OpenCredo

A visual introduction to Event Sourcing and CQRS

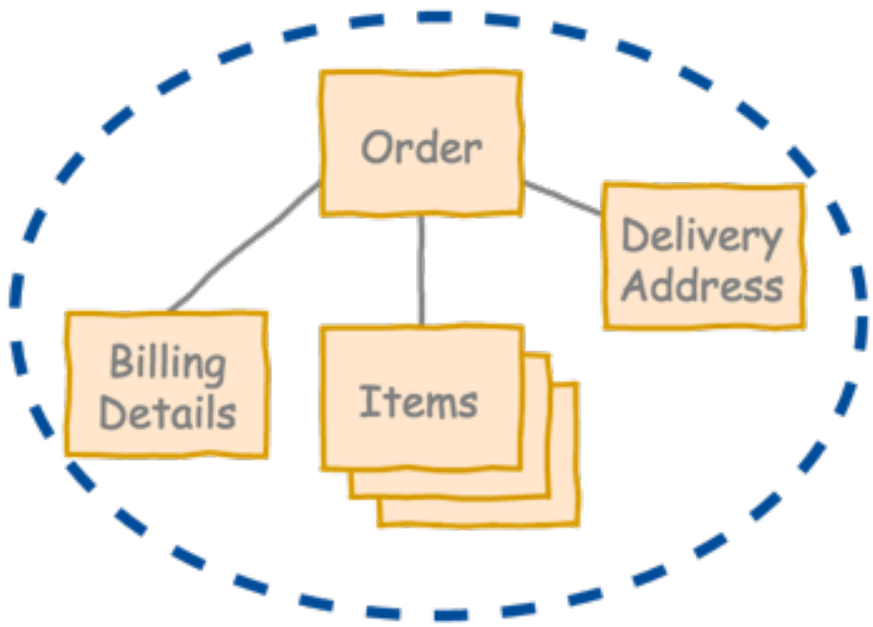
Lorenzo Nicora
Senior Consultant @ OpenCredo

@nicusX
https://opencredo.com/author/lorenzo/

A couple of concepts from DDD

OpenCredo

Aggregate

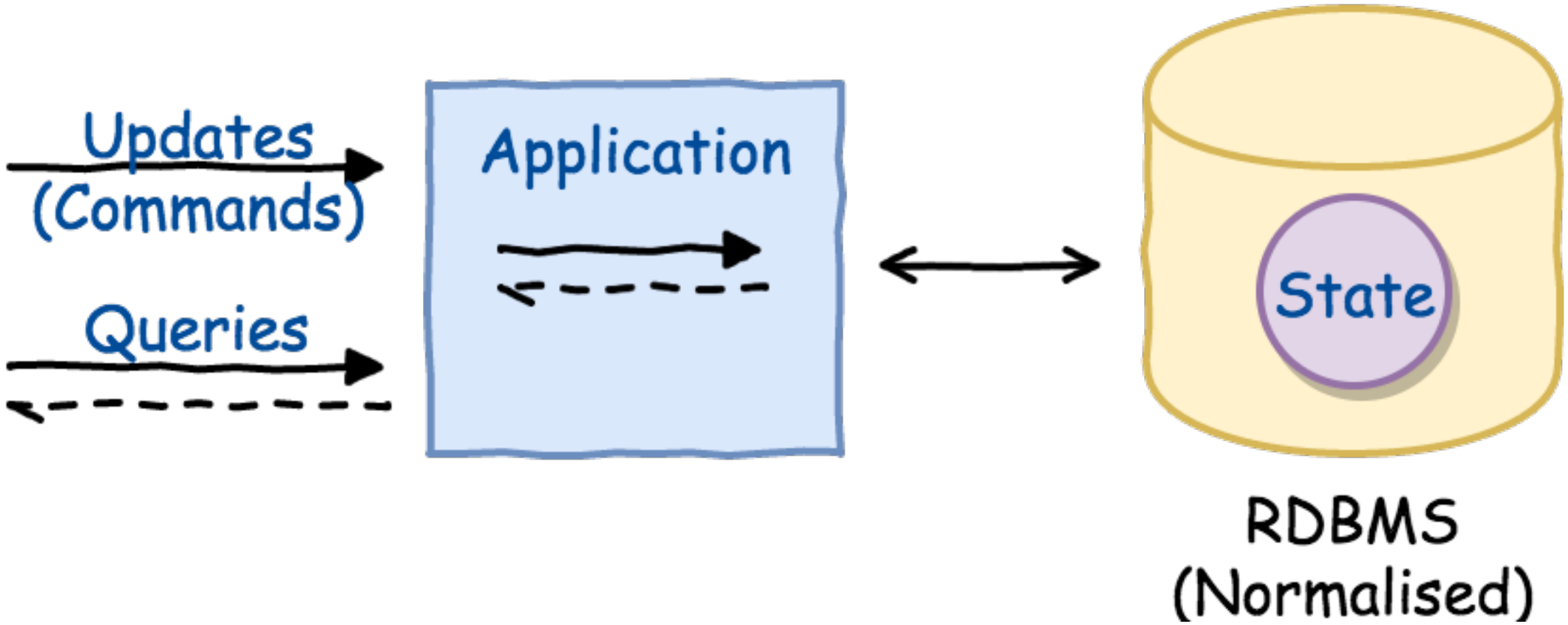


(Current) State of the Aggregate

Once upon a time…

OpenCredo

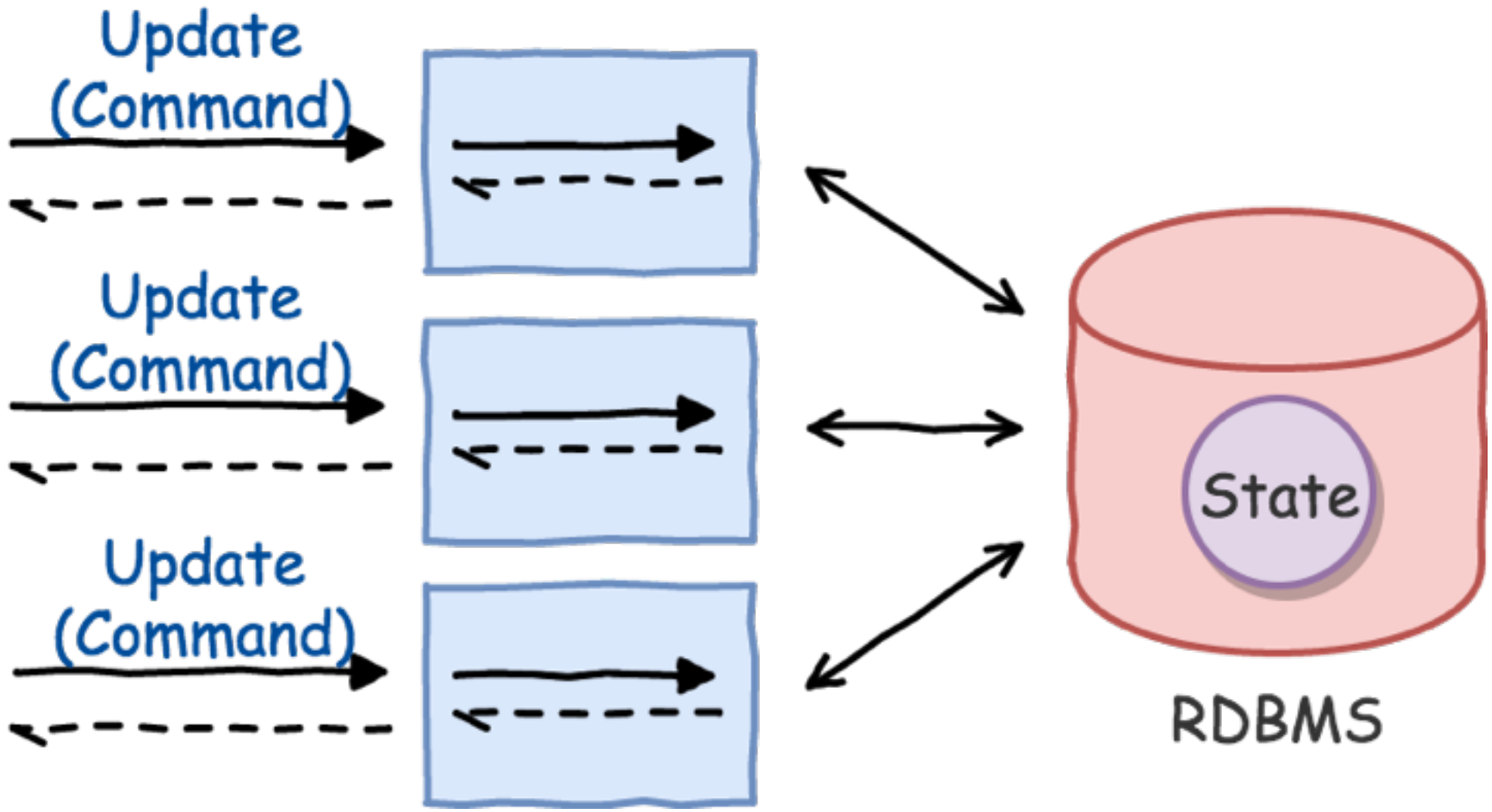
Everything
is synchronous



Request - Response

Scaling up…

OpenCredo



Updates -> Locks -> Contention!

