Distributed REScala:
un update algorithm for distributed Reactive Programming

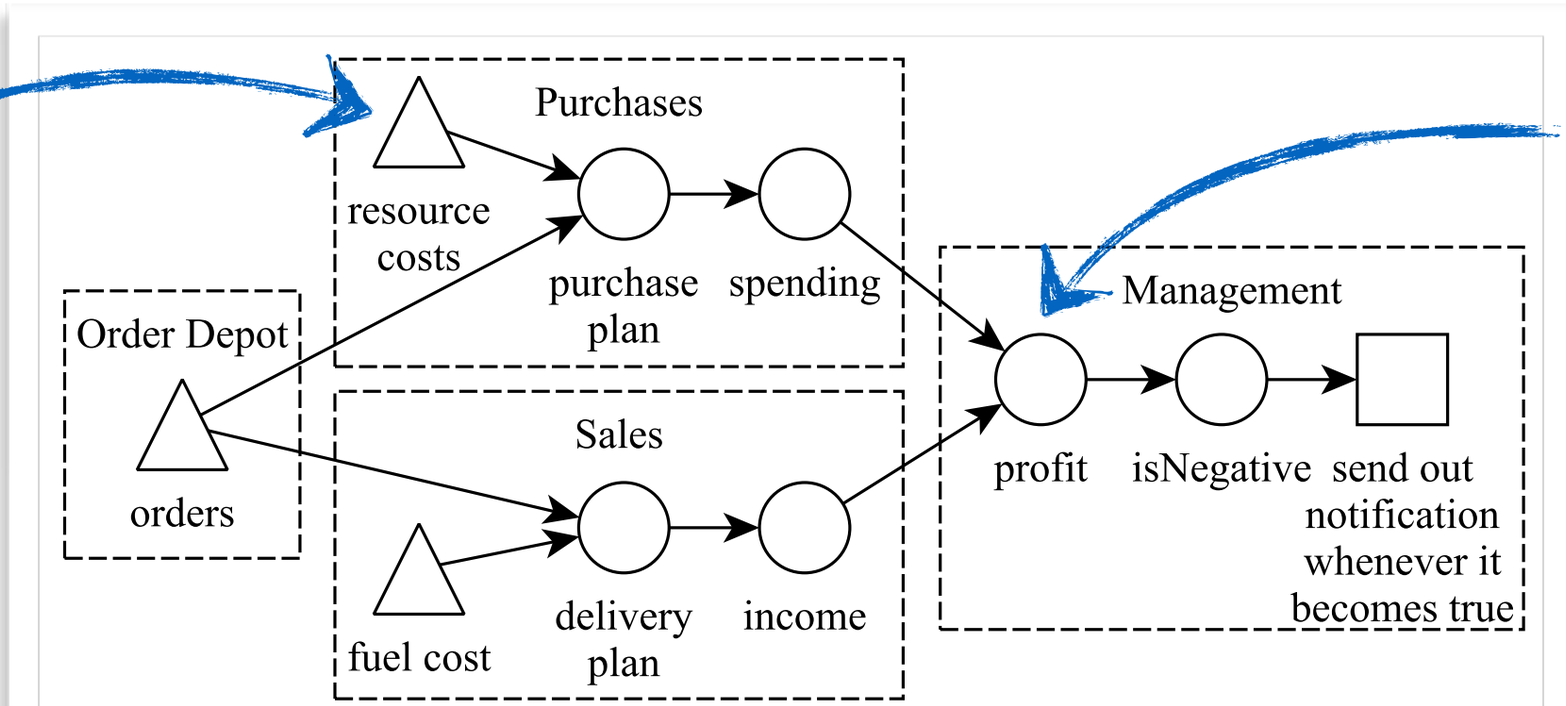
(José Proença @ 1 April)

OOPSLA'14 - Joscha Drechsler, Guido Salvaneschi,
Ragnar Mogk, and Mira Mezini
(TU-Darmstad)

Context and Motivation

source node

dependent node



Lack of generic distributed algorithms
- only client-side, no glitch-freedom, or cannot be distributed

Distributed RP vs. observers + remote obj.

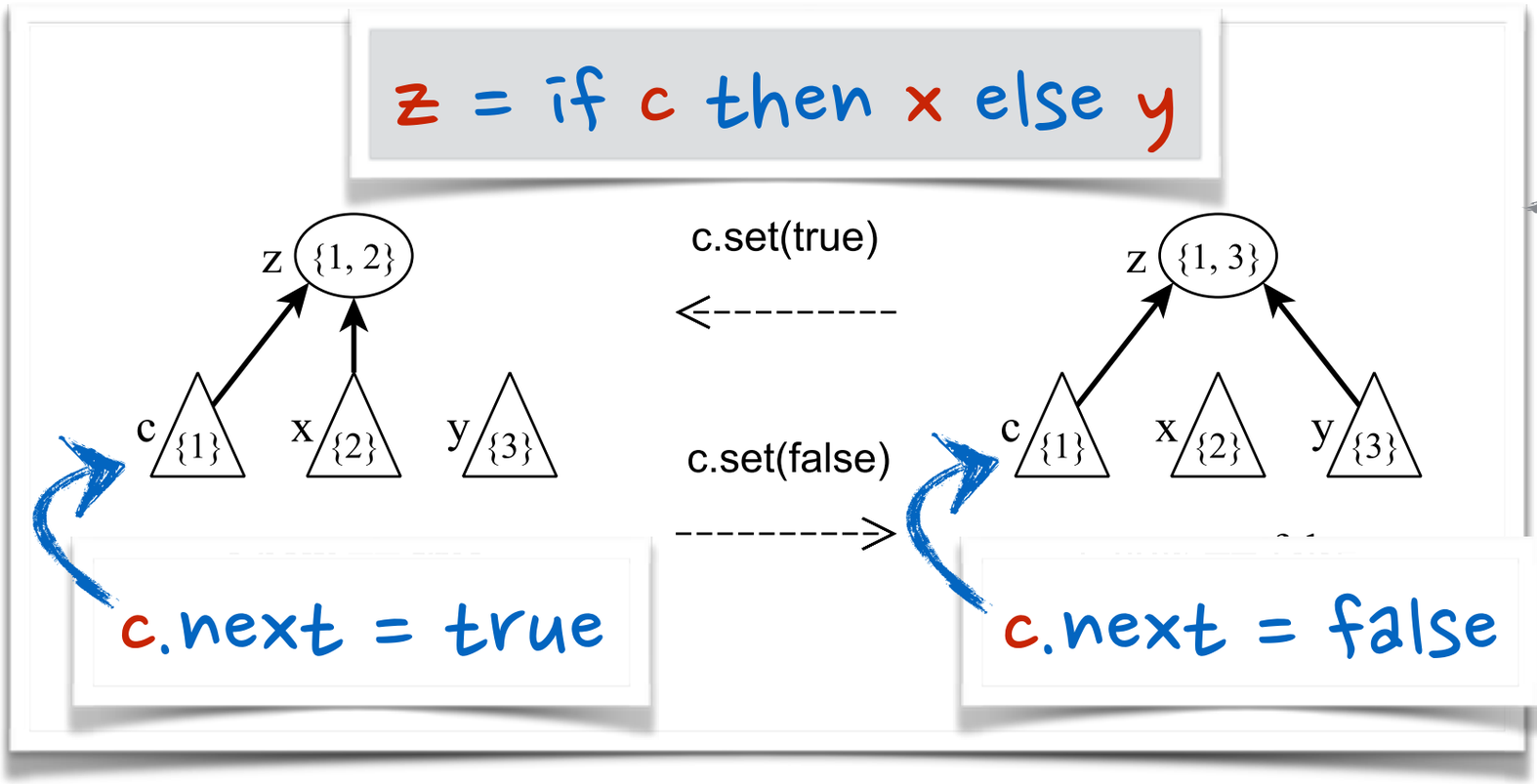
Contributions

Algorithms for distributed glitch-freedom:

- **Scala.React** — Topological sorting + priority queue
- **Scala.Rx** — Parallel propagator version
- **ELM** — Decentralised flooding
- **SID-UP** — NEW: send & store more data

