



MONOLITH TO MICROSERVICES

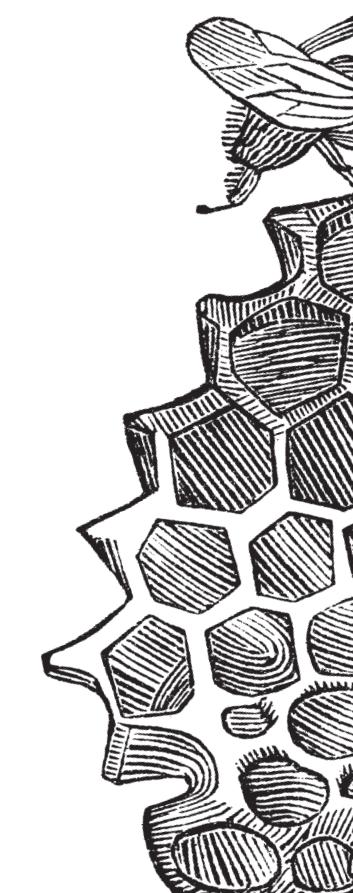
DATA DECOMPOSITION PATTERNS

Sam Newman

O'REILLY®

Building
Microservices

DESIGNING FINE-GRAINED SYSTEMS



O'REILLY®

Building Microservices

Designing Fine-Grained Systems



Sam Newman

Second
Edition

**Sam
Newman**
& Associates



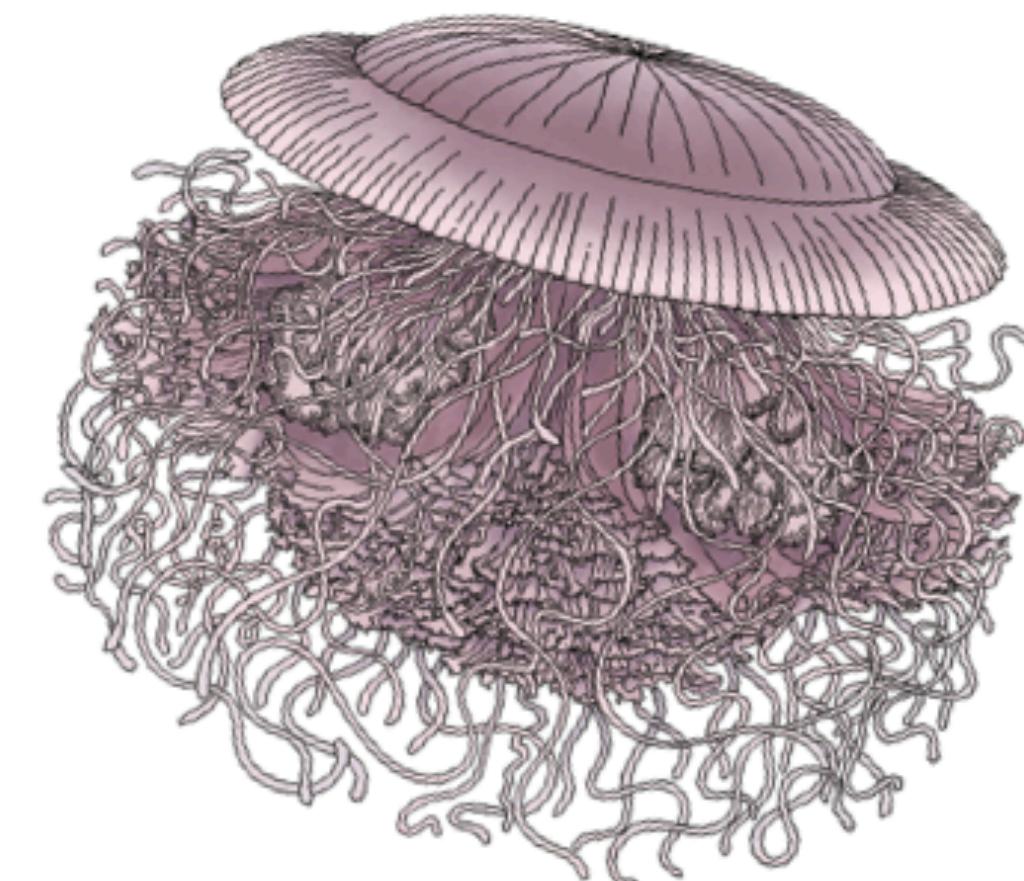
@samnewman

O'REILLY®

O'REILLY®

Monolith to Microservices

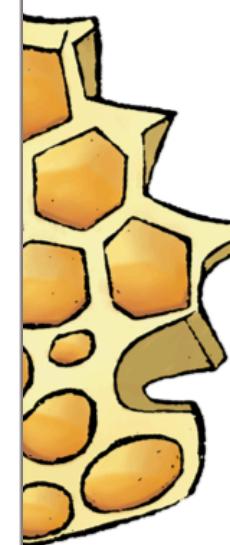
Evolutionary Patterns to Transform
Your Monolith



Sam Newman

Second
Edition

es



Sam Newman



**Sam
Newman
& Associates**



@samnewman

OTHER COURSES



<http://bit.ly/snewman-olt>

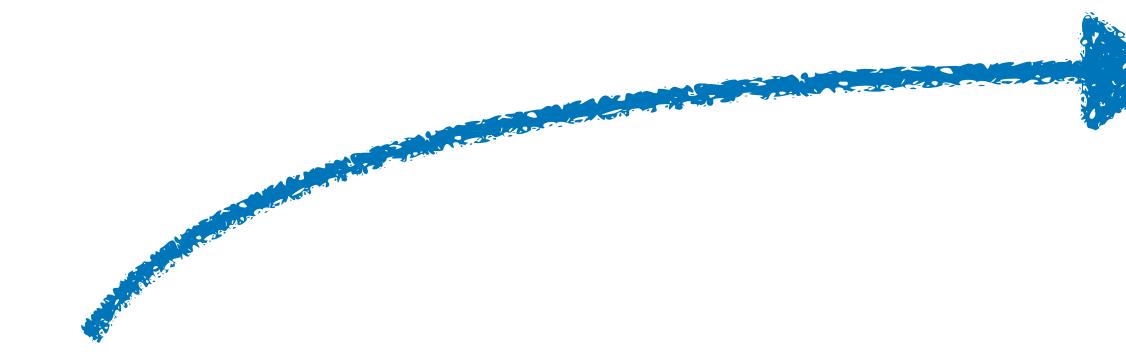
OTHER COURSES



MICROSERVICE FUNDAMENTALS

Sam Newman

<https://www.flickr.com/photos/snowpeak/42068450985/>



**MONOLITH TO MICROSERVICES
APPLICATION DECOMPOSITION PATTERNS**

Sam Newman

<https://www.flickr.com/photos/140077762@N04/37364813975/>

<http://bit.ly/snewman-olt>

@samnewman

OTHER COURSES



<http://bit.ly/snewman-olt>



3 hour session

3 hour session

Two 10-15 breaks

3 hour session

Two 10-15 breaks

Please ask questions using the Q&A widget

The session will be recorded, and
recordings will be available 24-48
hours after the class ends

AGENDA

AGENDA

Introduction

AGENDA

Introduction

Shared Data Patterns

AGENDA

Introduction

Shared Data Patterns

Migration Order

AGENDA

Introduction

Shared Data Patterns

Migration Order

Decomposition Patterns

AGENDA

Introduction

Shared Data Patterns

Migration Order

Decomposition Patterns

POLL: WHAT IS YOUR EXPERIENCE LEVEL WITH MICROSERVICES?

Only thinking about using them

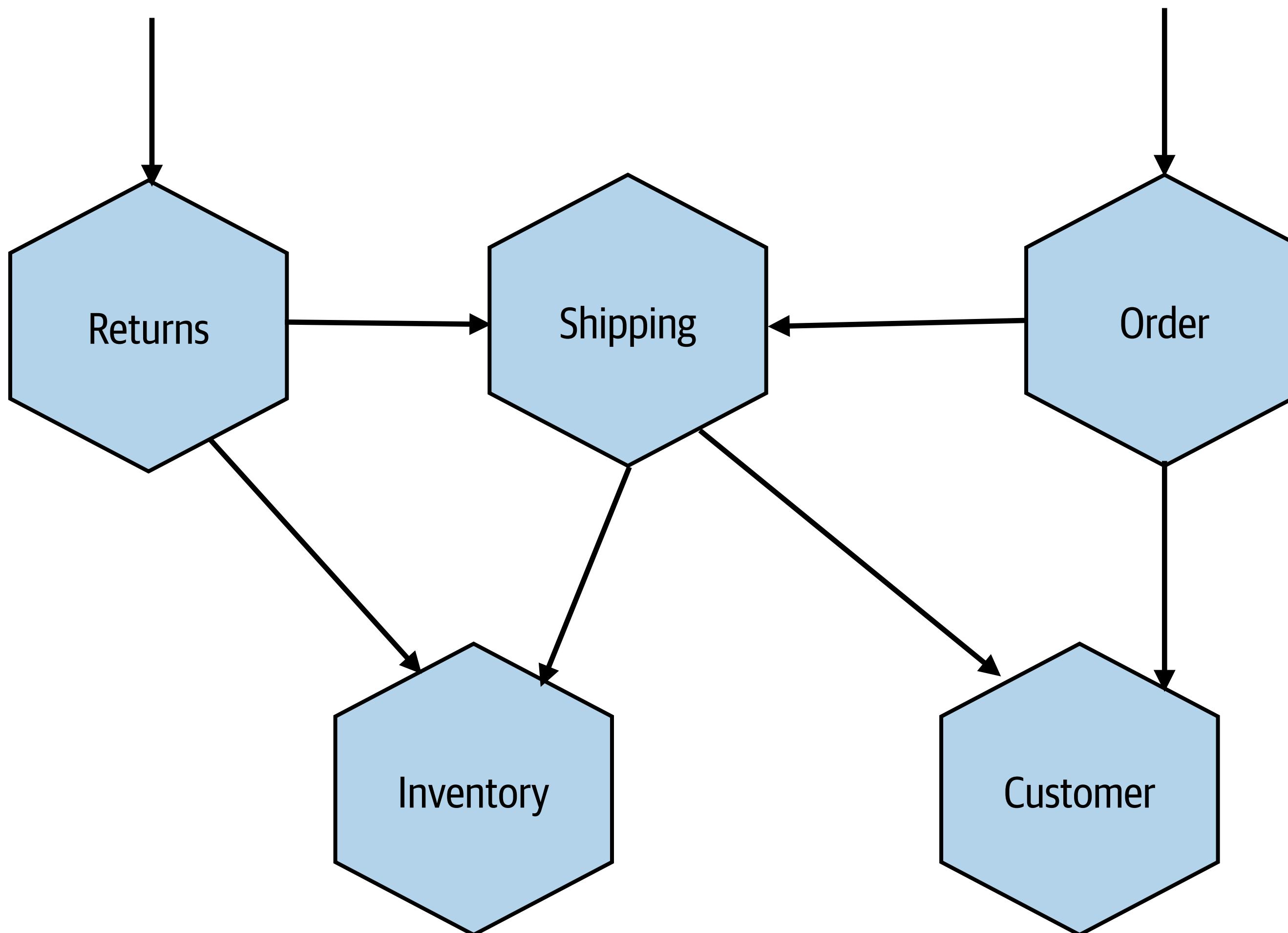
Just started using them

1-2 years experience

2+ years experience

What are microservices?

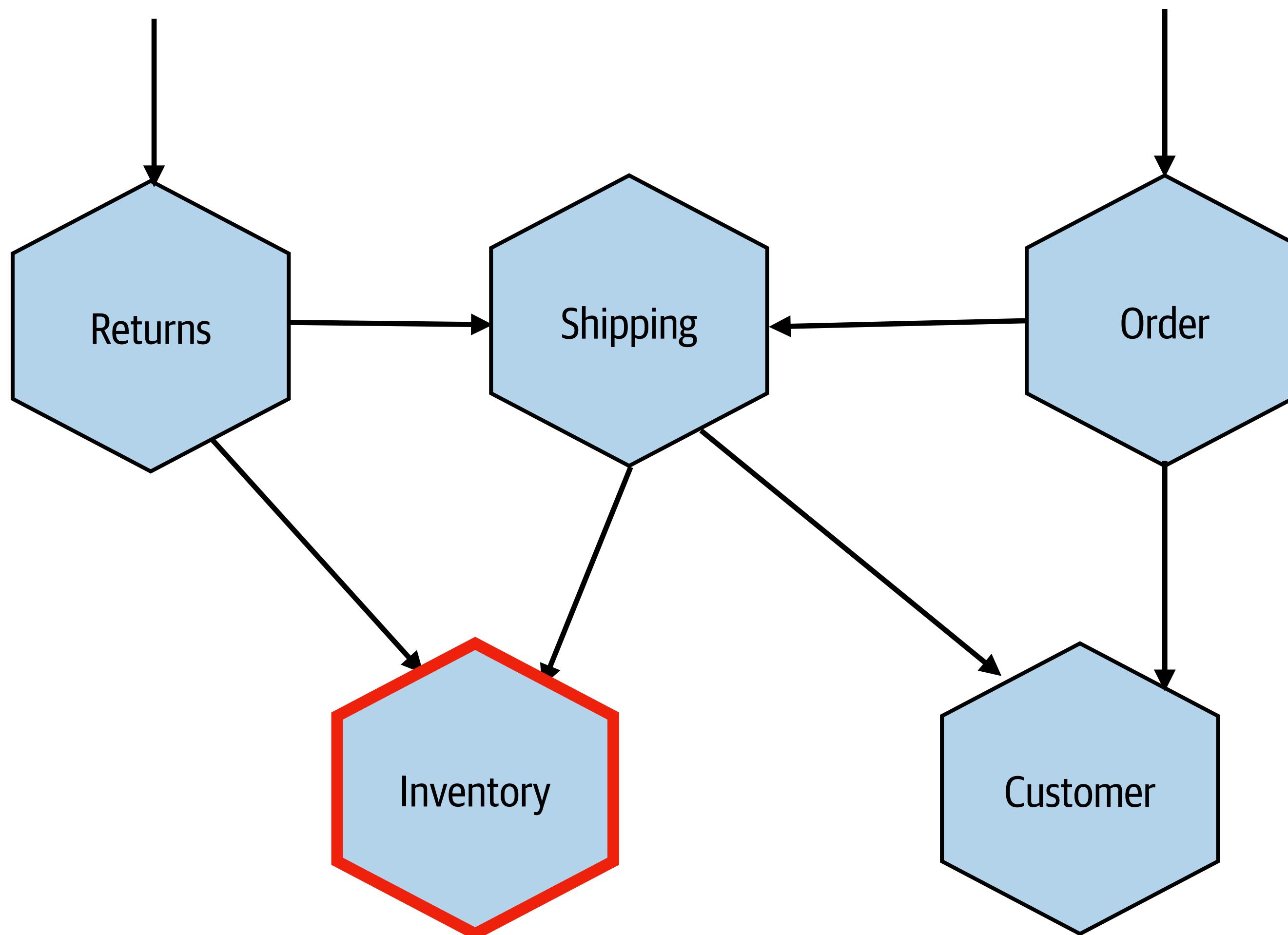
INDEPENDENT DEPLOYABILITY



When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

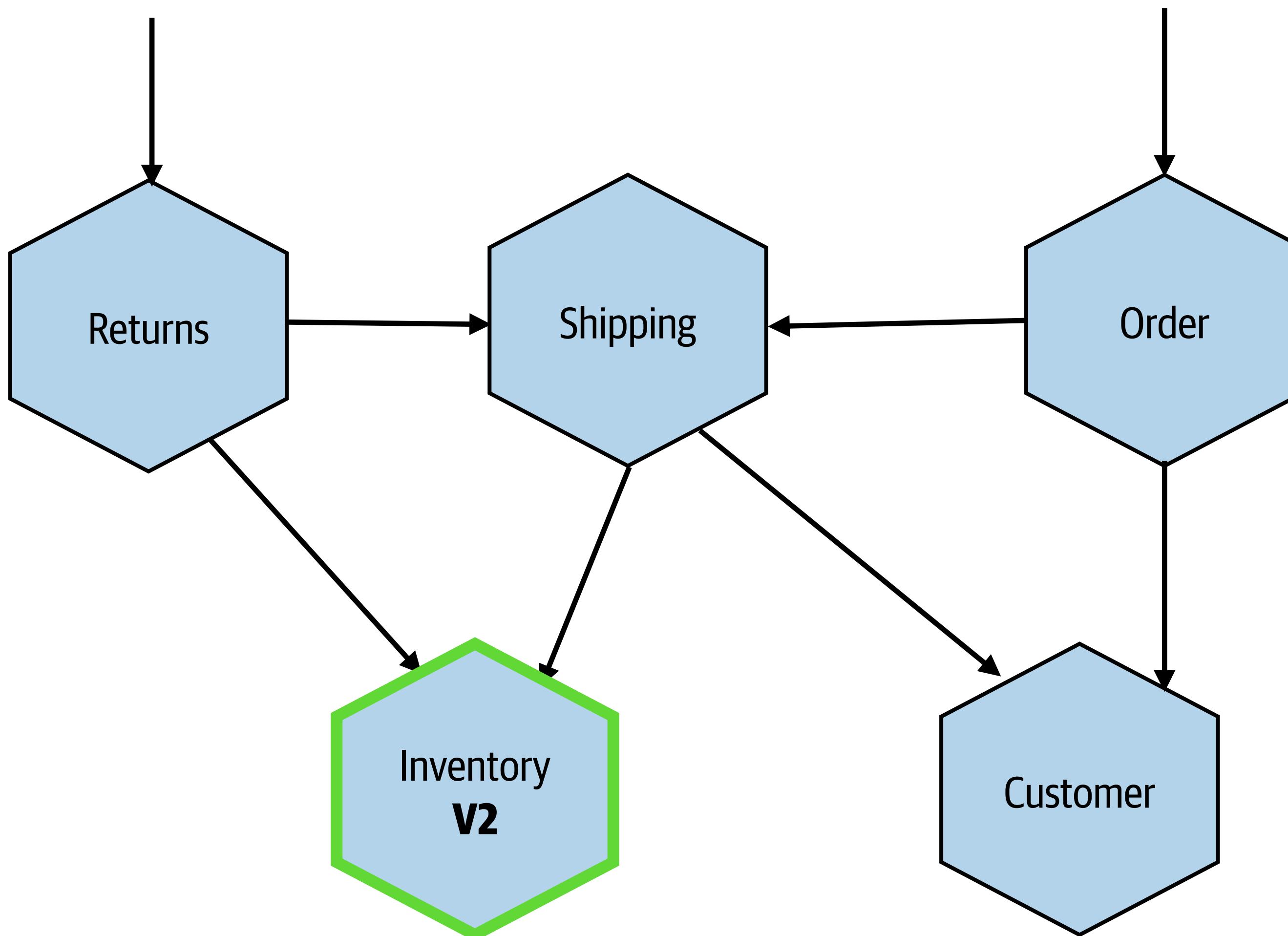
INDEPENDENT DEPLOYABILITY



When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

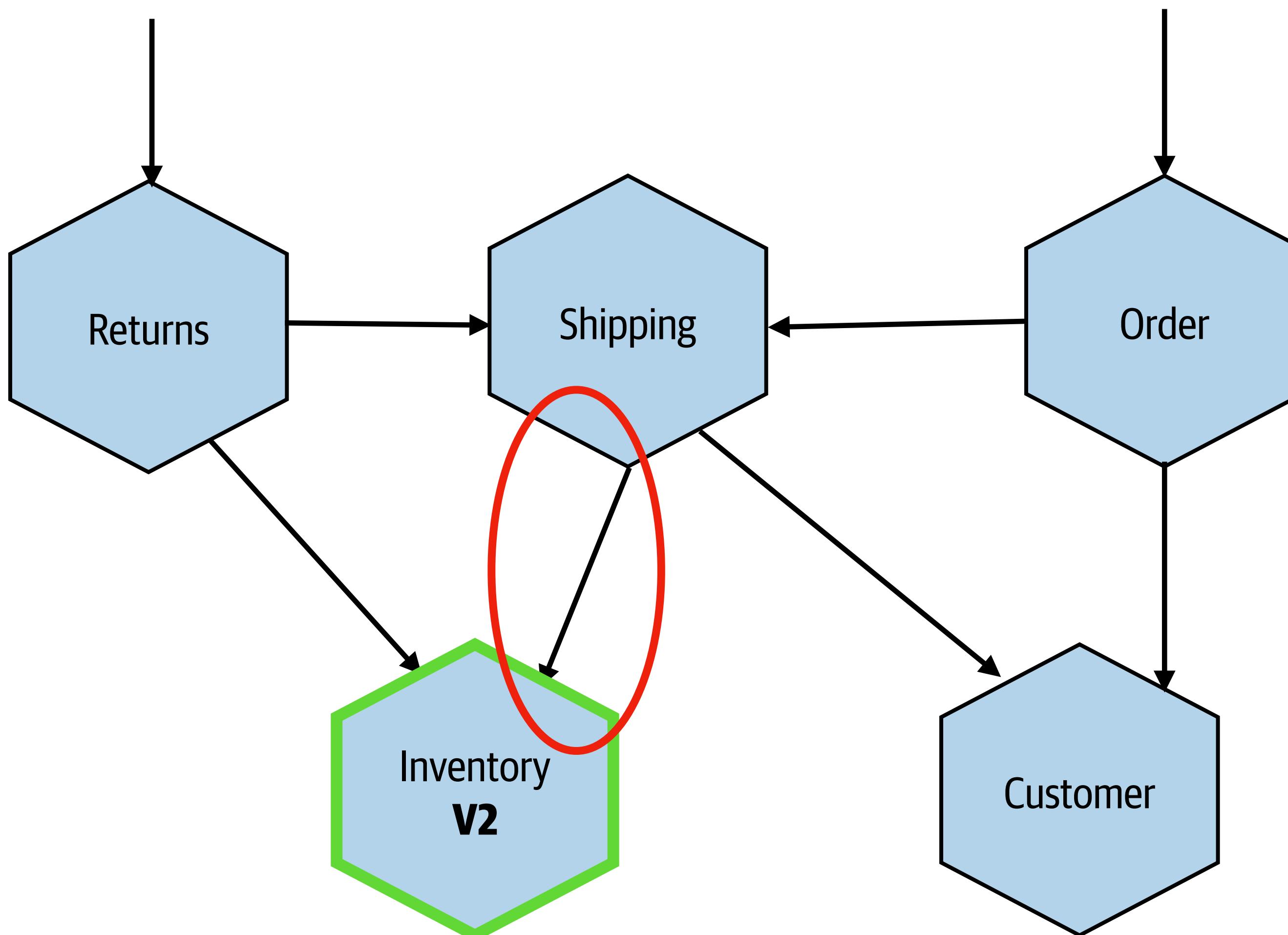
INDEPENDENT DEPLOYABILITY



When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

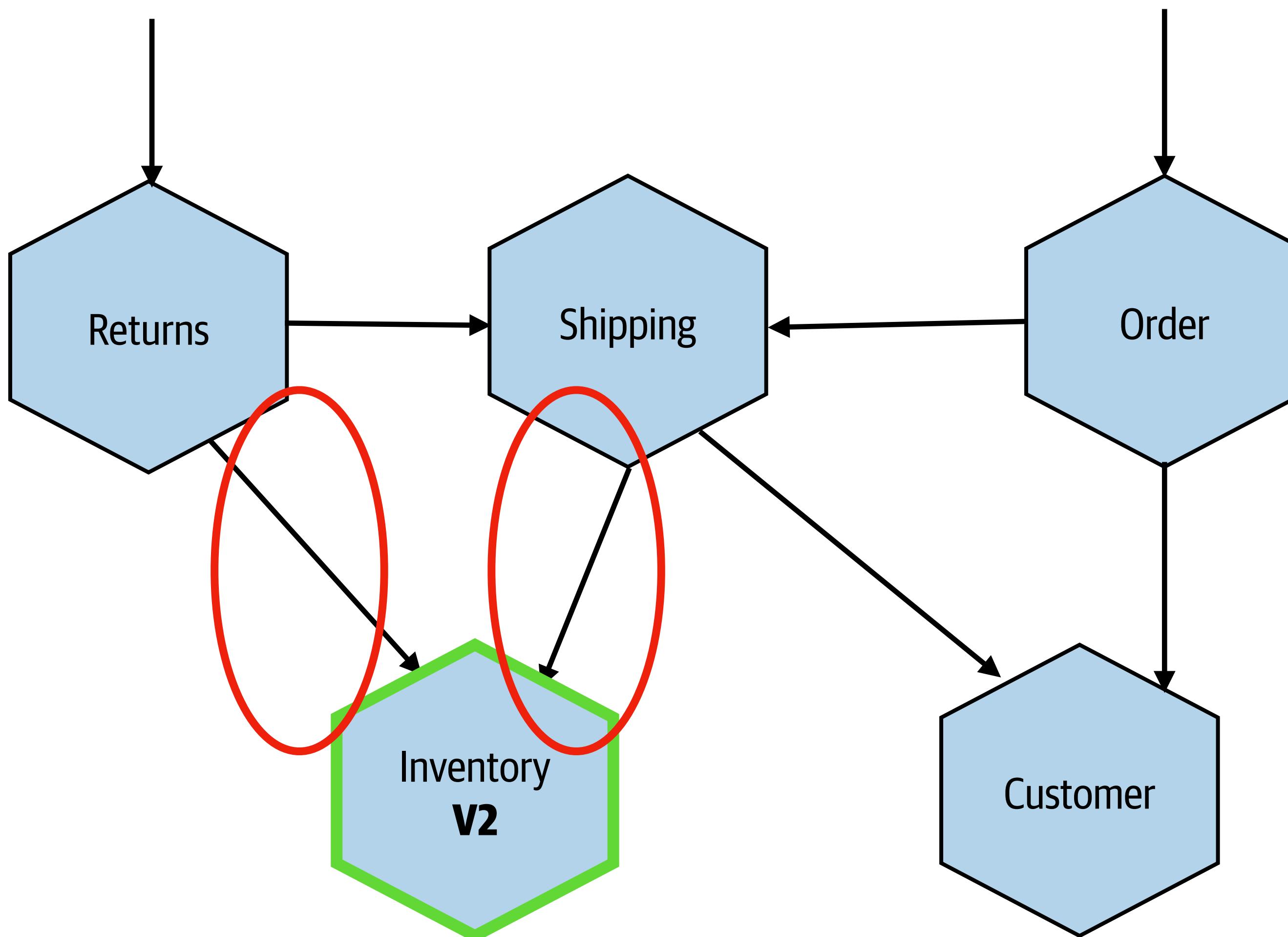
INDEPENDENT DEPLOYABILITY



When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

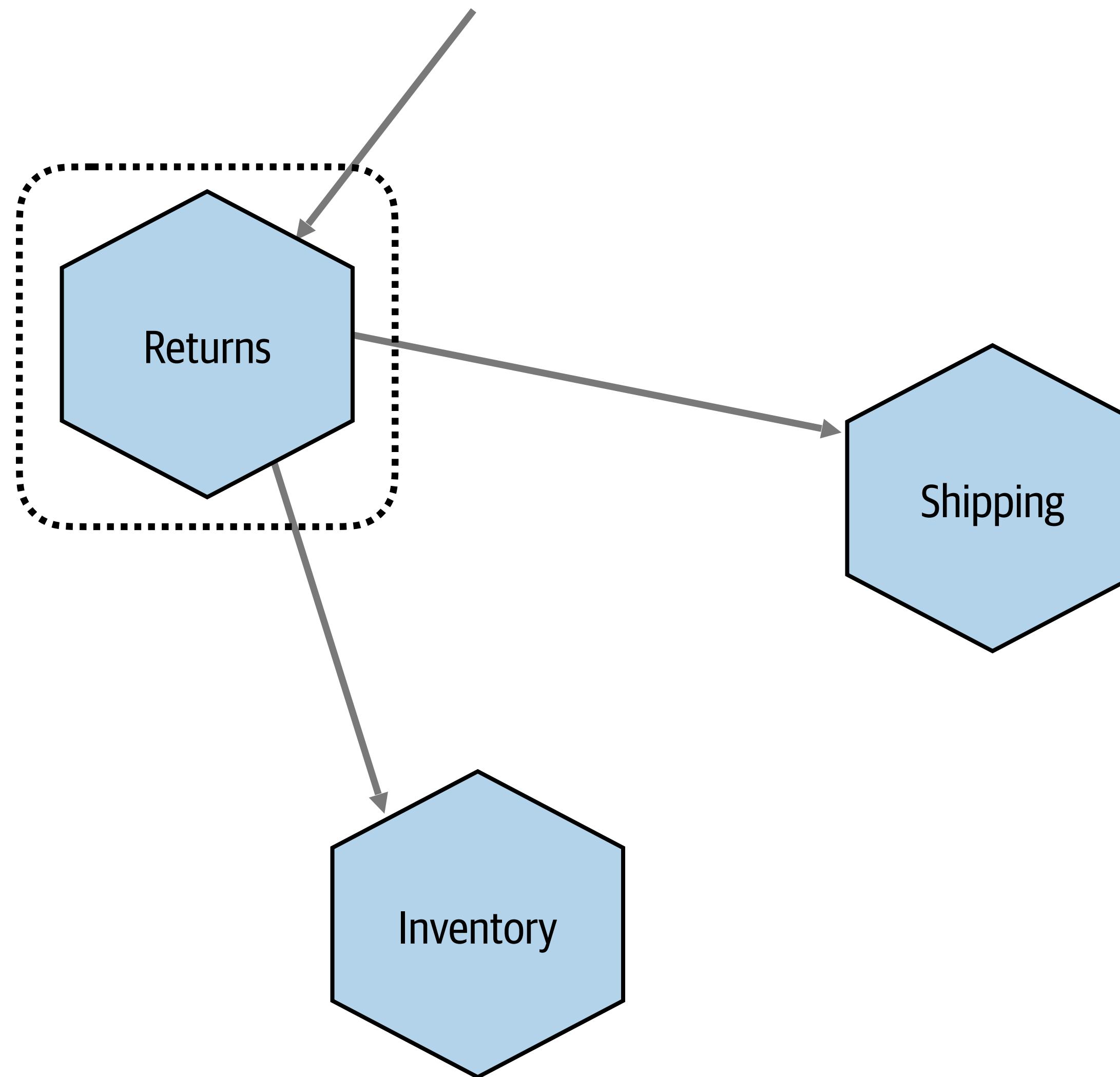
INDEPENDENT DEPLOYABILITY



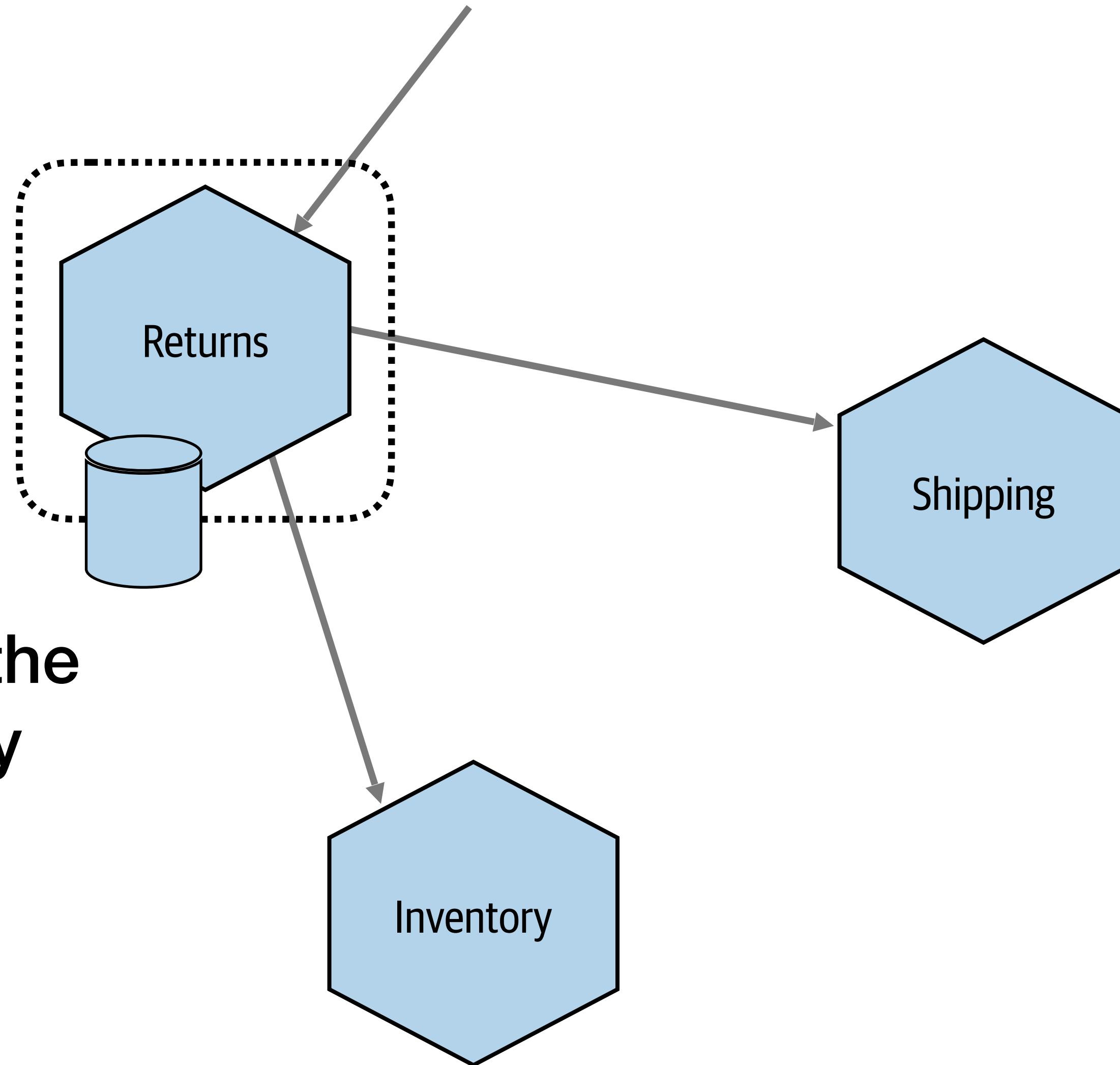
When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

DATA IS HIDDEN...

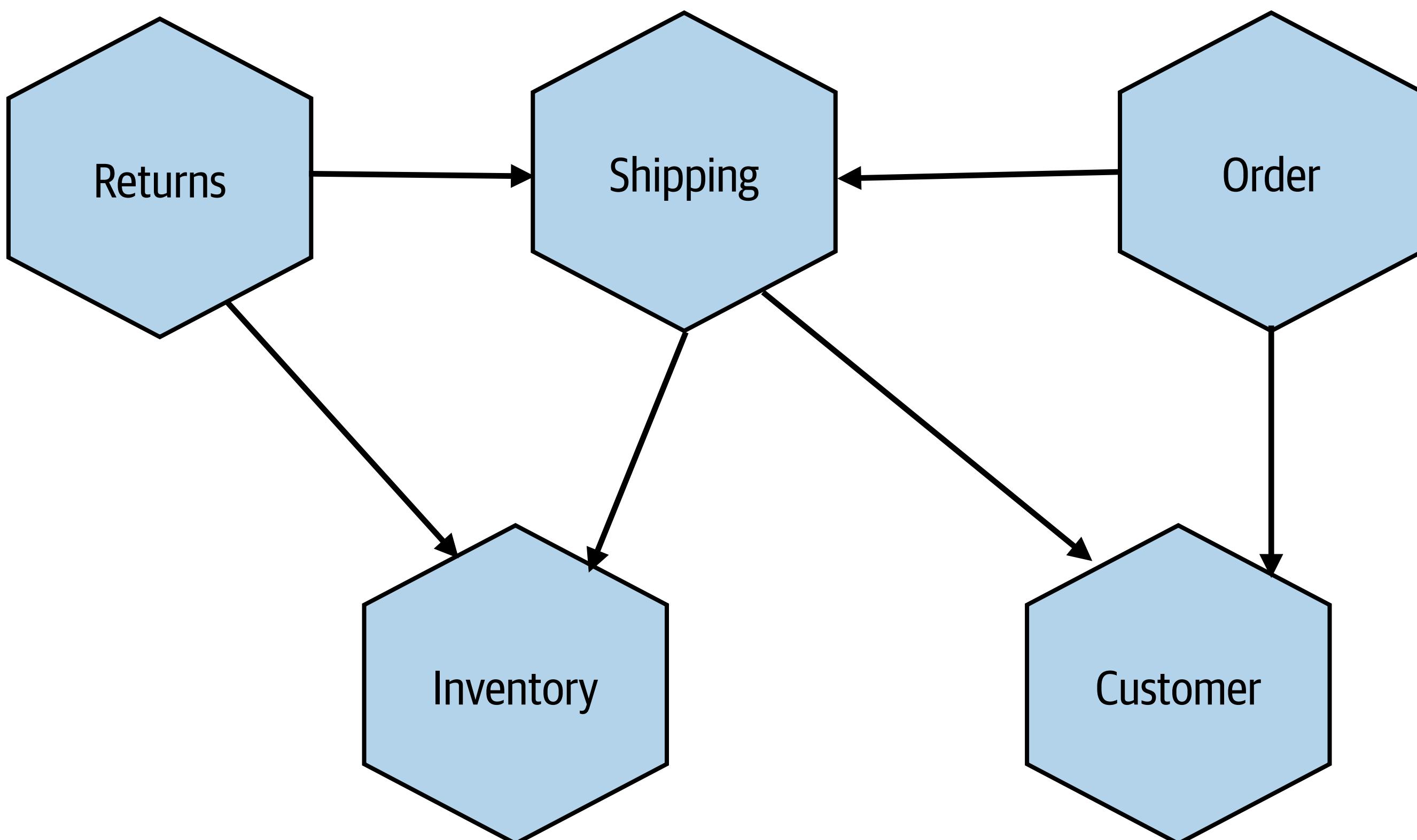


DATA IS HIDDEN...

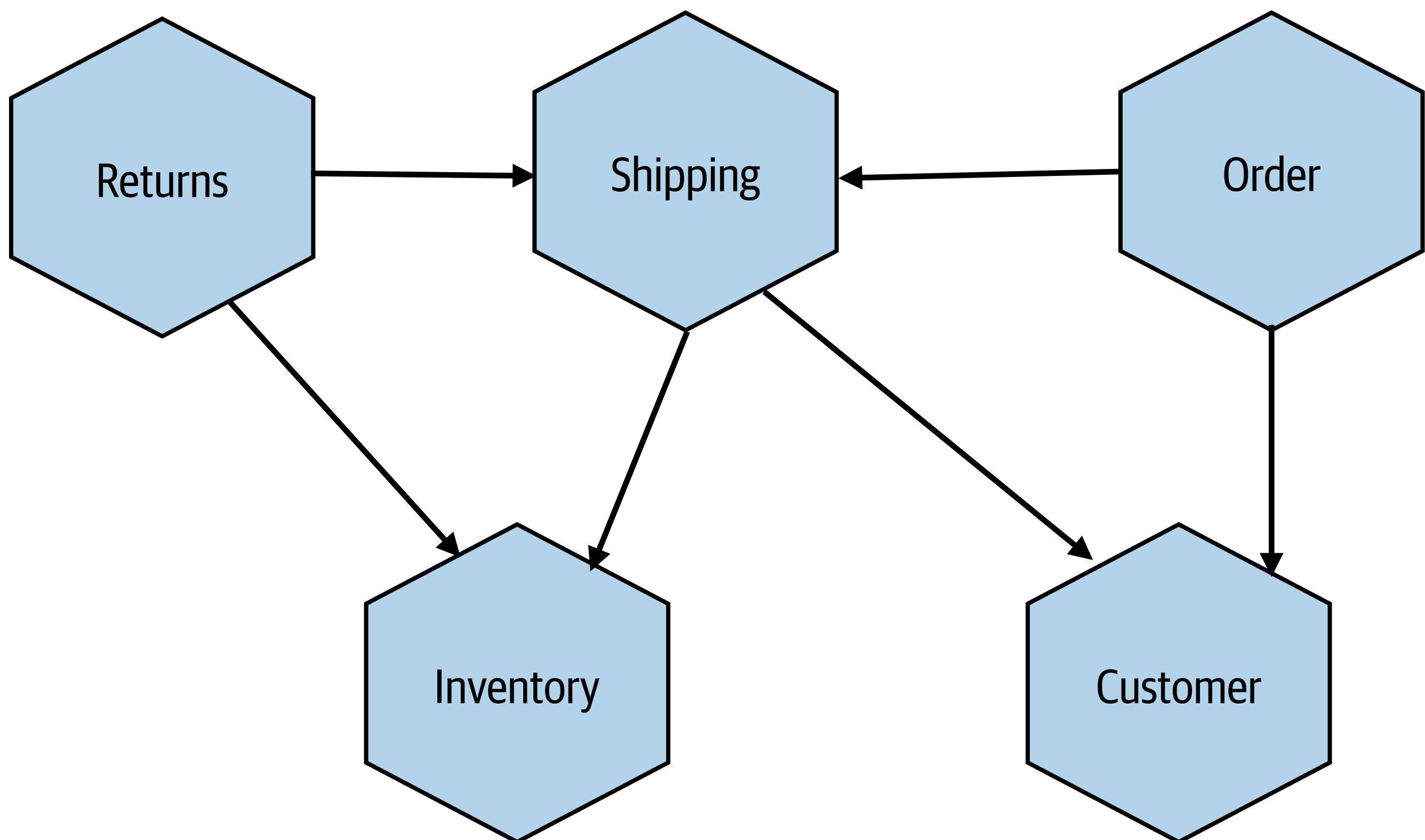


Data is hidden inside the microservice boundary

DATABASES

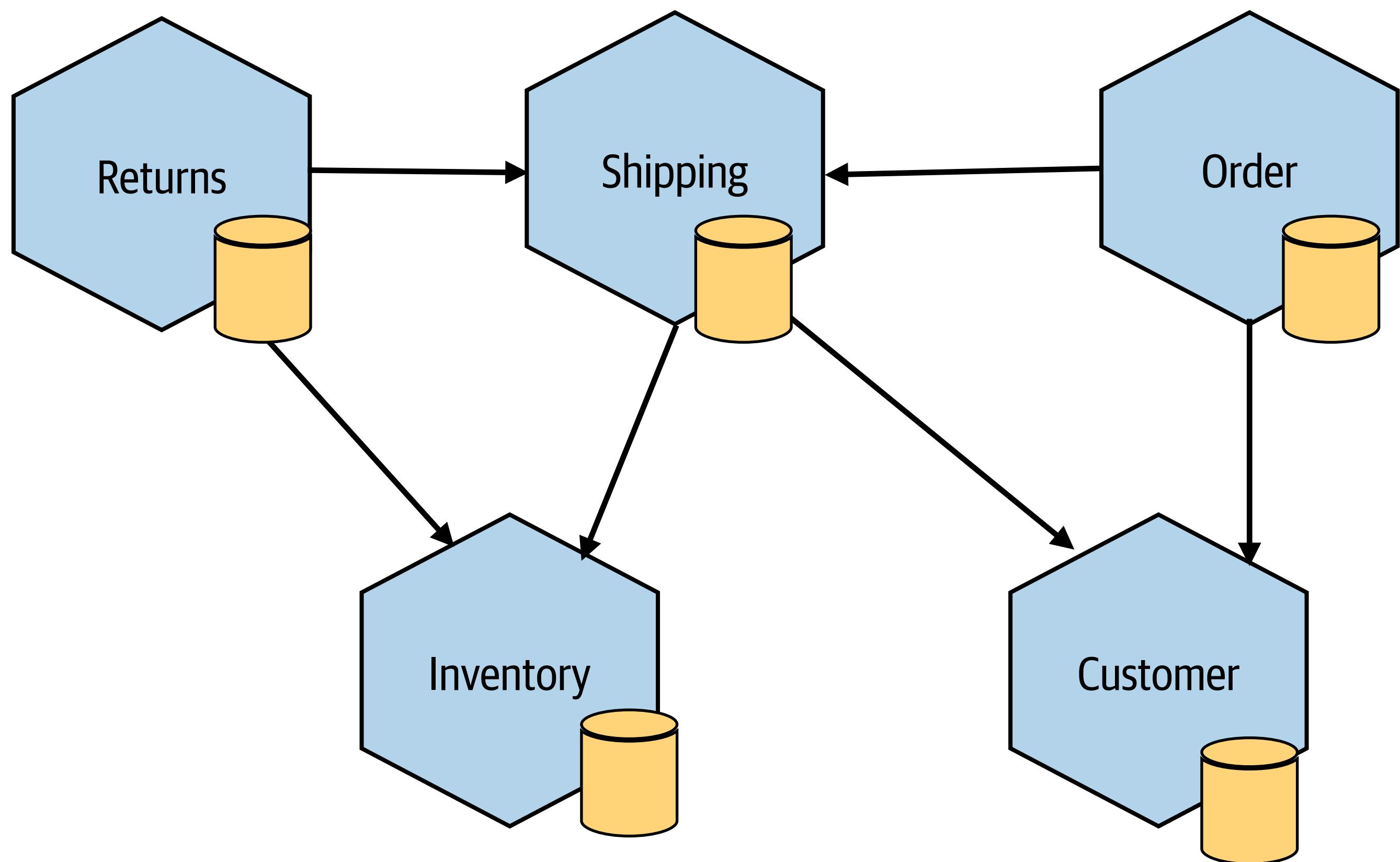


DATABASES



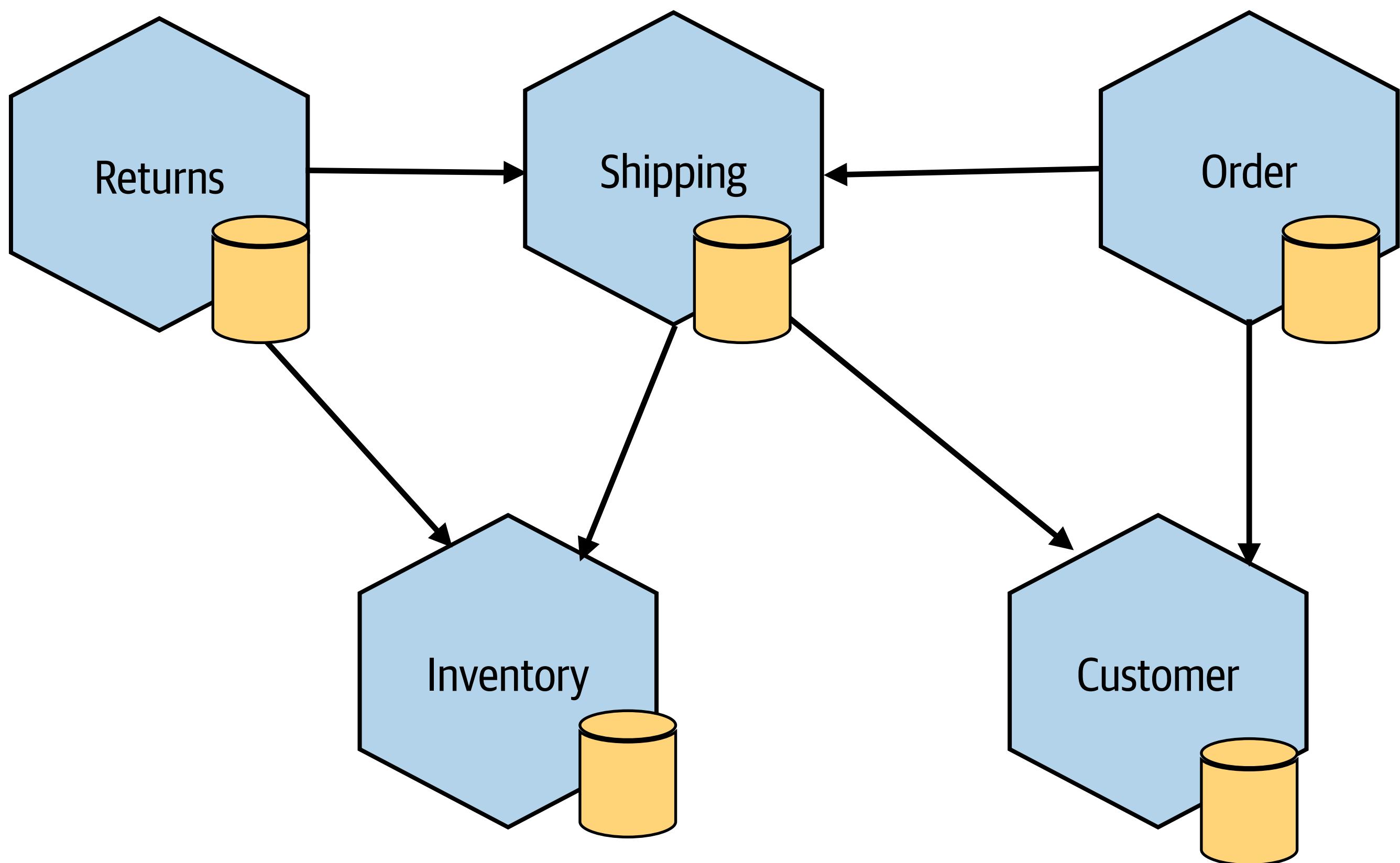
If microservices need to use a database to manage their state, this is hidden from other consumers

DATABASES



If microservices need to use a database to manage their state, this is hidden from other consumers

DATABASES



If microservices need to use a database to manage their state, this is hidden from other consumers

Does that mean lots of DB infrastructure to manage?

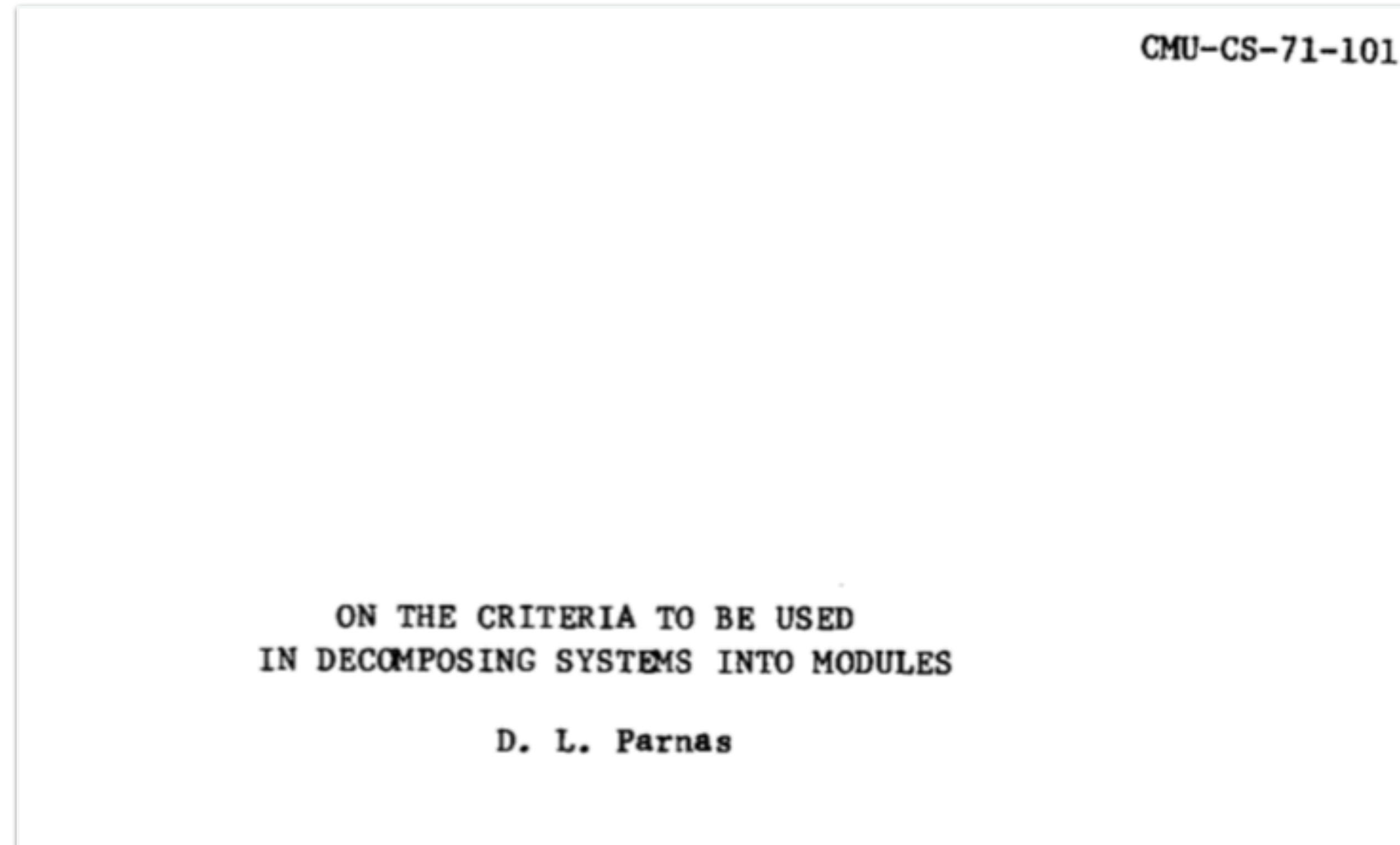
POLL: HOW MANY OF YOU HAVE ALL YOUR DATA IN A SINGLE DATABASE?

Yes

No

Not Sure!

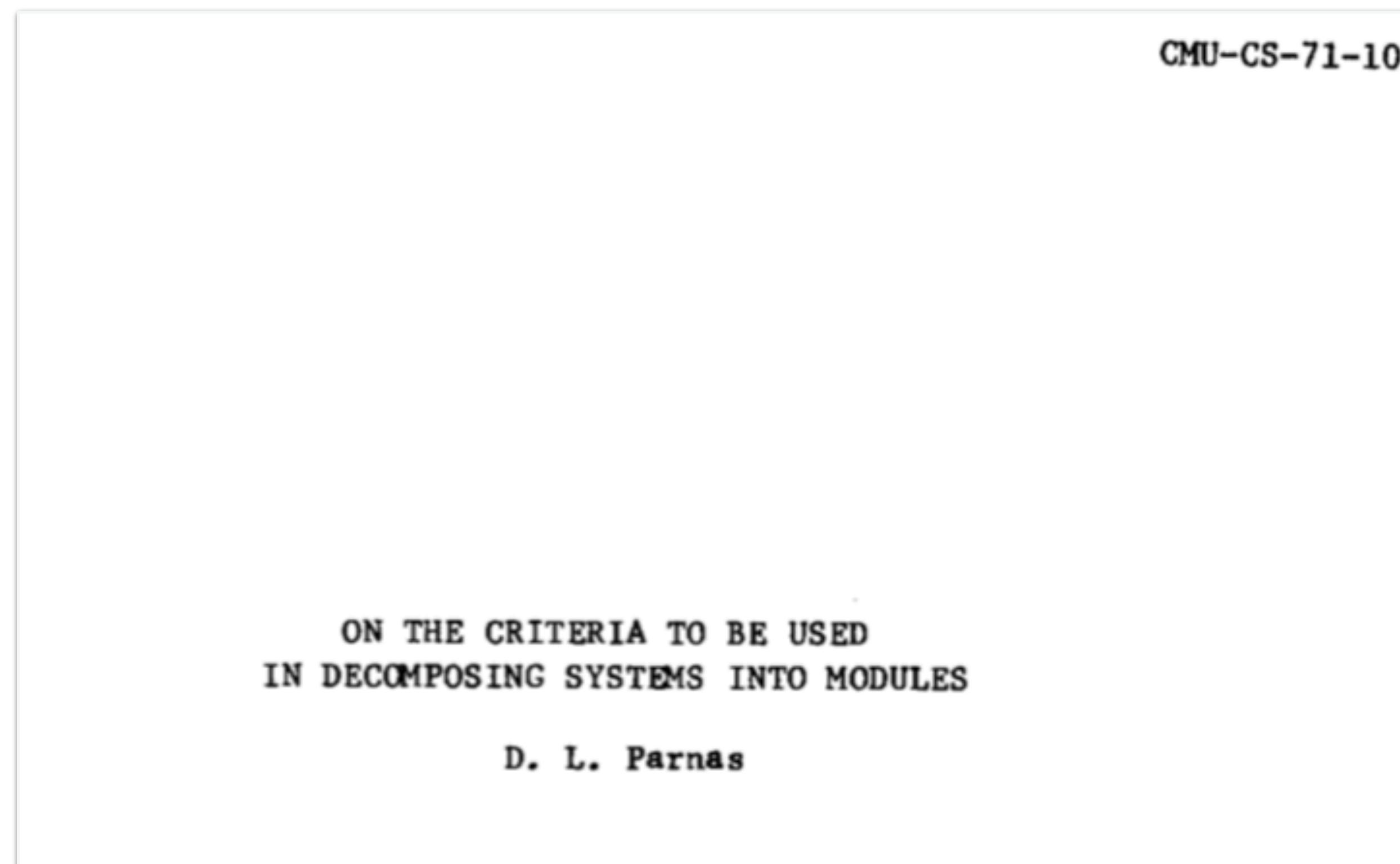
INFORMATION HIDING



<http://repository.cmu.edu/cgi/viewcontent.cgi?article=2979&context=compsci>

@samnewman

INFORMATION HIDING

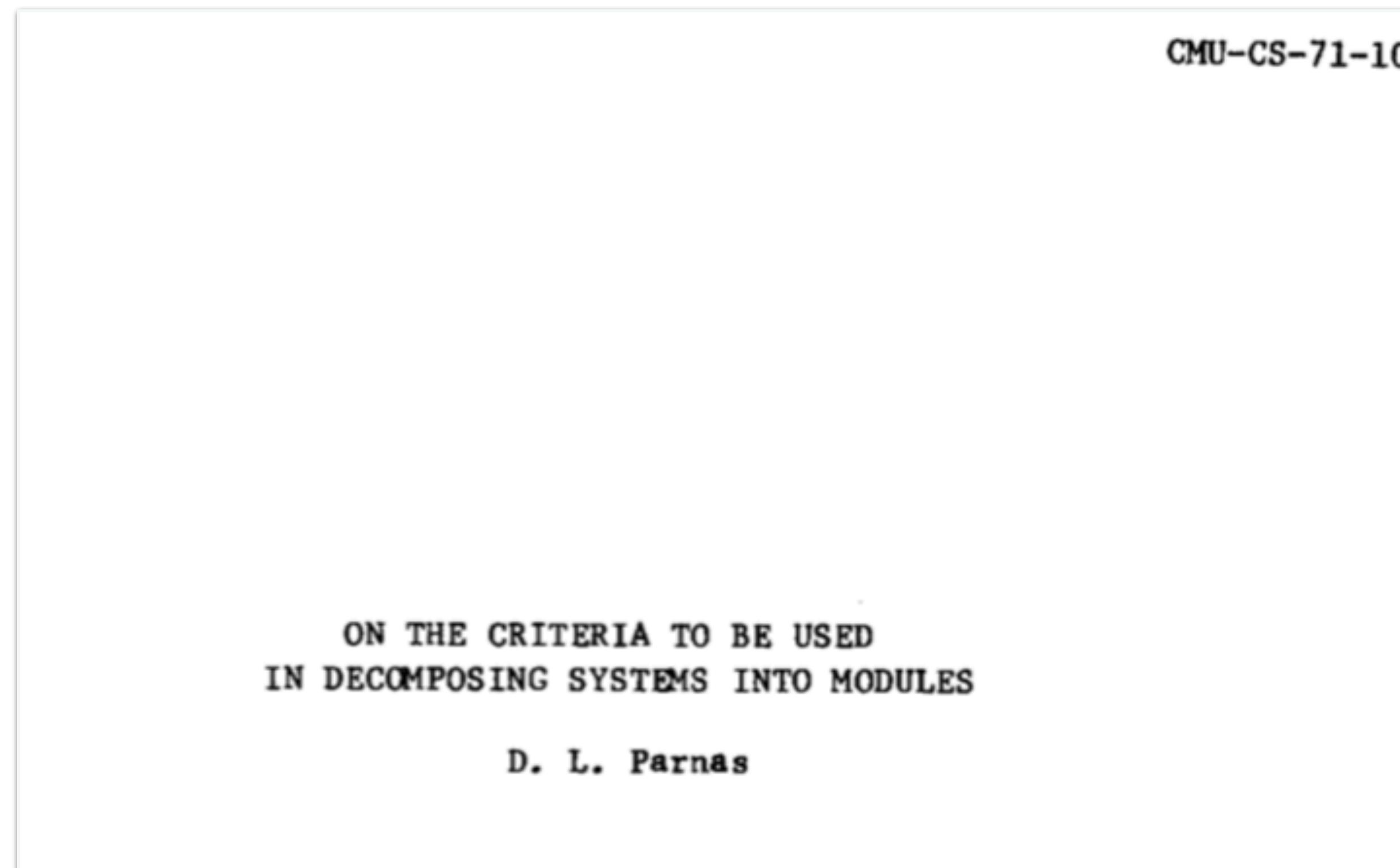


Published in 1971

<http://repository.cmu.edu/cgi/viewcontent.cgi?article=2979&context=compsci>

@samnewman

INFORMATION HIDING



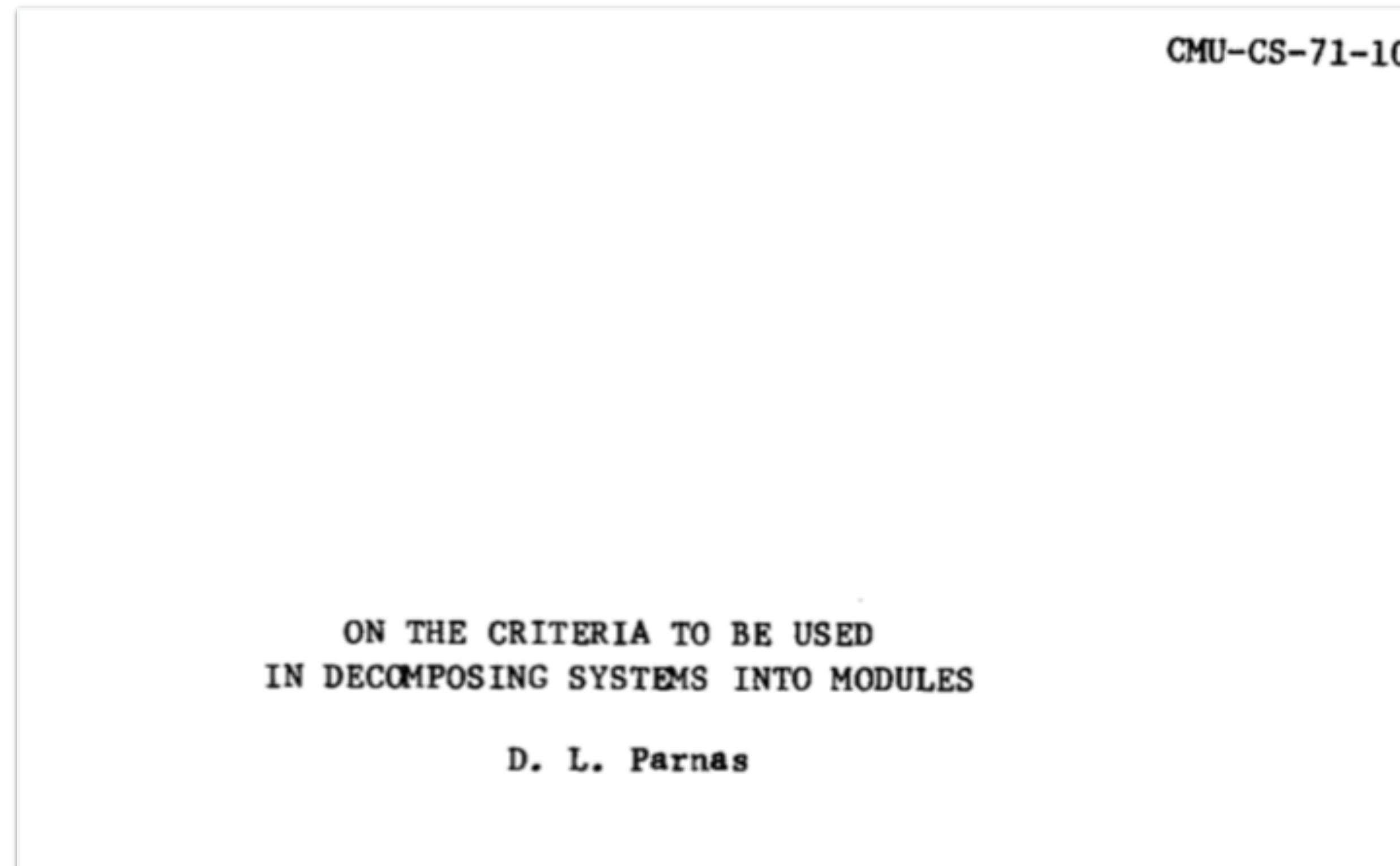
Published in 1971

Looked at how best to define
module boundaries

<http://repository.cmu.edu/cgi/viewcontent.cgi?article=2979&context=compsci>

@samnewman

INFORMATION HIDING

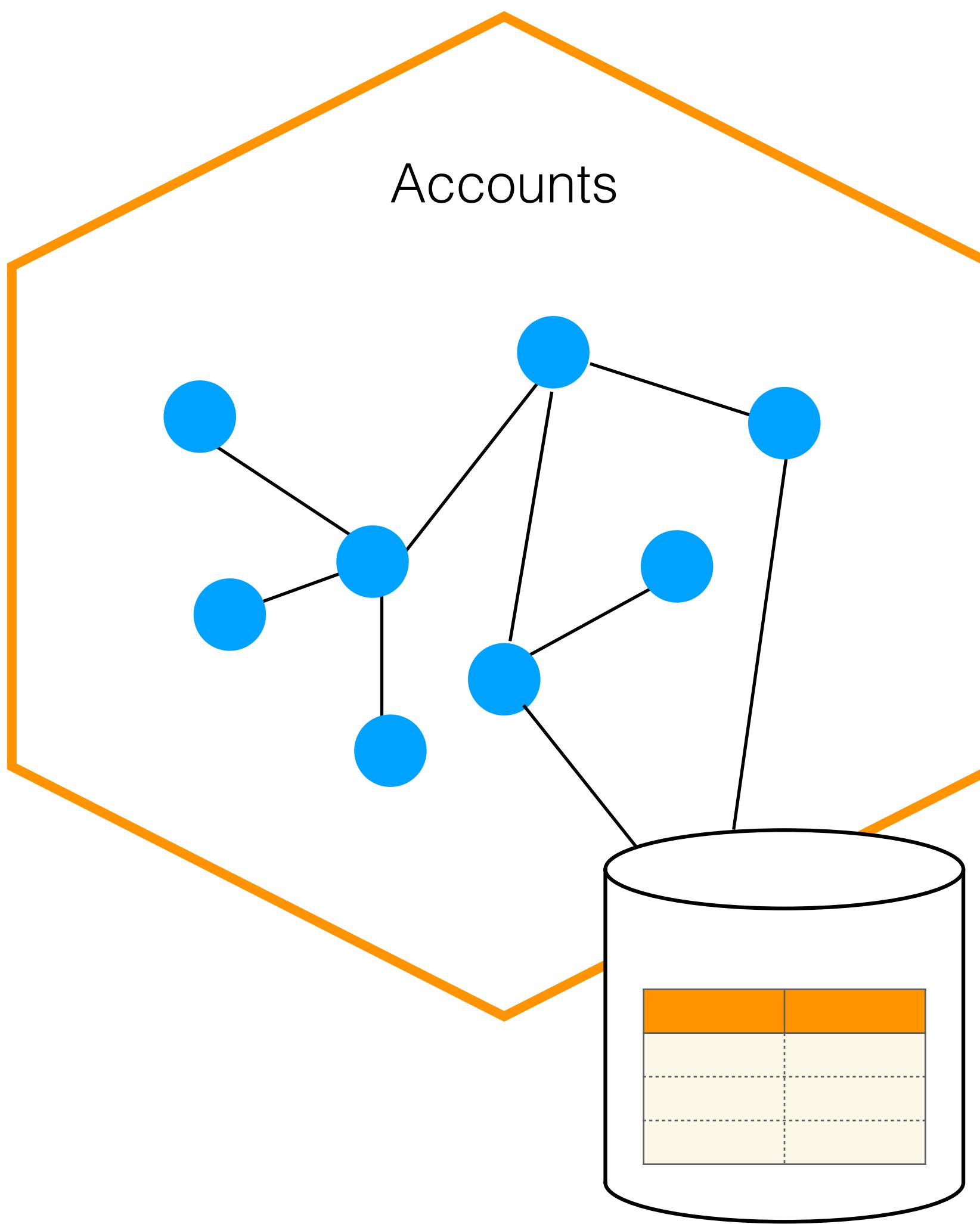


Published in 1971

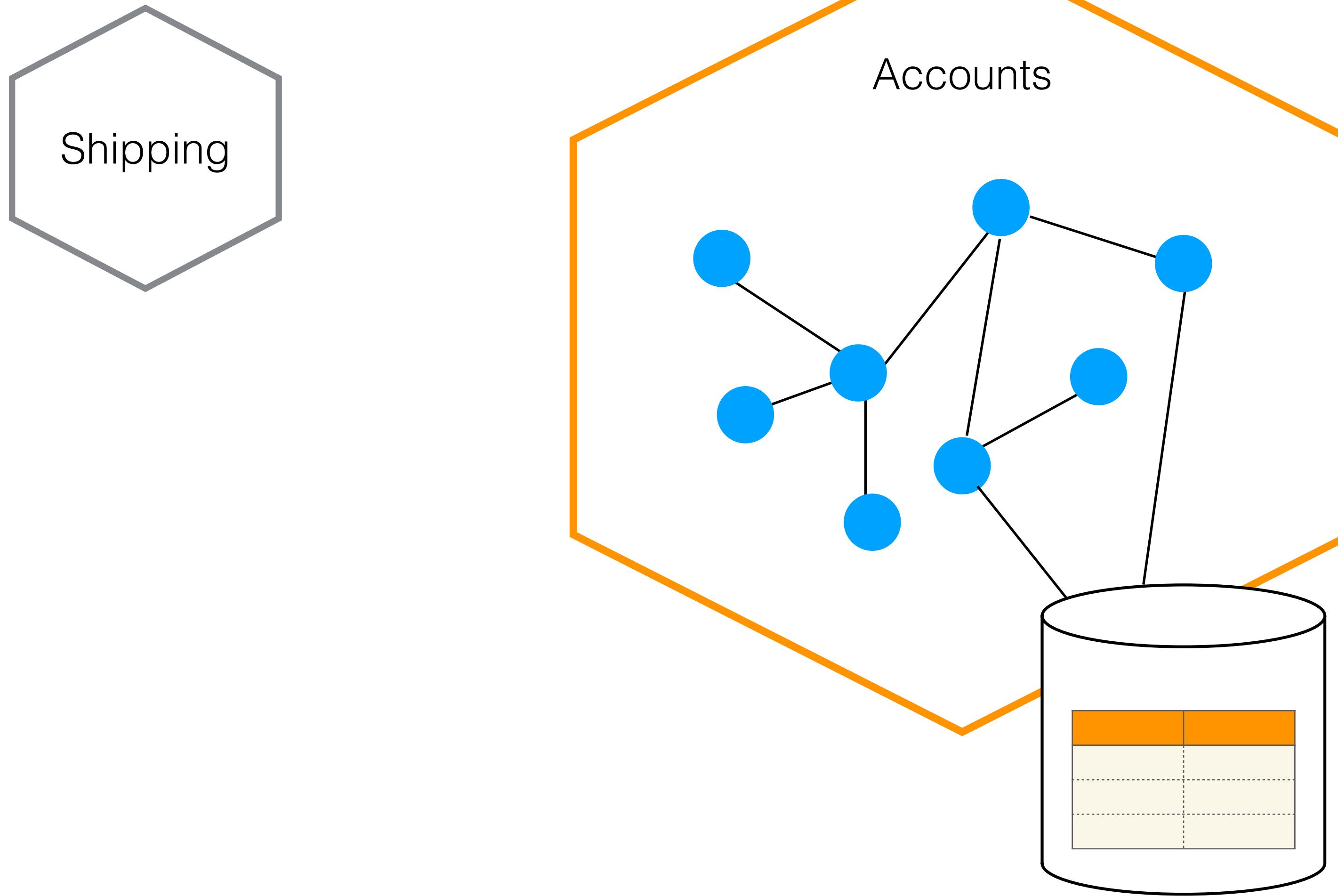
Looked at how best to define module boundaries

Found that “information hiding” worked best

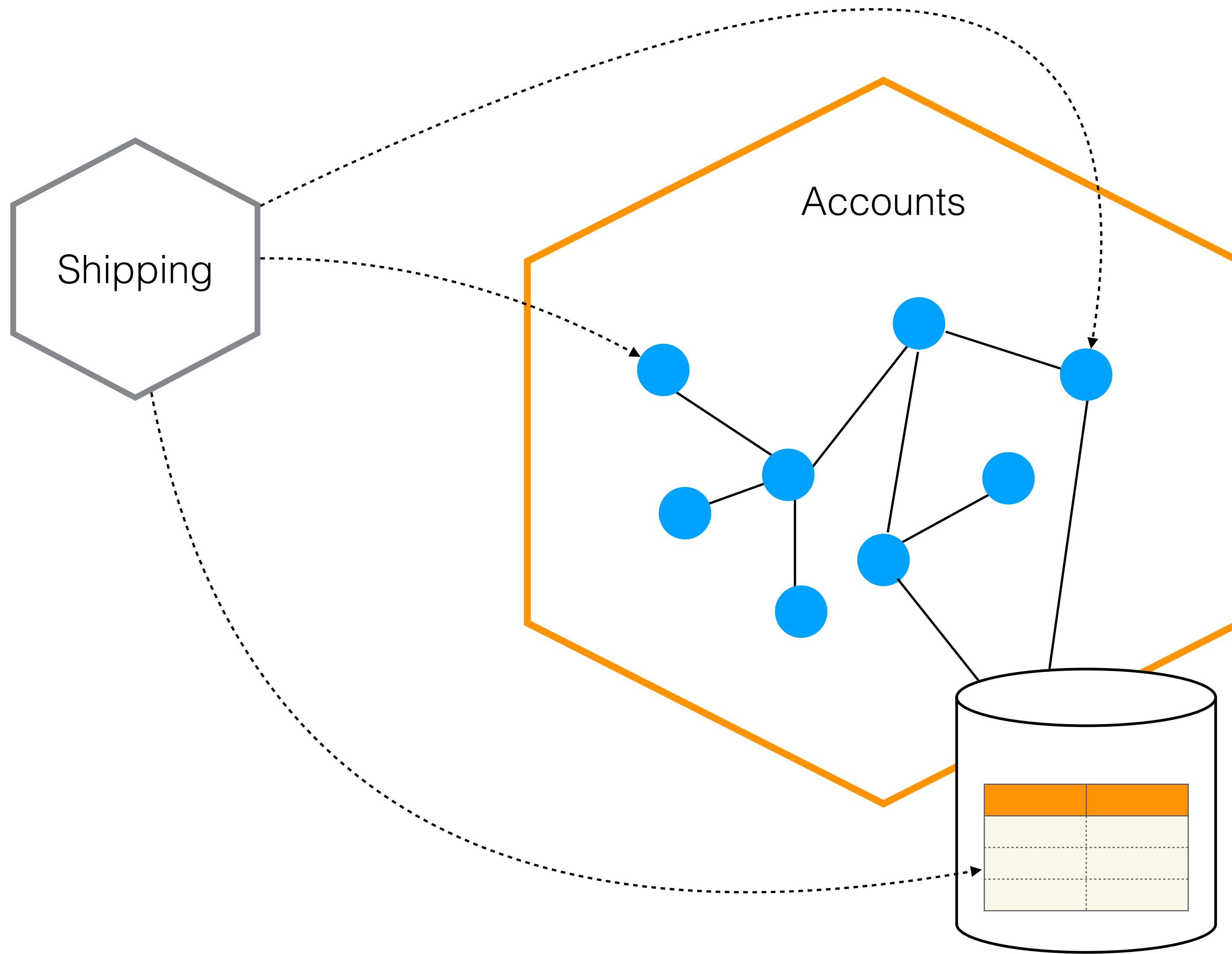
NOT HIDING?



NOT HIDING?

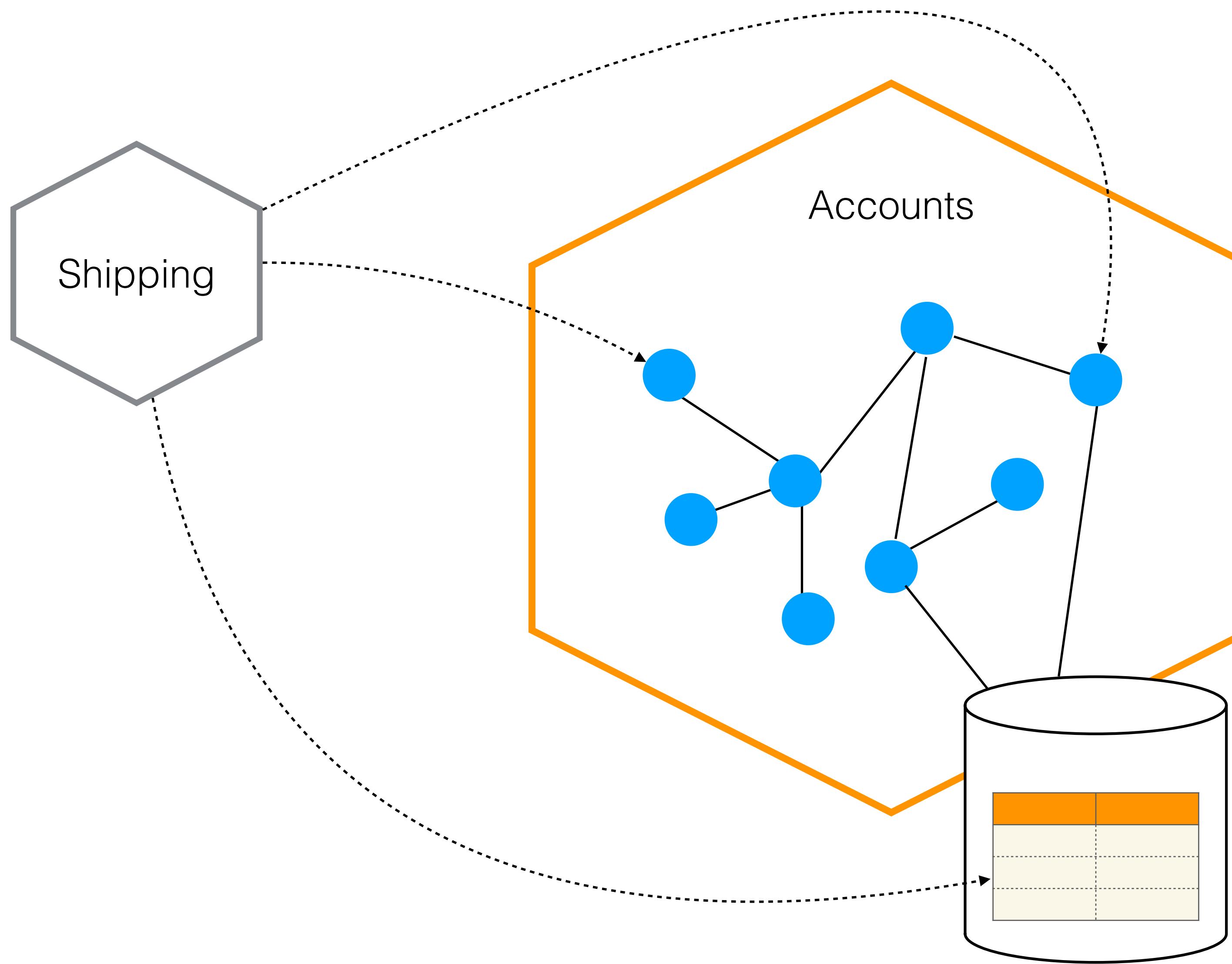


NOT HIDING?



If an upstream consumer can
reach into your internal
implementation..

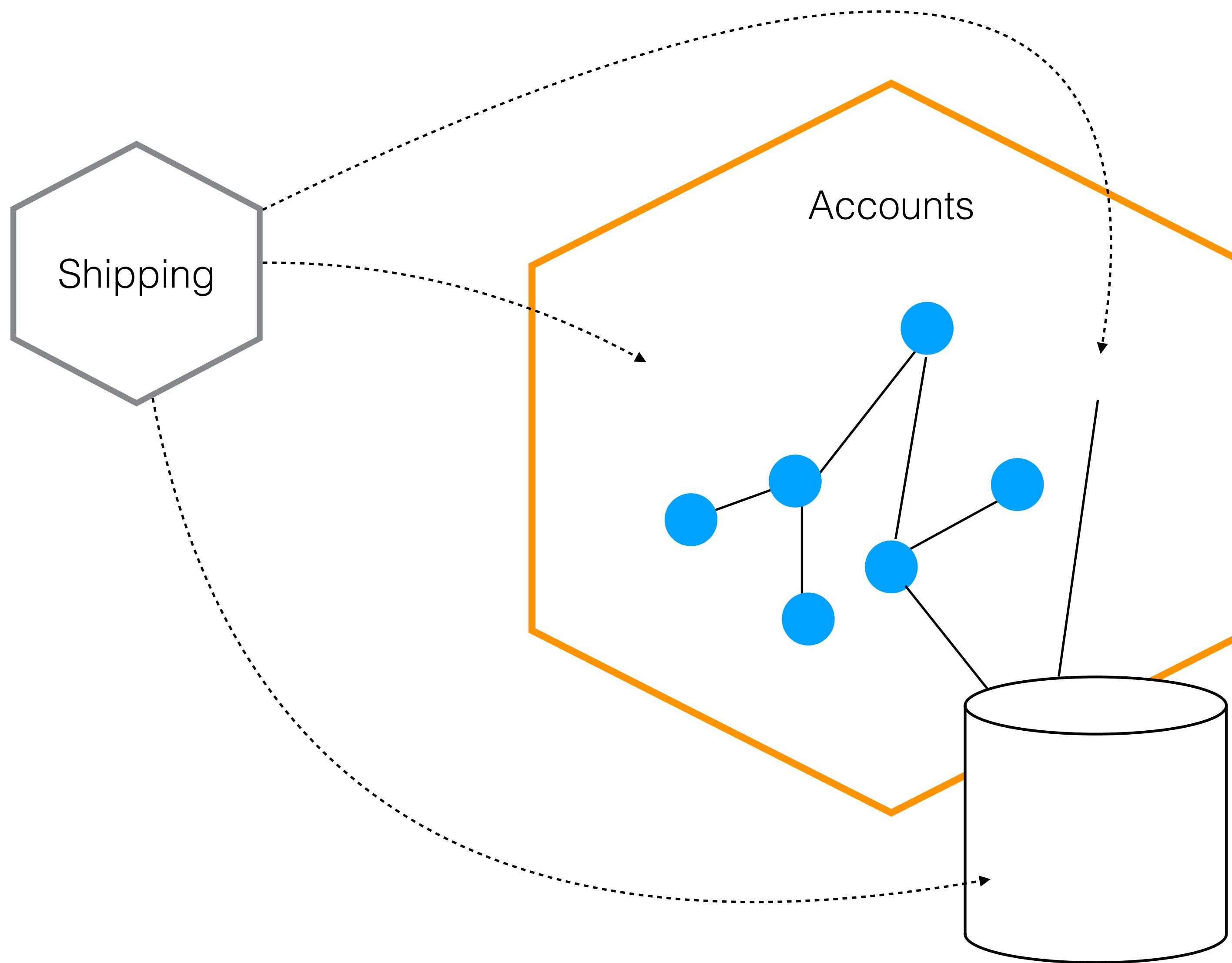
NOT HIDING?



If an upstream consumer can reach into your internal implementation..

...then you can't change this implementation without breaking the consumer

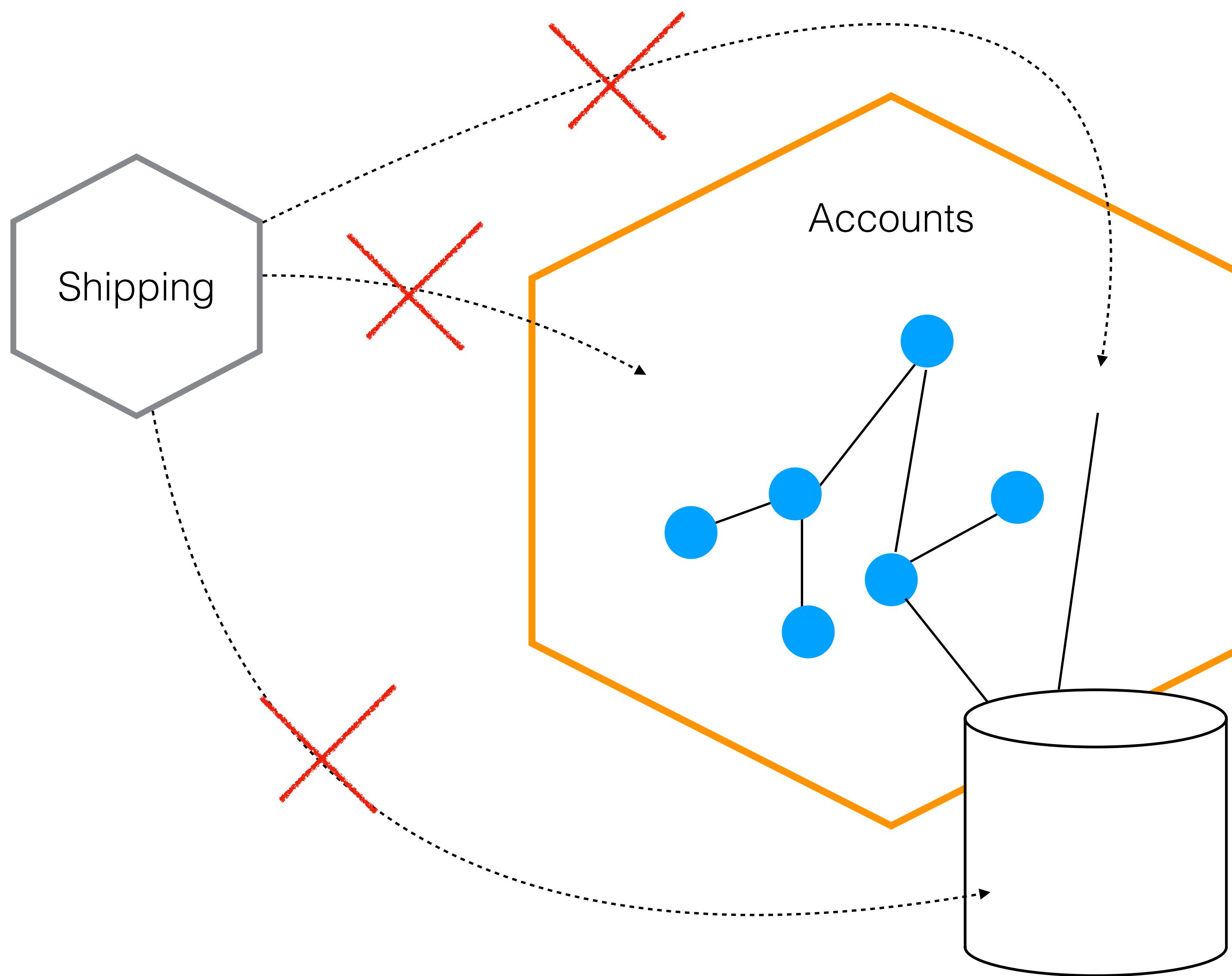
NOT HIDING?



If an upstream consumer can reach into your internal implementation..

...then you can't change this implementation without breaking the consumer

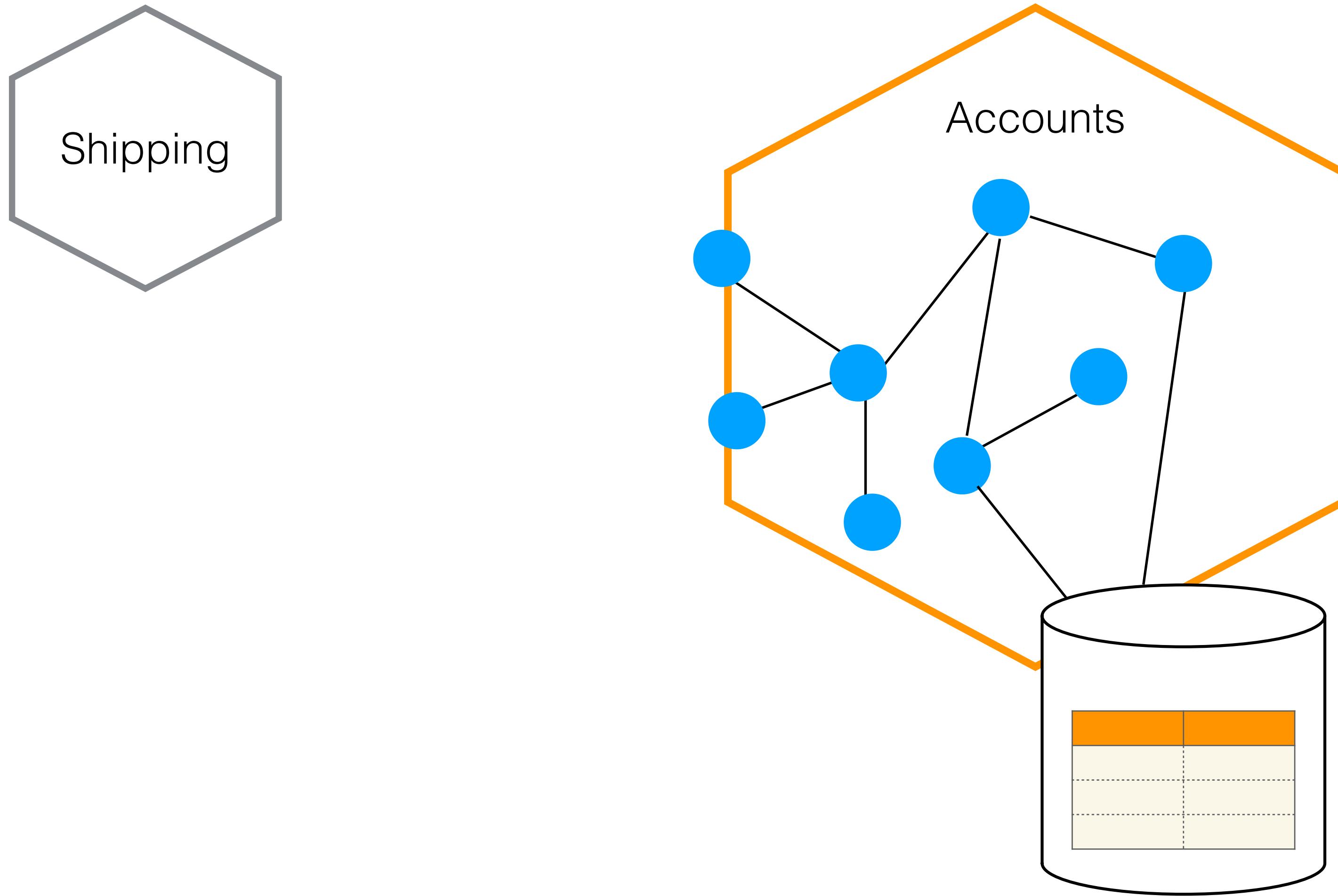
NOT HIDING?



If an upstream consumer can reach into your internal implementation..

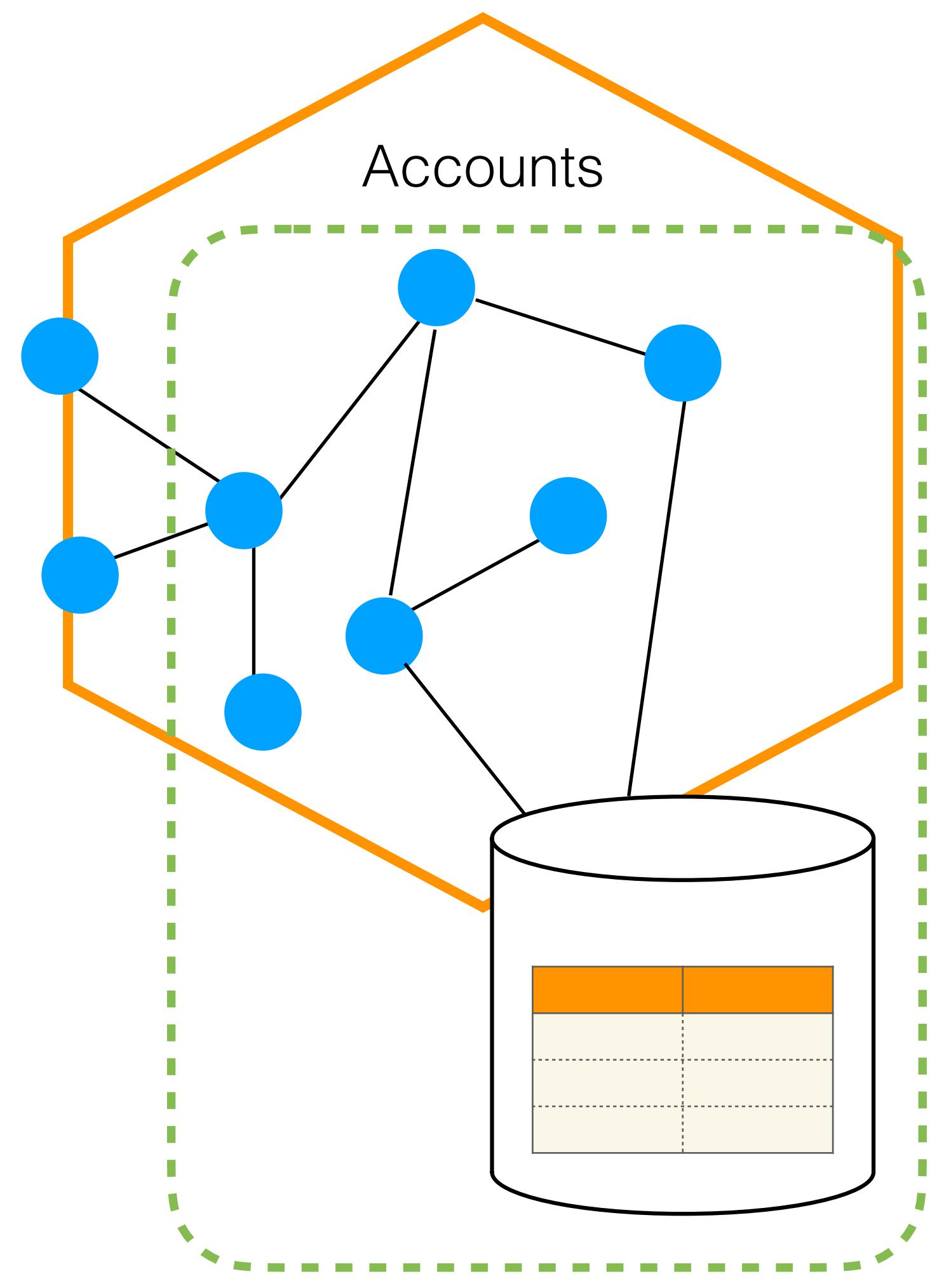
...then you can't change this implementation without breaking the consumer

HIDING!



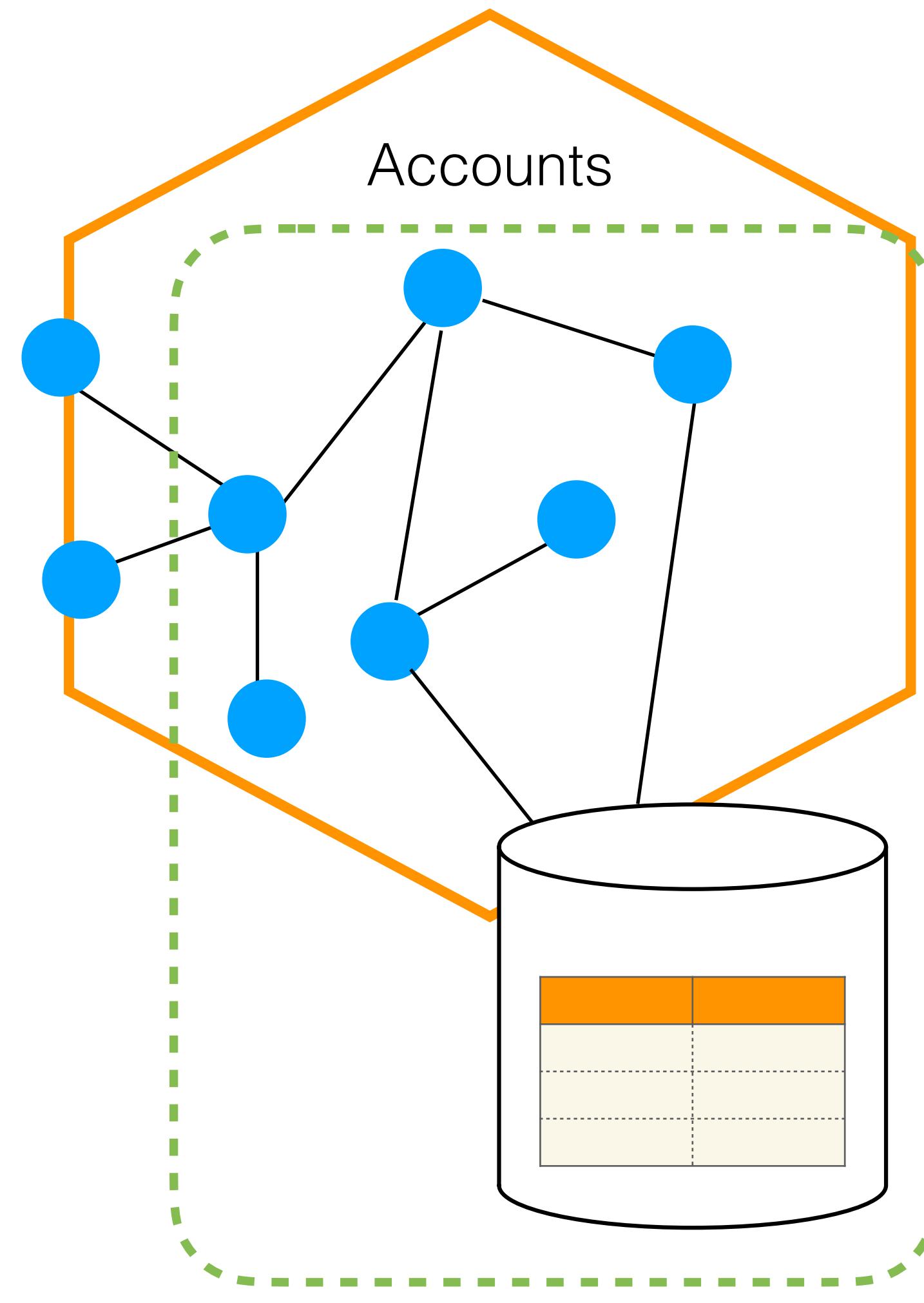
HIDING!

Hide your secrets!



Hidden

HIDING!

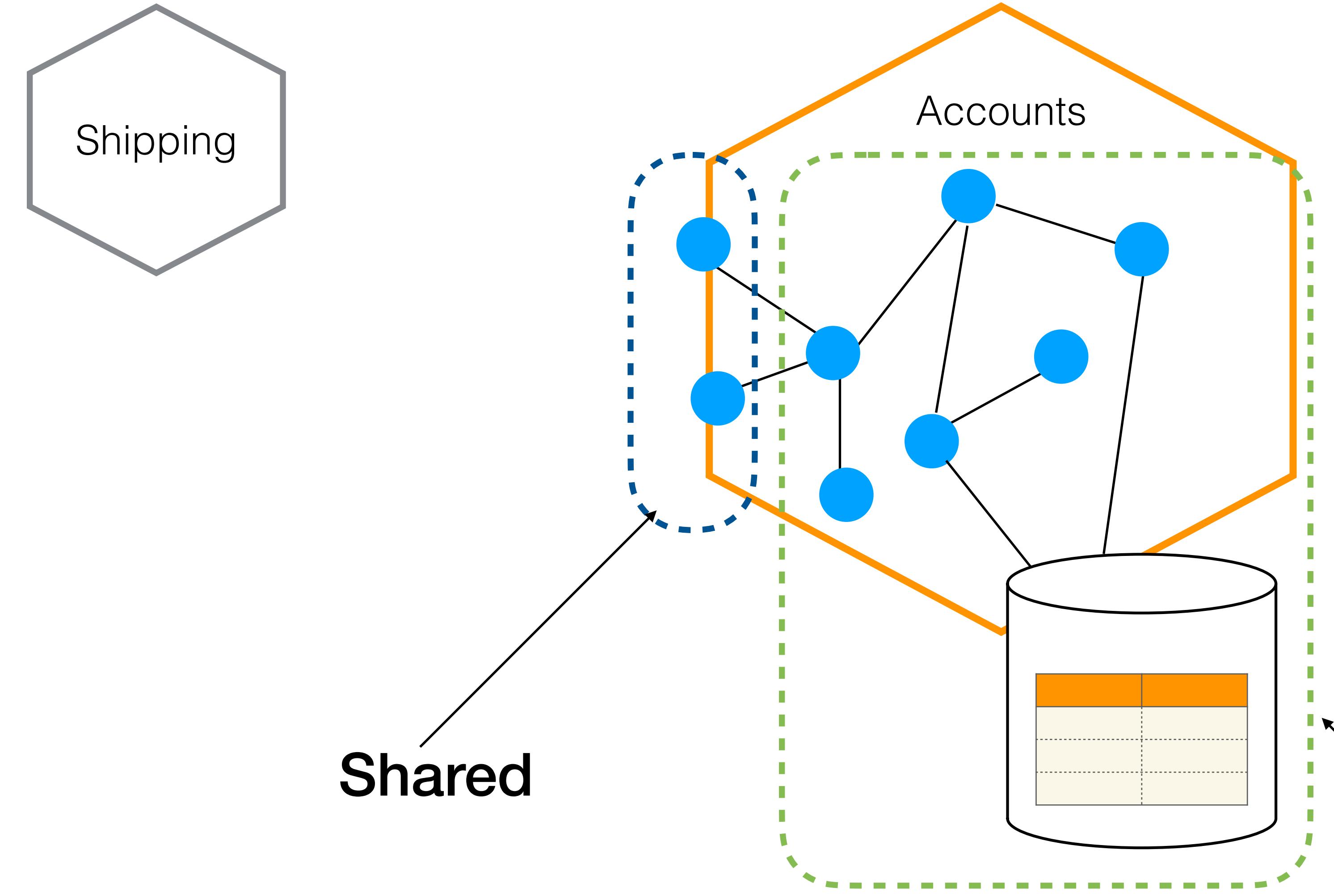


Hide your secrets!

Be explicit about what is shared, and what is hidden

Hidden

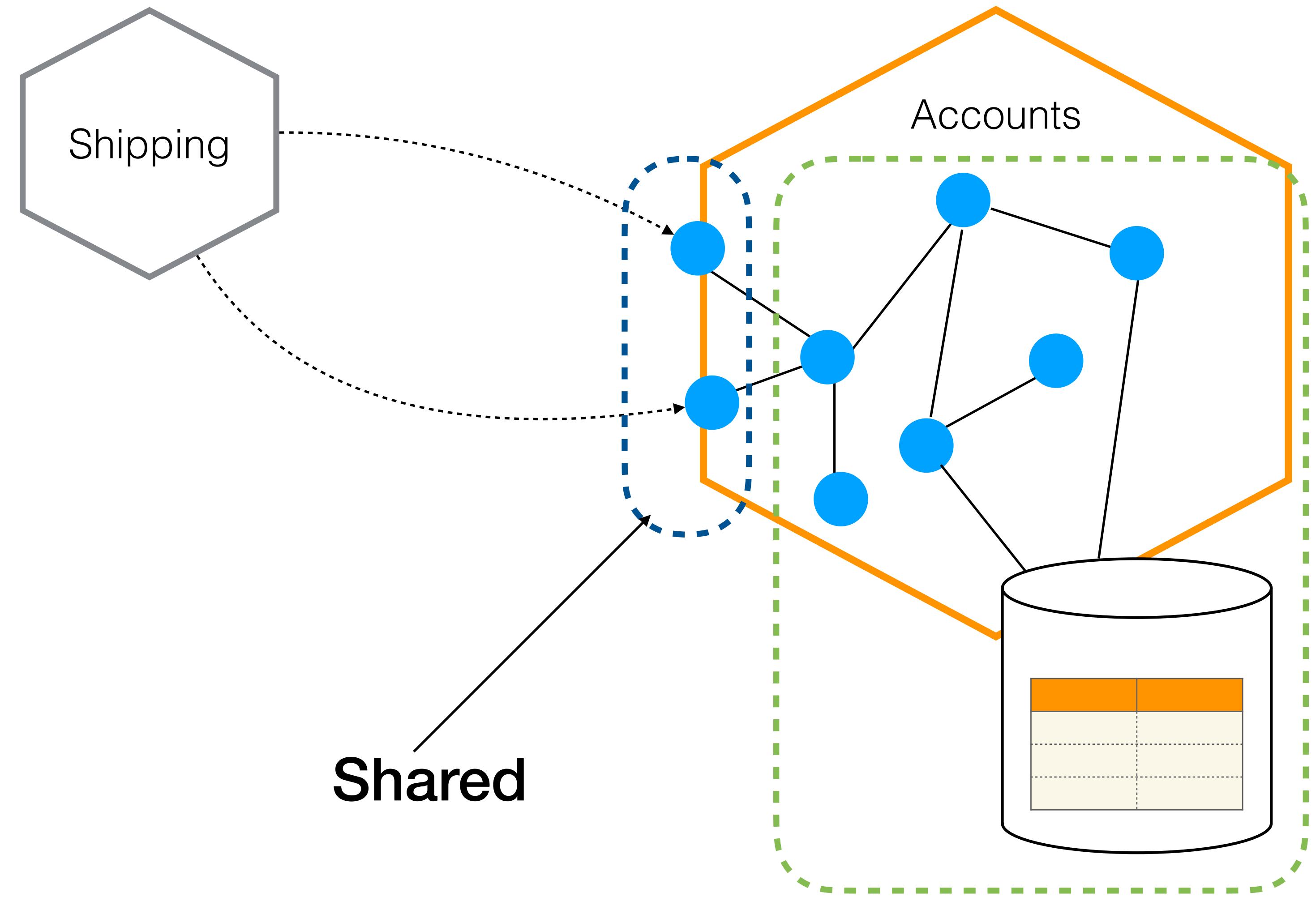
HIDING!



Hide your secrets!

Be explicit about what is shared, and what is hidden

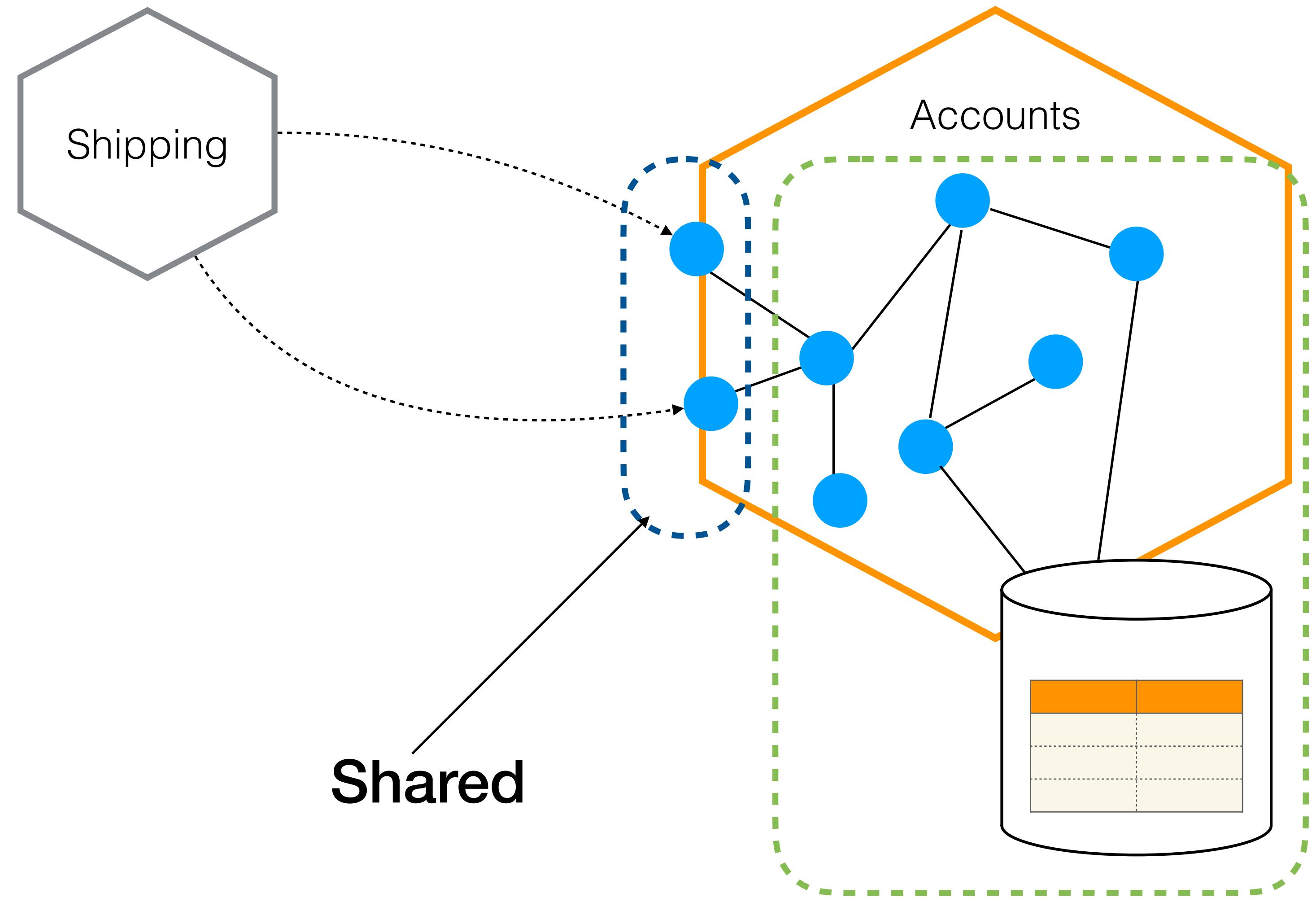
HIDING!



Hide your secrets!

Be explicit about what is shared, and what is hidden

HIDING!



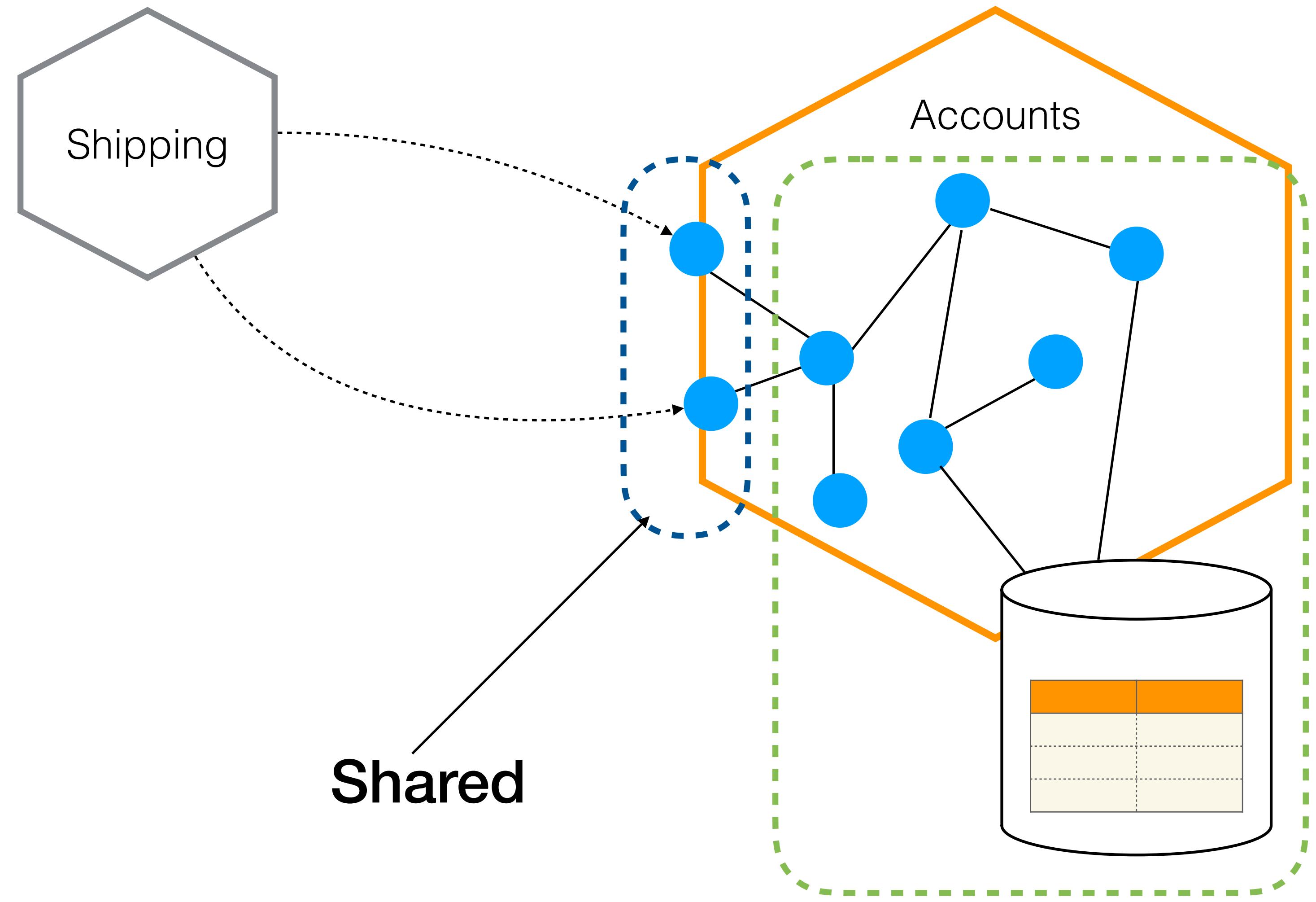
Hide your secrets!

Be explicit about what is shared, and what is hidden

Hidden things can change, shared things can't

Hidden

HIDING!



Hide your secrets!

