Across Time and Space

A Many-Worlds Interpretation of State

Göbel Sandstede bobkonf 19

Let's Talk About Time

- Edit History, Undo / Redo
- Multi-user Collaboration
- Heterogeneous Data Sources (sensors!)
- Speculative Transactions ("what-if?")
- Conflict Resolution

Let's Talk About Time

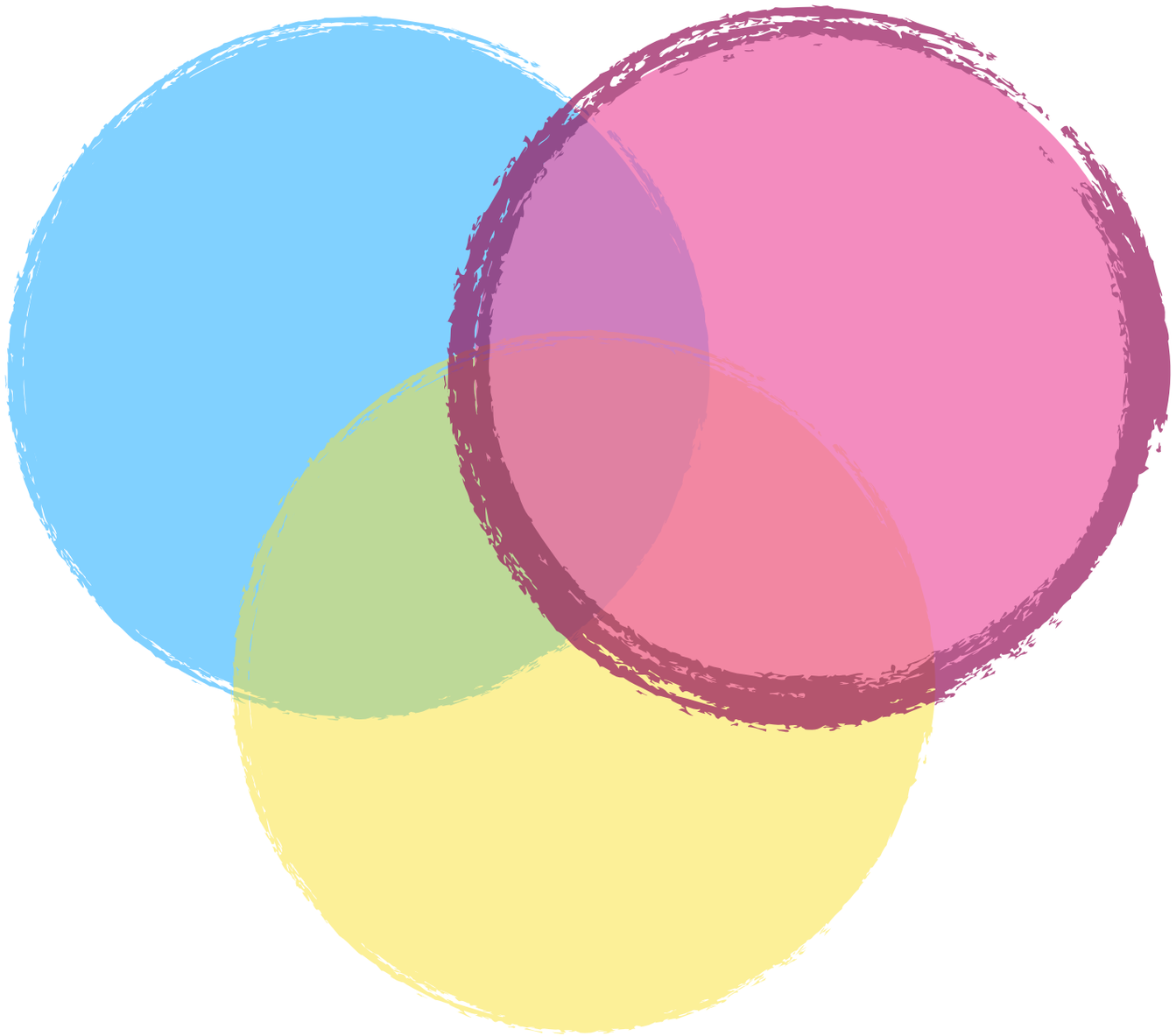
How can we…

(1) …model different time semantics?

(2) …implement those efficiently?

Disclaimer

Talk Focus



Nikolas Göbel
niko@clockworks.io

Malte Sandstede
malte@clockworks.io

stealing ideas from

Frank McSherry (ETH)

ETH zürich

Systems Group

clockworks

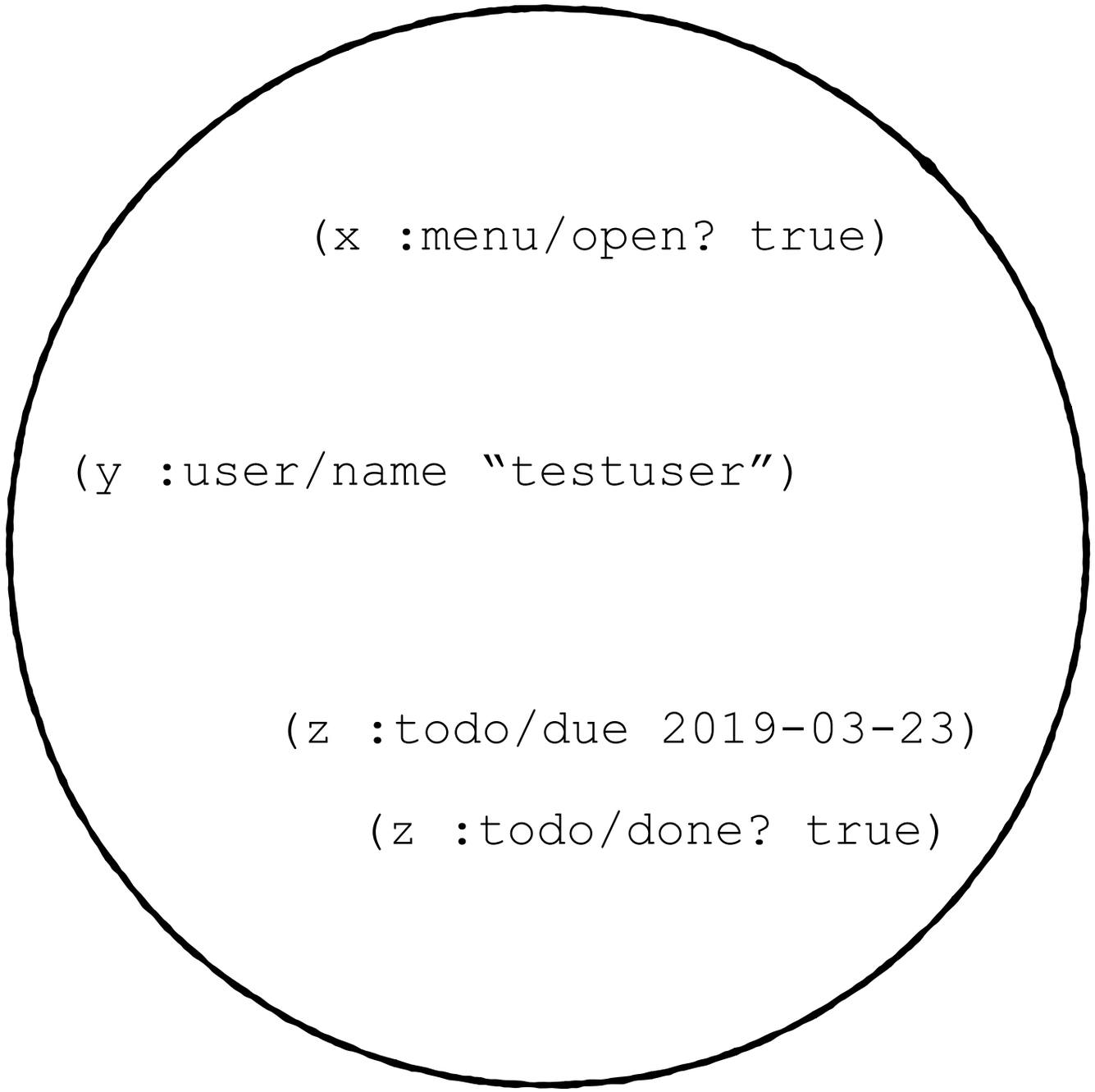
Definitions

Fact := (e a v)

