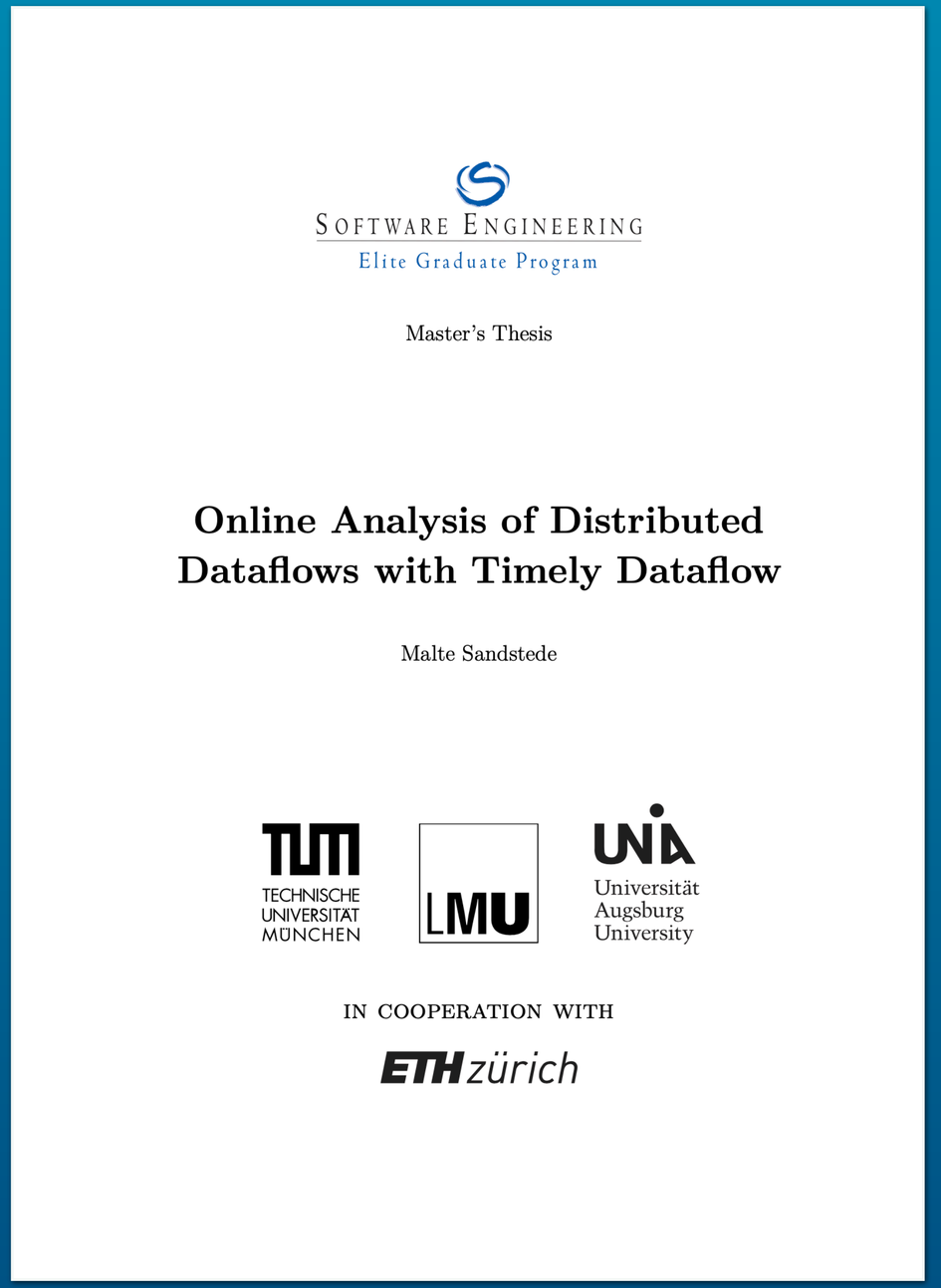
Online Analysis of Distributed Dataflows with Timely Dataflow

Malte Sandstede
TUM Chair for Database Systems, Nov 19

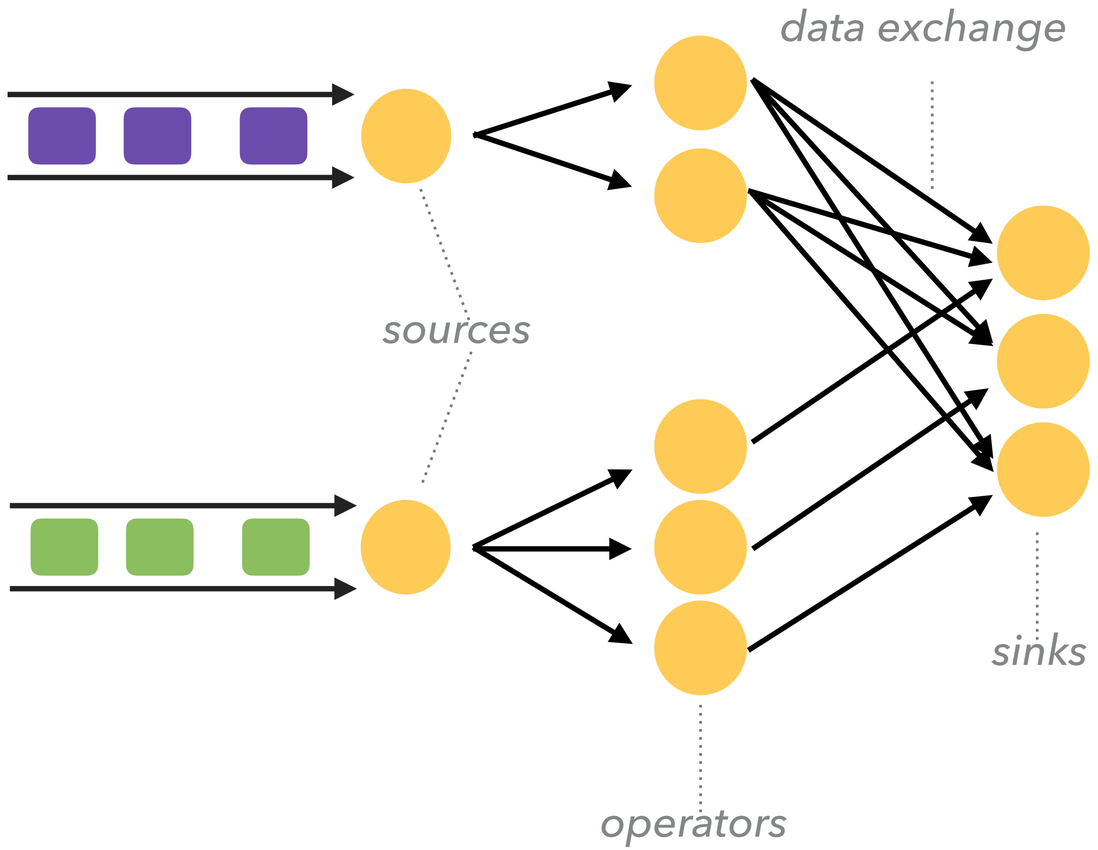


Author: Malte Sandstede
Submitted: 25 November 2019
Supervised by: Prof. Dr. Alfons Kemper (TUM)
Prof. Dr. Bernhard Bauer (UniA)
Prof. Dr. Timothy Roscoe (ETHZ)
Dr. Vasiliki Kalavri (ETHZ)
Jan Böttcher (TUM)

IN COOPERATION WITH

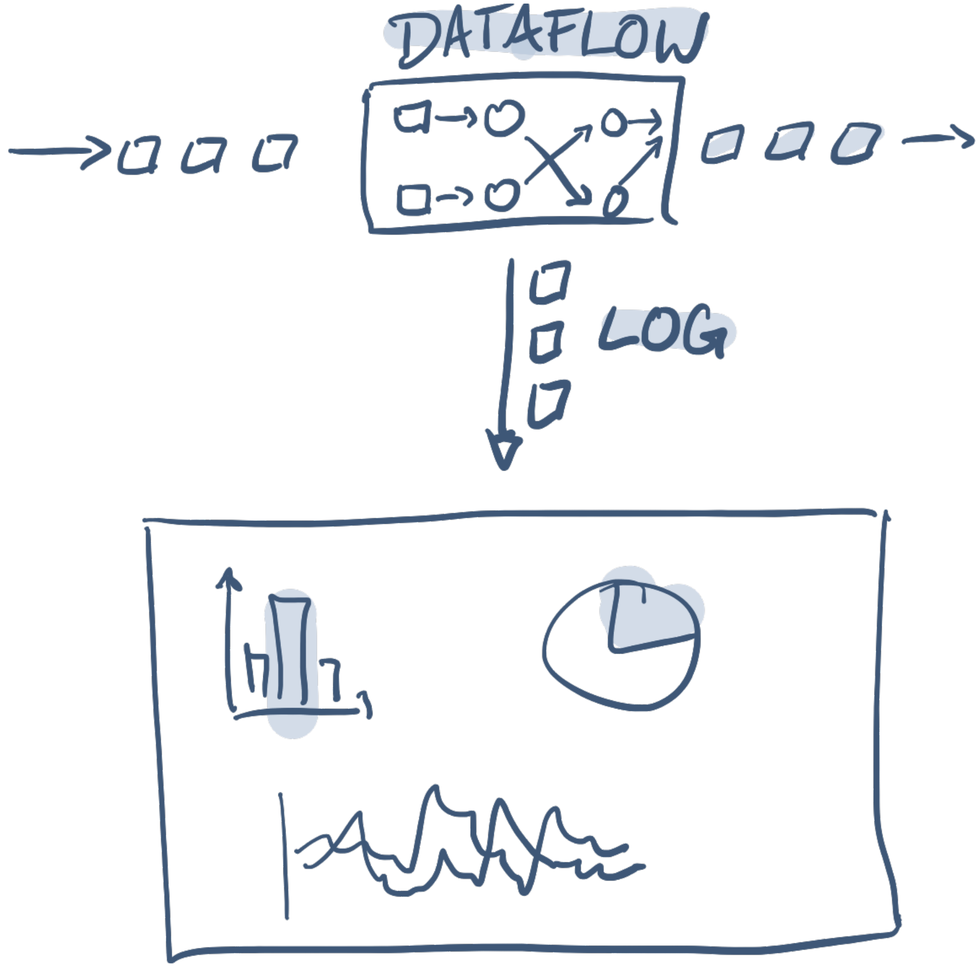
Motivation & Background

Dataflow Programming / Stream Processing



Online Dataflow Analysis

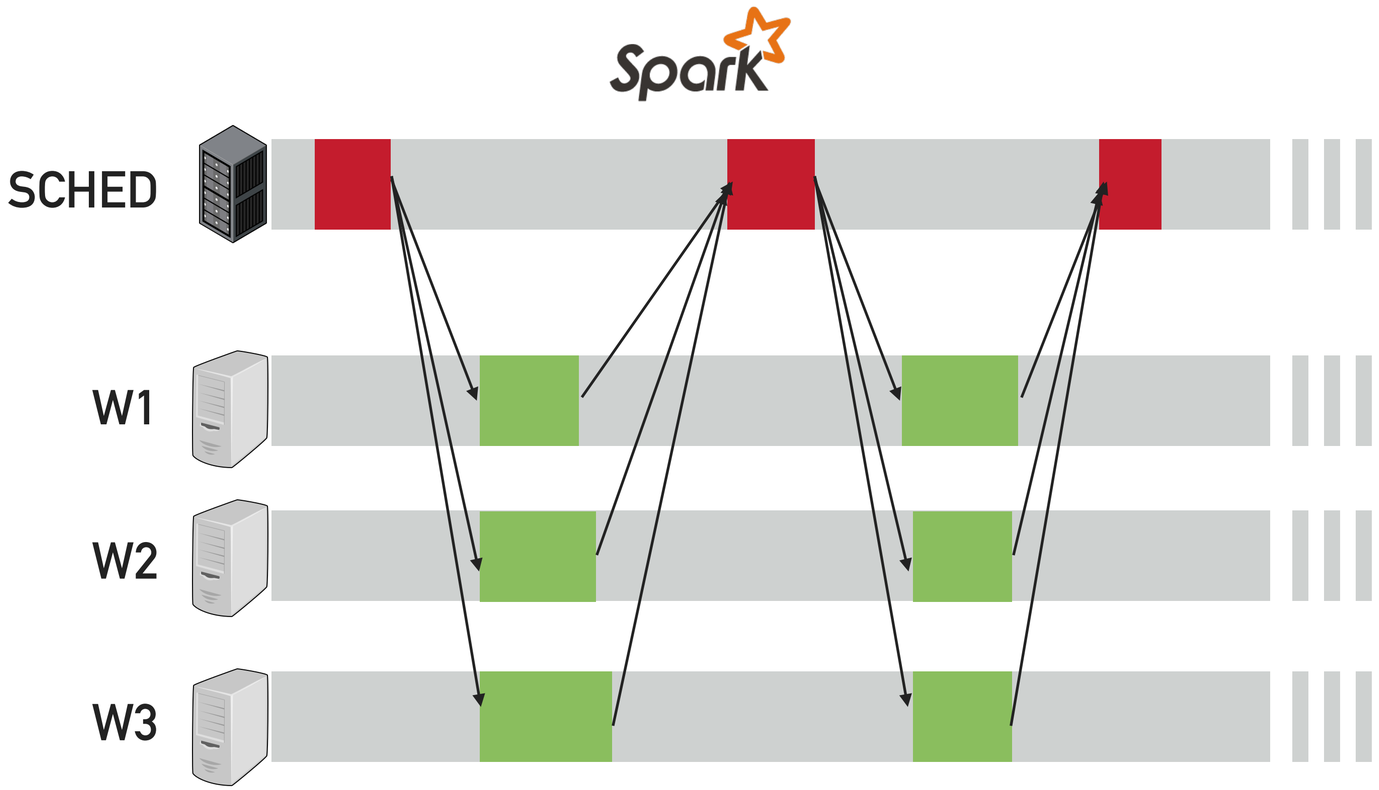
- “Yet another distributed system”
- Provenance & Explainability
- Performance Tuning
- Debugging
- SLAs

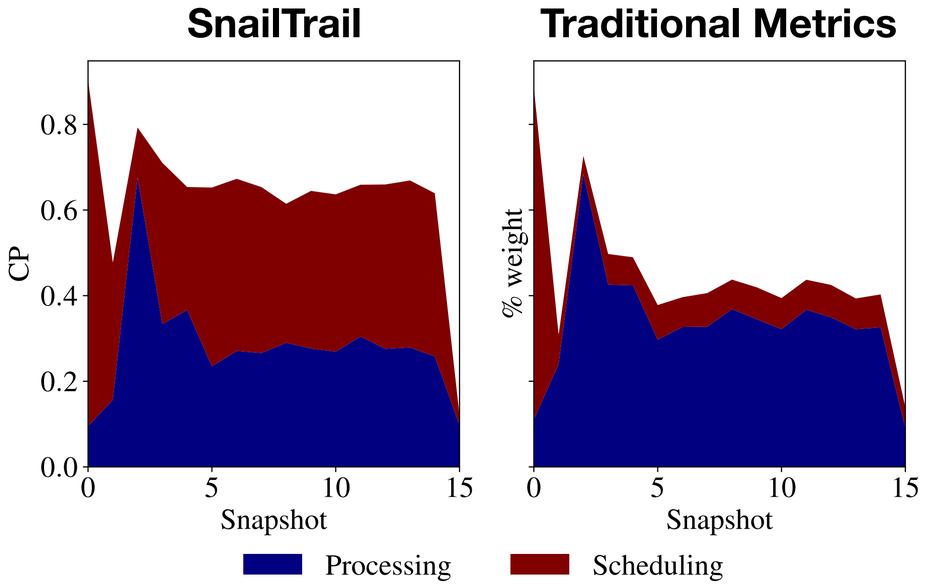


Online Dataflow Analysis

Issue	Solution
• Long-running, potentially unbounded workloads	• Continuous, correct results
• Stale information is not useful	• “Always online”-performance
• Many tasks, activities, operators, dependencies	• Complex queries: graph algos, pattern matching
• Heterogeneous analysis demands	• Multiple, accessible analyses

Example: Task Scheduling in Apache Spark





Apache Spark: Yahoo! Streaming Benchmark, 16 workers, 8s snapshots

Venkataraman, Shivaram, et al. "Drizzle: Fast and adaptable stream processing at scale." Spark Summit (2016).

