Architecture of Services

High Performance Computing
(Computació d'Altes Prestacions)

Josep Lluís Berral-García – josep.ll.berral@upc.edu
Jordi Torres Viñals – jordi.torres@upc.edu

Introduction

"Some applications depend on load and data,

others depend on clients and requests"

Session Objectives

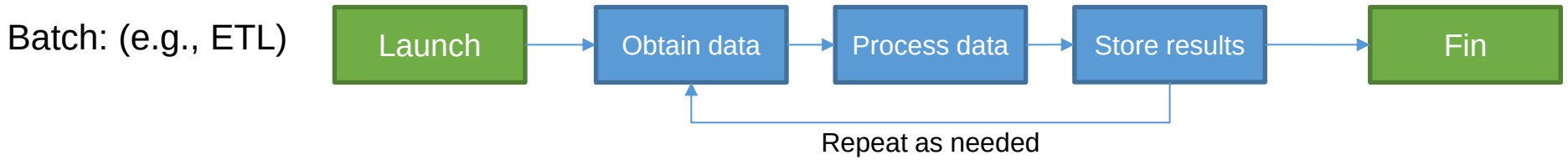
- Explain how transactional workloads use resources
- Explain how services (and pipeline of services) work
  - Describe the purposes of APIs…
  - … and how are used to communicate with services and applications
- Detail step by step a pipeline of common service applications

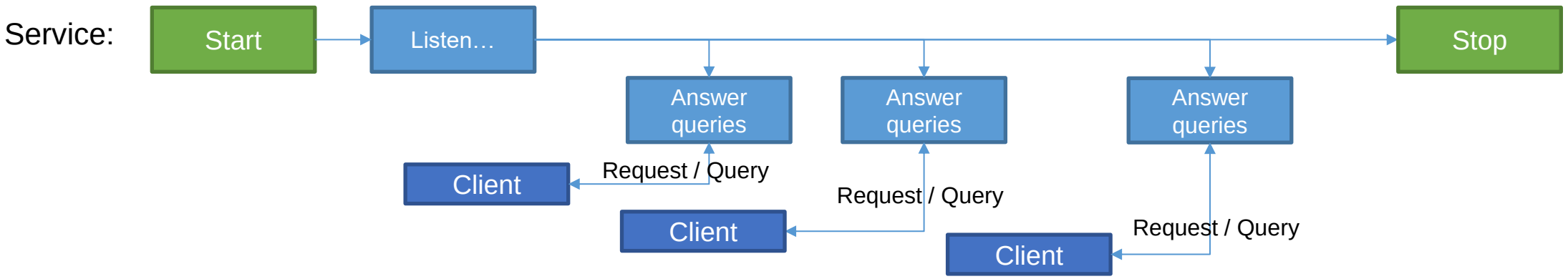
The Service – Client Model

TRANSACTIONAL APPLICATIONS

Transactional Workloads

- Batch vs. Transactional workloads
  - Batch: Depends on the Data | Process
  - Transactional: Depends on Clients | Requests



