Serverless workflows for orchestration hybrid cluster-based and serverless processing

Rustem Feyzkhanov

Instrumental

Strata

DATA CONFERENCE

Presentation takeaways

- Cloud native orchestrators are convenient for uniting serverless and cluster worlds

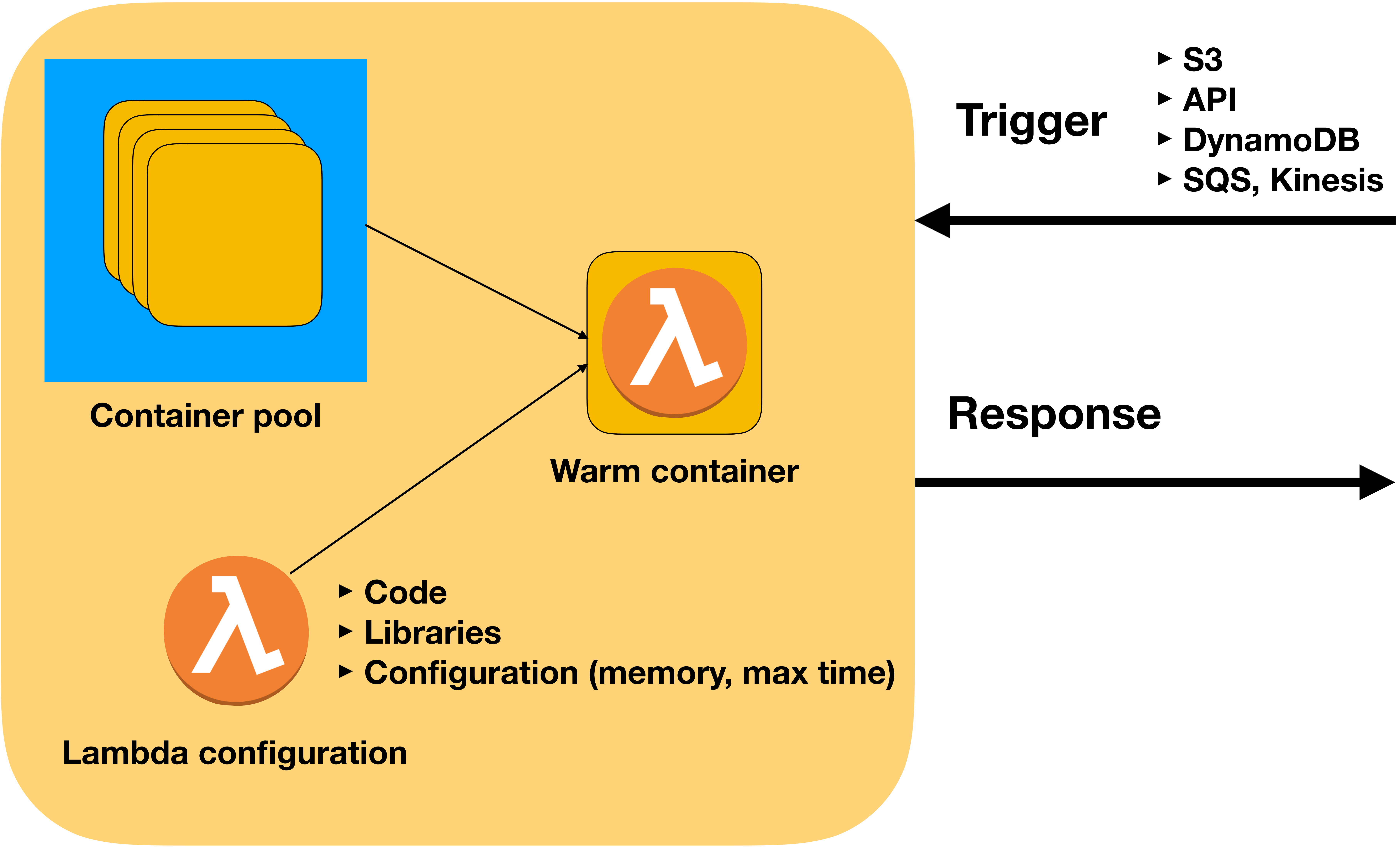
- They provide a way to shift from static cluster to on-demand cluster aka ‘serverless’ cluster

- They can be used for a number of applications and can be deployed easily together

Function as a service (FaaS)

On premise	IaaS	PaaS	FaaS	SaaS
Functions	Functions	Functions	Functions	Functions
Application	Application	Application	Application	Application
Runtime	Runtime	Runtime	Runtime	Runtime
Operating system	Operating system	Operating system	Operating system	Operating system
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Networking	Networking	Networking	Networking	Networking
Storage	Storage	Storage	Storage	Storage
Hardware	Hardware	Hardware	Hardware	Hardware

How typical FaaS works



Trigger

- S3
- API
- DynamoDB
- SQS, Kinesis

Container pool

Warm container

Response

- Code
- Libraries
- Configuration (memory, max time)

Lambda configuration

FaaS pros/cons/limits

Pros	Cons	Limits
Easy to deploy (no docker) Easy to connect to cloud native services Easy to scale Relatively cheap	No local debug Unpredictable warm containers Logging may not be exhaustive	Max execution time Max available RAM Hard disk

Microservice architecture

A set of services which are:

- Loosely coupled
- Independently deployable
- Each one of them implements a business capability
- Highly maintainable and testable

Microservice connectors

Rest API	Event query	Orchestrator
Synchronous process	Asynchronous process	Asynchronous process
Short-term process	Long-term process	Long-term process
Simple intermediate logic	Simple intermediate logic	Complex intermediate logic
Doesn’t trace the whole process	Doesn’t trace the whole process	Traces the process
Cheap	Cheap	Expensive