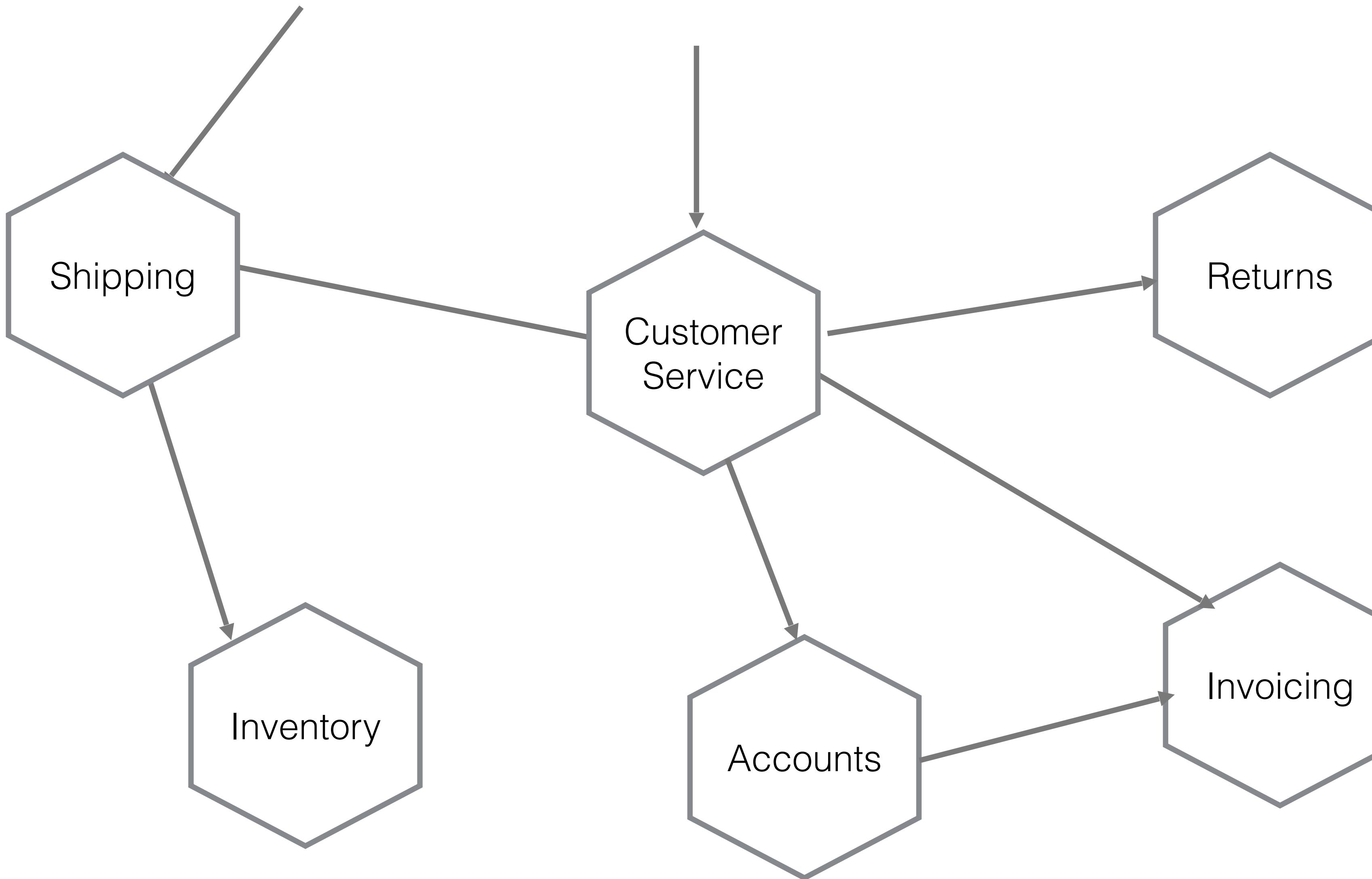
Be explicit about what is shared, and what is hidden

Hidden things can change, shared things can't

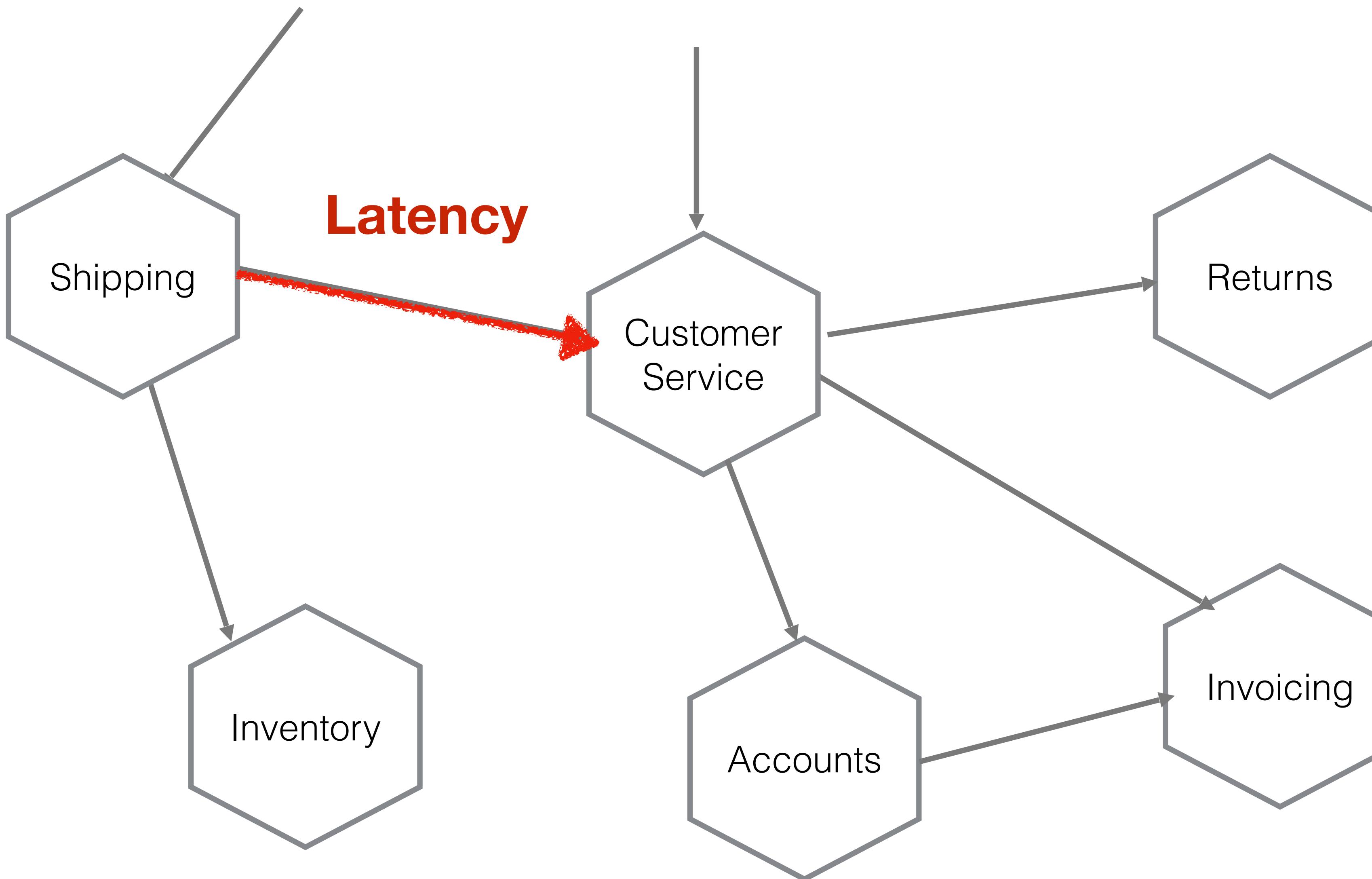
**Hiding the data storage of a microservice is
about hiding implementation detail**

**Information hiding makes independent
deployability possible**

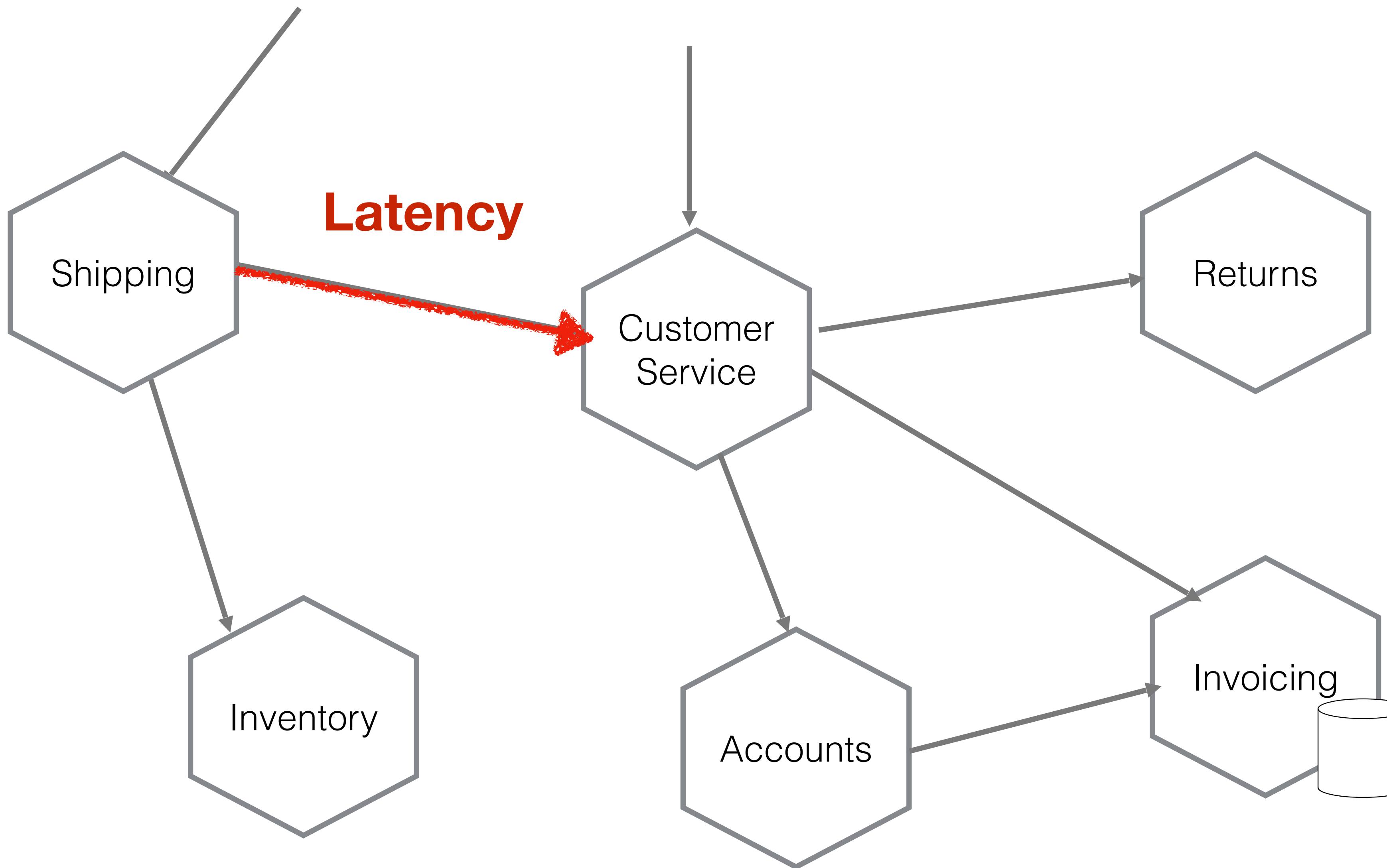
PROBLEMS WITH MICROSERVICES



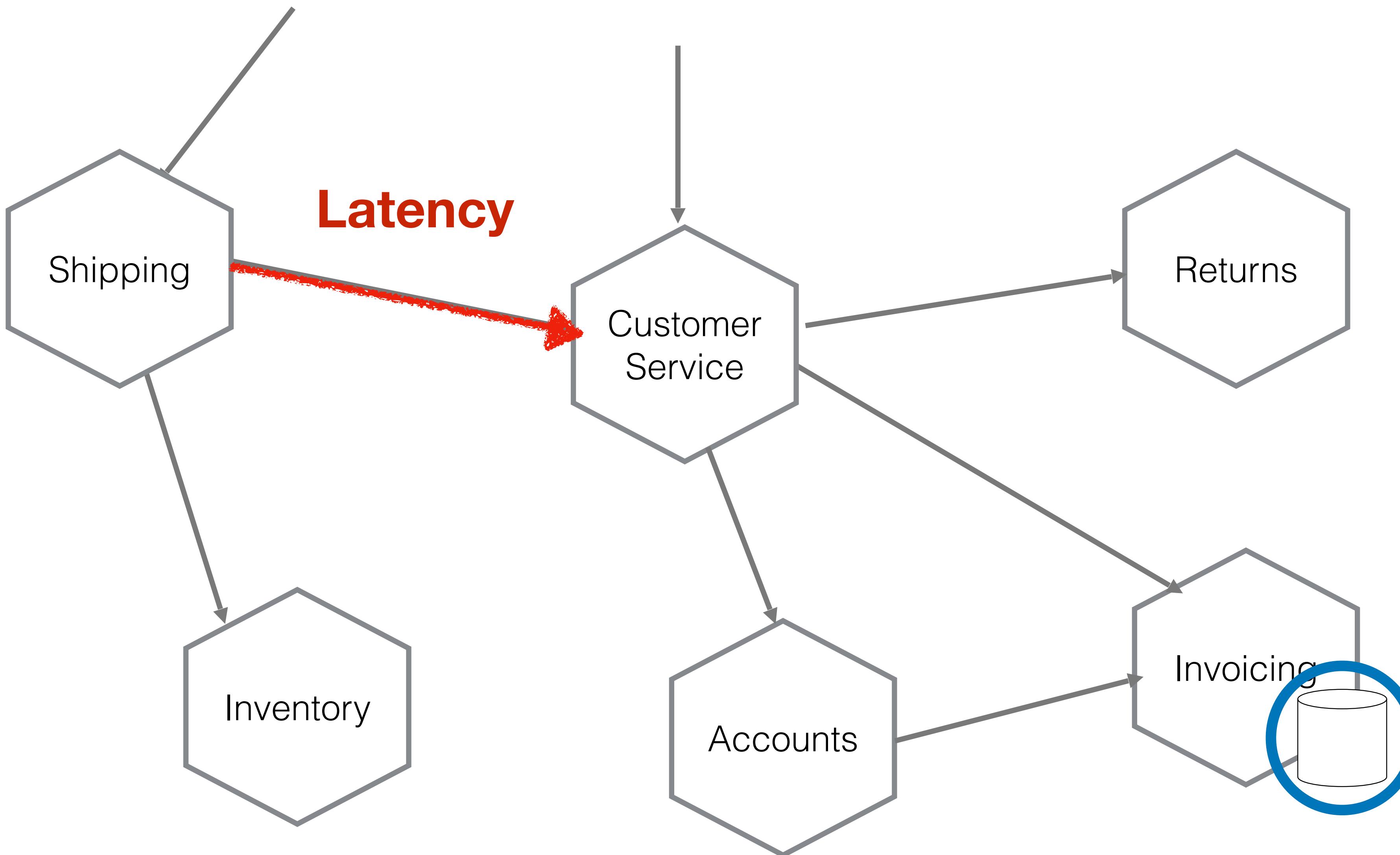
PROBLEMS WITH MICROSERVICES



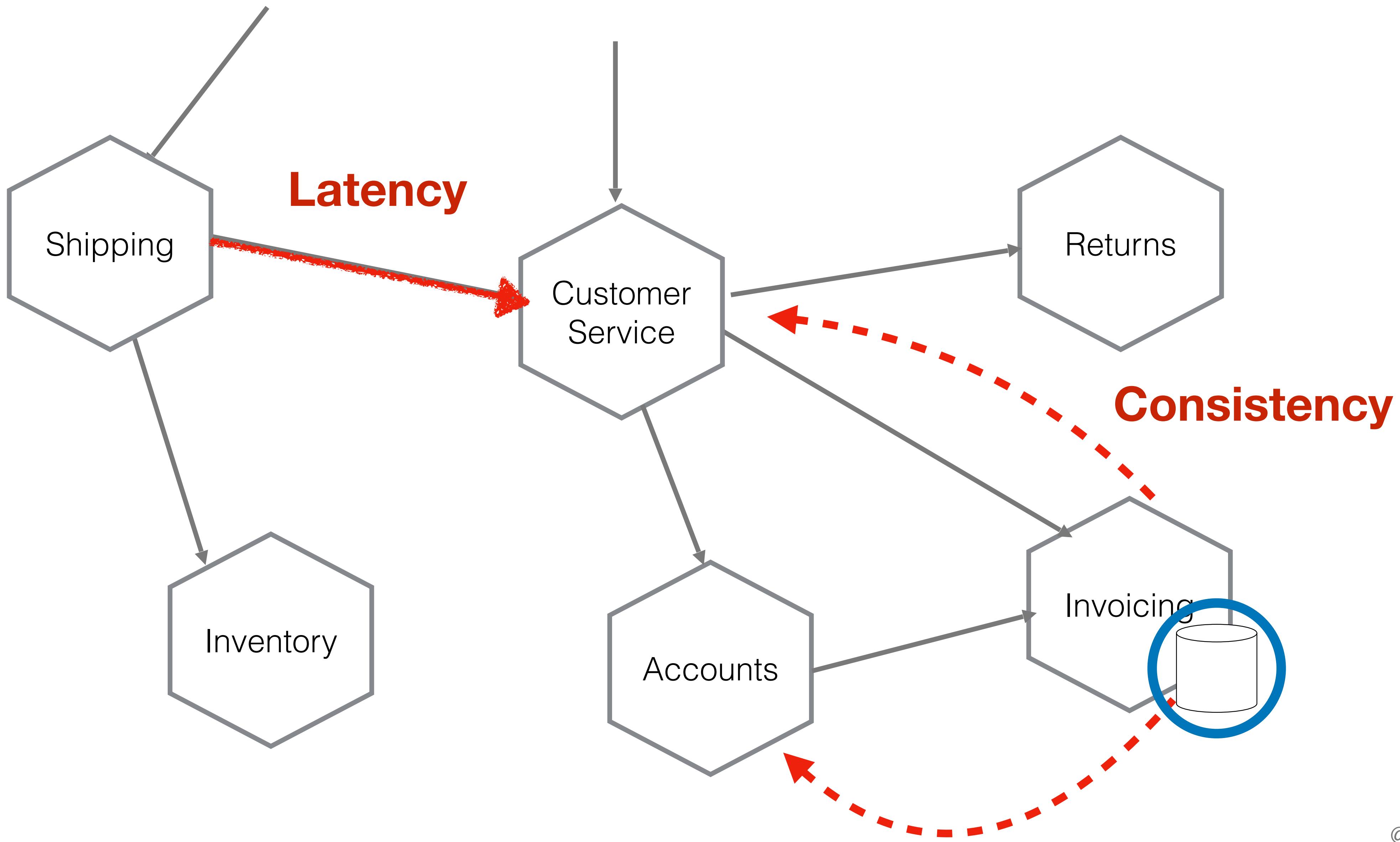
PROBLEMS WITH MICROSERVICES



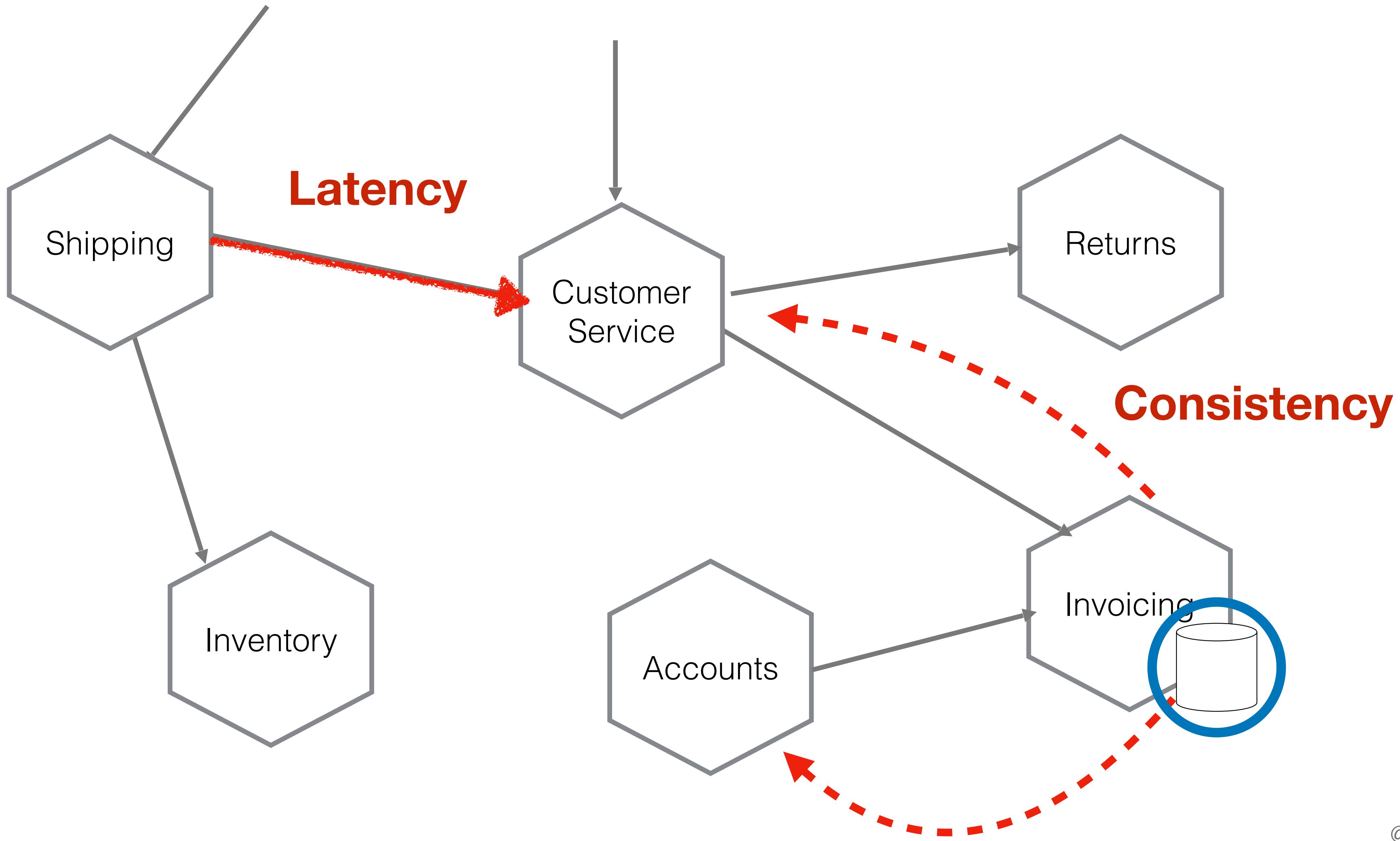
PROBLEMS WITH MICROSERVICES



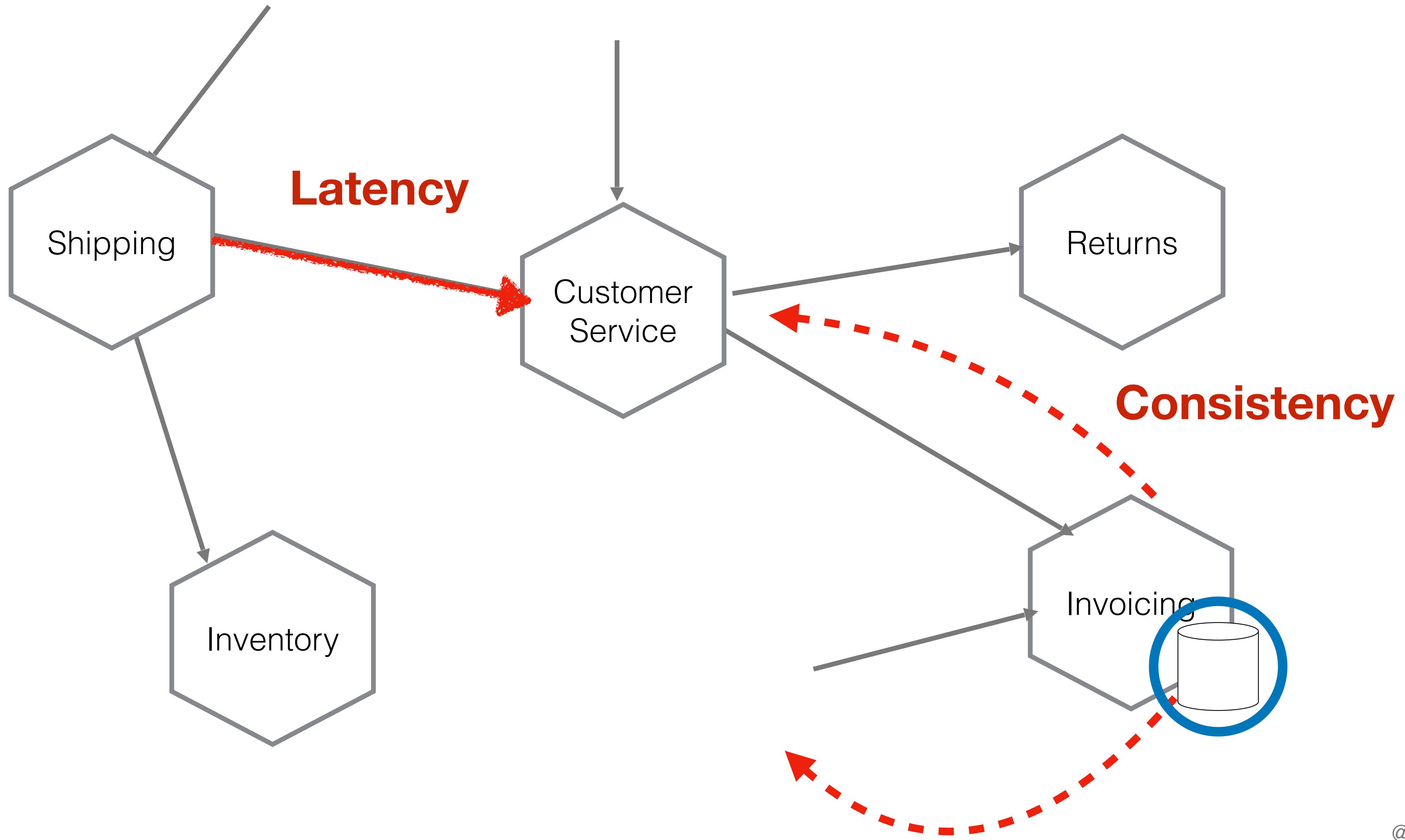
PROBLEMS WITH MICROSERVICES



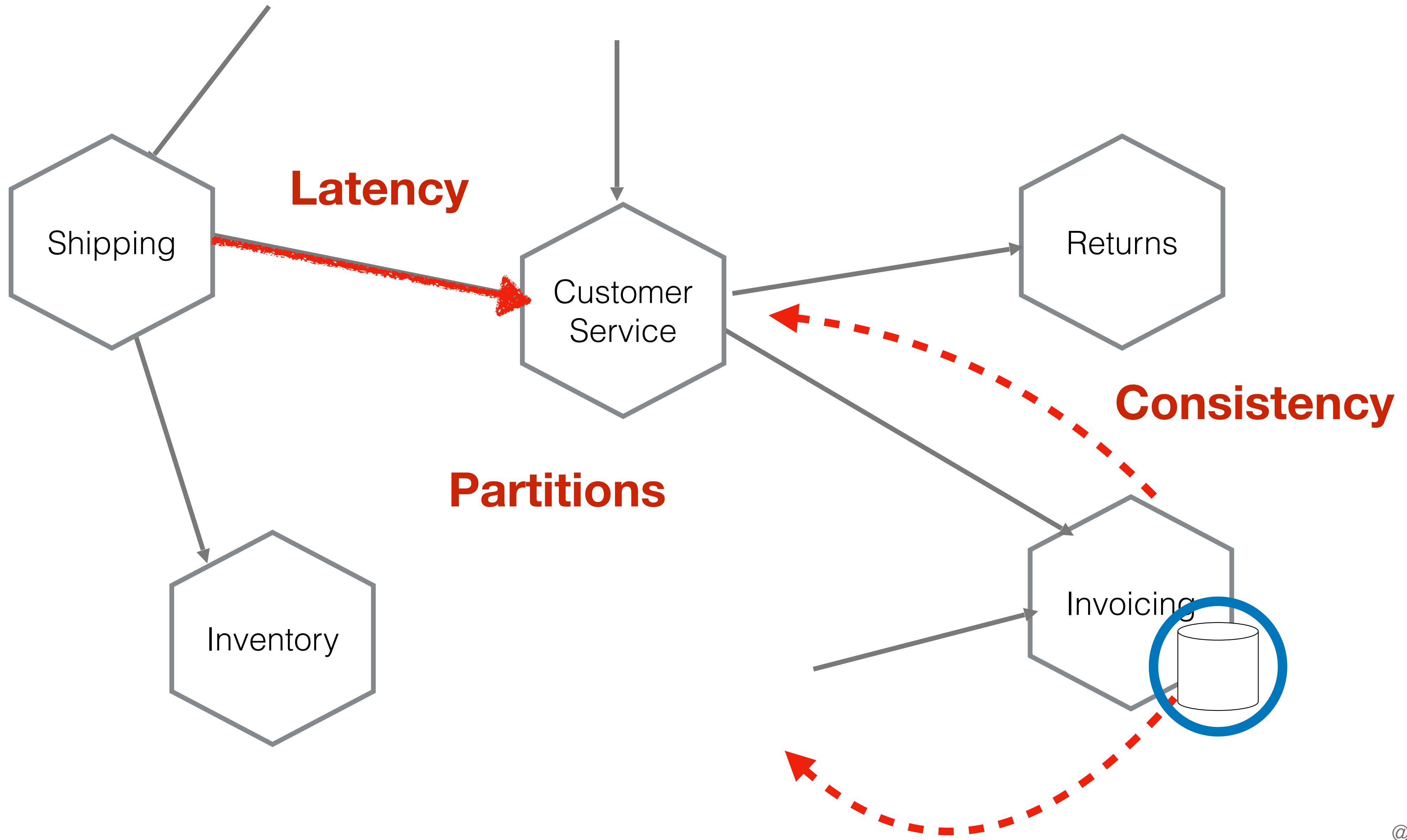
PROBLEMS WITH MICROSERVICES



PROBLEMS WITH MICROSERVICES



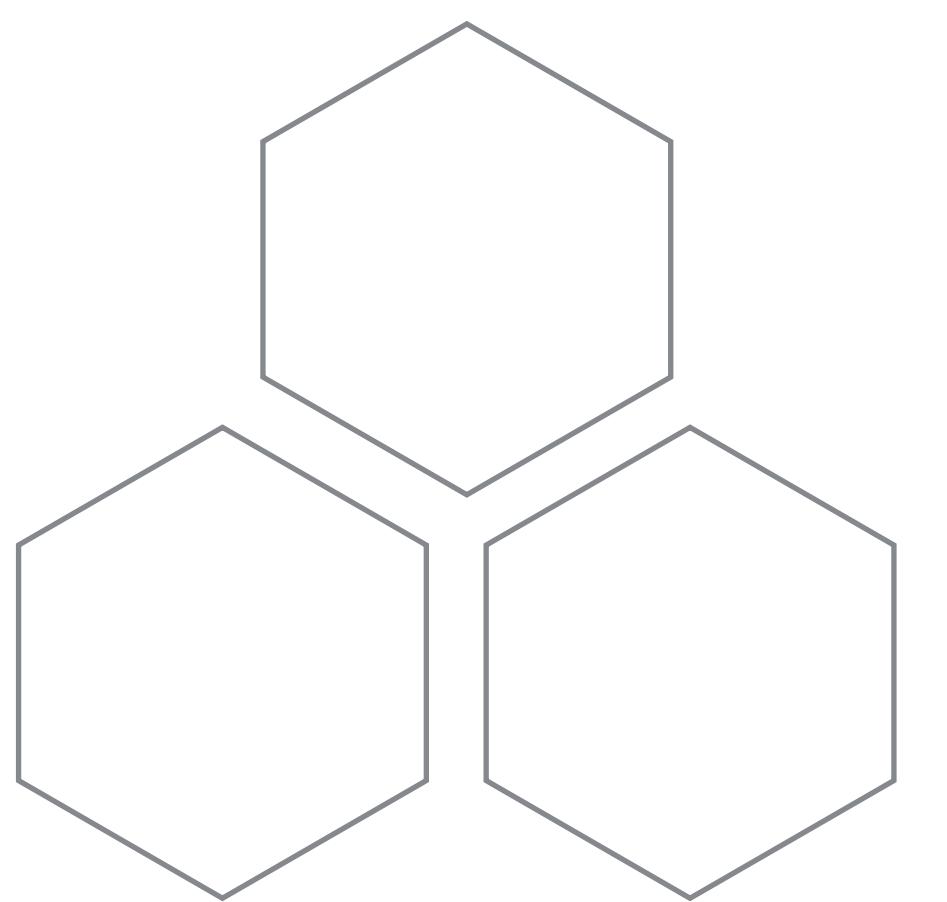
PROBLEMS WITH MICROSERVICES

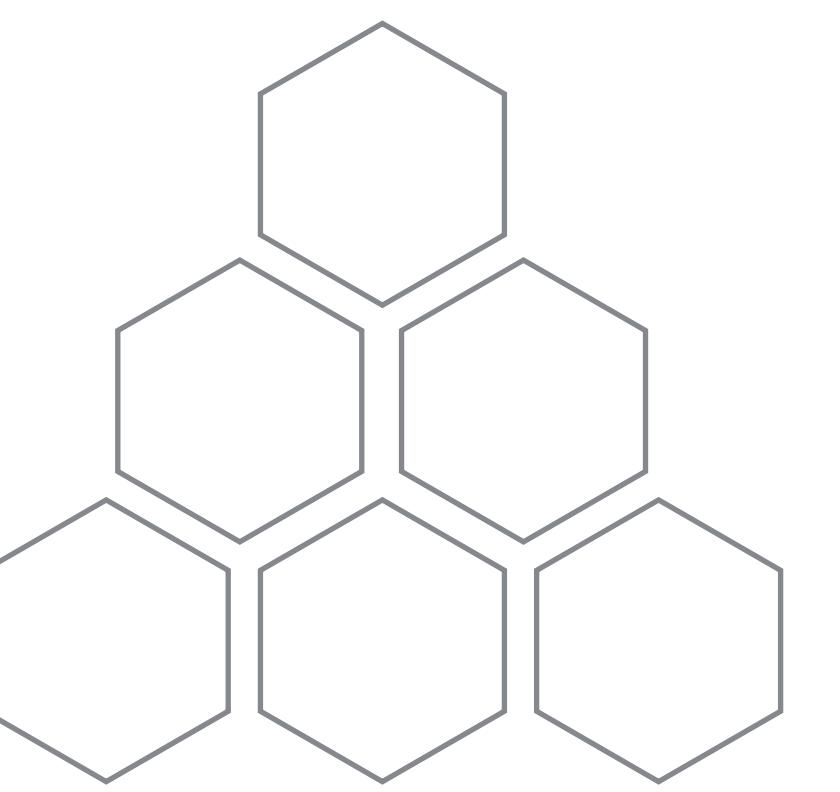
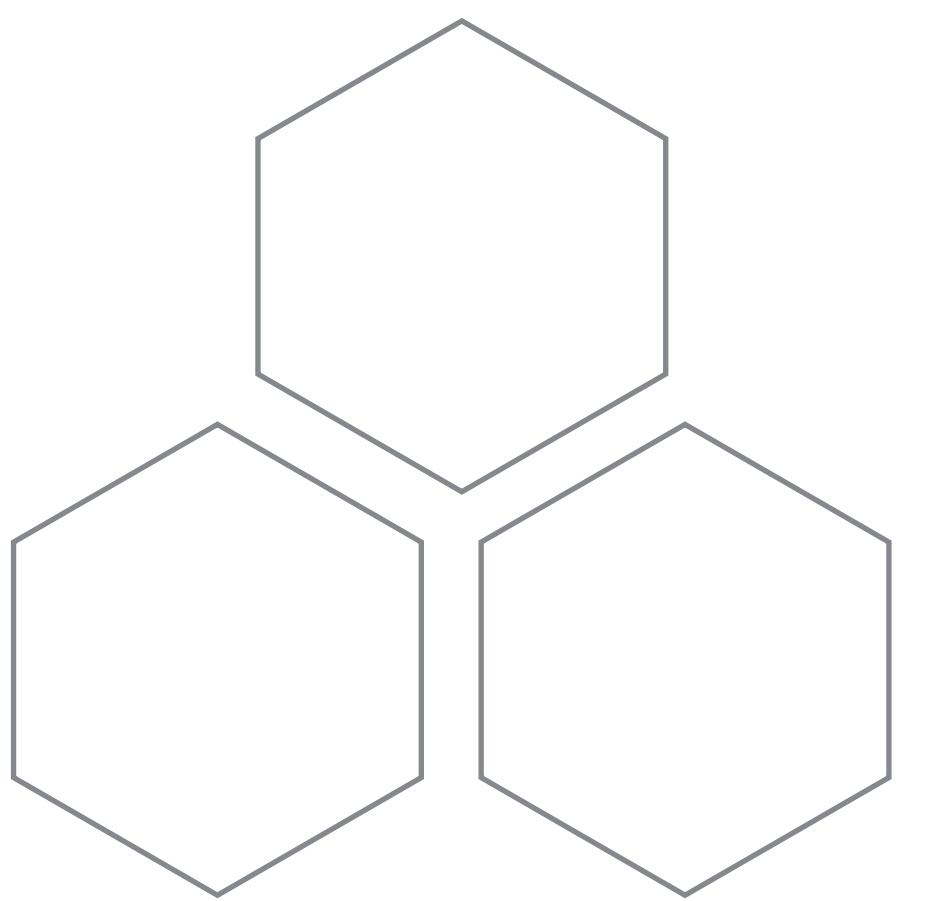


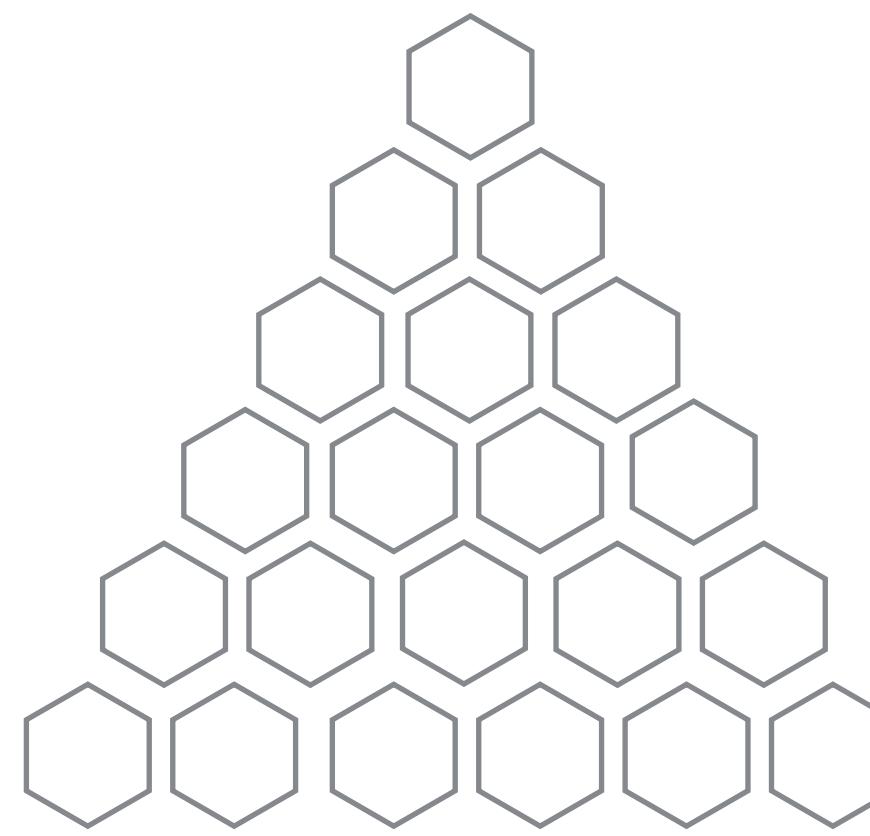
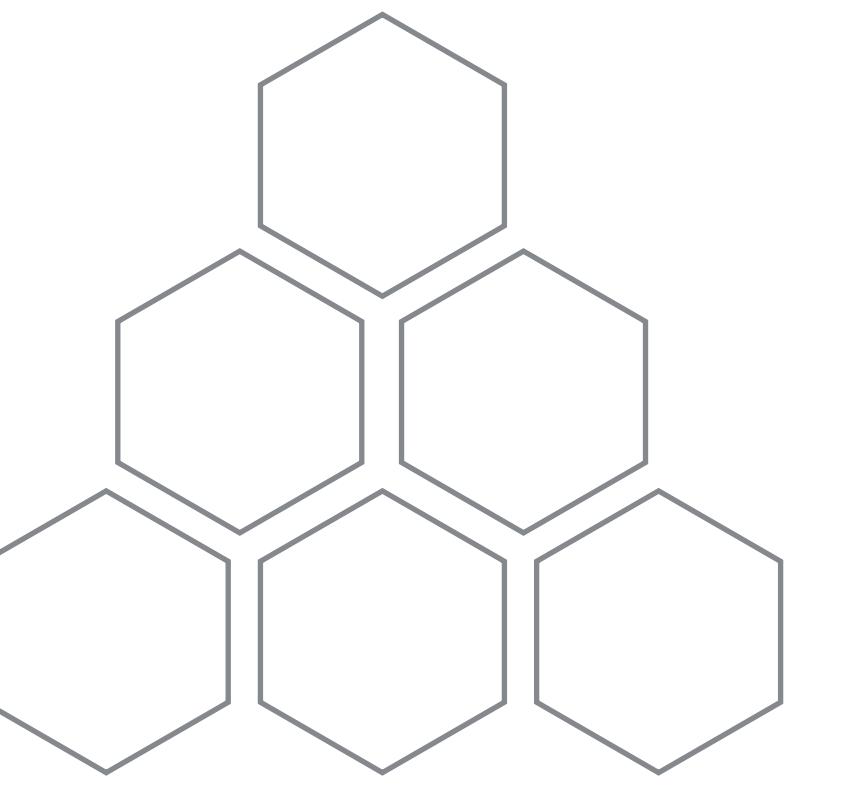
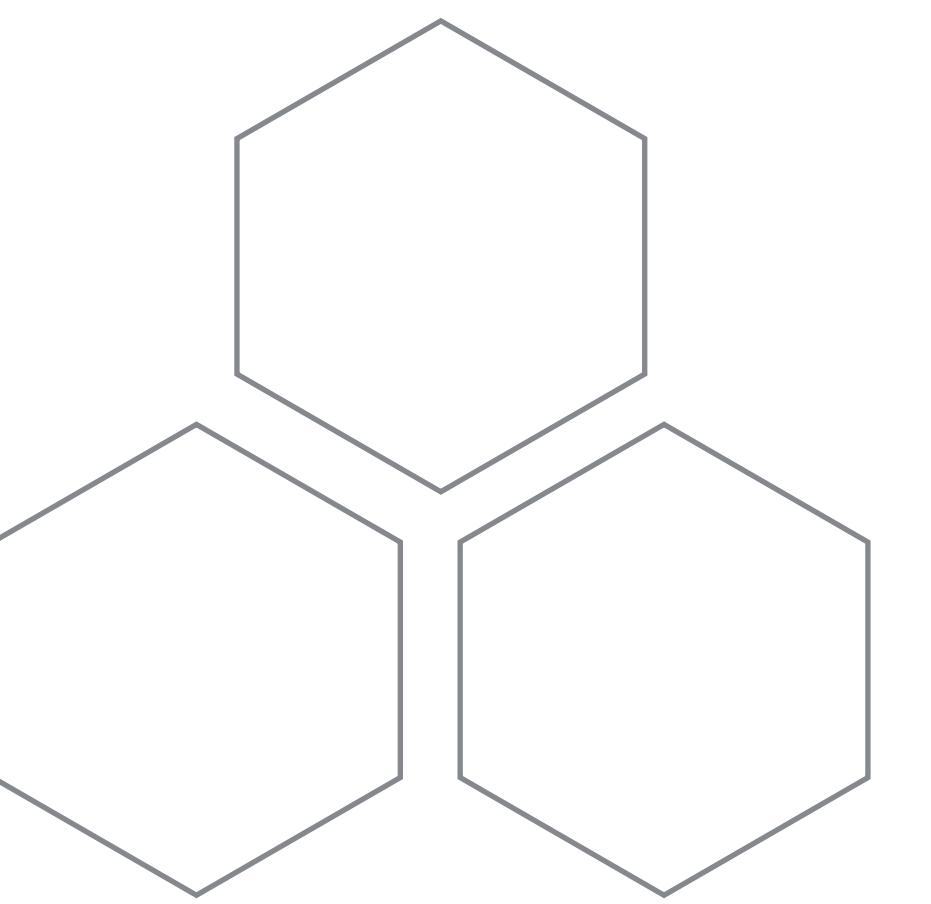
LOUDNESS

10









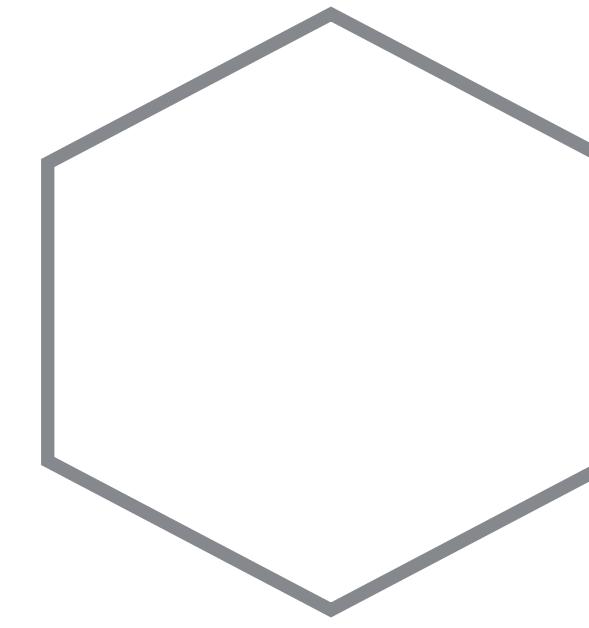
You won't appreciate the true horror,
pain and suffering of microservices
until you're running them in
production

INCREMENTAL DECOMPOSITION FTW

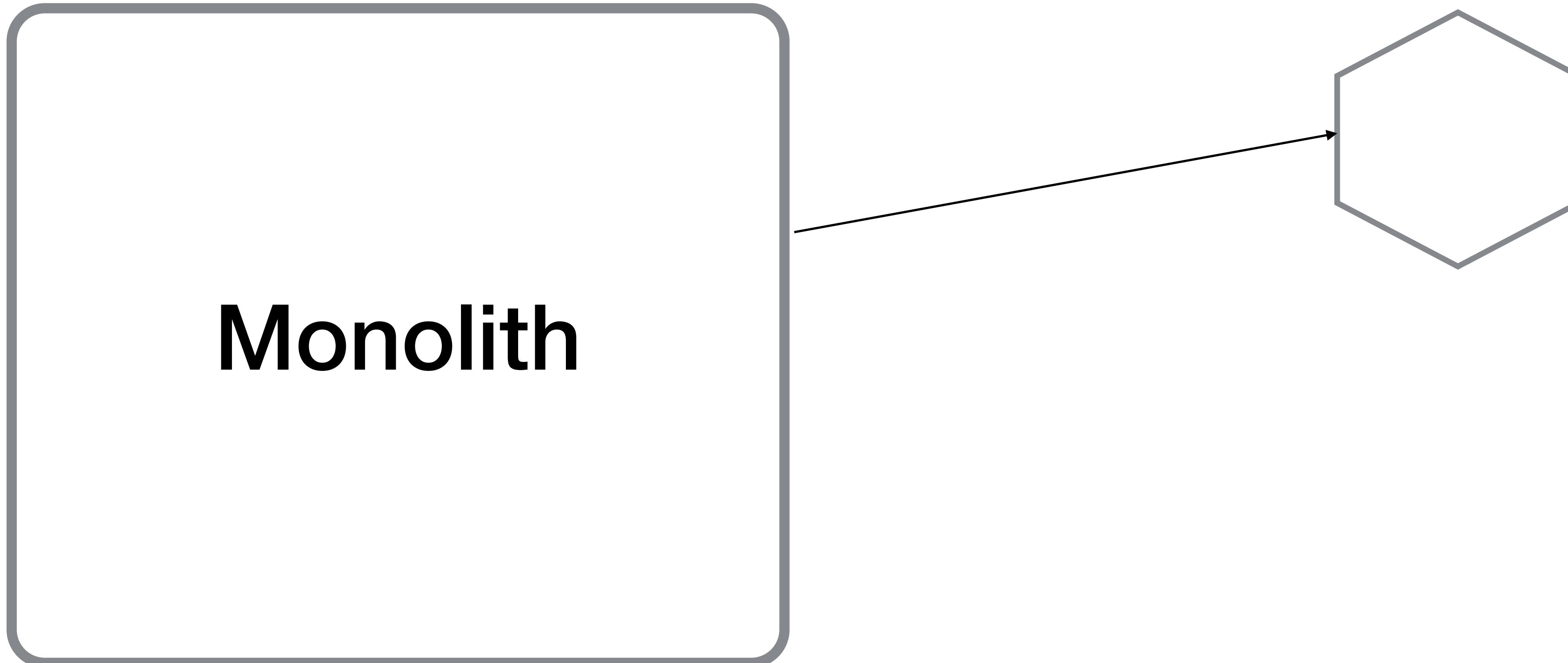


Monolith

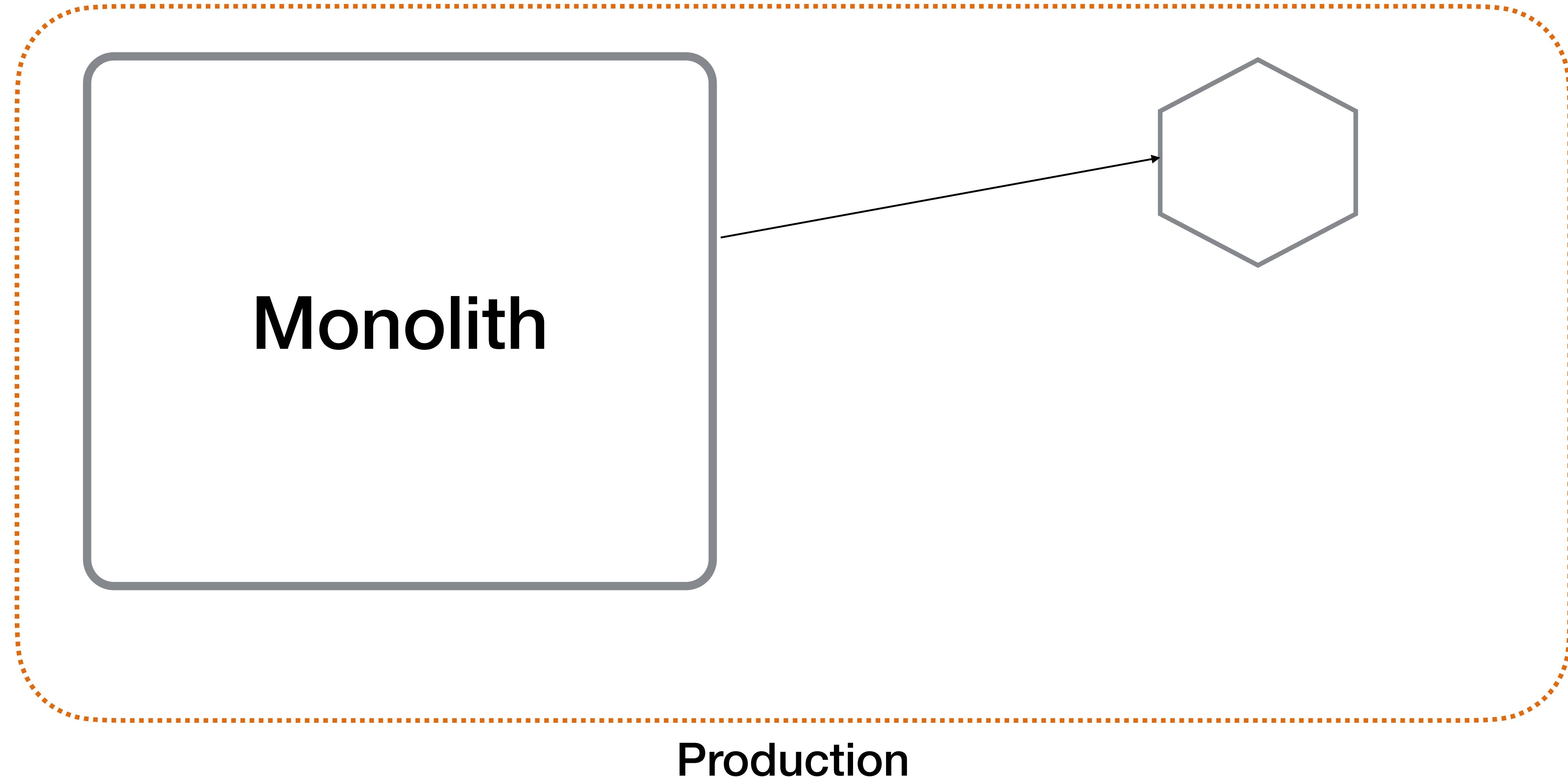
INCREMENTAL DECOMPOSITION FTW



INCREMENTAL DECOMPOSITION FTW



INCREMENTAL DECOMPOSITION FTW



“If you do a big bang rewrite, the only thing you’re certain of is a big bang”

- Martin Fowler (paraphrased)

**Move functionality to microservices
a piece at a time**

**Move functionality to microservices
a piece at a time**

**Get it into production to start getting
value from it, and learning from the
experience**

AGENDA

Introduction

Shared Data Patterns

Migration Order

Decomposition Patterns

AGENDA

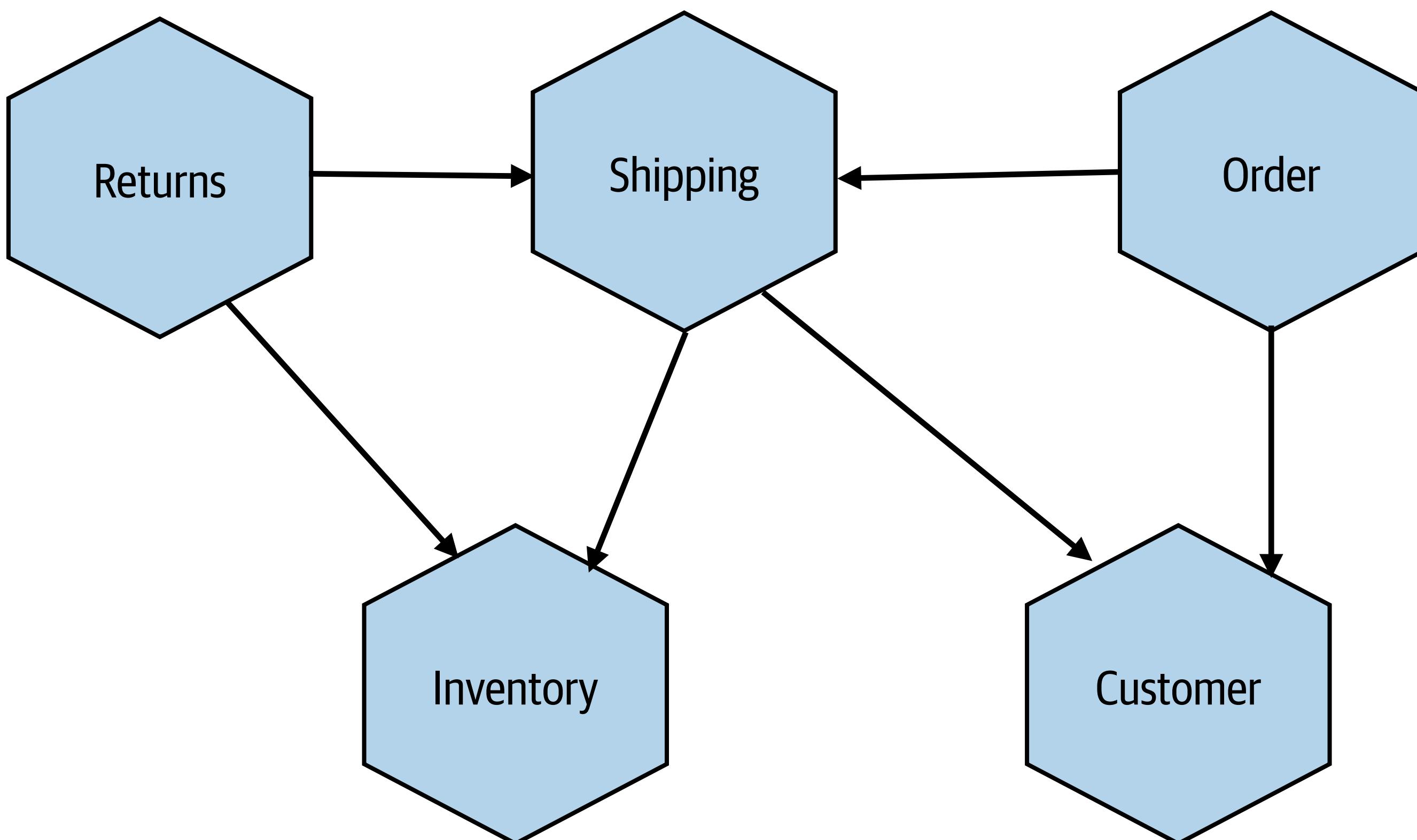
Introduction

Shared Data Patterns

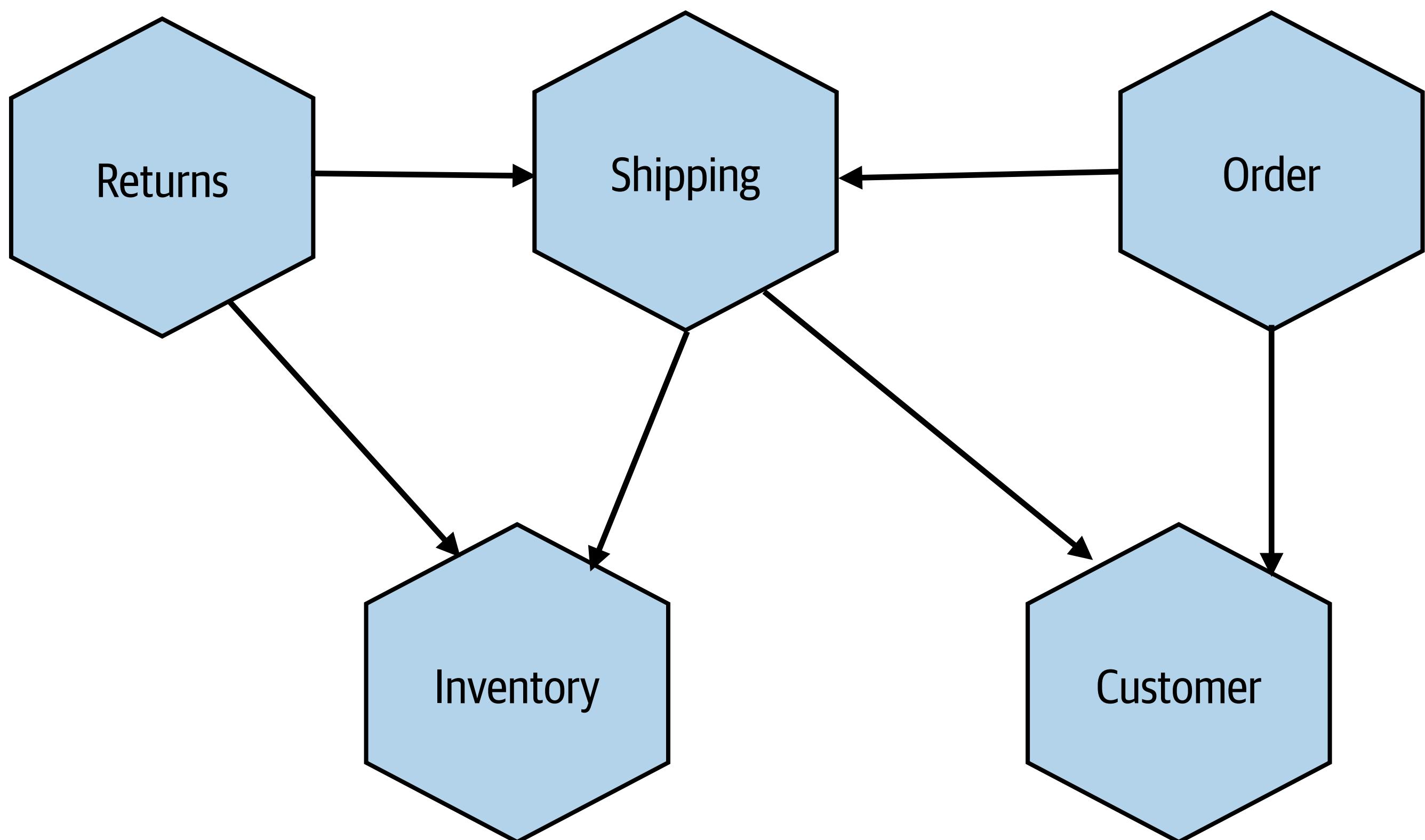
Migration Order

Decomposition Patterns

DATABASES

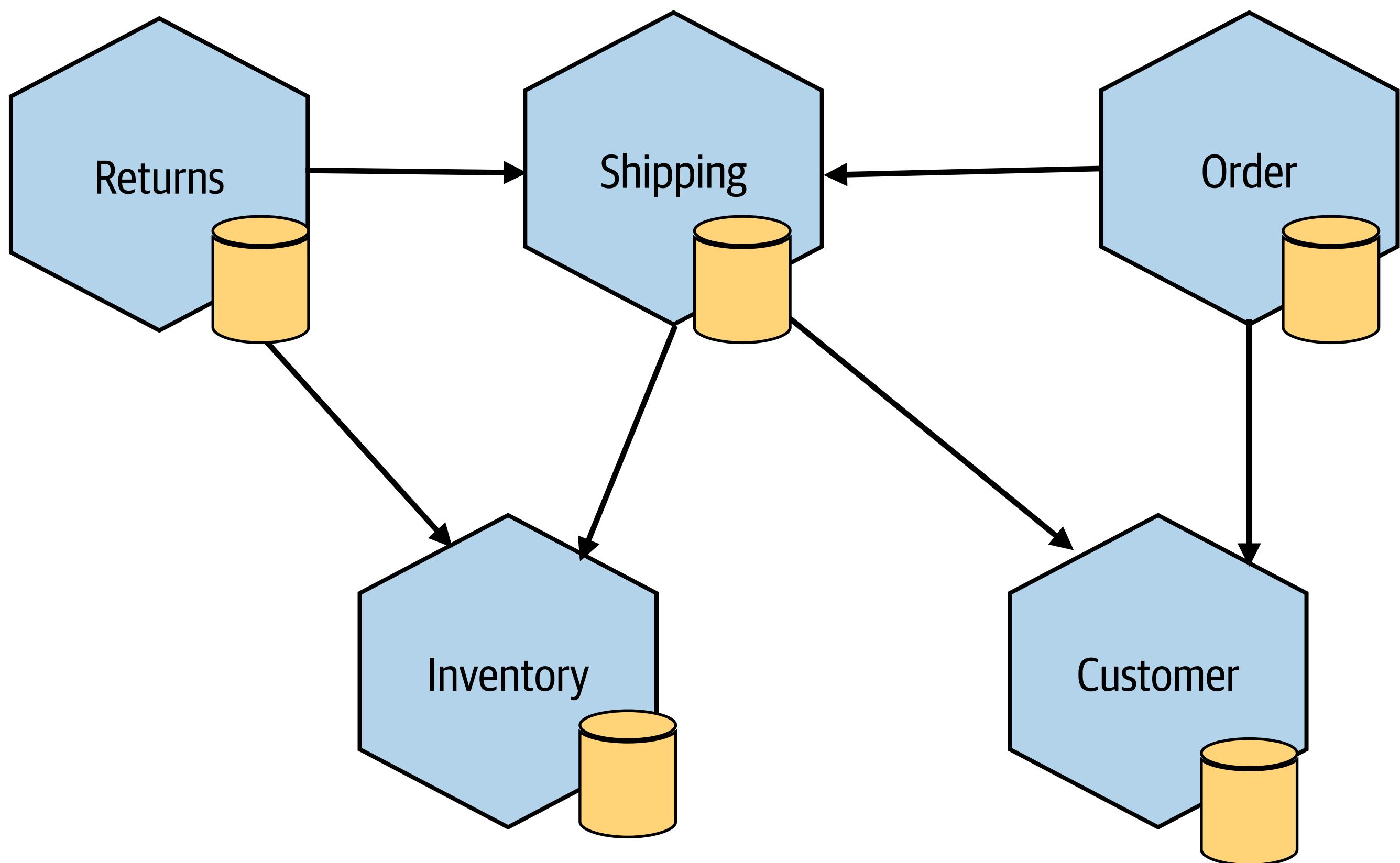


DATABASES



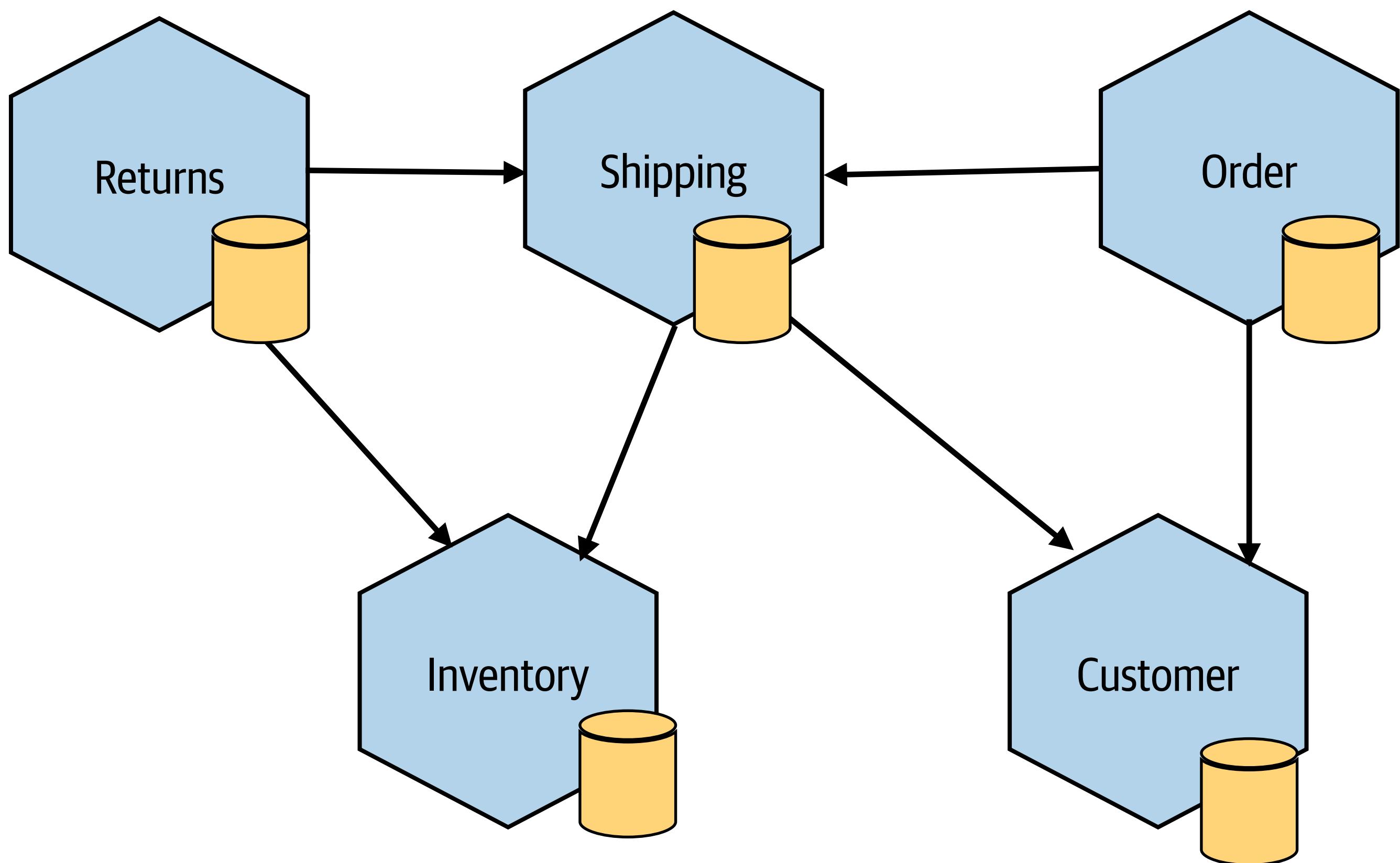
If microservices need to use a database to manage their state, this is hidden from other consumers

DATABASES



If microservices need to use a database to manage their state, this is hidden from other consumers

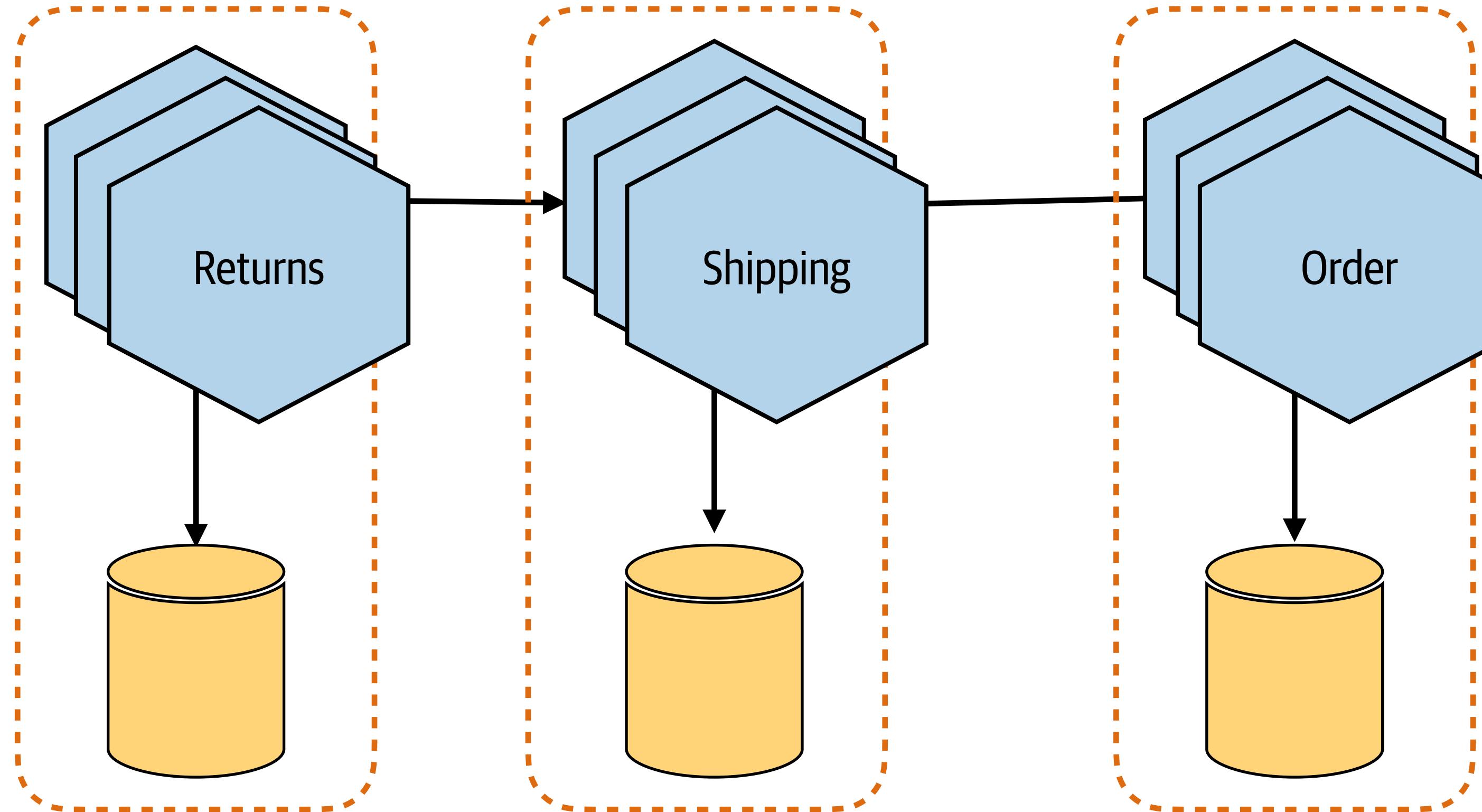
DATABASES



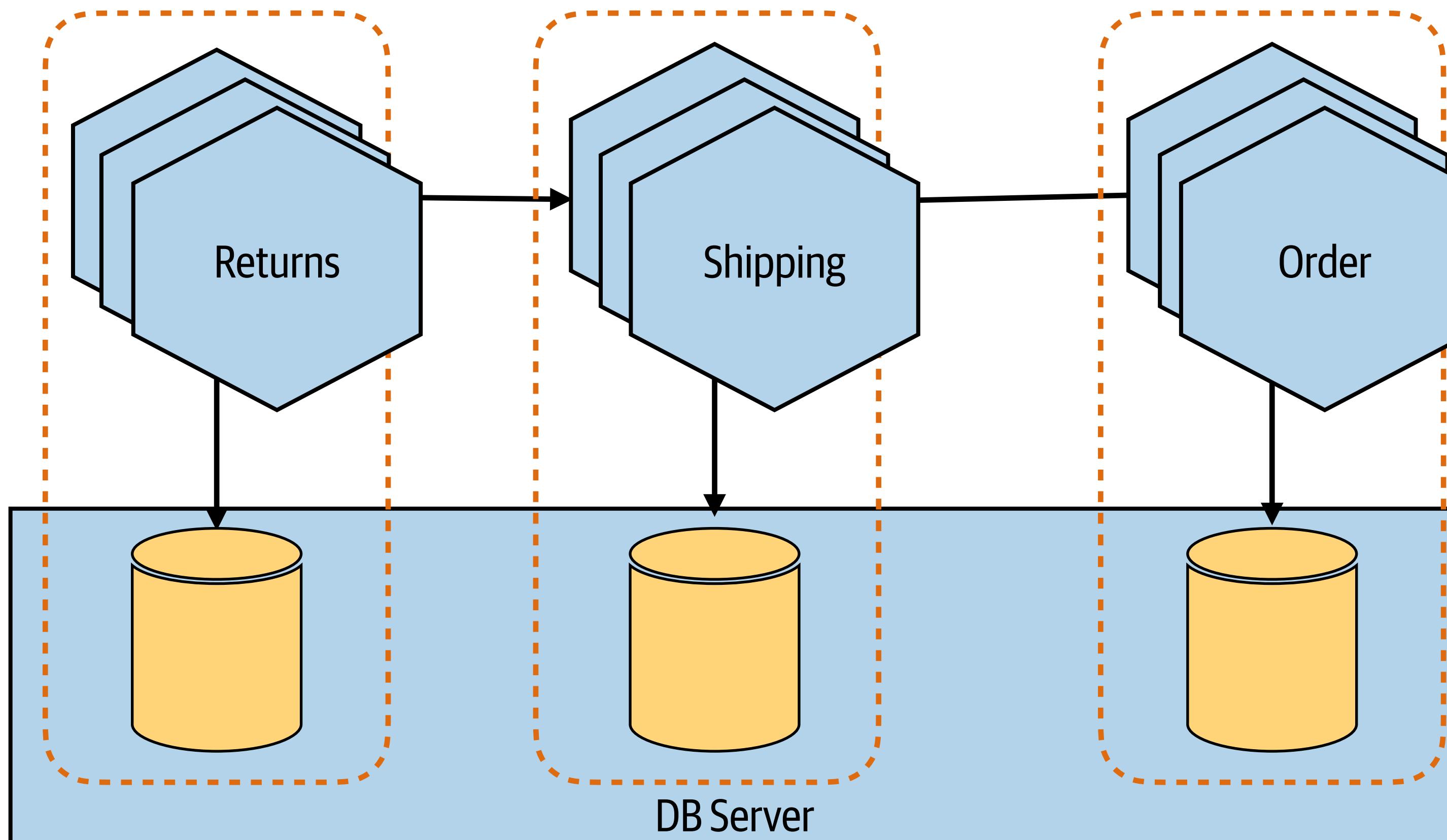
If microservices need to use a database to manage their state, this is hidden from other consumers

Does that mean lots of DB infrastructure to manage?

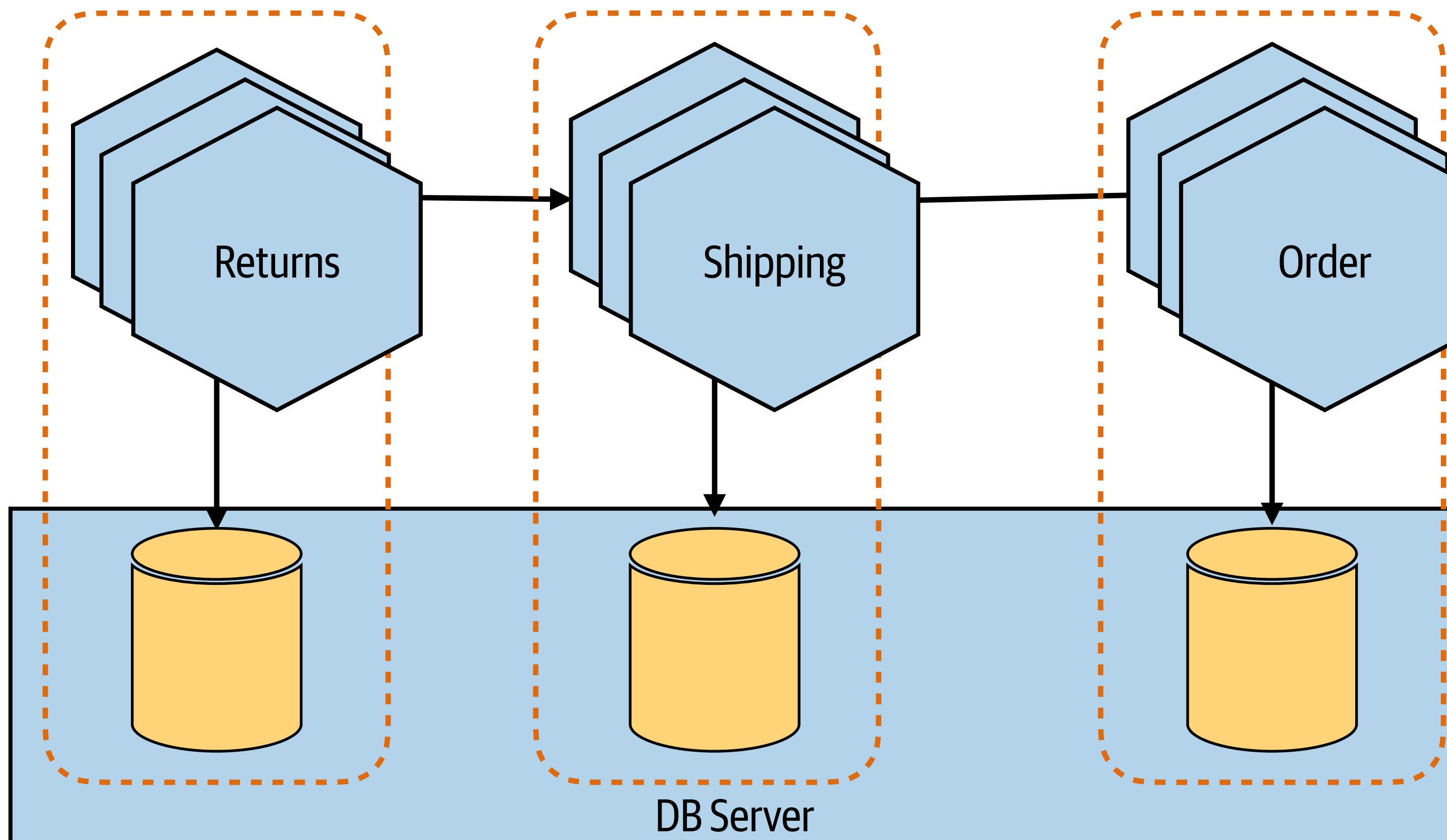
SHARED DB INFRASTRUCTURE



SHARED DB INFRASTRUCTURE

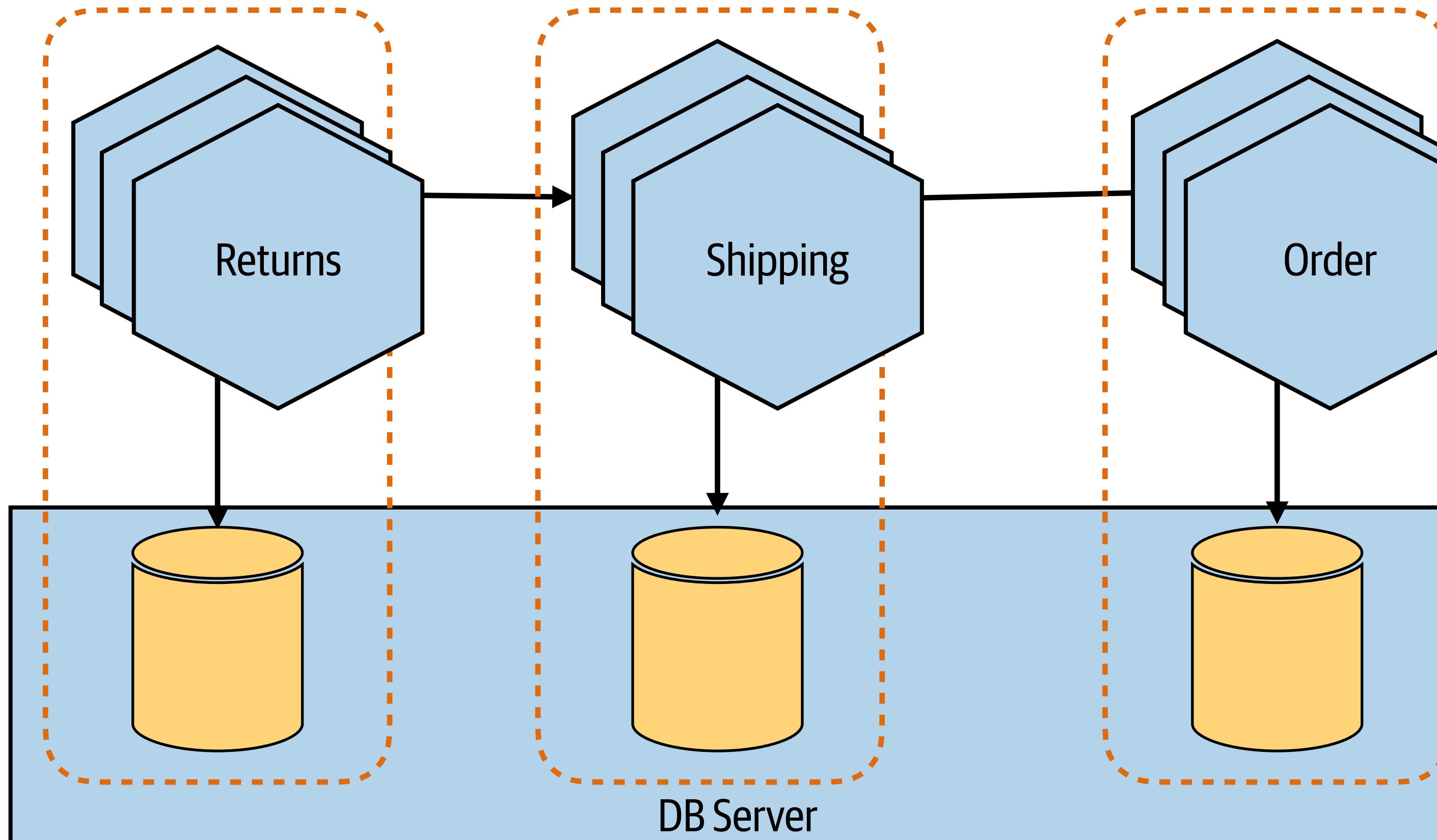


SHARED DB INFRASTRUCTURE



Reduces the amount of DB infra and associated admin

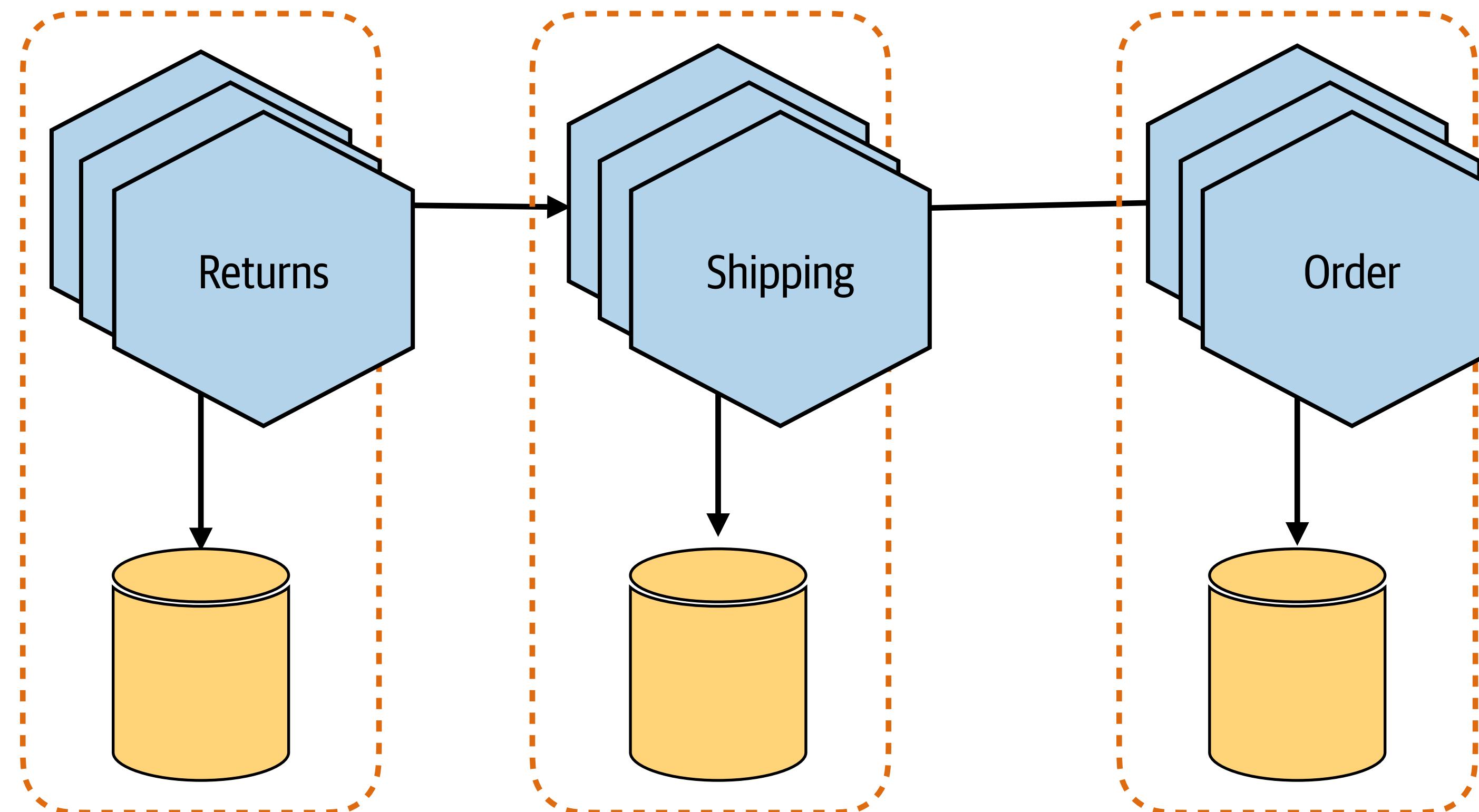
SHARED DB INFRASTRUCTURE



Reduces the amount of DB infra and associated admin

Single source of contention

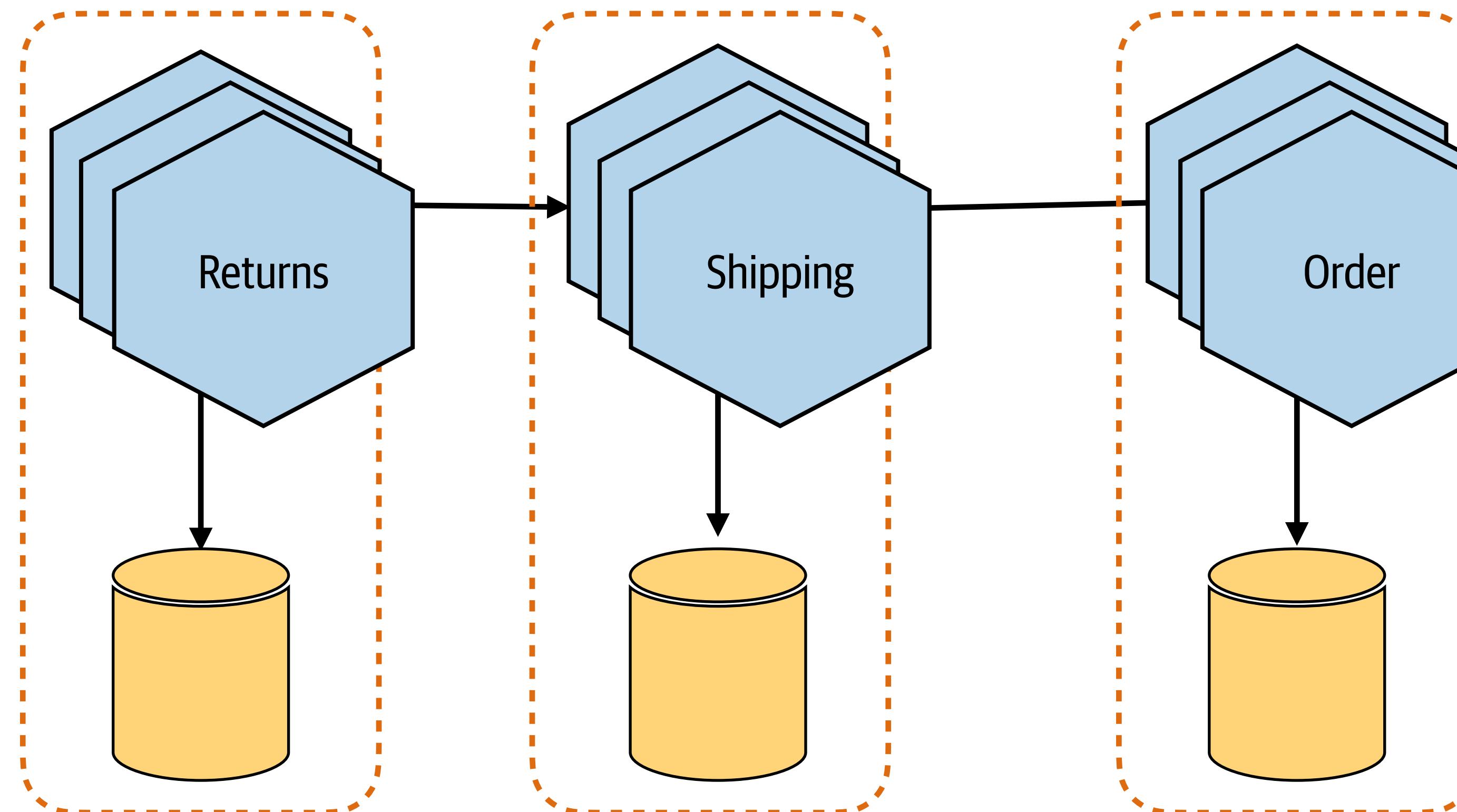
SHARED DB INFRASTRUCTURE



Reduces the amount of DB infra and associated admin

Single source of contention

SHARED DB INFRASTRUCTURE

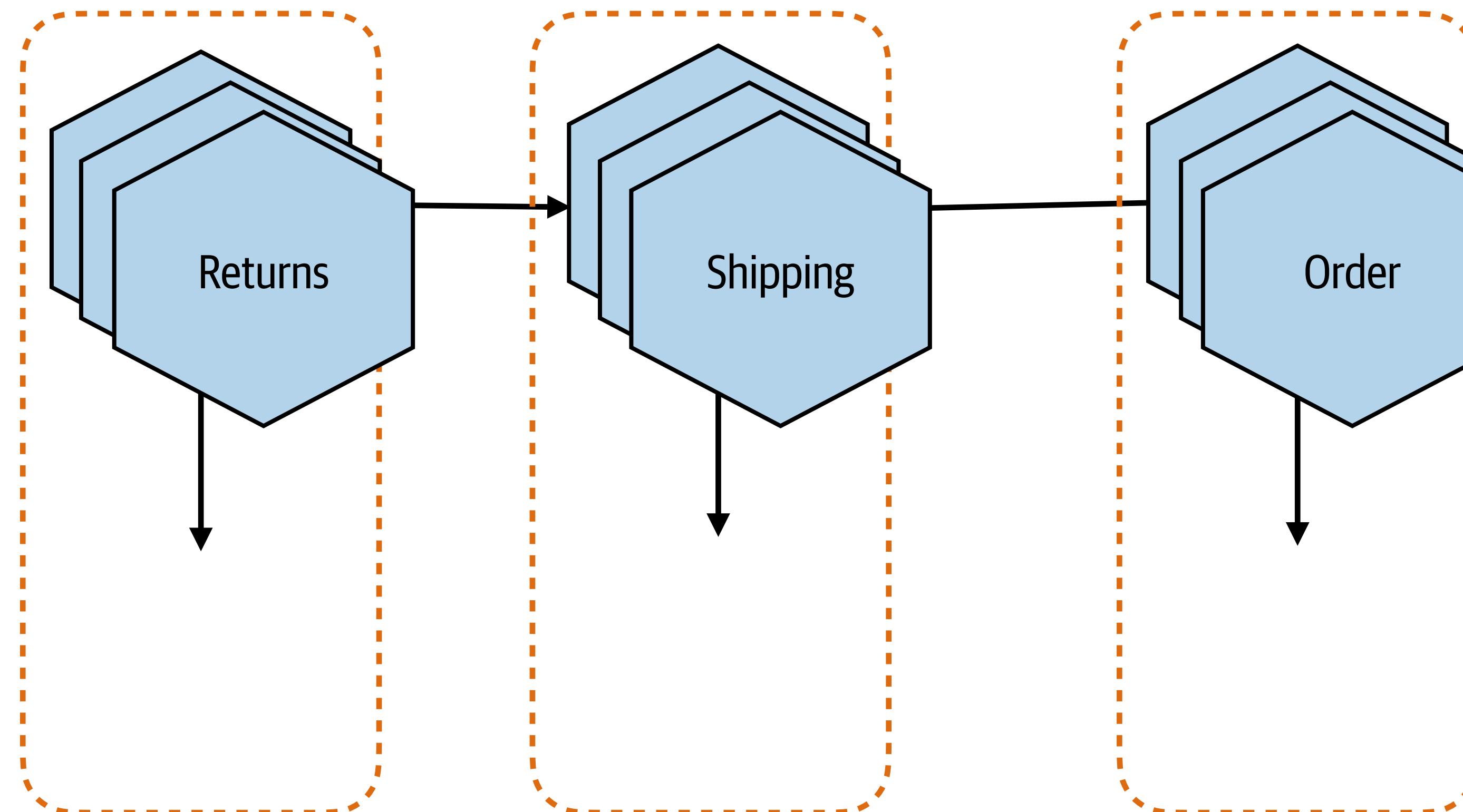


Reduces the amount of DB infra and associated admin

Single source of contention

Single source of failure

SHARED DB INFRASTRUCTURE

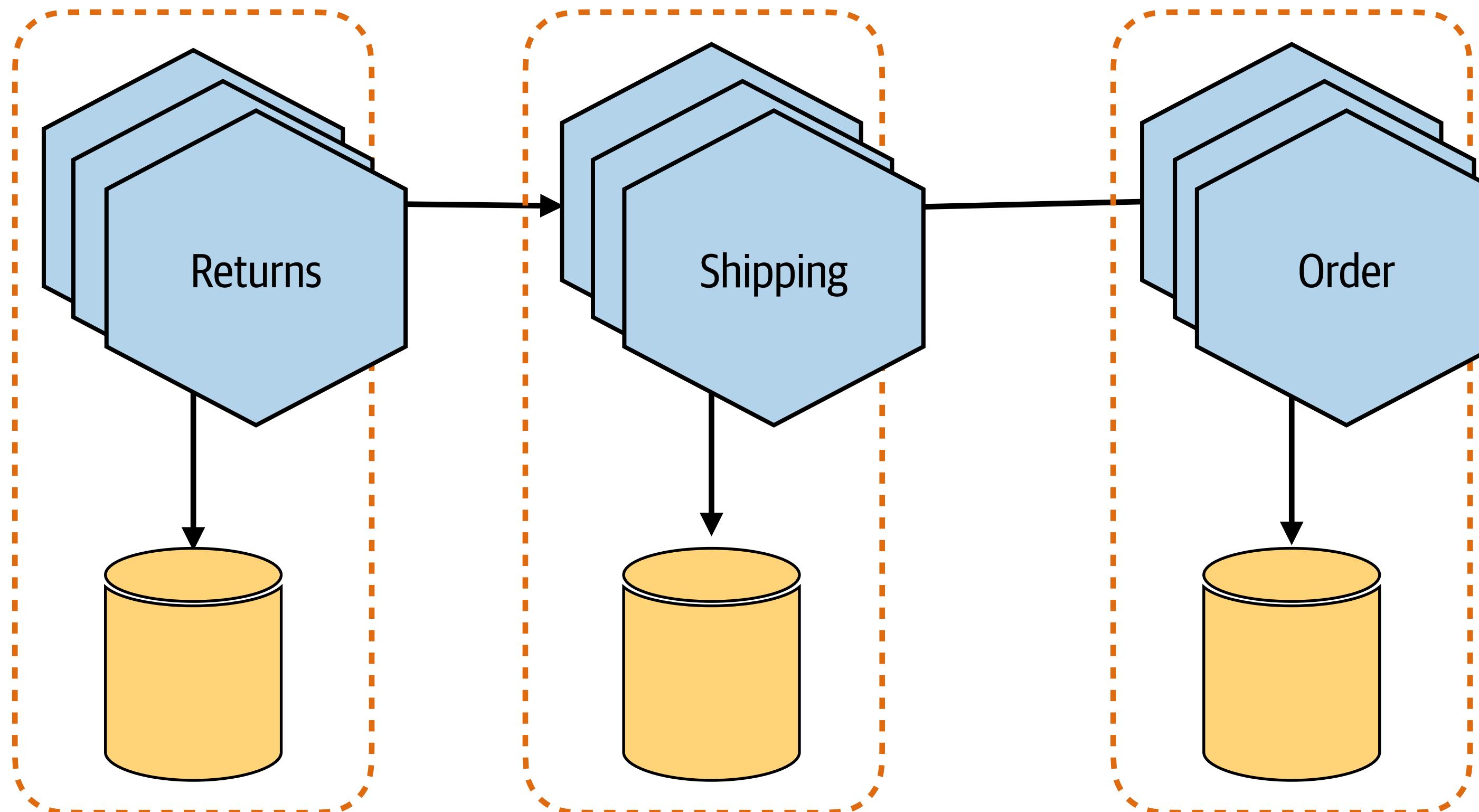


Reduces the amount of DB infra and associated admin

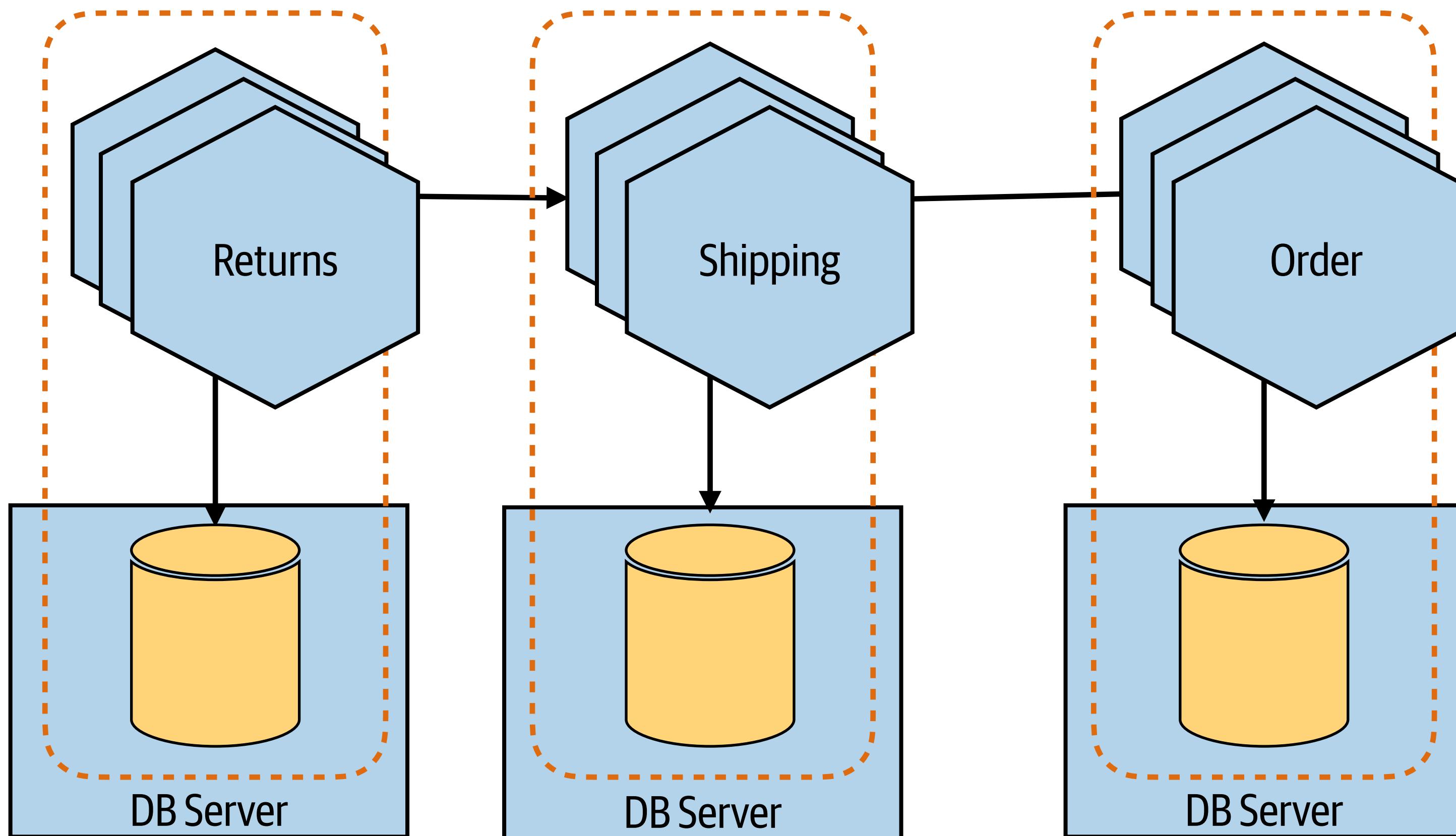
Single source of contention

Single source of failure

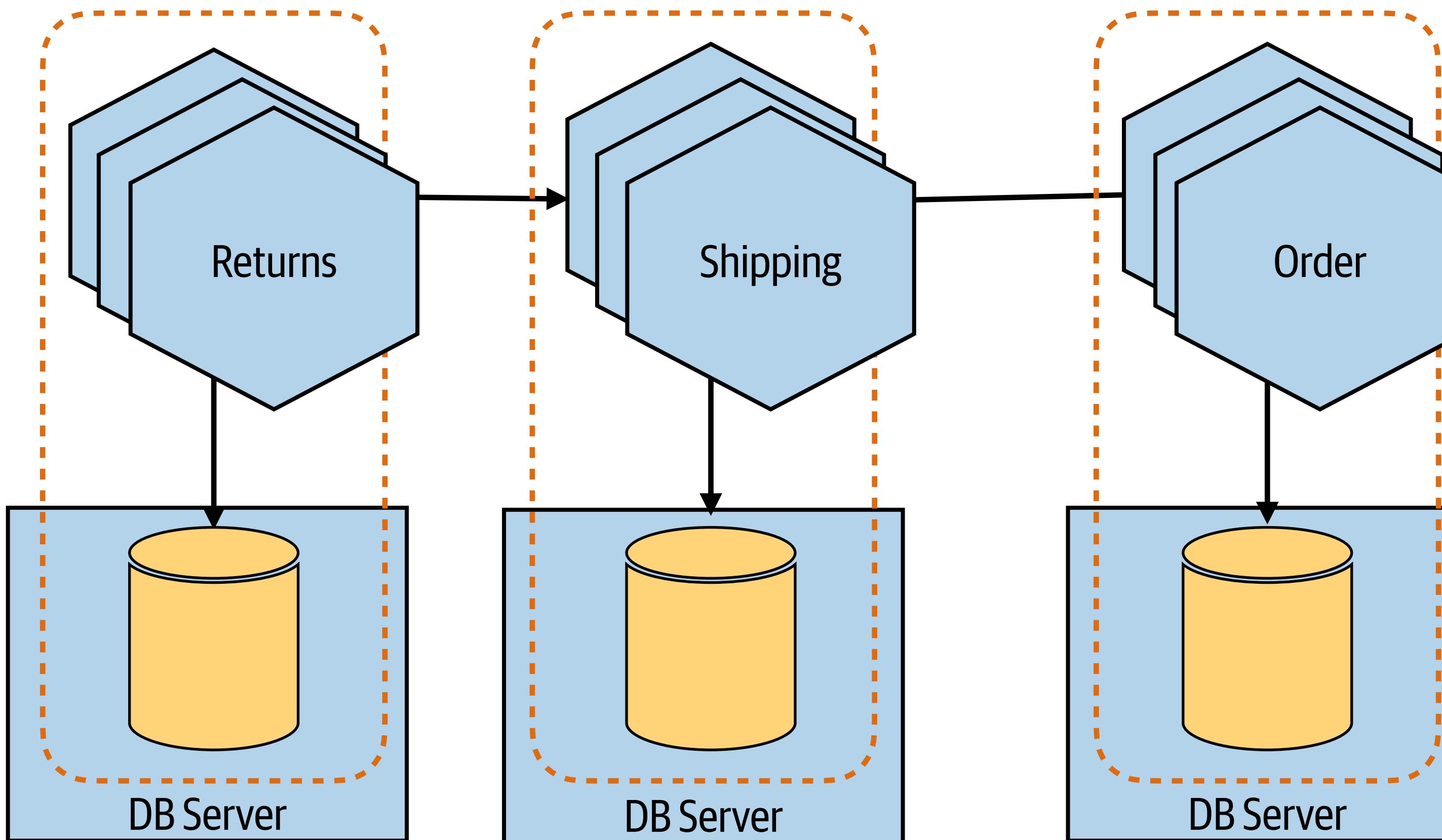
DEDICATED DB INFRASTRUCTURE



DEDICATED DB INFRASTRUCTURE

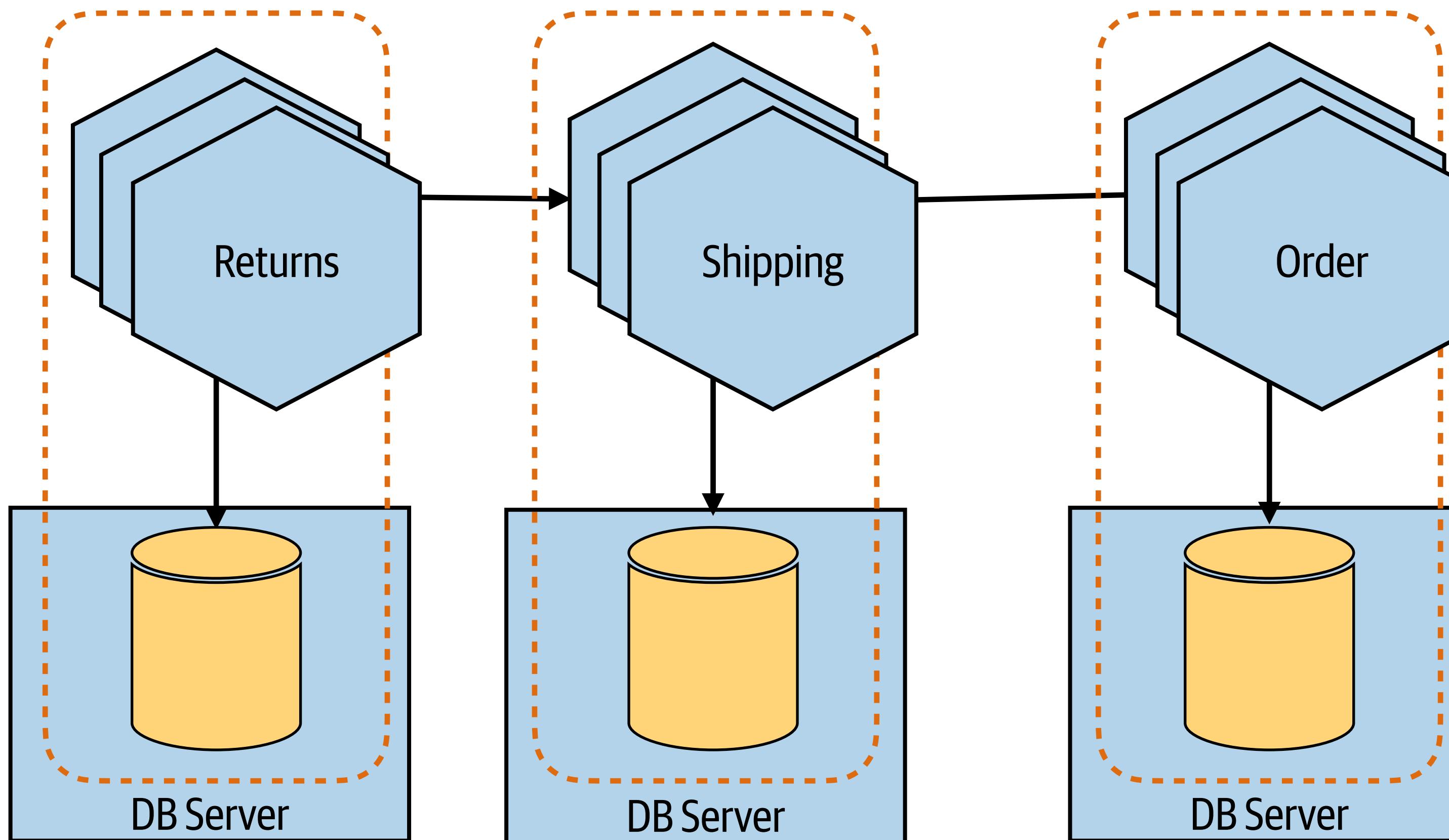


DEDICATED DB INFRASTRUCTURE



More infra to manage

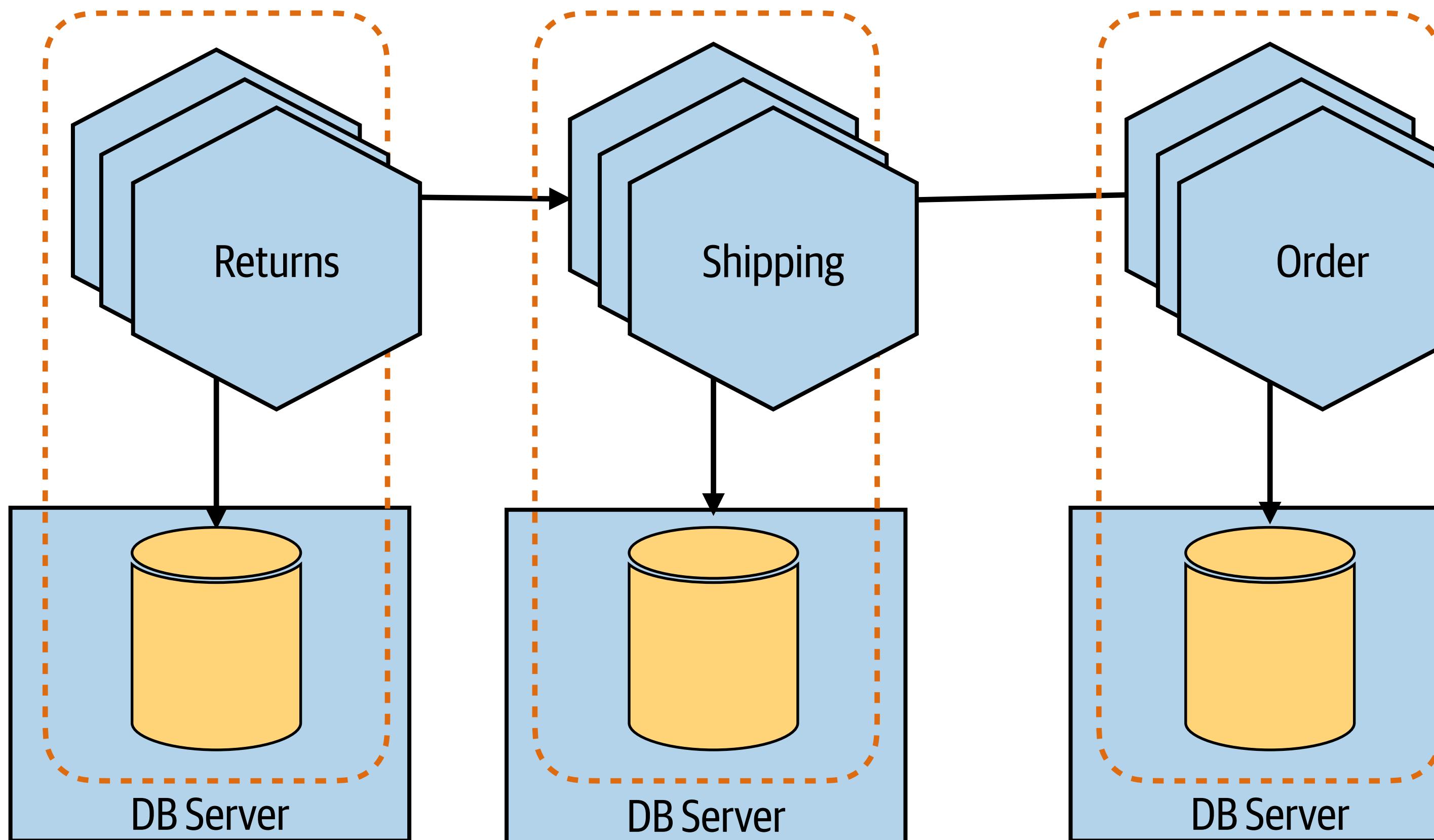
DEDICATED DB INFRASTRUCTURE



More infra to manage

Can use different technology

DEDICATED DB INFRASTRUCTURE

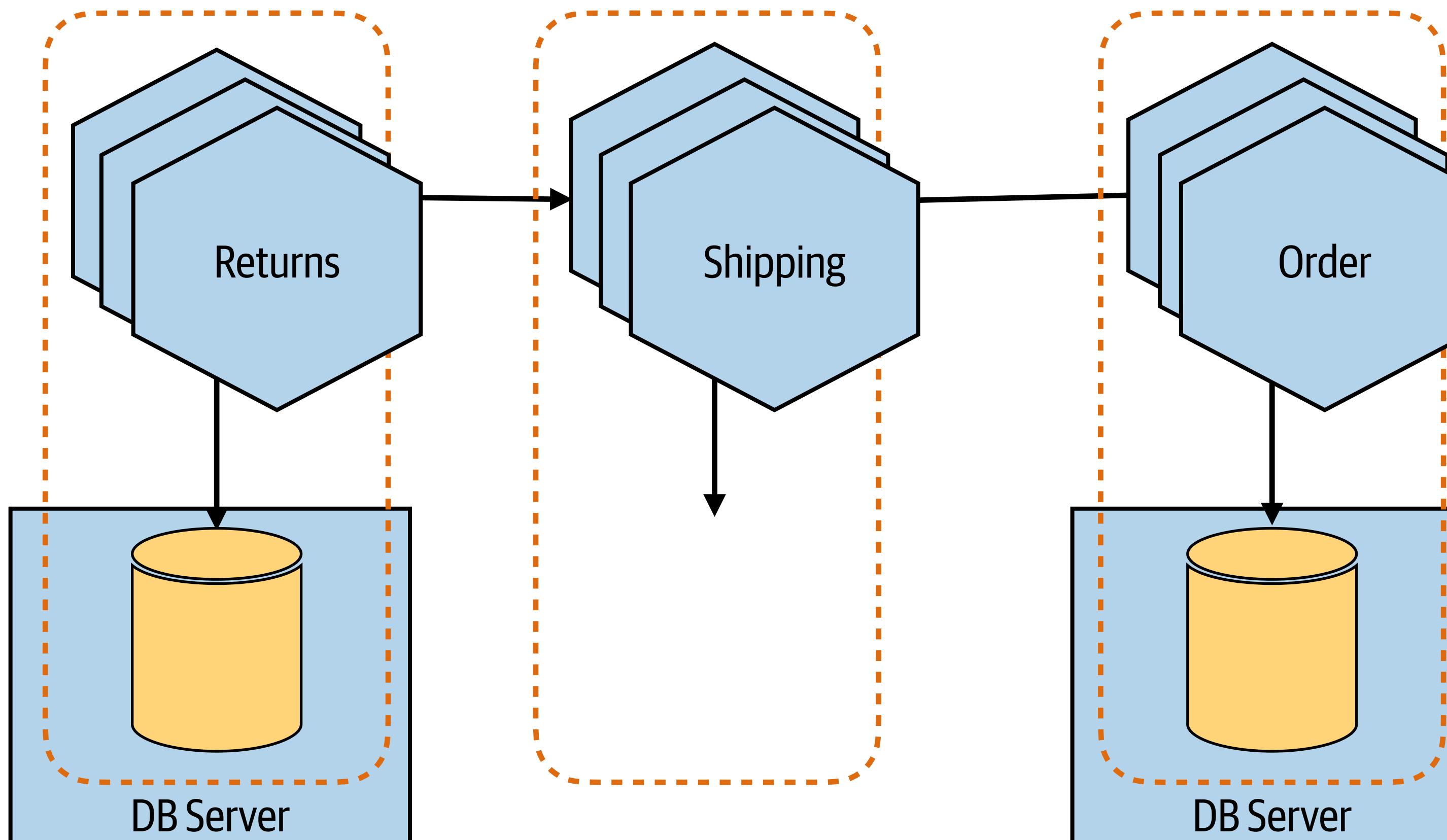


More infra to manage

Can use different technology

Avoids contention
and reduces blast
radius of any failure

DEDICATED DB INFRASTRUCTURE



More infra to manage

Can use different technology

Avoids contention
and reduces blast
radius of any failure

POLL: WHAT IS THE MAIN TYPE OF DATABASE TECHNOLOGY YOU USE?

