Aalto University
School of Science

Coordination Models and Techniques for Machine Learning Systems

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, *https://rdsea.github.io*

CS-E4660 Advanced Topics in Software Systems, Fall 2024
02/10/2024

Learning objectives

- Analyze the role of coordination techniques, their complexity and diversity in ML systems

- Understand and apply orchestration models, common tools and design patterns

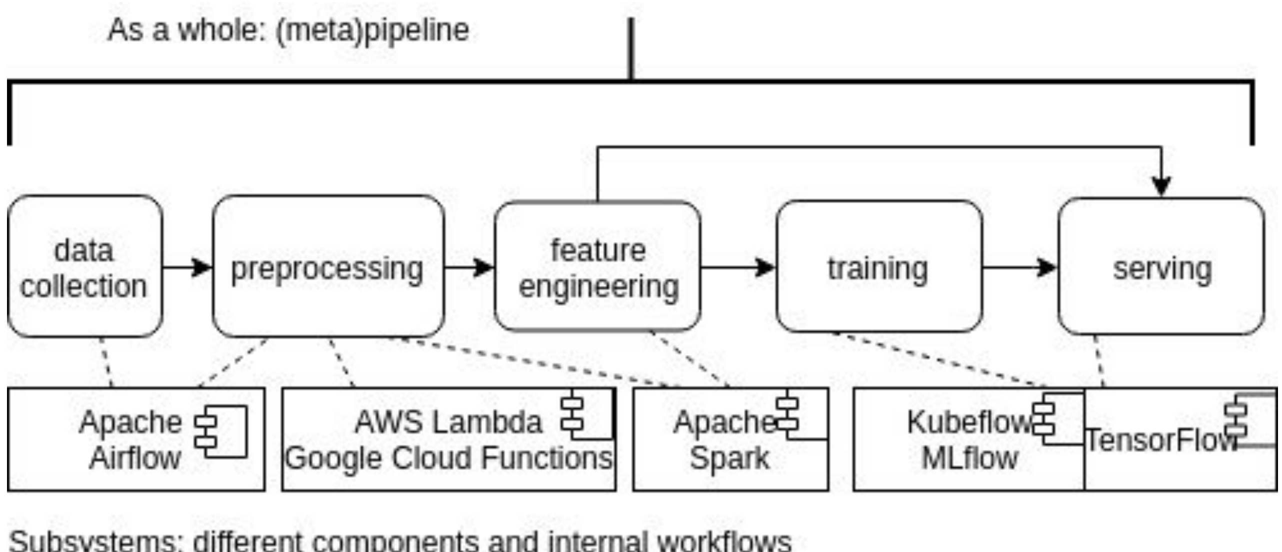
- Understand and apply choreography models, common tools and design patterns

- Understand, define and develop ML model serving

Coordination complexity and diversity

Multi-level pipelines in big data/ML systems

- **Meta-workflow or -pipeline**
- **Inside each phase: pipeline/workflow or other types of programs**



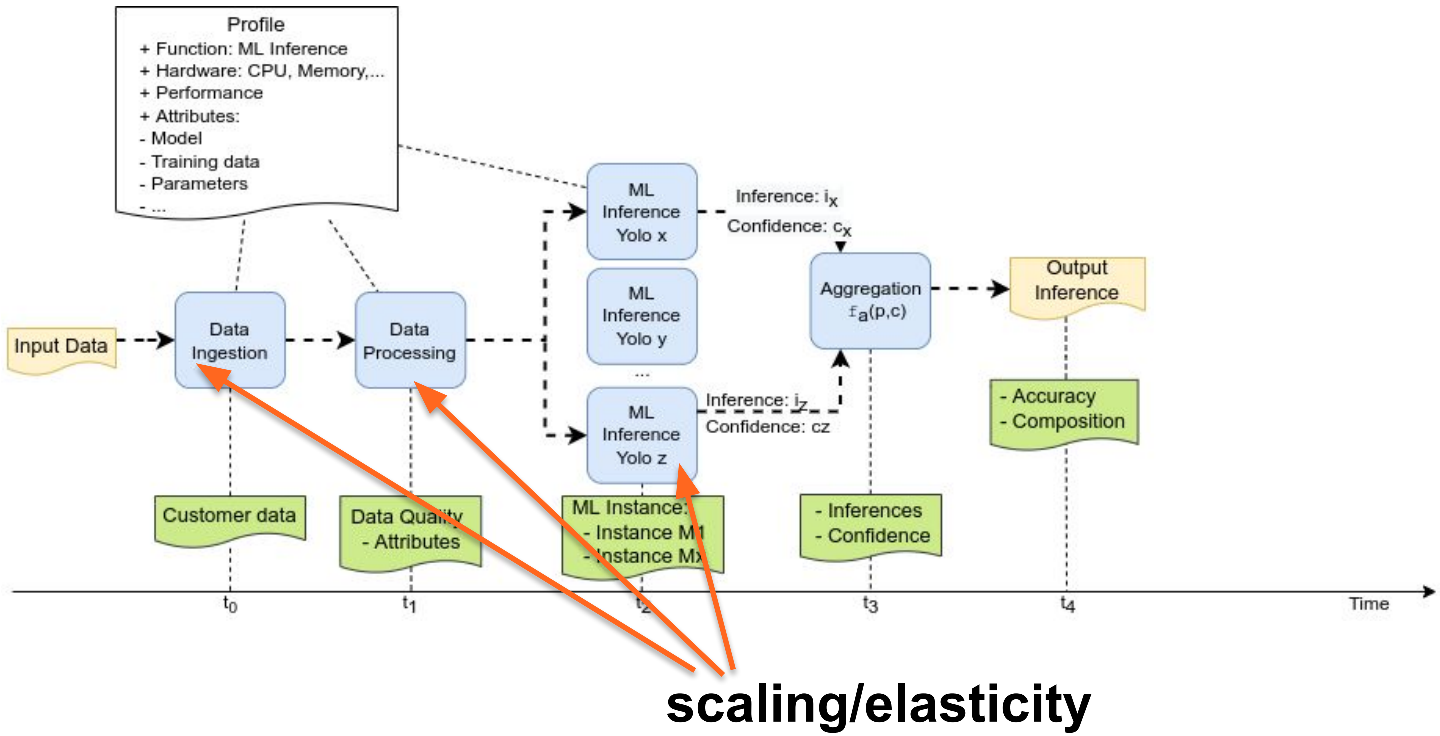
Coordination

- **Many tasks must be coordinated**
  - data analytics, ML inference, service deployment, etc.
  - different implementation details and external requirements

- **How do we arrange tasks? in which order?**

- **How do we manage tasks at runtime, including failure recovery?**

Think about some key metrics, like high throughput and service time for inferences. If you want your service to be fast? What will you do?



Where do we need “coordination” in ML systems? e.g., scaling/elasticity

- **Scale and control data processing**
  - data preparation/movement
  - feature engineering
- **Scale and control training and serving tasks**
- **Dynamic serving**
  - manage loads and ML models
- **Scaling needs monitoring**
  - logging, tracing, monitoring of infrastructures, consumer requests and ML/big data tasks
- **Scaling needs coordination**
  - orchestration or choreography techniques

Main issues related to coordination

- **Differences between the coordination of computing phases and of tasks in a system**
  - execution model: locality, dependencies and granularity
- **Different types of software artefacts and resources for ML systems**
  - management and on-demand provisioning
- **Distributed computing in training, testing and experimenting**
  - trial computing configurations, inputs/results collection
- **Various observability and layers related to R3E for the pipeline execution**
  - end-to-end R3E requires coordination

W3H: what, when, where and how for coordination

Where: within a phase, across phases, within a component, a subsystem, etc.

What: preparing data and machines, performing inferences, carrying out observability

When: triggered by data flows or control flows or events? Internal triggers vs external triggers

Coordination

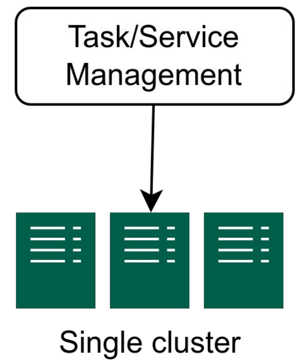
How: which tools, models?

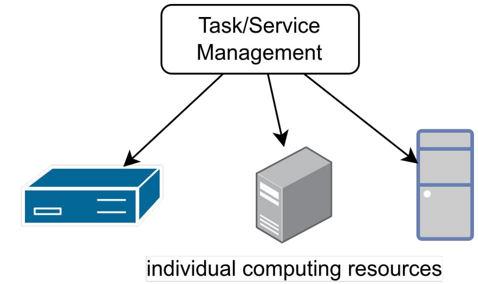
Aalto University
School of Science

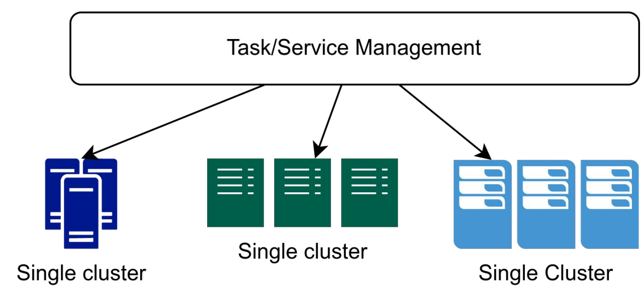
Coordination models & techniques

Resource views

- **Tasks/services**
  - deployed and run atop computing resources
- **Using multiple computing resources:**
  - single cluster of machines
  - multiple of clusters of machines
  - set of computing machines
- **Resource roles: worker nodes, head node/controller, support nodes, etc.**



