SHANGHAI JIAO TONG UNIVERSITY

Service Management for Stream Processing

Yalun Lin Hsu

2020-10-26

SAIL

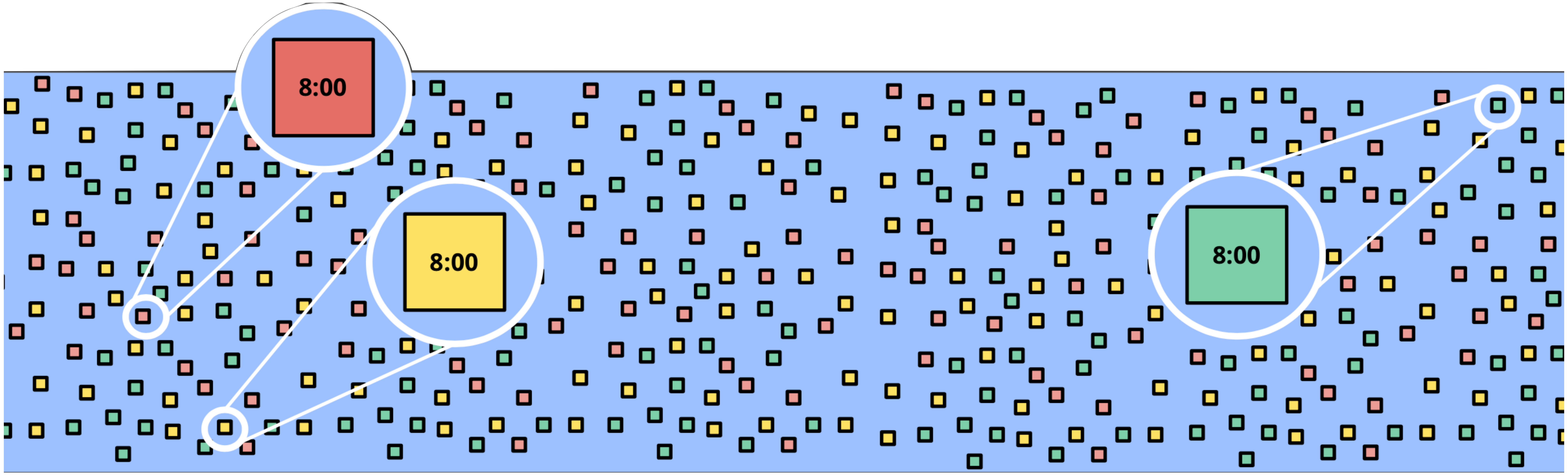
ICDE '20

Turbine

Facebook’s Service Management Platform for Stream Processing

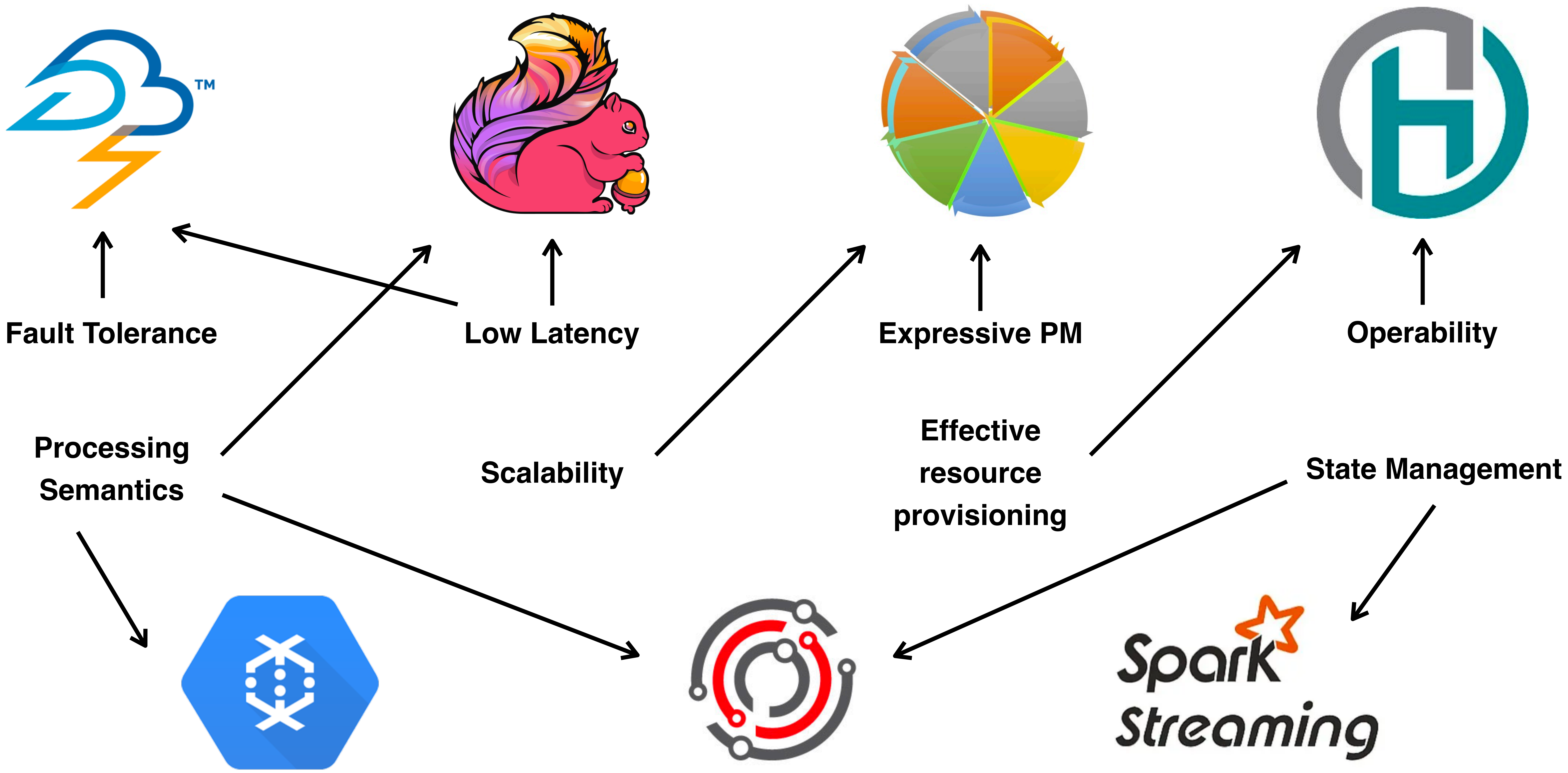
https://doi.org/10.14778/3137765.3137777

Review: Streaming

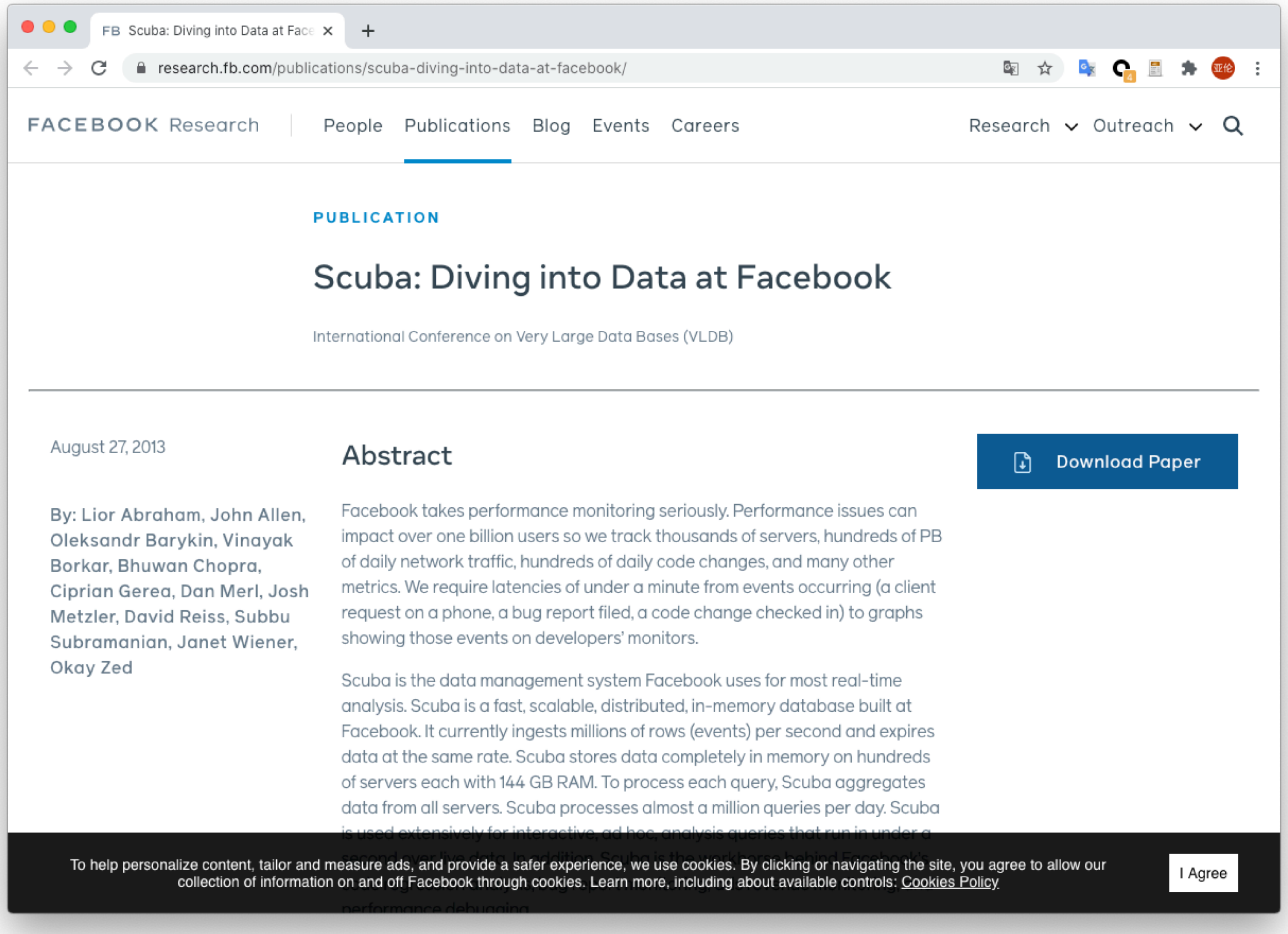


Data can be infinitely big with unknown delays.

