

Distributed Dataflow Systems

FID3008

Max Meldrum <mmeldrum@kth.se>

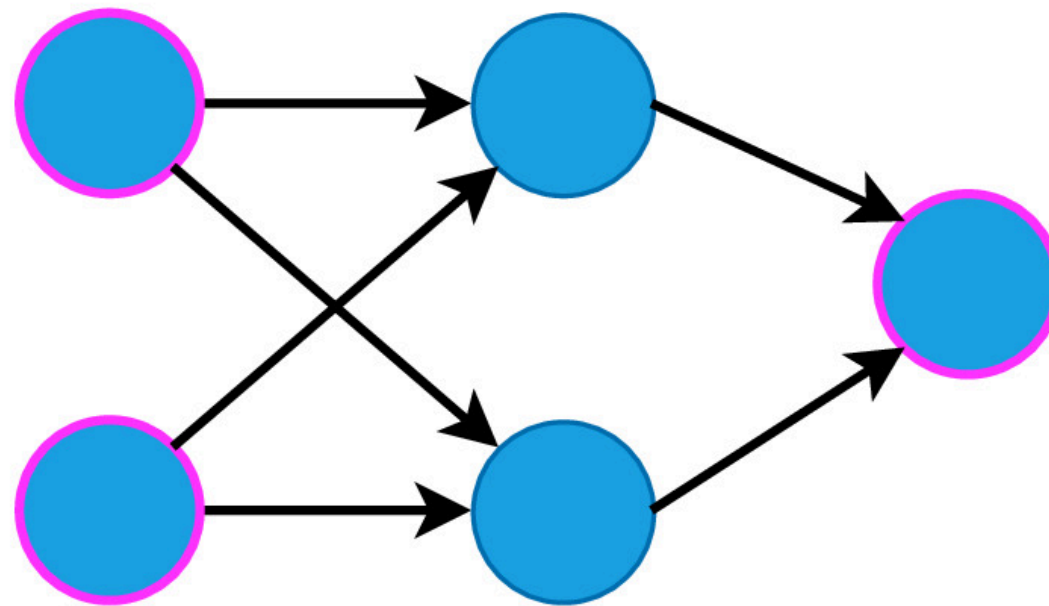
1. Milwheel: Fault-Tolerant Stream Processing at Internet Scale
2. Naiad: A Timely Dataflow System
3. Ray: A Distributed Framework for Emerging AI Applications

Outline

1. Background
2. Millwheel (Google)
3. Naiad (Microsoft)
4. Ray (UC Berkeley)
5. Thoughts/Rant
6. Summary

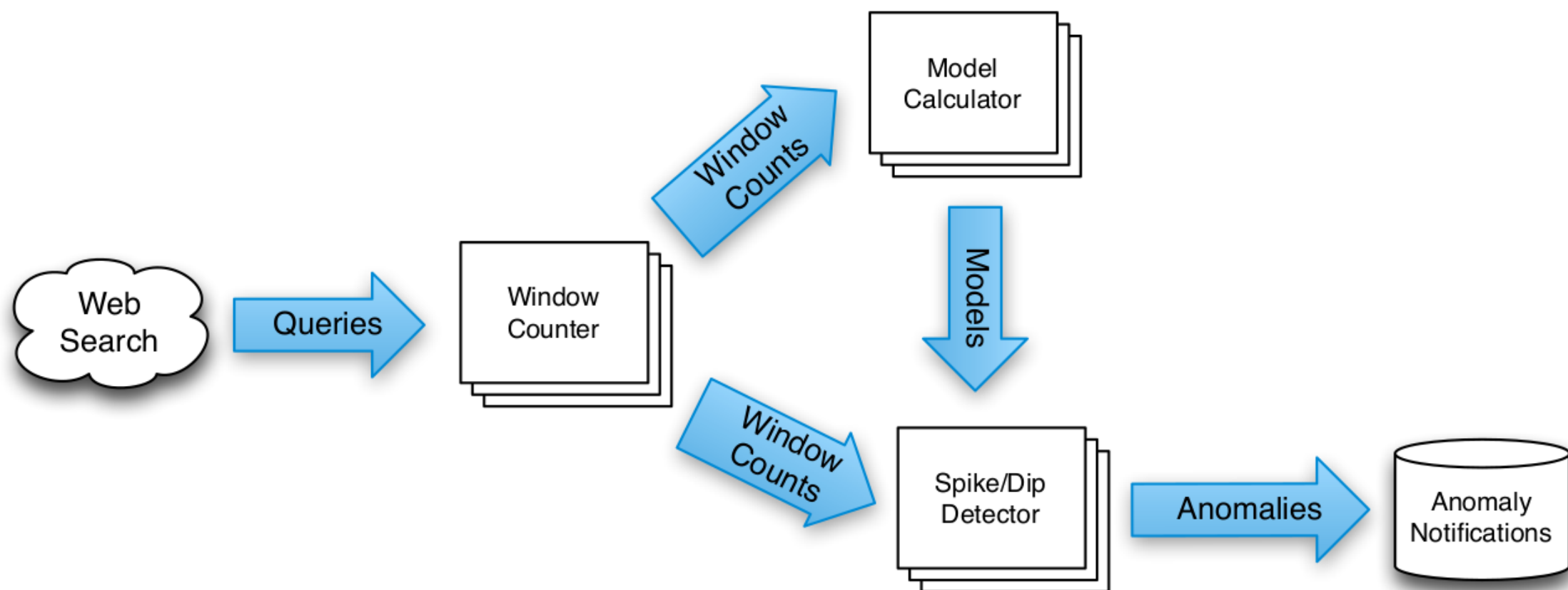
Dataflow Processing

- $G = (V, E)$, V = computational task, E = data channel
- Static vs. Dynamic

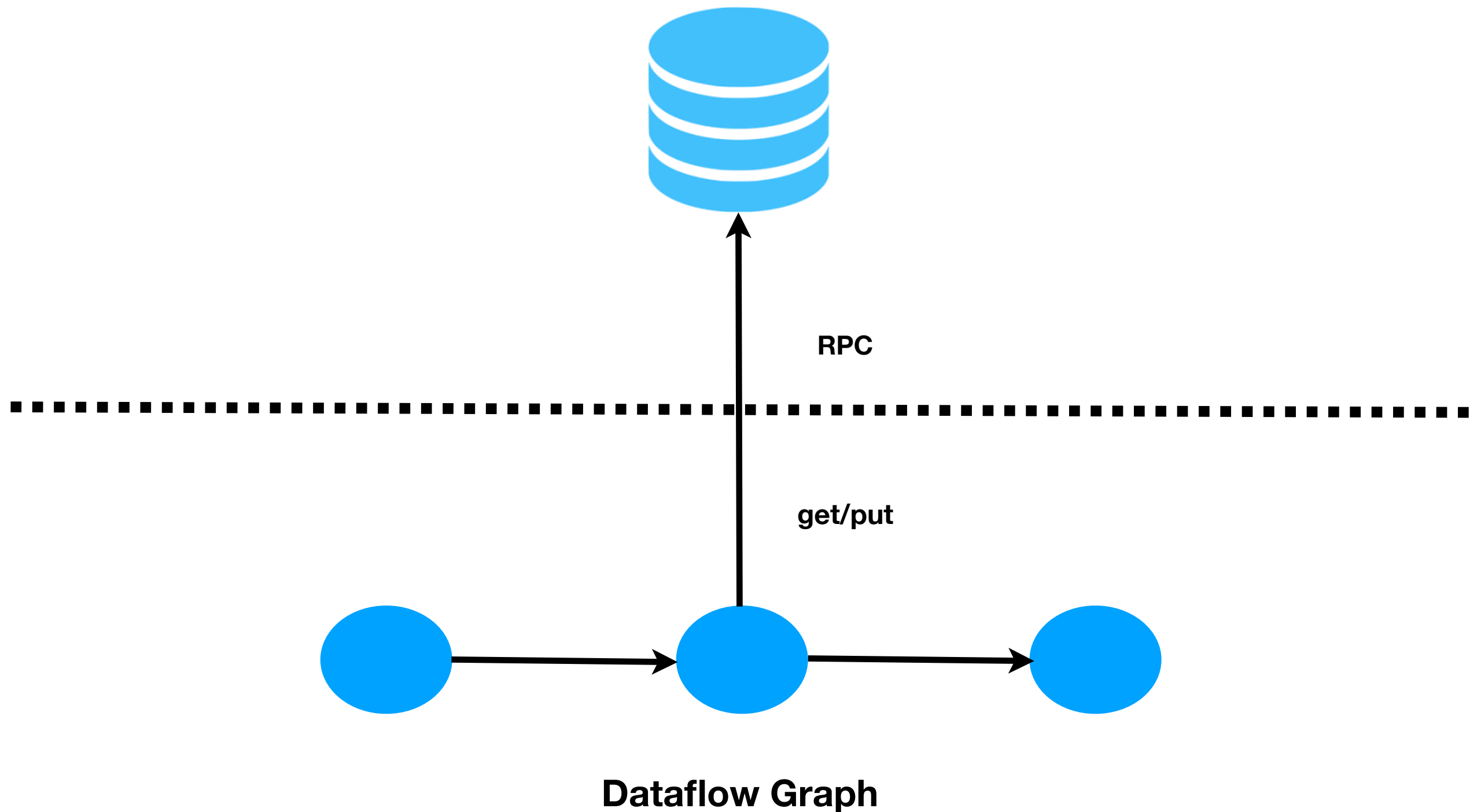


MillWheel (VLDB 13)