Cloud native orchestrators

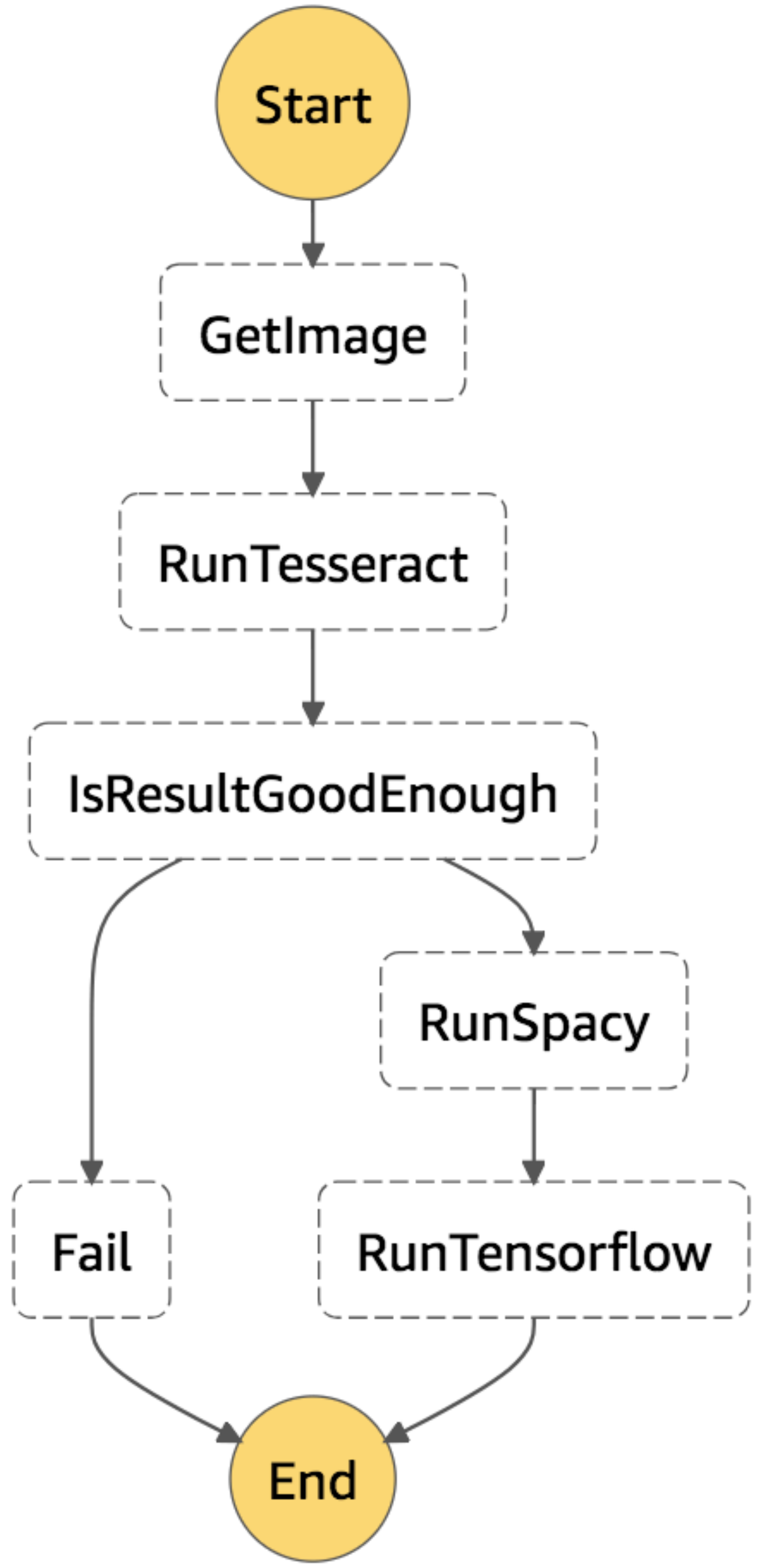
- Native support for FaaS
- Central monitoring
- Central logging and tracing
- On-demand scaling

Types of cloud native orchestrators

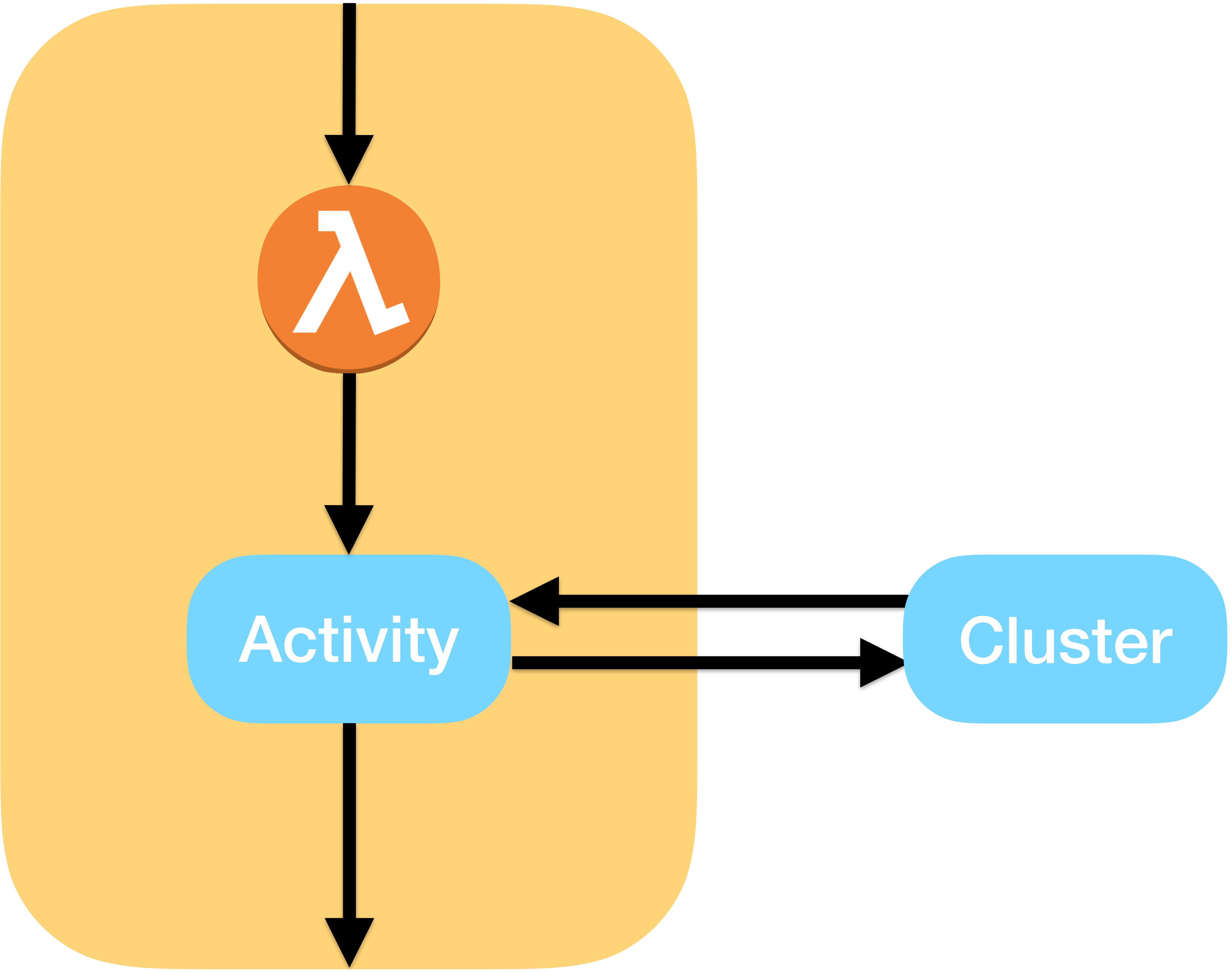
- AWS Step Functions
  - Integration with other AWS services
  - Static parallelization
- Azure logic apps
  - Integration with other services' APIs
  - Dynamic parallelization

