Lessons Learnt from Launching Millions of Spark Executors

Zhou Jiang, Aaruna Godthi

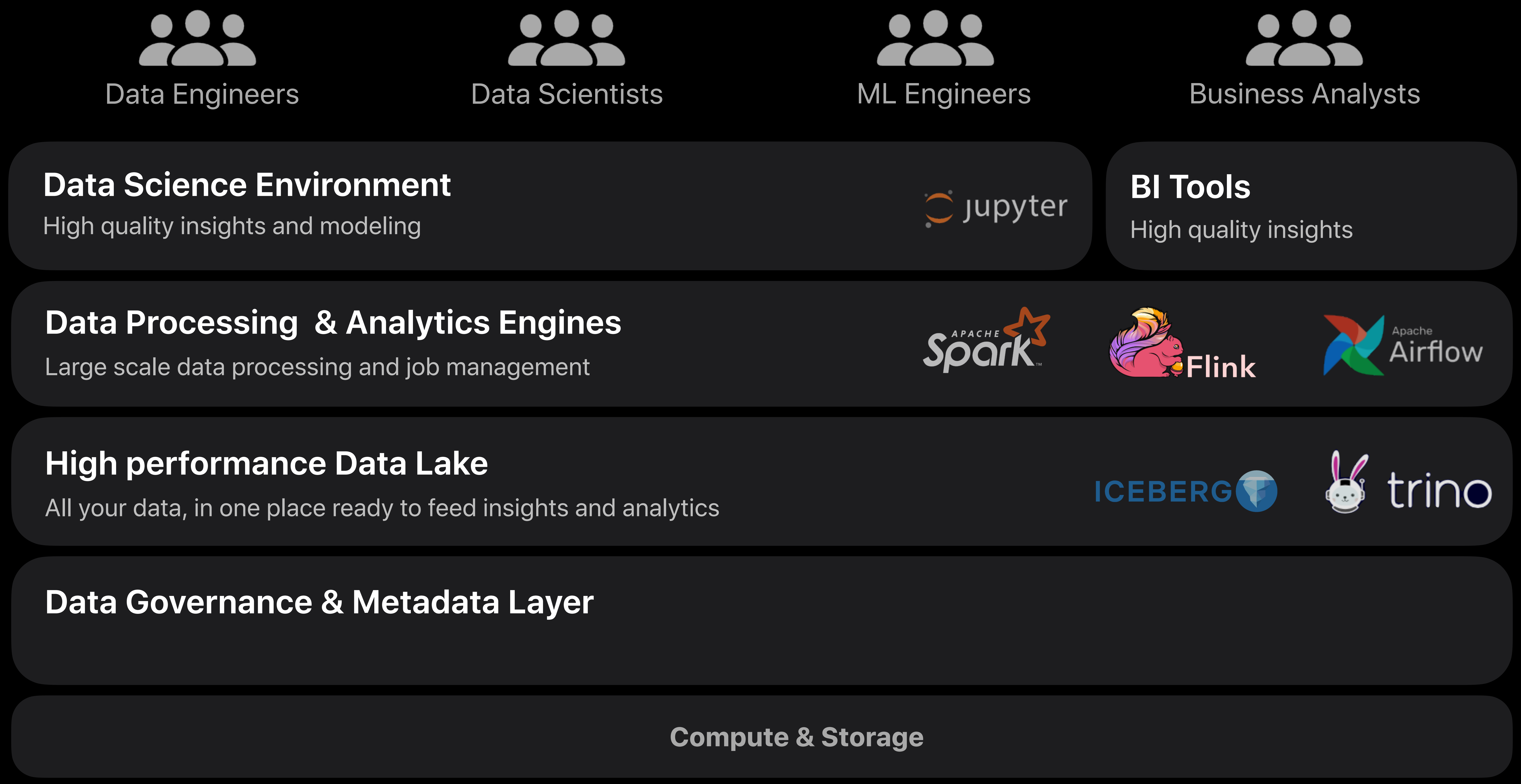
About Us

Zhou Jiang is a software engineer building a high performance data analytics platform for software engineers and data scientists at Apple.

Aaruna Godthi leads the team that provides an on-demand, secure, fully managed and elastic Apache Spark service to various teams at Apple.

