Lecture 15: Vertex AI deployment and pipelines

AC215

Pavlos Protopapas
SEAS/Harvard

Announcements

- Midterm Presentations - 10/31

  Schedule and Location

- **Submit slides on Canvas by 10/31 - Noon**

- Submit commit hash on Canvas by 10/31 - 9PM

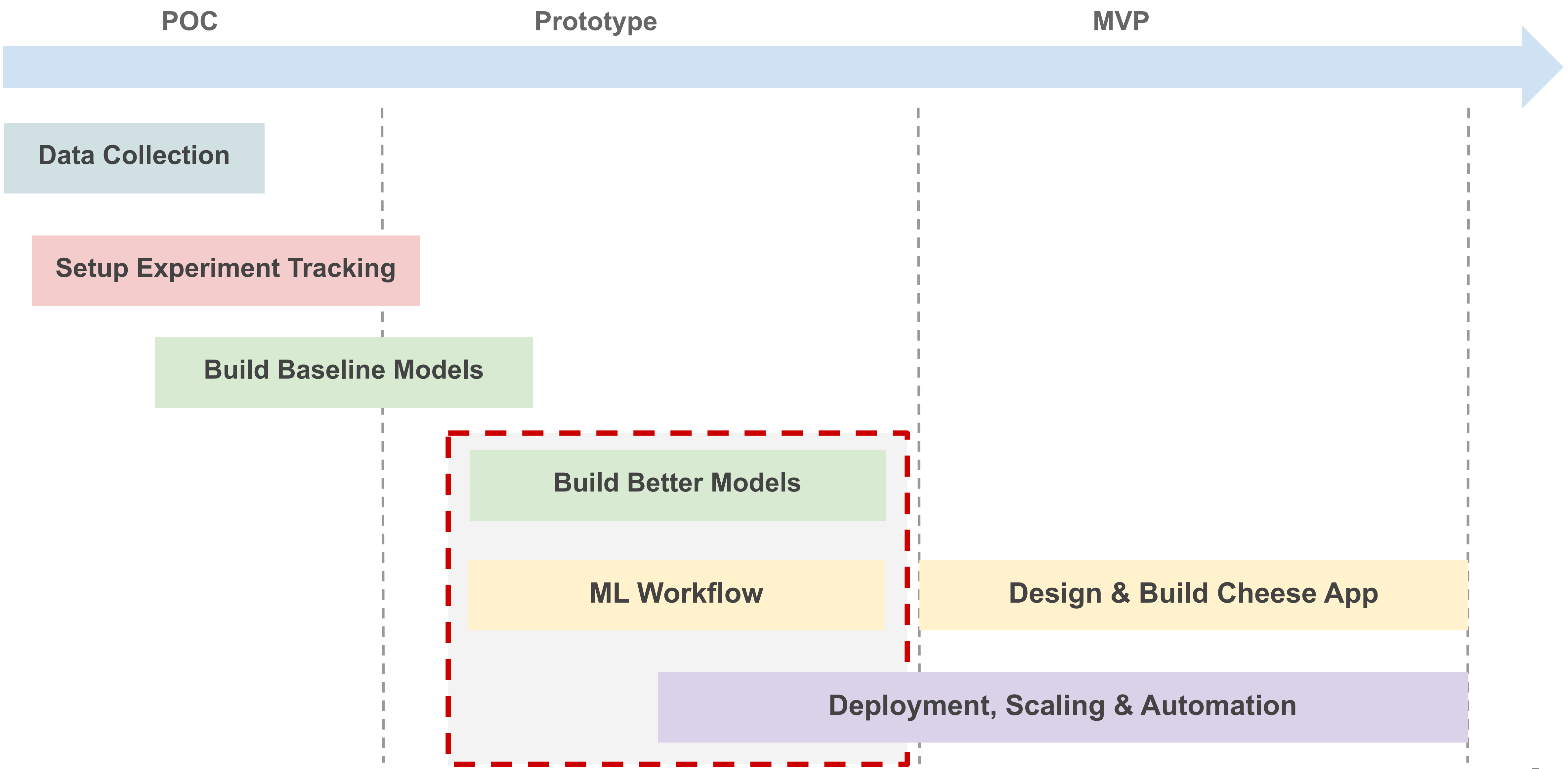
Outline

1. Recap
2. Serverless: Model Deployment with Vertex AI Online Prediction
3. ML Workflow Management
4. Vertex AI Pipelines