<- Block <-

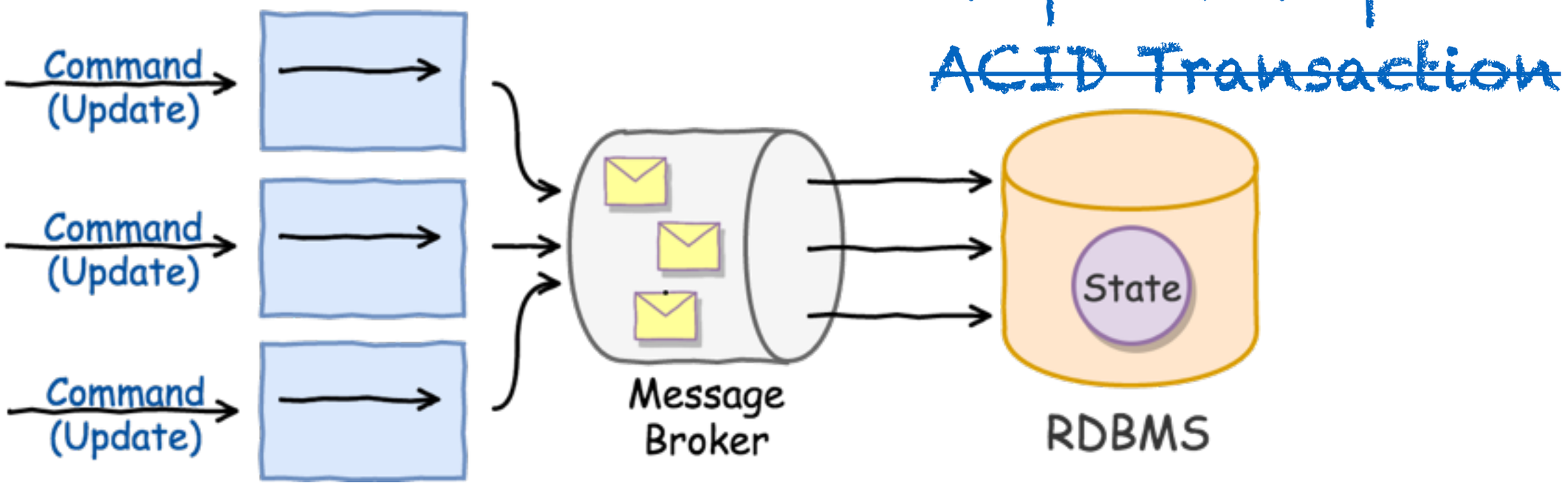
Let's go Asynchronous

OpenCredo

Asynchronous, Message-driven

~~Request/Response~~

~~ACID Transaction~~



Distributed, Message-based

-> No order guaranteed

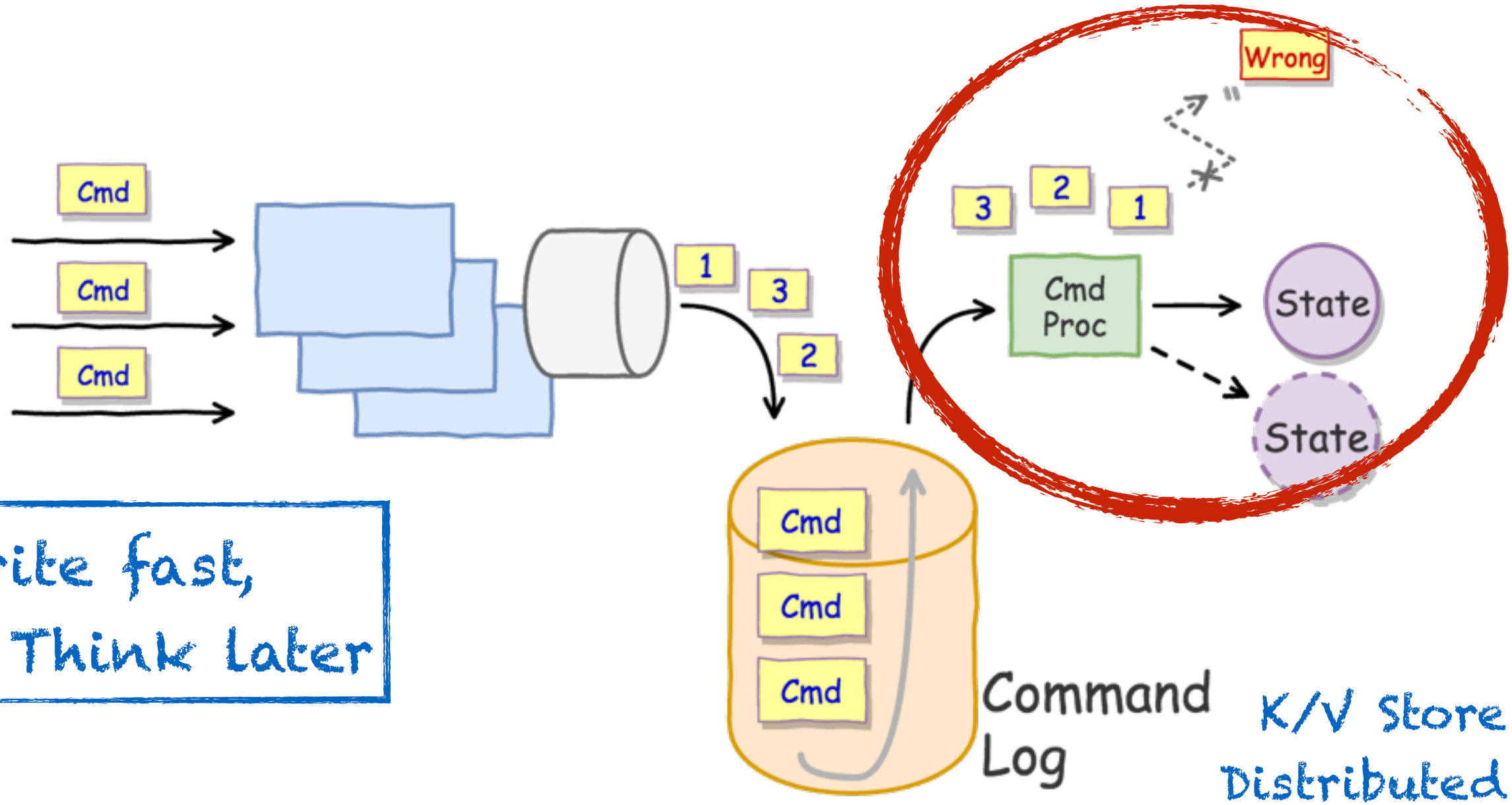
Pwd -> "secret" ===> Pwd -> "secret"

Pwd -> "12345" Out of Pwd -> "54321"

Pwd -> "54321" Order Pwd -> "12345"

Command Sourcing

OpenCredo



Write fast,
Think later

K/V Store
Distributed

- Append Only -> No Contention
- Build State from Command history

Commands vs Events

OpenCredo

Command

"Submit Order!"

