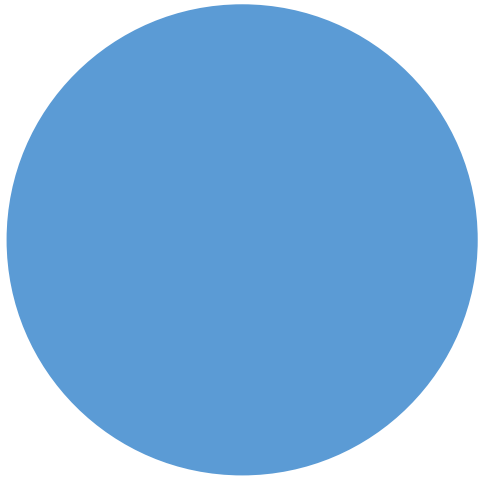
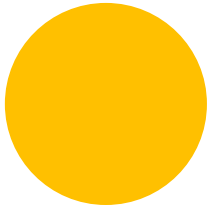



Assignment 1 Details

Mocking Up a Science Gateway with Microservices

The Story: Weather Forecasting Research

You are building a science gateway to support weather forecasting research.

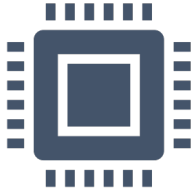
US-based numerical weather prediction is undergoing dramatic changes.

Available input data from satellites, radar, drones, etc is growing in both volume and diversity	Government agencies are academic researchers are consolidating on a single modeling framework (FV3)	**Machine learning** is helping researchers to **identify important features** in very large data sets.

Challenges for the Research Community

Input data sets are large, diverse, and difficult to handle

Different researchers may use different data combinations, different filters



FV3 and other numerical weather prediction codes are difficult to install and use

Analysis requires use of machine learning to identify important features in very large data sets

You Need a Science Gateway

Each stage of the process requires special resources

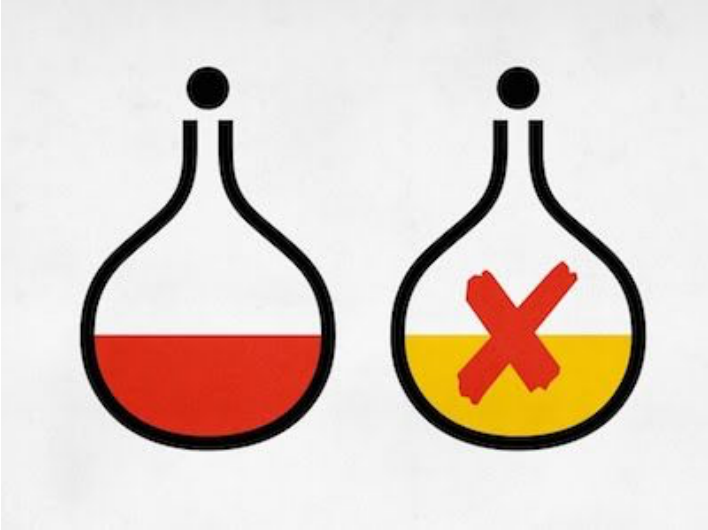
- Supercomputers
- Massive, high-performance file systems
- Massive archives

You can’t do this on your laptop or workstation.

Scientists in many fields have a cloud computing problem

But this is just a problem of technology. There is also something more fundamental.