The Stack

Timely Dataflow
(low latency runtime for distributed, complex dataflows)

- Originally developed at Microsoft Research (“Naiad”)
- (Re-)implemented in Rust
- Authored by Frank McSherry at ETH’s Systems Group
- Underlying framework for this thesis

github.com/**TimelyDataflow**

for internals, cf. appendix

The Stack

Differential Dataflow
(incrementalized, relational operators)

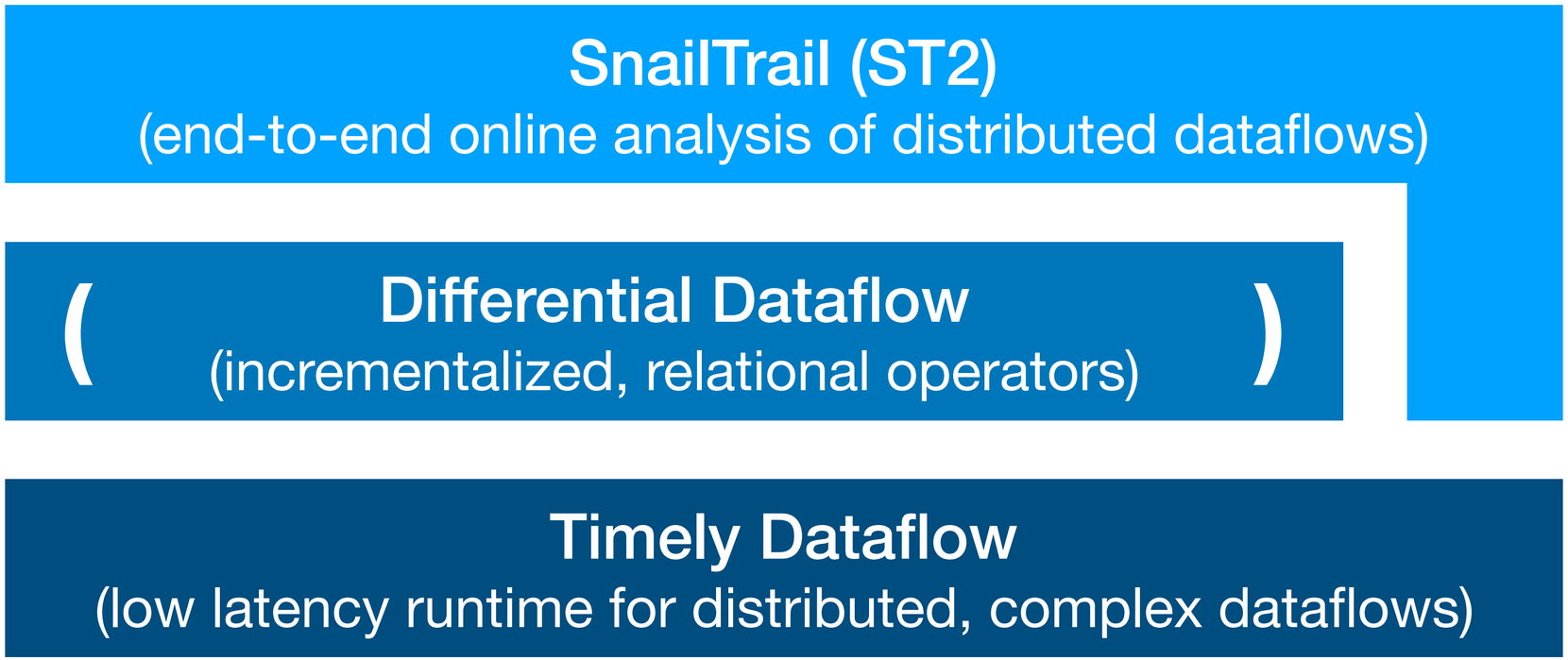
- “Add-On” to Timely Dataflow
- Differential computation
- Authored by Frank McSherry at ETH’s Systems Group
- Evaluated during this thesis, ultimately discarded

Timely Dataflow
(low latency runtime for distributed, complex dataflows)

github.com/**TimelyDataflow**

for internals, cf. appendix

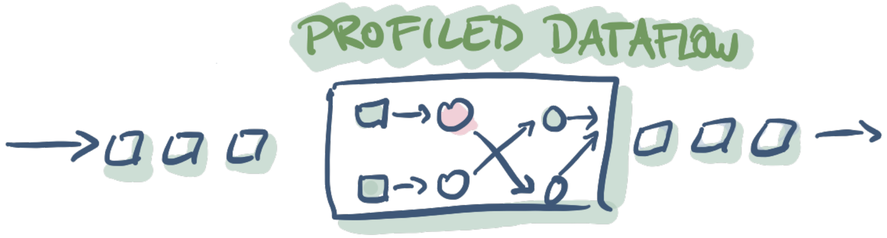
The Stack



github.com/**TimelyDataflow** github.com/li1/**SnailTrail** *for internals, cf. appendix*

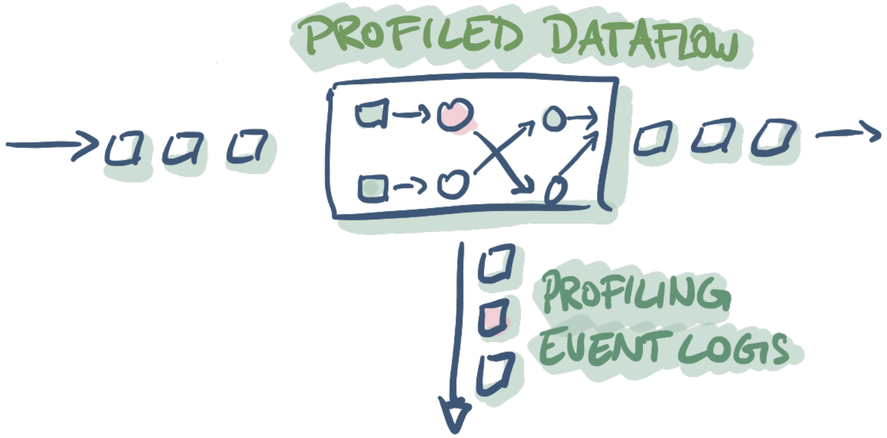
SnailTrail (ST2)

1. Run Profiled Computation



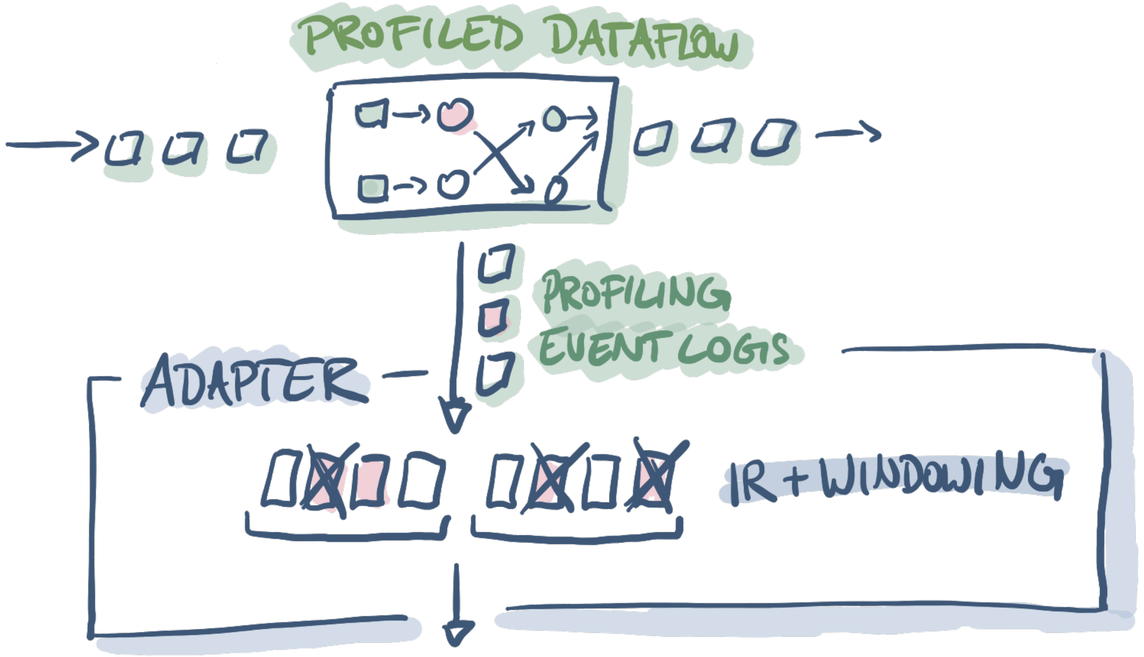
Profiled computation: Spark, Flink, Timely Dataflow, Differential Dataflow, Heron, Tensorflow, …

2. Extract Stream of Log Events



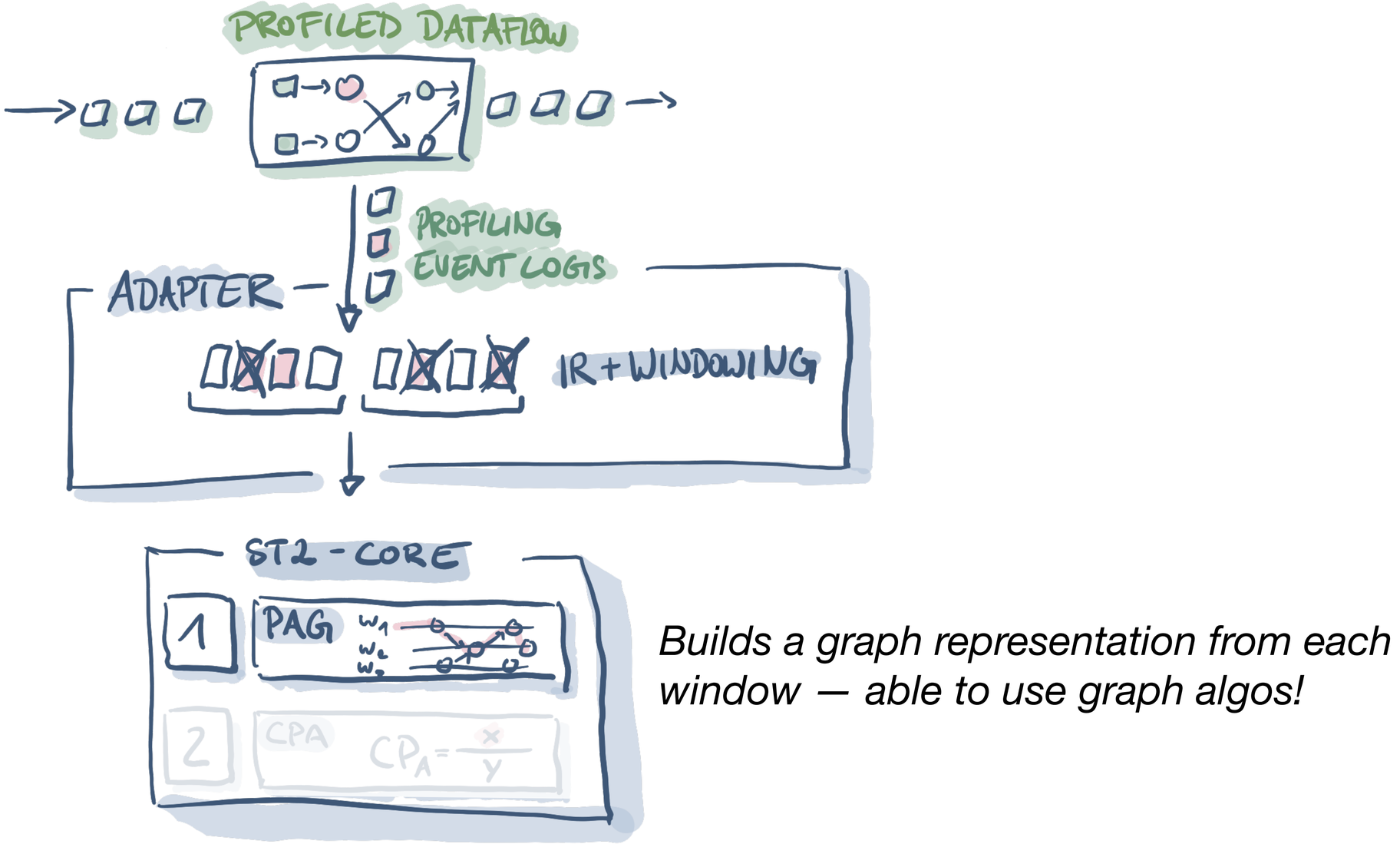
Emitted log events: scheduling, processing, data messages, control messages, …

3. Setup Time and Window Semantics



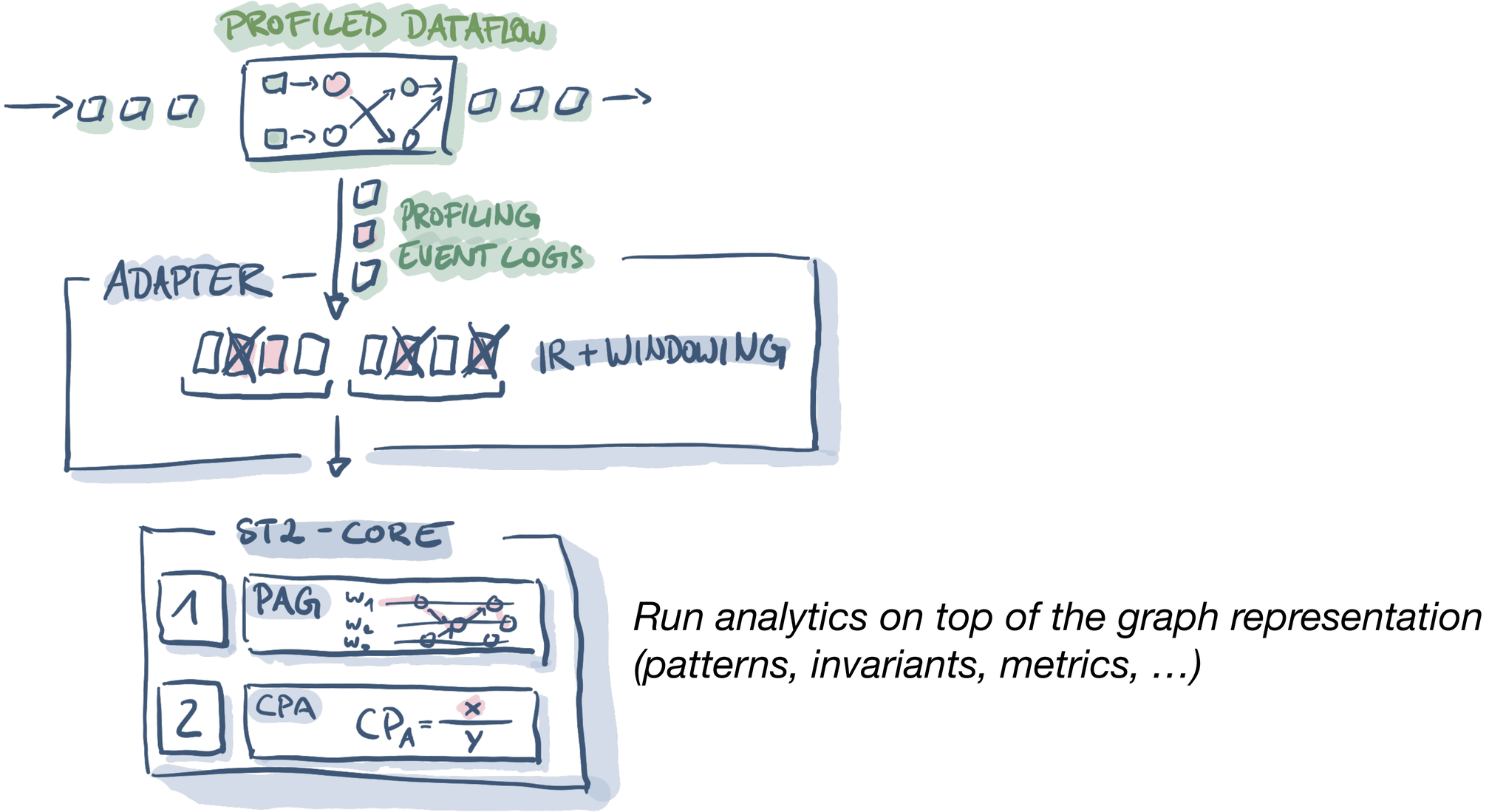
Creates IR from log events, grouped into windows (“epochs”)

4. Build Program Activity Graph



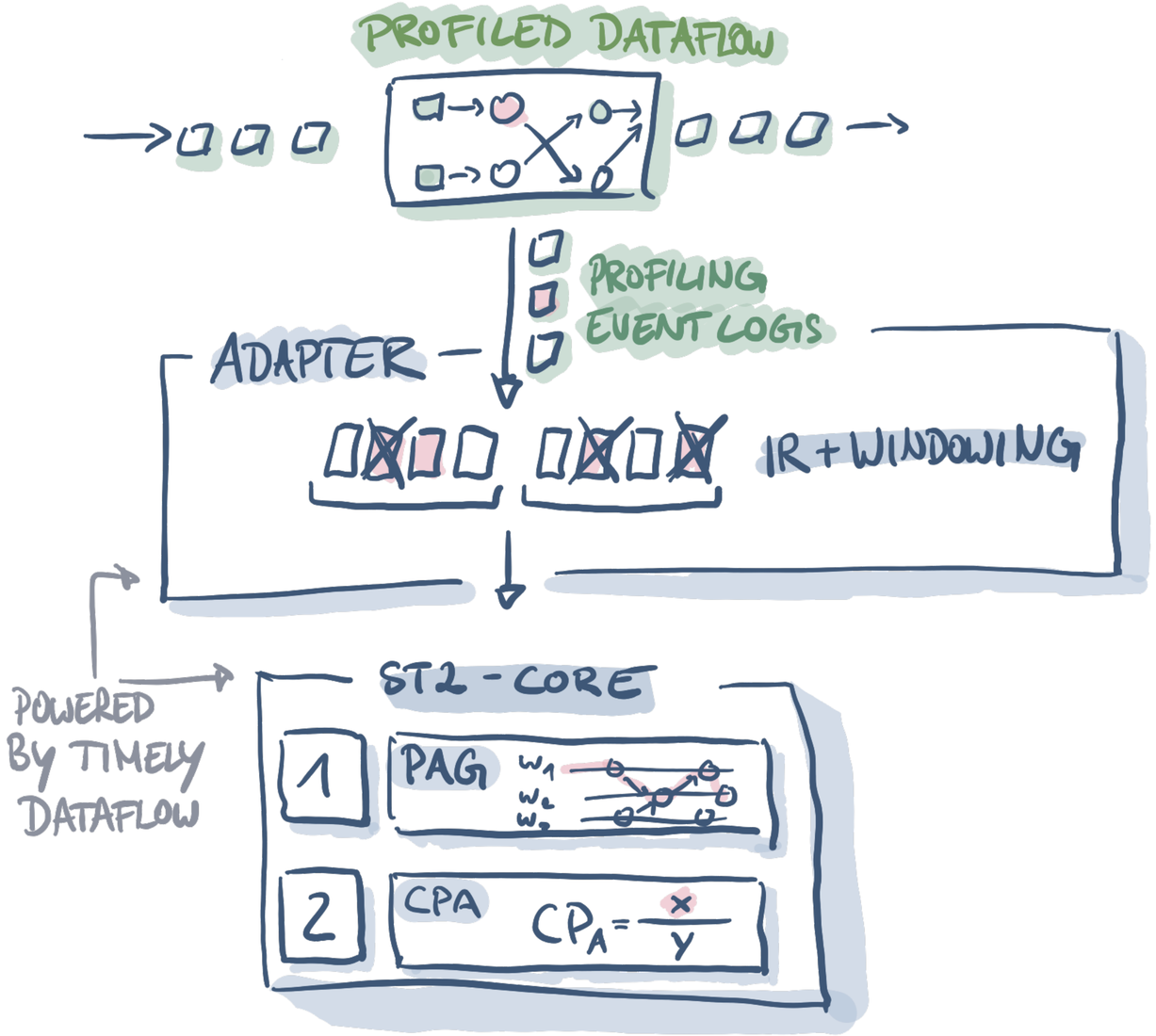
Builds a graph representation from each window — able to use graph algos!