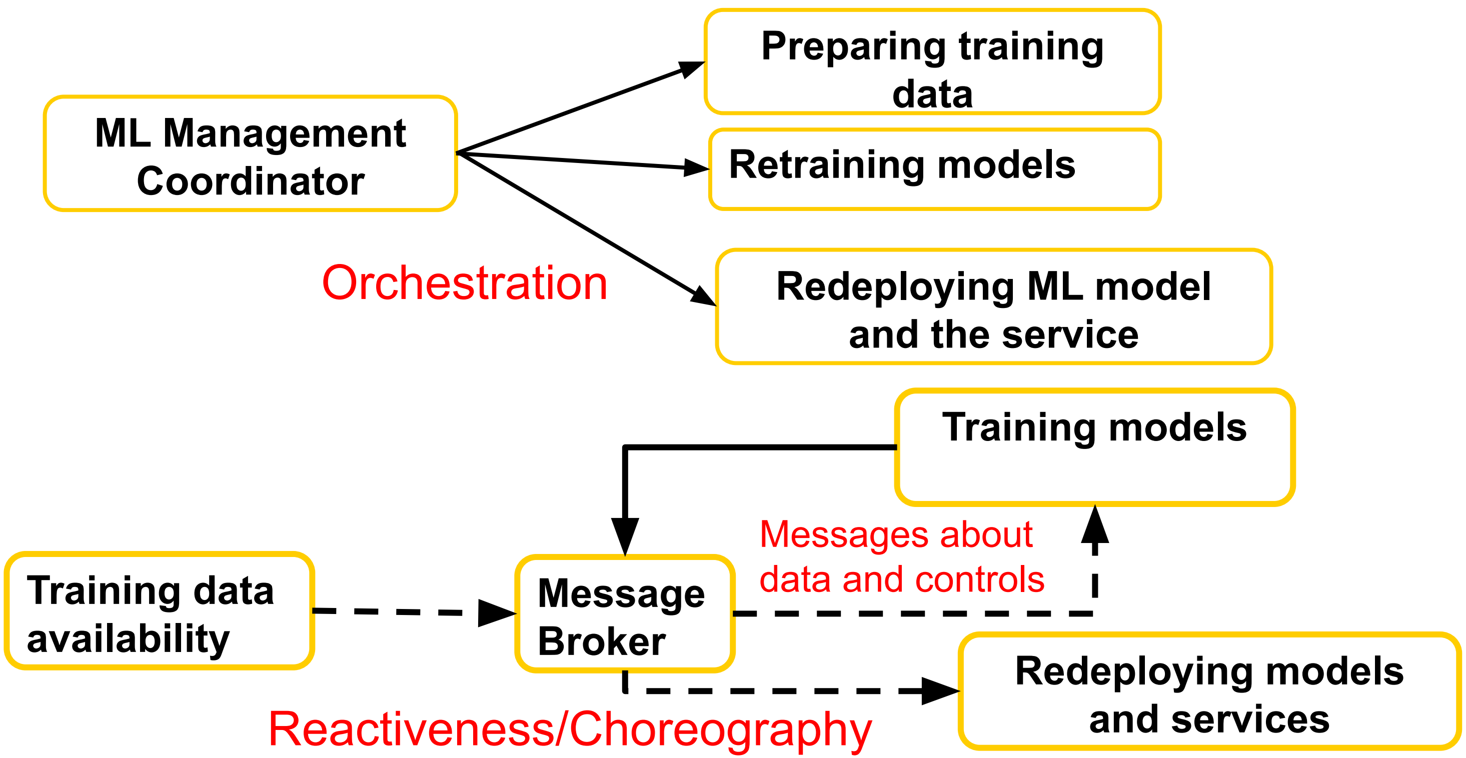




Coordination styles

- **Coordination models**
  - orchestration and reactiveness/choreography
- **Orchestration**
  - task graphs and dependencies are based on control or data flows
  - dedicated orchestrator
    - *tasks triggered based on completeness of other tasks or the availability of data*
  - often implemented as workflows
- **Reactiveness/choreography**
  - follow reactive model
    - *tasks are reacted/triggered based on messages*

Orchestration and reactiveness



Aalto University
School of Science

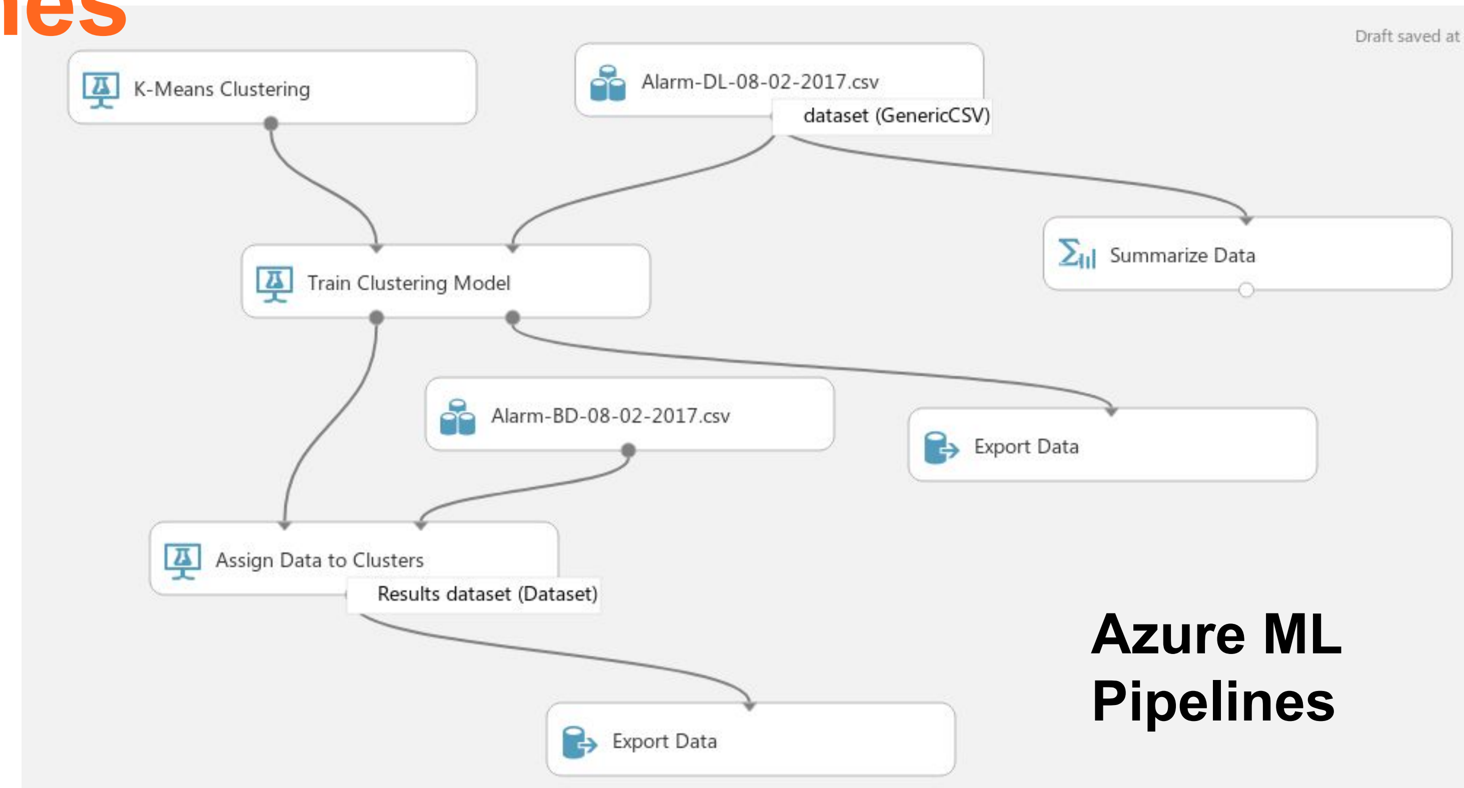
Coordination with workflow techniques - orchestration

The orchestration style

Orchestration architectural style: design

- **Workflow architectures are well-known**
  - Big Data/ML systems: leverage many types of services and cloud technologies
- **Required components**
  - workflow/pipeline specifications/languages (also UI)
  - data and computing resource management, external services
  - orchestration engines (with different types of schedulers)
- **Execution environments**
  - cloud platforms (e.g., VMs, containers, Kubernetes)
  - heterogeneous computing resources (PC, servers, Raspberry PI, etc.)