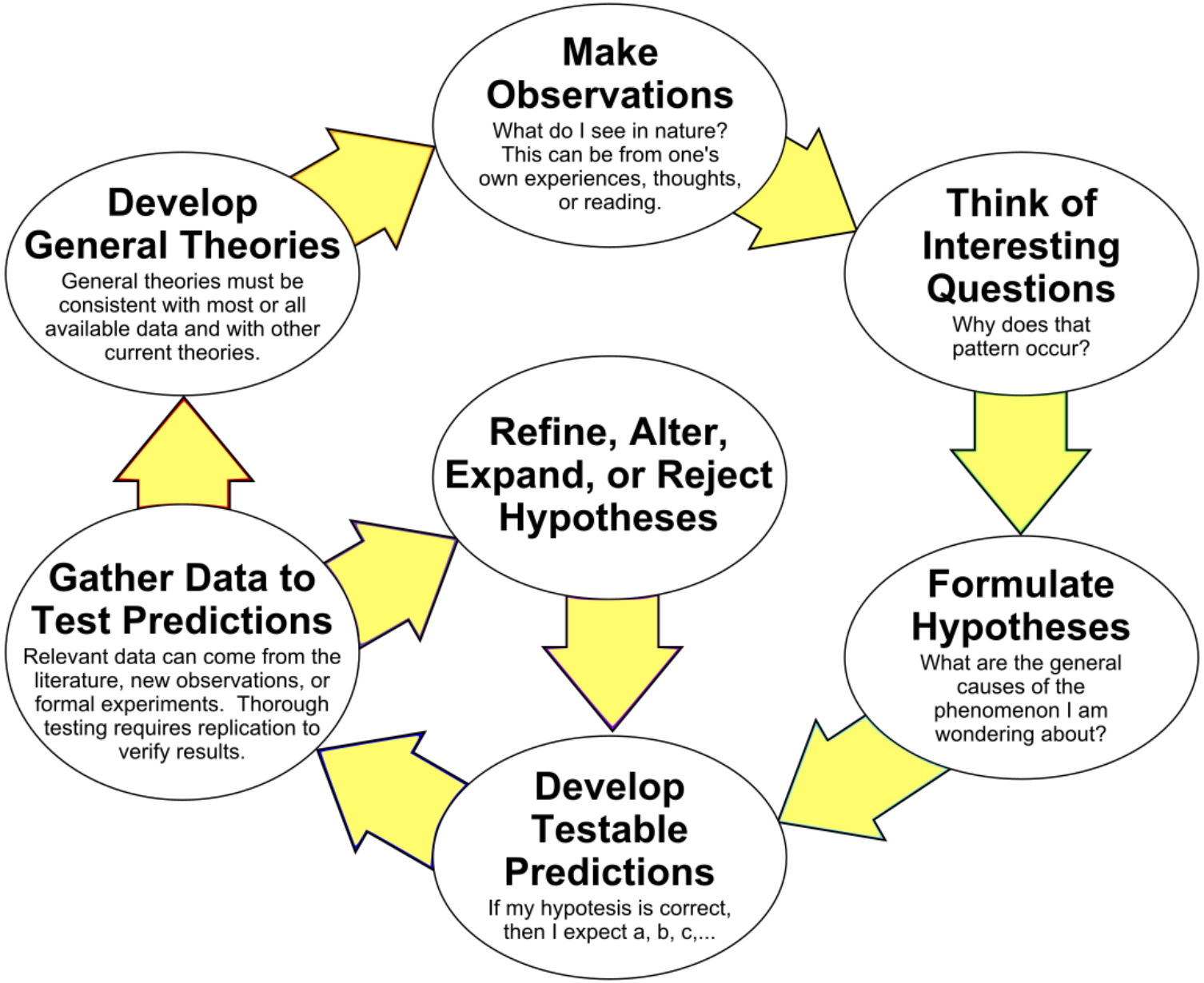
There's Another Problem: Reproducibility



- Different researchers will use different settings
  - Different input data
  - Different settings and configurations of FV3
  - Different analysis methods
- If I claim to have the best method for detecting, say, precursors to tornados on a national scale, how can other scientists check my work?

A Fundamental Problem: Using IT to Support the Scientific Method

The Scientific Method as an Ongoing Process



Mock Up an NWP Science Gateway

Data Retrieval

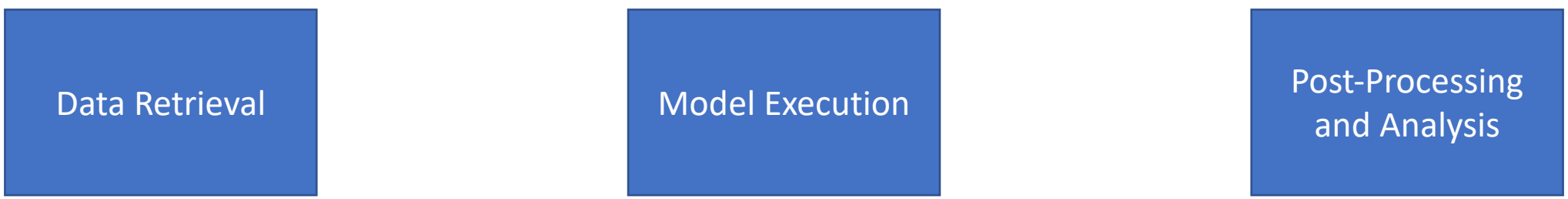
- Fetch observational data that will be used as input for forecasting models.
- Data sets are often very large