5. Run Analytics

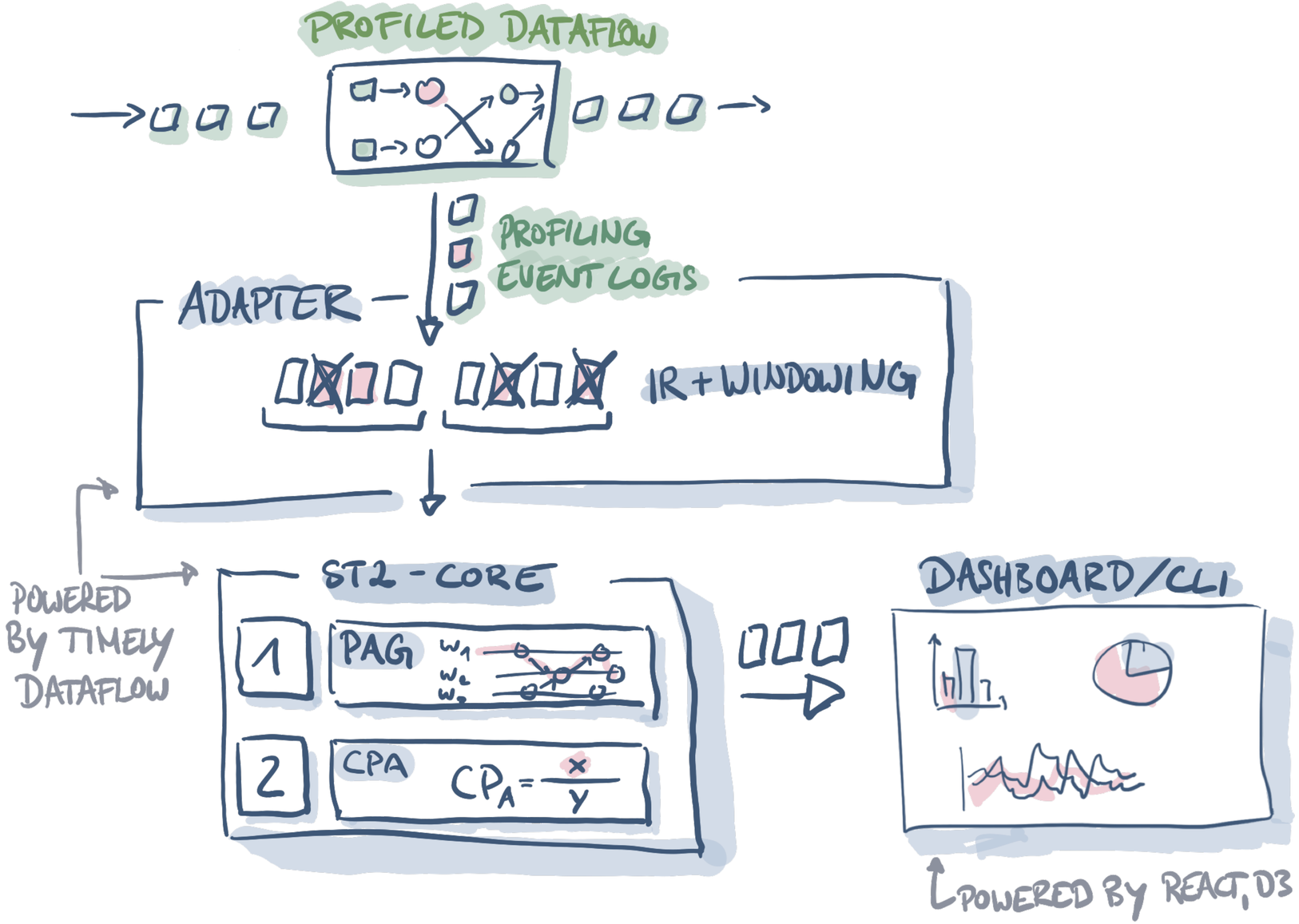


Run analytics on top of the graph representation (patterns, invariants, metrics, …)

5. Run Analytics

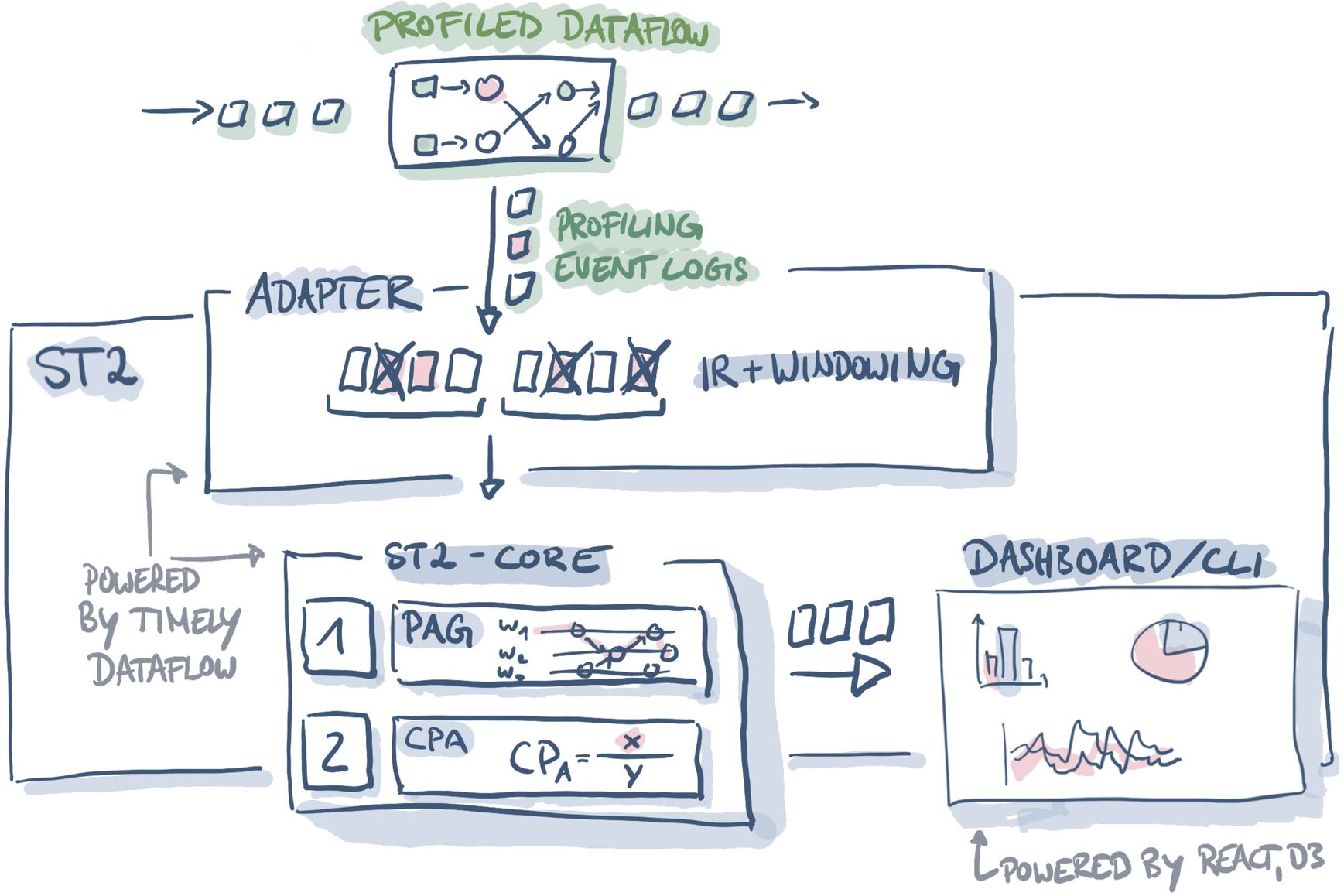


6. Present Results in Dashboard

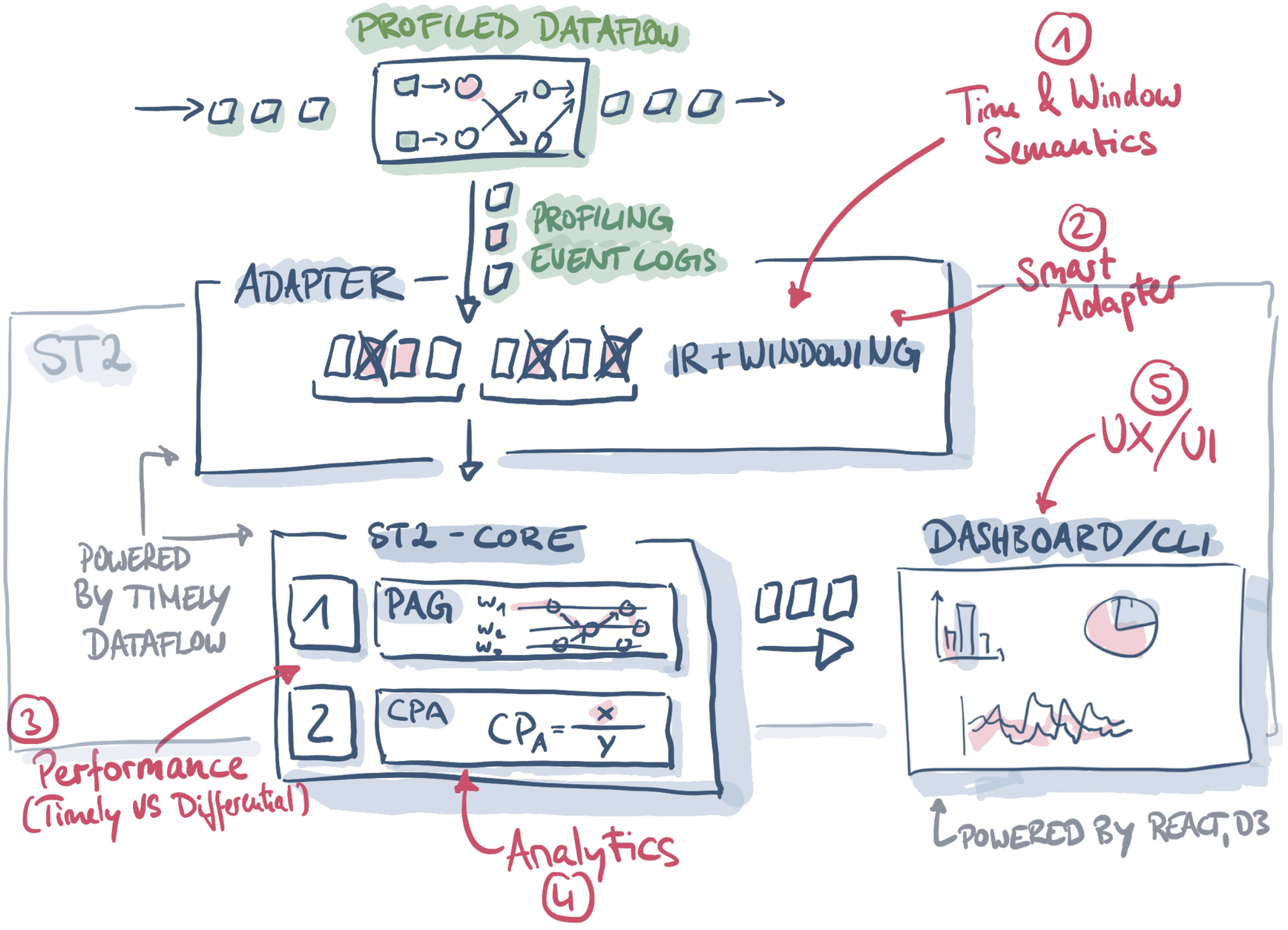


Visualize results in dashboard / CLI

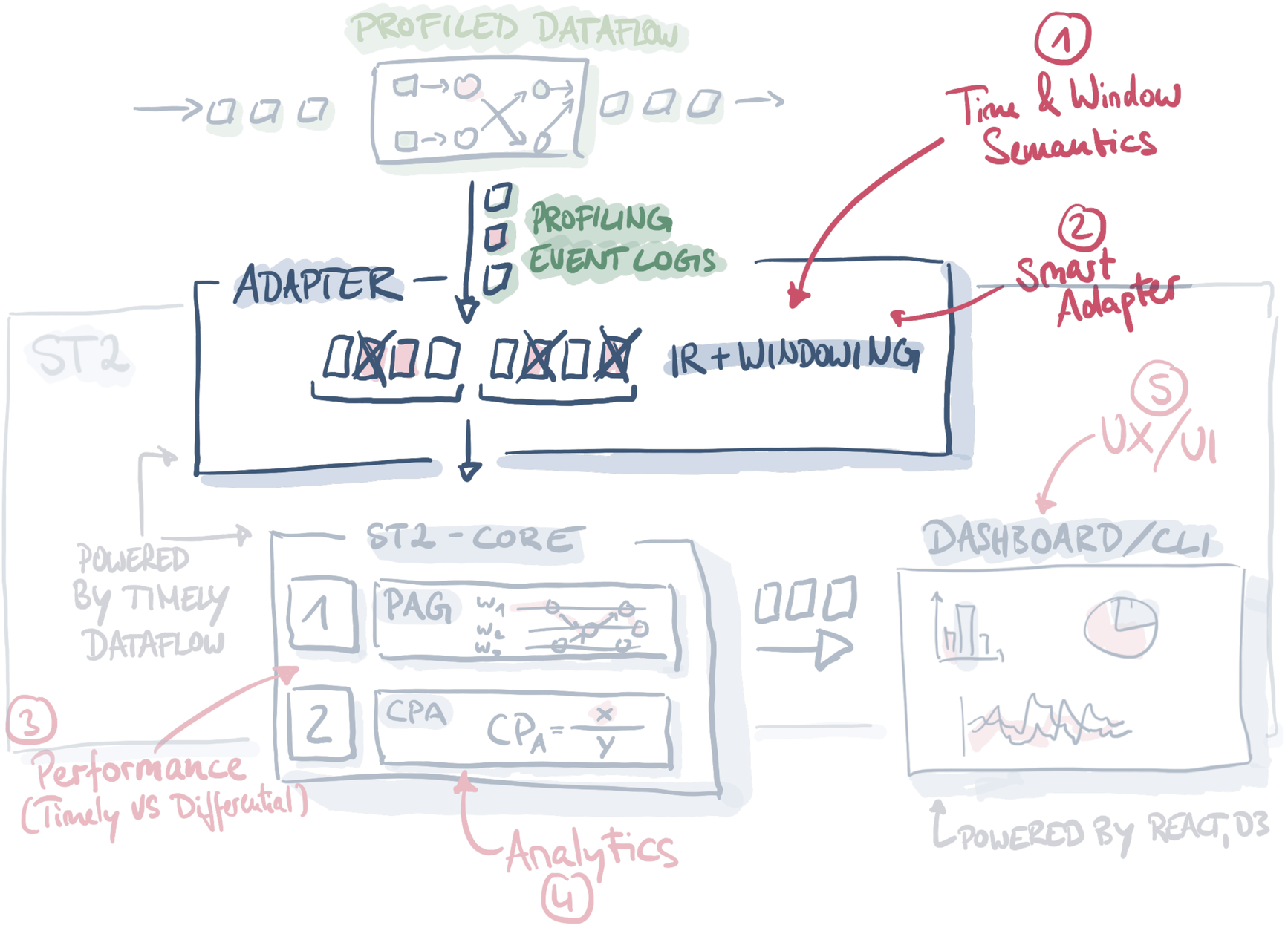
The “TODO List”



The “TODO List”



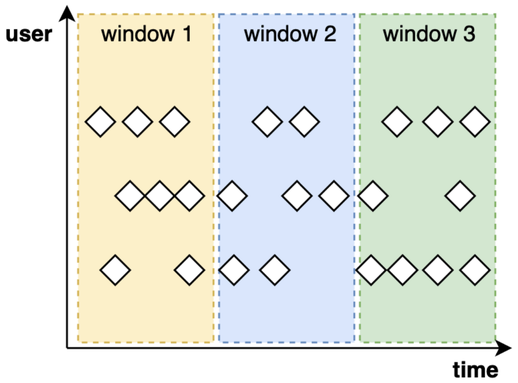
The “TODO List”



