Build your Operational Data Layer with MongoDB

How to optimize your legacy stores to be prepared for the future

mongoDB

Agenda

- Actual Situation
- Problems
- Legacy Optimization
  - Data Lake
  - Operational Data Layer
- How to Implement an Operational Data Layer
- The future

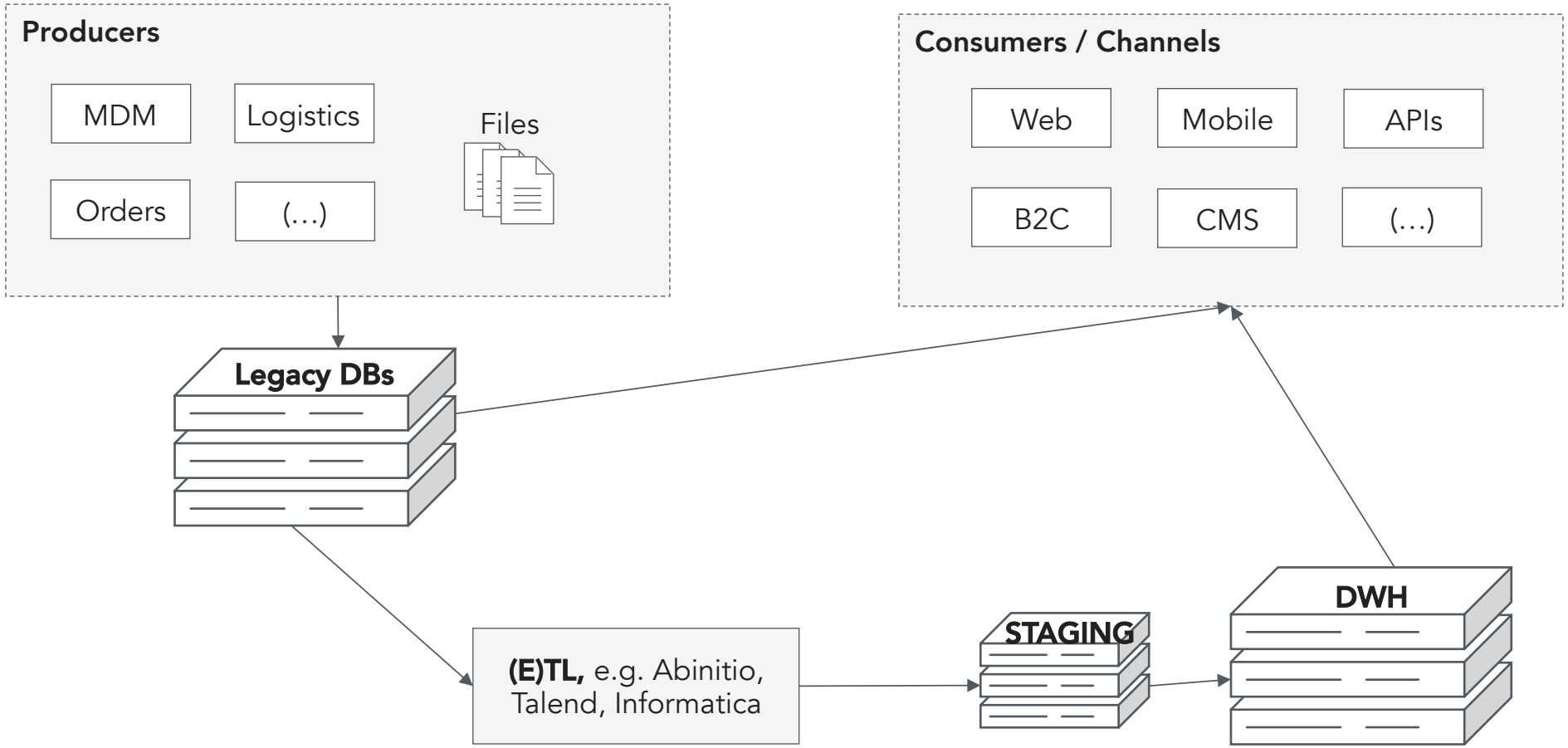
How Much Data?

- One thing is not missing to the modern enterprises: data
  - IoT, Sensors data
  - Social Sentiment
  - Server Logs
  - Mobile Apps
- Analyst estimate a growth of data volume of 40% y2y, 90% of them will be unstructured.
- The legacy technologies (some of them designed more than 40 years ago) are not sufficient anymore.

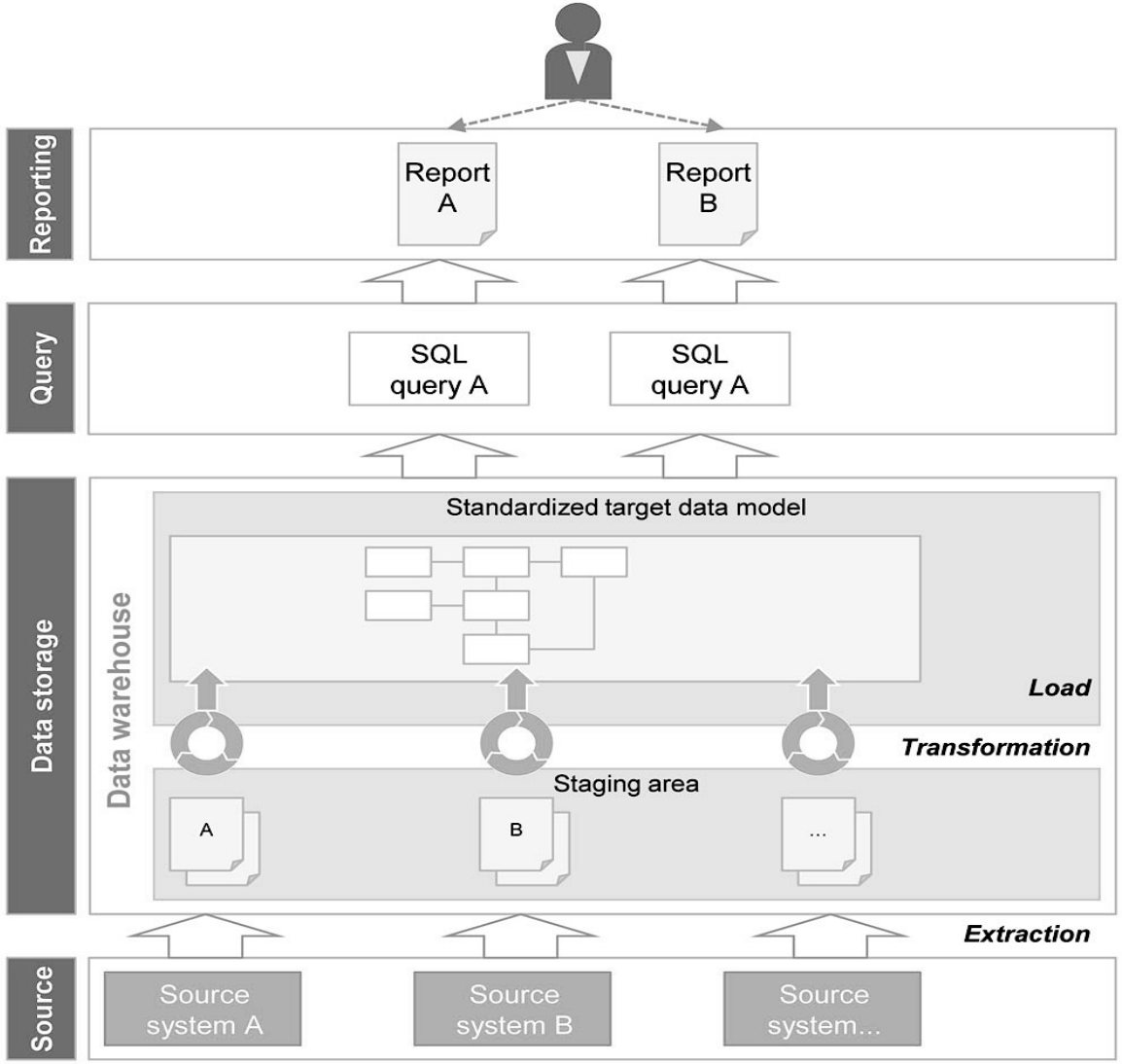
The “Big Data” Promise

- To discover information collecting and analyzing all the data brings 2 things:
  - Competitive Advantage
  - Cost Reduction
- A common example is the usage of big data technologies to build a “Single View”: to aggregate all what is known of a client to improve the engagement and revenues
- The traditional EDW doesn’t support the load, struggling due to the volume and variety of data (and high cost).

Reference Architecture



Typical DWH Structure



mongoDB

Limits

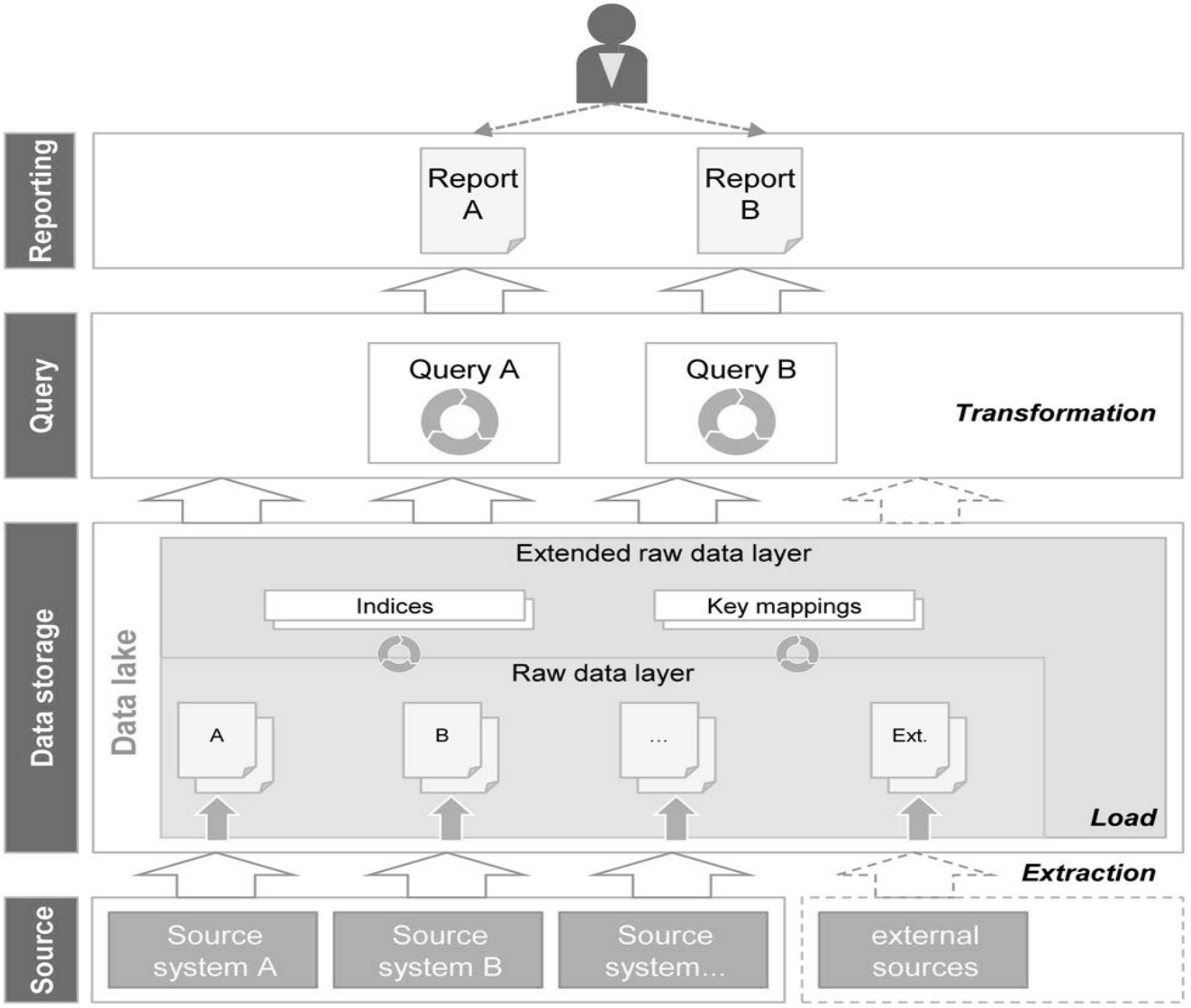
- High Implementation Effort
  - Transformation and standardization in a harmonic data model is the DWH heart.
- Rigidity
  - The data model is pre-defined and rigid and is difficult to expand it to integrate additional external sources
- No Raw Data.
- Data Volume
  - DWH can manage high volume of data but with dedicated databases and optimized hardware.

