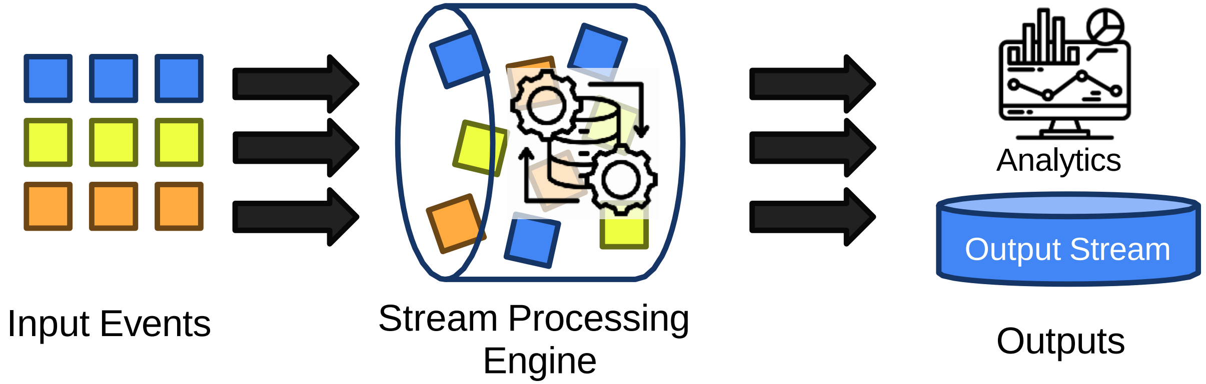
ATC'23_Sponge: Fast Reactive Scaling for Stream Processing with Serverless Frameworks