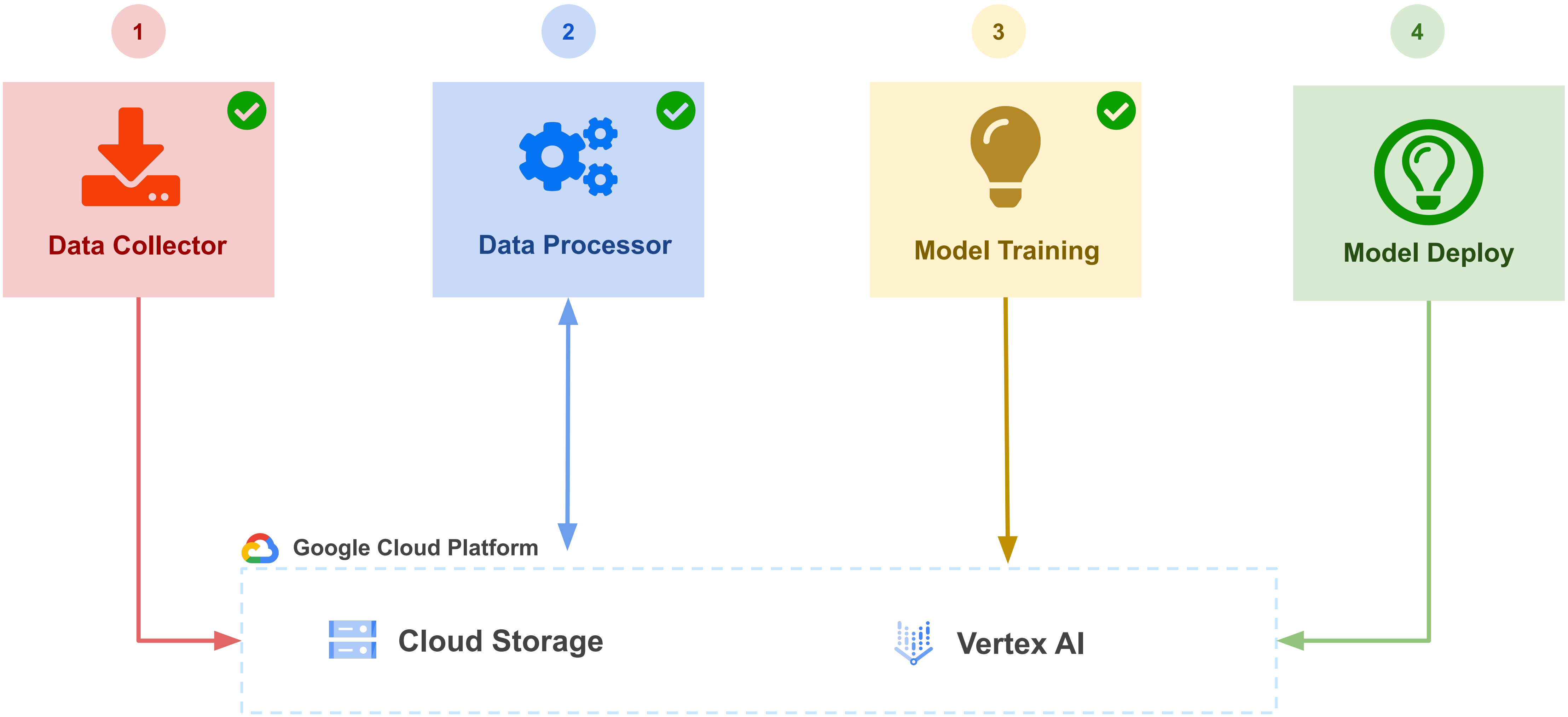
Outline

1. **Recap**
2. Serverless: Model Deployment
3. ML Workflow Management
4. Vertex AI Pipelines