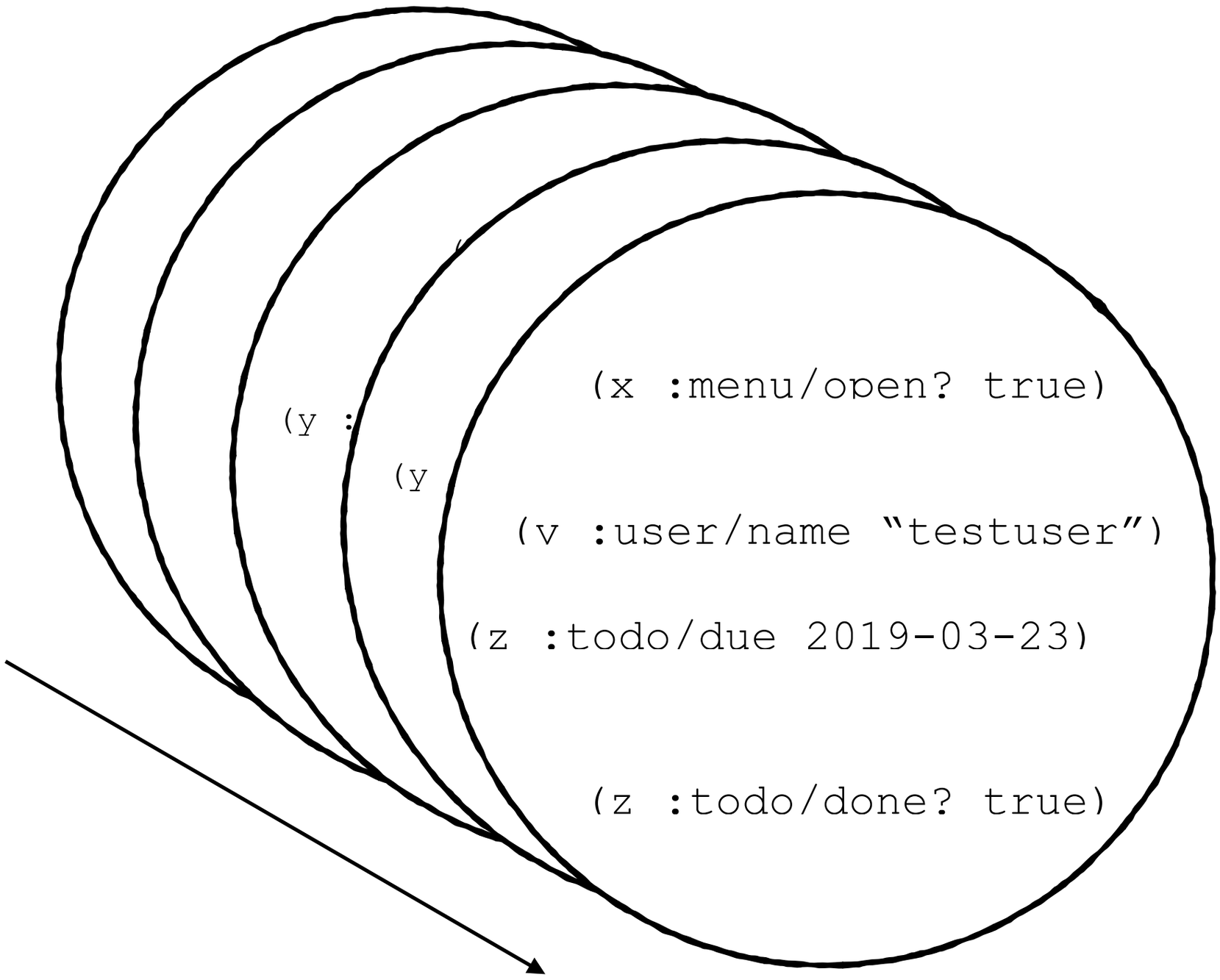
Entity *Attribute* *Value*

Person X	:age	23
TODO Y	:done?	True
Account Z	:owner	Person X

State := Set of Facts

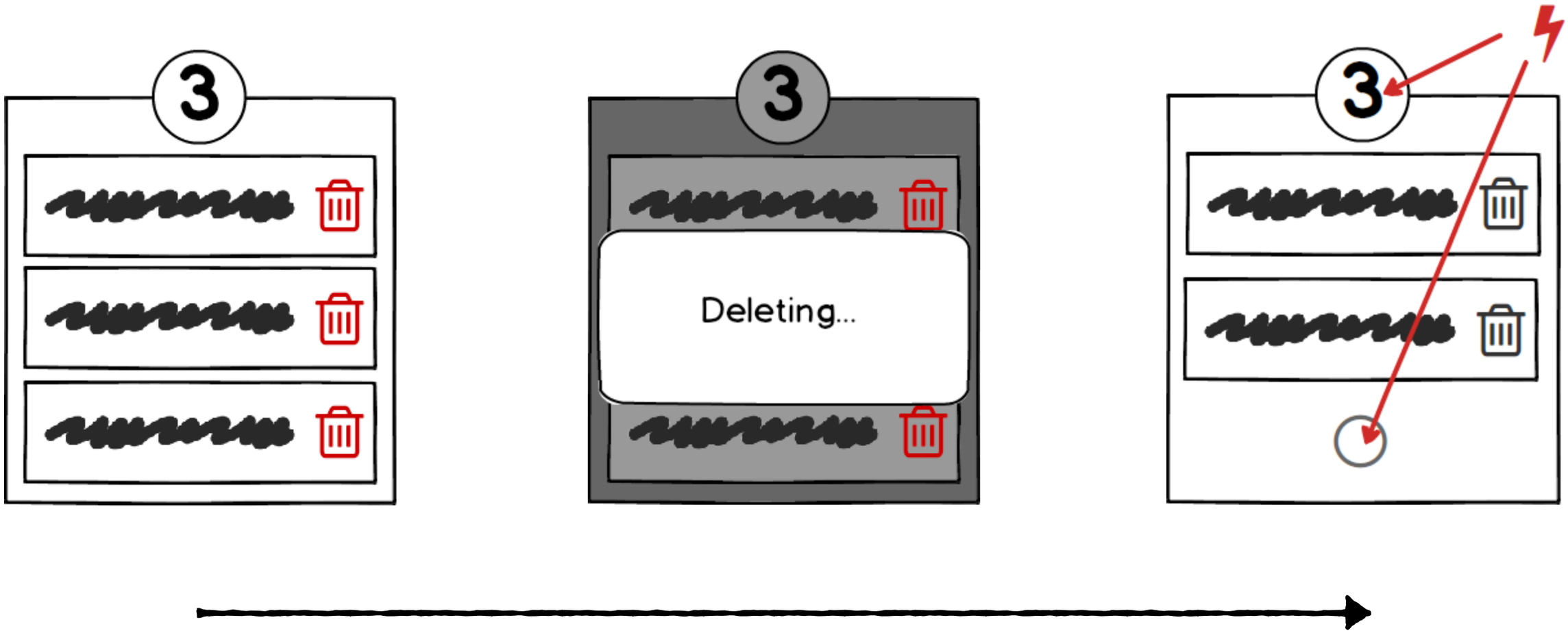


Time := An Order on States



(I)

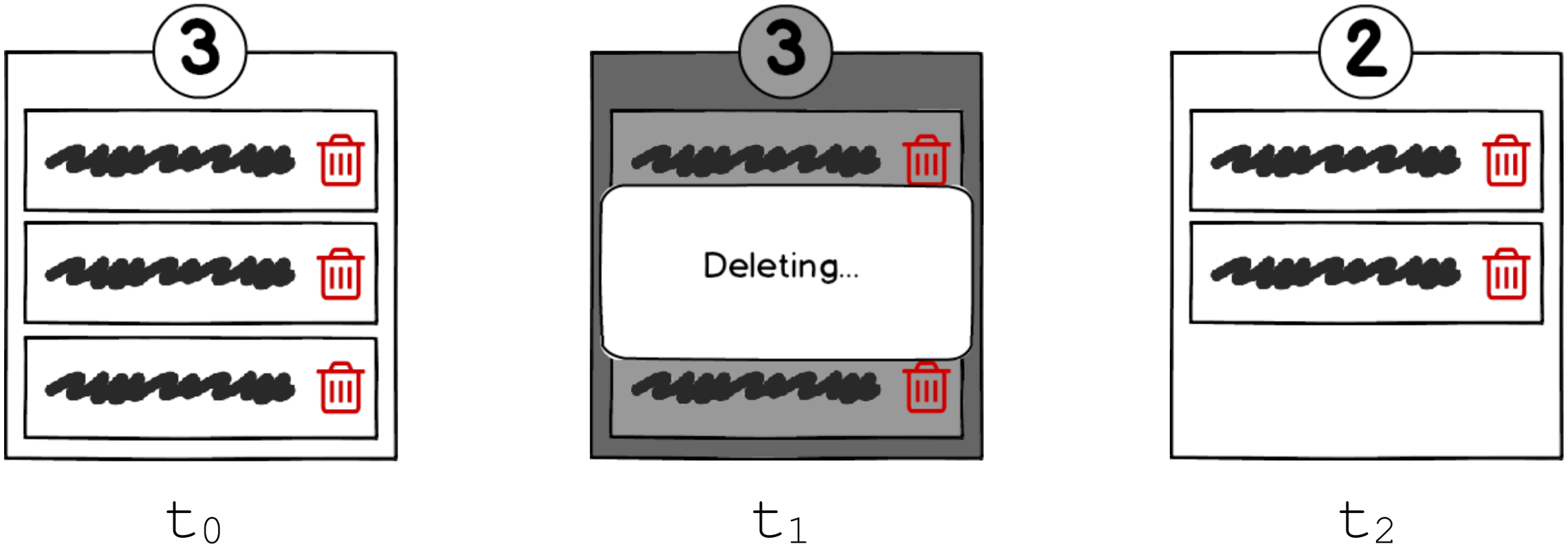
No Time, Only Space



- Wild West: Every component for themselves
- State stored directly in DOM
- No single source of truth ("DB"), but roughly {(e a v), …}

(II)

Epochal Snapshots



- World as a flip book: Redux, Datomic, …
- Transactions: DB -> Tx -> DB
- DB := {(e a v **t**), …}

Snapshot

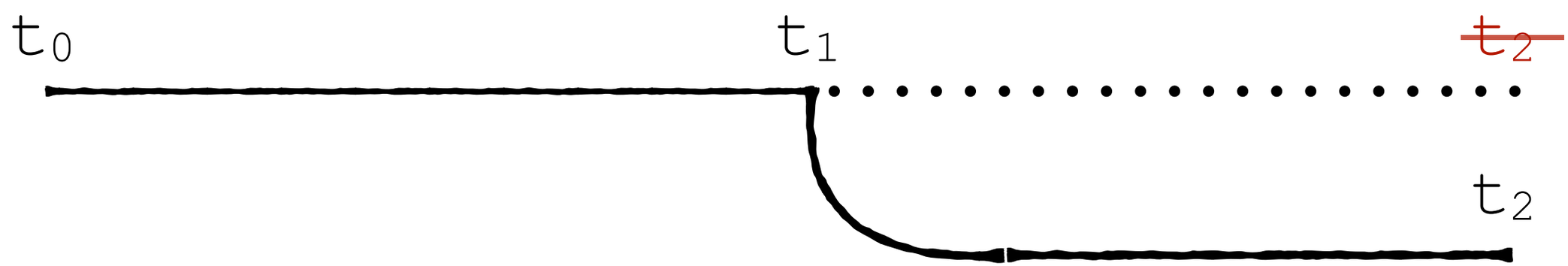
$$DB_{t^*} = \sum_{\{(e\ a\ v\ t)\ |\ t \leq t^*\}}$$

"happened before"

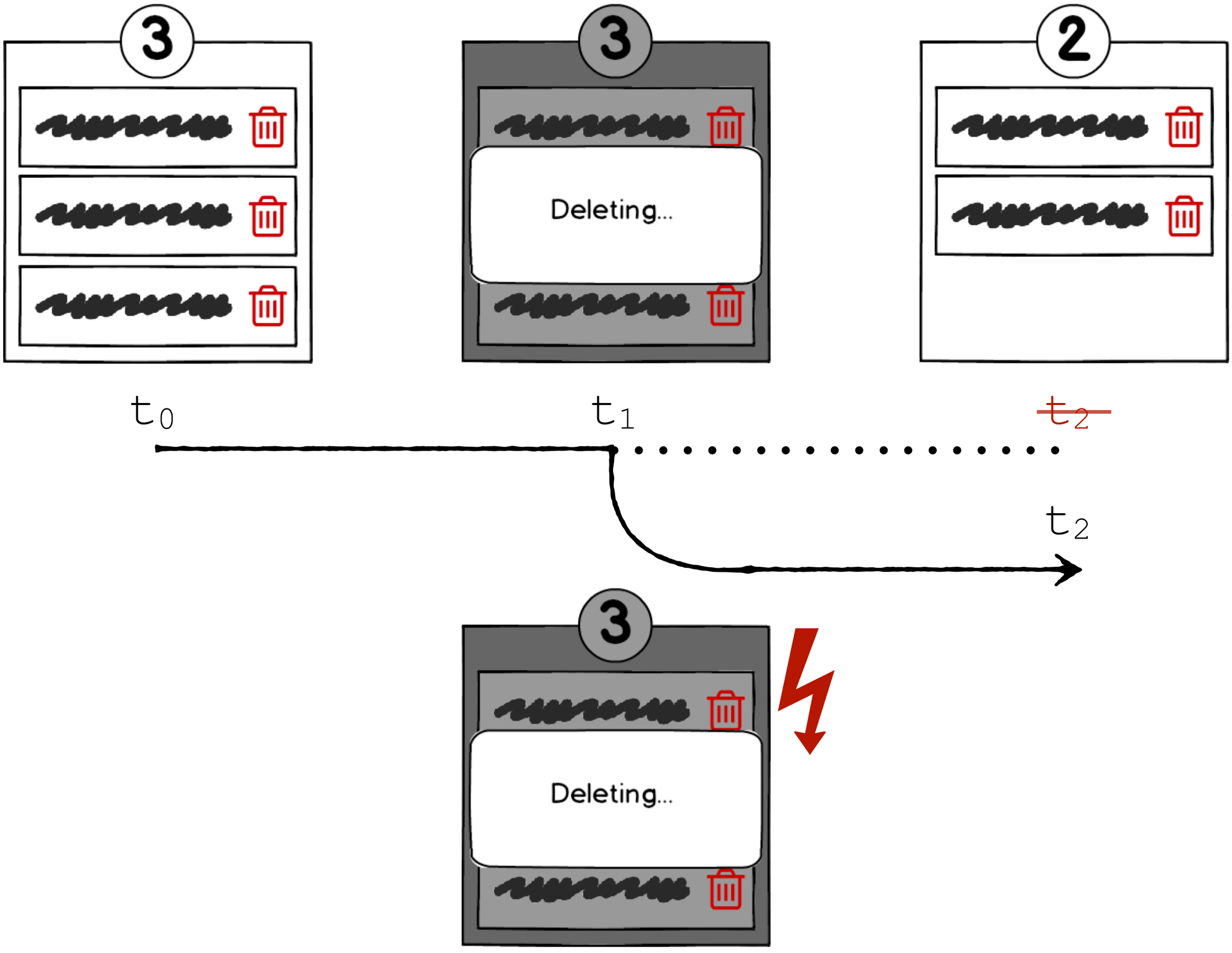
No Retractions (for now)

