second solution is to map each of the classes in the corresponding schema and choose one of the schemas as the most suitable for the introduction of a foreign key attribute for the other schema. This attribute should also be defined as unique within that schema.
Rule 9:	Association navigability in general has no impact on the mapping process. The exception lies in one-to-one associations, when they are complemented with navigation cues it helps in the selection of the schema that should include the foreign key attribute.
Rule 10:	Aggregation and composition associations have a minimal impact on the mapping process, which may correspond to the definition of constraints cascade ("CASCADE") in changing operations and/or removal of tuples.
Rule 11:	<p>The mapping of generalization associations in general presents three solutions.</p> <p>The first solution consists in crushing the hierarchy of classes in a single schema corresponding to the original superclass. This solution is appropriate when there is a significant distinction in the structure of sub-classes and/or when the semantics of their identification is not strong.</p> <p>The second solution is to consider only schemas corresponding to the sub-classes and duplicate the attributes of the super-class in these schemas; in particular it works if the super-class is defined as abstract.</p> <p>The third solution is to consider all the schemas corresponding to all classes of the hierarchy, resulting in a mesh of connected schemas and maintained at the expense of referential integrity rules. This solution has the advantage of avoiding duplication of information among different schemas, but suggests a dispersion of information by various schemas, and might involve a performance penalty in query operations or updating of data by requiring the execution of various join operations (i.e. "JOIN") and/or validation of referential integrity.</p>

A5. MediaLibrary Relational Schema

R01	user(<u>id</u> , email UK NN , name NN , obs, password NN , img, is_admin NN)
R02	author(<u>id</u> , name NN , img)
R03	collection(<u>id</u> , name NN)
R04	work(<u>id</u> , title NN , obs, img, year NN CK year > 0, id_user -> user NN , id_collection -> collection)
R05	author_work(<u>id_author</u> -> author, <u>id_work</u> -> work)
R06	nonbook(<u>id_work</u> -> work, type NN CK type IN Types)
R07	publisher(<u>id</u> , name NN)
R08	book(<u>id_work</u> -> work, edition, isbn UK NN , id_publisher -> publisher)
R09	location(<u>id</u> , name NN , address NN , gps)
R10	item(<u>id</u> , id_work -> work NN , id_location -> location NN , code NN , date NN DF Today)
R11	loan(<u>id</u> , id_item -> item NN , id_user -> user NN , start_date NN , end_date NN CK end > start)
R12	review(<u>id_user</u> -> user, <u>id_work</u> -> work, date NN DF Today , comment NN , rating NN CK rating > 0 AND rating < = 5)
R13	wish_list(<u>id_user</u> -> user, <u>id_work</u> -> work)

A5. Schema Validation and Refinement

- To validate the Relational Schema obtained from the Conceptual Model,
 - **all functional dependencies** are identified and
 - the normalization of all relations' schemas is accomplished.
- Should it be necessary, in case the scheme is not in the Boyce–Codd Normal Form (BCNF), the relational schema is refined using **normalization**.

A5. Problems of Redundancy

- **Redundancy** is at the root of several problems associated with relational schemas:
 - redundant storage;
 - insert / delete / update anomalies.
- Integrity constraints, in particular **functional dependencies**, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).

A5. Normal Forms

- If a relation is in a certain normal form (BCNF, 3NF, etc), it is known that certain kinds of problems are avoided / minimized.
- Boyce-Codd normal form (BCNF) is a slightly stronger version of the third normal form (3NF). It deals with certain types of anomalies not addressed by the 3NF.
- A relation R is in BCNF if and only if, for every functional dependency, at least one of the following conditions holds:
 - $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$),
 - X is a superkey for schema R.

A5. MediaLibrary Validation and Schema Refinement

Table R01 (user)	
Keys: {id}, {email}	
Functional Dependencies	
FD0101	{id} → {email, name, obs password, img, is_admin}
FD0102	{email} → {id, name, obs, password, img, is_admin}
Normal Form	BCNF

Table R02 (author)	
Keys: {id}	
Functional Dependencies	
FD0201	{id} → {name, img}
Normal Form	BCNF

Table R05 (author_work)	
Keys: {id_author, id_work}	
Functional Dependencies	<i>none</i>
Normal Form	BCNF

A5. MediaLibrary SQL Code

- The A5 artifact only includes types and tables creation statements in SQL.
- The SQL creation script is expanded in the A6 to include indexes, triggers, and transactions.
- Test the SQL creation script in the production server
- Include it as a file in the repository, with a reference in the wiki.

A5. Checklist

A5. Relational Schema	
Artefact	1.1 The artefact reference and name are clear 1.2 The goal of the artefact is briefly presented (1, 2 sentences)
Schema	2.1 The compact notation is correctly used 2.2 Each relation has a unique reference 2.3 Relation names are lowercase and in snake_case when needed 2.4 All UML classes are mapped 2.5 All class attributes are mapped 2.6 All associations are mapped 2.7 All relations have a PK 2.8 No natural keys are used as PK 2.9 All FK attributes reference a relation 2.10 In 1-1 associations, a FK is used considering the directionality 2.11 In 1-N associations, a FK is used 2.12 In N-N associations, a relation is defined with a composite PK of two FKs 2.13 Generalisations are correctly mapped and the choices well justified 2.14 Domains are defined and used if necessary
Integrity rules	3.1 All NN attribute restrictions are included 3.2 All UK attribute restrictions are included 3.3 All date attributes have adequate restrictions 3.4 All numeric attributes have adequate restrictions

Schema validation	4.1 Schema validation section is included 4.2 For each relation, all candidate keys are listed 4.3 For each relation, all FD are listed 4.4 Each relation's normal form is identified 4.5 The schema's normal form is identified and a justification is provided
SQL Code	5.1 The SQL script is included 5.2 The SQL script contains the creation statements 5.3 The SQL script cleans up the current database state 5.4 The SQL script is cleaned (e.g. excluded from export comments) 5.5 Code highlighting is used for readability 5.6 All domains are included in the SQL script 5.7 All relations are included in the SQL script 5.8 PK are defined as SERIAL 5.9 FK are not defined as SERIAL 5.10 The SQL script works without errors 5.11 SQL script is included in the group's repository 5.12 The production database (at db.fe.up.pt) has been set up with the SQL script

A5. Discussion

- Mapping generalizations:
- Examples:
 - Media and media types (books, movies, etc);
 - Students and mobility students;
 - Post, and different post types (question, answer, comment).
- Map the three approaches;
- Consider reading, writing, and updating scenarios.

A6. Indexes, Triggers, Transactions and Database Population

- This artifact contains the physical schema of the database,
 - the identification and characterization of the **indexes**,
 - the support of data integrity rules with **triggers**,
 - the definition of the database **user-defined functions**,
 - and the identification and characterization of the database **transactions**.
- This artifact also includes the complete database creation script, including all SQL code necessary to define all integrity constraints, indexes, triggers and transactions.
- Also, the database creation script and the database population script should be included as separate elements.

PostgreSQL

PostgreSQL

- PostgreSQL is the database management system used in LBAW.
- PostgreSQL is a free and open-source RDBMS that follows a client-server paradigm.
- A PostgreSQL production environment is available, which must be used in the application's production version, at db.fe.up.pt
- Each group has a user account Ibaw22G, and a database Ibaw22G.
- This service is managed by UPdigital and is only available using the VPN to FEUP.

Setup PostgreSQL Connection

- To configure a connection to the database, use the following settings:
 - Host: db.fe.up.pt
 - Port: 5432
 - Database: lbaw22G
 - Username: lbaw22G
 - Password: <group password, given in class>
- For development, groups can use a local PostgreSQL instance through Docker containers.
- Instructions can be found at: <https://git.fe.up.pt/lbaw/template-postgresql>

PostgreSQL Clients

- PostgreSQL clients exist for all major operating system environments.
- https://wiki.postgresql.org/wiki/PostgreSQL_Clients
- pgAdmin, used in LBAW, is one of the most popular clients - www.pgadmin.org
- Binary packages exist, but simply use Docker to quickly setup a local instance.
- A command line interface client is available with the psql command.
- Connect with: **psql -h db.fe.up.pt -d lbaw22G -U lbaw22G**

About the PostgreSQL Production Environment (**important!**)

- A PostgreSQL database contains one or more schemas, which in turn contains one or more tables.
- All databases contain a public schema, which is used as default.
- In PostgreSQL's command line interface, you can view the current active schema with: **show search_path;**
- To change the schema for the current session use: **SET search_path TO <schema>;**
- **In the PostgreSQL setup at FEUP (db.fe.up.pt), the public schema is shared between all accounts,**
 - Tables created in the public schema are visible to all users (although not accessible).
If you look at the tables in the public schema, you will find a long list of tables.
 - **It is important to not use the public schema and instead create a schema with the name of your group (lbaw22G).**
- To create this schema, use the following command: **CREATE SCHEMA <lbaw22G>;**
- To always use this schema as the default in your project, add the following line to the beginning of your SQL scripts.
 - **SET search_path TO <lbaw22G>;**

PostgreSQL Local Development with Docker

Docker

- Docker is a lightweight virtualization platform for infrastructure management.
 - *Manage the infrastructure like software / Program the infrastructure.*
 - Applications are run in containers, e.g. PostgreSQL, Apache, etc.
 - Docker Hub (hub.docker.com) is a public registry that stores Docker images.
 - Contains official images, like PostgreSQL, nginx, Apache, etc.
 - And also user images.
 - Docker images are used to build Docker containers (an instance of an image).
 - Gitlab Container Registry is used to store project images.
- Install Docker: <https://docs.docker.com/get-docker/>

Start PostgreSQL 11 Server

- Official PostgreSQL Docker images: https://hub.docker.com/_/postgres
- Start a Docker container with a PostgreSQL server, version 11.3
 - \$ docker run --name pgsql11 -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 -d postgres:11.3
- You can connect to the PostgreSQL server using the command line by starting a command line prompt (bash) on the container:
 - \$ docker exec -it pgsql11 bash
- Or directly a PostgreSQL prompt with:
 - \$ docker exec -it pgsql11 psql -U postgres

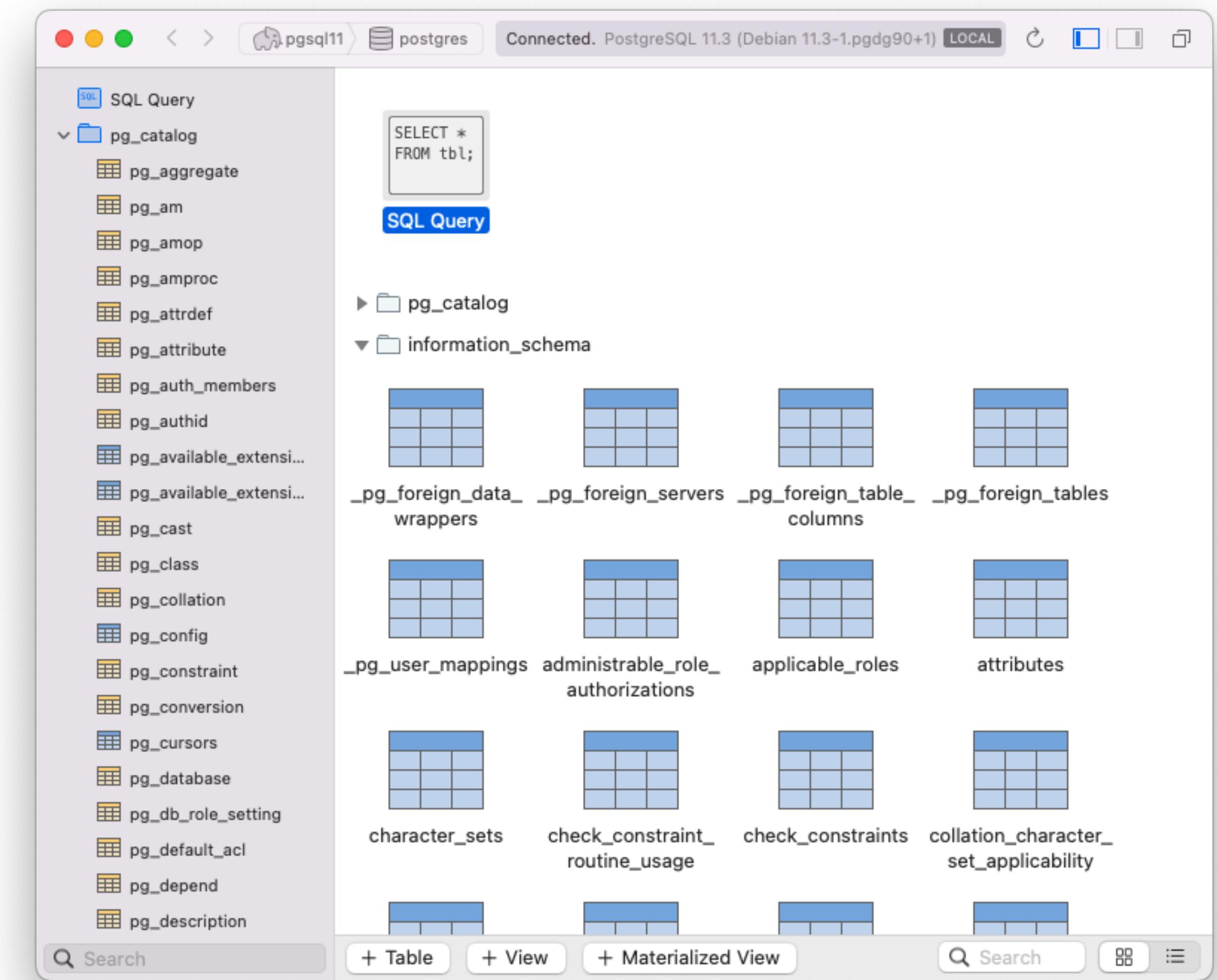
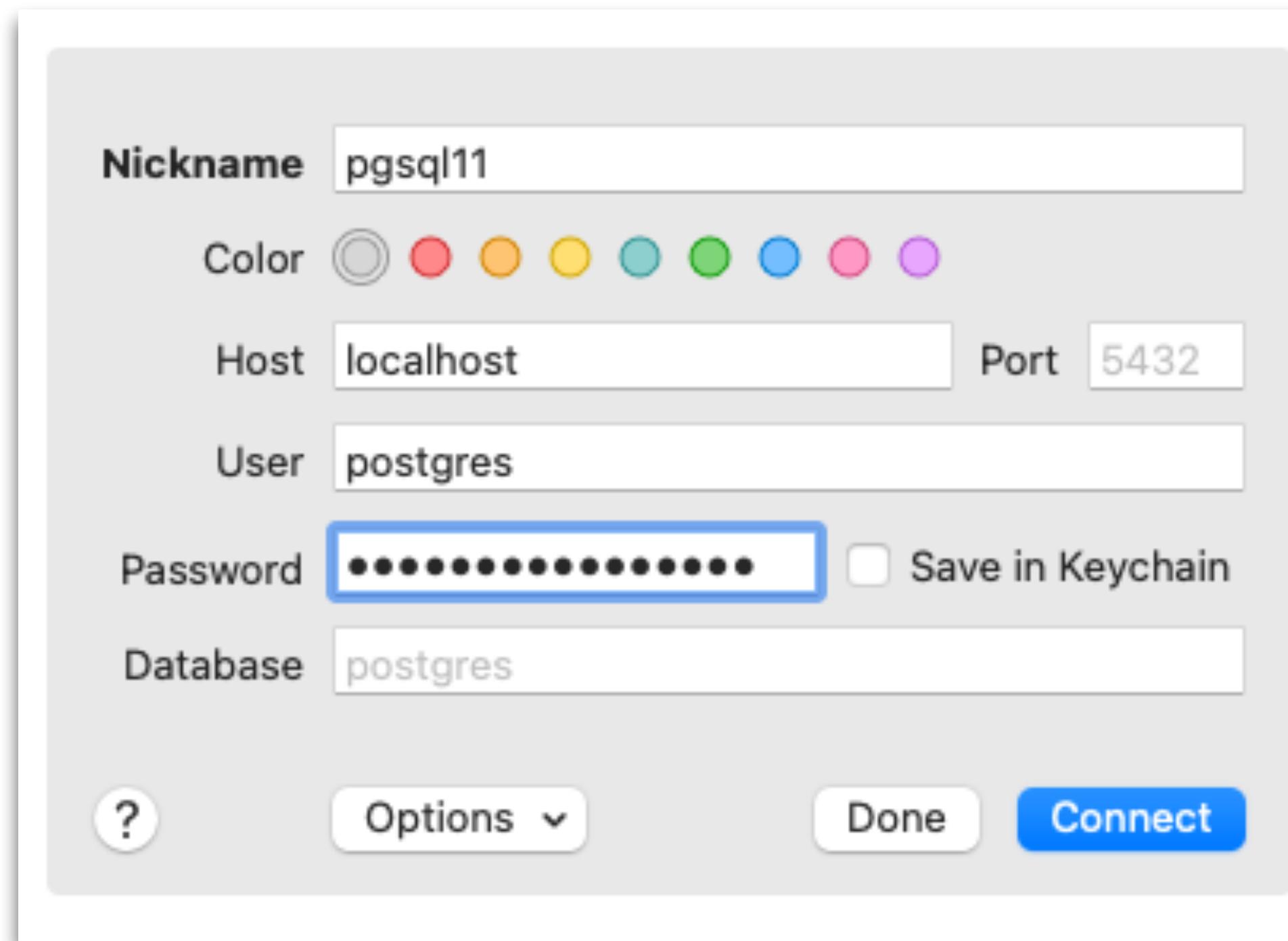
Docker Desktop Dashboard

The screenshot shows the Docker Desktop interface. On the left, there's a sidebar with icons for Containers, Images, Volumes, and Networks, along with a '+' button to create new resources. The main area is titled 'Containers' with a 'Give Feedback' link. It displays a single item: 'Showing 1 items'. A search bar is at the top right. The table lists the container details:

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
EXT	pgsql11 88fa8568bef5	postgres:11.3	Running	5432	3 minutes ago	

At the bottom, status metrics are shown: RAM 6.91GB, CPU 0.25%, and a note that it's 'Not connected to Hub'. The version is v4.12.0.

Native App (OSX Postico)



The screenshot shows the Postico application's main interface after connecting to the PostgreSQL database. The title bar indicates "Connected. PostgreSQL 11.3 (Debian 11.3-1.pgdg90+1) LOCAL". The left sidebar shows the tree structure of the pg_catalog schema, with "information_schema" expanded. The right pane displays the contents of the pg_catalog schema, including tables like pg_aggregate, pg_am, pg_amop, pg_amproc, pg_attrdef, pg_attribute, pg_auth_members, pg_authid, pg_available_extensions, pg_available_extensions_wrappers, pg_cast, pg_class, pg_collation, pg_config, pg_constraint, pg_conversion, pg_cursors, pg_database, pg_db_role_setting, pg_default_acl, pg_depend, and pg_description. A SQL query editor at the top right contains the command "SELECT * FROM tbl;".

pgAdmin4

- pgAdmin4 is a web-based open source administration tool for PostgreSQL.
- You can use another Docker container to run pgAdmin4.
 - Official Docker image: <https://hub.docker.com/r/dpage/pgadmin4>
- \$ docker run --name pgadmin -e PGADMIN_DEFAULT_EMAIL=none@example.com -e PGADMIN_DEFAULT_PASSWORD=123 -p 5050:80 -d dpage/pgadmin4
- Access at <http://localhost:5050/> (use credentials defined in the previous command)
- Create a connection to the container running PostgreSQL. To know the IP address of the psql11 container use:
 - \$ docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' psql11

pgAdmin4

The screenshot displays two windows of the pgAdmin4 application. On the left is a connection configuration dialog for a server named 'pgsql11'. The 'Connection' tab is selected, showing the host as '172.17.0.2' and port as '5432'. Other tabs include 'General', 'SSL', 'SSH Tunnel', and 'Advanced'. On the right is the main pgAdmin4 interface showing a dashboard with various metrics and a database browser.

Connection Configuration (pgsql11):

- General: Host name/address - 172.17.0.2, Port - 5432, Maintenance database - postgres, Username - postgres, Kerberos authentication? - Off, Role, Service.
- Connection: Host name/address - 172.17.0.2, Port - 5432, Maintenance database - postgres, Username - postgres, Kerberos authentication? - Off, Role, Service.
- SSL: SSL mode - Off, SSL certificate - None, SSL key - None.
- SSH Tunnel: SSH port - 22, Tunnels - None.
- Advanced: Advanced options - None.

pgAdmin4 Dashboard:

- Servers:** Shows 1 server named 'pgsql11' with 1 database named 'postgres'.
- Database sessions:** Shows 1 total session, 1 active session, and 0 idle sessions.
- Transactions per second:** Shows a chart with a single data series (Transactions) peaking at approximately 4.
- Tuples in:** Shows 1 insert, 0 updates, and 0 deletes.
- Tuples out:** Shows 80 fetched tuples and 0 returned tuples.
- Block I/O:** Shows 140 reads and 0 hits.
- Database activity:** Shows sessions, locks, and prepared transactions. One session is listed: PID 132, User postgr..., Application pgAdmin 4 - DB:post..., Client 172.17.0.3, Backend start 2022-09-30 08:32:39, Transaction start 2022-09-30 08:35:21, State active, Wait event act... .

References

Bibliography and Further Reading

- Scott Ambler, The Object Primer, Cambridge University Press, 3rd Edition, 2004.
- Alberto Rodrigues da Silva, Carlos Videira, UML – Metodologias e Ferramentas CASE, 2^a Edição, Centro Atlântico Editora, Maio 2005.
- Raghu Ramakrishnan, Johannes Gehrke. Database Management Systems. McGRAW-Hill International Editions, 3rd Edition, 2003.

Lab Class #2

- Work on the ER component.
- Develop and discuss the actors and user stories (A2).
- Develop and discuss the information architecture (A3).

SQL – Transactions

Carla Teixeira Lopes

Bases de Dados

Licenciatura em Engenharia Informática e Computação, FEUP+FCUP

Based on Jennifer Widom slides

Agenda

Introduction

Properties

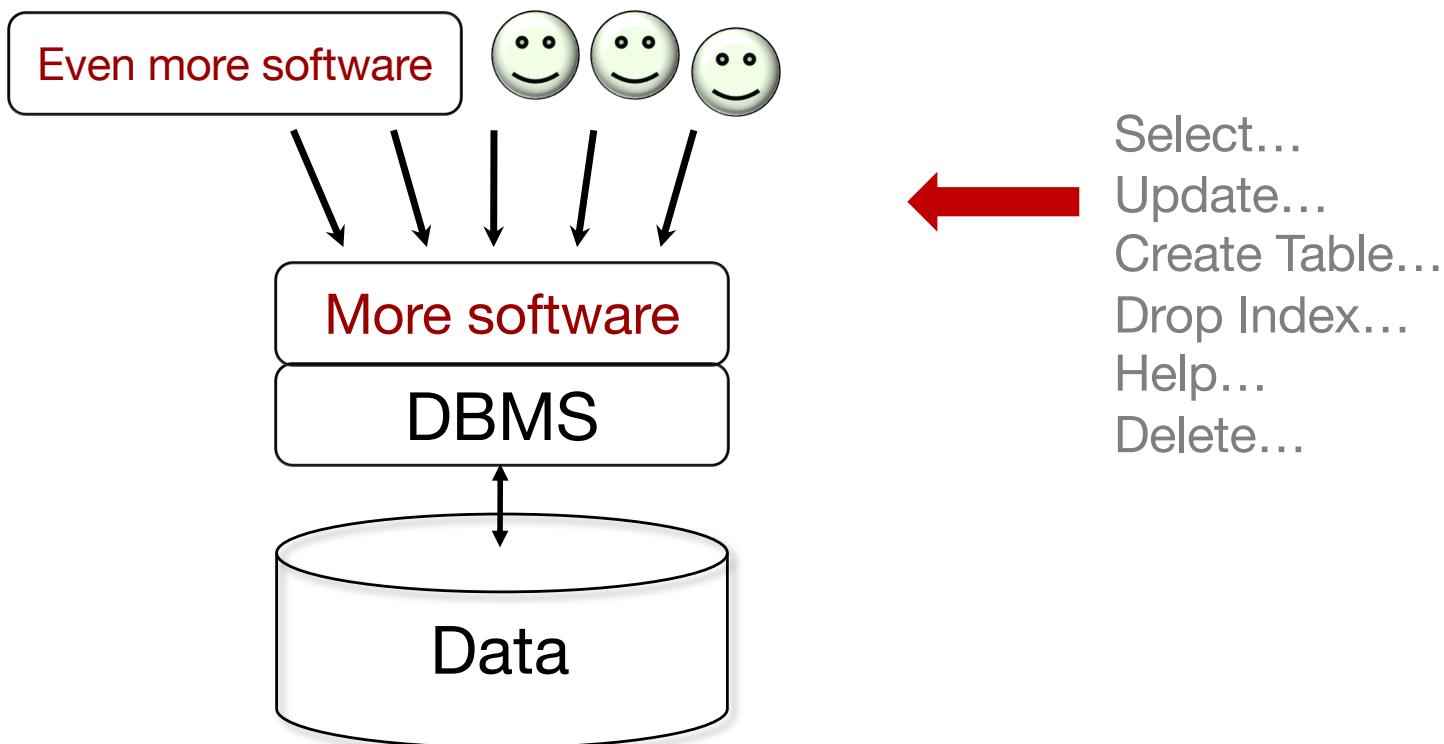
Isolation levels

Motivation for transactions

Concurrent database access

Resilience to system failures

Concurrent Database Access



Attribute-level Inconsistency

Update College Set enr = enr + 1000 **Where** cName = ‘Stanford’

concurrent with ...

Update College Set enr = enr + 1500 **Where** cName = ‘Stanford’

	15000	

get; modify; put

$$15\ 000 + 2\ 500 = 17\ 500$$

$$15\ 000 + 1\ 000 = 16\ 000$$

$$15\ 000 + 1\ 500 = 16\ 500$$

Tuple-level Inconsistency

Update Apply Set major = 'CS' Where sID = 123

concurrent with ...

Update Apply Set dec = 'Y' Where sID = 123



sID	major	dec
123		

get; modify; put

both changes

One of the two changes

Table-level Inconsistency

Update Apply Set decision = 'Y'

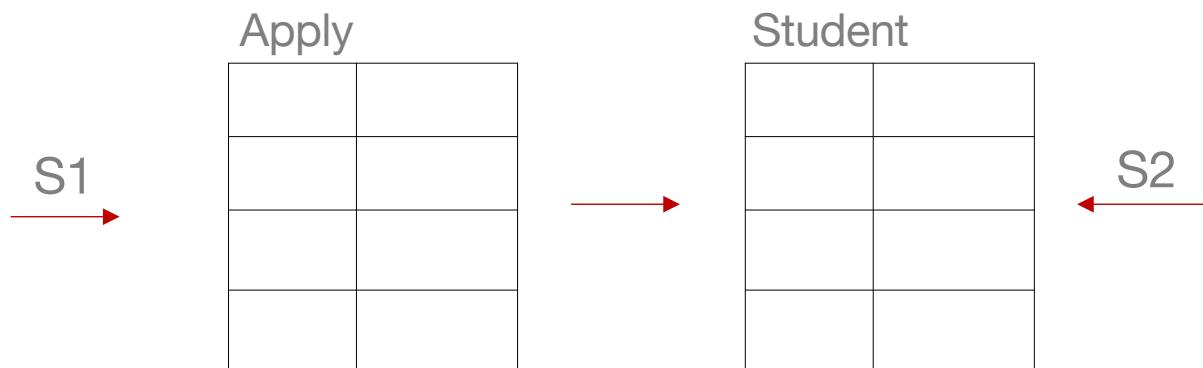
Where sID In (Select sID From Student Where GPA > 3.9)

} S1

concurrent with ...

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500

} S2



Multi-statement Inconsistency

Insert Into Archive Select * From Apply Where decision = 'N';

C1

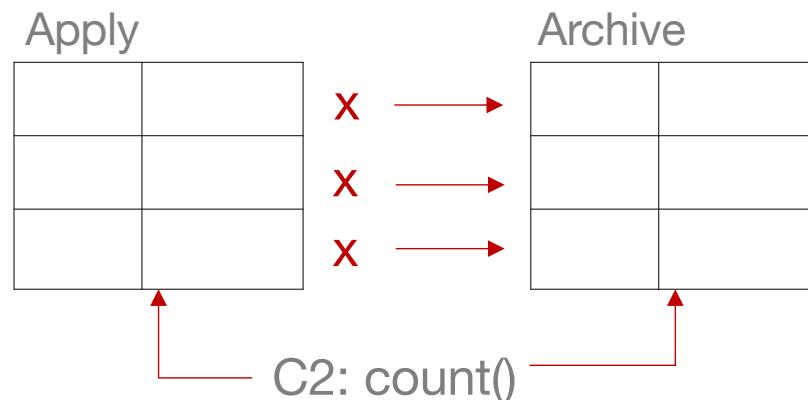
Delete From Apply Where decision = 'N';

concurrent with ...

Select Count(*) From Apply;

C2

Select Count(*) From Archive;



Concurrency goal

Execute sequence of SQL statements so they appear to be running in isolation

Simple solution: execute them in isolation

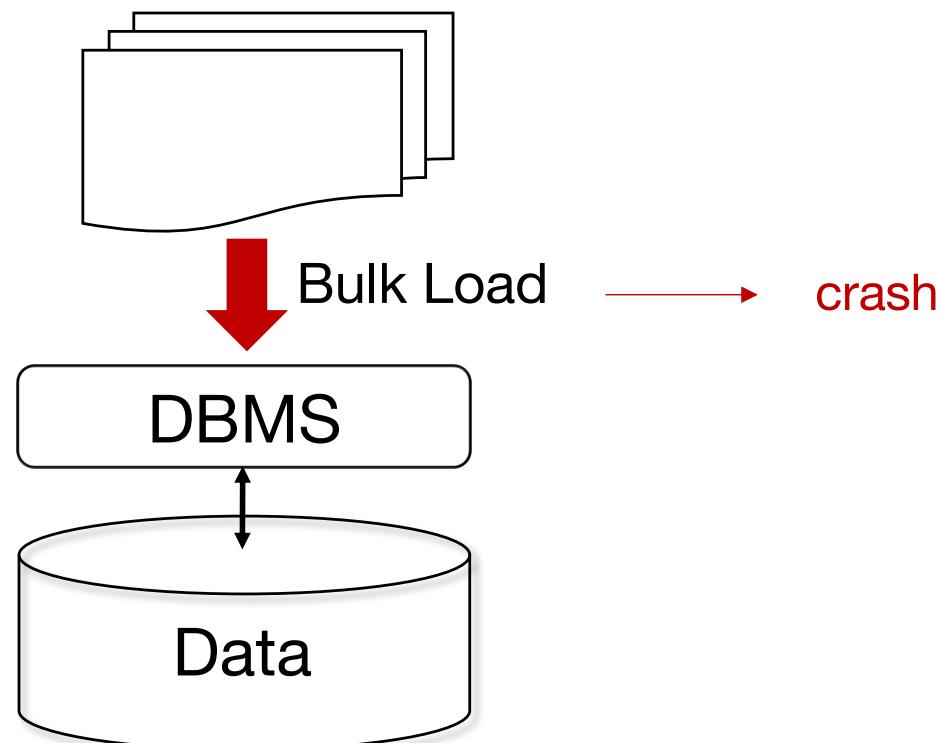
But want to enable concurrency whenever safe to do so

Multiprocessor system

Multithreaded system

Asynchronous I/O

Resilience to System Failures

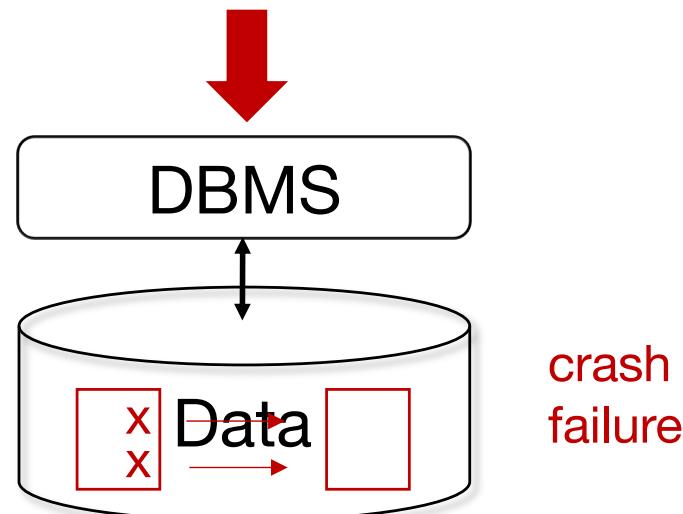


Resilience to System Failures

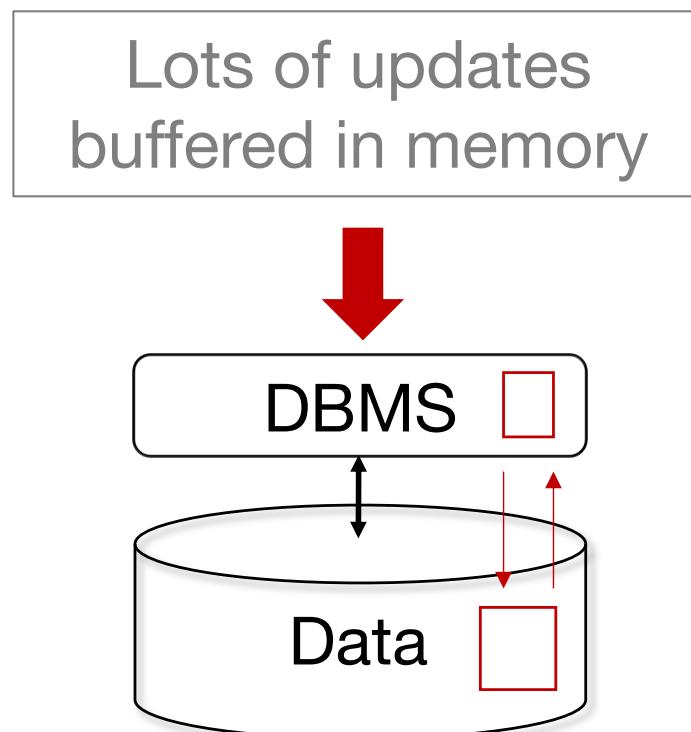
Insert Into Archive

Select * From Apply Where decision = 'N';

Delete From Apply Where decision = 'N';

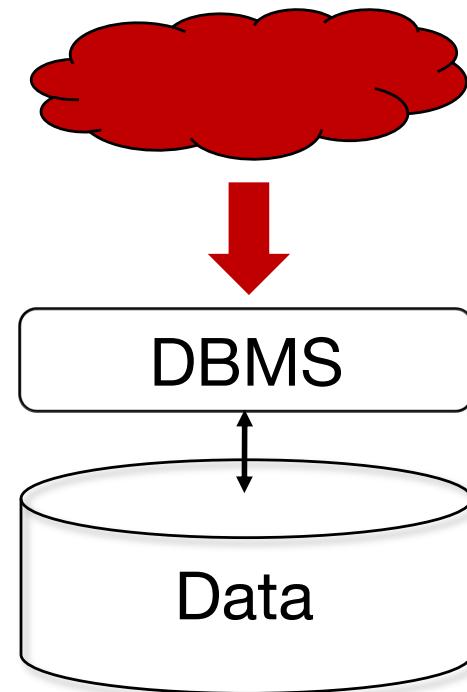


Resilience to System Failures



System-Failure Goal

Guarantee all-or-nothing execution, regardless of failures



Transactions

Solution for both concurrency and failures

A transaction is a sequence of one or more SQL operations treated as a unit

Transactions appear to run in isolation

If the system fails, each transaction's changes are reflected either entirely or not at all

Transactions: SQL standard

Transaction begins automatically on first SQL statement

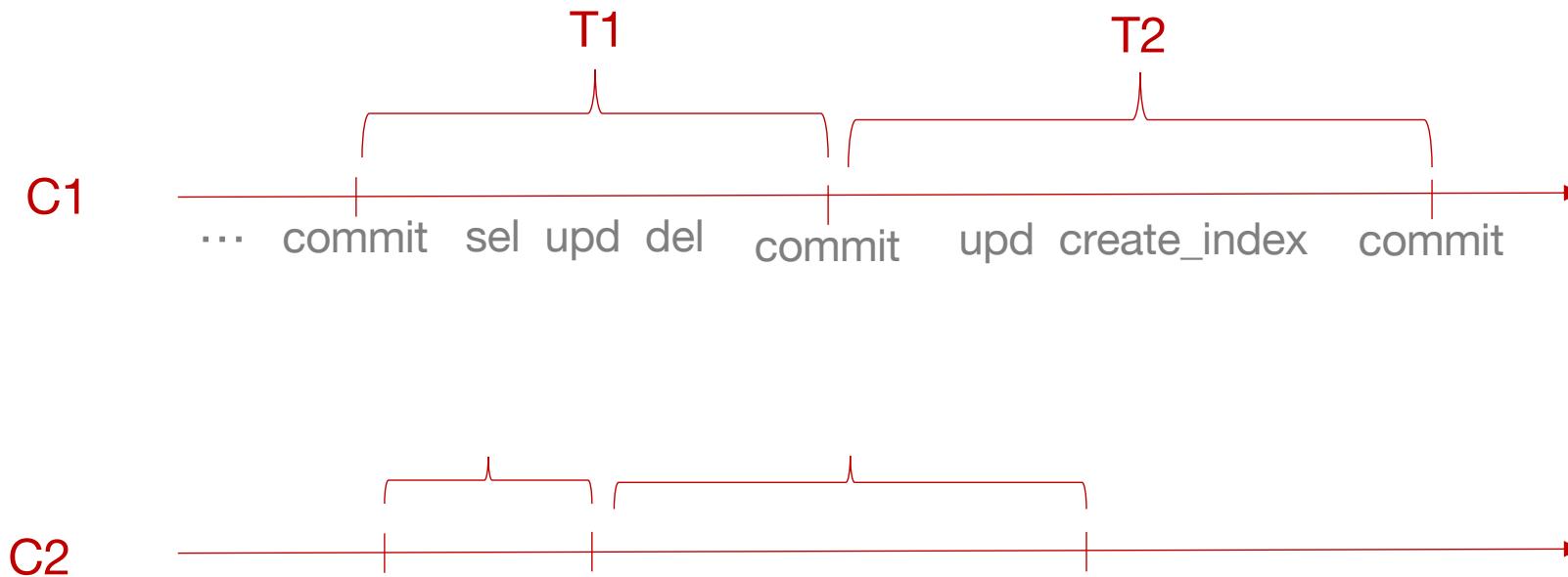
On “**commit**” transaction ends and new one begins

Current transaction ends on session termination

“**Autocommit**” turns each statement into transaction

Transactions

A transaction is a sequence of one or more SQL operations treated as a unit



Agenda

Introduction

Properties

Isolation levels

ACID Properties

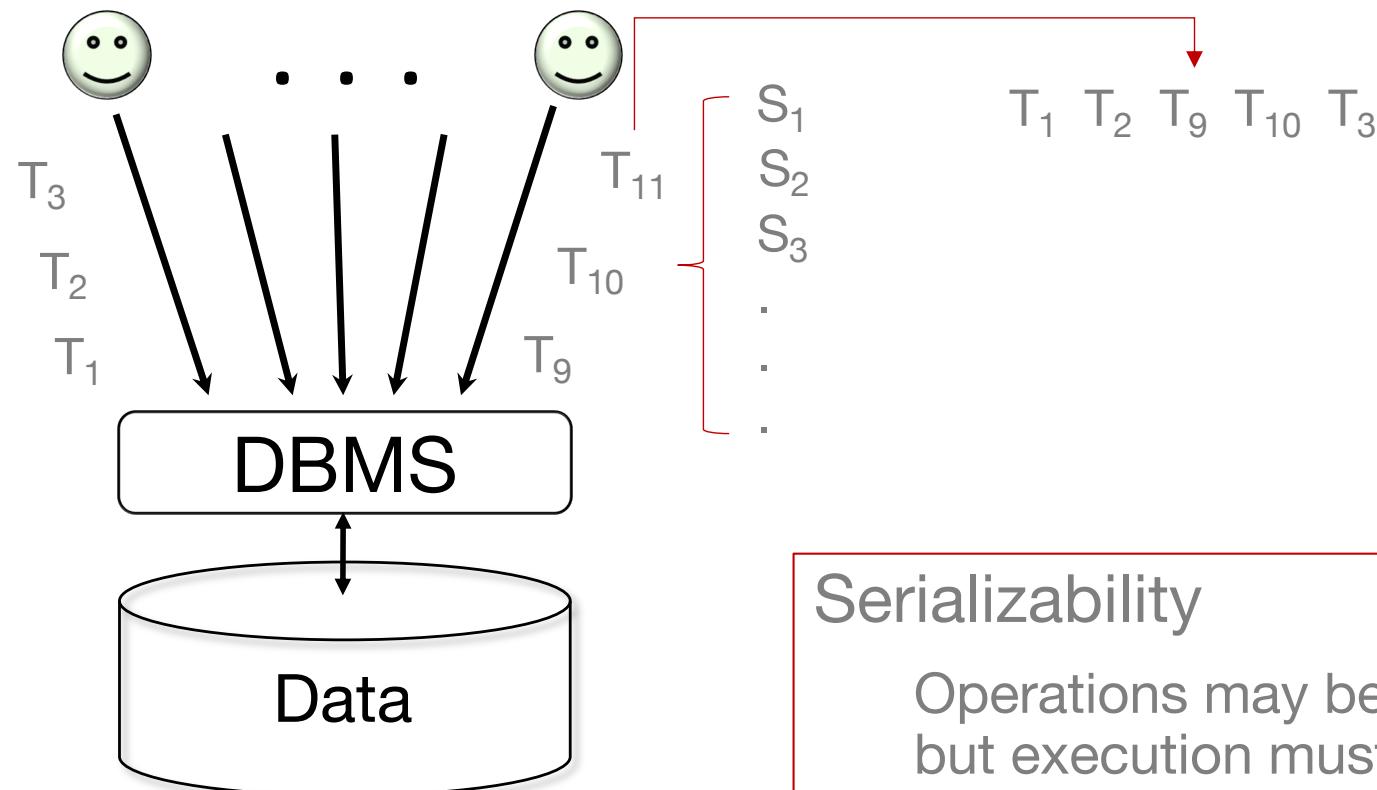
A_{ttom}icity 3

C_{onsistency} 4

I_{solation} 1

D_{urability} 2

ACID Properties: Isolation



Serializability

Operations may be interleaved,
but execution must be
equivalent to some sequential
(serial) order of all transactions

Attribute-level Inconsistency

Update College Set enr = enr + 1000 **Where** cName = ‘Stanford’ T_1

concurrent with ...

Update College Set enr = enr + 1500 **Where** cName = ‘Stanford’ T_2

If serializability is guaranteed

$T_1; T_2$ \longrightarrow 15 000 \rightarrow 17 500
 $T_2; T_1$

Tuple-level Inconsistency

Update Apply Set major = 'CS' Where sID = 123

T₁

concurrent with ...

Update Apply Set dec = 'Y' Where sID = 123

T₂

If serializability is guaranteed

T₁; T₂
T₂; T₁ → Both changes

Table-level Inconsistency

Update Apply Set decision = 'Y'

Where sID In (Select sID From Student Where GPA > 3.9)

} T₁

concurrent with ...

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500

} T₂

If serializability is guaranteed

T₁; T₂
T₂; T₁



Order
matters



DBMS don't guarantee the exact sequential order if the transactions are being issued at the same time

Multi-statement Inconsistency

Insert Into Archive Select * From Apply Where decision = 'N';

T₁

Delete From Apply Where decision = 'N';

concurrent with ...

Select Count(*) From Apply;

T₂

Select Count(*) From Archive;

If serializability is guaranteed

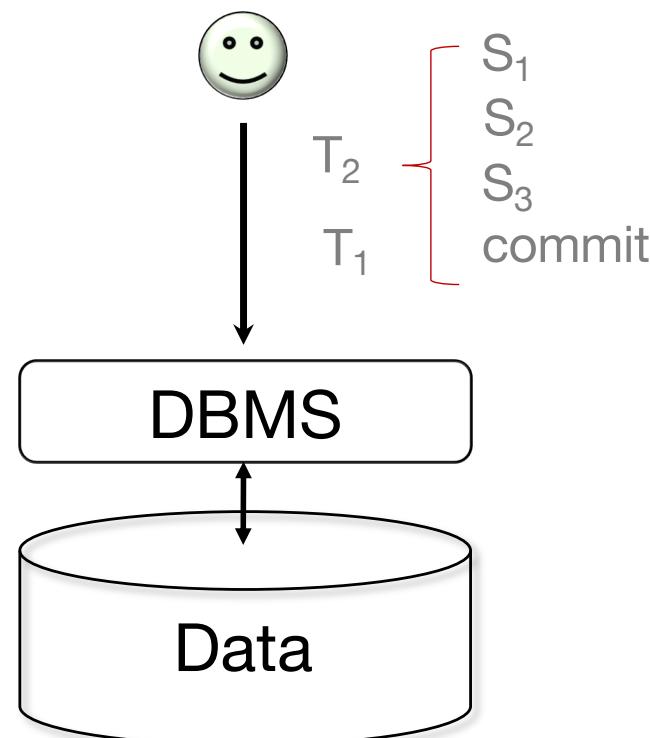
T₁; T₂
T₂; T₁



Order matters

ACID Properties: Durability

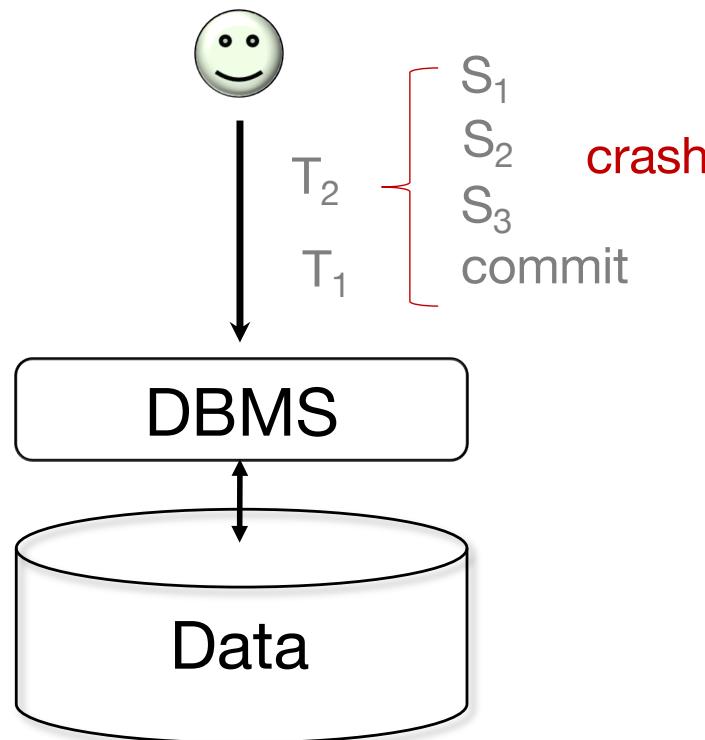
If system crashes after transaction commits, all effects of transaction remain in database



ACID Properties: Atomicity

Each transaction is “all-or-nothing”, never left half done

Using a logging mechanism, partial effects of transactions at the time of crash are undone



Transaction Rollback (= Abort)

Undoes partial effects of transaction

Can be system- or client-initiated

```
Begin Transaction;  
<get input from user>  
SQL commands based on input  
<confirm results with user>  
If ans='ok' Then Commit; Else Rollback;
```

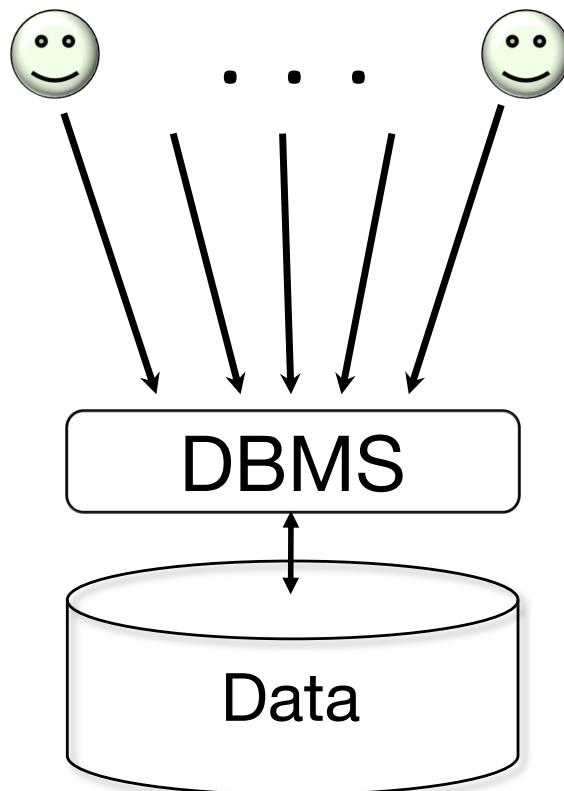
Transactions should
be constructed to run
quickly

→ Not wait arbitrary
amounts of time

Locking

Only undoes effects on the data itself

ACID Properties: Consistency



Each client, each transaction:

Can assume all constraints hold when transaction begins

Must guarantee all constraints hold when transaction ends

Serializability

Constraints always hold

T_1 T_2 T_3

Three red arrows point upwards from below, each labeled with a transaction identifier (T_1 , T_2 , T_3). This visualizes the execution of multiple transactions over time.

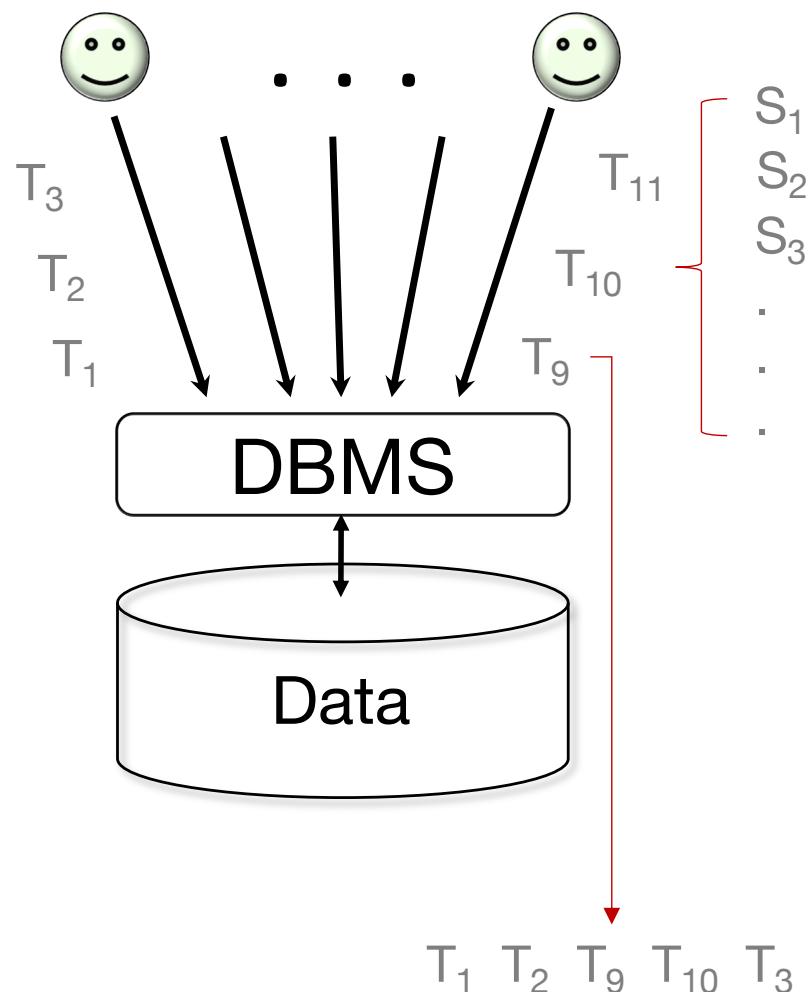
Agenda

Introduction

Properties

Isolation levels

ACID Properties: Isolation



Serializability

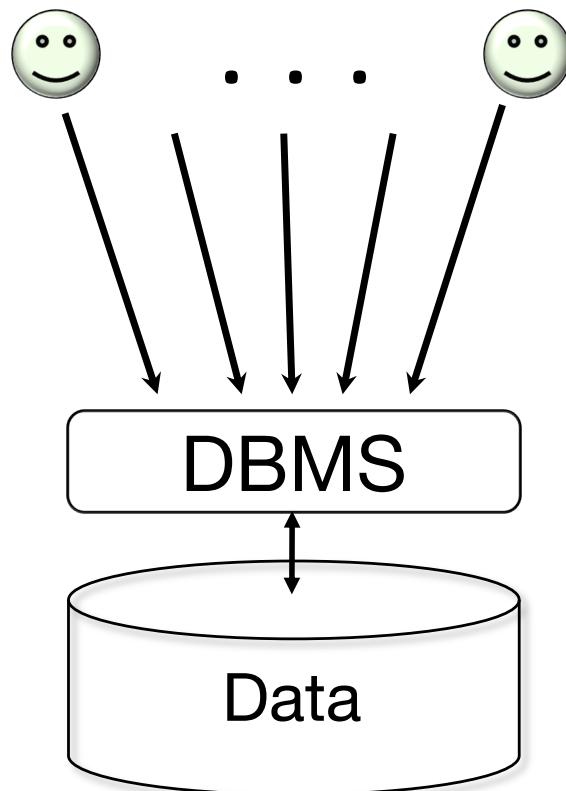
Operations may be interleaved, but execution must be equivalent to some sequential (serial) order of all transactions

Disadvantages

Overhead in locking

Reduction in concurrency

ACID Properties: Isolation



Weaker “Isolation Levels”

Read Uncommitted

Read Committed

Repeatable Read

Serializable

Weak
↓
Strong

↓Overhead in locking

↑Concurrency

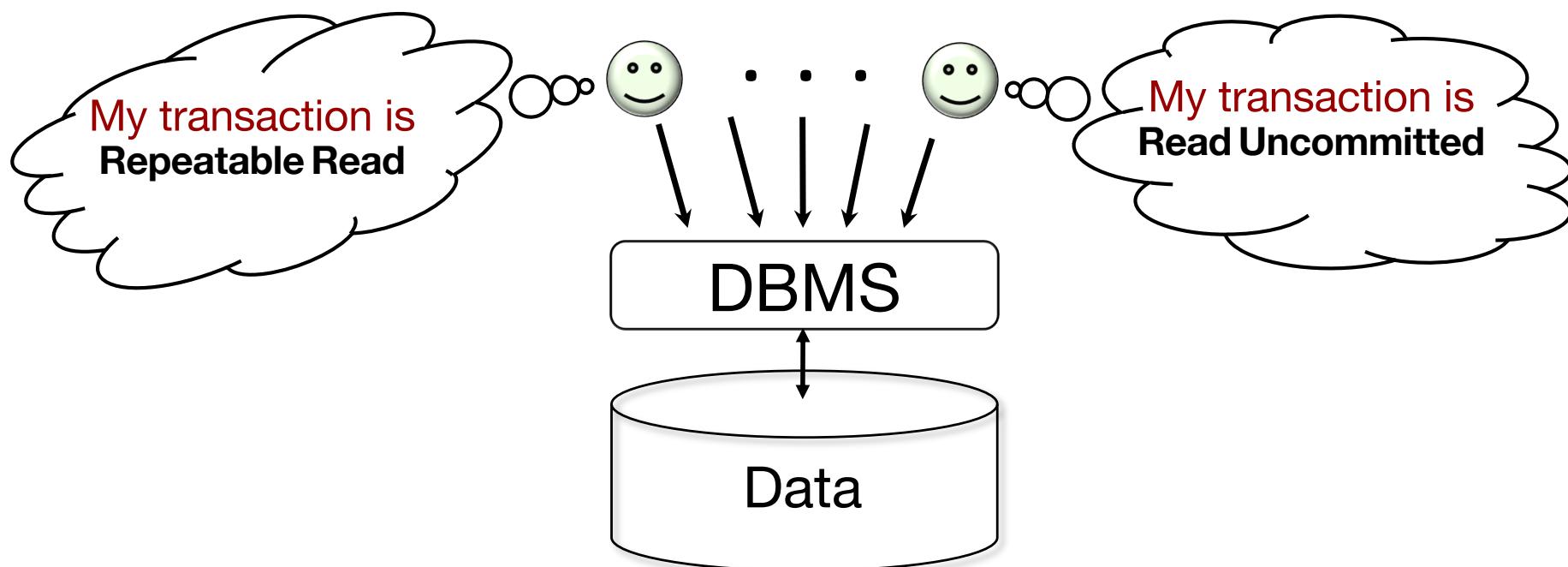
↓Consistency Guarantees

Isolation Levels

Per transaction

It does not affect the behaviour of any other transaction

Specific to Reads



Dirty Reads

“Dirty” data item: written by an uncommitted transaction

Update College Set enr = enr + 1000 Where cName = ‘Stanford’

T₁

concurrent with ...

Select avg(enr) From College

T₂



If read before T1 commits, this value is known as dirty

Assume there is a commit at the end of each box

Dirty Reads – Example 2

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500

T₁

concurrent with ...

Select GPA From Student Where sID=123

T₂

concurrent with ...

Update Student Set sizeHS=2600 Where sID=234

T₃

Where can we have dirty data items?

There are no
dirty reads
within the same
transaction

Read Uncommitted

A transaction may perform dirty reads

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500

T₁

concurrent with ...

Select avg(GPA) From Student

T₂

If transactions are serializable

T1; T2 or

T2; T1

Read Uncommitted

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500

T₁

concurrent with ...

Set Transaction Isolation Level Read Uncommitted;

Select avg(GPA) From Student;

T₂

We don't have serializable behaviour

We might don't care that much about consistency

Read Committed

A transaction may **not** perform dirty reads

Still does not guarantee global serializability

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500

T₁

concurrent with ...

Set Transaction Isolation Level Read Committed;

Select avg(GPA) From Student

Select max(GPA) From Student

T₂

Repeatable Read

A transaction may **not** perform dirty reads

An item read multiple times cannot change value

Still does not guarantee global serializability

Update Student Set GPA = (1.1) * GPA Where sizeHS > 2500;

Update Student Set sizeHS=1500 Where sID = 123;

T₁

concurrent with ...

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student

Select avg(sizeHS) From Student

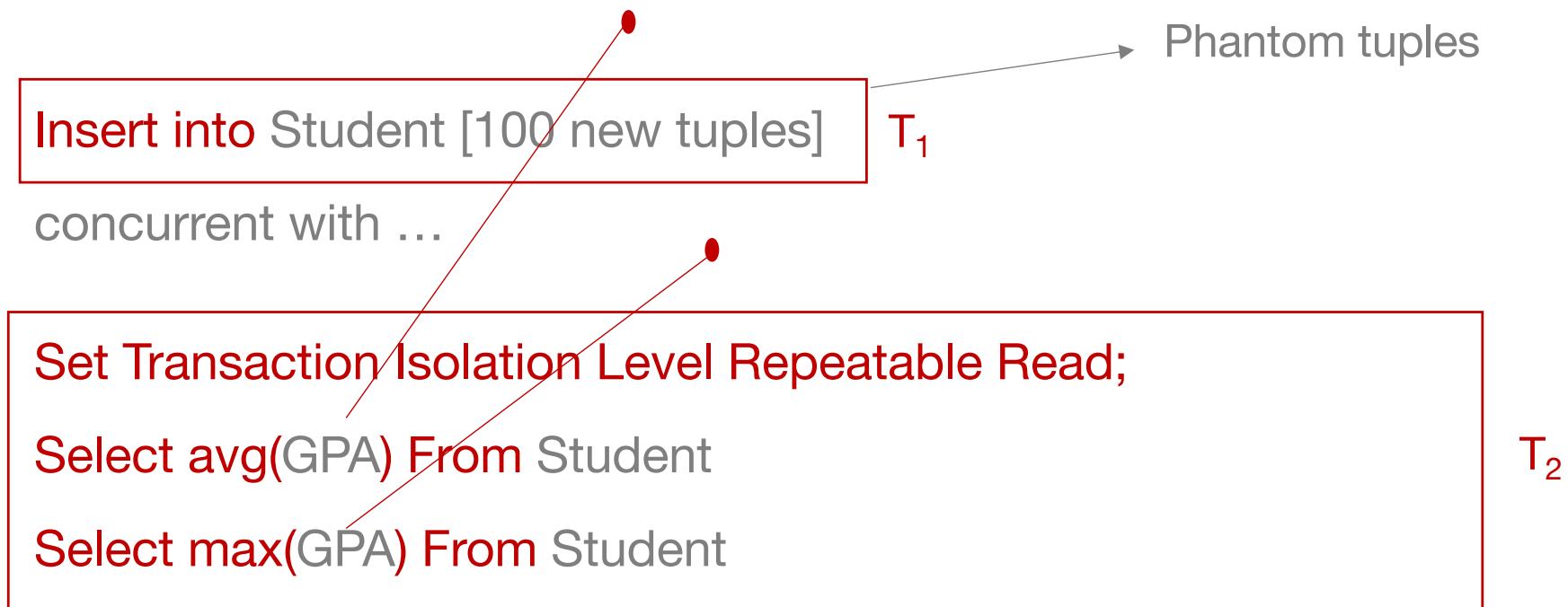
T₂

Repeatable Read

A transaction may **not** perform dirty reads

An item read multiple times cannot change value

But a relation *can* change: “phantom” tuples



Repeatable Read

A transaction may **not** perform dirty reads

An item read multiple times cannot change value

But a relation *can* change: “phantom” tuples

Delete from Student [100 new tuples]

T₁

concurrent with ...

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student

T₂

Select max(GPA) From Student

Once read, values get locked and deletion is not possible in the middle of T₂

Read Only Transactions

Helps system optimize performance

Independent of isolation level



Not going to perform modifications to the database within the transaction

Set Transaction Read Only;

Set Transaction Isolation Level Repeatable Read;

Select avg(GPA) From Student

Select max(GPA) From Student

Isolation Levels: Summary

	dirty reads	nonrepeatable reads	phantoms
weak			
Read Uncommitted	Y	Y	Y
Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N
strong			

Isolation Levels: Summary

Standard default: Serializable

Weaker isolation levels

Increased concurrency + decreased overhead = increased performance

Weaker consistency guarantees

Some systems have default Repeatable Read

Isolation level per transaction

Each transaction's reads must conform to its isolation level

Kahoot time!

Any doubts?

Readings

Jeffrey Ullman, Jennifer Widom, A first course in
Database Systems 3rd Edition

Section 6.6 – Transactions in SQL

Database Indexes, Triggers and Transactions

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Database Specification (EBD) development (A6)
 - Indexes
 - Triggers
 - Transactions
 - Database Population
- PostgreSQL setup

LBAW Plan

- Plan: *Moodle Lecture #4*
 - 4th week of classes;
 - Continue development on the second component (EBD);
 - Lab classes:
 - work on component (EBD);
 - work on the conceptual data model (A4); [**important**]
 - work on the relational schema (A5).
- Monitor sessions: Tuesdays, from 16h to 17h30
 - PostgreSQL setup and use.

Database Specification (EBD) Development

Database Specification (EBD) Component

review

- The EBD component groups the artifacts to be made by the development team in order to support the storage and retrieval requirements identified in the requirements specification.
- It consists of three artifacts:
 - A4: Conceptual Data Model
 - A5: Relational Schema, Validation and Schema Refinement
 - A6: Indexes, Triggers, Transactions and Database Population

A4. Conceptual Data Model

review

- In this artifact the data requirements of the system are detailed.
- The Conceptual **Domain** Model contains the identification and description of the entities of the domain and the relationships between them.
- The Conceptual Domain Model is simplified to include only concepts (entities and relationships) of the domain that are stored in the database.
- The Conceptual **Data** Model is obtained by using a UML class diagram containing the classes, associations, multiplicity and roles.
- For each class, the attributes, associations and constraints are included in the class diagram.
- Business rules not included in the class diagram are described by words or using OCL (Object Constraint Language) included as UML notes.

A5. Relational Schema, Validation and Schema Refinement

review

- The A5 artifact contains the Relational Schema obtained by mapping from the Conceptual Data Model.
- The Relational Schema includes each relation schema, attributes, domains, primary keys, foreign keys and other integrity rules: UNIQUE, DEFAULT, NOT NULL, CHECK.
- Relation schemas are specified in the compact notation.
- In addition to this representation, the relational schema is also presented in SQL as an annex.
- To validate the Relational Schema obtained from the Conceptual Model, all functional dependencies are identified and the normalization of all relation schemas is accomplished.
- Should it be necessary, in case the scheme is not in the Boyce–Codd Normal Form (BCNF), the relational schema is refined using normalization.

A5. Relational Schema Compact Notation

review

- Relation schemas are specified in the compact notation:
- `table1(id, attribute NN)`
`table2(id, attribute → Table1 NN)`
`table3(id1, id2 → Table2, attribute UK NN)`
`table4((id1, id2) → Table3, id3, attribute)`
- Primary keys are underlined. UK means UNIQUE and NN means NOT NULL.
- The specification of additional domains can also be made in a compact form, using the notation:
- `Today DATE DEFAULT CURRENT_DATE`
`Priority ENUM ('High', 'Medium', 'Low')`
- **In PostgreSQL use lower case and the "snake_case" convention.**

A5. Relational Schema Mapping

review

Summary of Mapping Rules from Logical UML Models to Relational Schemas

Translated from:

UML – Metodologias e Ferramentas CASE, Vol. 1, 2^a Edição, pp. 314-315

Alberto Silva e Carlos Videira, Centro Atlântico (2005)

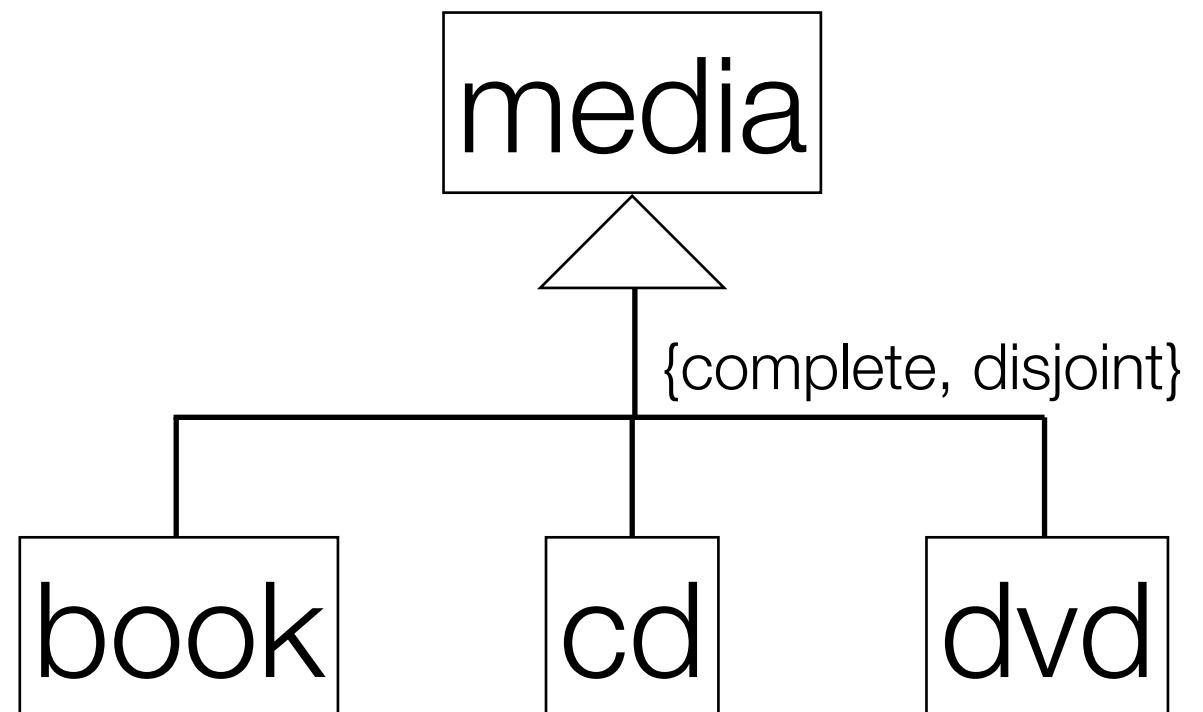
Rule 1	Classes are mapped into relation schemas
Rule 2	Class attributes are mapped to attributes of relations.
Rule 3	Operations of classes are generally not mapped. They can nevertheless be mapped to <i>stored procedures</i> , stored and executed in the global context of the database involved.
Rule 4	Objects are mapped into tuples of one or more relations.
Rule 5	<p>Each object is uniquely identified.</p> <p>If the identification of an object is defined explicitly by the <i>OID (object identifier)</i> stereotype, associated with one or more attributes, this attribute is mapped to primary key in the relation schema.</p> <p>Otherwise, we assume implicitly that the corresponding primary key is derived from a new attribute with the name of the relation and common suffix (e.g. "PK", "ID").</p>
Rule 6:	The mapping of many-to-many associations involves the creation of a new relation schema, with attributes acting together as primary key, and individually as foreign key for each of the schemas derived from the classes involved.
Rule 7:	The mapping of one-to-many associations involves the introduction, in the relation schema corresponding to the class that has the constraint "many", of a foreign key attribute for the other schema.