Recap: Cheese App Status



Cheese App Development



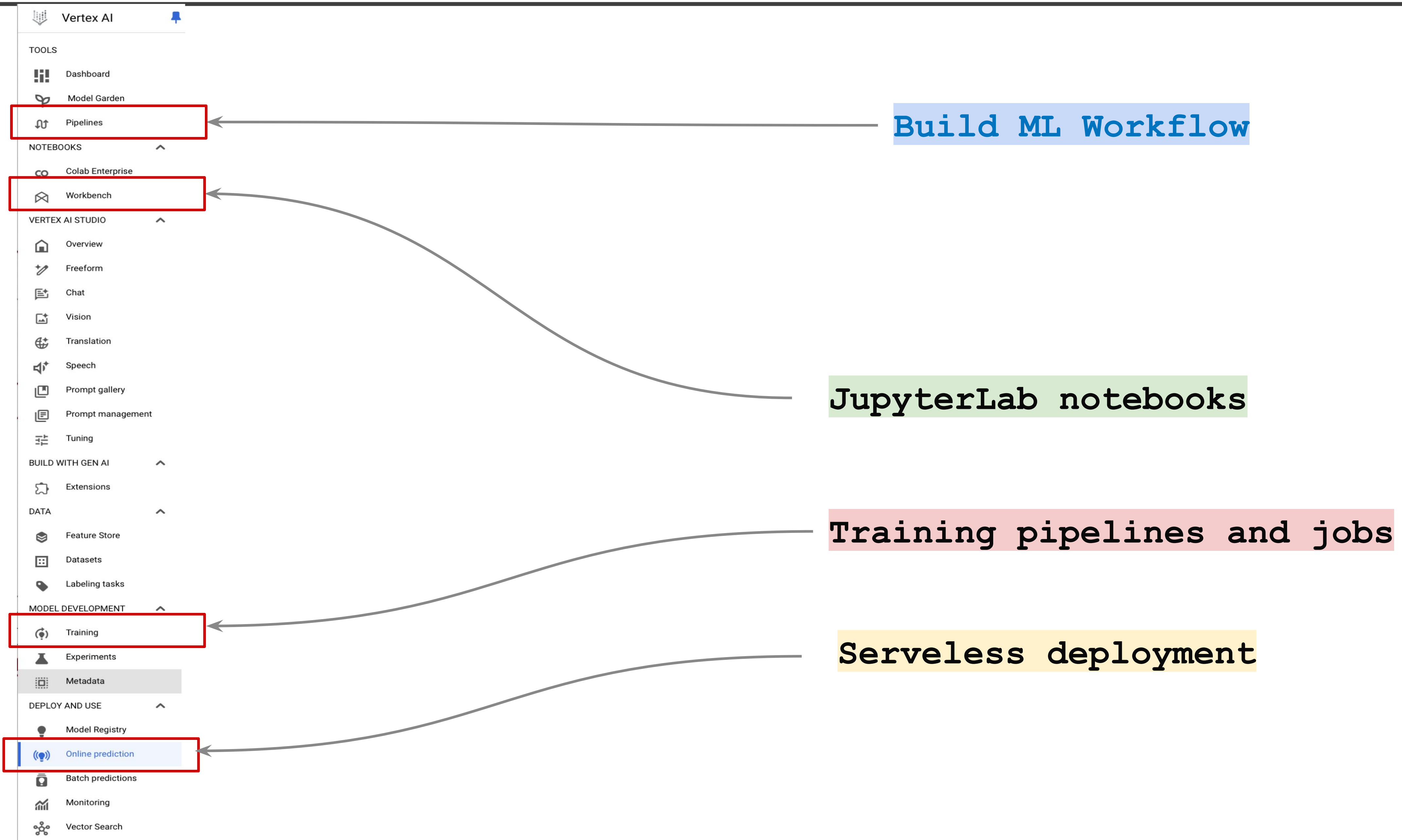
Recap

1. Serverless: Cloud Functions
2. Serverless: Cloud Runs

Outline

1. Recap
2. **Serverless: Model Deployment with Vertex AI Online Prediction**
3. ML Workflow Management
4. Vertex AI Pipelines