Data Platform

Securely accelerate the creation of immersive data experiences



Data Engineers

Data Scientists

ML Engineers

Business Analysts

Data Science Environment

High quality insights and modeling

BI Tools

High quality insights

Data Processing & Analytics Engines

Large scale data processing and job management

High performance Data Lake

All your data, in one place ready to feed insights and analytics

Data Governance & Metadata Layer

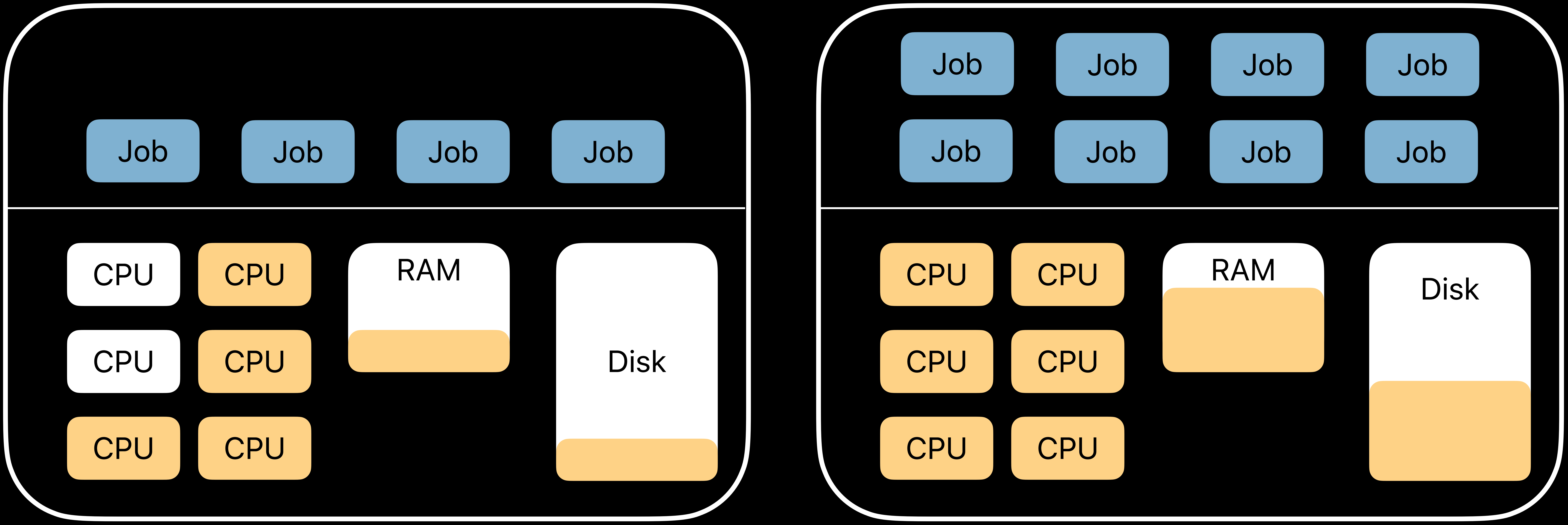
Compute & Storage

Managed Spark at Apple

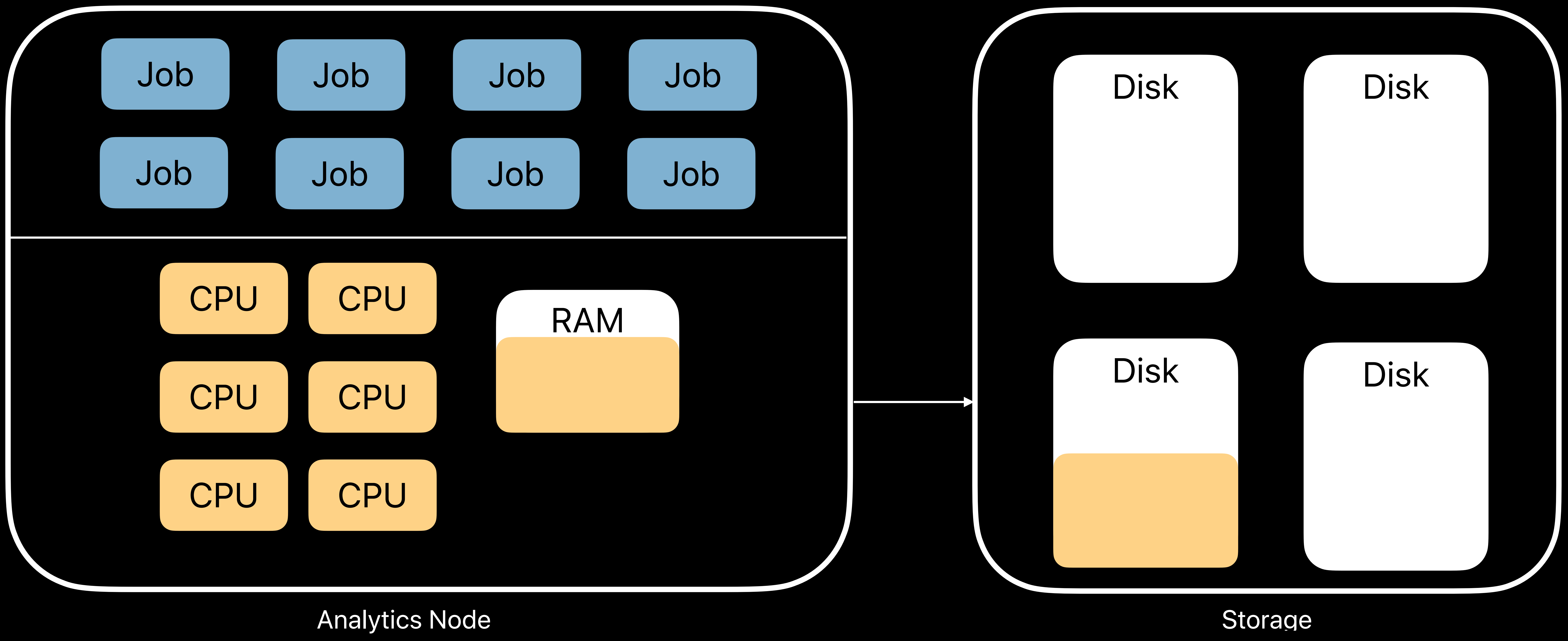
What & Why ?

Elastic Self Service Spark

Why?

