It’s About Time: An Introduction to Timely Dataflow

Data Council, October ‘19

clockworks

Malte Sandstede

malte@clockworks.io / @MalteSandstede

Nikolas Göbel

niko@clockworks.io / @NikolasGoebel

David Bach

david@clockworks.io

Moritz Moxter

moritz@clockworks.io

In collaboration with:

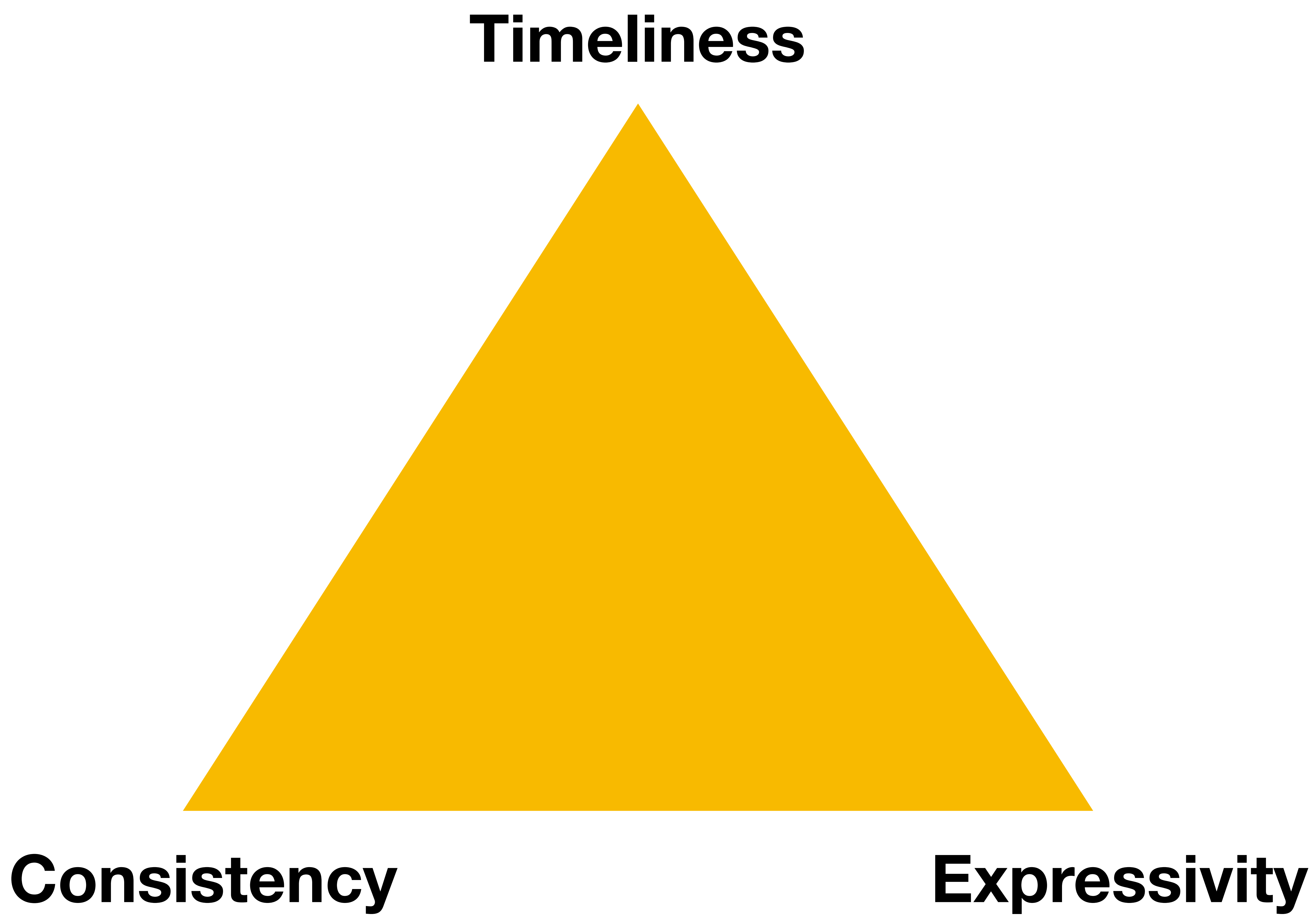
Frank McSherry

Vasia Kalavri (ETH)

ETH zürich

Systems Group

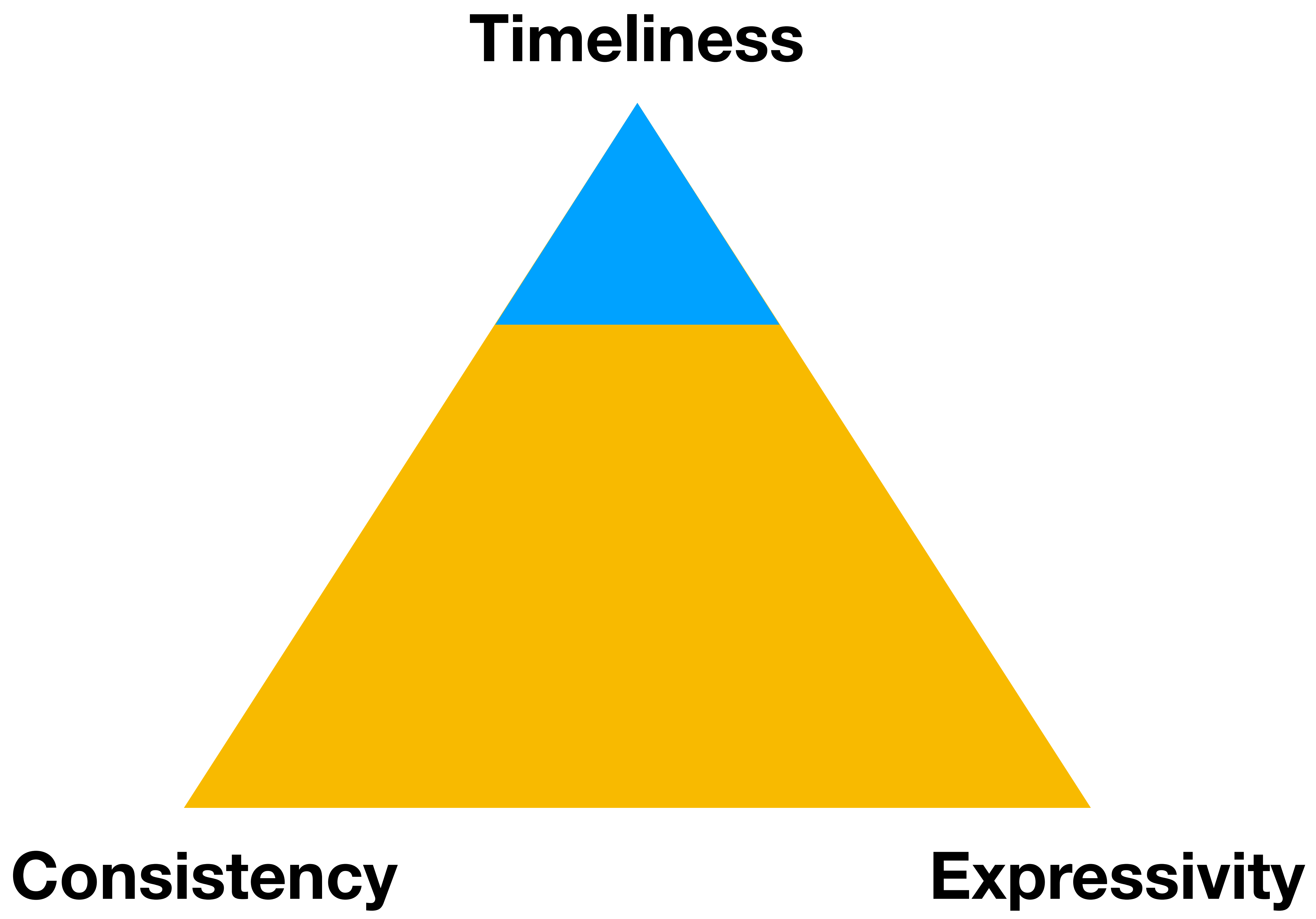
Stream Processing's Trifecta



Stream Processing's Trifecta

Naive Stateless Processing

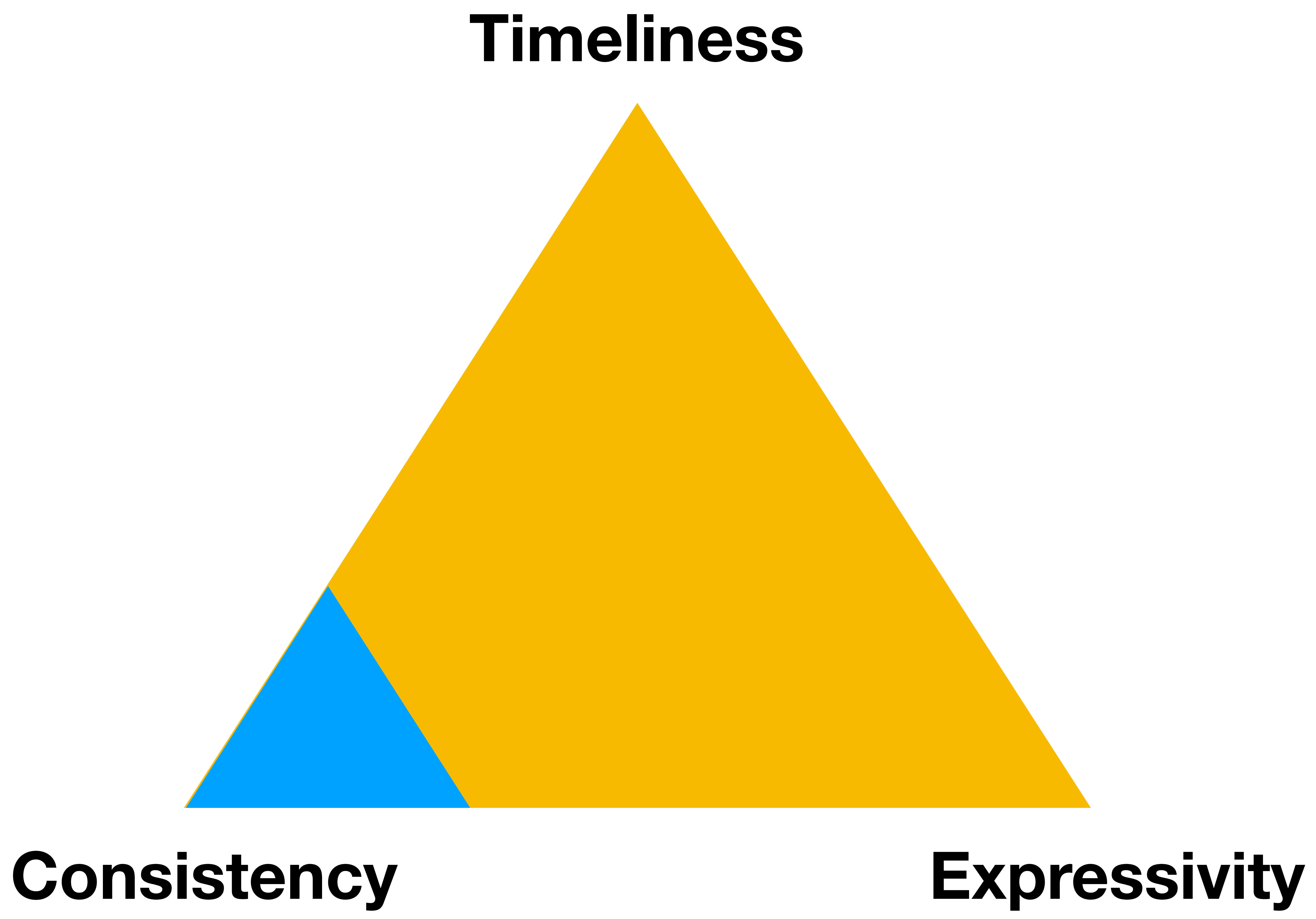
- Low latency
- Issue: Late arrivals
- Issue: Complex computations



Stream Processing's Trifecta

MapReduce

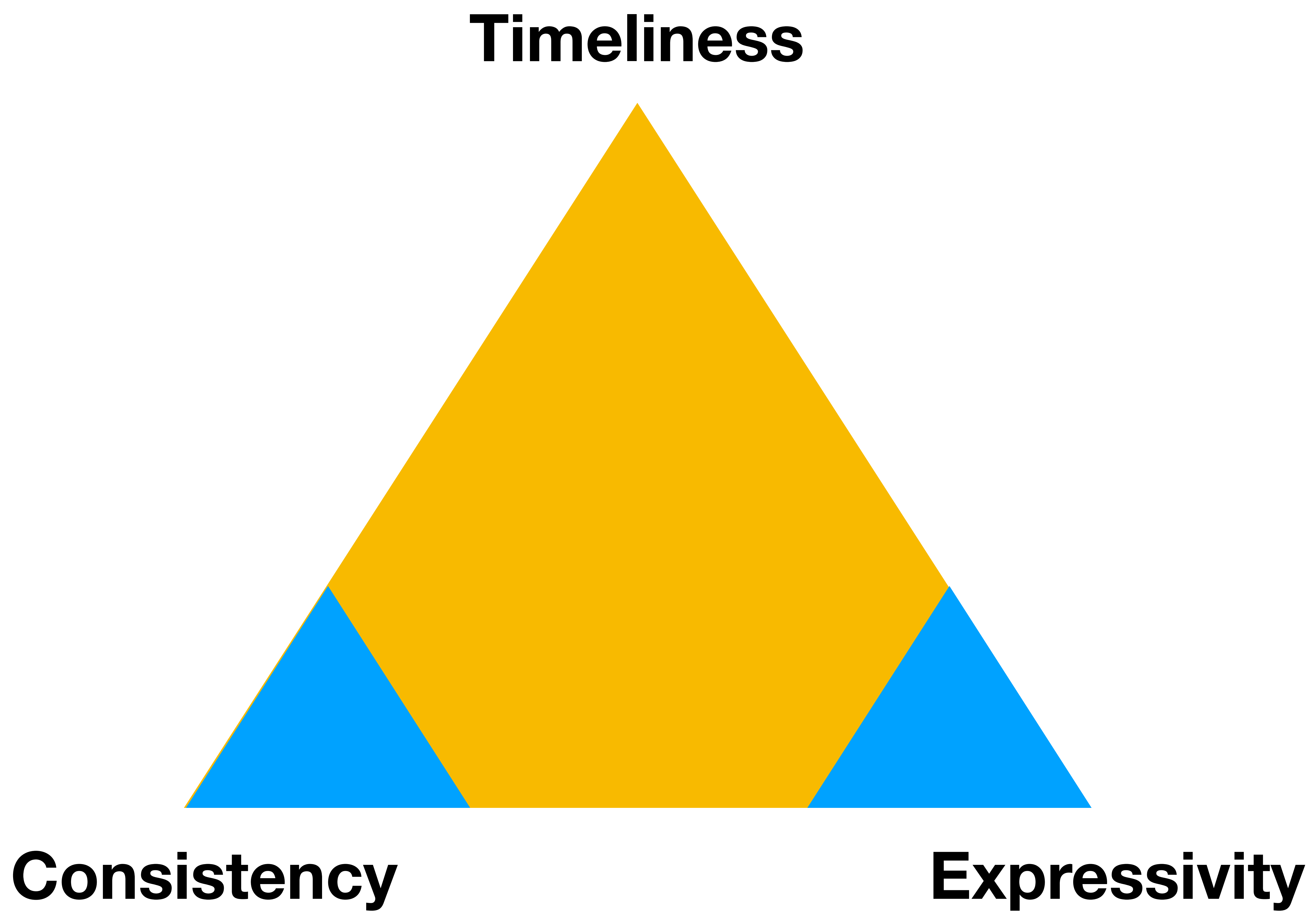
- No late arrivals (by definition)
- Easy to scale
- Issue: Complex computations
- Issue: High latency



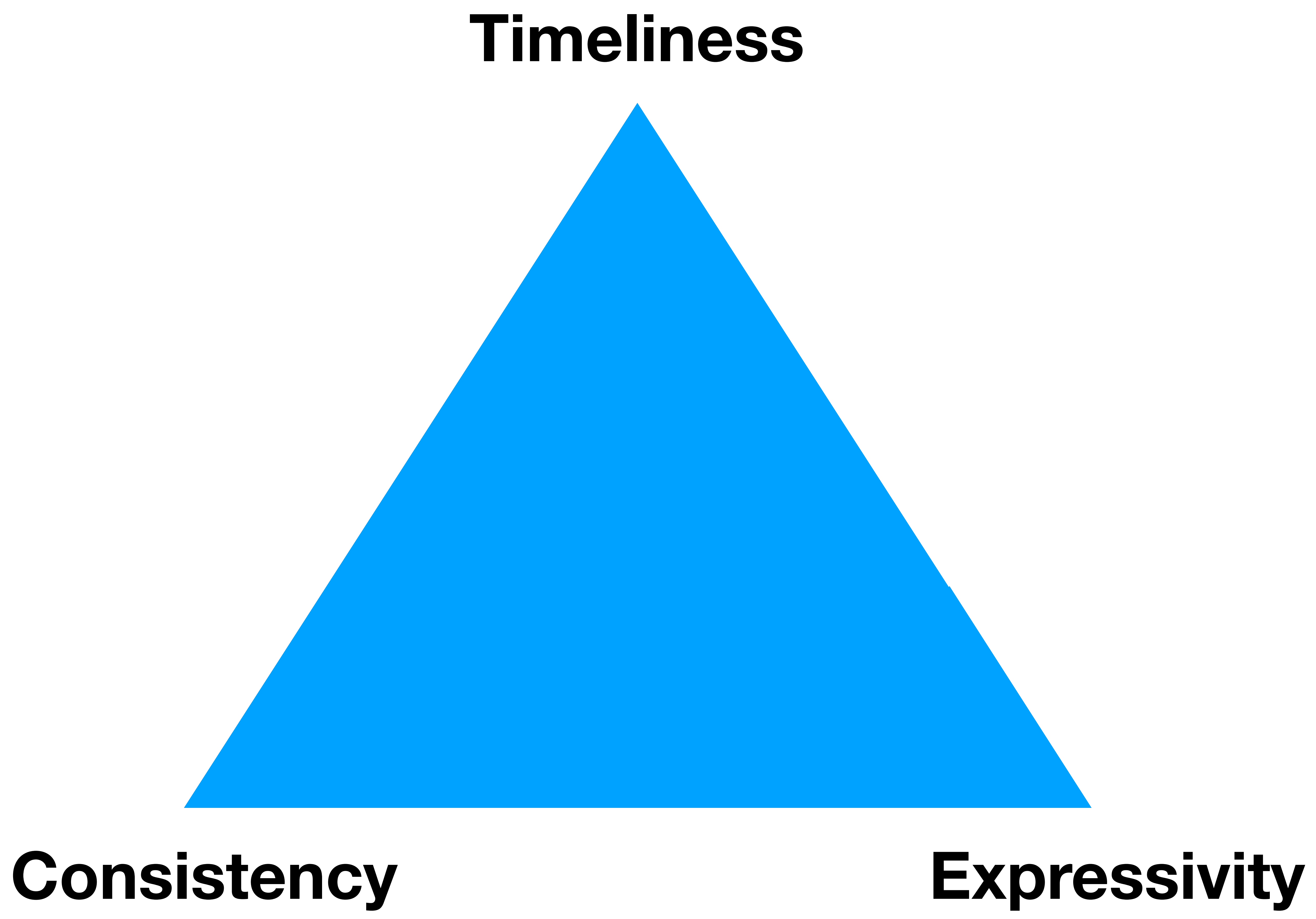
Stream Processing's Trifecta

Database

- No late arrivals
- High expressivity
- ACID
- Issue: Not realtime!

