COMP30670
Software Engineering (Conversion)

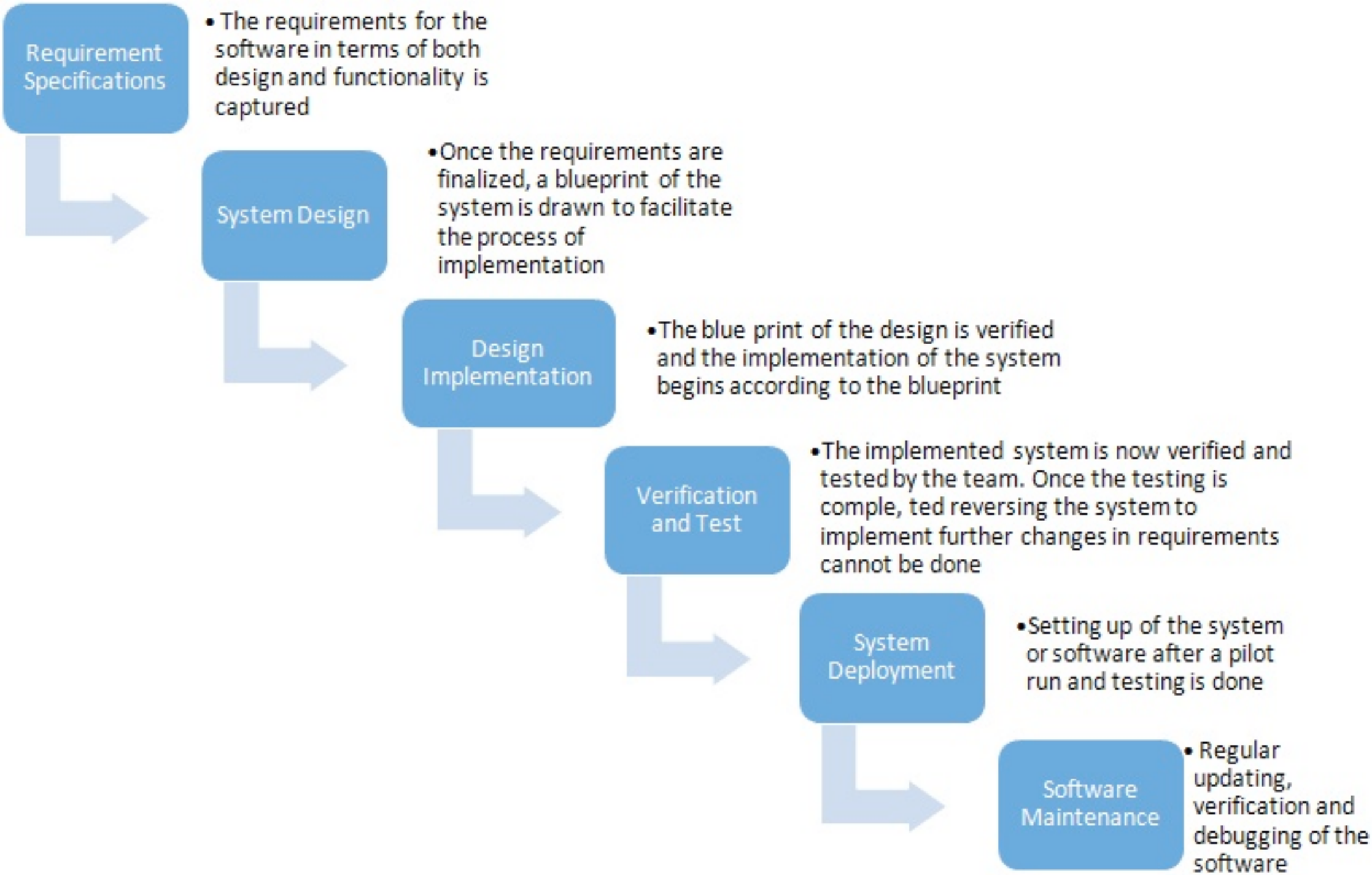
17 - Software Architecture
03/04/2017

Dr. Aonghus Lawlor

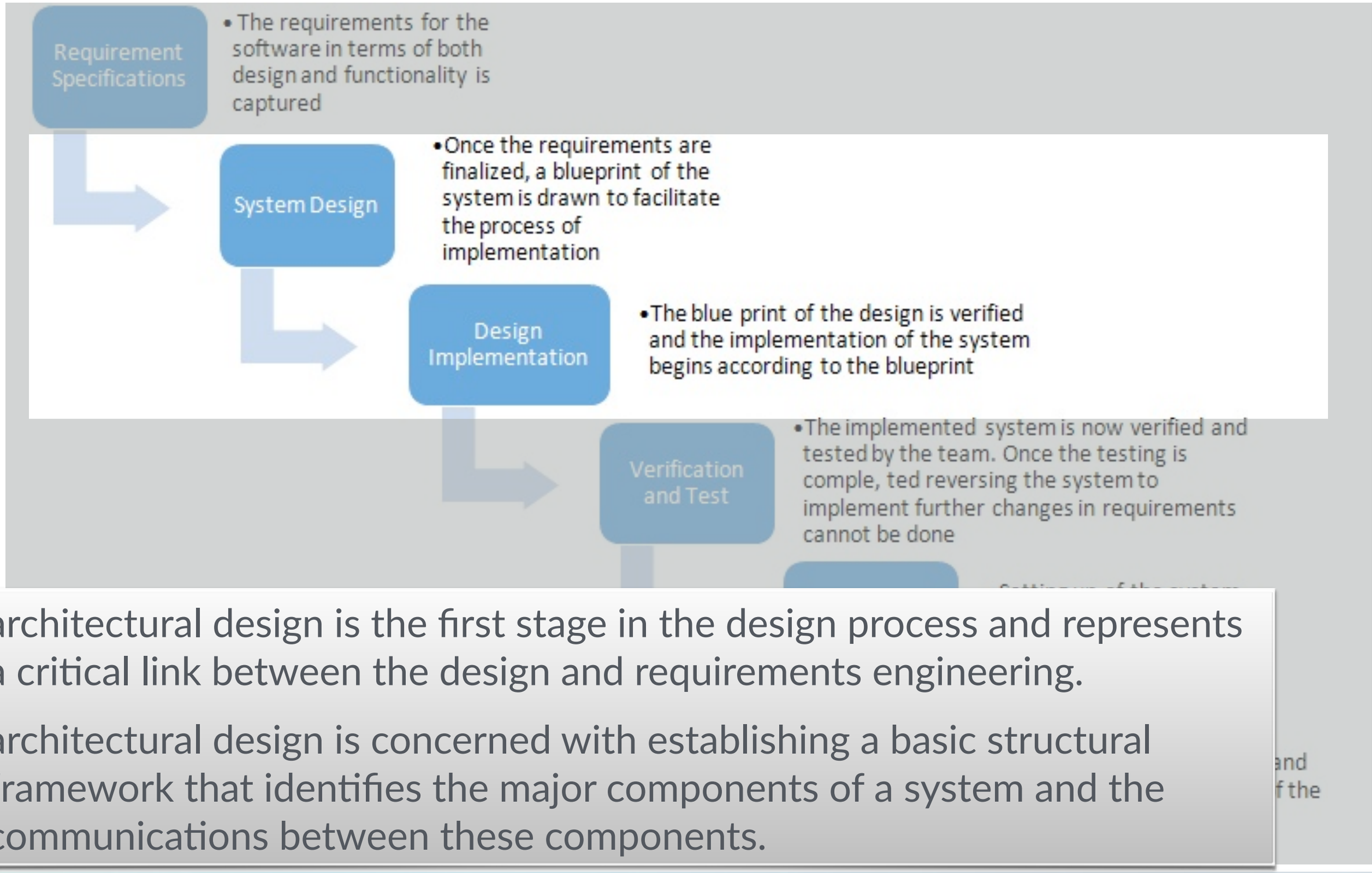
aonghus.lawlor@insight-centre.org

Software Architecture

Software Lifecycle



Design



- architectural design is the first stage in the design process and represents a critical link between the design and requirements engineering.
- architectural design is concerned with establishing a basic structural framework that identifies the major components of a system and the communications between these components.

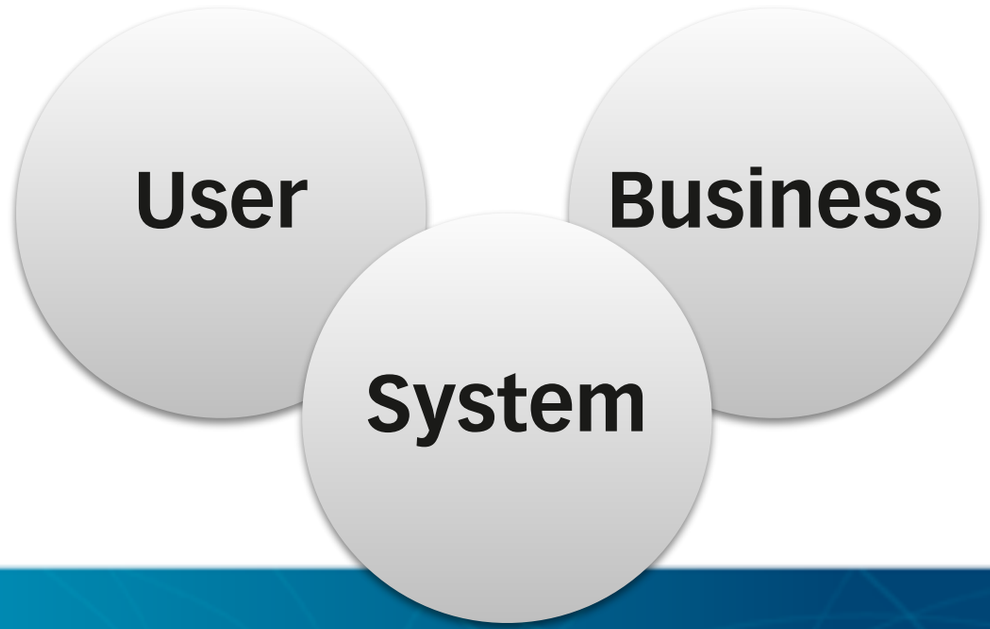
Software Architecture - Definition

"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them and properties of both."

(Bass, Clements, Kazman 2013)

Why is Architecture important?

- to avoid failures important to design for common problems and understand long-term consequences of key decisions
- The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.
- Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals.



Architecture/Design

- balance must often be found between competing requirements in different areas (eg. overall user experience can depend on the business or IT infrastructure and changes in one or the other can significantly affect the user experience).
- performance might be a major user and business goal, but should be balanced with what is achievable
- Architecture focuses on how the major elements and components within an application are used by, or interact with, other major elements and components within the application
- selection of data structures and algorithms or the implementation details of individual components are design concerns.
- Architecture and design concerns very often overlap

Architecture

- **high level architecture should:**
  - Expose the structure of the system but hide the implementation details.
  - Realise all of the use cases and scenarios.
  - Try to address the requirements of various stakeholders.
  - Handle both functional and quality requirements.

Architecture

- Build to change instead of building to last. Think about how the application may need to change over time to address new requirements and challenges, and incorporate the flexibility to support this
- model to analyse and reduce risk. Use design tools, modelling systems such as Unified Modelling Language (UML), and visualisations to help capture requirements and architectural and design decisions, and to analyse their impact.
- keep the model informal so that it does not interfere with the ability to iterate and adapt the design easily.
- Use models and visualisations as a communication and collaboration tool. Efficient communication of the design, the decisions you make, and ongoing changes to the design, is critical to good architecture. Good modelling enables rapid communication and the design is easier to change.
- Identify key engineering decisions and try to get them right the first time so that the design is more flexible and less likely to be broken by changes

- **Advantages of Explicit Architecture:**

- **Stakeholder communication**
  - Architecture may be used as a focus of discussion by system stakeholders
- **System analysis**
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- **Large-scale reuse**
  - The architecture may be reusable across a range of systems.