Systems for large-scale distributed streaming



Example: Scuba Tailer in Facebook



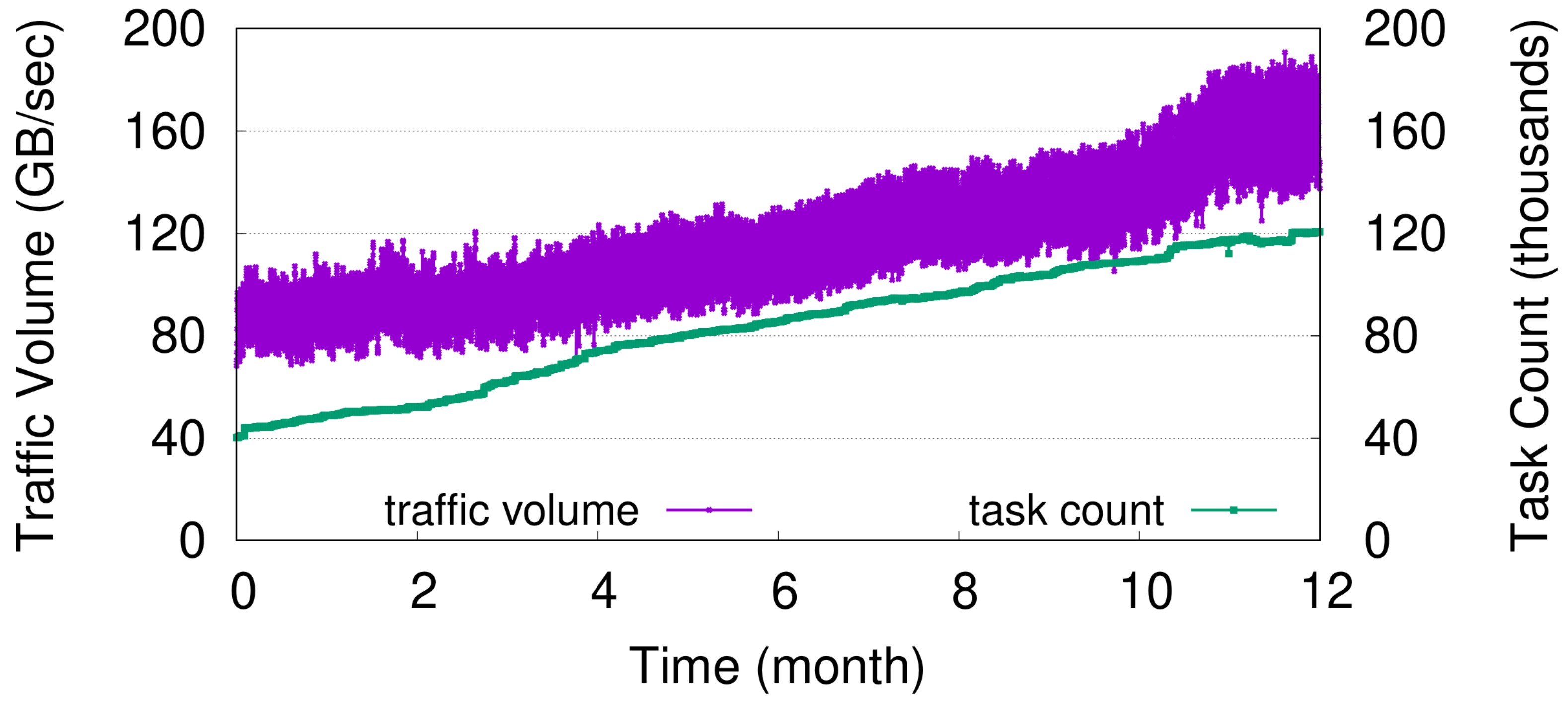


Fig. 1: The growth of Scuba Tailer service in a one-year interval, in terms of task count and input traffic volume.

Requirements of Streaming in Facebook

- Low latency analysis of site content interaction, recommendation-related activities …
- A large number of stateless and stateful stream processing applications;
- Strict Service Level Objectives (SLOs) with low downtime and processing lag

Turbine

- A scheduling and lifecycle management framework featuring fast task scheduling and failure recovery;
- An efficient predictive auto scaler that adjusts resource allocation in multiple dimensions;
- An ACIDF application update mechanism.

Architecture

Turbine Architecture

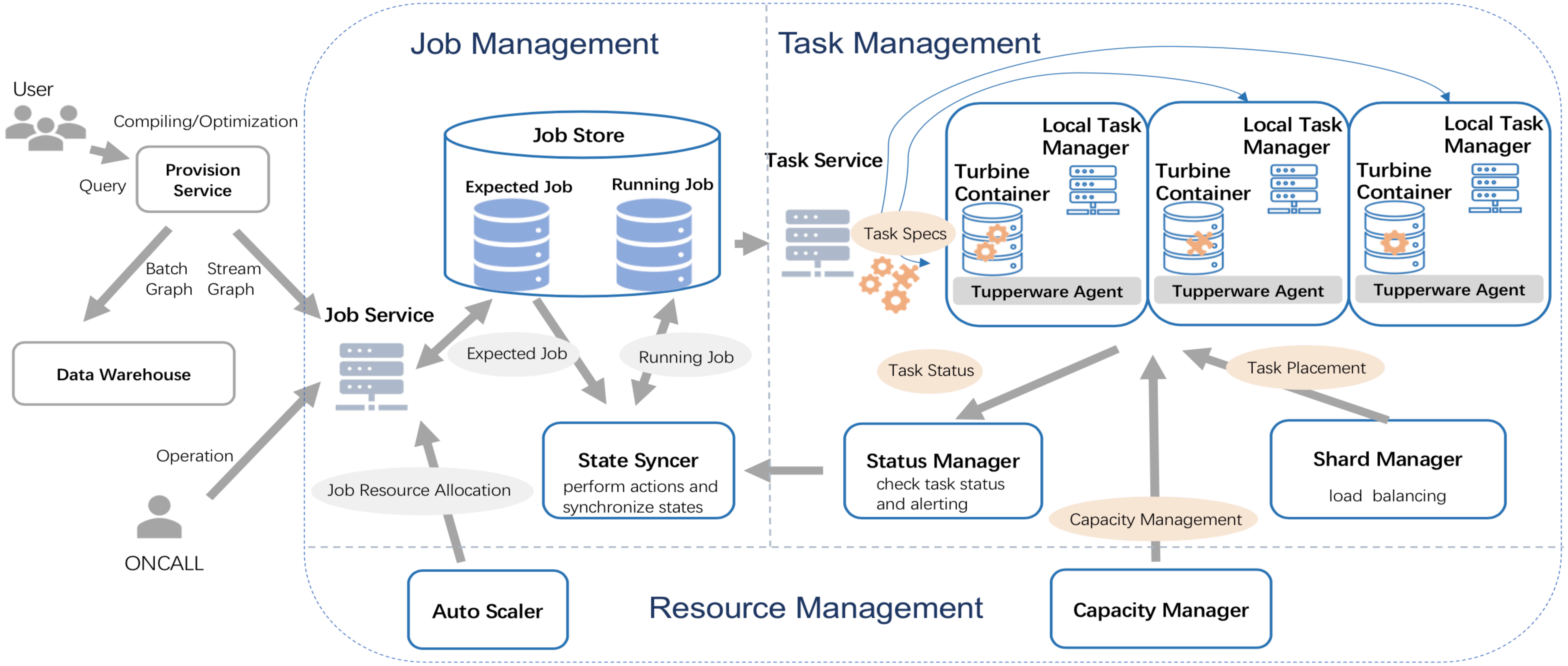
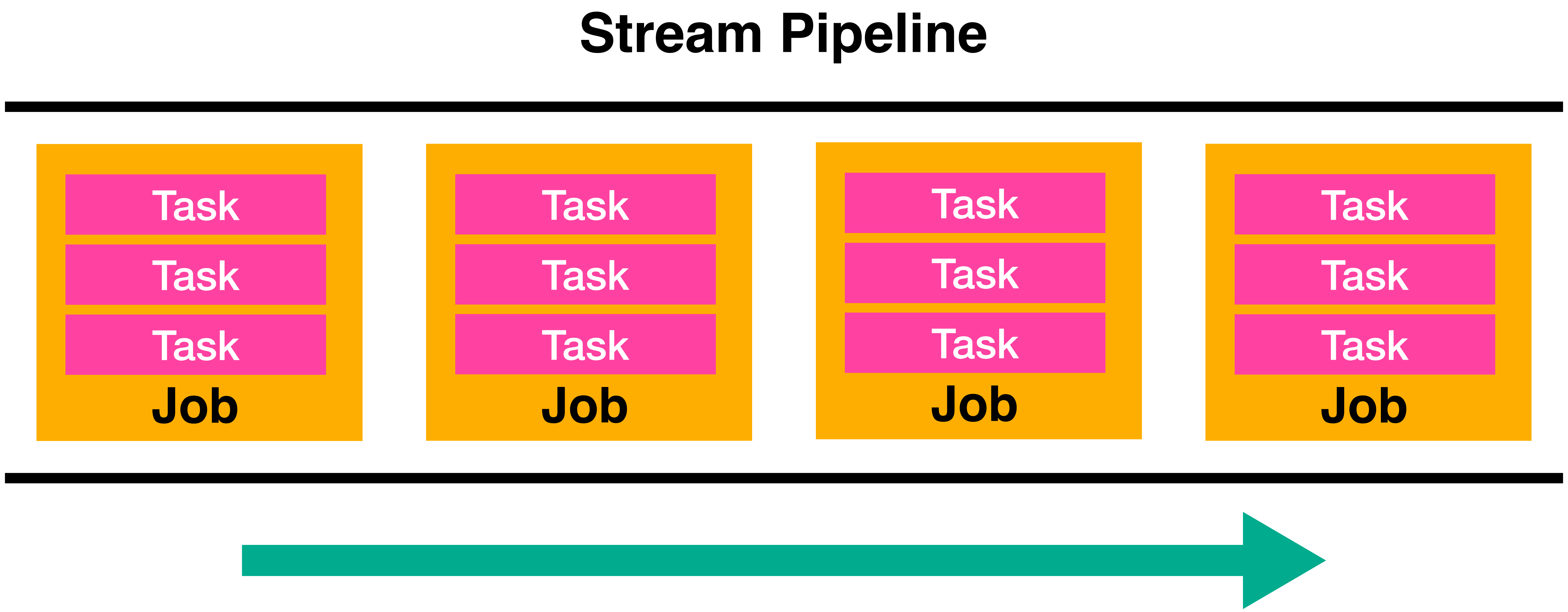


Fig. 2: Turbine's system architecture to manage stream processing services.