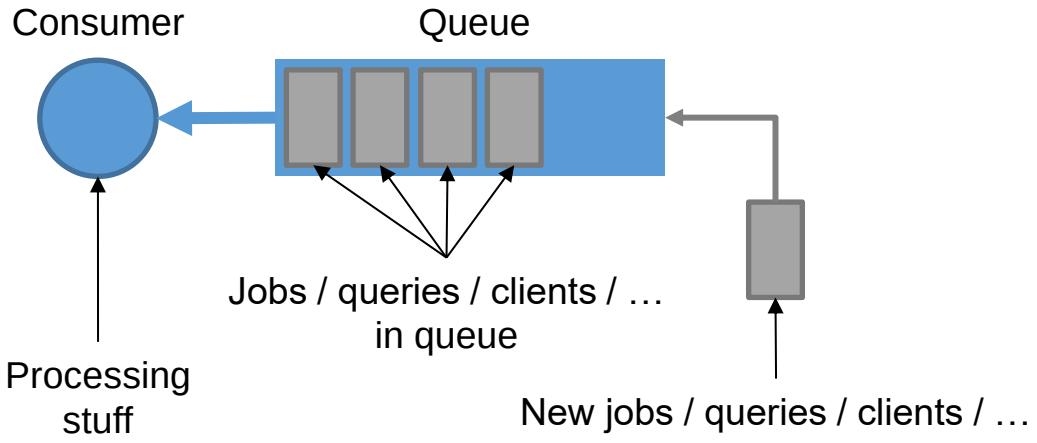


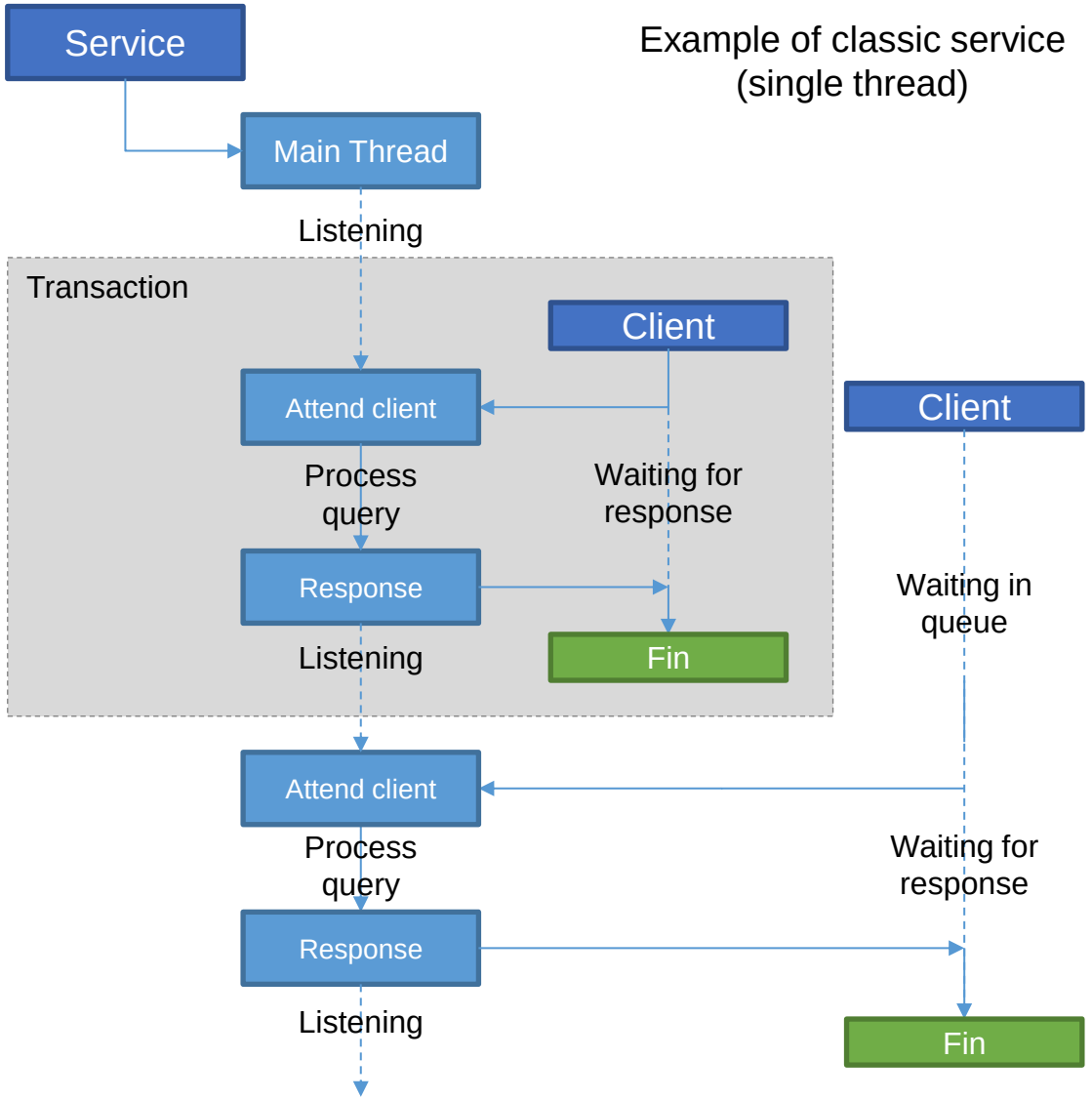
Transactional Workloads

- HPC vs. Transactional workloads
  - Batch workloads (classic and HPC)
    - Demand resources depending on the program
    - Can have "phases". E.g.:
      - Load data (use of I/O resources like Disk or Network, increase Memory usage, …)
      - Process data (use of CPU in a greedy manner)
      - Shuffling data (use of network / communication bus between CPUs, to exchange data)
      - Store data (use of I/O resources, …)
  - Transactional workloads (a.k.a "services")
    - Demand resources depending on the received requests
    - Idle state
      - There are no requests
      - The service is "listening"
      - Very low resources usage, just for the thread listening
    - Working state
      - A client makes "requests" (a.k.a. "queries")
      - The service (usually) creates a thread (from the listening thread) to attend the client
      - The thread demands resources (CPU, Memory, I/O, …) to process the query (the processing could be treated as an HPC job)
      - The thread ends when the client disconnects / the query is fully served / the client doesn't respond for some time
      - The listening thread is always active

