Web Applications

Software Engineering 2017

Alessio Gambi - Saarland University

Based on the work of Cesare Pautasso, Christoph Dorn, Andrea Arcuri, and others

ReCap

Software Architecture

A software system's architecture is the set of principal design decisions made about the system.

N. Taylor et al.

Abstraction	Communication
Visualization and Representation	Quality Attributes

Every system a software architecture has

Prescriptive Architecture

Descriptive Architecture

Realization

Intent

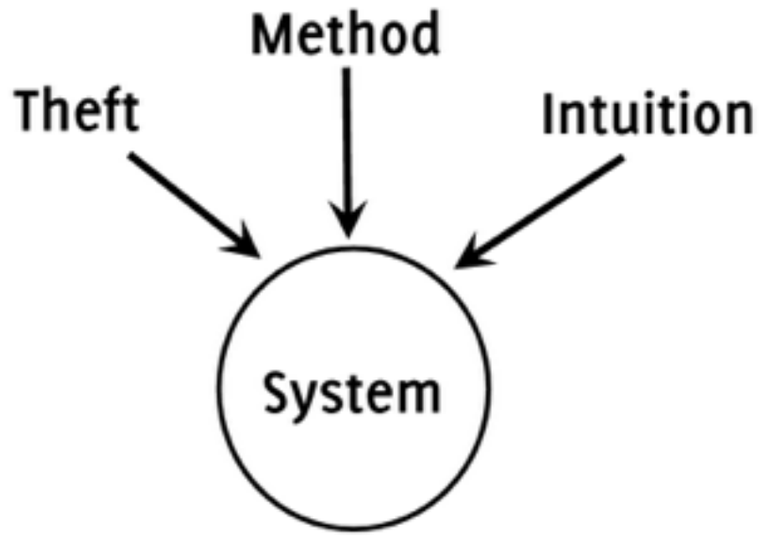
System Artifacts

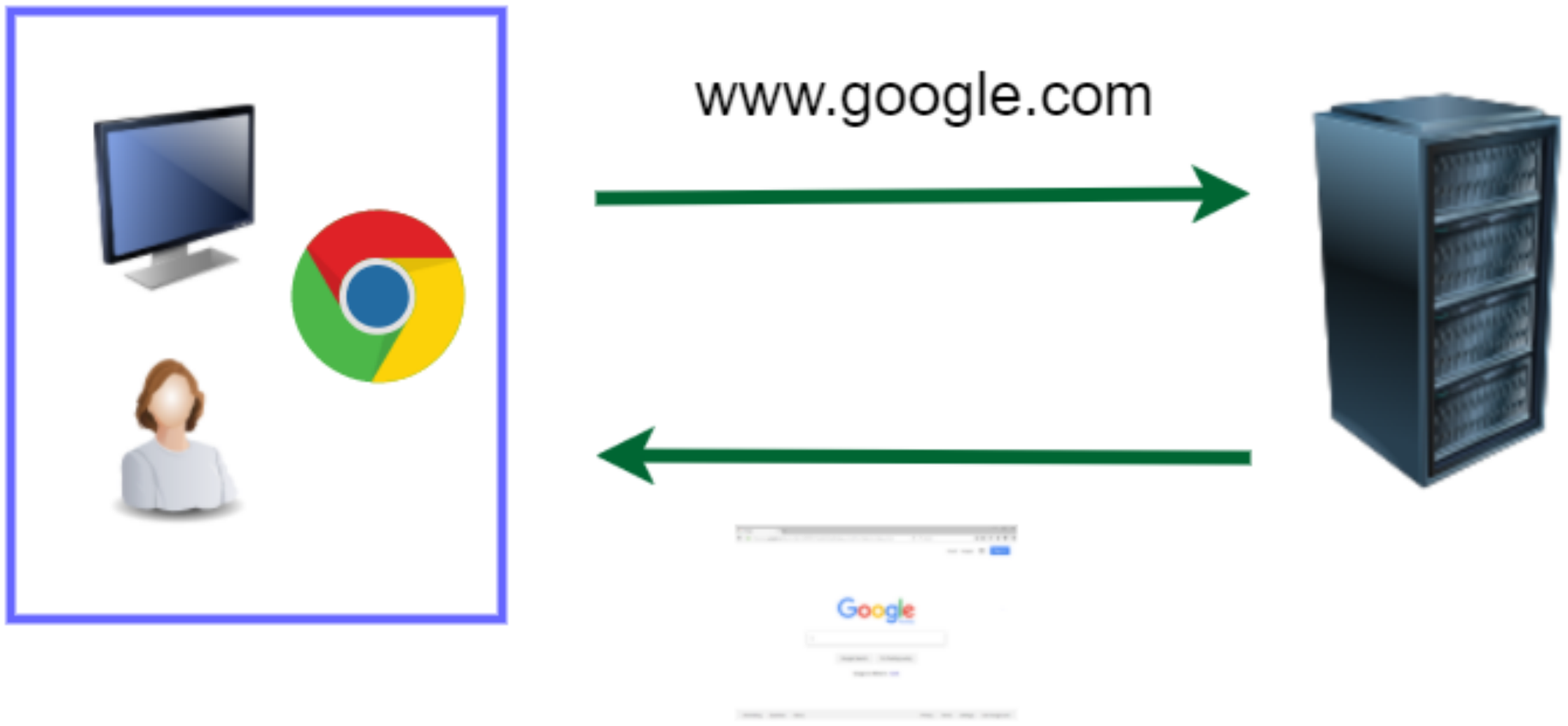
Recovery

What designers want

Design

- Architectural Styles
- Architectural Patterns
- Building Blocks
  - *Software Components*
  - *Component API/Interfaces*
  - *Software Connectors*



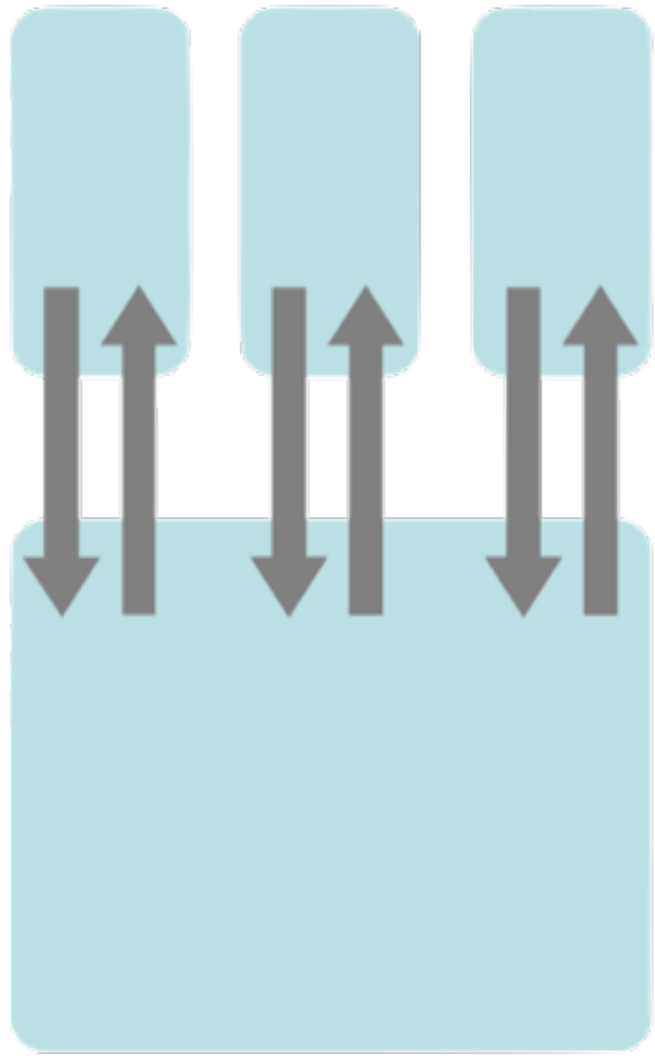


Send HTTP request, and get HTTP response containing the HTML page

Browser visualizes it

Client/Server

- Many clients, active, close to users
- One server, passive, close to data
- Single point of failure, scalability
- Security, scalability



HTTP

The Hypertext Transfer Protocol

HTTP

The Hypertext Transfer Protocol

Connector or Component ?

HTTP

The Hypertext Transfer Protocol

Connector or Component ?

Synch or Asynch ?

HTTP

The Hypertext Transfer Protocol

Connector or Component ?

Synch or Asynch ?

Stateful or stateless ?

HTTP Request

- Action: verb, express the intent
- Headers: meta-data
- Body: **optional**, can be anything, a stream of bytes, form data, session information, etc.

HTTP Actions

Have precise semantic, and
a web application might not implement all of them

GET	*retrieve a resource*
POST	*send data and/or create a resource*
PUT	*replace an existing resource*
DELETE	*delete a resource*
HEAD	*retrieve HEADers but not body*
OPTIONS	*check the methods available on the resource*

Web applications must to implement the semantic right

HTTP Response

Headers and status

Status codes, organized in families:

1xx: information
2xx: success
3xx: redirection
4xx: user error
5xx: server error

Delivers a resource: a page HTML, a CSS file for the style, images, JS libraries, etc.

HTTP Body

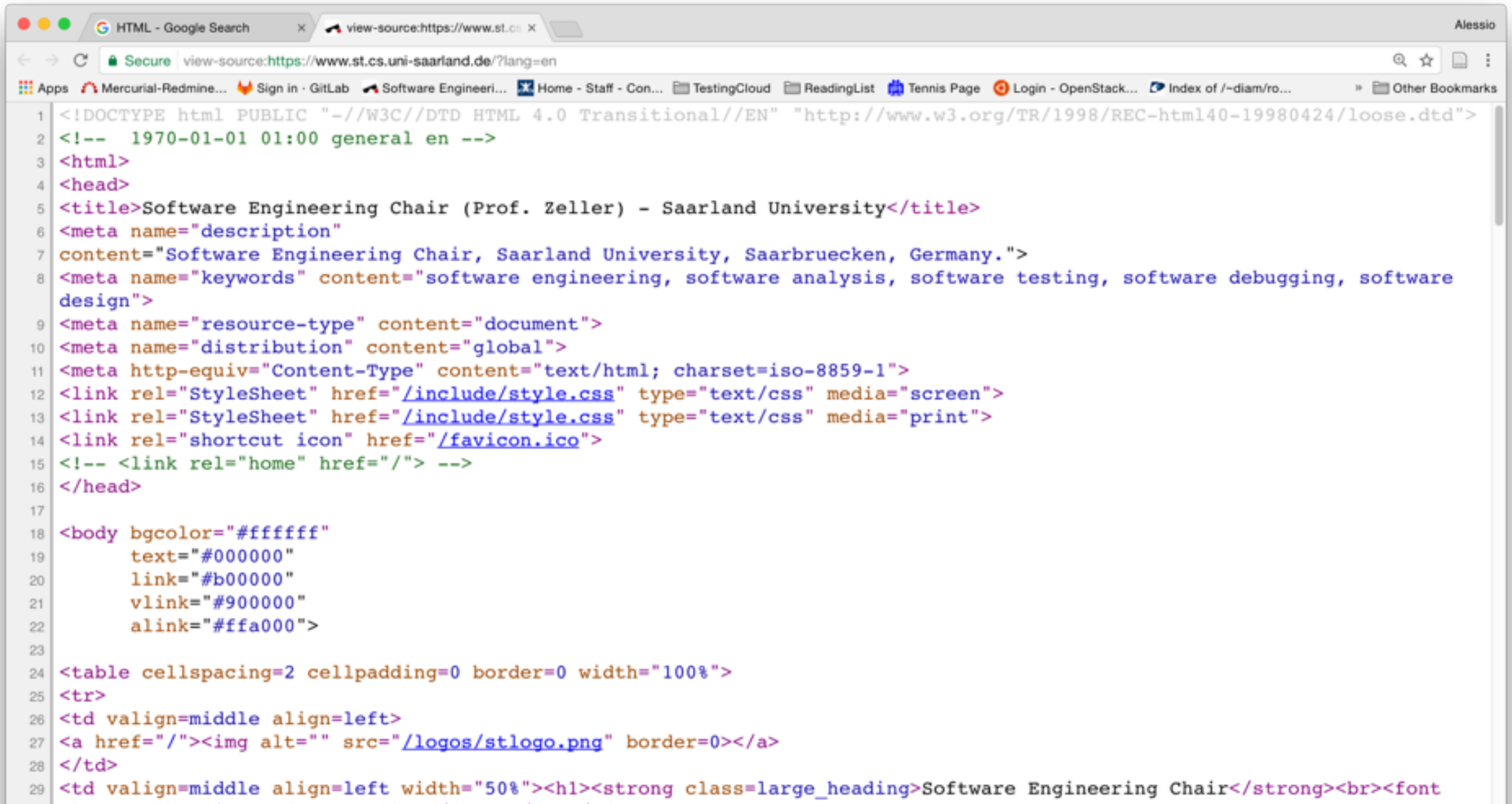
- Transfer the main part of the data, but not the only way to send data

  query params, custom headers

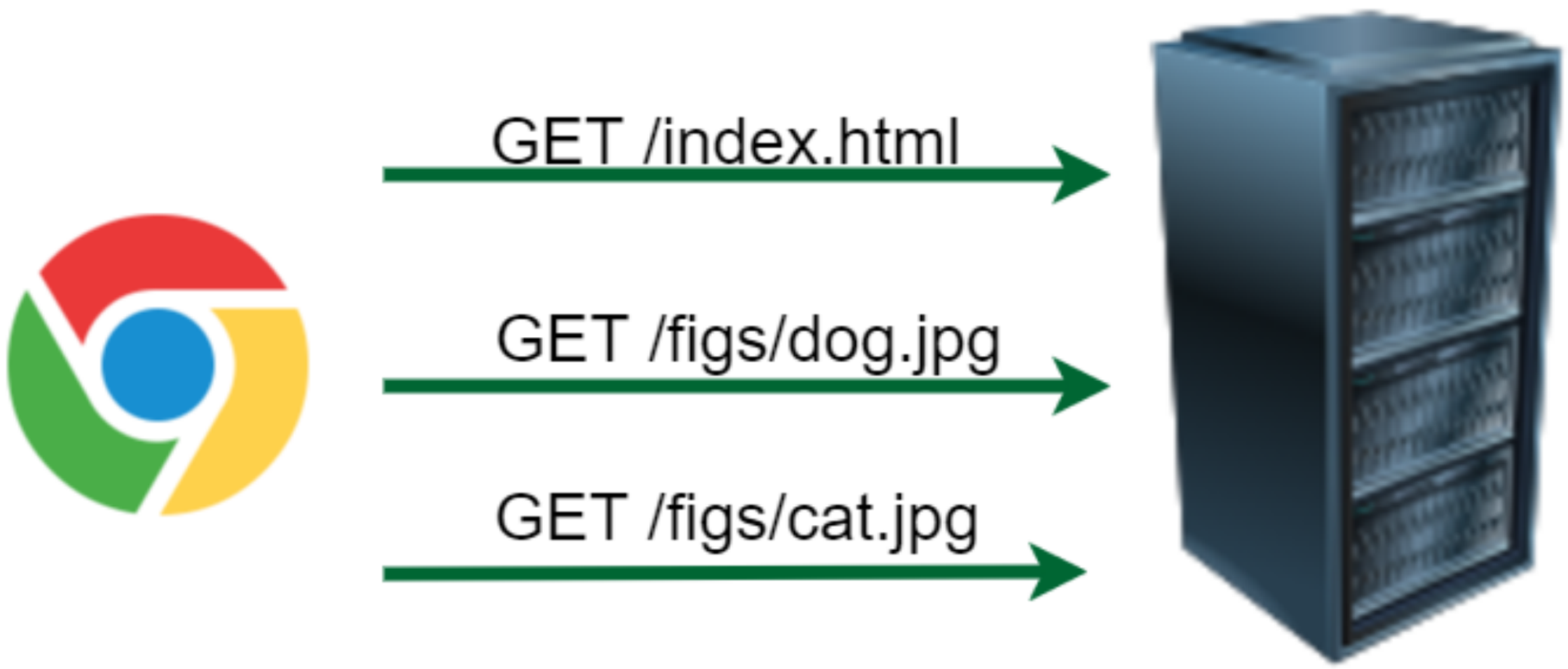
- Required in POST and PUT requests
- Required in responses to GET requests
- HEAD must not provide one

HTML

The Hypertext Markup Language



Resources



One request per resources, multiple requests in parallel

All requests must complete before a page is fully displayed

Static vs Dynamic Pages

Static:

Files are served as they are (index.html)
content does not change

Static vs Dynamic Pages

Static:

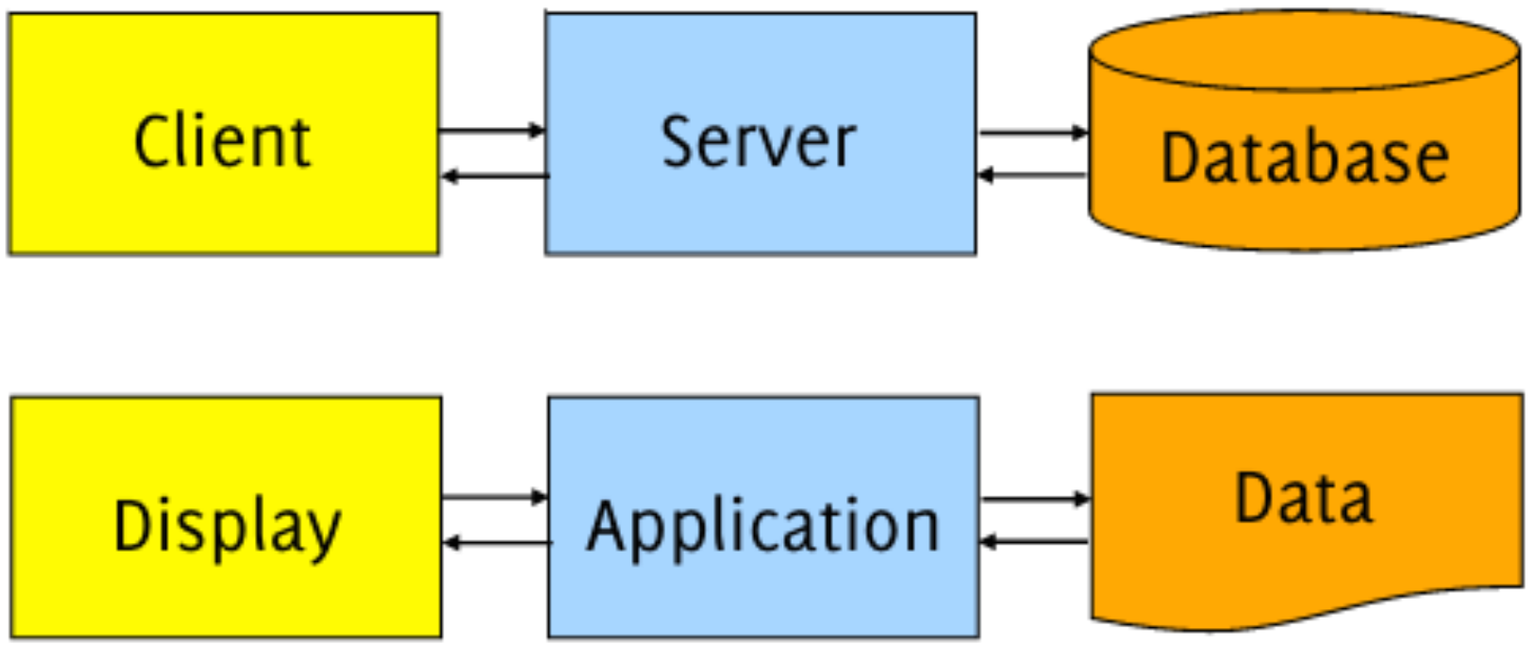
Files are served as they are (index.html)
content does not change

Dynamic:

The HTML (or part of it) is generated upon request based on data

State-Logic-Display

cluster elements that change at the same rate



Server-Side HTML Rendering

- The HTML page is created on the server and sent back to the client

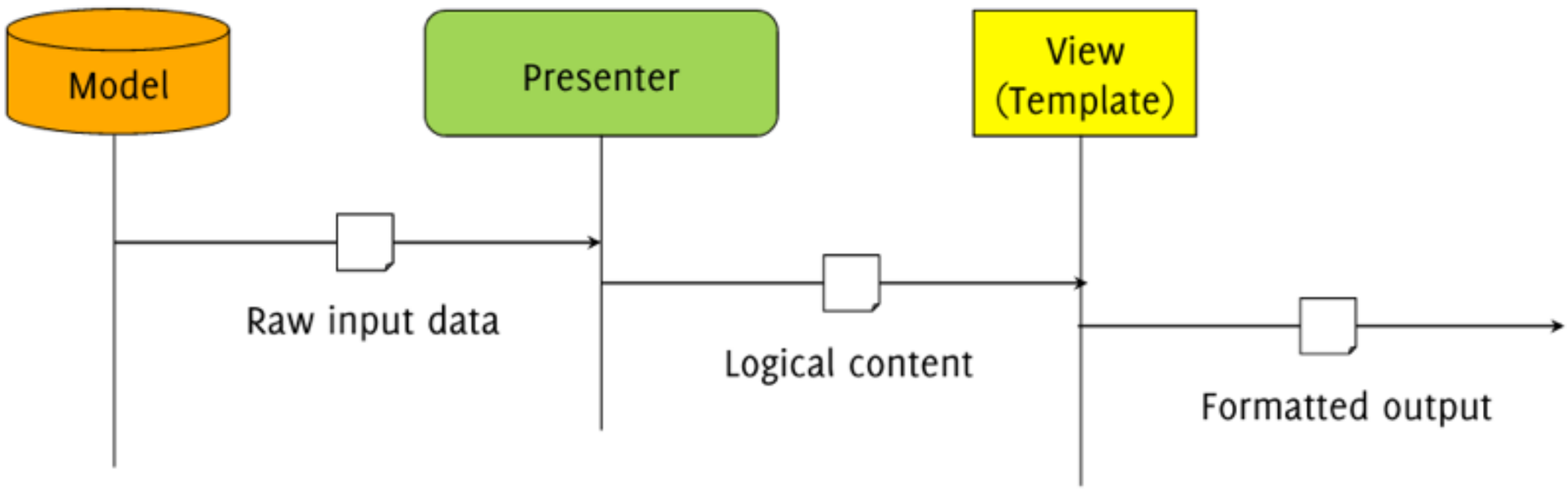
- Overhead in processing each request if the page is created from scratch

- Same content for different displays

    Desktop vs Tablet vs Mobile

Presenter-View

extract the content from the model to be presented from the rendering into screens/web pages



Server-Side HTML Rendering

- Based on **HTML templates** that mix together HTML tags, data, and code

HTML = Template + Data

Cleaning Supplies
* mop
* broom
* duster

```
<h1><%= title %></h1>
<ul>
<% for(var i=0; i<supplie
   <li><%= supplies[i] %>
<% } %>
</ul>
```

```
{ title:
    'Cleaning Supplies',
  supplies:
    ['mop',
     'broom',
     'duster'] }
```

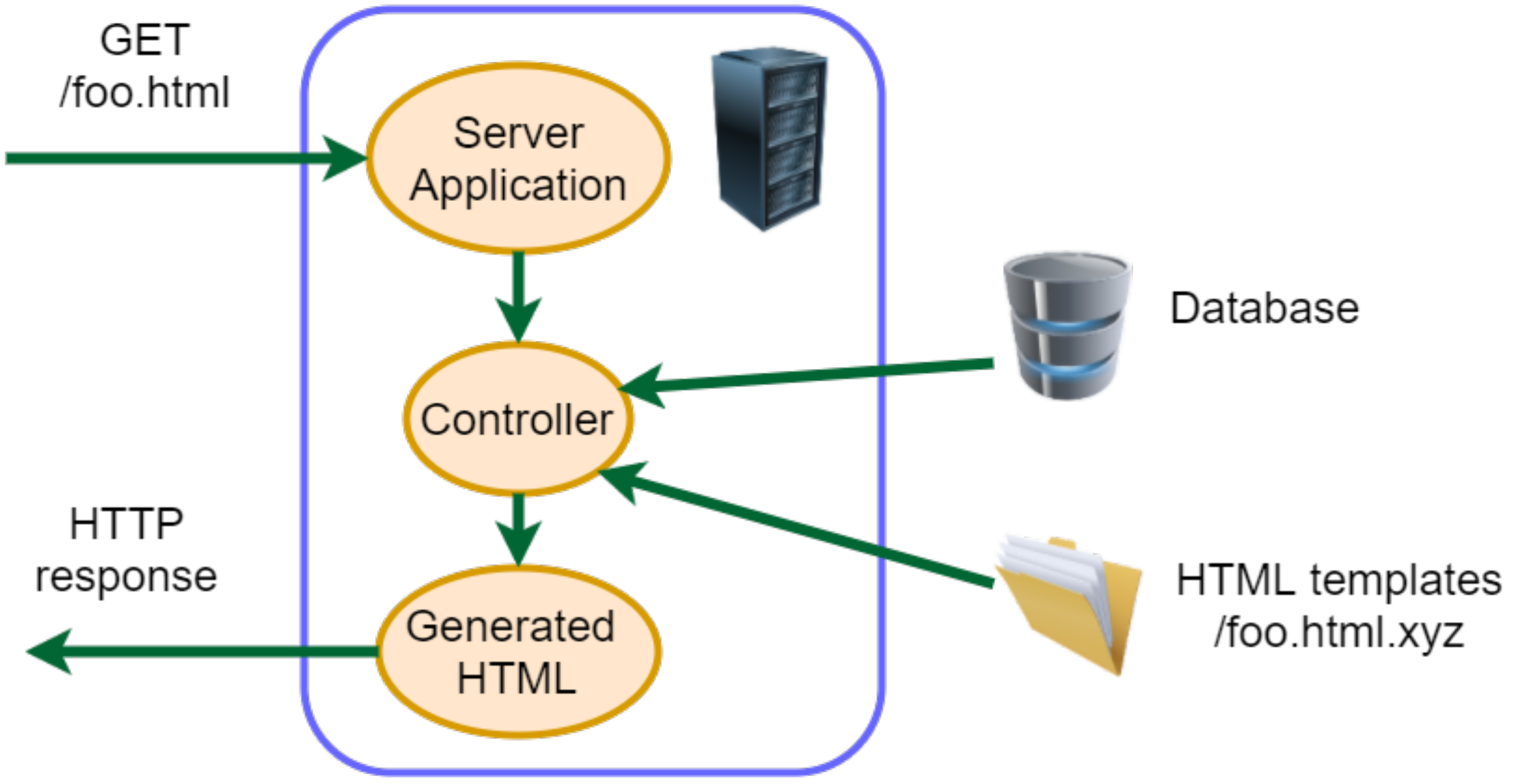
```
html= new EJS({url:'template.ejs'}).render(data)
```

http://www.embeddedjs.com/getting_started.html

Server-Side HTML Rendering

- Based on **HTML templates** that mix together HTML tags, data, and code
- Different technologies:

  PHP scripts — index.php
  JavaServer Faces (JSF) — index.xhtml
  Embedded Ruby (ERB) — index.html.erb