Orchestrators for hybrid architecture

- Graph-based description
- Processing nodes: FaaS or Clusters
  - Task state and waiting for the node
  - Invocation of processing node
- Logic for error handling
- Parallel execution
- Branching and loops
- Scheduler

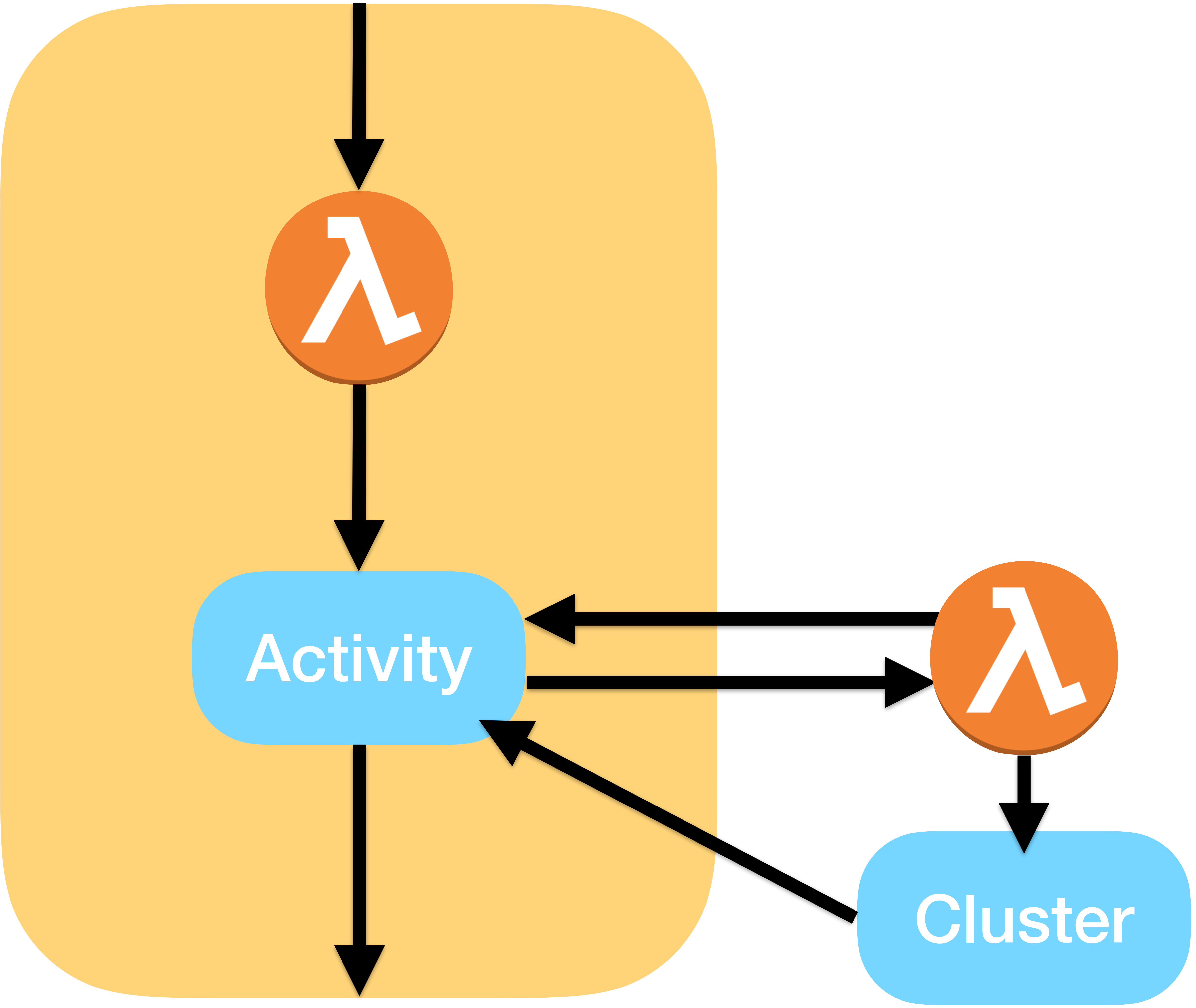


How task state works



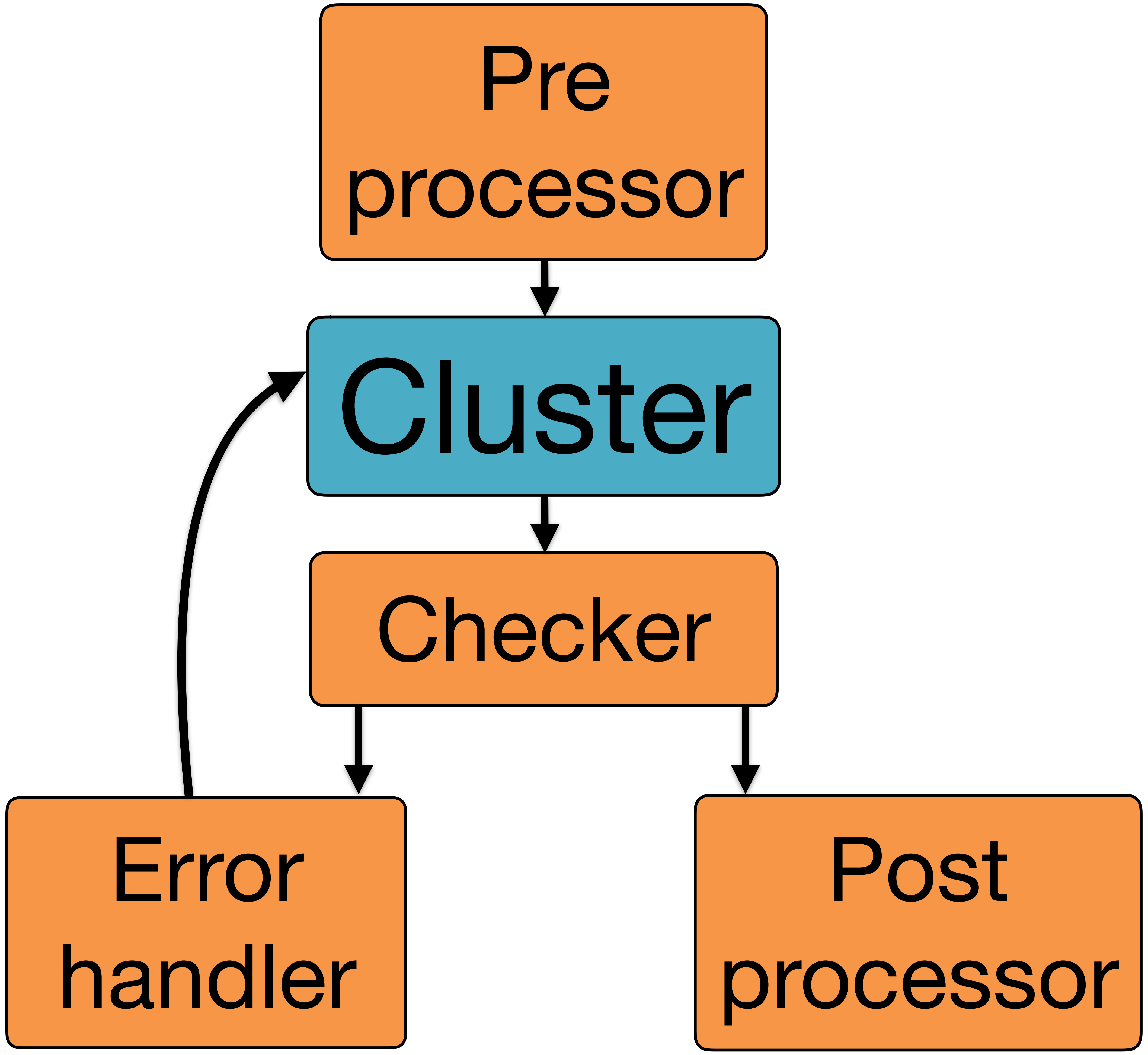
- Cluster makes constant requests to Step Functions service
- Receives input json with token and runs task
- Returns output with the token of specific Step function