Forecasting Model Execution

- This is typically done on supercomputers
- Outputs are

Post-Processing and Feature Identification

- This is done with machine learning, statistical, and other techniques


Data Retrieval
Model Execution
Post-Processing and Analysis

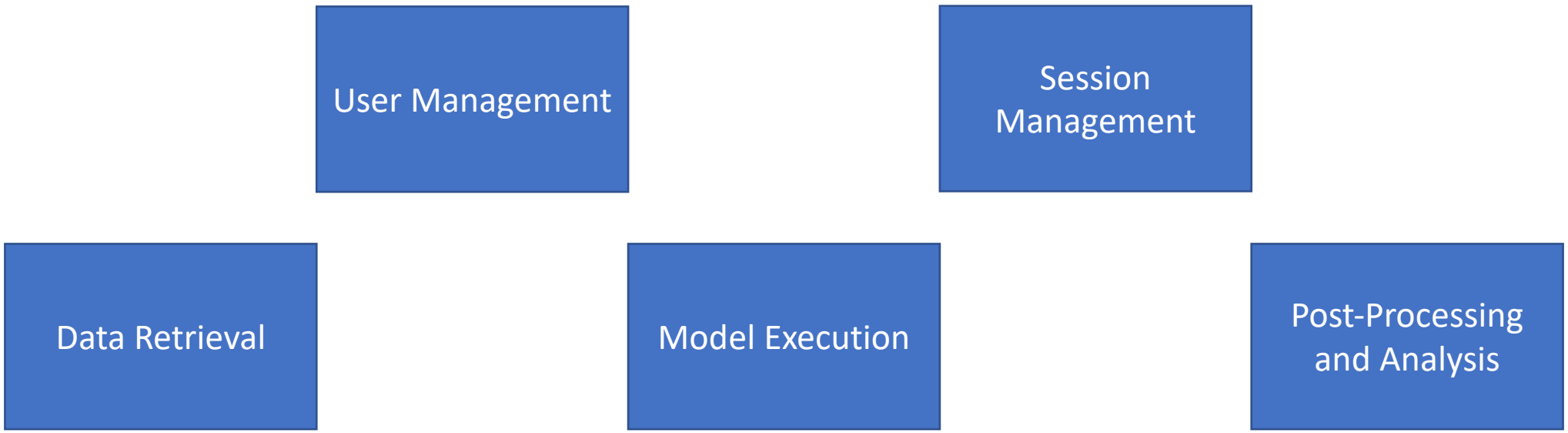
Two More Backend Services

User Management

- You need to authenticate users

Persistent Session Management

- You need to record what users do for them
- Users may want to review and modify their previous sessions.


User Management
Session Management
Data Retrieval
Model Execution
Post-Processing and Analysis