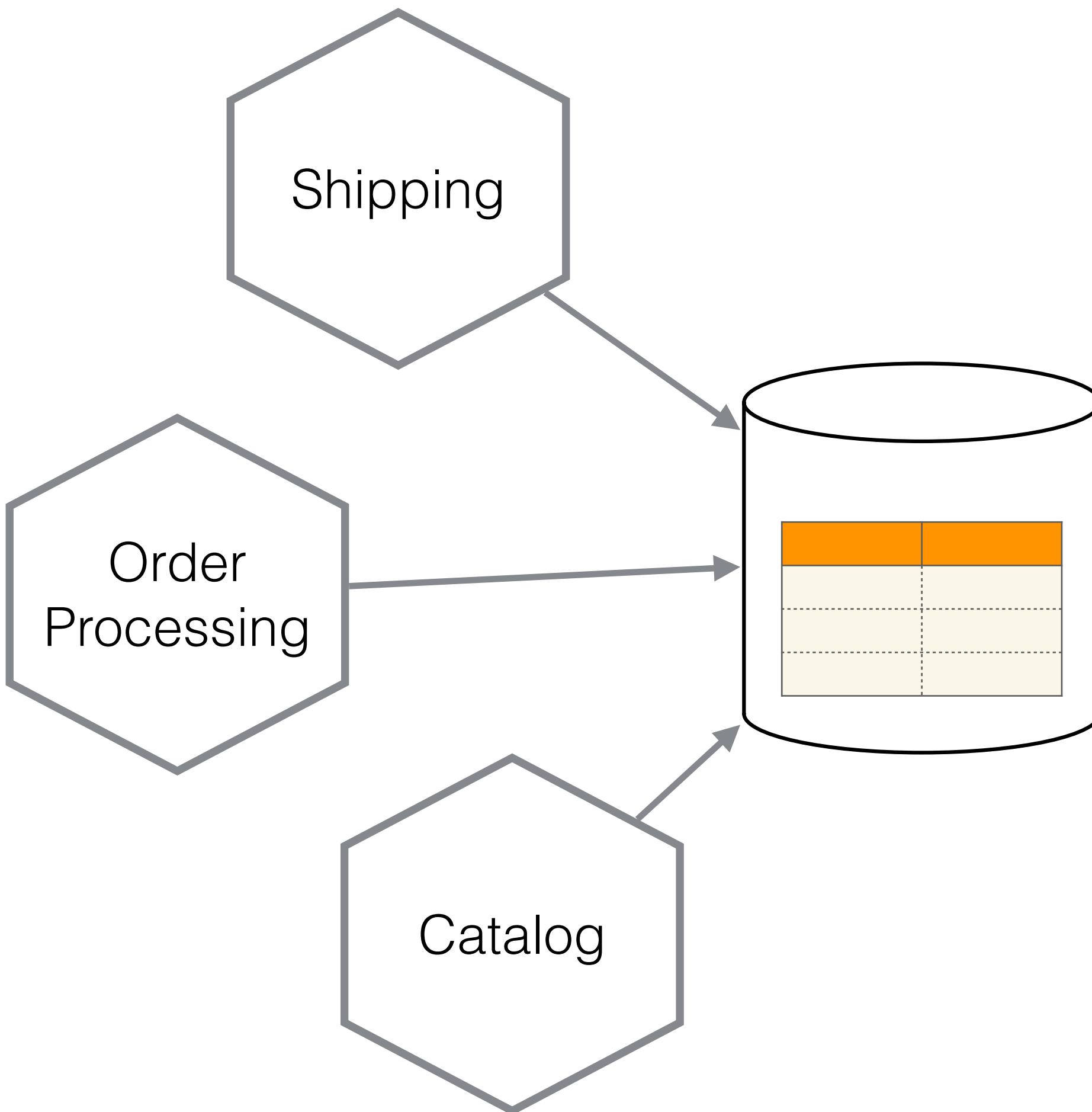
Relational (e.g. mysql, postgres, oracle)

Document (e.g. couchDB, mongo)

Column (e.g. Cassandra)

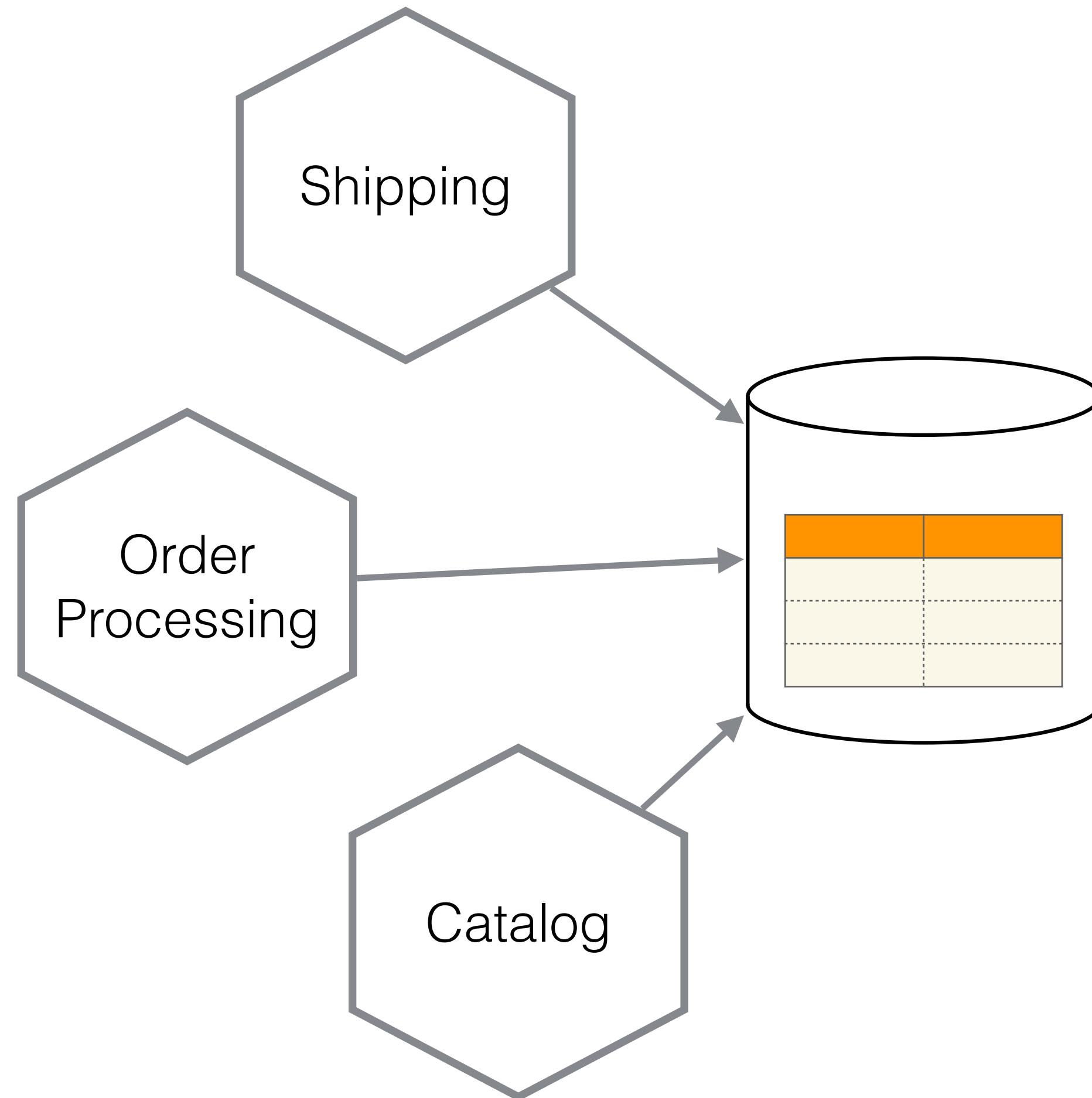
Graph (e.g. Neo)

PATTERN: SHARED DATABASE



Multiple services making use of the same database

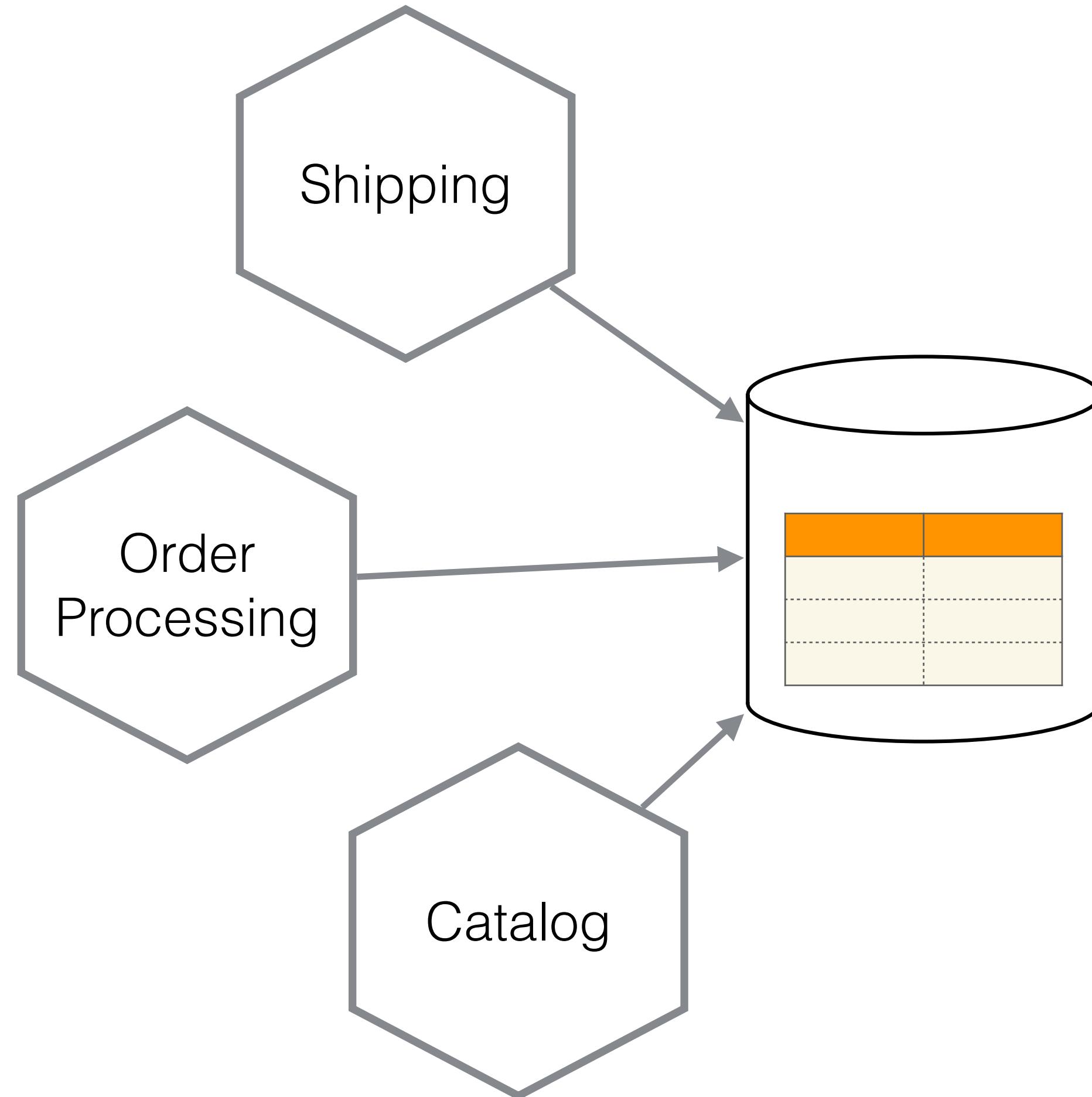
PATTERN: SHARED DATABASE



Multiple services making use of the same database

Changing the shared DB can be very disruptive

PATTERN: SHARED DATABASE

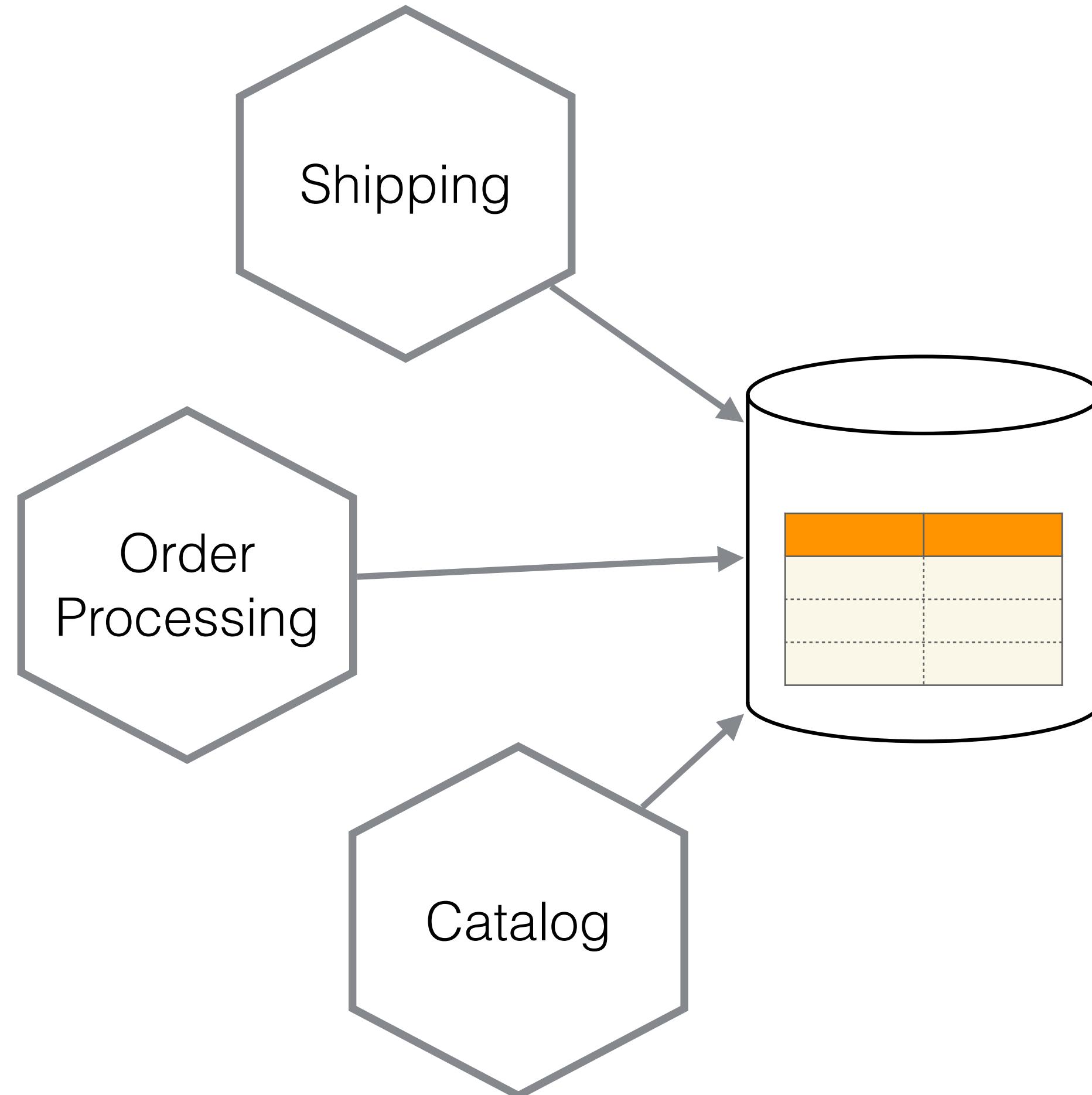


Multiple services making use of the same database

Changing the shared DB can be very disruptive

Potential bottleneck

PATTERN: SHARED DATABASE



Multiple services making use of the same database

Changing the shared DB can be very disruptive

Potential bottleneck

Logic for managing the data is spread around the system

**Be really cautious about using a shared
database**

Be really cautious about using a shared
database

It can make information hiding, and
therefore independent deployability, much
more difficult

WHEN TO USE A SHARED DATABASE?

WHEN TO USE A SHARED DATABASE?

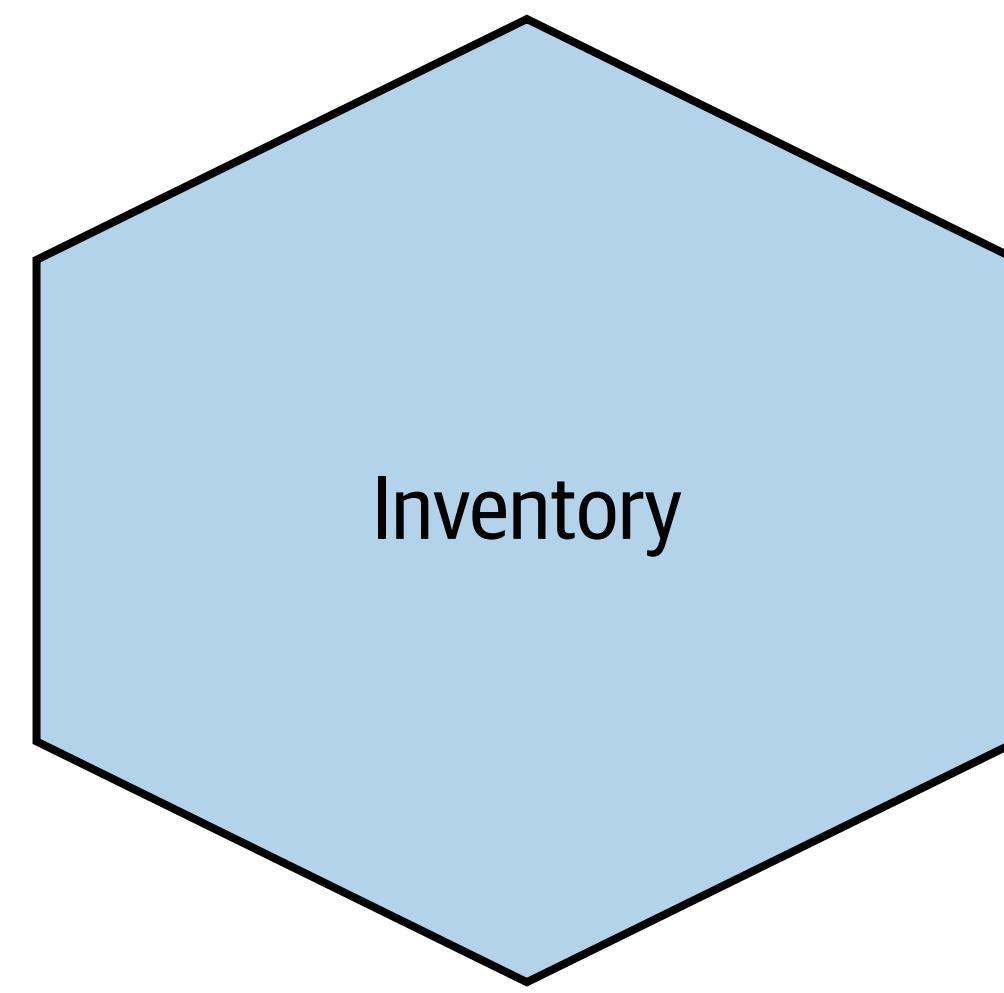
**Static read-only reference
data**

WHEN TO USE A SHARED DATABASE?

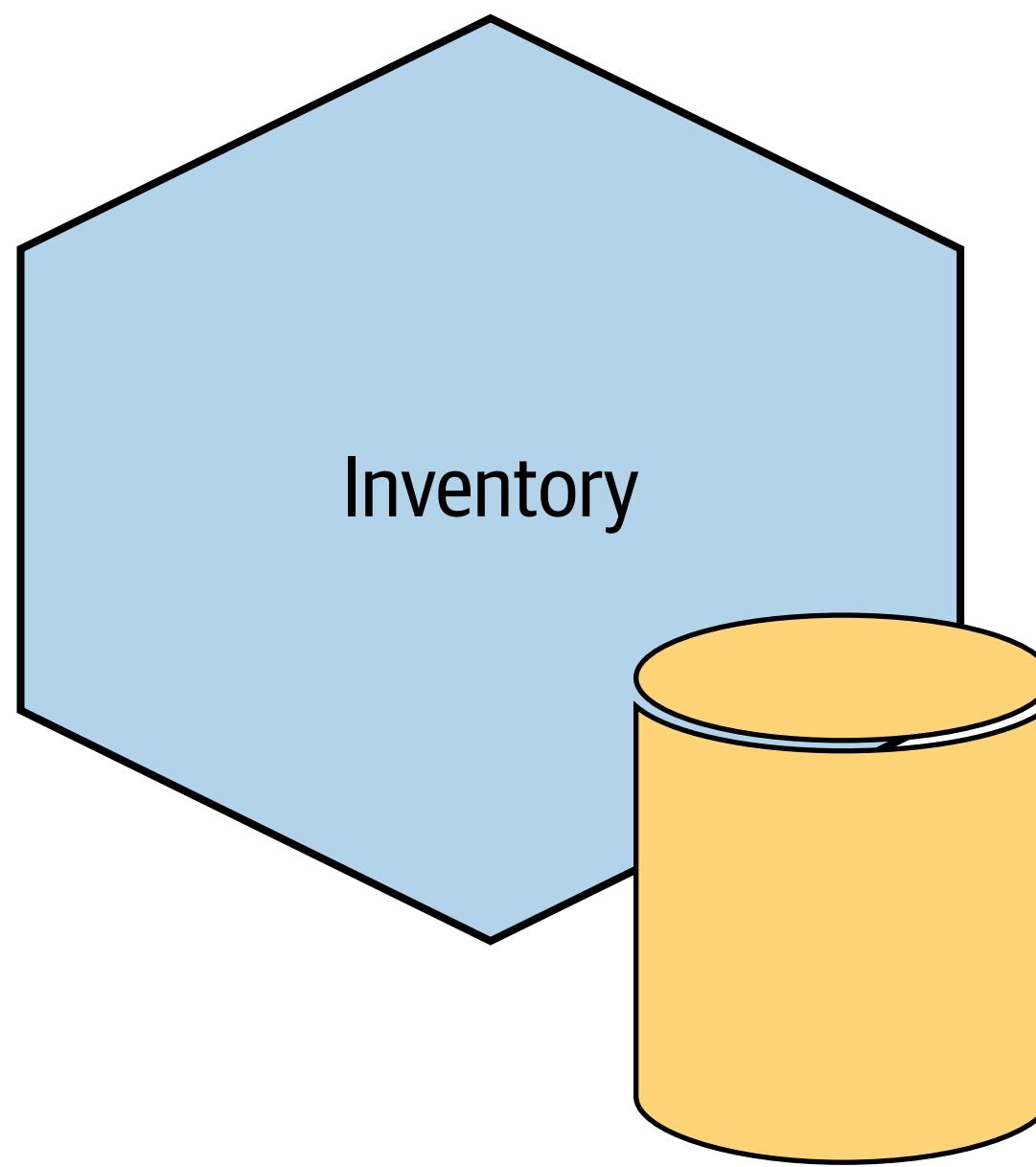
**Static read-only reference
data**

**As part of an incremental
decomposition**

CONSUMER NEEDS...

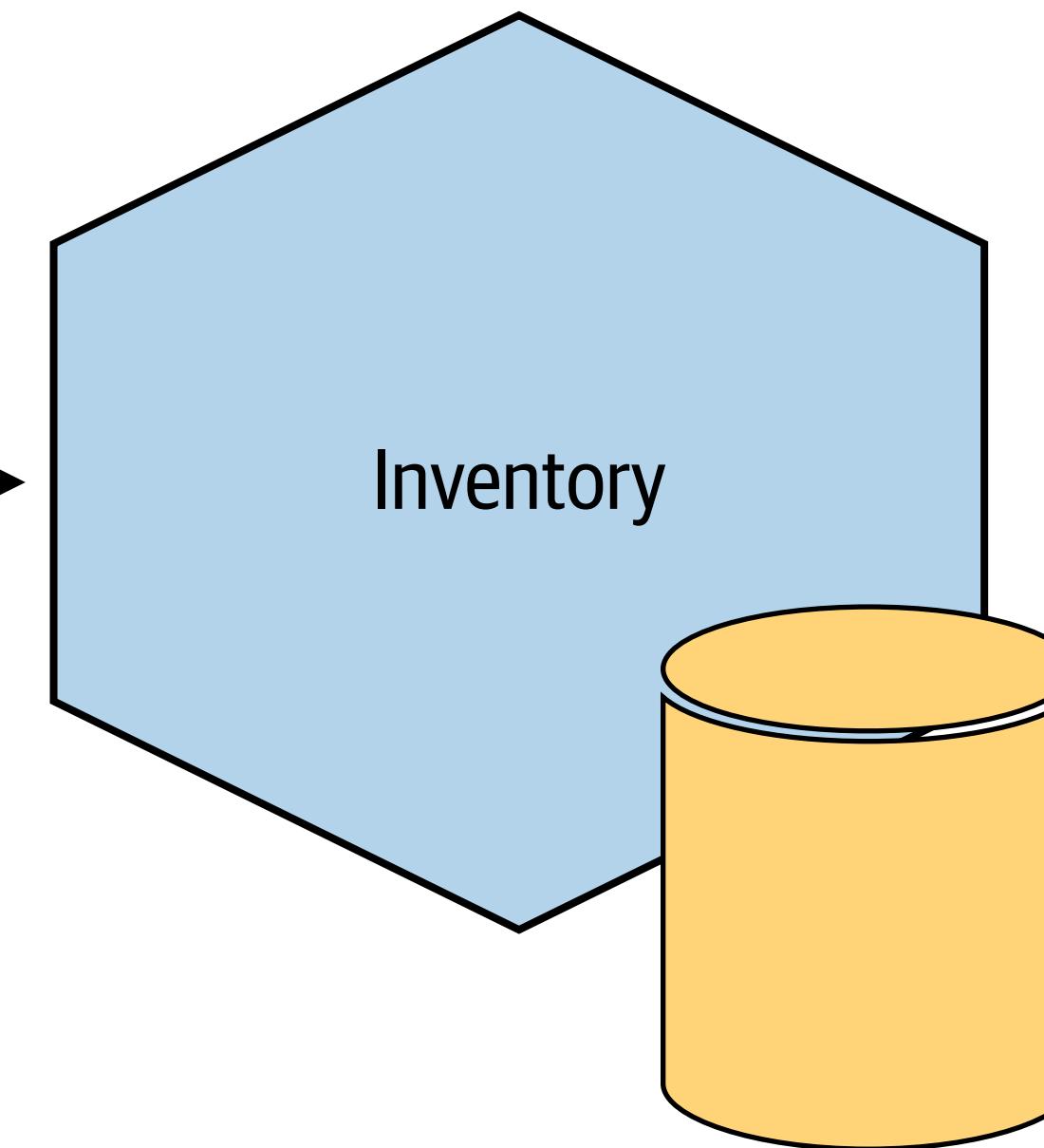


CONSUMER NEEDS...



CONSUMER NEEDS...

How much Bieber do we have?

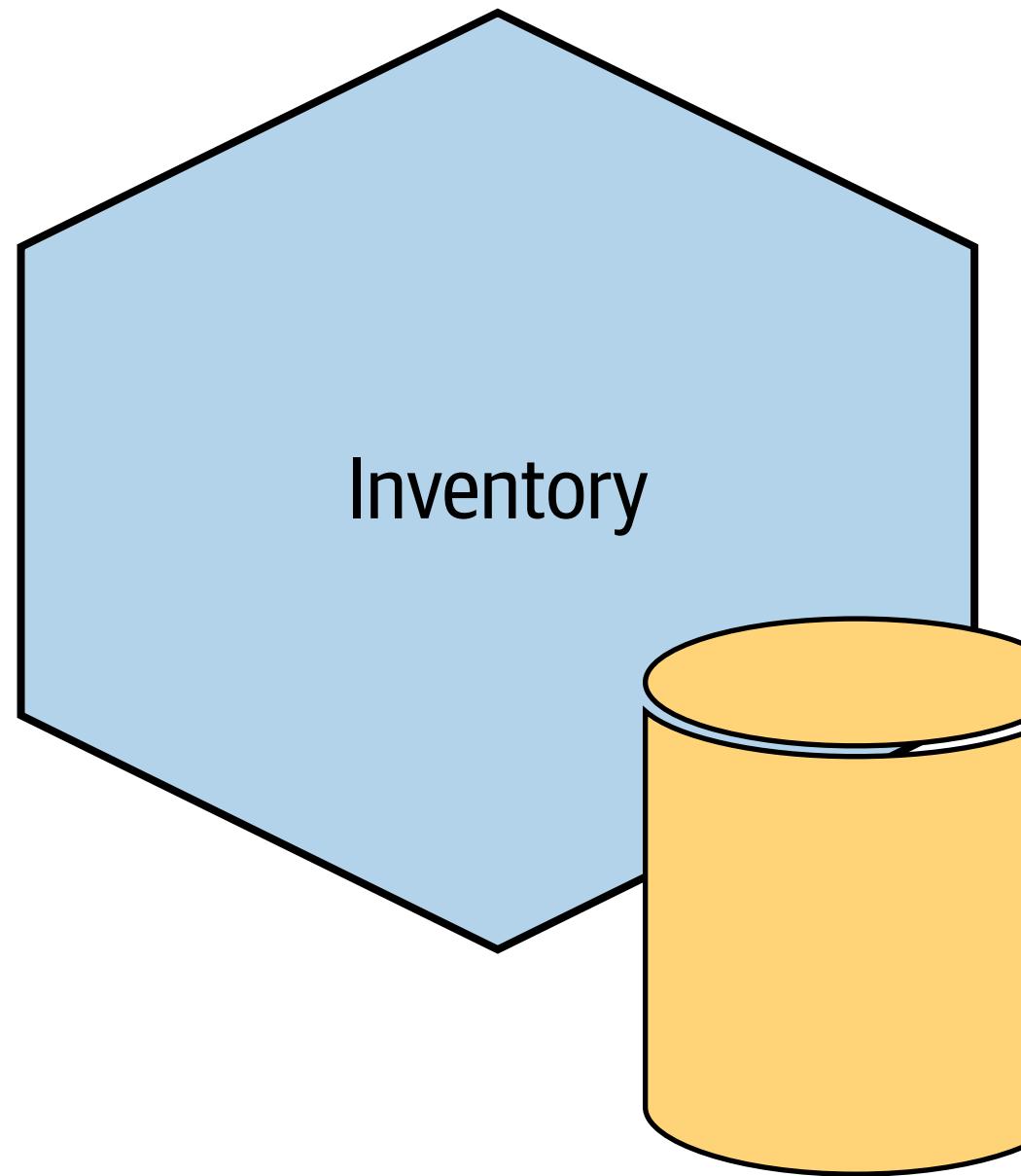


CONSUMER NEEDS...

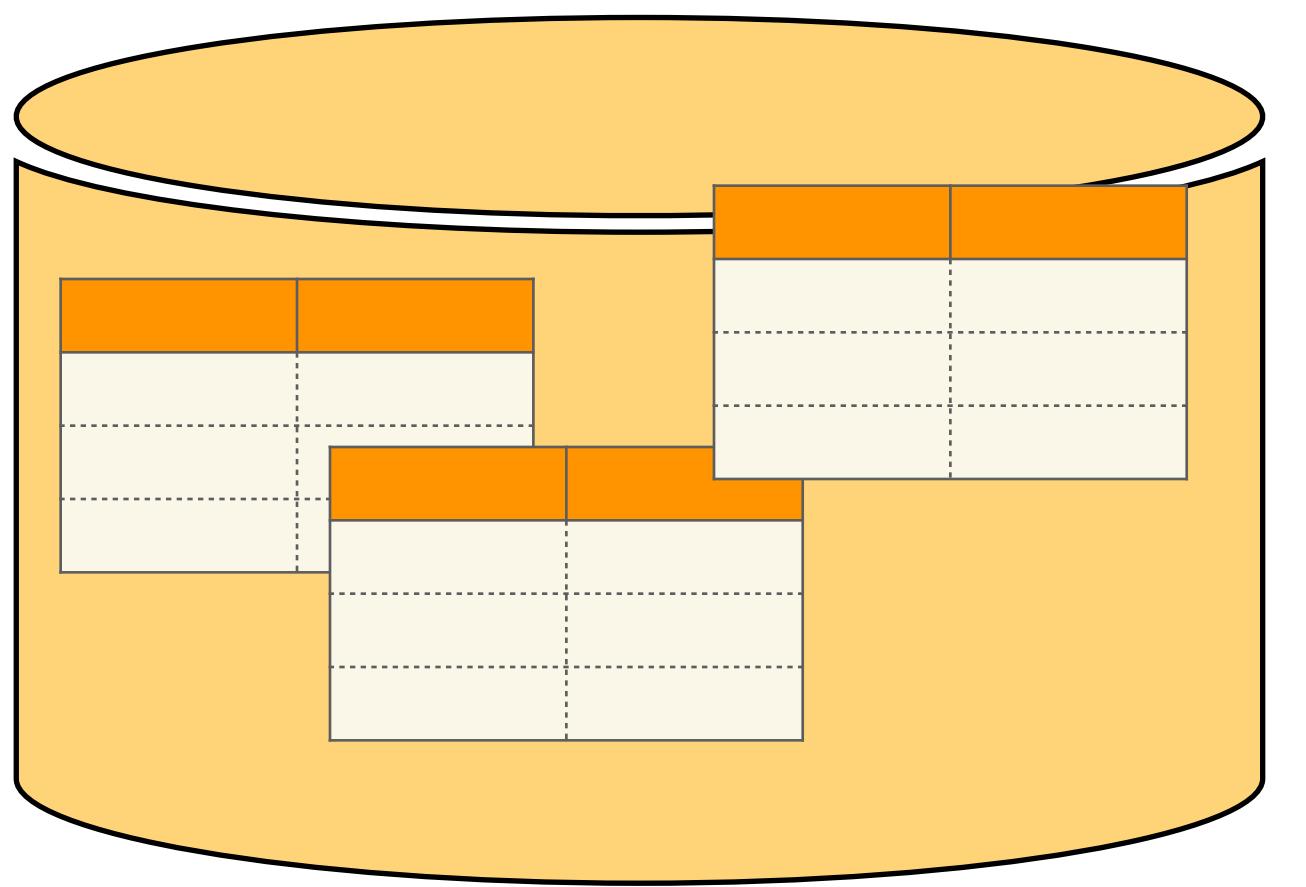
How much Bieber do we have?



```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

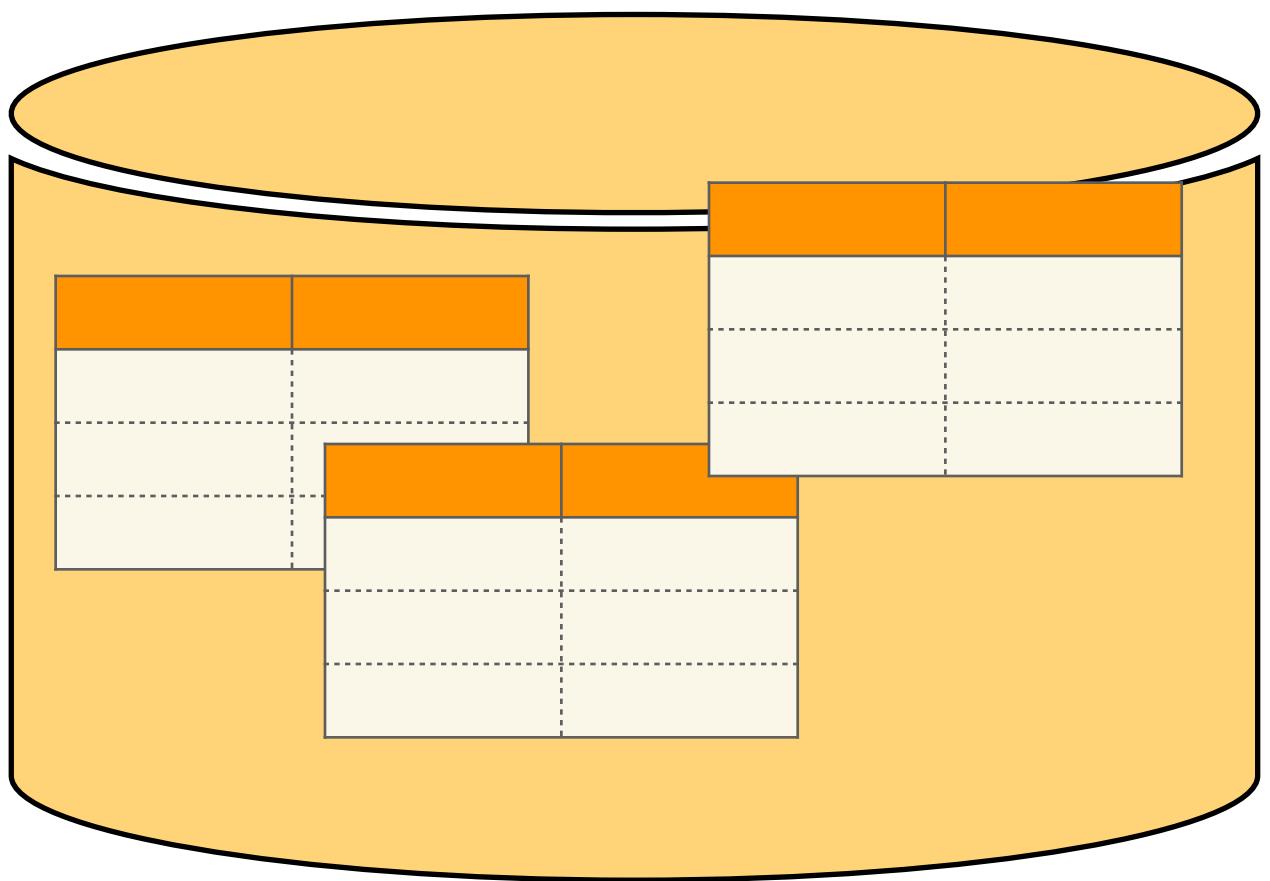


EXAMPLE - INVENTORY MANAGEMENT



Inventory

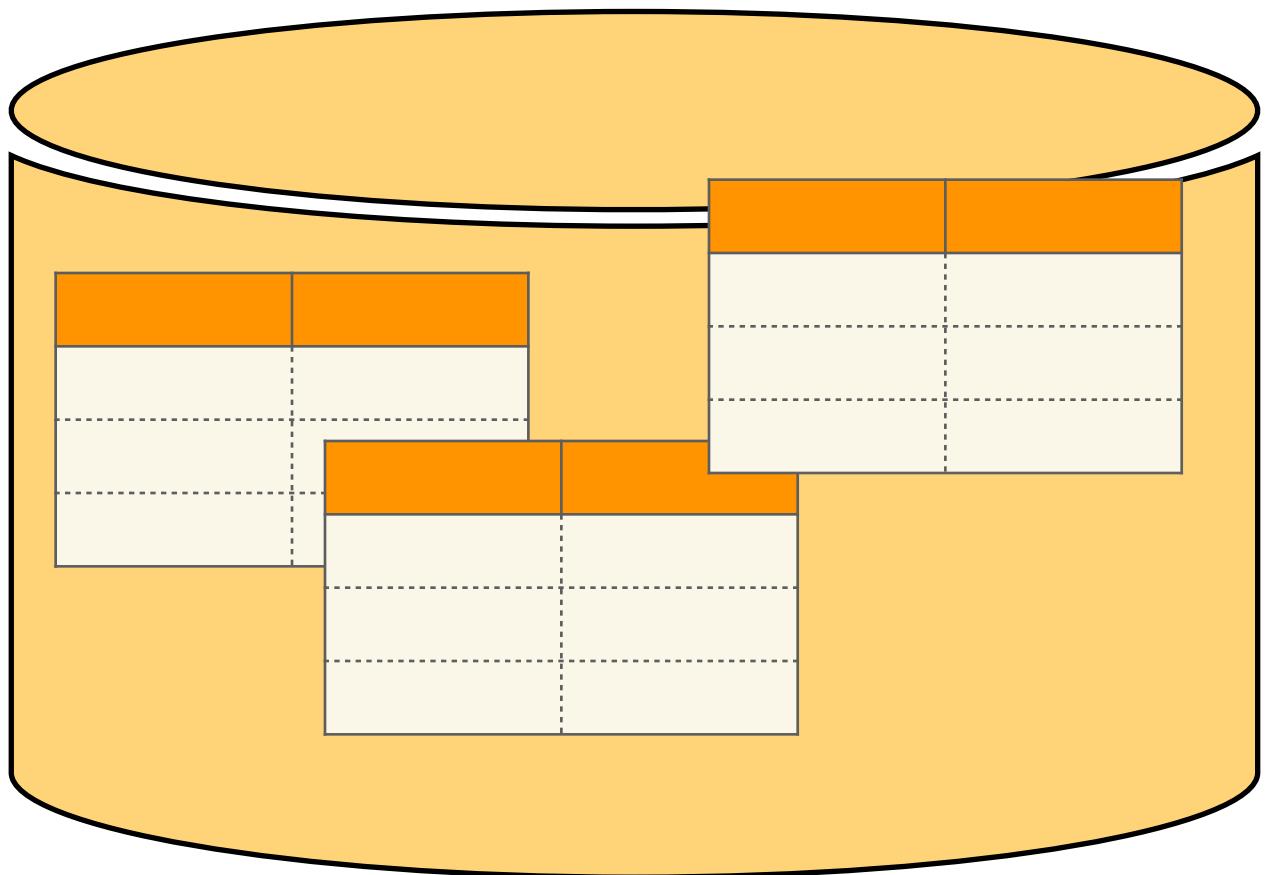
EXAMPLE - INVENTORY MANAGEMENT



Inventory

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1
Item		

EXAMPLE - INVENTORY MANAGEMENT



Inventory

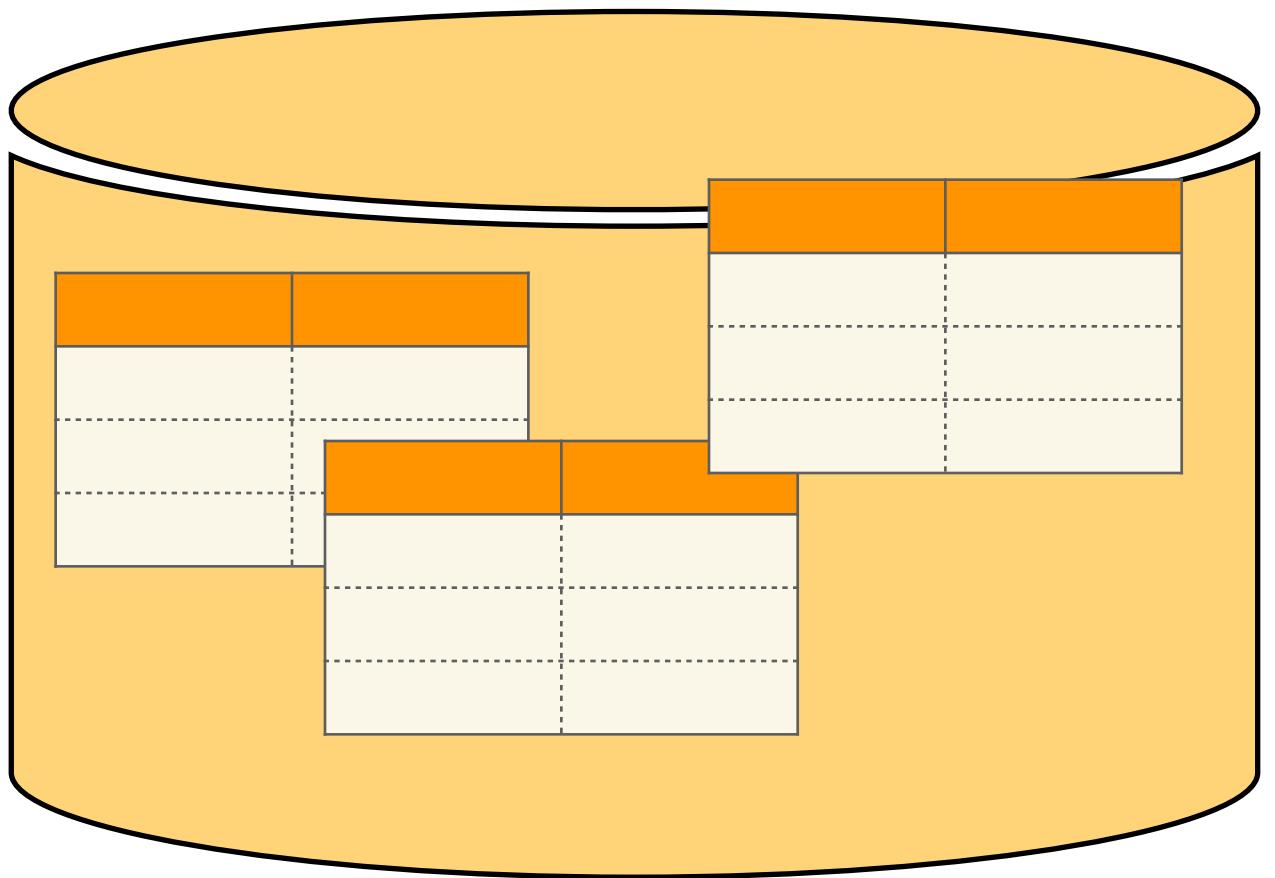
SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

EXAMPLE - INVENTORY MANAGEMENT



Inventory

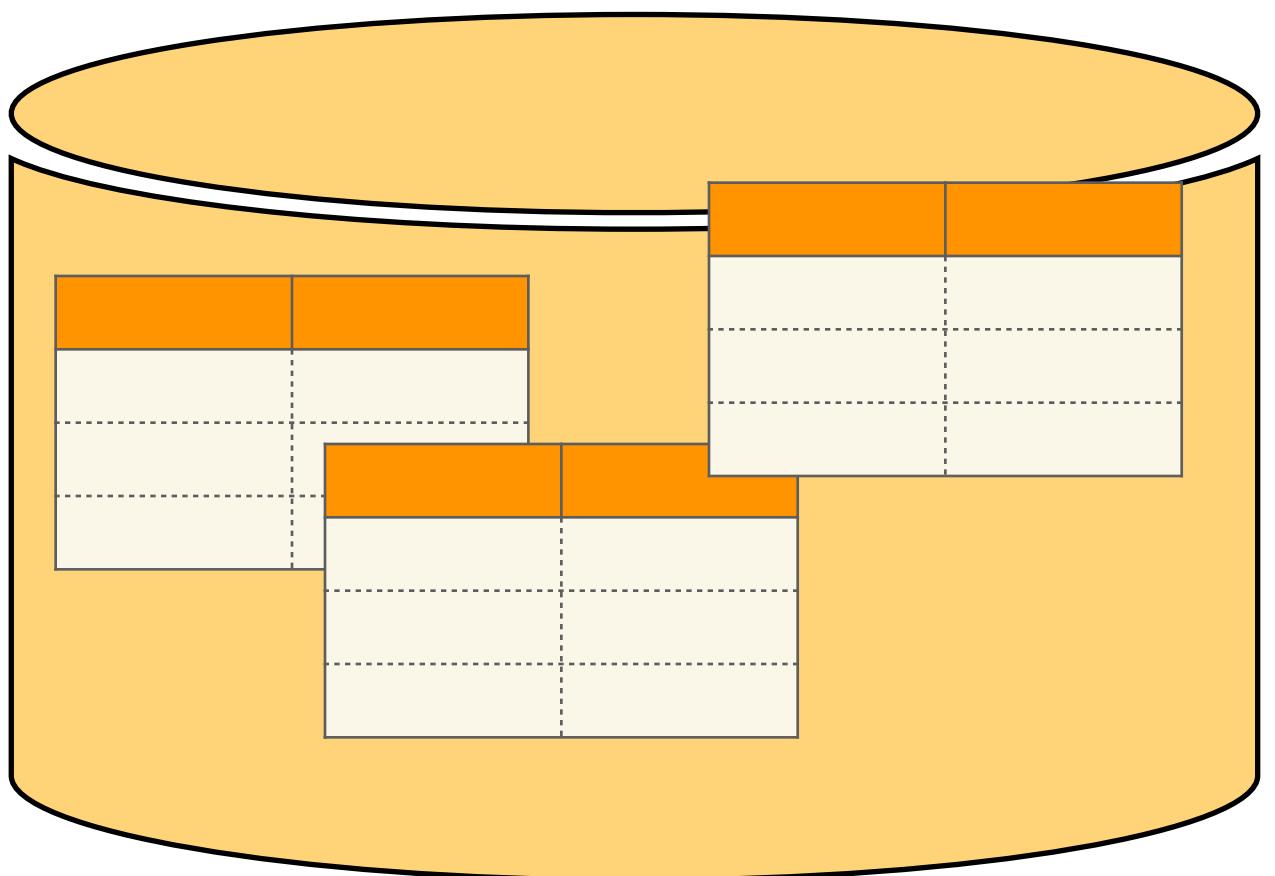
SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

EXAMPLE - INVENTORY MANAGEMENT



Inventory

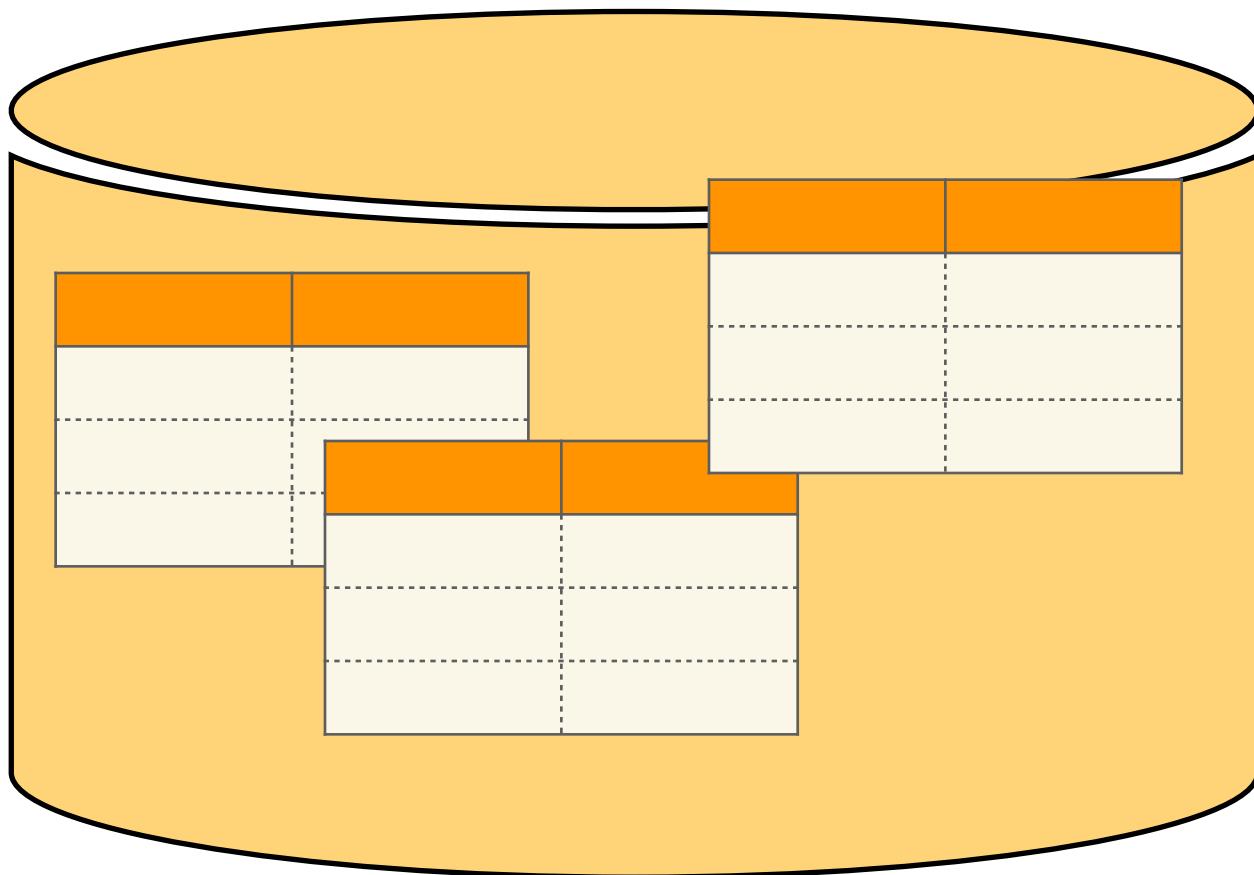
SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

EXAMPLE - INVENTORY MANAGEMENT



Inventory

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

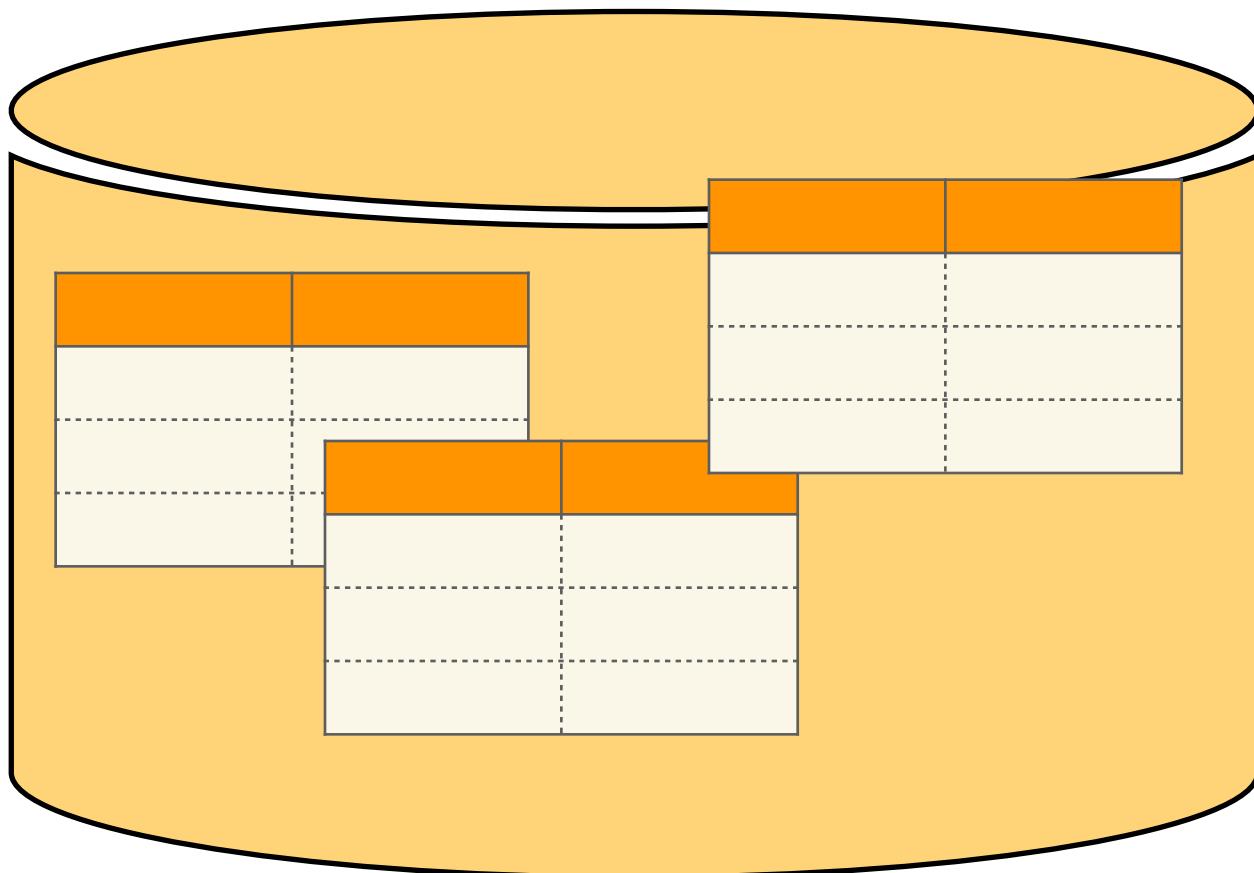
SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

EXAMPLE - INVENTORY MANAGEMENT



Inventory

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

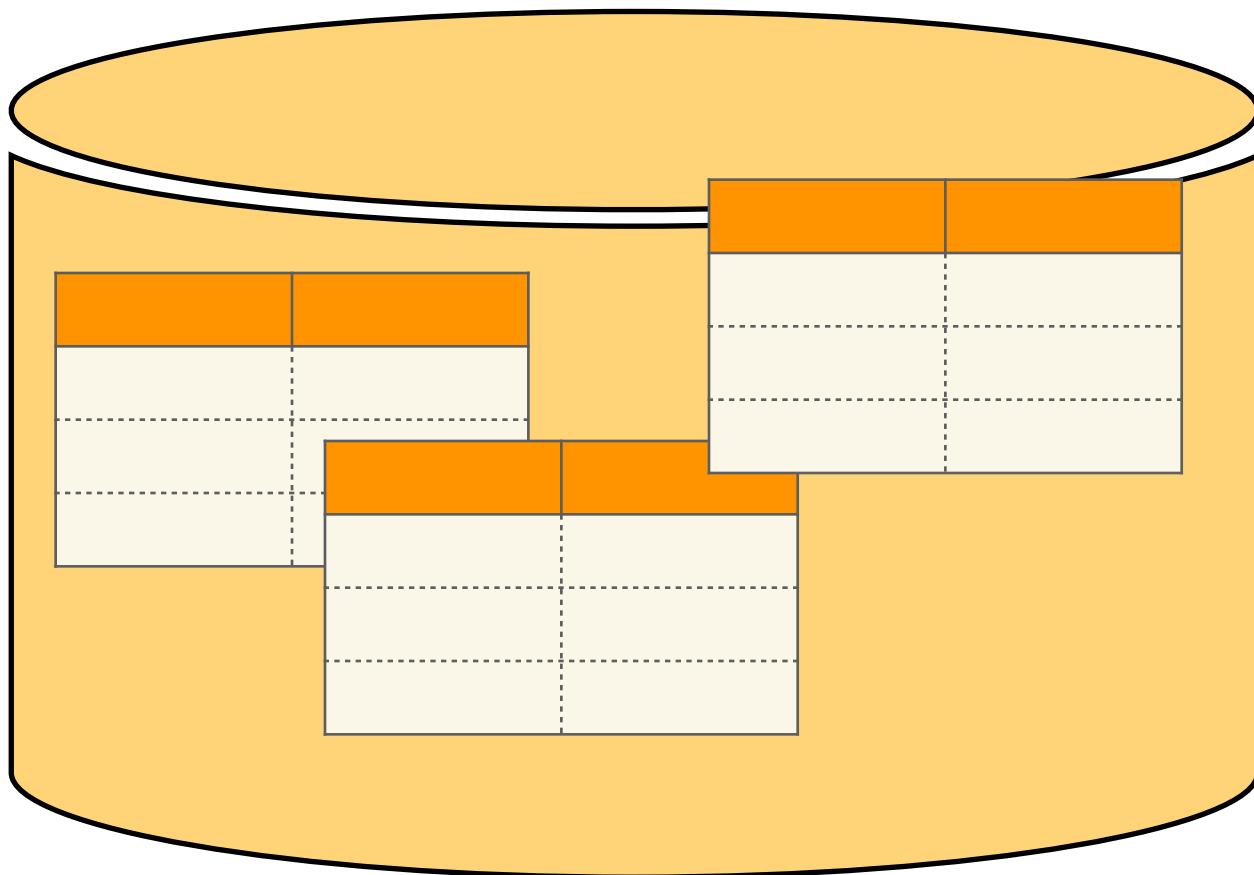
SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

EXAMPLE - INVENTORY MANAGEMENT



Inventory

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

LOTS OF DETAIL CAN BE HIDDEN...

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

LOTS OF DETAIL CAN BE HIDDEN...

```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

LOTS OF DETAIL CAN BE HIDDEN...

```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

LOTS OF DETAIL CAN BE HIDDEN...

```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

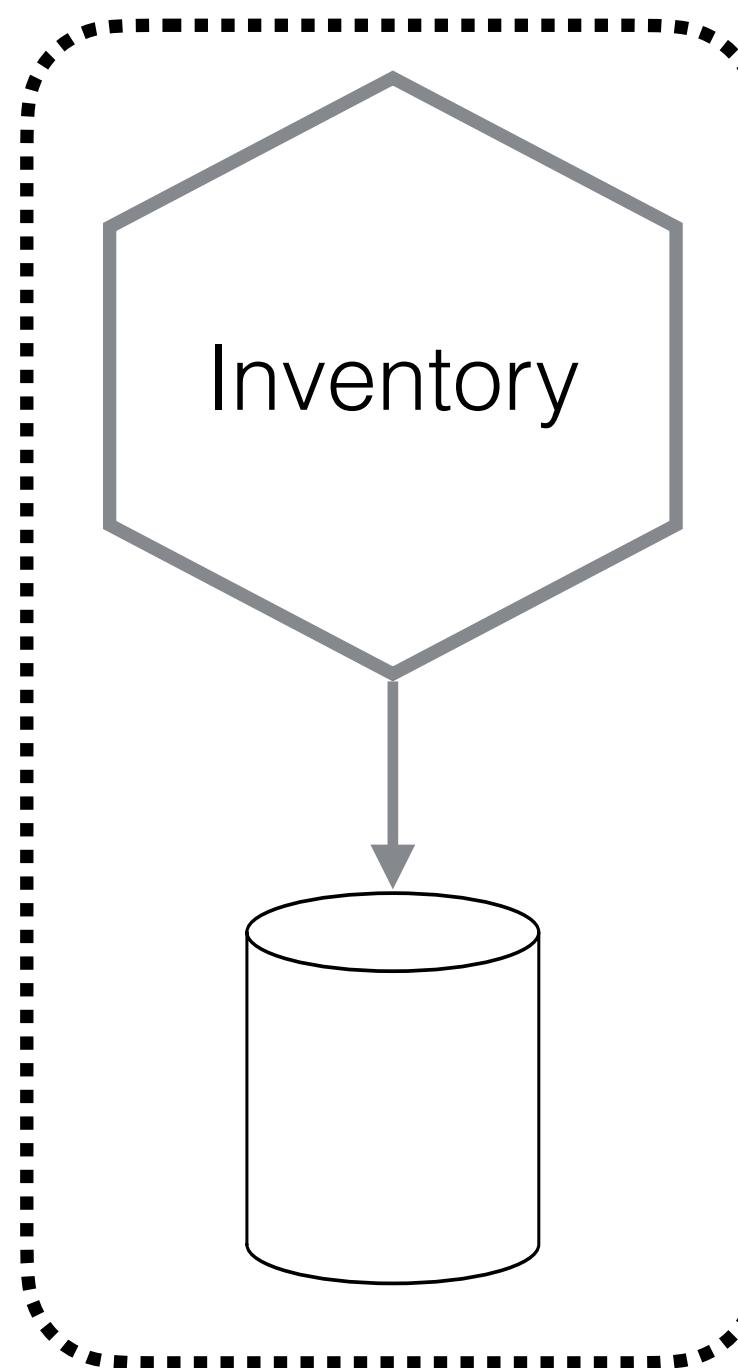
SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

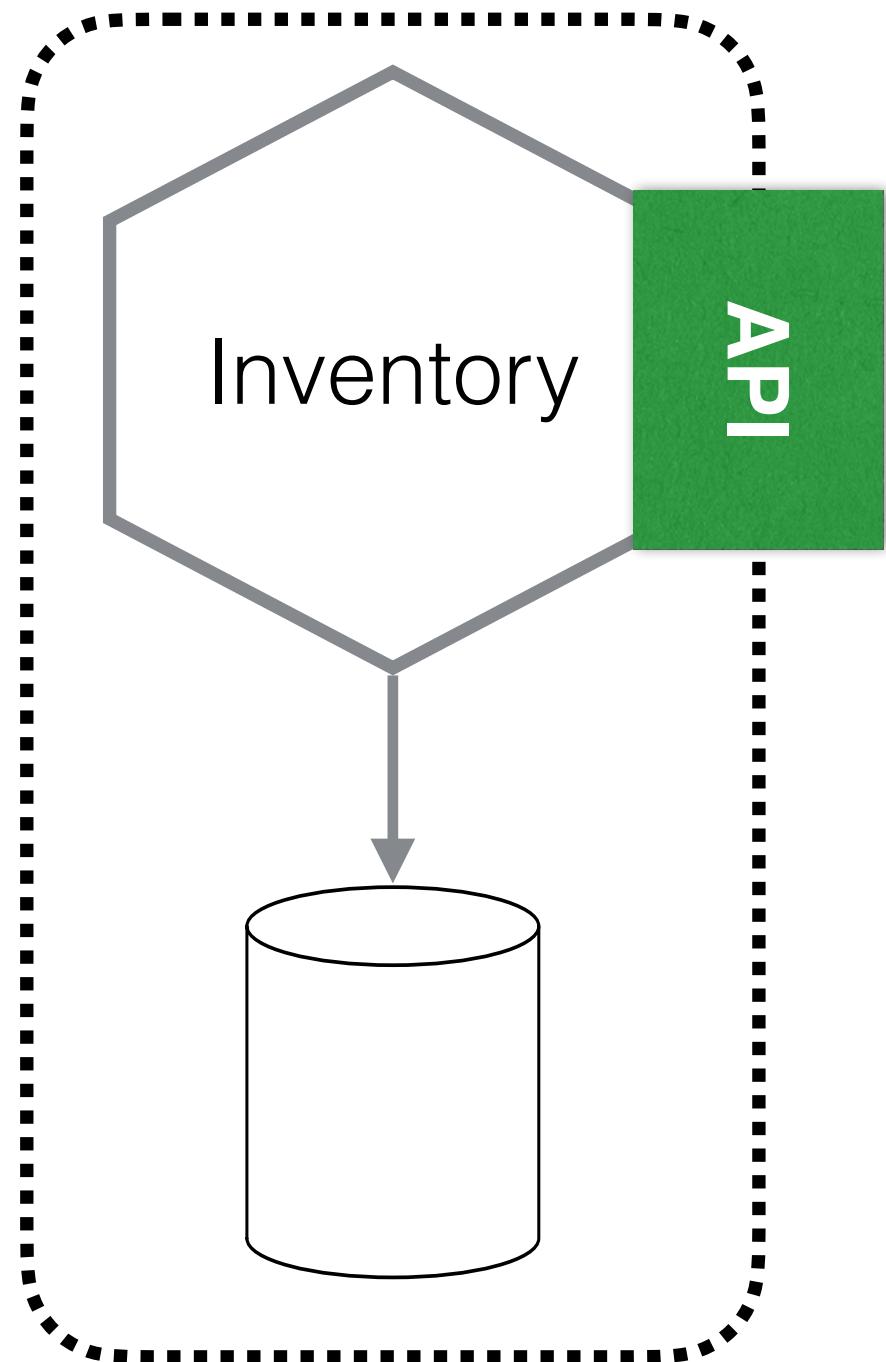
ID	Company Name	Homepage	VAT	Contact
abc-1	Acme

Supplier Table

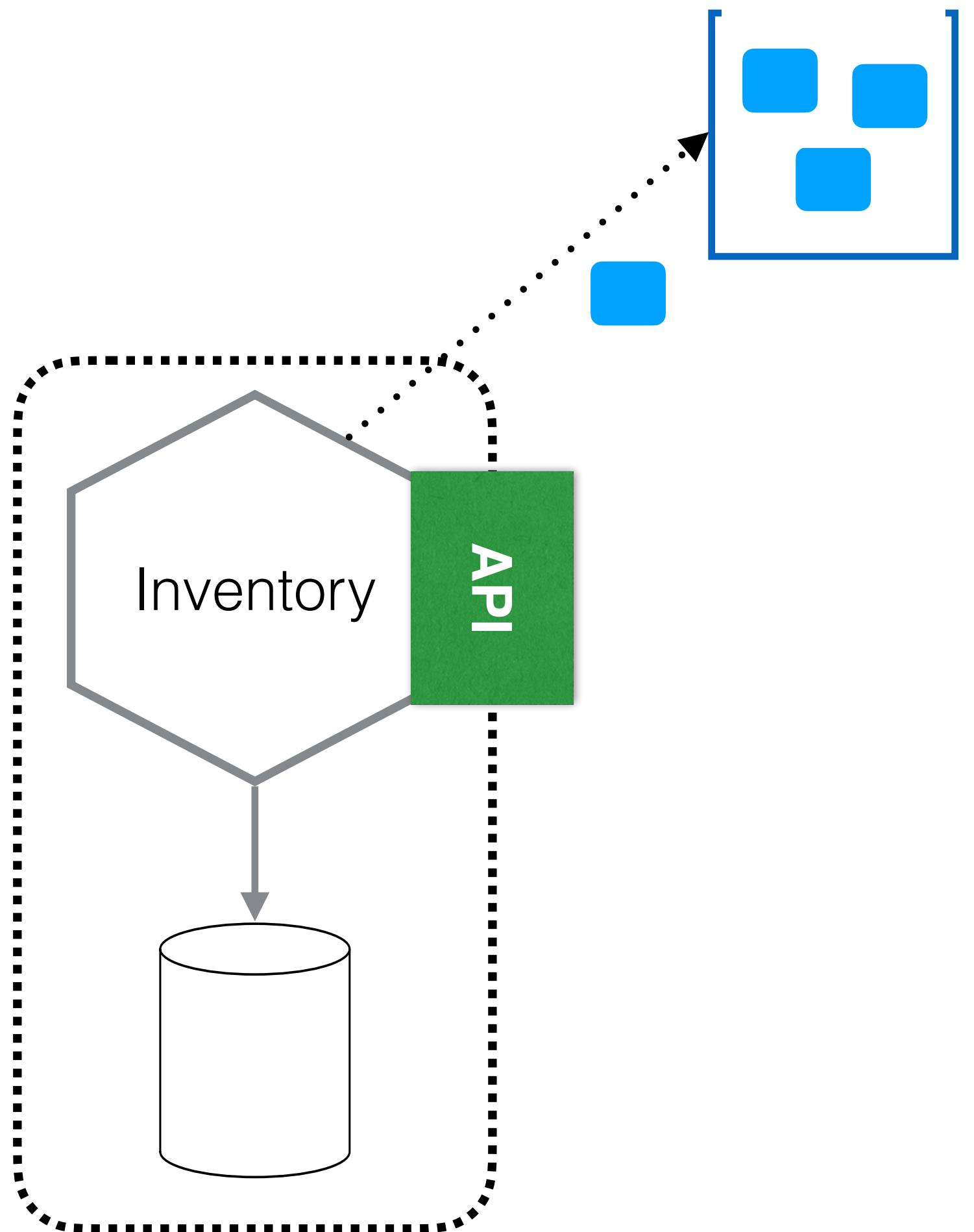
PATTERN: DATABASE AS AN ENDPOINT



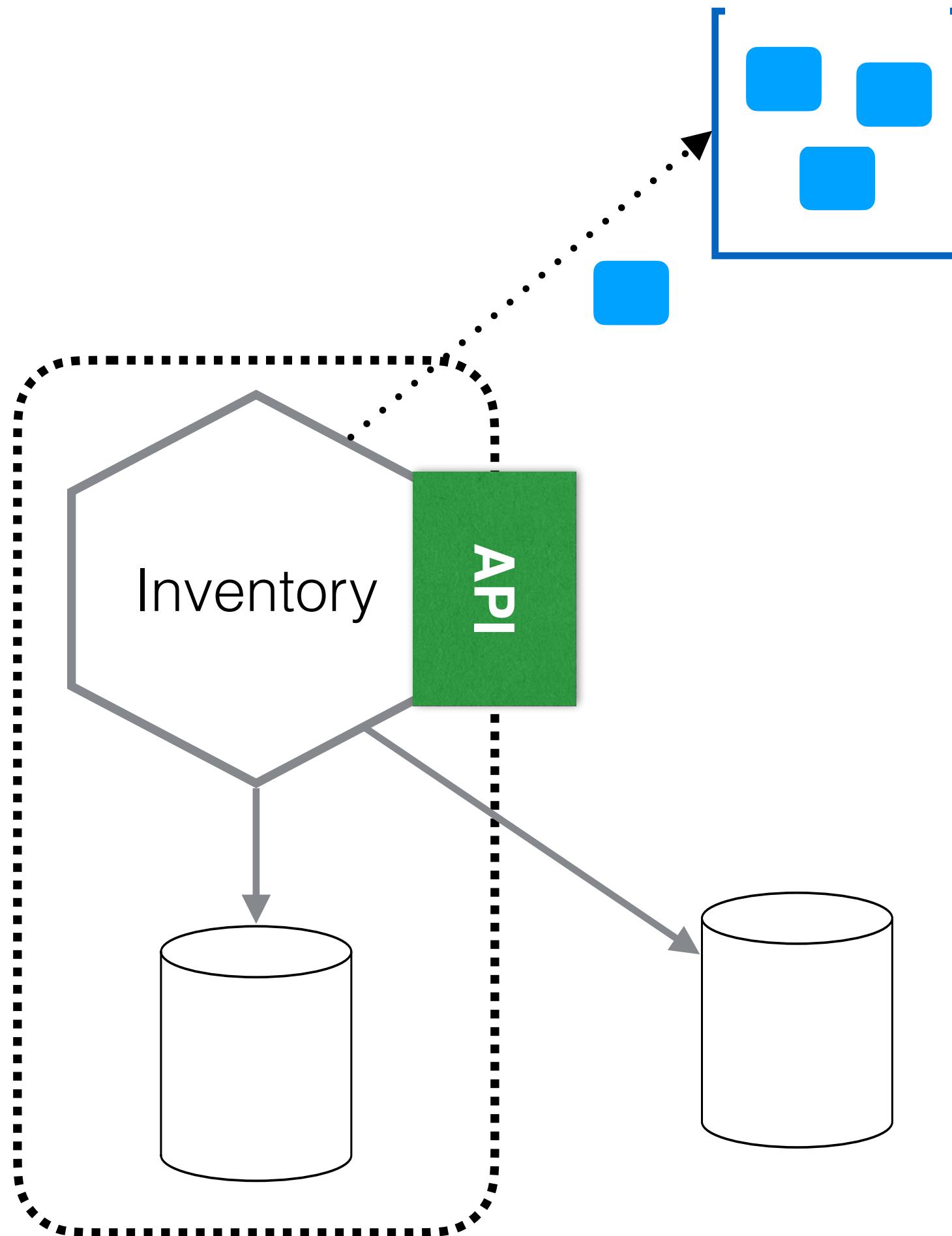
PATTERN: DATABASE AS AN ENDPOINT



PATTERN: DATABASE AS AN ENDPOINT

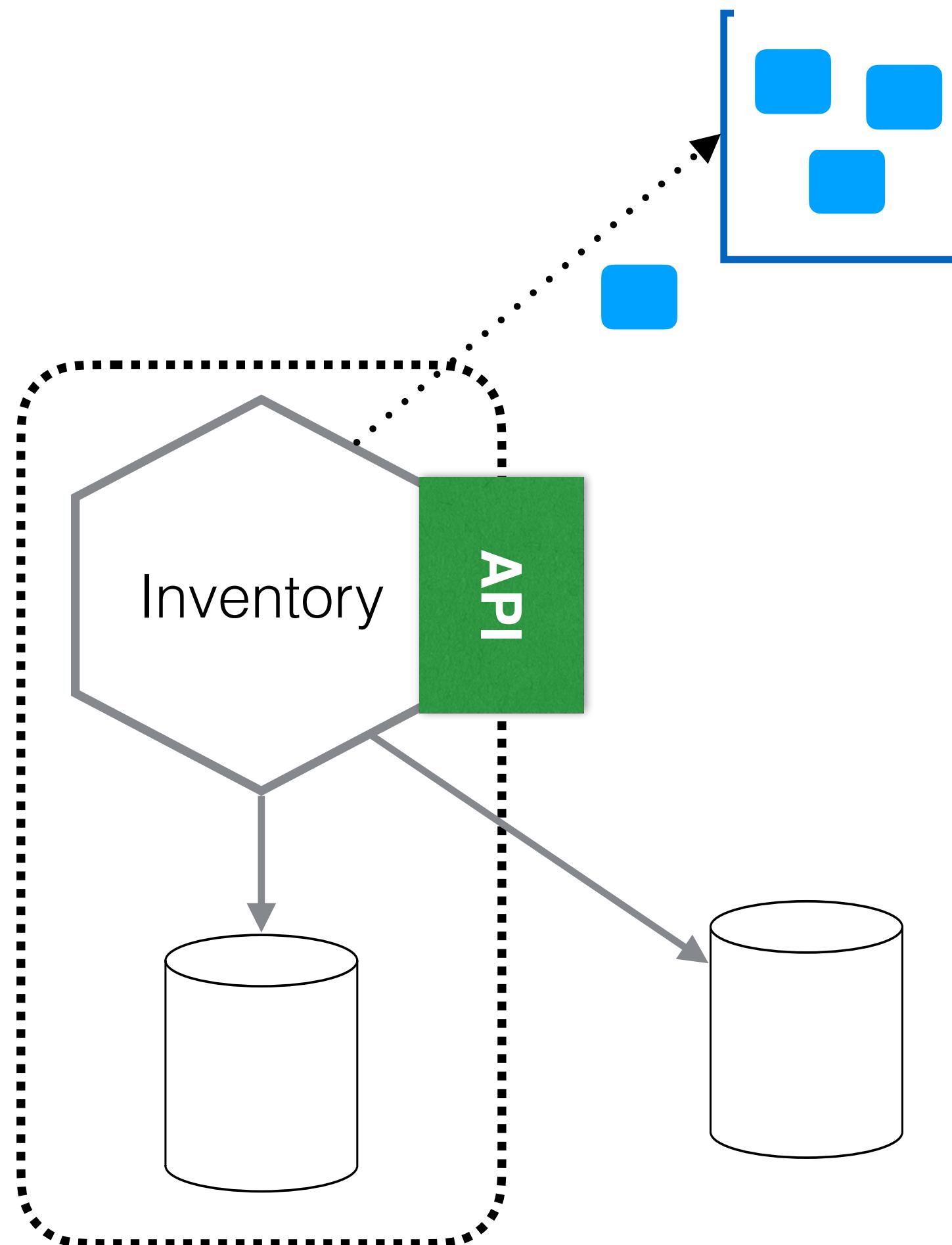


PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

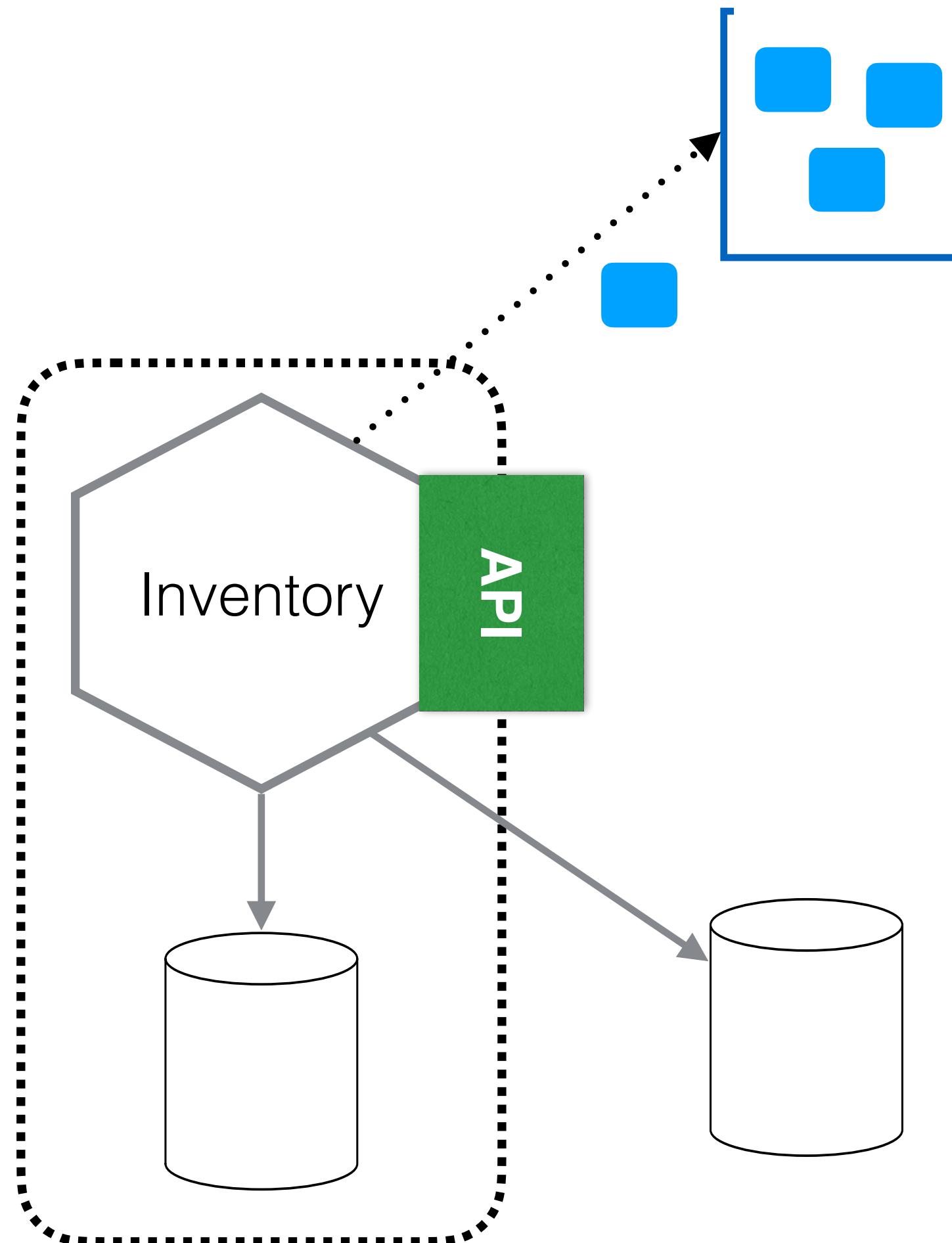
PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

Views are one way to implement this pattern

PATTERN: DATABASE AS AN ENDPOINT

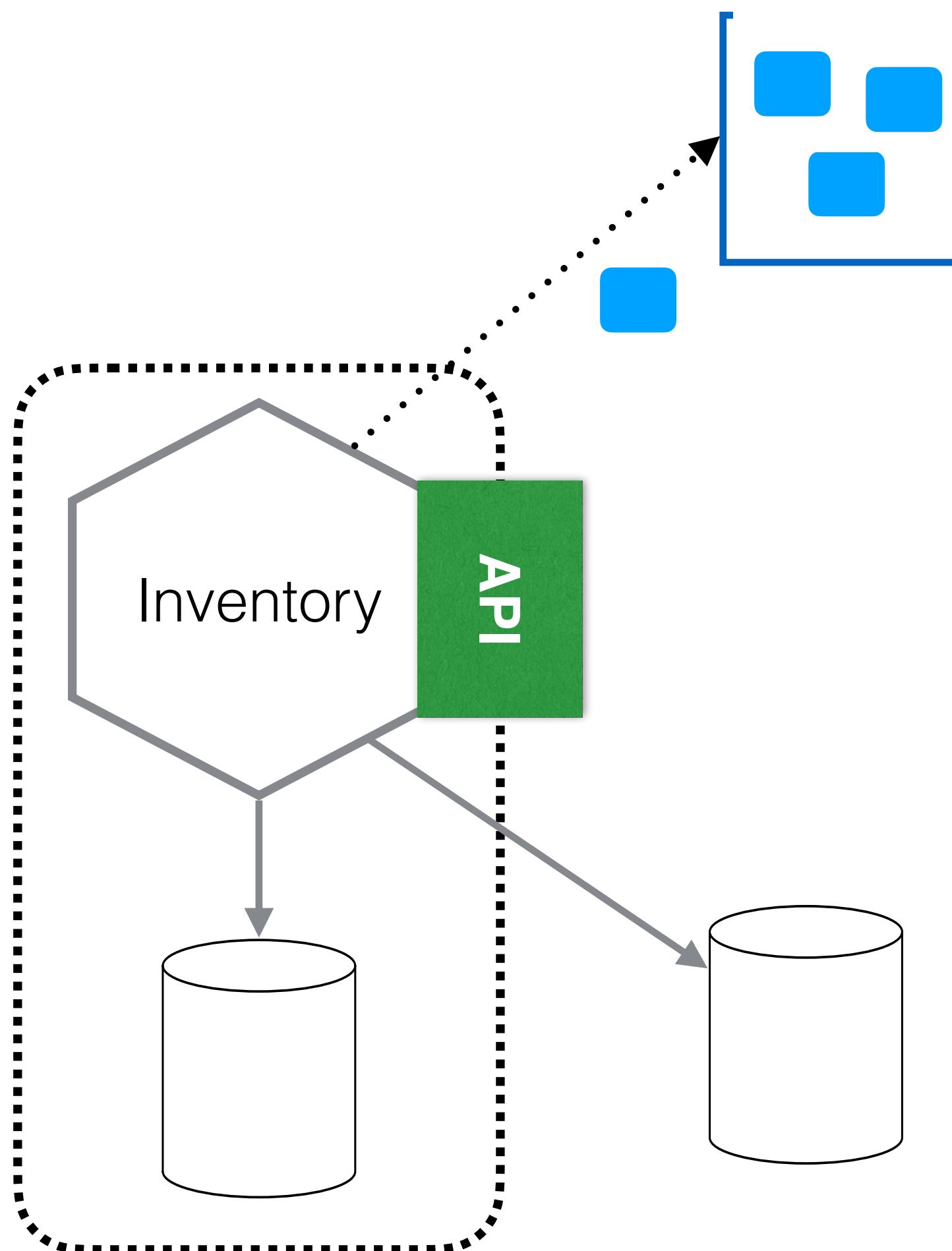


Expose a full DB as an endpoint

Views are one way to implement this pattern

Typically read-only

PATTERN: DATABASE AS AN ENDPOINT



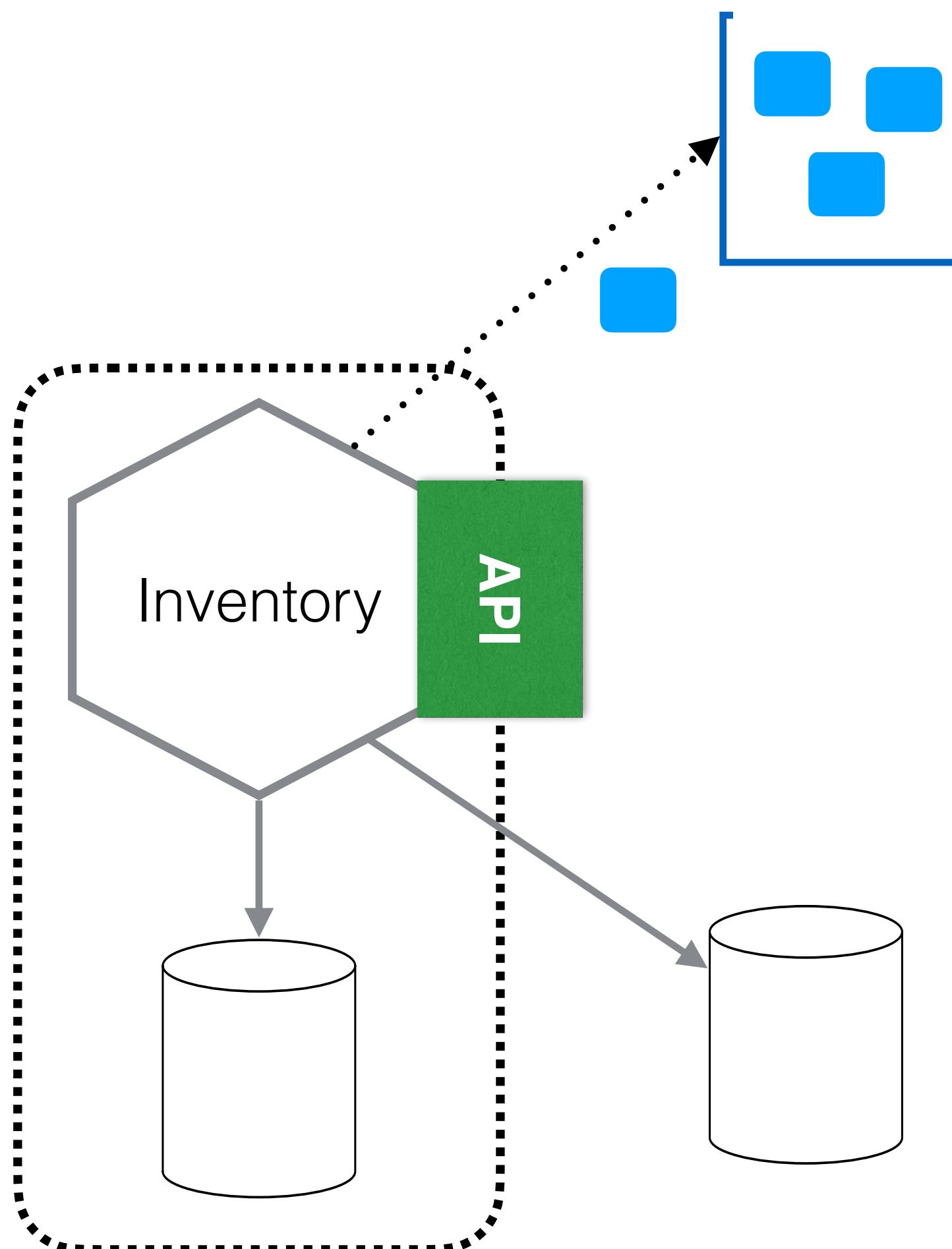
Expose a full DB as an endpoint

Views are one way to implement this pattern

Typically read-only

Really useful for legacy applications

PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

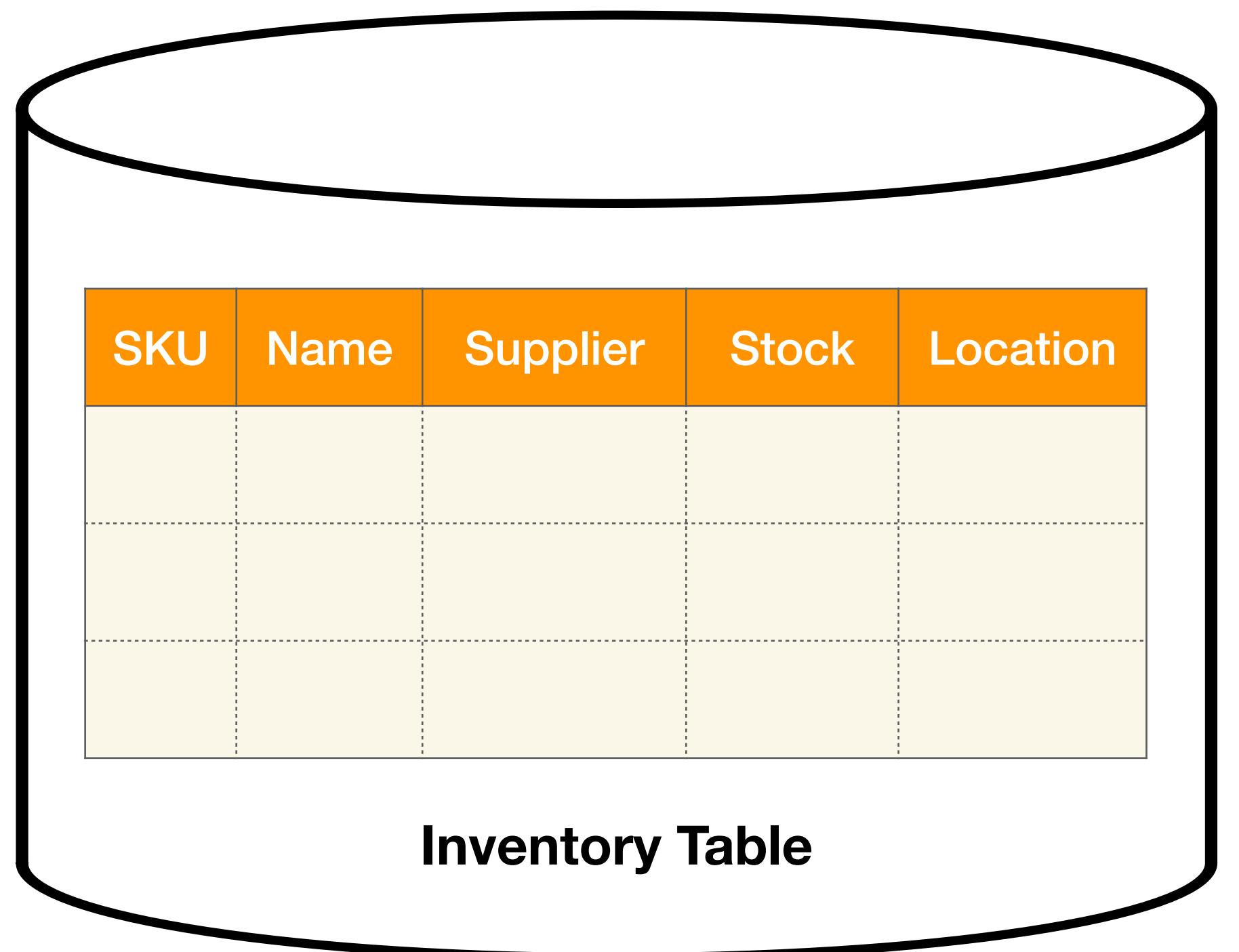
Views are one way to implement this pattern

Typically read-only

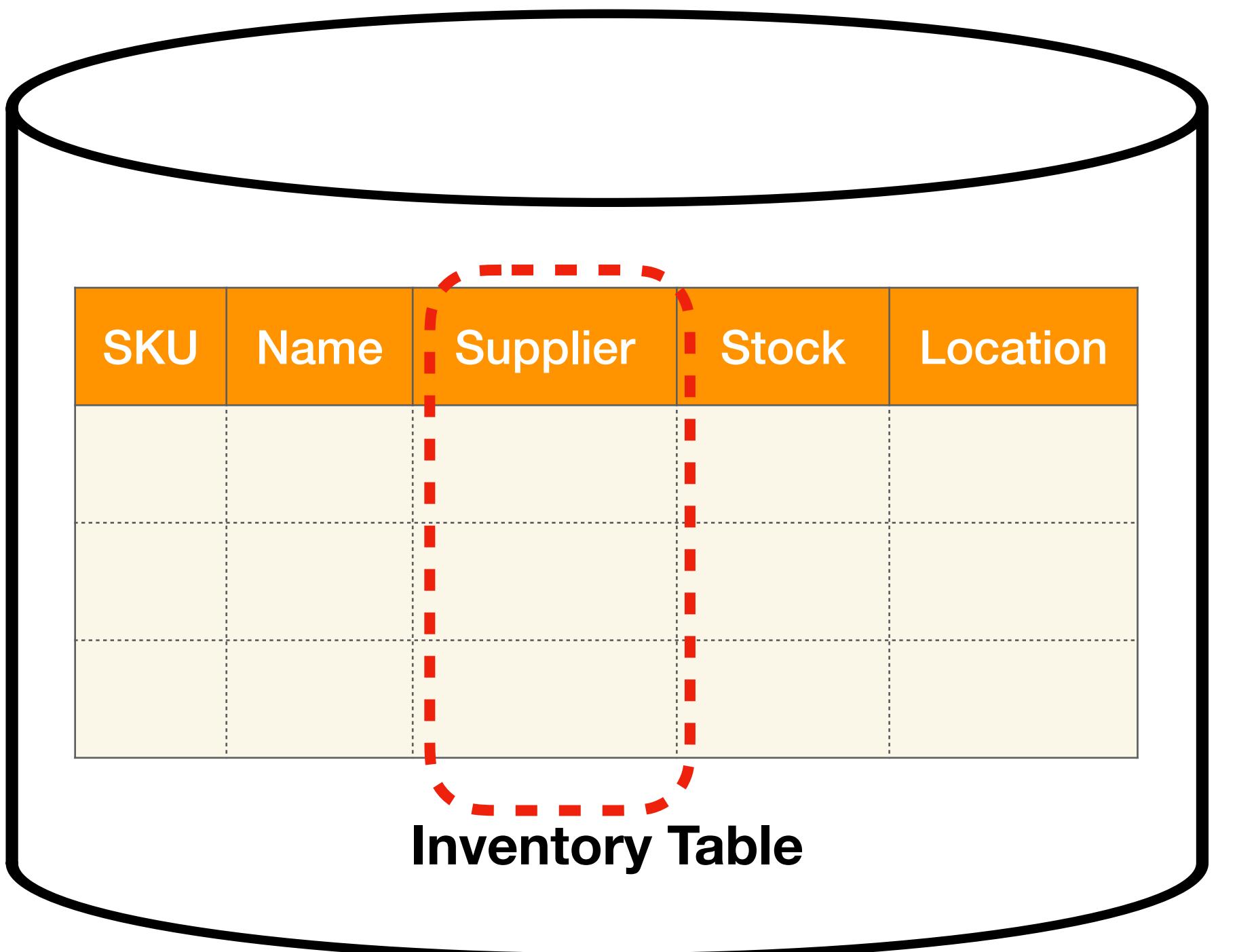
Really useful for legacy applications

Can allow for expensive, ad-hoc queries
more easily (e.g. joins)

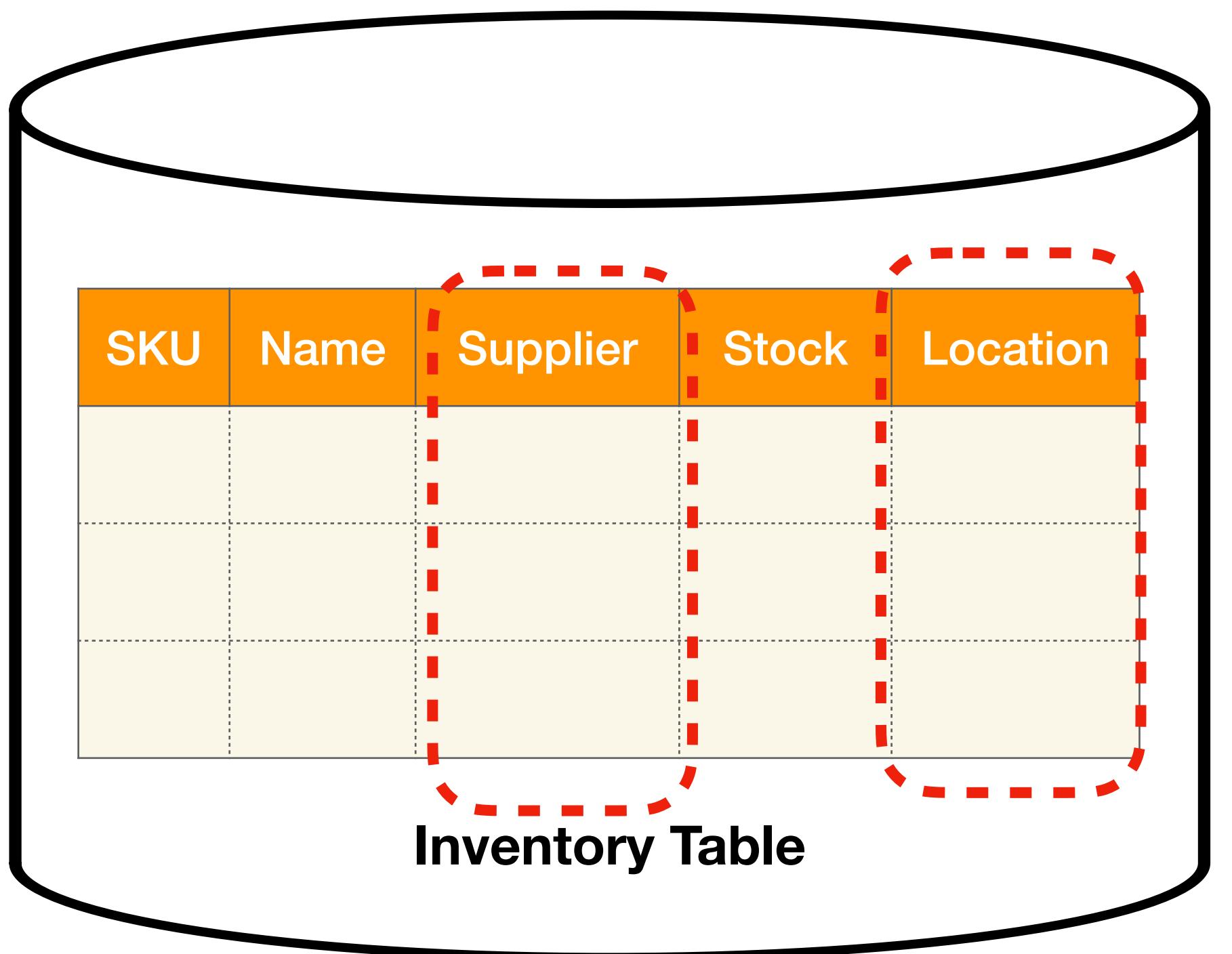
PATTERN: DATABASE VIEWS



PATTERN: DATABASE VIEWS

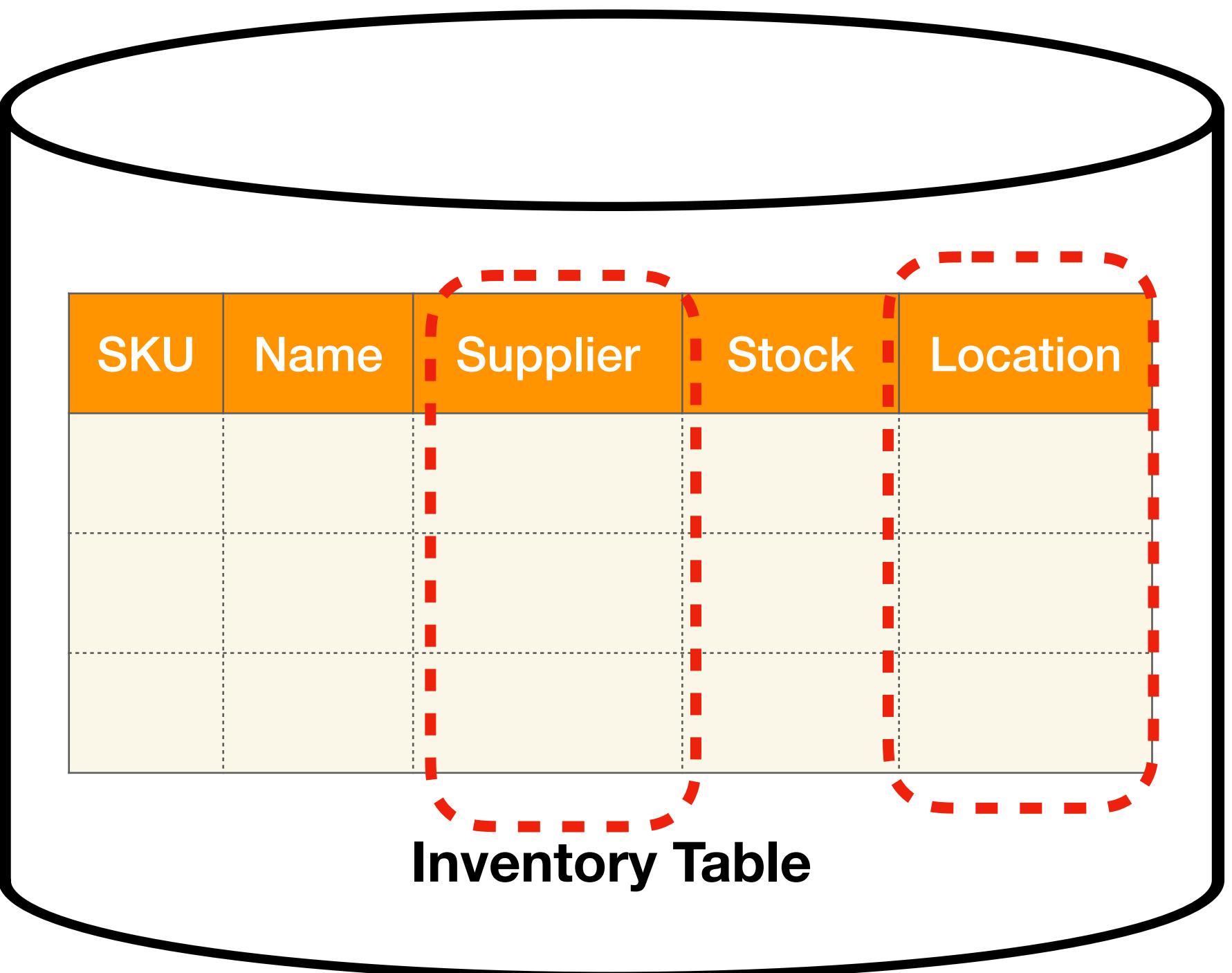


PATTERN: DATABASE VIEWS



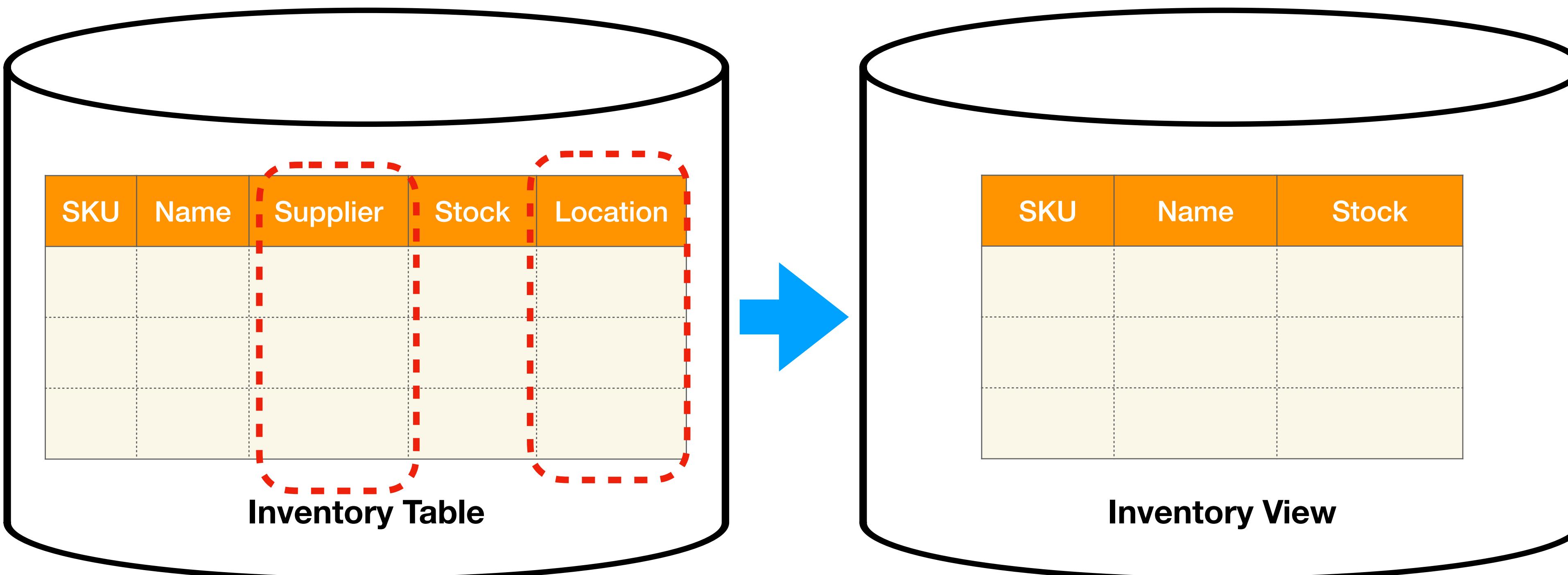
PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view



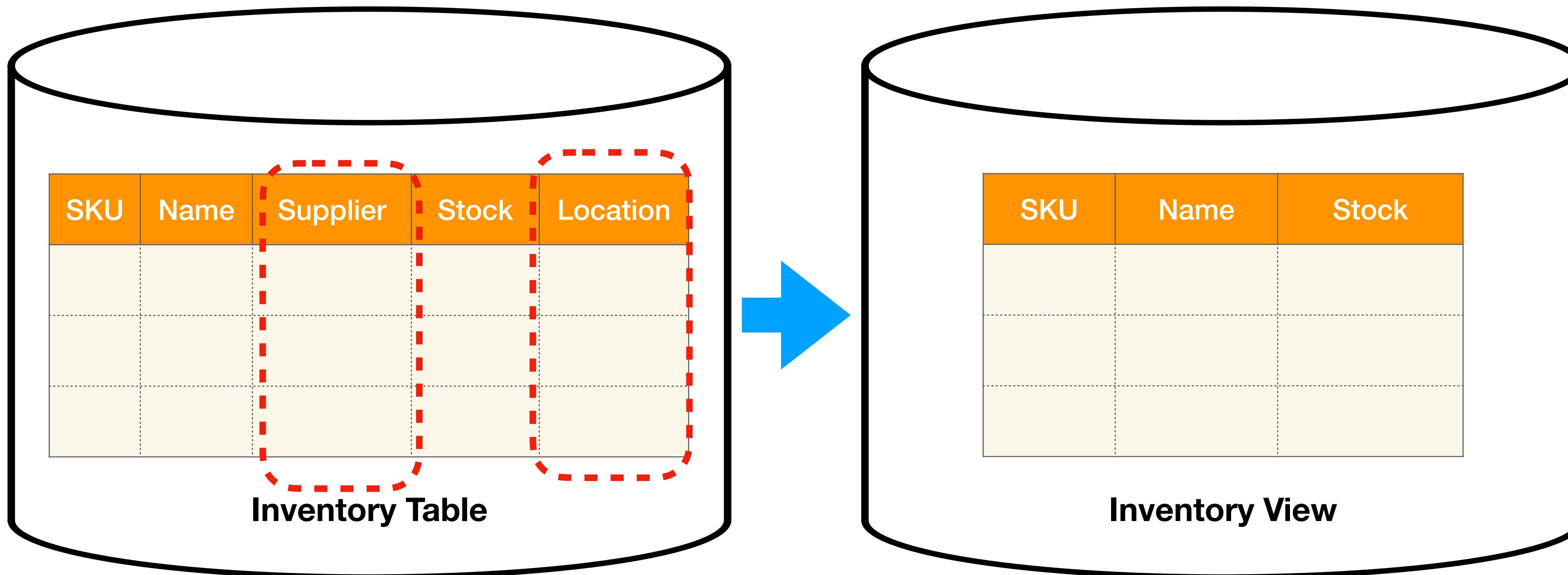
PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view



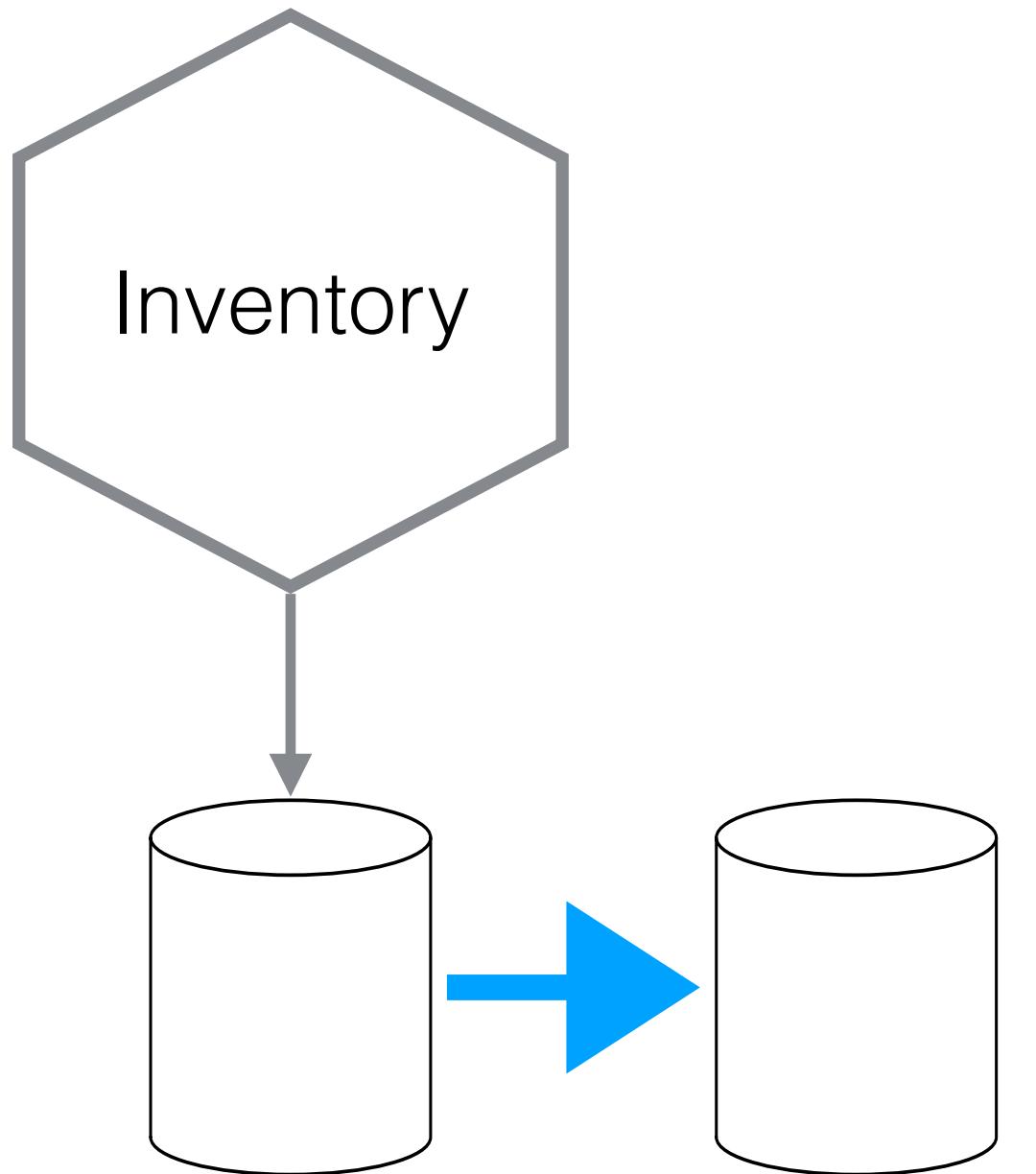
PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view