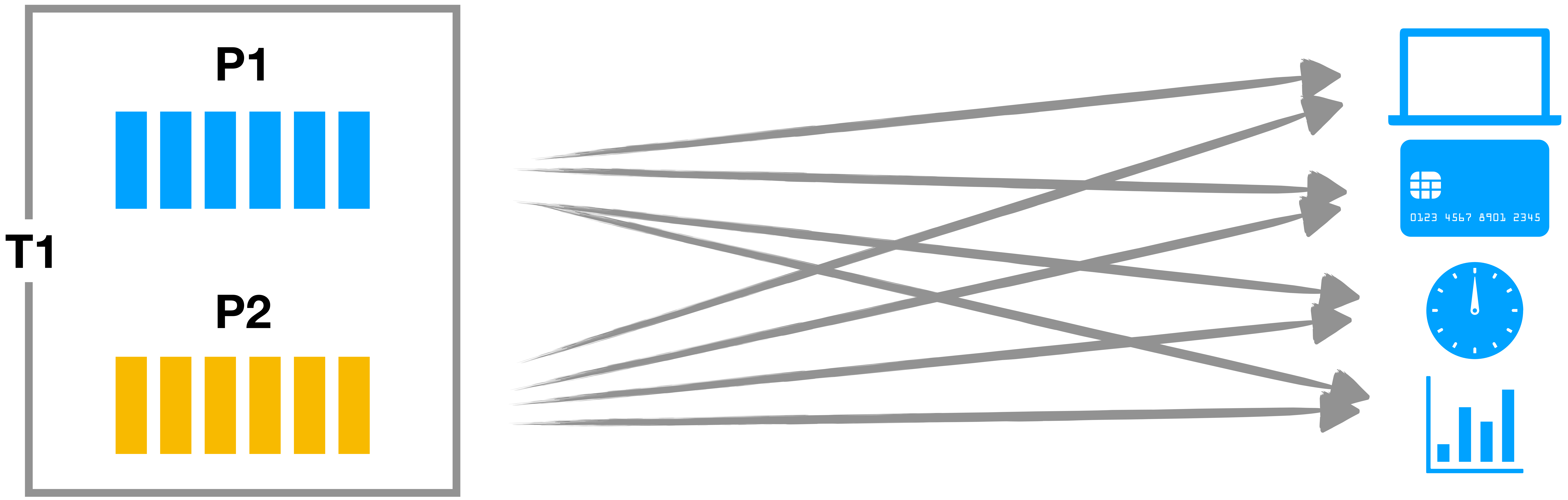


Stream Processing’s Trifecta

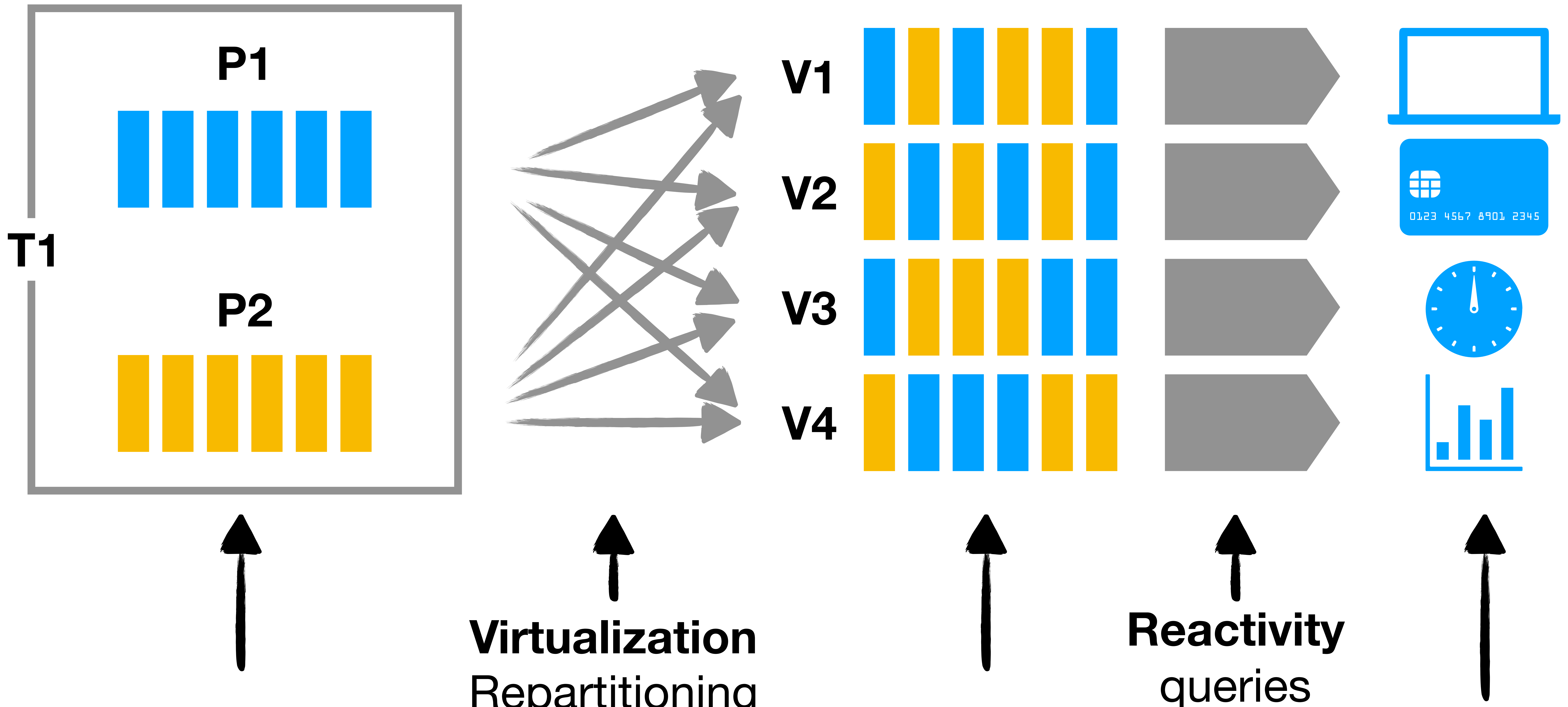


Use Case: Kafka Superpowers

(Partitions complect physical representation & use case)