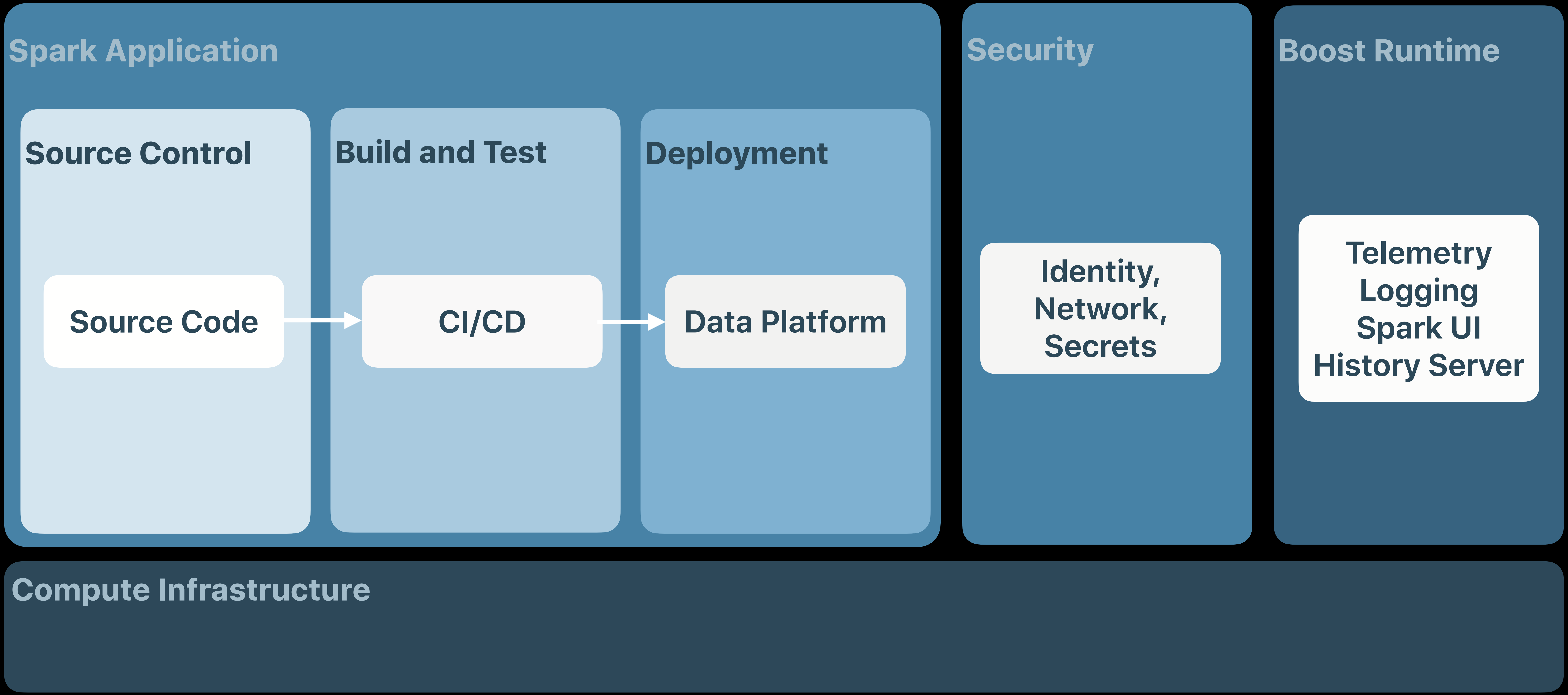


Elastic Self Service Spark



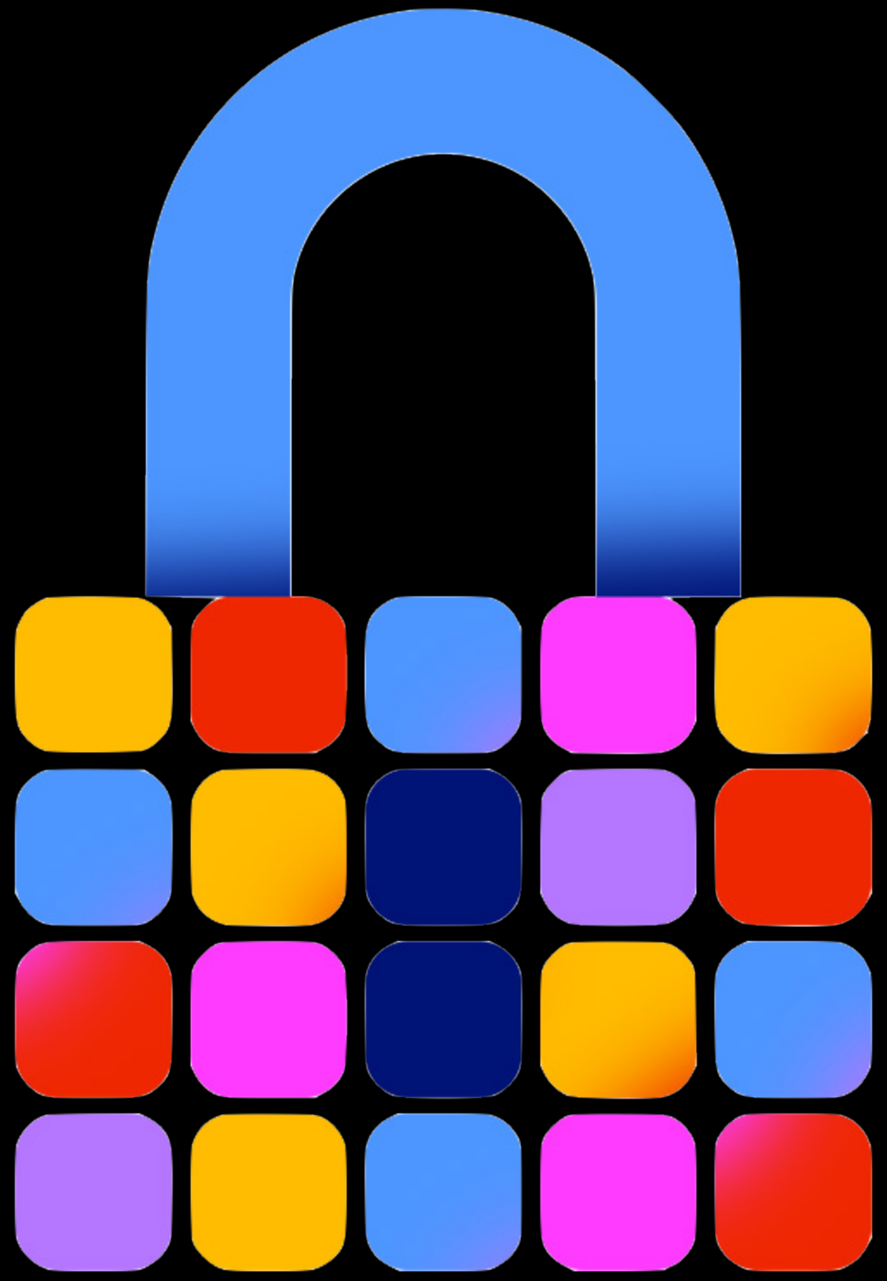
Code to Deployment

Develop, Build, Deploy, Run



Security

- Application certificates
- Network ACLs
- Encryption
- Secrets management

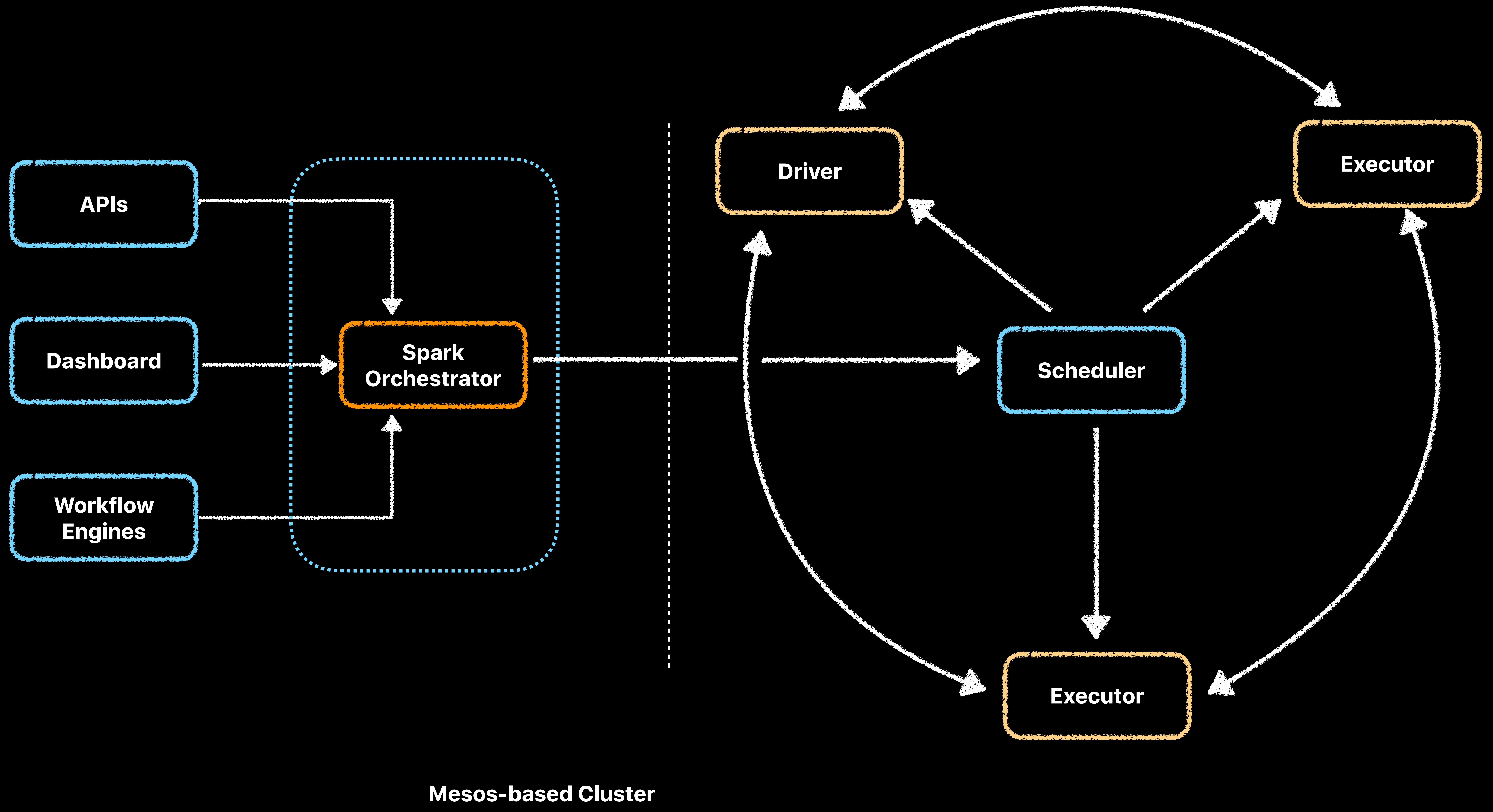


Monitoring

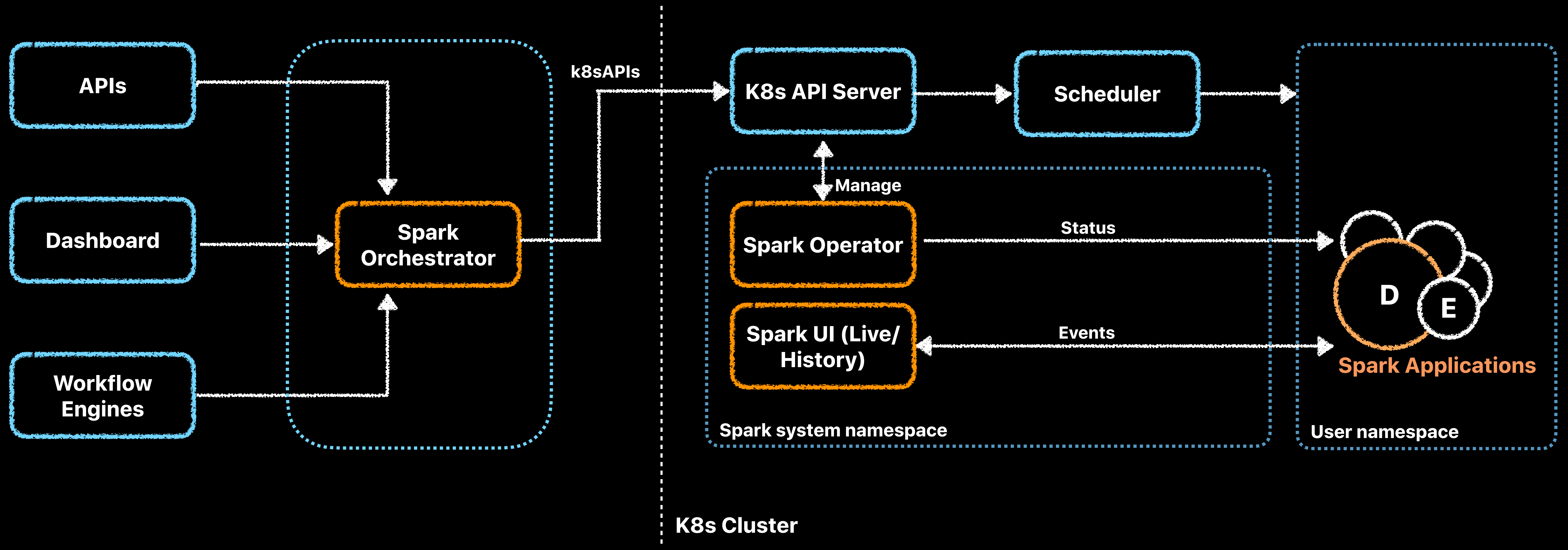
- Logging Integration
- Telemetry System Integration
- User Defined Metrics
- Alert on Key Metrics