Time Travel

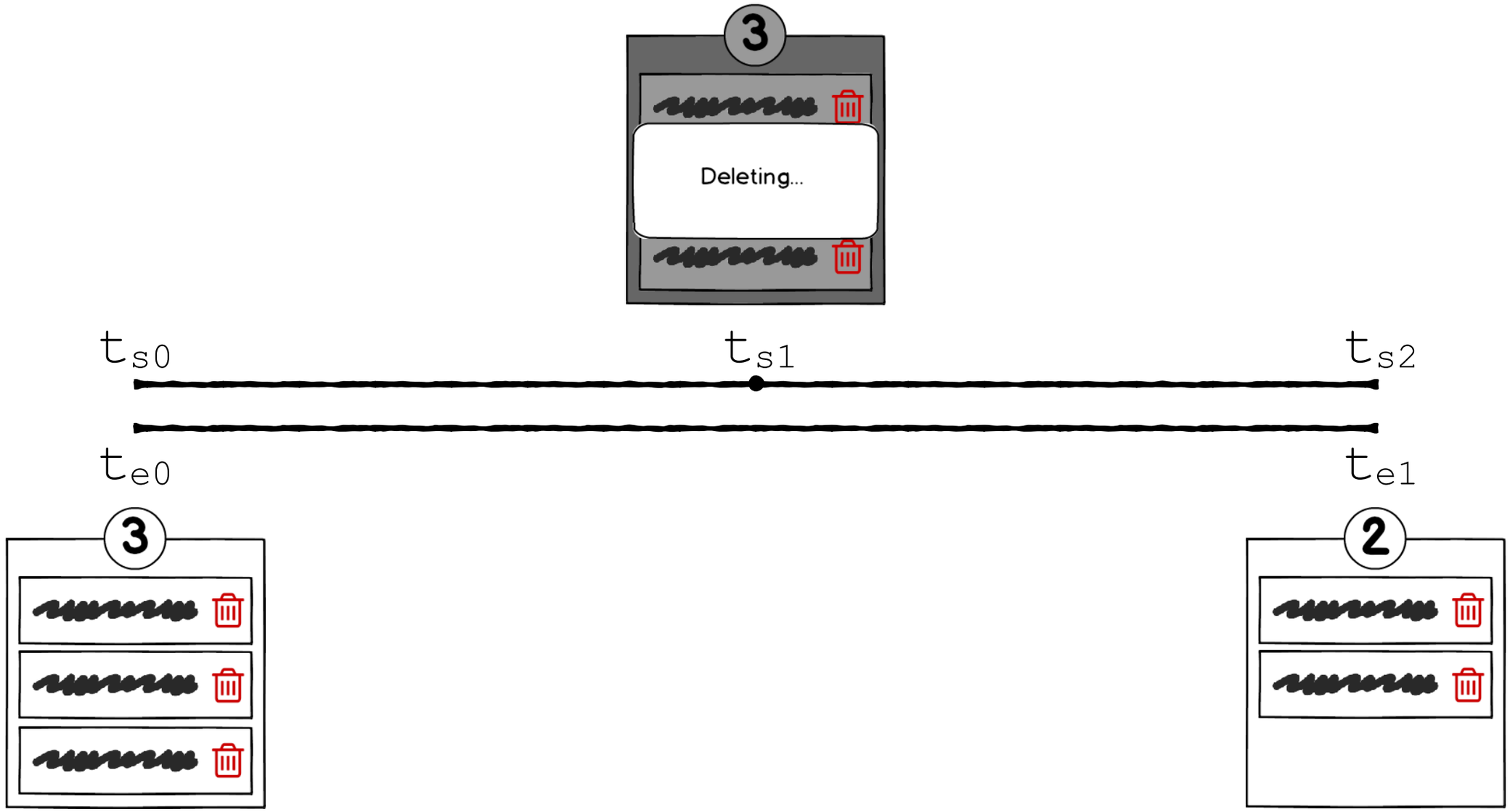
Scrubbing through Snapshots



- Applications: undo / redo
- Destructive writes (single timeline)
- <u>But</u>: All-or-nothing problem


3
3
Deleting...
2
t_0
t_1
t_2
t_2
3
Deleting...

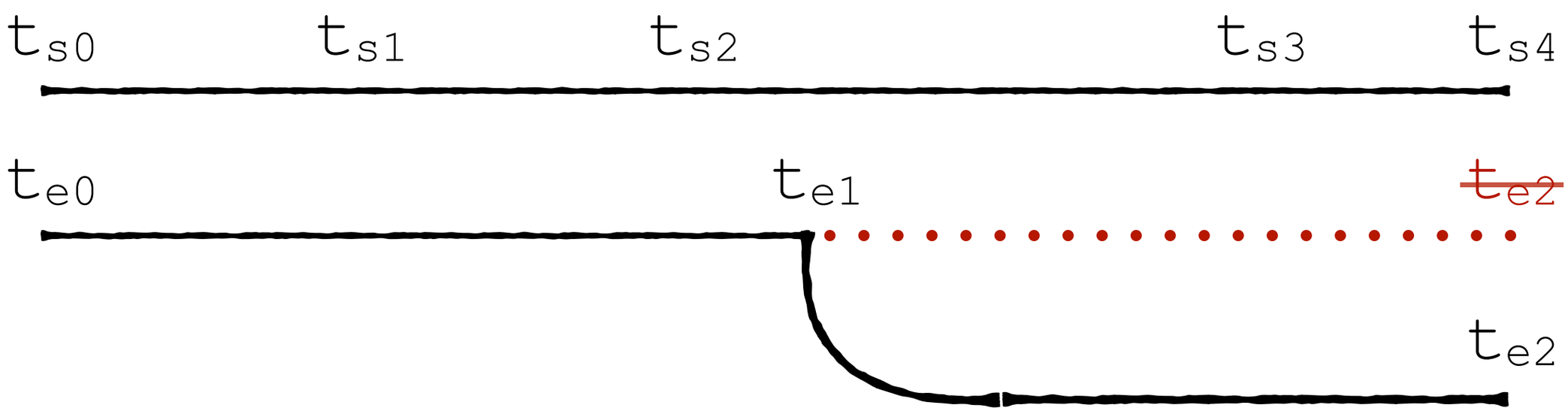
(III)
Bitemporal Snapshots



- 2 Timelines: $t_{\mathbf{system}}$, $t_{\mathbf{event}}$
- System progress independent of domain
- DB := {(e a v ($t_{\mathbf{system}}$,$t_{\mathbf{event}}$)), …}

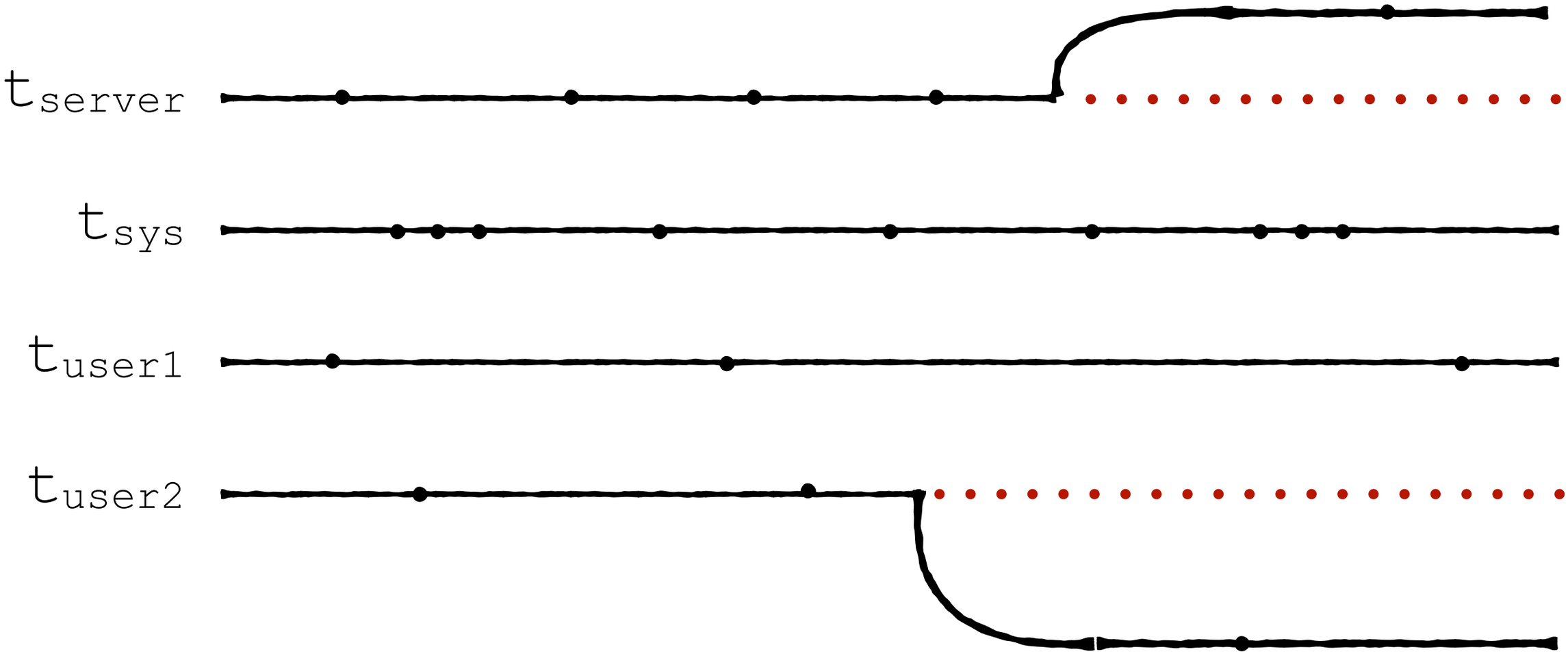
Time Travel II

Solving the All-or-nothing problem



- Scrubbing along $t_{\mathbf{event}}$, UI along $t_{\mathbf{system}}$
- Destructive writes (per timeline)
- All-or-nothing problem solved!

(IV)
Multitemporal Snapshots



- Bitemporal snapshots separate system events from user actions
- Multitemporal snapshots separate *n* sources
- DB := {(e a v ($t_{\mathbf{server}}$, $t_{\mathbf{sys}}$, $t_{\mathbf{user1}}$, $t_{\mathbf{user2}}$, …)), …}

Recap

(and some orders)

$$DB_\star = \sum_{\{(e\ a\ v\ \bullet)\ |\ \bullet \leq \star\}}$$

(I)	(II)	(III)	(IV)
$-$	t	(t_{system}, t_{event})	$(t_{server}, t_{sys}, t_{user1}, t_{user2}, \ldots)$

Recap

(and some orders)

$$DB_\star = \sum_{\{(e\ a\ v\ \bullet)\ |\ \bullet \boxed{\leq} \star\}}$$

"happened before"

(I)	(II)	(III)	(IV)
–	t	(t_{system}, t_{event})	$(t_{server}, t_{sys}, t_{user1}, t_{user2}, \ldots)$
totally ordered		partially ordered	

Recap

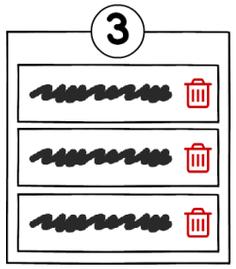
(and some **total** orders)

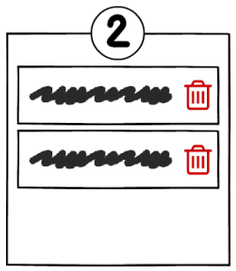
"happened before"

$\leq$	0	1	2
0	✓	✓	✓
1		✓	✓
2			✓

Recap

(and some **total** orders)






```
((msg :deleted? true) t1)
((sys :show-menu? true) t1)

((sys :show-menu? false) t2)
```

Total order coalesces event sources.

Recap

(and some **<u>product partial</u>** orders)

"happened before"

$$\downarrow$$

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

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

Recap

(and some **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!

Recap

(and some **product partial** orders)

"happened before"

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

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

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

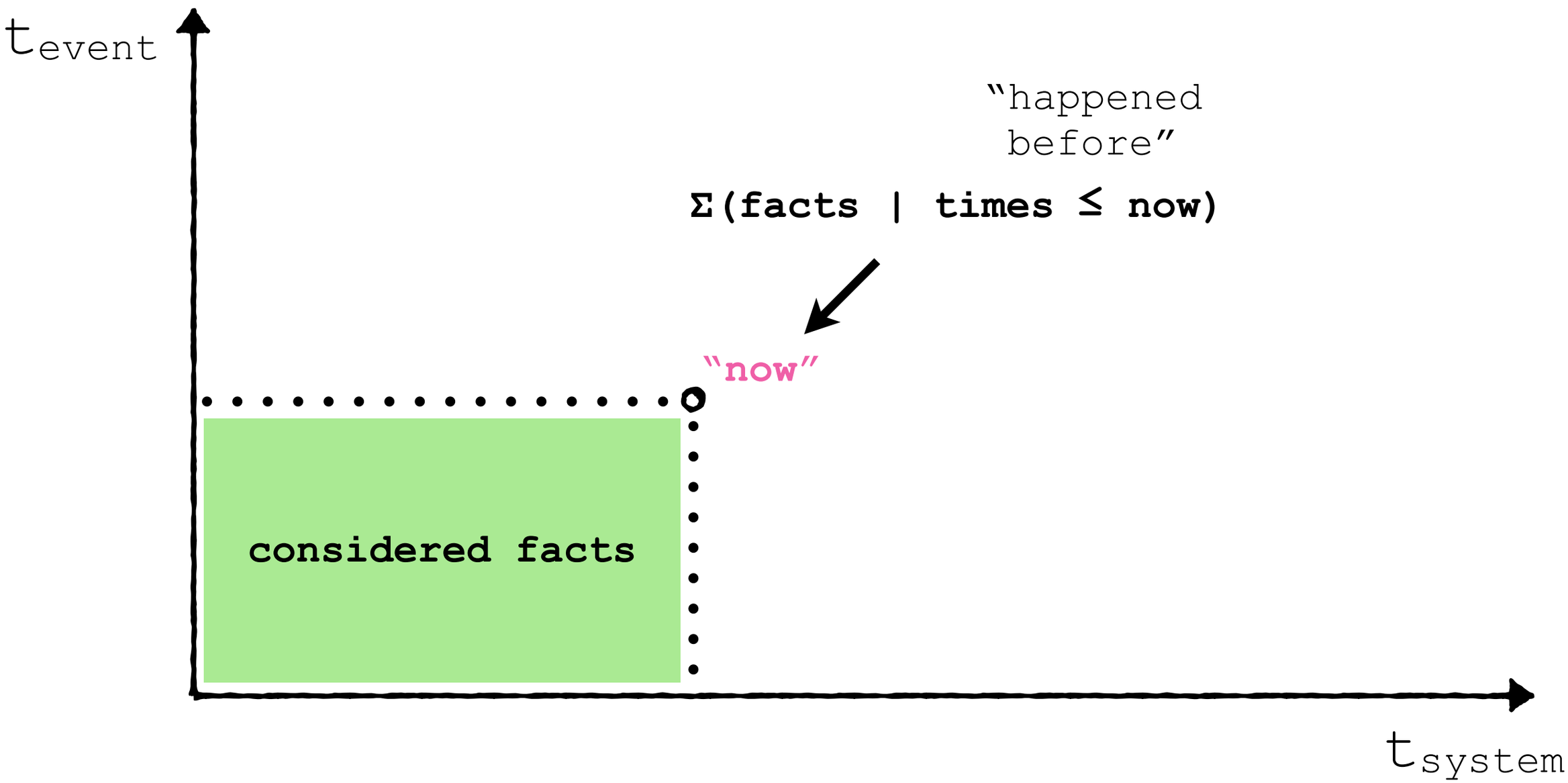
$$(1\ \ 9) \not\leq (2\ \ 5)$$

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

no order!