mongoDB

Rise of Data Lakes

- Many companies started to look at a Data Lake architecture
  - Platform to manage data in a flexible
  - To aggregate and put in relation data cross-silos in a single repository
  - Exploration of all the data
- The most common platform is Hadoop
  - Allows horizontal scalability on commodity hardware
  - Allows storing heterogeneous data with a read optimized model
  - Include working layer in SQL and common languages
  - Great references (Yahoo e Google)

Struttura del Data Lake



mongoDB

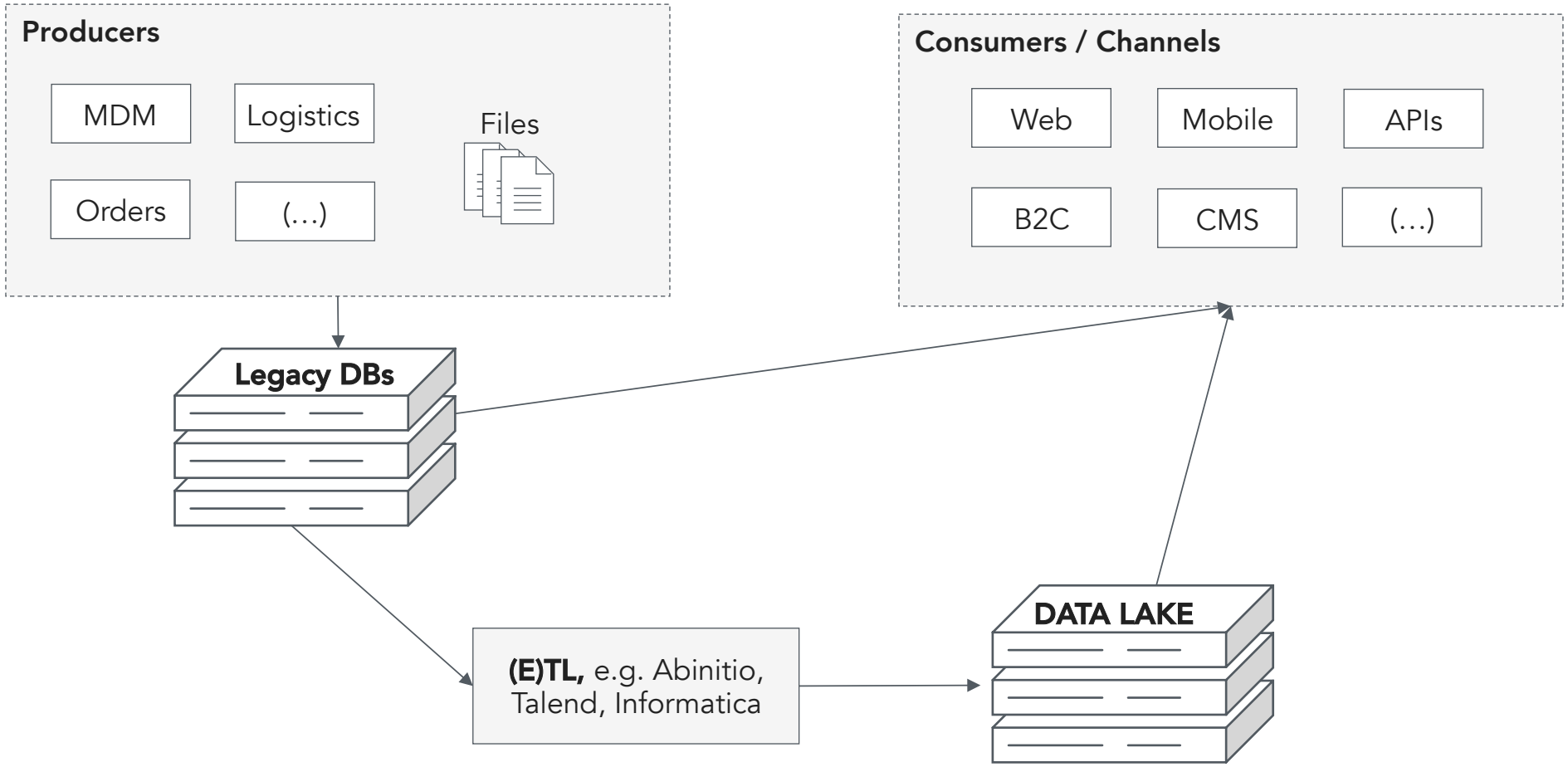
How is Working?

- Source data are loaded as they are in a raw data layer without any transformation
- The Technology is not based on a RDBMS but on a file system (HDFS di Hadoop for example)
- The queries are executed directly on the raw data and are much more complex to write since must contains also the logic of harmonization and consolidation of data
- The add of the source of information is quite easy

Why Hadoop?

- Hadoop Distributed Filesystem is designed to scale on huge batch operations
- Provides a write-one read-many append-only model
- Optimized for long scan of TB or PB of data
- This capability to manage multi-structured data is used:
  - Customers segmentations for marketing campaigns and recommendation
  - Predictive Analysis
  - Risk management

Reference Architecture / 2



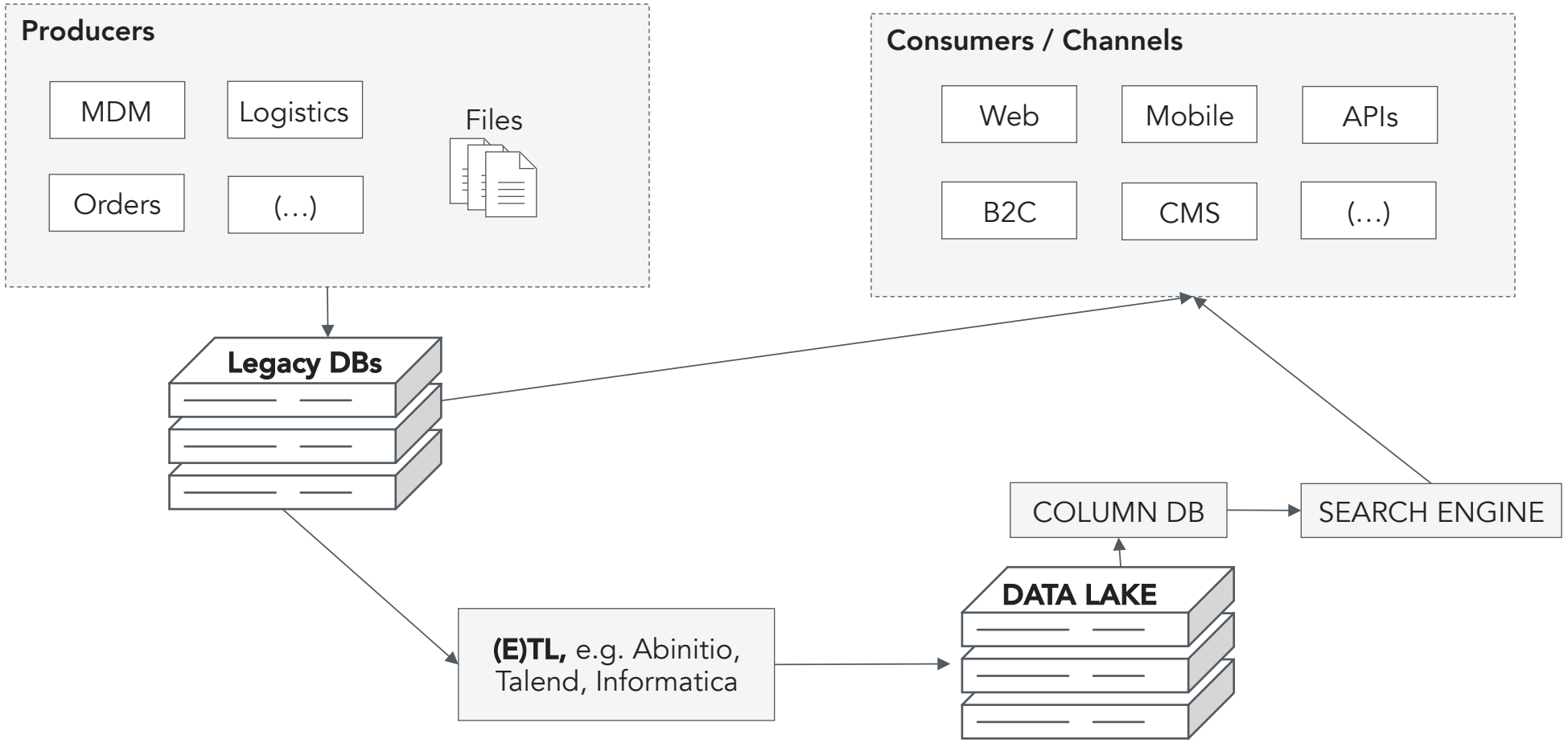
Hadoop can do everything?

- In our data-driven world, the milliseconds are important
  - IBM researchers state that 60% of data lose value after few milliseconds after they are created
  - For example identify a fraud stock exchange transaction is useless after some minutes.
- Gartner predicts that 70% of Hadoop installations will fail because will not reach the cost reduction objectives and revenues increase

The Data Lakes can do everything?

- Data Lakes are designed to provide the Hadoop output to online applications. These applications can have requirements like:
  - Response time in ms
  - Random Access on small indexed subset of data
  - Expressive queries
  - Frequent real-time updates

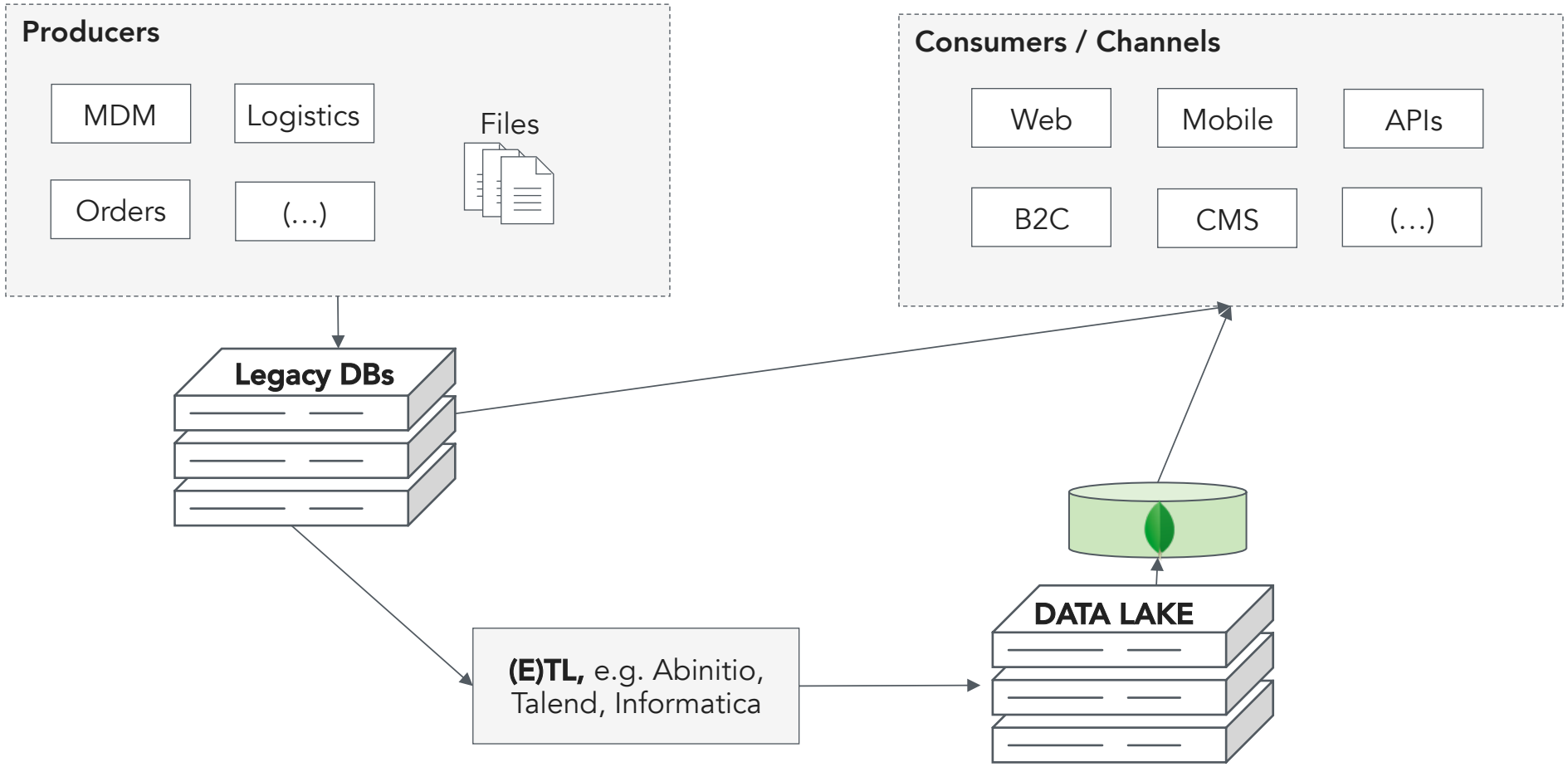
Reference Architecture / 3



Data Lakes with MongoDB?