How

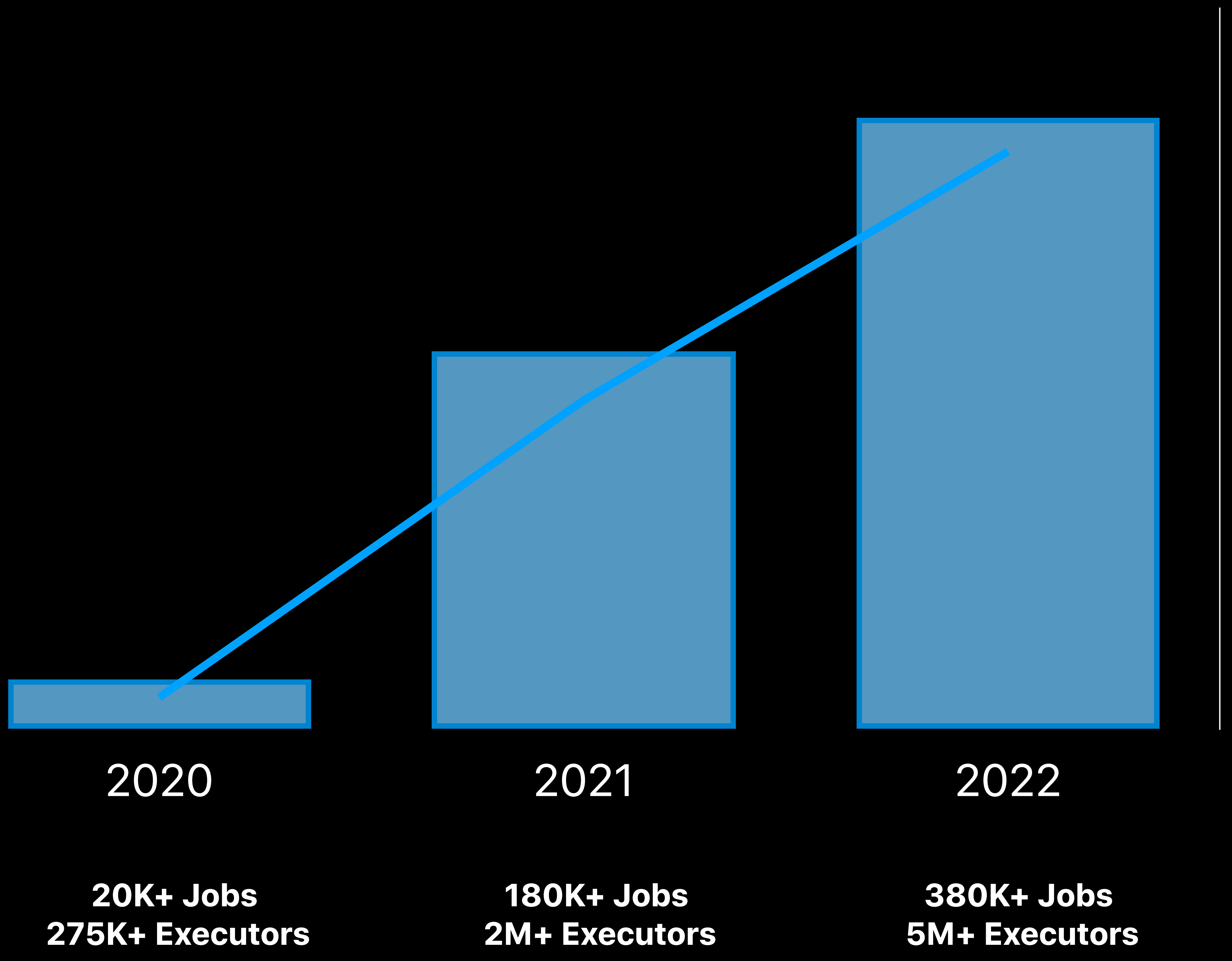
Orchestration Architecture



Orchestration Architecture



Our Scale


2020
2021
2022
20K+ Jobs
275K+ Executors
180K+ Jobs
2M+ Executors
380K+ Jobs
5M+ Executors

Scale up Spark On Kubernetes

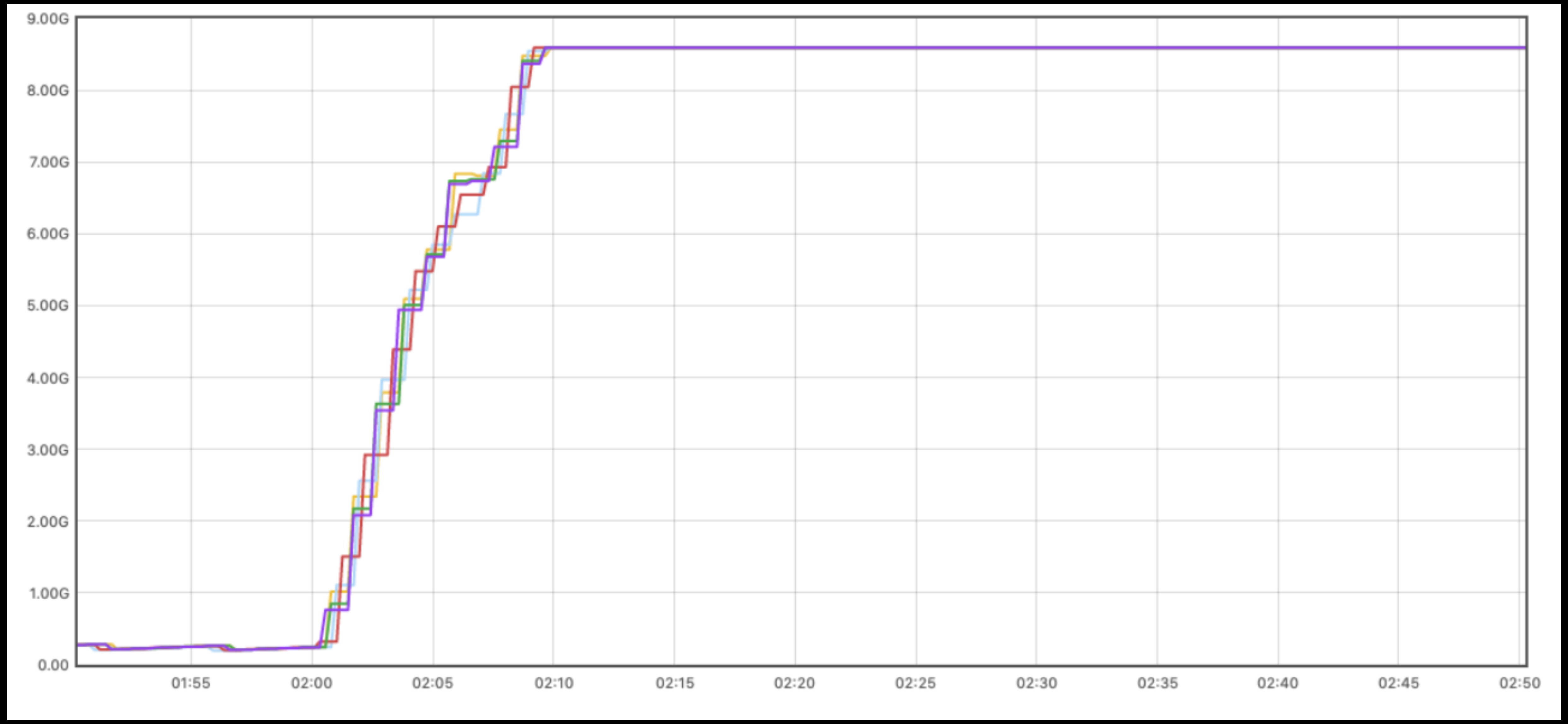
Challenges

Varying Workload Pattern

- "Wide" - simultaneously schedule 5k -10k Spark applications with small number of executors
- "Deep" - schedule a few jobs with 1k - 8k executors with heavy I/O on external FileSystem
- "Wide and Deep"- continuously schedule around 2k applications per minute, each app requests hundreds executors.
- Fluctuating batches applications / scheduled daily or weekly jobs
- Requirements for gang / batch scheduling