A5. Relational Schema Mapping

review

Rule 8:	The mapping of one-to-one associations has in general two solutions. The first corresponds to the fusion of the attributes of the classes involved in one common schema. The second solution is to map each of the classes in the corresponding schema and choose one of the schemas as the most suitable for the introduction of a foreign key attribute for the other schema. This attribute should also be defined as unique within that schema.
Rule 9:	Association navigability in general has no impact on the mapping process. The exception lies in one-to-one associations, when they are complemented with navigation cues it helps in the selection of the schema that should include the foreign key attribute.
Rule 10:	Aggregation and composition associations have a minimal impact on the mapping process, which may correspond to the definition of constraints cascade ("CASCADE") in changing operations and/or removal of tuples.
Rule 11:	<p>The mapping of generalization associations in general presents three solutions.</p> <p>The first solution consists in crushing the hierarchy of classes in a single schema corresponding to the original superclass. This solution is appropriate when there is a significant distinction in the structure of sub-classes and/or when the semantics of their identification is not strong.</p> <p>The second solution is to consider only schemas corresponding to the sub-classes and duplicate the attributes of the super-class in these schemas; in particular it works if the super-class is defined as abstract.</p> <p>The third solution is to consider all the schemas corresponding to all classes of the hierarchy, resulting in a mesh of connected schemas and maintained at the expense of referential integrity rules. This solution has the advantage of avoiding duplication of information among different schemas, but suggests a dispersion of information by various schemas, and might involve a performance penalty in query operations or updating of data by requiring the execution of various join operations (i.e. "JOIN") and/or validation of referential integrity.</p>

Mapping Generalizations



Superclass approach

```
media(id, type CHK {book, cd, dvd} ...)
```

ER approach

```
media(id, ...)
book(id->media, ...)
cd(id->media, ...)
dvd(id->media, ...)
```

Object Oriented

```
book(id, [media attributes], ...)
cd(id, [media attributes], ...)
dvd(id, [media attributes], ...)
```

A6. Indexes, Triggers, Transactions and Database Population

- This artifact contains the physical schema of the database,
 - the identification and characterization of the indexes,
 - the support of data integrity rules with triggers,
 - the definition of the database user-defined functions,
 - and the identification and characterization of the database transactions.
- This artifact also includes the complete database creation script, including all SQL code necessary to define all integrity constraints, indexes, triggers and transactions.
- Also, the database creation script and the database population script should be included as separate elements.

Indexes

A6. Indexes

- The **workload** is a study of the predicted system load, including an estimate on the number and growth of tuples in each relation.
- **Performance indexes** are applied to improve the performance of select queries.
 - At most, three performance indexes can be proposed, identifying the ones that are likely to have the biggest impact on the performance of the application.
- For each proposed index, it is necessary to indicate and justify the type chosen (B-tree, Hash, GiST, GIN), and also if clustering is recommended. As a last resource, controlled redundancy may be introduced (de-normalisation).
- The system being developed must provide full-text search features supported by PostgreSQL. Thus, it is necessary to **specify the fields where full-text search will be available** and the associated setup, namely all necessary configurations, indexes definitions and other relevant details.

Indexes in PostgreSQL (1)

- Indexes are secondary data structures used to improve data access (*useful metaphor - the alphabetical back-of-the-book index*).
- Finding and retrieving specific rows is much faster with indexes, but they add an overhead to the execution.
 - Without indexes, tables are usually sequentially scanned to find the matching entry.
 - With indexes, the number of steps to find the matching records can be drastically reduced.
- Two main types:
 - B-tree indexes: use a tree-like data structure that maintains data sorted and allow for search, order, range search in log time.
 - Hash indexes: use a hash-function to map keys to values; are only considered when an equality operator is used (no sorting or ranges).

PostgreSQL example

```
-- Create new schema.  
CREATE SCHEMA index_tests;  
  
-- Create sample table.  
CREATE TABLE sample (x NUMERIC);  
  
-- Insert into table 10,000 records. High cardinality values (sampled from 1...10,000,000).  
INSERT INTO sample  
SELECT random() * 10000  
FROM generate_series(1, 10000000);  
  
CREATE INDEX idx_numeric ON sample(x);  
CREATE INDEX idx_numeric ON sample USING BTREE(x);  
CREATE INDEX idx_numeric ON sample USING HASH(x);  
DROP INDEX idx_numeric;  
  
EXPLAIN ANALYZE  
SELECT * FROM sample  
WHERE X = 30;
```

PostgreSQL example (no index)

The screenshot shows the pgAdmin 4 interface connected to a PostgreSQL 14.5 database. The left sidebar shows the schema tree with 'SQL Query' selected. The main pane displays a SQL script and its execution plan.

```
-- Create new schema.  
CREATE SCHEMA index_tests;  
-- Create sample table.  
CREATE TABLE sample (x NUMERIC);  
-- Insert into table 10,000 records. High cardinality values (sampled from 1...10,000,000).  
INSERT INTO sample  
SELECT random() * 10000  
FROM generate_series(1, 10000000);  
  
CREATE INDEX idx_numeric ON sample(x);  
CREATE INDEX idx_numeric ON sample USING BTREE(x);  
CREATE INDEX idx_numeric ON sample USING BTREE(x);  
DROP INDEX idx_numeric;  
  
EXPLAIN ANALYZE  
SELECT * FROM sample  
WHERE X = 30;
```

QUERY PLAN

```
Gather (cost=1000.00..107138.43 rows=1 width=11) (actual time=760.326..767.183 rows=0 loops=1)  
  Workers Planned: 2  
  Workers Launched: 2  
    -> Parallel Seq Scan on sample (cost=0.00..106138.33 rows=1 width=11) (actual time=718.623..718.623 rows=0 loops=3)  
      Filter: (x = '30'::numeric)  
      Rows Removed by Filter: 3333333  
Planning Time: 0.094 ms  
JIT:  
  Functions: 6  
  Options: Inlining false, Optimization false, Expressions true, Deforming true  
  Timing: Generation 1.625 ms, Inlining 0.000 ms, Optimization 0.648 ms, Emission 14.527 ms, Total 16.800 ms  
Execution Time: 768.155 ms
```

Result 1 Export... 12 rows 770 ms

PostgreSQL example (hash)

The screenshot shows the pgAdmin 4 interface for PostgreSQL 14.5. The query editor window displays the following SQL code:

```
1 -- Create new schema.
2 CREATE SCHEMA index_tests;
3
4 -- Create sample table.
5 CREATE TABLE sample (x NUMERIC);
6
7 -- Insert into table 10,000 records. High cardinality values (sampled from 1...10,000,000).
8 INSERT INTO sample
9 SELECT random() * 10000
10 FROM generate_series(1, 10000000);
11
12
13 CREATE INDEX idx_numeric ON sample(x);
14 CREATE INDEX idx_numeric ON sample USING BTREE(x);
15 CREATE INDEX idx_numeric ON sample USING HASH(x);
16 DROP INDEX idx_numeric;
17
18 EXPLAIN ANALYZE
19 SELECT * FROM sample
20 WHERE X = 30;
```

The code includes comments explaining the steps: creating a schema, a sample table with 10,000 random numeric values, and three different index types (B-Tree, Hash, and BTREE) on the table. It concludes with an EXPLAIN ANALYZE command for a WHERE clause.

The pgAdmin interface also shows a tree view on the left with nodes for pg_catalog and information_schema. At the bottom, there are buttons for Search, Result 1, Export..., and a status bar indicating 4 rows and 3 ms execution time.

PostgreSQL example (btree)

The screenshot shows the pgAdmin 4 interface for PostgreSQL 14.5. The main window displays a SQL query editor with the following code:

```
1 -- Create new schema.
2 CREATE SCHEMA index_tests;
3
4 -- Create sample table.
5 CREATE TABLE sample (x NUMERIC);
6
7 -- Insert into table 10,000 records. High cardinality values (sampled from 1...10,000,000).
8 INSERT INTO sample
9    SELECT random() * 10000
10   FROM generate_series(1, 10000000);
11
12
13 CREATE INDEX idx_numeric ON sample(x);
14 CREATE INDEX idx_numeric ON sample USING BTREE(x);
15 CREATE INDEX idx_numeric ON sample USING HASH(x);
16 DROP INDEX idx_numeric;
17
18 EXPLAIN ANALYZE
19 SELECT * FROM sample
20 WHERE X = 30;
```

The line `CREATE INDEX idx_numeric ON sample USING BTREE(x);` is highlighted with a blue selection bar.

Below the code editor, the "QUERY PLAN" section shows the execution plan for the final query:

```
Index Only Scan using idx_numeric on sample (cost=0.43..4.45 rows=1 width=11) (actual time=0.021..0.021 rows=0 loops=1)
  Index Cond: (x = '30'::numeric)
  Heap Fetches: 0
Planning Time: 0.243 ms
Execution Time: 0.032 ms
```

At the bottom of the interface, there are search, result count, export, and performance metrics (5 rows, 2 ms).

Indexes in PostgreSQL (2)

- Indexes can be created for more than one attribute (multicolumn).
 - CREATE INDEX name ON table (a, b);
 - Work when searching for both attributes simultaneously or just a. Not just b.
- Indexes can also be created for expressions.
 - SELECT * FROM test1 WHERE lower(col1) = 'value';
 - CREATE INDEX test1_lower_col1_idx ON test1 (lower(col1));
- Index usage can be analyzed with the EXPLAIN command.

Unique Indexes

- Indexes can also be used to enforce uniqueness of a column's value, or the uniqueness of the combined values of more than one column.
- CREATE UNIQUE INDEX name ON table (column [, ...]);
- When an index is declared unique, multiple table rows with equal indexed values are not allowed. Null values are not considered equal.
- PostgreSQL automatically creates a unique index when
 - a unique constraint is used or
 - a primary key is defined for a table.

Clustering

- Clustering a table, results in the physical re-ordering of data in disk based on the index information. To cluster a table, an index must already be defined.
- Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered.
- If needed, clustering can be set to run periodically using cron.
- Clustering will help when multiple records are read together and an index can group them. It will be irrelevant when single rows are randomly accessed in a table.

Cardinality

- The uniqueness of data values contained in a particular column.
- The lower the cardinality, the more duplicate values in the column.
- Examples:
 - high cardinality - primary key
 - medium cardinality - last name in a customer table
 - low cardinality - boolean column
- Cardinality is used by the PostgreSQL planner, amongst other statistics, to estimate the number of rows returned by a WHERE clause. This is then used to decide if, and what, indexes should be used.

Full Text Search

- How can you search for a work in text fields? And multiple words?
- Using the LIKE operator is not feasible
 - There is no linguistic support (e.g. singular / plural).
 - No ranking is provided, only a set of results.
 - Multiple words search is not supported.
 - There is no index support.
- It is necessary to index each word individually.
- This is called *full text search*, or simply *text search*, on PostgreSQL.
- Key first step – define what is a document in your search system and what information is relevant.

tsvector Type

- Text is broken into lexemes, a normalized representation of words,
e.g. normalization includes converting to lowercase, identifying the stem, etc.
- The tsvector data type is used to store distinct lexemes.
 - SELECT to_tsvector('english', 'The quick brown fox jumps over the lazy dog')
 - 'brown':3 'dog':9 'fox':4 'jump':5 'lazi':8 'quick':2
- The function to_tsvector returns a **tsvector** with duplicates removed, stop words removed, and the number of position of each lexeme recorded.

tsqueries Type

- Queries, i.e. searches, are represented as **tsqueries**.
- The `to_tsquery` and `plainto_tsquery` functions convert a text query to a tsquery, a structure optimized for searching tsvectors.
- `SELECT plainto_tsquery('english','sail boats');`
 - `'sail' & 'boat'`
- `SELECT plainto_tsquery('portuguese','o velho barco');`
 - `'velh' & 'barc'`

Matching tsqueries to tsvectors

→ The @@ operator is used to **assert if a tsvector matches a tsquery**:

→ `SELECT to_tsvector('portuguese','o velho barco') @@ plainto_tsquery('portuguese','barca');`

→ t

→ `SELECT to_tsvector('portuguese','o velho barco') @@ plainto_tsquery('portuguese','carro');`

→ f

→ `SELECT title FROM posts
WHERE to_tsvector('english', title || ' ' || body) @@ plainto_tsquery('english', 'jumping dog');`

FTS Weights

- Sometimes we want to give more importance to some specific fields.
- We can use the **setweight** function to attach a weight to a certain tsvector.
- Weights go from 'A' (more important) to 'D' (less important).
- SELECT

```
setweight(to_tsvector('english', 'The quick brown fox jumps over the lazy dog'), 'A') ||  
setweight(to_tsvector('english', 'An English language pangram. A sentence that contains  
all of the letters of the alphabet.'), 'B')
```
- 'alphabet':24B 'brown':3A 'contain':17B 'dog':9A 'english':11B 'fox':4A 'jump':5A
'languag':12B 'lazi':8A 'letter':21B 'pangram':13B 'quick':2A 'sentenc':15B

Ranking FTS Results

- PostgreSQL provides two predefined ranking functions, which take into account lexical, proximity, and structural information:
 - how often the query terms appear in the document;
 - how close together the terms are in the document;
 - how important is the part of the document where they occur.
- Different applications might require additional information for ranking, e.g., document modification time. The built-in ranking functions are only examples.

Ranking FTS Results

- The ts_rank and ts_rank_cd functions, return a score for each returned row for a certain match between a tsquery and tsvector.
- ```
SELECT ts_rank(
 setweight(to_tsvector('english', 'The quick brown fox jumps over the lazy dog'), 'A') ||
 setweight(to_tsvector('english', 'An English language pangram. A sentence that contains all of the
letters of the alphabet.'), 'B'),
 plainto_tsquery('english', 'jumping dog')
)
```
- 0.9524299
- You can also change the weights of the tsvector classes (A to D) and set how normalization, due to different document lengths, should be performed.

# Pre-calculate FTS

---

- For performance reasons, we should consider adding a column to tables where FTS is to be performed containing the tsvector values of each row.
- This column should be updated whenever a row changes or is inserted. This can be done easily using a trigger.

```
CREATE FUNCTION post_search_update() RETURNS TRIGGER AS $$
BEGIN
 IF TG_OP = 'INSERT' THEN
 NEW.search = to_tsvector('english', NEW.title);
 END IF;
 IF TG_OP = 'UPDATE' THEN
 IF NEW.title <> OLD.title THEN
 NEW.search = to_tsvector('english', NEW.title);
 END IF;
 END IF;
 RETURN NEW;
END
$$ LANGUAGE 'plpgsql';
```

# Indexing FTS

---

- To select all posts containing both ‘jumping’ and ‘dog’ we can use the following query
  - ```
SELECT title FROM posts
WHERE search @@ plainto_tsquery('english', 'jumping dog')
ORDER BY ts_rank(search, plainto_tsquery('english', 'jumping dog')) DESC
```
- Note that ‘search’ is a pre-calculated column containing the tsvector of the columns we want to search.
- To improve the performance of our full text searches, we can use GIN or GiST indexes:
 - `CREATE INDEX search_idx ON posts USING GIN (search);`
 - `CREATE INDEX search_idx ON posts USING GIST (search);`
- “*As a rule of thumb, GIN indexes are best for static data because lookups are faster. For dynamic data, GiST indexes are faster to update.*”, PostgreSQL documentation.

PostgreSQL example (full text search)

```
- Movies loaded from $ curl https://datasets.imdbws.com/title.akas.tsv.gz -o movies.tsv.gz

-- Create a new schema.
CREATE SCHEMA movies;

-- Create a new table with a single column 'title'.
CREATE TABLE movies (title TEXT);

-- Import data from the movies file. Consider only the 3rd column, and ignore the first line (i.e. headers). ~>30 Millions
\copy movies FROM PROGRAM 'zcat movies.tsv.gz | cut -f3 | tail -n +2' WITH (FORMAT TEXT);

-- Add a new column for the tsvectors.
ALTER TABLE movies ADD COLUMN tsvector TSVECTOR;

-- Create and store ts_vectors on the column named 'tsvectors'.
UPDATE movies SET tsvector = to_tsvector('english', title);

-- ...takes some time.

-- Create an index on the ts_vectors.
CREATE INDEX idx_titles ON movies USING GIN(tsvector);

-- Query fast!
SELECT title, ts_rank(tsvector, query) AS rank
FROM movies, plainto_tsquery('english','documentary nature') query
WHERE tsvector @@ query
ORDER BY rank DESC;
```

PostgreSQL example (trigrams)

```
-- Load Trigram extension for fast trigram search.
CREATE EXTENSION pg_trgm;

CREATE INDEX idx_titles_trgm ON movies USING GIN(title gin_trgm_ops); -- takes some time.
DROP INDEX idx_titles_trgm;

-- EXPLAIN ANALYZE
SELECT title FROM movies WHERE title ILIKE 'spa%' LIMIT 10;
```

A6. MediaLibrary Indexes (Performance)

1. Database workload

Understanding the nature of the workload for the application and the performance goals, is essential to develop a good database design. The workload includes an estimate of the number of tuples for each relation and also the estimated growth.

Relation	Relation name	Order of magnitude	Estimated growth
R01	user	10 k (tens of thousands)	10 (tens) / day
R02	author	1 k (thousands)	1 (units) / day
R03	collection	100 (hundreds)	1 / day
R04	work	1 k	1 / day
R05	author_work	1 k	1 / day
R06	nonbook	100	1 / day
R07	publisher	100	1 / day
R08	book	1 k	1 / day
R09	location	100	1 / day
R10	item	10 k	10 / day
R11	loan	100 k	100 (hundreds) / day
R12	review	10 k	10 / day
R13	wish_list	10 k	10 / day

2. Proposed Indexes

Indexes are used to enhance database performance by allowing the database server to find and retrieve specific rows much faster. An index defined on a column that is part of a join condition can also significantly speed up queries with joins. Moreover, indexes can also benefit UPDATE and DELETE commands with search conditions.

After an index is created, the system has to keep it synchronised with the table, which adds overhead to data manipulation operations. As indexes add overhead to the database system as a whole, they are used sensibly.

2.1. Performance indices

Performance indexes are applied to improve the performance of select queries. At most, three performance indexes can be proposed, identifying the ones that have the biggest impact on the performance of the application.

Indexes should be proposed considering queries that are frequently used and involve large relations. Additionally, this section includes an analysis of the execution plan for two central, non-trivial, and frequently used SQL queries significantly impacted by the proposed performance indexes.

Index	IDX01
Index relation	work
Index attribute	id_users
Index type	B-tree
Cardinality	Medium
Clustering	Yes
Justification	Table 'work' is very large. Several queries need to frequently filter access to the works by its owner (user). Filtering is done by exact match, thus an hash type index would be best suited. However, since we also want to apply clustering based on this index, and clustering is not possible on hash type indexes, we opted for a b-tree index. Update frequency is low and cardinality is medium so it's a good candidate for clustering.
SQL Code	<pre>CREATE INDEX user_work ON work USING btree (id_users); CLUSTER work USING user_work;</pre>

A6. MediaLibrary Indexes (Full Text Search)

2.2. Full-text Search indexes

Full-text search indexes are applied to provide keyword based search over records of the database. Results using FTS are ranked by relevance and can use signals from multiple tables and with different weights. The first step in the process of defining FTS indices is to define what is a 'document' for the search features to support.

Index	IDX11
Index relation	work
Index attributes	title, obs
Index type	GIN
Clustering	No
Justification	To provide full-text search features to look for works based on matching titles or observations. The index type is GIN because the indexed fields are not expected to change often.
SQL Code	<pre>-- Add column to work to store computed ts_vectors.</pre>

SQL Code
<pre>-- Add column to work to store computed ts_vectors. ALTER TABLE work ADD COLUMN tsvector TSVECTOR; -- Create a function to automatically update ts_vectors. CREATE FUNCTION work_search_update() RETURNS TRIGGER AS \$\$ BEGIN IF TG_OP = 'INSERT' THEN NEW.tsvector = (setweight(to_tsvector('english', NEW.title), 'A') setweight(to_tsvector('english', NEW.obs), 'B')); END IF; IF TG_OP = 'UPDATE' THEN IF (NEW.title <> OLD.title OR NEW.obs <> OLD.obs) THEN NEW.tsvector = (setweight(to_tsvector('english', NEW.title), 'A') setweight(to_tsvector('english', NEW.obs), 'B')); END IF; END IF; RETURN NEW; END \$\$ LANGUAGE plpgsql; -- Create a trigger before insert or update on work. CREATE TRIGGER work_search_update BEFORE INSERT OR UPDATE ON work FOR EACH ROW EXECUTE PROCEDURE work_search_update(); -- Finally, create a GIN index for ts_vectors. CREATE INDEX search_idx ON work USING GIN (tsvector);</pre>

A6. Indexes Checklist

A6. Indexes, Integrity and Populated Database		
Artefact	1.1	The artefact reference and name are clear
	1.2	The goal of the artefact is briefly presented (1, 2 sentences)
Workload	2.1	The workload section is included
	2.2	The relations' magnitude and growth estimation section is included
	2.3	For each relation, magnitude and growth is estimated
Indexes	3.1	Performance indexes are proposed
	3.2	For each index, a relation and attribute(s) is defined
	3.3	For each index, the type is defined
	3.4	For each index, the cardinality is defined
	3.5	For each index, clustering is defined
	3.6	The impact of the indices is analysed for two illustrative queries
	3.7	Full-text search (FTS) indexes over multiple fields are proposed
	3.8	For FTS indexes, field weighting is used
	3.9	For each index, a justification is provided
	3.10	For each index, the SQL code is included
	3.11	Indexes are not proposed for PK
	3.12	Indexed are not proposed for UK

Triggers and User Defined Functions

A6. Triggers and User Defined Functions

- To enforce integrity rules that cannot be achieved in a simpler way, the necessary triggers are identified and described by presenting the event, the condition, and the activation code.
- User-defined functions, and trigger procedures, that add control structures to the SQL language, or perform complex computations, are identified and described to be trusted by the database server.
- Every kind of function (SQL functions, Stored procedures, Trigger procedures) can take base types, composite types, or combinations of these as arguments (parameters). In addition, every kind of function can return a base type or a composite type. Functions can also be defined to return sets of base or composite values.
- Common examples:
 - User cannot post in groups when he is not a member;
 - When a vote is cast, the rating (karma) of the author is updated;
 - When *an event happens*, relevant notifications are sent.

User-Defined Functions

- A user-defined function provides a mechanism for extending the functionality of the database server by adding a function.

- Advantages of using *stored procedures*:
 - Reduce the number of round trips between application and database server
 - Increase the application performance, because user-defined functions are pre-compiled and stored in the database server uses its full-power
 - Be able to be reused in many applications

- Disadvantages of *stored procedures*:
 - Slow software development because it requires specialized skills that many developers do not possess (PL/SQL)
 - Make it difficult to manage versions and hard to debug
 - Less portable code to other database management systems (MySQL, SQL Server, PostgreSQL, Oracle, DB2)

UDF Example

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype AS $name$
DECLARE
declaration;
[...]
BEGIN
< function_body >
[...]
RETURN { variable_name | VALUE }
END;
$name$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION totalRecords ()
RETURNS INTEGER AS $total$
DECLARE
    total INTEGER;
BEGIN
    SELECT COUNT(*) INTO total FROM company;
    RETURN total;
END;
$total$ LANGUAGE plpgsql;

SELECT totalRecords();
```

Triggers

- Triggers are *event-condition-action* rules:
 - Event, a change to the database that activates the trigger
 - Condition, a query or test that is run when the trigger is activated
 - Action, a procedure that is executed when the trigger is activated and its condition is true
- The action can be executed *before, after or instead of the trigger* event
- The action may refer the *new values and old values of records inserted, updated or deleted* in the trigger event
- The programmer specifies that the action is performed:
 - once for each modified record (FOR EACH ROW)
 - once for all records that are changed on a database operation

Triggers Example

```
CREATE FUNCTION loan_item() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF EXISTS (SELECT * FROM loan WHERE NEW.id_item = id_item AND end_t > NEW.start_t) THEN
        RAISE EXCEPTION 'An item can only be loaned to one user in every moment.';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER loan_item
    BEFORE INSERT OR UPDATE ON loan
    FOR EACH ROW
    EXECUTE PROCEDURE loan_item();
```

A6. MediaLibrary Triggers

3. Triggers

Triggers and user defined functions are used to automate tasks depending on changes to the database. Business rules are usually enforced using a combination of triggers and user defined functions.

Trigger	TRIGGER01
Description	An item can only be loaned to one user at a time.
SQL Code	
	<pre>CREATE FUNCTION loan_item() RETURNS TRIGGER AS \$BODY\$ BEGIN IF EXISTS (SELECT * FROM loan WHERE NEW.id_users = id_users AND end_t > NEW.start_t) THEN RAISE EXCEPTION 'An item can only be loaned to one user in every moment.'; END IF; RETURN NEW; END \$BODY\$ LANGUAGE plpgsql; CREATE TRIGGER loan_item BEFORE INSERT OR UPDATE ON loan FOR EACH ROW EXECUTE PROCEDURE loan_item();</pre>

A6. Triggers Checklist

Triggers and User Defined Functions	4.1	Triggers and functions are proposed
	4.2	Restrictions not yet covered in the schema are defined for high priority US
	4.3	For each trigger, a justification is included
	4.4	For each trigger, the SQL code is included

Transactions

A6. Transactions

- Transactions bundle multiple steps into a single, all-or-nothing operation, ensuring data integrity with concurrent accesses.
- For each necessary transaction, include:
 - Justification
 - Isolation level
 - SQL code to create it
- Common examples:
 - insert data (e.g. generalizations, latest generated PK)
 - delete data (e.g. user deletes account)
 - checkout purchase (move products from cart to purchase)
 - *many more*

Transactions Overview

Why Transactions

- Most databases operate in a multi-user context.
 - Many applications can access the data in parallel, both for reading and writing.
 - *E.g., a transfer between two bank accounts (i.e., two operations).*
- Even in single access contexts, complex operations need to be executed as a unit.
 - The system must be resilient in face of severe failures, such as power outages, disk or network failures.
 - *E.g., checking out a shopping cart requires moving items from the cart to the purchase.*
- Users should never be confronted with an inconsistent database state.
- **Transactions are a solution for both concurrency and failures.**

ACID Properties

- DBMS must support ACID properties.
- **Atomicity** guarantees that multiple operations are treated as an indivisible unit.
- **Consistency** guarantees that the database moves from one consistent state into another consistent state (clients must ensure the application logic follows the domain rules, being the primary responsible for the consistency of the system).
- **Isolation** guarantees that the outcome of multiple transactions executed concurrently is the same as if every transaction was executed in isolation.
- **Durability** refers to the fact that the effects of a committed transaction should be persisted into the database.

Transactions

- A transactions is a set of database operations that is considered as a single unit.
 - A transaction either succeeds or fails in its entirety.
 - The intermediate states between the steps of a transaction are not visible to other concurrent transactions.
 - Transactions renders a database from one consistent state into another consistent state.
- A DBMS supporting transactions includes:
 - Transaction management, i.e. features to support and control transaction execution.
 - Recovery, i.e. features to support recovery from failures.
 - Concurrency control, i.e. features to control concurrent execution of transactions.

Sample Transaction in PostgreSQL Syntax

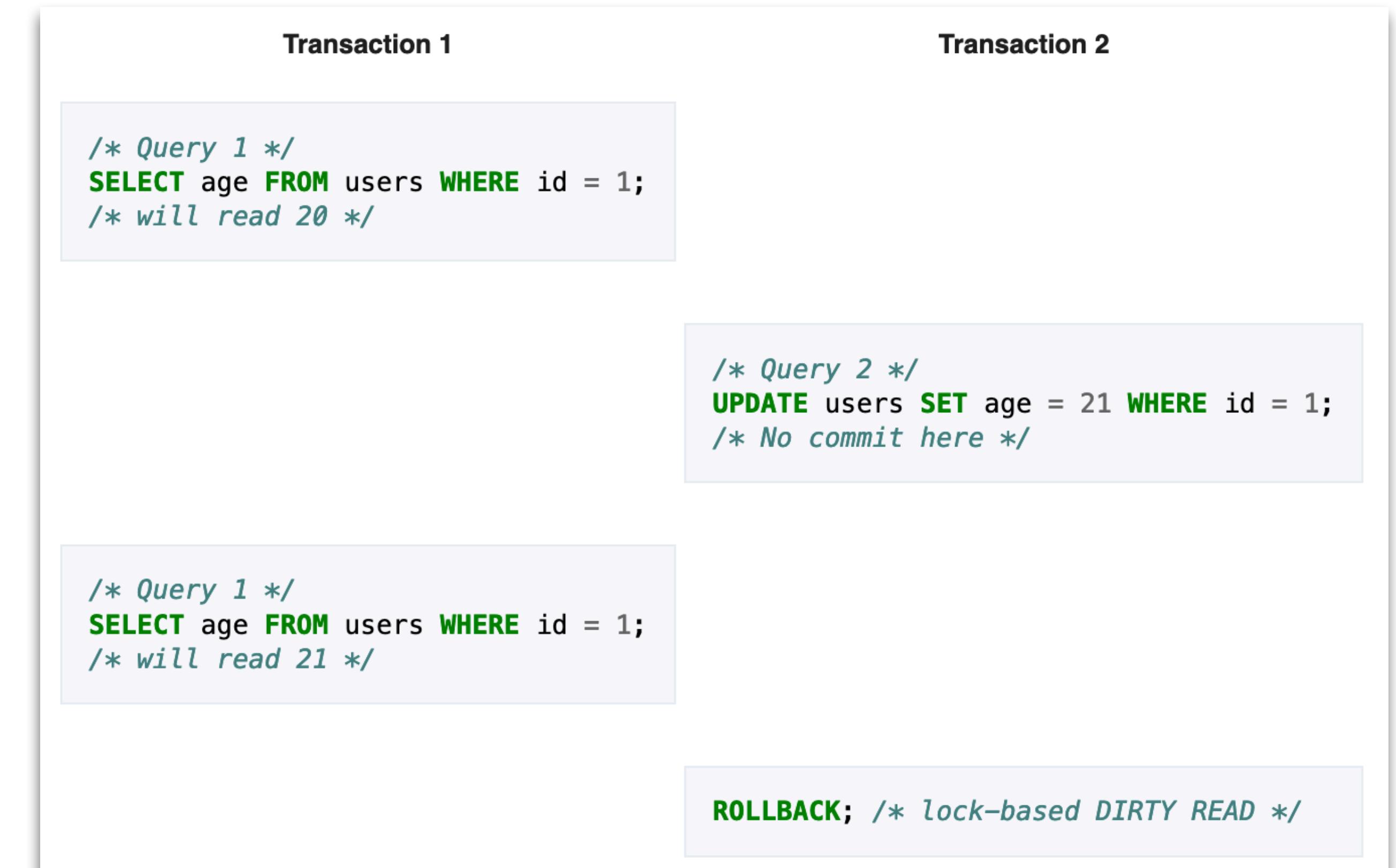
```
-- Explicitly start the transaction.  
BEGIN;  
  
UPDATE accounts SET balance = balance - 100.00  
    WHERE name = 'Alice';  
  
-- Savepoints can be used to return to during the transaction.  
SAVEPOINT my_savepoint;  
  
UPDATE accounts SET balance = balance + 100.00  
    WHERE name = 'Bob';  
  
-- Rollback to the savepoint.  
-- The previous UPDATE statement is discarded.  
ROLLBACK TO my_savepoint;  
  
UPDATE accounts SET balance = balance + 100.00  
    WHERE name = 'Wally';  
  
-- End the transaction.  
COMMIT;
```

Concurrency Problems

- Different problems can occur when concurrent transactions execute.
- **Dirty reads**, a transaction reads data written by a concurrent uncommitted transaction.
- **Non-repeatable reads**, a transaction re-reads data and finds that data has been modified by another transaction. The same query returns different results during a transaction.
- **Phantom reads**, a transaction re-executes a query and finds that the results have changed by another transaction. The values have not changed but different rows are being returned.
- **Serialization anomaly**, the result of committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

Dirty reads

- Dirty reads, a transaction reads data written by a concurrent uncommitted transaction.
- Transaction 1 will read uncommitted data that was latter rolled back by Transaction 2.



Example from Wikipedia.

[https://en.wikipedia.org/wiki/Isolation_\(database_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))

Non-repeatable reads

- Non-repeatable reads, a transaction re-reads data and finds that data has been modified by another transaction. The same query returns different results during a transaction.
- Transaction 1 will read the same data twice and obtain different values, because a concurrent transaction committed changes during the execution of Transaction 1.



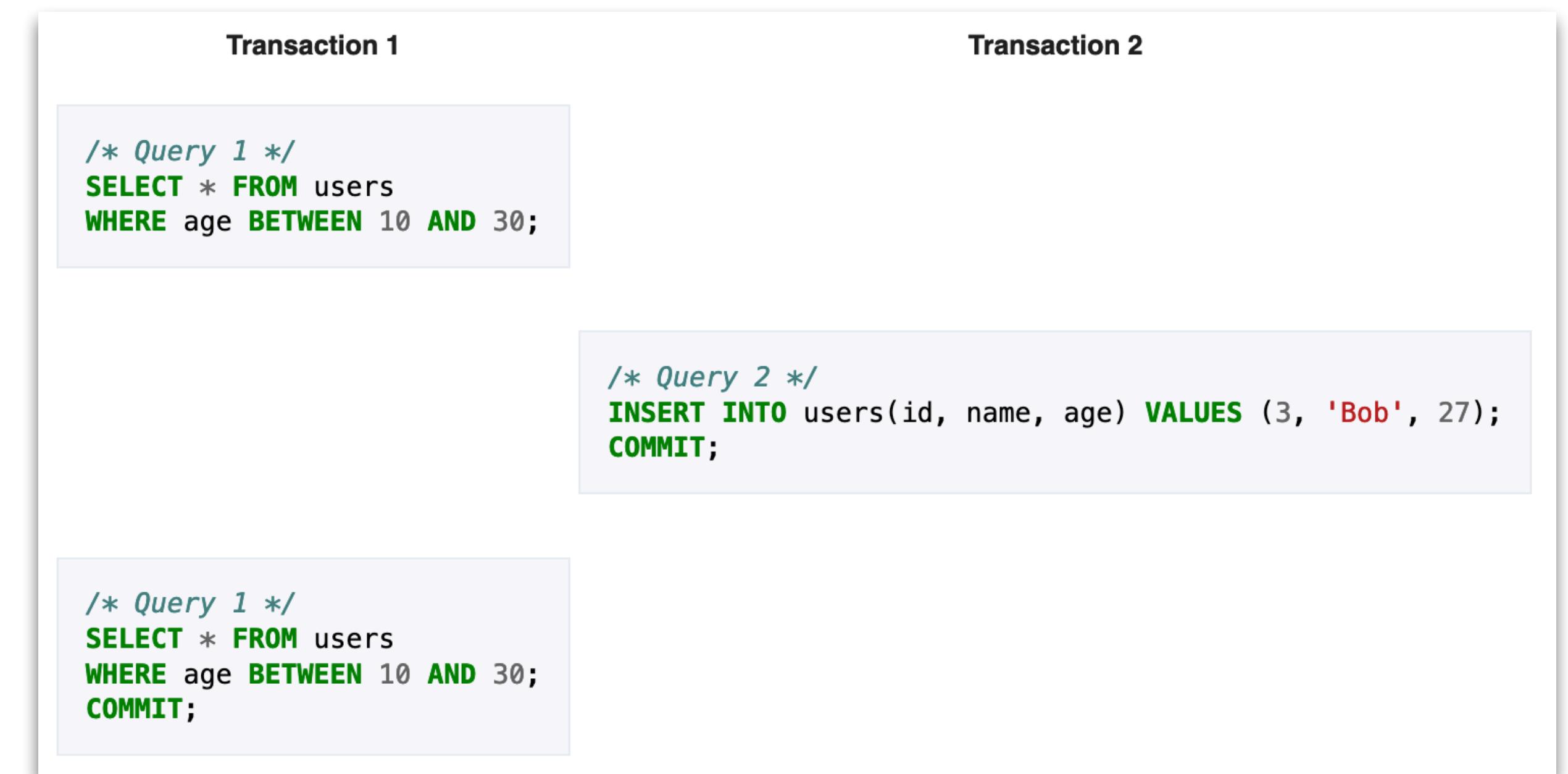
Example from Wikipedia.

[https://en.wikipedia.org/wiki/Isolation_\(database_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))

Phantom reads

- Phantom reads, a transaction re-executes a query and finds that the results have changed by another transaction. The values have not changed but different rows are being returned.

- Different values are read by Transaction 1 for the same query because Transaction 2 added new data (no changes made to existing data).



Example from Wikipedia.

[https://en.wikipedia.org/wiki/Isolation_\(database_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))

Serialization anomaly

→ Serialization anomaly, the result of committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

→ The result is inconsistent with all the possible orderings of the two transactions.

→ Using isolation level serializable in PostgreSQL solves this problem. When committing, one of the transactions would fail.

/* Transaction #1 */

/* Query 1 */
SELECT * FROM bank_teller;

id	owner	balance
1	alice	+100
2	bob	+120

/* Query 2 */
INSERT INTO bank_teller (owner, balance)
VALUES ('sum', 220);

/* Query 3 */
SELECT * FROM bank_teller;

id	owner	balance
1	alice	+100
2	bob	+120
3	sum	+220

/* Transaction #2 */

/* Query 1 */
SELECT * FROM bank_teller;

id	owner	balance
1	alice	+100
2	bob	+120

/* Query 2 */
INSERT INTO bank_teller (owner, balance)
VALUES ('sum', 220);

/* Query 3 */
SELECT * FROM bank_teller;

id	owner	balance
1	alice	+100
2	bob	+120
3	sum	+220

/* Result */

id	owner	balance
1	alice	+100
2	bob	+120
3	sum	+220
4	sum	+220

/* Expected */

id	owner	balance
1	alice	+100
2	bob	+120
3	sum	+220
4	sum	+440

Example from Wikipedia.

[https://en.wikipedia.org/wiki/Isolation_\(database_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))

Transaction Isolation

- DBMS offer different isolation levels to deal with concurrency problems (achieved mostly by locking access to tables).
- Stricter or looser isolation levels will allow less or more concurrent accesses.
- We should aim to the less restrictive isolation level that still guarantees that data is consistent
 - Advice: declare transactions as READ ONLY when possible.
- Isolation levels (from lowest/relaxed to highest/stricter):
 - **read uncommitted**, records still uncommitted can be read; Typically only allowed for read-only transactions.
In PostgreSQL read uncommitted behaves as read committed.
 - **read committed** (*default in PostgreSQL*), transactions see only data that is committed at the moment it is read; it never sees data changed, and uncommitted, by other concurrent transactions. Uses long-term write locks and short-term read locks.
 - **repeatable read**, transactions only see data committed before the transaction began; it never sees either uncommitted data or changes committed during transaction execution by concurrent transactions. Uses long-term locks for both writes and reads.
 - **serializable**, transactions see only data committed before the transaction began and never sees uncommitted data or changes;

Transaction Isolation Levels in PostgreSQL (11)

Table 13.1. Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

Transactions Summary

- The SQL standard default isolation level is serializable.
- Weaker isolation levels:
 - Increased concurrency + decreased overhead = increased performance
 - Weaker consistency guarantees
 - PostgreSQL default isolation level level is read committed.
- Stronger isolation levels:
 - Reduced concurrency + increased overhead = decreased performance
 - Strong consistency guarantees
- NOSQL systems abandon the ACID properties to achieve large scale distributed performance.

A6. MediaLibrary Transactions

4. Transactions

Transactions are used to assure the integrity of the data when multiple operations are necessary.

Transaction	TRAN01
Description	Get current loans as well as information about the items
Justification	In the middle of the transaction, the insertion of new rows in the loan table can occur, which implies that the information retrieved in both selects is different, consequently resulting in a Phantom Read. It's READ ONLY because it only uses Selects.
Isolation level	SERIALIZABLE READ ONLY
SQL Code	<pre>BEGIN TRANSACTION; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY; -- Get number of current loans SELECT COUNT(*) FROM loan WHERE now() < end_t; -- Get ending loans (limit 10) SELECT loan.end_t, loan.start_t, item.*, work.*, users.id, users.name FROM loan INNER JOIN item ON item.id = loan.id_item INNER JOIN work ON work.id = item.id_work INNER JOIN users ON users.id = loan.id_users WHERE now () < loan.end_t ORDER BY loan.end_t ASC LIMIT 10; END TRANSACTION;</pre>

A6. Transactions Checklist

Database transactions	5.1	Database transactions section is included
	5.2	Each transaction has an isolation type defined and justified
	5.3	Each transaction has a justification
	5.4	Transactions' SQL syntax is correct
	5.5	No unnecessary transactions are included
	5.6	All transactions for high priority users stories are included

A6. Database Population

- The EBD Component includes two SQL scripts
 - **Database creation script**, including SQL creation statements for all tables, key constraints, performance indexes, full text search indexes, triggers, user defined functions;
 - **Database population script**, including SQL insert statements to populate a database with test data with an amount of tuples suitable for testing and with plausible values for each field type.
- These scripts must run, *as-is*, in the production PostgreSQL environment.

A6. MediaLibrary Database Population

A.1 Database schema

```
-- Drop old schema

-----
DROP TABLE IF EXISTS wish_list CASCADE;
DROP TABLE IF EXISTS review CASCADE;
DROP TABLE IF EXISTS loan CASCADE;
DROP TABLE IF EXISTS item CASCADE;
DROP TABLE IF EXISTS nonbook CASCADE;
DROP TABLE IF EXISTS book CASCADE;
DROP TABLE IF EXISTS author_work CASCADE;
DROP TABLE IF EXISTS work CASCADE;
DROP TABLE IF EXISTS collection CASCADE;
DROP TABLE IF EXISTS author CASCADE;
DROP TABLE IF EXISTS location CASCADE;
DROP TABLE IF EXISTS publisher CASCADE;
DROP TABLE IF EXISTS users CASCADE;

DROP TYPE IF EXISTS media;

-----
-- Types

CREATE TYPE media AS ENUM ('CD', 'DVD', 'VHS', 'Slides', 'Photos',
'MP3');

-----
-- Tables

-----
-- Note that a plural 'users' name was adopted because user is a
```

A.2 Database population

```
-- Populate the database

-----
INSERT INTO users (id,email,name,obs,password,img,is_admin)
VALUES (1,'sodales.at@Curae.co.uk','Zeph Griffin','rhoncus. Donec
est. Nunc ullamcorper,','GUL95ZXR9EX','Praesent',TRUE);
INSERT INTO users (id,email,name,obs,password,img,is_admin)
VALUES (2,'aliquam.iaculis.lacus@amet.co.uk','Noah Gibson','nunc ac
mattis ornare, lectus','TYT71DOD7YN','sollicitudin',TRUE);
INSERT INTO users (id,email,name,obs,password,img,is_admin)
VALUES (3,'amet.ante@faucibusleo.net','Aladdin Davidson','nisI
elementum purus, accumsan interdum','OFK00XCC7OD','vel',TRUE);
INSERT INTO users (id,email,name,obs,password,img,is_admin)
VALUES (4,'facilisis.magna.tellus@sociis.net','Thor
Villarreal','Nunc quis arcu vel quam','PZJ77DK02VZ','Cdm',FALSE);

-- removed for brevity

-----
-- end
```

A6. Database Creation and Population Checklist

SQL	6.1	The SQL schema script is included
	6.2	The SQL script resets the database state (includes DROPs + CREATEs)
	6.3	The SQL schema script executes without errors
	6.4	The SQL population script is included
	6.5	The SQL population script is included in the group's repository
	6.6	The SQL population script executes without errors
	6.7	The SQL schema script is included in the group's repository
	6.8	The production database (at db.fe.up.pt) has been updated with the SQL scripts

PostgreSQL

Docker

- Docker is a key technology in LBAW (PostgreSQL, pgAdmin, Laravel)
- It is mandatory for deploying your prototypes and final products
- Docker is a lightweight virtualization environment, widely used to package applications and its dependencies in isolated containers.
- With Docker you can manage your product infrastructure as applications.
- Available for Windows, Mac and Linux - <https://docs.docker.com/get-docker/>
- Important: don't postpone using Docker.

PostgreSQL Docker Container

→ Official PostgreSQL are available at: https://hub.docker.com/_/postgres

→ Start a local PostgreSQL server with:

→ `docker run --name pgsql11 -e POSTGRES_PASSWORD=mysecretpassword -d postgres:11`

→ Run a local pgAdmin installation (available at localhost:80) with:

→ `docker run -p 80:80 \
-e 'PGADMIN_DEFAULT_EMAIL=user@domain.com' \
-e 'PGADMIN_DEFAULT_PASSWORD=SuperSecret' \
-d dpage/pgadmin4`

Docker Compose

- Docker Compose is used to setup multi-container Docker applications.
- A YAML file is used to configure the containers to start and run.
- The LBAW 'template-postgresql' repository, sets up two containers - <https://git.fe.up.pt/lbaw/template-postgresql>

```
version: '3'
services:

  postgres:
    image: postgres:11.13
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pg!password
    ports:
      - "5432:5432"