- Distributed Stream Processor
- Uses the Watermark notion to measure progress
- Provides exactly-once semantics
- Separates Compute from Storage



Separating Storage and Compute



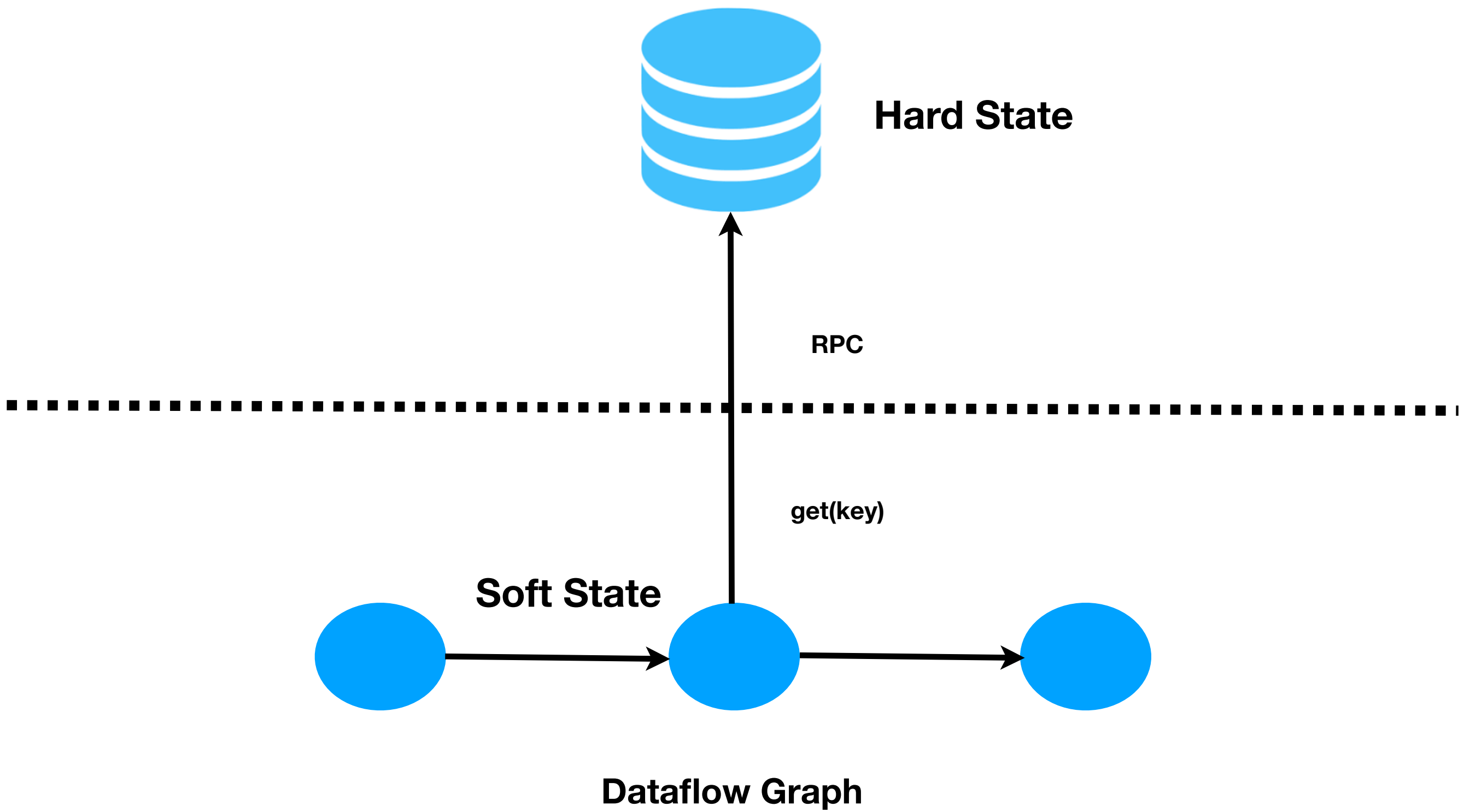
State Manipulation in MillWheel

- **Hard State**

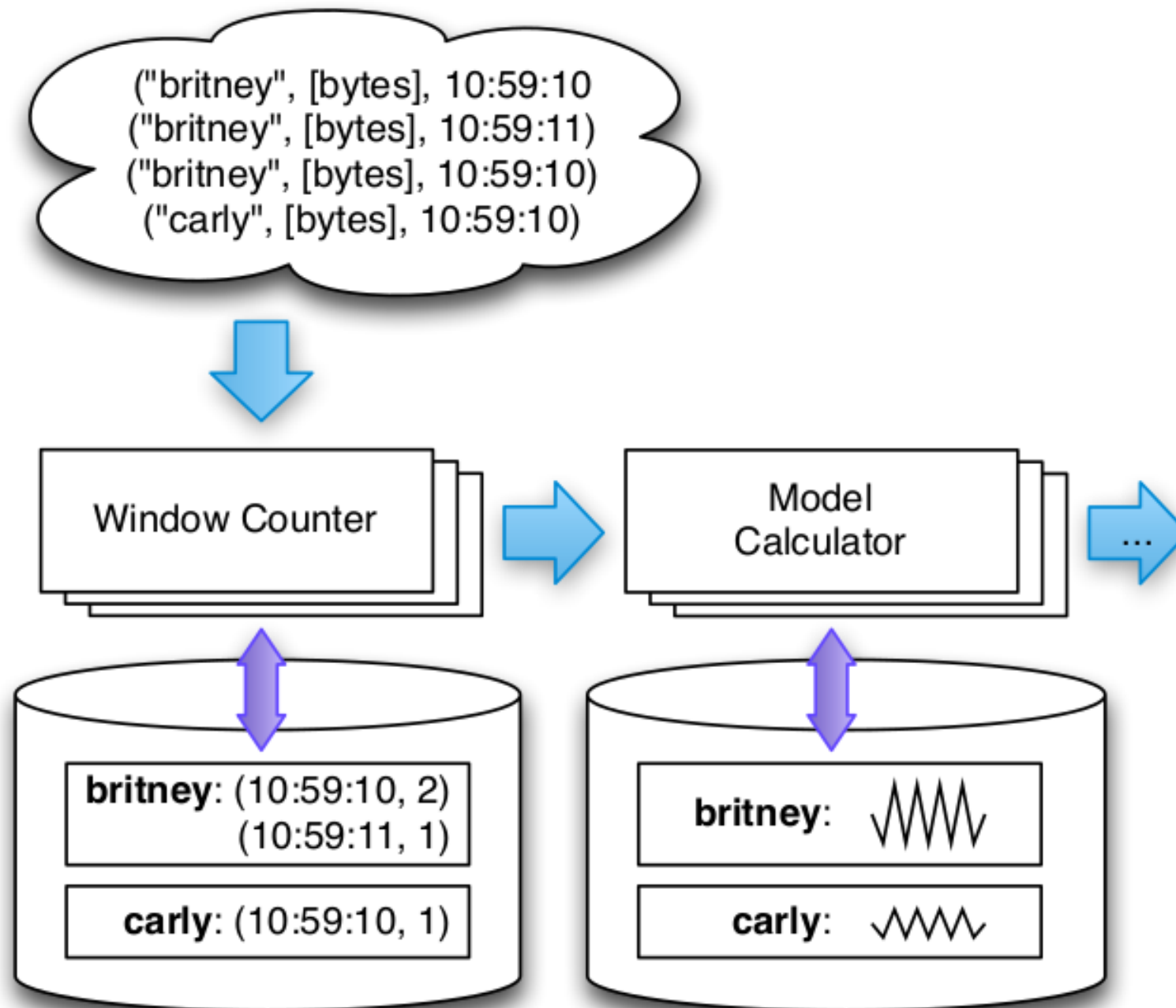
- Persisted in a global store

- **Soft State**

- In-memory caches/aggregates



Key Abstraction



Timers

Timers are per-key triggers that may be configured using:

- A low watermark value
- A specific wall time

Checkpointing & Delivery Guarantees

Checkpointing:

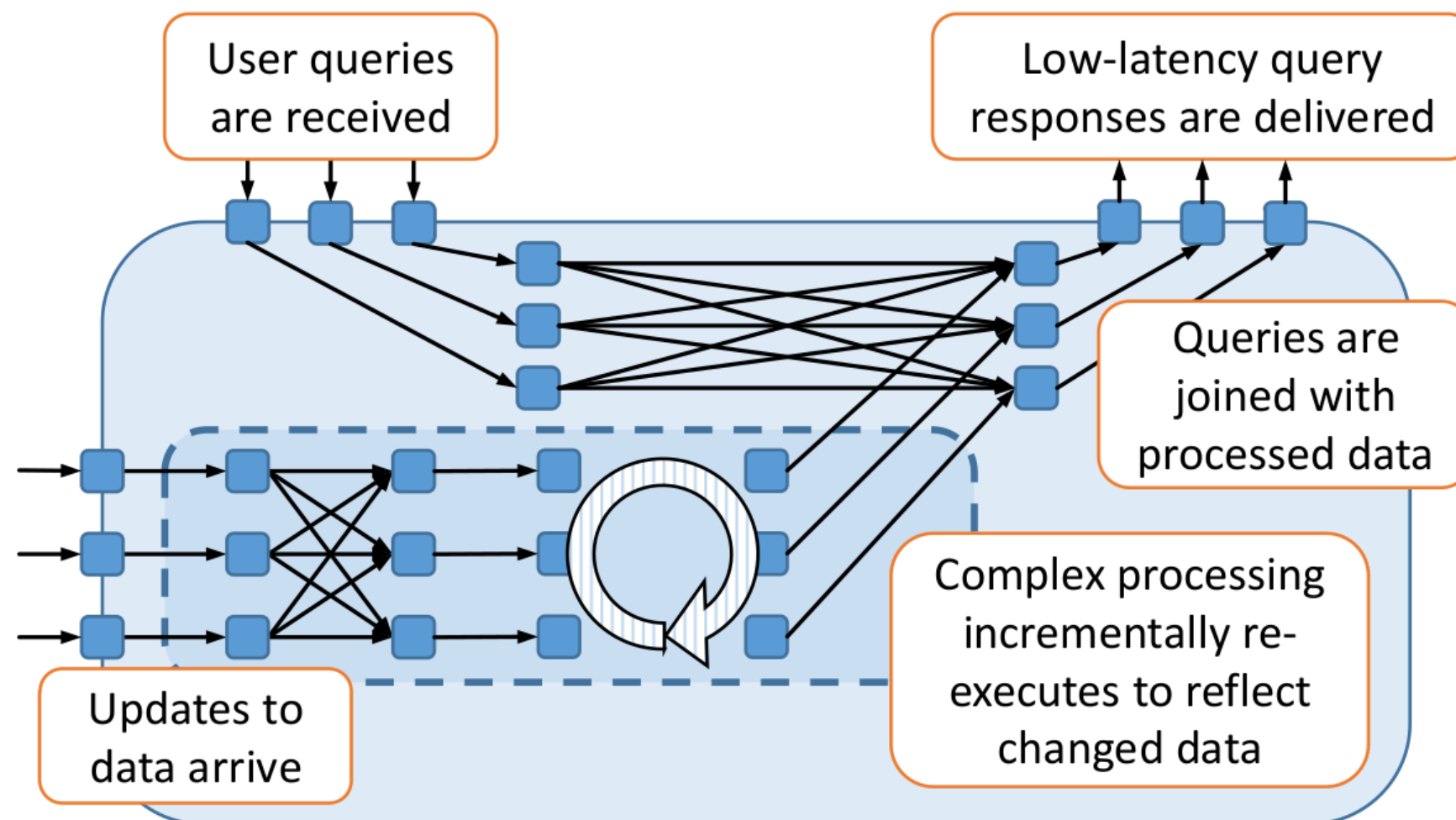
1. Happens per-key (includes state, timers)
2. Atomically committed into a single row in the global store

Delivery Guarantees:

1. Exactly-once
2. Strong Productions -- Checkpoint before delivery
3. Weak Productions -- Optimistic delivery

Naiad (SOSP 13)

Distributed Implementation of the Timely Dataflow model



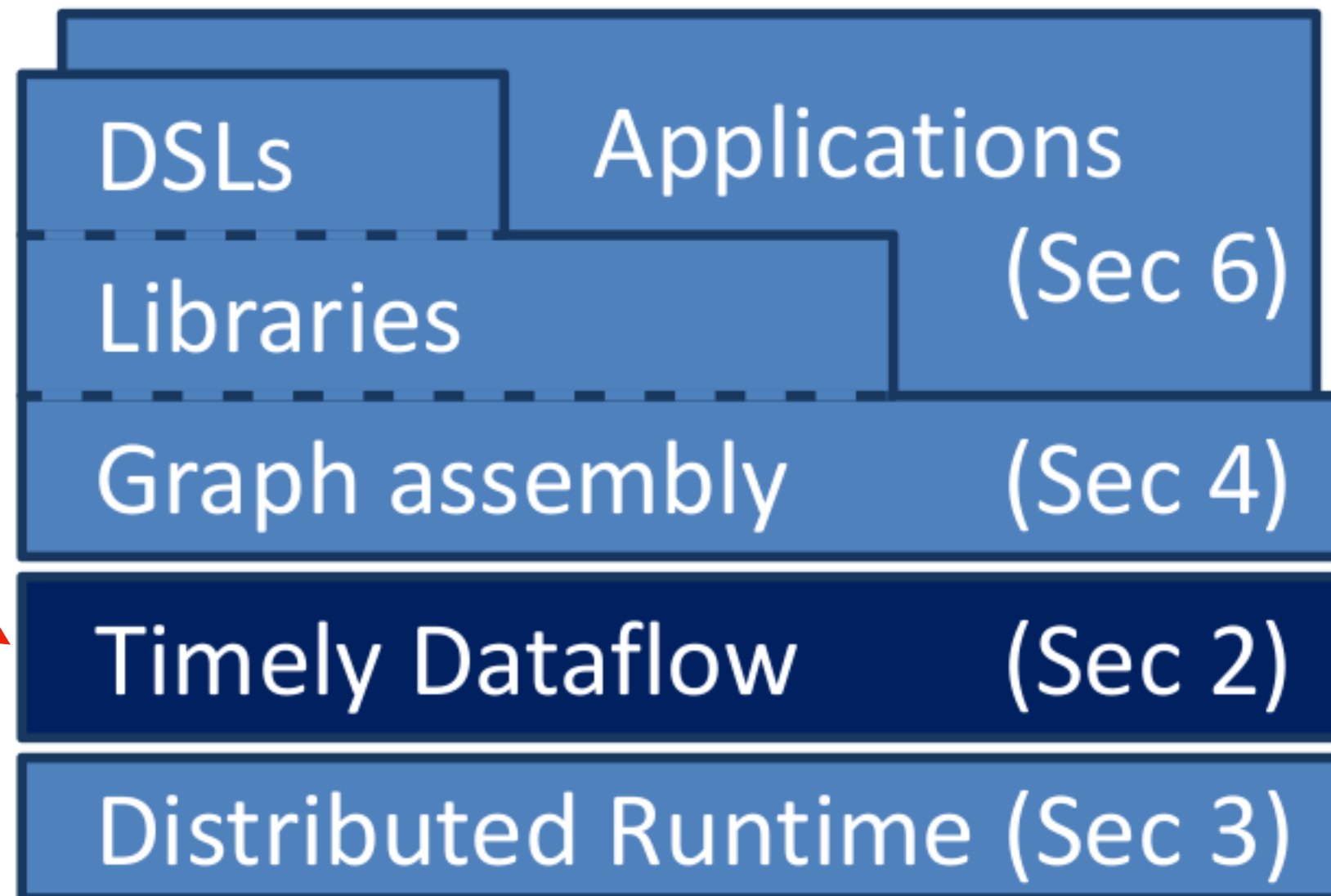
Naiad's Goal

Design a general-purpose system that offers:

- 1. High throughput**
- 2. Low-latency**
- 3. Ability to perform iterative and incremental computations**

