Flumina: Correct Distribution of Stateful Streaming Computations

Konstantinos Kallas*; Filip Nikšić*; Caleb Stanford*; Rajeev Alur

UCSD -- February 2020

Motivation

Stream Processing

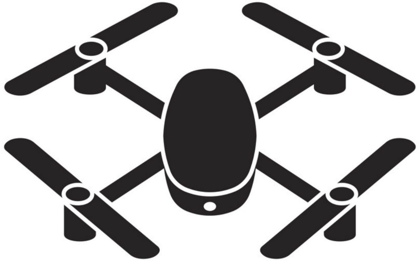
Compared to batch processing:

- Low response times
- Can support larger datasets
- More natural for some applications

Many applications

Video Streaming

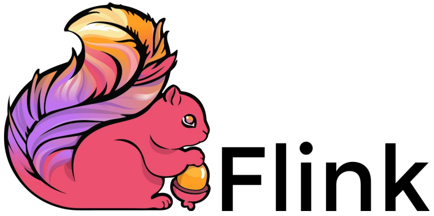
Drones



Medical Devices

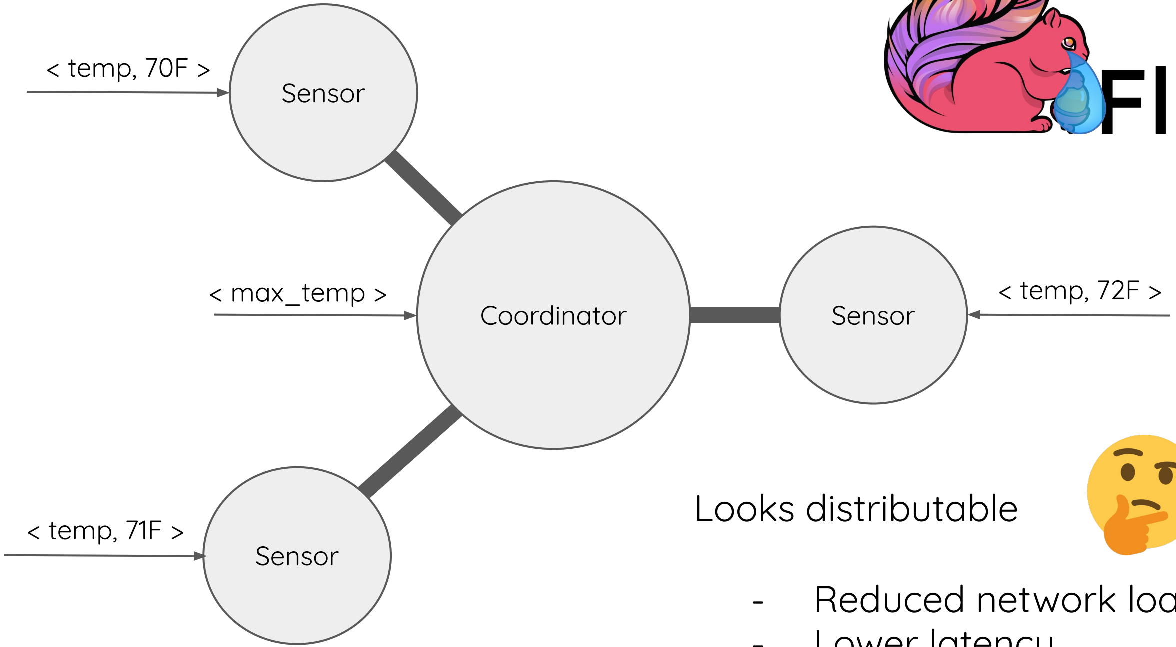
More Video Streaming

Solutions for Distributed Stream Processing



- Dataflow or SQL
- Great Performance
- High-level
- Support many computations
  - map
  - filter
  - keyBy

What about this?

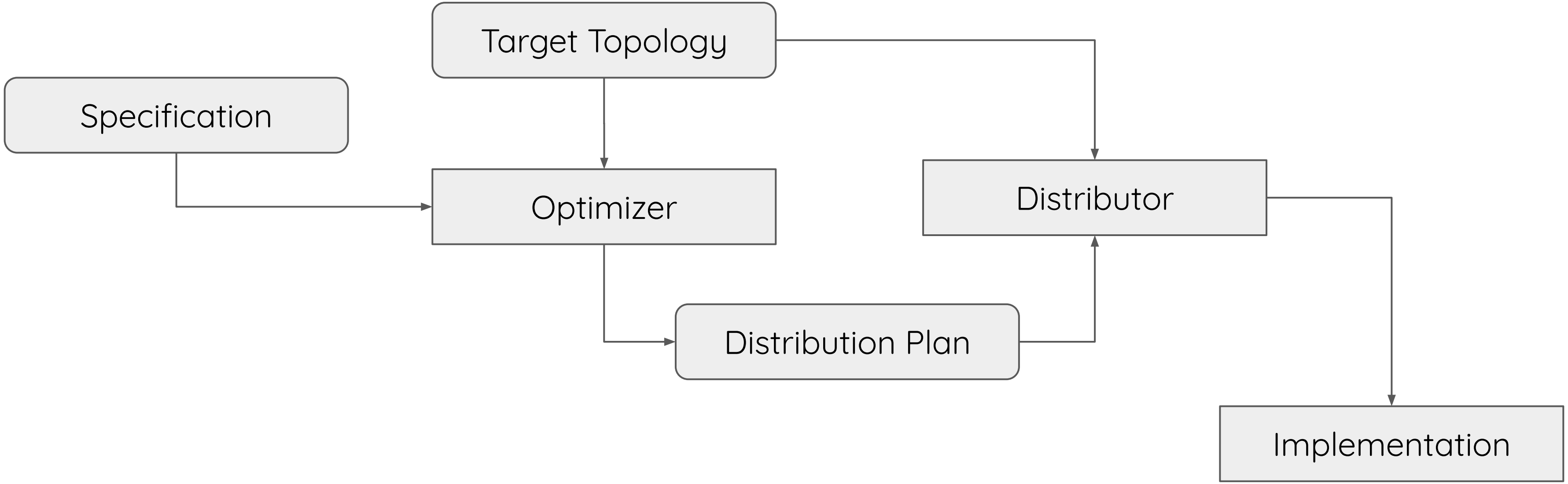


Flink

Looks distributable 🤔

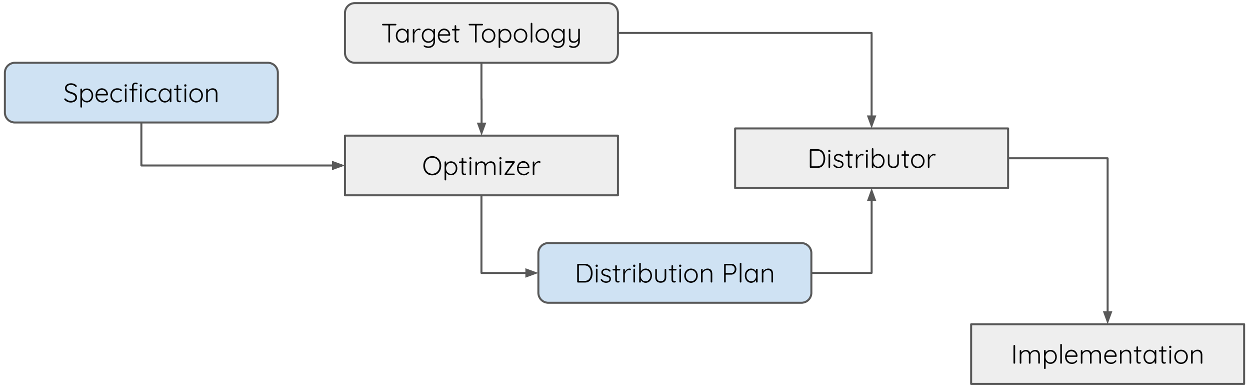
- Reduced network load
- Lower latency
- Privacy