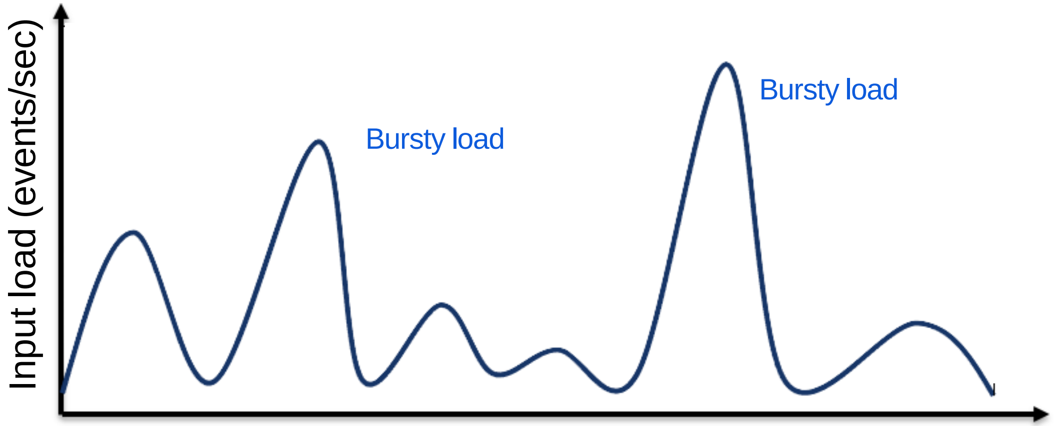
2023/08/23

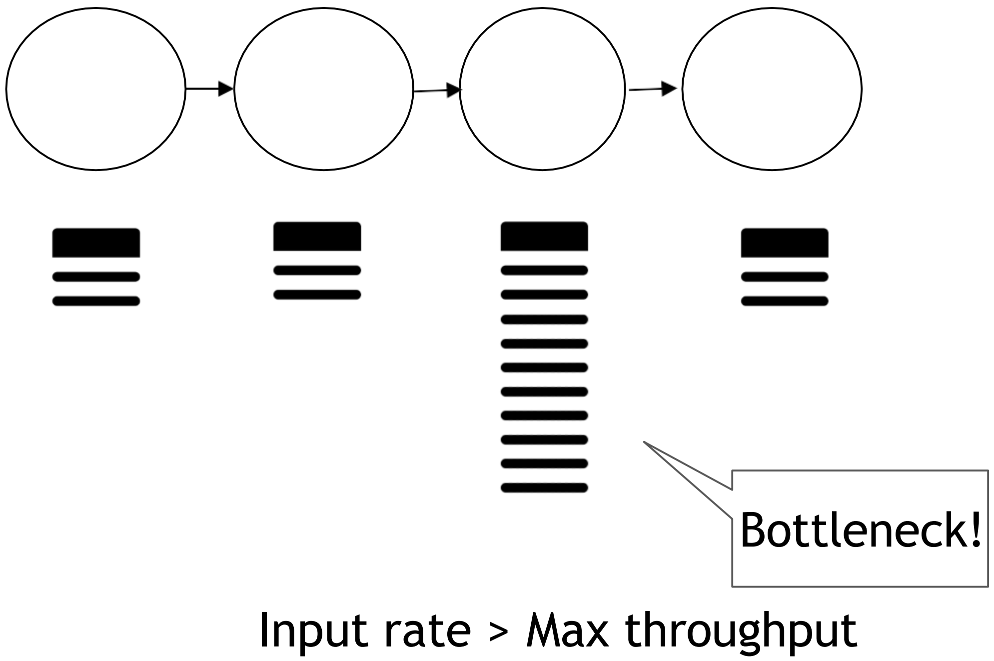
Background: Stream Processing

Stream Processing

- Query -> DAG
- Bursty Load -> Bottleneck







Background: On-Demand Resource Provisioning

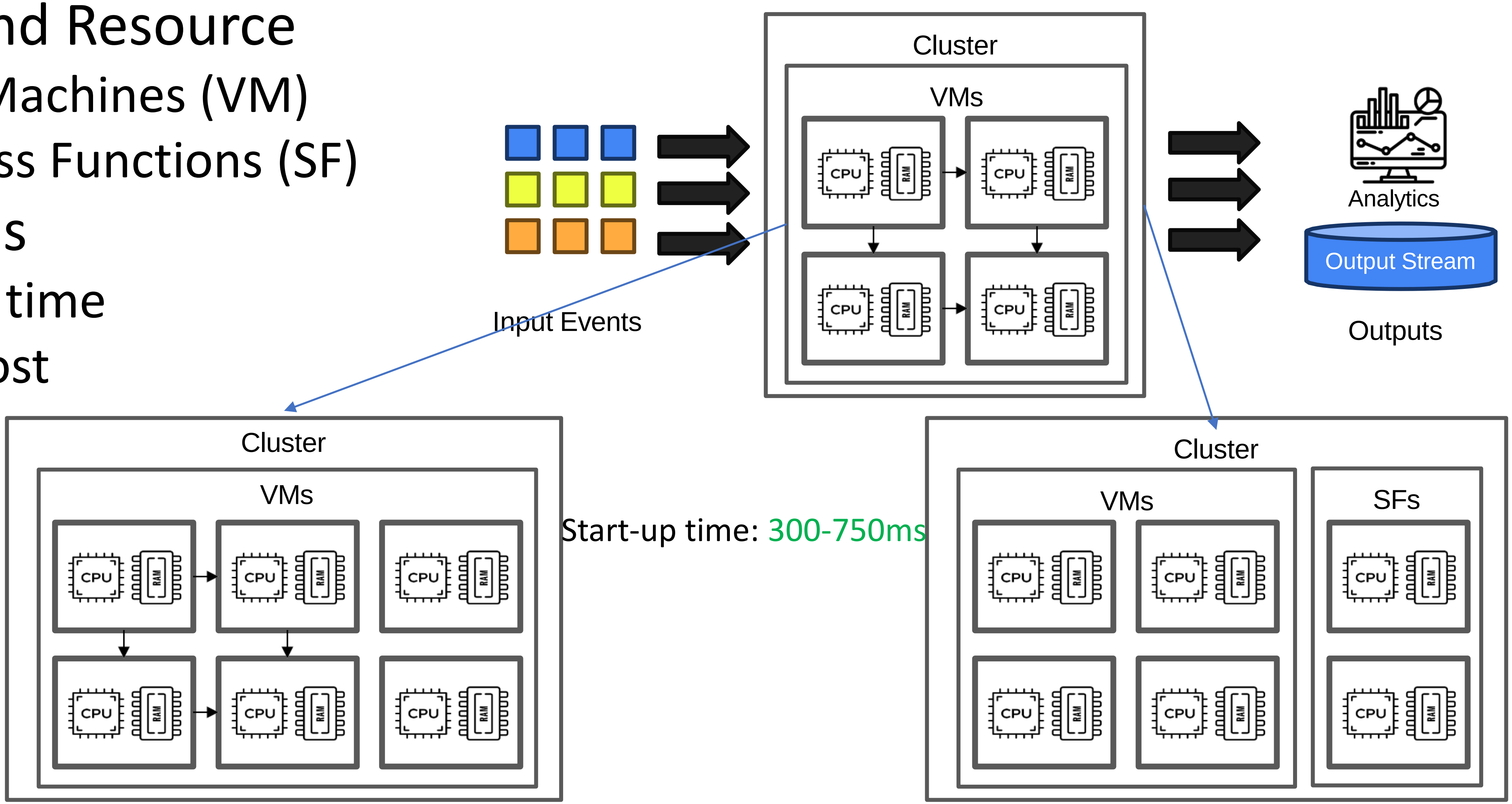
On-Demand Resource

- Virtual Machines (VM)
- Serverless Functions (SF)

Characteris

- Start-up time
- Usage cost

Start-up time: 25s+



Usage cost： VM < SF

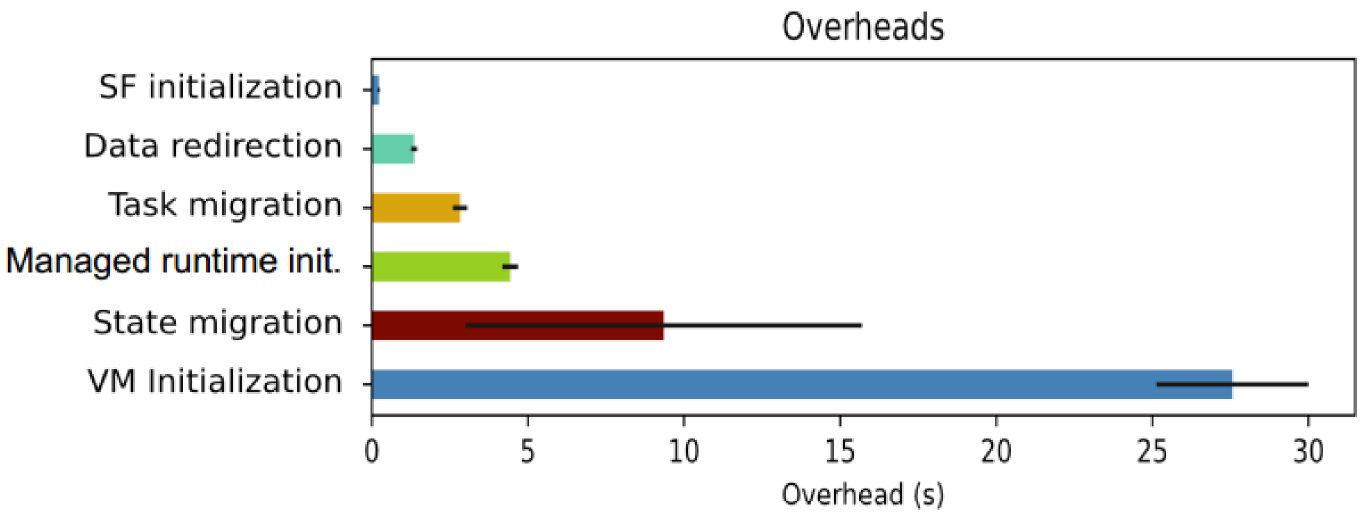
Background

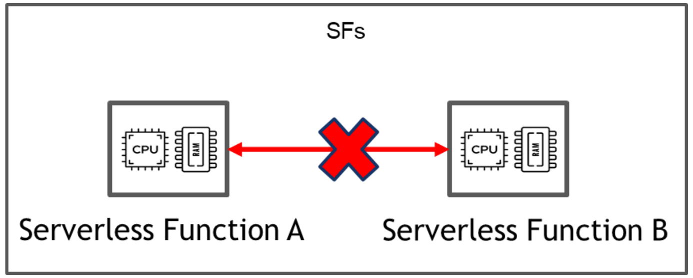
Goal

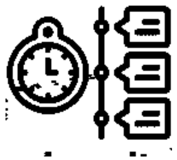
- Quickly detect bursty loads and reduce migration state overhead.

Challenges

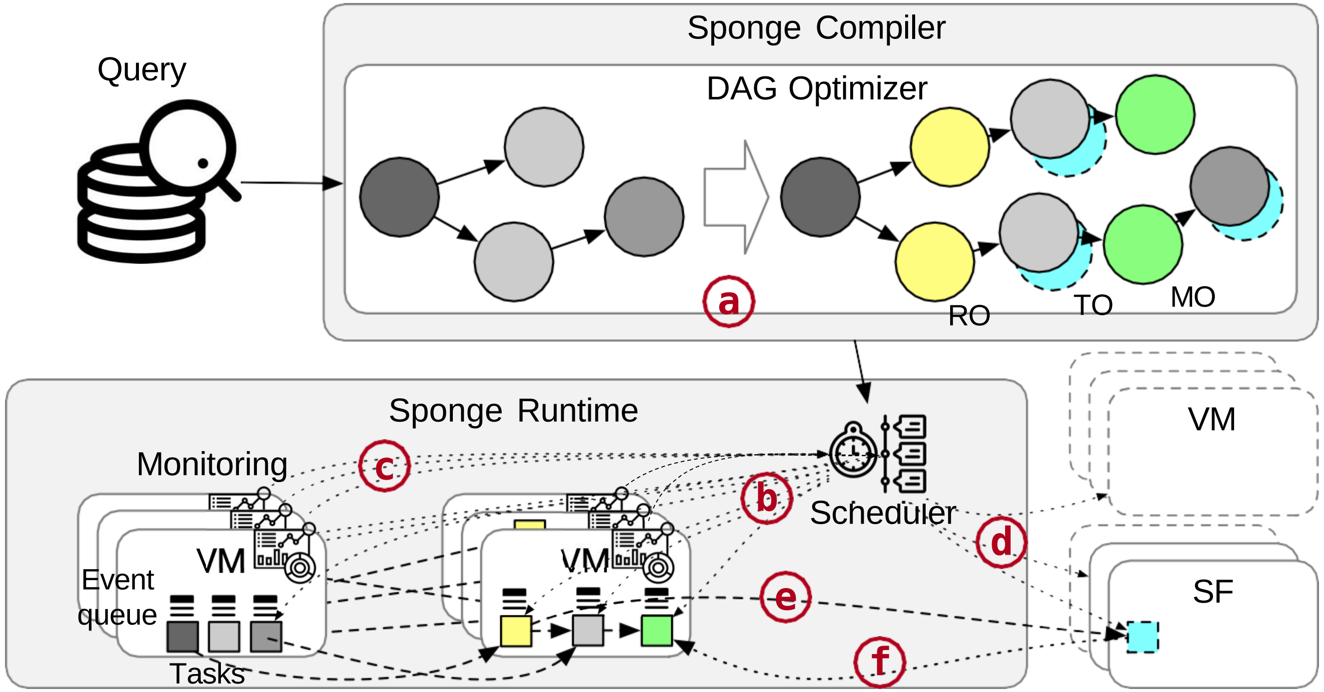
- Migration with large operator states.
- Indirect data communication between SF instances.
- Quick decision making and scaling.