- MongoDB can solve these problems adding an operational layer that allows to:
  - Maintain a subset of data
  - Execute queries on an indexed portion of data
  - Execute atomic queries
  - Interact with BI tools
- Hadoop Integration con Hadoop
  - MongoDB has a connector to interact with HDFS

Reference Architecture / 4



Issues

- Data Model
  - Is designed for Analytics not for the applications
- Not Real-Time
  - ETL process, long time to traverse all the components
- SearchDB limited in query capabilities
- ColumnDB limited in data model flexibility and HA

Legacy Optimization with MongoDB

Business Drivers

- Extending legacy applications to new channels & users
  - Mobile & web, IoT sensor data integration, single view of data
- Different data access, performance and scalability requirements
- Conserve valuable MIPS, avoid resource contention

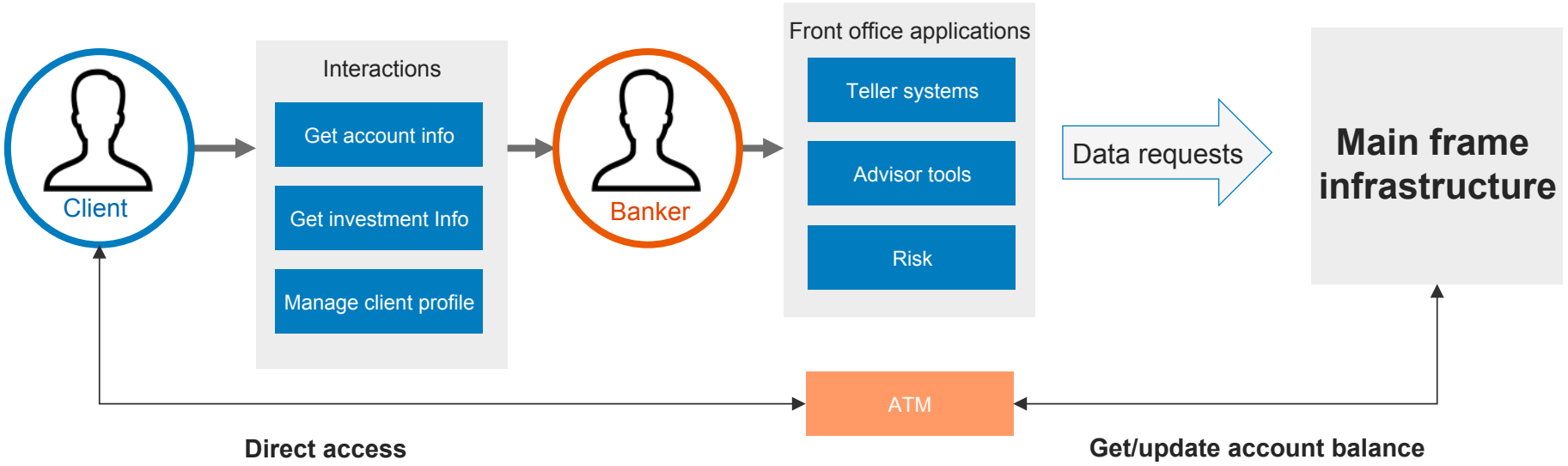
Solution

- Operational Data Store built on MongoDB

Banks Situation: The Past

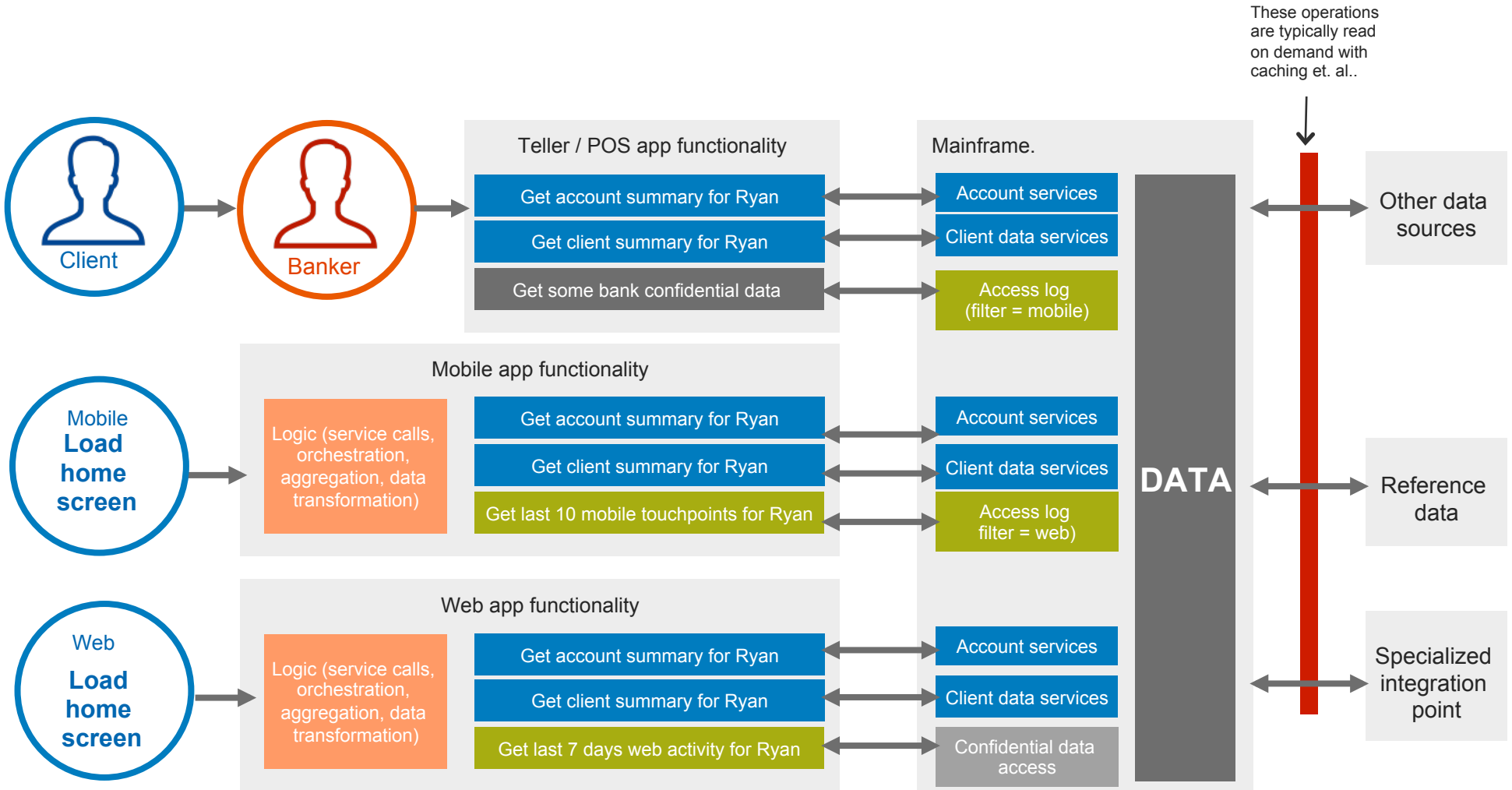
- Banks used mainframe systems to store critical data to that was at the core of their business. Account data, transaction data, other core financial data, etc.
- Internal front office systems were used by bank employees (“Bankers”) to access this critical data and deliver it to clients.
- Critical bank practices (data governance, backup, HA/DR, auditing, etc.) evolved with mainframe systems as a core element of the practices and solutions. Mainframes were the golden source of data before we invented the term “golden source” .

The Past



mongoDB

Present


These operations are typically read on demand with caching et. al..
Client
Banker
Teller / POS app functionality
Get account summary for Ryan
Get client summary for Ryan
Get some bank confidential data
Mainframe.
Account services
Client data services
Access log (filter = mobile)
Mobile
Load home screen
Mobile app functionality
Logic (service calls, orchestration, aggregation, data transformation)
Get account summary for Ryan
Get client summary for Ryan
Get last 10 mobile touchpoints for Ryan
Account services
Client data services
Access log filter = web)
Web
Load home screen
Web app functionality
Logic (service calls, orchestration, aggregation, data transformation)
Get account summary for Ryan
Get client summary for Ryan
Get last 7 days web activity for Ryan
Account services
Client data services
Confidential data access
DATA
Other data sources
Reference data
Specialized integration point

Use Case

Montly UK Payday

Situation

Almost everyone gets paid on the same day here in London (not like in NA), once a month at the end of the calendar month.

Almost everyone in London uses their bank mobile app to check their balance on the same day, once a month.

Technology Changes

People used to have to go to the bank to get your balance but now everyone has a phone on their app they can use to check their balance.

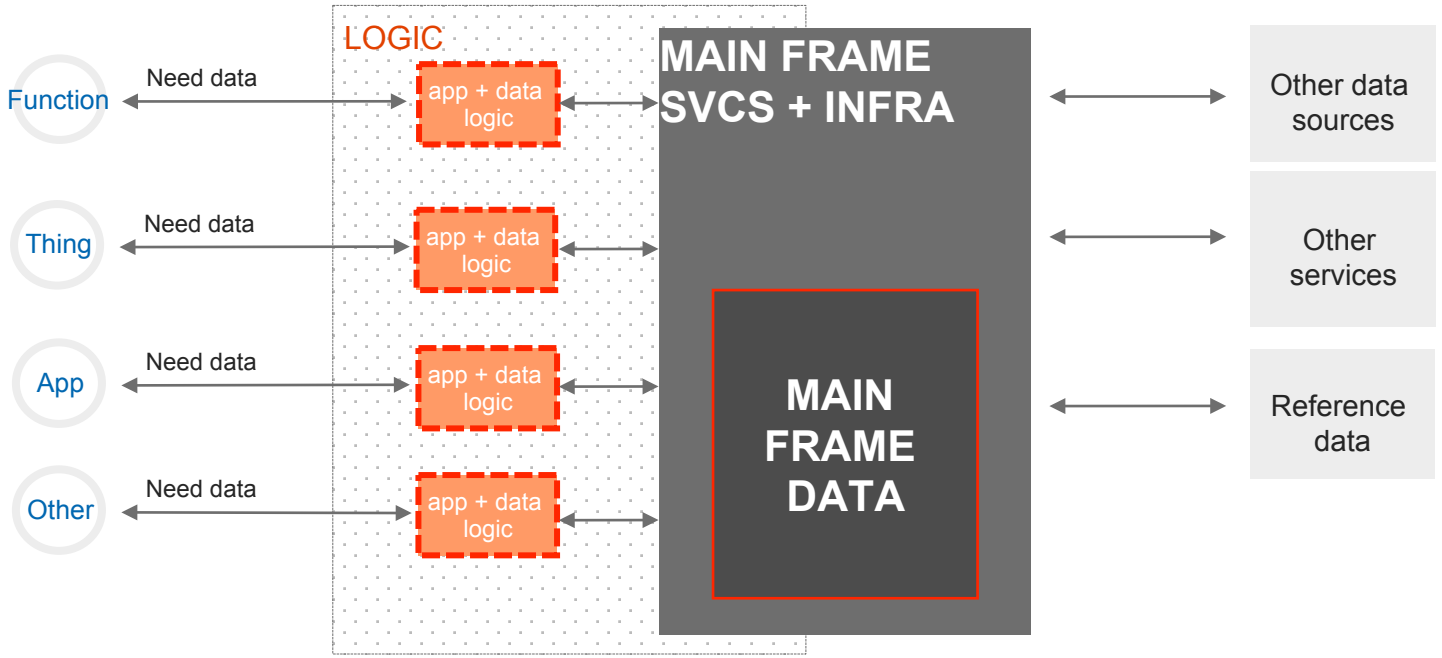
The fact that it's so easy to check bank balance combined with the significant time between paydays means that most people check their balance on payday at least once if not multiple times.

