O'REILLY®

Software Architecture

ENGINEERING THE FUTURE OF SOFTWARE

From CRUD to Event Sourcing an Investible Stock Universe

softwarearchitecturecon.com

#oreillysacon

O'Reilly Software Architecture Conference March 18, 2015 #oreillysacon

TIM Group

Marc Siegel
Team Lead (@ms_yd)

Brian Roberts
Senior Developer / Team Lead (@flicken)

What's in it for you?

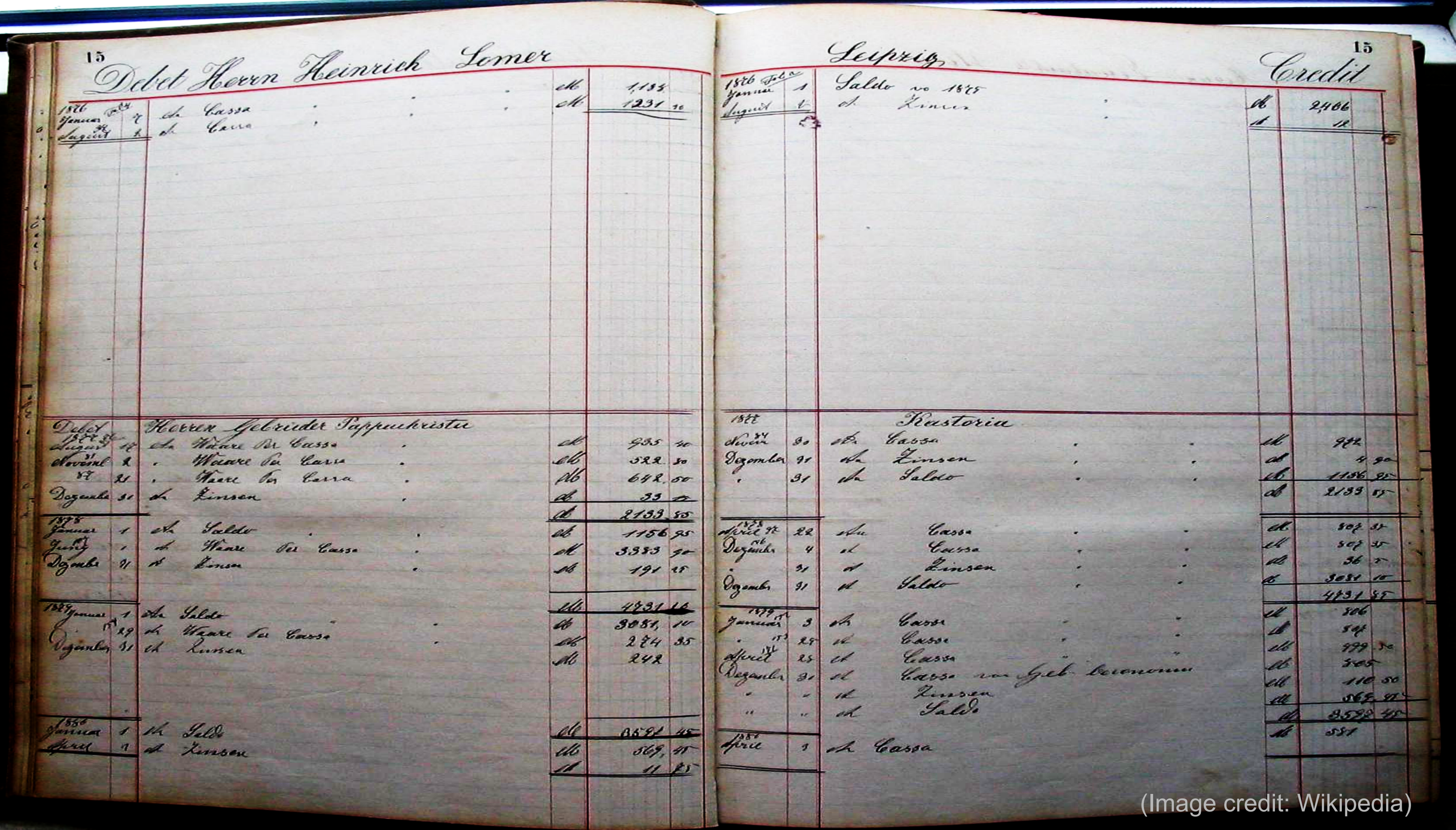
- Answer the Unanswerable
    - both now and future

Vocabulary

1. Domain-Driven Design (DDD)
   - Event
   - Command
   - Aggregate

2. Event Sourcing (ES)
   - Projection
   - Read Model

How does Event Sourcing Work?



15 Debet Herrn Heinrich Lomer

1876 Fol. 27					
Januar	7	An Cassa	M	1,133	
August	2	An Cassa	M	1231	10

Leipzig Credit 15

1876 Fol. 2					
Januar	1	Saldo vo 1875	M	2466	
August	2	An Zinsen	M	12	

Debet Herren Gebrüder Pappachristu

1877 Fol. 34					
August	18	An Waare Per Cassa	M	935	40
Novemb 31	2	" Waare Per Cassa	M	522	30
87	21	" Waare Per Cassa	M	642	50
Dezember	31	An Zinsen	M	33	—
			M	2133	85
1878					
Januar	1	An Saldo	M	1156	95
Juny	1	An Waare Per Cassa	M	3383	90
Dezember	31	An Zinsen	M	191	25
			M	4731	10
1879 Januar	1	An Saldo	M	3081,	10
15	29	An Waare Per Cassa	M	274	35
Dezember	31	An Zinsen	M	242	
1880					
Januar	1	An Saldo	M	3597	45
April	7	An Zinsen	M	569,	45
			M	11	85

Kastoria

1877					
Novemb 34	30	An Cassa	M	972	
Dezember	31	An Zinsen	M	4	90
"	31	An Saldo	M	1156	95
			M	2133	85
1878					
April 97	22	An Cassa	M	507	35
Dezember 106	4	An Cassa	M	507	35
"	31	An Zinsen	M	36	5
Dezember	31	An Saldo	M	3081	10
				4731	85
1879					
Januar 151	3	An Cassa	M	306	
" 153	25	An Cassa	M	507	
April 171	25	An Cassa	M	999	50
Dezember	31	An Cassa von Geb. [illegible]	M	505	
"	"	An Zinsen	M	110	50
"	"	An Saldo	M	569	97
			M	3597	45
1880					
April	3	An Cassa	M	551	

(Image credit: Wikipedia)

How does Event Sourcing Work?

(Quotes from Greg Young)

"State transitions are an important part of our problem space and should be modeled within our domain"

How does Event Sourcing Work?

(Quotes from Greg Young)

"State transitions are an important part of our problem space and should be modeled within our domain"

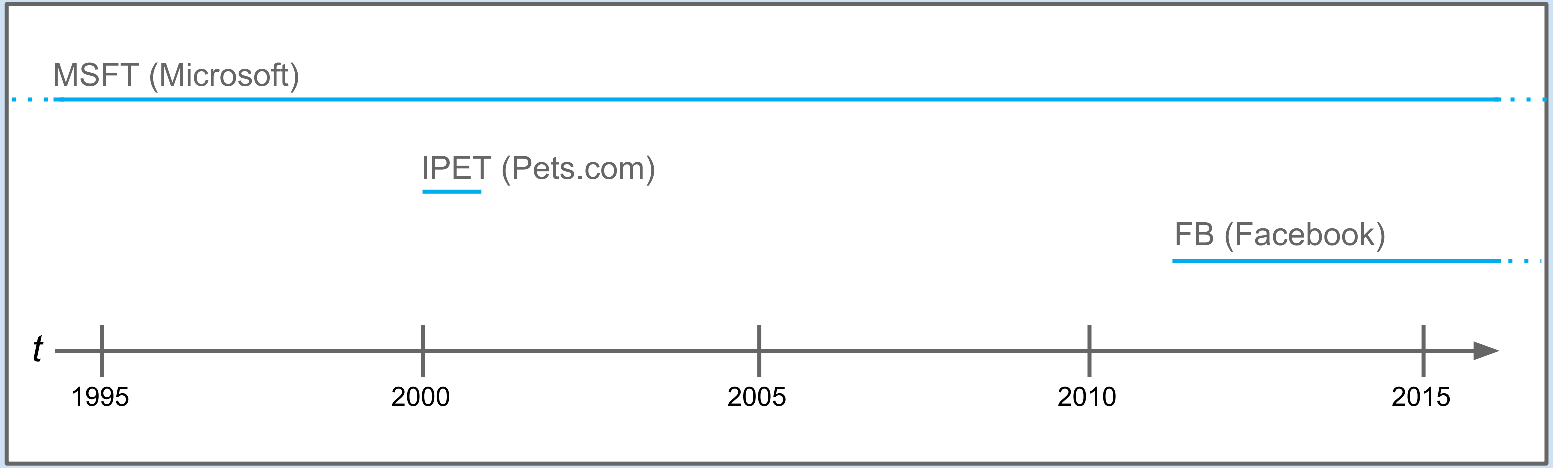
"Event Sourcing says all state is transient and you only store facts."

Vocabulary

1. Domain-Driven Design (DDD)
   - **Event:** *something that happened in the past; a fact; a state transition*
   - Command
   - Aggregate

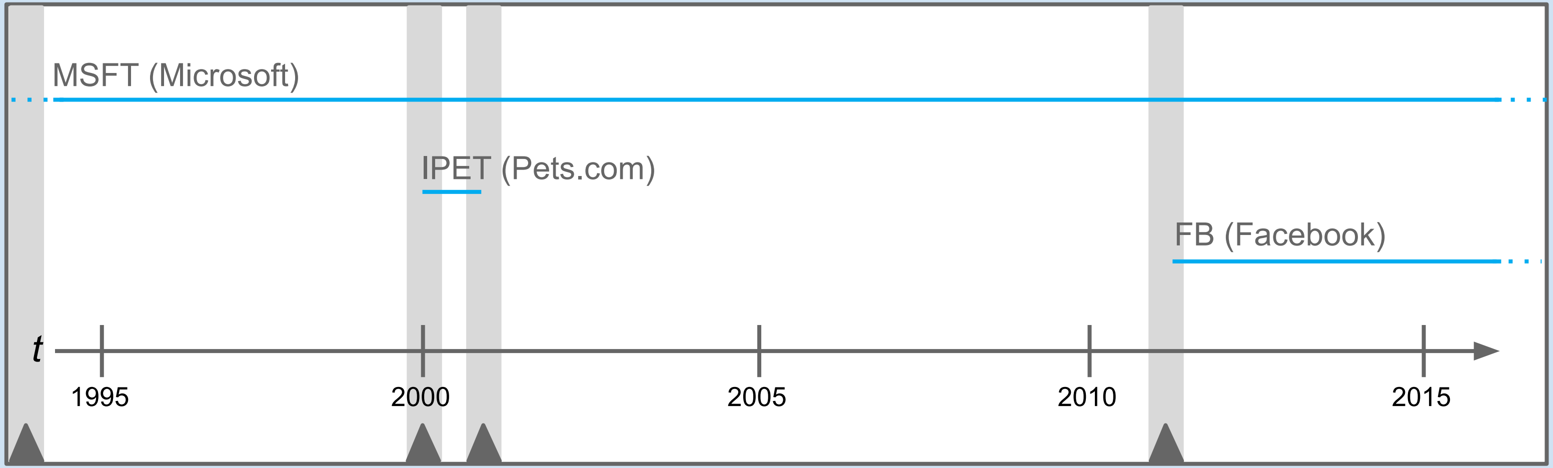
2. Event Sourcing (ES)
   - Projection
   - Read Model

Lifecycle of a Listing (Simplified)



Q: How to represent the meaningful state transitions of the domain as events?