Flumina

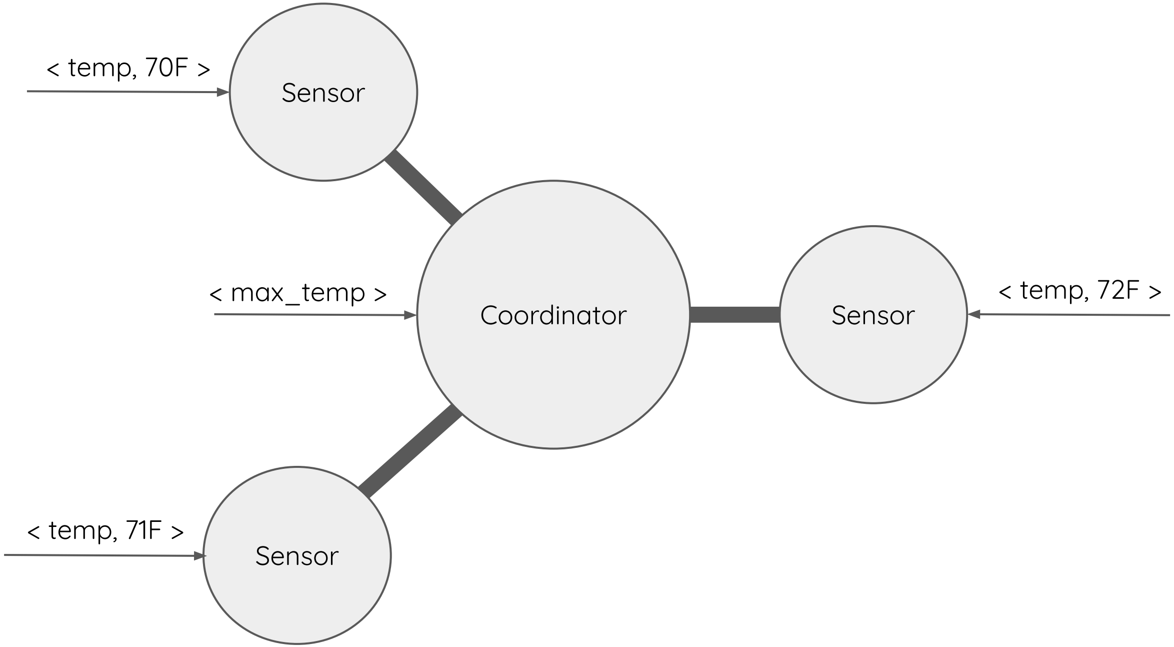


Main idea:
View streams as partial orders

Conceptual model



Example



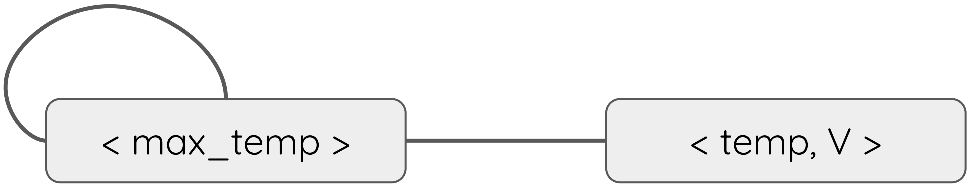
```
state  := int // max temp so far
temp_e := <temp, int>

update_temp :: temp_e -> state -> state
update_temp <temp, Val> OldMax :=
    return max(OldMax, Val)
```

```
max_e := <max_temp>

update_max :: max_e -> state -> state
update_max <max_temp> OldMax :=
    output(<day_max_temp, OldMax);
    return 0
```

Dependency relation

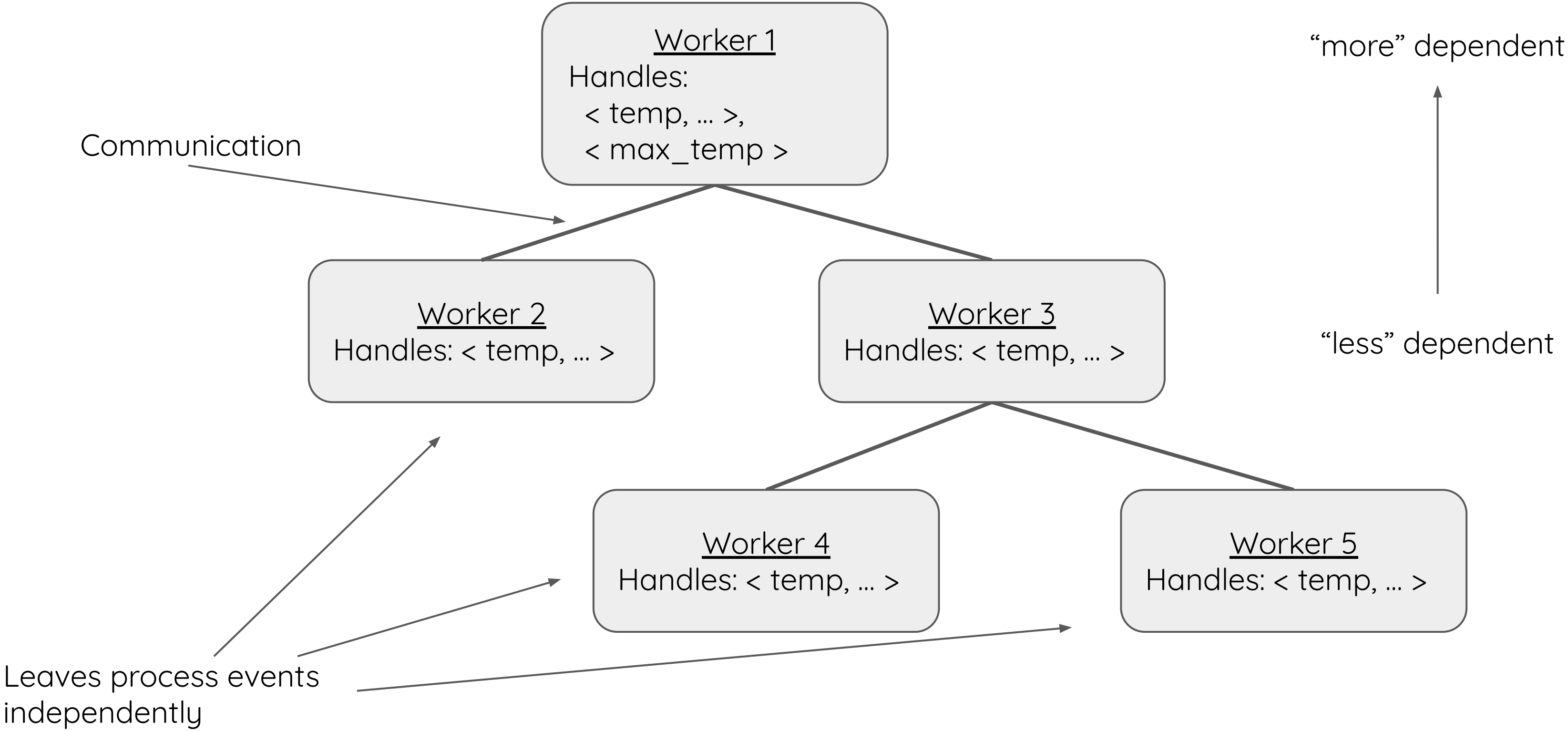
< max_temp > events depend on < temp, V > events