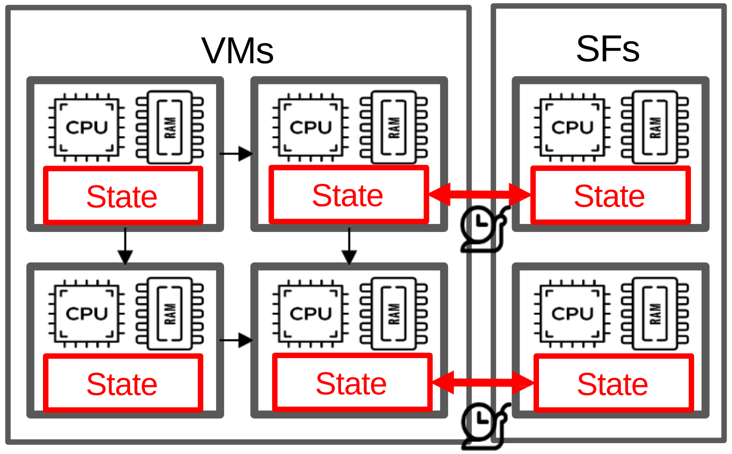
Main Idea

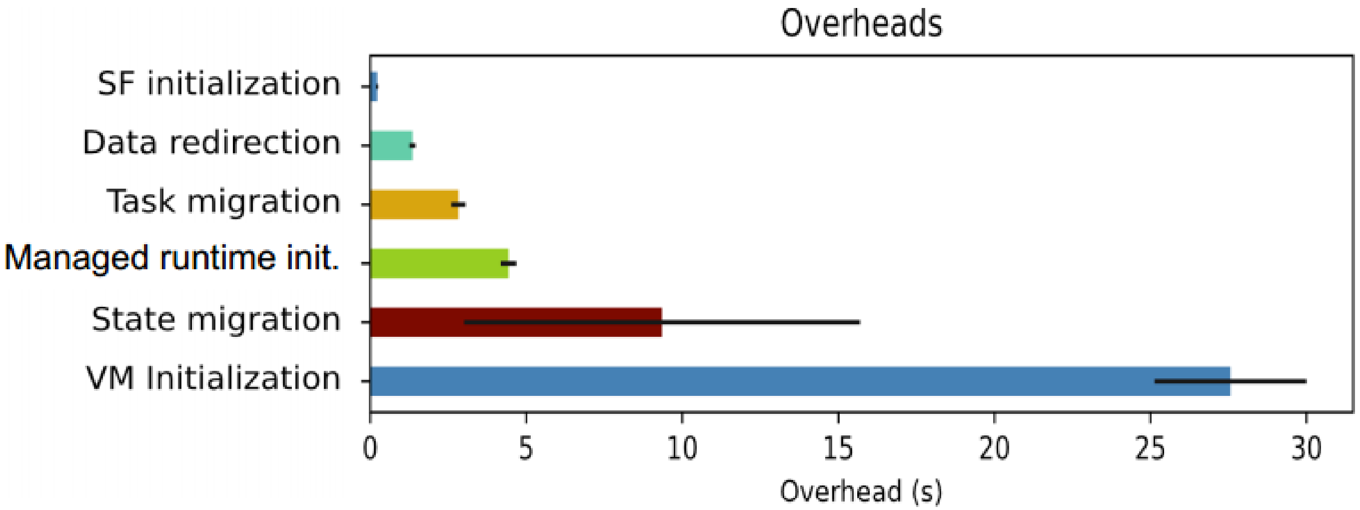


- Redirect-and-merge
  - Compile-time Graph Rewriting Algorithm
  - Reducing Cold Start Latency
  - Watermark message
- Fast reactive scaling
  - Dynamic Offloading Policy

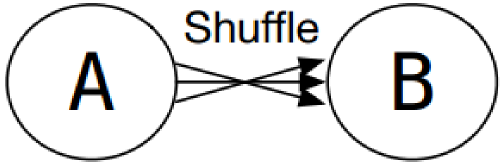
Redirect-and-merge

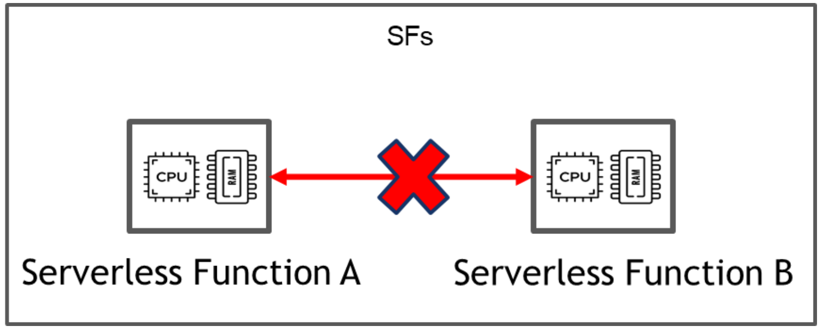
- C1. Migration with large operator states.



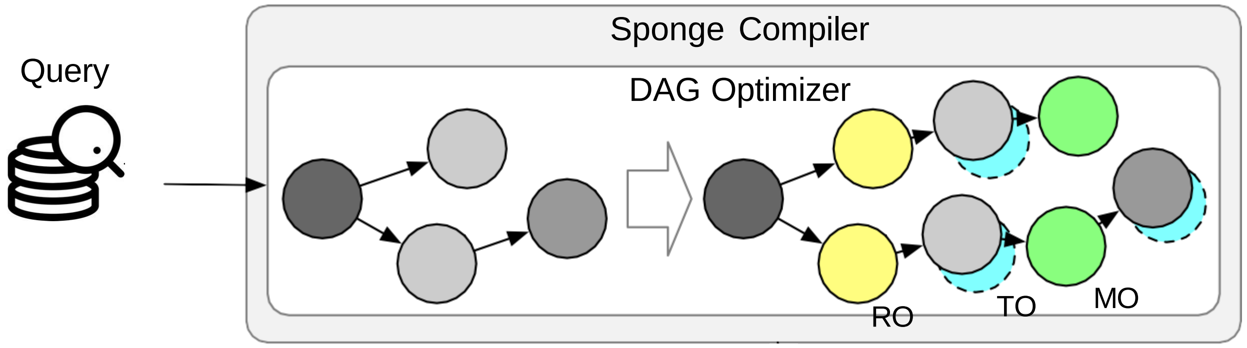


- C2. Indirect data communication between SF instances.