Listing Lifecycle Events



Events:

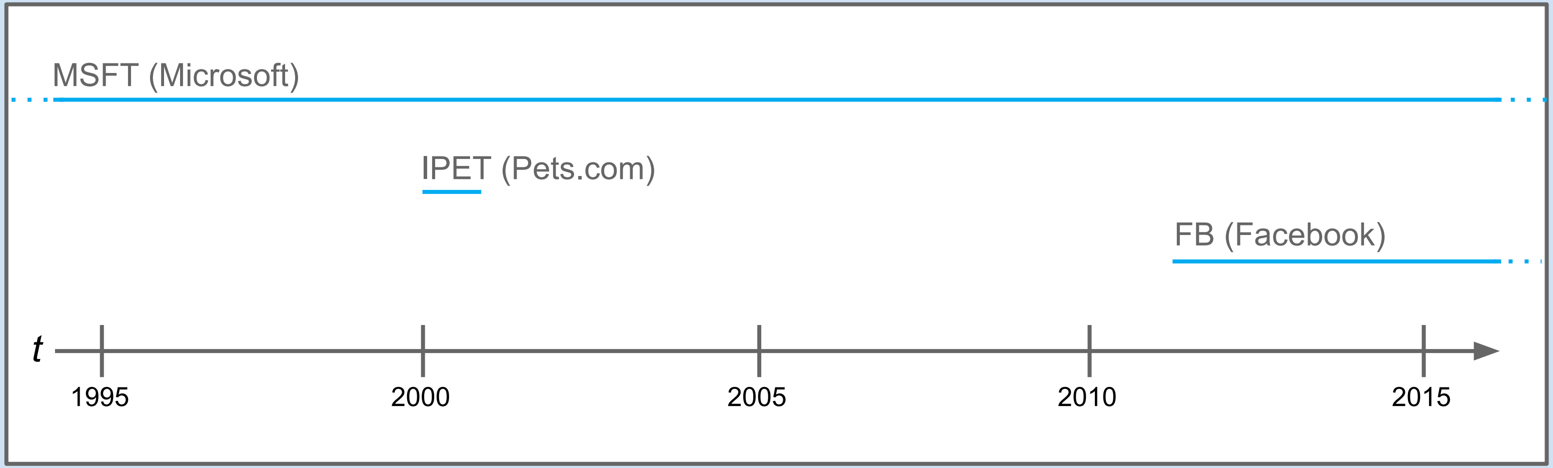
TickerListed
ticker: MSFT
date: 1986

TickerListed
ticker: IPET
date: 2000

TickerDelisted
ticker: IPET
date: 2000

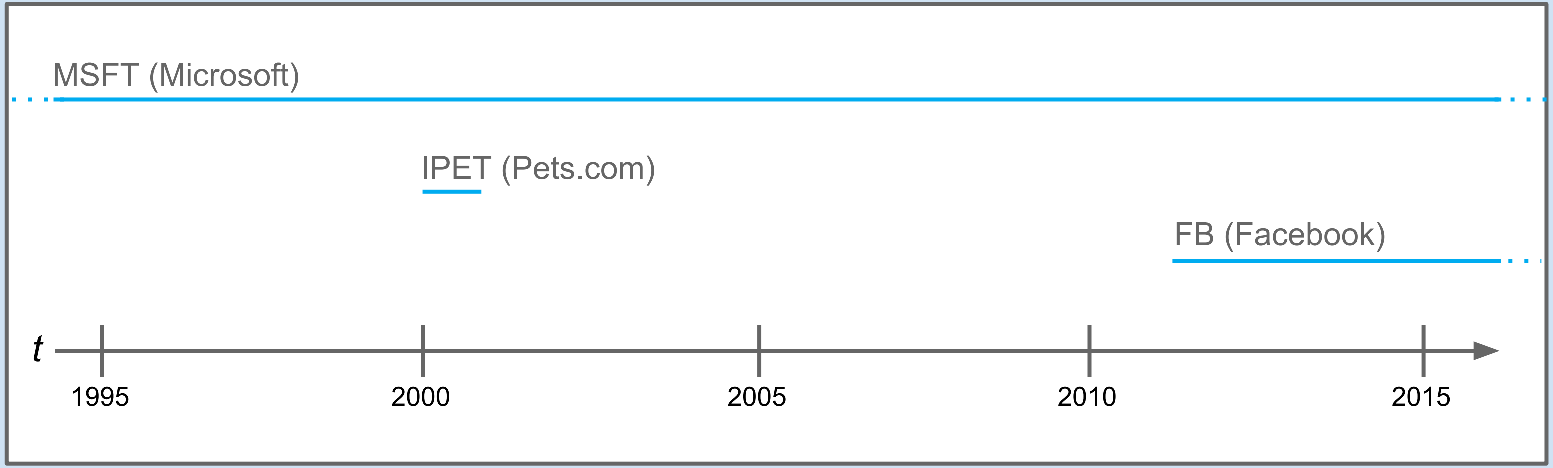
TickerListed
ticker: FB
date: 2012

Determine Current State



Q: How do we determine the state as of a given year? Or as of today?

Determine Current State



Q: How do we determine the state as of a given year? Or as of today?

A: "When we talk about Event Sourcing, current state is a left-fold of previous behaviors" -- Greg Young

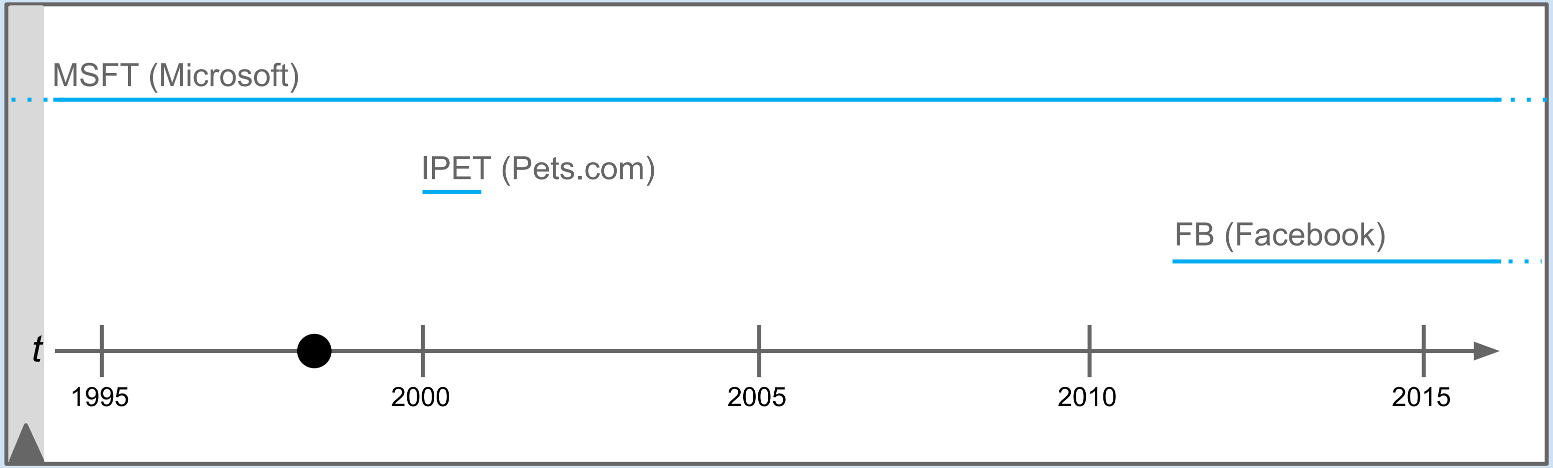
Current State is a Left Fold of Events λ

- FP: Left Fold aggregates a collection via a function and an initial value
  - Ruby: `[1, 2, 3].inject(0, :+)            ==    6    # symbol fn name`
  - Scala: `List(1, 2, 3).foldLeft(0)(_ + _)    ==    6    // anon function`

- Provide an initial state s_0 and a function f : (S, E) => S

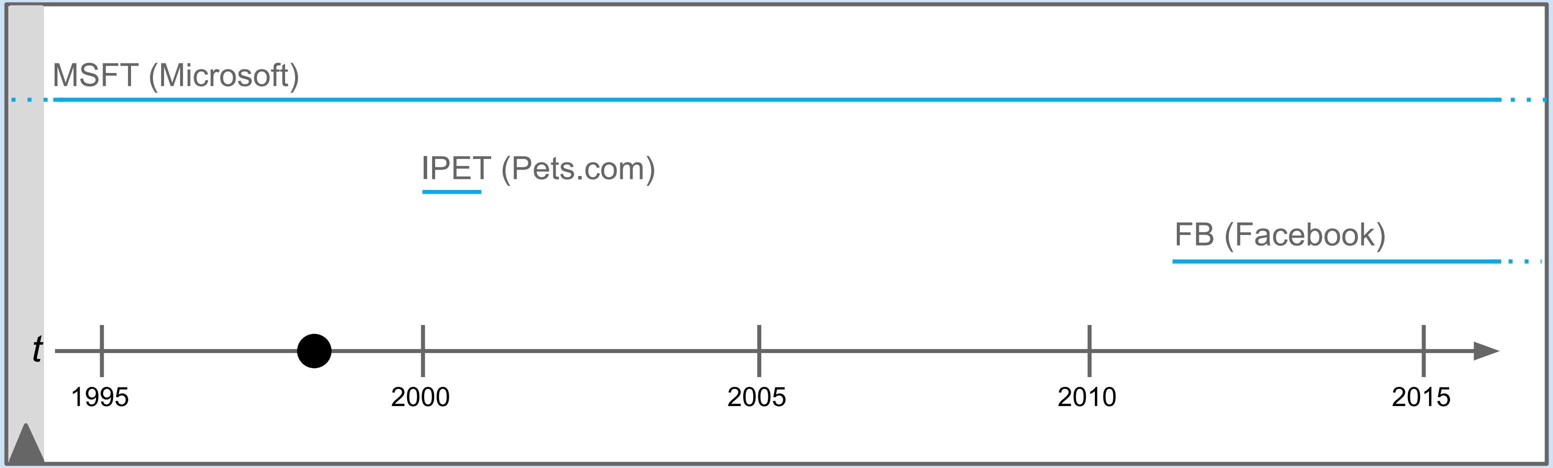
- Current State after event e_3 is:
  - $= \mathtt{leftFold}(\ [e_1,\ e_2,\ e_3],\ s_0,\ f\ )$
  - $= \mathtt{f}(\ \mathtt{f}(\ \mathtt{f}(\ s_0,\ e_1),\ e_2),\ e_3)$

Replay to Earlier State: 1998



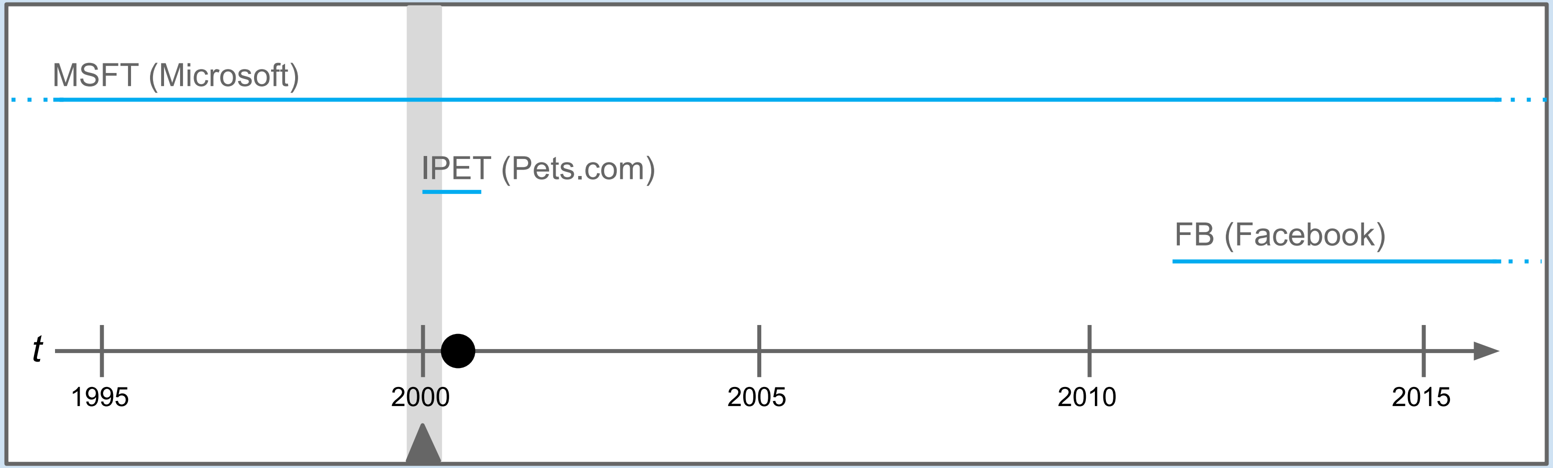
State in 1998 = [ ] **.apply(** **TickerListed** ticker: MSFT date: 1986 **)**