  pgadmin:
    image: dpage/pgadmin4:6
    environment:
      PGADMIN_DEFAULT_EMAIL: postgres@lbaw.com
      PGADMIN_DEFAULT_PASSWORD: pg!password
    ports:
      - "4321:80"
    depends_on:
      - postgres
```

About the PostgreSQL Production Environment (**important!**)

review

- A PostgreSQL database contains one or more schemas, which in turn contains one or more tables.
- All databases contain a public schema, which is used as default.
- In PostgreSQL's command line interface, you can view the current active schema with: **show search_path;**
- To change the schema for the current session use: **SET search_path TO <schema>;**
- **In the PostgreSQL setup at FEUP (db.fe.up.pt), the public schema is shared between all accounts,**
 - Tables created in the public schema are visible to all users (although not accessible).
If you look at the tables in the public schema, you will find a long list of tables.
 - **It is important to not use the public schema and instead create a schema with the name of your group (lbaw21gg).**
- To create this schema, use the following command: **CREATE SCHEMA <lbaw21gg>;**
- To always use this schema as the default in your project, add the following line to the beginning of your SQL scripts.
 - **SET search_path TO <lbaw21gg>;**

References

Bibliography and Further Reading

- PostgreSQL Manual, Chapter 11. Indexes, <https://www.postgresql.org/docs/current/indexes.html>
- PostgreSQL Manual, Chapter 12. Full Text Search, www.postgresql.org/docs/current/textsearch.html
- Scott Ambler, The Object Primer, Cambridge University Press, 3rd Edition, 2004.
- UML – Metodologias e Ferramentas CASE (2^a Edição)
Alberto Rodrigues da Silva, Carlos Videira, Centro Atlântico Editora, Maio 2005.
- Database Management Systems
Raghu Ramakrishnan, Johannes Gehrke. McGRAW-Hill International Editions, 3rd Edition, 2003.
- Principles of Database Management
Wilfried Lemahieu, Seppe Vanden Broucke, Bart Baesens. Cambridge University Press, 2018.

Lab Class #4

- Discuss the conceptual data model (A4)
- Develop and discuss the relational schema (A5)
 - Map the classes and relationships of the conceptual schema into relation schemas
 - For each relation, identify the functional dependencies (FD) that apply
 - Check if each relation is in BCNF
 - If the relation is not in BCNF and there are no other impediments, look for several possible decompositions (lossless)
 - If there is no satisfactory decomposition to BCNF and if the relation is no longer in 3NF, consider the decomposition lossless for 3NF, preserving the functional dependencies
 - Develop and test a first version of the database creation script in SQL
- Test the local development environment for PostgreSQL.
- Test the connection to the production PostgreSQL server at db.fe.up.pt

Web Applications

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Current Status

- Plan:
 - 5th week of classes;
 - Lecture: database modeling? + database transactions + web applications (start);
 - Labs: continue database specification (EBD = A4 + A5 + A6).
 - In two weeks: EBD delivery.
- Monitor sessions: Tuesdays, at 16h00, in room I003.
 - Previous sessions: Git and GitFlow; (participation?)
 - This week: PostgreSQL setup and use + questions you bring.

Outline for Today

- Web Applications
 - History
 - Technologies
 - Architectures
- Architecture Specification and Prototype (EAP) component
- A7: Web Resources Specification

LBAW Artifacts

1. Requirements & UI

A1: Project Presentation

A2: Actors and User Stories

A3: Information Architecture

2. Database Specification

A4: Conceptual Data Model

A5: Relational Schema

A6: Indexes and Triggers

3. Web Architecture & Prototype

A7: Web Resources Specification

A8: Vertical Prototype

4. Product and Presentation

A9: Product

A10: Presentation and Discussion

Introduction

Web and Internet

- What is the Internet?
- What is the Web?
- The **Internet** is the worldwide communication network made of interconnected physical networks — a network of networks.
- The **World Wide Web** is a distributed information system that runs over the Internet.

Who is who?



Who is who?



Web Applications

- What are Web Applications?
- A **Web Application**, or web app, is a software system, based on web standards and technologies, that is accessible through a web browser.
- Web applications changed significantly over time due to technological advancements, namely asynchronous interactions (AJAX), JavaScript developments, the new HTML5 standards, and the broad adoption of mobile devices.
- Examples: GMail, Google Search, Facebook, SIGARRA.
- Why web applications? Advantages and disadvantages?

Pros of Web Apps

- Platform independence.
- Easier updates & bug fixes.
- Only one version of the application.
- Access from anywhere.
- Reduced "piracy".
- No installation hurdles.
- Developers can measure user interaction in real-time.

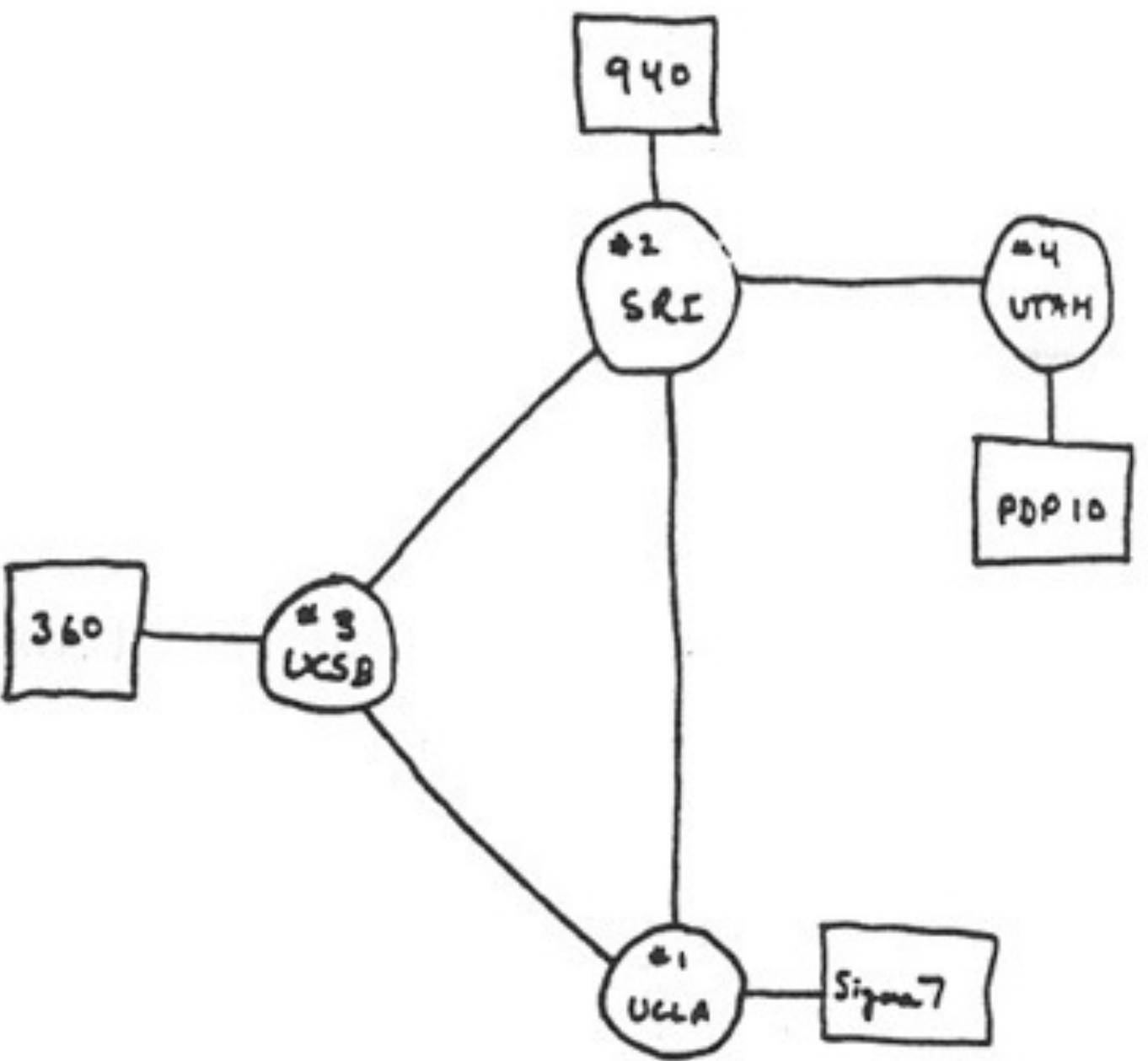
Cons of Web Apps

- Depends on network connectivity.
- Less sophisticated UIs.
- Limited hardware access.
- Reduced OS integration (e.g. drag&drop).
- Need to address browser versions.
- Harder to debug.
- Higher security risks.
- Infrastructure costs.

The Internet

Internet Origins

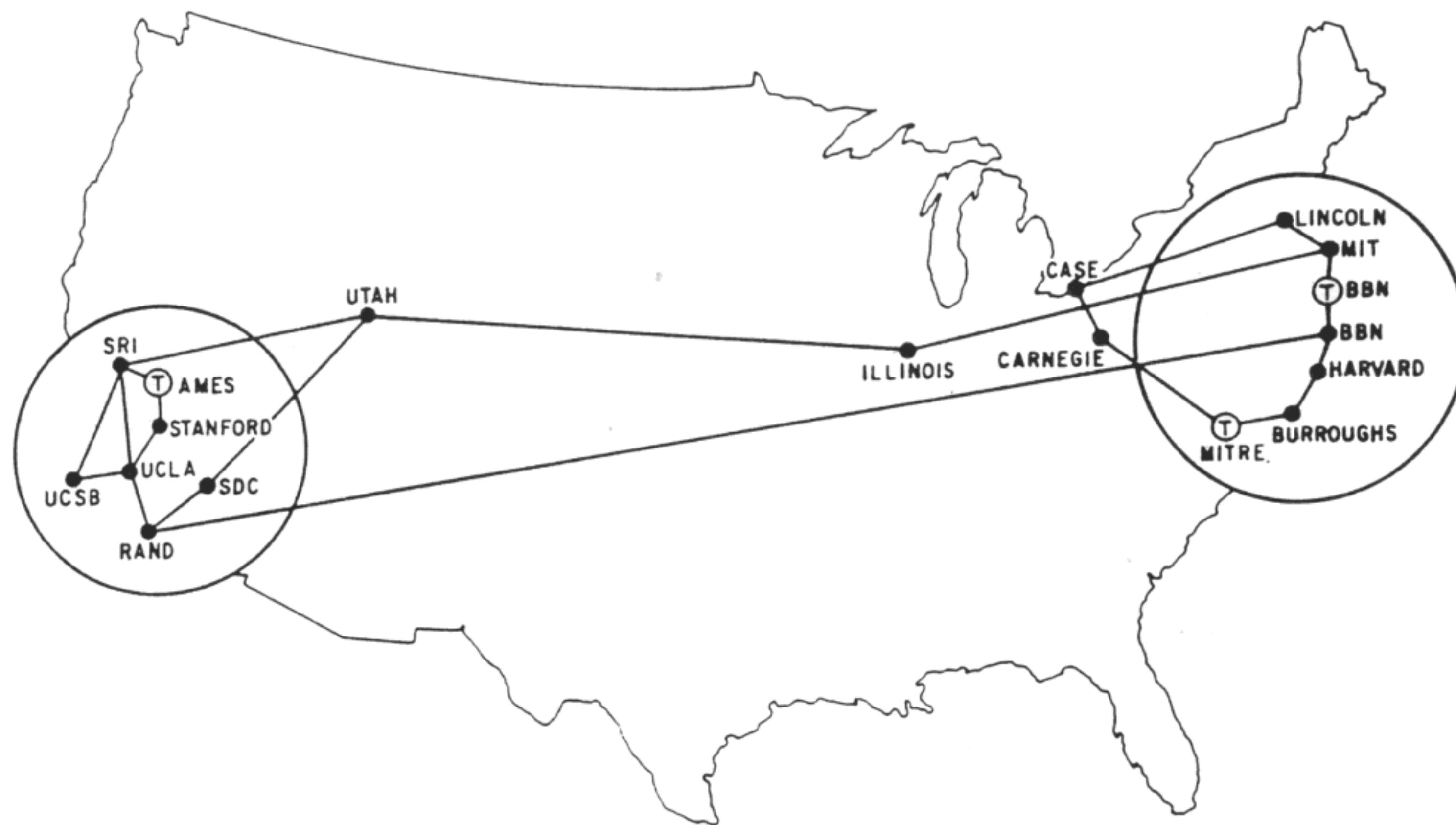
- Started as a project of the USA Department of Defense (DoD) Advanced Research Projects Agency (ARPA) in 1958.
- Developed the idea of “networked computers” which led to the creation of the ARPANET in 1967. The core challenge was how to connect separate physical networks without requiring permanent links between them.
- Other networks appeared worldwide during the same period.
E.g. JANET, X.25, USENET, CompuServe.



THE ARPA NETWORK

DEC 1969

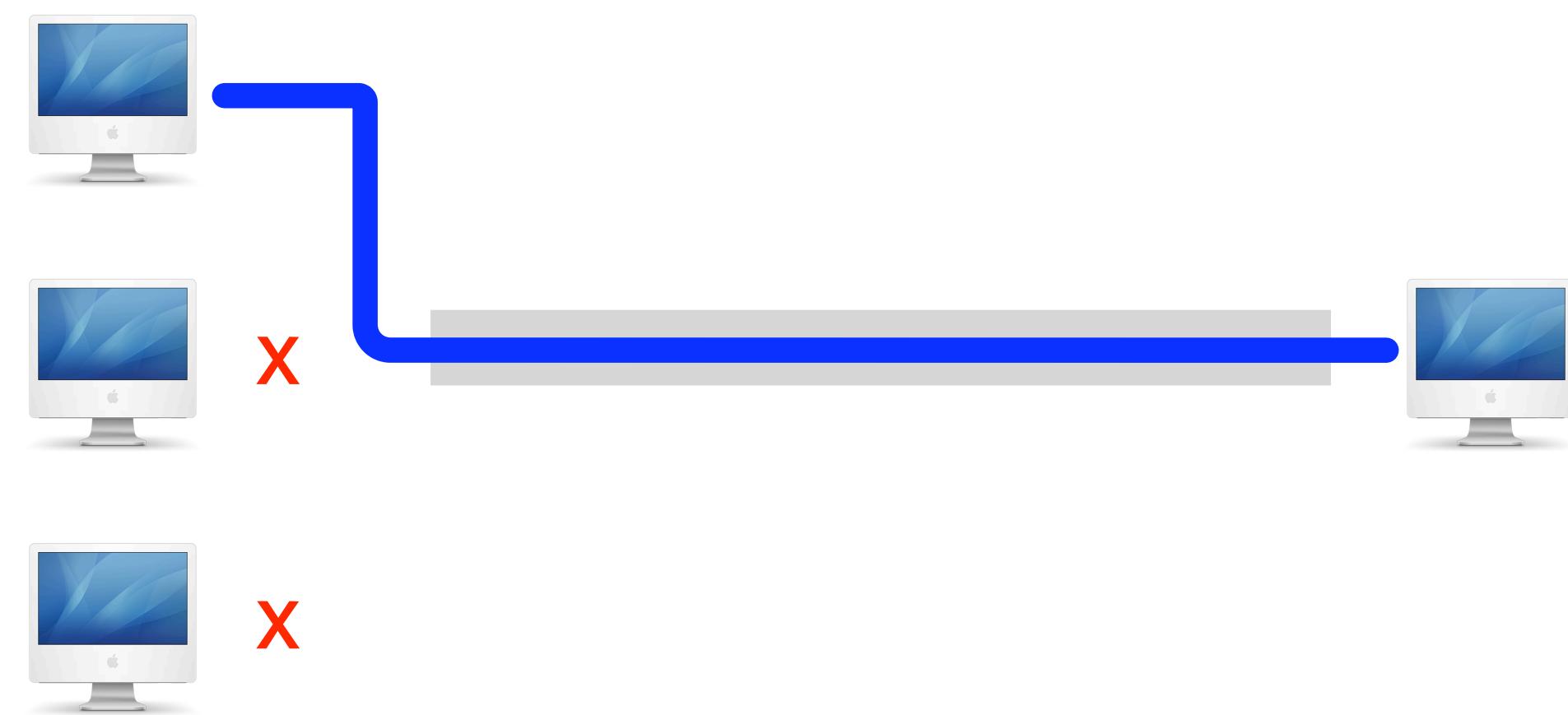
4 NODES



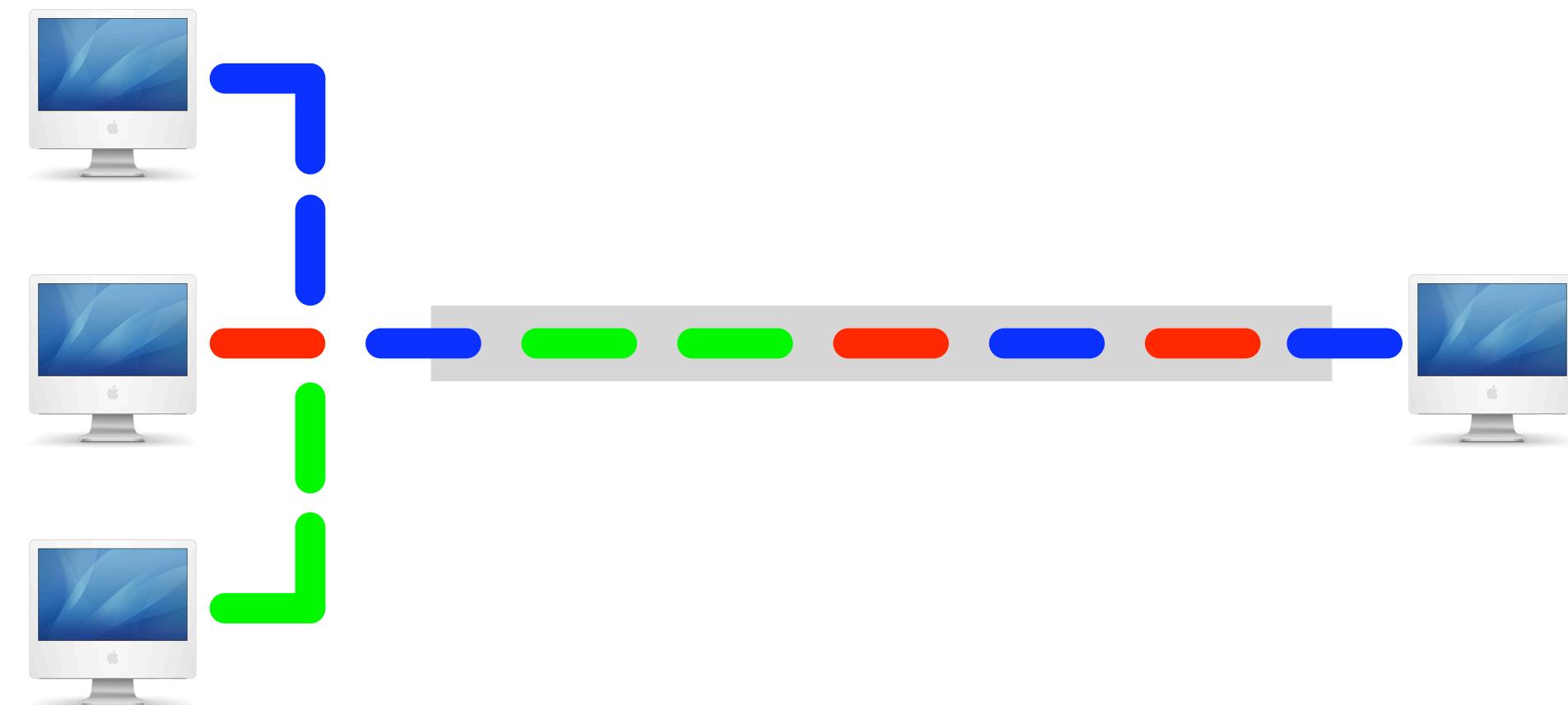
MAP 4 September 1971

Packet-Switching

- In the late 1960s, computers were expensive and rare. Communication networks were used to make these resources available to wider audiences.
- One of the first challenges faced was how to connect separate physical networks without dedicated links.
- Implementing point-to-point connections does not scale. The only option was to share the available links to optimize resource usage.
- **Packet switching** is a method where data is divided in small chunks and sent out separately.
- With packet switching it is possible to have multiple connections over the same link, i.e. share the communication medium.



Continuous transmission



Packet-based transmission

Internetworking

- The proliferation of different networking technologies and protocols became a problem when trying to connect different networks.
- No network technology is ideal for all scenarios (e.g. Ethernet, Wireless, DSL), thus different technologies will always co-exist. Each network technology was becoming an island, isolated from all others.
- To overcome this problem, and achieve a homogeneous service across heterogeneous networks, both **hardware** and **software** were combined.
- Hardware: routers.
- Software: protocols, specifically TCP/IP.

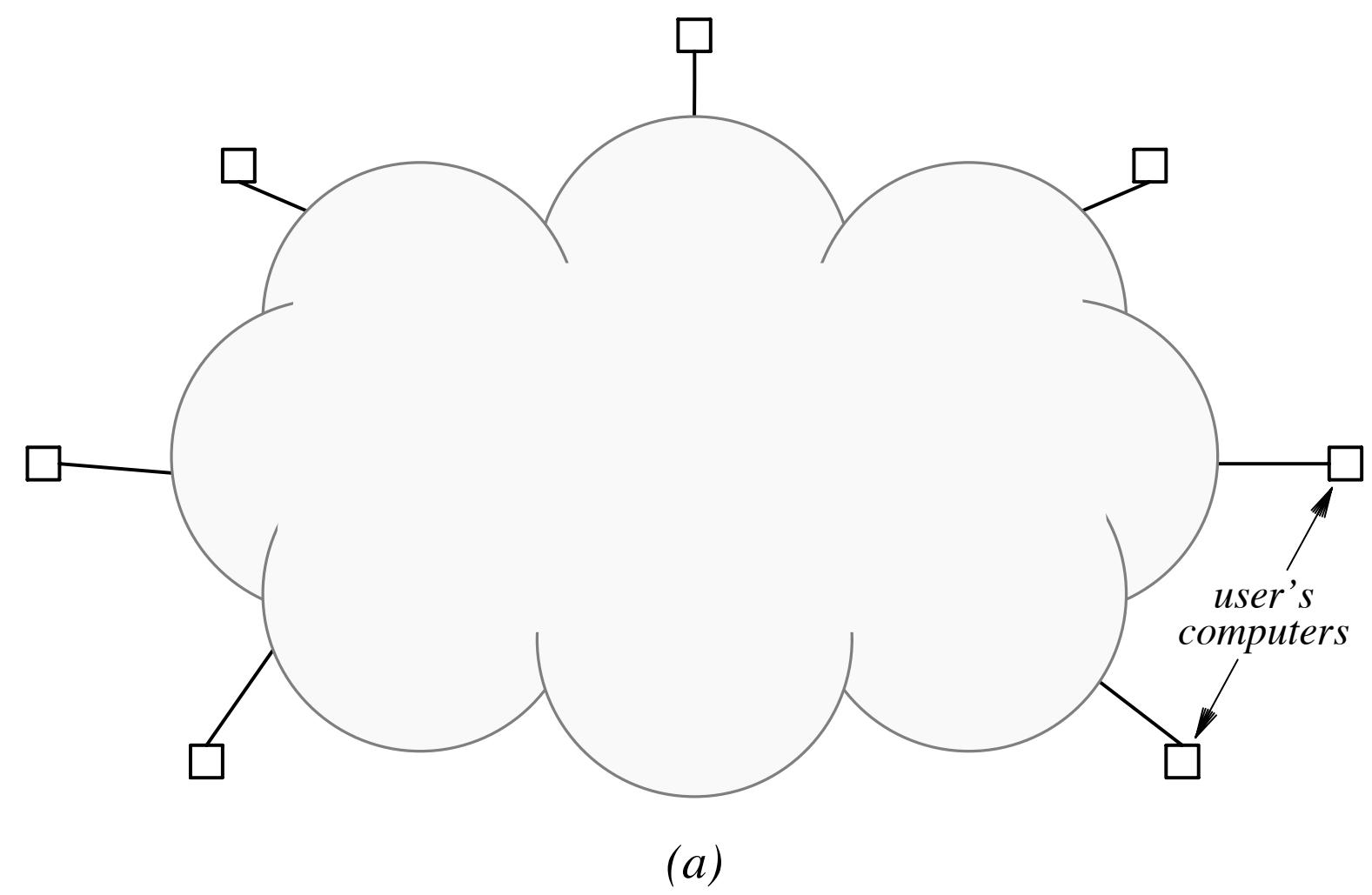
Hardware – Routers

- The router is the core hardware equipment used to connect networks using different physical technologies. There is a vast number of router types.
- An internet (note the lowercase) is a set of networks connected by routers.

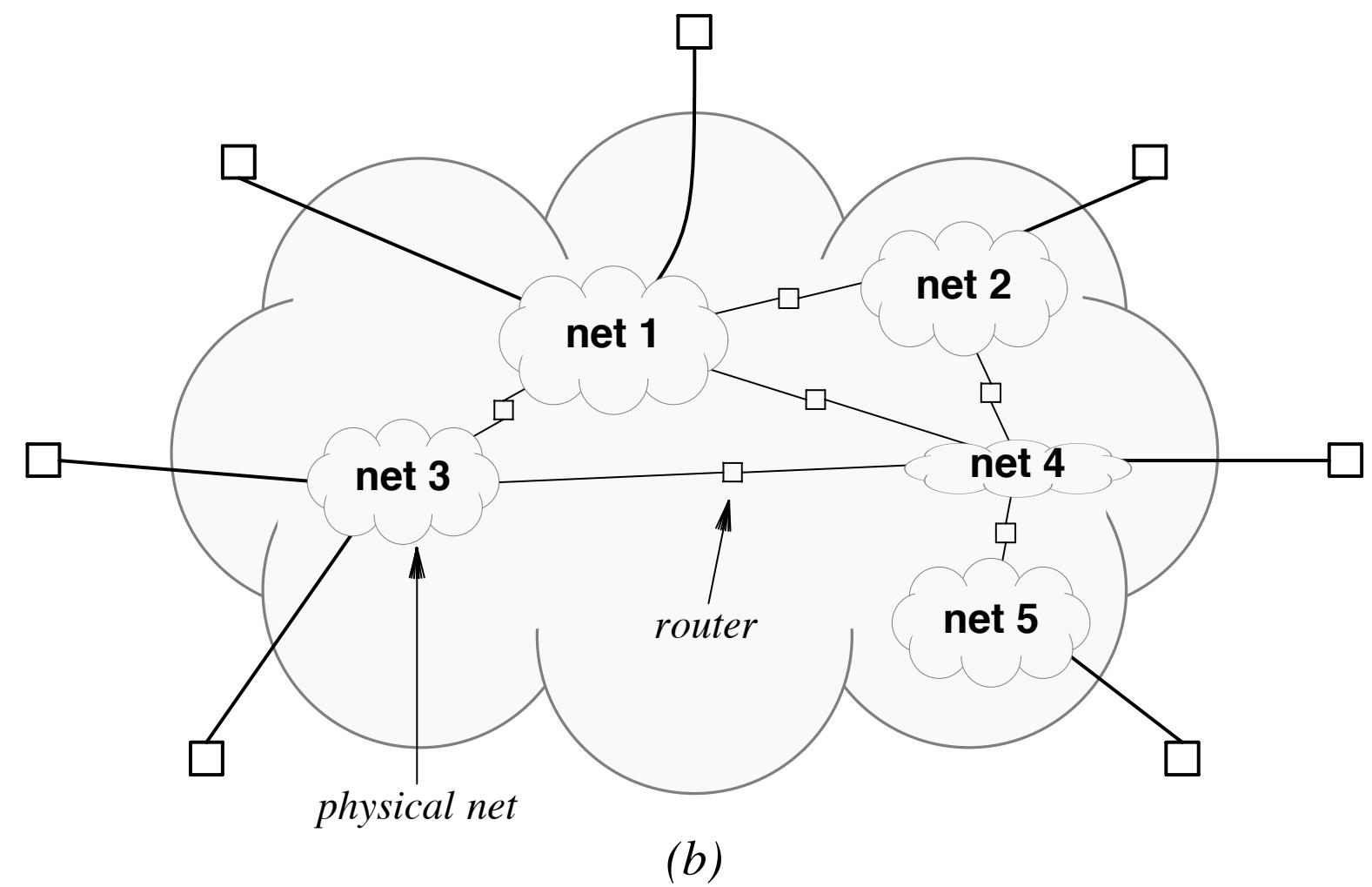


Software – Protocols

- To connect different networks we need communication protocols.
- These protocols establish message formats and message exchanging rules.
- The most important protocols for connecting different networks are called the Internet Protocols or TCP/IP Protocols.
- These protocols were developed in the 1970s and approved as standards in the 1980s.

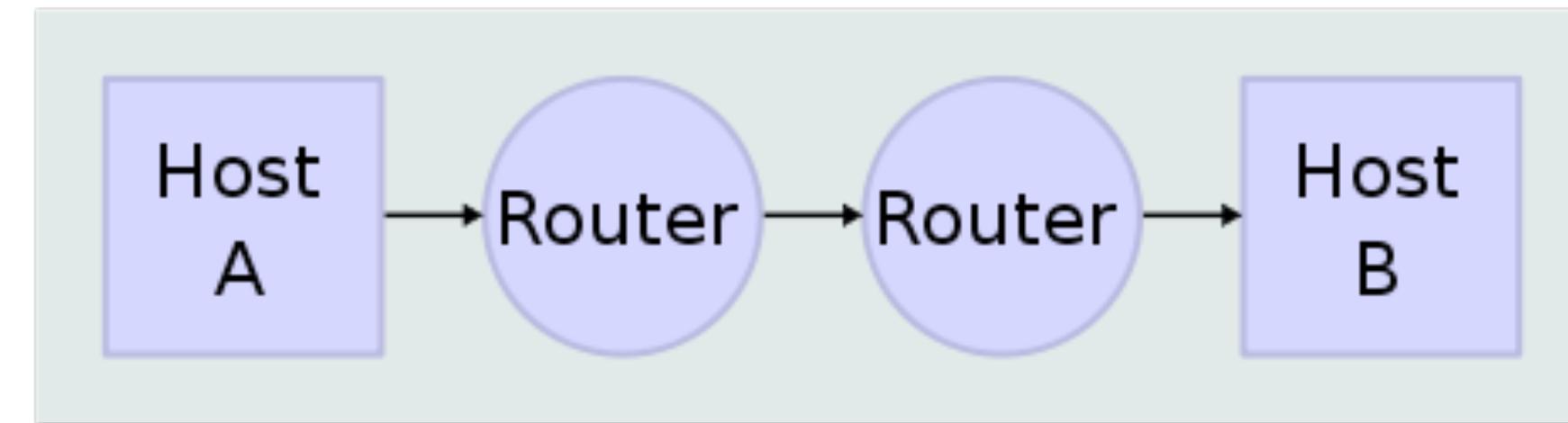


An internet is a virtual network because in reality it is built by combining many physical networks.

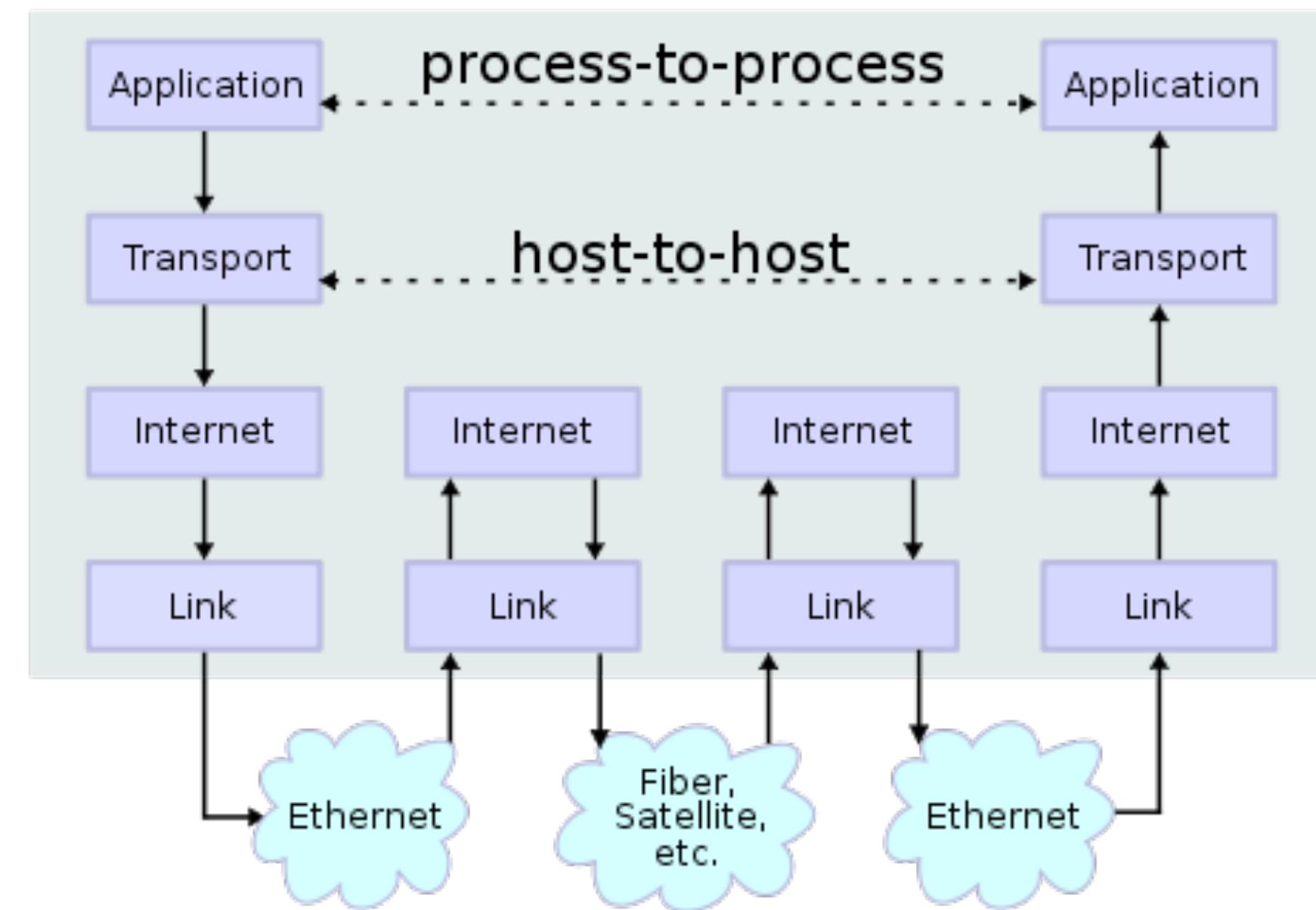


An internet is a network of networks, not a network of computers.

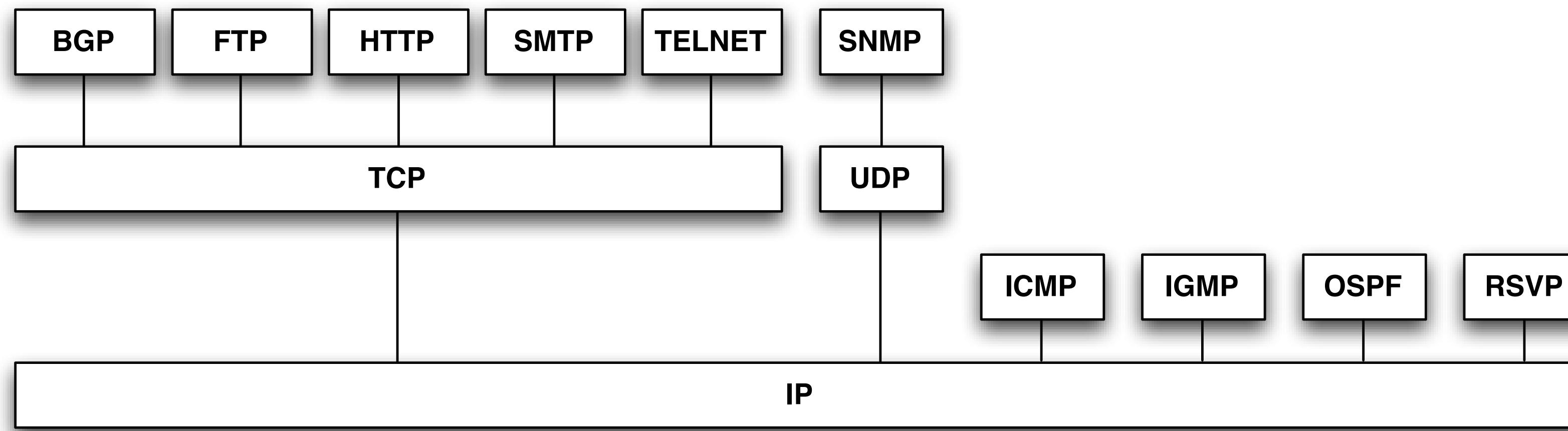
Network Topology



Data Flow



Internet Protocols



BGP = Border Gateway Protocol

FTP = File Transfer Protocol

HTTP = HyperText Transfer Protocol

ICMP = Internet Control Message Protocol

IGMP = Internet Group Management Protocol

IP = Internet Protocol

OSPF = Open Shortest Path First

RSVP = Resource ReSerVation Protocol

SMTP = Simple Mail Transfer Protocol

SNMP = Simple Network Management Protocol

TCP = Transmission Control Protocol

UDP = User Datagram Protocol

Internet Protocol (IP)

- The main function of the Internet Protocol is to offer a virtual network, hiding the underlying physical networks.
- It offers two fundamental services:
 - Addressing system (IP addresses).
 - Datagram structure (packets).

Transmission Control Protocol (TCP)

- The Transmission Control Protocol offers a reliable and ordered delivery of packets between applications in different computers.
- Handles problems not addressed in the lower layers: packet duplication and loss, packet ordering, communication delays, among others.
- Supports important applications such as the WWW, e-mail, FTP, etc.

Internet Services

- DNS – Maps IP addresses to symbolic names.
- SMTP – Handles electronic messages (e-mail).
- SFTP – Offers a mechanism for secure file transfers between computers.
- WWW (HTTP) – A hypertext-based distributed information system.
- ...

Domain Name System (DNS)

- The Domain Name System is an application layer service of the Internet.
- Translates human-readable symbolic names to numeric addresses (IP).
- Symbolic names are organized hierarchically.
The right-most element is the top-level domain (TLD).
- There are different groups of TLDs, namely country TLDs and generic TLDs.
- Country TLDs (ccTLD): .pt, .es, .uk, .usa, .fr, .it, .ao, .fi, .io, .tv ...
- Generic TLDs (gTLD): .com, .net, .org, .name, .biz ...

Generic TLDs

- Generic top-level domains (TLDs) are maintained by the Internet Assigned Numbers Authority (IANA), a department of the Internet Corporation for Assigned Names and Numbers (ICANN), a nonprofit private USA organization.
- Initial list of general purpose domains (1984): com, edu, gov, mil, org and net.
- Recent gTLDs: .biz, .info, .jobs, .mobi, .name, .guru, .today, .xyz ...
<https://www.gandi.net/en/tlds>
- In 2012, ICANN decided to open the creation of new gTLDs through a process where organizations can propose new strings.
There is some debate around this expansion.
- There are currently more than 1,500 TLDs.

.pt Domain

- .pt is the country code top-level domain (ccTLD) for Portugal.
- The .pt ccTLD is currently managed by Associação DNS.PT, which replaced the Fundação para a Computação Científica Nacional (FCCN) as the domain name registry in Portugal.
- FCCN kept very strict rules during the first years. Multiple adjustments have been made during the years. FCCN also tried to promote the .com.pt subdomain as a more flexible solution.
- In March 2019 there were approximately 350,000 active .pt domain names. More than a one million registered. <https://www.dns.pt/estatisticas>

WHOIS Protocol

- WHOIS is a query/response protocol used to query databases that contain information about internet resources, such as domain names or IP addresses.
- The protocol presents the information in a human-readable format.
- It is the *de facto* standard for querying domain name information.

Pesquisa Domínio / WHOIS

Pesquise os Domínios disponíveis e obtenha informação sobre os titulares já registados.

Nome do Domínio .pt
à á â ã ç é ê í ó ô ú

Mais Pesquisas

Estou em: Homepage » WHOIS

WHOIS

Domínio	up.pt
Data Submissão	11-10-1991
Data de Expiração	30-05-2016
Estado	ACTIVE
Titular	Universidade do Porto Reitoria - Praça Gomes Teixeira Porto 4099-002 Porto contacto_dnsup@reit.up.pt
Entidade Gestora	Universidade do Porto contacto_dnsup@reit.up.pt
Responsável Técnico	José António Pacheco e Sousa jasousa@reit.up.pt
Informação do Nameserver	up.pt NS dns1.up.pt. up.pt NS dns4.up.pt. up.pt NS dns2.up.pt. dns1.up.pt. A 193.137.55.20 dns4.up.pt. A 193.136.37.10 dns1.up.pt. AAAA 2001:690:2200:a10::20 dns2.up.pt. A 193.137.55.21 dns2.up.pt. AAAA 2001:690:2200:a10::21 dns4.up.pt. AAAA 2001:690:2200:910::10

<https://www.dns.pt/pt/ferramentas/whois/detalhes/?site=up&tld=.pt>

Whois Search Results

Search again (.aero, .arpa, .asia, .biz, .cat, .com, .coop, .edu, .info, .int, .jobs, .mobi, .museum, .name, .net, .org, .pro, or .travel) :

- Domain (ex. internic.net)
- Registrar (ex. ABC Registrar, Inc.)
- Nameserver (ex. ns.example.com or 192.16.0.192)

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered with many different competing registrars. Go to <http://www.internic.net> for detailed information.

Domain Name: FACEBOOK.COM
Registrar: MARKMONITOR INC.
Whois Server: whois.markmonitor.com
Referral URL: <http://www.markmonitor.com>
Name Server: A.NS.FACEBOOK.COM
Name Server: B.NS.FACEBOOK.COM
Status: clientDeleteProhibited
Status: clientTransferProhibited
Status: clientUpdateProhibited
Status: serverDeleteProhibited
Status: serverTransferProhibited
Status: serverUpdateProhibited
Updated Date: 28-sep-2012
Creation Date: 29-mar-1997
Expiration Date: 30-mar-2020

>>> Last update of whois database: Tue, 18 Mar 2014 23:01:39 UTC <<<

https://reports.internic.net/cgi/whois?whois_nic=facebook.com&type=domain

who.is Search Domain name or IP address 

193.137.55.1 address profile

Overview Diagnostics

Overview for 193.137.55.1
Updated 40 seconds ago

```
% This is the RIPE Database query service.  
% The objects are in RPSL format.  
%  
% The RIPE Database is subject to Terms and Conditions.  
% See http://www.ripe.net/db/support/db-terms-conditions.pdf  
  
% Note: this output has been filtered.  
%       To receive output for a database update, use the "-B" flag.  
  
% Information related to '193.137.48.0 - 193.137.55.255'  
  
% Abuse contact for '193.137.48.0 - 193.137.55.255' is 'report@cert.pt'  
  
inetnum:      193.137.48.0 - 193.137.55.255  
netname:      PTUNET-UP-6  
descr:        Universidade do Porto  
descr:        Porto  
country:      PT  
admin-c:      LMR2-RIPE  
tech-c:       FC1899-RIPE  
tech-c:       FL514-RIPE  
tech-c:       RC14028-RIPE  
tech-c:       RR9044-RIPE  
status:       ASSIGNED PA  
remarks:      SERVIP-UP  
remarks:      created 19951220  
mnt-by:       AS1930-MNT  
mnt-lower:    AS1930-MNT  
source:       RIPE # Filtered  
  