Vertex AI

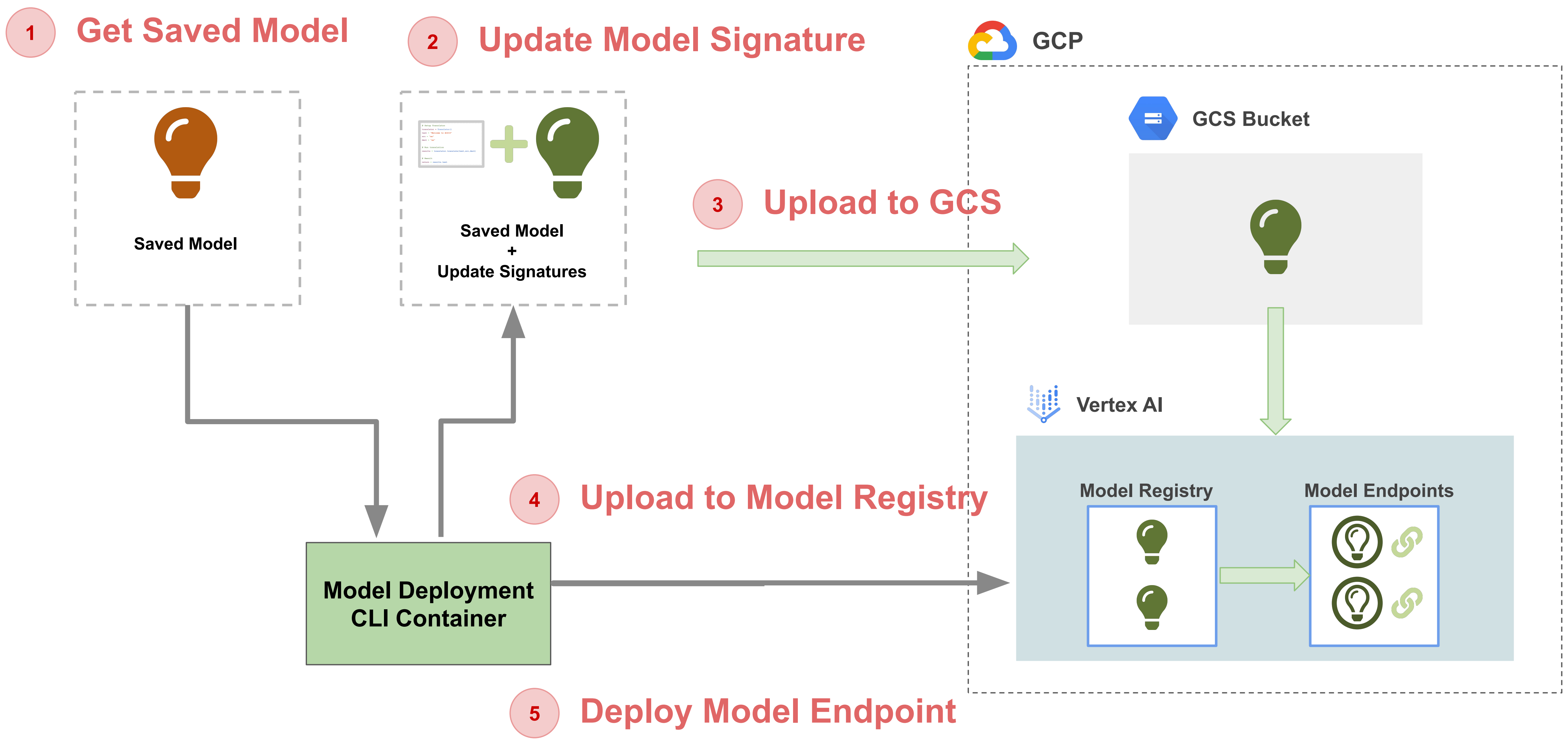

Vertex AI
TOOLS
Dashboard
Model Garden
Pipelines
NOTEBOOKS
Colab Enterprise
Workbench
VERTEX AI STUDIO
Overview
Freeform
Chat
Vision
Translation
Speech
Prompt gallery
Prompt management
Tuning
BUILD WITH GEN AI
Extensions
DATA
Feature Store
Datasets
Labeling tasks
MODEL DEVELOPMENT
Training
Experiments
Metadata
DEPLOY AND USE
Model Registry
Online prediction
Batch predictions
Monitoring
Vector Search
Build ML Workflow
JupyterLab notebooks
Training pipelines and jobs
Serveless deployment

Serverless Model Deployment with **Vertex AI Online Prediction**

What is serverless model deployment?

- Deploy our trained model for predictions with no servers.
- Setup **online** or **batch** prediction modes
- For **online** predictions there is an ongoing cost
- Access GPU or CPU hardware for inference
- Scale out easily
- Alert: Continuous cost to keep endpoint up

Serverless Model Deployment



Serverless Model Deployment: Model Signature

Why do we need to update the model signature?

- Make model input to accept a raw image
- Perform data preprocessing steps prior to model inference
- Combine data preprocessing & model inference in one endpoint

Serverless Model Deployment: Update Model Signature

```python
# Preprocess Image
def preprocess_image(bytes_input):
    resized = ...
    return resized

# Define tf functions
@tf.function(input_signature=[tf.TensorSpec([None], tf.string)])
def preprocess_function(bytes_inputs):
    decoded_images = tf.map_fn(
        preprocess_image, bytes_inputs, dtype=tf.float32, back_prop=False
    )
    return {"model_input": decoded_images}

@tf.function(input_signature=[tf.TensorSpec([None], tf.string)])
def serving_function(bytes_inputs):
    images = preprocess_function(bytes_inputs)
    results = model_call(**images)
    return results

# Update model signature and save
tf.saved_model.save(
    prediction_model,...,
    signatures={"serving_default": serving_function},
)
```