-> A request (imperative sentence)

-> May fail

-> May affect multiple Aggregates

~~Rebuild Aggregate State from Commands~~

Event

"Order submitted"

-> Statement of facts (past tense)

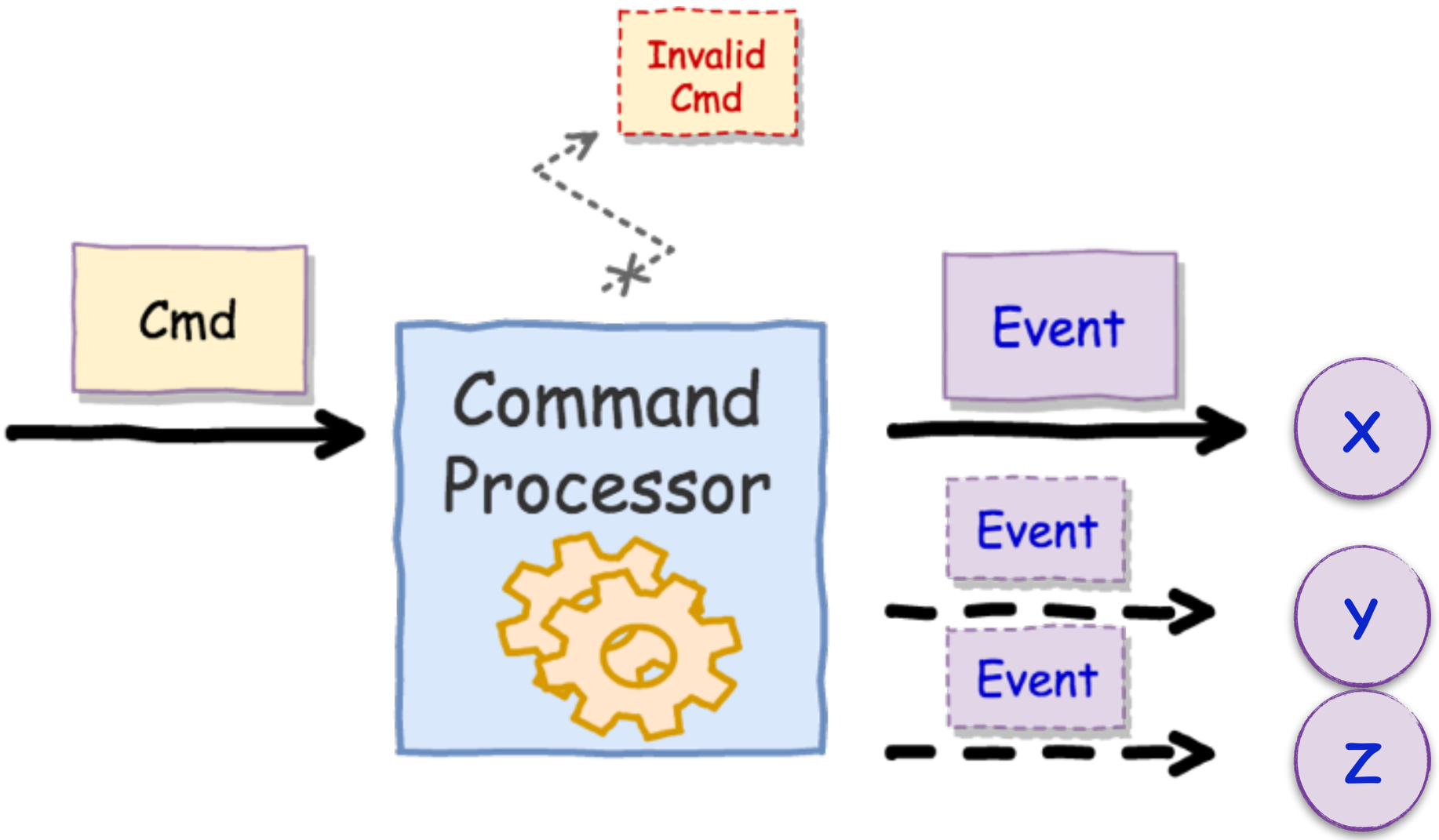
-> Never fails

-> May affect a single Aggregate

Rebuild Aggregate State from Events

Commands to Events

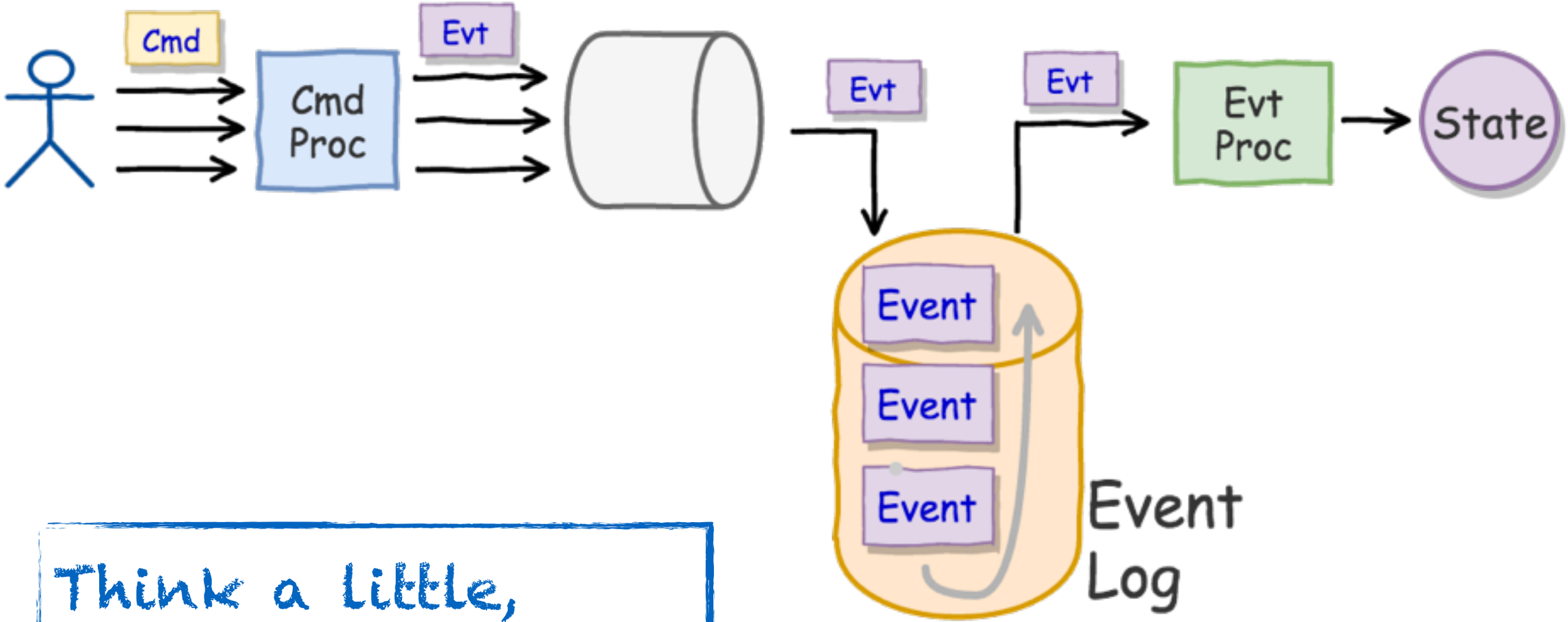
OpenCredo



(DDD patterns: Aggregate / Process Manager)

