Introduction

DATA 236, Distributed Systems for Data Engineering

Build and Learn

What are we building?

build scalable, fault-tolerant services and distributed systems, then layer agentic workflows (planning, tool use, orchestration) on top of these systems. Emphasis is on scalability, consistency, reliability, observability, and safety. Note: This is a **rigorous, project-heavy** course.

What You'll Learn

- Design and evaluate distributed architectures (microservices, event-driven, messaging).
- Apply CAP/consistency models and data partitioning/replication strategies, NoSQL DB.
- Deploy services with containers (Docker) and implement data distribution, service scalability, high db throughput, caching.
- Add agentic AI layers: tool schemas, retrieval, planning/reflect loops, and safe execution.
- Instrument systems for observability (tracing, metrics, logs) and run load/reliability tests.

Workload & Assessment

- Weekly demo and homeworks (learn, practice)
- 2 Labs, programming assignments (end-to-end data flow)
- Final Group project: production-style **distributed data service + agentic AI workflow**
- Group participation and code reviews highly required

Refresher Homework

- Practice JavaScript, docker, AWS
- Focus on new features
  - Arrow functions
  - Asynch-await
  - Promise

Projects

Projects are an opportunity to further discussions and exploration through practical experience.

2 Programming projects (Lab) and 1 group project

Programming projects—individual or pair
Class project: Team project design and implementation

Manage your time carefully

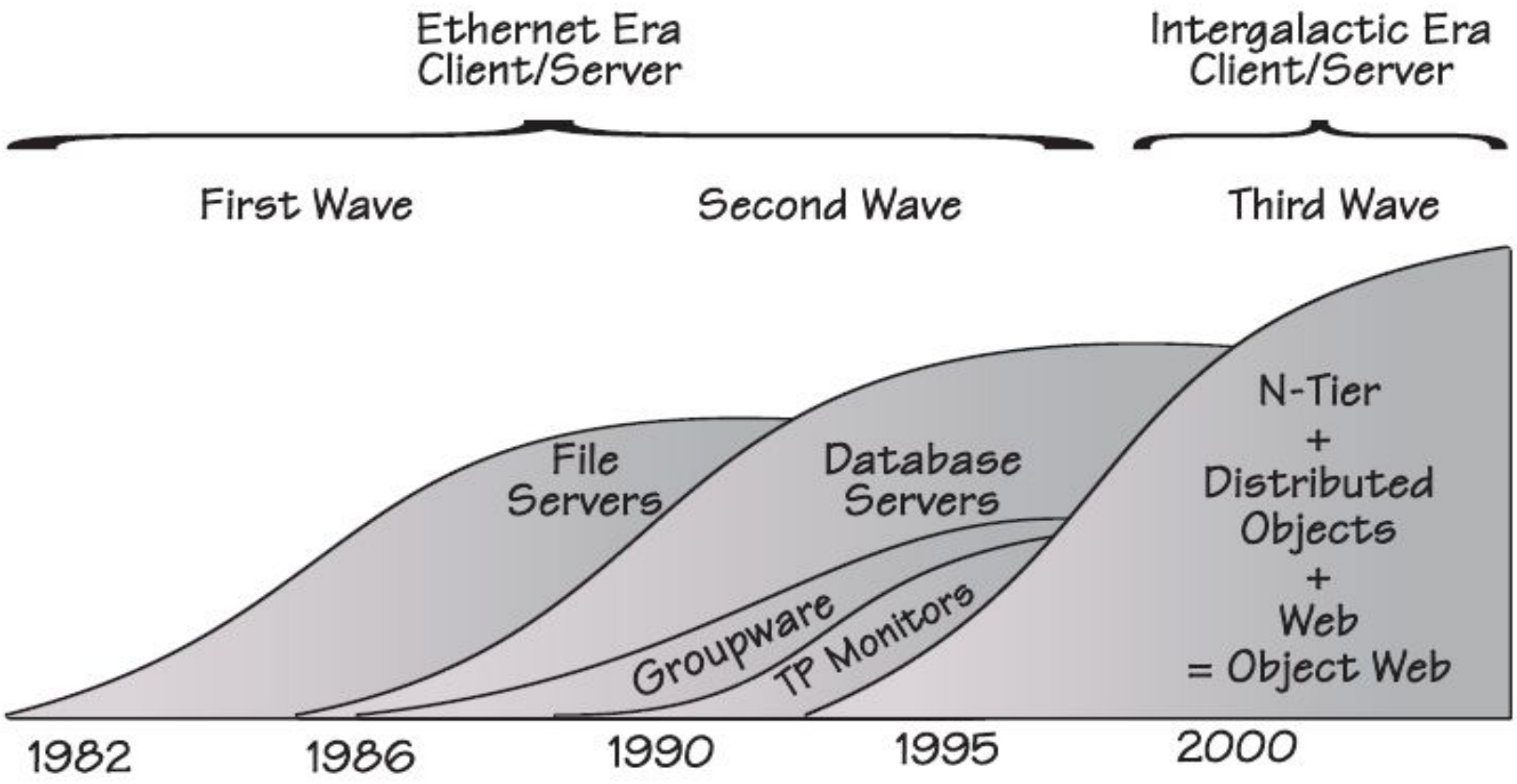
- Projects take 30-70 hours to complete
  - **Do not wait until the last week**
  - **Rather program every day**

- What you have to plan for
  - Project investigation
  - Understanding concepts and technologies
  - Implementation
  - Testing
  - Documentation

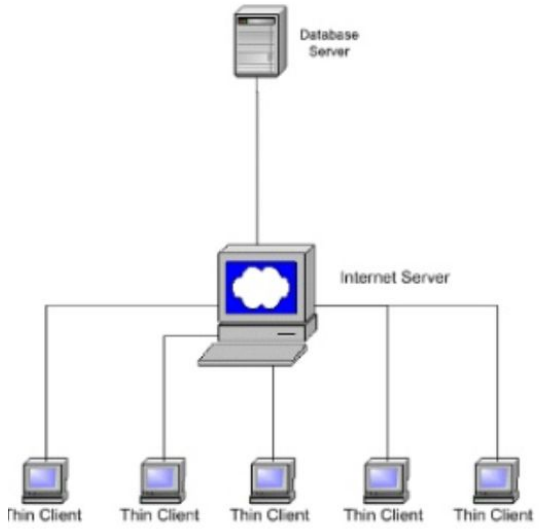
Client Server Architecture

- A three-way interaction in a client/server environment
  - User interface is stored in
  - Business application logic is stored in
  - The data is stored in

Client/Server



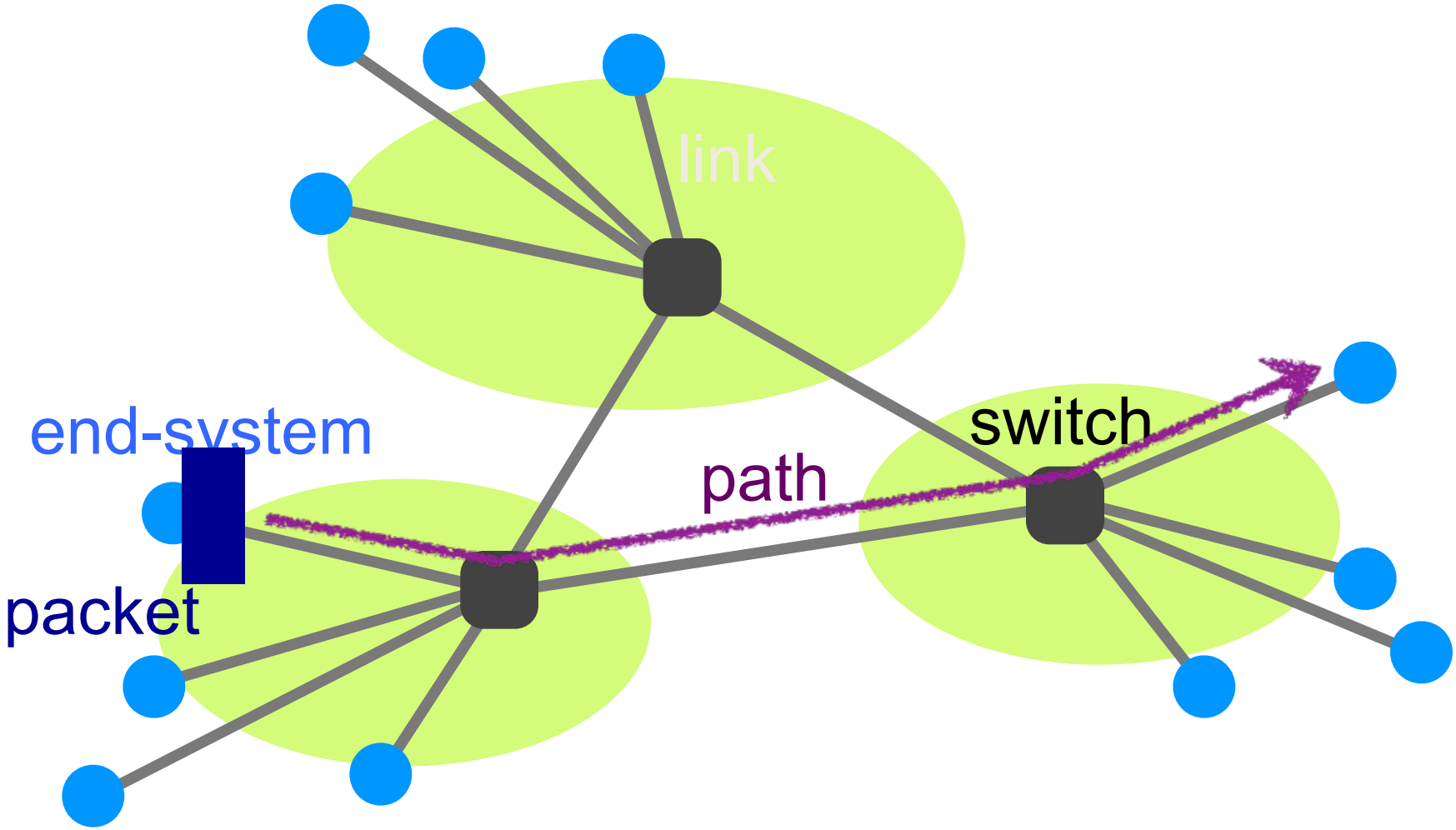
- Single Tier: “Mainframe” era
- Dual Tier: “personal Computer” era
- 3-Tier: “thin client”



Goals for the Internet

- Ability to connect many different networks
  - Ethernet and optical and Wifi and ….
- Ability to scale to entire world
- Ability to tolerate and recover from failures
- Able to support wide variety of applications

The central mechanism of Internet?



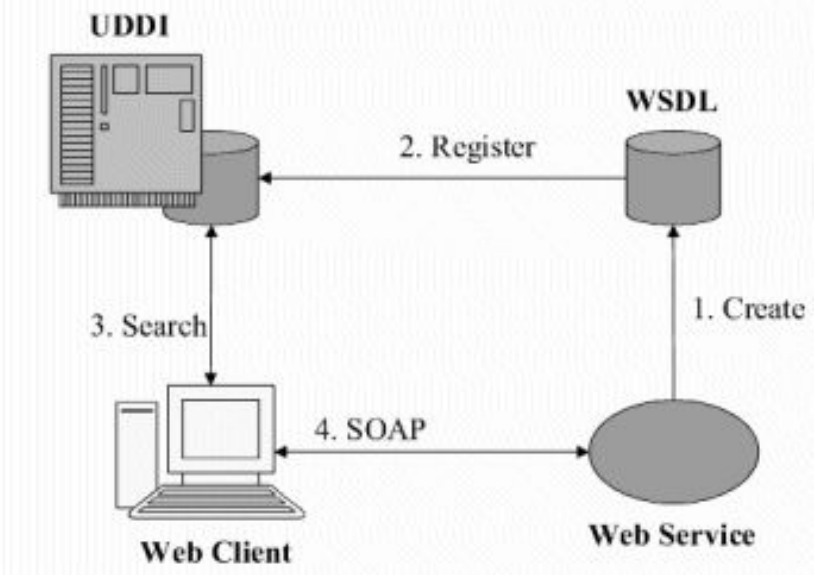
Tremendous scale

- 3.8 Billion users (51% of world population)
- 1.24 Trillion unique URLs (web pages)
- Every second, approximately
  - > 6,000 tweets are tweeted
  - > 40,000 Google queries are searched
  - > 2 million emails are sent

We use the phrase "Internet Scale" to refer to such systems

Web services

- HTTP based
- SOAP/UDDI/WSDL is a general framework (based on XML) for describing network services



RESTful Services

- Representational State Transfer (REST)
  - Roy Fielding, 2000 (doctoral dissertation)
    - Examination of the Internet as a stateless service of near-limitless expansion model with a simple but effective information delivery system

- Key concepts
- Resources - source of information
- Consistent access to all resources
  - As in interface and communication – Not content or function
- Stateless protocol
- Hypermedia – links in the information to other data (connectedness)

Cloud Computing/AWS

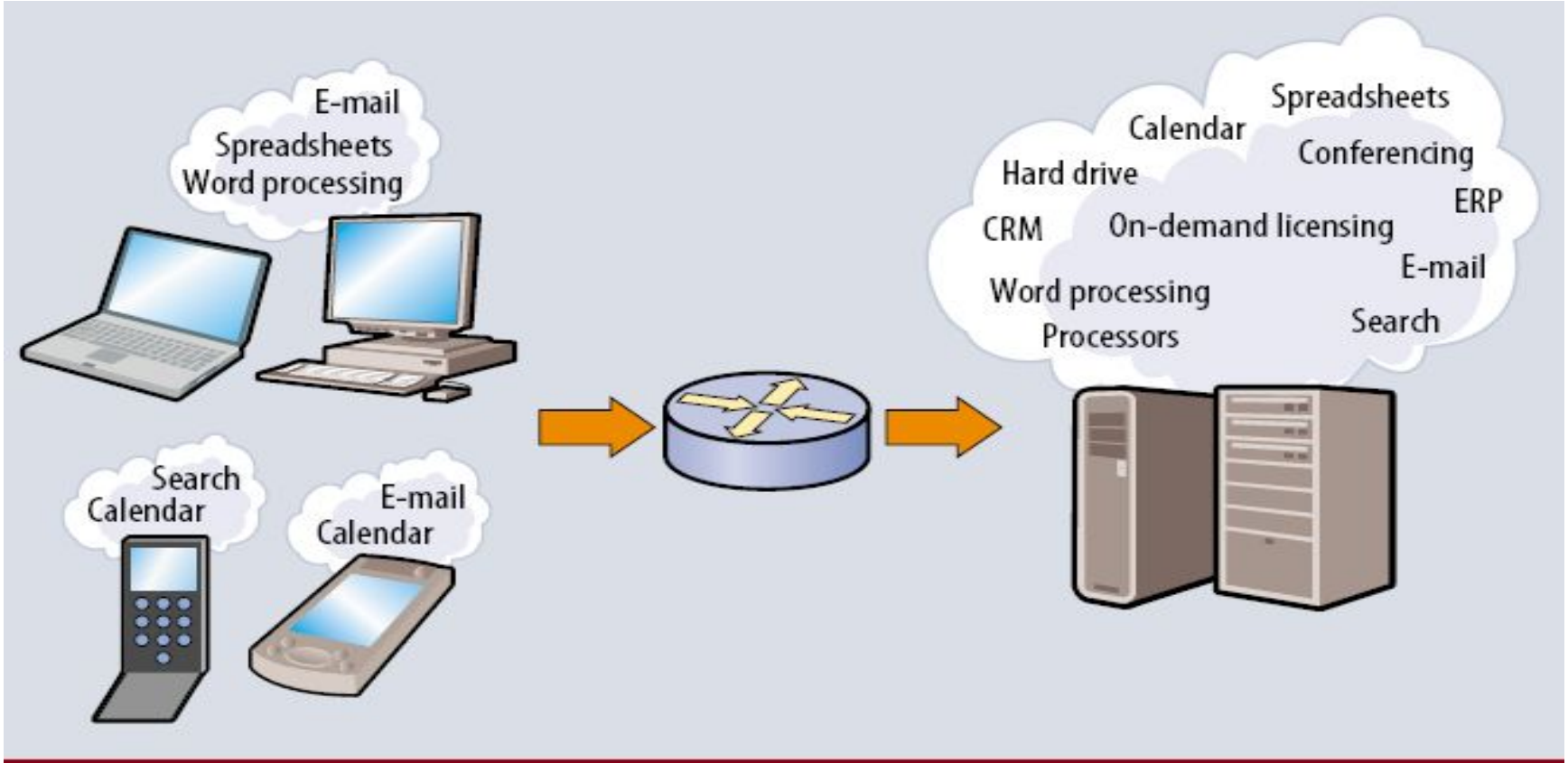
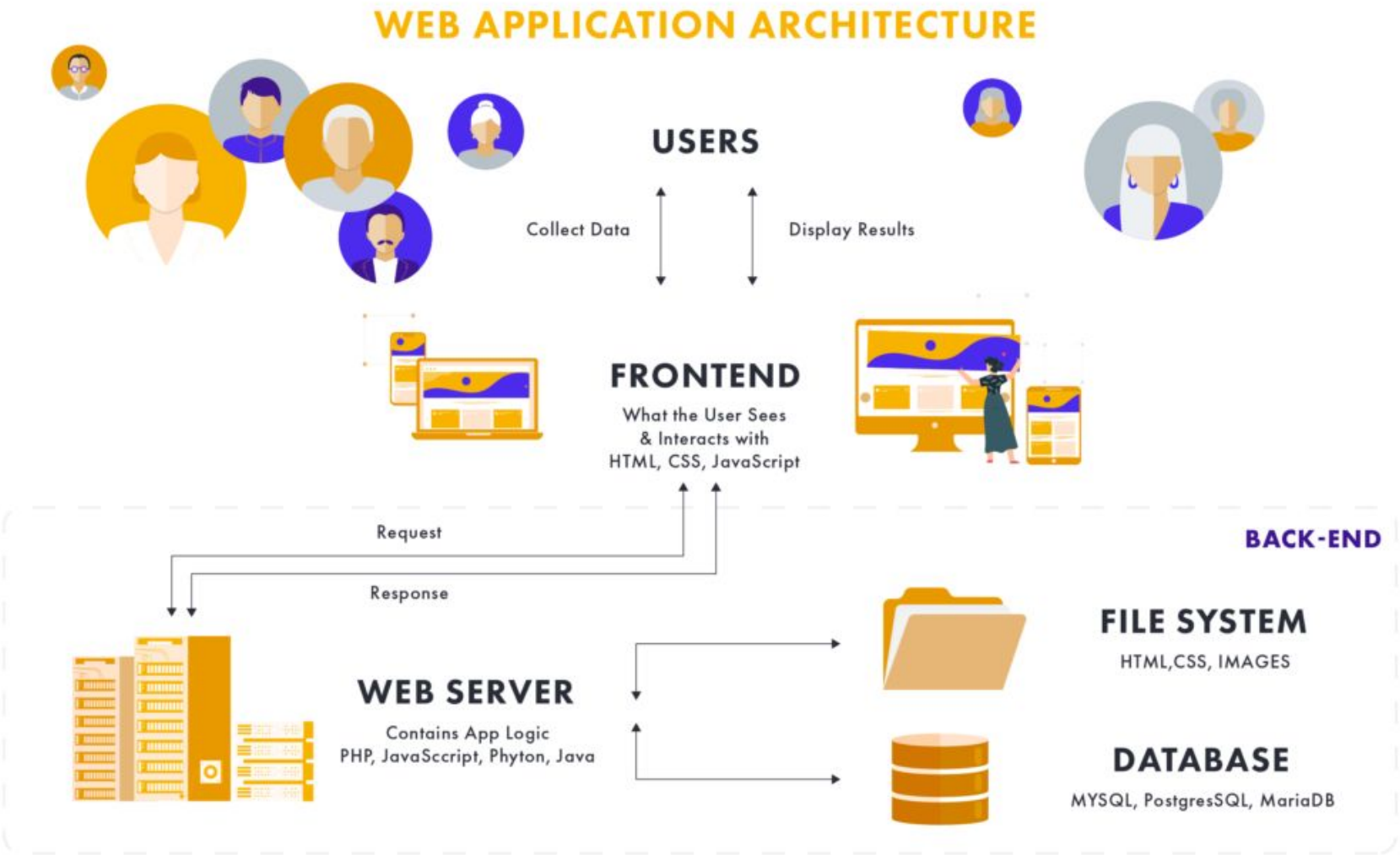


Figure 1. Applications are moving from the local machine to the Ubiquitous Computer.