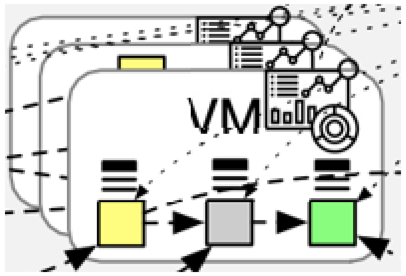
Design1: Compile-time Graph Rewriting Algorithm

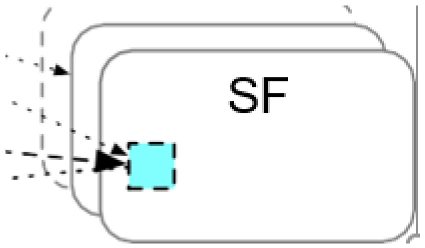


1. Router operators (ROs) enable redirection of input events to specific instances

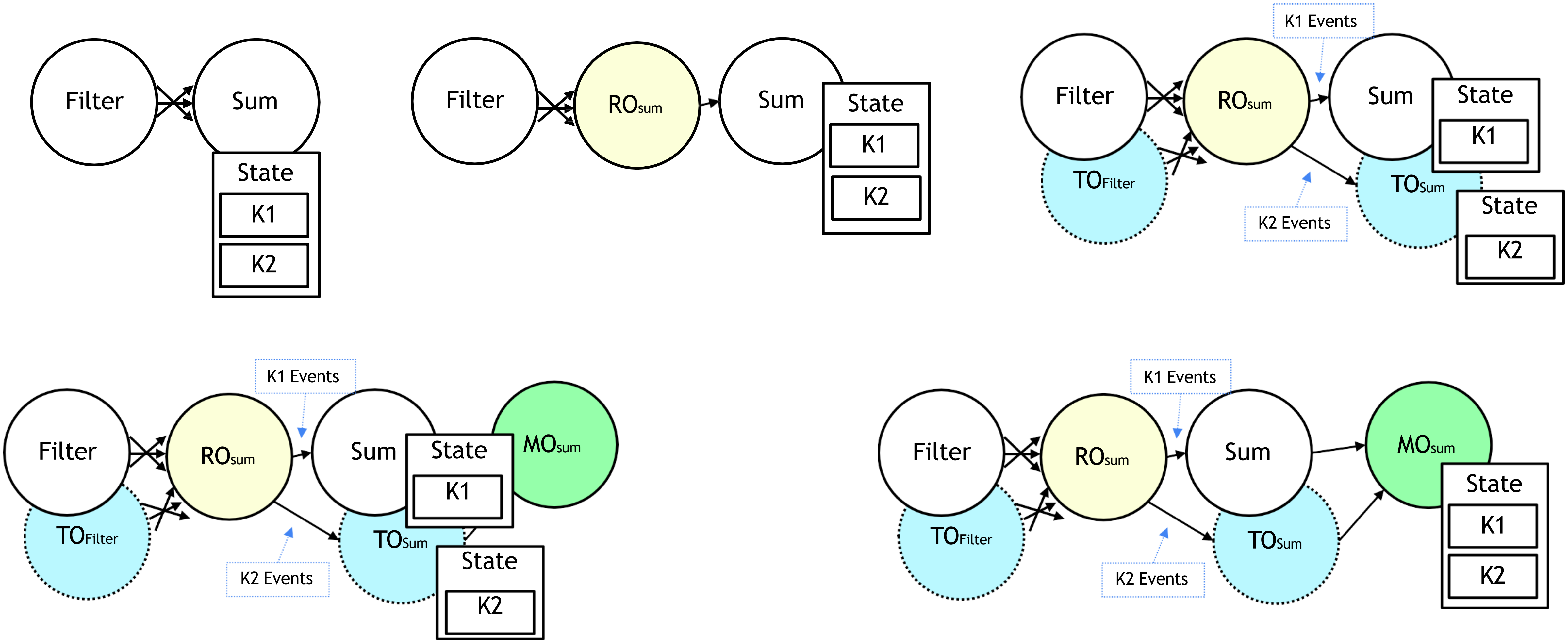
2. Transient operators (TOs) enable execution of cloned operators on SFs

3. Merge operators (MOs) enable merges on partial states



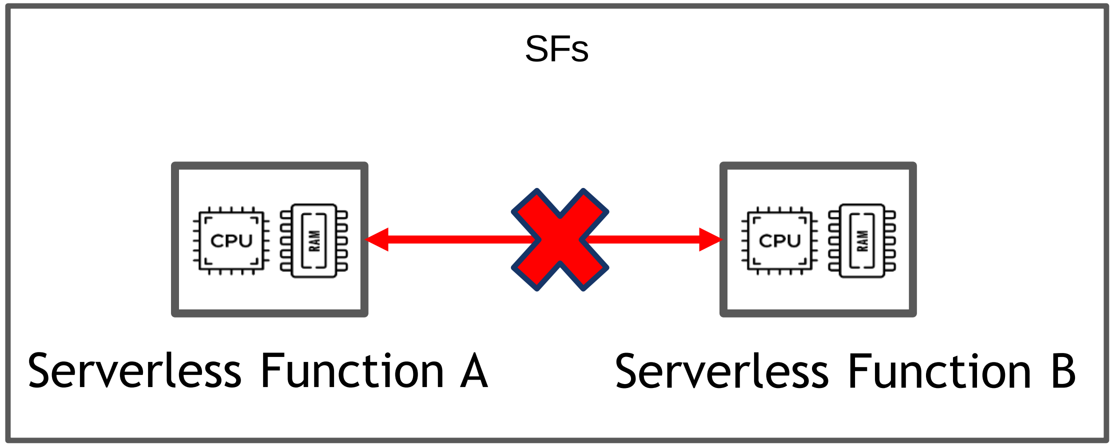


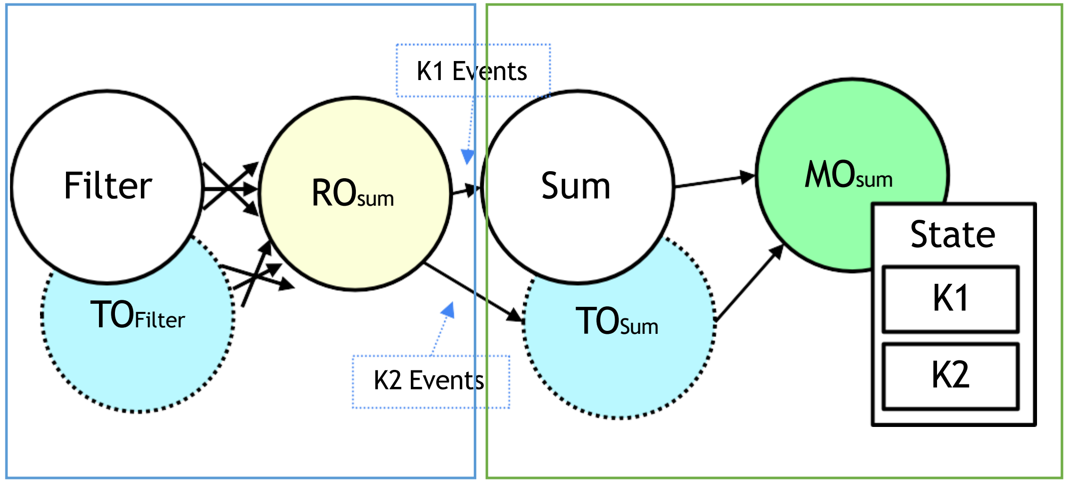
Design1: Compile-time Graph Rewriting Algorithm