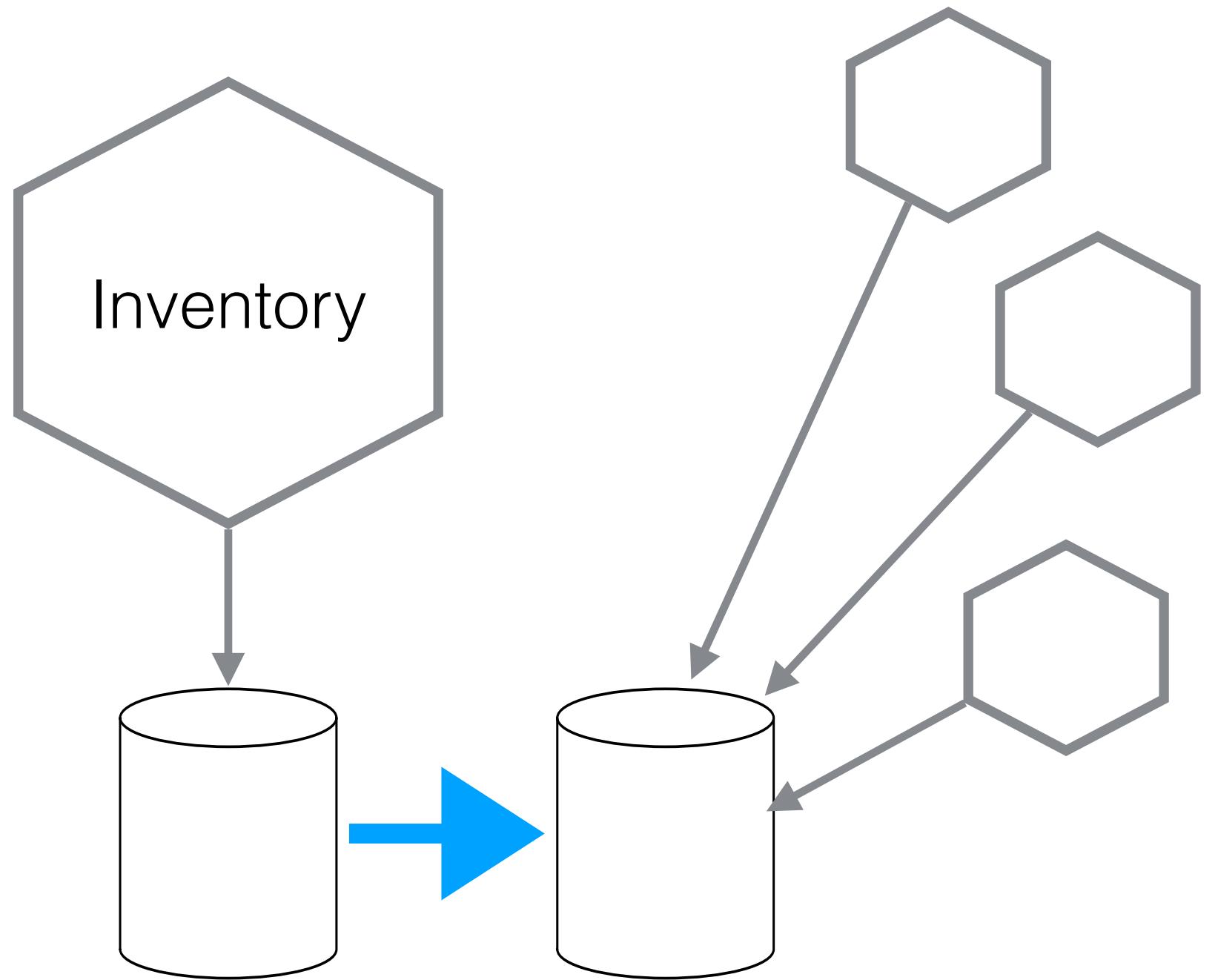


Allows for limited information hiding for direct DB access

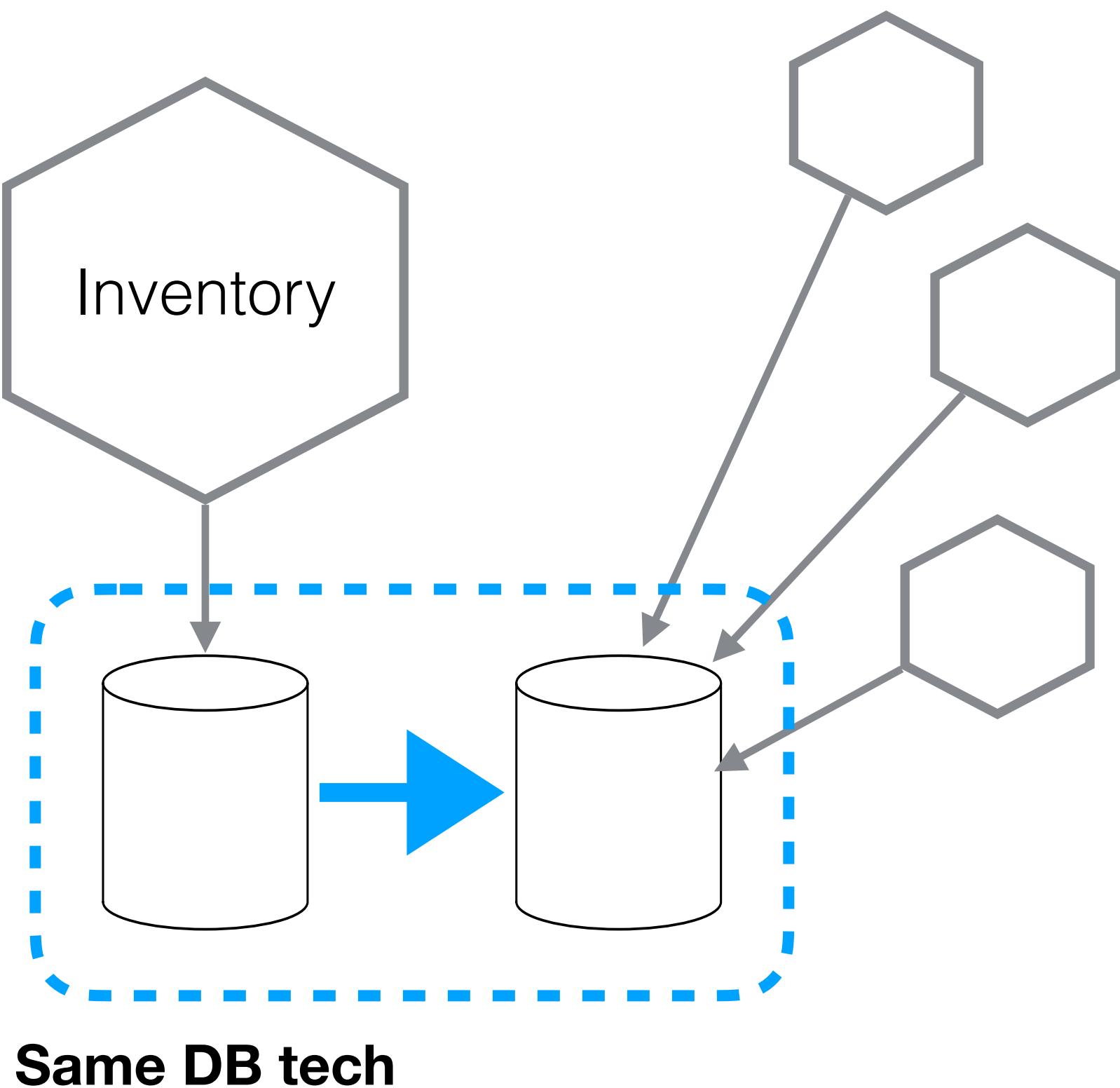
DATABASE VIEW - PROS AND CONS



DATABASE VIEW - PROS AND CONS



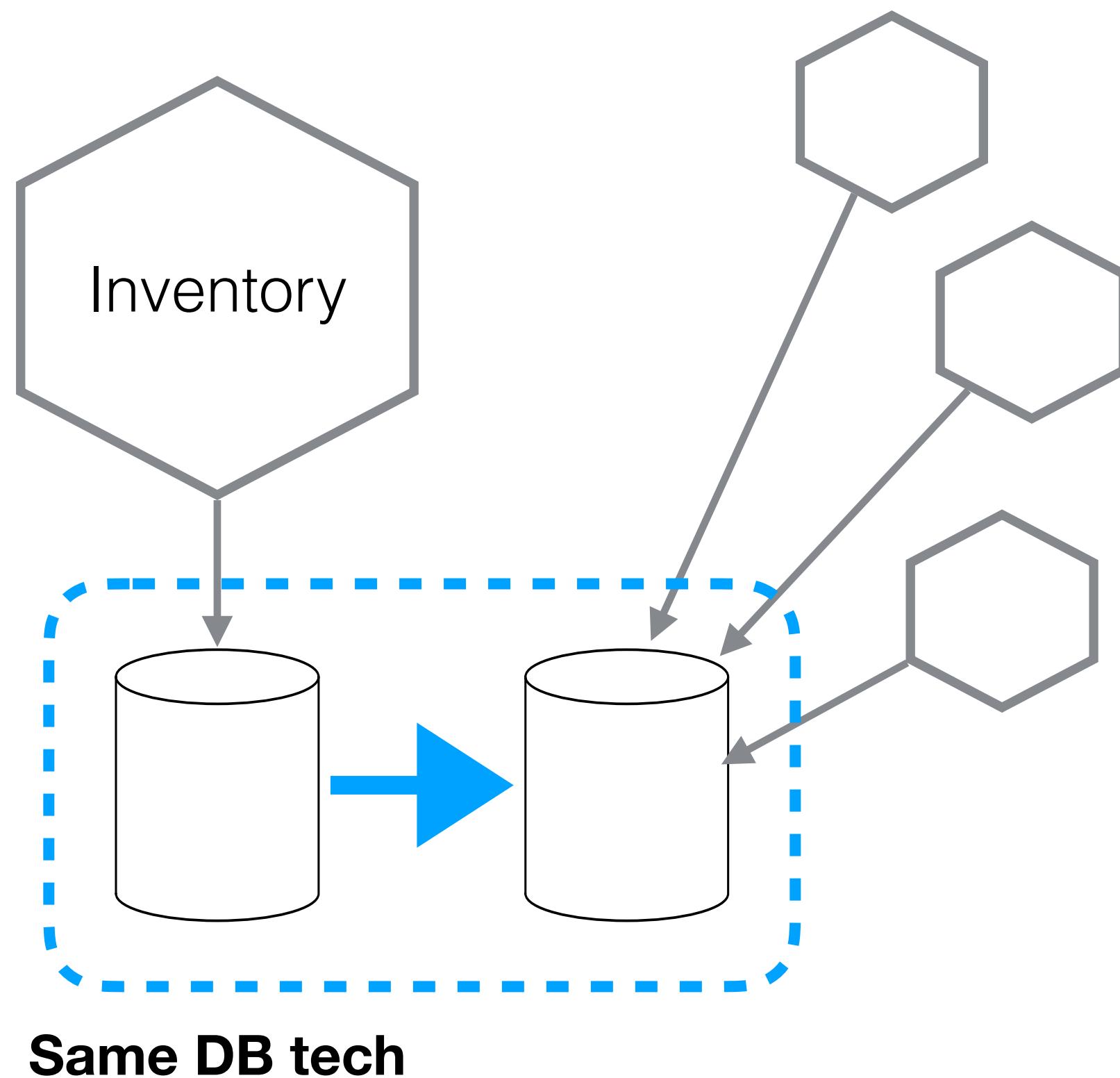
DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation

Same DB tech

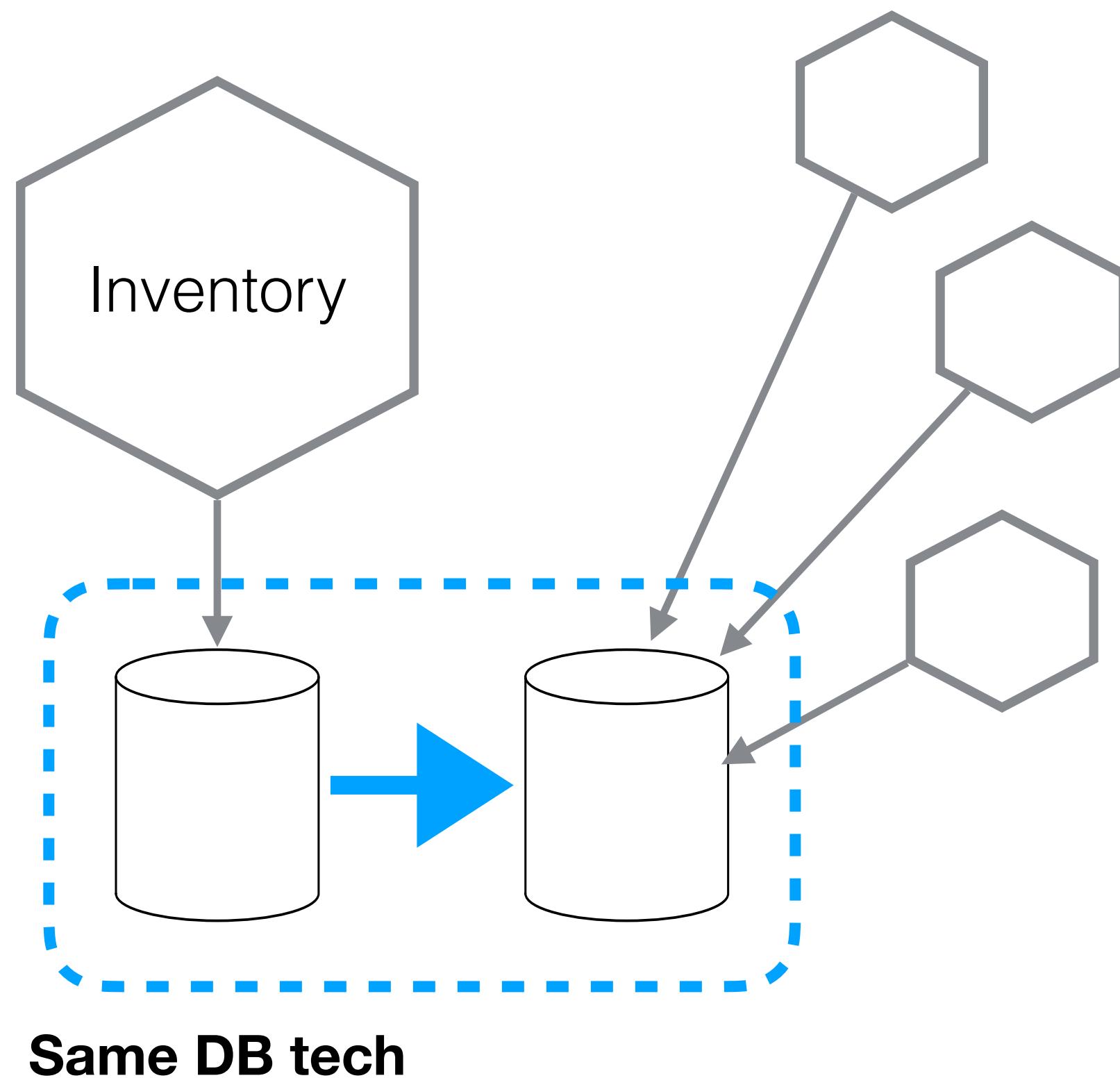
DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation

Doesn't solve writes

DATABASE VIEW - PROS AND CONS

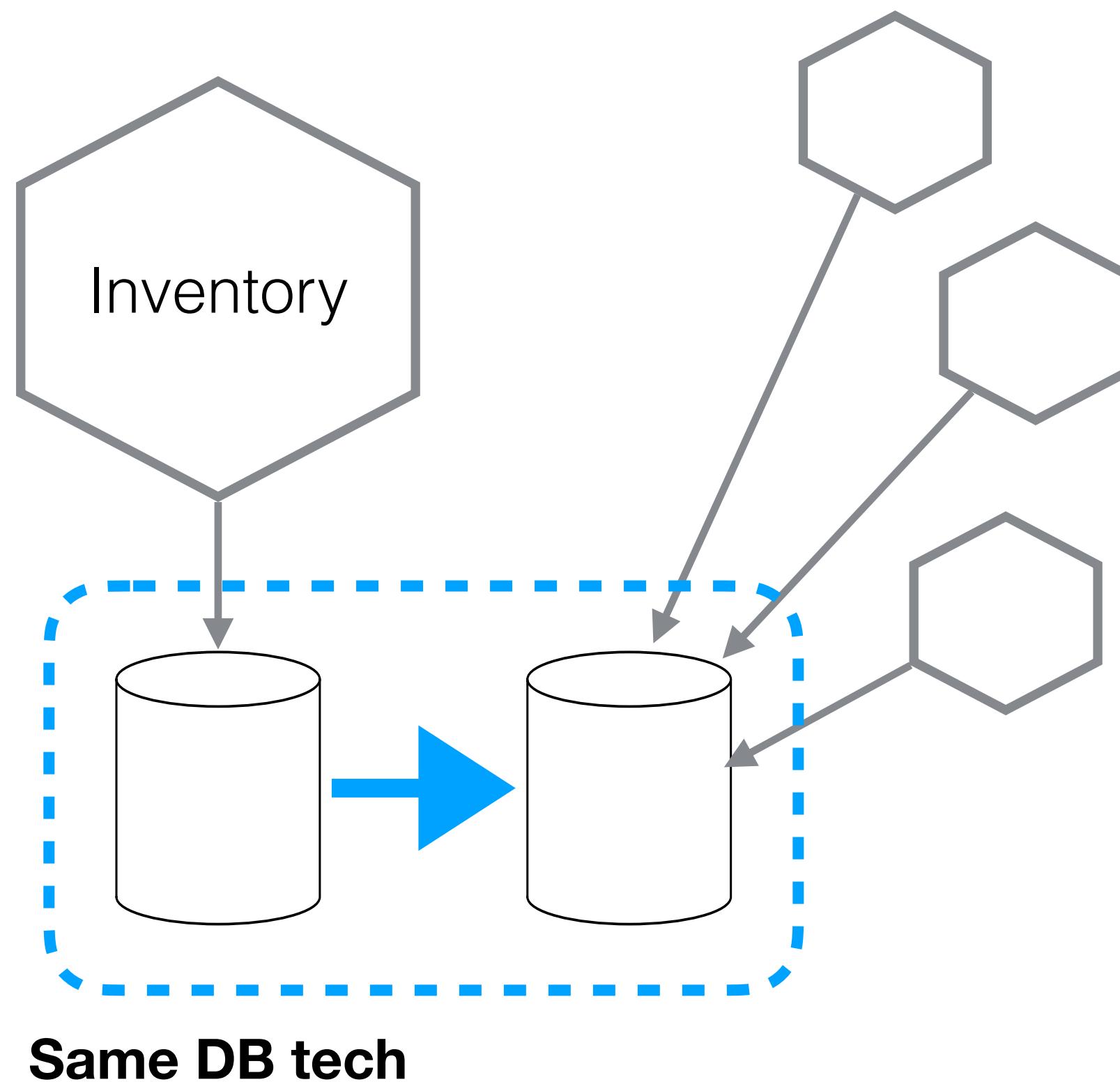


Tied to an underlying implementation

Doesn't solve writes

View mapping needs to be maintained if the source DB changes

DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation

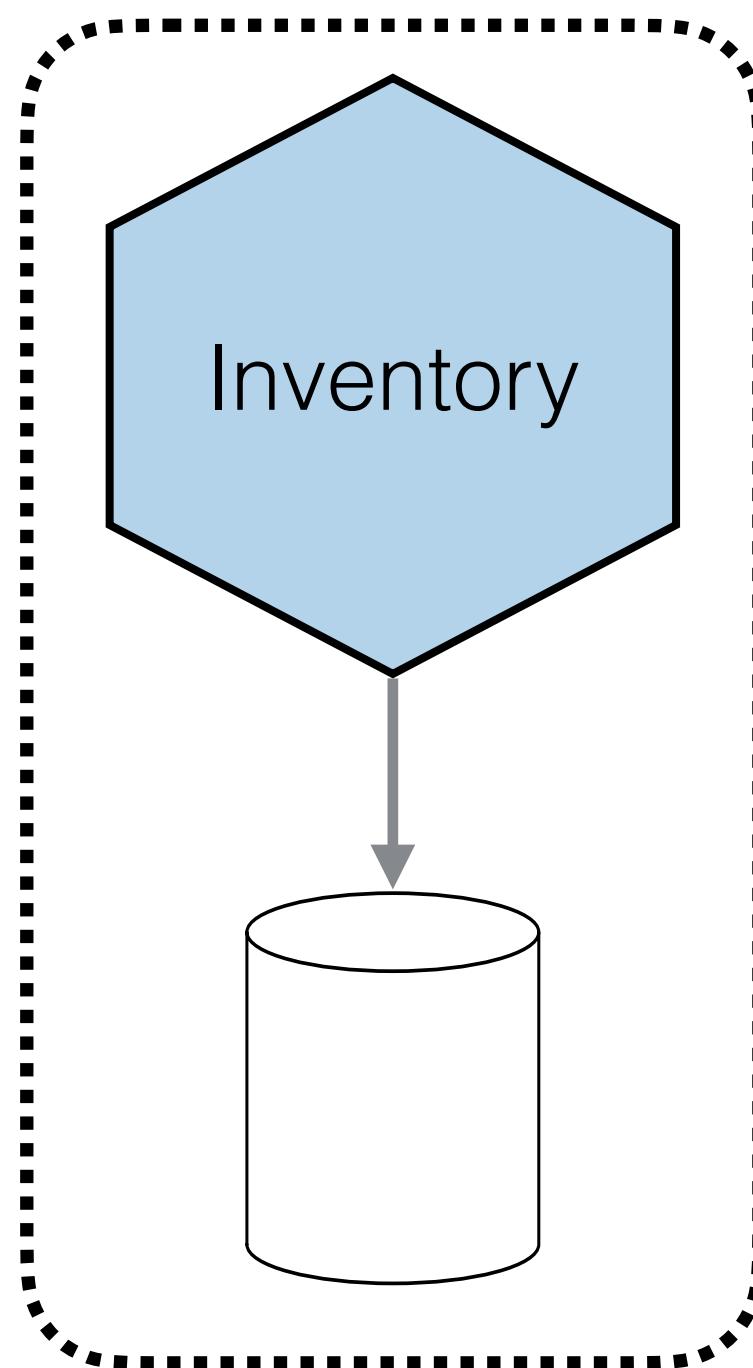
Doesn't solve writes

View mapping needs to be maintained if the source DB changes

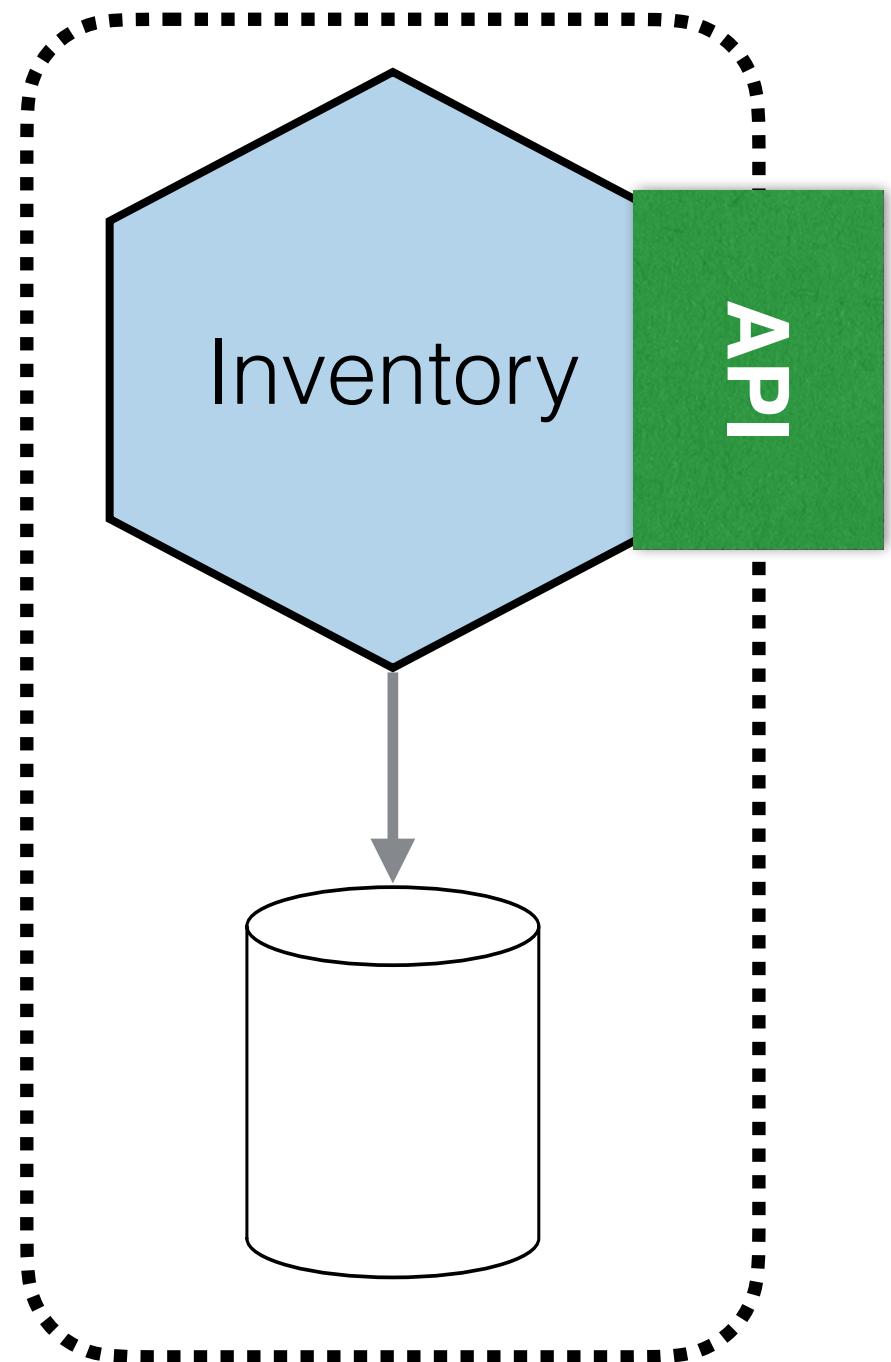
Better than direct DB access

What if I need direct DB access?

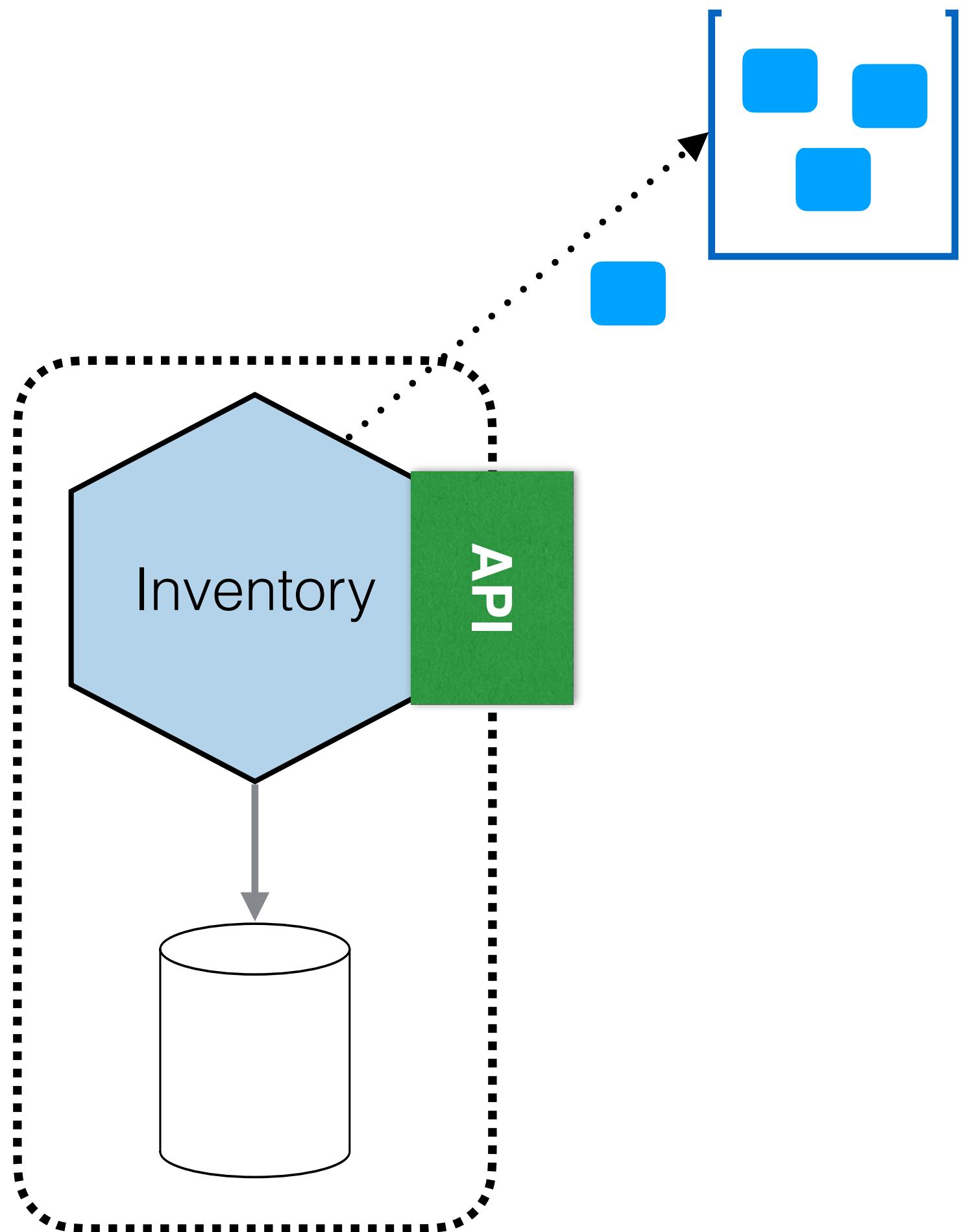
PATTERN: DATABASE AS AN ENDPOINT



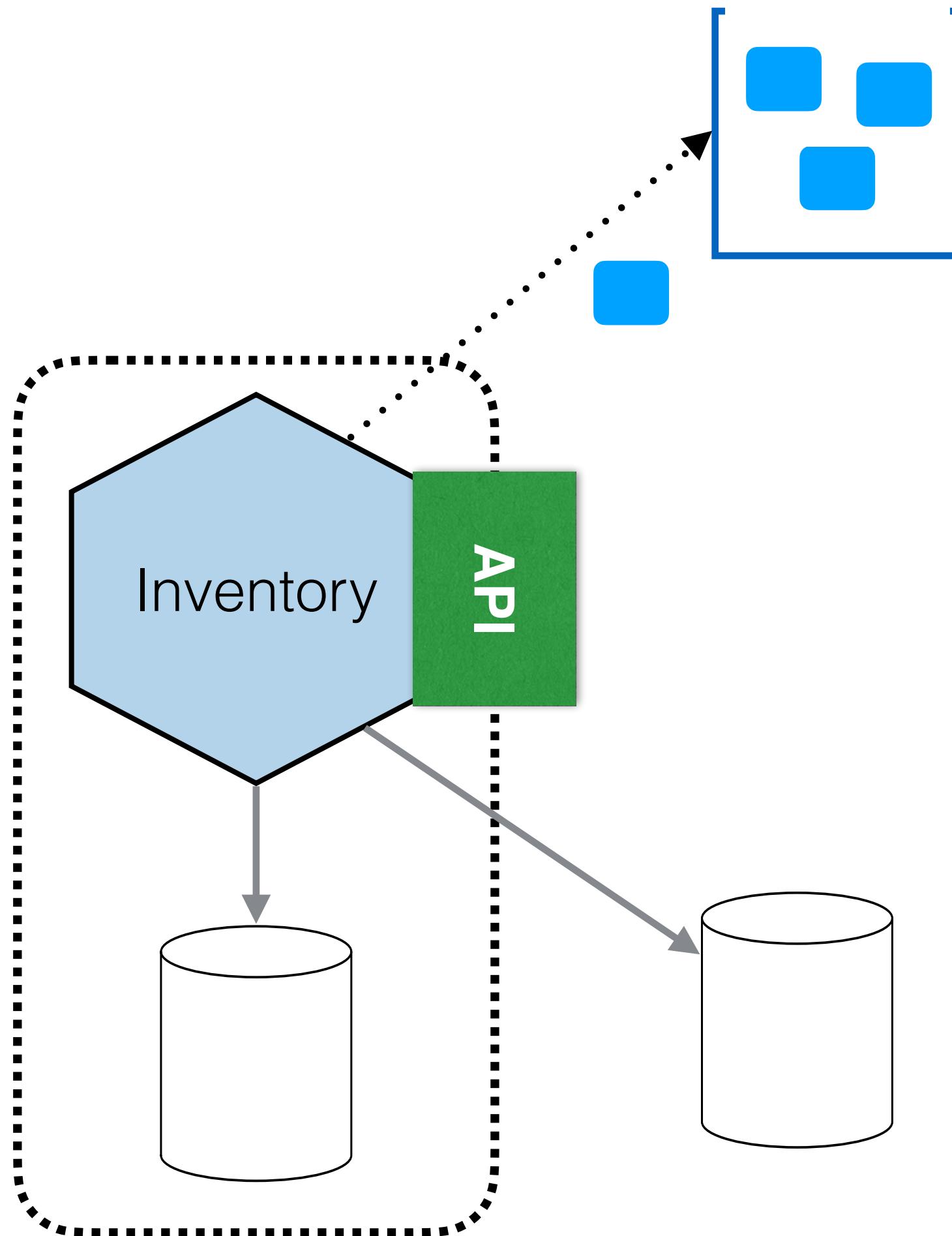
PATTERN: DATABASE AS AN ENDPOINT



PATTERN: DATABASE AS AN ENDPOINT

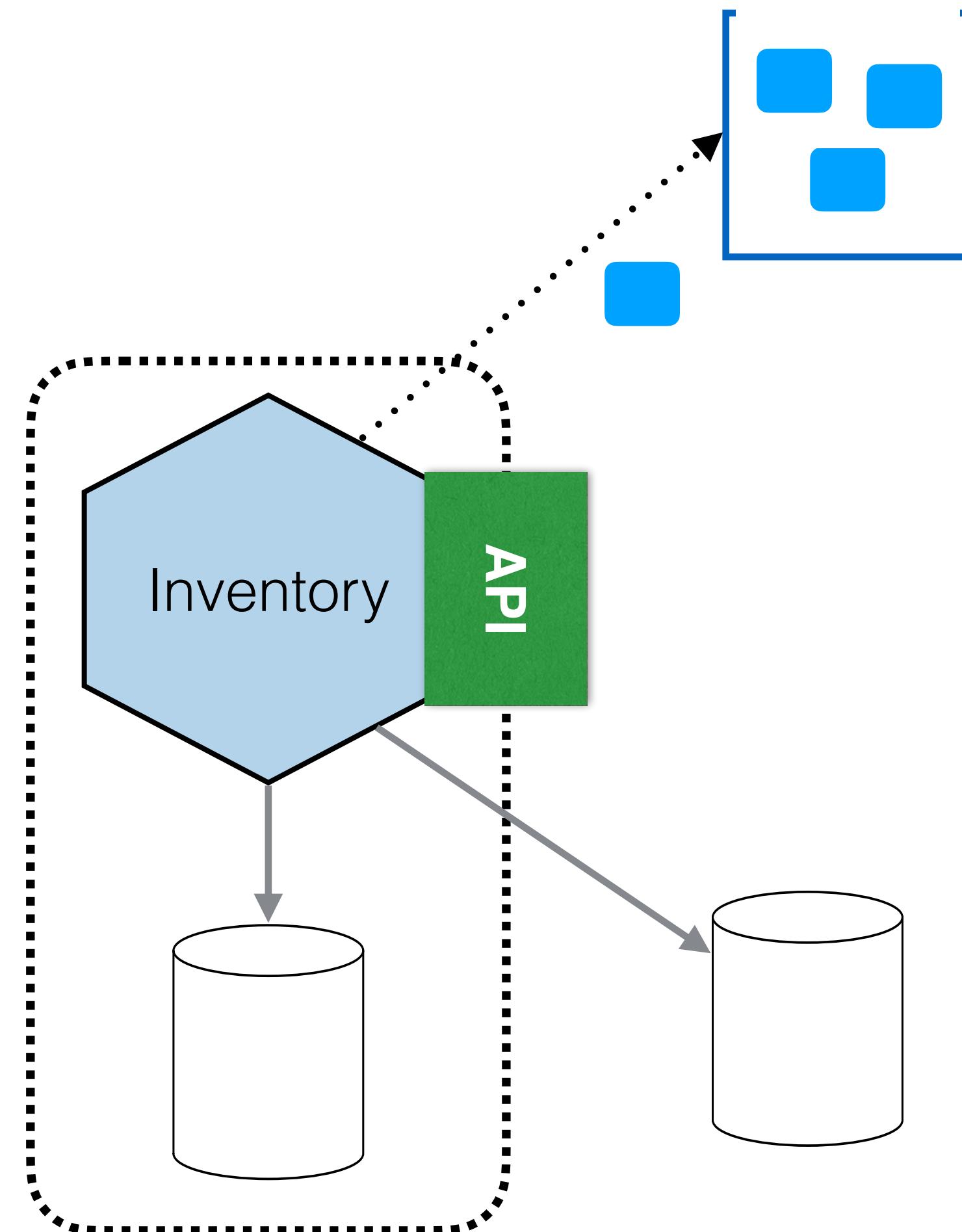


PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

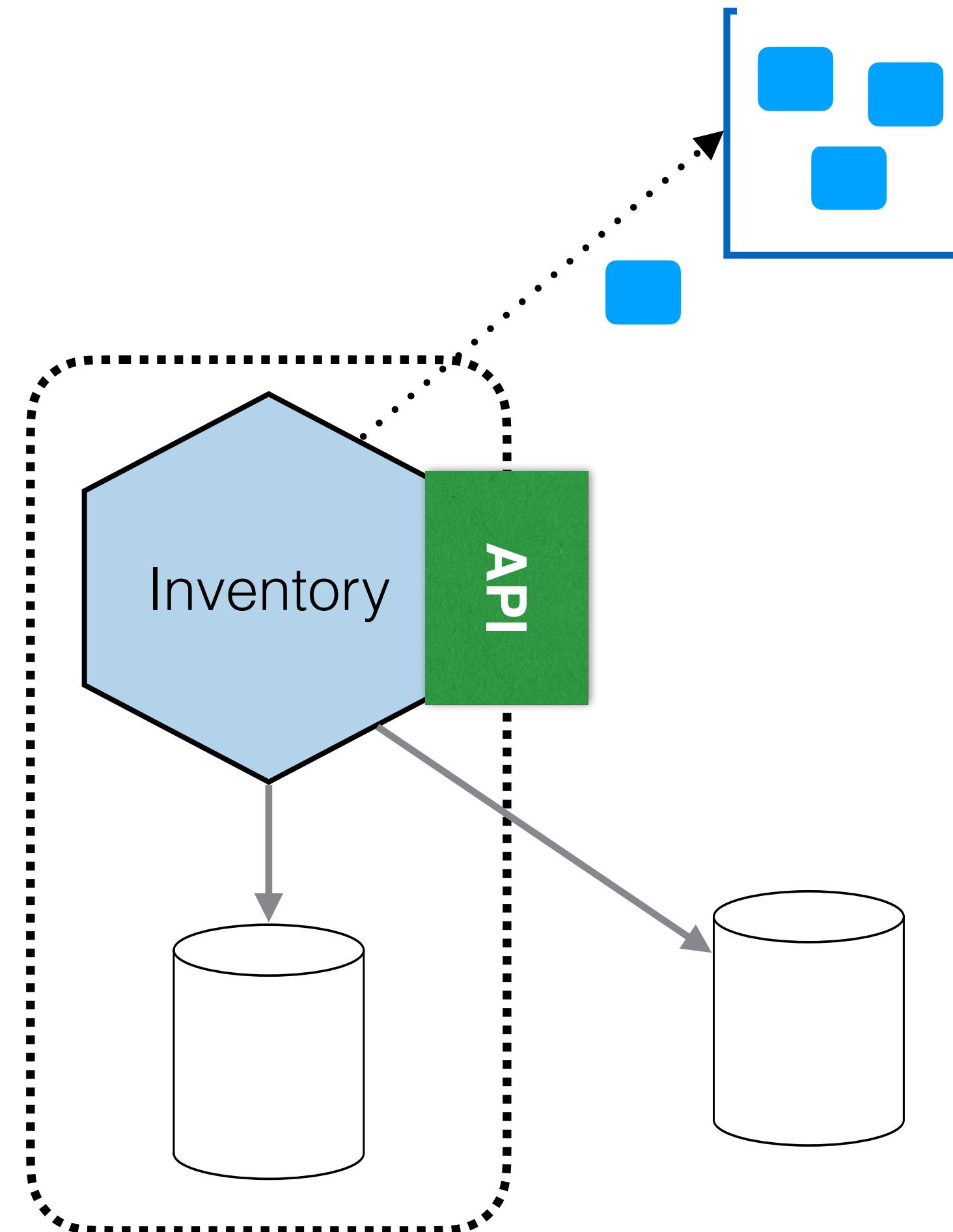
PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

Views are one way to implement this pattern

PATTERN: DATABASE AS AN ENDPOINT

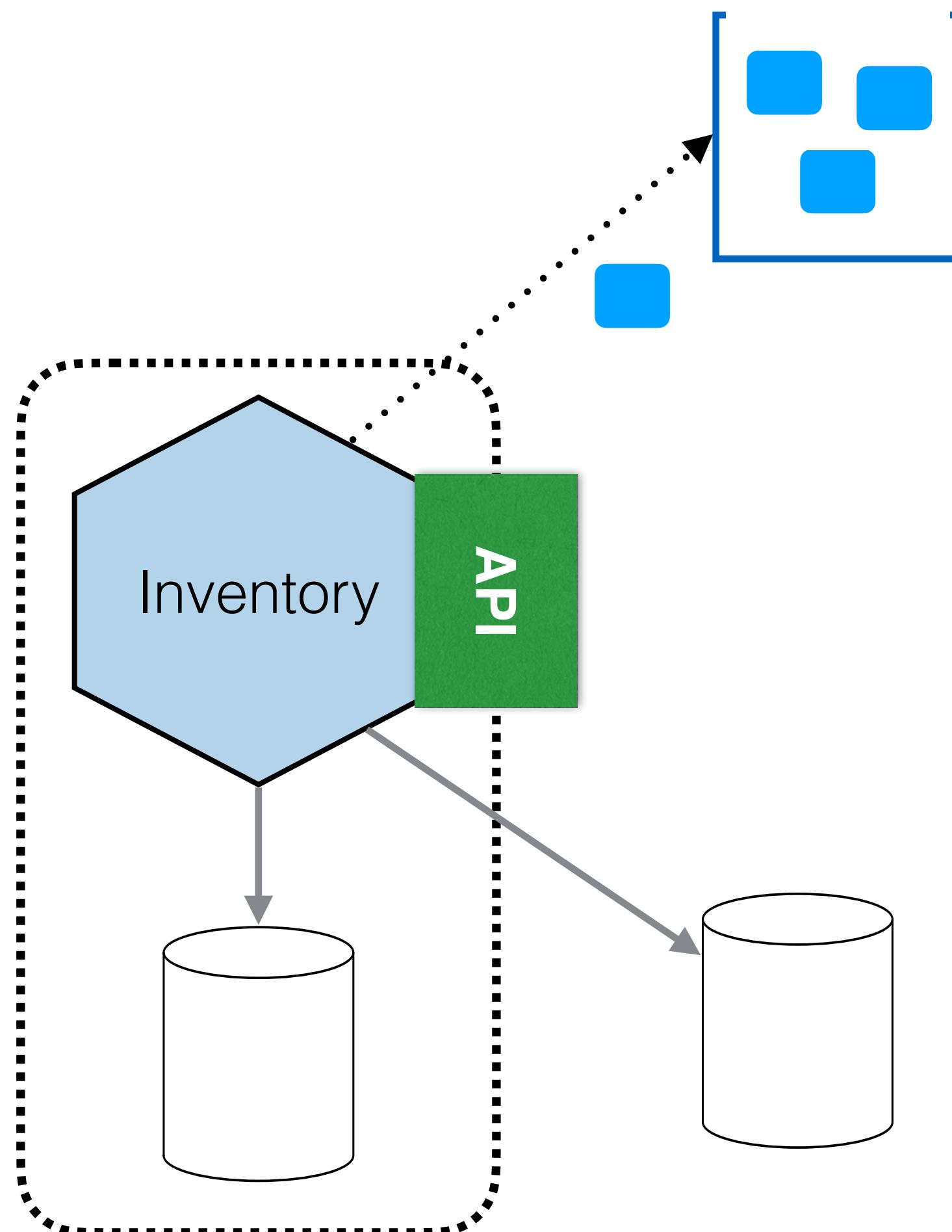


Expose a full DB as an endpoint

Views are one way to implement this pattern

Read-only

PATTERN: DATABASE AS AN ENDPOINT



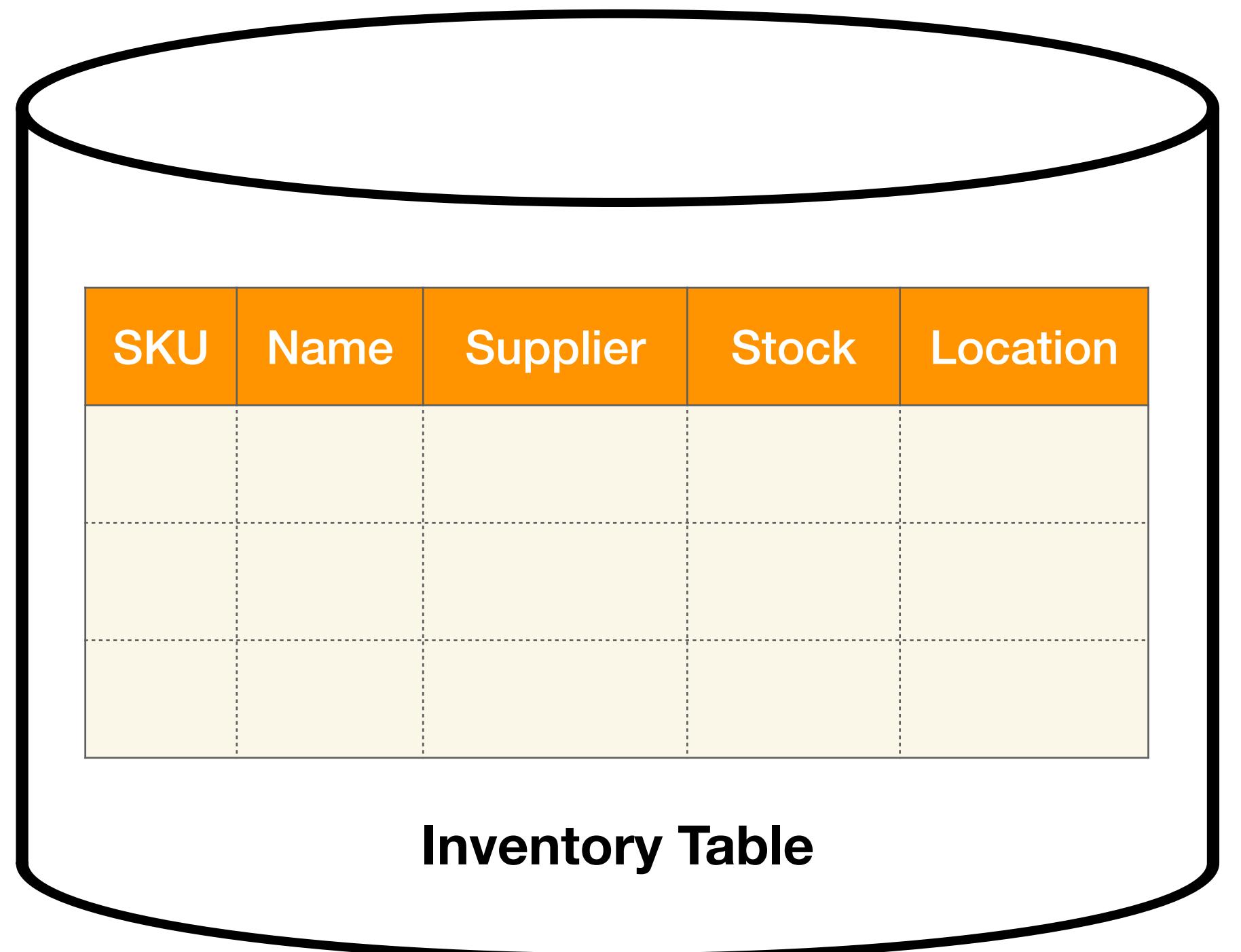
Expose a full DB as an endpoint

Views are one way to implement this pattern

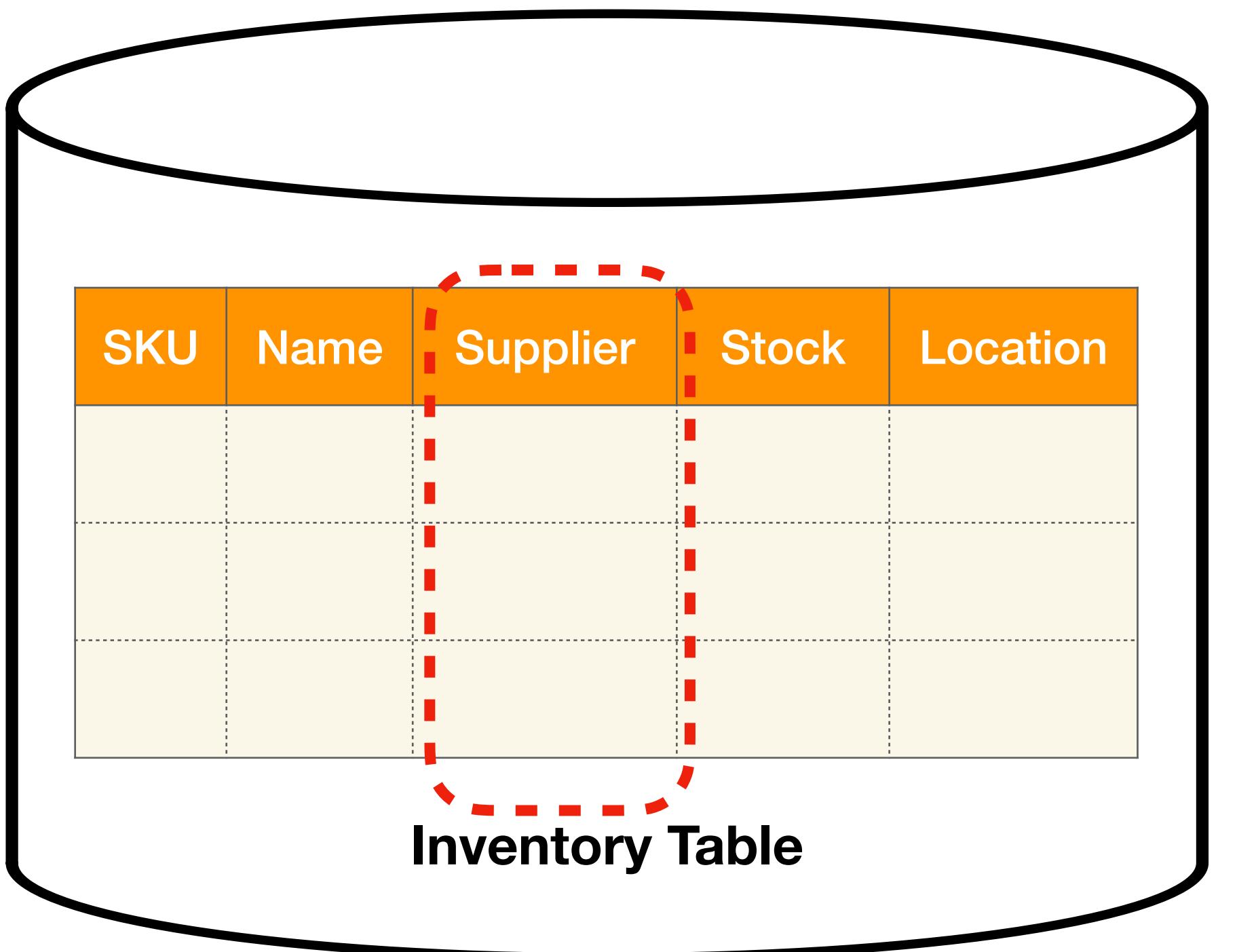
Read-only

Can allow for expensive, ad-hoc queries more easily (e.g. joins)

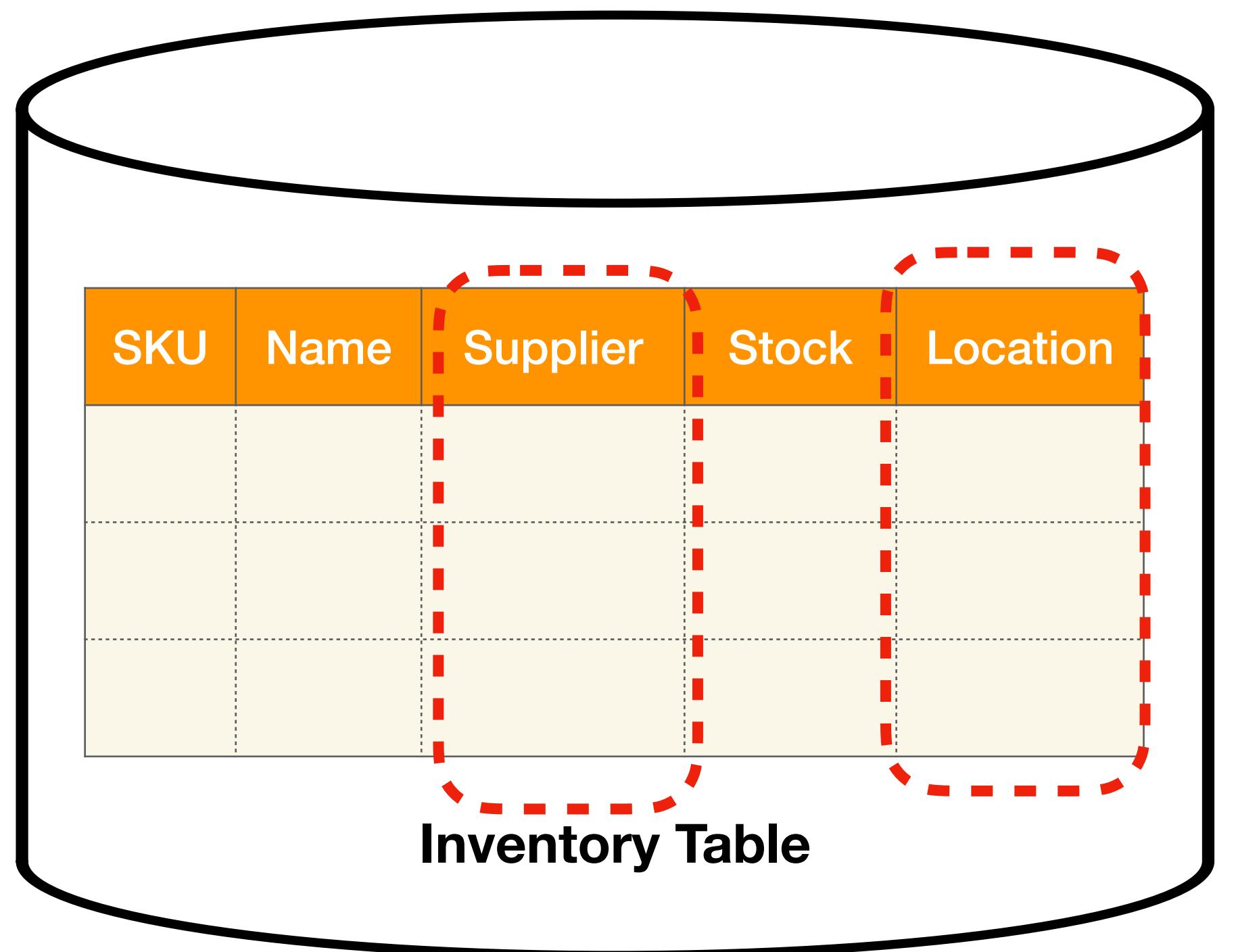
PATTERN: DATABASE VIEWS



PATTERN: DATABASE VIEWS

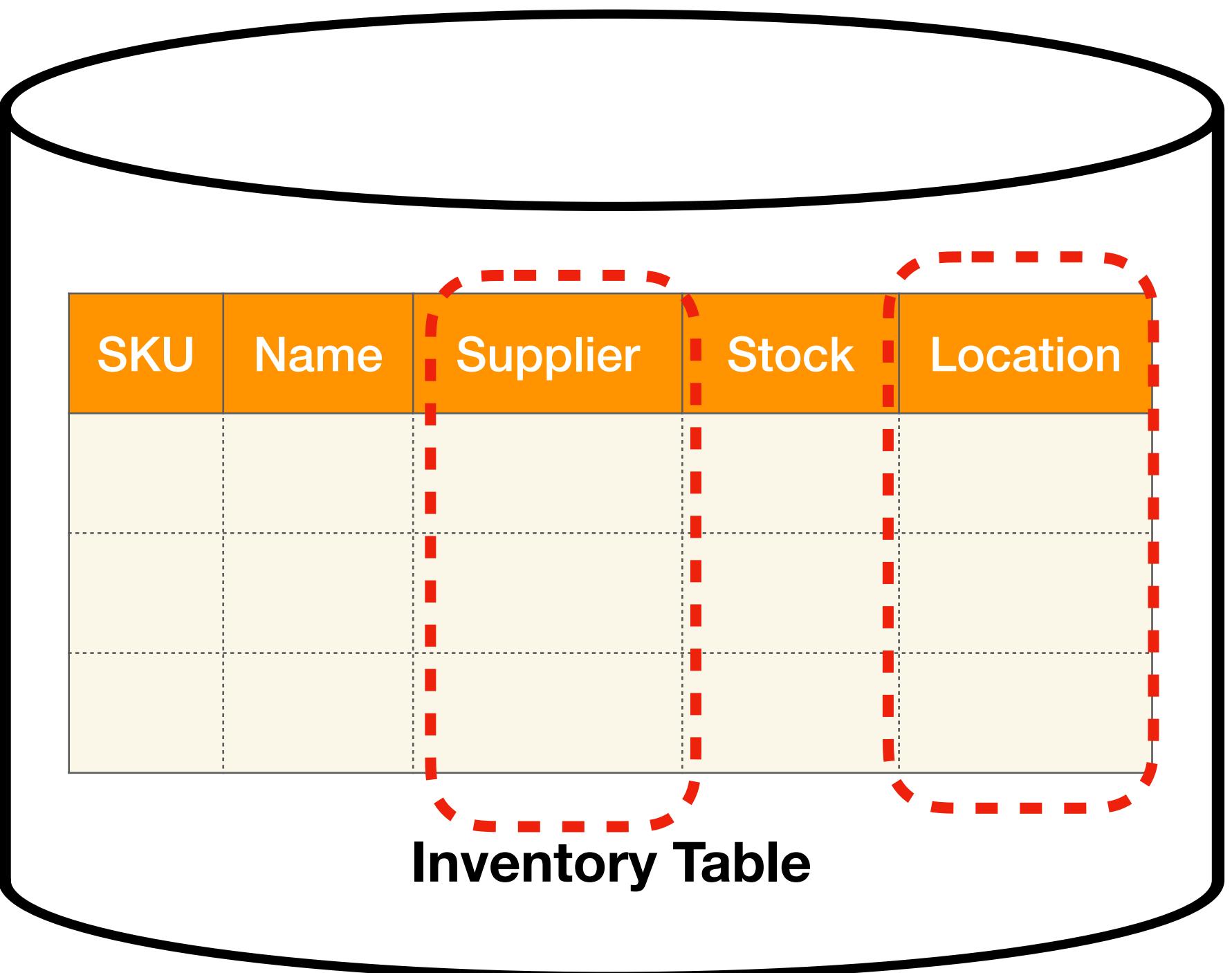


PATTERN: DATABASE VIEWS



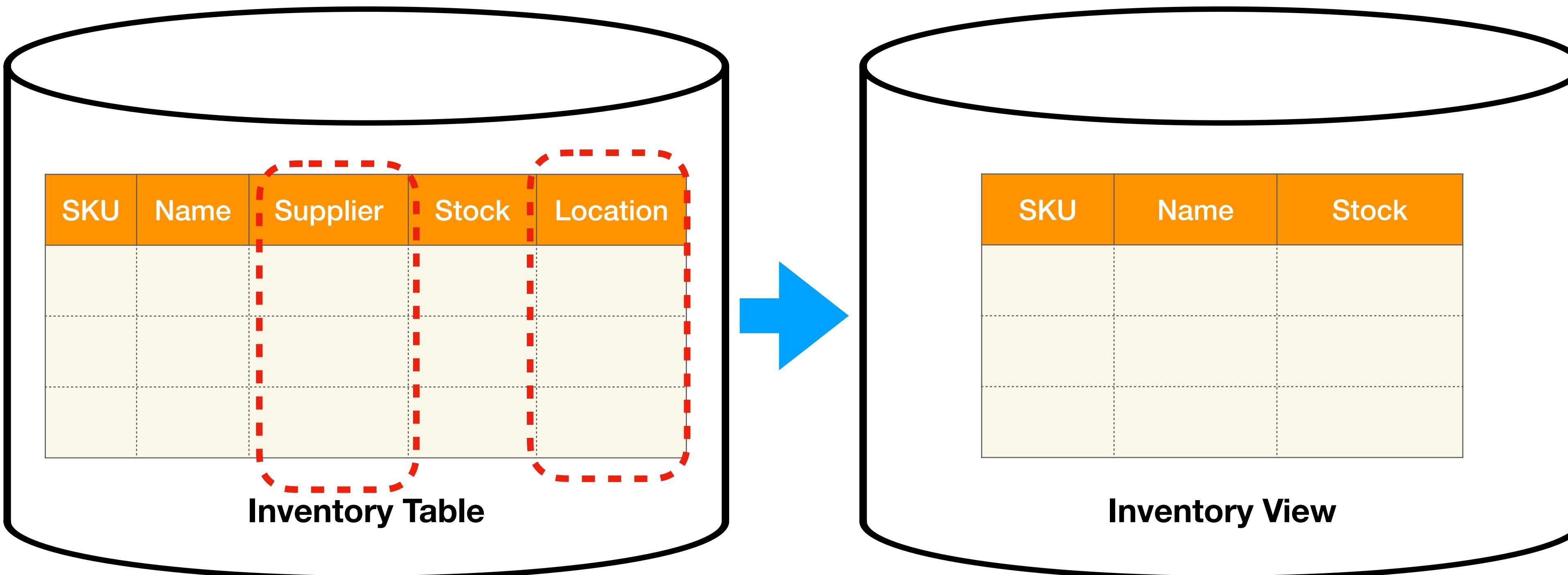
PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view



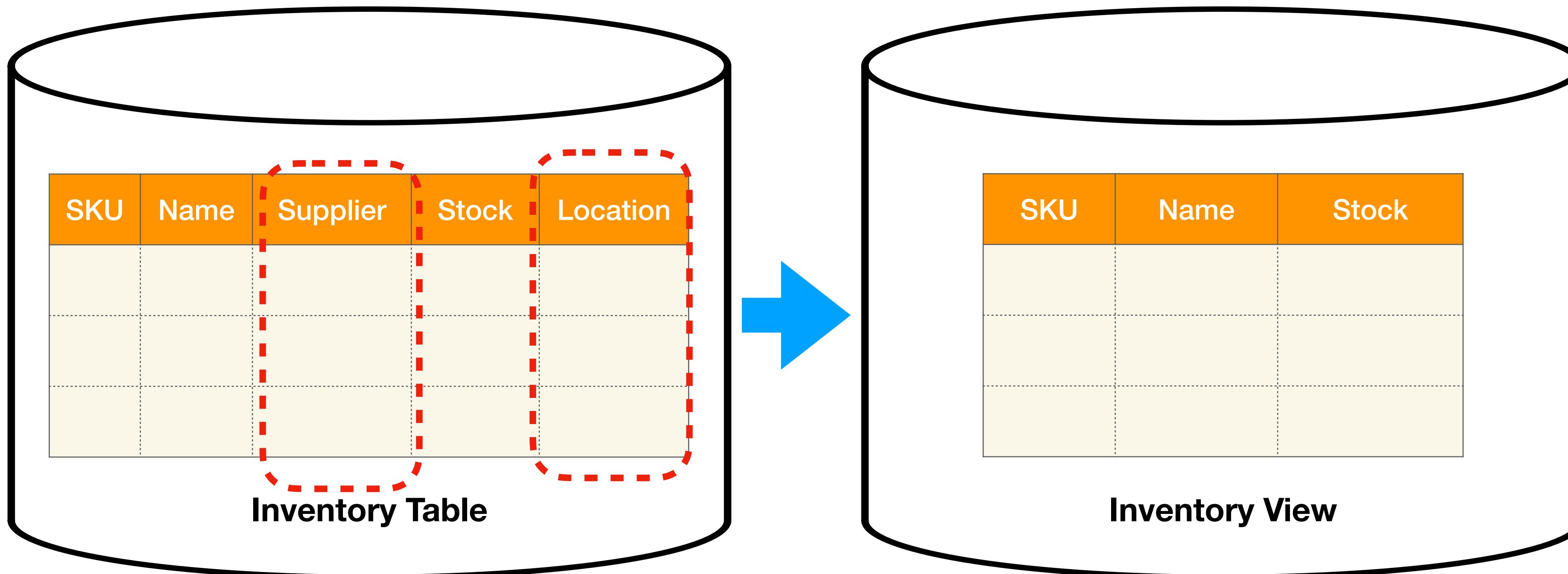
PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view



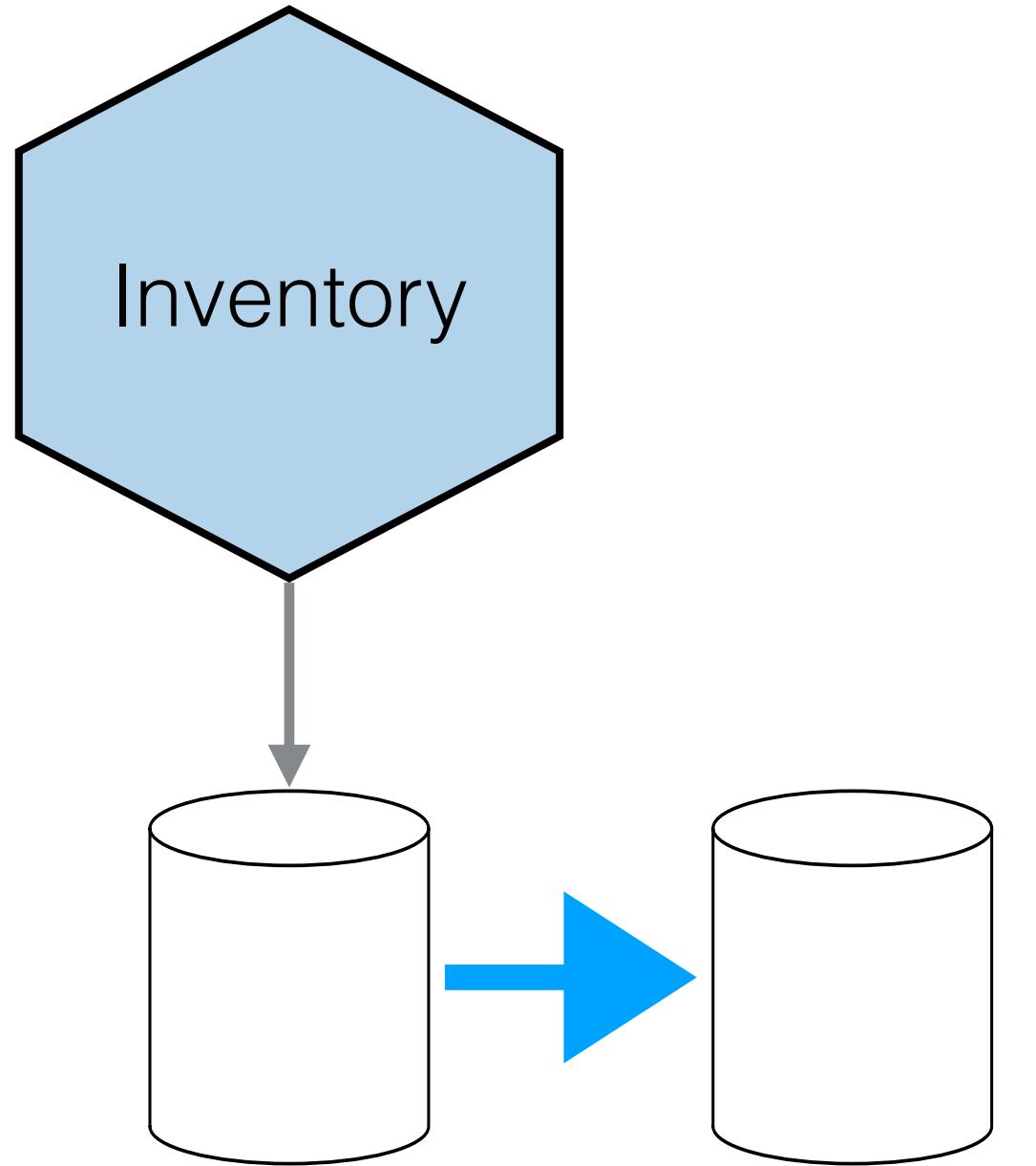
PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view

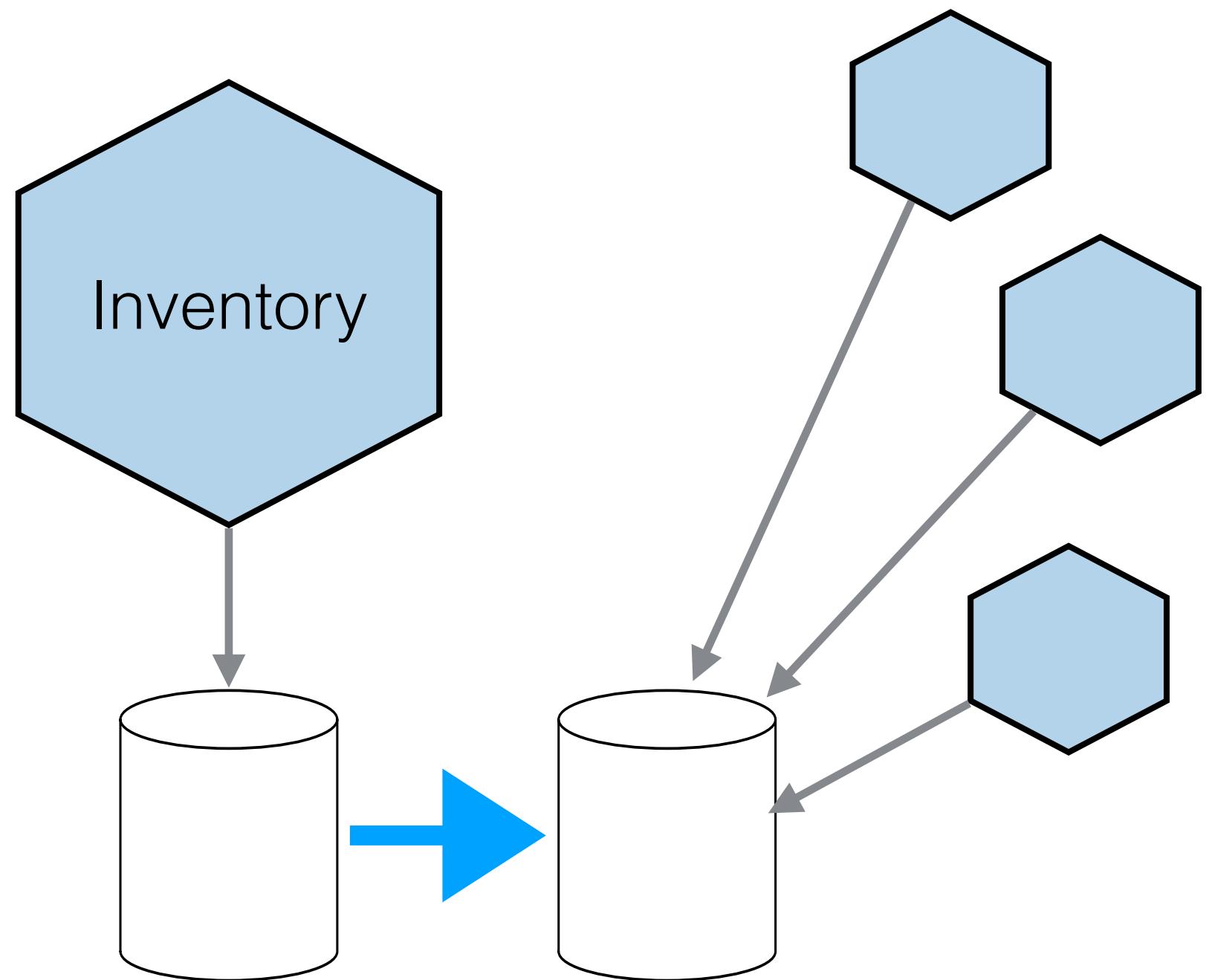


Allows for limited information hiding for direct DB access

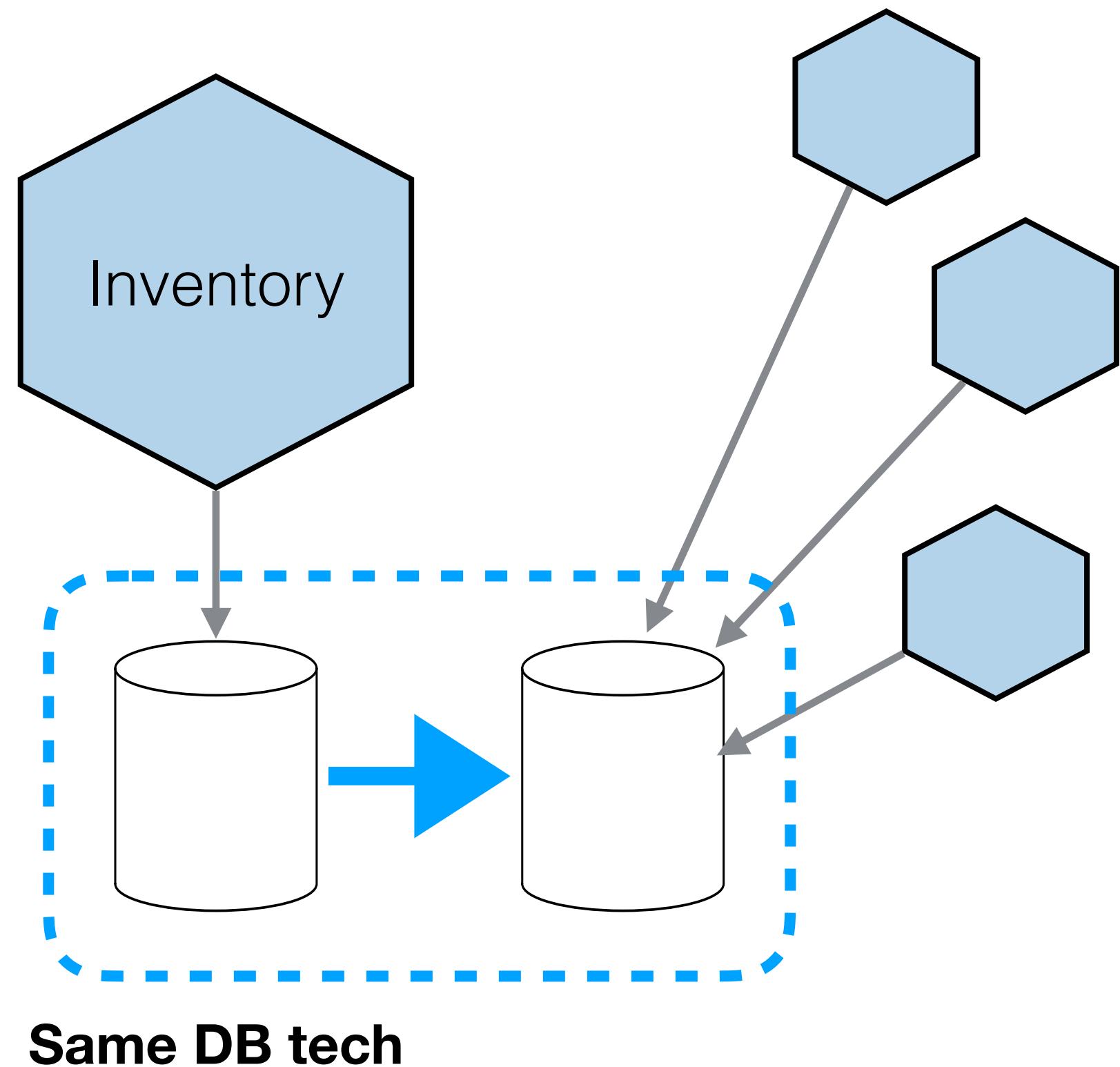
DATABASE VIEW - PROS AND CONS



DATABASE VIEW - PROS AND CONS

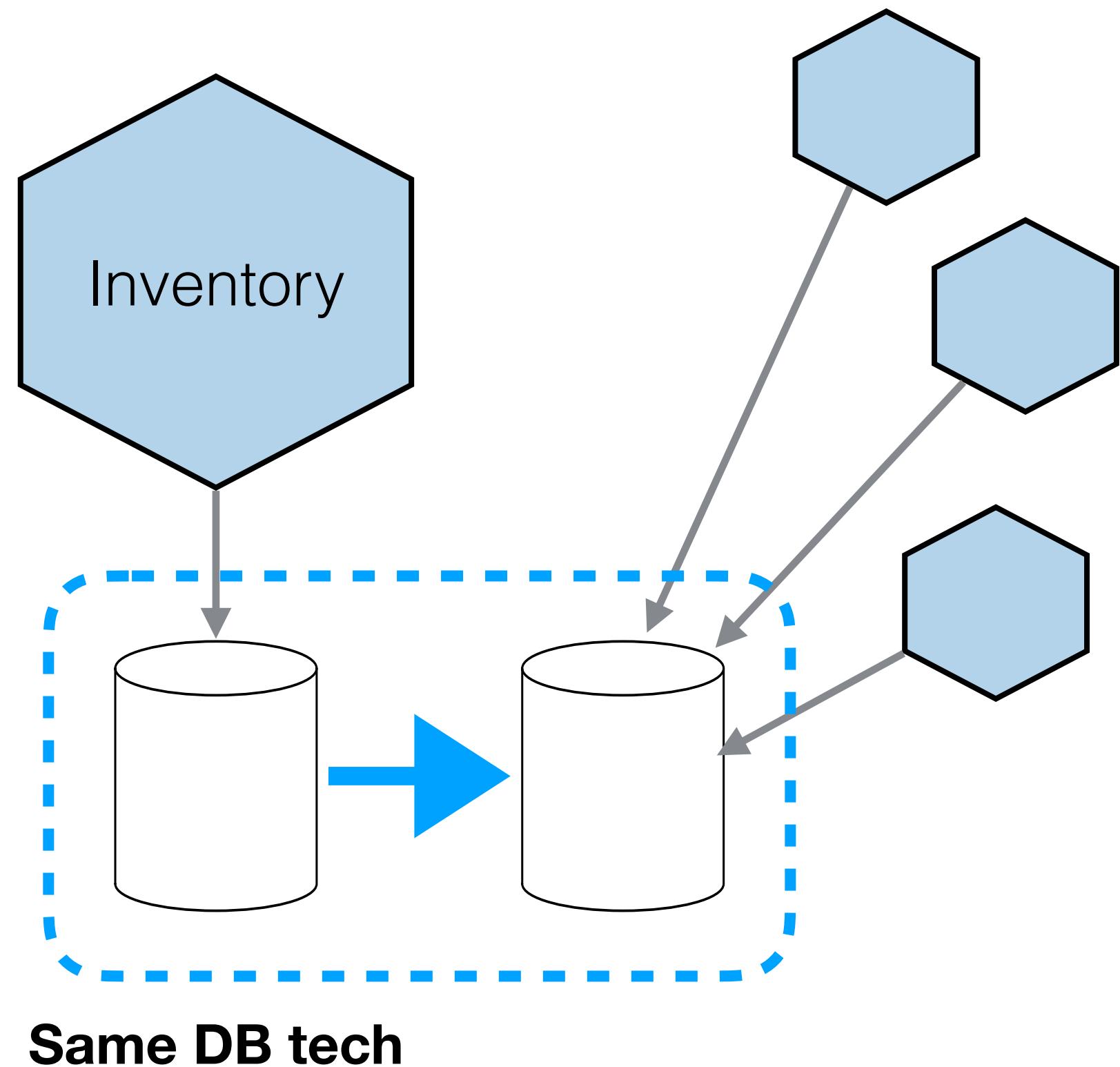


DATABASE VIEW - PROS AND CONS



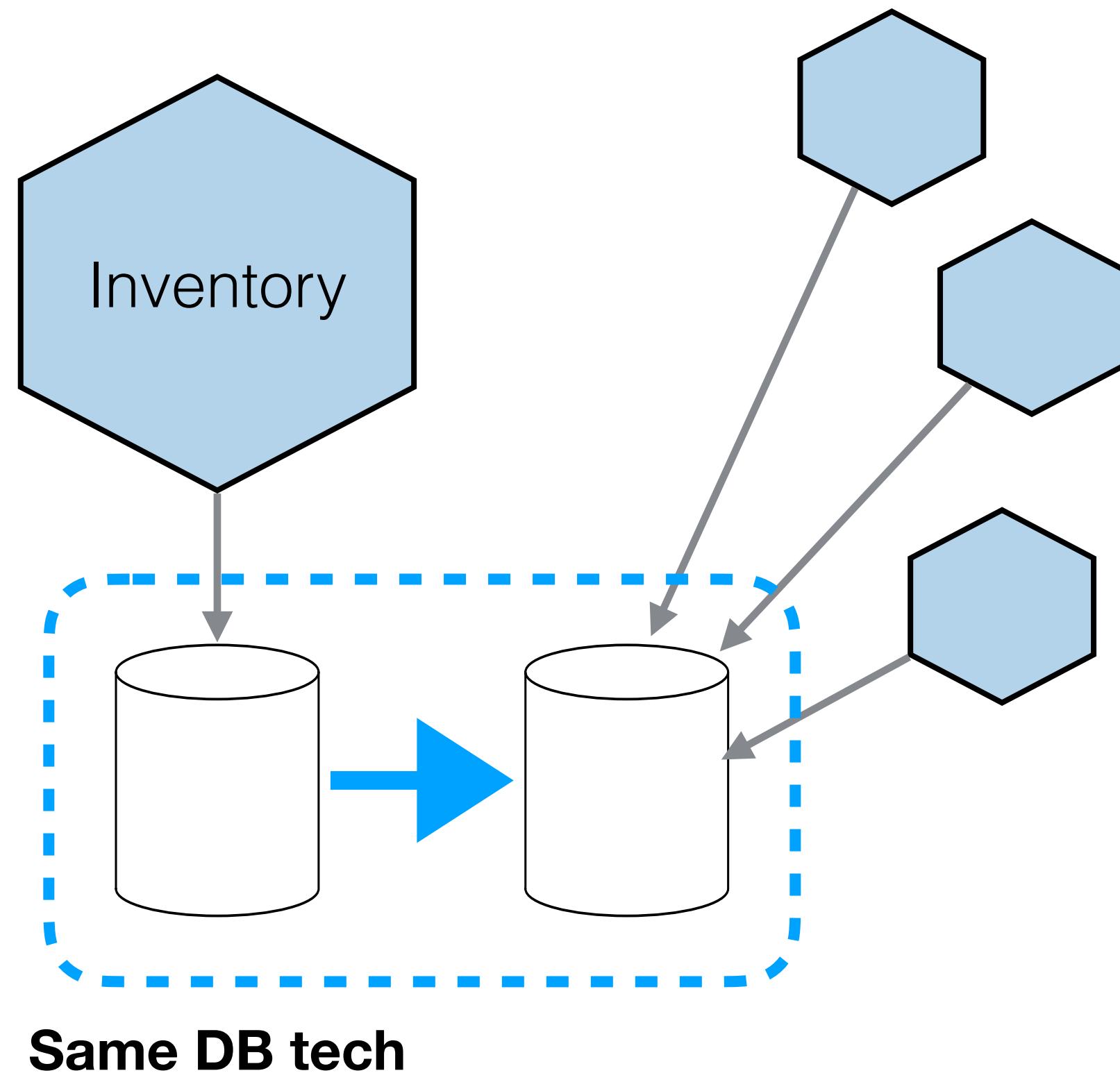
Tied to an underlying implementation

DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation
Doesn't solve writes

DATABASE VIEW - PROS AND CONS

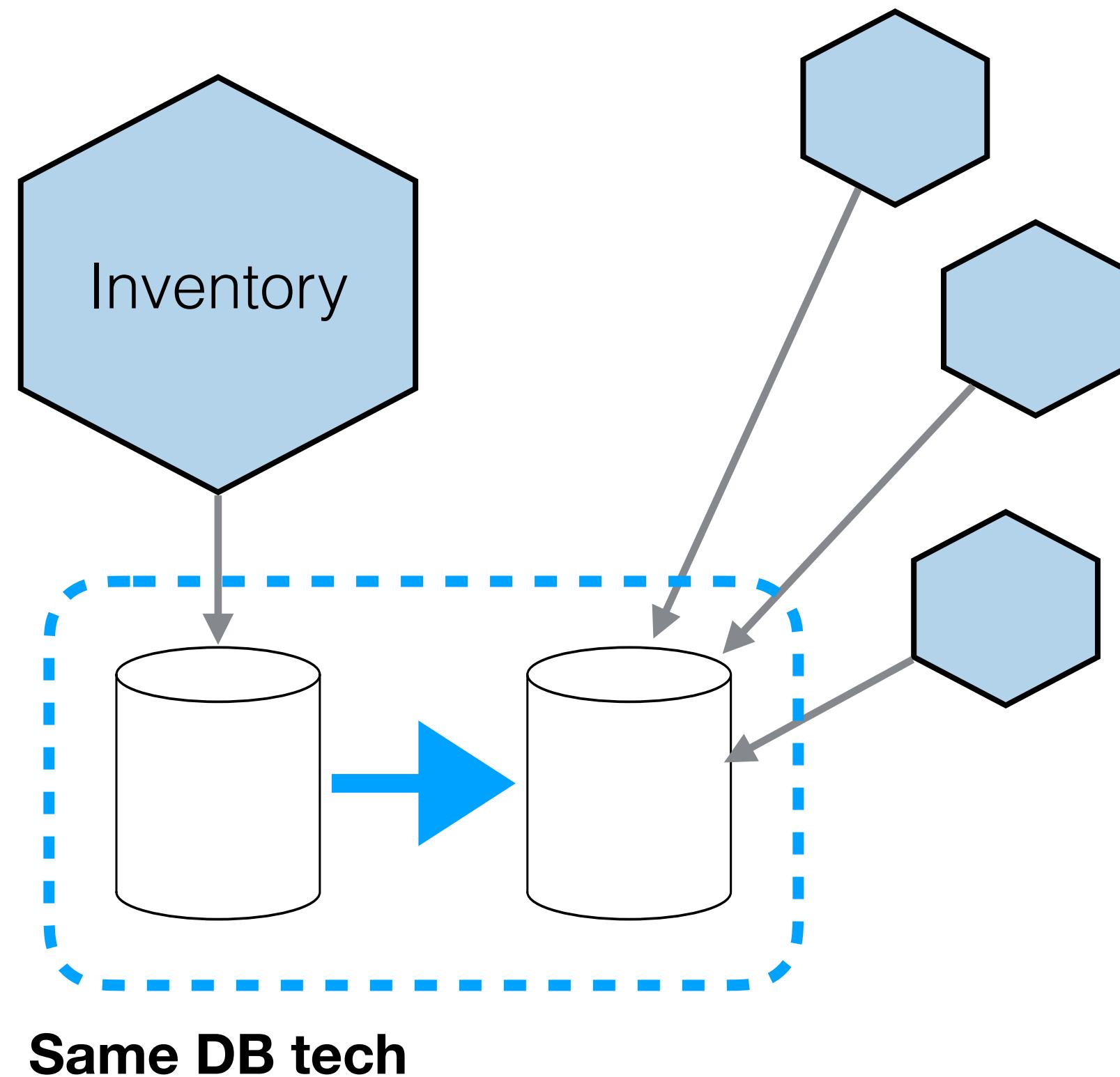


Tied to an underlying implementation

Doesn't solve writes

View mapping needs to be maintained if
the source DB changes

DATABASE VIEW - PROS AND CONS



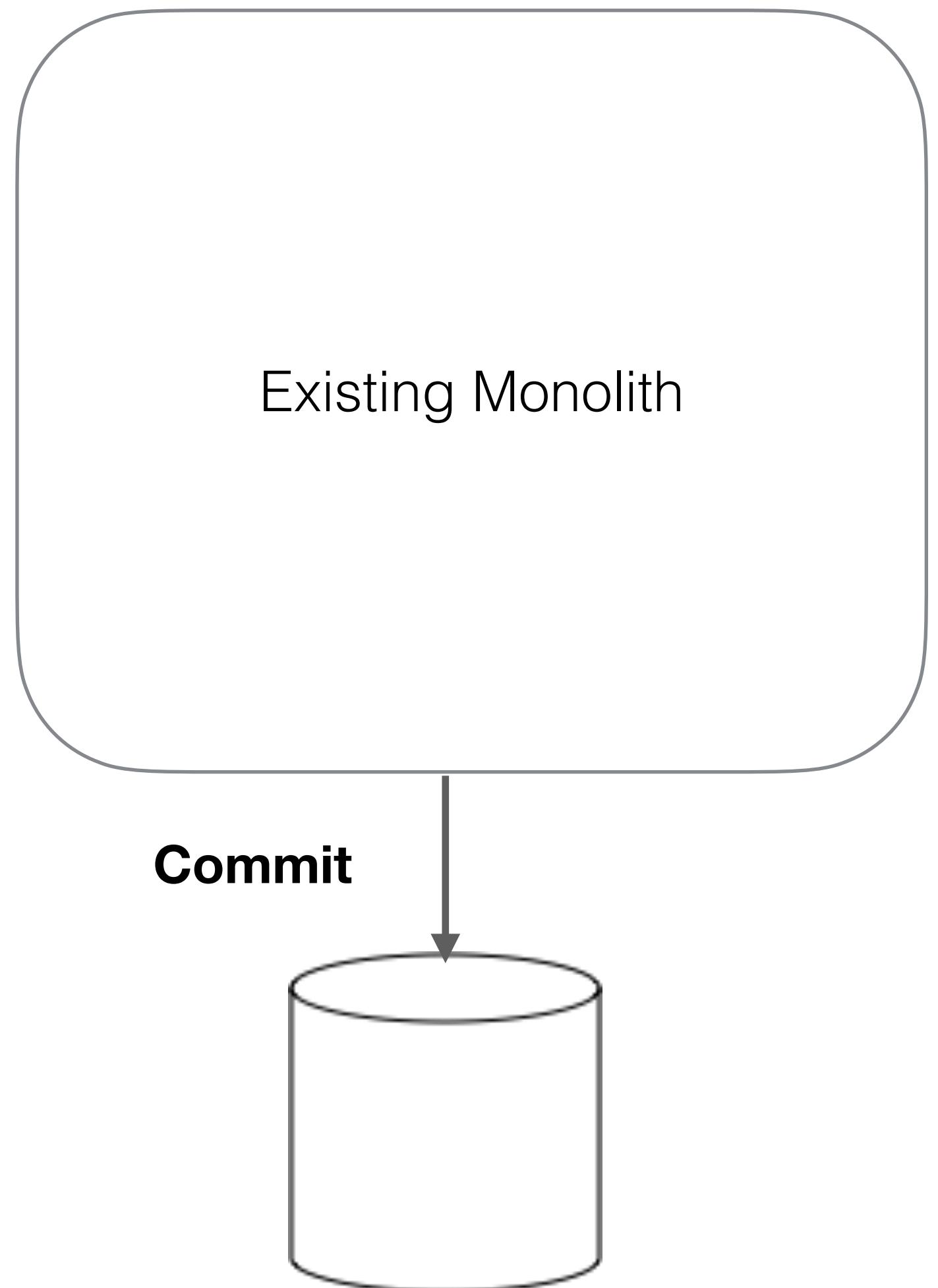
Tied to an underlying implementation

Doesn't solve writes

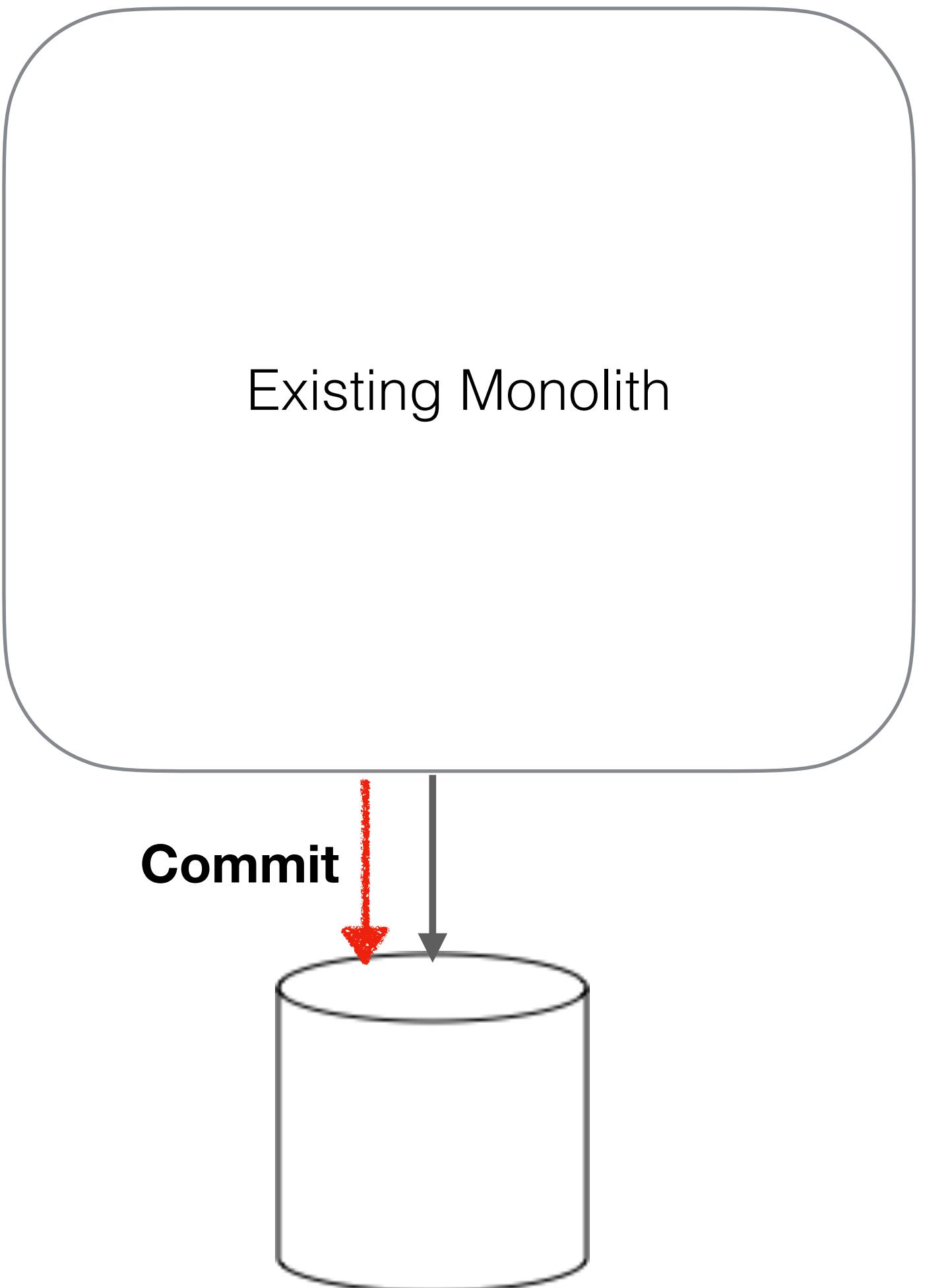
View mapping needs to be maintained if
the source DB changes

Better than direct DB access

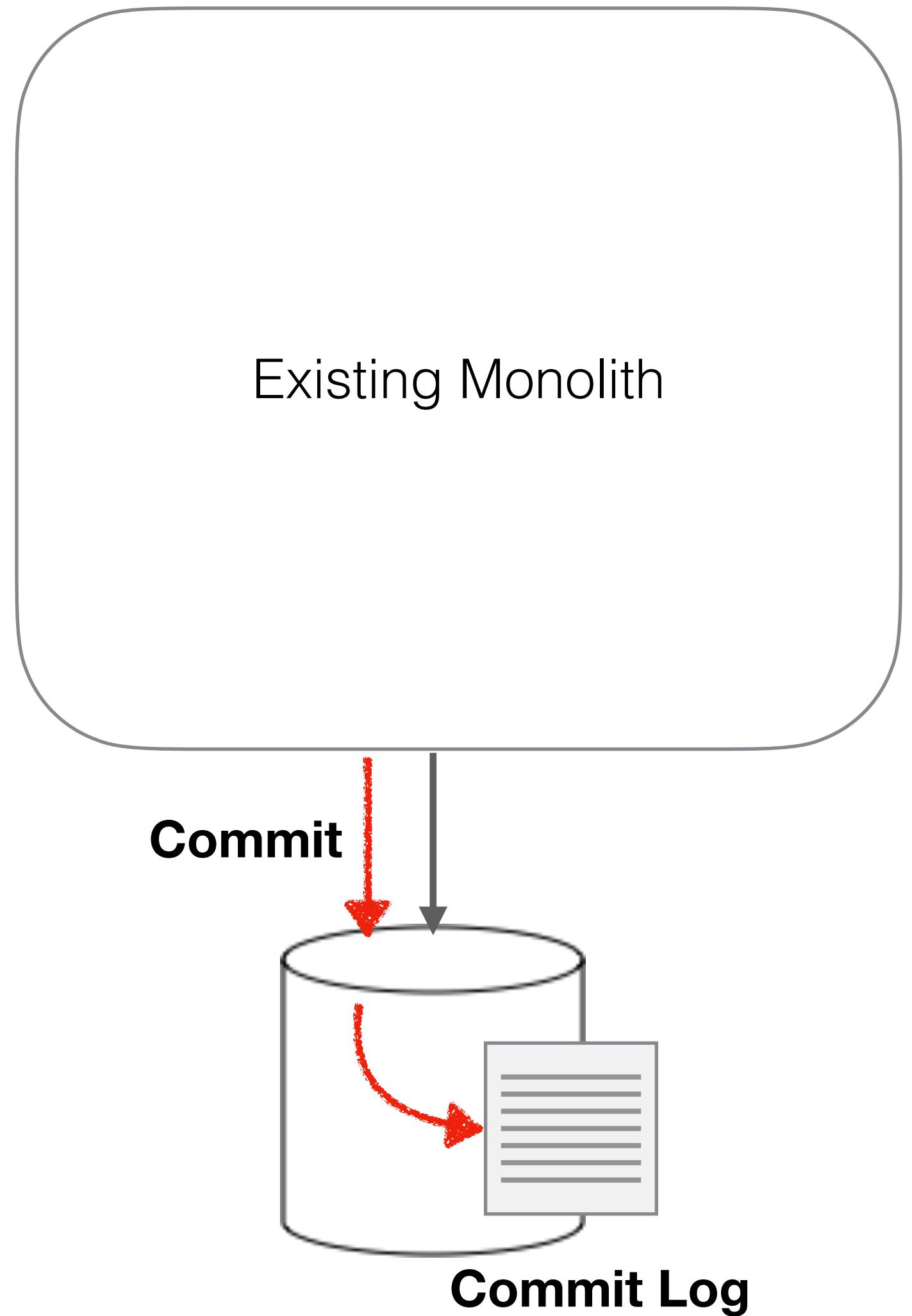
PATTERN: CHANGE DATA CAPTURE



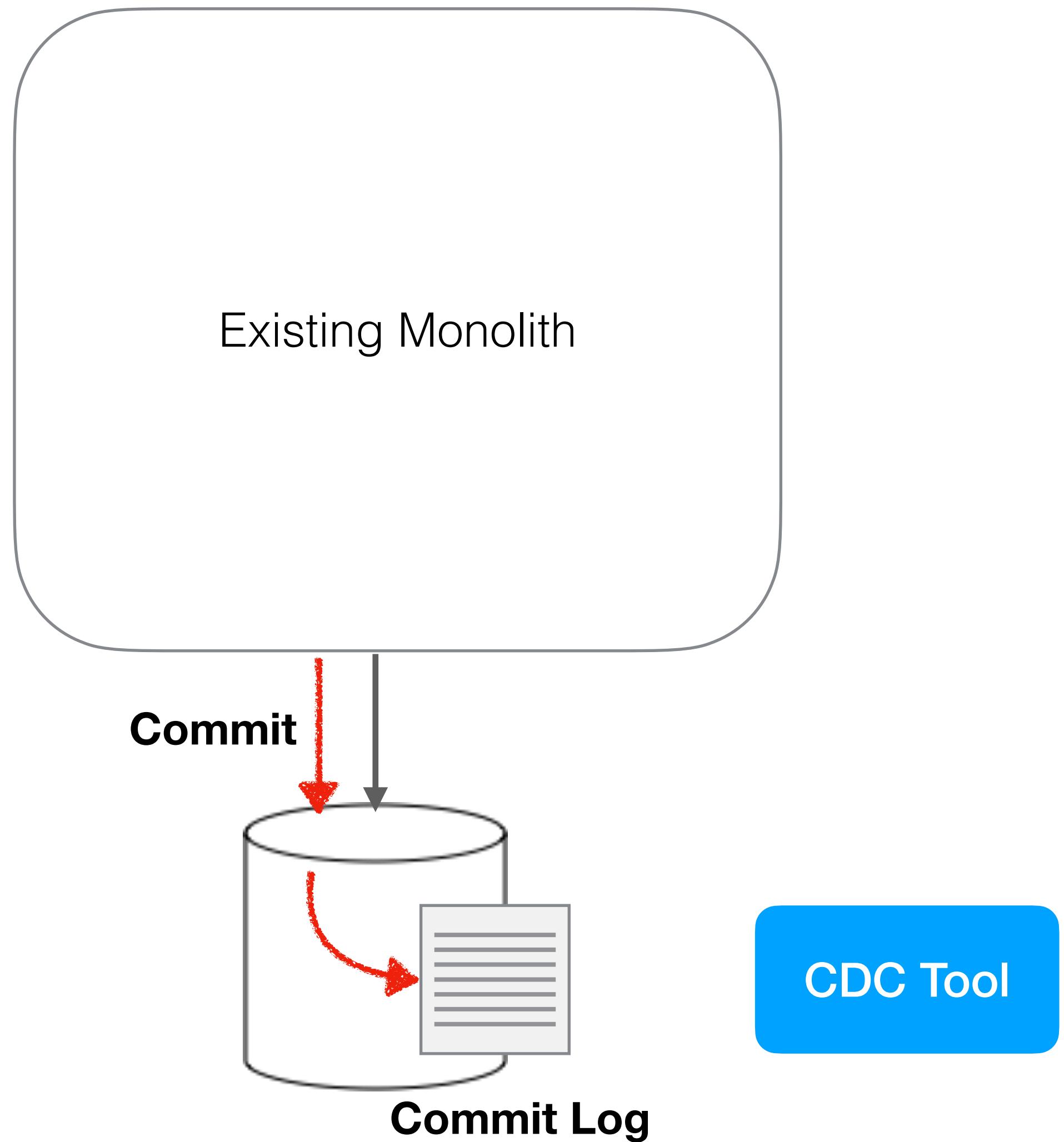
PATTERN: CHANGE DATA CAPTURE



PATTERN: CHANGE DATA CAPTURE

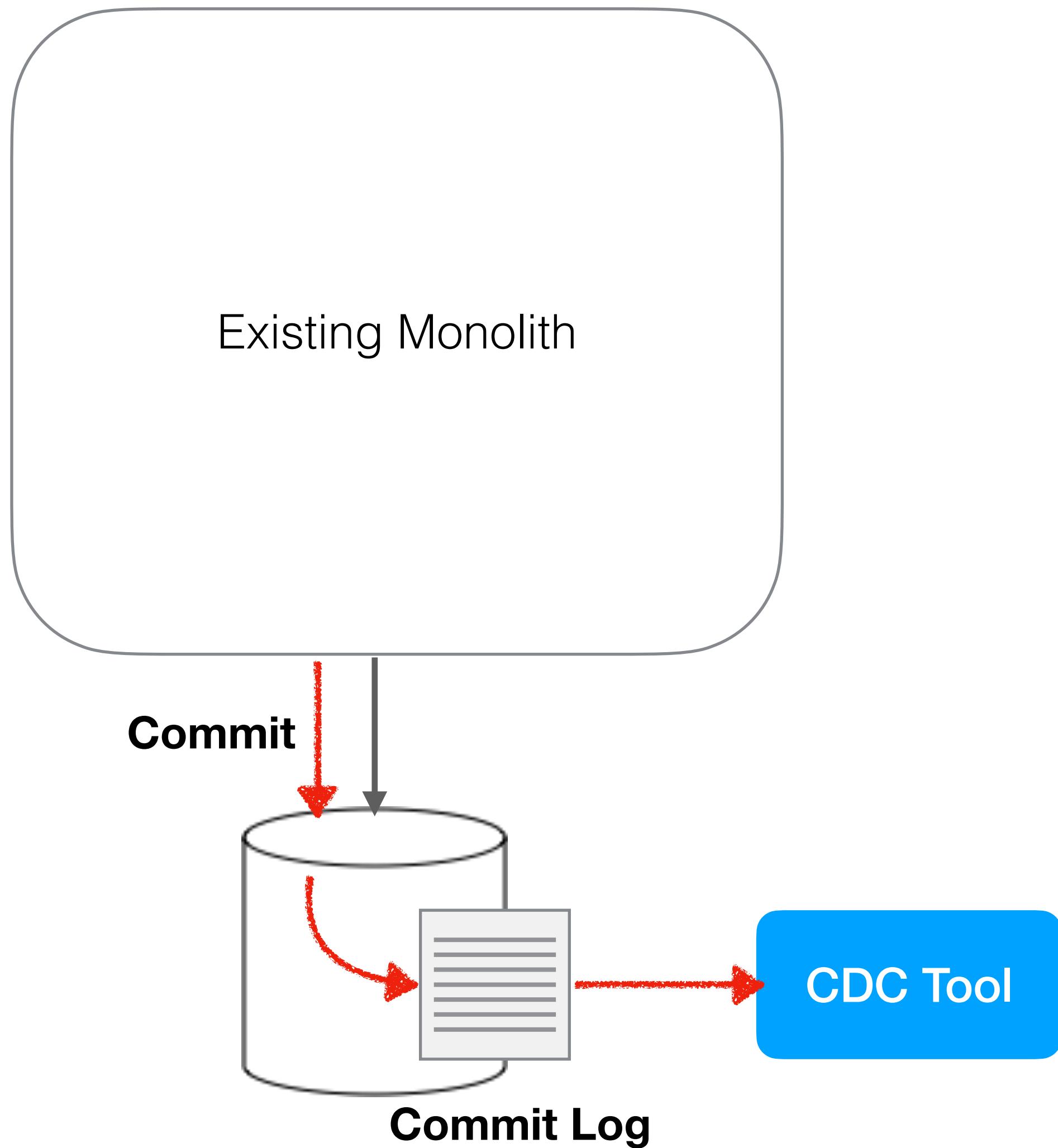


PATTERN: CHANGE DATA CAPTURE

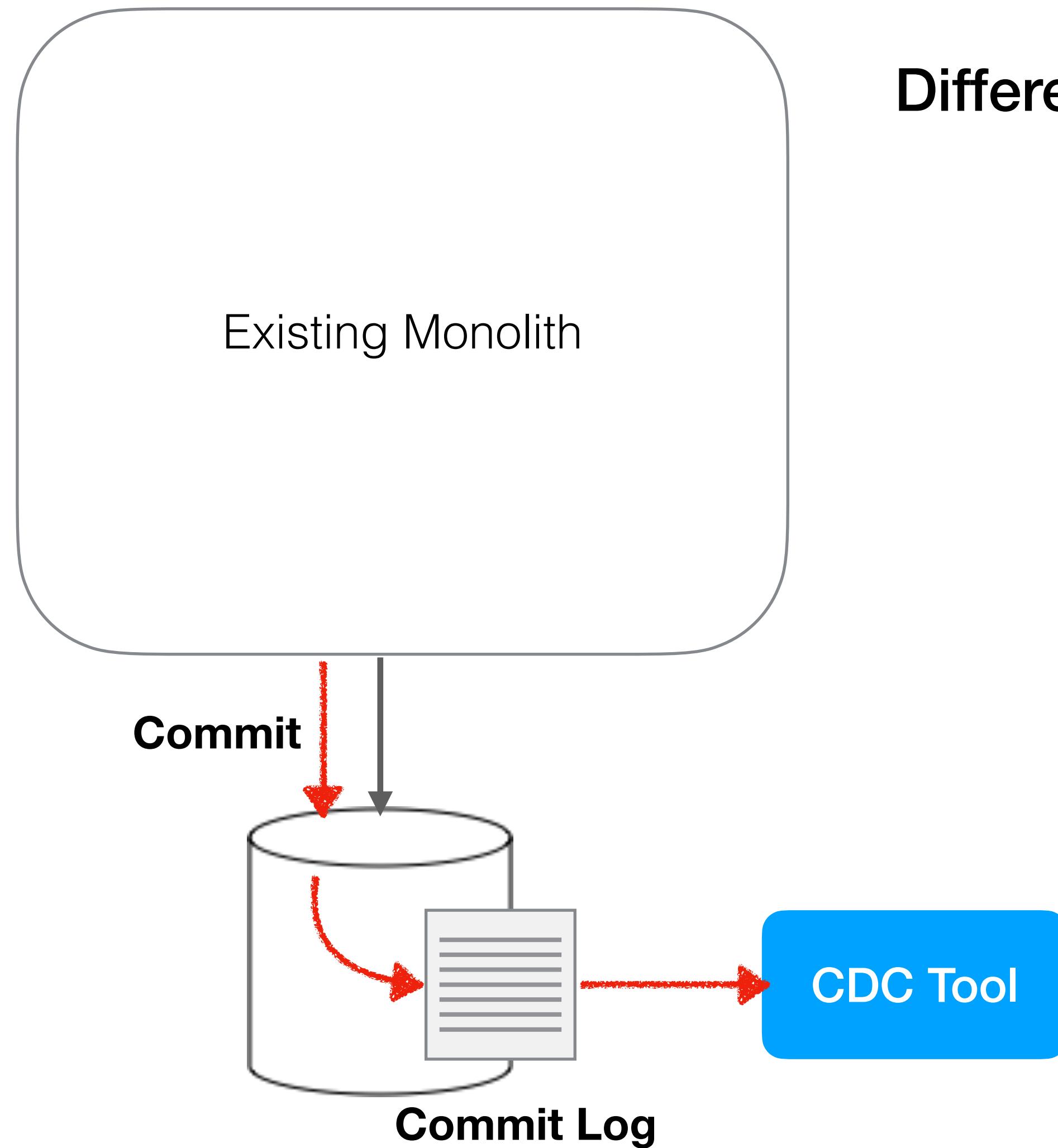


PATTERN: CHANGE DATA CAPTURE

Capture data as its inserted



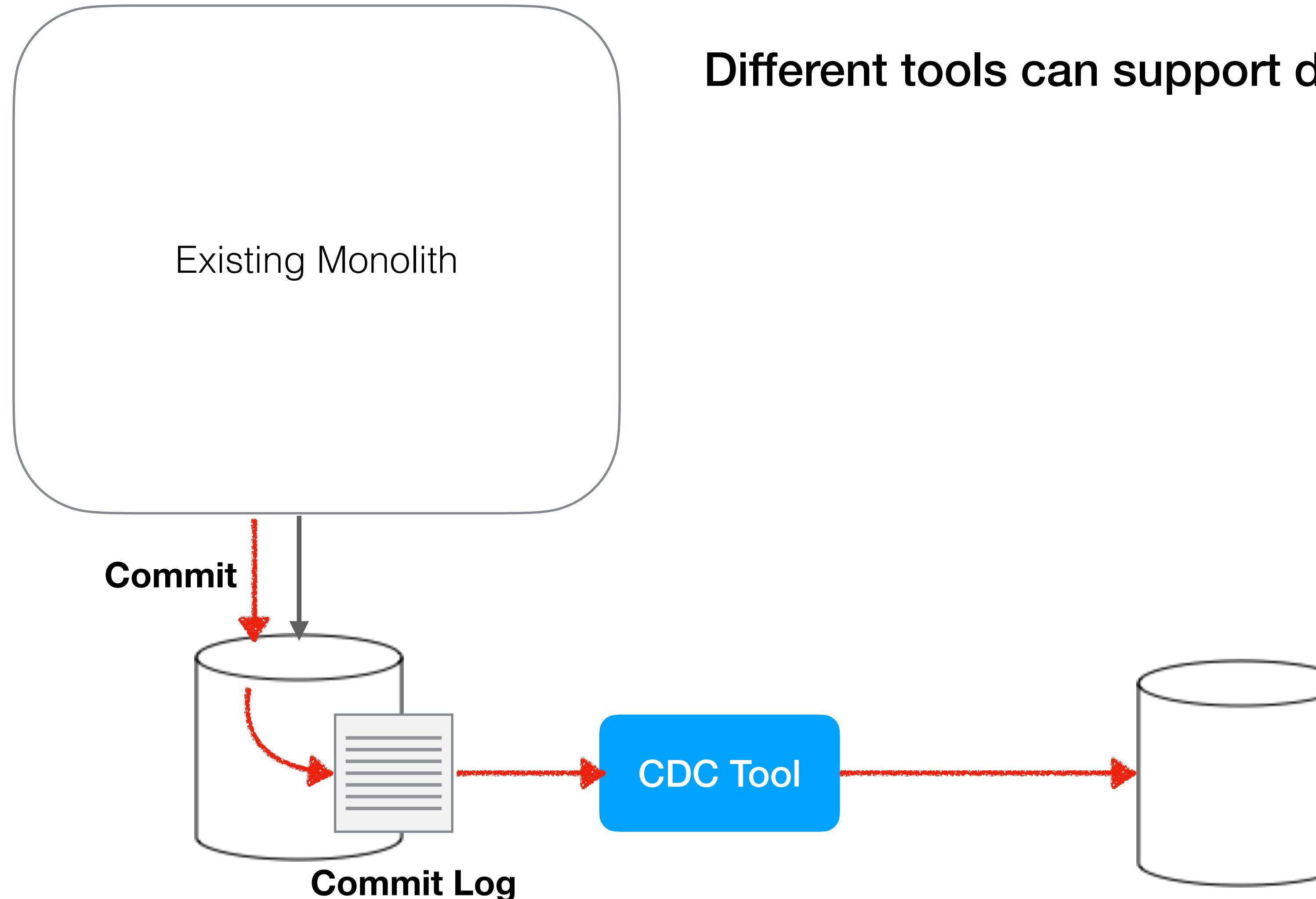
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

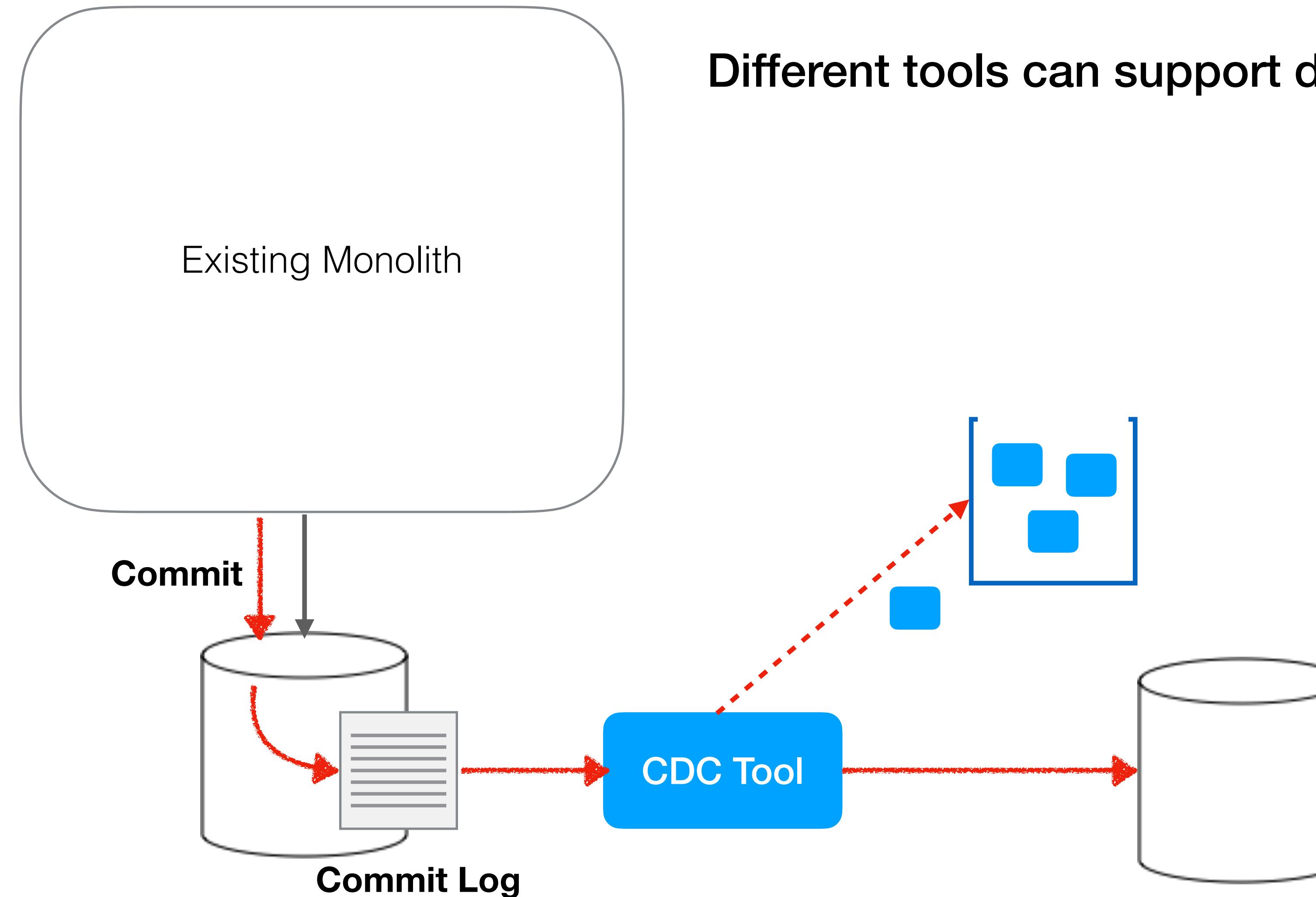
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

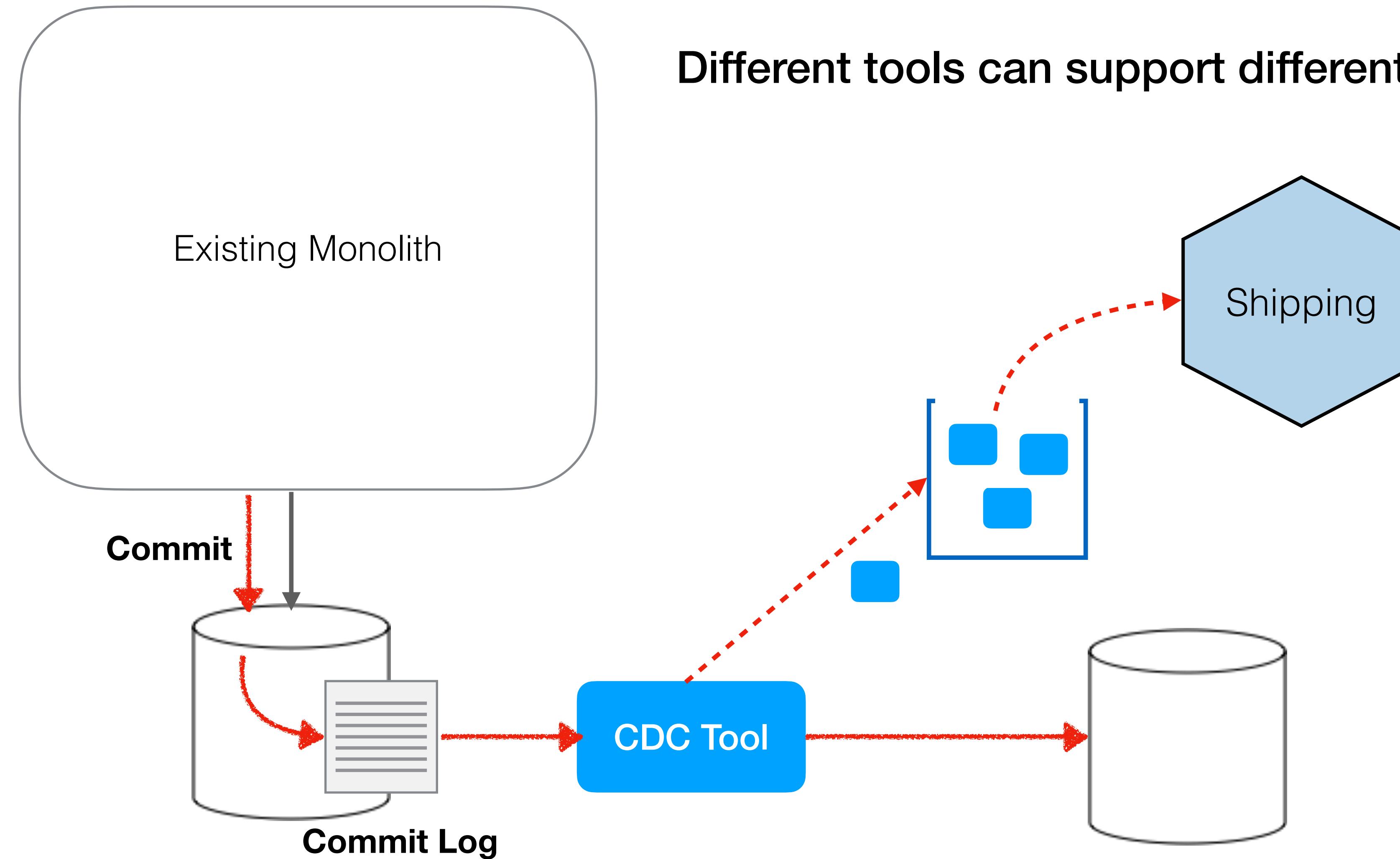
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