< max_temp > events depend on < max_temp > events



Independent events can be processed concurrently

Distribution model



Worker 1

Handles:

< temp, … >,

< max_temp >

Worker 2

Handles: < temp, … >

Worker 3

Handles: < temp, … >

Worker 4

Handles: < temp, … >

Worker 5

Handles: < temp, … >

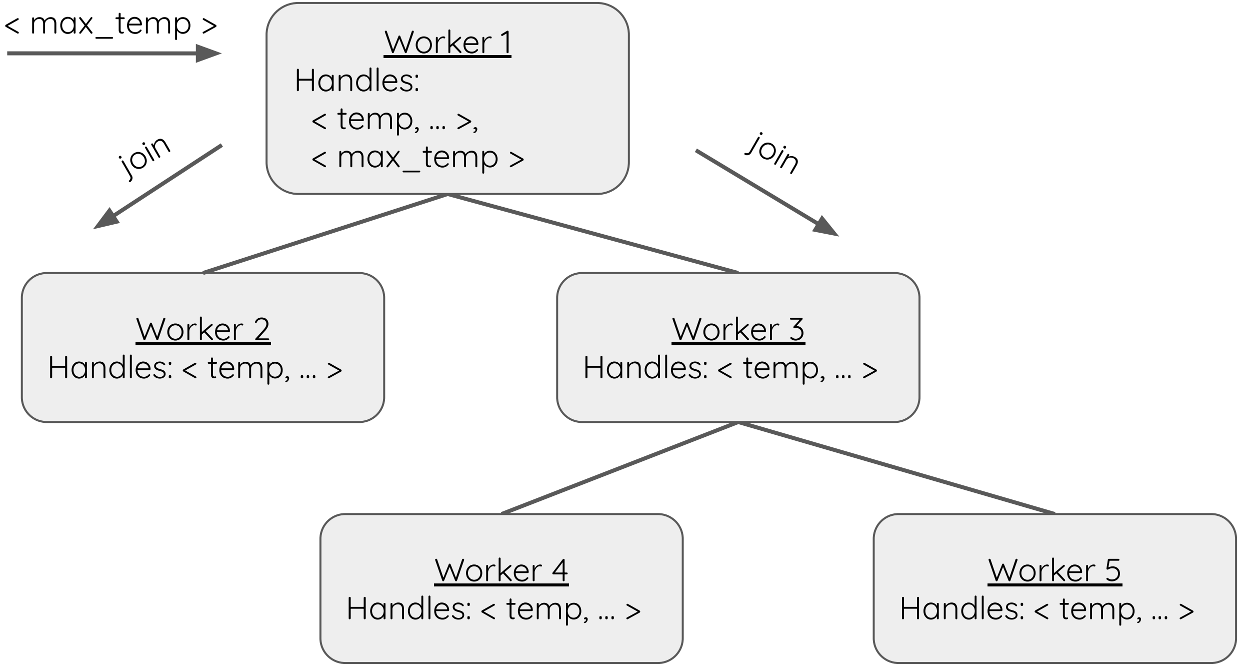
Communication

Leaves process events independently

“more” dependent

“less” dependent

Processing dependent events



< max_temp >

Worker 1
Handles:
< temp, … >,
< max_temp >

join

join

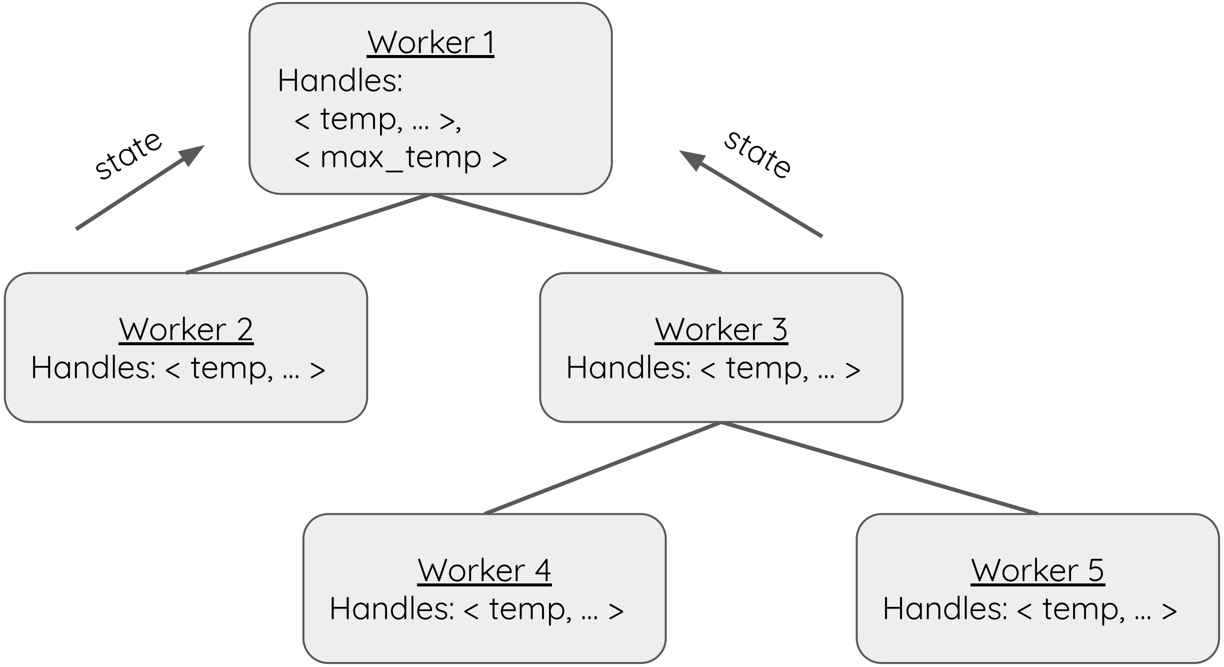
Worker 2
Handles: < temp, … >

Worker 3
Handles: < temp, … >

Worker 4
Handles: < temp, … >

Worker 5
Handles: < temp, … >

Processing dependent events



Worker 1

Handles:

< temp, … >,

< max_temp >

state

state

Worker 2

Handles: < temp, … >

Worker 3

Handles: < temp, … >

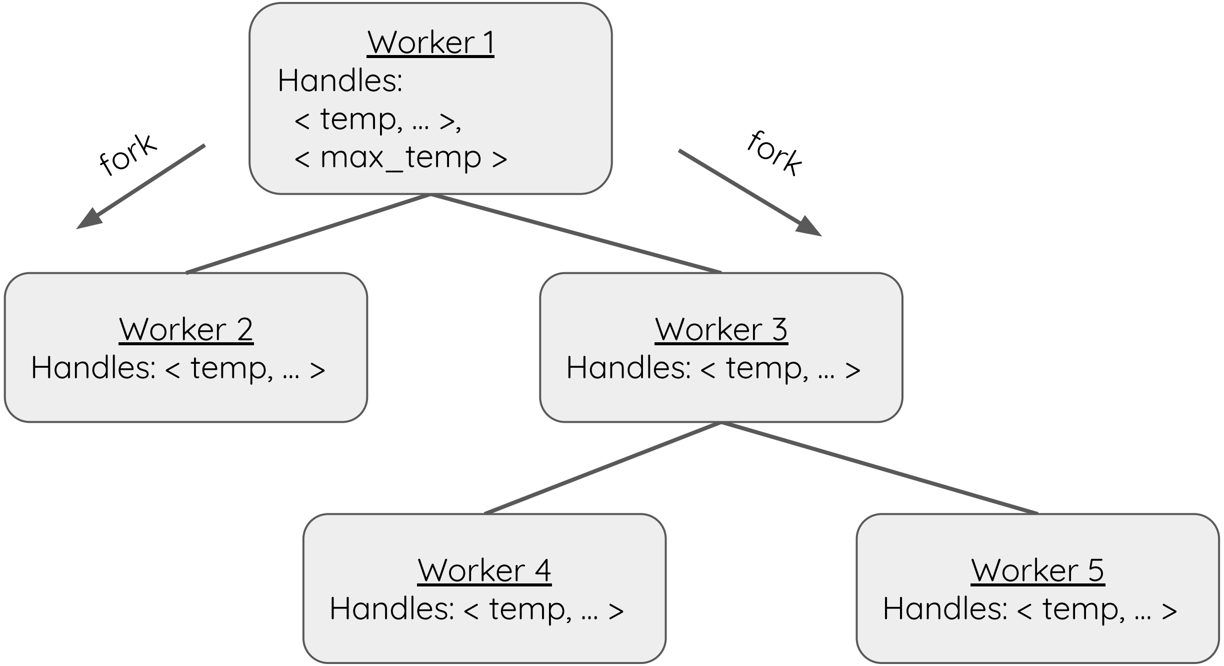
Worker 4

Handles: < temp, … >

Worker 5

Handles: < temp, … >

Processing dependent events



Worker 1

Handles:

< temp, … >,

< max_temp >

fork

fork

Worker 2

Handles: < temp, … >

Worker 3

Handles: < temp, … >

Worker 4

Handles: < temp, … >

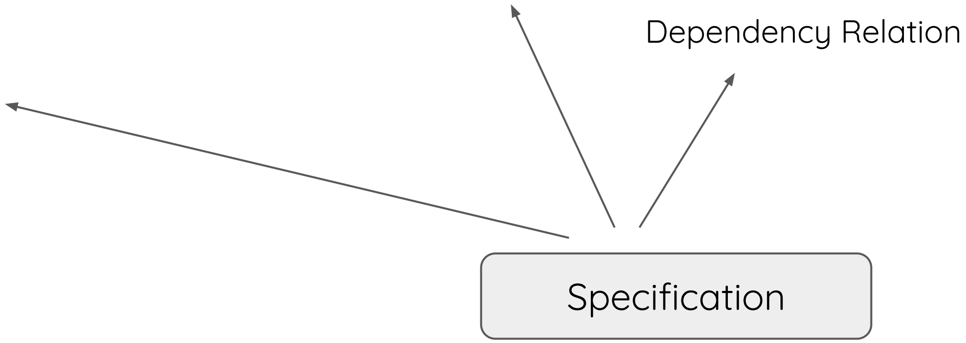
Worker 5

Handles: < temp, … >

Fork - Join

```
// State
state  := int // max temp so far

// Events
temp_e := <temp, int>
max_e := <max_temp>

update_temp :: temp_e -> state -> state
update_temp <temp, Val> OldMax :=
    return max(OldMax, Val)

update_max :: max_e -> state -> state
update_max <max_temp> OldMax :=
    output(<day_max_temp, OldMax);
    return 0
```

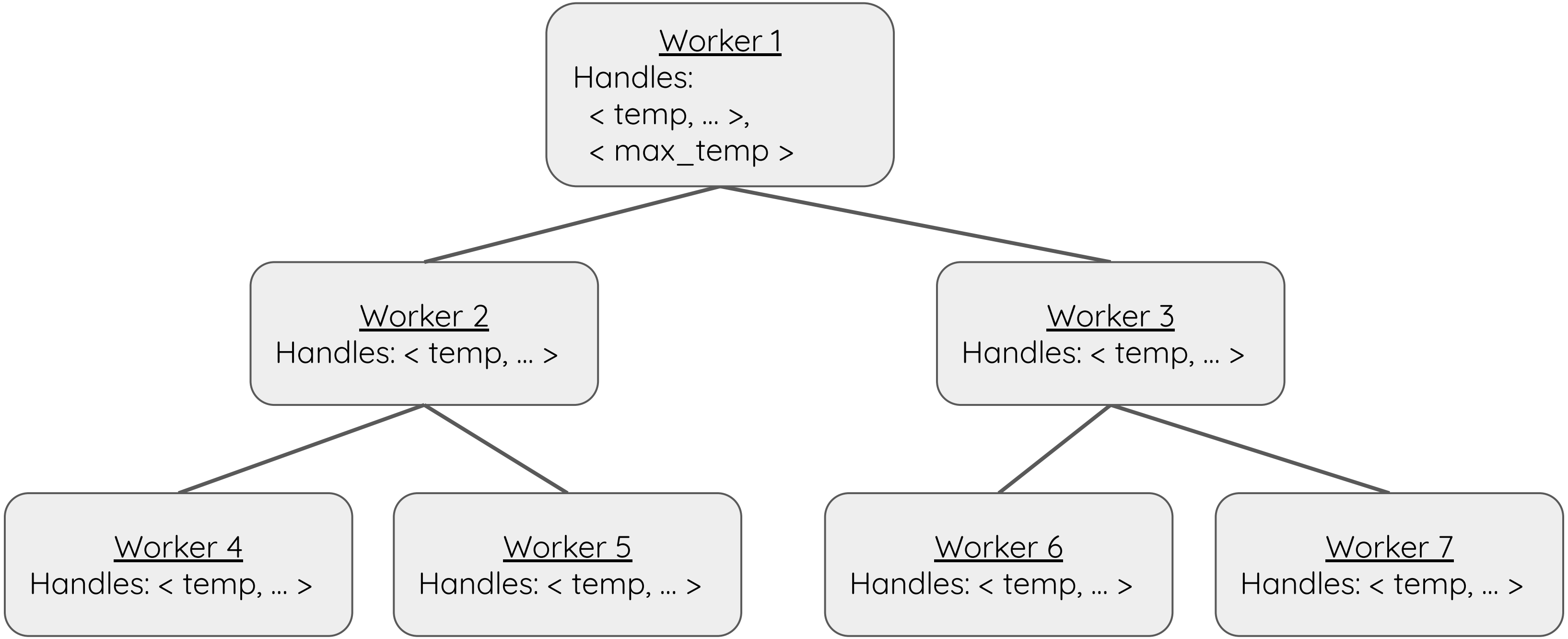
```
fork :: state -> (state * state)
fork Max :=
    return (Max, Max)

join :: (state * state) -> state
join Max1 Max2 :=
    return max(Max1, Max2)
```



