Real-time Financials with Microservices and Functional Programming

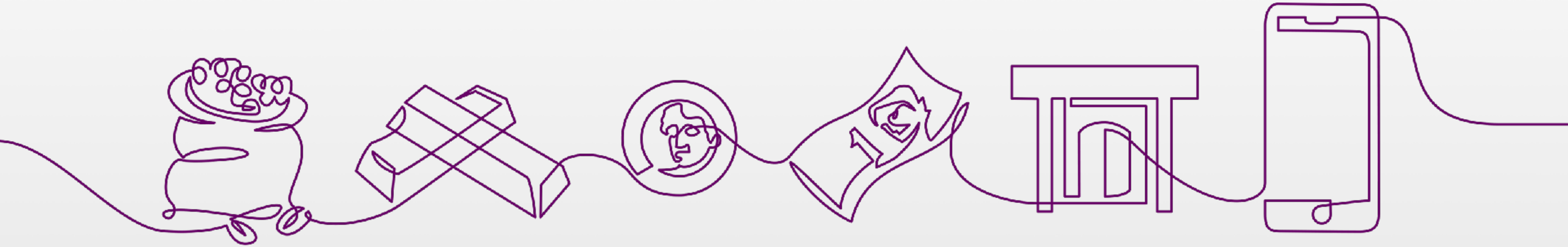
Vitor Guarino Olivier
vitor@nubank.com.br
@ura1a
https://nubank.com.br/