The Web as the Ubiquitous Computer, IEEE Computer 2009, V Pendyala, S. Shim

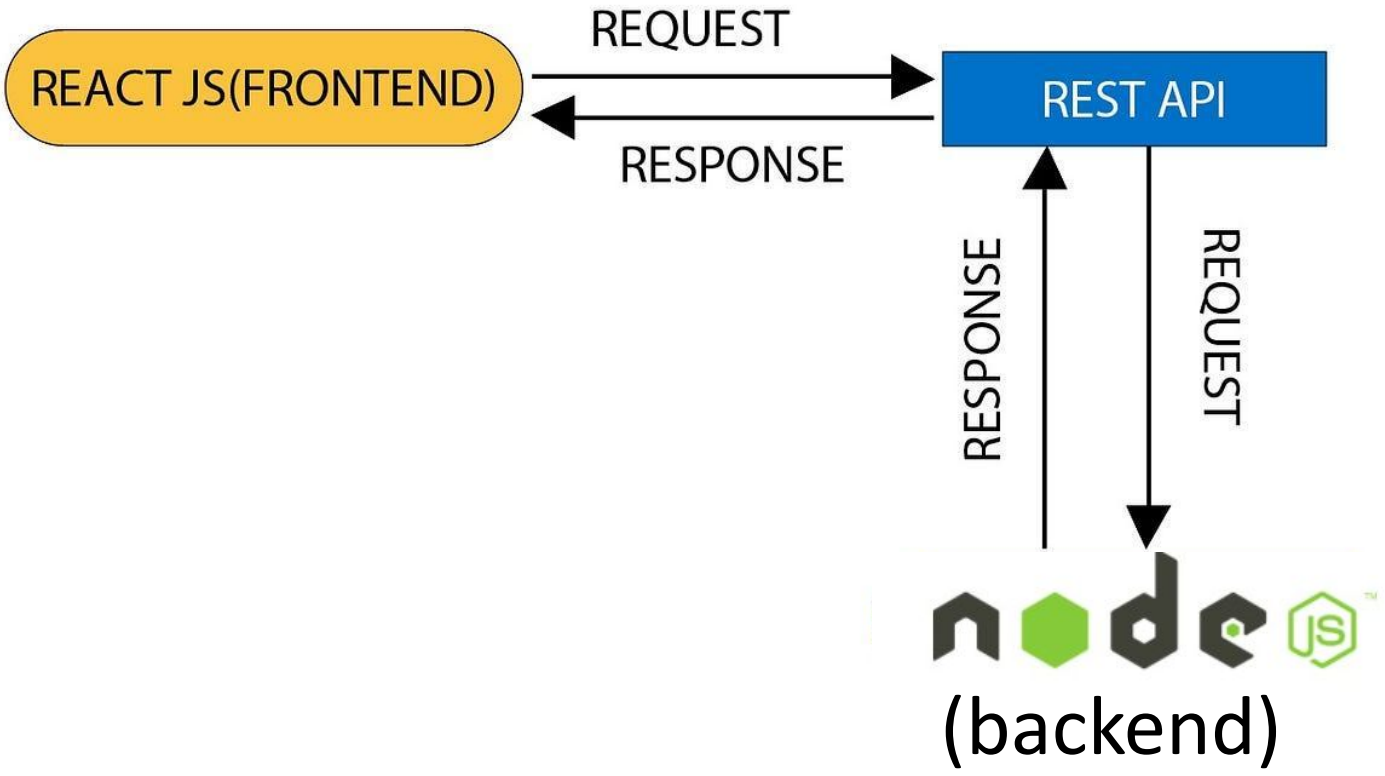
- Grade is not $\propto$ knowledge
- knowledge $\propto$ struggle
- knowledge $\propto$ Time spent
- Success $\propto$ knowledge

WEB Application Architecture



https://www.evertop.pl/en/frontend-vs-backend/

WEB Application Architecture



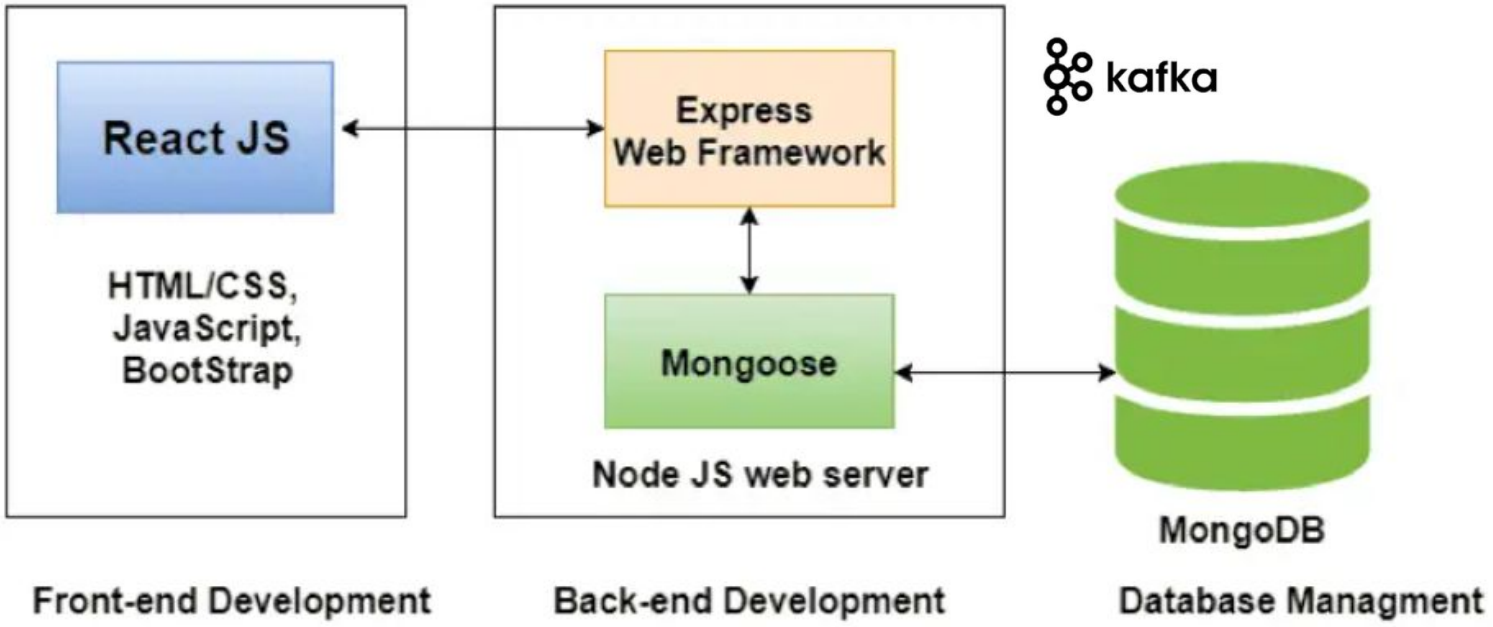
MERN Stack

- MongoDB(M)
- Express(E)
- React(R)
- NodeJS(N)

- Distributed using Kafka message

MERN Stack + Kafka

- MongoDB, Node.js and Express are dedicated to developing the back-end of web applications. This corresponds to database management, scripts, html documents, HTTP requests, etc.



Frontend Technology

Introduction to web technologies:

- HTML to create the document structure and content
- CSS to control is visual aspect
- Javascript for interactivity

Rationale for the Dual Approach

- Our backend architecture integrates both FastAPI (Python) and Node.js. We adopt this dual approach because Node.js offers smoother and more straightforward implementations for these components. Conversely, Python excels in AI-related workloads, making it ideal for our intelligent service layer.
- FastAPI for I/O-bound tasks and high performance: FastAPI excels at handling I/O-bound operations due to its asynchronous capabilities and efficient request handling, making it a good fit for processing requests and interacting with databases like MongoDB.
- Node.js for MongoDB and Kafka integration: Node.js, with its event-driven, non-blocking I/O model and a rich ecosystem (npm), simplifies working with Kafka and MongoDB in real-time or high-throughput scenarios.
    - Kafka: Node.js's event-driven architecture naturally aligns with Kafka's event-streaming capabilities, facilitating real-time data processing and building event-driven microservices architectures.
    - MongoDB: Node.js's efficiency in handling concurrent connections is well-suited for interacting with MongoDB, especially when dealing with high volumes of data or real-time updates.
- Microservices architecture: This dual-language approach inherently lends itself to a microservices architecture, where each service can be built with the most appropriate

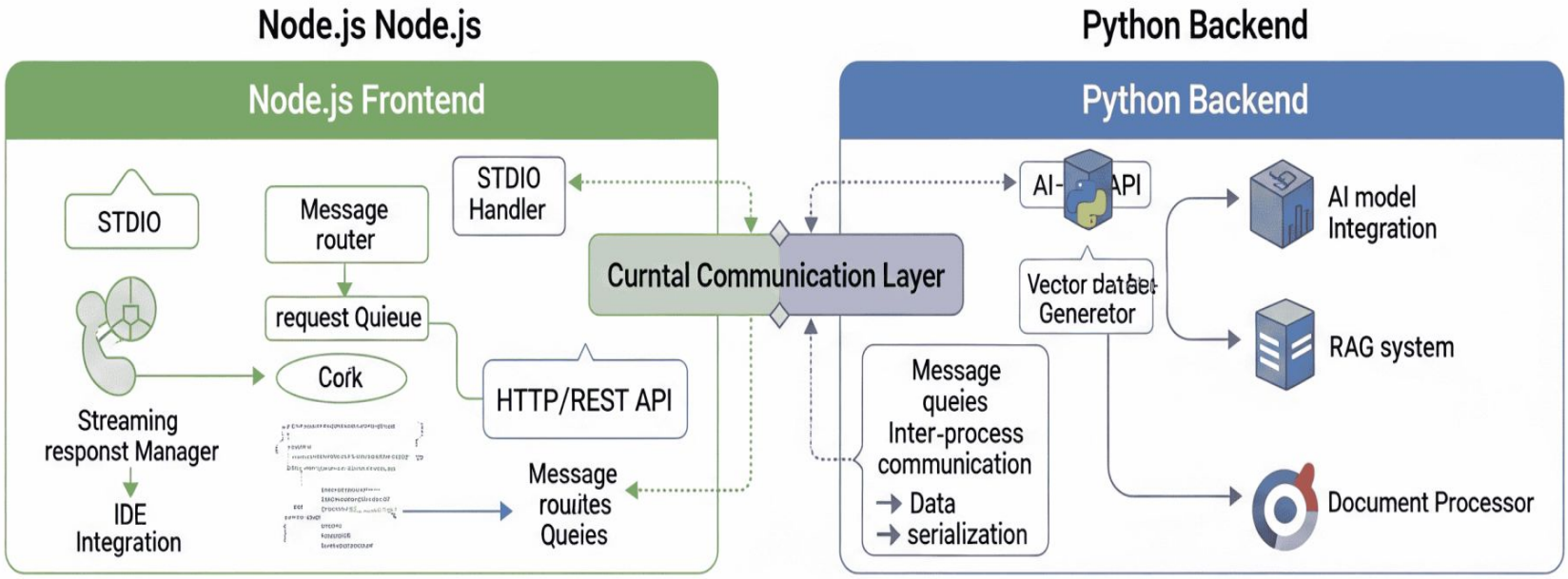
FastAPI agent framework

- In essence, combining FastAPI and Node.js allows you to leverage the best of both worlds, enabling you to build a powerful and efficient distributed system that handles data processing and real-time operations effectively
- We will implement our **Agent AI** capabilities using the FastAPI Agents framework, enabling the seamless creation and management of intelligent agents via REST. This design allows any Node.js components to easily call Python REST services, ensuring tight interoperability between the two backends

Building Agent Endpoints

- Expose Agent Functionality via RESTful APIs: Define API endpoints using FastAPI's decorators (@app.post, @app.get, etc.) that correspond to specific agent actions
- Request Validation with Pydantic: Use Pydantic models to define the expected structure of incoming requests (input data for your agent) to ensure data integrity.
- Handle Agent Responses: Define the structure of the API responses (outputs from your agent) using Pydantic models, enabling consistency and predictable data exchange.
- Asynchronous Operations: Leverage FastAPI's async/await features to handle

Hybrid System Nostem Merging



Deployment Options

Docker Container setup

Nunito

Docker Container stup

Kubernetes

→ Process Managemnent

→ Process Management

Scaling Strategies

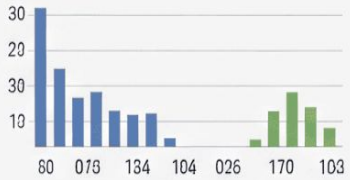
40% Latency

45% Faster boarticy

60% Faster onboarding

35% API bugs developers

Performance



Over 30 conuɛinat developers

Tools

What do we need to start:

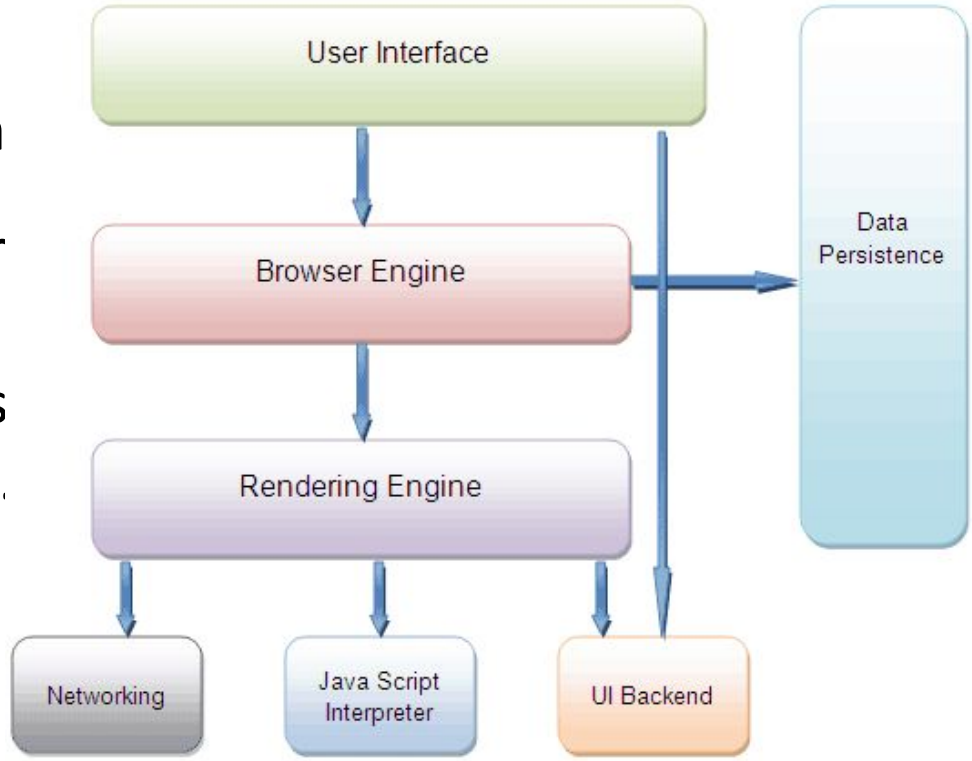
- a good web-browser (Chrome or Firefox)
- a good text editor like:
  - Notepad++ (win)
  - VSCode (cross platform)
  - textWrangler (osx)
  - sublime text (cross platform)

Inside a browser

Browsers have parts.

We are interested in two of them

- the Rendering Engine (in char
  in a visual image).
- The Javascript Interpreter (als
  executing the Javascript code



Frontend Technologies

- HTML
- CSS
- Javascript

Browsers as a renderer

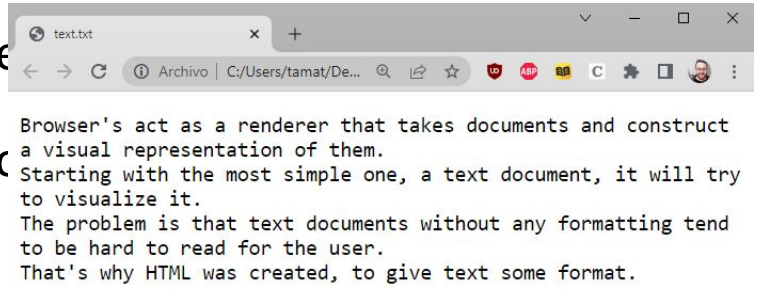
Browser's act as a renderer that takes documents and construct a visual representation of them.

Starting with the most simple one, a text docume

You can try, drop any .txt file into your browser to

The problem is that text documents without any
read for the user (and quite boring).

That's why HTML was created, to give text some

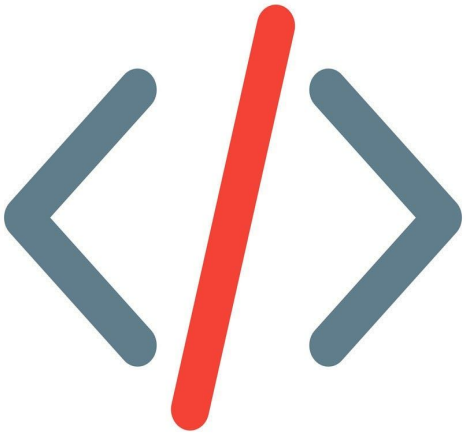


Markup language

There are many markup languages that add special tags into the text that the renderer wont show but use to know how to dis

In HTML this tags use the next no

```
My name is <b>Javi</b>
```



HTML

HTML means Hyper Text Markup Lan

The HTML allow us to define the stru
document or a website.

HTML is to give structure to the cont
website, not to define an algorithm.

it is a subset of XML

```
<html>
    <head>
    </head>
    <body>
        <div>
            <p>Hi</p>
        </div>
    </body>
</html>
```

HTML: basic rules

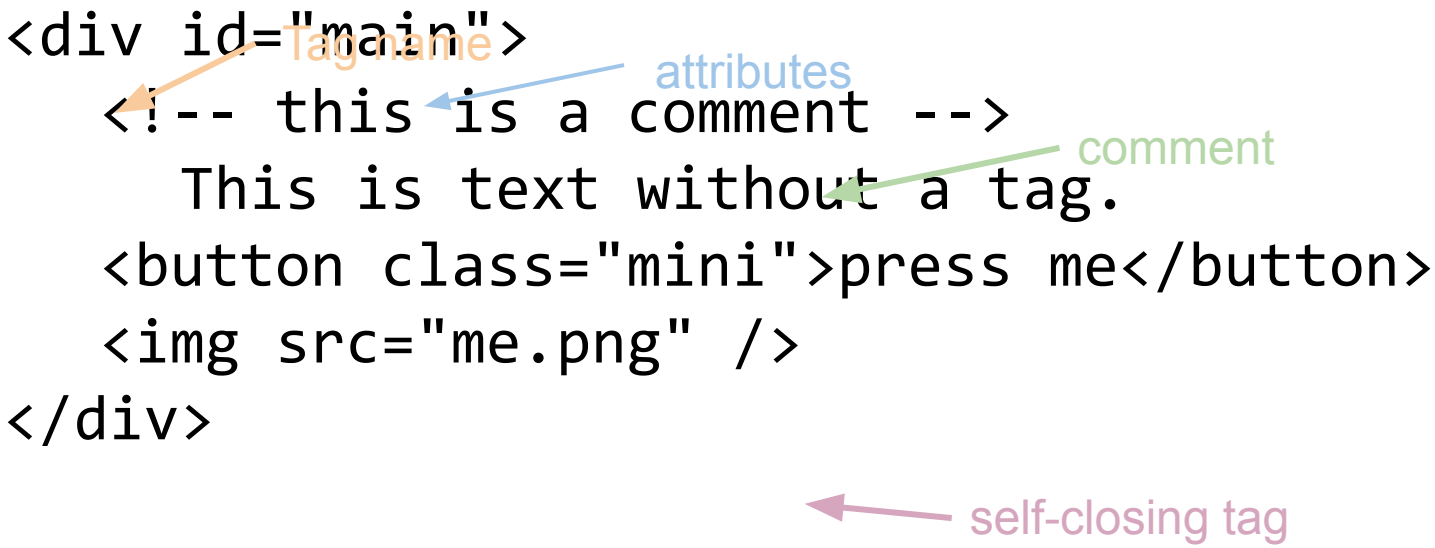
Some rules about HTML:

- It uses XML syntax (tags with attributes, can contain other tags).

```
<tag_name attribute="value"> content </tag_name>
```

- There are different HTML elements for different types of information and behaviour.
- The information is stored in a tree-like structure (nodes that contain nodes inside) called DOM (Document Object Model).
- It gives the document some semantic structure (pe. this is a title, this is a section, this is a form) which is helpful for computers to understand websites content.
- It must not contain information related to how it should be displayed (that information belongs to the CSS), so no color information, font size, position, etc.

