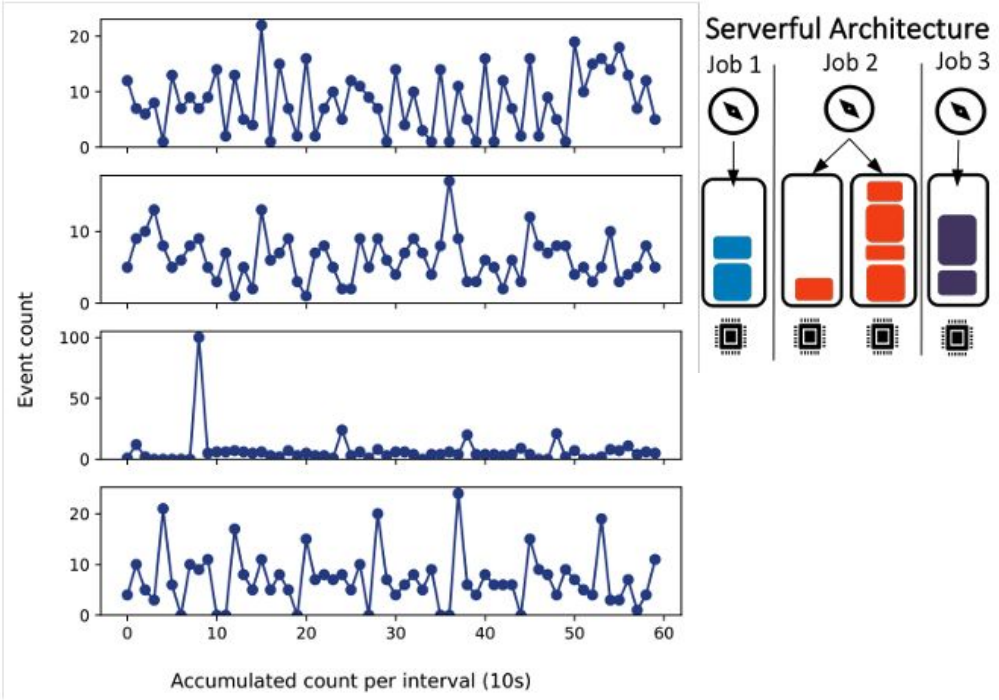
Dirigo: Self-scaling Stateful Actors For Serverless Real-time Data Processing

Background of Distributed Real-time Data Processing

- Unpredictable properties
  - Volume, velocity, arrival patterns
- …which brings challenges to distributed real-time data processing engines:
  - Handling unpredictable properties
  - Satisfying various user-specified performance targets
- Requirement: **Elastic resource provisioning**

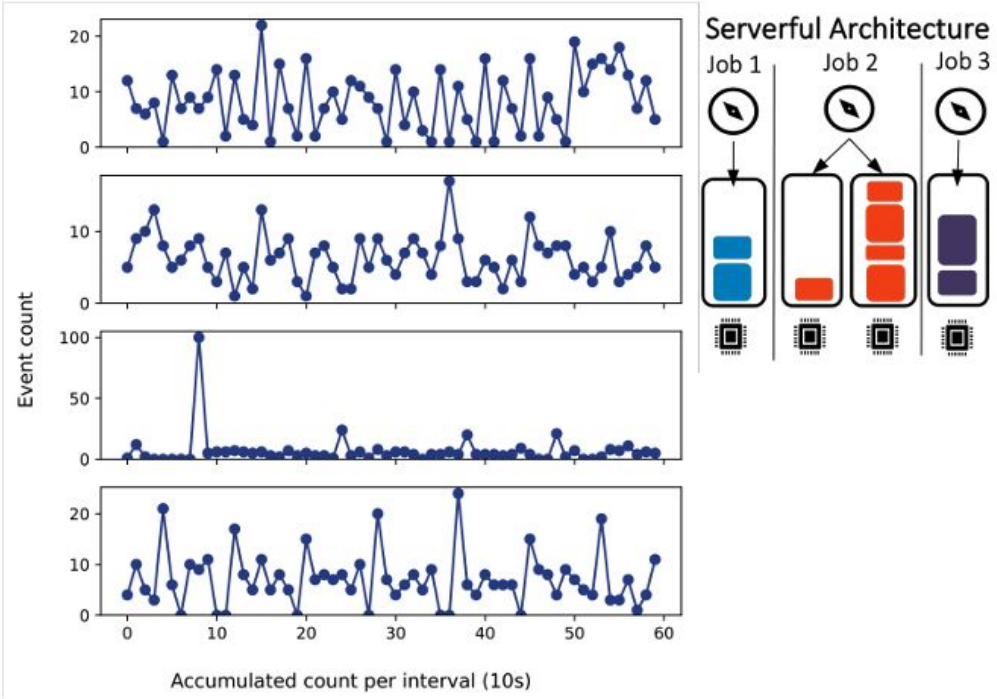
State-of-the-Art Real-time Dataflow Engines