Turbine Stream Pipeline



Turbine Architecture

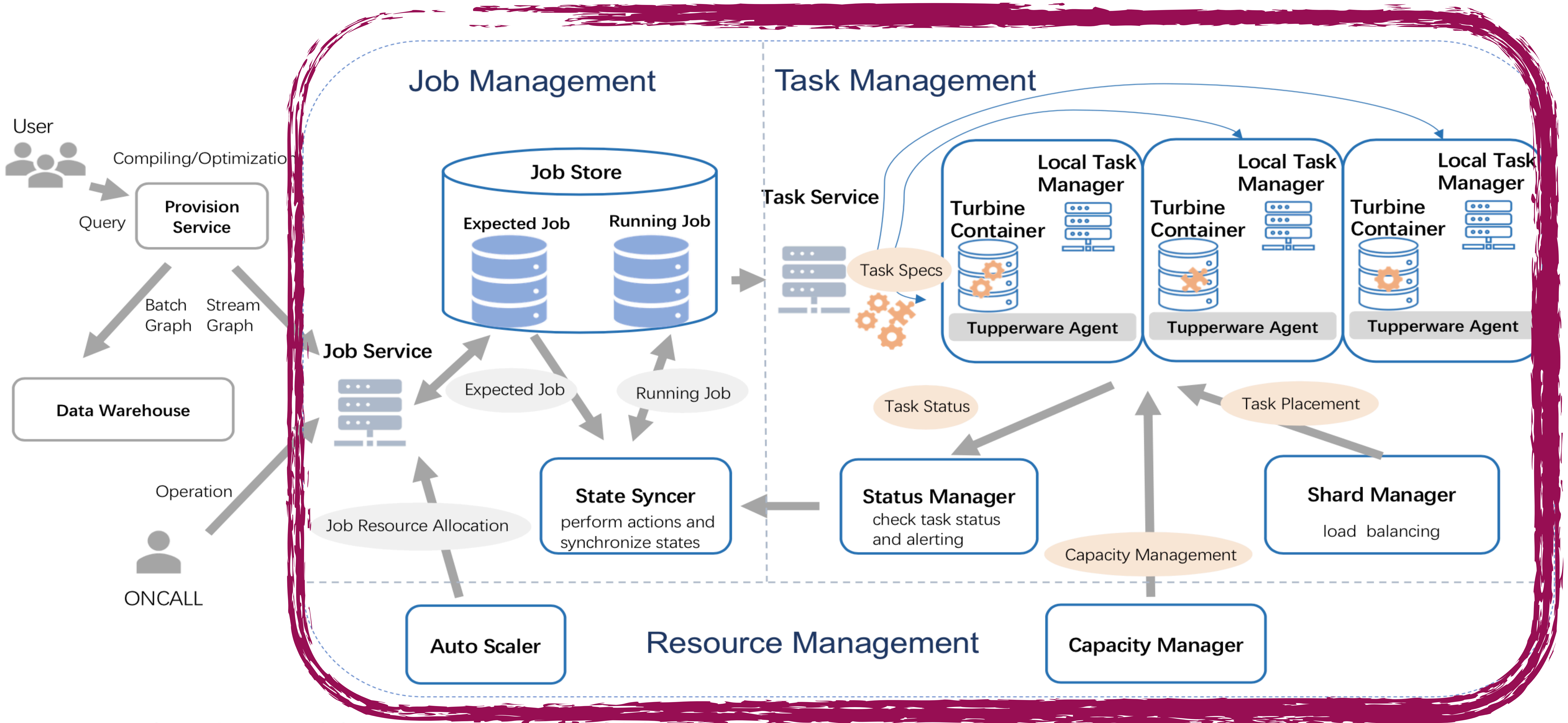


Fig. 2: Turbine's system architecture to manage stream processing services.