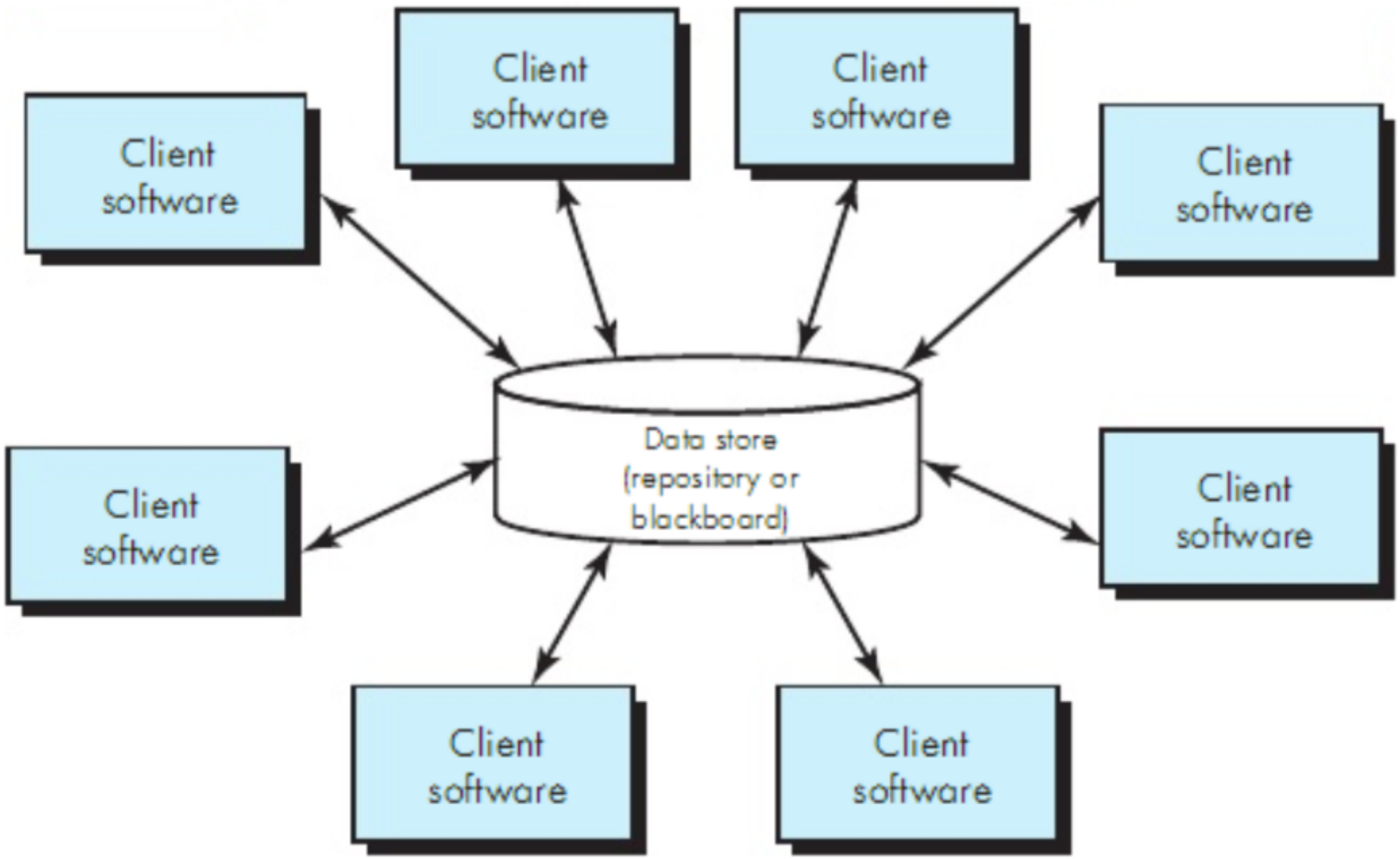
Architecture Design Decisions

- How will the users be using the application?
- How will the application be deployed into production and managed?
- What are the quality attribute requirements for the application, such as security, performance, concurrency, internationalisation, and configuration?
- How can the application be designed to be flexible and maintainable over time?
- What are the architectural trends that might impact your application now or after it has been deployed?

Architectural Styles

- The architectural model of a system may conform to a generic architectural model or style.
- An awareness of these styles can simplify the problem of defining system architectures.
- However, most large systems are heterogeneous and do not follow a single architectural style.

Data Centred Architecture



Data Centred Architecture

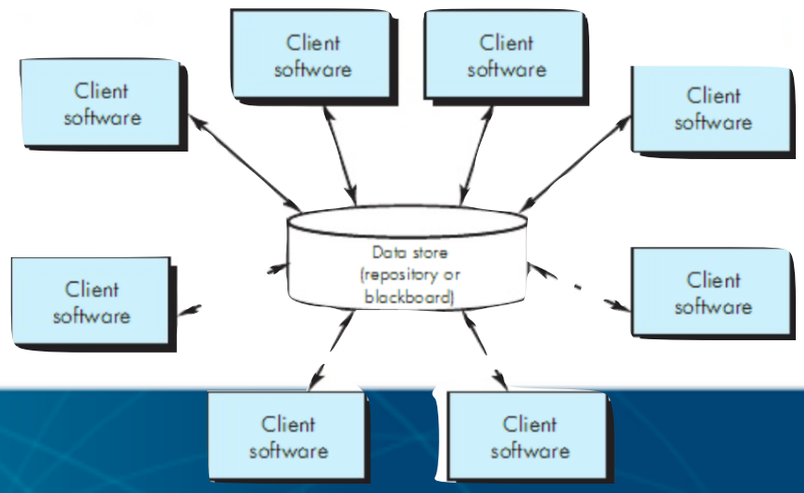
a data store (eg. file/DB) is at the centre of this architecture and is accessed frequently by other components that update, add, delete or modify data in the store

client software accesses a central repository

sometimes the data store is passive (client accesses the data independently of any changes to the data or other clients)

sometimes the data store is active (the repository is more like a “blackboard” that sends notifications to the client software when data of interest to the client changes)

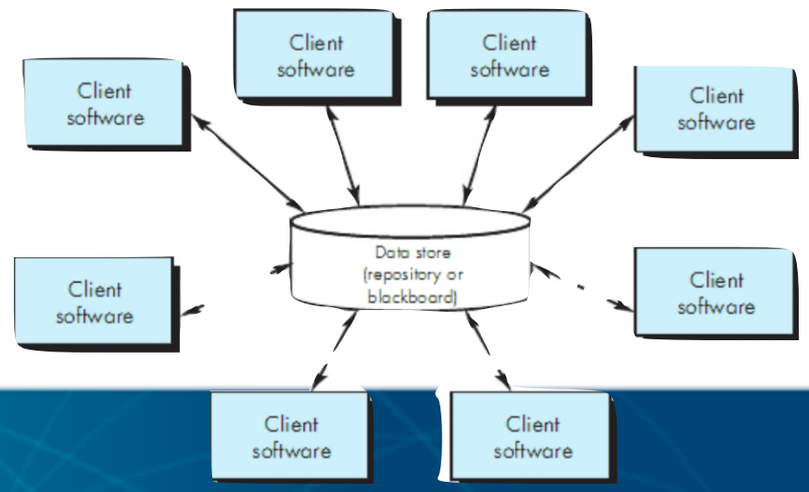
integrability: existing components are easy to change and new client components can be added



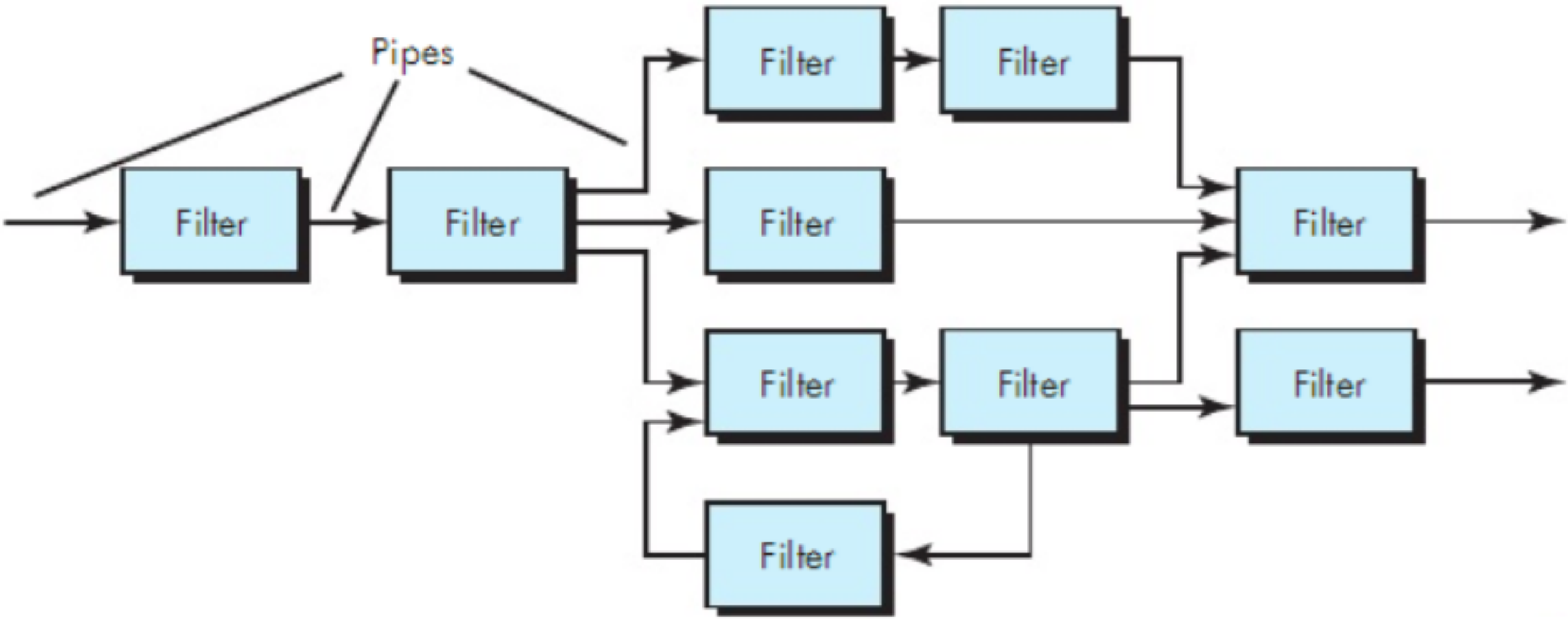
Data Centred Architecture

A common example of the blackboard pattern is in speech recognition

separate threads can process different parts of the sound sample, updating the blackboard with words that have been recognised. Then another process can pick up these words and perform grammar and sentence formation. Meanwhile more words and meanings are coming in, and eventually even higher level processes can pick up the formed sentences and various alternative guesses and begin to formulate it's meaning, then further intelligence systems can start to choose the most appropriate answer. All these systems have access to the blackboard and work together through it's central platform.



Data Flow architectures



Pipes and filters

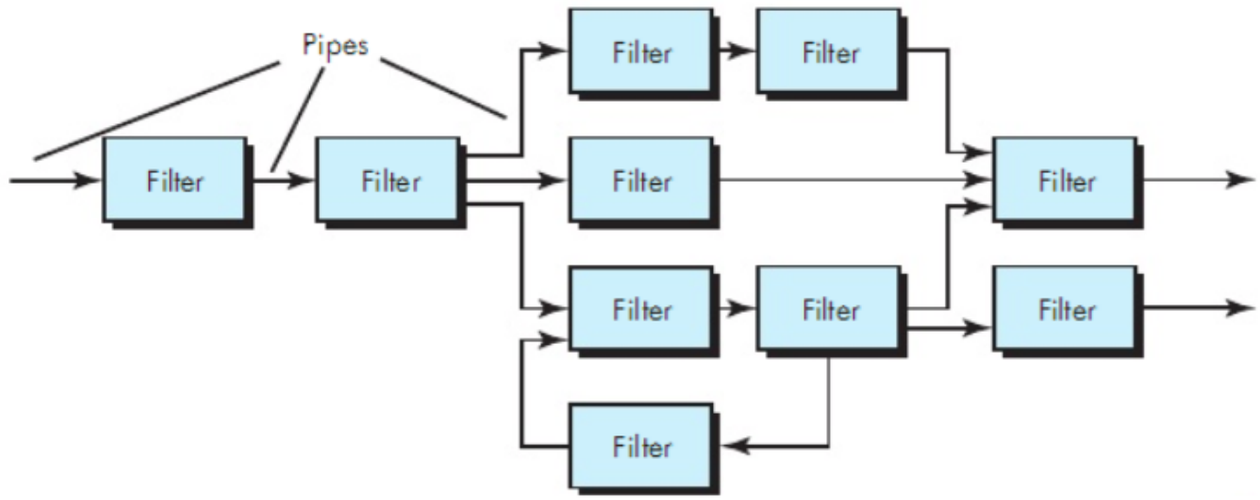
Data Flow architectures

input data is transformed by a series of components into the output data

the “pipe and filter” pattern has a set of components (filters) connected by “pipes” that transmit data from one component to the next