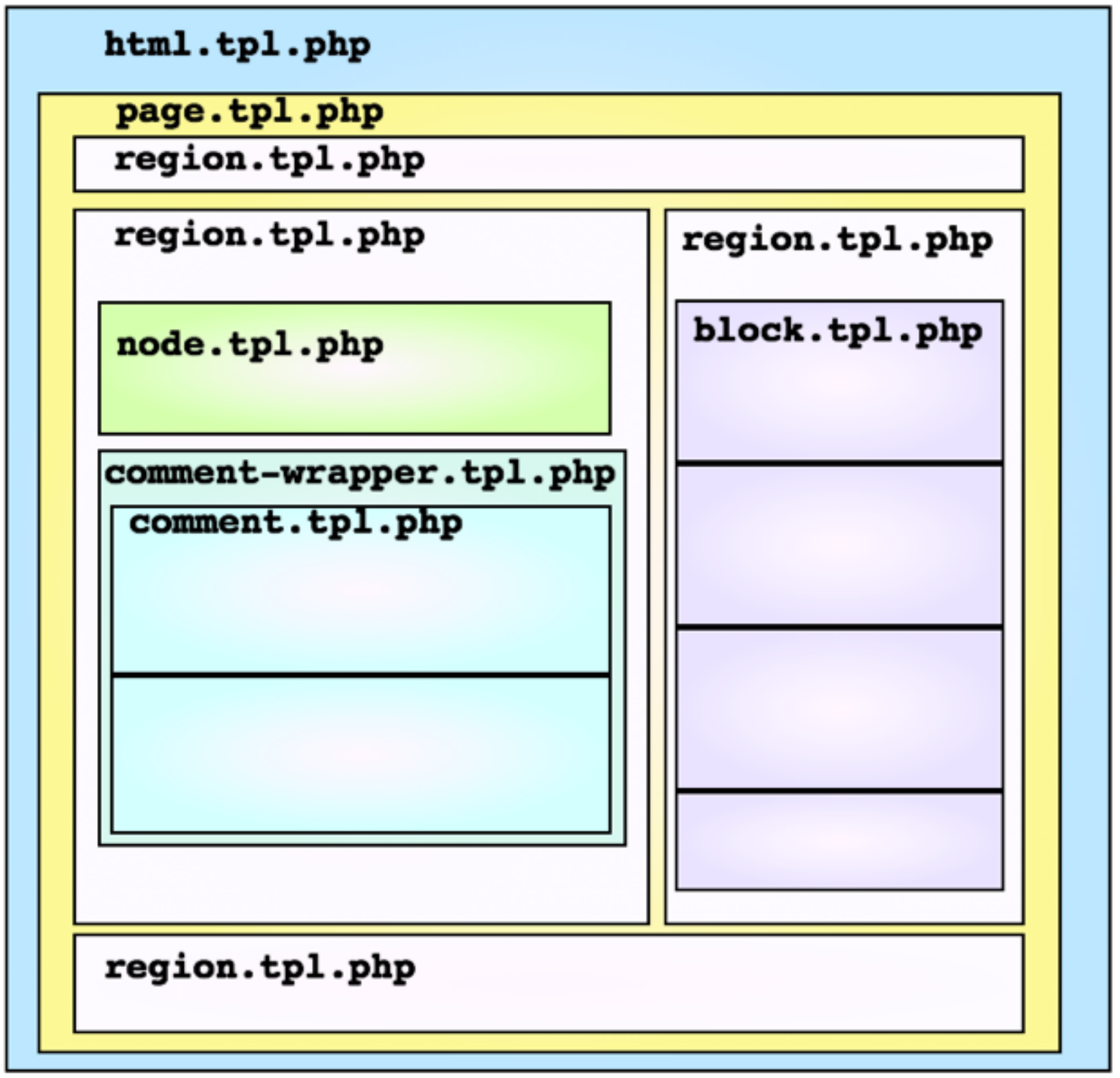

GET
/foo.html
Server
Application
Controller
Database
HTTP
response
Generated
HTML
HTML templates
/foo.html.xyz

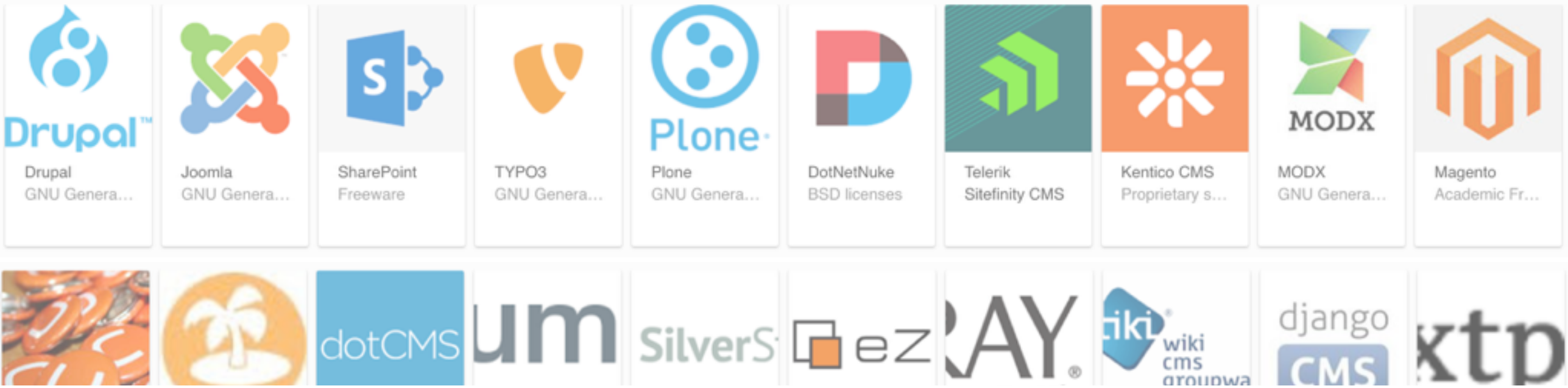
Server-Side HTML Rendering

- Based on **HTML templates** that mix together HTML tags, data, and code
- Different technologies:
  PHP scripts — index.php
  JavaServer Faces (JSF) — index.xhtml
  Embedded Ruby (ERB) — index.html.erb

- Templates do not necessarily target the entire page and they might not be stored in “files”

Components

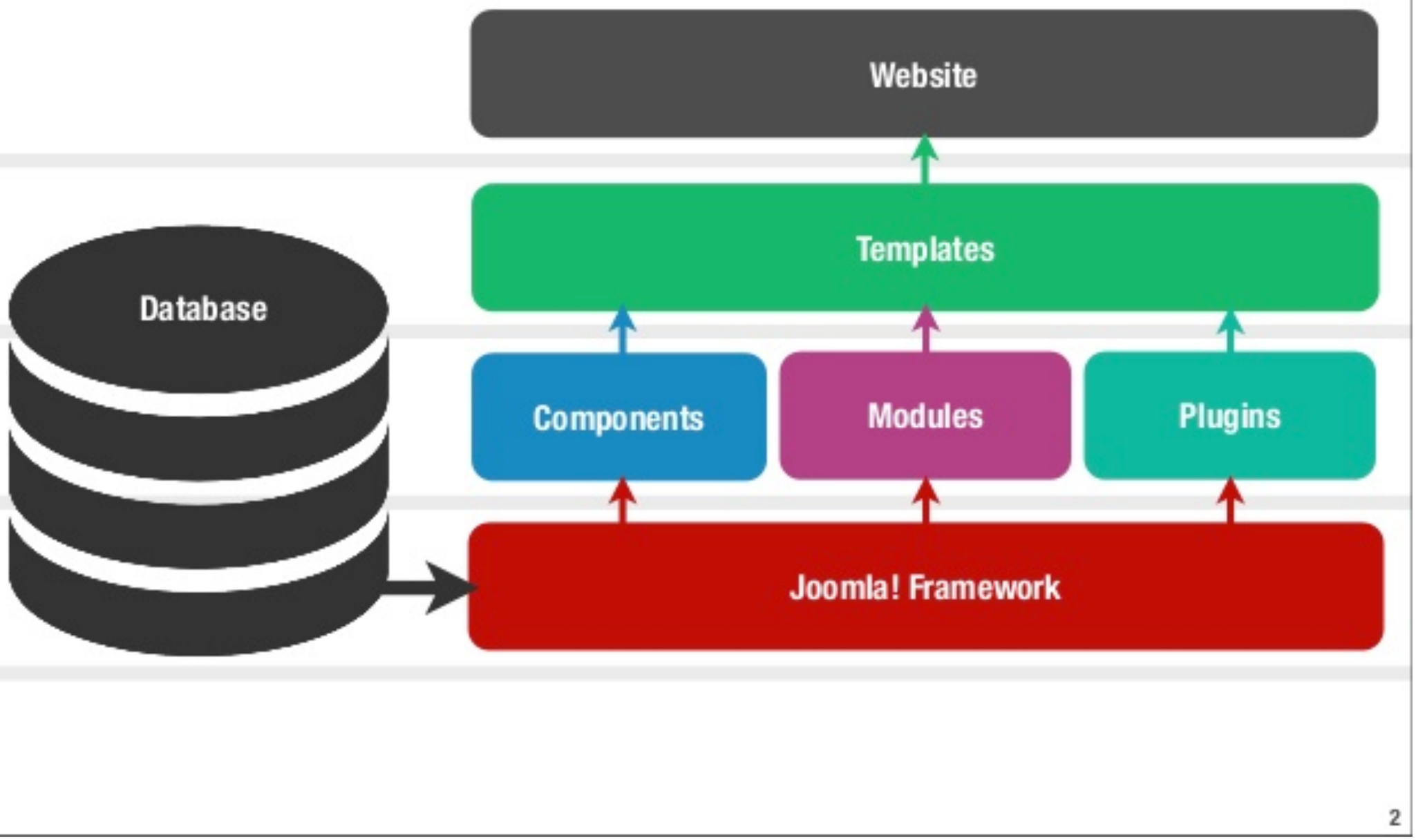

html.tpl.php
page.tpl.php
region.tpl.php
region.tpl.php
node.tpl.php
comment-wrapper.tpl.php
comment.tpl.php
region.tpl.php
block.tpl.php
region.tpl.php



Content Management Systems (CMS)

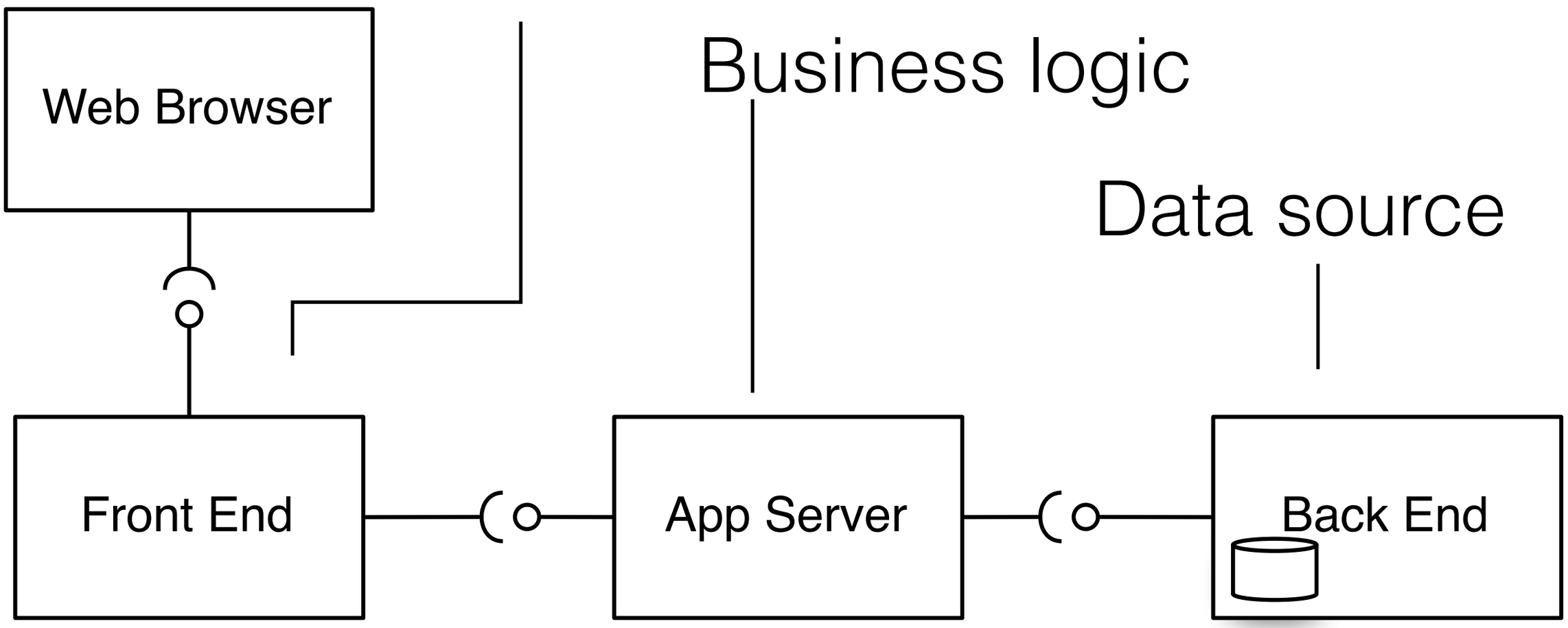
Joomla! CMS Architecture

Lets take a look under the hood ....



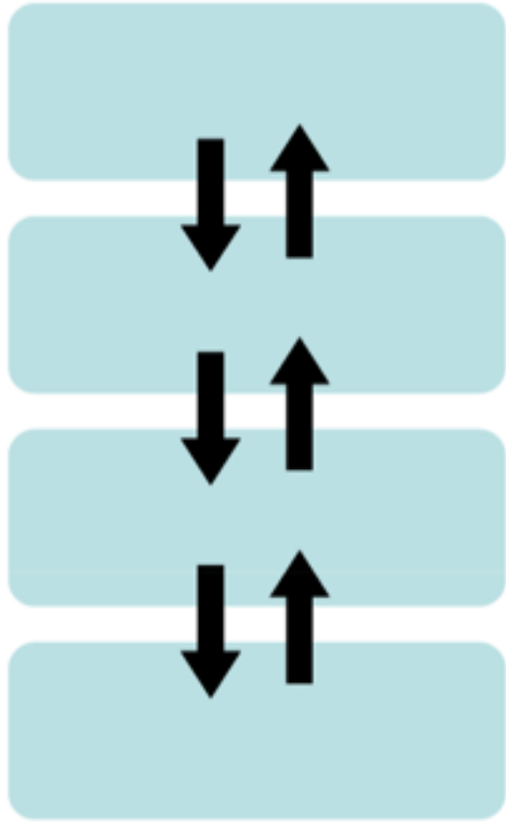
Design

3-Tier Architecture



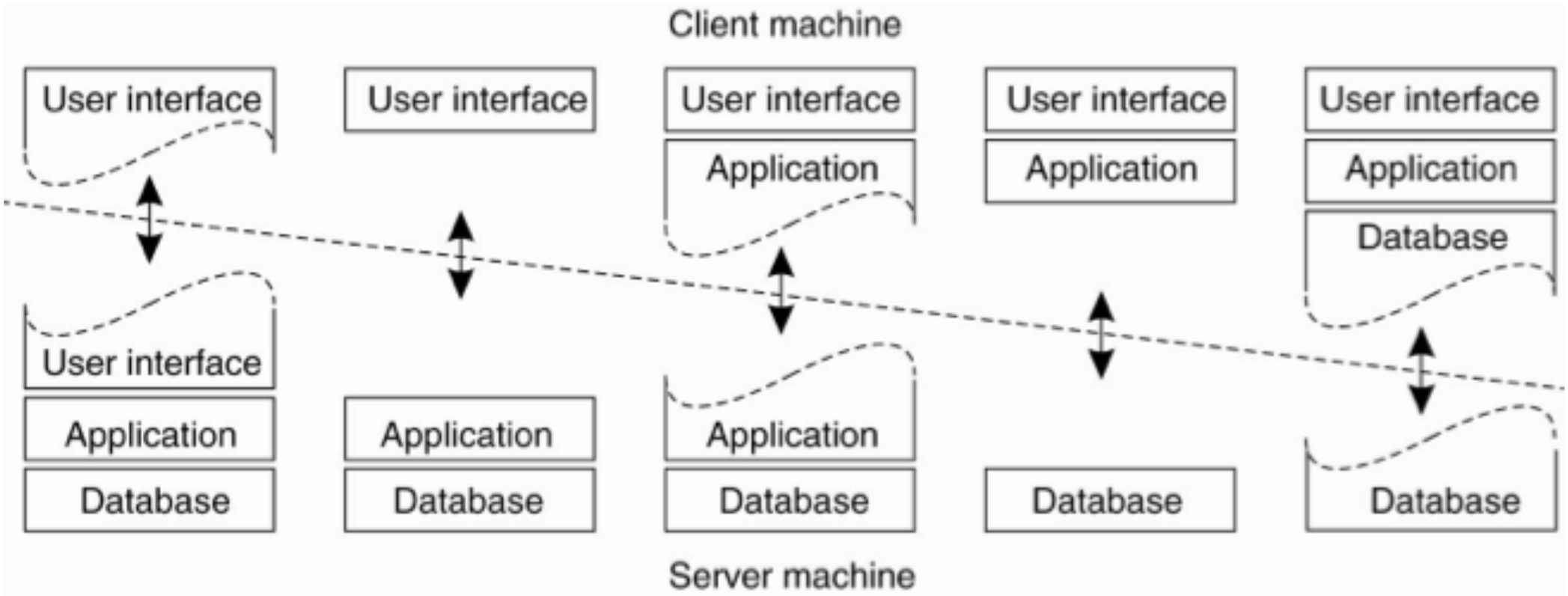
Layered (Style)

- Communications 1 layer up/down
- Information hiding, no circular deps
- Possibly bad performance
- Good evolvability

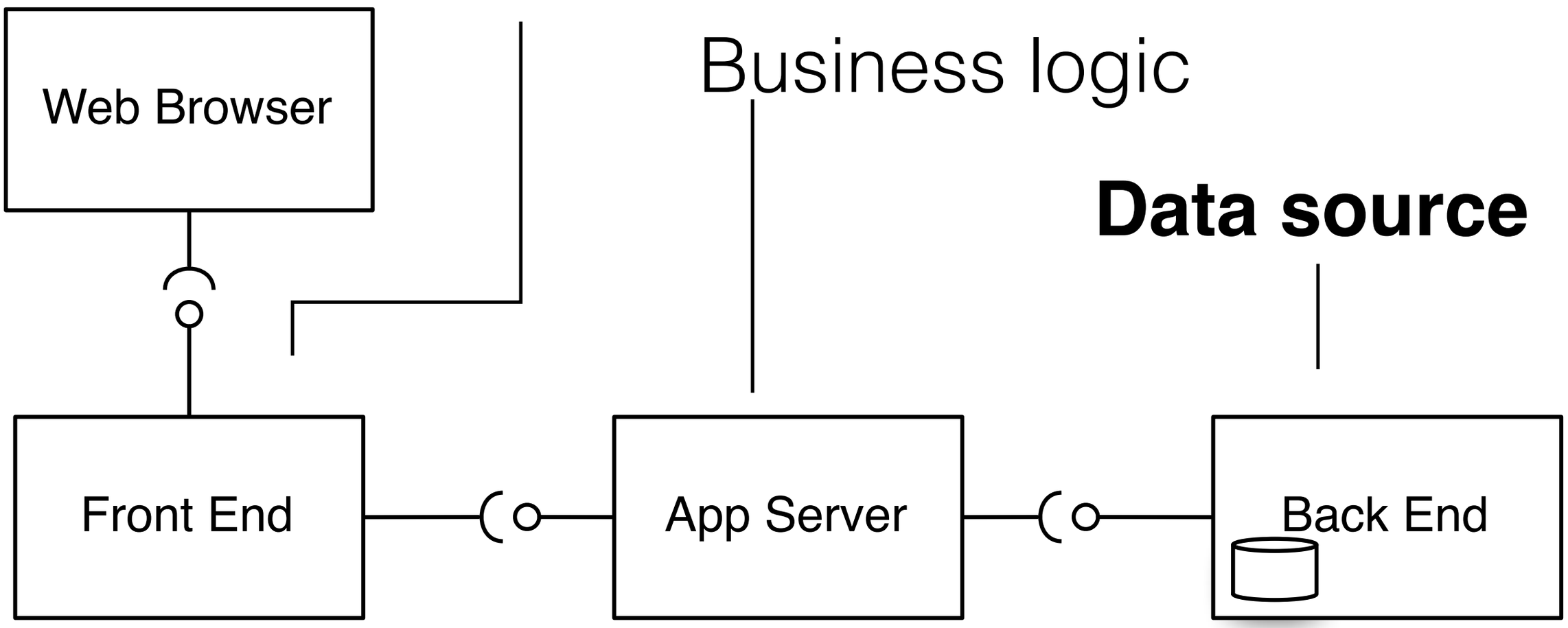


What run where?

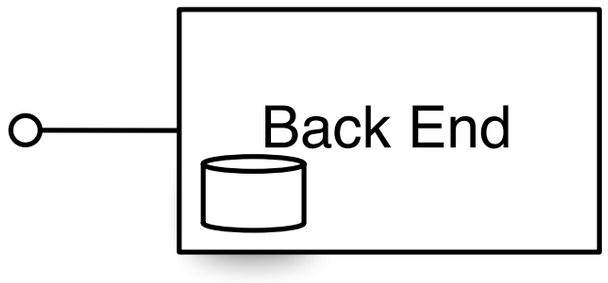
thin-client *fat-client*



3-Tier Architecture

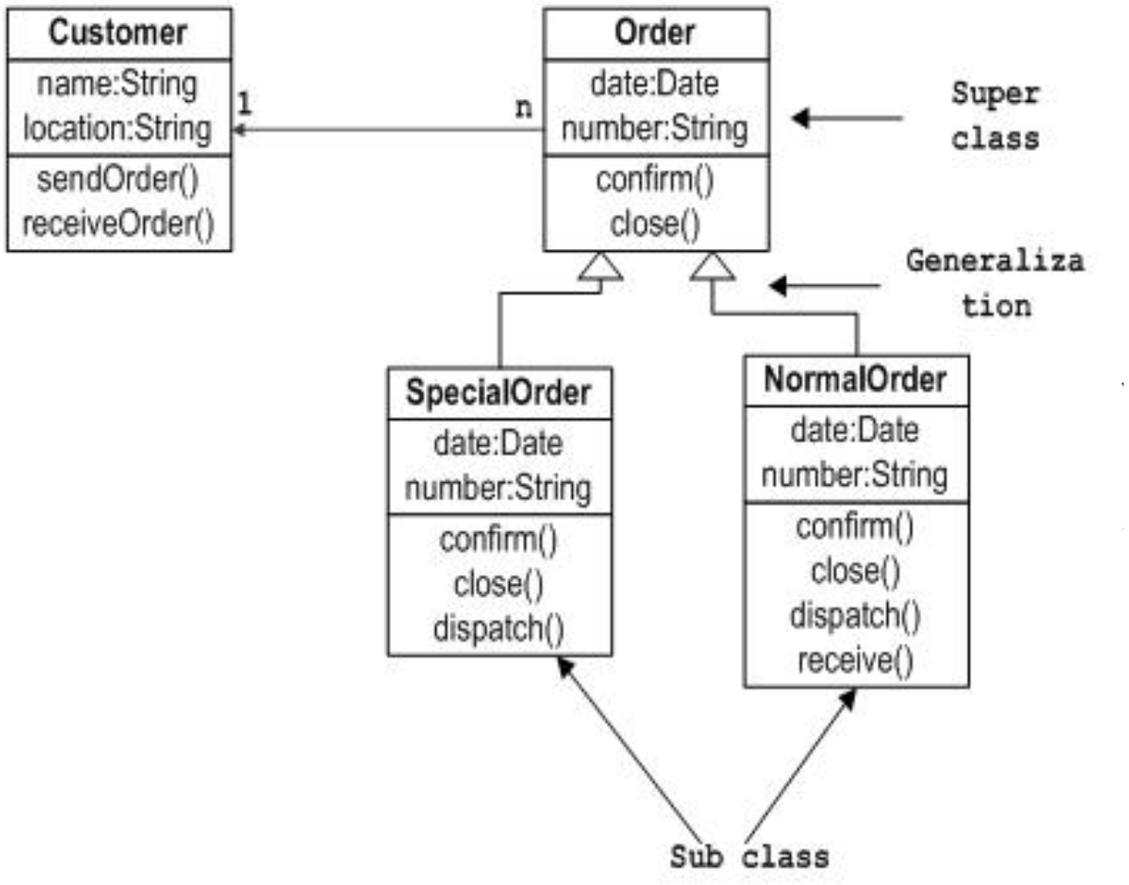


Data Layer

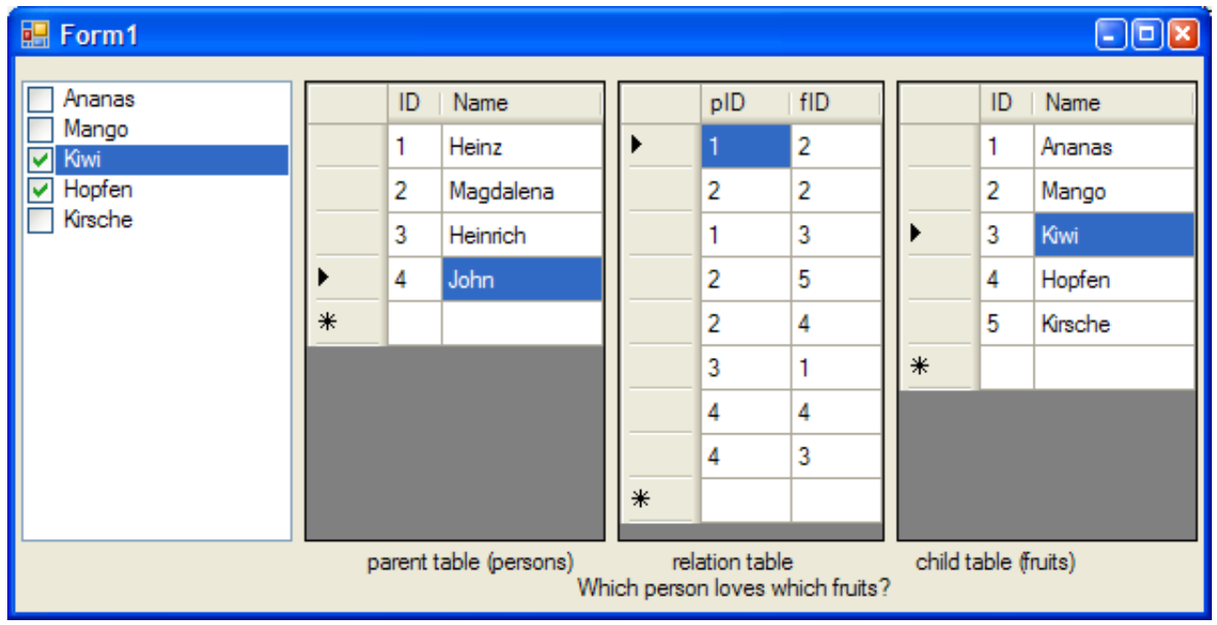


- Persistence
- Storage

A Mapping Problem



Domain model



Data storage

Domain Model

- Represent concepts in the domain and their relations, not as rows in a database

- Network of interconnected concepts

- Abstract Data Type
  data and the behavior

Storage model

How to store data?

