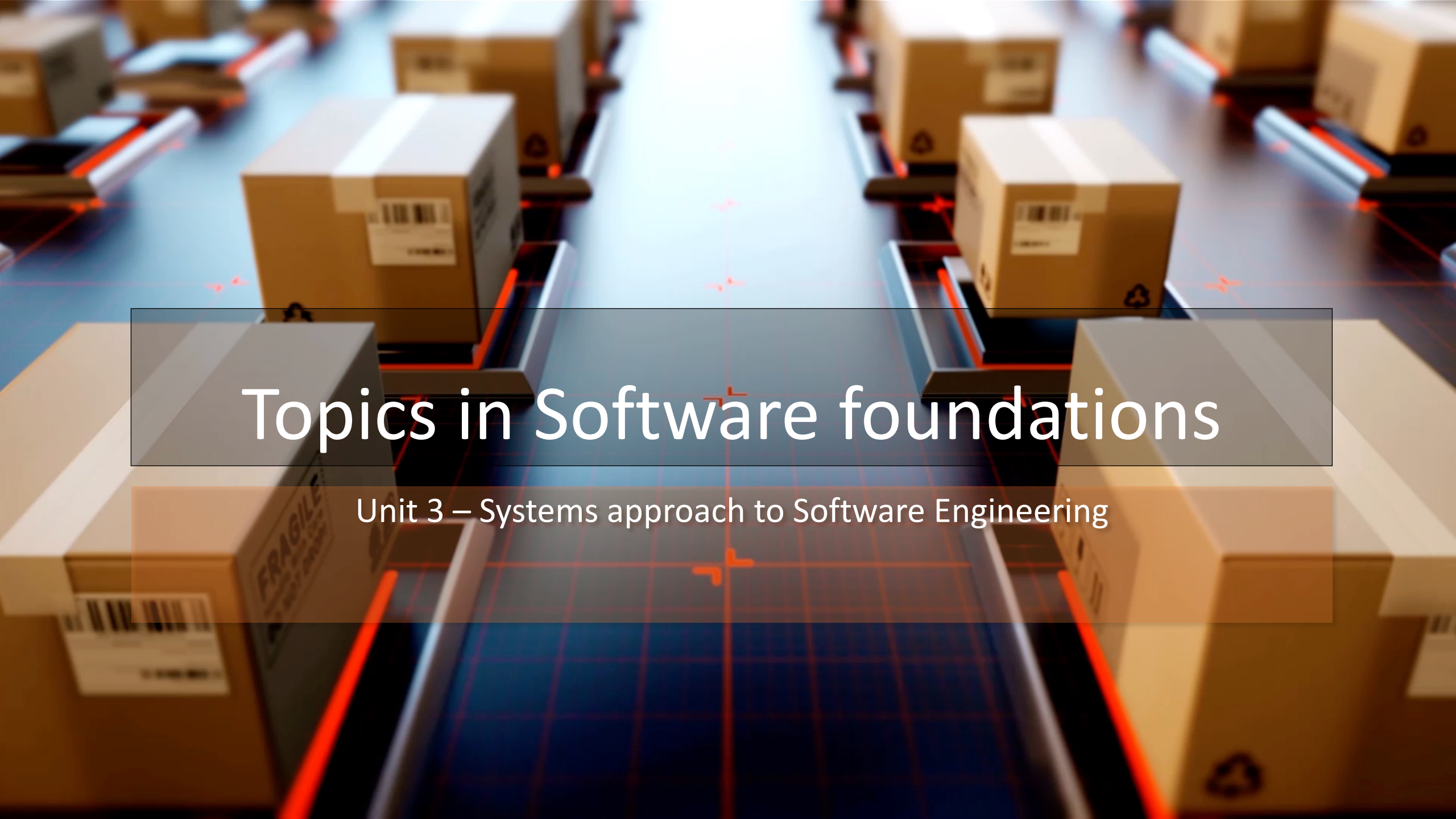


Topics in Software foundations

Unit 3 – Systems approach to Software Engineering

Topics we cover

- Introduction to systems modeling
- Modeling System of Systems (SoS)
- Modeling communications
- Modeling UI systems
- Modeling data access
- Formalizing TS as a modeling language

Recap/continuation from the last class

System modifications for various types of problems

Richness of UI

- There may be multiple data sources that compose a page
  - Multiple MVC systems at play, web page view is composed of multiple of these Views.
- A page component may include data from multiple sources
  - A new data model is created to map to the required view, MVC using this model is used.

Quality of client-server connection

- Low bandwidth
  - Reduce functionality (impacts U), reduce user elements (impacts Y)
- High latency
  - Take local decisions on client – impacts transition functions. Requires sync with server for decisions, other issues
- Frequent connection drops
  - Reduce connection setup time, apply high latency solution

Quality of client device

- Low processing power (low-end android phone)
  - Lite version (functionally)
- Low display capabilities (watch)
  - Lite version (visually)

Patterns of application access

- The user may use multiple devices to access the system simultaneously
  - As long as observer is in place and in use, this should work. Concurrent access needs to be available.
- A large number of users access the system simultaneously
  - Gateway to route the traffic - split into batches which can be handled by existing system
  - Refine the solution to have better scale characteristics

System modifications for various types of problems

There may be multiple data sources that compose a page

- Multiple MVC systems at play, web page view is composed of multiple of these Views.

A page component may include data from multiple sources

- A new data model is created to map to the required view, MVC using this model is used.

Low bandwidth

- Reduce functionality (impacts U), reduce user elements (impacts Y)

High latency

- Take local decisions on client (impacts F). Requires sync with server for decisions made locally
- Reduce message chatter with server

Frequent connection drops

- Reduce connection setup time, apply high latency solution

Low processing power (low-end android phone)

- Lite version (functionally)

Low display capabilities (watch)

- Lite version (visually)

The user may use multiple devices to access the system simultaneously

- As long as observer is in place and in use, this should work. Concurrent access needs to be available.

A large number of users access the system simultaneously

- Gateway to route the traffic - split into batches which can be handled by existing system
- Refine the solution to have better scale characteristics

Modeling Data Access

Unit 3 – TiSF S'23

Session 7 (2023-04-20)

Two categories of data

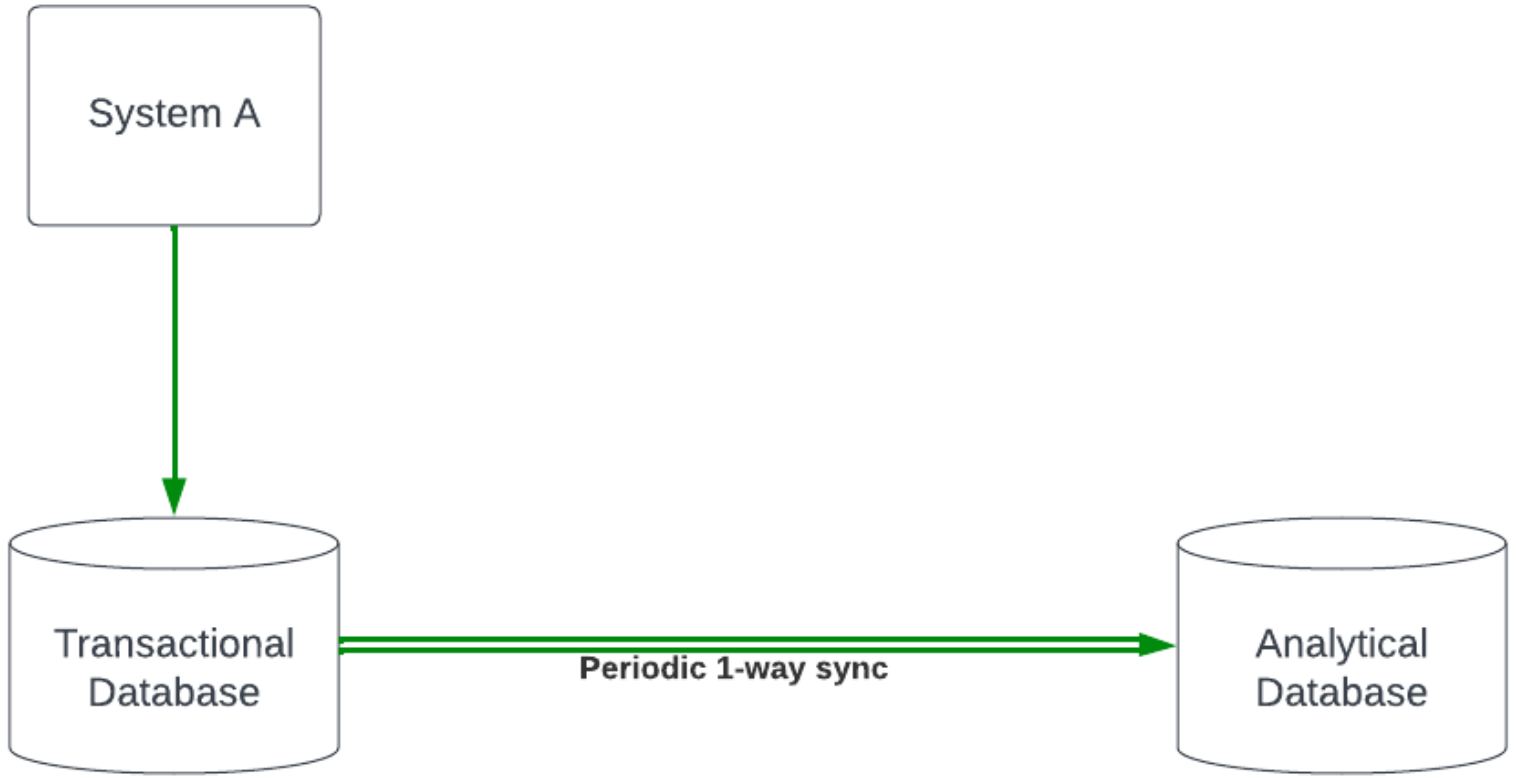
Transactional

- State data for a system (and SoS)
  - Appointments booked by this user this year
  - Money collected in every transaction in a salon since beginning of time
- Action (U) and Observable (Y) may have data associated with them
  - Booking required for a specific user for a particular date and time
  - All empty calendar slots in response to a query about availability