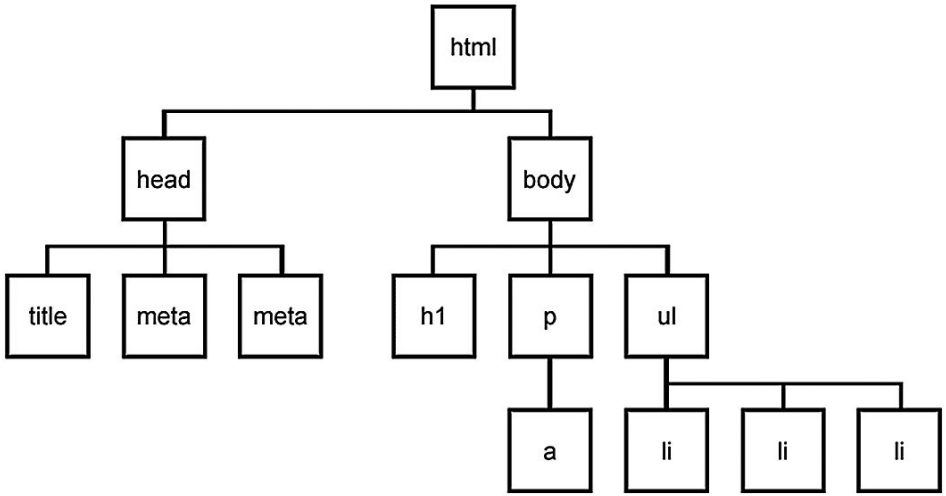
HTML: syntax example

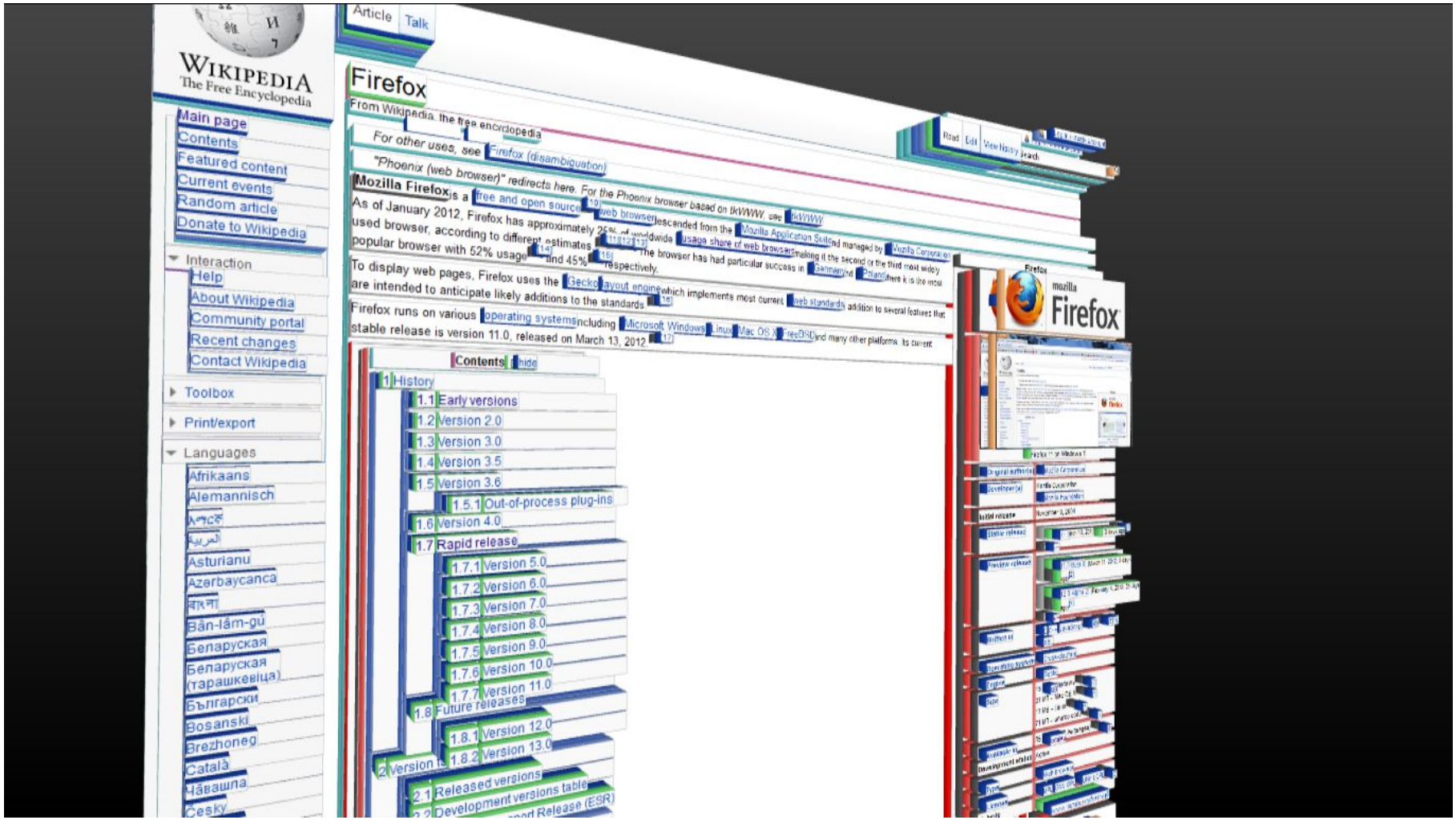



```
<div id="main">
   <!-- this is a comment -->
     This is text without a tag.
   <button class="mini">press me</button>
   <img src="me.png" />
</div>
```

DOM is a tree

Every node can only have one parent, and every node can have several
looks like a tree.




Wikipedia
The Free Encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
Afrikaans
Alemannisch
Asturianu
Azərbaycanca
Bân-lâm-gú
Беларуская
Беларуская (тарашкевіца)
Български
Bosanski
Brezhoneg
Català
Чăвашла
Česky
Article
Talk
Firefox
From Wikipedia, the free encyclopedia
For other uses, see Firefox (disambiguation)
"Phoenix (web browser)" redirects here. For the Phoenix browser based on tkWWW, see tkWWW
Mozilla Firefox is a free and open source web browser descended from the Mozilla Application Suite and managed by Mozilla Corporation.
As of January 2012, Firefox has approximately 25% of worldwide usage share of web browsers, making it the second or the third most widely used browser, according to different estimates. The browser has had particular success in Germany and Poland where it is the most popular browser with 52% usage and 45% respectively.
To display web pages, Firefox uses the Gecko layout engine which implements most current web standards addition to several features that are intended to anticipate likely additions to the standards.
Firefox runs on various operating systems including Microsoft Windows, Linux, Mac OS X, FreeBSD and many other platforms. Its current stable release is version 11.0, released on March 13, 2012.
Contents hide
1 History
1.1 Early versions
1.2 Version 2.0
1.3 Version 3.0
1.4 Version 3.5
1.5 Version 3.6
1.5.1 Out-of-process plug-ins
1.6 Version 4.0
1.7 Rapid release
1.7.1 Version 5.0
1.7.2 Version 6.0
1.7.3 Version 7.0
1.7.4 Version 8.0
1.7.5 Version 9.0
1.7.6 Version 10.0
1.7.7 Version 11.0
1.8 Future releases
1.8.1 Version 12.0
1.8.2 Version 13.0
2 Version history
2.1 Released versions
2.2 Development versions table
Read
Edit
View history
Search
mozilla
Firefox

HTML: main tags

Although there are lots of tags in the HTML specification, 99% of the webs use a subset of HTML tags with less that 10 tags, the most important are:

- : a container, usually represents a rectangular area with information inside.
- <img/>: an image
- <a>: a clickable link to go to another URL
- <p>: a text paragraph
- <h1>: a title (h2,h3,h4 are titles of less importance)
- <input>: a widget to let the user introduce information
- <style> and <link>: to insert CSS rules
- <script>: to execute Javascript
- : a null tag (doesn't do anything), good for tagging info

HTML: other interesting tags

There are some tags that could be useful sometimes:

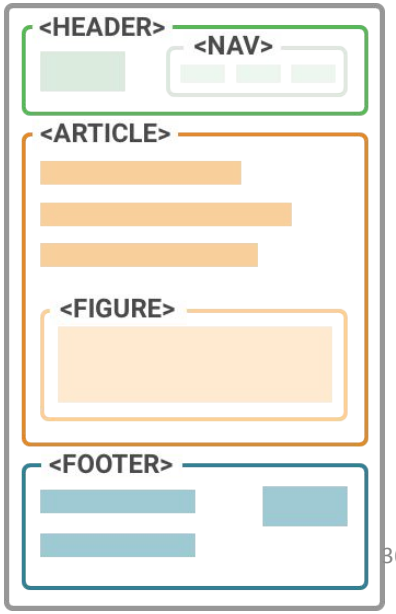
- <button>: to create a button
- <audio>: for playing audio
- <video>: to play video
- <canvas>: to draw graphics from javascript
- <iframe>: to put another website inside ours

HTML: wrapping the info

We use HTML tags to wrap different information on our site.

The more structure has the information, the easier will be to access it and present it.

We can change the way the information is represented on the screen depending on the tags where it is contained, so we shouldn't be worried about using too many tags.



How to learn HTML/CSS

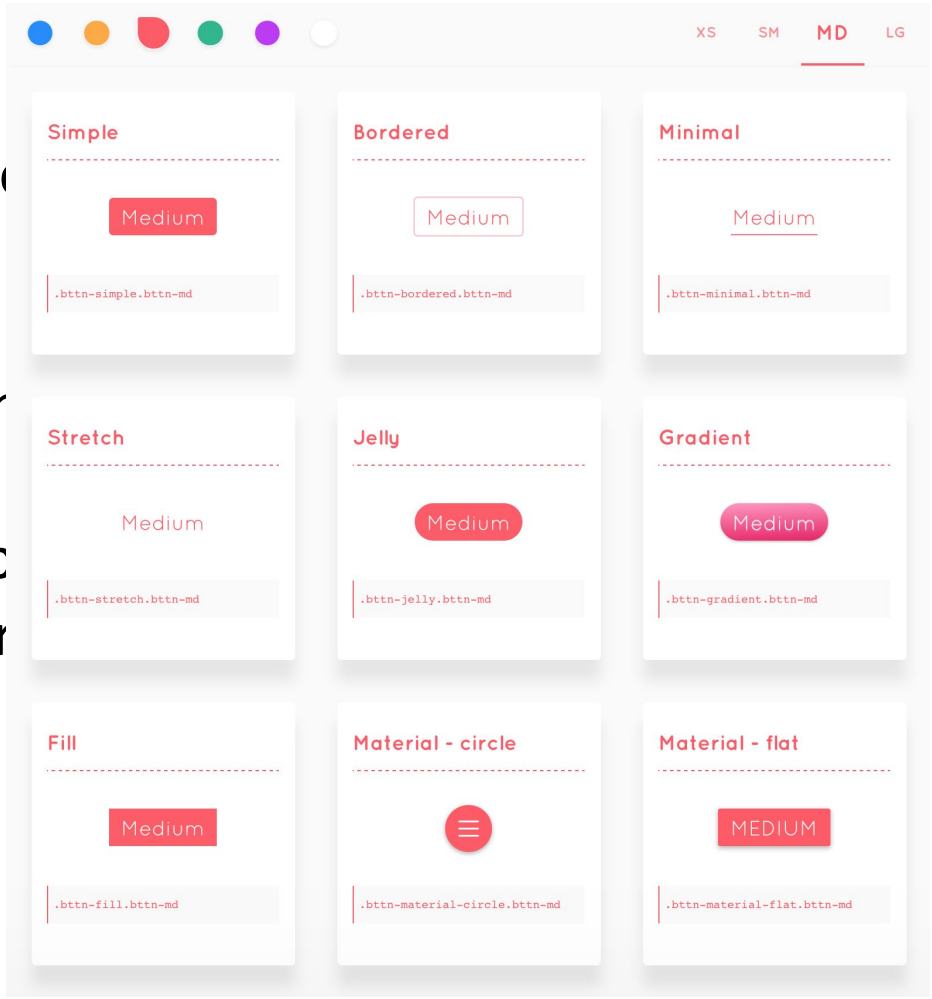
- https://www.w3schools.com/
- HTML Reference: a description of all HTML tags.
- The 25 Most used tags: a list with information of the more common tags.
- HTML5 Good practices: some tips for starters

CSS

CSS allows us to specify how to pr
stored in the HTML.

Thanks to CSS we can control all th
and some other features:

- Colors: content, background, b
- Margins: interior margin, exter
- Position: where to put it
- Sizes: width, height
- Behaviour: changes on mouse



CSS example

```
* {
   color: blue; /*a comment */
   margin: 10px;
   font: 14px Tahoma;
}
```

This will change all the tags in my web (‘*‘ means all) to look blue with font Tahoma with 14px, and leaving a margin of 10px around.

CSS fields

Here is a list of the most common CSS fields and an example:

- `color: #FF0000;  red;  rgba(255,00,100,1.0); //different ways to specify colors`
- `background-color: red;`
- `background-image: url('file.png');`
- `font: 18px 'Tahoma';`
- `border: 2px solid black;`
- `border-top: 2px solid red;`
- `border-radius: 2px; //to remove corners and make them more round`
- `margin: 10px; //distance from the border to the outer elements`
- `padding: 2px; //distance from the border to the inner elements`
- `width: 100%;  300px;  1.3em;  //many different ways to specify distances`
- `height: 200px;`
- `text-align: center;`
- `box-shadow: 3px 3px 5px black;`
- `cursor: pointer;`
- `display: inline-block;`
- `overflow: hidden;`

CSS how to add it

There are four ways to add CSS rules to your website:

- Inserting the code inside a style tag
```
<style>
   p { color: blue }
</style>
```
- Referencing an external CSS file
```
<link href="style.css" rel="stylesheet" />
```
- Using the attribute style on a tag
```
<p style="color: blue; margin: 10px">
```
- Using Javascript (we will see this one later).

CSS selectors

Let's start by changing the background color of one tag of our website:

```
div {
    background-color: red;
}
```

This CSS rule means that every tag DIV found in our website should have a red background color. Remember that DIVs are used mostly to represent areas of our website.

We could also change the whole website background by affecting the tag body:

```
body {
    background-color: red;
}
```

CSS further reading

- https://www.w3schools.com/
- One line layouts tutorials
- Understanding the Box Model: a good explanation of how to position the information on your document.
- All CSS Selectors: the CSS selectors specification page.
- CSS Transition: how to make animations just using CSS
- TailwindCSS: a CSS Framework

What is JavaScript?

- Created by Netscape, allows interactivity, browser manipulation, and now standard exists
- A scripting language
  - In the case of JavaScript, the client is the browser.
  - A server-side language is one that runs on the Web server. Examples: PHP, Python
- Interpreted on-the-fly by the client
  - Each line is processed as it loads in the browser
  - vs. Java - programs are compiled and can be run as stand alone applications

How does it work?

- Embedded within HTML page
- Simple programming statements combined with HTML tags
- Interpreted

Learning JavaScript

- Special syntax to learn
- Learn the basics and other samples
- Write it in IDE, view results in browser

Including JavaScript in HTML

- Two ways to add JavaScript to Web pages
  - Use the <script>…</script> tag
  - Include the script in an external file -- more about this later in the semester
- Initially, we will only use the <script>…</script> tag

JavaScript keywords

```
break    case    catch   continue debugger
default delete    do      else   finally
for   function if   in          instanceof
new   return    switch this    throw
try   typeof    var void       while
with
```

- Reserved words

```
class    const   enum    export    extends
import   implements interface let package
private protected public static super yield
```

JavaScript Tutorial

W3school: https://www.w3schools.com/js/

The Modern JavaScript Tutorial: https://javascript.info/

JavaScript tutorial: https://www.javascripttutorial.net/

Node.js

- Node.js is **‘server-side JavaScript’**.
- Node.js is a high-performance **network applications framework**, well optimized for high-concurrency environments.
- It’s a **command line** tool.
- In ‘Node.js’ , ‘**.js’** doesn’t mean that its solely written JavaScript. It is 40% JS and 60% C++.
- From the official site:
  - ***‘Node's goal is to provide an easy way to build scalable network programs.’ -*** (from nodejs.org)

Node.js

- Node.js uses an **event-driven**, **non-blocking I/O** model, which makes it lightweight. (from nodejs.org)
- It makes use of **event-loops** via JavaScript's **callback** functionality to implement the non-blocking I/O.
- Programs for Node.js are written in JavaScript but not in the same JavaScript we are used to. There is no DOM implementation provided by Node.js,
- Everything inside Node.js runs in a **single-thread**.

Who is using Node.js in production?

1. LinkedIn
2. Netflix
3. Uber
4. Trello
5. PayPal
6. NASA
7. eBay
8. Medium
9. Groupon
10. Walmart
11. Mozilla
12. GoDaddy

According to the information available on the Node.js official website and other sources, companies like Netflix, PayPal, Trello, Uber, Walmart, LinkedIn, NASA, eBay, GoDaddy, and Medium are prominent users of Node.js, utilizing its capabilities for backend development and real-time applications.

A partial list can be found at:

https://www.simform.com/blog/companies-using-nodejs/#:~:text=Here%20are%20the%20companies%20using,Last%20Updated%20January%2020%2C%202025

Node.js further reading

- Visit www.nodejs.org for Info/News about Node.js

- https://www.w3schools.com/
- widely used Node.js modules at: http://blog.nodejitsu.com/top-node-module-creators
- Watch Node.js tutorials @ http://nodetuts.com/
- For Info on MongoDB: http://www.mongodb.org/display/DOCS/Home

Other backend framework

- Spring(java) at https://spring.io/, spring boot at https://spring.io/projects/spring-boot
- Django(python) at https://www.djangoproject.com/
- Symfony(php) at https://symfony.com/

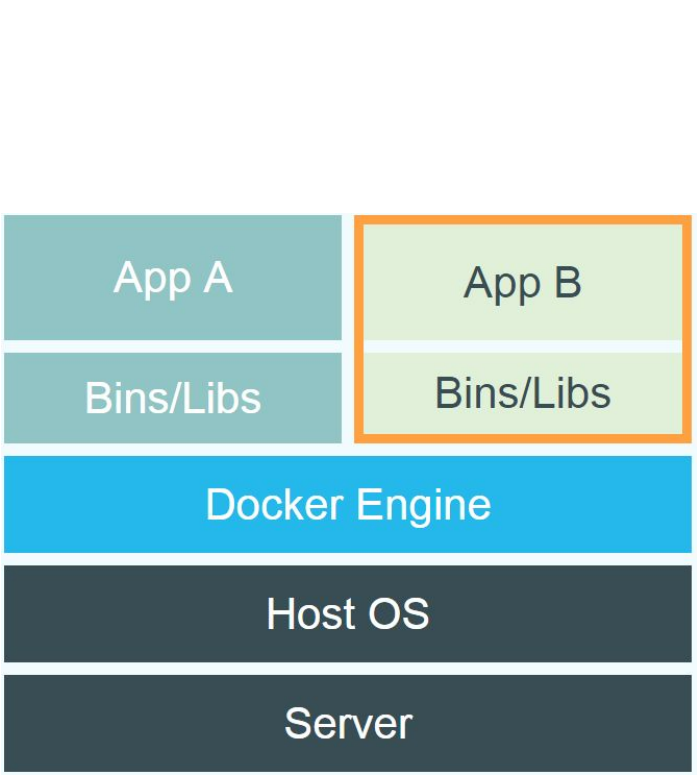
What is Docker?

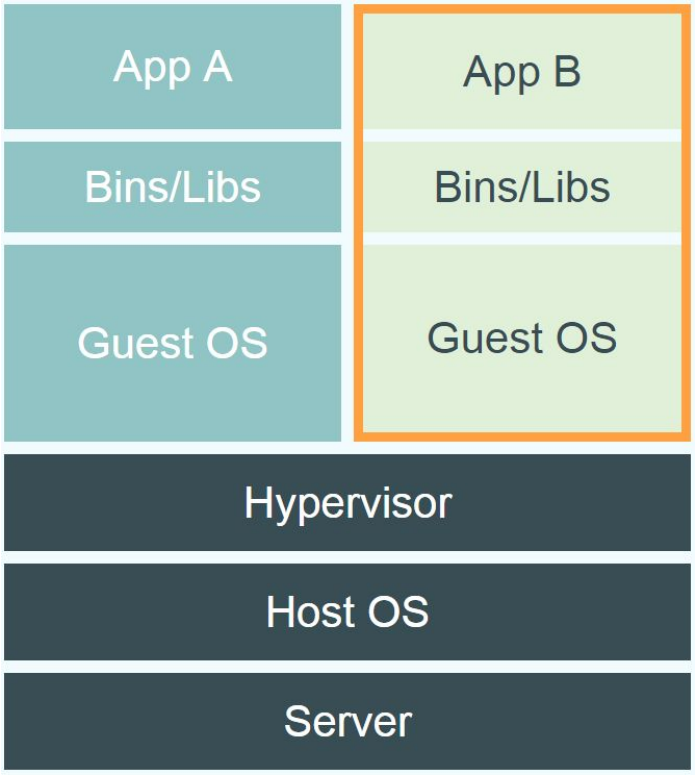
- **Docker** is an open platform for **developing, shipping,** and **running** applications.
- **Docker** is designed to deliver your applications faster.
- Docker you can separate your applications from your **infrastructure** and treat your infrastructure like a managed application

Docker

- Provide a uniformed wrapper around a software package: *«Build, Ship and Run Any App, Anywhere»* [www.docker.com]
  - Similar to shipping containers: The container is always the same, regardless of the contents and thus fits on all trucks, cranes, ships, ...