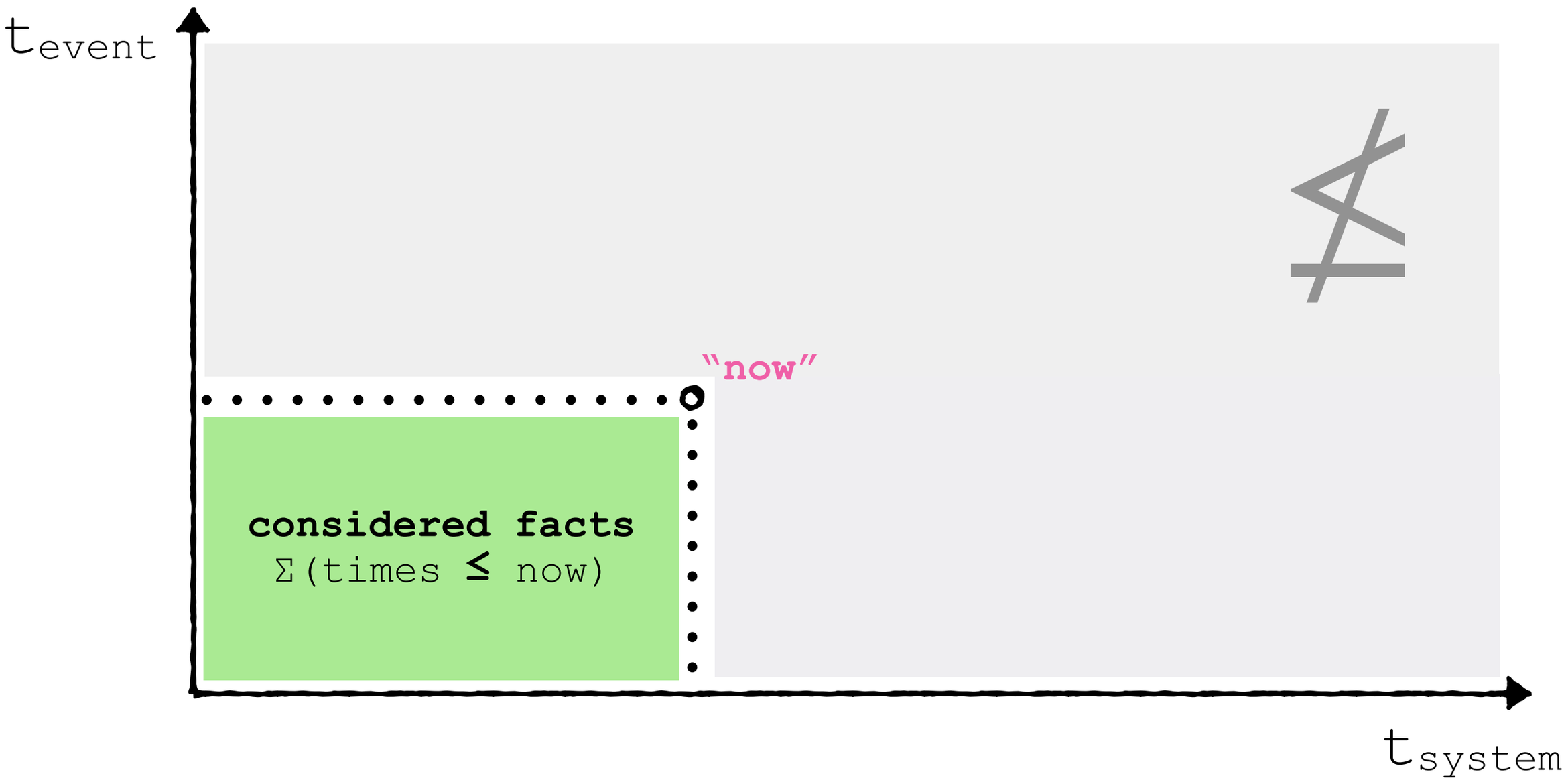
Recap

(and some **product partial** orders)



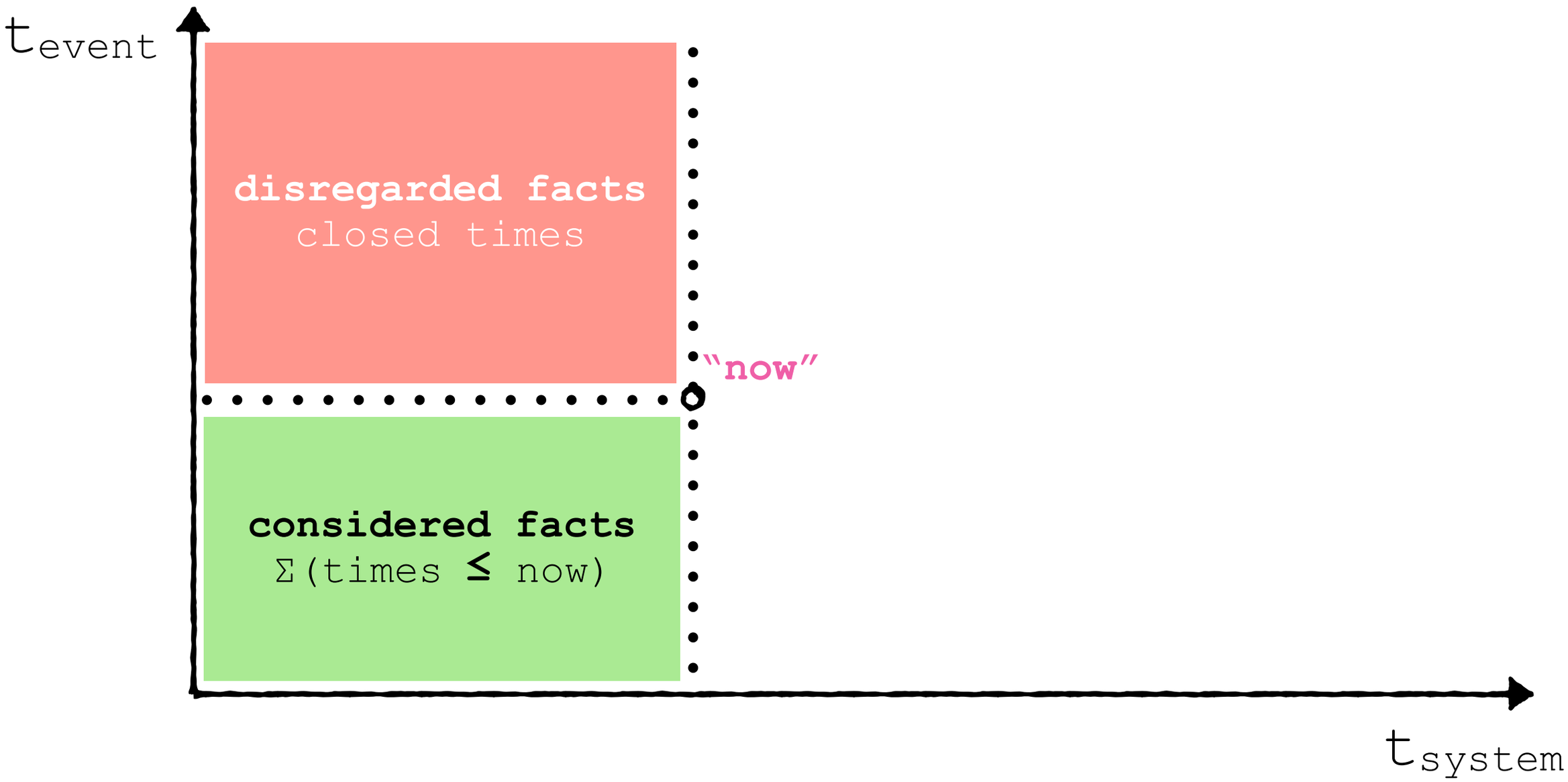
Recap

(and some **product partial** orders)



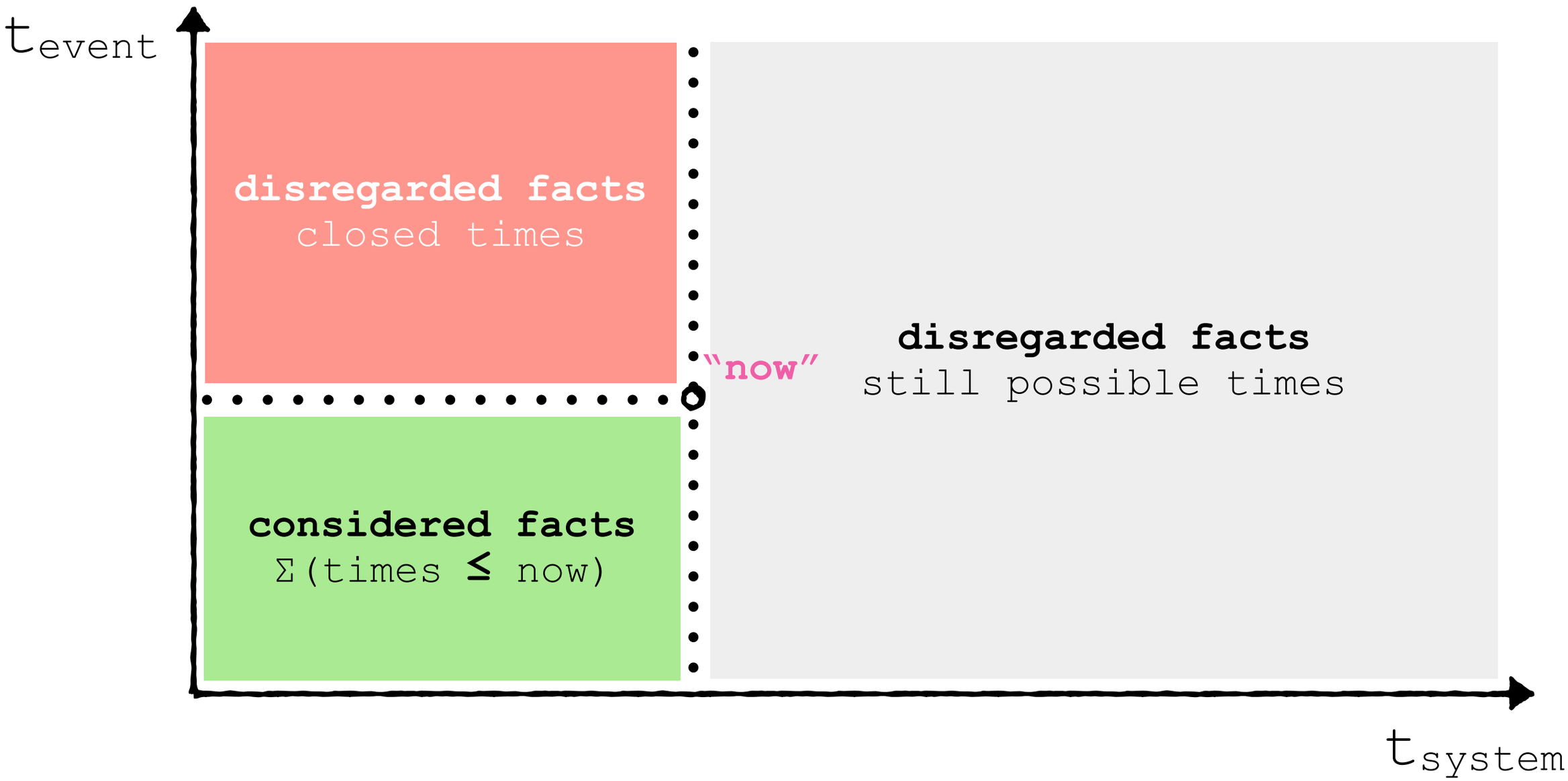
Recap

(and some **product partial** orders)