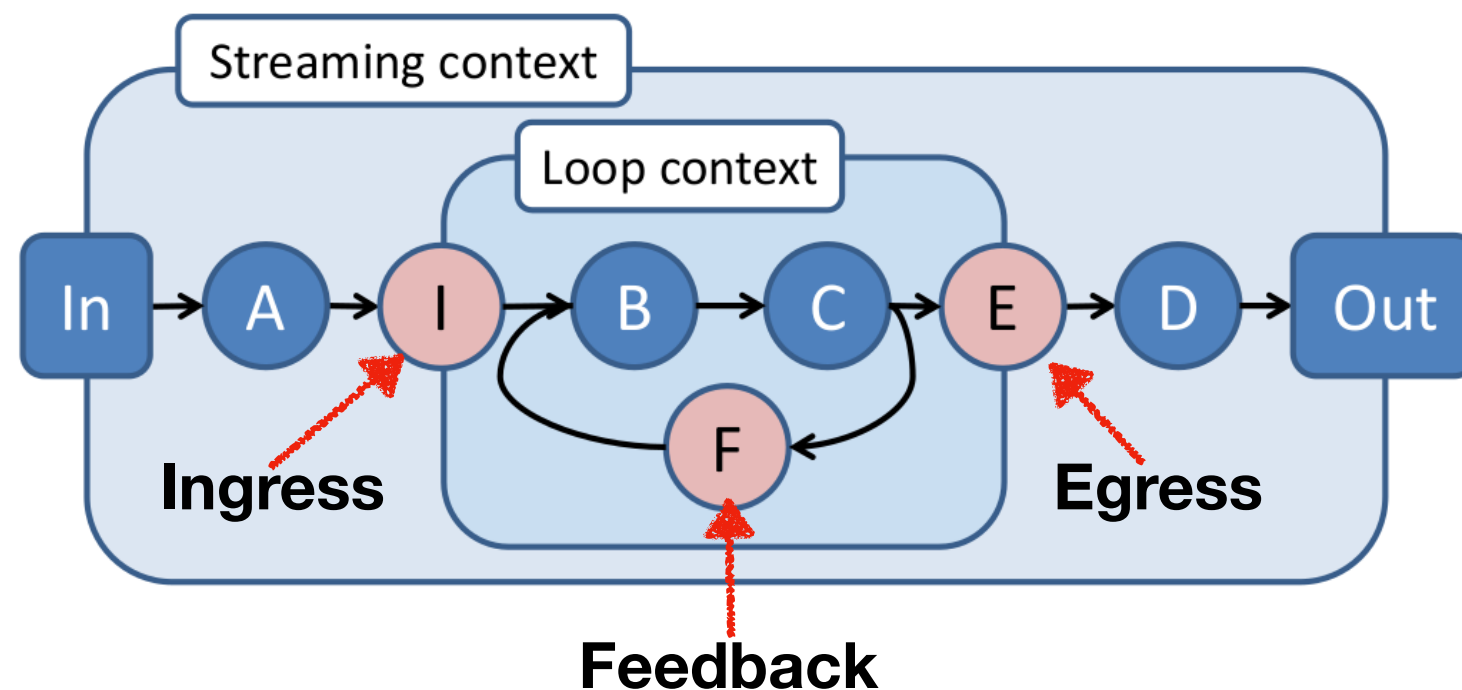
The paper claim that existing systems support some but not all of the requirements.

The Naiad Stack



Timely Dataflow Model

1. Graphs are directed but may contain cycles
2. Structured Loops are introduced to enable dataflow feedback



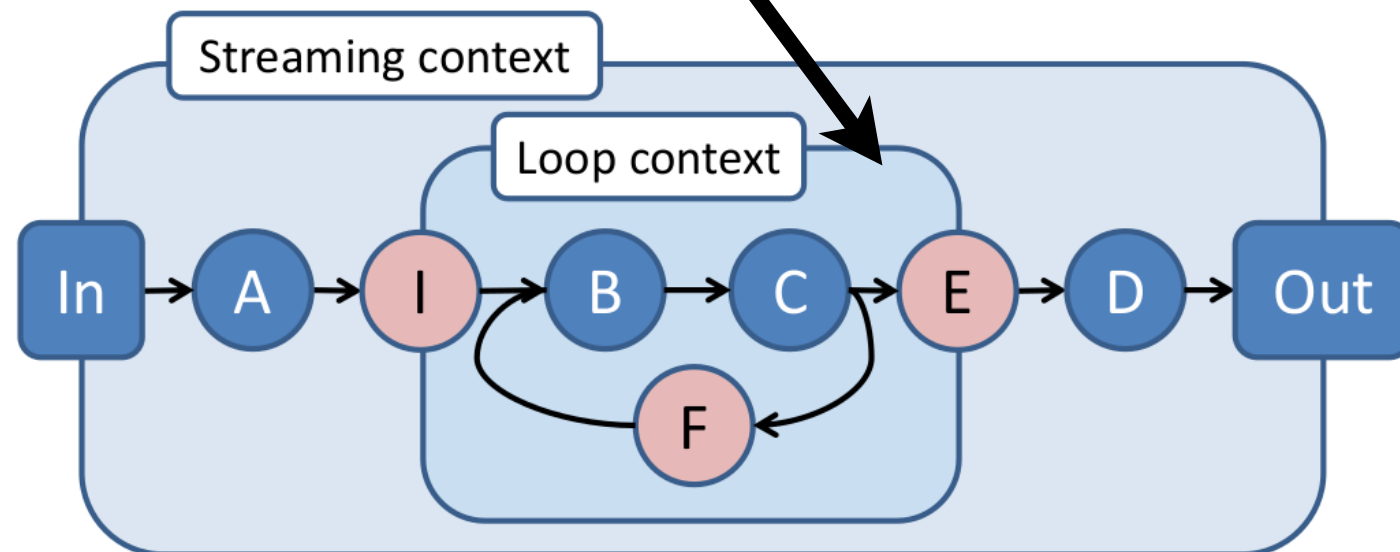
3. Utilises logical timestamps to track global progress

Logical Timestamps

The timestamps are defined as:

$$\text{Timestamp} : (\overbrace{e \in \mathbb{N}}^{\text{epoch}}, \overbrace{\langle c_1, \dots, c_k \rangle \in \mathbb{N}^k}^{\text{loop counters}})$$

Loop counters helps the system identify which iteration it is in



Vertex Computation

Vertices send and receive timestamped messages

Each vertex v implements the following callbacks:

$v.ONRECV(e : \text{Edge}, m : \text{Message}, t : \text{Timestamp})$

$v.ONNOTIFY(t : \text{Timestamp}).$

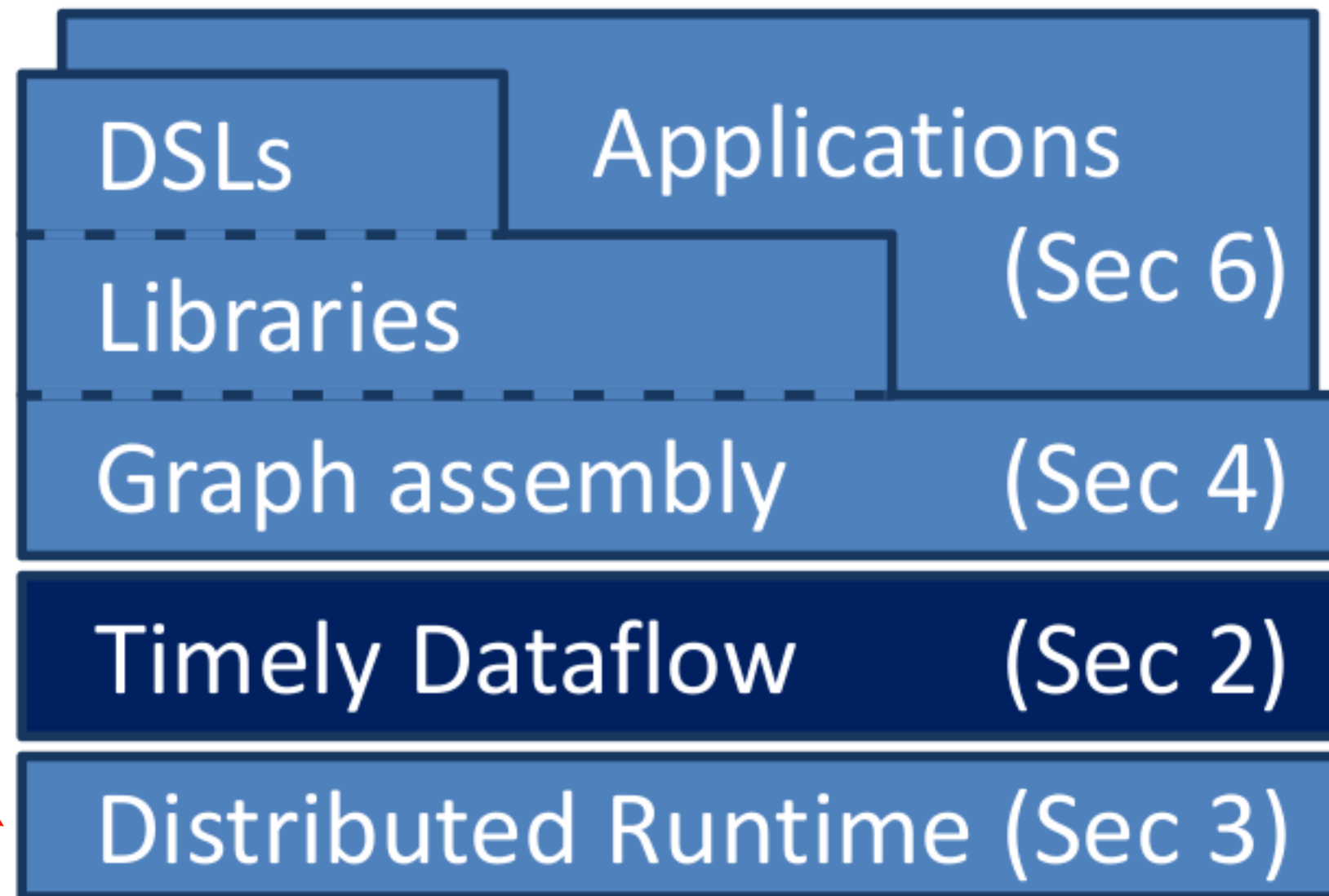


All messages up to timestamp t received

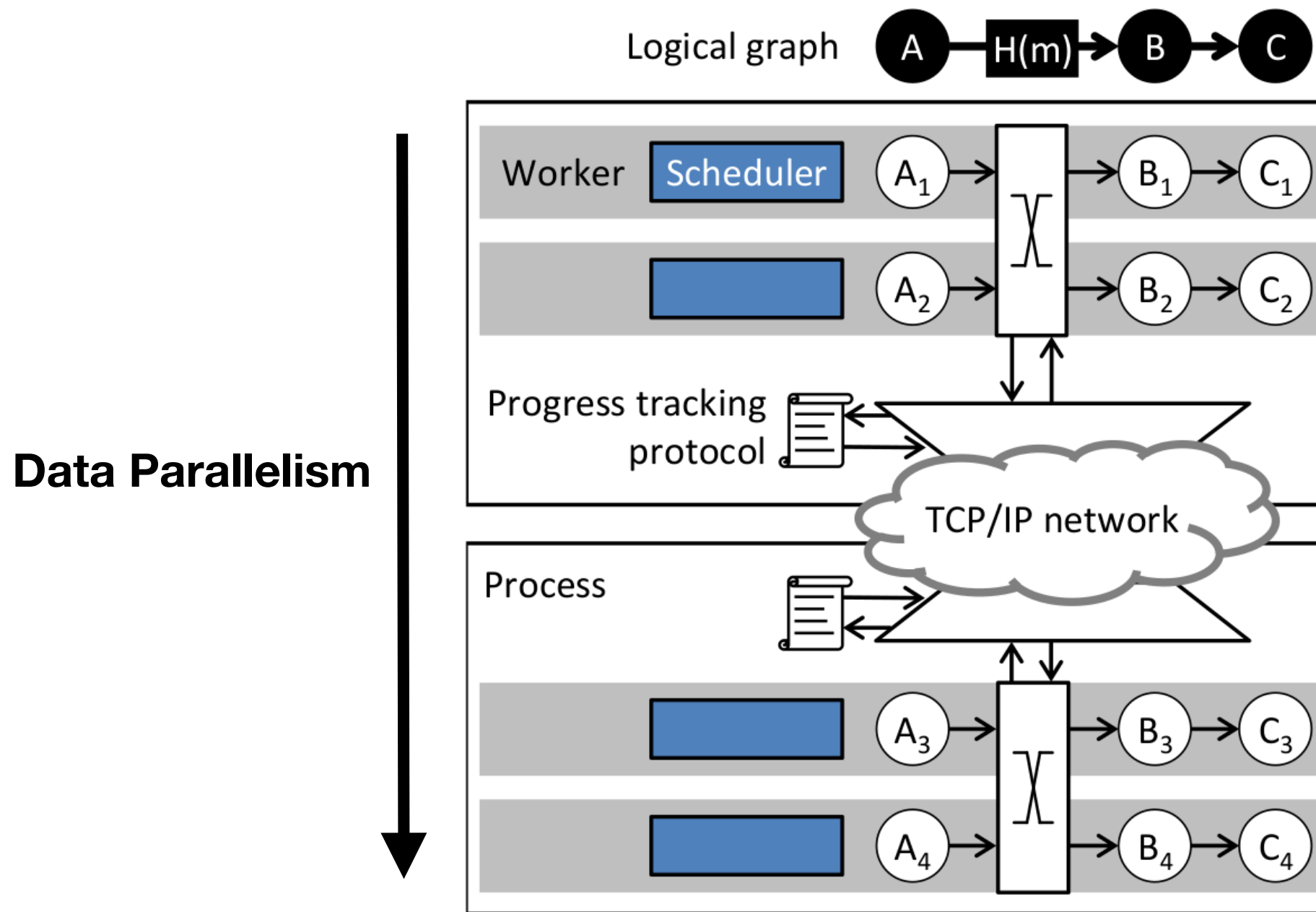


Handle msg

The Naiad Stack

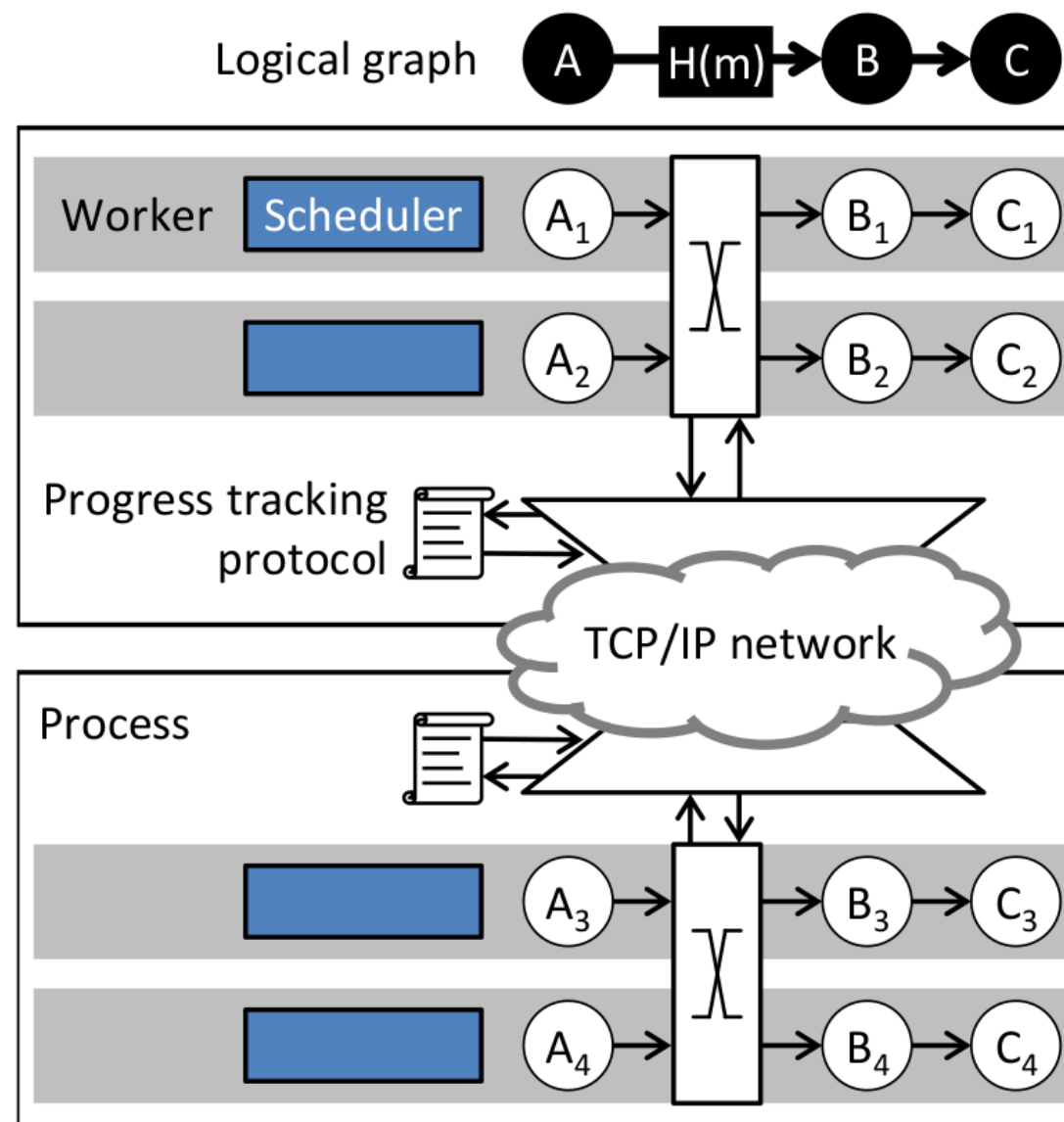


Naiad Cluster



Distributed Progress Tracking

1. Process keeps a local view of the progress
2. Process sends updated view to central coordinator
3. Coordinator broadcasts effects to all workers