Design1: Compile-time Graph Rewriting Algorithm

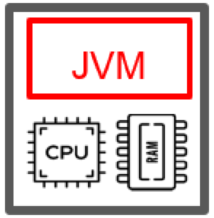
- C2. Indirect data communication between SF instances.
  - SF <-> origin VM
  - VM <-> VM





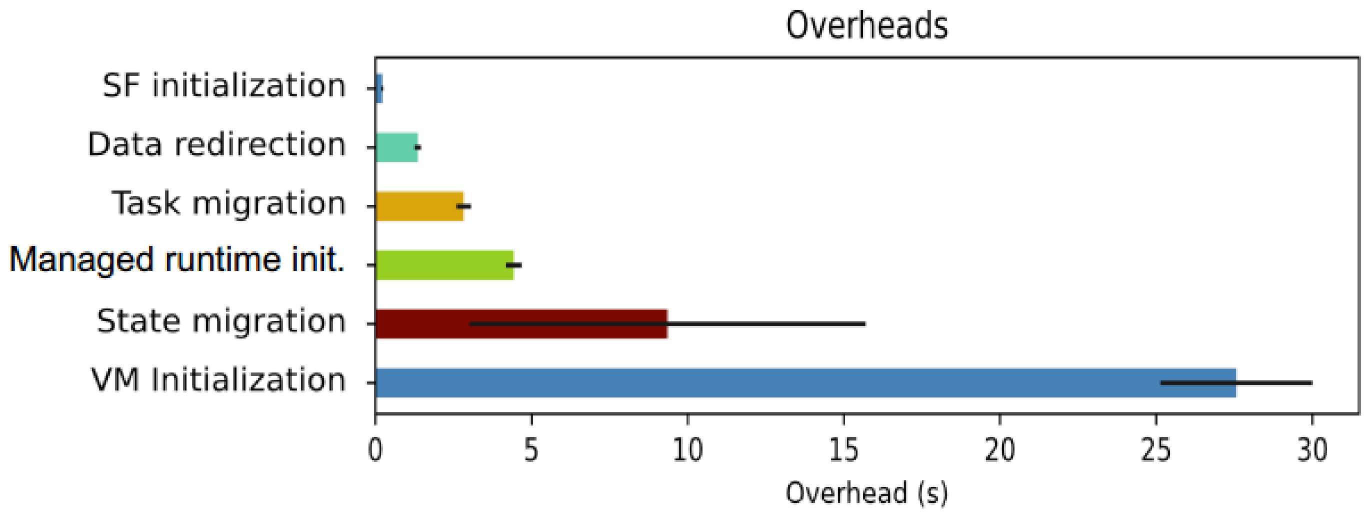
Design2: Reducing Cold Start Latency

Timely gain access to SFs
- Warm-up SF workers
- Cache snapshots of SFs



Serverless Function

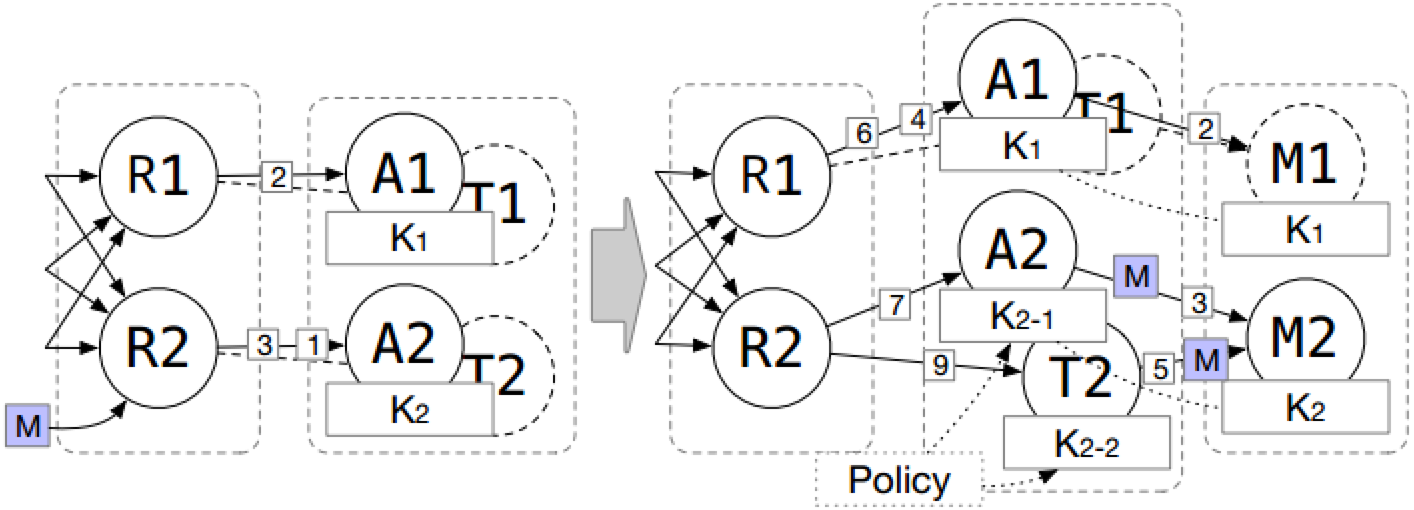
Managed runtimes (e.g., JVM) incur launch overheads (~4 seconds)