Cost

The mainframe infrastructure to support this spike in traffic due to this use case costs banks millions dollars a year.

An abstract, logical view

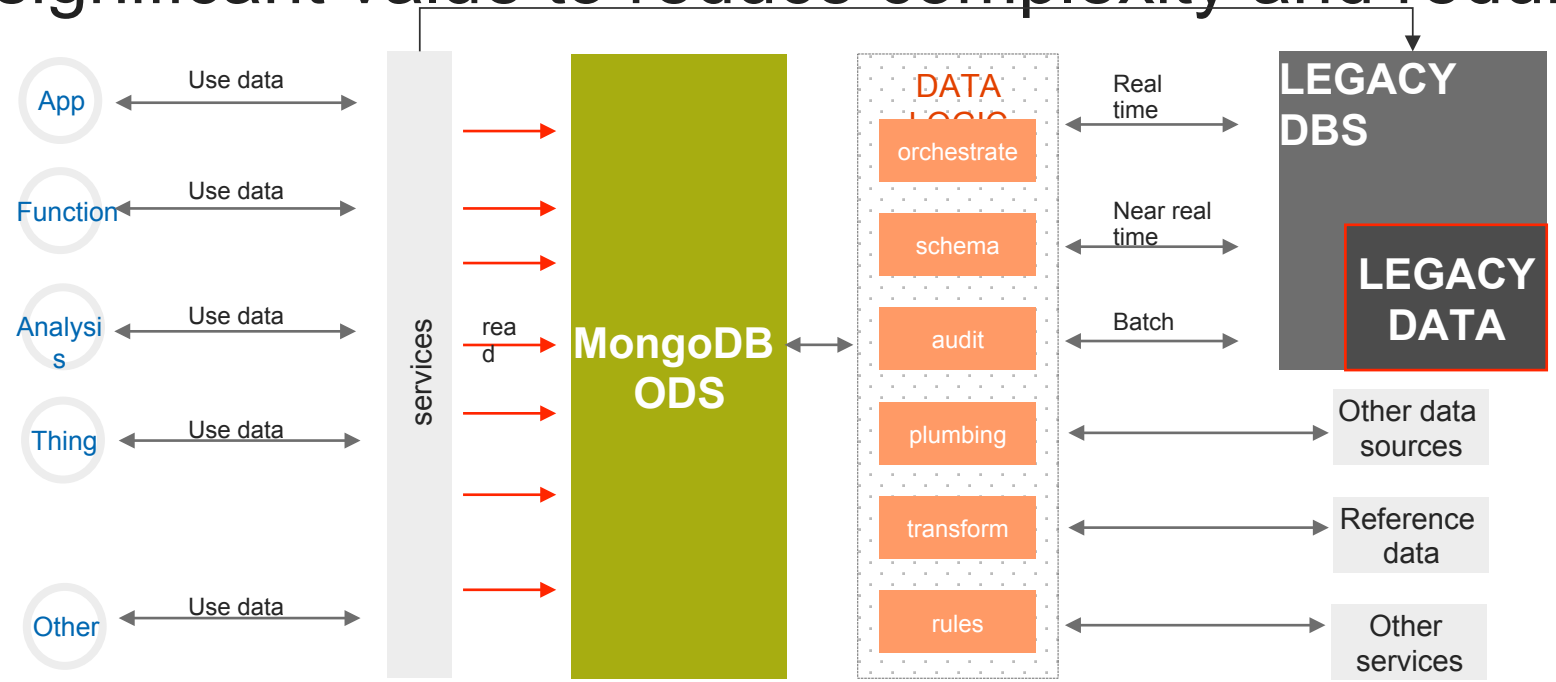
If we strip away some details to investigate a more fundamental, abstract, logical view of the current state, it looks like this.



mongoDB

Some simple changes

moving the app/data logic out of clients and replacing it with data logic between the legacy DB and the MongoDB ODS, we can deliver significant value to reduce complexity and redundancy.



mongoDB

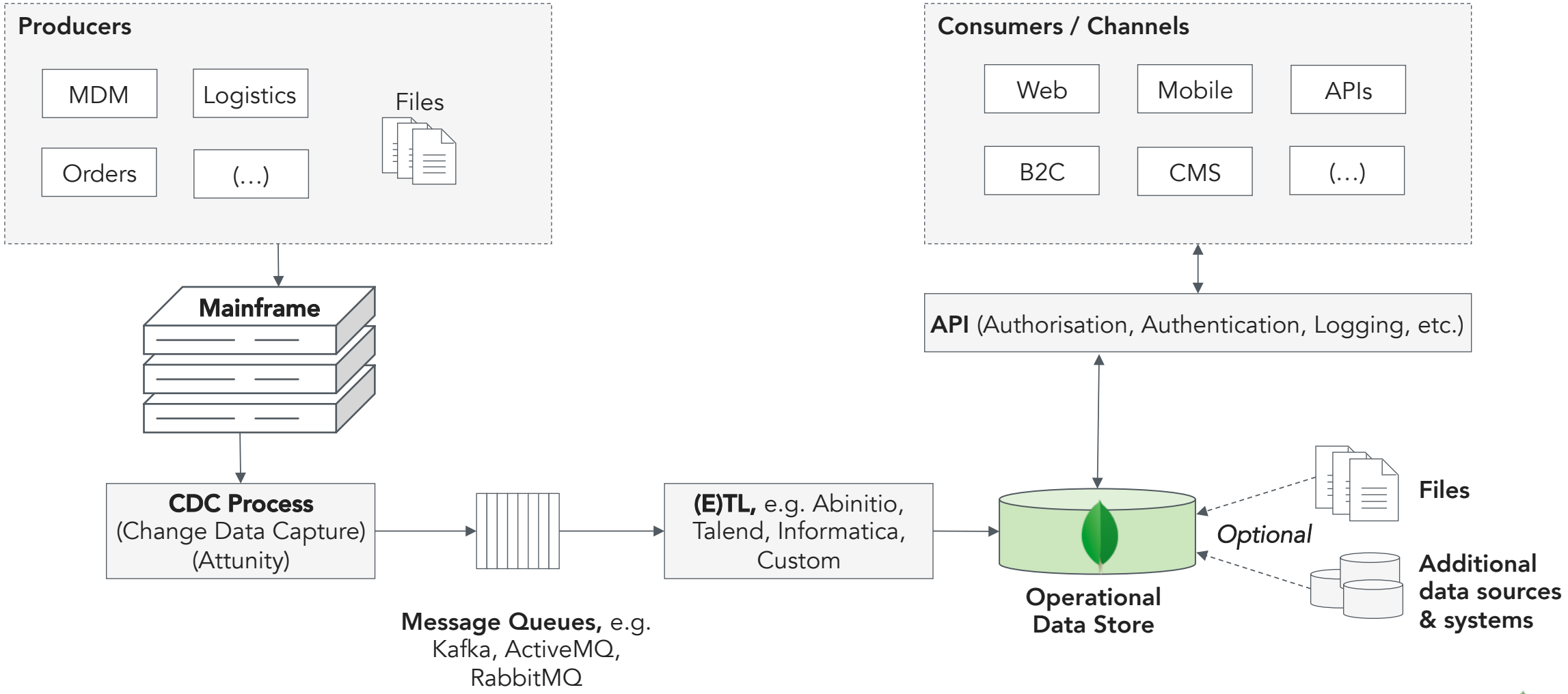
Why MongoDB

- Data Model Optimized
  - The data are transformed during the replication phase to a customer-centric data model, instead of a product-centric one, this is called ”Single View”
- Flexibility
  - The flexible document-based model remove the need to update the schema to store new info
- Architecture for Modern Applications
  - MongoDB has a scalable, geographically distributed architecture for always on services.
- Integration with Analytical Systems

Benefits

- Reduce Development Time
  - The customer-centric data model and the MongoDB flexibility and document data representation reduce by more than 50% the development time
- Improve the SLA
  - MongoDB scalable always on architecture improves the uptime and the performance, providing a better customer experience
- Reduce Costs
  - Move MIPS from legacy to commodity low cost hardware