person:       Fernando Correia  
address:      Universidade do Porto - Reitoria  
address:      Praca Gomes Teixeira  
address:      4099-002 Porto  
address:      PORTUGAL  
phone:        +351 220408000  
fax-no:       +351 220408186  
nic-hdl:      FC1899-RIPE  
mnt-by:       AS1930-MNT  
source:       RIPE # Filtered
```

The RIPE database contains registration details of IP addresses.

Domain Name Registrars

- Domains are reserved and managed by accredited organizations according to the rules of each specific TLD.
- Domain names are rented for a limited period (e.g. 1 year).
At the end of the registration period, the owner can decide to renew it or not.
- Different TLDs have different registration prices, some examples:
.com, .net, .org (~8€/year), .pt (~15€/year), .io (~90€/year), .biz (~15€/year).

Prices for Portugal, 23% VAT tax included, in € (EUR), for A rates. [Change](#)

Extensions	Creation	Renewal	Transfer
<input type="text" value="Find your own domain name"/>	Bulk registration		
.com	€15.42	€15.99	€11.73
.pt	€20.44	€20.44	Free
 .site	€13.53 €1.22	€35.74	€35.74
 .net	€20.91 €10.46	€20.91	€17.22
.io	€39.36	€39.36	€36.90
.fr	€14.76	€14.76	€14.76
 .club	€17.50 €4.92	€17.50	€14.83
.app	€24.48	€24.48	€20.74
 .online	€14.76 €7.37	€52.96	€44.88
.page	€16.45	€16.45	€13.94
.museum	€82.15	€82.15	€69.62
Want to see prices for our 700+ extensions?		See more	

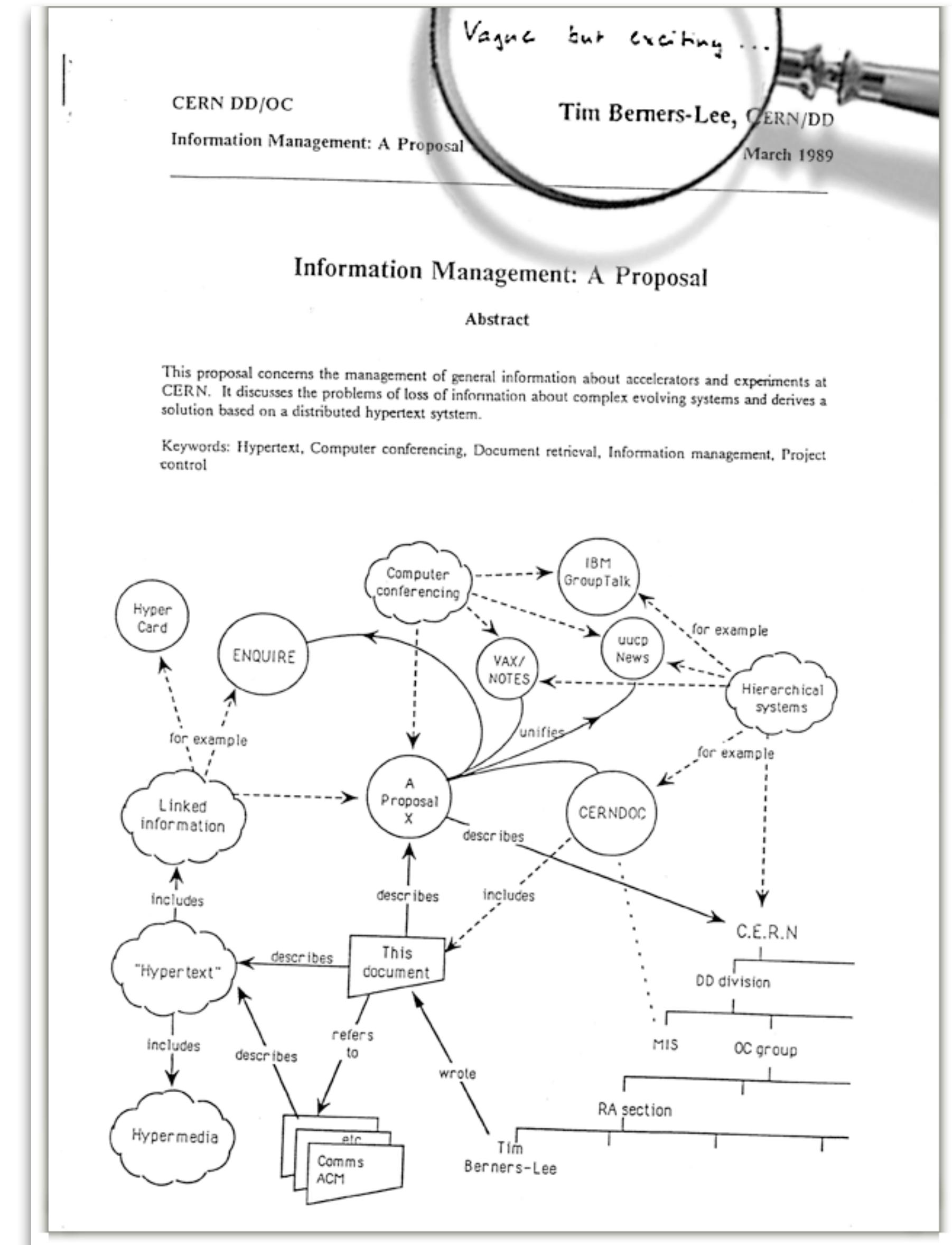
Registo e renovação de domínios

Domínio	1 ano	2 anos	3 anos	5 anos
.pt	19,95 €	–	59,00 €	98,00 €
.com.pt, .org.pt*	19,95 €	–	59,00 €	98,00 €
.com, .net, .biz	19,95 €	39,50 €	59,00 €	98,00 €
.org	19,95 €	39,50 €	59,00 €	98,00 €
.info	19,95 €	39,50 €	59,00 €	98,00 €
.eu	19,95 €	39,50 €	59,00 €	98,00 €
.me	22,95 €	45,90 €	68,85 €	114,75 €
.tv	32,95 €	65,90 €	–	164,75 €
.co.uk	19,95 €	39,50 €	59,00 €	98,00 €
.co	27,95 €	55,90 €	83,85 €	139,75 €
.es	19,95 €	39,50 €	59,00 €	98,00 €
.xxx	90,00 €	156,00 €	234,00 €	385,00 €

The World Wide Web

WWW Origins

- The World Wide Web was invented in 1989 at the European Council for Nuclear Research (CERN), Europe.
- It was a joint work by Tim Berners-Lee and Robert Cailliau to share and link information of various kinds, where the user could browse at will.
- Basically, a distributed information system over the Internet, designed to facilitate content sharing across different computer systems and technologies.
- Initial proposal “WorldWideWeb”, or simply WWW or W3.
- “Information Management: A Proposal”, May 1989 <http://www.w3.org/History/1989/proposal.html>
- 30 years in 2019. See: <https://web30.web.cern.ch>



The original proposal: “Vague but exciting...”

Initial Success

- The WorldWideWeb source code was released into the public domain in 1993. The software that was open-sourced included a simple client, a server, and a common library. The protocols were also released royalty-free.
- The royalty-free license was a key factor in the initial success of the WWW when compared to similar alternatives, e.g. WAIS, Gopher, ARCHIE.
- The NCSA released Mosaic, a software program that was a combined WWW browser and Gopher client.
- Mosaic's popularity was determinant to the growth of the World Wide Web. Mosaic introduced significant innovations at the graphical interface level, namely the integration of text and images in a single page.

930480

**ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH**

STATEMENT CONCERNING CERN W3 SOFTWARE RELEASE INTO PUBLIC DOMAIN

TO WHOM IT MAY CONCERN

Introduction

The World Wide Web, hereafter referred to as W3, is a global computer networked information system.

The W3 project provides a collaborative information system independent of hardware and software platform, and physical location. The project spans technical design notes, documentation, news, discussion, educational material, personal notes, publicity, bulletin boards, live status information and numerical data as a uniform continuum, seamlessly integrated with similar information in other disciplines.

The information is presented to the user as a web of interlinked documents .

Access to information through W3 is:

- via a hypertext model;
- network based, world wide;
- information format independent;
- highly platform/operating system independent;
- scalable from local notes to distributed data bases.

Webs can be independent, subsets or supersets of each other. They can be local, regional or worldwide. The documents available on a web may reside on any computer supported by that web.

...

Declaration

The following CERN software is hereby put into the public domain:

- W3 basic ("line-mode") client
- W3 basic server
- W3 library of common code.

CERN's intention in this is to further compatibility, common practices, and standards in networking and computer supported collaboration. This does not constitute a precedent to be applied to any other CERN copyright software.

CERN relinquishes all intellectual property rights to this code, both source and binary form and permission is granted for anyone to use, duplicate, modify and redistribute it.

CERN provides absolutely NO WARRANTY OF ANY KIND with respect to this software. The entire risk as to the quality and performance of this software is with the user. IN NO EVENT WILL CERN BE LIABLE TO ANYONE FOR ANY DAMAGES ARISING OUT THE USE OF THIS SOFTWARE, INCLUDING, WITHOUT LIMITATION, DAMAGES RESULTING FROM LOST DATA OR LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES.

Geneva, 30 April 1993


W. Hoogland
Director of Research


H. Weber
Director of Administration

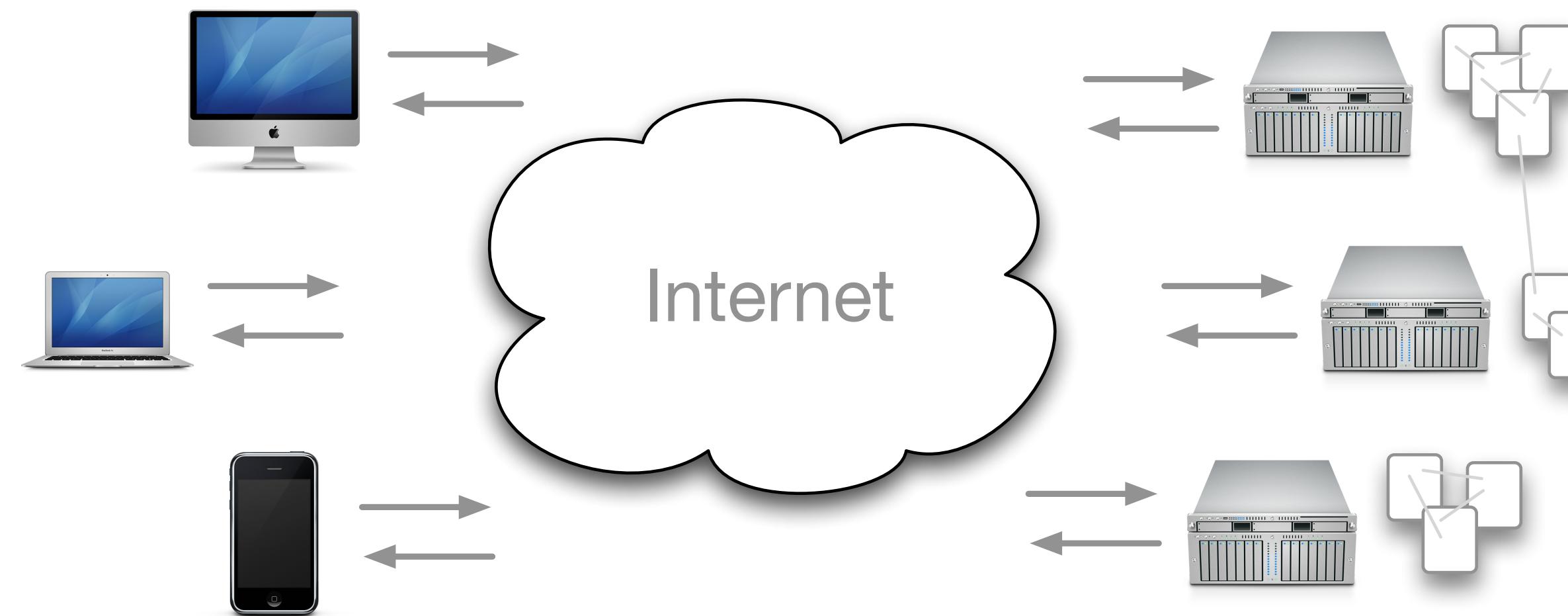
copie certifiée conforme
fait à Genève le 03-05-93



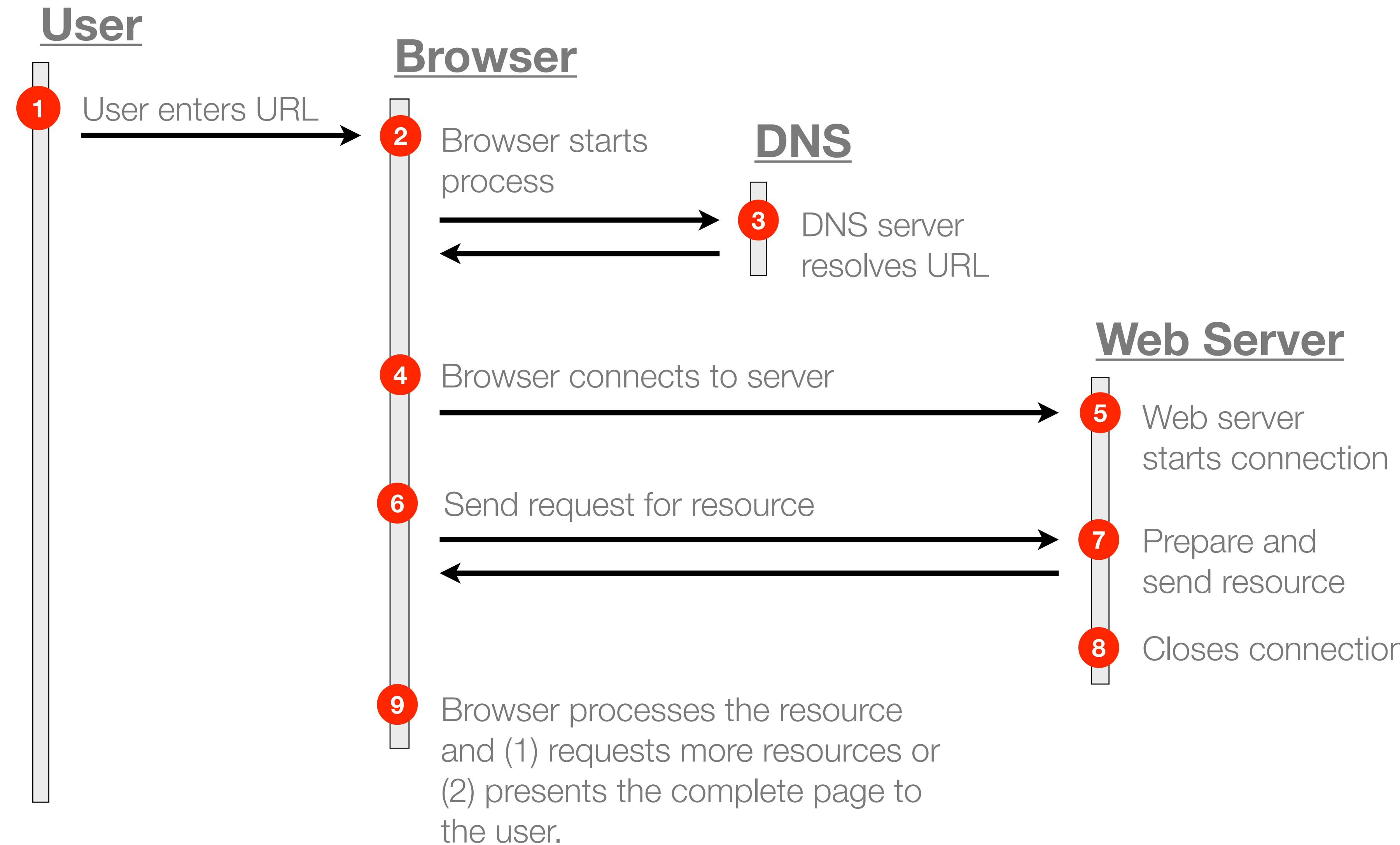
April 1993, CERN puts the WWW in the public domain

WWW Architecture

- The Web's architecture follows a standard **client-server model**, where servers provide a function and clients initiate requests for those services.
- **Servers** are machines that are running server applications waiting for requests from clients. Each server can simultaneously serve multiple clients.
- **Clients** are typically web browsers that initiate the communication session with servers. Interactions are simple, one request results in one response.

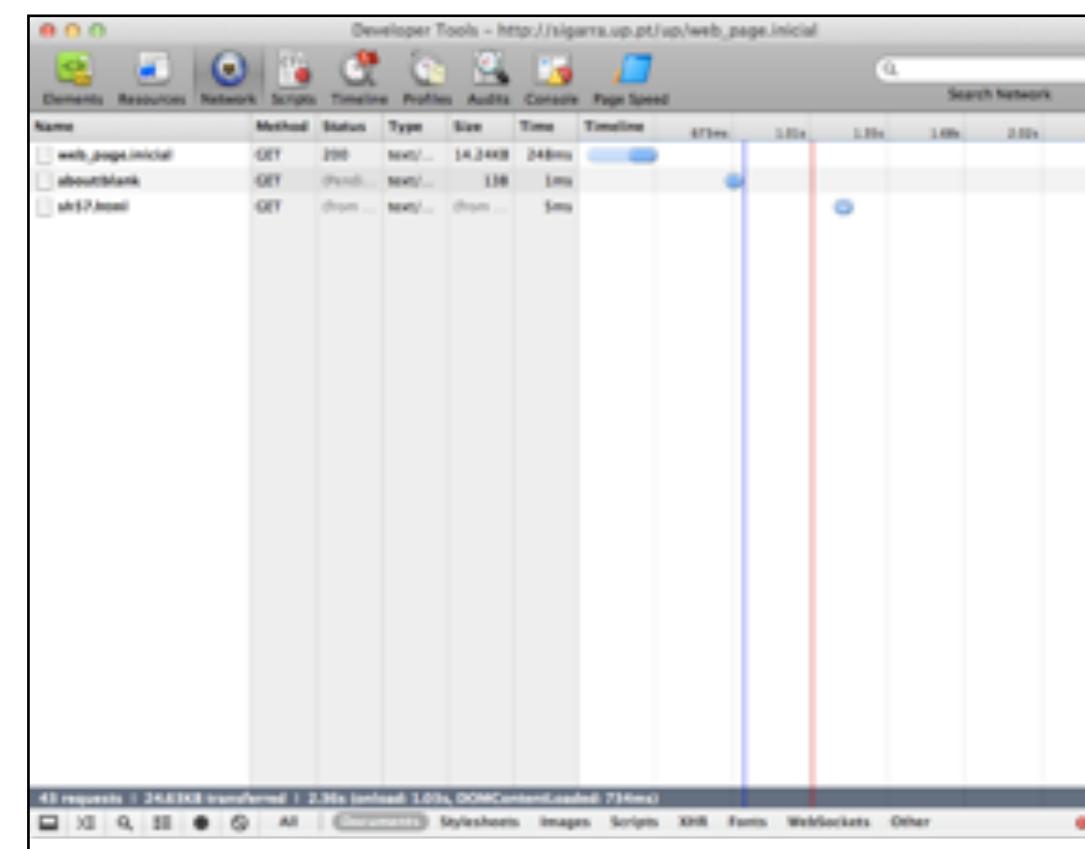


Client Server Interaction

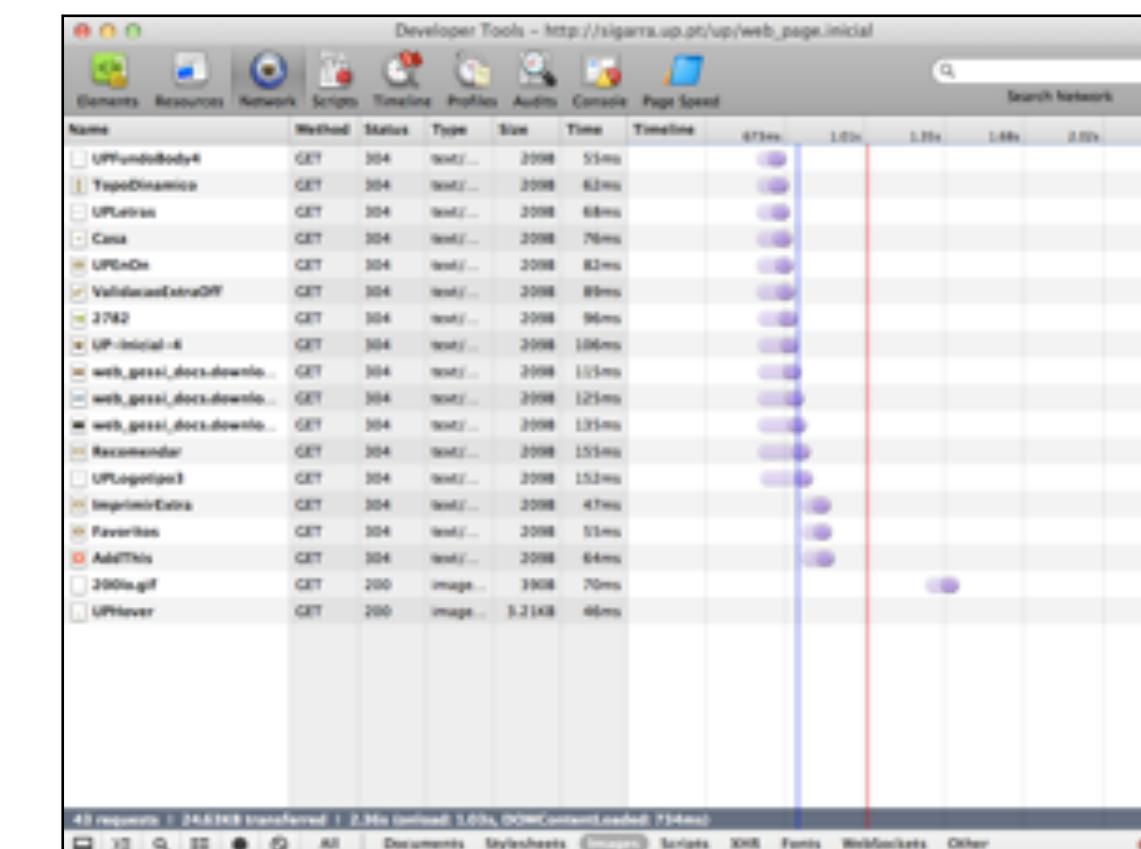


Requesting a Web Page

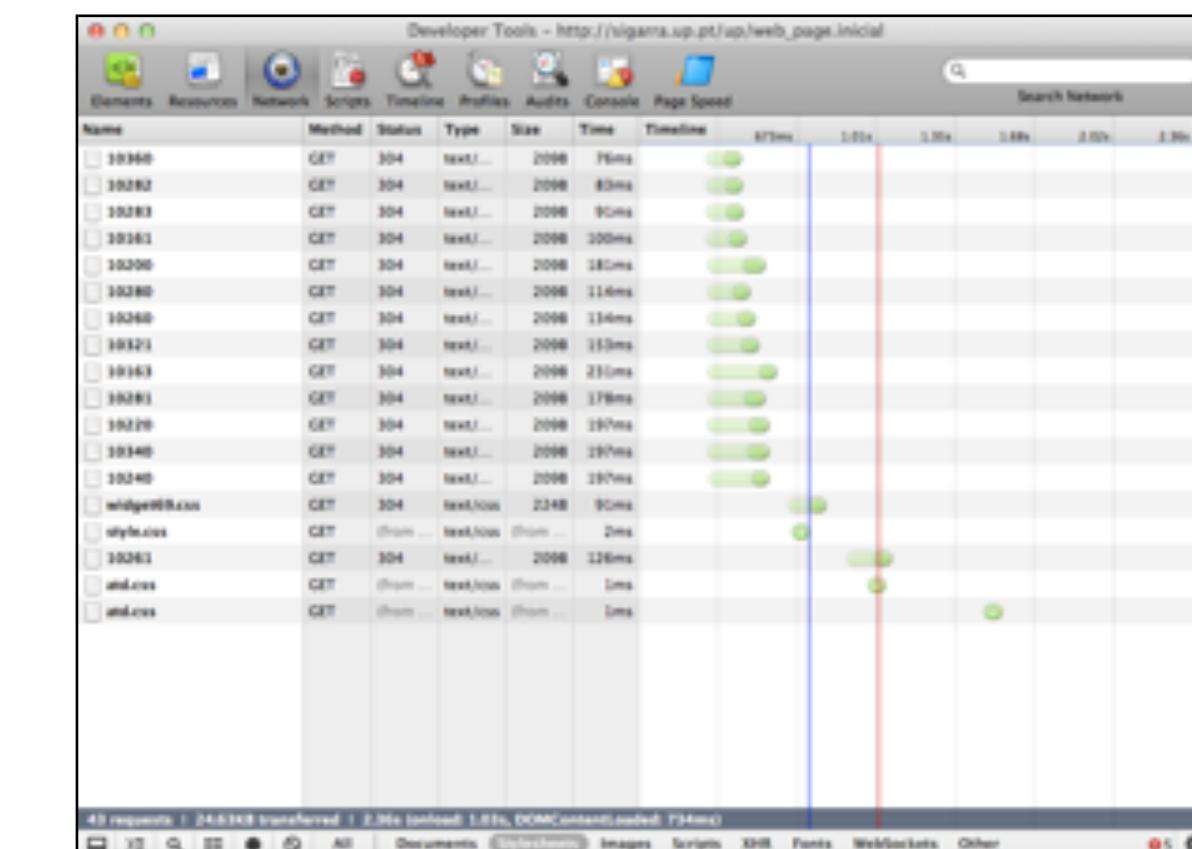
- Web pages combine text, images, and other resources.
- Web browsers issue multiple requests when preparing a single web page, one request for each individual resource (e.g. HTML document, CSS files, images, JavaScript files, etc).



Base documents requested.



Images requested.



CSS files requested.

Web Clients

- World Wide Web client applications, commonly known as web browsers, are software applications capable of retrieving, presenting and transversing information resources available on the World Wide Web.
- Web browsers communicate with web servers using the HTTP protocol.
- Web browsers are increasingly sophisticated.
- The first web browser was called WorldWideWeb and was bundled with the releases of the WorldWideWeb system.
- Mosaic, developed at NCSA, was the first popular browser. It was the first to integrate text and images in a single page.

WorldWideWeb (later Nexus) Browser



WorldWideWeb (1990), developed by Tim Berners-Lee, was the first web browser.

Mosaic

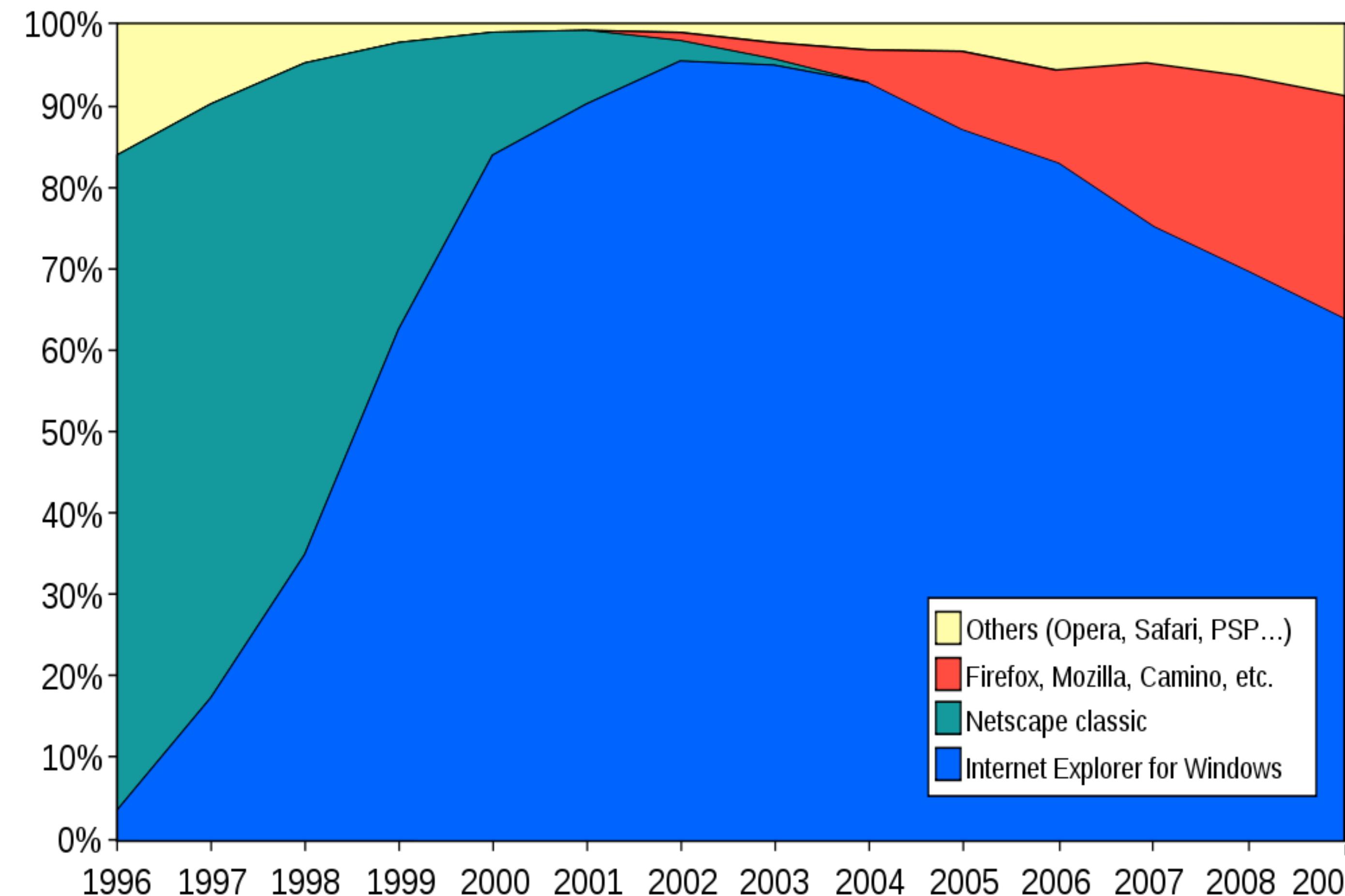


Mosaic was the first mainstream web browser (1993).

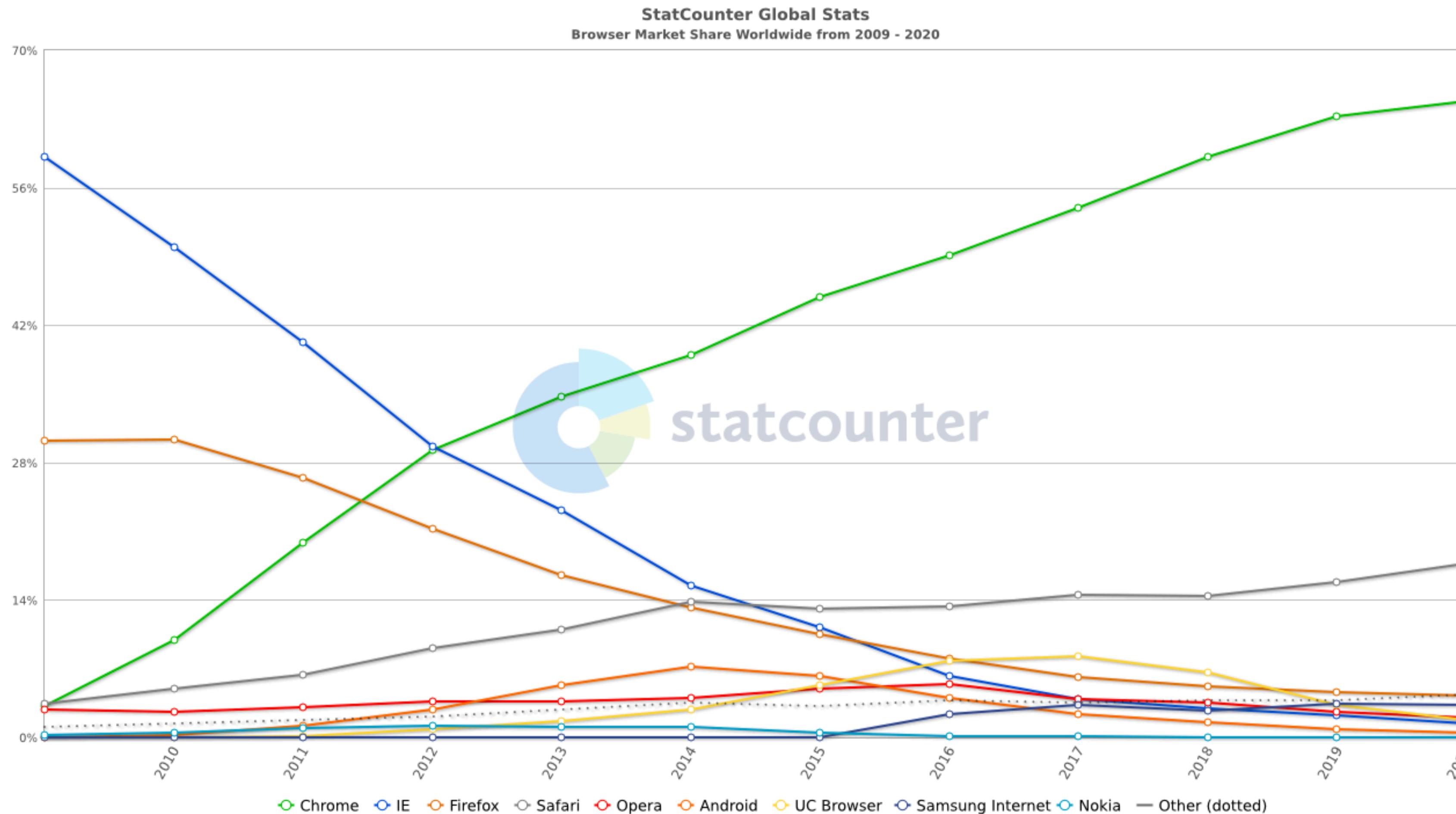
Mosaic

- Marc Andreessen and Jim Clark left NCSA to start Netscape Communications in 1994. Later that year, Netscape version 1 was released.
- NCSA licensed Mosaic technology to Microsoft to form the basis of Internet Explorer. Version 1 was released in 1995.
- The “Browser Wars” started for the dominance of the web browser market.
- Internet Explorer had a major advantage — it was bundled with every copy of Windows. This latter led to the USA vs. Microsoft case on monopoly abuse.

Browser Wars

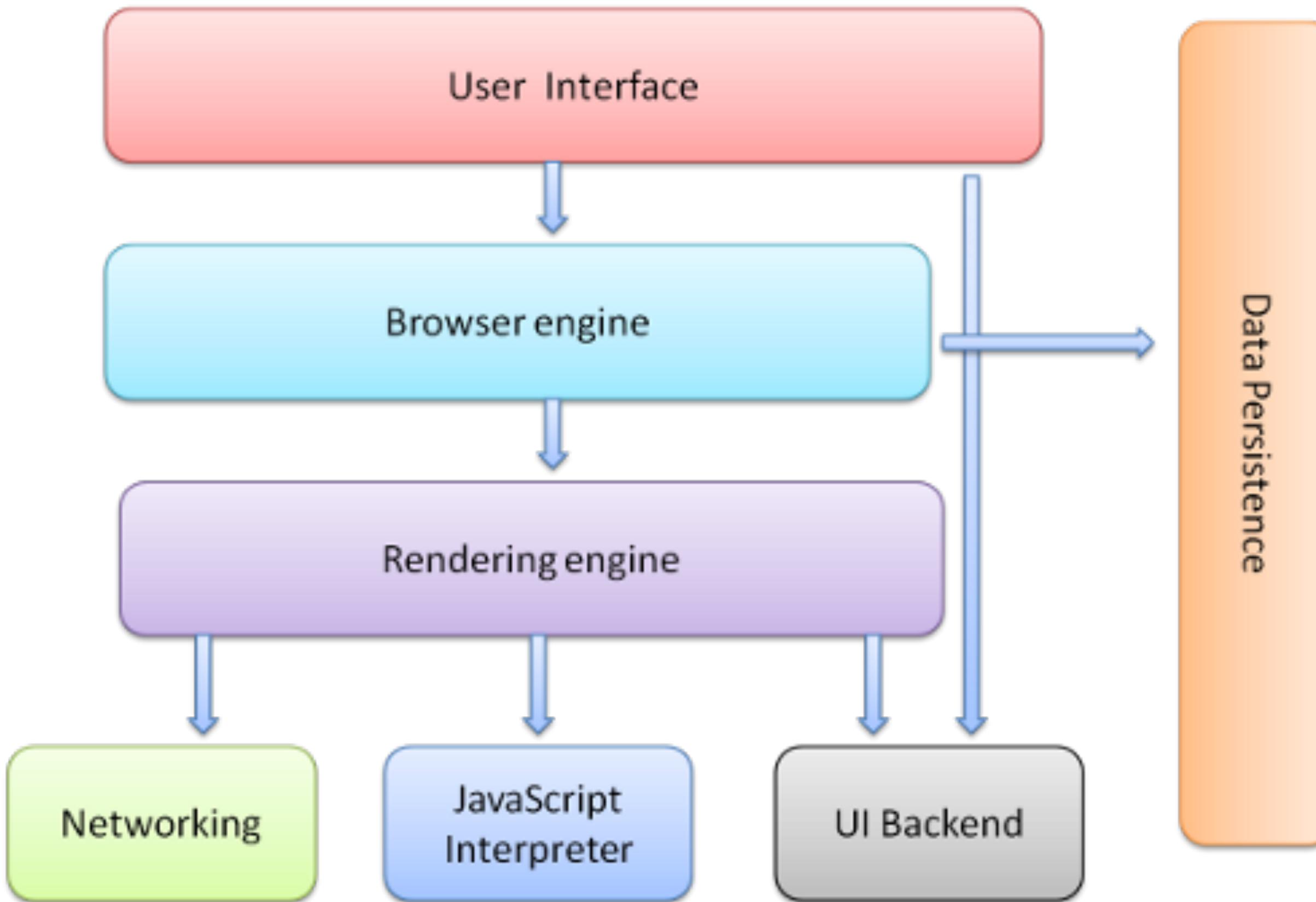


Source: Wikipedia - “Browser Wars”



Source: Wikipedia - “Usage share of web browsers”

Browser's High Level Structure



- User Interface - browser controls
- Browser engine - mapping
- Rendering engine - HTML & CSS
- Networking - network calls
- JS Interpreter - execute javascript
- UI Backend - drawing widgets
- Data Persistence - saves data

Notable Layout Engines

- **Trident** — Developed by Microsoft for use in Internet Explorer.
- **Gecko** — Developed by the Mozilla Foundation, used in Firefox, Camino.
- **WebKit** — A fork of KHTML developed by Apple, used in Safari.
- **Blink** — A fork of WebKit developed by Google, used in Chrome, Opera and Edge.
- **Presto** — Developed by Opera Software, used in Opera (until 2013).
- **EdgeHTML** — New Microsoft rendering engine launched in 2015. RIP December, 2018.

“Comparison of Layout Engines”, Wikipedia

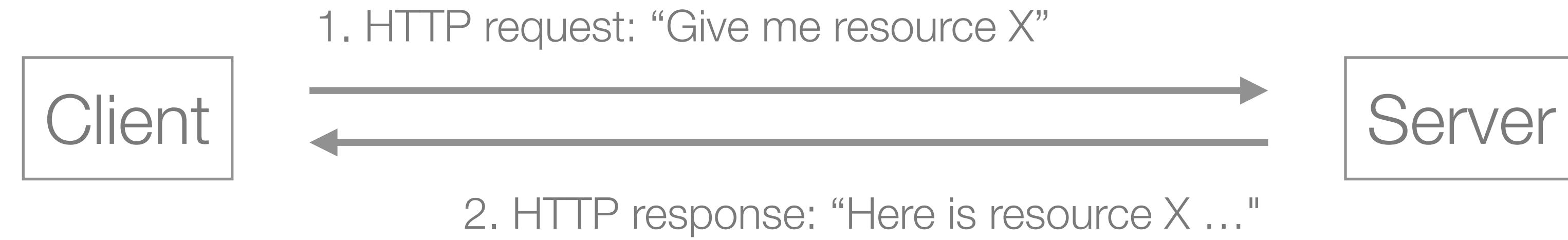
https://en.wikipedia.org/wiki/Comparison_of_layout_engines

“Timeline of web browsers”, Wikipedia

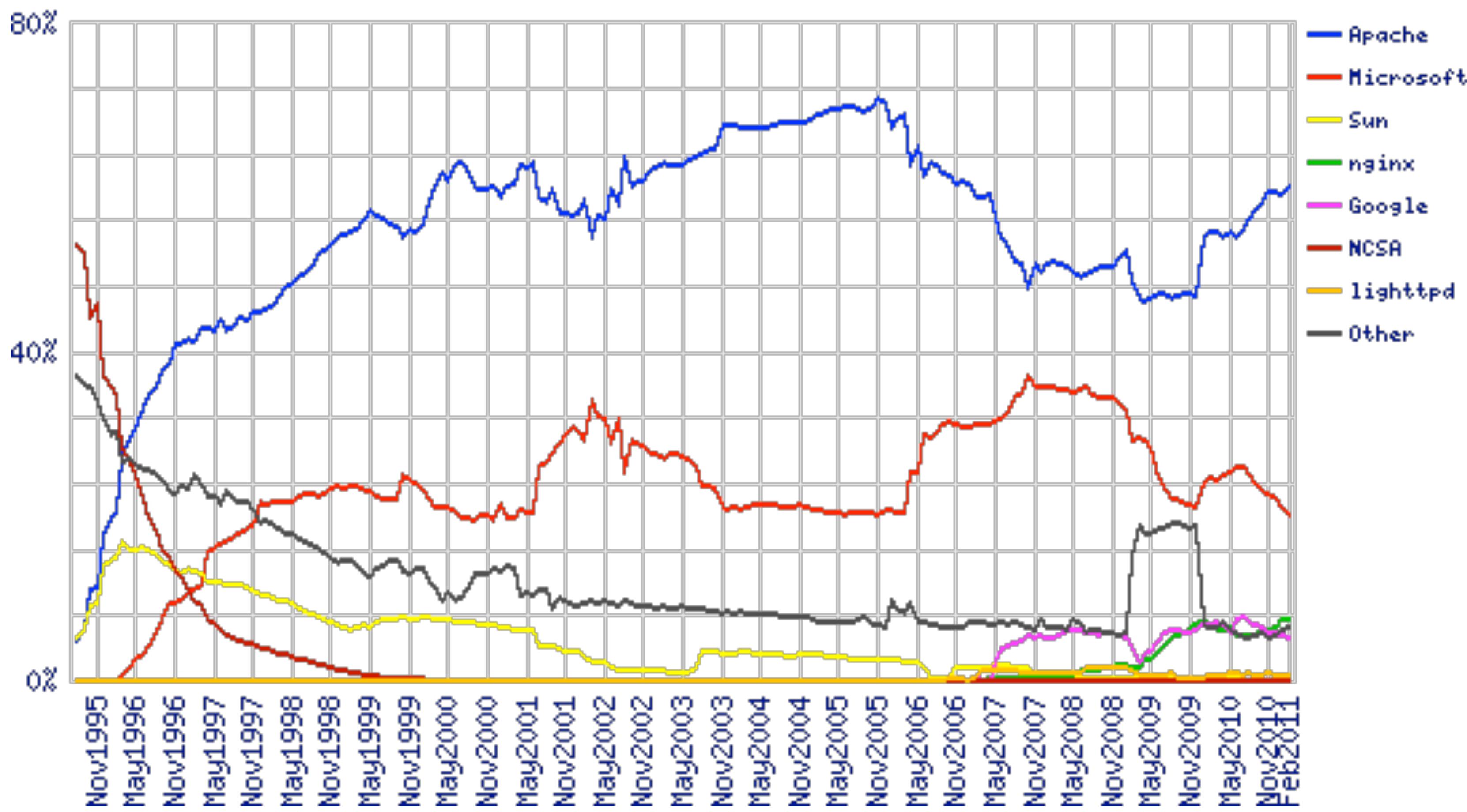
https://en.wikipedia.org/wiki/Timeline_of_web_browsers

Web Servers

- A web server is a program whose primary function is to deliver resources on clients' requests.
Only acts when requests arrive.
- Web servers handle multiple web clients simultaneously.
Servers and clients communicate using the HTTP protocol.

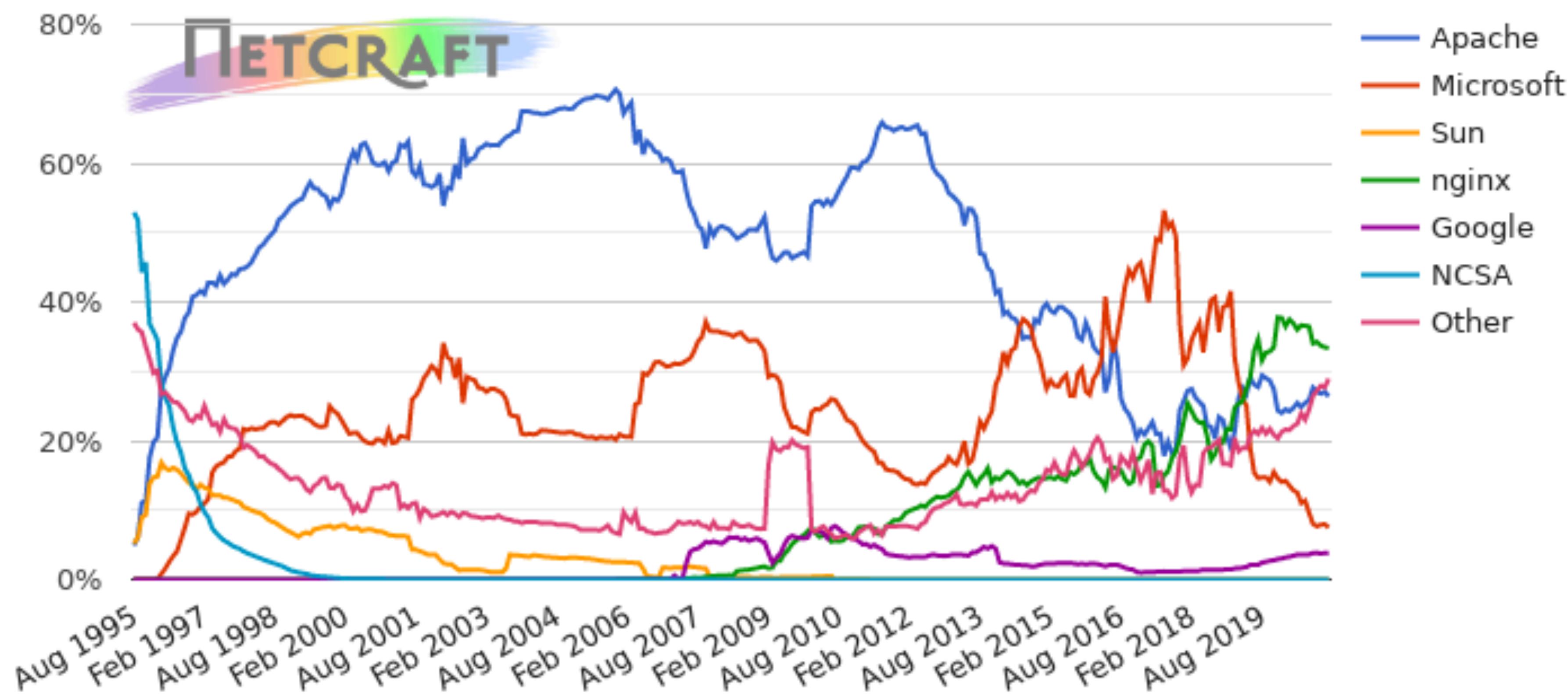


- The most common web servers are the Apache HTTP Server and Microsoft's Internet Information Server (IIS). Lightweight alternatives: nginx, lighttpd.
- Typically, different web servers coexist in a production environment.



Source: Netcraft

Web server developers: Market share of all sites



Source: Netcraft

WWW's Core Technologies

- The Web is supported by three core technologies:
 - **URL: Uniform Resource Locator**
Used to identify the resources available on the web.
 - **HTTP: HyperText Transfer Protocol**
Used to specify how clients communicate with servers.
 - **HTML: HyperText Markup Language**
Used to represent and interlink documents on the web.

URL: Uniform Resource Locator

- A URL establishes a unique address for a World Wide Web resource, e.g. pages, images, etc. URLs are used to locate web resources.
- Syntax (simplified): protocol://machine:port/directory/file.type
 - Protocol, e.g. http://, ftp://, file://
 - Machine, e.g. www.up.pt or 193.137.55.13
 - Port, e.g. 80 (the default), 1000, 20
 - Resource path, the directory path to the file
- Example: http://www.up.pt:80/sobre/index.html

HTTP: HyperText Transfer Protocol

- The HTTP protocol defines how web client communicate with web servers to access web resources.
- HTTP was developed as a joint work of IETF and W3C.
- It is a request-response protocol, i.e. client issues a request and waits for the server to respond. Timeouts can occur if servers take too long.
- HTTP is a stateless protocol, i.e. each request is treated as an independent transaction. This results in a simpler design, but requires additional information to be send in each request.

HTTP: Request Methods

- HTTP supports several request commands, called HTTP methods. A total of nine methods are defined in the HTTP standard. GET and POST are the most commonly used by web browsers.
- GET – Requests the resource from the server. Idempotent operation.
- HEAD – Requests only the headers (without the content).
- POST – Submits data to be processed to the identified resource.
- PUT – Uploads data into the specified resource.
- DELETE – Deletes the specified resource.

HTTP: Status Codes

- All HTTP responses include a numeric status code, indicating if the request succeeded or if other actions are required. Codes are organized in five classes of responses.
 - 1xx — Informational
 - 2xx — Success (e.g. 200 OK, 201 Created)
 - 3xx — Redirection (e.g. 301 Moved Permanently)
 - 4xx — Client Error (e.g. 404 Not Found, 403 Forbidden)
 - 5xx — Server Error (e.g. 500 Internal Server Error)

HTTP is Stateless

- Web servers do not keep any information about clients.
Each request is isolated from all others.
- **State must be maintained by web applications.**
- How can we implement a stateful user experience over a stateless protocol (e.g. shopping cart, authenticated access)? Two options:
 - Cookies – client-side pieces of data generated by the server and attached to each HTTP request.
 - Sessions – server-side files with unique identifiers (session IDs), these can be passed in URLs or Cookies.

HyperText Markup Language – HTML

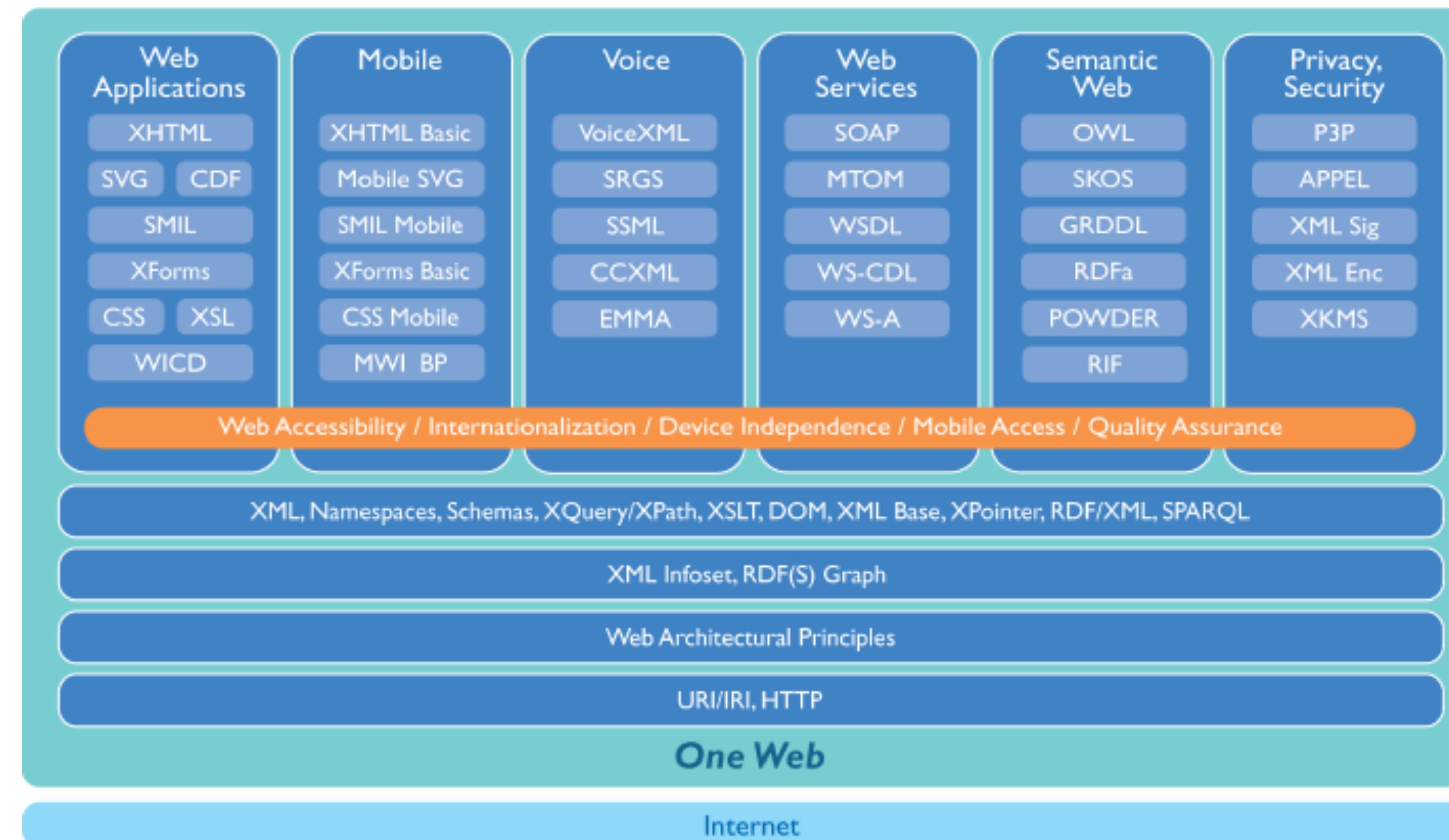
- The HyperText Markup Language (HTML) is used to define the content and structure of hypertext documents.
- First standard was published in 1995 – HTML 2.0.
- HTML 4.01 was published in 1999.
- XHTML was a reformulation of HTML as XML. W3C tried to “force” authors to write well-formed code. Later abandoned due to low adoption by web developers.
- HTML5 is the latest major revision to HTML.

W3C

- The World Wide Web Consortium was founded in 1994 by Tim Berners-Lee to standardize the protocols and technologies used to build the web.
- The W3C is an international standards organization, composed by member organizations and full time staff, that develops technical specifications and guidelines for the web.
- Mission: “Lead the Web to its full potential”.
- W3C Process: (1) members propose new technologies or ideas; (2) working groups are formed; (3) recommendations are developed and approved by consensus.
- The W3C does not enforce their recommendations.

W3C Standards

W3C standards are many and in different areas, from technical specifications to guidelines.



W3C technology stack (circa 2004).
<http://www.w3.org/Consortium/techstack-desc.html>

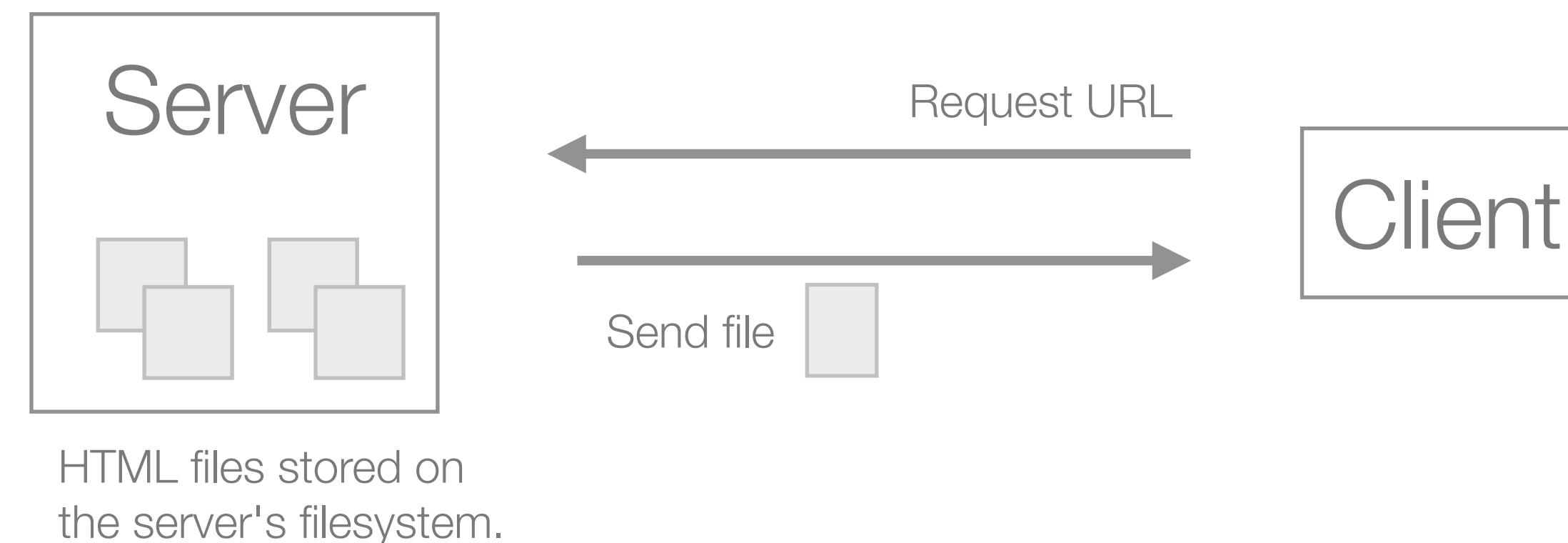
W3C Process

- People generate interest in a particular topic (e.g., web services). For instance, Members express interest in the form of Member Submissions, and the Team monitors work inside and outside of W3C for signs of interest.
- When there is enough interest in a topic, the Director announces the development of a proposal for a new Activity or Working Group charter, depending on the breadth of the topic of interest.
- There are three types of Working Group participants: Member representatives, Invited Experts, and Team representatives. Team representatives both contribute to the technical work and help ensure the group's proper integration with the rest of W3C.
- Working Groups generally create specifications and guidelines that undergo cycles of revision and review as they advance to W3C Recommendation status.

Web Applications

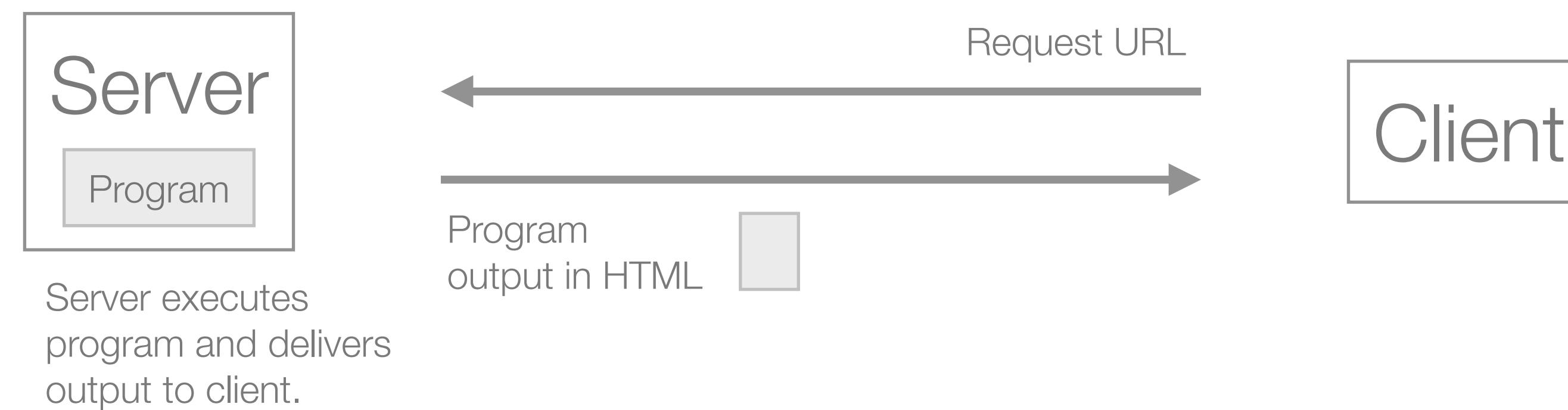
Static Web Pages (early 1990s)

- In the early days of the web, most pages were static files served directly from the filesystem. **Pages are constructed at design time.**
- Developing web pages involved few technologies, mostly just HTML. Learning by example, using the “view source” option, was an important method for knowledge dissemination.



Dynamic Web Pages (mid 1990s)

- Dynamic web pages emerged in the mid 1990s. Instead of serving static files from the filesystem, software applications produce web pages when requested.
Pages are constructed at run time, when “called” by the browser.
- The Common Gateway Interface (CGI) is a specification that defines how web requests and responses interact with an application program.
- There are several alternatives: Apache modules, IIS plug-ins, FastCGI, WSGI.



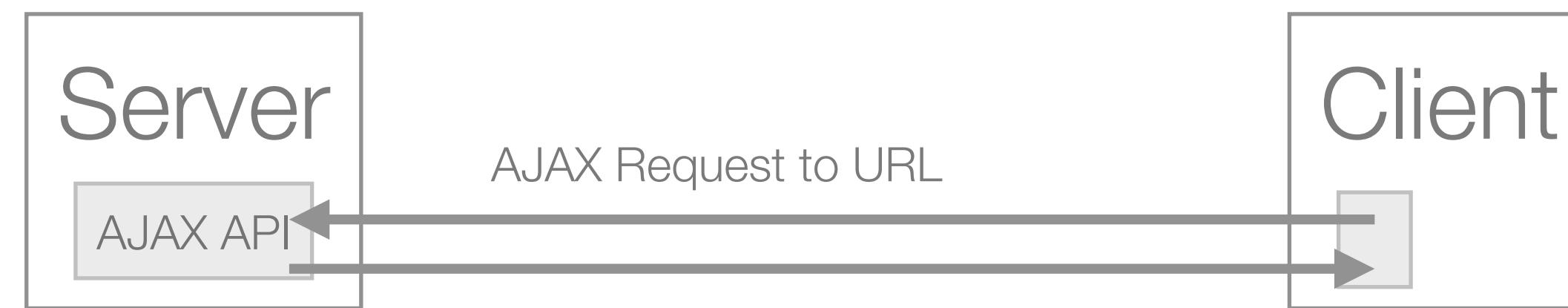
Dynamic Web Sites (1990s ...)

- Sites emerged as a collection of multiple and coherent web pages. Instead of treating each page independently, multiple web pages were handled by shared functions and libraries.
- Provide a stateful experience to the user (e.g. shopping cart).
- Typically a common data layer was implemented across pages.
- Libraries and frameworks to address repetitive and common tasks.
- Richer user interfaces (e.g. JavaScript).

Web Applications (mid 2000s ...)

- Strong developments in client-side technologies and methodologies led to richer and interactive user interfaces.
- AJAX enabled the creation of asynchronous web applications. No need to reload full pages on each user interaction.
- Web documents became applications itself, i.e. code runs on the document.
- Rise of new web frameworks – Ruby on Rails, Django, etc.
- Wide adoption of HTML5.
- A new interaction paradigm with mobile devices.

AJAX

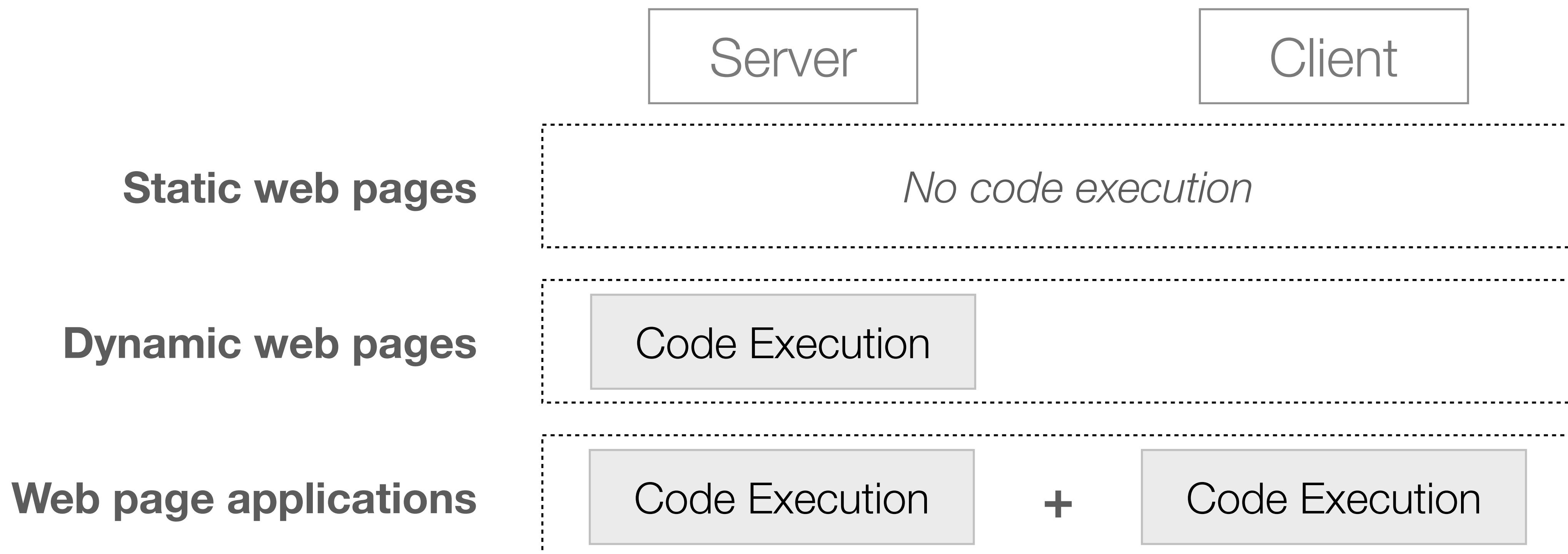


Client

Requests are made to the server, **from the document using JavaScript**, without loading a new document.

The HMTL document is dynamically altered depending on the answer and using JavaScript.

Code Execution



The Three-Tier Architecture

- Web applications are typically structured in three tiers, corresponding to three core aspects of a web system: presentation, business logic and data access.
- Main advantages of this approach: separation of concerns, maintainability.

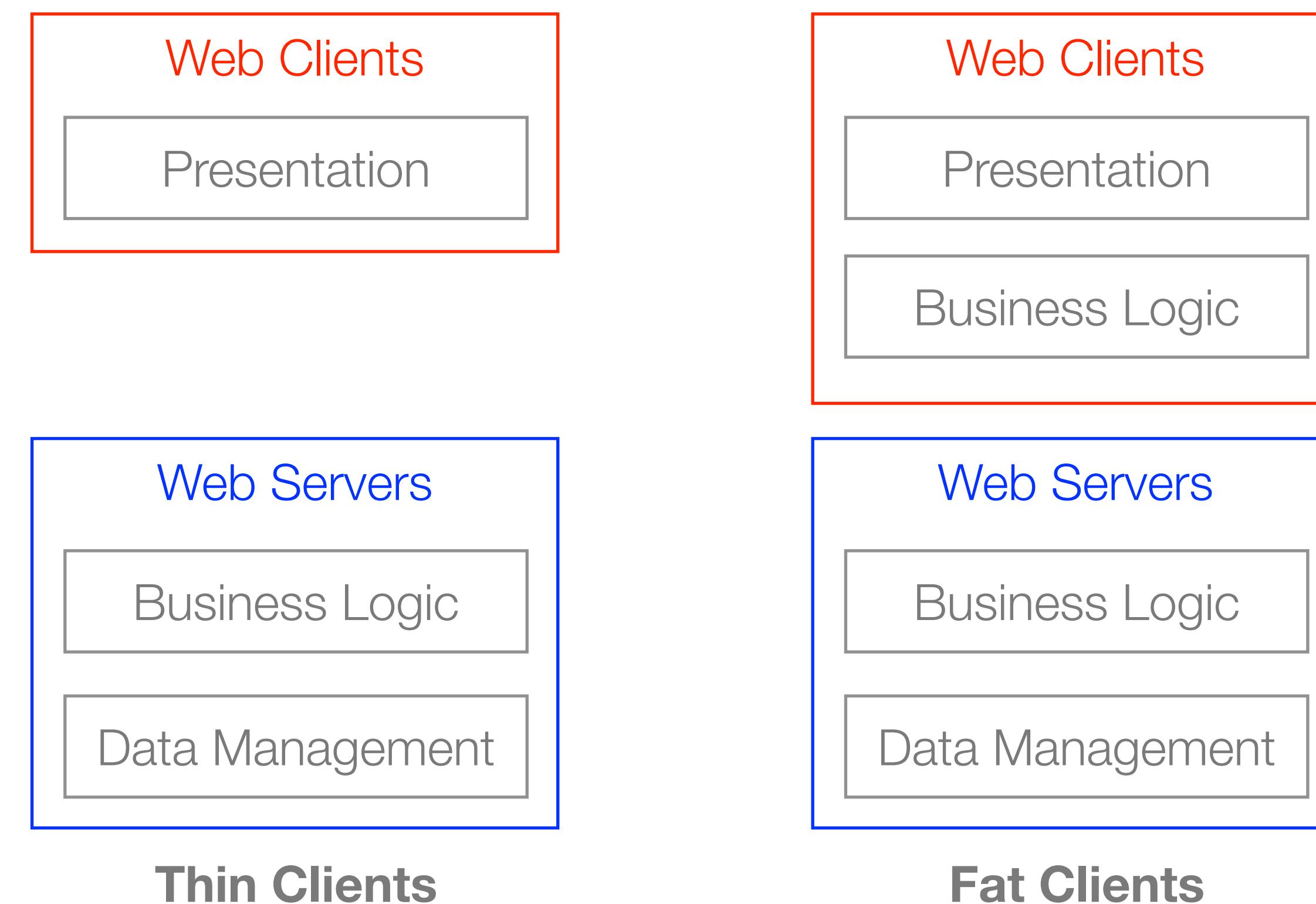
Presentation

Business Logic

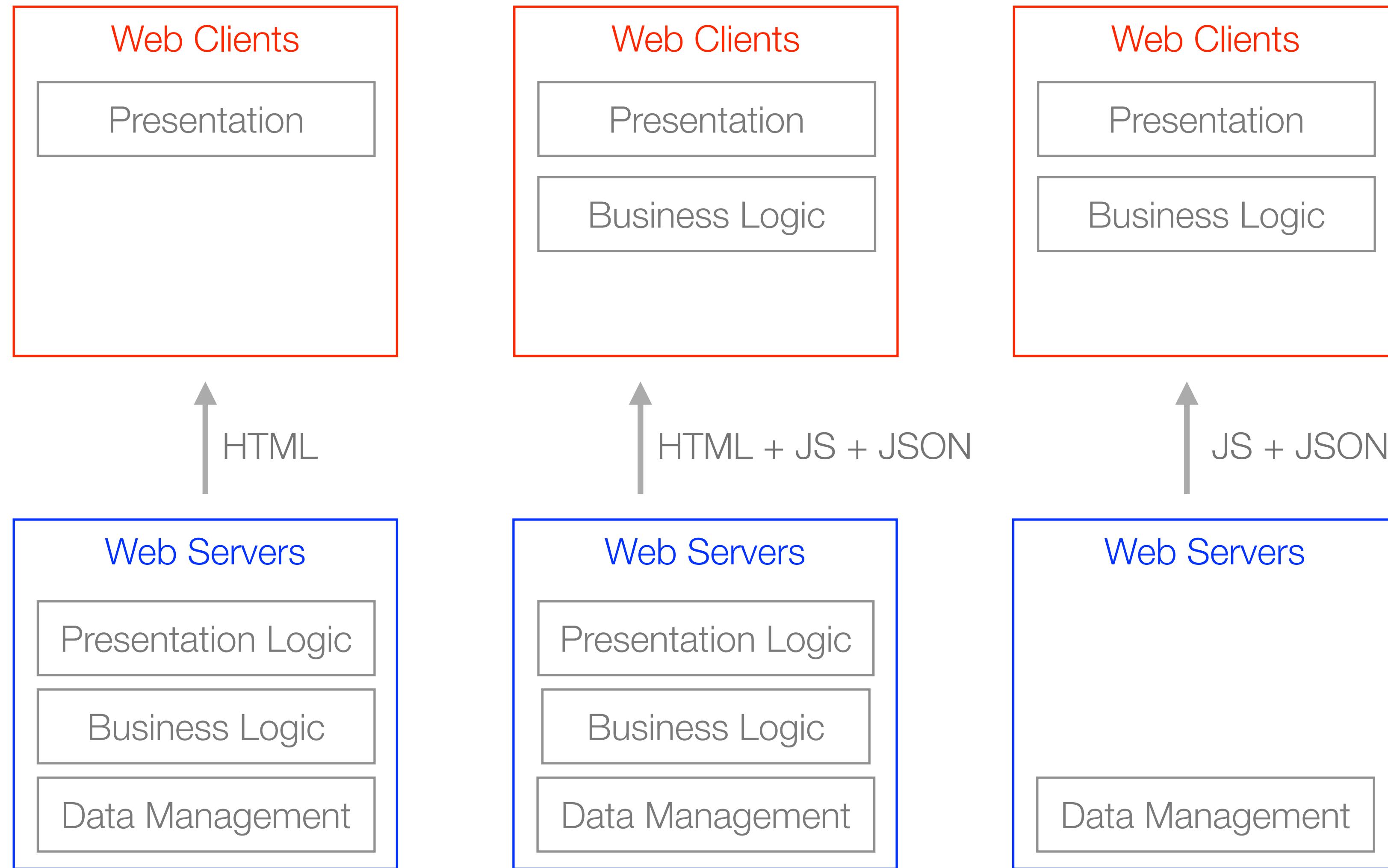
Data Management

Architecture of Web Applications

- Over time, the architecture of web applications has changed significantly. In the beginning, web clients were only used to present the user interface. Recently, application logic has moved partially to the clients.

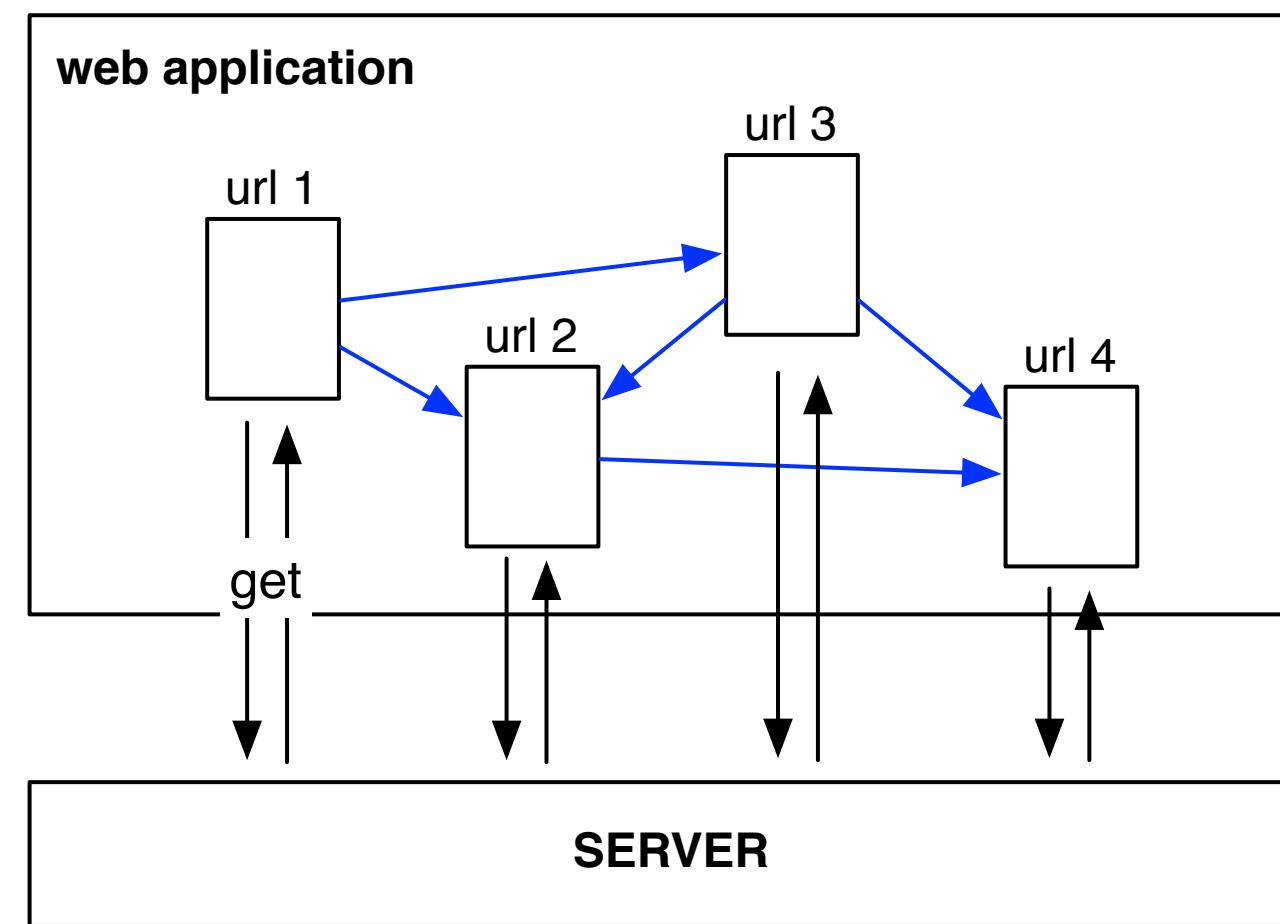


Different Architectural Options



Multi-Page Web Applications

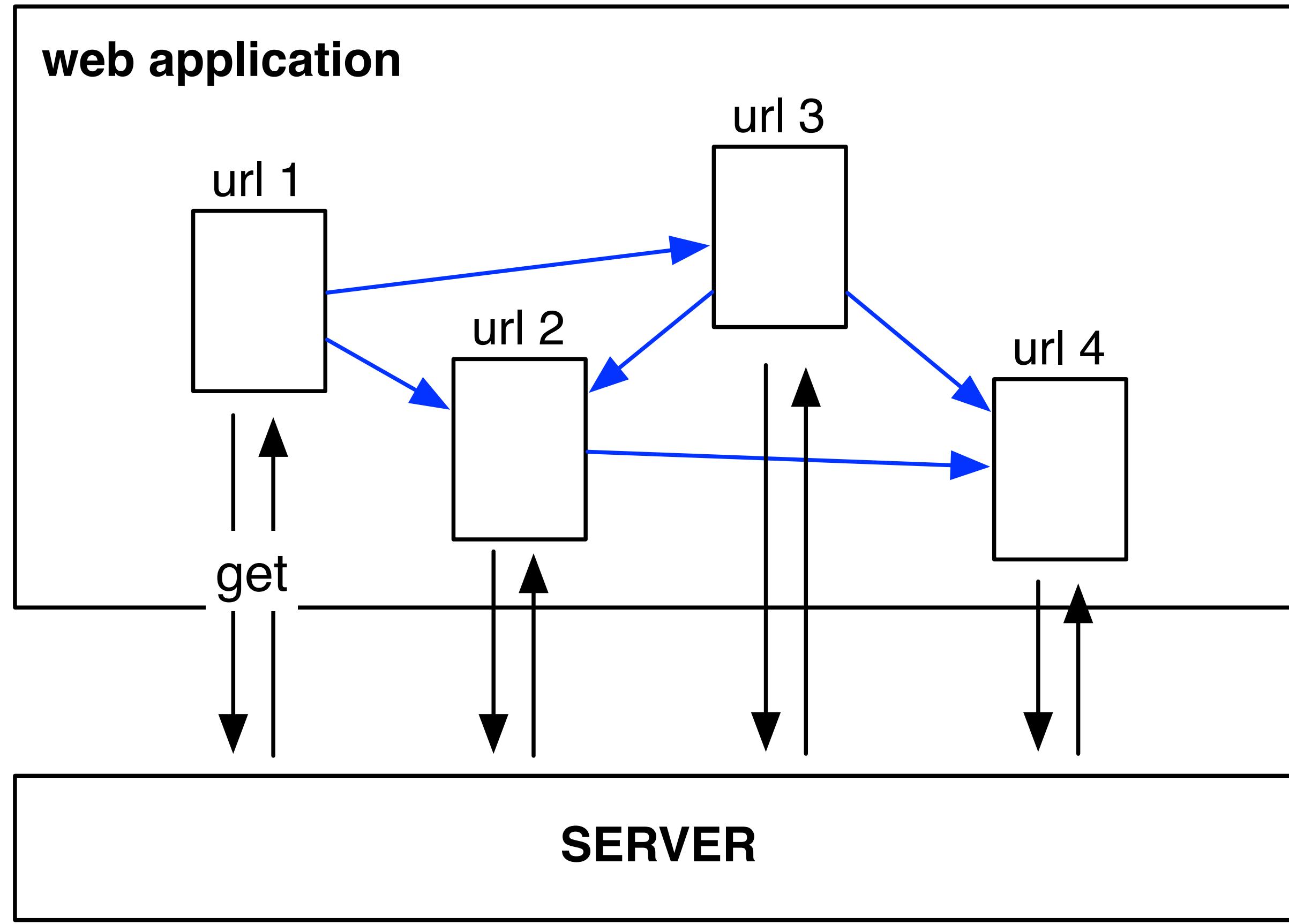
- The web application is implemented as a collection of multiple web pages. The user interacts with the application navigating through these pages. Each page is prepared on the server and only presentation details are sent to the browser.



Each page corresponds to a HTTP GET request. Application logic is all maintained on the server.

E.g. Amazon, SIGARRA

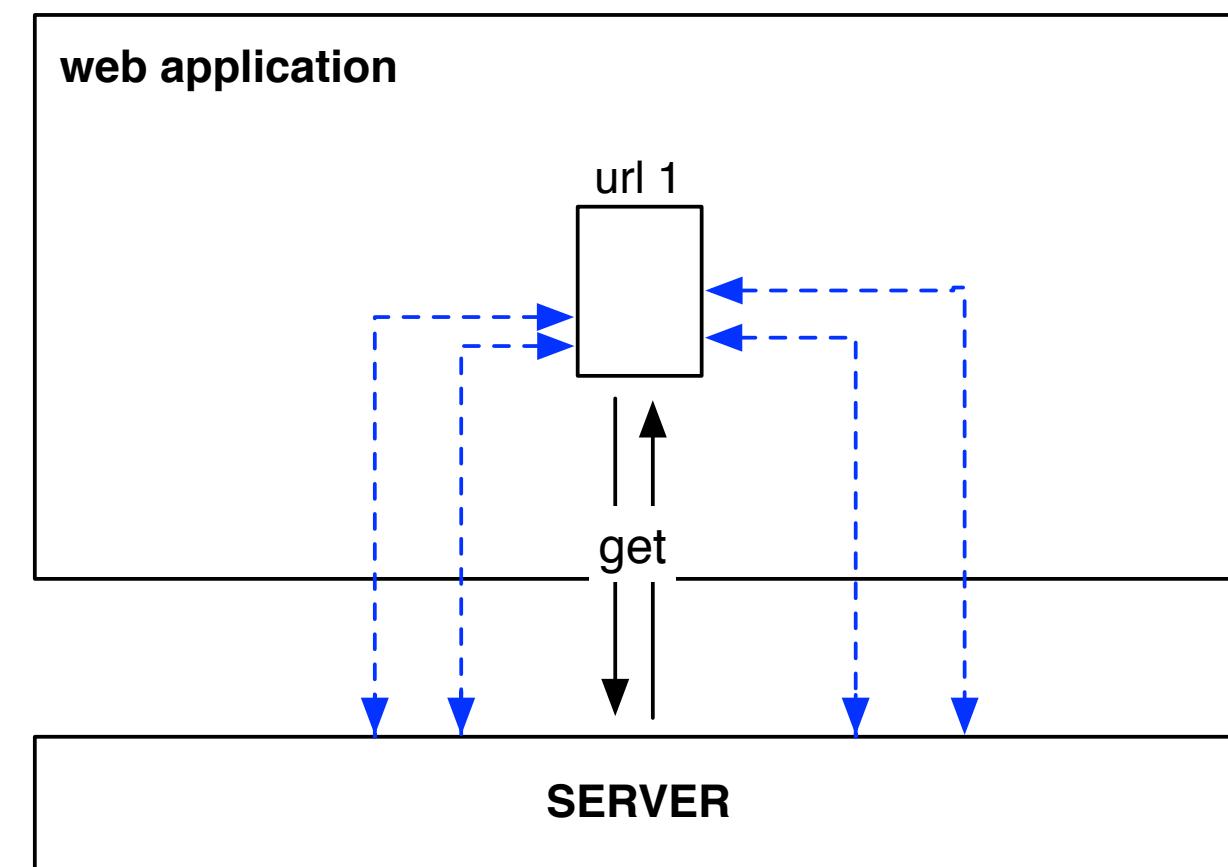
- Advantages: REST style, client independent, consistency across browsers, broad technological ecosystem, application logic is kept on the server.
- Disadvantages: slow performance and responsiveness, fragmented code, no way to deliver updates to an open web page.



Each page corresponds to a HTTP
GET request. Application logic is
maintained on the server.

Single-Page Web Applications

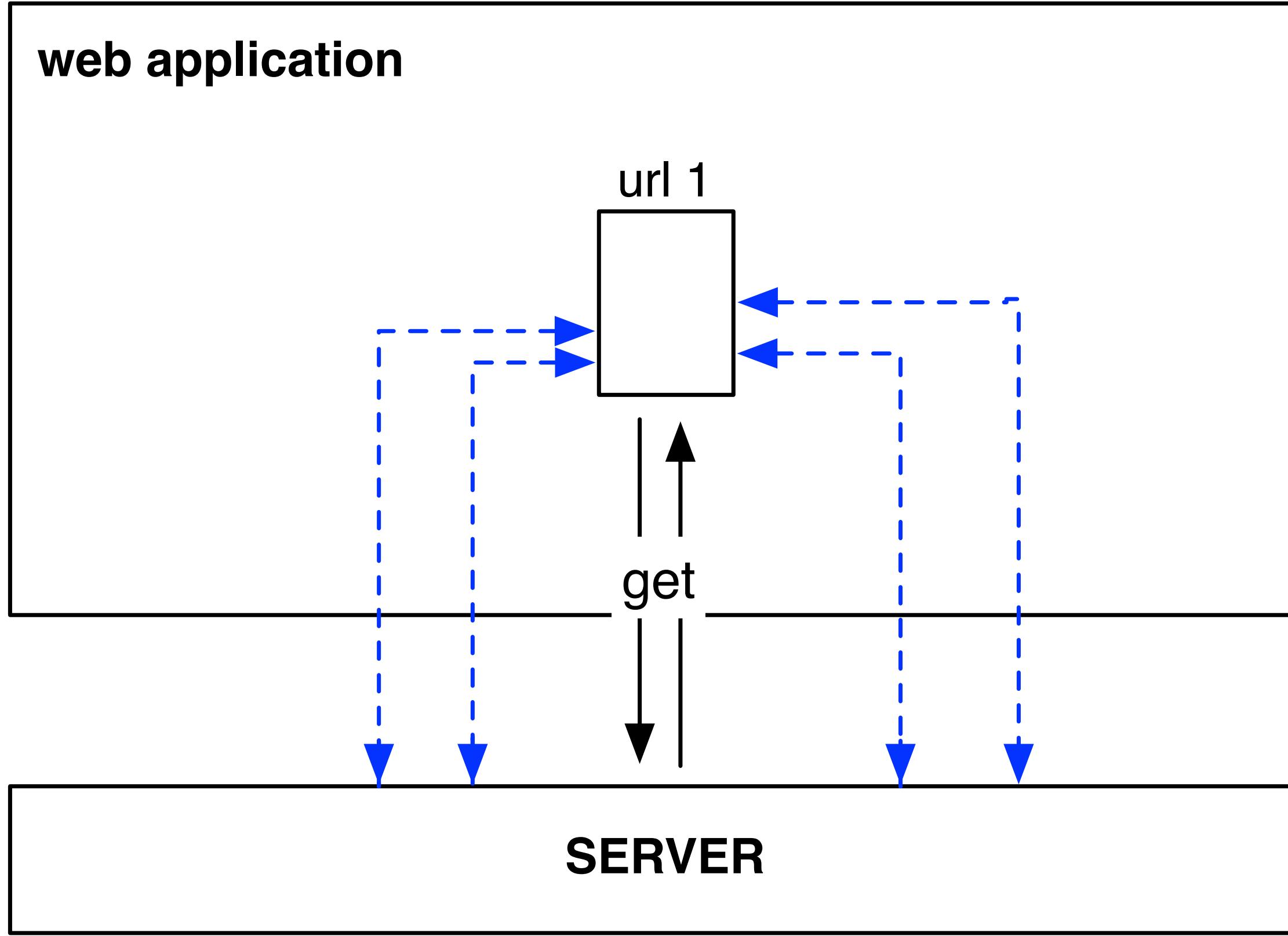
- The web application is implemented as a single web page. All necessary resources are loaded or dynamically added to the page (e.g. using Ajax). Application logic is pushed to the client — fat client architecture.



An initial page is loaded using HTTP GET. All other resources are loaded dynamically following user interactions.

E.g. Slack

- Advantages: improved user experience, reduced bandwidth consumption, decoupled client and server, reusable server interfaces, reusable client code.
- Disadvantages: JavaScript required, breaks browser history, increased browser dependency (versions, features, performance), no REST, difficult to crawl and index.

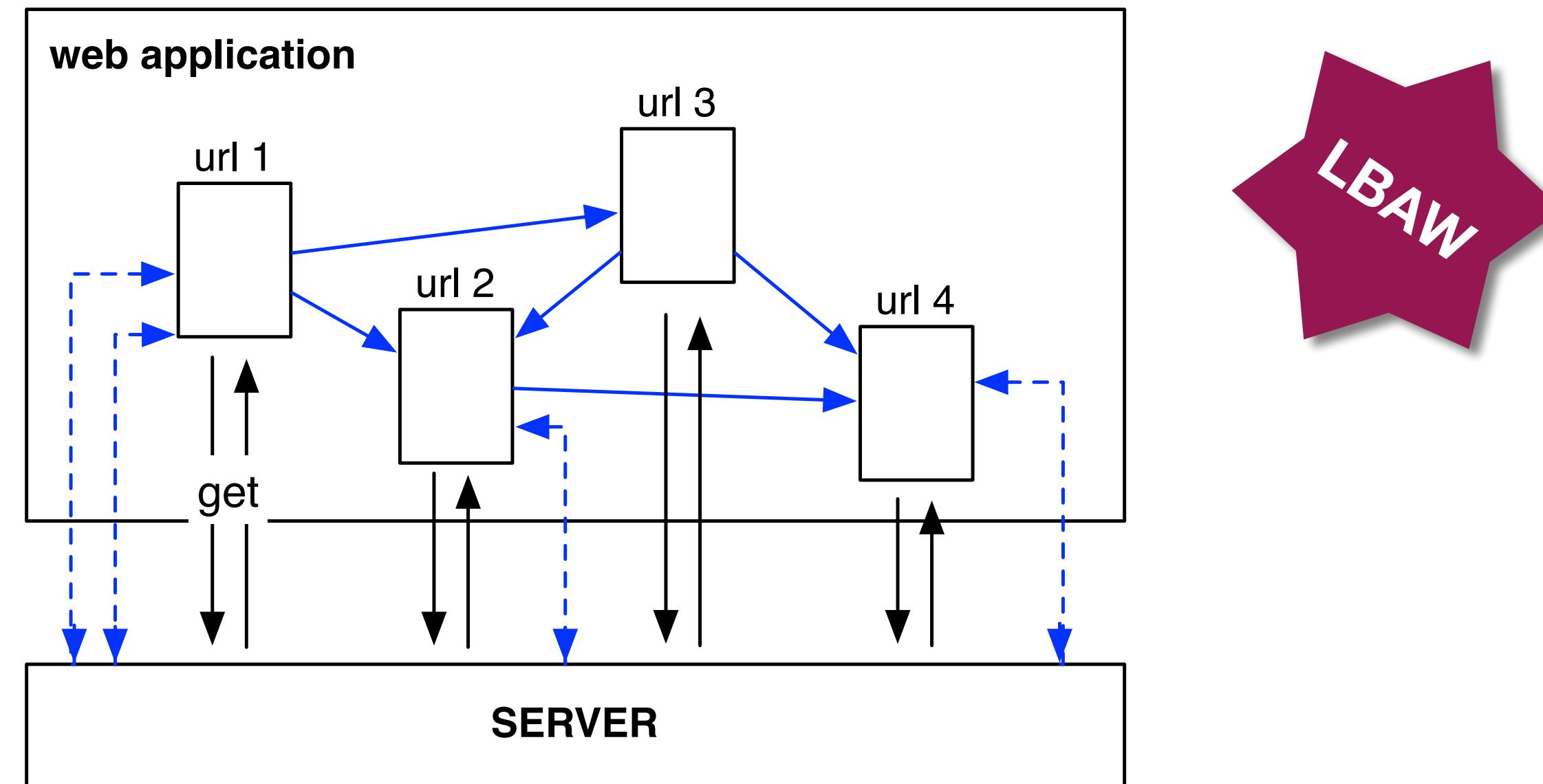


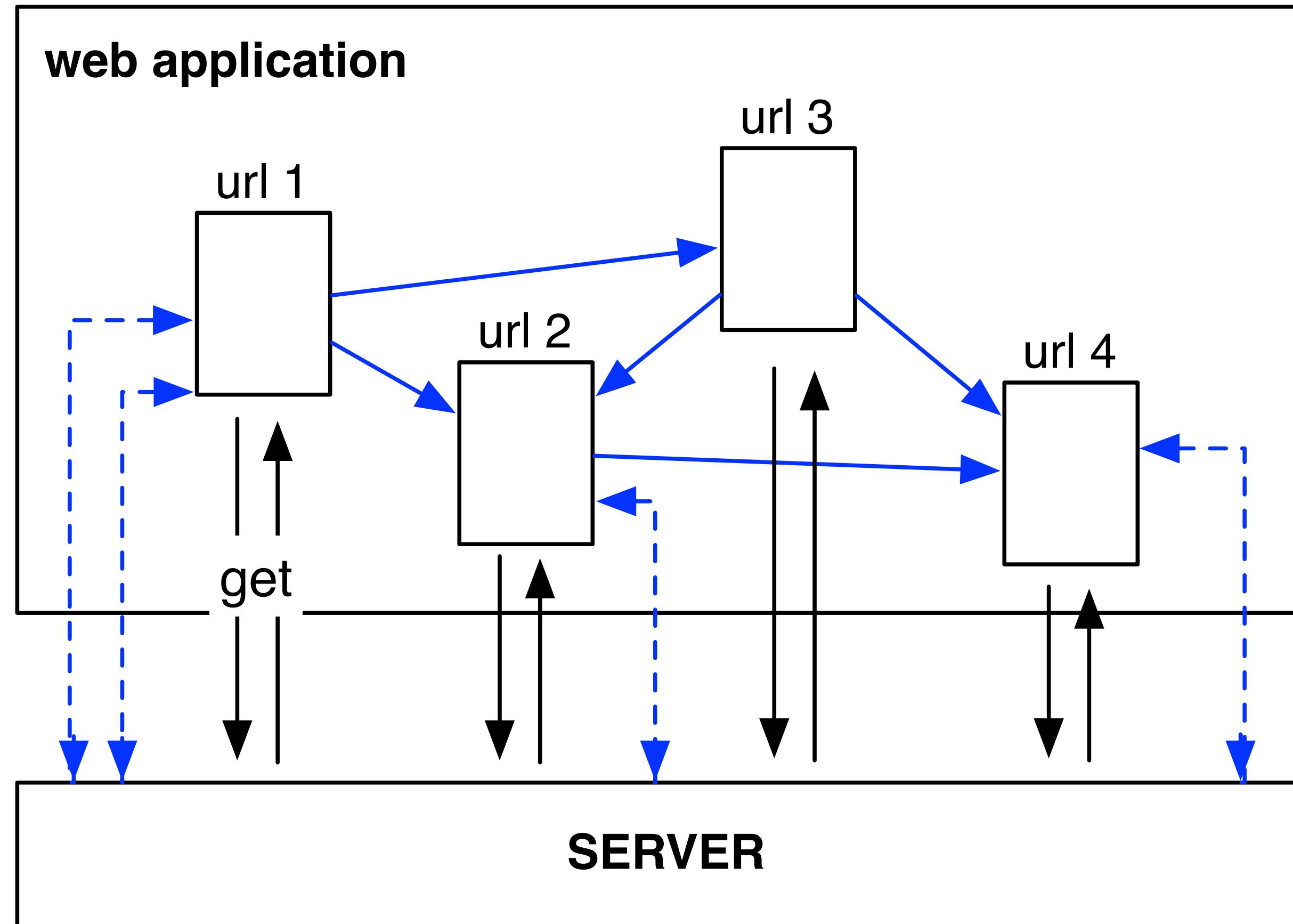
An initial page is loaded using HTTP GET, including the initial HTML document plus JavaScript code.

All other resources are loaded dynamically following user interactions.

Mixed Approach

- The two architectural styles can be mixed to combine the benefits of both.
- Use different web pages to setup the core structure of the application.
- Define data APIs to provide updates to the web pages and support different devices.
- Use asynchronous requests to improve performance and user experience.





Main application resources have a URL.
User interaction is improved by using
asynchronous calls to data APIs.

Web Architecture Summary

- Criteria to consider in choosing an architecture:
 - **Usability**
User friendly? Instant updates possible? Browser history?
 - **Search / Share**
Search engine friendly?
 - **Linking**
Mapping between URLs and views? 1-to-1?
 - **Performance**
Consistent across platforms? Optimized content loading? Client effort?
 - **Productivity**
Developers background? Modularity?
 - **Testing**
Modularity? What to test? Client dependency?
- **Choosing an Architecture == Choosing the Challenges**

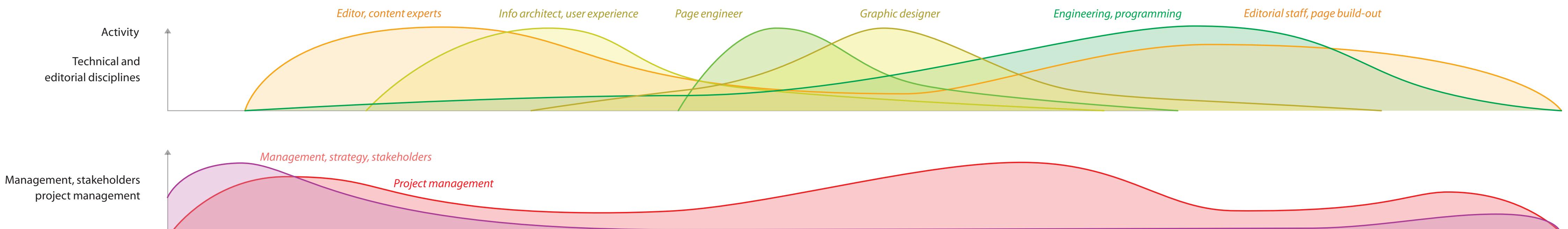
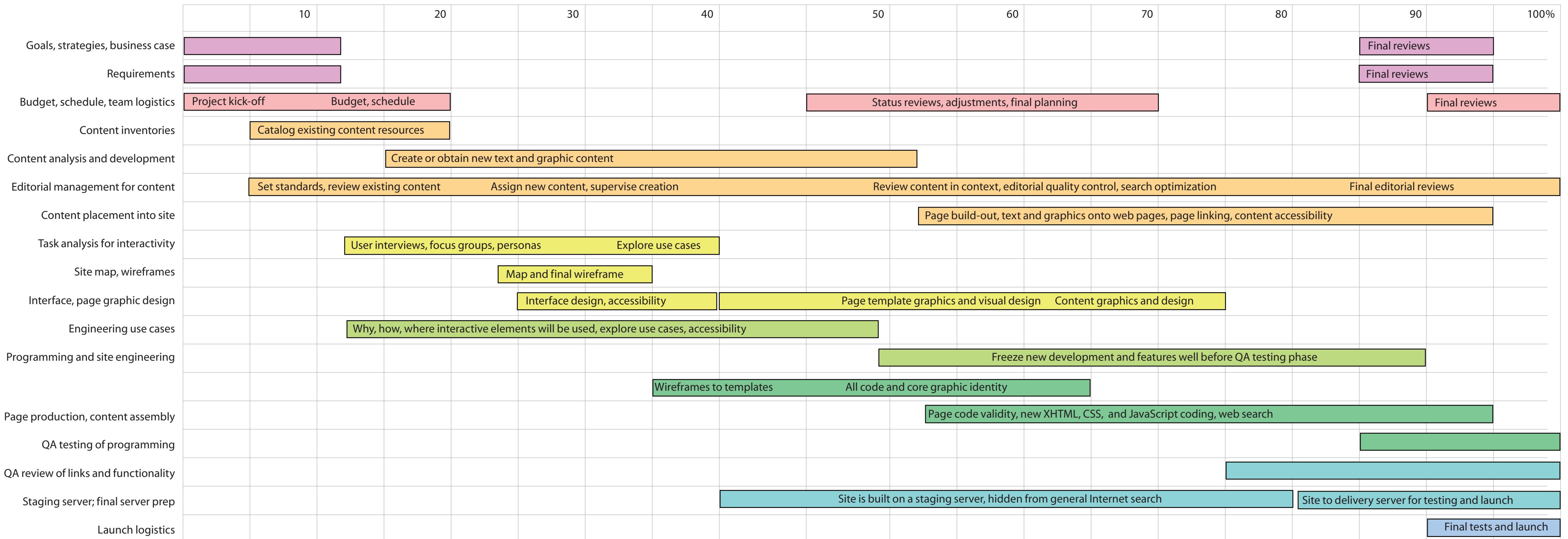
Rendering on the Web

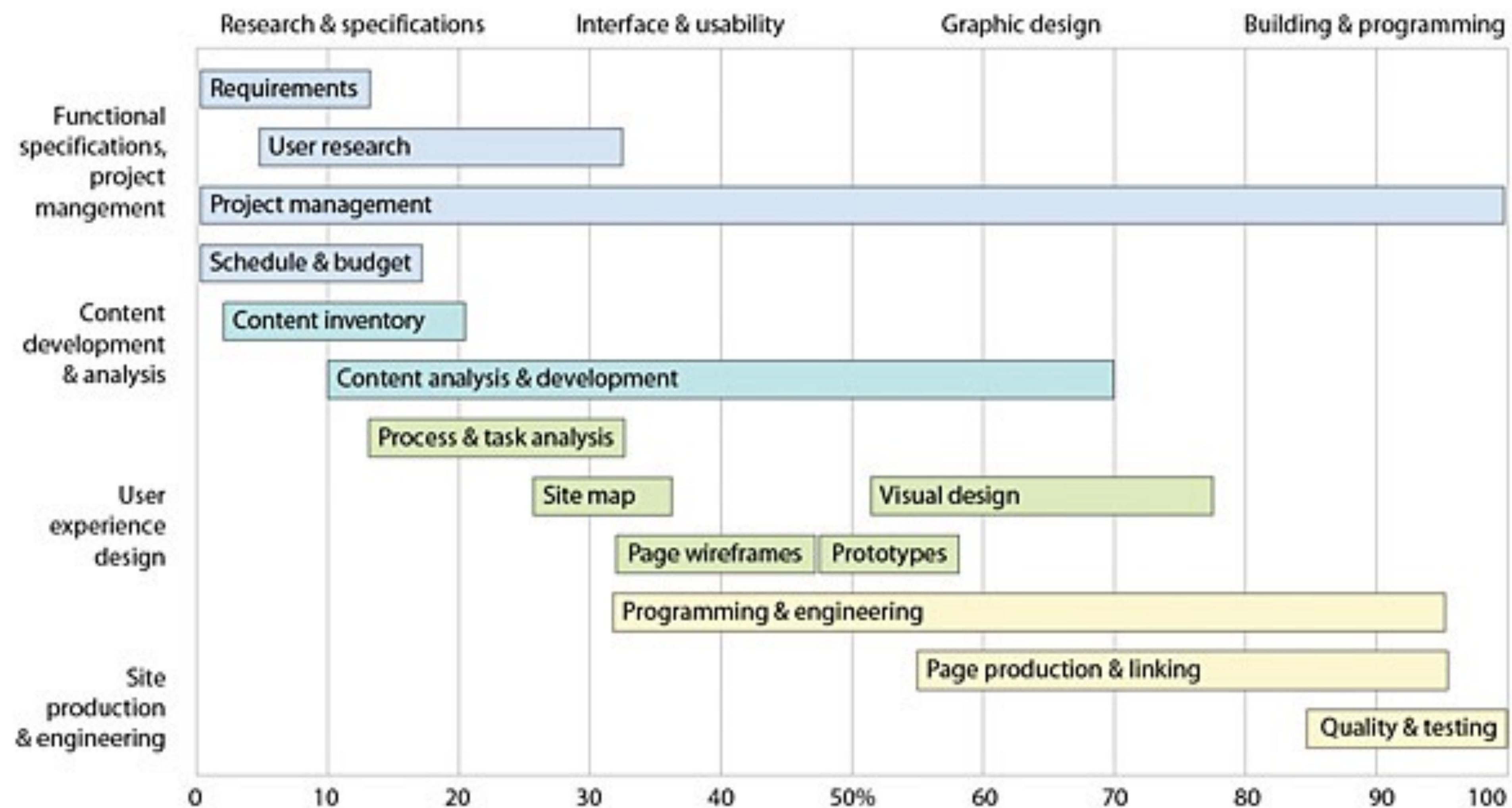
- **Exclusive server-side rendering**
 - Static – HTML corresponds to pre-built files on the server
 - Dynamic – HTML results from applications executed (on request) on the server
- **Server and client-side**
 - Server-side rendering with hydration
HTML is dynamically built on the server with updates on the client (logic mostly on the server)
 - Client-side rendering with server pre-rendering
HTML initial version is prepared on the server but the client then takes over (logic mostly on the client)
- **Exclusive client-side rendering**
 - Full client-side rendering
Only a minimal skeleton is served from the server, all the application logic and rendering is done on the client

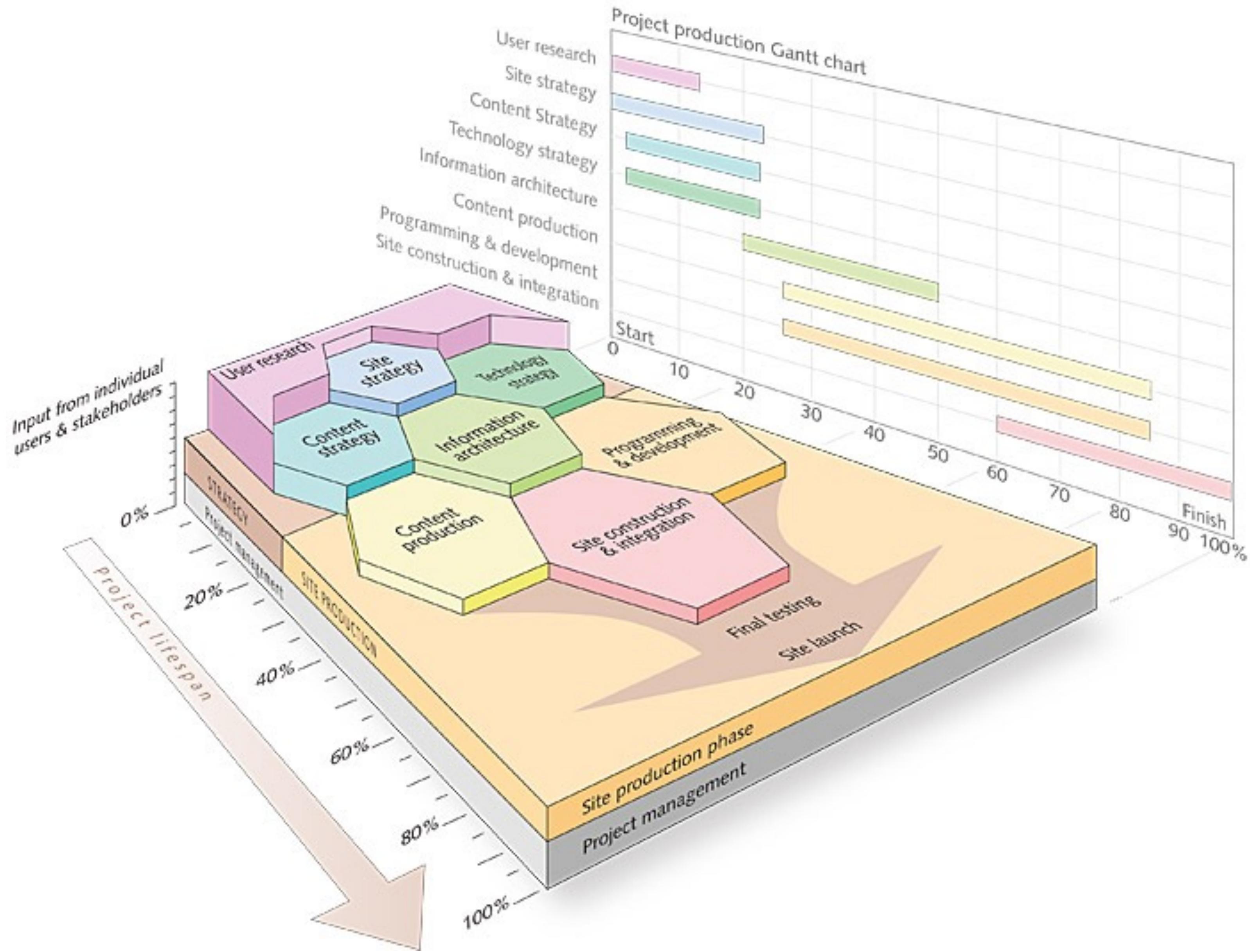
Discussion on Architectural Options

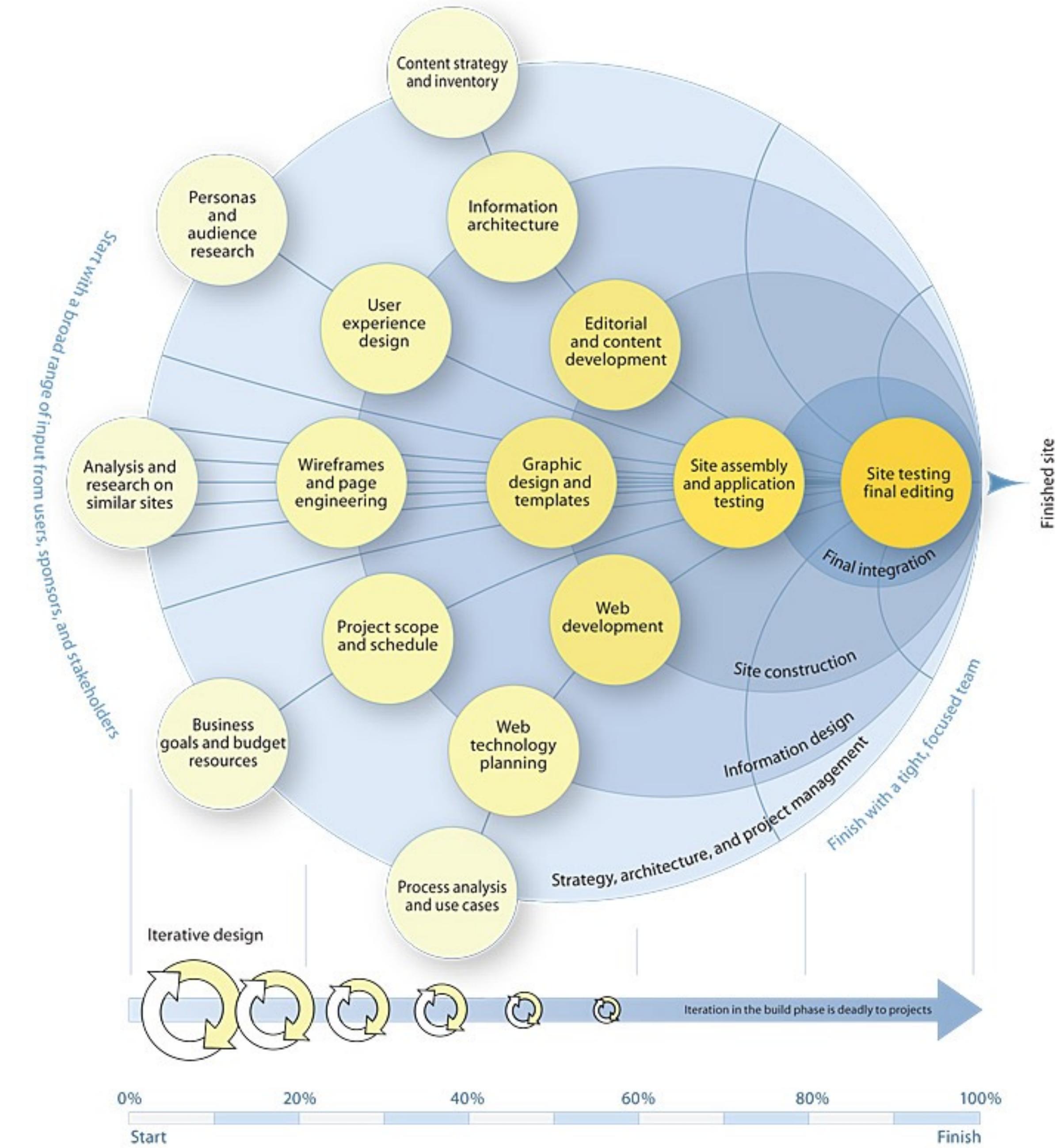
- Personal web site
- Personal web site with comments
- Amazon
- SIGARRA
- Slack

Web Site Development Process









Web Technologies

The Web Stack of Technologies (1)

- **Networking** – DNS, TCP/IP.
- **Web Protocols** – HTTP, REST.
- **Data Formats** – HTML, JSON, XML.
- **Web Servers** – Apache, IIS, nginx, lighttpd, node.js.
- **Data Storage** – SQL, NOSQL, cache system, file system.
- **Server-side Programming** – Languages, libraries, frameworks.
- **Content Management Systems** – Reusable building blocks (e.g. Wordpress).
- **Library and framework management**
- **Building and packaging** – How to distribute web products: hosting, CDNs.

The Web Stack of Technologies (2)

- **Device Capabilities** – Which features are supported.
- **Template Design** – Reusable, modular solutions.
- **Content Presentation** – HTML, CSS.
- **Interface Programming** – Also HTML and CSS but a lot of JavaScript.
- **Performance** – How to improve response speed.
- **Resilience** – How to make reliable systems.
- **Security** – Critical and complex in web systems.
- **Navigation Design** – How to organize content and navigation.
- **Graphics** – Images, SVG, Canvas.

Web Technologies

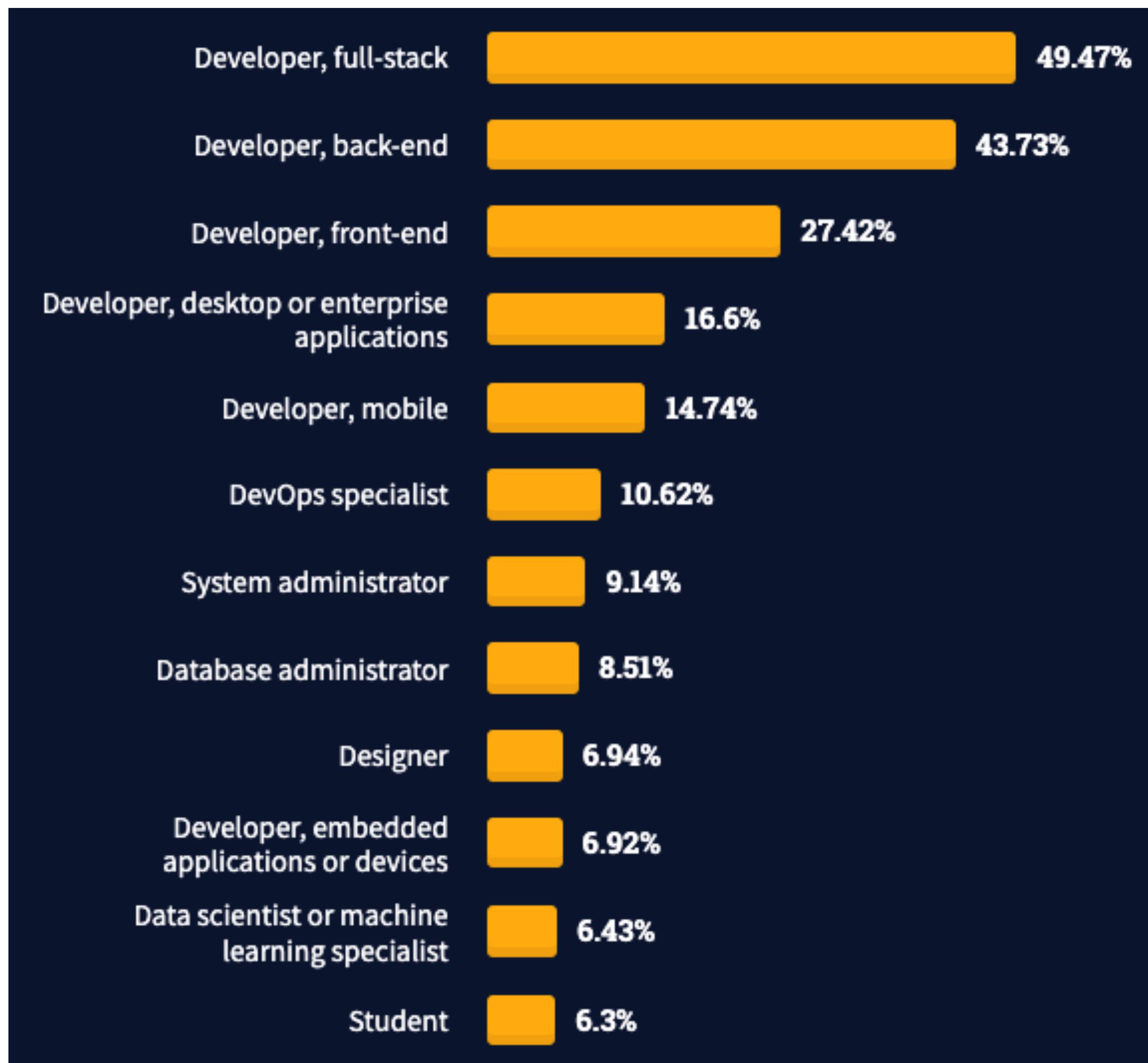
- **The modern web technology stack is huge.**
- There are many different options (at different tiers) for the same task.
- No one is a “web expert”.
- But knowing about the full web stack is important for collaboration.
- Web development is organized in two major areas:
client-side and **server-side** development.

Stack Overflow Developer Survey 2021

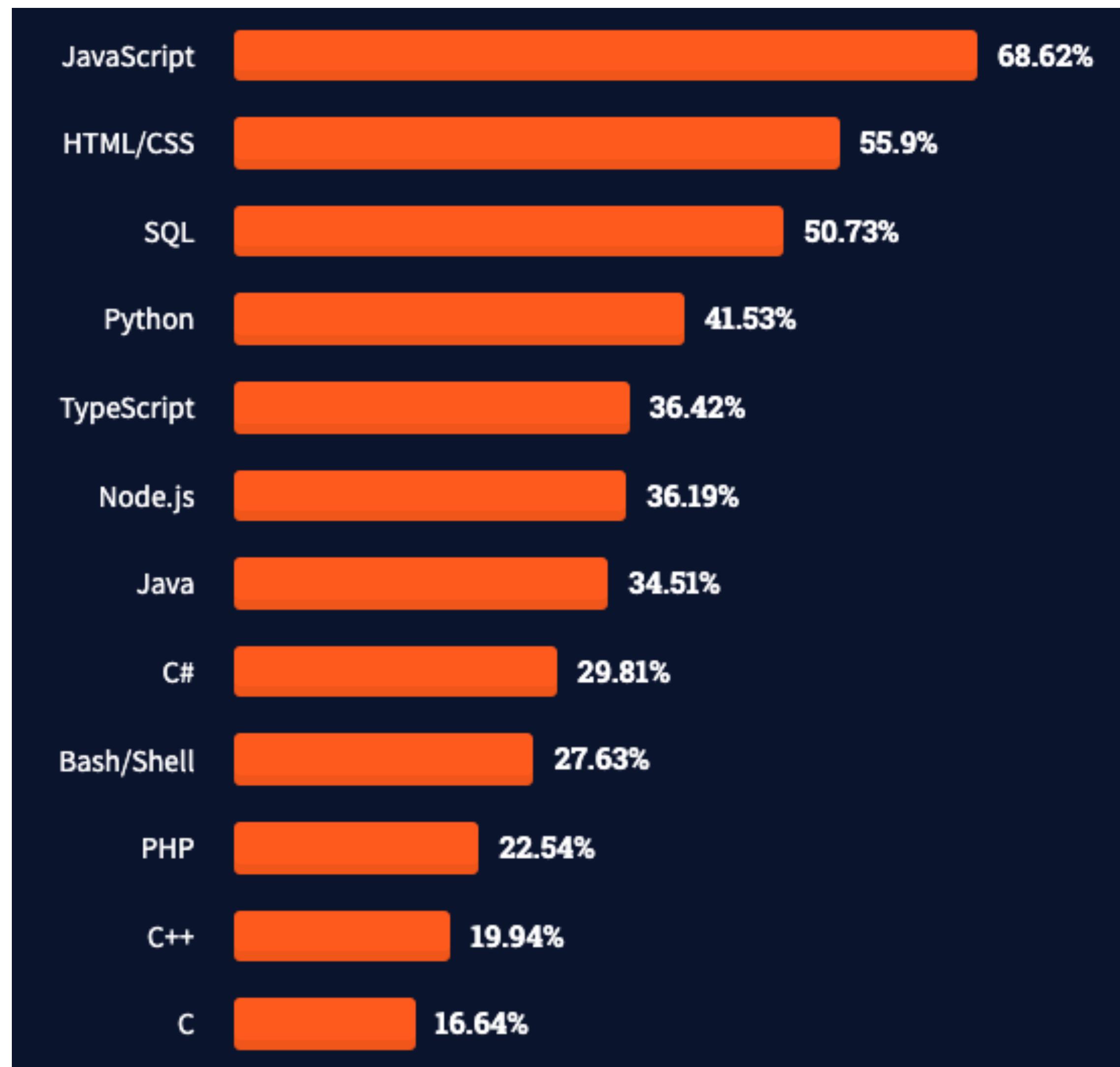


- <https://insights.stackoverflow.com/survey/2021>

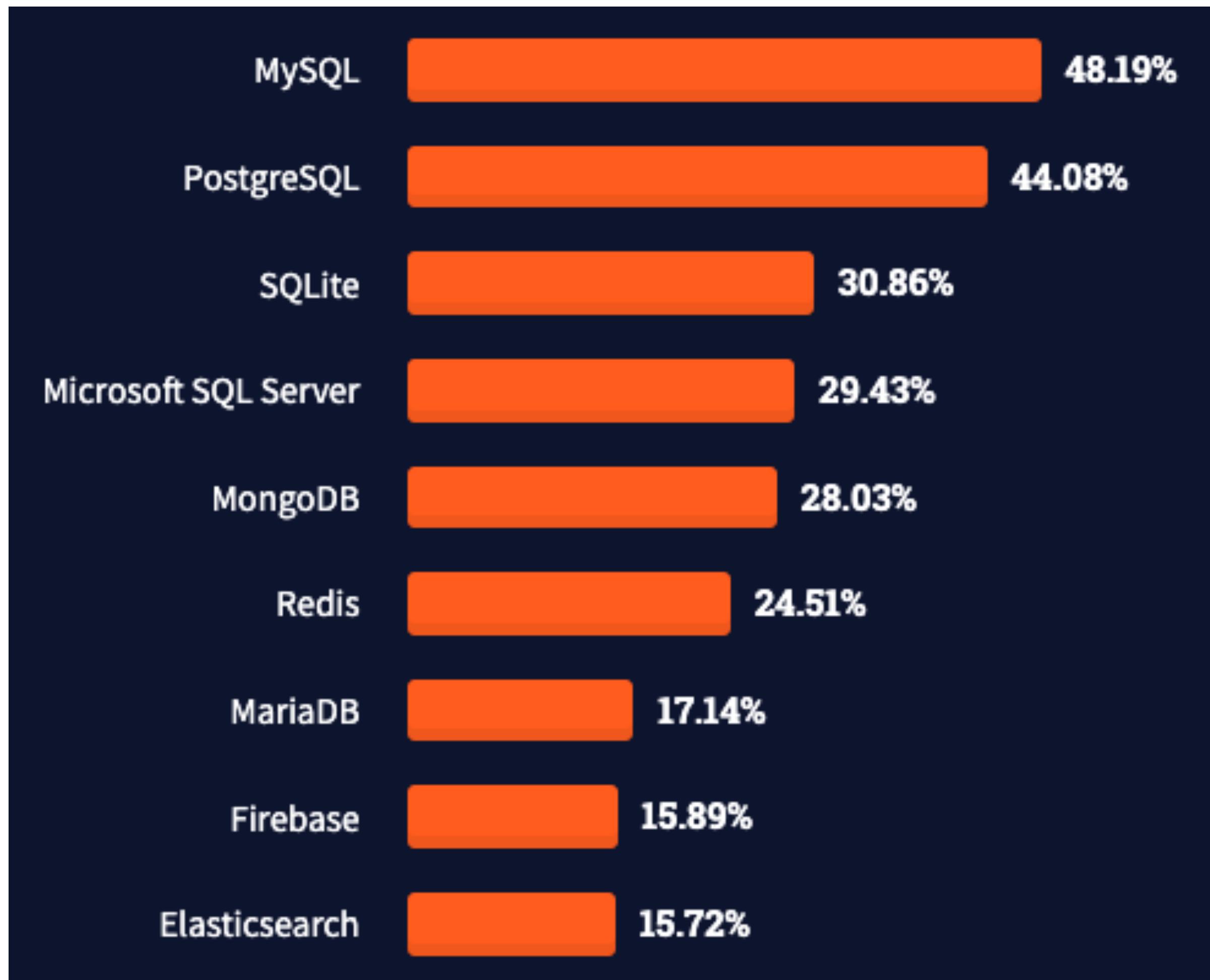
Developer Roles



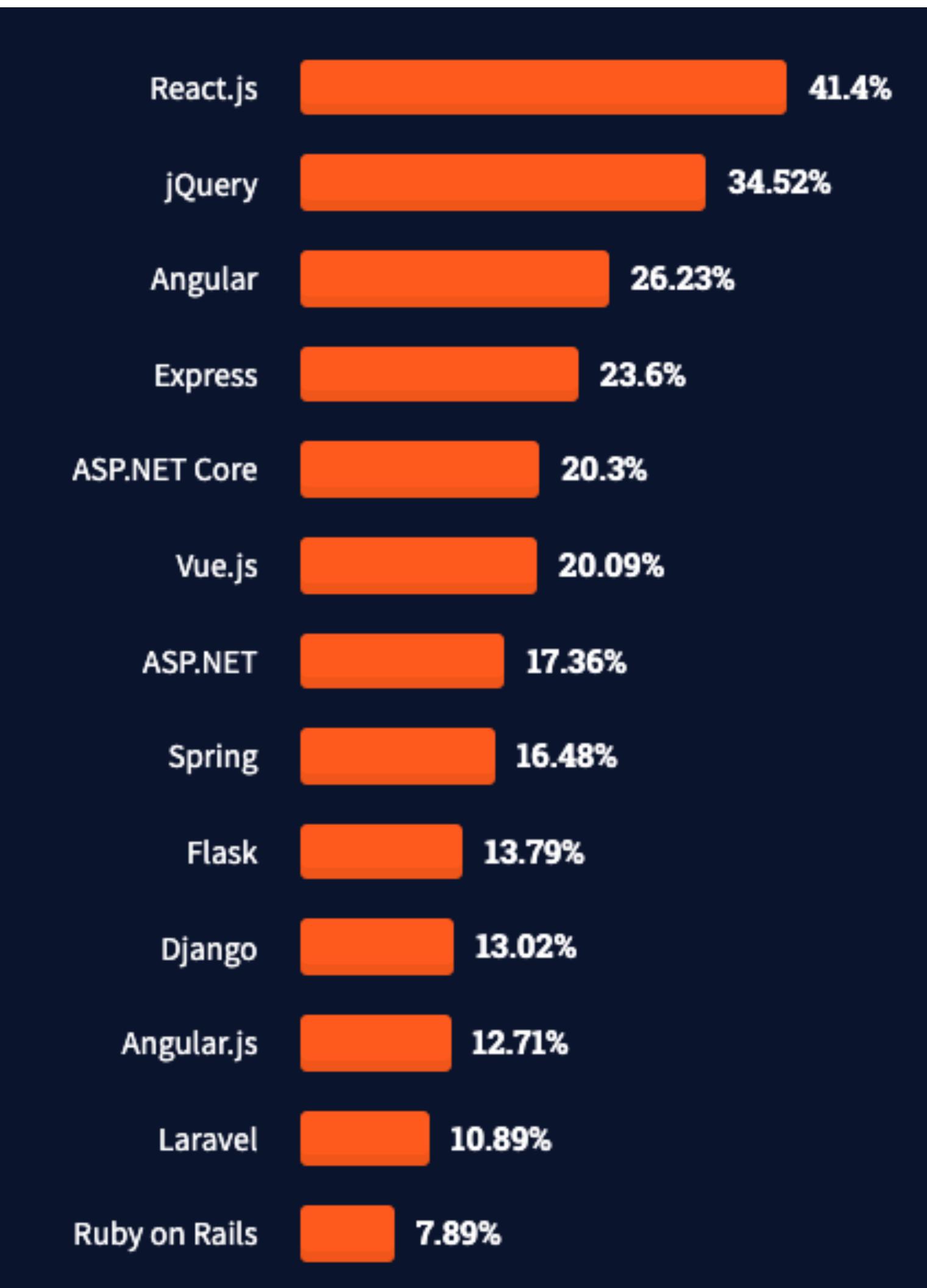
Popular Technologies



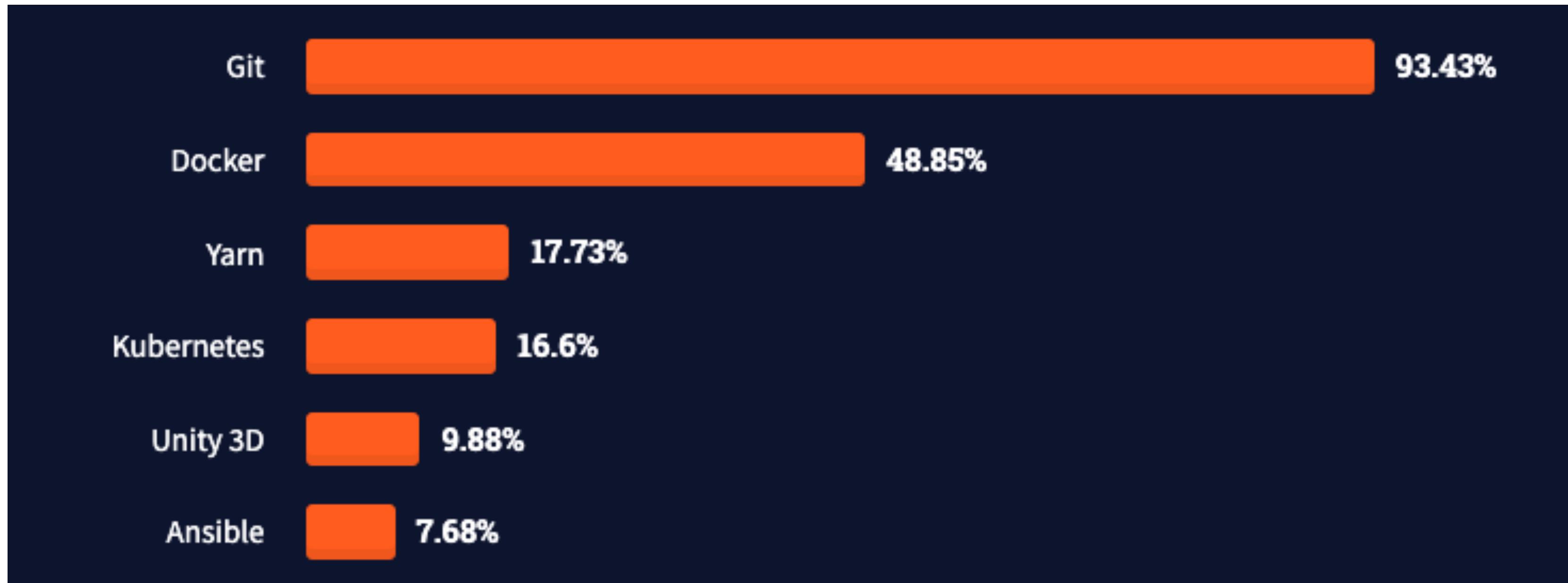
Database Technologies



Web Frameworks



Other Tools



Server-Side Technologies

- There are many options in server-side technologies. In theory as many as programming languages. Most popular options: PHP, Java, Python, Ruby, Perl, C#, Go. Also JavaScript with node.js.
- Server-side technologies also includes data management solutions, e.g. filesystem, database management systems. Most popular options: MySQL, PostgreSQL, Oracle, SQL Server, SQLite.
- **In LBAW we will use PHP, Laravel framework, and PostgreSQL.**

Client-Side Technologies

- Client-side technologies run on the web client.
- Three main technologies: HTML, CSS and JavaScript. HTML and CSS are declarative languages, while JavaScript is a full-fledged programming language.
- There is an increasing number of libraries and *boilerplates* to support client-side web development.
- **In LBAW we will use HTML, CSS, JavaScript and AJAX.**
- Also notes on web performance, security, accessibility and usability.

Further Reading

- "As We May Think"
Vannevar Bush. Atlantic Monthly, 1945
- Computer Networks and Internets
Douglas Comer. Prentice Hall, 2004
- HTTP: The Definitive Guide
David Gourley, Brian Totty. O'Reilly, 2002
- Opera Web Standards Curriculum
<http://dev.opera.com/articles/wsc/>
- The Modern Web: Multi-Device Web Development with HTML5, CSS2 and JavaScript
Peter Gasston. No Starch Press, 2013
<http://modernwebbook.com/>

Server-Side Web Development

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Current Status

- Plan:
 - 6th week of classes;
 - Project: ER feedback; last class before EBD delivery;
 - Lecture: server-side web development; review web architectures;
 - Labs: finish database specification (EBD);
- Monitor sessions: Tuesday, at 16h
 - Previous sessions: Git and GitFlow; PostgreSQL (setup);
 - This week: Postgres indexes, triggers and transactions

Outline for Today

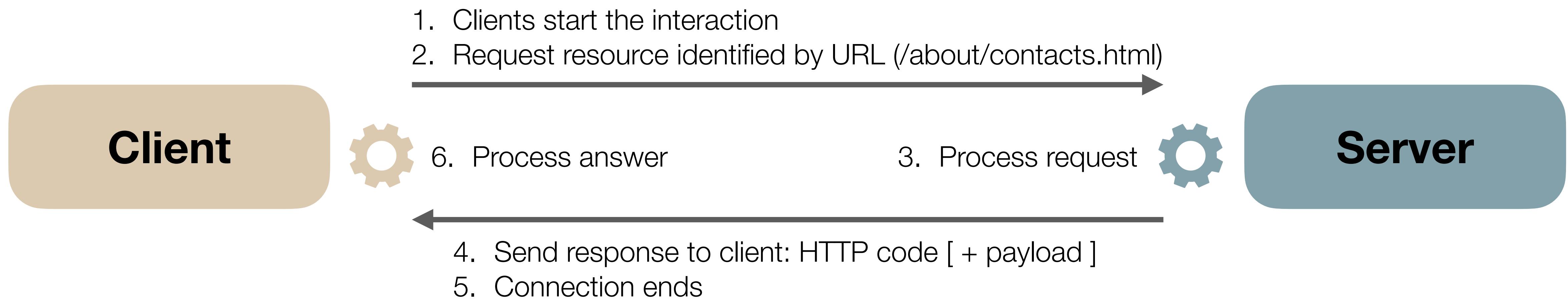
- Server-Side Web Development
 - Overview and main concepts
 - AJAX
 - Review Web Architectures
- Architecture Specification and Prototype (EAP) component
- A7: Web Resources Specification

Rendering on the Web

The World Wide Web

- A distributed information system, with
 - Web servers, waiting for client requests to handle
 - Web clients (browsers and others), used to navigate in this 'information space'
- Core technologies
 - URL, defines how to address information resources published on the web
 - HTML, defines how to represent information on the web
 - HTTP, defines how web clients can interact with web servers

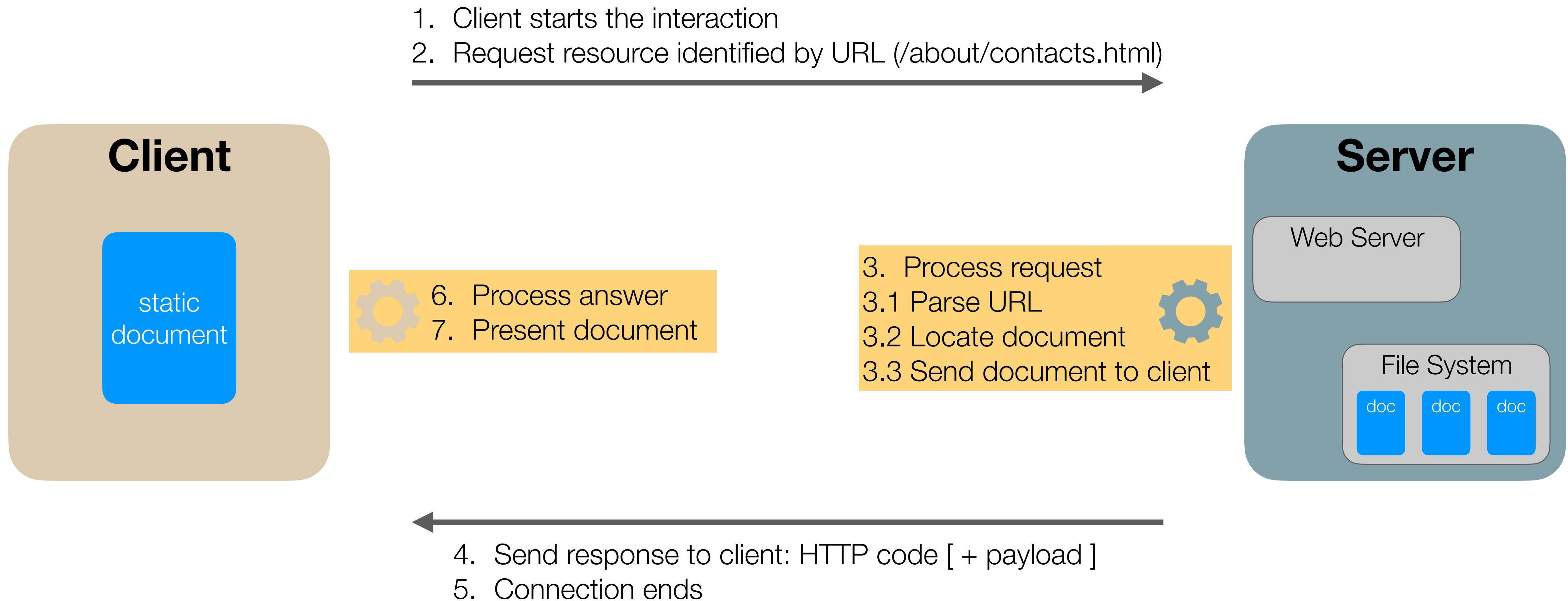
Client-Server Paradigm



Static and Dynamic Web Documents

- Web documents (or resources) are HTML, CSS, JS, JSON, images, media, i.e. *anything*.
- **Static web documents** have no processing involved
 - Web servers simply locate the resource and send it to clients. No rendering or code execution, documents were previously produced and are served without changes.
 - Use cases: images, CSS, HTML documents without 'live data', rarely updated.
- **Dynamic web documents** are prepared when requested
 - Dynamic web documents don't exist in advance, they are prepared in the moment for the client that issued the request. Can make use of data in databases, external sources, APIs, client information, etc.
 - Use cases: HTML with information read from a database; personalized web page; web application.

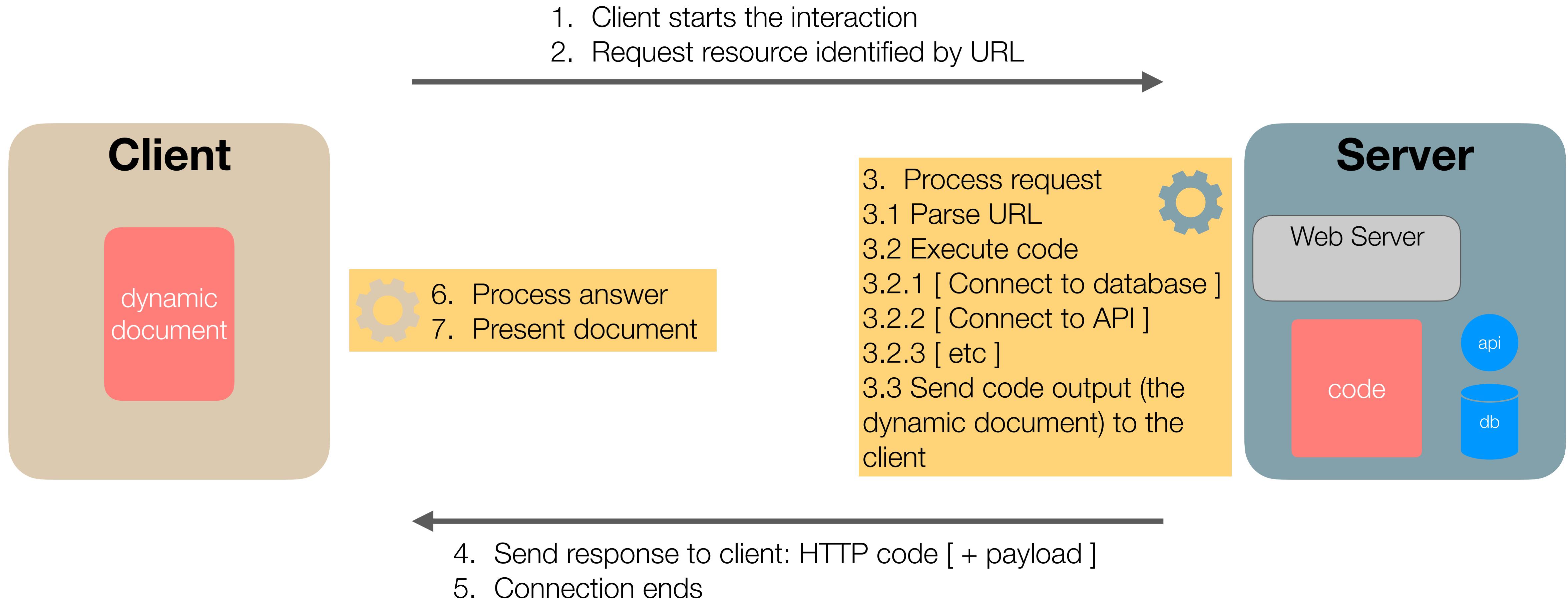
Static Web Documents



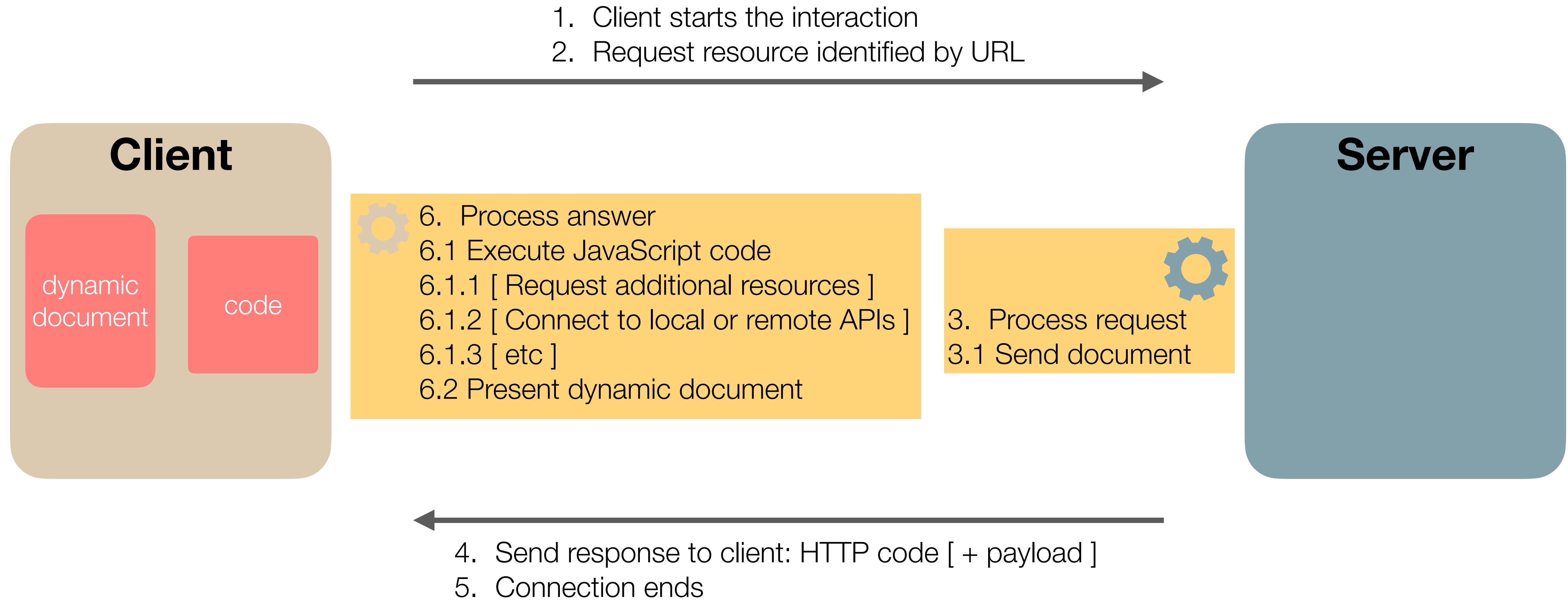
Dynamic Web Documents

- Dynamic web documents are prepared on the fly in response to a specific client request and can be prepared on the server or on the client.
- Server-side rendering (SSR)
 - The document is prepared on the server and the result is sent to the client.
 - Code execution is on the server.
- Client-side rendering (CSR)
 - The server send to the client the necessary elements to build the document (JS code in particular)
 - The client builds the document using JavaScript.
- A mix of these techniques can be implemented, e.g. AJAX.

Server-Side Dynamic Web Documents



Client-Side Dynamic Web Documents



Rendering on the Web

- Deciding where to render web resources is an architectural decision involving multiple trade-offs.
 - No solution (as always!) is best for all cases.
-
- Static web documents
 - Fast, but rigid, i.e. no updates, no personalization, hard to maintain in scale.
 - Dynamic web documents
 - Server-side: complex documents with live information, but require full round trips to the server for user interaction.
 - Client-side: complex interactive document (feel like a local app), but heavy on the client and not really hypertext.
-
- In practice, multiple solutions coexist for each particular use case, i.e. multiple architectural options in different parts of web systems, but also multiple architectural options on individual web documents.

Server-Side Web Development

Server-Side Web Development

- In LBAW we adopt a server-side approach.
- With client-side code in specific use cases to improve user experience and support immediate changes without the need of a new full-page request (using AJAX).
- In LBAW, to simplify code, improve modularity, and organize development, we define two types of web resources (endpoints or simply pages).
 - View web resources, only access data for presentation — output is HTML.
 - Action web resources, change data and then redirect to a view web resource.

Server-Side Web Development

- A web-based software system based on the server publishes a set of endpoints.
 - /
 - /about.html
 - /students/view_student.php
 - /search?q=flowers
- Each endpoint, also called web resource, accepts HTTP requests and outputs HTTP responses.
 - HTTP requests includes a method and, optionally, parameters and a payload.
 - HTTP responses include a code and, optionally, a payload.

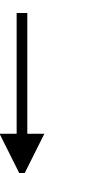
HTTP Request
[Method + Parameter + Payload]



Server-Side Code

- validate and process request
- obtain data (files, databases, ...)
- process data
- prepare output (HTML, JSON, ...)
- send response

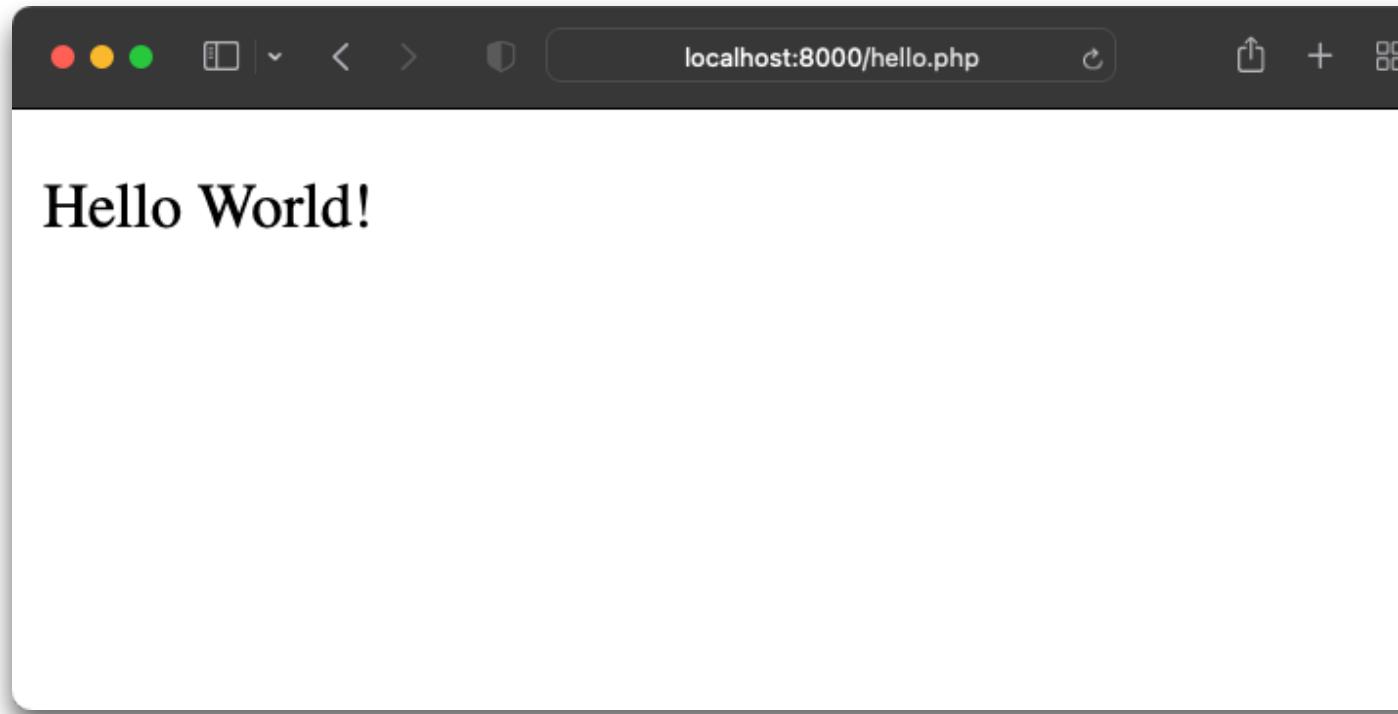
typical execution flow



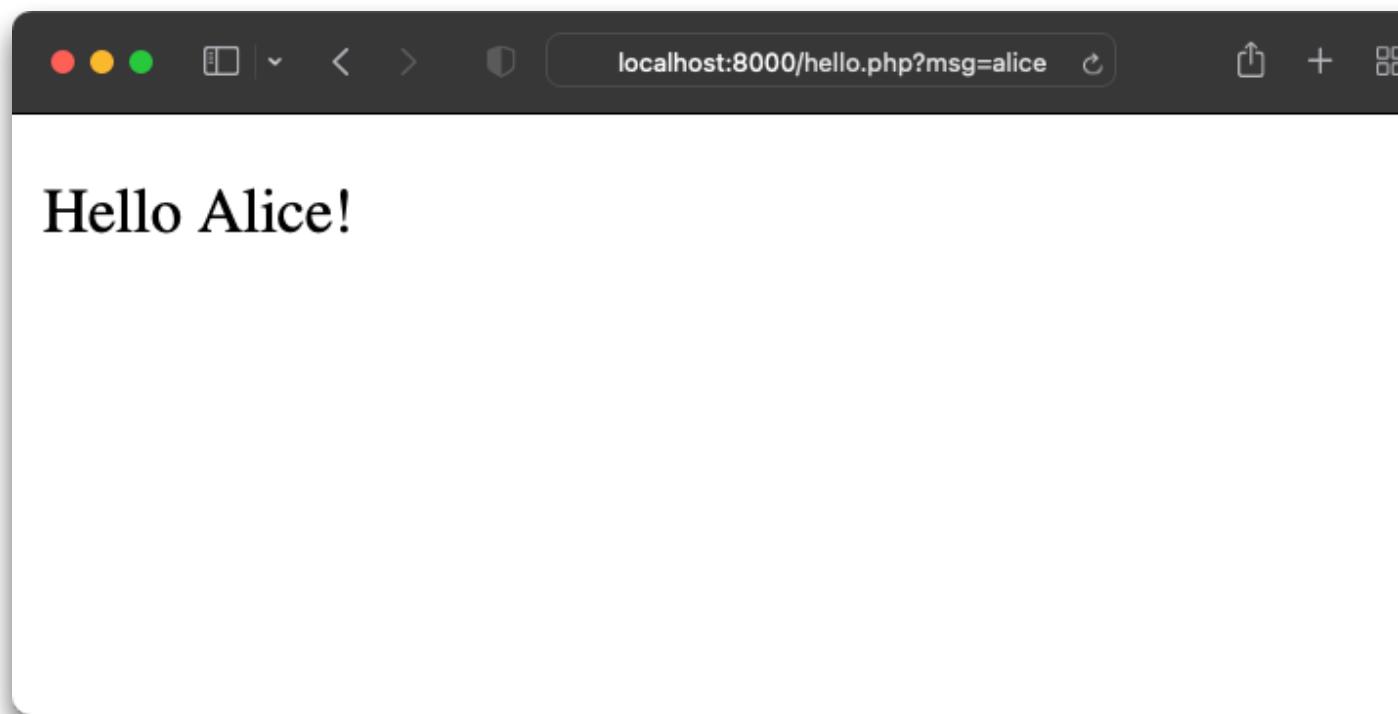
HTTP Response
[Code + Payload]

Example in PHP

/hello.php



/hello.php?msg=alice