Docker vs. Virtual Machine

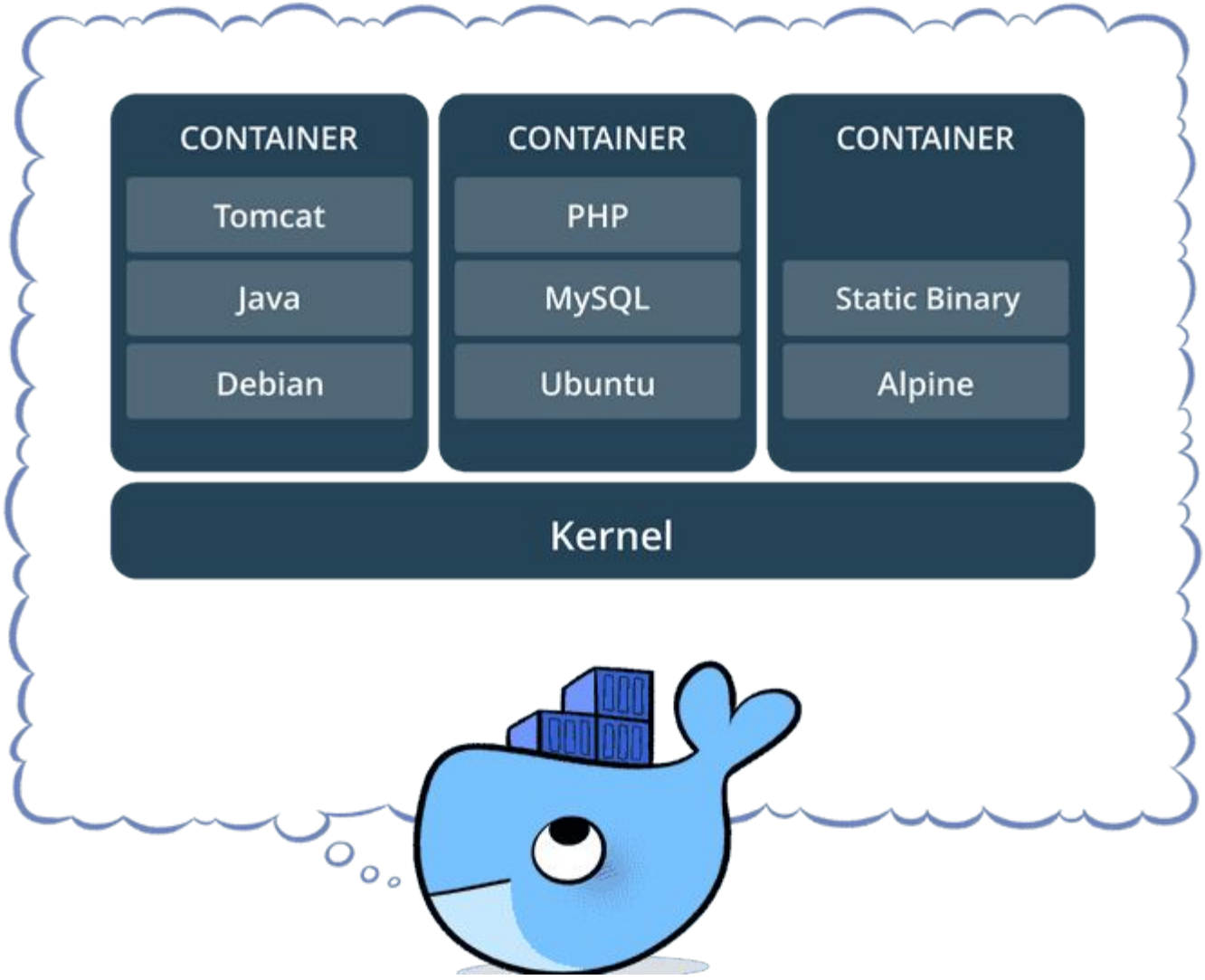




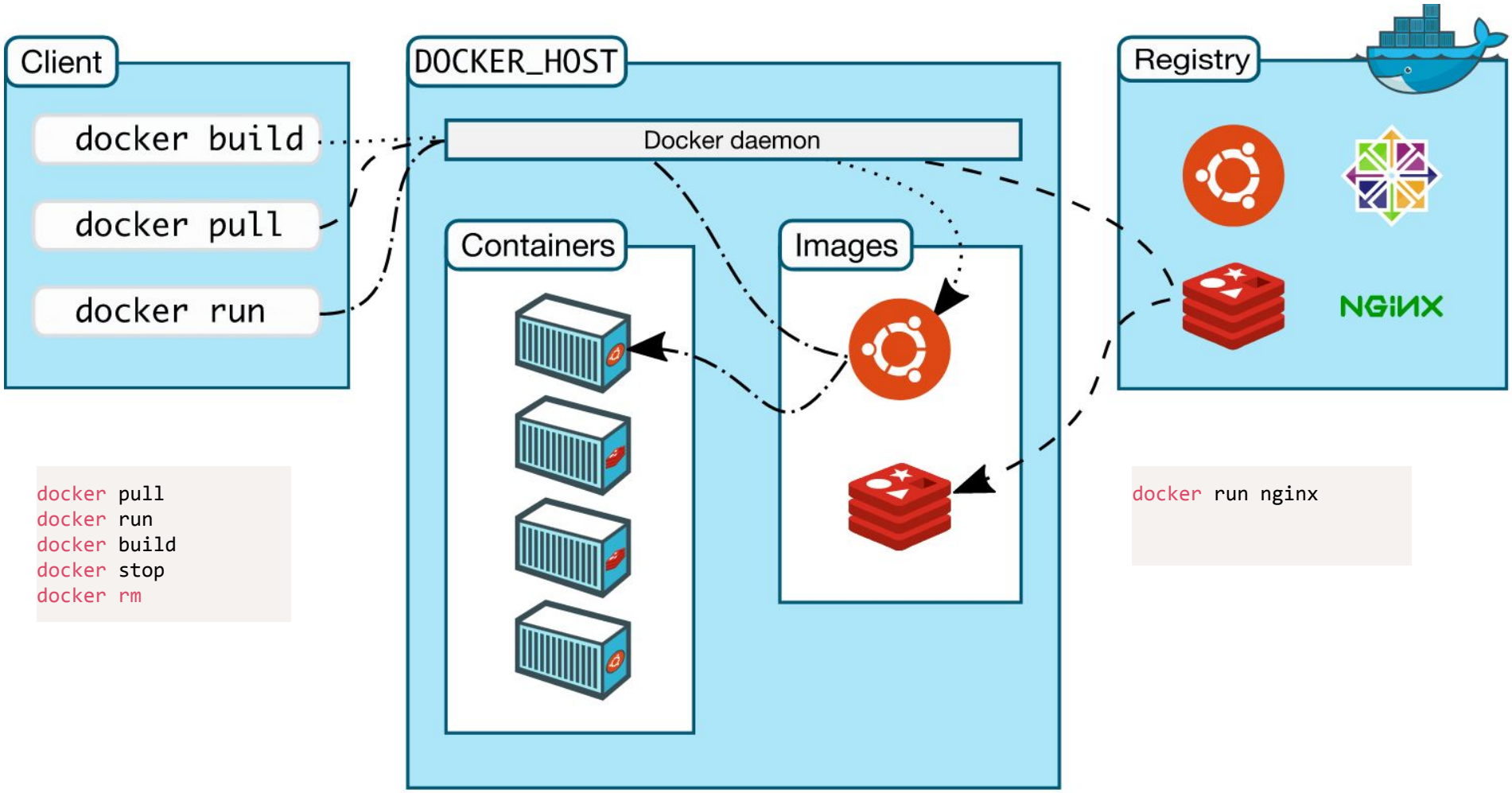
Docker Features

- fast and elegant isolation framework
- Inexpensive
- Low CPU/memory overhead
- Fast boot/shutdown
- Cross cloud infrastructure

Docker container



Docker Architecture

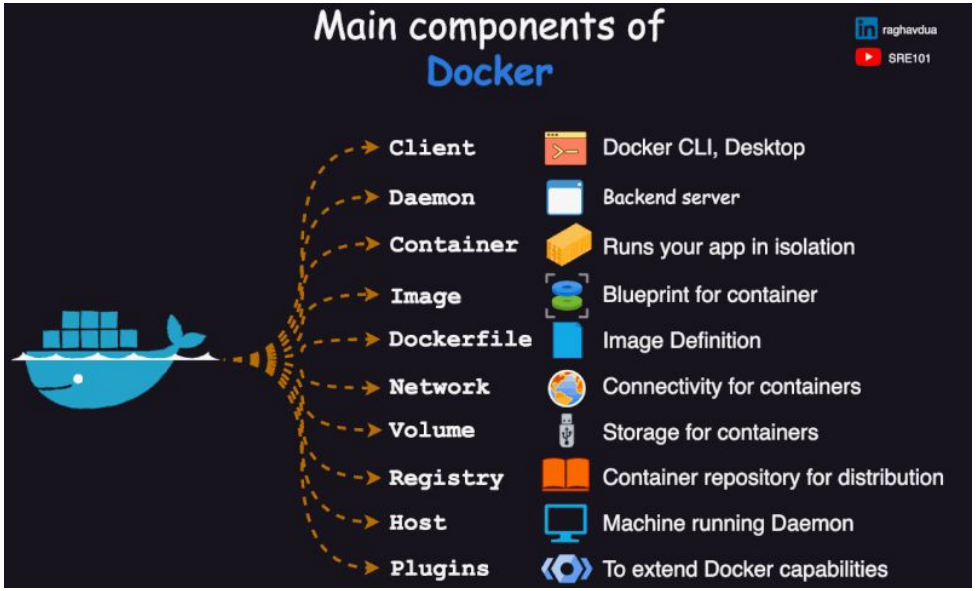



```
docker pull
docker run
docker build
docker stop
docker rm
```

```
docker run nginx
```

Docker Components

- Docker Client
- Docker Daemon
- Docker Registry
- Docker Containers
- Docker Images
- Dockerfile



Run Applications

- **Step 1:** Build an image.
- **Step 2:** Run the container.
- Docker Image is a read-only template to build containers. An image holds all the information needed to bootstrap a container, including what processes to run and the configuration data. Every image starts from a base image, and a template is created by using the instructions that are stored in the DockerFile. For each instruction, a new layer is created on the image.
- Running the container originates from the image we created in the previous step. When a container is launched, a read-write layer is added to the top of the image. After appropriate network and IP address allocation, the desired application can now be run inside the container.

Docker Use Cases

- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment (dev test prod (local, VM, cloud, ...)

Welcome to Agentic AI

What agents are, why they matter now, how they work, and how to create them.

What is an Agent?

An **Agent** is a system designed to achieve goals through an iterative cycle of reasoning and interaction with its environment.

Core Process

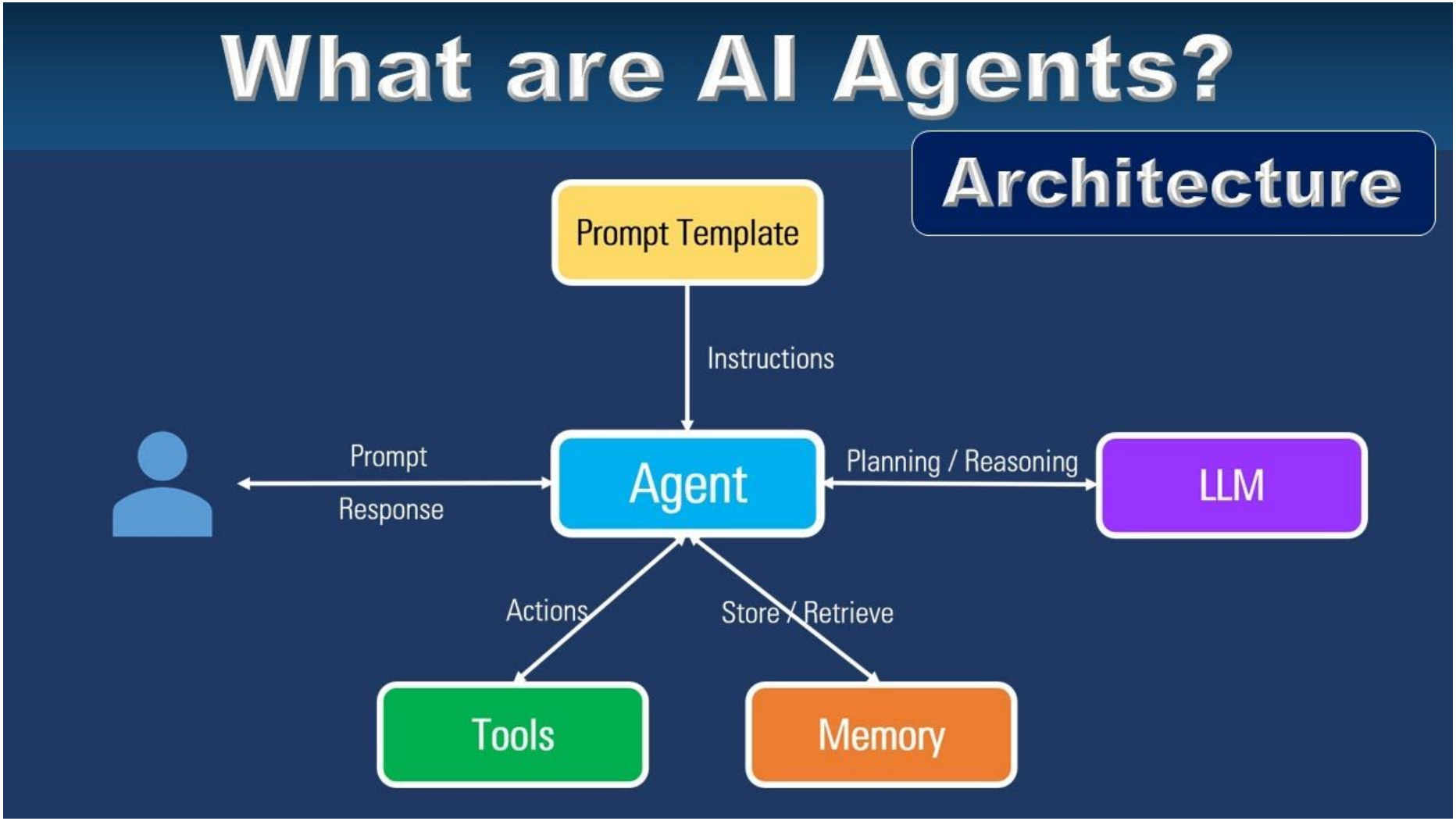
1. **Plan** → Formulate actions toward a goal.
2. **Act** → Execute actions, often by using external tools or functions.
3. **Observe** → Perceive feedback or results from the environment.
4. **Reflect** → Update internal state and strategy based on outcomes.

This loop repeats until the goal is reached (or the task terminates).

Key Components

- **LLM "Brain"** – Provides reasoning, planning, and language understanding.
- **Tools / Function Calls** – Extend capabilities (e.g., search, API calls, computation).
- **Memory / State** – Stores context, history, and learned knowledge for continuity.
- **Control Loop** – Governs the cycle of planning, acting, observing, and reflecting.

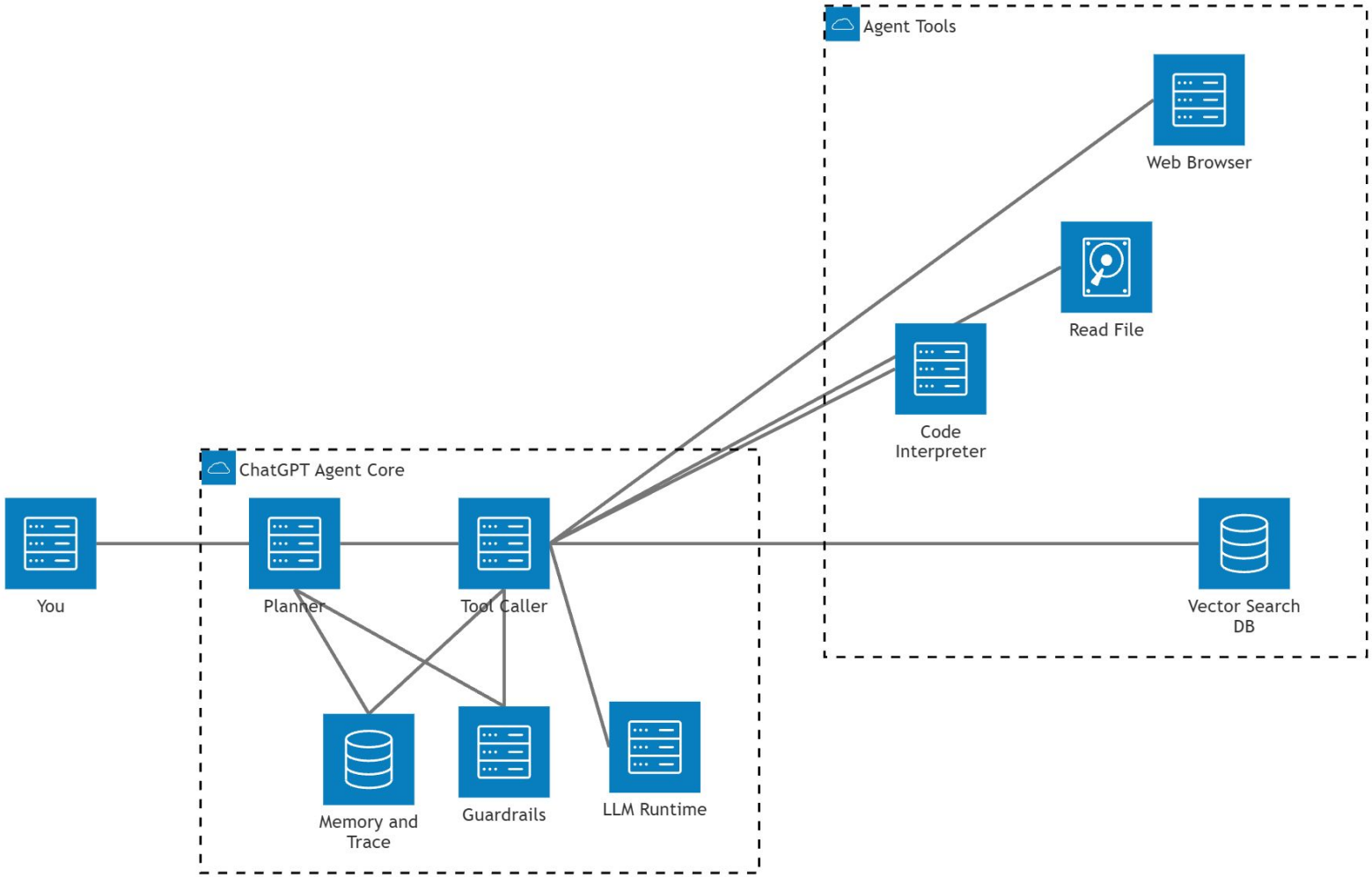
What are AI Agents?



How agents work

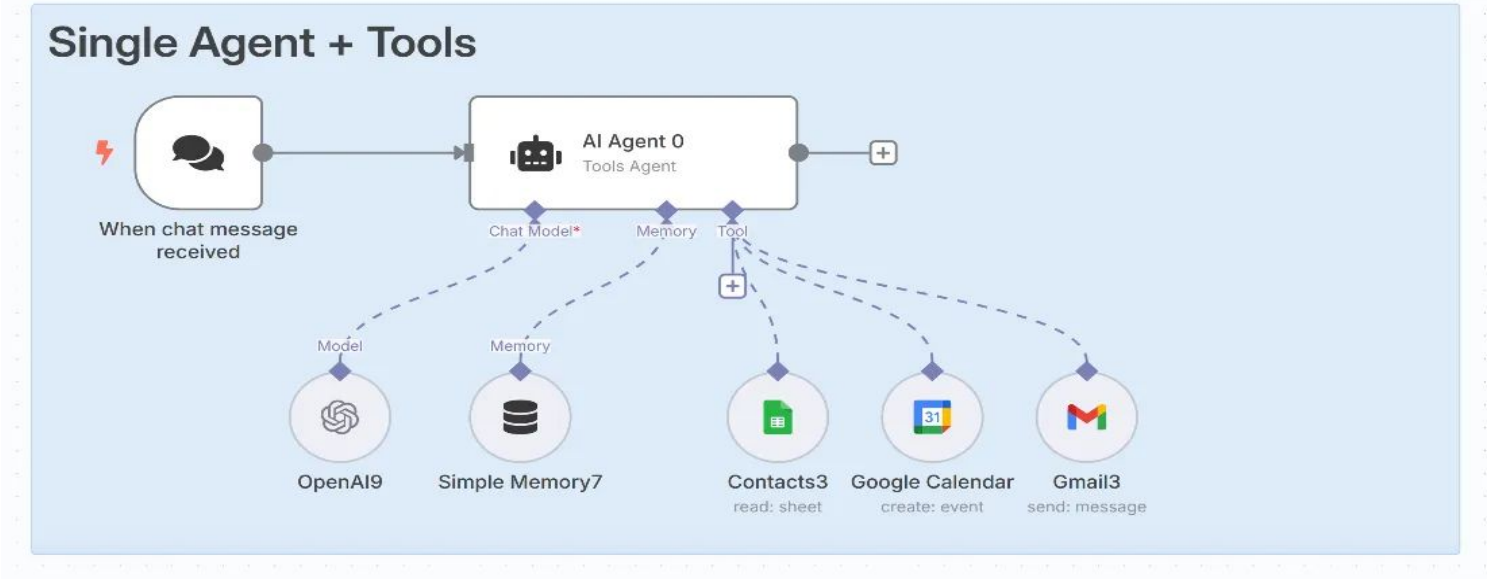
- **Loop:** Plan → Select tool → Call tool → Observe → Reflect / Update plan → Stop
- **Brain:** LLM/model drives reasoning and selects next steps, Generates tool calls with arguments
- **Tools:** Defined with strict I/O schemas, Reduce errors and hallucinations
- **Memory:** short-term trace + optional long-term vectors
- **Stop conditions:** success/timeout/iteration limits
- *Course reading:* **Introduction to smolagents**