Stress on Kubernetes

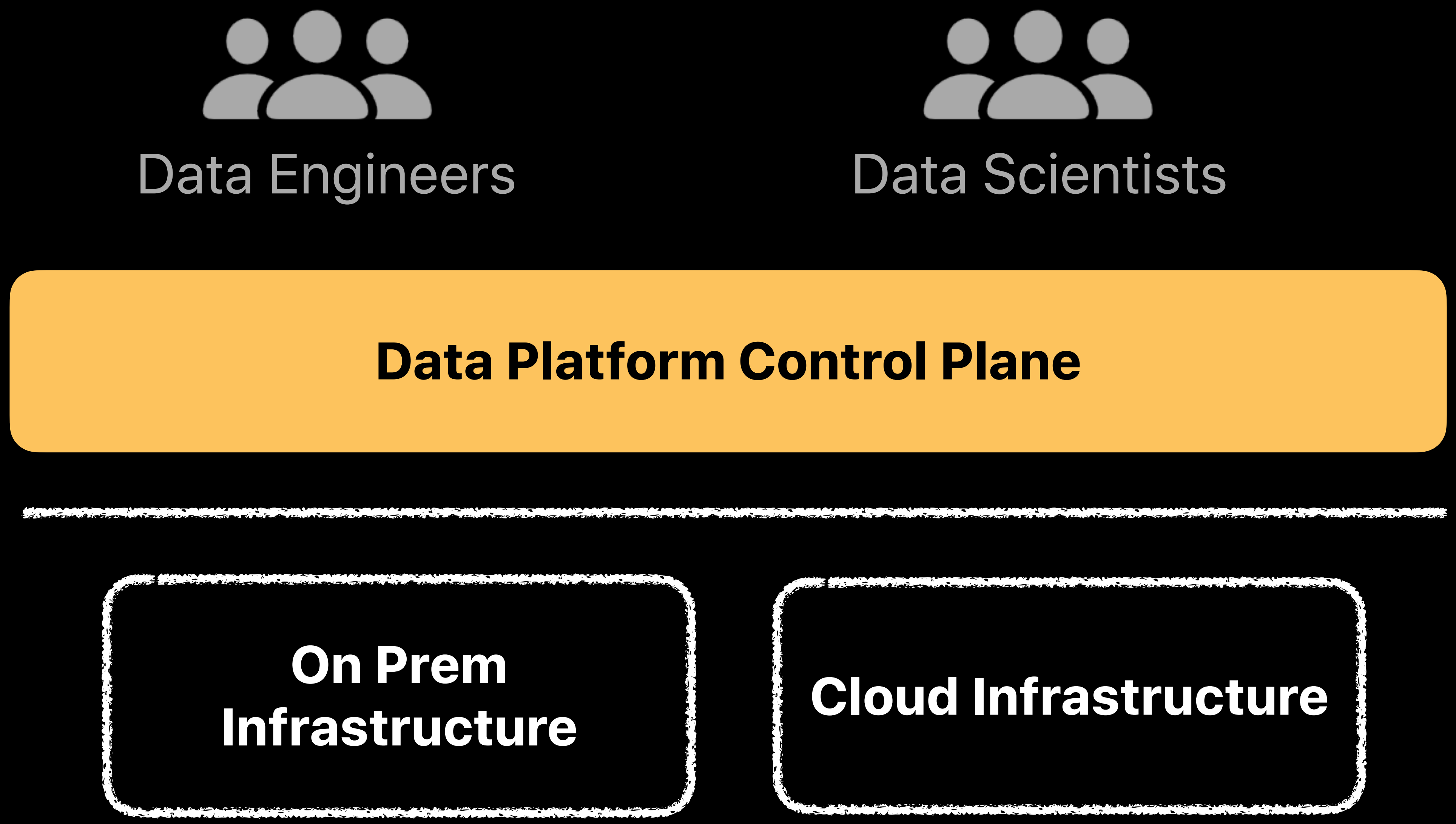
- Expect massive Spark Applications created simultaneously at peak time
- Each UPDATE of Spark Application CRD results in a new version
- Pod churn



Fills 8GB ETCD in 10 mintes

One Interface over Multi-Cloud

- Requirements for bring-you-own-cloud
  - Leverage additional compute resources
  - Feature Parity with On-prem
- Fast & Easy cluster onboarding
- Access control



Strategies

Optimize Kubernetes for Spark Workload

- Kubernetes optimization for write throughput
  - Increase ETCD size to beyond 8GB
  - Compaction tuning on ETCD
  - Separate storage for resources and events
- Working with cluster auto-scaler
- Priority class and preemption
- IPv4 exhaustion
  - Use cluster-routable IPs whenever possible
  - IPv6 upgrade for all services

Spark Orchestration at Scale

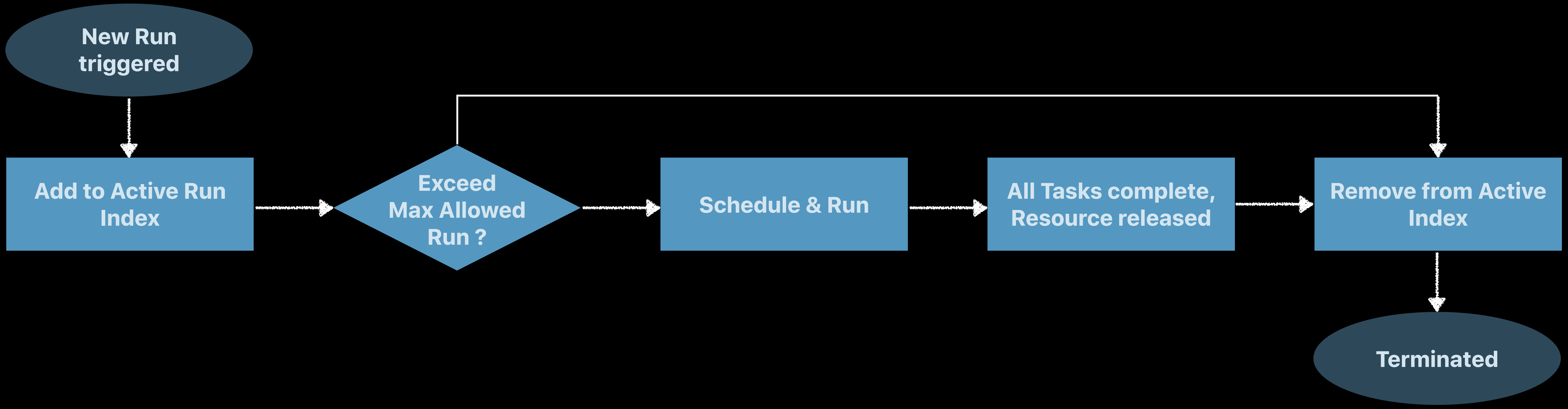
Concurrency Check

- Typical concurrency policies (available in oss operator)
  - e.g. Allow / Forbid / Replace
  - Possibility to flood cluster for batch job ('wide' case)
- Advanced concurrency control
  - Limit the max number of concurrent runs globally
  - Scheduled / ad-hoc runs

```
spec:
    schedule: "@every 5m"
    concurrencyPolicy: Allow
```

