Event sourcing after page 1

How we built a cloud native bank

KCD Denmark 2023 / Thomas Bøgh Fangel / LUNAR

Image rendered by NightCafé

Me

- Joined Lunar in 2016
- Tech lead in squad Orion responsible for domestic clearing integrations
- Distributed systems since 2004
- Part of the Lunar journey from Rails monolith to event driven microservices

Reach me at

@tbfangel

thomasboeghfangel

LUNAR®

- Founded 2015
- “Technology company running a bank”
- Live (as Lunar Way) April 2016
- **Banking license 2019**
- Live as Lunar in Q1 2020
- 750k customers across DK/NO/SE

- Cloud Native from Day 1
- Live on Kubernetes in Q1 2017
- Event sourcing used since 2019
- ~300 application services in K8S prod
- ~100 services using event sourcing

How would you build the core of a modern bank?

LUNAR

Core Values

We're dealing with people's money…

1. **Correctness**
   (No surprise 🤪)

2. **Explainability**
   Bugs happen, unexpected things **will** happen in production.

   "We should always be able to understand and explain the state of the system"

LUNAR

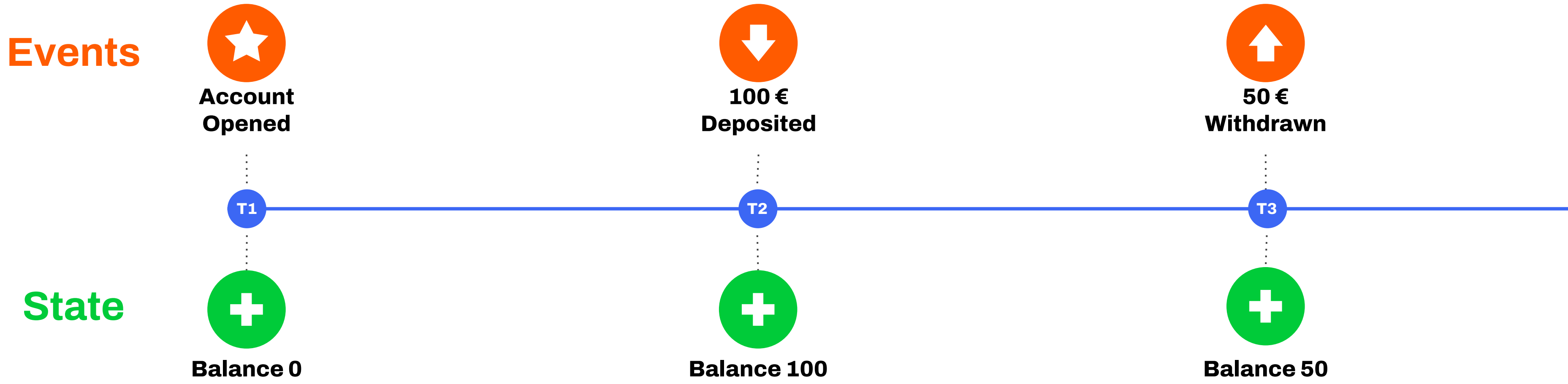
Event Sourcing

“ A persistence model where all changes to a system is stored as an immutable sequence of events”

Page 1 Example

“ A good example of a software system using event sourcing as a persistence model is a financial transaction system or a banking application.”