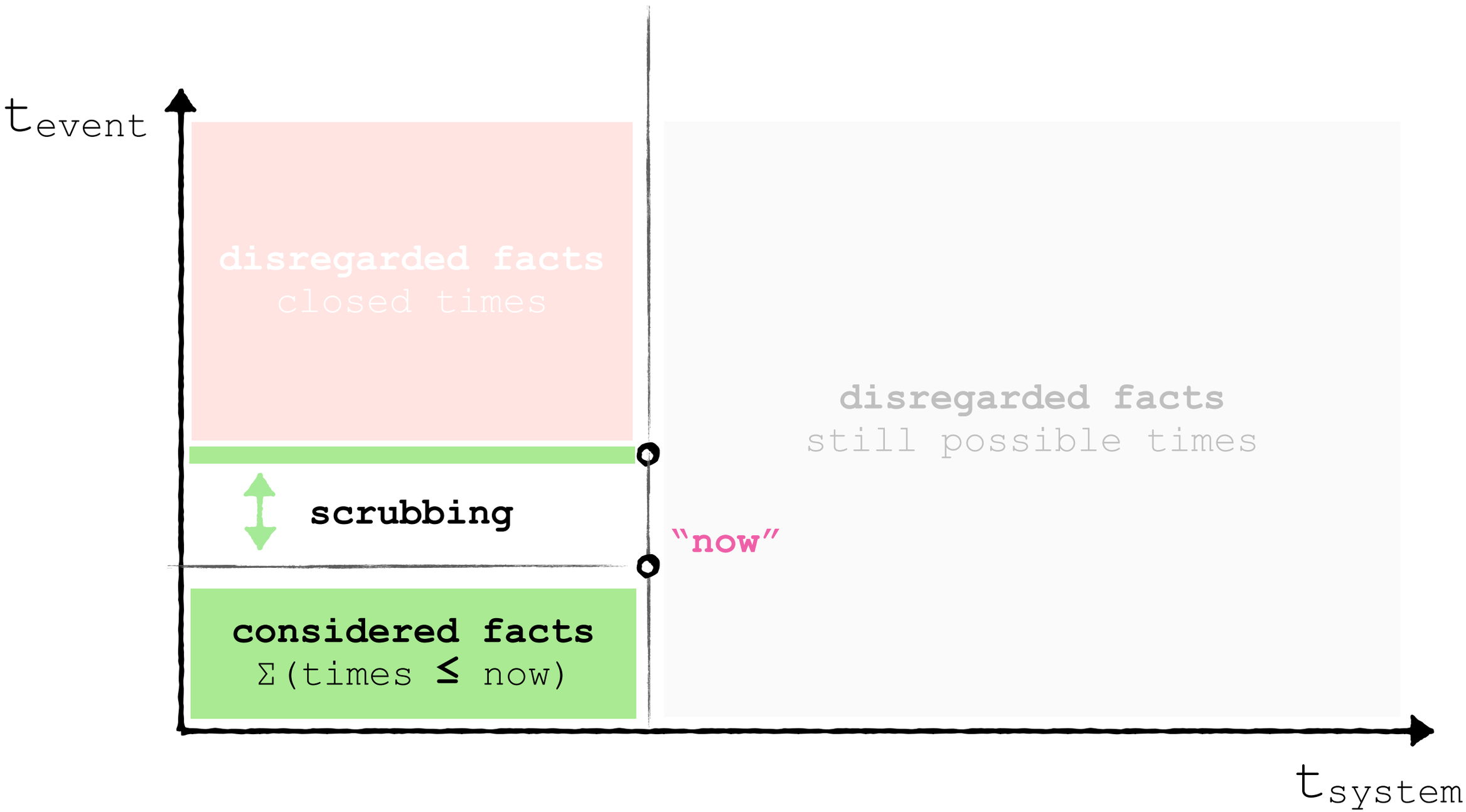
Recap

(and some **product partial** orders)



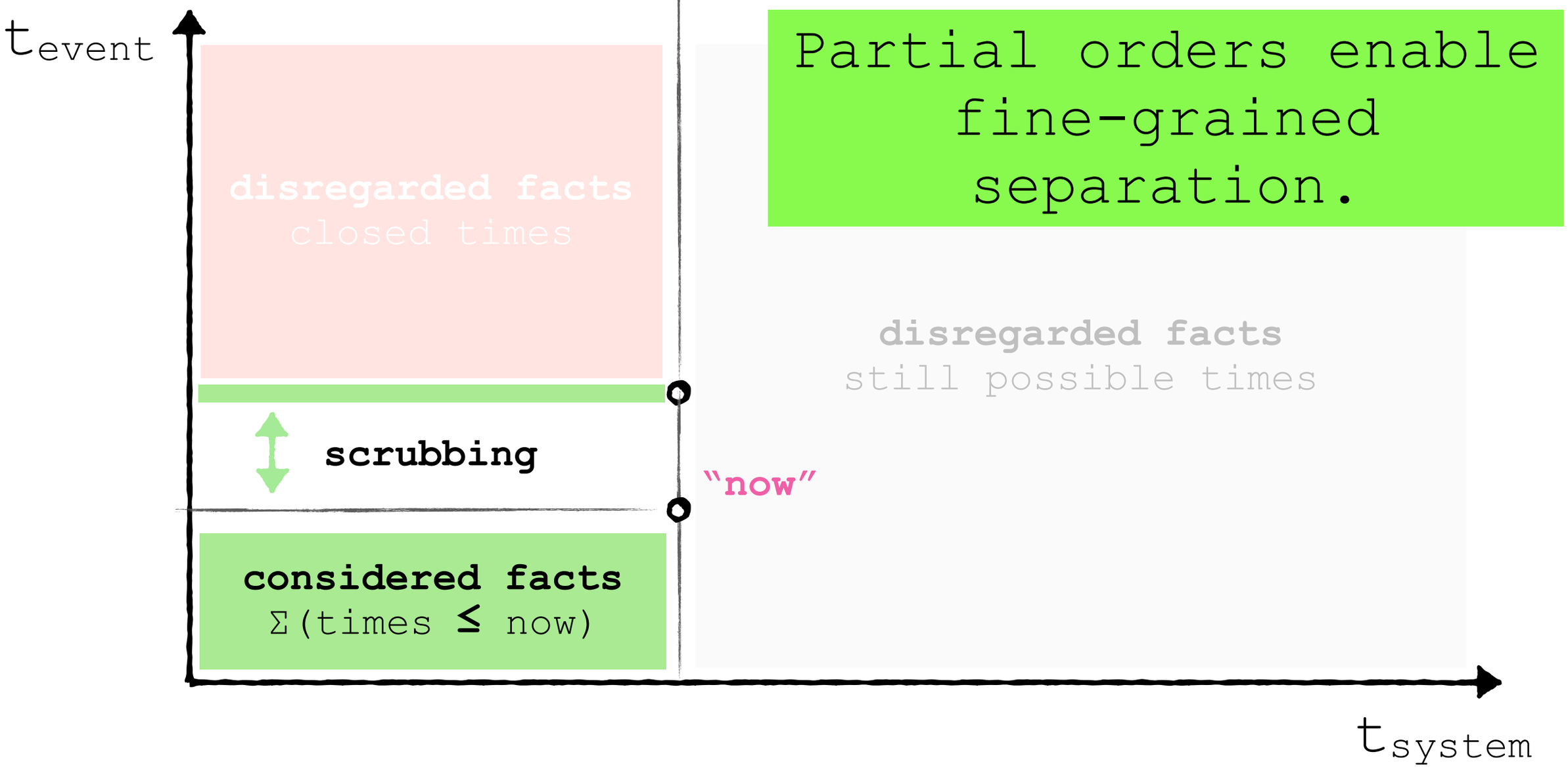
Recap

(and some **product partial** orders)



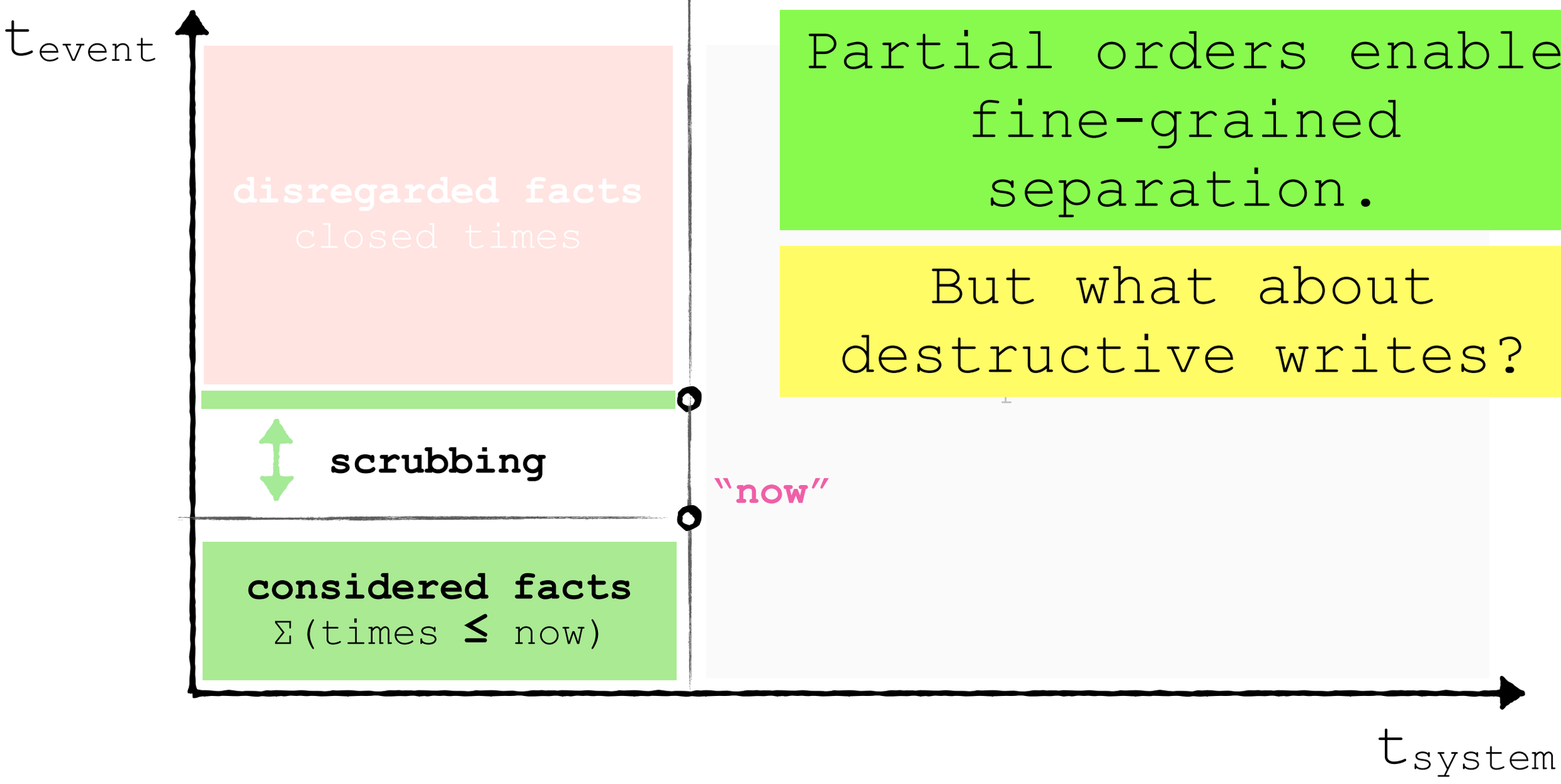
Recap

(and some **product partial** orders)



Recap

(and some **product partial** orders)



Partial orders enable fine-grained separation.

But what about destructive writes?