```
<?php
// hello.php file

// Check if parameters were sent.
if (isset($_GET["msg"])) {
    $msg = $_GET["msg"];
} else {
    $msg = "world";
}

// Capitalize message.
$msg = ucfirst($msg);

// Output HTML.
echo "<!DOCTYPE html>\n";

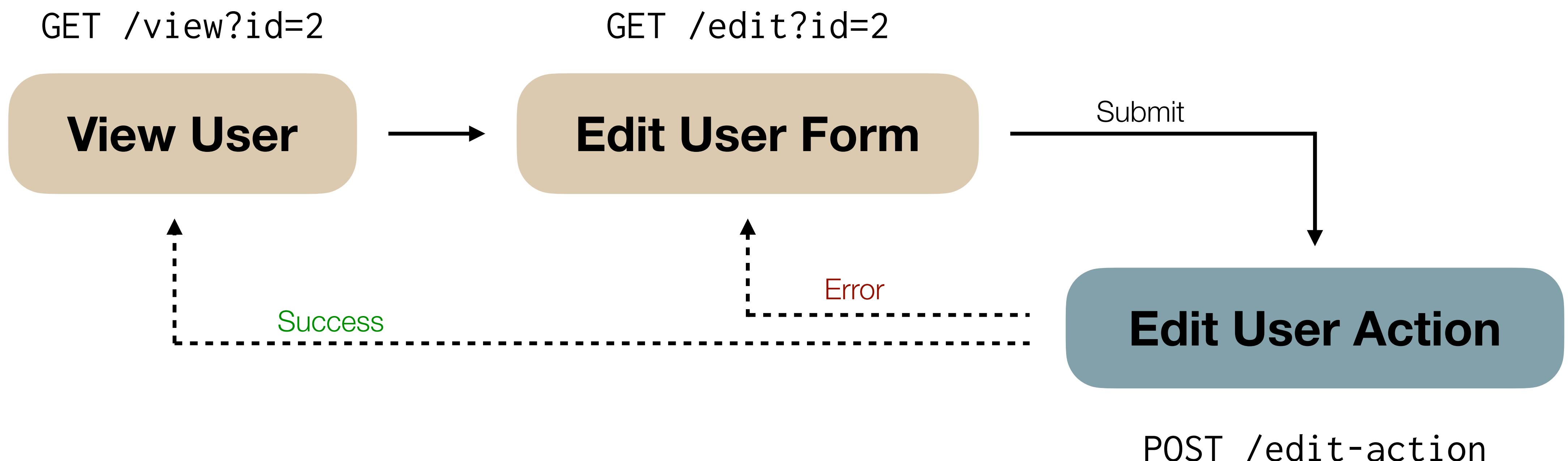
// Example of multiline printing.
echo <<<HTML_HEAD
<html>
<body>
HTML_HEAD;

// Print the paragraph with the message.
echo "\t<p>Hello $msg!</p>\n";
?>

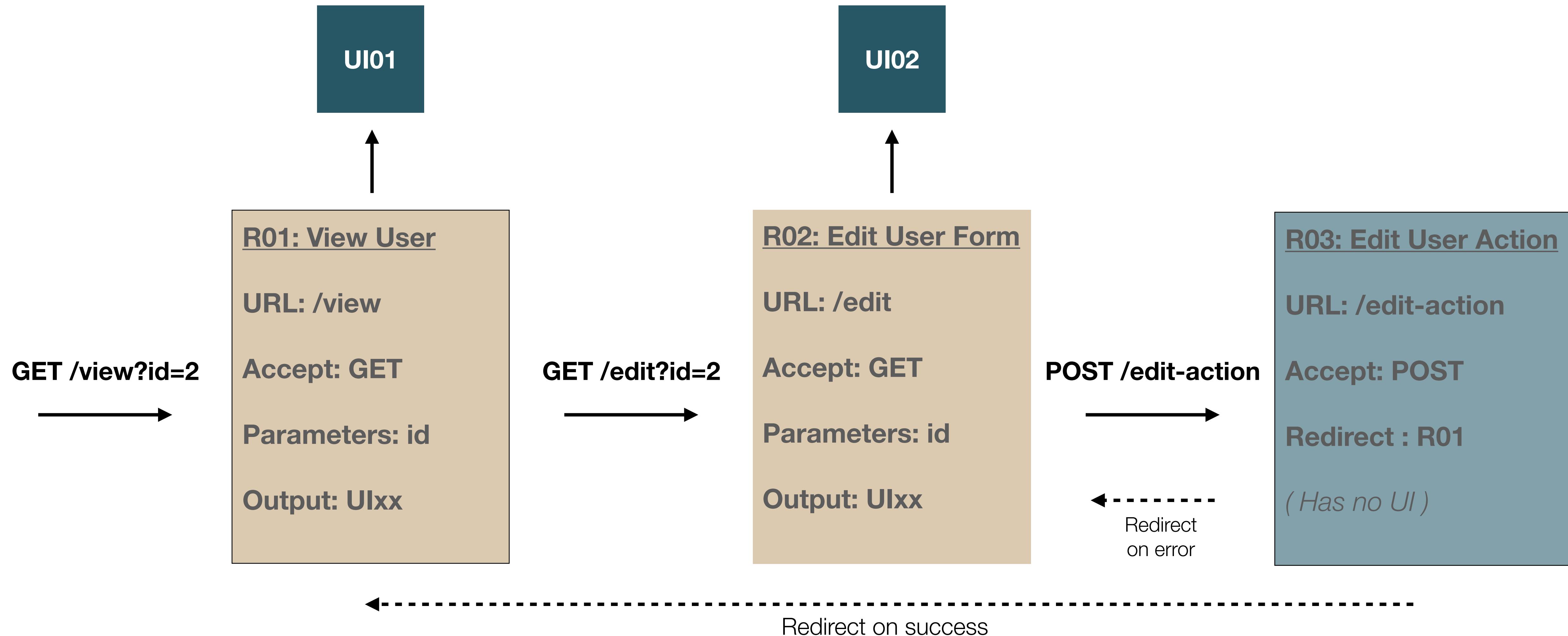
<!-- This is outside the PHP block. Just plain HTML. -->
</body>
</html>
```

View and Action Web Resources

- View pages read and present data. No changes to data.
- Action pages alter data and redirect to a view page. No presentation of data.



Corresponding Web Resources Specification



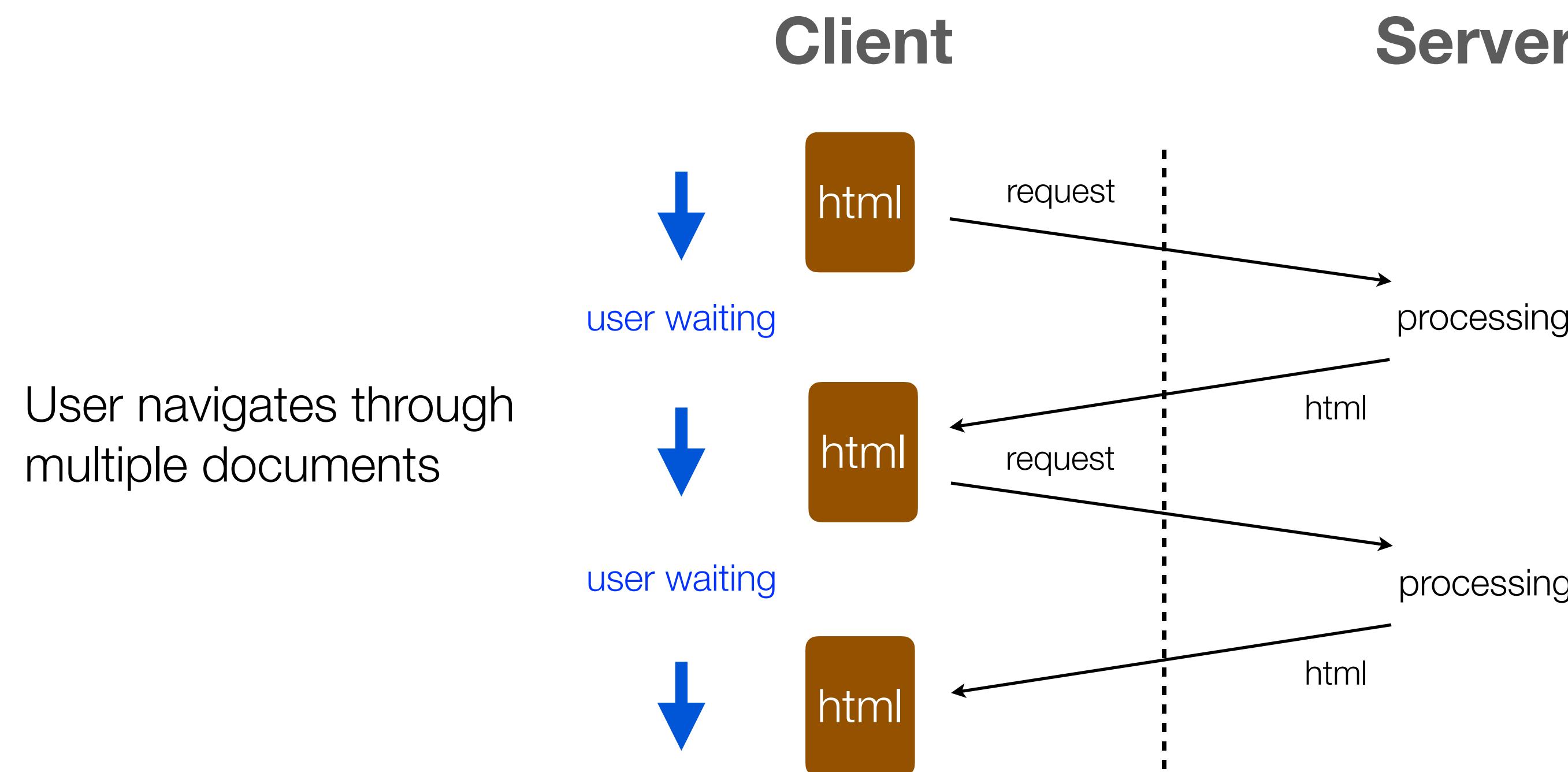
Ajax

Ajax

- Set of web development techniques to implement asynchronous web interactions.
- In standard synchronous web interaction, the user navigates across independent documents. There is a one-to-one mapping between application views and web pages.
- With AJAX, user interactions can be implemented within the same document. A n-to-one mapping between application views and web pages is possible.
- Basic use case: like action in a post does not result in leaving the current document.
- Using AJAX techniques, web documents can make requests to the server.

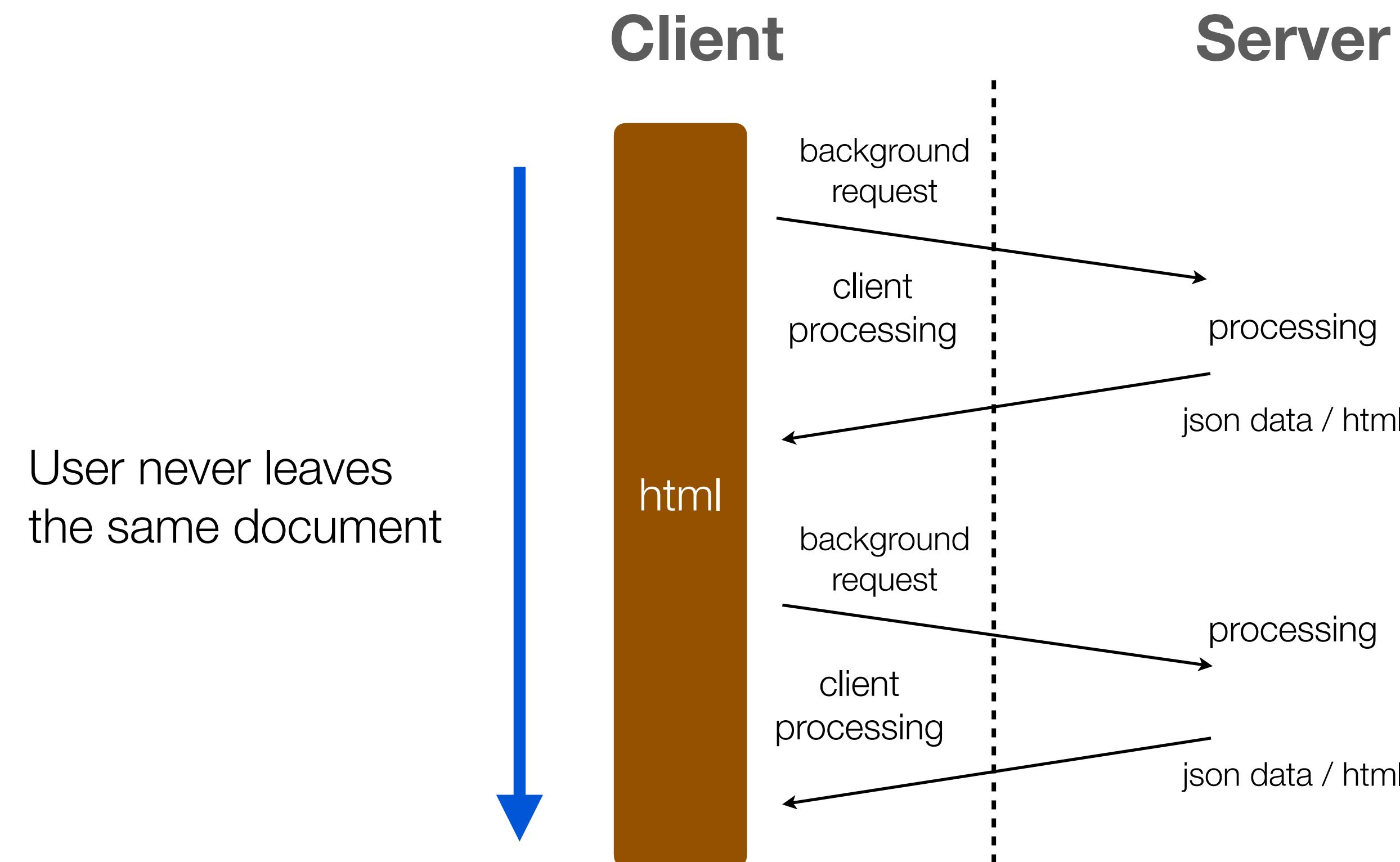
Synchronous User Interaction

- In the classic web interaction model, a click results in the request of a new document: round trip to the server, the server prepares a new document, the client needs to render the document from zero.



Asynchronous User Interaction

→ In asynchronous user interaction, the document makes requests to the server.



Server-Side Technologies

- In server-side web development there is a large and diverse set of options
- Web servers, e.g. Apache, IIS, nginx, cloud, ...
- Web frameworks, e.g. Laravel, Django, Ruby on Rails, Spring, ...
- Storage solutions, e.g. relational, document, key-value, ...
- Programming languages, e.g. Python, PHP, Go, Ruby, ...
 - Routing libraries
 - Templating libraries
 - ...
- Next: a lot of common and recurring server-side use cases led to the development of web frameworks.

Summary

- In LBAW we adopt a server-side architecture using Laravel, a PHP framework.
- AJAX is used to improve user experience.
- In the Architecture Specification and Vertical Prototype component (EAP)
 - Define the architecture of the application by specifying the web resources;
 - Implement a vertical prototype to validate technology stack.

References

- PHP: The Right Way
<https://phptherightway.com/>
- MDN Web Docs, Server-side website programming
<https://developer.mozilla.org/docs/Learn/Server-side>
- MDN Web Docs, Ajax
<https://developer.mozilla.org/docs/Web/Guide/AJAX>
- Google Developer, Rendering on the Web (2019)
<https://developers.google.com/web/updates/2019/02/rendering-on-the-web>

A7 - Web Architecture Specification

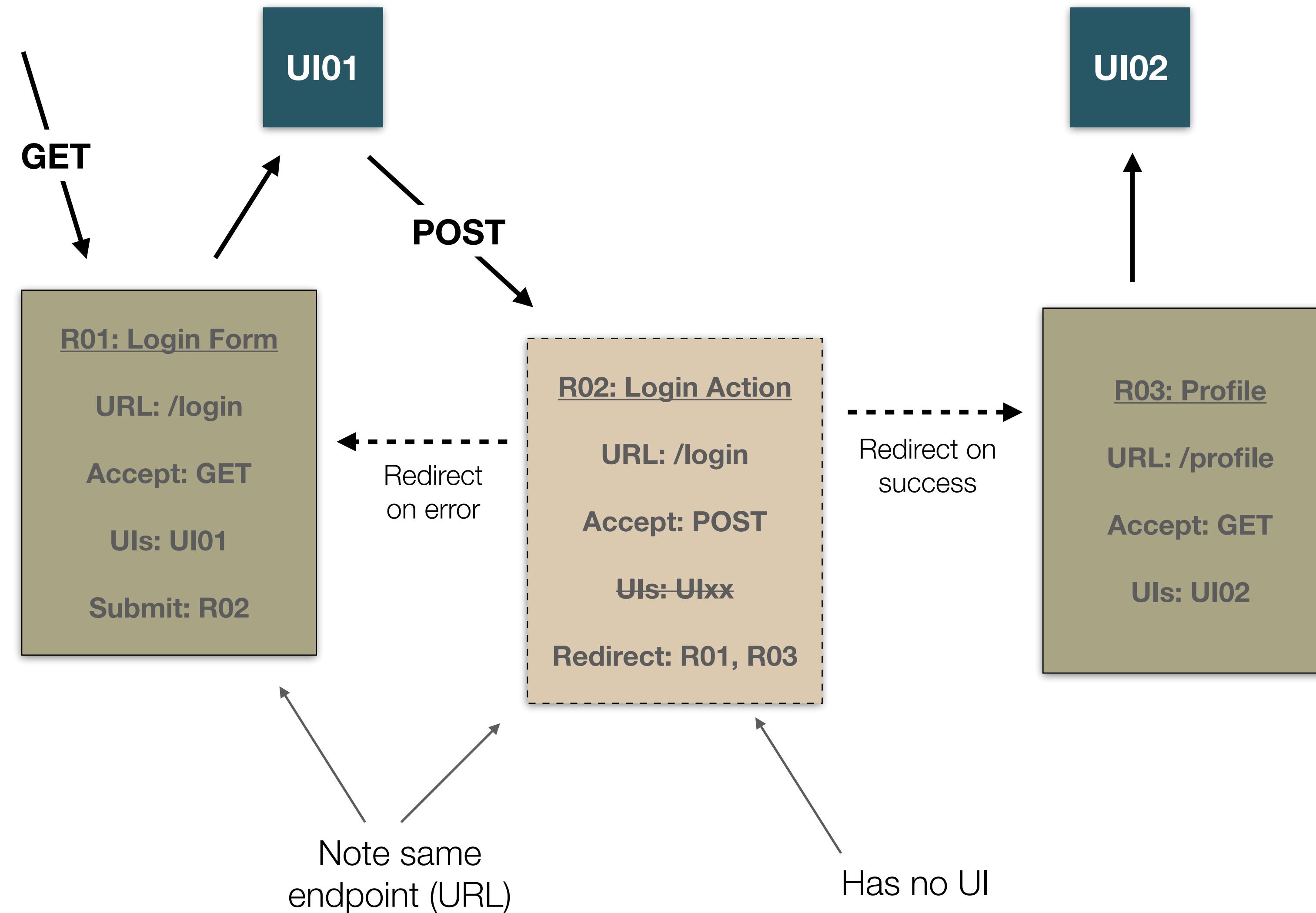
Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Web Resources

- 'Web resources' represent your web application endpoints.
- The A7 artifact corresponds to the identification and description of endpoints.
- Types of endpoints:
 - **View endpoints:** show user interface to the user (A3 UIs), returns HTML;
 - **Action endpoints:** process information then redirects to view endpoint;
 - **Ajax endpoints:** available for asynchronous calls, returns JSON / HTML*.
- * *JSON output requires rendering on the client; HTML is pre-rendered on the server.*

Overall Architecture – "View-Action Pages Pattern"



OpenAPI

- Goal: standardize on how APIs are described.
- <https://spec.openapis.org/oas/v3.1.0>
 - "The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic."
- The OpenAPI Initiative (OAI) is anchored under the Linux Foundation.
- An OpenAPI document is a JSON object, and may be represented either in JSON or YAML.

Example OpenAPI Document (YAML)

```
openapi: 3.0.0

info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200': # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Metadata

```
openapi: '3.0.0'

info:
  version: '1.0'
  title: 'LBAW MediaLibrary Web API'
  description: 'Web Resources Specification (A7) for MediaLibrary'
```

External Documentation

```
externalDocs:  
  description: Find more info here.  
  url: https://web.fe.up.pt/~ssn/wiki/teach/lbaw/medialib/a07
```

Servers

- The servers section specifies the API server and base URL. You can define one or several servers, such as production and sandbox.
- All API paths are relative to the server URL.

```
servers:  
- url: http://lbaw.fe.up.pt  
  description: Production server
```

Tags

- Used to grouping API operations (endpoints).
- Use for the defined modules.

```
tags:  
  - name: 'M01: Authentication and Individual Profile'  
  - name: 'M02: Works'  
  - name: 'M03: Reviews and Wish list'  
  - name: 'M04: Loans'  
  - name: 'M05: User Administration and Static pages'
```

Paths

- In OpenAPI terms, **paths** are endpoints (resources) that your API exposes, and operations are the HTTP methods used to manipulate these paths, such as GET, POST or DELETE.

```
paths:  
  /ping:  
    ...  
  /users:  
    ...  
  /users/{id}:  
    ...
```

View Page (HTTP GET) OpenAPI

```
/login:  
  get:  
    operationId: R101  
    summary: 'R101: Login Form'  
    description: 'Provide login form. Access: PUB'  
  
  tags:  
    - 'M01: Authentication and Individual Profile'  
  
  responses:  
    '200':  
      description: 'Ok. Show log-in UI'
```

Action Page (HTTP POST) – Form Submission OpenAPI (1/3)

→ Endpoint header.

```
[/login:]  
  
post:  
  operationId: R102  
  summary: 'R102: Login Action'  
  description: 'Processes the login form submission. Access: PUB'  
  tags:  
    - 'M01: Authentication and Individual Profile'
```

Action Page (HTTP POST) – Form Submission OpenAPI (2/3)

- Request body section defines the accepted parameters.
- Note the different types, and the required parameters.

```
[/login:]  
[post:]  
  
requestBody:  
  required: true  
  content:  
    application/x-www-form-urlencoded:  
      schema:  
        type: object  
        properties:  
          email:          # <!-- form field name  
            type: string  
          password:      # <!-- form field name  
            type: string  
        required:  
          - email  
          - password
```

Action Page (HTTP POST) – Form Submission OpenAPI (3/3)

→ All action pages have redirects.

```
responses:  
  '302':  
    description: 'Redirect after processing the login credentials.'  
    headers:  
      Location:  
        schema:  
          type: string  
    examples:  
      302Success:  
        description: 'Successful authentication. Redirect to user profile.'  
        value: '/users/{id}'  
      302Error:  
        description: 'Failed authentication. Redirect to login form.'  
        value: '/login'
```

Action Page – Logout

```
/logout:

post:
  operationId: R103
  summary: 'R103: Logout Action'
  description: 'Logout the current authenticated user. Access: USR, ADM'
  tags:
    - 'M01: Authentication and Individual Profile'

  responses:
    '302':
      description: 'Redirect after processing logout.'
      headers:
        Location:
          schema:
            type: string
      examples:
        302Success:
          description: 'Successful logout. Redirect to login form.'
          value: '/login'
```

View Page – Register Form

```
/register:  
  
  get:  
    operationId: R104  
    summary: 'R104: Register Form'  
    description: 'Provide new user registration form. Access: PUB'  
    tags:  
      - 'M01: Authentication and Individual Profile'  
  
    responses:  
      '200':  
        description: 'Ok. Show sign-up form UI'
```

Action Page – Register (1/3)

```
post:  
  
    operationId: R105  
    summary: 'R105: Register Action'  
    description: 'Processes the new user registration form submission. Access: PUB'  
    tags:  
        - 'M01: Authentication and Individual Profile'
```

Action Page – Register (2/3)

```
requestBody:  
  required: true  
content:  
  application/x-www-form-urlencoded:  
    schema:  
      type: object  
      properties:  
        name:  
          type: string  
        password:  
          type: string  
        email:  
          type: string  
        picture:  
          type: string  
          format: binary  
required:  
  - email  
  - password
```