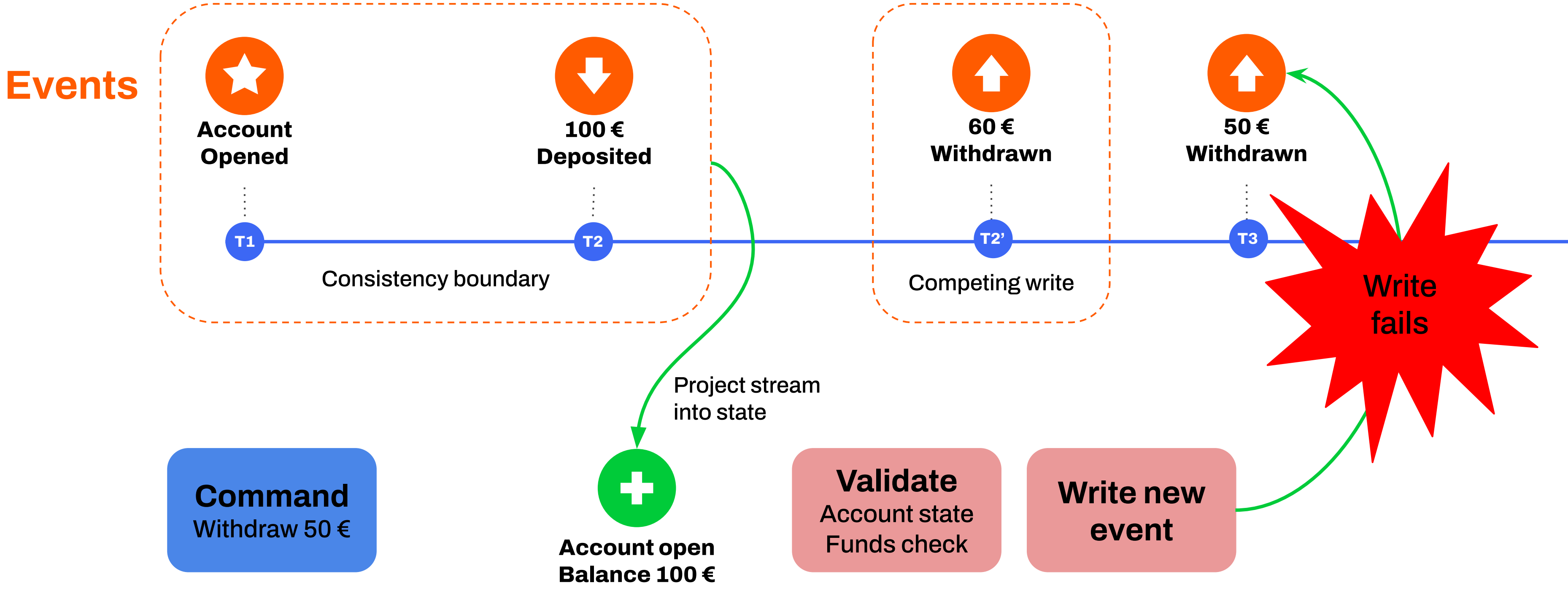
Page 1 example: the account



State = FoldLeft(zero, []events)

Implementing event sourcing

Writing to a stream



LUNAR

Implementing event sourcing

Build

- Complete control, but you're on your own
- Maintenance burden
- Lunar Go library (closed source)
  - Postgres as storage - well known technology is a strength
  - Simple SQL unique constraint to guarantee consistent writes

Buy

- Available products:
  - EventStore, Axon Framework, Lagom, EventFlow
  - Open source alternatives
- Evaluate cost and complexity
- Technology and platform match