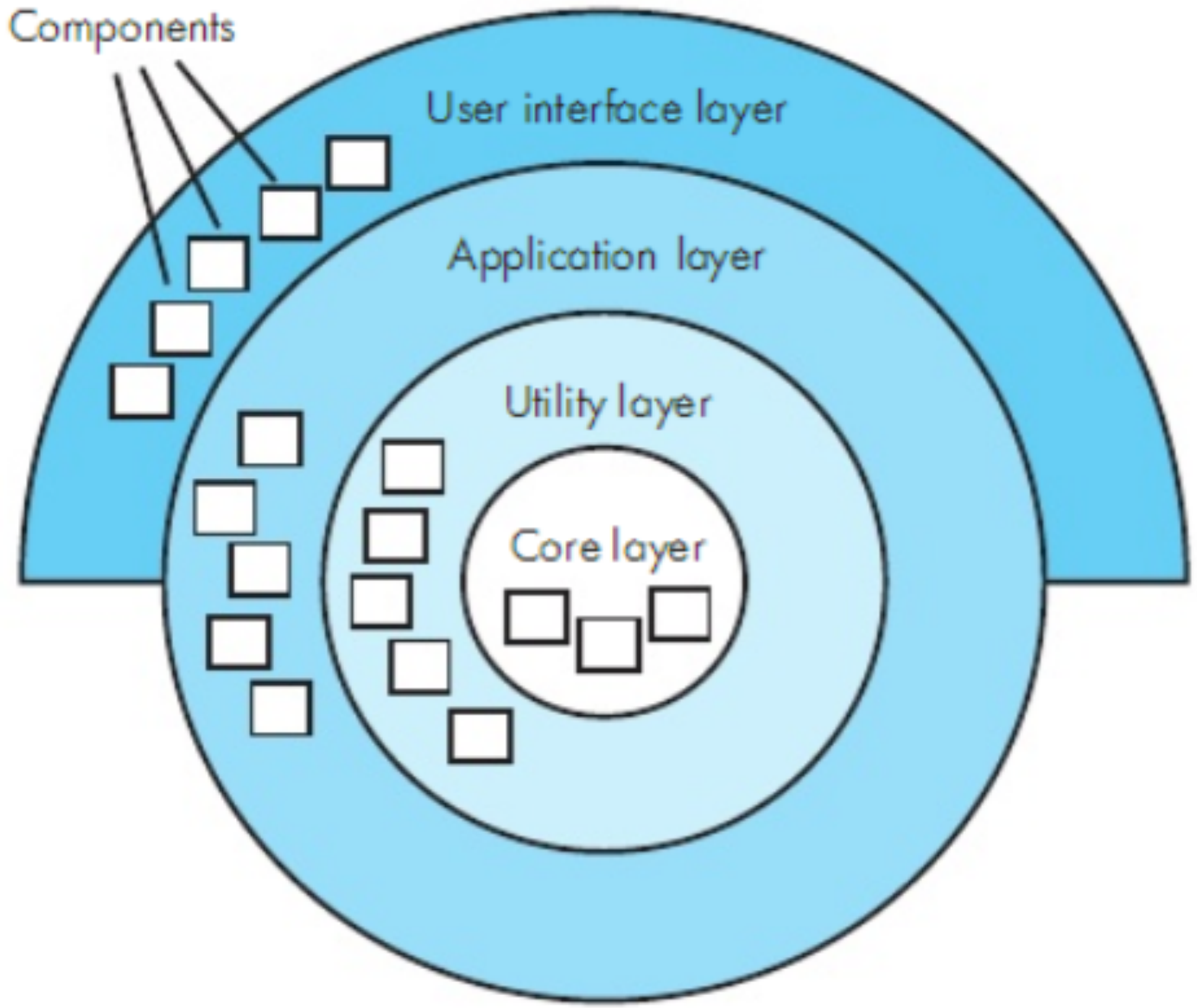
each filter works independently of the upstream components

expects data in a certain form and produces output to the next in a certain form



Pipes and filters

layered architecture

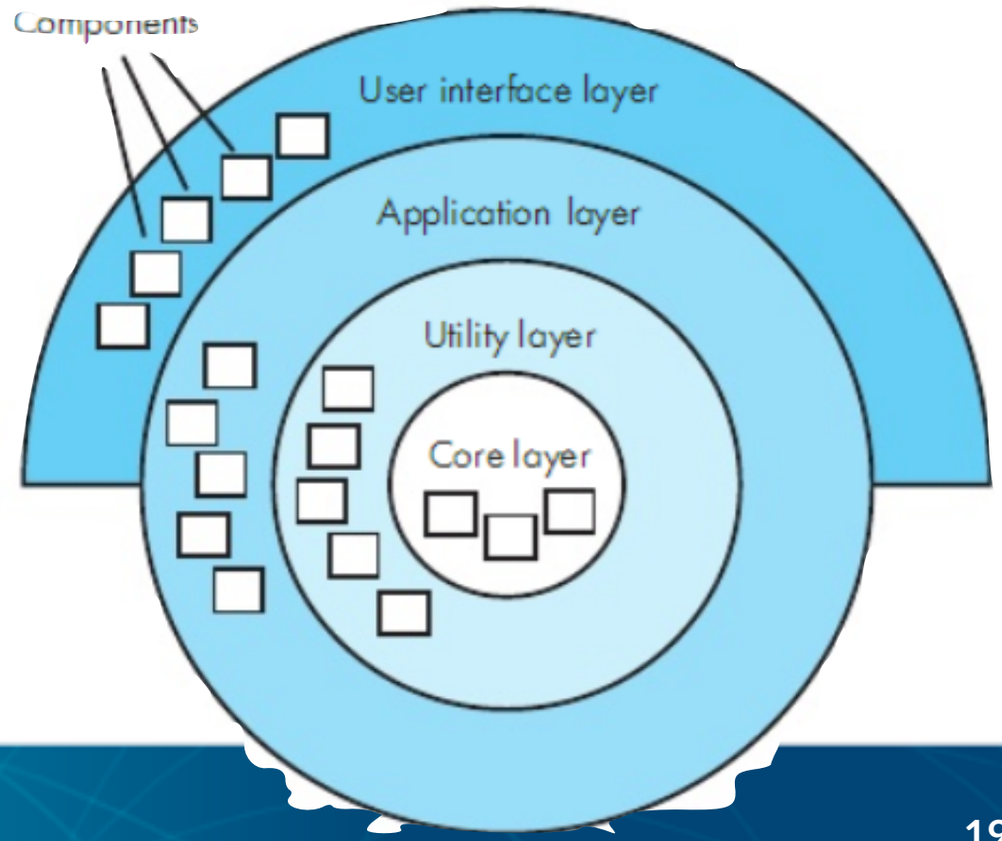


layered architecture

a number of different layers are defined each accomplishing operations that progressively become closer to the machine instructions

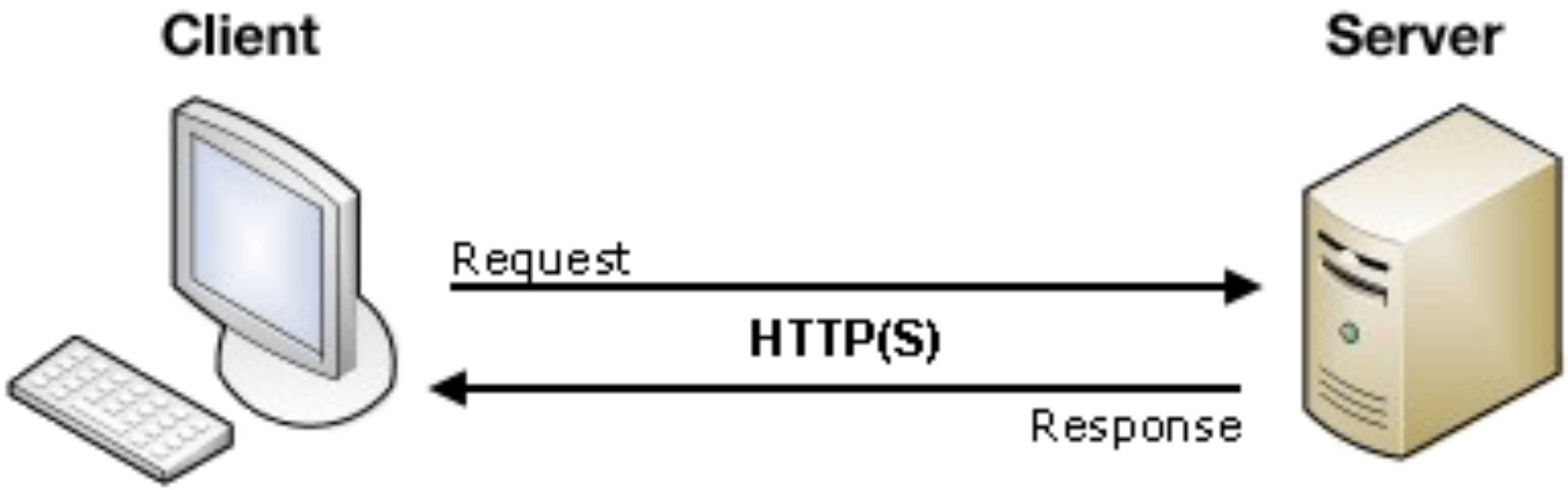
at the inner layer components interface with the operating system

intermediate layers provide utility services and application software functions



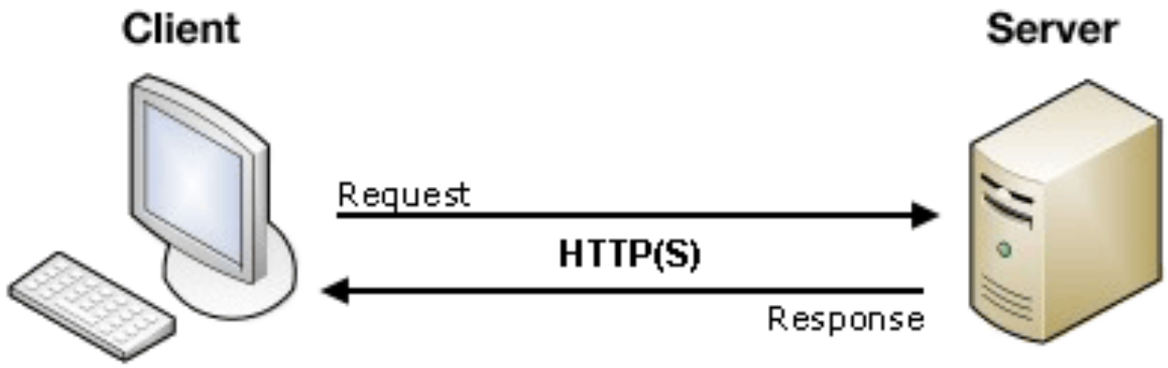
Client/Server Architecture

- The client/server architectural style describes distributed systems that involve a separate client and server system, and a connecting network.
- The simplest form of client/server system involves a server application that is accessed directly by multiple clients



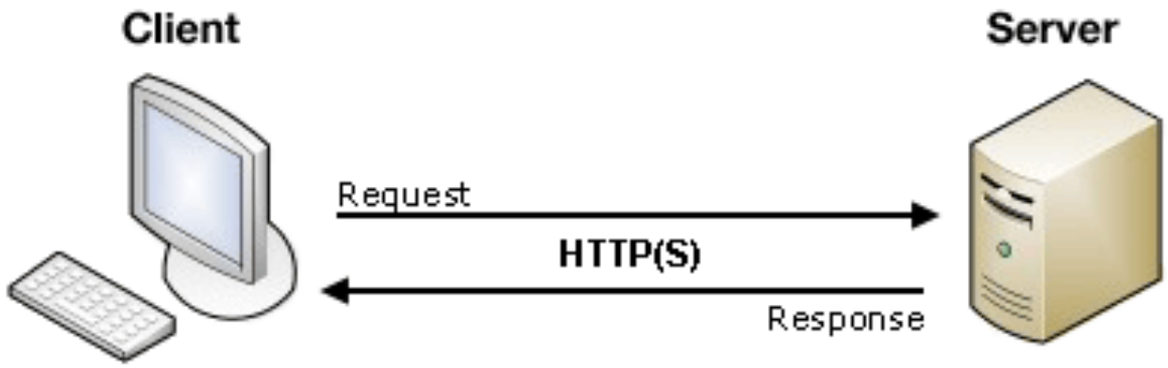
Client/Server Architecture

- generally, the client/server architectural style describes the relationship between a client and one or more servers, where the client initiates one or more requests, waits for replies, and processes the replies on receipt.
- The server may send responses using a range of protocols and data formats to communicate information to the client.
- some examples of the client/server architectural style include web browsers, e-mail readers, FTP clients, and database query tools, tools and utilities that manipulate remote systems (such as system management tools and network monitoring tools).



