FINASTRA

IMCS LONDON 2018

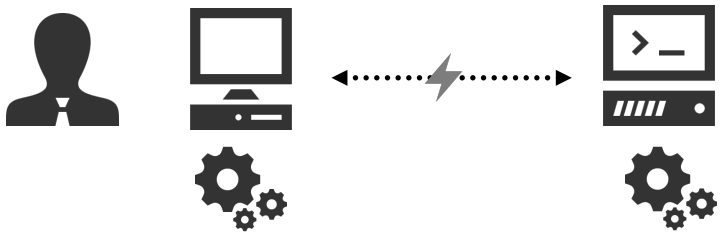
Pricing tale @ Finastra

Romain Gilles
Dev Manager

25 June 2018

WHERE WE WERE

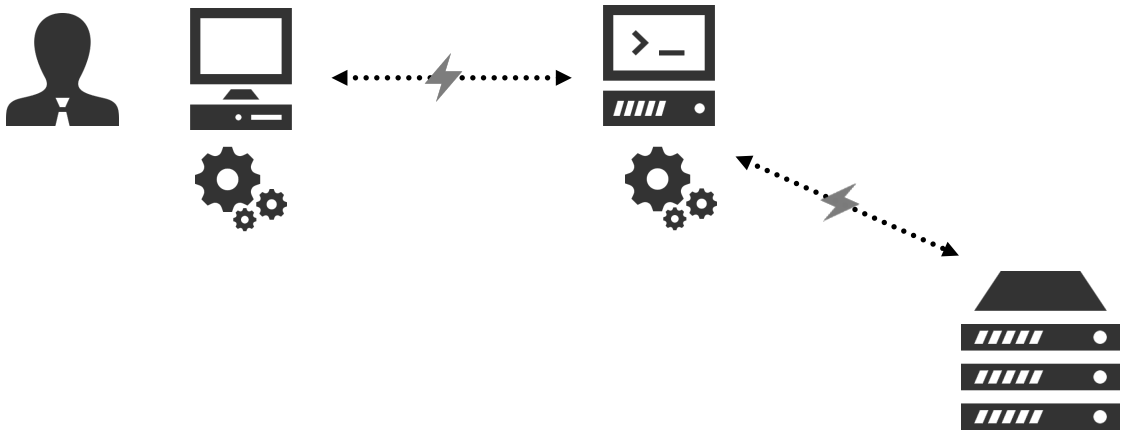
30 to 20 years ago



- Historical **2-tier architecture** with thick C/C++ client and RDBMS servers
- **Model driven** solutions i.e. schema on write. Data can fit into a single big server
- **Expensive servers** to achieve hardware resilience
- Reports from couple of **minutes** to couple of **hours** even more

WHERE WE WERE

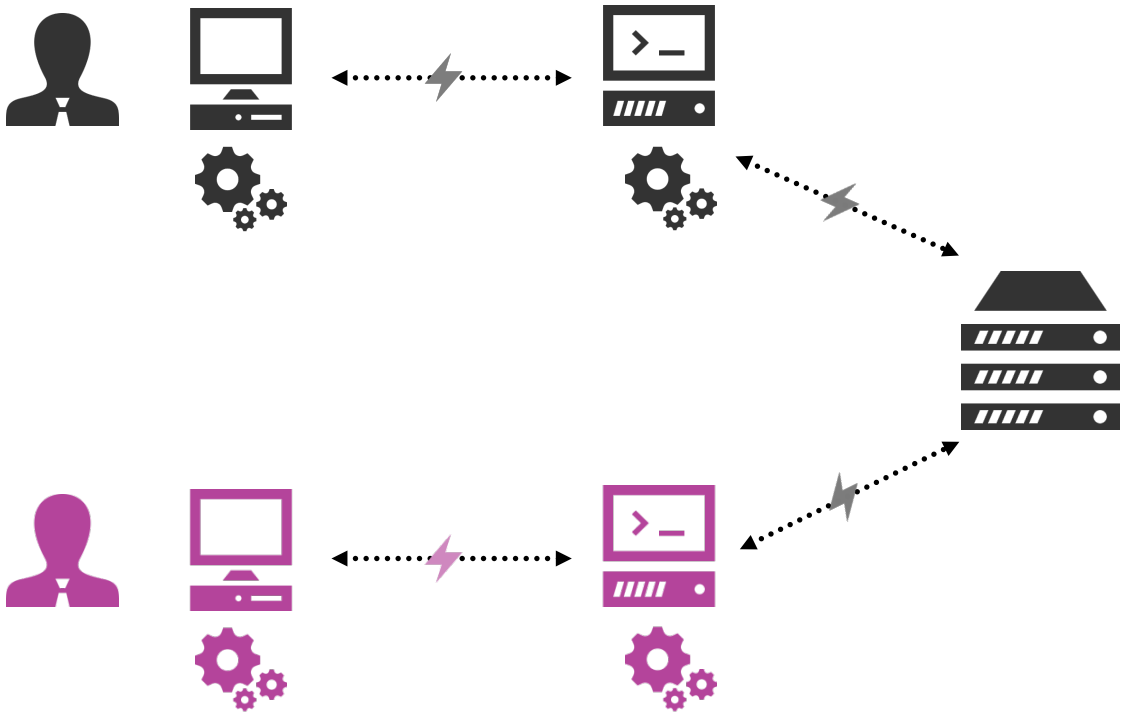
15 to 10 years ago



- **3-tier** architecture
- More **complex** data and computation do not scale up anymore
- Computations move to **compute grid**. But they are heavy data consumers
- Network and RDBMS start to become the **bottleneck**. Introduction of caching level