The challenge of long and long lived event streams

LUNAR

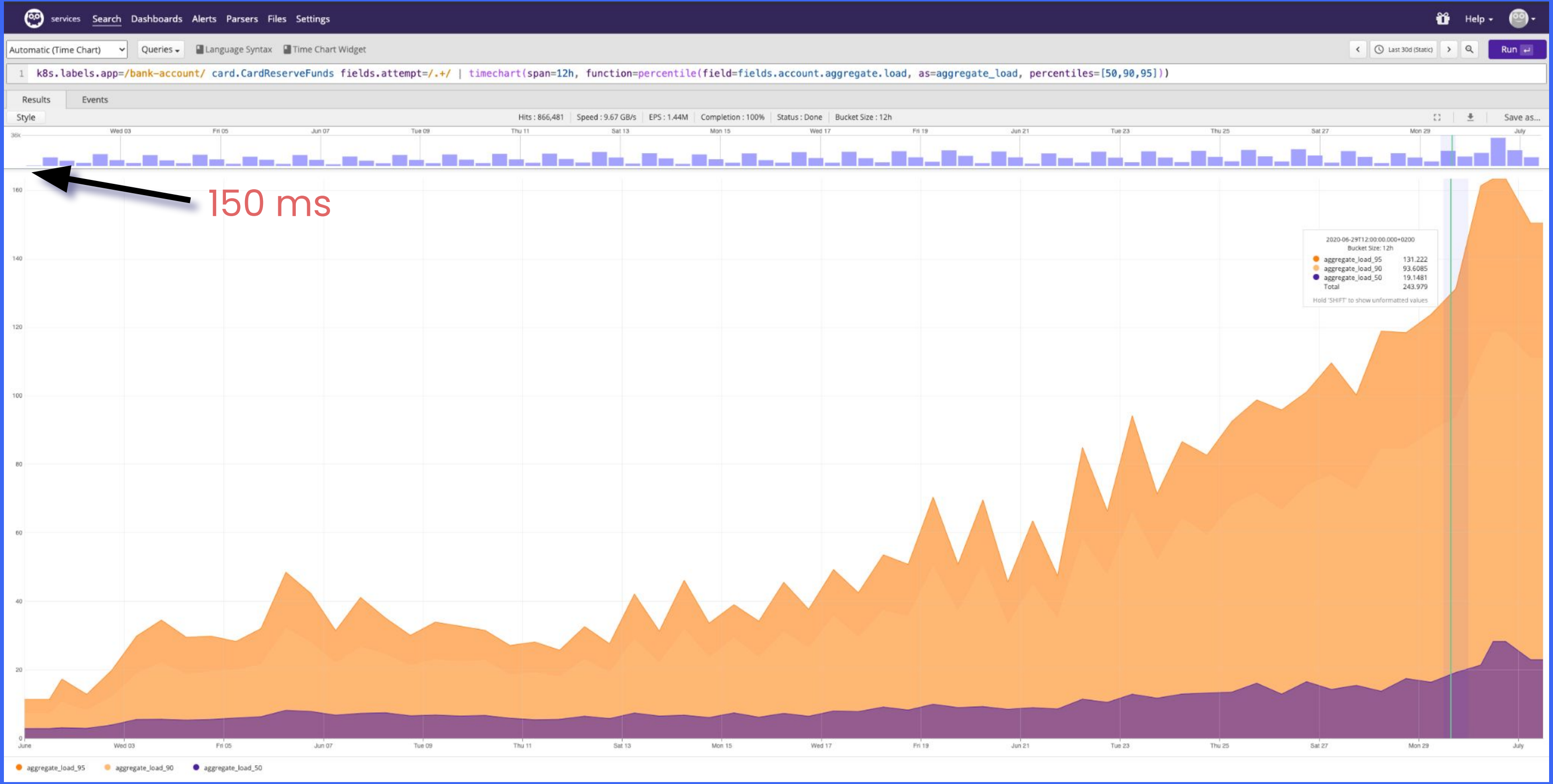
The account event stream

Stats

1. 800k account streams
2. 50k+ events on some streams
3. 3+ years
4. Evolution of our understanding of the domain, so lots of event evolution

LUNAR

What happens when loading long event streams?



```