Evaluation

Dynamic dependencies



Conditionals

High-order reactives

```
getPerson()
 .getNames()
 .selectFst()
```

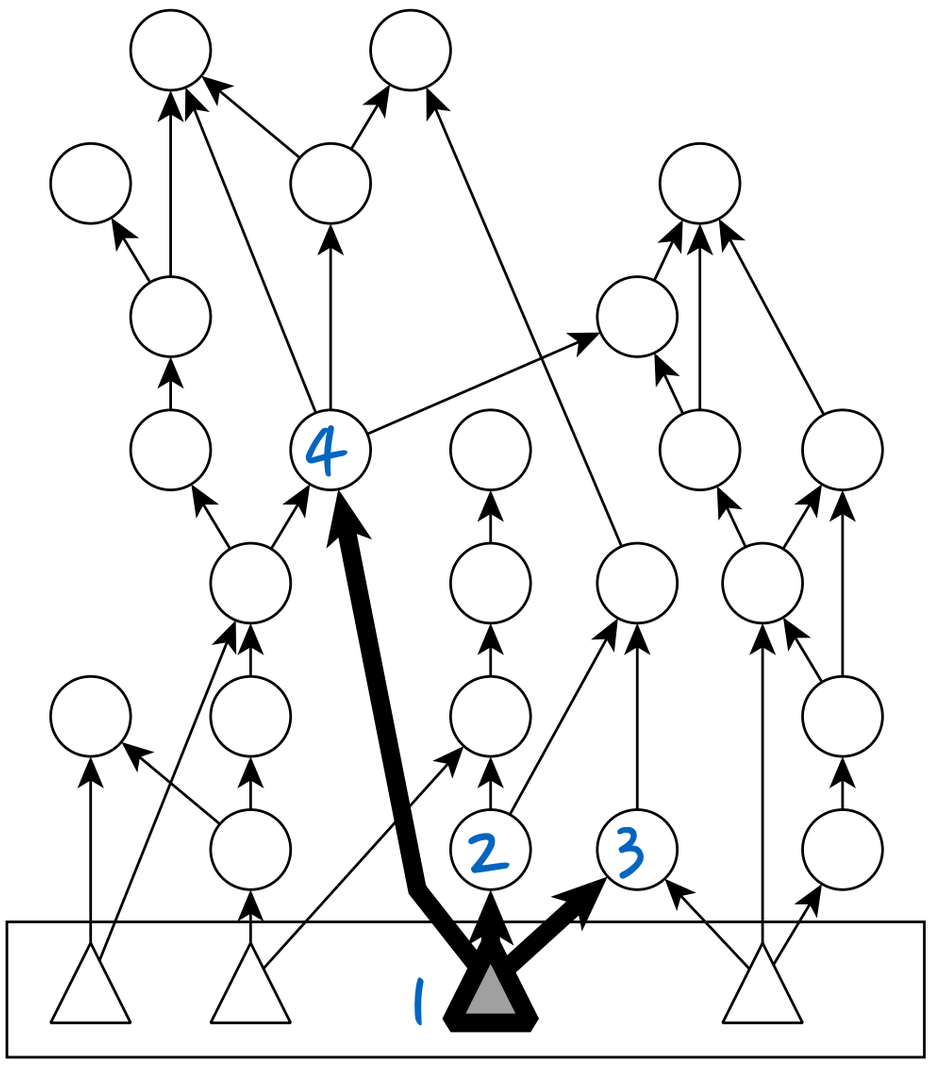
→ Stream[Person]

→ Stream[Stream[Name]]

→ Stream[Name]