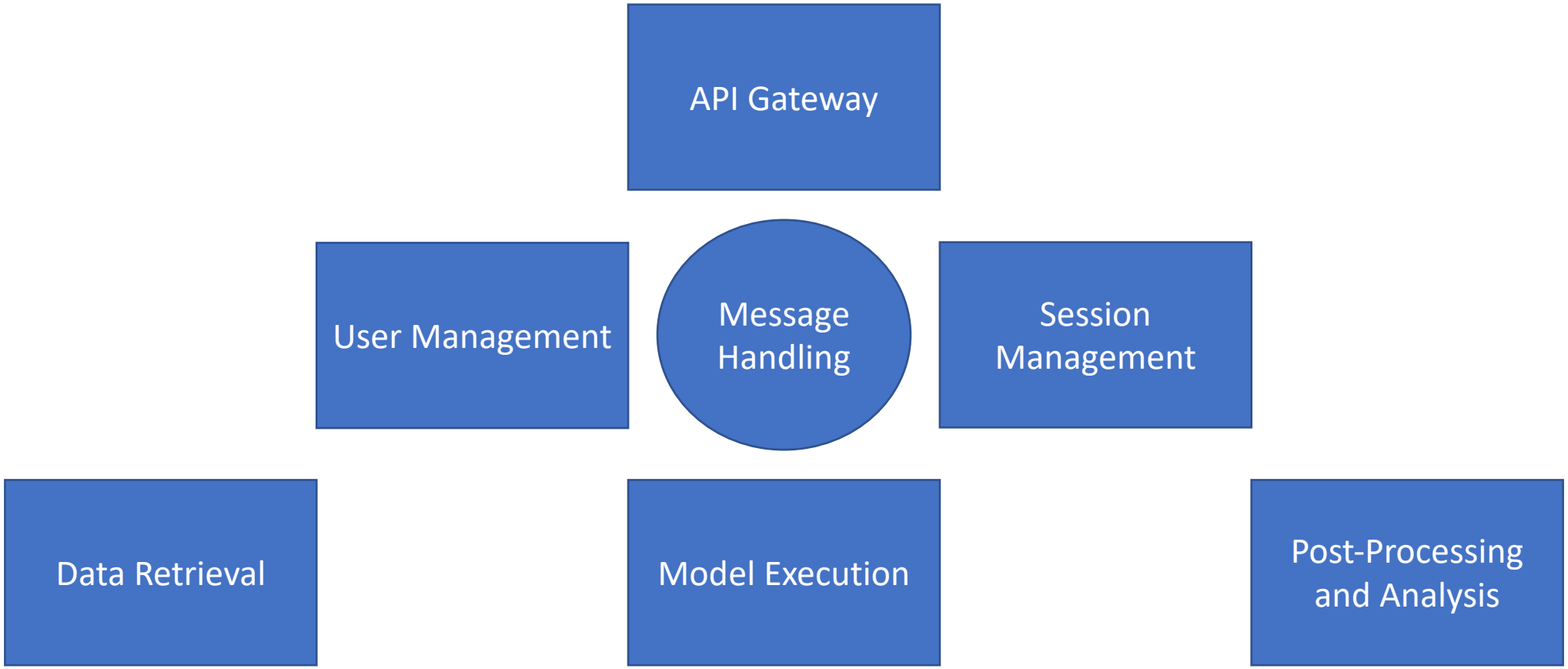
Build for Expandability

Science gateways are complicated.

- Your existing services will need to be enhanced
- There are a lot more services that you will need to add over time

Conclusions:

- You need a way for services to talk to each other that is not hard-coded
- You need to hide the details of the internal services behind an API gateway


API Gateway
User Management
Message Handling
Session Management
Data Retrieval
Model Execution
Post-Processing and Analysis

Last Thing for our Minimal Gateway: User Interface

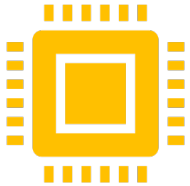
Users don’t interact with backend services directly. They use the API server