nu bank

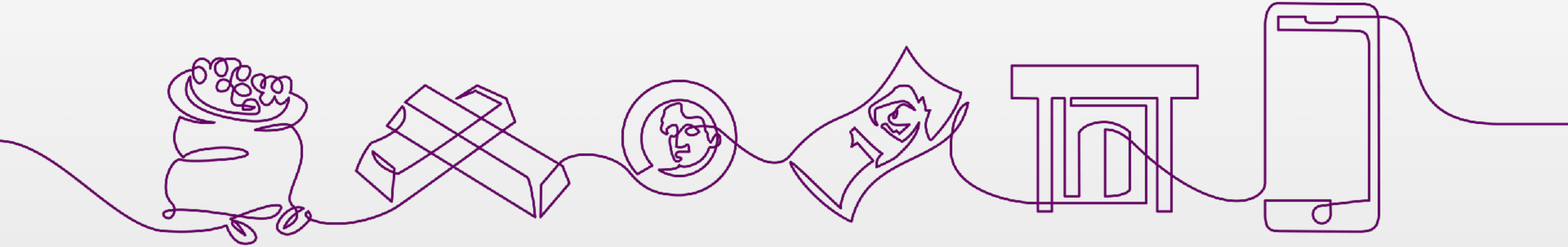
MAIN PRODUCT

Live since September 2014

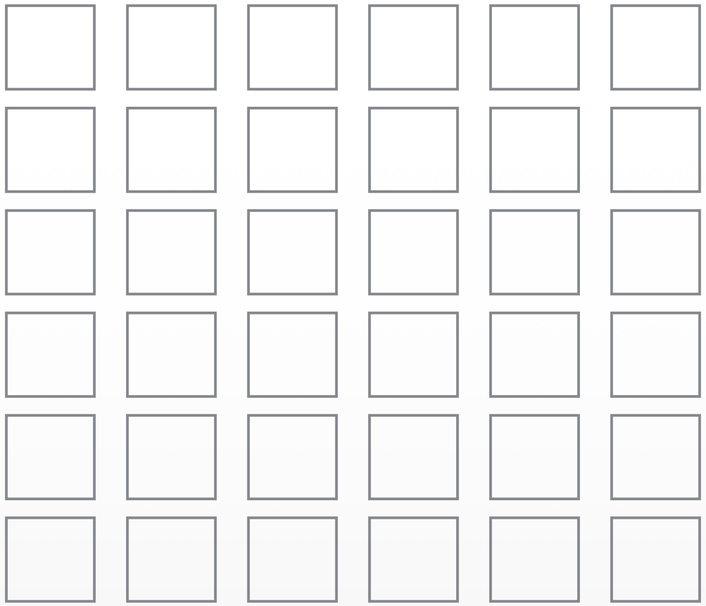
A TECHNOLOGY DRIVEN APPROACH TO FINANCIAL SERVICES



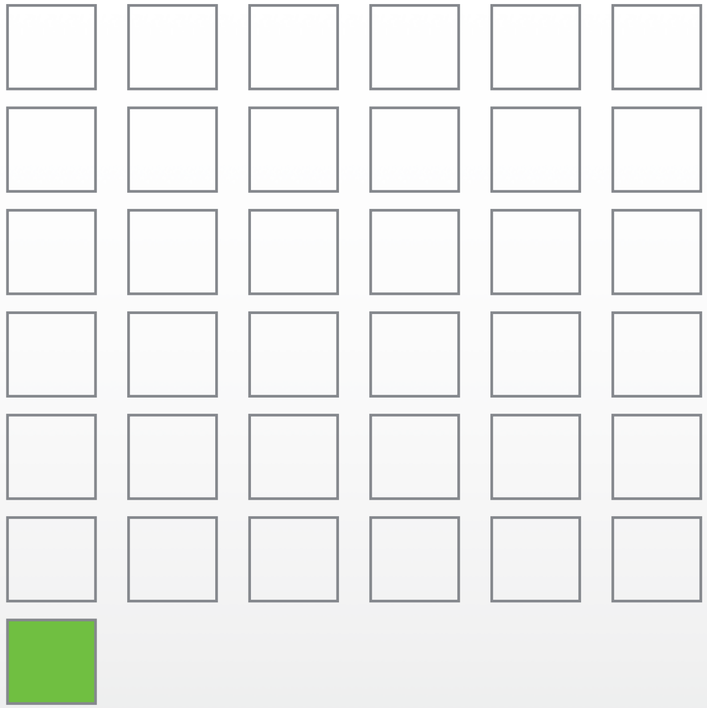
CONTINUOUS DELIVERY



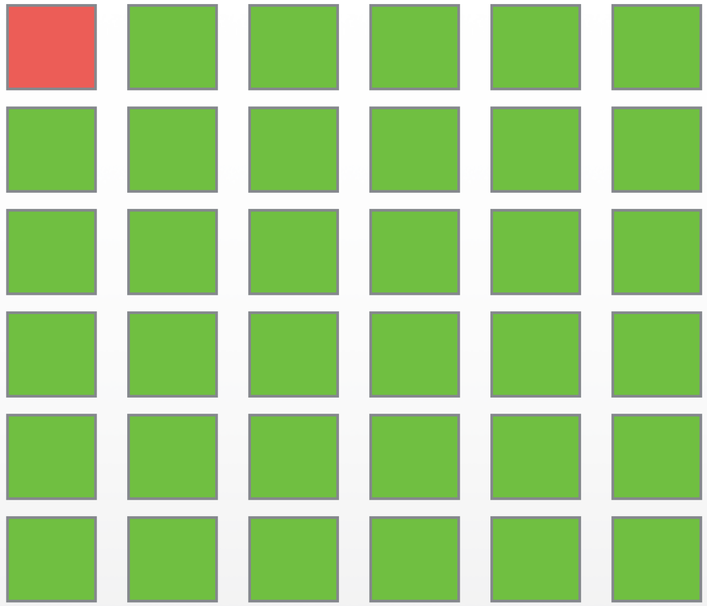
MICROSERVICES



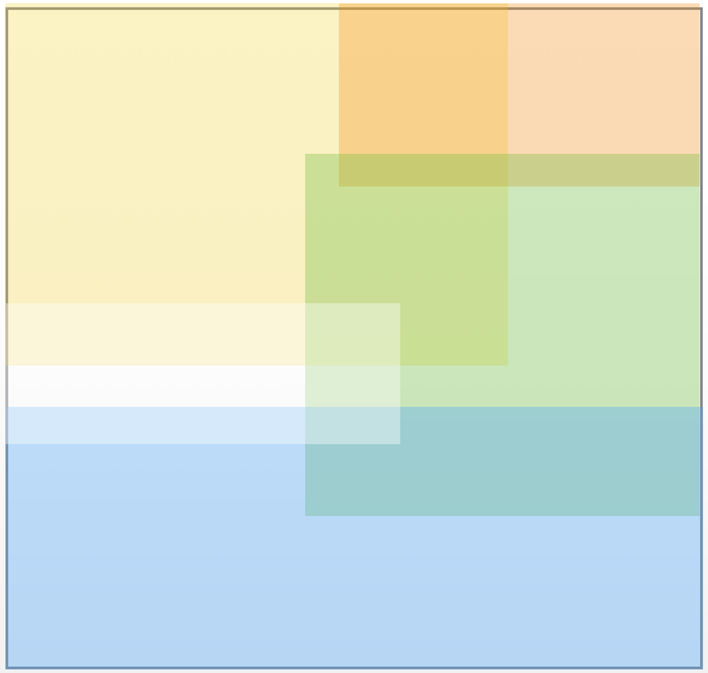
INDEPENDENTLY AND CONTINUOUSLY DEPLOYABLE