Turbine Core Components

Job Management

What to run

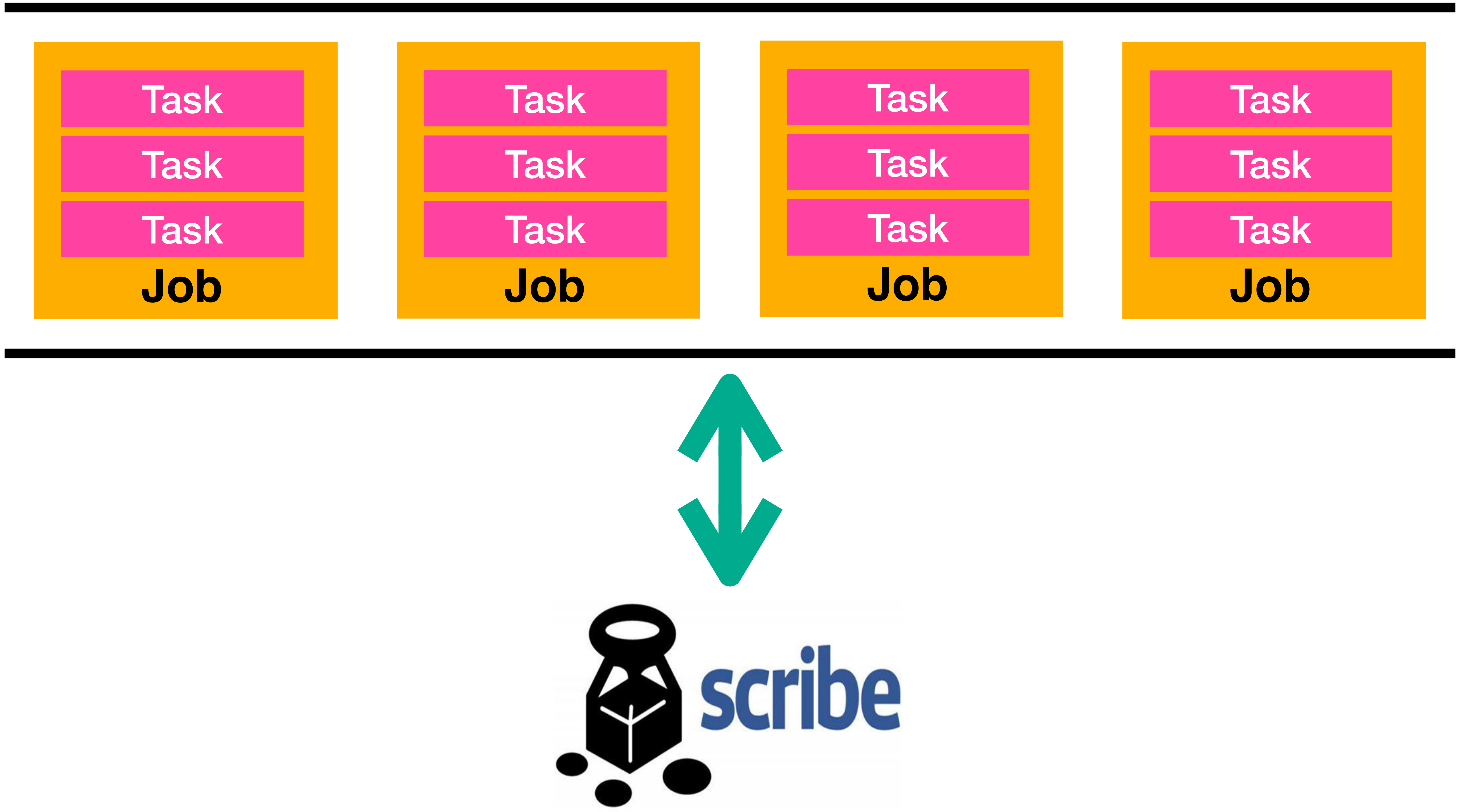
Task Management

Where to run

Resource Management

How to run

Communication between Jobs

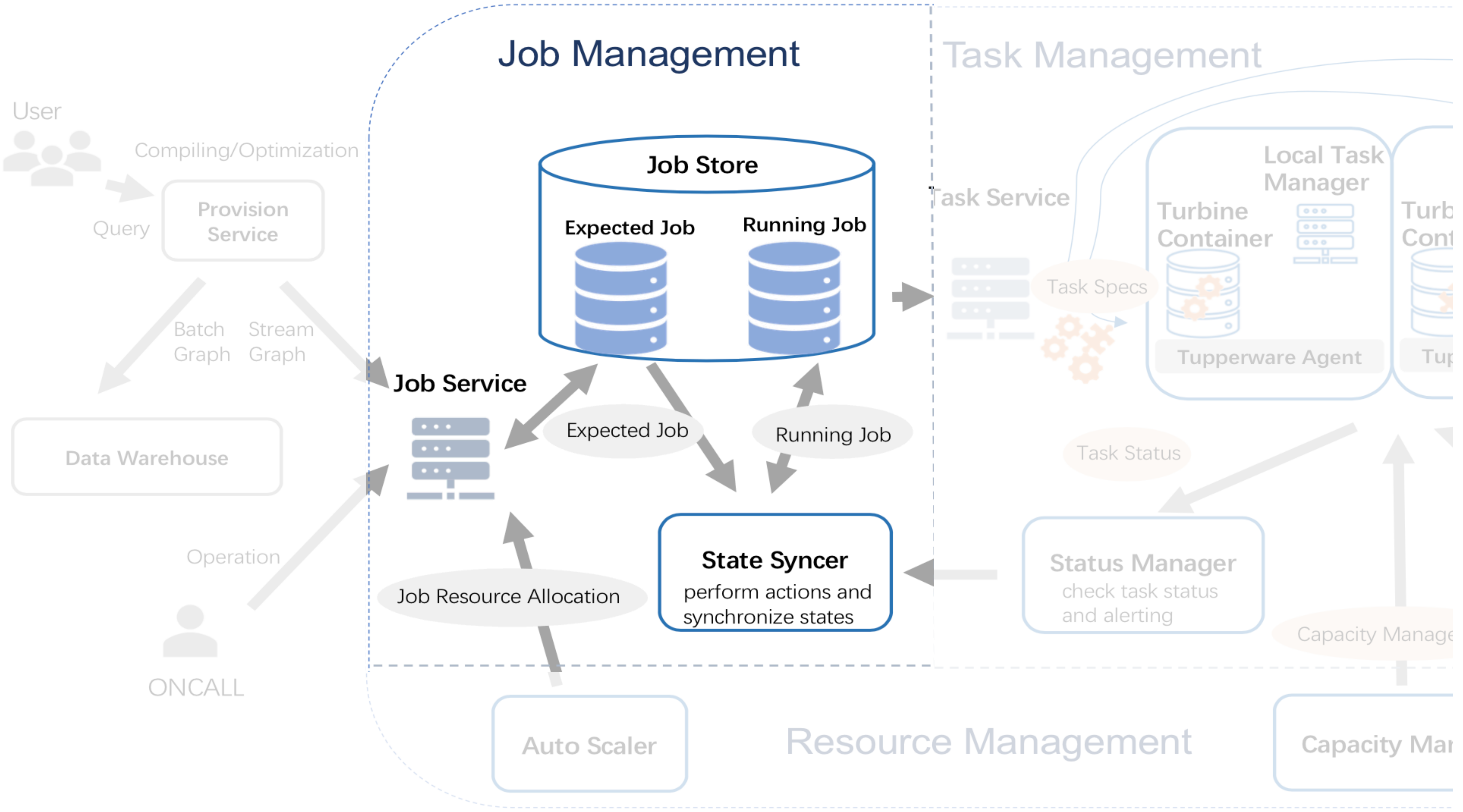


Job Management

Job Management Components

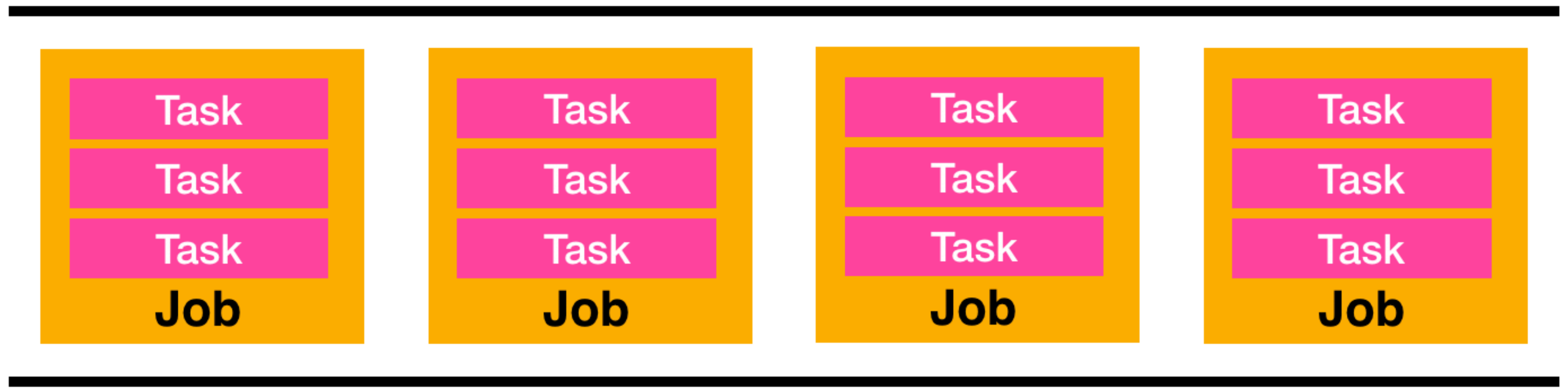
To guarantee ACIDF:

- **Job Store**: Contains the current and desired config params for each job;
- **Job Service**: Guarantees job changes are committed;
- **State Syncer**: Executes job update actions.



Job Configurations Update Issue

- **Isolation**
- **Consistency**



T0: Run 10 tasks

T1: Auto Scaler (Internal) - 15 tasks

T1: Oncall1 (External User) - 20 tasks

T1: Oncall2 (External User) - 30 tasks

Hierarchical Expected Job Configuration

Expected Job Table
Base Configuration
Provisioner Configuration
Scaler Configuration
Oncall Configuration

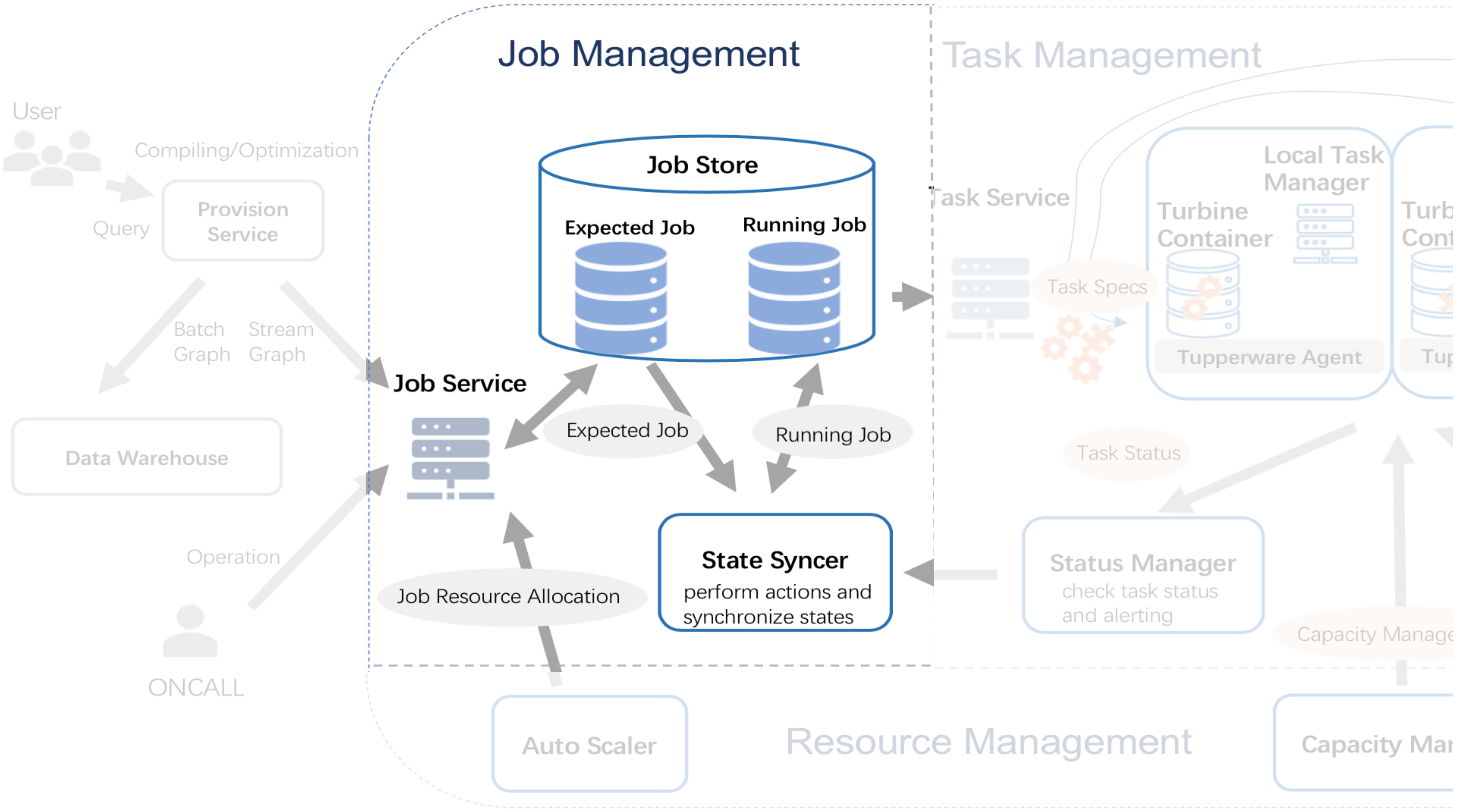
Running Job Table
Running Configuration

Algorithm 1 Merge JSON Configs

```
procedure LAYERCONFIGS(bottomConfig, topConfig)
    layeredConfig ← bottomConfig.copy()
    for each (key, topValue) ∈ topConfig do
        if isInstance(topValue, JsonMap) && (key in bottomConfig) then
            layeredValue ← layerConfigs(bottomConfig.get(key), topValue)
            layeredConfig.update(key, layeredValue)
        else
            layerConfig.update(key, topValue)
        end if
    end for
    return layeredConfig
end procedure
```