Window Semantics

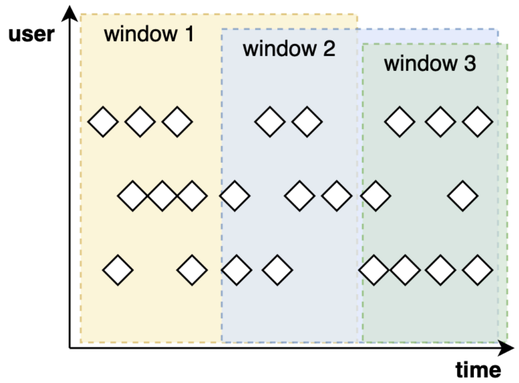
no (single) window

- state accumulates
- operators have to be incrementalized
- stale information



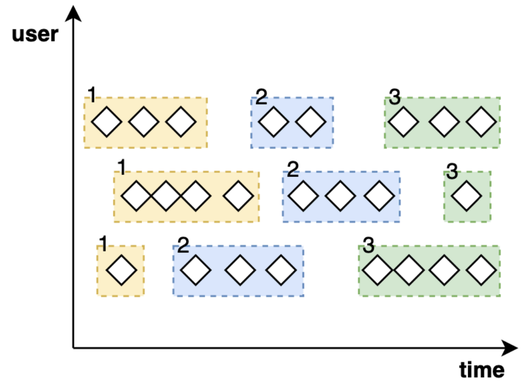
fixed window

- arbitrary splits
- Differential: retract/ add whole windows



sliding window

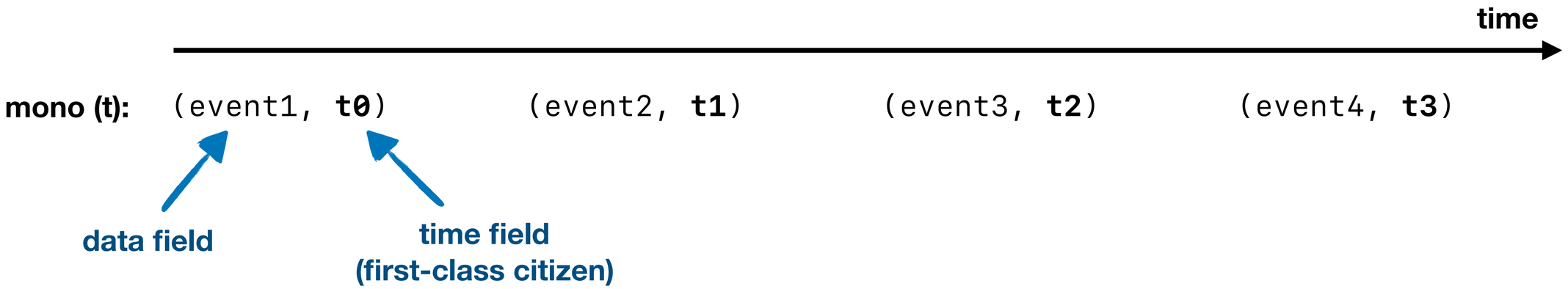
- generalization of fixed window
- arbitrary splits
- Differential: “smooth” retracts/adds



session window

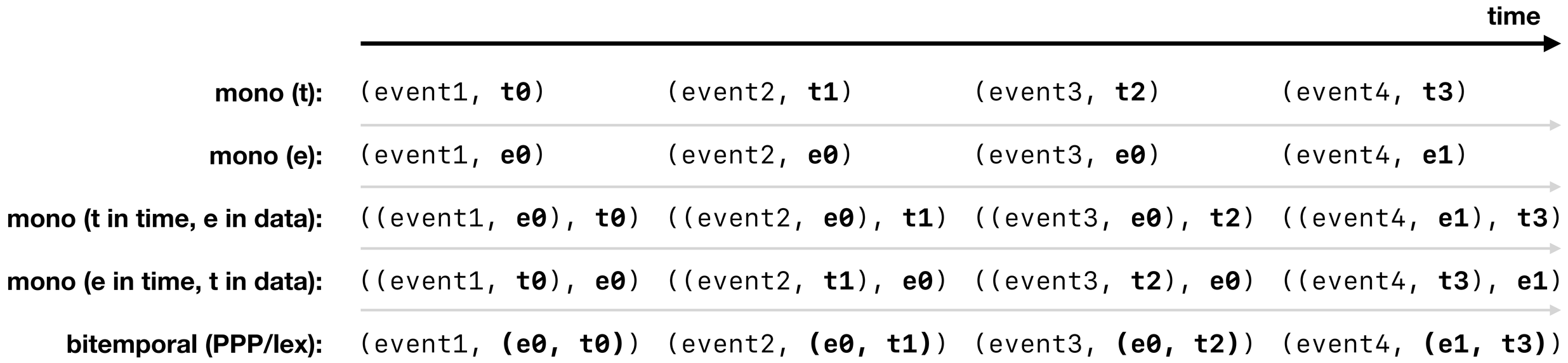
- semantic splits
- Differential: similarity across windows?
- more intrusive

Time Semantics



	distinct epochs	ordered events	expressive	non-blocking
mono (processing)	no	yes	no	yes

Time Semantics



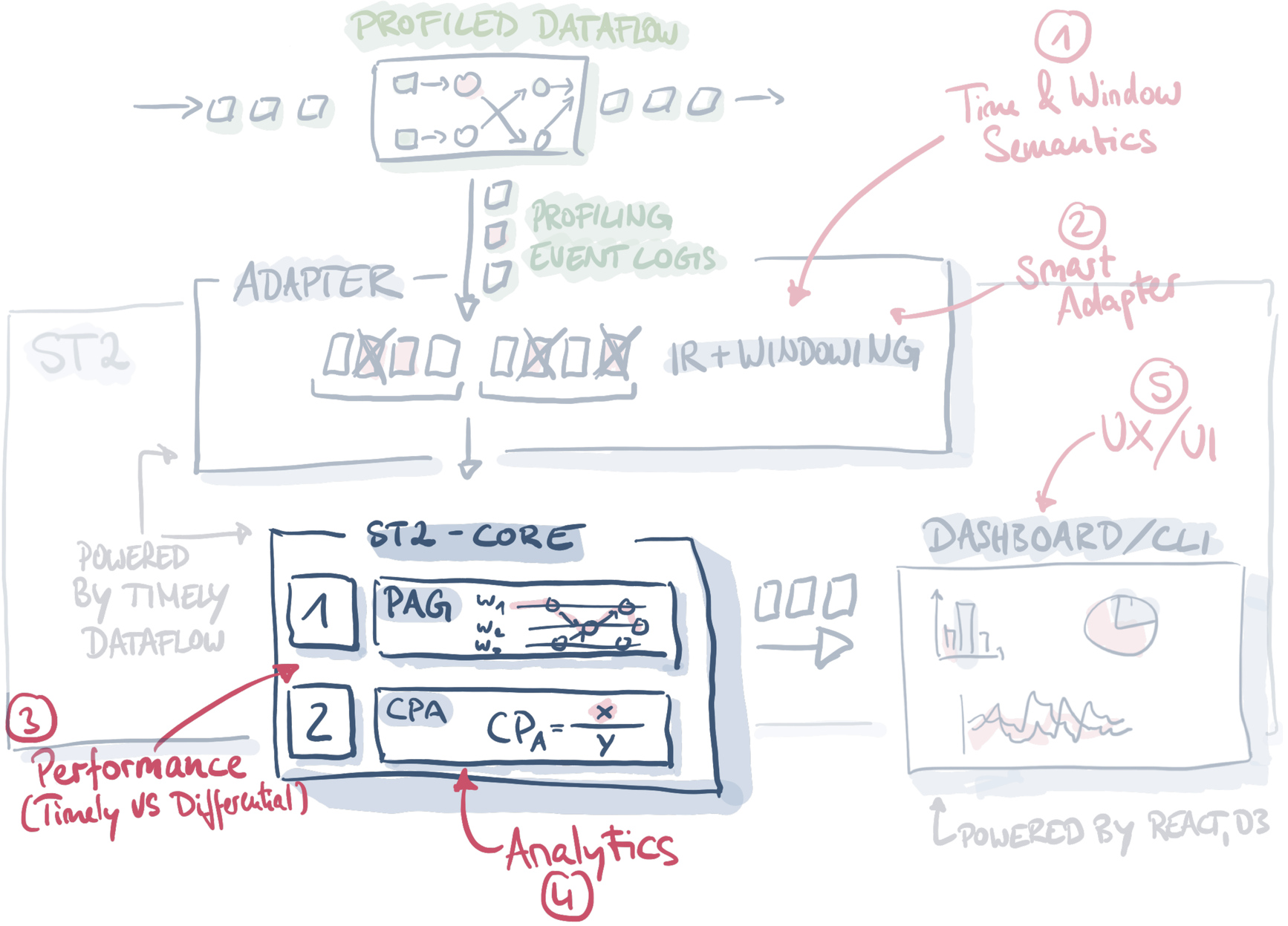
	distinct epochs	ordered events	expressive	non-blocking
mono (processing)	no	yes	no	yes
mono (epoch)	yes	no	yes	no
mono (processing, epoch in data)	yes	yes	no	yes
mono (epoch, processing in data)	yes	yes	yes	no
bi (product partial order)	yes	yes	yes	yes
bi (lexicographical total order)	yes	yes	yes	yes

Adapter Library

- Specialized for analyzed stream processor
- Constructs system-agnostic IR
- Offline (file) & online (TCP) mode
- Avoids irrelevant log event transmissions (~40%)
- Decouples scalability (round-robin event writer)

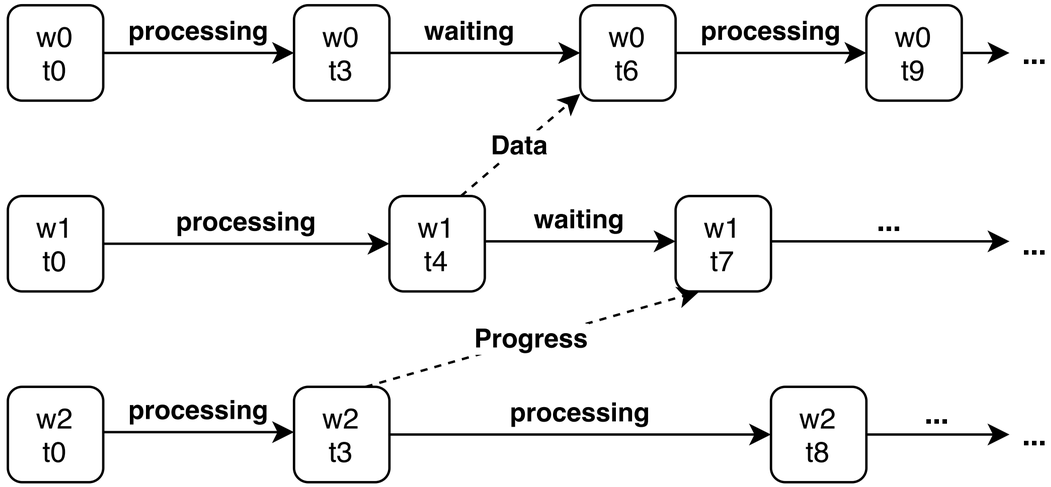
```rust
use timely::dataflow::InputHandle;
use timely::dataflow::operators::{Input, Exchange, Inspect, Probe};

use st2_timely::connect::Adapter;

fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        // (A) Create SnailTrail adapter at the beginning of the worker closure
        let adapter = Adapter::attach(worker);

        // Some computation
        let mut input = InputHandle::new();
        let probe = worker.dataflow(|scope|
            scope.input_from(&mut input)
                 .exchange(|x| *x as u64 + 1)
                 .inspect(move |x| println!("record {}", x))
                 .probe()
        );

        for round in 0..100 {
            if worker.index() == 0 { (0..20).for_each(|i| input.send(i) ) }
            input.advance_to(round + 1);
            while probe.less_than(input.time()) { worker.step(); }

            // (B) Communicate epoch completion
            adapter.tick_epoch();
        }
    }).unwrap();
}
```

The “TODO List”



PAG Construction

- Transform stream of IR records to graph representation…
- …i.e., a stream of edges
- Starting from IR:
  - Strip outer layers
    (antijoin against outer layers)
  - Join local, join remote nodes
  - Construct & interpret edges




```rust
pub struct PagEdge {
    /// The source node
    pub source: PagNode,
    /// The destination node
    pub destination: PagNode,
    /// The activity type
    pub edge_type: ActivityType,
    /// An optional operator ID
    pub operator_id: Option<OperatorId>,
    /// Edge dependency information
    pub traverse: TraversalType,
    /// record count
    pub length: Option<usize>,
}
```

```rust
pub struct PagNode {
    /// Timestamp of the event
    pub timestamp: Timestamp,
    /// event's worker ID
    pub worker_id: Worker,
    /// Epoch of PagNode
    pub epoch: u64,
    /// seq_no of PagNode
    pub seq_no: u64,
}
```

PAG Construction

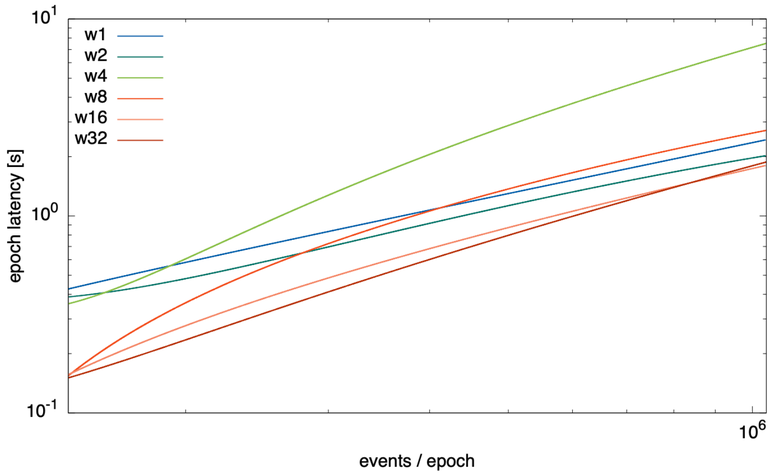
Differential PAG:
local edge join

```
fn make_local_edges(&self) -> Stream {
  let edge_start = self.map(|x| (x.seq_no, x));
  let edge_end = self.map(|x| (x.seq_no + 1, x));
  edge_start.join_map(&edge_end,
    |_key, curr, prev| build_local_edge(&prev, &curr))
}
```

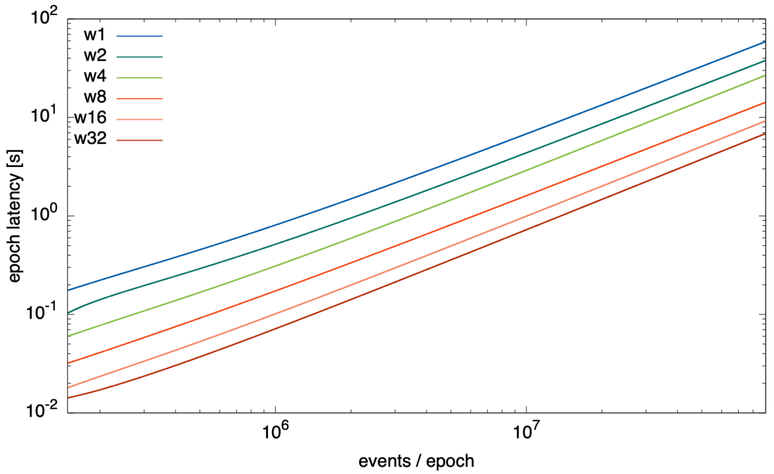
Timely PAG:
custom local edge operator

```
fn make_local_edges(&self) -> Stream {
  let mut prev = None;
  let mut prev2 = None;

  self.unary_frontier(Pipeline, move |input, output| {
    input.for_each(|time, logrecords| {
      for lr in logrecords {
        if let Some(prev) = prev {
          if let Some(prev2) = prev2 {
            output.give(build_local_edge(prev2, prev, lr));
          }

          // prev -> prev2
          prev2 = Some(prev);
        }

        // lr -> prev
        prev = Some(lr);
      }
    });
  })
}
```

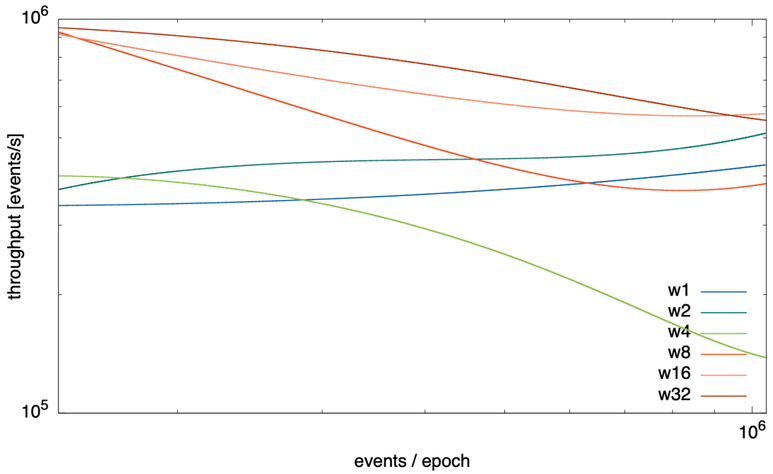
PAG Construction



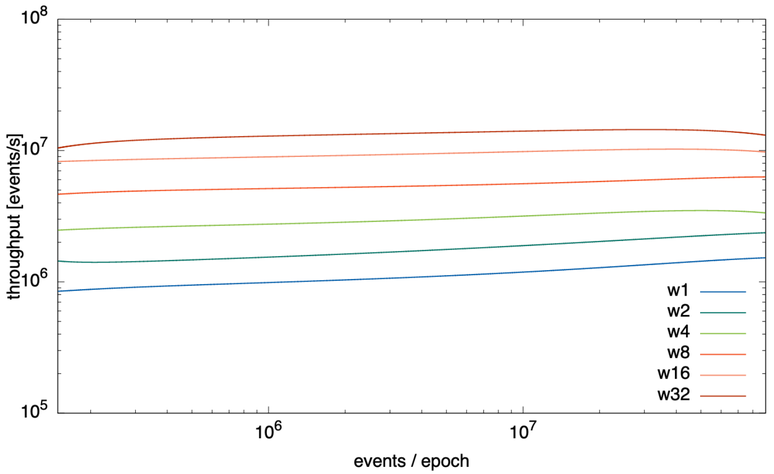
(a) *Differential PAG latency scaling*



(b) *Timely PAG latency scaling*



(c) *Differential PAG throughput scaling*



(d) *Timely PAG throughput scaling*

- Differential: ~1 OOM worse
  - not hand-rolled (e.g. join exchanges)
  - not incrementalized
  - no partial orders
- Timely: >10M events/s with 32 threads
  - favorable scaling characteristics
  - ~1 OOM faster than prev. version (“3 OOM faster than necessary”)
  - “All of LinkedIn processes 13M events/s”
- Online experiment keeps up with source computation