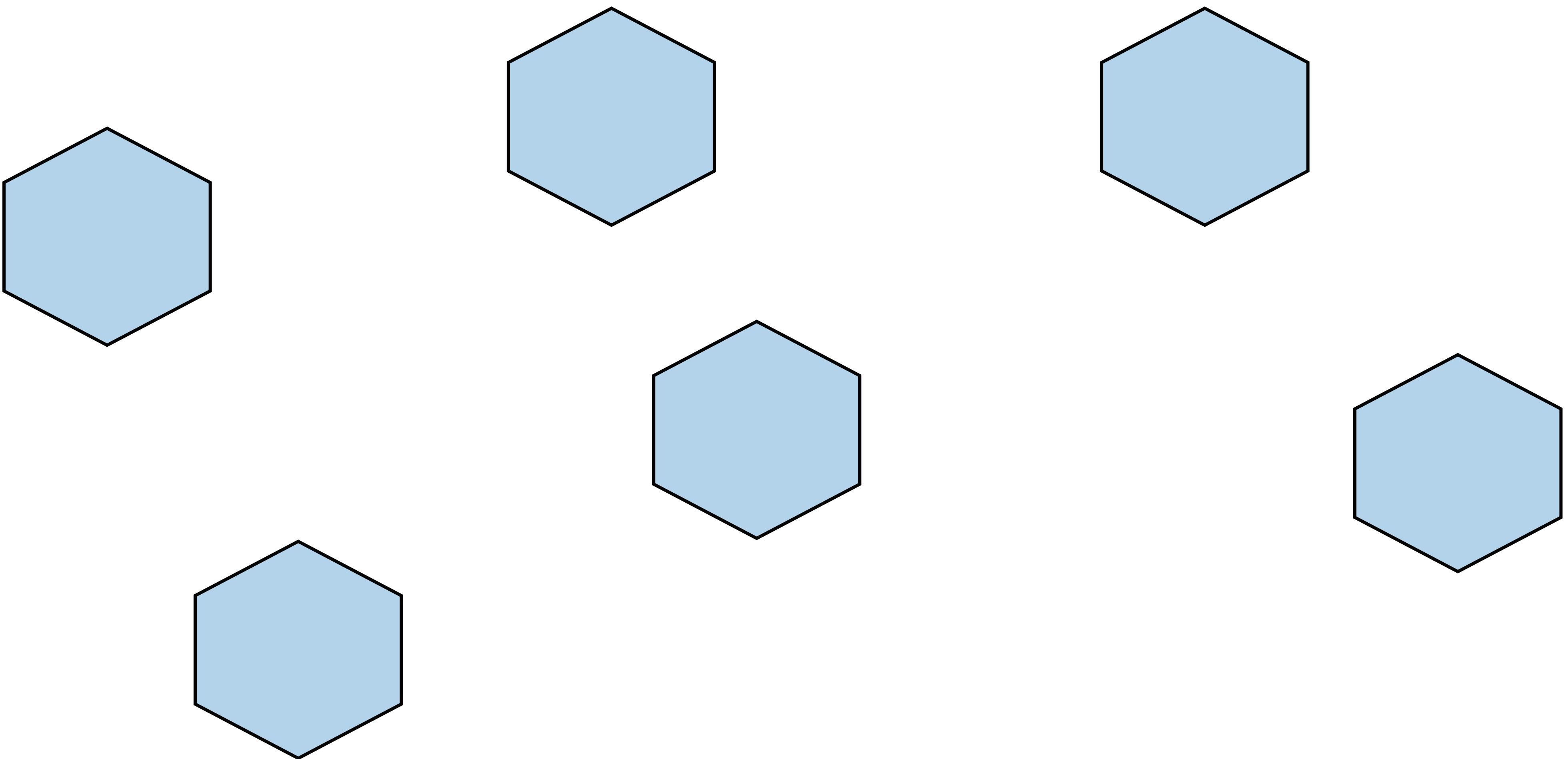
Different tools can support different destinations

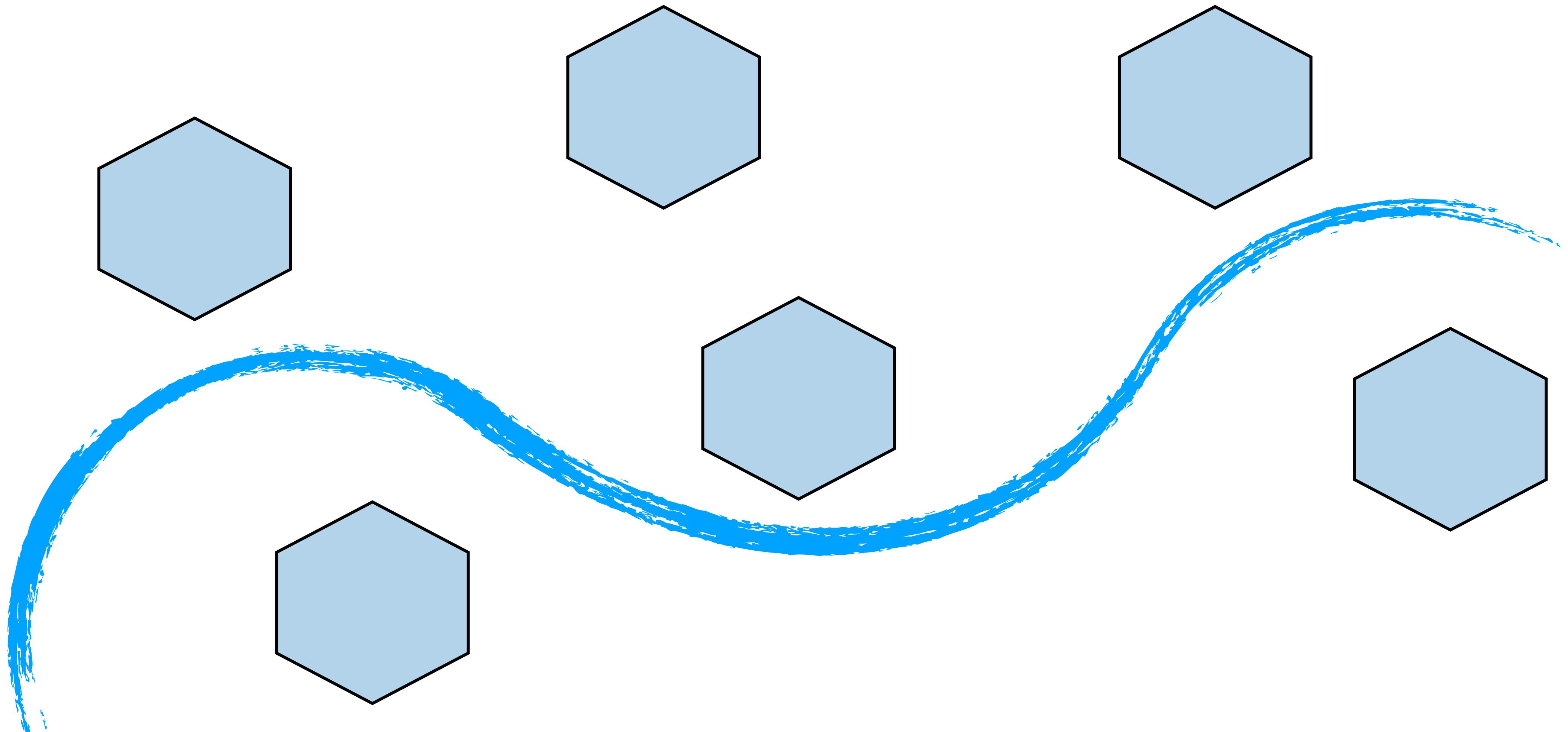
PATTERN: CHANGE DATA CAPTURE

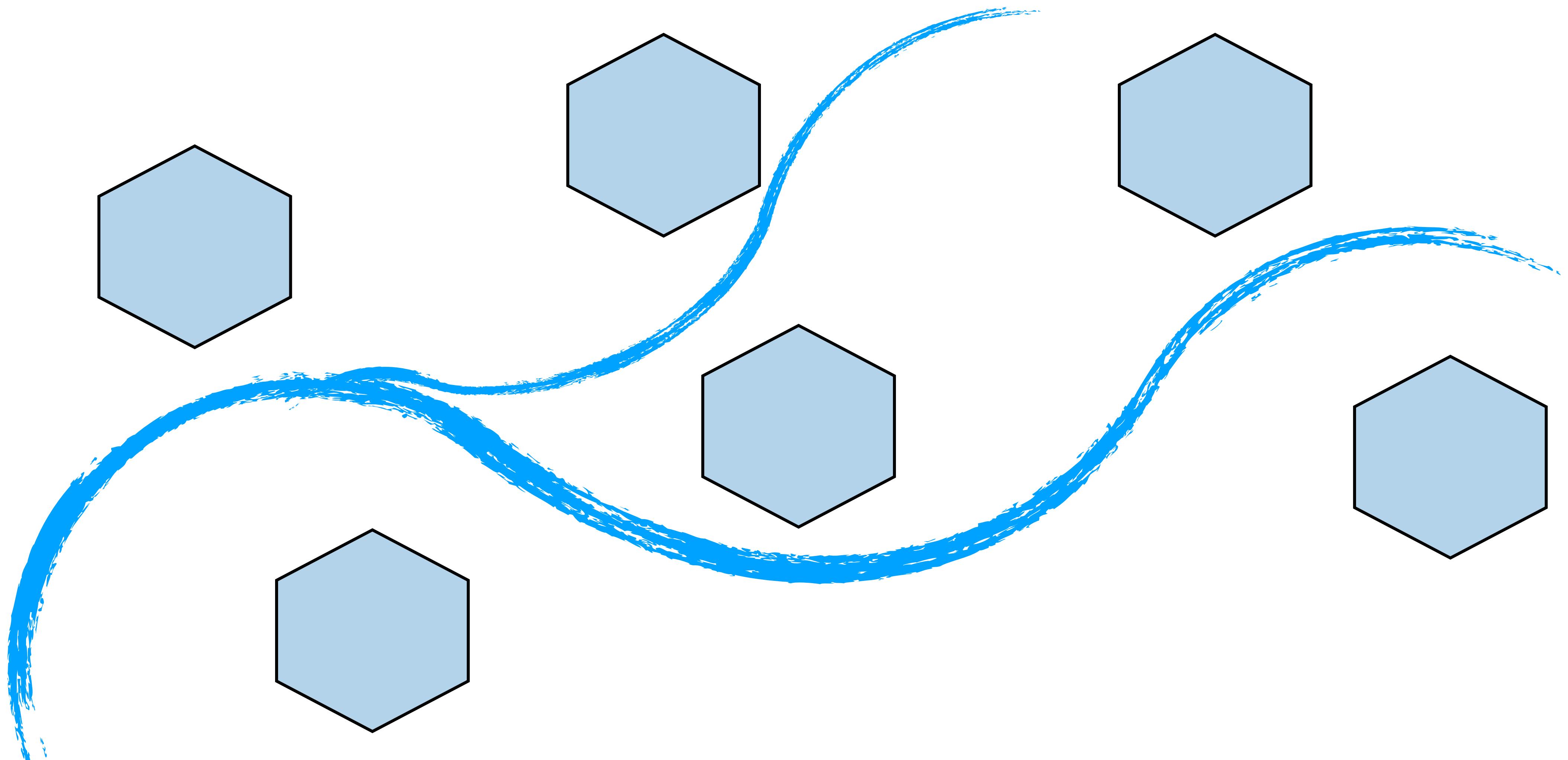


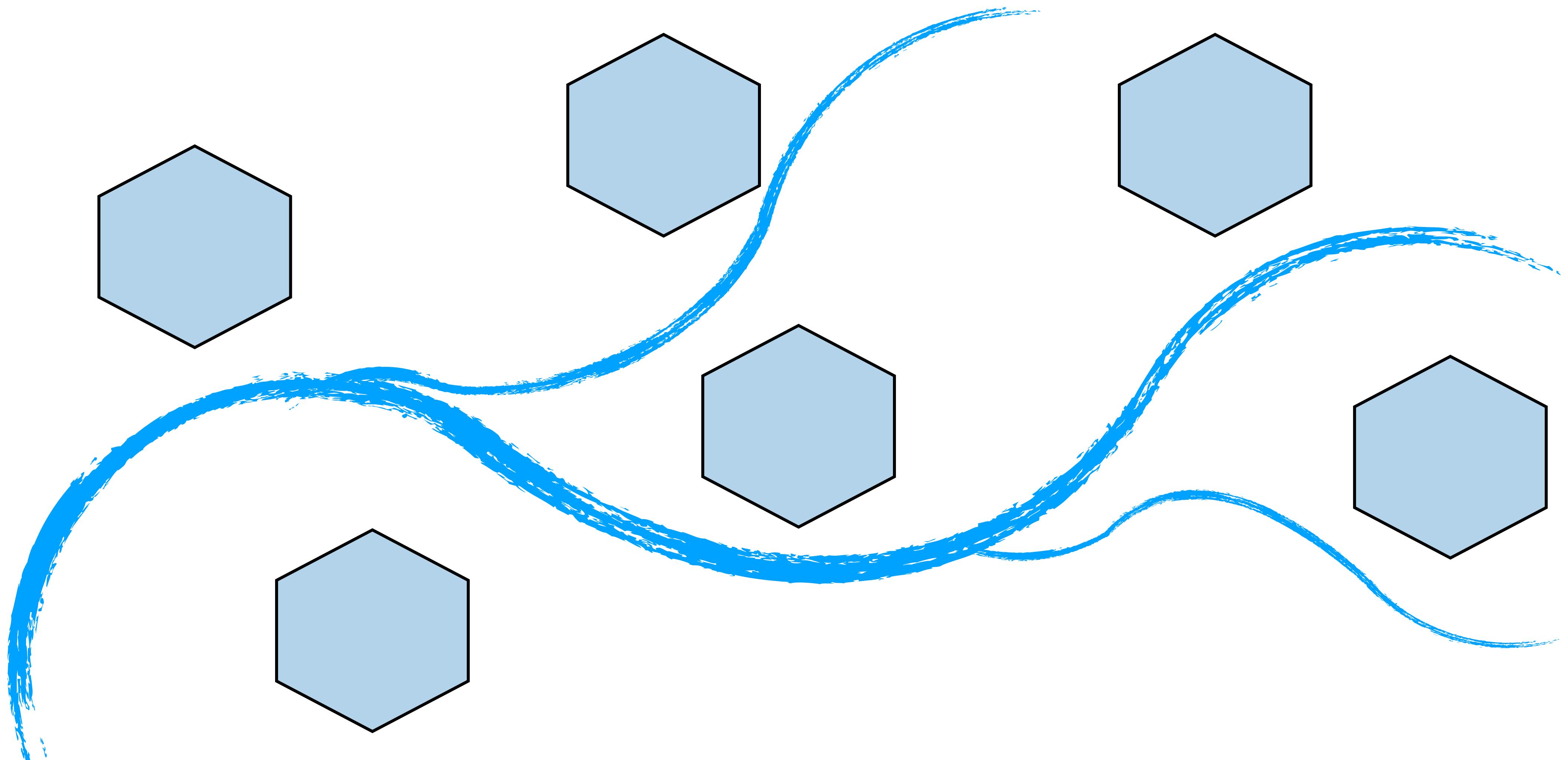
Capture data as its inserted

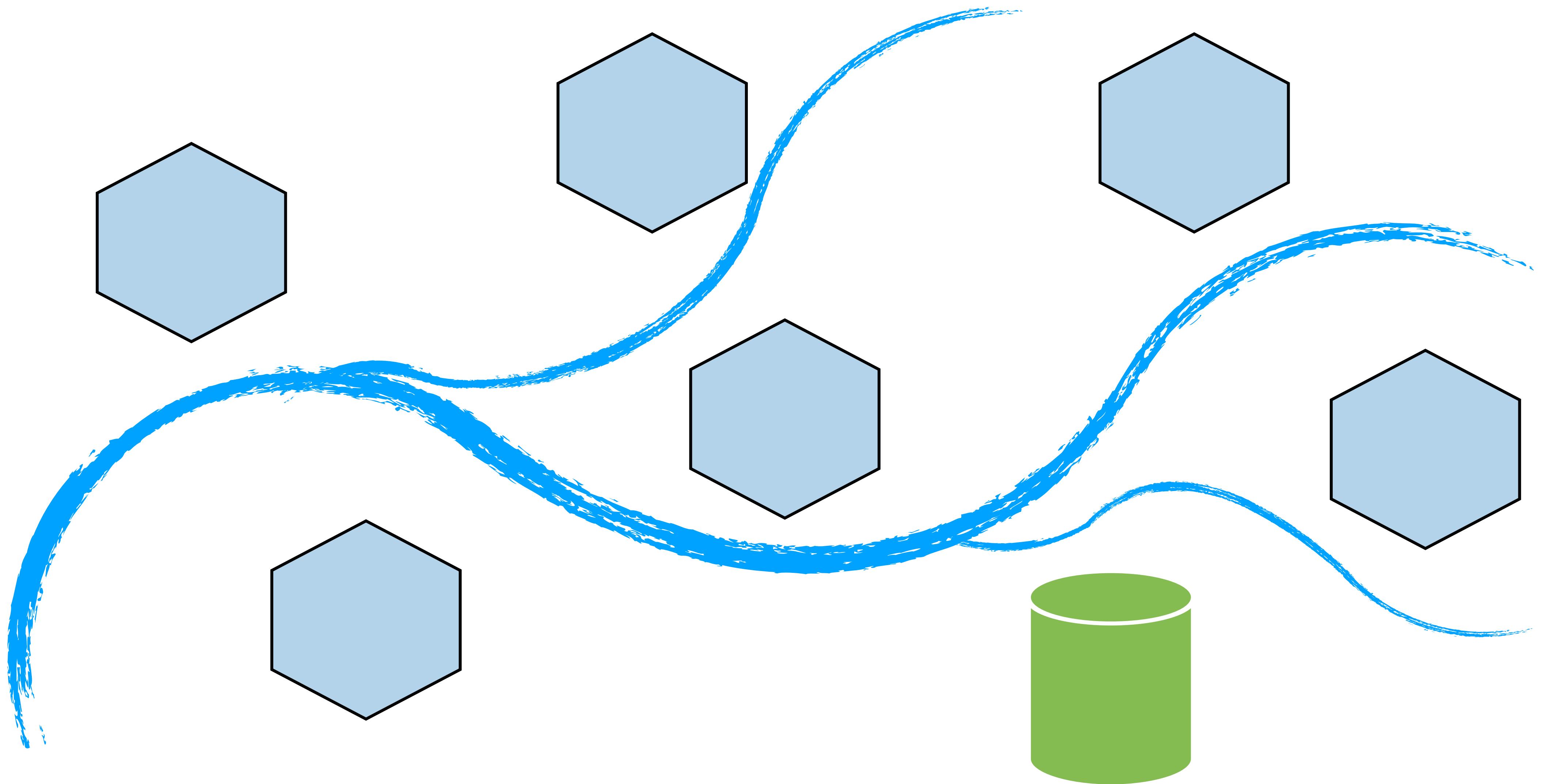
Different tools can support different destinations

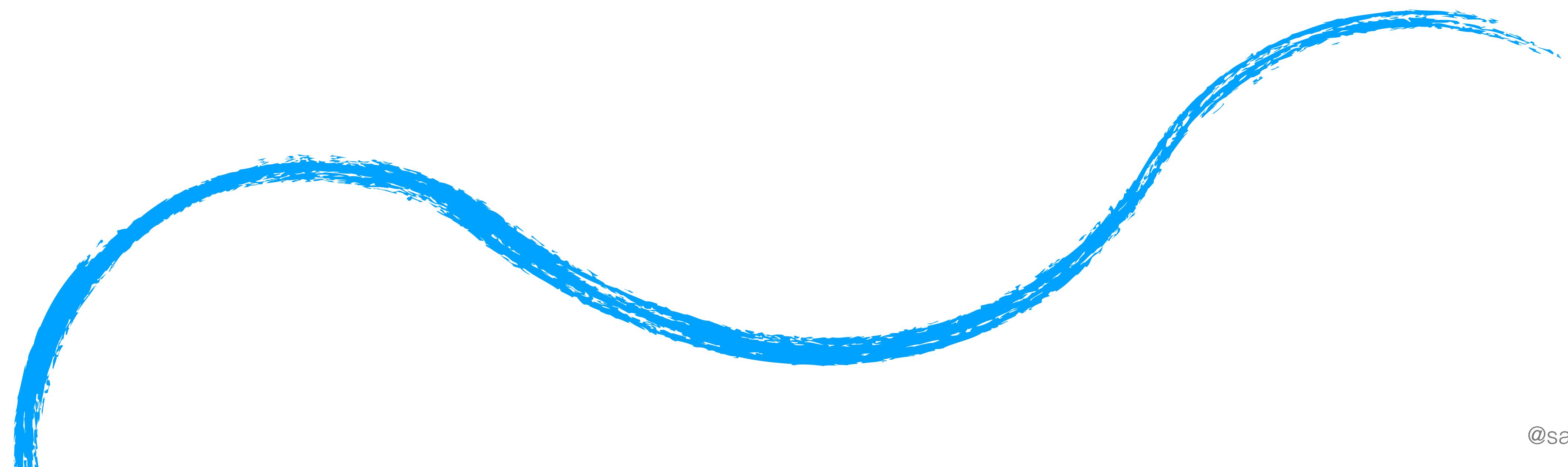
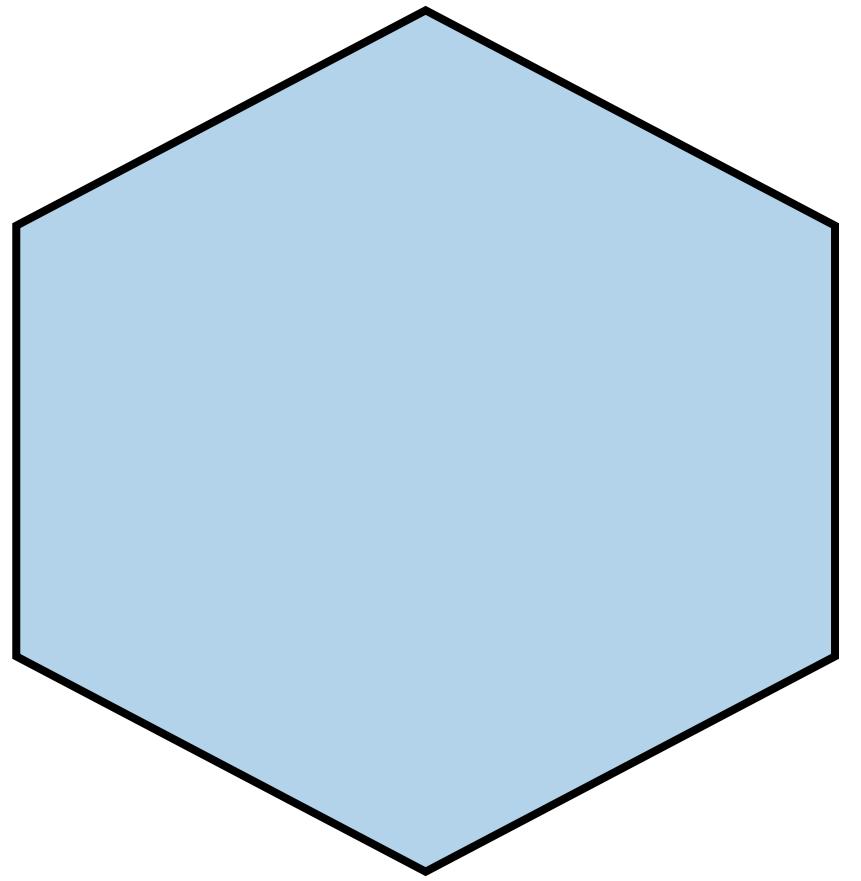




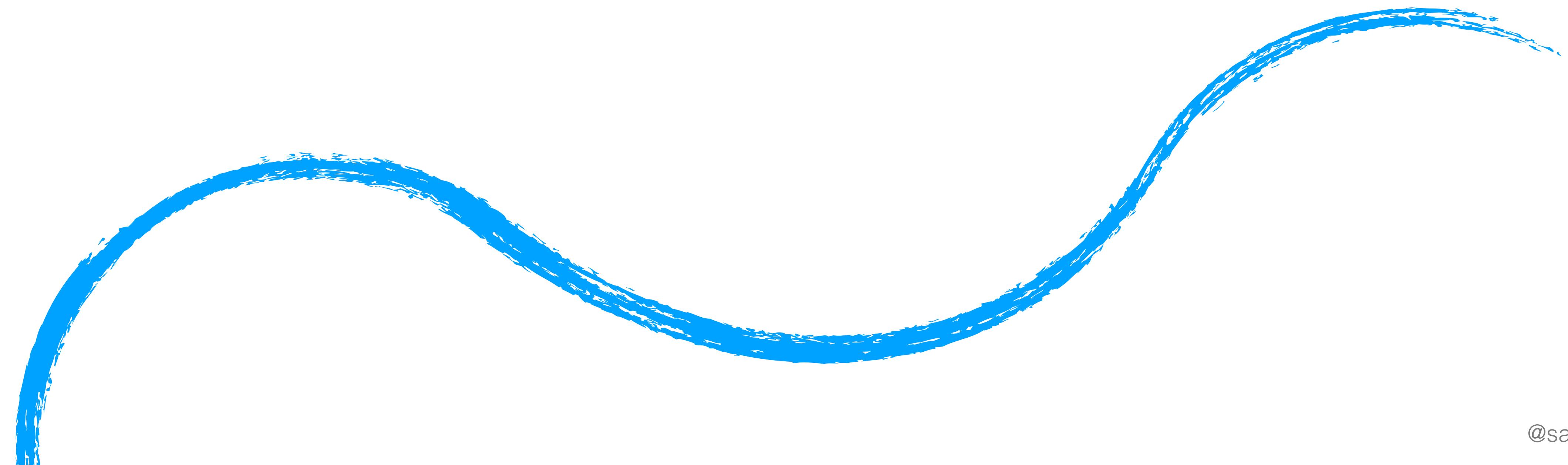
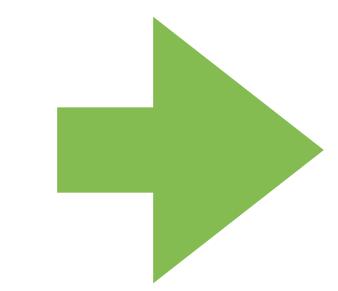
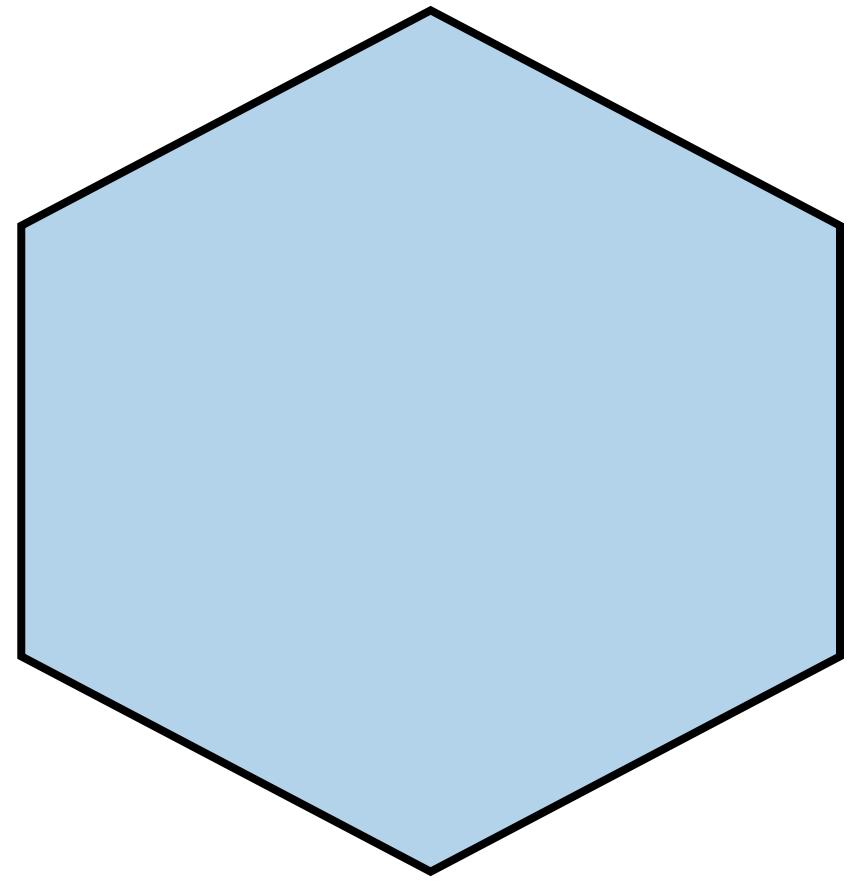






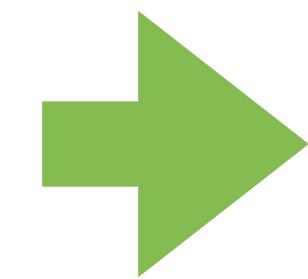
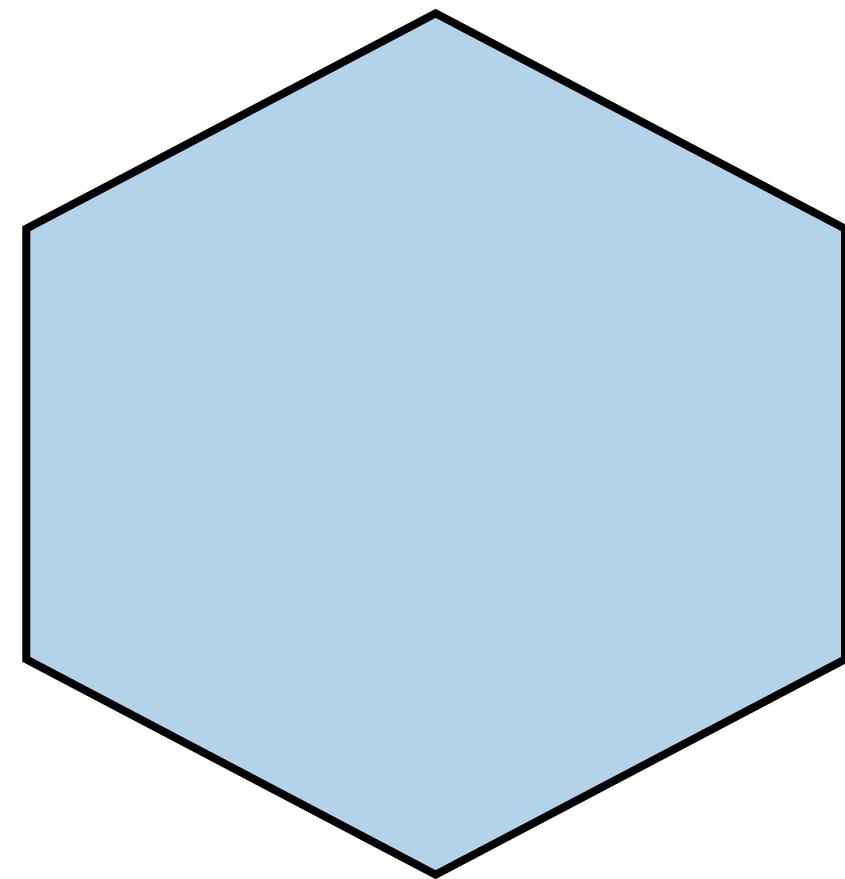


@samnewman

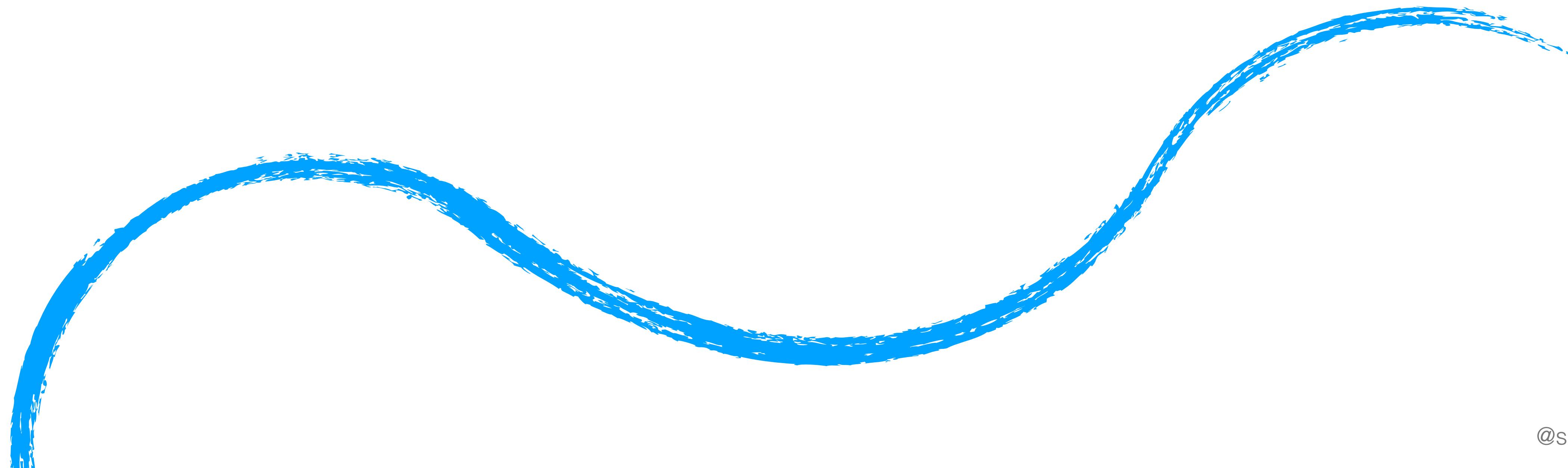


@samnewman

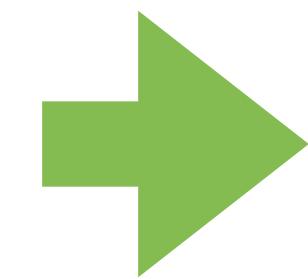
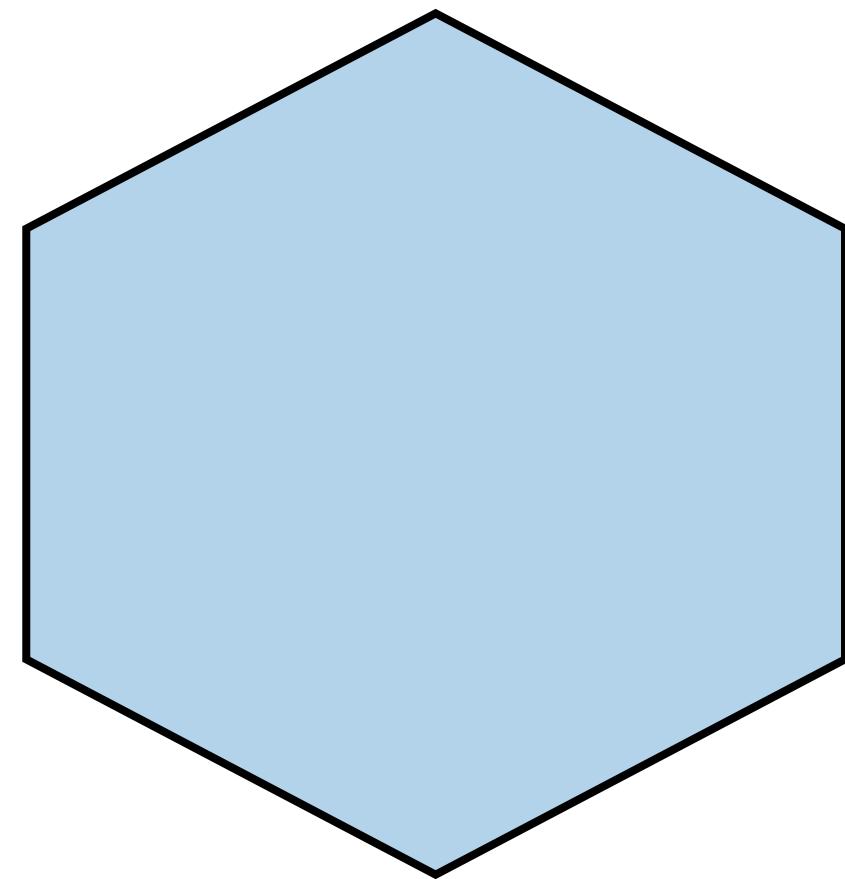
Event



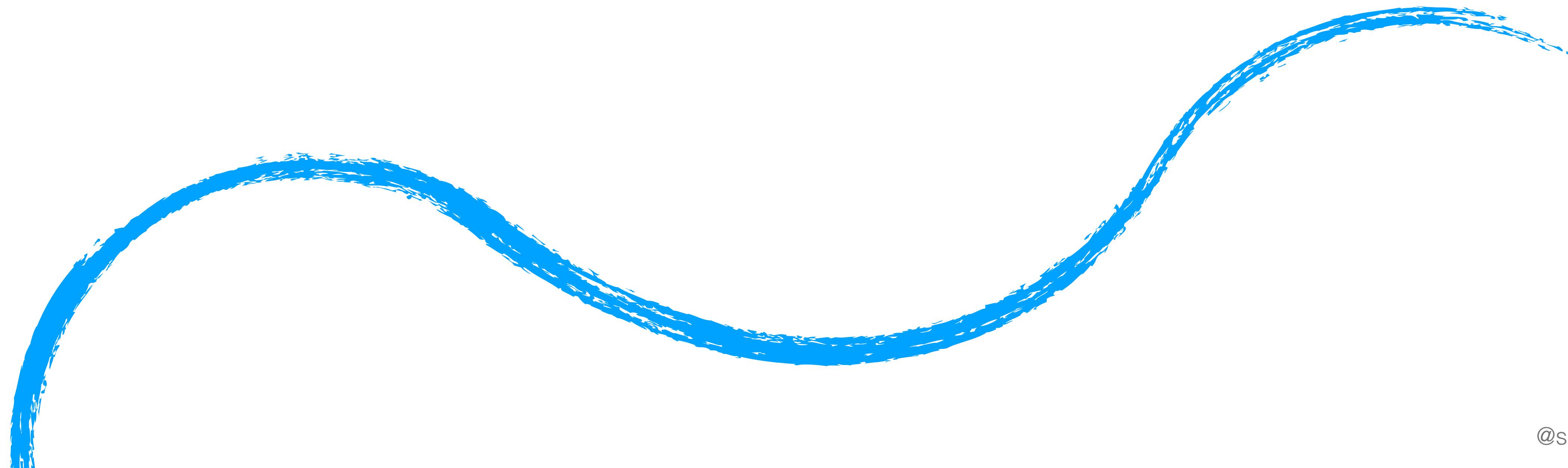
Order Updated!



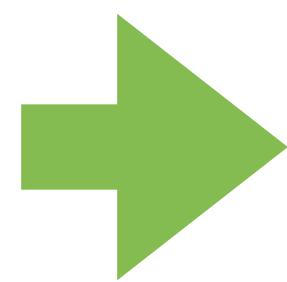
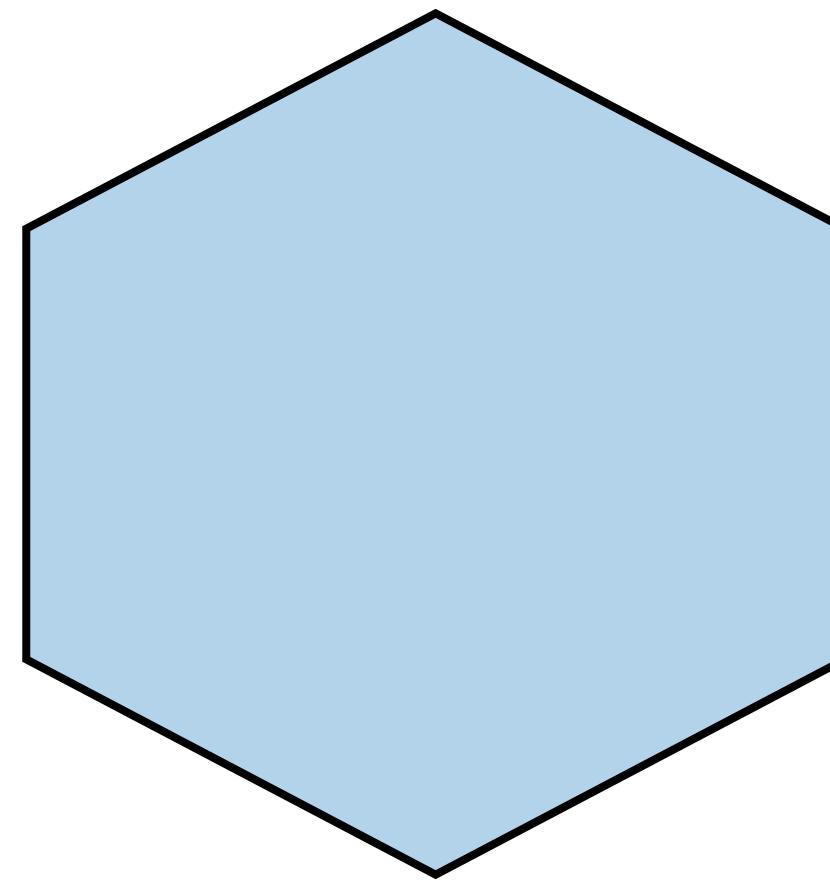
Event



Order Updated!
04/08/2020



Event

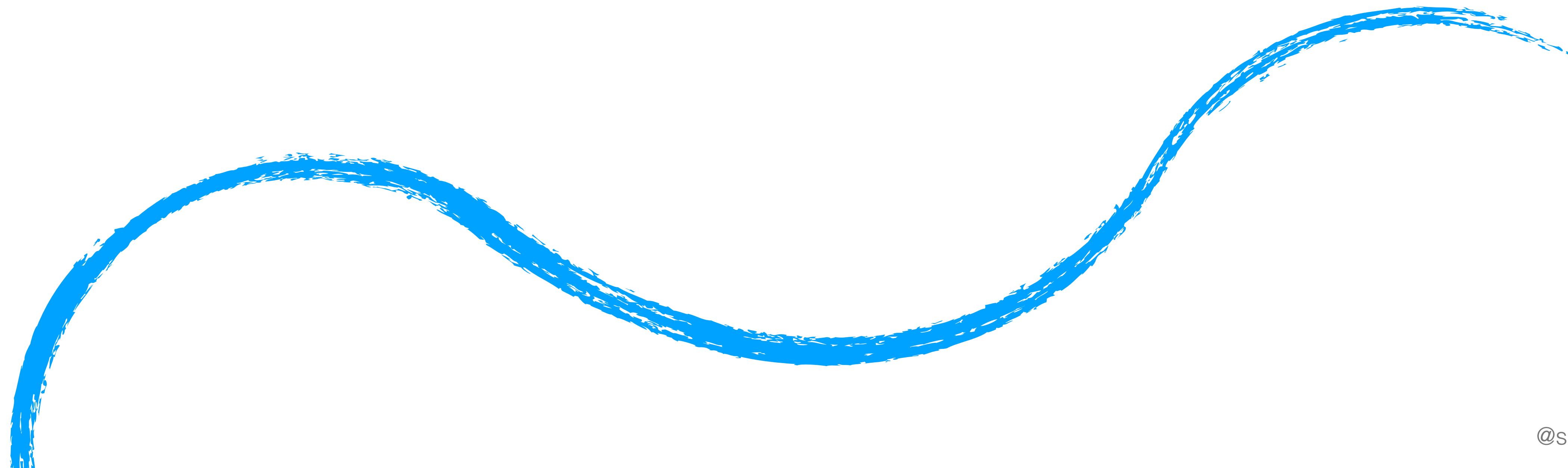


Order Updated!

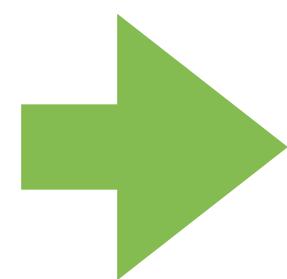
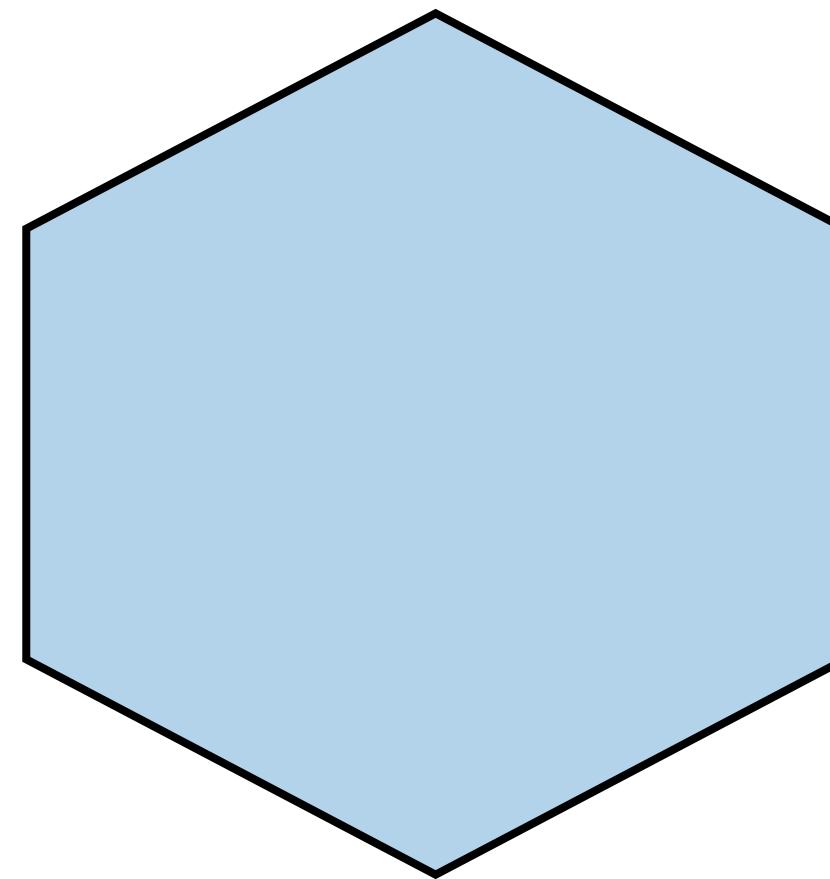
04/08/2020

Customer: Sam Newman

Order Value: \$130.25



Event

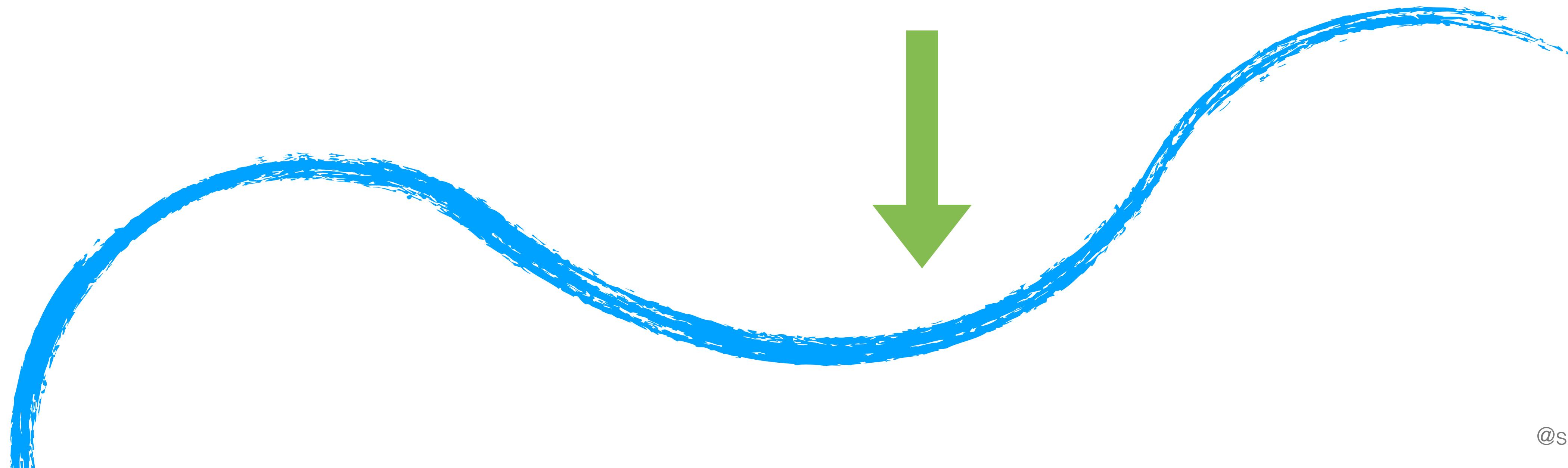
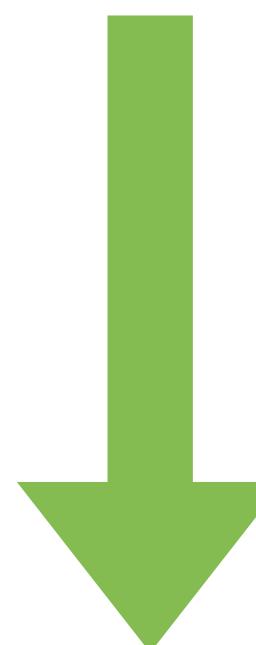


Order Updated!

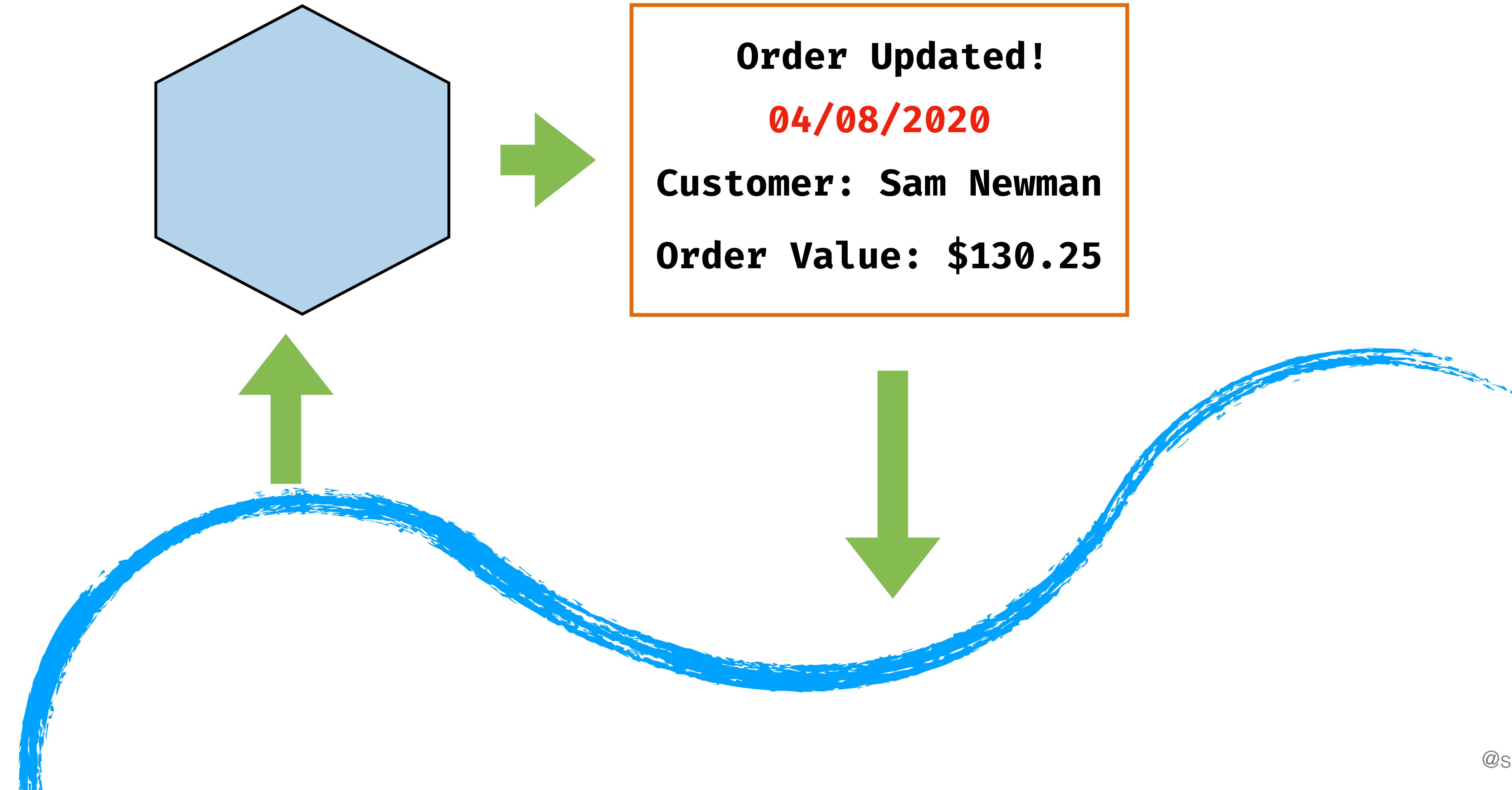
04/08/2020

Customer: Sam Newman

Order Value: \$130.25



Event



POLL: ARE ANY OF YOU USING KAFKA?

Yes - we run our own Kafka cluster

Yes - we use a managed solution

No - but we use something similar

No

AGENDA

Introduction

Shared Data Patterns

Migration Order

Decomposition Patterns

AGENDA

Introduction

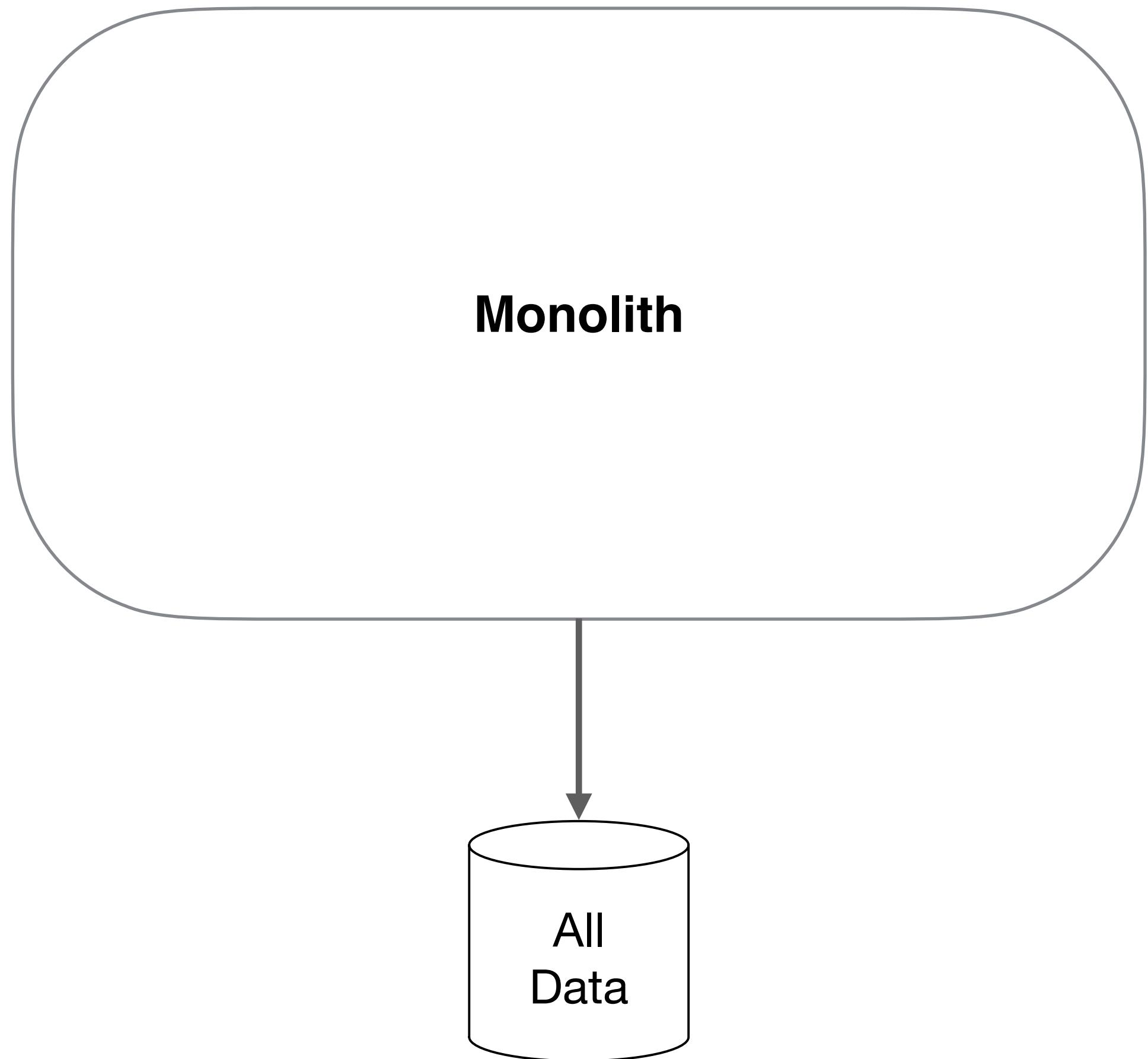
Shared Data Patterns

Migration Order

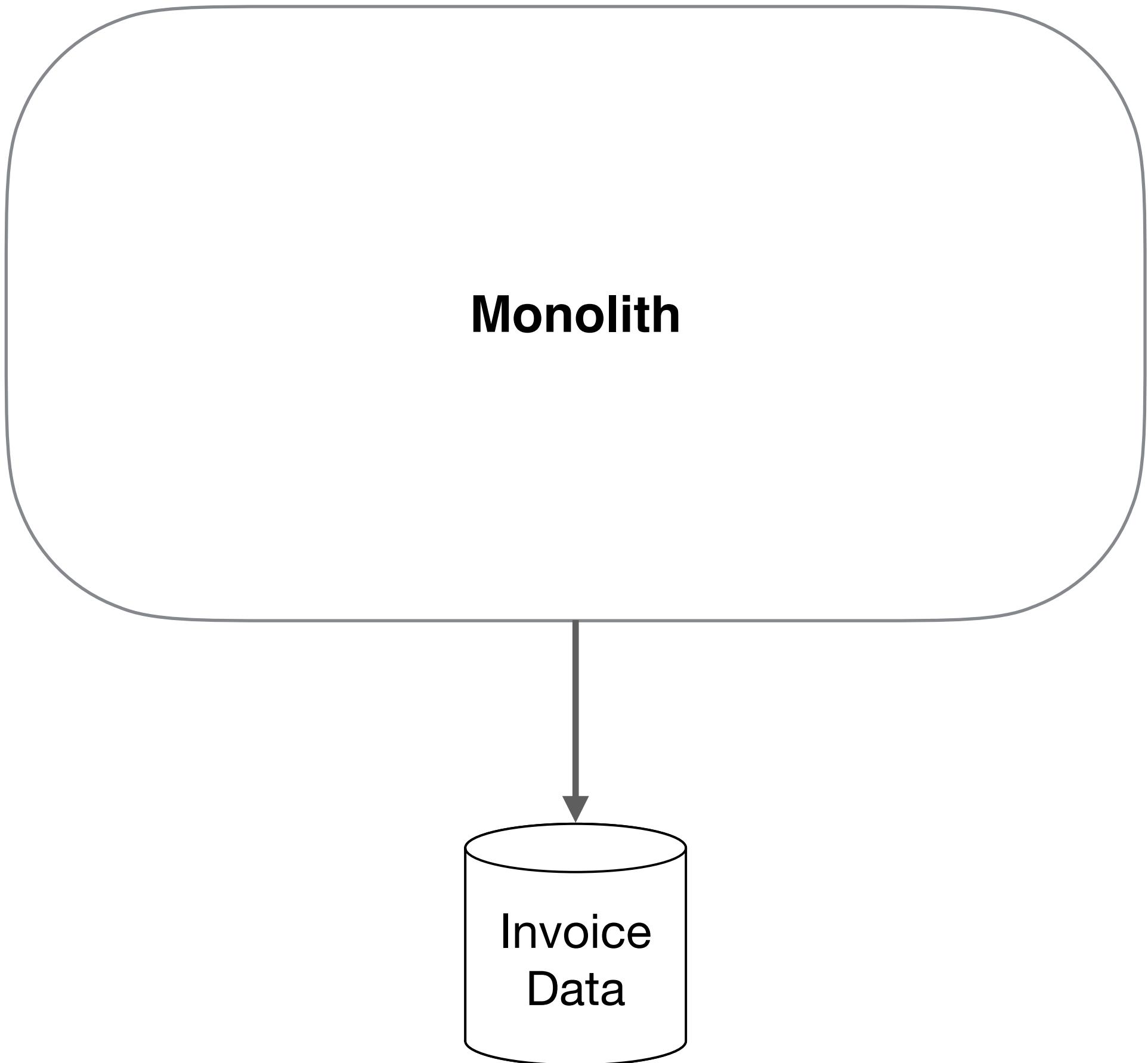
Decomposition Patterns

Extract data or app first?

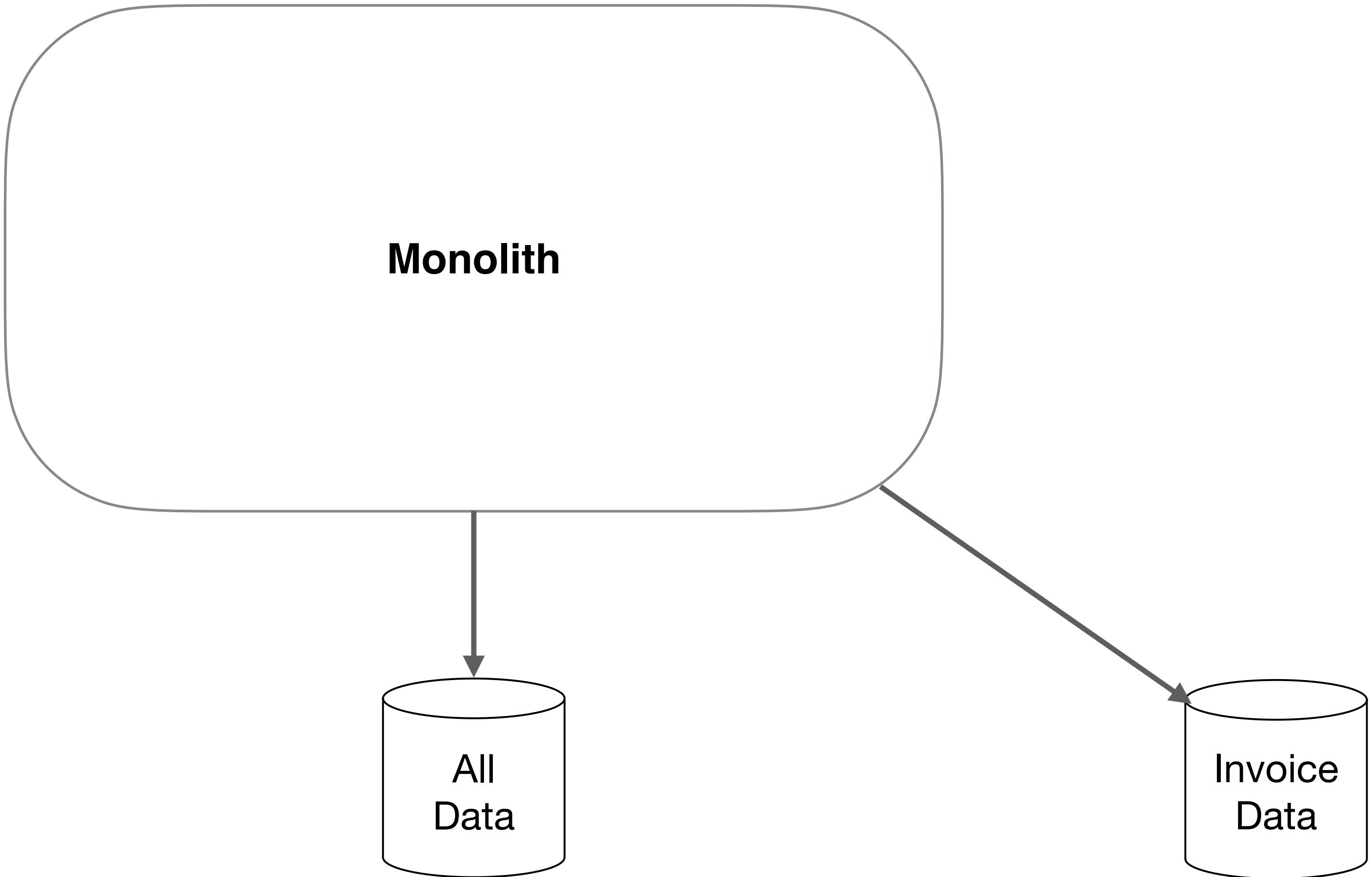
DATA FIRST?



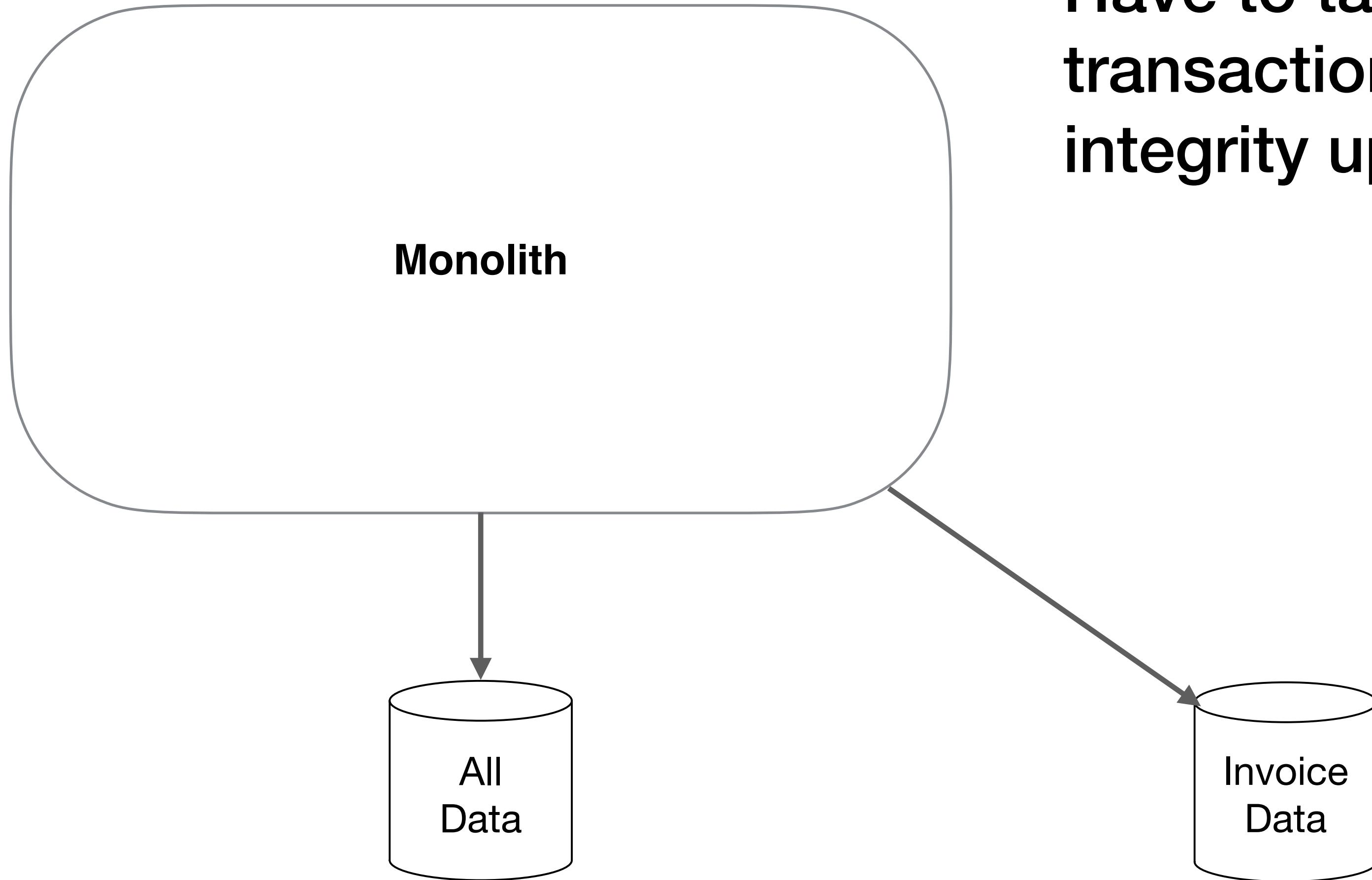
DATA FIRST?



DATA FIRST?

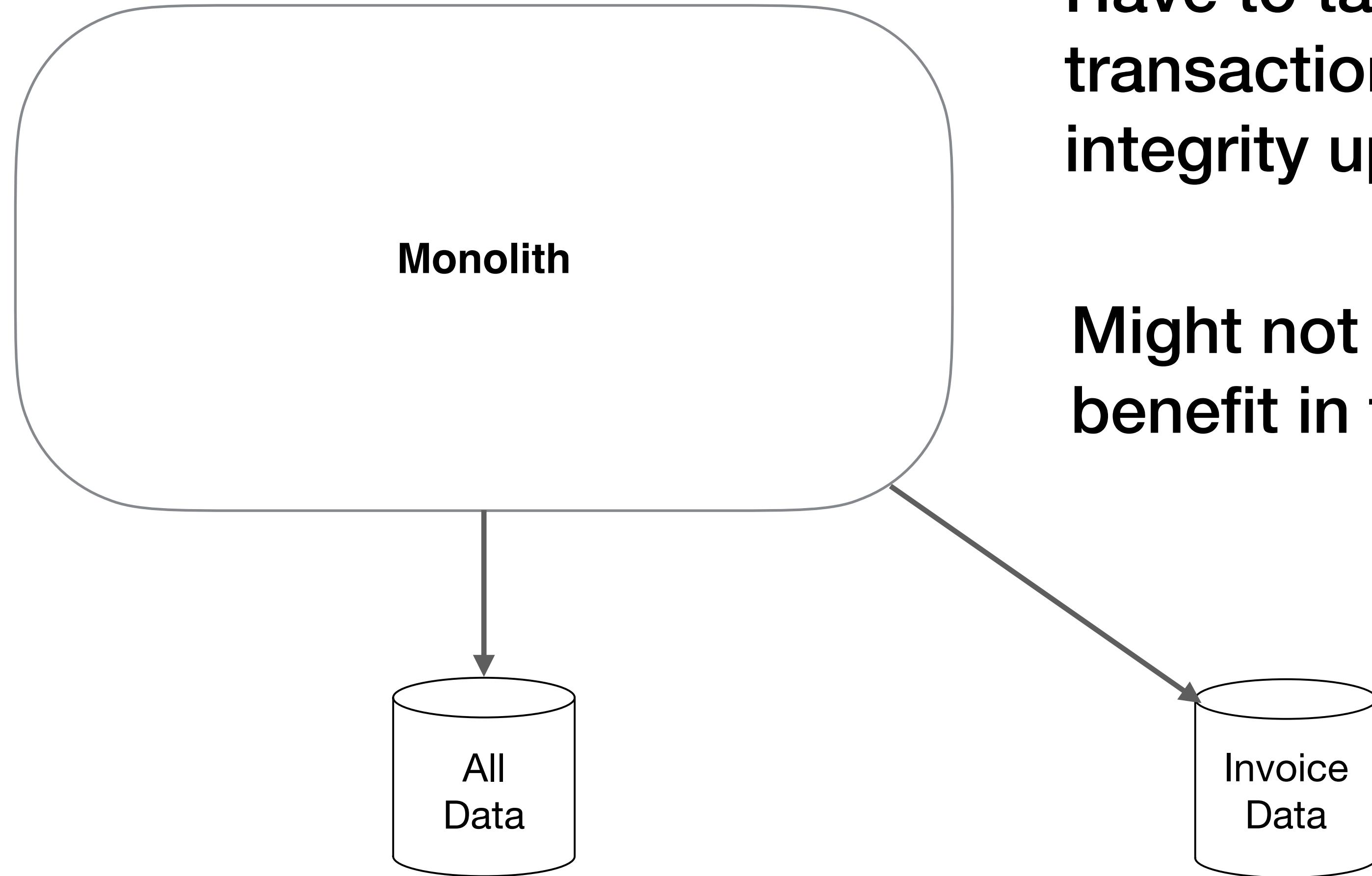


DATA FIRST?



Have to tackle issues around transactions, referential integrity up front

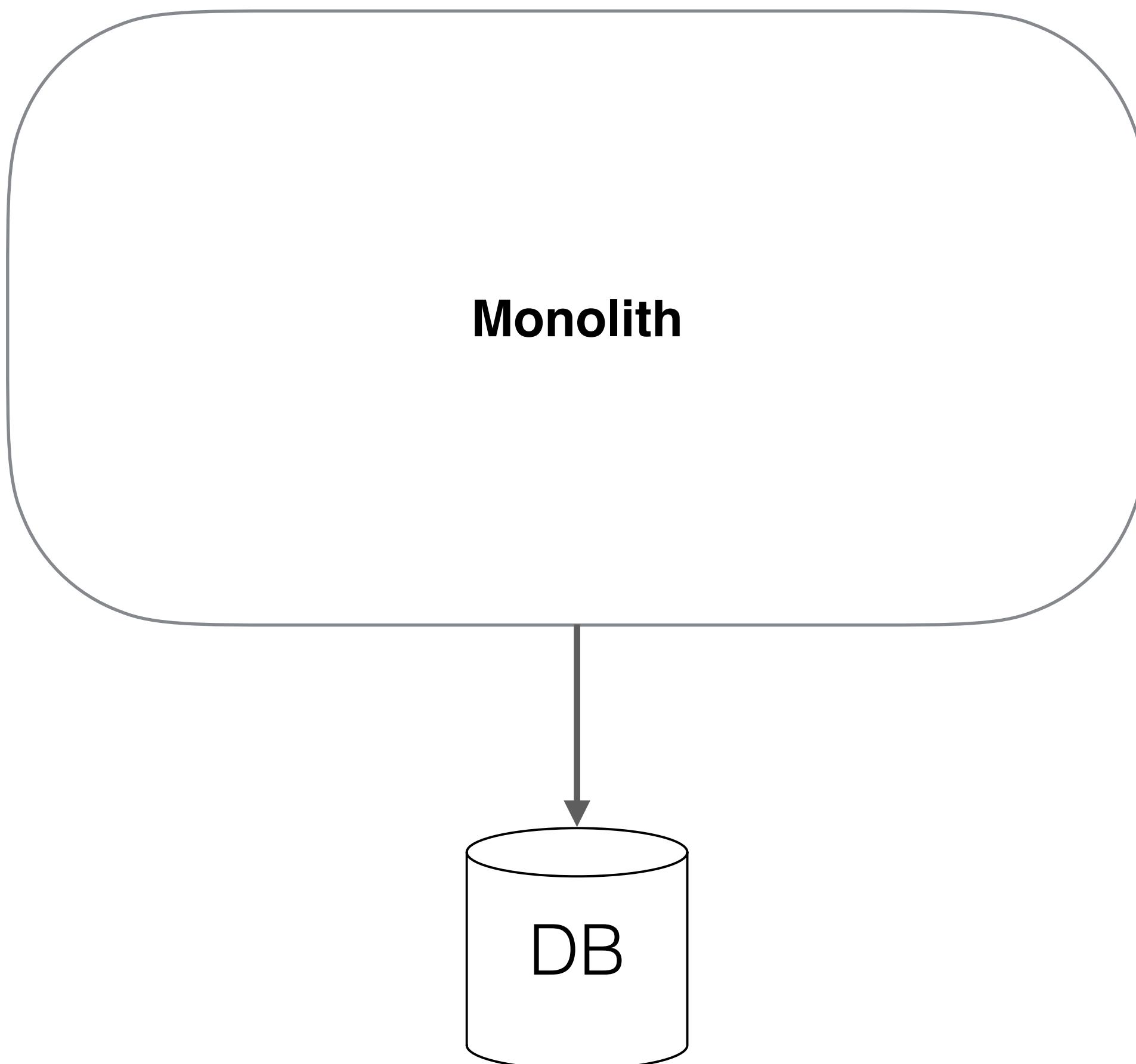
DATA FIRST?



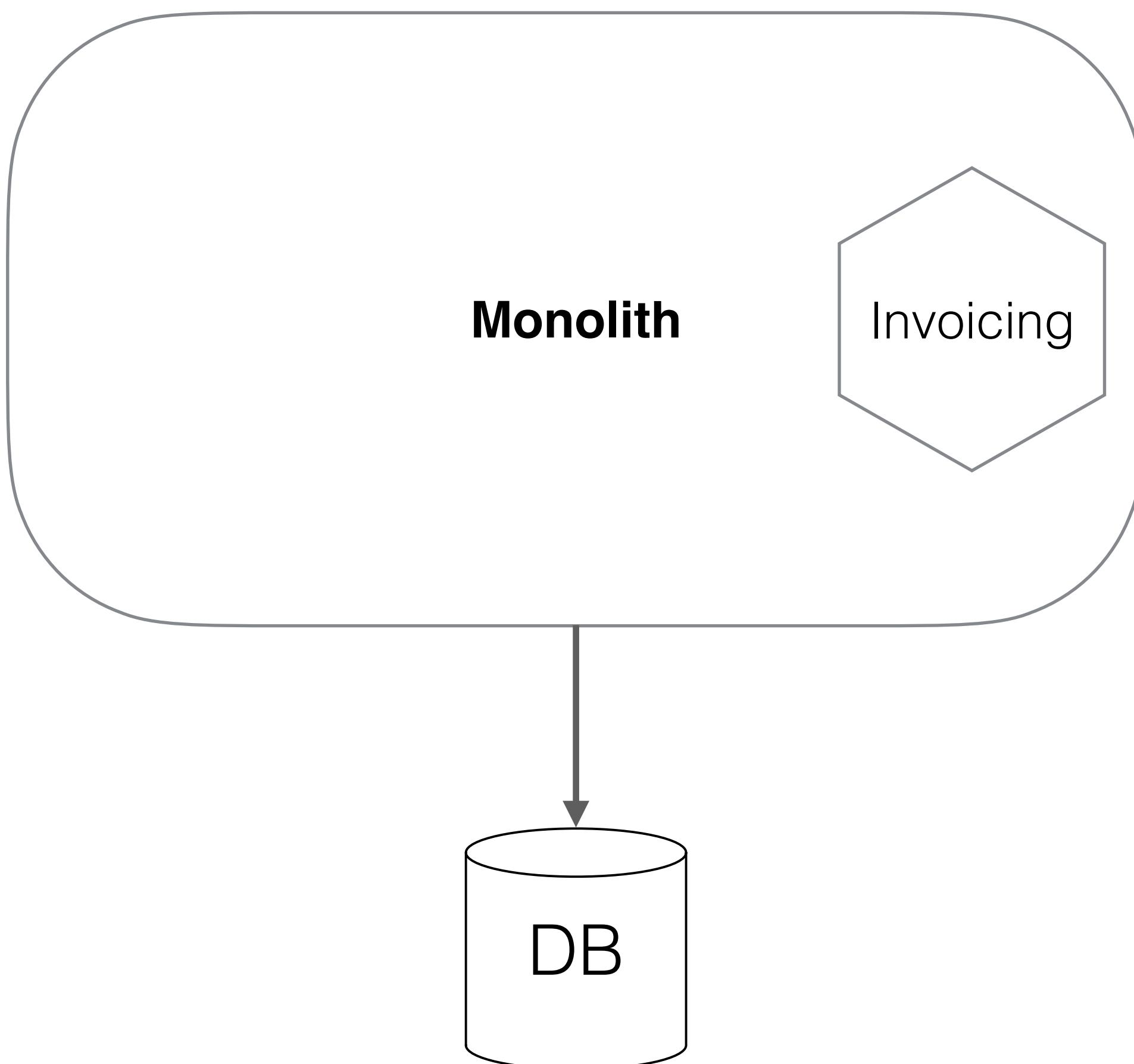
Have to tackle issues around transactions, referential integrity up front

Might not deliver much benefit in the short term

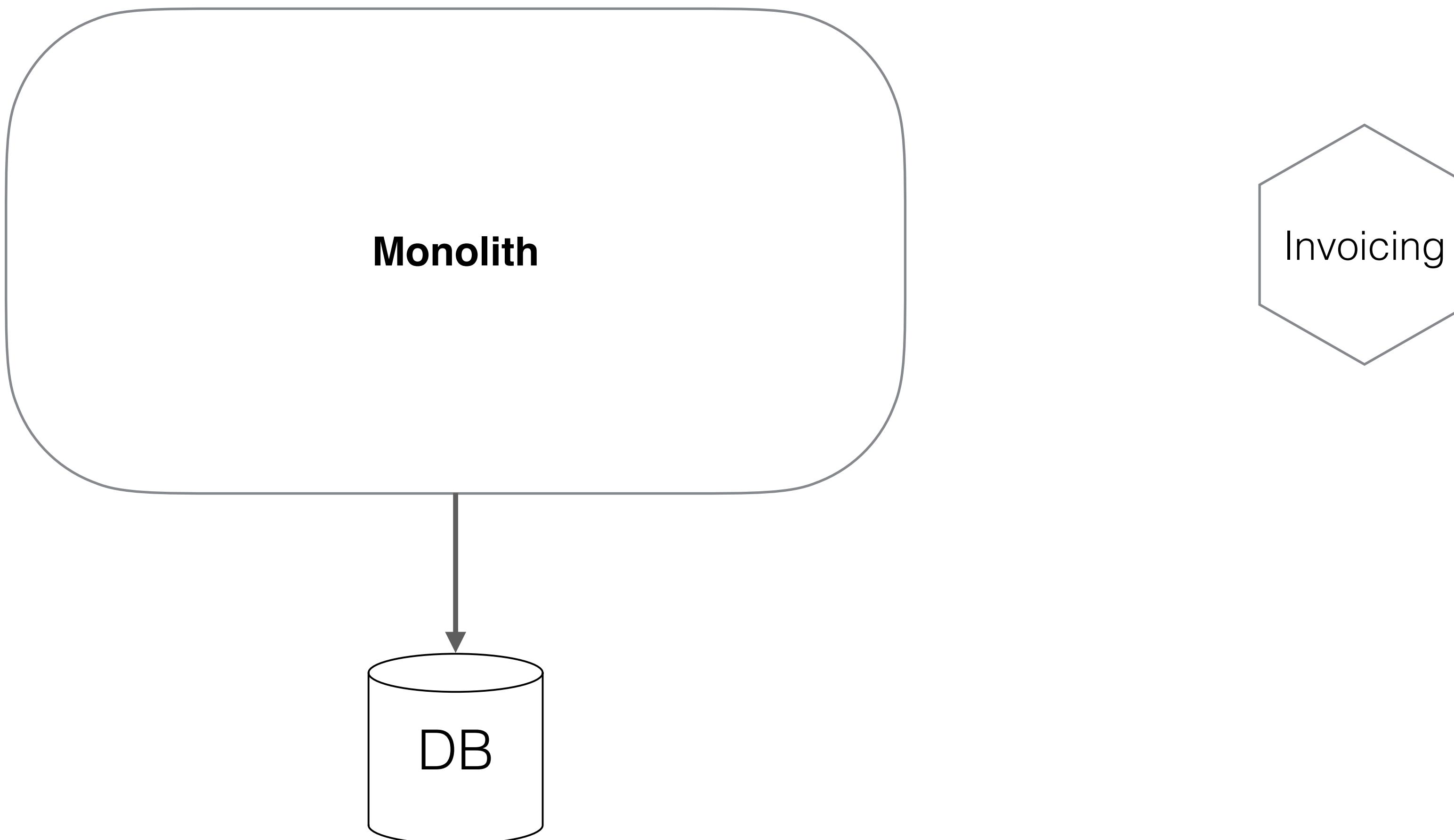
EXTRACTING CODE FIRST



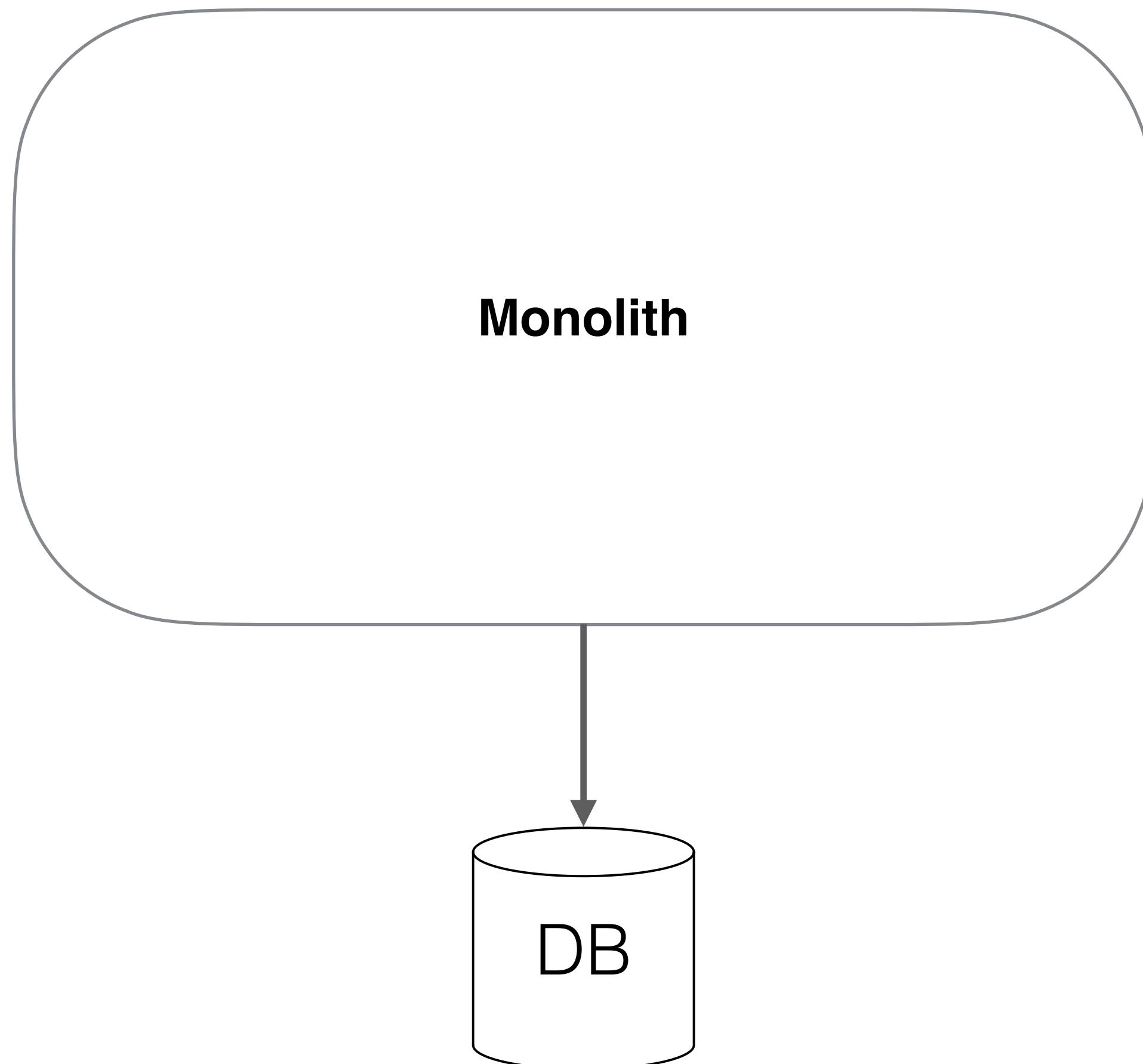
EXTRACTING CODE FIRST



EXTRACTING CODE FIRST



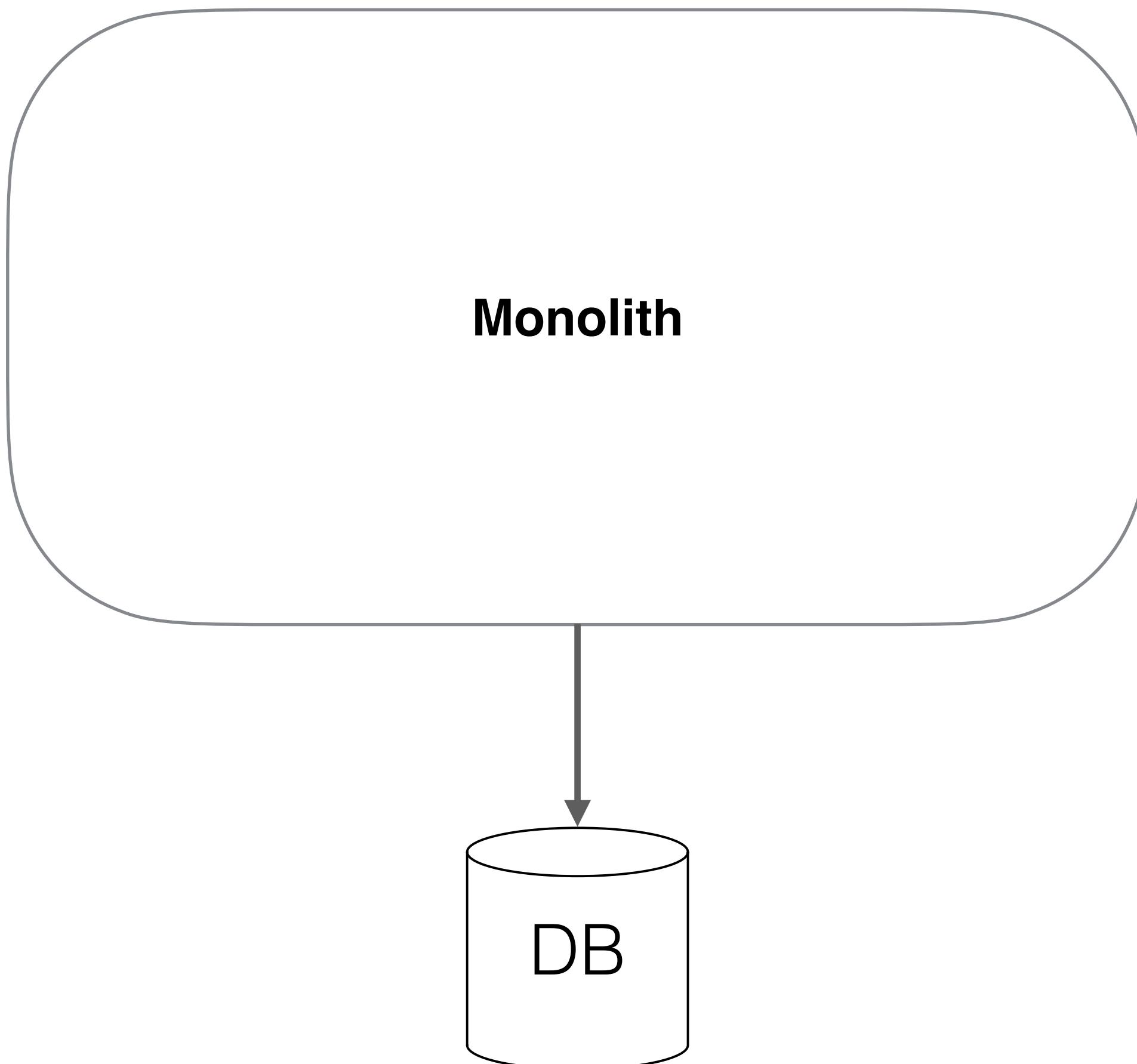
EXTRACTING CODE FIRST



Gains some short term benefit



EXTRACTING CODE FIRST

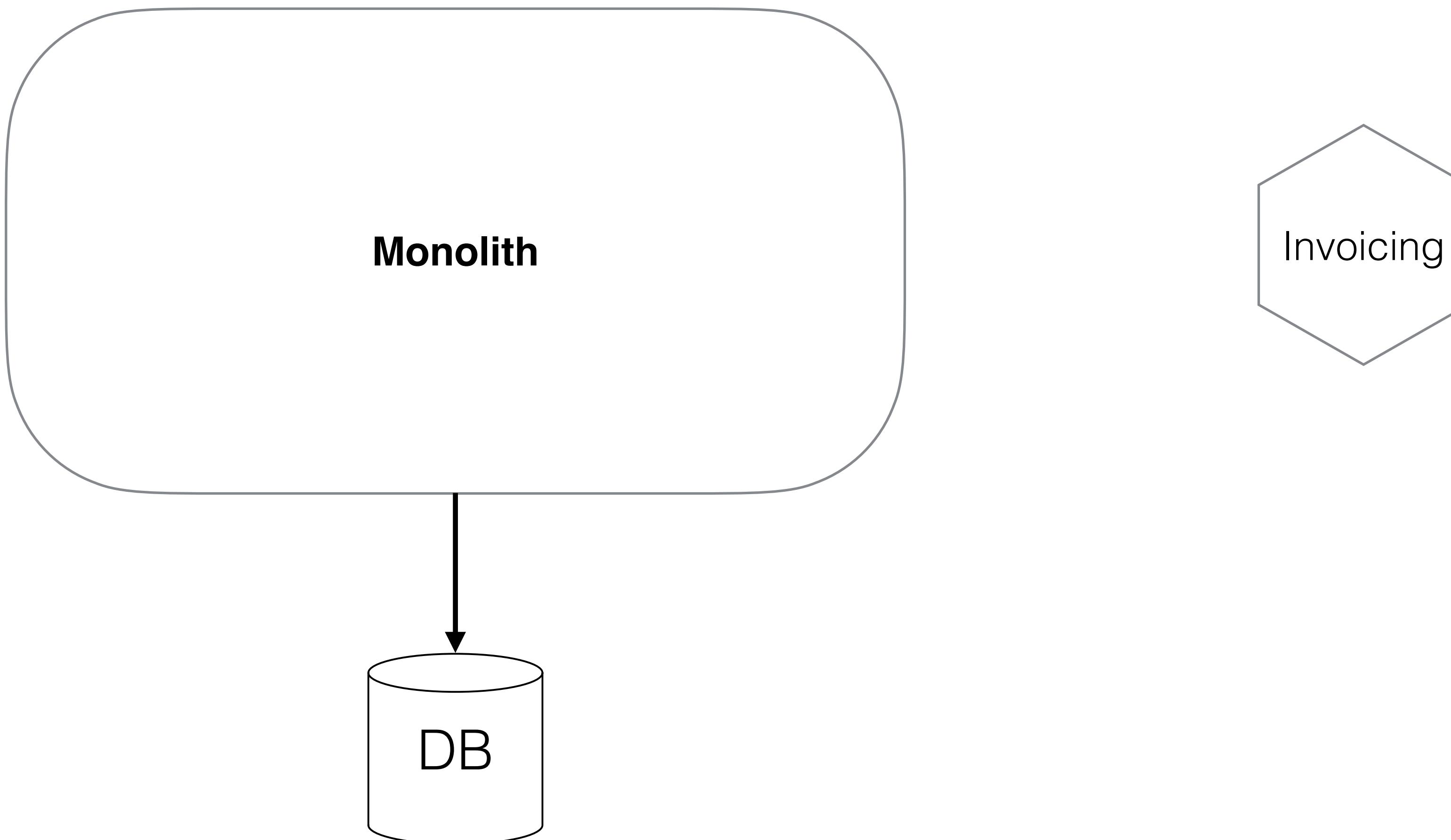


Gains some short term benefit

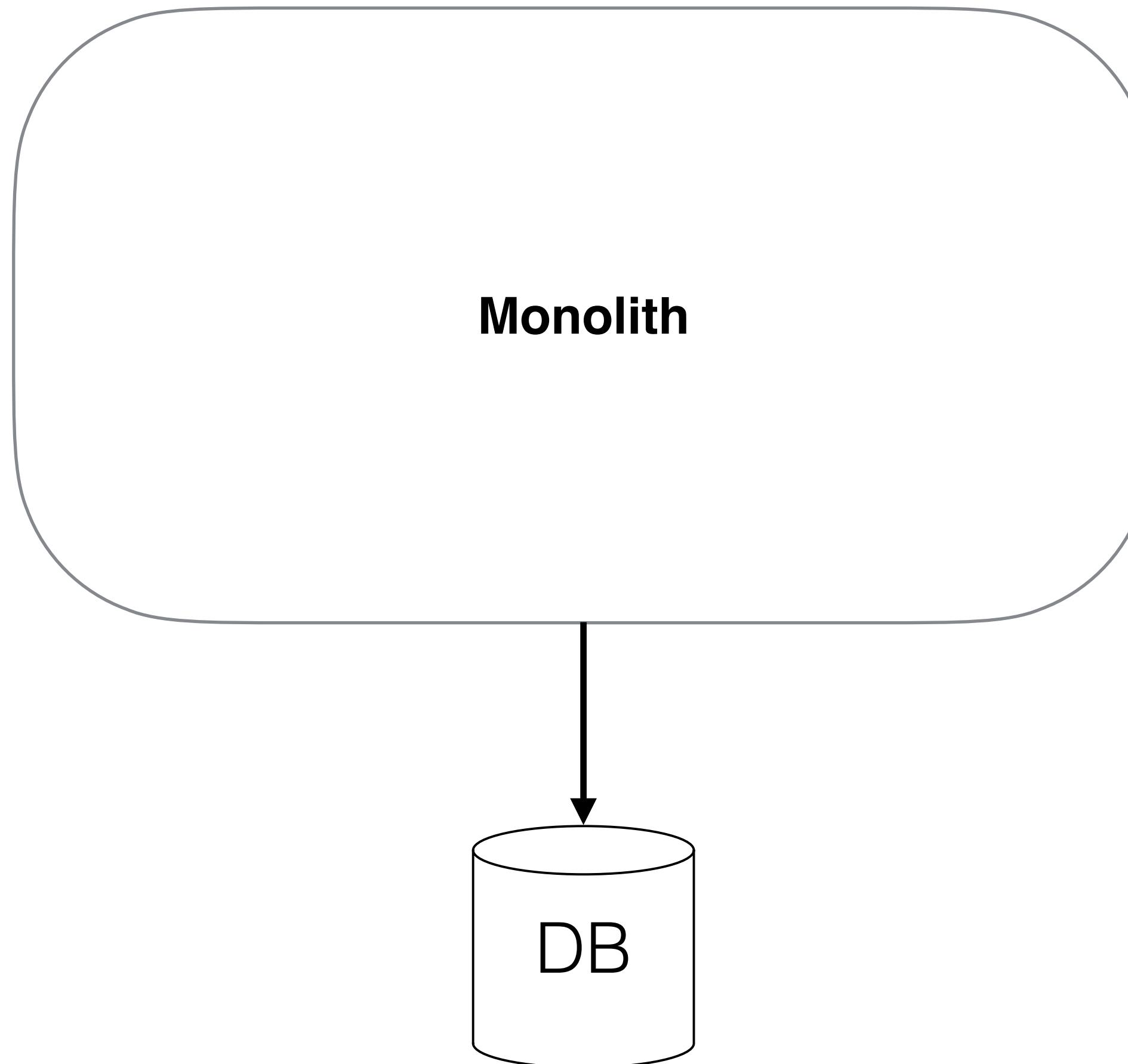


But what about data access?

USE THE MONOLITH DB DIRECTLY



USE THE MONOLITH DB DIRECTLY

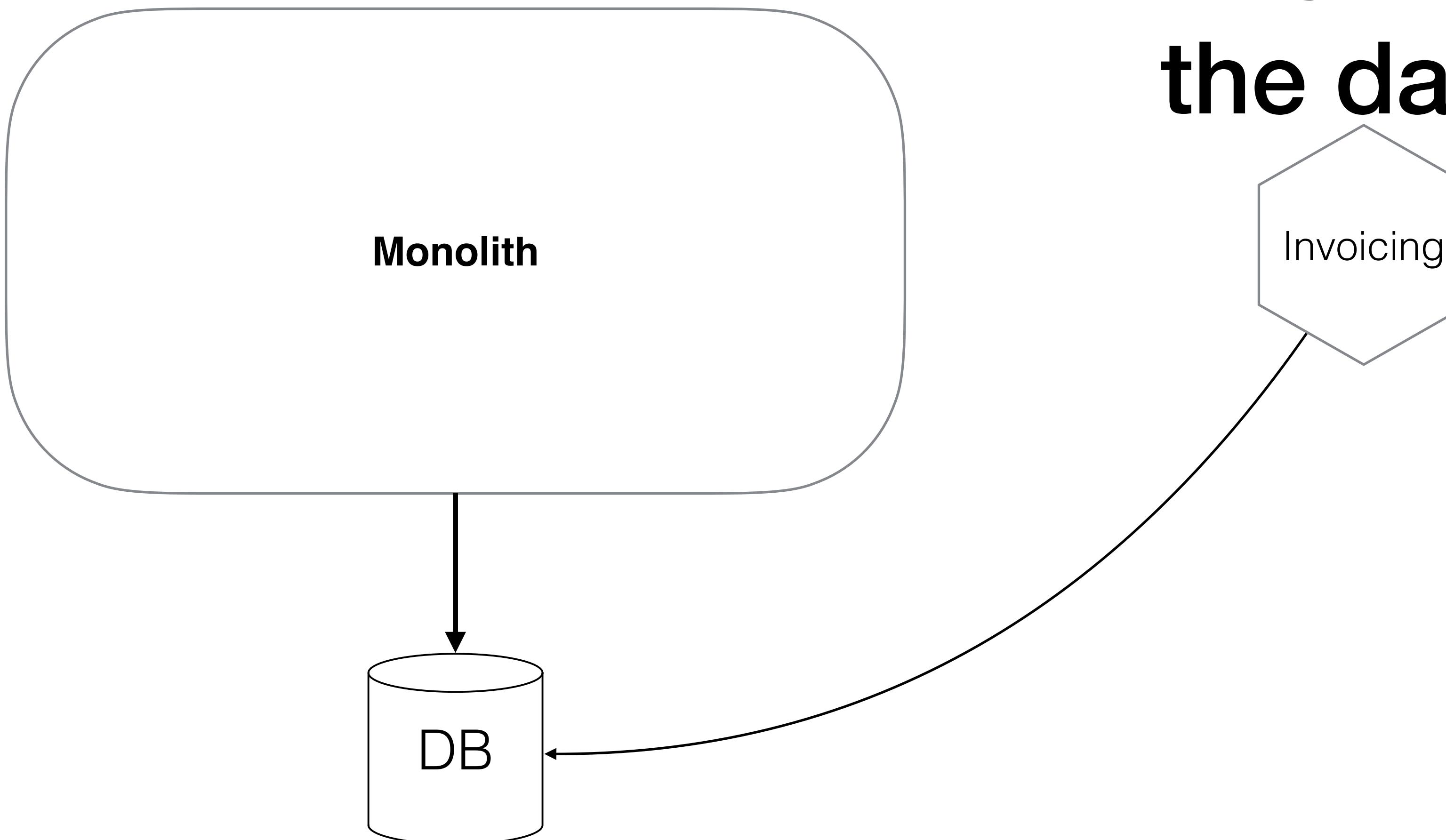


**Directly access
the data**

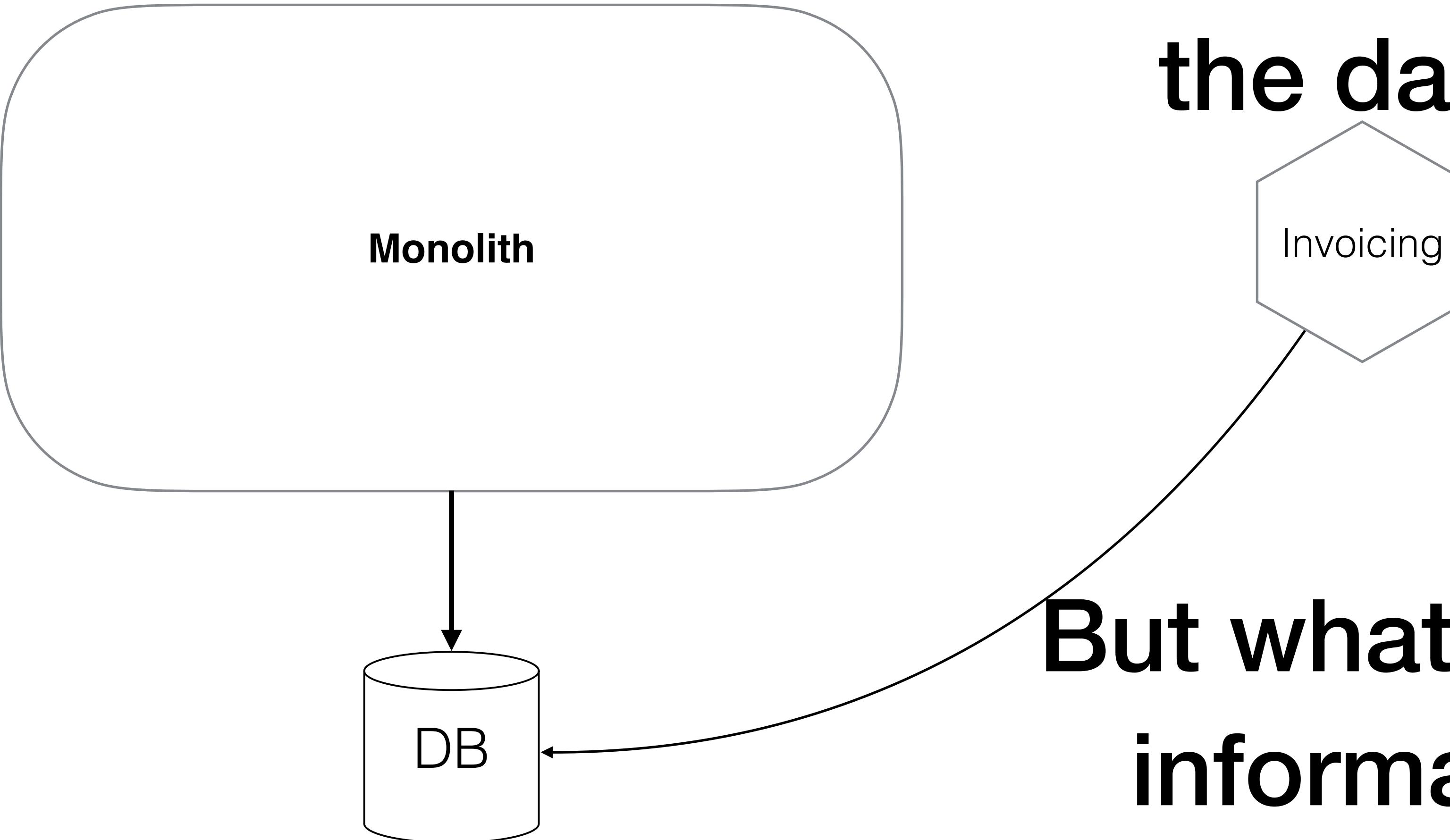


USE THE MONOLITH DB DIRECTLY

**Directly access
the data**



USE THE MONOLITH DB DIRECTLY



**Directly access
the data**

**But what about
information
hiding?**

**OK for a short period of time as
part of a transition...**

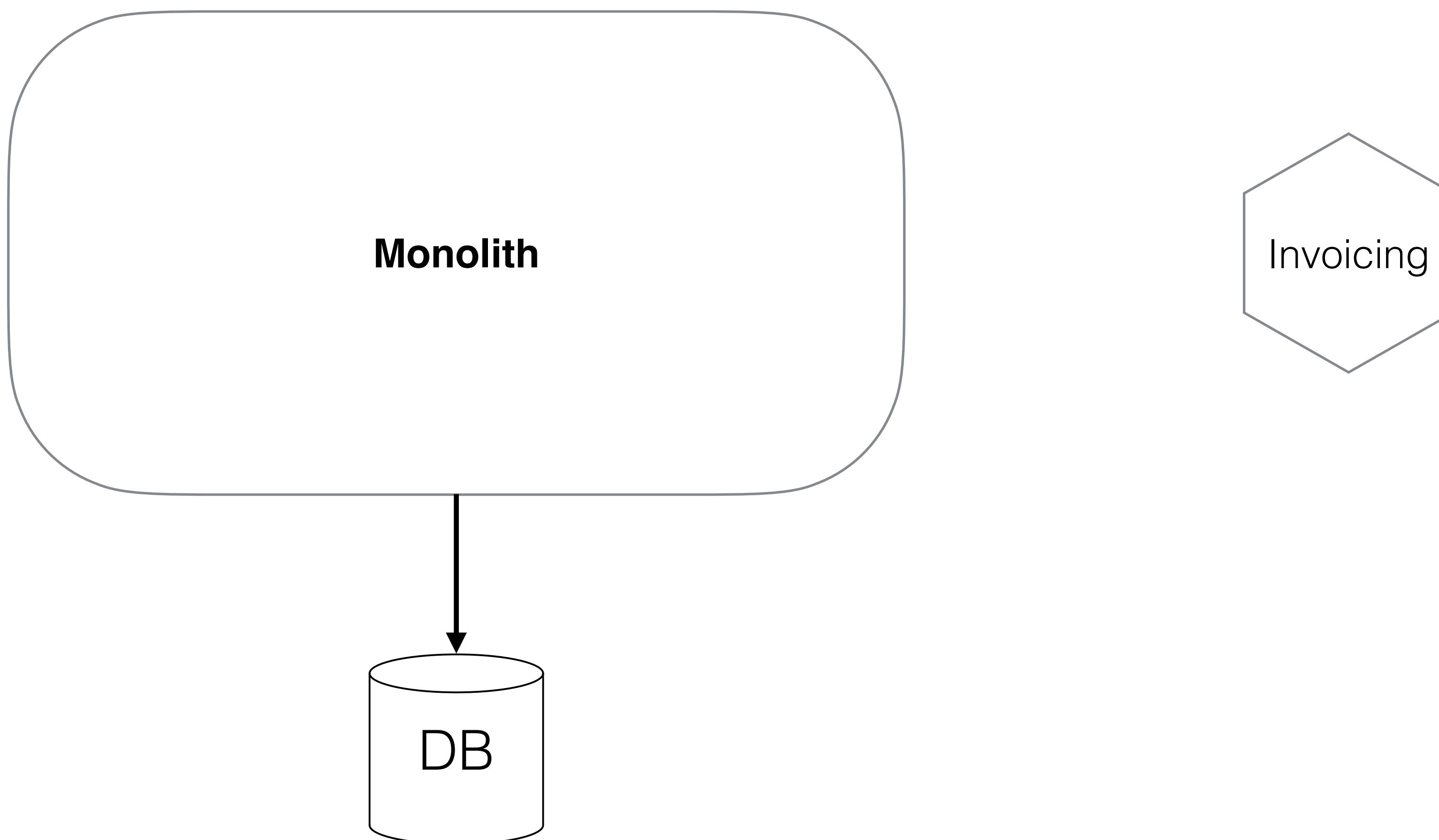
OK for a short period of time as
part of a transition...