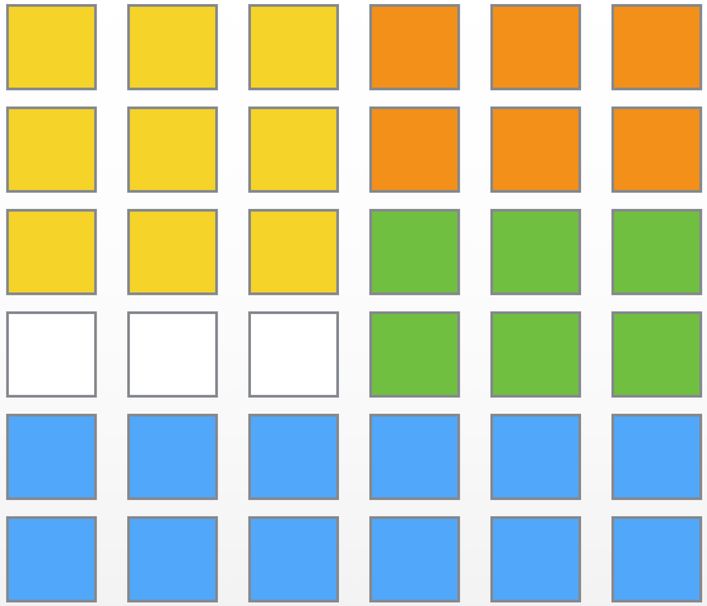


DECOUPLED AND EASY TO REPLACE



BOUNDED BY CONTEXT AND INDEPENDENTLY DEVELOPED



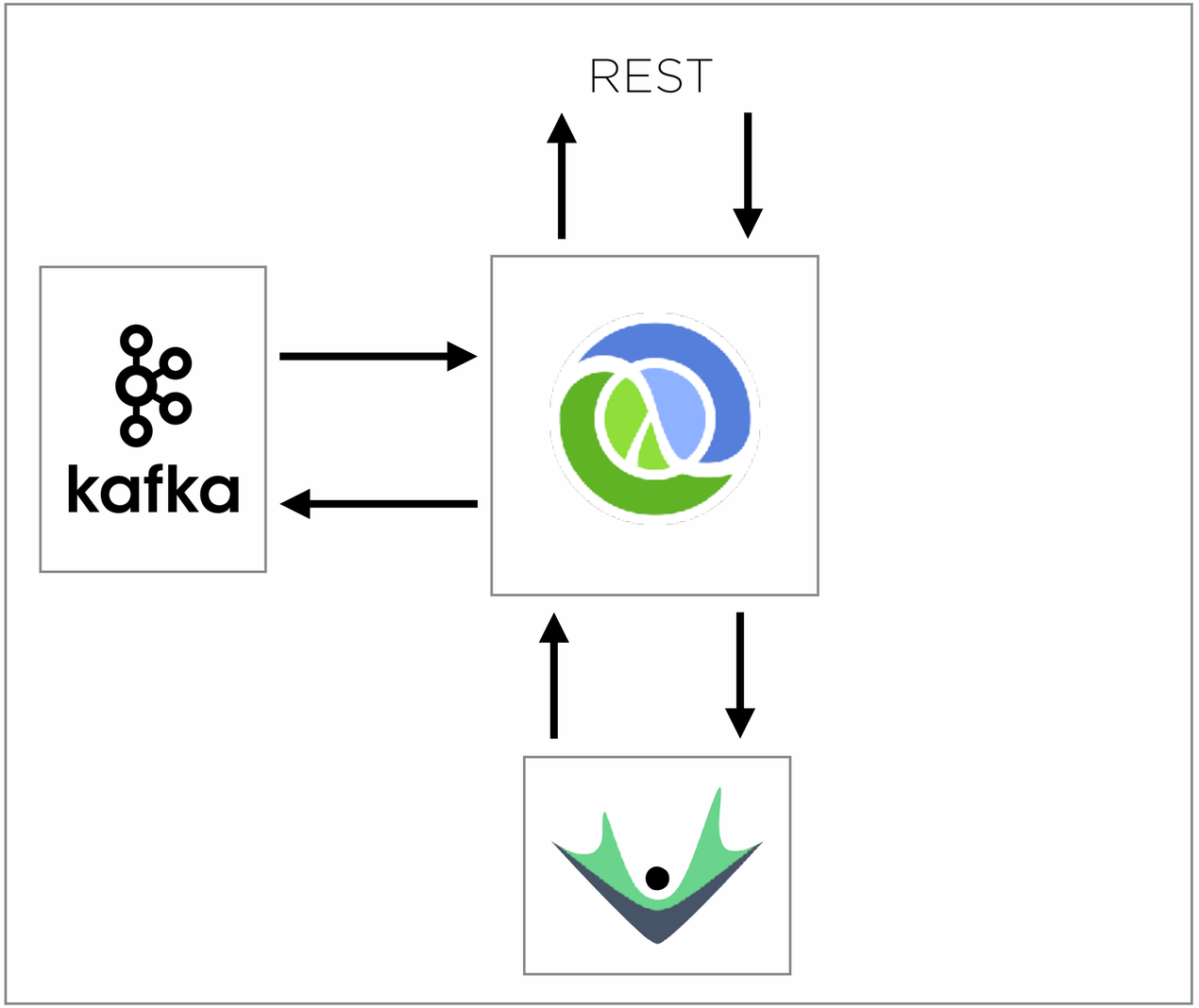


WHAT HAPPENS WHEN WE NEED TO COMBINE DATA ACROSS SEVERAL SERVICES?

ESPECIALLY IN REAL-TIME

SERVICE ARCHITECTURE

- Written in Clojure (functional)
- Producer/Consumer to Kafka
- Persistence with Datomic
- REST APIs
- Running on AWS, 2 AZs, config as code, immutable infra, horizontally scalable, sharded by customers

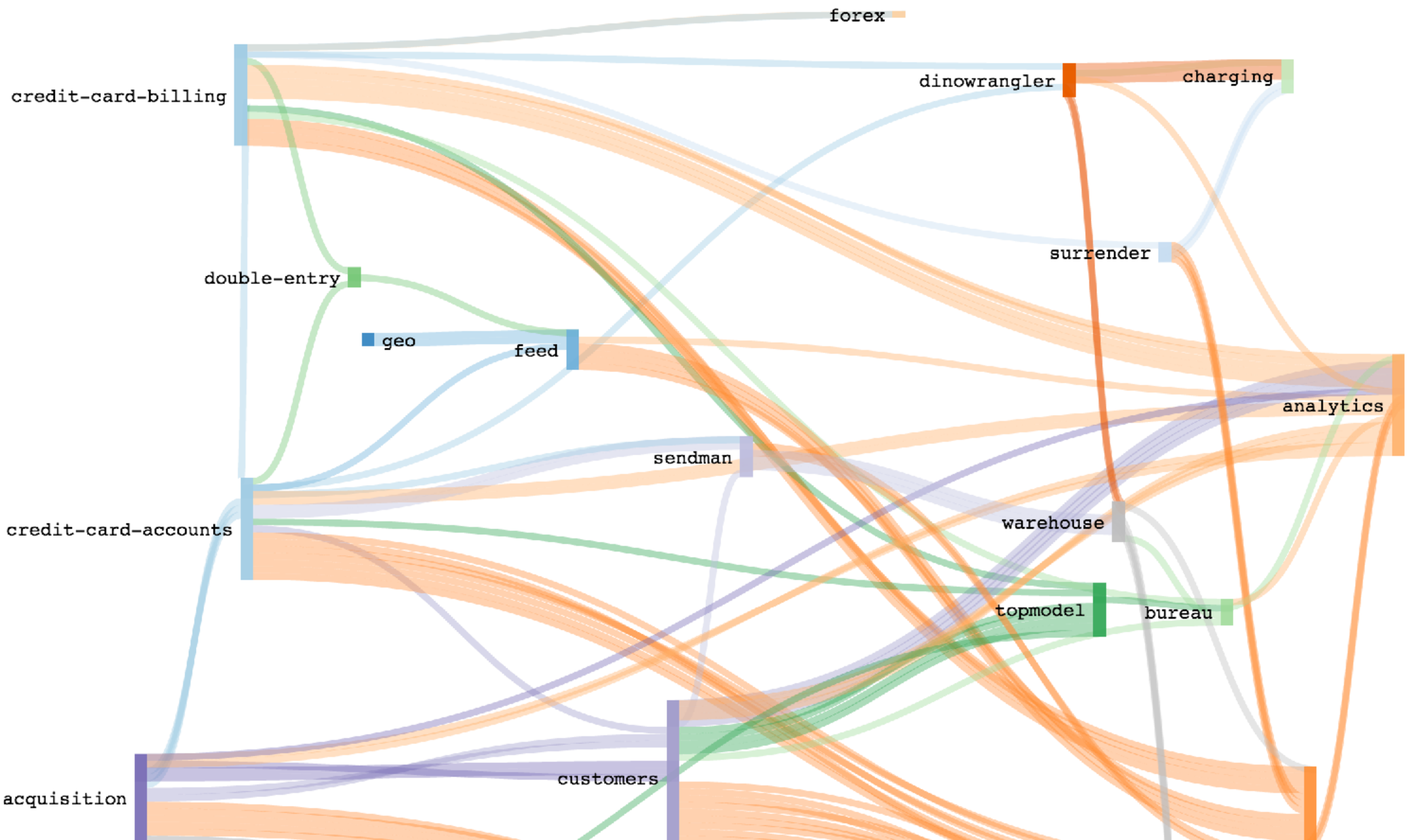


DATOMIC

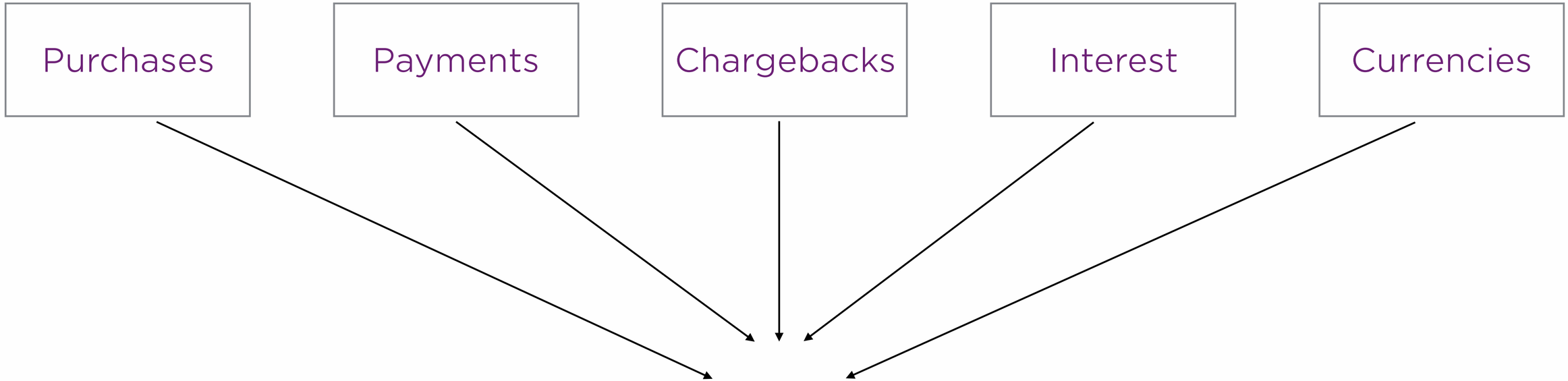
- Immutable, append-only database
- A database that works a lot like git
- ACID on writes (atomic, consistent, isolated, durable)