Action Page – Register (3/3)

```
responses:  
  '302':  
    description: 'Redirect after processing the new user information.'  
    headers:  
      Location:  
        schema:  
          type: string  
    examples:  
      302Success:  
        description: 'Successful authentication. Redirect to user profile.'  
        value: '/users/{id}'  
      302Failure:  
        description: 'Failed authentication. Redirect to register form.'  
        value: '/register'
```

View Page – View User

```
/users/{id}:
  get:
    operationId: R106
    summary: 'R106: View user profile'
    description: 'Show the individual user profile. Access: USR'
    tags:
      - 'M01: Authentication and Individual Profile'

    parameters:
      - in: path
        name: id
        schema:
          type: integer
        required: true

  responses:
    '200':
      description: 'Ok. Show the view user profile UI'
```

AJAX Page – Search Works (1/3)

```
/api/works:  
  get:  
    operationId: R202  
    summary: 'R202: Search Works API'  
    description: 'Searches for works and returns the results as JSON. Access: PUB.'
```

AJAX Page – Search Works (2/3)

```
parameters:
  - in: query
    name: query
    description: String to use for full-text search
    schema:
      type: string
      required: false
  - in: query
    name: item
    description: Category of the works
    schema:
      type: string
      required: false
  - in: query
    name: loaned
    description: Boolean with the loaned flag value
    schema:
      type: boolean
      required: false
  - in: query
    name: owner
    description: Boolean with the owner flag value
    schema:
      type: boolean
      required: false
  - in: query
    name: classification
    description: Integer corresponding to the work classification
    schema:
      type: integer
      required: false
```

AJAX Page – Search Works (3/3)

```
responses:  
  '200':  
    description: Success  
    content:  
      application/json:  
        schema:  
          type: array  
          items:  
            type: object  
            properties:  
              id:  
                type: string  
              title:  
                type: string  
              obs:  
                type: string  
              year:  
                type: string  
              owner:  
                type: string  
              type:  
                type: string
```

continue -->

--> continued

```
example:  
  - id: 1  
    title: Rihanna - Unapologetic  
    obs: Good pop music album.  
    year: 2012  
    owner: Joana Lima  
    type: MP3  
  - id: 15  
    title: Mr. Bean  
    obs: The best comedy movie.  
    year: 1995  
    owner: Manuel Teixeira  
    type: DVD
```

GitLab OpenAPI Rendering

- GitLab provides an interactive human-readable view to a OpenAPI specification
- File type must be YAML (.yaml) and include 'openapi' in filename
- Recommended: a7_openapi.yaml
- MediaLibrary: [GitLab > Ibaw > code-examples > medialibrary > a7_openapi.yaml](#)

GitLab OpenAPI Rendering

LBAW MediaLibrary Web API 1.0 OAS3
[/lbaw/mediabinary/-/raw/master/docs/a7_openapi.yaml](#)

Web Resources Specification (A7) for MediaLibrary
[Find more info here.](#)

Servers
http://lbaw.fe.up.pt - Production server ▾

M01: Authentication and Individual Profile ▾

GET /login R101: Login Form ▾

POST /login R102: Login Action ▾

POST /logout R103: Logout Action ▾

GET /register R104: Register Form ▾

POST /register R105: Register Action ▾

GET /users/{id} R106: View user profile ▾

M02: Works ▾

GET /api/works R202: Search Works API ▾

M03: Reviews and Wish list ▾

M04: Loans ▾

M05: User Administration and Static pages ▾

GitLab OpenAPI Rendering

M01: Authentication and Individual Profile

GET /login R101: Login Form

Provide login form. Access: PUB

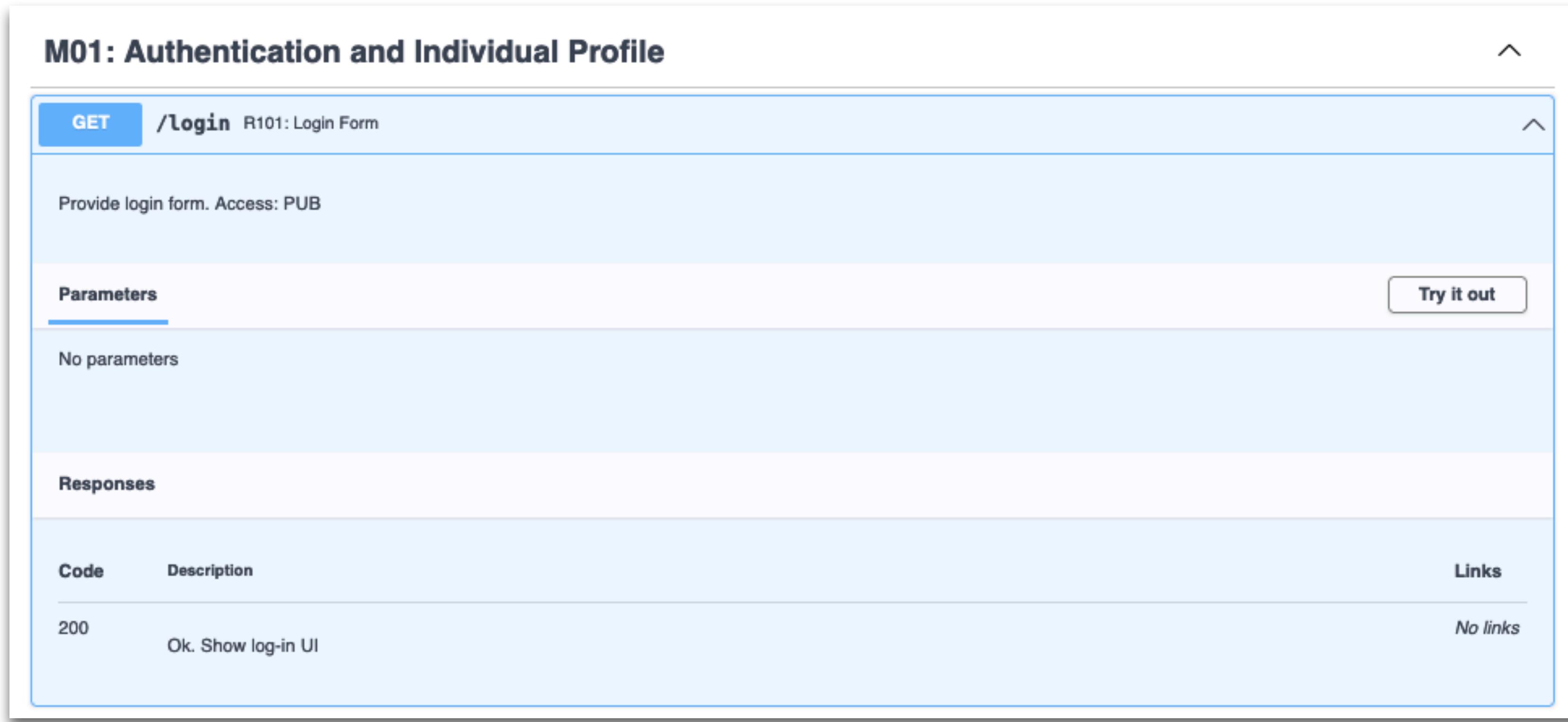
Parameters

No parameters

Responses

Code	Description	Links
200	Ok. Show log-in UI	No links

Try it out



GitLab OpenAPI Rendering

POST /login R102: Login Action

Processes the login form submission. Access: PUB

Parameters

No parameters

Request body required

application/x-www-form-urlencoded

email * required
string

password * required
string

Execute

Responses

Code	Description	Links						
302	Redirect after processing the login credentials. Headers: <table border="1"><thead><tr><th>Name</th><th>Description</th><th>Type</th></tr></thead><tbody><tr><td>Location</td><td></td><td>string</td></tr></tbody></table>	Name	Description	Type	Location		string	No links
Name	Description	Type						
Location		string						

OpenAPI References

- There is a growing collection of OpenAPI related tools
 - Data Validators
 - Documentation
 - GUI Editors
 - SDK Generators
 - Text Editors
 - ...
- Check <https://openapi.tools>

Web Development Frameworks

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

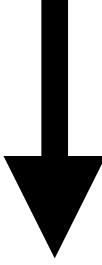
Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Software frameworks
- History of web frameworks
- Typical components in web frameworks
- Server-side frameworks

Server-Side Web Development

HTTP Request
[Method + Parameter + Payload]



Server-Side Code

validate and process request

obtain data (files, databases, ...)

process data

prepare output (HTML, JSON, ...)

send response

|
typical execution flow
↓

HTTP Response
[Code + Payload]

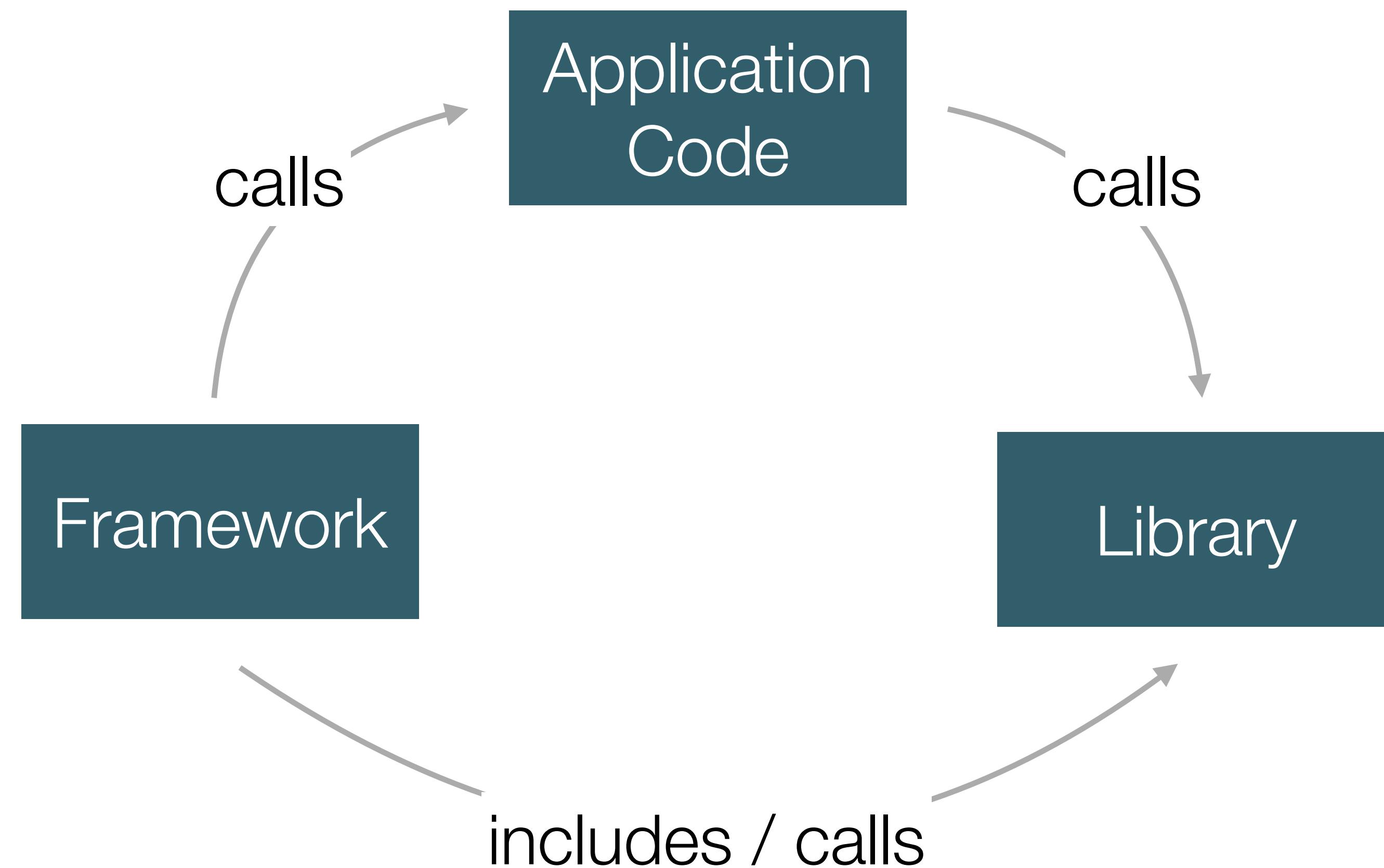


Web Development Frameworks

Software Frameworks

- Software frameworks provide a generic software foundation over which custom application-specific code can be written.
- Software frameworks often include multiple libraries, in addition to tools and rules on how to structure and use these components.
- Libraries are used by the application-specific code to support specific features.
- Frameworks control the application flow and call application-specific code.

Frameworks and Libraries



Why Frameworks

- **Advantages?**
 - Implementation speed
 - Tested, proven solutions
 - Access to expertise and off-the-shelf solutions
 - Maintenance (i.e. updates, patches)
- **Disadvantages?**
 - Reduced independence
 - Lower performance
 - Dependence on external entities
 - Technological lock-in

Choosing Frameworks

- **What to consider?**
 - Team expertise on language, libraries and framework
 - Existing code base
 - Licensing model
 - Maturity
 - Community support
 - ...

Web Development Frameworks

- Web development frameworks are designed to support the development of web applications, providing generic and integrated solutions to common use cases.
- These frameworks provide tools, libraries, and rules to address common tasks. Integration is central in contrast to the use of individual libraries.
- Currently, there is a big and diverse ecosystem of libraries and frameworks to support both backend and frontend development.
- Examples of libraries: jQuery, Bootstrap, Twig, PEAR DB.
- Examples of frameworks: ReactJS, Vue.js, Ruby on Rails, Django, Laravel.

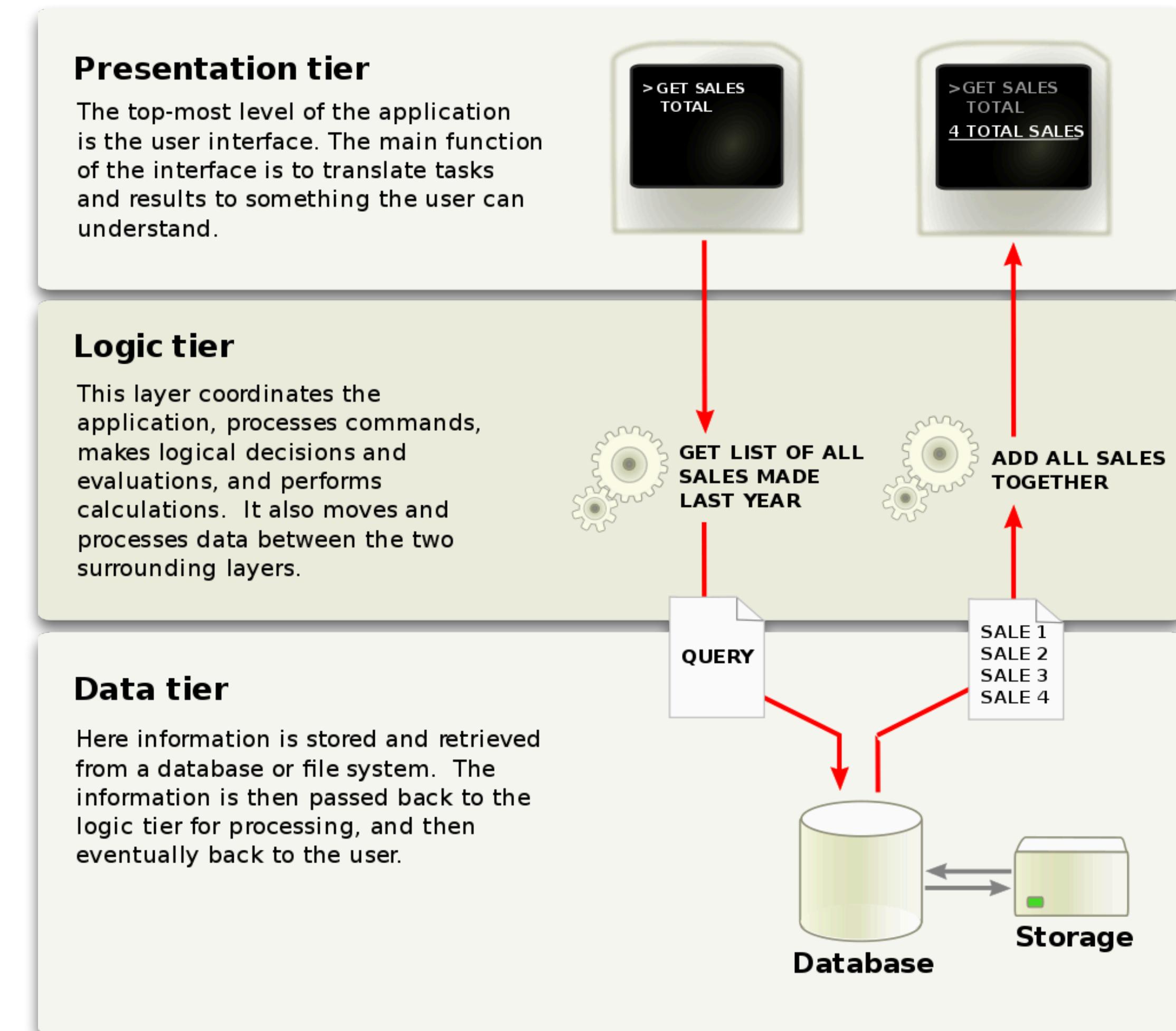
History

- In 1993, the Common Gateway Interface (CGI) was designed to enable the communication between browsers and applications, i.e. "programs as web pages". First popular web development libraries were designed to support CGI.
- During the 90s there was a strong development of libraries targeted at common use cases, e.g. outputting HTML (templating), accessing data, interface with mail, managing user input, etc.
- In the early 2000s, the first modern full-stack server-side frameworks for web development started to appear, e.g. Drupal, Ruby on Rails, Symfony, Django.
- The growth of client-side supported web applications led to the development of multiple frontend libraries, e.g. jQuery, Mustache.
- Recently, full-stack client-side frameworks for web development emerged, e.g. ReactJS, AngularJS, Vue.js.

Three-tier Architectures

- The Three-tier Architecture is a software architecture pattern commonly adopted in web applications.
- **Presentation tier**, interface with the user; process user interactions; present views to the user.
- **Logic tier**, coordinate the application; decide on the application flow; process data; move data between the two other layers.
- **Data tier**, manage information; persist information; handle consistency; translate between physical models and conceptual model.

Three-tier Architecture



Source: https://en.wikipedia.org/wiki/Multitier_architecture

Common Problems in Web Development

- **Common problems in web applications development?**

- Handle access requests.
- Manage user interface components.
- Access and manipulate data.
- Session and authentication management.
- Access control management.
- Validating inputs.
- Error handling.
- Interacting with email systems.
- ...

Server-side Frameworks

- Frameworks can be grouped in three types.
- **Micro Frameworks**, focused on routing HTTP request to a callback, commonly used to implement HTTP APIs.
- **Full-Stack Frameworks**, feature-full frameworks that includes routing, templating, data access and mapping, plus many more packages.
- **Component Frameworks**, collections of specialized and single-purpose libraries that can be used together to make a micro- or full-stack framework.

Framework Components

- Core components
 - Request Routing, match incoming HTTP requests to code.
 - Template Engine, structure and separate presentation from logic.
 - Data Access, uniform data access, mapping and configuration.
- Common components
 - Security, protection against common web security attacks.
 - Sessions, session management and configuration.
 - Error Handling, capture and manage application-level errors.
 - Scaffolding, quickly generate CRUD interfaces based on data model.
 - ...

Request Routing

- Request routing maps HTTP access requests to specific functions.
- URL design is handled independently from application code using request routing.
- Clean URLs (aka "friendly URLs") are an important part of a web application: usability, SEO, technology independence, etc.
- https://sigarra.up.pt/feup/pt/cur_geral.cur_view?pv_curso_id=742
- <https://sigarra.up.pt/feup/pt/curso/mieic>

Laravel Routing Example

```
Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) {  
    //  
});
```

```
Route::get('/user/{name}', function ($name) {  
    //  
})->where('name', '[A-Za-z]+');  
  
Route::get('/user/{id}', function ($id) {  
    //  
})->where('id', '[0-9]+');  
  
Route::get('/user/{id}/{name}', function ($id, $name) {  
    //  
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

```
Route::get('/user/{id}', function ($id) {  
    return 'User ' . $id;  
});
```

```
Route::get('/user/{name?}', function ($name = null) {  
    return $name;  
});
```

Template Engines

- Many of the first libraries for web development were template engines.
- Focused on the separation between presentation code and logic code.
- There are many independent libraries. Frameworks either use existing libraries or develop their own system.
- Notable solutions: Smarty (PHP), Blade (PHP), Jinja (Python), mustache (*).

Laravel Templating Example (Blade)

```
<html>
  <head>
    <title>App Name - @yield('title')</title>
  </head>
  <body>
    @section('sidebar')
      This is the master sidebar.
    @show

    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

```
@if (count($records) === 1)
  I have one record!
@elseif (count($records) > 1)
  I have multiple records!
@else
  I don't have any records!
@endif
```

```
Route::get('/', function () {
  return view('greeting', [ 'name' => 'Finn' ]);
});
```

Data Access

- The data access layer can be managed with different levels of automatism and control.
- Data layer independence can be achieved using libraries that provide a uniform access to different technologies. Example: PHP PDO.
- A higher level of coupling can be achieved by providing access to data through a mapping between the underlying data structures and an object layer (ORM). Example: ActiveRecord (Laravel, RoR).
- This coupling between the data layer and the application's data model can imply database migrations.

Laravel ORM Example (Eloquent)

```
$user = DB::table('users')->where('name', 'John')->first();

return $user->email;
```

```
$count = Flight::where('active', 1)->count();

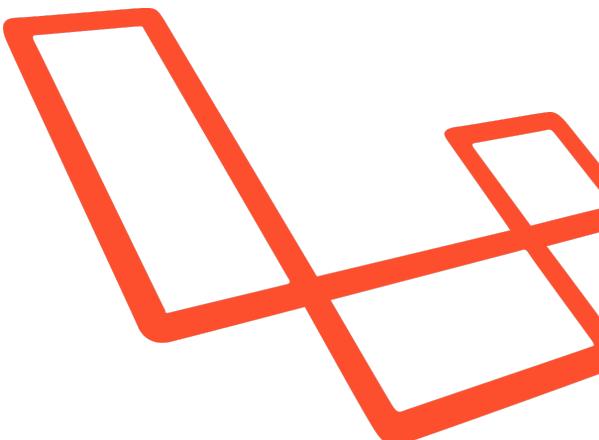
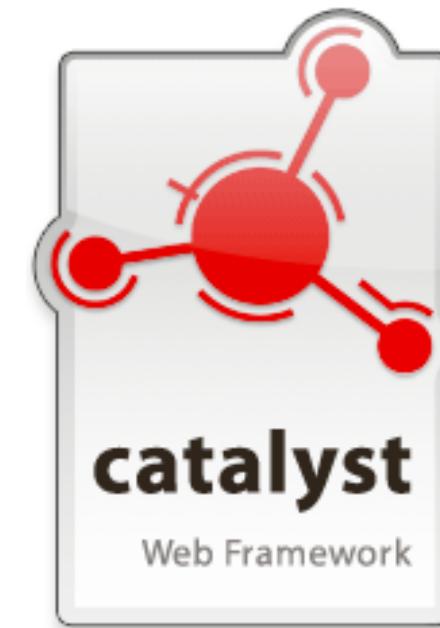
$max = Flight::where('active', 1)->max('price');
```

```
$users = DB::table('users')
    ->select(DB::raw('count(*) as user_count, status'))
    ->where('status', '<>', 1)
    ->groupBy('status')
    ->get();
```

Notable Web Frameworks

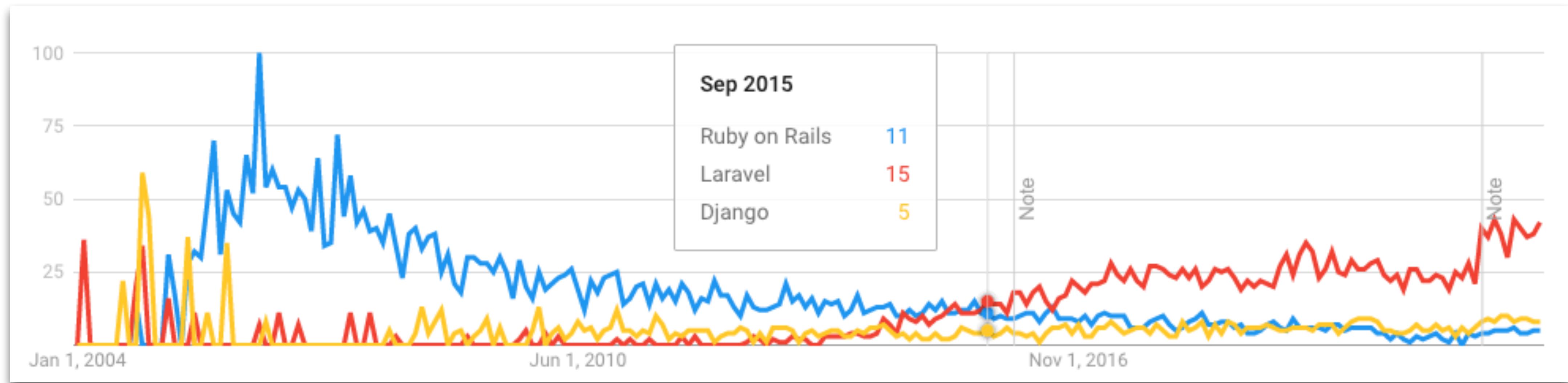


django



https://en.wikipedia.org/wiki/Comparison_of_web_frameworks

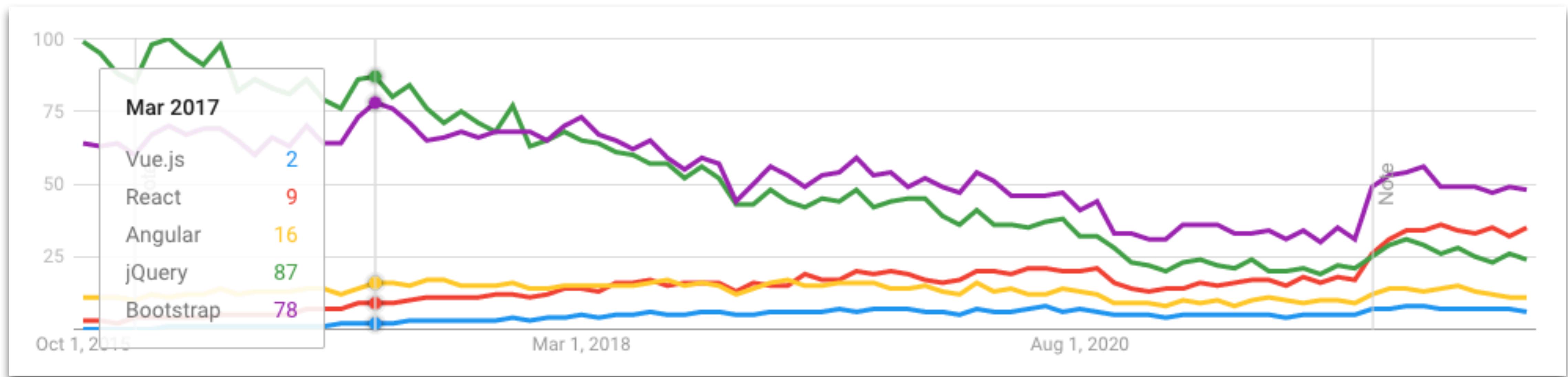
Trends



Client-side Frameworks

- Client-based web applications, rely on frontend technologies to manage both the Presentation and the Logic layers. In a nutshell, the current web page is dynamically manipulated instead of loading a new page.
- Single-page web applications (SPA) are an extreme example of this architecture. All application views are mapped to a single document.
- Frontend web frameworks typically adopt an MVC pattern (or variants) to support this architectural style, including: data-bindings, templates, routing.
- Most notable solutions: AngularJS (Google), ReactJS (Facebook), Vue.js.

Trends



Laravel

Laravel Overview

- A free open-source PHP backend web framework first released in 2011.
- Version 9.x is the current stable version (Feb, 2022).
- Follows the model-view-controller architectural pattern.
- Strong community and documentation.
 - laravel.com
 - laracasts.com

Laravel Concepts

- **Routes**, define how requests for URLs are handled.
- **Models**, using Eloquent ORM, tables are mapped to models supporting insert, update and delete data.
- **Views**, using Blade template engine, views are defined for presentation.
- **Artisan** is Laravel's command line interface (CLI),
 - E.g., create new application, create model skeleton, serve application, etc.

Laravel in LBAW

- template-laravel includes a sample app - Thingy!
 - <https://git.fe.up.pt/lbaw/template-laravel>
- A8: Quickstart guide describes how to go from,
 - wireframes (A3) +
 - database (A6) +
 - web architecture (A7),
- to a working vertical prototype.

Quickstart guide workflow

- Model > Controllers > Routes > Blade
- Start by creating the models to interact with the database.
 - Typically one model per each table.
- Define the associations between models.
- Implement the controllers logic (application logic, validation, authorization, etc).
- Define the web resources endpoints.
- Implement the HTML views using Blade.

Other Topics

- Content Management Systems, e.g. WordPress.
- Static-site generators.
- ...

References

- MDN, Server-side web frameworks,
[developer.mozilla.org/docs/Learn/Server-side/First steps/Web frameworks](https://developer.mozilla.org/docs/Learn/Server-side/First_steps/Web_frameworks)
- MDN, Understanding client-side JavaScript frameworks,
[developer.mozilla.org/docs/Learn/Tools and testing/Client-side JavaScript frameworks](https://developer.mozilla.org/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks)
-

A8 - Vertical Prototype

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

EAP: Architecture Specification and Prototype

- This component groups artifacts related to the high-level architecture specification of the information system to be developed and the vertical prototype implemented to validate the architecture.
- **A7: Web Resources Specification**
This artifact presents an overview of the web resources to implement, organized into modules. It also includes the permissions used in the modules to establish the conditions of access to resources.
- **A8: Vertical Prototype**
Includes the implementation of the features marked as necessary (with an asterisk) in the common and theme requirements documents. This artifact aims to validate the architecture presented, also serving to gain familiarity with the technologies used in the project.

A8: Vertical Prototype

A8: Vertical Prototype

- The A8 artifact:
 - corresponds to the implementation of the high priority user stories (features with *);
 - is used to validate the architecture presented;
 - also serves to gain familiarity with the technologies used in the project.
- It must:
 - be based on the LBAW Framework and
 - include work on all layers of the architecture: user interface, business logic and data access.
- The LBAW Framework (template-laravel) includes an authentication system that must be adapted by each group.
- The user stories must include at least a form, an action, an AJAX request, search, and update to the database.

A8: Vertical Prototype

- The vertical prototype must be based on the LBAW Framework.
- PostgreSQL for data persistence.
- Laravel for server-side development.
- HTML, CSS and JavaScript for frontend development.
- Docker for deployment of the product as a Docker container.

LBAW Computational Setup

PostgreSQL

- PostgreSQL is the relational database management system adopted.
- Groups have a production database at [db.fe.up.pt](#).
- For development, a local PostgreSQL can be setup using Docker.
- SQL is managed using the group's GitLab repository.
- Do not create or alter the database using the graphical interface.
- Both the Vertical Prototype and the Product must work on the [db.fe.up.pt](#).

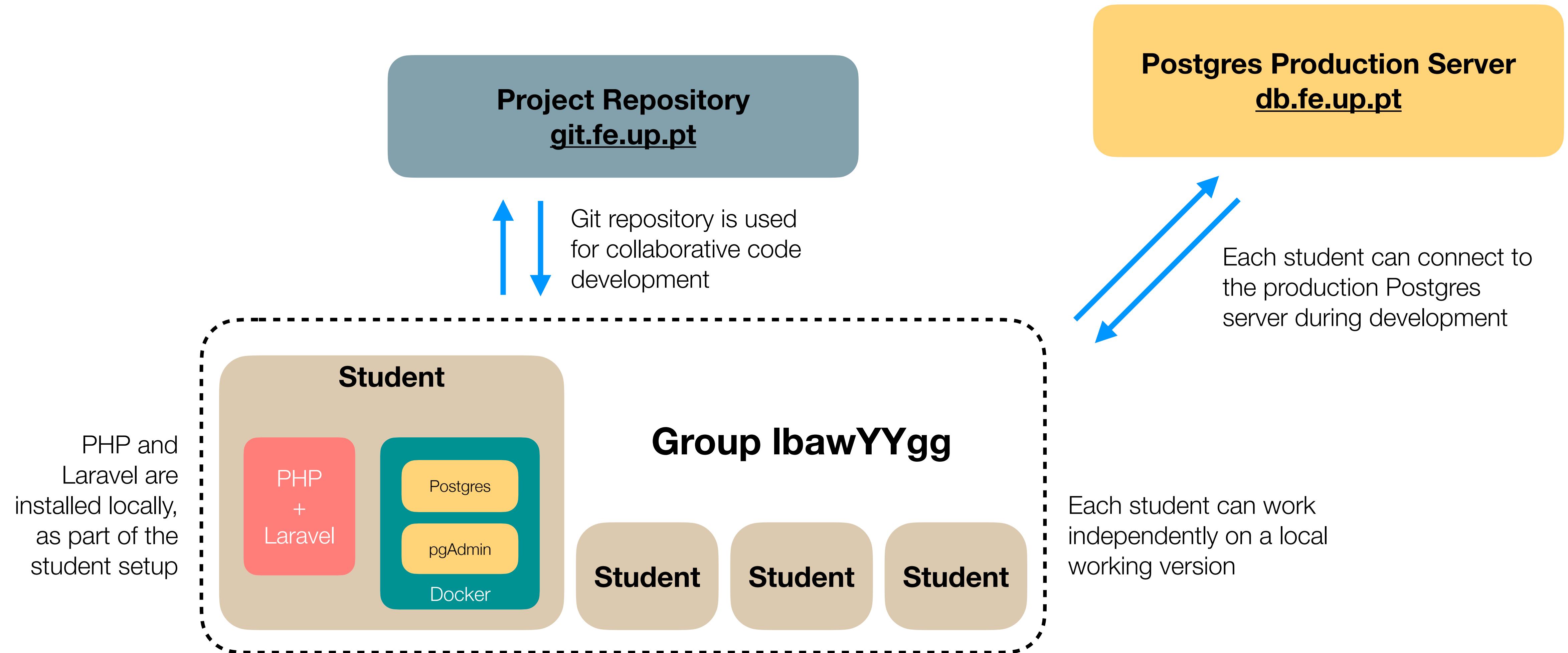
Deploying the Project

- Docker is used to deploy the application, both for
 - the Vertical Prototype (A8)
 - the final Product (A9)
- Docker is used for
 - Local setup of PostgreSQL and pgAdmin
 - Publishing the application at lbaw.fe.up.pt via Gitlab Container Registry
- Each group builds a Docker image of the application and publishes using Gitlab's Container Registry.
- The images are regularly pulled to lbaw.fe.up.pt and available at lbaw22gg.lbaw.fe.up.pt.
- Section “Publishing your image” in template-laravel guide.

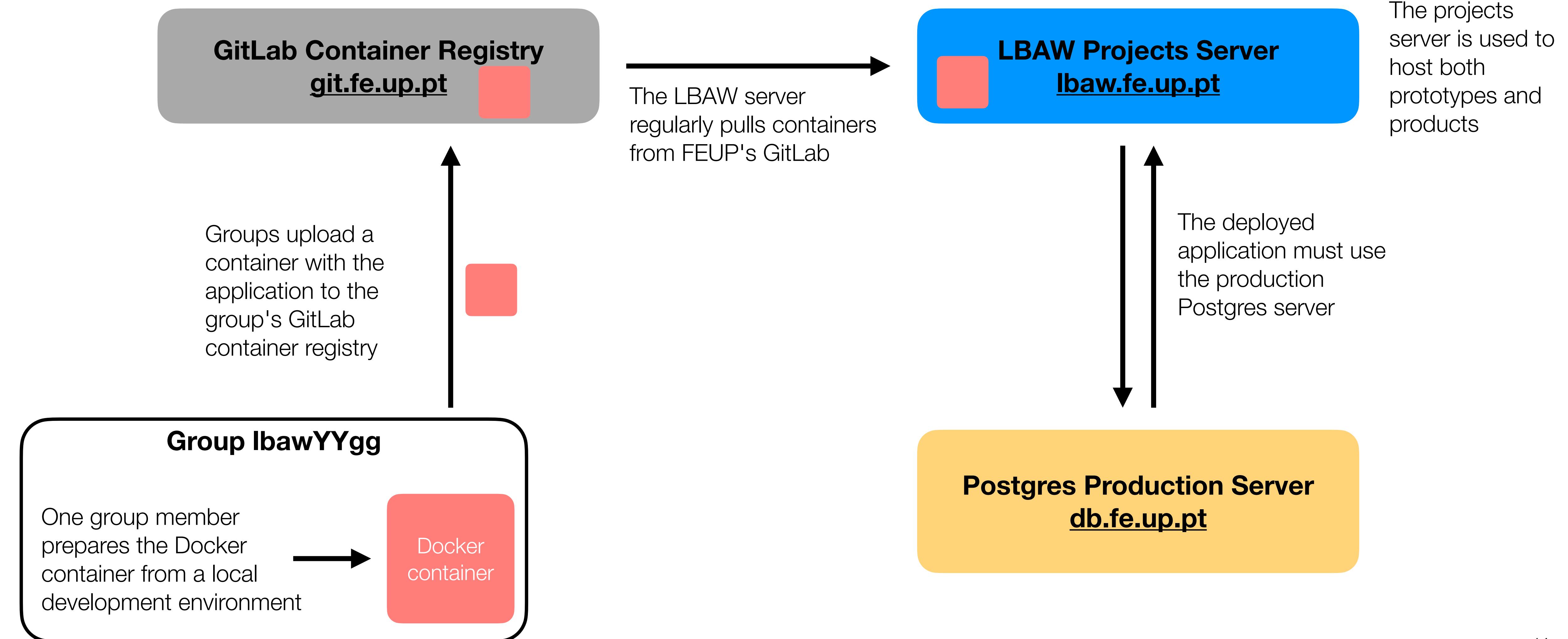
Laravel for Server-Side Web Development

- The group develops a web application using the Laravel server framework.
- The group starts the project by copying the files available at the template repository (git.fe.up.pt/lbaw/template-laravel) to their own repository, and then follow the instructions detailed in the README to set up the development requirements.
- The project uses the PostgreSQL database management system. A local instance is created using the bundled Docker compose file (see the README for the required steps to start it).
- The group should keep an up to date Docker image of their project in the group's Gitlab Container Registry (see the template README for the required steps).
- There is a practical guide available to help you build your prototype using the work done in the previous artifacts, namely A3, A6, and A7, see [LBAW A8: Putting it all together](#).

Computational Setup for Development



Computational Setup for Deployment



Summary

- Each student has a local Postgres (Docker) and Laravel (local) installation.
- A GitLab repository is used for code development.
- Database deployment is done on the production server at [db.fe.up.pt](#).
- Web application deployment is done
 - Preparing a Docker container with the web application;
 - Uploading the container to a group's Gitlab Container Registry;
 - Wait for the server to pull it and show at [lbaw22gg.lbaw.fe.up.pt](#).

Resources

- A repository with a Laravel setup and instructions to use as a starting point
 - <https://git.fe.up.pt/lbaw/template-laravel>
 - This repository includes a demo application — Thingy!
 - A pre-recorded video with a presentation about the repository and development of the demo application, see Panopto.
- A practical guide to help you build your prototype using the work done in the previous artifacts (A3, A6, A7), see "LBAW A8: Putting it all together".
- A8 MediaLibrary example.
- A8 Checklist.
- Next monitor session will be focused on developing the Vertical Prototype.

MediaLibrary Example

A8 MediaLibrary - Implemented Features

1. Implemented Features

1.1. Implemented User Stories

The user stories that were implemented in the prototype are described in the following table.

User Story	Name	Priority	Description
US01	Sign-in	high	As a <i>Visitor</i> , I want to authenticate into the system, so that I can access privileged information
US02	Sign-up	high	As a <i>Visitor</i> , I want to register myself into the system, so that I can afterwards authenticate myself
US11	Home page	high	As a <i>User</i> , I want to access the home page, so that I can see a brief presentation of the website
US12	About page	high	As a <i>User</i> , I want to access the about page, so that I can see who is responsible for the

A8 MediaLibrary - Implemented Web Resources

1.2. Implemented Web Resources

The web resources that were implemented in the prototype are described in the next section.

Module M01: Authentication and Individual Profile

Web Resource Reference	URL
R101: Login Form	GET /login
R102: Login Action	POST /login
R103: Logout Action	POST /logout
R104: Register Form	GET /register
R105: Register Action	POST /register
R106: View Profile	GET /users/{id}
R107: Edit Profile Form	GET /users/{id}/edit
R108: Edit Profile Action	POST /users/{id}/edit
R109: Password Reset Form	GET /password/reset
R110: Password Reset Action	POST /password/reset

A8 MediaLibrary - Prototype

2. Prototype

The prototype is available at <http://medialibrary.lbaw.fe.up.pt/>

Credentials:

- admin user: admin@fe.up.pt/password
- regular user: userx@fe.up.pt/password

The code is available at <https://git.fe.up.pt/lbaw/medialibrary/tree/proto>

A8 Checklist

A8 Checklist - Implemented Features

Implemented features	2.1	Implemented features section is included
	2.2	Authentication is implemented
	2.3	Logout is implemented
	2.4	Features marked as necessary for the vertical prototype are implemented
	2.5	References to the implemented user stories are included
	2.6	Access features are implemented
	2.7	Creation features are implemented
	2.8	Update features are implemented
	2.9	Delete features are implemented
	2.10	AJAX and API features are implemented
	2.11	Permissions control using Policies is implemented
	2.12	Feedback messages (e.g. errors) are implemented

A8 Checklist - Code and Prototype

Code quality	3.1	Source code repository is updated
	3.3	The LBAW framework is used
	3.4	All non-essential LBAW template code was removed (e.g. Thingy! code)
	3.5	No additional libraries or tools are used
	3.6	Laravel routes are correctly used
	3.7	Laravel controllers are correctly used
	3.8	Laravel templates are correctly used
	3.9	Laravel data access is correctly used
Prototype access	4.1	Prototype URL is included
	4.2	Prototype is online and working at the production machine
	4.3	User credentials for testing are provided

Laravel

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Laravel Overview
- Laravel Setup
- Routing and Controllers
- Blade Templating
- Database and Eloquent
- Laravel at LBAW

Laravel Overview

Frameworks

- Libraries and packages handle isolated components.
- A framework groups a collection of components together with settings and configurations, directory structure, code templates and skeletons, etc.
- Frameworks include decisions not only on which components but on how these components work together.
- Alternative: decide on each individual component; setup on how they work together; maintain component's evolution.
- Frameworks bring tested configurations, consistency, robustness, communities.
- Frameworks also bring commitment, reduced freedom, vendor lock-in, third-party dependency.

History of PHP Frameworks

- Ruby on Rails (2004) is a landmark in the development of web application frameworks. Popular concepts introduced by RoR include: MVC, RESTful JSON APIs, convention over configuration, Active-Record.
- Notable PHP web frameworks:
 - CakePHP (2005), cakephp.org
 - Symfony (2005), symfony.com
 - CodeIgniter (2006), codeigniter.com
 - Slim (2010), slimframework.com
 - Laravel (2011), laravel.com

Laravel Highlights

- Laravel is a backend web development framework, highlighting:
- **Rapid application development framework**, i.e. fast learning curve, support for most common tasks, consistent API, predictable structure, large tools and packages ecosystem.
- **Convention over configuration**, i.e. default configurations out of the box.
- **Simplicity**, i.e. start simple approach philosophy, and then bring more complex solutions if needed; use of simpler PHP syntax and coding practices.

Laravel Community

- Laravel has a strong and active community.
- A large number of open and freely available resources.
- Official documentation, laravel.com/docs
- From the community
 - Laracasts, laracasts.com
 - Laravel News, laravel-news.com
 - Laravel Podcast, laravelpodcast.com
 - Discussion Forums, laracasts.com/discuss
- Awesome Laravel, github.com/chiraggude/awesome-laravel

Laravel Origins: A PHP Documentary



<https://www.youtube.com/watch?v=127ng7botO4>

Laravel Setup

PHP Composer

- getcomposer.org
- Composer is PHP's package and dependency manager.
- It works on a per-project basis, i.e. there are no global installations.
- Packages are installed in a 'vendor' directory inside the project.
- Declare which packages are needed for a project
- Composer finds and downloads package versions and dependencies that need to be installed.
- Packages updates are managed with the update command.

Composer Basic Usage

- Composer uses a JSON file — composer.json — per-project to manage its dependencies.
- Best practice is to build the **composer.json** file using composer.
- Packages can be found at Packagist, packagist.org
- Example: install phpunit, PHP unit testing framework:
 - `composer require phpunit/phpunit`
 - A composer.json file will be created (or edited) and the package will be downloaded.
- A **composer.lock** file is used to fix the versions used in a project.
 - composer install will not get the latest versions but the ones specified in composer.lock.
 - composer.lock should be committed to the version control system.

Laravel “Hello World”

Laravel “Hello World” Example

- Build a simple Laravel web application that shows posts stored in a SQLite database.
- Receive a GET request at /posts/{slug}.
- Read post information from the SQLite database.
- Present the result as an HTML page.

Install Laravel 9.x

- Requirements:
 - PHP (>= 8.0)
 - Composer, getcomposer.org
- Use Composer to start a new Laravel project
 - `composer create-project laravel/laravel example-app`
- Change to the ‘example-app’ directory
 - `cd example-app`
- Start a local web server (using PHP’s or Laravel’s):
 - `php -S localhost:8000 -t public`
 - `php artisan serve`
- View at <http://localhost:8000>

Database Schema

- database/example-app.sql

```
DROP TABLE IF EXISTS posts;

CREATE TABLE posts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    date DATE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    slug TEXT NOT NULL UNIQUE,
    title TEXT NOT NULL,
    body TEXT
);

INSERT INTO posts (slug, title, body) VALUES ('hello-world', 'Hello world!', 'This is a first text.');
INSERT INTO posts (slug, title, body) VALUES ('test', 'Testing', 'We are testing the system.');
INSERT INTO posts (slug, title, body) VALUES ('fox', 'Fox?', 'The quick brown fox jumps over the lazy dog.');
```

Create Database

- Define the SQL schema (in previous slide)
 - `database/example-app.sql`
- Create a new SQLite database
 - `touch database/example-app.sqlite`
- Create the database schema from the SQL file.
 - `sqlite3 database/example-app.sqlite < database/example-app.sql`

Database Connection

- Setup database access in the environment configuration file
 - `.env`
- Remove previous DB_* settings and add:
 - `DB_CONNECTION=sqlite`
 - `DB_DATABASE={ full path }/database/example-app.sqlite`
- To test the database connection you can open a CLI to the database:
 - `php artisan db`
- And then, in SQLite, run:
 - `.tables`

Automate Database Creation (1)

- Laravel's seeding functions can be used to create the database from a SQL file.
- Change [[database/seeders/DatabaseSeeder.php](#)]

```
<?php

namespace Database\Seeders;

// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $path = 'database/example-app.sql';
        DB::unprepared(file_get_contents($path));
        $this->command->info('Database seeded!');
    }
}
```

Automate Database Creation (2)

- The database can be dropped with:
 - `php artisan db:wipe`
- And re-created with:
 - `php artisan db:seed`

Models

- Laravel models control database access.
- Create a Laravel model for ‘posts’ using Artisan
 - `php artisan make:model Post`
- A new file will be created at
 - `app/Models/Post.php`
- Disable automatic timestamps by editing the model file.

```
class Post extends Model
{
    use HasFactory;

    // Don't add create and update timestamps in database.
    public $timestamps = false;
}
```

Models Usage

- Models can be used to interact with the database.
- Test the ‘Post’ model with Artisan’s tinker tool (CLI).
 - `php artisan tinker`

```
>>> App\Models\Post::find(1);
=> App\Models\Post {#4626
    id: 1,
    date: "2022-11-03 19:03:07",
    slug: "hello-world",
    title: "Hello world!",
    body: "This is a first text.",
}

>>> App\Models\Post::find(1)->slug;
=> "hello-world"
```

```
>>> App\Models\Post::all()
=> Illuminate\Database\Eloquent\Collection {#4402
    all: [
        App\Models\Post {#4360
            id: 1,
            date: "2022-11-03 19:03:07",
            slug: "hello-world",
            title: "Hello world!",
            body: "This is a first text.",
        },
        App\Models\Post {#4629
            id: 2,
            date: "2022-11-03 19:03:07",
            slug: "test",
            title: "Testing",
            body: "We are testing the system.",
        },
        App\Models\Post {#4628
            id: 3,
            date: "2022-11-03 19:03:07",
            slug: "fox",
            title: "Fox?",
            body: "The quick brown fox jumps over the lazy dog.",
        },
    ],
}
```

Controllers

- Laravel controllers are where each resource logic is implemented.
- To create a controller for the Post model use:
 - `php artisan make:controller PostController`
- A new file will be created at:
 - `app/Http/Controllers/PostController.php`

Post Controller

- The Post controller defines a show function that receives a ‘slug’ and:
 - If found, show the associated post;
 - If not, show a not found message.
- [app/Http/Controllers/PostController.php](#)

```
namespace App\Http\Controllers;

use App\Models\Post;
use Illuminate\Http\Request;

use Illuminate\Database\Eloquent\ModelNotFoundException;

class PostController extends Controller
{
    /**
     * Show the Post for a given slug.
     *
     * @param  $slug
     * @return \Illuminate\Http\Response
     */
    public function show($slug)
    {
        try {
            // Show the 'posts.show' view based on the slug in the route.
            return view('posts.show', [
                'post' => Post::where('slug', $slug)->firstOrFail()
            ]);
        }
        catch(ModelNotFoundException $e) {
            // TBD: redirect to a search page based on the not found slug text.
            return "Post not found.";
        }
    }
}
```

Views

- Laravel views are used to define the user interface.
- Views are written in Blade and defined under:
 - `resources/views`
- To create a new post.show view, create a new folder:
 - `resources/views/posts`
- And inside create a new file:
 - `resources/views/posts/show.blade.php`
- This view is called from the controller with:
 - `return view('posts.show', ['post' => ...]);`

Post View

- Laravel views are written in Blade.
- `resources/views/posts/show.blade.php`

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ $post->title }}</title>
</head>

<body>
    <h1>{{ $post->title }}</h1>
    <p>{{ $post->body }}</p>
</body>

</html>
```

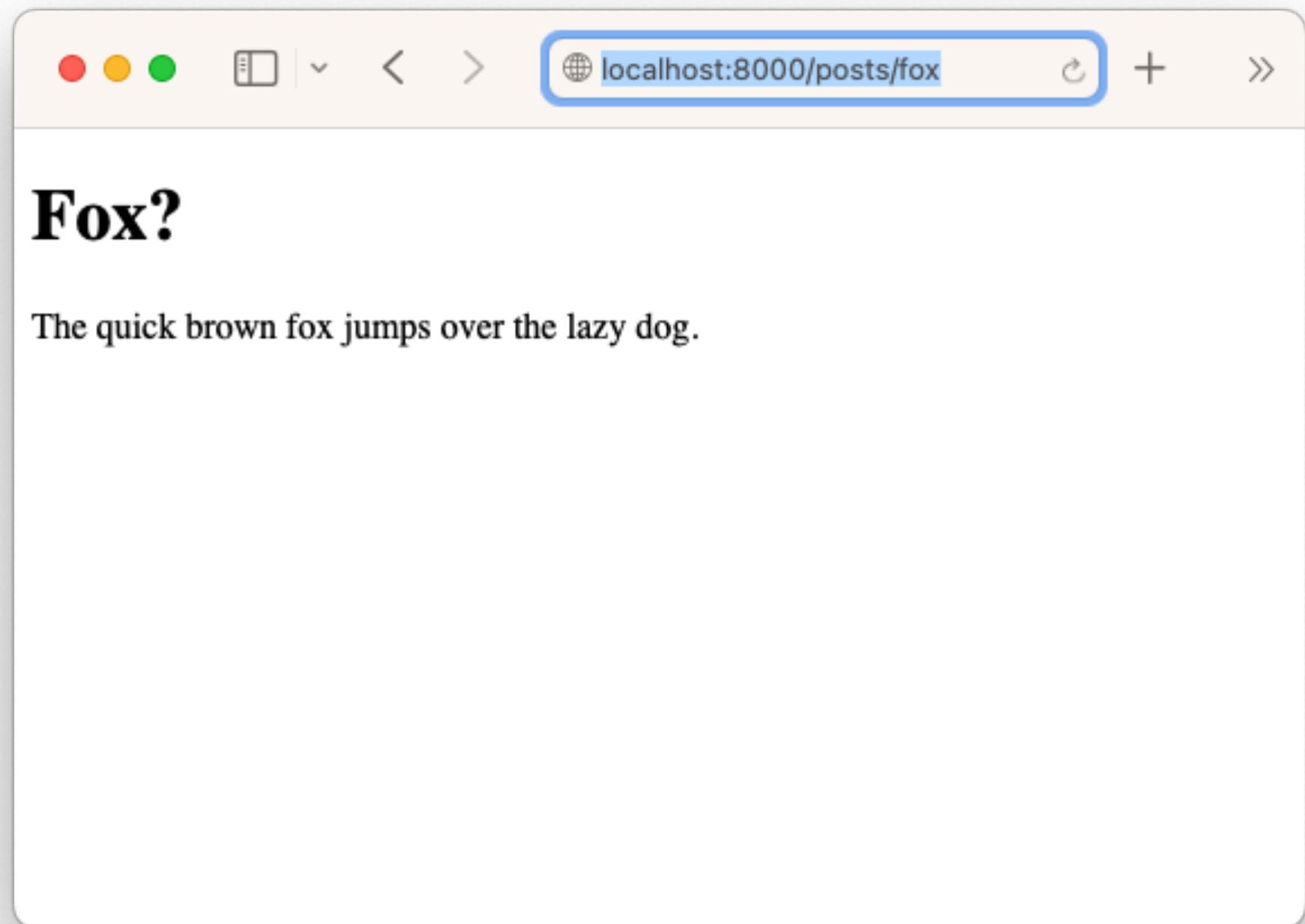
Routes

- Laravel routes define how web requests are processed.
- Web routes are defined in the file:
 - `routes/web.php`
- Create a new route for a GET /posts/{slug} to be handled by the show function in the PostController.

```
use App\Http\Controllers\PostController;  
  
Route::get('posts/{slug}', [PostController::class, 'show']);
```

Test

- Start Laravel's built-in web server:
 - `php artisan serve`
- Test URLs
 - localhost:8000/posts/fox
 - localhost:8000/posts/not



```
php artisan serve
INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server

2022-11-04 09:50:14 ...
2022-11-04 09:50:19 ...
2022-11-04 09:50:27 ...
2022-11-04 09:50:28 /favicon.ico ...
2022-11-04 09:50:31 ...
2022-11-04 09:51:01 ...
2022-11-04 09:51:02 ...
2022-11-04 09:51:02 ...
2022-11-04 09:51:02 ...
2022-11-04 09:51:03 ...
2022-11-04 09:51:32 ...
2022-11-04 10:24:53 ...
2022-11-04 10:24:54 /favicon.ico ...
2022-11-04 10:25:12 ...
2022-11-04 10:25:13 ...
2022-11-04 10:25:13 ...
2022-11-04 10:25:13 /favicon.ico ...
```

Laravel Details

Configuration

- Configuration files are stored in the `config` directory.
- Current configuration options can be shown with:
 - `php artisan about`
- The `.env` file can be used to set up configurations that differ between environments, e.g. local development, production server.
 - Typical configurations: application settings (name, debug, url, ...), database (host, password, ...).
 - `APP_DEBUG` is used to enable or disable debugging mode.

Directory Structure

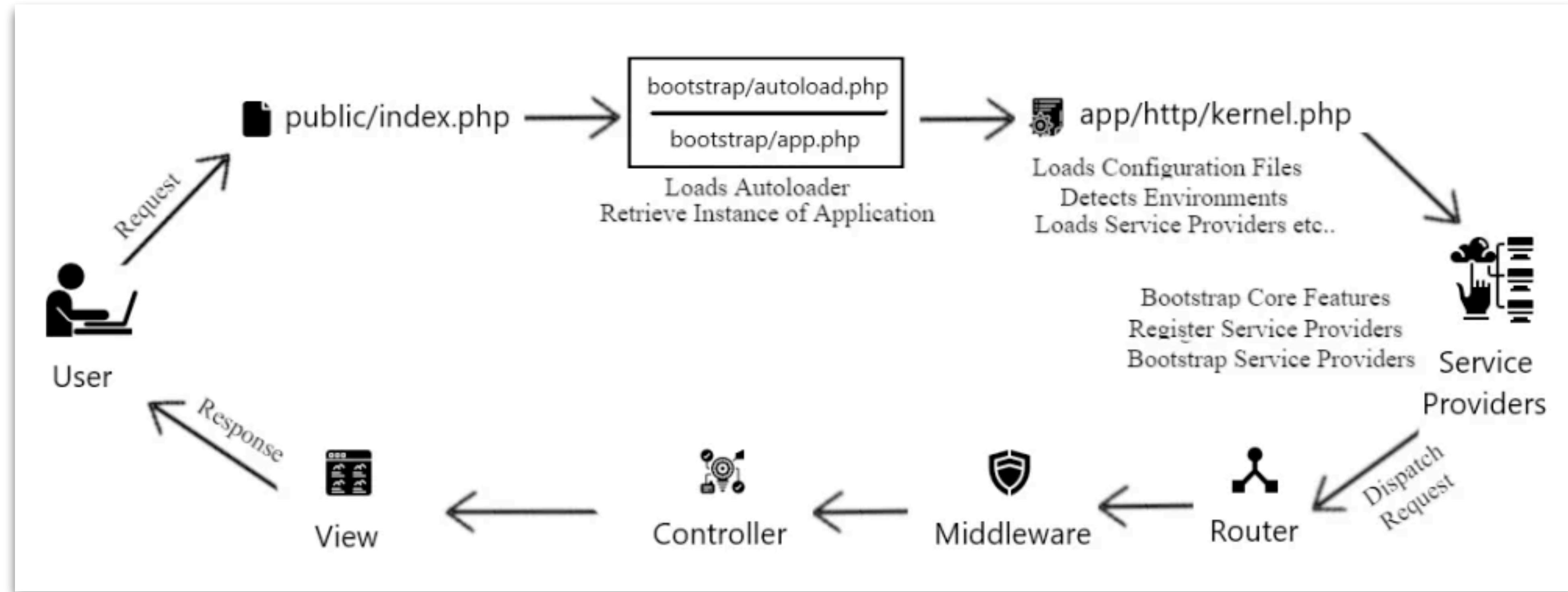
- **app**, core code of the application, including models and controllers.
- **bootstrap**, framework bootstrapping files, namely app.php, and a cache directory.
- **config**, configuration files and settings.
- **database**, database migrations, model factories, and seeds.
- **lang**, language files.
- **public**, contains index.php, the entry point for all requests, and assets (images, CSS, JavaScript).
- **resources**, contains views and uncompiled assets (CSS, JavaScript).
- **routes**, all routes definitions, including web and api routes.
- **storage**, caches, logs, and compiled Blade templates.
- **tests**, automated unit and integrated tests.
- **vendor**, composer dependencies.

```
.  
├── README.md  
├── artisan  
├── composer.json  
├── composer.lock  
├── package.json  
├── phpunit.xml  
├── vite.config.js  
├── app  
├── bootstrap  
├── config  
├── database  
├── lang  
├── public  
├── resources  
├── routes  
├── storage  
├── tests  
└── vendor
```

Request Lifecycle

- All requests are first handled by **public/index.php**
- The first step is to obtain an instance of the Laravel application.
- The incoming request is sent to a **kernel**, typically the HTTP kernel (handles web requests).
 - Configure error handling, logging, set up the app environment,
- The request goes through a set of **middleware** (multilayers) for session handling, security, etc.
- The request then moves through **service providers**, which bootstrap various components, e.g. database, validation, routing.
 - An important service provider is the Route Service Provider, which manages the routes, and dispatches requests to the specific controller methods.
- The response sent by the controller goes back to the route's middleware, and is finally sent to the client.

Request Lifecycle



Request and Response Objects

- You can inspect the request and response objects with specific methods.

```
Route::get('test', function () {
    dump(request());
    dump(response());
});
```

```
Illuminate\Http\Request {#43 ▼ // routes/web.php:25
  +attributes: Symfony...\ParameterBag {#45 ▶}
  +request: Symfony...\InputBag {#44 ▶}
  +query: Symfony...\InputBag {#51 ▶}
  +server: Symfony...\ServerBag {#47 ▶}
  +files: Symfony...\FileBag {#48 ▶}
  +cookies: Symfony...\InputBag {#46 ▶}
  +headers: Symfony...\HeaderBag {#49 ▶}
  #content: null
  #languages: null
  #Charsets: null
  #encodings: null
  #acceptableContentTypes: null
  #pathInfo: "/test"
  #requestUri: "/test"
  #baseUrl: ""
  #basePath: null
  #method: "GET"
  #format: null
  #session: Illuminate\Store {#266 ▶}
  #locale: null
  #defaultLocale: "en"
  -preferredFormat: null
  -isHostValid: true
  -isForwardedValid: true
  #json: null
  #convertedFiles: null
  #userResolver: Closure($guard = null) {#231 ▶}
  #routeResolver: Closure() {#240 ▶}
  basePath: ""
  format: "html"
}
```

```
Illuminate\Routing\ResponseFactory {#278 ▼ // routes/web.php:26
  #view: Illuminate\Factory {#275 ▶}
  #redirector: Illuminate\Redirector {#302 ▶}
}
```

Routing and Controllers

Route Definitions

- Web routes are defined in `routes/web.php`
- API routes are defined in `routes/api.php`
- Currently defined routes can be listed with:
 - `php artisan route:list`

- Two basic route definitions.
 - GET /
 - GET /about

```
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    // Return a string.
    return 'Hello, World!';
});

Route::get('/about', function () {
    // Return the 'about' Blade view.
    return view('about');
});
```

Route Methods

- The router allows the registration of routes responding to any HTTP verb.

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

- Also allows the registration of routes that match multiple HTTP verbs (any, match).

```
Route::match(['get', 'post'], '/', function () {
    //
});

Route::any('/', function () {
    //
});
```

Redirect Routes

- The redirect method can be used to define a route redirection.
- By default, the redirect method uses a 302 status code.
- A specific HTTP status code can be defined as the third optional argument.

```
// By default a 302 status code is used.  
Route::redirect('/here', '/there');  
  
// The specific HTTP code can be defined.  
Route::redirect('/here', '/there', 301);  
  
// This returns a 301 status code.  
Route::permanentRedirect('/here', '/there');
```

View Routes

- The view method provides a shortcut for simple view routes.
- The first argument is the URI of the route.
- The second argument is the view to use.
- The third, optional, argument is an array of data to pass to the view.

```
Route::view('/about', 'about');
```

```
Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

Route Parameters

- Route parameters allow the definition of variable segments within a URI.
- Multiple route parameters can be required in a single route.
- Parameters can be defined as optional with a ?.

```
// Route parameters are defined with curly brackets.  
Route::get('/user/{id}', function ($id) { ... });  
  
// Multiple parameters can be used in a single route.  
Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) { ... });  
  
// Parameters can be set as optional.  
Route::get('/user/{name?}', function ($name = 'John') { ... });
```

Route Regular Expressions

- Regular expressions can be defined to establish constraints in routes.
- Requests will only be handled by the route if they match the pattern.
- If no matches are found, a 404 HTTP response is returned.

```
Route::get('/user/{name}', function ($name) {
    //
})->where('name', '[A-Za-z]+');

Route::get('/user/{id}', function ($id) {
    //
})->where('id', '[0-9]+');

Route::get('/user/{id}/{name}', function ($id, $name) {
    //
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Named Routes

- Named routes are convenient for generating URLs and handling redirects.
- A name can be specified for a route by chaining the name() method onto the route.
- Names can be used in redirects.

```
// Name a route as 'profile'.
Route::get('/user/profile', function () {
    //
})->name('profile');

// Generate a redirect to 'profile' route.
return redirect()->route('profile');
```

Route Behavior

- Route behavior can be defined with a closure (i.e. anonymous function).

```
Route::get('/greeting', function () {
    return 'Hello World';
});
```

- The **standard approach** is to handle it through a controller method.
- Routes can be grouped to share the same controller.

```
// Route /users is handled by the index method defined in the UserController.
Route::get('/user', [UserController::class, 'index']);

// Two routes grouped to use different methods from the same controller.
Route::controller(OrderController::class)->group(function () {
    Route::get('/orders/{id}', 'show');
    Route::post('/orders', 'store');
});
```

Controllers

- Controllers organize the logic of one or more related routes under the same class.
- Controllers are grouped under
 - `app/Http/Controllers`.
- Artisan provides commands to create a controller template.
 - `php artisan make:controller UserController`

```
<?php

namespace App\Http\Controllers;

use App\Models\User;

class UserController extends Controller
{
    /**
     * Show the profile for a given user.
     *
     * @param int $id
     * @return \Illuminate\View\View
     */
    public function show($id)
    {
        return view('user.profile', [
            'user' => User::findOrFail($id)
        ]);
    }
}
```

Resource Controller

- Creating methods for all standard CRUD operations is a common use case.
- Laravel provides mechanisms to streamline this scenario.
- First, create a standard resource controller with Artisan:
 - `php artisan make:controller PhotoController --resource`
- This will create methods all for standard HTTP CRUD operations.
- These methods can be registered as routes with a single command:
 - `Route::resource('photos', PhotoController::class);`

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Data Validation

- Laravel includes many features to support validation of incoming data.
- The standard approach is to use the `validate` method in all incoming HTTP requests.
- Validation rules are passed into the `validate` method. If it fails, a validation exception will be thrown.

```
// Store a new blog post.
public function store(Request $request)
{
    $validated = $request->validate([
        'title' => ['required', 'unique:posts', 'max:255'],
        'body' => ['required'],
    ]);
    // The blog post is valid...
}
```

- Official documentation:
 - Validation, laravel.com/docs/9.x/validation
 - Validation rules, laravel.com/docs/9.x/validation#available-validation-rules

Blade Templating

Views

- Views enable the separation of the presentation layer (from the model and controller layers), and are defined under the `resource/views` directory.
- Views can be nested within subdirectories and then referenced using “dot notation”.

```
// Calls the view stored at resources/views/about.blade.php
return view('about', $data);
```

```
// Calls the view stored at resources/views/admin/profile.blade.php
return view('admin.profile', $data);
```

Passing Data to Views

- Data can be passed directly as an array of key / value pair.
- Or, using the `with` function can be used with the `view` method.

```
// Data can be passed as an array of key / value pair.  
return view('greetings', ['name' => 'Victoria']);  
  
// Individual pieces of data can be passed using with.  
return view('greeting')  
    ->with('name', 'Victoria')  
    ->with('occupation', 'Astronaut');
```

Blade Templates

- Blade is the templating engine included with Laravel.