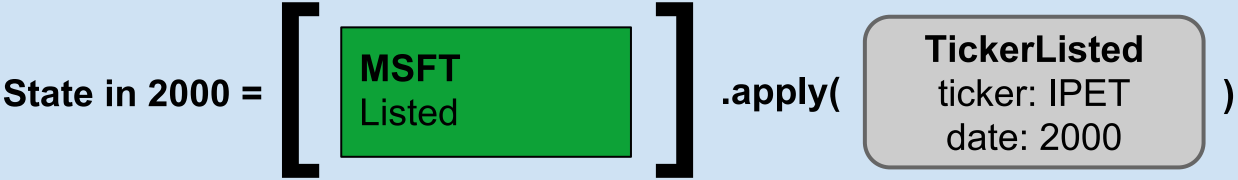
Replay to Earlier State: 1998



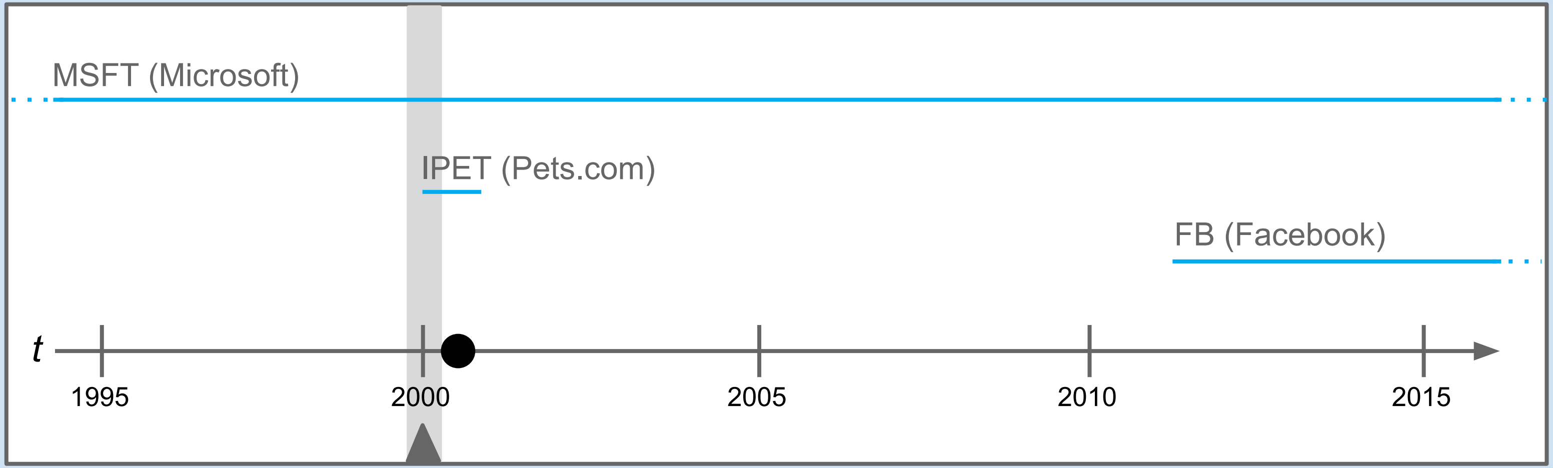
State in 1998 = [**MSFT** Listed]

Replay to Earlier State: 2000



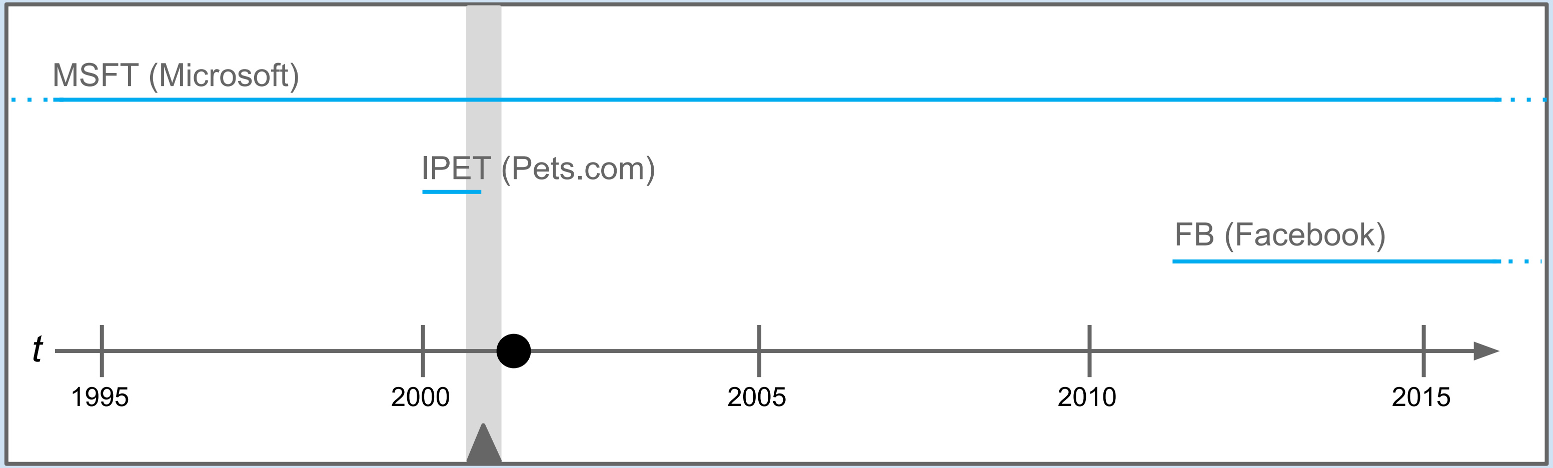

State in 2000 = [MSFT Listed] .apply(TickerListed ticker: IPET date: 2000)

Replay to Earlier State: 2000



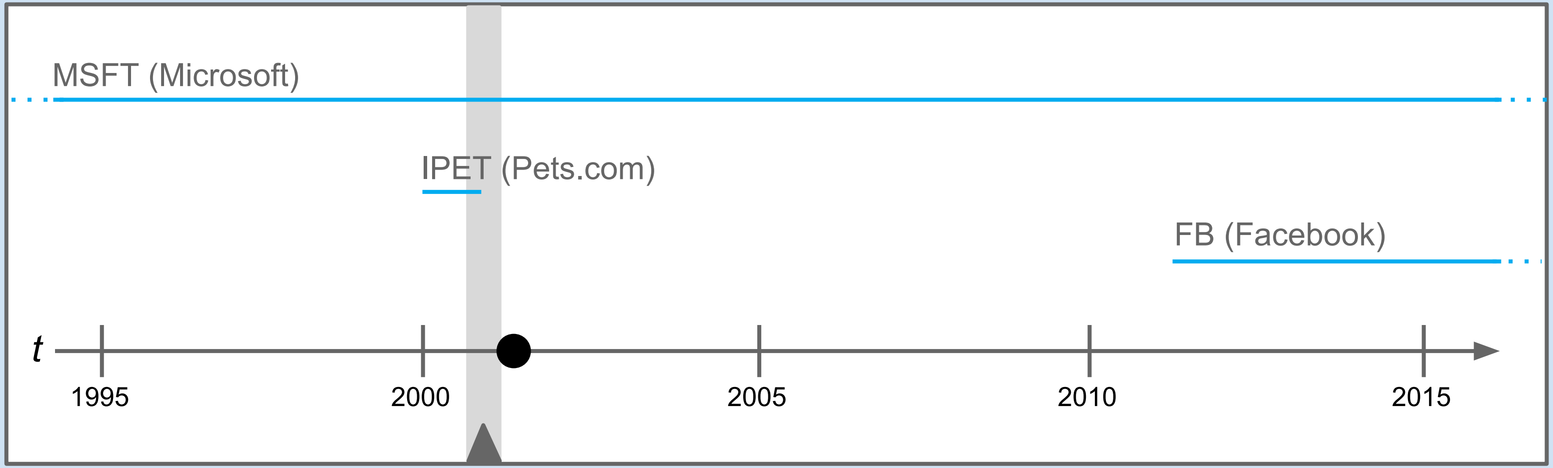
State in 2000 = [**MSFT** Listed | **IPET** Listed]

Replay to Earlier State: 2001



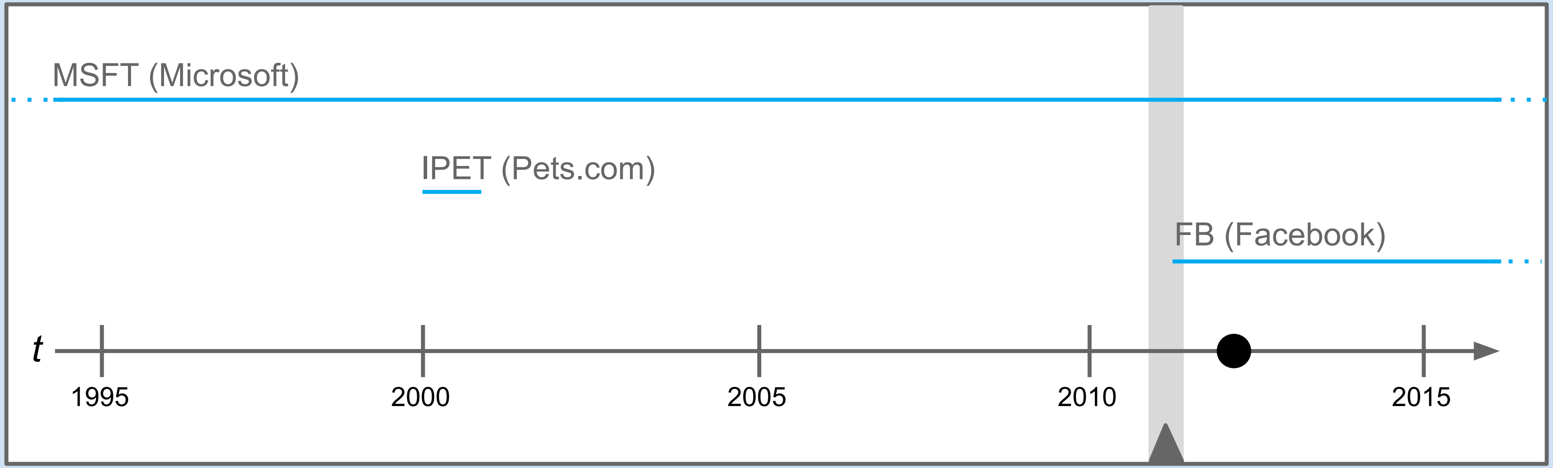
State in 2001 = [**MSFT** Listed | **IPET** Listed] **.apply(** **TickerDelisted** ticker: IPET date: 2000)