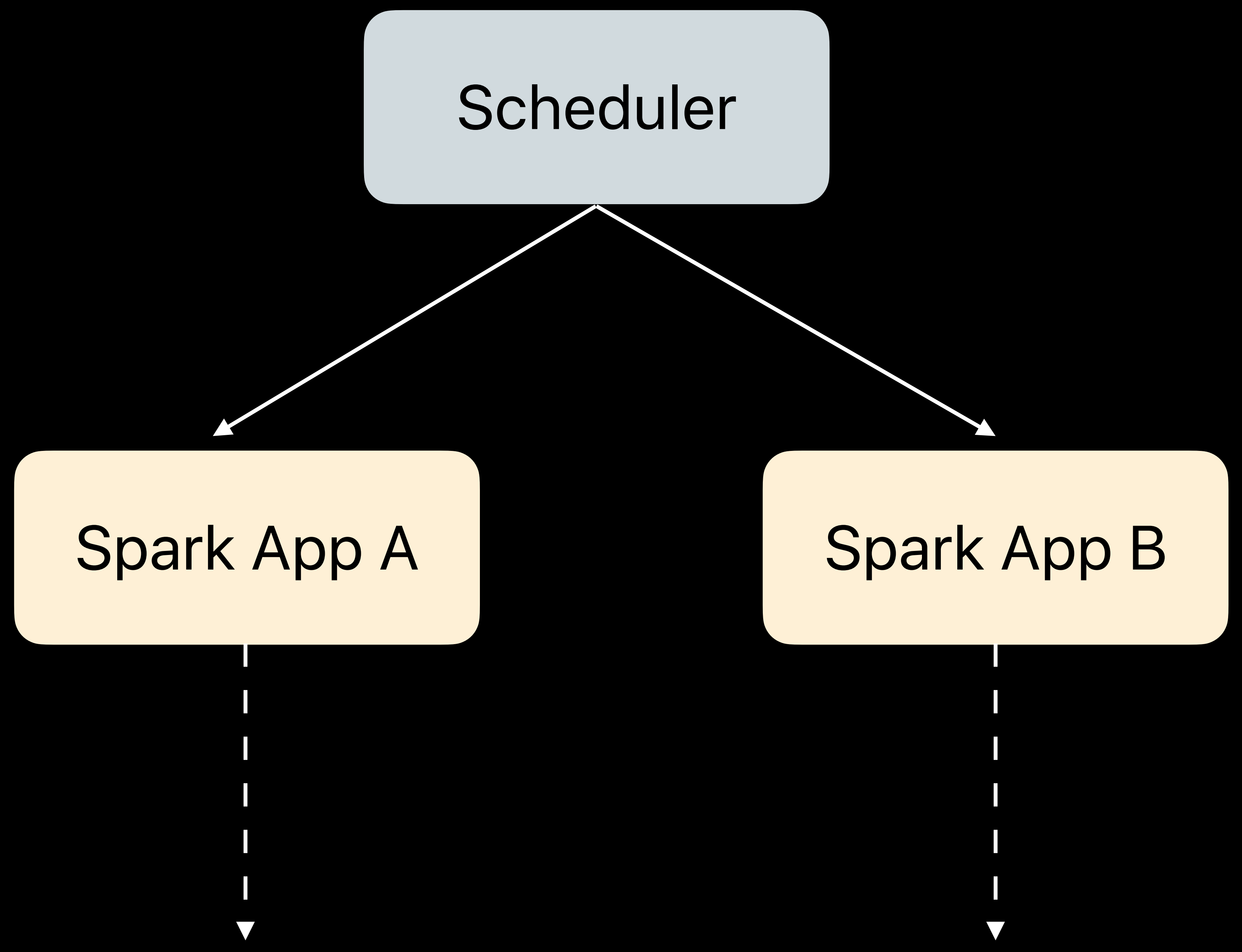
Granular Concurrency Check at Orchestration

```
properties:
    spark.executor.instances: 4
    spark.apple.job.max.concurrent.runs: 100                    // global max concurrent runs
    spark.apple.manualRun.max.concurrent.runs: 10               // max concurrent ad-hoc run
    spark.apple.triggers.hourlyJob.maxConcurrentRuns: 5    // max concurrent scheduled run from trigger
```



Avoid Partially Running Applications

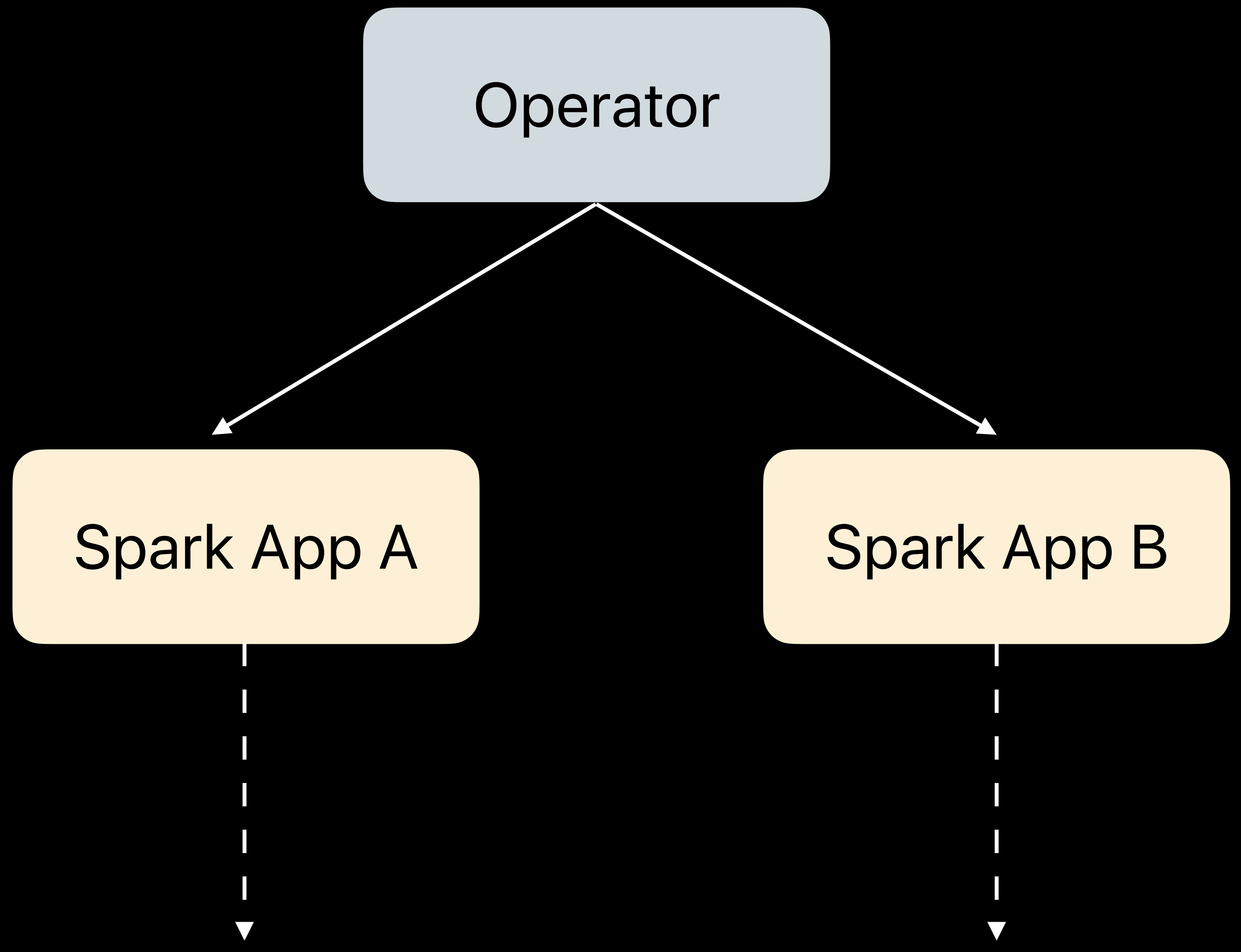
- Gang-scheduling solutions
    - Batch integration for operator
    - Driver / executor pod group support
    - Apple collaboration in the community
- Operator-side timeouts helps in
    - Proactively terminates when not enough executors registered after given threshold
    - Restart policy for infrastructure reasons



Neither can proceed at partial capacity,
waiting for more resources,
causing deadlock