State Syncer

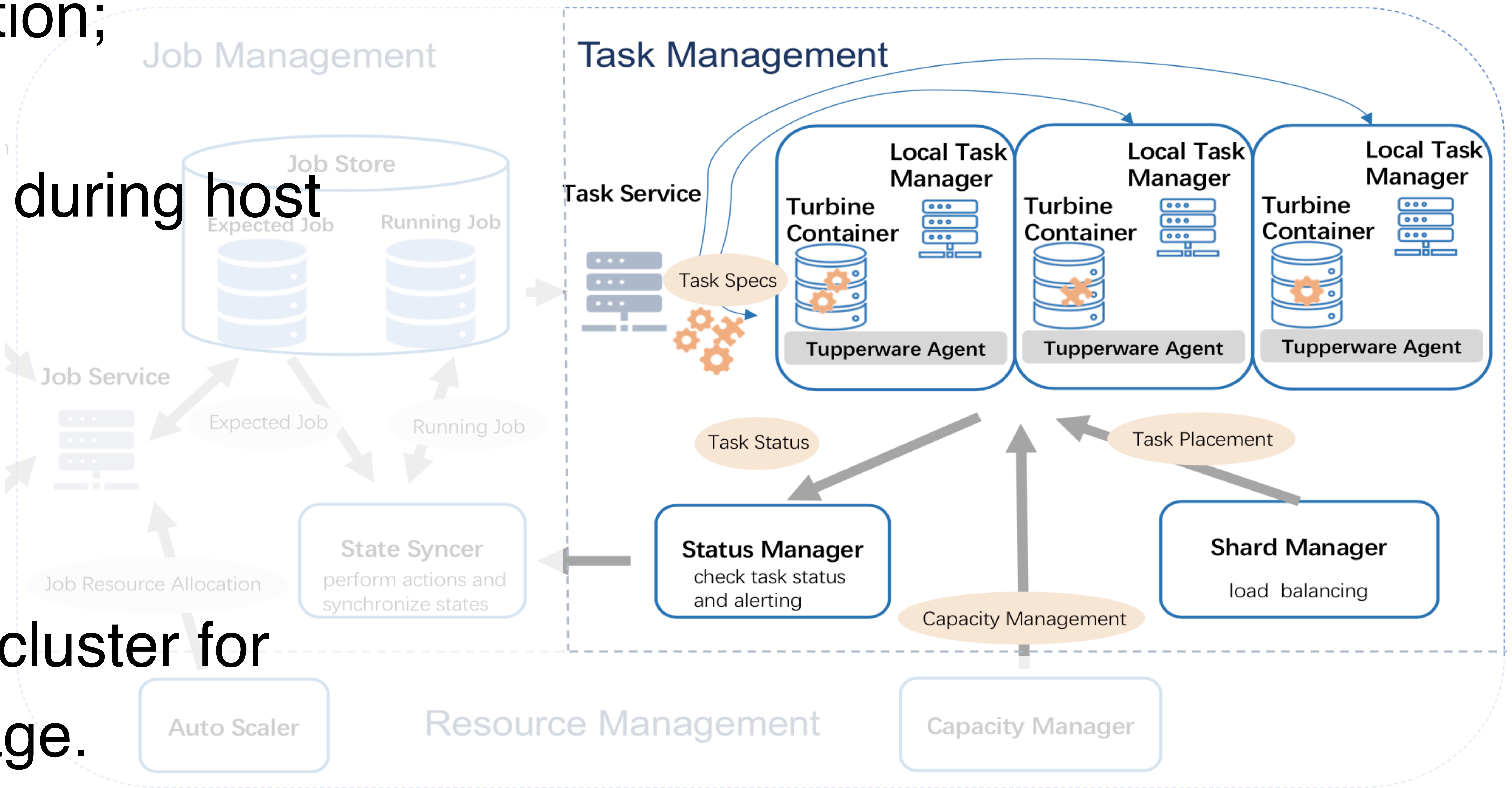
- **Atomicity**
- **Fault-tolerance**
- **Durability**



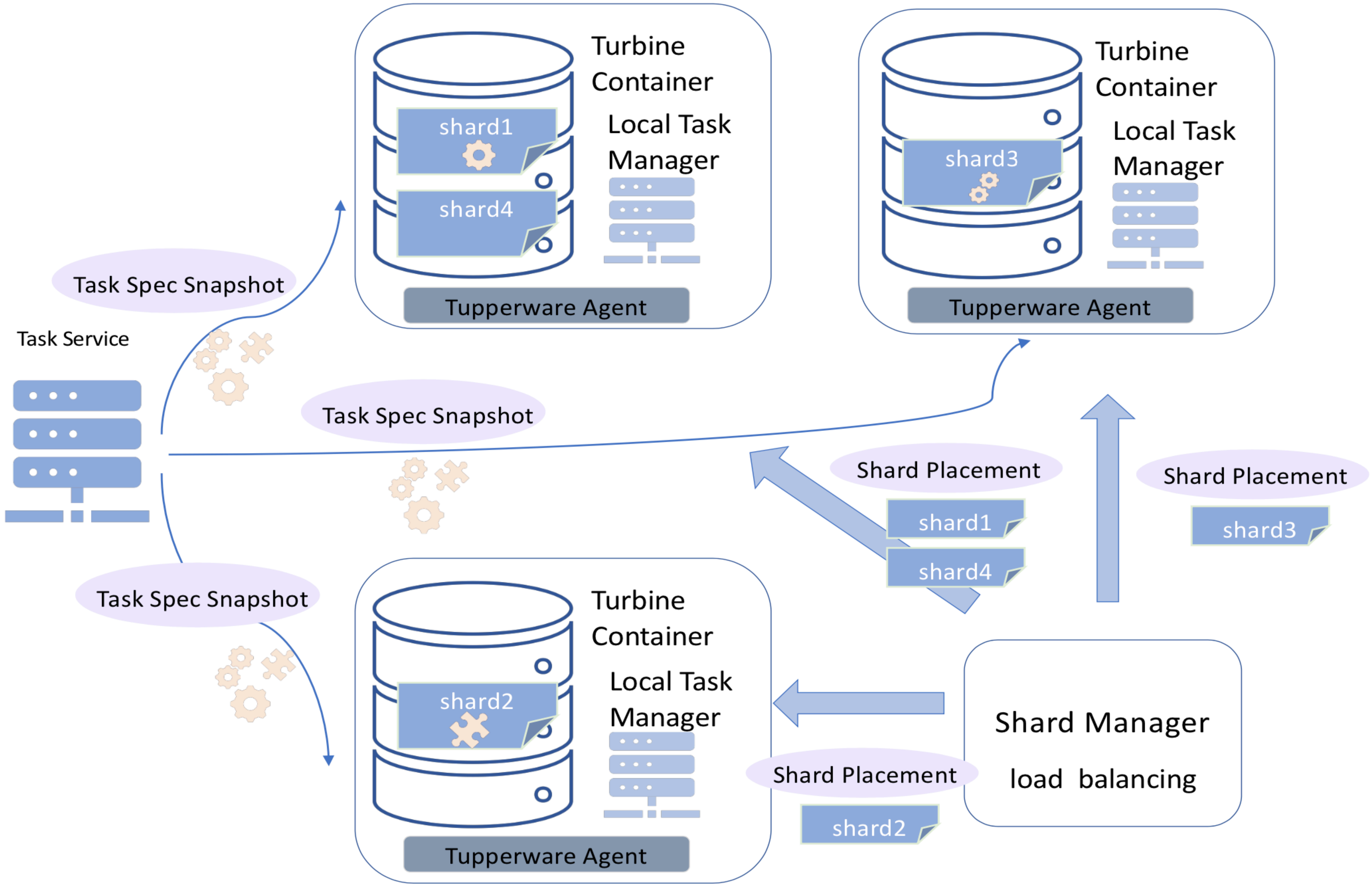
Task Placement & Load Balancing

Goal of Task Management

- Schedule tasks without duplication;
- Fail-over tasks to healthy hosts during host failures;
- Restart tasks upon crashes;
- Load balance tasks across the cluster for even CPU, Memory and IO usage.



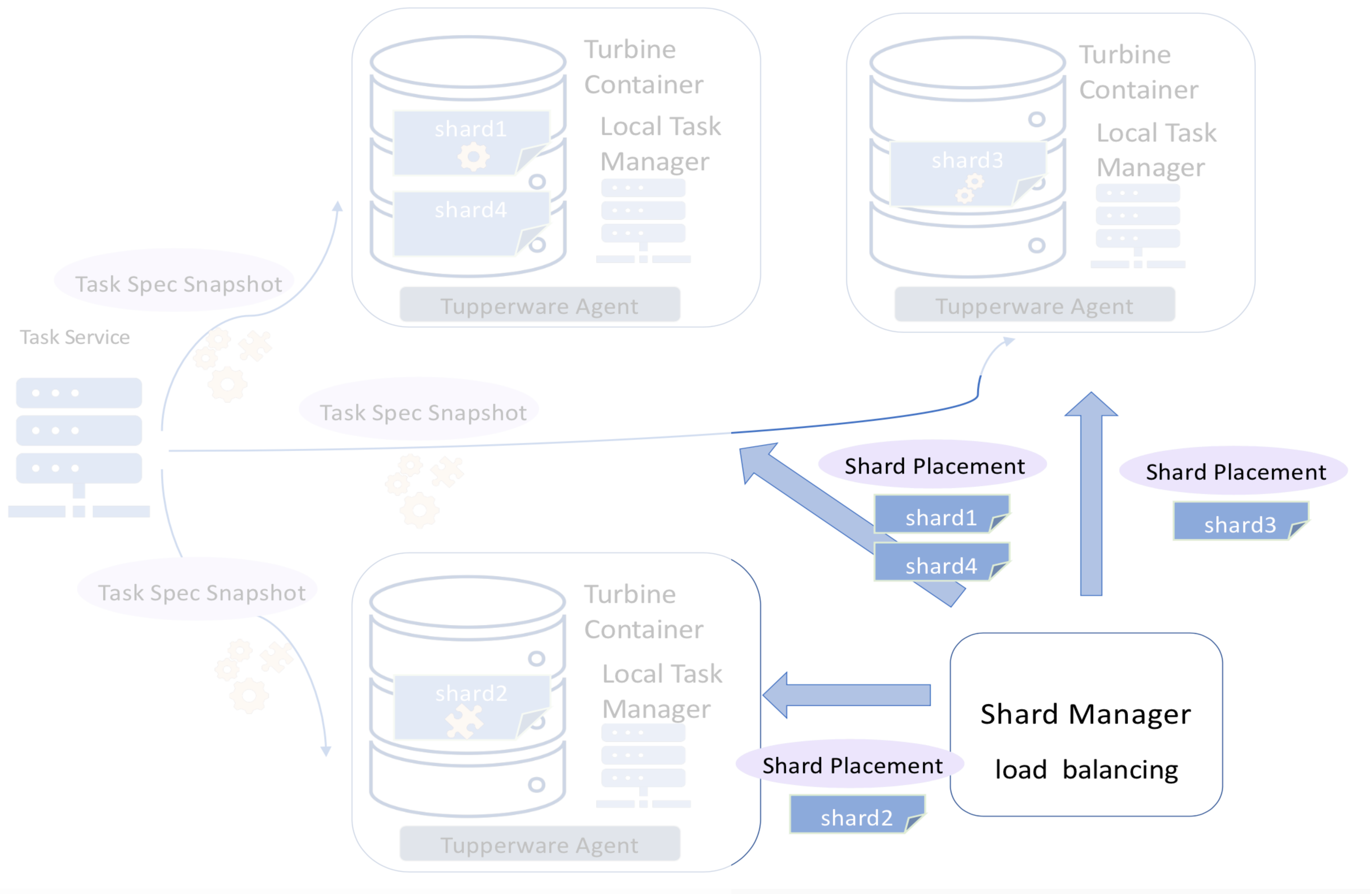
Task Management Overview



Scheduling

Task to Shard Mapping:

- Local Task Manager determines *tasks <—> shard;*
- Fetch Task Service periodically;
- Can handle load balance and failover even when Task Service is down

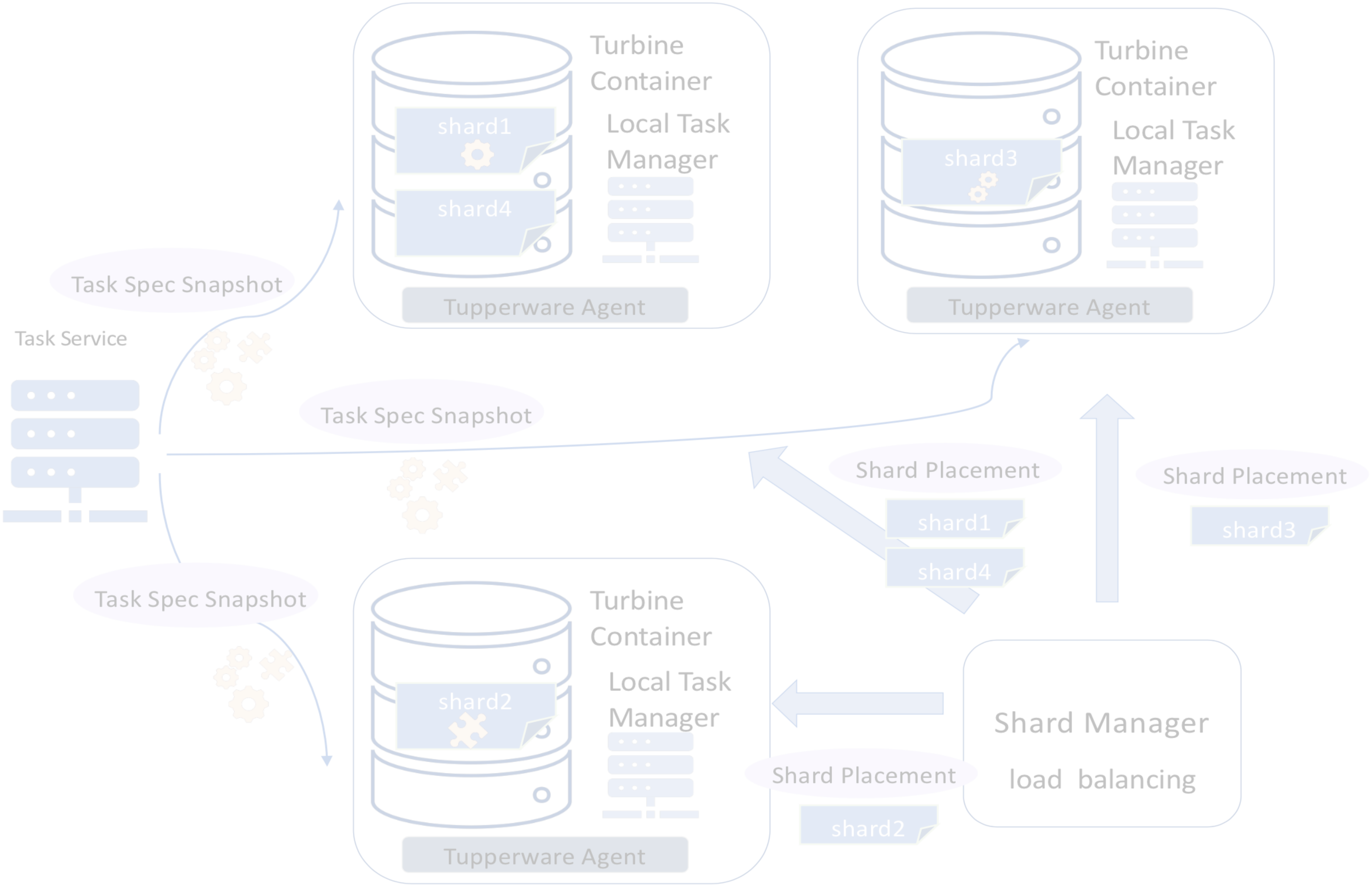


Load Balancing

- Container —> capacity
  Shard —> load

- A bin-packing problem

- Define shard load is **IMPORTANT**
  - Dynamic resource usage (C, C++)
  - Xmx, cgroup usage (JAVA)
  - A background load aggregator thread in each Task Manager



Elastic Resource Management

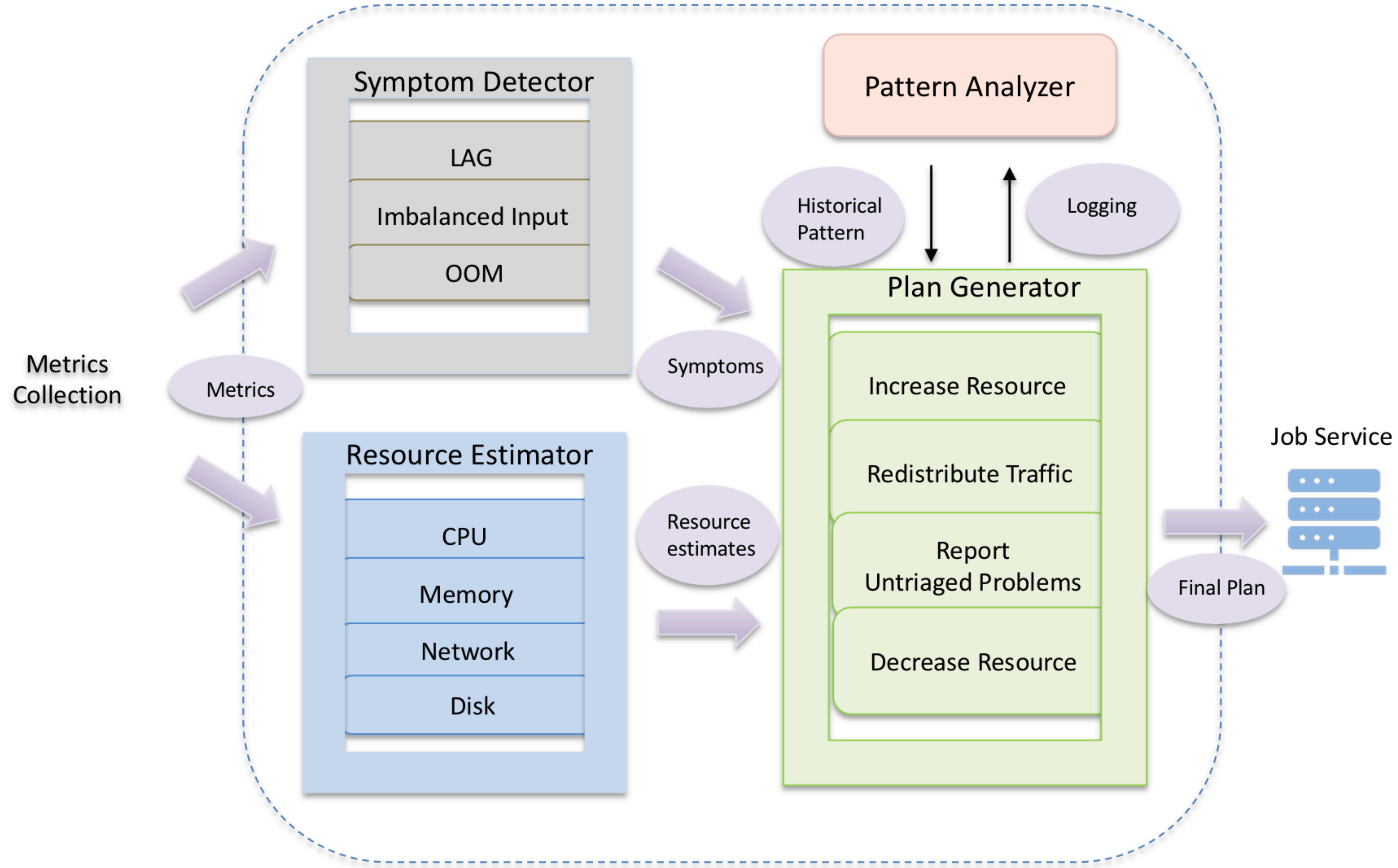
Reactive Auto Scaler — 1st Generation

Use ***Symptom Detectors*** & ***Diagnosis Resolvers*****:**

- Lack of accurate estimation of required resources;
- Without knowing the lower bounds on the resource requirements for a given job;
- Making scaling decisions without understanding the root cause of a particular symptom may amplify the original problem.

The amount of resources needed for a given job is often predictable.