The Problem

WE HAVE OVER 90 SERVICES



THE PROBLEM:
A LOT OF BUSINESS LOGIC DEPENDS ON DATA ACROSS MANY SERVICES



Should I authorize a purchase? Should I block a card? Should I charge interest?

THE PROBLEM:
WE ARE SHOWING THESE NUMBERS TO THE CUSTOMER IN REAL TIME

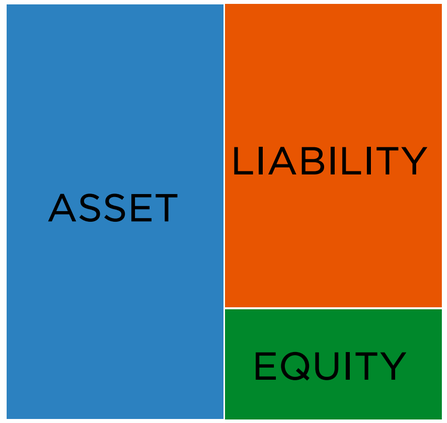
THE PROBLEM:
NO CANONICAL DEFINITION OF OUR KEY NUMBERS

- Ad-hoc definitions created by analysts and engineers

- Analysis vs. operational definition gap

- Nubank, investors, customers, and regulators
are all worried about the same numbers.

A BALANCE SHEET IS THE CANONICAL WAY OF REPRESENTING FINANCIAL INFO



- We can apply **generally accepted accounting principles** (verifiable, unbiased)

- **Conservation of money** (every credit should have a debit)

- One of the **original event-sourced systems**

THE MODEL

- Book-account: A customer owned balance sheet account
ex: cash, prepaid, late, payable

- Entry: represents a debit and a credit to two book-accounts

- Balance: cumulative sum of entries of a book account

- Movement: a collection of entries. Maps one Kafka message to one db transaction

- Meta-entity: it's a reference to the external entity that originated the event

-Algebraic Models For Accounting Systems
by Salvador Cruz Rambaud and José Garcia Pérez

Double-entry accounting service

nu bank

OUR GOAL FOR OUR ACCOUNTING LEDGER (aka DOUBLE-ENTRY SERVICE)

- Event-driven, via kafka. (we could subscribe to existing topics)

- High availability to other services, clients, and analysts in real-time

- Traceability of when and why we were inconsistent (strong audit trail)