k8s.labels.app=/bank-account/ card.CardReserveFunds fields.attempt=/.+/ | timechart(span=12h, function=percentile(field=fields.account.aggregate.load, as=aggregate_load, percentiles=[50,90,95]))
```

Hits : 866,481 | Speed : 9.67 GB/s | EPS : 1.44M | Completion : 100% | Status : Done | Bucket Size : 12h

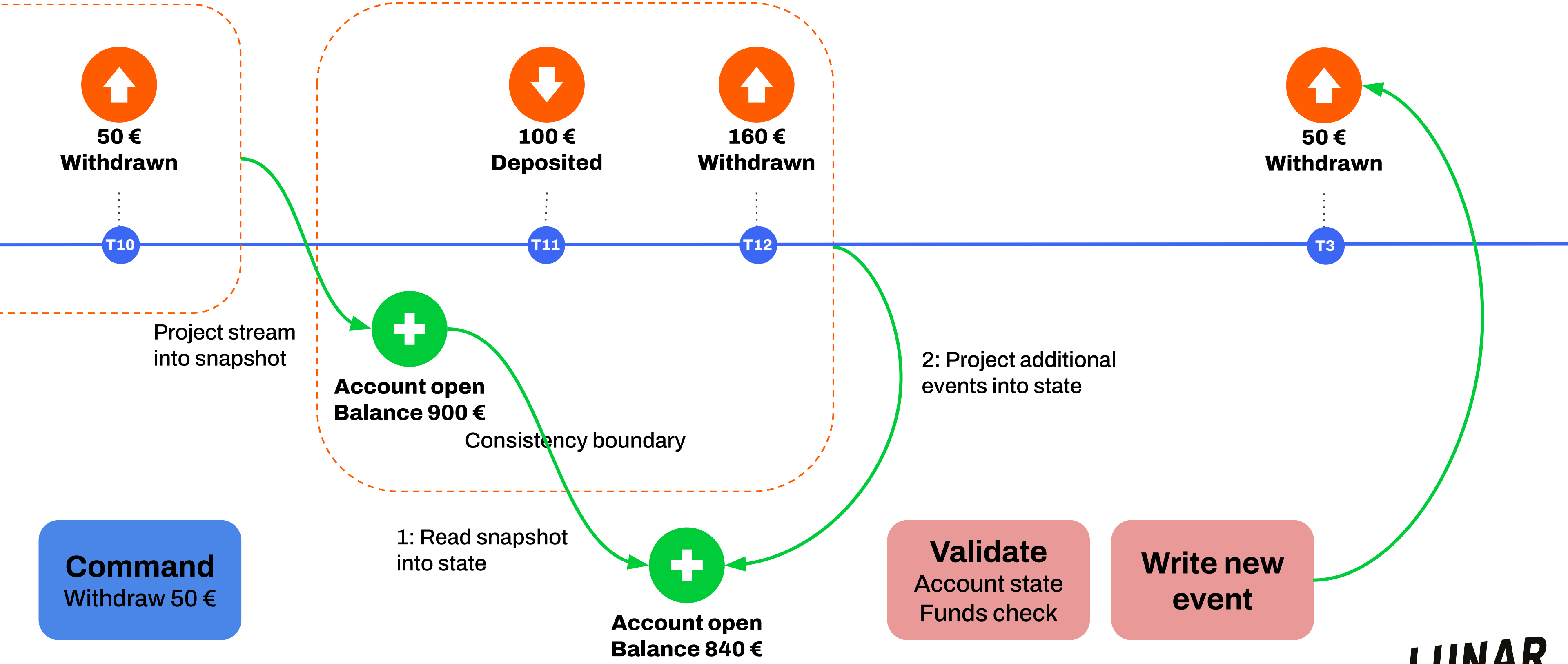
2020-06-29T12:00:00.000+0200
Bucket Size: 12h