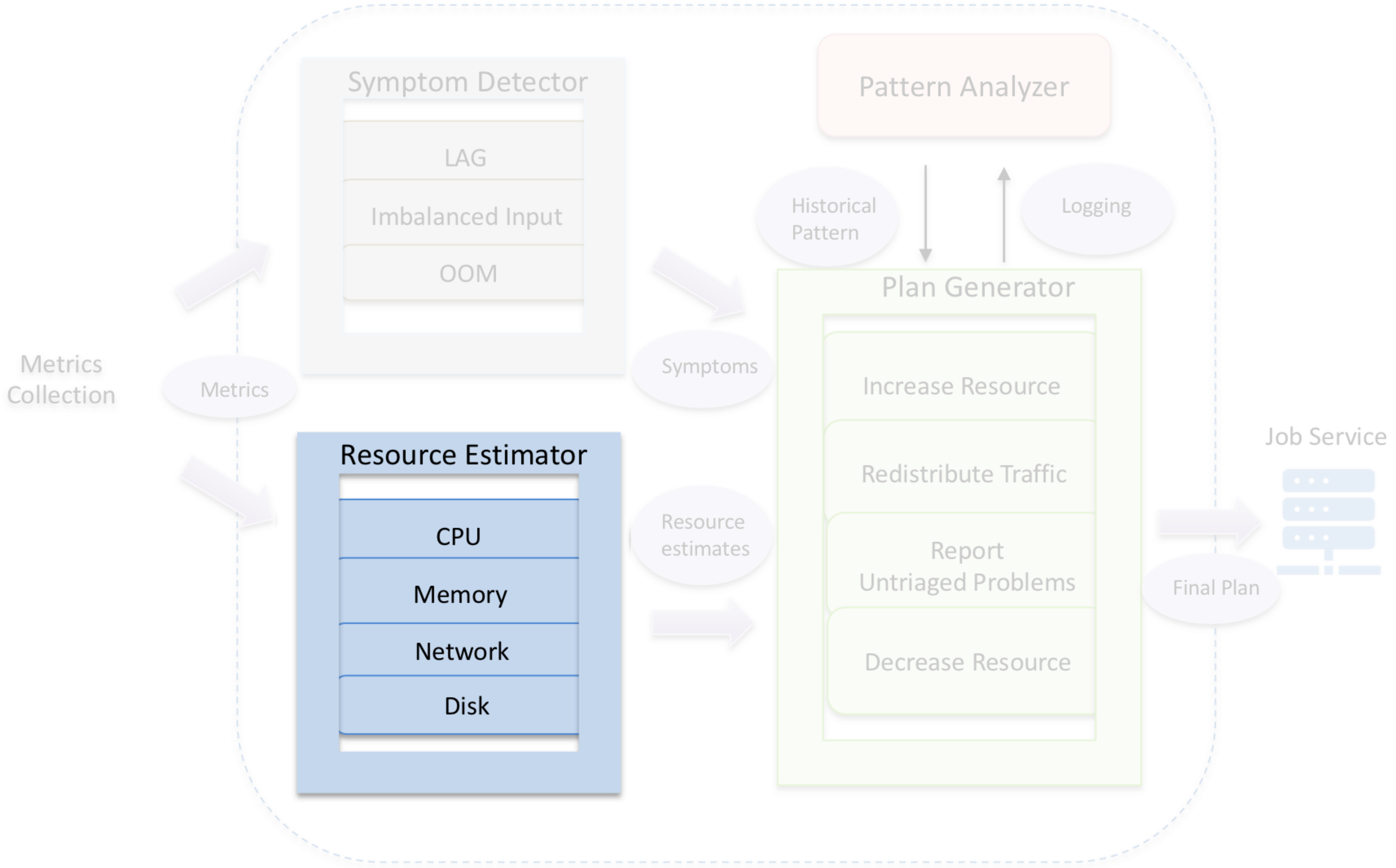
2nd Auto Scaler Overview



Resource Estimator

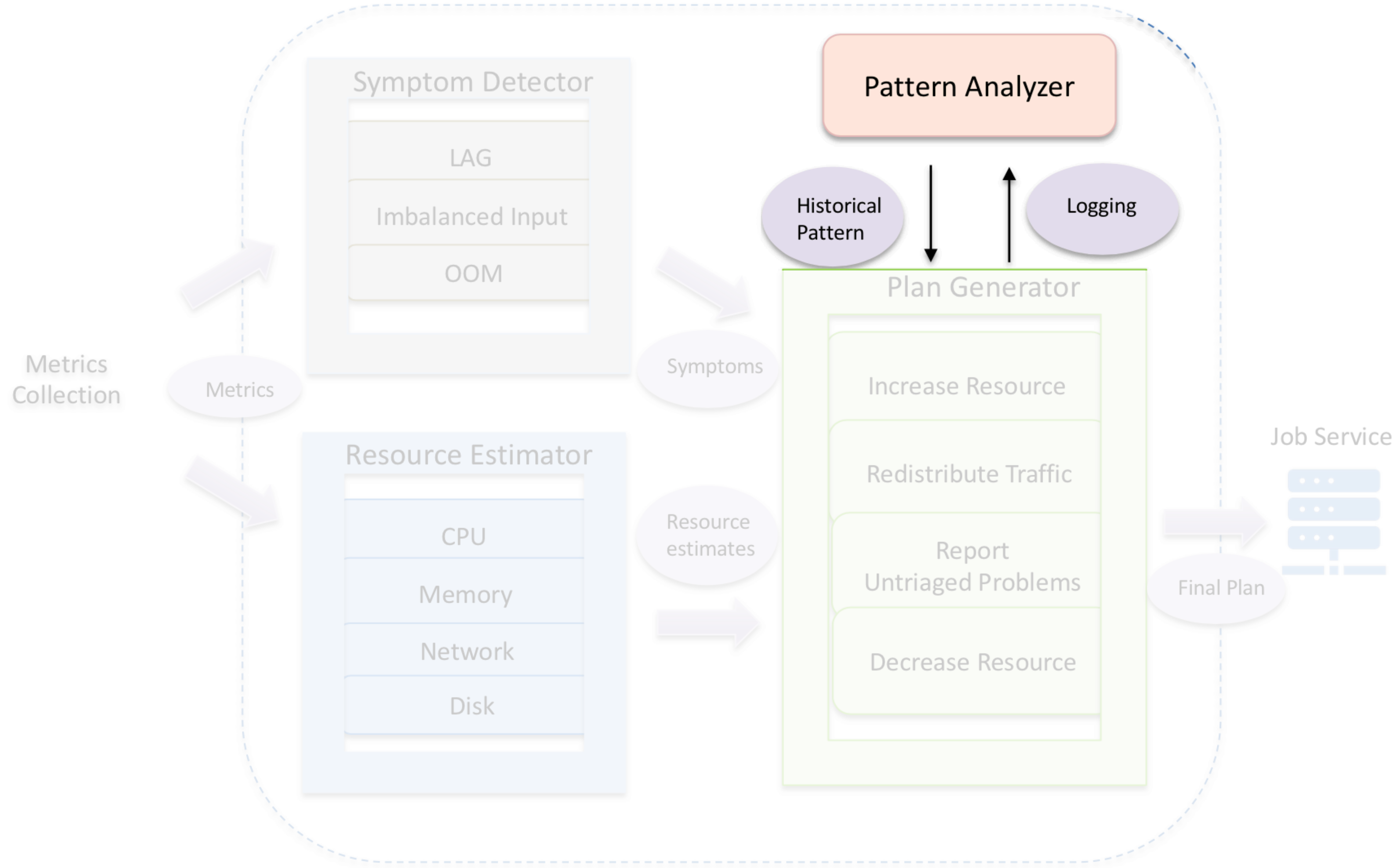
- **Stateless Job**:
  - Ofen CPU intensive;
  - CPU% = (X + B/t) / (P * k * n);

- **Stateful job**:
  - CPU + memory + disk;
  - Aggregation job — the memory size is proportional to the input data kept in memory.
  - Join operator — the memory/disk size is proportional to the join window size, input matching, and input disorder.



More on Paper …

- **Pattern Analyzer**
  - Resource Adjustment Data
  - Historical Workload Patterns
- **Vertical vs Horizontal**
- **Capacity Management**



Production Experience

Scuba Application

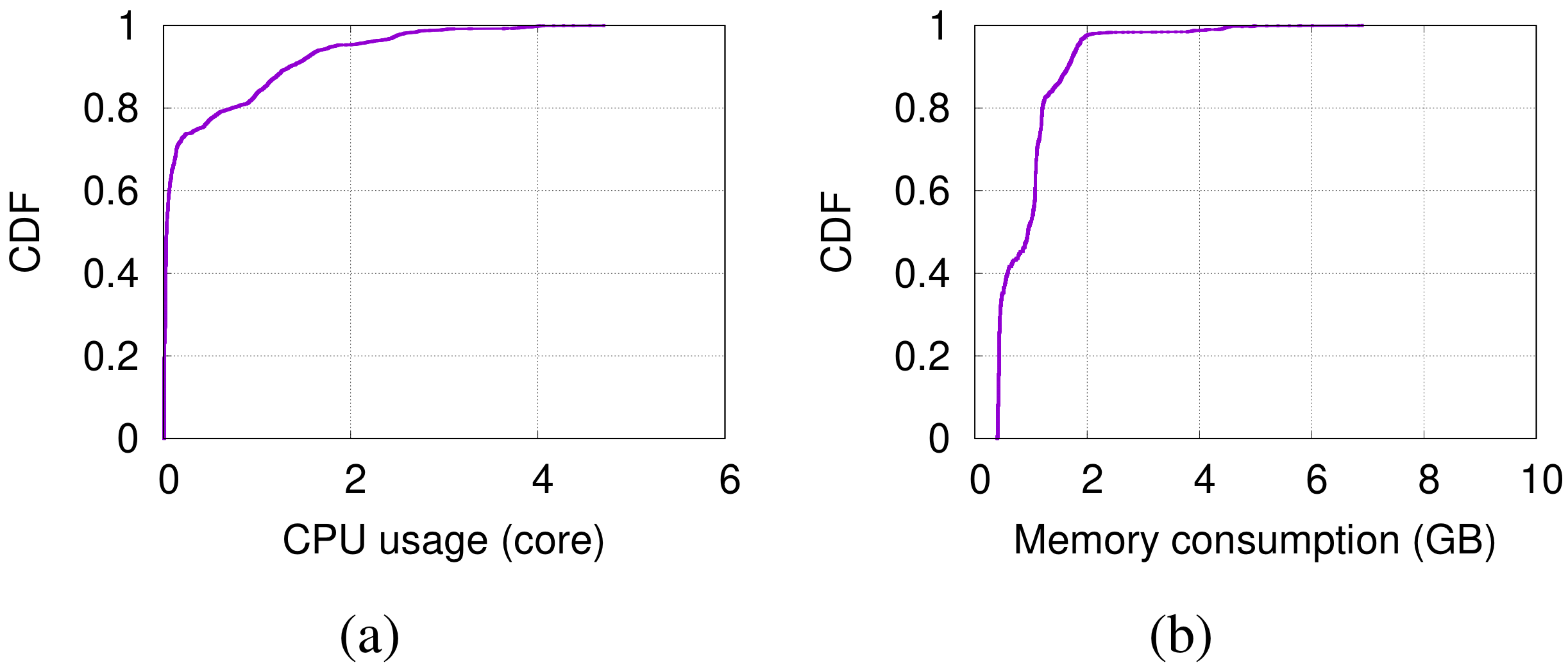
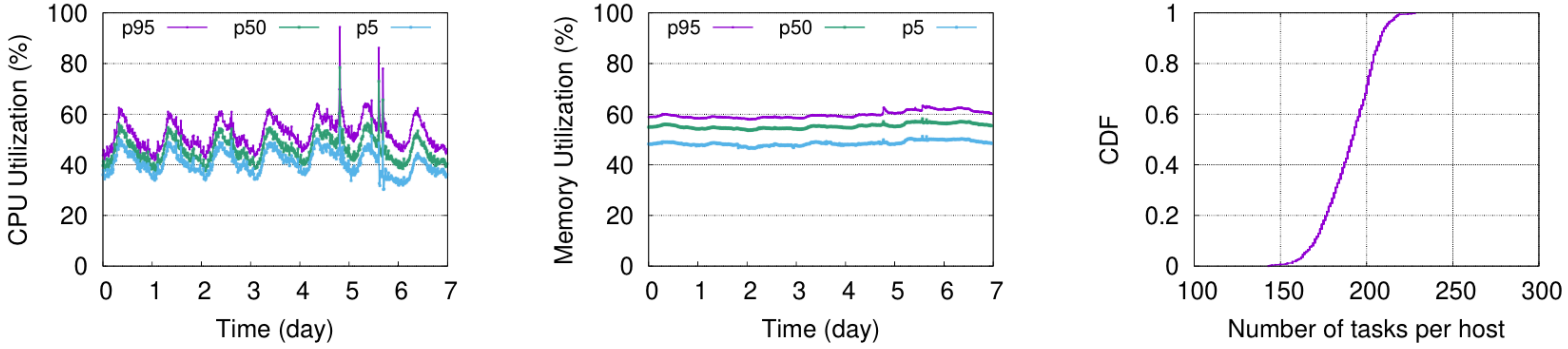