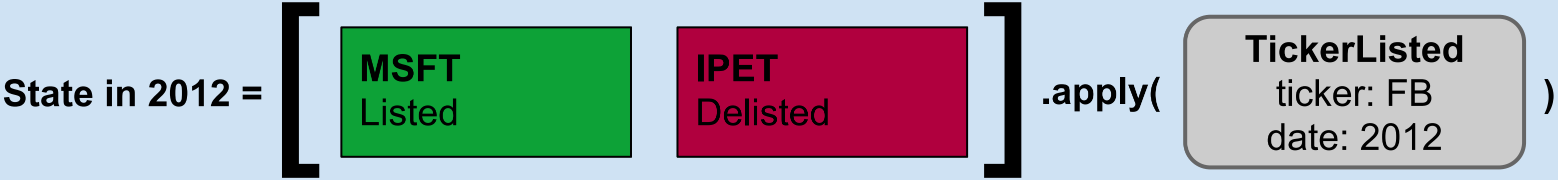
Replay to Earlier State: 2001



State in 2001 = [**MSFT** Listed | **IPET** Delisted]

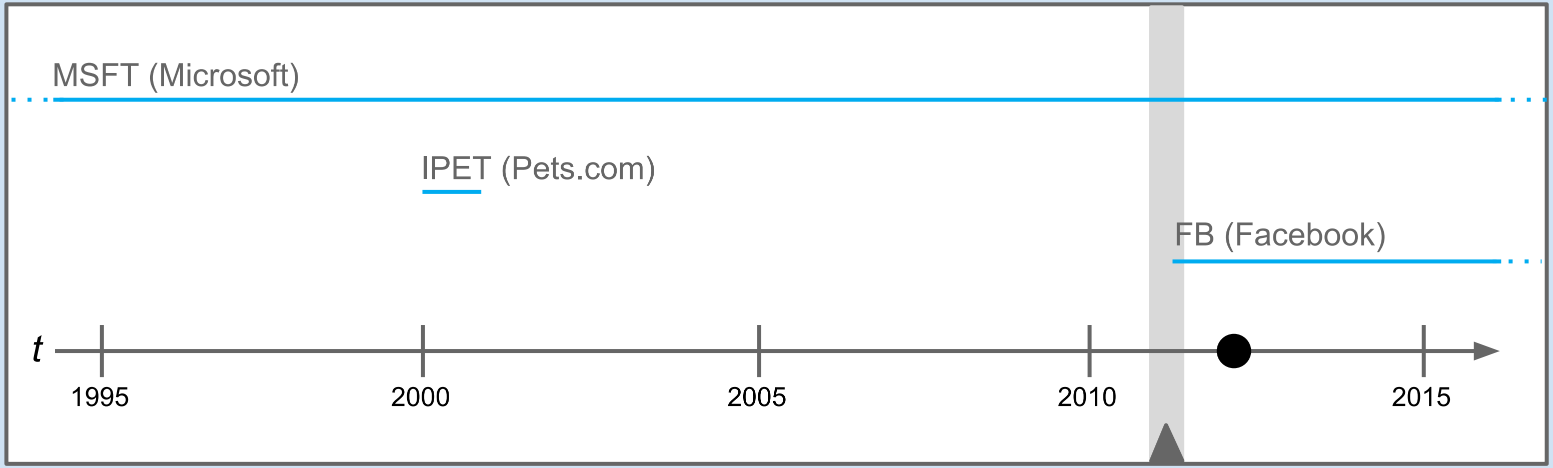
Replay to Earlier State: 2012





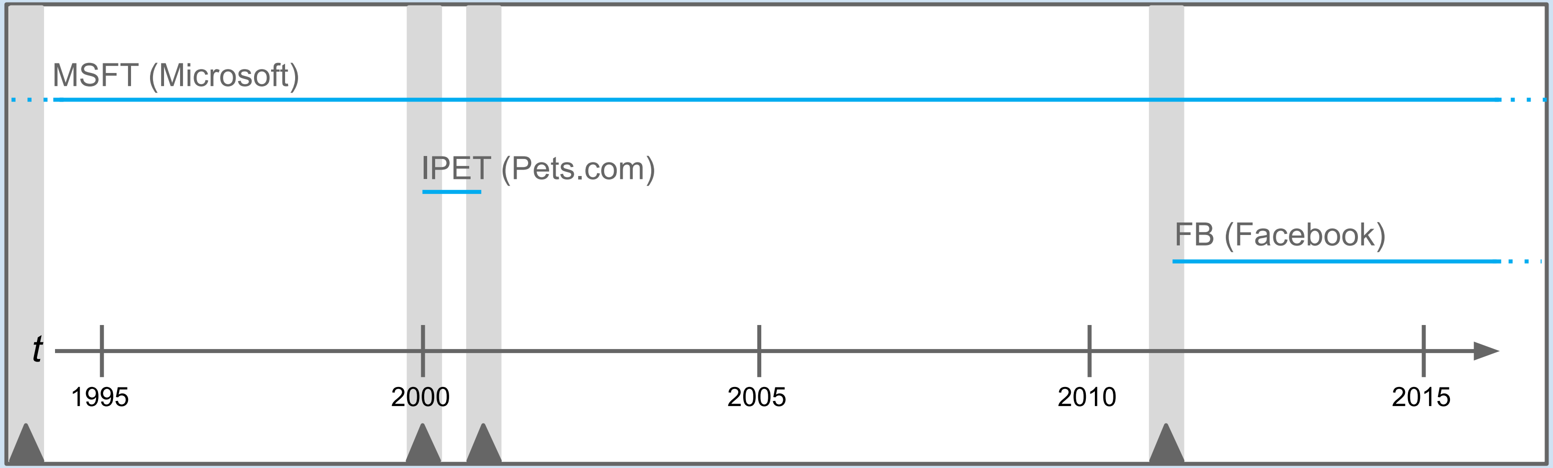
State in 2012 = [**MSFT** Listed | **IPET** Delisted] .apply(**TickerListed** ticker: FB date: 2012)

Replay to Earlier State: 2012

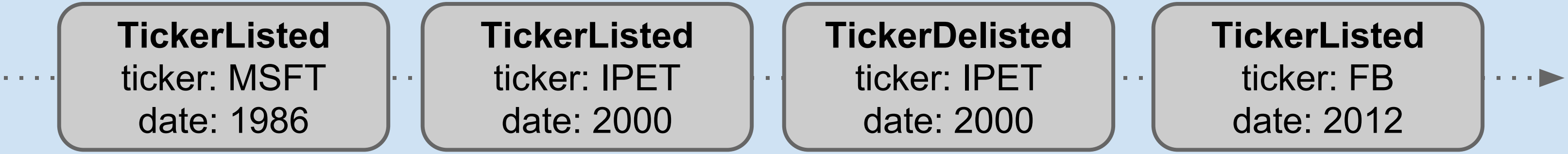


State in 2012 = [**MSFT** Listed | **IPET** Delisted | **FB** Listed]

Review - Only the Events are Stored



Events:



TickerListed
ticker: MSFT
date: 1986

TickerListed
ticker: IPET
date: 2000

TickerDelisted
ticker: IPET
date: 2000

TickerListed
ticker: FB
date: 2012

Potential Benefits

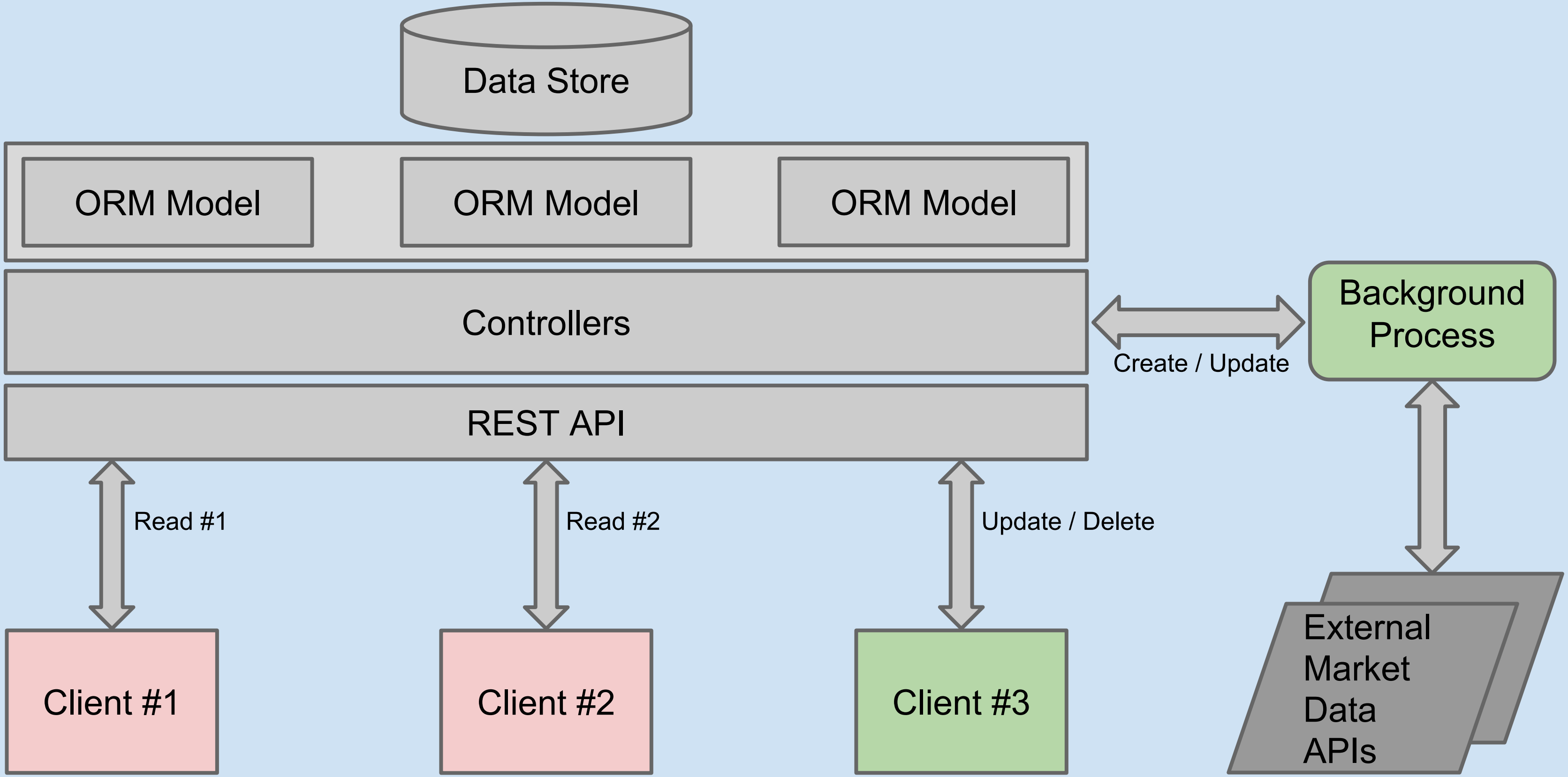
- Answer the unanswerable (via history replay)
- Debugging of historical states deterministically (via history replay)
- Never Lose Information (write-only store)
- Edit the Past (via new events effective at older times)
- Optimize reads (purpose-built read models)
- Enhanced Analytics (analyze all history as it occurred)