● aggregate_load_95	131.222
● aggregate_load_90	93.6085
● aggregate_load_50	19.1481
Total	243.979

Hold 'SHIFT' to show unformatted values

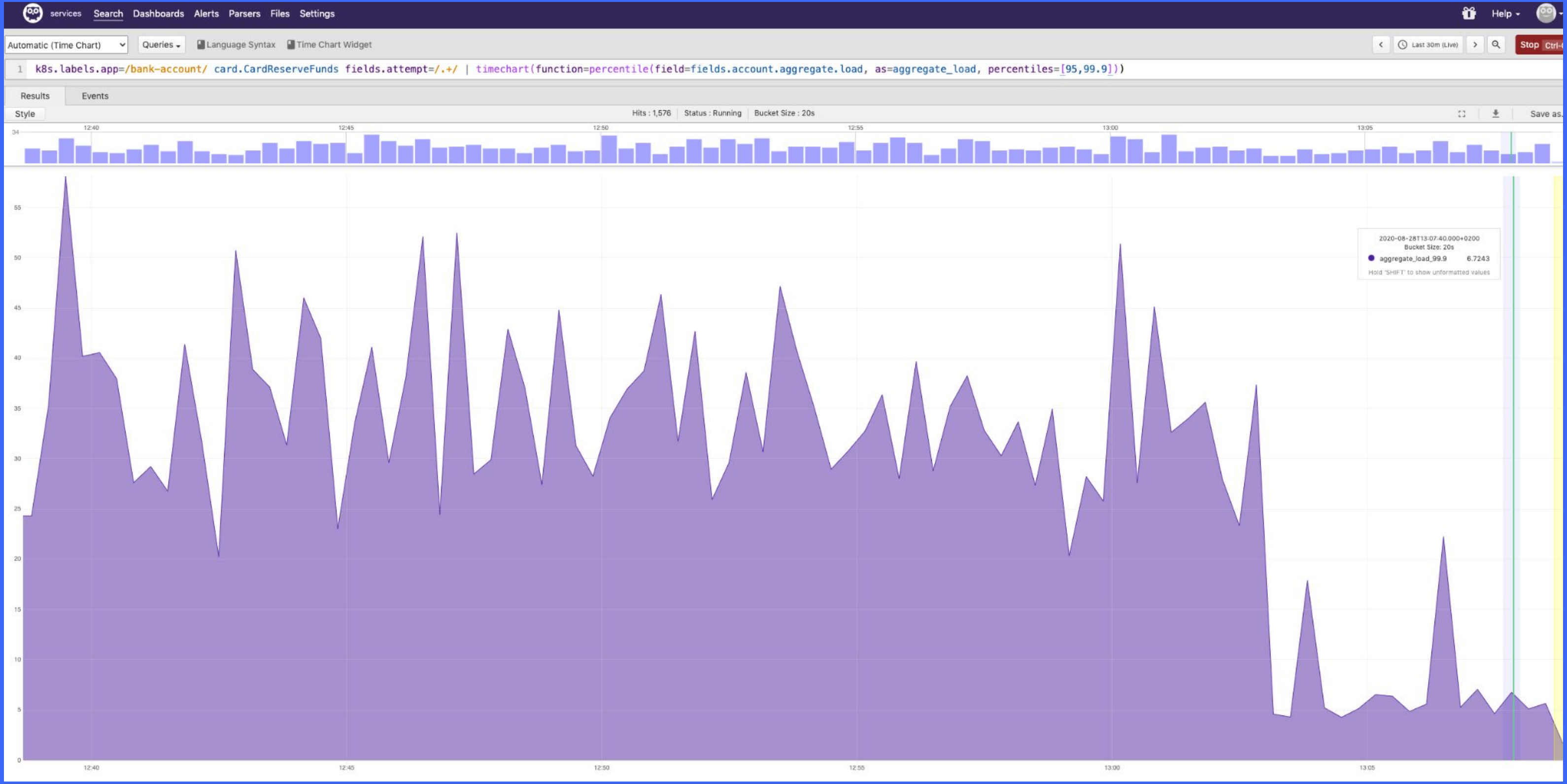
LUNAR

Snapshotting



LUNAR

The effect of snapshotting



```
k8s.labels.app=/bank-account/ card.CardReserveFunds fields.attempt=/.+/ | timechart(function=percentile(field=fields.account.aggregate.load, as=aggregate_load, percentiles=[95,99.9]))
```

Hits : 1,576 | Status : Running | Bucket Size : 20s

2020-08-28T13:07:40.000+0200
Bucket Size: 20s
● aggregate_load_99.9 6.7243
Hold 'SHIFT' to show unformatted values

LUNAR

Key take aways

Snapshotting

1. Absolute must-have for long event streams
2. Independently of writing new events
3. Evolving the state becomes a challenge - warm-up snapshots on version bumps

LUNAR

```go
type AccountState struct {
  ID          ID
  Created     *time.Time
  Closed      *time.Time
  Balance     decimal.Decimal

  // Transactions keeps track of
  // transactions.
  // NB! This map is unbounded.
  Transactions map[ID]bool

  // Reservations keeps track of
  // reservations
  // NB! This map is unbounded.
  Reservations map[ID]Reservation
}
```

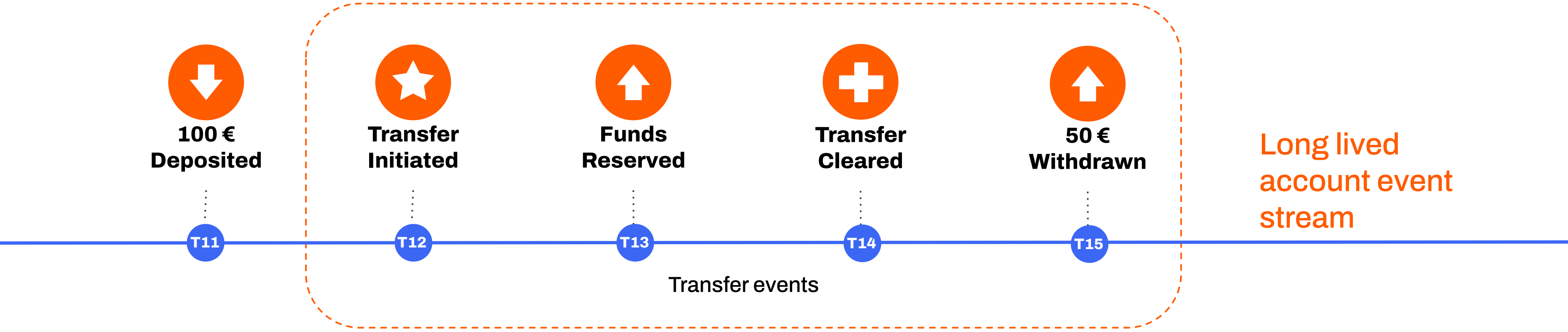
Unbounded State

- Beware of unbounded state in projections
- Solve idempotency differently - move out of the projection