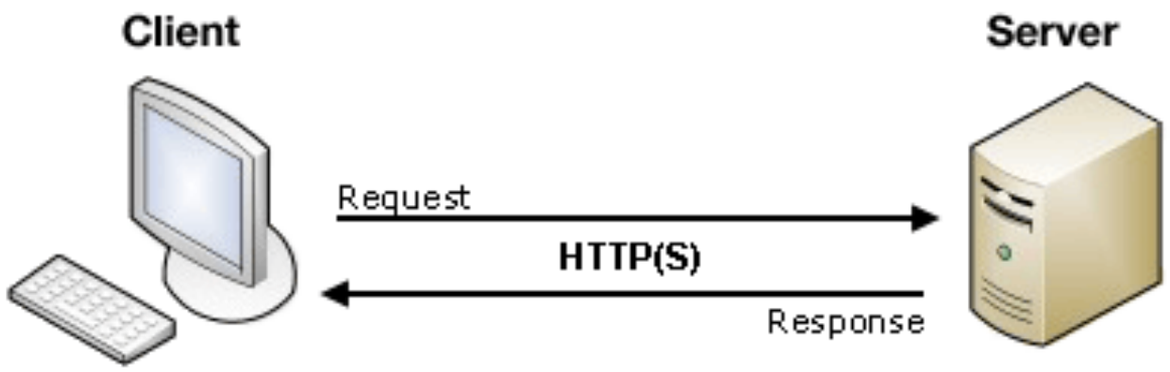
Client/Server Architecture - variations

- Client-Queue-Client systems. This approach allows clients to communicate with other clients through a server-based queue. Clients can read data from and send data to a server that acts simply as a queue to store the data. This allows clients to distribute and synchronise files and information.
- eg. web crawlers



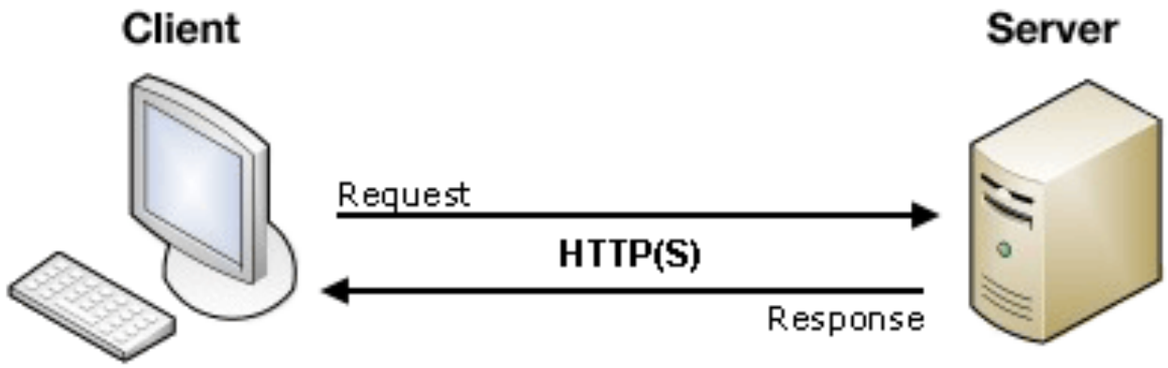
Client/Server Architecture - variations

- Peer-to-Peer (P2P) applications. Developed from the Client-Queue-Client style, the P2P style allows the client and server to swap their roles in order to distribute and synchronise files and information across multiple clients. It extends the client/server style through multiple responses to requests, shared data, resource discovery, and resilience to removal of peers.
- e.g.. bittorrent



Client/Server Architecture - variations

- Application servers. A specialised architectural style where the server hosts and executes applications and services that a thin client accesses through a browser or specialised client installed software.
- eg: is a client executing an application that runs on the server through a framework such as Terminal Services or Java application servers, where the server behaves like an extended java VM



Agile Architecture

- **The key objectives for the Agile architect are:**
  - Deliver working solutions
  - Maximise stakeholder value
  - Find solutions which meet the goals of all stakeholders
  - Enable the next effort
  - Manage change and complexity

Agile Architecture

- the architect must enable further effort, whether that means documenting the architecture such that a successor can continue to maintain it or designing an architecture extensible and responsive to requirements imposed by interacting systems and projects. The goal is that the architecture never becomes a burden or setback to further work.

- everything around the architecture - the stakeholders, the requirements, the business - will change. The cost of change in the architecture of any major system is significant, but the architect must balance the planning for change and the minimisation of this cost against other goals.

Agile Architecture

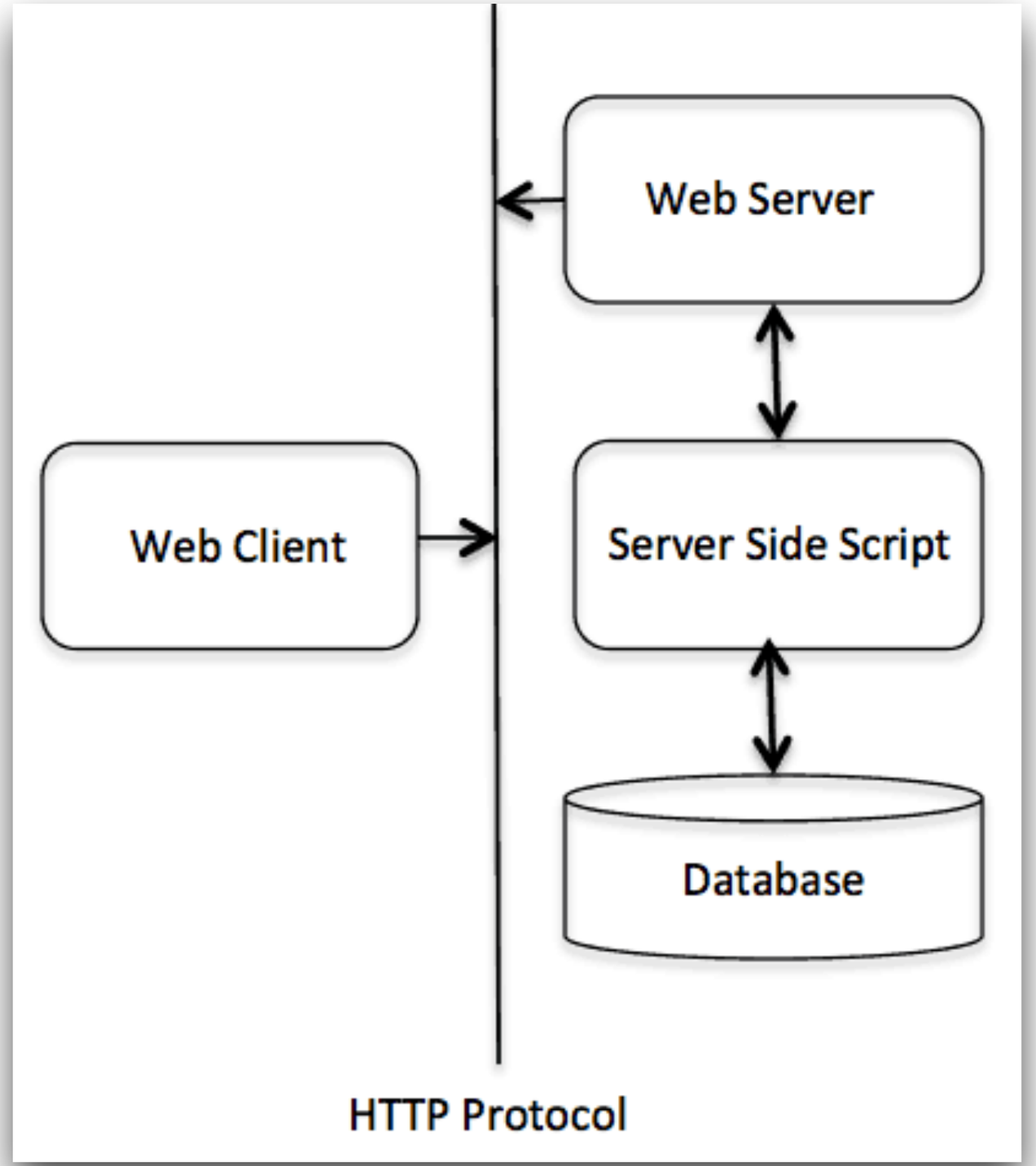
- up-front planning or big design is the first step. This means eliciting the relevant quality attributes, producing high-level diagrams, and making major hardware and software decisions. The only difference here is that rather than settling on a firm approach, several options are left on the table until the better options are made more evident by empirical knowledge gained throughout the development process.

- In this process, architectural work is performed in sprints.

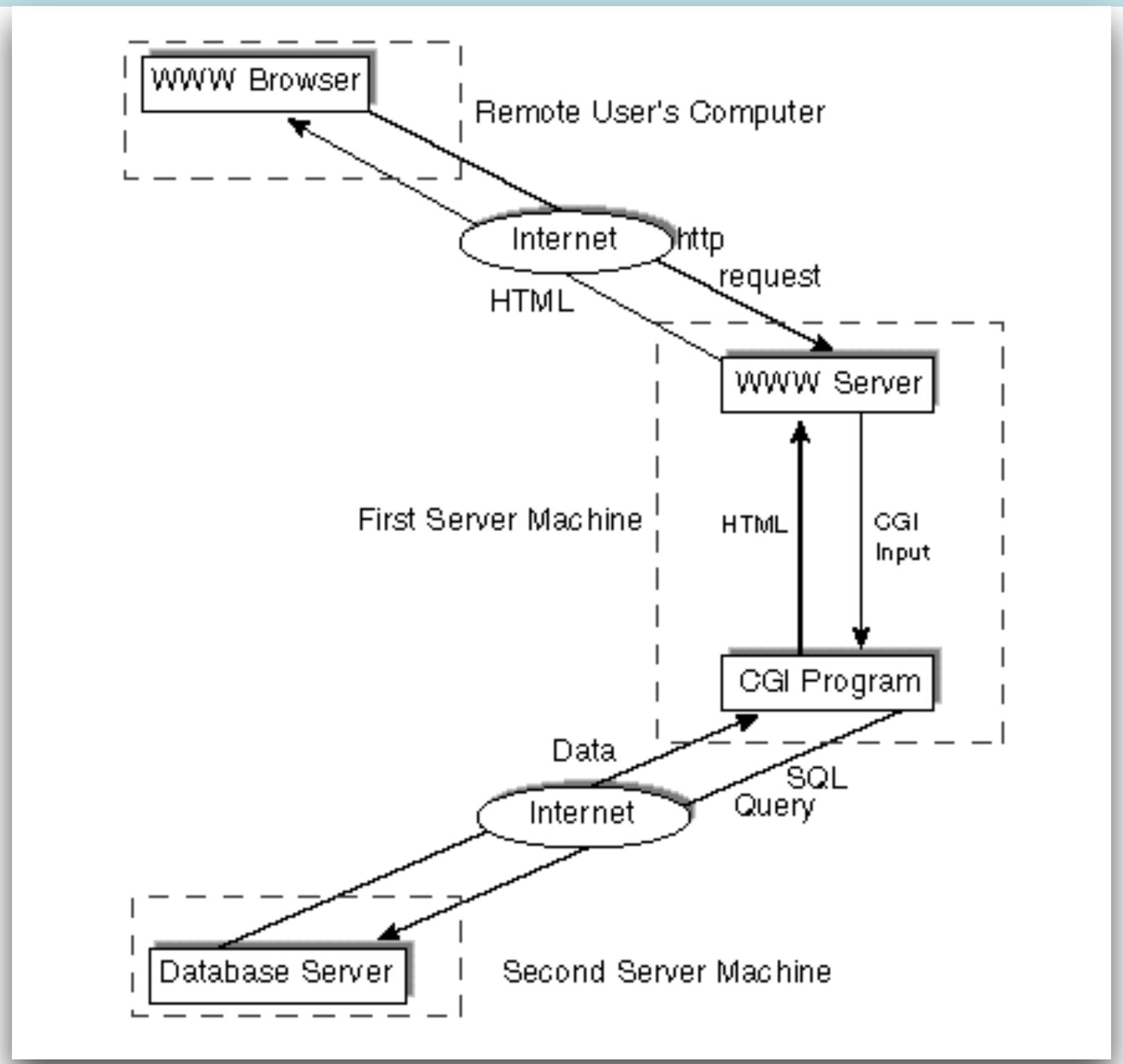
- an architectural backlog is kept separate from the main product backlog - this keeps both backlogs clear and focused, and makes it easy for the architect to retain ownership of architectural items.