Use Case: Kafka Superpowers



Physical Representation

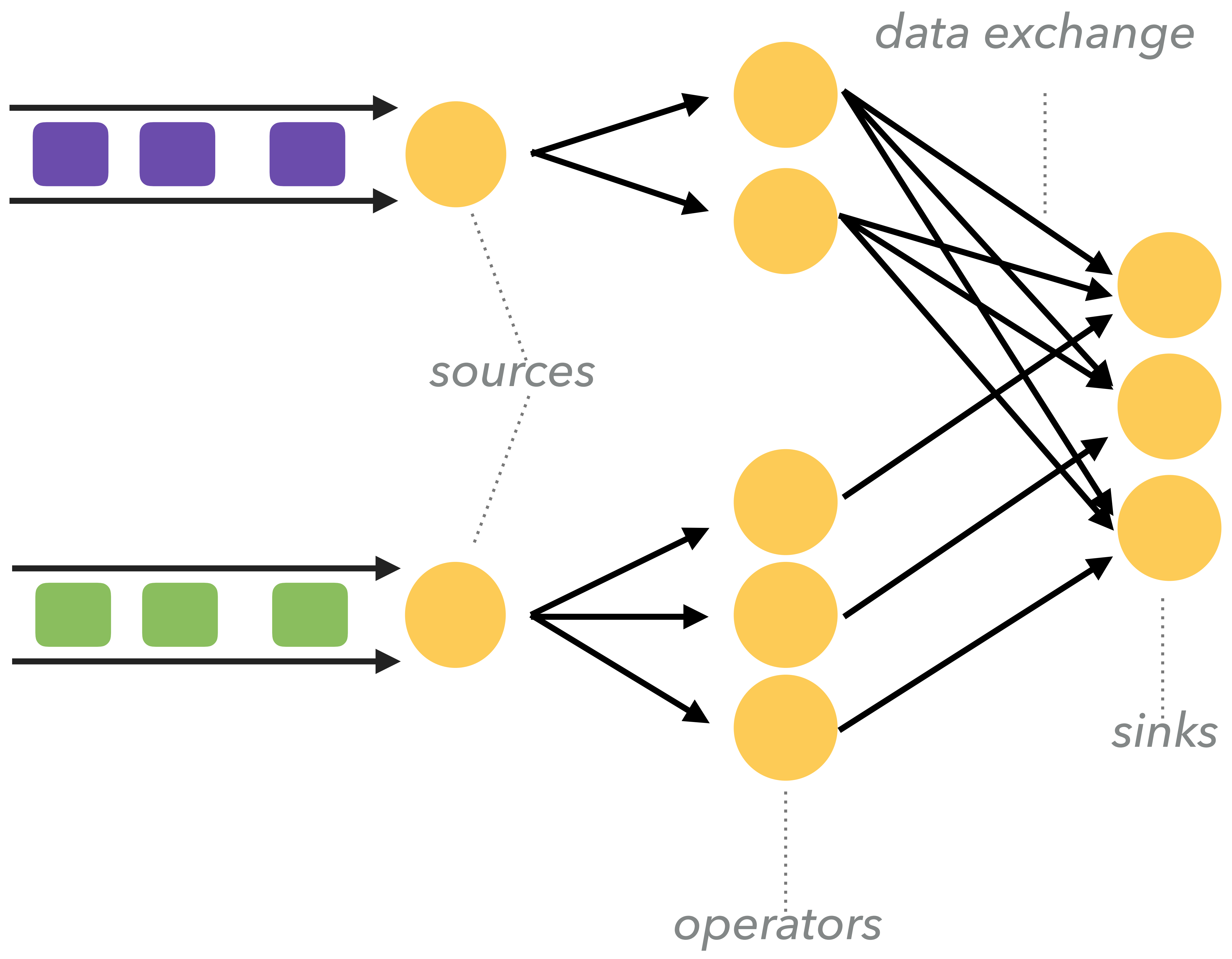
Virtualization
Repartitioning
Joins

Virtual Partitions
time order

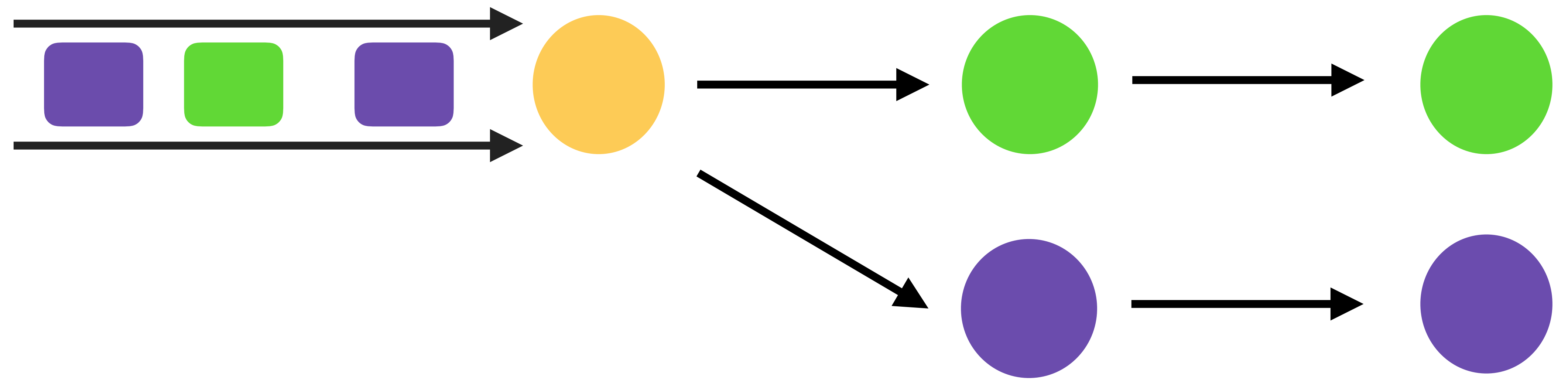
Reactivity
queries

Business Logic

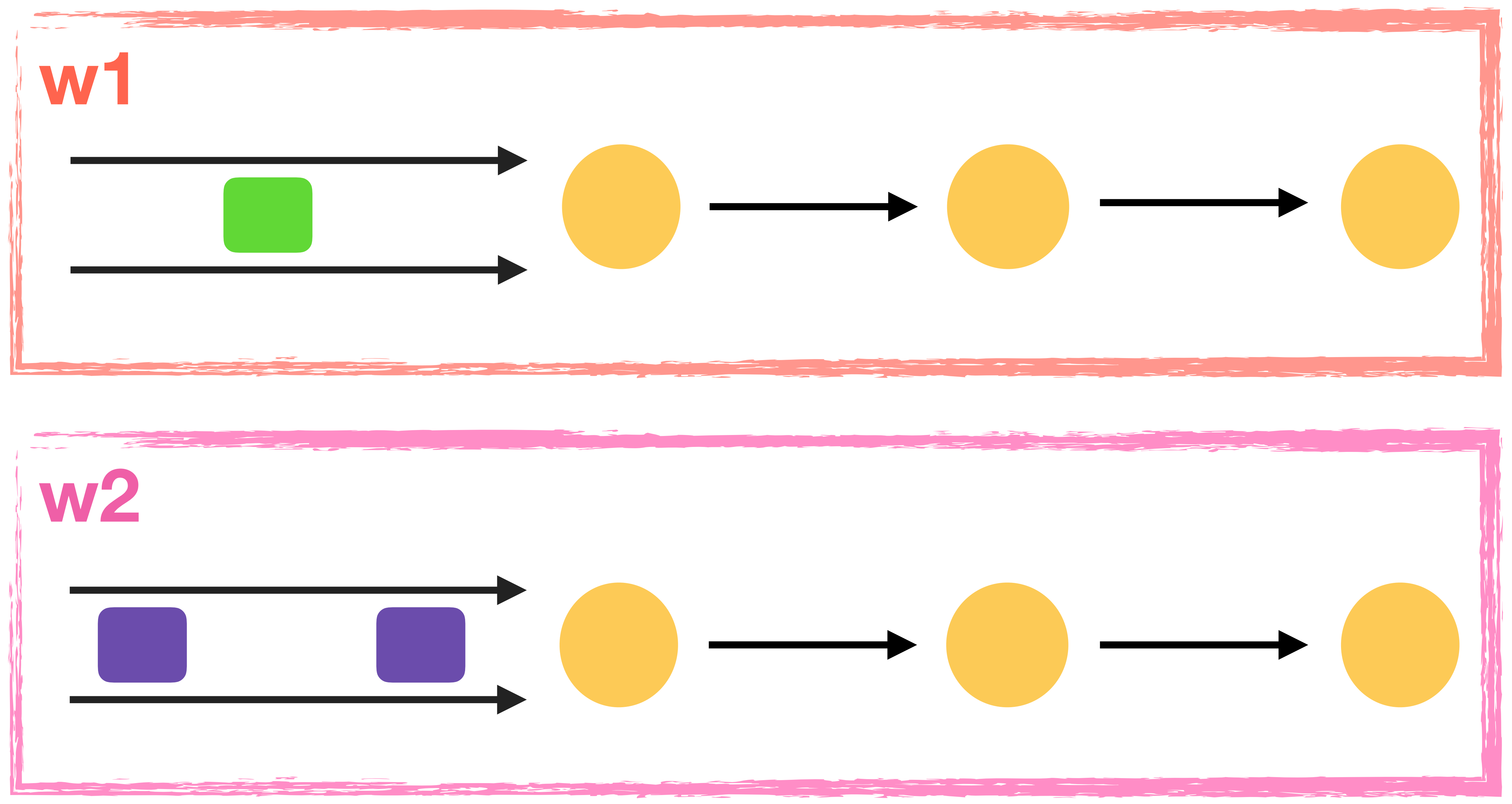
Stream Processing as Dataflow



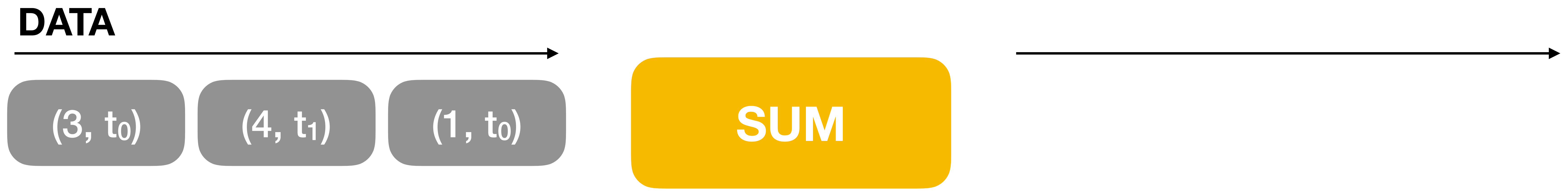
Dataflow Parallelism



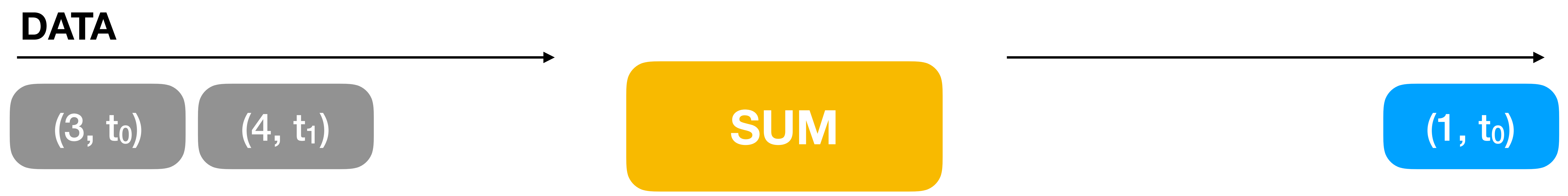
Dataflow Distribution



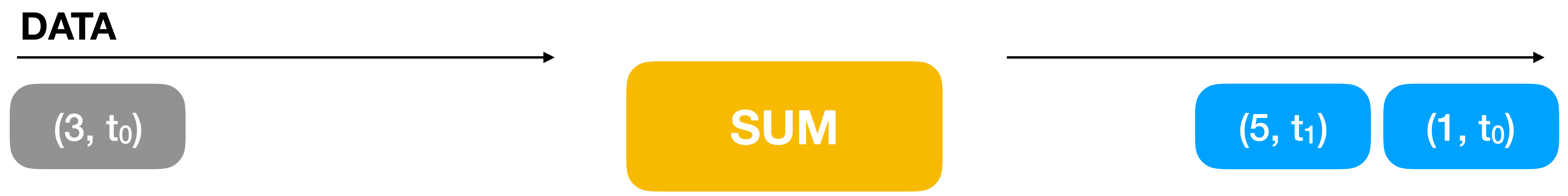
Correctness Troubles



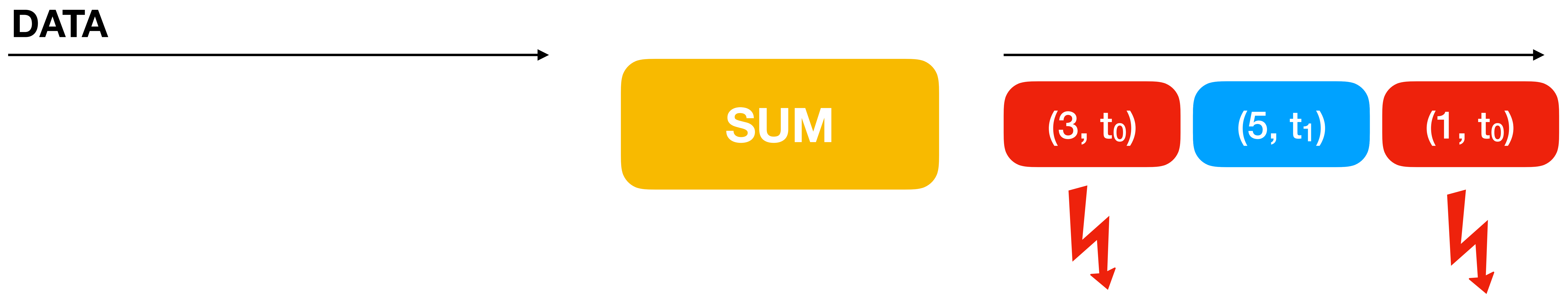
Correctness Troubles



Correctness Troubles



Correctness Troubles

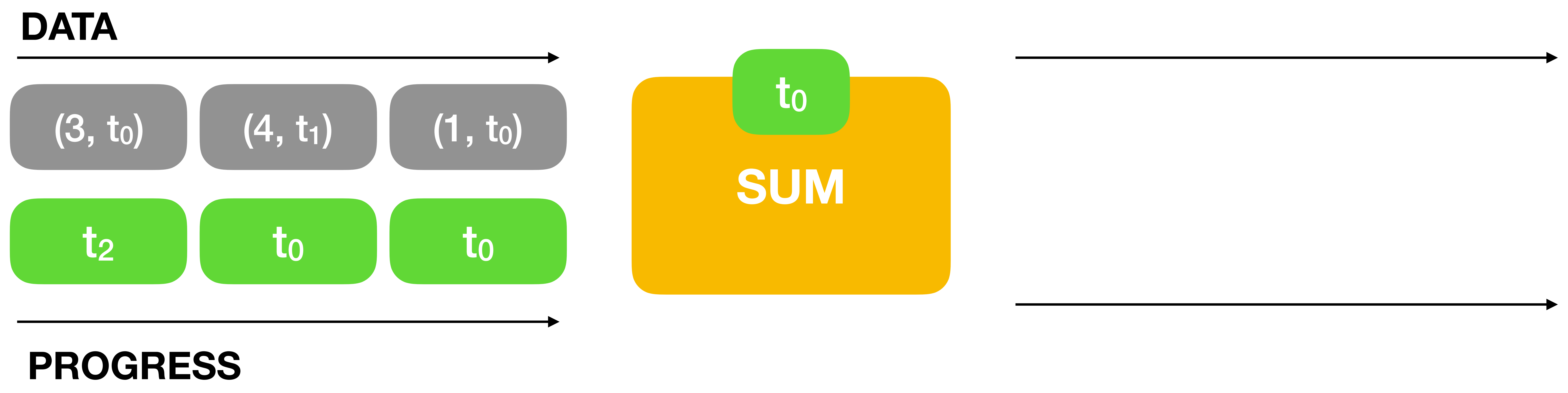


Timely Dataflow

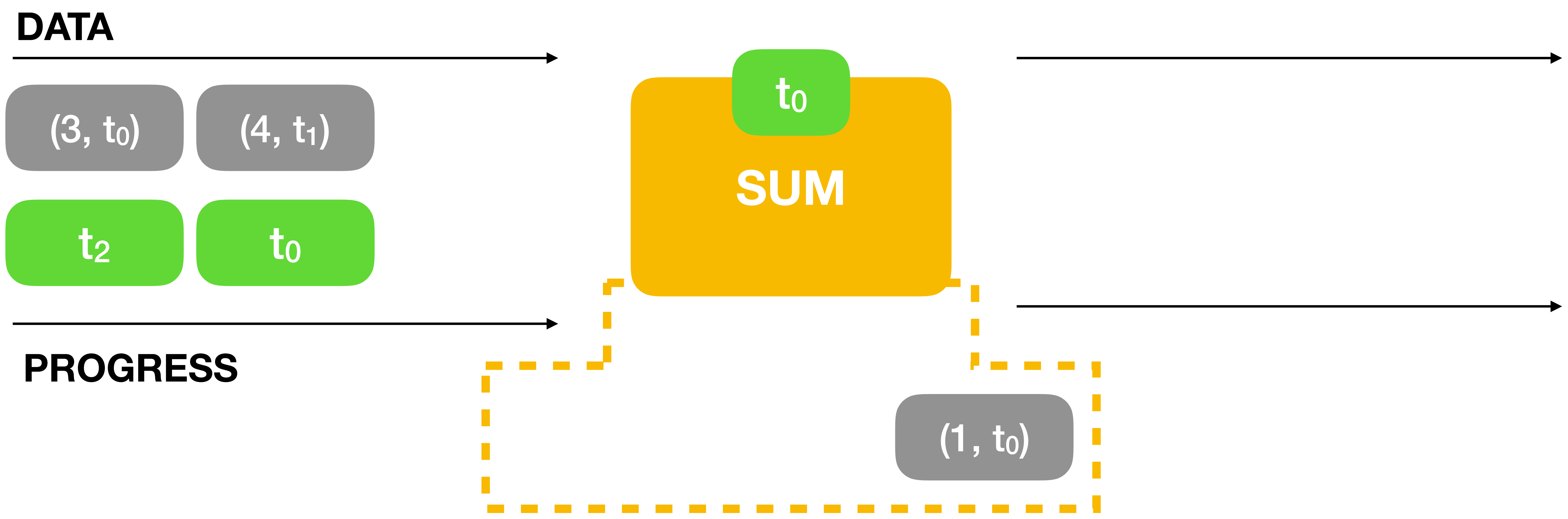
A low-latency runtime for
distributed cyclic dataflows