Server-Client Model

- Service-Client model
  - Queues and/or Threads
  - The "shop" example

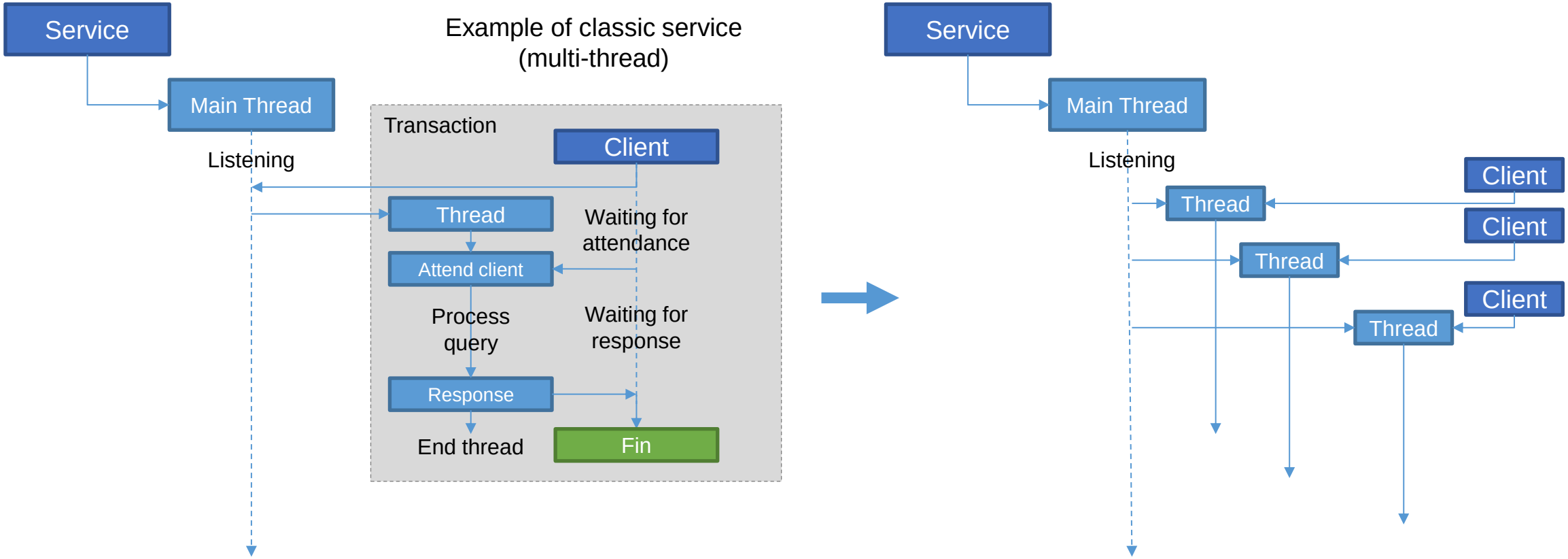




Example of classic service
(single thread)

Server-Client Model

- Service-Client model
  - Multi-Thread / Pool of Threads



Server-Client Model

- Service-Client model
  - Queues and/or Threads

- Single-thread service:
  - One client at a time:
    - The service listens to any client entering
    - Once a client arrives, the thread attends the client
    - Any other client arriving at that moment has to wait in queue
    - Once the client is responded, the thread attends the next in queue

- Queues:
  - Usually:
    - First Come First Serve: First In - First Out (FIFO)
    - Priority queues:
      - Each query receives a priority
      - There are different algorithms to solve priorities
      - The objective is that a low-priority query is attended at some point

Server-Client Model

- Multi-thread service:
  - Each client is assigned a thread to serve it
    - The service listens to any client entering
    - Once a client arrives, a thread is created
    - The thread attends the client
    - Any other client arriving at that moment is assigned another thread
    - Once the client is responded, the thread finishes
  - If the number of threads exceeds the capacity of the service (application process) or the server (machine):
    - New clients go to a queue
- Queues:
  - If there are no resources available:
    - All current threads are using all the resources
    - New processes wait in queue

Little's Law: L = λW

- Performance is related to demand and capacity
  - Little's Law → L = λW
- Example (batch jobs)
  - Experiment:
    - 1 CPU per experiment
  - We submit 100 experiments per hour
    - Demand of 100 CPUs in that hour
    - λ = 100 CPU / hour
  - Experiments take an average of half hour
    - W = 0.5 hours
  - Average number of exps. on our system:
    - L = 50 CPUs = 50 experiments
    - **Average use** = 50 CPUs = 50 experiments in average