Web Server Architecture

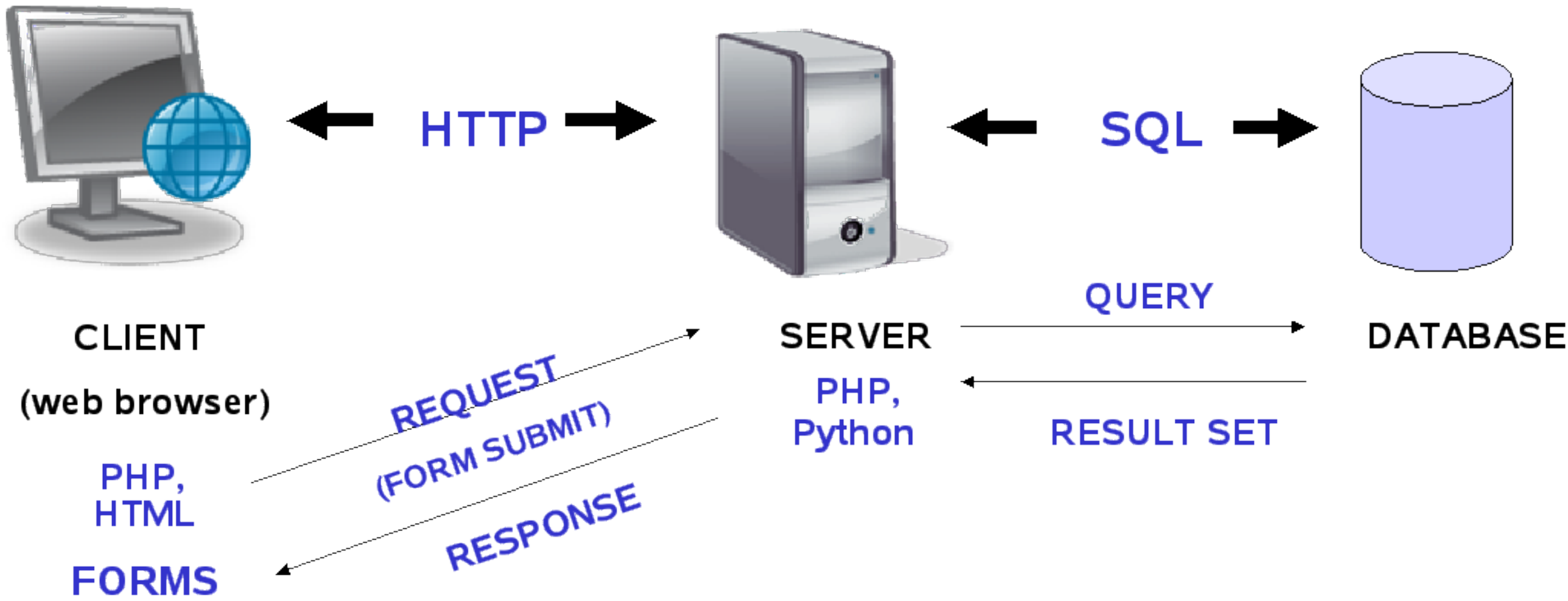
```
 GNU nano 2.2.2     File: /etc/apache2/sites-enable

<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
```



Web Architecture



Web Architecture (PHP)

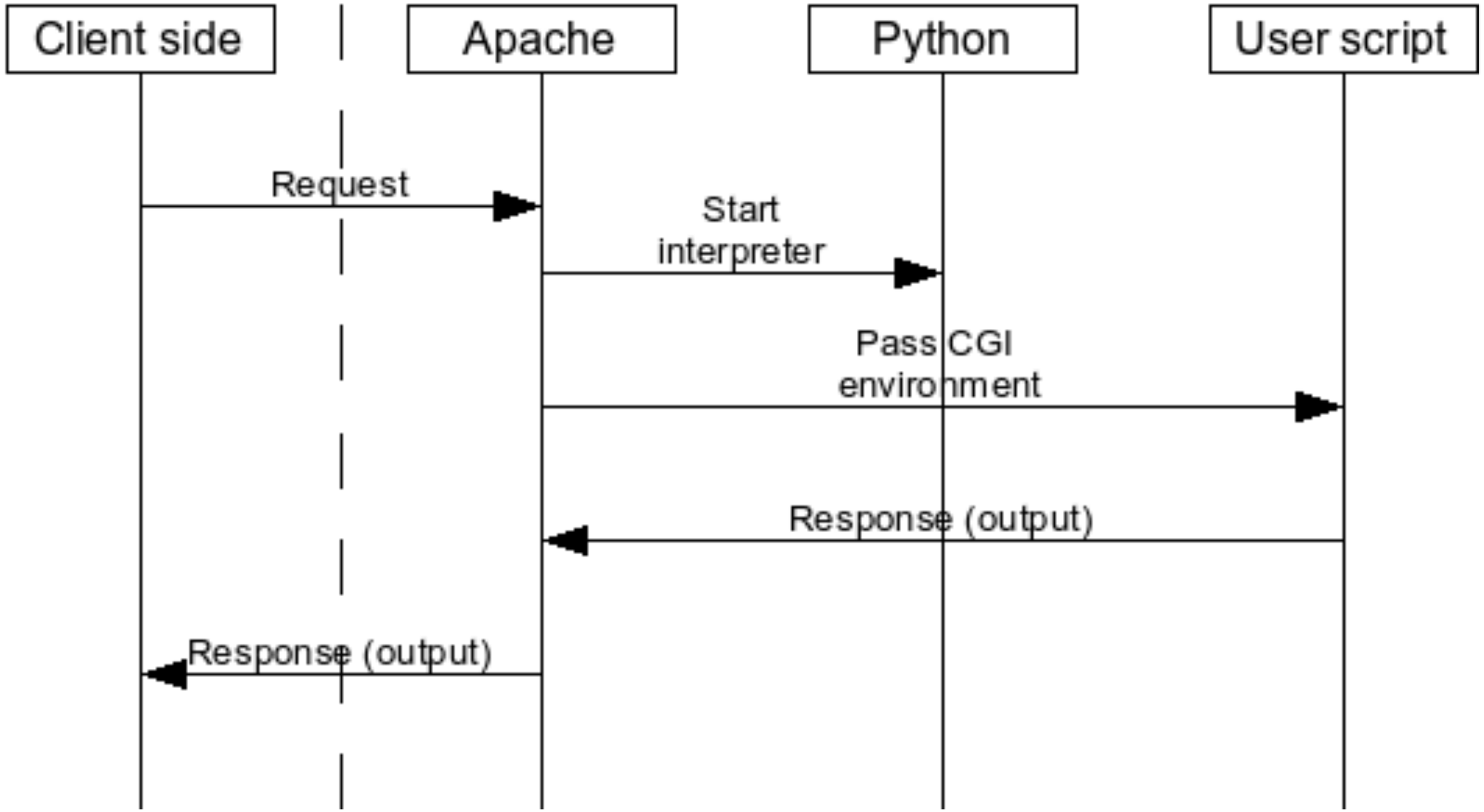


php example code from moodle

```
if ($path) {
    if ($fullpath) {
        $reportfile = make_temp_directory('phpcs') . '/phpcs_' . random_string(10) . '.xml';
        $cli = new local_codechecker_codesniffer_cli();
        $phpcs->setIgnorePatterns(local_codesniffer_get_ignores($exclude));
        $phpcs->process(local_codechecker_clean_path($fullpath),
                local_codechecker_clean_path($CFG->dirroot . '/local/codechecker/moodle'));
        // Save the xml report file to dataroot/temp.
        $phpcs->reporting->printReport('local_codechecker', false, $cli->getCommandLineValues(),
$reportfile);
        // Load the XML file to proceed with the rest of checks.
        $xml = simplexml_load_file($reportfile);
        // Look for other problems, not handled by codesniffer.
        local_codechecker_check_other_files(local_codechecker_clean_path($fullpath), $xml);
        list($numerrors, $numwarnings) = local_codechecker_count_problems($xml);
        // Output the results report.
        echo $output->report($xml, $numerrors, $numwarnings);
        // And clean the report temp file.
        @unlink($reportfile);
    } else {
        echo $output->invald_path_message($path);
    }
}
$mform->display();
echo $OUTPUT->footer();
```

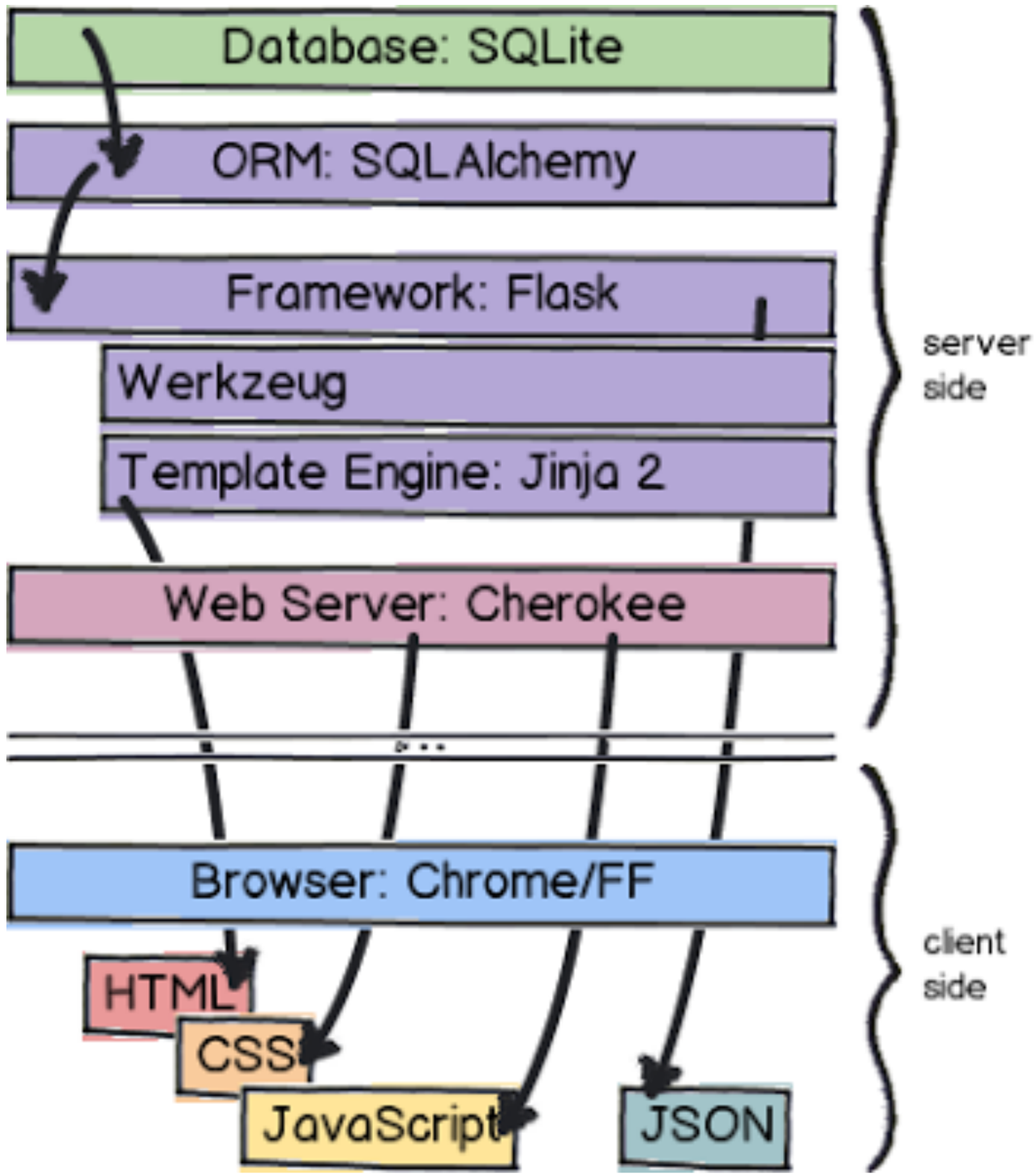
Web Server Architecture (Python)