Define preprocessing function

Define @tf.function for new model signature

Save Model with the new model signature

Tutorial: Serverless Model Deployment

Steps to perform **Serverless Model Deployment** on cheese classification model:

- Create a GCS bucket to store saved model.
- Update Model Serving Signature
- Upload Model to Vertex AI Model Registry.
- Deploy Model as an Endpoint.
- For detailed instructions, please refer to the following link
  - [Serverless Model Deployment](https://github.com/dlops-io/model-deployment). (https://github.com/dlops-io/model-deployment)
  - [View Model Endpoints](https://console.cloud.google.com/vertex-ai/online-prediction/endpoints). (https://console.cloud.google.com/vertex-ai/online-prediction/endpoints)
  - [View Model Registry](https://console.cloud.google.com/vertex-ai/models). (https://console.cloud.google.com/vertex-ai/models)

Vertex AI Online Prediction vs. Cloud Run for Model Deployment

Feature	Vertex AI Online Prediction	Cloud Run
Purpose	Managed ML model serving for real-time predictions	General-purpose serverless platform for containerized apps
Ease of Deployment	Deploys models directly, minimal setup	Requires containerization, more configuration
Infrastructure	Fully managed, ML-specific infrastructure with autoscaling and load balancing	Autoscaling serverless platform, not specific to ML
Model Monitoring	Built-in model performance monitoring and data drift detection	No built-in ML monitoring; requires custom implementation
Scaling	Optimized for real-time ML workloads, automatic scaling for high availability	Serverless scaling based on HTTP traffic
Model Versioning	Supports versioning, A/B testing for models	No native support for model versioning
Batch Prediction Support	Yes, with both batch and online prediction capabilities	No specific batch prediction feature; needs custom setup
Framework Support	TensorFlow, PyTorch, XGBoost, and other ML frameworks	Framework-agnostic, supports any containerized code
Security and Permissions	Integrated with IAM for granular access control	Integrated with IAM but without ML-specific controls
Best For	Real-time ML model serving with minimal infrastructure management	Custom APIs and services that may include ML models

Outline

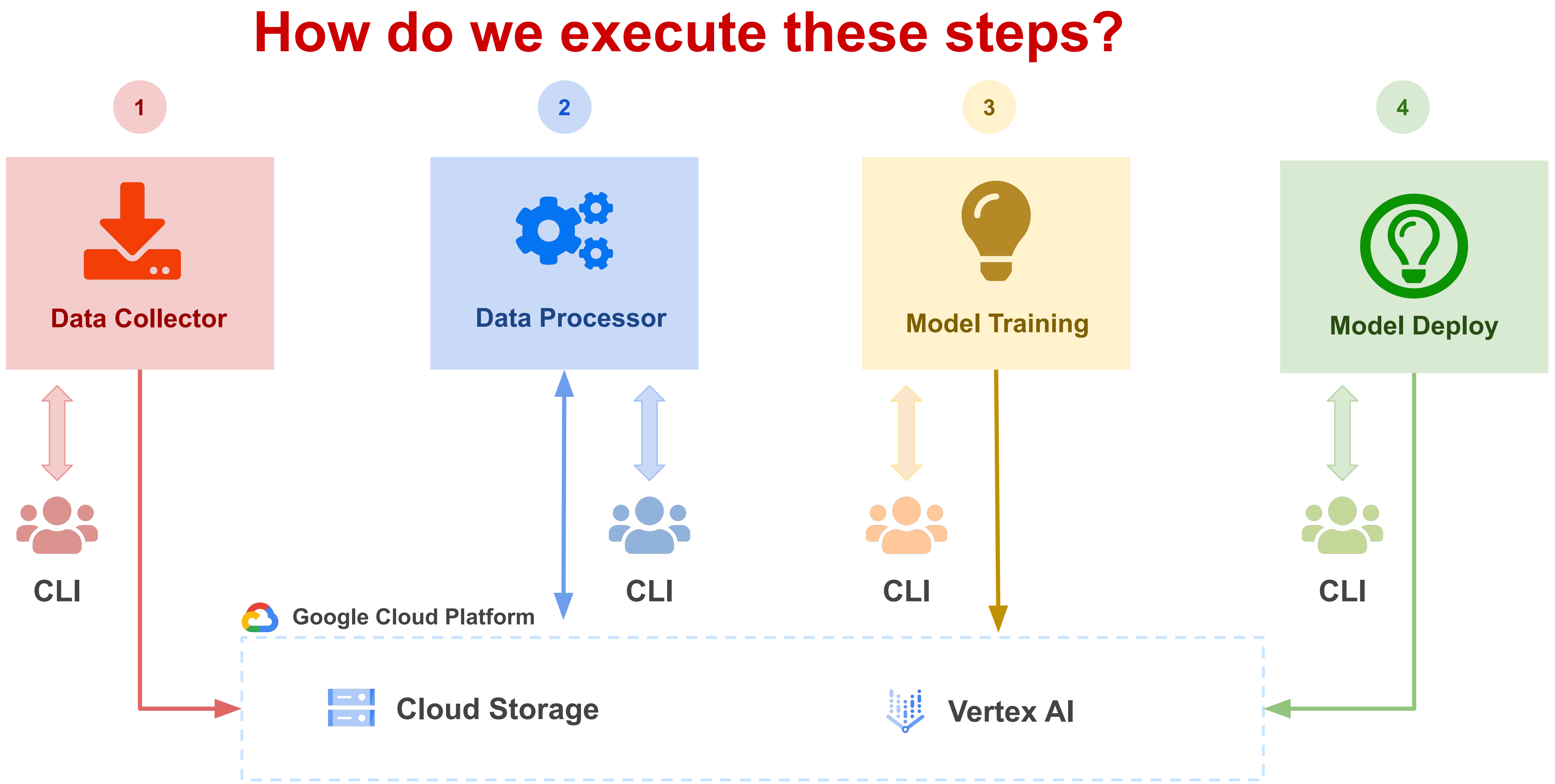
1. Recap
2. Serverless: Model Deployment
3. **ML Workflow Management**
4. Vertex AI Pipelines