Our System requires 50 CPUs

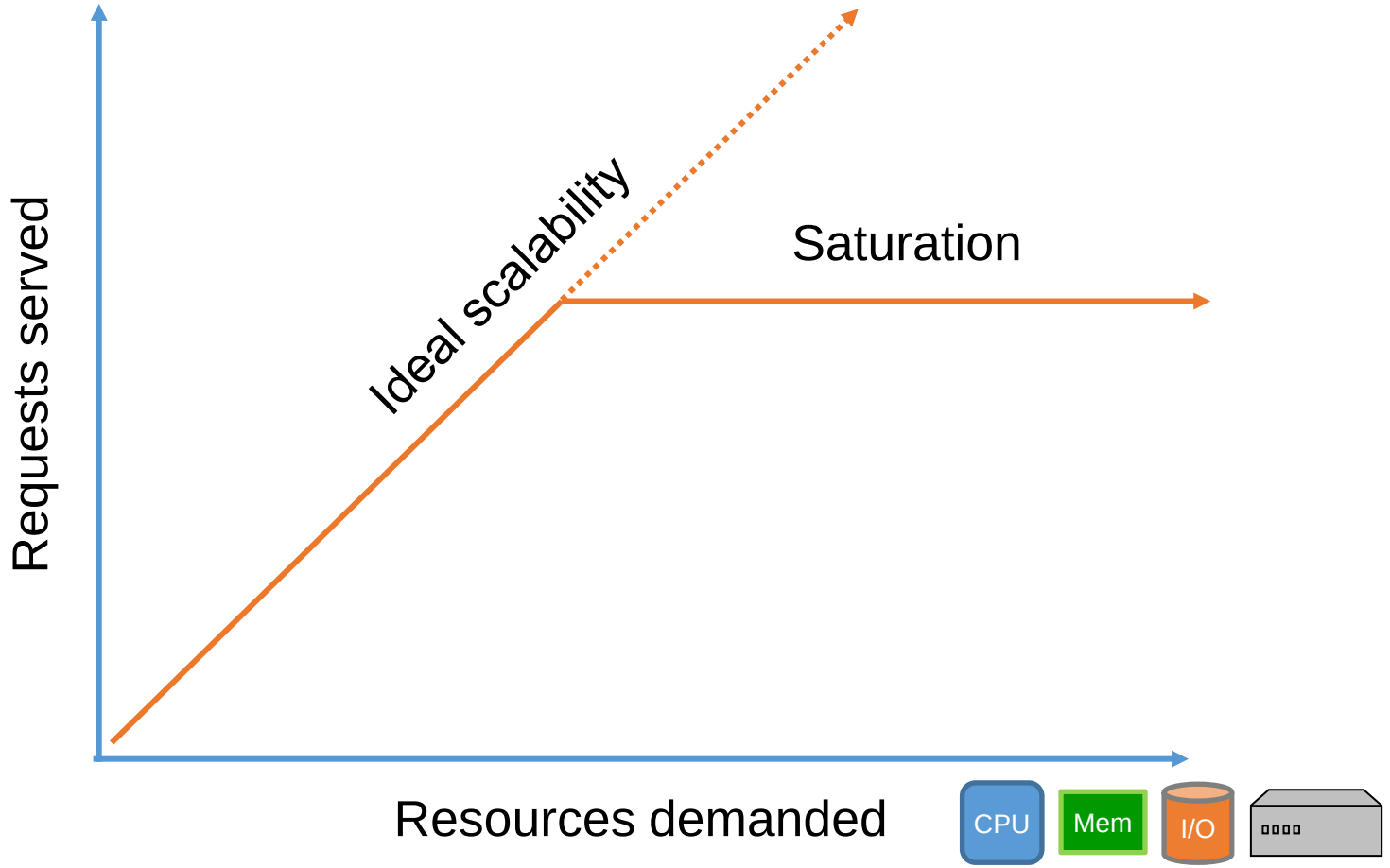
Little's Law: L = λW

- Performance is related to demand and capacity
  - Little's Law → L = λW
- Example (transactional jobs)
  - Client submitting a query:
    - 1 Thread per client
  - We receive 100 clients constantly in average
    - Demand of 100 Threads concurrently
    - λ = 100 Threads
  - Client queries require 0.5 CPU in average
    - W = 0.5 CPU / Thread
  - Average number of CPUs required on our system:
    - L = 50 CPUs
    - **Average load** = 50 CPUs to serve these 100 clients concurrently

Our system requires 50 CPUs to support an average load of 100 clients concurrently

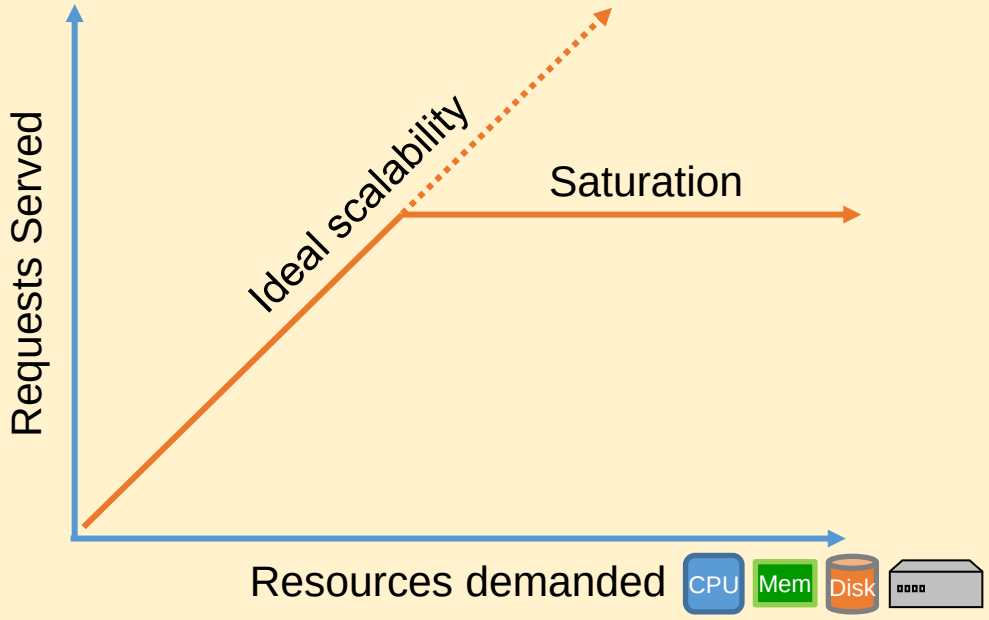
Performance (Reprise)

- Performance



Performance (Reprise)

- Performance
  - Throughput: Queries (clients) served per time-unit
  - Working zone:
    - We can admit more queries (clients)
    - The more clients, the more resources used
  - Saturation zone:
    - We don't have resources for more queries
    - Queries (clients) go to a queue
  - Ideal scalability:
    - Once we reach the saturation zone…
    - … we open a new server, and send the exceeding requests to it



Services Communication and Pipelines

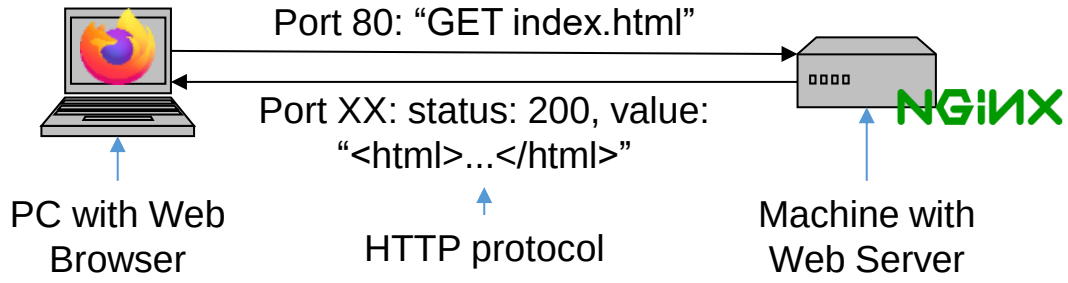
APP PROGRAMMING INTERFACES (API)

APIs and Interfaces

- Application Programming Interface (API)
  - Communication Protocol for an application