Potential Drawbacks

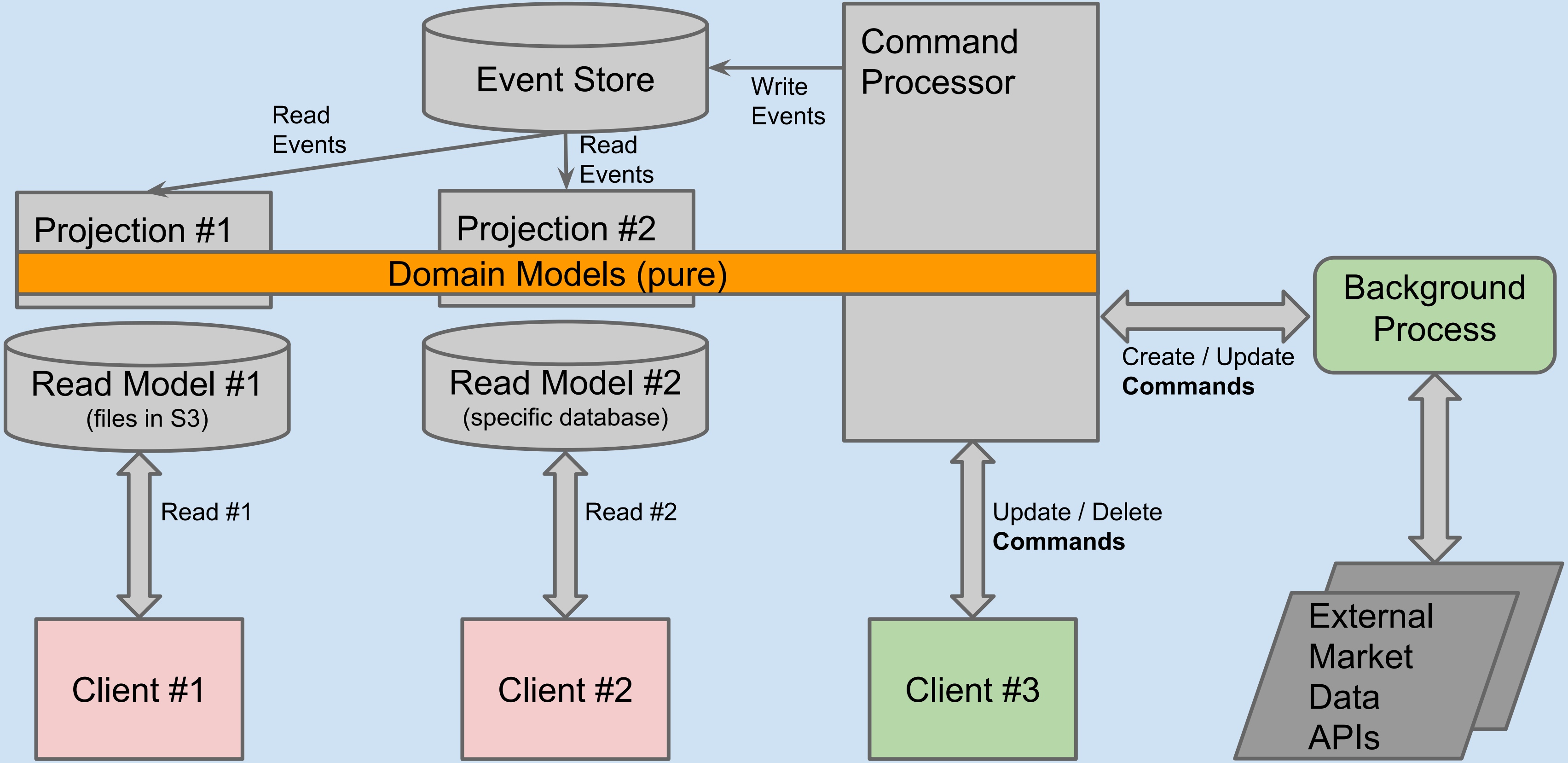
- Eventual Consistency
- No built-in querying of domain models (SELECT name WHERE …)
- Risks of using a new architectural pattern
- Lack of agreement on tools and infrastructure
- Increased storage requirements

What is different from CRUD?

CRUD Micro-Service



Event-sourced Micro-Service



Vocabulary

1. Domain-Driven Design (DDD)
    - Event**:** *something that happened in the past; a fact; a state transition*
    - **Command:** *a request for a state transition in the domain*
    - Aggregate

2. Event Sourcing (ES)
    - Projection
    - Read Model

Aside: Domain Model is Pure?

- FP: A "pure" function doesn't cause any side effects
  - No reads or writes that modify the world
  - No altering a mutable data structure
  - Substitute `f(x)` for its result without changing meaning of program

- An event-sourced domain can be two pure functions
  - `process(currentState, command)   => e1, e2, e3`
  - `apply  (currentState, event)     => nextState`

- Separates the logic of the model from interactions with any changing state in the world

What is different from CRUD?

(Quotes from Greg Young)

"The model that a client needs for the data in a distributed system is screen-based and different than the domain model."

Trade-offs

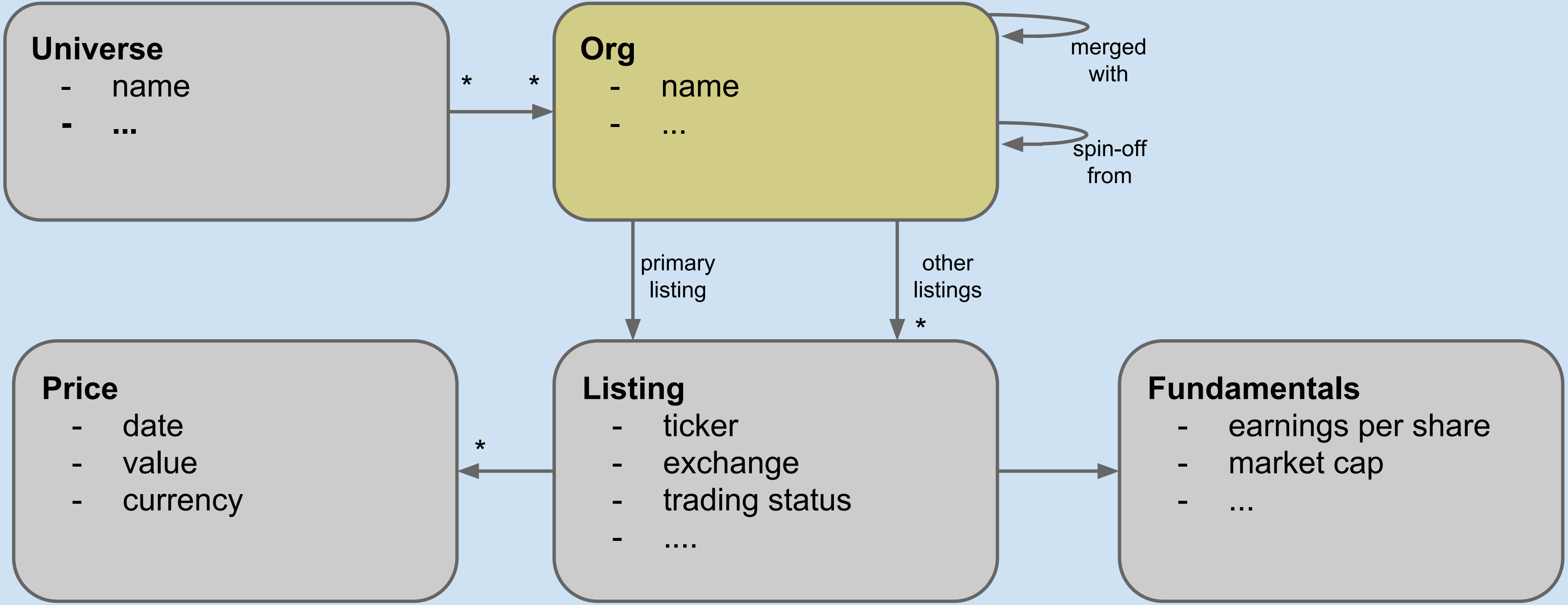
- ACID vs. Eventual Consistency
  - CRUD w/ ACID database
    - Once a row is written, subsequent reads reflect it
    - But: no help with domain-level consistency!
  - Event Sourced
    - Once event is written, subsequent reads reflect it
    - Projections eventually consistent

- Up-front costs
  - Domain modeling is hard!

CRUD Models of Market Data

←1-9
WALL ST

CRUD Models of Market Data



Universe
- name
- ...

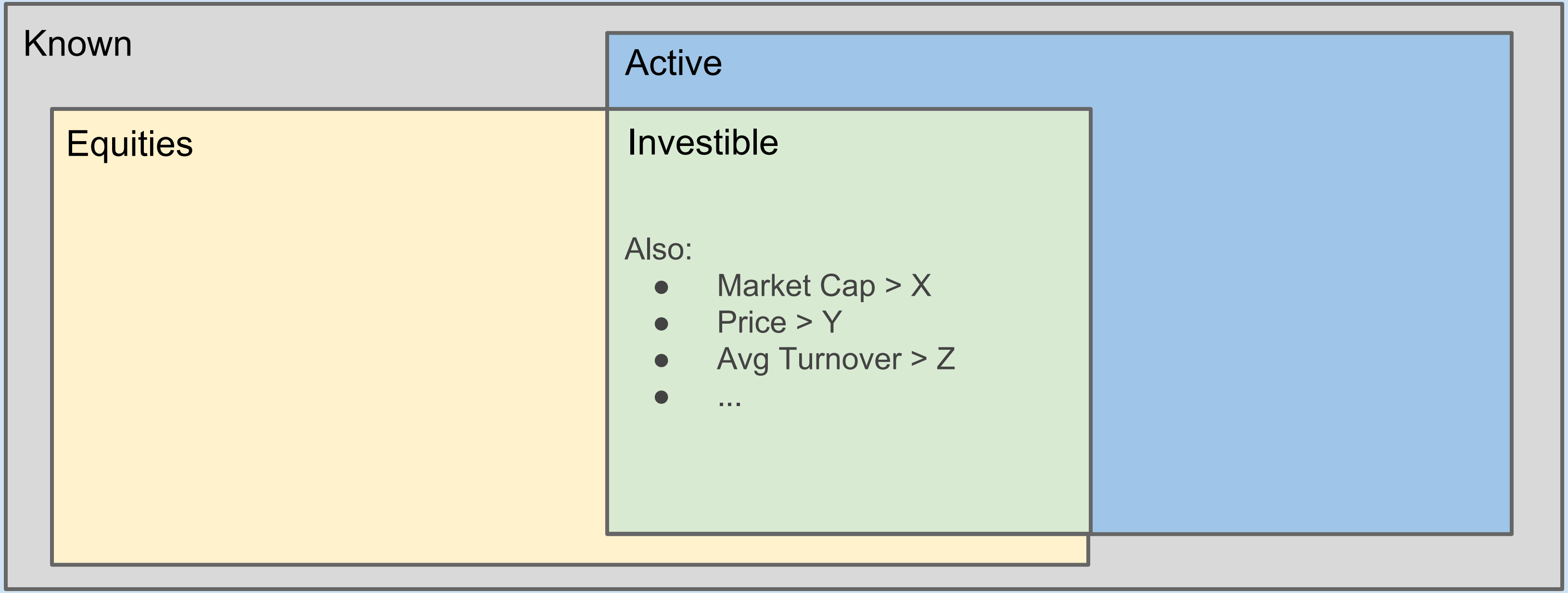
Org
- name
- ...

Listing
- ticker
- exchange
- trading status
- ....

Price
- date
- value
- currency

Fundamentals
- earnings per share
- market cap
- ...

Answerable?