WHERE WE ARE

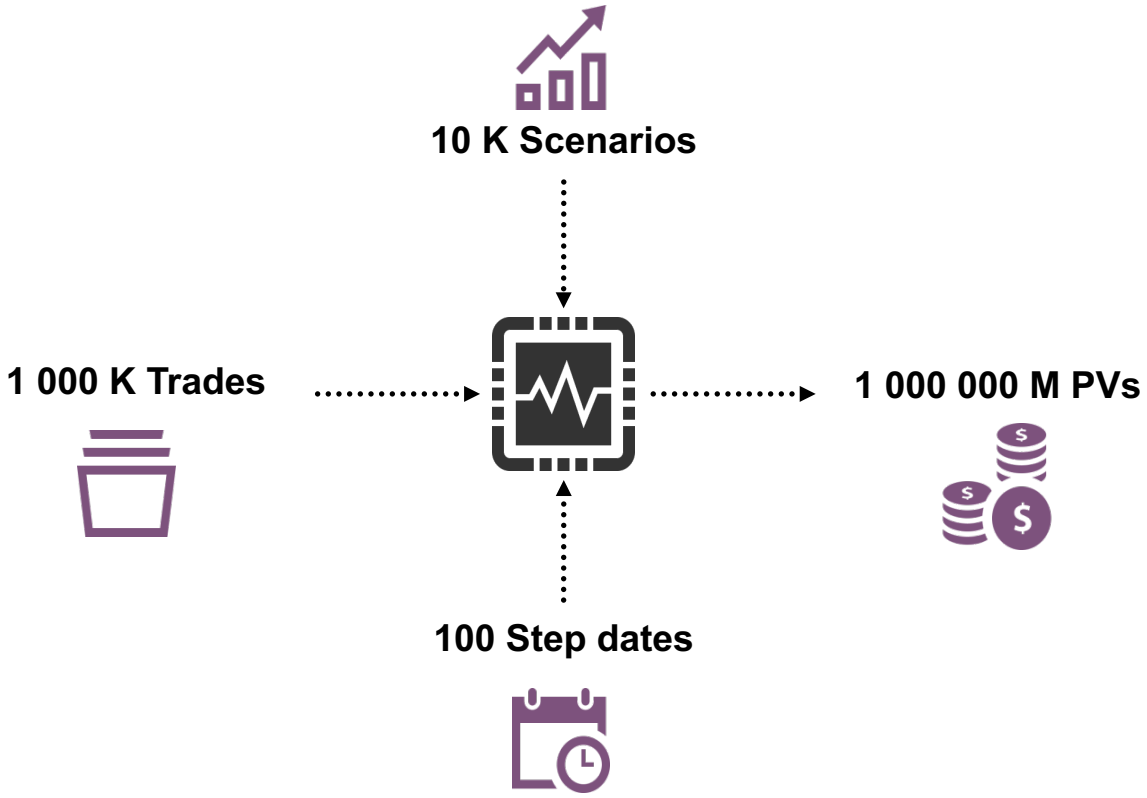
now



- Customers have **different providers**. Some of them claim to be the strongest in one domain. They want to **compare results**
- Finastra is a **mix of dozen** of Front, Middle, Back Office and Retail Banking solutions
- Now what about **consistent cross provider** reports

BUSINESS DIMENSION

The PFE example



- Pricing computation can be simplified by 3 **orthogonal inputs**
- The **trade** is the main data who constantly increase
- **Scenarios** are potential values of the market data
- **Step dates** are projection dates

WHAT WE WANT TO ACHIEVE

Business target

~10% of Daily Trade Finance

25% of US Wire payments

$5T Assets Under Management

43% of Total Syndicated Loans

~8% of Daily FX Trading

30% of UK faster Payments

175M+ Retail Accounts

~25% of all U.S. Financial Institutions

- **Scale** with the **data**
- **Scale** with the continuous increase of **computation** needs
- Be '**fast**' on large reports and be '**faster**' on small ones
- Support multiple **heterogeneous** sources of data
- Maintain report in **soft real time**