- Resilient to distributed systems craziness

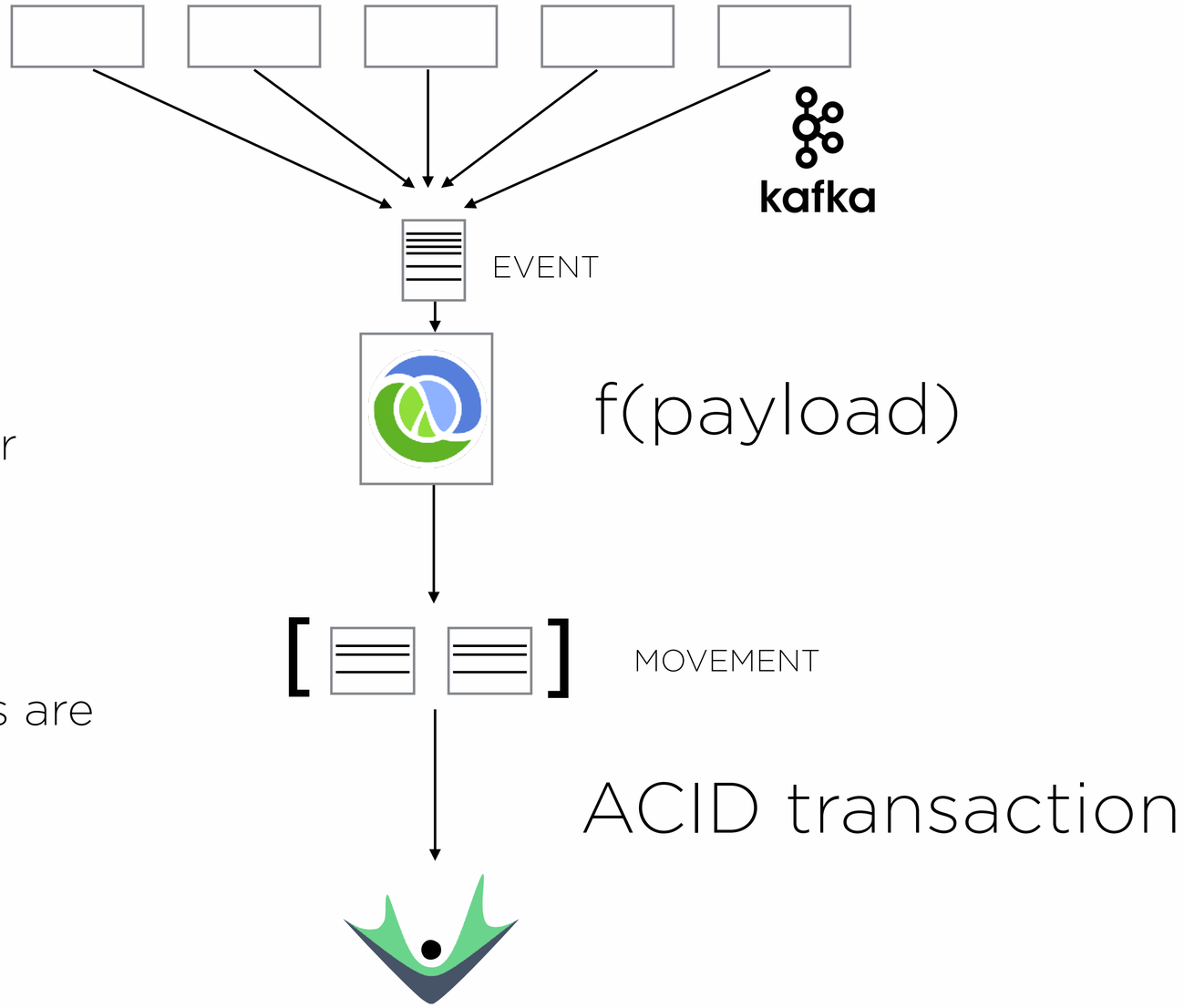
THE IDEAL FLOW

- No mutable state

- Event ordering doesn't matter

- Thread safe

- Needs to guarantee all events are consumed



Initial Balances:
Current Limit R$ 1000, Current Limit Offset R$ 1000

```
{:purchase
 {:id           (uuid)
  :amount       100.0M
  :interchange  1M
  :post-date    "2016-12-01"}}
```

```
[{:entry/id              (uuid)
  :entry/amount          100.0M
  :entry/debit-account   :asset/settled-purchase
  :entry/credit-account  :liability/payable
  :entry/post-date       "2016-12-01"
  :entry/movement        new-purchase}

 {:entry/id              (uuid)
  :entry/amount          100M
  :entry/debit-account   :liability/current-limit
  :entry/credit-account  :asset/current-limit
  :entry/post-date       "2016-12-01"
  :entry/movement        new-purchase}

{:entry/id               (uuid)
  :entry/amount          1M
  :entry/debit-account   :liability/payable
  :entry/credit-account  :pnl/interchange-revenue
  :entry/post-date       "2016-12-01"
  :entry/movement        new-purchase}
]
```

recognize receivable/payable

reduce limit

recognize revenue

Final Balances:
Current Limit: R$ 900, Current Limit: Offset R$ 900
Settled Purchase: R$ 100, Payable: R$ 99, Interchange Revenue: R$ 1

WE CAN'T GUARANTEE CONSISTENCY, BUT WE CAN MEASURE IT

- Service downtime

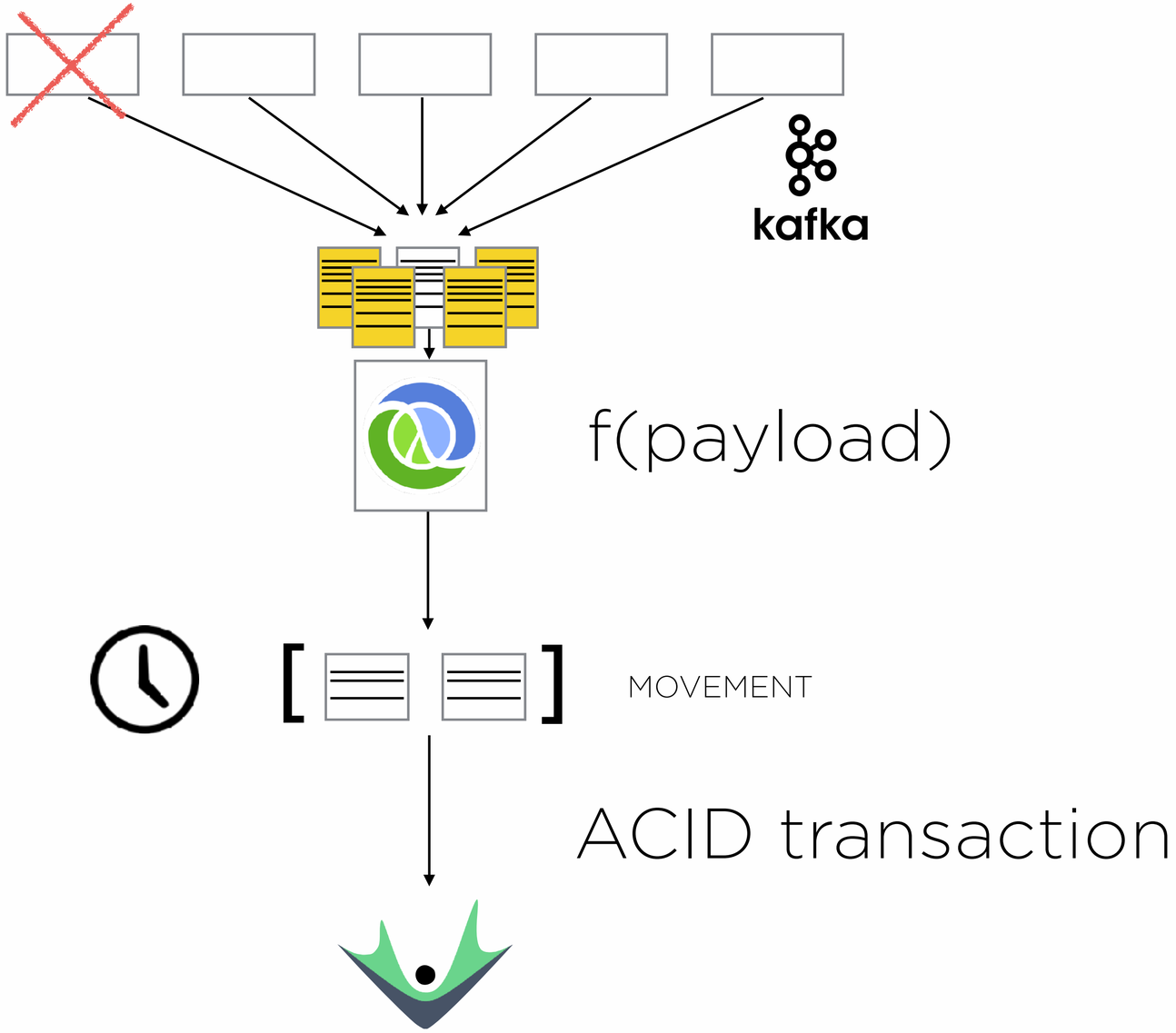
  post-date vs. produced-at

- Kafka Lag

  produced-at vs. consumed-at

- Processing time

  consumed-at vs. db/txInstant



PURE FUNCTIONS OF THE PAYLOAD WON'T ALWAYS WORK

The Stateful Flow

nu bank

Initial Balances:
Current Limit: R$ 900, Current Limit Offset: R$ 900
Late: R$ 100, Payable: R$ 99, Interchange Revenue: R$ 1

```
{:payment
 {:id          (uuid)
  :amount      150.00M
  :post-date   "2016-12-01"}}
```

```
[{:entry/id              (uuid)
  :entry/amount          100.0M
  :entry/debit-account   :asset/cash
  :entry/credit-account  :asset/late
  :entry/post-date       "2016-12-01"
  :entry/movement        new-payment}

 {:entry/id              (uuid)
  :entry/amount          100M
  :entry/debit-account   :asset/current-limit
  :entry/credit-account  :liability/current-limit
  :entry/post-date       "2016-12-01"
  :entry/movement        new-payment}

{:entry/id               (uuid)
  :entry/amount          50M
  :entry/debit-account   :asset/cash
  :entry/credit-account  :liability/prepaid
  :entry/post-date       "2016-12-01"
  :entry/movement        new-payment}
]
```

amortize debt

increase limit

recognize prepaid amount

Final Balances:
Current Limit: R$ 1000, Current Limit Offset: R$ 1000
Cash: R$ 150, Prepaid R$ 50, Payable: R$ 99, Interchange Revenue: R$ 1