github.com/**TimelyDataflow**

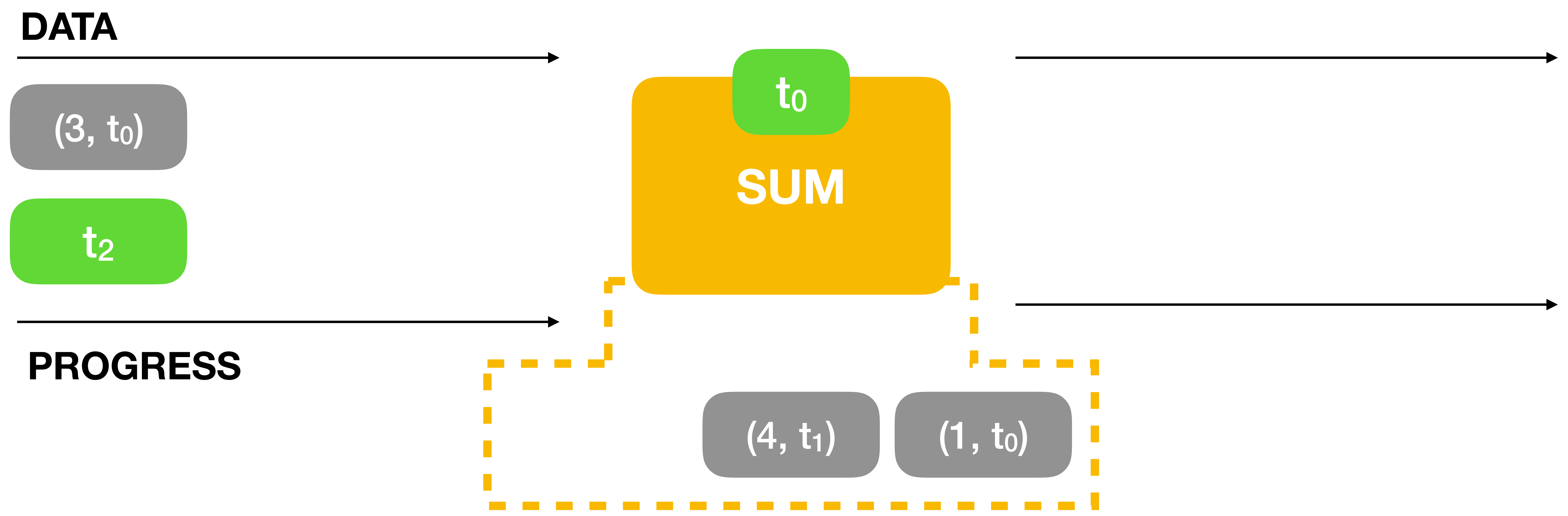
Correctness with Progress Tracking



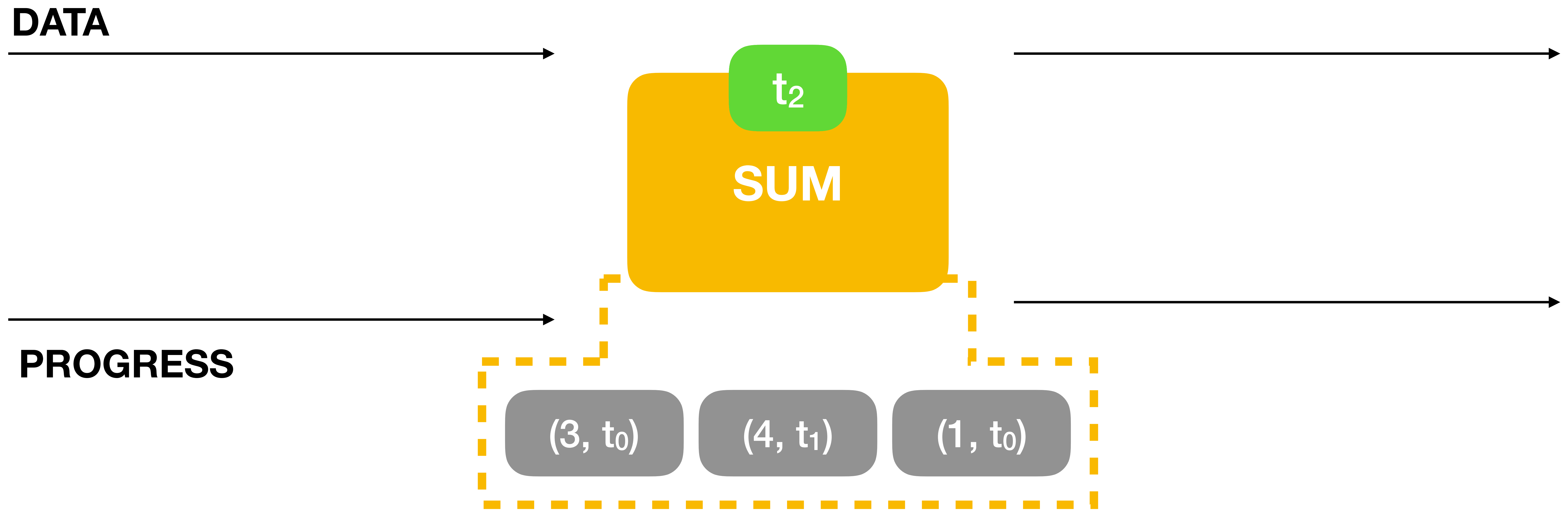
Correctness with Progress Tracking



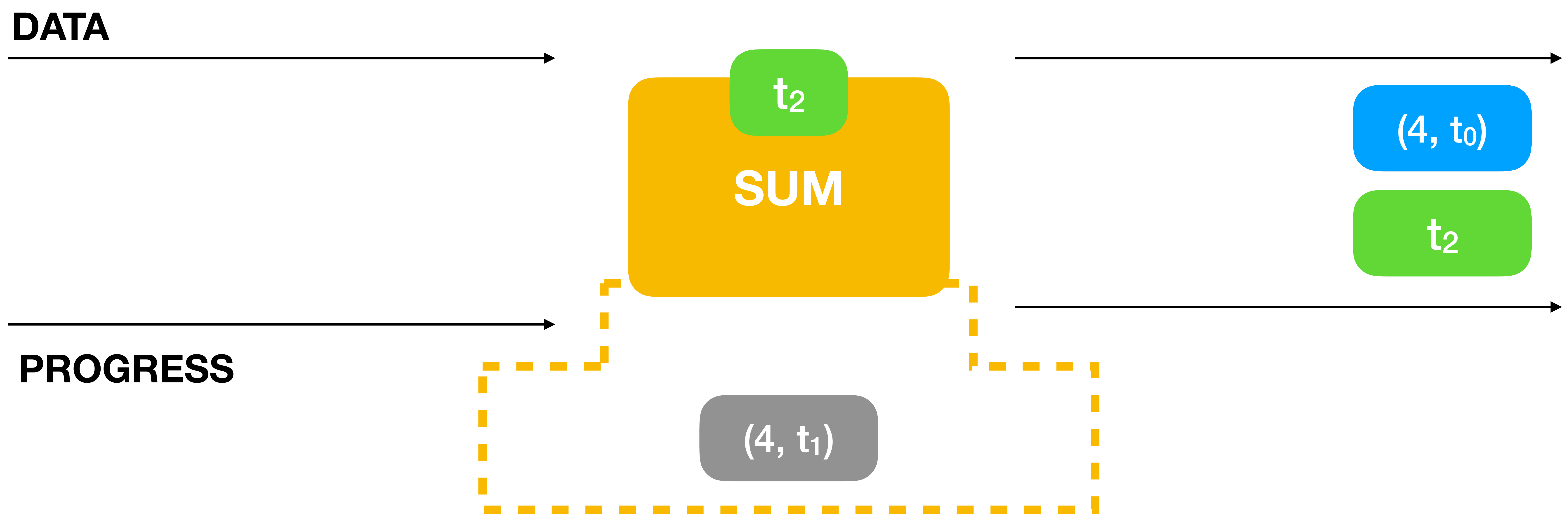
Correctness with Progress Tracking



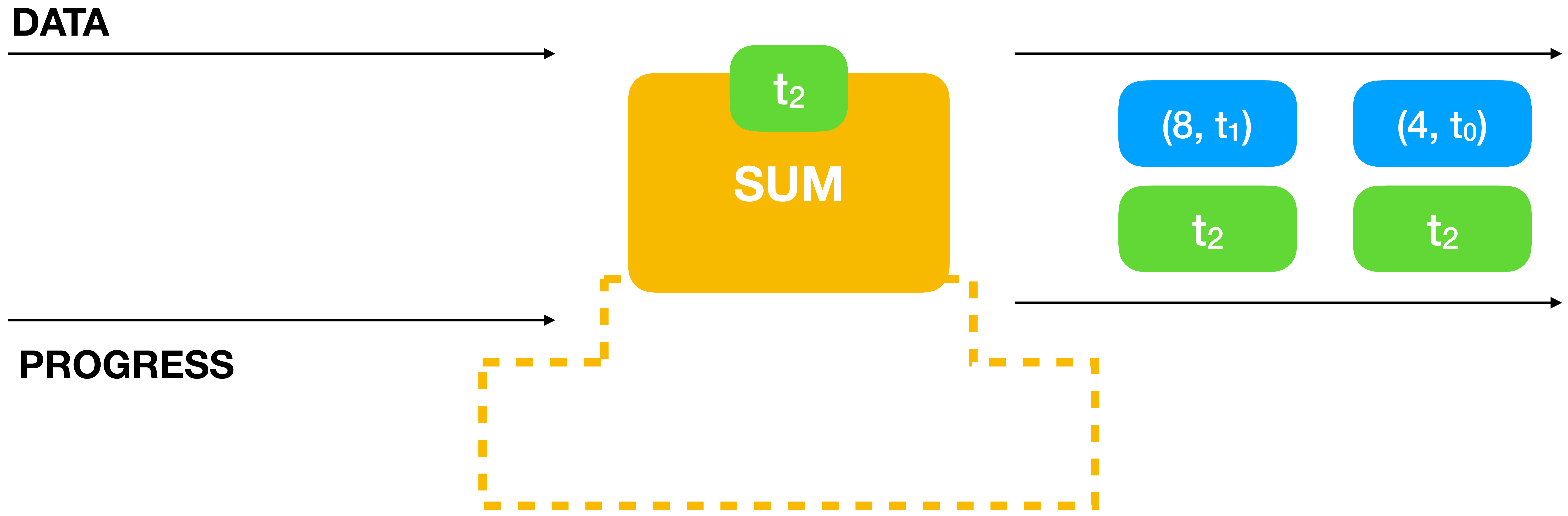
Correctness with Progress Tracking



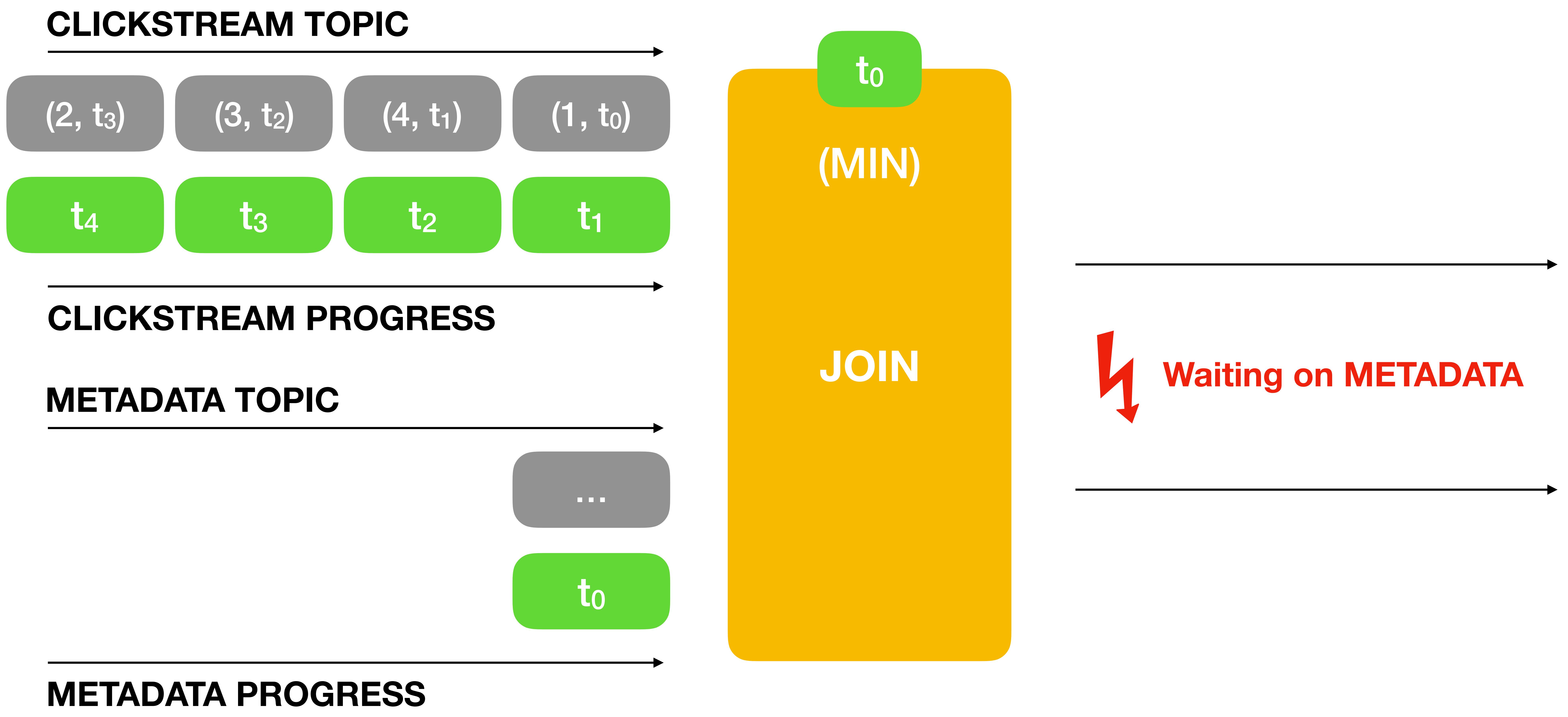
Correctness with Progress Tracking



Correctness with Progress Tracking

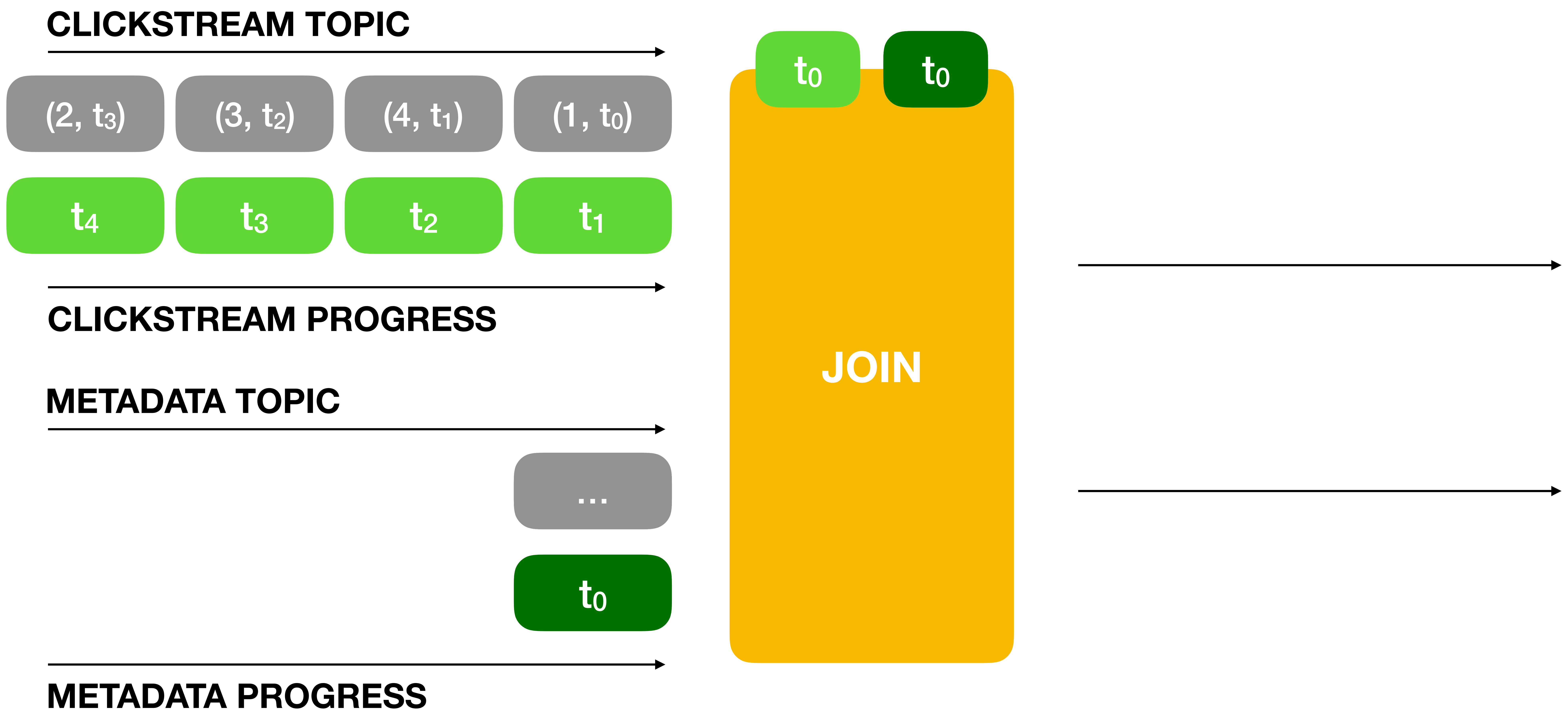


Progress Tracking… without Progress?
(data sources with different event frequencies)



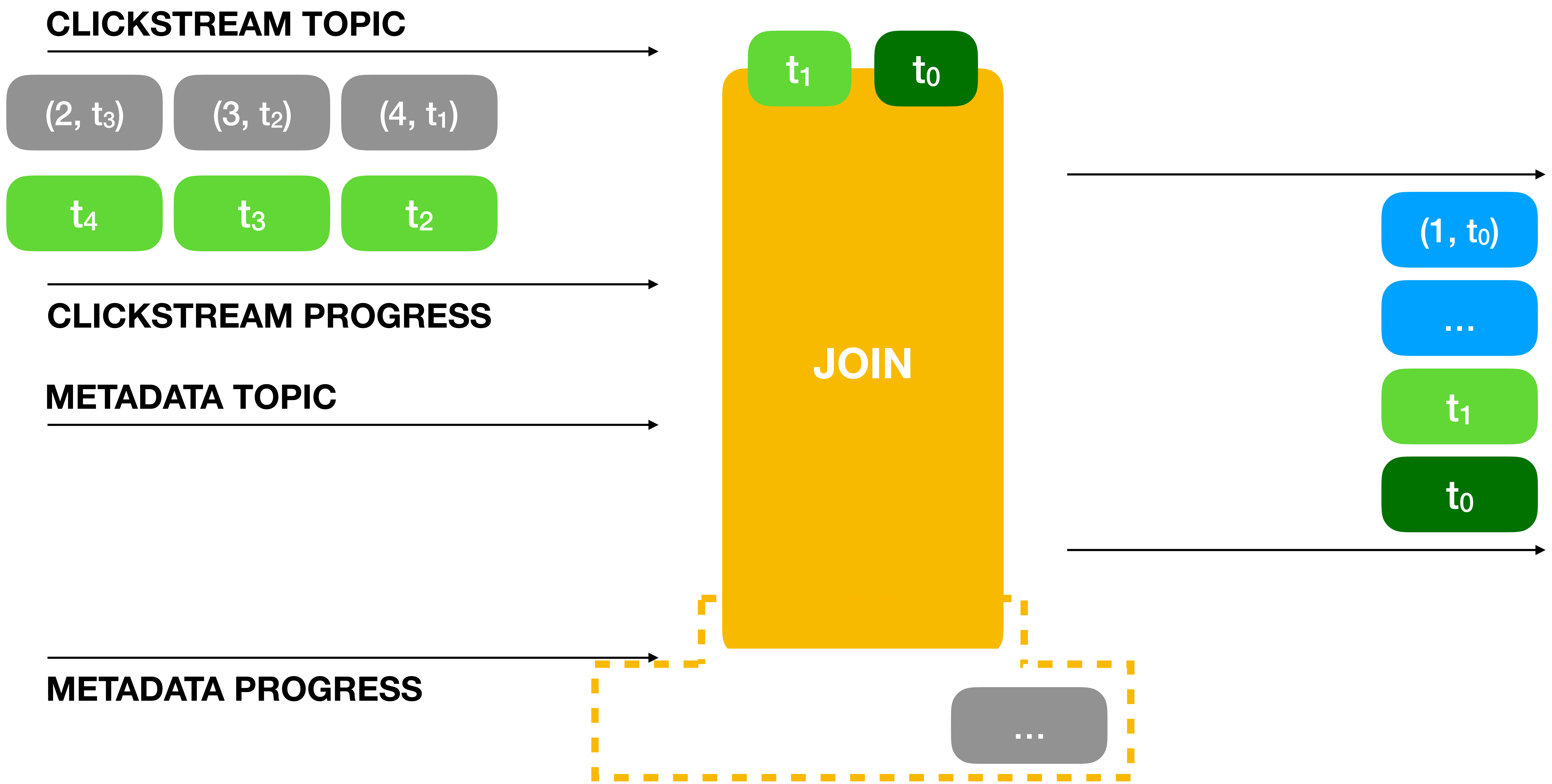
Multidimensional Progress Tracking

(track sources along independent timelines)

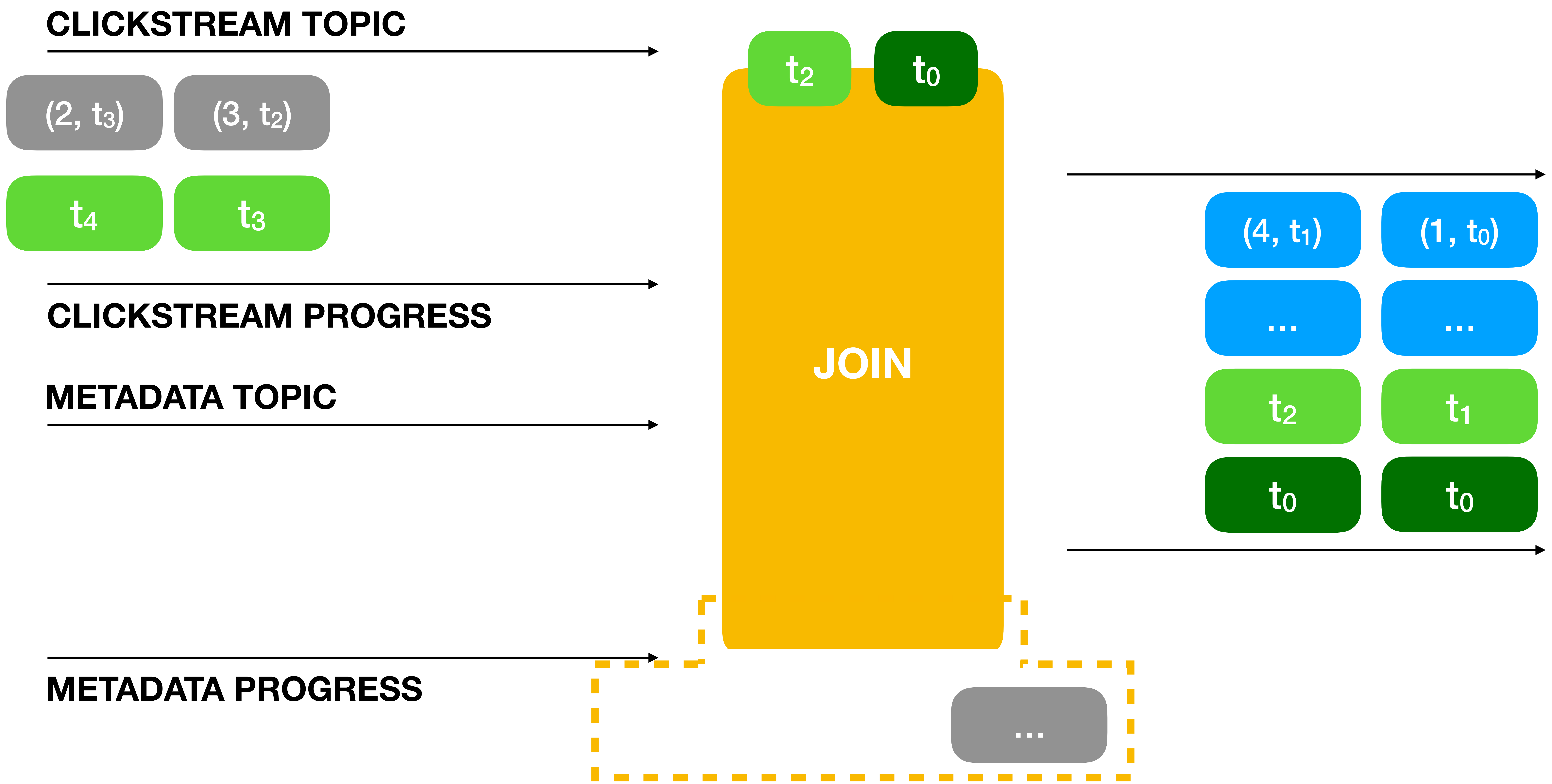


Multidimensional Progress Tracking

(track sources along independent timelines)



Multidimensional Progress Tracking
(track sources along independent timelines)



Creating 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();
}
```

Creating Dataflows with Timely

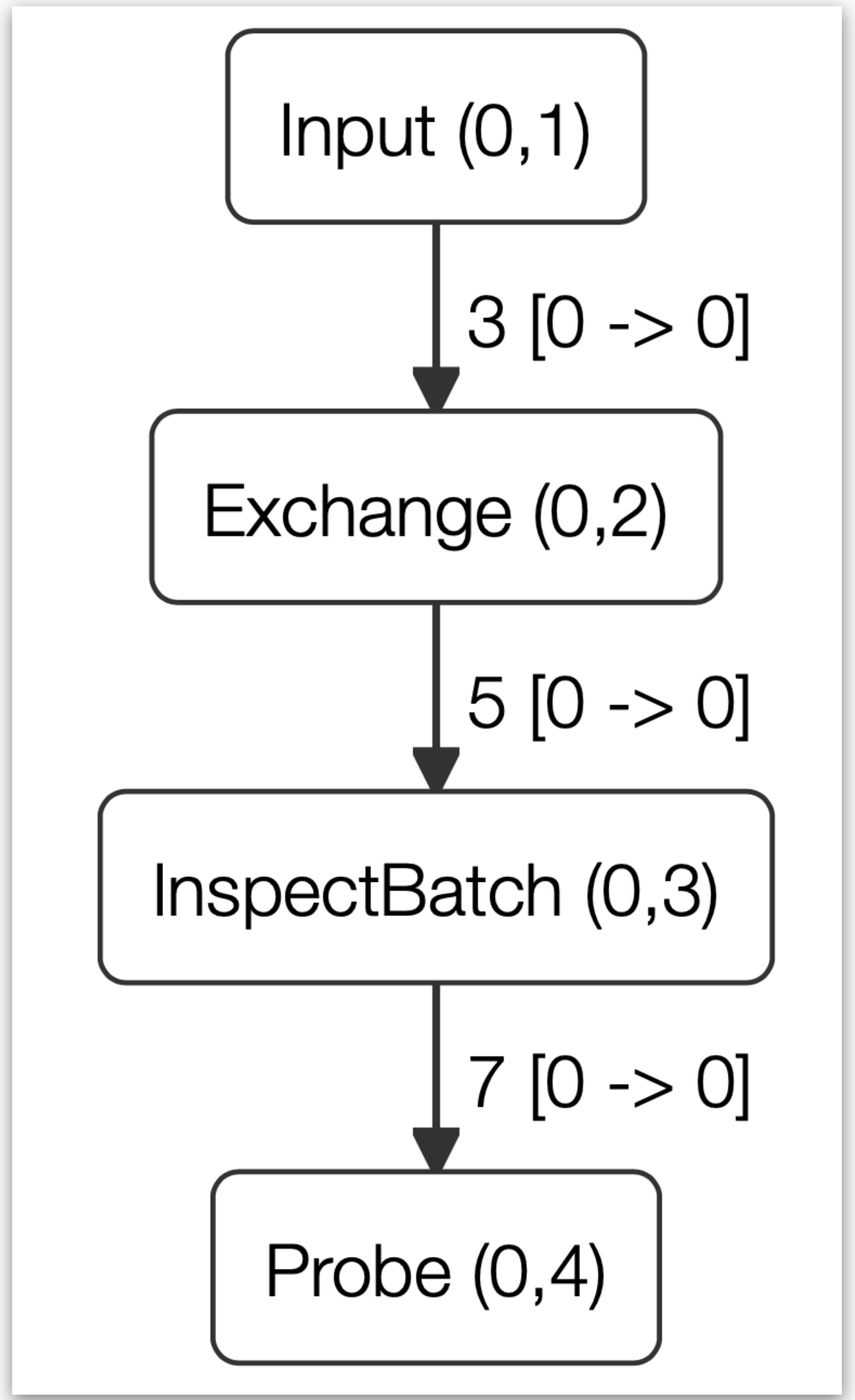
```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {


        // 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();
}
```

Creating 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();
}
```