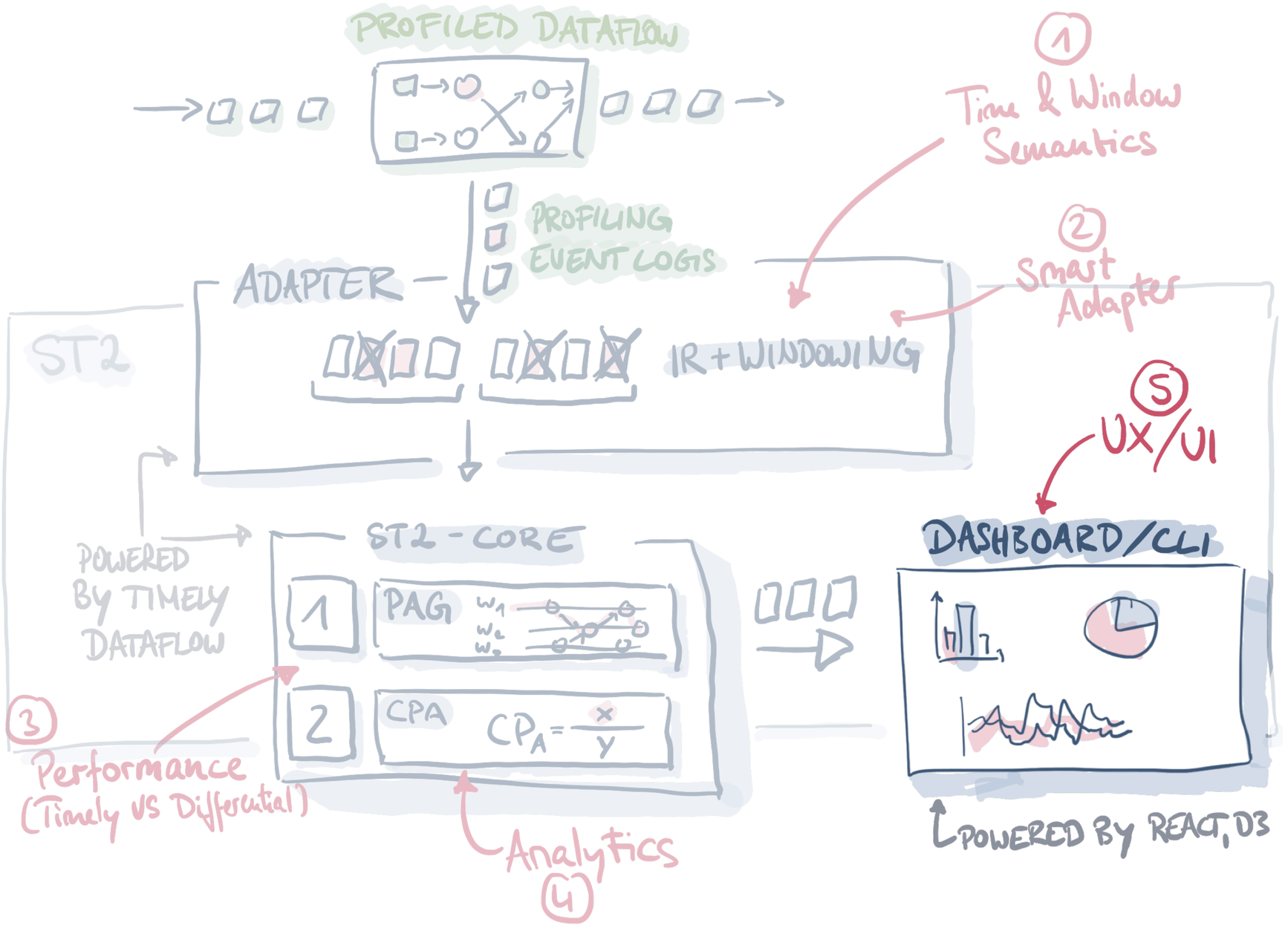
DD Triangles computation,
Intel Xeon E5-4650 v2 2.40 GHz,
32 physical threads, 512GB RAM, Debian 8 (“Jessie”)

Analytics / CLI

- Various analytics
  - graph patterns (e.g. k-hop)
  - aggregate metrics
  - invariant checking
- Extendable with Timely / Differential operators
- Accessible via self-documenting CLI
- Small performance impact (retains >10M events/s)

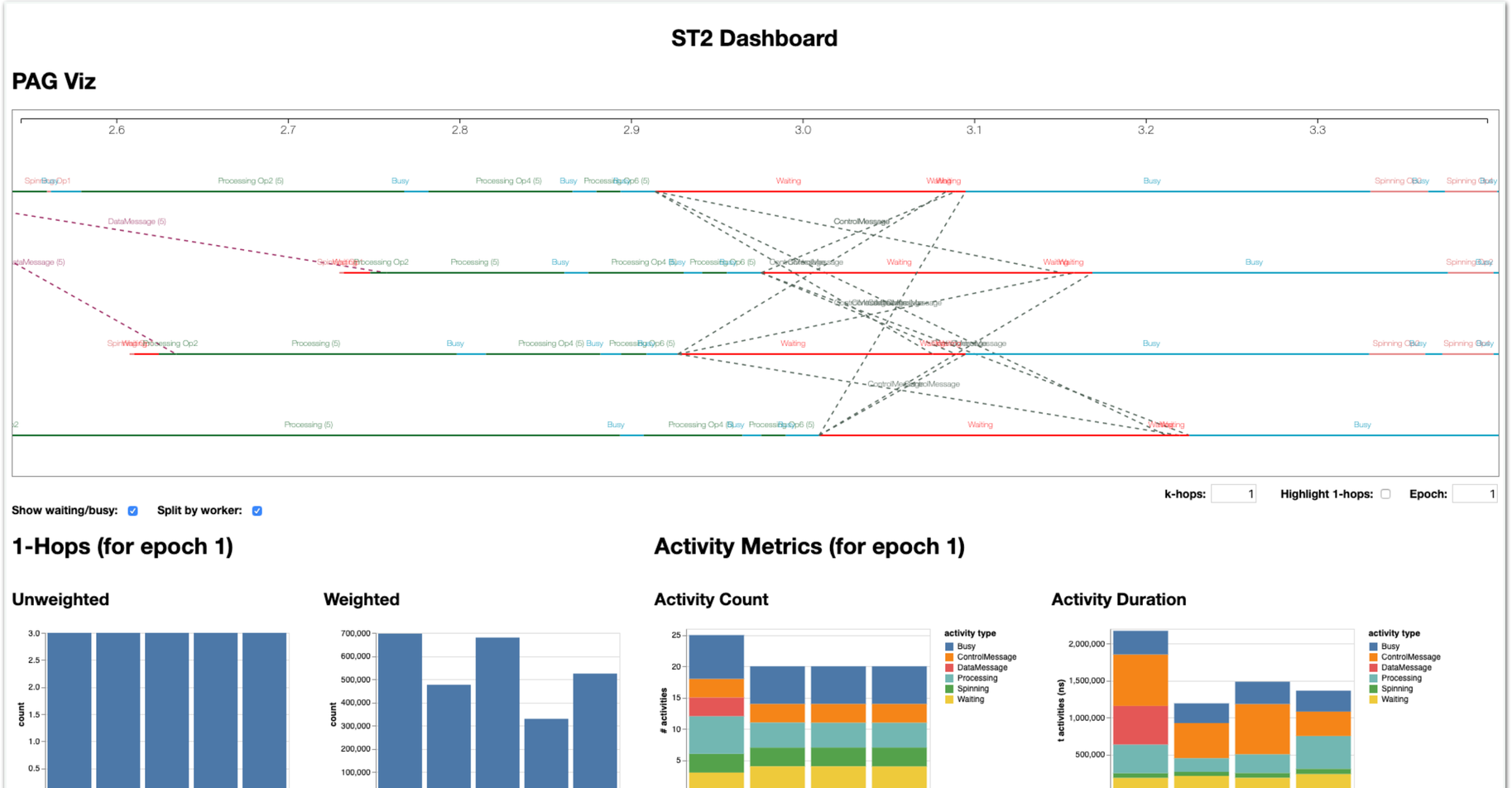
```
// Ensure that no message of the source computation takes
// longer than the provided duration in ms.
if let Some(temporal_message) = temporal_message {
    let temporal_message = Duration::from_millis(temporal_message);
    self
        .filter(|edge|
                (edge.edge_type == ActivityType::ControlMessage ||
                edge.edge_type == ActivityType::DataMessage))
        .filter(move |edge|
                edge.destination.timestamp - edge.source.timestamp > temporal_message)
        .inspect(move |(edge, _t, _diff)| {
            println!("Temporal Issue: {:?} (payload: {:?}) in e{}, w{} to w{} ran \
                      from {:?} to {:?}, taking {:?}. \
                      Maximum allowed is {:?}.",
                     edge.edge_type,
                     edge.length,
                     edge.source.epoch,
                     edge.source.worker_id,
                     edge.destination.worker_id,
                     edge.source.timestamp,
                     edge.destination.timestamp,
                     (edge.destination.timestamp - edge.source.timestamp),
                     temporal_message)
        })
        .probe_with(probe);
}
```

The “TODO List”



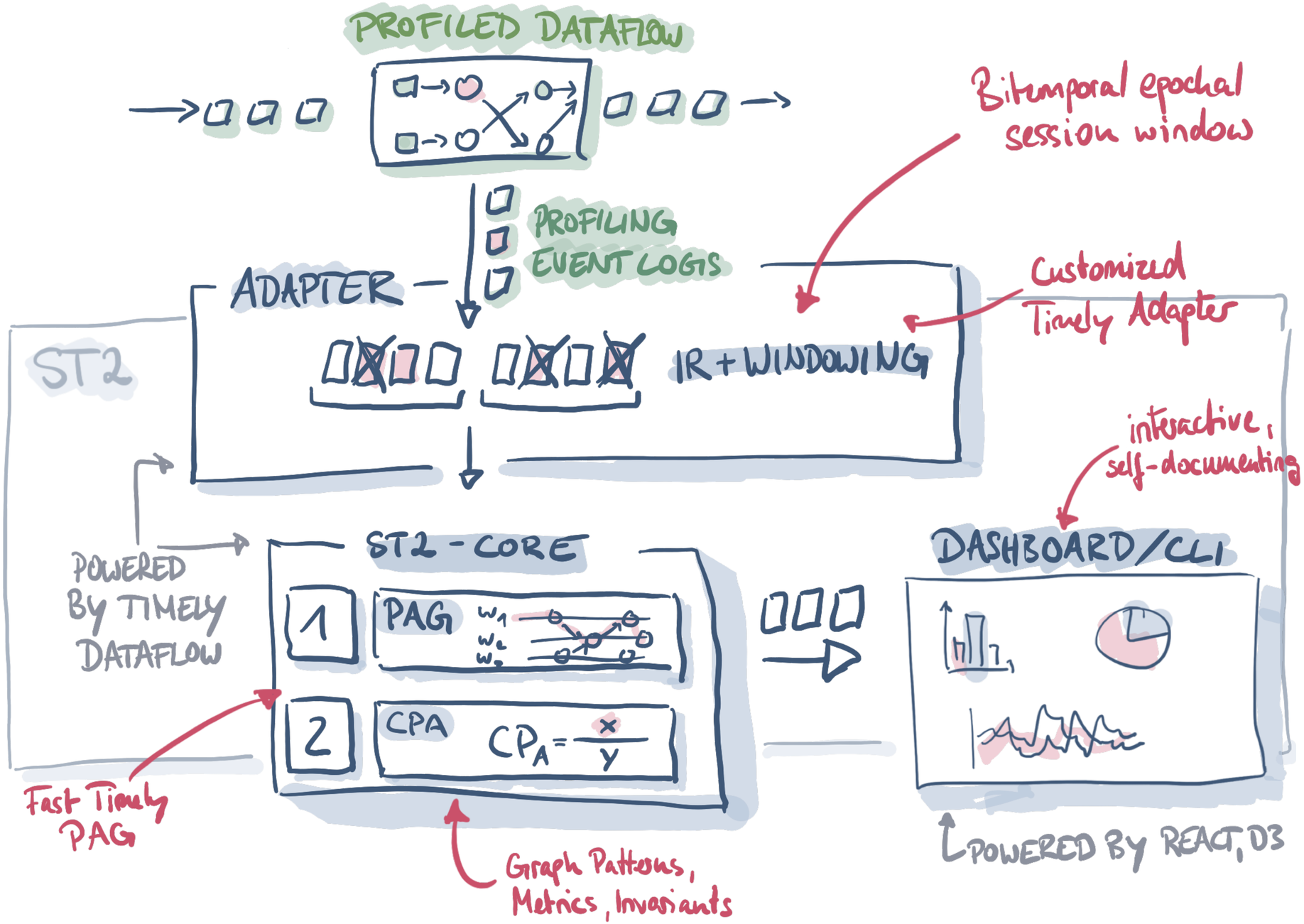
Dashboard

Combines dataflow & pattern visualization with prepared analysis diagrams



End-to-End Online Dataflow Analysis

Want	Get via
• Continuous, correct results	• non-blocking time semantics, custom adapter
• “Always online”-performance	• Efficient PAG/analytics operators, custom adapter
• Complex queries: graph algorithms, pattern matching	• Custom implementation on top of Timely Dataflow
• Multiple, accessible analyses	• CLI, Dashboard



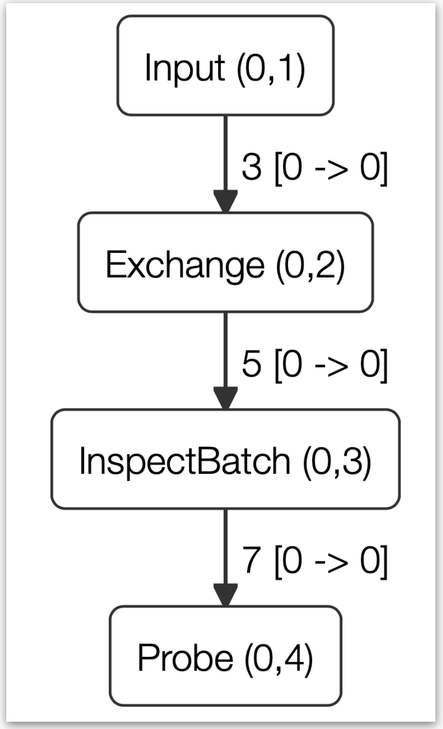
github.com/li1/**snailtrail**

Appendix

Timely Internals

Running Dataflows with Timely

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        // Some computation
        let mut input = InputHandle::new();
        let probe = worker.dataflow(|scope|
            scope.input_from(&mut input)
                 .exchange(|x| *x as u64 + 1)
                 .inspect(move |x| println!("record {}", x))
                 .probe()
        );

        for round in 0..100 {
            if worker.index() == 0 { (0..20).for_each(|i| input.send(i) ) }
            input.advance_to(round + 1);
            while probe.less_than(input.time()) { worker.step(); }