Command > Event Sourcing

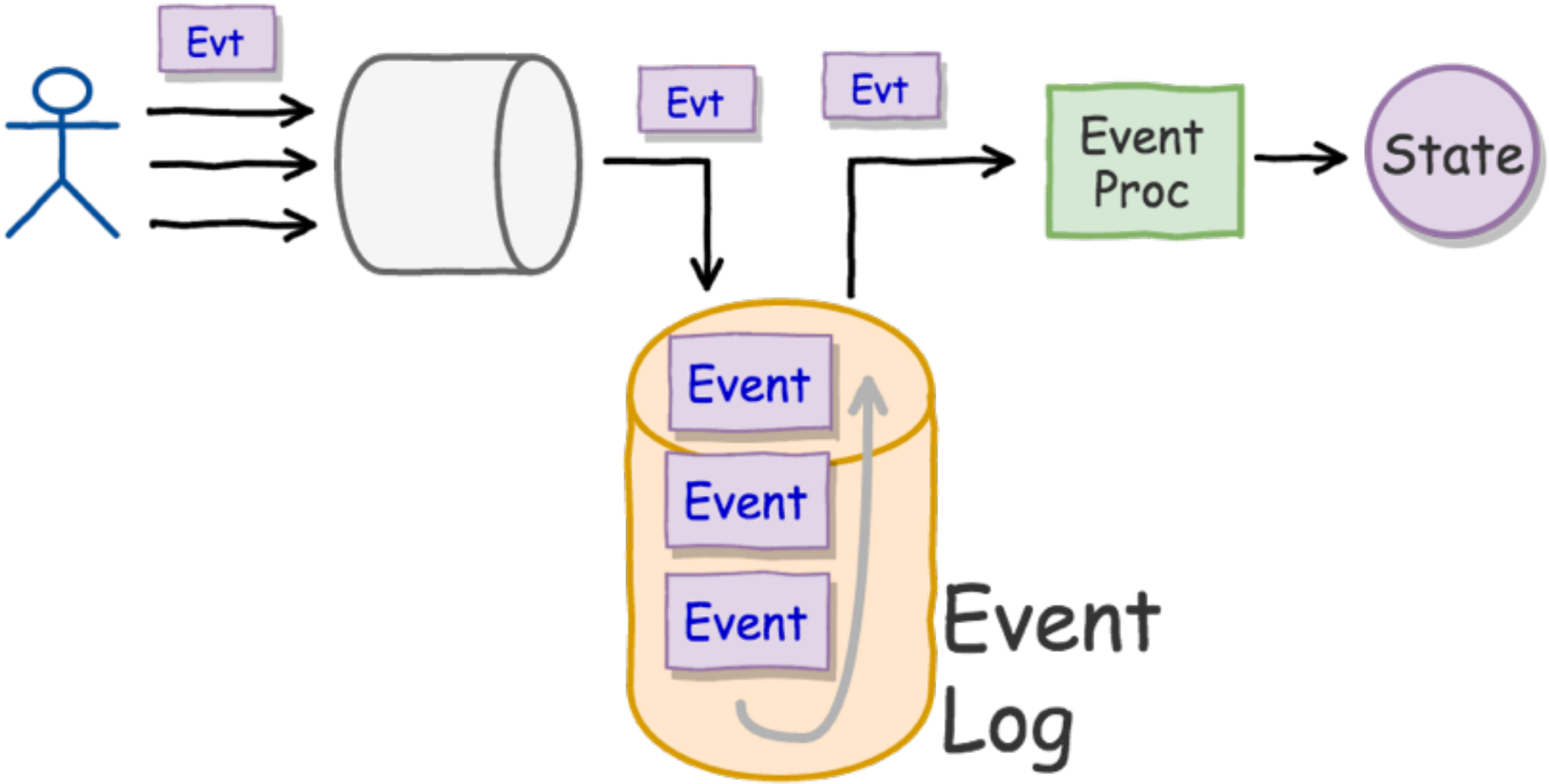
OpenCredo



Think a little,
Write,
Think later

Event Sourcing

OpenCredo



In many domains

Commands ≃ Events

Additional Benefits

OpenCredo

Easy
Eventual Business Consistency

-> Corrective Events

Robust to data corruption

(bugs, fat fingers...)

-> Rebuild state ignoring wrong events

Additional Benefits

OpenCredo

History
(for free)

Rebuild State
at a point in Time

Main Benefit

OpenCredo

Scalable

-> Append only

-> Fits distributed k/v stores

-> Low-latency writes

-> Allows asynchronous processing

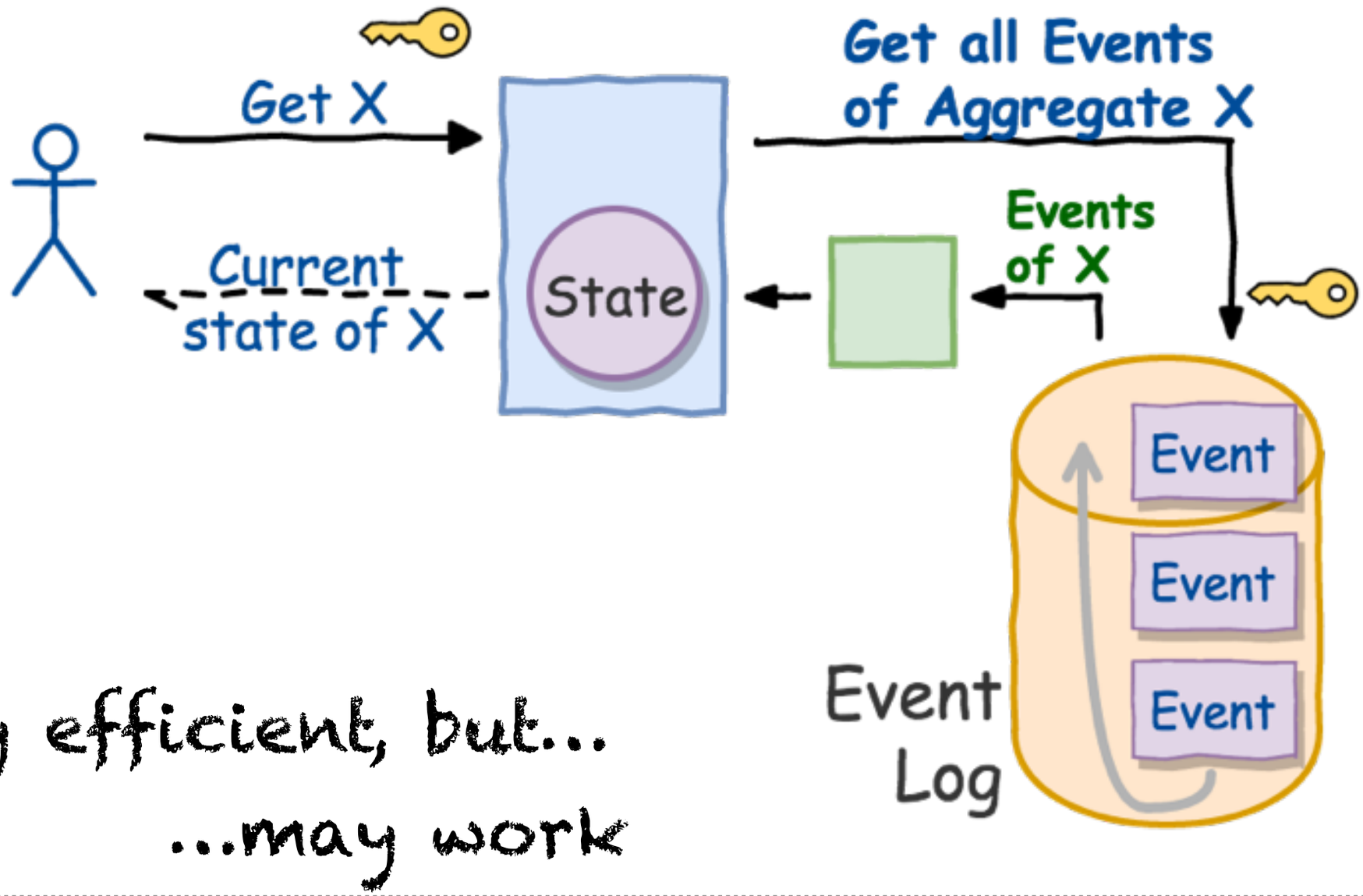
What about reads?

Retrieving the State

OpenCredo

How do I retrieve the State?

"Get details of Order 'AB123'"



not very efficient, but...