Modeling the domain with event sourcing

LUNAR

Modeling a transfer

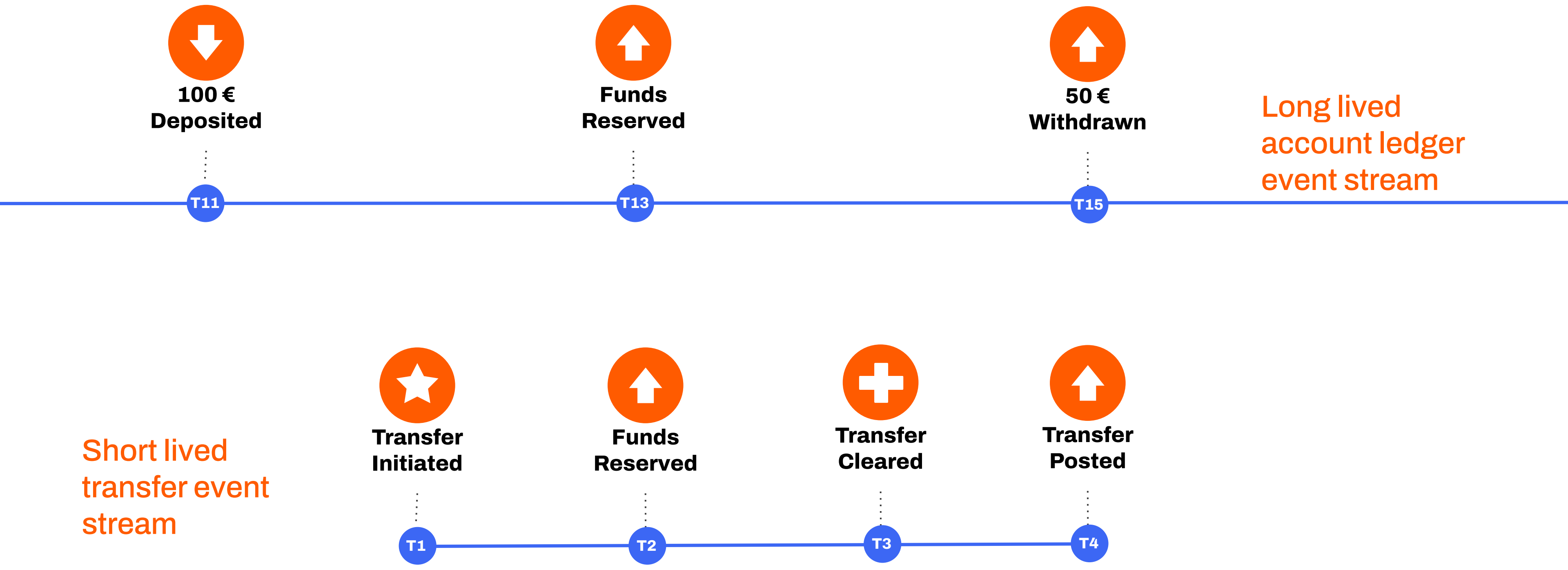


Problems

1. Responsibility
2. Evolving the transfer implementation
3. Idempotency

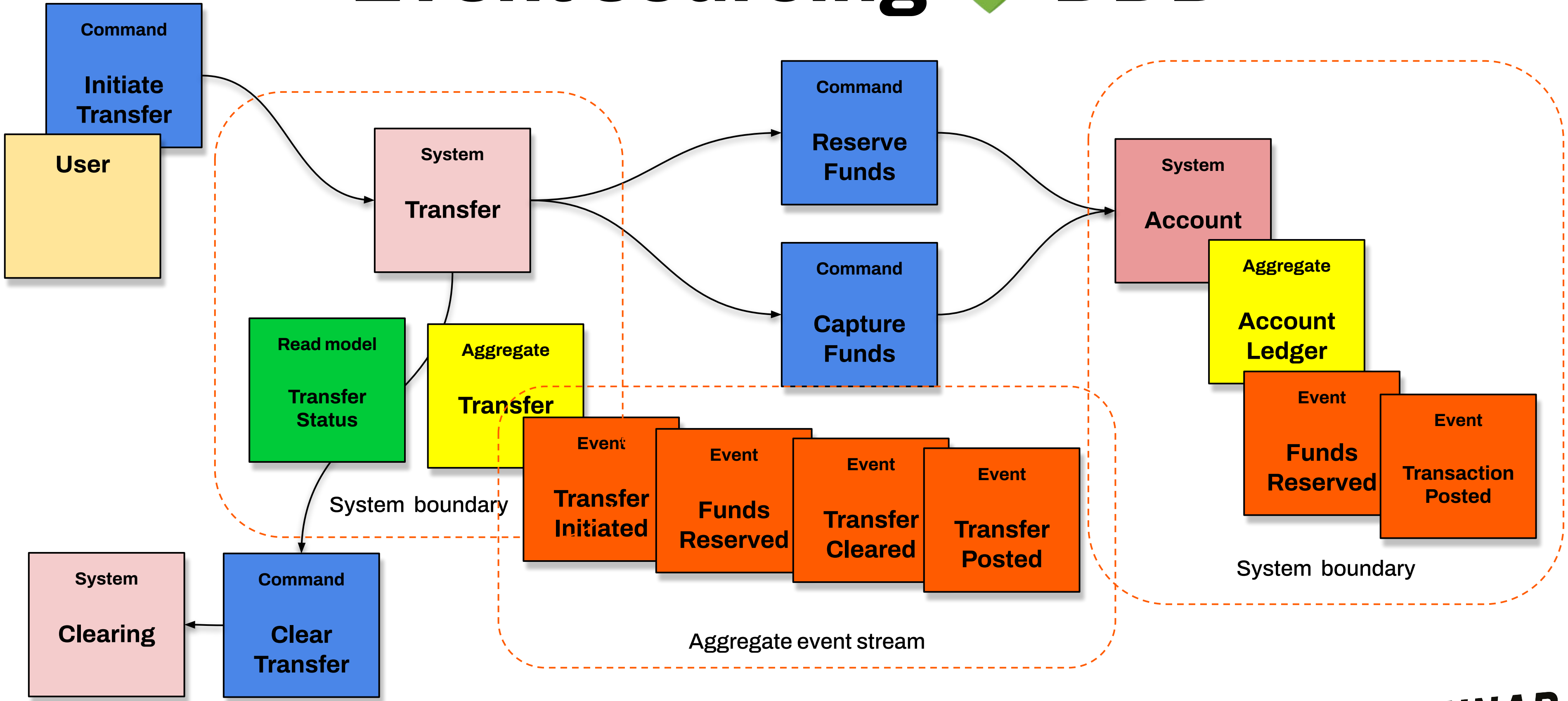
LUNAR

Modeling a transfer



LUNAR

Event sourcing 💚 DDD



LUNAR

Key take aways

Modeling with event sourcing

1. Define event streams around a single responsibility
2. Model single actions/workflows in separate event streams
3. Support for event evolution will eventually be required
4. Embrace DDD
    a. boundaries and stream modeling
    b. event naming