The UI itself will also need to evolve

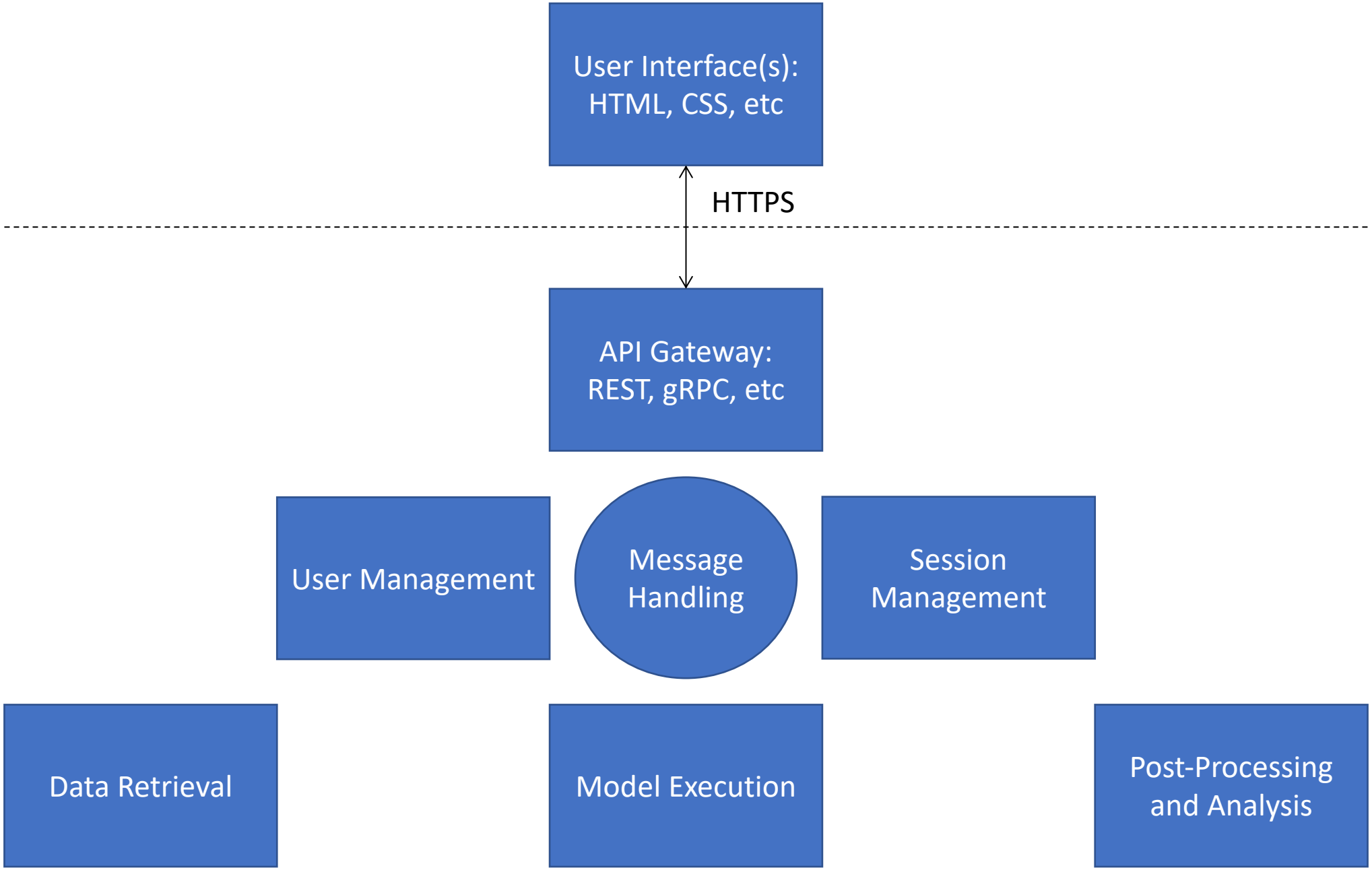
Support a richer user experience

Support different types of researchers

Support new features as they are added to the system



If you design your infrastructure thoughtfully, you can have more than one type of gateway interacting with your middleware.


User Interface(s):
HTML, CSS, etc
HTTPS
API Gateway:
REST, gRPC, etc
User Management
Message
Handling
Session
Management
Data Retrieval
Model Execution
Post-Processing
and Analysis

Assignment Requirements

- Each of the components in the previous diagram is a microservice.
  - Each must run as a separate process.
- You must use at least 3 different programming languages
  - For example, one service in Python, one in Go, one in Java, ...
- You must use at least one DB technology
  - Only one microservice can connect to each DB
- You must choose and implement an internal communication strategy for your microservices
- You must define your API based on this lecture and other discussions
- Prototype your continuous integration and deployment
  - Your entire system must be easily deployable by your peers, graders, and instructors

And Remember…

- This is a mock-up gateway
  - Components are placeholders for the real implementations for running jobs on supercomputers, executing machine learning applications, etc.
- Use Test Driven Design-like approaches
- “Fake it ‘til you make it”