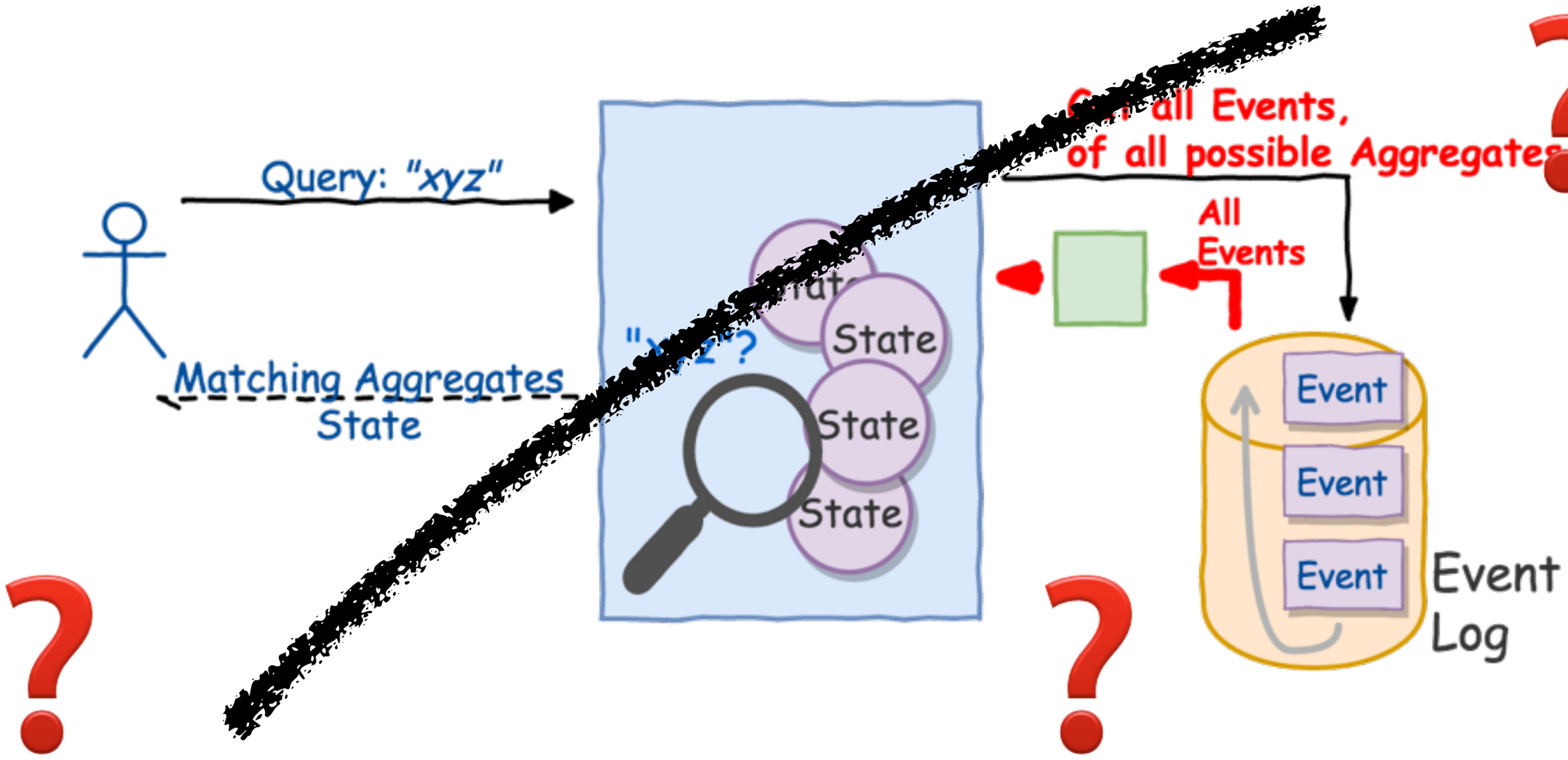
...may work

Querying (Searching) the State

OpenCredo

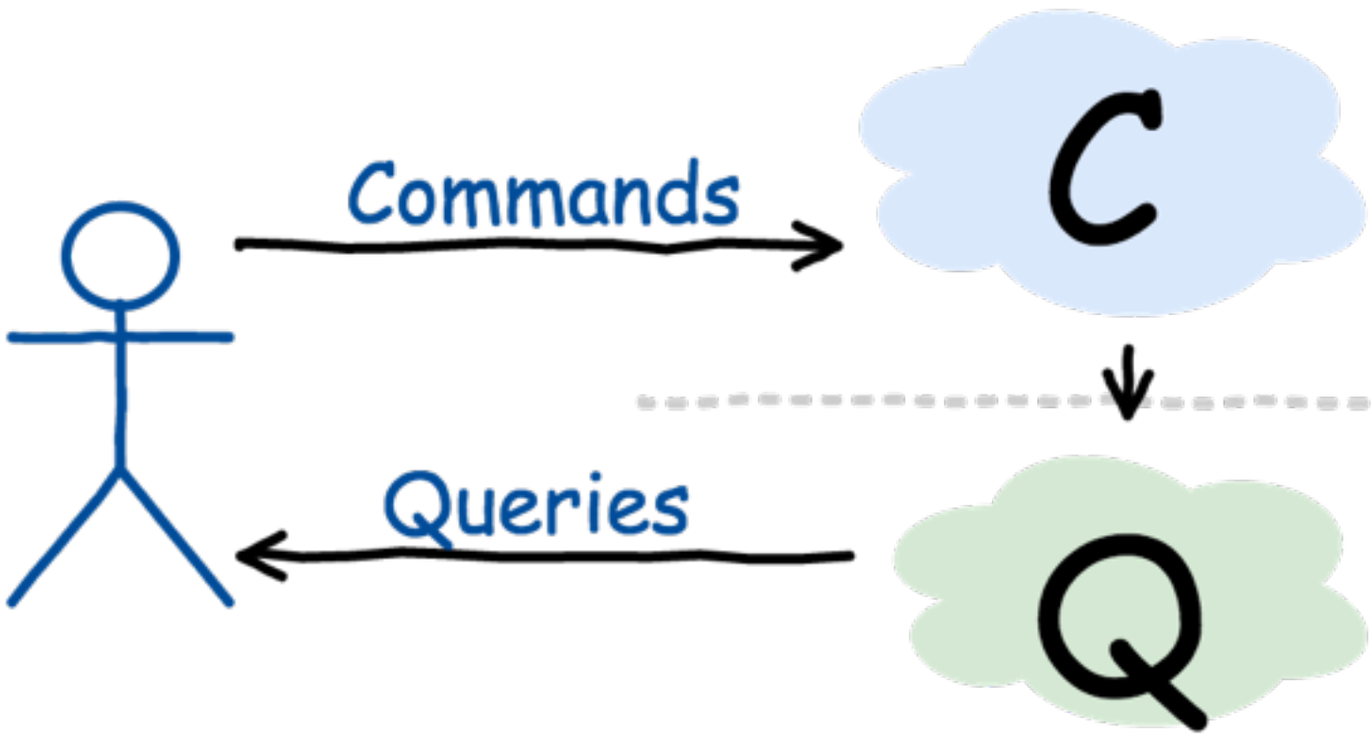
How do query the State?

"Get all Orders delivered to 'SE1 0NZ'"