Scala.React

Topological sorting + priority queue

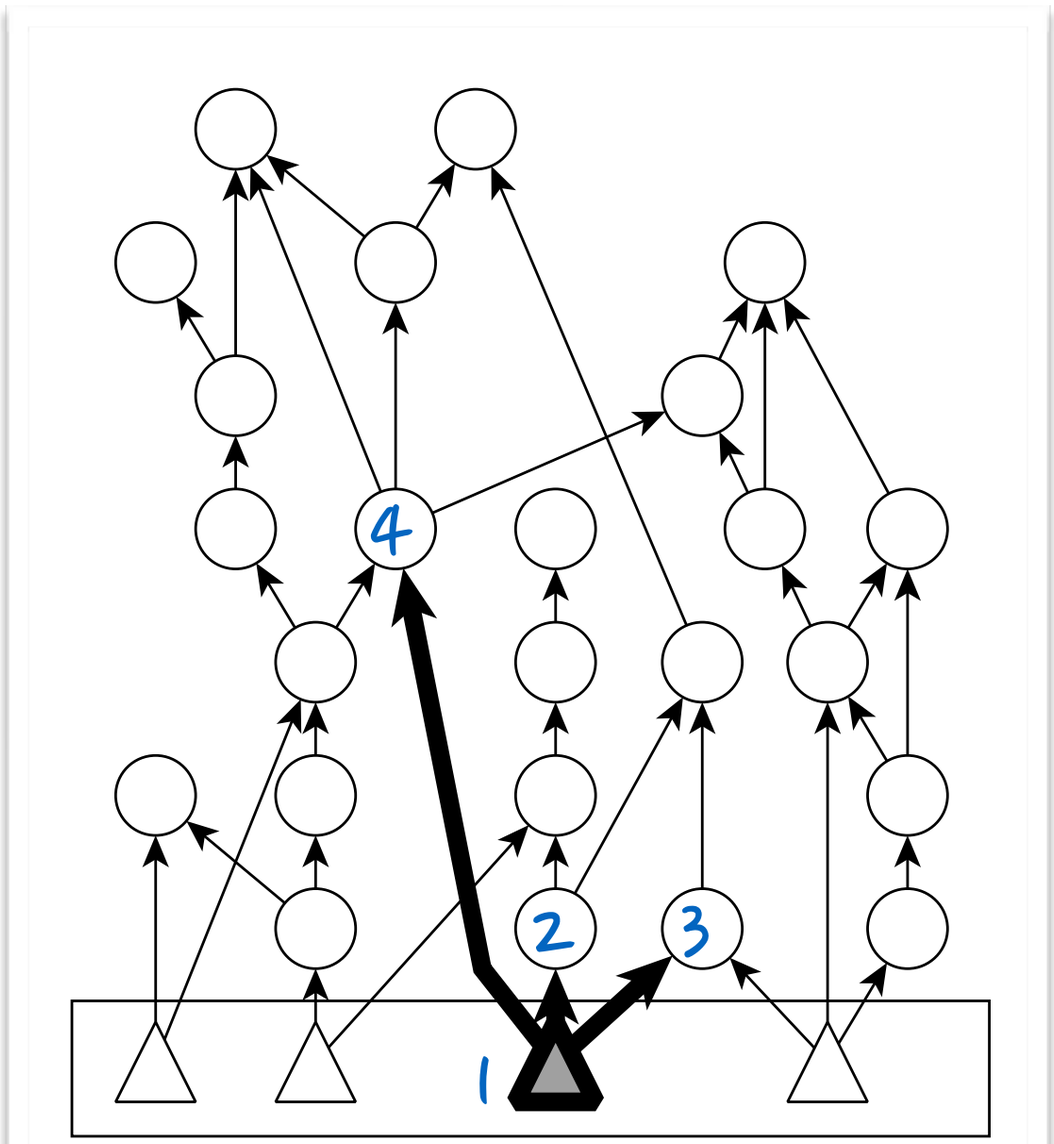


Priority queue

1

Scala.React

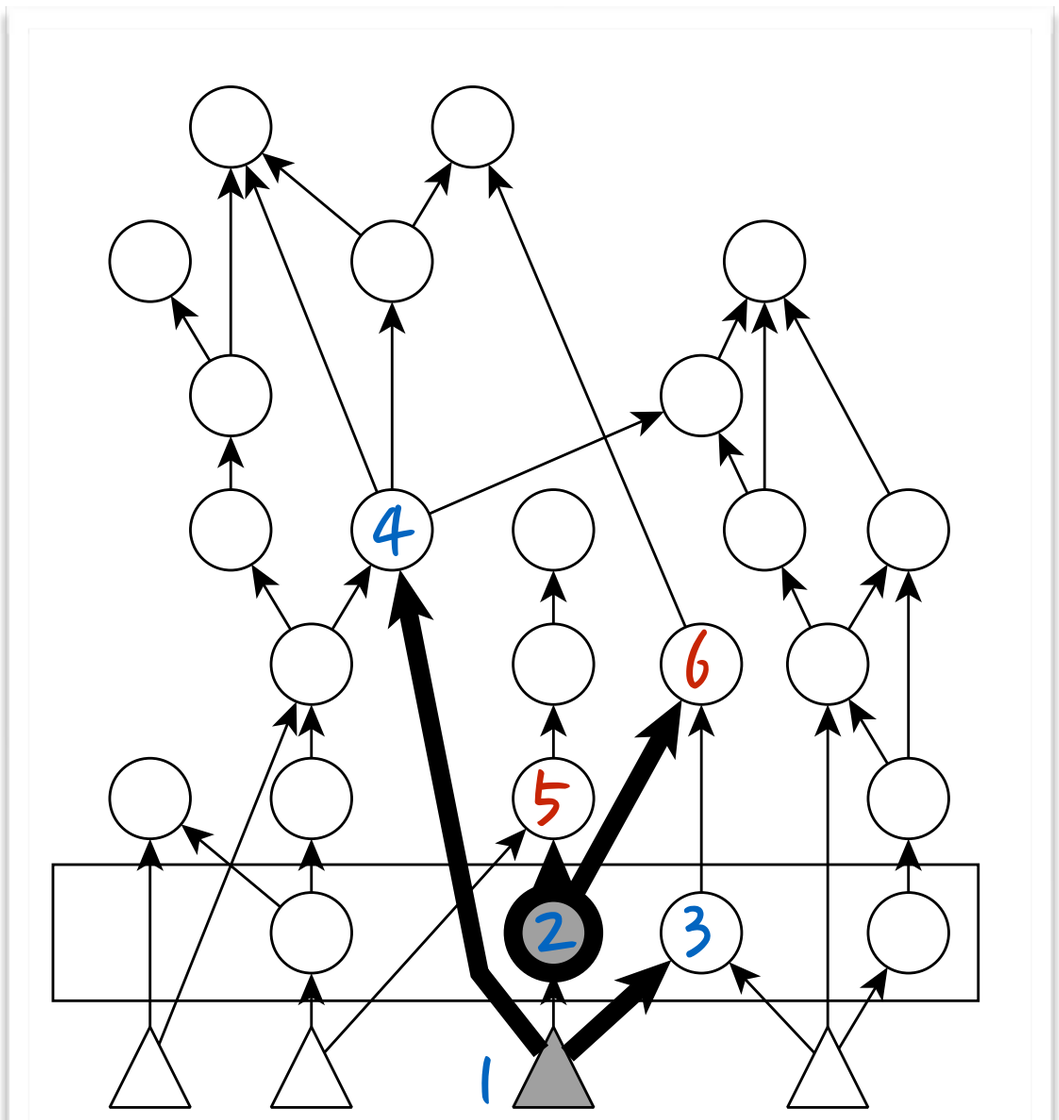
Topological sorting + priority queue

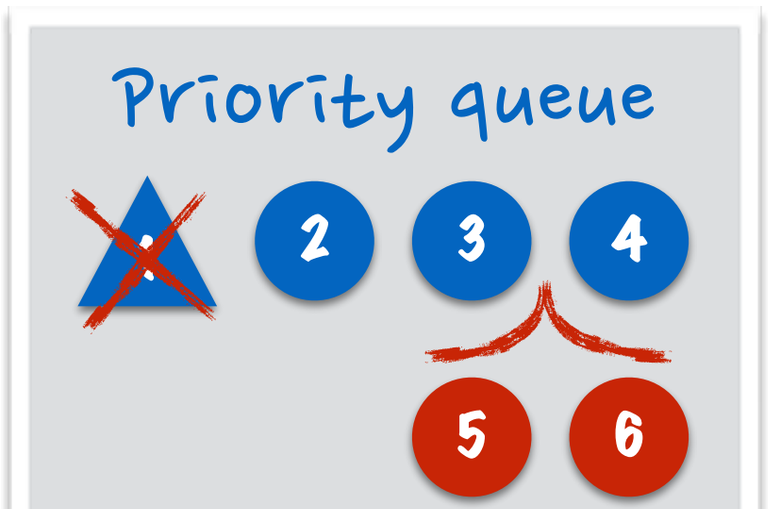


Priority queue

Scala.React

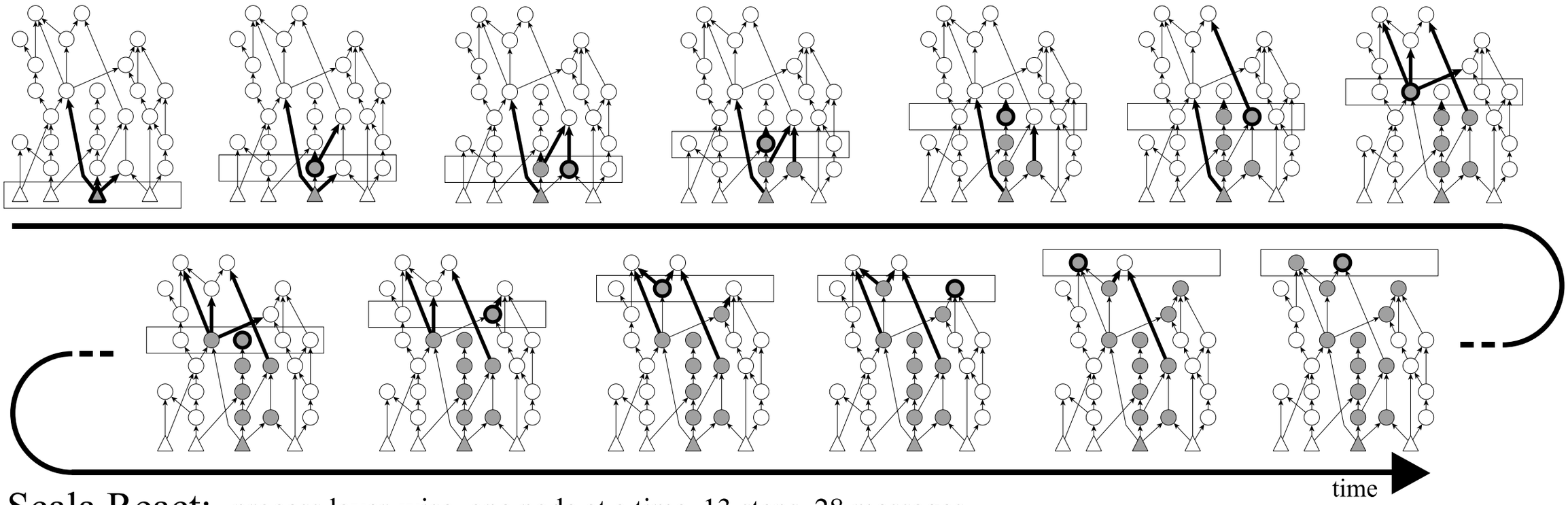
Topological sorting + priority queue





Scala.React

Topological sorting + priority queue



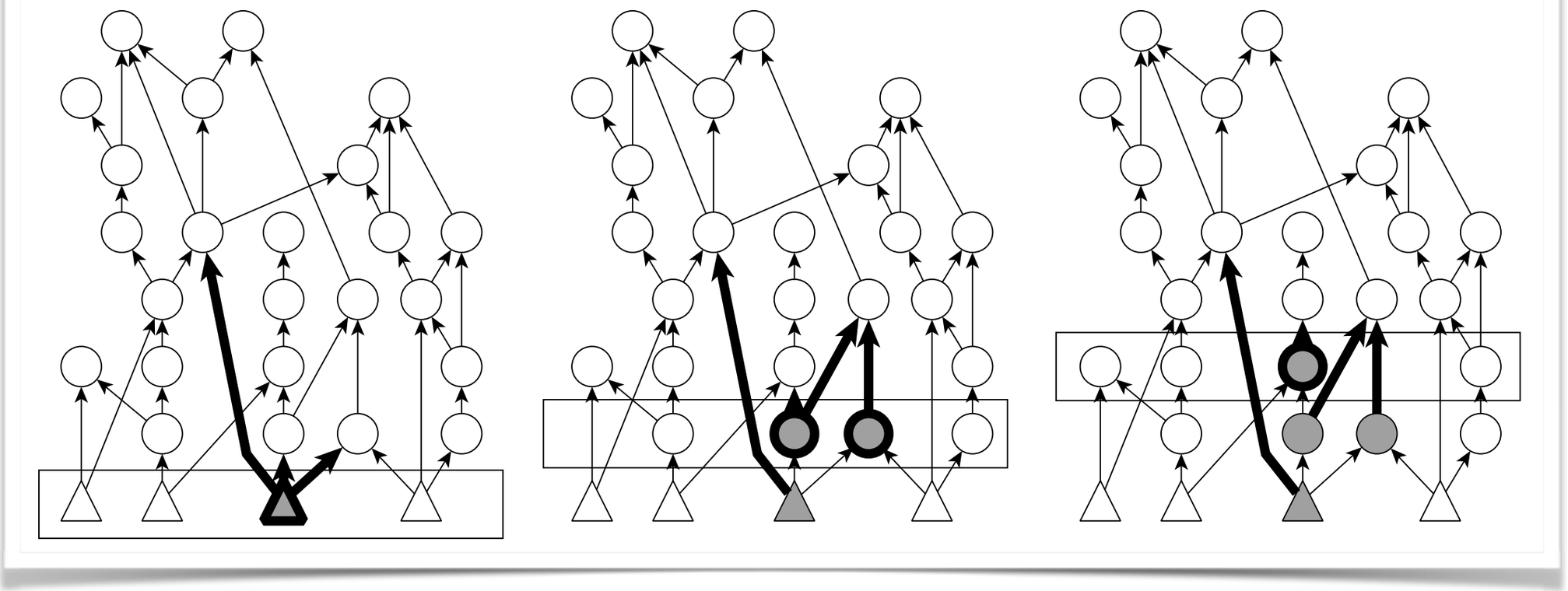
Scala.React: process layer-wise, one node at a time. 13 steps. 28 messages.