- Serverful environment
  - Fixed number of workers for each application
  - Different applications are isolated
- How they achieve elastic resource provisioning?
  - Reactive, job-level reconfiguration
  - Monitoring processing pipelines
  - Generate new resource execution plan once bottleneck detected



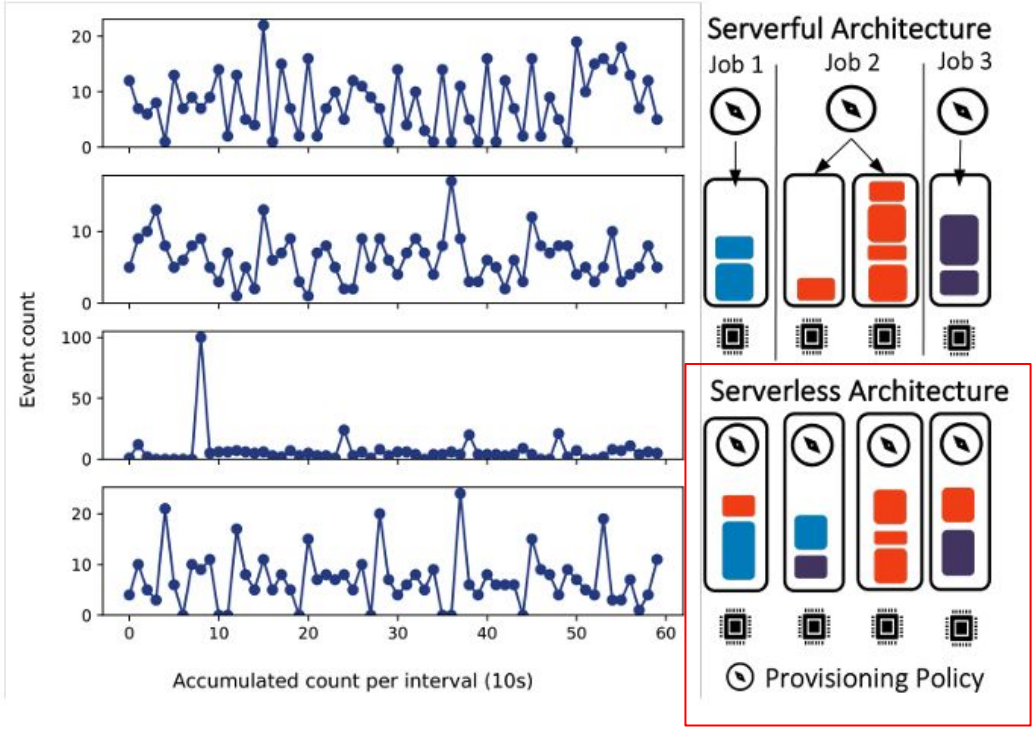
State-of-the-Art Real-time Dataflow Engines

- Problem: Server resource under-utilization
  - Shared-nothing, prevent resources being transfered between jobs efficiently
  - Coarse grained reconfiguration, it can only adopt to long-term, significant workload changes