```
<!DOCTYPE html>
<html lang="{{ app()>getLocale() }}>
    <head>
        <!-- CSRF Token -->
        <meta name="csrf-token" content="{{ csrf_token() }}>

        <title>{{ config('app.name', 'Laravel') }}</title>

        <!-- Styles -->
        <link href="{{ asset('css/milligram.min.css') }}" rel="stylesheet">
        <link href="{{ asset('css/app.css') }}" rel="stylesheet">
    </head>
    <body>
        <main>
            <header>
                <h1><a href="{{ url('/cards') }}>Thingy!</a></h1>
                @if (Auth::check())
                    <a class="button" href="{{ url('/logout') }}> Logout </a> <span>{{ Auth::user()->name }}</span>
                @endif
            </header>
            <section id="content">
                @yield('content')
            </section>
        </main>
    </body>
</html>
```

Displaying Data

- Data can be displayed by using curly brackets to wrap a variable.
- Results of PHP functions can also be displayed inside a Blade template.
- The `@verbatim / @endverbatim` directives can be used to display raw text.

```
Route::get('/', function () {
    return view('welcome', ['name' => 'Samantha']);
});
```

```
// Displaying a variable.
Hello, {{ $name }}.
```

```
// Displaying the result of a PHP function.
The current UNIX timestamp is {{ time() }}.
```

```
@verbatim
<div class="container">
    Hello, {{ name }}.
</div>
@endverbatim
```

Conditional Directives (if)

- Blade provides shortcuts to common PHP control structures, e.g. if statements.

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif

@unless (Auth::check())
    You are not signed in.
@endunless
```

```
@isset($records)
    // $records is defined and is not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
```

Conditional Directives (environment)

- Blade also includes conditional authentication directives and environment directives.

```
@auth
    // The user is authenticated...
@endauth

@guest
    // The user is not authenticated...
@endguest
```

```
@production
    // Production specific content...
@endproduction

@env('staging')
    // The application is running in "staging"...
@endenv

@env(['staging', 'production'])
    // The application is running in "staging" or "production"...
@endenv
```

Loop Directives

- Blade provides directives to work with PHP's loop structures
- The loop variable can be used to further customize the template.

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

```
@foreach ($users as $user)
@if ($loop->first)
    This is the first iteration.
@endif

@if ($loop->last)
    This is the last iteration.
@endif

<p>This is user {{ $user->id }}</p>
@endforeach
```

Including Subviews

- The `@include` directive can be used to include a Blade view inside another view.
- All variables available to the parent view are also made available to the included view.
- Conditional includes are possible with `@includeIf`, `@includeWhen`, `@includeUnless`.

```
<div>
    @include('shared.errors')

    <form>
        <!-- Form Contents -->
    </form>
</div>
```

```
@includeWhen($boolean, 'view.name', ['status' => 'complete'])

@includeUnless($boolean, 'view.name', ['status' => 'complete'])
```

Building Layouts

- Template inheritance can be used to break-up complex designs.
- The `@section` directive defines a section of content.
- The `@yield` directive is used to display the contents of a given section.

```
<!-- resources/views/layouts/app.blade.php -->

<html>
  <head>
    <title>App Name - @yield('title')</title>
  </head>
  <body>
    @section('sidebar')
      This is the master sidebar.
    @show

    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

```
@extends('layouts.app')

@section('title', 'Page Title')

@section('sidebar')
  @parent

  <p>This is appended to the master sidebar.</p>
@endsection

@section('content')
  <p>This is my body content.</p>
@endsection
```

Database and Eloquent

Database

- Laravel provides a suit of tools for interacting with databases.
- Database interaction is supported through the use of:
 - raw SQL, a query builder, and Eloquent ORM.
- Laravel provides first-party support for:
 - MariaDB, MySQL, PostgreSQL, SQLite, SQL Server.
- Configuration is done through environment variables (.env) and located in
 - config/database.php

Raw SQL Queries

- Once the database connection is configured, SQL queries are run using the DB facade methods: select, update, insert, delete, and statement.

```
use Illuminate\Support\Facades\DB;

// Basic example.
$users = DB::select('select * from users');

foreach ($users as $user) {
    echo $user->name;
}

// Using named bindings in a select SQL statement.
$results = DB::select('select * from users where id = :id', ['id' => 1]);

// Insert SQL statement.
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Marc']);

// General SQL statement.
DB::statement('drop table users');
```

SQL Query Builder

- Laravel query builder can be used to interact with the database through a convenient method-based interface.
- Extensive support for:
 - aggregates
 - joins
 - where clauses
 - ordering, grouping, limit, offset
 - etc.
- Query builder documentation
 - laravel.com/docs/9.x/queries

```
use Illuminate\Support\Facades\DB;

// Basic example using query builder.
$users = DB::table('users')->get();

foreach ($users as $user) {
    echo $user->name;
}

// Retrieving a single row and column.
$user = DB::table('users')->where('name', 'John')->first();

return $user->email;

// Retrieving a list of column values
$titles = DB::table('users')->pluck('title');

foreach ($titles as $title) {
    echo $title;
}
```

Eloquent ORM

- Laravel includes Eloquent, an object-relational mapper (ORM).
- Using Eloquent, each database table has a corresponding model that is used to interact with that table.
- Model classes can be generated using Artisan, e.g.:
 - `php artisan make:model Flight`
- This is the approach adopted in LBAW.

Eloquent Naming Conventions

- Table names:
 - Eloquent will assume that a `Flight` model will link to a `flights` table; while an `AirTrafficController` table will store records in an `air_traffic_controller` table.
 - If the naming does not fit this convention, the `$table` property on the model can be used to specify the table name.
- Primary keys:
 - Eloquent will assume that each model's corresponding database table has a primary key column named `id`.
 - The `$primaryKey` property can be used to specify a different column name.
 - Eloquent also assumes that primary keys are automatically incrementing integers.

timestamps

- Eloquent expect two timestamps columns,
 - `created_at` and `updated_at`
 - Laravel uses these to manage migrations.
 - To disable this, set `$timestamps` to false.
- These are not used in LBAW (not migrations).

Retrieving Models

- Once models are created, they can be used to retrieve data.
- Each model will work as a specialized query builder.

```
// Using the Flight model to query the database.  
use App\Models\Flight;  
  
foreach (Flight::all() as $flight) {  
    echo $flight->name;  
}  
  
// Each model serves as a query builder.  
$flights = Flight::where('active', 1)  
    ->orderBy('name')  
    ->limit(10)  
    ->get();
```

Inserting Models

- To insert a new record to the database, a new model is instantiated.
- The `save` method is used to store the instance in the database.

```
use App\Models\Flight;

// Create a new Flight instance, set data, and store it.
$flight = new Flight;

$flight->name = "John Smith";

flight->save();

// Single statement call.
$flight = Flight::create([
    'name' => 'London to Paris',
]);
```

Eloquent Relationships

- To navigate between related records, using each model's relationships, Eloquent allows for the definition of relationships.
- This allows for chaining additional query constraints, e.g.:
 - `$post->comments()->where('title', 'foo')->first();`
 - `$comment->post->title;`
- Example, a Post has many Comments (one-to-many relationship).
 - Post hasMany Comments
 - Comment belongsTo Post

```
class Post extends Model
{
    // Get the comments for the blog post.
    public function comments() {
        return $this->hasMany(Comment::class);
    }
}
```

```
class Comment extends Model
{
    // Get the post that owns a comment.
    public function post() {
        return $this->belongsTo(Post::class);
    }
}
```

- See Eloquent Relationships for details on how to map
 - laravel.com/docs/9.x/eloquent-relationships

Database Migrations

- Database migrations are a solution to manage database schema evolutions.
 - Migrations can be compared to version control for a database.
- The migration class contains two methods, `up` and `down`.
 - The `up` method is used to add new tables, columns, or indexes.
 - The `down` method is used to reverse the operations performed by the `up` method.
- **Migrations are not used in LBAW.**
 - Database design is not made through Laravel.

Database Migrations Example

- The following migration creates a `flight` table.

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    // Run the migrations.
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('flights');
    }
};
```

Authentication and Authorization

Authentication

Authentication

- Laravel includes a built-in authentication service.
 - Official documentation at laravel.com/docs/9.x/authentication
- LBAW's template-laravel already includes authentication features that should be adapted to each project.
- See “Manually Authenticating Users” in Laravel’s documentation.
 - laravel.com/docs/9.x/authentication#authenticating-users
- Alternatively, a pre-configured quick starter kit is available with Laravel Breeze, which includes a simple implementation of all authentication features, including login, registration, password reset, email verification, and password confirmation.
 - laravel.com/docs/9.x/starter-kits#laravel-breeze

Manual Authentication Setup

- A default Laravel project includes a User model.
 - Disable timestamps in the database with:
 - `public $timestamps = false;`
 - Add any necessary relationships.
- Required database changes:
 - You need to add a column named `remember_token` to store a token for the “remember me” option. This column must be nullable and support 100 character strings.
- Create LoginController
 - See “Manually Authenticating Users”, laravel.com/docs/9.x/authentication#authenticating-users
- Add authentication routes.

template-laravel Setup

- Authentication is already set up in template-laravel.
 - Under `app/Http/Controllers/Auth`
- You may need to make changes if your table or column names differ from the defaults [users, username, and password].
 - In the User model, set the `$table` property for a different table name.
 - In the LoginController, define a `username` function to return the field name for the username field.
 - In the User model, define a `getAuthPassword` function to return the field name for the password field.
- To support the “Remember me” feature you need to add a `remember_token` column to your users’ table schema.

Authorization

Authorization

- Laravel also provides features to authorize user actions against given resources.
- Two main solutions exist, and both can be used in a single application,
 - Gates, can be compared to routes, i.e. closure-based approach.
 - Policies, can be compared to controllers, i.e. define logic associated to a model or resource.
- Gates are most applicable to actions not related to any model or resource, e.g. view the admin panel.
- Policies are used to authorize actions for a particular model or resource, and offer a more robust and fine-grained control to define authorization.
- LBAW's template-laravel makes use of policies.

Policies

- Policies are defined within the `app/Policies` directory.
- Policies are organized around specific models or resources.
 - E.g., `app/Policies/PostPolicy`
- New policies can be created by hand or automatically created using Artisan.
 - `php artisan make:policy PostPolicy`
 - `php artisan make:policy PostPolicy --model=Post`
- The use of the `model` parameter instructs Artisan to create skeletons for methods related to viewing, creating, updating, and deleting resources.

Registering Policies

- Policies need to be registered to inform Laravel which policy to use when authorizing against a given model type.
- This is done in the `app/Providers/AuthServiceProvider.php` file, setting the **policies** property.
- The following configuration instructs Laravel to use the **PostPolicy** class when authorizing actions against the **Post** Eloquent model.

```
class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        Post::class => PostPolicy::class,
    ];
    ...
}
```

Policy Auto-Discovery

- Instead of manually registering model policies, Laravel can automatically discover policies as long as model and policy follow standard naming conventions.
 - Policies must be placed in the Policies directory.
 - The policy name must match the model name and have “Policy” as a suffix.
 - E.g., `app/Models/User` and `app/Policies/PostPolicy`.

Policy Methods

- The methods defined in each policy class, define the actions to be authorized.
- For example, an `update` method on the `PostPolicy` class determines if a given `User` can update a given `Post` instance.
 - The `update` method receives a `User` and a `Post` instance as arguments.
 - And should return `true` or `false`.
 - The following code verifies if the user's `id` matches the `user_id` on the post.

```
public function update(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

Authorizing Actions

- Using policies, actions can be authorized at different levels, specifically via:
- The user model, using `can` and `cannot` methods, e.g.:
 - `if ($request->user()->can('update', $post)) { ... }`
- Controller helpers, using the `authorize` method, e.g.:
 - `$this->authorize('create', $item);`
 - `$this->authorize('update', $post);`
- Blade templates, using `@can` and `@cannot` directives, e.g.:
 - `@can('update', $post)`
 - `@elsecan('create', App\Models\Post::class)`

template-laravel example

```
// app/Http/Controllers/ItemController.php

class ItemController extends Controller
{
    /**
     * Creates a new item.
     *
     * @param int $card_id
     * @param Request request containing the description
     * @return Response
     */
    public function create(Request $request, $card_id)
    {
        $item = new Item();
        $item->card_id = $card_id;

        // Throws an exception if the user is not authorized to create this item.
        $this->authorize('create', $item);

        $item->done = false;
        $item->description = $request->input('description');
        $item->save();

        return $item;
    }
}
```

```
// app/Policies/ItemPolicy.php

use Illuminate\Auth\Access\HandlesAuthorization;

class ItemPolicy
{
    use HandlesAuthorization;

    public function create(User $user, Item $item)
    {
        // User can only create items in cards they own.
        return $user->id == $item->card->user_id;
    }
}
```

Laravel Ecosystem

Laravel Ecosystem

- Laravel has a mature and dynamic ecosystem.
- Many tools, integrations and packages, both from first-party and third-parties.
- Laravel Vapor, serverless deployment platform
 - <https://vapor.laravel.com>
- Laravel Conference
 - <https://laracon.eu>
- Laravel Bootcamp
 - <https://bootcamp.laravel.com>
- Laravel Jobs
 - <https://larajobs.com>

Additional Topics

- Much more to explore and use. Some highlights.
- Artisan Console
 - <https://laravel.com/docs/9.x/artisan>
- Data Validation
 - <https://laravel.com/docs/9.x/validation>
- Testing
 - <https://laravel.com/docs/9.x/testing>
- Telescope Package
 - <https://laravel.com/docs/9.x/telescope>

LBAW ‘template-laravel’

LBAW ‘template-laravel’

- A sample Laravel project is provided with template-laravel.
 - <https://git.fe.up.pt/lbaw/template-laravel>
- This project implements a simple task manager called Thingy!
- Is expected to be used as a starting point for the development of the prototype.
 - User registration and authentication is already included and can be adapted.
 - Thingy! specific features need to be removed.

Publishing the Project

- The project is published using Docker.
 - A container is built using the provided Dockerfile.
 - The container is uploaded to the group's Gitlab Container Registry
 - https://git.fe.up.pt/lbaw/template-laravel/container_registry
 - Each container is periodically fetched to lbaw.fe.up.pt
 - <http://template-laravel.lbaw.fe.up.pt>
- The script **upload_image.sh** handles the details:
 - https://git.fe.up.pt/lbaw/template-laravel/-/blob/master/upload_image.sh

Project Container Dockerfile

```
FROM ubuntu:21.10

# Install dependencies
env DEBIAN_FRONTEND=noninteractive
RUN apt-get update
RUN apt-get install -y --no-install-recommends libpq-dev vim nginx php8.0-fpm php8.0-mbstring php8.0-xml php8.0-pgsql

# Copy project code and install project dependencies
COPY --chown=www-data . /var/www/

# Copy project configurations
COPY ./etc/php/php.ini /usr/local/etc/php/conf.d/php.ini
COPY ./etc/nginx/default.conf /etc/nginx/sites-enabled/default
COPY .env /var/www/.env
COPY docker_run.sh /docker_run.sh

# Start command
CMD sh /docker_run.sh
```

References

- Laravel: Up & Running, Matt Stauffer. O'Reilly, 2019
- Laravel Official Documentation (version 9.x), <https://laravel.com/docs/9.x>

Client-Side Web Technologies

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
DEI, FEUP, U.Porto

The Big Picture

→ Web browsers issue requests to web servers, which produce and return HTML documents for browsers to parse and display.



1. HTTP request

2. HTTP answer + HTML document

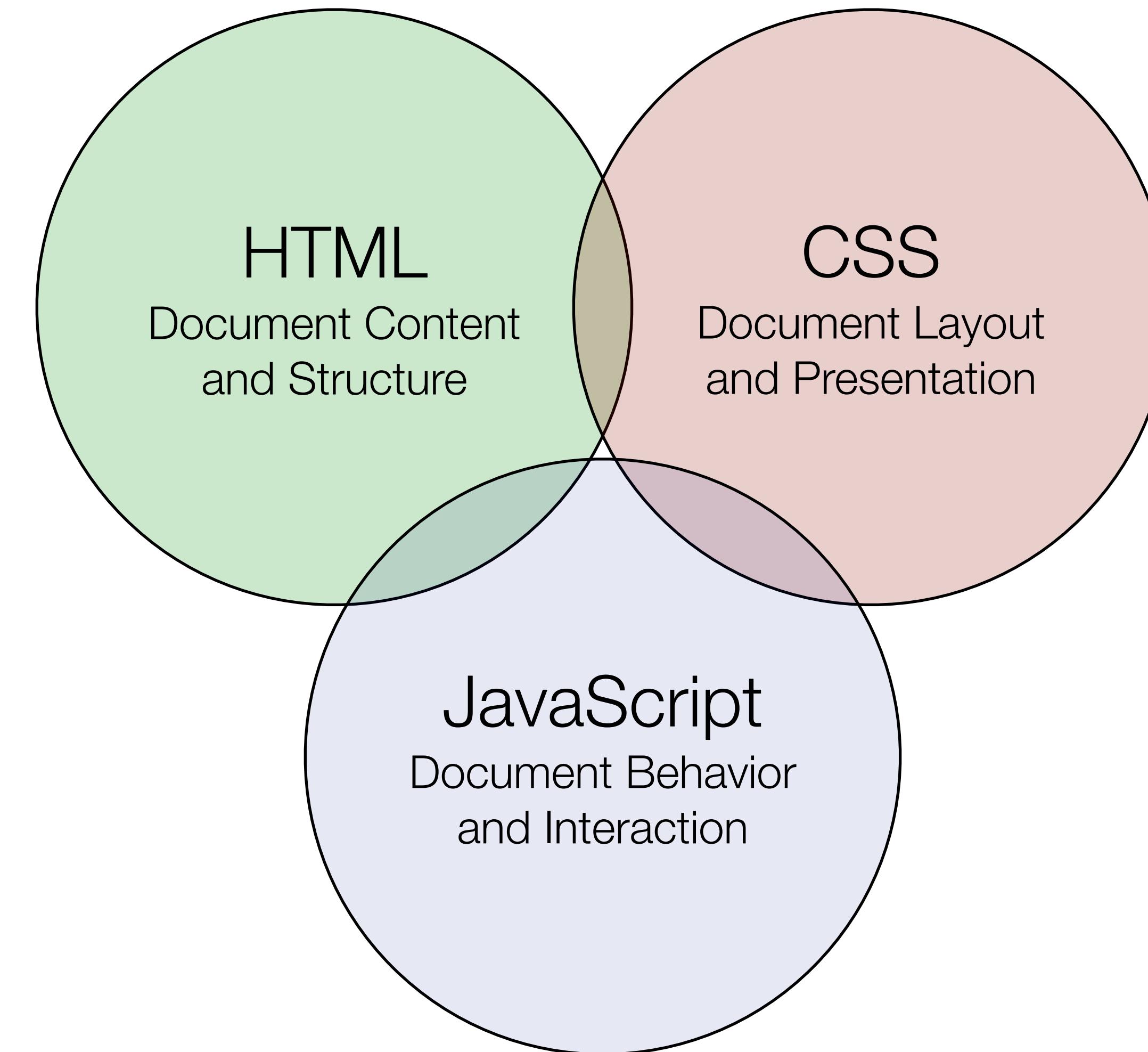


Client

Server

A screenshot of a browser window titled 'view-source:www.facebook.com'. The content is a large block of raw HTML code, showing the structure of the Facebook homepage. The code includes various HTML tags like <html>, <head>, <body>, and <div>. It also contains CSS styles and JavaScript scripts. The code is heavily nested, reflecting the complex nature of a modern web application's front-end.

Client-side Web Technologies



HTML: HyperText Markup Language

HyperText Markup Language

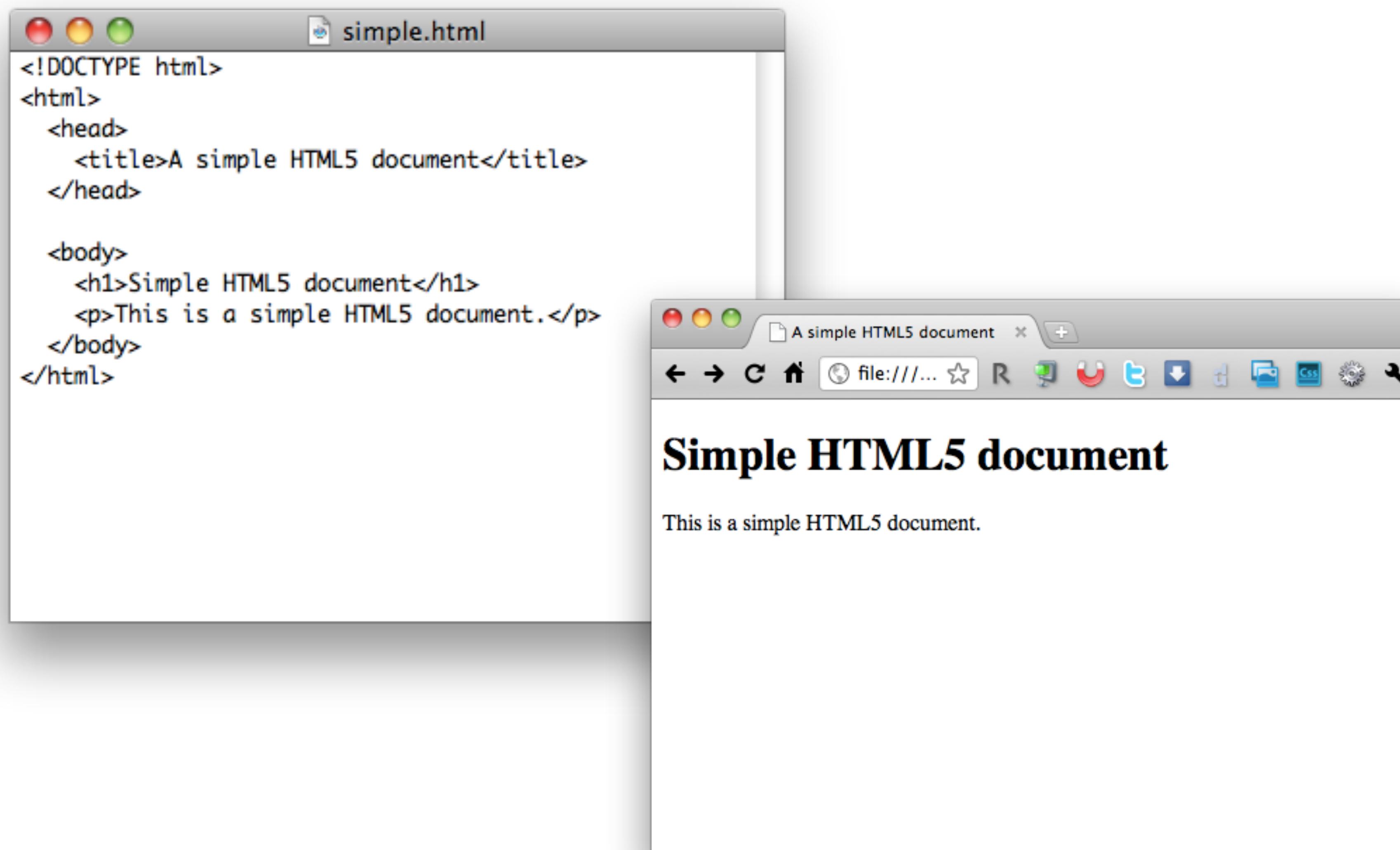
- HTML is an acronym for HyperText Markup Language and is a format for providing linked structured information.
- HTML documents are simply text files containing marked-up text using tags.
- An HTML document is an hypertext node within an hypertext network.

Hypertext

- Concept defined by Ted Nelson in the 1950s.
- A way to organize text (and information) in a non-linear fashion.
- “Hypertext: Human-readable information linked together in an unconstrained way.”
- From the original WorldWideWeb: Proposal for a HyperText Project (1990)
 - “HyperText is a way to link and access information of various kinds as a web of nodes in which the user can browse at will.

It provides a single user-interface to large classes of information (reports, notes, data-bases, computer documentation and on-line help).”

Basic HTML Document



The image shows a comparison between the source code of an HTML file and its rendered output in a web browser.

Left Window (Code View):

```
<!DOCTYPE html>
<html>
  <head>
    <title>A simple HTML5 document</title>
  </head>

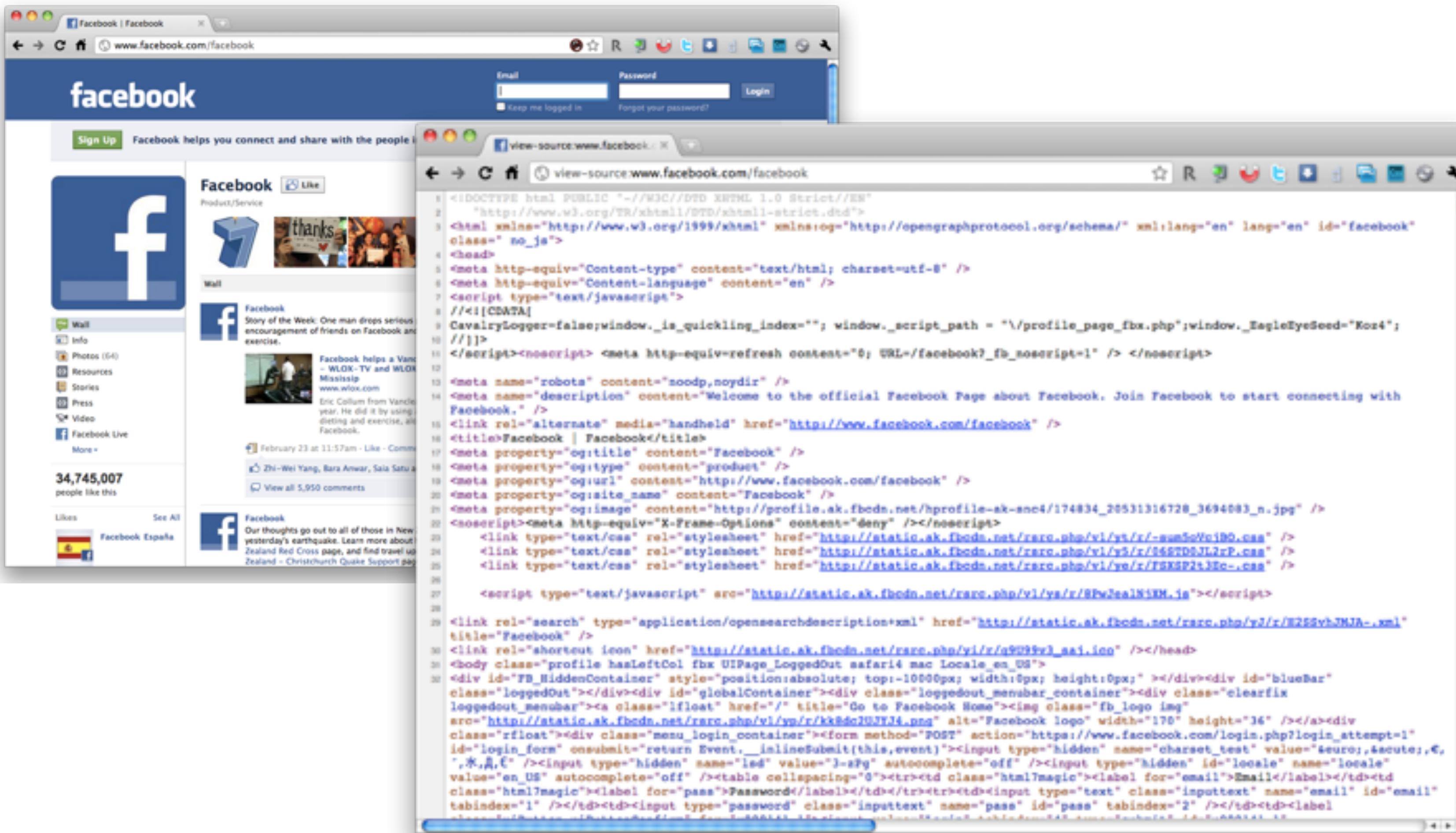
  <body>
    <h1>Simple HTML5 document</h1>
    <p>This is a simple HTML5 document.</p>
  </body>
</html>
```

Right Window (Preview):

A simple HTML5 document

This is a simple HTML5 document.

View Source



A Brief History of HTML

Origins of HTML

- Created by Tim Berners-Lee and Robert Cailliau at CERN in the late 1980s.
- Main goal was to facilitate document sharing between researchers.
- CERN released it as royalty free in 1993.
- First official version published by IETF in 1993.
- World Wide Web Consortium (W3C) was created to define common standards for browsers and developers to adhere to.

HTML Proposal

→ **Information Management: A Proposal**

<https://www.w3.org/History/1989/proposal.html>

- “This proposal concerns the management of general information about experiments at CERN.”
- “It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.”
- Some practical requirements: remote access, heterogeneity, non-centralization, text-based, “live links”.
- Problems being addressed:
 - Information loss - “Often, the information has been recorded, it just cannot be found.”
 - Constantly changing information. Keeping a “book-like” organization of all information at CERN is impractical. Changes are distributed.
 - Tree-like organizations and keyword-based organization are also not feasible. Too strict and inflexible.

HTML Timeline

- During its first years (1990-1995), HTML revisions and extensions were first hosted at CERN and then IETF.
- Development was moved to the W3C after its creation in 1994.
- HTML development stopped in 1998 with the publication of HTML 4.
- W3C decided to migrate to an XML-based equivalent, named XHTML.
- XHTML was not widely adopted by web authors.
- HTML development continued outside W3C, with the WHATWG, whose work is now the basis for HTML5.
- WHATWG - Web Hypertext Application Technology Working Group

The Early Days (1989 - 1993)

- From proposal (1989) to Mosaic release (1993).
- Web users were mostly from academia and research institutions.
- Few browsers, most of them text-based.
- HTML documents were simple and usually written by hand.

Growth Years (1994 - 2002)

- Wide adoption of the web to the dot.com bubble (1995-2000).
- Companies dispute the web browser market (aka “browser wars”).
- Browser development focused on new features, less on standards support.
- Wide differences between rendering engines.
Many web pages “designed for browser version x.x”.
- Extensive use of tables and sliced graphics to achieve “pixel perfect” layouts - “print-like design”. Resulted in ugly and complex HTML code.

Modern Era (2003 -)

- Wide adoption of modern web browsers.
- Separation of content and structure from layout and presentation.
- HTML controls content and structure.
- CSS controls layout and presentation.
- Clean and simple code (again!).
- CSS (2003), AJAX (2005), mobile (2007).
- A platform for (web) applications.

HTML

XHTML

- In 1998, the W3C decided to abandon HTML development and focus on a XML-based equivalent, named XHTML.
- XHTML 1.0 was completed in 2000.
- W3C then moved to XHTML 2.0, introducing several new features and less backward compatibility.
- Real world adoption of XHTML was small.
- In 2004, a proposal to refocus on HTML was discarded by the W3C, leading to outside development of HTML.

WHATWG

- Members of the W3C formed a new group: the Web Hypertext Application Technology Working Group (WHATWG).
- WHATWG didn't follow a consensus-based approach, so it was able to move much faster.
- In 2006, the W3C acknowledged that XHTML wasn't being adopted and work on HTML was resumed.
- Instead of starting from scratch, the W3C decided to use the work from WHATWG.
- Work on XHTML 2.0 ended in 2009.

W3C and WHATWG

- WHATWG continues working on HTML as a "living standard" (no versions).
<https://html.spec.whatwg.org/>
- Latest published W3C version of HTML is 5.2.
<https://www.w3.org/TR/html52/>
- Ongoing discussions on how to manage the work and collaboration between WHATWG and W3C, e.g. stop publishing two separate specifications.
- More details: <https://wiki.whatwg.org/wiki/W3C>

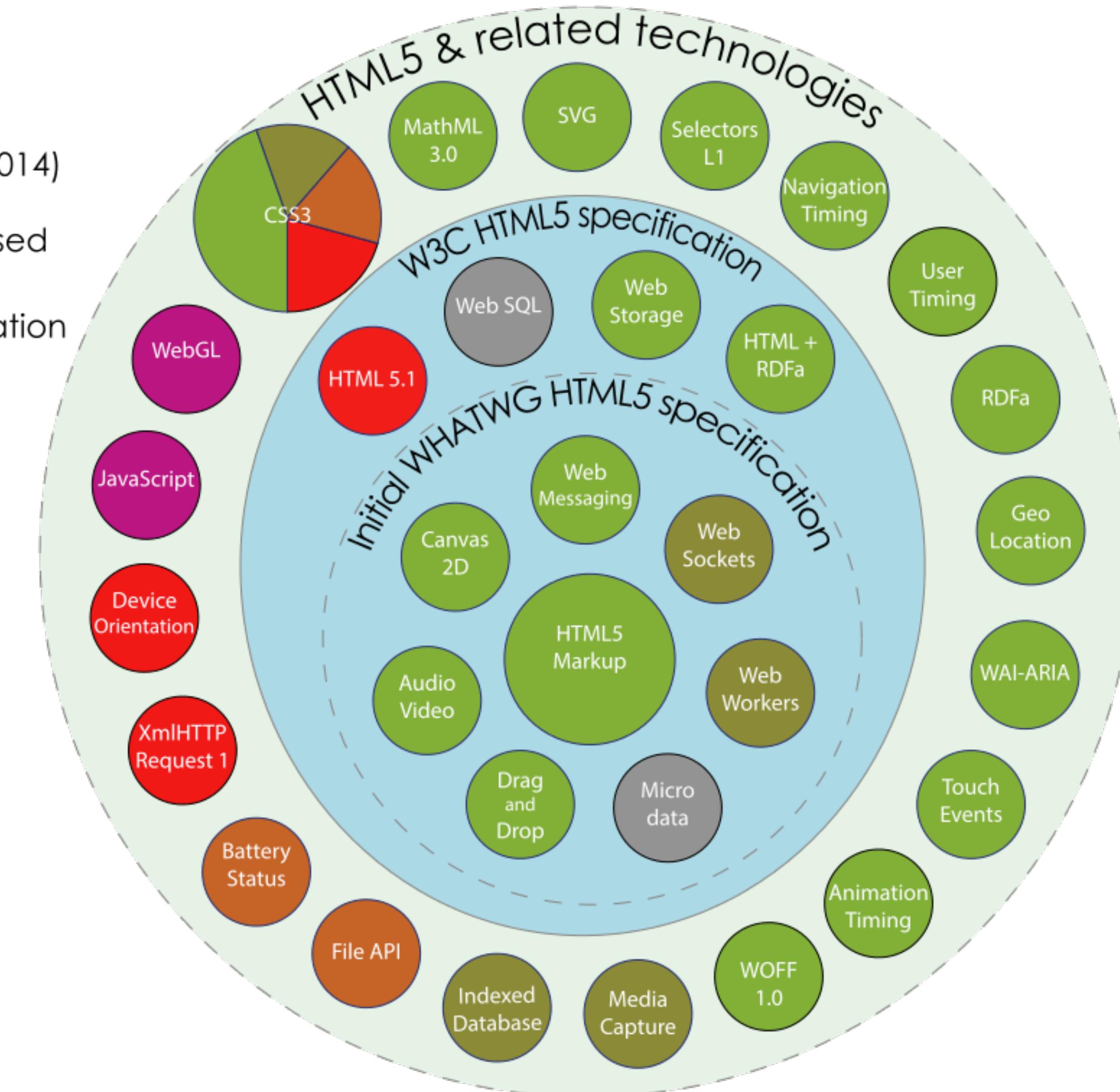
HTML5 Technologies

- HTML5 is a collection of features and technologies.
- Language / Markup features
- Document Model Definition (DOM)
- APIs for supporting JavaScript interaction with the DOM

HTML5

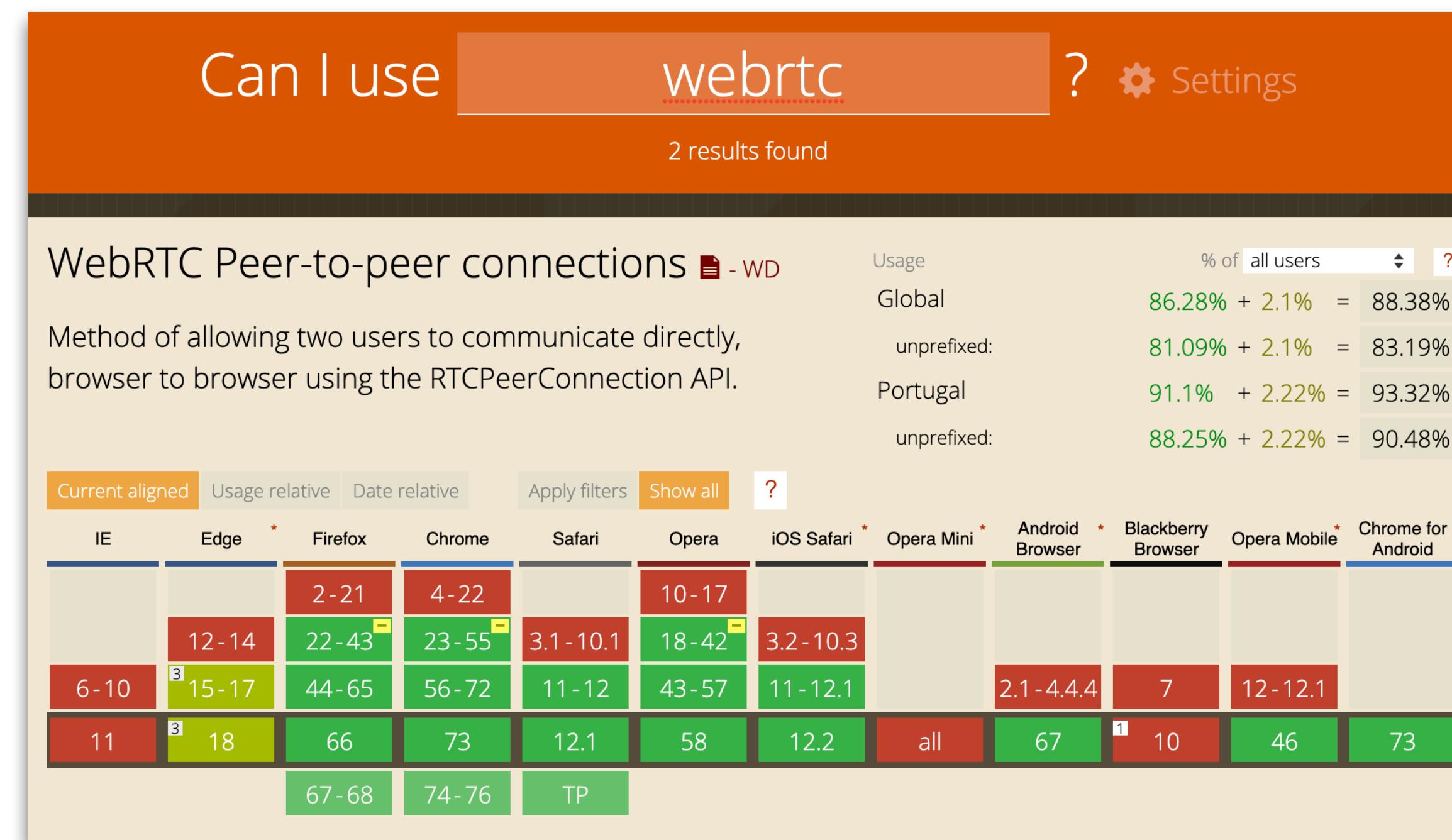
Taxonomy & Status (October 2014)

- Recommendation/Proposed
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated or inactive



Browser Support

- Support for these technologies has different levels of support in browsers.
- "Can I Use" provides up-to-date information about browser support of front-end technologies. <https://caniuse.com>



HTML Microdata

HTML Microdata

- Extension to define new attributes and embed simple machine-readable data in HTML documents.
- Goal: annotate content with machine-readable labels.
- Common use case: search engines can better 'understand' and index information that has been annotated using schema.org vocabulary.
- Microdata provides a mechanism to identify items and define their properties.
 - The `itemscope` attribute creates an item.
 - The `itemprop` attribute descends of `itemscope` and defines an item property.
 - With `itemtype` is possible to associate a vocabulary to an item.
 - An `itemid` can be used to define a global unique identifier for the item.

Microdata Example

→ Defines an item with two properties.

```
<div itemscope>
  <p>Flavors in my favorite ice cream:</p>
  <ul>
    <li itemprop="flavor">Lemon sorbet</li>
    <li itemprop="flavor">Apricot sorbet</li>
  </ul>
</div>
```

Schema.org

- Vocabularies define concepts and relationships used to describe and represent areas of concern. Can be very simple (one or two concepts) or very complex (thousands of terms).
- A shared vocabulary makes it possible to have a common understanding of defined concepts and relationships.
- Schema.org is a collaborative, community driven initiative to create, maintain, and promote the use of schemas for structured data on the web. Founded by Google, Microsoft, Yahoo, and Yandex.
- Schema.org defines more than 600 types and >900 properties. Such as CreativeWork, Book, Movie, Event, Organization, Person, Place, Restaurant, etc.

Microdata Example using Vocabulary

- Example using Schema.org vocabulary.
- Defines an item of the type LocalBusiness, as defined by the Schema.org vocabulary, containing three properties, one of which is a item of the type PostalAddress, containing four properties.

```
<div itemscope itemtype="http://schema.org/LocalBusiness">
  <h1 itemprop="name">Beachwalk Beachwear & Giftware</h1>
  <span itemprop="description"> A superb collection [...].</span>
  <div itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
    <span itemprop="streetAddress">3102 Highway 98</span>
    <span itemprop="addressLocality">Mexico Beach</span>,
    <span itemprop="addressRegion">FL</span>
  </div>
  Phone: <span itemprop="telephone">850-648-4200</span>
</div>
```

HTML Microdata References

- W3C Editor's Draft - Microdata (April 2021)
<https://w3c.github.io/microdata/>
- HTML Standard Microdata Specification
<https://html.spec.whatwg.org/#microdata>
- Schema.org
<https://schema.org/>
- Semantic Web (aka Web of Data)
<https://www.w3.org/standards/semanticweb/>

Web APIs

Web APIs

- In addition to the language specification, HTML5 introduced several Web APIs that can be used with JavaScript. There is a large number of APIs in different stages of development.
- Documents manipulation APIs (e.g. DOM, Drag and Drop)
- Fetch remote data APIs (e.g. Fetch, Web Sockets)
- Drawing and graphics manipulation APIs (e.g. Canvas, WebGL)
- Audio and Video APIs (e.g. Web Audio, WebRTC)
- Device APIs (e.g. Notification, Vibration, Fullscreen)
- Client-side storage APIs (e.g. Web Storage, IndexedDB)

Geolocation API

Geolocation API

- The Geolocation API provides scripted access to geographical location information associated with the device.
- Common sources of location information include Global Positioning System (GPS) and location inferred from network signals such as IP address, RFID, WiFi and Bluetooth MAC addresses, and GSM/CDMA cell IDs, as well as user input.
- Available both as single-shot request or continuous tracking.
 - `navigator.geolocation.getCurrentPosition(callback)`
 - `navigator.geolocation.watchPosition(callback)`
- Geolocation API Specification
<https://www.w3.org/TR/geolocation-API/>

Web Storage API

Web Storage API

- Local storage is an important feature for web applications.
- Cookies can be used for persistent local storage but are limited in size and are included in every HTTP request, slowing down the communication and exposing data.
- The Web Storage API specifies a mechanism to persistently store data in web clients, as key-value pairs. Unlike cookies, this data is never shared with the server and can only be accessed by the client.
- Data can be kept during page sessions, using `sessionStorage`, or persisted even when the browser is closed, using `localStorage`.
- Web Storage API Specification
<https://www.w3.org/TR/webstorage/>

Web Storage API

- Data can be stored and retrieved using keys.
 - `localStorage.setItem("key", data)`
 - `localStorage.getItem("key")`
- It is possible to keep track of changes trapping the `storage` event.
- For structured data, the IndexedDB API can be used. This API specified a low-level API for storing and indexing large volumes of data in the client.
- Indexed Database API 3.0, W3C Working Draft (March 2021)
<https://www.w3.org/TR/IndexedDB/>

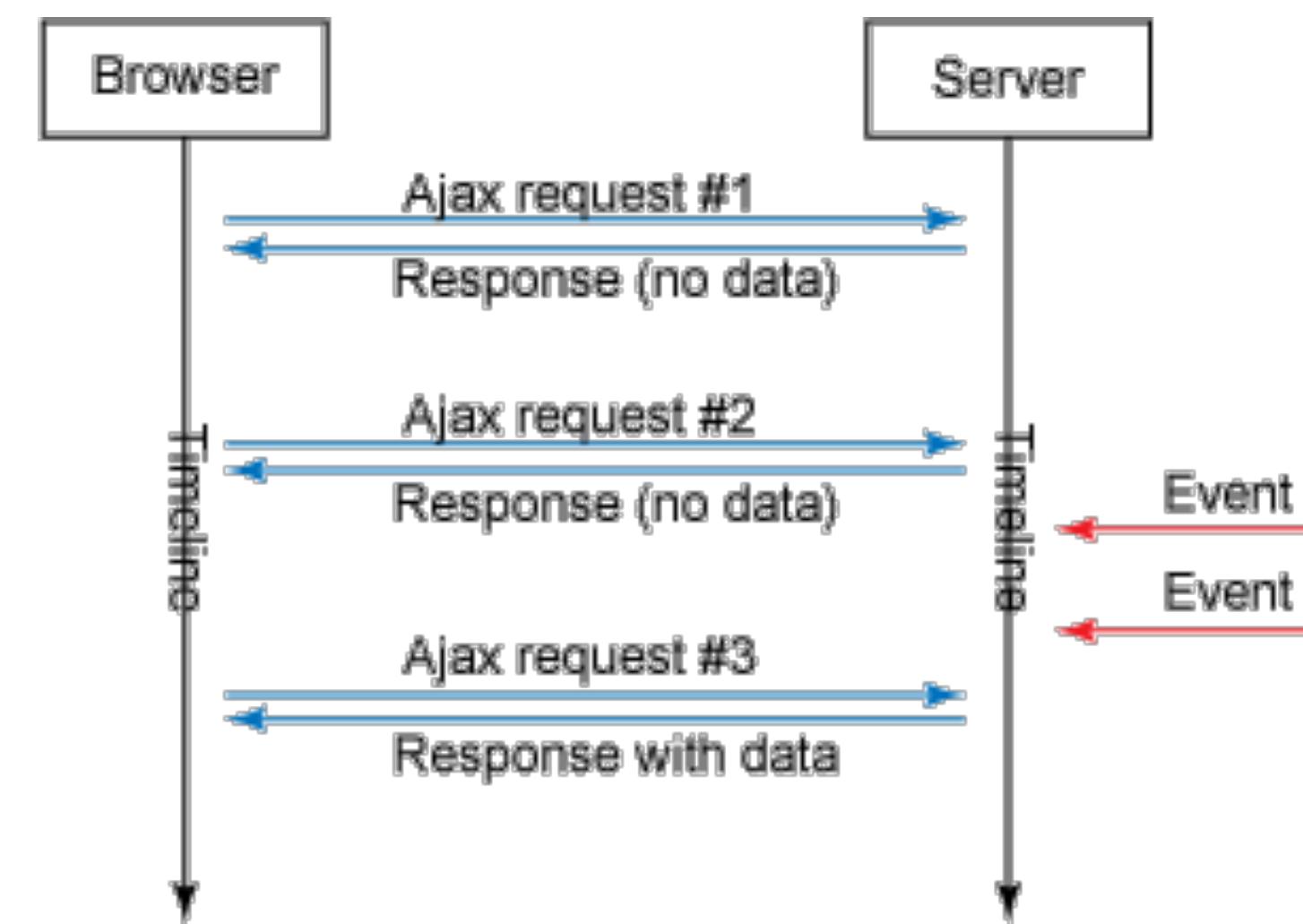
Web Sockets API

Web Sockets API

- Web applications are not restricted to request-response interaction.
- A particularly important use case is the need for server initiated communication (aka "server push").
- Common scenarios include notifications on long running tasks, chat systems, multi-user collaboration systems (e.g. live collaborative text editors).
- How to push information from the server to the client?

Polling

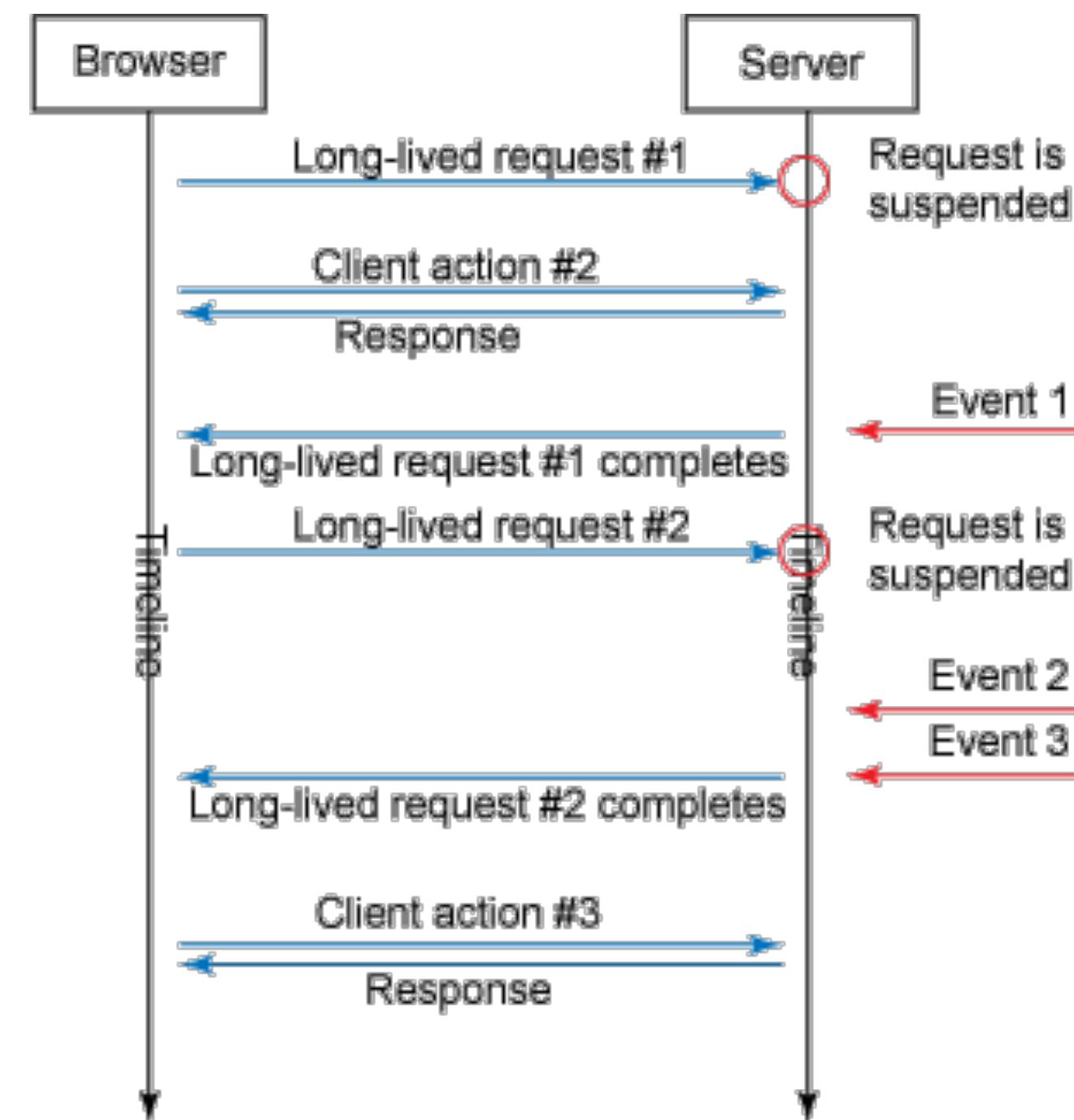
- Make periodic requests to the server to check for new data.



- The smaller the interval between requests the more up to date the data is.
- Drawbacks: resource and bandwidth consumption even when no new data is available. Does not scale well and doesn't guarantee low-latency.

Comet

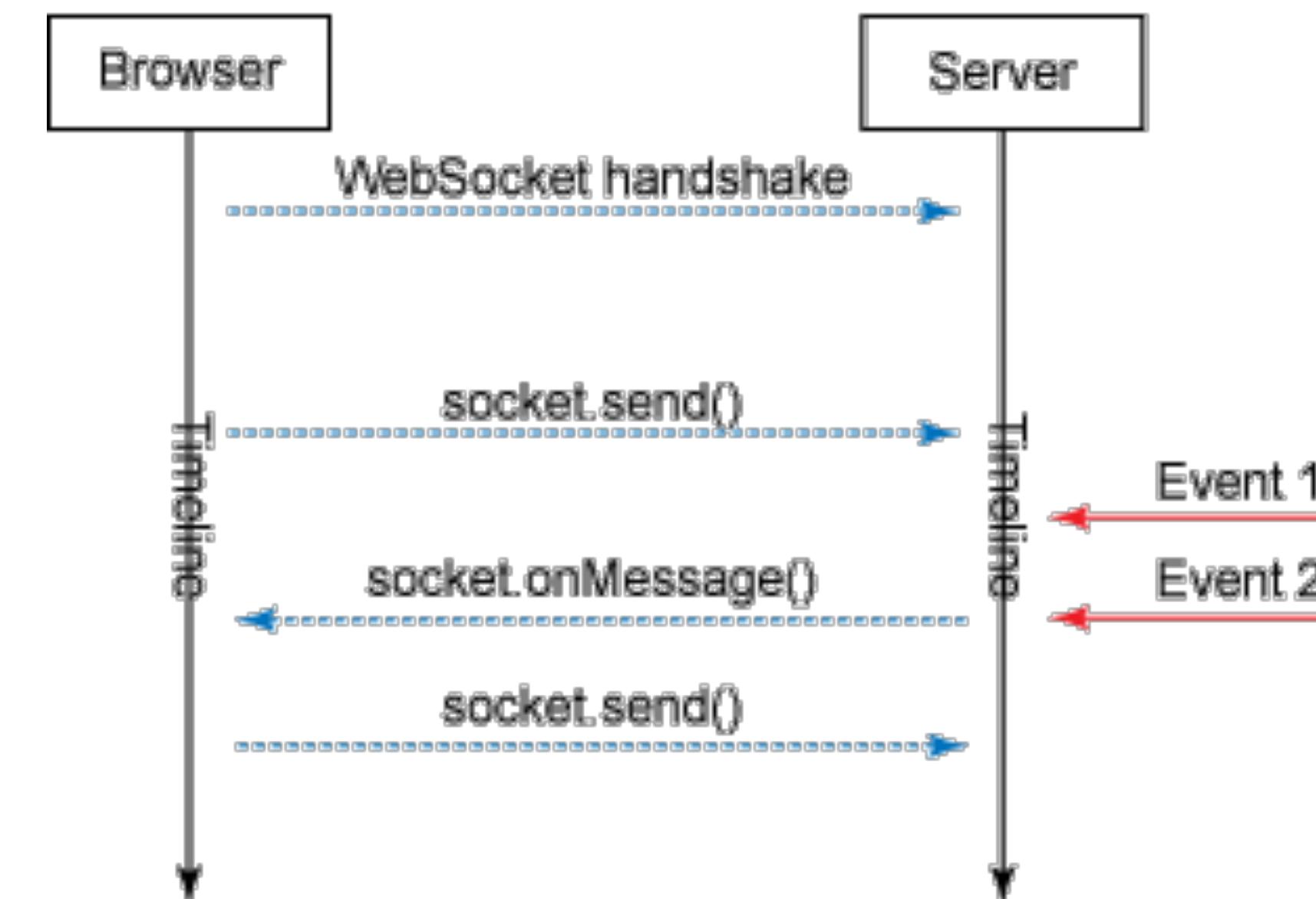
- Requests are initiated by clients and kept alive for long periods, until a timeout occurs or a response is sent.



- On the server, the request is suspended or paused until a response is ready.

Web Sockets

→ Web Sockets enables bidirectional communications between the web browser and the web server. No polling is needed to get messages from the server.



Web Socket Example

```
// Create WebSocket connection.
const socket = new WebSocket('ws://localhost:8080');

// Connection opened
socket.onopen = function (event) {
    socket.send('Hello Server!');
};

// Listen for messages
socket.onmessage = function (event) {
    console.log('Message from server ', event.data);
};
```

Web Sockets References

→ The CometD Reference Book

<https://docs.cometd.org/current/reference/>

→ The WebSocket API | MDN web docs

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

→ The WebSocket API | W3C

<https://www.w3.org/TR/websockets/>

WebRTC API

WebRTC API

- WebRTC (Web Real-Time Communications) is a technology which enables communication between browsers without requiring an intermediary.
- It includes the building blocks for high-quality communications on the web, such as network, audio and video components used in voice and video chat.
- Example file sharing P2P web application: <https://www.sharedrop.io/>
- More: <https://webrtc.github.io/samples/>
- WebRTC Home
<https://webrtc.org/>
- WebRTC API Specification
<https://www.w3.org/TR/webrtc/>

Web Workers API

Web Workers API

- Web Workers provide support for background execution of scripts.
- JavaScript execution is single-threaded. Web Workers are designed to bring concurrency to web applications through the execution of scripts in background threads, independently of any user interface scripts.
- Example use cases:
 - Perform background computationally expensive task.
 - Periodically prefetch data.
 - Share state between multiple clients using a shared worker.
 - Split computationally expensive tasks between clients.

Web Workers API

- Generally, workers are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost.
- There are two kinds of workers: dedicated workers, which are used by a single script, and shared workers, that can be used by multiple scripts.
- Data is shared between the main thread and workers using messages.
- HTML Standard – Web workers (April 2021)
<https://html.spec.whatwg.org/multipage/workers.html>

Web Workers Example

```
<p>The highest prime number discovered so far is: <output id="out"></output></p>
<script>
  var worker = new Worker('worker.js');
  worker.onmessage = function (event) {
    document.getElementById('out').textContent = event.data;
  };
</script>
```

```
var n = 1;
search: while (true) {
  n += 1;
  for (var i = 2; i <= Math.sqrt(n); i += 1)
    if (n % i == 0)
      continue search;
  // found a prime!
  postMessage(n);
}
```

worker.js

Progressive Web Applications

- Progressive Web Applications (or PWAs) represent a new type of web applications, that combine multiple technologies and design patterns to improve user experience.
- Characteristics of progressive web apps: discoverable, installable, linkable, network independent, progressive, responsive, safe.
- Key technology: web workers, which intercept page requests and can use the local storage to provide an answer or make server requests.
- Other relevant technologies: web app manifest, web storage, notifications, etc.
- Progressive Web Apps
<https://developers.google.com/web/progressive-web-apps/>

Progressive Web Apps

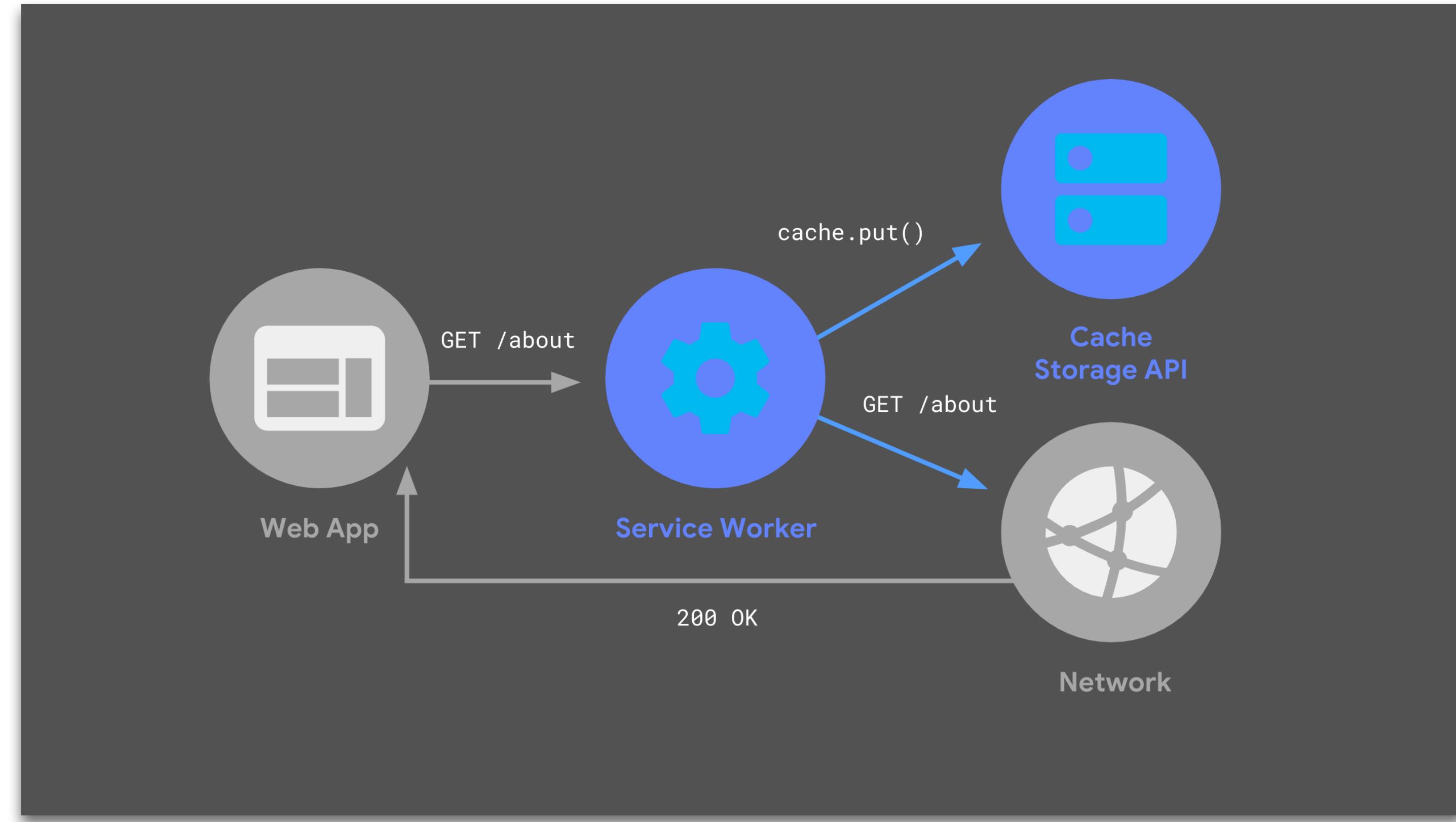


Image from Beyond SPAs: alternative architectures for your PWA (2018)
<https://developers.google.com/web/updates/2018/05/beyond-spa>

HTML References

→ **HTML: HyperText Markup Language | MDN**

<https://developer.mozilla.org/en-US/docs/Web/HTML>

→ **Latest version of HTML**

<https://www.w3.org/TR/html/>

→ **WHATWG HTML Specification**

<https://html.spec.whatwg.org/multipage/>

→ **Dive Into HTML5**

<https://diveintohtml5.info/>

→ **HTML Dog: HTML, CSS and JavaScript tutorials**

<https://htmldog.com/>

→ **Chapter 2 - A history of HTML**

<https://www.w3.org/People/Raggett/book4/ch02.html>

Web Performance

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
DEI, FEUP, U.Porto

Web Performance

- Web optimization techniques are designed to improve the overall response time of a web application to the end-user.
- Usability studies show that page speed has a direct impact on conversion rates. Ideally a web page should load in less than 0.1 seconds, giving the user the feeling of an instantaneous response.
- A response time of less than 1 second keeps the user's flow seamless. Up to 10 seconds the user attention is kept. Over 10 seconds, the user is more likely to leave the page.
- Optimization opportunities both at the back-end or the front-end level.
 - Front-end: reduce images, reduce HTTP calls, etc.
 - Back-end: improve hardware, tune database, etc.

The Golden Rule

- In most web pages, less than 10-20% of the end user response time is spent getting the HTML document. To achieve significant improvements in response times, it is important to focus on front-end optimizations.
- **80% of the end-user response time is spent on the front-end.**
- Where the time is spent:
 - Parsing HTML, Scripts, CSS, and images.
 - Retrieving other page components (scripts, CSS, and images).
- Start with front-end optimizations:
 - Greater potential for improvements.
 - Simpler and proven to work.

Rules for High Performance Web Sites

From: High Performance Web Sites by Steve Souders (2007) &

Best Practices for Speeding Up Your Web Site (Yahoo)

Make Fewer HTTP Requests

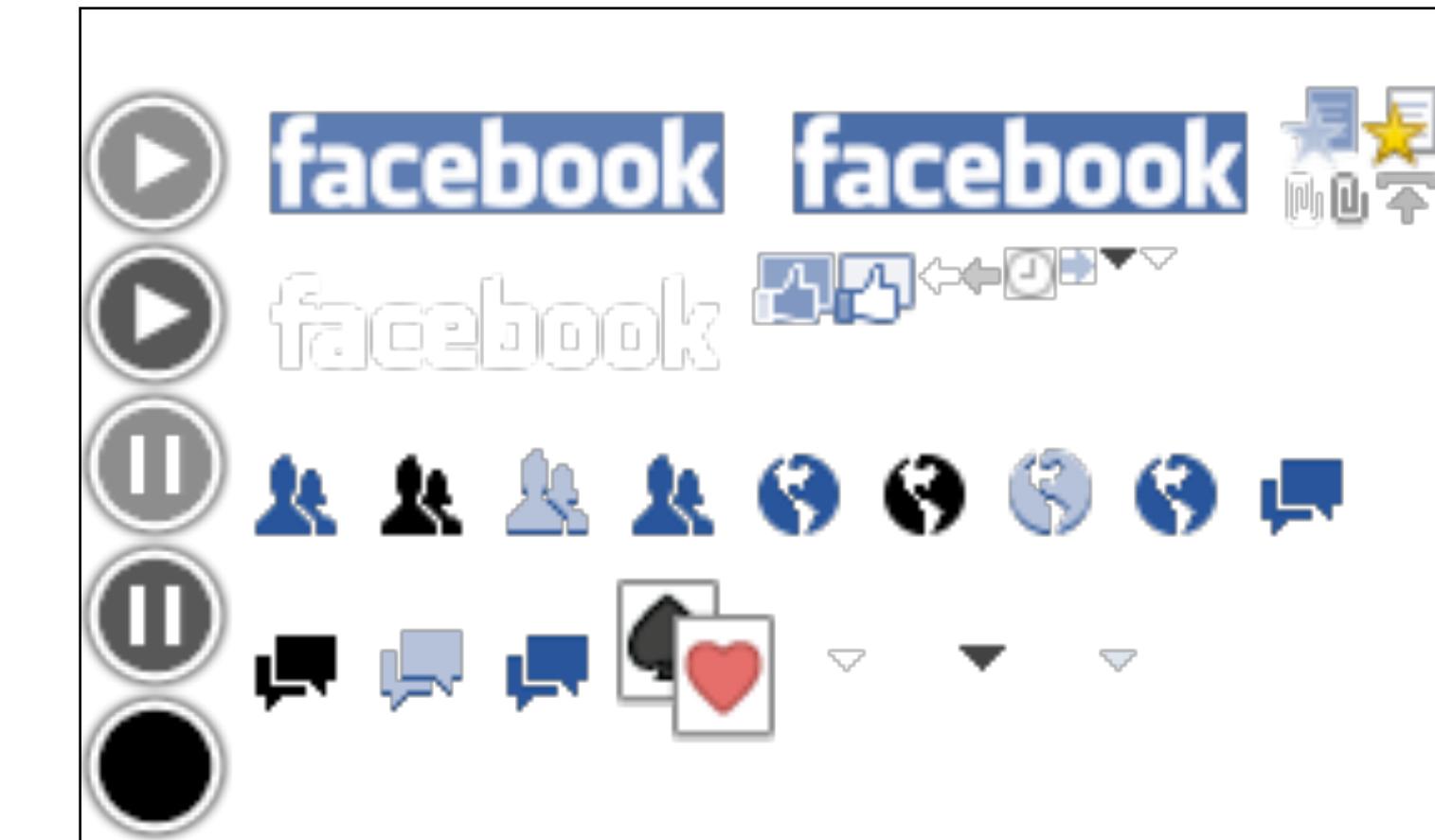
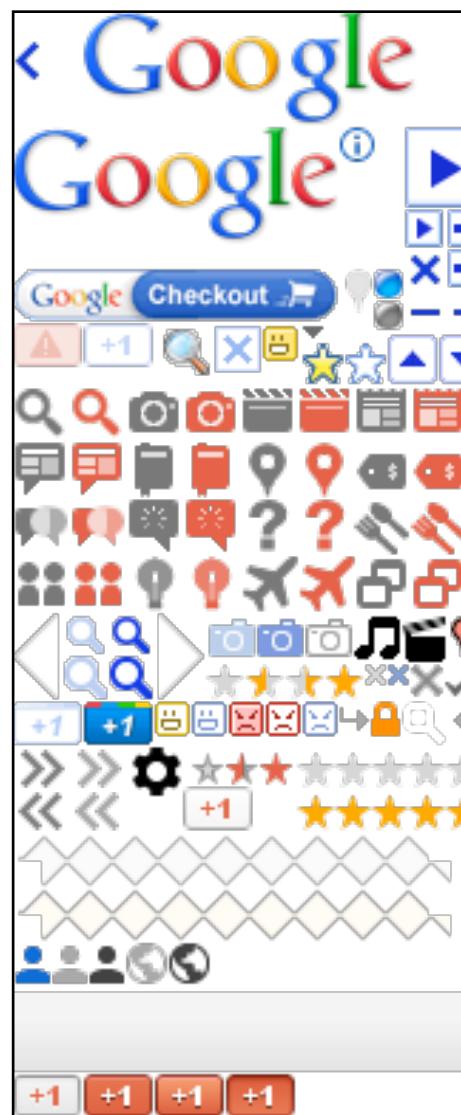
- Given that 80-90% of the time is spent making HTTP requests for all the components (images, scripts, stylesheets, etc), a simple way to reduce response time is to reduce the number of HTTP requests.
- These techniques can reduce response times by as much as 50%.
- Main techniques:
 - Image Maps
 - CSS Sprites
 - Combine Scripts and Stylesheets

Image Maps

- An image map combines multiple images into a single image.
- The overall size is about the same, but reducing the number of HTTP requests speeds up the page. Image maps only work if the images are contiguous in the page, such as a navigation bar.
- Drawbacks:
 - Defining the coordinates of image maps is tedious and error prone.
 - Has accessibility limitations, thus should be avoided for important tasks.

CSS Sprites

- Using CSS sprites, multiple images are combined into a single file and displayed using CSS rules. This is the preferred method for reducing the number of image requests.
- Drawbacks: sprites are hard to maintain.



CSS Sprites Basic Example



sprite.png