Reference Architecture



mongoDB

Attunity Replicate

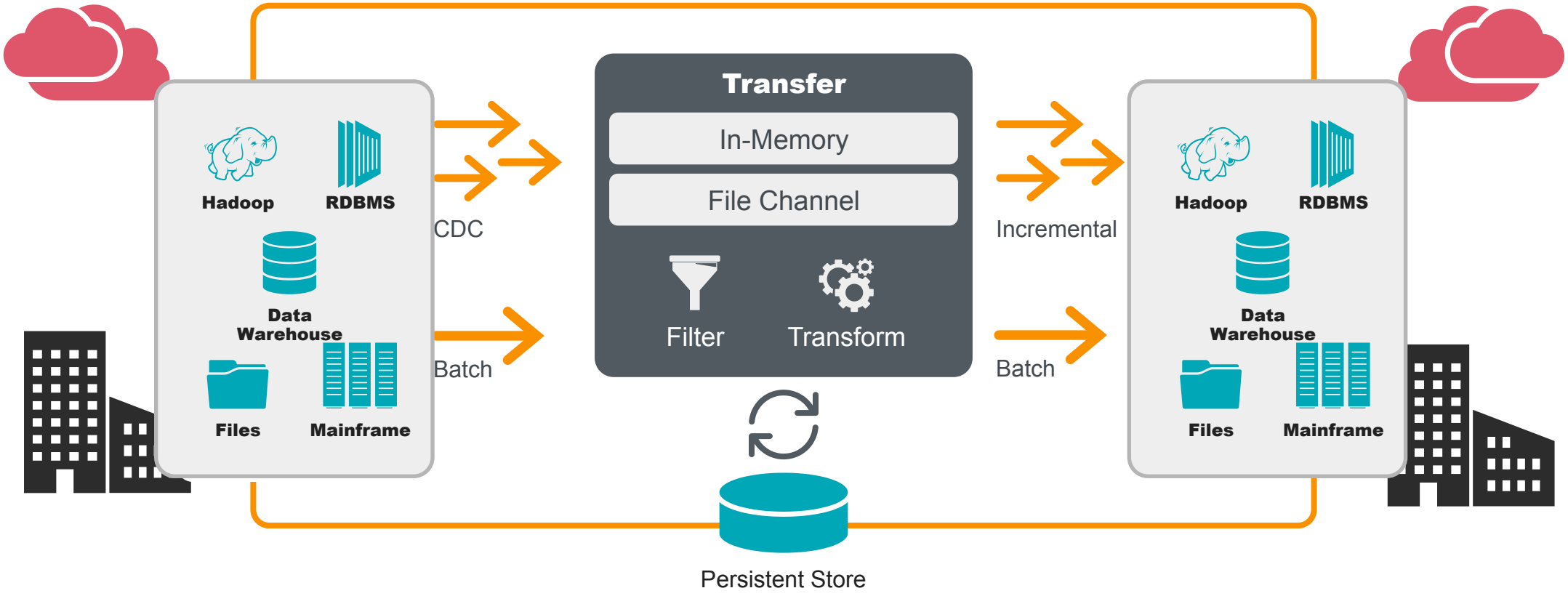
Universal Platform for Data Replication

Simpler + Real-Time + Universal

#1 Independent Provider of Streaming CDC

mongoDB

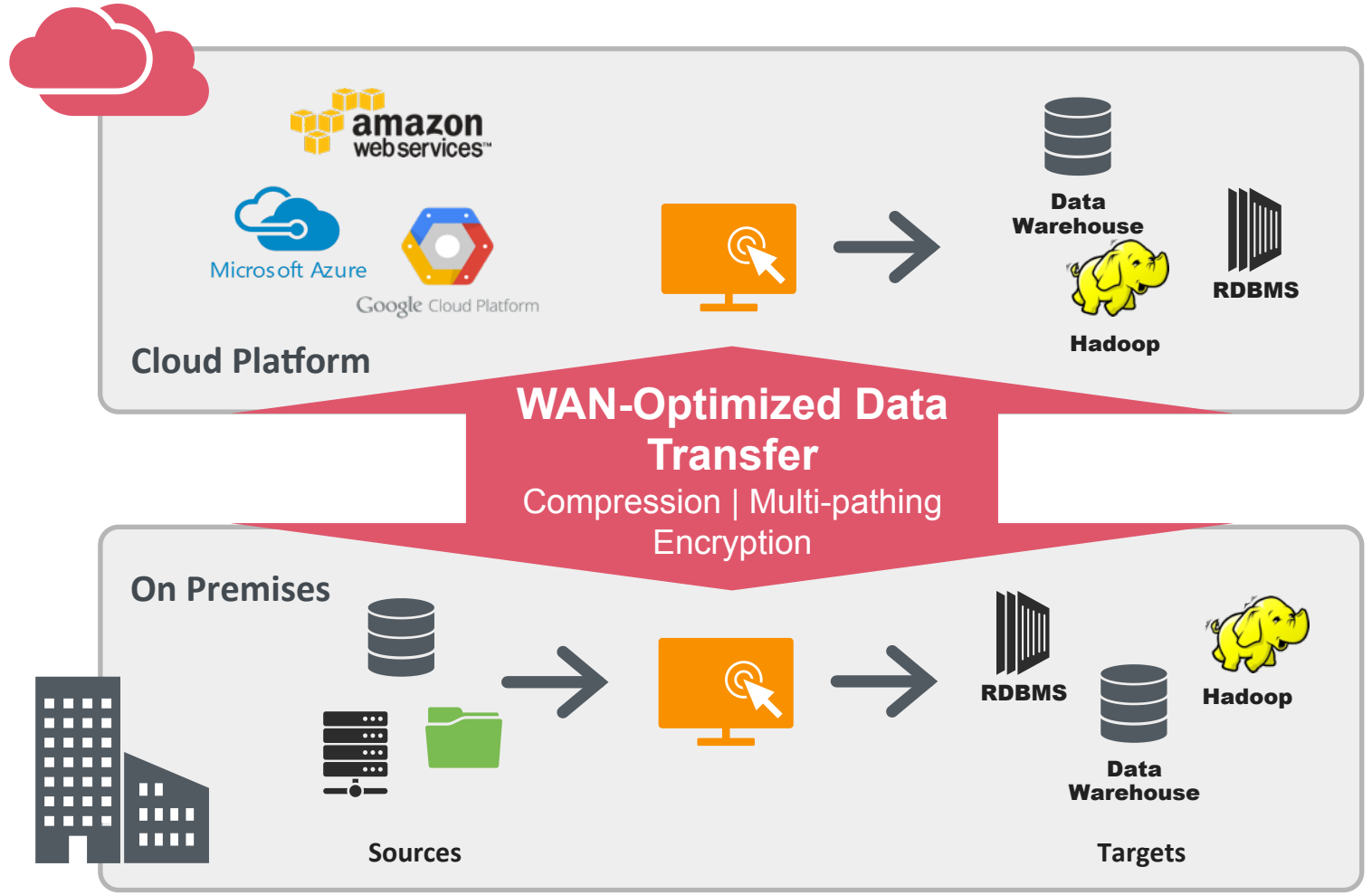
Architecture



mongoDB

Attunity Replicate

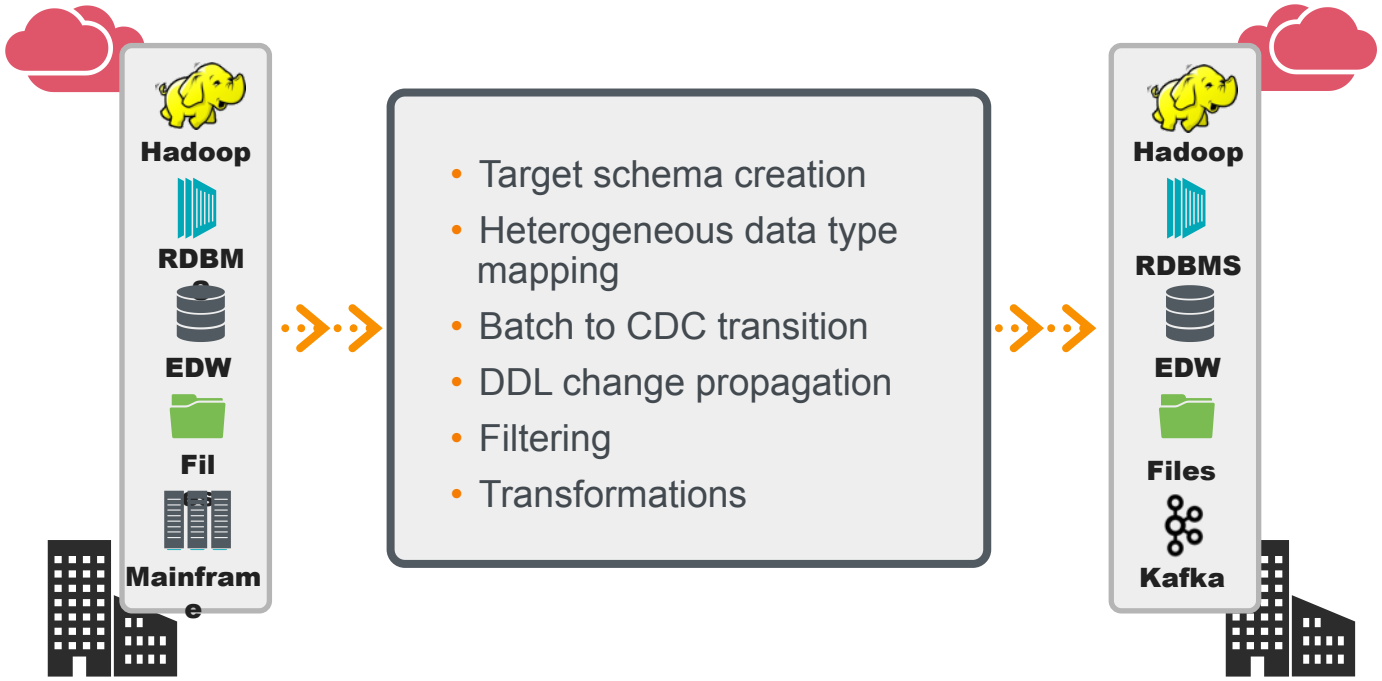
Rapidly Move Data Across Complex Hybrid Environments



mongoDB

Attunity Replicate

Go Agile with Automated Processes

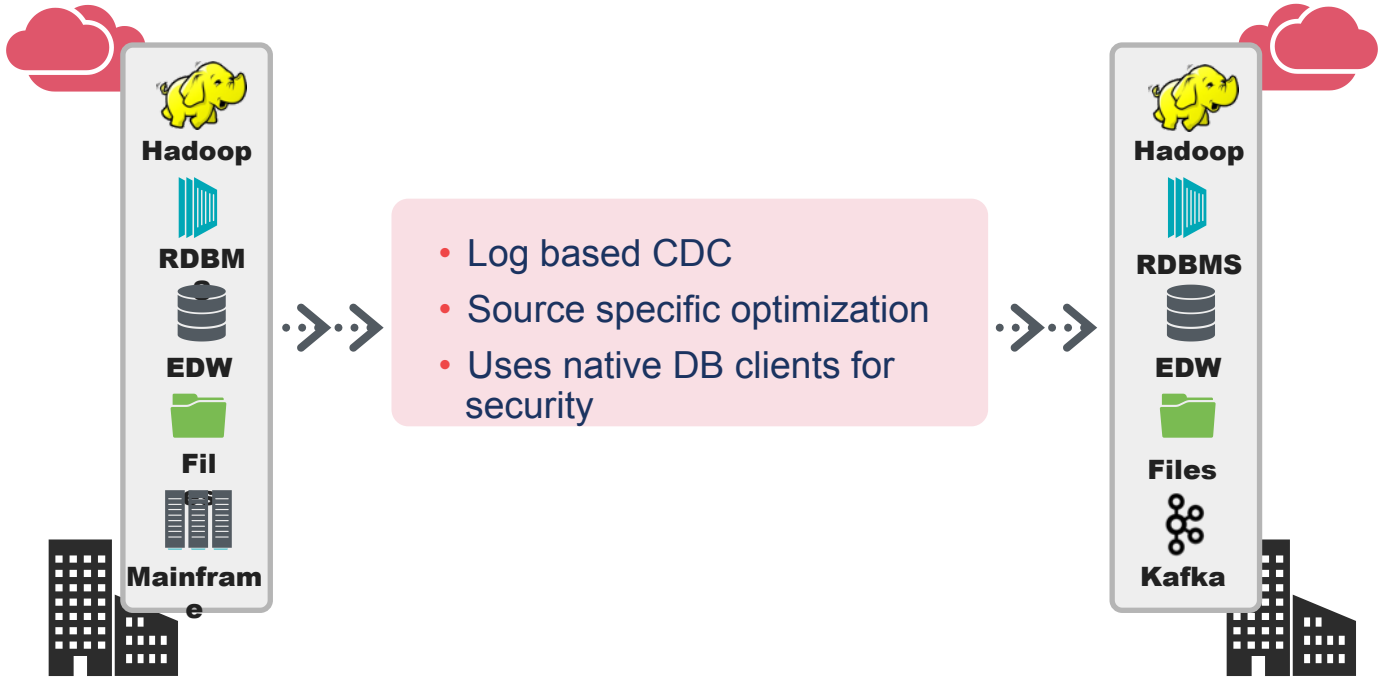


Attunity Replicate

Zero Footprint Architecture

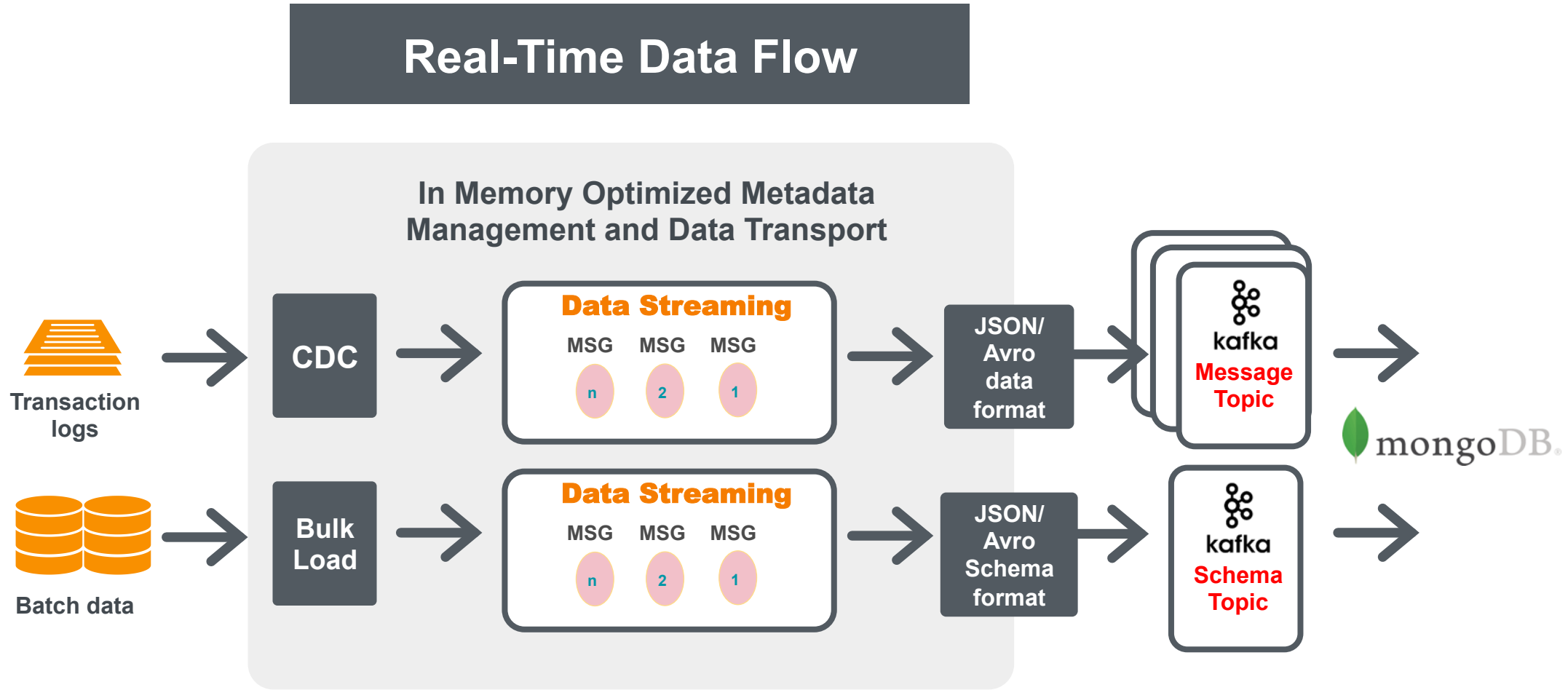
- CDC identifies source updates by scanning change logs
- No software agents required on sources or t
- Minimizes administrative overhead