Avoid Partially Running Applications

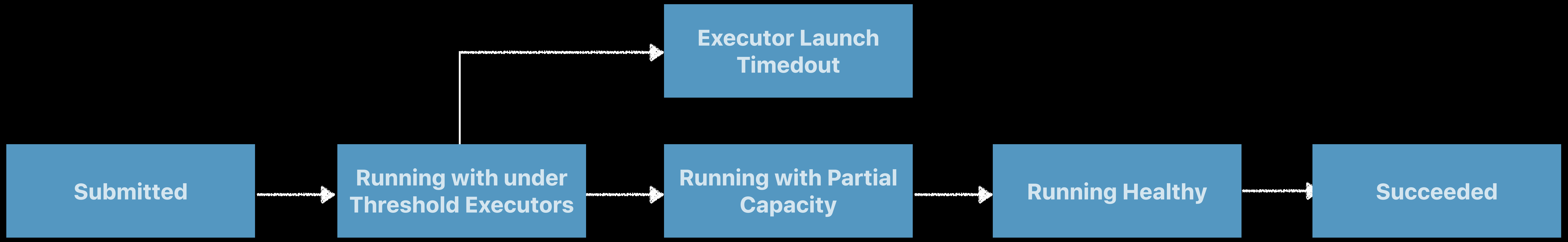
- Gang-scheduling solutions
  - Batch integration for operator
  - Driver / executor pod group support
  - Apple collaboration in the community
- Operator-side timeouts helps in
  - Proactively terminates when not enough executors registered after given threshold
  - Restart policy for infrastructure reasons



Terminates partially running applications if needed

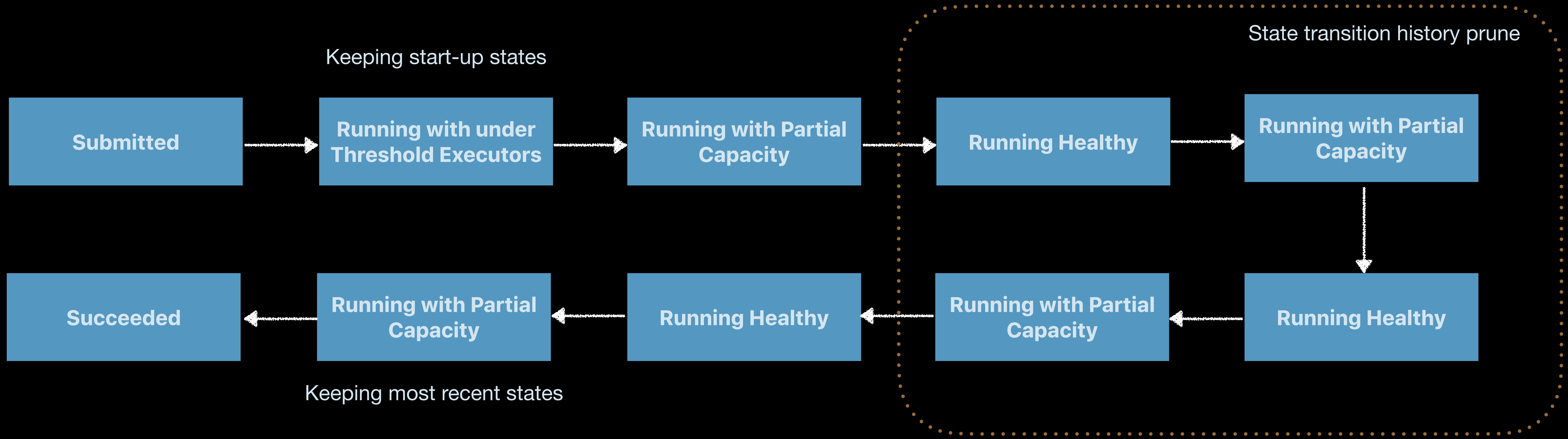
Timeout Partially Running Applications

```
properties:
  spark.executor.instances: 400
  spark.apple.executors.min.threshold.ratio: 0.8    // requires at least 80% of total executors
  spark.apple.executors.startup.timeout: 60000    // terminate if cannot get enough executors after 10 min
  spark.apple.backoff.duration.on.failure.ms: 30000   // backoff 5 min before attempting restart
```


Executor Launch Timedout
Submitted
Running with under Threshold Executors
Running with Partial Capacity
Running Healthy
Succeeded

Mitigate Cluster Storage Stress

- Detailed running state may results in large state transition history
  - Executor lost / evicted / preempted, job may swing between states
  - History pruning within same attempt
- Orchestrator acknowledgement-based state history pruning

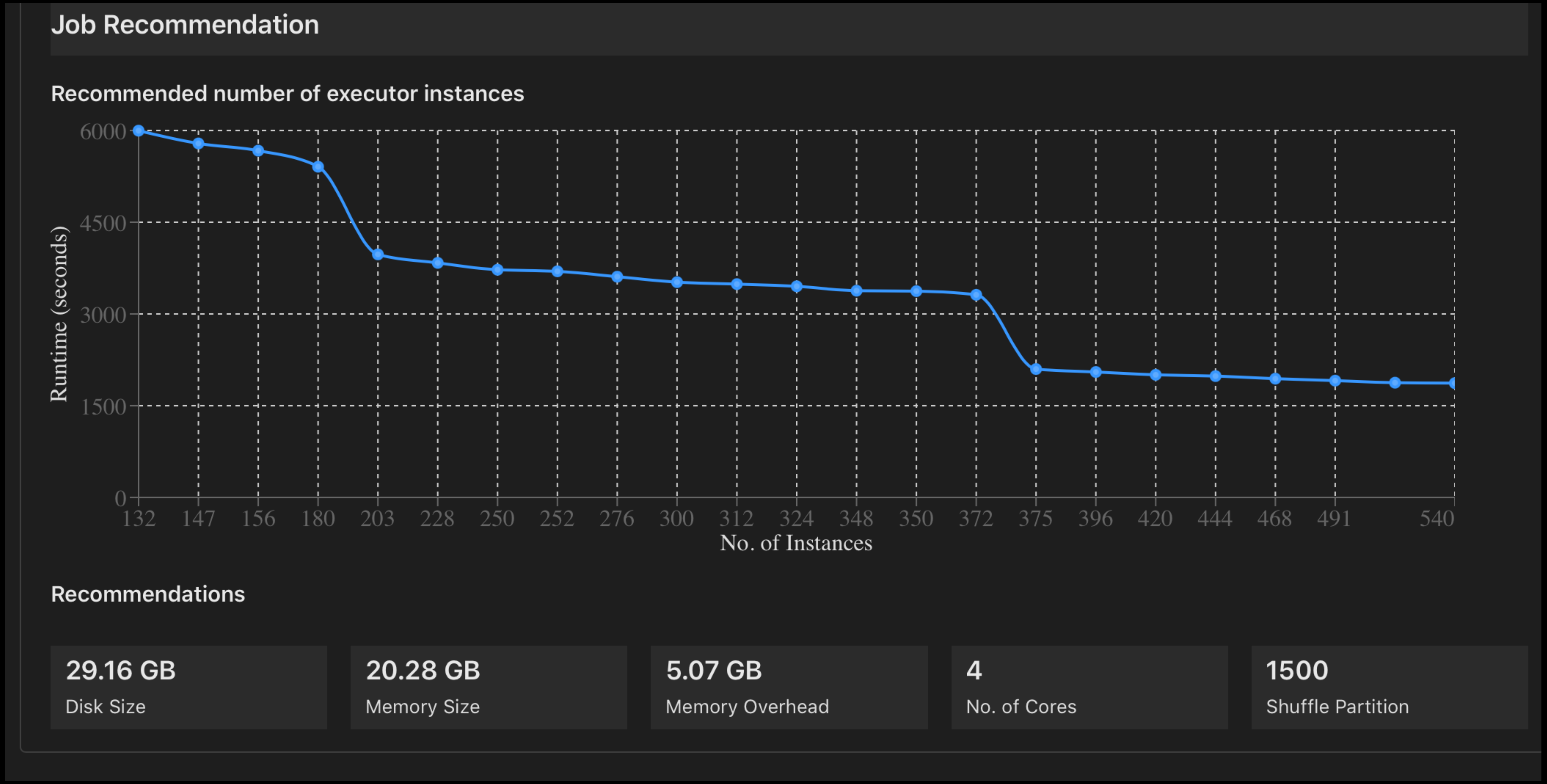


Keeping start-up states
State transition history prune
Submitted
Running with under Threshold Executors
Running with Partial Capacity
Running Healthy
Running with Partial Capacity
Running Healthy
Running with Partial Capacity
Running Healthy
Running with Partial Capacity
Succeeded
Keeping most recent states