Fig. 5: CPU and memory usage of Scuba Tailer tasks.

Load Balance



(a) (b) (c)

Fig. 6: In one Turbine cluster with > 600 hosts, CPU and memory utilization numbers are very close across hosts. With each host running hundreds of tasks, the reserved per-host headroom can tolerate simultaneous input traffic spike from many tasks.

Workload Change

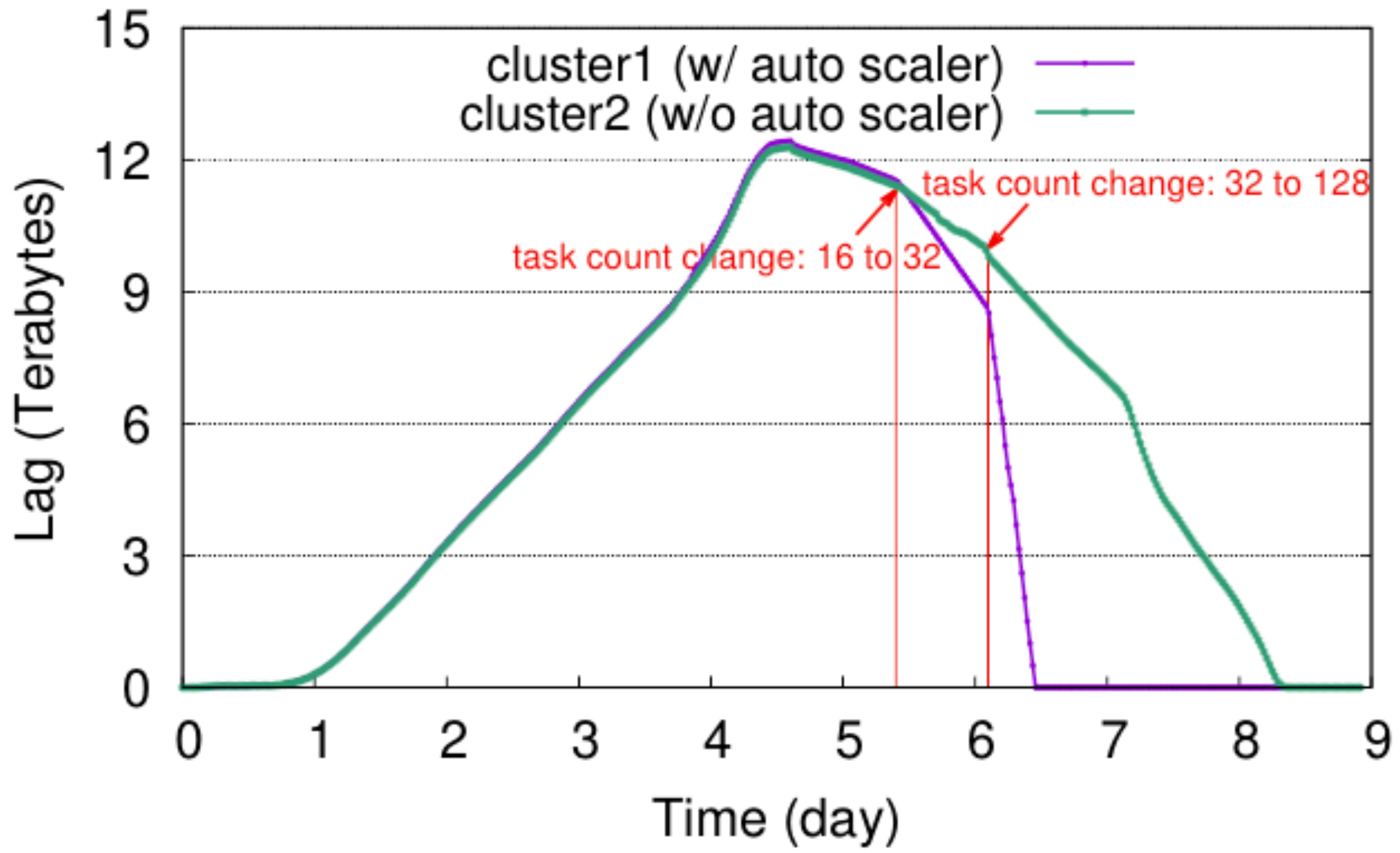


Fig. 8: Turbine's Auto Scaler helped a backlogged Scuba tailer job recover much faster.

Workload Change

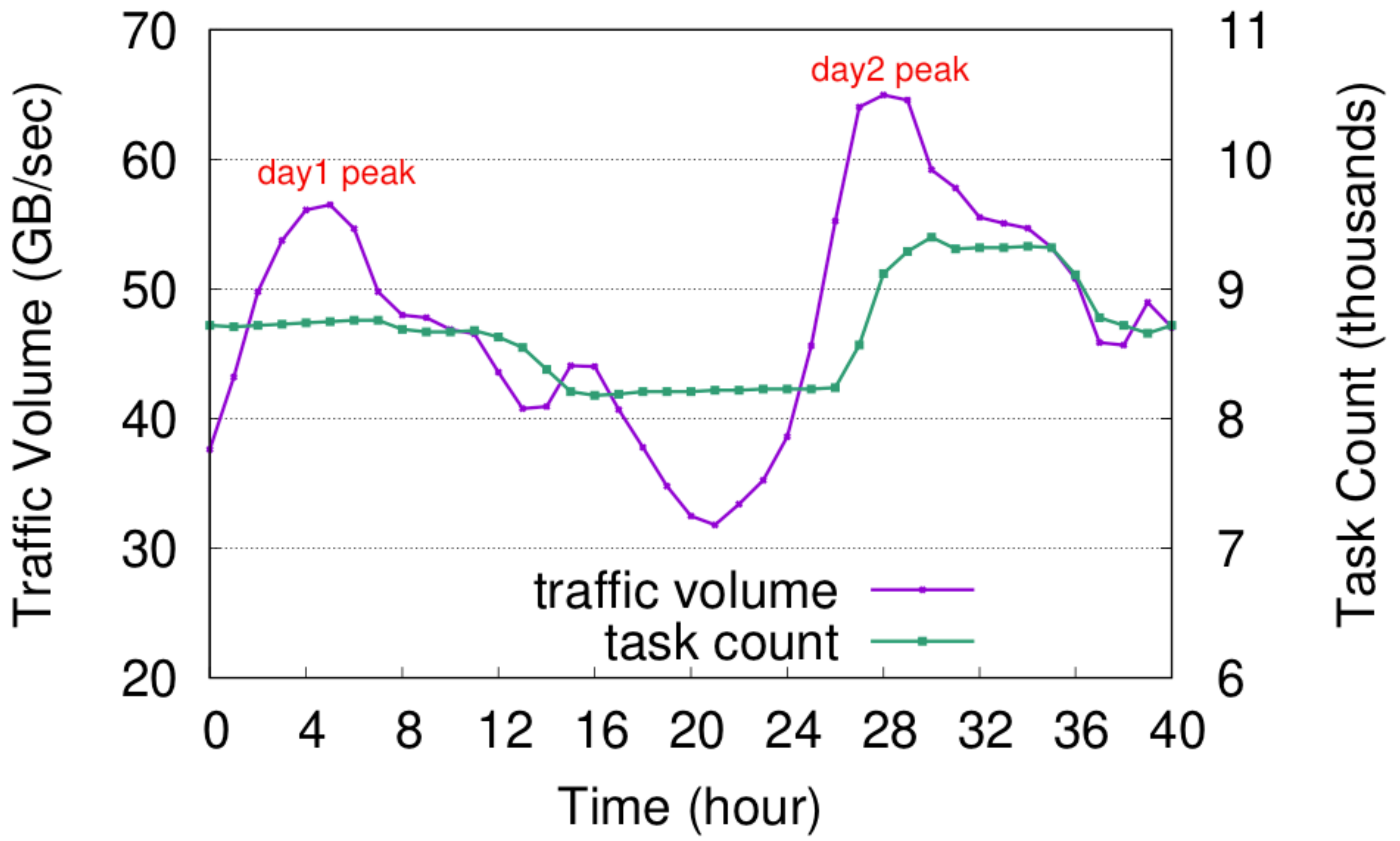


Fig. 9: Turbine's Auto Scaler reacted to Storm by performing horizontal scaling at cluster level.

Thanks