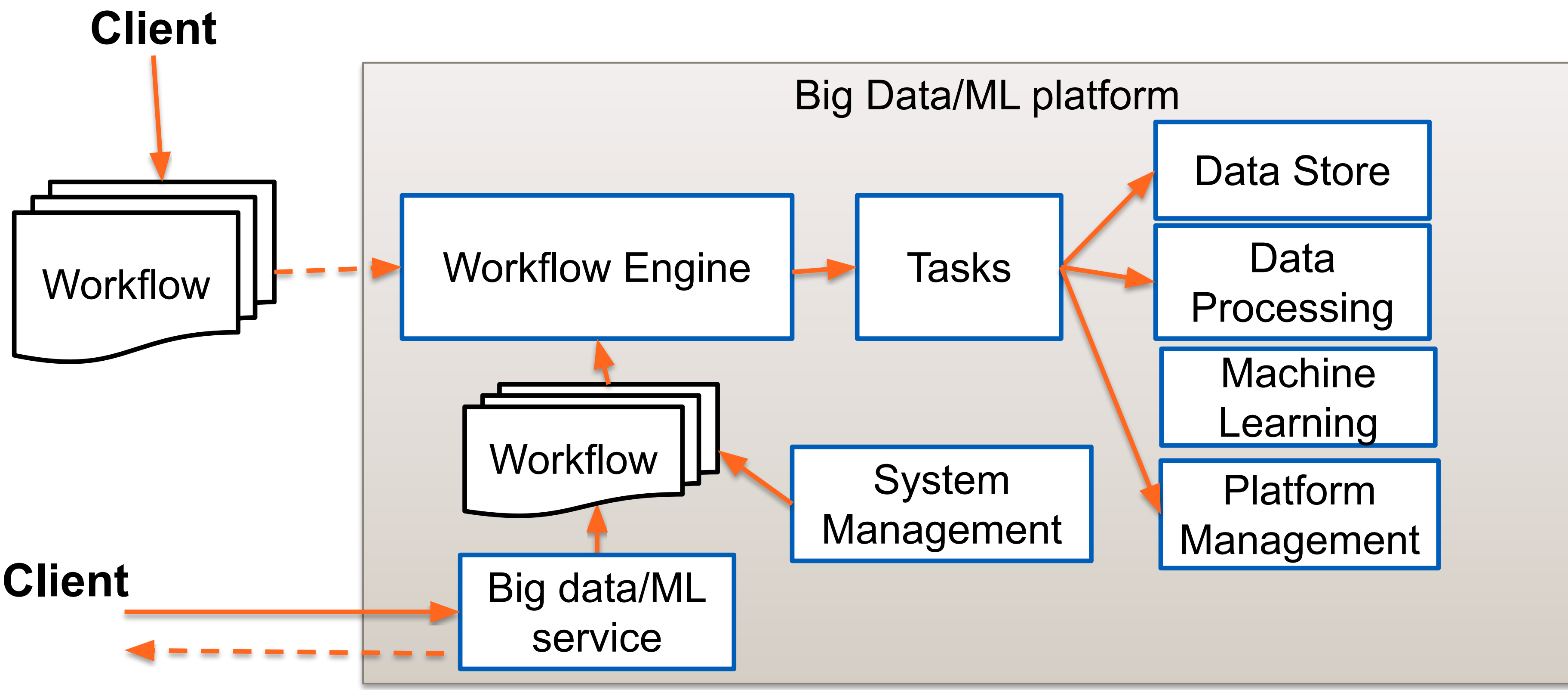
Example: workflow used in ML pipelines



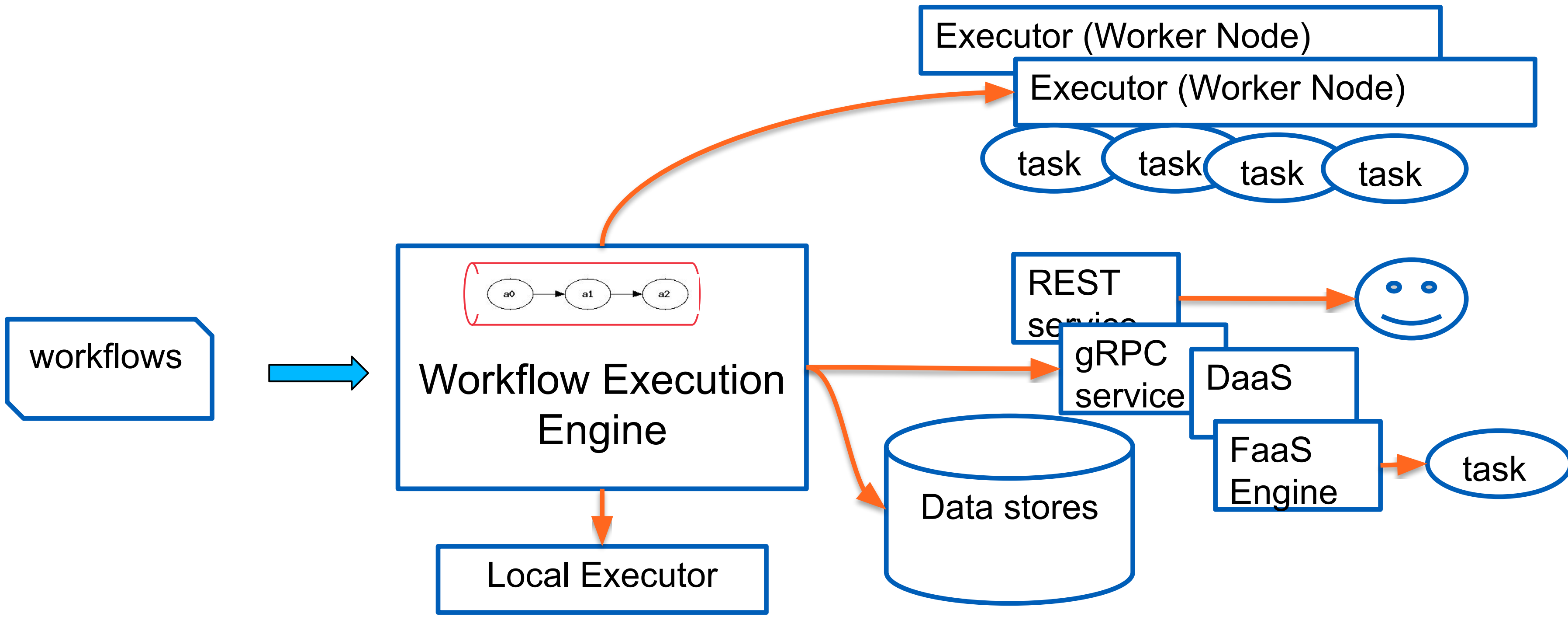
So what is behind the scene?

Azure ML Pipelines

Workflows in big data/ML systems



Common workflow execution models



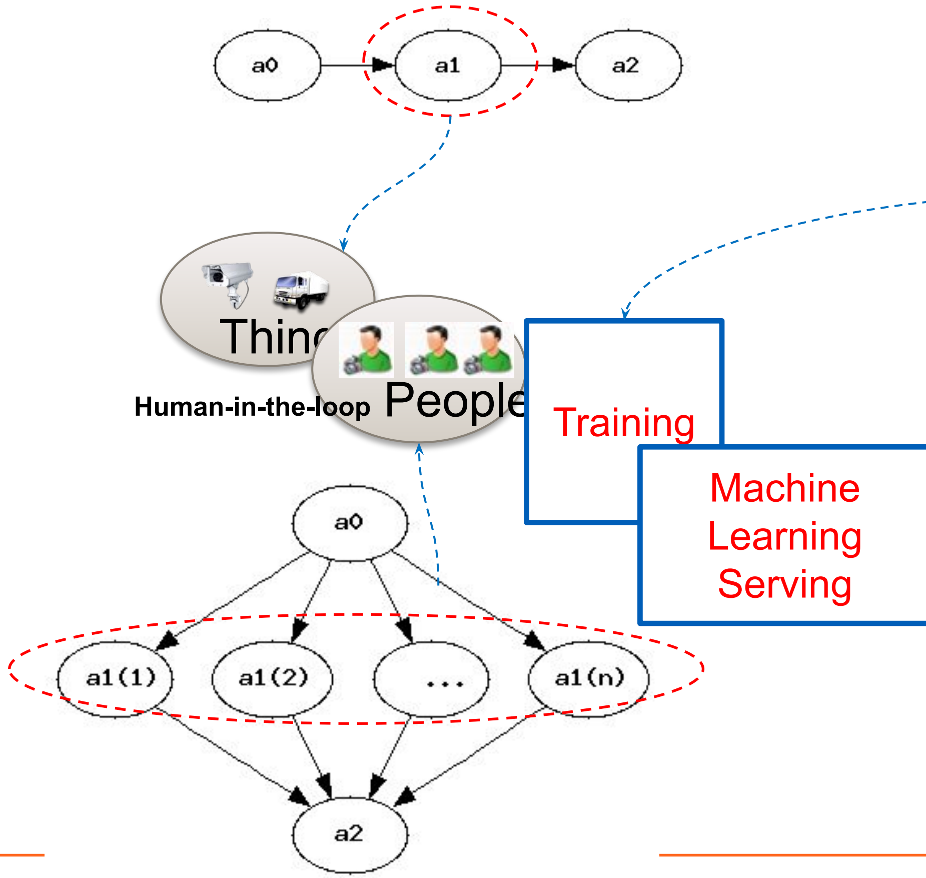
Executors: containers, common OS processes, Spark, …

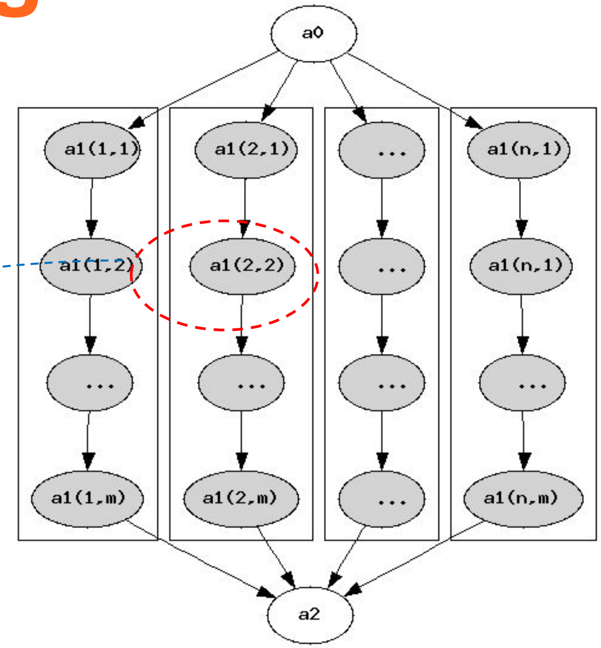
Resource management: Kubernetes, OpenStack, Batch Job Scheduler