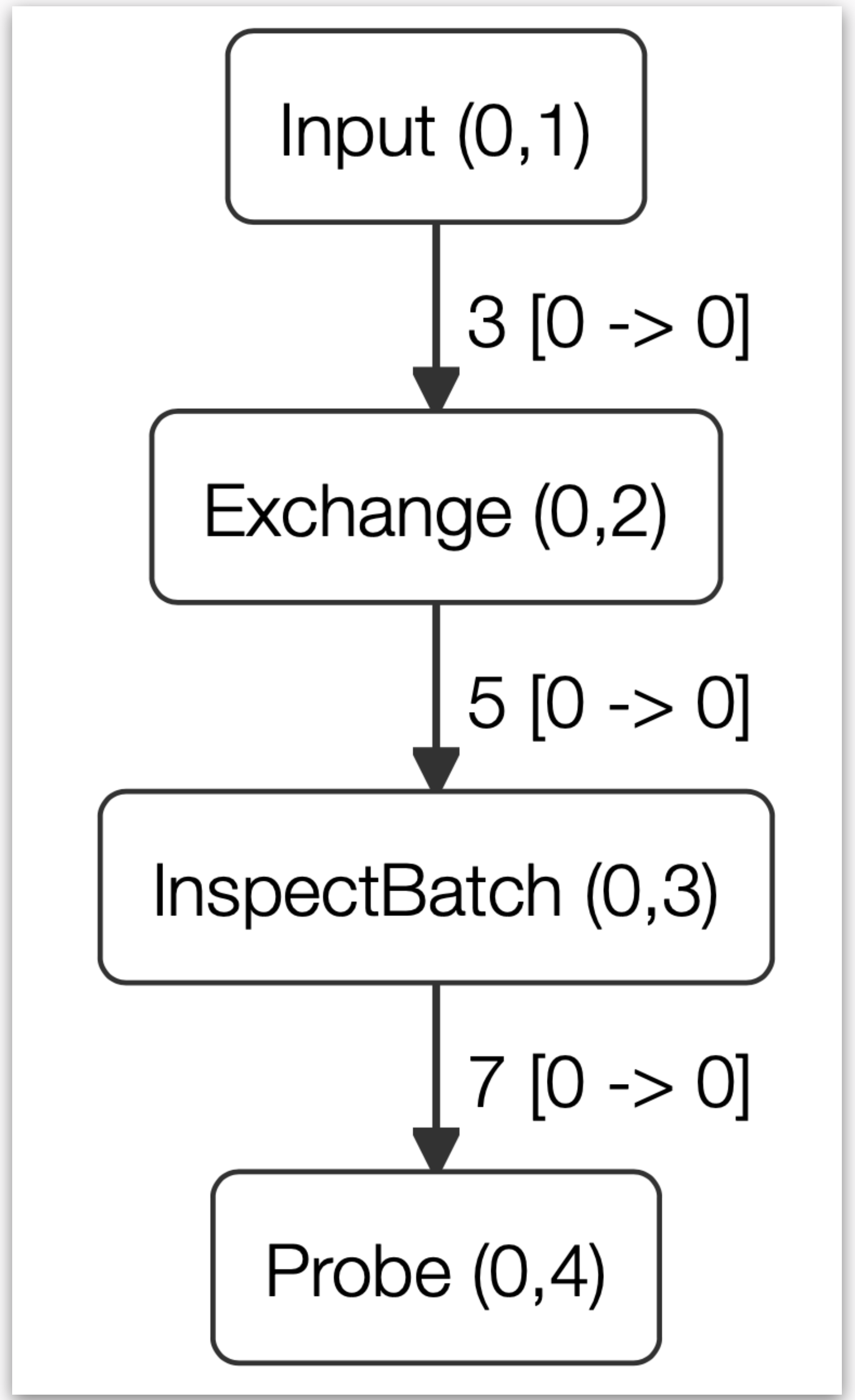
Creating Dataflows with Timely

```
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();
}
```


$$\begin{tikzcd}
\boxed{\text{Input (0,1)}} \arrow[d, "{3\ [0 \to 0]}"] \\
\boxed{\text{Exchange (0,2)}} \arrow[d, "{5\ [0 \to 0]}"] \\
\boxed{\text{InspectBatch (0,3)}} \arrow[d, "{7\ [0 \to 0]}"] \\
\boxed{\text{Probe (0,4)}}
\end{tikzcd}$$

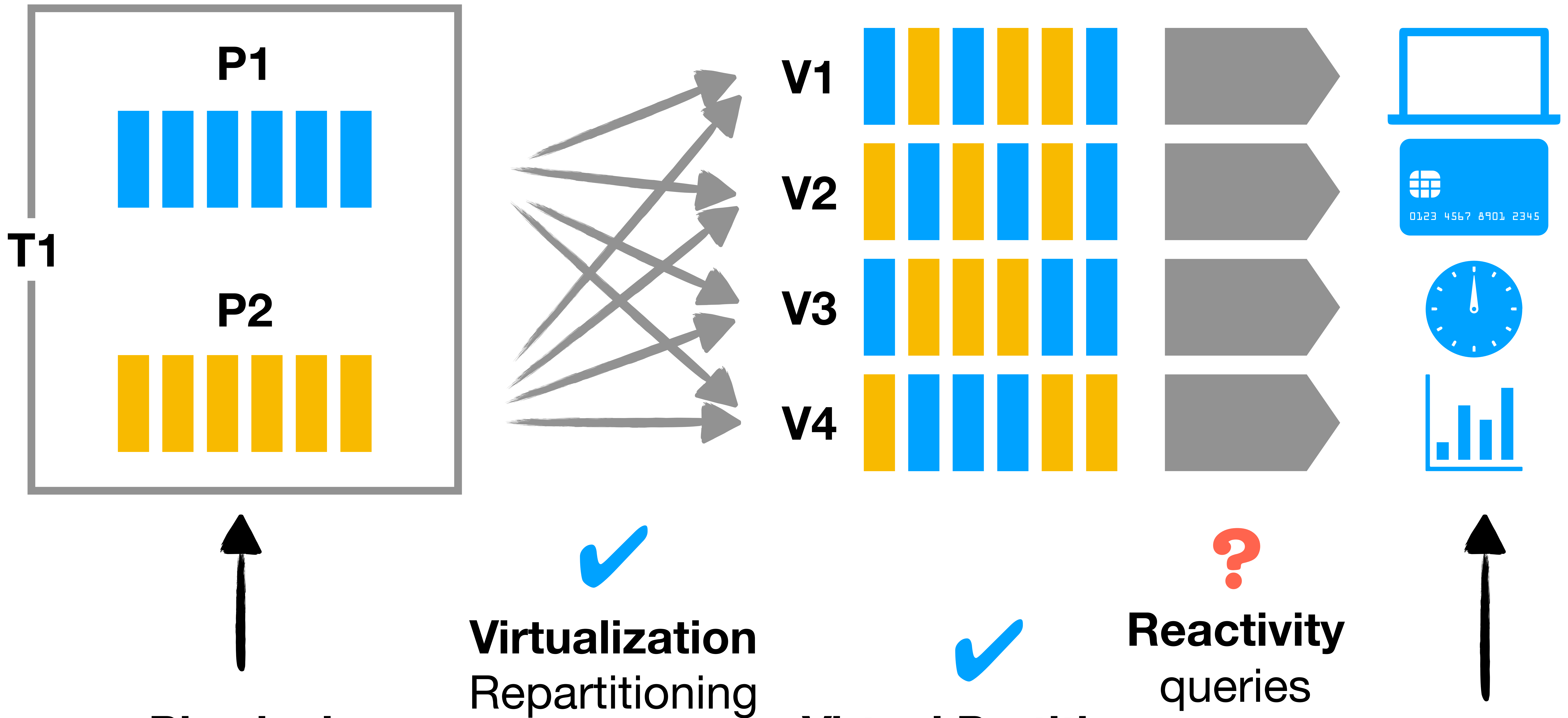
Running Dataflows with Timely

```
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();
}
```


$$\begin{tikzcd}[row sep=large] \text{Input (0,1)} \arrow[d, "3\ [0 \to 0]"] \\ \text{Exchange (0,2)} \arrow[d, "5\ [0 \to 0]"] \\ \text{InspectBatch (0,3)} \arrow[d, "7\ [0 \to 0]"] \\ \text{Probe (0,4)} \end{tikzcd}$$

ArrangedSource (0,3,56)
149 [0 -> 0]
120 [0 -> 1]
ArrangedSource (0,3,58)
Map (0,3,71)
ArrangedSource (0,3,72)
124 [0 -> 1]
152 [0 -> 0]
153 [0 -> 1]
LookupMap (0,3,59)
LookupMap (0,3,73)
ArrangedSource (0,3,74)
126 [0 -> 0]
156 [0 -> 0]
157 [0 -> 1]
,49)
Partition (0,3,60)
ArrangedSource (0,3,61)
ArrangedSource (0,3,65)
LookupMap (0,3,75)
> 1]
129 [0 -> 0]
137 [1 -> 0]
130 [0 -> 1]
138 [0 -> 1]
159 [0 -> 0]
dSource (0,3,51)
LookupMap (0,3,62)
ArrangedSource (0,3,63)
LookupMap (0,3,66)
ArrangedSource (0,3,67)
Partition (0,3,76)
ArrangedSource (0,3,77)
ArrangedSource (0,3,81)
109 [0 -> 1]
133 [0 -> 0]
134 [0 -> 1]
141 [0 -> 0]
142 [0 -> 1]
162 [0 -> 0]
170 [1 -> 0]
163 [0 -> 1]
171 [0 -> 1]
LookupMap (0,3,64)
LookupMap (0,3,68)
LookupMap (0,3,78)
ArrangedSource (0,3,79)
LookupMap (0,3,82)
ArrangedSource (0,3,83)
144 [0 -> 0]
145 [0 -> 1]
166 [0 -> 0]
167 [0 -> 1]
174 [0 -> 0]
175 [0 -> 1]
Concatenate (0,3,69)
LookupMap (0,3,80)
LookupMap (0,3,84)
147 [0 -> 0]
177 [0 -> 0]
178 [0 -> 1]
Map (0,3,70)
Concatenate (0,3,85)
183 [0 -> 1]
180 [0 -> 0]
Concatenate (0,3,87)
Map (0,3,86)
185 [0 -> 0]
186 [0 -> 1]
Concatenate (0,3,88)
187,189 [0 -> 0]
Map (0,4)
193 [0 -> 0]
Probe (0,5)