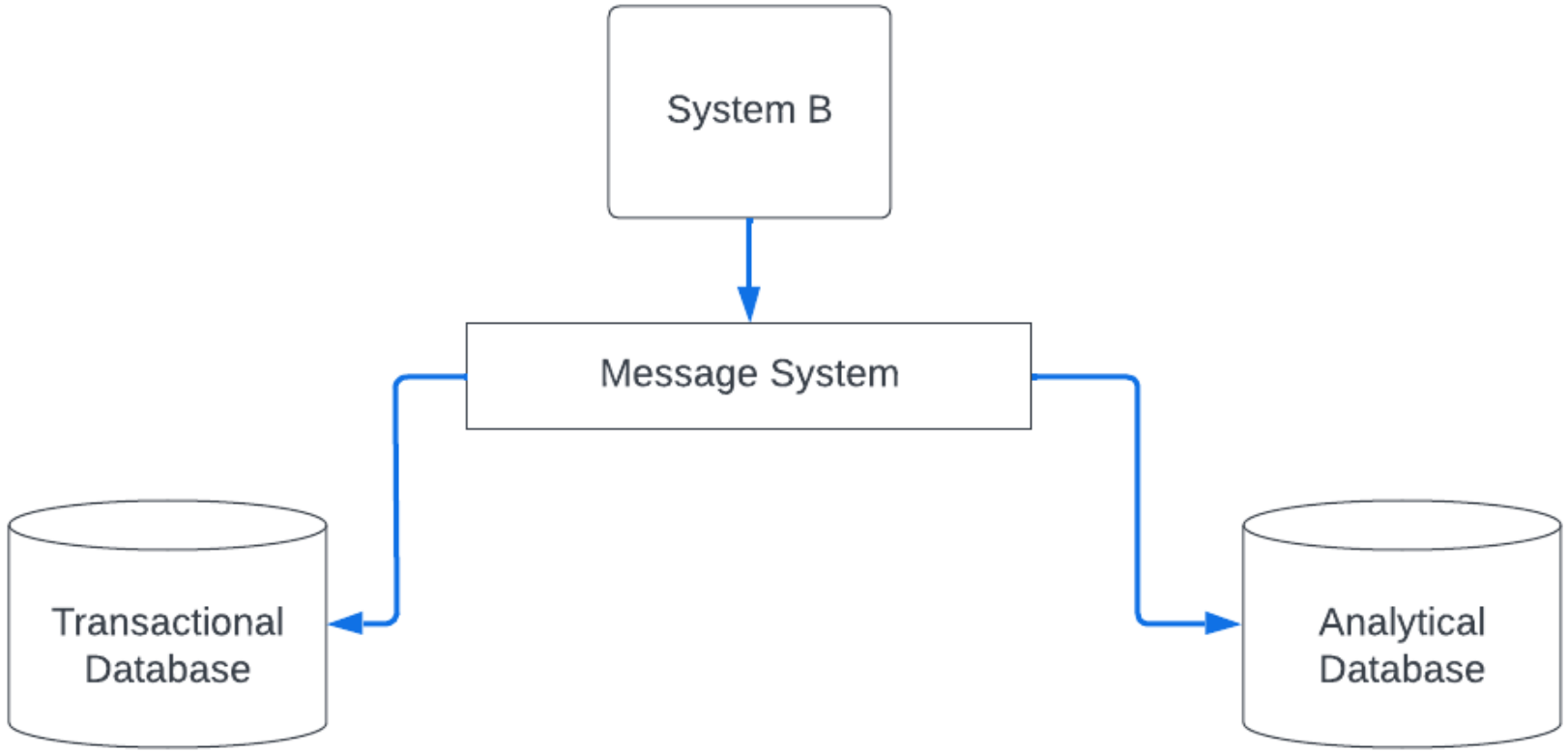
Analytical

- Record for later use (Archive)
  - Log every system communication message
  - Audit trails of all payment collection
- Analysis and insights
  - Data warehouse for business performance reporting
  - Demand forecast system using transaction data

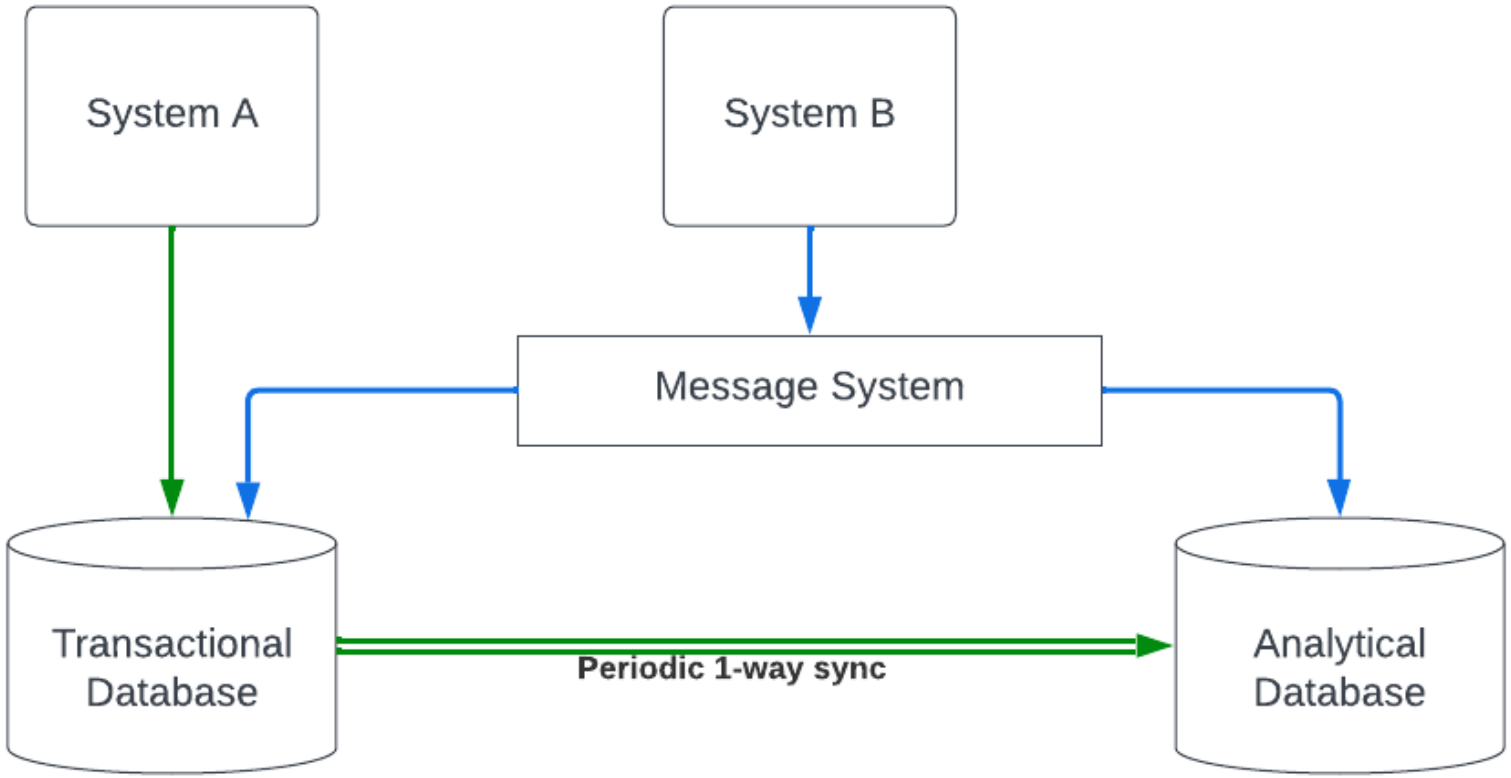
Transaction and analytical data – Offline sync