Key components

- **Tasks/activities**
  - describe a single work (it does not mean small)
  - tasks can be carried out by humans, executables, scripts, batch applications, stream applications, and Web services
- **Workflow languages**
  - how to structure/describe tasks, dataflows, and control flows
- **Workflow engine**
  - execute the workflow by orchestrating tasks
  - usually call remote services to run tasks

Tasks orchestration, e.g. in ML





Things: edge ML/tinyML
People: Human-in-the-ML loop

Runtime aspects

- **Parallel and distributed execution**
  - tasks are deployed and running in different machines
  - multiple workflows can be running in the system (multi-tenancy)
- **Long running for machine learning, data analysis and computation**
  - can be hours! ⇒ resilient, debugging, logging
- **Maybe short for control flows/decision**
- **Checkpoint and recovery, monitoring and tracking**
  - which tasks are running, where are they?
- **Data exchange**
- **Stateful management**
  - dependencies among tasks w.r.t control and data

Data exchange among tasks

- **Data and systems conditions**
  - big files/big dataset, small/fast data (e.g., in realtime ML), etc.
  - shared nothing or not among computing resources
- **Some mechanisms**
  - shared file systems/data volume
    - *can be read/write only or one or many*
  - middleware: object/blob storage (like S3 style)
  - collective communications among tasks – using high-level libraries (e.g., Gloo, MPI, NCCL)
  - direct exchange (e.g., known sender/receiver)
  - messaging

Describing workflows

- **Programming languages with procedural code**
  - general- and specific-purpose programming languages, such as Java and Python
  - common ways in big data and ML platforms
  - low-level programming, suitable for programmer
- **Descriptive languages with declarative schemas**
  - BPEL, YAML, and several languages designed for specific workflow engines
  - common in business, scientific and data science workflows
  - YAML is also popular for big data/ML workflows in native cloud environments

Generic workflow frameworks

- **Generic workflows**
  - use to implement different tasks, such as data processing, machine provisioning, service calls, data retrieval
- **Examples:**
  - Airflow (https://airflow.apache.org), Argo Workflows (https://argoproj.github.io/argo), Prefect (https://www.prefect.io), Uber Cadence (https://github.com/uber/cadence), Temporal IO (https://temporal.io/), Kedro (https://docs.kedro.org/), Tekton (https://tekton.dev/)
- **Serverless-based workflows implemented in different tools**
  - E.g., Amazon Step Function, Alibaba Cloud Serverless Workflow, CNCF Serverless Workflow

Specific workflow frameworks

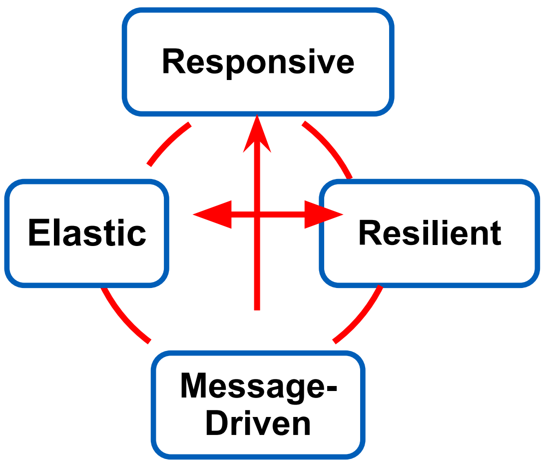
- **Specific workflows for specific ML purposes**
- **Examples**
  - Kubeflow (https://www.kubeflow.org/docs/components/pipelines/v2/introduction/)
  - MLRun (https://docs.mlrun.org/)
  - ZenML (https://www.zenml.io/)

Coordination techniques with messaging - choreography

The reactiveness style

Choreography: reactive systems for Big Data/ML

Reactive systems



Source: https://www.reactivemanifesto.org/

- Responsive: quality of services
- Resilient: deal within failures
- Elastic: deal with different workload and quality of analytics
- Message-driven: allow loosely coupling, isolation, asynchronous

Reactive systems for Big Data/ML: methods

- **Have different components as services**
  - components can come from different software stacks
  - components for doing computation as well as for data exchange
- **Elastic computing platforms**
  - platforms should be deployed on-demand in an easy way
- **Using messages to trigger tasks carried out by services**
  - messages for states and controls as well as for data
  - heavily relying on message brokers and lightweight triggers/controls (e.g., with serverless/function-as-a-service)

Which frameworks?

- **Low level messaging systems**
  - Kafka, RabbitMQ, ZeroMQ, Amazon SQS, ...
  - types of messages and semantics must be defined clearly
- **Triggers and controls**
  - the serverless/function-as-a-service model: trigger a function/task based on a message
    - *AWS Lambda, Google Cloud Function, Knative, Kubeless, OpenFaaS, Azure Functions*
    - *Use serverless function for "coordination"*
  - the worker model:
    - *light weighted microservices and job workers listening messages to trigger (remote) functions/tasks*
  - other:
    - *https://kestra.io/*

Diversity and complexity

- **Diversity**
  - so many tools/frameworks in a single big data/ML system

    ⇒ a single coordination model/tool might not be enough
  - there exist many coordination systems (included your specific implementation)

    ⇒ which ones should we select?
- **Complexity, due to the large-scale**
  - integration models with big data/ML components and infrastructures
  - runtime management: performance, failures, and states

Aalto University
School of Science

Coordination in ML

Using workflows and serverless for coordination

- **Training preparation**
  - before running a training: you move data from sources to stage, ship the code and prepare the computing environment
- **Coordination of ML phases**
  - do the coordination of three phases: data preprocessing, training and take the best model to deploy to a serving platform
  - automate the train -> test -> deployment (like in DevOps)
- **Experiment results gathering**
  - you run experiments in different places. There are several logs of results, you gather them and put the result into a database

Workflow for training

- **Prepare data**
  - move training data to the right place, e.g., with distributed resources
  - push or pull (e.g., from edge-cloud storage)
- **Run training tasks**
  - run training
  - store logs to (centralized) logging service
  - potentially perform incremental data download and upload
  - update experiment information to (centralized) experiment services
- **Clean up**
  - logs, data, etc

Examples: Kubeflow

- **End-to-end orchestration**
- **Pipeline orchestration is based on workflows**
  - Argo Workflow
- **Training and serving operator abstractions**
  - low level training/serving tasks via different frameworks

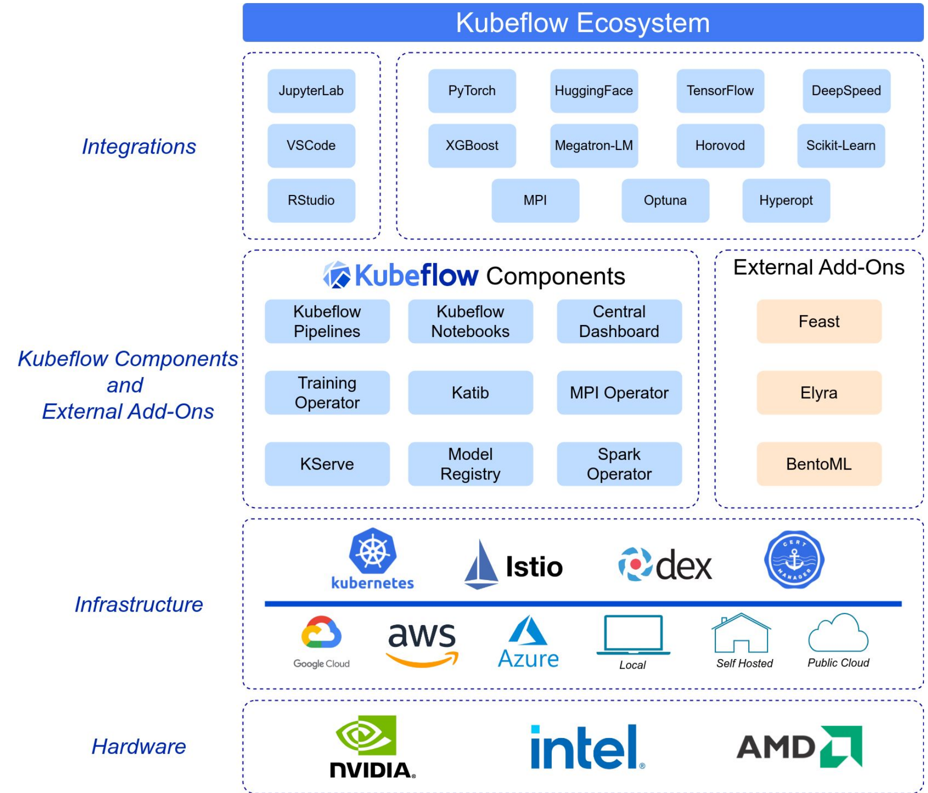
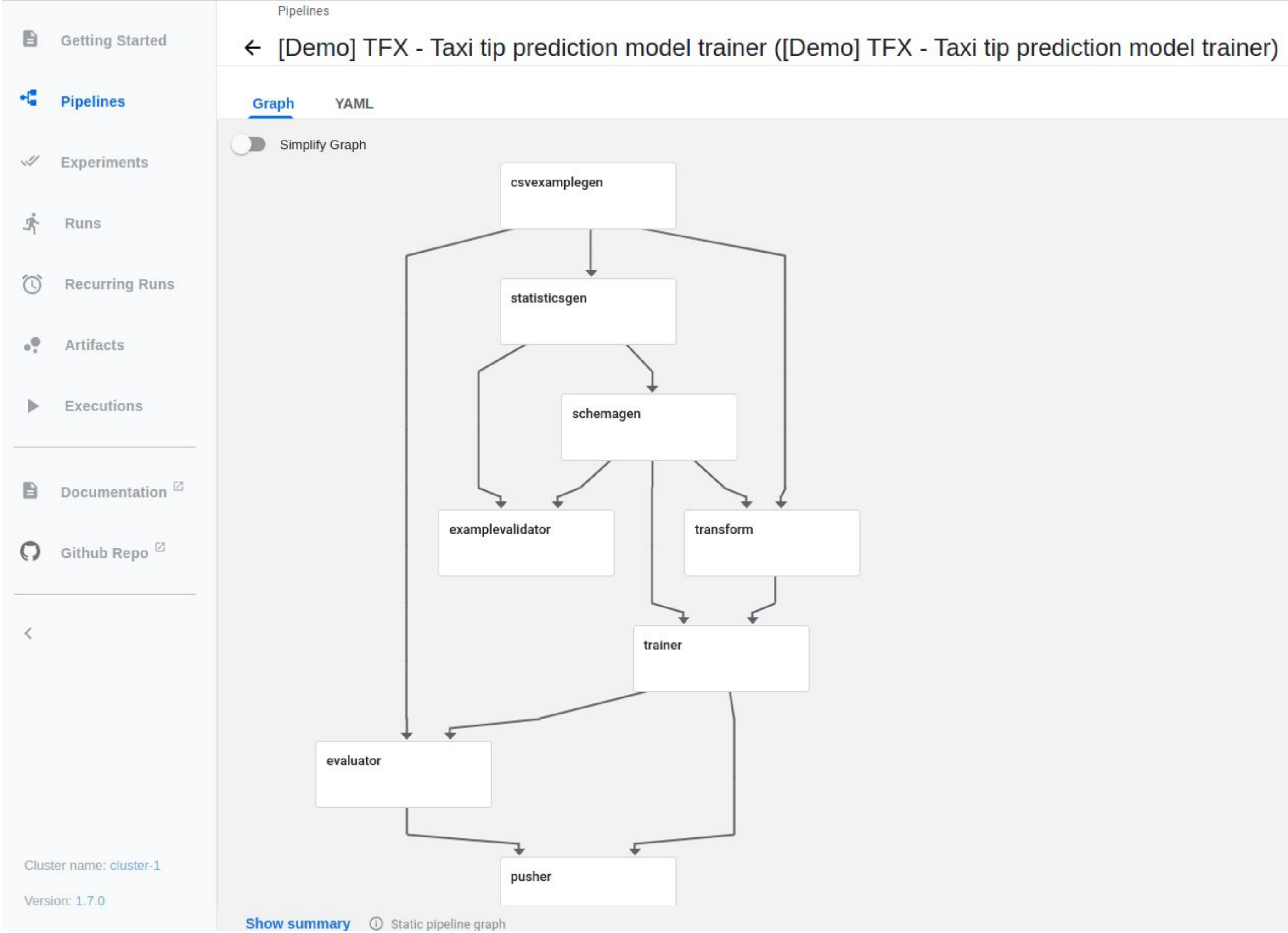