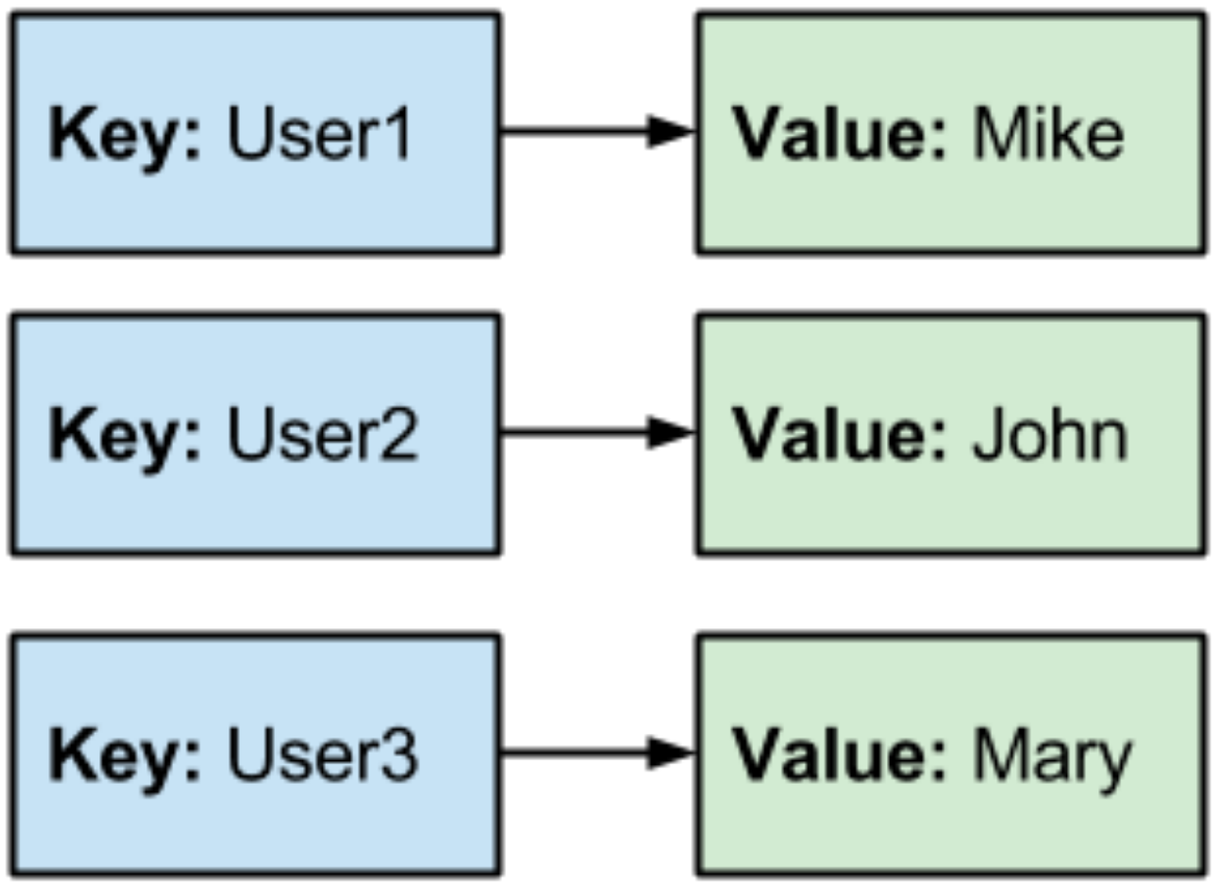
- Key-Value Model
  list of keys and values (hashtable style)

- Relational Model
  traditional SQL model

- Document-oriented Model
  schema-less documents

- Graph-oriented Model
  data is stored as an interconnected graph

Key-Value



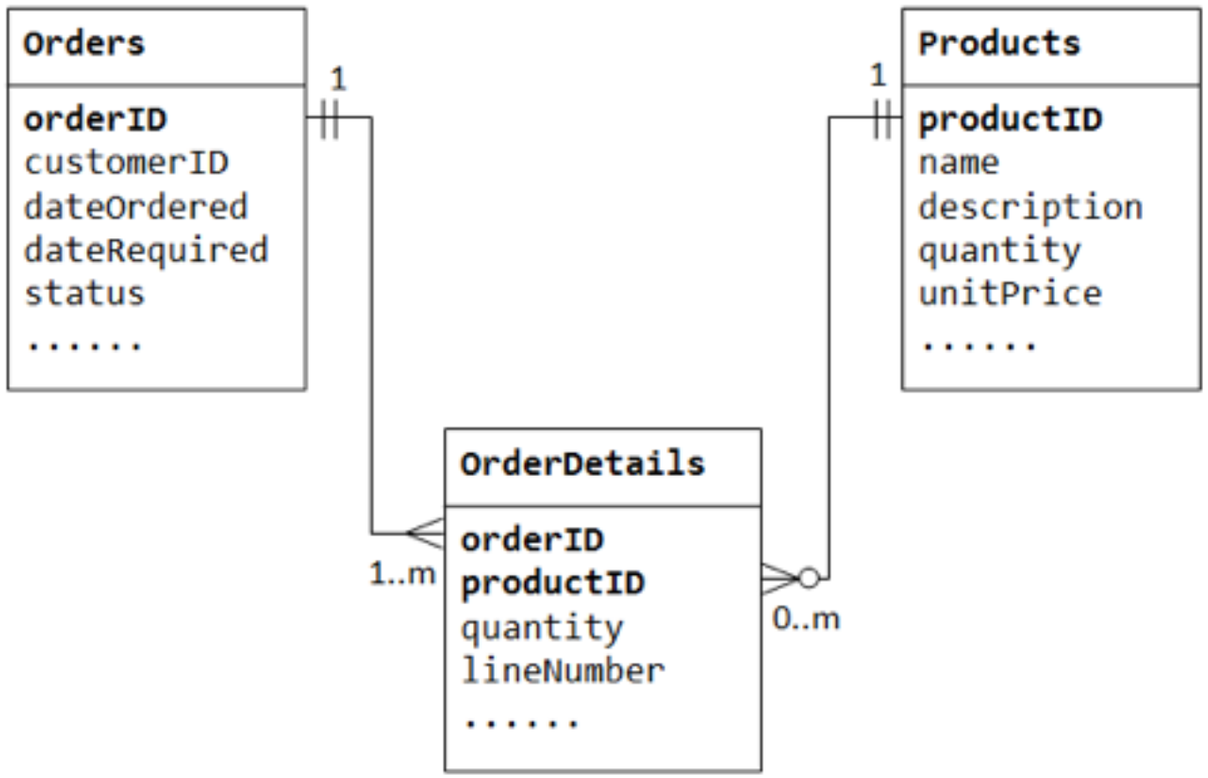
- Implement a map
- Values have no schema

riak

DynamoDB

amazon web services

Relational

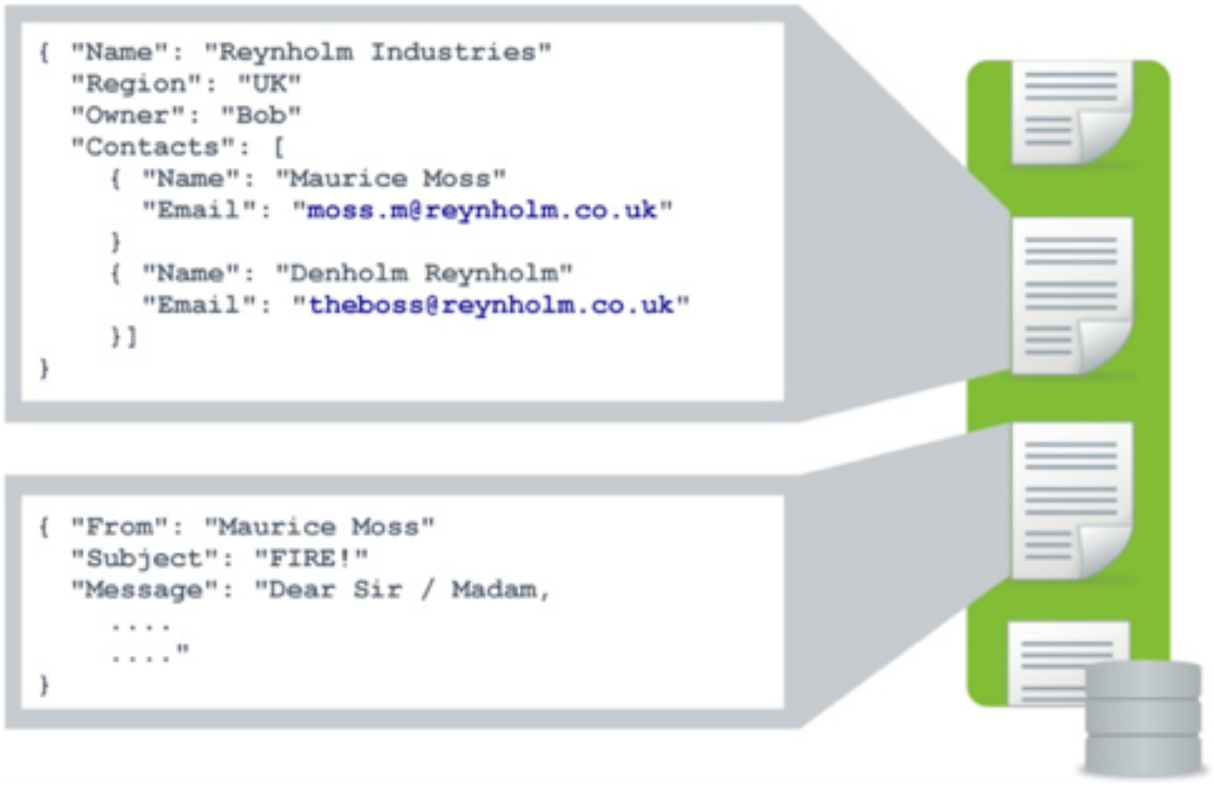


- Set-theory
- Collection of tables with rows and columns

MySQL

PostgreSQL

Document-oriented

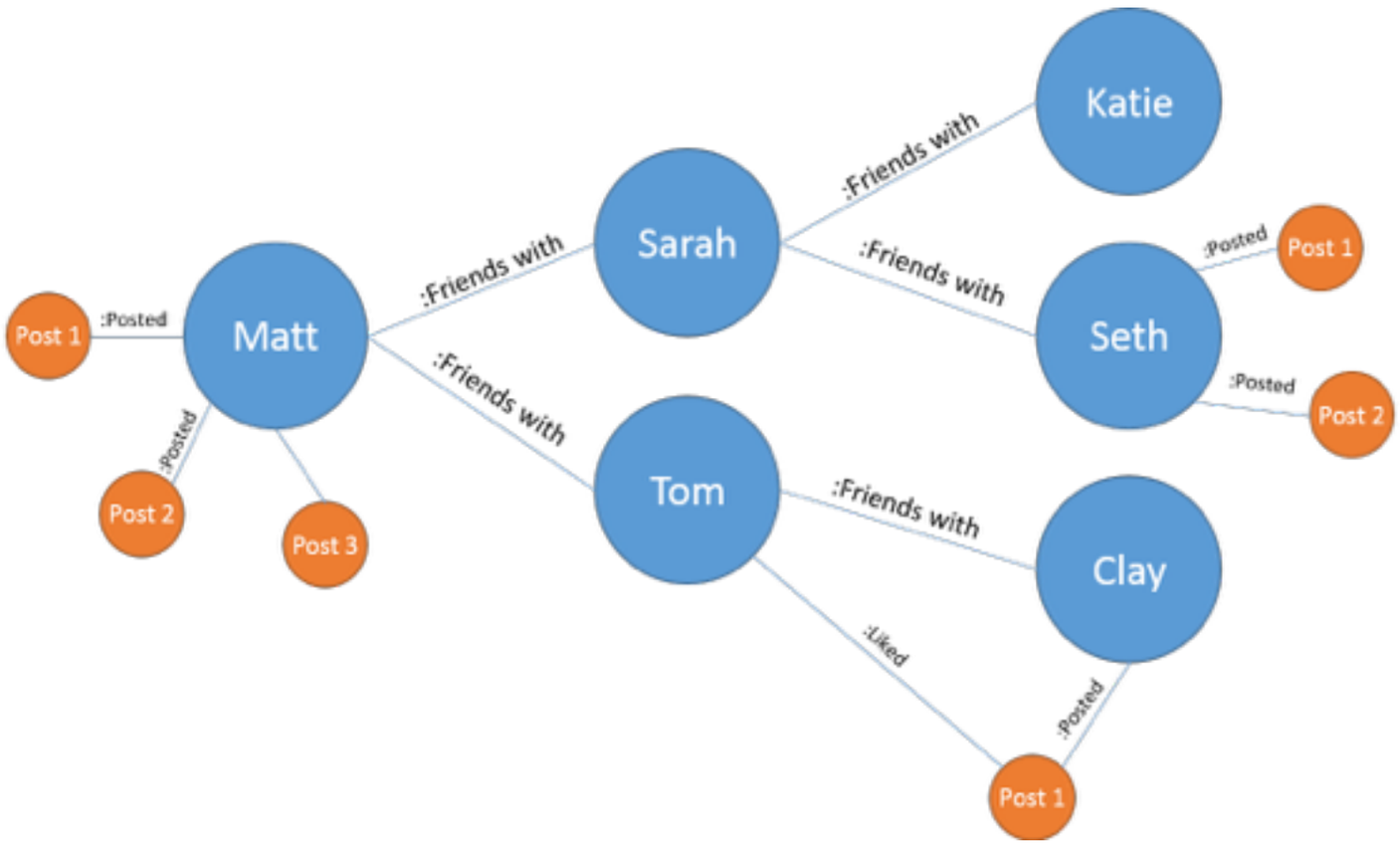


- Data stored in document but not relations
- Extends Key-Value

CouchDB

mongoDB.

Graph-oriented

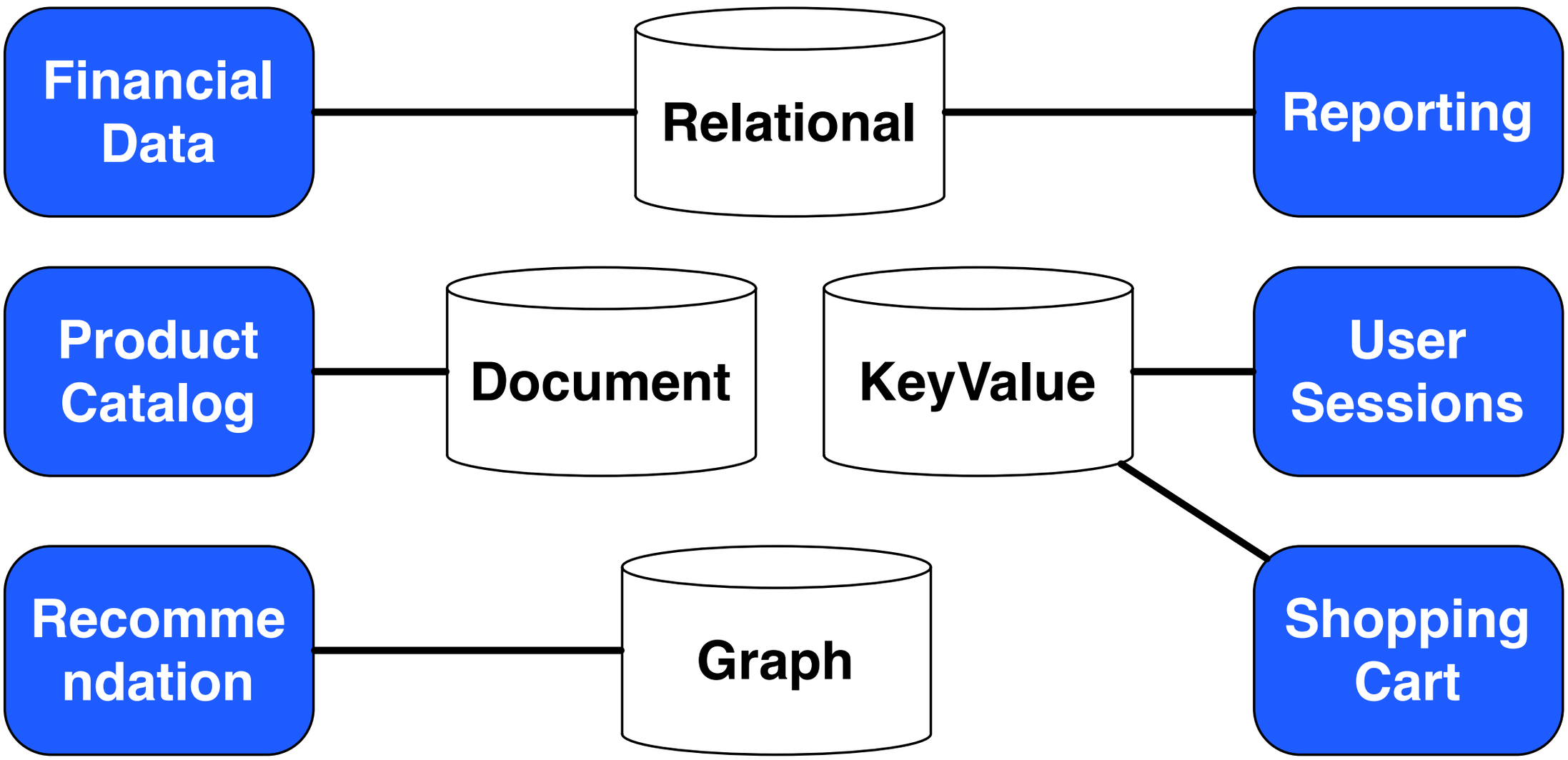


- Data stored as network graph
- Relations first-class citizens

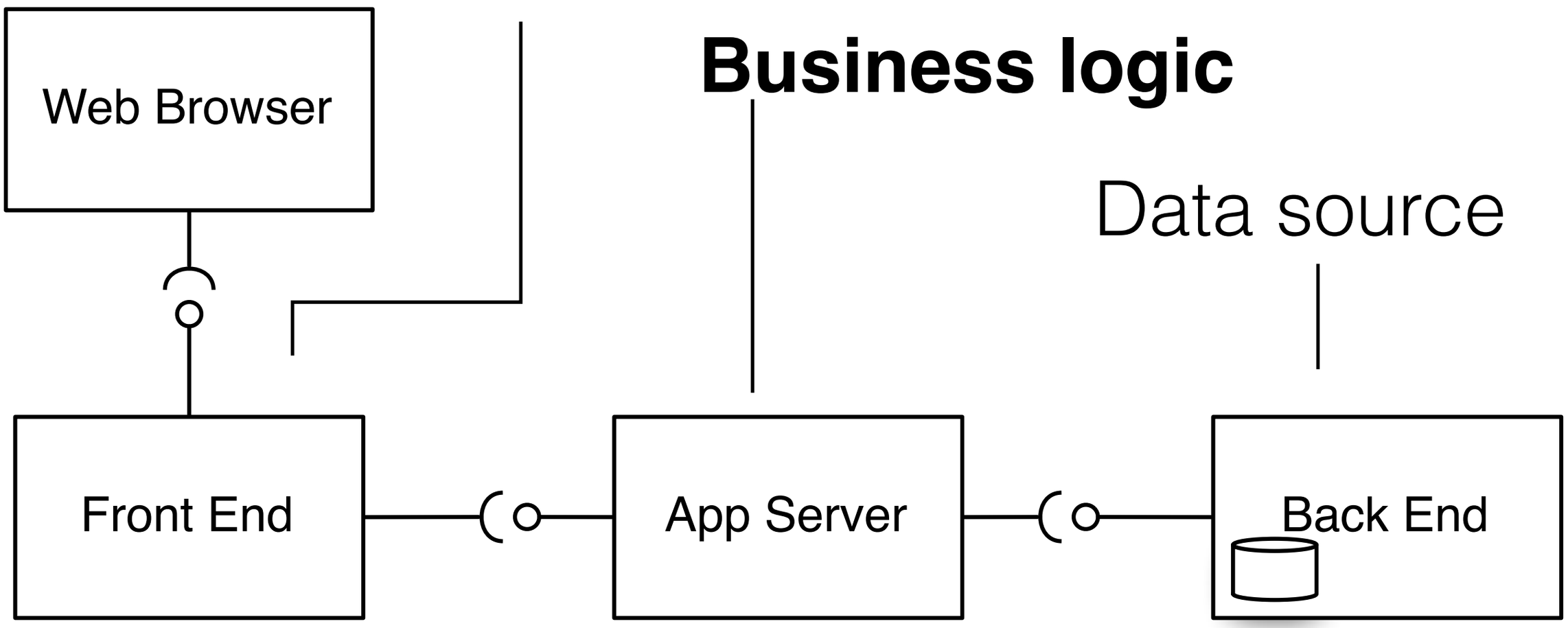
neo4j

Polyglot Persistence

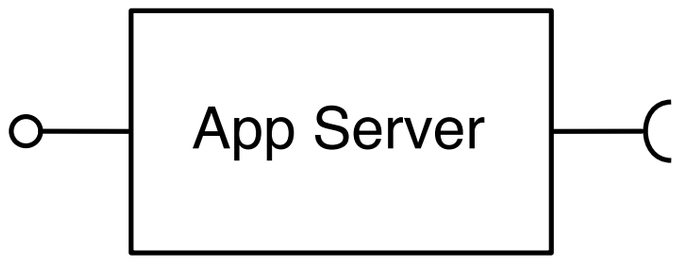
Multiple ways to store data



3-Tier Architecture



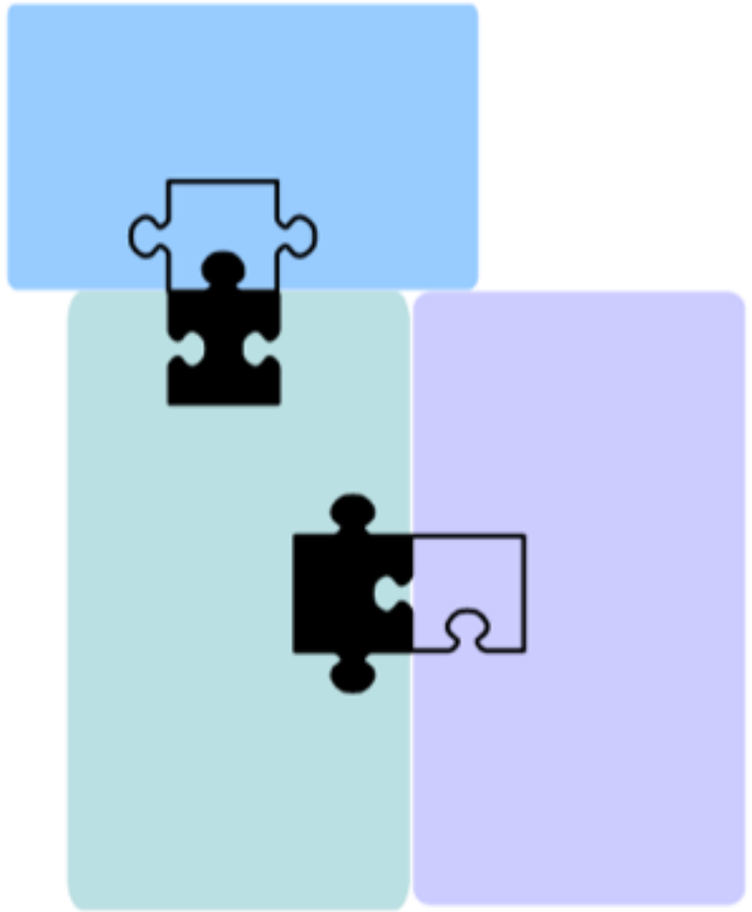
Application Layer



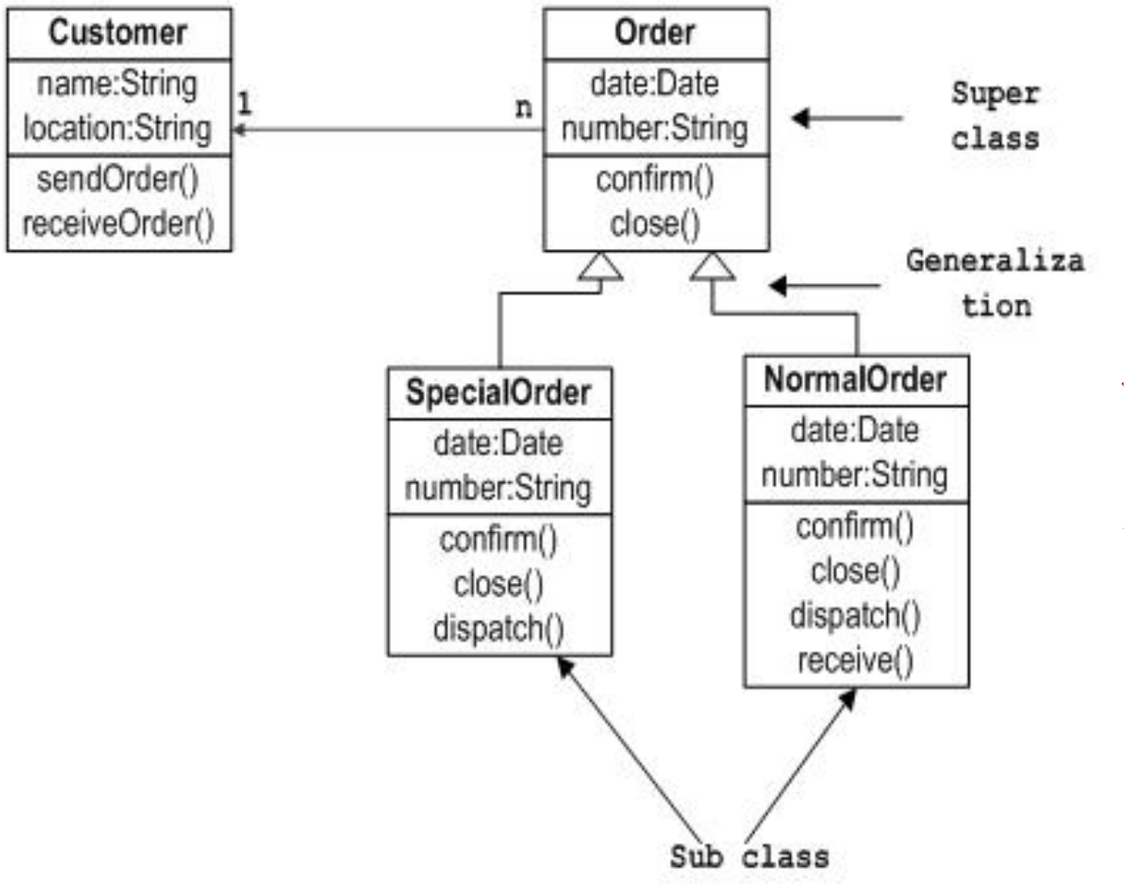
- Data access, navigation, and persistence
- Data processing (Business logic)

Plugin

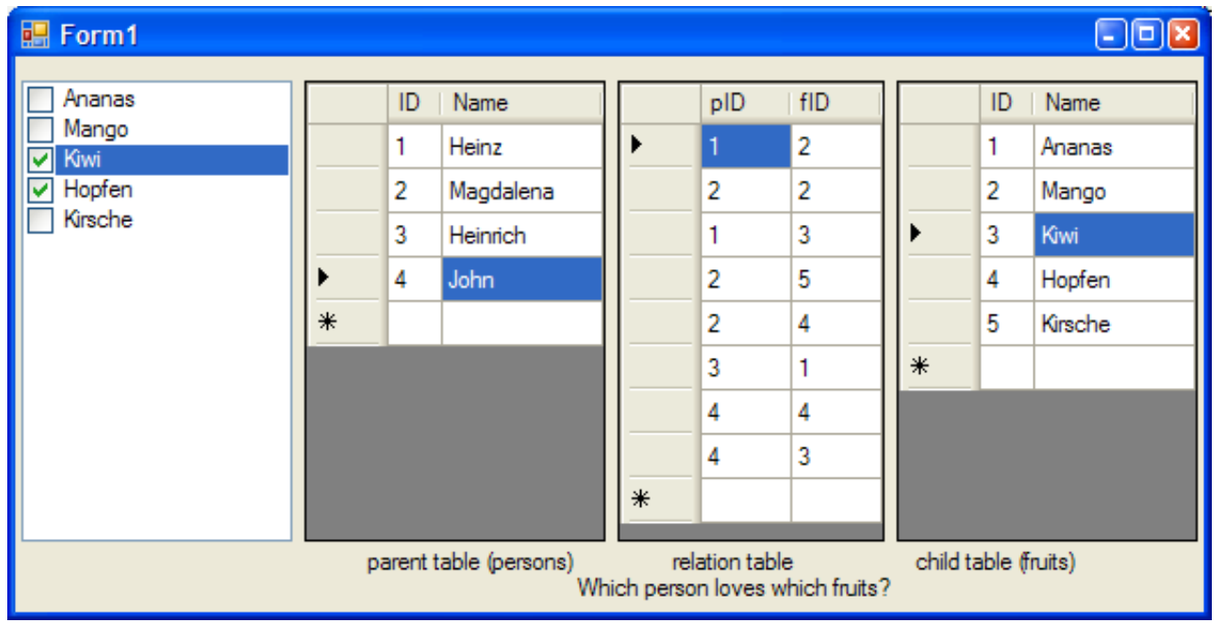
- Explicit extension points
- Static/Dynamic composition
- Low security (3rd party code)
- Extensibility and customizability



A Different Problem (?)