LUNAR

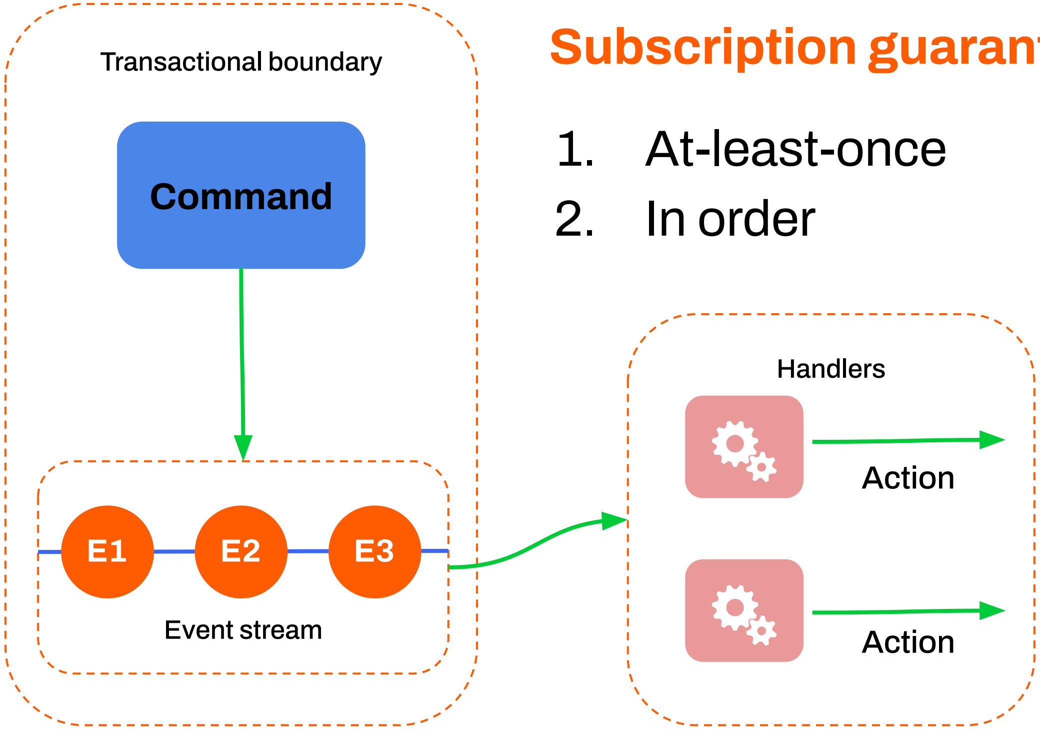
Event subscriptions and side effects

LUNAR

Event subscriptions

Questions

1. What about reads?
2. What about side effects?



Subscription guarantees

1. At-least-once
2. In order

LUNAR

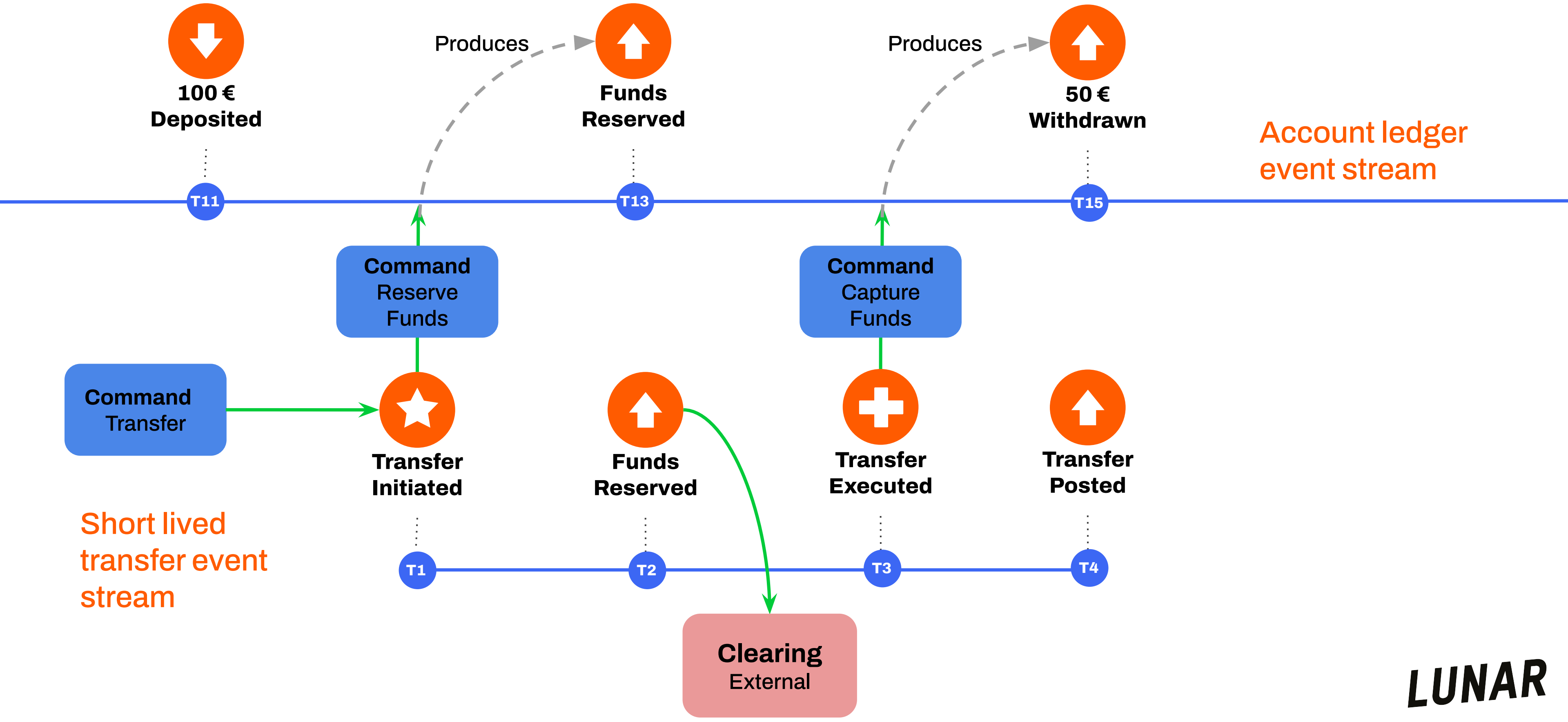
Use cases

Side effects

1. Internal read models
2. External read models - the Query side of CQRS
3. Execute commands - inside and outside domain
4. Publish integration events
5. Process orchestration (sagas)

LUNAR

Orchestrating a transfer



LUNAR

Key take aways

Event subscriptions

1. Must be independent of writing new events
2. Idempotency of actions is really important
3. Embrace eventual consistency
4. Event replay must be supported
5. Difficult to get right - the guaranteed ordering is hard

LUNAR

Event sourcing in a cloud native world

LUNAR

Event sourcing and Cloud Native

Has being Cloud Native made implementing event sourcing easier?

Well, not directly 🤷‍♂️… but it helps

✅ Mind set

✅ Tooling: Backstage, 🚀 Shuttle, K8S, Humio, Prometheus, Grafana

✅ Development speed

LUNAR

Final take aways

Evaluating event sourcing

1. Delivers on the promise of explainability - but not 100%
2. Attractive model - but complexity is higher
3. Don’t use it for everything!

LUNAR