Figure source:
https://www.kubeflow.org/docs/started/architecture/

Captured from Kubeflow



Example: serverless as functions within ML workflows

- **Tasks in ML can be implemented as a function**
- **A workflow of functions can be used to implement ML pipelines**
  - using serverless to implement data preprocessing/training
  - serverless functions for inferences

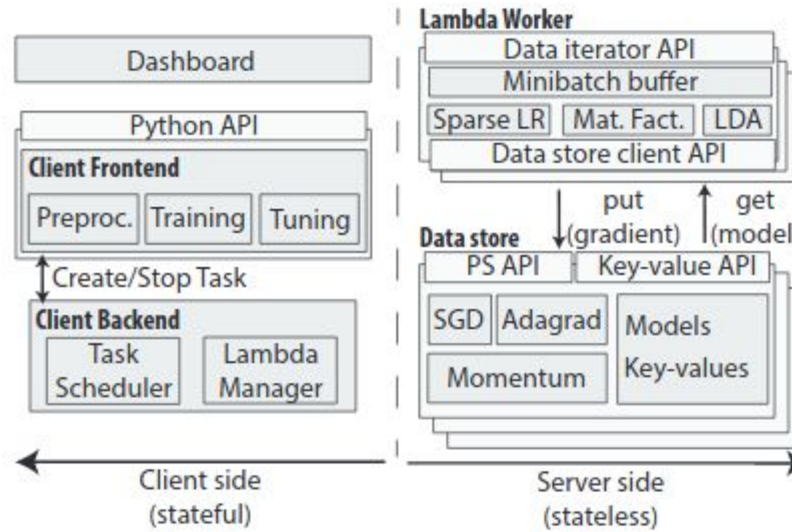
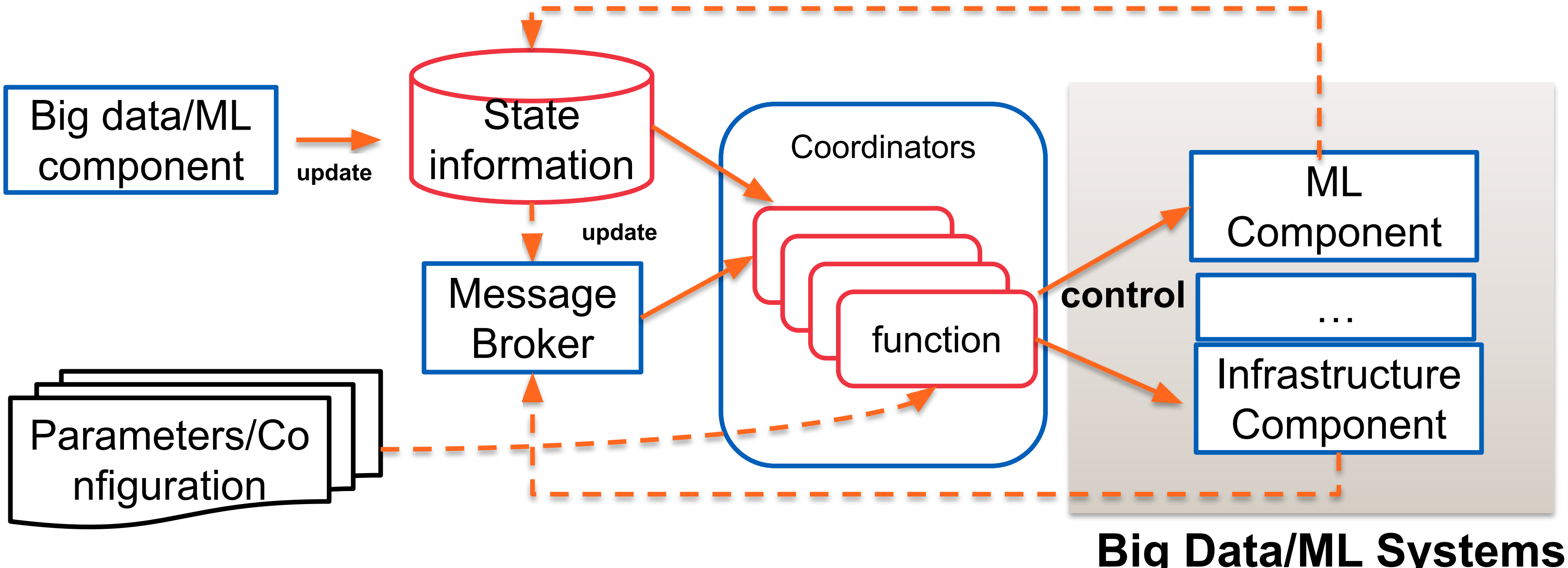


Figure source: Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: a Serverless Framework for End-to-end ML Workflows. In Proceedings of the ACM Symposium on Cloud Computing (SoCC '19). DOI:https://doi.org/10.1145/3357223.3362711

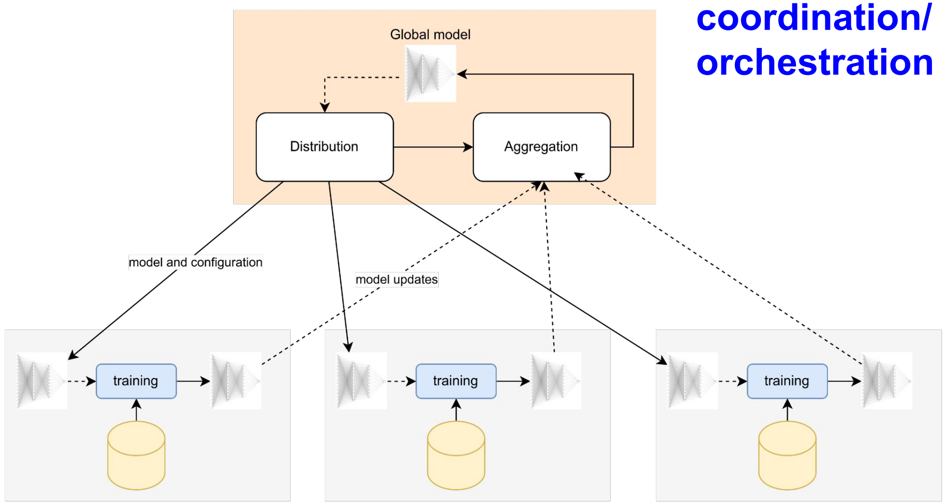
Common architecture with serverless coordination



Coordination in federated learning

"... multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider ..."

Central service/ service provider: coordination/ orchestration



entities (clients): participants

Participants: (i) cross-silo use cases (few) vs cross-device use cases (huge), (ii) heterogeneity in terms of data, computing capabilities, networks, reliability, management, etc.