Solution: Adopting Serverless Architecture

- Idea: Do not assign fixed server for jobs (like there is no server at all)
- Modelling stream operators as serverless functions
- Fine-grained, time-sharing of resources



Adopting Serverless Architecture into DSPS: Challenges

Challenge 1: Achiving fine-grained performance isolation

- Fail to satisfy user intent for latency-critical apps
  - Shared FIFO queue without priority could cause high delay
- Too many latency-critical apps result in performance degradation for other apps
- Requirement: A scheduling strategy for the DSPS to…
  - Dynamically interpret performance targets
  - Translate and practice resource provisioning decisions
  - Must be carried frequently and immediately

Adopting Serverless Architecture into DSPS: Challenges

Key mechanism 1: Data-plane scheduler & scheduling API

- Enable resource provisioning on data-plane
  - Each message's execution path is associated with a series of preset hooks, which are implemented with scheduling policies to be invoked
- Scheduling API for customized scheduling policies
  - Transfer user requirements to real-time scheduling decisions

Adopting Serverless Architecture into DSPS: Challenges

Challenge 2: Lack of support for auto-scaling streaming operators

- Efficient scaling of state access operations:
  - For the (shared) states to stay consistent and reliable
- Ensure ordering of processing input events
  - For the application to produce correct, timely results
- Requirement: DSPS should…
  - Manage state during scaling
  - Enable user to explicitly specify processing orders, state partition & combination strategies
  - Even with non-determined ordering, ensure state consistency during auto scaling

Adopting Serverless Architecture into DSPS: Challenges

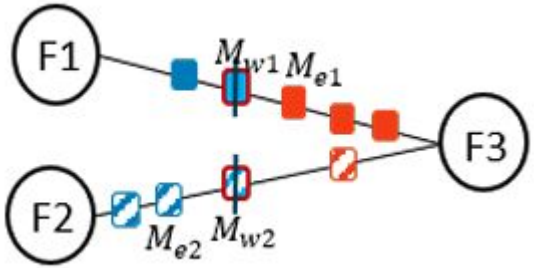
Key mechanism 2: Dual-mode virtual actor model

- Switches between parallel and sequential execution modes
- Sequential: create critical region for a streaming operator to execute messages in a single-thread fashion
- Parallel: parallelize a streaming operator outside of the critical region
- (Details to be explained later)

Dirigo: System Overview

Dirigo adopts DAG based, message-triggered semantics

- Each application is mapped to a DAG
  - Vertex: function
  - Edge: message flow
- Two types of messages
  - User message: generated by user functions
  - Control message: generated by system for scaling control