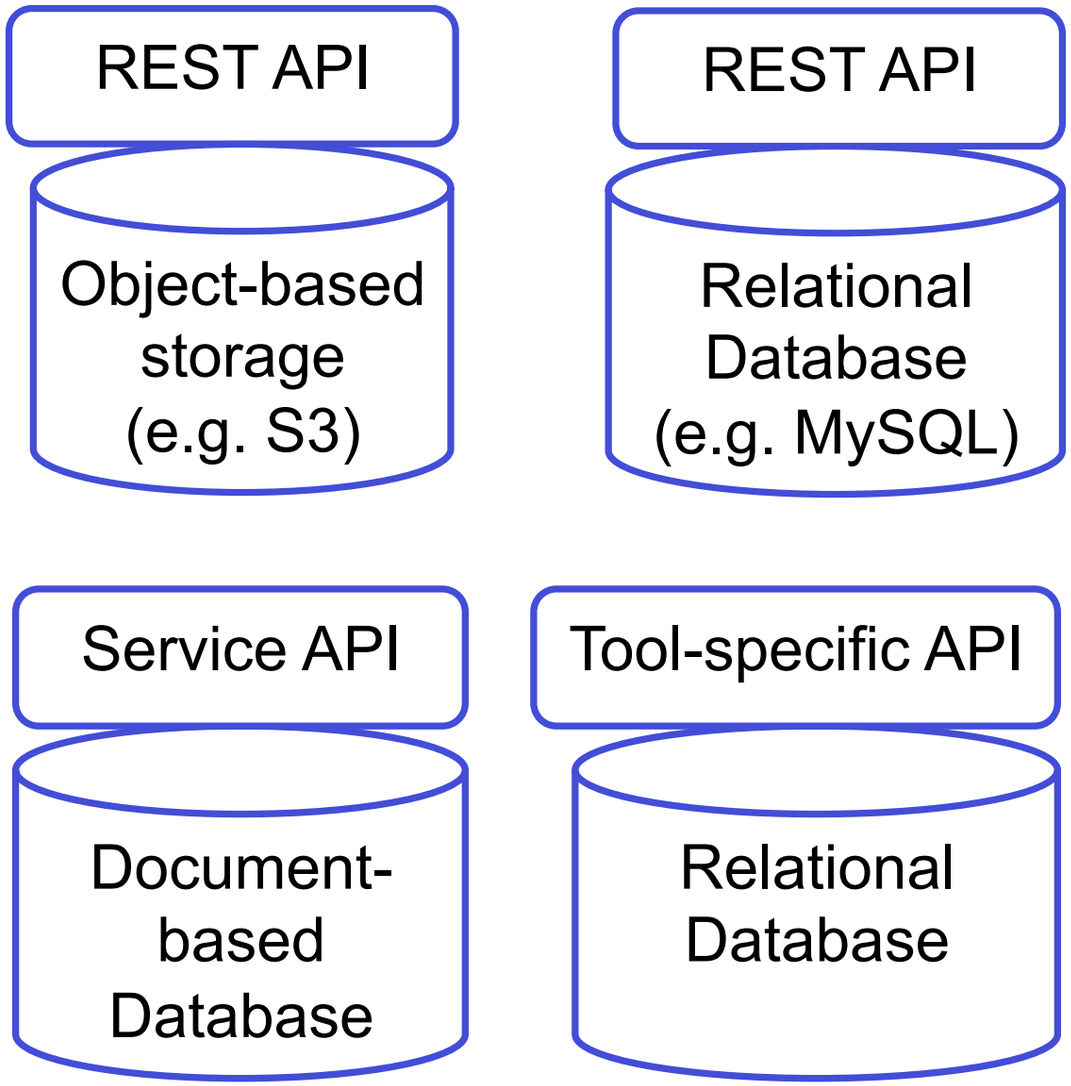


Programming Language



Storage Architecture

Data Access API



- Add an abstraction layer that the represent the database in the application
- Wrap the communication with the data store and expose it as domain model

Data Source Patterns

- Row Data Gateway

  One instance per row returned by a query

- Table Data Gateway

  One instance per table

- Active Record

  Encapsulates DB access and adds business logic to data

- Data Mapper

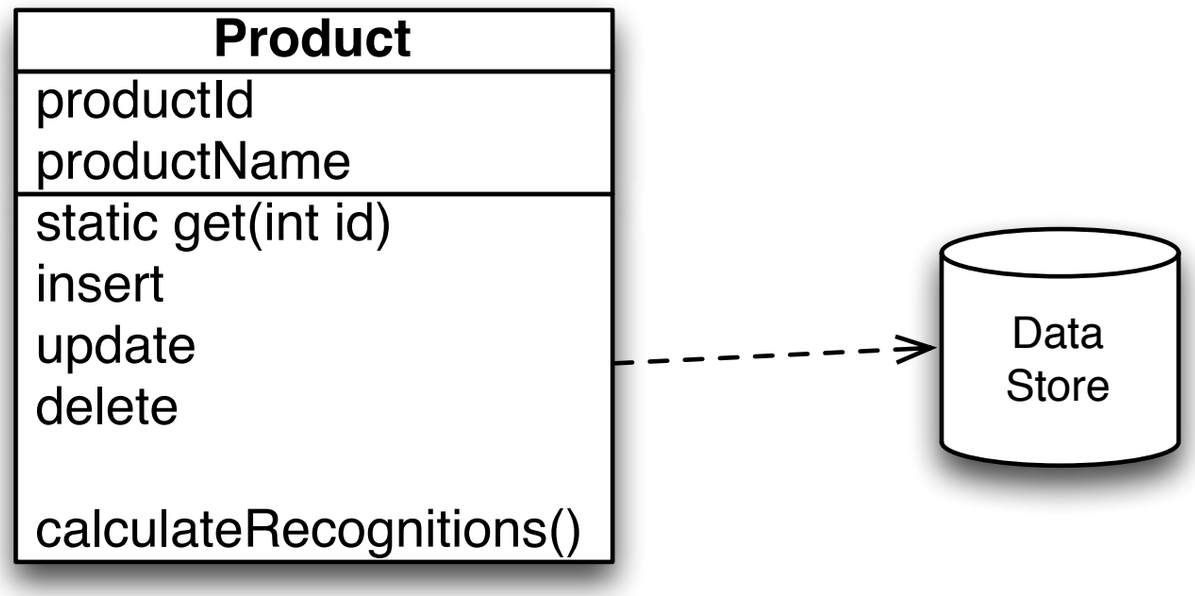
  loads DB into Domain Model, and vice-versa

Row and Table Gateways

- Based on table structure
- Conversion of object type to database format
- Typically stateless
- Push back and forth data

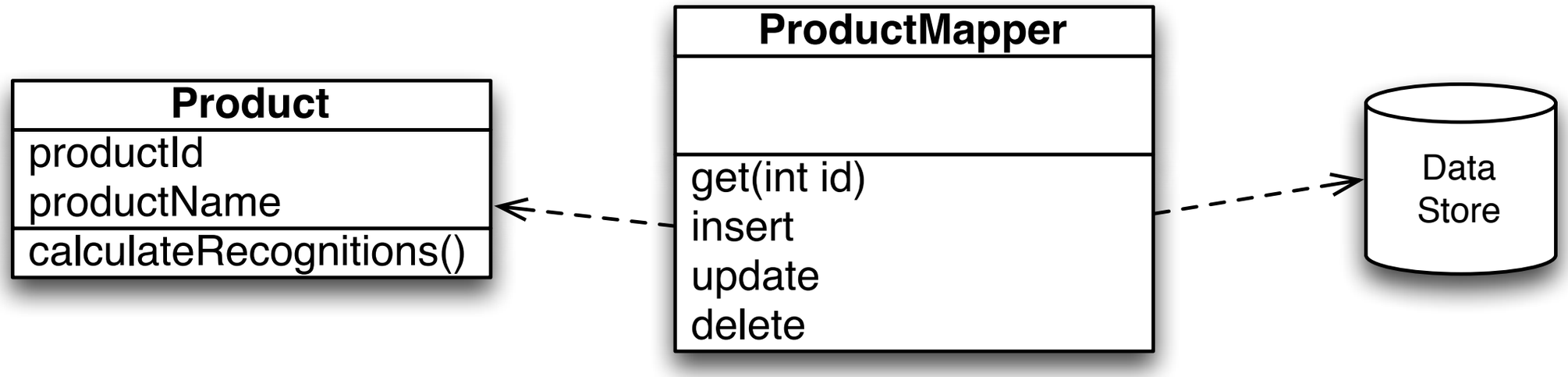
Active Record

Row Data Gateway + Business Logic



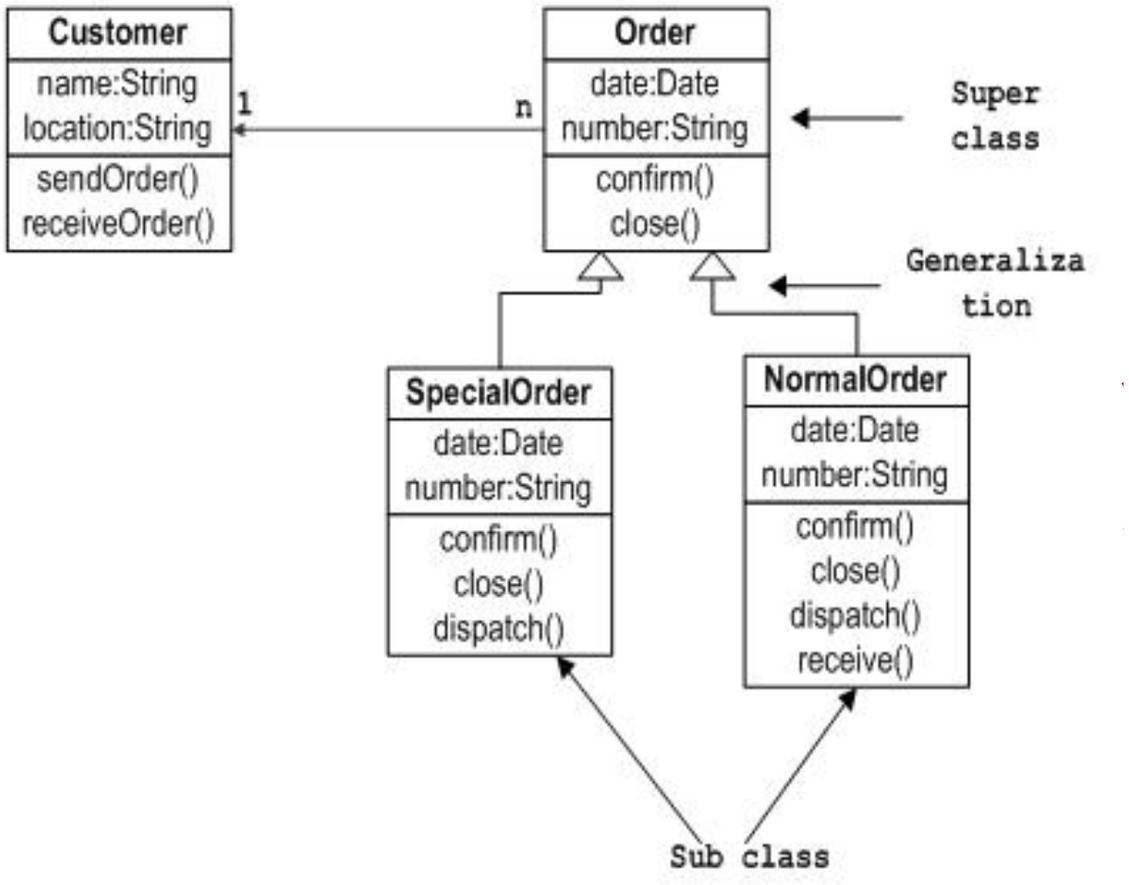
- Methods for:
  - Create instances from SLQ results
  - Insert new instances in the data store
  - Update data store based on instances data
  - Find relevant instances

Data Mapper

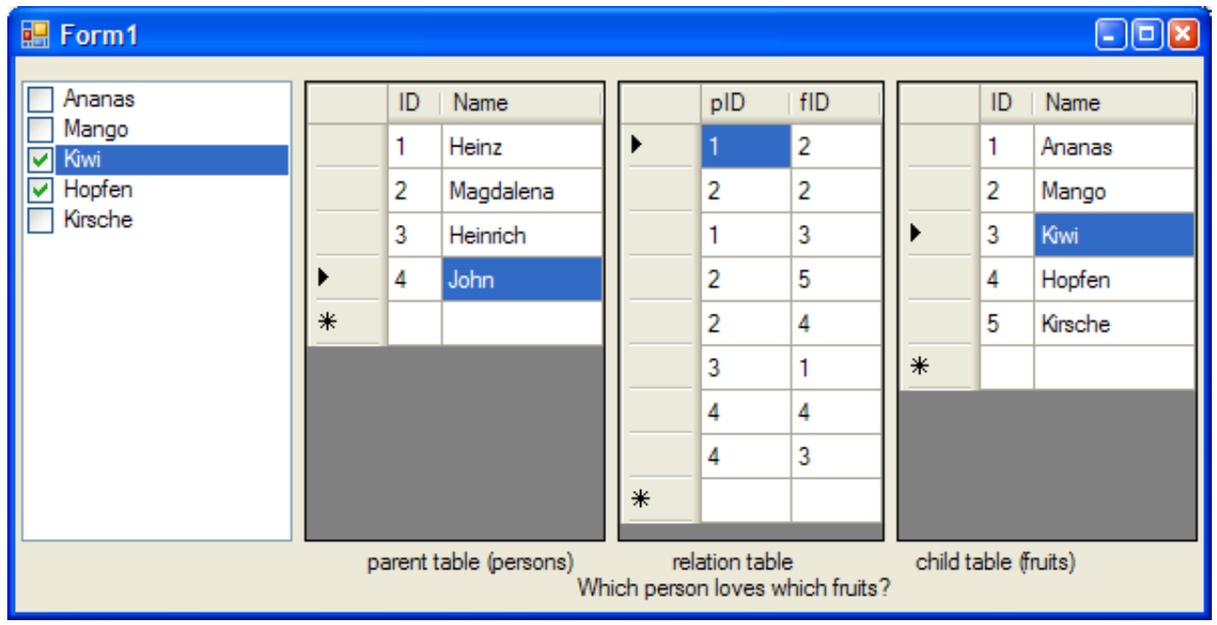


- Decouple objects structure from database structure
- There may be more than one mapper per domain object

Navigate (Relational) Data

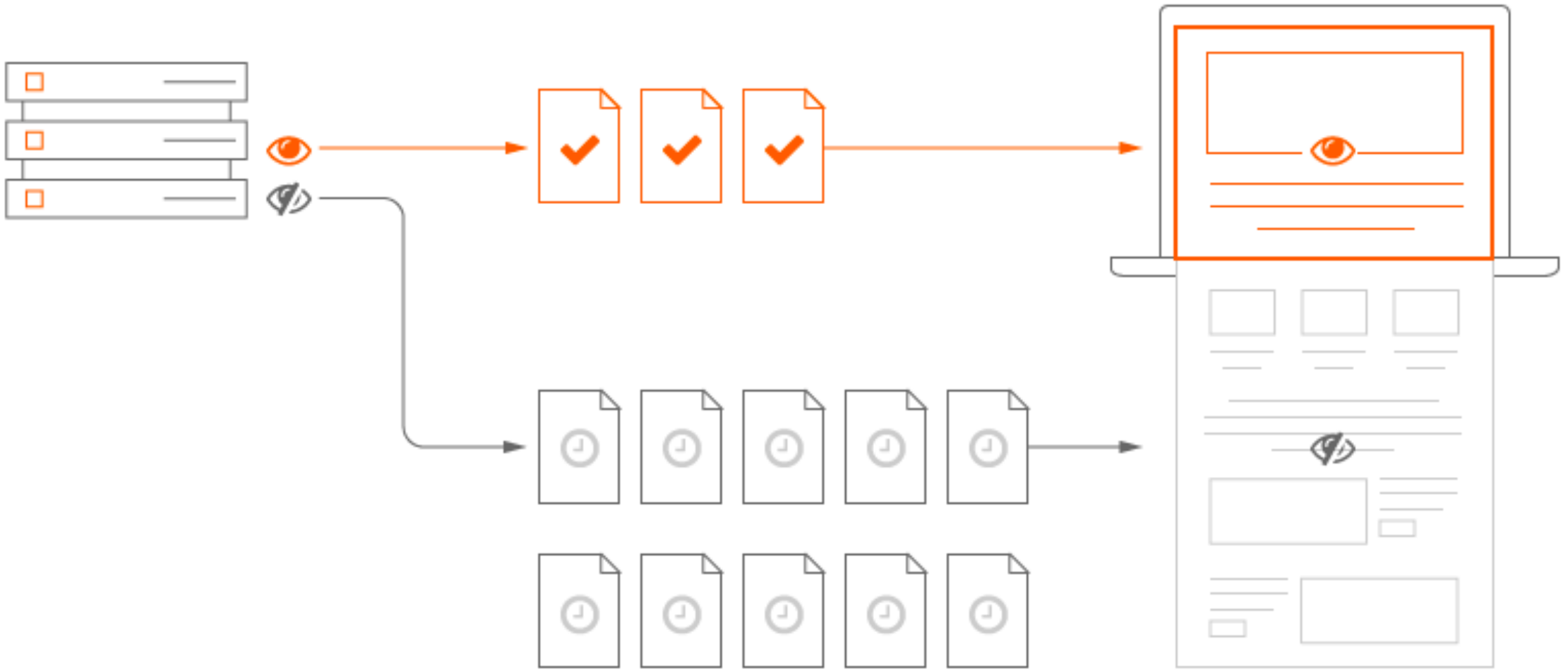


Traverse object graph

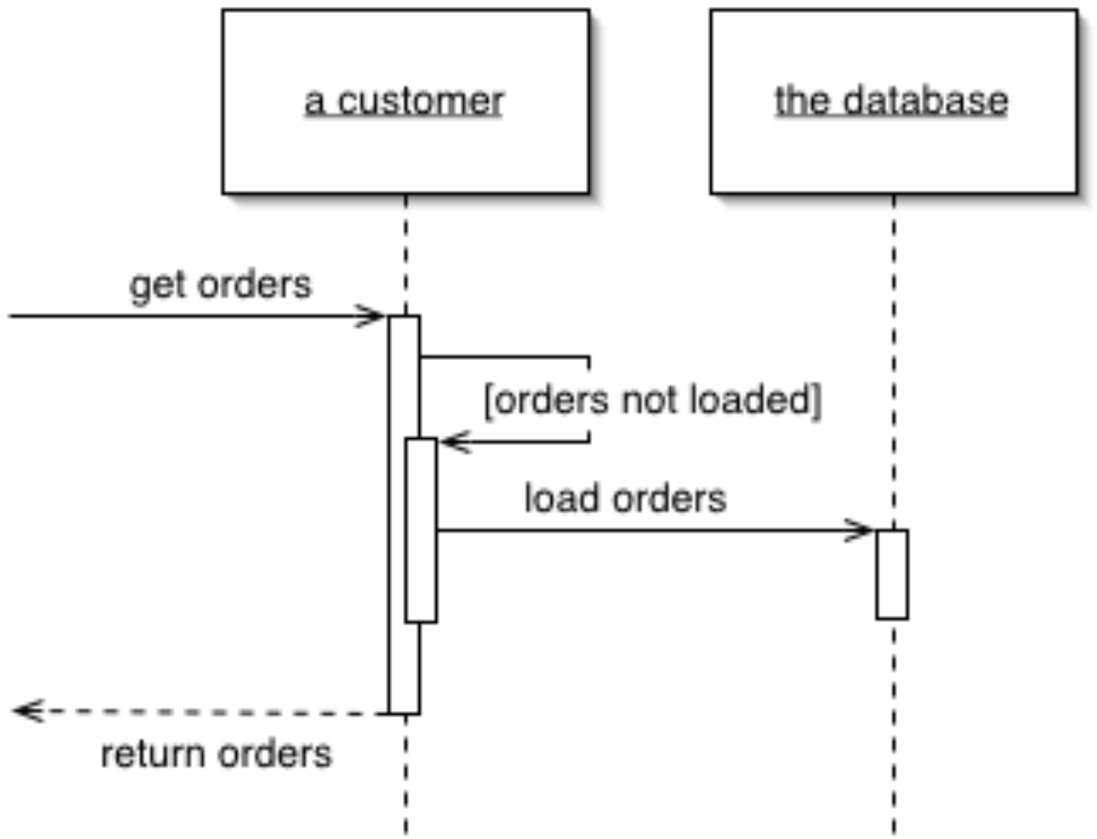


Join over foreign keys

Lazy Loading



Lazy Loading



Interrupt the load at some point and resume it later only if needed

Lazy Loading Patterns

- Lazy Initialization

  Checks if field is null at every access

- Value Holder

  Wraps lazy-loaded objects

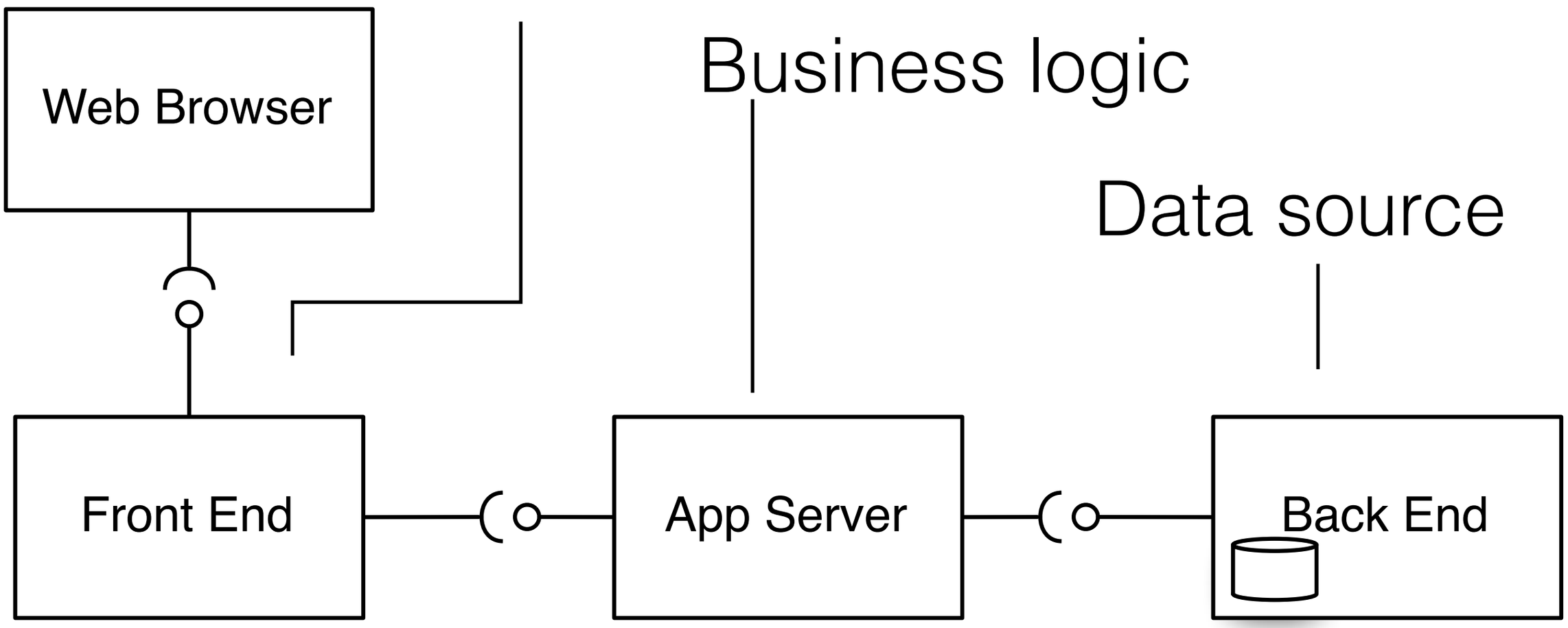
- Virtual Proxy

  Mocks field access and loads values on the demand

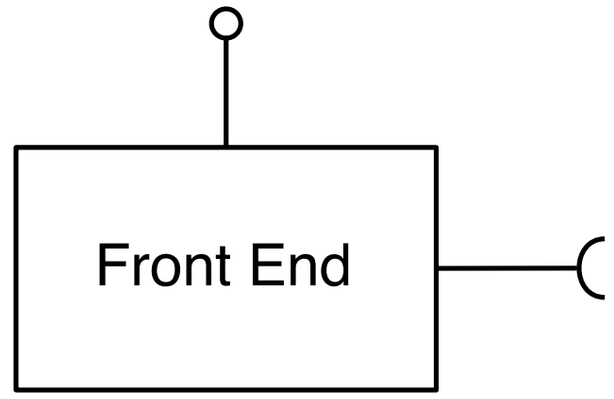
- Ghost

  Real object but partially loaded, missing data loaded on first access

3-Tier Architecture



Front End

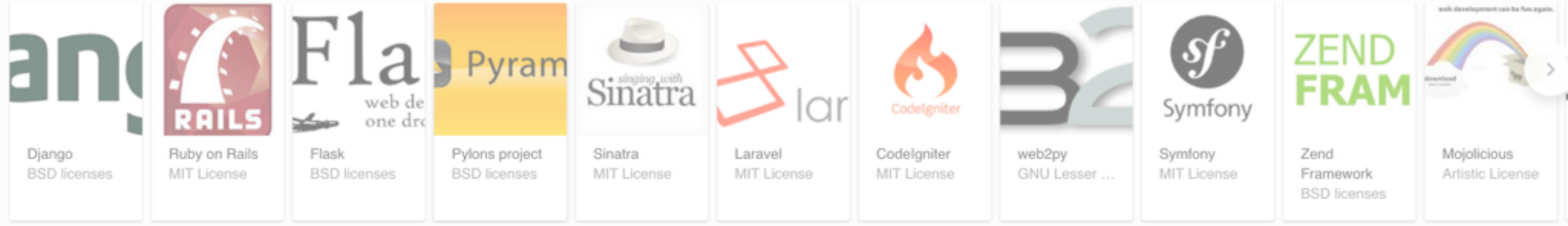


- Generate the HTML based on request and data from the backend

- Can handle client side interactions both inside the server and the client's browser

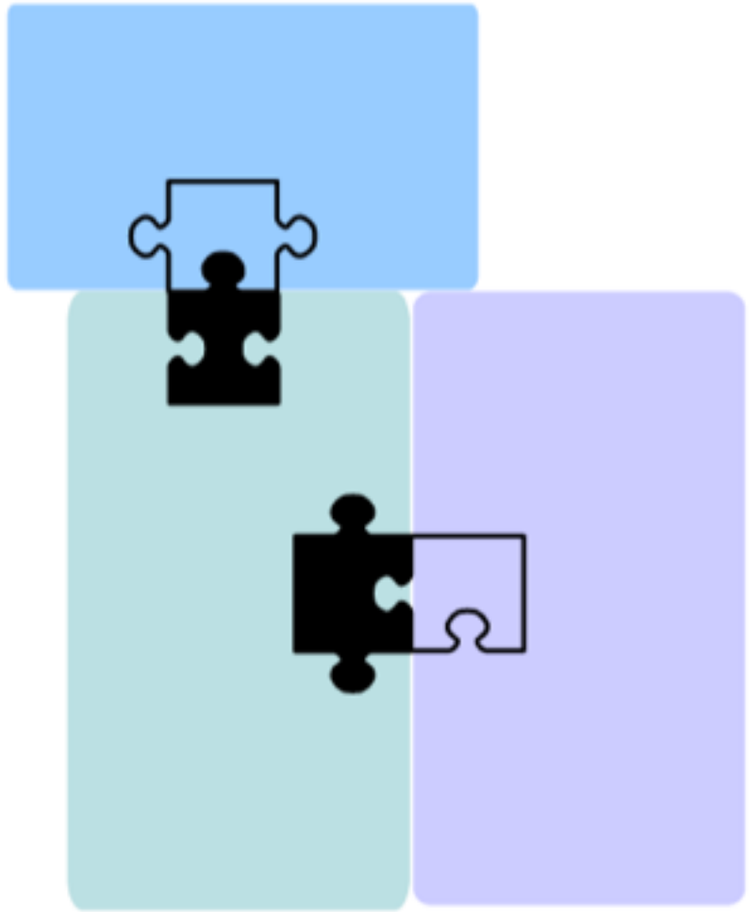
- Security, input validation, responsiveness, etc.