CQRS

OpenCredo



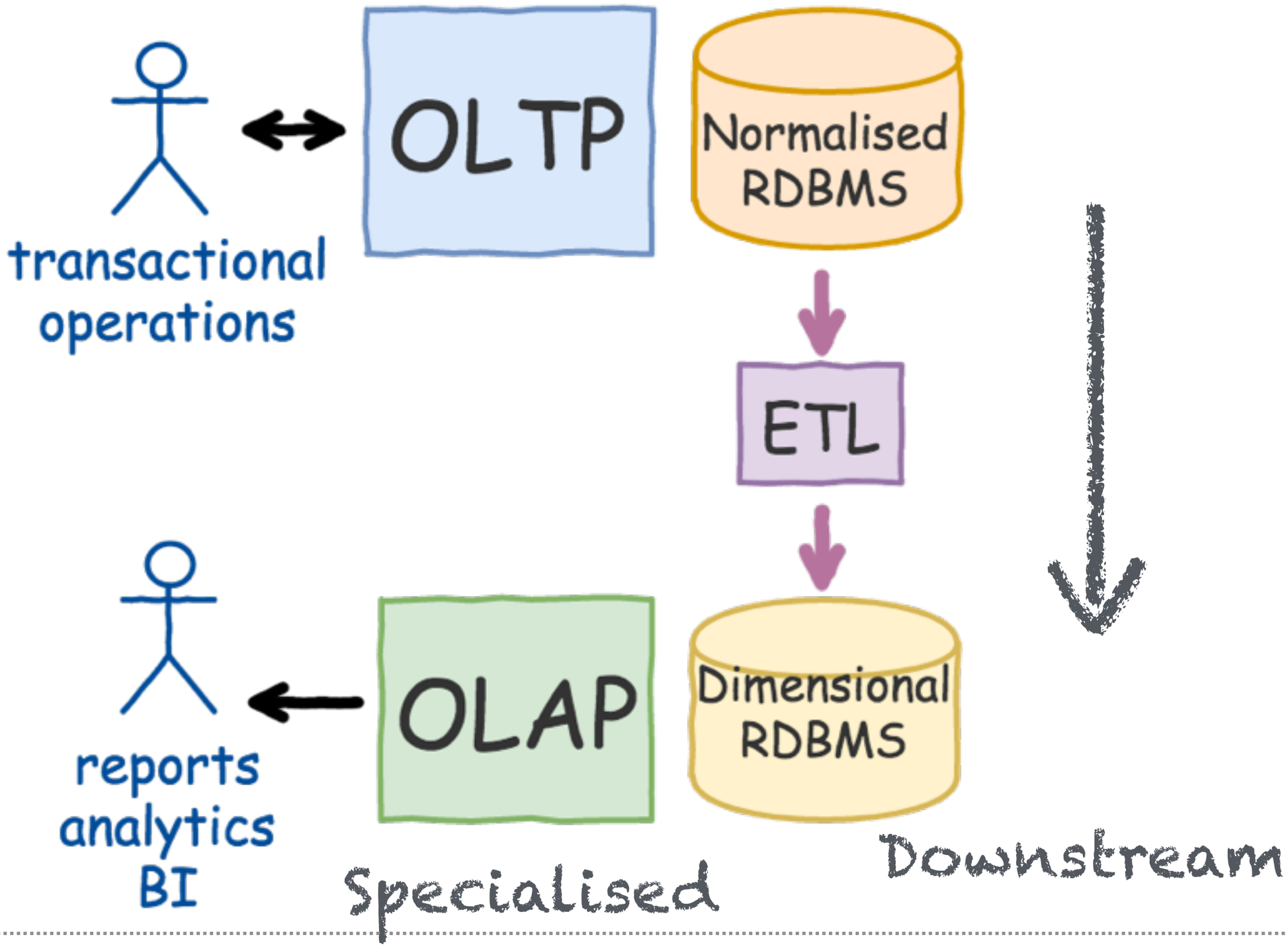
Separate
- Code
- muService
- Datastore

Command -> Update
Query —> Retrieve
Responsibility
Segregation

"Query" datastore is downstream

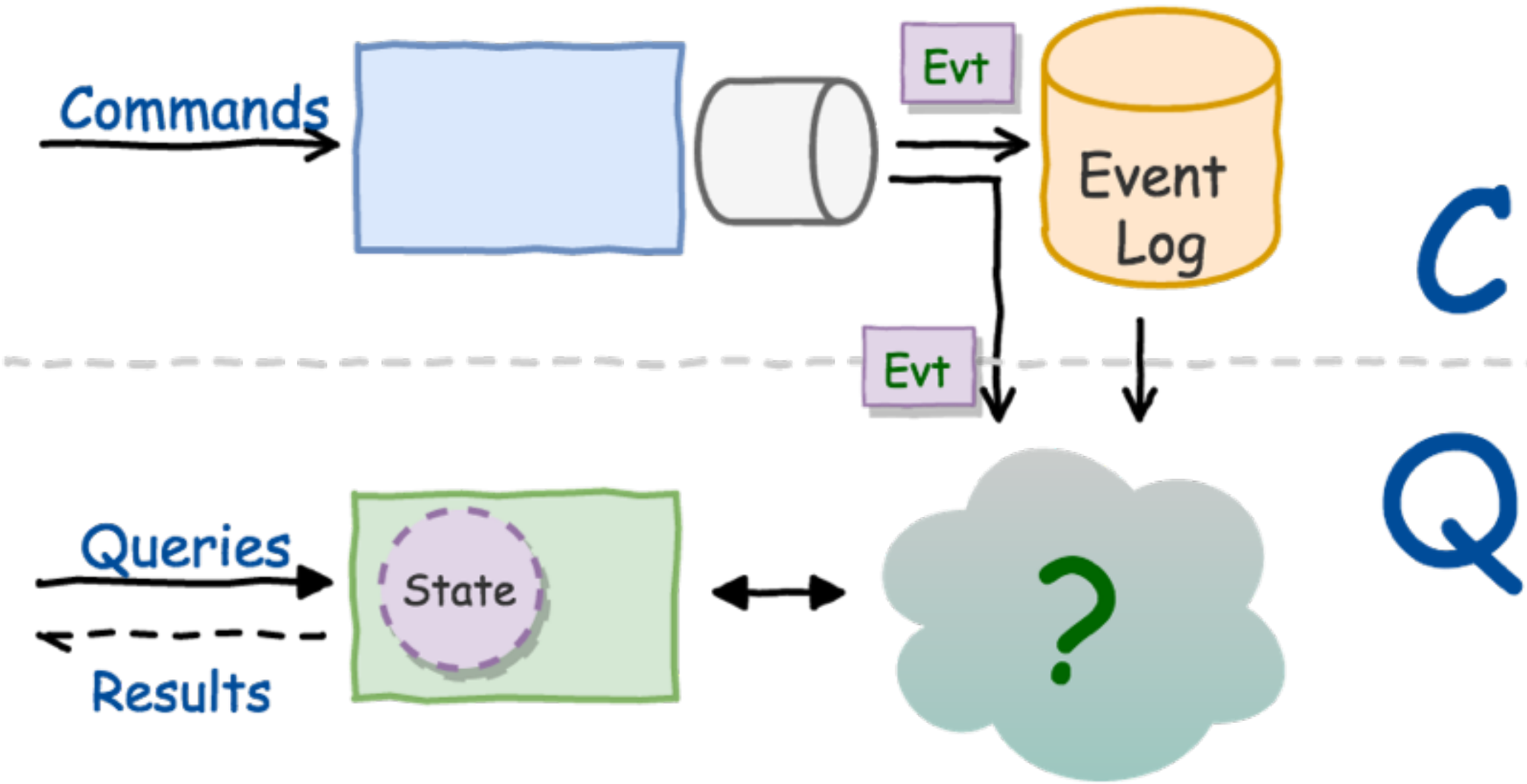
Not a new idea

OpenCredo



CQRS and Event Sourcing

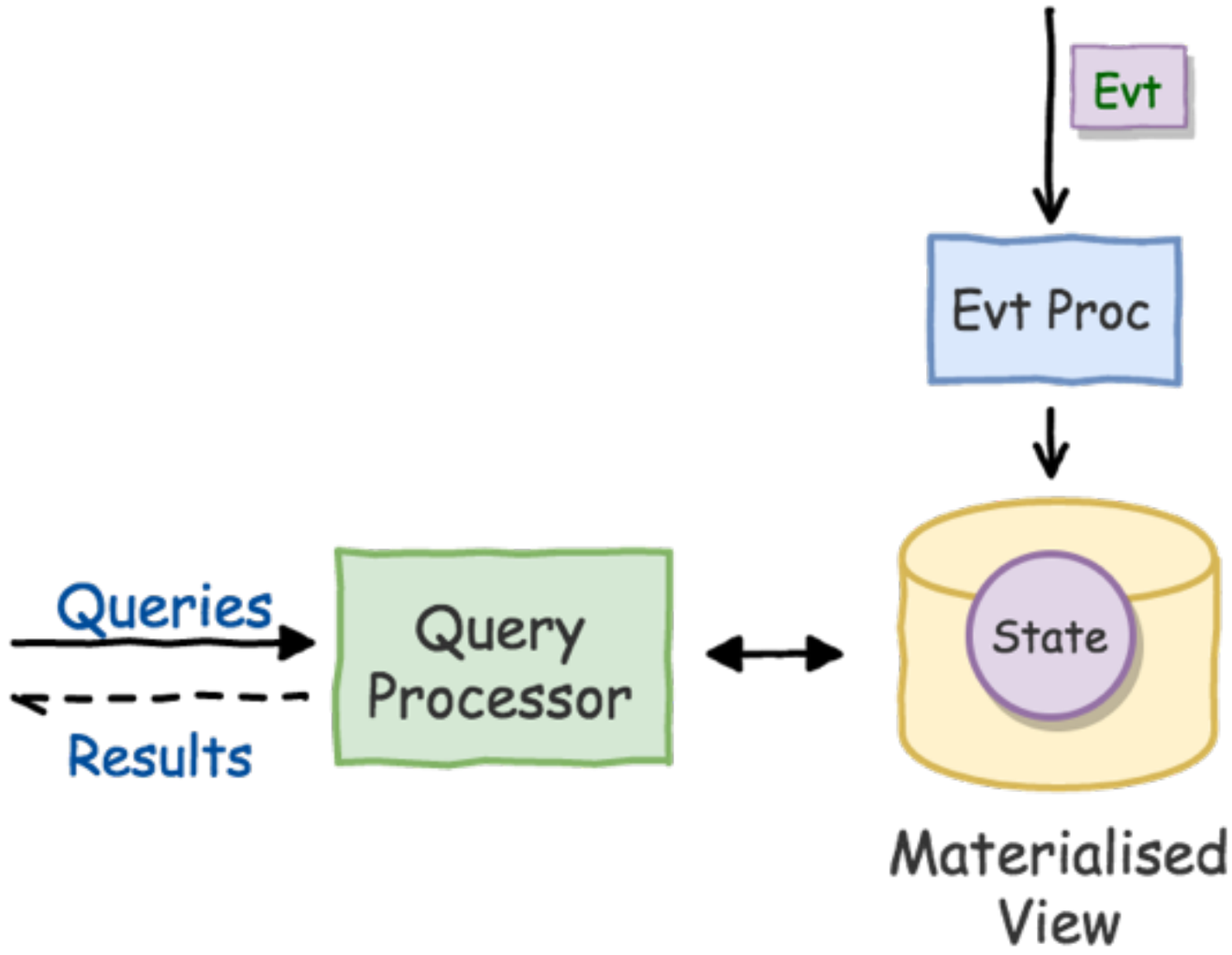
OpenCredo



Materialised Views (of current State)

Materialised Views (of State)

OpenCredo



In Memory
K/V Store
Graph
...
RDBMS

(Persistent)
Rebuildable
from Events

Latest (known) State

Delayed

Materialised View of State