Transaction and analytical data – real-time sync



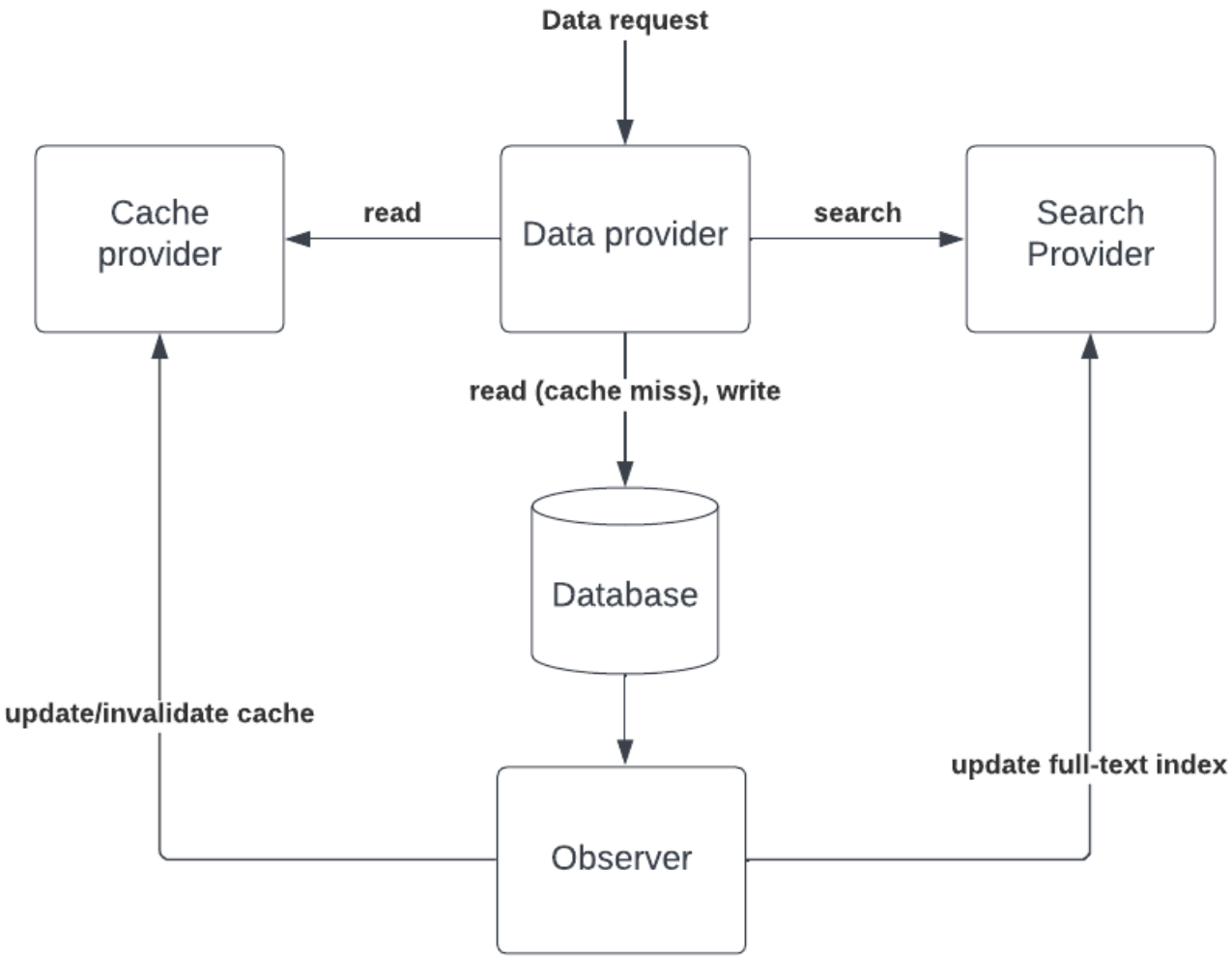
Transaction and analytical data - both



Various ways data can be held by the system

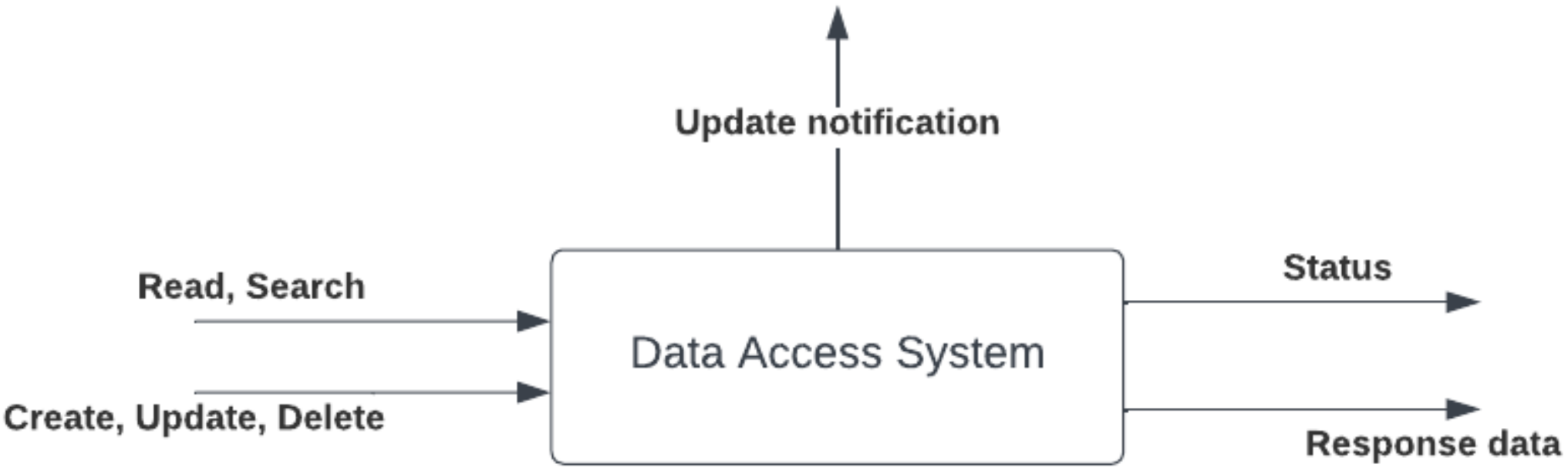
<table>
<tr><td colspan="4">Data Access System</td></tr>
<tr><td>Internal Service API</td><td>External Service API</td><td>Querying language API</td><td>File API</td></tr>
<tr><td>Internal data service</td><td>External data service</td><td>Database system</td><td>File System</td></tr>
<tr><td colspan="4">Operating System Services (Network, Storage)</td></tr>
</table>