Checkpointing & System Optimisations

Checkpointing:

Halting Snapshot Algorithm

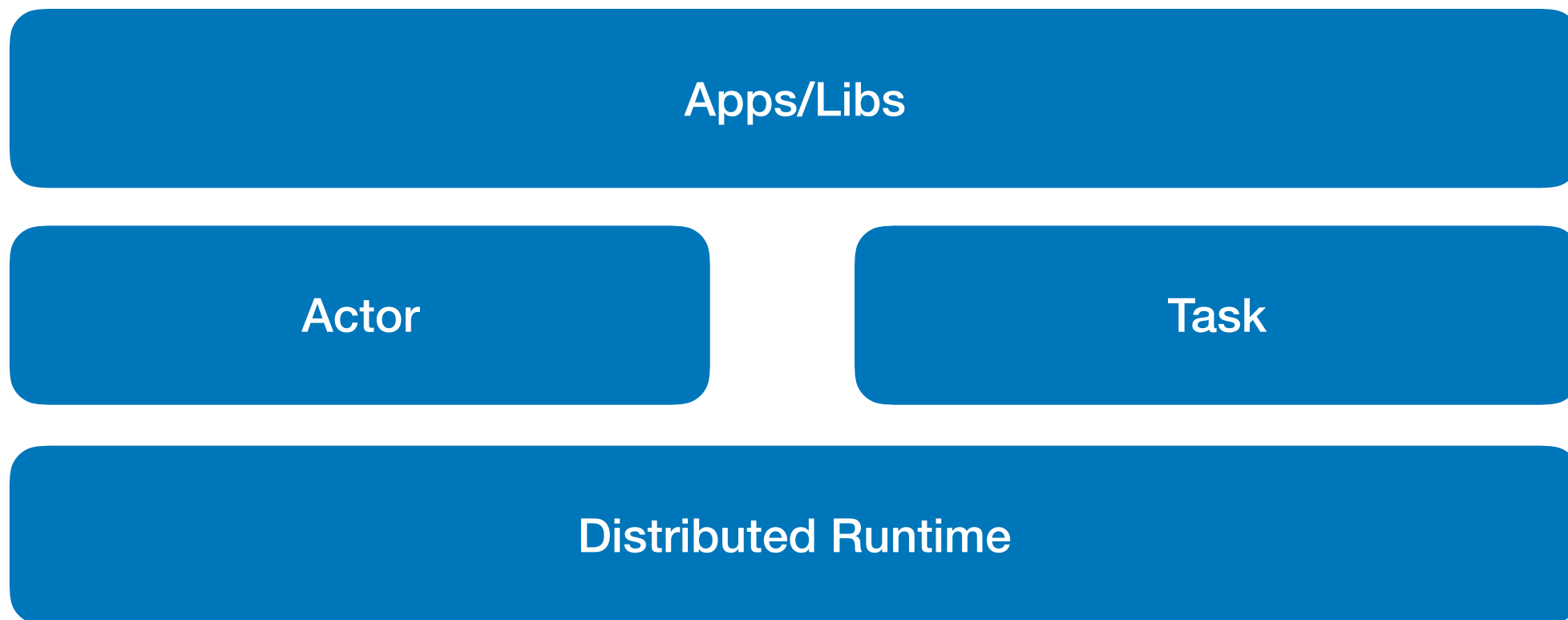
System Optimisations:

1. TCP: Disable the Nagle Algorithm
2. Avoid GC by recycling
3. Avoid data structure contention

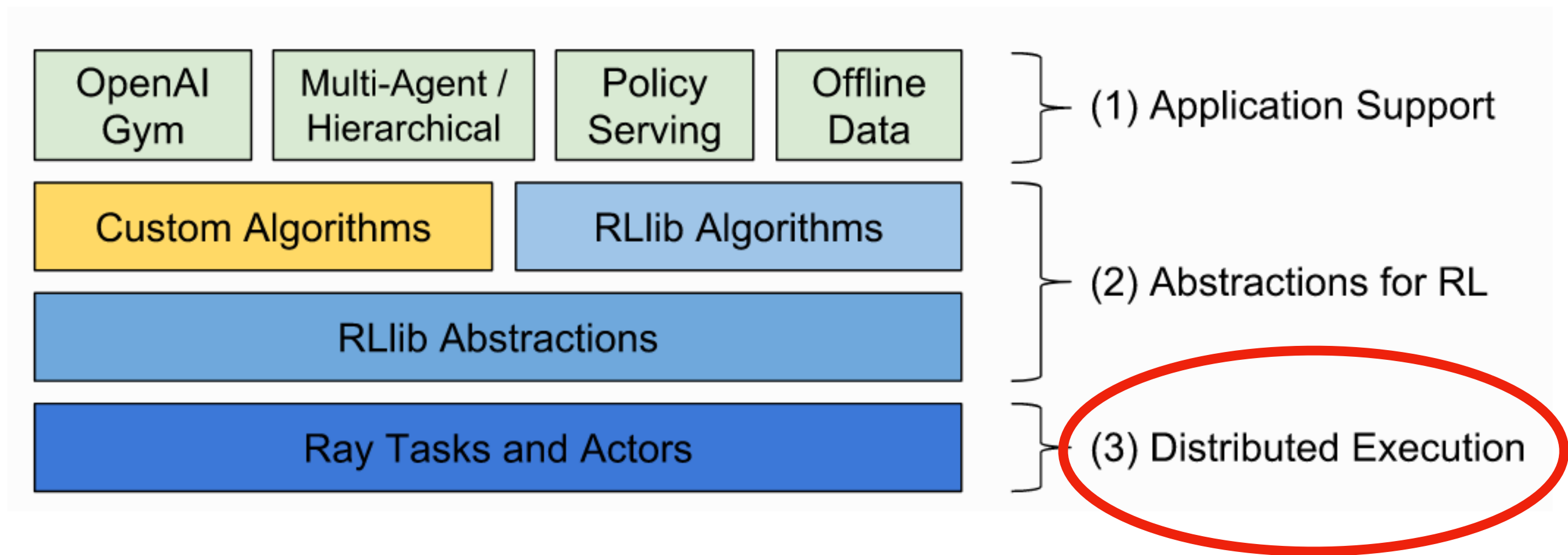
Ray (OSDI 18)

- A Distributed Framework for AI
- Python API / C++ Backend
- Utilises a Dynamic Task graph

The Ray Stack



RLlib



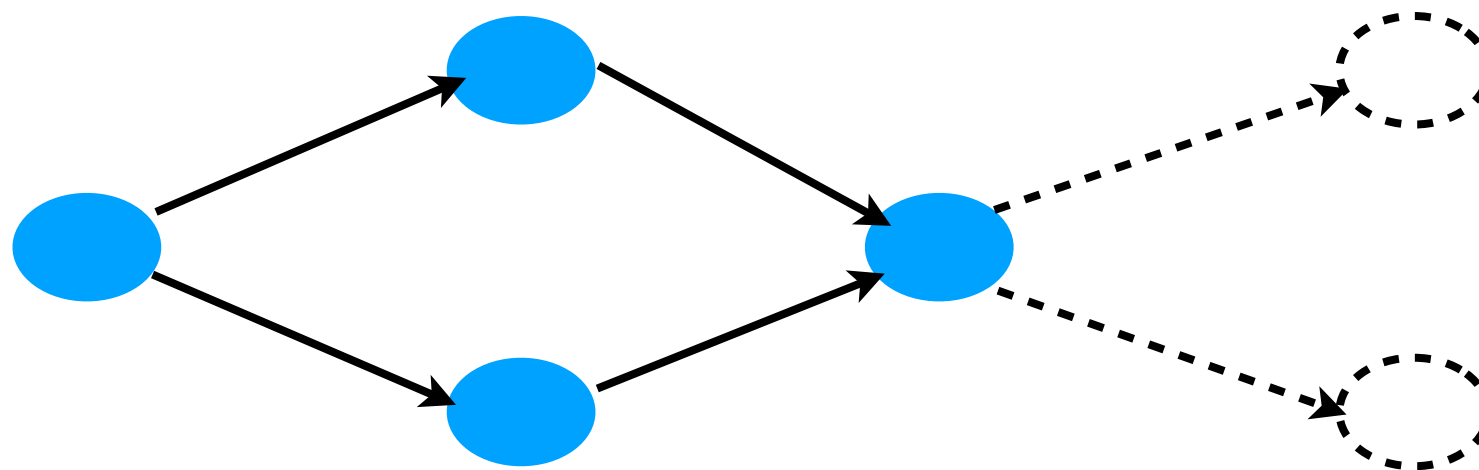
System Requirements

Flexible Computational Model:

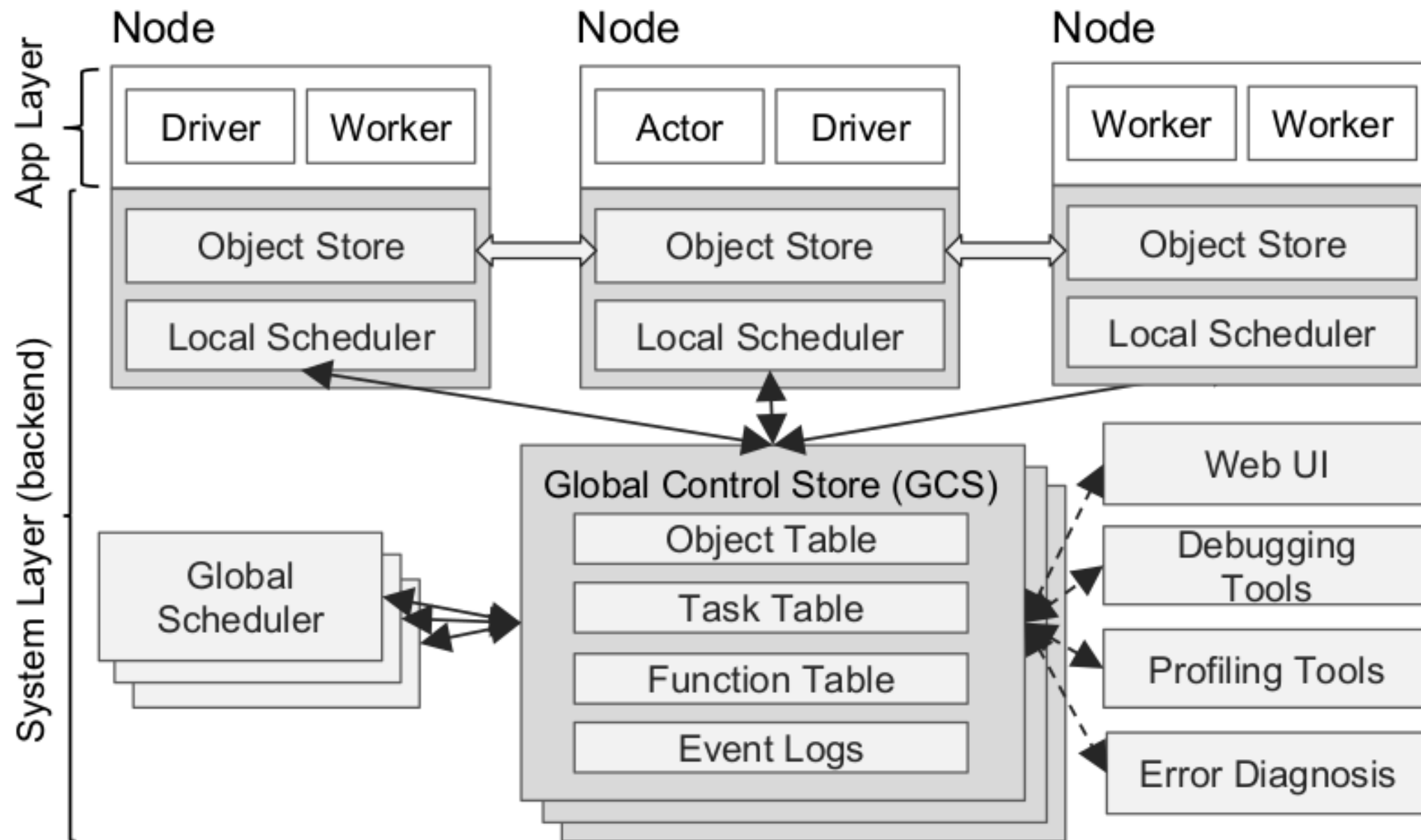
Stateful (Ray Actor) and Stateless (Ray Tasks) computation

Dynamic Execution:

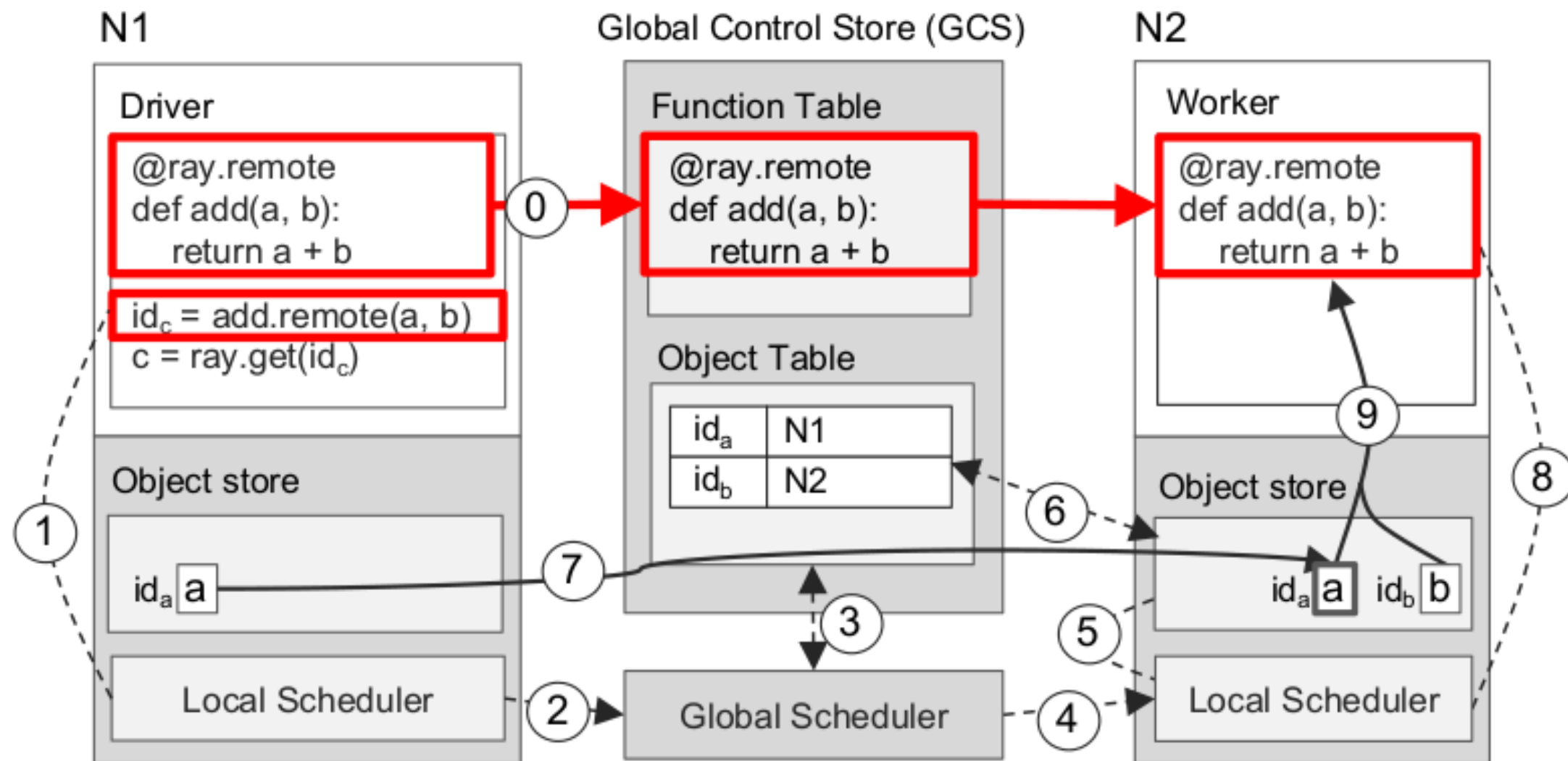
Ability to spawn new tasks (e.g., simulation)