How task state also works



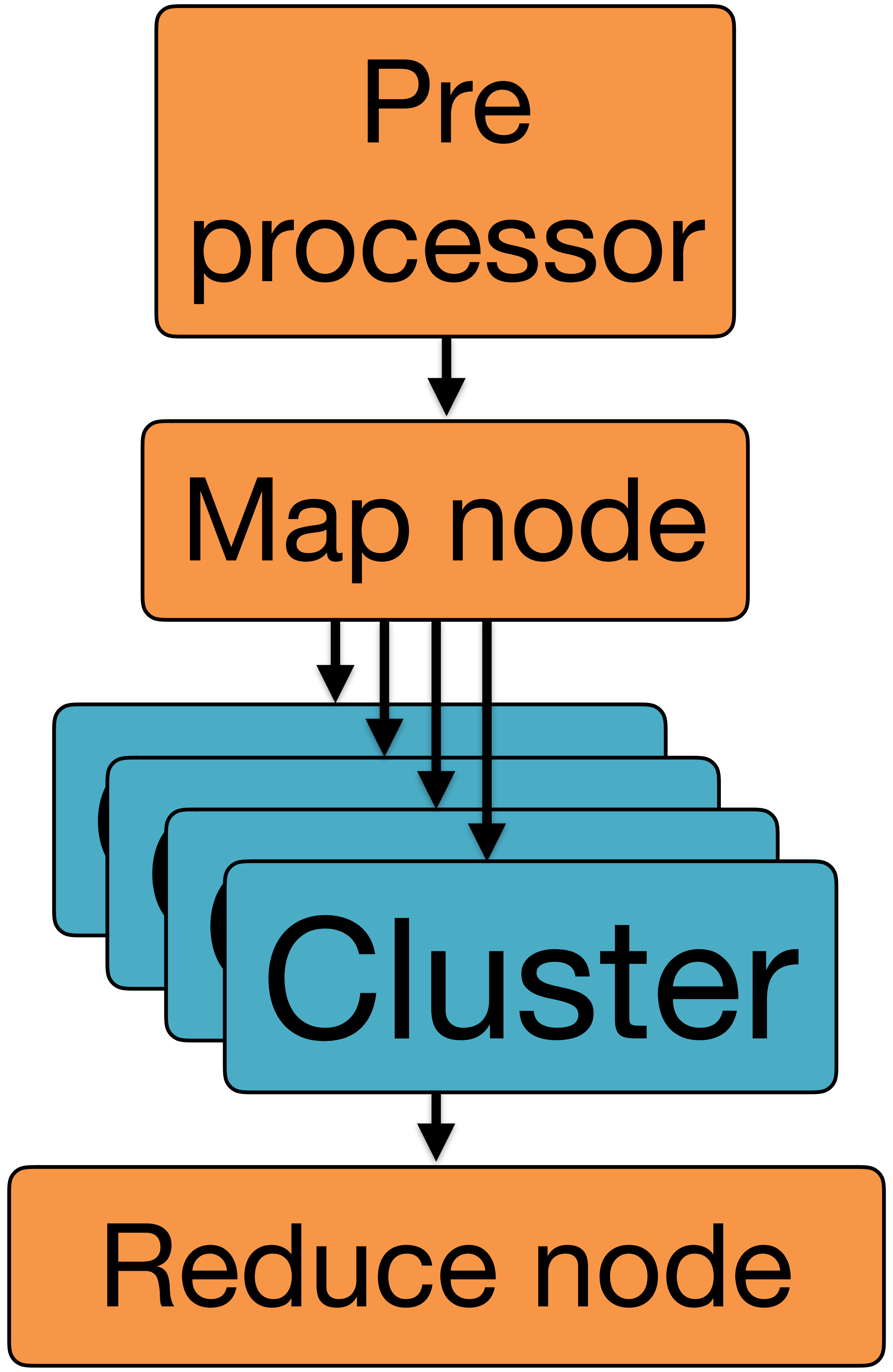
- Lambda makes constant requests to Step Functions service
- Receives input json with token and sends task to cluster
- Cluster returns output with the token of specific Step function

Example patterns



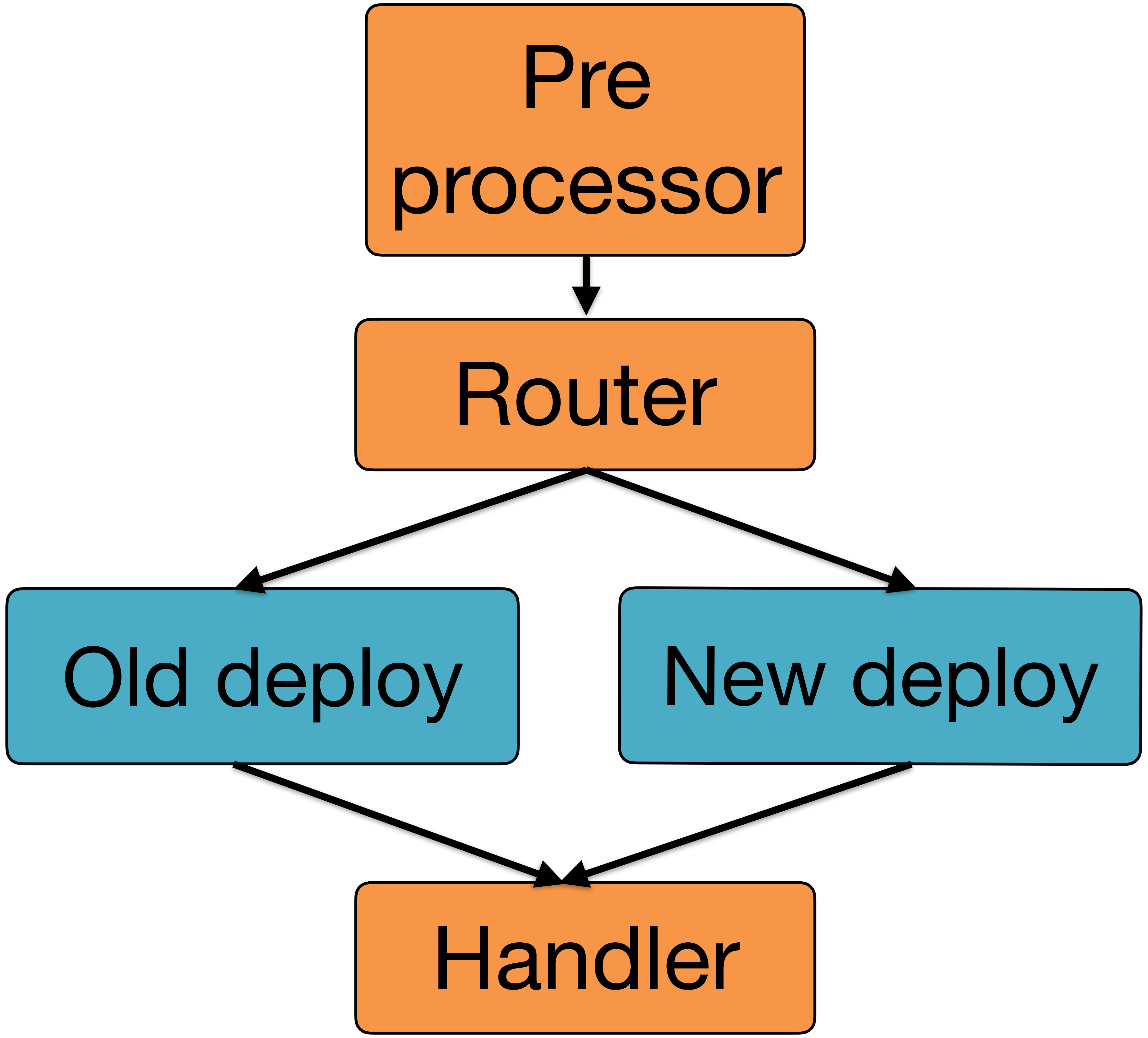
- Wrapper for high tier services
  - Timeout for the task
  - Error handling
  - Retry logic
  - Falling back to alternative service