...just remember that the microservice
decomposition isn't finished until you
remove the use of the shared DB

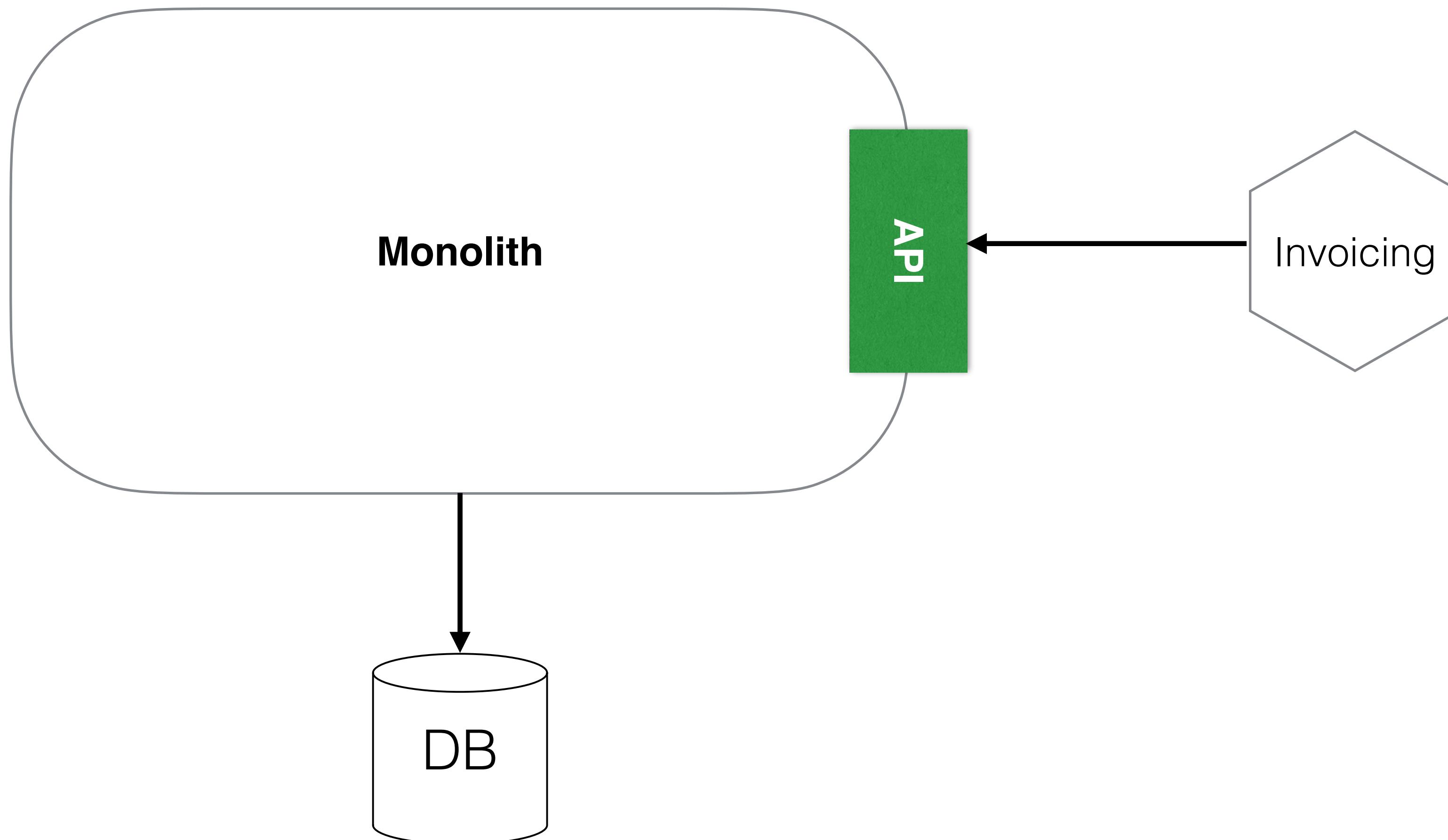
@samnewman

**So if we extract the application code first,
how should we access the data?**

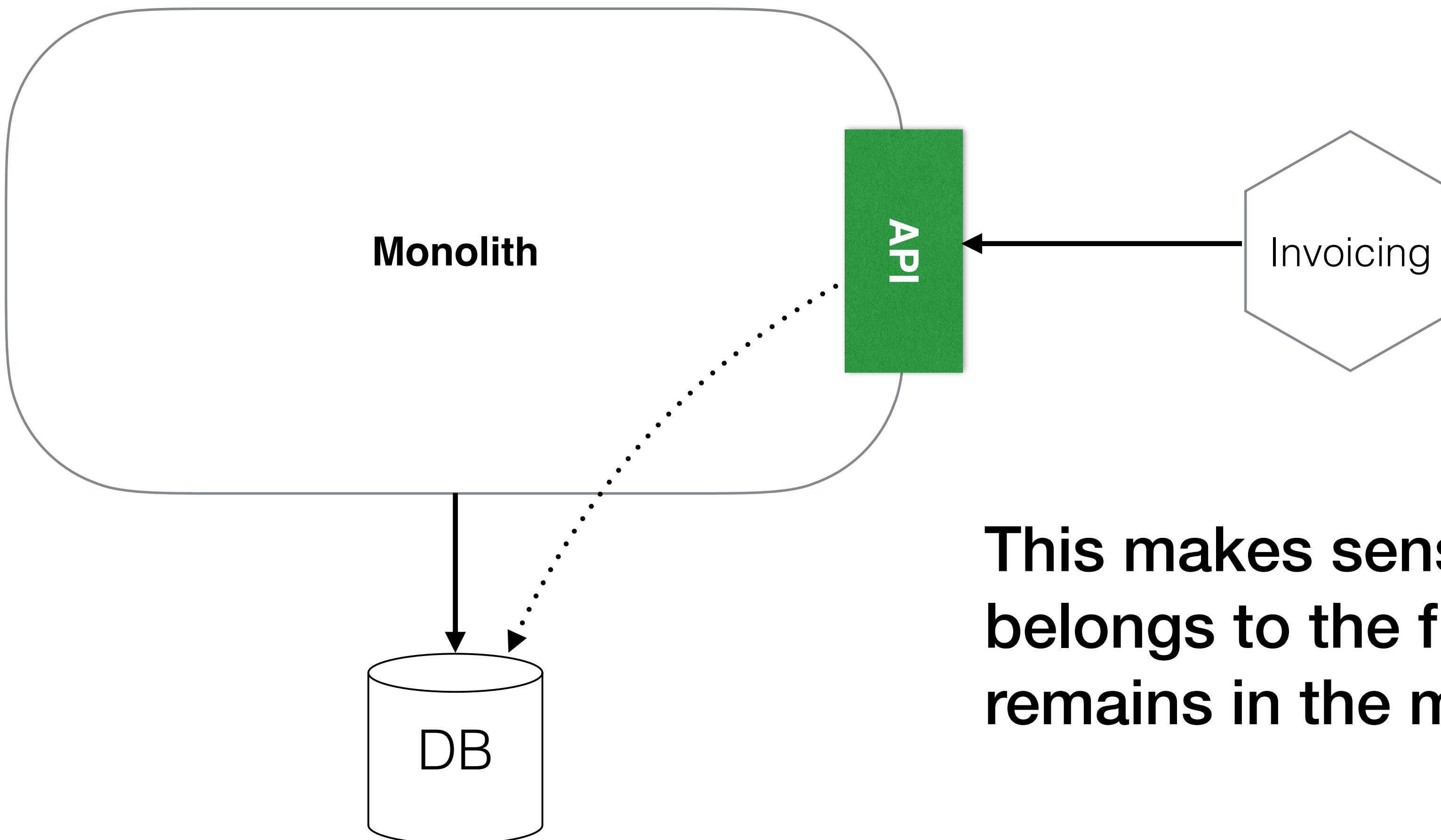
EXPOSE THE DATA VIA THE MONOLITH



EXPOSE THE DATA VIA THE MONOLITH

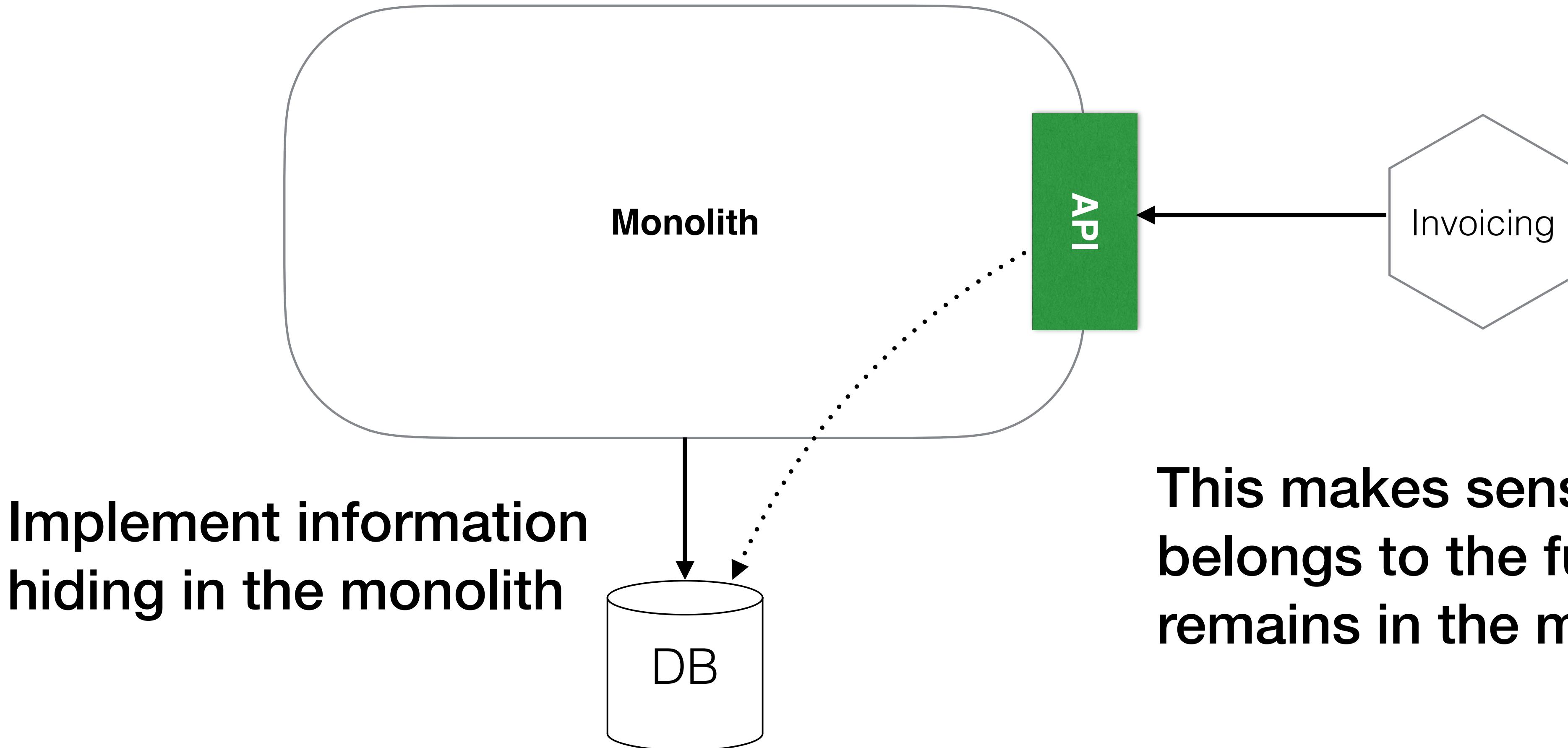


EXPOSE THE DATA VIA THE MONOLITH

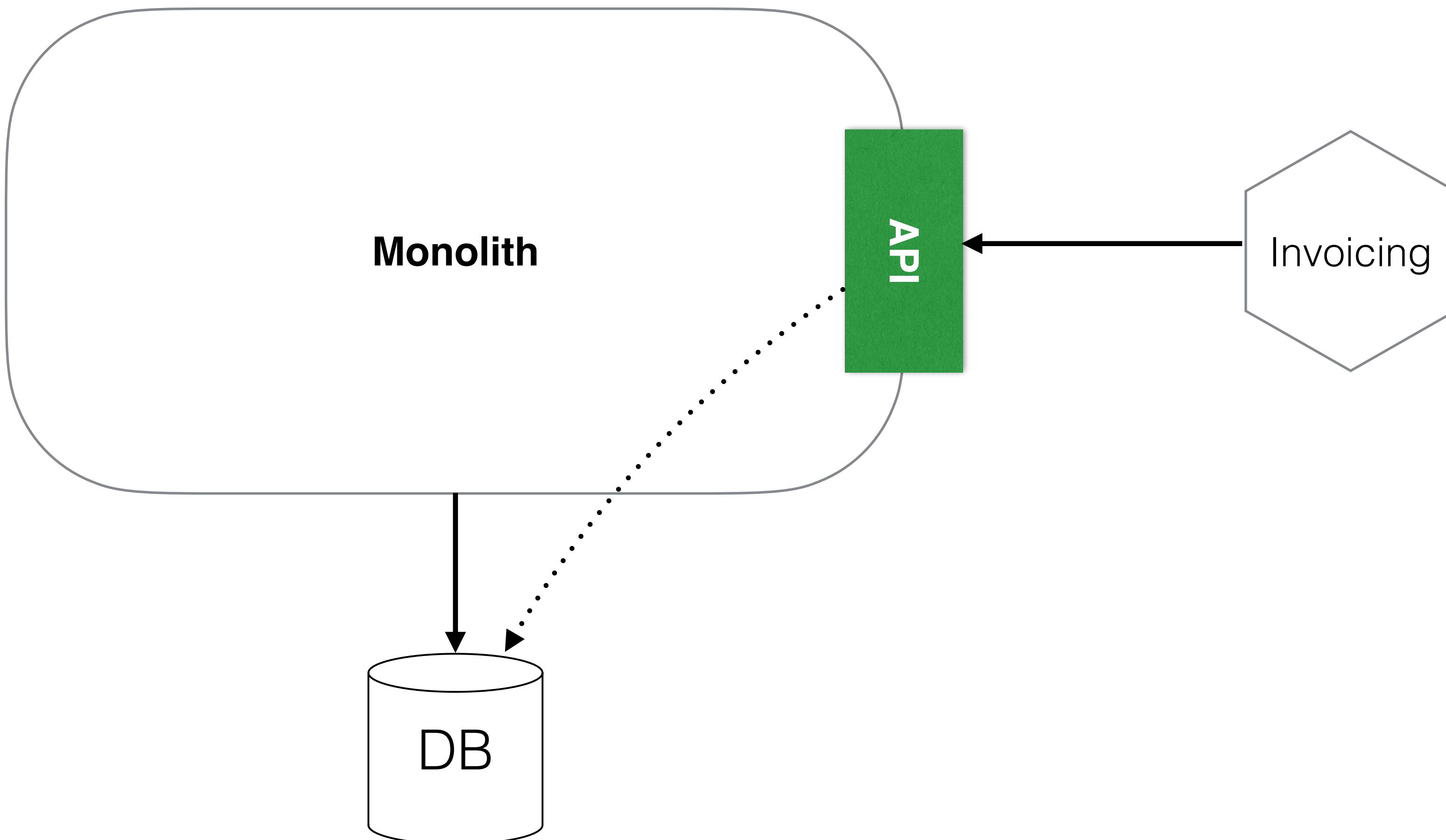


This makes sense if the data belongs to the functionality that remains in the monolith

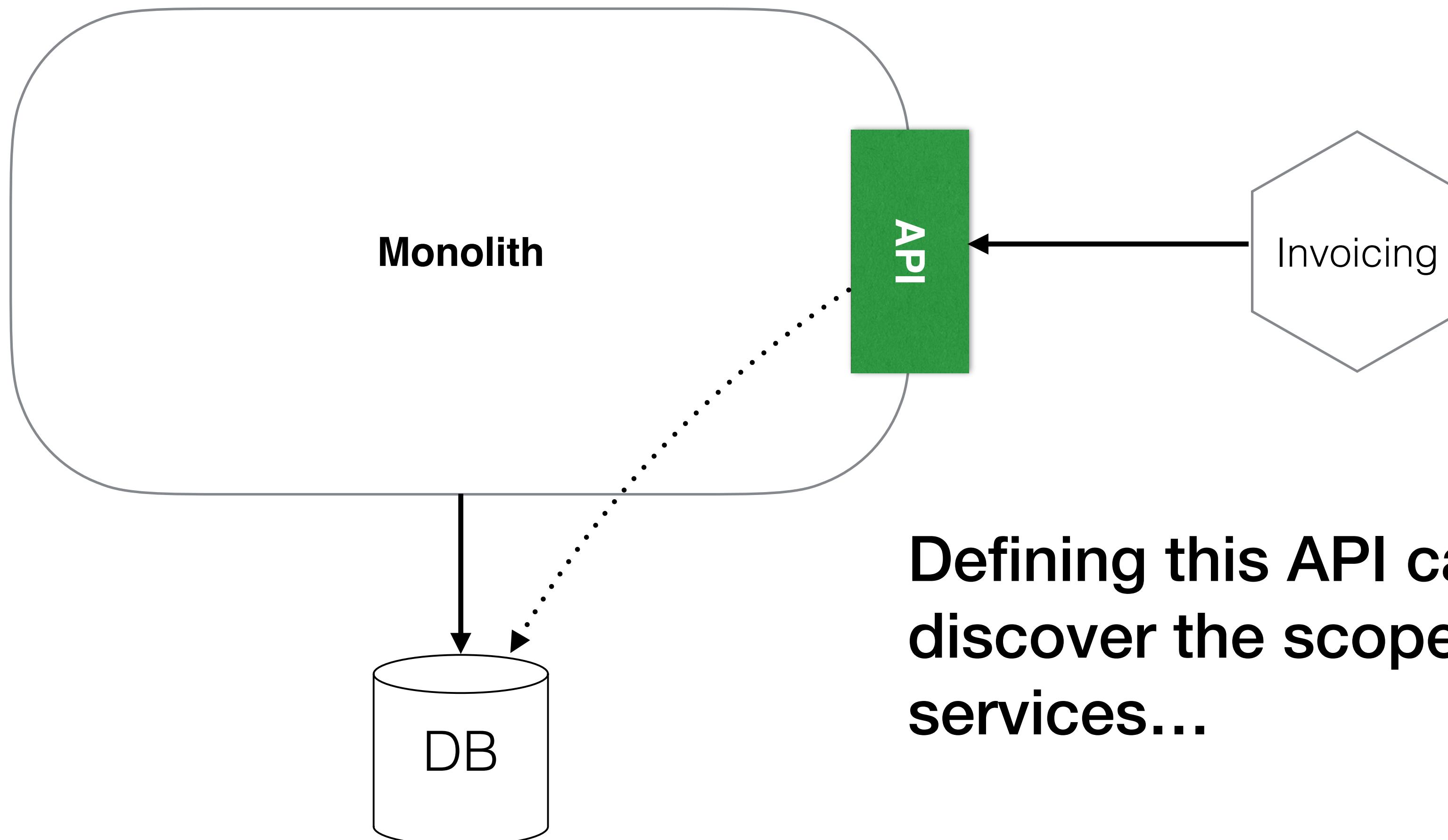
EXPOSE THE DATA VIA THE MONOLITH



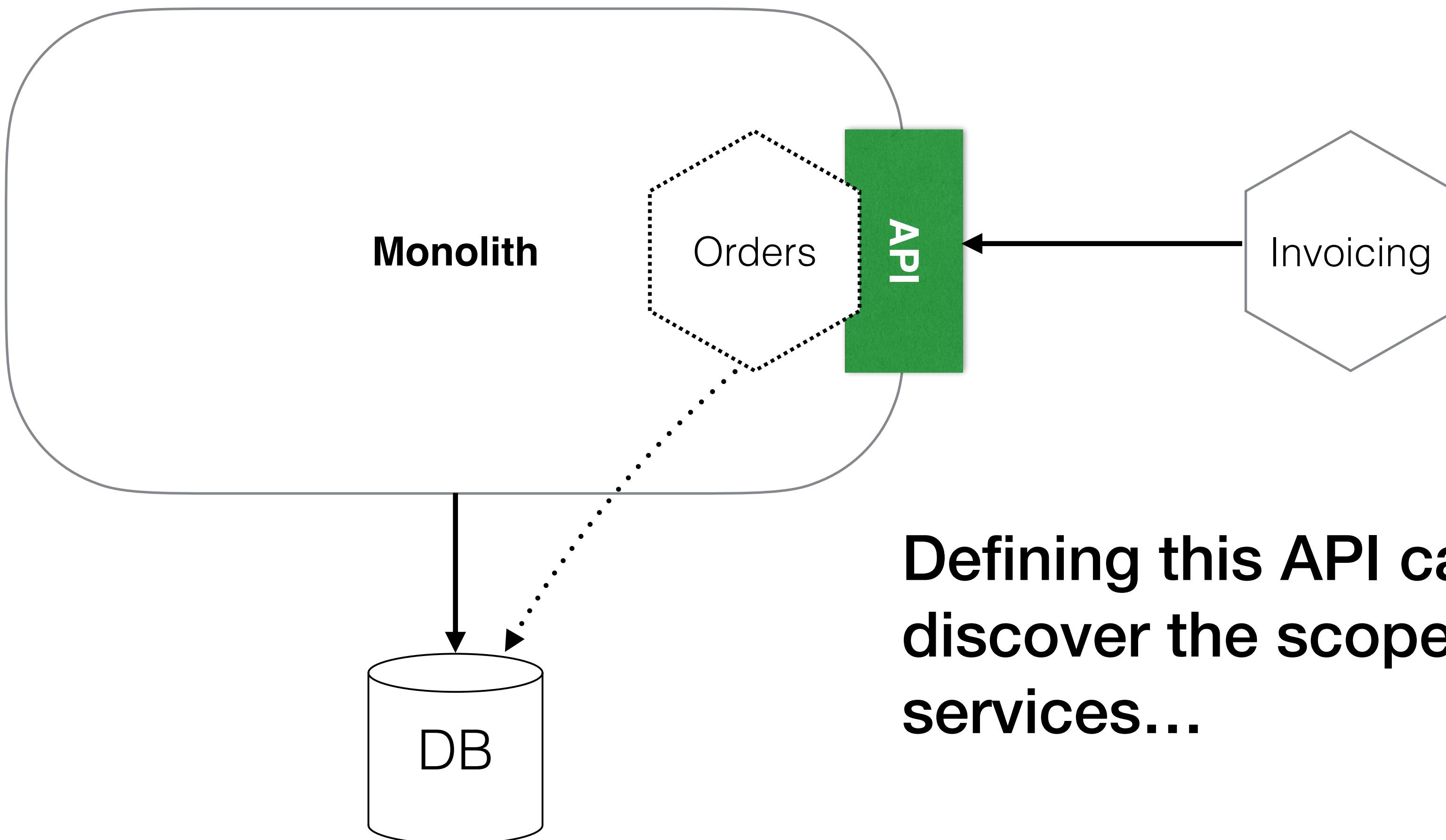
IDENTIFY FURTHER POTENTIAL MICROSERVICES



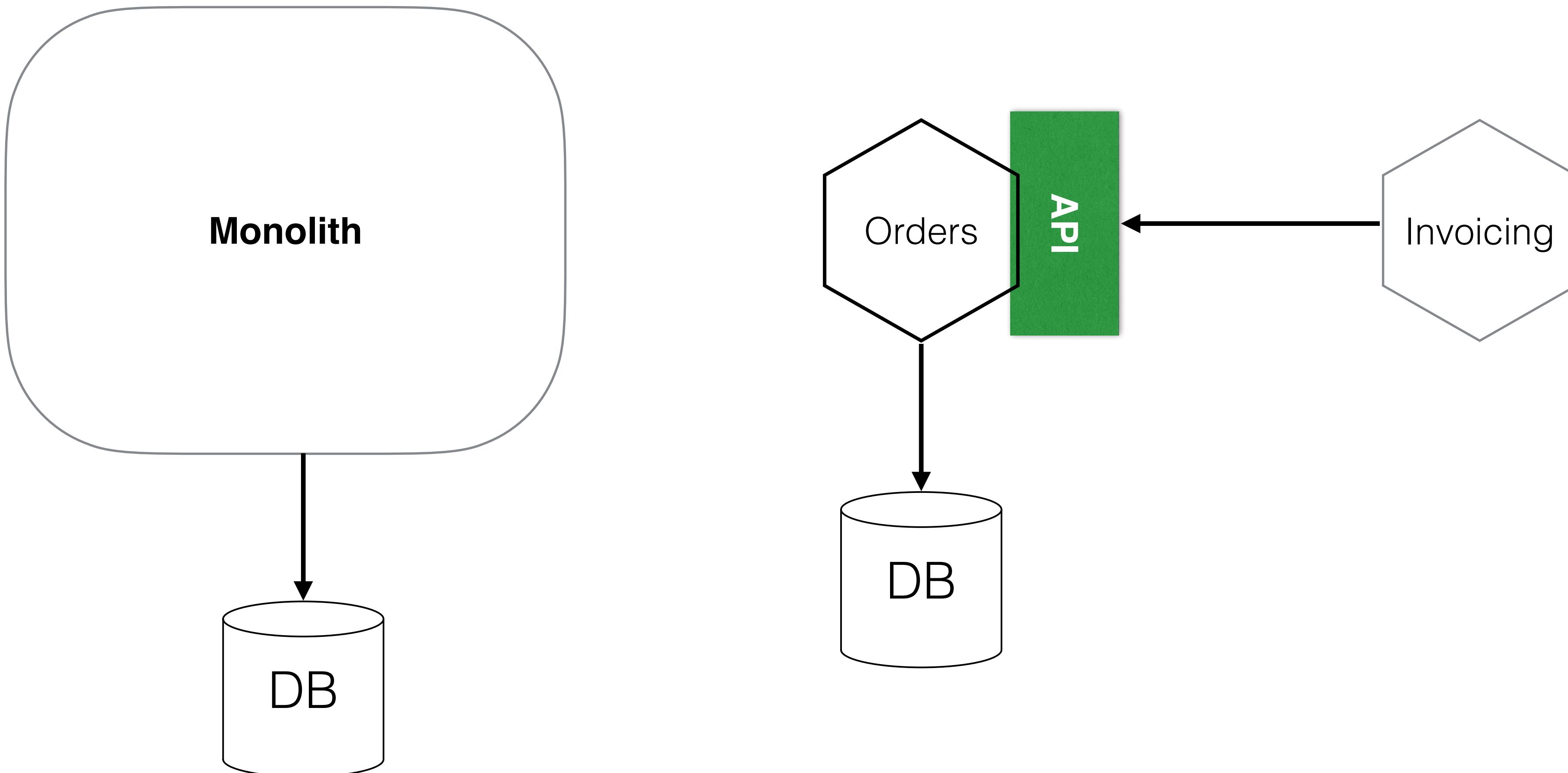
IDENTIFY FURTHER POTENTIAL MICROSERVICES



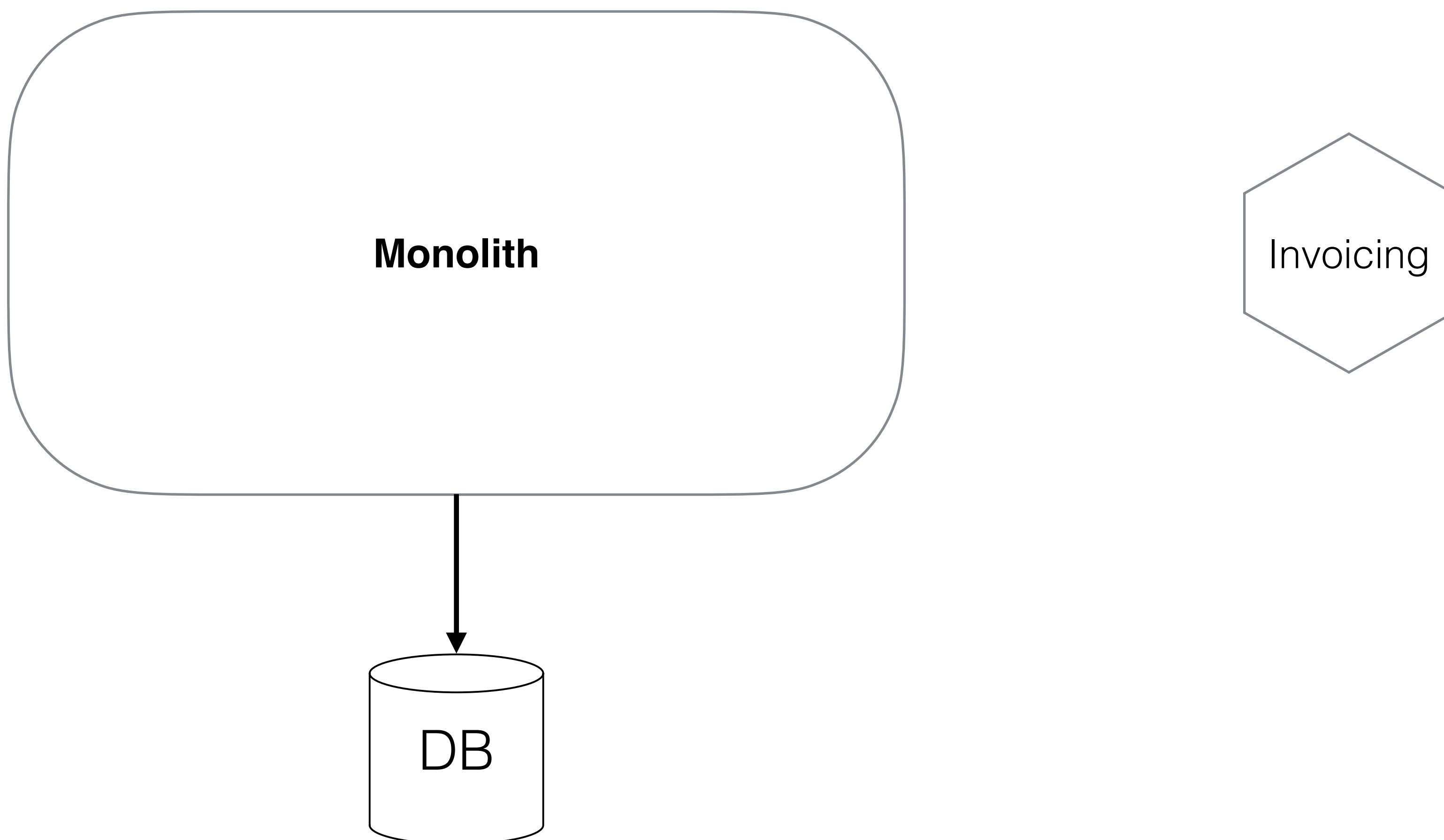
IDENTIFY FURTHER POTENTIAL MICROSERVICES



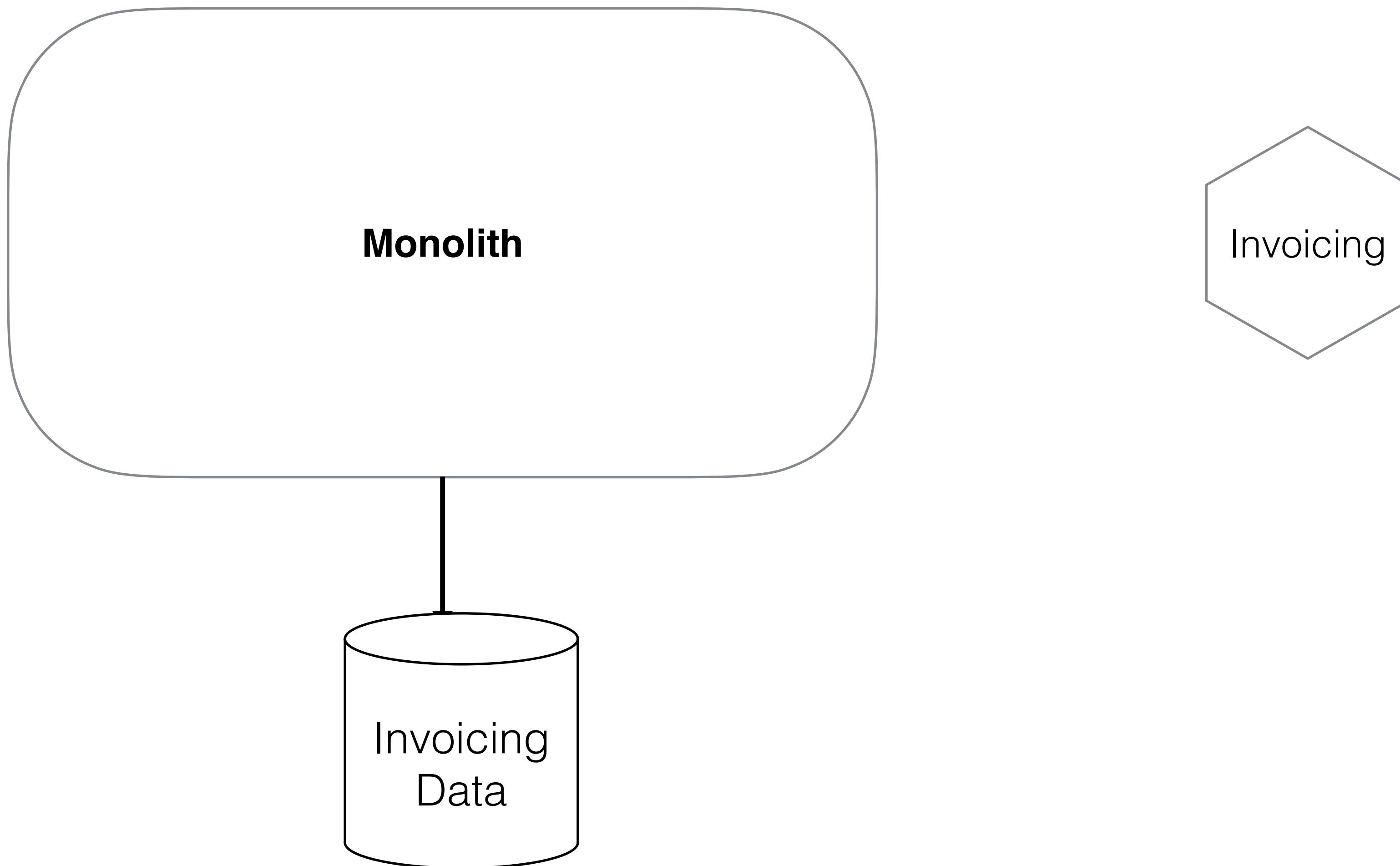
FINDING NEW MICROSERVICES?



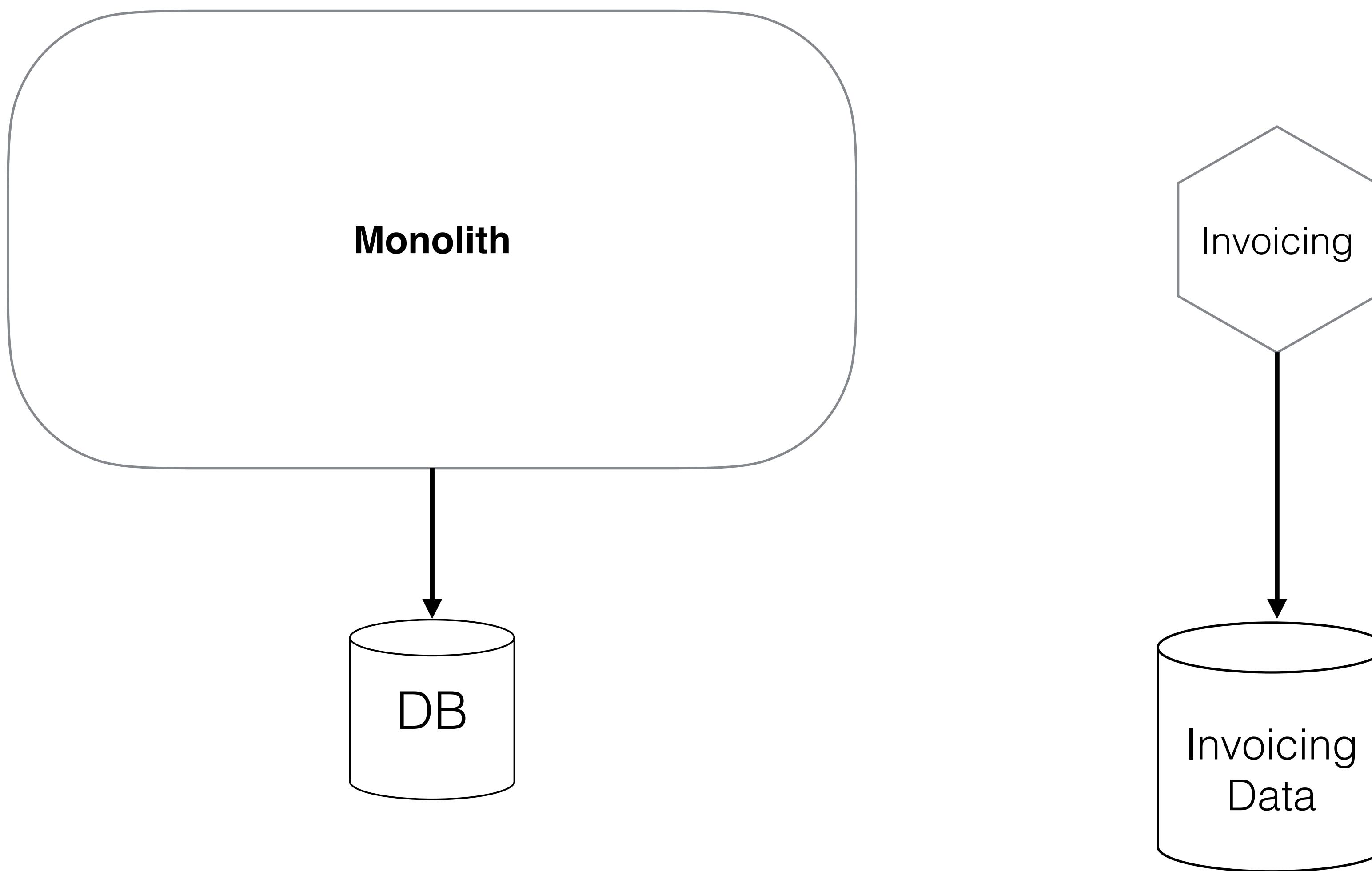
MOVE THE DATA!



MOVE THE DATA!

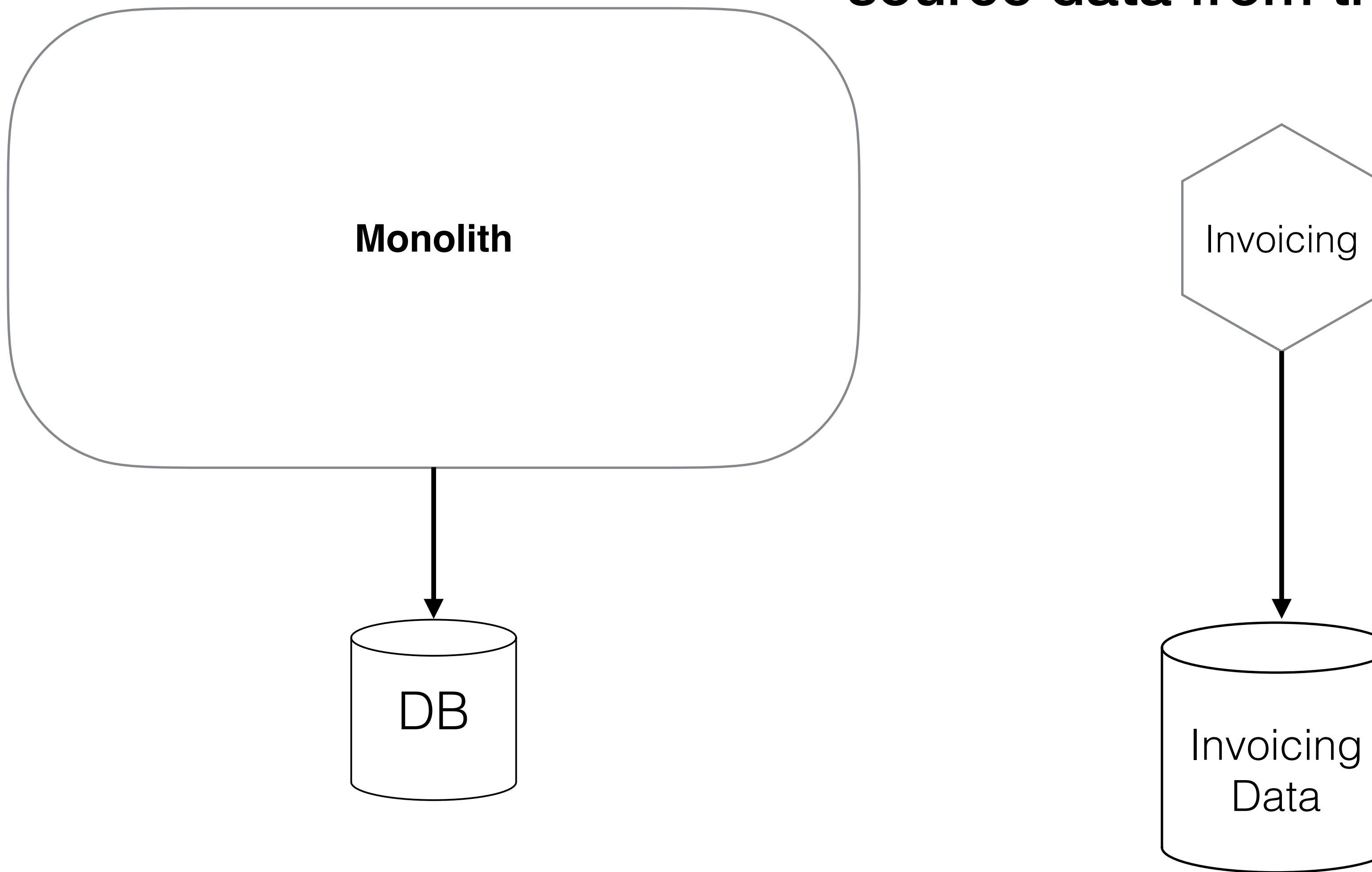


MOVE THE DATA!



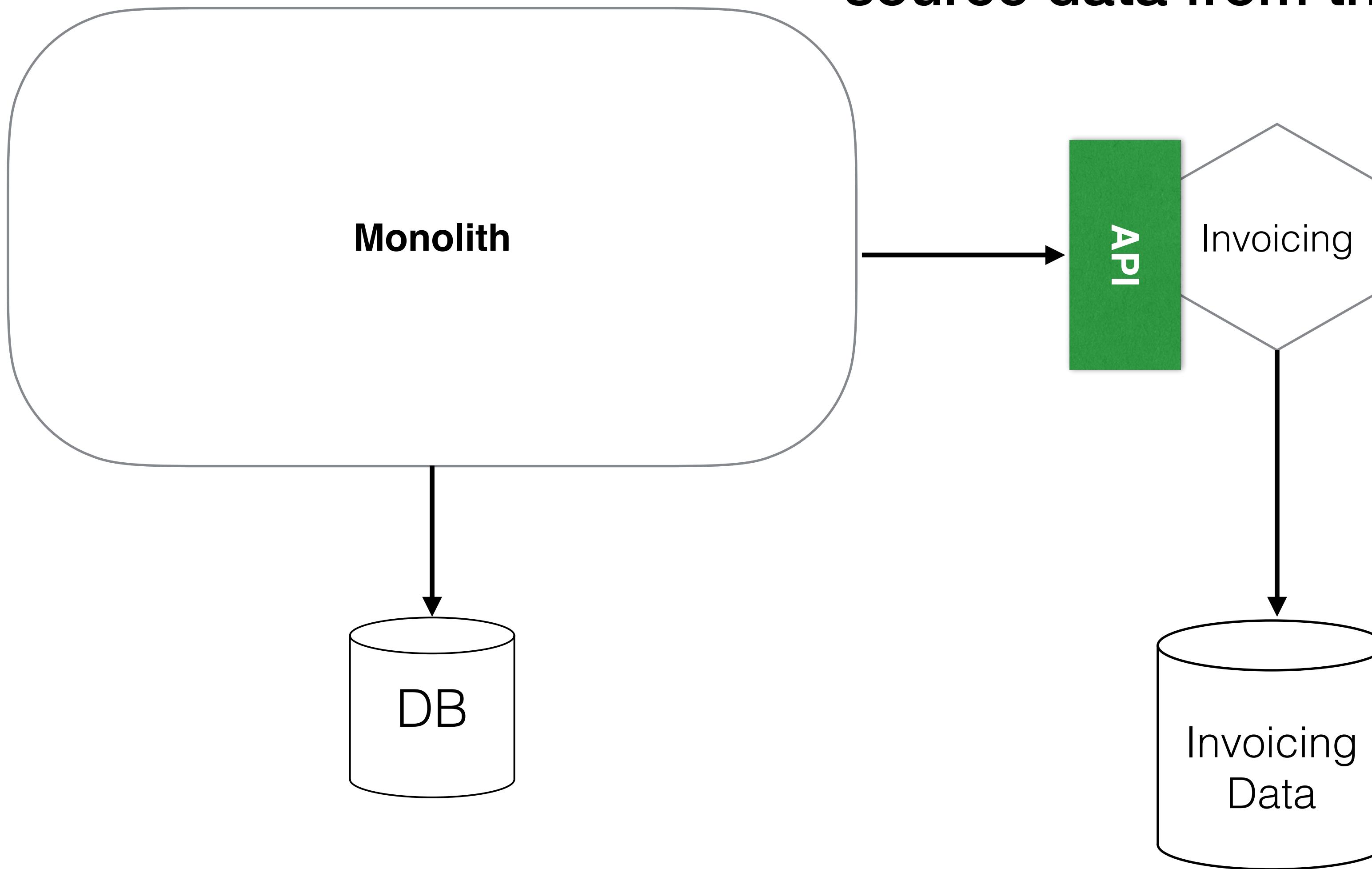
MOVE THE DATA!

Monolith needs to be changed to source data from the new service



MOVE THE DATA!

Monolith needs to be changed to source data from the new service



Let the pain and suffering begin

AGENDA

Introduction

Shared Data Patterns

Migration Order

Decomposition Patterns

AGENDA

Introduction

Shared Data Patterns

Migration Order

Decomposition Patterns

DATA REFACTORING TOOLS

GET STARTED DOWNLOAD / PRICING DOCUMENTATION BLOG



Flyway™
by Redgate

Version control for your database.
Robust schema evolution across all your environments.
With ease, pleasure and plain SQL.

[Get Started with Flyway 6.4.4 →](#)

Download - Release Notes - Apache v2 license
Works on:      

Stay updated with our newsletter

<https://flywaydb.org/>

@samnewman

DATA REFACTORING TOOLS

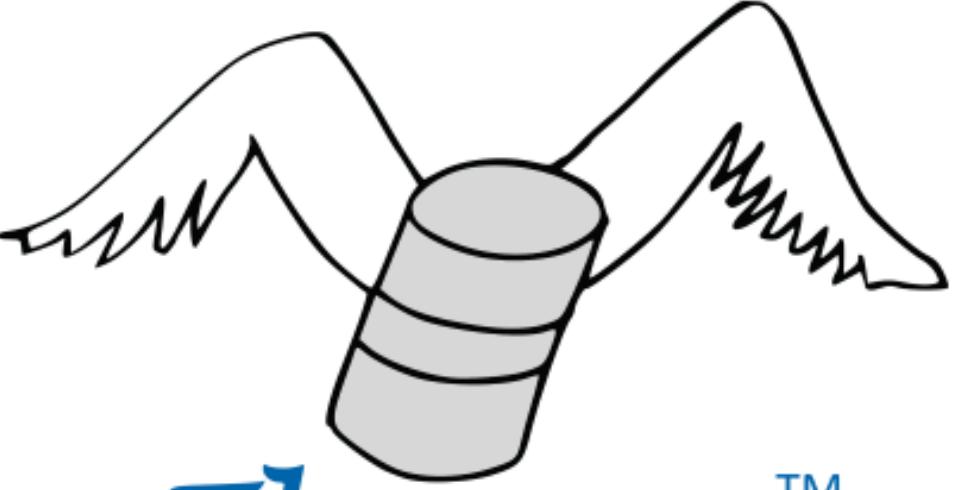
The screenshot shows the homepage of the Flyway website. At the top, there is a navigation bar with links for "GET STARTED", "DOWNLOAD / PRICING", "DOCUMENTATION", and "BLOG". Below the navigation is a logo featuring a grey cylinder (resembling a database) positioned between two stylized mountain peaks. The word "Flyway" is written in blue lowercase letters next to the cylinder, with a trademark symbol (TM) at the end. Below the logo, the text "by Redgate" is visible. The main headline reads: "Version control for your database. Robust schema evolution across all your environments. With ease, pleasure and plain SQL." A prominent blue button below the headline says "Get Started with Flyway 6.4.4 →". At the bottom of the page, there are links for "Download", "Release Notes", and "Apache v2 license". It also states "Works on: Windows, macOS, Linux, Docker, AWS Lambda, Google Cloud Functions, and IBM Cloud Functions". A newsletter sign-up section at the very bottom encourages users to "Stay updated with our newsletter".

Each DB refactoring is stored in version control

<https://flywaydb.org/>

@samnewman

DATA REFACTORING TOOLS



Flyway™
by Redgate

Version control for your database.
Robust schema evolution across all your environments.
With ease, pleasure and plain SQL.

[Get Started with Flyway 6.4.4 →](#)

Download - Release Notes - Apache v2 license
Works on: Windows, macOS, Linux, Docker, Oracle Database, MySQL, PostgreSQL, SQLite, Microsoft SQL Server, IBM DB2, SAP HANA, Oracle Database, MySQL, PostgreSQL, SQLite, Microsoft SQL Server, IBM DB2, SAP HANA

Stay updated with our newsletter

<https://flywaydb.org/>

Each DB refactoring is stored in version control

Can be run in a safe, deterministic, idempotent manner

DATA REFACTORING TOOLS

The screenshot shows the homepage of the Flyway website. At the top, there is a navigation bar with links for "GET STARTED", "DOWNLOAD / PRICING", "DOCUMENTATION", and "BLOG". Below the navigation is the Flyway logo, which features a stylized grey cylinder (resembling a database) positioned between two wavy lines that suggest mountains or waves. The word "Flyway" is written in blue lowercase letters next to the cylinder, with a trademark symbol (TM) at the end. Below the logo, the text "by Redgate" is visible. The main headline reads "Version control for your database. Robust schema evolution across all your environments. With ease, pleasure and plain SQL." Below this, a blue button contains the text "Get Started with Flyway 6.4.4 →". At the bottom of the page, there is a link to "Download - Release Notes - Apache v2 license" and a section titled "Works on:" followed by icons for various operating systems and databases. A newsletter sign-up form is also present at the bottom.

Each DB refactoring is stored in version control

Can be run in a safe, deterministic, idempotent manner

Do it throughout your deployment pipeline to ensure it's rigorously tested

<https://flywaydb.org/>

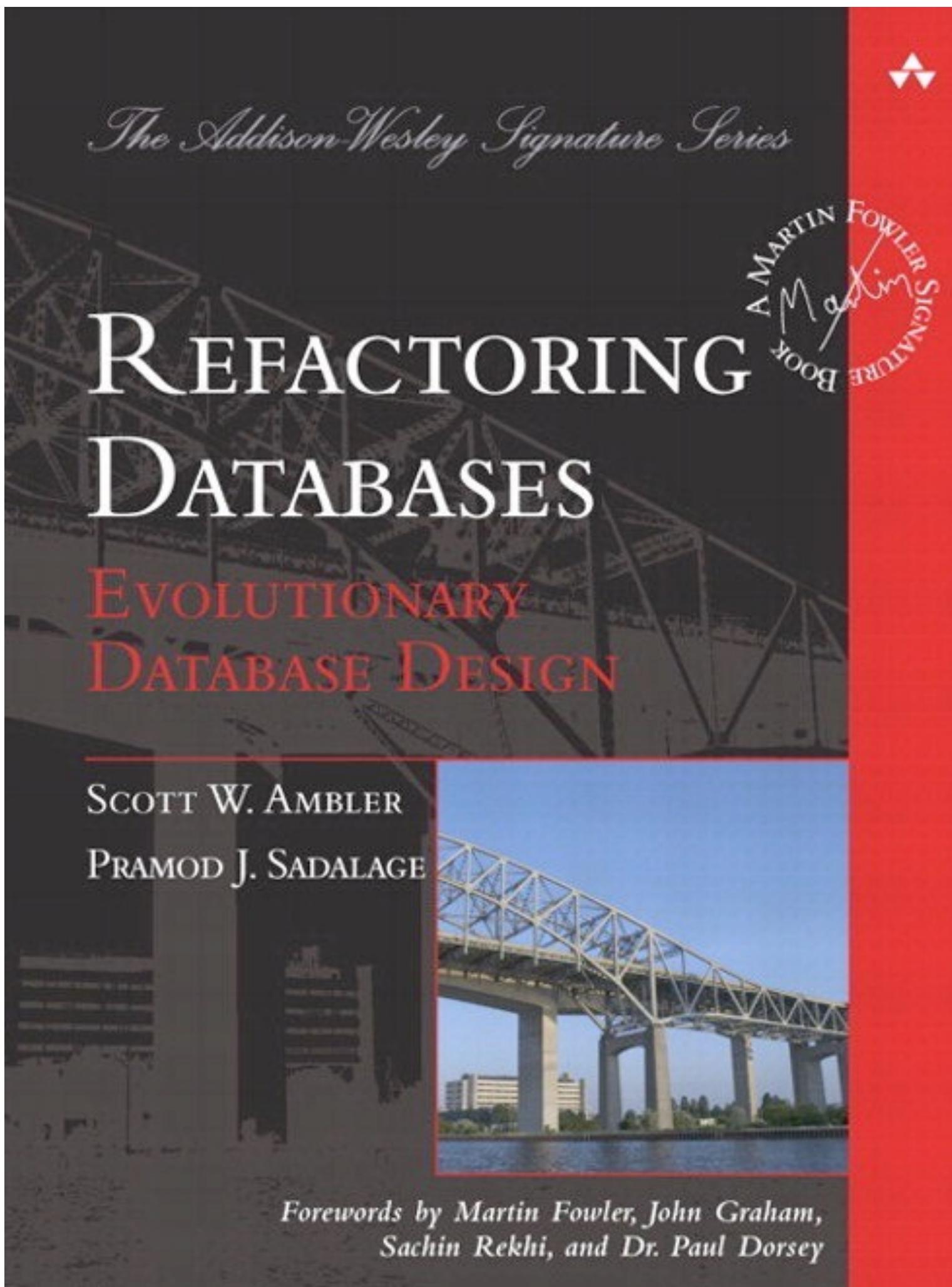
@samnewman

POLL: HAVE YOU USED A DB REFACTORING TOOL?

Yes

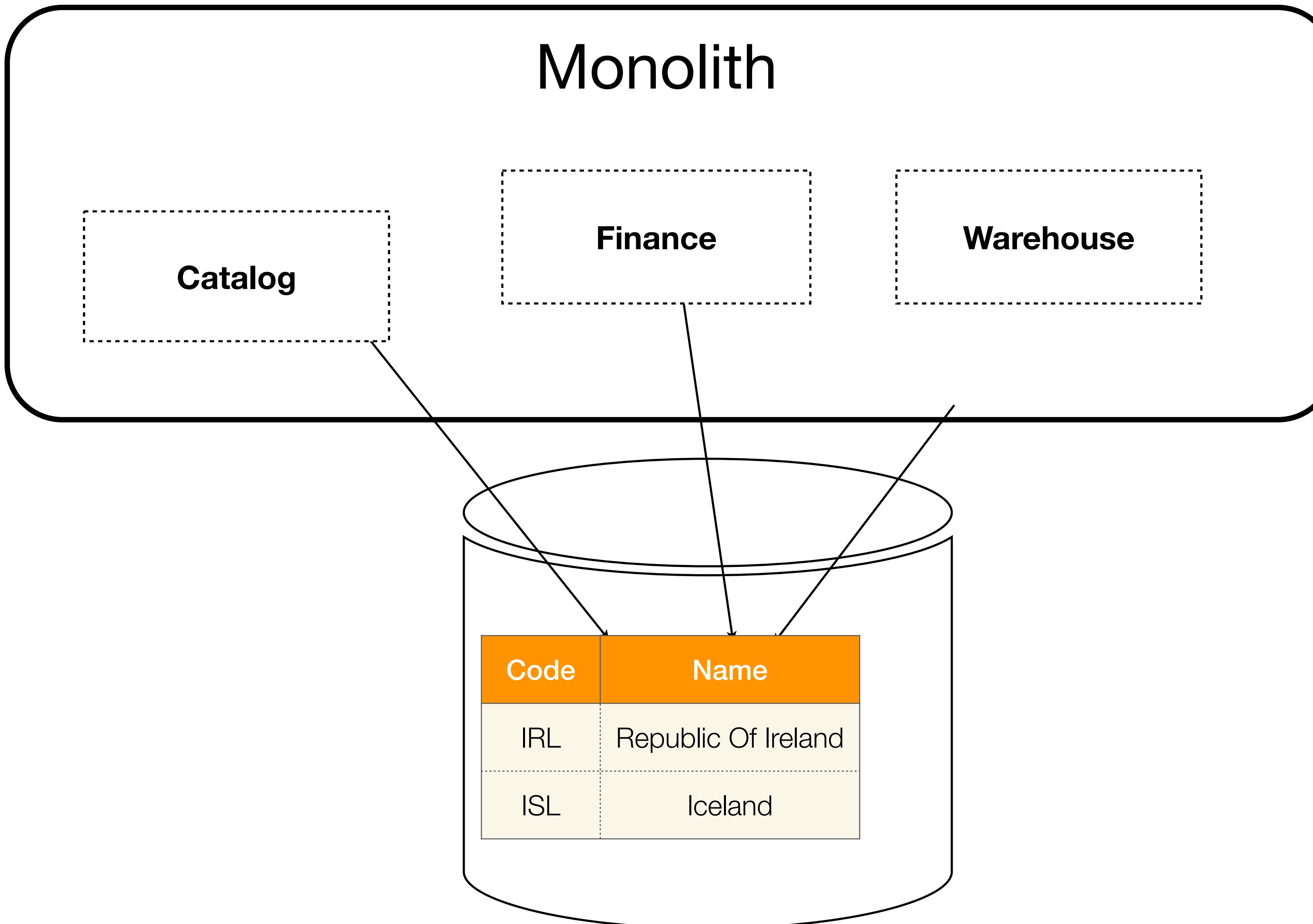
No

REFACTORING DATABASES - FURTHER READING

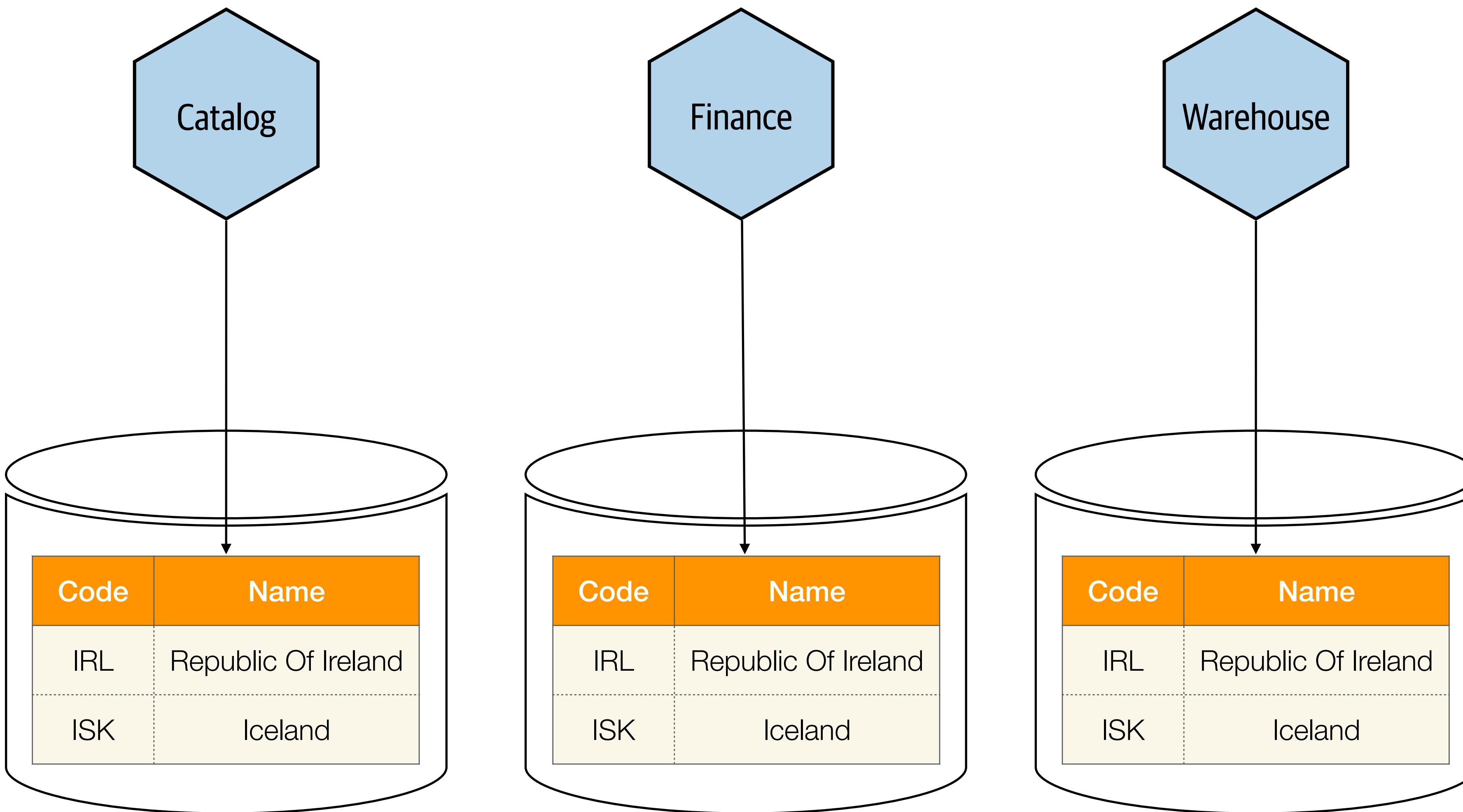


Shared Reference Data

STATIC REFERENCE DATA



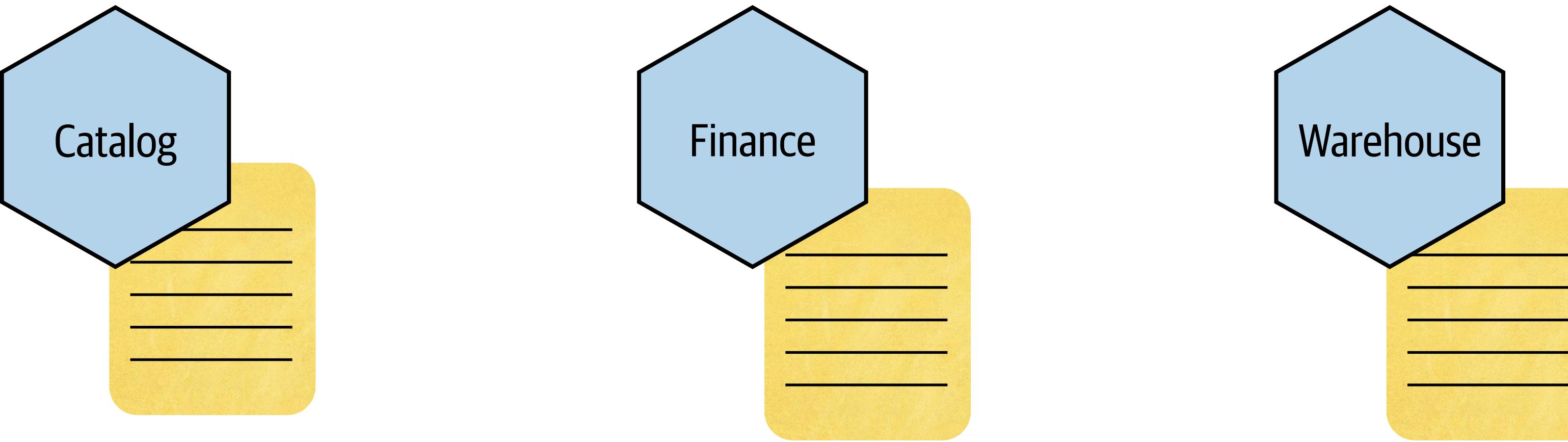
DUPLICATE THE DATA?



Duplication isn't always bad...

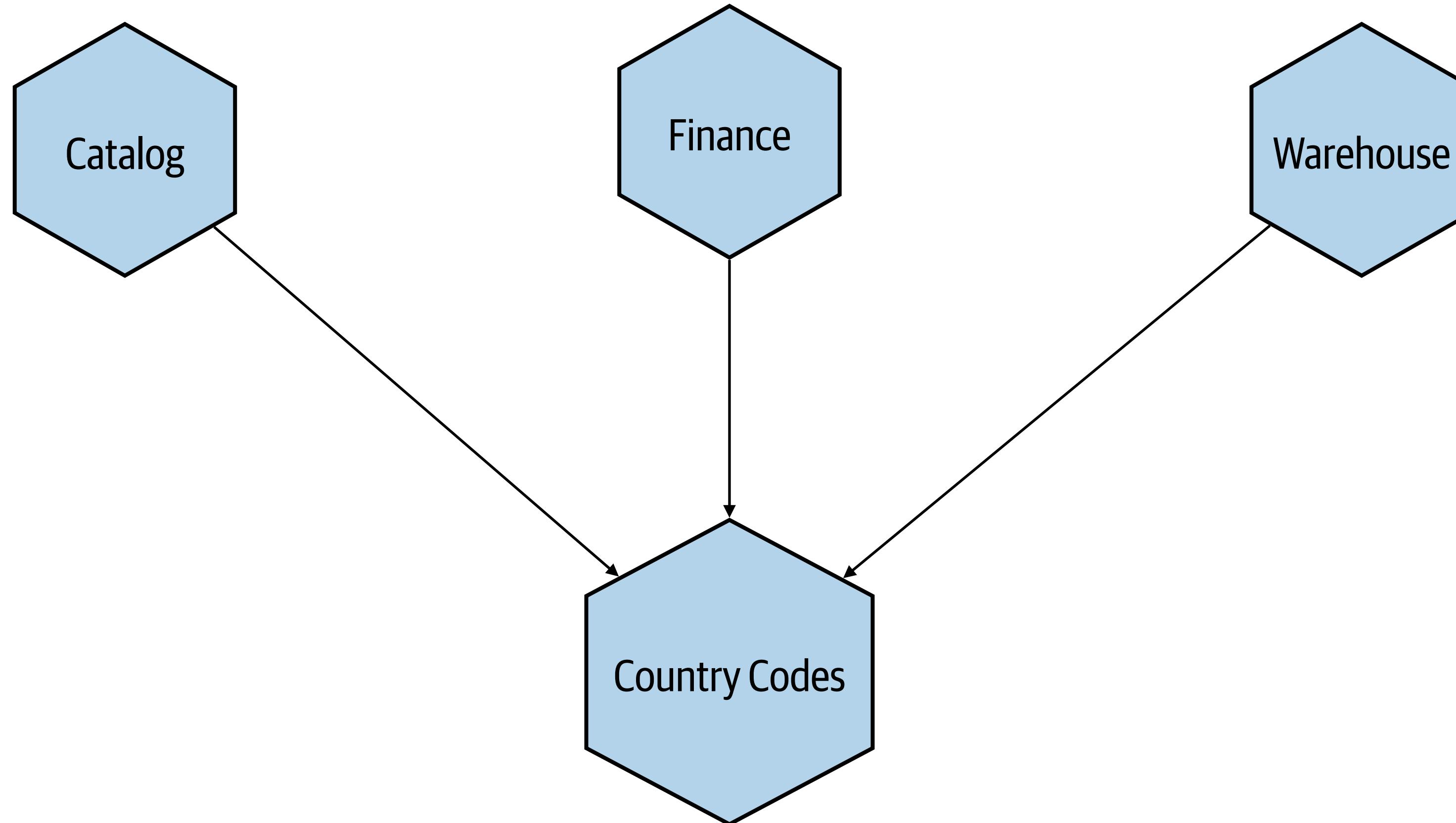
...but watch for inconsistency

MOVE DATA INTO CONFIGURATION, OR LIBRARIES

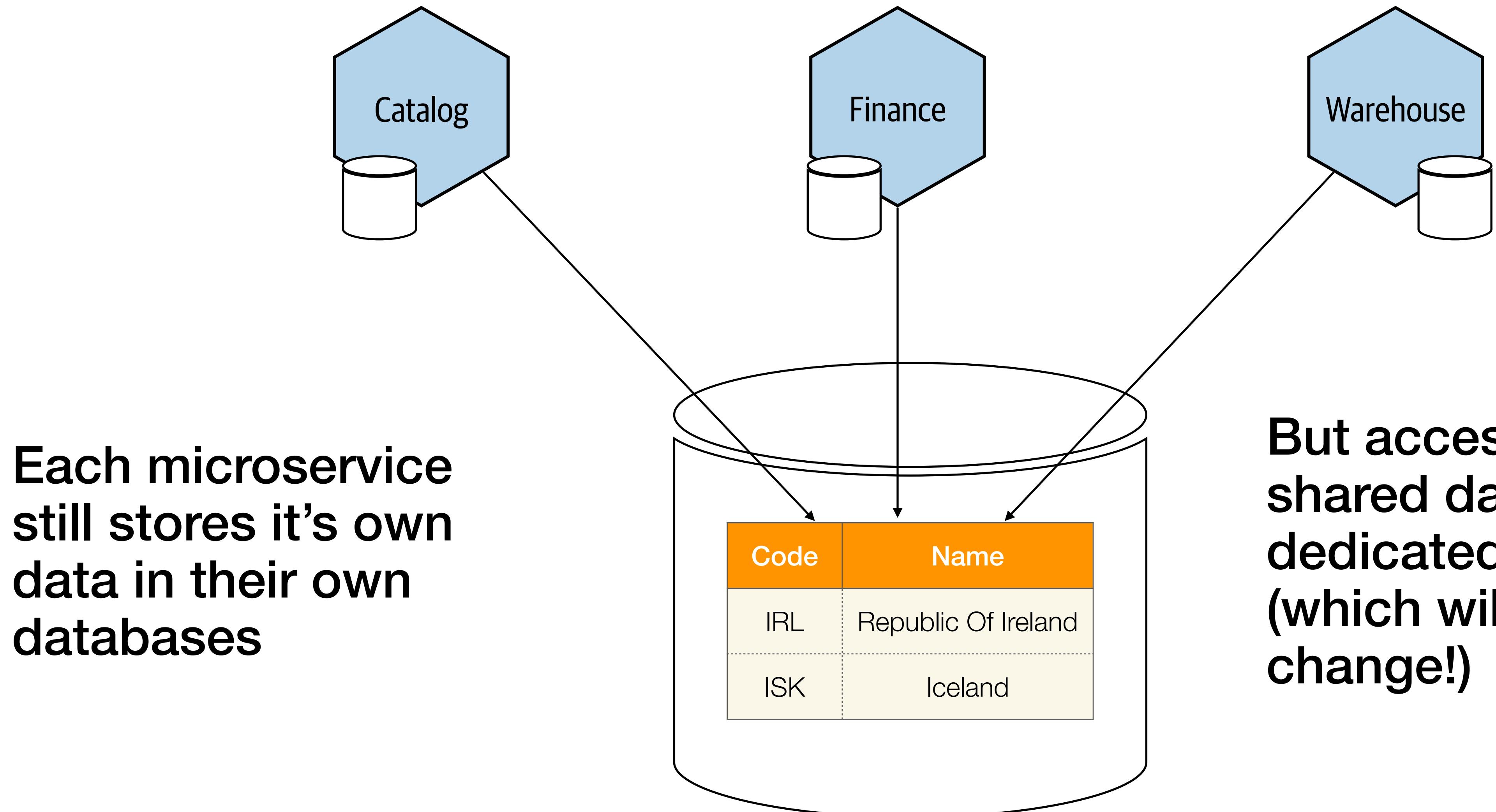


**Static data is packaged inside
each microservice instance**

CREATE A DEDICATED SERVICE



CREATE A DEDICATED DB



WHICH TO USE?

WHICH TO USE?

If you need to see the
same data everywhere

WHICH TO USE?

If you need to see the
same data everywhere

Use a dedicated DB for
reference data

WHICH TO USE?

If you need to see the same data everywhere

Use a dedicated DB for reference data

Use a dedicated microservice if you can justify it

WHICH TO USE?

If you need to see the same data everywhere

Use a dedicated DB for reference data

Use a dedicated microservice if you can justify it

If you are ok with potentially having different sets of data

WHICH TO USE?

If you need to see the same data everywhere

Use a dedicated DB for reference data

Use a dedicated microservice if you can justify it

If you are ok with potentially having different sets of data

For small data sets, a shared library or config file works well

WHICH TO USE?

If you need to see the same data everywhere

Use a dedicated DB for reference data

Use a dedicated microservice if you can justify it

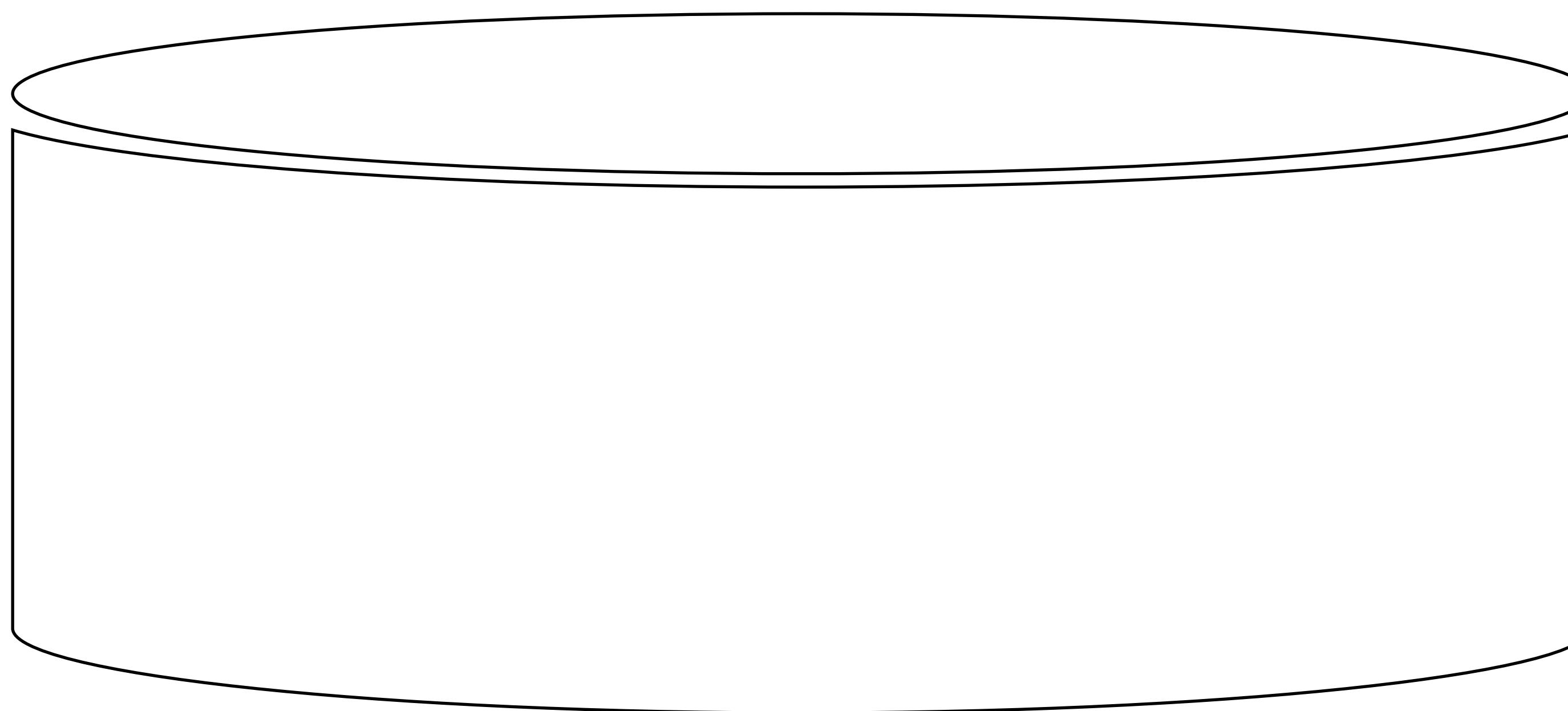
If you are ok with potentially having different sets of data

For small data sets, a shared library or config file works well

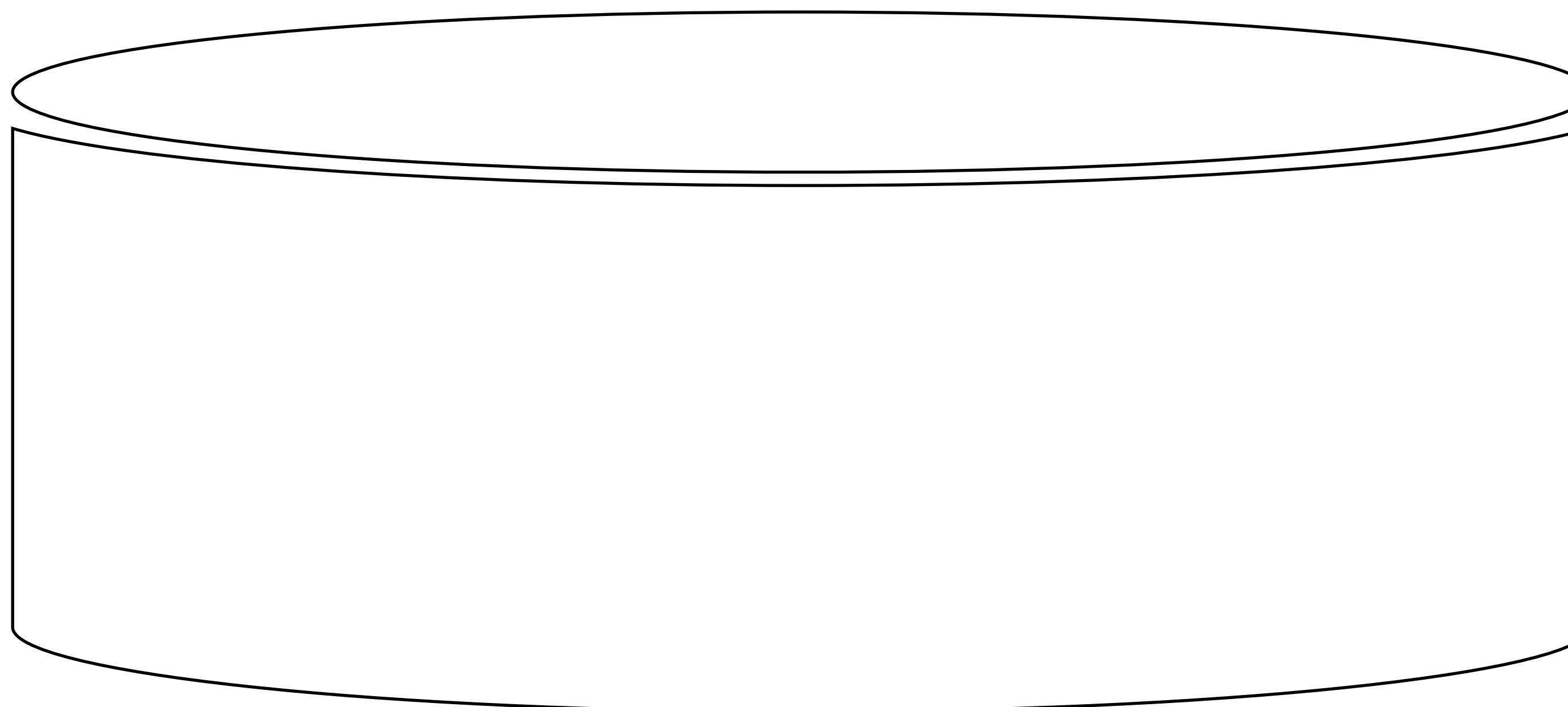
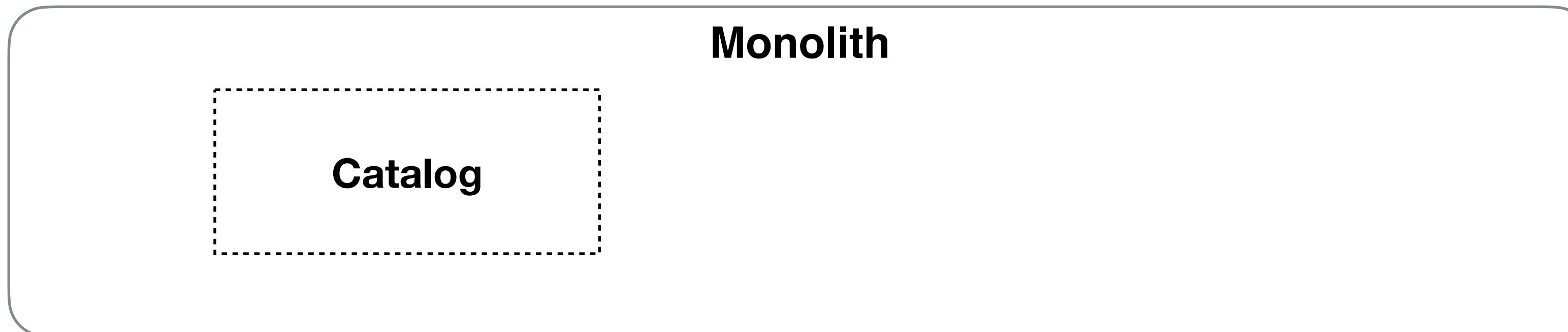
For larger datasets, copy data into each microservices' DB

TOP CHARTS!

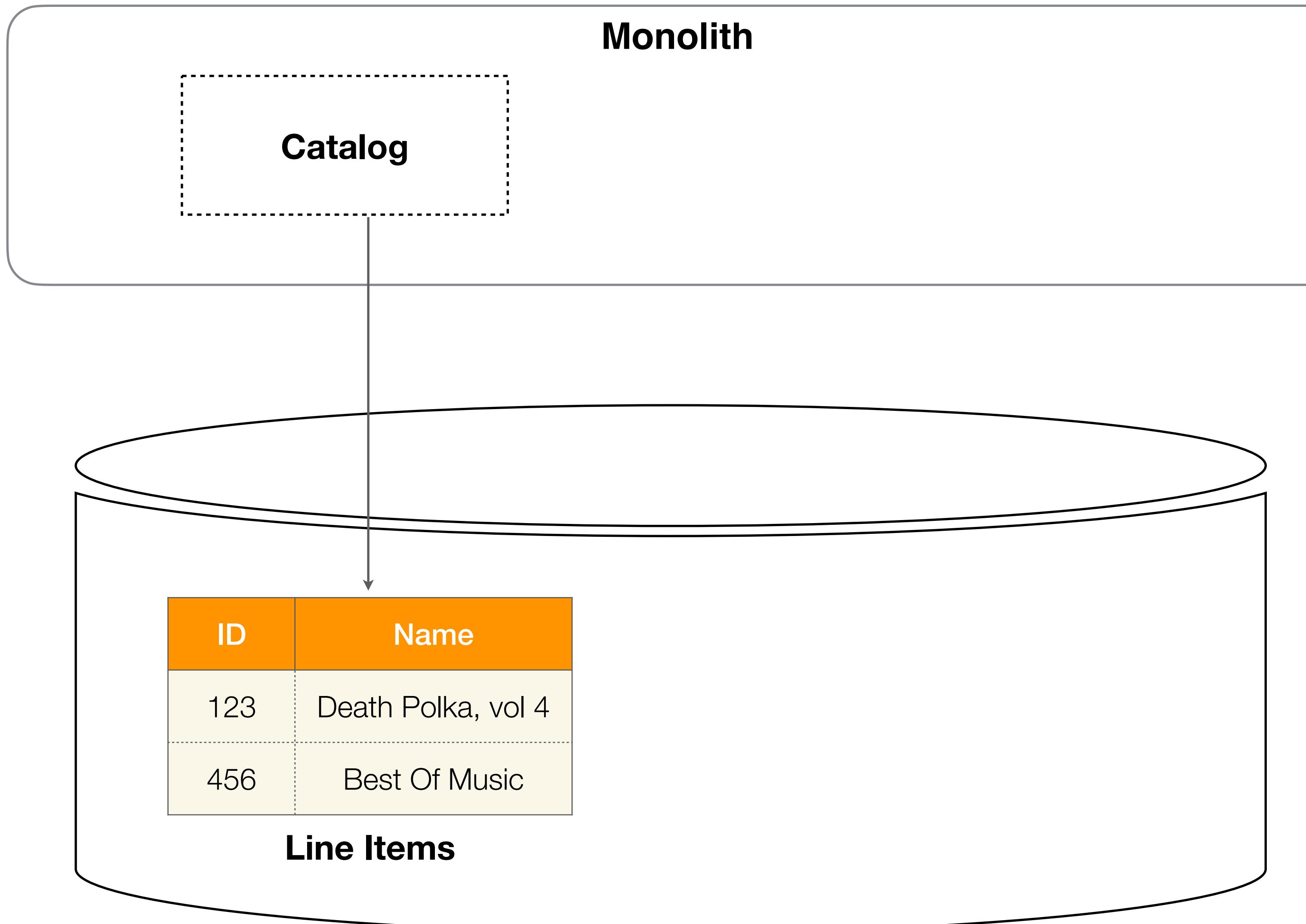
Monolith



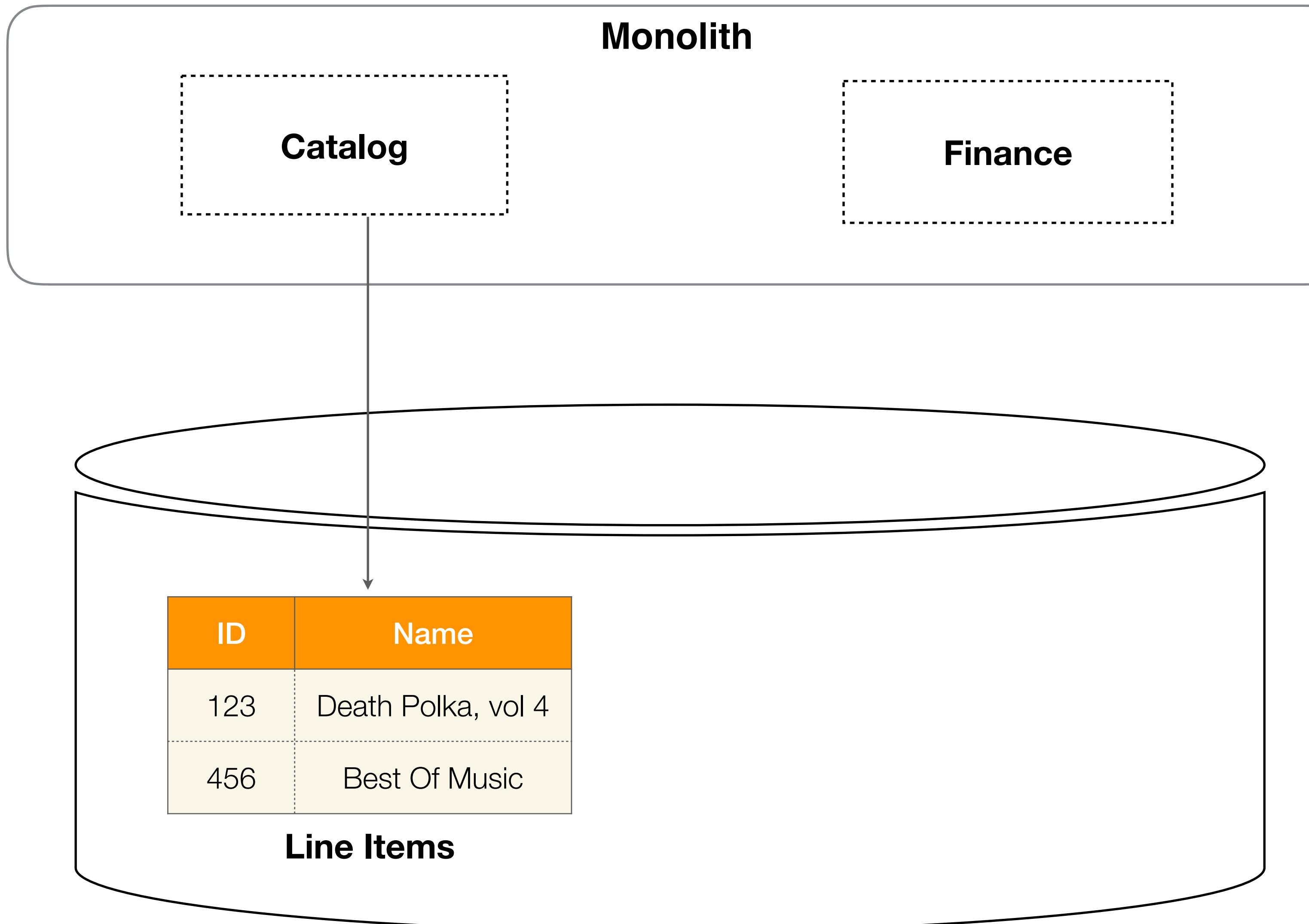
TOP CHARTS!



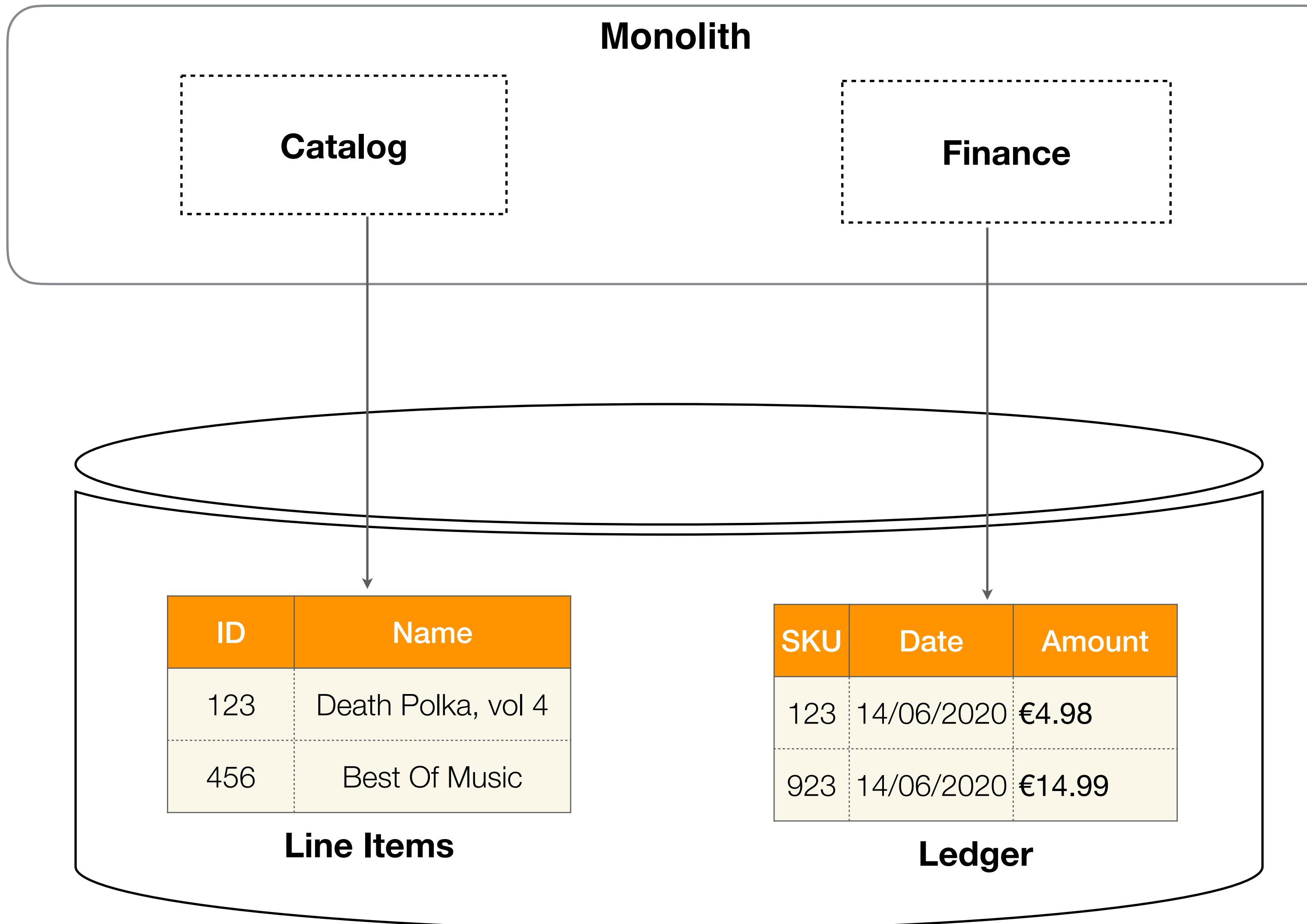
TOP CHARTS!



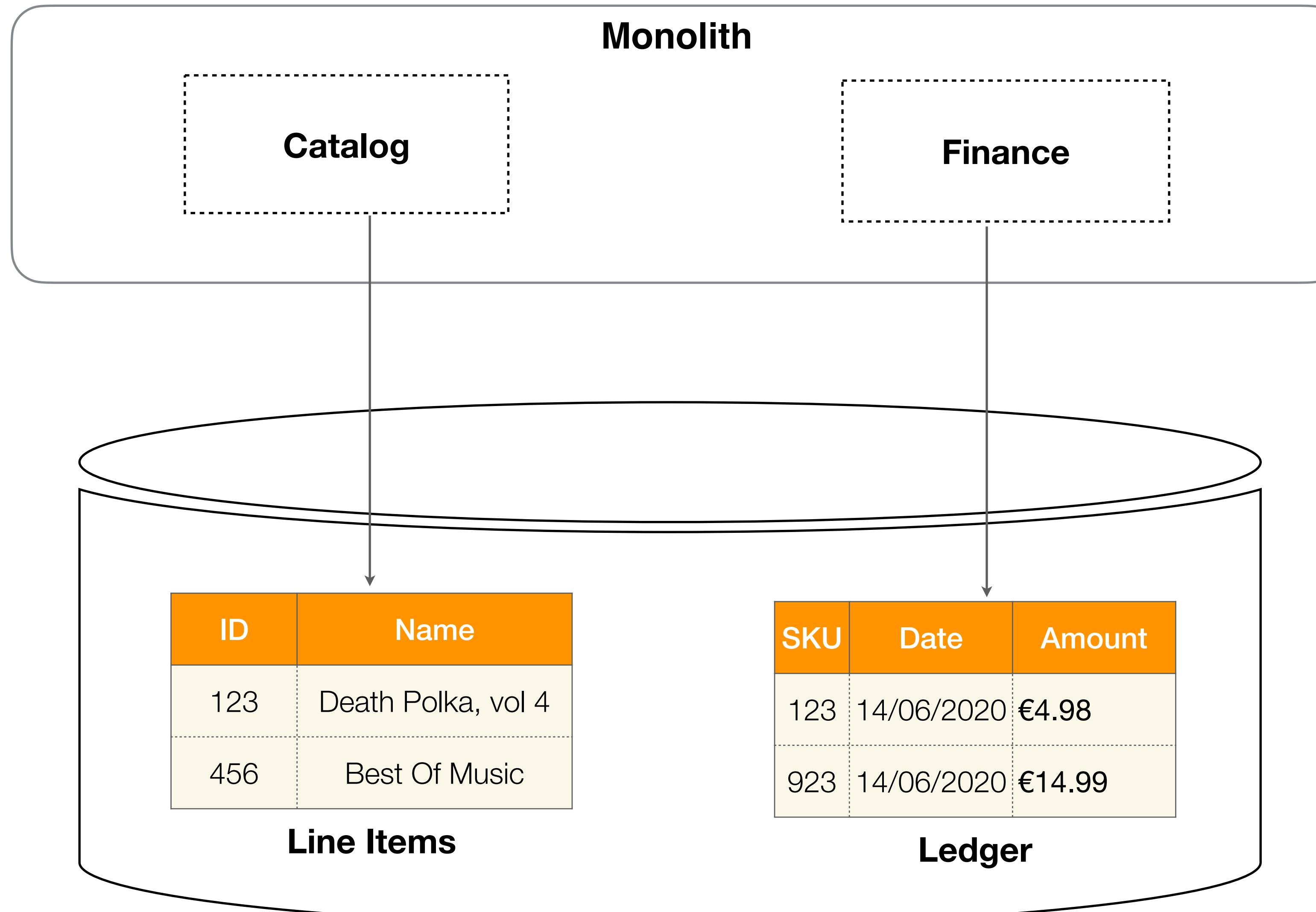
TOP CHARTS!



TOP CHARTS!



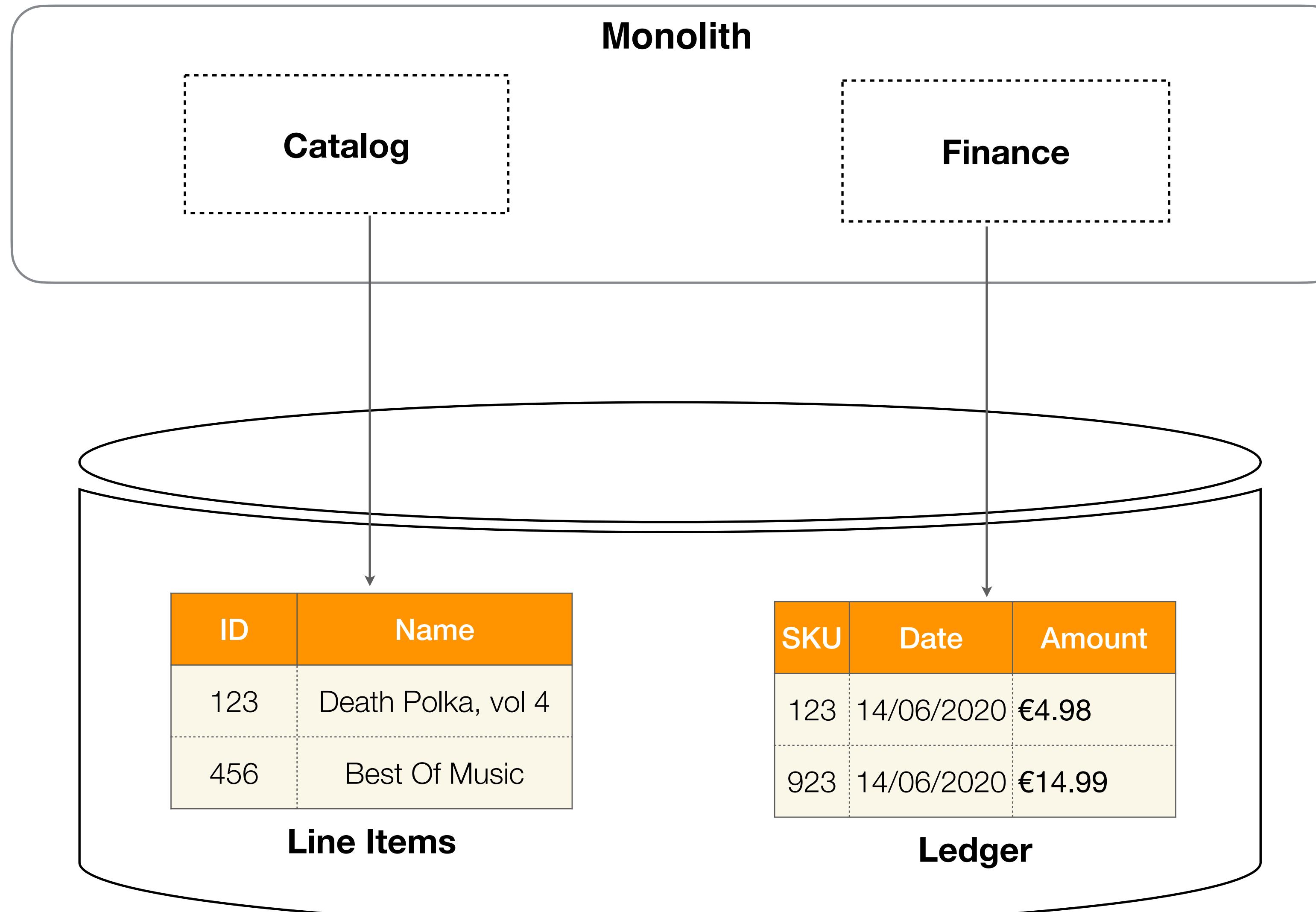
TOP CHARTS!



Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

TOP CHARTS!

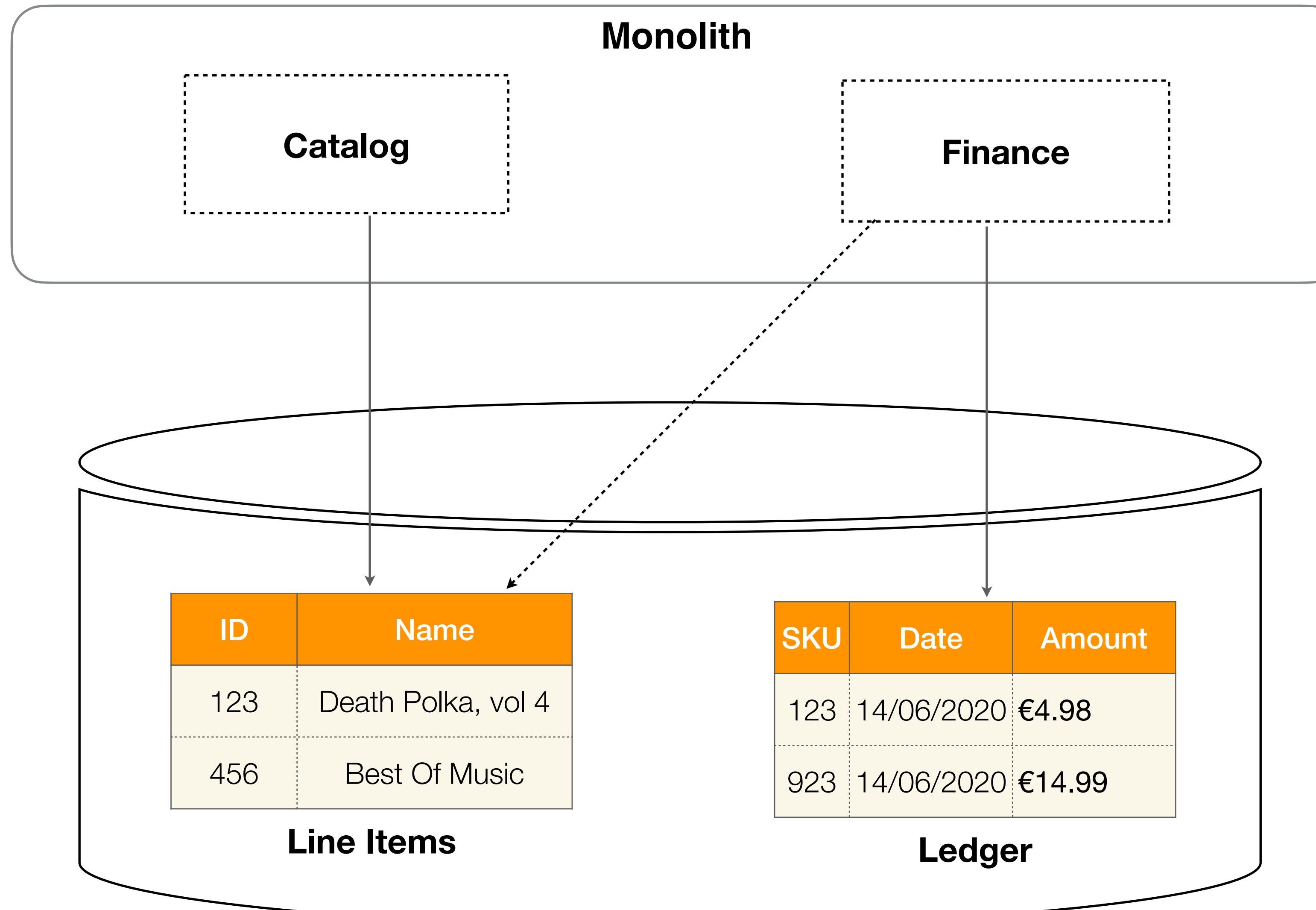


Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

The ledger contains the ID of what sold when, but we need the name....

TOP CHARTS!

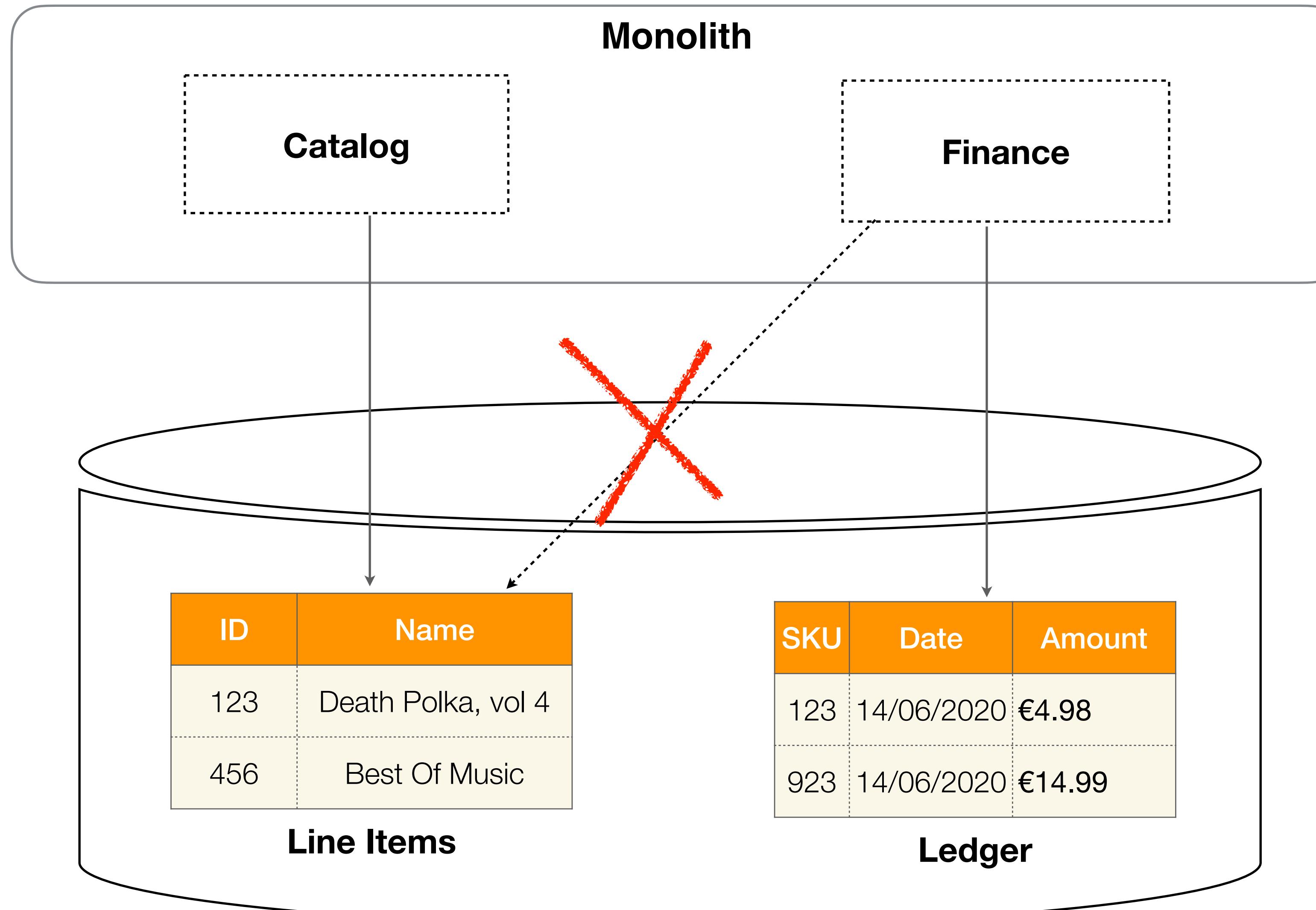


Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

The ledger contains the ID of what sold when, but we need the name....

TOP CHARTS!

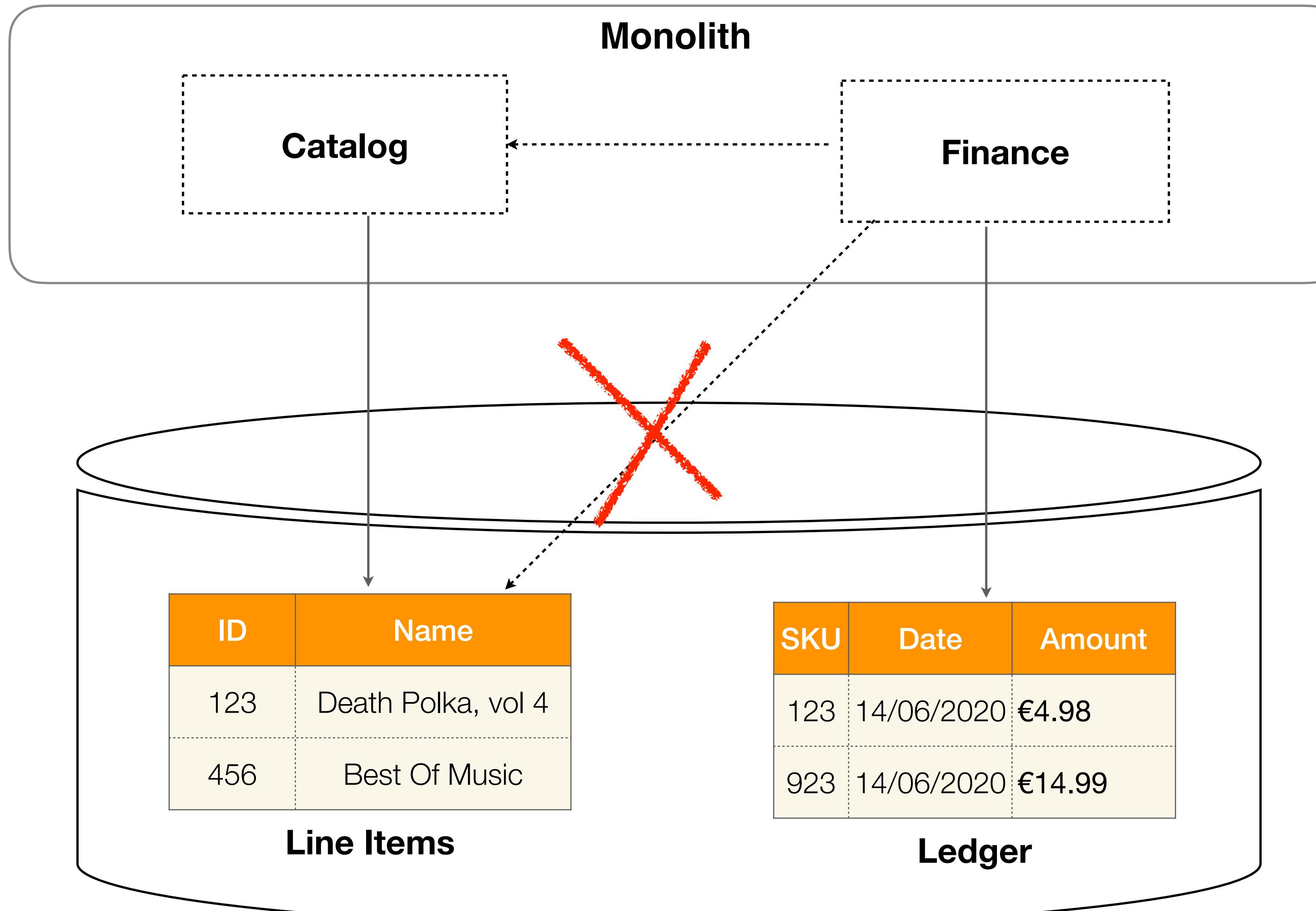


Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

The ledger contains the ID of what sold when, but we need the name....

TOP CHARTS!

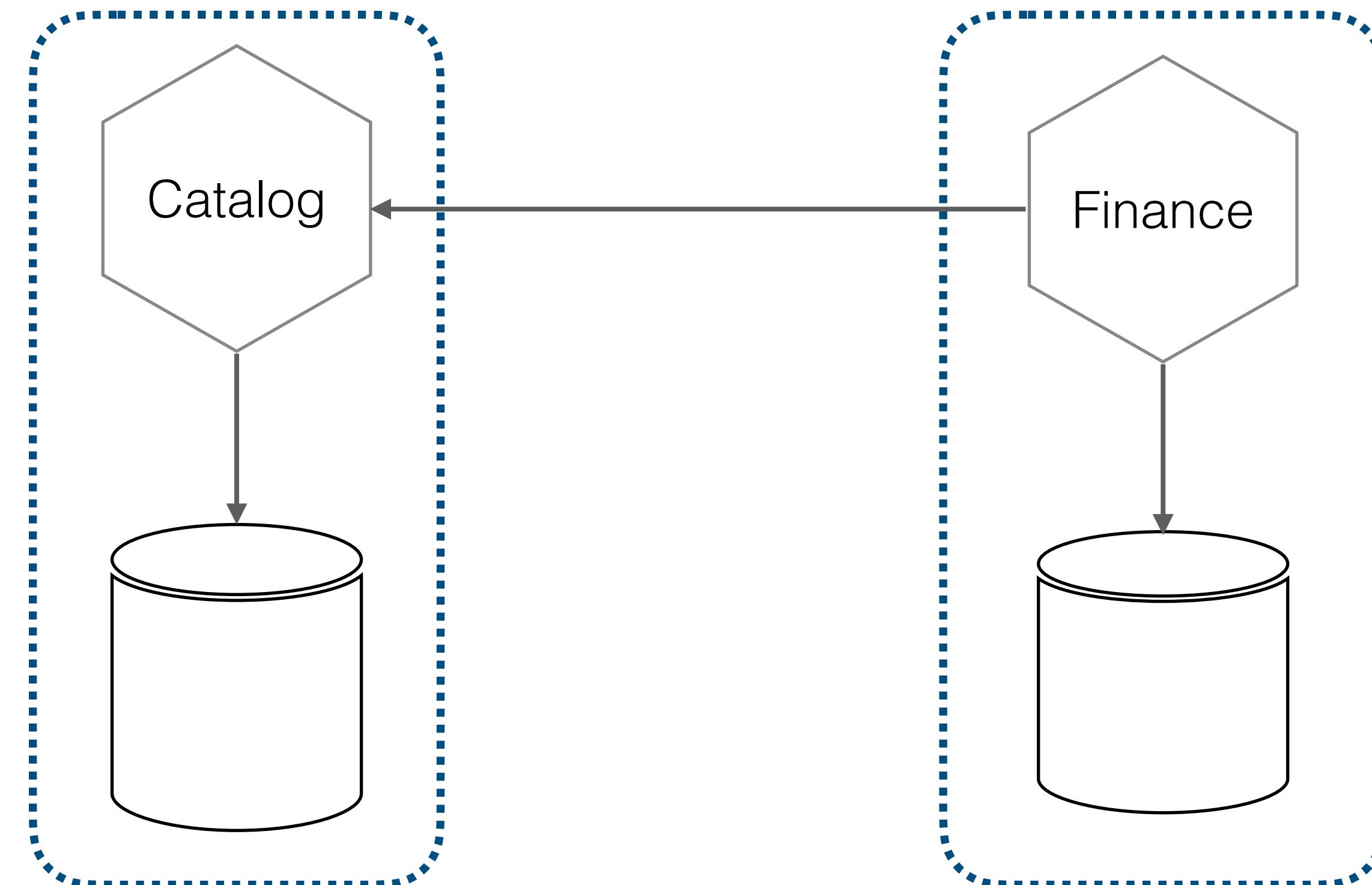


Best Sellers This Week!

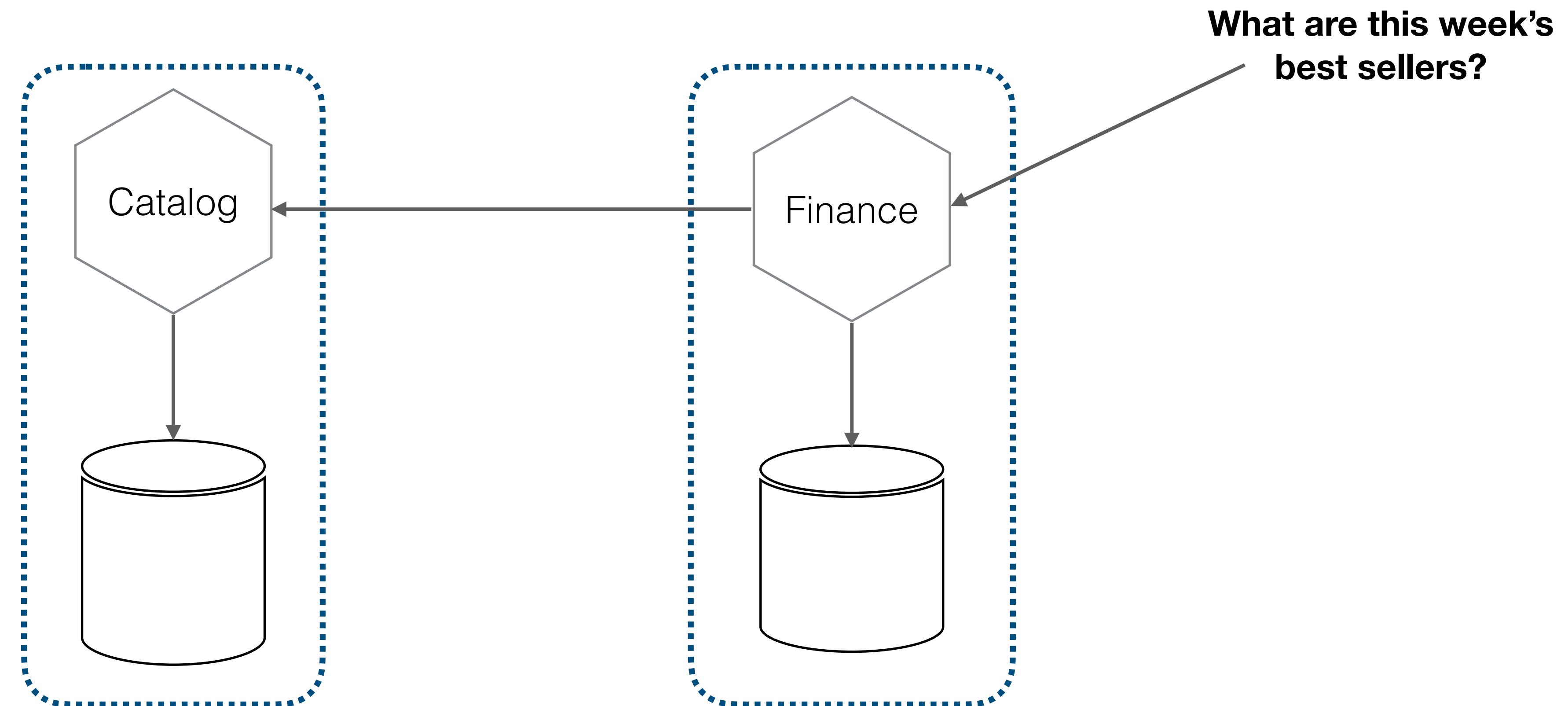
1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

The ledger contains the ID of what sold when, but we need the name....