Web Frameworks



Plugin

- Explicit extension points
- Static/Dynamic composition
- Low security (3rd party code)
- Extensibility and customizability



Client Side

JavaScript

- a programming language executed in the browser
  nothing to do with java

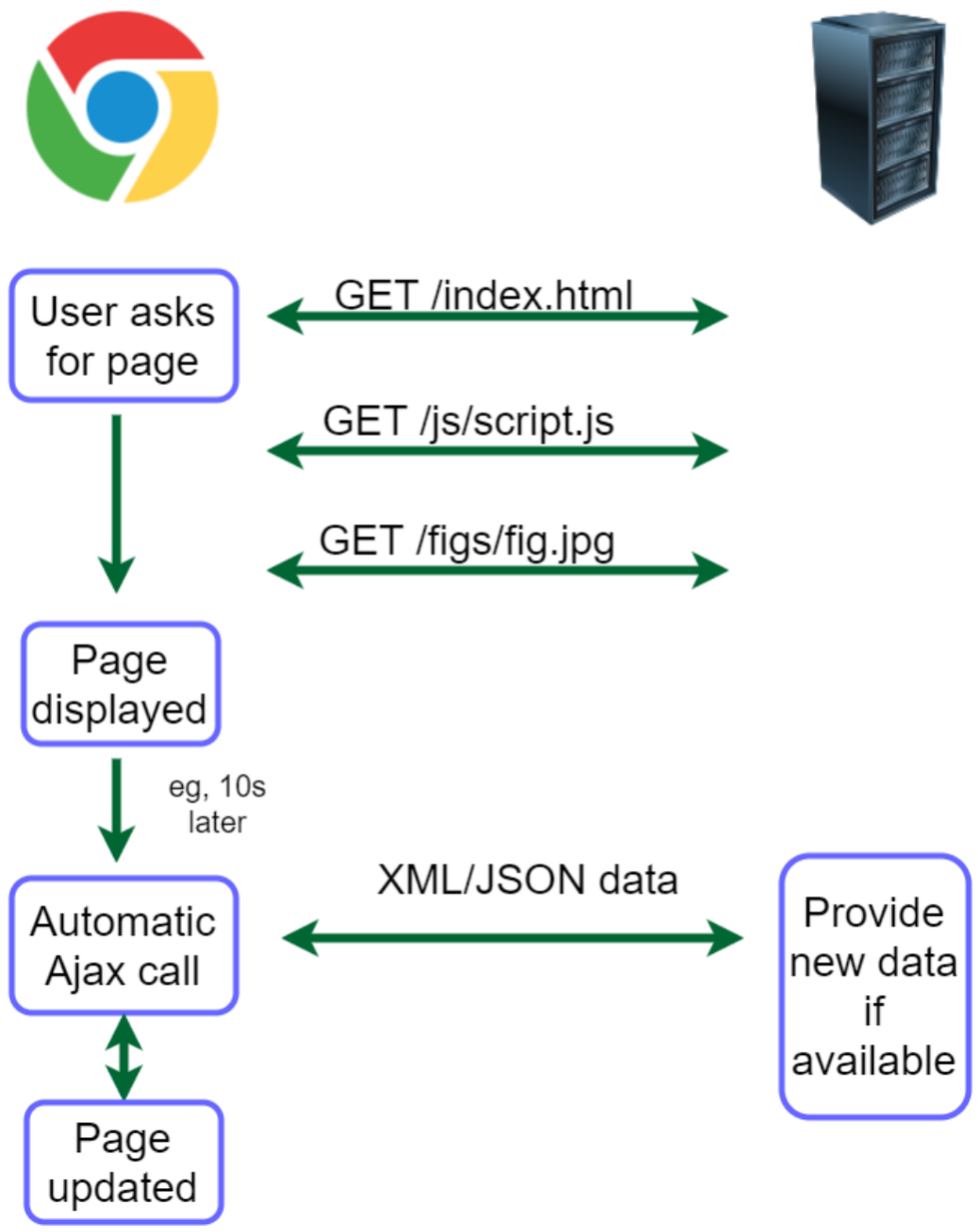
- JS files/libraries referenced by the web page
  like any other resource

- Can be inlined

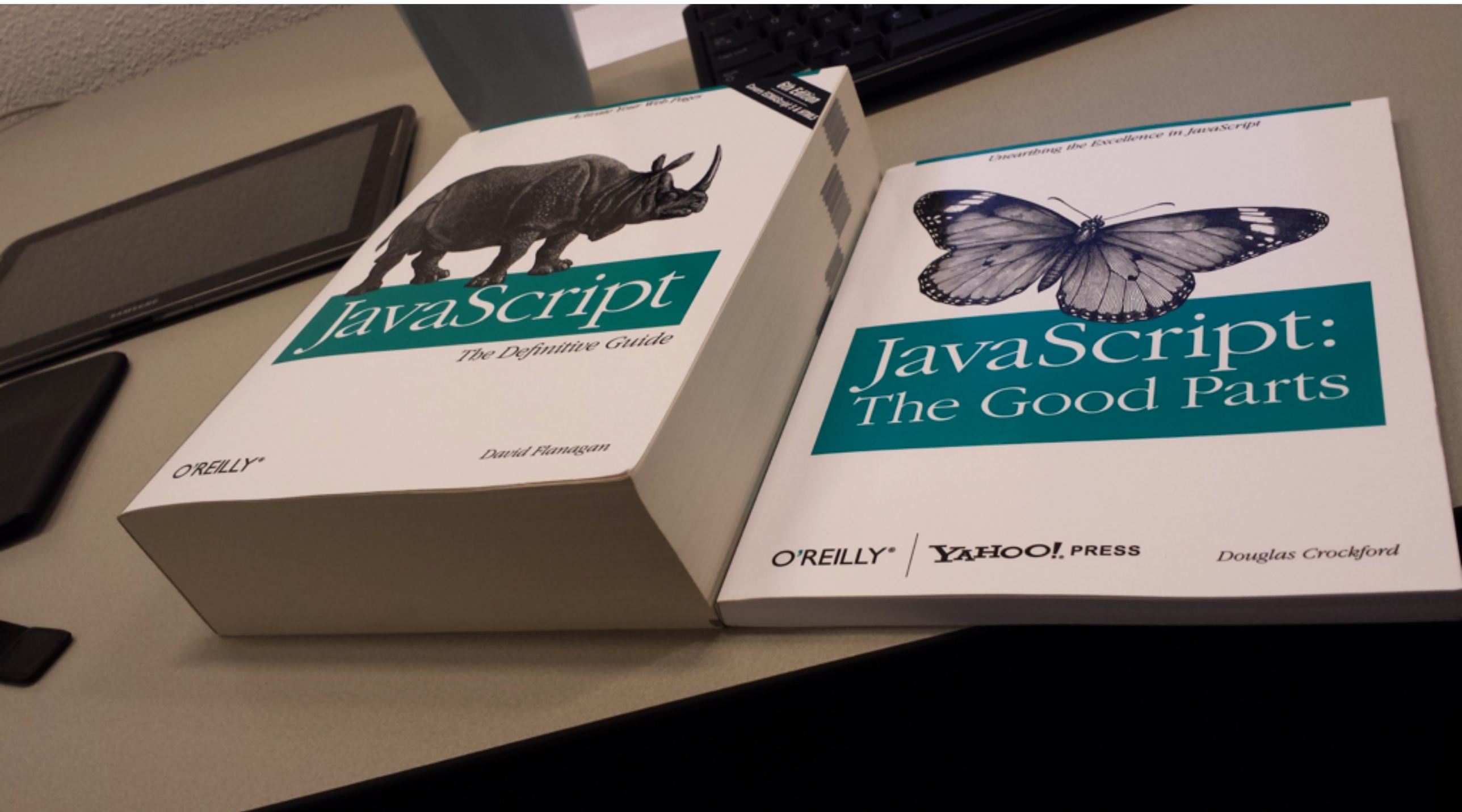
- Dynamically manipulates the page Document Object Model (DOM) to alter page's content, structure, and behavior

AJAX

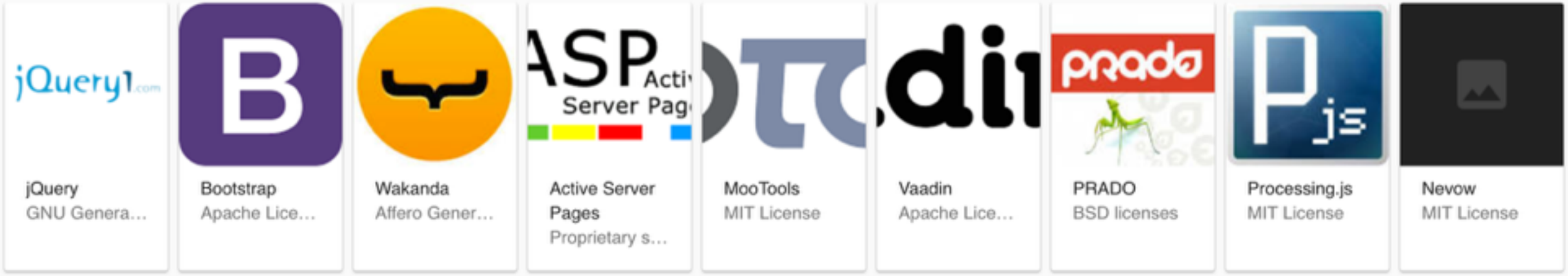
Asynchronous JavaScript and XML



JS not so (well designed) easy



JS Frameworks



Case Study

Main Scenarios

- A **user** requests an article during normal operation and gets the rendered article HTML page.
- An **editor** saves an edited article during normal operation and the article is saved.

How To Install Linux, Apache, × Alessio

https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04

Apps | Mercurial-Redmine... | Sign in · GitLab | Software Engineeri... | Home - Staff - Con... | TestingCloud | ReadingList | Tennis Page | Login - OpenStack... | Index of /~diam/ro... | Sign in · GitLab | Other Bookmarks

Brennen Bearnes Subscribe Share

Contents

Mark as Complete

How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 16.04

105

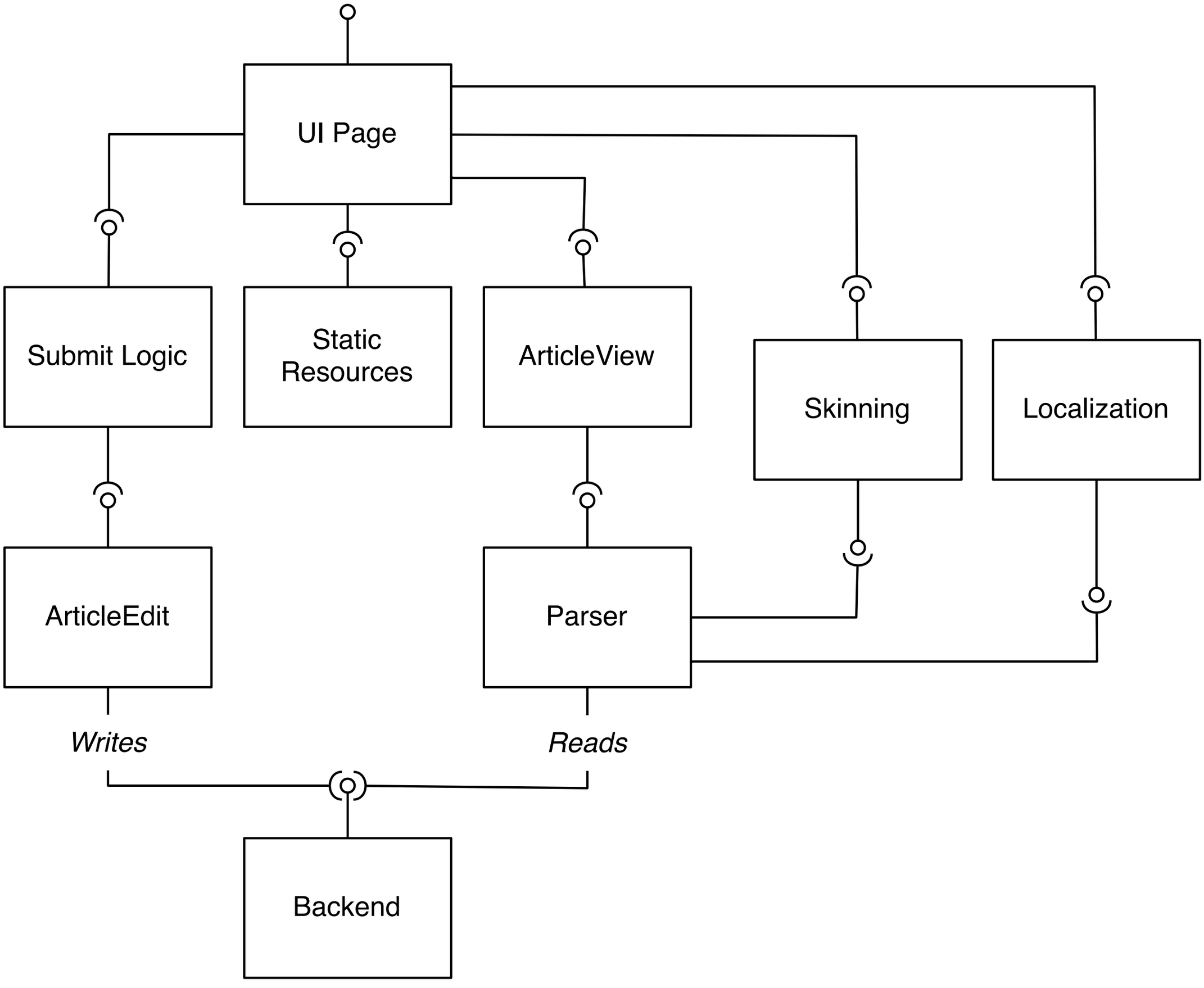
Posted April 21, 2016 1.2m LAMP STACK PHP MYSQL APACHE UBUNTU 16.04

Introduction

A "LAMP" stack is a group of open source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the **L**inux operating system, with the **A**pache web server. The site data is stored in a **M**ySQL database, and dynamic content is processed by **P**HP.

Introduction

A "LAMP" stack is a group of open source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the **L**inux operating system, with the **A**pache web server. The site data is stored in a **M**ySQL database, and dynamic content is processed by **P**HP.


UI Page
Submit Logic
Static Resources
ArticleView
Skinning
Localization
ArticleEdit
Parser
Writes
Reads
Backend

Qualities

- “Basic” implementation
  - Limited scalability
  - Single point of failure
  - Limited Security

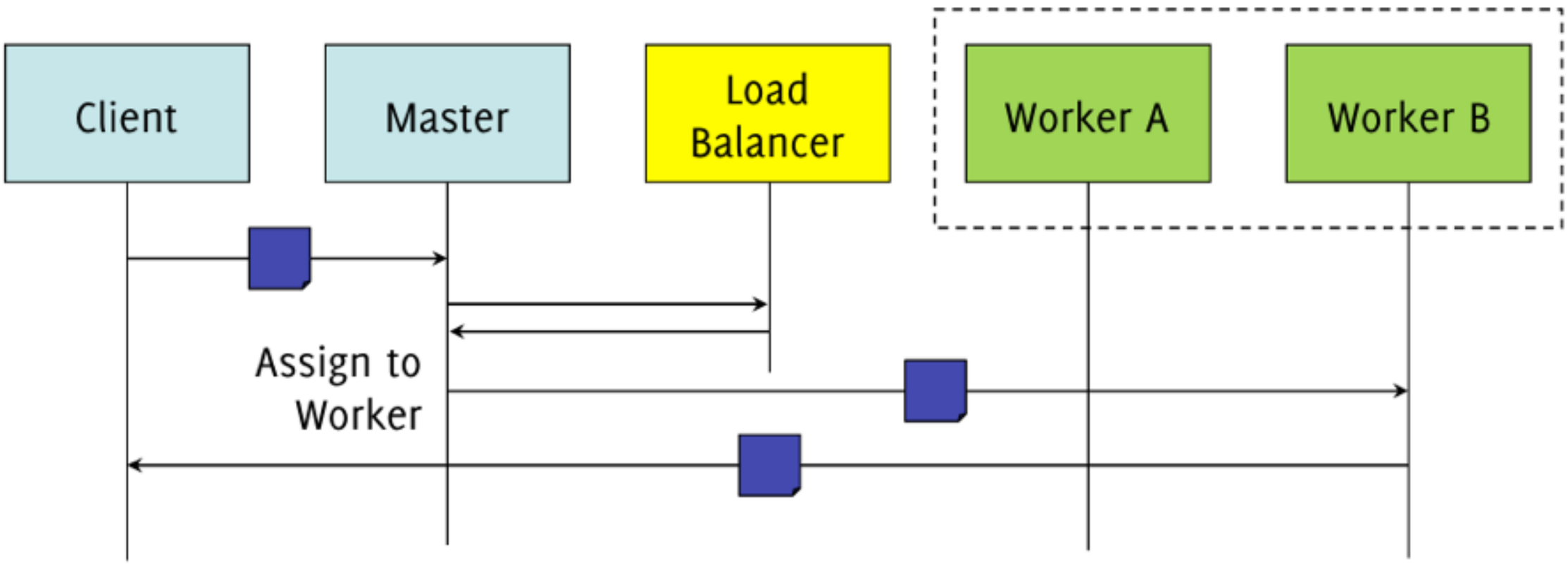
Performance Tactics

- Control Resource Demand
  - *Increase the resource efficiency (caching)*
  - *Reduce overhead (pre-generate HTML from PHP)*
- Manage Resources
  - *Schedule resources (load balancer)*

https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application

Load Balancing

deploy many replicated instances of the server on multiple machines



squid - Google Search | squid : Optimising Web Delive | Squid (software) - Wikipedia | Alessio

www.squid-cache.org

Apps | Mercurial-Redmine... | Sign in · GitLab | Software Engineeri... | Home - Staff - Con... | TestingCloud | ReadingList | Tennis Page | Login - OpenStack... | Index of /~diam/ro... | Sign in · GitLab | Other Bookmarks

squid-cache.org

Optimising Web Delivery

docs | download | donate | support | about | contact | shop | blog

Squid: Optimising Web Delivery

Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.

Introduction

About Squid
Why Squid?
Squid Developers
How to Donate
How to Help Out
Getting Squid
Squid Source Packages
Squid Deployment Case-Studies
Squid Software Foundation

Documentation

Configuration:
- Reference
- Examples

FAQ and Wiki

Guide Books:
- Beginners
- Definitive

Non-English
More...

Support

Making the most of your Internet Connection

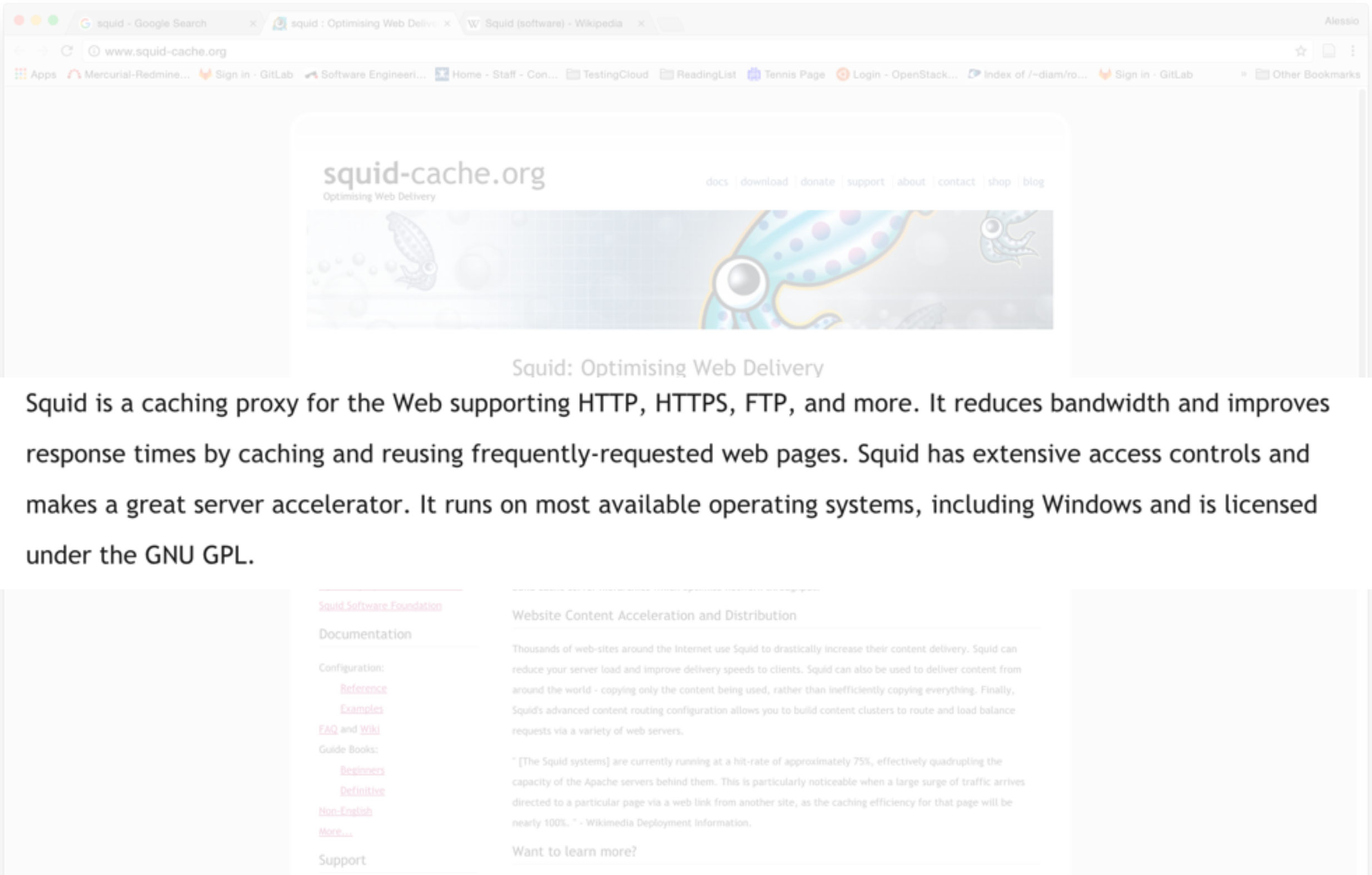
Squid is used by hundreds of Internet Providers world-wide to provide their users with the best possible web access. Squid optimises the data flow between client and server to improve performance and caches frequently-used content to save bandwidth. Squid can also route content requests to servers in a wide variety of ways to build cache server hierarchies which optimise network throughput.

Website Content Acceleration and Distribution

Thousands of web-sites around the Internet use Squid to drastically increase their content delivery. Squid can reduce your server load and improve delivery speeds to clients. Squid can also be used to deliver content from around the world - copying only the content being used, rather than inefficiently copying everything. Finally, Squid's advanced content routing configuration allows you to build content clusters to route and load balance requests via a variety of web servers.

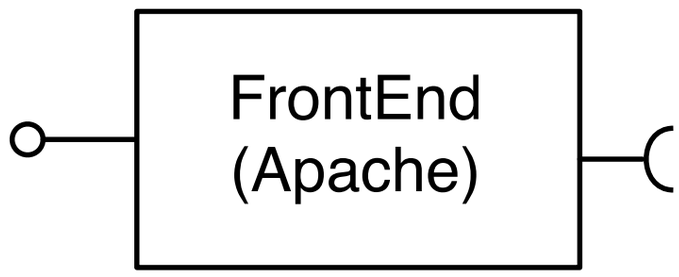
" [The Squid systems] are currently running at a hit-rate of approximately 75%, effectively quadrupling the capacity of the Apache servers behind them. This is particularly noticeable when a large surge of traffic arrives directed to a particular page via a web link from another site, as the caching efficiency for that page will be nearly 100%. " - Wikimedia Deployment Information.

Want to learn more?