```
a {  
background: url("sprite.png")  
0px 0px no-repeat;  
}
```

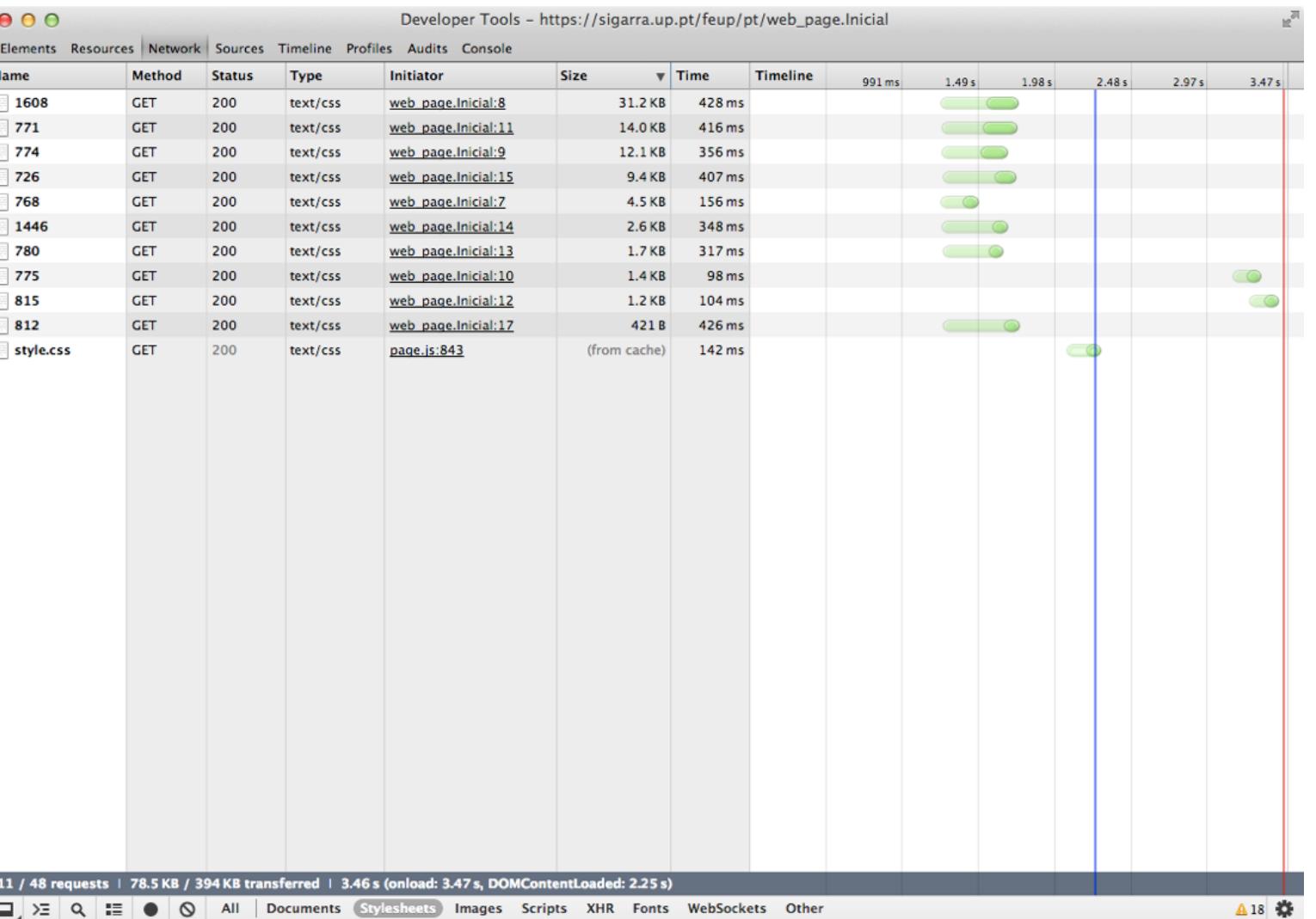


```
a:hover {  
background-position: 0px -100px;  
}
```

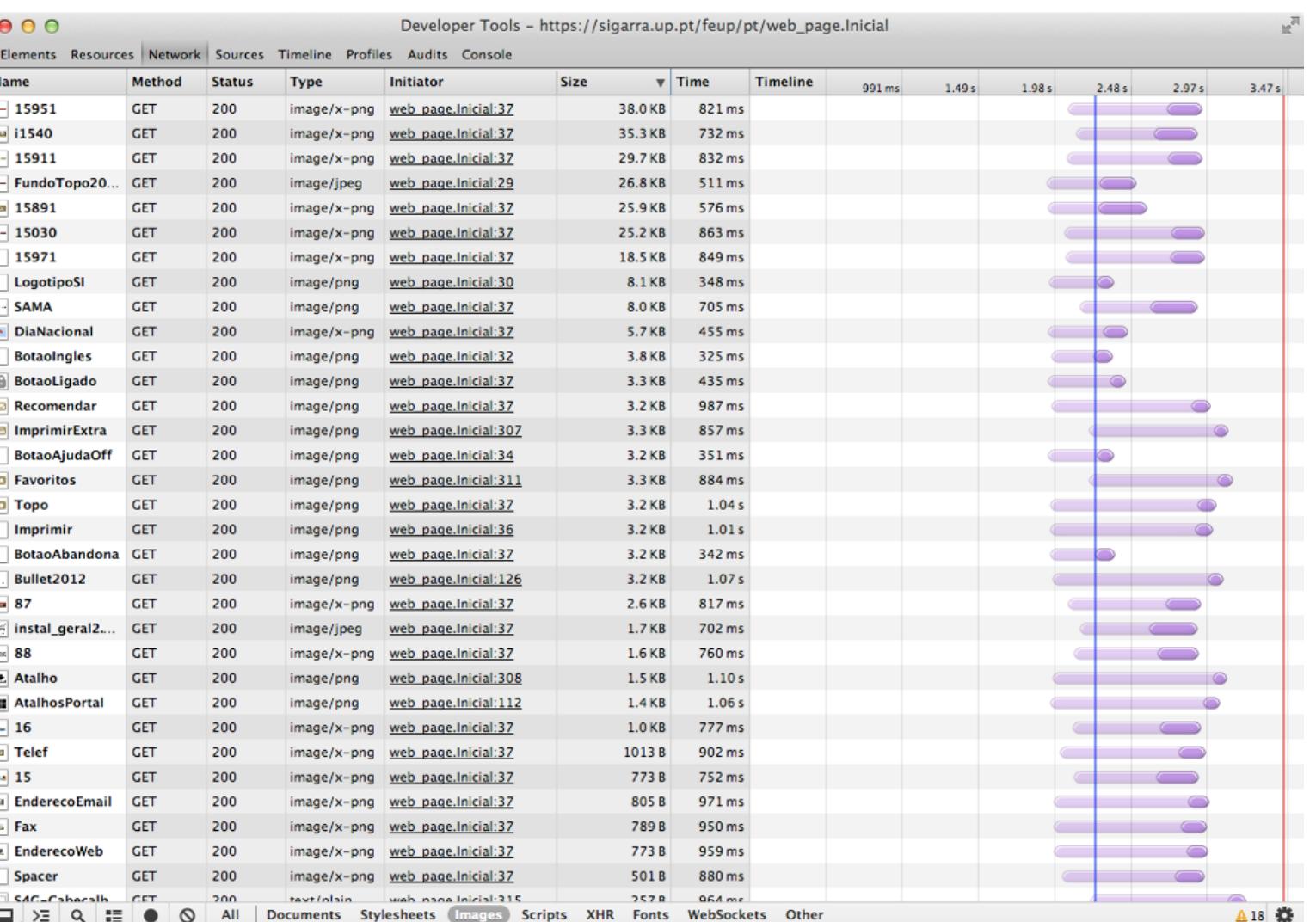
Combine Scripts and Stylesheets

- One way to reduce the number of HTTP requests, is by combining all scripts into a single script, and similarly combining all CSS into a single stylesheet.
- Might be challenging when scripts and stylesheets vary from page to page.

The screenshot shows the homepage of the Faculdade de Engenharia da Universidade do Porto (FEUP). It features a navigation bar with links to 'Sobre a FEUP', 'Órgãos de Gestão', 'Departamentos', 'Serviços', 'Estudantes', 'Pessoal', 'Cursos', 'IBD', 'Cooperação', 'Candidatos', 'Empresas', 'Notícias', 'Pesquisa', 'Autenticação', and 'Utilizador'. Below the navigation is a login form with fields for 'Senha:' and 'Validar'. A map of the university's installations is also present. The main content area includes sections for 'CANDIDATURAS 1.ª FASE' (Applications 1st Phase), 'ELEIÇÃO Conselho Geral Universidade do Porto', 'MIT Portugal APPLICATIONS ARE OPEN EXECUTIVE MASTERS UNTIL JUNE 15, 2013', 'CURSOS DE EDUCAÇÃO CONTÍNUA', and 'Últimas Notícias' (Latest News) which lists various research and institutional news items.



11 CSS resources



34 image resources

Optimize Images

- Use the right formats: JPEG for photos (lossy), PNG for graphics (lossless).
- Don't resize using with HTML/CSS.
- Optimize for the web: optimize for the web features.
- Yahoo! Smush.it – Image optimization service (lossless tool).
ysmush.it (discontinued, March 2015)
- Alternative: <http://resmush.it/>

Use a Content Delivery Network

- The user's proximity to the web server has impact on a page's response time.
- A content delivery network (CDN) is a collection of web servers distributed across multiple locations to deliver content to users more efficiently.
- CDNs are used to deliver static content, such as images, scripts, stylesheets, binaries, and Flash. Serving dynamic HTML pages involves specialized hosting requirements.
- Top CDN providers: Akamai, CloudFlare, Mirror Image, Limelight, SAWVIS.

Add an Expires Header

- A first-time visitor to a web page needs to make several requests to obtain all elements. By using a future Expires header, these components can be made cacheable, and thus re-used in following requests.
- Most commonly used with images, but should be used on all components, including scripts, stylesheets, etc.
- The Expires header is sent in the HTTP response.
- If a far future date is used (e.g. years), the filename must be changed if the component changes.

Gzip Components

- Response times can be reduced either by reducing the number of requests, or by reducing the size of the response in each request.
- Gzip encoding can be used to compress HTTP response, and thus reduce network response times.
- Using gzip generally reduces the response size by about 70%. Approximately 90% of today's Internet traffic travels through browsers that claim to support gzip.
- Configured at the web server.

Make JavaScript and CSS External

- Using inline CSS or JavaScript makes HTML documents bigger.
- Using external files results in more HTTP requests, but cacheable.
- The key factor in deciding which option is better is the frequency with which external JavaScript and CSS components are cached relative to the number of HTML documents requested.

Reduce DNS Lookups

- The Domain Name System (DNS) maps hostnames to IP addresses.
- A DNS lookup for a given hostname typically costs 20-120 milliseconds.
- DNS lookups can be reduced by using fewer hostnames (ideal: 2-4).

Minify JavaScript and CSS

- Minification is the practice of removing unnecessary characters from code to reduce its size thereby improving load times.
- When code is minified all comments are removed, as well as unneeded white space characters (space, newline, and tab). In the case of JavaScript and CSS, this improves response time performance because the size of the downloaded file is reduced.
- Popular tools:
 - JSMin – www.crockford.com/jsmin.html
 - YUI Compressor – yui.github.io/yuicompressor/
- The YUI compressor can also minify CSS.

Avoid Redirects

- Redirects are achieved using 3xx status codes, mostly 301 and 302.
- Redirects slow down the user experience since nothing in the page can be rendered and no components can start being downloaded.
- One of the most wasteful redirects happens when a trailing slash (/) is missing from a URL that should otherwise have one. For example, going to <http://example.com/tag> results in a 301 response containing a redirect to <http://example.com/tag/>.
- Although redirects degrades the user experience, it can reduce the complexity for developers in several situations.

Remove Duplicate Scripts

- It hurts performance to include the same JavaScript file twice in one page.
- Two main factors increase the odds of a script being duplicated in a single web page: team size and number of scripts.
- Hurts performance because the scripts are downloaded (in some browsers) and executed multiple times.

Configure ETags

- Entity tags (ETags) are a mechanism that web servers and browsers use to determine whether the component in the browser's cache matches the one on the origin server.
- The problem with ETags is that for a single entity there are always differences across servers (eg. file timestamps). Using multiple servers is a common situation in large web sites.
- ETags should not be used if the number of servers is larger than 1.

Make AJAX Cacheable

- Some of the previous rules also apply to AJAX components (e.g. JSON, scripts), namely:
 - Gzip Components
 - Reduce DNS lookups
 - Minify JavaScript
 - Avoid Redirects
 - Configure ETags
- A personalized response should still be cacheable by that person.

Performance Evaluation Tools

Google Performance Evaluation Tools

→ developers.google.com/speed/

→ **PageSpeed Insights**

developers.google.com/speed/pagespeed/insights/

→ PageSpeed Insights for www.fe.up.pt

→ **Google Lighthouse** - Open-source tool (Chrome, command line, online)

developers.google.com/web/tools/lighthouse/

→ **WebP** - Image format

developers.google.com/speed/webp

YSlow

→ YSlow analyzes web pages and why they're slow based on Yahoo!'s rules for high performance web sites. – <http://yslow.org>

The screenshot shows the YSlow extension interface within a browser window. The title bar indicates the extension's name and version: "chrome-extension://ninejjcohidippngpapi.lnmkgllmakh/yslow.html#290". The main area is titled "Grade C" with a yellow exclamation mark icon. Below it, it says "Overall performance score 73 Ruleset applied: YSlow(V2) URL: https://sigarra.up.pt/feup/web_page.initial". There are buttons for "Tweet" and "Share". A sidebar on the left lists various optimization rules with their grades: E, F, A, and n/a. The main content area shows a detailed breakdown of the "Make fewer HTTP requests" rule, which has a grade of E. It states: "This page has 13 external stylesheets. Try combining them into one." Below this, there is a link to "Read More" and a note about combining files, scripts, and CSS. At the bottom of the main content area, it says "Copyright © 2012 Yahoo! Inc. All rights reserved."

References

→ **Yahoo's Exceptional Performance Team**

developer.yahoo.com/performance/ [archived]

→ **Best Practices for Speeding Up Your Web Site**

developer.yahoo.com/performance/rules.html

→ **Make the Web Faster | Google Developers**

developers.google.com/speed/

→ **High Performance Web Sites**

by Steve Souders. O'Reilly 2007.

→ **Even Faster Web Sites**

by Steve Souders. O'Reilly 2009.

Product and Presentation (PA Component)

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Product and Presentation (PA) Component

PA: Product and Presentation

- This component includes the artifacts related to the delivery of the final product.
- **Deadline is the same for all groups**, first week of January — 3rd Jan.
 - But you can/should (!) submit earlier.
- *Presentation dates **tentative**: 5th and 6th of January.*
 - Still to be confirmed.

A9: Product

A9: Product

- This artifact includes:
 - implemented features;
 - updated openAPI for the product;
 - instructions for installation and use;
 - access credentials.
- The product version considered for evaluation is the one available on the group's GitLab Container Registry by the delivery deadline.
- That container will be made available at <http://lbaw22gg.lbaw.fe.up.pt> and will be used, from that URL, for the demonstration (A10). No update to the image at the production machine will be done after the submission deadline.

A9: Product

- In detail, the product should:
 - identify the development status (in %) of each of the requirements (user stories) initially set in A2 — expected high and medium priority user stories;
 - describe the web architecture of the product with an updated OpenAPI specification;
 - describe the type of assistance supported by the web application (context-sensitive help, notifications, help pages, etc.);
 - describe and briefly justify the revisions introduced to the initially defined requirements and architecture.

A9: Product Evaluation

- The product is evaluated by observing the operation of the developed web application and the delivered code.
- The following items will be assessed:
 - Product usability — accessibility, usability of interfaces and online assistance (e.g. help online).
 - The features implemented — minimum set of features considered essential.
 - The architecture and the use of technology — modularity of the solution, the quality of the source code, proper use of languages and technologies.
 - The robustness of the product — absence of failures during operation, error handling (the access to the database and business logic in PHP), form validation in HTML and JS.

A9: Checklist (1/4) - Product Information

A9. Product		
Artefact	1.1	The artefact reference and name are clear
Product information	1.2	The goal of the artefact is briefly presented (1, 2 sentences)
	2.1	List of implemented user stories are included
	2.2	All user stories have team members associated
	2.3	All user stories have completion state information (as a percentage)
	2.4	Revision history section with changes to A2 and A7 is included
	2.5	Link to the final release source code is included
	2.6	Docker command to start the image from GitLab Container Registry is included
	2.7	URL to the product at the production machine is included
	2.8	Credentials for different roles are included
	2.9	"README.md" is updated with the Docker command, URL and credentials

A9: Checklist (2/4) - Features

Features	<ul style="list-style-type: none">3.1 Contextual help is supported3.2 About page is implemented3.3 Input validation on the client (frontend) is implemented3.4 Input validation on the server (backend) is implemented3.5 All high and medium priority user stories are implemented3.6 User authentication is implemented3.7 User registration is implemented3.8 User account deletion is implemented3.9 Password recovery is implemented3.10 User access control is implemented3.11 Administration features are implemented3.12 Full-text search is implemented3.13 Full-text search with multiple weighted fields is implemented3.14 Advanced search using filters is implemented3.15 AJAX interactions are implemented3.16 Feedback messages (e.g. errors, notifications) are implemented
----------	---

A9: Checklist (3/4) - Architecture and Technologies

Architecture and technologies	4.1	OpenAPI specification in YAML for the final product is included
	4.2	All non-essential LBAW template code was removed
	4.3	List of additional libraries or packages used is included
	4.4	Non-authorized libraries or packages are not used
	4.5	Complex code blocks are explained with comments
	4.6	Unnecessary code removed (e.g. commented code, debug code)
	4.7	No temporary files are left on the final product
	4.8	Laravel routes are correctly used
	4.9	Laravel controllers are correctly used
	4.10	Laravel templates are correctly used
	4.11	Laravel data access is correctly used
	4.12	Laravel policies are correctly used
	4.13	HTML validation report is included
	4.14	CSS validation report is included
	4.15	HTML semantic elements are used
	4.16	Minimal redundancy in CSS used (i.e. use cascading features)
	4.17	Product not vulnerable to URL manipulation
	4.18	Product not vulnerable to user input manipulation
	4.19	Internal errors are handled
	4.20	Laravel APP_DEBUG variable is set to TRUE in .env

A9: Checklist (4/4) - Product Usability and Accessibility

Product usability and accessibility	5.1	Usability checklist report is included
	5.2	Accessibility checklist report is included
	5.3	Responsive design is implemented

A10: Presentation and Discussion

A10: Presentation and Discussion

- This artifact includes the promotion, a presentation, the demonstration of the product, and a discussion of the developed features.
- The promotion should include a short 2-minutes video, showcasing the main features of the product.
- For the presentation, the group must follow a presentation script (not to be delivered), and demonstrate the product accessing the official URL at the production machine: <https://lbaw22gg.lbaw.fe.up.pt>. The web application will be available until the grades are published.
- Each group has a 20 minutes slot: 15 min for demonstration + 5 min for questions.

A10: Checklist (1/2)

A10. Presentation		
Artefact	1.1	The artefact reference and name are clear
	1.2	The goal of the artefact is briefly presented (1, 2 sentences)
Materials	2.1	Brief product presentation is included
	2.2	URL to the product is included
	2.3	URL to the video is included
	2.4	Video includes textual or audio descriptions
	2.5	Product data is realistic (e.g. no 'lorem ipsum' style)

A10: Checklist (2/2)

Presentation	3.1	Application is readily available in production
	3.2	The presentation is well structured
	3.3	Communication is clear
	3.4	The time limit is not exceeded
	3.5	Main features are shown
	3.6	No bugs nor unexpected fails happen
	3.7	Answers during the discussion are clear
Features Presented	4.1	Authentication & Authorisation
	4.2	CRUD operations
	4.3	User input validation on client (HTML, JS)
	4.4	User input validation on server
	4.5	User account creation
	4.6	User account deletion
	4.7	Password recovery
	4.8	Help features (contextual and others)
	4.9	AJAX calls
	4.10	Full-text search
	4.11	User administration

Client-Side Development Topics

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Client-Side Frameworks
- Vue.js Overview
- Client-Side Tools

Client-Side Frameworks

Client-Side Frameworks

- Software frameworks are designed to speed-up software development by providing pre-configured selection of libraries and configurations on how to use specific technologies.
- Frameworks exist for HTML, CSS, and JavaScript.
 - **HTML frameworks** include <https://html5boilerplate.com>.
 - **CSS frameworks** include getbootstrap.com, get.foundation, tailwindcss.com.
 - **JavaScript frameworks** include angularjs.org, reactjs.org, svelte.dev, vuejs.org.

JavaScript Frameworks

- Modern web applications make extensive use of JavaScript (used on ~95% of all web sites).
- The central problem that JavaScript frameworks address is
 - managing application state and synchronizing the user interface.
- JavaScript frameworks typically support:
 - Developer tools and pipelines;
 - Client-side routing;
 - Client-side templating, e.g. Handlebars;
 - Domain-specific languages, e.g. TypeScript;
 - Components, e.g. JSX (JavaScript and XML);
 - Dependency management.

Vue.js Example

Vue.js

- Vue.js is a client-side JavaScript framework.
- The initial release was in 2014 and is published under a MIT License.
- Vue.js is written in TypeScript and provides first-class TypeScript support.
- Compared to other JavaScript frameworks, Vue.js can be used to enhance existing HTML – providing facilities for progressive enhancement.

Installation

- Vue.js can be installed from a CDN.
 - From: [unpkg](#), [jsdelivr](#), [cdnjs](#)
 - Latest version
 - <https://unpkg.com/vue>
 - Specific versions
 - <https://unpkg.com/vue@3>
 - Production version
 - <https://unpkg.com/vue@3/dist/vue.global.prod.js>
- Or locally, using [npm](#) and [create-vue](#), Vue's scaffolding tool.

Without Build Step (plug-in library)

Hello World

```
<!DOCTYPE html>
<html>
  <head>
    <title>Vue.js Hello World</title>
    <script src="https://unpkg.com/vue"></script>
  </head>
  <body>
    <div id="app">{{ message }}</div>
  </body>

  <script>
    const { createApp } = Vue

    createApp({
      data() {
        return {
          message: 'Hello Vue!'
        }
      }
    }).mount('#app')
  </script>
</html>
```

Binding User Data and Forms

- `v-model` is a directive that creates a two-way data binding between a value in the template and the data properties – when data changes, the input will change; if input changes, the data will change.

```
<body>
  <div id="app">
    <label for="name">Name:</label>
    <input id="name" type="text" v-model="name" />
    <p>Name variable: {{ name }}</p>
    <p>Again the name var: {{ name }}</p>
  </div>
</body>
```

```
<script>
  const { createApp } = Vue

  createApp({
    data() {
      return {
        name: ''
      }
    }
  }).mount('#app')

</script>
```

Binding Properties Manipulation

- Element properties can be changed with the `v-bind` directive.
- The class property can be changed with `v-bind:class` or the shorthand `:class`.
- Class `red` will be present if the `active` property is `true`.

```
<style>
  .red {
    color: red;
  }
</style>

<body>
  <div id="app">
    <button @click="active = !active">Toggle me</button>
    <p :class="{ red: active }">Paragraph with toggling styles.</p>
  </div>
</body>
```

```
<script>
  const { createApp } = Vue

  createApp({
    data() {
      return {
        active: false
      }
    }
  }).mount('#app')
</script>
```

Event Handling

- **v-on**, or the shortcut **@**, is used to capture DOM events and run JavaScript code.
 - **v-on:click="handler"**
 - **@click="handler"**

```
<body>
  <div id="app">
    <button @click="count++>Add +1</button>
    <p>Count is currently {{ count }}</p>
  </div>
</body>
```

```
<script>
  const { createApp } = Vue

  createApp({
    data() {
      return {
        count: 0
      }
    }
  }).mount('#app')
</script>
```

Other Directives

- Conditional rendering:
 - v-if v-else v-else-if**

```
<button @click="awesome = !awesome">Toggle</button>

<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no!</h1>
```

- List rendering:
 - v-for**

```
data() {
  return {
    items: [{ message: 'Foo' }, { message: 'Bar' }]
  }
}
```

```
<li v-for="item in items">
  {{ item.message }}
</li>
```

Components

- Vue.js components can be used to build modular interfaces.

```
<script type="module">
  import SpecialParagraph from "./SpecialParagraph.js";

  const app = Vue.createApp({
    components: {SpecialParagraph},
    template: `
      <h1>Components</h1>
      <SpecialParagraph></SpecialParagraph>
    `,
    ).mount("#app");
</script>
```

```
export default {
  data() {
    return {
      msg: "Hello world from component"
    }
  },
  template: `
<p>
  <strong>{{msg}}</strong>
</p>
  `
}
```

SpecialParagraph.js

Client-Side Tools

Developer Tools

- Developer tools play an essential role in modern development environments.
- Tools can be structured in three broad categories:
 - Tools design to support **code development**.
 - Tools to **transform code** between different representations (CSS, JavaScript).
 - Tools for **testing or deployment**, after code is written.

Code Development

- **Source code control**, for backup and team work (e.g. Git).
- **Browser developer tools**, for code inspection and debugging.
- **Linters** check through the code to highlight existing errors and guidelines violations.
 - ESLint, for JavaScript, <https://eslint.org>
 - csslint, for CSS, <http://csslint.net>
- **Bundlers/Packages** prepare the code for production, e.g. remove non-used code, minify code for deployment.
 - Parcel, <https://parceljs.org>
 - Webpack, <https://webpack.js.org>

Code Transformation

- Code transformation tools allow developers to use
 - the latest language features, i.e. “future code”,
 - other development languages, e.g. TypeScript.
- Transformation tools will generate browser-compatible code to be used in production.
- There are code transformation tools for HTML, CS and JavaScript.

Use Latest Language Features

- Write code using the latest language features and have that code transformed into code that works in older devices.
- **Babel**, is a JavaScript compiler that transforms code using the latest JavaScript features to old JavaScript code, thus supported in more browsers.
 - <https://babeljs.io>
- **PostCSS**, is a similar tool for CSS, i.e. transforms modern CSS to older browser-supported CSS code.
 - <https://postcss.org>

Use Another Language

- Write code in a different language and transform it into a web standard language.
- **Sass/SCSS**, a stylesheet language that is compiled to CSS.
 - <https://sass-lang.com>
- **TypeScript**, a superset of JavaScript that offers additional features.
 - <https://www.typescriptlang.org>

Testing and Development

- **Testing tools** can automatically run tests against the code before moving further in the production pipeline (e.g. unit testing).
 - Popular tools: www.travis-ci.com, www.jenkins.io
- **Deployment tools** are used to automatically publish a web application.
 - Popular tools: pages.github.com

References

- MDN Web Docs, developer.mozilla.org
 - Introduction to client-side frameworks, [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Learn/Client-side/Frameworks)
 - Getting started with Vue, [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Learn/Client-side/Frameworks/Vue.js)
- Vue.js
 - vuejs.org
 - Vue.js Guide, vuejs.org/guide/introduction.html
 - vue-community.org

Web Accessibility and Web Usability

Databases and Web Applications Laboratory (LBAW)
Bachelor in Informatics Engineering and Computation (L.EIC)

Sérgio Nunes
Dept. Informatics Engineering
FEUP · U.Porto

Outline

- Web Accessibility
- Web Usability
- Web Usability Patterns
- Web Usability Techniques

Web Accessibility

Web sites should be designed to ensure that everyone, including users with disabilities, can use them.

The World Wide Web is a medium that poses a new opportunity to improve content accessibility.

Why?

- Access to information and communication technologies is recognized as a **basic human right** by the United Nations.
- Promote **equal access and equal opportunity** to people with diverse abilities.
- Accessibility **supports social inclusion** for people with disabilities as well as others, such as older people, people in rural areas or people in developing countries.
- It is **mandatory by law** for public services in many countries.

Common Disabilities

- **Vision:** Blindness, Low vision, Color-blindness.
- **Hearing:** Deafness.
- **Motor:** Inability to use a mouse, Limited fine motor control skills.
- **Cognitive:** Learning disabilities, Inability to remember information.

Assistive Technologies

- **Screen readers**
- **Screen magnification**
- **Text enlargers**
- **Alternative input devices:** voice recognition, eye tracking.
- [en.wikipedia.org/wiki/Assistive_technology]

W3C Web Accessibility Initiative (WAI)

- "The Web Accessibility Initiative (WAI) develops strategies, guidelines, and resources to help make the Web accessible to people with disabilities."
<http://www.w3.org/WAI/>
- This initiative was launched in 1997 and has several groups working on guidelines, technical reports, educational materials, and other documents related with web accessibility.
- The Web Content Accessibility Guidelines 1.0 were published by the W3C in 1999. The WCAG is a set of guidelines for making web content more accessible to persons with disabilities.

Accessibility Guidelines

- Design forms for users using assistive technologies
- Do not use color alone to convey information
- Enable users to skip repetitive navigation links
- Provide text equivalents for non-text elements
- Do not require style-sheets for content readability
- Ensure that scripts allow accessibility

SAPO Accessibility Checklist

→ <https://ux.sapo.pt/checklists/acessibilidade/>

The screenshot shows the SAPO UX website interface. At the top, there's a dark header with the 'SAPOUX' logo and a navigation bar with links for 'USABILIDADE', 'ACESSIBILIDADE', 'SEO', and 'CHECKLISTS'. The 'CHECKLISTS' link is highlighted in orange. To the right of the navigation is a search bar with a magnifying glass icon. Below the header, on the left, is a sidebar with four items: 'USABILIDADE (WEB)', 'USABILIDADE (MOBILE)', 'ACESSIBILIDADE' (which is bolded), and 'SEO'. The 'ACESSIBILIDADE' item has a right-pointing arrow next to it. The main content area starts with a section titled 'CHECKLIST DE PONTOS DE VERIFICAÇÃO DE: Acessibilidade'. It contains text explaining what a checklist is and how it can be used. Below this, under the heading 'Elementos não textuais', there's a list item with a checkbox and text about image alt attributes. At the bottom of the page, a footer bar displays the text 'A SUA PONTUAÇÃO: 0/18'.

Accessibility Resources

→ **W3C Web Accessibility Initiative (WAI)**

www.w3.org/WAI/

→ **Usability.gov**

www.usability.gov

→ **WebAIM: Web Accessibility in Mind**

webaim.org/

→ **acessibilidade.gov.pt**

www.acessibilidade.gov.pt/

Web Usability

Usability is about creating effective user interfaces.

What is Usability?

- Usability corresponds to a combination of factors that affect the user's experience with the application, including:
 - Ease of Learning
 - Efficiency of Use
 - Memorability
 - Error Frequency and Severity
 - Subjective Satisfaction

Challenges in Web Usability

- Wide range of users, both in experience and in expectations
- Wide range of platforms and technologies
- User's experience is formed on other web sites
- Web users have a very low tolerance for poor usability
- Other products are only a click away

Top Ten Guidelines for Homepage Usability (2002)

1. Include a One-Sentence Tagline
2. Write a Window Title with Good Visibility in Search Engines and Bookmark Lists
3. Group all Corporate Information in One Distinct Area
4. Emphasize the Site's Top High-Priority Tasks
5. Include a Search Input Box
6. Show Examples of Real Site Content
7. Begin Link Names with the Most Important Keyword
8. Offer Easy Access to Recent Homepage Features
9. Don't Over-Format Critical Content, Such as Navigation Areas
10. Use Meaningful Graphics

Top Ten Mistakes in Web Usability (1999)

1. Using Frames
2. Gratuitous Use of Bleeding-Edge Technology
3. Scrolling Text, Marquees, and Constantly Running Animations
4. Complex URLs
5. Orphan Pages
6. Long Scrolling Pages
7. Lack of Navigation Support
8. Non-Standard Link Colors
9. Outdated Information
10. Overly Long Download Times

Top Ten Web Design Mistakes of 2005

1. Legibility Problems — too small. no contrast.
2. Non-Standard Links — identifiable, differentiate visited.
3. Flash — bad use of technology.
4. Content That's Not Written for the Web — short, scannable, to the point.
5. Bad Search — users expect search functions.
6. Browser Incompatibility — don't turn away users that use a different platform.
7. Cumbersome Forms — usually too big. avoid mandatory fields.
8. No Contact Information or Company Information — most common use case.
9. Frozen Layouts with Fixed Page Widths — problems with big/small monitors.
10. Pop-ups — most hated advertising technique.

(Some) Basic Principles for Interface Design, Bruce Tognazzini

→ Anticipation

Interfaces should attempt to anticipate the user's wants and needs.

→ Consistency

- Make objects consistent with their behavior.
- Object that act differently should look different.

→ Efficiency of the User

- Design for user productivity, not computer productivity.

→ Explorable Interfaces

- Make actions reversible. Always allow a way out.

→ Fitts' Law

The time to acquire a target is a function of the distance to and size of the target.

→ Latency Reduction

- Reduce the user's experience of latency. Make it work faster.

→ Track State

- Track the user state as needed.

→ Visible Navigation

- Avoid invisible navigation.

SAPO Usability Checklist

→ <https://ux.sapo.pt/checklists/usabilidade/>

The screenshot shows the SAPO UX website interface. The header features the SAPOUX logo and a navigation bar with links for Home, Usabilidade, Acessibilidade, SEO, and CHECKLISTS (which is highlighted). A search bar is also present. On the left, a sidebar lists categories: USABILIDADE (WEB), USABILIDADE (MOBILE), ACESSESSIBILIDADE, and SEO. The main content area is titled "CHECKLIST DE PONTOS DE VERIFICAÇÃO DE: Usabilidade (Web)". It describes what a web usability checklist is and how it can be used. Below this, a section titled "Navegação e Feedback" contains a checklist item: "É sempre fornecido feedback sobre as ações do utilizador". The text explains that users should receive immediate feedback about their actions. At the bottom of this section, a note states "A SUA PONTUAÇÃO: 0/28".

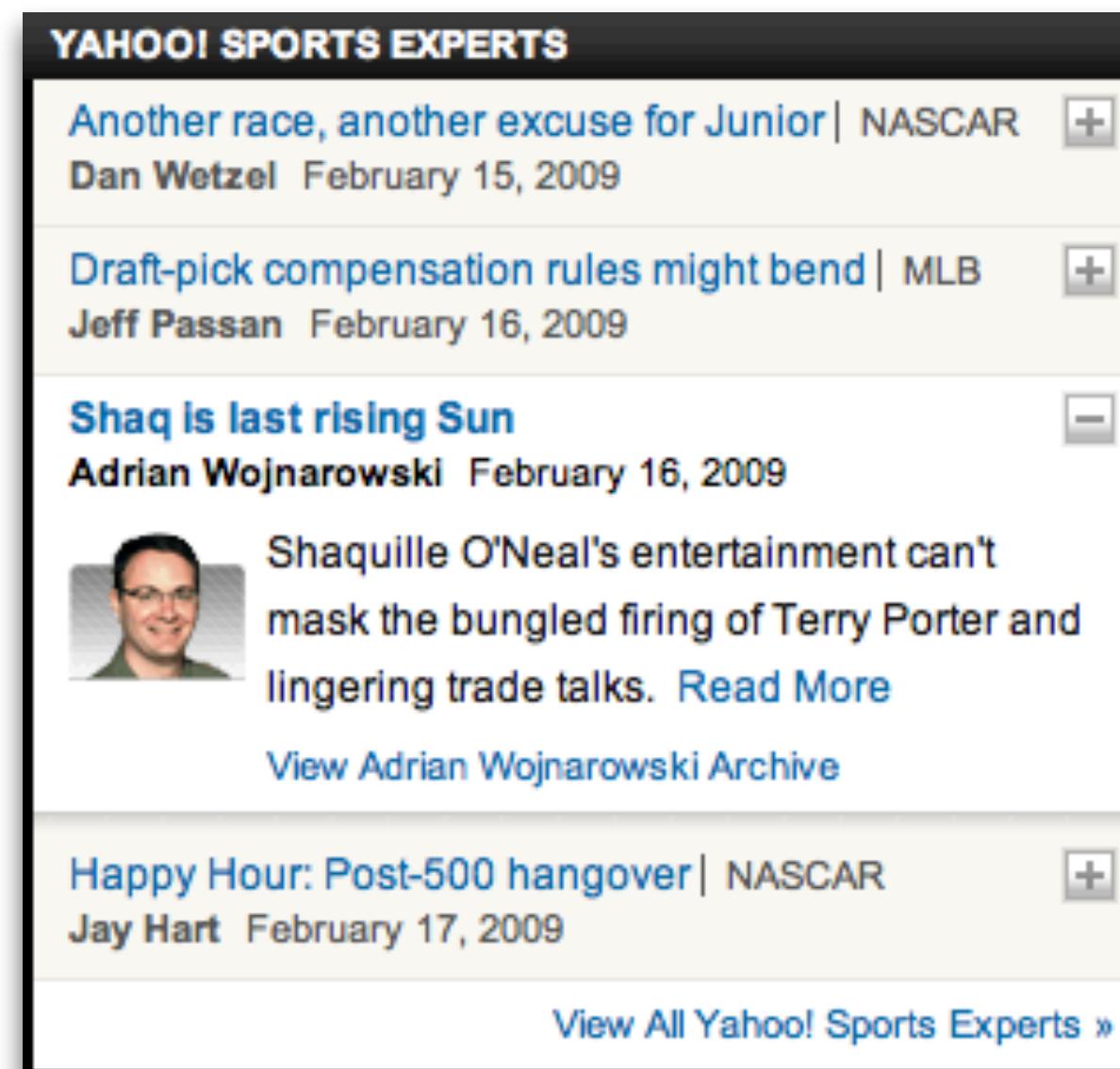
Web Design Patterns

- Web Design Patterns are recurring solutions that solve common web design problems.
- User Interface Design Patterns — ui-patterns.com
- Patterns in Interaction Design — www.welie.com/patterns [archived]
- Yahoo! Design Pattern Library — developers.yahoo.com/ypatterns [archived]

Yahoo! Design Patterns

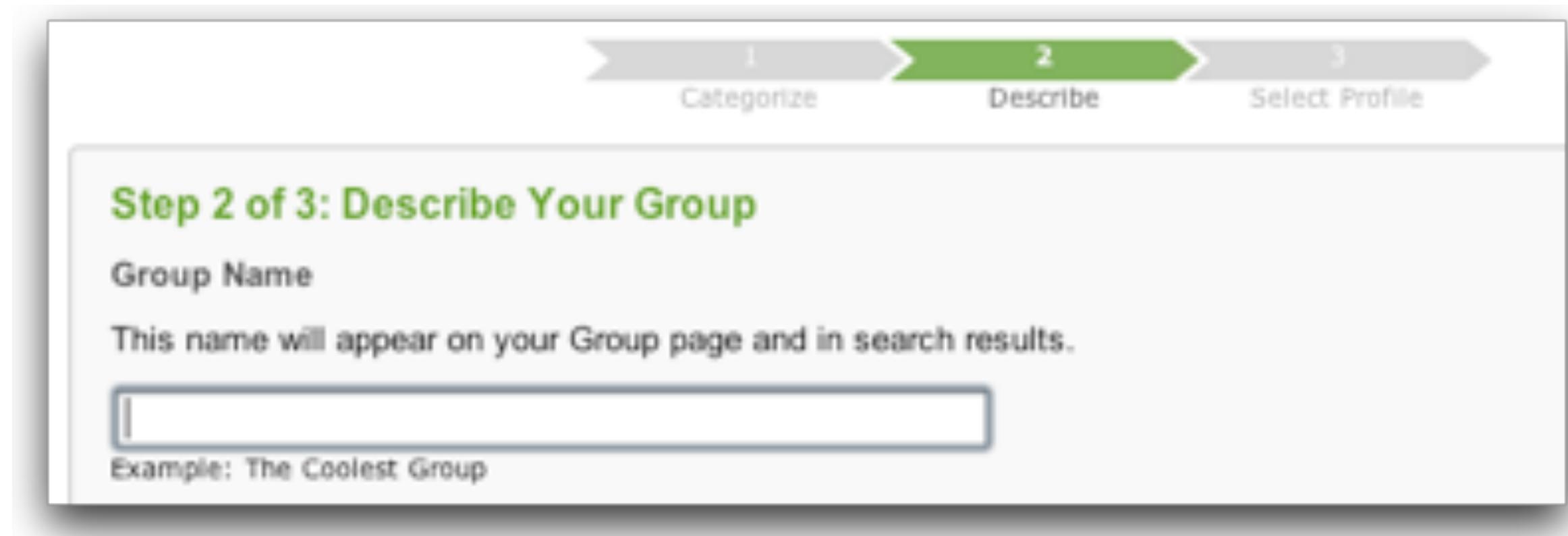
Accordion

- "An accordion is a grouped set of collapsible panels that provides access to a large number of links or other selectable items in a constrained space.
- "Use when the number of options is large, the space is constrained, and the list of items can be logically grouped into smaller, roughly equal sized chunks.



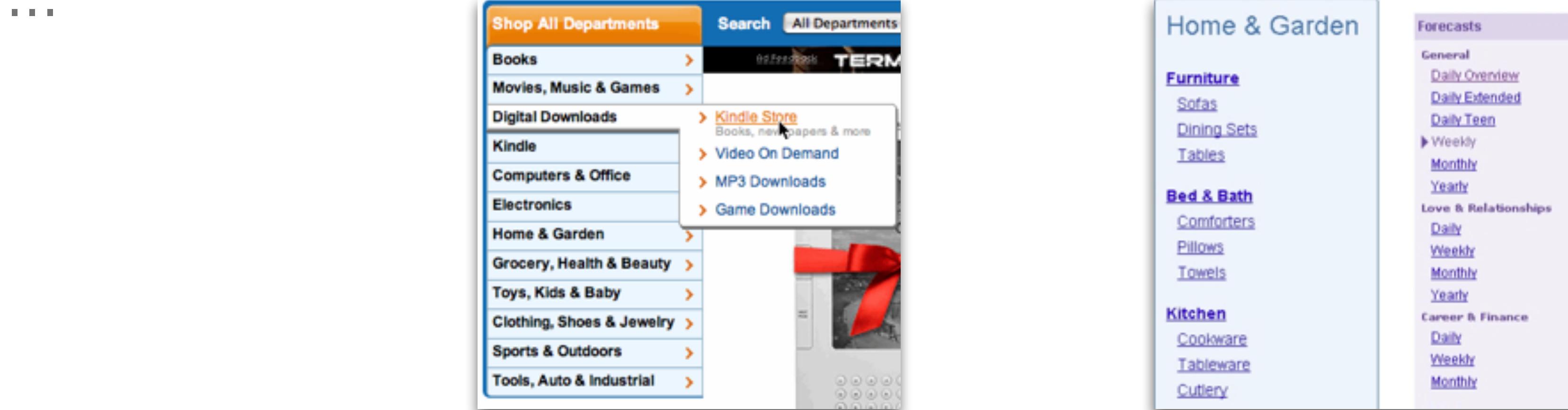
Progress Bar

- "A progress bar can help set expectations for length of process and for what to generally expect throughout the process, and can also let users know where they are in the flow.
- "Use a progress bar in a wizard or other predefined multistep process that the user may only ever have to complete one time, or at most on rare occasions.



Left Navigation Bar

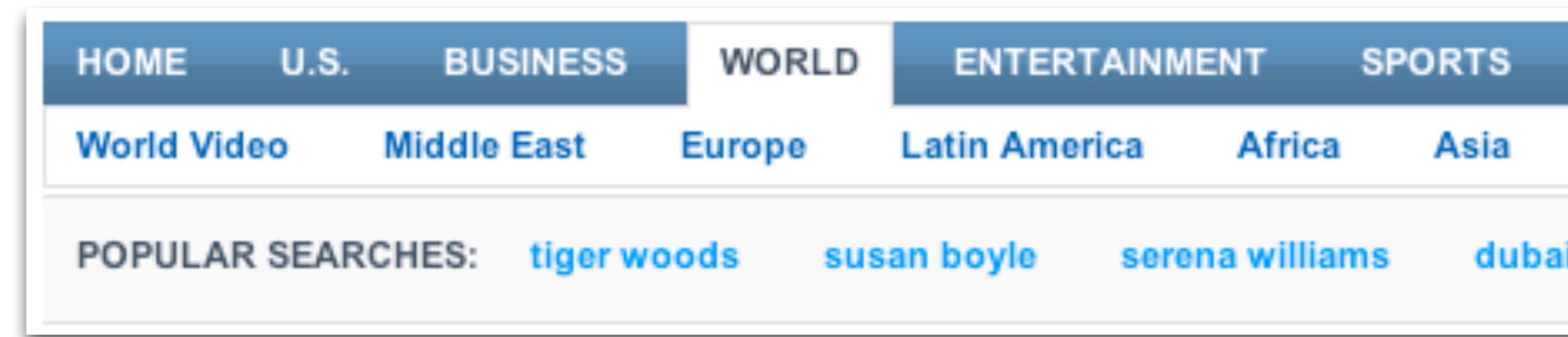
- "A left navigation bar provides quick access to categorized content in a horizontally compact display space.
- "Use when there are at least 4 categories.
Use when there may be more than 10 categories.
Use when page width is not an issue.



Top Navigation Bar

- "A top navigation bar provides quick access to categorized content in a vertically compact display space.
- "Use when there are 2-12 category titles
 - Use when the category titles are relatively short and predictable
 - Use when the number of categories are not likely to change.
 - Use to present the highest-level navigation options on a single web product.

...



Item Pagination

- "The user needs to view a subset of data that will not be easy to display within a single page.
- "There is more information than can comfortably fit within one screen.
The items of interest can be usually found on the first few pages.
If the data needs to be explored deeply, consider displaying the content in a scrolled area instead.



ObjectNames 1-10 of 27 First ◀ Prev ◀ ▶ Next ▶ Last

Popular Interaction Patterns

→ Lazy Registration

Let the user take actions before forcing them to register, e.g. Amazon.

→ Progressive Disclosure

Show most relevant features or information to user, and delay further details or complex features until requested, e.g. Facebook.

→ Breadcrumbs

Clearly show the path from the front page to the current location.

→ Account Registration

Require only necessary information.

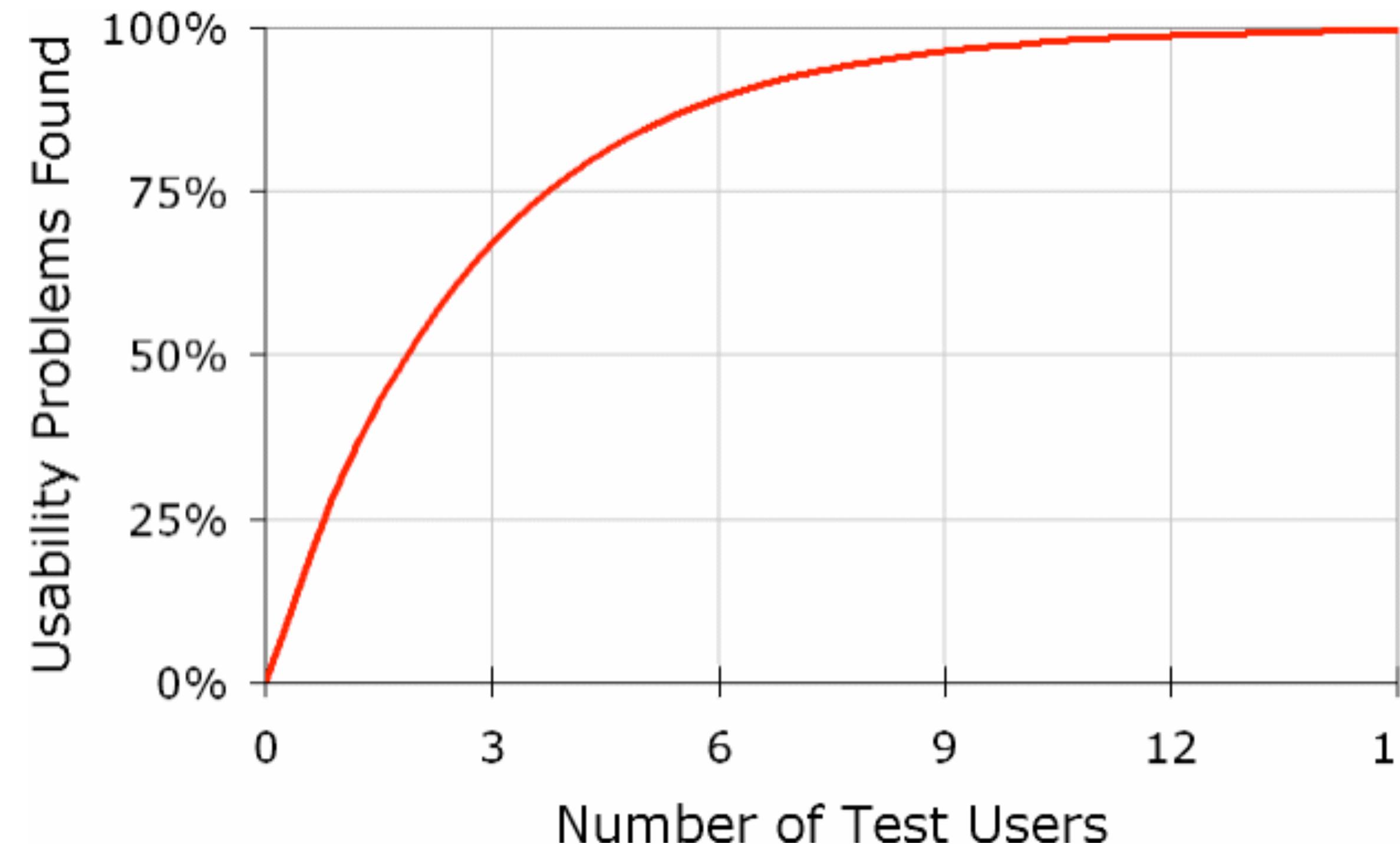
→ Required Field Marker

Make obvious the required fields in any form.

Usability Techniques

Web Usability Testing

→ Testing with only 5 users will find > 80% of the problems.



Discount Usability

→ Simplify User Testing

Handful of participants, a focus on qualitative studies, and use of the think-aloud method.

→ Narrowed-down Prototypes

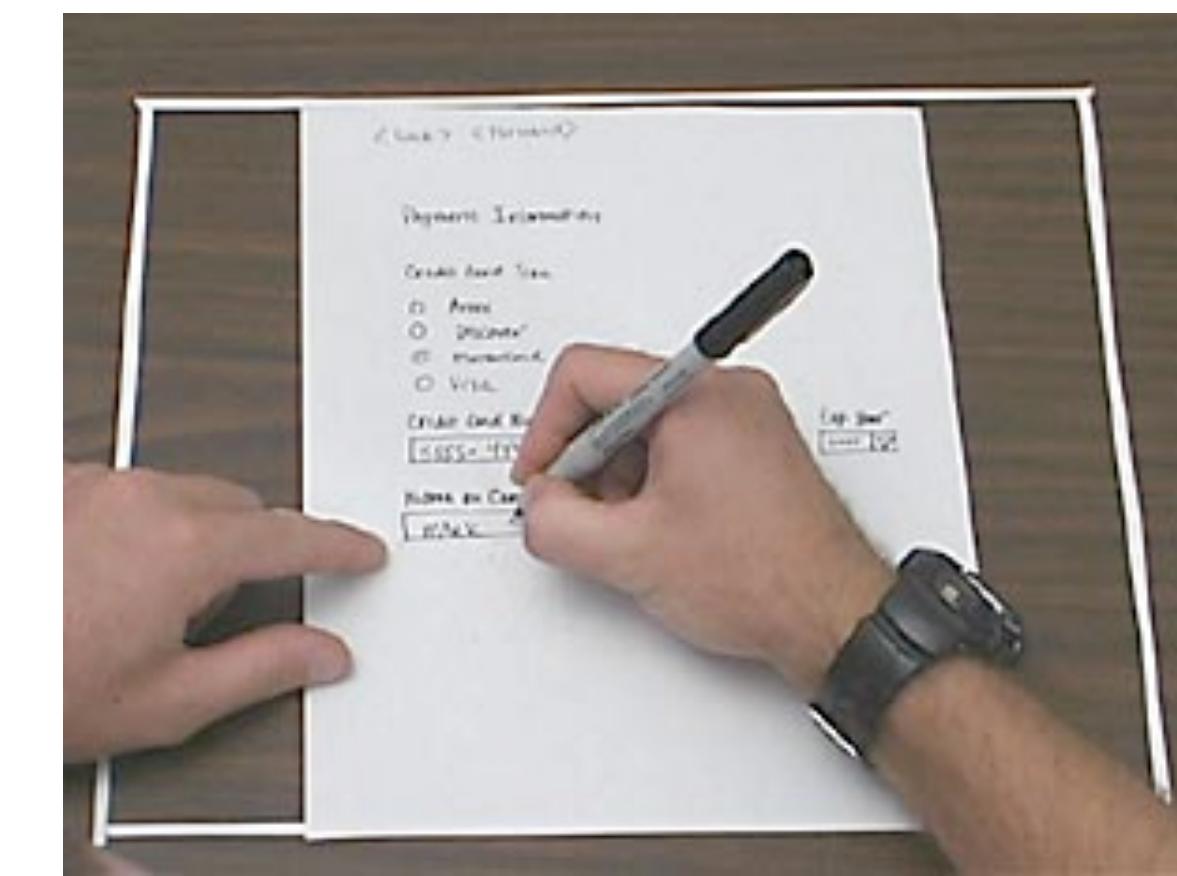
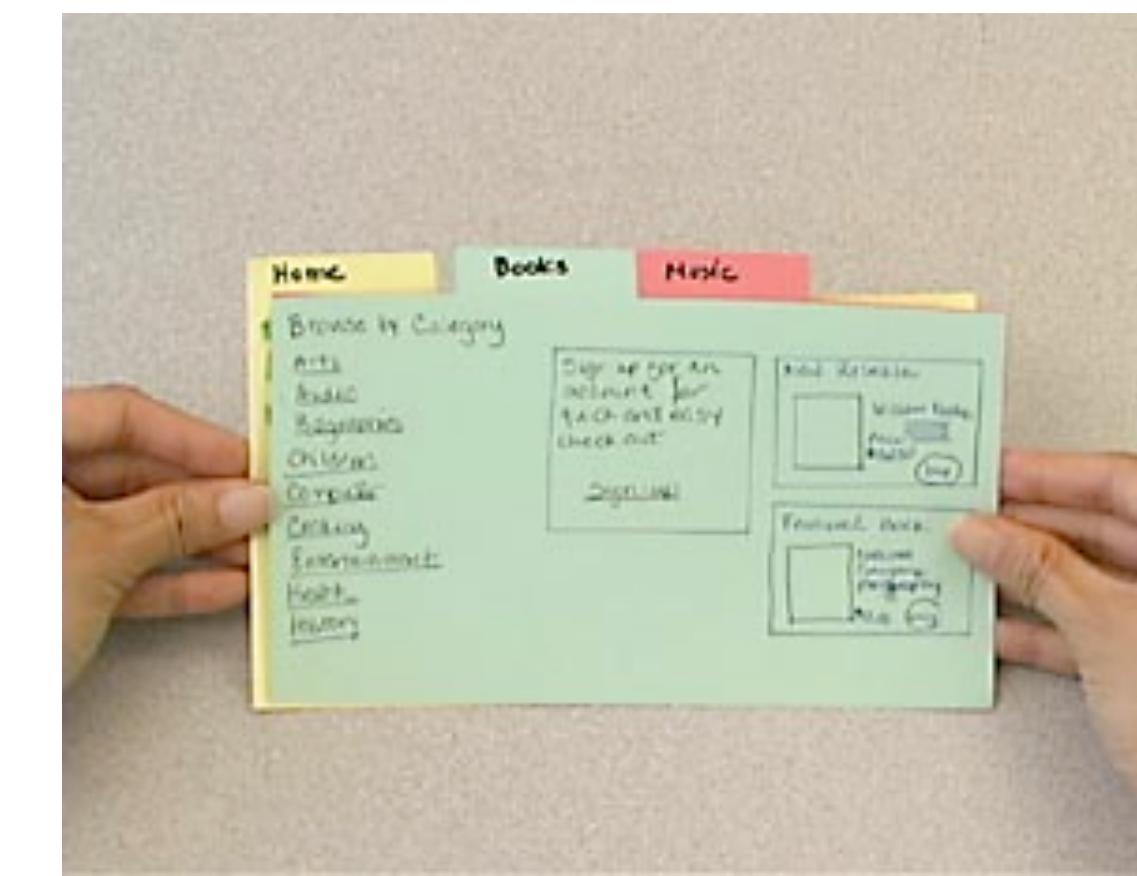
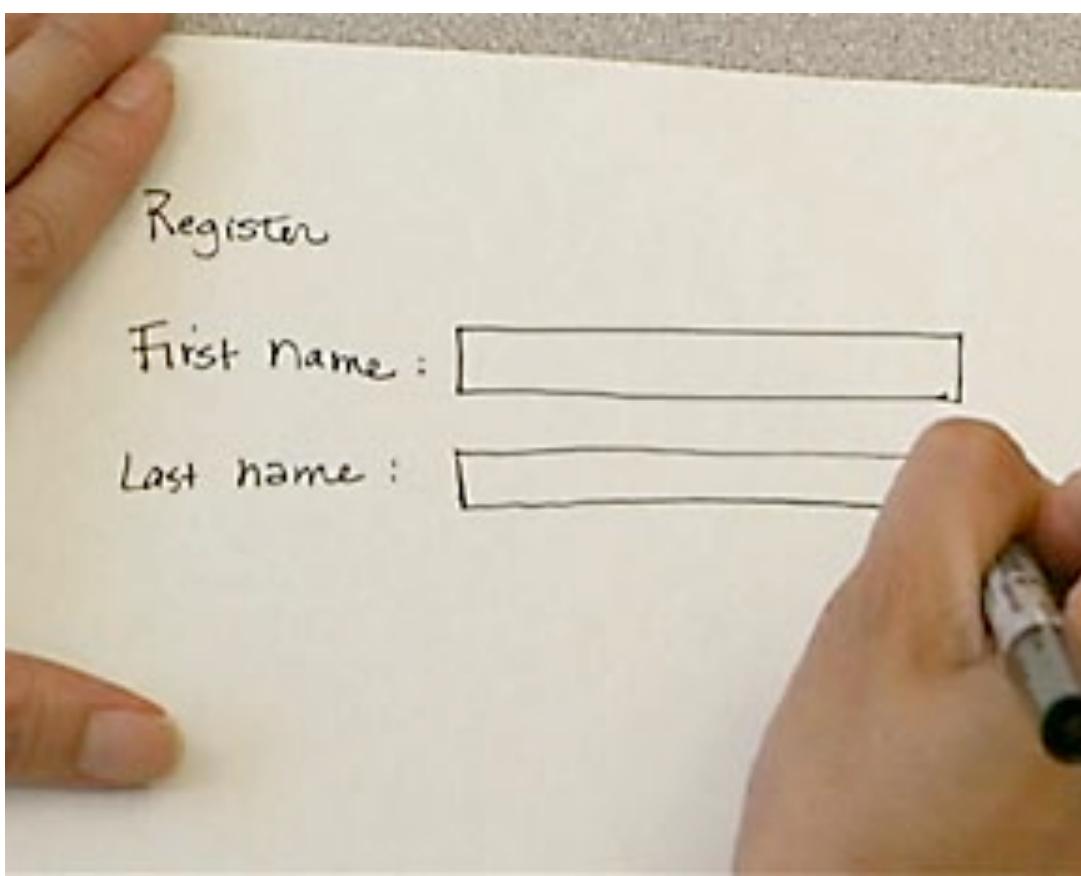
Use narrowed-down prototypes, such as paper prototypes, for user testing.

→ Heuristic Evaluation

Inspect user interfaces by comparing them to established usability guidelines.

Paper Prototyping

- Paper prototyping is one of the fastest and cheapest techniques that can be employed in a user-centered design process.
- The biggest improvements in user experience comes from gathering usability data as early as possible. It is estimated to be 100 cheaper to make a change before any code has been written than after the implementation.

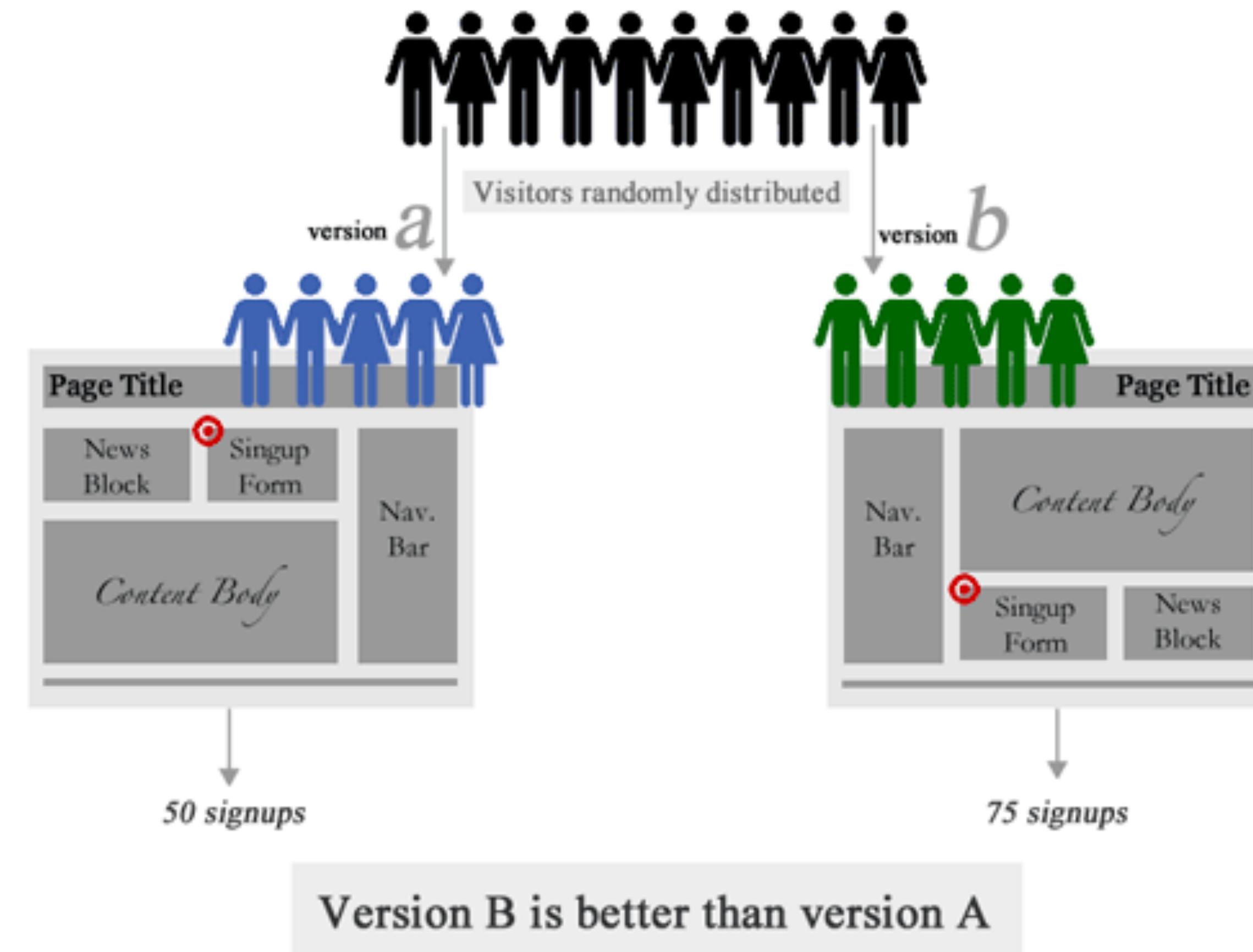


www.nngroup.com/articles/web-discount-usability/

A/B Testing

- Randomized controlled experiments to evaluate the effects of two different versions (A and B) of a given web site, page or interaction. Two competing designs are shown to real users.
- Main benefits: measures actual user behavior, can measure very small performance differences, results in objective measures, is cheap.
- Limitations: requires a clear, well-defined goal (e.g. sales, subscriptions, clicks), not everything is measured, requires fully implemented designs.

A/B Testing



Source: Smashing Magazine

Usability References

- **Don't Make Me Think! A Common Sense Approach to Web Usability** (2nd)
Steve Krug. New Riders (2005)
- **The Research-Based Web Design & Usability Guidelines**
guidelines.usability.gov
- **The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience**
Douglas K. van Duyne, James A. Landay, Jason I. Hong. Addison-Wesley (2002)
- **Designing for the Web**
Mark Boulton (2010)
designingfortheweb.co.uk