Example patterns



- Tasks which require parallel execution
  - Map node for the parallel tasks
  - Error handling at each parallel branch
  - Retry logic for each branch

Example patterns



- Canary deployment
  - Redirects traffic to different nodes based on custom logic
  - Gathers stats based on execution
  - Can fall back to another service

‘Serverless’ cluster

- Container-as-a-Service
- On-demand cluster which scales with your consumption
- Services:
  - AWS Batch
  - AWS Fargate

Serverless cluster comparison

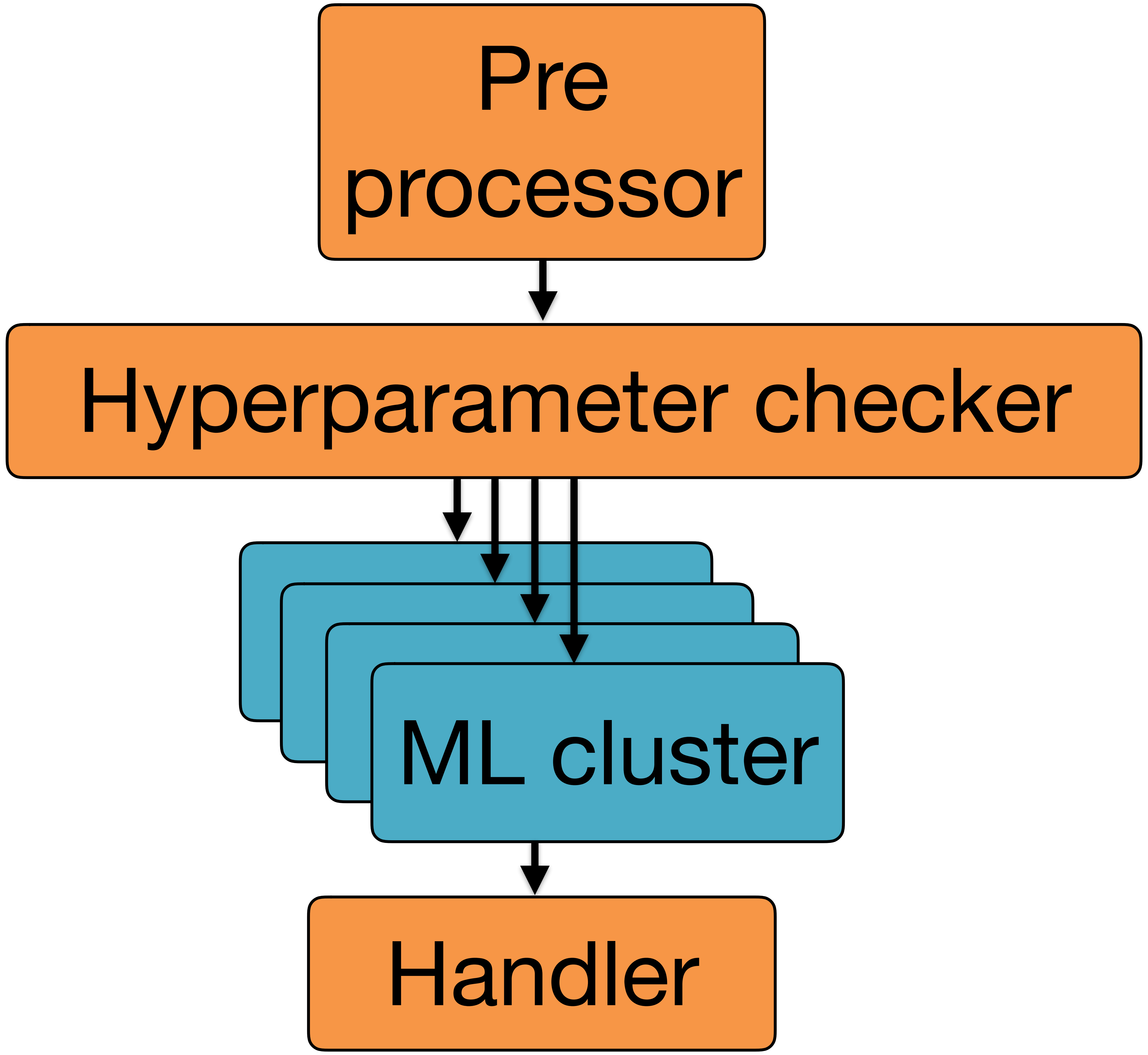
Lambda	Fargate	Batch
	Pure container as a service	Service which starts cluster and executes jobs on it
FaaS	Customizable instances	
Short term processes	Only CPU instances	Spot instances available
Fast startup time (~100ms)	Medium startup time (~10-20s)	Slow startup time (~1-4min)
Price per 100ms	Price per 1s (min 1 min)	Price per 1s (min 1 min)