Dependency Relation

Specification

Fork - Join



Worker 1
Handles:
< temp, … >,
< max_temp >

Worker 2
Handles: < temp, … >

Worker 3
Handles: < temp, … >

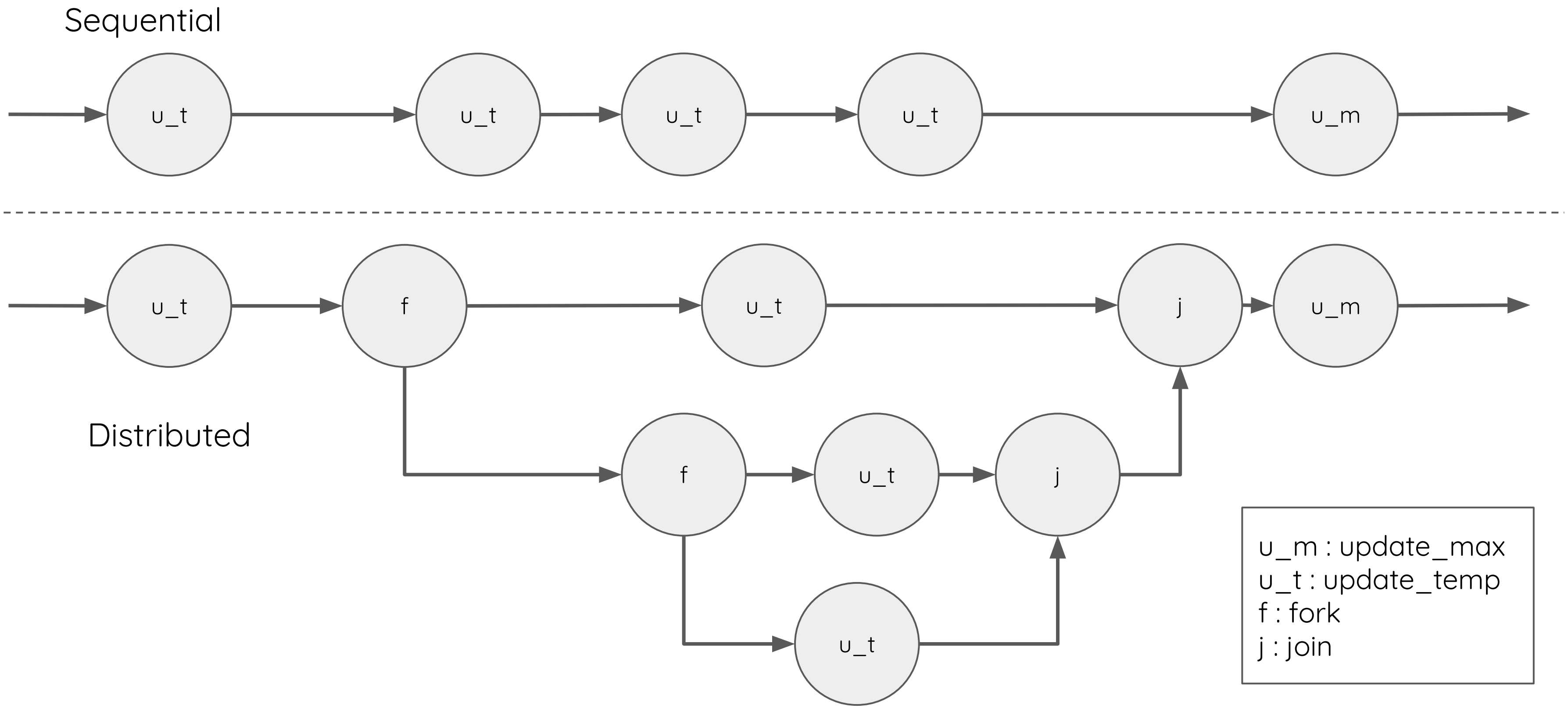
Worker 4
Handles: < temp, … >

Worker 5
Handles: < temp, … >

Worker 6
Handles: < temp, … >

Worker 7
Handles: < temp, … >

Correctness

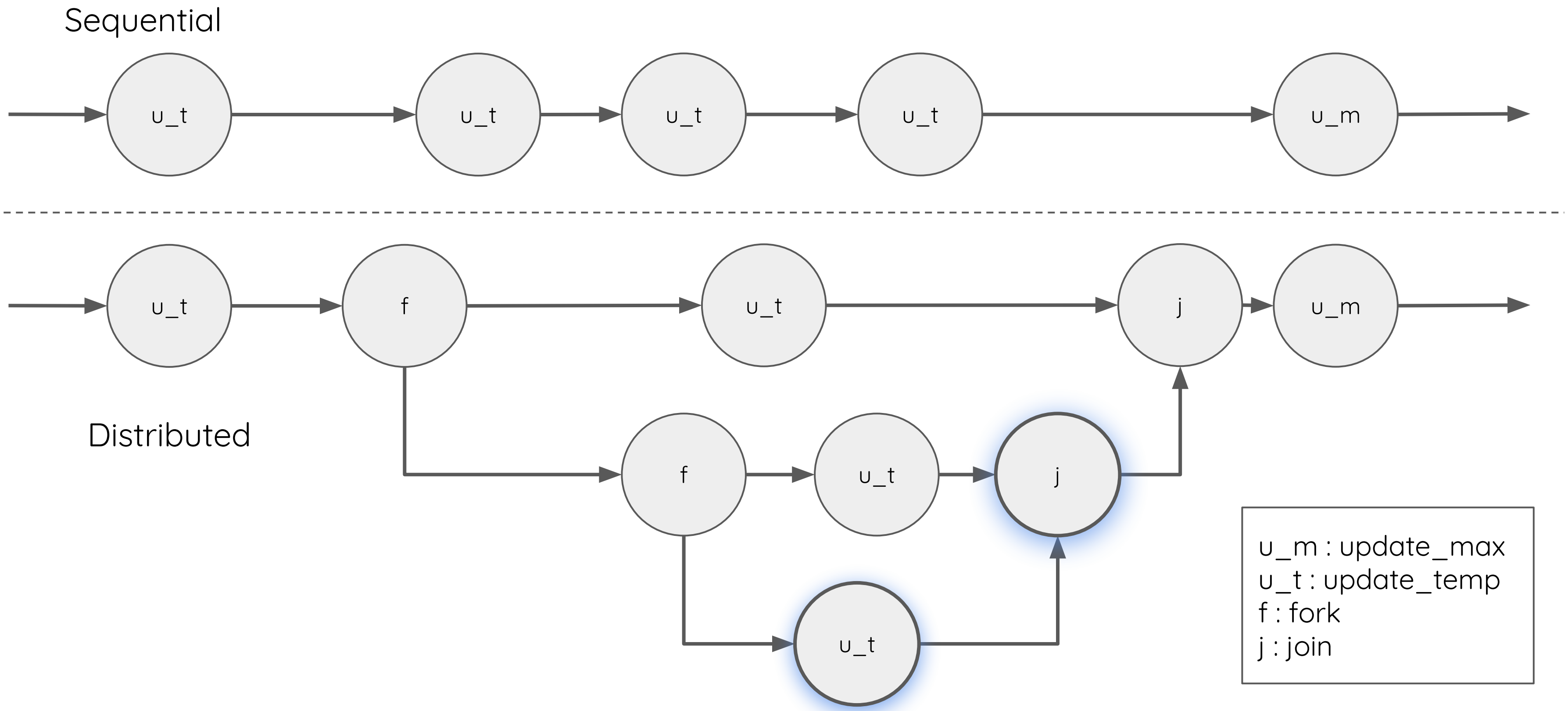


Sequential

Distributed

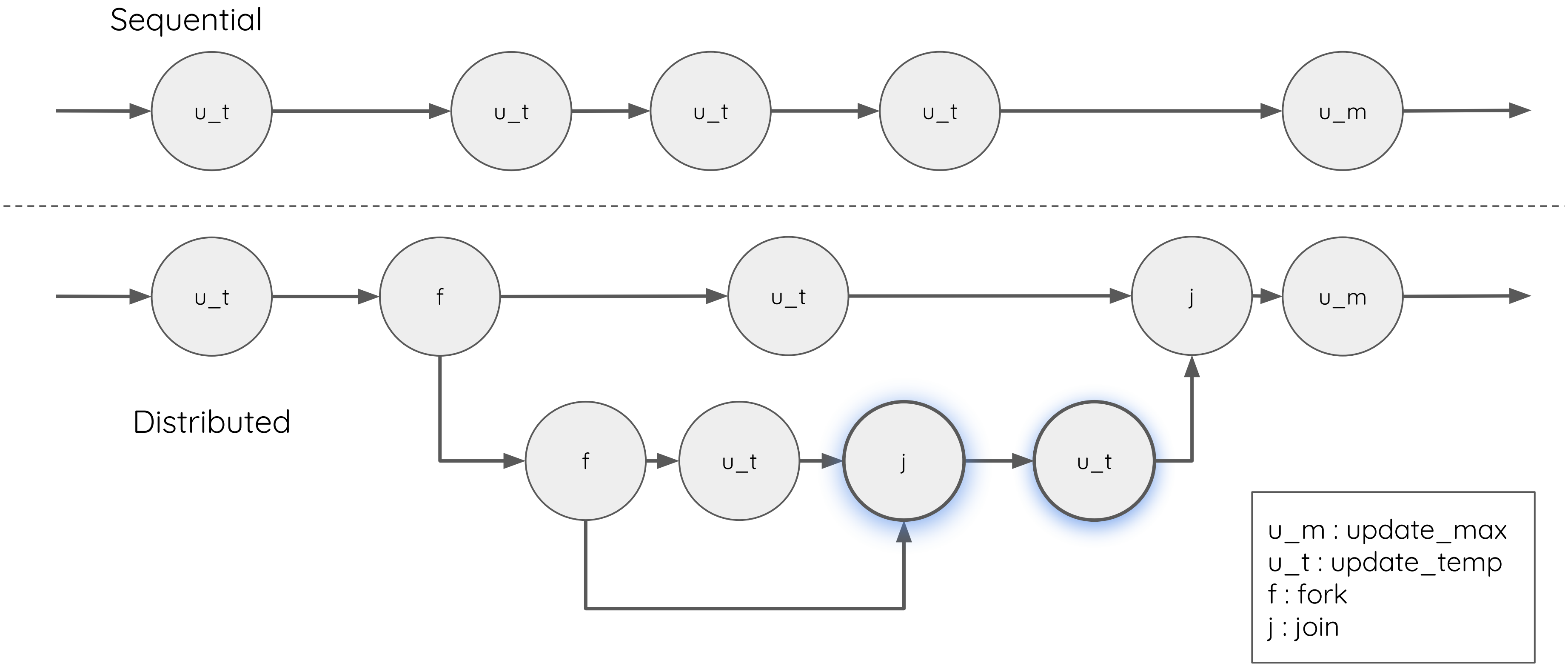
u_m : update_max
u_t : update_temp
f : fork
j : join