13 steps

28 messages

Scala.Rx

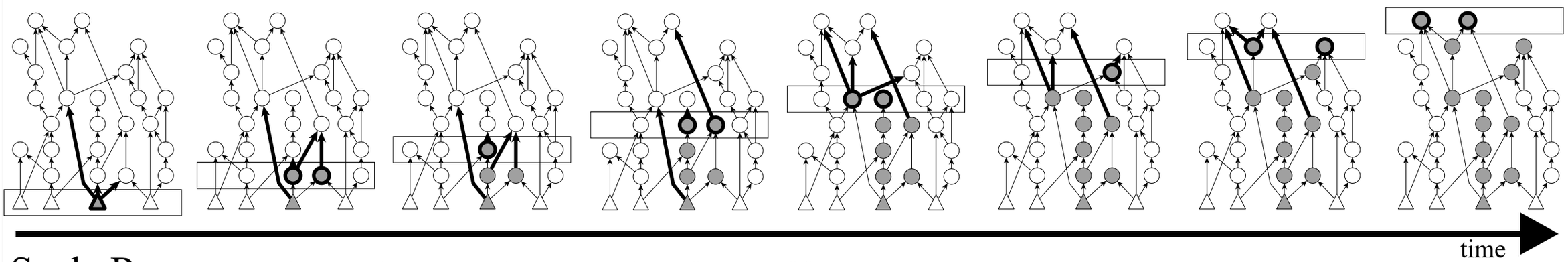
Parallel propagator version



Entire layer at a time!

Scala.Rx

Parallel propagator version



Scala.Rx: process layer-wise, entire layer at a time. 8 steps. 28 messages.

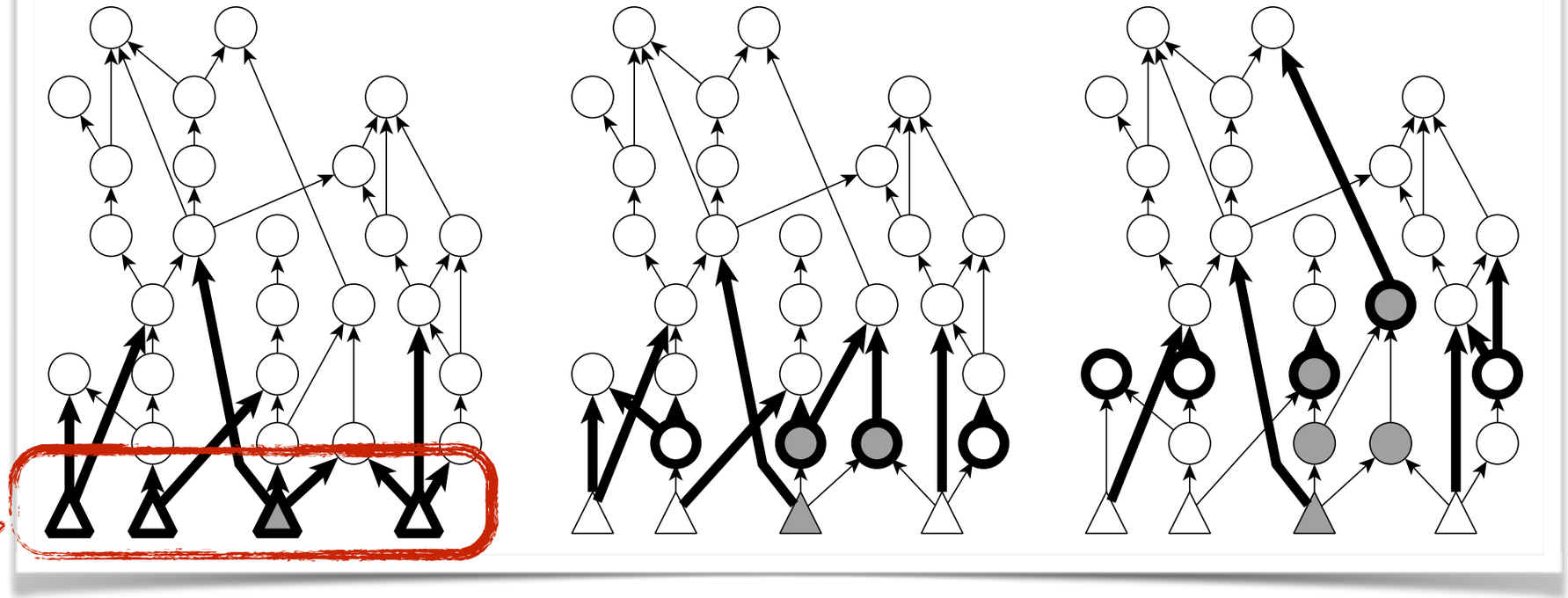
8 steps

~~13 steps~~

28 messages

ELM

Decentralised flooding

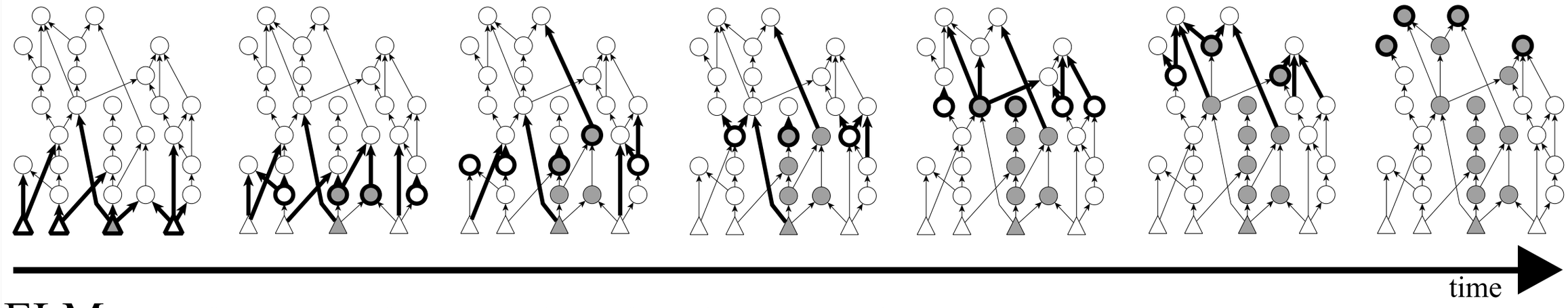


Round initiator

Each node sends:
- "change" pulse, or
- "no-change" pulse

ELM

Decentralised flooding



ELM: wait for message from every incoming edge. 7 steps. 37 messages.