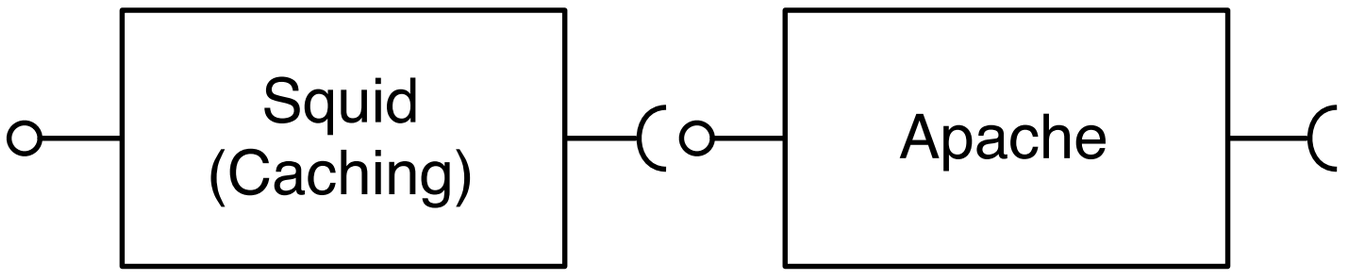


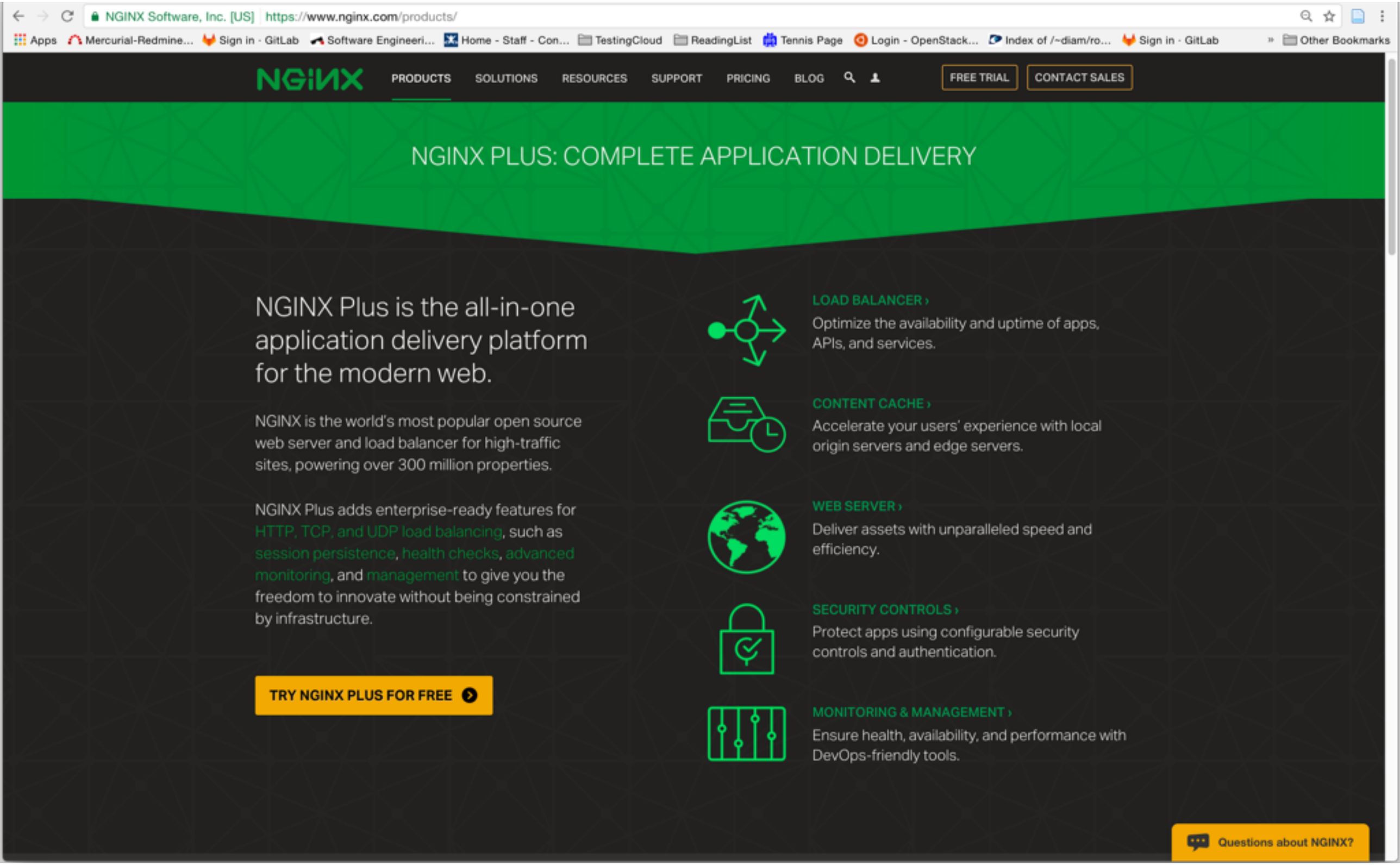
Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.

Caching + Load Balancing

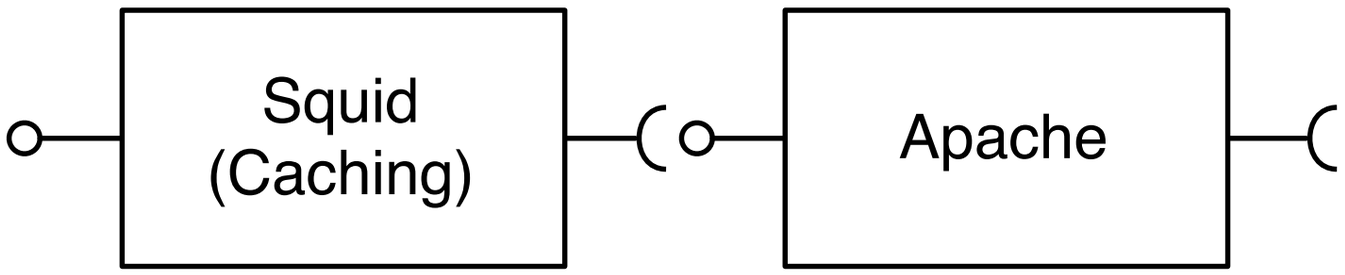


Caching + Load Balancing

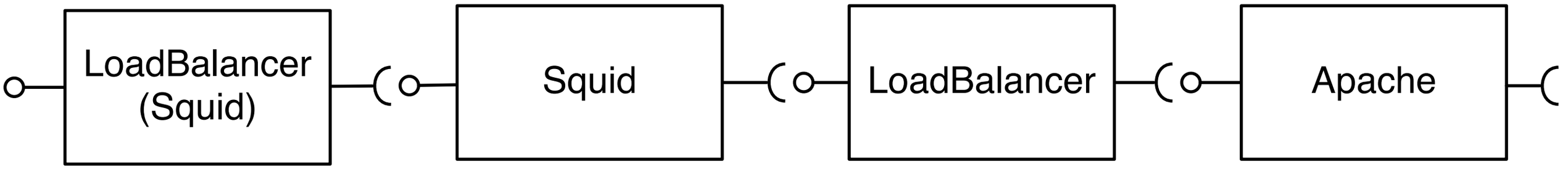



NGINX Software, Inc. [US] https://www.nginx.com/products/
Apps
Mercurial-Redmine...
Sign in · GitLab
Software Engineeri...
Home - Staff - Con...
TestingCloud
ReadingList
Tennis Page
Login - OpenStack...
Index of /~diam/ro...
Sign in · GitLab
Other Bookmarks
NGINX
PRODUCTS
SOLUTIONS
RESOURCES
SUPPORT
PRICING
BLOG
FREE TRIAL
CONTACT SALES
NGINX PLUS: COMPLETE APPLICATION DELIVERY
NGINX Plus is the all-in-one application delivery platform for the modern web.
NGINX is the world's most popular open source web server and load balancer for high-traffic sites, powering over 300 million properties.
NGINX Plus adds enterprise-ready features for HTTP, TCP, and UDP load balancing, such as session persistence, health checks, advanced monitoring, and management to give you the freedom to innovate without being constrained by infrastructure.
TRY NGINX PLUS FOR FREE
LOAD BALANCER ›
Optimize the availability and uptime of apps, APIs, and services.
CONTENT CACHE ›
Accelerate your users' experience with local origin servers and edge servers.
WEB SERVER ›
Deliver assets with unparalleled speed and efficiency.
SECURITY CONTROLS ›
Protect apps using configurable security controls and authentication.
MONITORING & MANAGEMENT ›
Ensure health, availability, and performance with DevOps-friendly tools.
Questions about NGINX?

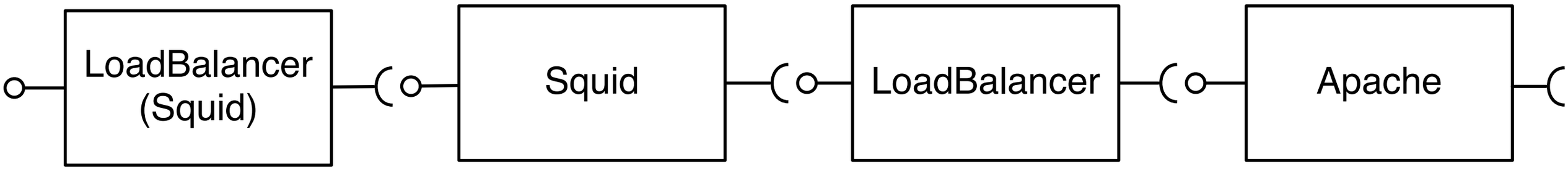
Caching + Load Balancing

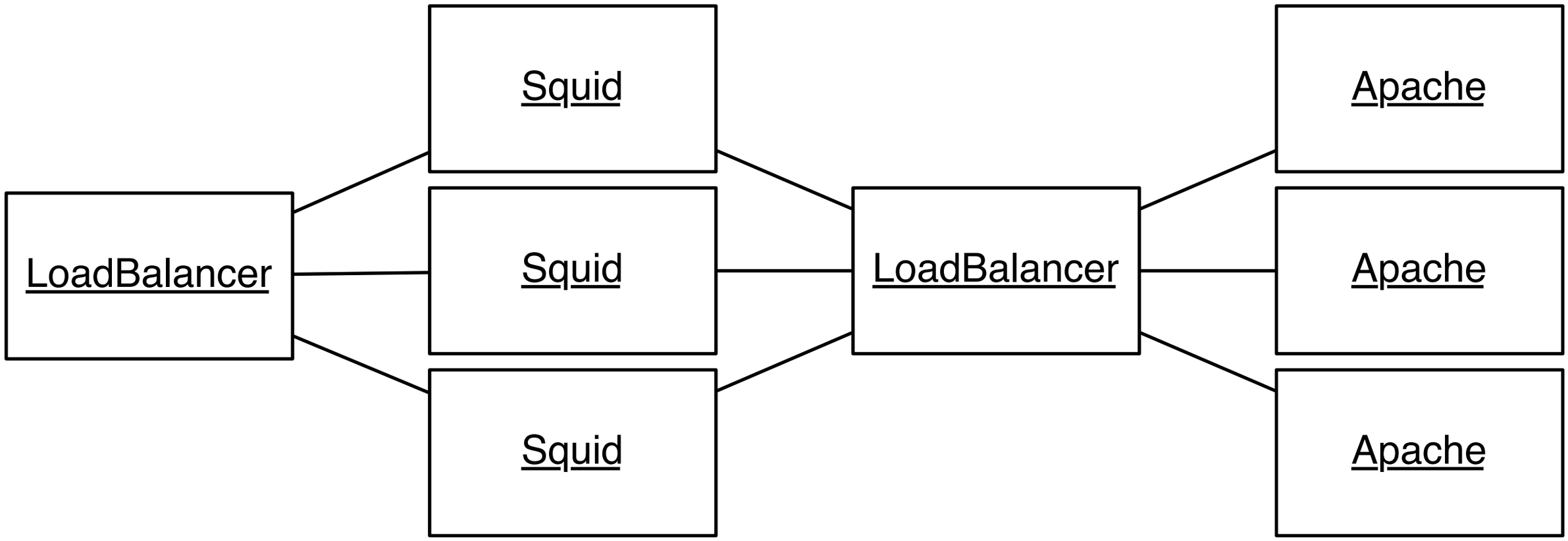


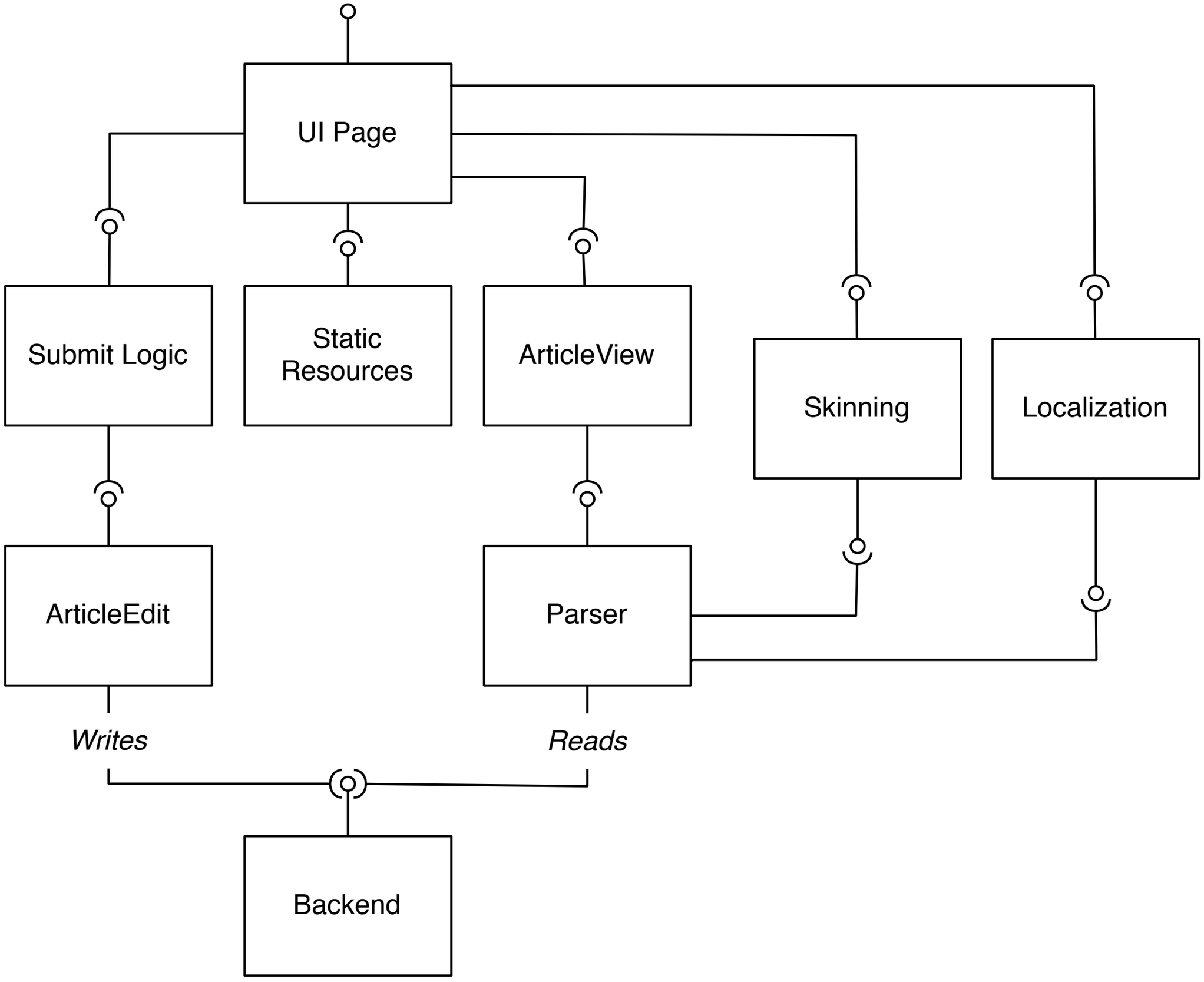
Caching + Load Balancing

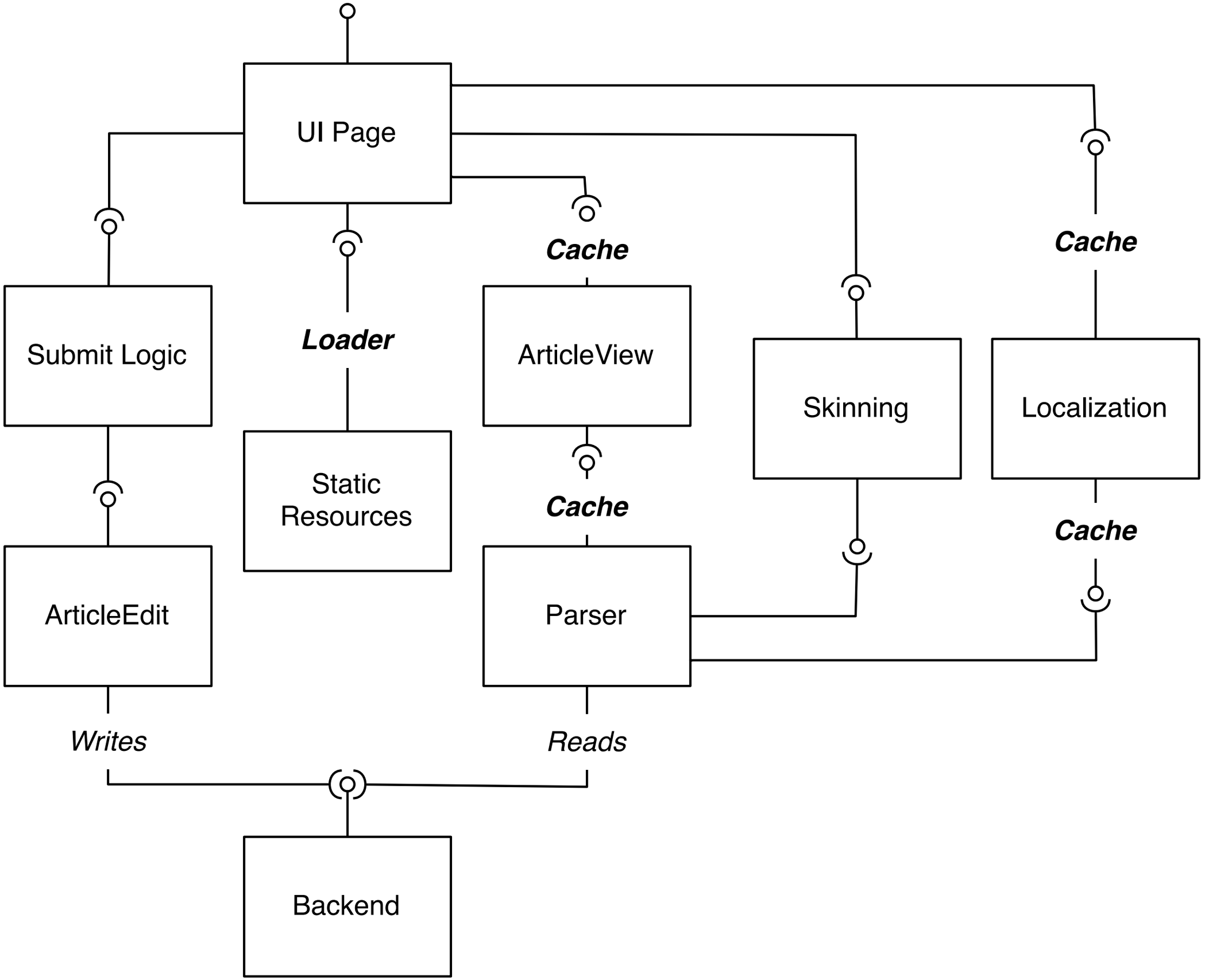


Caching + Load Balancing






UI Page
Submit Logic
Static Resources
ArticleView
Skinning
Localization
ArticleEdit
Parser
Writes
Reads
Backend


UI Page
Submit Logic
Loader
Cache
Skinning
Cache
ArticleView
Localization
Static Resources
Cache
Cache
ArticleEdit
Parser
Writes
Reads
Backend

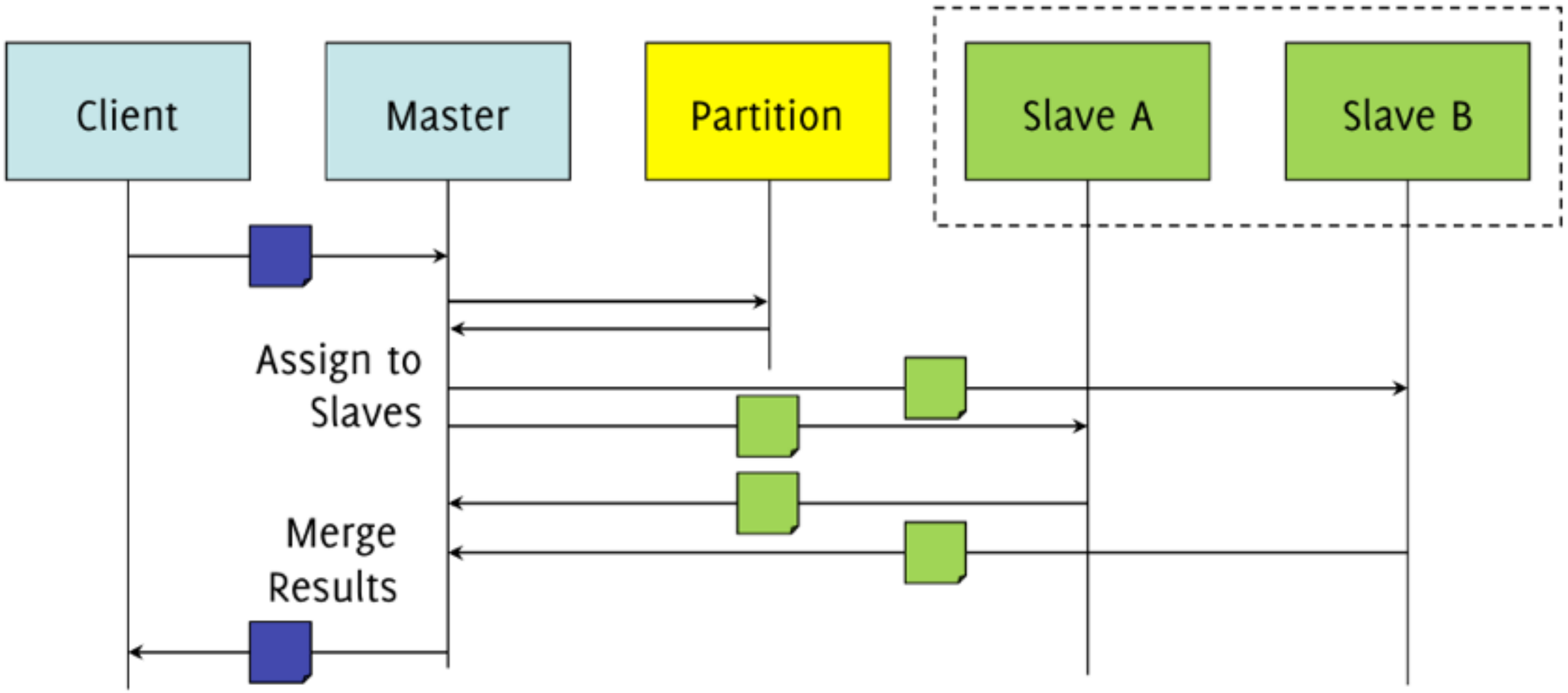
Performance Tactics

- Control Resource Demand
  - *Prioritize events (deferred article updates)*
- Manage Resources
  - *Introduce concurrency (distributed database)*
  - *Schedule resources (load balancer)*
  - *Multiple copies of data and computations*

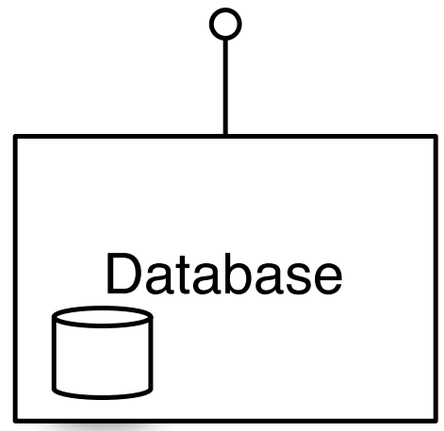
https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application

Master/Slave

split a large job into smaller independent partitions which can be processed in parallel



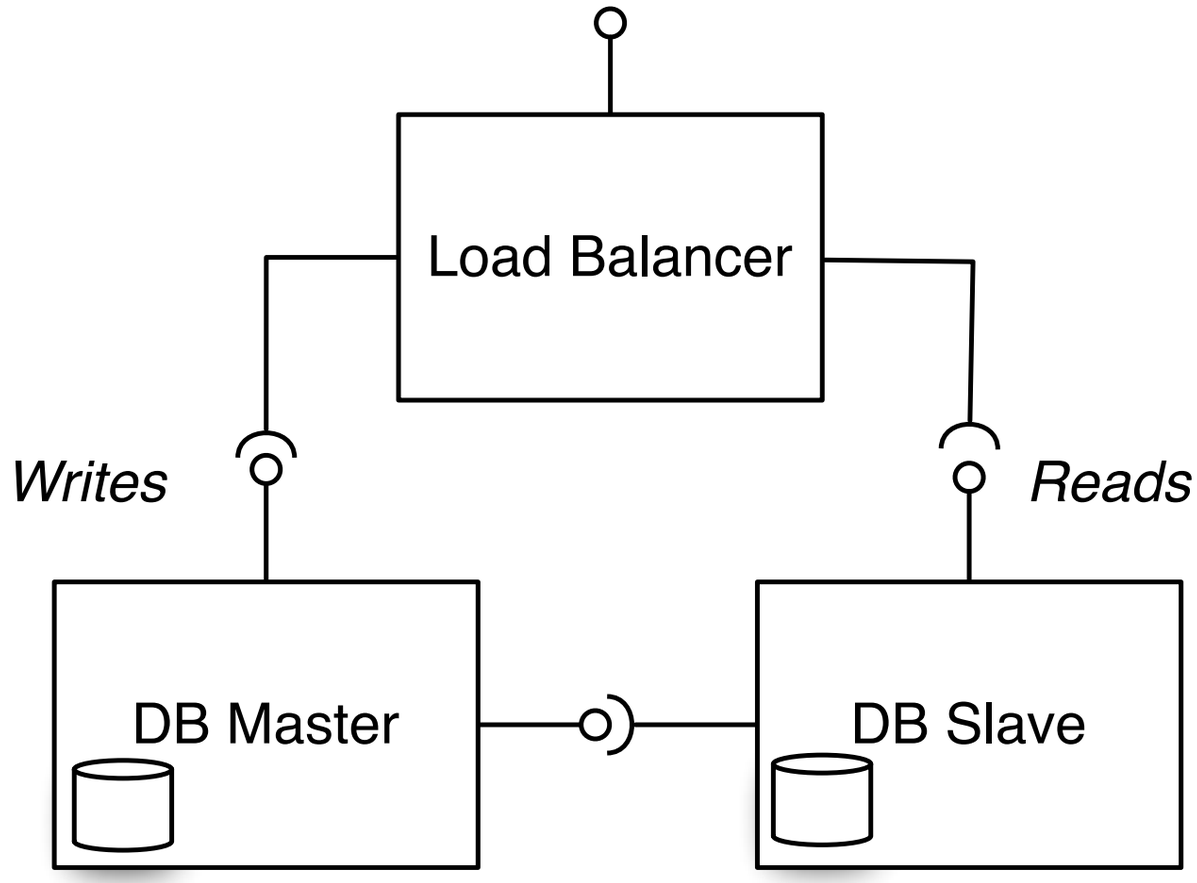
Distribution + Replication



More reads than writes

Near-live updates (no strict consistency requirements)

Distribution + Replication

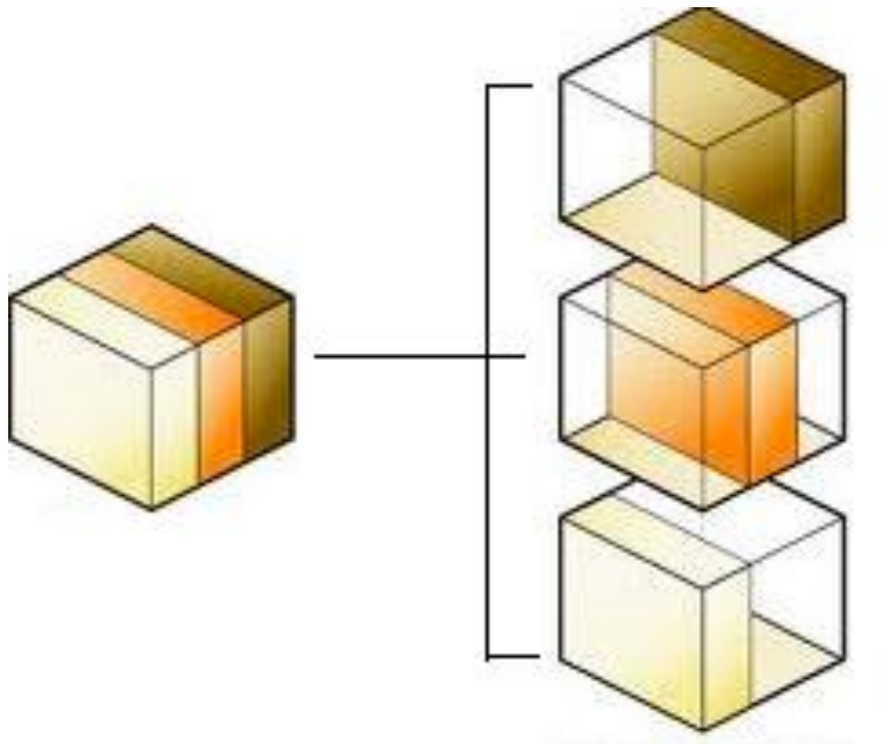


More reads than writes

Near-live updates (no strict consistency requirements)