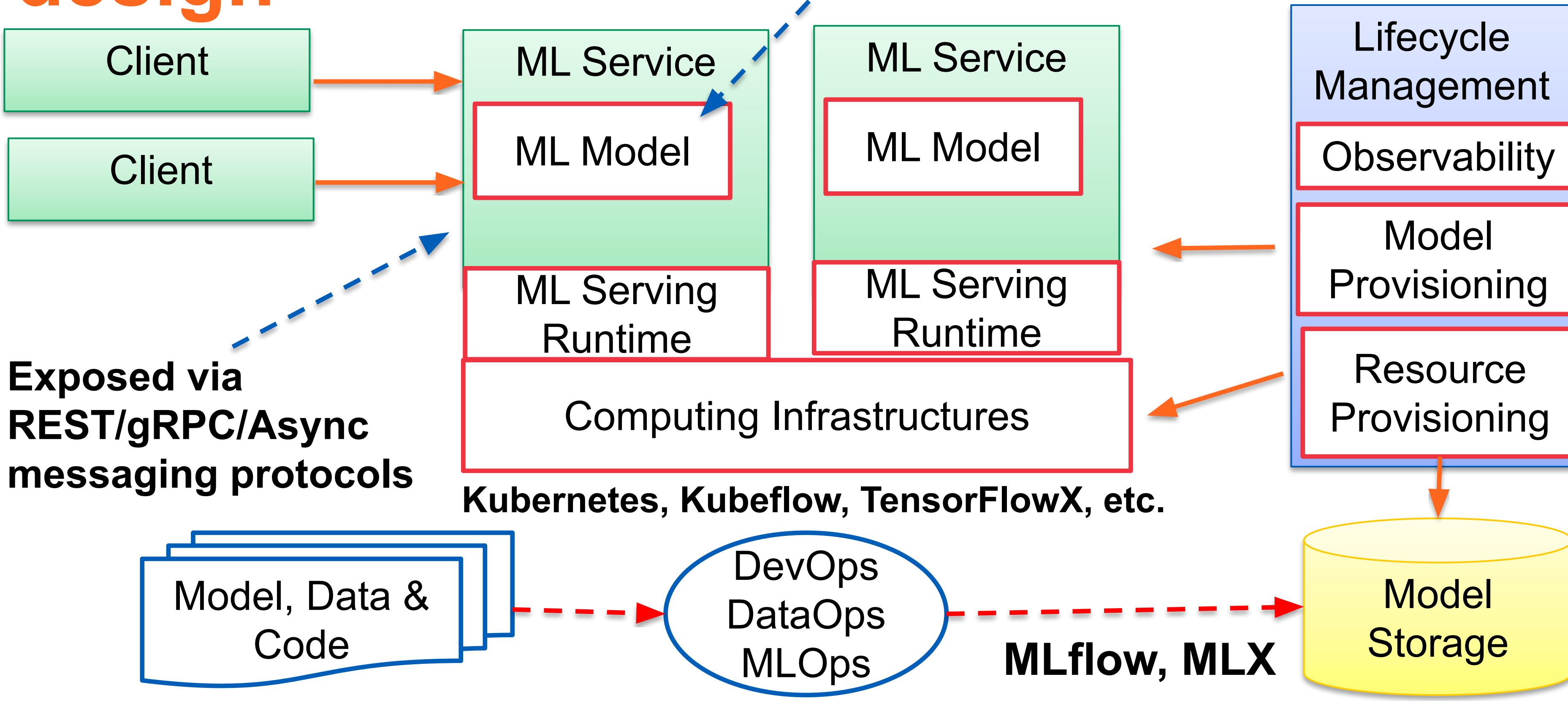
Coordination in Dynamic ML Serving

ML model serving

- **Allow different versions of ML models to be provisioned**
  - runtime deployment/provisioning of models
  - "model as code" or "model as a service" ⇒ can be deployed into a hosting environment
- **Why? Anything related to R3E?**
  - concurrent deployments with different SLAs
  - A/B testing and continuous delivery for ML (https://martinfowler.com/articles/cd4ml.html)
- **Existing platforms**
  - increasingly support by different vendors as a concept of "AI as a service" (check https://github.com/EthicalML/awesome-production-machine-learning#model-deployment-and-orchestration-frameworks)

ML Model Serving design

Different integration techniques (containerization, external process execution, in-process execution)



ML Service

- **Long runtime inferencing services**
  - with well defined interfaces for invoking ML models
  - accept continuous requests and serve in near-real time
- **Containerized services with REST/gRPC & messaging protocols**
  - for on-demand serving or for scaling long running serving
- **Serverless function wrapping ML models for short serving time**
- **Batch serving services**
  - not near real time serving due to the long inferencing time
- **Embedded ML models into application processes**

Question: which are the best forms for which situations? What about the underlying distributed computing for ML services?

Key technical features (1)

- **Service endpoint exposing**
  - ML model -> serving inference unit -> composition of units (dependency graph or horizontal/replica models) -> APIs
- **Serving handles:**
  - different function of routing, composition, load balancing
  - common techniques: HTTP/gPRC proxy, elasticity controller, replica management, and underlying infrastructure orchestrator
- **Serving (dynamically) loads (updated) models**
- **Coupled with deployment configuration**
  - given deployment tools can decide how to deploy serving units

Key technical features (2)

- **Serving platforms/toolkits:**
  - Ray, BentoML, Seldon, KServe, etc.
  - Also Nvidia Triton, AMD Inference, etc., serving runtime
- **Modes, e.g.**
  - Batch serving, autoscaling, asynchronous serving
- **Varying parameters, e.g,**
  - batch serving (batch size, timeout, latency/response)
  - resources and autoscaling (replicas, CPU/GPU, memory)
  - queuing (concurrent requests)

⇒ many ways for optimizing R3E in serving!

Example of exposing ML models

BentoML

```python
import bentoml
import numpy as np
@bentoml.service(name="eei_kmeans", resources={"cpu": "1"})
class EEIKMeans:
    eei_kmeans = bentoml.models.get("eei_kmeans:wtyoald5ycovoziy")
    def __init__(self):
        import joblib
        self.model = joblib.load(self.eei_kmeans.path_of("model.pkl"))
    @bentoml.api
    def classify(self, input_series: np.ndarray) -> np.ndarray:
        return self.model.predict(input_series)
```

Source: https://docs.bentoml.org/en/latest/concepts/service.html

Ray serving

```python
from starlette.requests import Request
from ray import serve
from transformers import pipeline

@serve.deployment(num_replicas=2, ray_actor_options={"num_cpus": 0.2, "num_gpus": 0})
class Translator:
    def __init__(self):
        self.model = pipeline("translation_en_to_fr", model="t5-small")

    def translate(self, text: str) -> str:
        model_output = self.model(text)
        translation = model_output[0]["translation_text"]
        return translation

    async def __call__(self, http_request: Request) -> str:
        english_text: str = await http_request.json()
        return self.translate(english_text)

translator_app = Translator.bind()
```

Source: https://docs.ray.io/en/latest/serve/getting_started.html

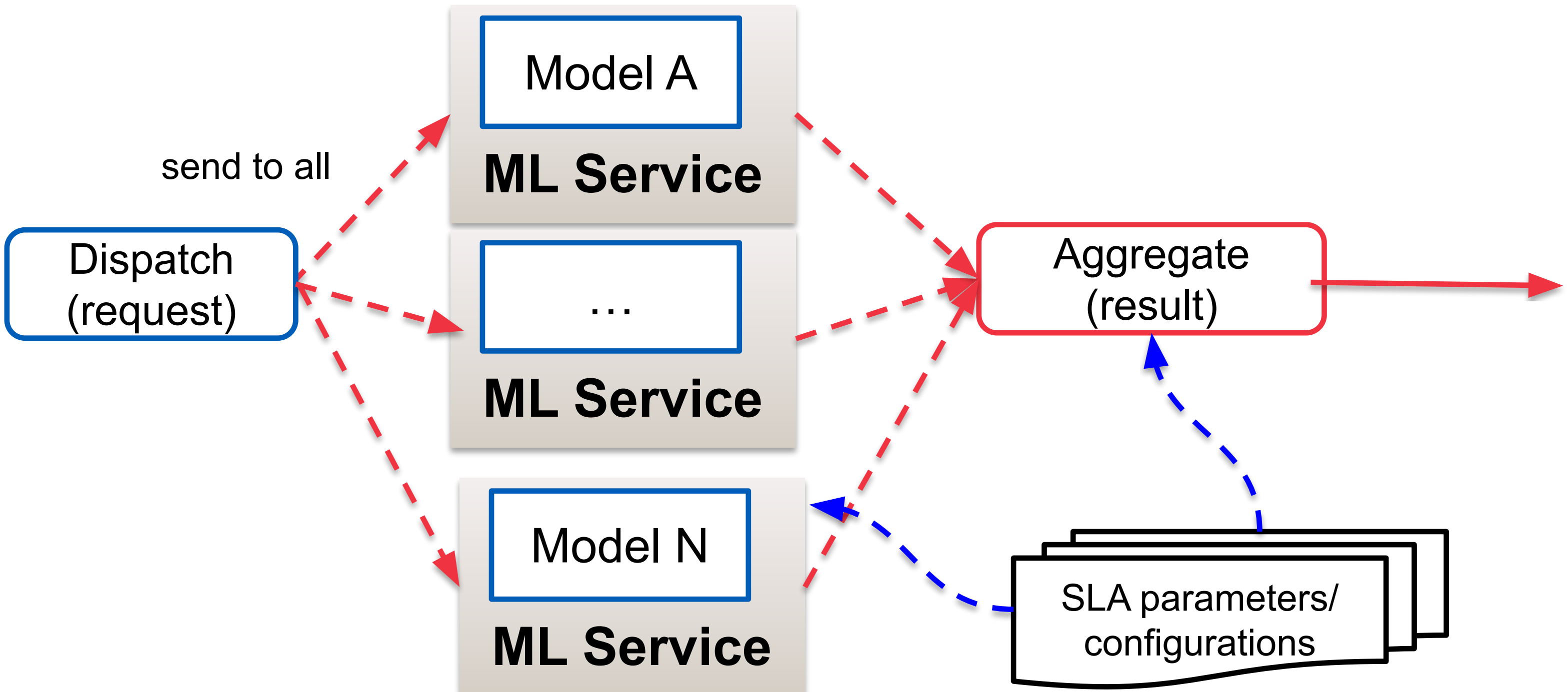
Tensorflow Serving

```
tensorflow_model_server --port=8500 --rest_api_port=8501 \
  --model_name=${MODEL_NAME} --model_base_path=${MODEL_BASE_PATH}/${MODEL_NAME}
```