Initial Balances:

Late: R$ 100

```
{:payment
 {:id          (uuid)
  :amount      150.00M
  :post-date   "2016-12-01"}}
```

```
[{:entry/id              (uuid)
  :entry/amount          100.0M
  :entry/debit-account   :asset/cash
  :entry/credit-account  :asset/late
  :entry/post-date       "2016-12-01"
  :entry/movement        new-payment}

 {:entry/id              (uuid)
  :entry/amount          100M
  :entry/debit-account   :asset/current-limit
  :entry/credit-account  :liability/current-limit
  :entry/post-date       "2016-12-01"
  :entry/movement        new-payment}

{:entry/id               (uuid)
  :entry/amount          50M
  :entry/debit-account   :asset/cash
  :entry/credit-account  :liability/prepaid
  :entry/post-date       "2016-12-01"
  :entry/movement        new-payment}
]
```

amortize debt

increase limit

recognize prepaid amount

Final Balances:

Prepaid R$ 50

THE STATEFUL FLOW

- Adapters are a function of the event payload AND current balances

- Balances can't change during calculations

- Movements in the past will modify all future balances

- Can't allow for data to be corrupted depending on the order of the events

INVARIANTS

INVARIANTS

- We can establish invariants that must hold true at all times

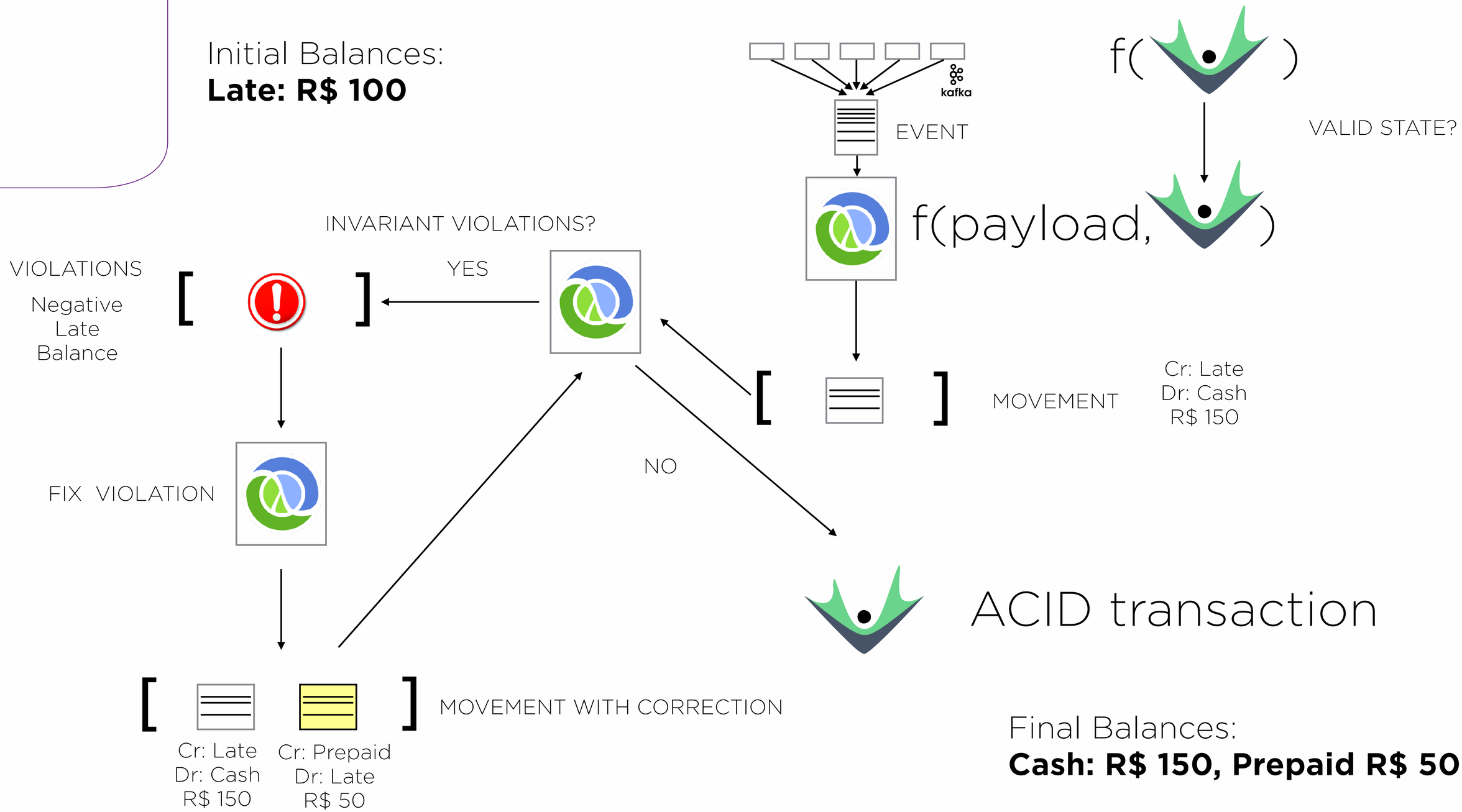
- Some balances can’t coexist (no late alongside prepaid)

- Some balances can’t be negative (cash)

- Some can’t be positive (credit-loss)

THE STATEFUL FLOW

Initial Balances:
Late: R$ 100



Final Balances:
Cash: R$ 150, Prepaid R$ 50

CHALLENGES

CHALLENGES

- Fixing invariants logic is extremely complex.

- Other services bugs may generate incorrect entries that will need to be fixed

- Datomic indexing is tested until 10 billion facts.