Price comparison

C5 Large Instance - 2 vCPU 4GB RAM

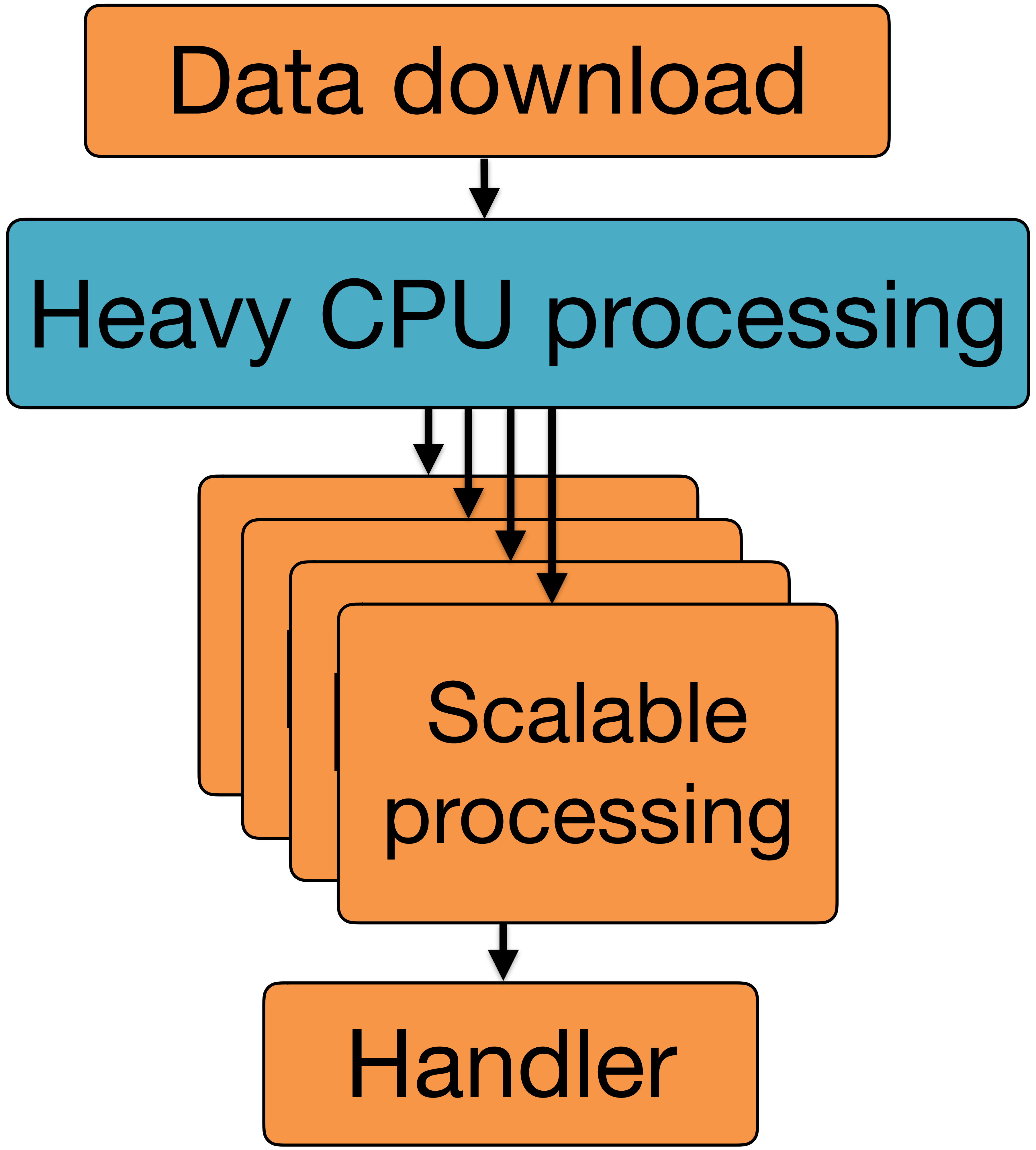
- AWS Lambda
  - 3GB RAM x 0.00001667 x 3600 **= 0.18$ per hour**
- AWS Fargate
  - 4GB RAM x 0.0044 + 2 vCPU x 0.0404 **= 0.098$ per hour**
- AWS Batch
  - C5 Large On Demand **= 0.085$ per hour**