Non-destructive Writes

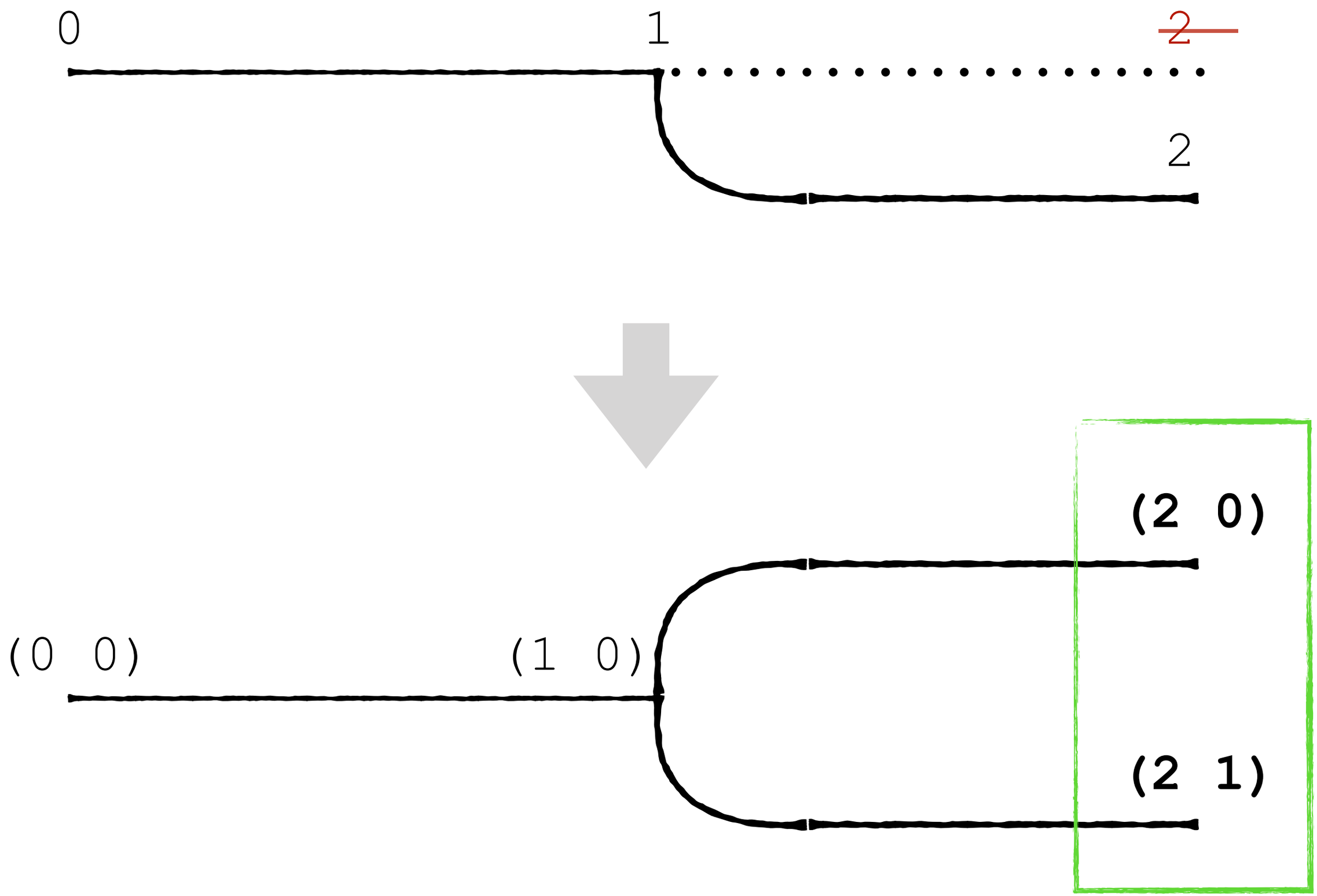
Why bother?

The pasts, the presents, <u>the futures</u>!

- Simulations & what-if
- Collaboration (conflicting edits)
- Functional becomes logic programming

(V)

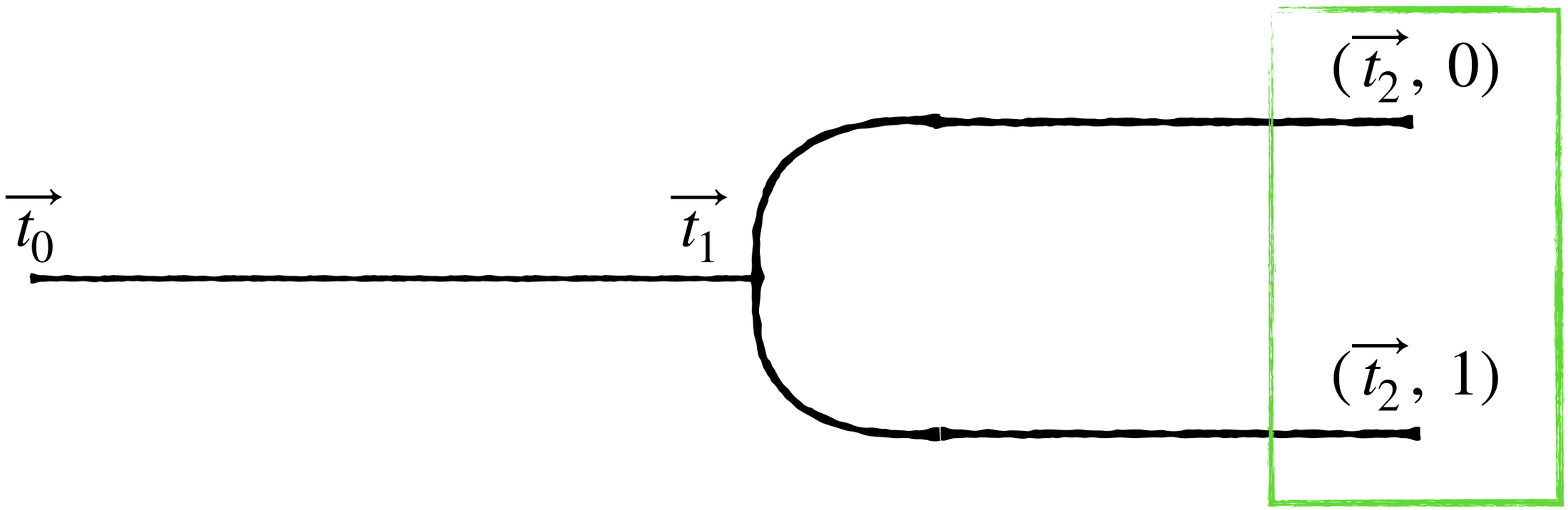
N-DIMENSIONAL


0
1
2
2
(0 0)
(1 0)
(2 0)
(2 1)

Why Not?

(2 0) “happened-before” (2 1)

- Versions should be isolated…
- …but version (2 1) can see version (2 0)
- Reality is leaking!



- Parallel, *isolated* universes
- DB -> Tx -> DB becomes DB -> Tx -> **[**DB**]**
- DB := {(e a v ((t$_{sys}$,t$_{user1}$,…), **η**)), …}

A New Order

bi-temporal

(a, b) `happened-before` (c, d)

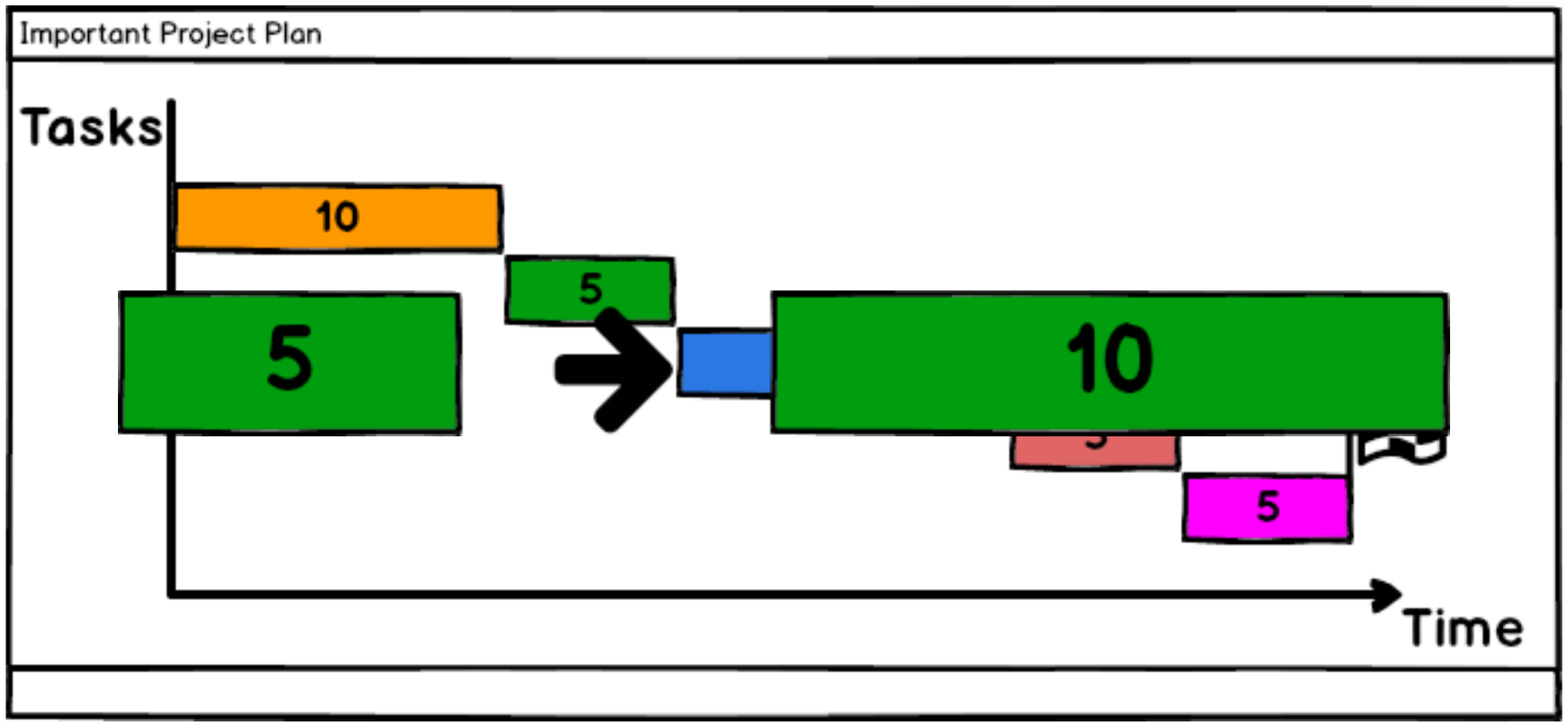
$$\Leftrightarrow (a \leq c) \wedge (b \leq d)$$