- Datomic isn’t the best option for analytical workload, especially with sharded dbs

GENERATIVE TESTING

- Write a function that describes a property that should always hold true instead of describing input and expected output,

- Properties that should hold true are the same invariants that are guaranteed in prod

- We generate random events from our schemas (bill, purchases, payments, etc)

- Embed the least amount of domain logic assumptions

GENERATIVE TESTING

```clojure
(ns double-entry.controllers.rulebook-test
  (:require [midje.sweet :refer :all]
            [clojure.test.check.properties :as prop]
            [clojure.test.check :as tc]
            [schema-generators.generators :as g]
            [clojure.test.check.generators :as gen]))
```

```clojure
(def balances-property
  (prop/for-all [account (g/generator Account)
                 events  (gen/vector (gen/one-of [(g/generator Purchase)
                                                  (g/generator Payment)
                                                  ...]))]
    (->> datomic
         (consume-all! account events)
         :db-after
         (balances-are-positive!)))

(fact (tc/quick-check 500 balances-property) => (th/embeds {:result true}))
```

MONITORING / REPLAY HISTORY TOOLING

- We set sanity checks to make sure events aren't missing

- Other services have republish endpoints
(same payload and meta data as original thanks to datomic)

- We have an endpoint that can retract all entries for a customer
(resets business timeline, but not DB)

SHARDING BY CUSTOMER / TIME

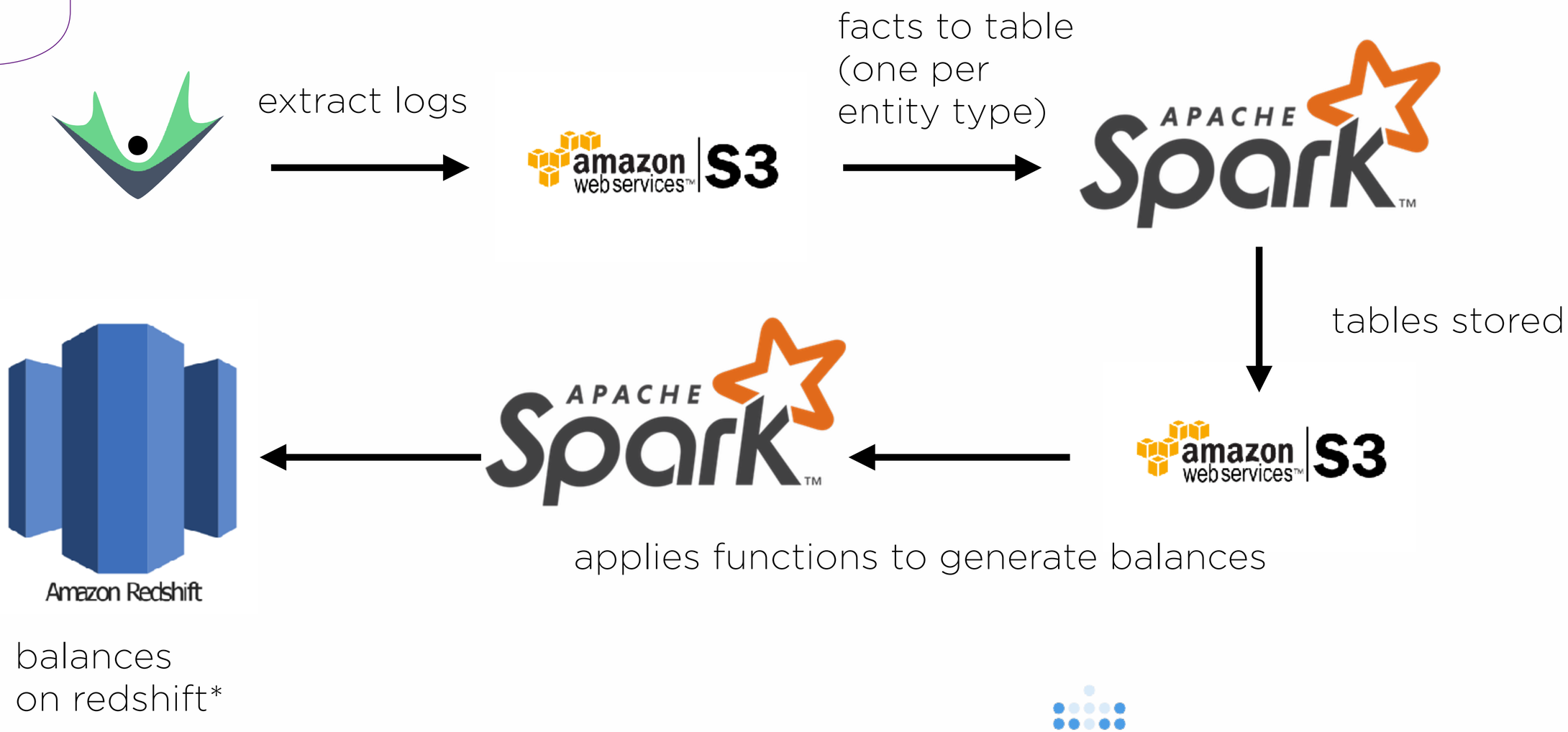
- No cross customer entries allows for per customer sharding

- As time passes, any single customer's db will approach infinite datoms

- simple representation of the end state of the customer at a time shard: final balance of each of the book accounts

- We shard the database by time fairly often.