  - C5 Large Spot **= 0.033$ per hour**

Examples - ML training



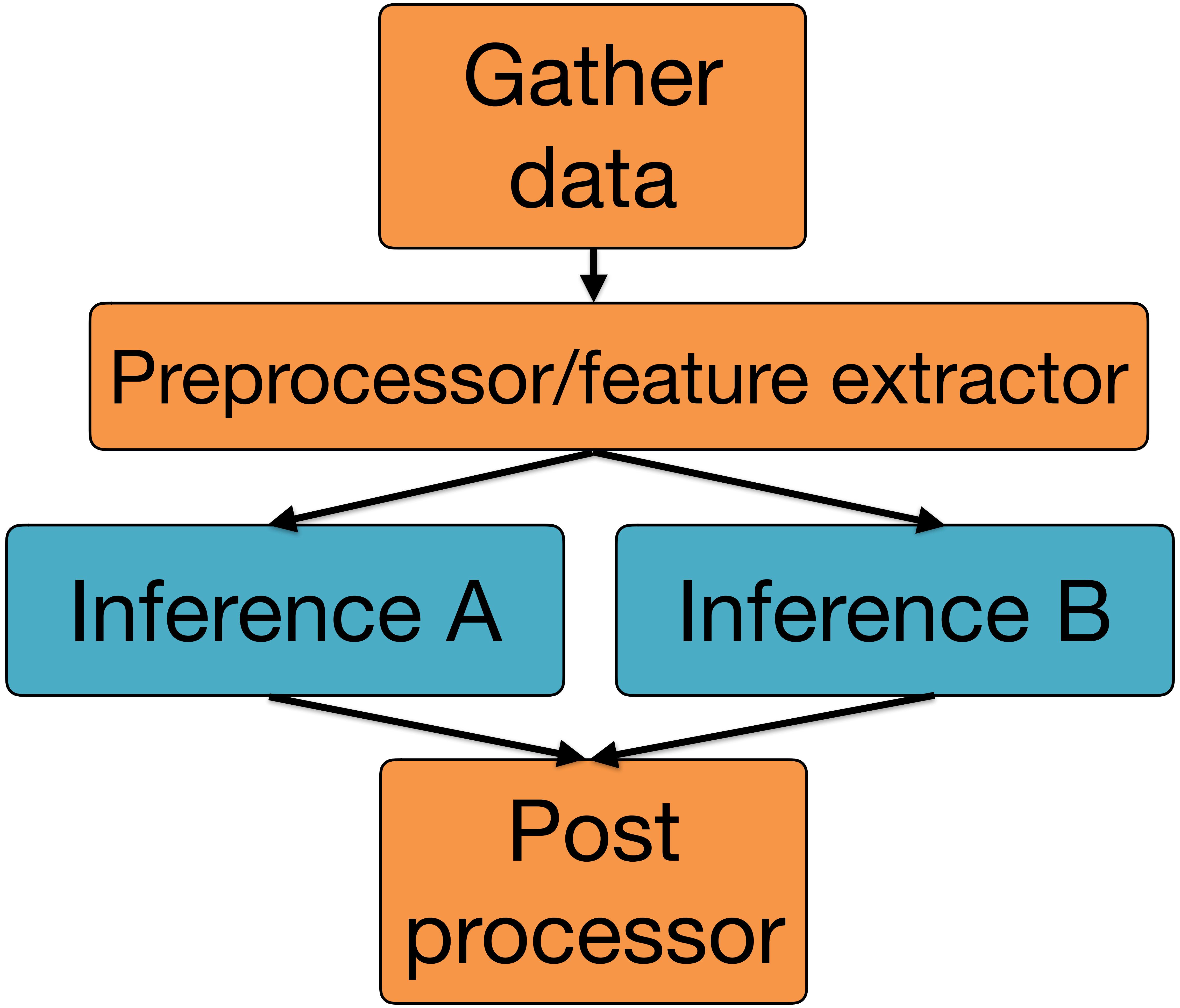
- Parallel training on multiple sets of hyper parameters
- Central gathering of the results
- Handling error on each branch
- Capability for feedback loop