Typical data access system model

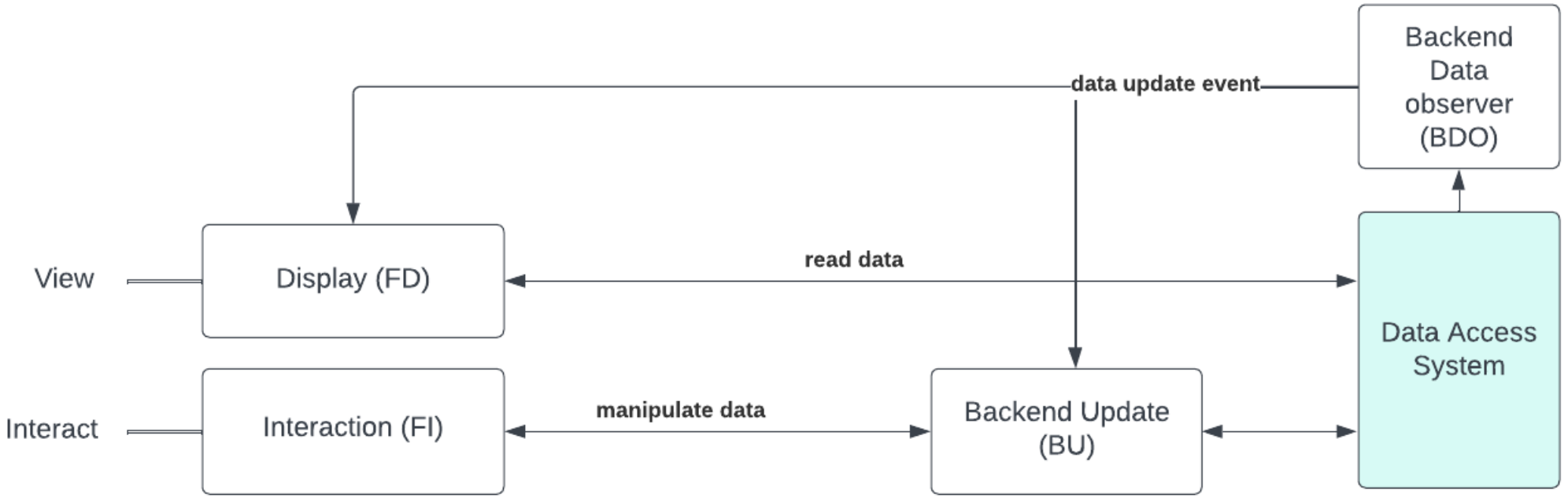


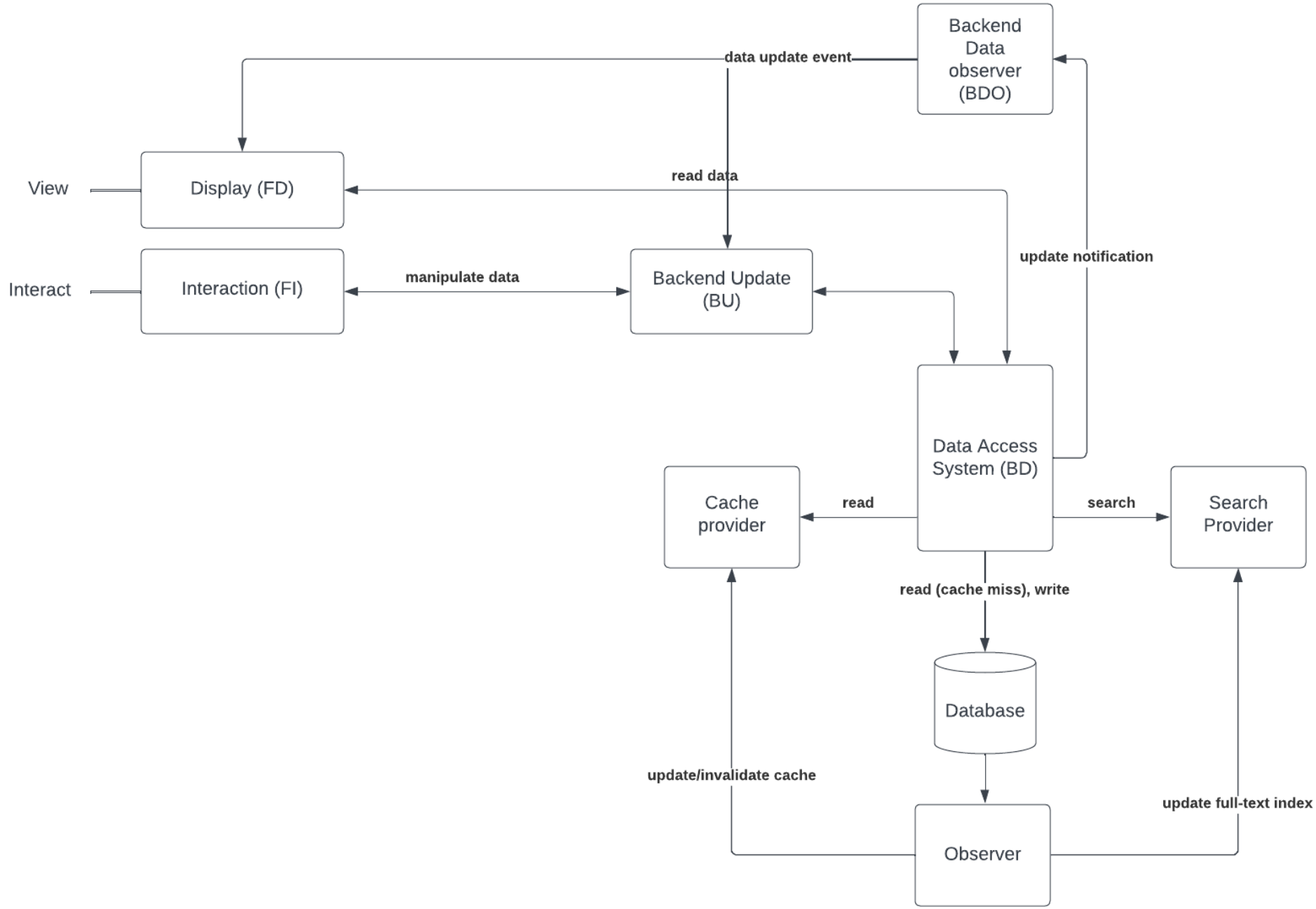
Designing data-intensive applications

Data access system



UI System (discussed in last class)




Backend Data observer (BDO)
data update event
View
Display (FD)
read data
Interact
Interaction (FI)
manipulate data
Backend Update (BU)
update notification
Data Access System (BD)
Cache provider
read
search
Search Provider
read (cache miss), write
Database
update/invalidate cache
update full-text index
Observer

Data scenarios that will require model tweaks

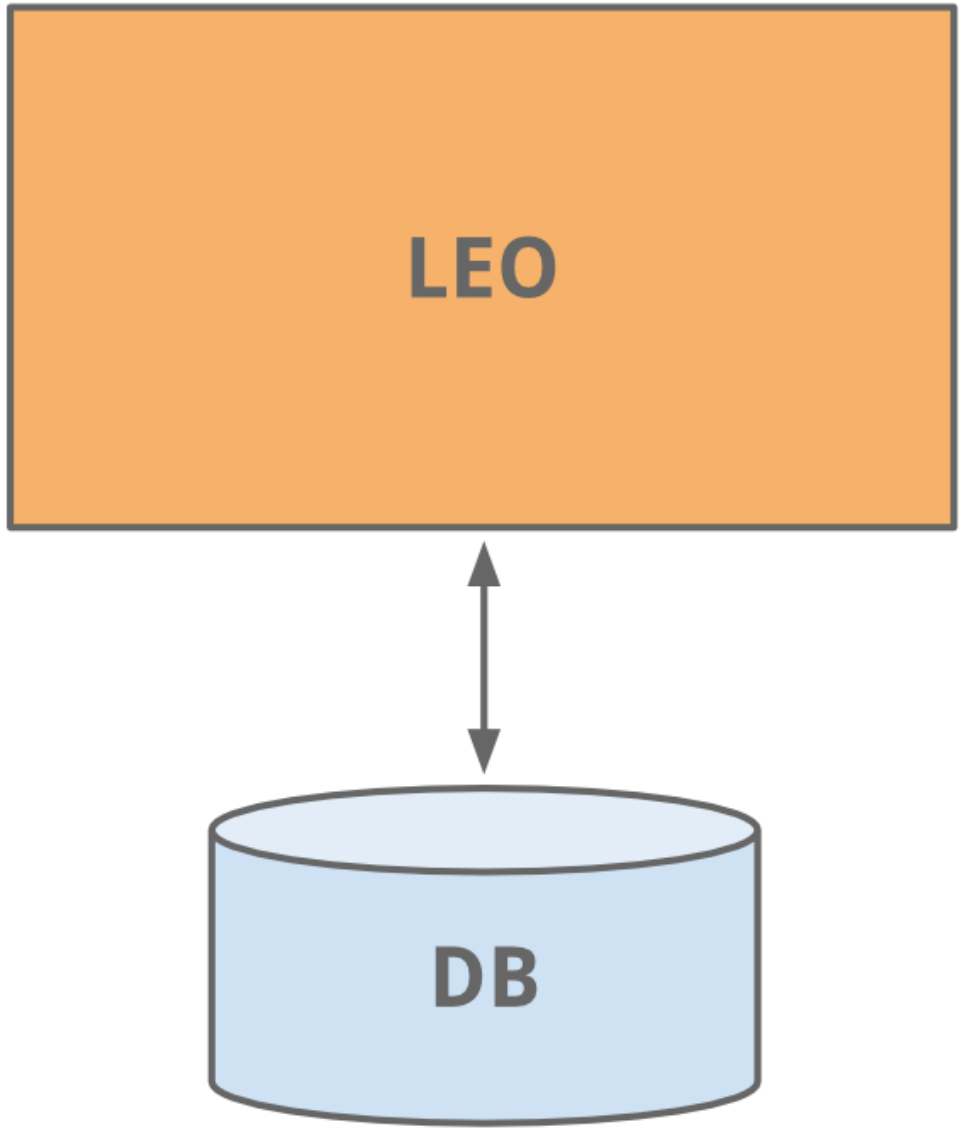
A large amount of data in every read response
Too many data access requests to render a UI
Many more reads compared to writes
Database server going down
Too much data for a single database server