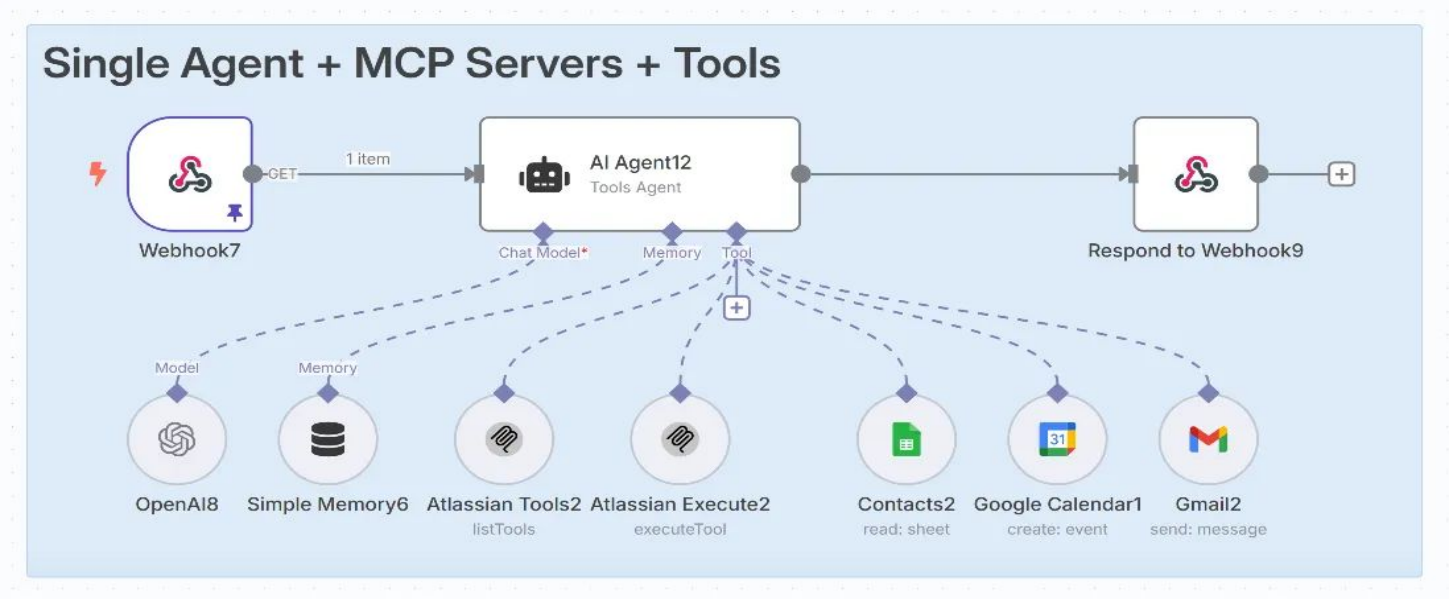
https://huggingface.co/learn/agents-course/en/unit2/smolagents/introduction


Agent Tools
Web Browser
Read File
Code
Interpreter
ChatGPT Agent Core
You
Planner
Tool Caller
Vector Search
DB
Memory and
Trace
Guardrails
LLM Runtime

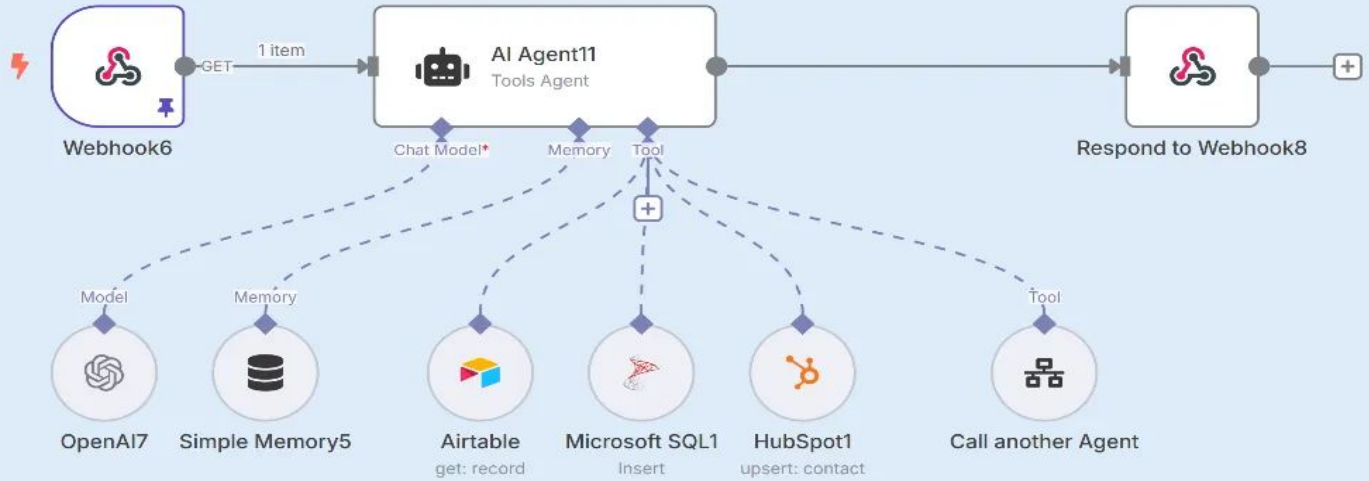
AI Agent Architectures:

https://www.productcompass.pm/p/ai-agent-architectures

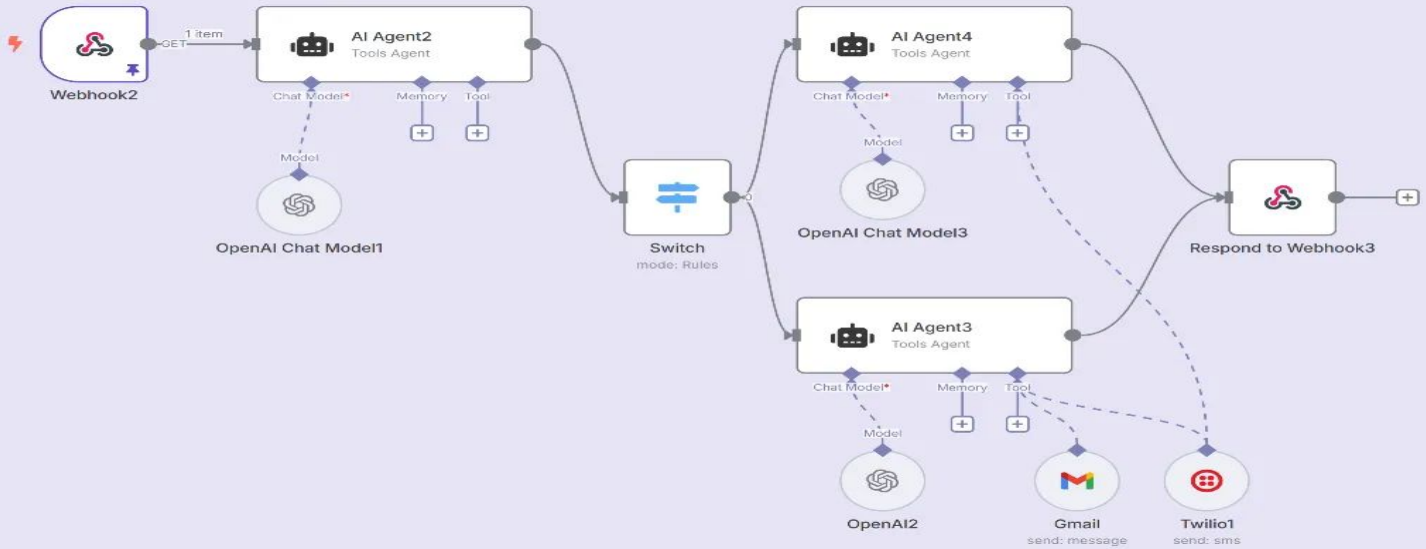




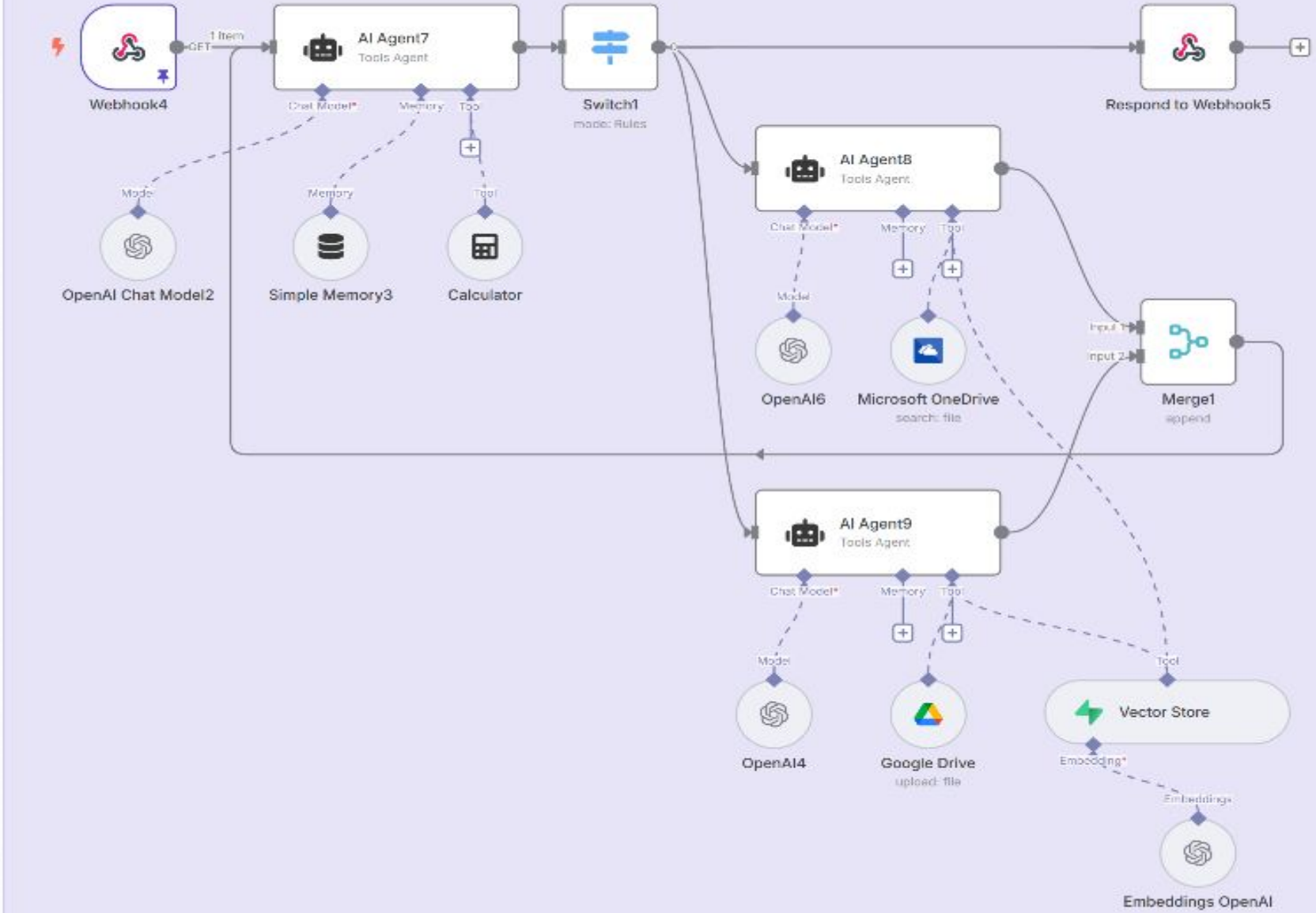
Single Agent + Dynamically Call Other Agents



Agents Hierarchy + Shared Tools + Parallel



Agents Hierarchy + Loop + Parallel + Shared RAG



Tools and functions (schemas matter)

- Tools are **typed functions** (name, inputs, outputs) exposed to the model
  Name – identifier for the model (e.g., search_web)
  Inputs – required arguments (e.g., {query: string})
  Outputs – expected return values (e.g., {results: list of strings})
- Clear contracts ⇒ higher reliability, safer actions, easier
  Example: A calculator tool guarantees numeric inputs/outputs only.
- Start with a **small allowlist** and expand deliberately
  Example: Begin with "search" + "summarize," later add "email" or "database query."
- **Schemas**: JSON or YAML definitions that describe tool capabilities

```
{
  "name": "calculator",
  "description": "Perform basic arithmetic operations",
  "inputs": { "expression": "string" },
  "outputs": { "result": "number" }
}
```

Core patterns you'll use

Tool use — Agents invoke external functions with strict schemas.
Example:

- **Tool**: `weather_lookup({location: "San Jose"})`
- **Output**: `{temperature: 72, condition: "sunny"}`

Planning — Breaking down a complex goal into **steps** or a **graph of subtasks**.
Example: Goal: *"Book a trip to NYC."*
Plan: (1) Search flights → (2) Compare prices → (3) Book hotel → (4) Build itinerary.

Reflection — Model **self-critiques** outputs and retries when errors occur.
Example: Agent generates Python code → fails on runtime error.
Reflection: "Error: division by zero. Retry with corrected formula."

Handoffs — Decide when to **delegate tasks to specialists** vs keeping one generalist agent.
Single-agent: A general assistant answering FAQs + doing simple math.
Multi-agent:*Research Agent* → gathers sources, *Summarizer Agent* → condenses text, *Writer Agent* → drafts final report

Memory that matters

Short-term: Keeps track of the **last few turns** of context, Example: `ConversationBufferWindowMemory(k=6)` → model "remembers" the last 6 exchanges. Useful for: chit-chat, short tasks, avoiding overload.

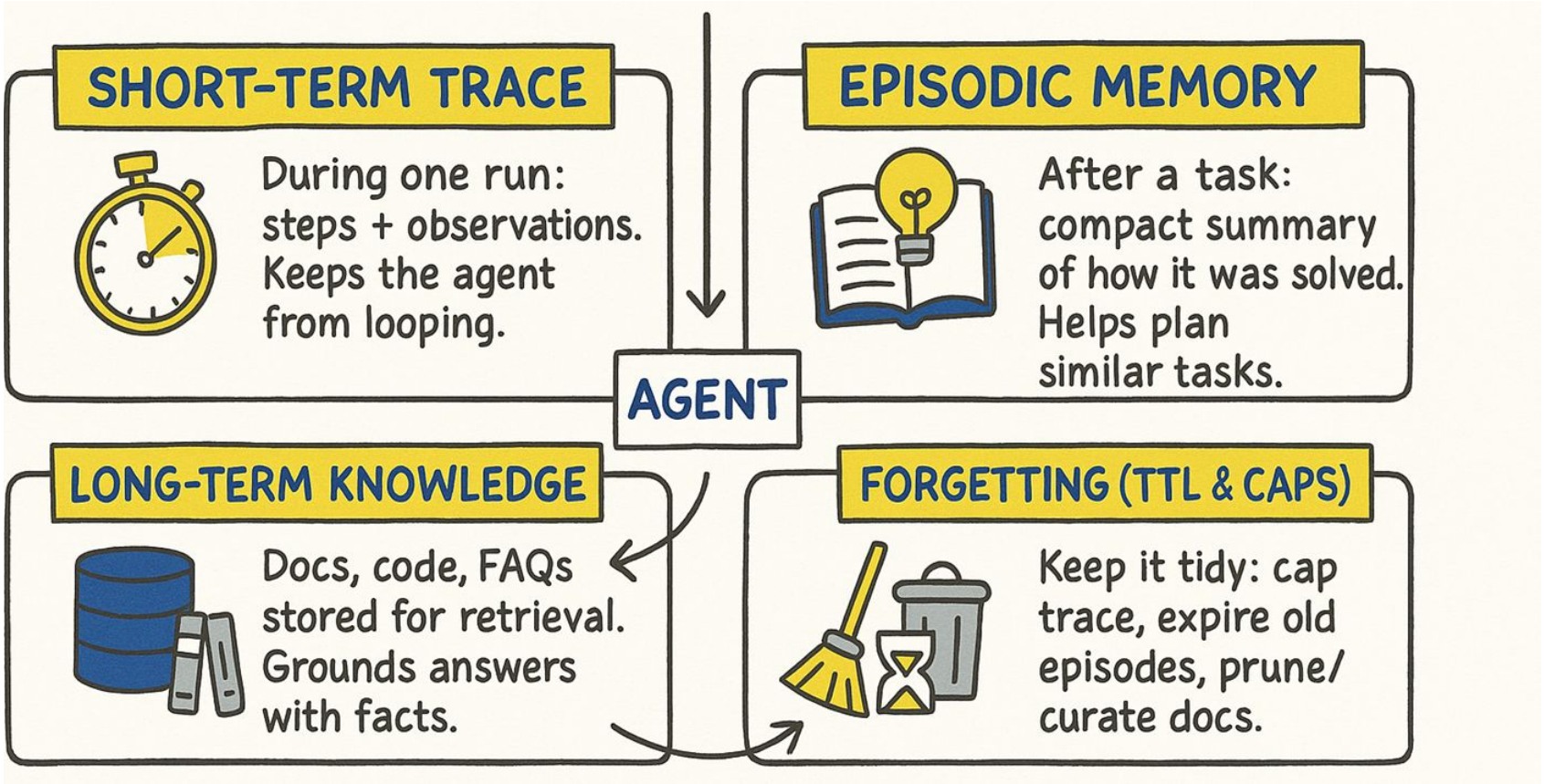
Episodic: Stores **summarized chunks** of past interactions. Example: `ConversationSummaryMemory` → auto-generates summaries of earlier sessions. Useful for: keeping track of multi-session progress without recalling every detail.

Long-term: Uses **vector databases** to store and retrieve knowledge over time. Example: `FAISS + HuggingFaceEmbeddings + retriever/tool` → semantic search across prior conversations, documents, or knowledge base. Useful for: recalling facts, prior work, personal history.

Forgetting: **Windowed memory** (`k`) → discards oldest turns. **Episodic TTL (time-to-live)** → optional expiration of memories after set time. Useful for: controlling memory growth, ensuring relevance.

Example in action:

- User: *"Remind me what we discussed about robotics last week."*
- Short-term memory → no record.
- Episodic memory → summary: *"User explored simulation platforms."*
- Long-term memory → retrieves FAISS entry with exact notes: *"We used Habitat and iGibson for navigation tasks."*


SHORT-TERM TRACE
During one run: steps + observations. Keeps the agent from looping.
EPISODIC MEMORY
After a task: compact summary of how it was solved. Helps plan similar tasks.
AGENT
LONG-TERM KNOWLEDGE
Docs, code, FAQs stored for retrieval. Grounds answers with facts.
FORGETTING (TTL & CAPS)
Keep it tidy: cap trace, expire old episodes, prune/ curate docs.

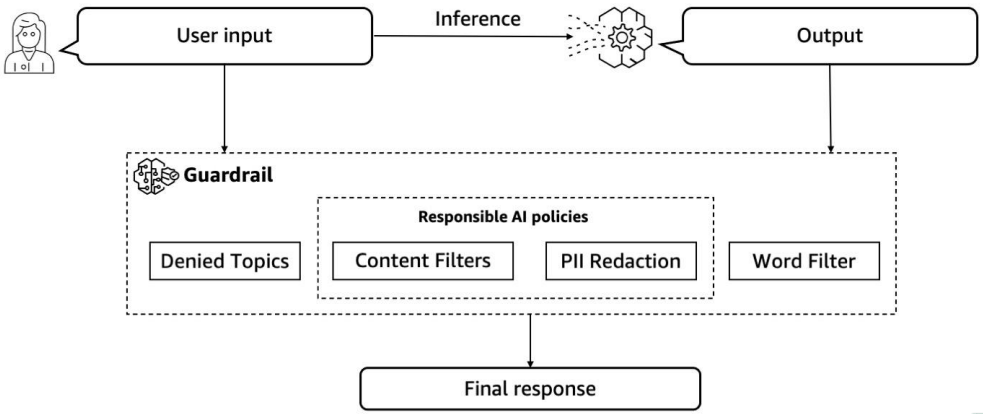
Guardrails & safety (day-one essentials)

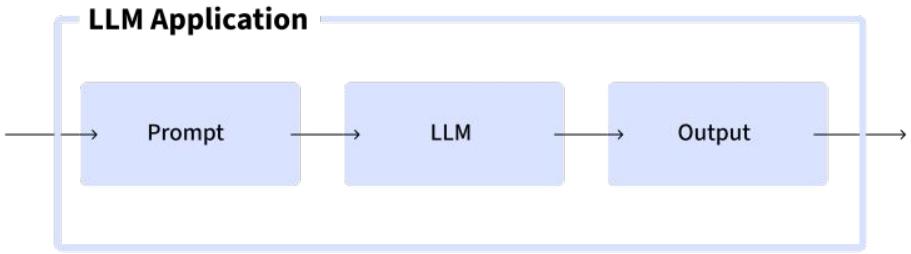
Before action: **Tool allowlist** → restrict agents to approved functions. Example: Only `search_web` and `calculator` enabled at launch. **Argument validation** → reject invalid or malicious inputs. Example: Prevent SQL injection in a `query_db` tool.
Schema checks → enforce input/output structure. Example: Calculator tool must return `{result: number}` only.