ML Workflow Management

What is ML workflow management?

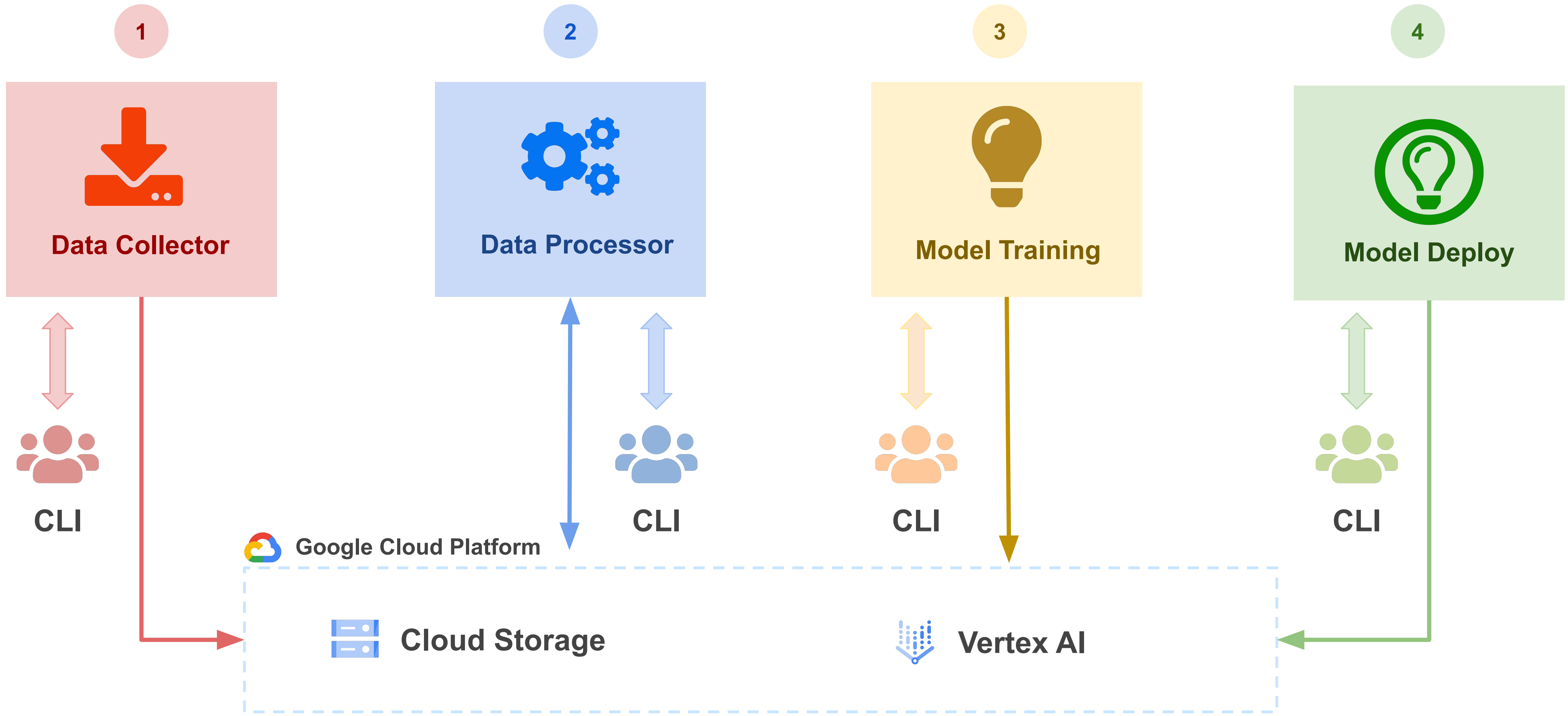
- Helps us efficiently manage end-to-end ML tasks from data collection to model deployment
- Helps orchestrate various and automated pipeline execution
- Manages collaboration, integration, and scalability