Correctness



u_m : update_max
u_t : update_temp
f : fork
j : join

Correctness

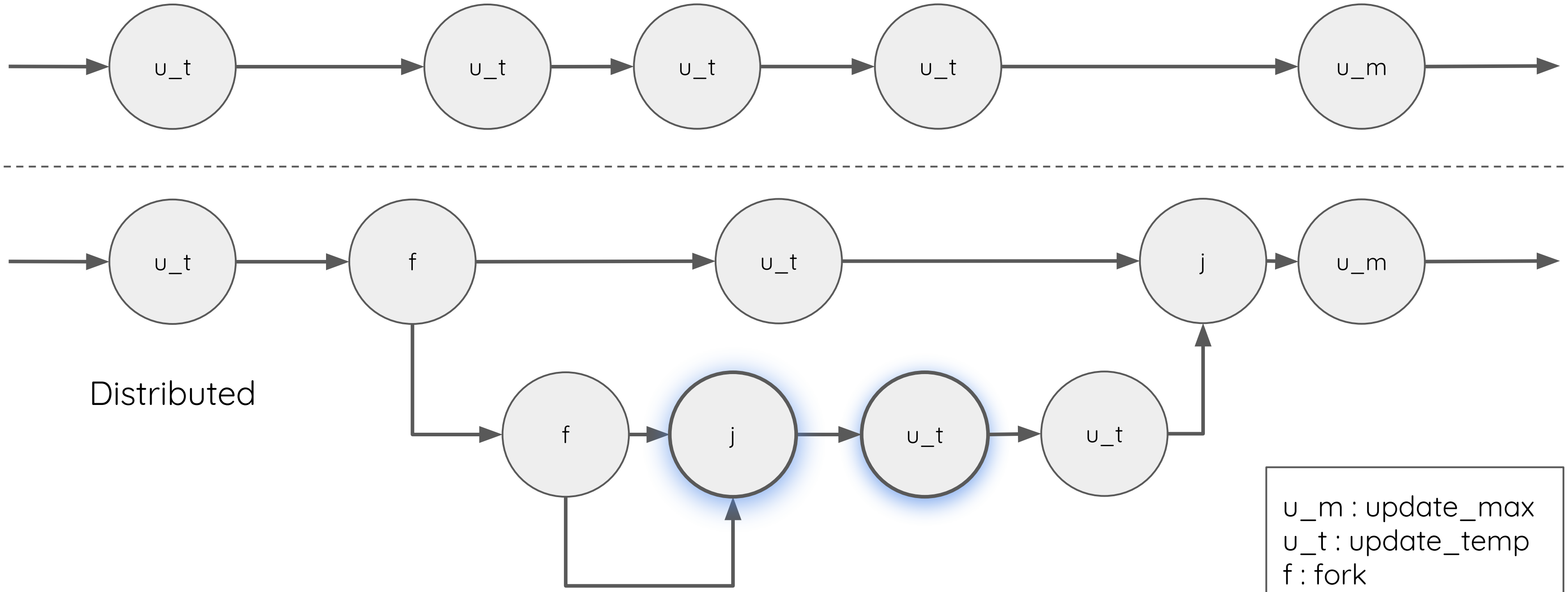


u_m : update_max
u_t : update_temp
f : fork
j : join

Correctness

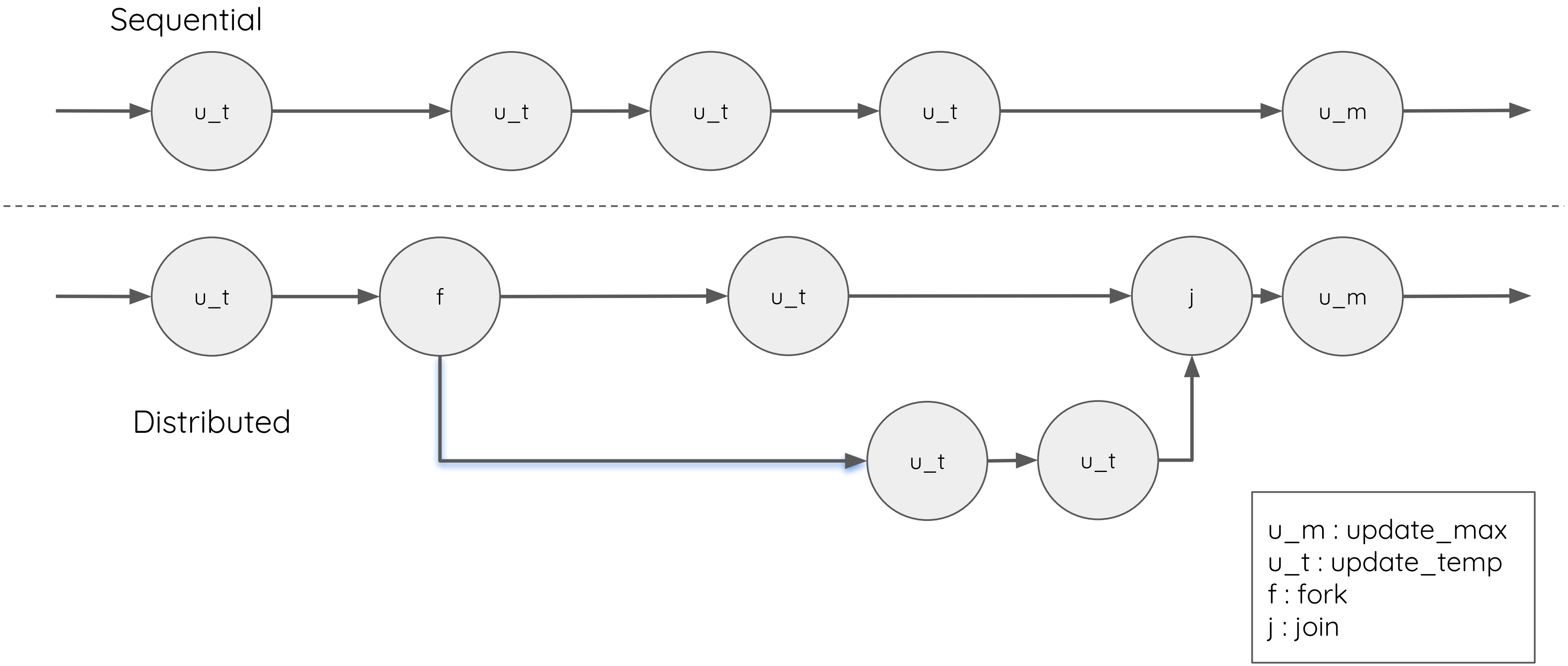
Sequential

Distributed



u_m : update_max
u_t : update_temp
f : fork
j : join