Examples - Data pipeline



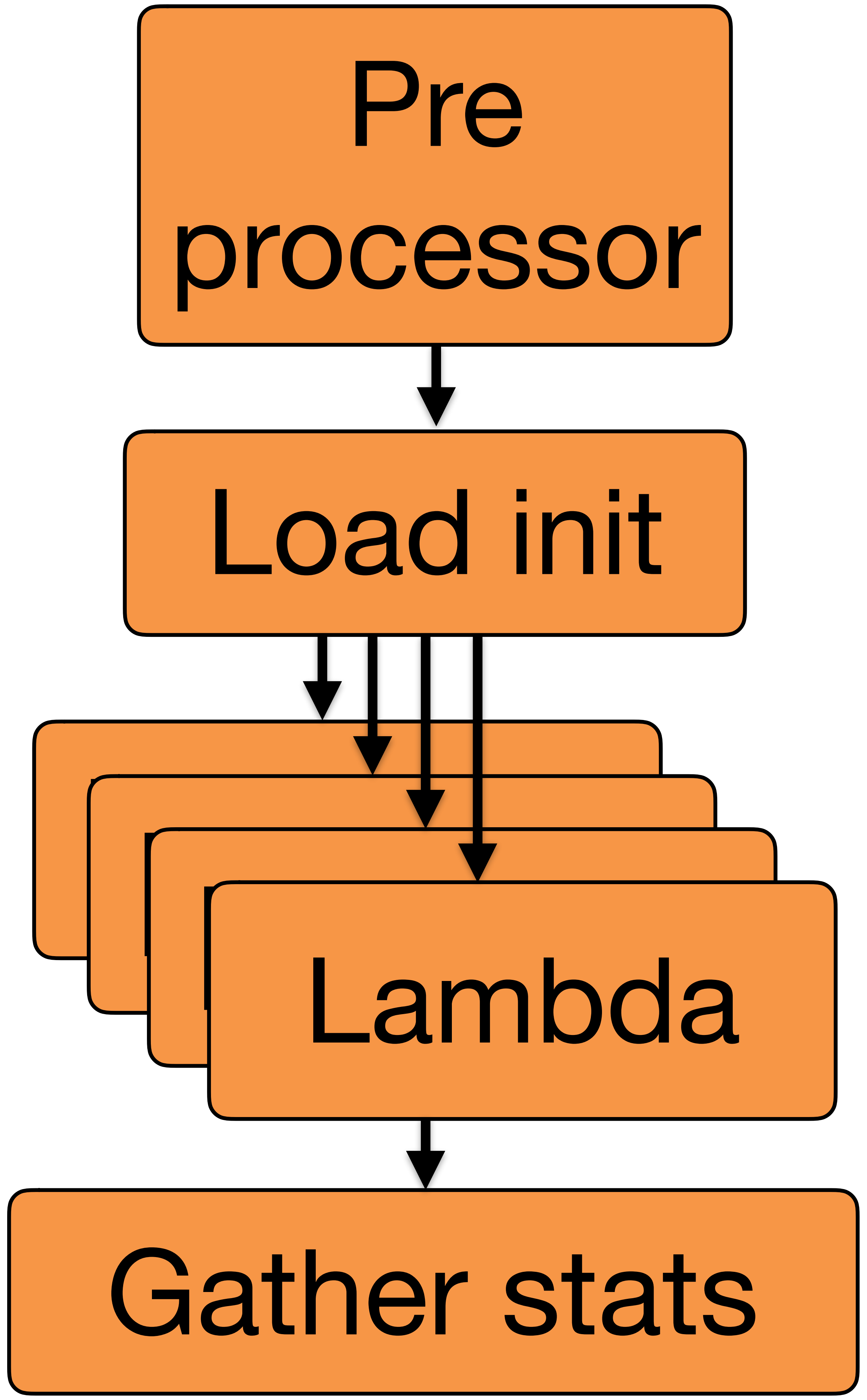
- Modular approach
- Parallel data download and parsing
- FaaS for parallel processing
- Cluster for heavy processing

Examples - ML pipeline



- A/B testing to test performance of multiple models - either in parallel or separately
- Scalable inference which allows to run batches in parallel
- Allows modular approach (multiple frameworks)

Examples - Load testing



- Multiple parallel lambdas => low cost short-term heavy load
- Handles parallel start of multiple AWS Lambdas
- Can be scheduled (in a static or dynamic way)

Github repo + demo

- https://github.com/ryfeus/stepfunctions2processing
- Configuration for serverless framework deploys:
  - AWS Step functions
  - AWS Lambdas
  - AWS Fargate
  - AWS Batch