Design3: Watermark message

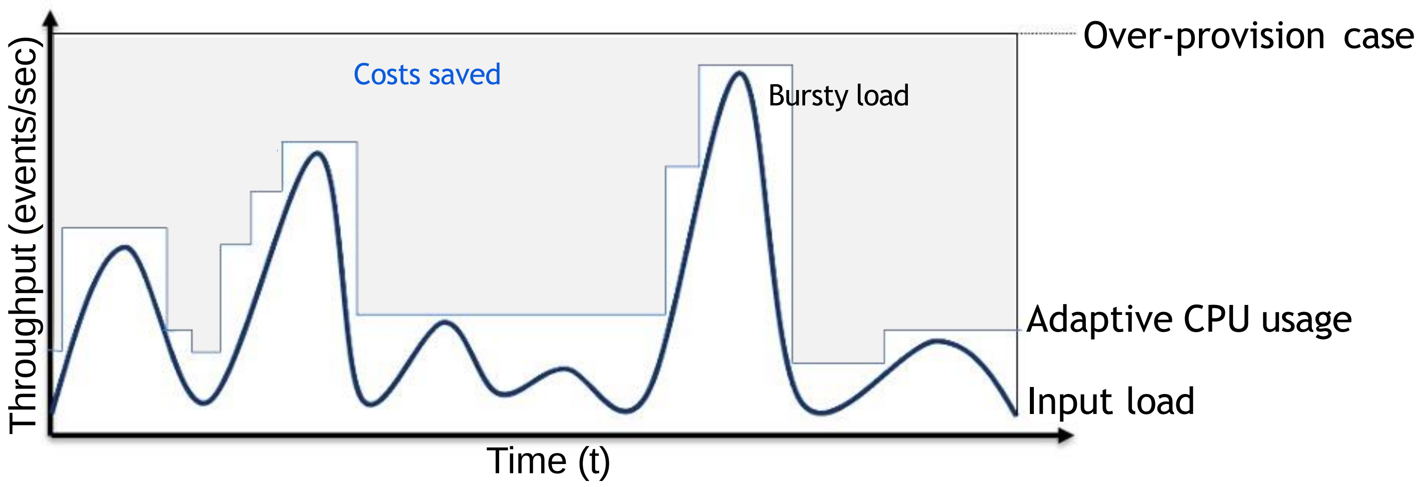
Correctness

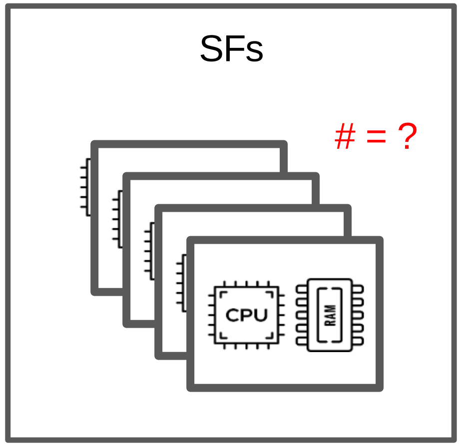
- Watermark as control message
- All events are processed in the same environment



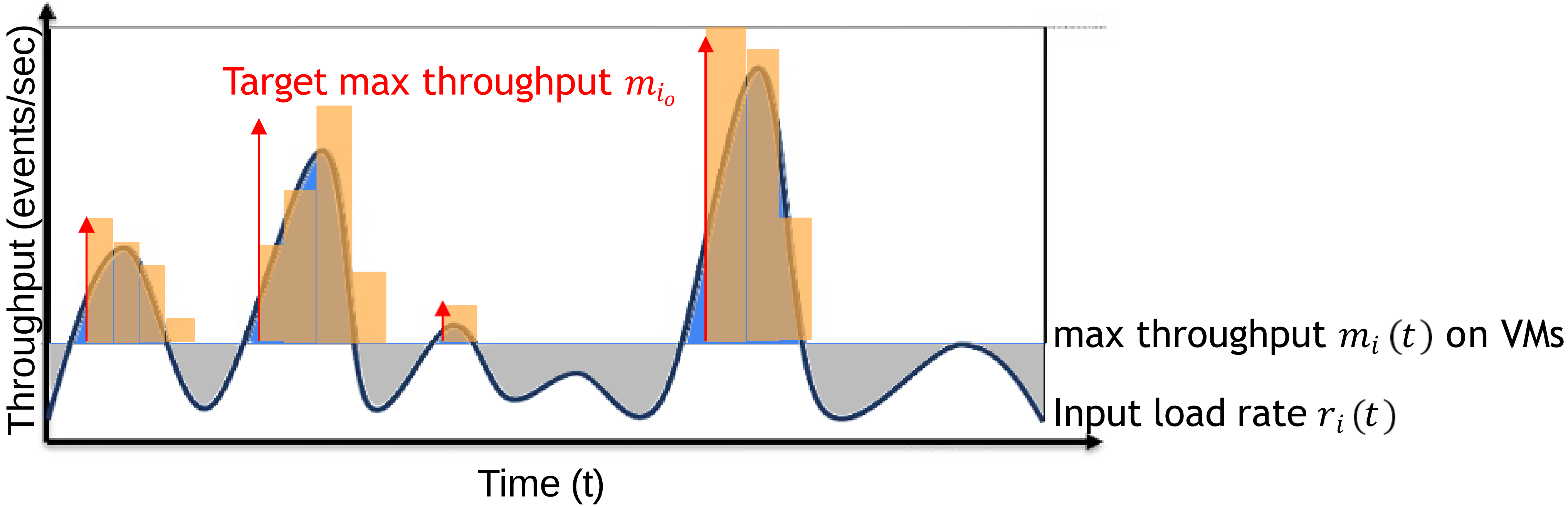
Fast reactive scaling

- C3. Quick decision making and scaling.
- describe when Sponge triggers offloading, how many SF instances it uses



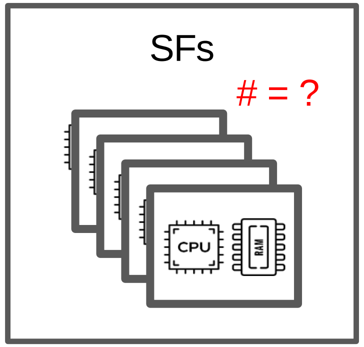


Design4: Dynamic Offloading Policy



$$\textit{Data piled up in the event queue} \leq \textit{Data to process within our target deadline}$$

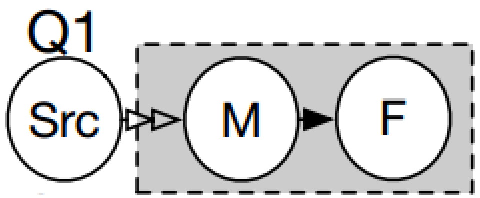
$$(\textit{Existing throughput} * \textit{time} \leq \textit{Target throughput} * \textit{time})$$