OpenCredo

Query a RDBMS?!?

Wasn't it the old way?

RDBMS is just one of our options:

easy to use, easily become a bottleneck

Materialised Views of State

OpenCredo

* Views are optimised for
  specific query use cases
  -> multiple Views from same Events

* Updated asynchronously, delayed
  -> to scale
  -> may reorder Events

Materialised Views of State

OpenCredo

* Easy to evolve or fix

  -> change or fix logic;
     rebuild view from events

Event Log
is our Source of Truth

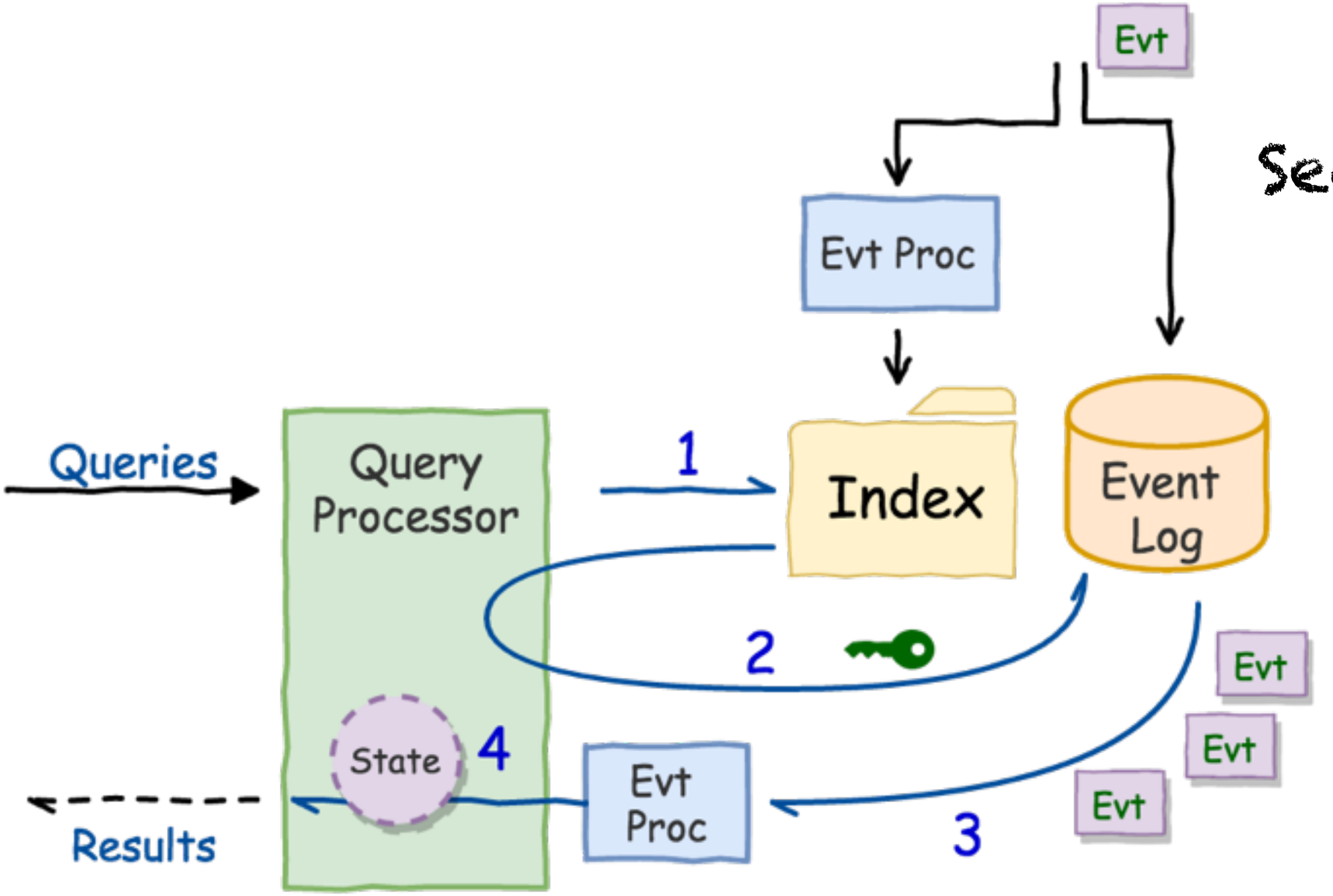
(not the View)