        }
    }).unwrap();
}
```



Correctness Troubles

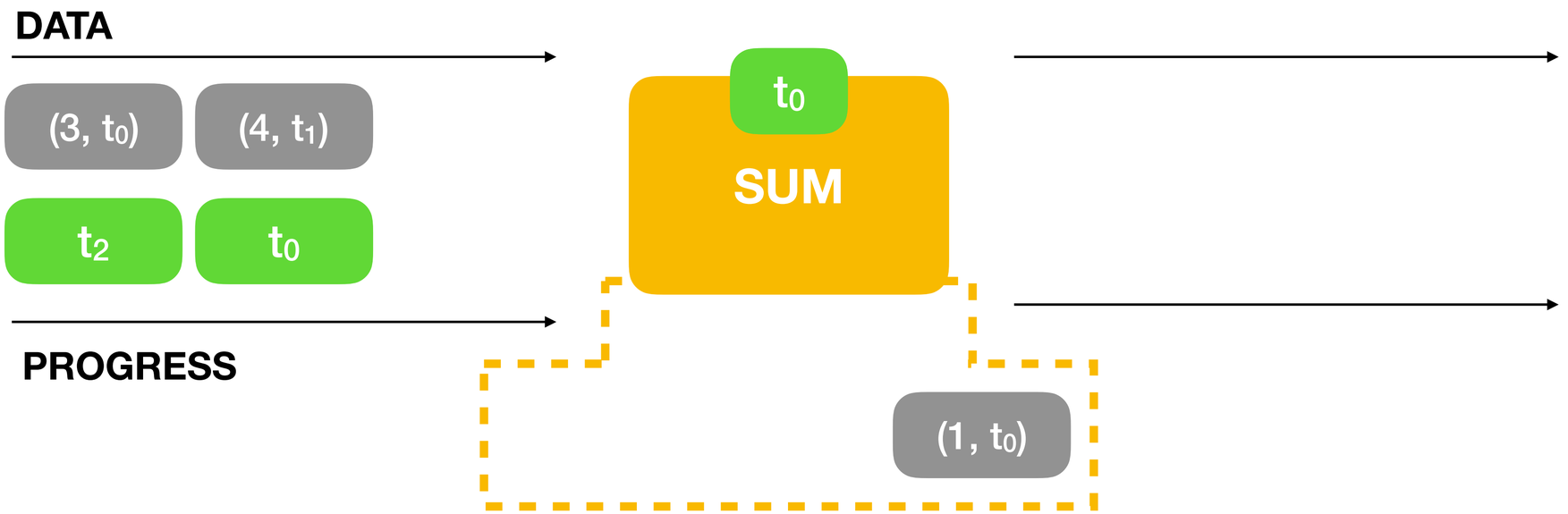
Correctness Troubles

Correctness Troubles

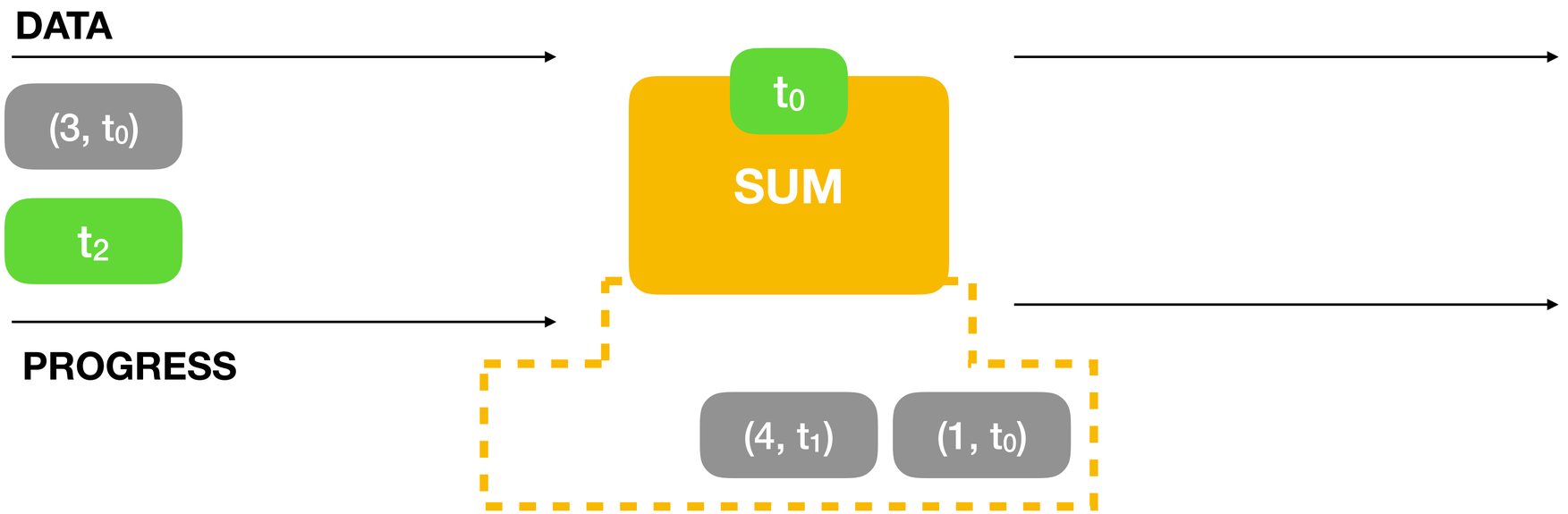
Correctness Troubles

Correctness with Progress Tracking

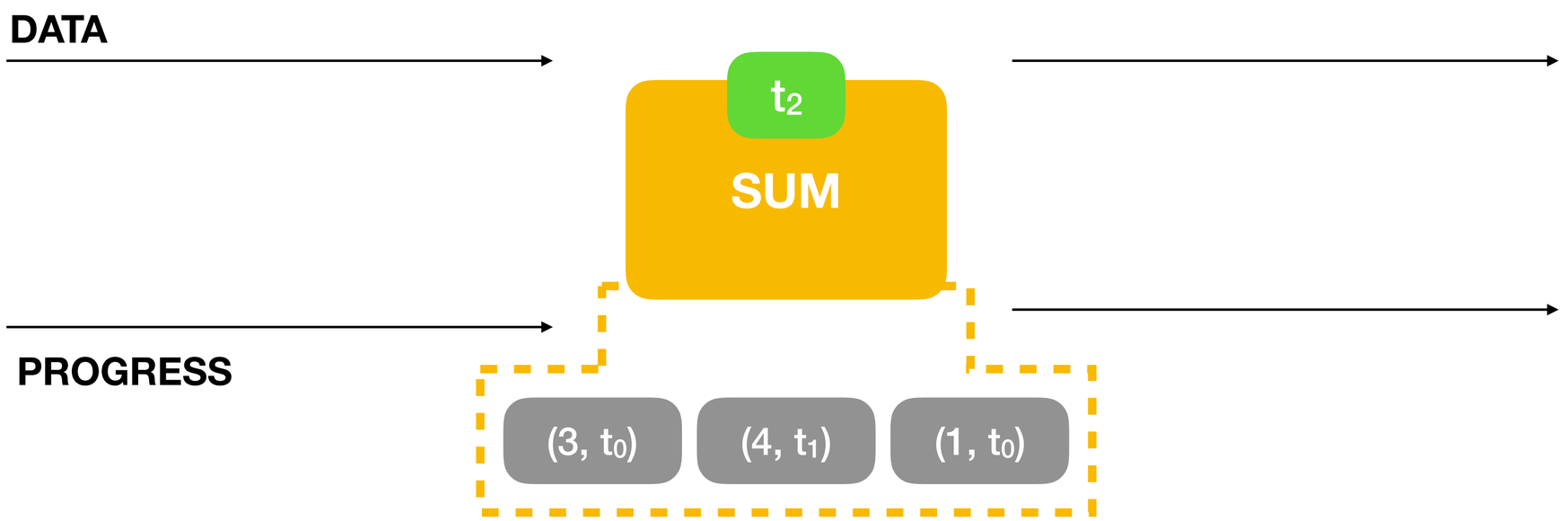
Correctness with Progress Tracking



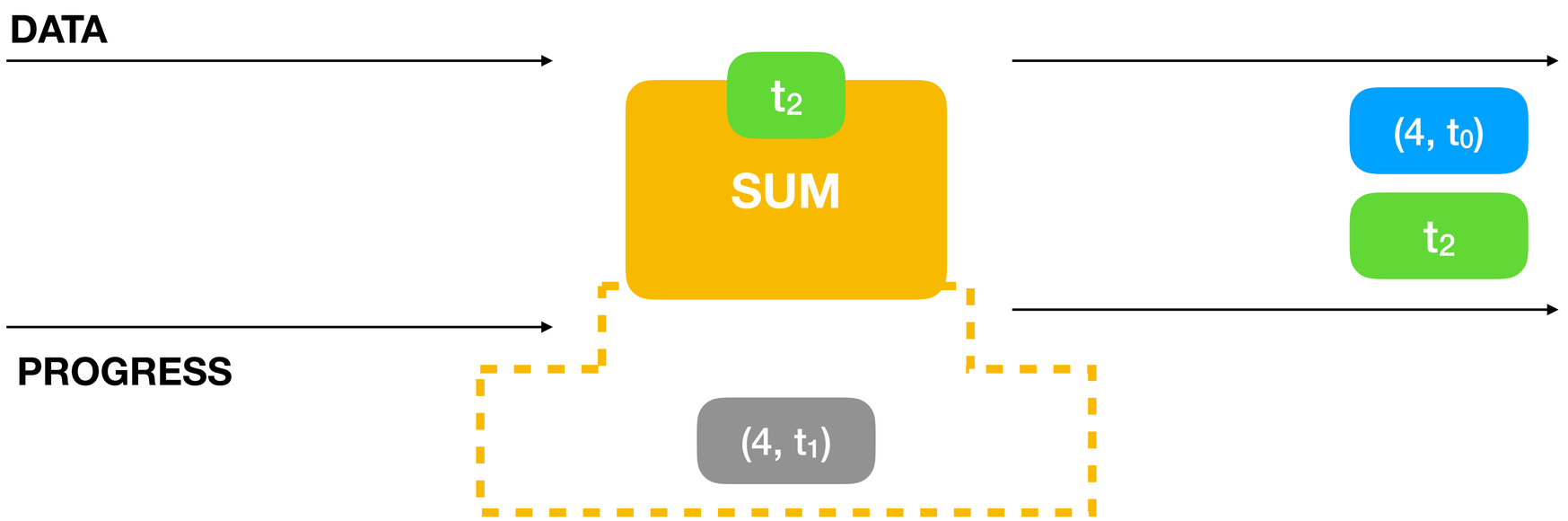
Correctness with Progress Tracking



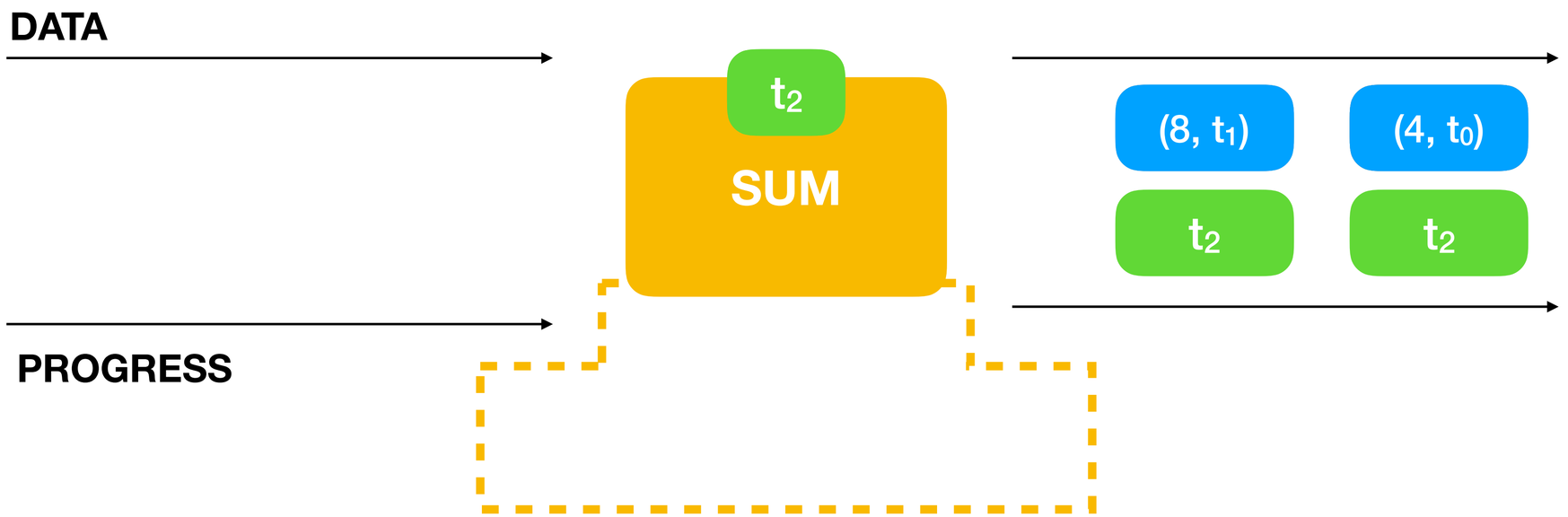
Correctness with Progress Tracking



Correctness with Progress Tracking

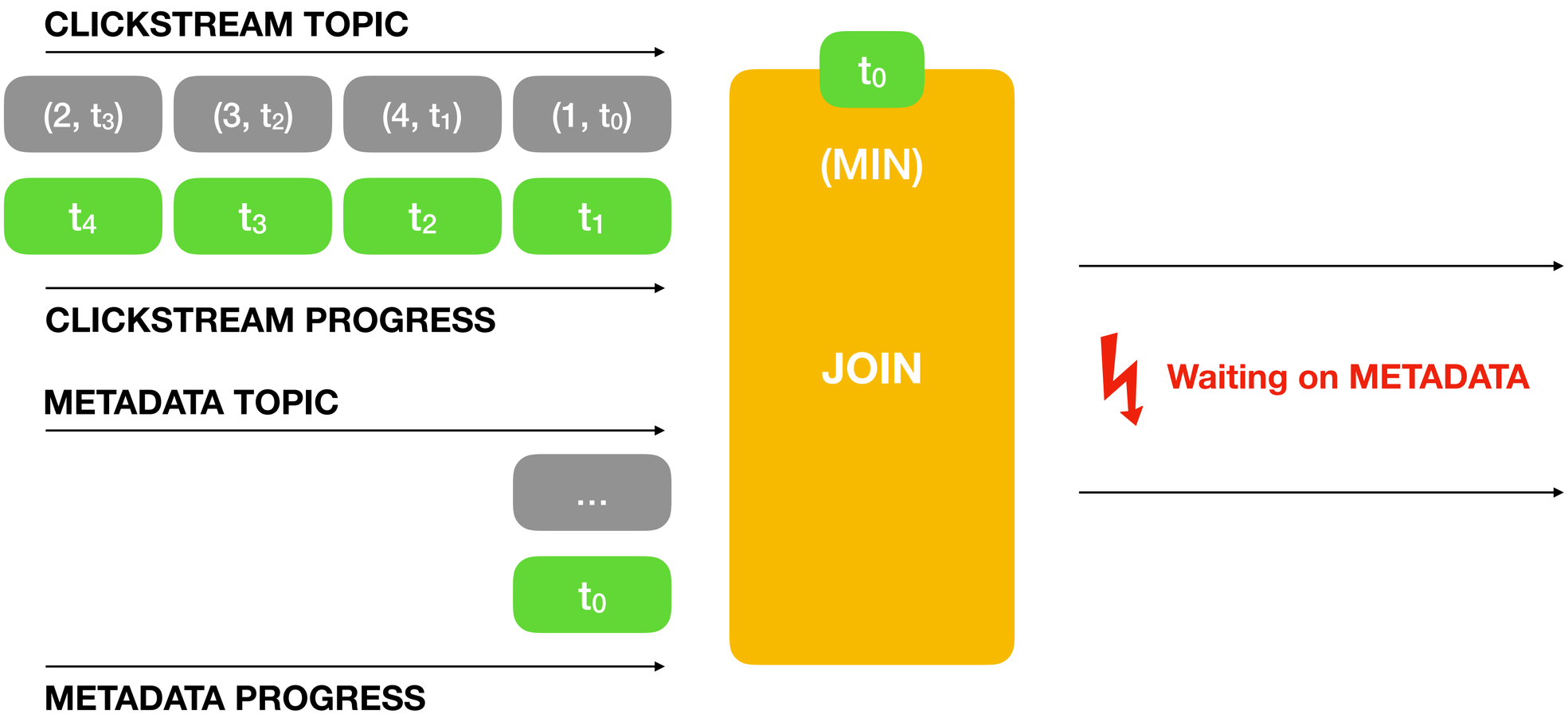


Correctness with Progress Tracking

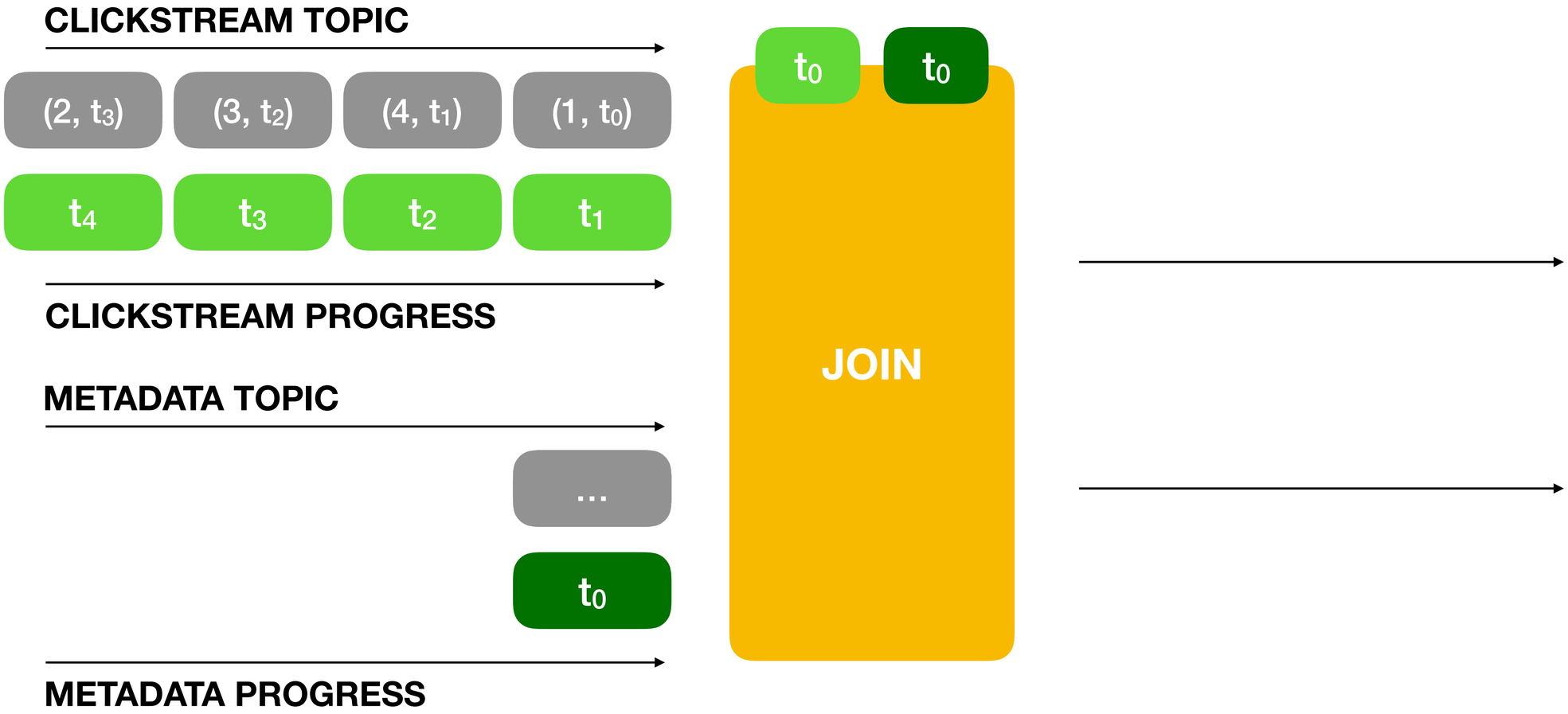