Source: https://github.com/tensorflow/serving/blob/master/tensorflow_serving/g3doc/docker.md

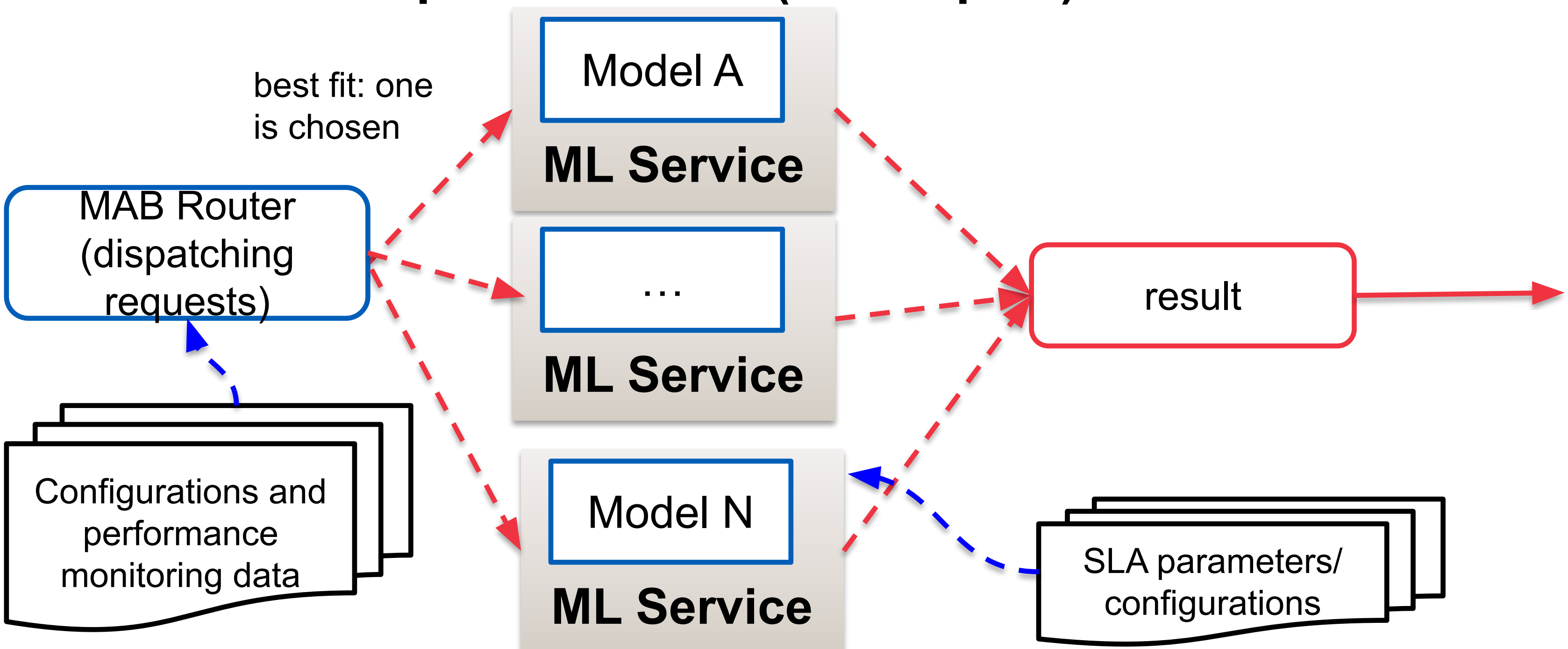
Composition - Ensemble

Ensembles of different models with different qualities/SLAs (R3E topics)



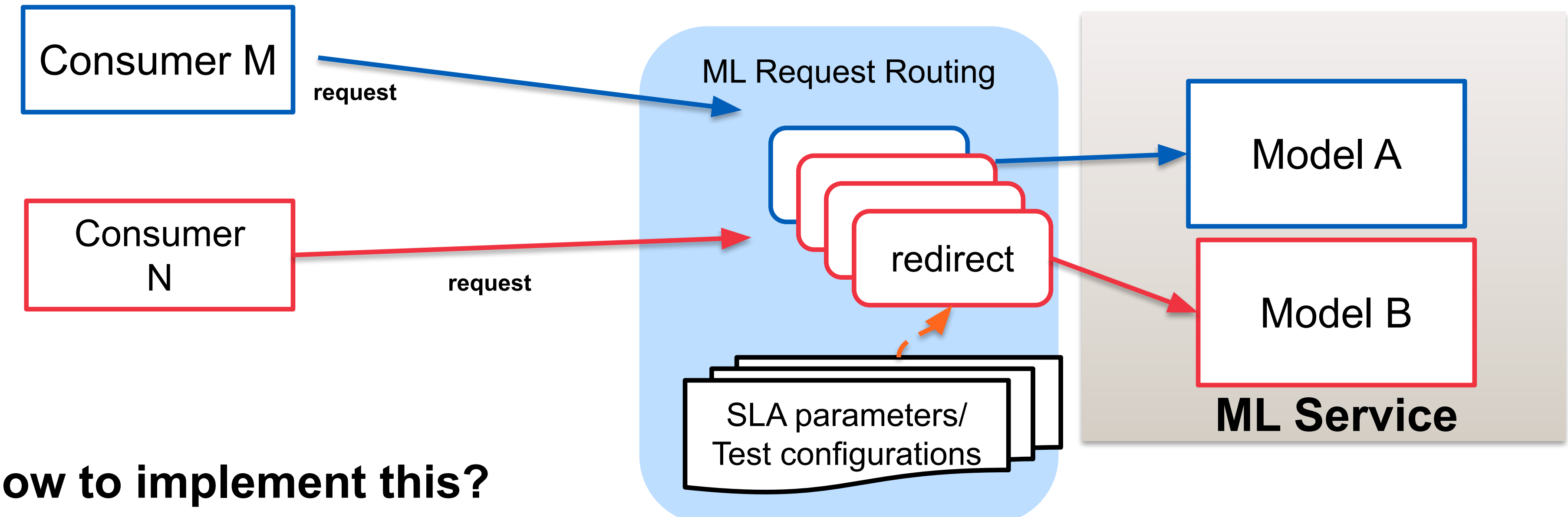
Composition - Multi-armed bandits

Decision on the most suitable service based on different models with different qualities/SLAs (R3E topics)



A/B testing and SLA-based serving

Different models with different qualities/SLAs (R3E topics)



How to implement this?

Amazon Sagemaker Example:
https://docs.aws.amazon.com/sagemaker/latest/dg/model-ab-testing.html

Load balancing/scaling model serving

- **ML inferencing capability in a ML model is encapsulated into a microservice or a task**
- **As a service**
  - with well-defined APIs (e.g., REST, gRPC), e.g., Dockerized service
  - using load balancing and orchestration techniques, such as Kubernetes
- **As a task**
  - using workflow management techniques to trigger new tasks
  - support scheduling, failure management and performance optimization by leveraging batch processing techniques

KServe

- Inference platform atop Kubernetes
- Support most common ML frameworks
- Various scaling and deployment supports

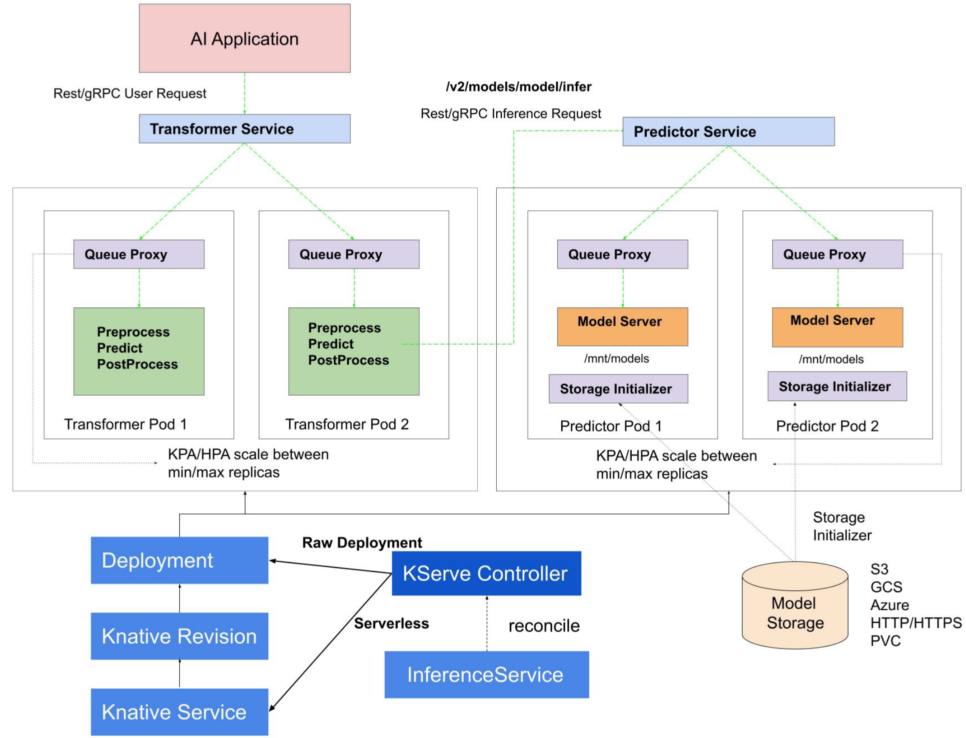


Figure source: *https://kserve.github.io/website/latest/modelserving/mms/modelmesh/overview/*