* Views can be rebuilt from Events

Indexes

Indexes

OpenCredo

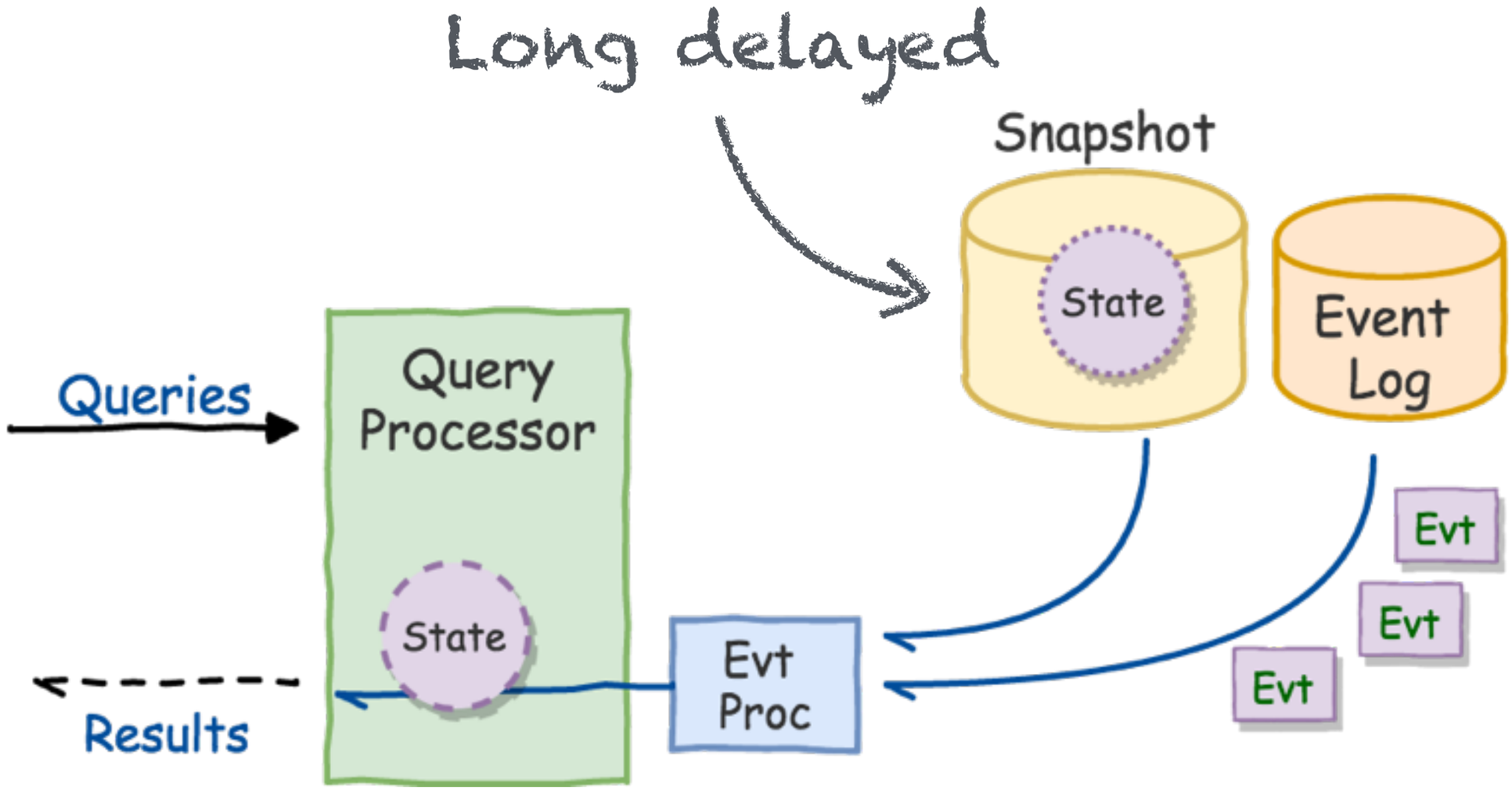


- Optimised for querying (less for retrieving)
- Latest State; rebuild on the fly

Hybrid solutions

Hybrid solutions: e.g. Snapshots

OpenCredo

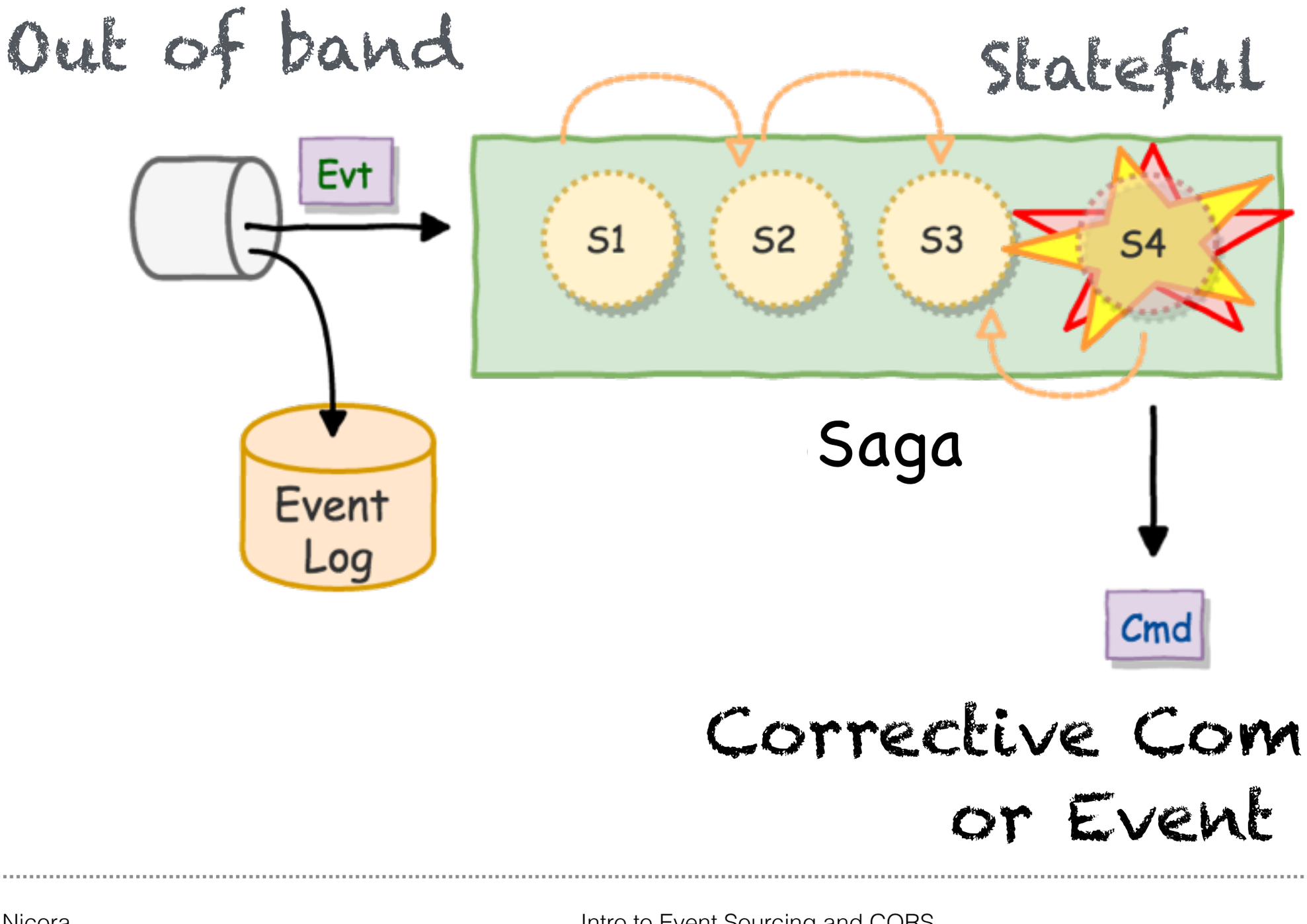


- Speed up rebuilding the current State
- Use recent Events to rebuild up-to-date

Eventual (Business) Consistency

Guess -> Apologies -> Compensate

Eventual Consistency: Sagas



Lesson from the Trenches

Lesson Learned #1

OpenCredo

If you put data in...

...you will eventually have to get them out!

The "Query" side is not secondary

Lessons Learned #2

OpenCredo

In old days:

normalising one DB

to support as many queries as possible

With CQRS:

multiple denormalised "data stores"

optimised for different queries

No single "Q" implementation

for all your queries

Lessons Learned #3

OpenCredo

A central, shared Event Store
may not be the best option

No Event-sourced Monolith

Prefer persistence
per Bounded-Context

+++ Summing up +++

ES/CQRS Optimal Use Cases

OpenCredo

Event Sourcing + CQRS

High Volume

Low Latency writes

(big data)

Out-of-order Commands/Events

(IoT)

ES/CQRS Drawbacks

OpenCredo

x No "One-Size-Fits-ALL"

-> Multiple "Q" implementations

x Delayed reads

x No ACID Transactions

x Additional complexity (!)

ES/CQRS Benefits

OpenCredo

+ No "One-Size-Fits-ALL"

  -> "Q" are optimised for use cases

+ Eventual (Business) Consistency

+ History, Temporal queries

+ Robust to data corruption

That's all, Folks!

??? Questions ???

Thanks.