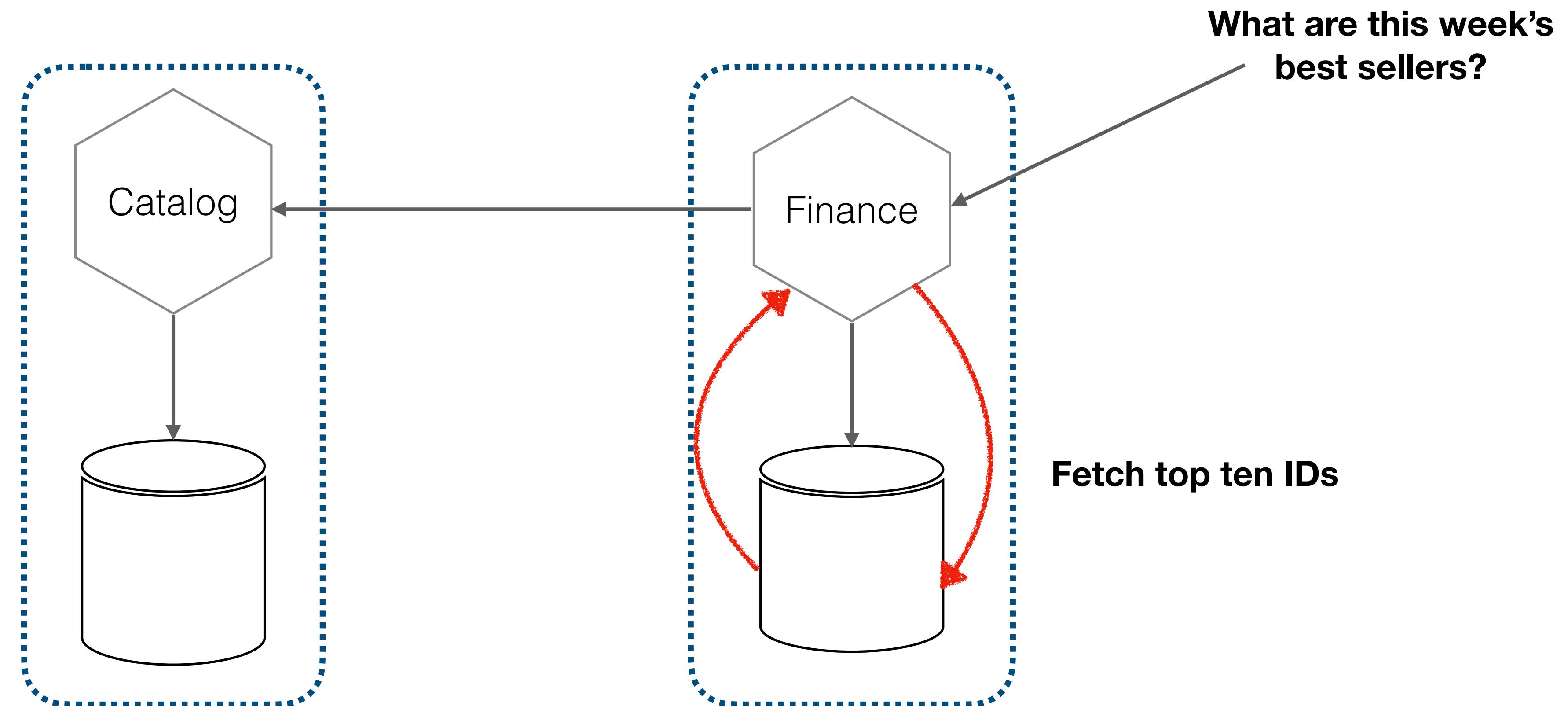
JOINS AT THE SERVICES TIER



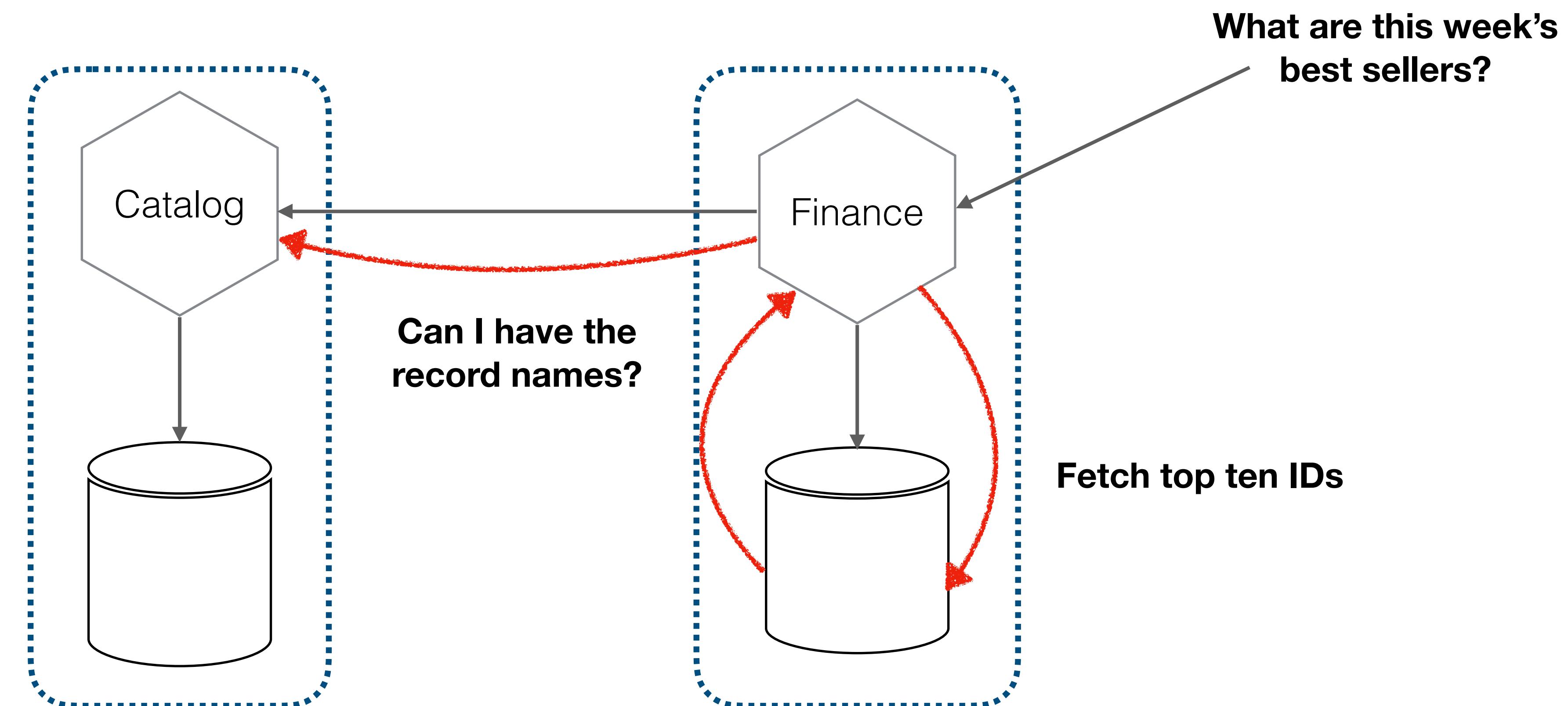
JOINS AT THE SERVICES TIER



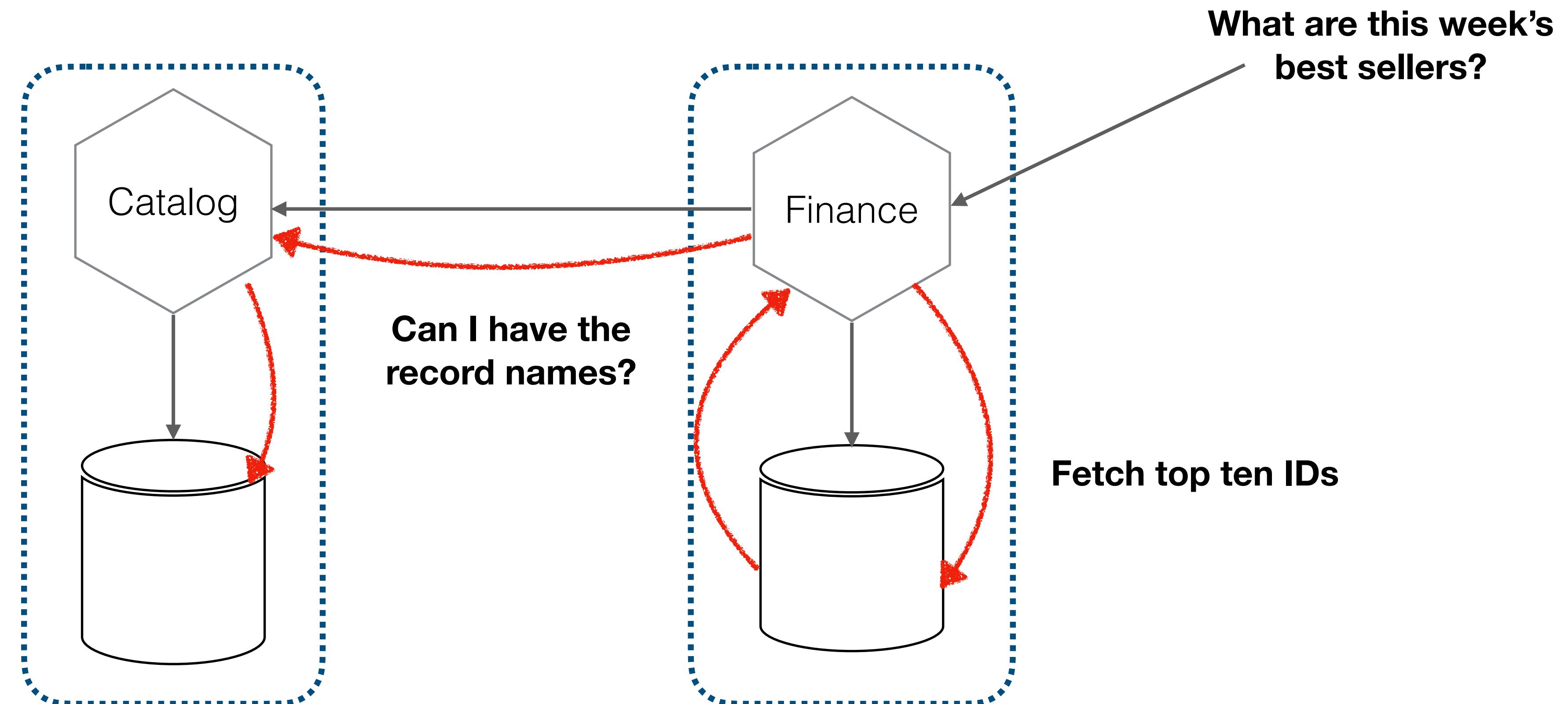
JOINS AT THE SERVICES TIER



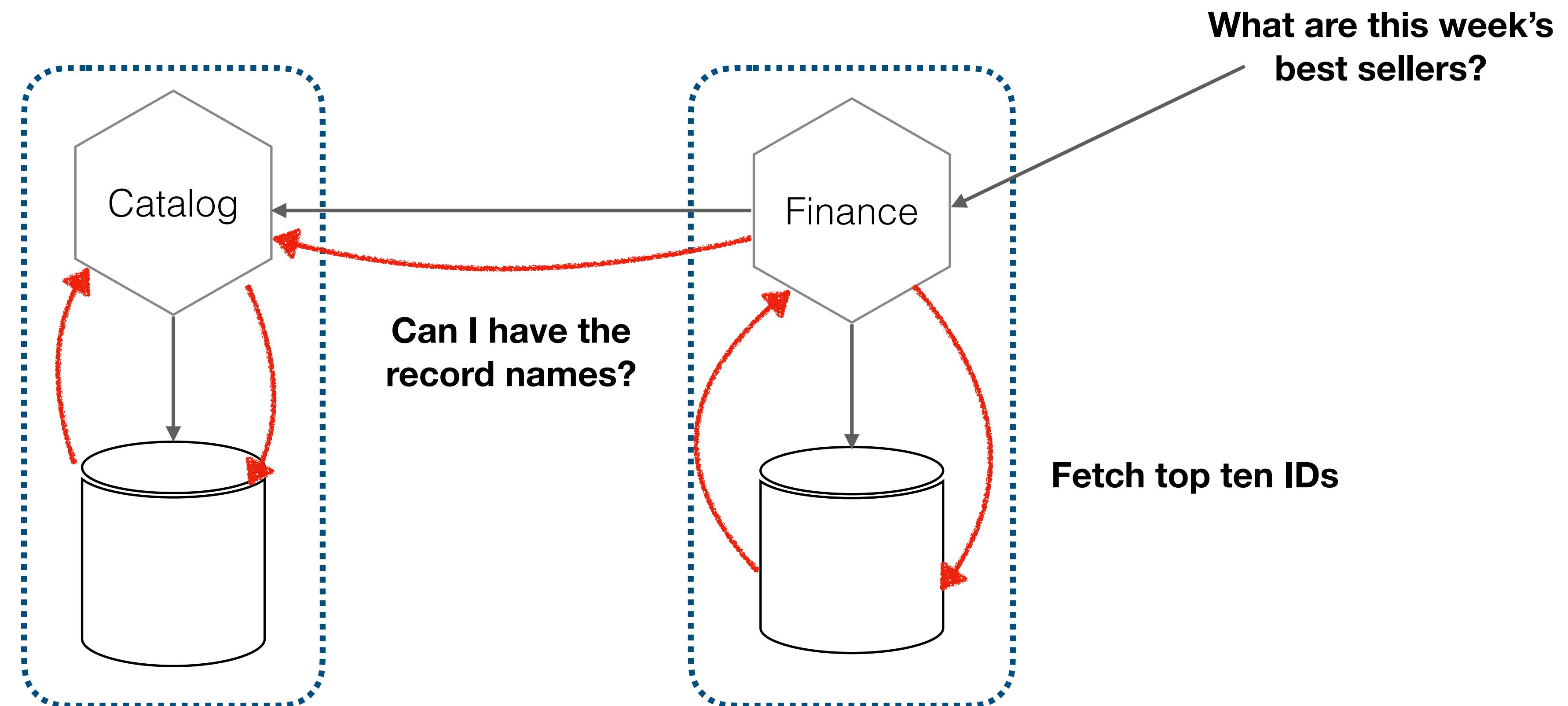
JOINS AT THE SERVICES TIER



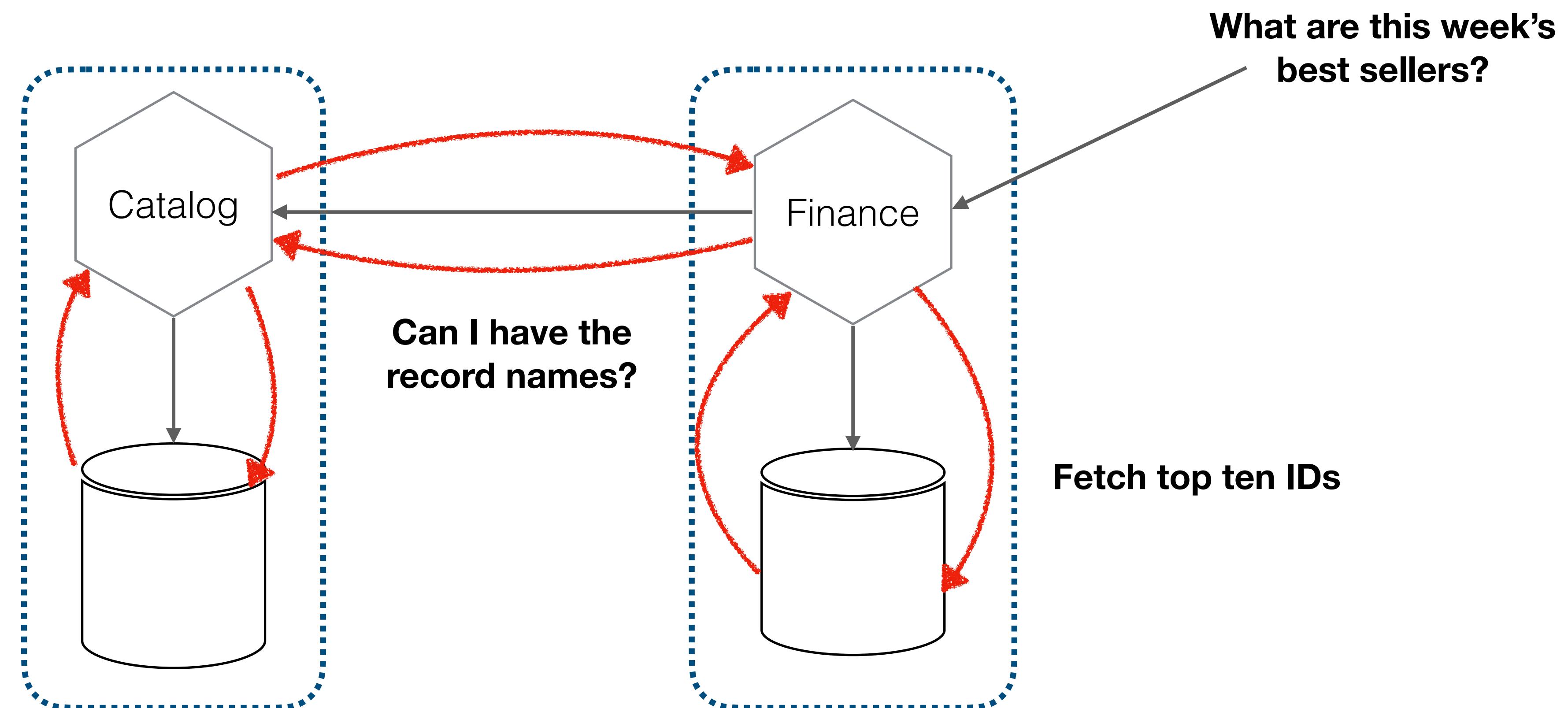
JOINS AT THE SERVICES TIER



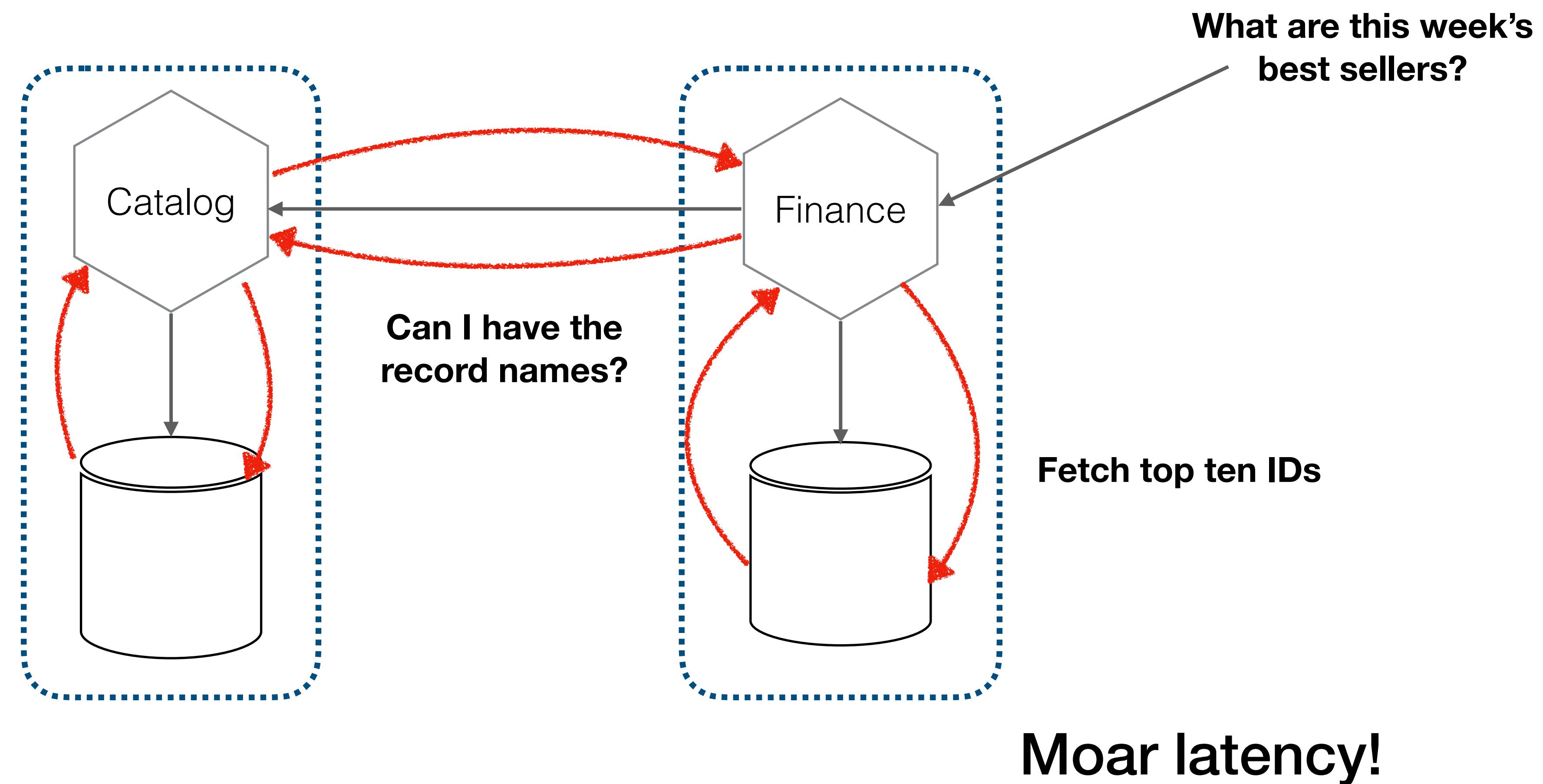
JOINS AT THE SERVICES TIER



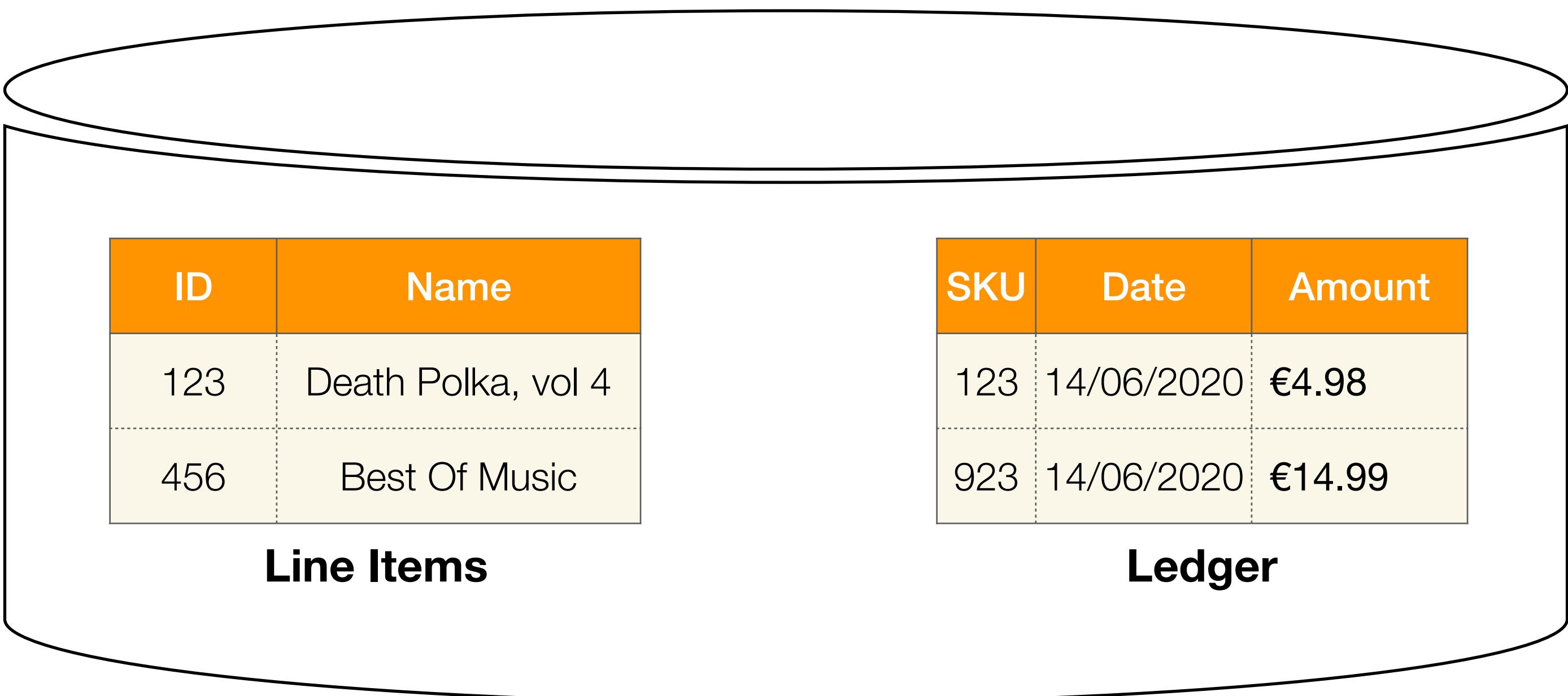
JOINS AT THE SERVICES TIER



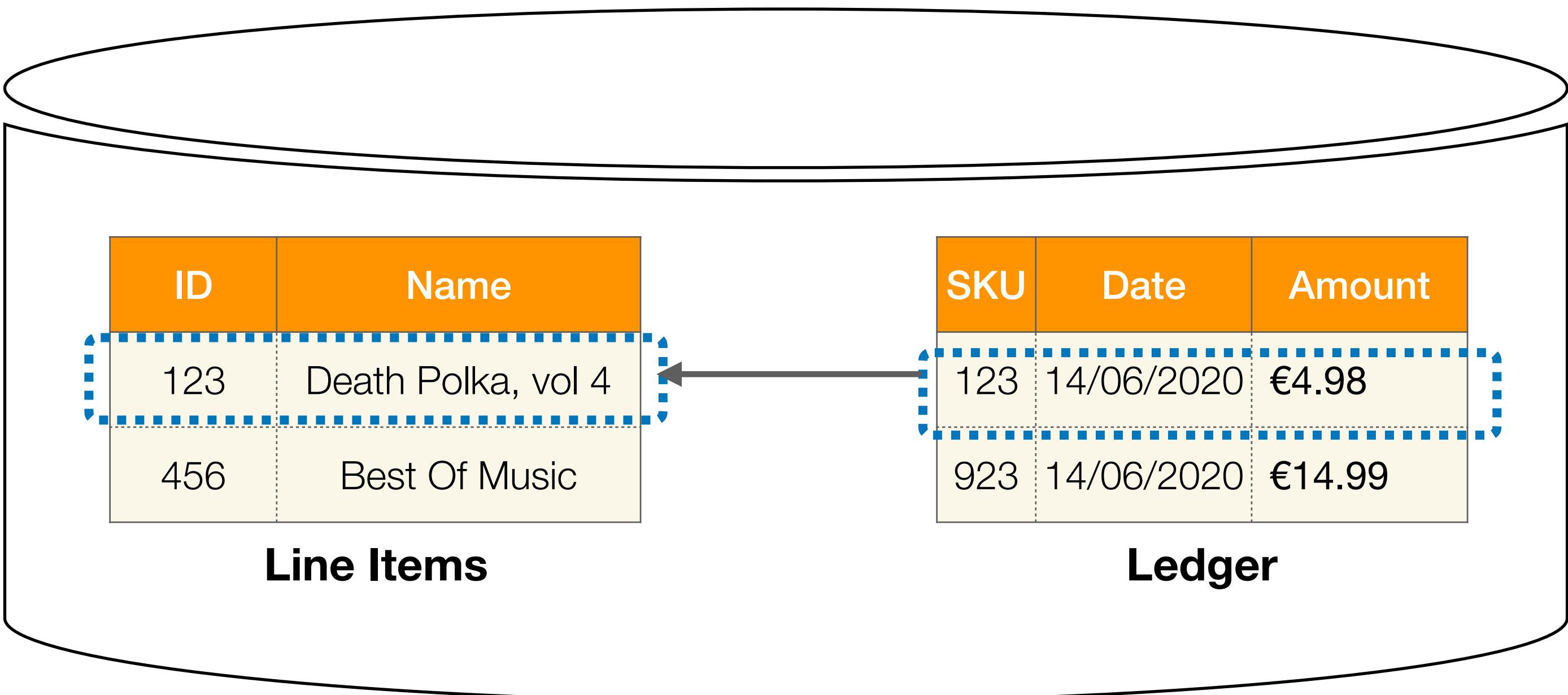
JOINS AT THE SERVICES TIER



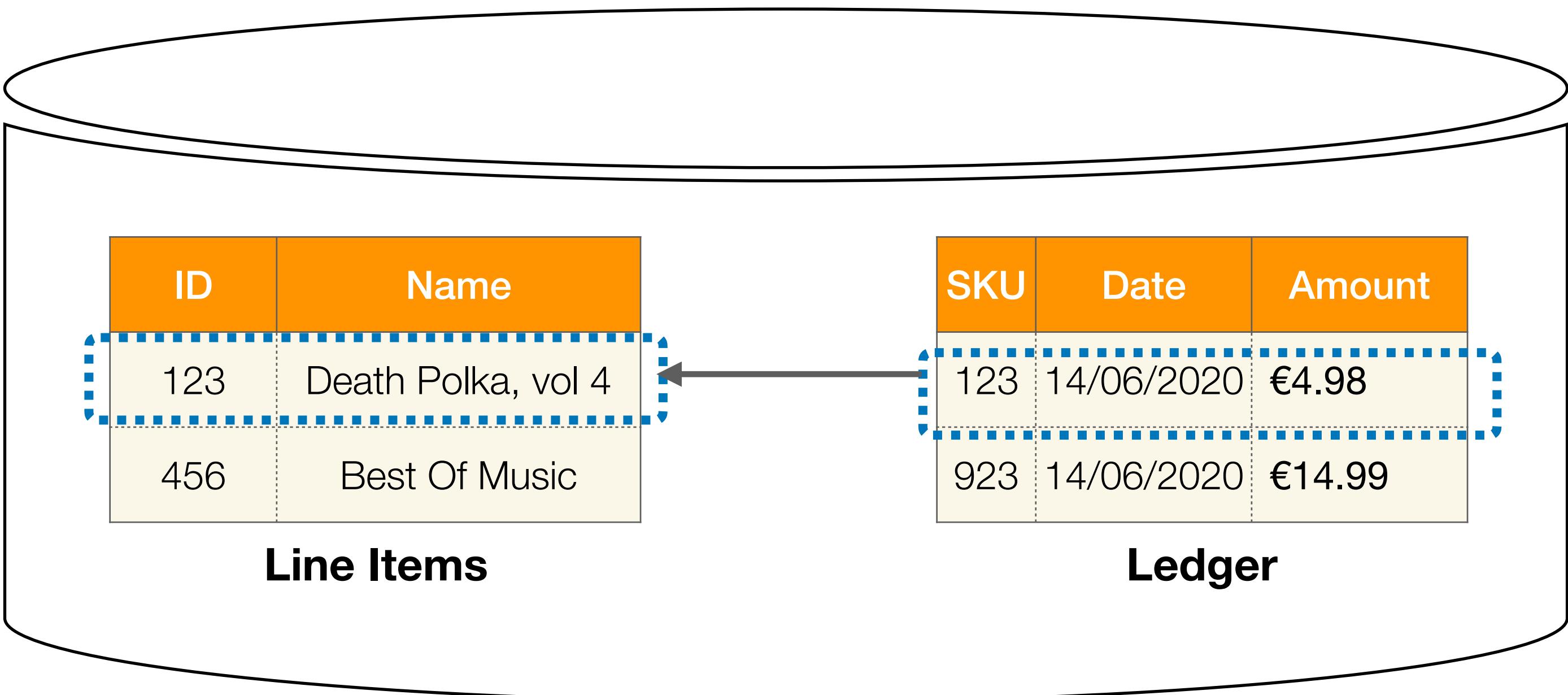
FOREIGN KEY RELATIONSHIPS



FOREIGN KEY RELATIONSHIPS

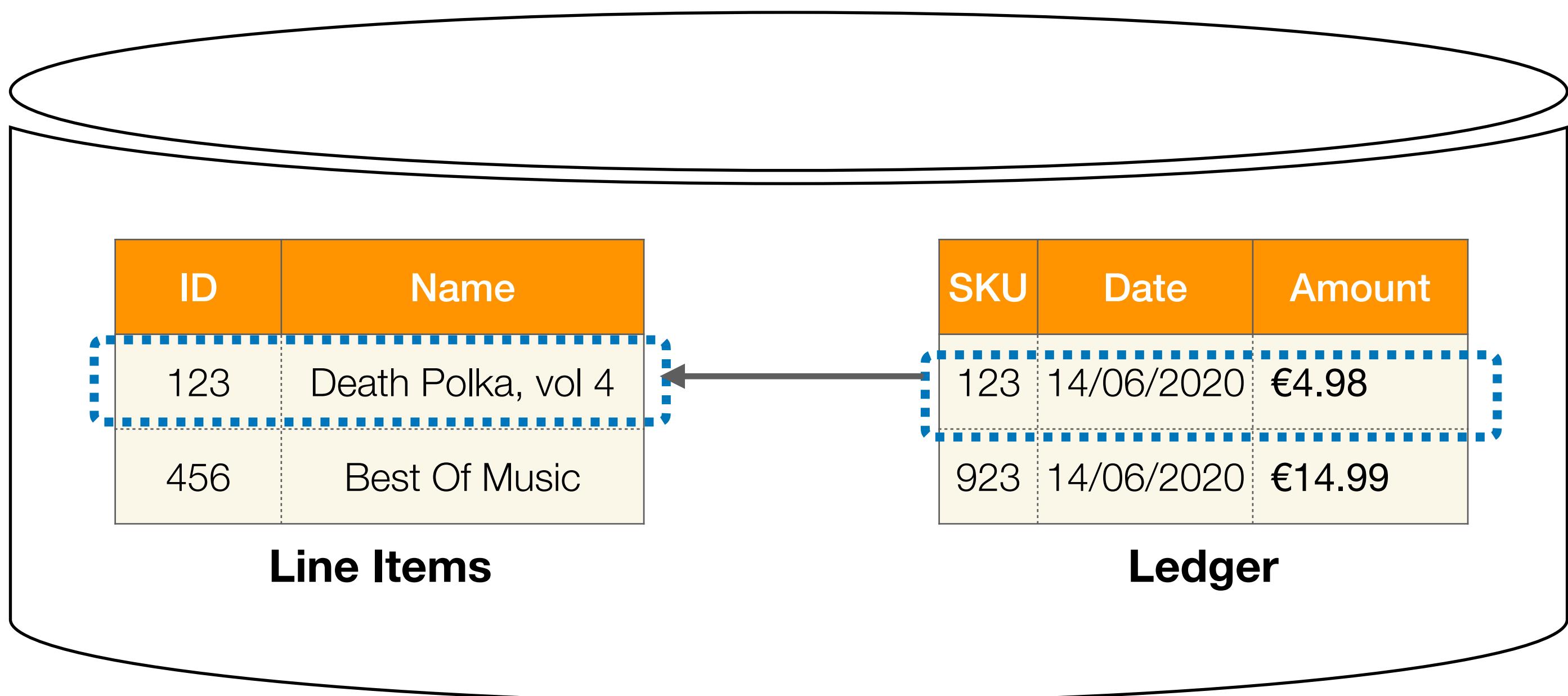


FOREIGN KEY RELATIONSHIPS



Enforce referential integrity

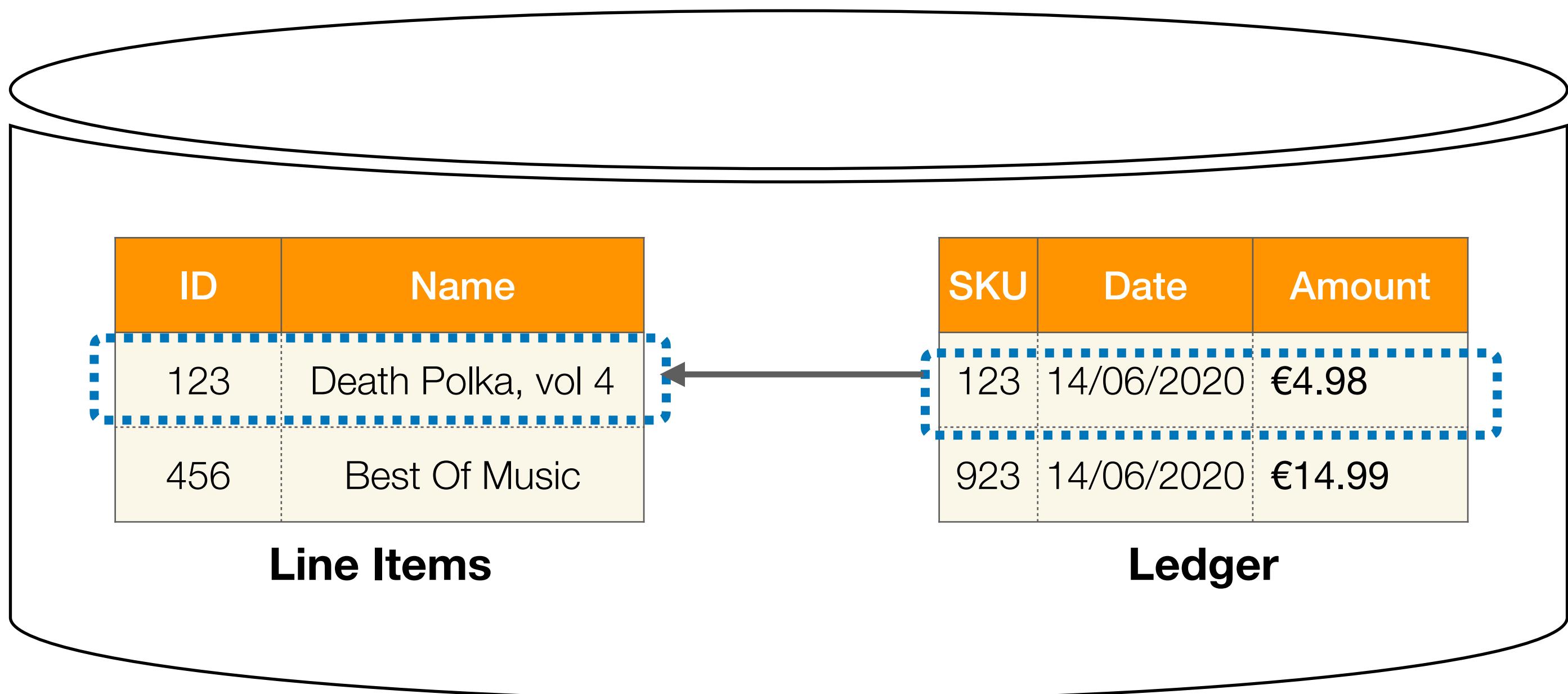
FOREIGN KEY RELATIONSHIPS



Enforce referential integrity

Make relationships explicit

FOREIGN KEY RELATIONSHIPS



Enforce referential integrity

Make relationships explicit

Aid join performance

MAKING REFERENCES EXPLICIT

Pattern: Using Pseudo-URIs with Microservices

Mar 22, 2017

• Microservices • Distributed Systems • Patterns •

I've spent some time talking about [the very basics you need to have in place before thinking about going down a microservices route](#), but even if you have these in place that doesn't mean that you aren't going to find some new surprises. Microservices impose a very distributed architecture, and this requires us to re-visit many concepts that are considered a solved problem in more traditional scenarios.

One of such concepts is how we implement identities for objects. This is usually a no-brainer in more monolithic architectures but it becomes a more interesting challenge as we have more distribution and collaboration between services.

How I understand object identity

Before we discuss how things change in a distributed services scenario, let's try to build a working definition of object identity.

The first thing to clarify is what mean by the word *object* in this text. While we are going to use some

pURIs: Borrowing a solution from the Internet

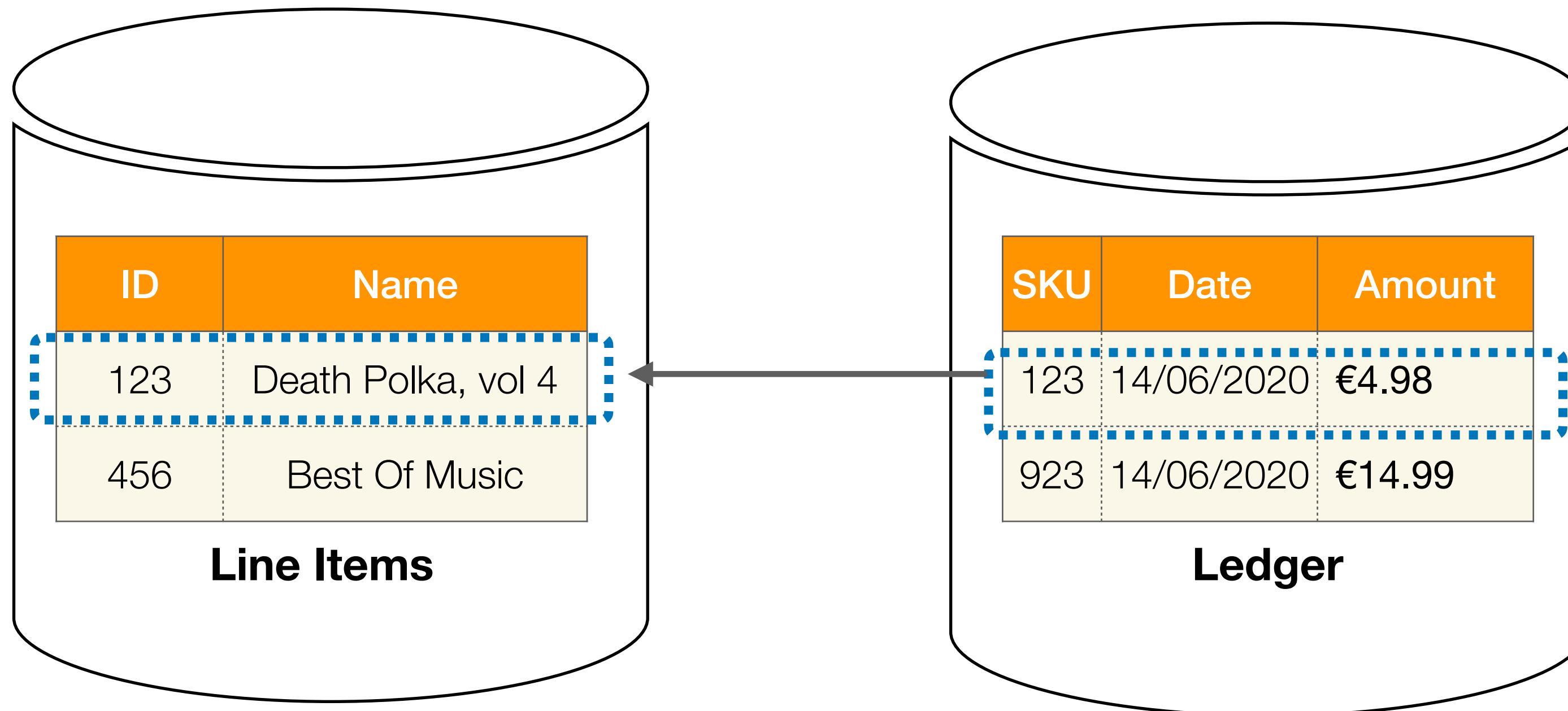
When facing the problems above, my team at SoundCloud started exploring alternatives that would allow for us to have simple, scalar values that were still rich enough to act as good identifiers across our hundreds of microservices. Reading through decades of industry work on the matter, we found something simple that could help us: [Uniform Resource Names, or URNs](#). URNs were a type of [Uniform Resource Identifiers \(URIs\)](#) that, as opposed to [URLs](#), were only concerned with the *identifier* for a resource, but not with how to locate it.

The specification allowed for us to create structured identifiers that would contain information about the object's type. This means that instead of the confusion created by an `id` of `123`, we would have `id`s that looked like:

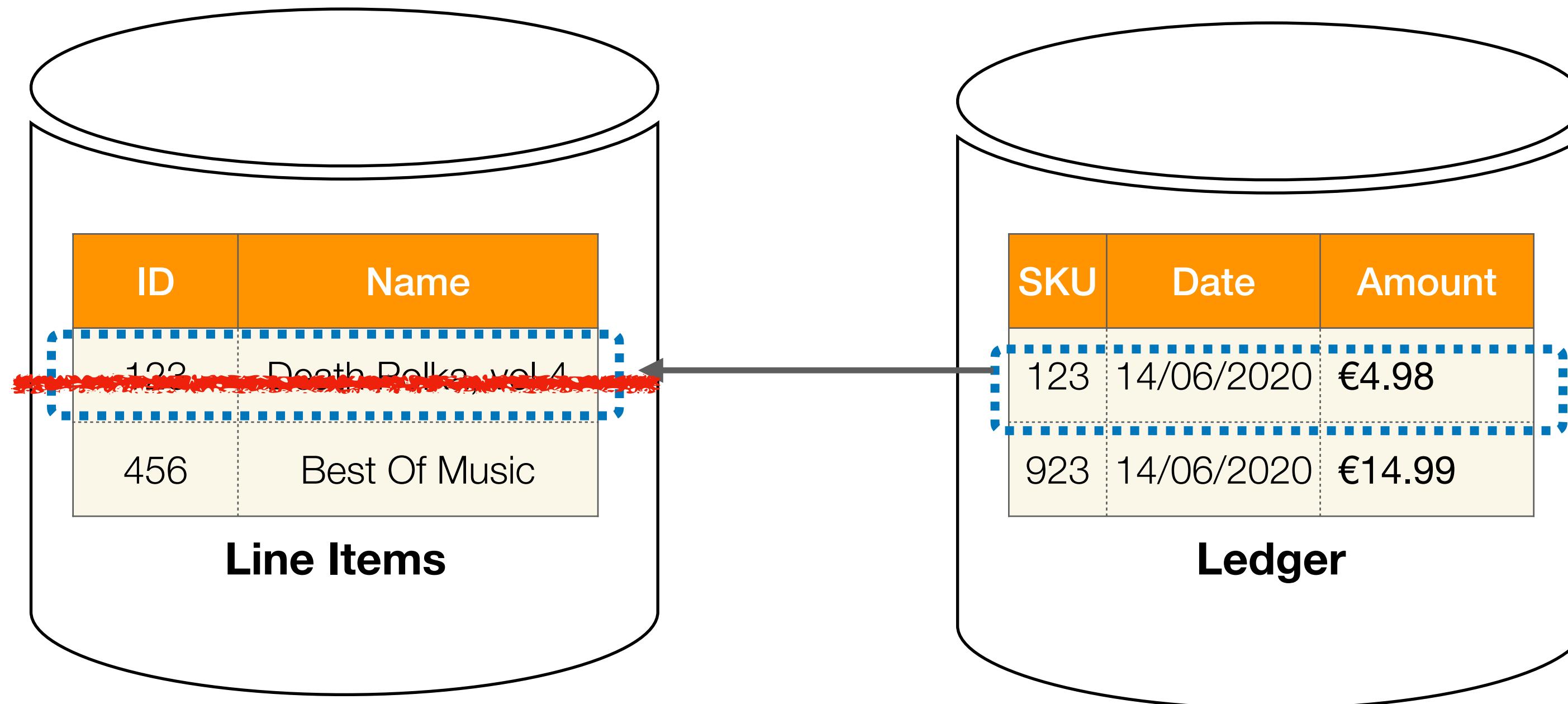
```
urn:tracks:123  
urn:users:123  
urn:comments:123  
urn:artwork:123  
urn:playlists:123
```

https://philcalcado.com/2017/03/22/pattern_using_seudo-uris_with_microservices.html

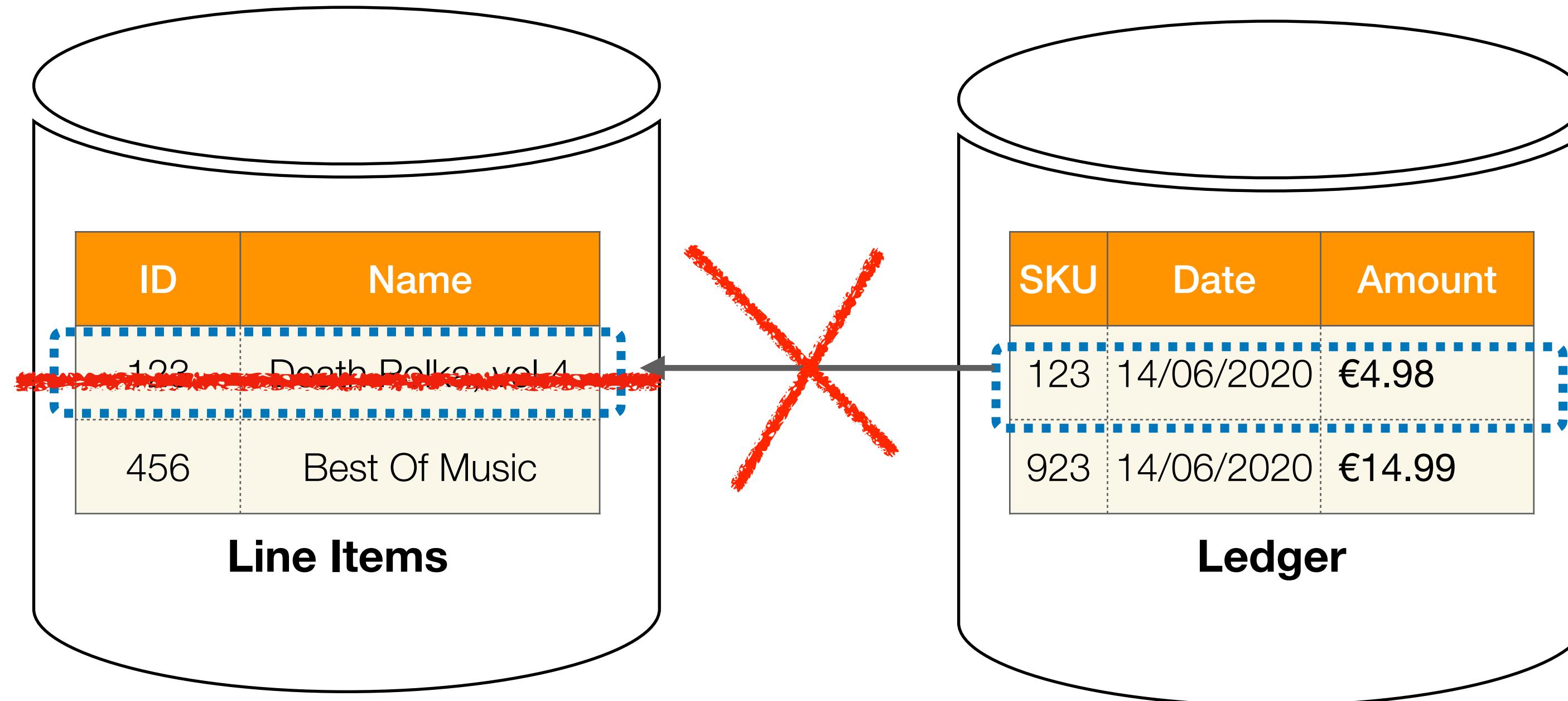
REFERENTIAL INTEGRITY ACROSS DATABASES?



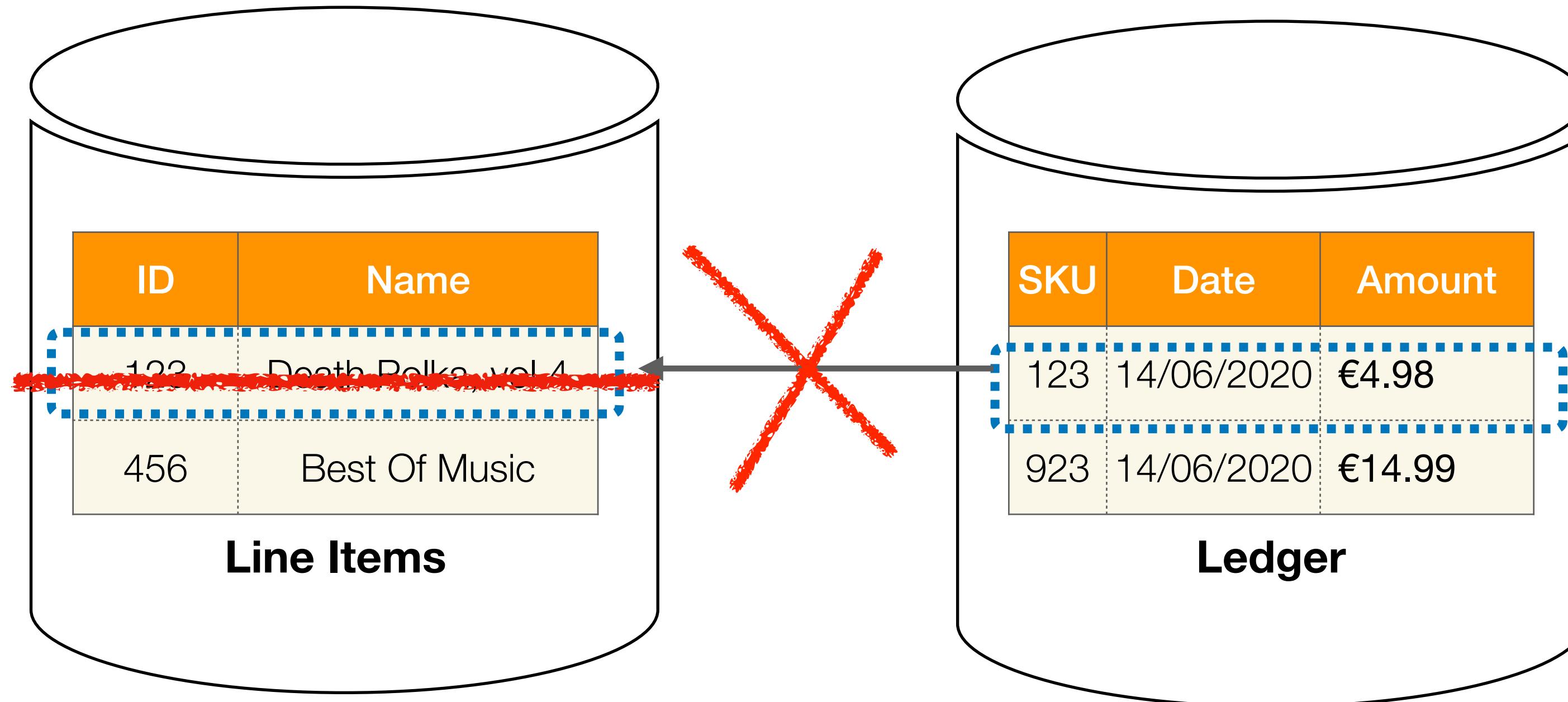
REFERENTIAL INTEGRITY ACROSS DATABASES?



REFERENTIAL INTEGRITY ACROSS DATABASES?

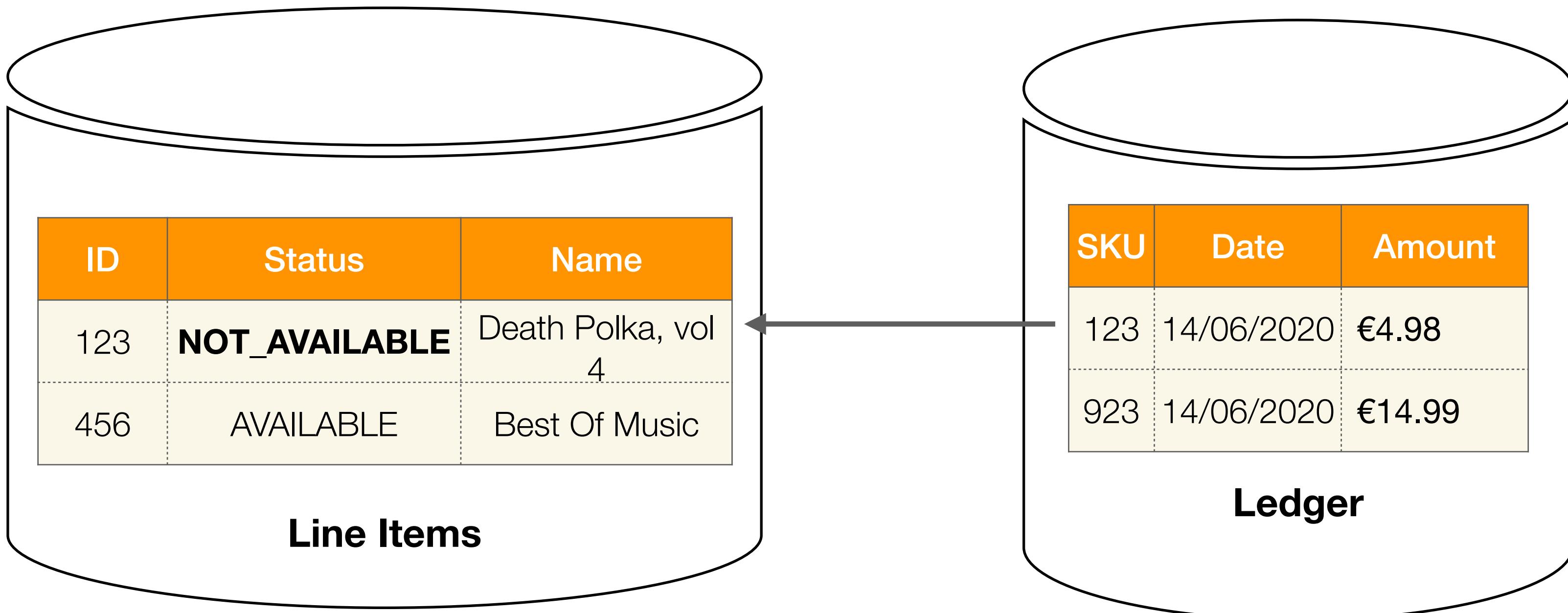


REFERENTIAL INTEGRITY ACROSS DATABASES?

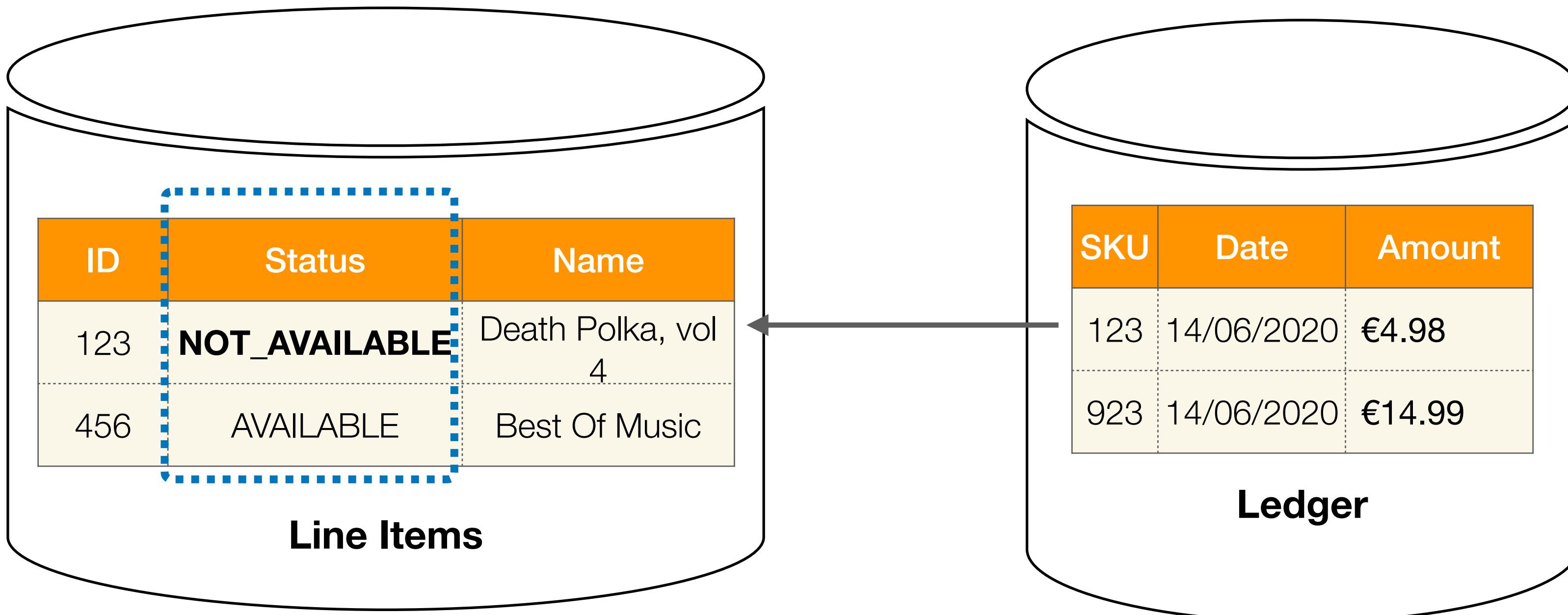


What are our options?

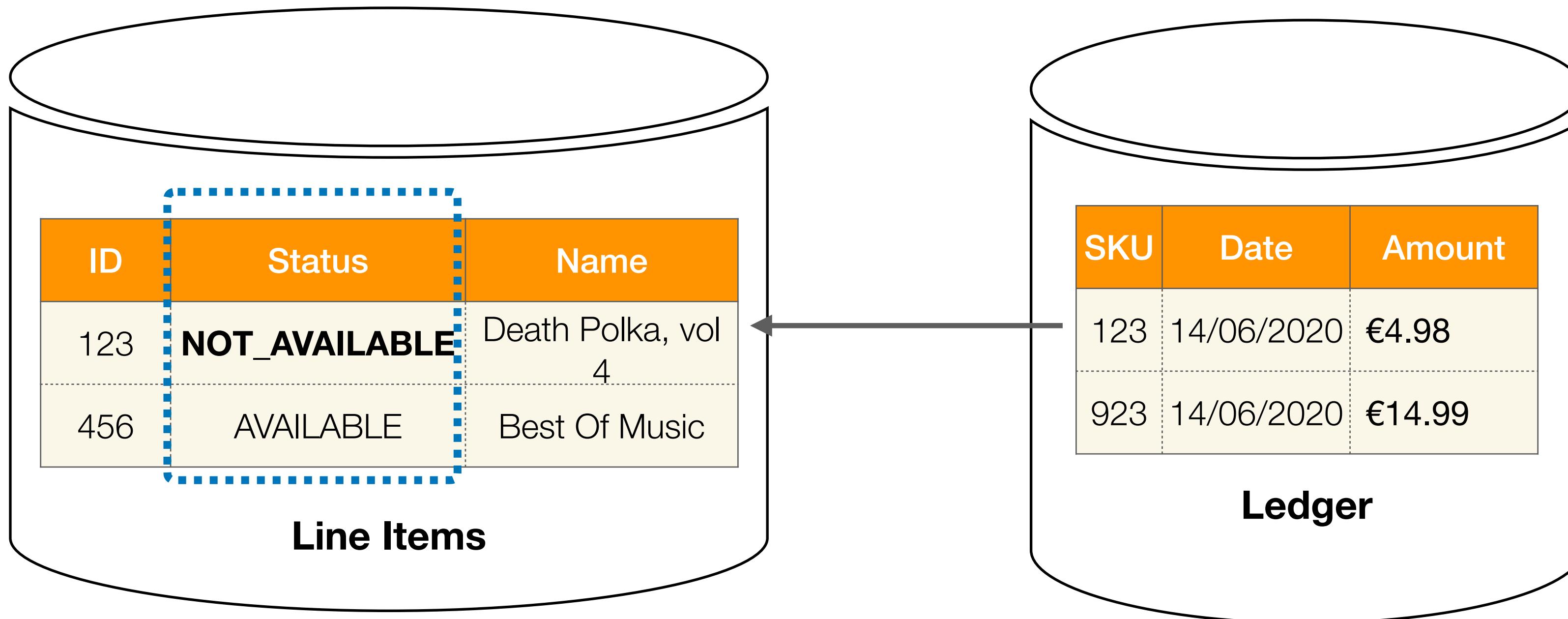
SOFT DELETE



SOFT DELETE



SOFT DELETE



Data not removed, allowing internal references to continue to work

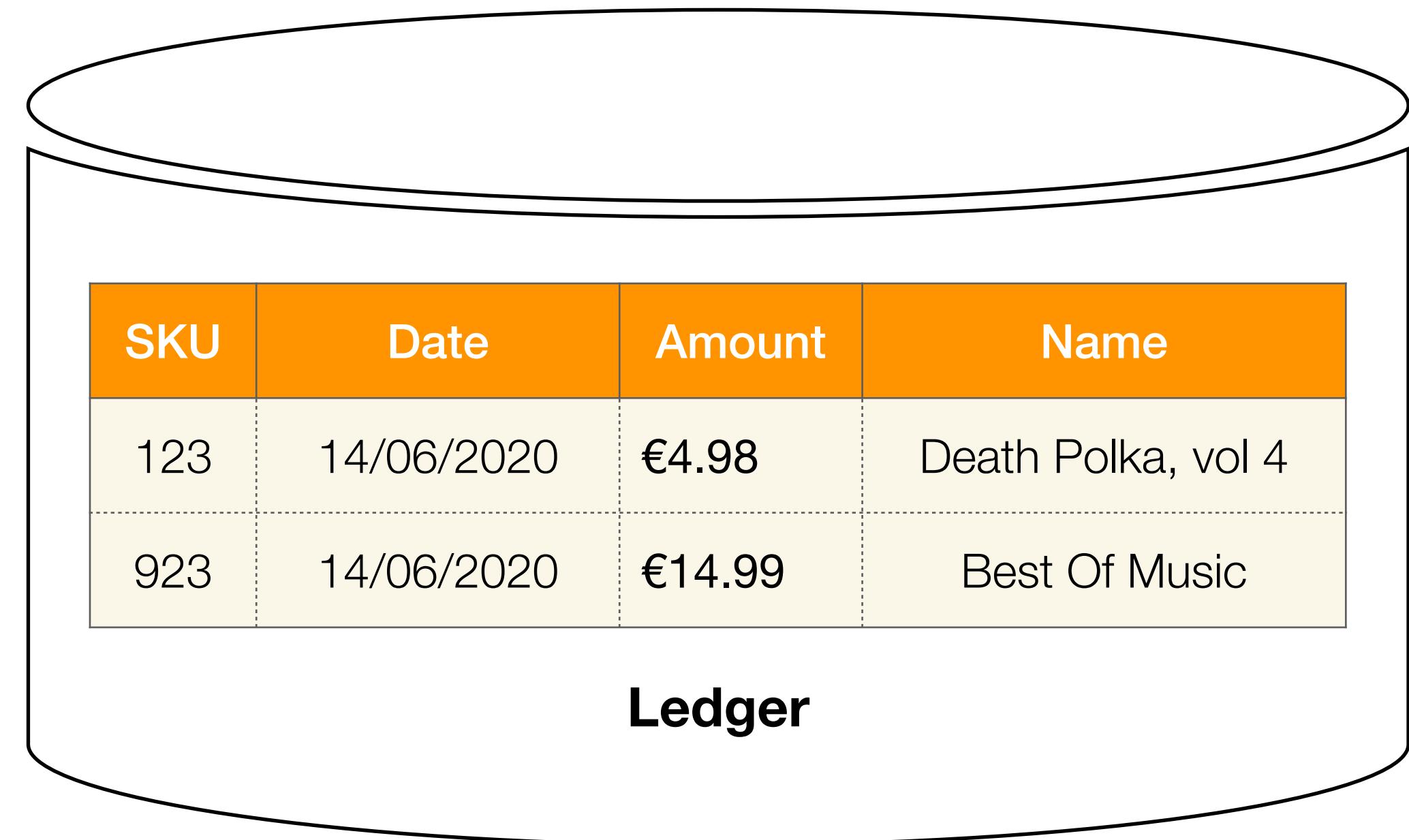
COPY DATA



A cylinder representing the Line Items database. It contains a table with two columns: ID and Name. The first row has an orange header and contains '123' and 'Death Polka, vol 4'. The second row has a light gray background and contains '456' and 'Best Of Music'.

ID	Name
123	Death Polka, vol 4
456	Best Of Music

Line Items



A cylinder representing the Ledger database. It contains a table with four columns: SKU, Date, Amount, and Name. The first row has an orange header and contains '123', '14/06/2020', '€4.98', and 'Death Polka, vol 4'. The second row has a light gray background and contains '923', '14/06/2020', '€14.99', and 'Best Of Music'.

SKU	Date	Amount	Name
123	14/06/2020	€4.98	Death Polka, vol 4
923	14/06/2020	€14.99	Best Of Music

Ledger

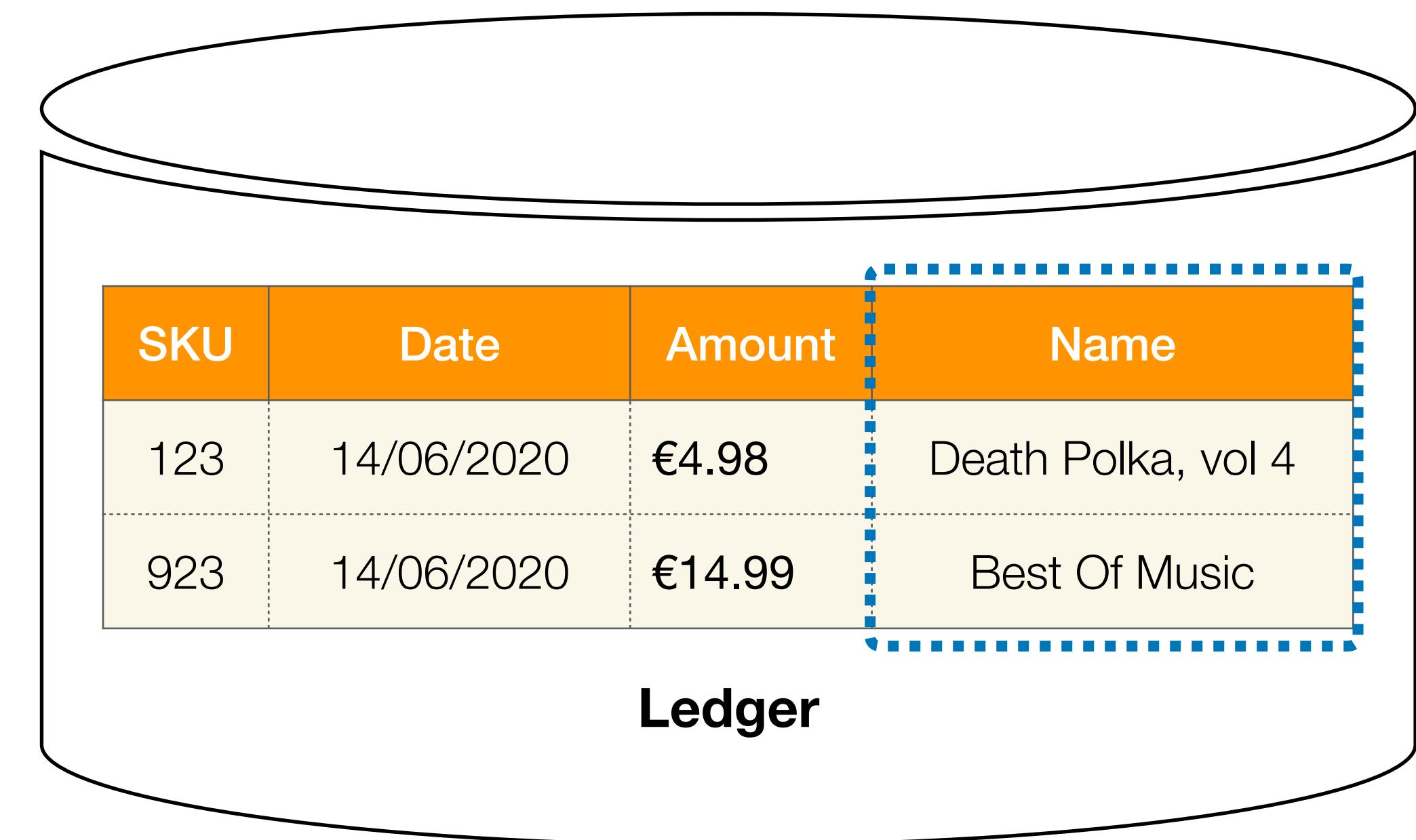
COPY DATA



A cylinder representing the "Line Items" database. It contains a table with two columns: "ID" and "Name". The data shows two items: "Death Polka, vol 4" with ID 123 and "Best Of Music" with ID 456.

ID	Name
123	Death Polka, vol 4
456	Best Of Music

Line Items

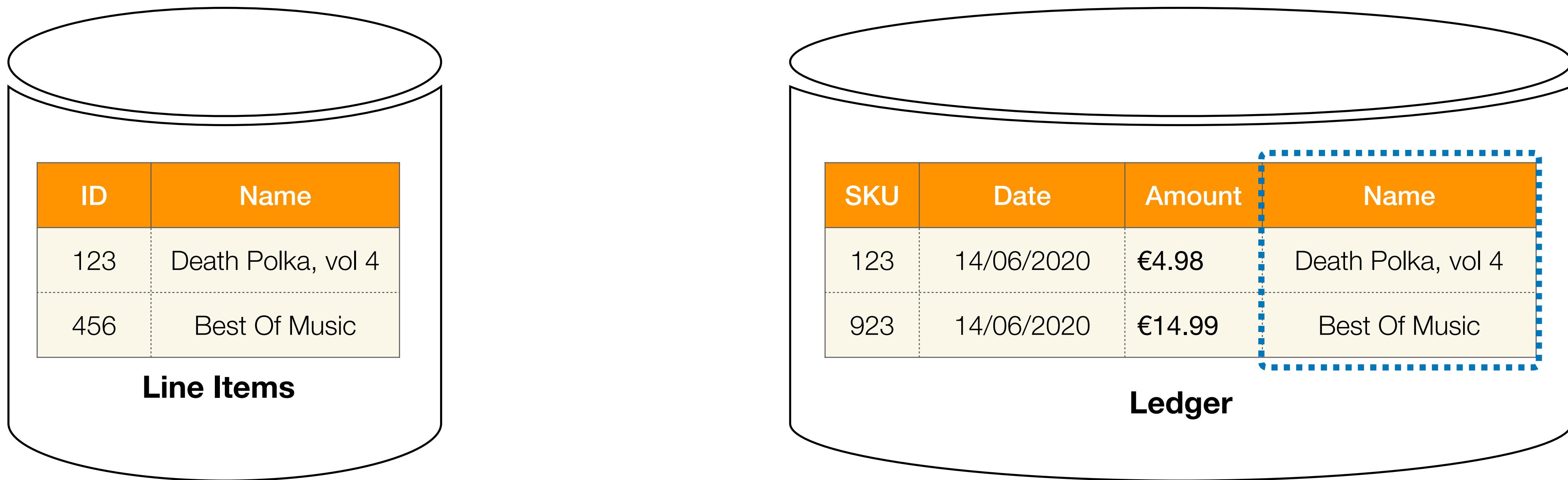


A cylinder representing the "Ledger" database. It contains a table with four columns: "SKU", "Date", "Amount", and "Name". The data shows two entries: one for "Death Polka, vol 4" with SKU 123, date 14/06/2020, and amount €4.98, and another for "Best Of Music" with SKU 923, date 14/06/2020, and amount €14.99. A blue dashed box highlights the "Name" column.

SKU	Date	Amount	Name
123	14/06/2020	€4.98	Death Polka, vol 4
923	14/06/2020	€14.99	Best Of Music

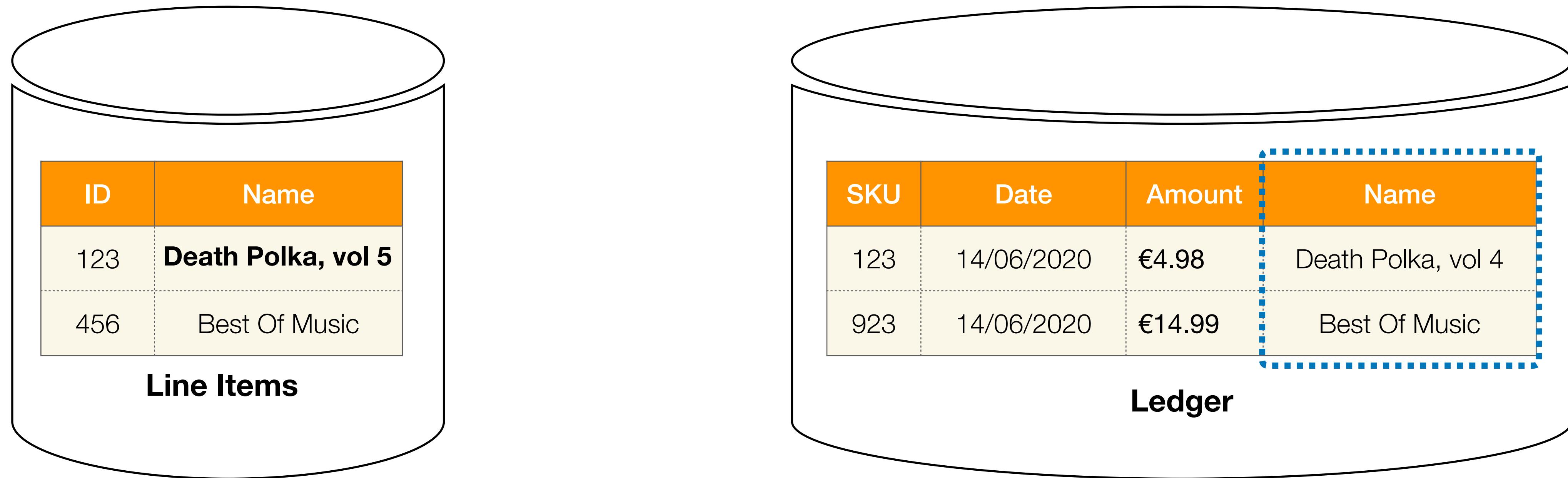
Ledger

COPY DATA



Avoids the need for the join if the name is the only thing needed

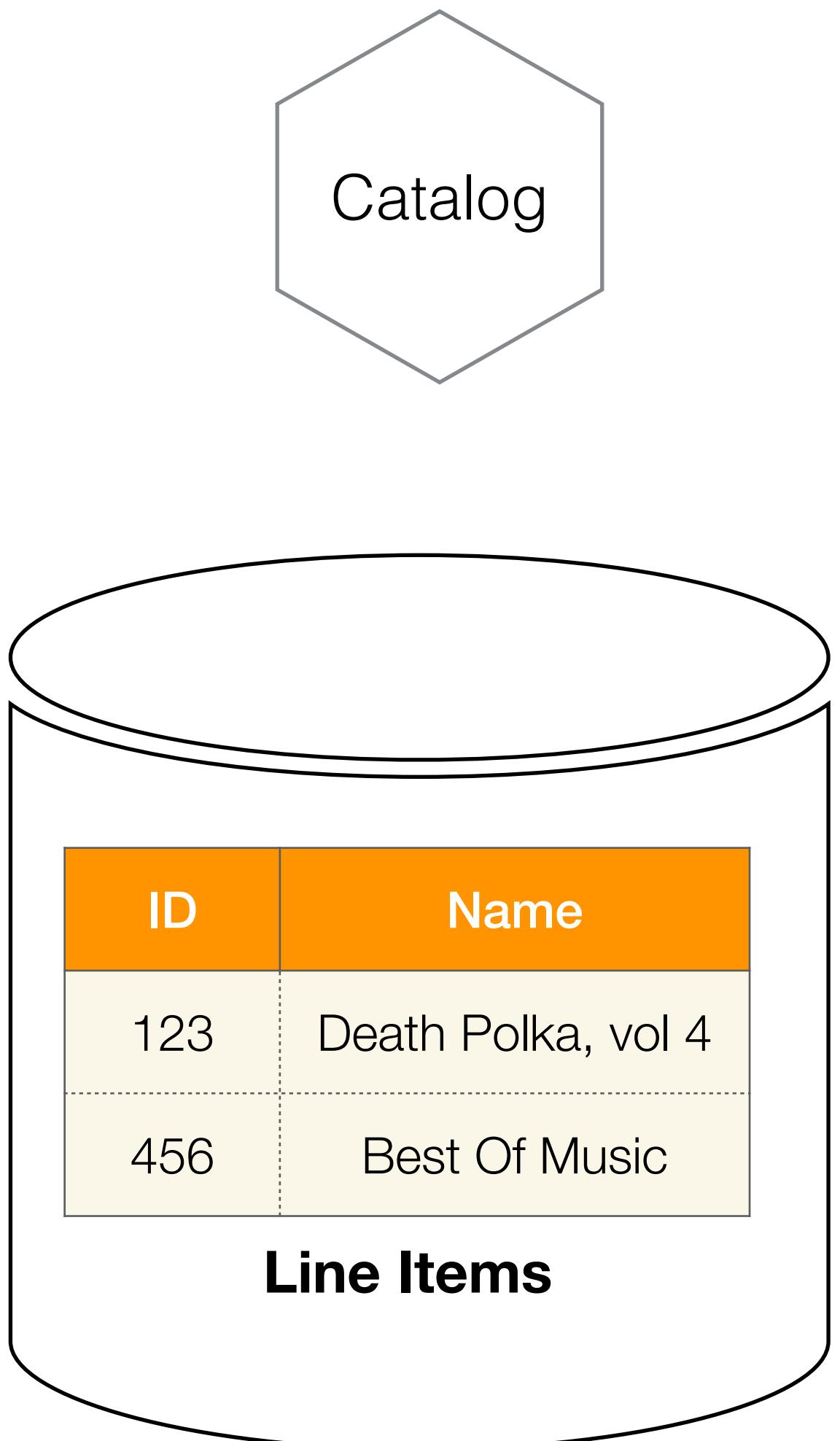
COPY DATA



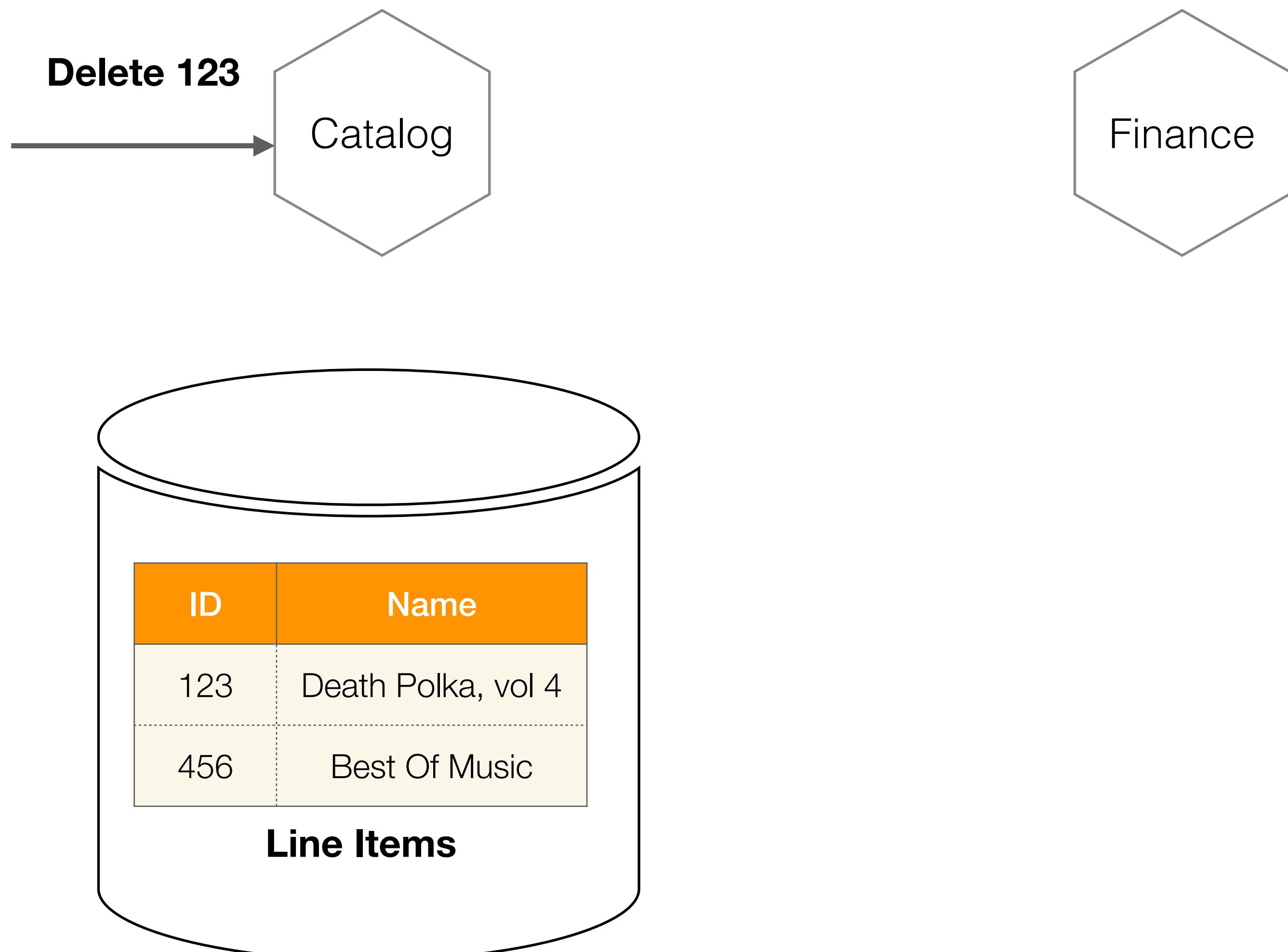
Avoids the need for the join if the name is the only thing needed

But what about if the source data changes?

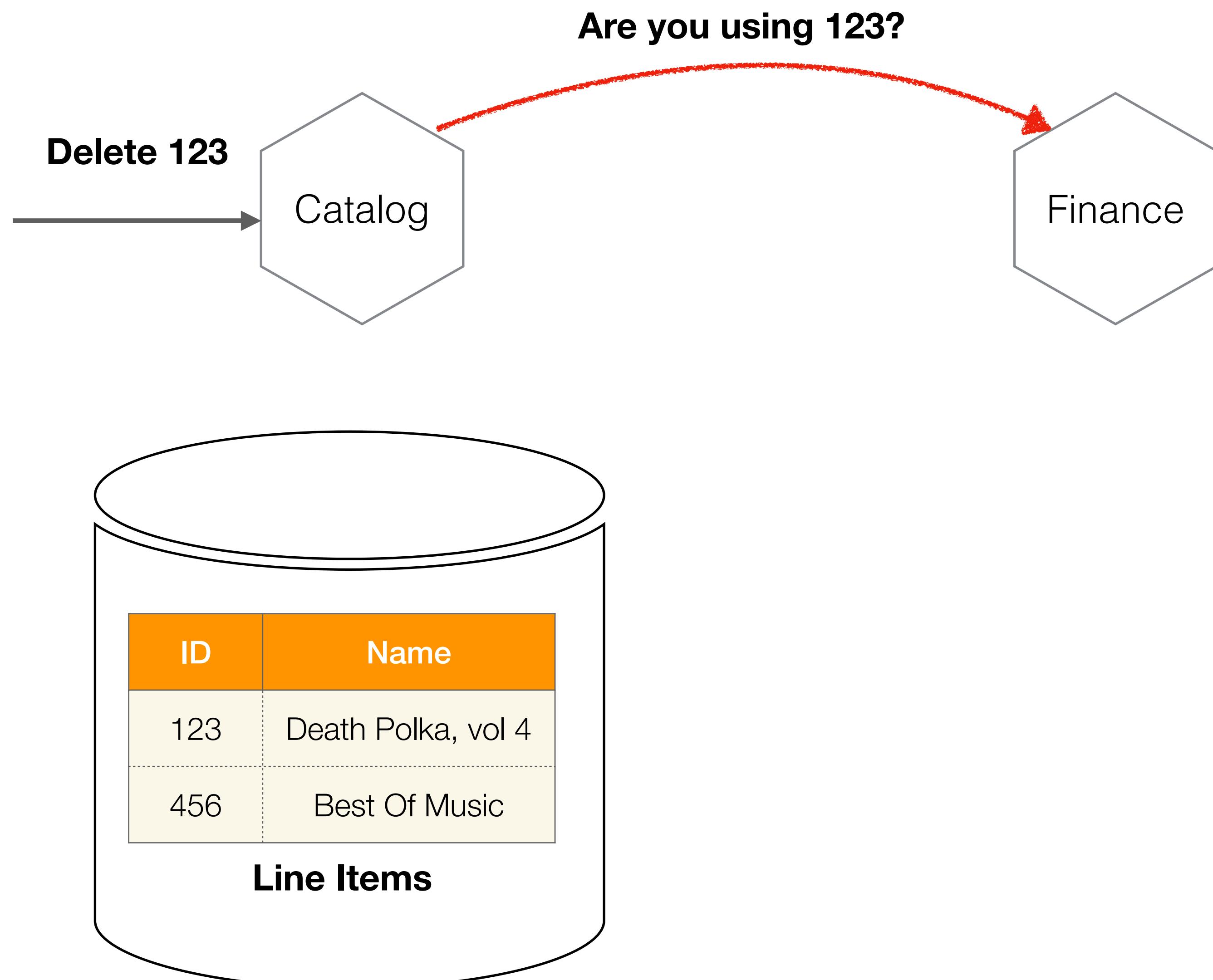
CHECK FOR USE BEFORE DELETE?



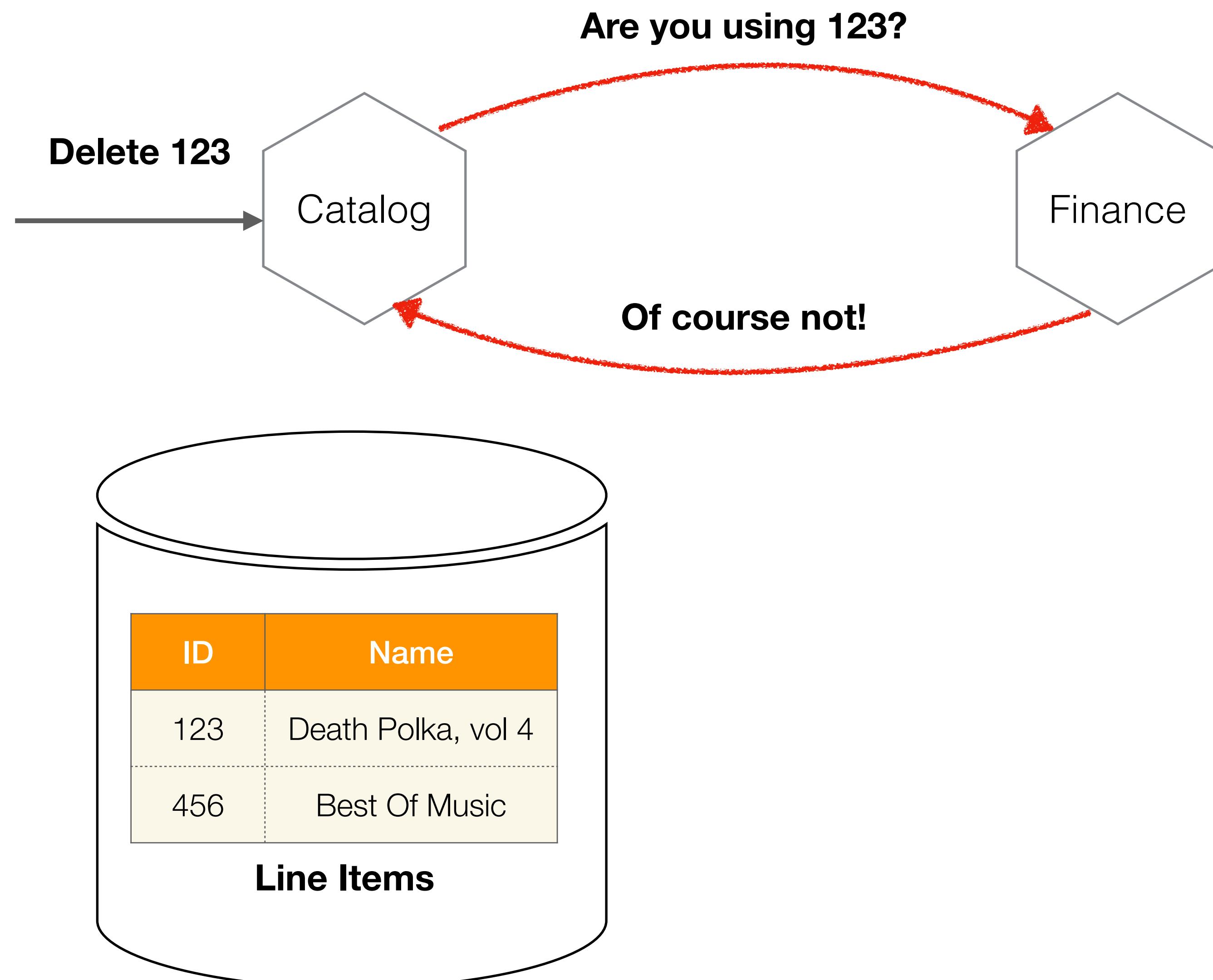
CHECK FOR USE BEFORE DELETE?



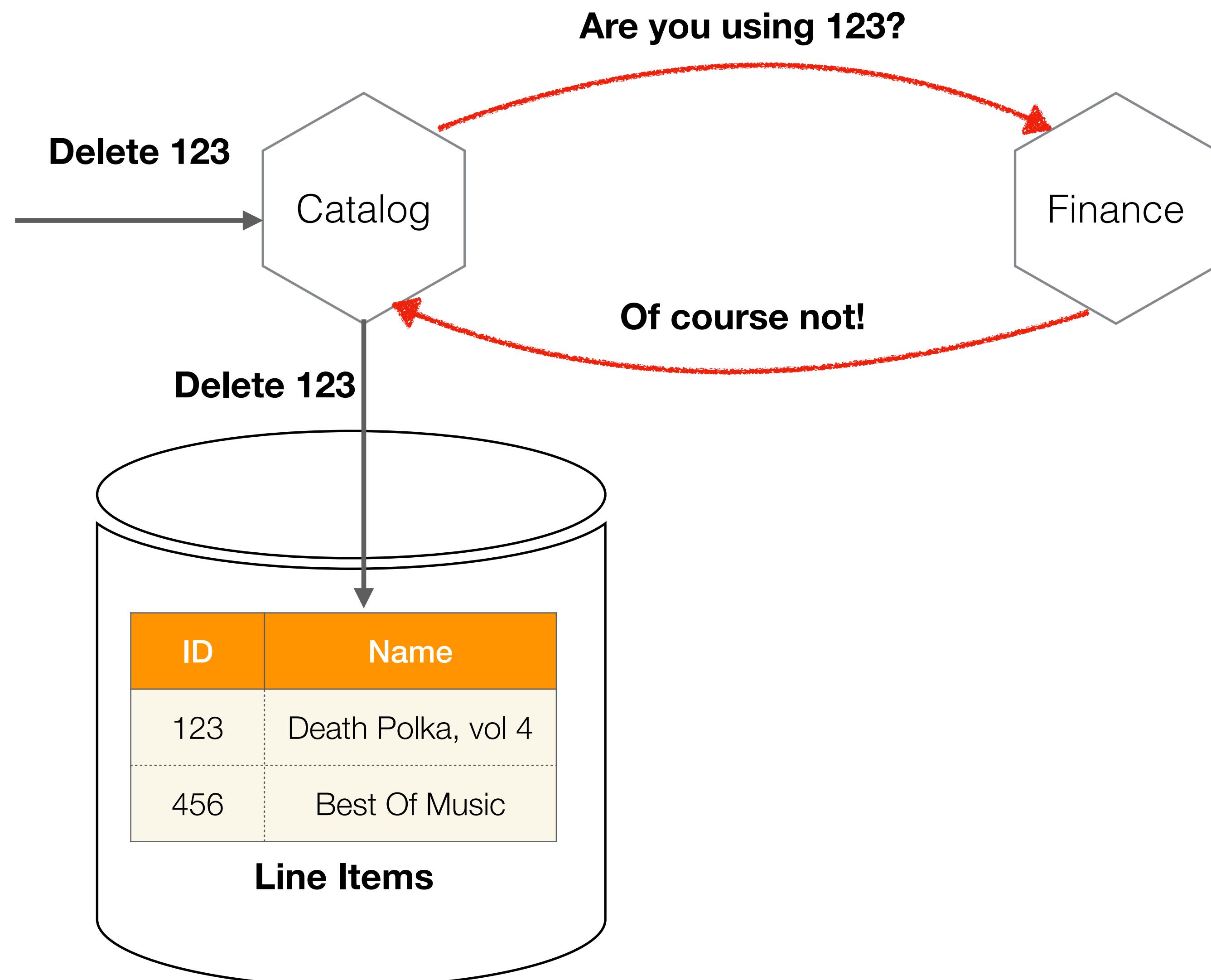
CHECK FOR USE BEFORE DELETE?



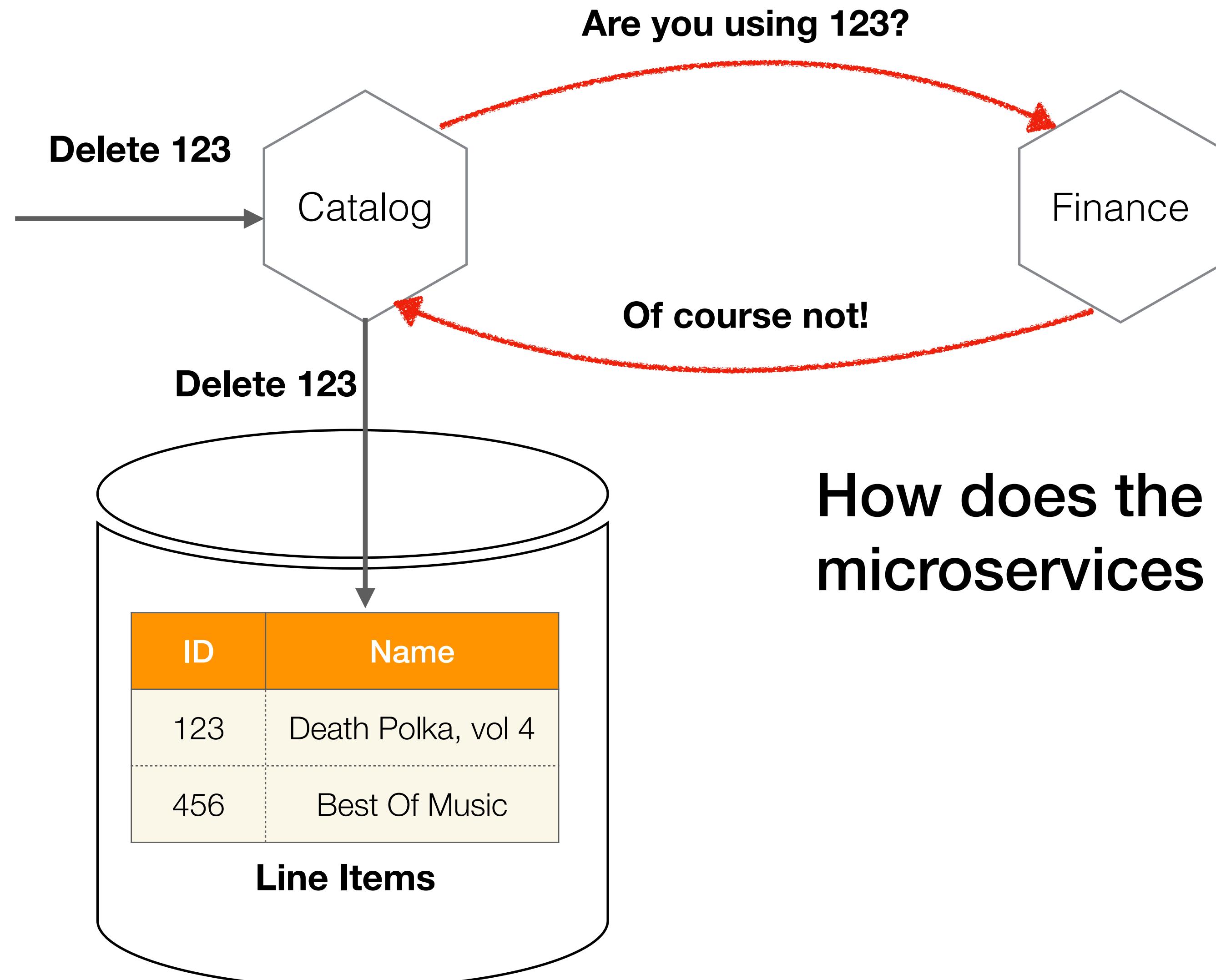
CHECK FOR USE BEFORE DELETE?



CHECK FOR USE BEFORE DELETE?

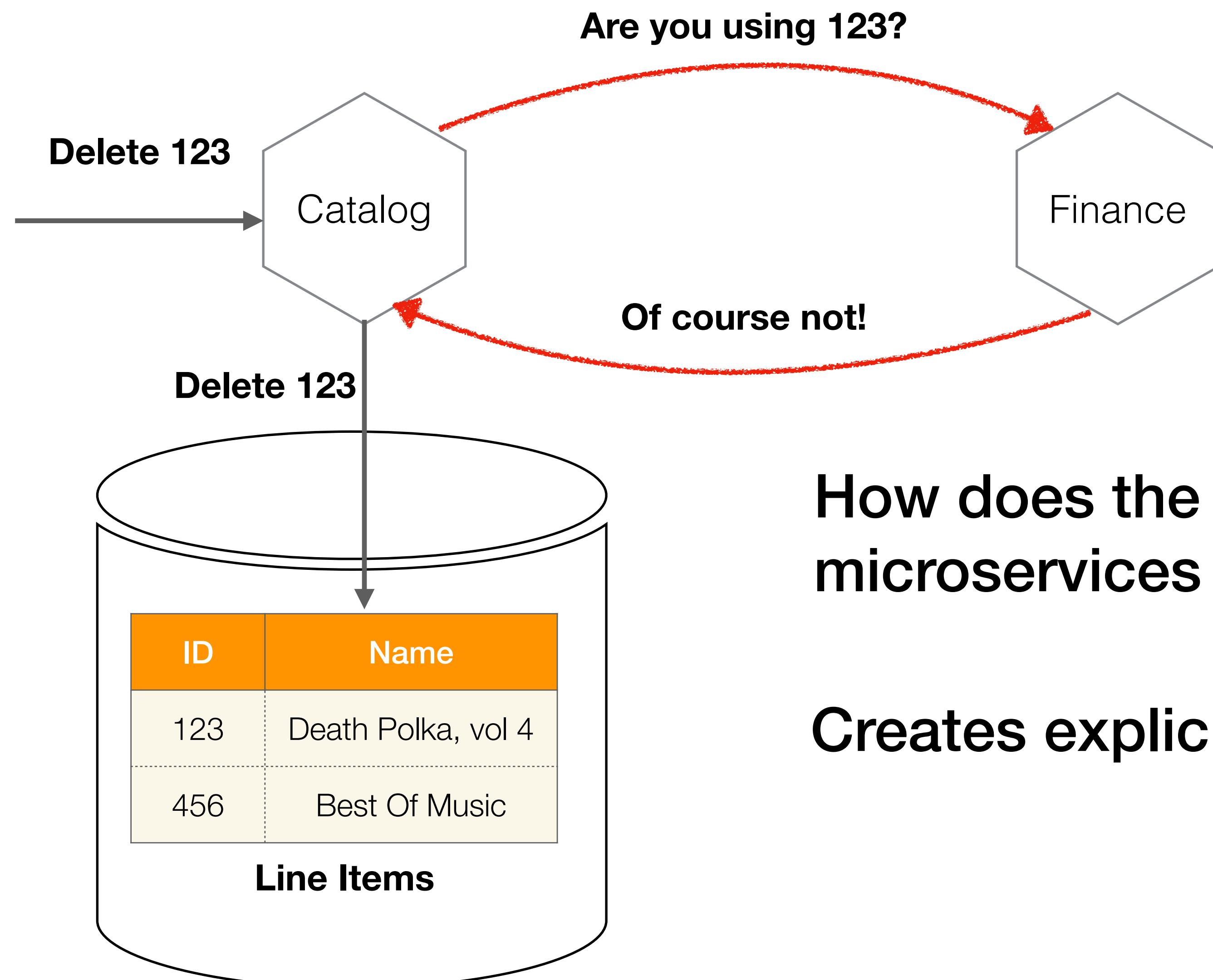


CHECK FOR USE BEFORE DELETE?



How does the catalog know which microservices to ask?

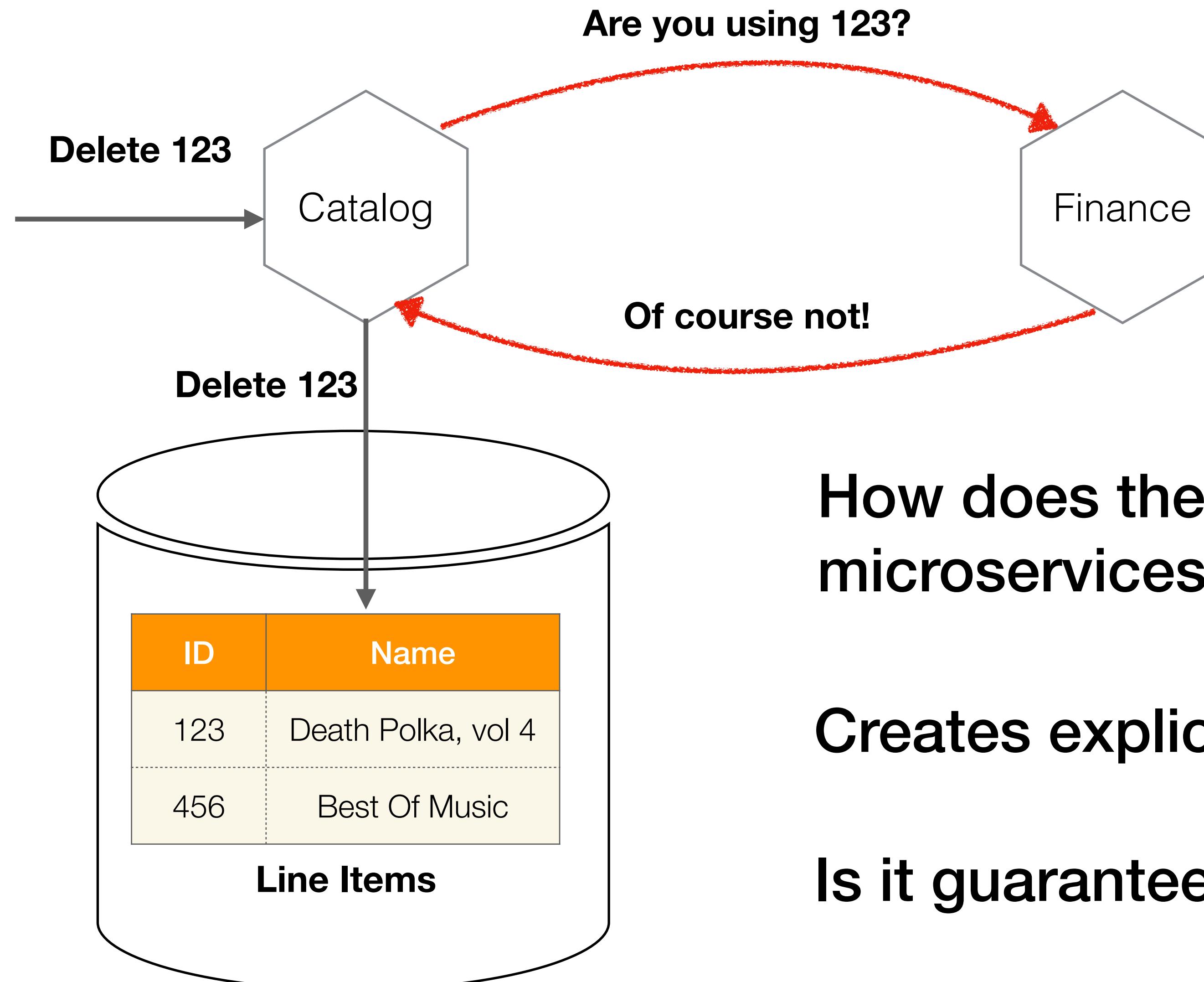
CHECK FOR USE BEFORE DELETE?



How does the catalog know which microservices to ask?

Creates explicit circular dependencies

CHECK FOR USE BEFORE DELETE?



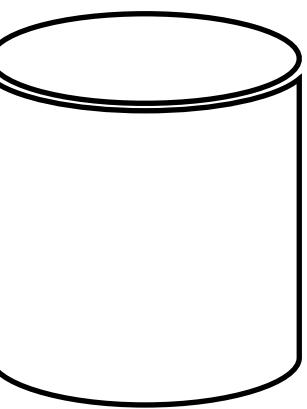
How does the catalog know which microservices to ask?

Creates explicit circular dependencies

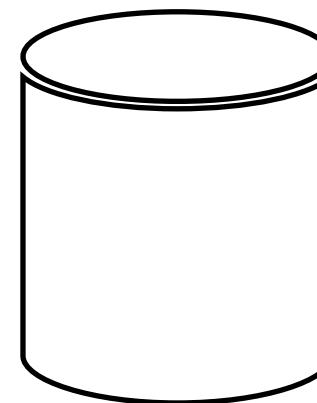
Is it guaranteed to work?

LOCKING REQUIRED?

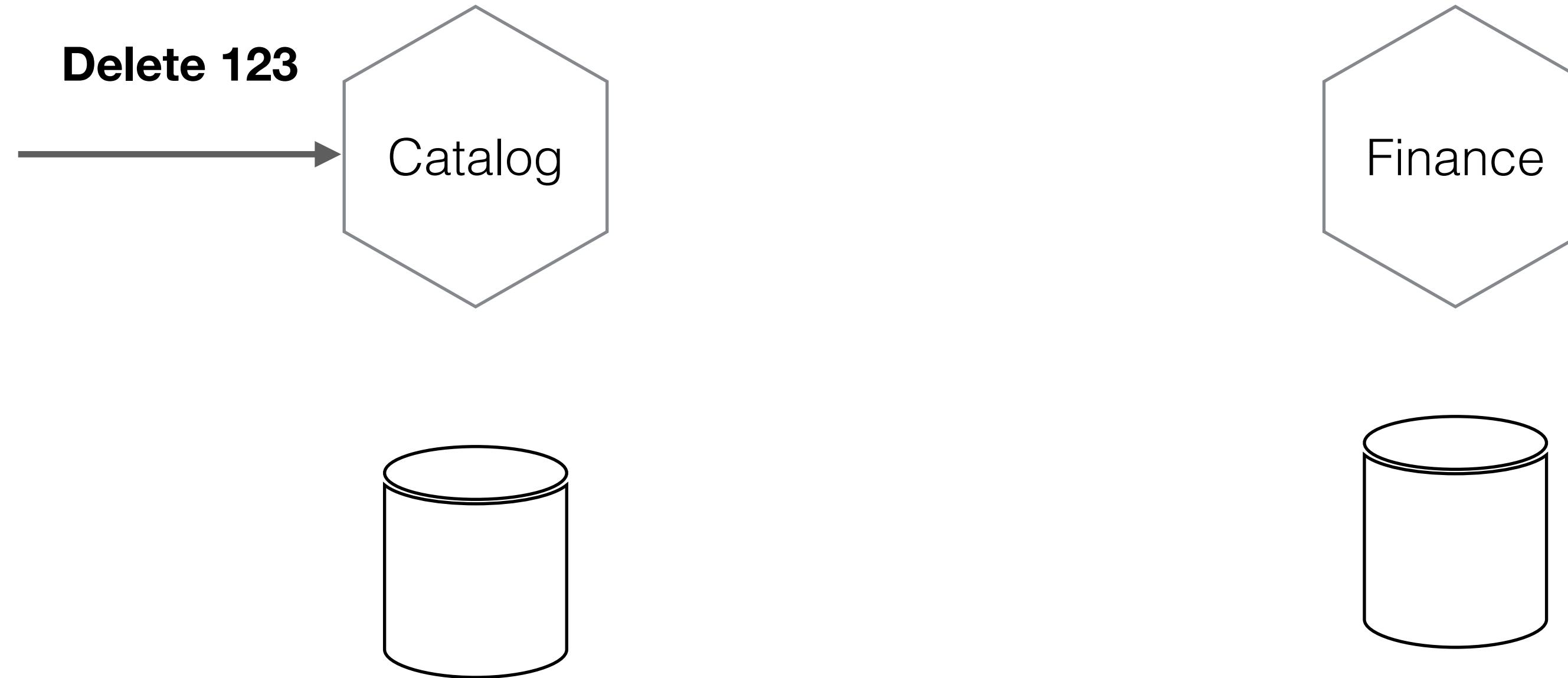
Catalog



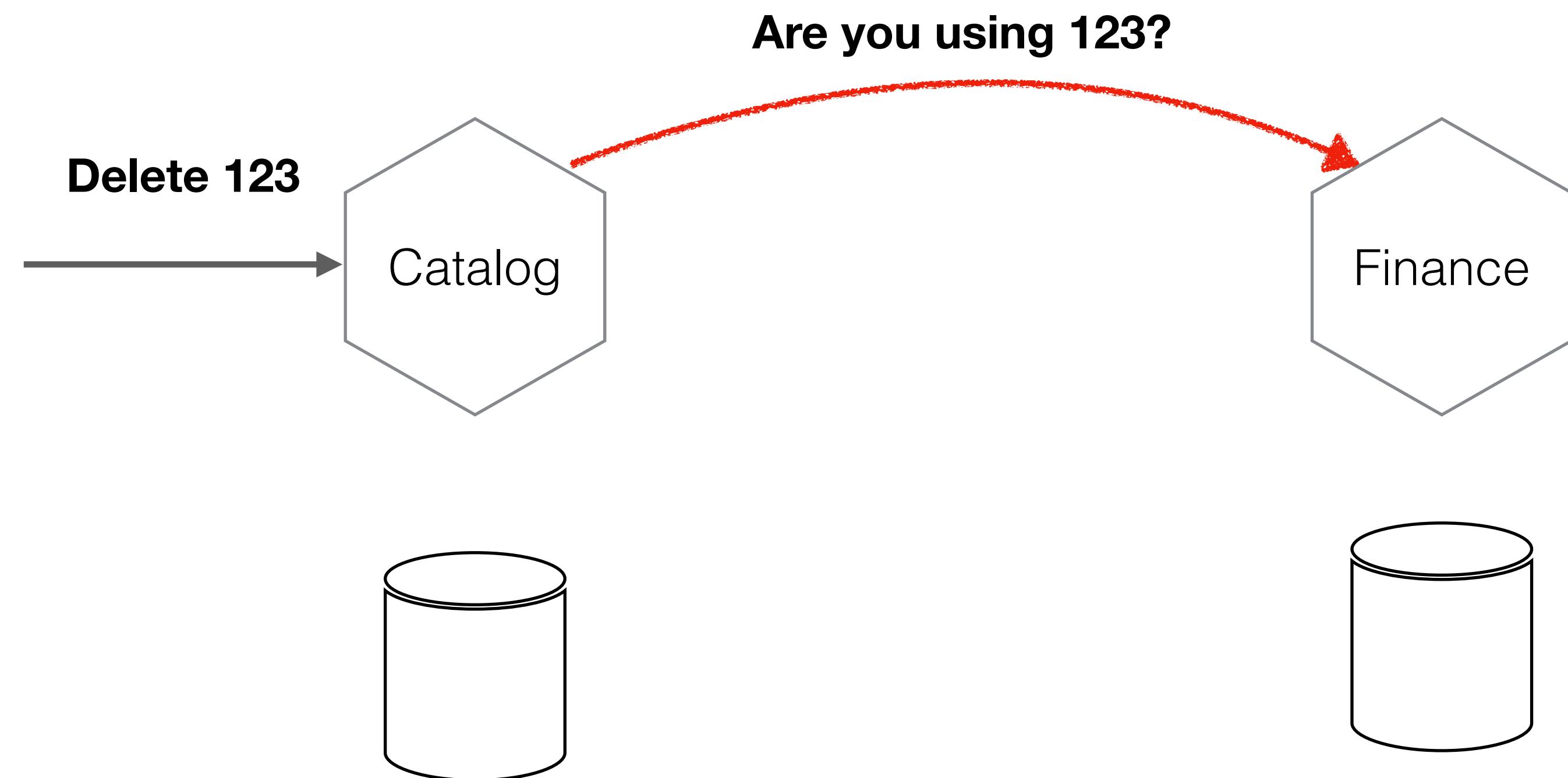
Finance



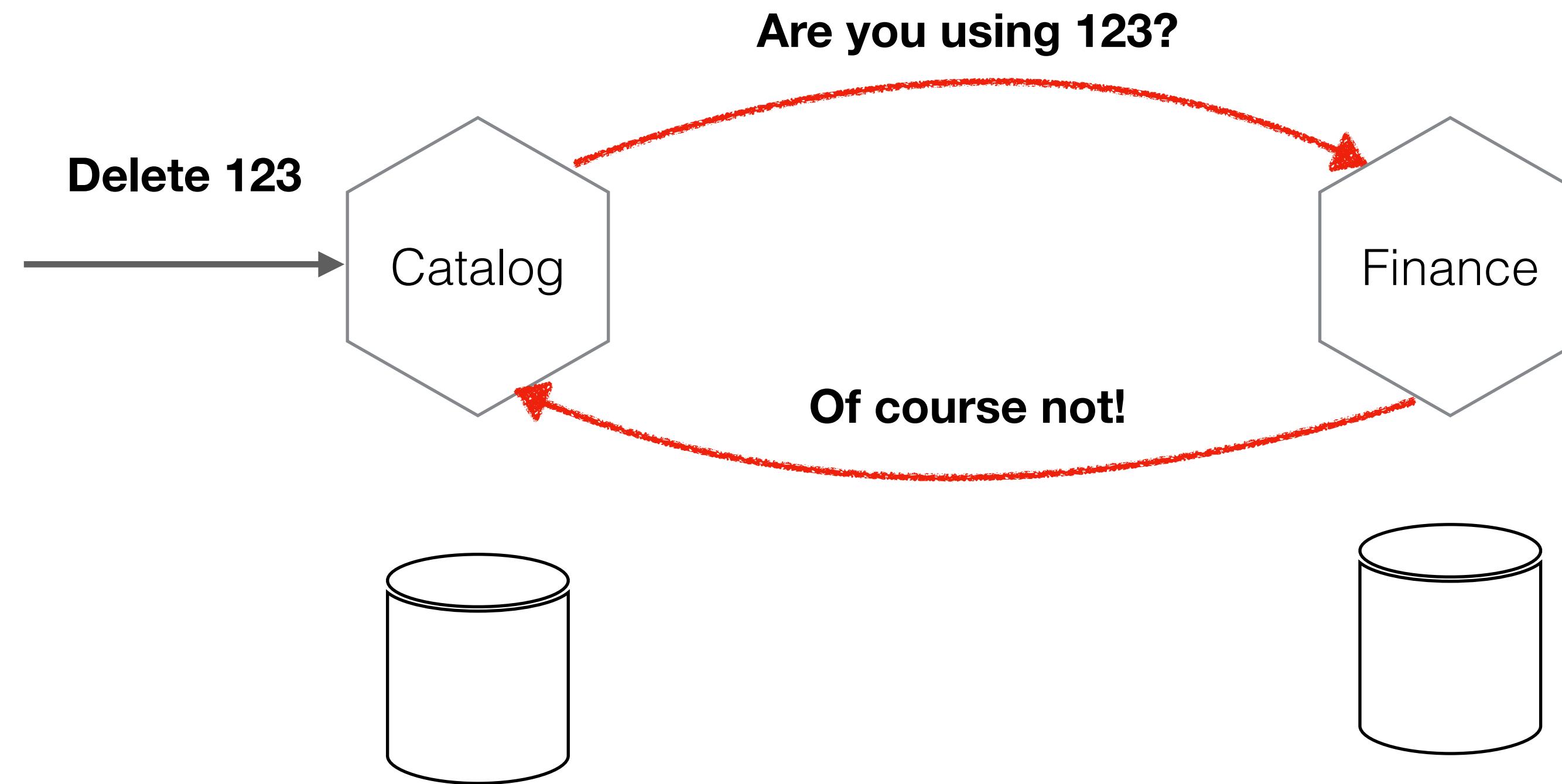
LOCKING REQUIRED?



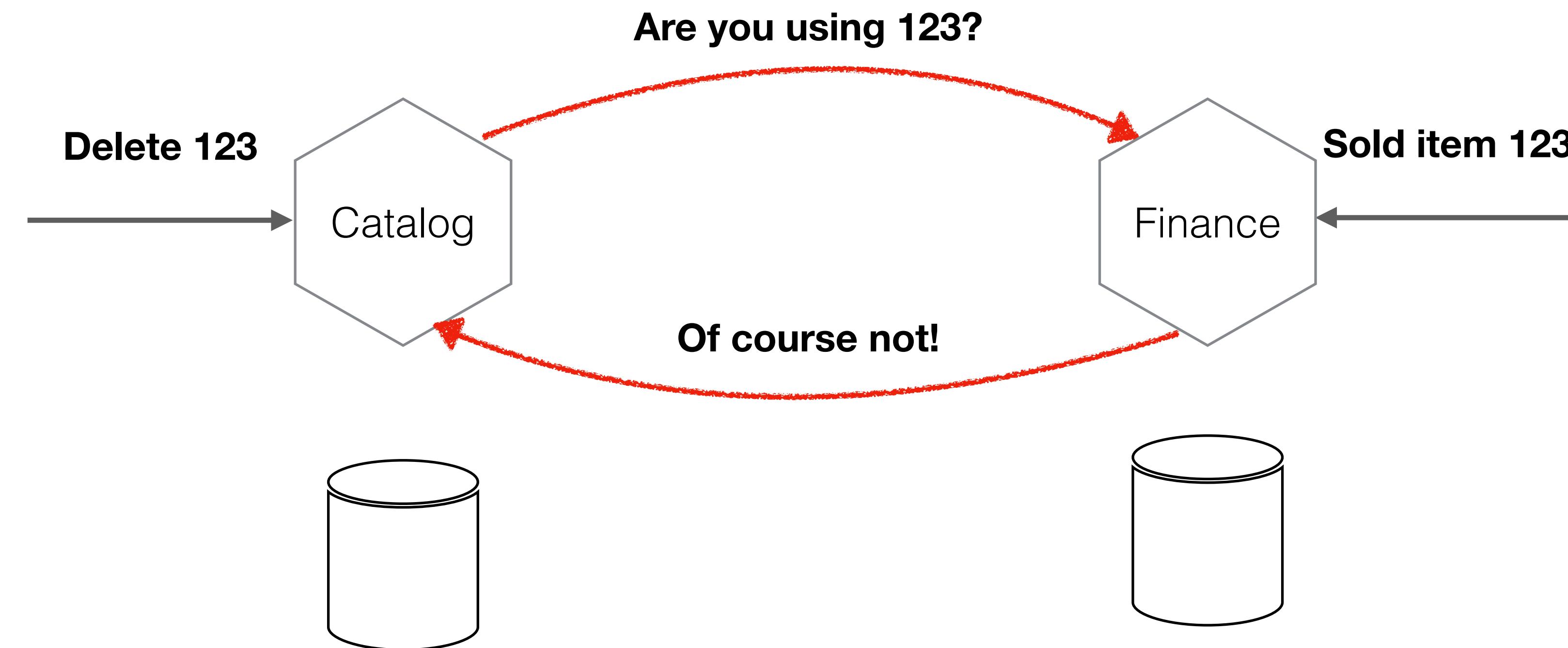
LOCKING REQUIRED?