Kafka Superpowers



Physical Representation

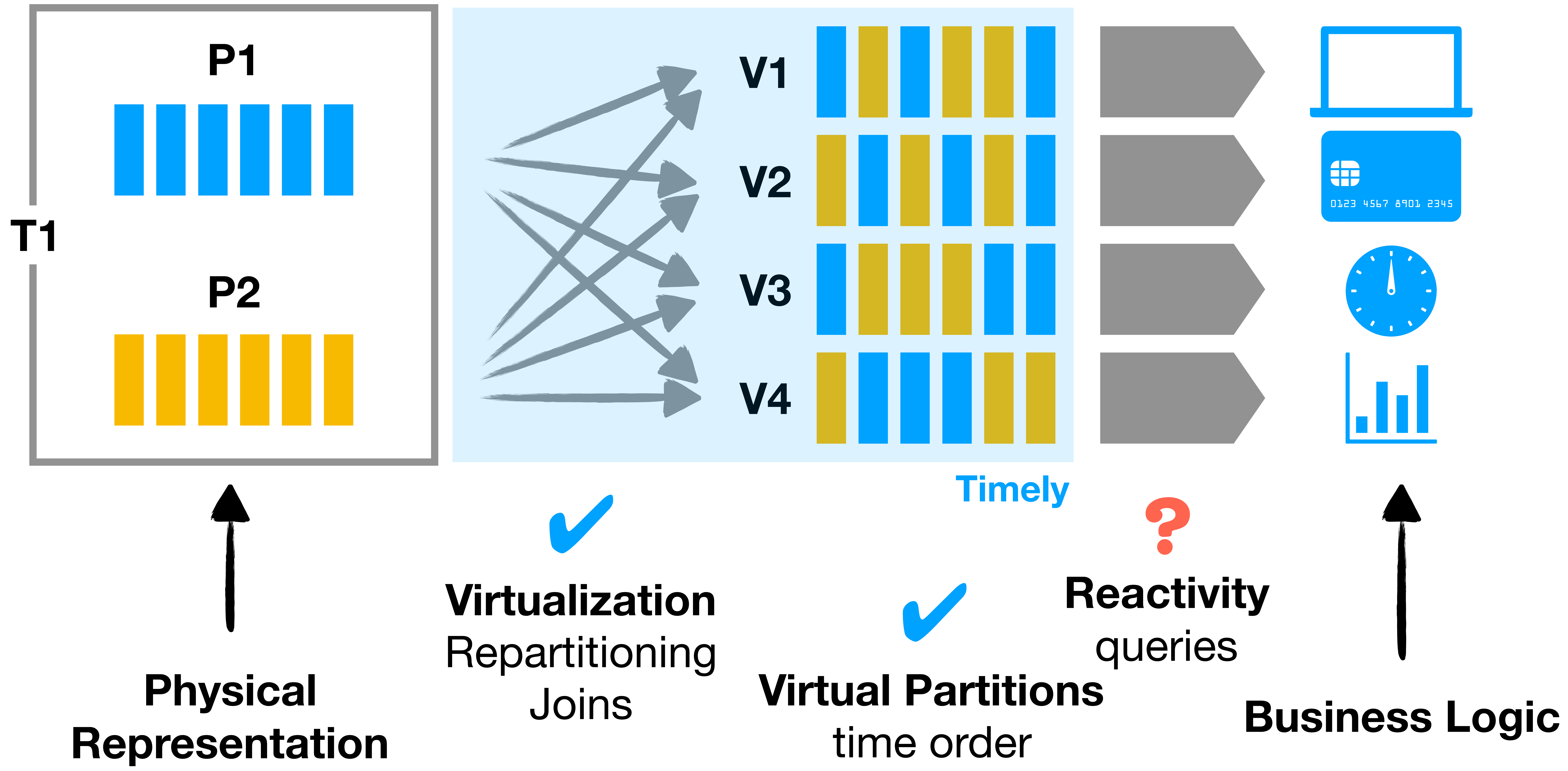
Virtualization
Repartitioning
Joins

Virtual Partitions
time order

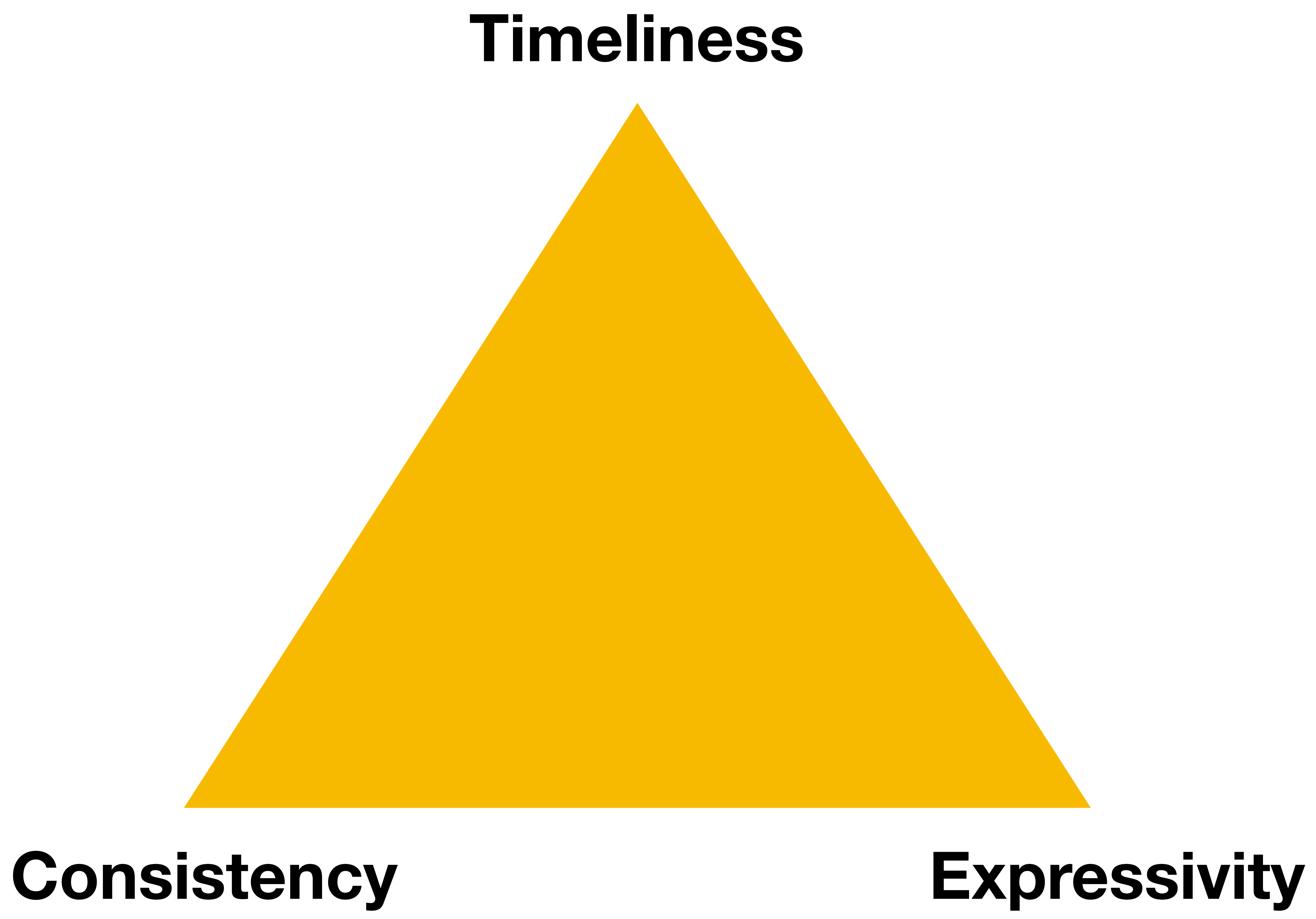
Reactivity
queries

Business Logic

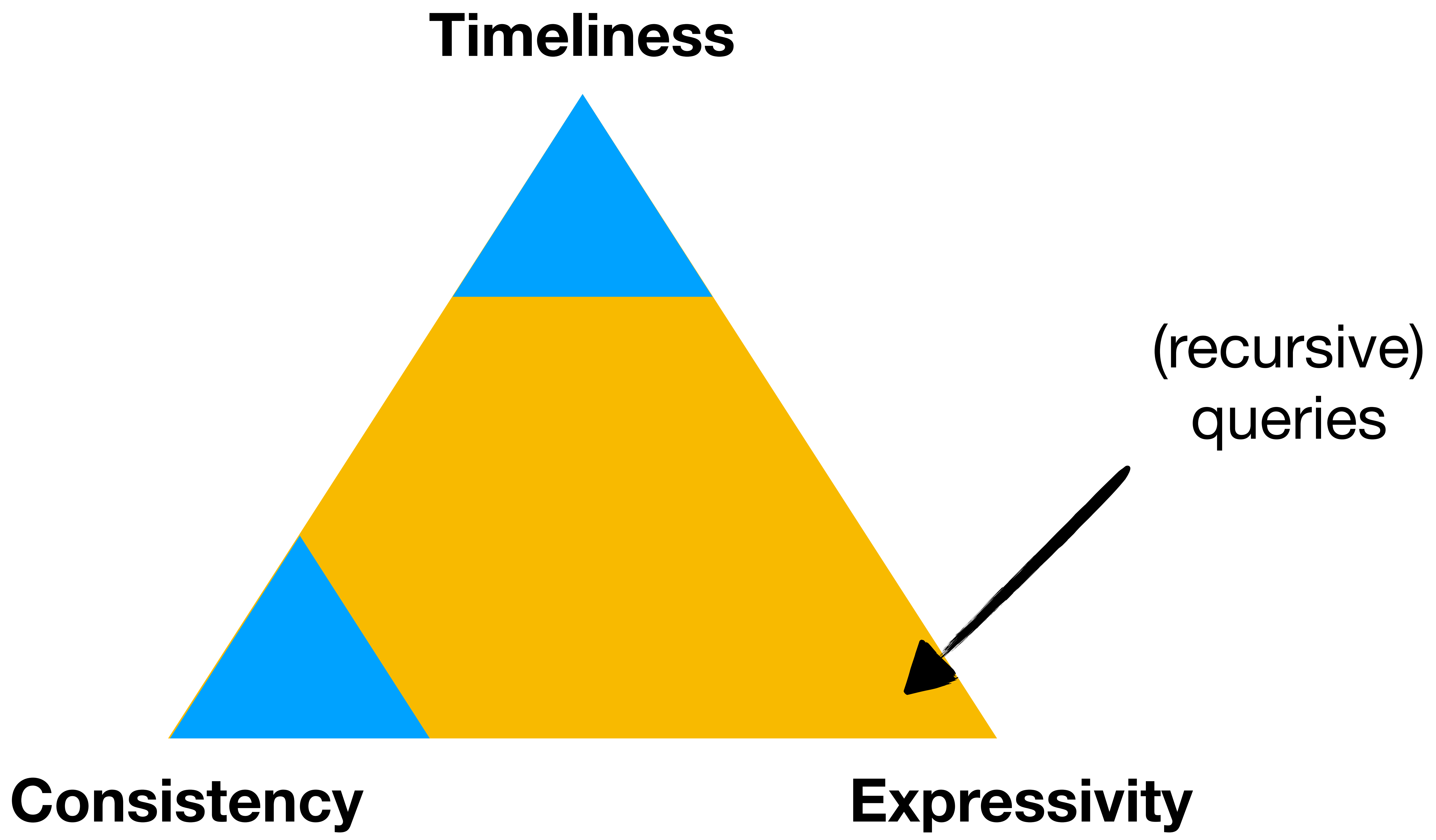
Kafka Superpowers



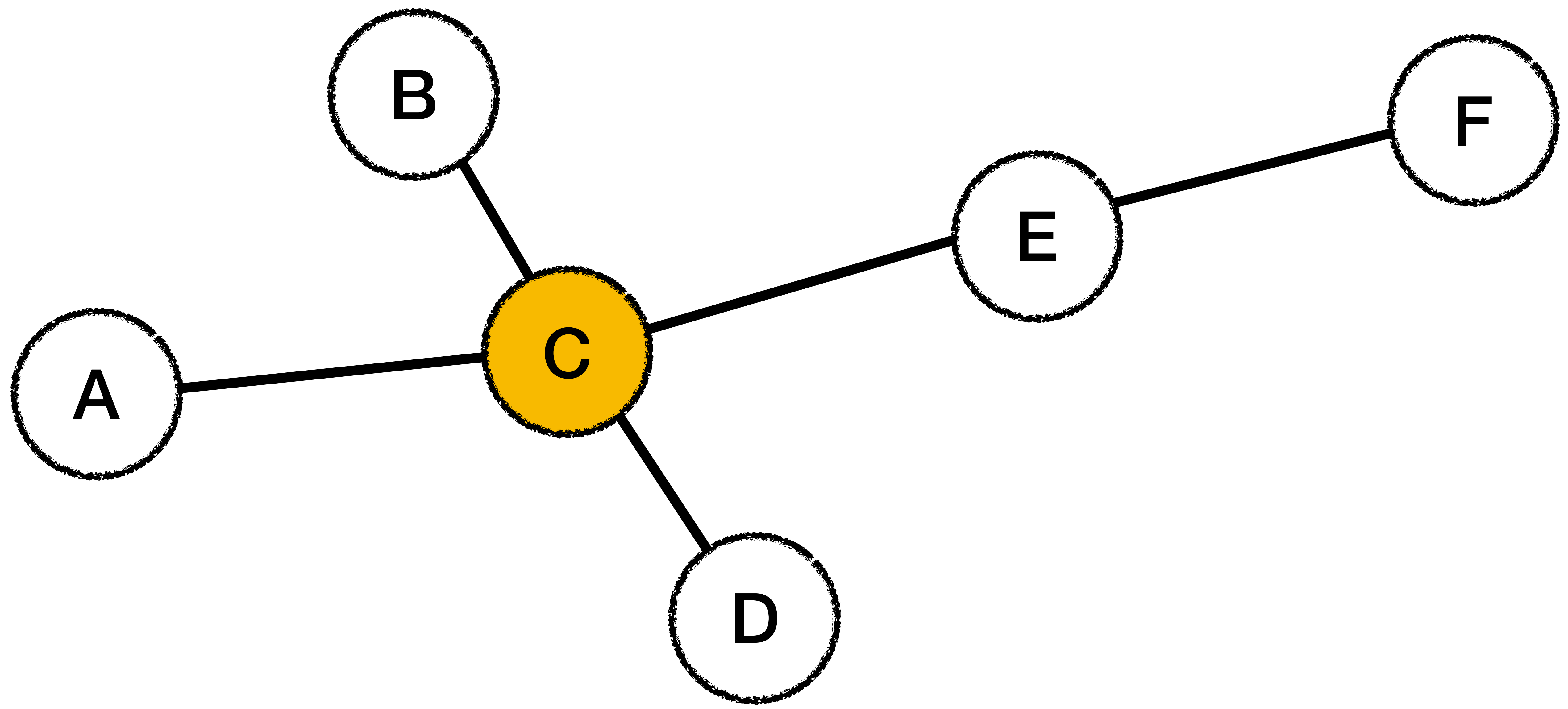
The Trifecta?



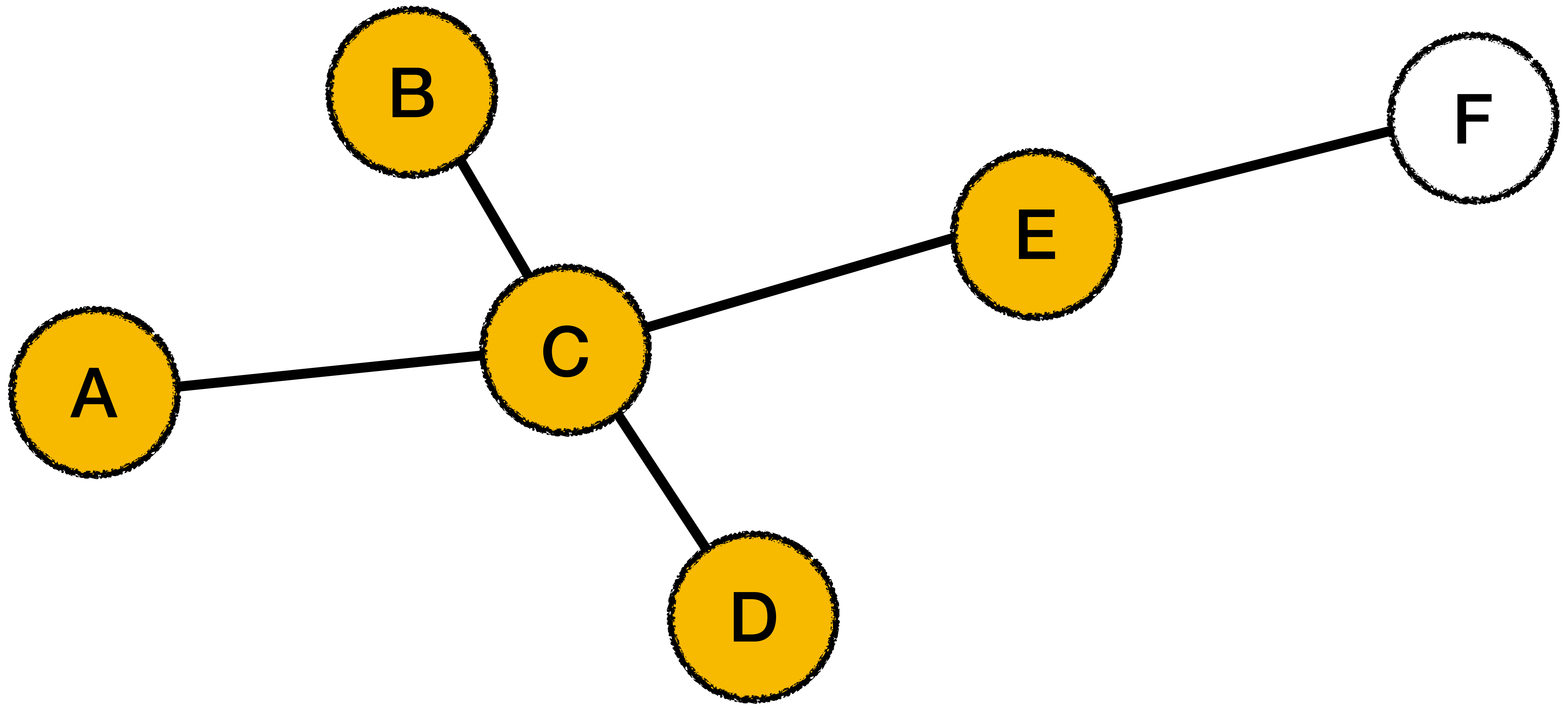
The Trifecta?



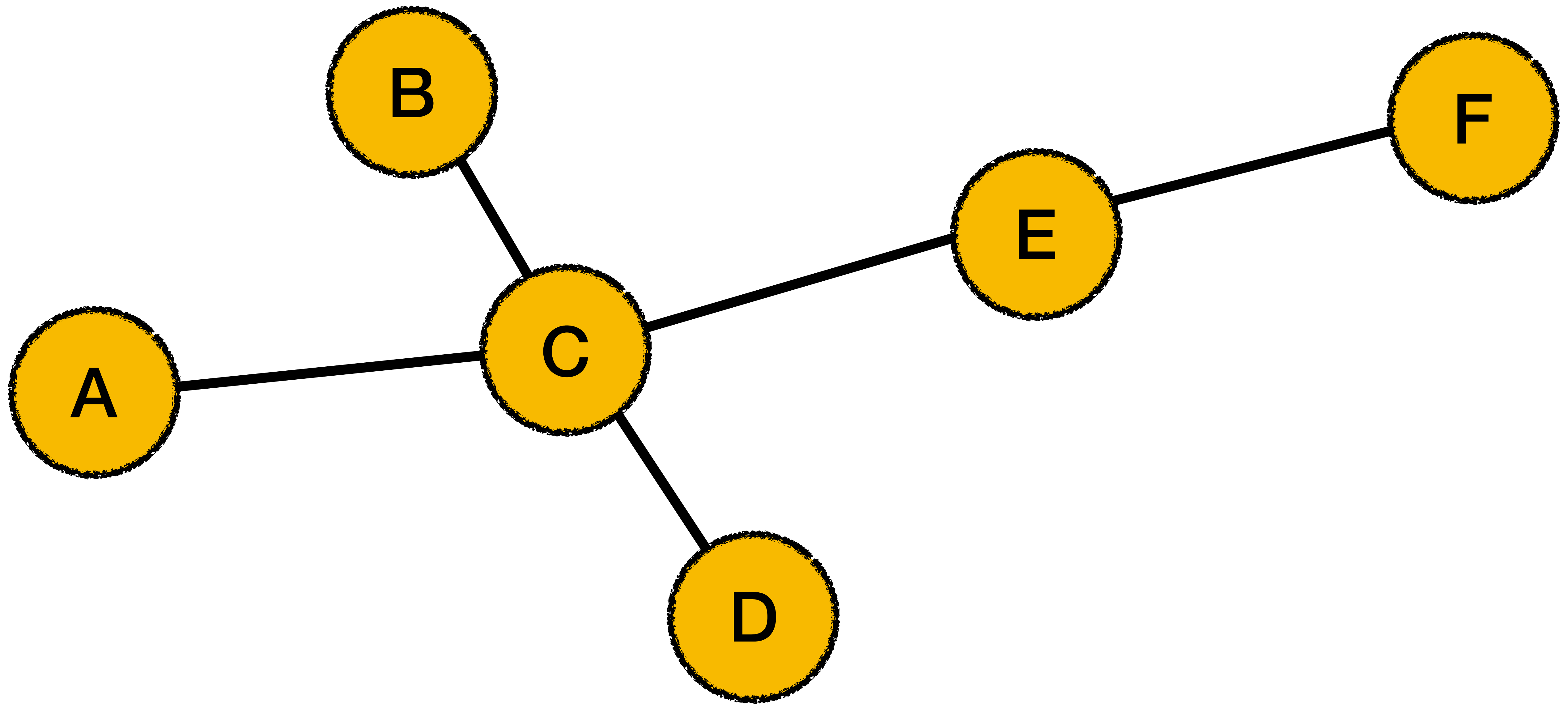
Recursive Graph Traversal



Recursive Graph Traversal

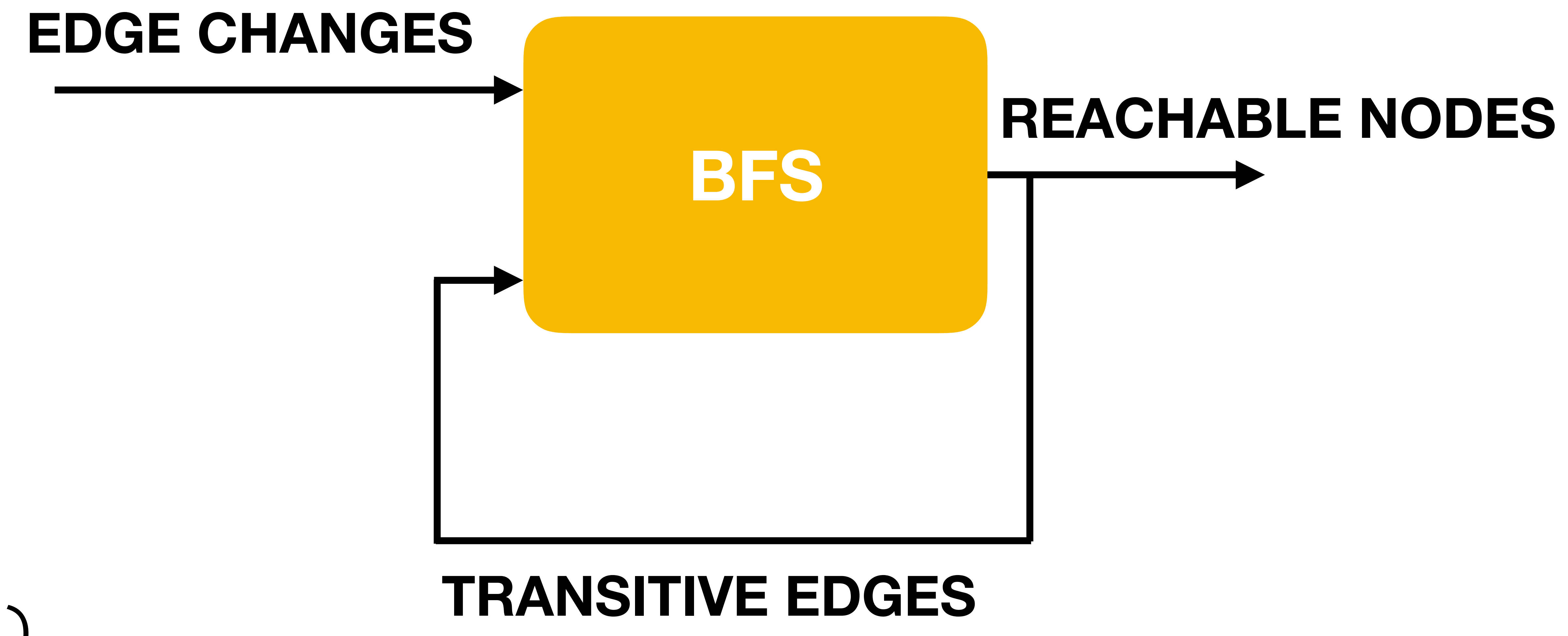


Recursive Graph Traversal



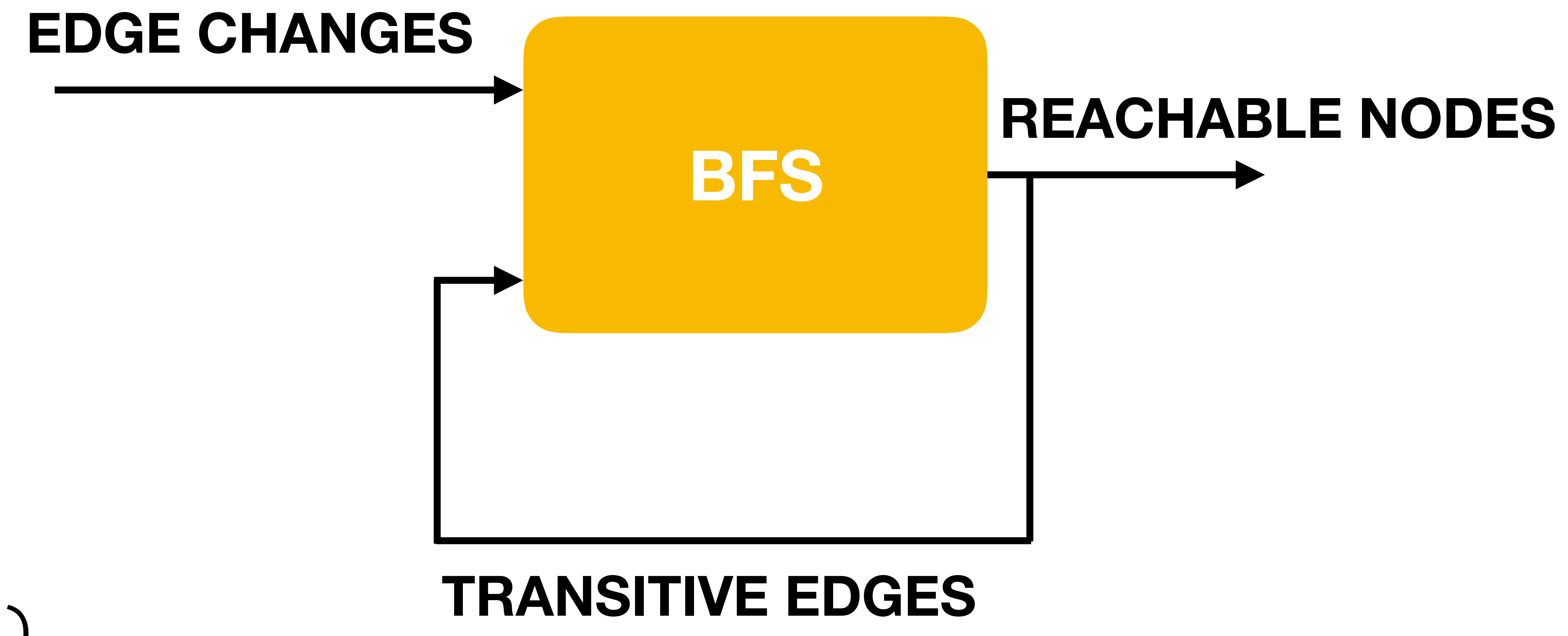
Recursive Dataflows

```
/// Breadth-First Search

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)))
})
```