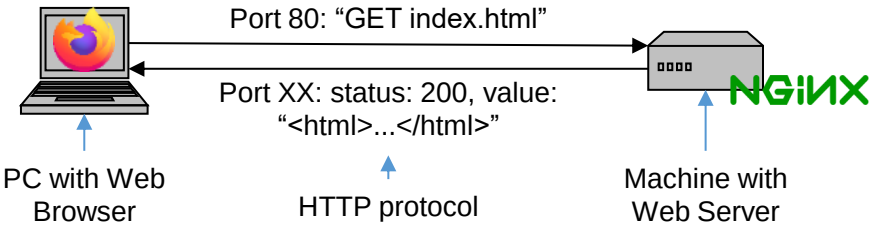
- How we speak to an application:
  - Where to find it → Address, port, …
  - Which language it uses → Returns a value, a message, a file, …
  - What can we say to it → Get or set a value, …

Example of a web service:



APIs and Interfaces

- Example "Web Server":
  - Protocol
    - HTTP
    - Default listening port: 80 (raw data http), 443 (secure layer https)
  - Operations available from Client
    - GET: ask for a file
    - PUT: upload/modify a file
    - POST: upload data
    - HEAD: ask for information of a file
  - Reply from Server
    - File: returns the requested file or information (GET, HEAD)
    - Status: returns a status code
      - 1XX: info, 2XX: OK, 3XX: redirect, 4XX: client error, 5XX: server error



APIs and Interfaces

- Application Protocol Interface
  - Every application defines how to interact with them
    - This can be users or other applications
  - The API must specify
    - Where to find them (e.g., port, file, etc.)
    - Which format the application understands and expects
      - E.g., a ML application might ask for a JSON file with fields
        - » "dataset_tr" and "dataset_ts" with the route to dataset files,
        - » "model" with the route where to store the trained model,
        - » "algorithm" with the algorithm to use for train,
        - » "hyperparameters" with a list of hyperparameters depending on the algorithm
    - Which format and data is returned
      - The client application calling the service application must understand the reply
      - It can be another formatted file with the reply, or indicating the status of the query, or a requested file, etc.

APIs and Interfaces

- Files and Formats
  - Examples
    - HTML / XML / JSON: file or message with tag format
    - Image: file with an image (JPEG, PNG, GIF, …)
    - Other: dedicated formats, specified in the app API
  - Example of a client request in JSON

```
{
        "autentification" : [
                "user" : "AAAA",
                "password" : "XXXXXX"
        ],
        "database" : "db1",
        "query" : "SELECT * FROM table WHERE J > 10"
}
```

APIs and Interfaces

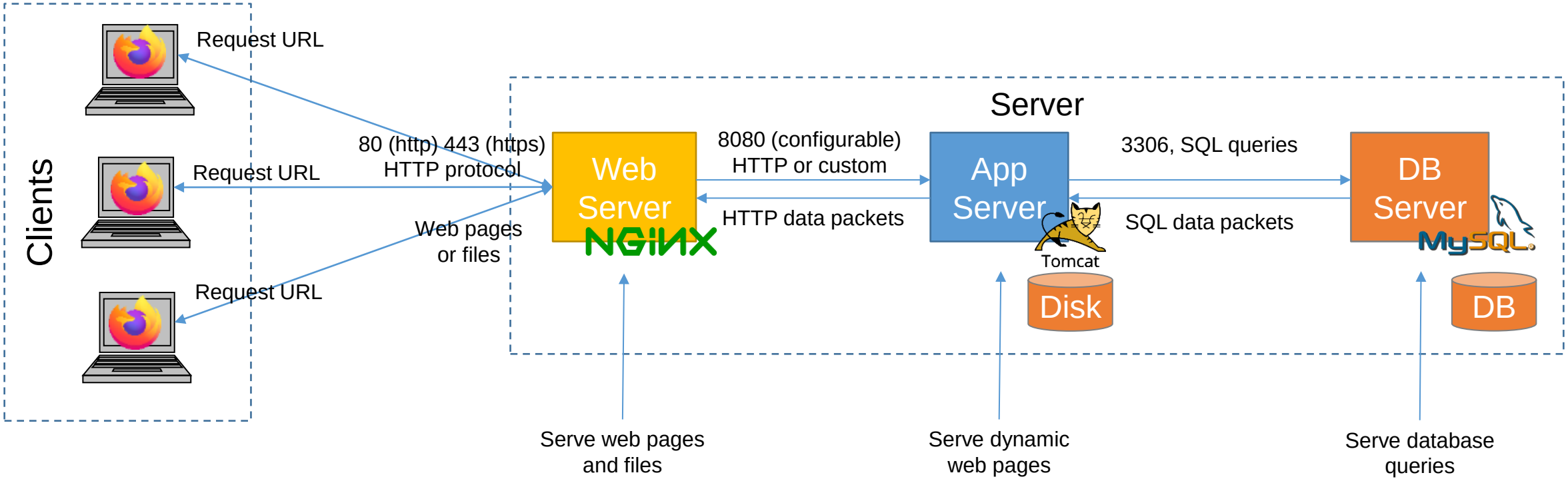
- Files and Formats
  - The API needs to specify the formats used
    - The client application uses this to talk to the service
    - (The client/user communicates to the service through a client application, such as a web-browser, a command application, etc.)
  - There are standard structured formats (HTML, XML, JSON, YAML, MD…)
    - These formats can be read by a machine without being ambiguous
    - The content (fields and values) are defined by every application
    - … so, the application must publish their API for others to use the service