Low Infrastructure Impact



Attunity Replicate

Streaming CDC to Apache Kafka

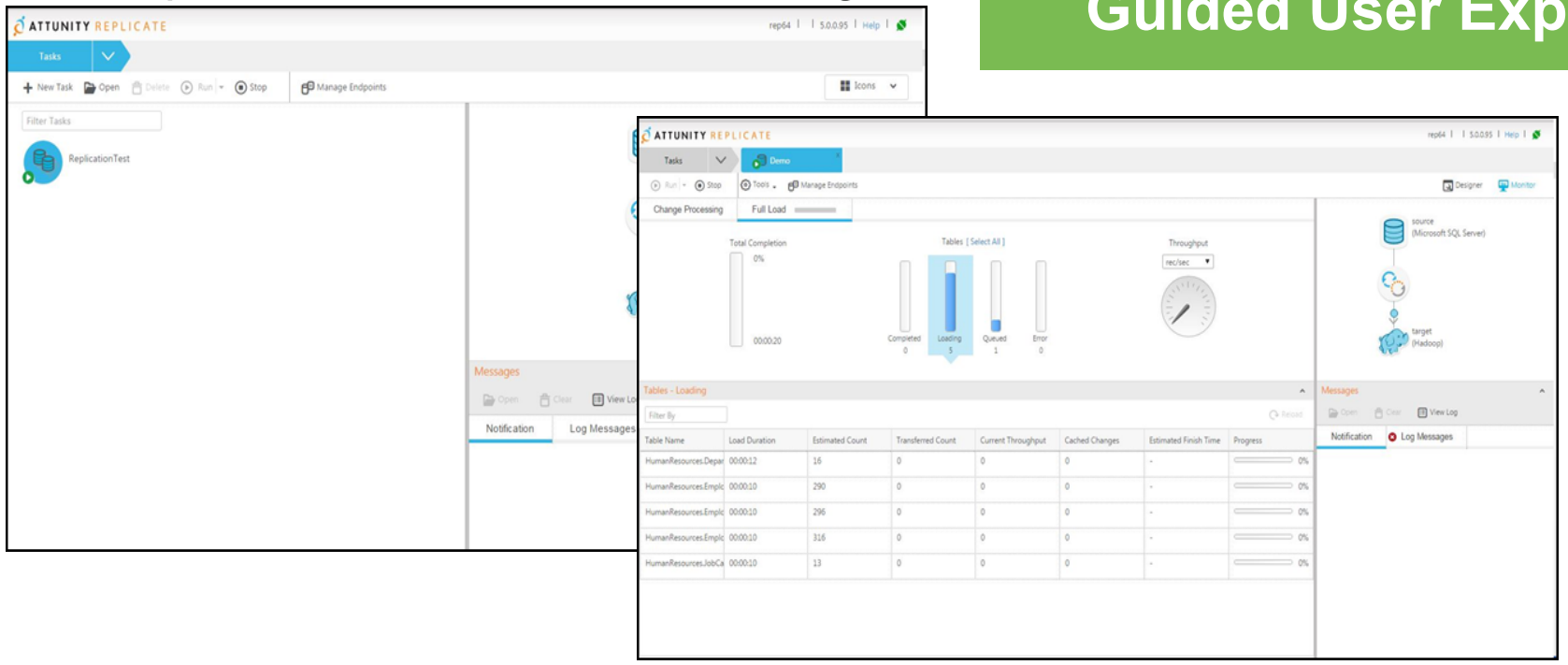


Attunity Replicate

User Interface

- Intuitive web-based GUI
- Drag and drop, wizard-assisted configuration steps
- Consistent process for all sources and targets

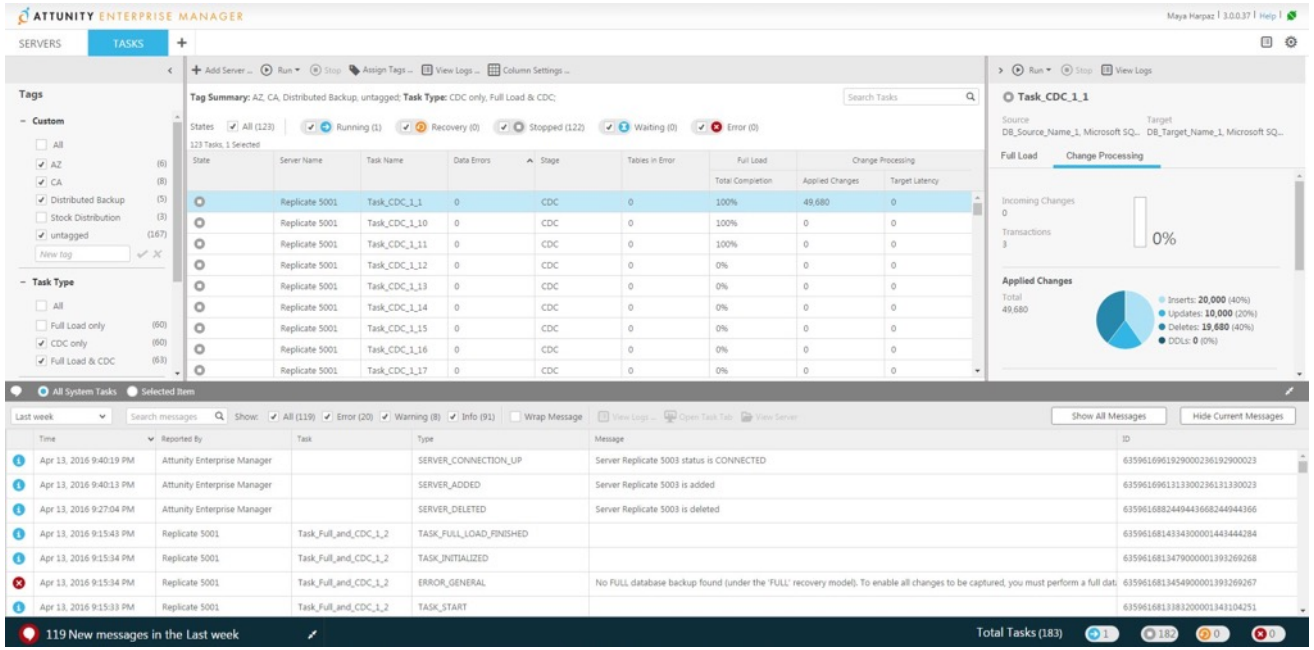
Guided User Experience



mongoDB

Attunity Enterprise Manager (AEM)

Manage Data Ingest and Replication At Scale



Leverage graphical dashboard and APIs (REST and .Net)

- Centralize design and control
  - Loading, mapping, DDL changes
  - Stop, start, resume, reload
  - Automated status discovery
- Manage security controls
  - Granular access controls
  - Full audit trail
- Customize your views
  - Group, search, filter, sort and drill down on tasks
  - Respond to real-time alerts

mongoDB

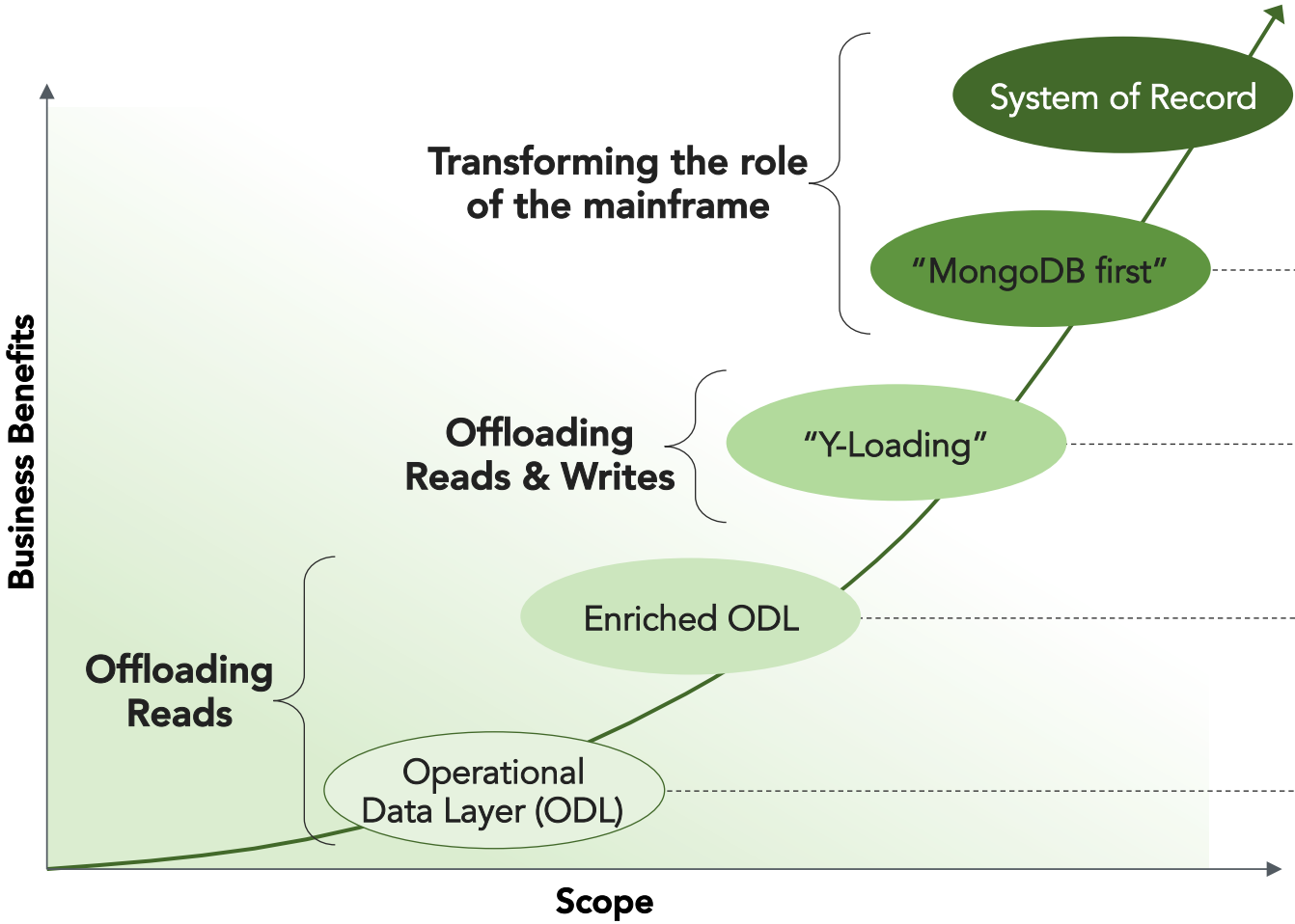
Future Challenges

- Open Banking and Open Data
  - Provides API for 3rd party applications
  - Unpredictable inbound traffic
  - Requires an elastic infrastructure
  - Provisioning time very high

- Blockchain
  - Will have a huge impact to manage transactions

5 phases of Legacy Offloading

MongoDB can help you offload MIPS from the legacy systems, save double-digits in cost and increase agility and capabilities for new use cases at the same time.



MongoDB serves as system of record for a multitude of applications, with deferred writes to the legacy if necessary.

Transactions are written first to MongoDB, which passes the data on to the legacy system of record.

Writes are performed concurrently to the legacy as well as MongoDB (Y-Loading), e.g. via a service-driven architecture.

The Operational Data Layer (ODL) data is enriched with additional sources to serve as operational intelligence platform for insights and analytics.

Records are copied via CDC/Delta Load mechanism from the legacyDB into MongoDB, which serves as Operational Data Layer (ODL), e.g. for frequent reads.

mongoDB

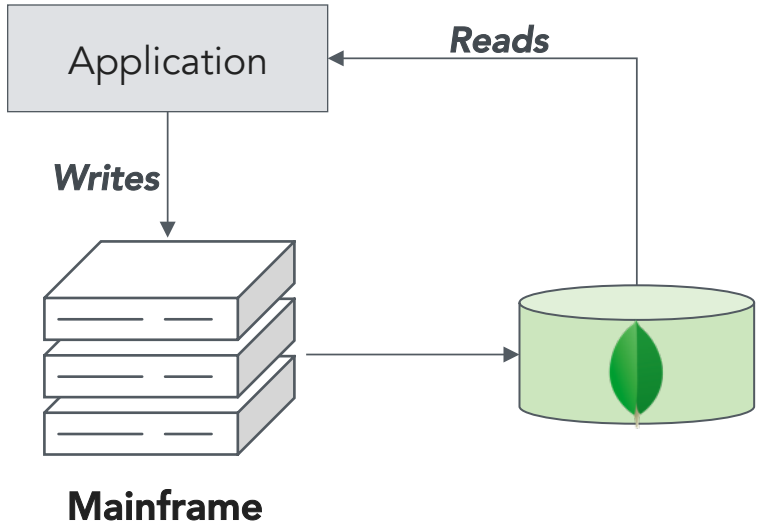
Offloading Reads

Initial use cases primarily focus on offloading costly reads, e.g. for querying large numbers of transactions for analytics or historical views across customer data.