Correctness

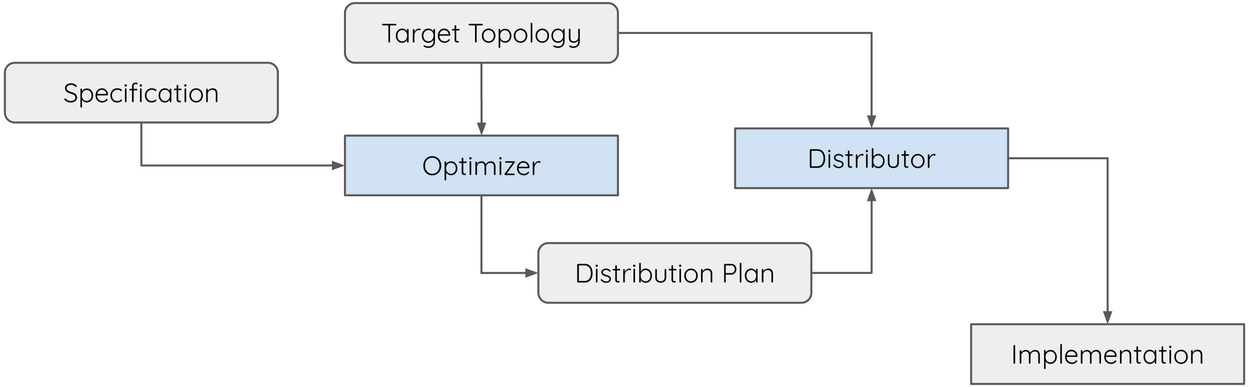


u_m : update_max
u_t : update_temp
f : fork
j : join

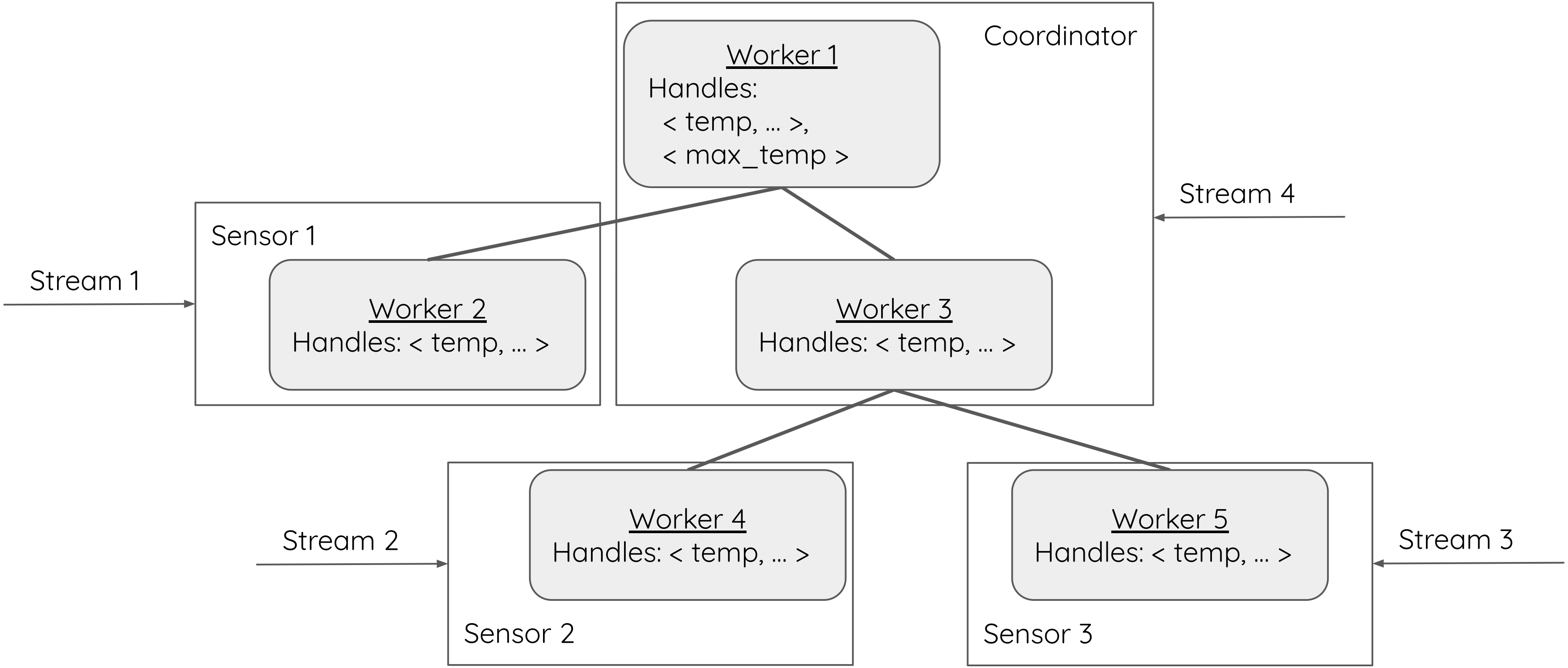
Small Recap

- Streams are partial orders
- Dependency relation encapsulates ordering requirements
- Forks-joins as distribution primitives
- Provably correct distribution