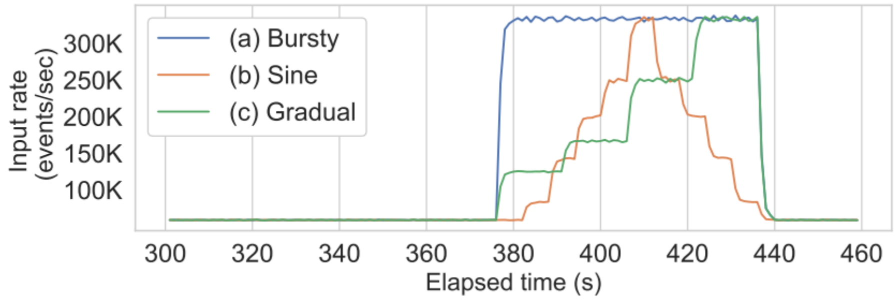


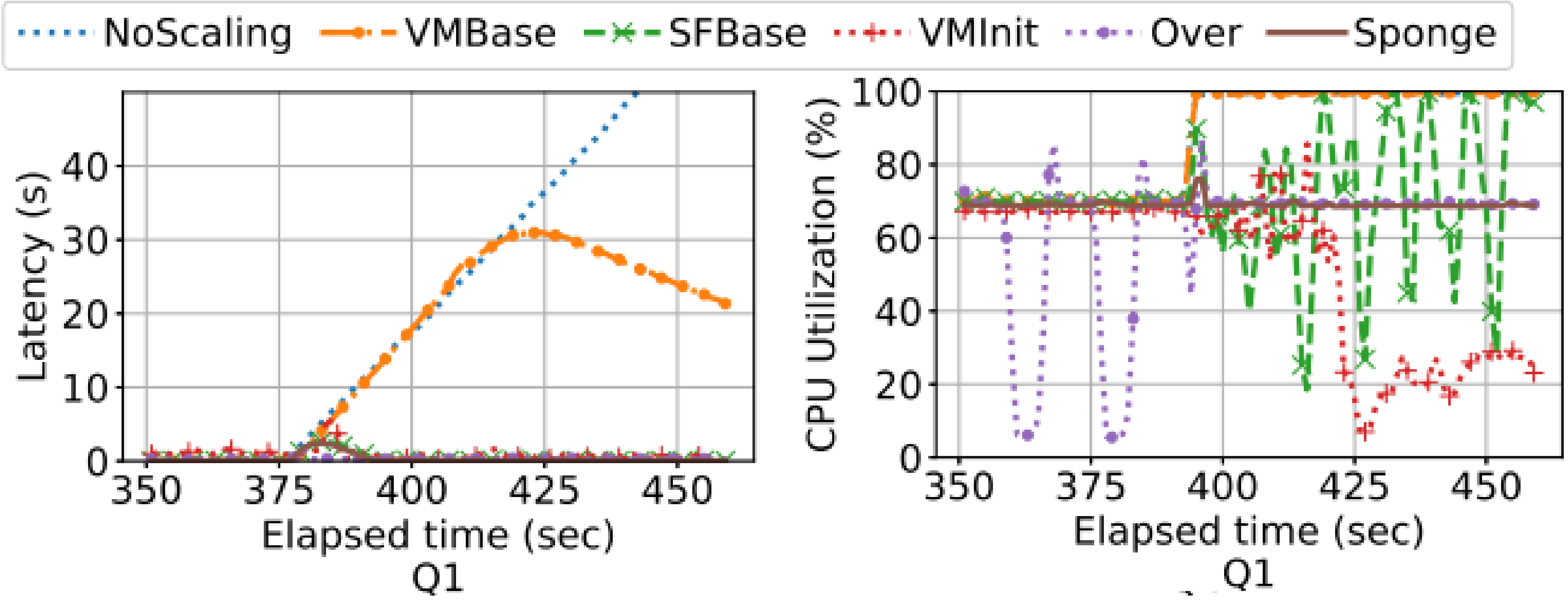
$$\textit{required SF cores} = \left\lceil \frac{\textit{Target additional throughput}}{70\% * \textit{Approx. throughput per SF core}} \right\rceil$$