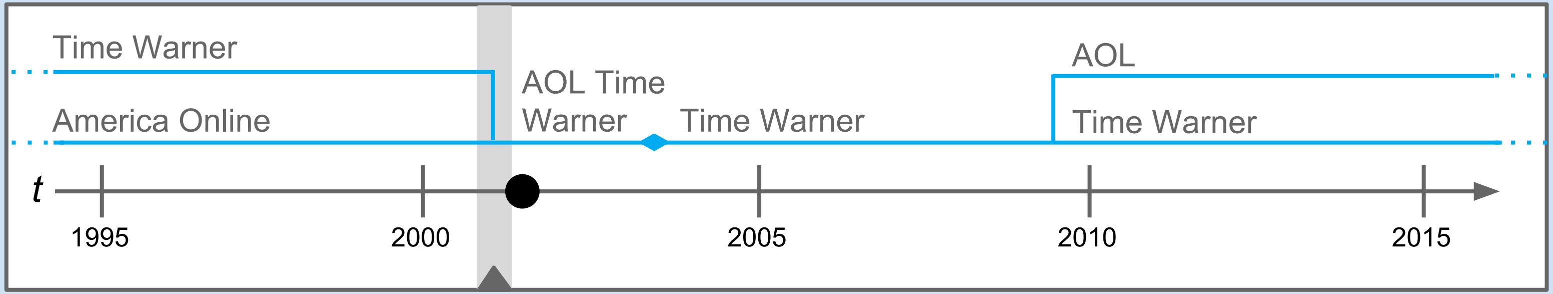


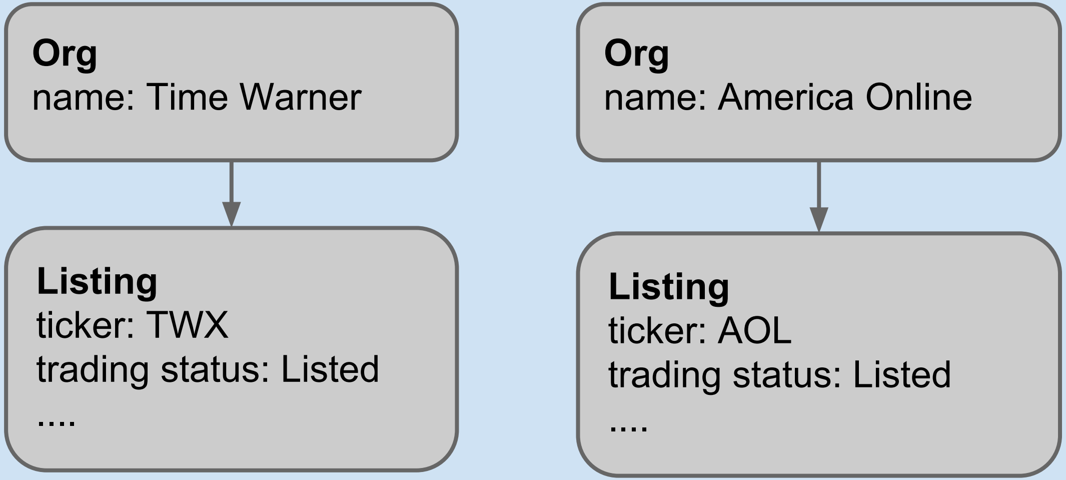
Also:

- Market Cap > X
- Price > Y
- Avg Turnover > Z
- ...

Q: Was AOL in **Investible Universe** in 1995?

2001 - Merger of AOL/Time Warner





Org
name: Time Warner

Listing
ticker: TWX
trading status: Listed
....

Org
name: America Online

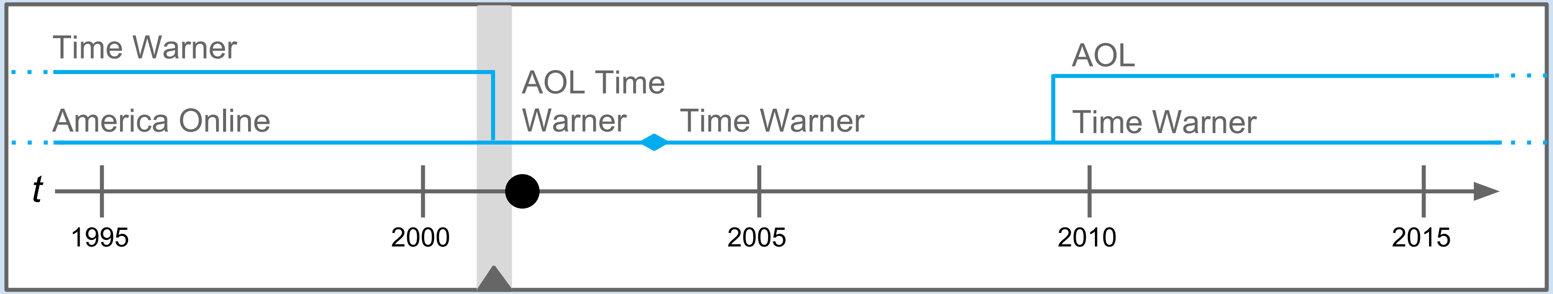
Listing
ticker: AOL
trading status: Listed
....

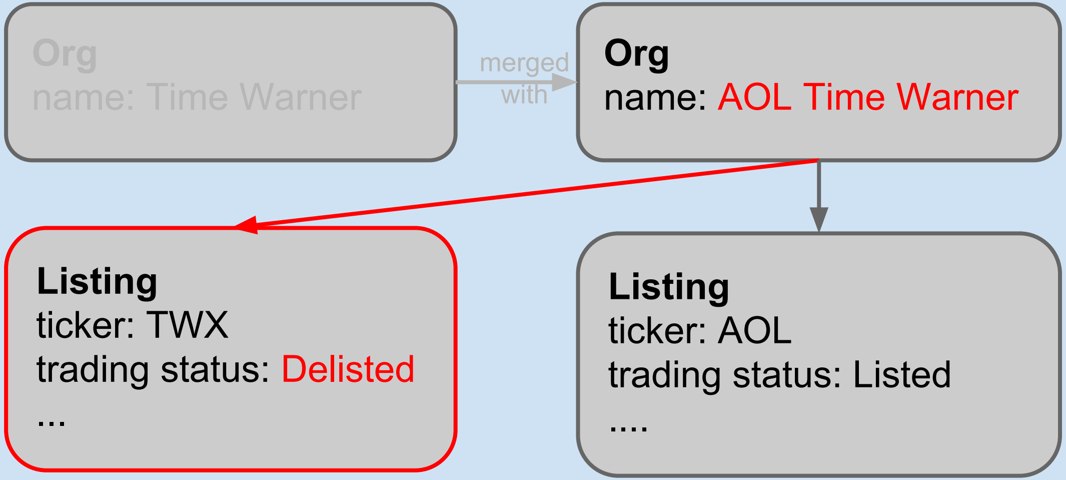
```
Orgs
  .findByName("Time Warner")
  .mergeWith("America Online")

Orgs
  .findByName("America Online")
  .setName("AOL Time Warner)
  .addListing("TWX")

Listings
  .findByTicker("TWX")
  .setTradingStatus("Delisted")
```

2001 - Merger of AOL/Time Warner





Org
name: Time Warner

merged with

Org
name: AOL Time Warner

Listing
ticker: TWX
trading status: Delisted
...

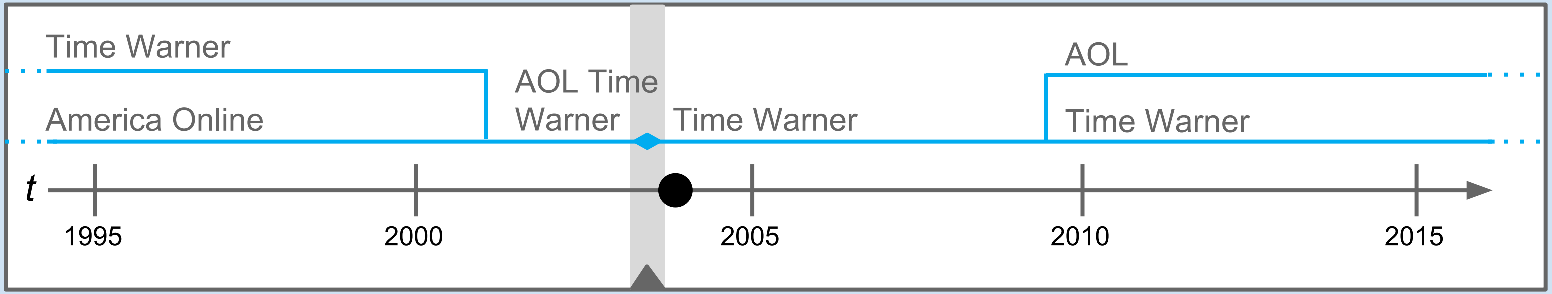
Listing
ticker: AOL
trading status: Listed
....

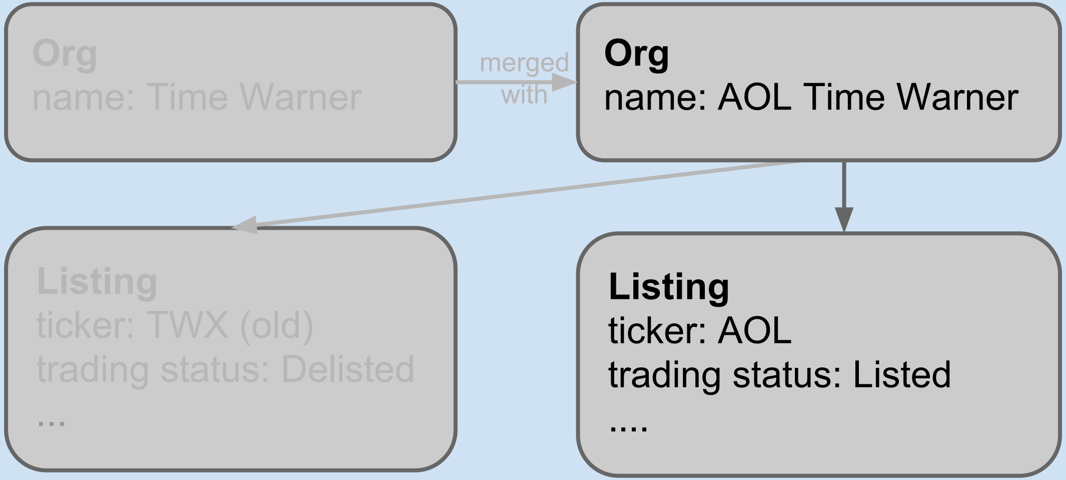
```
Orgs
  .findByName("Time Warner")
  .mergeWith("America Online")

Orgs
  .findByName("America Online")
  .setName("AOL Time Warner)
  .addListing("TWX")

Listings
  .findByTicker("TWX")
  .setTradingStatus("Delisted")
```

2003 - Name / Ticker Change





Org
name: Time Warner

Org
name: AOL Time Warner

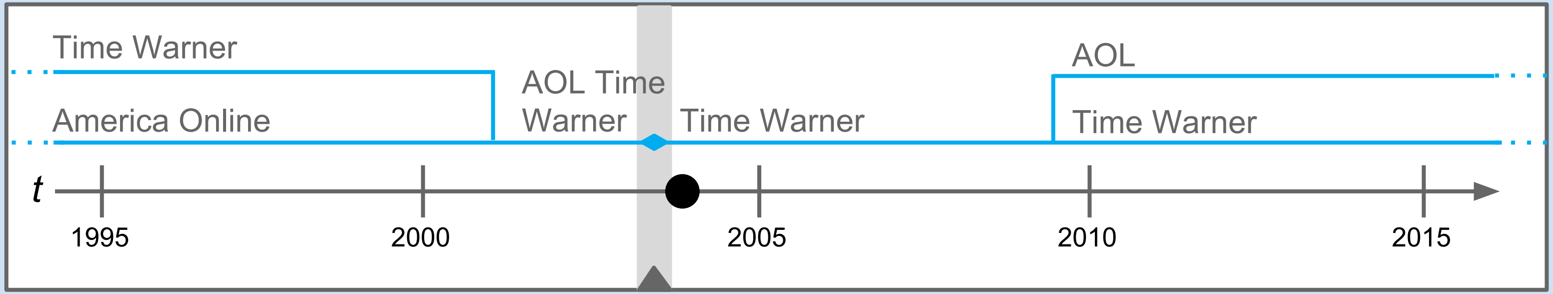
Listing
ticker: TWX (old)
trading status: Delisted
...