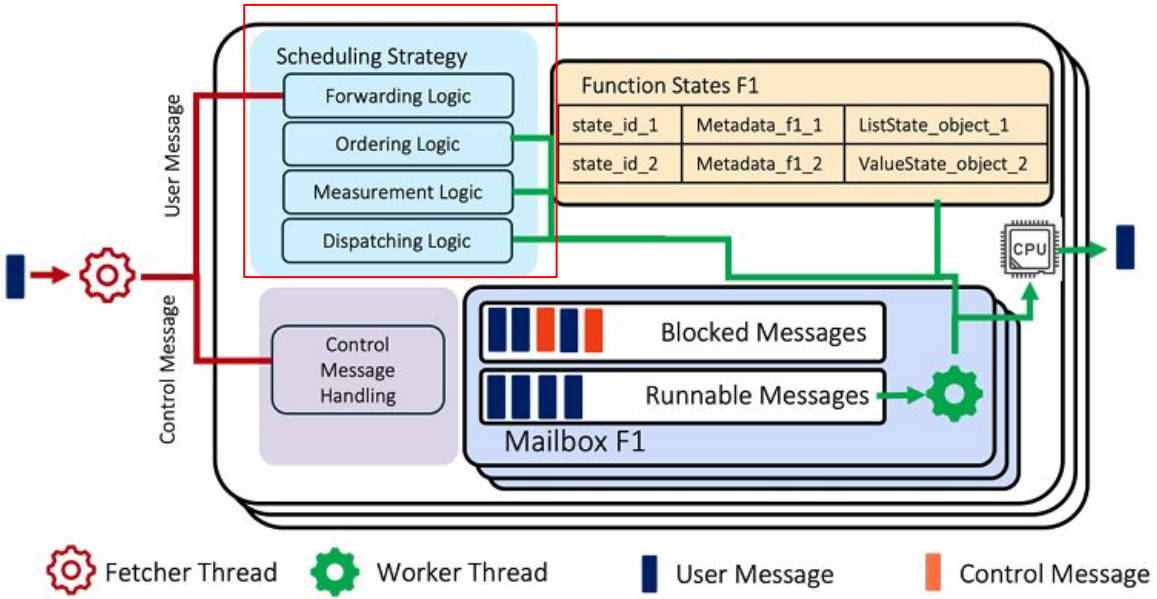


Key Mechanism 1: Data-plane Scheduling

- Enable resource provisioning on **per-message** level
- Each message's execution path (green line) is associated with preset **hooks**
- Hook exposes API for user to **customizes scheduling policies**

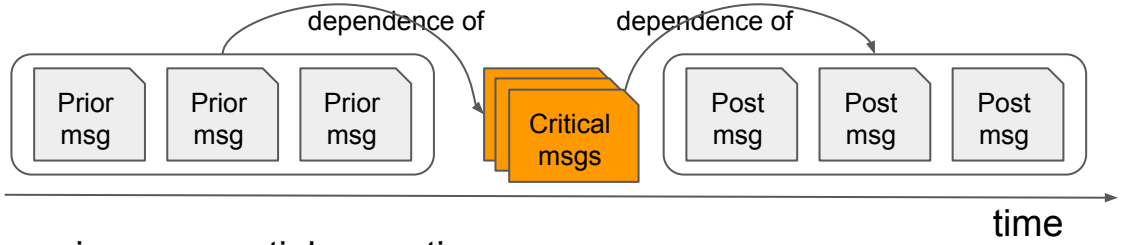
Benefits

- React to load changes quickly without acquiring control plane (e.g., external scheduler)
- Implicit scale up/down