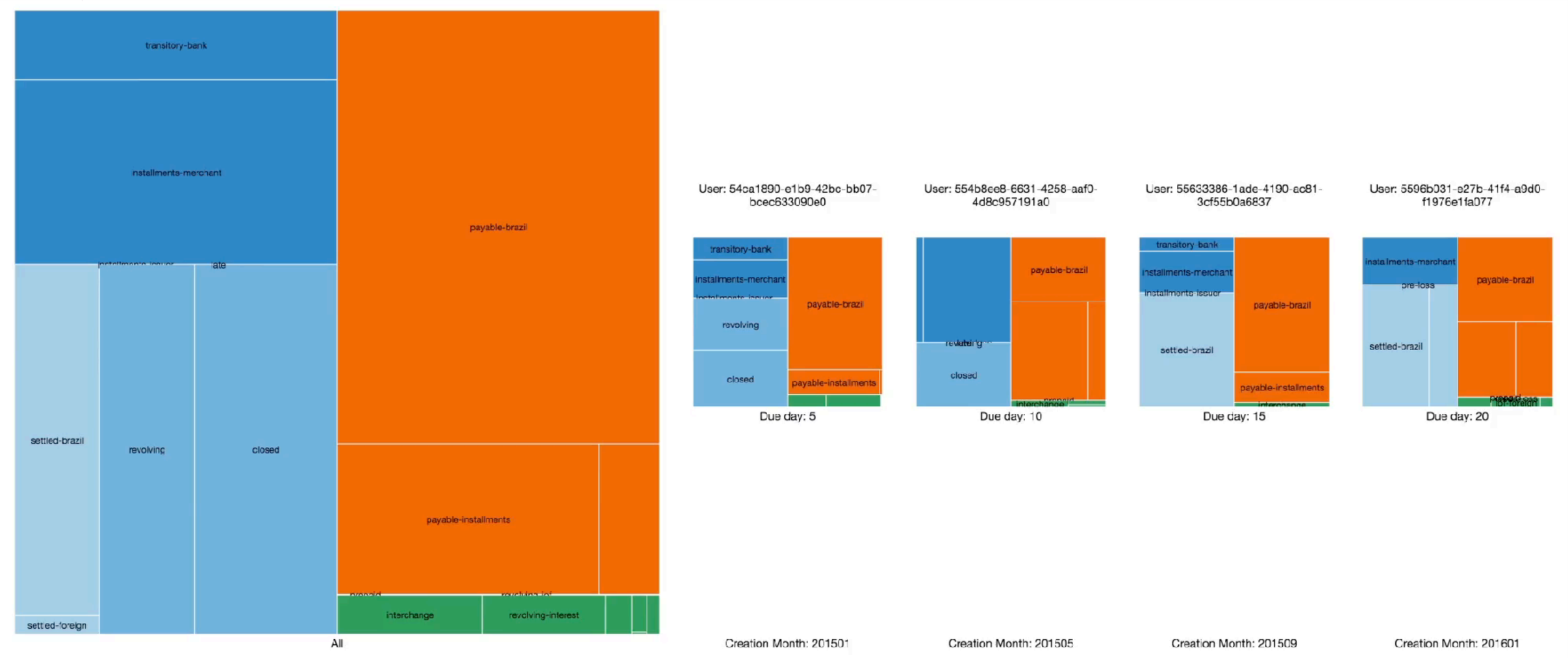
ETL



* also accessible through metabase

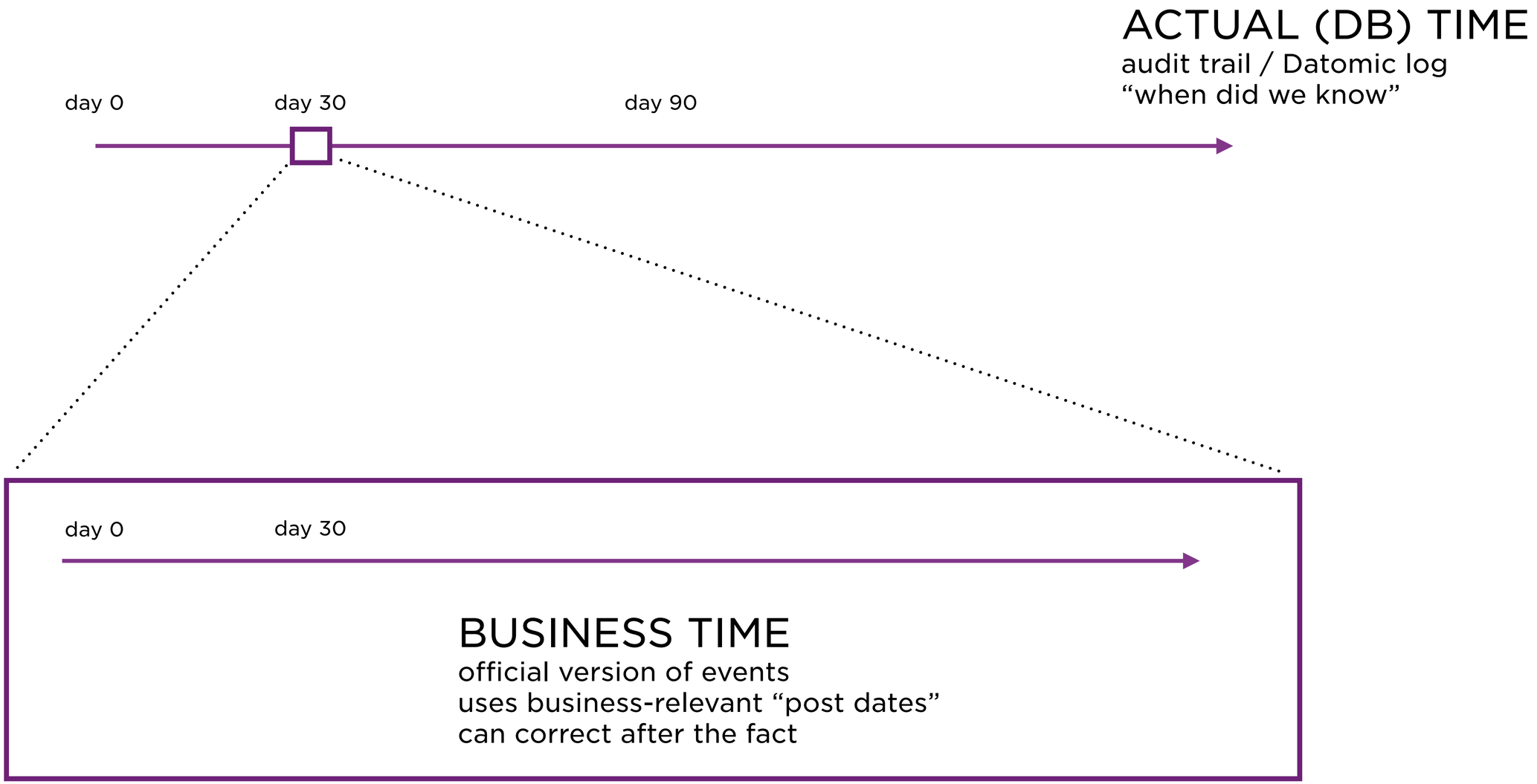
The Result

REAL TIME BALANCE SHEET



2015-01 2015-04 2015-07 2015-10 2015-01 2016-03

2 TIMELINES


ACTUAL (DB) TIME
audit trail / Datomic log
“when did we know”
day 0
day 30
day 90
day 0
day 30
BUSINESS TIME
official version of events
uses business-relevant “post dates”
can correct after the fact

WHAT WE LIKE

- **Canonical definition** of our most important numbers
- Financial analysis applied at a **the customer level in real-time**
- **Inconsistency traceability** allows us to react to it
- Business-specific **invariants provide safety**
- **Generative testing** finds real bugs
- **Ability to replay history** for a customer without losing data
- **Shardable** by time and by customer
- **Extensible** to other products (some don’t require stateful approach)

THANK YOU!

nubank.com.br/jobs

vitor@nubank.com.br

@ura1a

https://gist.github.com/ura1a to get snippets of our domain!