Automated Distribution



Automated Distribution



Evaluation

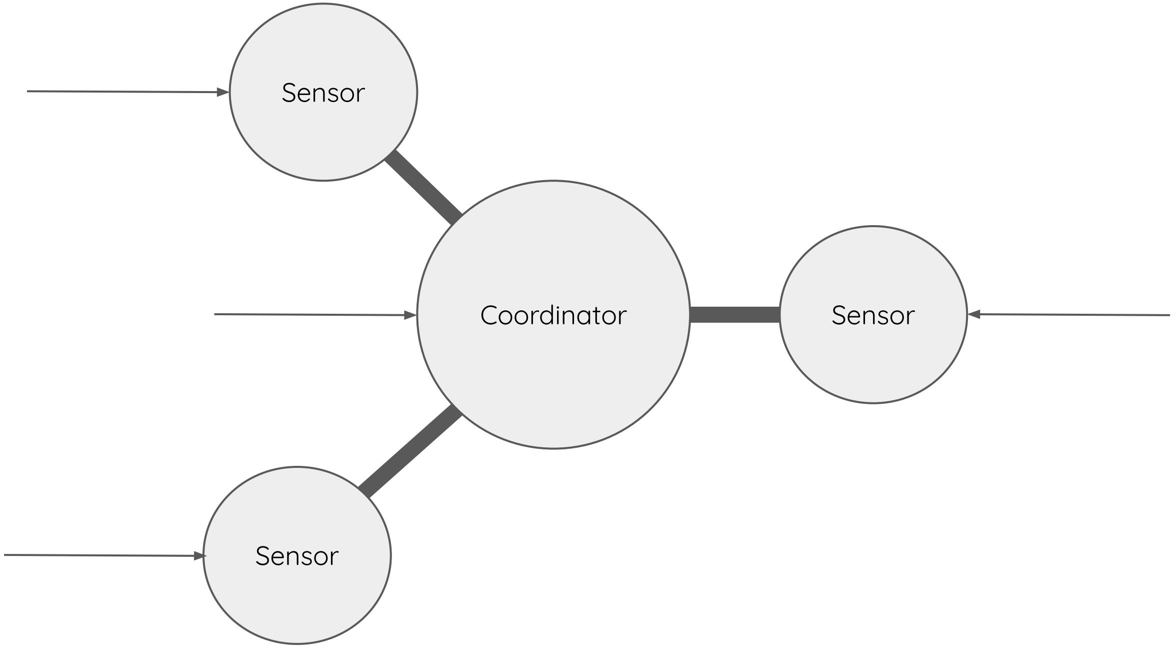
Implementation

Public on github: https://github.com/angelhof/flumina

Setup

Single machine with 18 cores

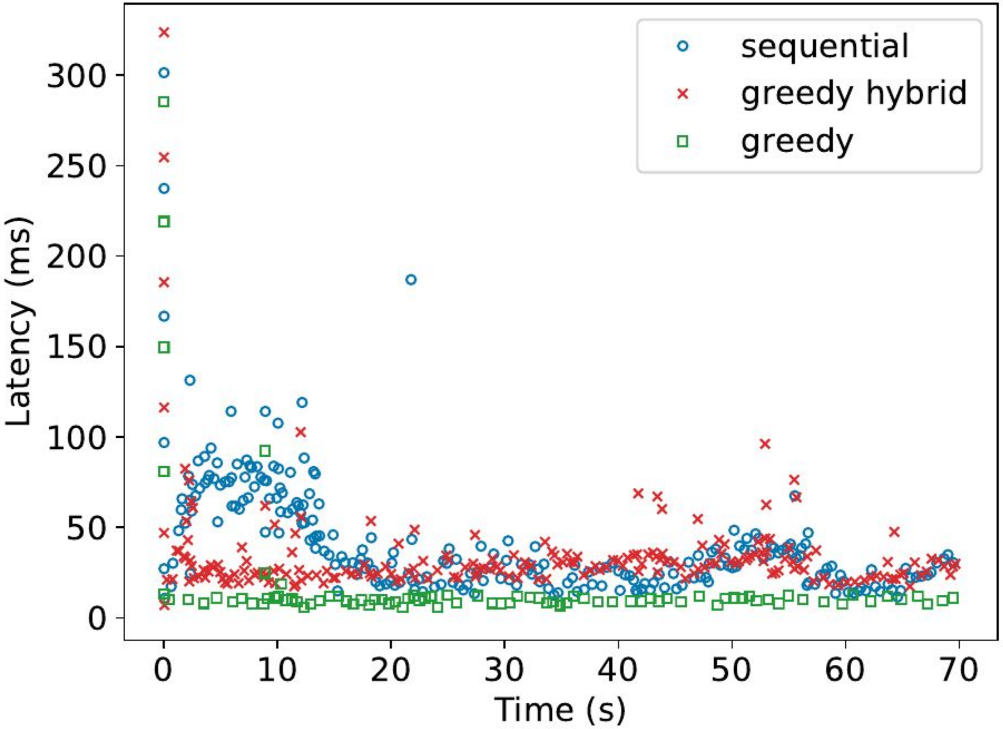
Microbenchmarks



```
state  := int // max temp so far
temp_e := <temp, int>

update_temp :: temp_e -> state -> state
update_temp <temp, Val> OldMax :=
    return max(OldMax, Val)
```

```
max_e := <max_temp>

update_max :: max_e -> state -> state
update_max <max_temp> OldMax :=
    output(<day_max_temp, OldMax);
    return 0
```

Optimizer Comparison -- Latency



Latency Comparison

4 sensors and 1 central node

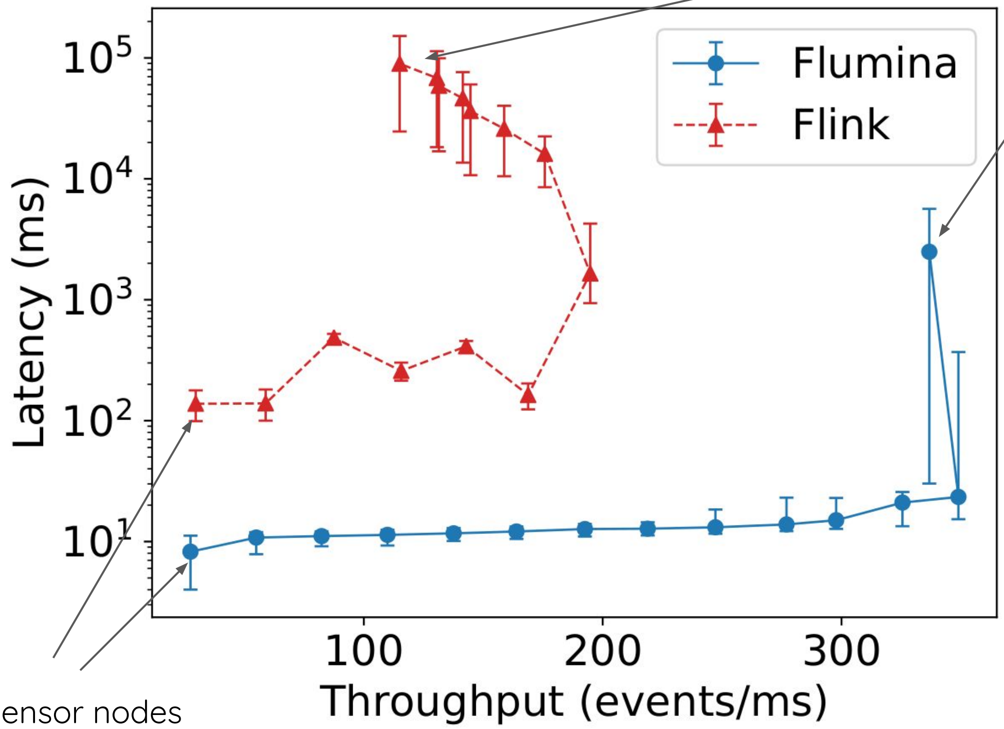
4 input temp streams (1 per sensor)

sequential: 1 centralized worker

greedy hybrid: 5 workers in central node

greedy: 5 workers (1 per node)

Flumina vs Flink -- Scaling



28 sensor nodes

2 sensor nodes

<u>Latency + Throughput</u>

1 central node and varying sensors

1 temp input stream per sensor

Case studies

- Distributed Outlier Detection
  - Sequential: 700 LoC
  - Distributed: + 50 LoC
  - Performance similar to original paper
- Energy Management
  - Sequential: 200 LoC
  - Distributed: + 60 LoC
  - Network Load: 350MB out of 29GB

Goal: Evaluate usability in complex applications

Conclusion

Conclusions

- Programming model
  - View input streams as partial orders
  - This enables correct distribution of more streaming computations
- Distributed computations in Flumina
  - Requires small effort to specify
  - Can be implemented efficiently in an automatic way

Future Work

- Verification of Flumina code
- Synthesis of fork-join pairs
- Online re-distribution
- High level query language
- Privacy

Thank you :)

Public on github: https://github.com/angelhof/flumina