Progress Tracking… without Progress?

(data sources with different event frequencies)

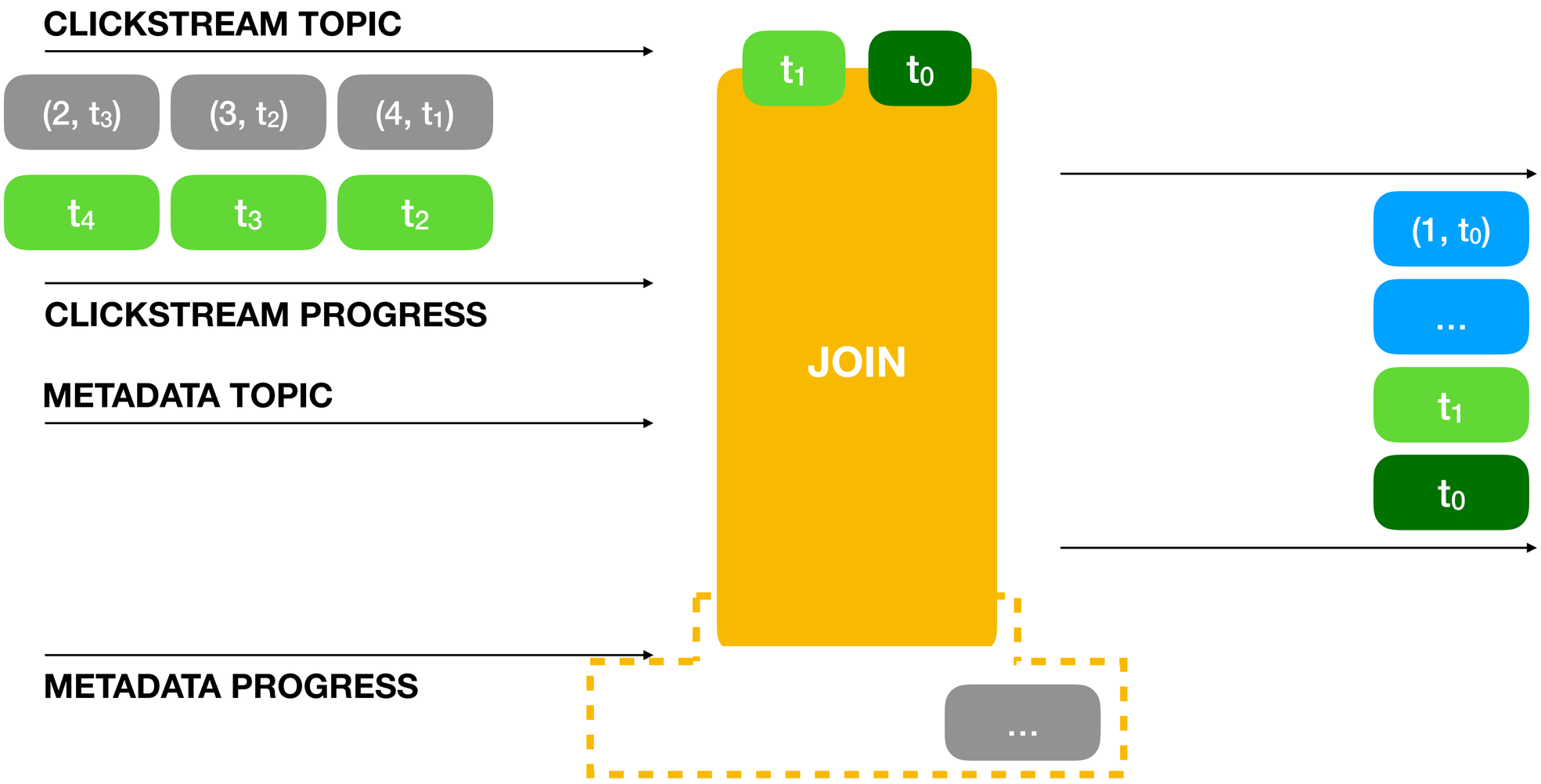


Multidimensional Progress Tracking
(track sources along independent timelines)



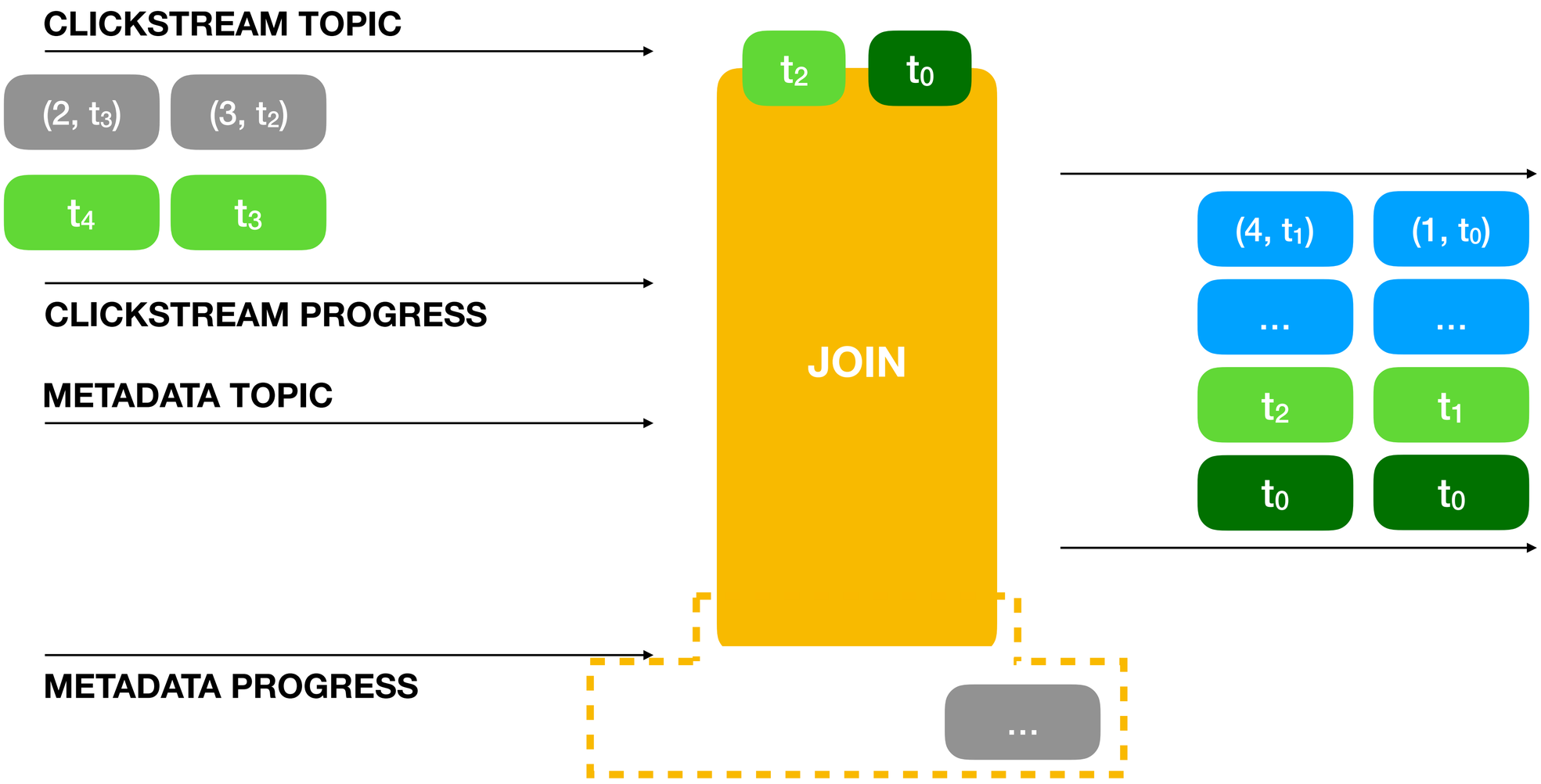
Multidimensional Progress Tracking

(track sources along independent timelines)



Multidimensional Progress Tracking

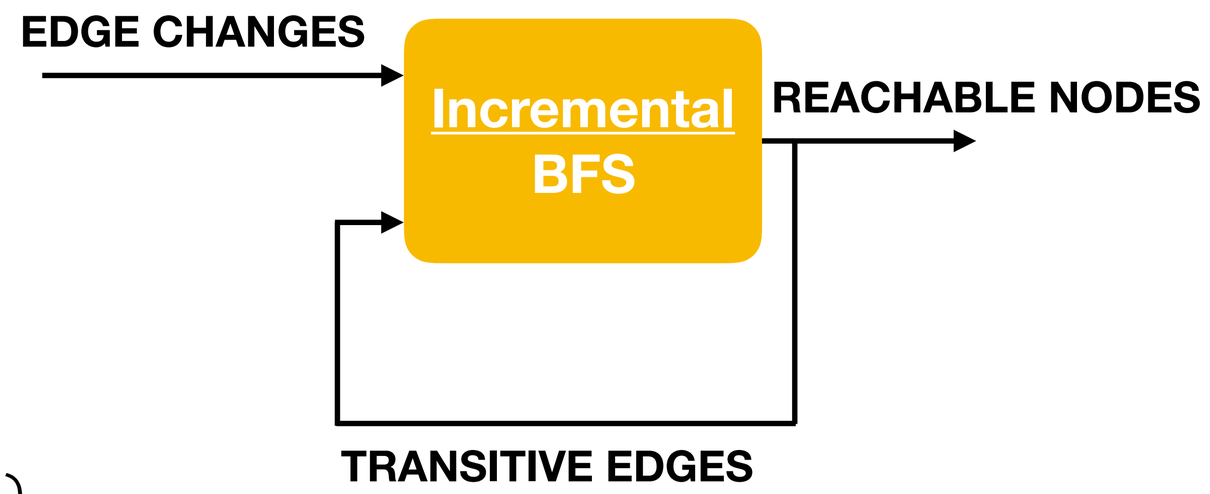
(track sources along independent timelines)



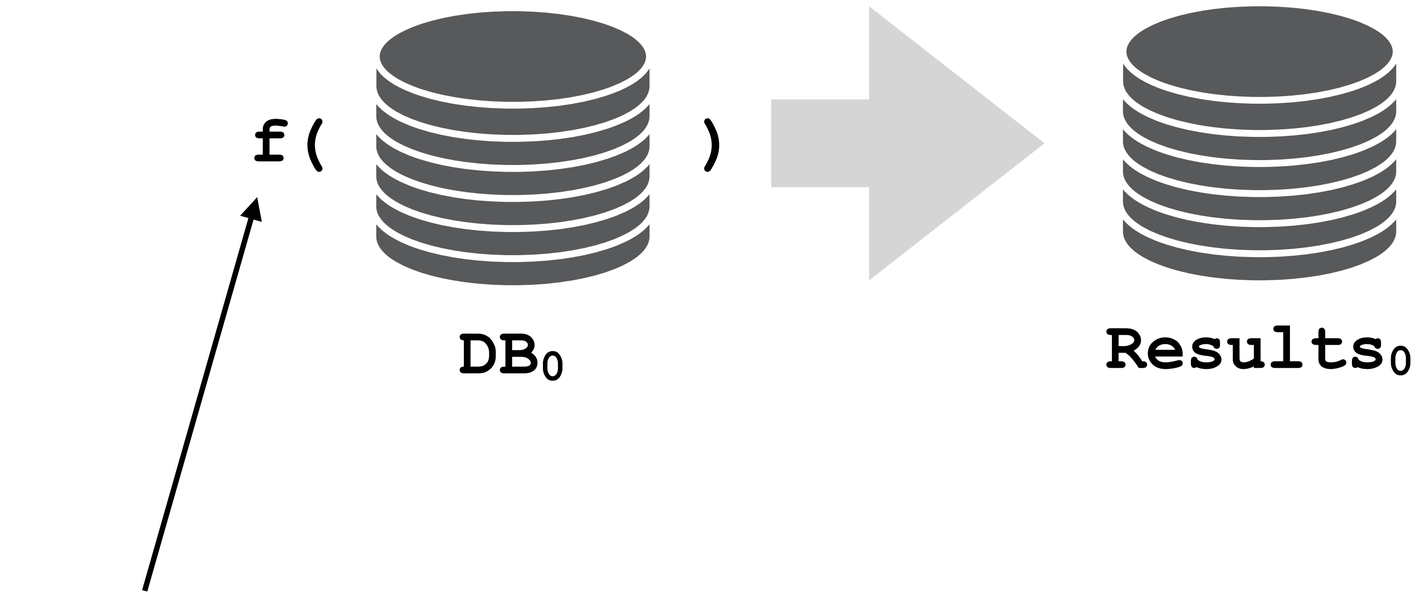
Differential Internals

Incremental BFS in Differential

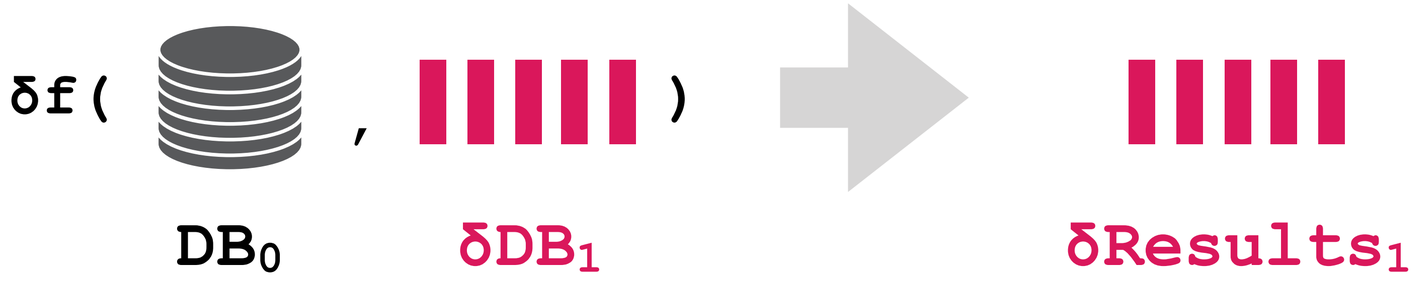
```
/// Friends of Friends