PROBLEMS TO SOLVE

FINASTRA

Data

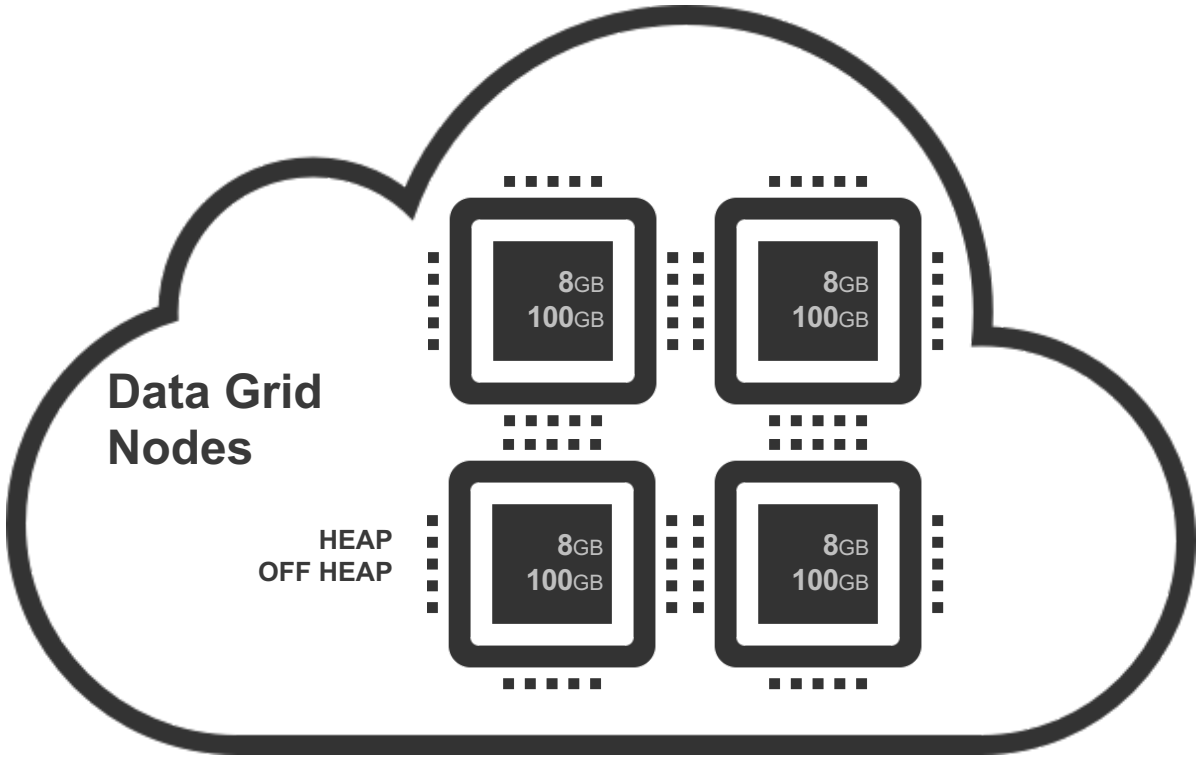
Multi-sources integration

Computation

Fast reports

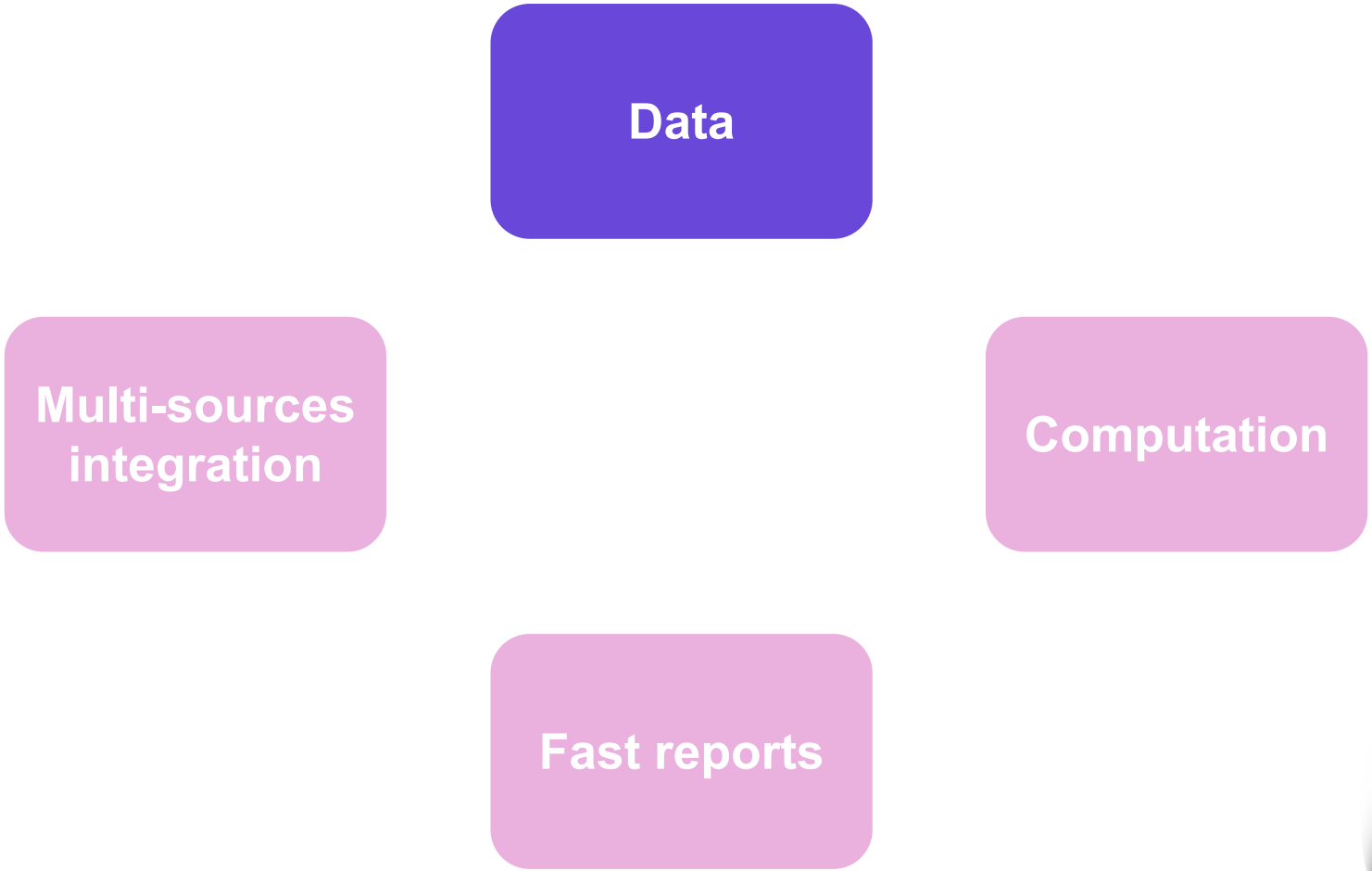
SCALE THE DATA

Ignite distributed cache



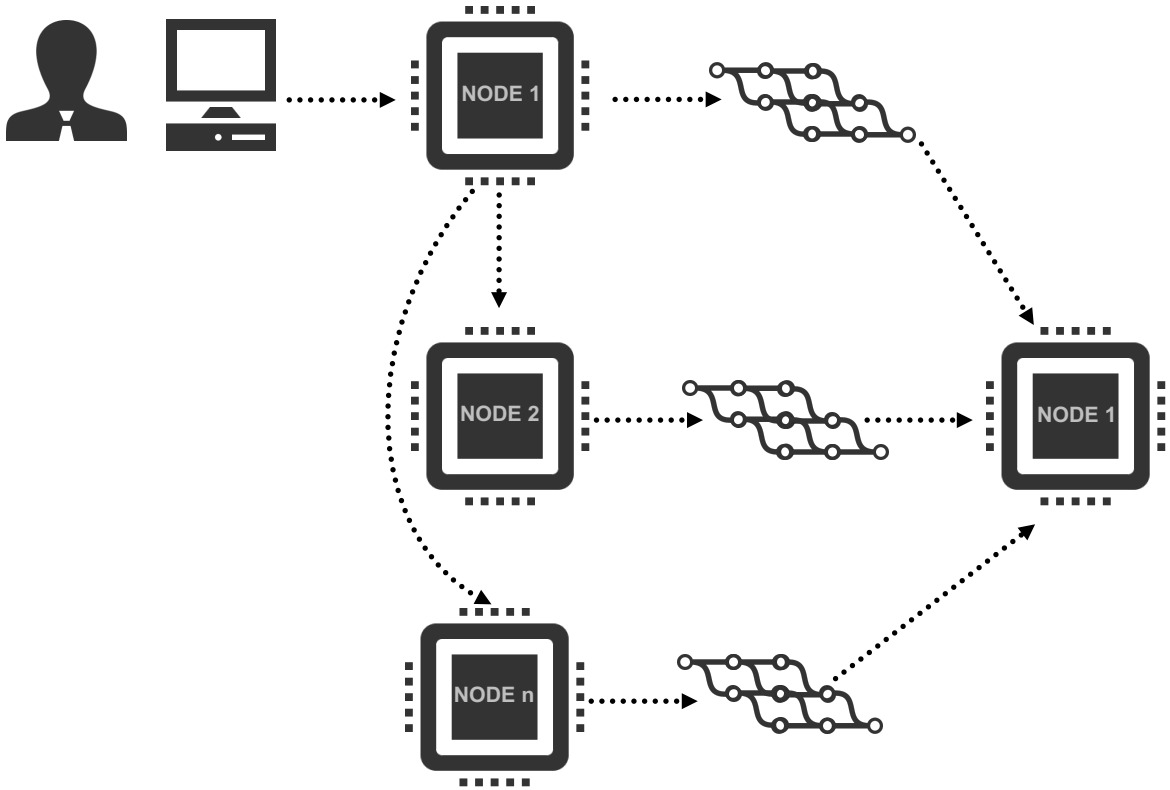
- **Categorize the data** from quantity, update frequency and usage, to identify cache mode
- **Trade data** is updated frequently, its size increase permanently and is pivot of the computation: **Partitioned** cache
- **Static data** is stable and in small quantity, used everywhere: **Replicated** cache
- **Market data** is big, frequently updated, used everywhere: **Partitioned with Near** cache
- Use **Off Heap** everywhere