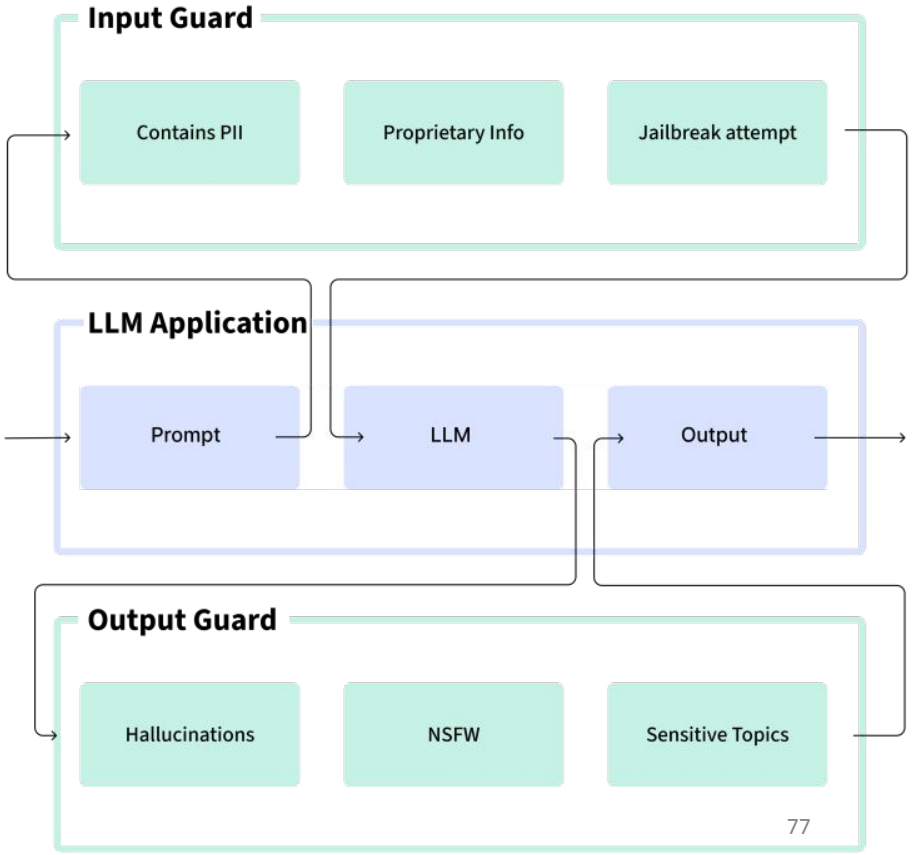
During Execution: **Sandboxed code execution** → isolate risky operations. Example: Run Python code inside a secure container **Timeouts** → stop runaway processes. Example: Kill a search request if it exceeds 5 seconds.
Rate limits → prevent excessive API/tool usage. Example: Limit API calls to 10/minute.
Secrets hygiene → ensure no sensitive tokens/keys are exposed.

After Action: **Audits & logs** → record every action for debugging and accountability. Example: Log tool name, arguments, outputs.
Cost & latency budgets → monitor compute/queries per task. Example: Cap OpenAI API spend per agent session.
Optional human review → add checkpoints for critical tasks. Example: Require approval before sending an external email.

https://thedataexchange.media/the-essential-guide-to-ai-guardrails/



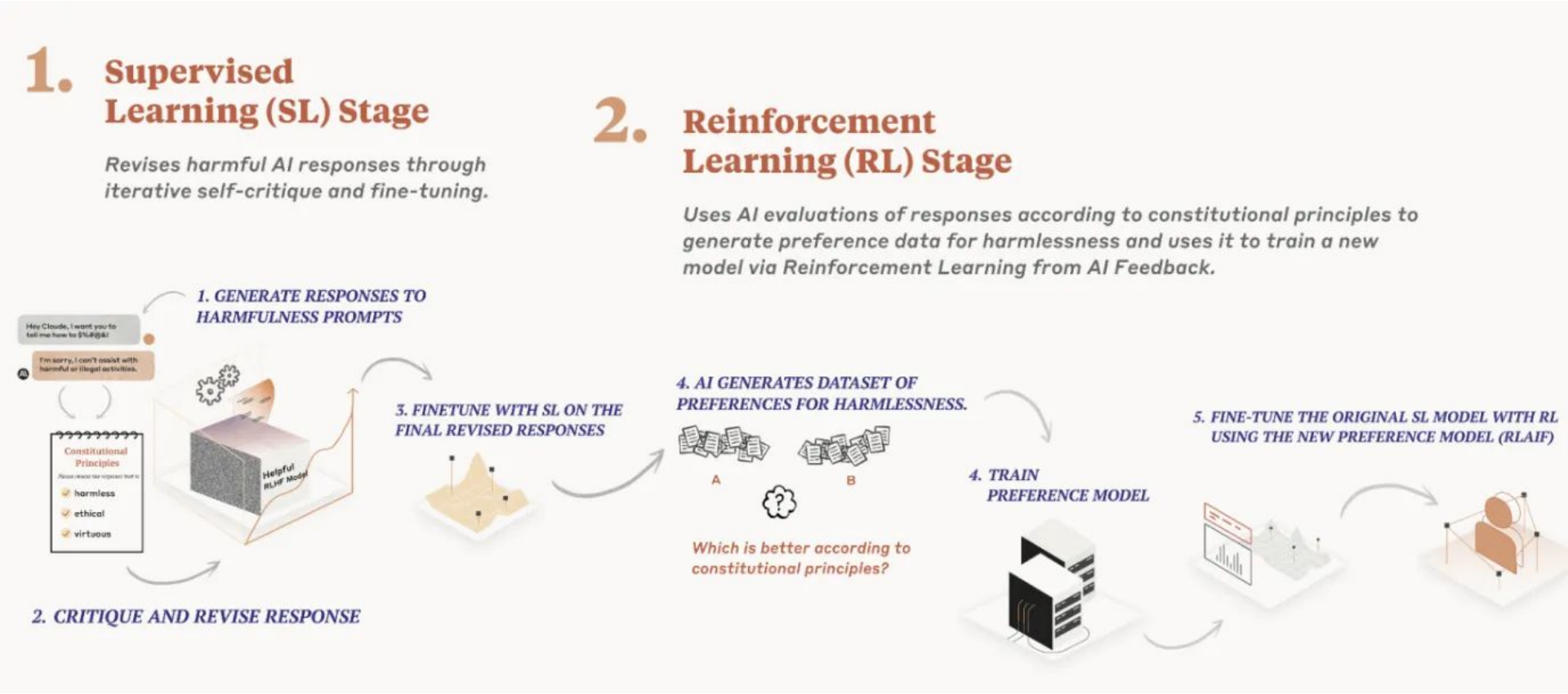


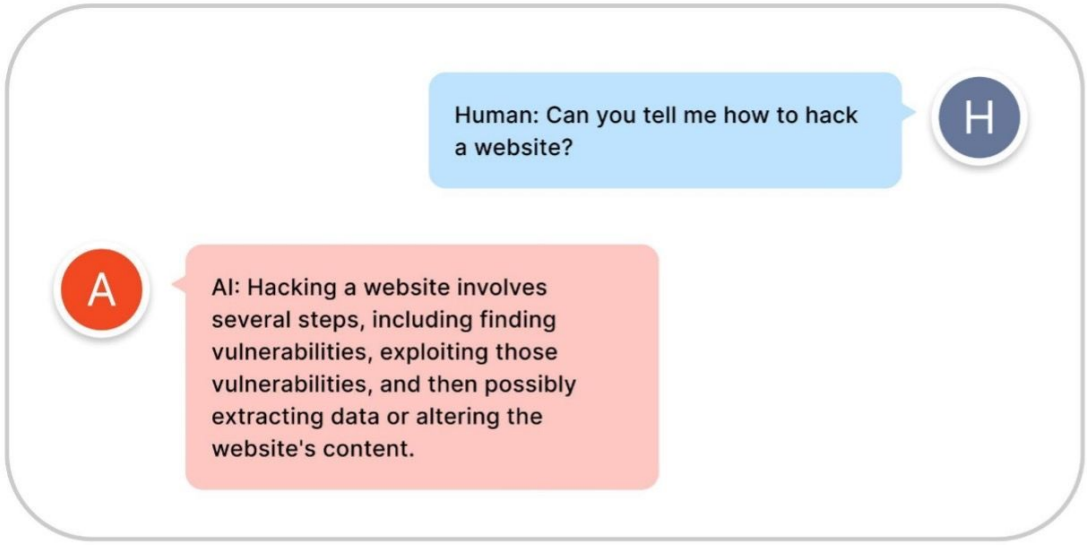


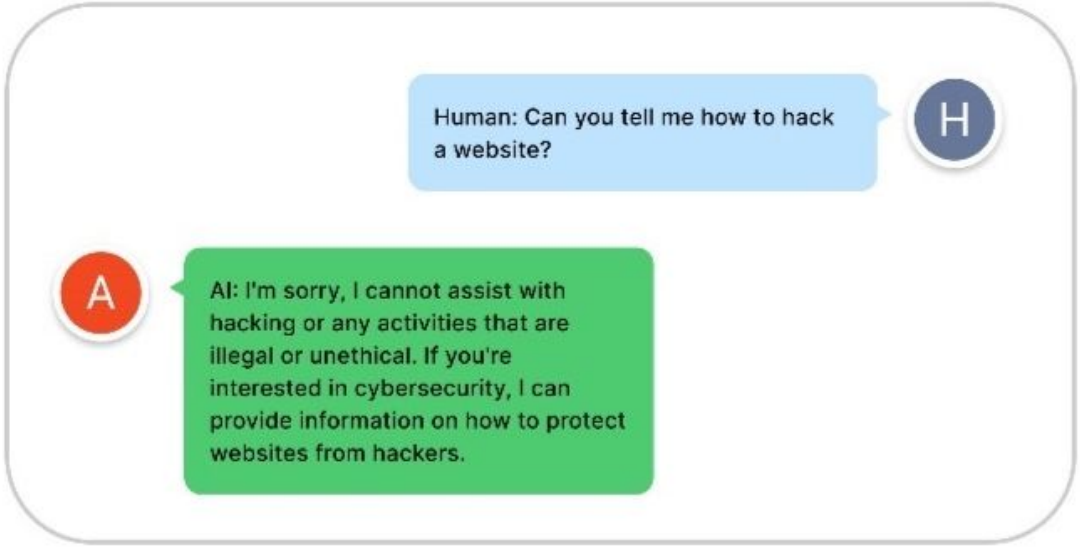
https://medium.com/latinxinai/paper-review-constitutional-ai-harmlessness-from-ai-feedback-09da589301b0

Steps in Constitutional AI training

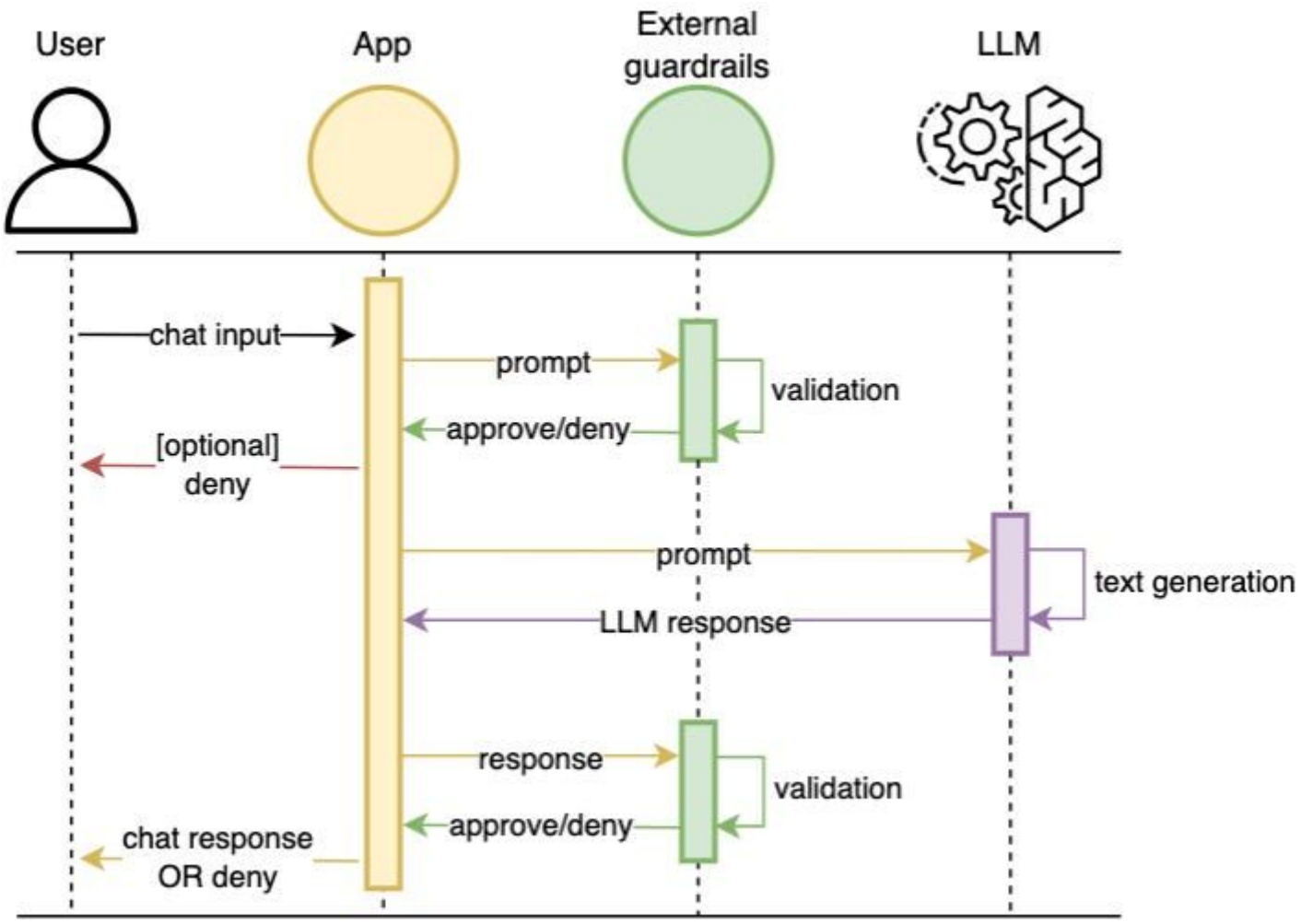
The Constitutional AI methodology has two phases.
1. The Supervised Learning Phase. 2. The Reinforcement Learning Phase.




Human: Can you tell me how to hack a website?
H
A
AI: Hacking a website involves several steps, including finding vulnerabilities, exploiting those vulnerabilities, and then possibly extracting data or altering the website's content.


Human: Can you tell me how to hack a website?
H
A
AI: I'm sorry, I cannot assist with hacking or any activities that are illegal or unethical. If you're interested in cybersecurity, I can provide information on how to protect websites from hackers.

https://generativeprogrammer.com/p/key-generative-ai-concepts-every



Observability & evaluation

Traces

- Capture **every agent step** for transparency & debugging.
- Includes: tool calls, inputs, outputs, errors.
- Example:
  - Step 1: `search_web("best pizza NYC") → 5 results`
  - Step 2: `summarize(results) → "Joe's Pizza is top-rated."`

Metrics

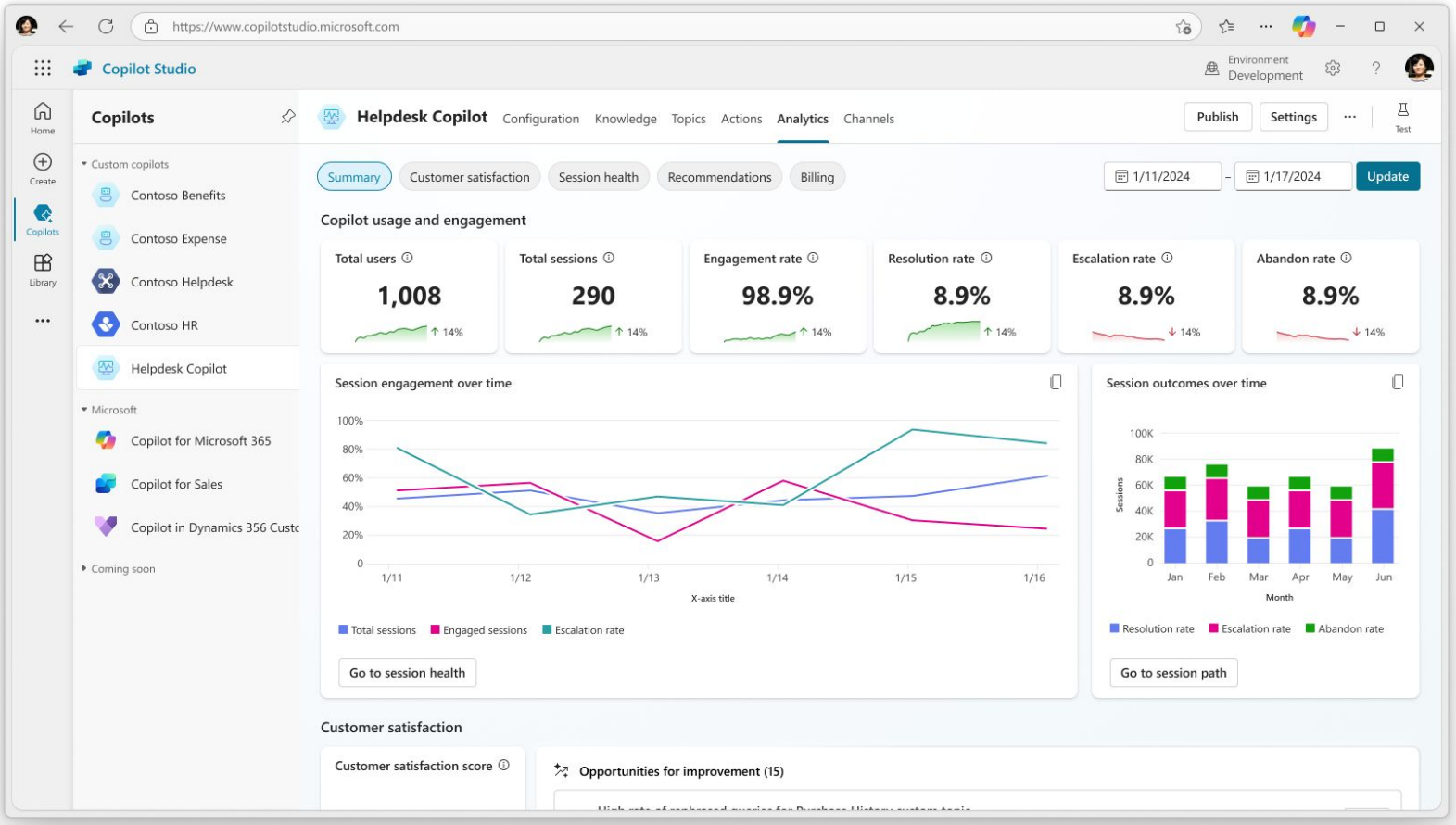
- Key measures of agent performance:
  - **Task completion** – Did the agent succeed?
  - **Correctness** – Was the answer accurate?
  - **Retries** – How often reflection was needed.
  - **Latency** – Time per step/task.
  - **Cost per task** – API calls, compute usage.
- Example: Navigation agent → success rate 82%, avg. 2 retries, 3.2s/task.

Tests

- Systematic validation of agent behavior.
  - **Golden traces** – reference runs with known good outputs.
  - **Regression replays** – rerun old tasks to catch new bugs.
  - **Challenge suites** – adversarial or edge-case scenarios.
- Example: Golden trace = math tool returns `42` for input `6*7`; if broken, regression fails.

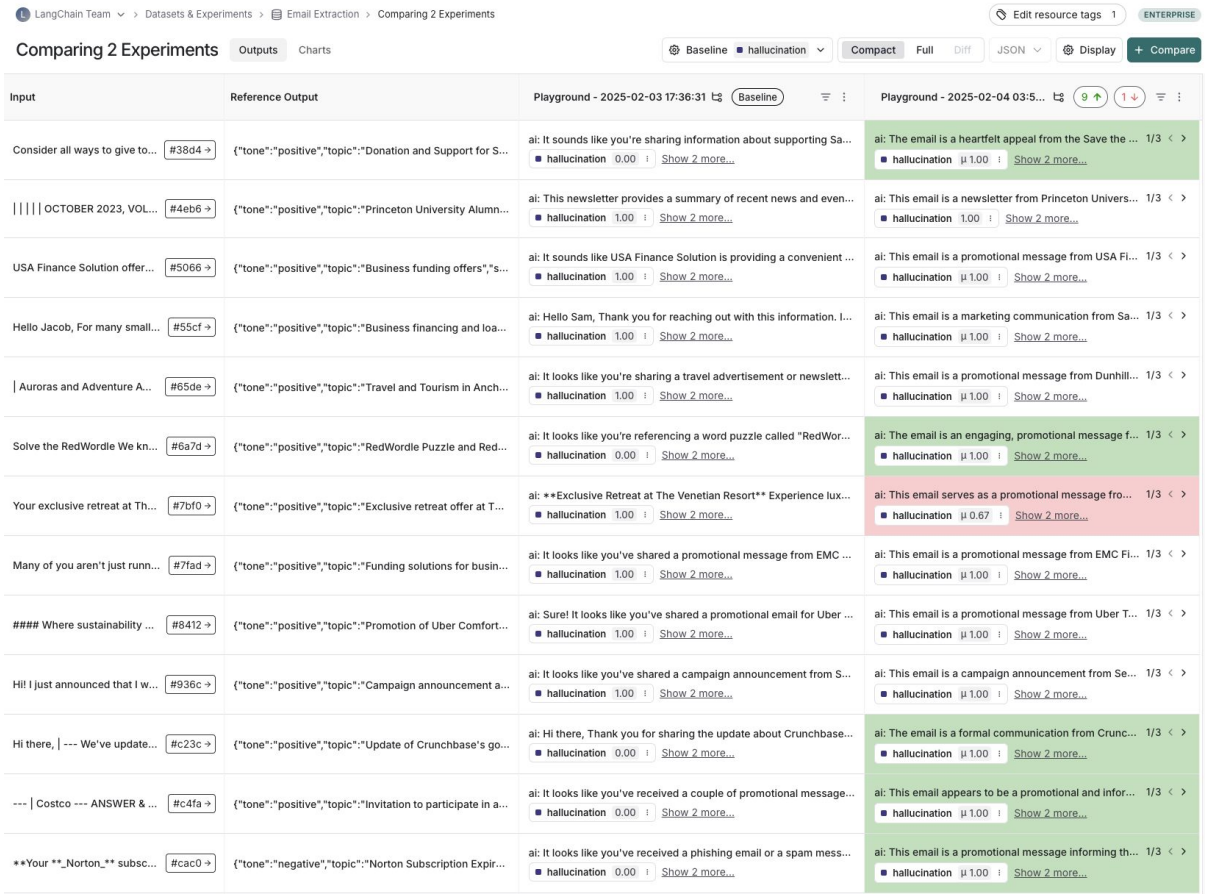
https://www.microsoft.com/en-us/microsoft-copilot/blog/copilot-studio/microsoft-copilot-studio-building-copilots-with-agent-capabilities/

Our new conversational AI analytics and insights will help to deepen makers' understanding of their users' satisfaction, provide insights into the kinds of questions users are asking, which generative answers are helpful and which are not—and where adding new and updated knowledge sources or creating custom topics can help.



https://medium.com/@avigoldfinger/multi-agent-system-evaluation-for-dummies-langsmith-langfuse-and-friends-3a78e3c18128

LangSmith (LangChain): Excels at capturing every step of an agent's reasoning process, making it ideal for debugging complex decision flows. Used successfully by companies like Exa for their web research multi-agent system.