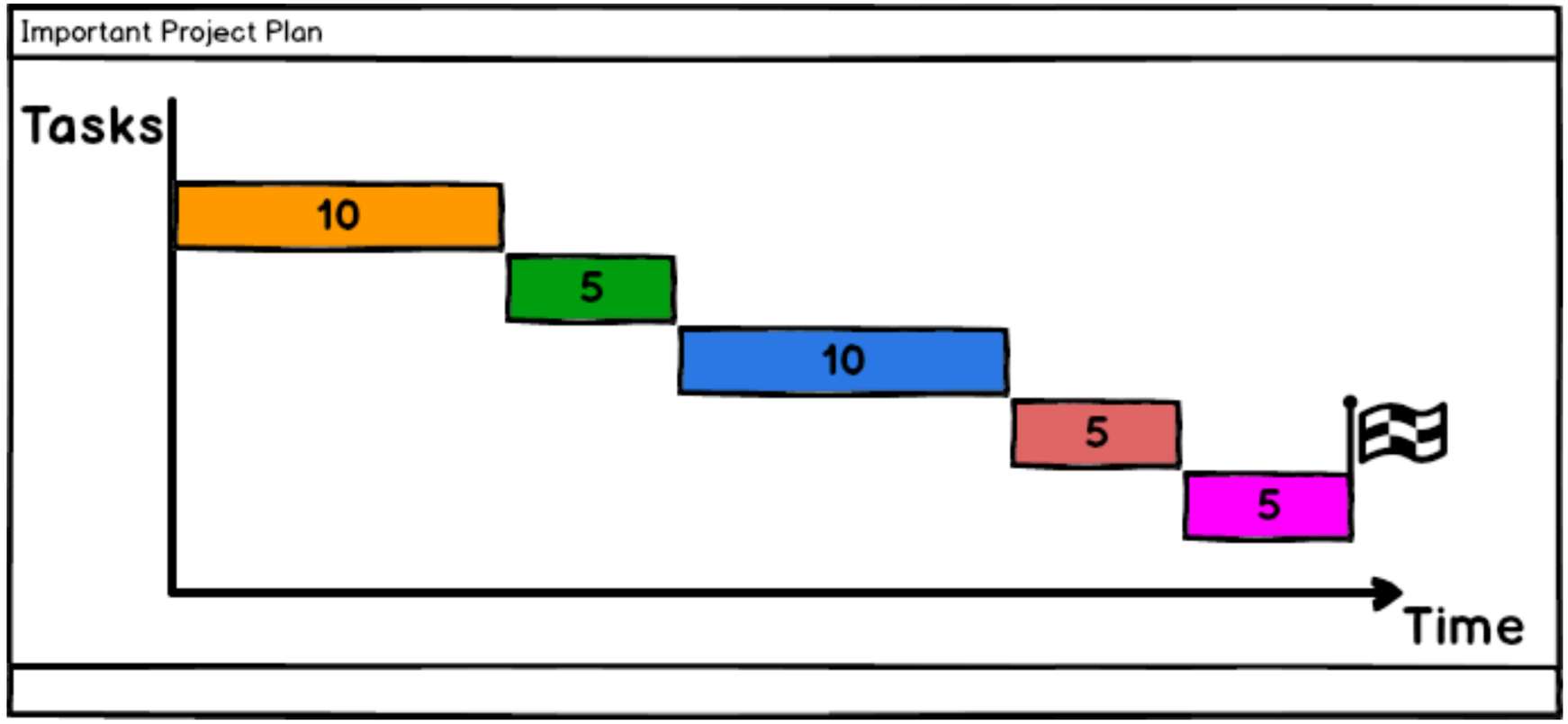
---

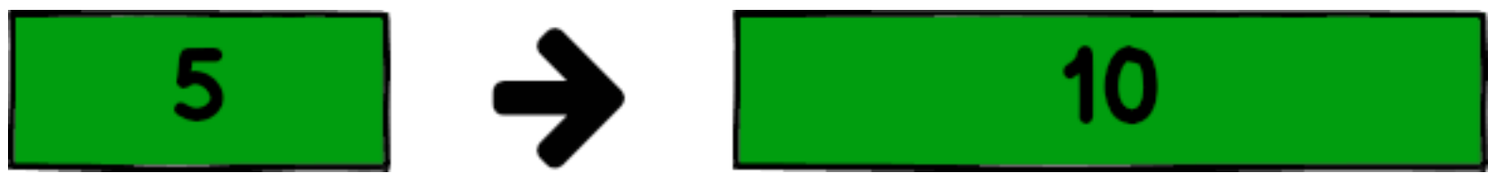
bi-dimensional

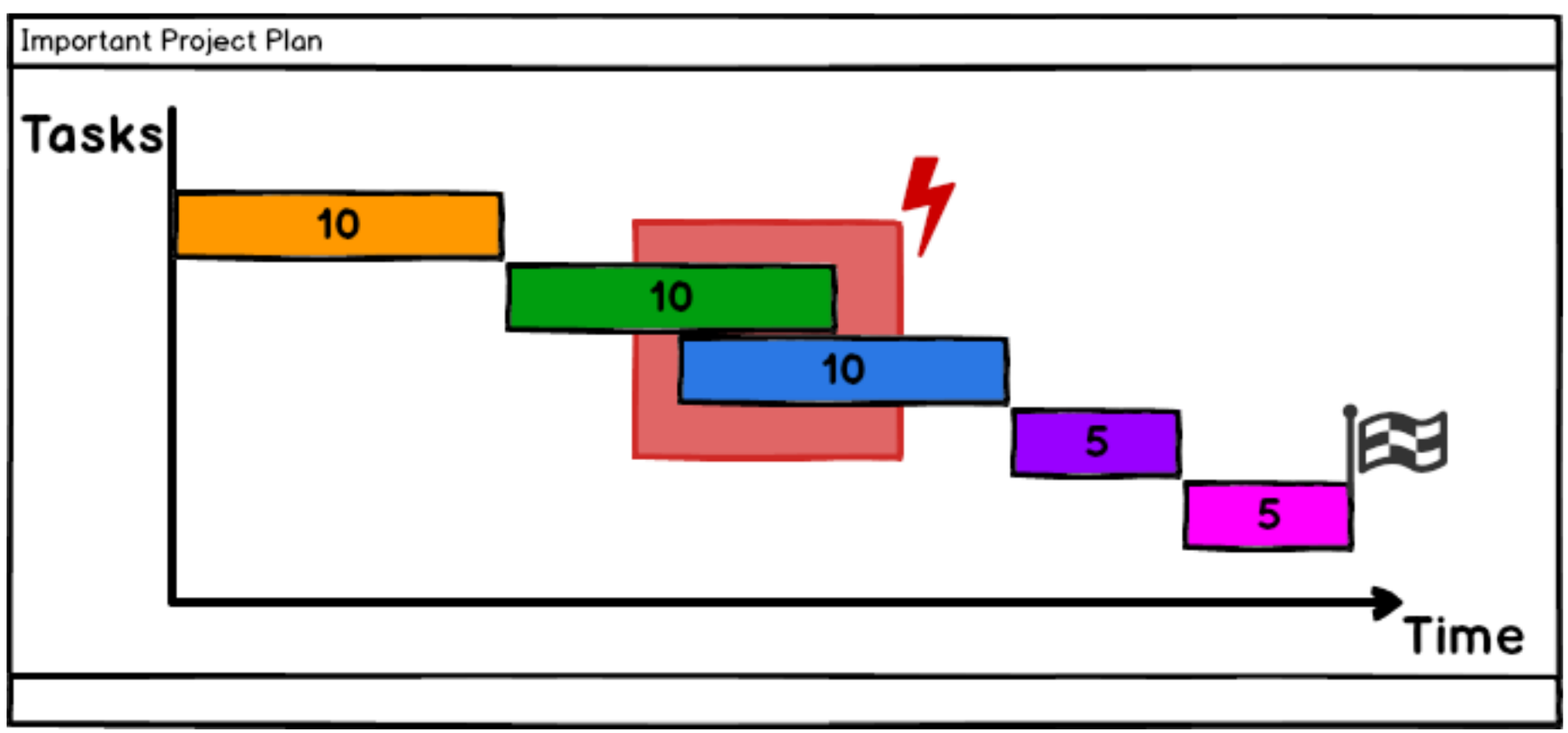
$((a, b), \eta_1)$ `happened-before` $((c, d), \eta_2)$

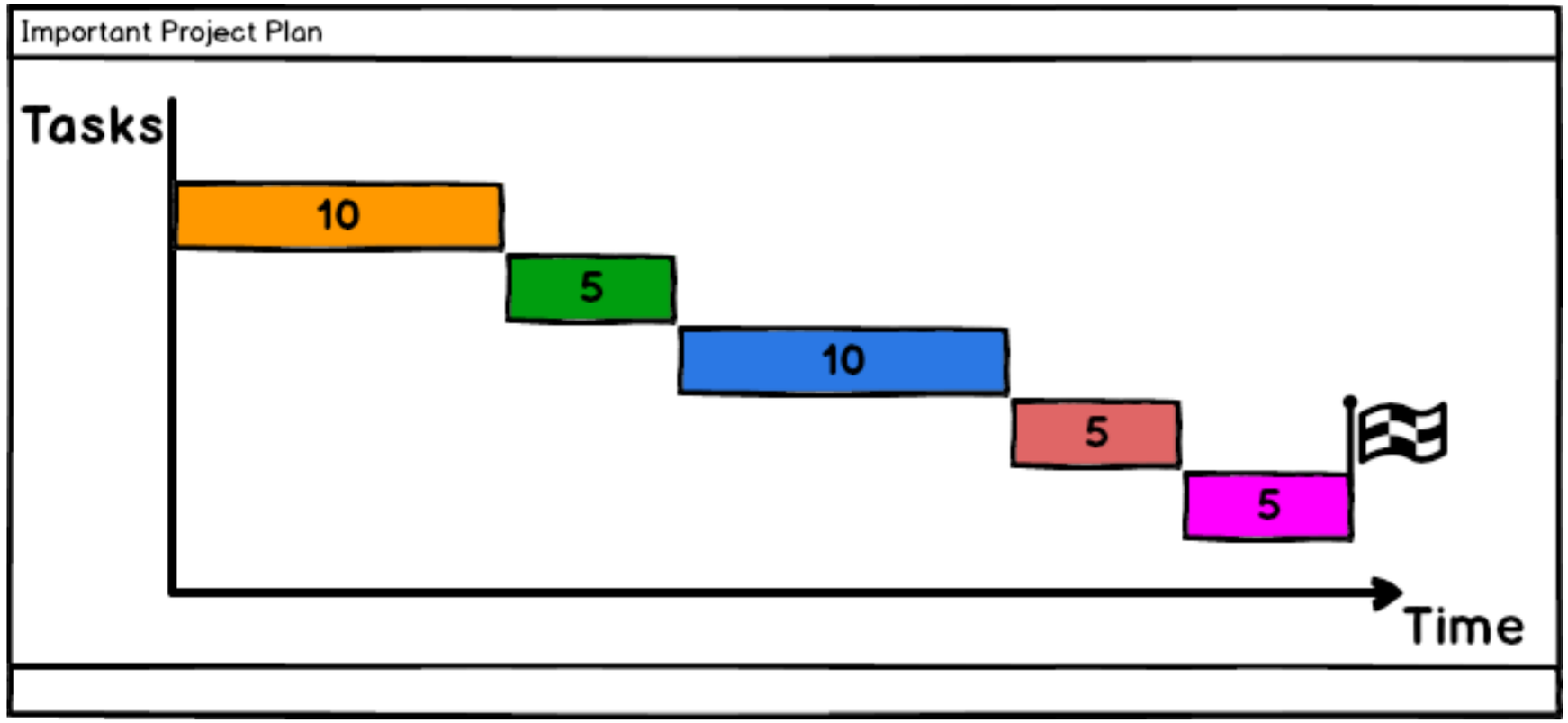
$$\Leftrightarrow (a \leq c) \wedge (b \leq d) \wedge (\eta_1 = \eta_2)$$