ML Workflow: Cheese App



ML Workflow: Cheese App

Can we automate this flow? → **Vertex AI Pipelines**



Outline

1. Recap
2. Serverless: Model Deployment
3. ML Workflow Management
4. **Vertex AI Pipelines**

Vertex AI Pipelines

What is Vertex AI Pipelines?

- Vertex AI is machine learning platform offered by Google in GCP.
- Vertex AI Pipelines helps you to automate, monitor, and govern your ML components by orchestrating your ML workflow in a serverless manner

Building Vertex AI Pipelines

```
# Import Kubeflow Pipelines
from kfp import dsl
# Define Components
@dsl.component
def square(x: float) -> float:
    return x**2
@dsl.component
def add(x: float, y: float) -> float:
    return x + y
@dsl.component
def square_root(x: float) -> float:
    return x**0.5
# Define Pipeline
@dsl.pipeline
def sample_pipeline(a: float = 3.0, b: float = 4.0) -> float:
    a_sq_task = square(x=a)
    b_sq_task = square(x=b)
    sum_task = add(x=a_sq_task.output, y=b_sq_task.output)
    return square_root(x=sum_task.output).output
```

Import kubeflow pipeline

Define pipeline components

Define Pipeline, an orchestration of how you want your component tasks to run

Building Vertex AI Pipelines

Define Pipeline, an orchestration of how you want your component tasks to run

```
...

# Define Pipeline
@dsl.pipeline
def sample_pipeline(a: float = 3.0, b: float = 4.0) -> float:
    a_sq_task = square(x=a)
    b_sq_task = square(x=b)
    sum_task = add(x=a_sq_task.output, y=b_sq_task.output)
    return square_root(x=sum_task.output).output

# Build yaml file for pipeline
compiler.Compiler().compile(
    sample_pipeline, package_path="sample-pipeline.yaml"
)
```

Compile pipeline into a yaml file

Running Vertex AI Pipelines

```python
# Initialize GCP
import google.cloud.aiplatform import aip

# Create a Pipeline Job in Vertex AI
job = aip.PipelineJob(
    display_name=DISPLAY_NAME,
    template_path="sample-pipeline1.yaml",
    pipeline_root=PIPELINE_ROOT,
    enable_caching=False,
)

# Run the Pipeline Job
job.run()
```