CGI



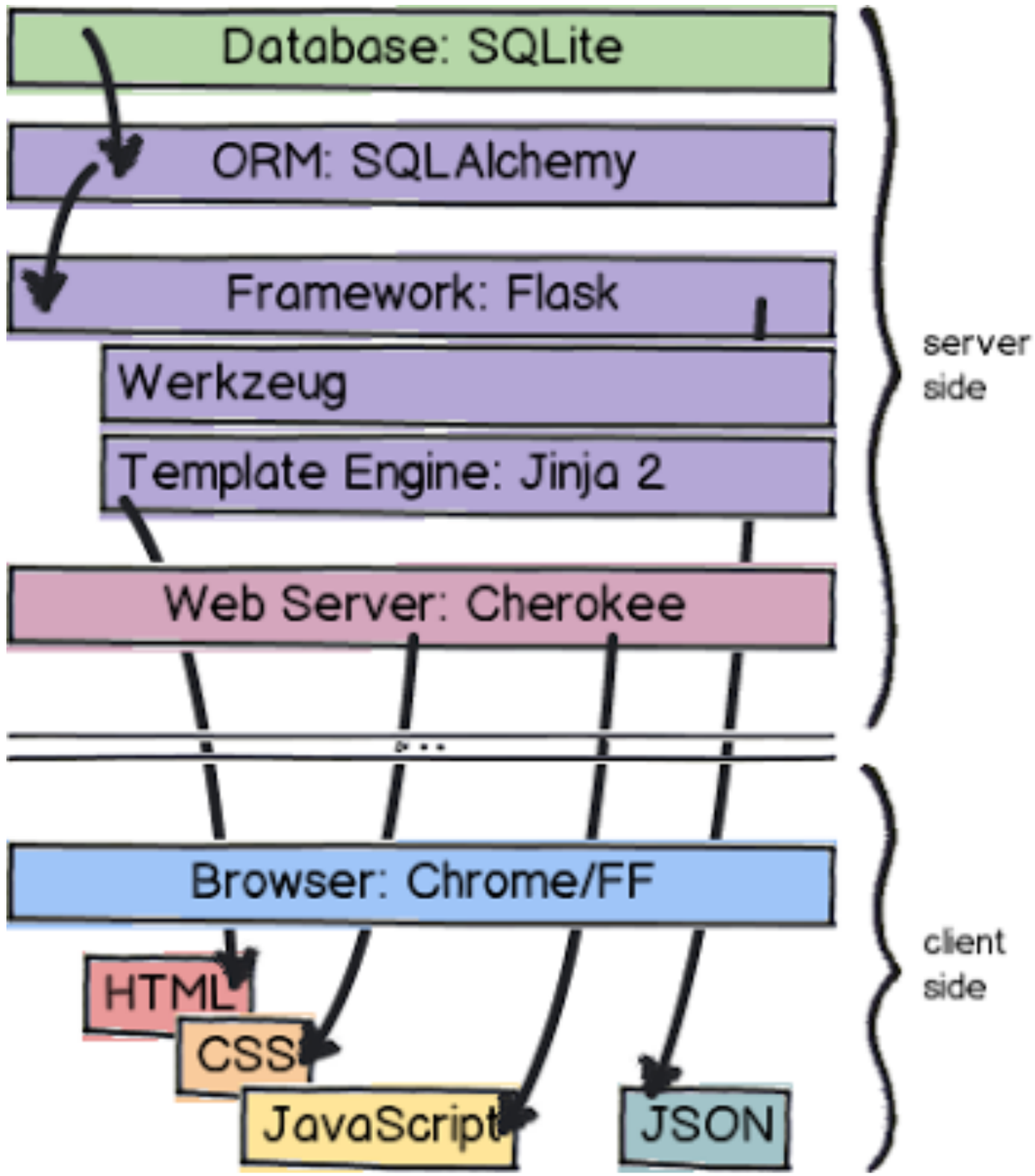
Web App Architecture

Database:
during the development stage you might use **SQLite**. It is a single file based system, light installation and maintenance efforts involved.
MySQL is a powerful server, which scales better



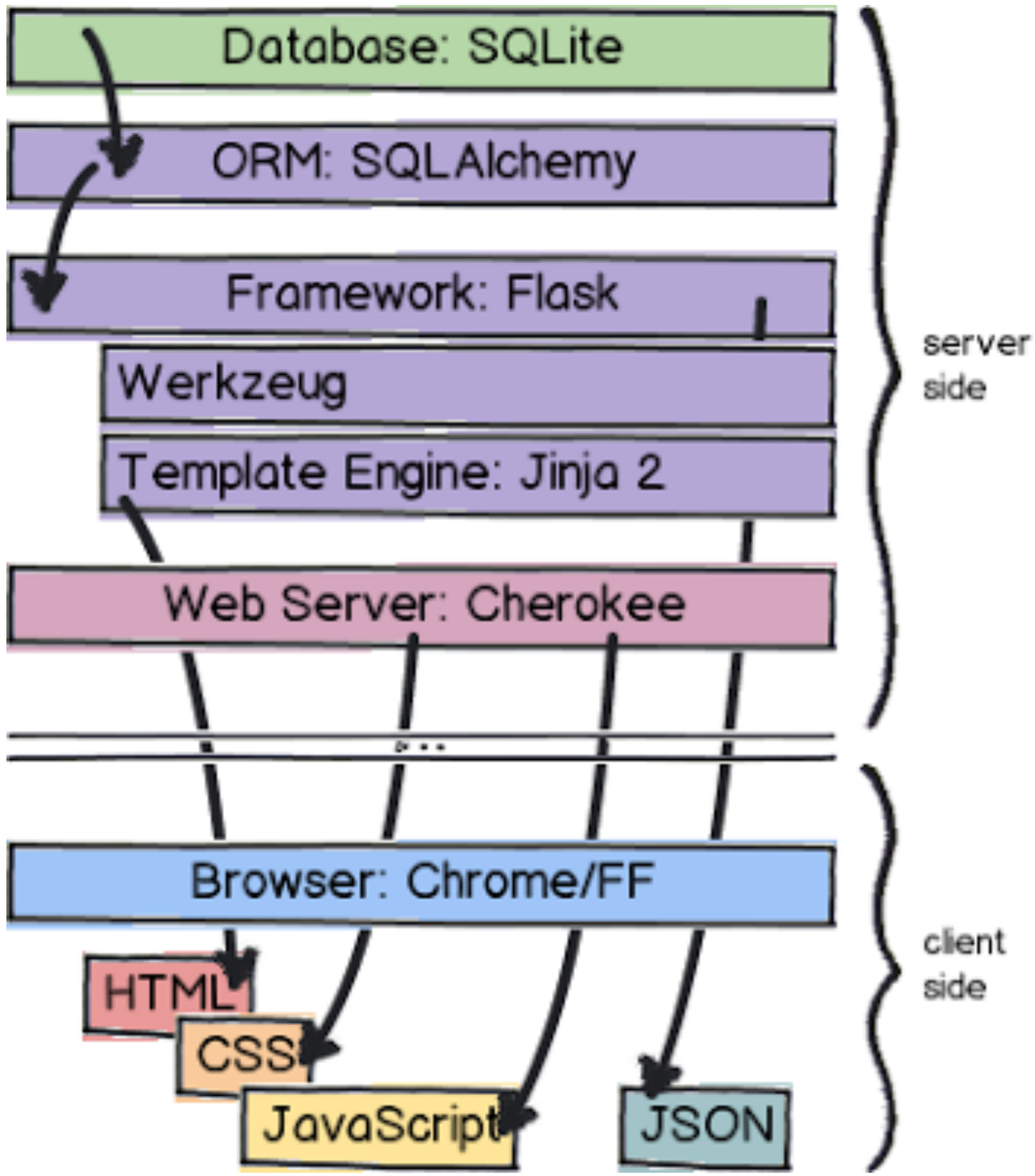
Web App Architecture

Database ORM: SQLAlchemy
Usually querying a DB directly with SQL is considered low level. Object relational mappers (ORM) like SQLAlchemy maps the relational model of a common database into an object (or class) model of an object oriented programming language.



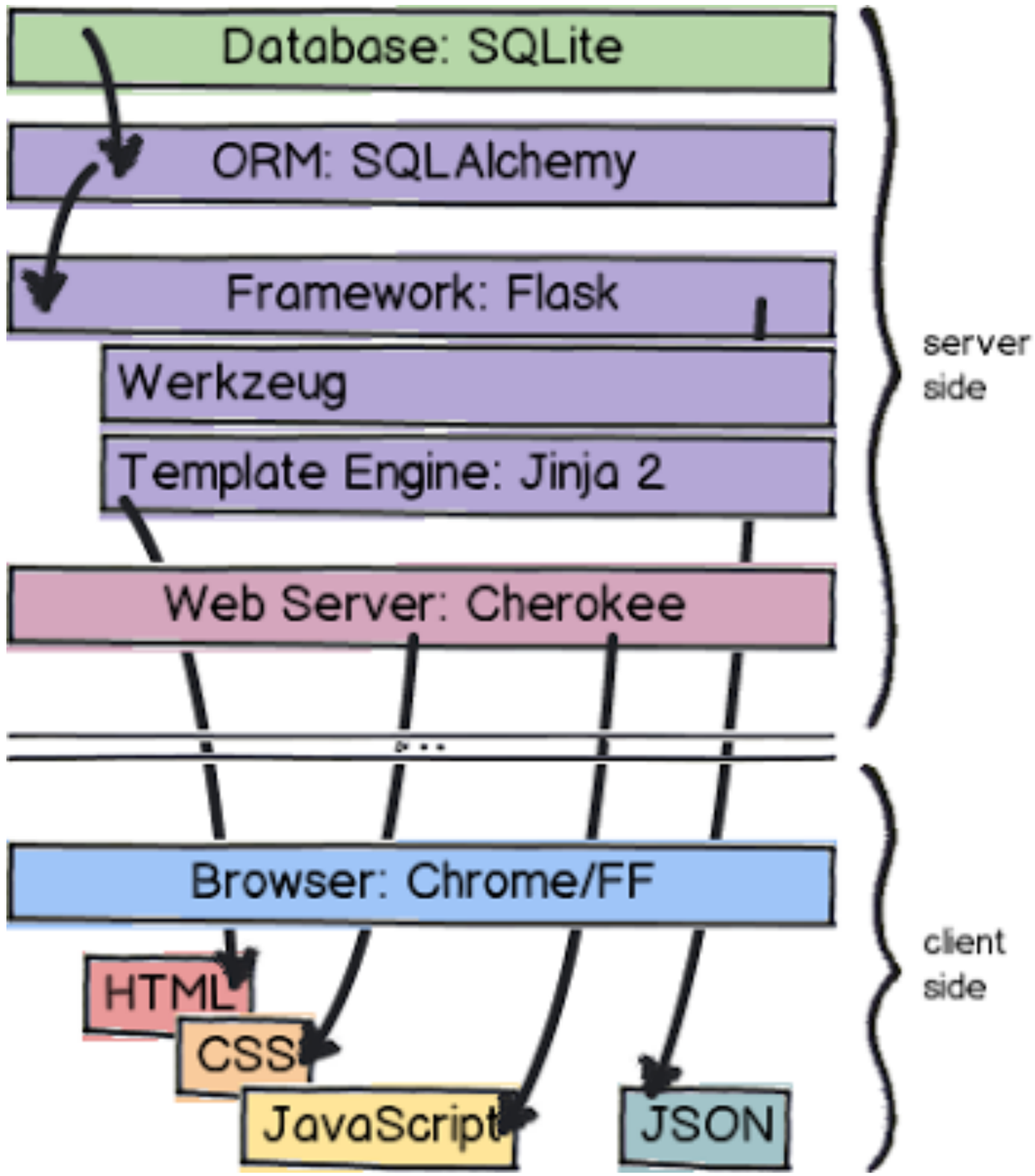
Web App Architecture

Framework: Flask
Flask is a micro-framework, which means that it does not try to solve everything for you, but provides an extendable core of functionality. It is written in the programming language Python and is based on the libraries *Werkzeug* and *Jinja 2*. Flask takes the data from the DB (by querying it with SQL by means of an ORM) and puts the acquired information into other formats (using Jinja 2) that can actually be used by a browser to display your web site.



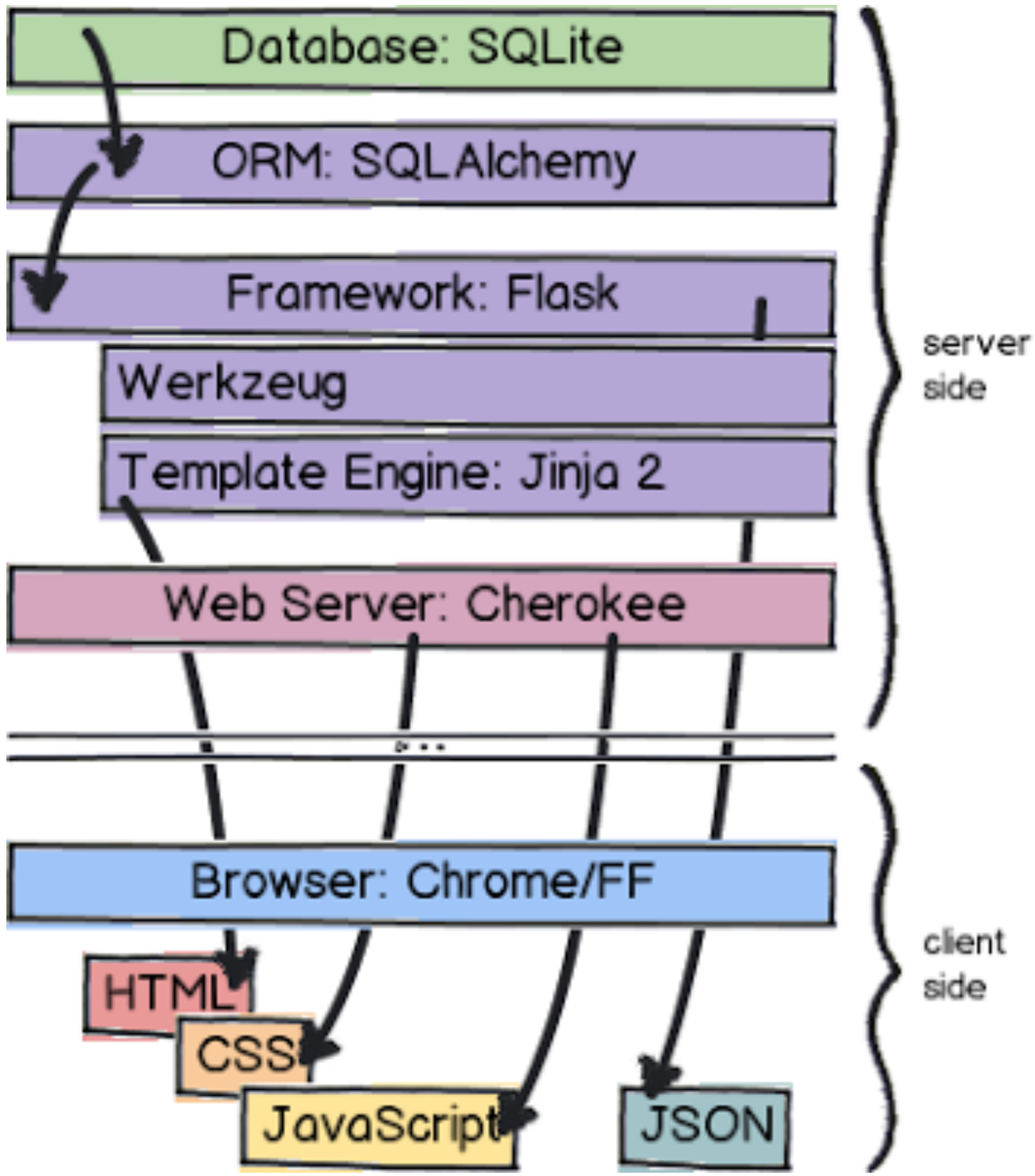
Web App Architecture

HTML, CSS, JS, JSON
HTML: hyper text markup language, is used the write the structure (and content) of a web site; CSS: cascading style sheets describes how a web site should look; JS: JavaScript, is used to implement the dynamic behaviour of a web site.
Combining all three elements a browser is able the render an nice looking, interactive web site, web page, web application etc.