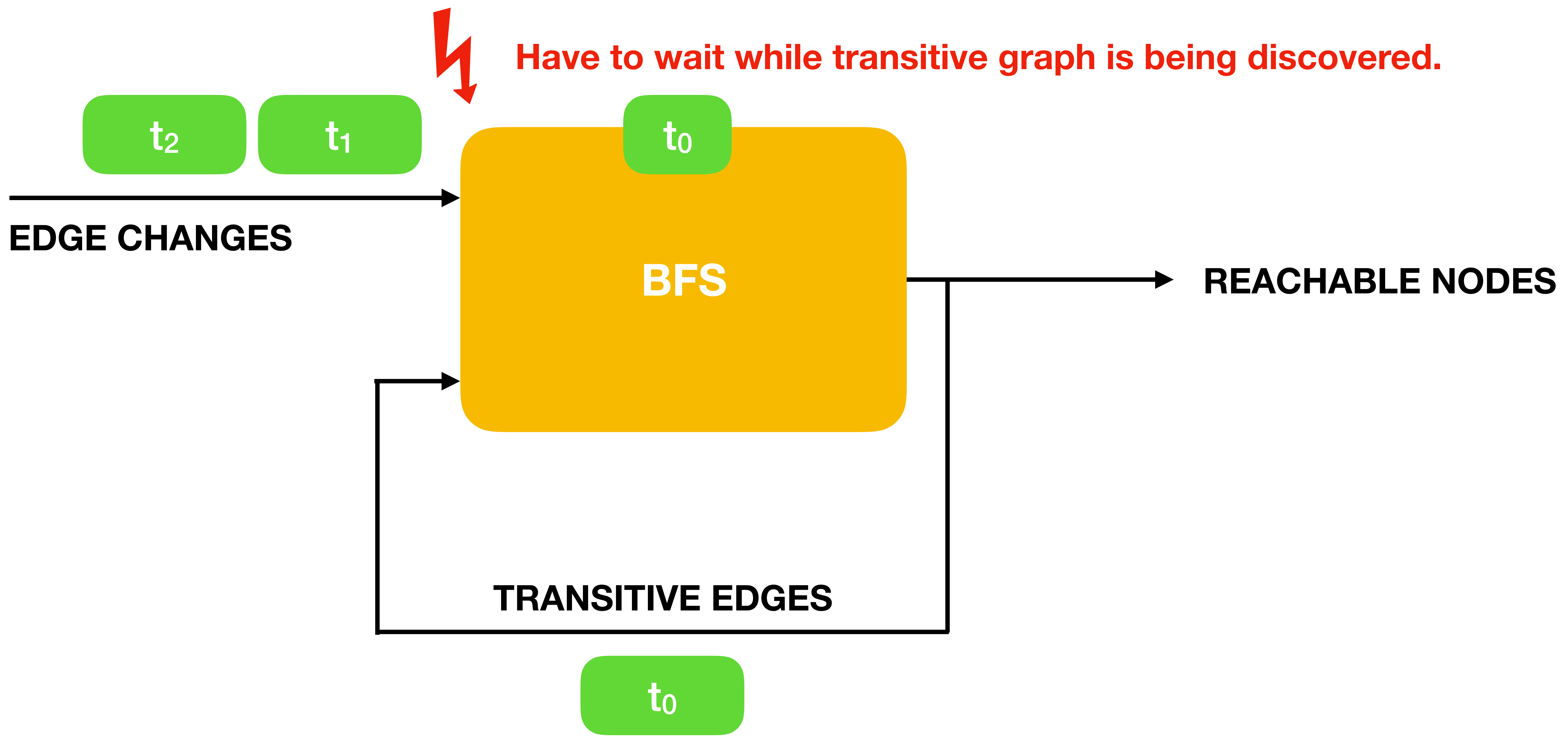
Recursive Dataflows

```
/// Breadth-First Search

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)))
})
```



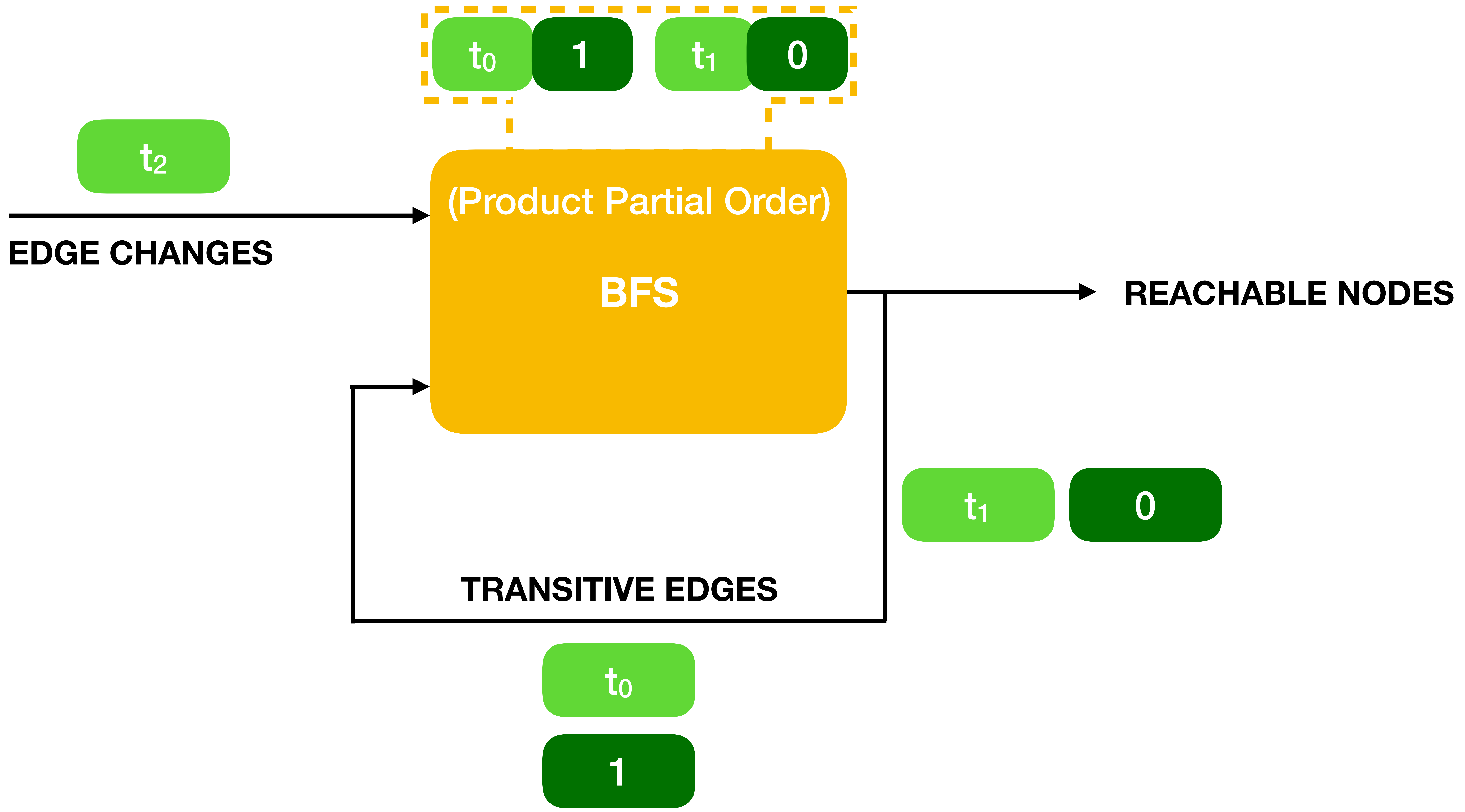
Progress Tracking… with Loops?

(have to finish iterating before we can handle next input)



Multidimensional Progress Tracking

(track iteration depth separately)



Lexicographical Order (Join)

(visibility for t_2 t_2)

	t_0	t_1	t_2	t_3
t_0	✓	✓	✓	✓
t_1	✓	✓	✓	✓
t_2	✓	✓	✓	✗
t_3	✗	✗	✗	✗

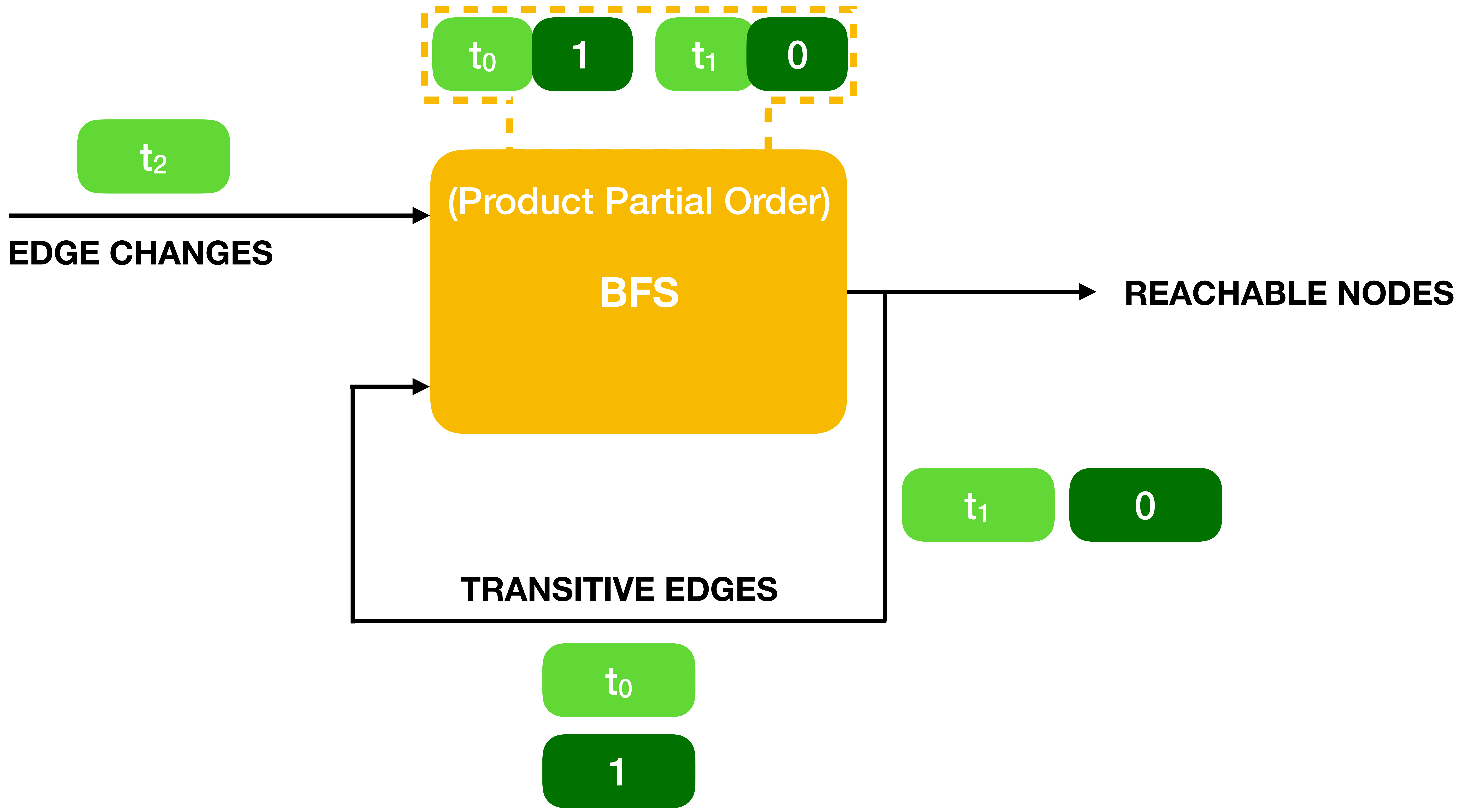
Product Partial Order (Iteration)

(visibility for t_2 2)

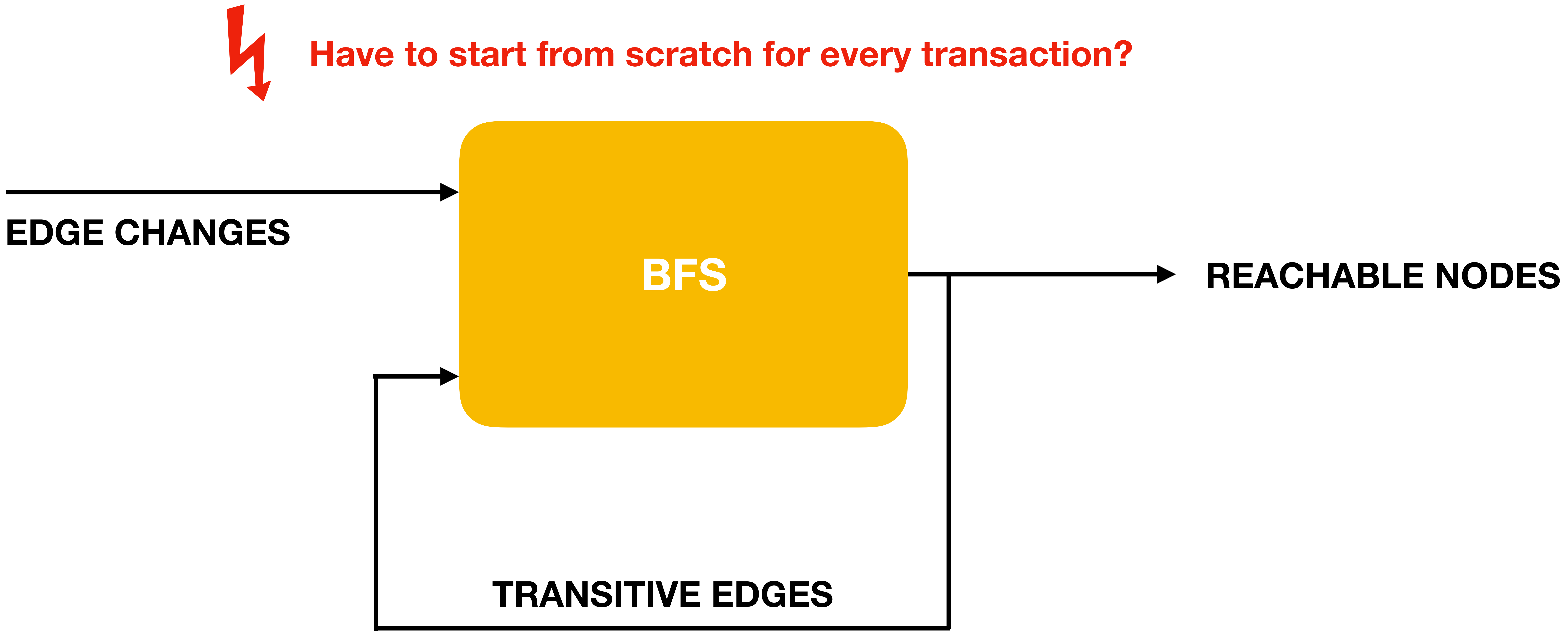
	0	1	2	3
t_0	✓	✓	✓	✗
t_1	✓	✓	✓	✗
t_2	✓	✓	✓	✗
t_3	✗	✗	✗	✗

Multidimensional Progress Tracking

(track iteration depth separately)



Incremental Execution?