PROBLEMS TO SOLVE



SCALE THE COMPUTATION

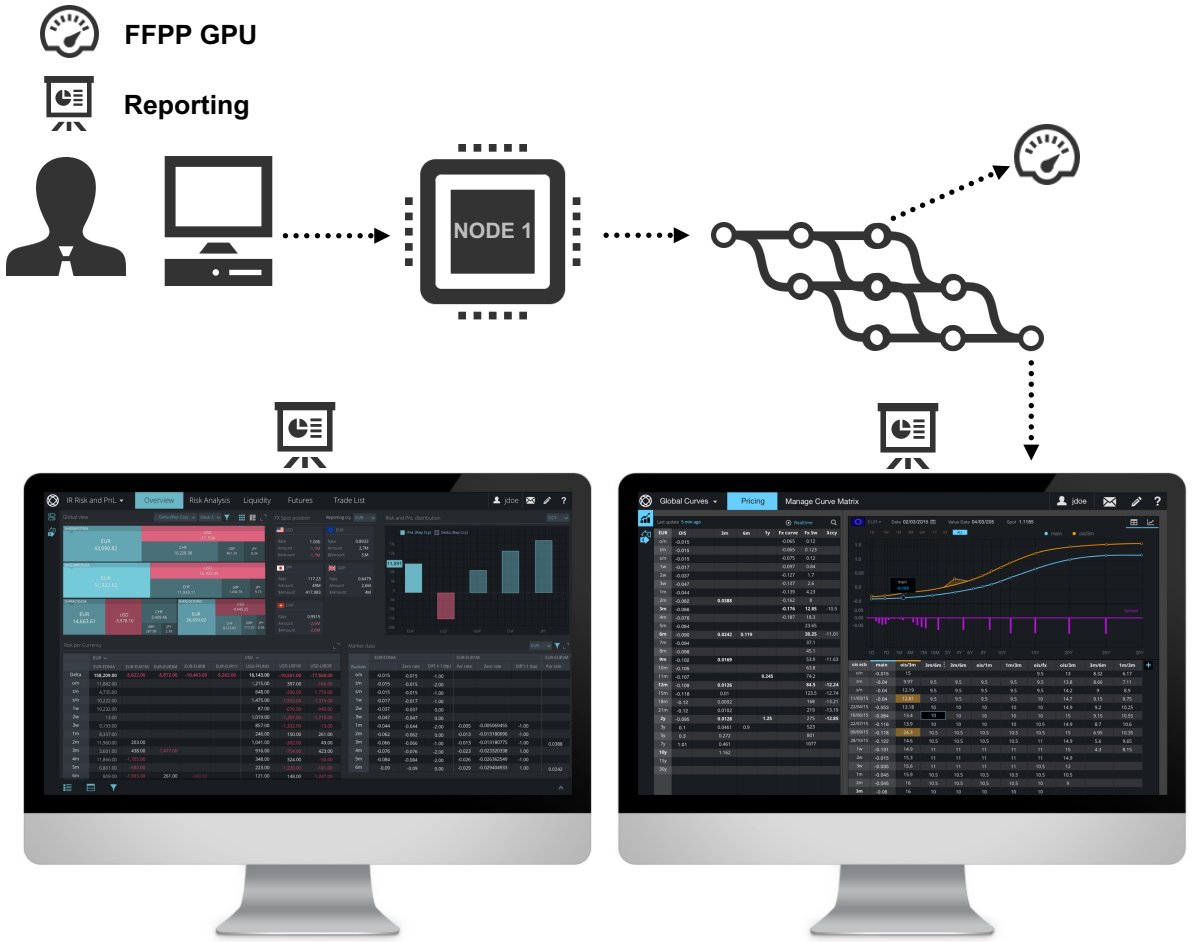
Collocate data and computation



- **Distribute computation through Ignite SQL queries or Continuous queries**
- **Collocate data and computation: sends the code near the data**
- **Complex business logic that requires a lot of data**
- **Streaming allows to reuse the same business logic for on-demand and soft real-time reports**