Utilization-based Allocation Recommendation

- Setting resource requests and limits
  - Low allocation leads eviction
  - Over allocation means resource waste

- Use Spark metrics from previous runs
  - Add listener for metrics collection
  - Aggregate historical run data over Spark
  - Provide recommendations for future run



Job Recommendation

Recommended number of executor instances

Recommendations

29.16 GB	20.28 GB	5.07 GB	4	1500
Disk Size	Memory Size	Memory Overhead	No. of Cores	Shuffle Partition

Dynamic Allocation

- Dynamic Allocation is enabled for Spark 2.4 and above
- Batching pod requests
- Shuffle tracking and graceful decommission [SPARK-20624]
- External shuffle storage based on PVC

```
// enable dynamic allocation
spark.dynamicAllocation.enabled
spark.dynamicAllocation.minExecutors
spark.dynamicAllocation.maxExecutors
spark.dynamicAllocation.executorIdleTimeout
spark.kubernetes.allocation.batch.size

// enable state tracking
spark.dynamicAllocation.shuffleTracking.enabled
spark.dynamicAllocation.shuffleTracking.timeout
spark.dynamicAllocation.cachedExecutorIdleTimeout
```

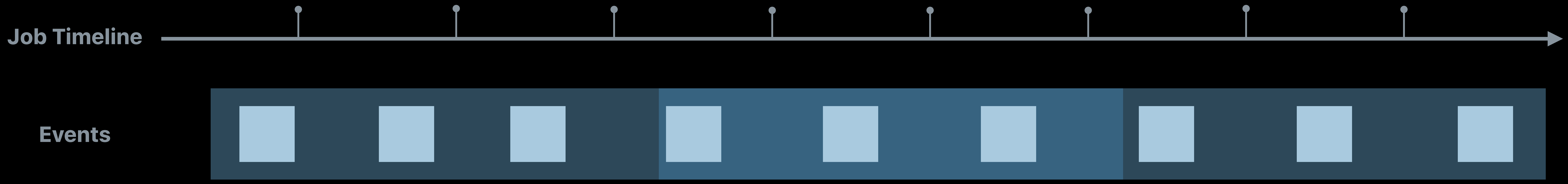
```
// or do dynamic allocation with migration
spark.decommission.enabled
spark.executor.decommission.killInterval
spark.storage.decommission.enabled
spark.storage.decommission.rddBlocks.enabled

// and external shuffle storage
spark.shuffle.externalStorage.enabled
spark.shuffle.externalStorage.backend
spark.shuffle.externalStorage.bucket
```

Scale up Spark on Kubernetes

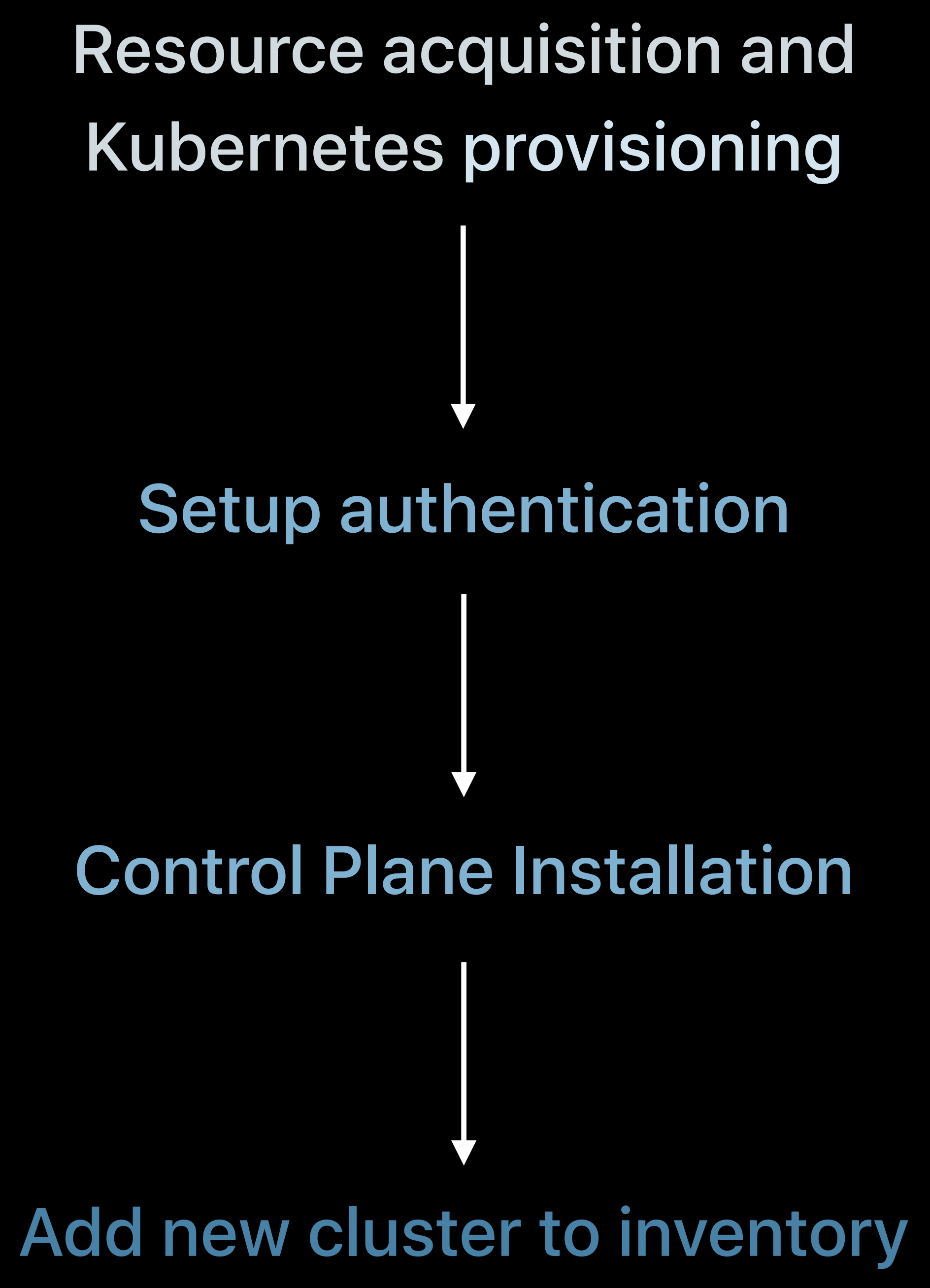
History Server

- Multi-tenant history server per cluster
- History server based off version 2.4, serving all versions
- Stores aggregated view of most recent jobs



Push-button Cloud Management

- Automated cluster installation
  - Cloud provider setup (resource acquisition, IAM .etc)
  - Kubernetes cluster provisioning
  - In-kubernetes components installation
  - Infrastructure as Code (IaC)
- Feature parity
  - CI / CD for control plane update
  - Logging & telemetry integration
  - Security
    - Team-based cluster access
    - Access control for Spark UI



Resource acquisition and
Kubernetes provisioning

Setup authentication

Control Plane Installation

Add new cluster to inventory

Lessons

Scale up Spark on Kubernetes

- Configure k8s etcd storage size and compaction for write throughput
- Batch scheduling and infrastructure timeouts
- Design concurrency policy for 'wide' use cases
- Avoid over-allocation by analyzing historical runs
- Cluster-level auto-scaler and app-level dynamic allocation for cost efficiency
- History-server scaling up
- Portable, provider-agnostic in-cluster controlplane components

We are hiring!