Common Examples on Services and Applications

SERVICE EXAMPLES

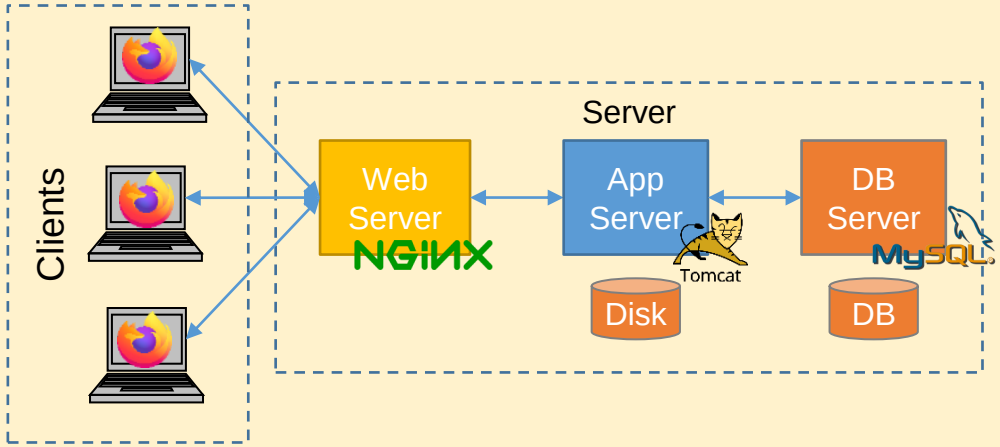
Service Examples – Web Service

- Example of a Web Server



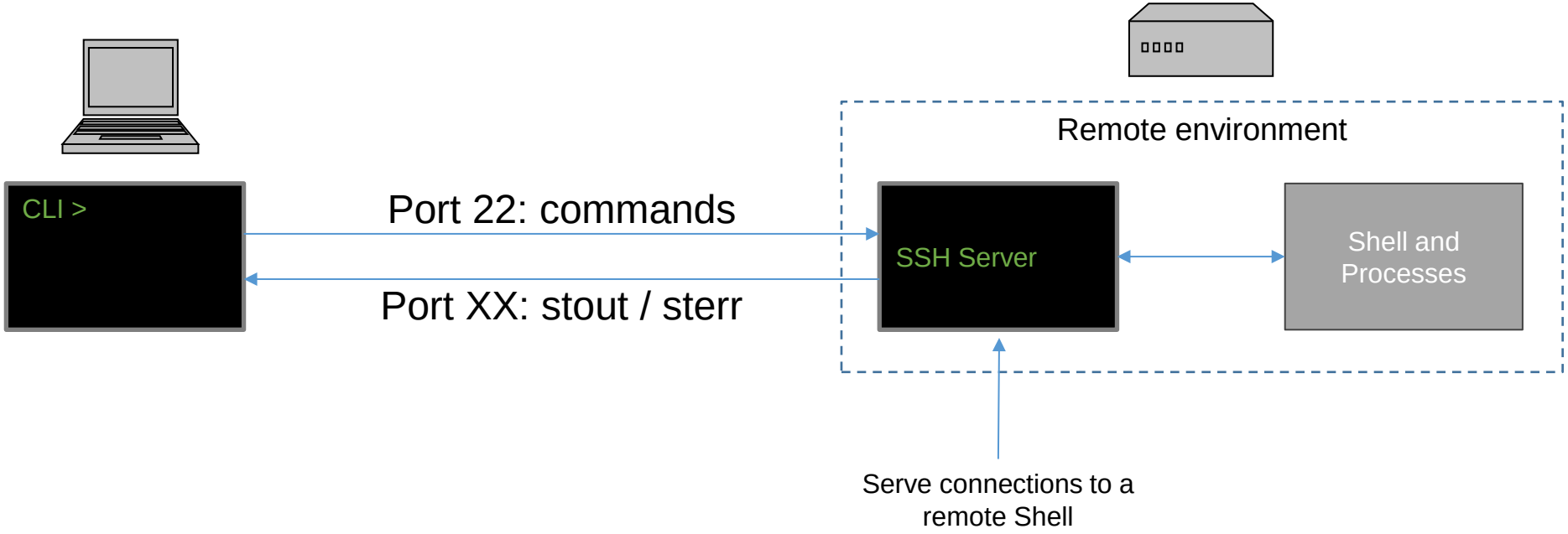
Service Examples – Web Service

- Example of a Web Server
  - Web Browser (Firefox)
    - User inserts URL for a web page or file
    - Browser → Talks to Web server through URL and port
  - Web Server (Nginx) – Serves web pages
    - Default port 80 (http), 443 (https)
    - HTTP protocol (GET, PUT, POST, HEAD)
      - » Returns HTML or other files, and status info
    - Dynamic pages → Talks to App Server
  - App Server (Tomcat) – Serves dynamic web pages
    - Default port configurable (e.g., 8080)
    - HTTP or custom protocol
      - » Returns data packets (can be HTML files), and status info
    - SQL queries → Talks to Data Base Server
  - Data Base Server (MySQL) – Serves SQL queries
    - Default port 3306
    - SQL protocol (data packet with SQL queries)
      - » Returns data packet with SQL response