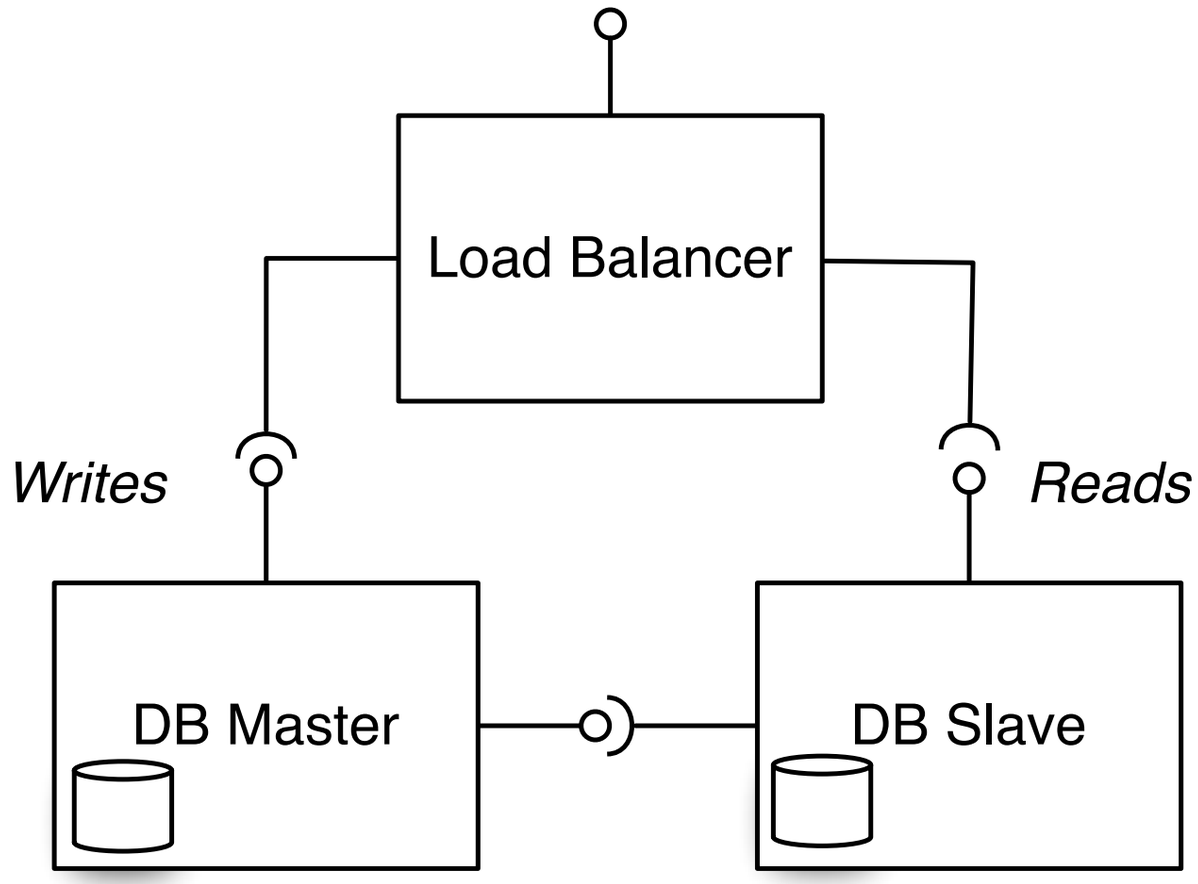
Distribution + Replication

Data Sharding

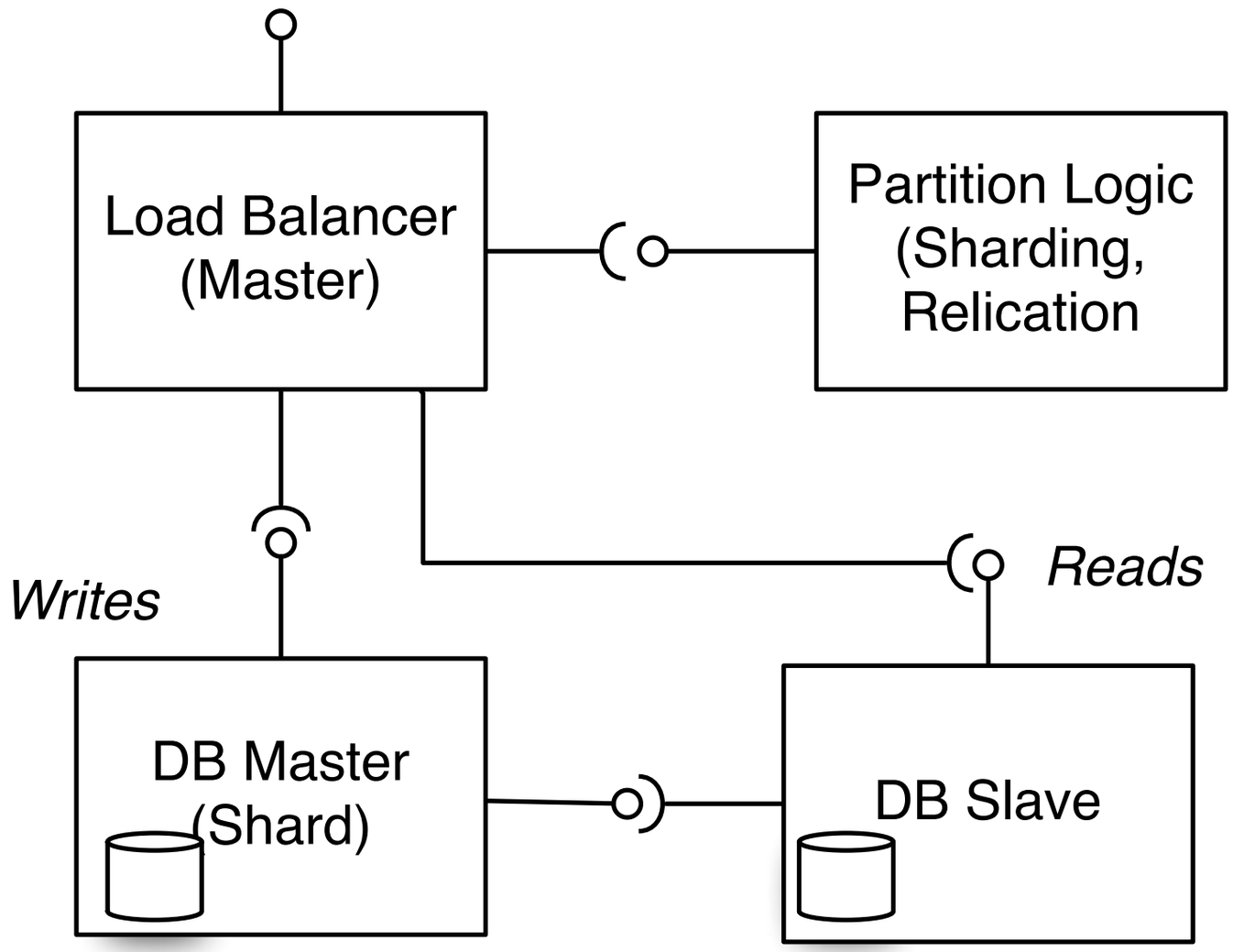


Clear data separation (Article Name: A-B, C-D, etc..)

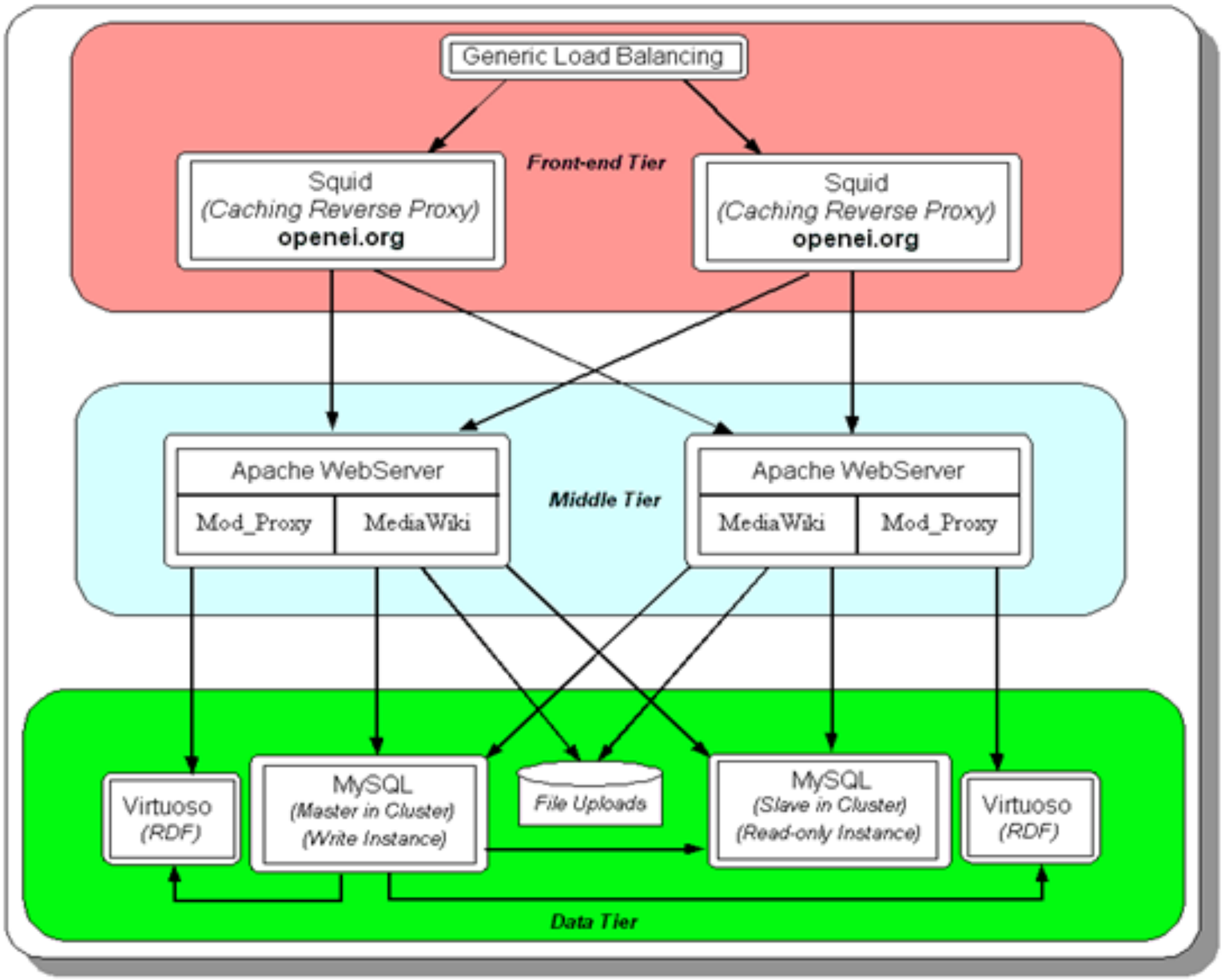
Distribution + Replication

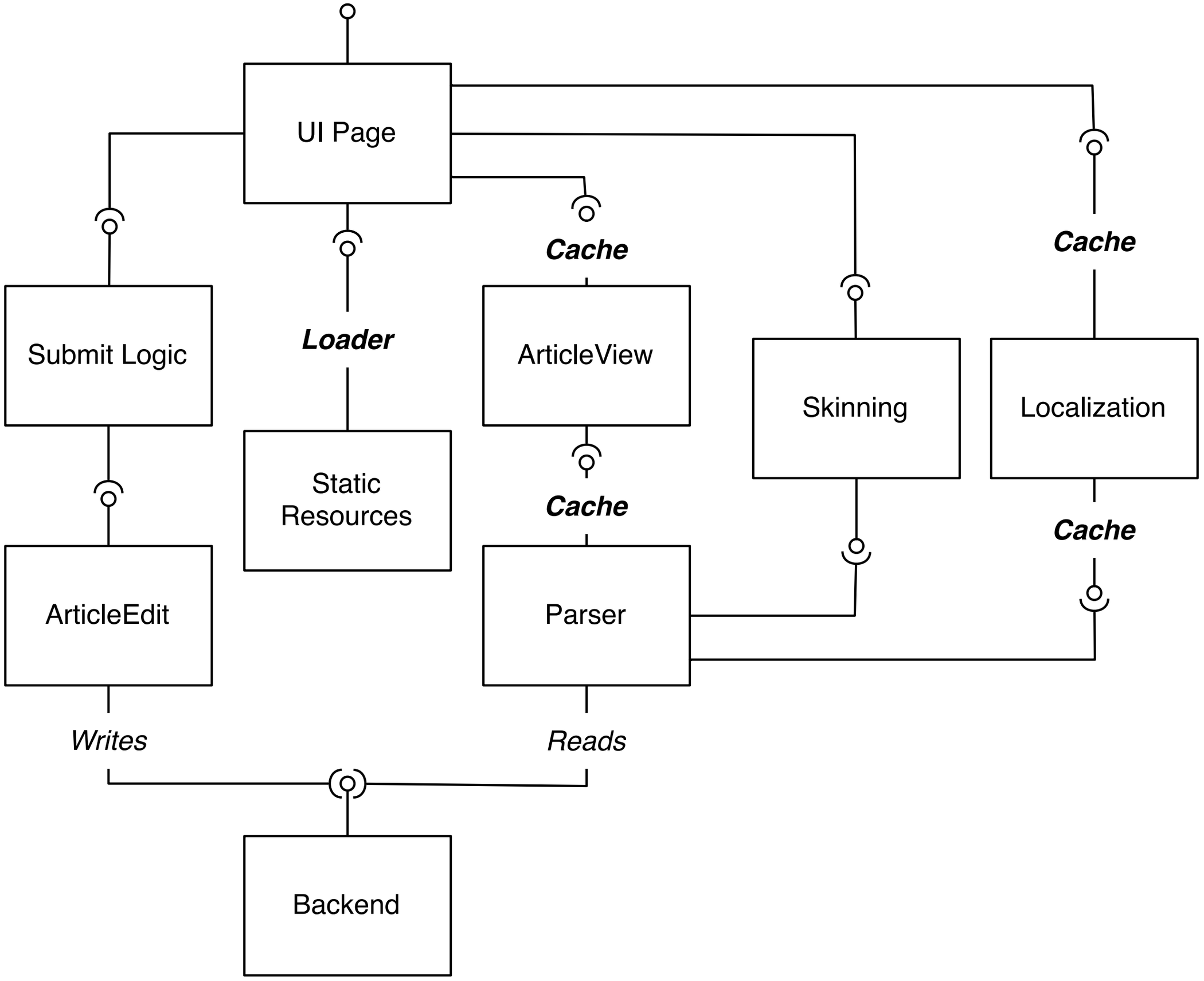


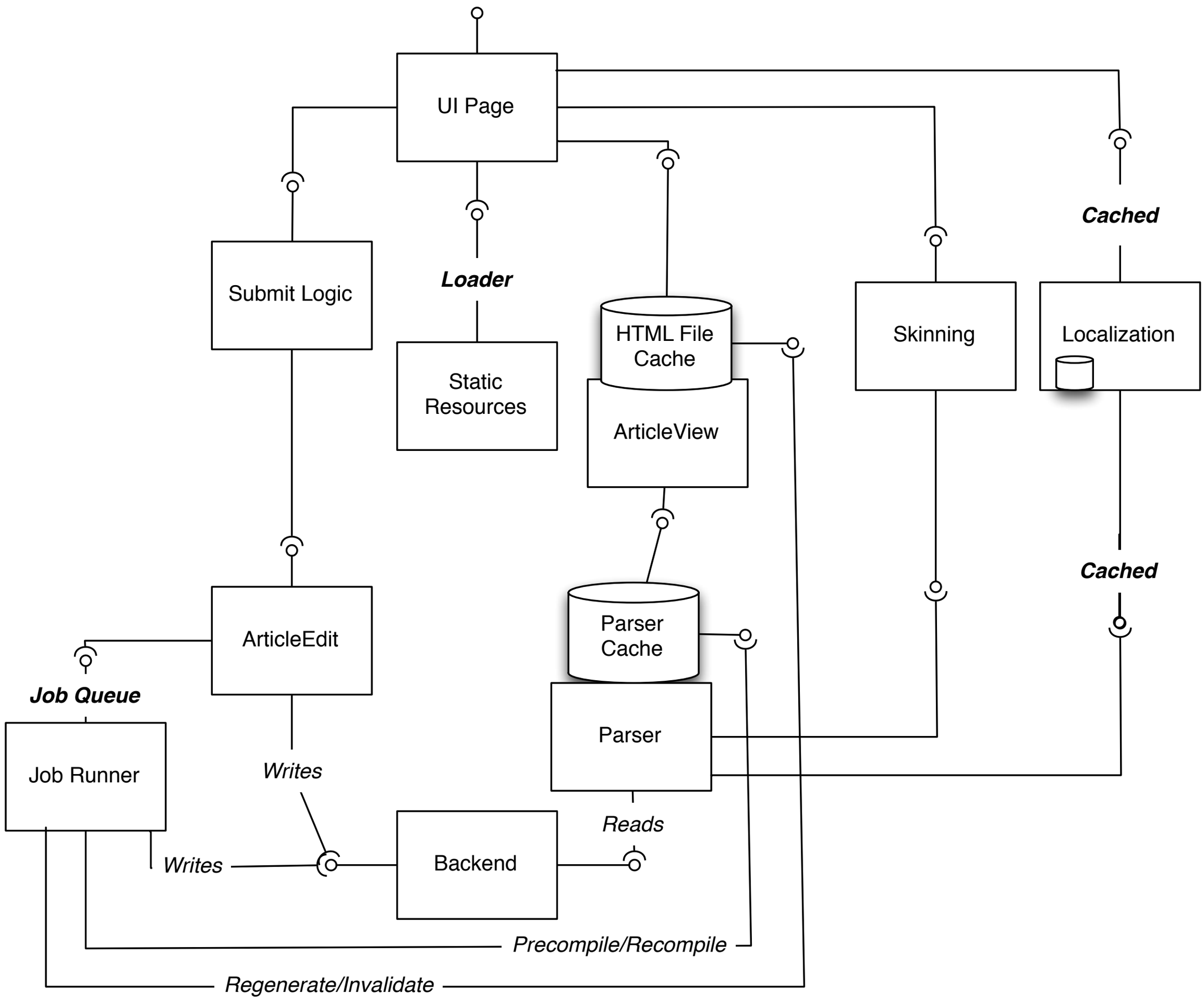
Distribution + Replication



Visualization




UI Page
Submit Logic
Loader
Cache
ArticleView
Skinning
Cache
Localization
Static Resources
Cache
Cache
ArticleEdit
Parser
Writes
Reads
Backend

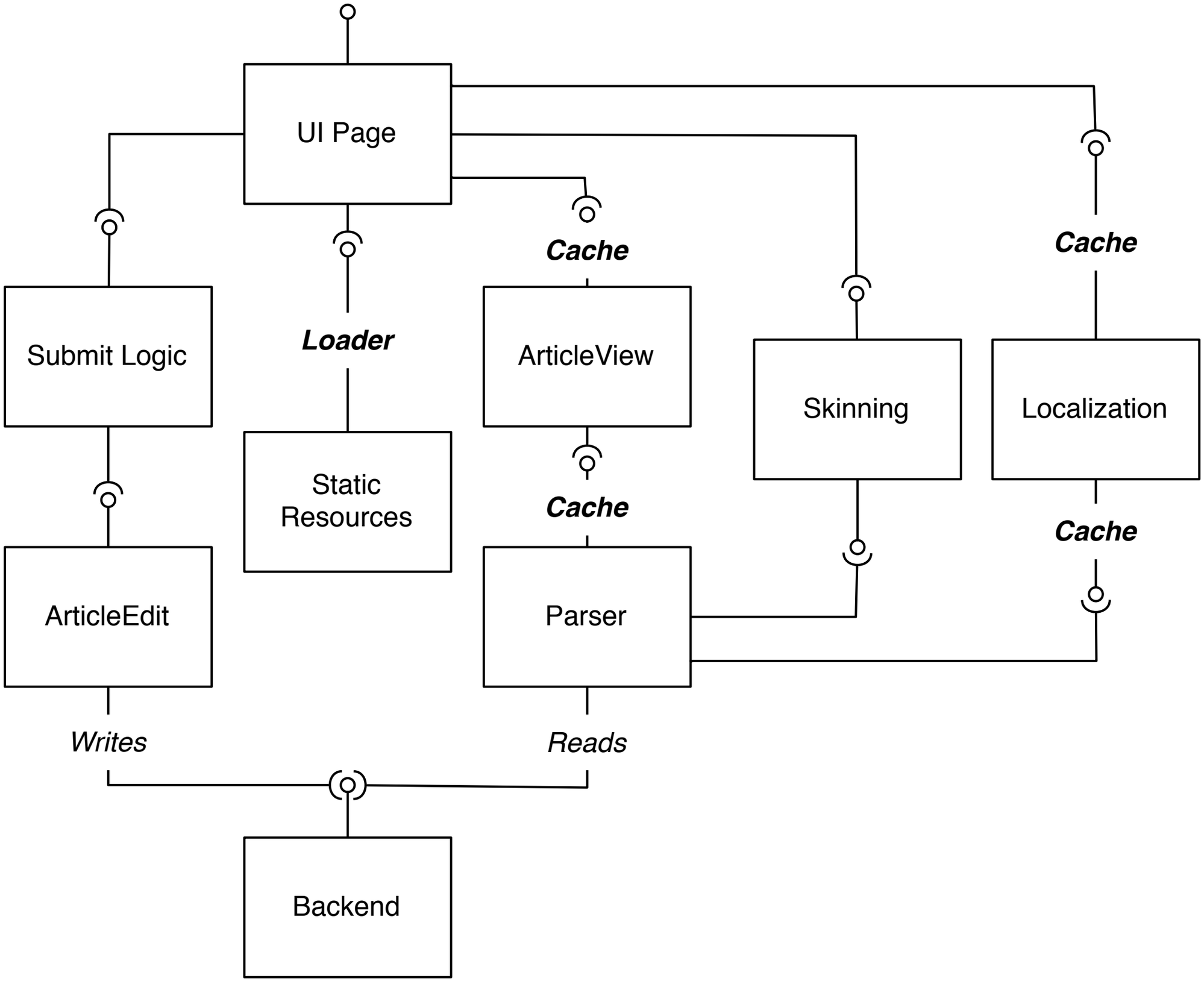

UI Page
Submit Logic
Loader
Static Resources
HTML File Cache
ArticleView
Skinning
Localization
Cached
Cached
ArticleEdit
Job Queue
Job Runner
Writes
Writes
Backend
Parser Cache
Parser
Reads
Precompile/Recompile
Regenerate/Invalidate

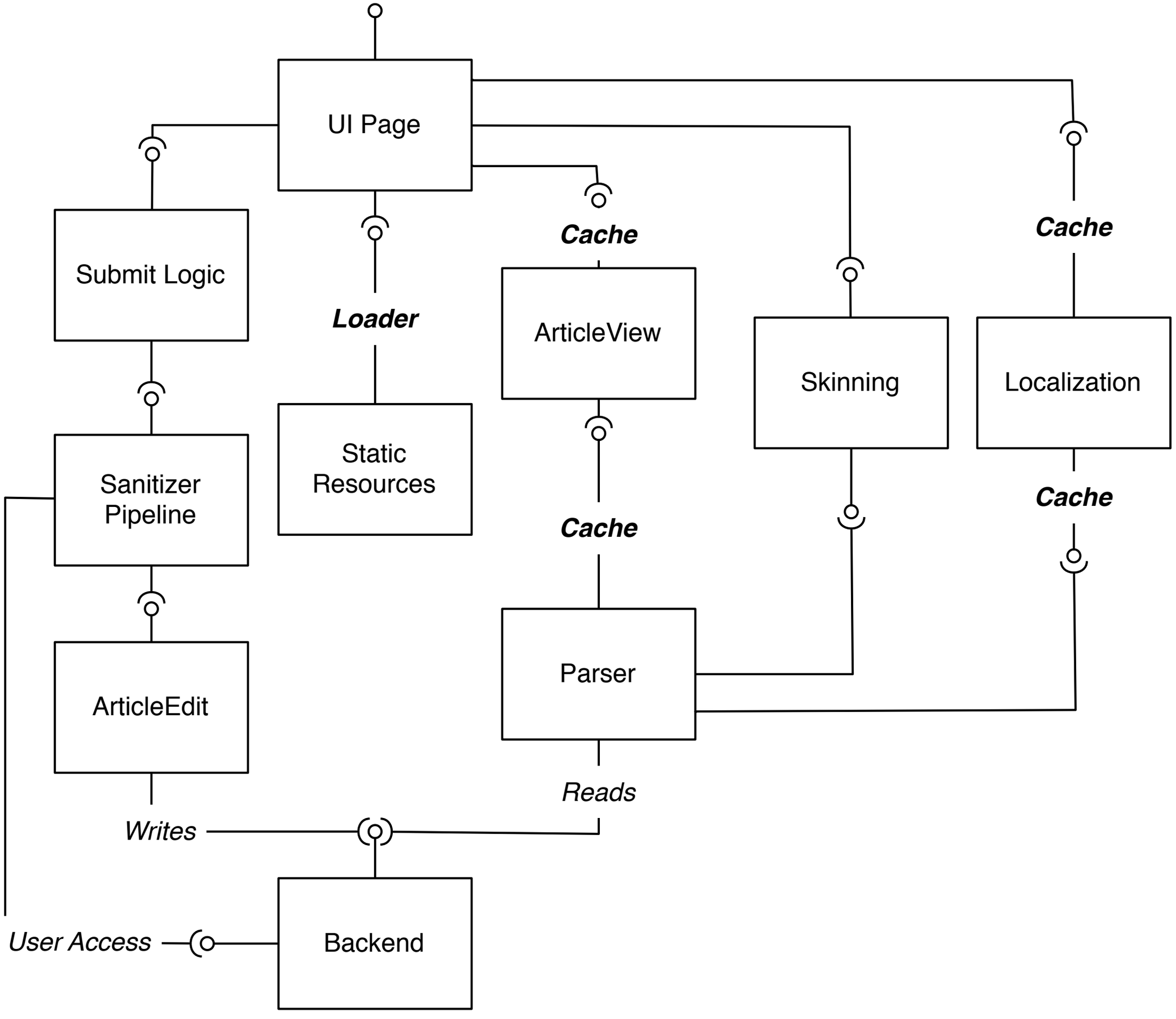
Security/Availability Tactics

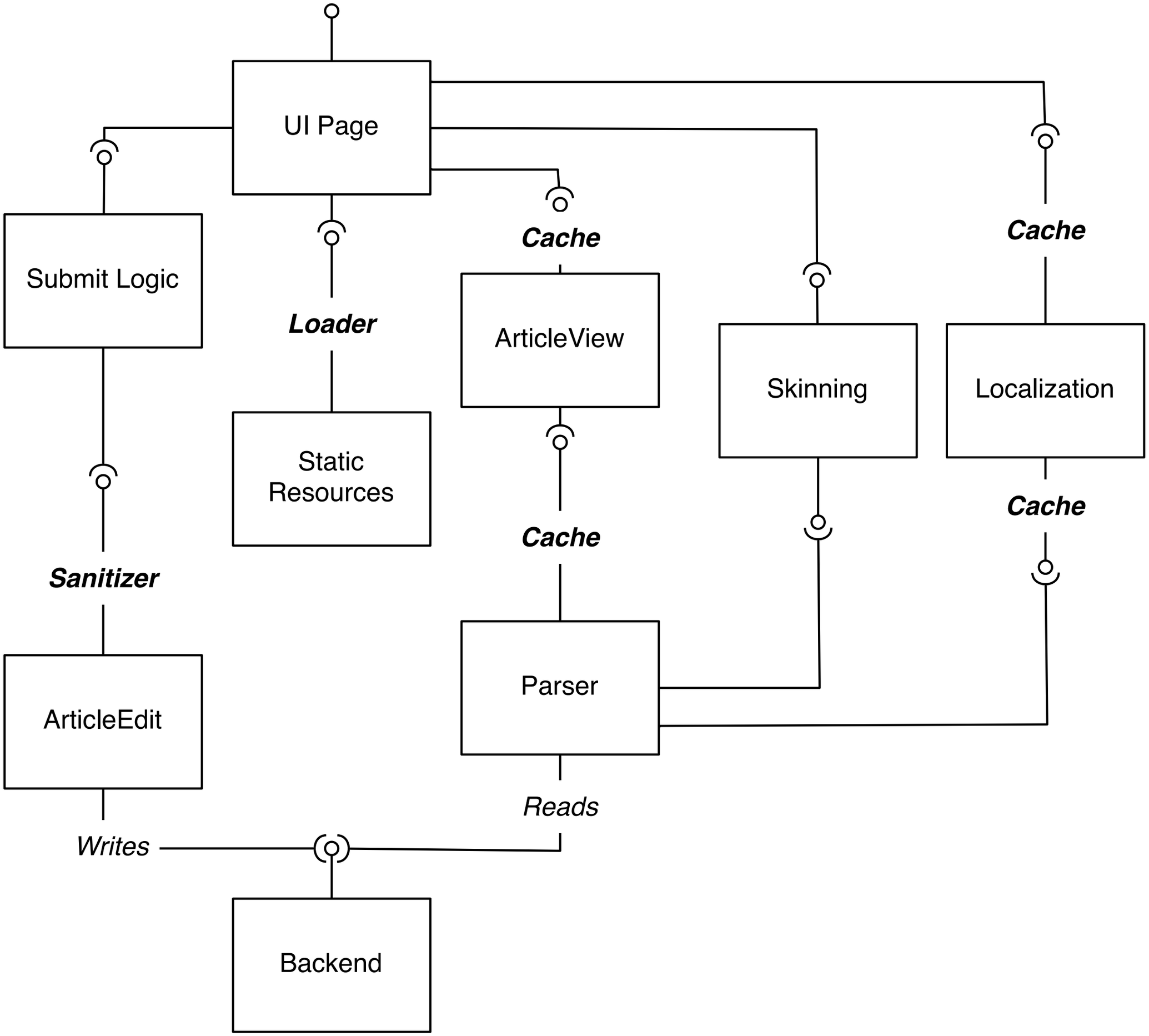
- Prevent Attacks
  - *Challenge Tokens (CSRF)*
  - *Validation (User) and Sanitization (SQL Injection, XSS)*