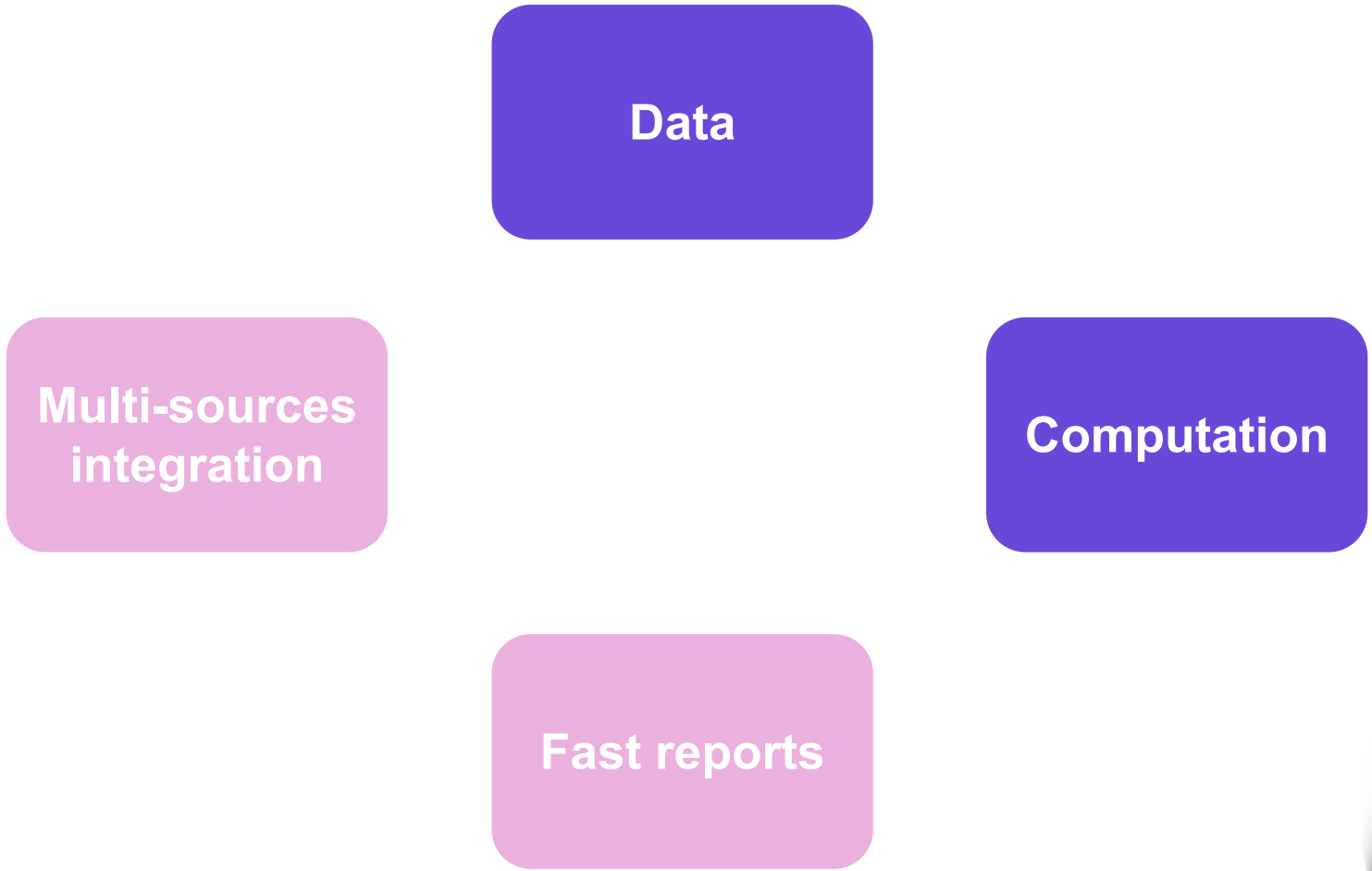
SCALE THE COMPUTATION

Collocate data and computation



- We are using **streaming** for computation
- Processor is the unit of **reusability**
- **Docflow**: compose processor unit in to a DAG assembled with an EDSL
- **Parallelize** processing by preparing data for pricing, push the pricing to the **GPU** collect data for future aggregation, push everything to the **column store** based reporting stack

PROBLEMS TO SOLVE

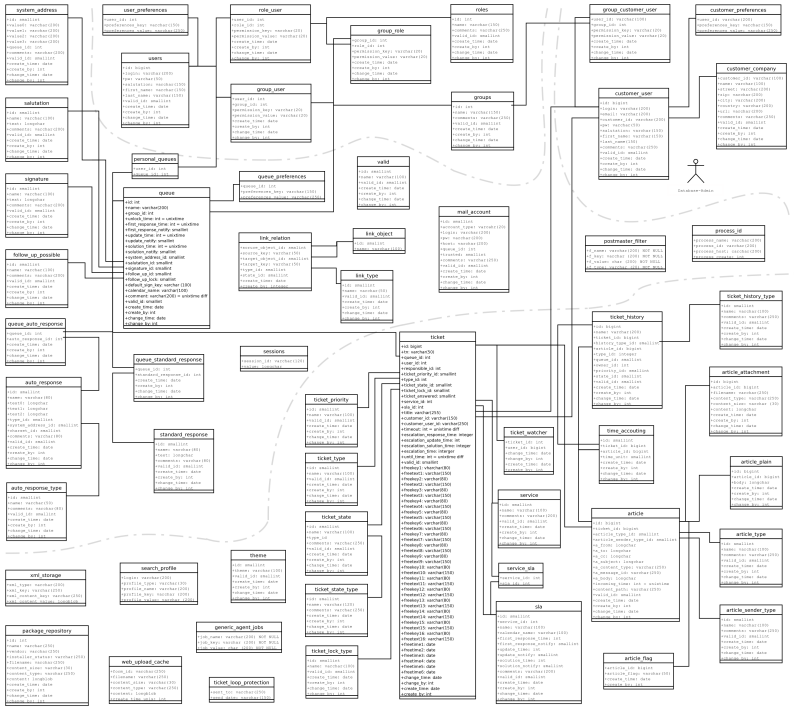


PIVOT

Schema on write

?

Consumer



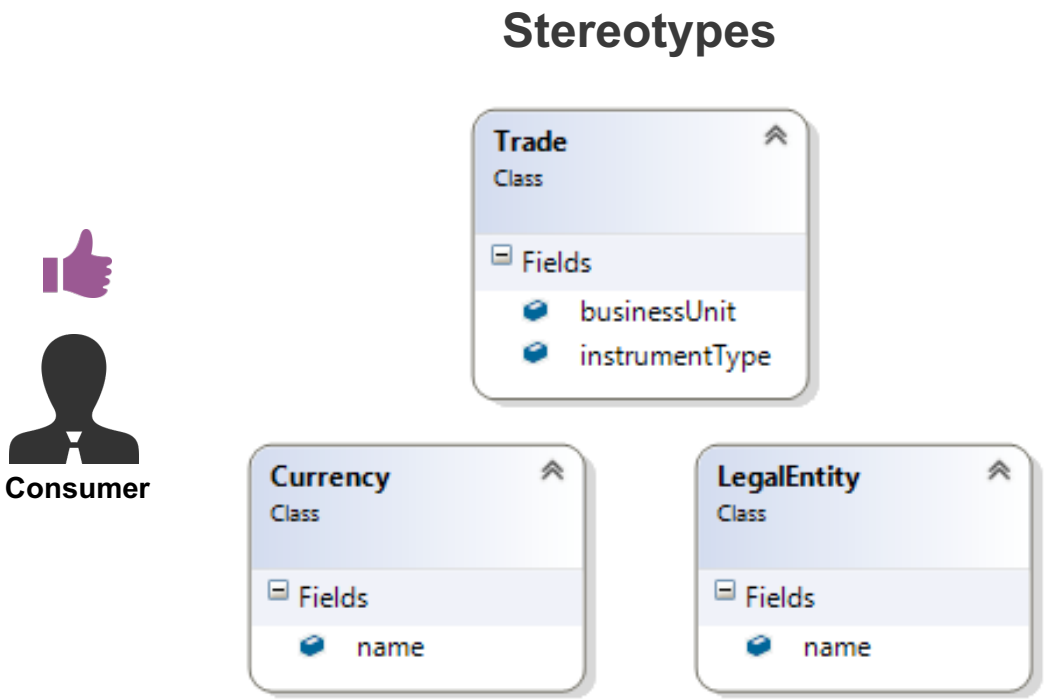
?