To be cont*inued in next class..*

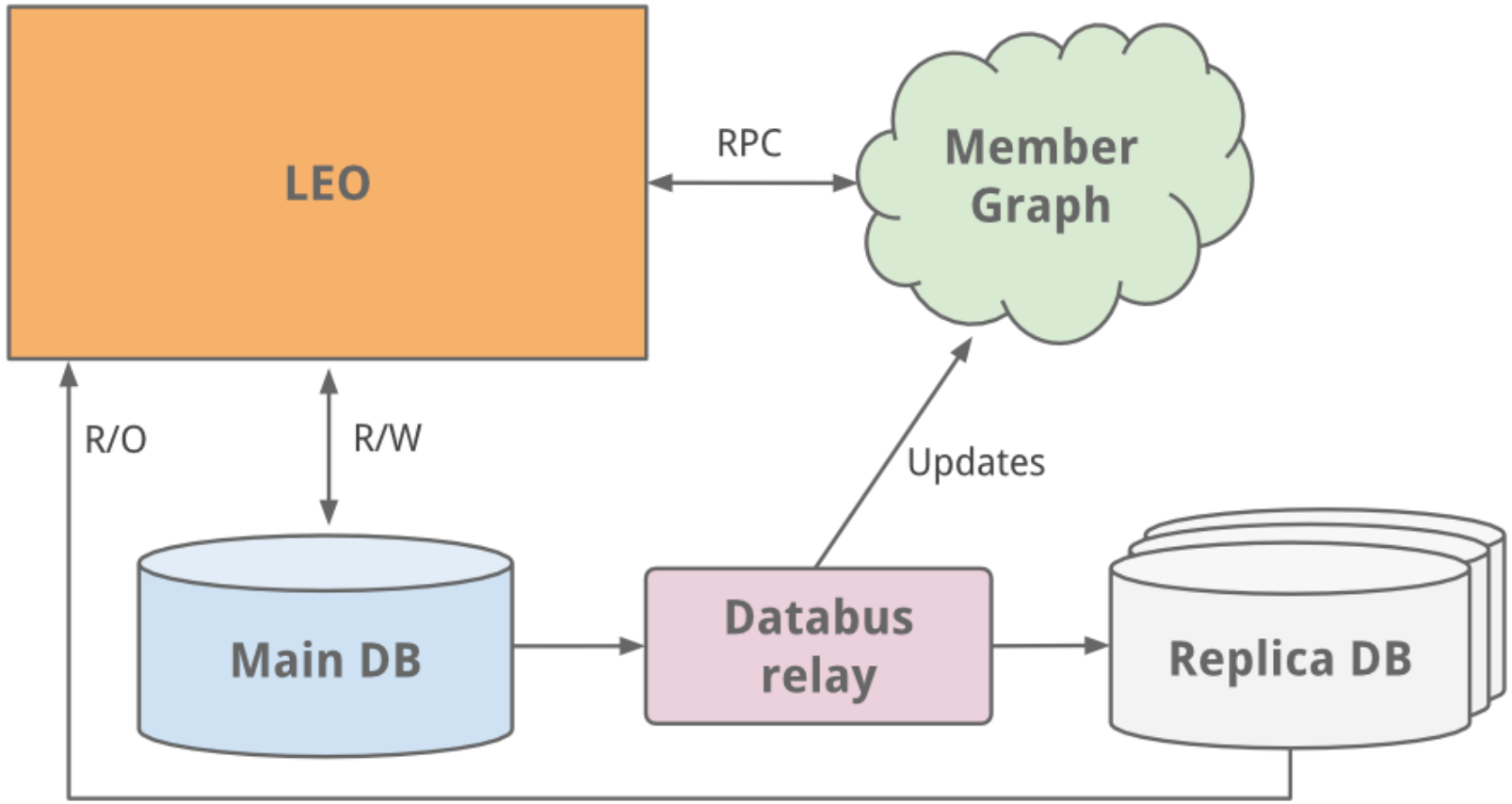
LinkedIn scaling story

- https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin

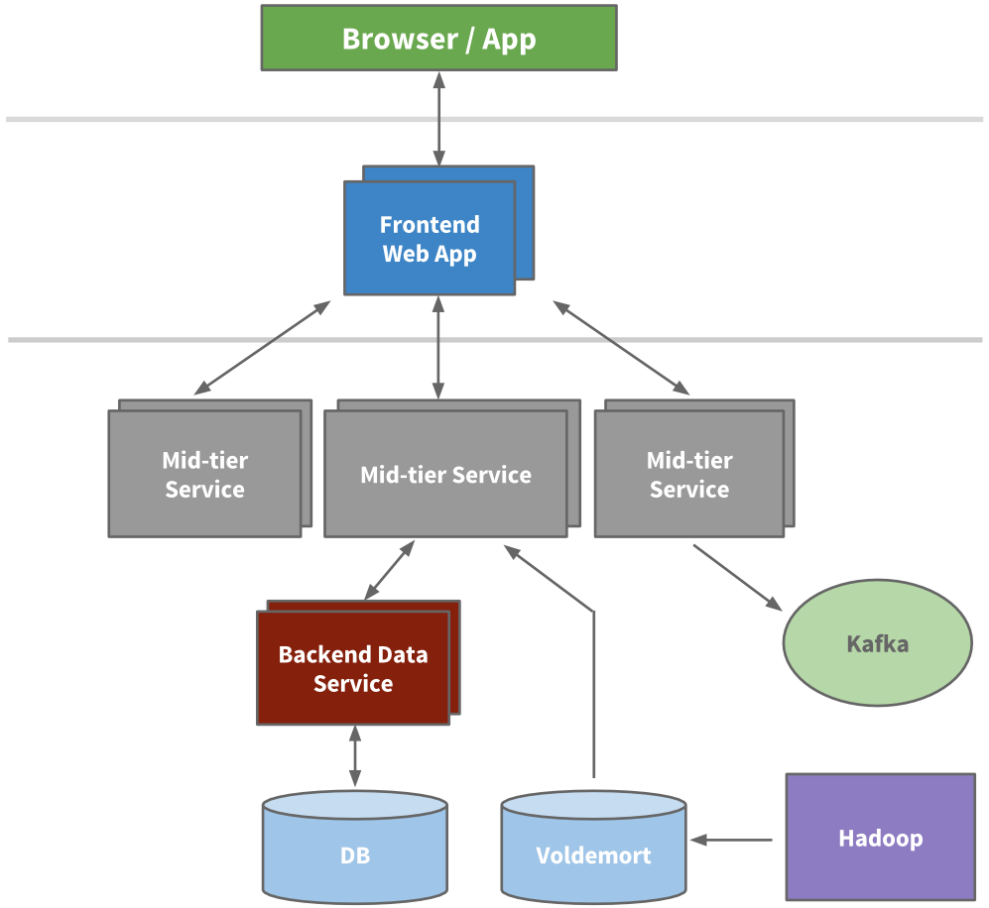
Starting up..