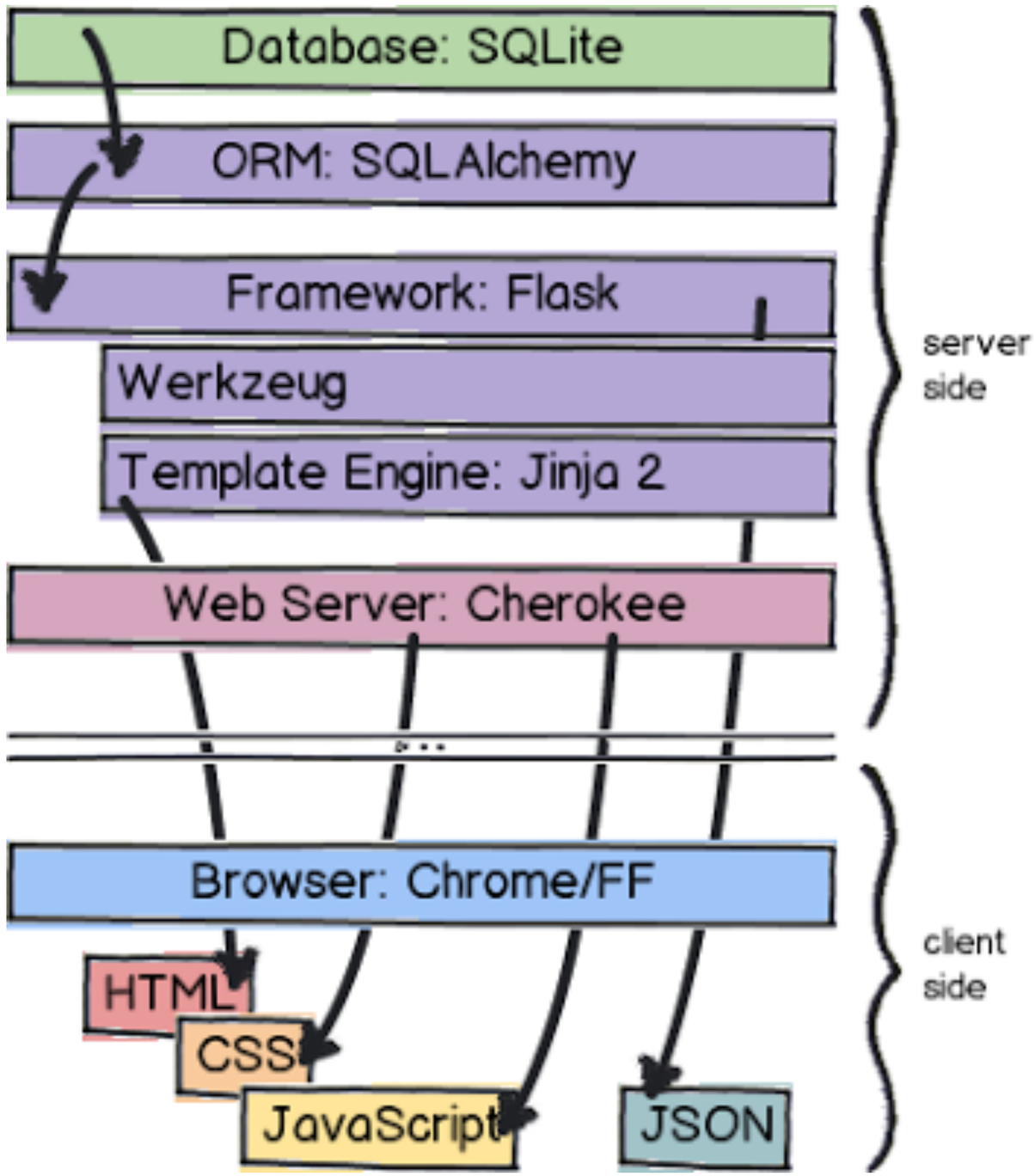
Web App Architecture

Webserver:
Between the client's browser and the framework usually there is another entity sitting: the webserver. In the development phase of a web site a web server is usually not required, since most the the frameworks have a built-in web server.

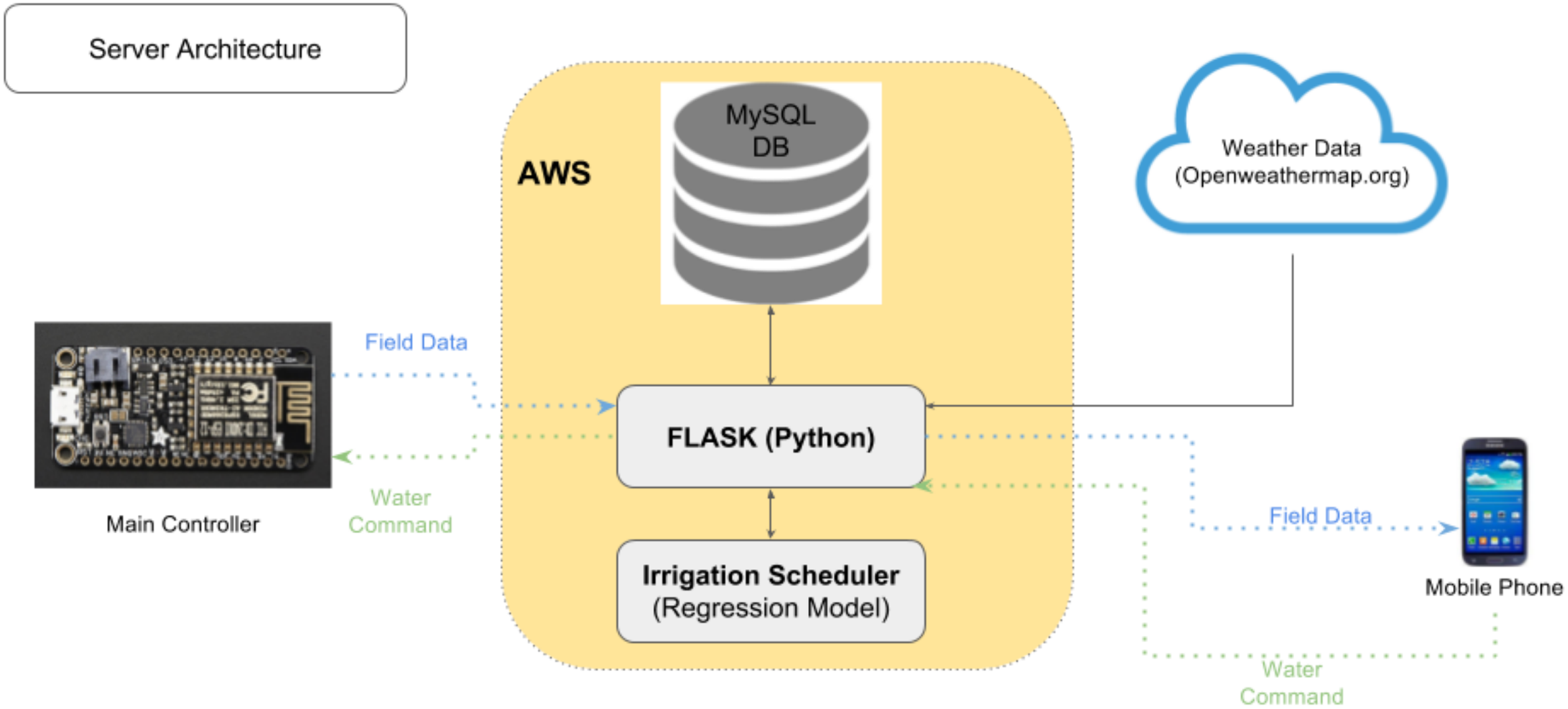


Web App Architecture

Webserver:
Between the client's browser and the framework usually there is another entity sitting: the webserver. In the development phase of a web site a web server is usually not required, since most the the frameworks have a built-in web server.