Provider

- As we had **strong schema** in the past, we tried to continue by defining a **pivot model**
- Model **consumers** don't know what to use
- Model **providers** don't know what is required
- It introduces a lot of **coupling** and doesn't scale in development: bottleneck effect

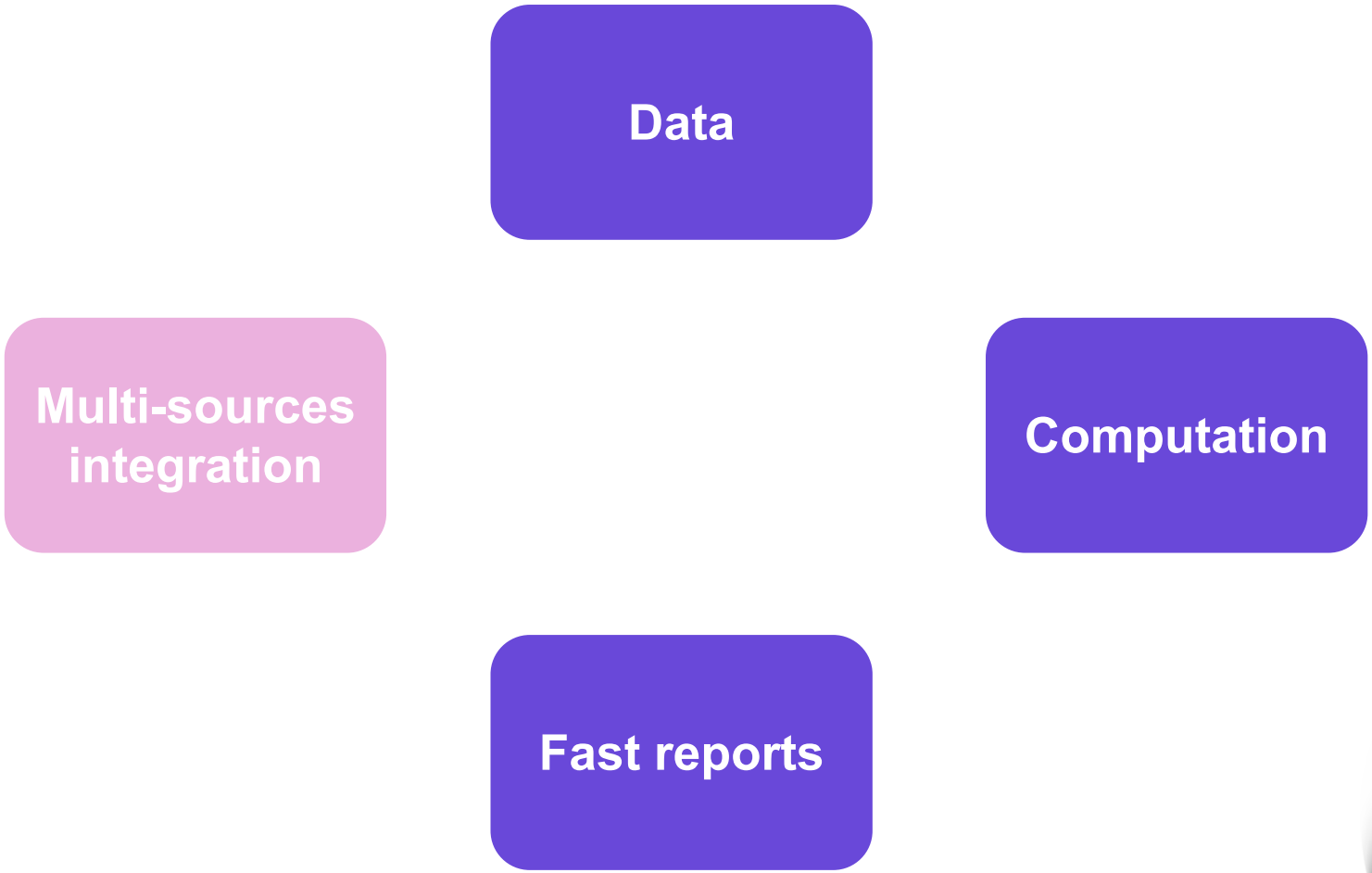
FAST FOR LARGE AND FASTER FOR SMALL

Lightweight schema on write a.k.a. Stereotype



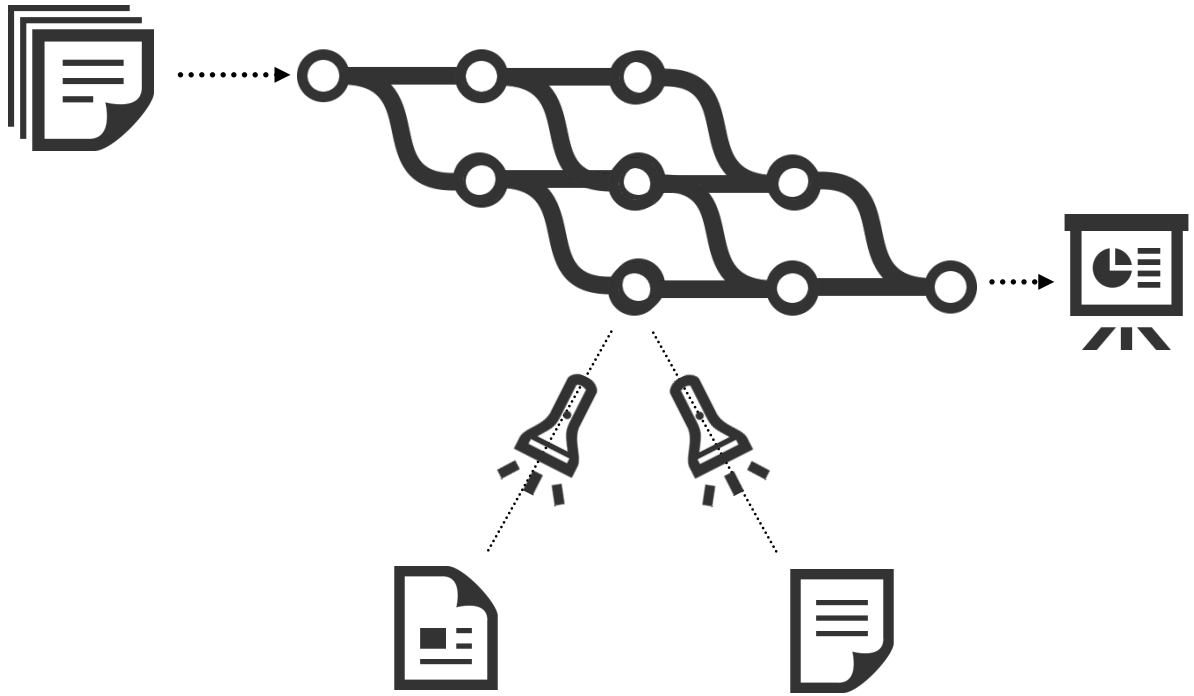
- **Lightweight schema** on write: a.k.a. **Stereotype**
- Only for query. First level of filtering
- **Minimize contention** on development
- Ensure **tradeoff** between query and development performance
- **Simplify** usage and integration
- Allow **indexing**