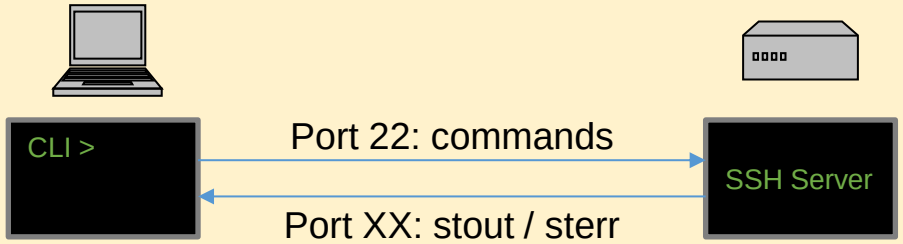
Service Examples – Secure Shell

- Example of Secure Shell



Service Examples – Secure Shell

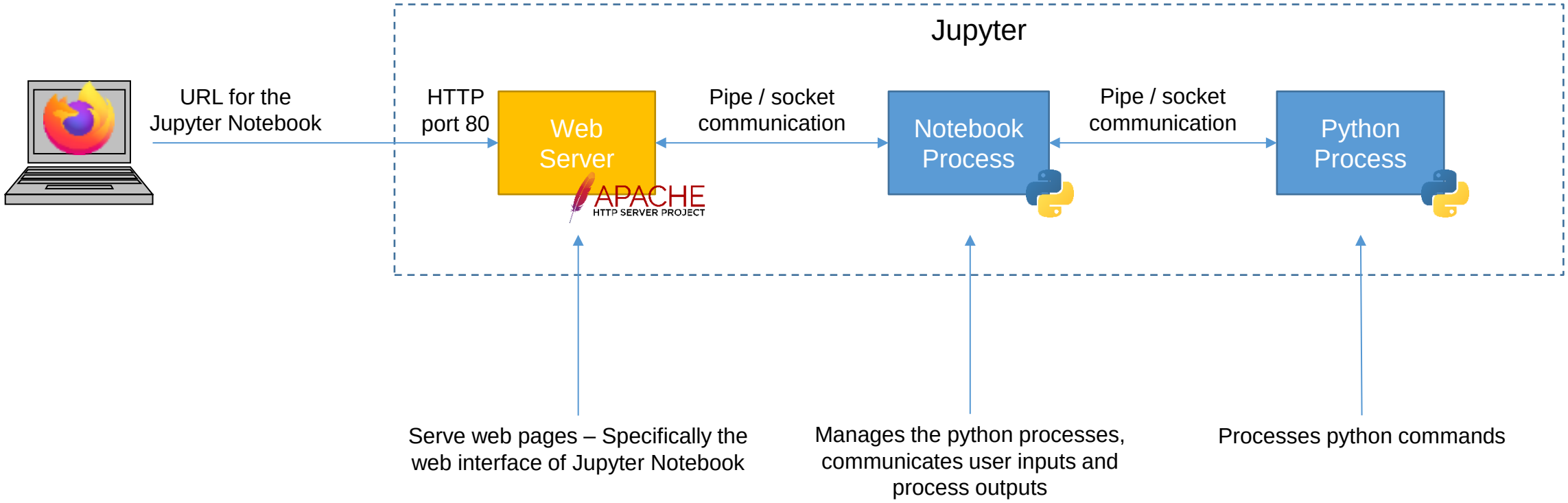
- Example of Secure Shell



  - Client terminal
    - User connects to remote machine
    - Provides user, password
    - Default port: 22 (ssh)
    - Protocol: SSH (exchange commands and standard outputs)
      - » Sends commands to be executed
  - SSH server
    - Receives connection from remote user
    - Executes received commands into a Shell
    - Protocol: SSH (exchange commands and standard outputs)
      - » Returns the standard outputs