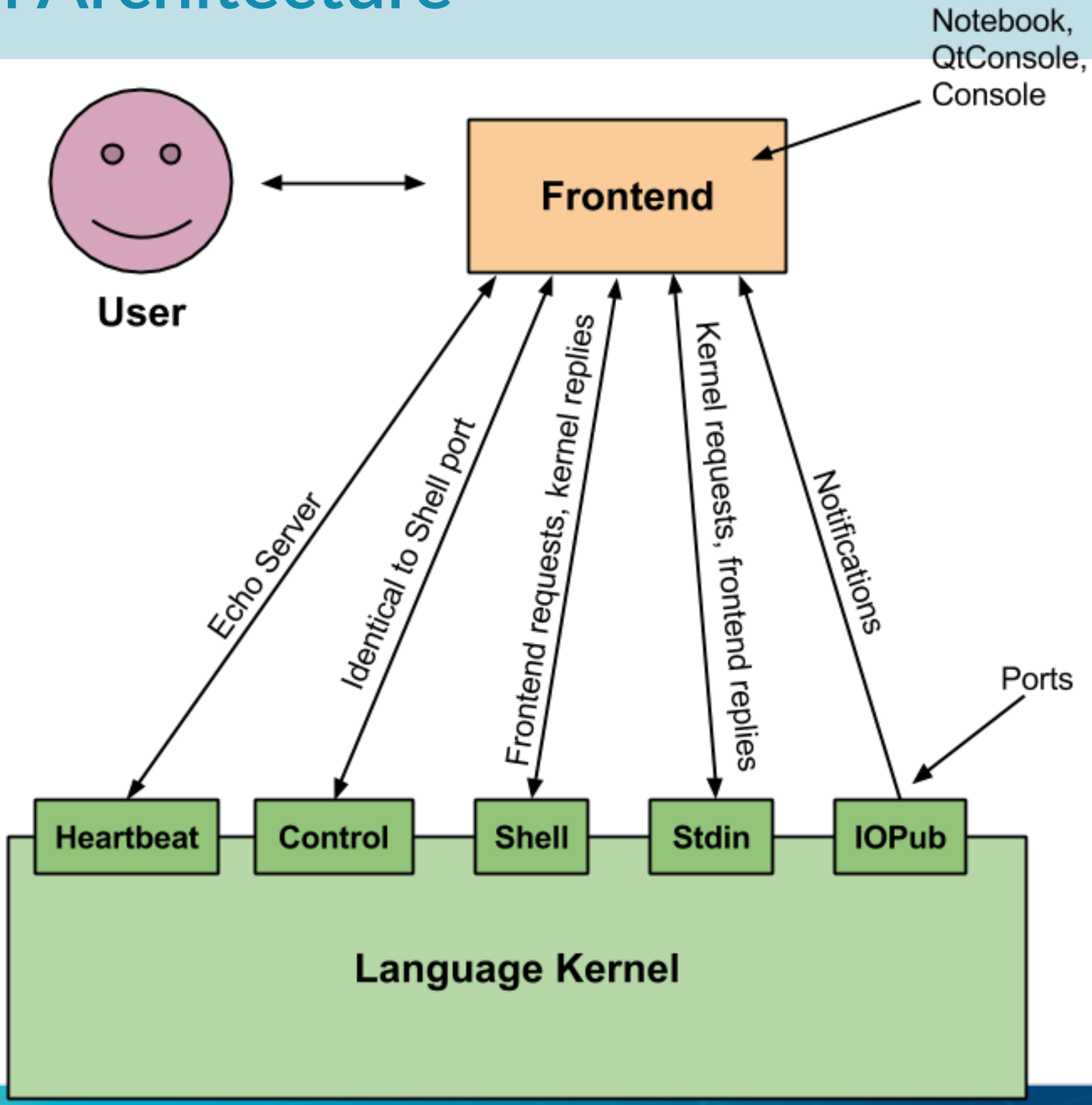


Web App Architecture

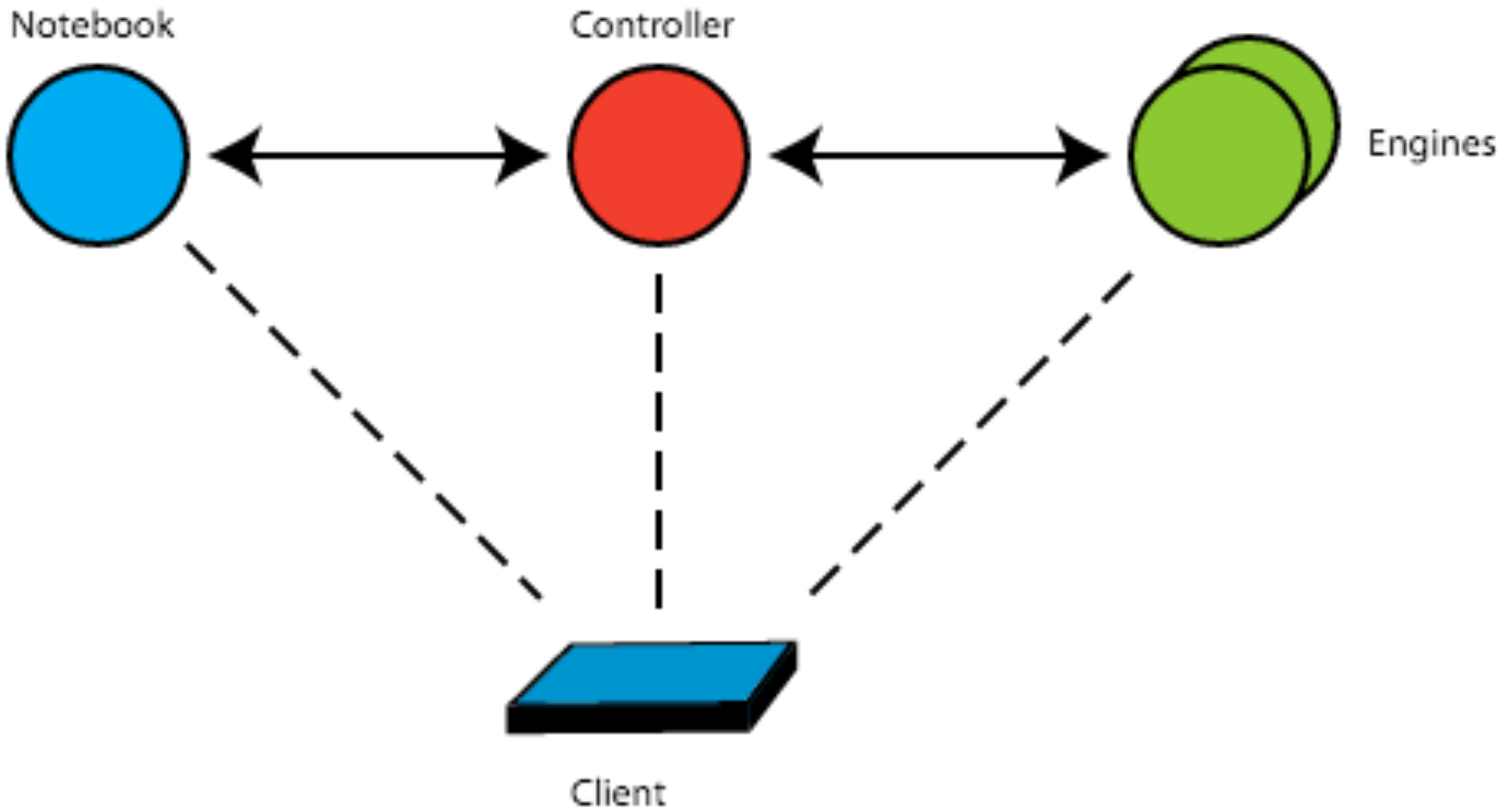



The world of Web Frameworks
by language
PHP
Symfony
CodeIgniter
nette
php
PHP
ZEND FRAMEWORK
Python
Flask
Pyramid
python
Bottle
django
Javascript
ANGULARJS
ember
JS
BABEL
jQuery
Node
Express
koa
flatiron
node
hapi
Java
Guice
GRAILS
Stripes
JSF
tapestry
Java
STRUTS
play
GWT
vaadin
springtutorials.com

iPython Architecture



iPython Architecture



REST

- **REST is an architectural style for networked applications**
- **REST: Representational state transfer**
- resource: the source of some specific information (web page is not a resource, but the representation of a resource)
- global permanent identifier: every resource is uniquely identified (URI)
- standard interface (eg. HTTP)
- set of constraints (eg. separation of concerns, stateless, uniform interface etc)

Request Types

HTTP Method	URI	Action
GET	http://[hostname]/todo/api/v1.0/tasks	Retrieve list of tasks
GET	http://[hostname]/todo/api/v1.0/tasks/[task_id]	Retrieve a task
POST	http://[hostname]/todo/api/v1.0/tasks	Create a new task
PUT	http://[hostname]/todo/api/v1.0/tasks/[task_id]	Update an existing task
DELETE	http://[hostname]/todo/api/v1.0/tasks/[task_id]	Delete a task

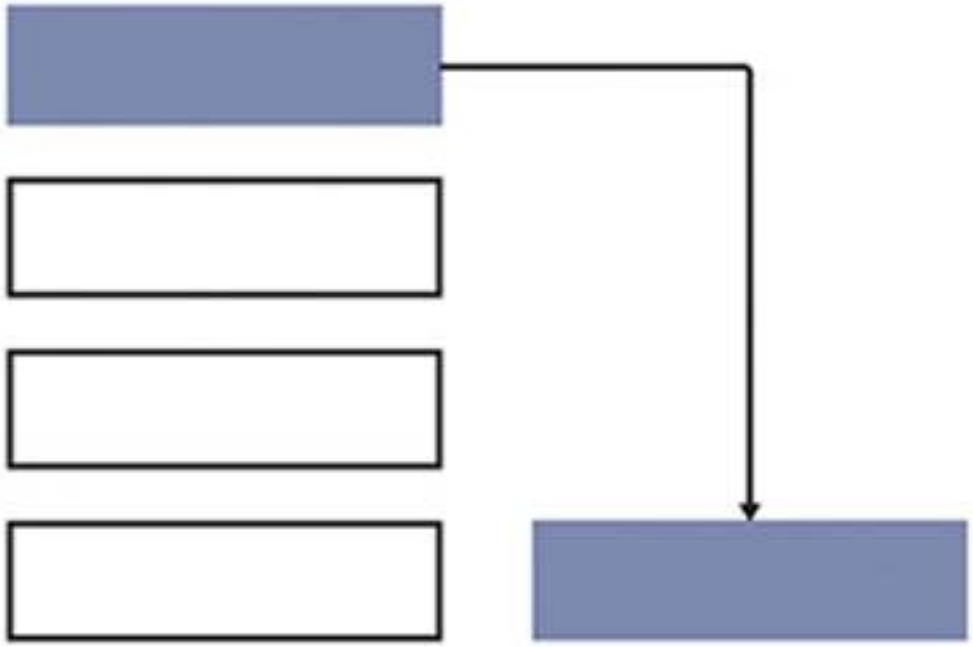
Flask sample

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
if __name__ == "__main__":
    app.run(debug=True)
```

Reasons for Choosing Flask

- simple and elegant
- minimal footprint
- heavily tested
- flexible

Scrum



Product backlog

The product backlog contains features and stories to be implemented.

Sprint backlog

Stories that have been committed to the sprint are moved to the sprint backlog. Each story is implemented in priority order.

Sprint

Each sprint lasts between one and four weeks. With each day of the sprint, the team works toward completing the committed stories.

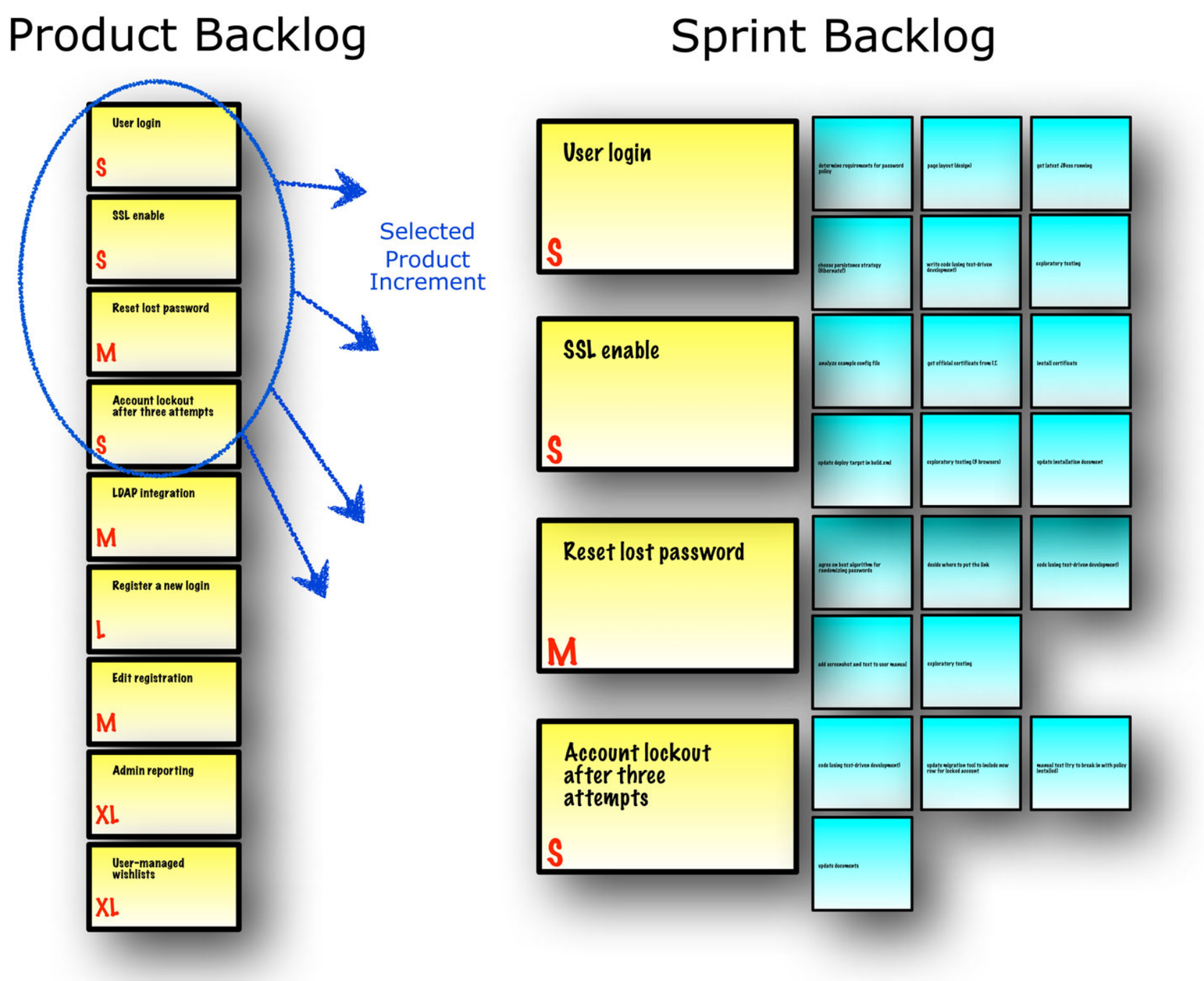
Release

After each sprint, newly implemented features are released in a new version of the product.

Scrum

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

Scrum


Product Backlog
User login
S
SSL enable
S
Reset lost password
M
Account lockout after three attempts
S
LDAP integration
M
Register a new login
L
Edit registration
M
Admin reporting
XL
User-managed wishlists
XL
Selected Product Increment
Sprint Backlog
User login
S
SSL enable
S
Reset lost password
M
Account lockout after three attempts
S
determine requirements for password policy
page layout (design)
get latest JBoss running
choose persistence strategy (Hibernate?)
write code (using test-driven development)
exploratory testing
analyze example config file
get official certificate from I.T.
install certificate
update deploy target in build.xml
exploratory testing (3 browsers)
update installation document
agree on best algorithm for randomizing passwords
decide where to put the link
code (using test-driven development)
add screenshot and text to user manual
exploratory testing
code (using test-driven development)
update migration tool to include new row for locked account
manual test (try to break in with policy installed)
update documents