Writes	*100%*	
Reads	*50-90%*	*10-50%*

Operational Data Layer (ODL)

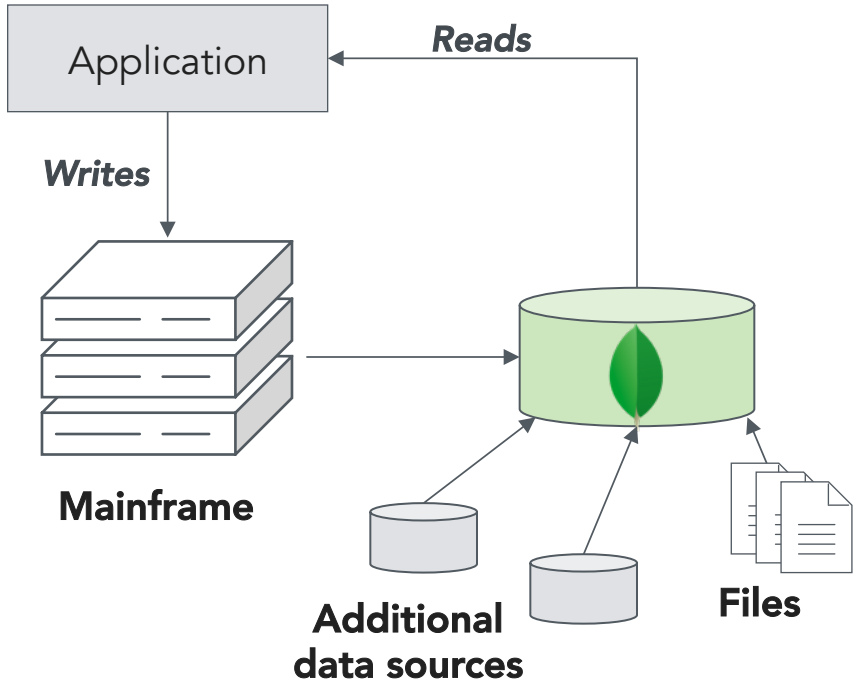
Using a change data capture (CDC) or delta load mechanism you create an operational data layer alongside the mainframe that serves read-heavy operations.



Writes	*100%*	
Reads	*25-75%*	*25-75%*

Enriched Operational Data Layer (ODL)

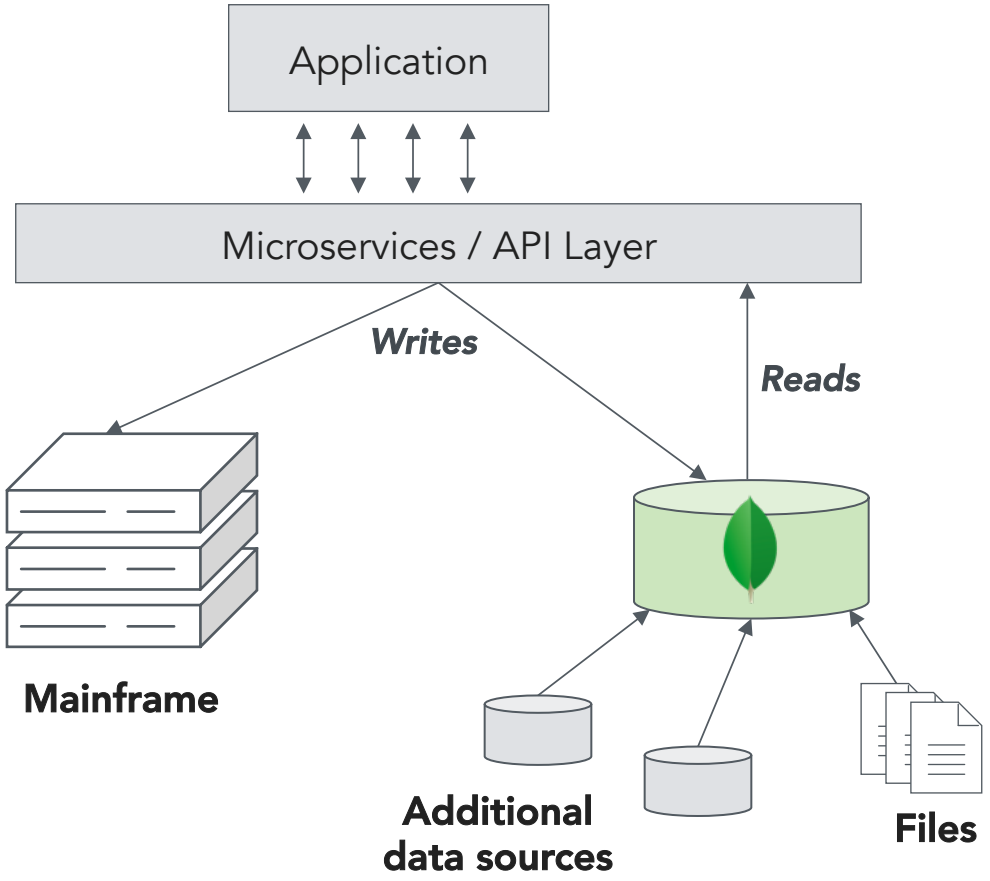
Additional data sourced are loaded into the ODS to create an even richer picture of your existing data and enable additional use cases like advanced analytics.



mongoDB

Offloading Reads & Writes

By introducing a smarter architecture to orchestrate writes concurrently, e.g. via a Microservices architecture, you can shift away from delayed CDC or delta load mechanisms.



Writes	*75-90%*	*10-25%*
Reads	*20-60%*	*40-80%*

Y-Loading

Writing (some) data concurrently into the mainframe as well as MongoDB enables you to further limit interactions with the mainframe technology .

It also sets you up for a more transformational shift of the role of the mainframe with regards to your enterprise architecture.

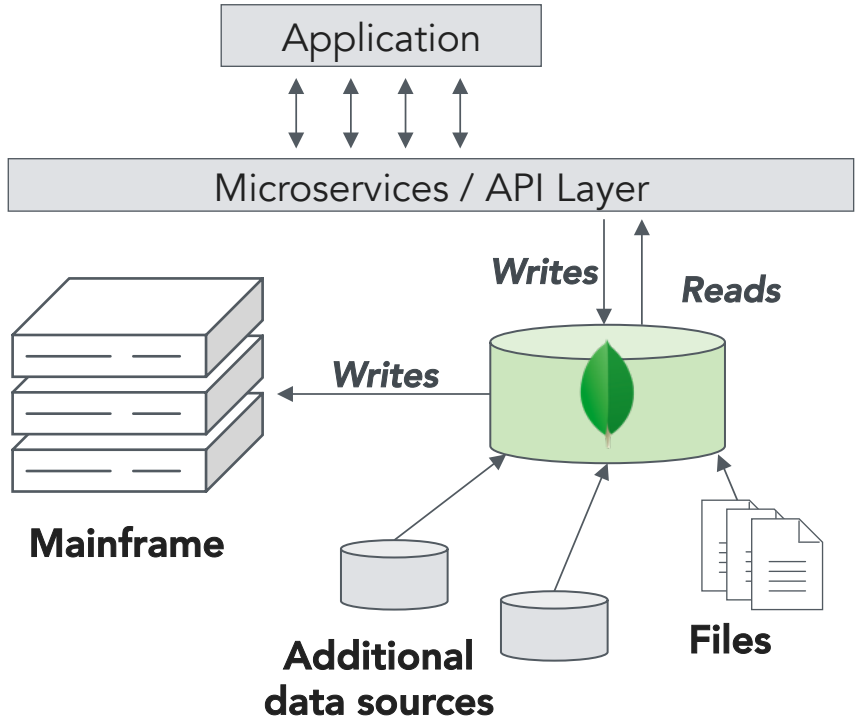
Transforming the role of the legacy

With a shift towards writing to MongoDB first before writing to the mainframe (if at all) you are further changing the meaning of "system of record" and "mainframe" within the organisation.

Writes	*50-80%*	*20-50%*
Reads	*10-40%*	*60-90%*

"MongoDB first"

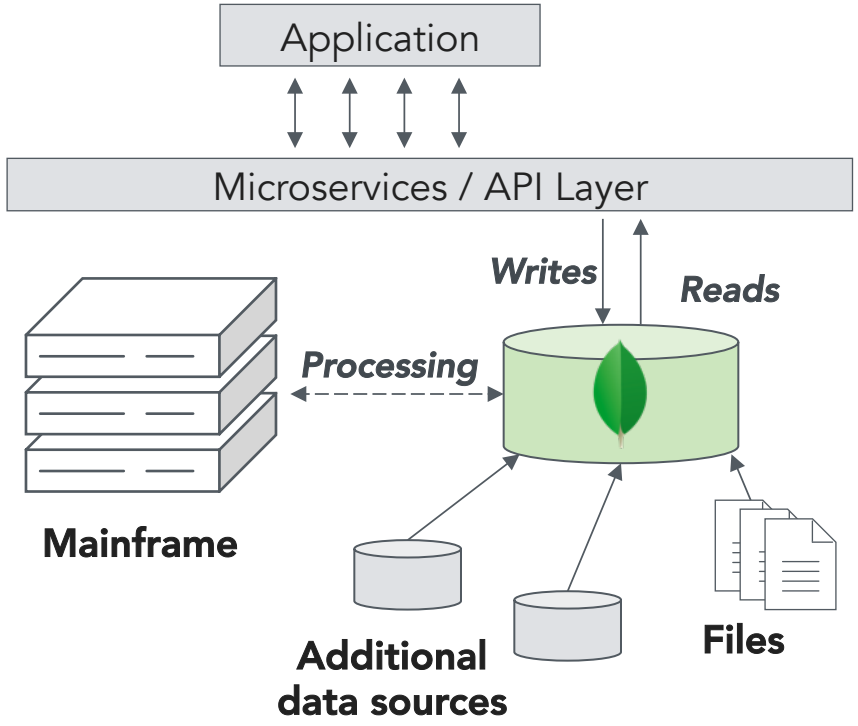
Transactions first write to MongoDB, which can serve as buffer before it passes transactions to the mainframe as System of Record.