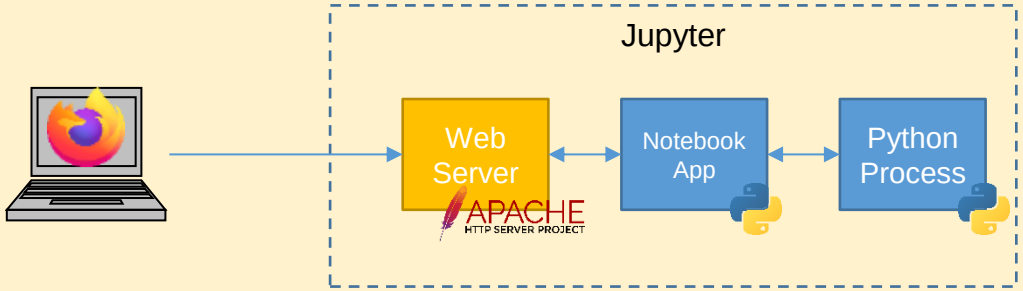
Services Examples – Notebooks

- Example of Notebooks



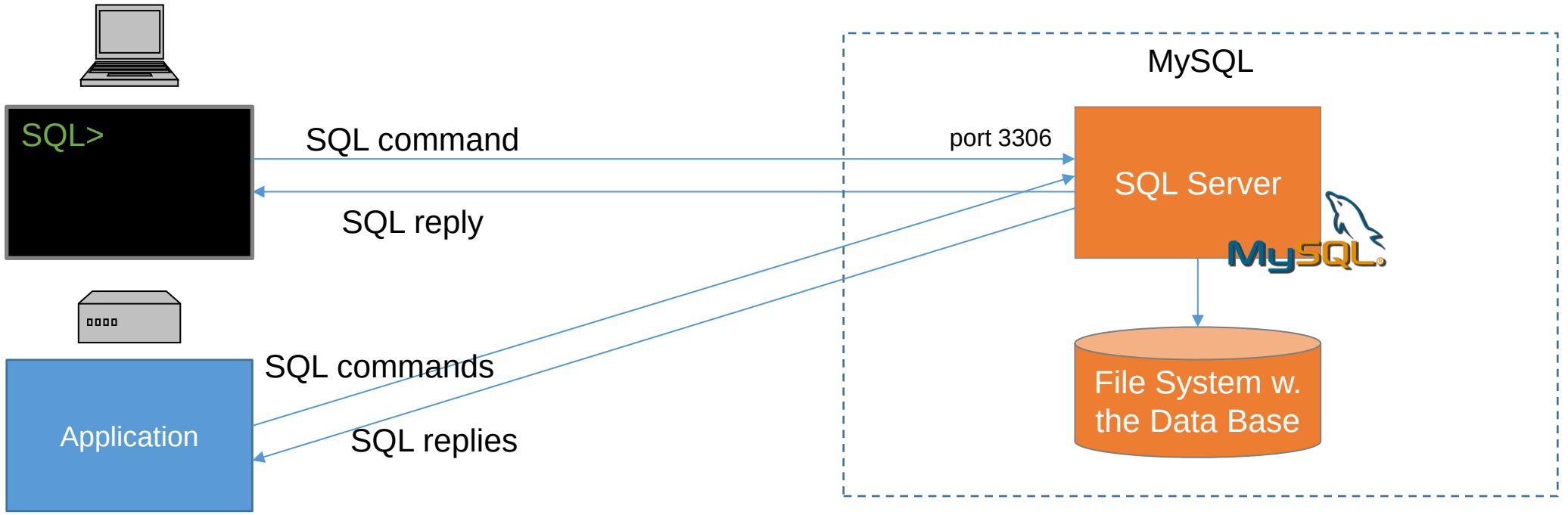
Services Examples – Notebooks

- Example of Notebooks
  - User
    - Web Browser (Firefox)
      - User inserts URL for the Jupyter Notebook
      - Browser → Talks to Web server through URL & port
  - Jupyter
    - Web Server (e.g., httpd)
      - Listens to port 80
      - Sends user input to Notebook process
        - » Shows results in web page
    - Notebook process
      - Listens to web server (pipes, sockets, …)
      - Sends python commands to python processes
        - » Returns results to web-server
    - Python
      - Listens to pipe from notebook process
      - Receives commands to execute
        - » Return results as text/image to notebook



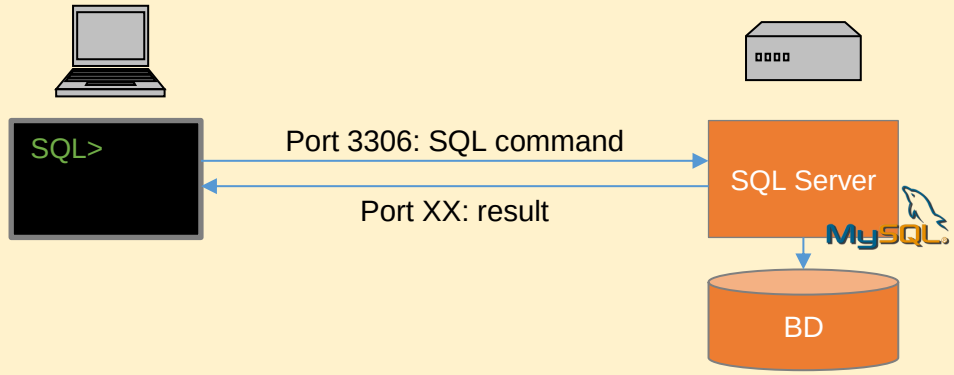
Services Examples – DBs & File Systems

- Example of DBs / File Systems



Services Examples – DBs & File Systems

- Example of DBs / File Systems
  - Option 1: User
    - SQL command line
      - User connects to remote DB
        - » Inserts server name or IP and port
        - » Authenticates with the server
      - Inserts SQL query
      - Client app → Talks to SQL server
  - Option 2: Application
    - Some Application
      - Connects to remote DB
        - » Uses server name or IP and port (in the app configuration)
        - » Authenticates with the server
      - Sends SQL query
      - Exchanges messages with the SQL server



  - Server
    - DB Server (e.g., MySQL)
      - Listens to port 3306
      - Processes SQL requests
        - » Returns results to the client

Session Objectives

- Explain how transactional workloads use resources
- Explain how services (and pipeline of services) work
  - Describe the purposes of APIs…
  - … and how are used to communicate with services and applications
- Detail step by step a pipeline of common service applications

Laboratori 3 – Serveis

PRÀCTICA DE SERVEIS

Laboratori

- Entorn:
  - Singularity → Hypervisor (gestiona i executa els contenidors)
  - Repositori de Singularity → Repositori públic de contenidors ja configurats
- Sistema Operatiu:
  - VM amb Ubuntu + Singularity
    - Ho executarem tot dins una VM amb Singularity pre-instal·lat
    - A la pràctica tindrem 2 nivells d'imbricació (màquina → VM → contenidor → App)
    - La VM s'haurà de configurar per a que els serveis creats es vegin "des de fora"
- Qüestionari:
  - Durant la pràctica cal resoldre preguntes respecte el que estem executant i observant

Laboratori

- Tenir llest l'entorn de Virtualització
  - Re-usar la VM preparada per a contenidors
- Crear una imatge de Contenidor per a Jupyter
  - Preparar una imatge "custom" que inclogui Python i Jupyter
  - Instanciar la imatge i veure que pot servir Notebooks
- Desplegar la imatge
  - Configurar la imatge per a mostrar-se a l'exterior
  - Connectar-se a la instància per navegador web