PROBLEMS TO SOLVE



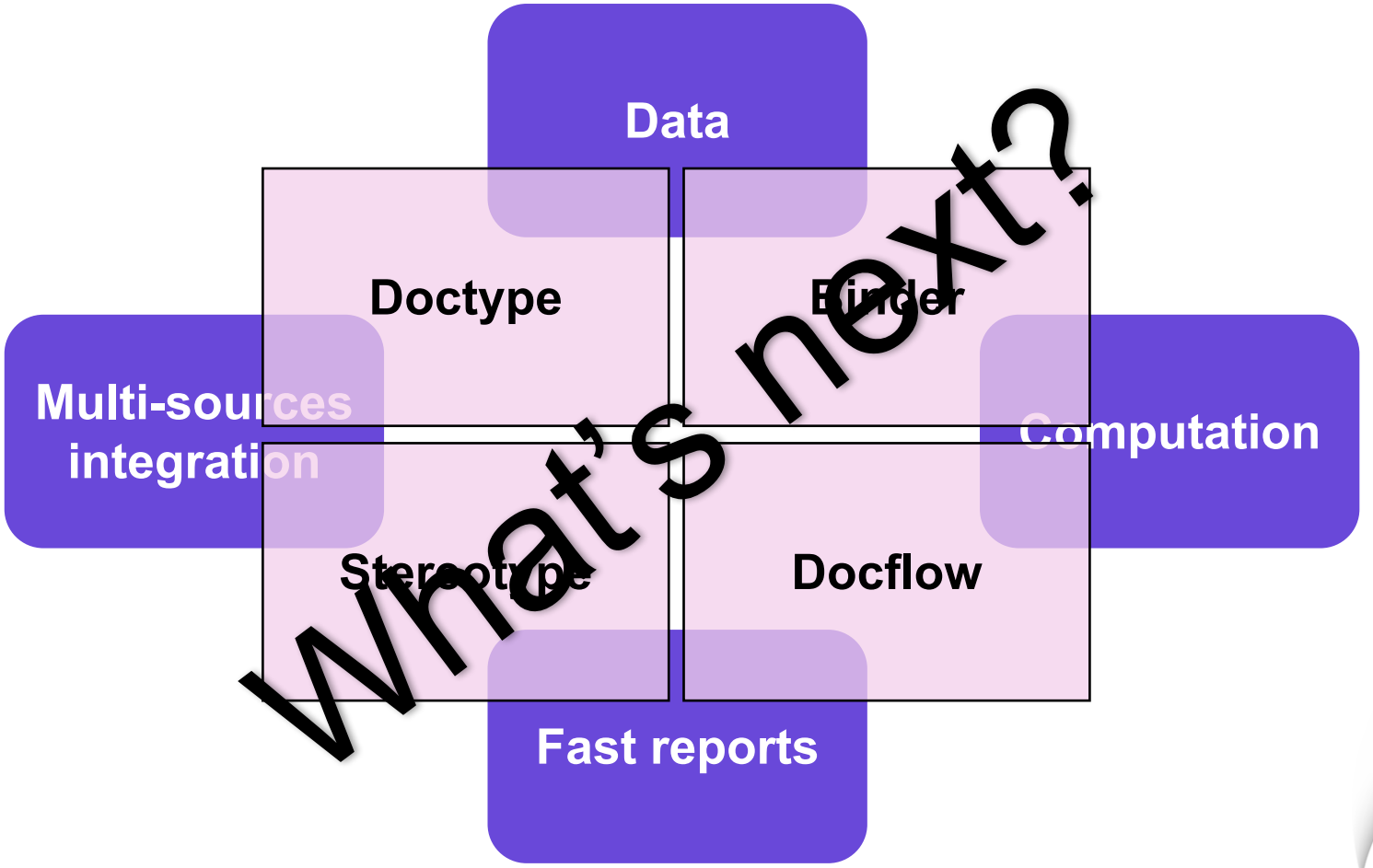
MULTI HETEROGENEOUS SOURCES OF DATA

Schema on read Sushi principle



- Processor functions define their **views**
- Data is **adapted** to the views through **binders**
- Framework calls binders when processor function request a view. This ensure **independency** from the underlying documents and improve **testability**
- Does not block future evolutions

PROBLEMS SOLVED



FINTECHS

WANT BETTER
COLLABORATION

BANKS

WANT FASTER
INNOVATION

FINTECHS

BANKS

COLLABORATION IS THE NEW INNOVATION

FINTECHS

BANKS

Announcing...

FusionFabric.cloud

A new platform for collaboration and innovation

FusionFabric.cloud - CONNECTING FINTECHS AND BANKS

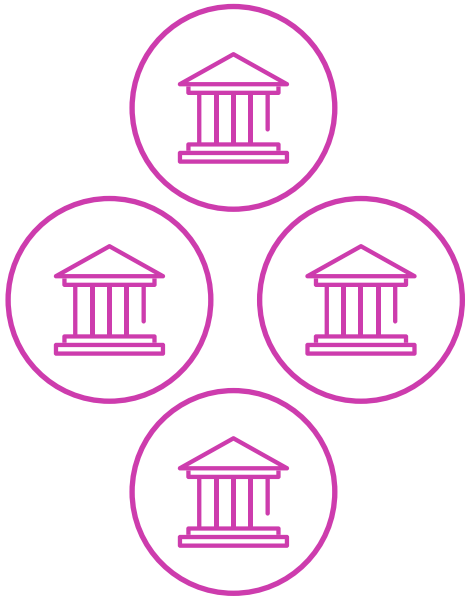
CREATORS
Fintechs, Banks,
ISV's, Universities