Writes	*10-50%*	*50-90%*
Reads	*0-10%*	*90-100%*

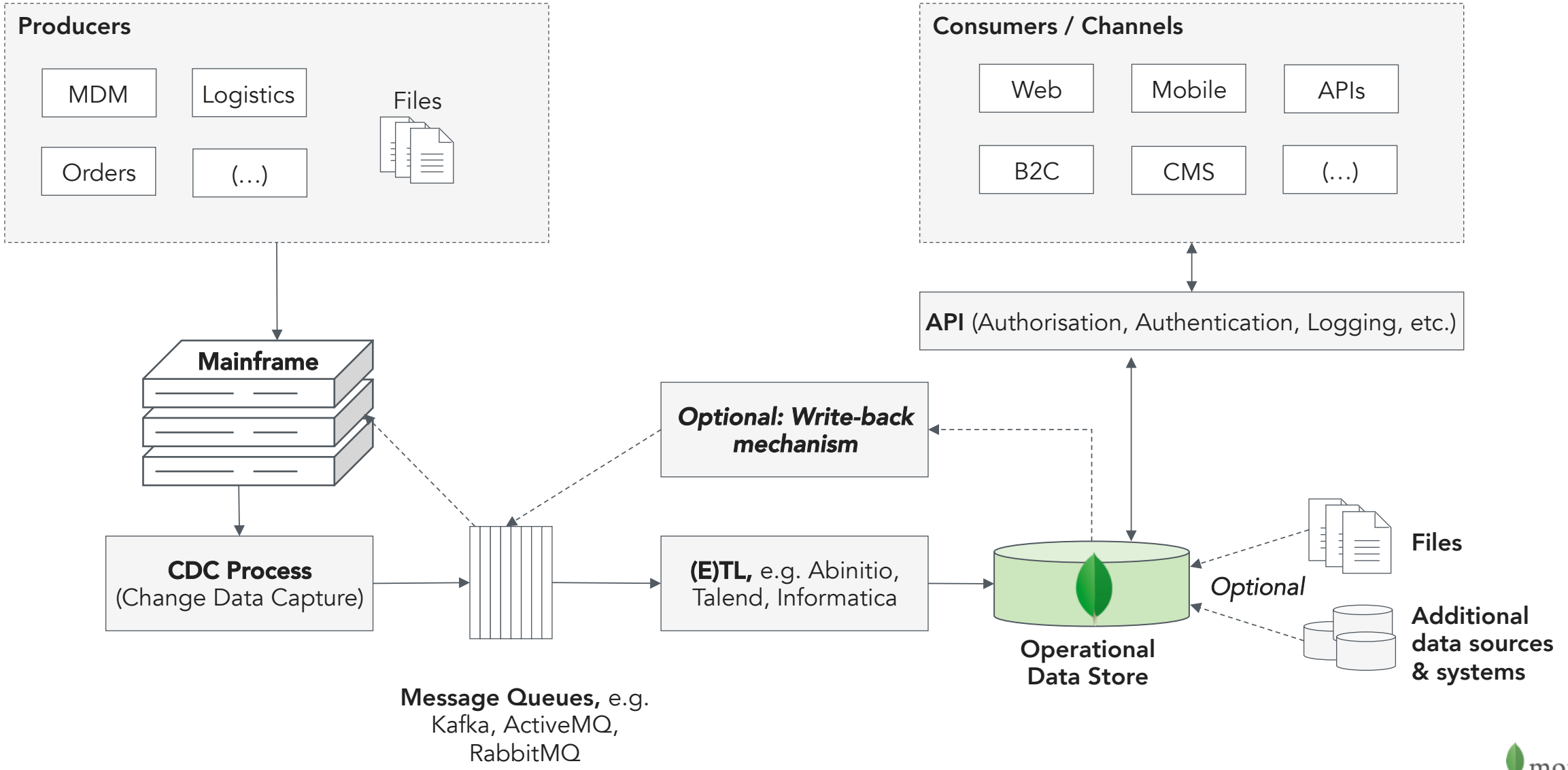
System of Record

MongoDB serves as main System of Record, with writes optionally being passed on to the mainframe for legacy applications only or it gets decommissioned entirely.



mongoDB

Future Architecture

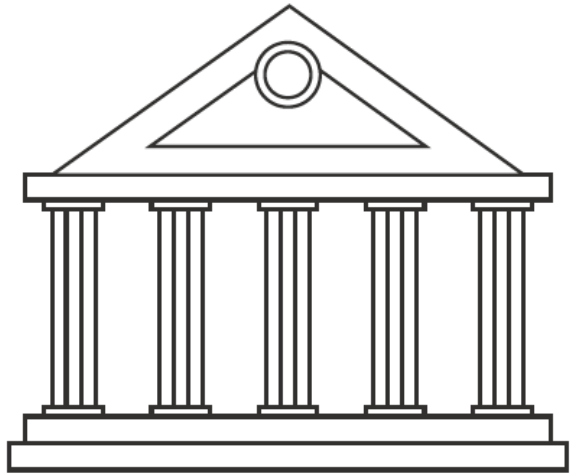


mongoDB

Why MongoDB?

DISTRIBUTED, MASSIVELY SCALABLE DATABASE

FLEXIBLE DATA MODEL WITH RICH QUERYING



ENTERPRISE GRADE >1/3RD FORTUNE 100

Document Model

RDBMS

PERSON

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

NO RELATION

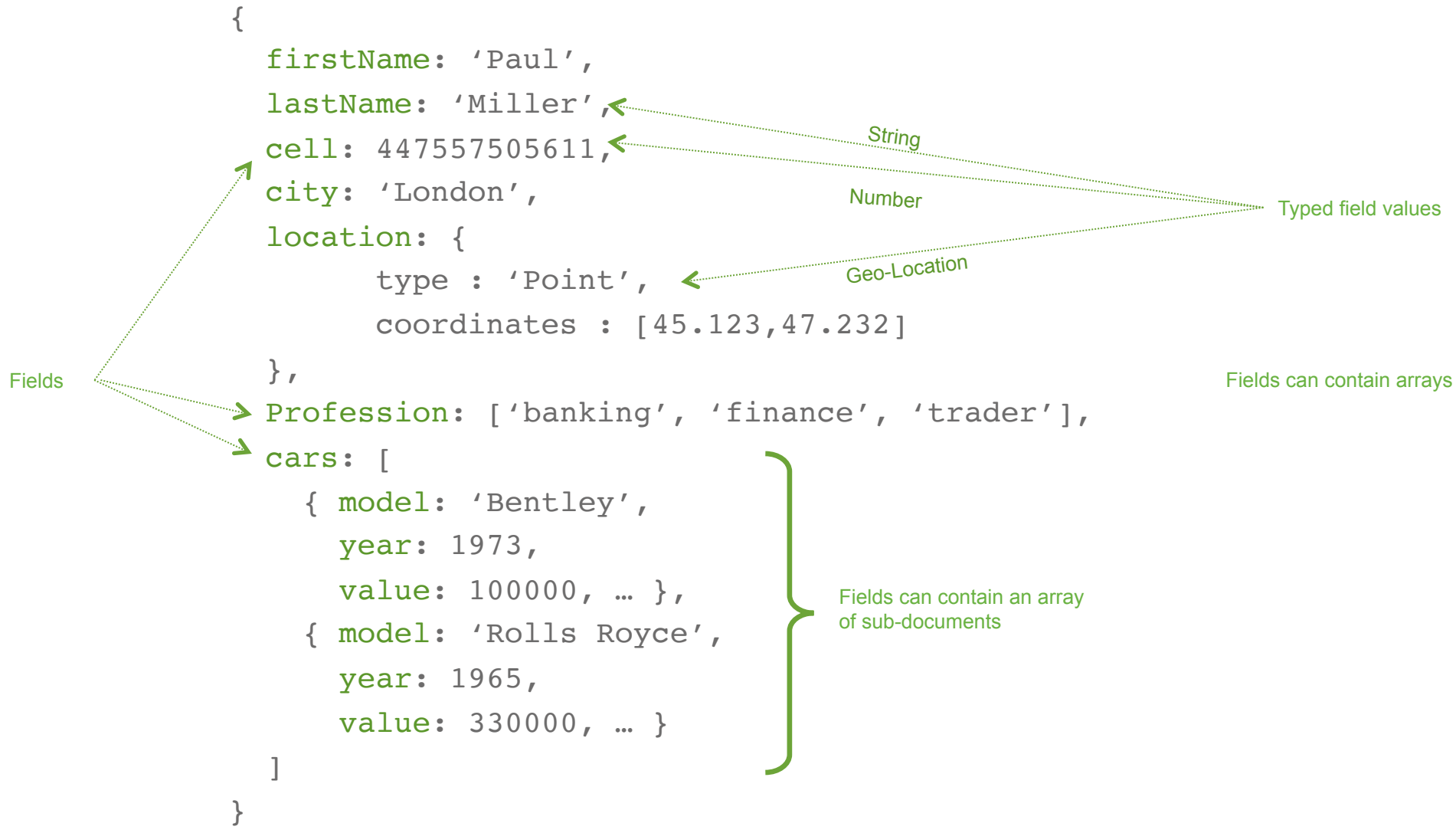
CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB

```
{
  firstName: ‘Paul’,
  lastName: ‘Miller’,
  city: ‘London’,
  location: {
        type : ‘Point’,
        coordinates : [45.123,47.232]
  },
  cars: [
    { model: ‘Bentley’,
      year: 1973,
      value: 100000, … },
    { model: ‘Rolls Royce’,
      year: 1965,
      value: 330000, … }
  ]
}
```

Documents are Rich



Documents are Flexible

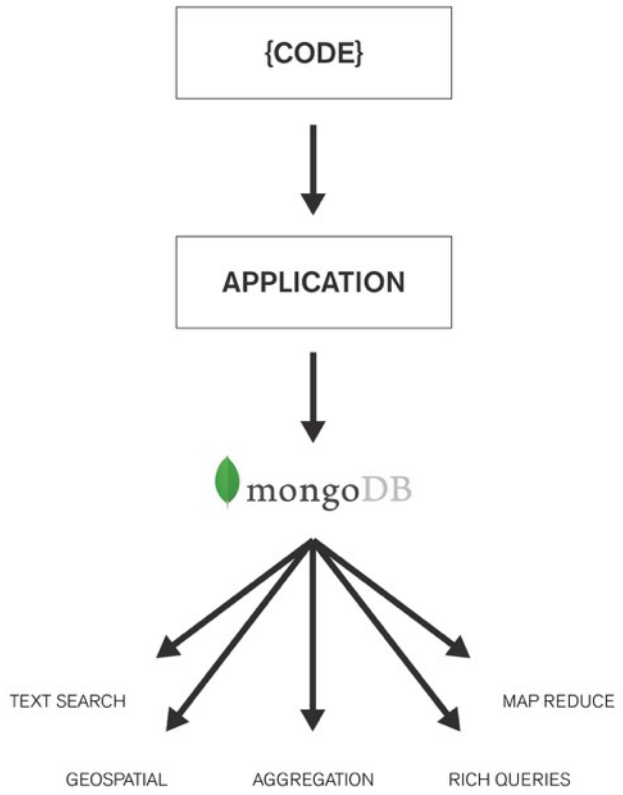
Documents in the same product catalog collection in MongoDB

```
{
  product_name: 'Acme Paint',
  color: ['Red', 'Green'],
  size_oz: [8, 32],
  finish: ['satin', 'eggshell']
}
```

```
{
  product_name: 'T-shirt',
  size: ['S', 'M', 'L', 'XL'],
  color: ['Heather Gray' … ],
  material: '100% cotton',
  wash: 'cold',
  dry: 'tumble dry low'
}
```

```
{
  product_name: 'Mountain Bike',
  brake_style: 'mechanical disc',
  color: 'grey',
  frame_material: 'aluminum',
  no_speeds: 21,
  package_height: '7.5x32.9x55',
  weight_lbs: 44.05,
  suspension_type: 'dual',
  wheel_size_in: 26
}
```

MongoDB is Full-Featured



Rich Queries	Find Paul's cars Find everybody in London with a car between 1970 and 1980
Geospatial	Find all of the car owners within 5km of Trafalgar Sq.
Text Search	Find all the cars described as having leather seats
Aggregation	Calculate the average value of Paul's car collection
Map Reduce	What is the ownership pattern of colors by geography over time (is purple trending in China?)

Operational Data Store

Data lake to store data from multiple sources for operations on the data. ODS is built to store and process read only customer transactions for business operations, analysis and reporting.

Before Scenarios

- With the advent of mobile banking, Customer has experienced a significant growth of traffic originating from mobile devices to Mainframe platforms that supports banking applications.
- Growth of traffic, which is expected to continue, has led to an increased cost of operations and decreased performance.
- Ability to provide high resiliency during mainframe outages.

Solution

- Existing ETL processes that load transaction data into Teradata on a daily basis are updated to additionally feed data to MongoDB paving the way for decommissioning of Teradata.
- In subsequent phases of the project, MongoDB will be updated in near real-time via a live transactions feed .
- De-normalized real time data store using MongoDB with the benefits to reduce growth.

After Scenarios

- Stand-in capability to support Resiliency during planned and unplanned outages across mainframe system and other source systems.
- Reduced cost of operations.
- Reduced number of read only transactions to Mainframes, thereby freeing up mainframe resources for additional growth or reducing MIPS usage and costs.

BBVA

Mainframe Offloading enables insight through Single View of Customer

Spanish bank replaces Teradata and Microstrategy to increase business insight and avoid significant cost

Problem

Branches required an application that offered all information about a given customer and all his contracts (accounts, loans, cards, etc.).

Multi-minute latency for accessing customer data stored in Teradata and Microstrategy.

In addition, accessing data frequently from the legacy systems would cause spikes in MIPS and related cost.

Solution

Offloaded to MongoDB where data is highly-available and can be accessed by new applications and channels.

Built single view of customer on top of MongoDB – flexible and scalable app, easy to adapt to new business needs.

Super fast, ad hoc query capabilities (milliseconds), and real-time analytics thanks to MongoDB's Aggregation Framework.

Can now leverage distributed infrastructure and commodity hardware for lower total cost of ownership and greater availability.

Results

Cost avoidance of 10M$+

Application developed and deployed in less than 6 months. New business policies easily deployed and executed, bringing new revenue to the company.

Current capacity allows branches to load instantly all customer info in milliseconds, providing a great customer experience.

New applications and services can be built on the same data platform without causing MIPS/cost or increasing risk by putting more stress on legacy systems.

mongoDB

Q&A

Thanks!