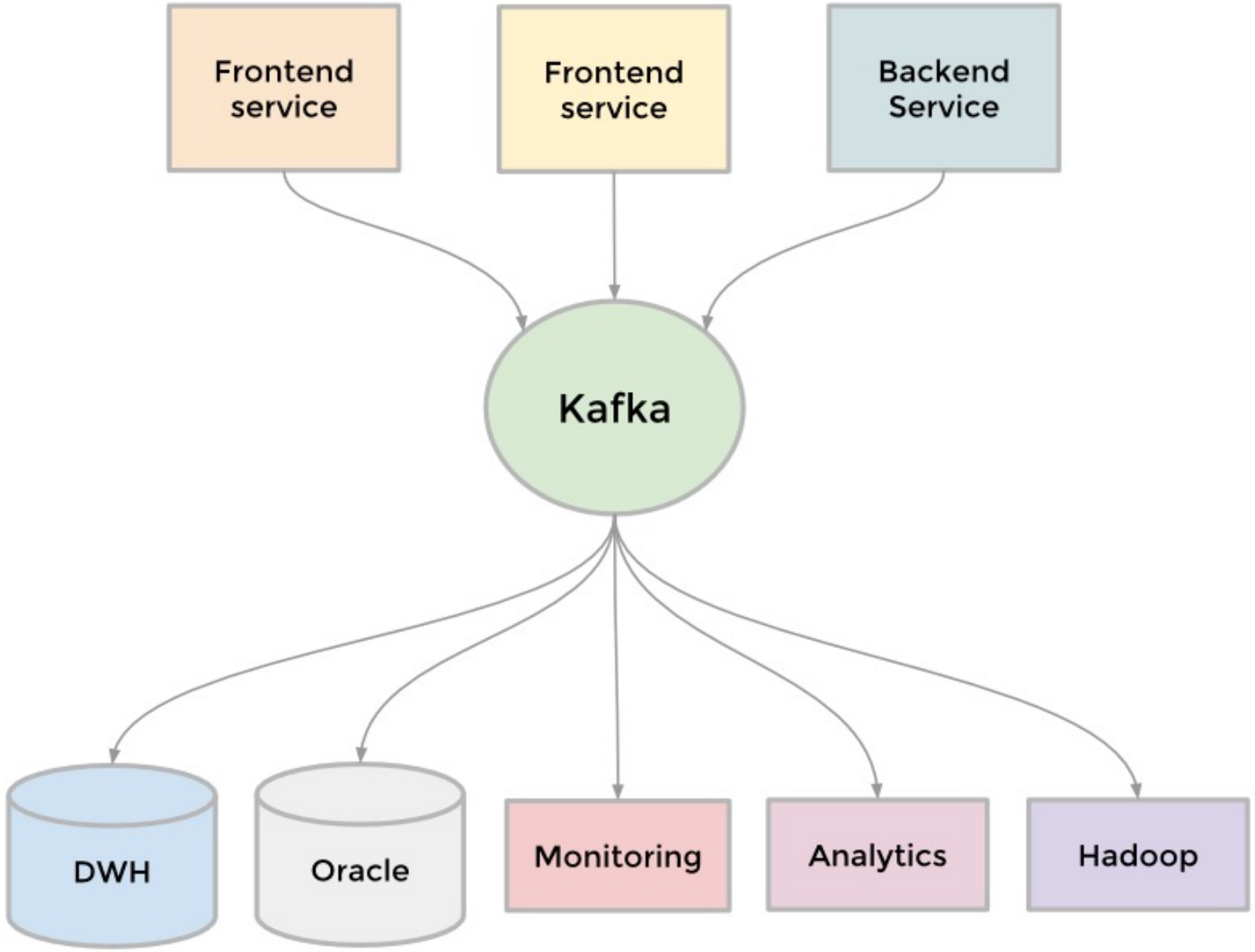
Monolith application



Separate service for member connections, replica for reads

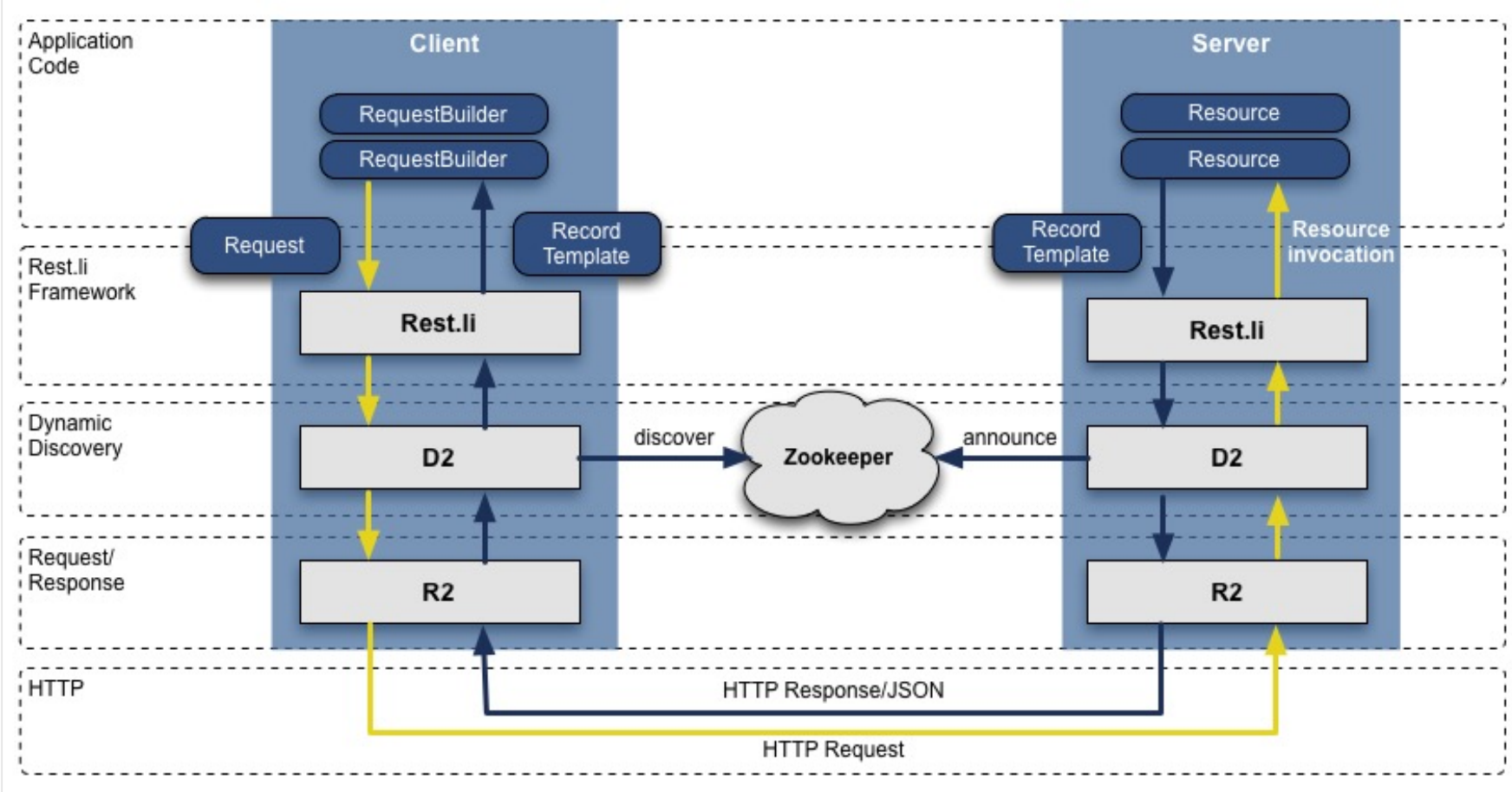


Migration to SOA, caching (including distributed datastore)



Scaling data streams using a messaging system

Modern years..

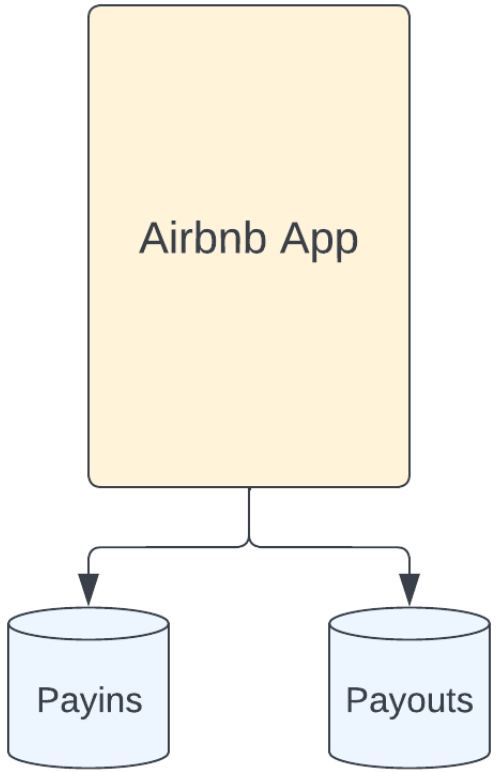


Rest.li – Data model centric architecture (similar to the AirBnB journey with entities)

AirBnB case study on data architecture

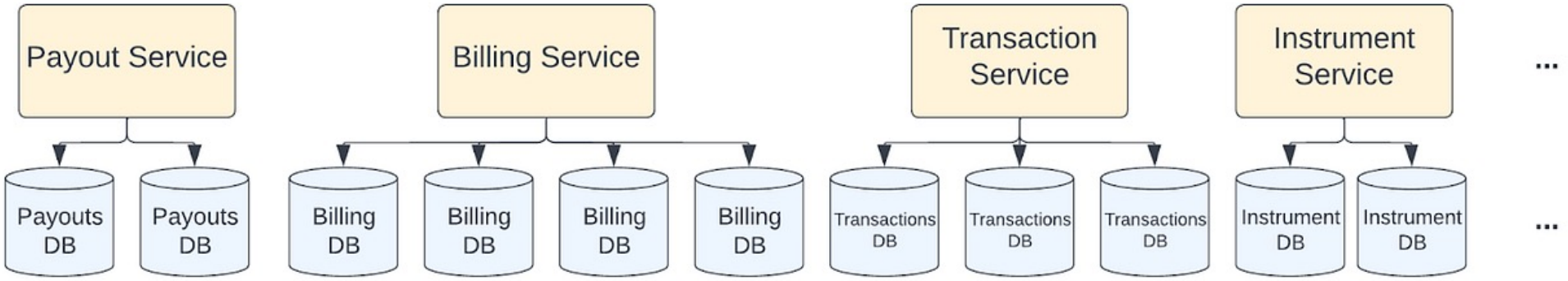
https://medium.com/airbnb-engineering/unified-payments-data-read-at-airbnb-e613e7af1a39

Once upon a time..



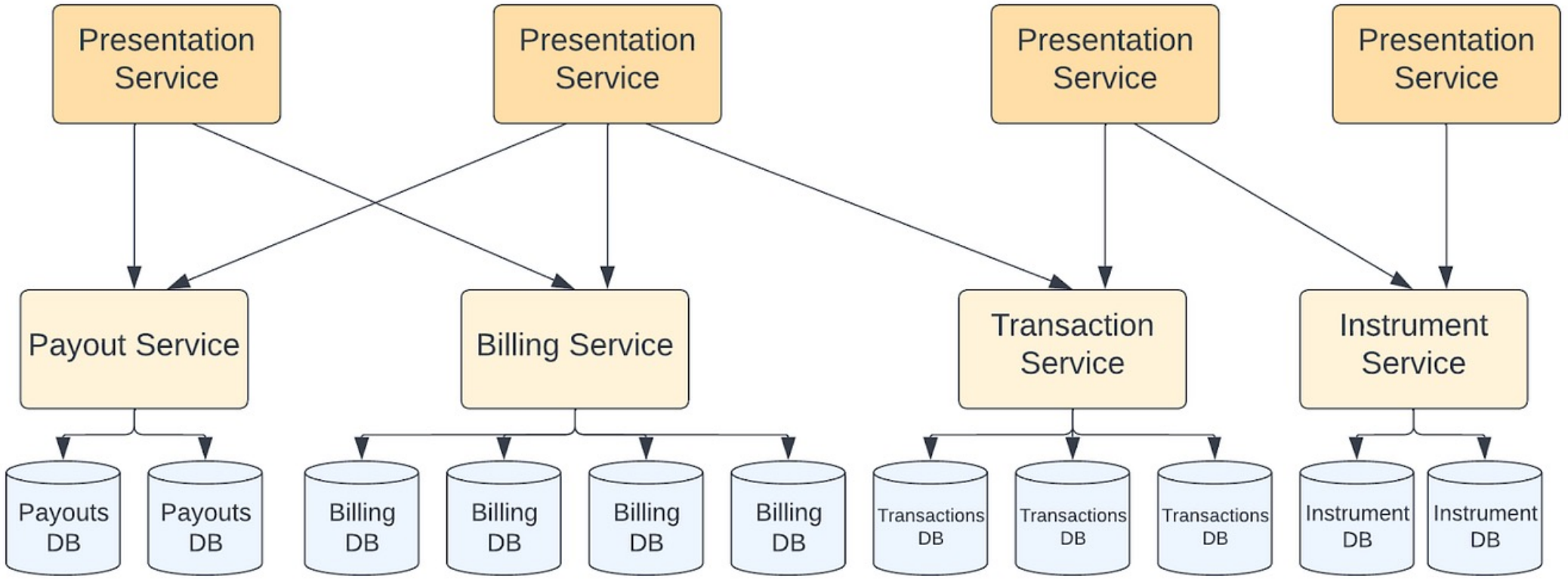
Too hard to scale!