FusionFabric.cloud

FusionCreator
Create applications in low-code environment

FusionOperate
Deploy and manage applications on Microsoft Azure

FusionStore
Consume and monetise applications



CONSUMERS
Banks and FI's

Core systems accessible through public REST APIs managed within FusionCreator

Retail Banking	Lending	Transaction Banking	Treasury and Capital Markets	Core Systems or Technology

FINASTRA CORE SYSTEMS | 3rd PARTY

FusionFabric.cloud - THE PLATFORM COMPONENTS

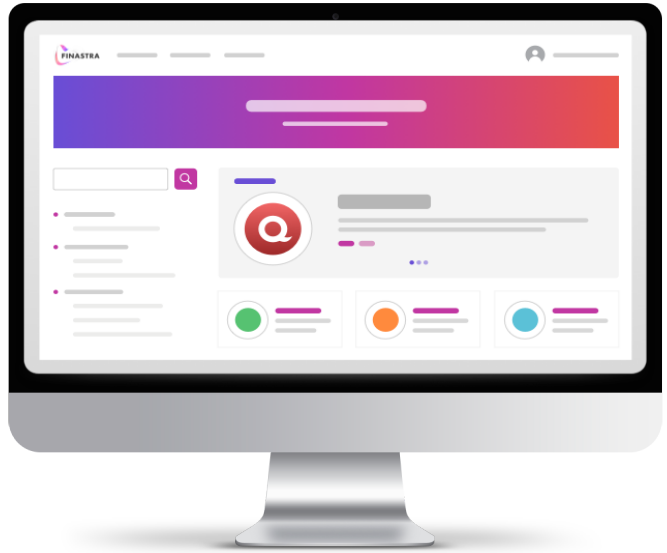
FusionCreator

- Visual low-code development
- REST API Management
- Sandbox test environment
- Full developer toolbox

FusionOperate

- Application deployment
- Powerful dashboards
- Secure data access
- Based on Microsoft Azure

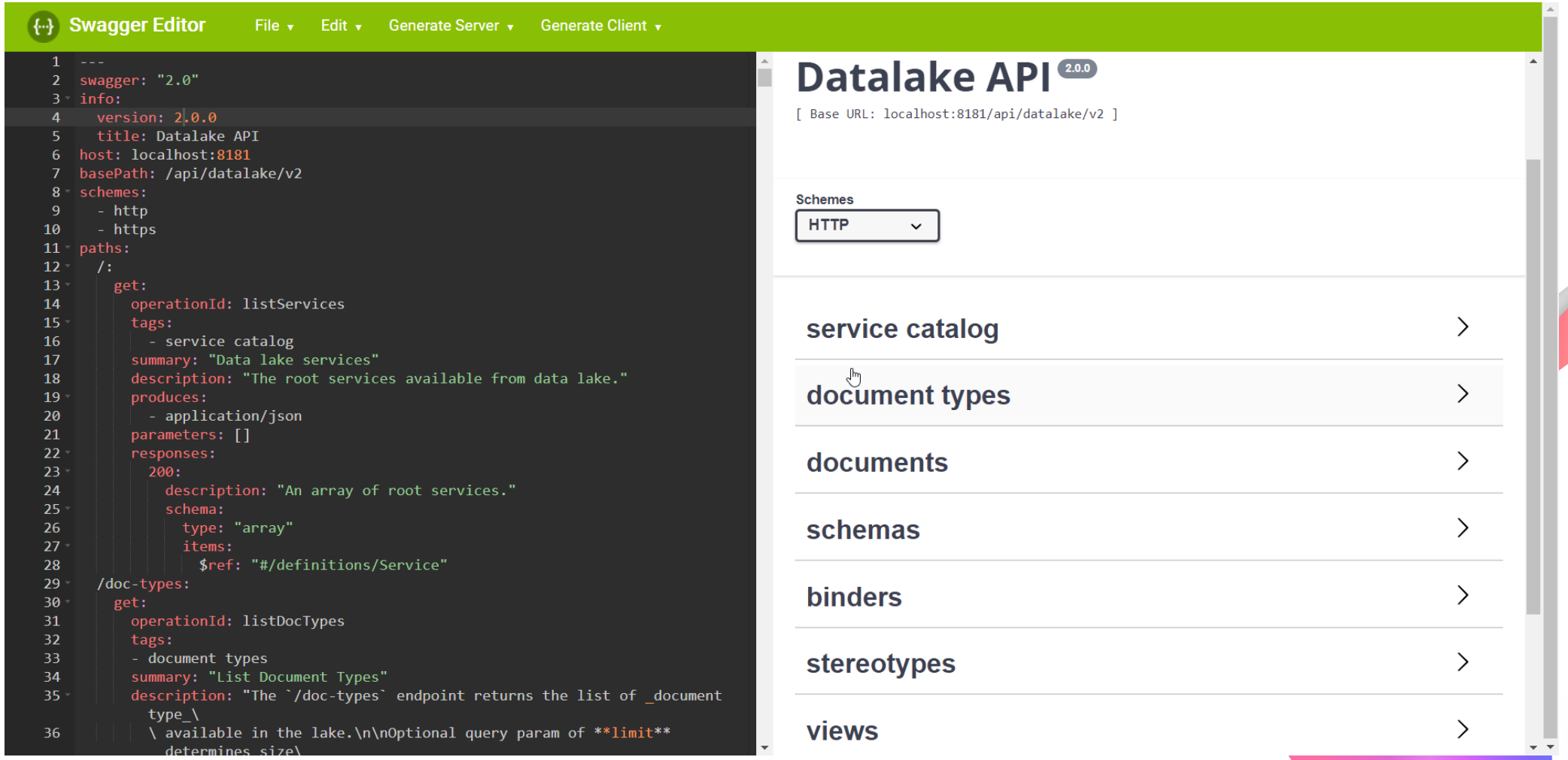
FusionStore



- Application distribution
- Invoicing & Payments
- Quality check
- Promotion & Marketing

DATALAKE

Dynamic RESTful API for customization



LESSONS LEARNED

From the battlefield

FINASTRA

- **Affinity** your best friend or… Affinity can lead to issues.
- **Fight with garbage** collocated data and computation is dangerous because of GC pressure and GC pause.
- **Understand your graph** is a key part, why is it so slow: **Open tracing**
- **Cultural change** RDBMS to document store. No more relation but composition. Key part for performance.
- **Contract first**. Test REST API from the generated client code.

LAUNCH VIDEO

Thank you

Romain Gilles
Dev Manager

romain.gilles@finastra.com


@FinastraFS


Finastra LinkedIn


Finastra YouTube