Key Mechanism 2: Dual-mode Virtual Actor

A few important definitions

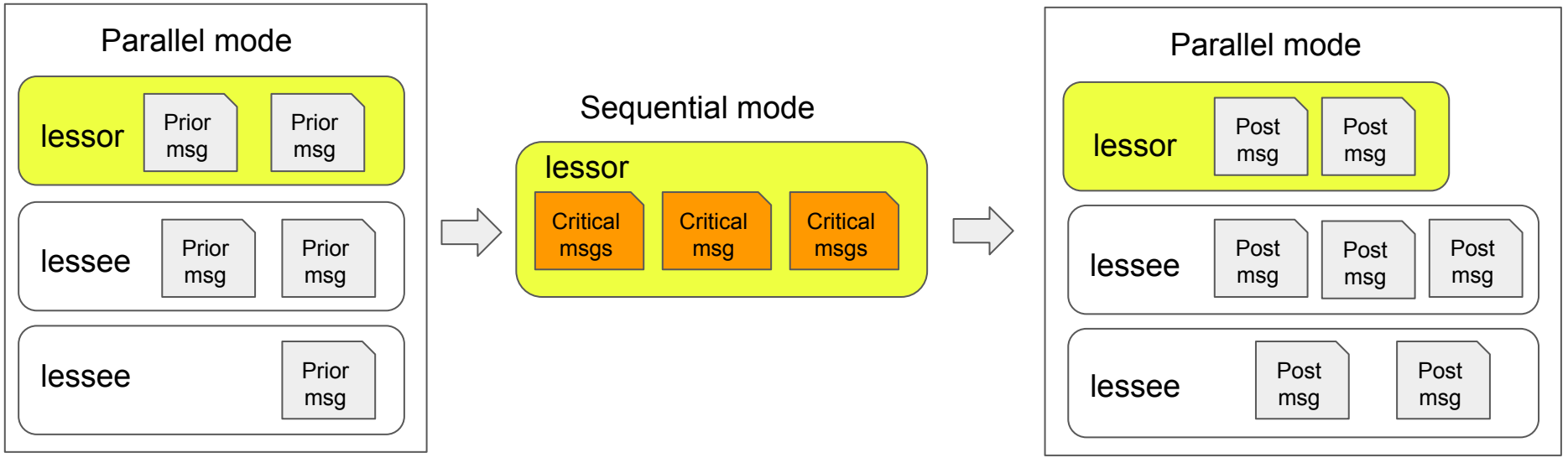


- Critical message
  - Special type of message that requires sequential execution
  - Must be executed after its Set{Prior msg} it depends on, and before Set{Post msg} depends on it.
- Barrier
  - A set of critical messages that must be processed together
- Dual-mode execution
  - As discussed earlier

Key Mechanism 2: Dual-mode Virtual Actor

Each streaming operator is one virtual actor, consists of:

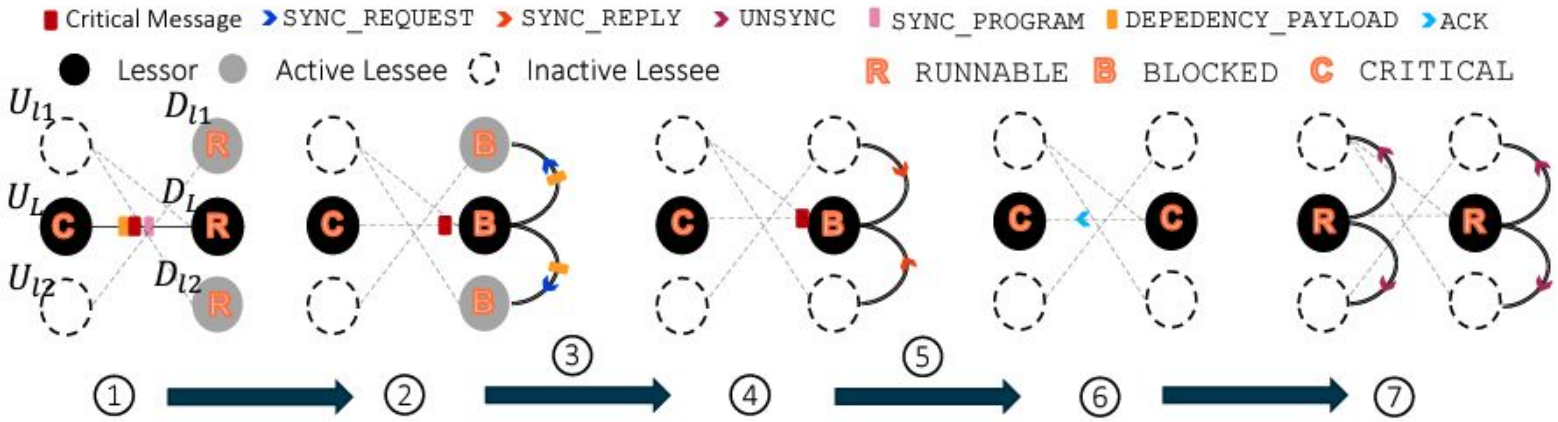
- One **lessor** thread (出租): default worker
- 0-N **lessee** threads (承租): shadow worker, created/destroyed during scaling up/down



Further Illustration on Dual-model Virtual Actor

Each thread switches among three states:

- Runnable: normal *parallel* execution
- Blocked: stop execution, consolidate partial states from *lessees* to *lessor*
- Critical: *sequential* execution, lessor thread executes *critical messages*



Experiments

- Implemented atop Flink StateFun, utilizing Flink as underlying message processor
- User-satisfaction Rate (latency-intensive apps):
  - Compared with FIFO without auto-scaling strategy
  - Achieves equal or up to 12% increase of user-satisfaction rate with 30% lower resource utilization
- Scales well as number of parallel instances and state sizes
- (Details refer to paper)