Service-oriented architecture for Payments



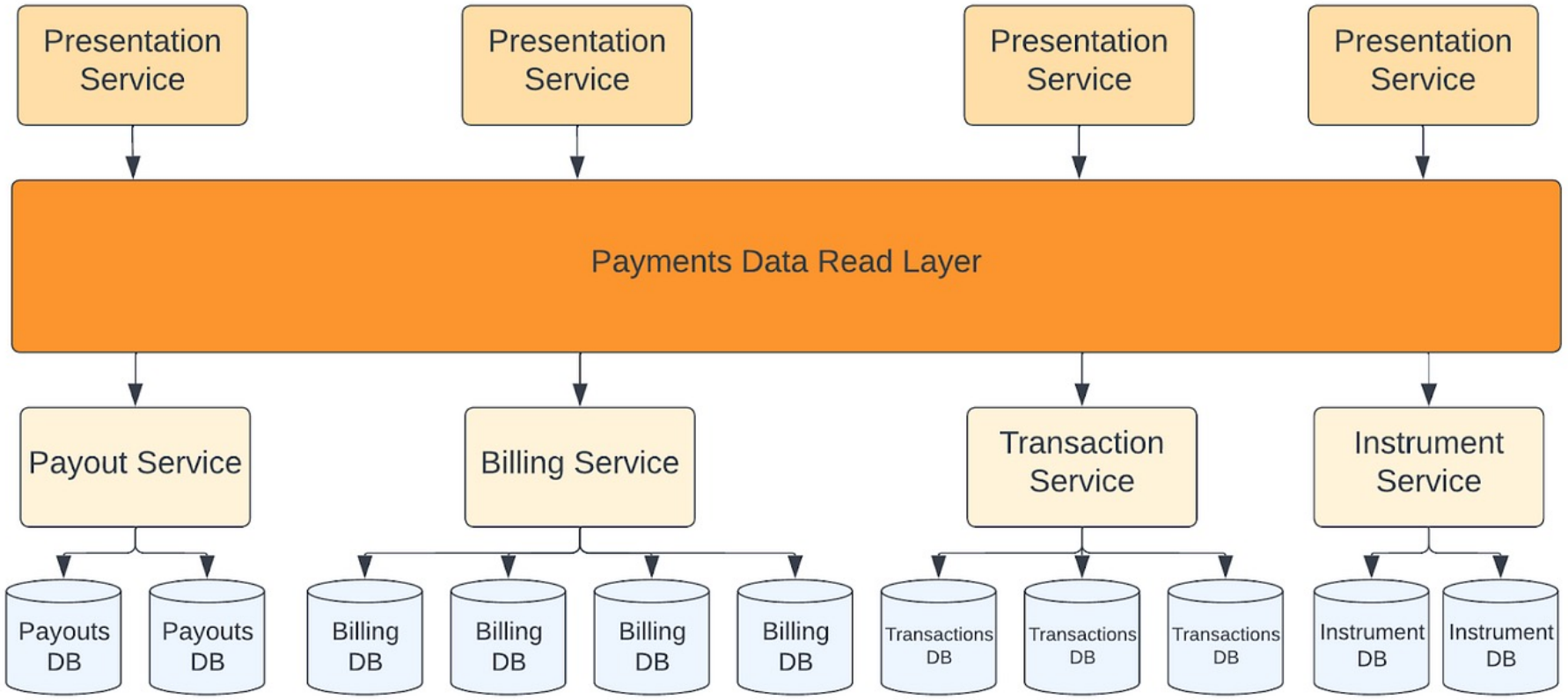
Payments data split across sub-domains

Requires cross-service data access



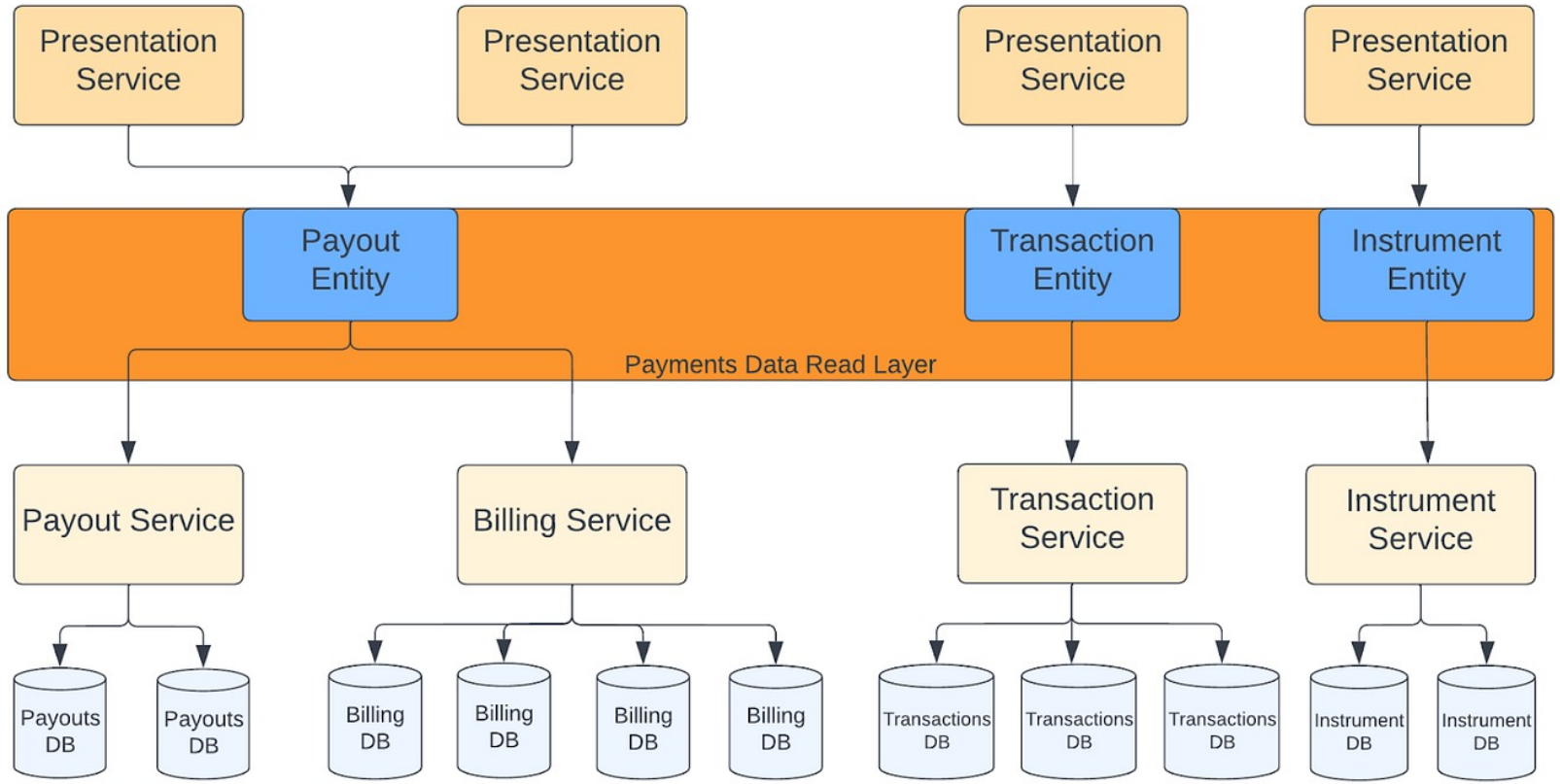
Which caused multiple service interaction for single use cases