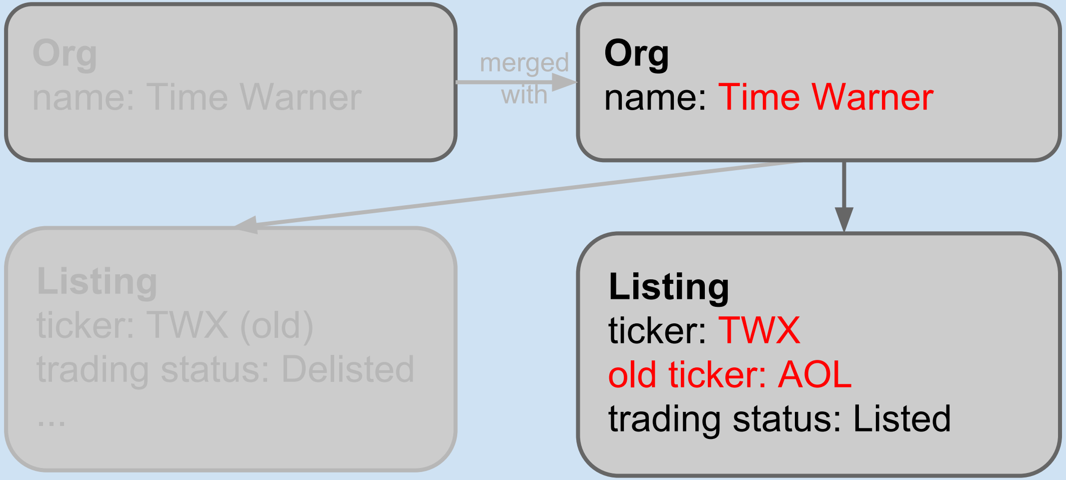
Listing
ticker: AOL
trading status: Listed
....

```
Org
  .findByName("AOL Time Warner")
  .setName("Time Warner")
```

```
Listings
  .findByTicker("AOL")
  .setTicker("TWX")
```

2003 - Name / Ticker Change





Org
name: Time Warner

Org
name: Time Warner

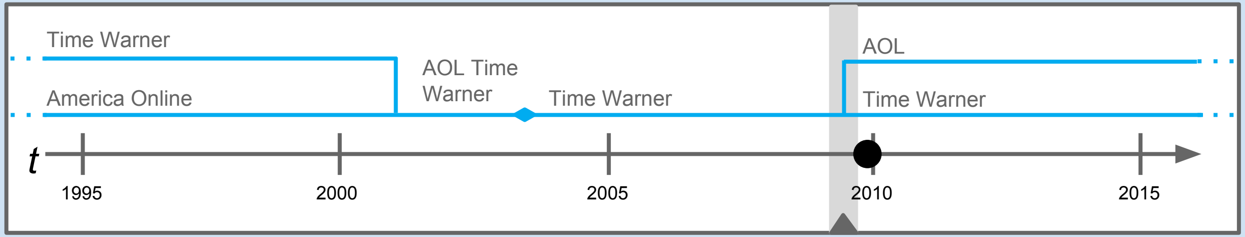
Listing
ticker: TWX (old)
trading status: Delisted
...

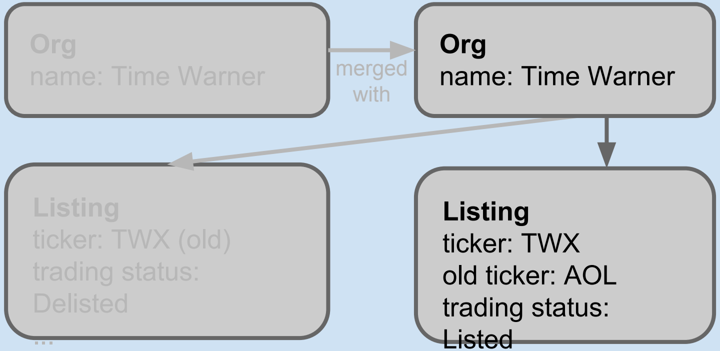
Listing
ticker: TWX
old ticker: AOL
trading status: Listed

```
Org
  .findByName("AOL Time Warner")
  .setName("Time Warner")
```

```
Listings
  .findByTicker("AOL")
  .setTicker("TWX")
```

2009 - AOL Spinoff





Org
name: Time Warner

Listing
ticker: TWX (old)
trading status:
Delisted
...

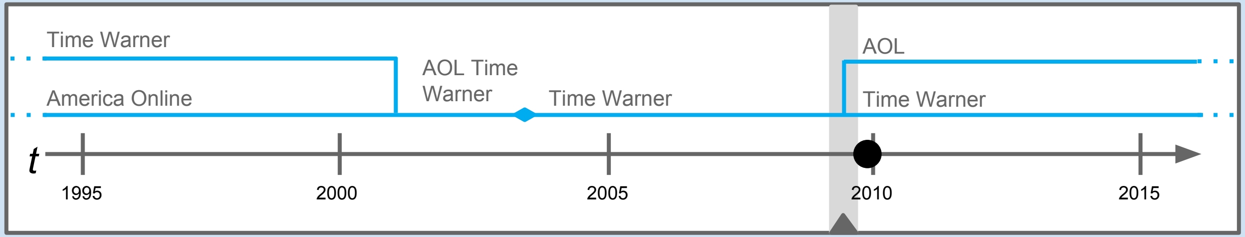
Org
name: Time Warner

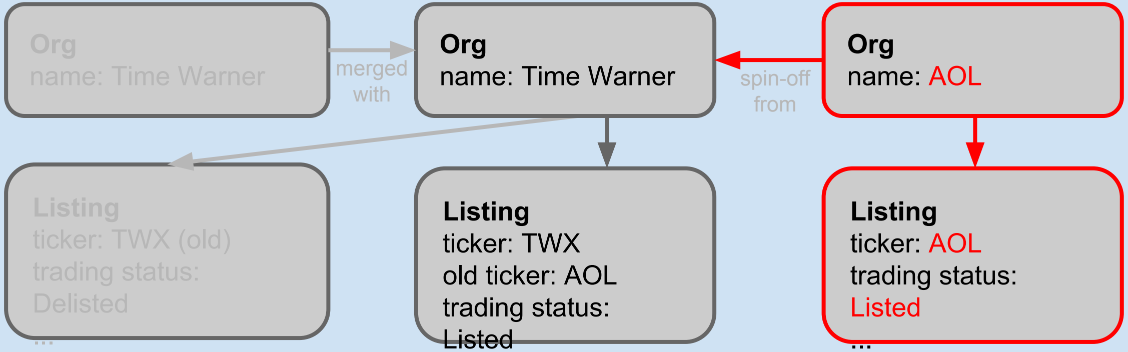
Listing
ticker: TWX
old ticker: AOL
trading status:
Listed

```
Org
  .findByName("Time Warner")
  .spinOff("AOL")
  .addListing("AOL")

Listings
  .newListing("AOL")
```

2009 - AOL Spinoff





Org
name: Time Warner

merged with

Org
name: Time Warner

spin-off from

Org
name: AOL

Listing
ticker: TWX (old)
trading status: Delisted
...

Listing
ticker: TWX
old ticker: AOL
trading status: Listed

Listing
ticker: AOL
trading status: Listed
...

```
Org
  .findByName("Time Warner")
  .spinOff("AOL")
  .addListing("AOL")

Listings
  .newListing("AOL")
```

Answerable with CRUD Models?

Q: Was AOL in **Investible Universe** in 1995?

A: No **and** Yes

- **No** current AOL org (didn't exist at time)
- **Yes** former America Online (now Time Warner)

Complexity in query, requires previous states

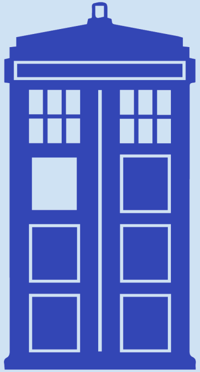
- Query against version columns with date ranges?
- Query against previous versions tables?

CRUD Models - How to Update?

- Update price of TWX on 1995-01-03
  - Original: ~~$5**6**.**2**2~~ Correct: $5**2**.**6**2
- Complexity in query
  - Wrong: `Listings.findByTicker("TWX") // this is AOL!`
  - Right: Complex historical query...
- Complexity in update
  - Org Primary Listing - any change?
  - Org Universe membership - any changes?
  - Support *Two-Dimensional Time* aka *as-of* query?

Problems

Main Problem



Unanswerable questions

- Time travel intractable
- Past not always reproducible

More Problems

- Correctness
  - Divergent interpretations of data
- Availability
  - How often can data be unavailable to clients?
- Performance
  - How fast must operations complete?
- Determinism
  - Reproducing prior states for reporting, debugging, etc.
- Auditability
  - Who changed what when and why?

Problems - Correctness

- Need a new definition of e.g. adjusted price
  - **Old**: unadjusted * splits * spin-offs
  - **New**: unadjusted * splits * spin-offs * dividends

- But...
  - Some client systems still need the old definition
  - CRUD data store didn't store the individual factors

- Common Solutions
  - Add past to relational model? Reprocess?

Problems - Availability

- How long can data be unavailable?
  - **Not long** - End-user client
  - **Hours to Days** - Reporting
- But...
  - Most stringent of client requirements applies to all
  - Cascading failures: unavailability propagates
- Common solutions
  - bulk-heading, circuit-breakers, more servers
  - more complex than necessary?

Problems - Performance

- How fast must operations complete?
  - Writes need to keep up with input
  - Reads have varying requirements
- But..
  - Due to contention on Shared Mutable State, badly performing Reads can impact everything else
- Common Solutions
  - Caching, sharding, more server resources
  - Trade-off with ACID consistency

Problems - Determinism

- Reproducing prior states
  - Reporting Consistently on a Past period
    - Apply adjustments **only** to end of the period
  - Debugging
    - Reproduce state of data in past
- But..
  - Not easy with Shared Mutable State!
- Common Solutions
  - Versioned rows, audit tables, database snapshots

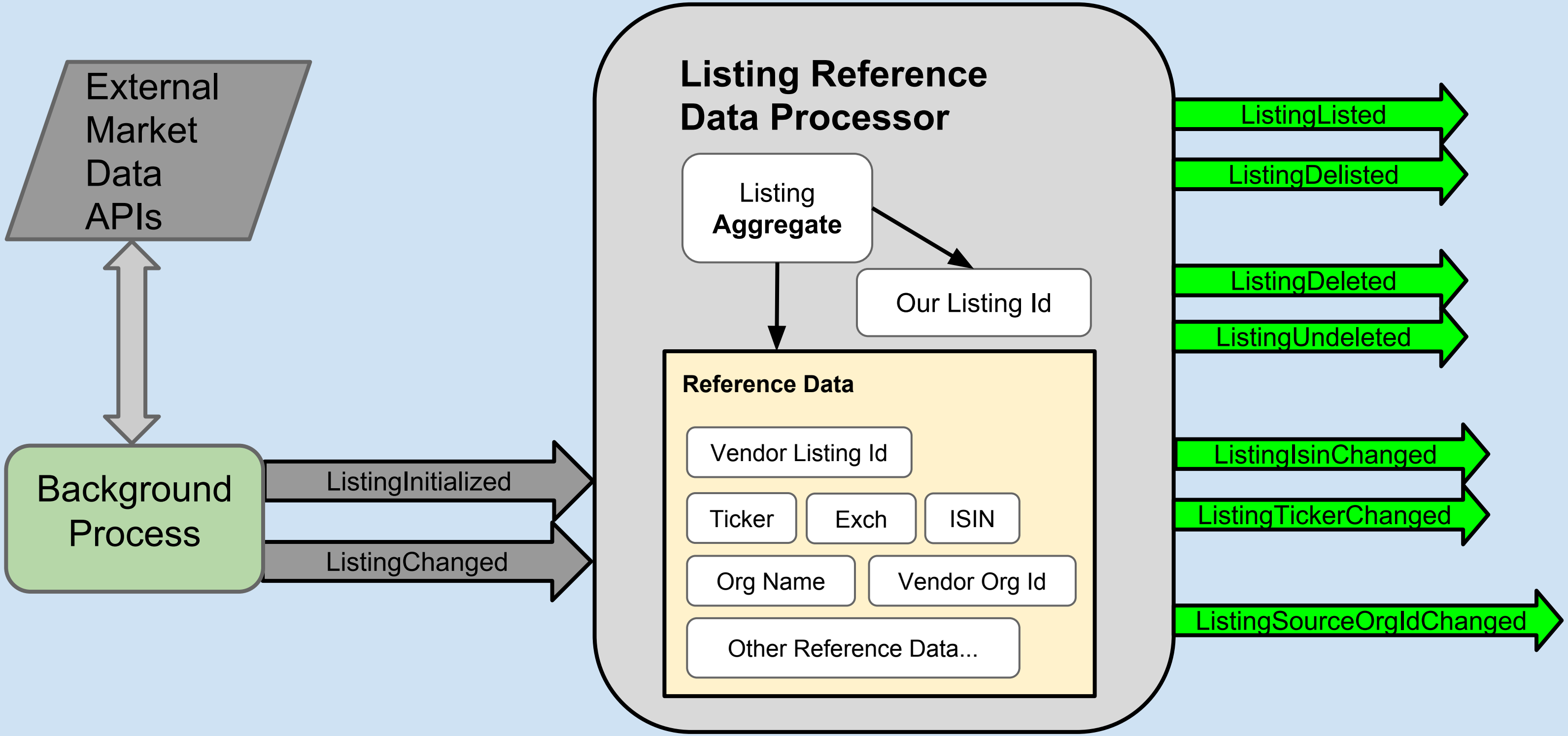
Problems - Auditability

- Why did data change?
    - Attribution (source of data)
    - Security (who did it)
    - History (what and when was previous value)
- But..
    - Not easy with Shared Mutable State!
- Common Solutions
    - Versioned rows, audit tables

Event Sourced Models of Market Data

Creditreform
GÖRTZ
GÖRTZ
GÖRTZ
GÖRTZ

Event Sourced Listings



External Market Data APIs

Background Process

ListingInitialized

ListingChanged

Listing Reference Data Processor

Listing **Aggregate**

Our Listing Id

Reference Data

Vendor Listing Id

Ticker

Exch

ISIN

Org Name

Vendor Org Id

Other Reference Data...