- Resist Attacks
  - *Maintain multiple copies of computations*
  - *Maintain multiple copies of data*

- Recover from Attacks
  - *DB Versioning (Recovery from data loss)*

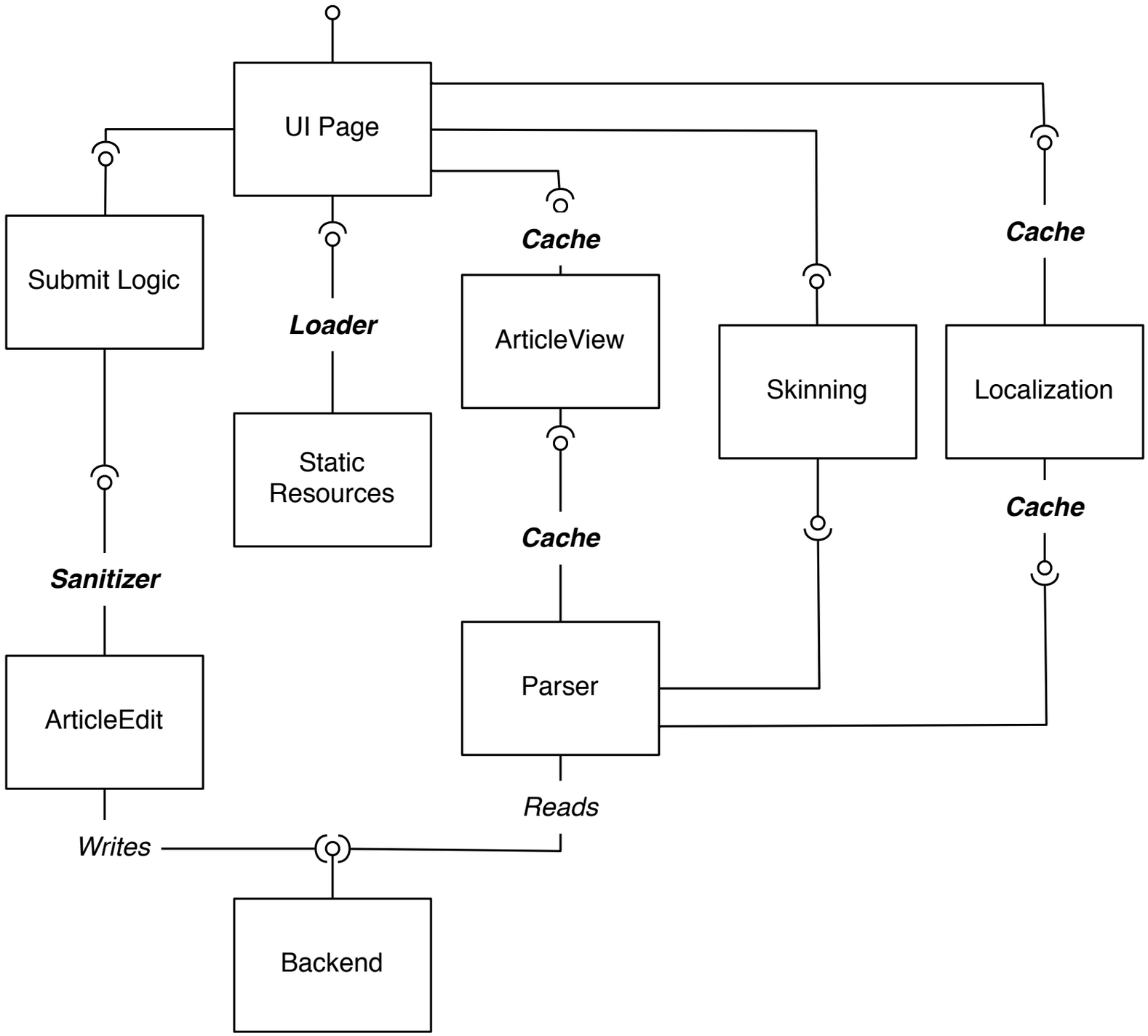

UI Page
Submit Logic
Loader
Cache
ArticleView
Skinning
Cache
Localization
Static Resources
Cache
Cache
ArticleEdit
Parser
Writes
Reads
Backend


UI Page
Submit Logic
Loader
Cache
Cache
Sanitizer Pipeline
Static Resources
ArticleView
Skinning
Localization
Cache
Cache
ArticleEdit
Parser
Reads
Writes
User Access
Backend

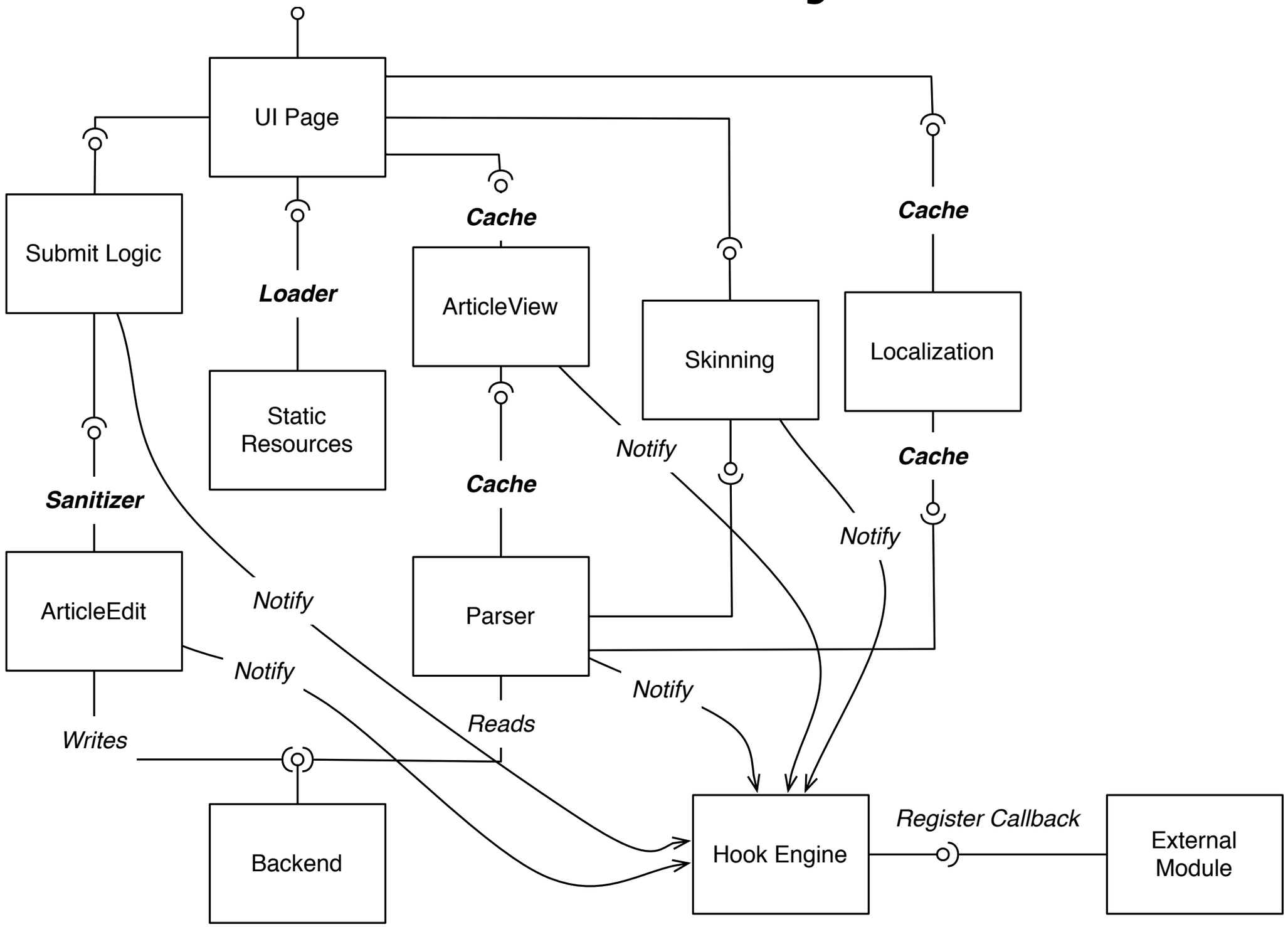

UI Page
Submit Logic
Loader
Cache
Cache
Sanitizer
Static Resources
ArticleView
Skinning
Localization
Cache
Cache
ArticleEdit
Parser
Reads
Writes
Backend

Additional Qualities

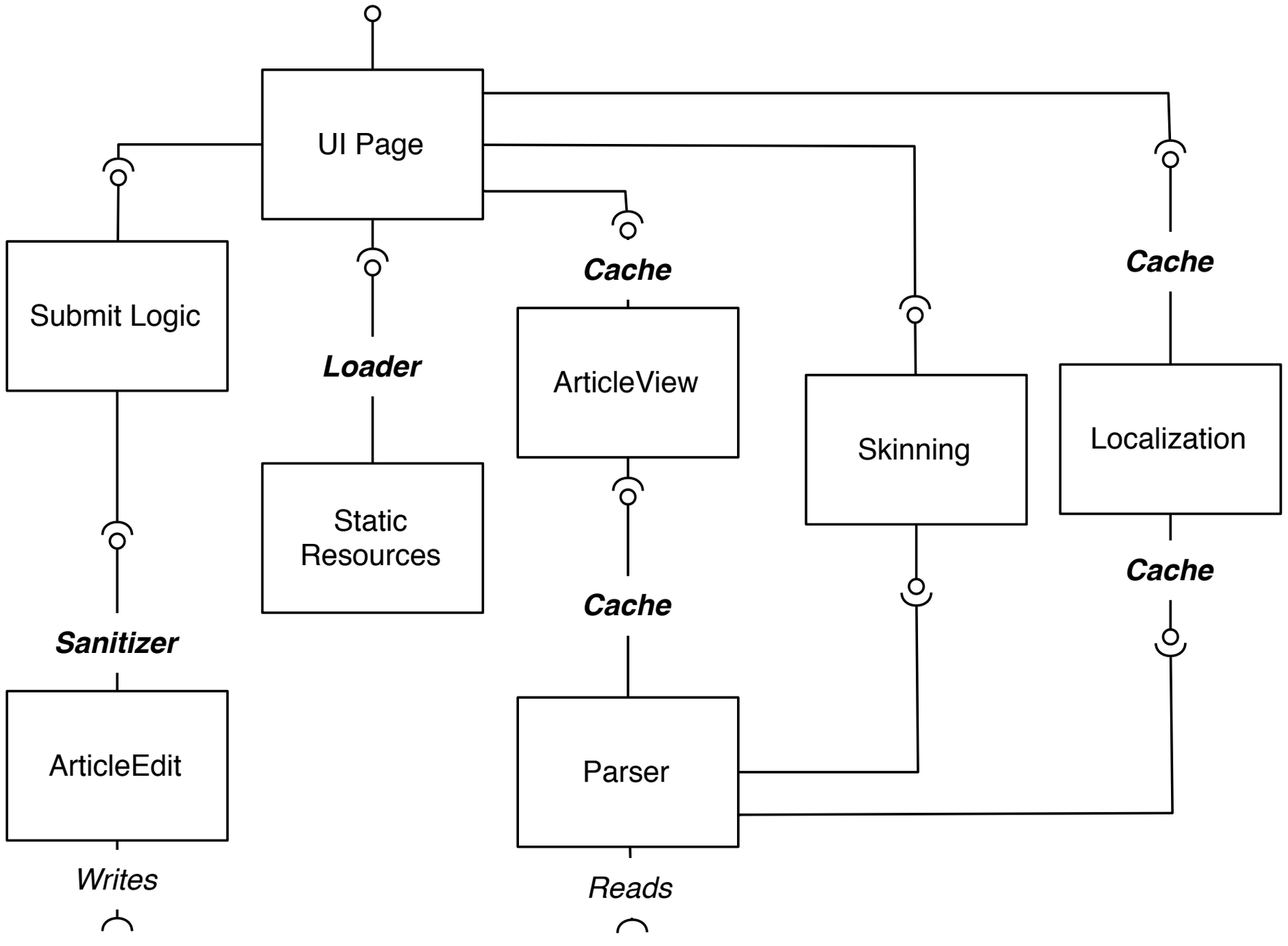
Extensibility



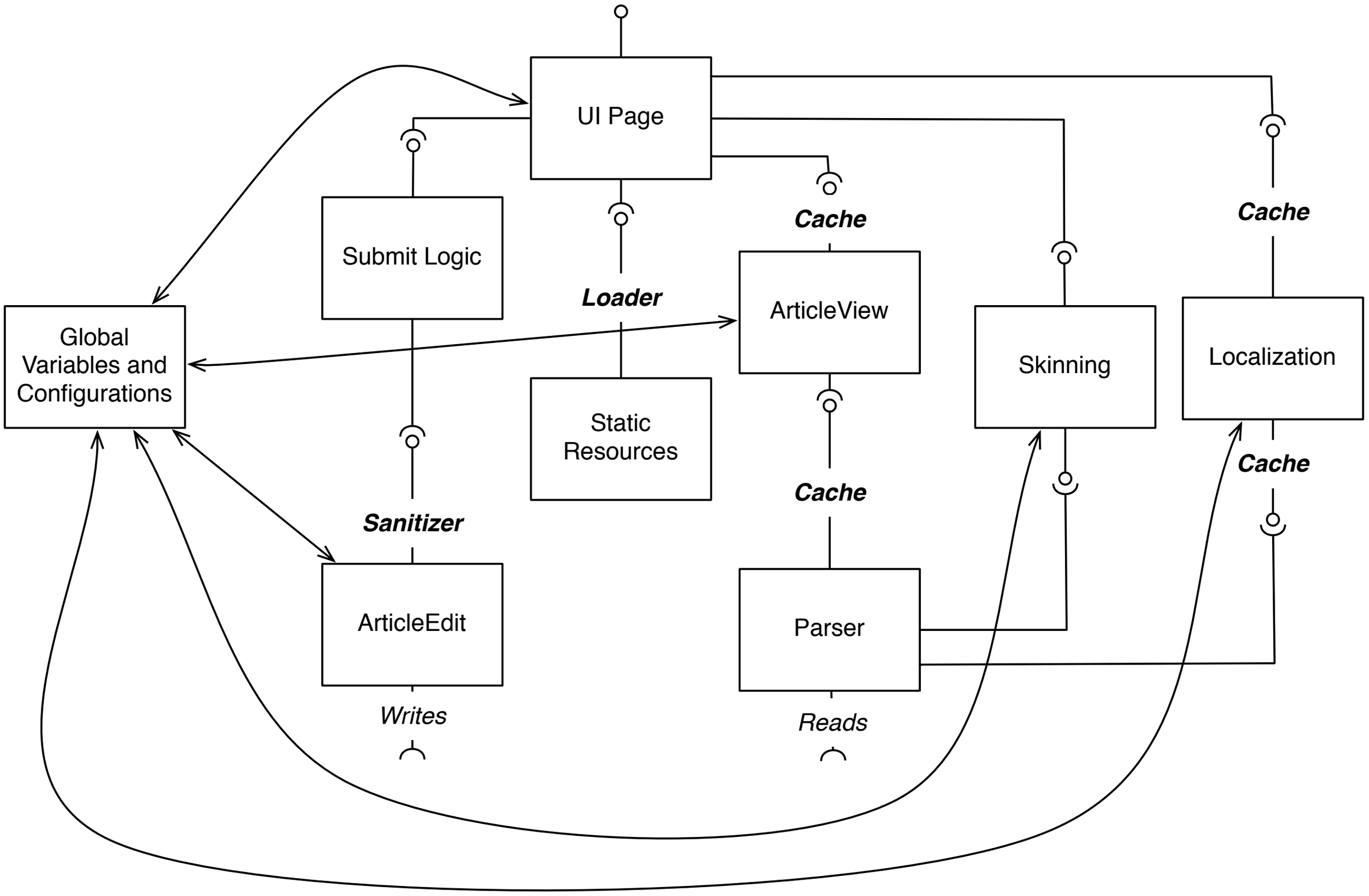
Extensibility



Configurability/Customizability



Configurability/Customizability

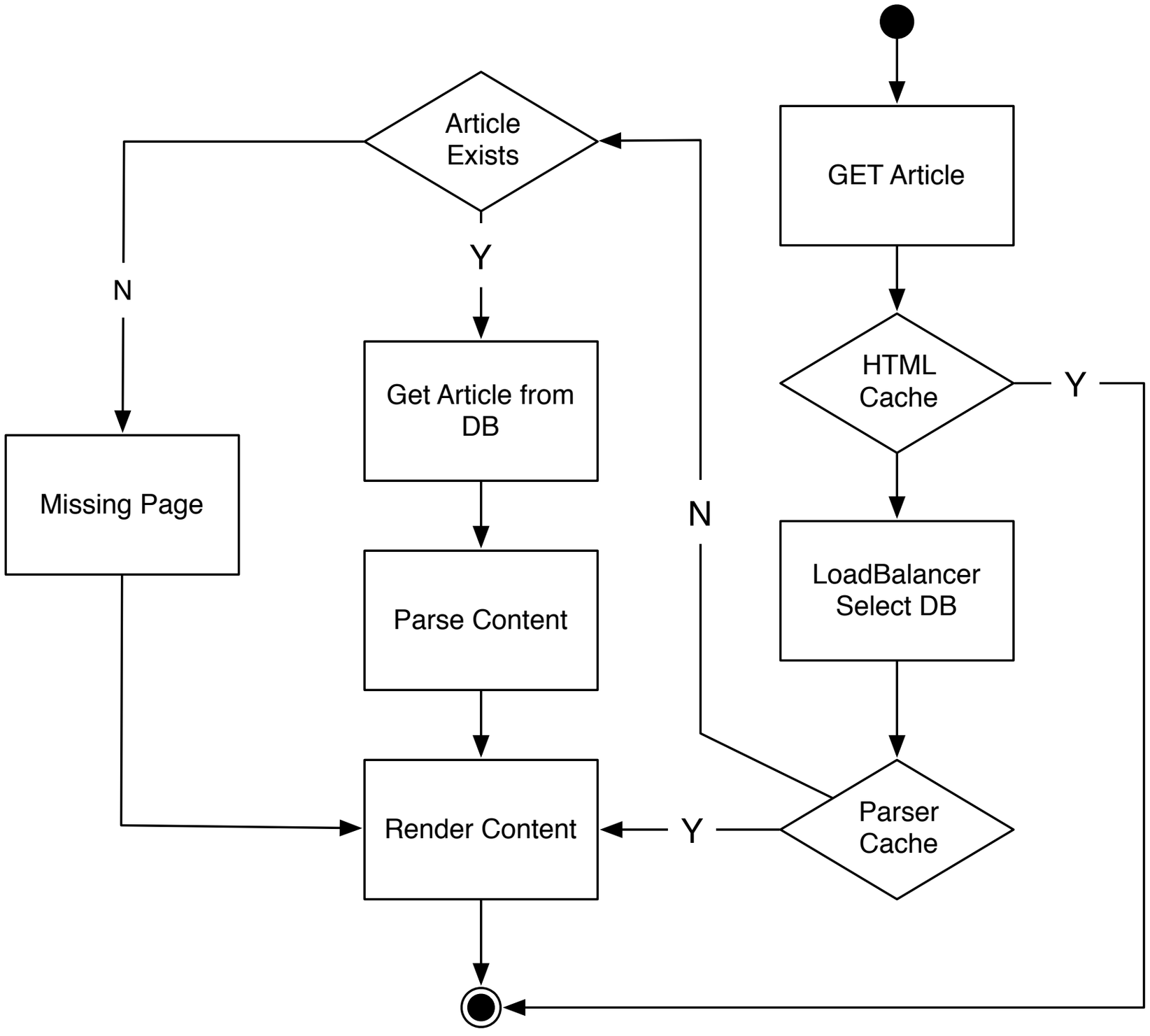


Process View

Read Article

A **user** requests an article during normal operation and gets the rendered article HTML page.

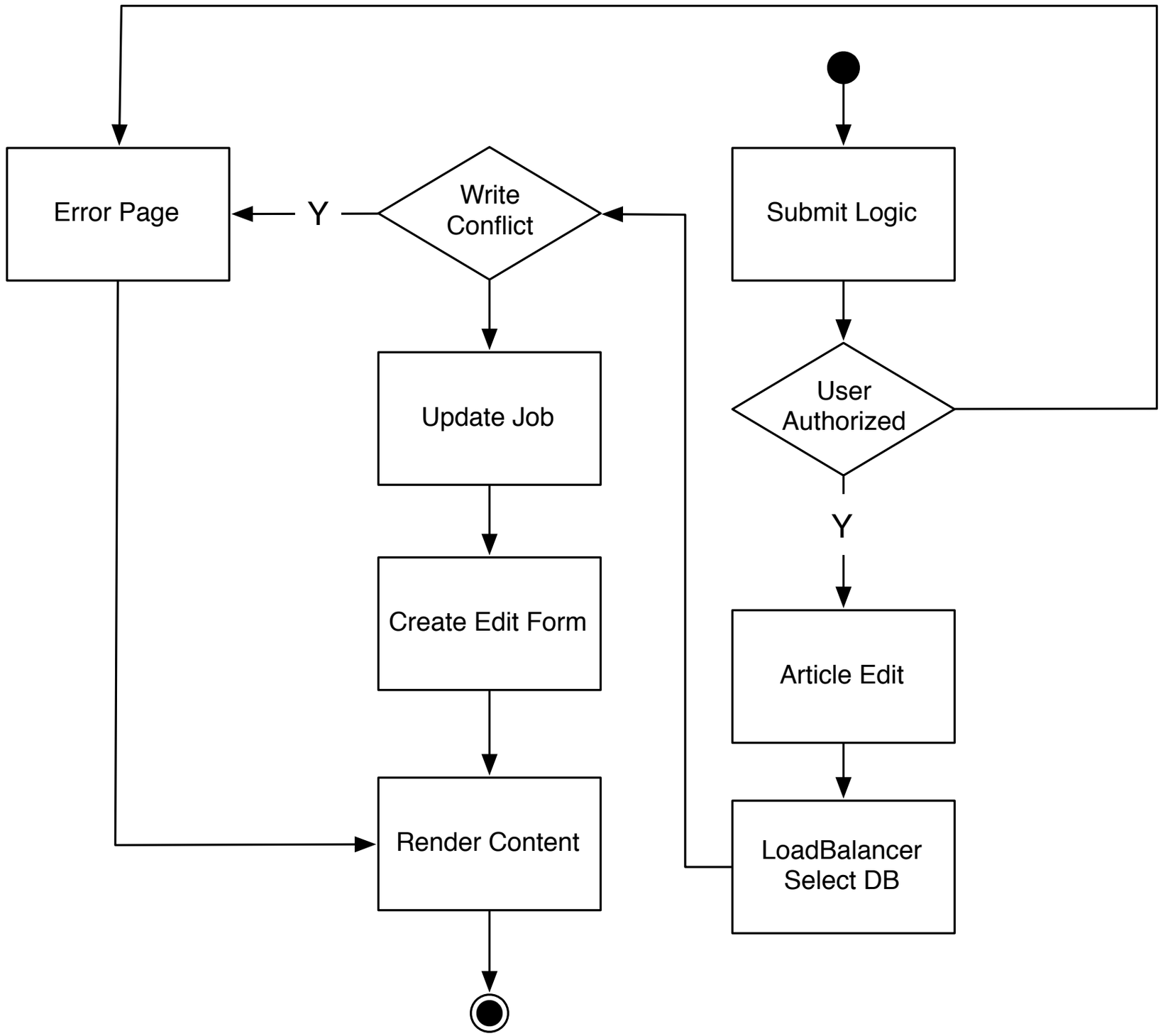
Read Article



Write/Edit Article

An **editor** saves an edited article during normal operation and the article is saved.

Write/Edit Article



Summary

- Work incrementally
- Use different architectural views
- Start the design from the domain model and go up in the layers
- Use frameworks whenever possible
- Each design decision has a rationale (hoisting)