Creating a new access layer



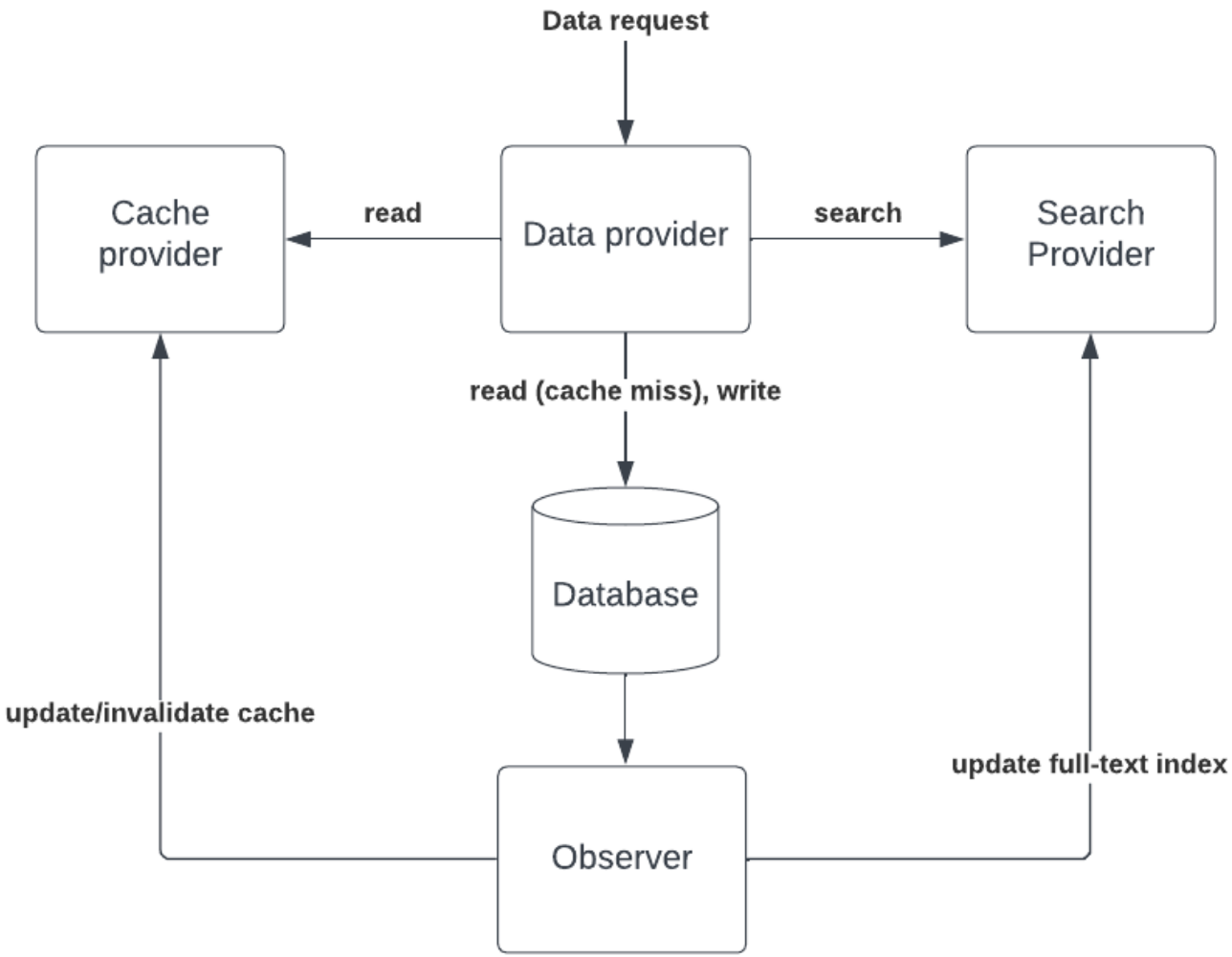
Expose entities instead of domain data to reduce API calls

Higher-level entities



Higher order entities to align with client needs

Transition system definition of the Data Access System



Designing data-intensive applications

5 systems

- Cache
- Search
- Database
- Data
- Observer

Cache tuples and ports

State space	$X = \{(key, value)\} \times valid, invalid$
Initial state space	$X^0 = Null$
Observation space	$Y = X$
Display map	$h = Id$
Action space	$U = Null$
Input ports	$P_{in} = \{(Data, key), (Observer, (key, value, datastatus))\}$
Output ports	$P_{out} = \{Data, (key, value, datastatus)\}$
Transition function	See Table 2

Table 1: Cache Provider System Description

Cache transition function

Guard	**Statement**
source = Data, X[key].status = ‘valid’	sendmsg(Data, (key, value, ‘found’)
source = Data, X[key] does not exist or X[key].status = ‘valid’	sendmsg(Data, (key, 0, ‘notfound’)
source = Observer, datastatus = ’valid’	X[key].value = value, X[key].status = ‘valid’
source = Observer, datastatus = ’invalid’	X[key].status = ‘invalid’

Table 2: Cache Provider System: Transition function. When Guard is true, the Statement is executed

Database ports

Input ports

$$P_{in} = \{(Data, (commandname, commanddata))\}$$

Output ports

$$P_{out} = \{ \\ (Data, (commandname, responsedata, responsestatus)), \\ (Observer, (changetype, data)) \\ \}$$

Database transition function

Guard	**Statement**
source = Data and commandname = 'Read'	applyDBcommand(commandname, commanddata); sendmsg(Data, ('Read', responsedata, responsestatus)
source = Data and commandname = 'Update'	applyDBcommand(commandname, commanddata); sendmsg(Data, ('Update', Null, responsestatus);sendmsg(Observer, ('Update', data))
source = Data and commandname = 'Create'	applyDBcommand(commandname, commanddata); sendmsg(Data, ('Create', Null, responsestatus);sendmsg(Observer, ('Create', data))
source = Data and commandname = 'Delete'	applyDBcommand(commandname, commanddata); sendmsg(Data, ('Delete', Null, responsestatus);sendmsg(Observer, ('Delete', data))

Data ports

Input ports

$$
\begin{aligned}
P_{in} = \{ & \\
& (CLIENT, (commandname, commanddata)), \\
& (Cache, (key, value, lookupstatus)), \\
& (Search, (searchstring, responsetext)), \\
& (Database, (commandname, responsedata, responsestatus)) \\
\} &
\end{aligned}
$$

Output ports

$$
\begin{aligned}
P_{out} = \{ & (Cache, key), \\
& (Search, searchstring), \\
& (Database, (commandname, commanddata)), \\
& (CLIENT, (commandname, responsedata, responsestatus)) \\
\} &
\end{aligned}
$$

Data transition system

Guard	**Statement**
source = CLIENT and commandname = 'Read'	sendmsg(Cache, getkey(commanddata))
source = CLIENT and commandname = 'Search'	sendmsg(Search, getsearch-string(commanddata))
source = CLIENT and commandname not in 'Read', 'Search'	sendmsg(Database, (commandname, commanddata))
source = Cache and lookupstatus = 'invalid'	sendmsg(Database, (commandname, commanddata))
(source = Cache and lookupstatus = 'valid') or (source = Search) or (source = Database)	sendmsg(CLIENT, (commandname, re-sponsedata, responsestatus))

Table 8: Data System Transition function. When Guard is true, the Statement is executed

Observer Ports

Input ports

$$P_{in} = \{(Database, (changetype, data))\}$$

Output ports

$$P_{out} = \{ \\ (Cache, (key, value, datastatus) \\ (Search, (key, text, status) \\ \}$$

Observer transition function

Guard	**Statement**
source = Data, X[key].status = ‘valid’	sendmsg(Data, (key, value, ‘found’)
source = Data, X[key] does not exist or X[key].status = ‘valid’	sendmsg(Data, (key, 0, ‘notfound’)
source = Observer, datastatus = ’valid’	X[key].value = value, X[key].status = ‘valid’
source = Observer, datastatus = ’invalid’	X[key].status = ‘invalid’

Key takeaways

Data parameters that impact the model (design-time)

Structure

File

Database

Message

Service

Schema

Tabular

Document

Name-value

Store

Partitioned

Replicated

Data parameters that impact the model (run-time)

Purpose

Transactional

Analytical

Batch

Pattern

Read vs. write

Data per access

Performance

Availability

Recovery

Response time

Classwork

Consider a scenario for a global application where we want to locate the data closer to the user by setting up more than one Data Centers (DC). For ex, North America users hit Virginia DC, India users hit Singapore DC, etc.

Each DC has one primary DB server which handles all writes and reads, and 1 secondary DB server which receives all the data updates from the primary DB of their DC and stays in sync.

All transactional read and write requests are directed to the primary DB and reports API requests (large dataset reads) are directed to the secondary DB server. A load balancing server takes care of routing the requests appropriately.

Primary DB servers in each DC forward the write requests to their counterparts in the other DCs so that they are in sync.

Tasks

Considering this a System of Systems, draw the component systems and their connections (assume asynchronous message interfaces). Also show the actions each system supports.

Write the flow of messages when a write request is made. Assume sendmsg(dst, msg), recvmsg(src, msg) primitives, name your src and dst systems as per your drawing.

Questions?