ELM

Decentralised flooding

Supports pipelining

BUT

No high-order reactives

ELM^S

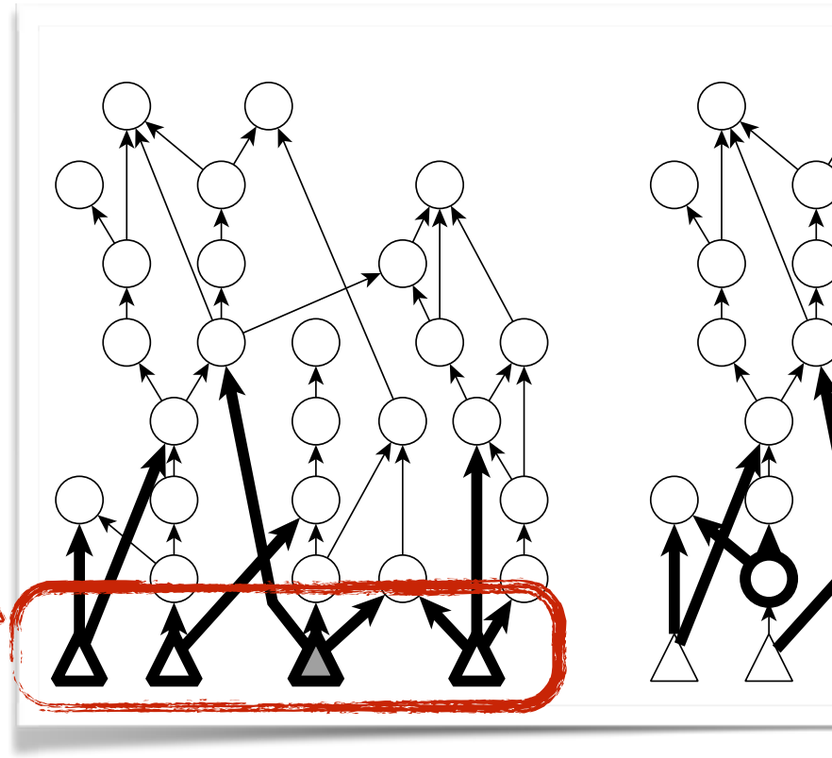
Supports HOR

BUT

No pipelining

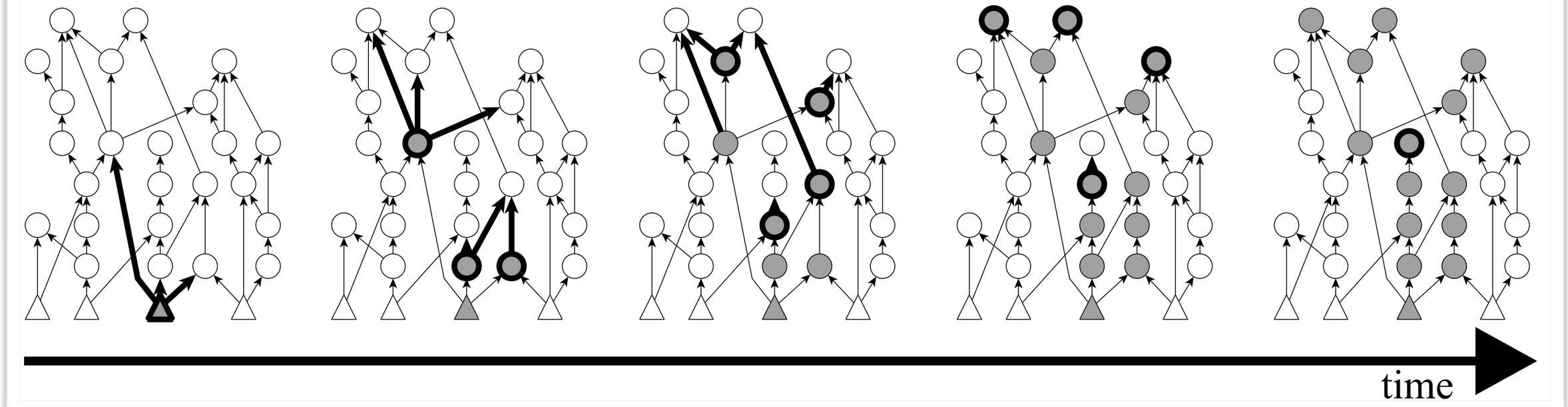
Round initiator

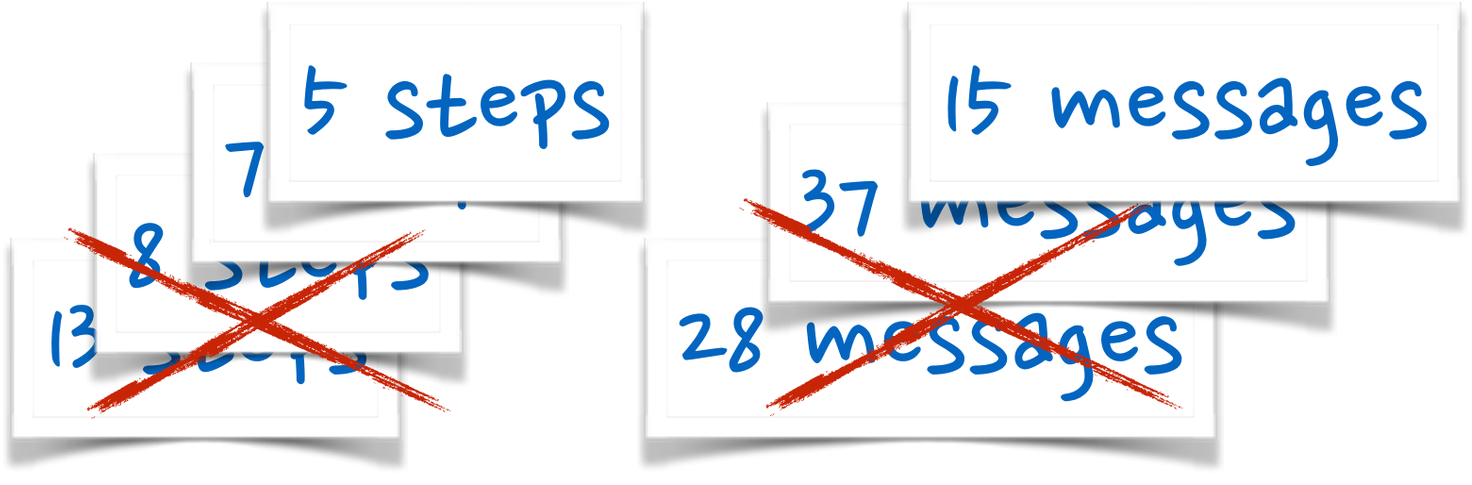
blocks until the end of the round



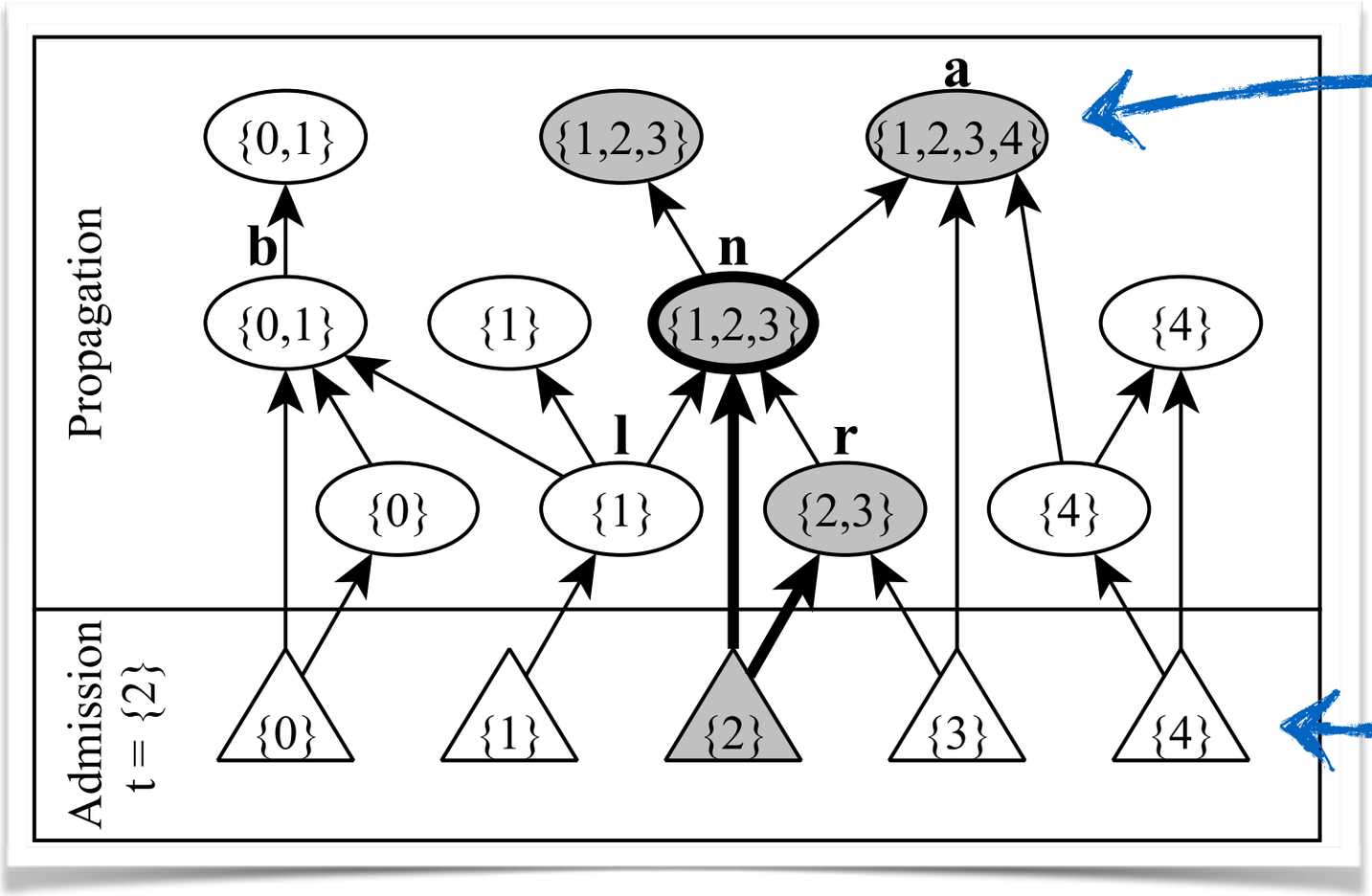
SID-UP

New algorithm





SID-UP



know dependent sources

unique ID's

Node states:

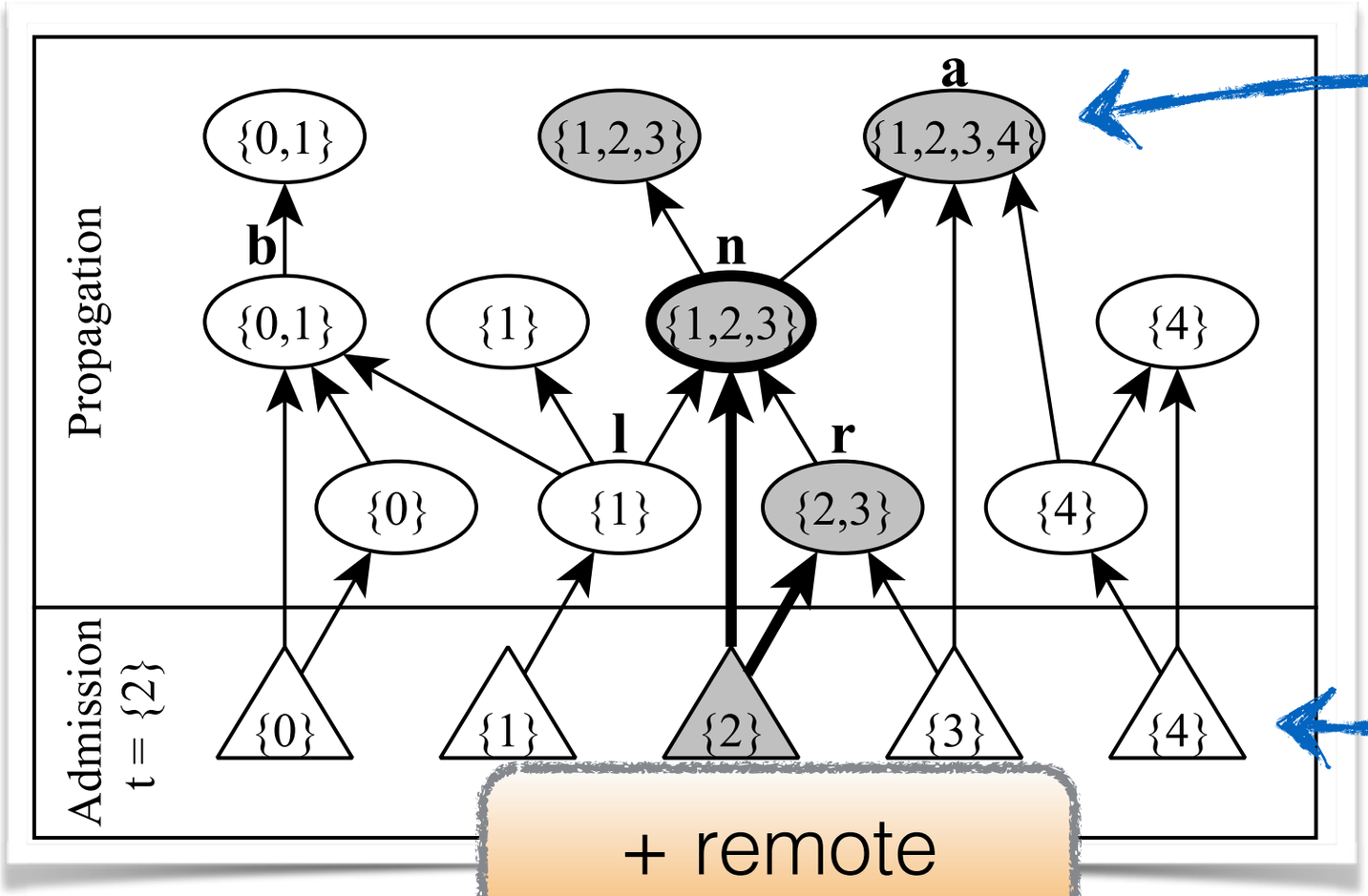
pending (not pulsed),

changed, or unchanged

When evaluated, each node sends:

- changed + changed sources, or
- unchanged + changed sources

SID-UP



know dependent sources

unique ID's

+ remote dependencies

Node state:

pending (not pulsed), changed, or unchanged

When evaluated, each node sends:

- changed + changed sources, or
- unchanged + changed sources

SID-UP

glitch-freedom

Less messages

No unbounded one-to-many communication

Support for dynamic dependencies

No coordinator when propagating

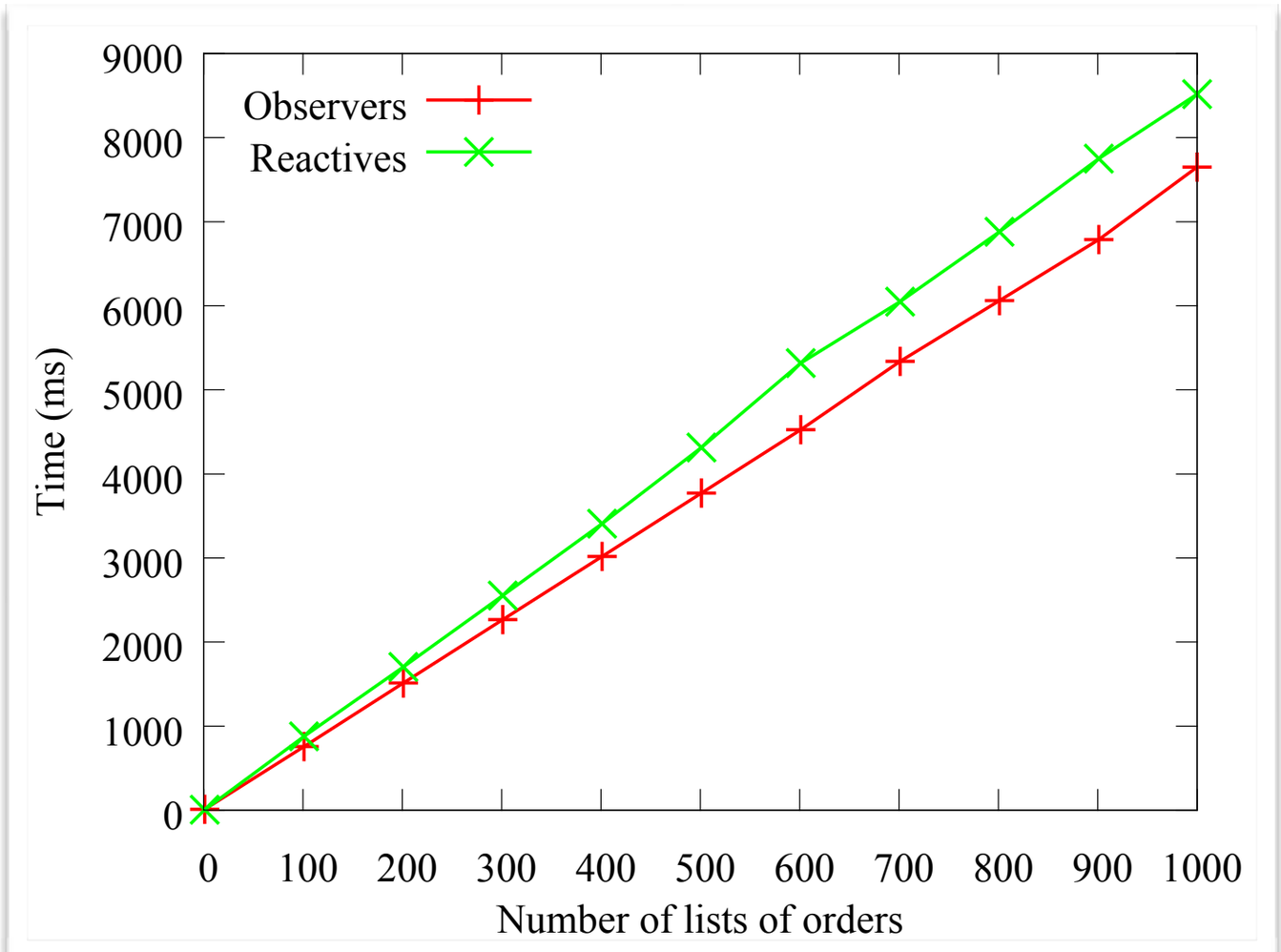
No pipelining

Faster than Scala.*

As fast as ELM

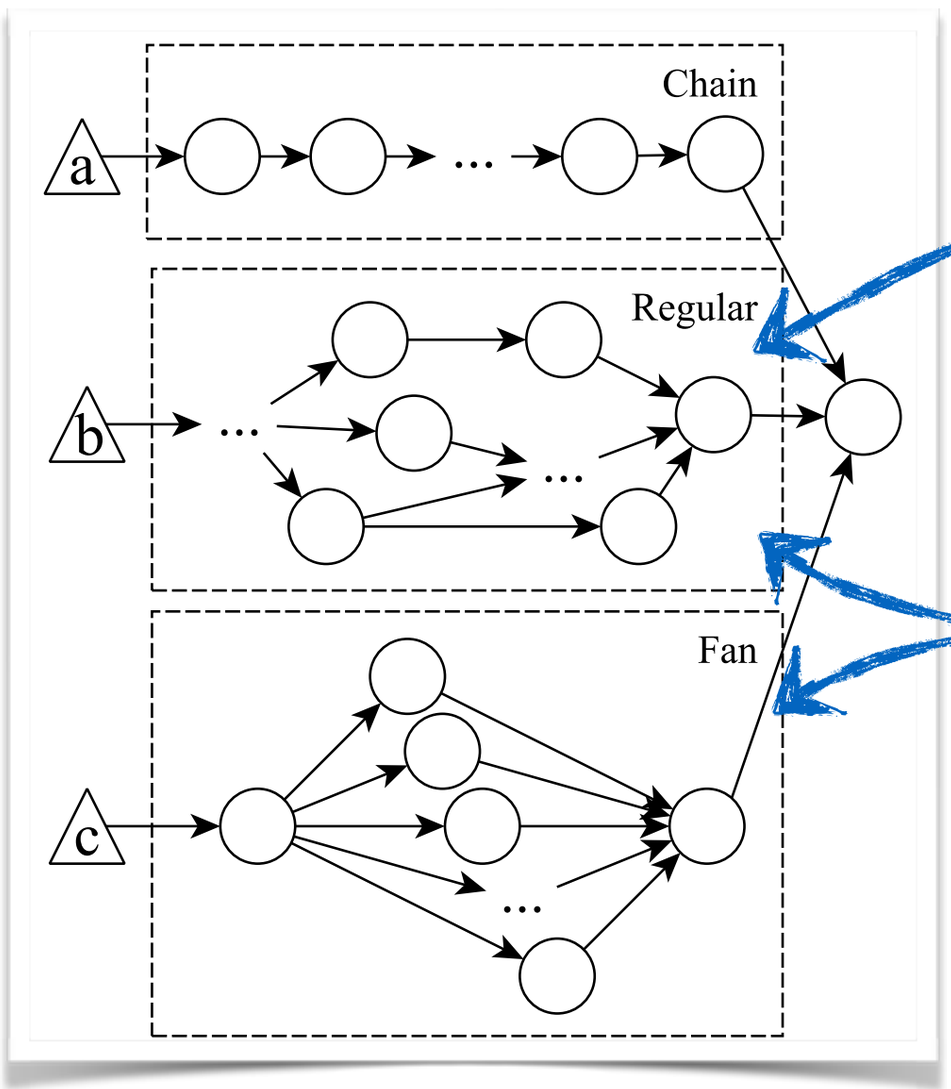
Finite & acyclic dep. graphs

Performance cost



only ± 10% slower than observers + manual glitch control

Benchmarks



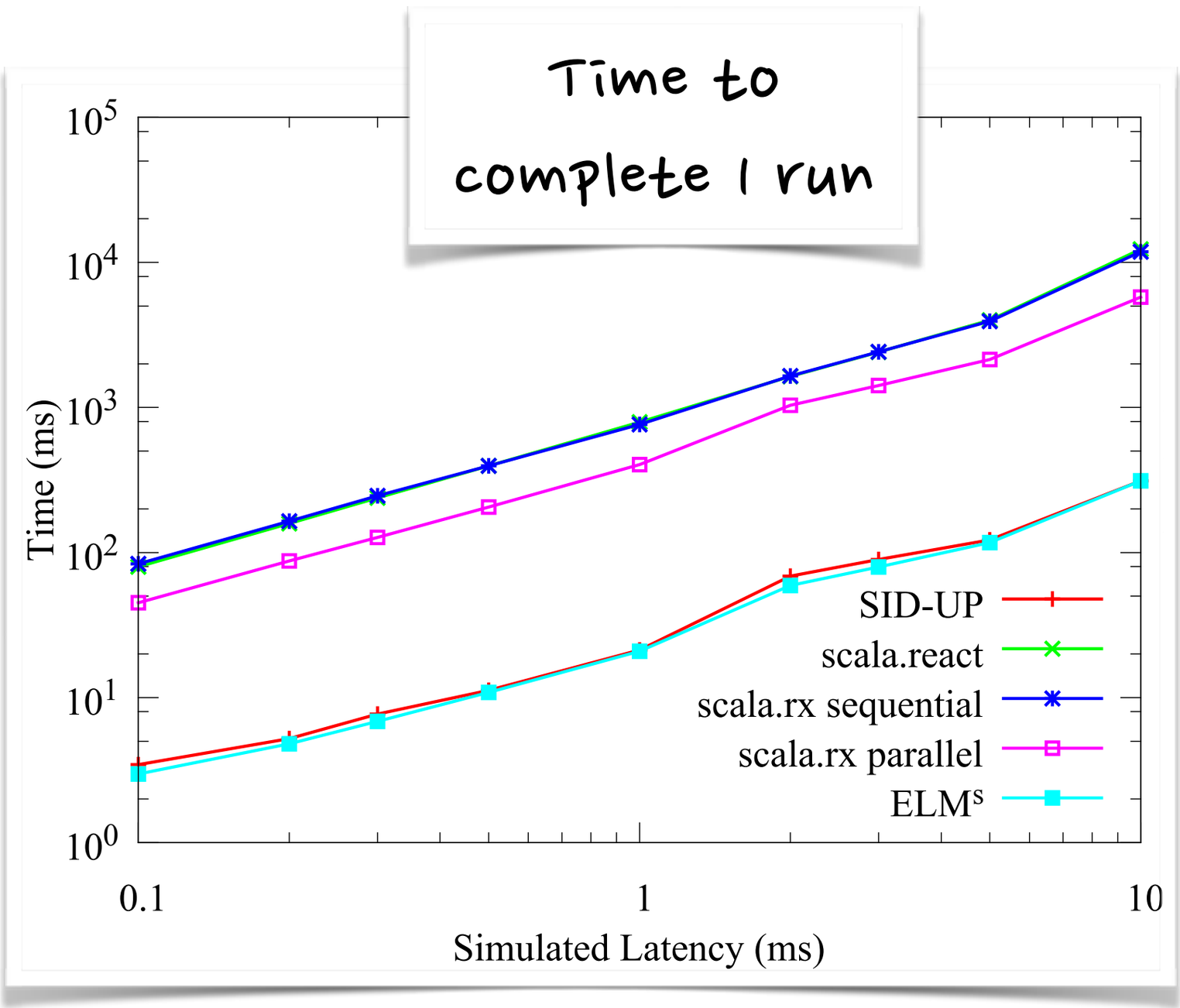
based on 20 previous case-studies

some nodes do not produce changes

Simulated network latency

3 + 25*3 + 1

Benchmarks

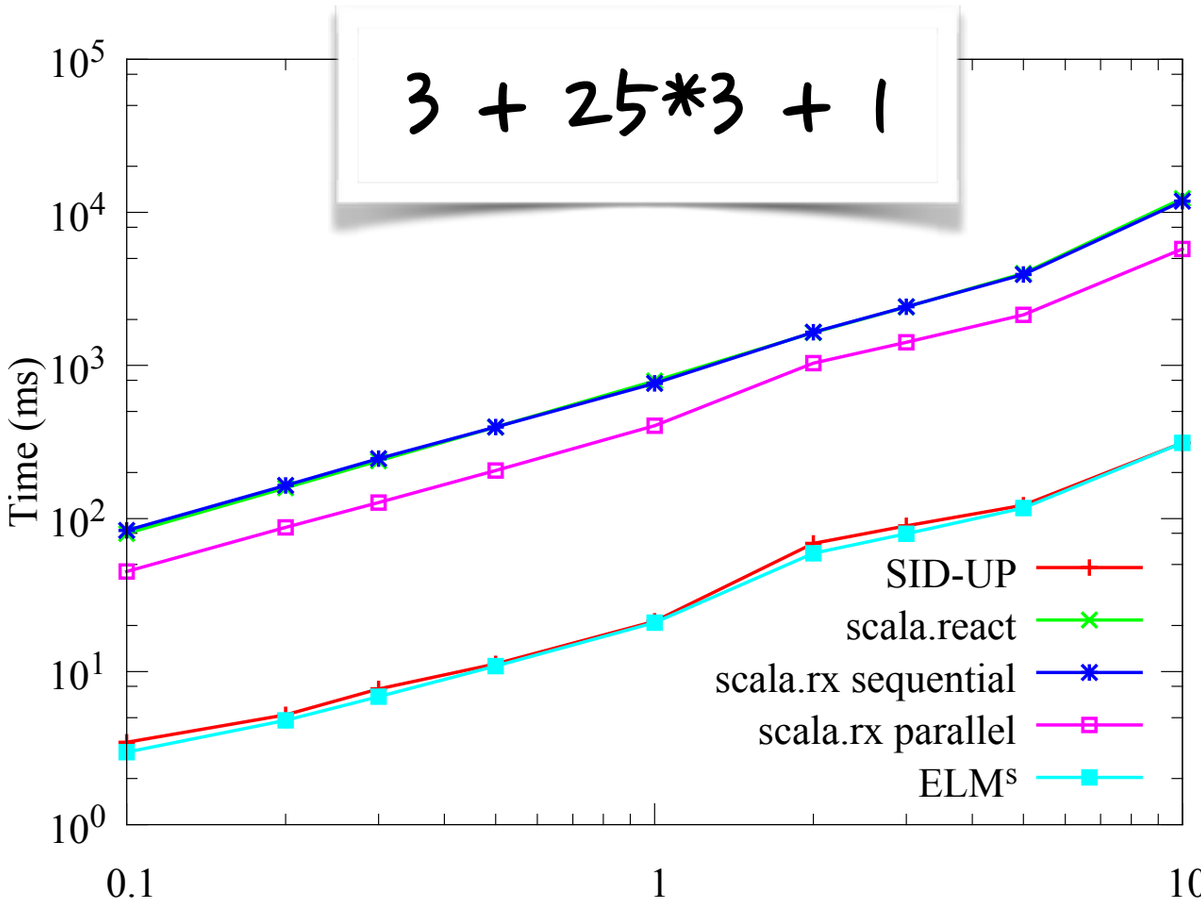


not ELM

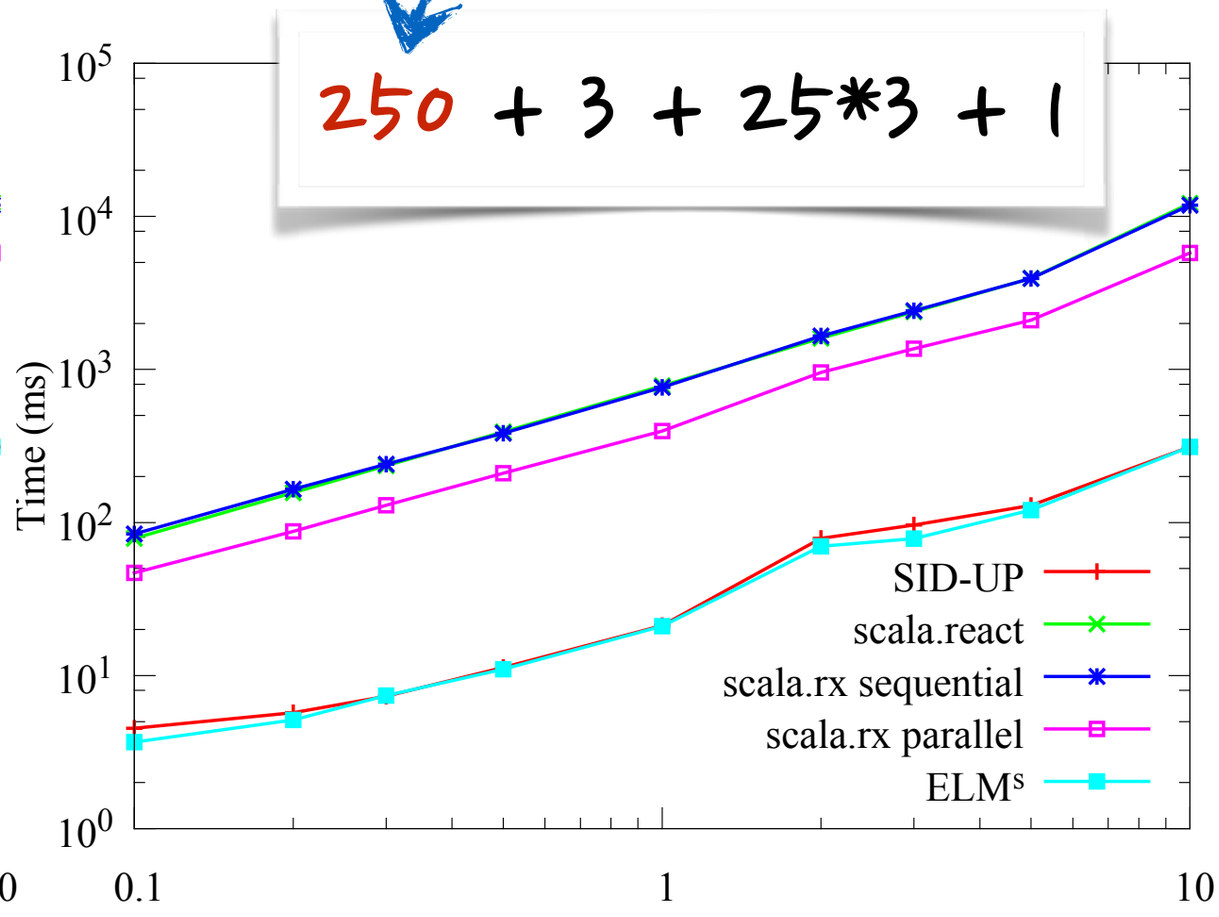
only 1 run

Benchmarks

Time to complete 1 run



unchanged sources



Time to complete 1 run

Time (ms)

10^3

10^2

10^1

10^0

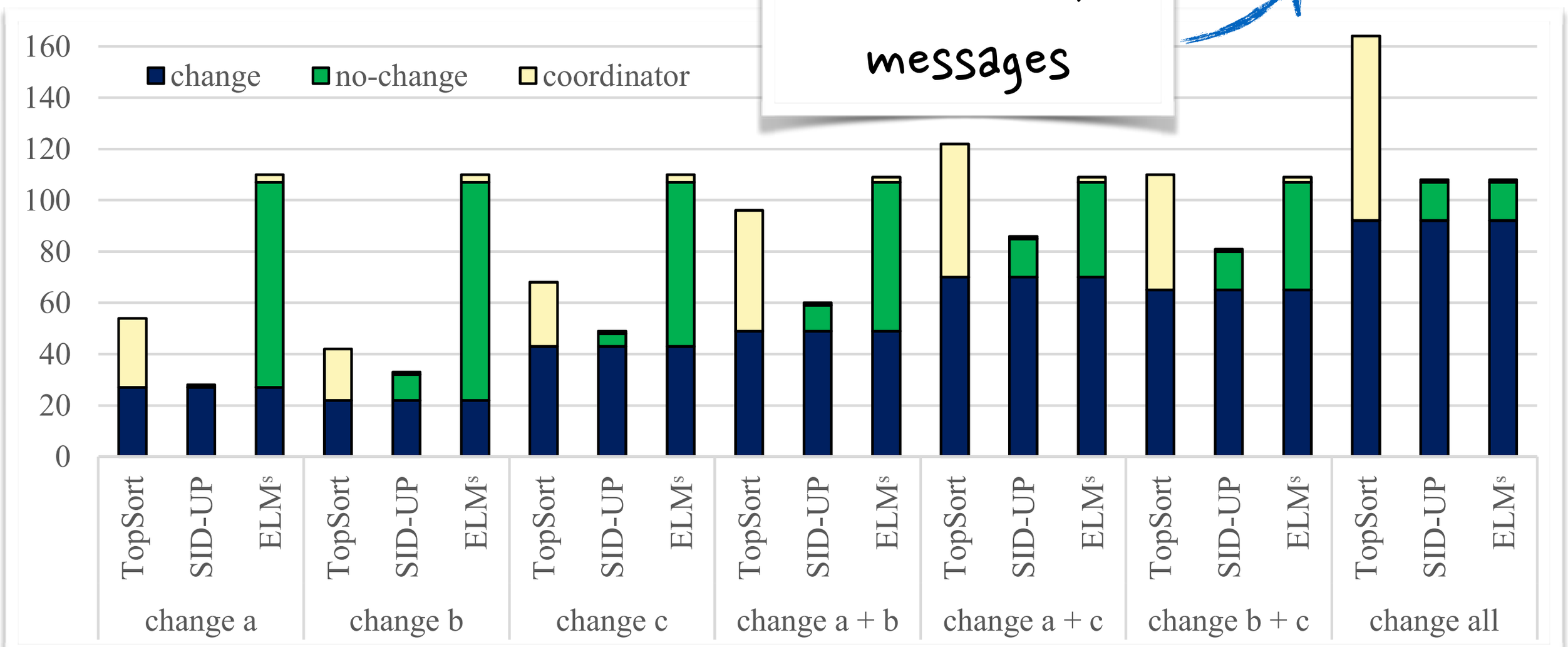
0.1

1

Benchmarks

to show it is better than ELMs

Number of messages



Some discussion

ELM: pipelining support - not evaluated (only 1 run)

Scala.*: pipelining possible?

ELM^S + SID-UP: How to detect the end of a run?

Lots of changed sources: not tested

Scalability? (lock at each round)

Dynamic dependencies:

can affect topology? Assumptions?

distributed clock?

Optimisations

REBLS ’14 - Splash workshop
“***Optimizing Distributed REScala***”
Joscha Drechsler and Guido Salvaneschi

If has 1 incoming dep.:

avoid iterations, intersections, waiting

If has no dyn. dep.:

just **count** incoming pulses

include **`valueChanged`** and **`sourcesChanged`** in pulse

Optimisations

REBLS '14 - Splash workshop

"***Optimizing***

Joscha D

Benchmarks without injected network delays

If has 1 incoming dep.:

avoid iterations, intersections, waiting

If has no dyn. dep.:

just **count** incoming pulses

include **`valueChanged`** and **`sourcesChanged`** in pulse