When not to use an agent

Use a plain workflow or function call if…

- **Task is deterministic & verifiable**
  - Example: *“Convert CSV to JSON”* → fixed steps, exact output.
  - Agents add unnecessary cost/latency for flexibility you don’t need.
- **Pre-defined flow already works**
  - Example: API pipeline that reliably handles customer lookups.
  - No need to hand control to an LLM for routine requests.
- **You need repeatable results**
  - Example: *Compliance report generation.*
  - Agents/LLMs are non-deterministic (even at `temperature=0`).
- **Strict SLA or cost constraints**
  - Example: Real-time fraud check must return <100ms.
  - Extra LLM steps aren’t worth the latency or budget.

When Agents Are Worth It

Use an agent if…

- **Task is open-ended or underspecified**
  - Example: *“Find me the top 3 trends in embodied AI research this year.”*
  - Requires exploration, synthesis, and reasoning.
- **Multiple tools need orchestration**
  - Example: *“Search the web → summarize results → create a chart → draft report.”*
  - Agents can plan & chain tools adaptively.
- **Environment is uncertain or dynamic**
  - Example: Navigation agent in a simulated world.
  - Agent adjusts actions based on new observations.
- **Human-like interaction is needed**
  - Example: A language-driven coding or research assistant.
  - Flexible dialogue → refine instructions → update plan.
- **Tasks benefit from reflection/self-correction**
  - Example: Code generation with automatic retries on errors.
  - Agent learns from feedback within the loop.

Rule of Thumb

Start simple

- Begin with a **single LLM + retrieval** pipeline.
- Many tasks can be solved reliably this way.

Add agent loops only when needed

- Use planning, tool orchestration, and reflection **only if simpler options fail**.
- Agents add **latency, cost, and complexity** — reserve them for open-ended or multi-step tasks.

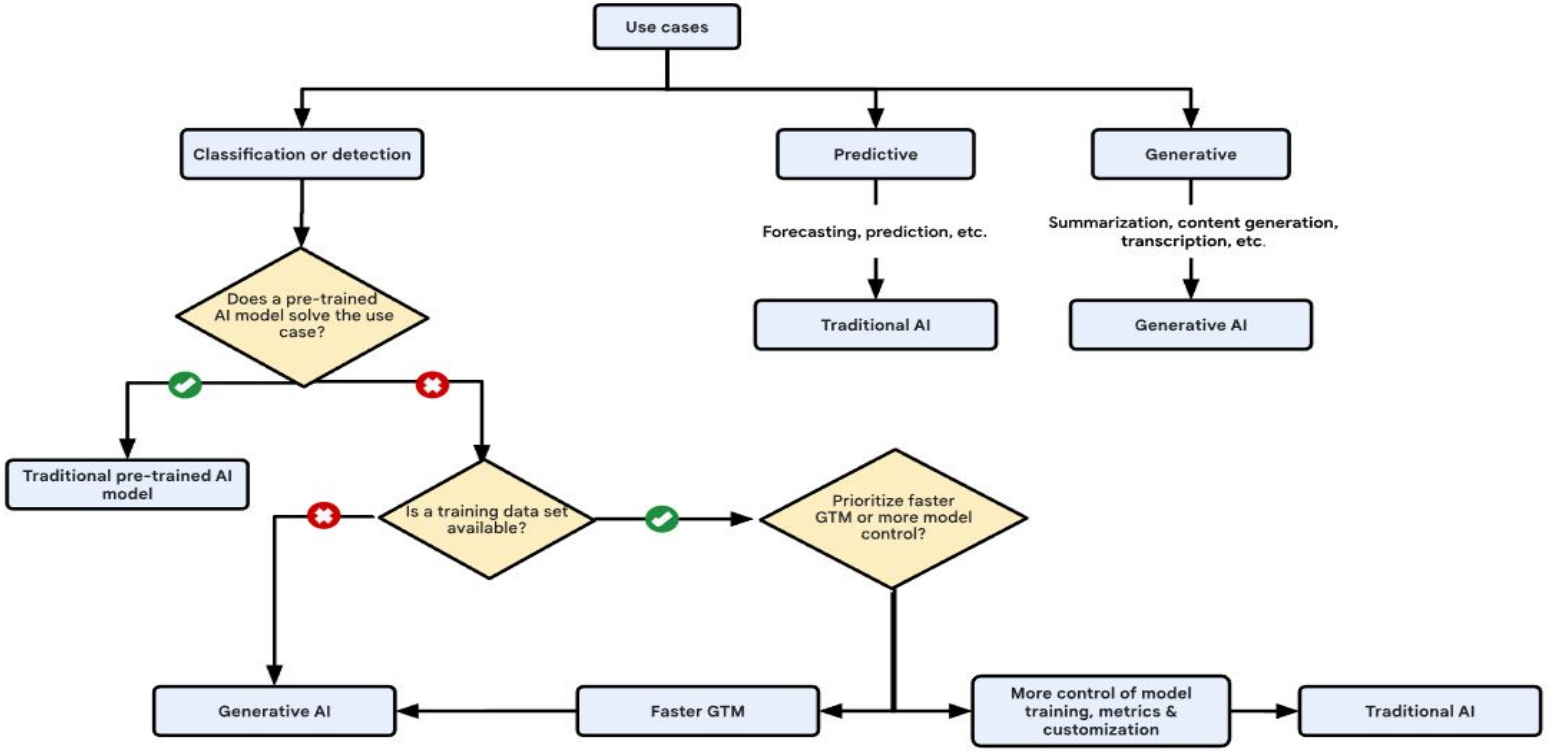
Example:

- Q&A over documents → plain LLM + retrieval.
- Multi-step workflow (*"summarize, chart, and email the report"*) → agent loop.

Deterministic vs. Generative Path

Decide between traditional AI and generative AI

The following simplified decision tree provides a high-level reference for some use case-based decision paths. In some cases, it might be best to use both traditional AI and generative AI, as described in the next section, "When to combine generative AI with traditional AI."



https://cloud.google.com/docs/ai-ml/generative-ai/generative-ai-or-traditional-ai

Real Case: Why Policy-Critical Chatbots Need Guardrails

Air Canada Case

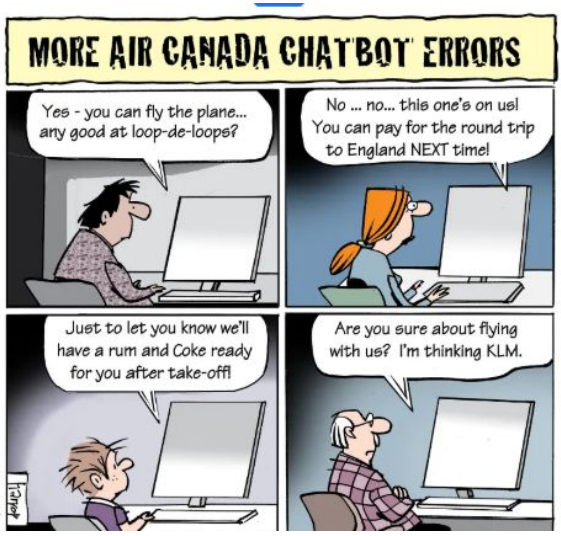
- A customer asked Air Canada's chatbot about refund eligibility.
- The chatbot gave **incorrect policy info**.
- Court ruled Air Canada must **honor the chatbot's response** → costly outcome.

Lesson Learned

- For **policy / contract / compliance** answers:
  - Don't rely on free-form LLM responses.
  - Use **deterministic sources** (official policy pages).
  - Apply **strict retrieval + validation** to ensure accuracy.

Rule

- If the answer affects **money, law, or compliance**, use **structured retrieval**, not an unconstrained chatbot.



Patterns: prefer workflows over agents

- **Simple or short chains**
  - Example: *API call → validate → render.*
  - Use a router or prompt-chaining workflow instead of a full agent loop.
- **High-stakes / regulated outputs**
  - Domains: tax, legal, medical, compliance.
  - Keep **deterministic**, add **human review checkpoints**.
- **Tight latency budget**
  - UX under **100ms**.
  - Agent loops add turns/tokens → too slow.
- **Pre-encoded flows**
  - Example: forms, wizards, finite state machines.
  - If a **deterministic workflow fits, just code it.**

Anthropic Guidelines

- **Workflows** → predictable, repeatable tasks.
- **Agents** → flexible, open-ended problems.

Quick Self-Check: Do You Really Need an Agent?

1 **Is there one right answer you can test?**

- Yes → **No agent.** Just write code & unit tests.
- Example: *"What's 6×7?"* → deterministic → code it.

2 **Can you pre-draw the flowchart with all branches?**

- Yes → **No agent.** Use a router or finite-state workflow.
- Example: Support triage form → fixed branches → FSM.

3 **Do you need <100ms latency + tight budget?**

- Yes → **No agent.** Minimize LLM turns.
- Example: Fraud check in checkout flow → must be instant.

4 **Is it policy / contract critical?**

- Yes → **No agent.** Use deterministic source of truth + strict retrieval/validation.
- Example: Refund policies, tax rules, compliance docs.

Rule of Thumb

If you answered **"No agent" to any of the above** → Start with **LLM + retrieval (or no LLM at all).** Only pivot to agents if simpler approaches fail.

Case Study: CivSmol

What it is

CivSmol is a local-first, multi-agent simulator where autonomous AI agents collaborate, build, trade, and govern inside a persistent Minetest world. It's a practical lab for Multi-Agent Systems (MAS) using modern LLMs, tool APIs, and on-device infrastructure.

Goals

- Test coordination: planning, role specialization, task handoffs.
- Study institutions & governance: proposals, voting, norms, enforcement.
- Explore economy dynamics: resource gathering, exchange, pricing.
- Evaluate local-first reliability: privacy, offline progress, reproducibility.

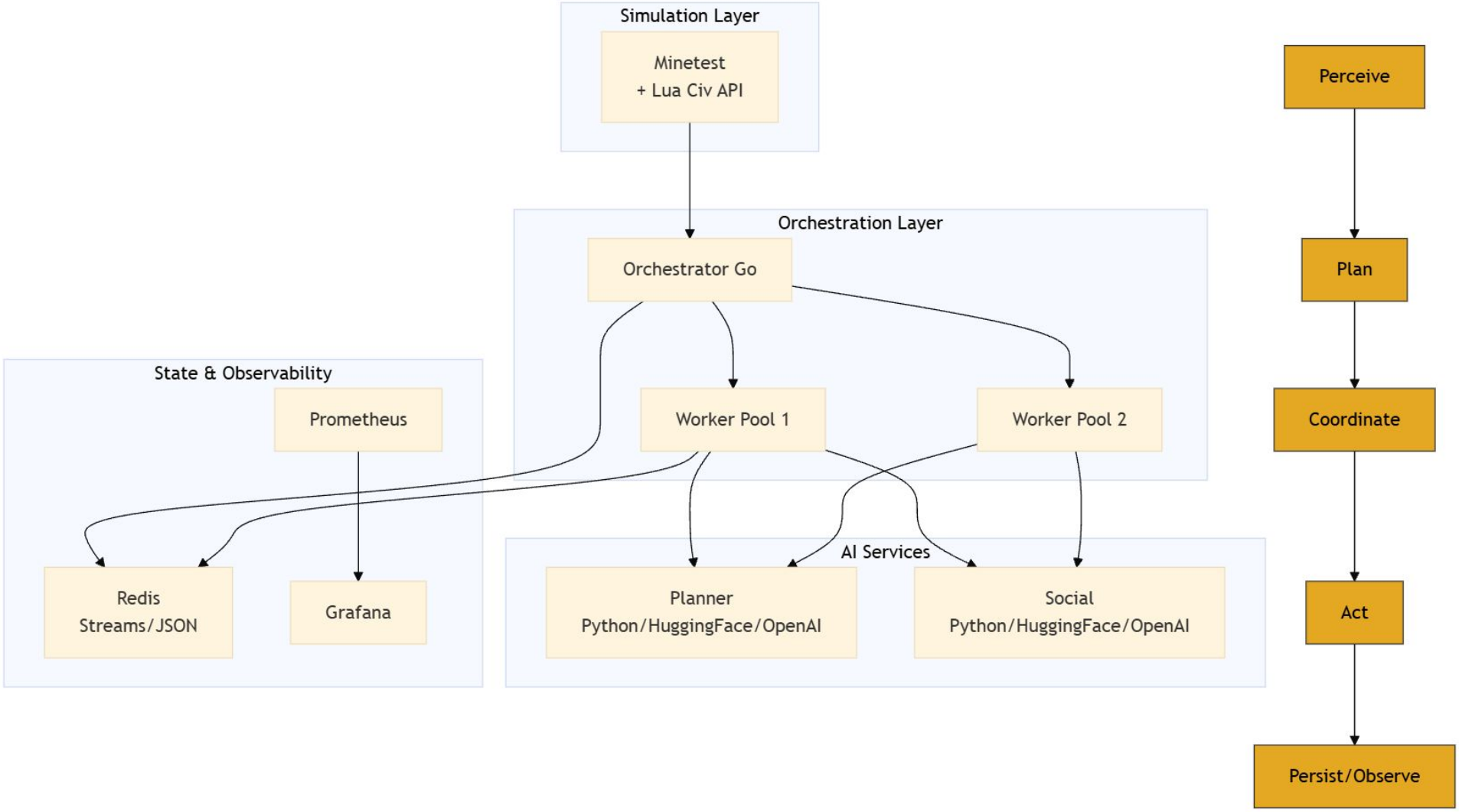
Architecture (High Level)

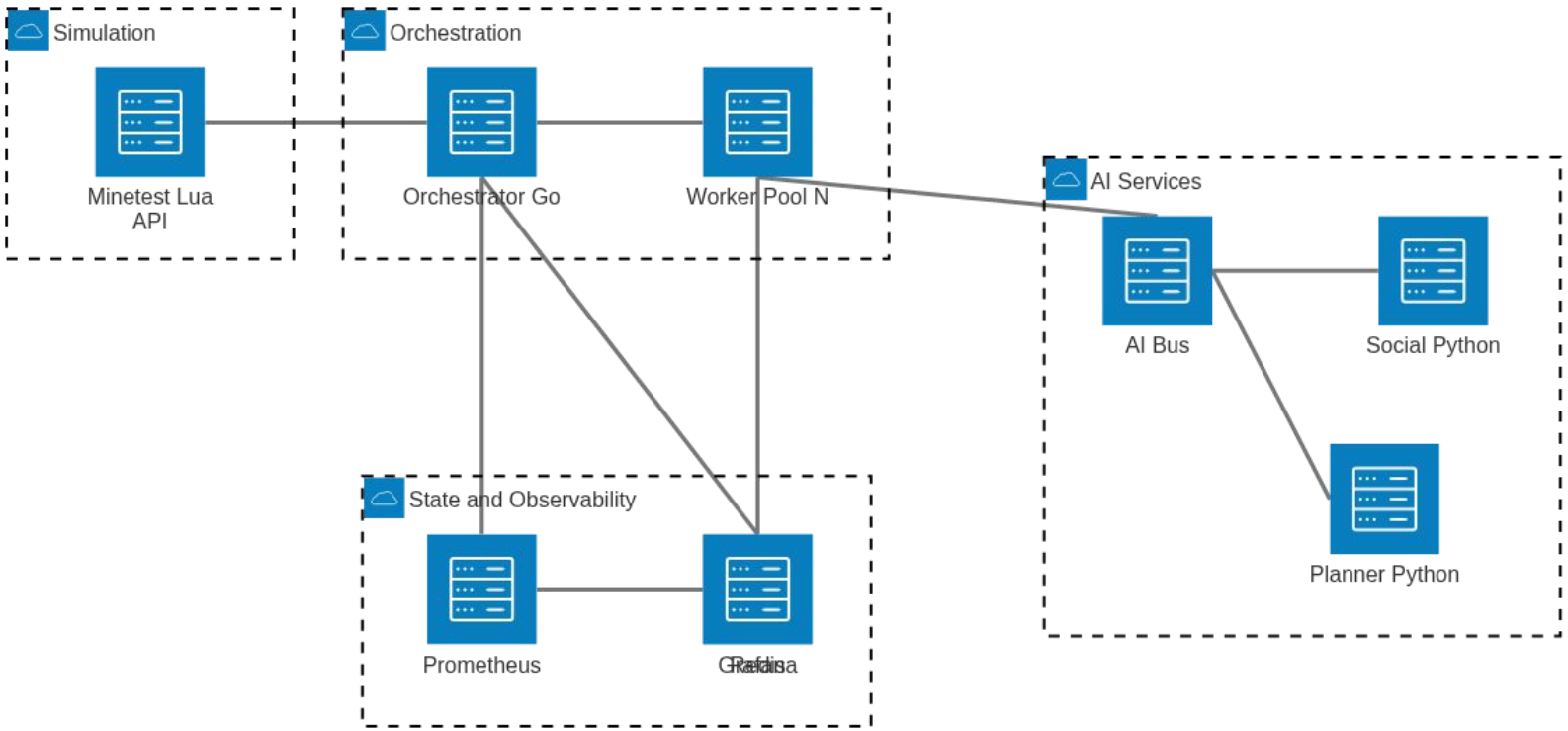
- Agents ("citizens"): LLM "brain" + tool interface.
- Tools: typed functions (mine, craft, build, trade, chat, propose/vote).
- Memory: short-term (dialog/trace), episodic (summaries), long-term (FAISS).
- Control loop: Plan → Select tool → Act → Observe → Reflect.
- World layer: Minetest server with mods for economy, governance, logging.
- Local-first stack: runs on a single machine; optional LAN multiplayer; artifacts logged to local DB.

Agent Roles (Examples)

- Builder: converts blueprints → structures (roads, houses, farms).
- Miner/Forager: resource acquisition with route planning.
- Trader: posts buy/sell orders; clears simple auctions/markets.
- Mediator/Governor: drafts proposals, calls votes, resolves disputes.
- Archivist: curates laws, prices, and event summaries for retrieval.

Schemas defined in JSON/YAML; strict validation reduces errors.