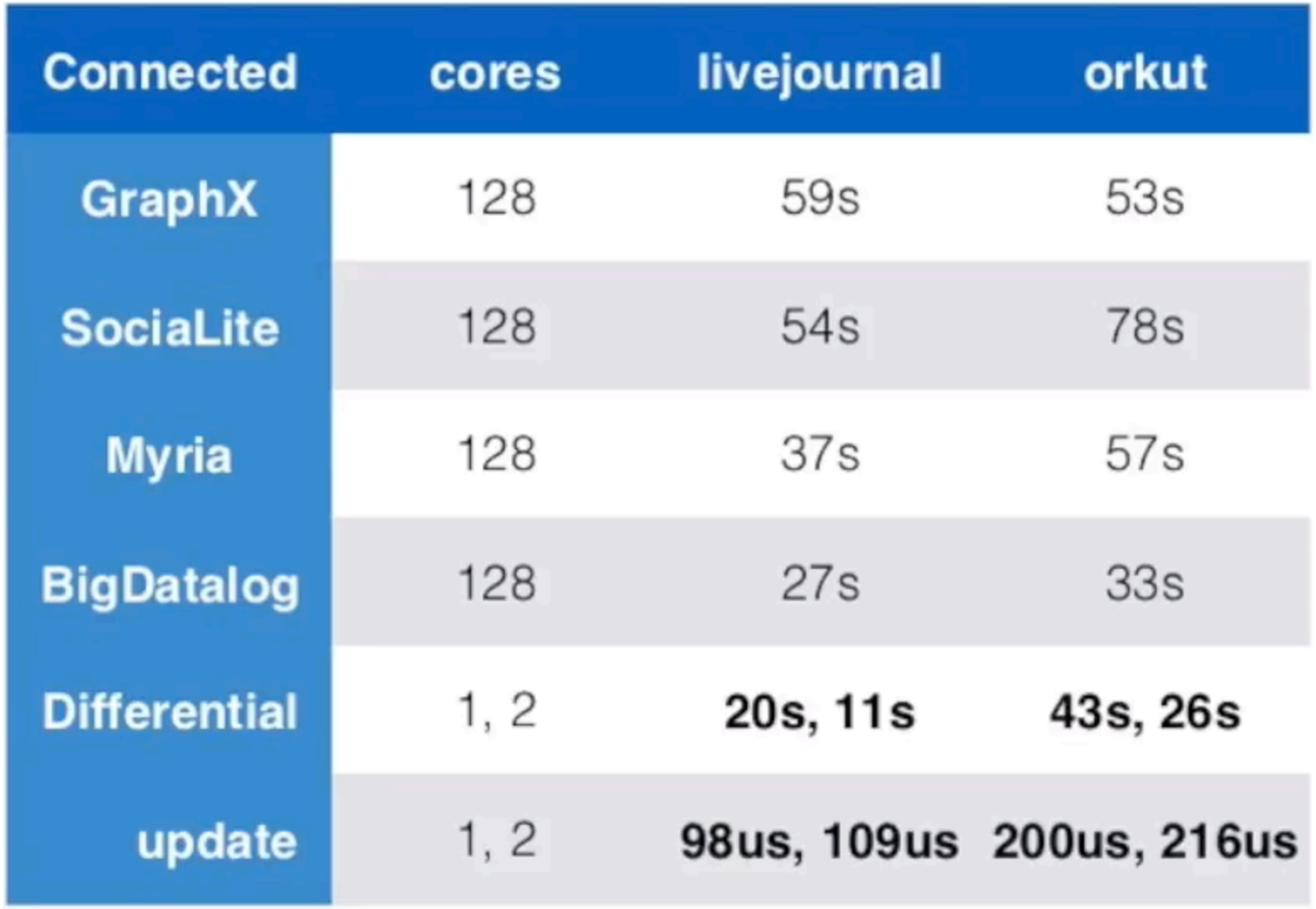


Differential Dataflow

Iterative, incrementalized operators for Timely

github.com/**TimelyDataflow**

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**

Streaming & Relational Queries

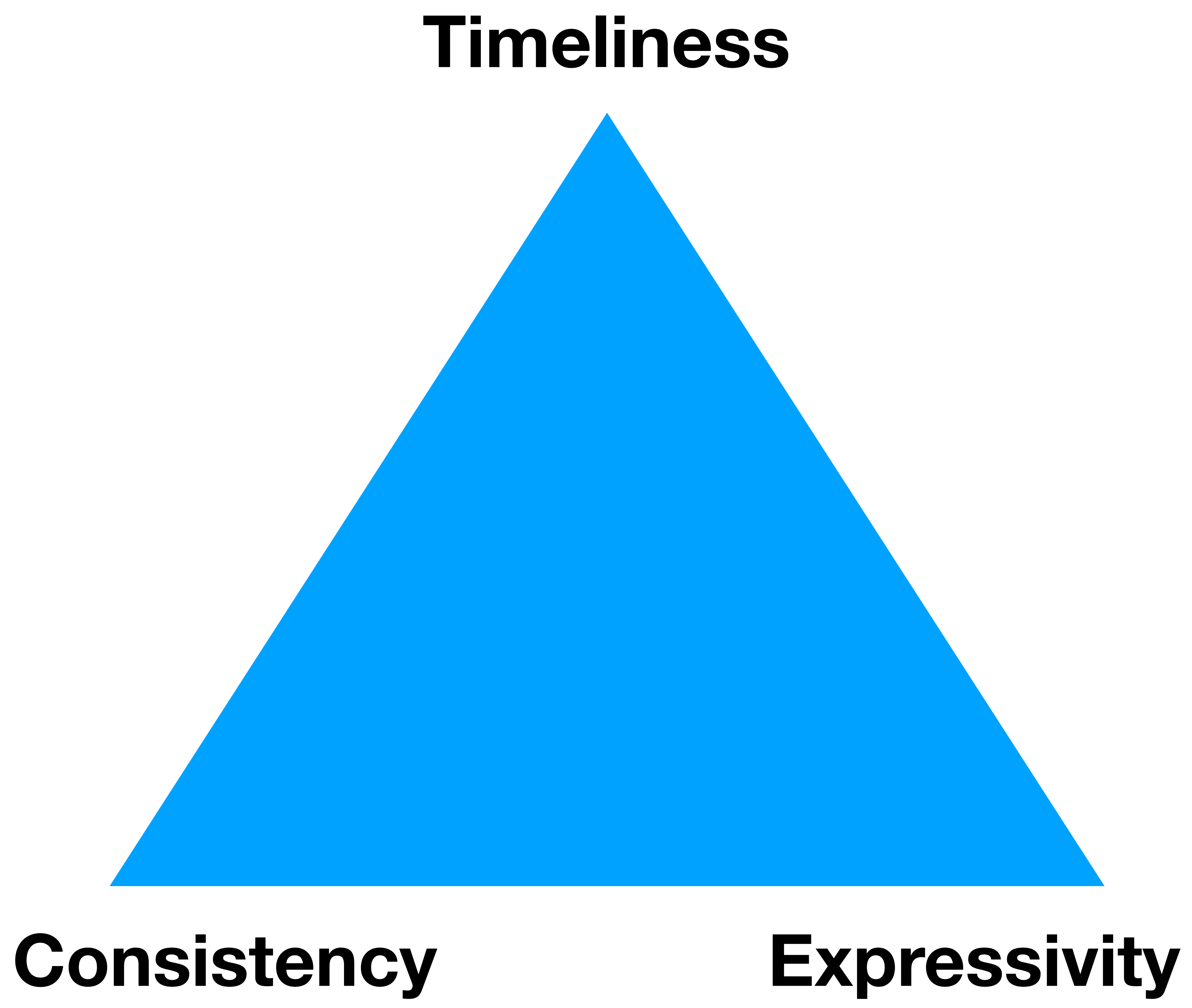
Declarative Differential Dataflows (3DF)

```
/// 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)
     .reduce(|_, s, t| t.push((*s[0].0, 1)))
})
```

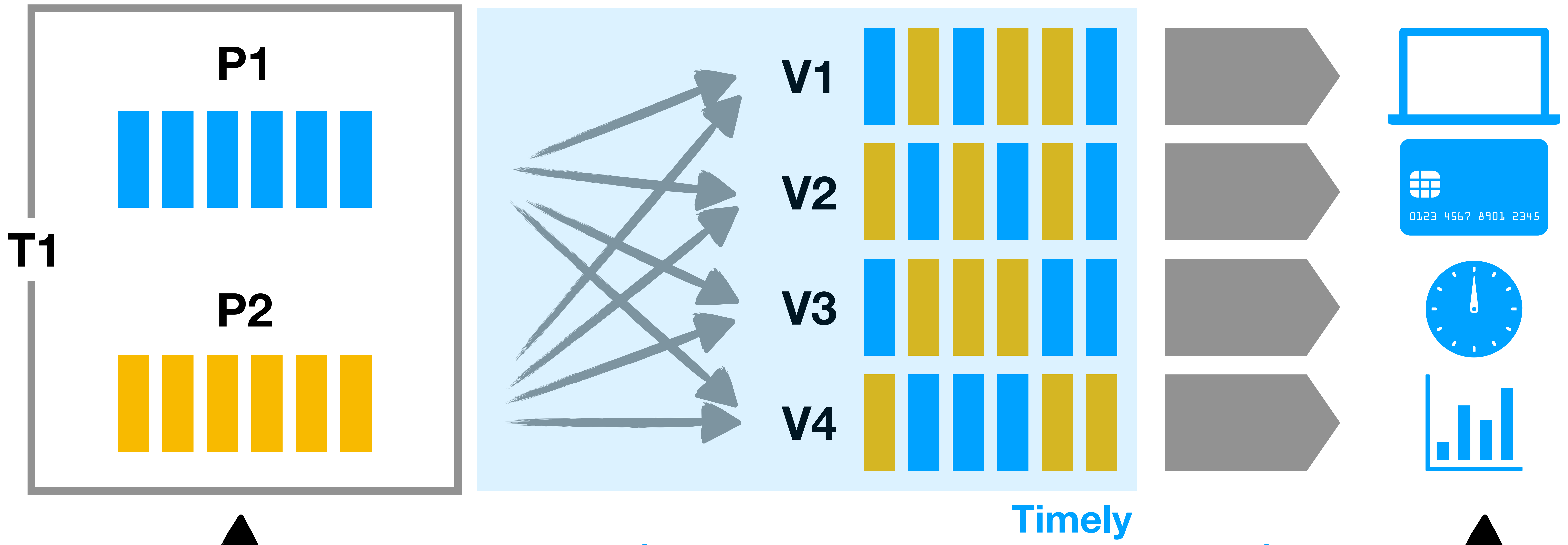
```
[[(bfs ?from ?to)
  [?from :edge ?to]]

 [(bfs ?from ?to)
  [?from :edge ?hop]
  (bfs ?hop ?to)]]
```

github.com/comnik/**declarative-dataflow**

The Trifecta!



Kafka Superpowers



✓ **Virtualization**
Repartitioning
Joins

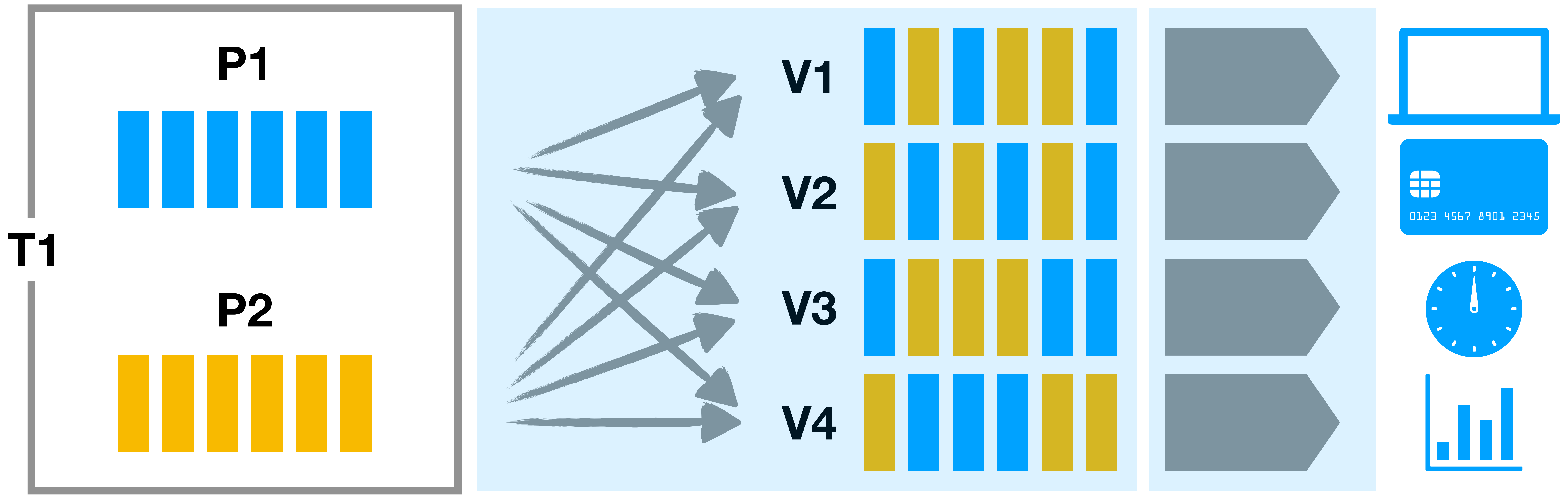
✓ **Virtual Partitions**
time order

✓ **Reactivity**
queries

Physical Representation

Business Logic

Kafka Superpowers



Timely

DD+3DF

Physical Representation

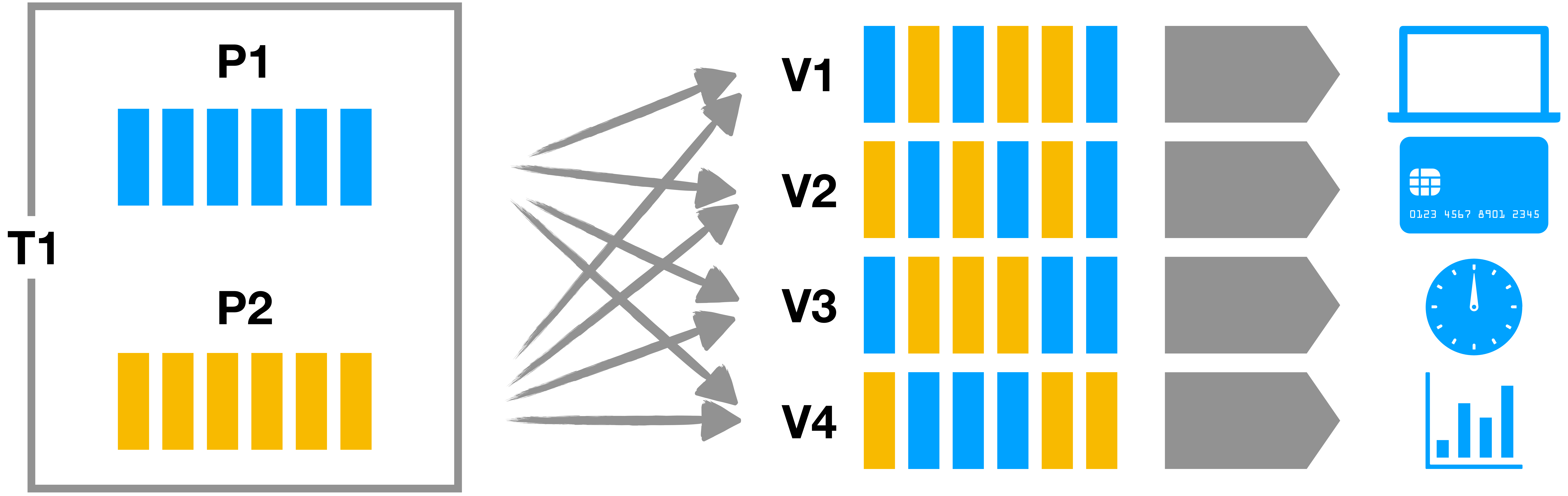
✓ **Virtualization**
Repartitioning
Joins

✓ **Virtual Partitions**
time order

✓ **Reactivity**
queries

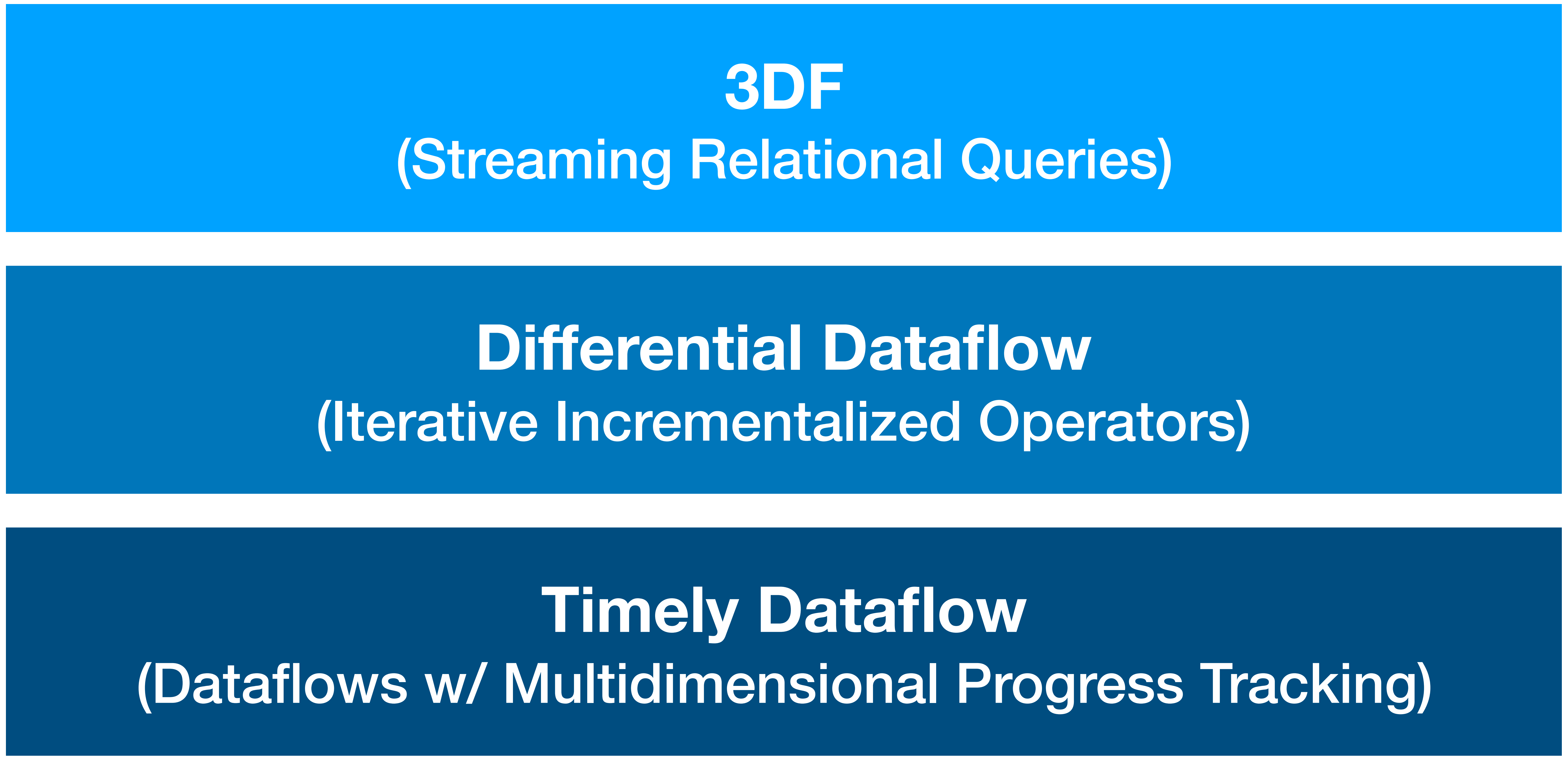
Business Logic

Kafka Superpowers



clockworks.io/kplex

Timely as a Programming Model


3DF
(Streaming Relational Queries)
Differential Dataflow
(Iterative Incrementalized Operators)
Timely Dataflow
(Dataflows w/ Multidimensional Progress Tracking)

github.com/TimelyDataflow

github.com/comnik/declarative-dataflow

Sources

clockworks

www.clockworks.io

{david, malte, moritz, niko}@clockworks.io

Repositories

- Timely: github.com/TimelyDataflow
- ST2: github.com/li1/snailtrail
- 3DF: github.com/comnik/declarative-dataflow
- Differential FAQ: github.com/eoxxs/differential-aggregate-query

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

Talks

- Reactive Datalog for Datomic (clojure/conj 2018): clockworks.io/2018/12/01/conj-talk.html
- Across Time and Space (BobKonf 2019): clockworks.io/2019/03/22/across-time-space.html

Blog Posts

- frankmcsherry.org
- Incremental Functional Aggregate Queries: clockworks.io/2019/07/06/Incremental-Functional-Aggregate-Queries.html
- Dataflows you can't refuse: clockworks.io/2019/02/10/dataflows-you-cant-refuse.html
- Reactive Datalog with Vega: clockworks.io/2018/11/25/reactive-datalog-with-vega.html
- Incremental Datalog with Differential Dataflows: clockworks.io/2018/09/13/incremental-datalaog.html