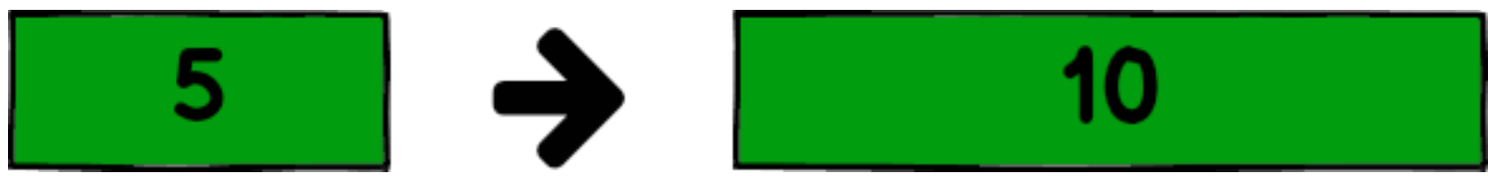

Important Project Plan
Tasks
10
5
5
10
5
Time

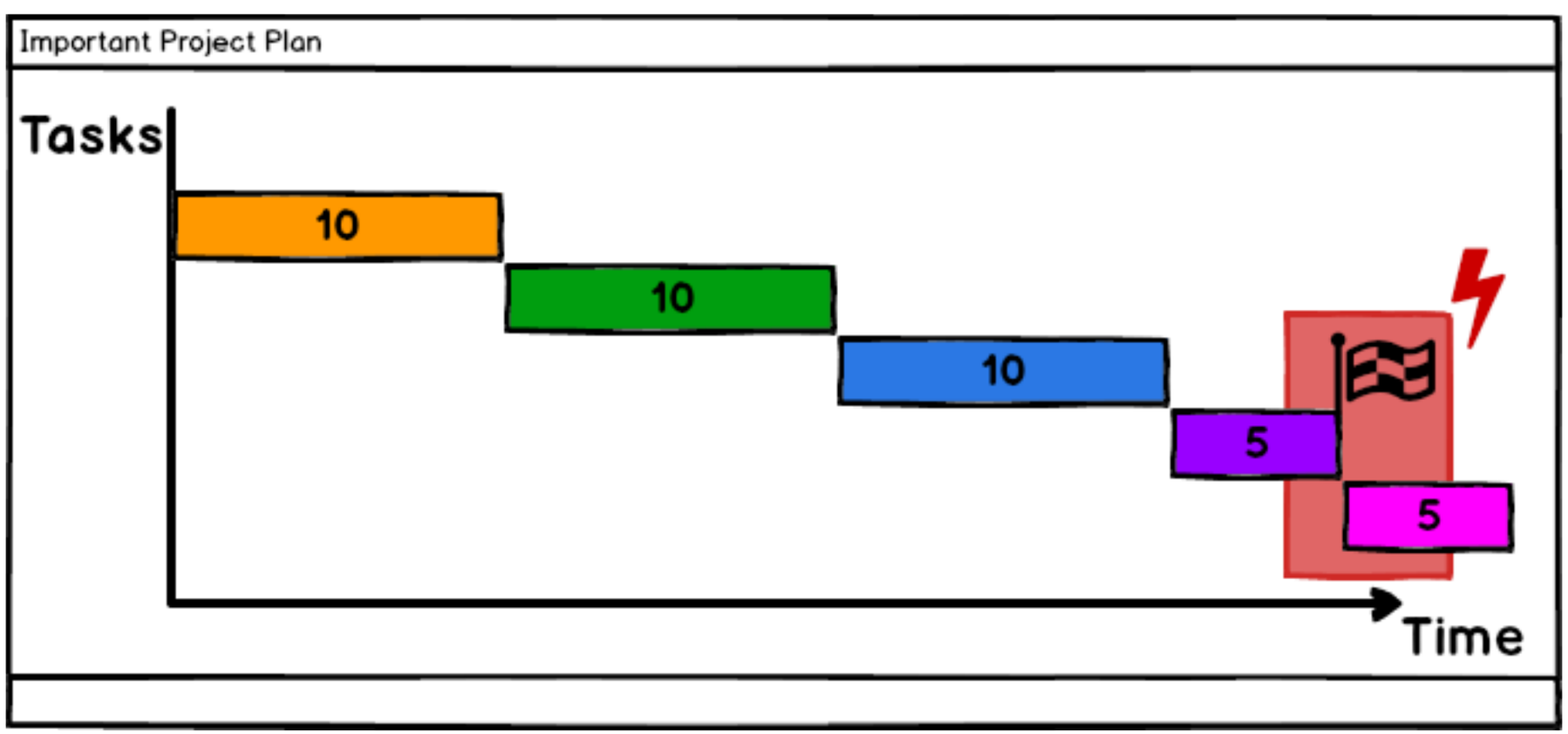

Important Project Plan
Tasks
10
5
10
5
5
Time

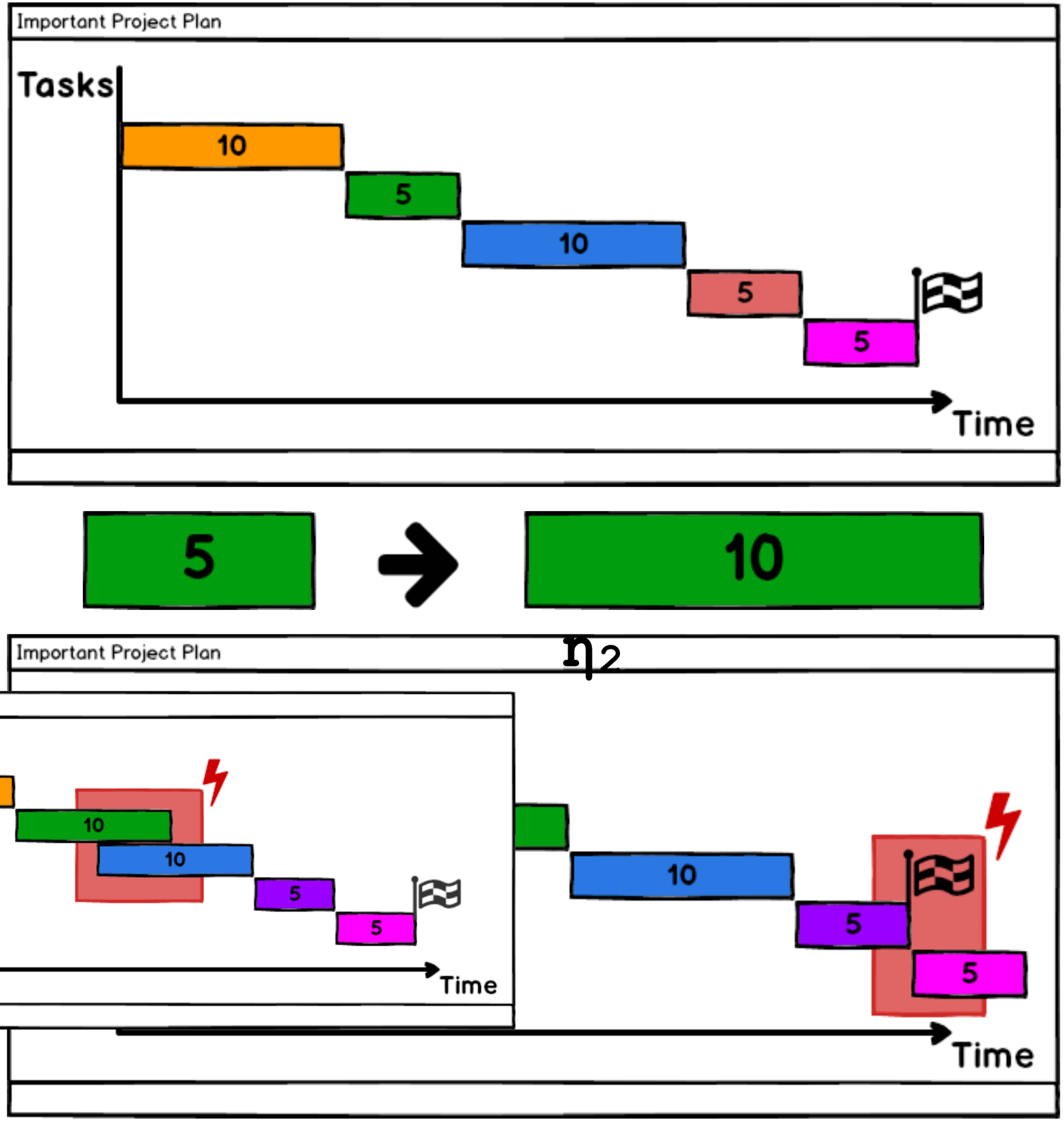

5
10


Important Project Plan
Tasks
10
10
10
5
5
Time


Important Project Plan
Tasks
10
5
10
5
5
Time


5
10


Important Project Plan
Tasks
10
10
10
5
5
Time


Important Project Plan
Tasks
10
5
10
5
5
Time
5
10
η1
Important Project Plan
Tasks
10
10
10
5
5
Time
η2
Important Project Plan
10
5
5
Time

(VI)

Differential Programming

Implementation Challenges?

(I) Can't *copy* state everywhere.

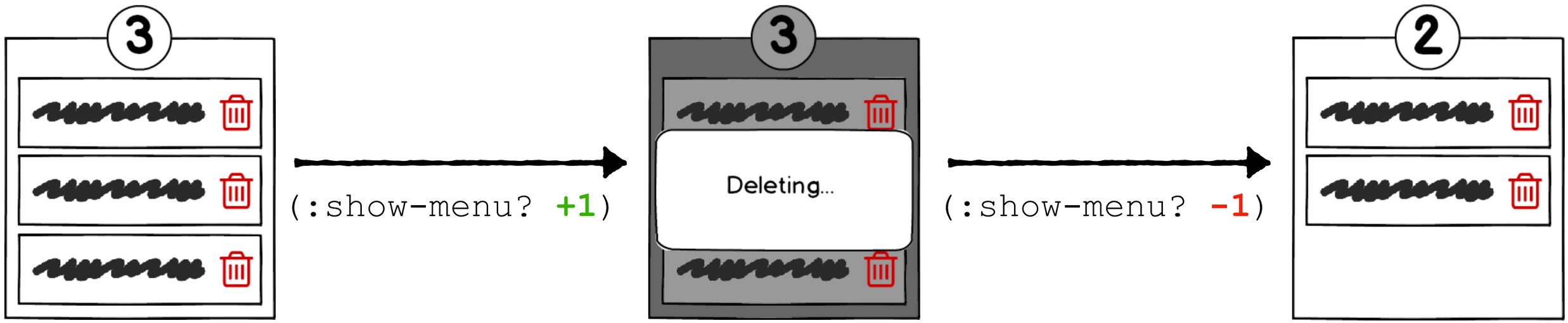
(II) Can't *re-compute* everything from scratch for each universe.

Snapshot

$$DB_{t^*} = \sum_{\{(e\ a\ v\ t)\ |\ t \leq t^*\}}$$

No Retractions (for now)

Multiplicities



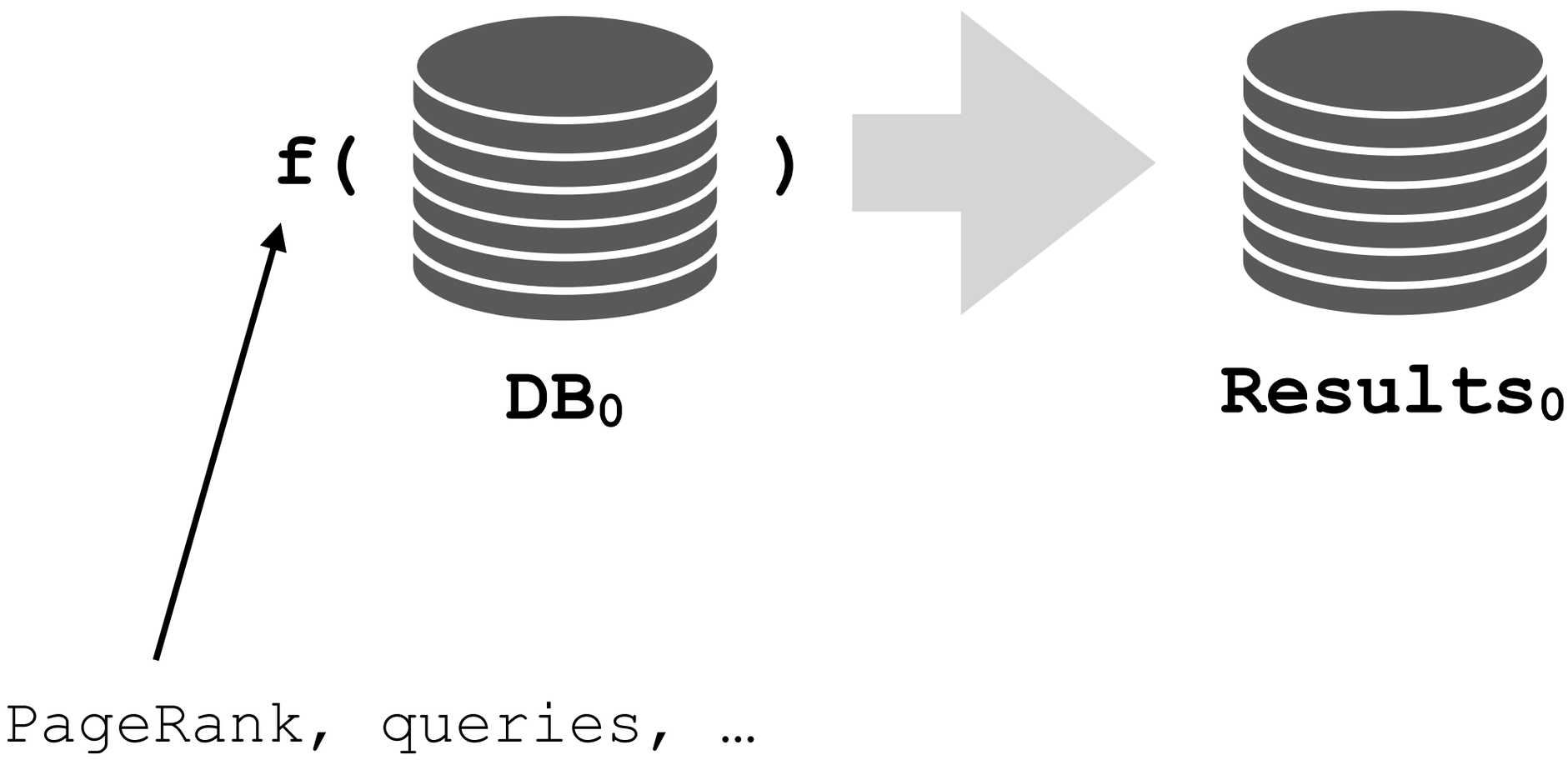
- Addition: **+1**, Retraction: **-1**
- Sets become Bags
- Store Diffs, not full Snapshots

Implementation Challenges?

(I) ~~Can't *copy* state everywhere.~~

(II) Can't *re-compute* everything from scratch for each universe.

Computing w/ Snapshots



Incremental Computation

How do we find ∂f?

Differential Dataflow

- Incrementalized computational framework
- Fine-grained concurrency: partial orders!
- Data-parallel & distributable
- github.com/TimelyDataflow

Think in f, get distributable δf for free!

```
/// 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)))
})
```

We Talked About Time!

1. Semantics
   - Epochs for basic sanity
   - Multitemporal for undo/redo
   - Multidimensional for versioning
2. Efficient Implementation
   - Store Diffs, not Snapshots
   - Incremental Computation
   - Differential Dataflow

thanks!

@NikolasGoebel

niko@clockworks.io

@MalteSandstede

malte@clockworks.io