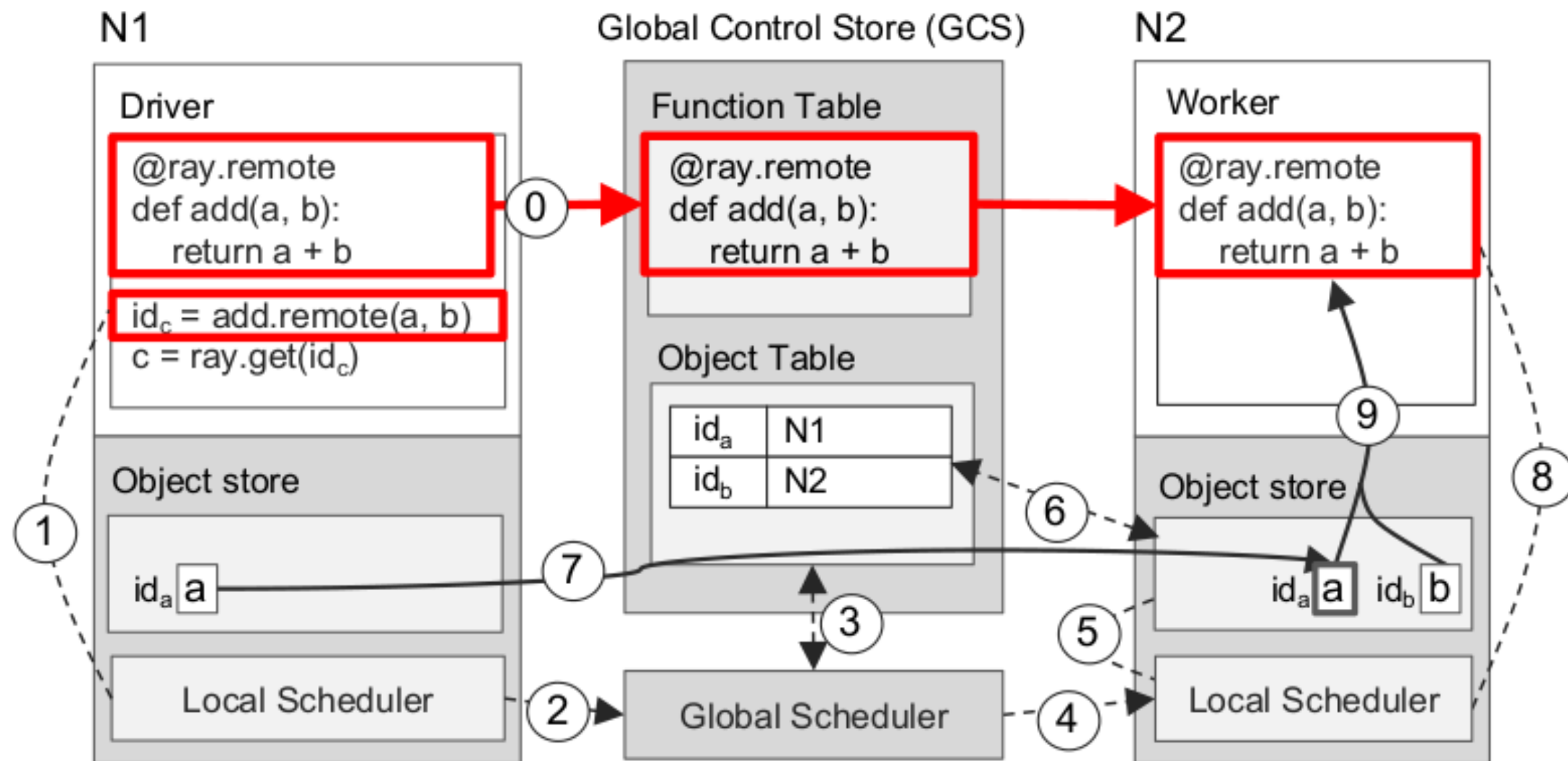
System Architecture



Task Execution



Failure Recovery?



Thoughts/Rant

I. Separating storage and compute?










Access Latencies

	L3	DRAM	SSD	HDD
Write	20 ns	60 ns	25,000 ns	10,000,000 ns
Read	20 ns	60 ns	300,000 ns	10,000,000 ns

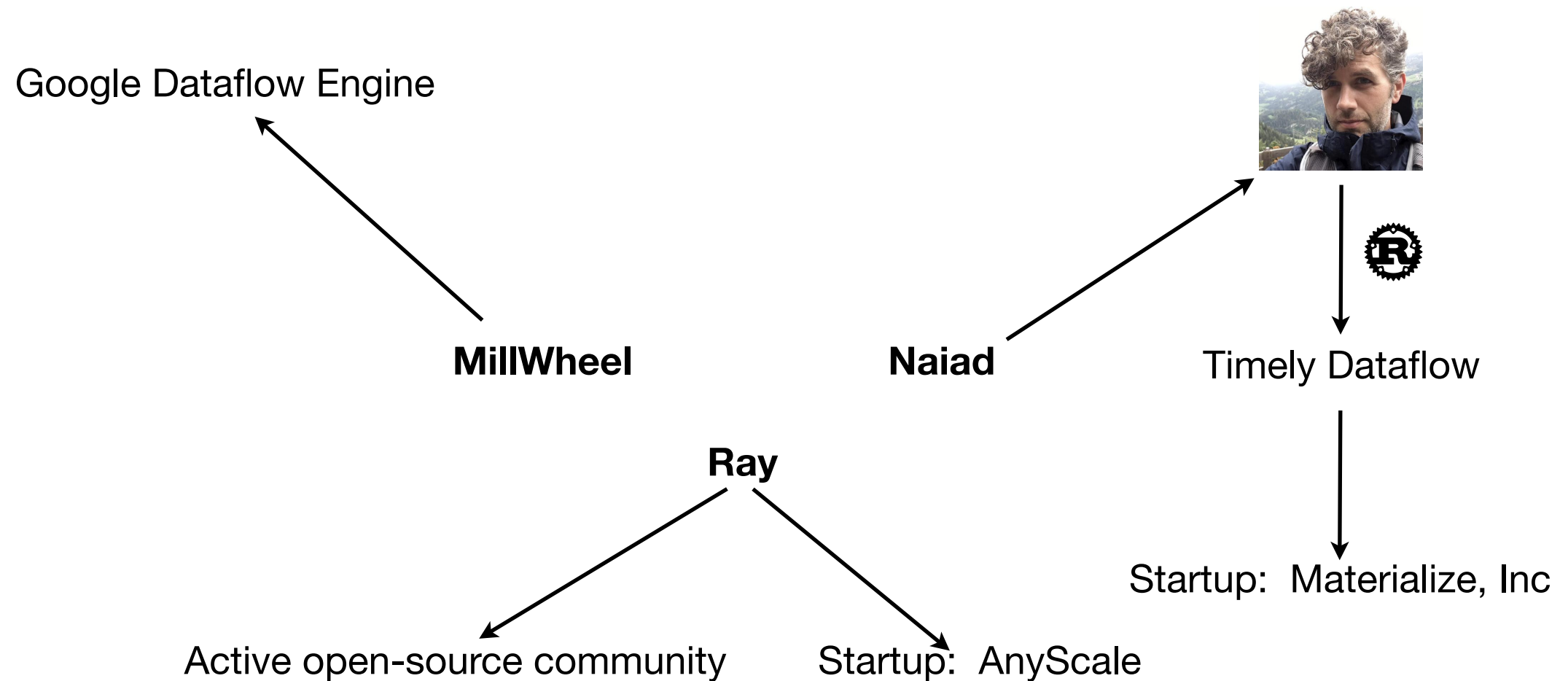
Thoughts/Rant

1. Separating storage and compute?
2. Assumption of the key abstraction
3. Measuring progress. Keep it simple
4. Consistency guarantees

Comparison Table

	Dataflow Type	Parallelism Focus	Progress Tracking	State	Main Use Case	Max's Complexity Ranking
MillWheel	Static	Data	Low Watermarks	Global Store	Streaming	 
Naiad	Static	Data	Central Coordinator + Broadcast	Local/ Global Store	Streaming + Iterative	   
Ray	Dynamic	Task	None	Global Store	Reinforcement learning	  

Where are they now?



Summary

MillWheel:

Distributed Stream Processor for
large-scale real-time processing at Google

Naiad:

Distributed implementation of the Timely Dataflow model
for data-parallel applications

Ray:

Distributed data processing
framework targeting AI applications (e.g., RL)