Evaluation: Latency and CPU Utilizations

The 99th-percentile tail latency and CPU utilization

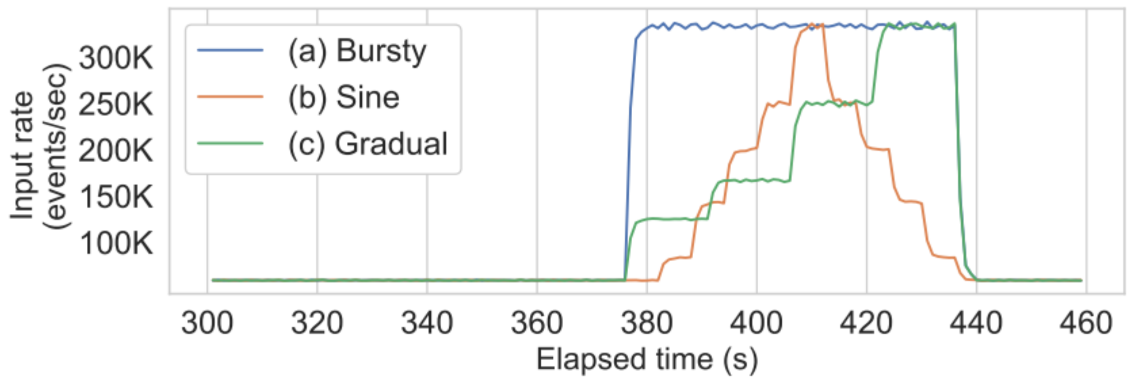


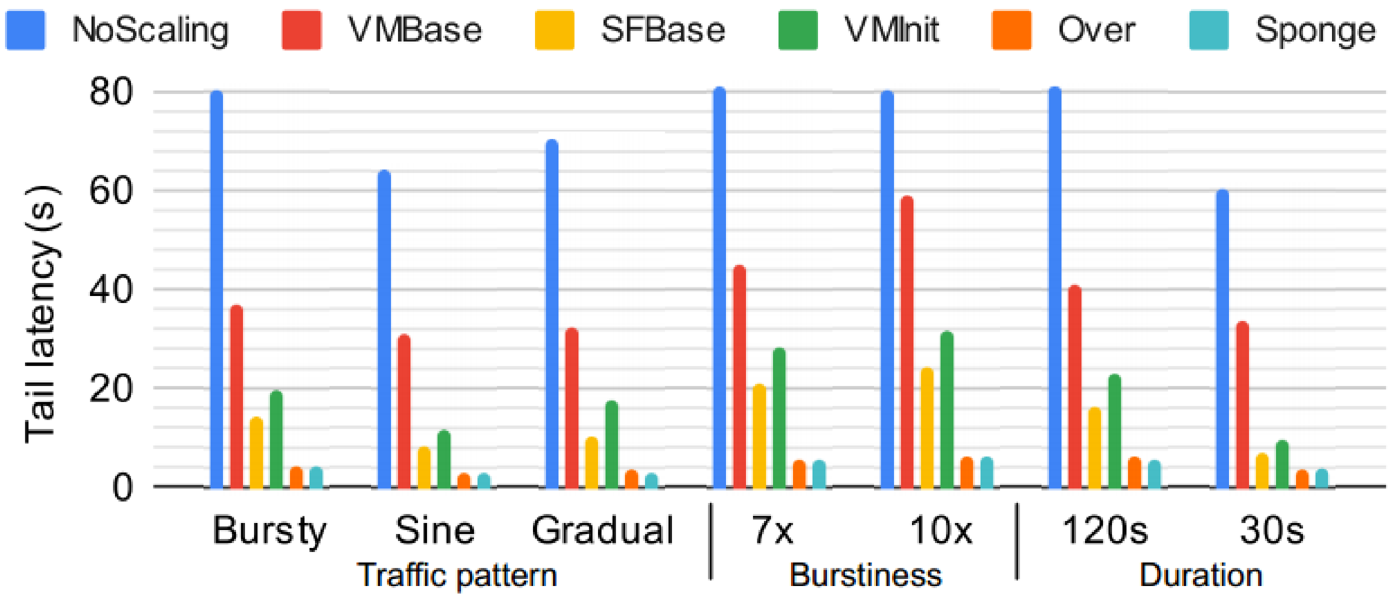




Evaluation: Latency and CPU Utilizations

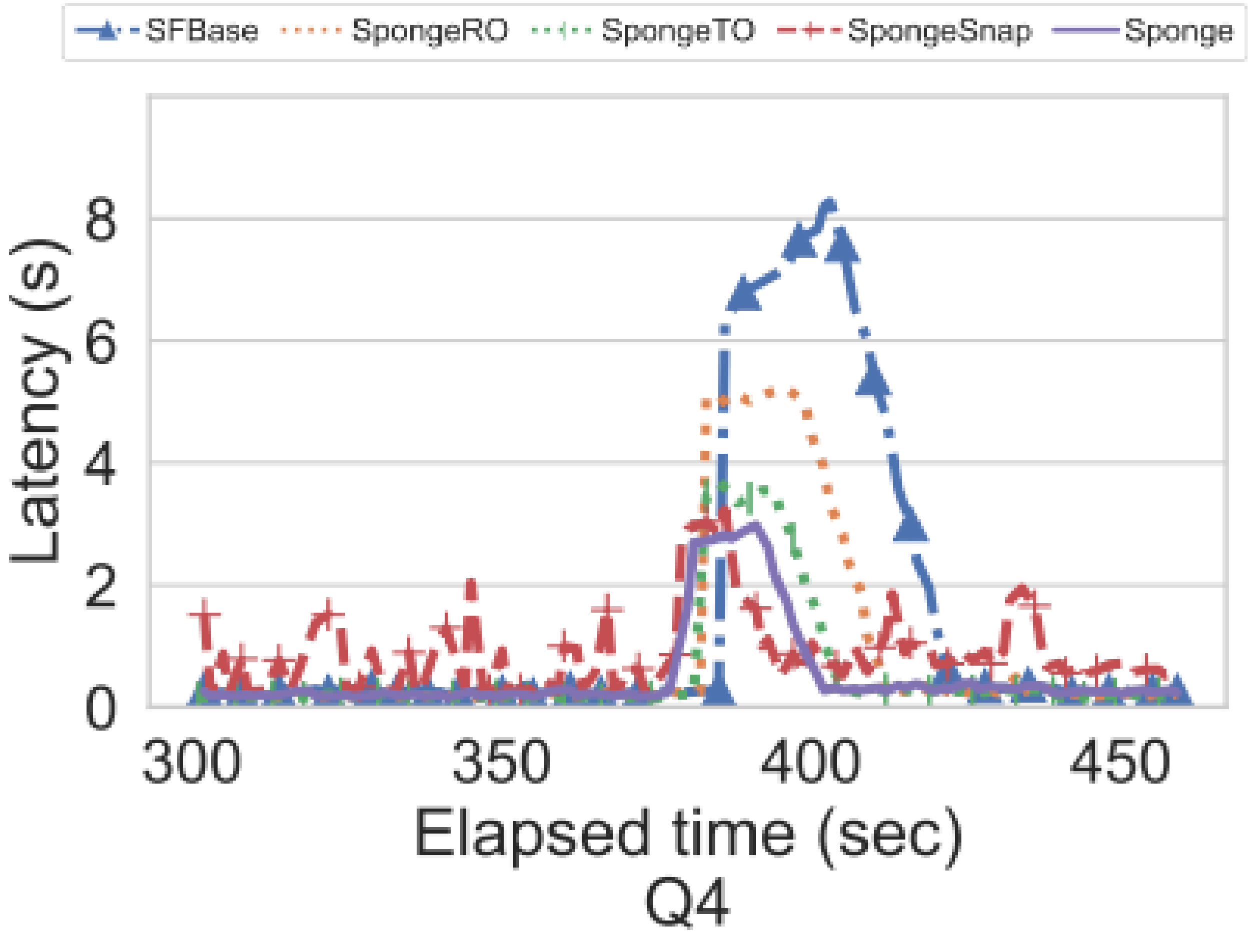
Input patterns





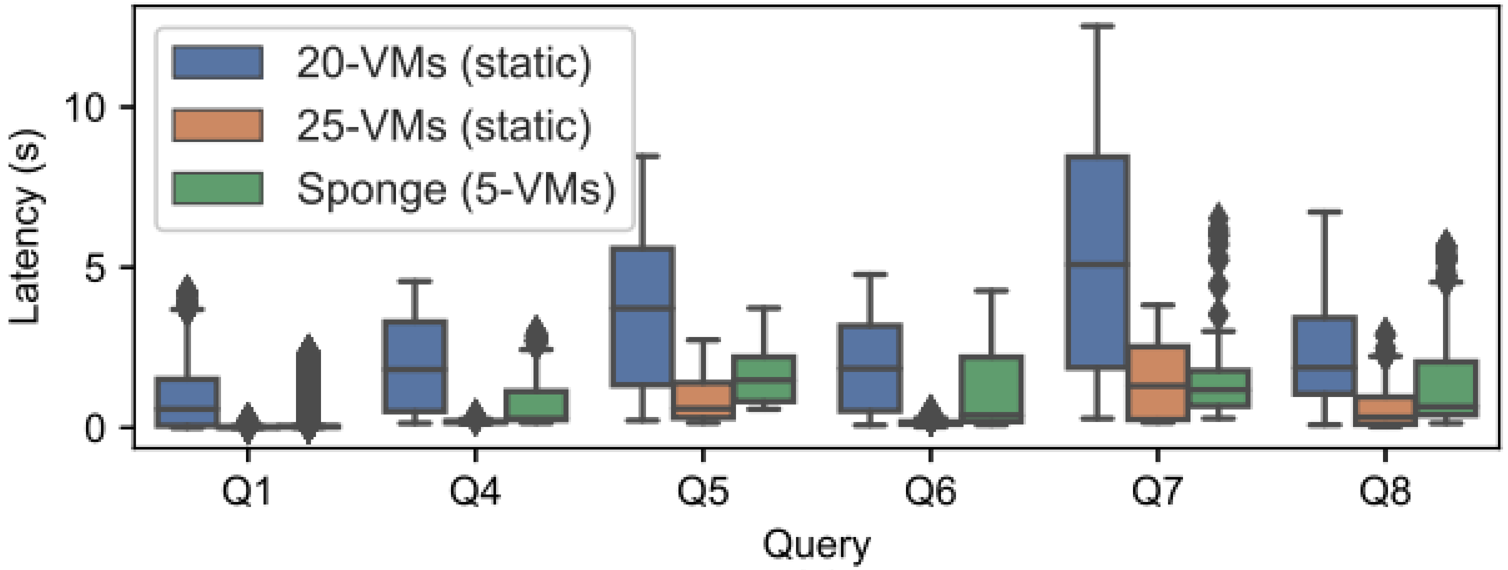
Evaluation: Graph Rewriting Effect

SFBase + RO + TO + MO + Warm-up

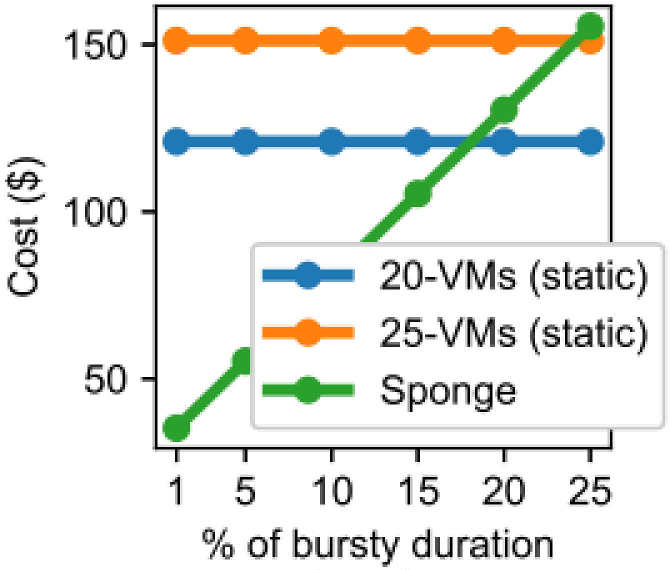


Evaluation: Latency-Cost Trade-Off

- Latency: 20-VMs > Sponge > 25-VMs
- Bursty duration < 15%



(a)



(b)

Paper Summary

Stream Processing

↓

Bursty Load

↓

- Redirect-and-merge
- Fast reactive scaling

Challenges

- **Migration with large operator states.**
- **Indirect data communication between SF instances.**
- **Quick decision making and scaling.**

Ideas

- **Compile-time Graph Rewriting Algorithm**
- **Reducing Cold Start Latency**
- **Watermark message**
- **Dynamic Offloading Policy**