let nodes = roots.map(|x| (x, 0));

nodes.iterate(|inner| {

 let edges = edges.enter(&inner.scope());
 let nodes = nodes.enter(&inner.scope());

 inner.join(&edges, |_k,l,d| (*d, l+1))
     .concat(&nodes)
     .reduce(|_, s, t| t.push((*s[0].0, 1)))
})
```



Computing w/ Snapshots



Incremental Computation



How do we find ∂f?

Naive δquery

Implementation Details

total orders

"happened before"

$\leq$	0	1	2
0	✓	✓	✓
1		✓	✓
2			✓

Implementation Details

product partial orders

"happened before"

$(2\ 5) \leq (7\ 9)$

$(2\ 5) \leq (3\ 5)$

Implementation Details

product partial orders

"happened before"

$$(2\ 5) \leq (7\ 9)$$

$$(2\ 5) \leq (3\ 5)$$

$$(2\ 5) \not\leq (1\ 9)$$

$$(1\ 9) \not\leq (2\ 5)$$

no order!

Implementation Details

product partial orders

"happened before"

$$(2\ 5) \leq (7\ 9)$$

$$(2\ 5) \leq (3\ 5)$$

$$(2\ 5) \not\leq (1\ 9)$$

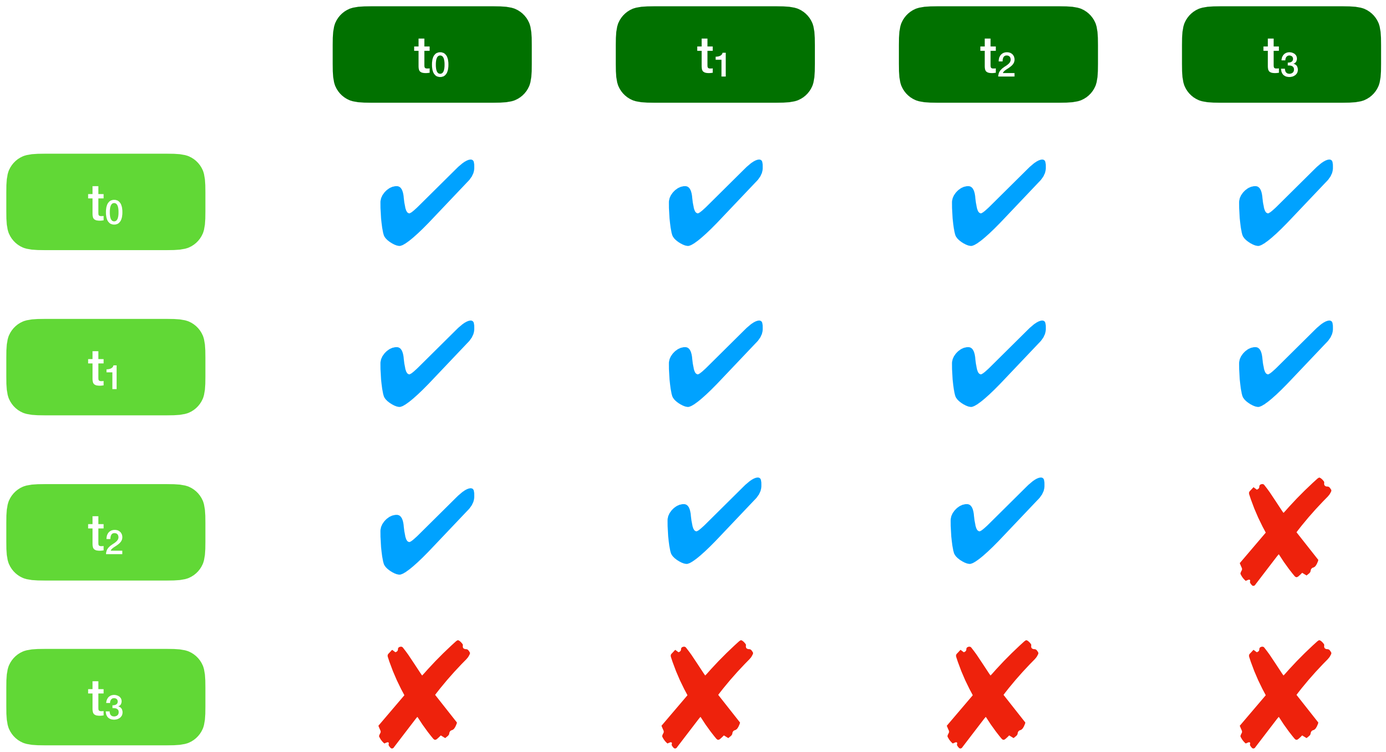
$$(1\ 9) \not\leq (2\ 5)$$

$$(1000\ 0) \not\leq (0\ 1)$$

no order!

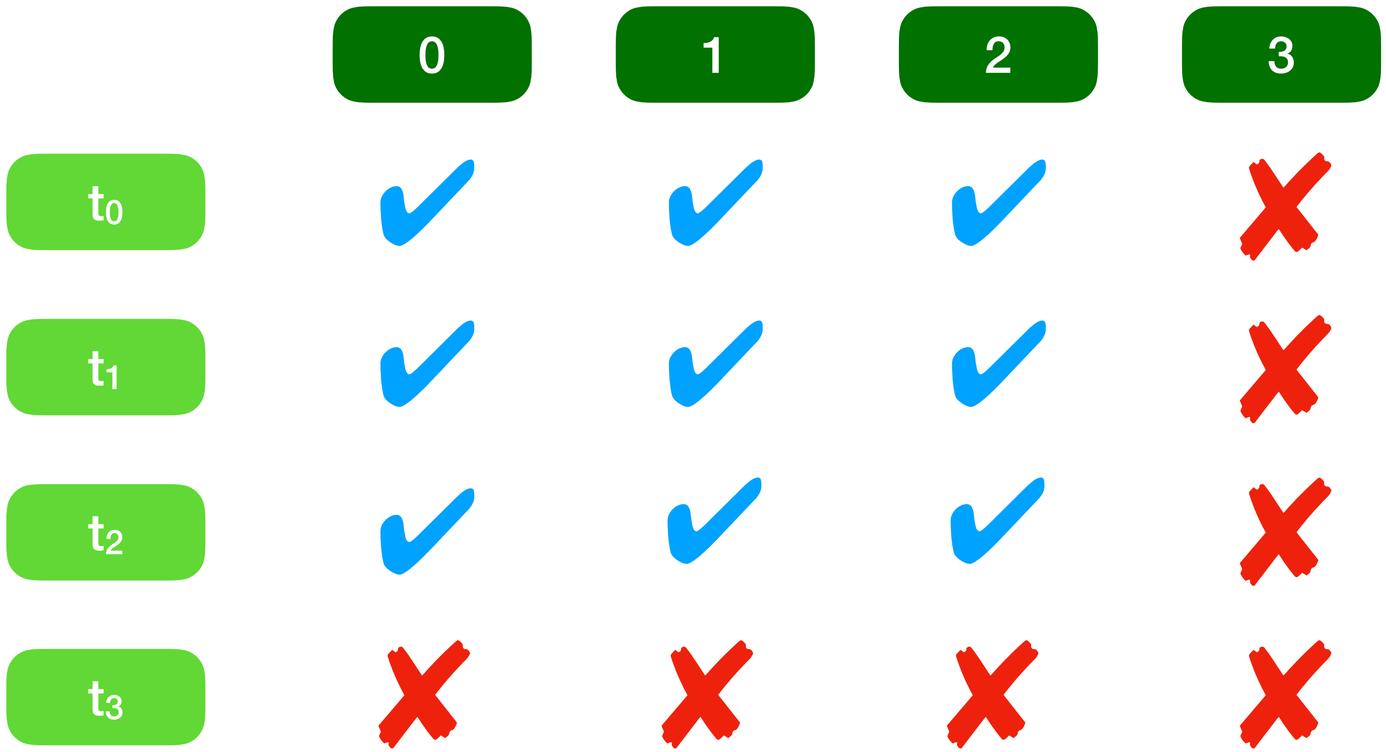
Lexicographical Total Order

(visibility for t_2 t_2)



Product Partial Order

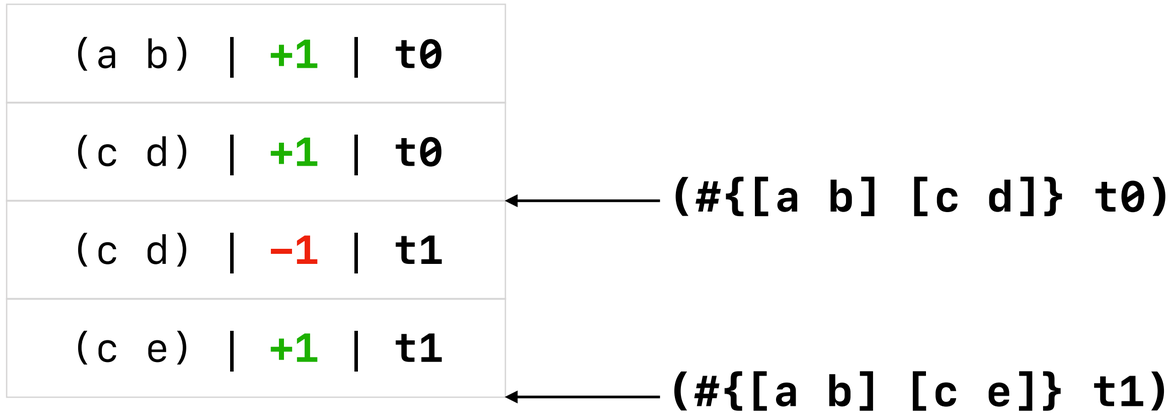
(visibility for t_2 2)



```
/// BFS

let nodes = roots.map(|x| (x, 0));

nodes.iterate(|inner| {

 let edges = edges.enter(&inner.scope());
 let nodes = nodes.enter(&inner.scope());

 inner.join_map(&edges, |_k,l,d| (*d, l+1))
     .concat(&nodes)
     .group(|_, s, t| t.push((*s[0].0, 1)))
})
```

Persistent Collections

:edge



Totally-ordered Times

(bfs :edge a)

(a b) | +1 | t0		
(c d) | +1 | t0		
(b c) | +1 | t1	(a c) | +1 | t1.1	(a d) | +1 | t1.2
		(c d) | −1 | t2

Partially-ordered Times

(bfs :edge a)

(a b)	+1	(t0 0)			
(c d)	+1	(t0 0)			
(b c)	+1	**(t1 0)**	(a c)	+1	**(t1 1)**	(a d)	+1	**(t1 2)**
(c d)	−1	(t2 0)			

Partially-ordered Times

(bfs :edge a)

(a b)	+1	(t0 0)						
(c d)	+1	(t0 0)						
(b c)	+1	(t1 0)	(a c)	+1	(t1 1)	(a d)	+1	**(t1 2)**
(c d)	−1	(t2 0)						

Partially-ordered Times

(bfs :edge a)

(a b)	+1	(t0 0)			
(c d)	+1	(t0 0)			
(b c)	+1	(t1 0)	(a c)	+1	(t1 1)	(a d)	+1	(t1 2)
(c d)	−1	(t2 0)			

Partially-ordered Times

(bfs :edge a)

(a b) | +1 | (t0 0)		
(c d) | +1 | (t0 0)		
(b c) | +1 | (t1 0)	(a c) | +1 | (t1 1)	(a d) | +1 | (t1 2)
(c d) | −1 | (t2 0)	–	

Partially-ordered Times

(bfs :edge a)

(a b) | +1 | (t0 0)		
(c d) | +1 | (t0 0)		
(b c) | +1 | (t1 0)	(a c) | +1 | (t1 1)	(a d) | +1 | (t1 2)
(c d) | −1 | (t2 0)	–	(a d) | −1 | (t2 2)

Performance

Timely Dataflow Performance

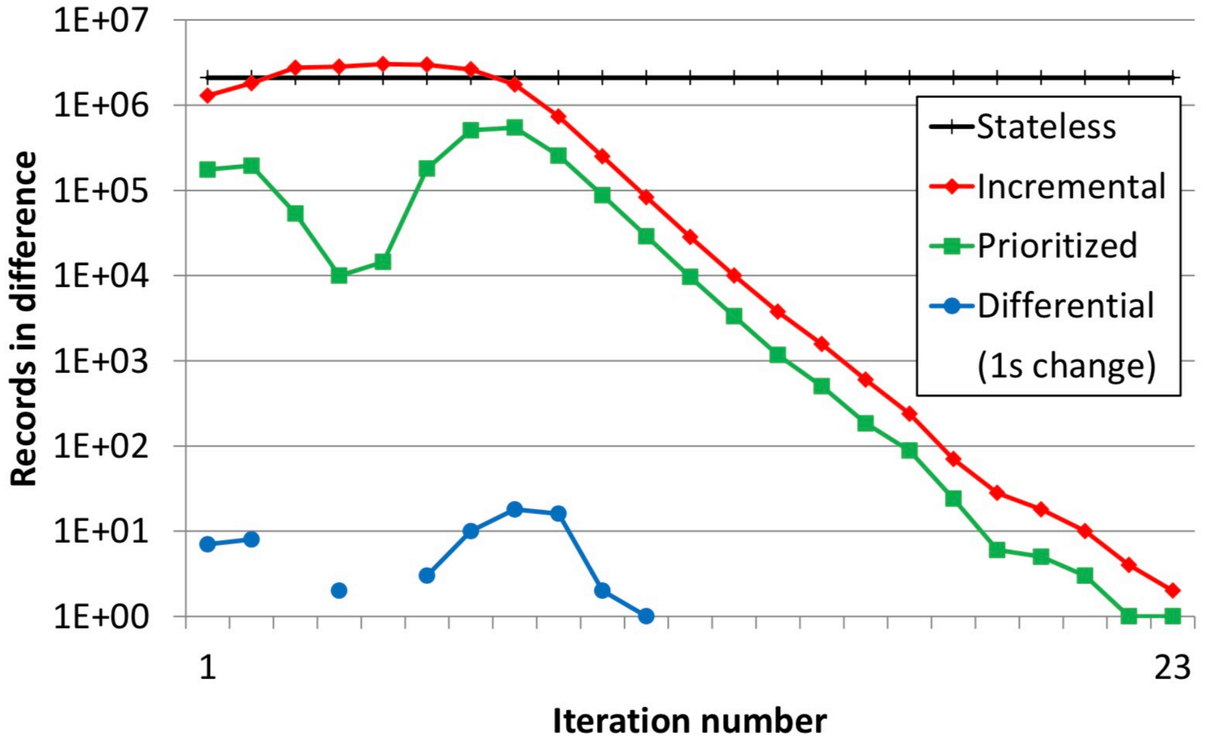
20xPR	cores	twitter_rv	uk_2007_05
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s
Laptop	1	**110s**	**256s**
Timely	128	**15s**	**19s**

McSherry, Frank; Isard, Michael; Murray, Derek G. (2015): „Scalability! But at what COST?“. In: *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association.

Differential Dataflow Performance

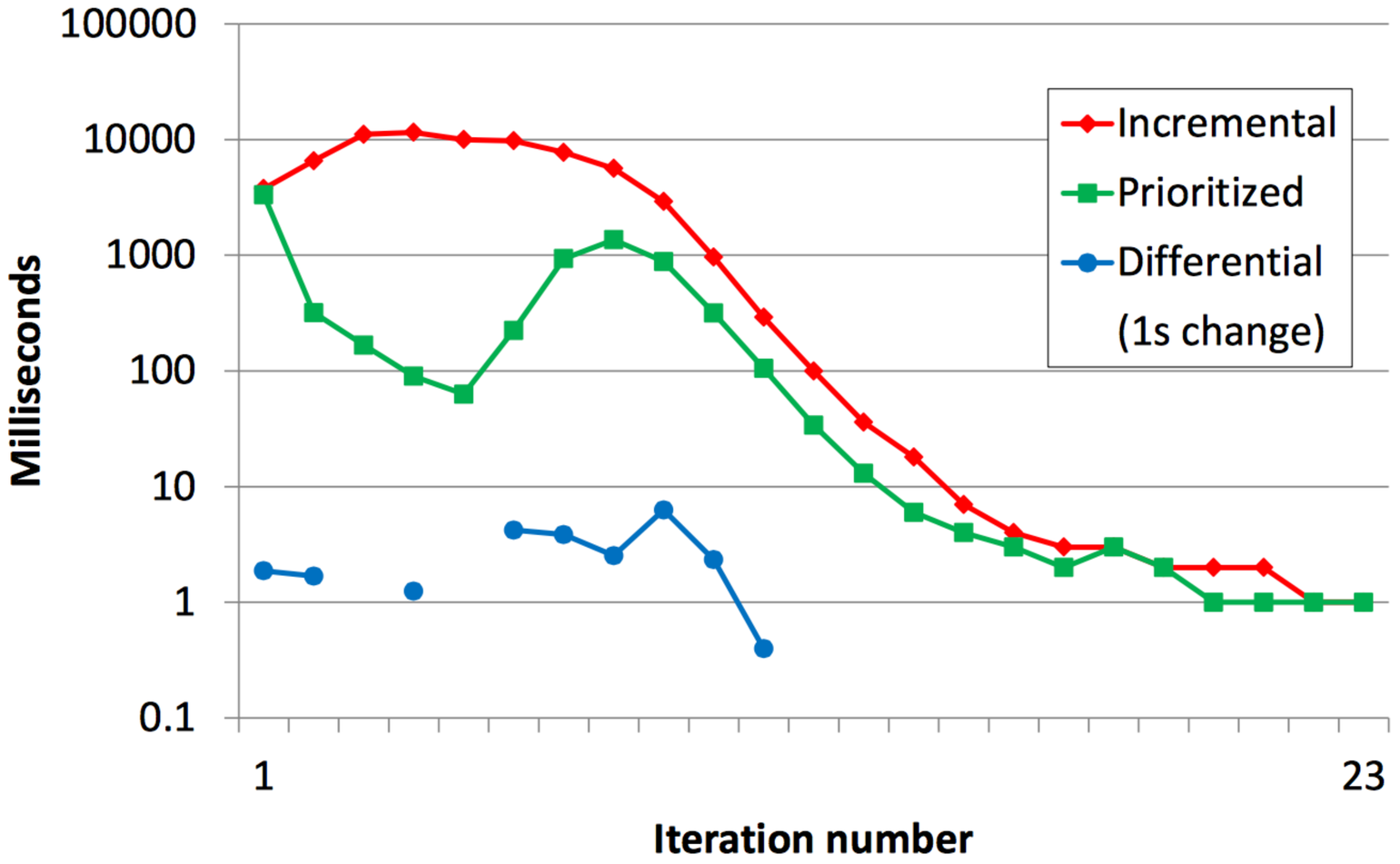
Connected	cores	livejournal	orkut
GraphX	128	59s	53s
SociaLite	128	54s	78s
Myria	128	37s	57s
BigDatalog	128	27s	33s
Differential	1, 2	**20s, 11s**	**43s, 26s**
update	1, 2	**98us, 109us**	**200us, 216us**

McSherry, F., Lattuada, A., & Schwarzkopf, M., K-pg: shared state in differential dataflows



Computation	Full	Update
gnp1	9.45s	18.29ms

McSherry, F., Lattuada, A., & Schwarzkopf, M., K-pg: shared state in differential dataflows



McSherry, F., Lattuada, A., & Schwarzkopf, M., K-pg: shared state in differential dataflows

Sources

Repositories

- Timely & Differential: github.com/TimelyDataflow
- ST2: github.com/li1/snailtrail, github.com/li1/st2-lite

Papers

- Naiad (Timely Dataflow): http://dl.acm.org/citation.cfm?doid=2517349.2522738
- Differential Dataflow: http://michaelisard.com/pubs/differentialdataflow.pdf, arxiv.org/abs/1812.02639
- SnailTrail: hdl.handle.net/20.500.11850/228581