Ray

- **ML Serving and other related data distributed computing components**
  - data processing and training
  - hyperparameter tuning
- **Computing resources**
  - single node, clusters, or Kubernetes-enabled systems
- **Rich ecosystems**
  - integrated with and used by many others

Example: Ray Serving

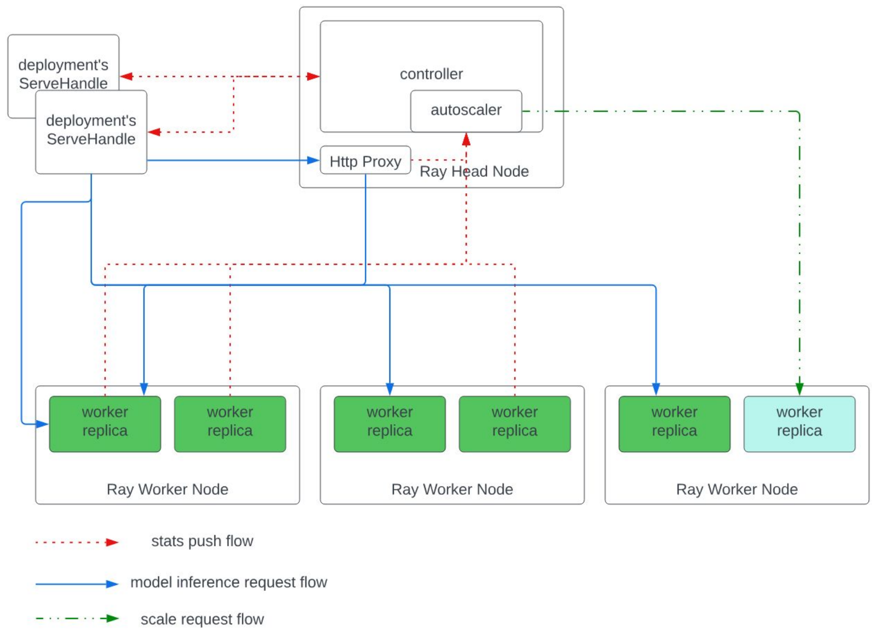


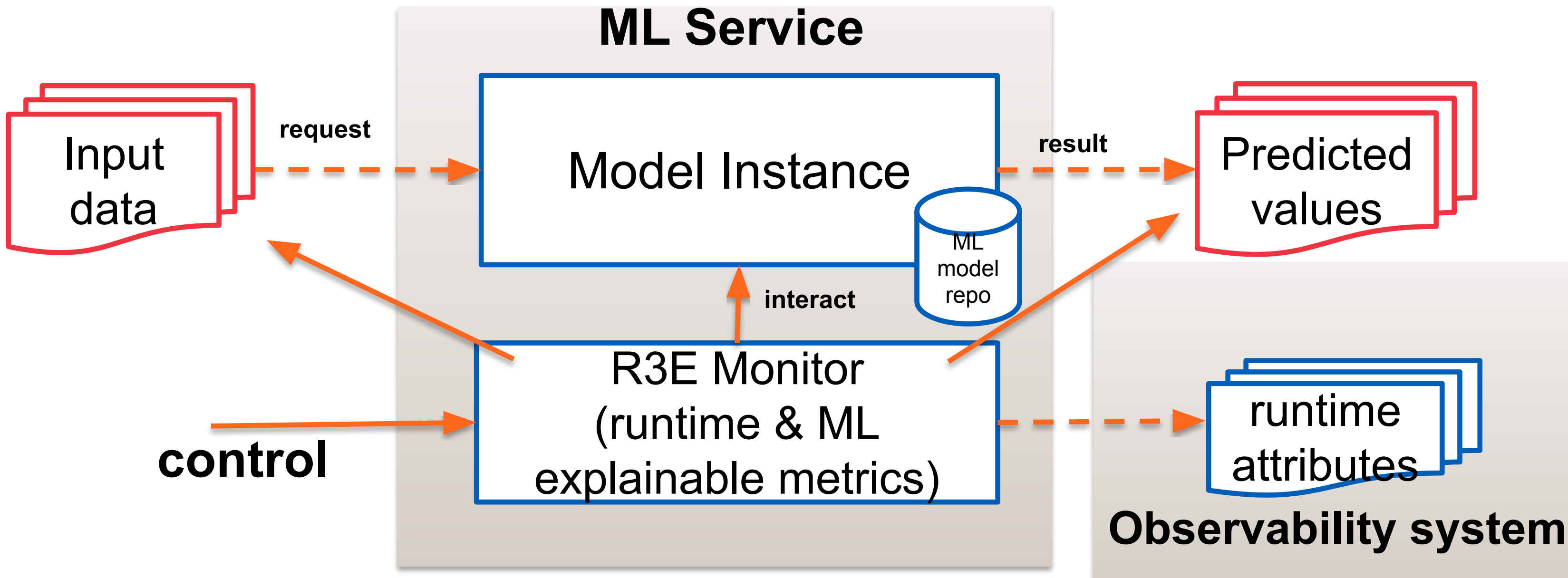
Figure source: https://docs.ray.io/en/latest/serve/architecture.html

Where are the difference in ML serving?

- **We can see common techniques like autoscaling, handles for encapsulating details & separation, proxy, multitenancy, etc.**
  - various design patterns from service and distributed computing, e.g.:
    - *https://learn.microsoft.com/en-us/azure/architecture/patterns/*
  - autoscaling, e.g.:
    - *https://docs.ray.io/en/latest/serve/autoscaling-guide.html*
- **But where would be the key different problems in ML serving?**

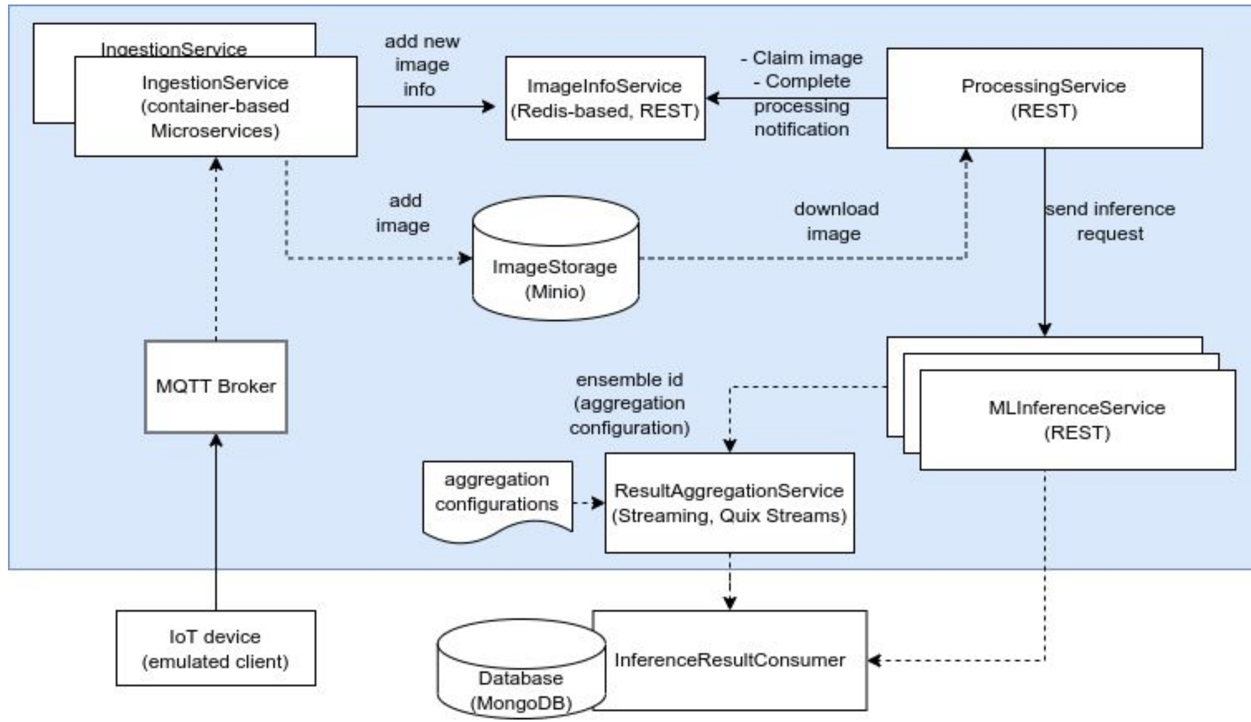
R3E runtime attributes?

How to capture important metrics for observability and dynamic serving?



Examples: object detection/classification pipeline

- Discussion: dealing R3E with ML workflows?
- Where, What, When and How



Study log

P1 - Take one of the following aspects:

- P1.1 - Robustness, Reliability, Resilience or Elasticity
- P1.2 – Automation management

P2 - Check one of the following aspects:

- Orchestration of ML pipelines or ML serving

In a *specific software framework* (F3) that you find interesting/relevant to your work:

discuss how do you see F3 supports P1 in doing P2

(the reading list also helps)

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io