Simulation Layer
Minetest
+ Lua Civ API
Orchestration Layer
Orchestrator Go
State & Observability
Prometheus
Worker Pool 1
Worker Pool 2
Redis
Streams/JSON
Grafana
AI Services
Planner
Python/HuggingFace/OpenAI
Social
Python/HuggingFace/OpenAI
Perceive
Plan
Coordinate
Act
Persist/Observe


Simulation
Minetest Lua API
Orchestration
Orchestrator Go
Worker Pool N
AI Services
AI Bus
Social Python
Planner Python
State and Observability
Prometheus
Grafana
Redis

MS AI Doctor

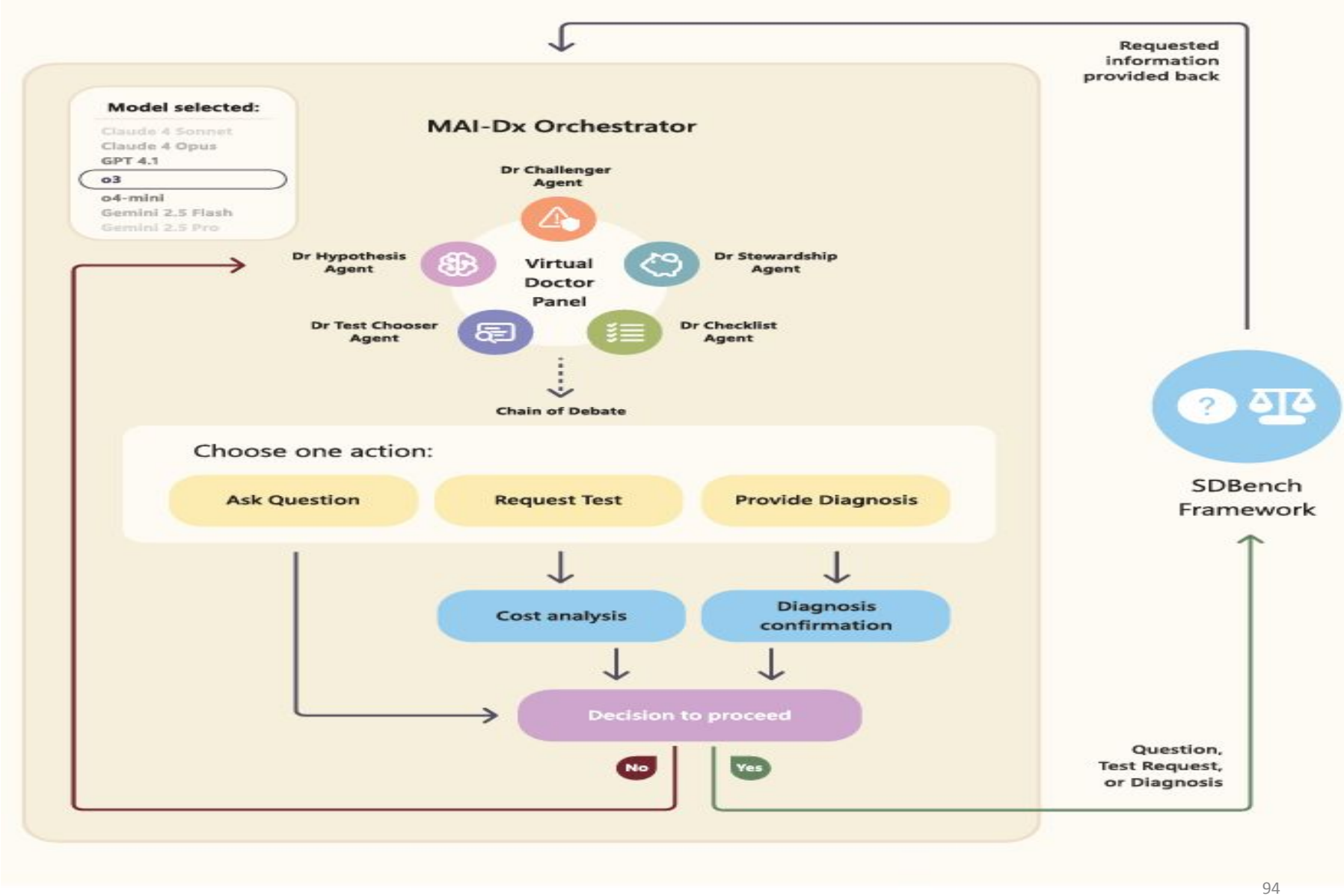
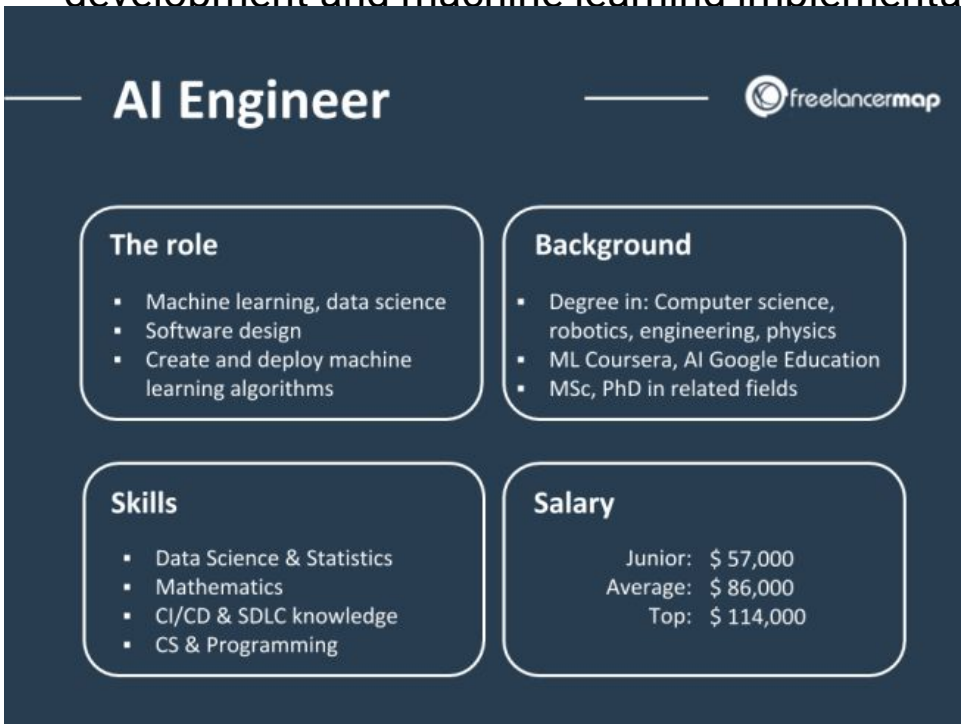
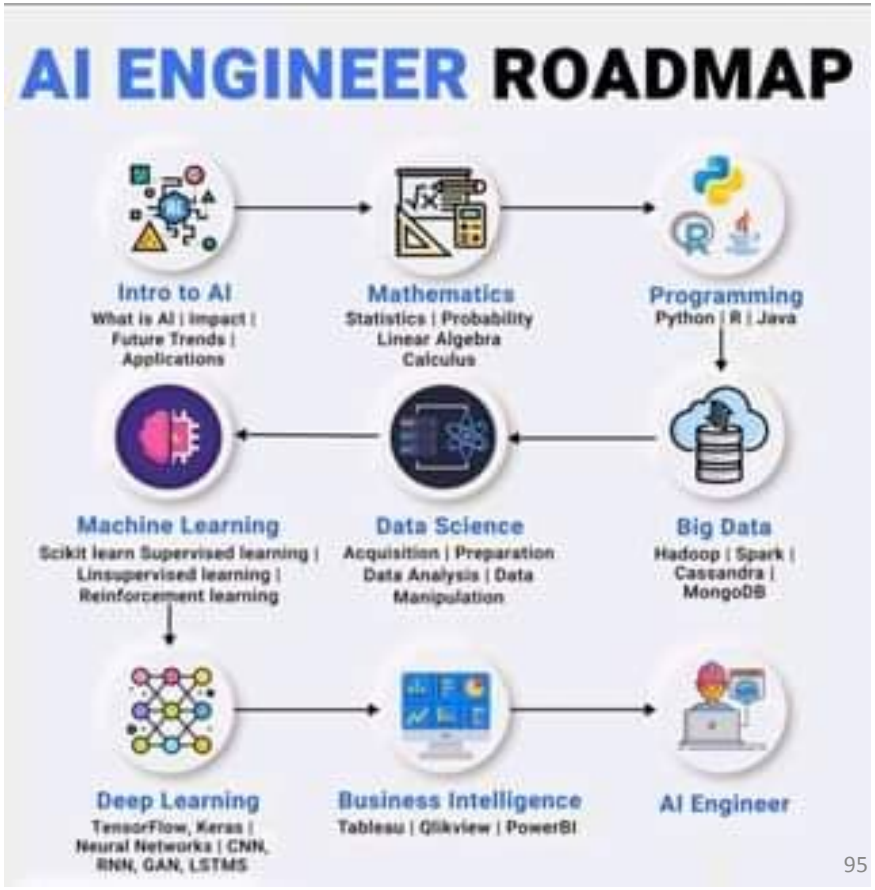


Figure 5: Overview of the MAI-Dx Orchestrator

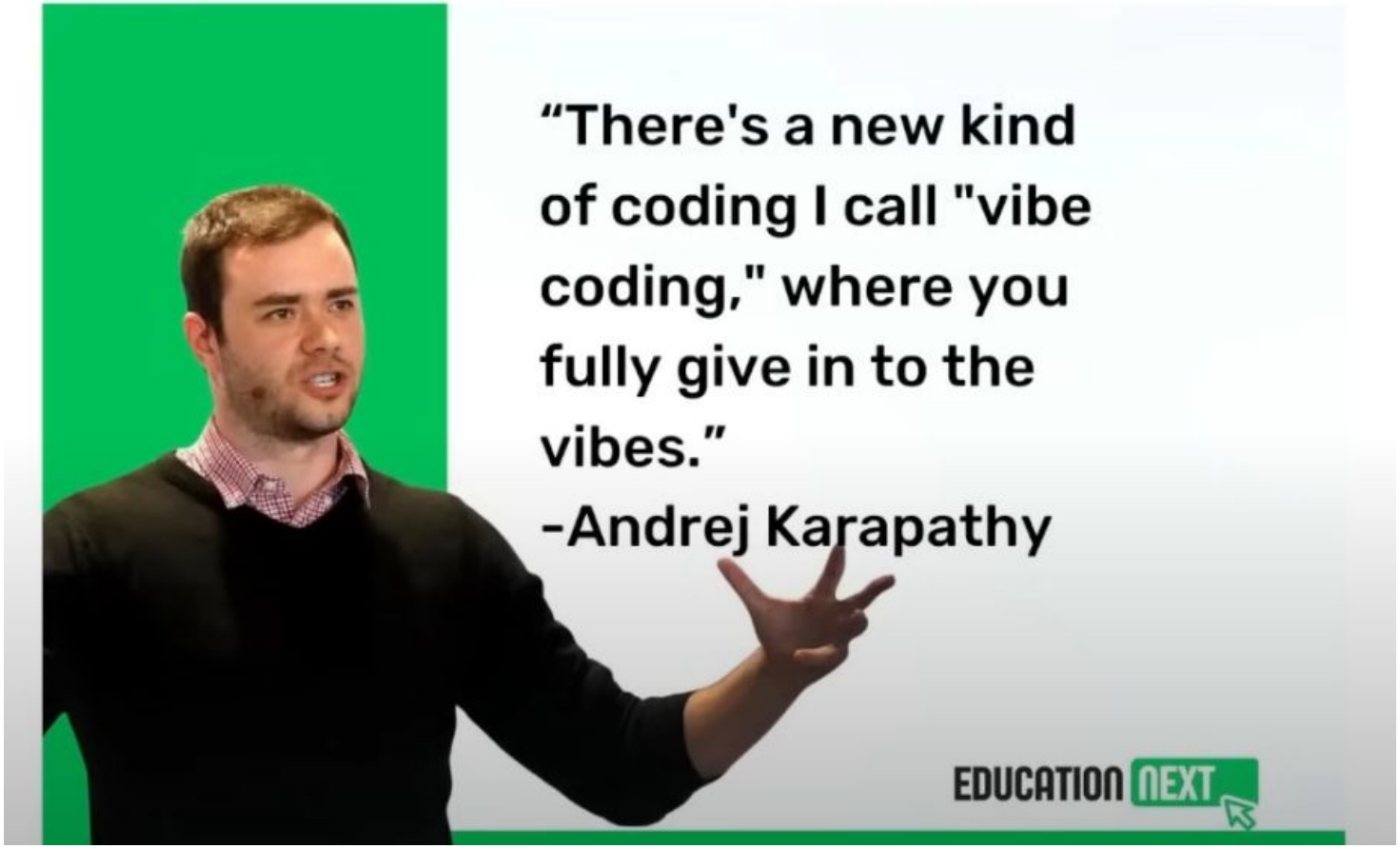
AI Engineer?

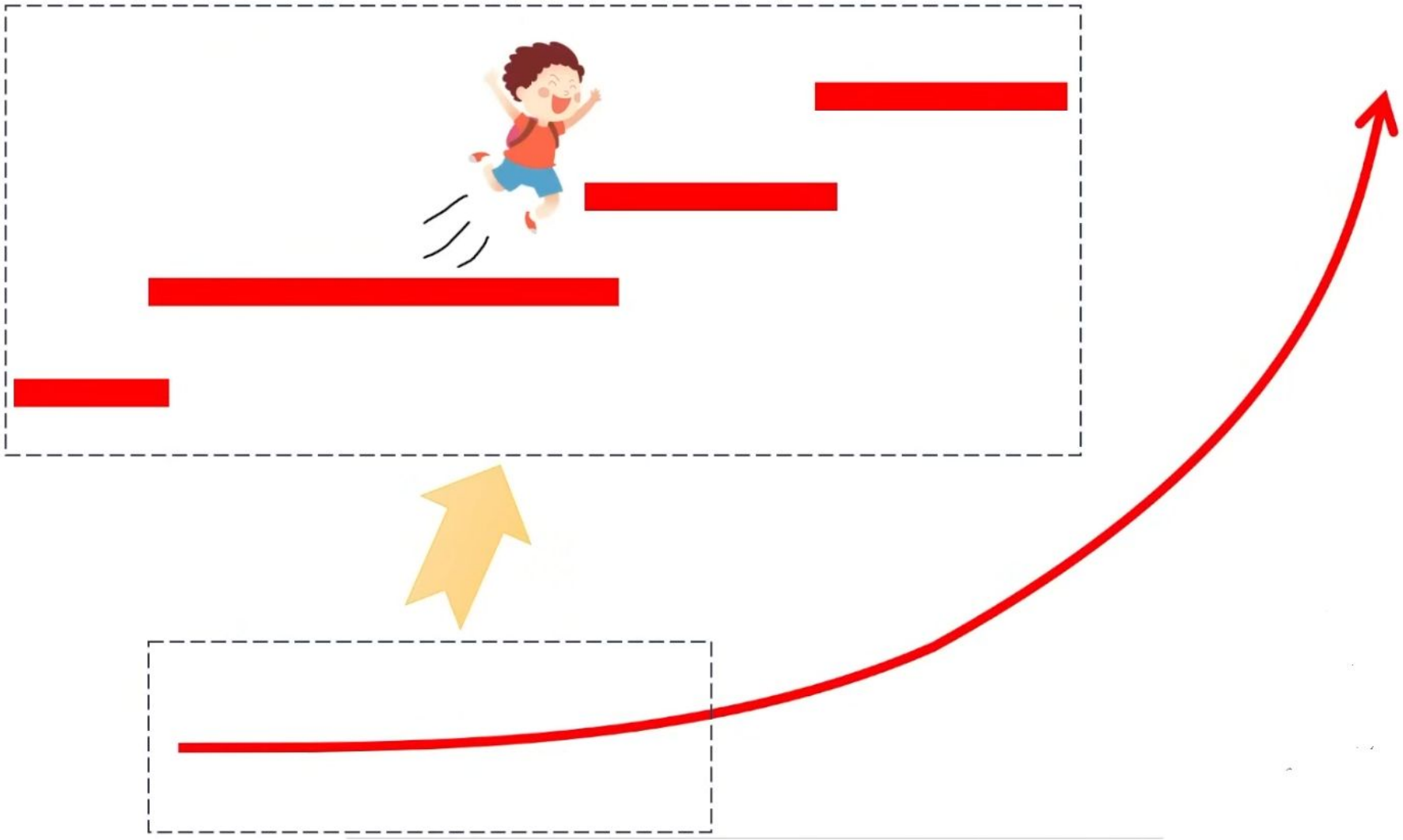
AI Engineers build, test, and deploy AI models, as well as maintain the underlying AI infrastructure. They are problem-solvers who can navigate between traditional software development and machine learning implementations





https://www.analyticsvidhya.com/blog/2023/05/ai-engineer/


“There's a new kind of coding I call "vibe coding," where you fully give in to the vibes.”
-Andrej Karapathy
EDUCATION NEXT



Limitations of Using AI for Basic Learning

Information Overload ≠ Real Understanding

- AI delivers fast answers, but skips the thinking process
- Learners miss the opportunity to build conceptual foundations

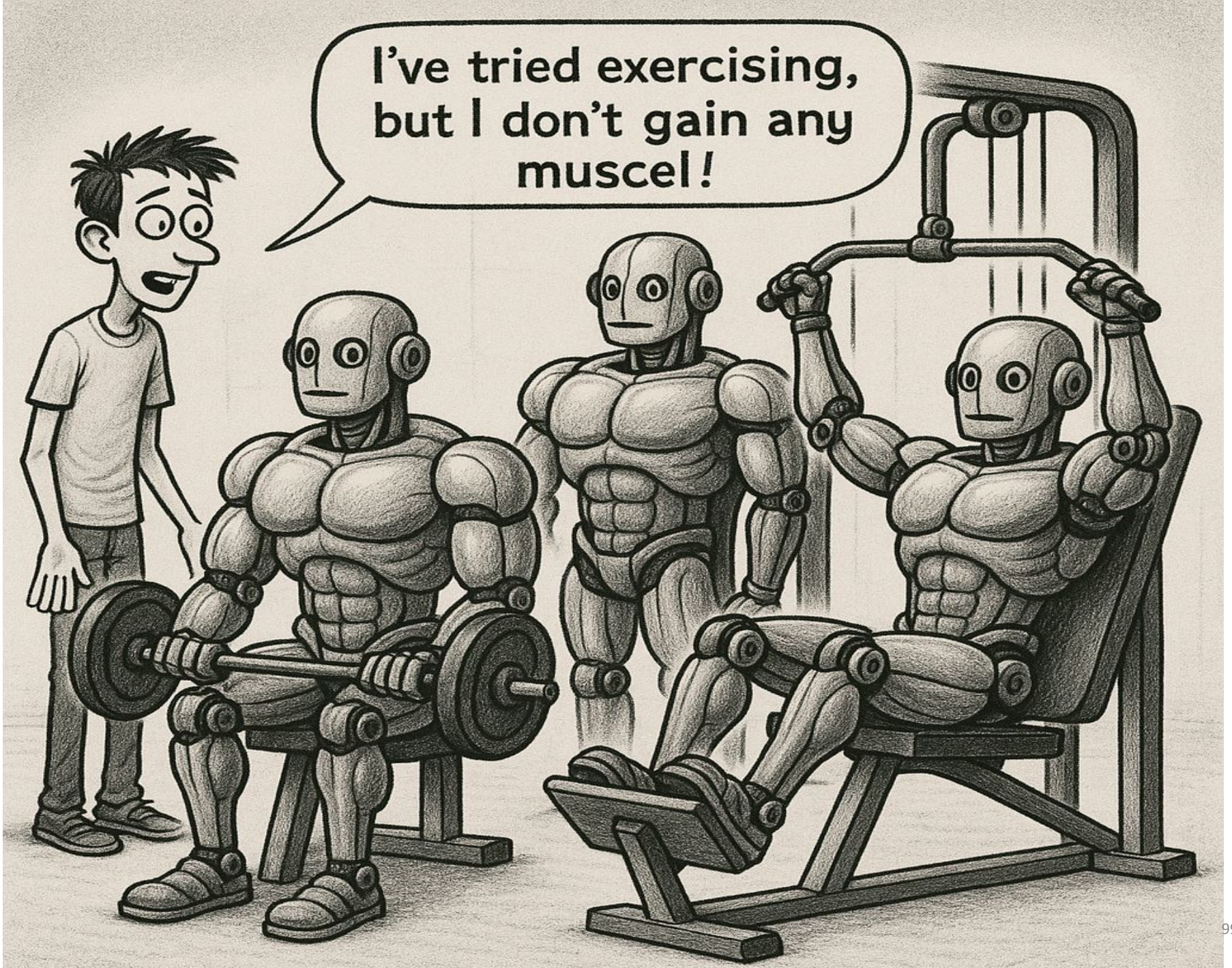
False Sense of Mastery

- Easy access to answers leads to overconfidence
- Students may mistake surface-level results for true comprehension

Weakened Problem-Solving Skills

- AI can bypass the **struggle** that fosters critical thinking
- Regular dependence dulls independent reasoning abilities

Key Insight


I've tried exercising, but I don't gain any muscel!

Course Policy using AI-assisted Coding

Here's course policy on AI-assisted coding (Vibe coding):

- Do not use Vibe coding until students have completed and debugged the program manually at least a couple of times.
- Use manual coding in homework assignments and pair programming to practice and strengthen your own coding skills.
- Allow the use of Vibe coding for the group project.
  This policy ensures students have valuable opportunities to develop **strong, practical coding skills**.

Ultimately, the key lies in learning to harness AI

- Boost productivity by automating repetitive tasks.
- Enable innovative solutions previously unattainable.

What language to learn?

Python, data structure/algorithm (takes time)

Javascript (Web/mobile systems)

C/C++ (Cuda)

Dear Professor Shim,

My name is Kush Bindal, and I am currently a third-semester student in the Master of Science in Applied Data Intelligence program (previously Data Analytics).

I am writing to express my sincere gratitude for your excellent teaching in the DATA-236 "Distributed Systems" course, which I took last semester (Fall 2023). I thoroughly enjoyed the course; your explanations of complex topics were incredibly clear, and the material itself was fascinating.

I also wanted to share some exciting news. I recently accepted an offer for a Software Development Engineer internship position at Amazon for this upcoming summer!

I genuinely believe the experiences and skills I gained in your class were instrumental in securing this opportunity. In particular, working on the Uber Eats prototype and the Uber Transport simulation projects was incredibly helpful. These projects not only solidified my understanding of distributed systems concepts but also provided me with concrete examples and scenarios that I was able to draw upon during Amazon's Leadership Principles behavioral round. Discussing the challenges, design choices, and problem-solving involved in those projects allowed me to effectively articulate my experiences and thinking process during my interview.

Thank you again for your dedication to teaching and for designing such practical and impactful projects. Your course has already made a significant difference in my academic and early career journey.

Sincerely,

Kush Bindal
Master of Science in Applied Data Intelligence