ListingListed

ListingDelisted

ListingDeleted

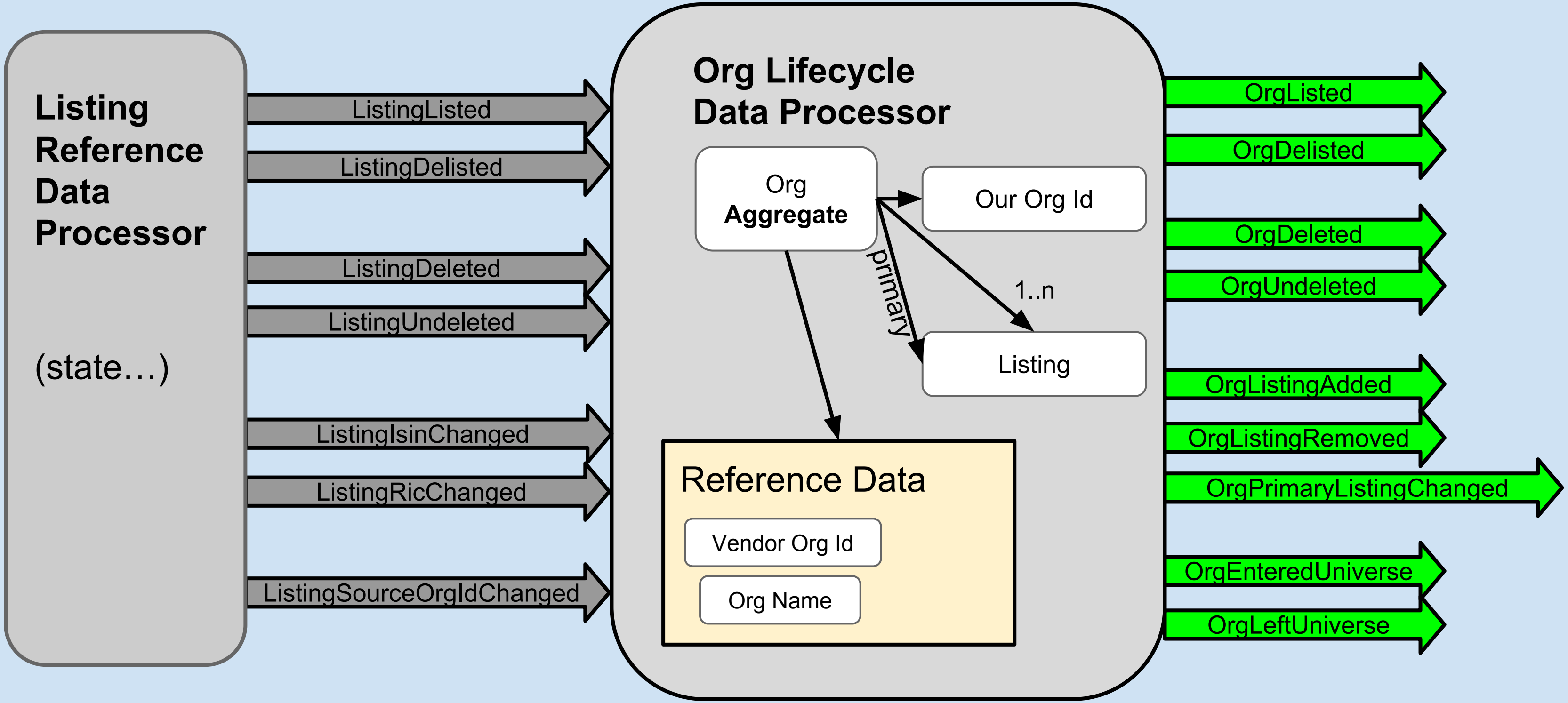
ListingUndeleted

ListingIsinChanged

ListingTickerChanged

ListingSourceOrgIdChanged

Event Sourced Orgs



Vocabulary

1. Domain-Driven Design (DDD)
   - Event**:** *something that happened in the past; a fact; a state transition*
   - Command: *a request for a state transition in the domain*
   - **Aggregate:** *domain objects in a transactionally-consistent unit*

2. Event Sourcing (ES)
   - Projection
   - Read Model

Resulting Read Models

Stock Universes - Read Model

- Generate as CSV files in S3, clients retrieve via REST API

```
GET /universes/1995-01-03
      Org Id,Known Universe?,Active Universe?,Investible Universe?,Primary Listing Id, … Ticker
      "49498",true,true,true,"0x00100b000b569402","US8873173038",..."AOL"
```

- Problems?
  - **Correctness**: Interpretation only for this use case
  - **Availability**: No impact on other use cases
  - **Performance**: No read-side calculations
  - **Determinism**: Can re-generate from event source data
  - **Auditability**: Available in event source data

- Consistency is at domain level -- entire history of universes in this case
  - Generate entire history to S3 bucket, API switches buckets atomically

Answerable via Event Sourcing?

Q: Was AOL in **Investible Universe** in 1995?

```
GET /universes/1995-01-03
    Org Id,Known Universe?,Active Universe?,Investible Universe?,Primary Listing Id, …, Ticker
    "49498",true,true,true,"0x00100b000b569402","US8873173038",...,"AOL"
```

A: **Yes!**   former America Online (now Time Warner)

Trivial query against purpose-built Read Model

Org History - Read Model

- Build directly from indexed event stream, clients retrieve via REST API

```
GET /orgs?ticker=TRW
      [ { eventType: "ListingListed", listing_id: "1", ticker: "AOL", processedAt: "...", effectiveAt: "...", …}
        { eventType: "OrgListed", … },
        { eventType: "ListingAdded", listing_id: "1",  …}, …
        { eventType: "ListingTickerChanged", listing_id: "1", old_ticker: "AOL", ticker: "TWX" …}, ... ]
```

- Problems?
    - **Correctness**: Interpretation only for this use case
    - **Availability**: No impact on other use cases
    - **Performance**: No read-side calculations
    - **Determinism**: Reads directly from event source data
    - **Auditability**: Available in event source data

- Consistency is at domain level -- entire history of single org in this case
    - Generate entire history on-the-fly directly from source (indexed!)

Vocabulary

1. Domain-Driven Design (DDD)
    - Event**:** *something that happened in the past; a fact; a state transition*
    - Command: *a request for a state transition in the domain*
    - Aggregate**:** *domain objects in a transactionally-consistent unit*

2. Event Sourcing (ES)
    - **Projection**: *to derive current state from the stream of events*
    - **Read Model**: *a model of current state designed to answer a query*

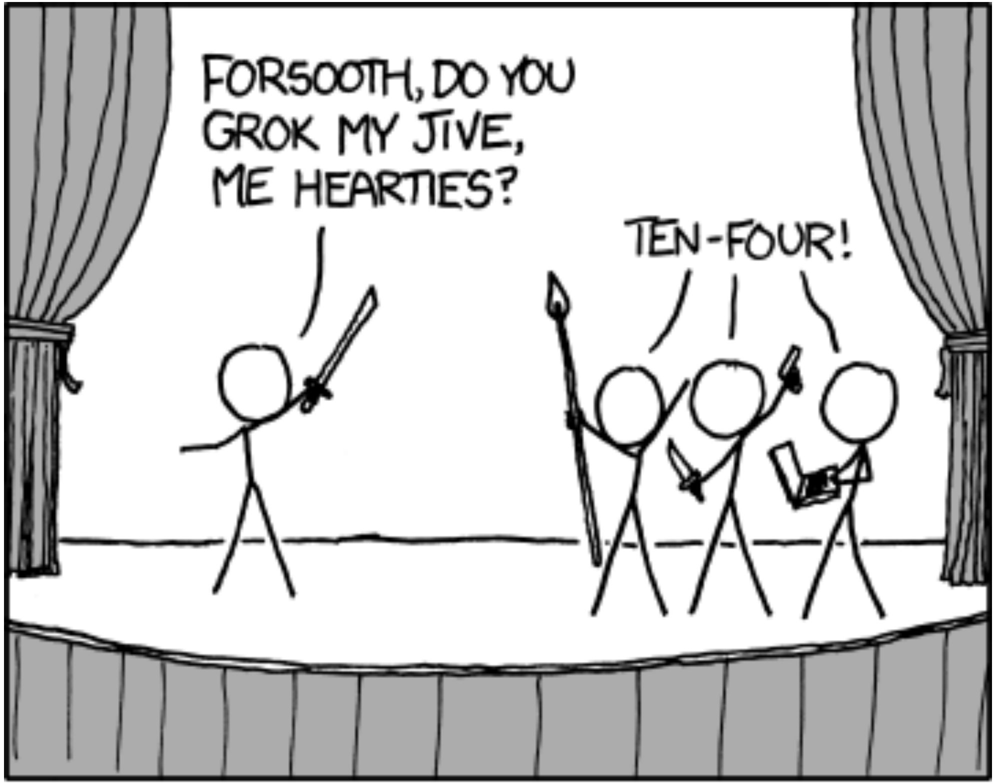
Read Model vs Cache

A **cache** is a query intermediary. It holds a previous response until invalidated, then queries again.

A **read model** does not query. Applying new events to it changes the answer it returns.

Example: count of people in room

- both may return 10 when the answer is now 11 or 9 - staleness
- cache counts the people - slow and may require locking the doors
- read model applies the entrance/exit events - like a click counter - no impact on people nor doors



A FEW CENTURIES FROM NOW, ALL THE ENGLISH OF THE PAST 400 YEARS WILL SOUND EQUALLY OLD-TIMEY AND INTERCHANGEABLE.

Conclusions

- Answer the "unanswerable"

- Avoid impacts on other use cases

- Keep facts that may answer future questions

Questions?

Follow up later

Marc Siegel

@ms_yd

Brian Roberts

@flicken

“So long,

and thanks for

all the fish.”

Events vs Audit Tables or Versioned Rows

An **audit table** or **row version column** use shared mutable state as the source of truth, and additionally store some history. Inconsistencies are hard to fix, edits of the past are challenging, new use cases can be challenging.

An **event store** is an append-only list of immutable facts. What has occurred is recorded, and can be replayed to interpret according to a future use case.

Example: count of people in room

- audit table is a logbook in the room - query may be complex, facts may not be consistent, depends on keeping it up to date
- versioned rows is a logbook carried by each person - same issues as audit table
- event store is "just the facts", only interpretation changes