Import Google Cloud SDK

Create a pipeline job

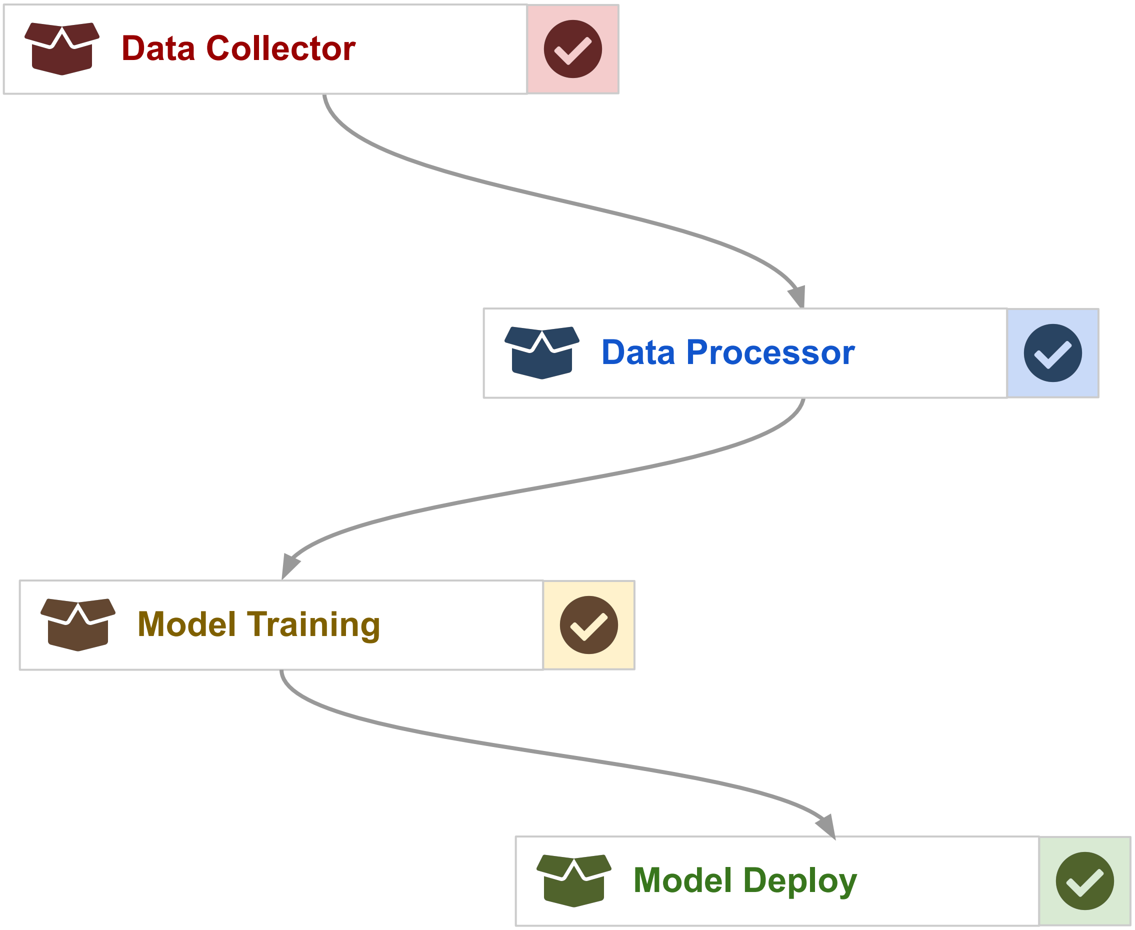
Run pipeline job in Vertex AI

Building Vertex AI Pipelines

Steps to build pipelines for your custom containers

- Make your containers callable
- Build & Push Container Images to a Container Registry
- Define a sequence of steps using a directed acyclic graph (DAG)

Building Vertex AI Pipelines



1. Download images
2. Uploads to GCP

1. Verify images
2. Check for duplicates
3. Convert to TF Records

1. Train model
2. Save model

1. Upload to Registry
2. Deploy model Endpoint

Making Container Callable

Dockerfile

```
# Use the official Debian-hosted ...
FROM python:3.9-slim-buster
.
.
.
# Add the rest of the source code.
RUN --chown=app:app . /app
# Entry point
ENTRYPOINT ["pipenv", “shell”]
```

Dockerfile

```
# Use the official Debian-hosted ...
FROM python:3.9-slim-buster
.
.
.
# Add the rest of the source code.
RUN --chown=app:app . /app
# Entry point
ENTRYPOINT ["/bin/bash","./docker-entrypoint.sh"]
```

Change entrypoint to a shell file

Making Container Callable

Development mode:
Authenticated to GCP
pipenv shell to test cli inside container

docker-entrypoint.sh

```
#!/bin/bash
args = "$@"

if [[ -z ${args} ]];
then
    # Authenticate gcloud using service account
    gcloud auth activate-service-account --key-file $GOOGLE_APPLICATION_CREDENTIALS
    # Set GCP Project Details
    gcloud config set project $GCP_PROJECT
    pipenv shell
else
  pipenv run python $args
fi
```

Production mode:
Run container using "docker run ... cli.py –search"

Tutorial: Vertex AI Pipelines

Steps to build **Vertex AI Pipelines** on the cheese app ML workflow components:

- Make Containers Callable.
- Build & Push Image.
- Build ML Pipeline.
- Run Pipeline in Vertex AI
- For detailed instructions, please refer to the following link
  - Cheese App Workflows. (https://github.com/dlops-io/ml-workflow#cheese-app-ml-workflow-management)
  - View Vertex AI Pipelines. (https://console.cloud.google.com/vertex-ai/pipelines/runs)


#RAW
DELETE!!
LIVE


#FALLONTONIGHT
Students
Prof. Pavlos