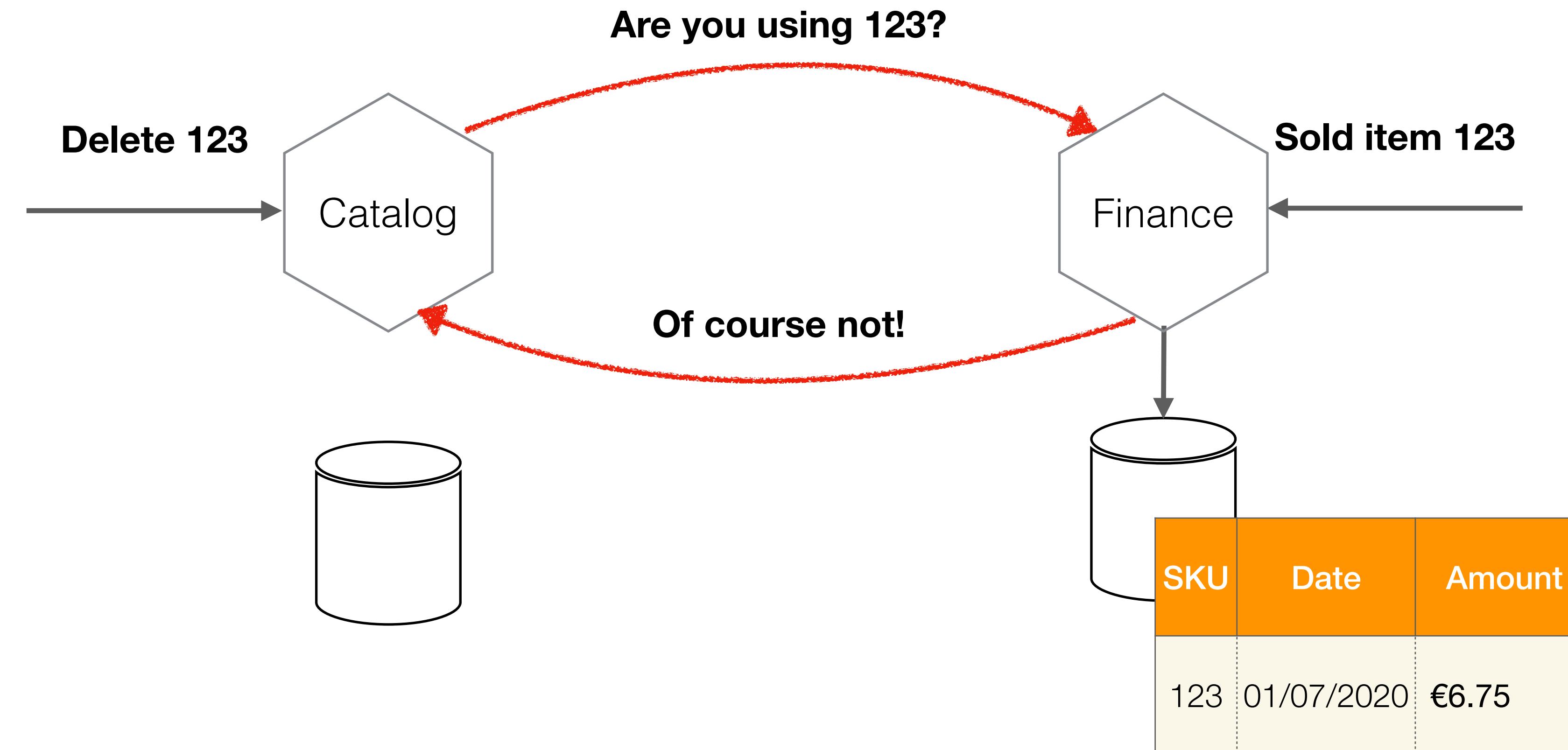
LOCKING REQUIRED?



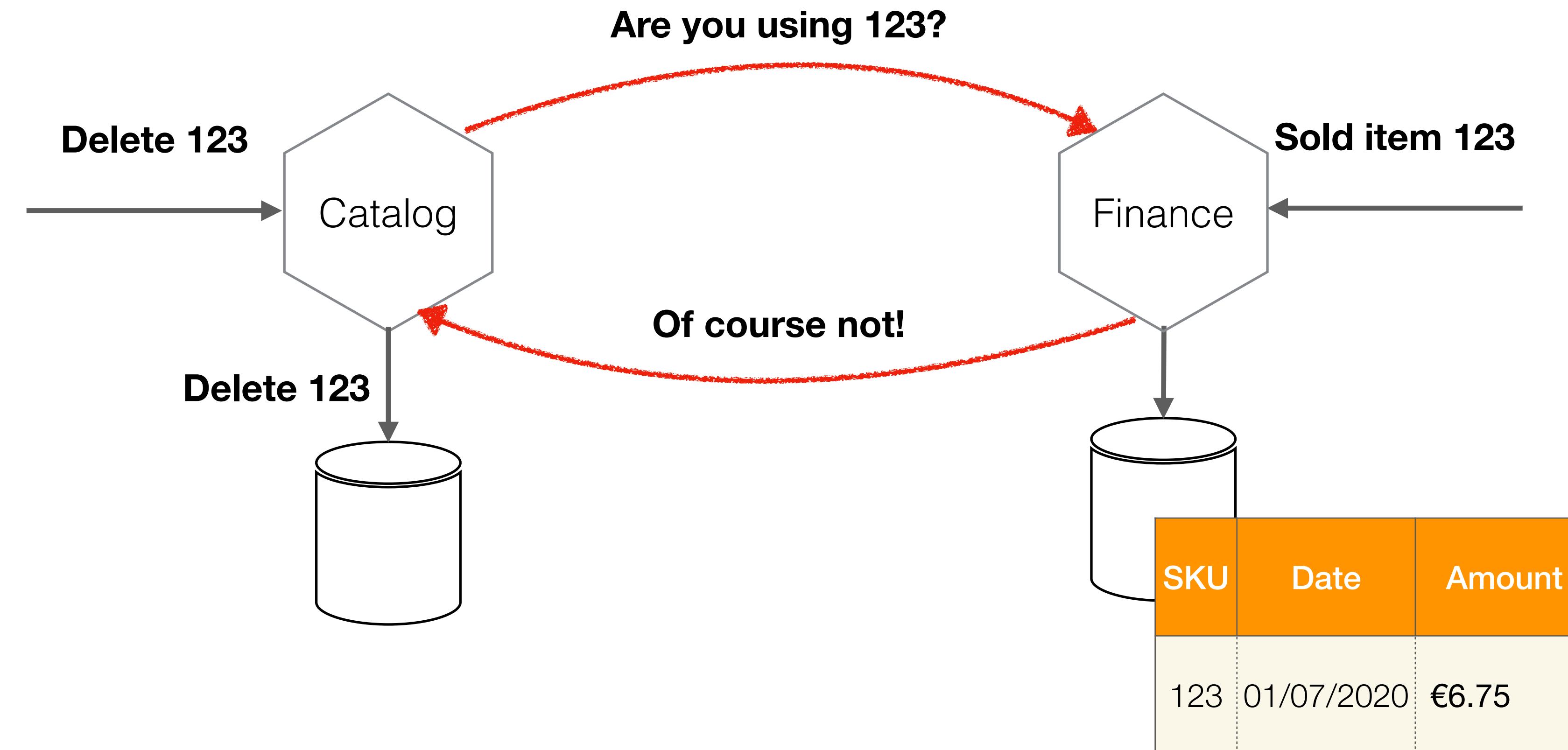
LOCKING REQUIRED?



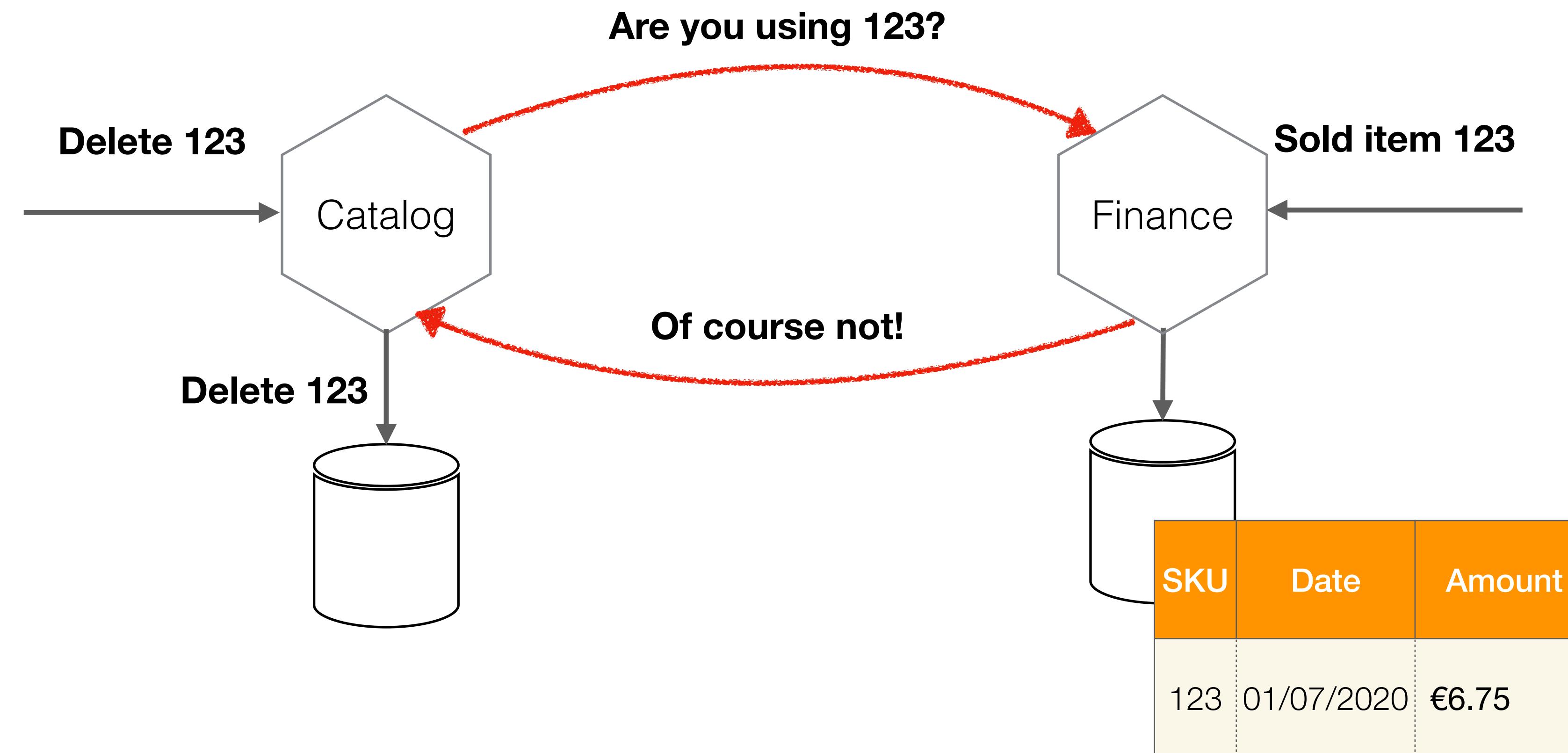
LOCKING REQUIRED?



LOCKING REQUIRED?

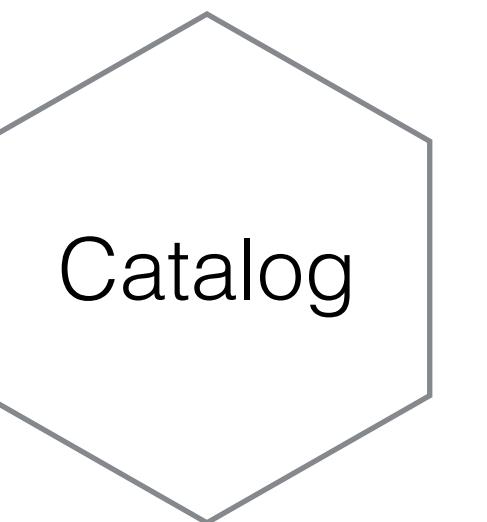


LOCKING REQUIRED?



You need to lock the ledger table if you want to be certain that no references to 123 exist

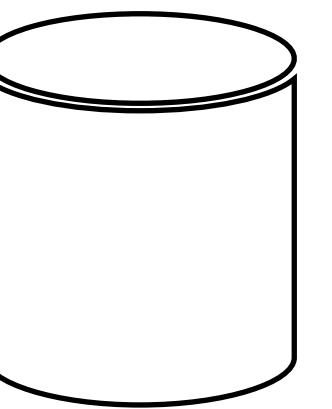
CASCADING DELETES?



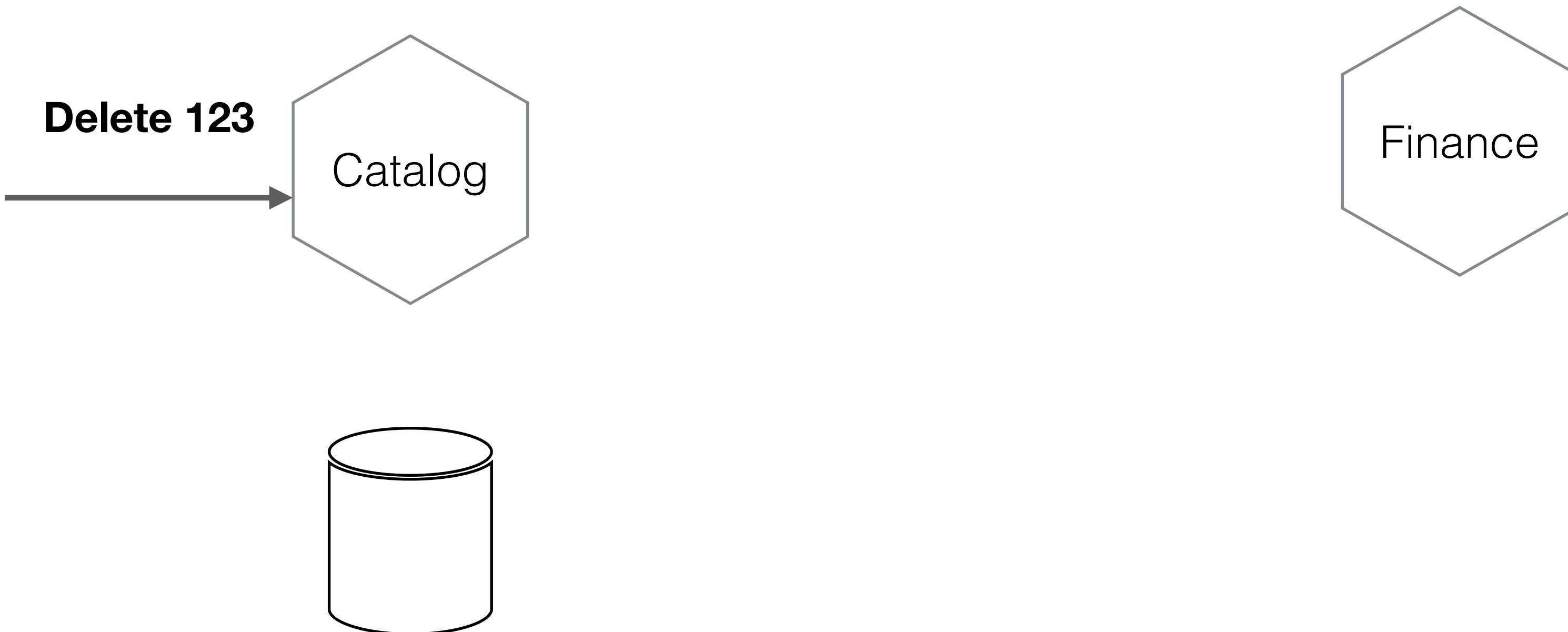
Catalog



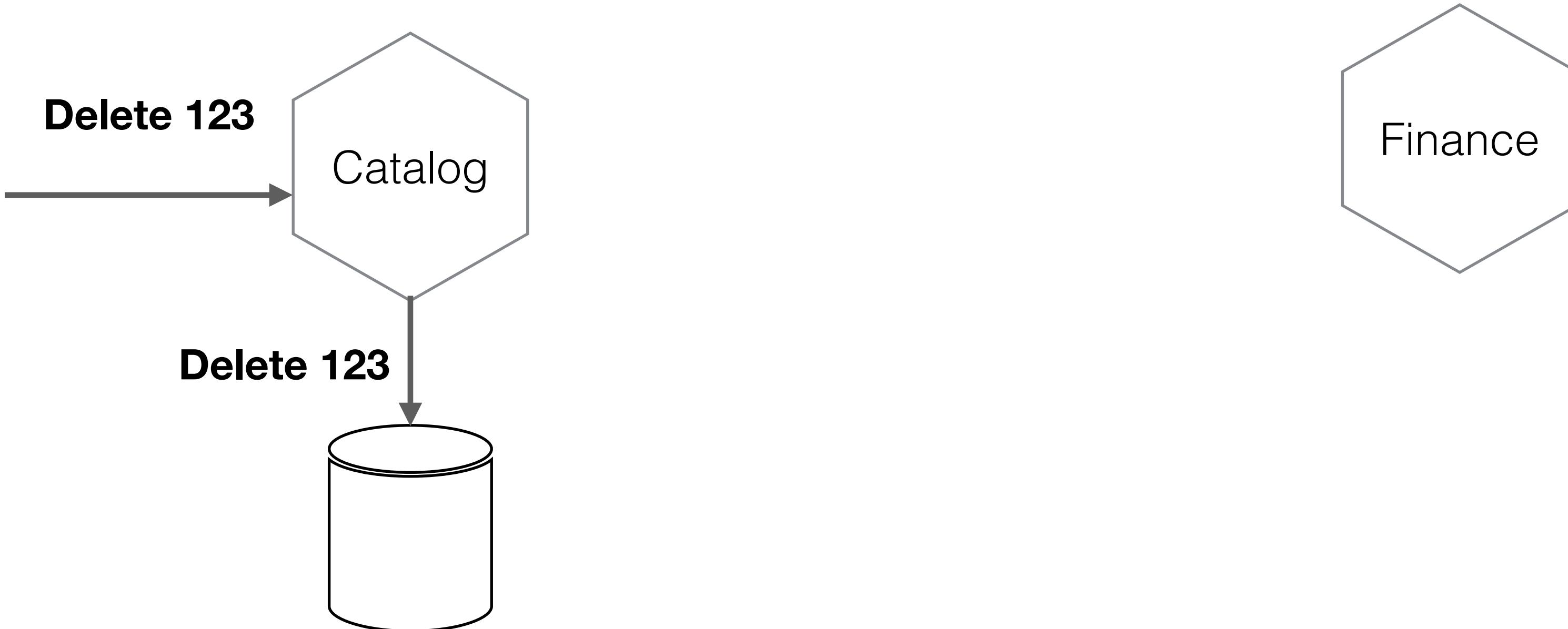
Finance



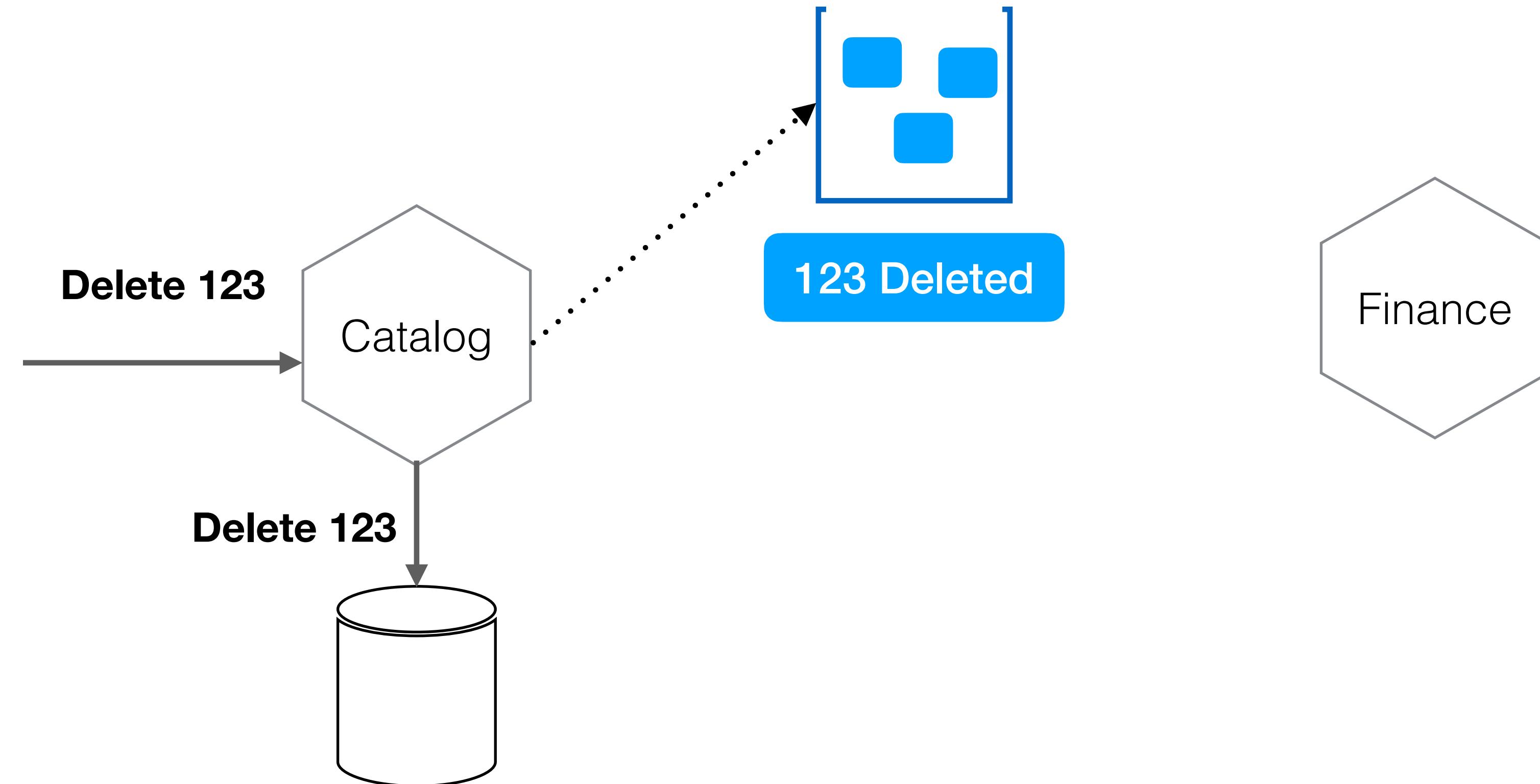
CASCADING DELETES?



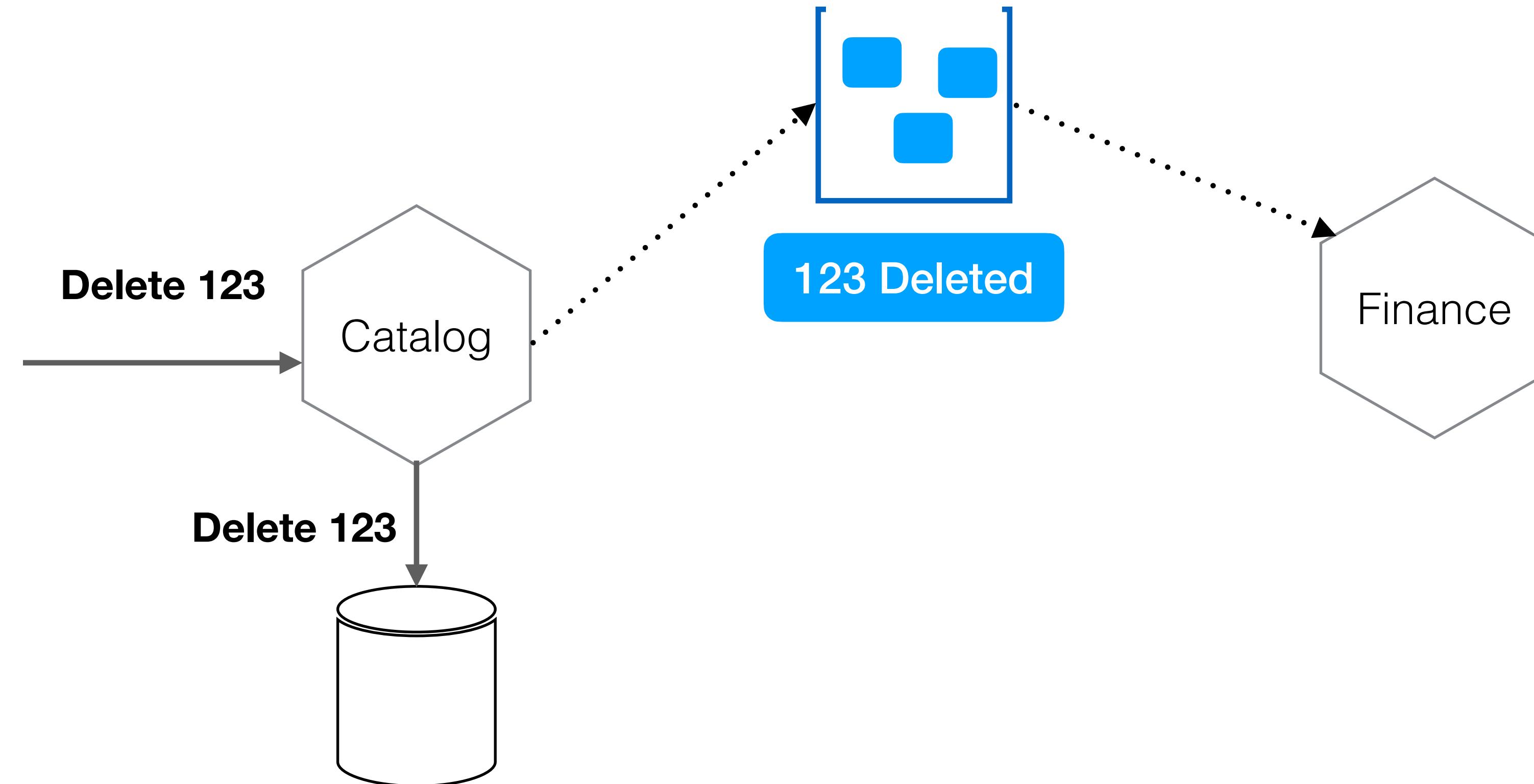
CASCADING DELETES?



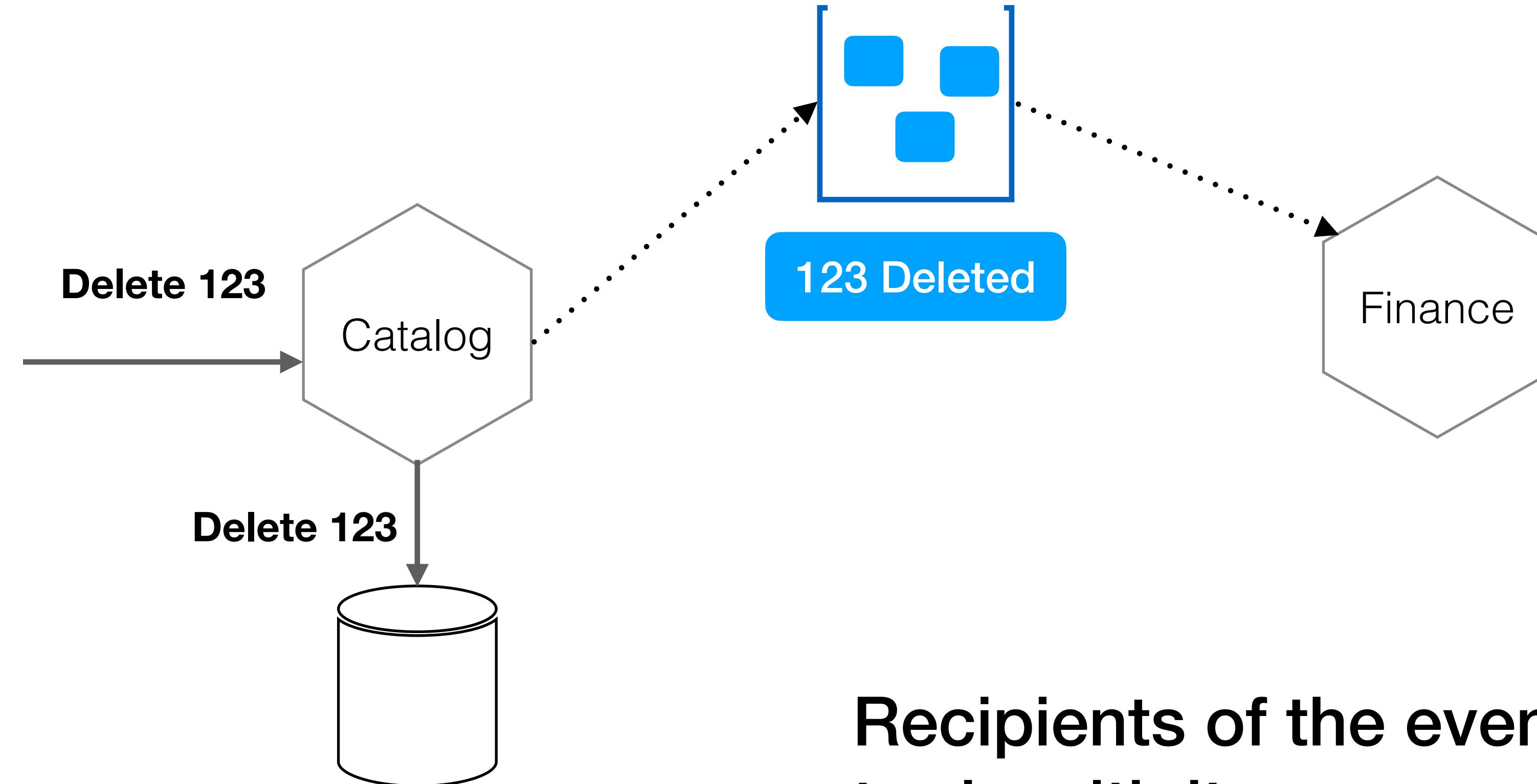
CASCADING DELETES?



CASCADING DELETES?

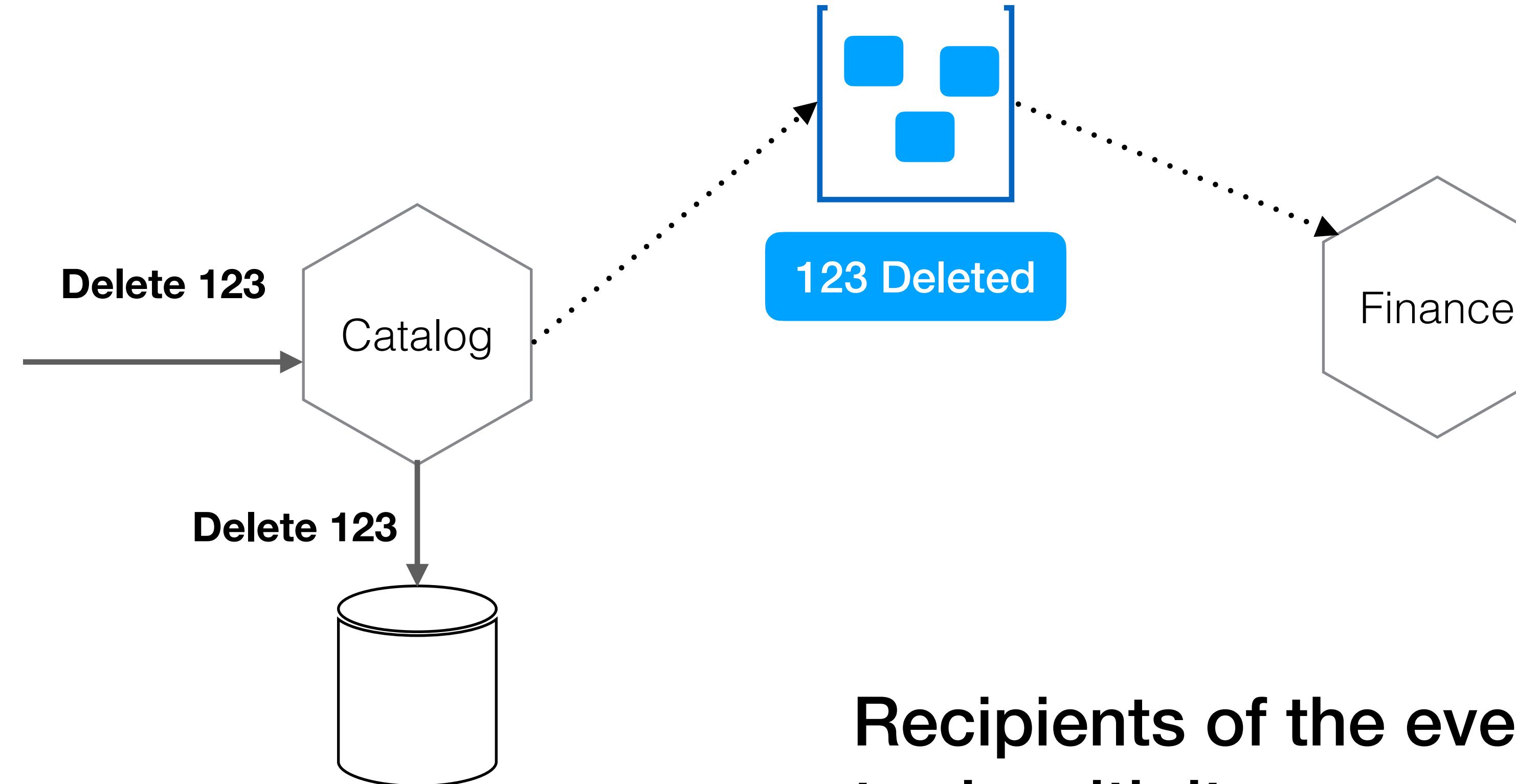


CASCADING DELETES?



**Recipients of the event can decide
to do with it...**

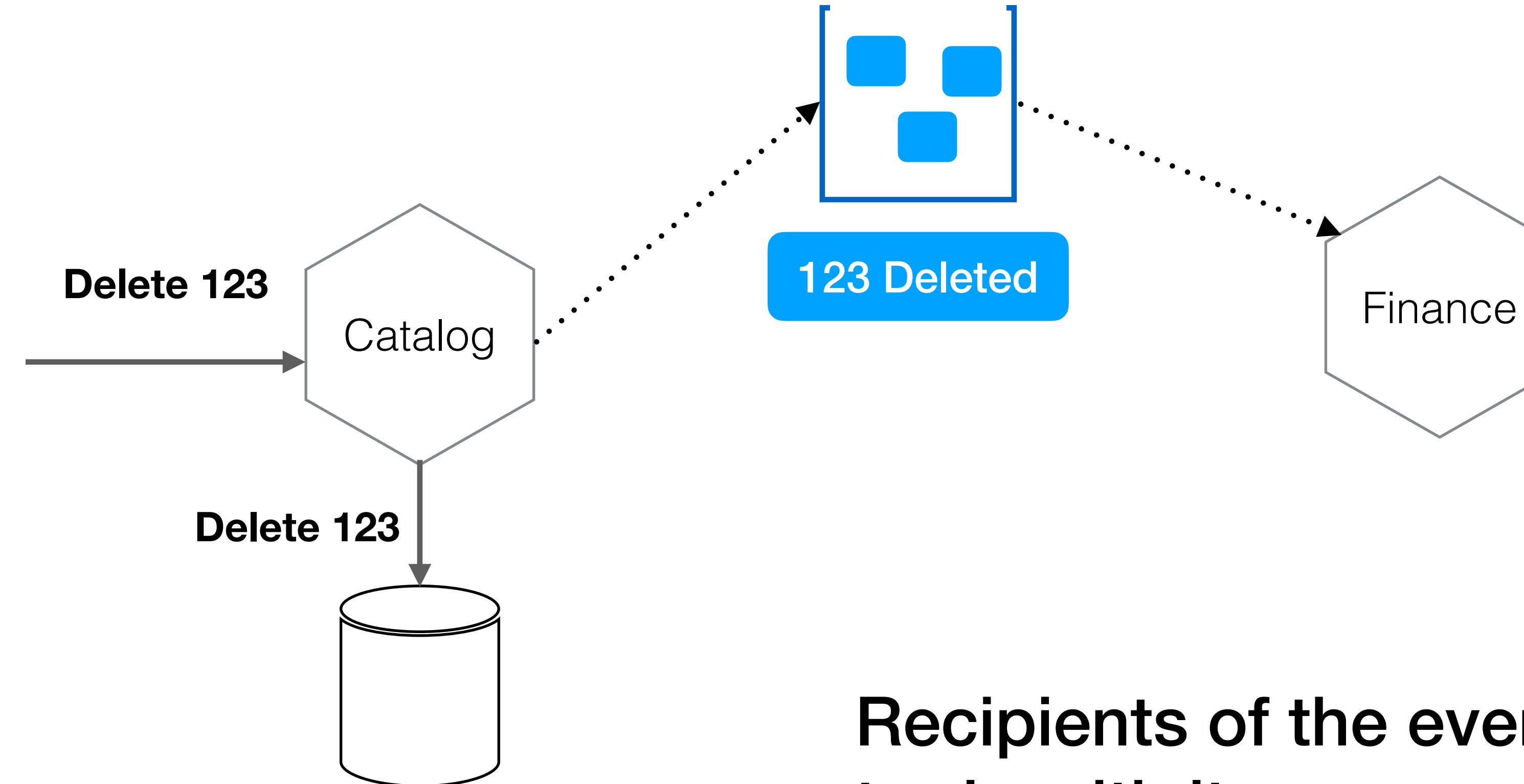
CASCADING DELETES?



**Recipients of the event can decide
to do with it...**

**...you probably don't want to
remove entries from the ledger!**

CASCADING DELETES?



Recipients of the event can decide to do with it...

...you probably don't want to remove entries from the ledger!

There is a delay between the event being fired and it being received

EVENTUAL CONSISTENCY

Eventually Consistent - Revisited

By Werner Vogels on 22 December 2008 04:15 PM | [Permalink](#) | [Comments \(\)](#)

I wrote a [first version of this posting](#) on consistency models about a year ago, but I was never happy with it as it was written in haste and the topic is important enough to receive a more thorough treatment. [ACM Queue](#) asked me to revise it for use in their magazine and I took the opportunity to improve the article. This is that new version.

Eventually Consistent - Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.

At the foundation of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost effectiveness, and they need to meet these requirements while serving millions of customers around the globe, continuously.

Under the covers these services are massive distributed systems that operate on a worldwide scale. This scale creates additional challenges, because when a system processes trillions and trillions of requests, events that normally have a low probability of occurrence are now guaranteed to happen and need to be accounted for up front in the design and architecture of the system. Given the worldwide scope of these systems, we use replication techniques ubiquitously to guarantee consistent performance and high availability. Although replication brings us closer to our goals, it cannot achieve them in a perfectly transparent manner; under a number of conditions the customers of these services will be confronted with the consequences of using replication techniques inside the services.



Contact Info

Werner Vogels
CTO - [Amazon.com](#)

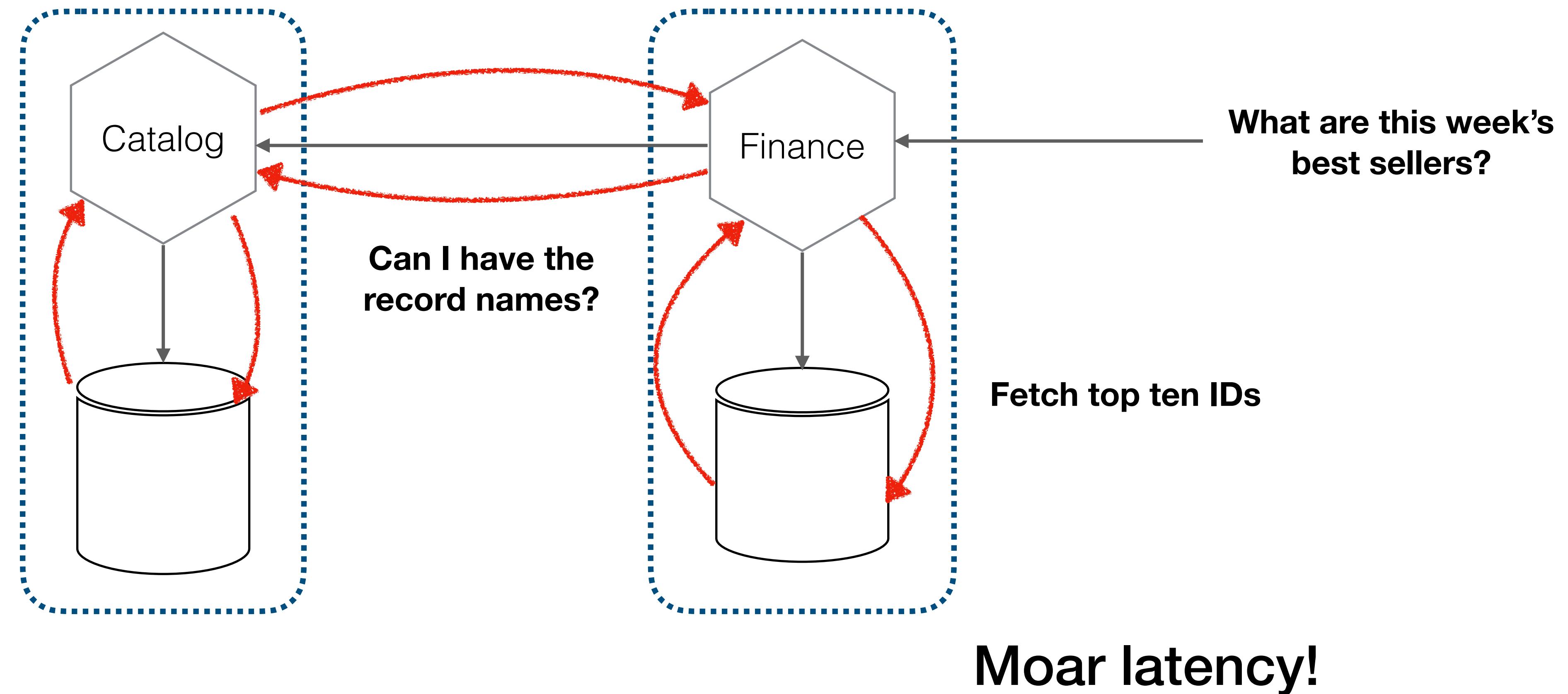
werner@allthingsdistributed.com

Other places

Follow werner on [twitter](#) if you want to know what he is current reading or thinking about.
At werner.ly he posts material that doesn't belong on this blog or on [twitter](#).

https://www.allthingsdistributed.com/2008/12/eventually_consistent.html

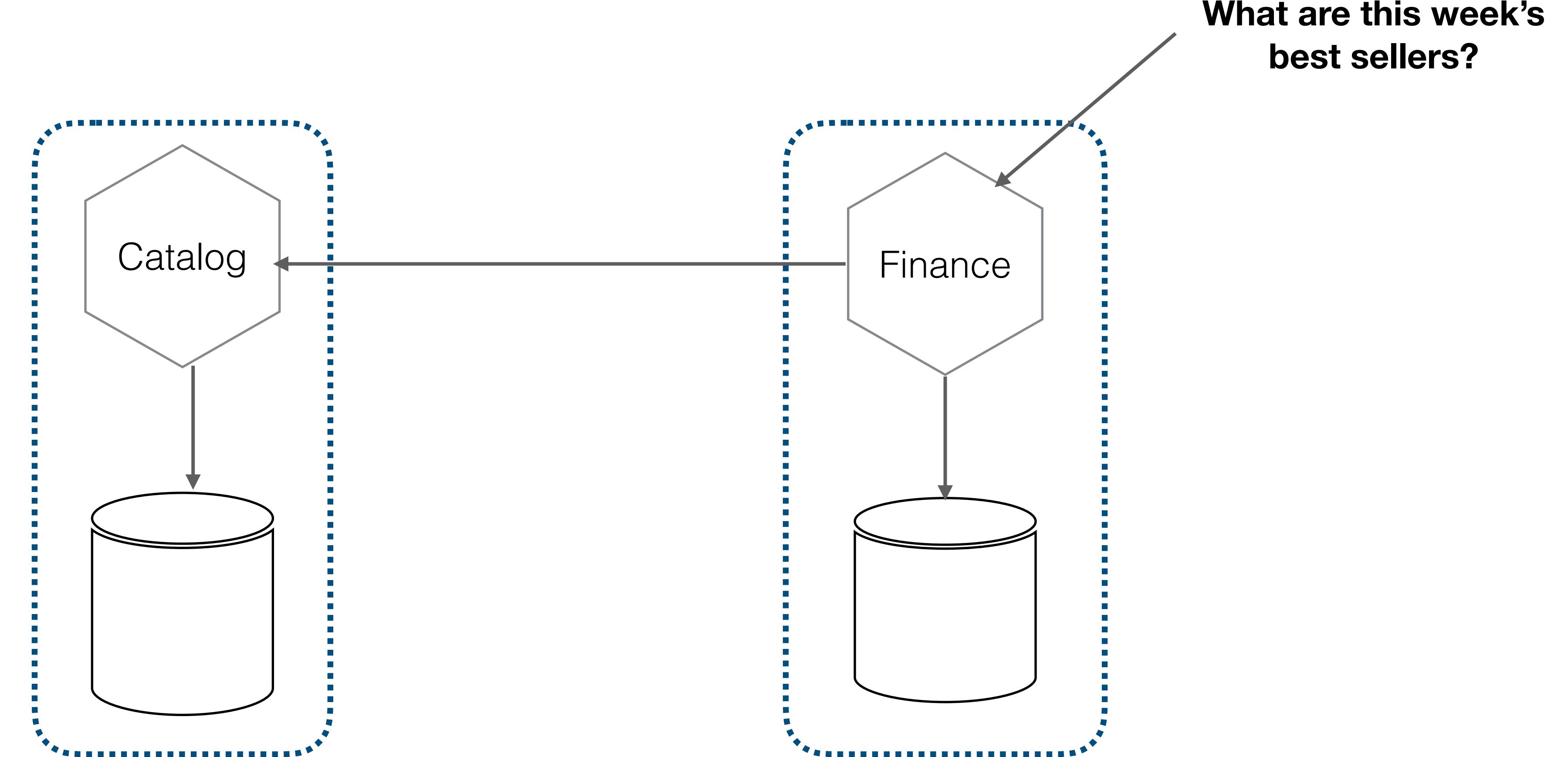
JOINS AT THE SERVICES TIER



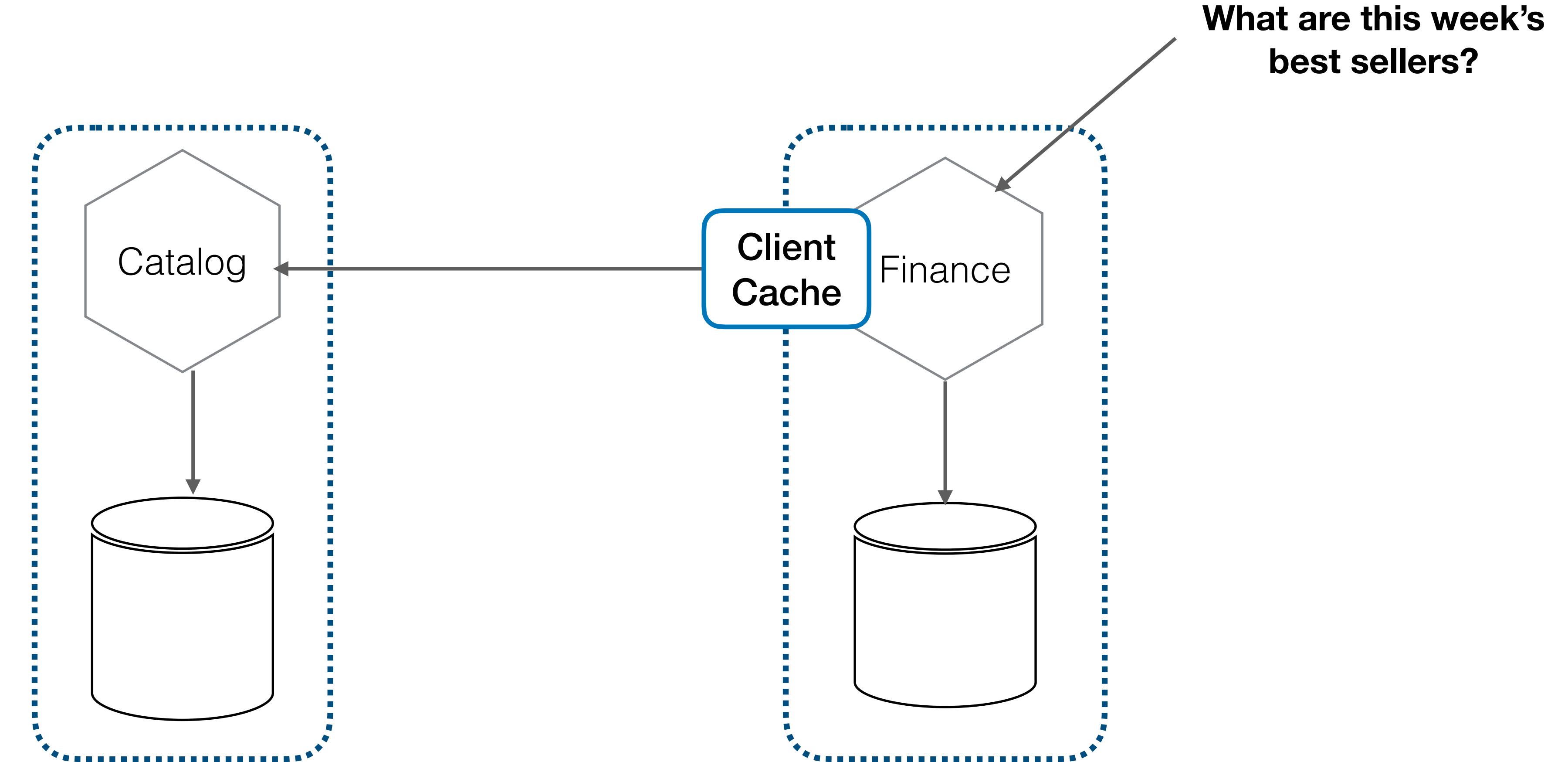
How can we make this join more efficient?

Caching!

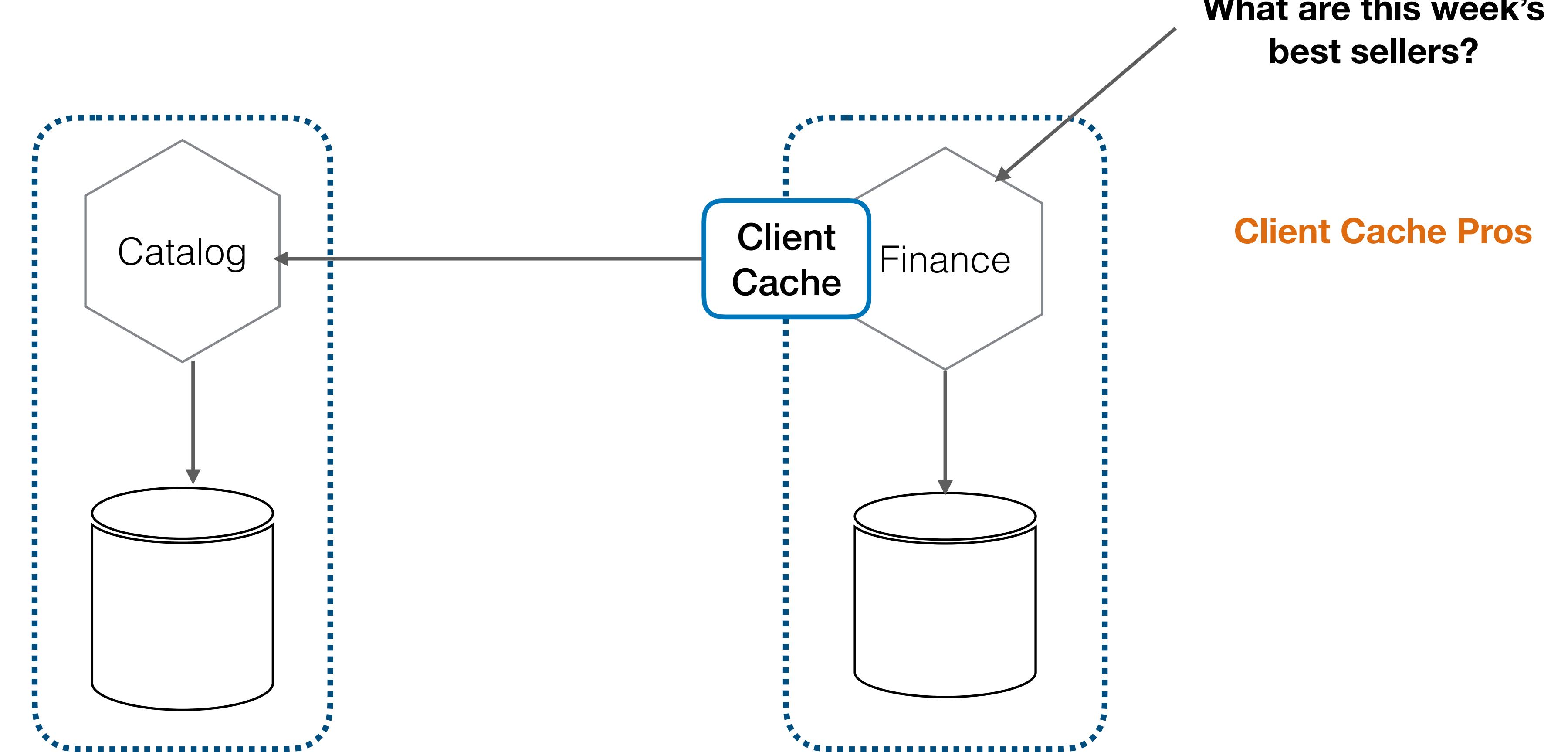
WHERE COULD WE CACHE?



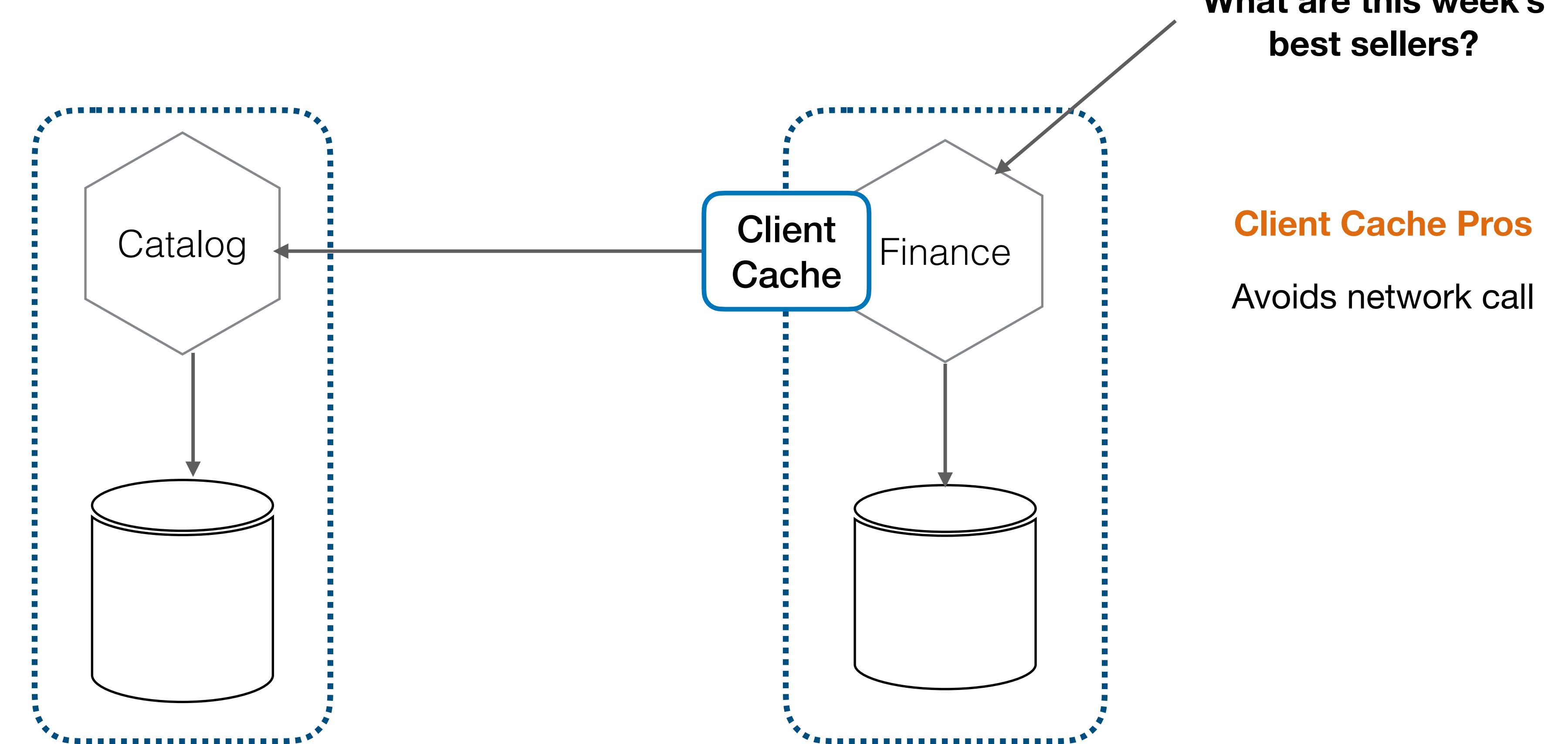
WHERE COULD WE CACHE?



WHERE COULD WE CACHE?



WHERE COULD WE CACHE?

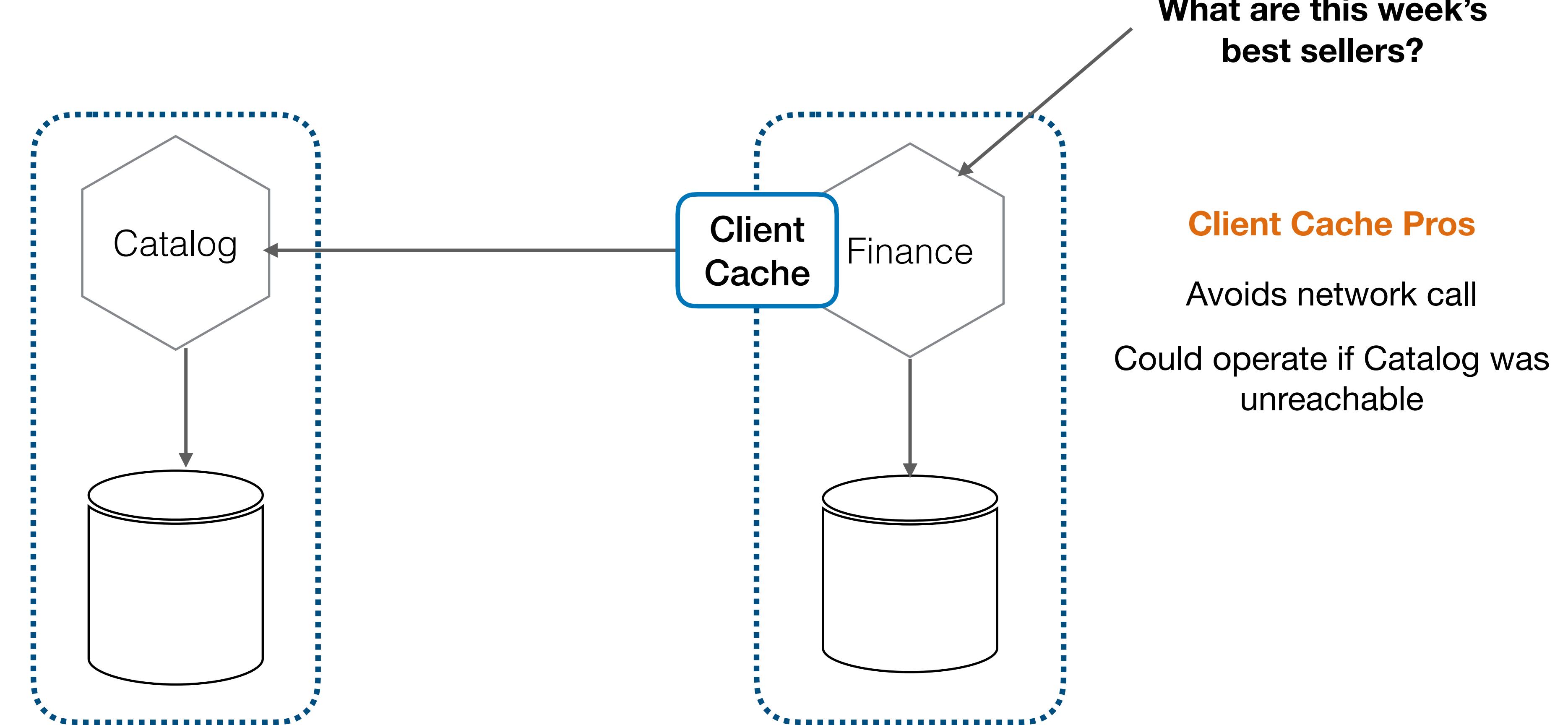


What are this week's
best sellers?

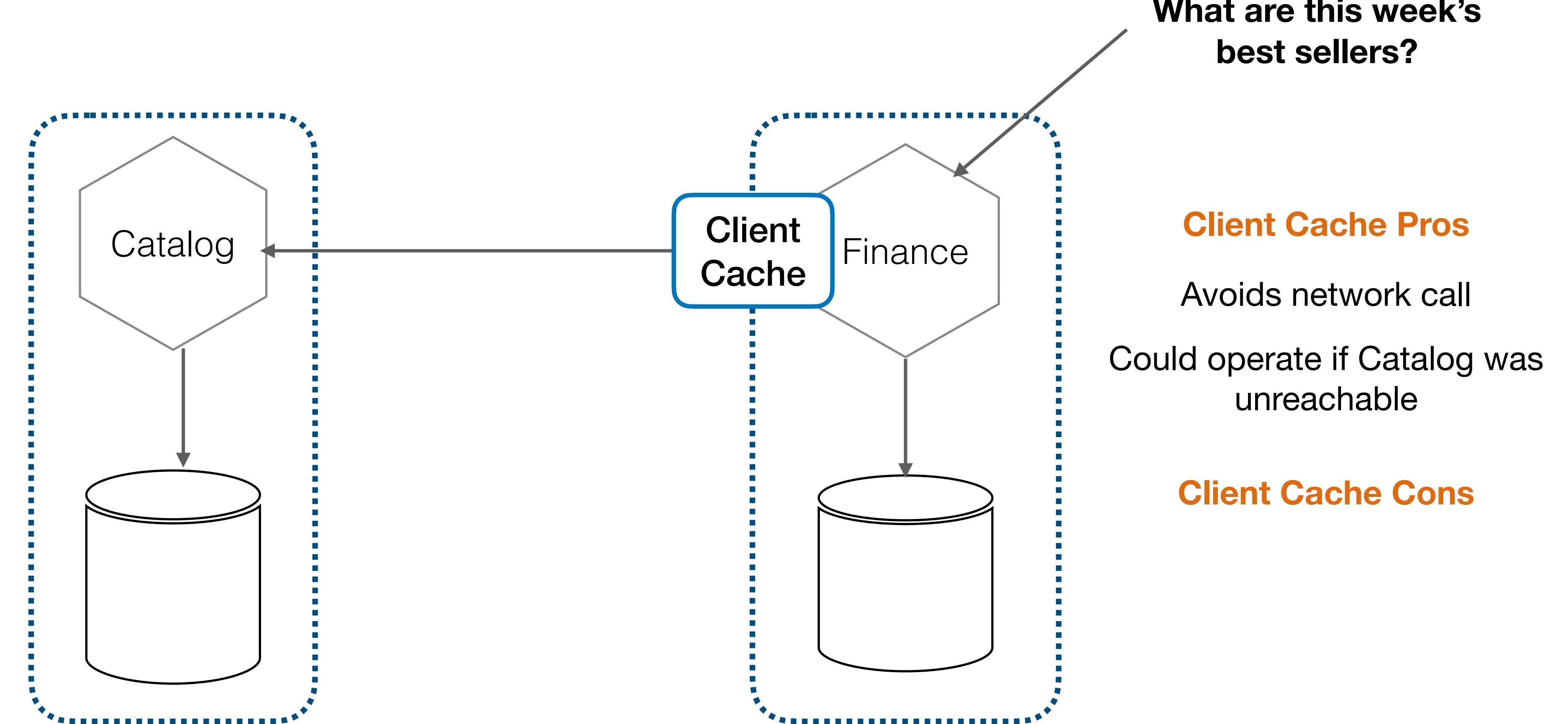
Client Cache Pros

Avoids network call

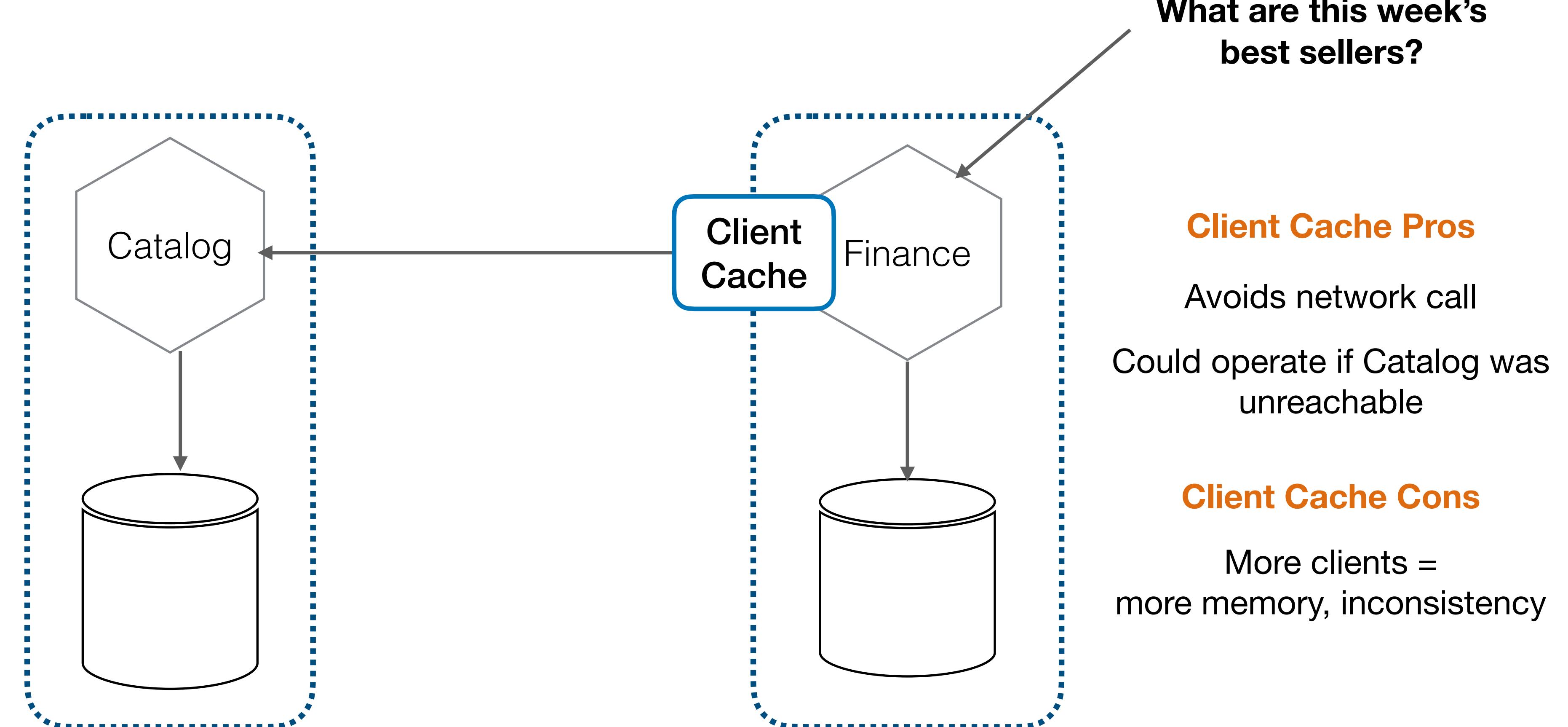
WHERE COULD WE CACHE?



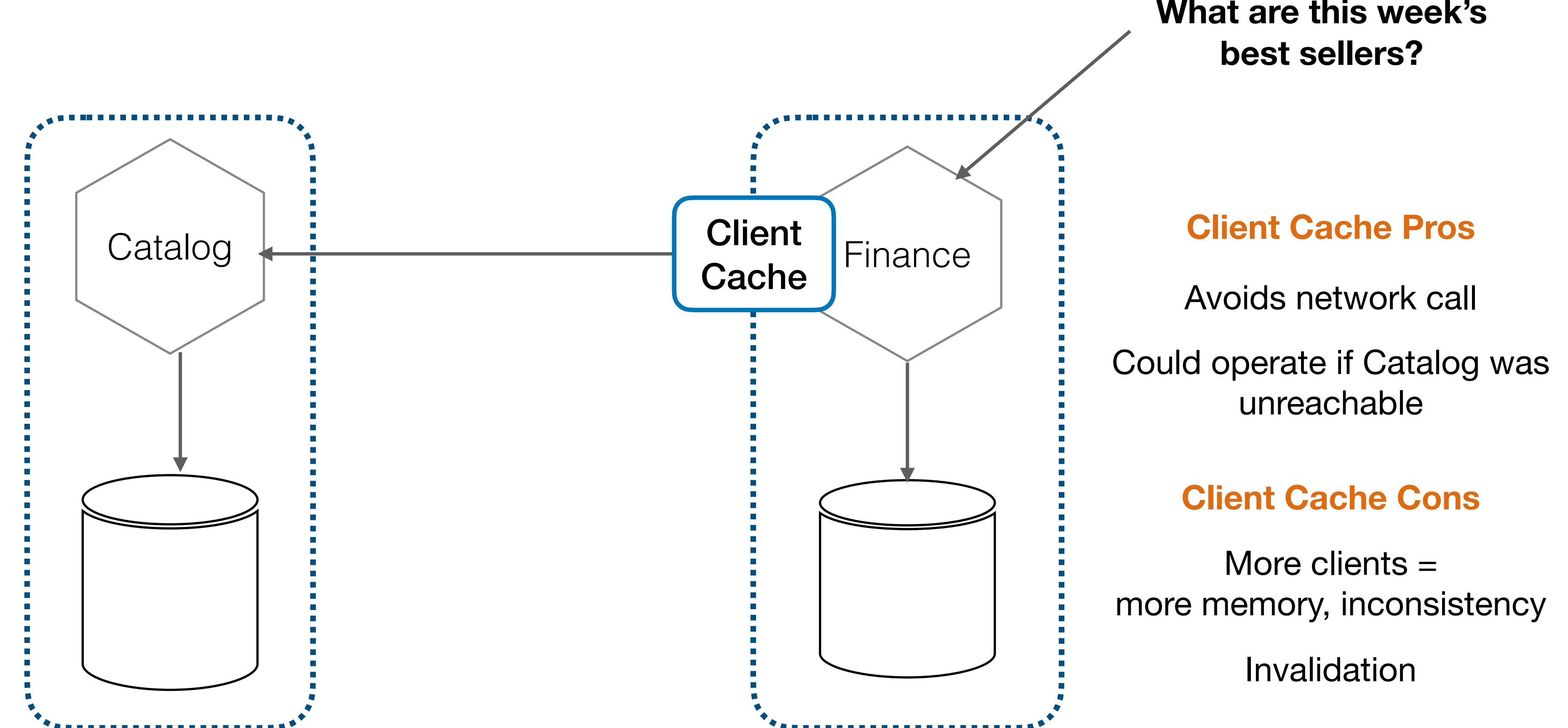
WHERE COULD WE CACHE?



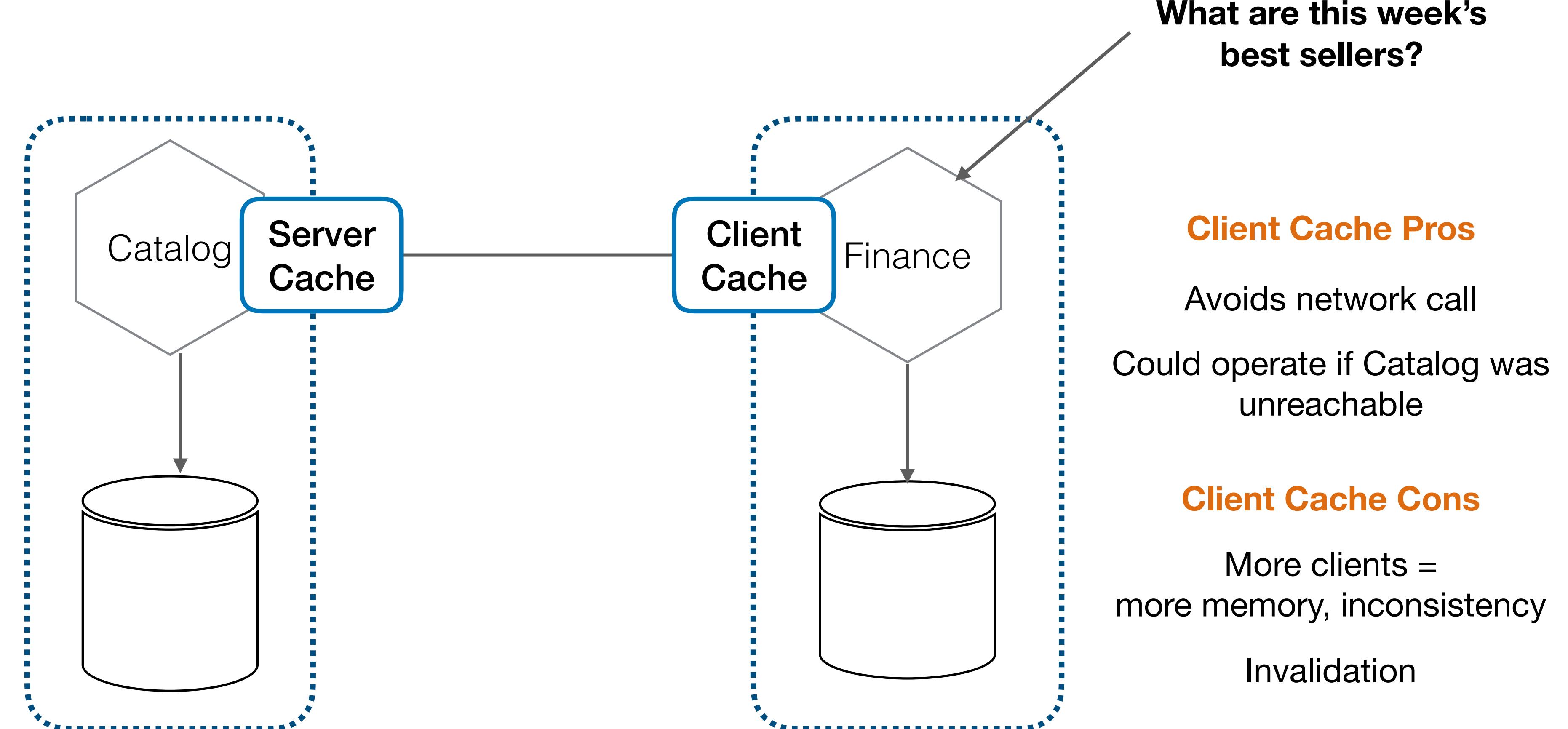
WHERE COULD WE CACHE?



WHERE COULD WE CACHE?

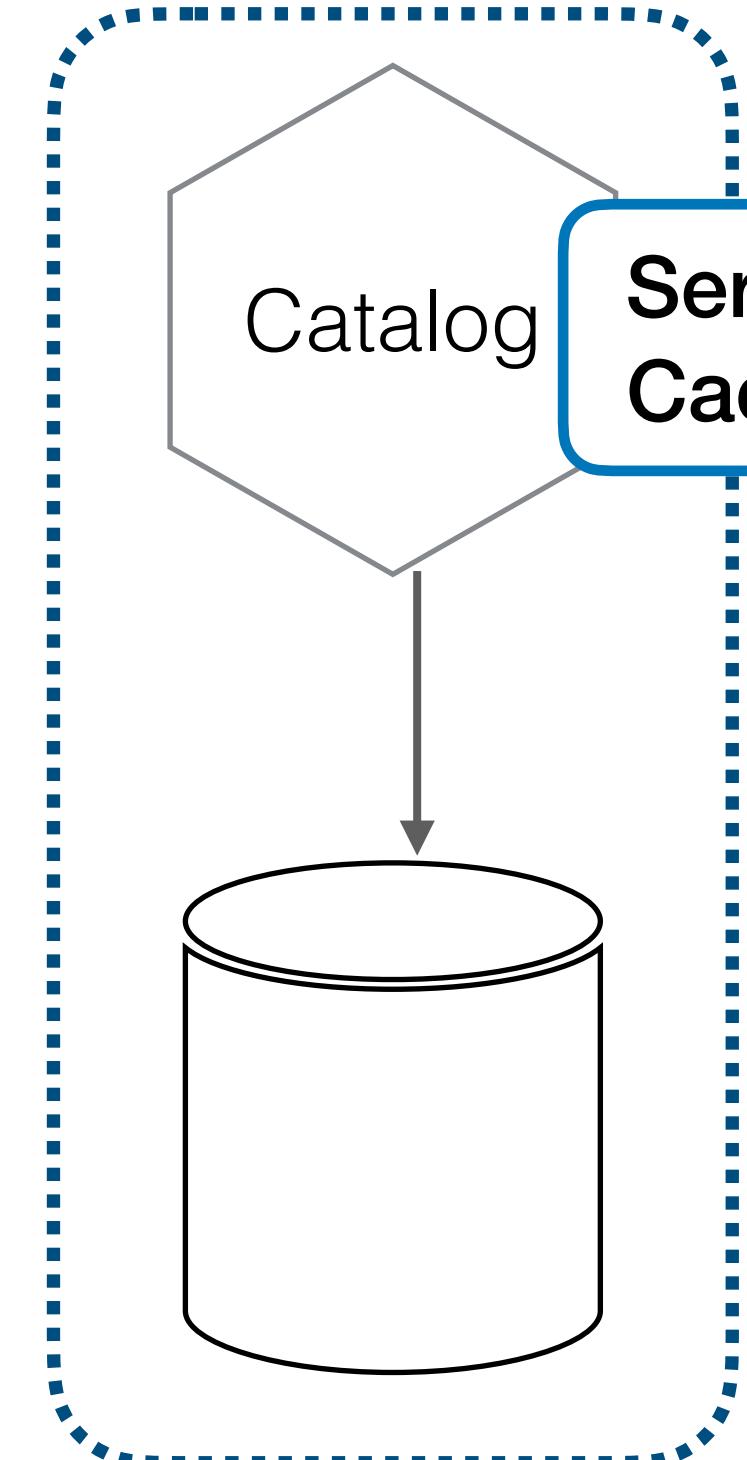


WHERE COULD WE CACHE?



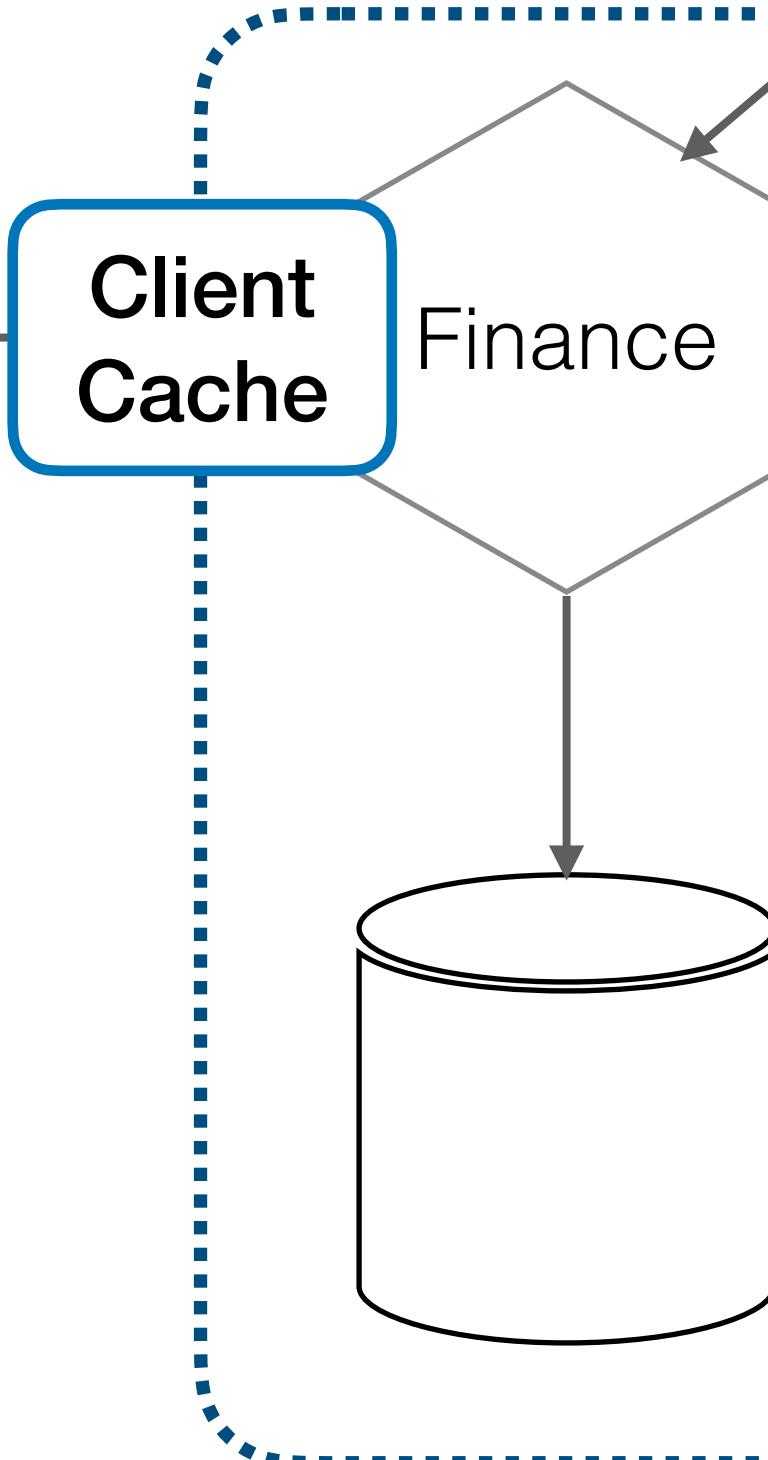
WHERE COULD WE CACHE?

Server Cache Pros



What are this week's best sellers?

Client Cache Pros



Avoids network call
Could operate if Catalog was unreachable

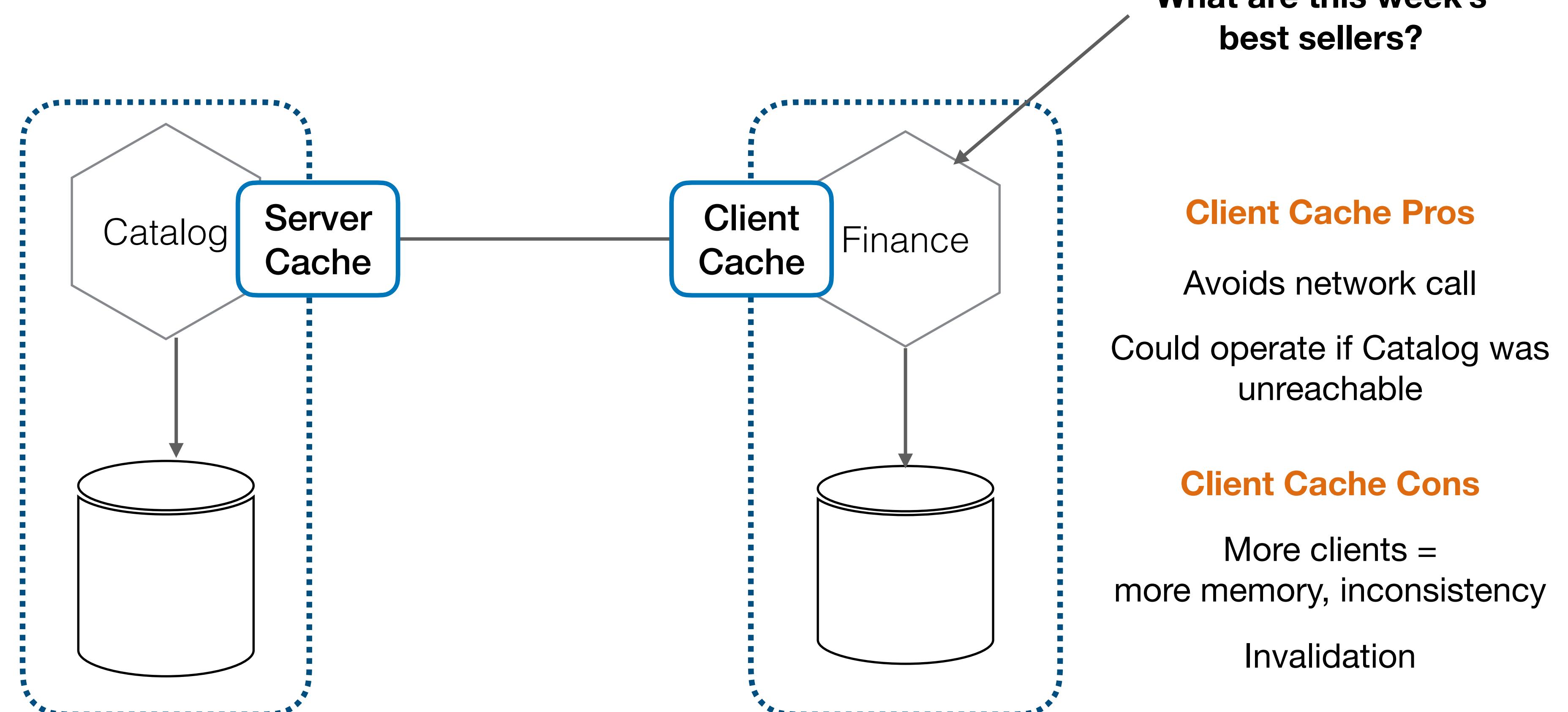
Client Cache Cons

More clients = more memory, inconsistency
Invalidation

WHERE COULD WE CACHE?

Server Cache Pros

Invalidation easier



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

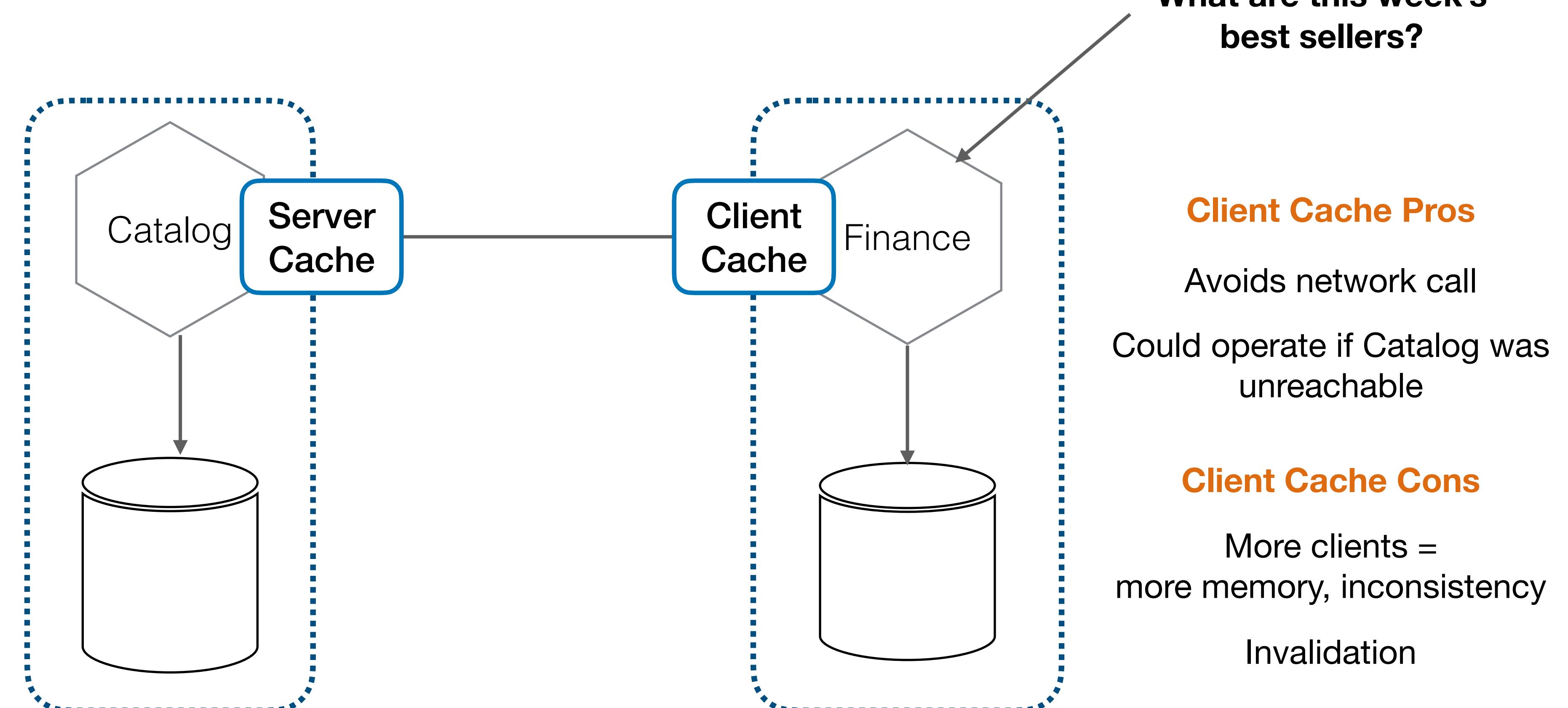
Invalidation

WHERE COULD WE CACHE?

Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

Invalidation

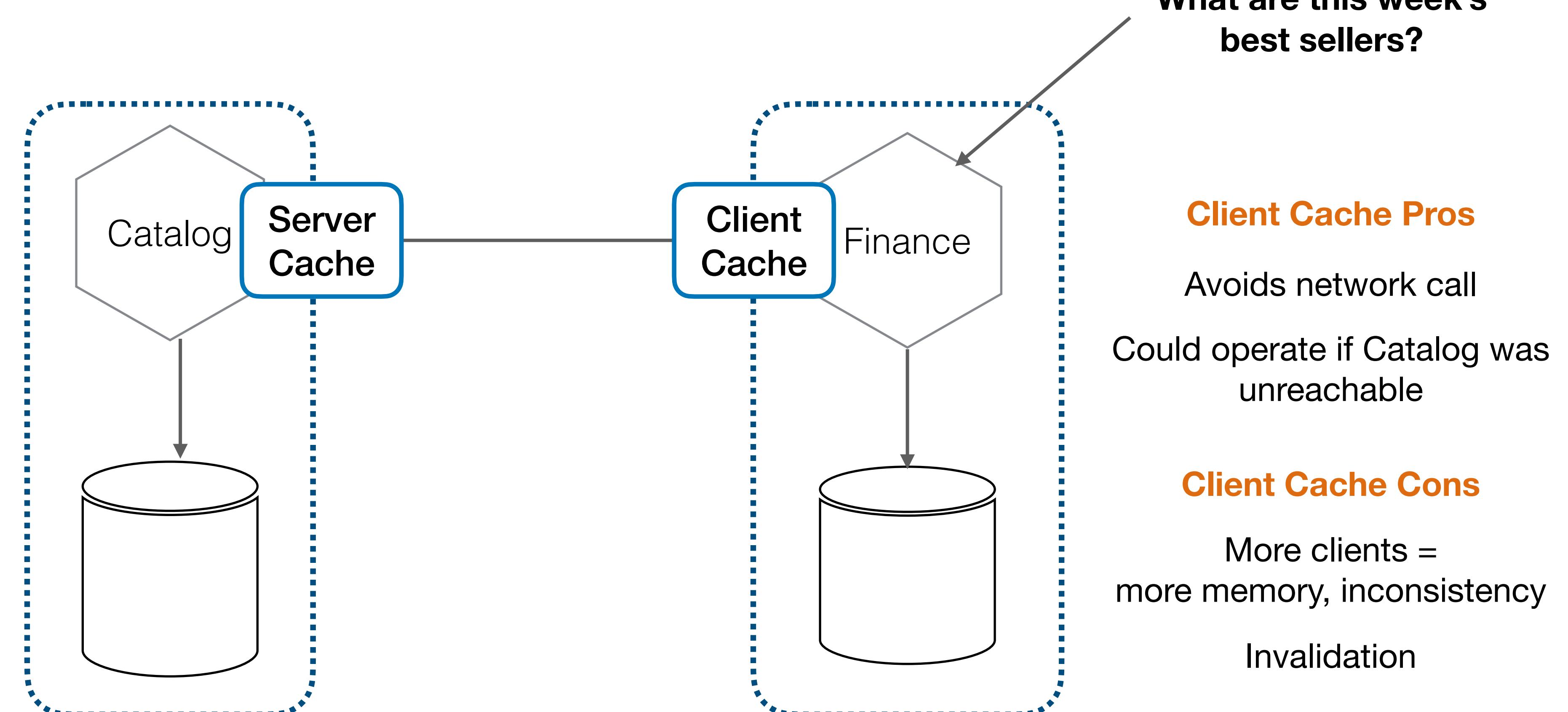
WHERE COULD WE CACHE?

Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

Server Cache Cons



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

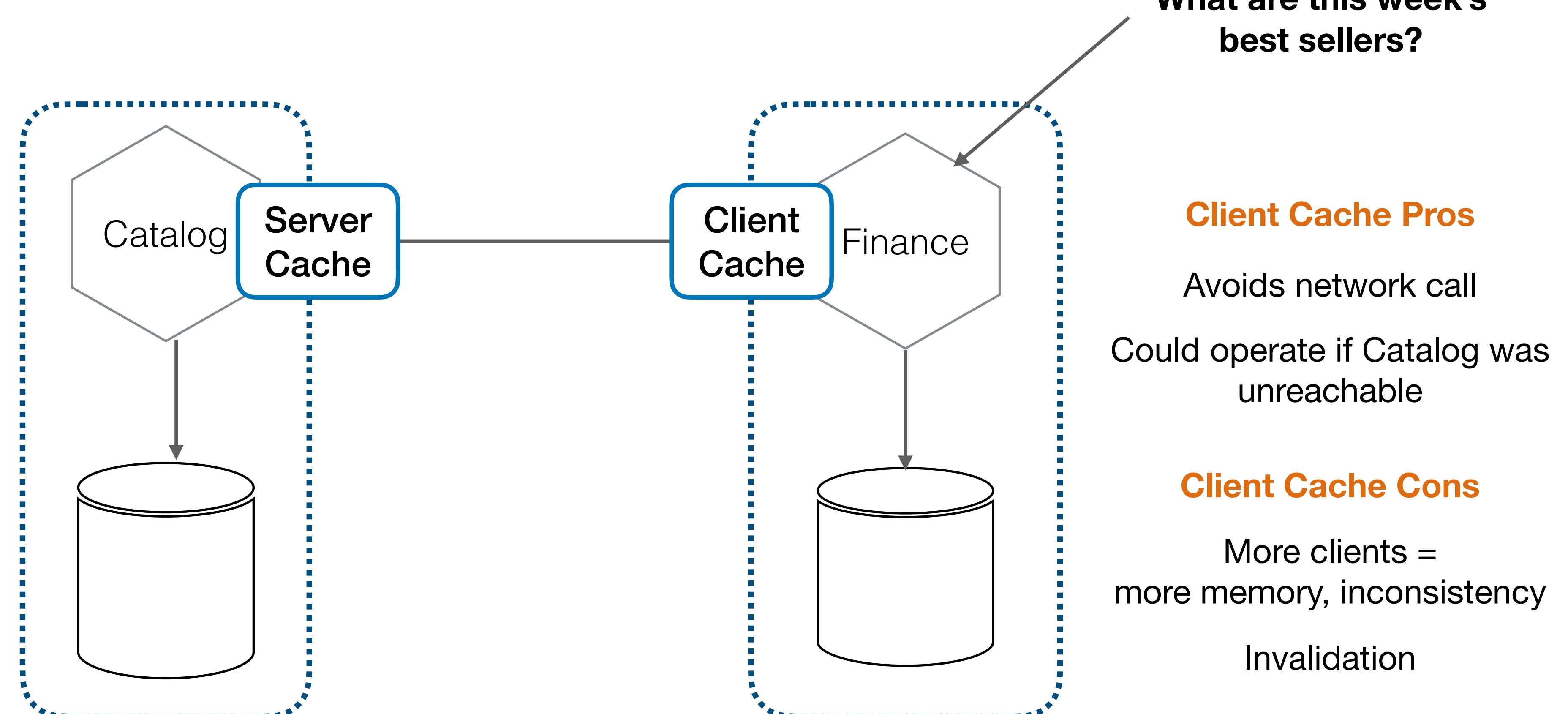
Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

Server Cache Cons

Call still needs to be made



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

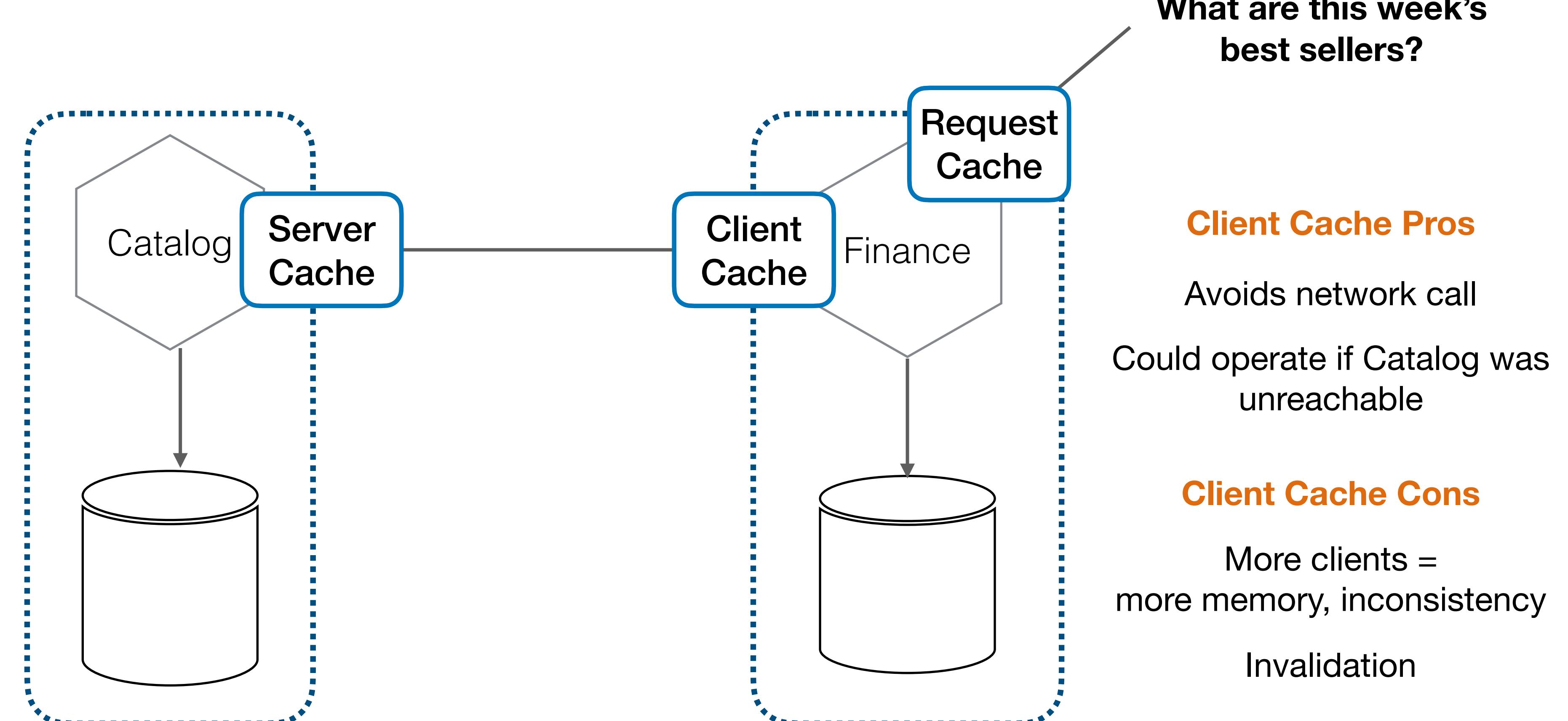
Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

Server Cache Cons

Call still needs to be made



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

Request Cache Pros

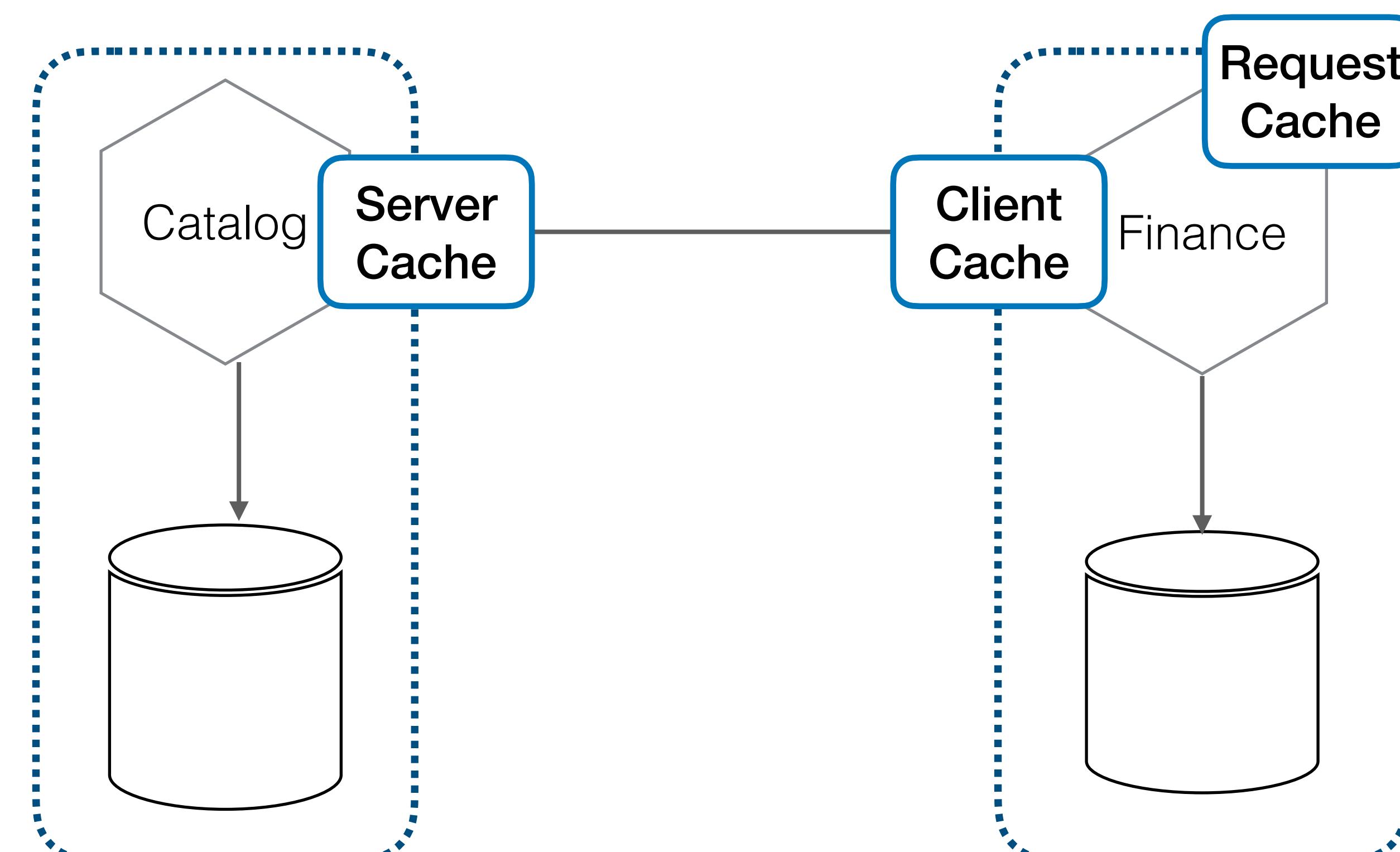
Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

Server Cache Cons

Call still needs to be made



What are this week's best sellers?

Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

Request Cache Pros

Super efficient

What are this week's best sellers?

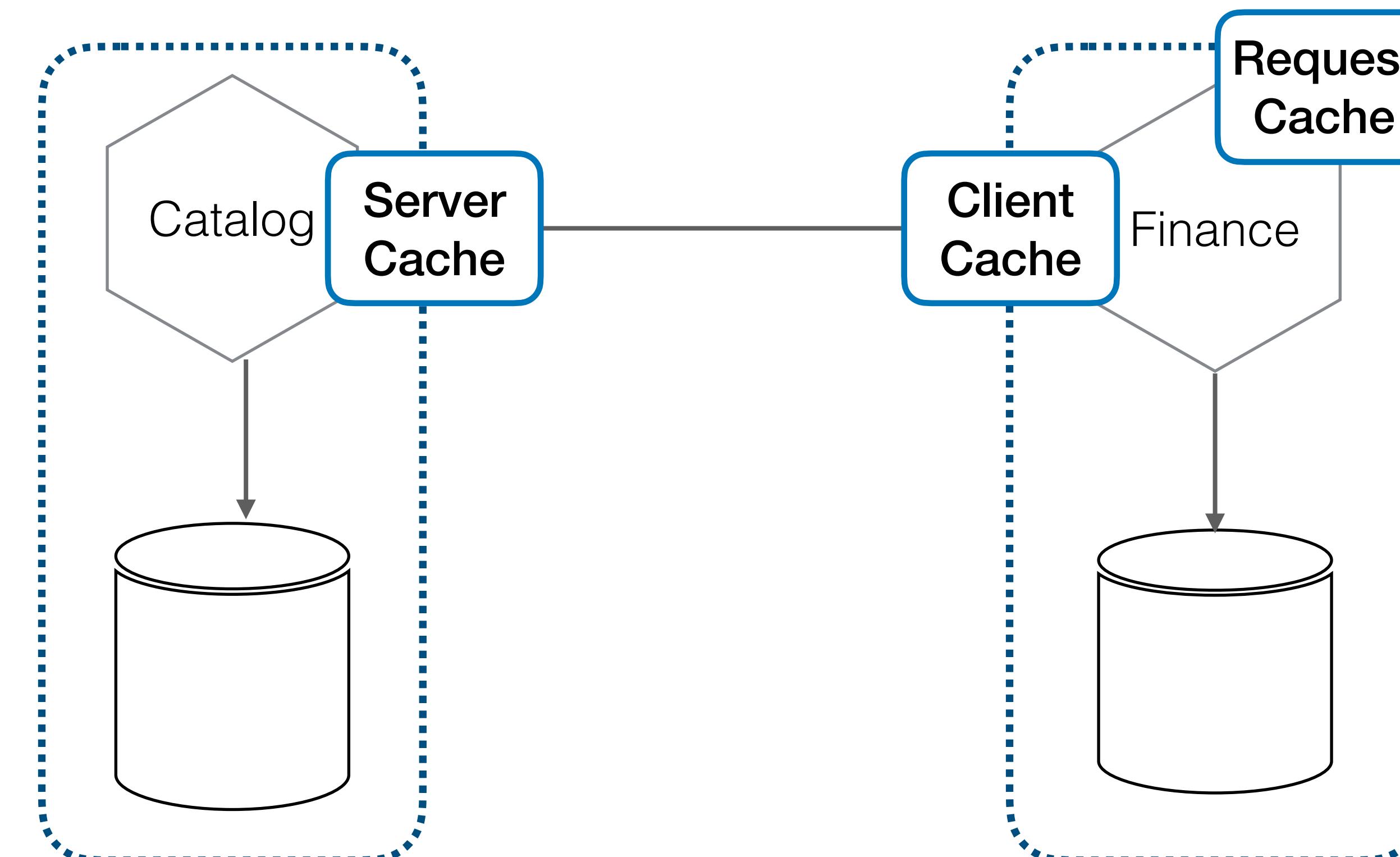
Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

Server Cache Cons

Call still needs to be made



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

Server Cache Cons

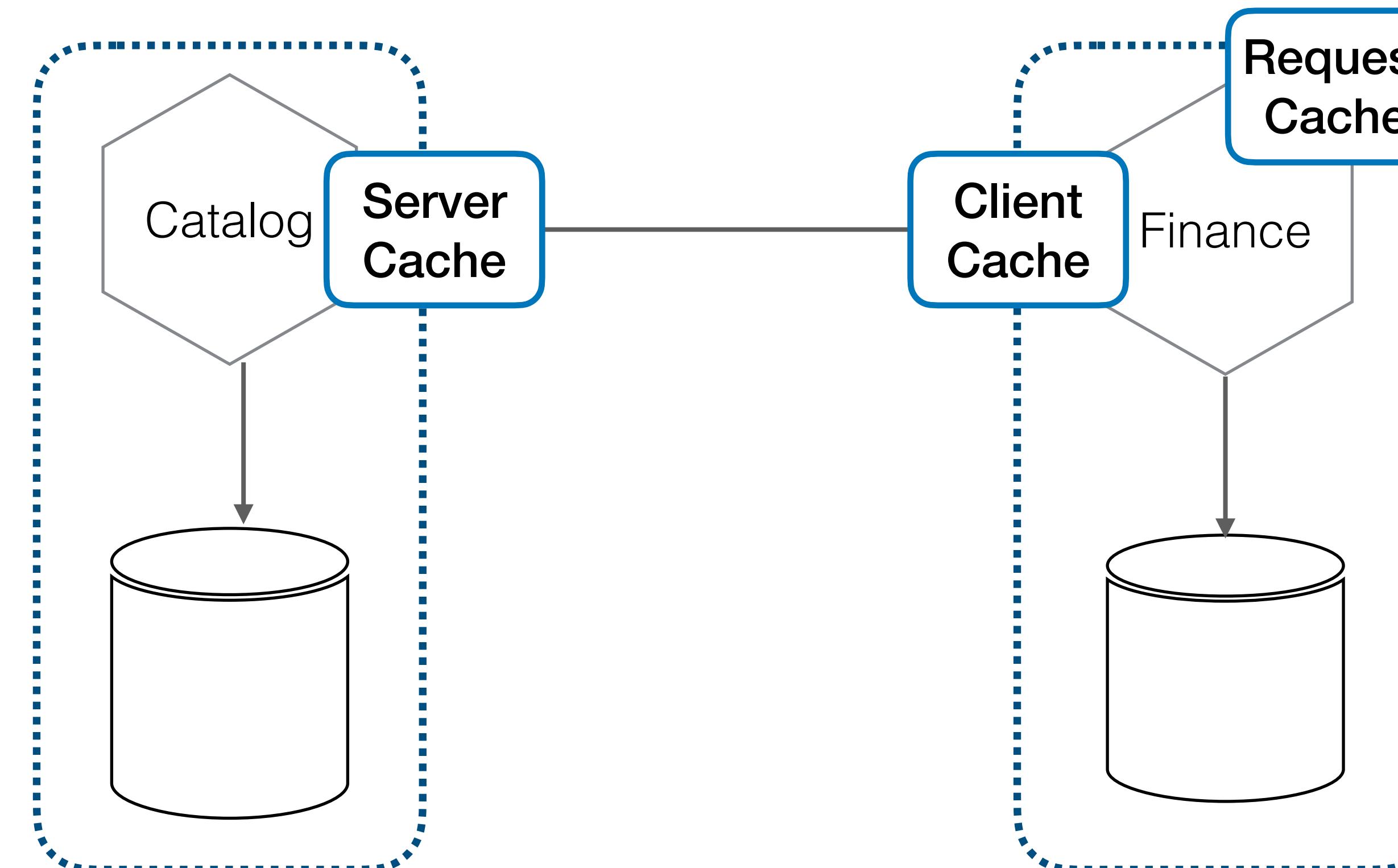
Call still needs to be made

Request Cache Pros

Super efficient

Client Cache Cons

What are this week's best sellers?



Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

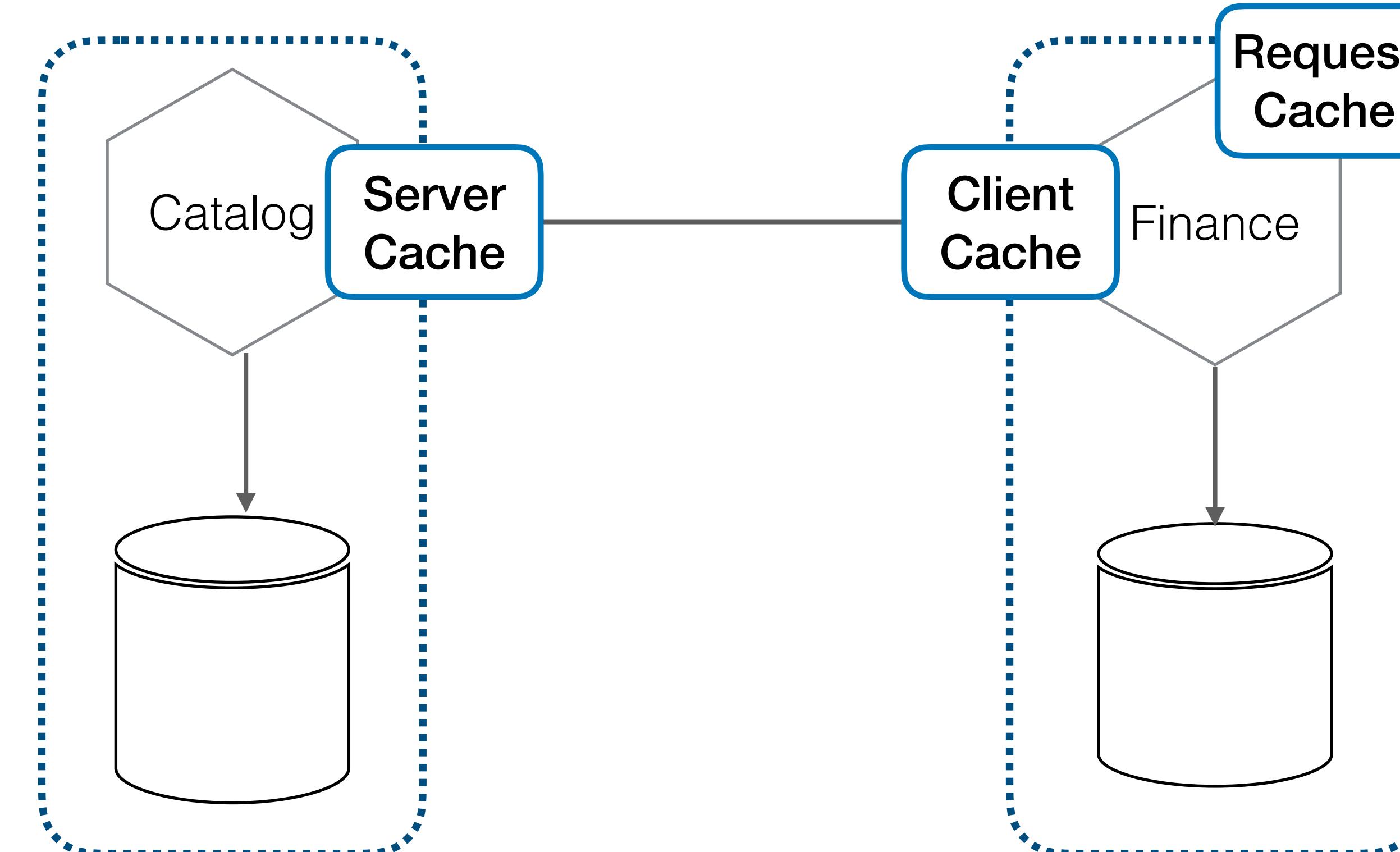
More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

Server Cache Pros
Invalidation easier
Can be more efficient in terms of memory

Server Cache Cons
Call still needs to be made



Request Cache Pros

Super efficient

Client Cache Cons

Highly-specific cache

What are this week's best sellers?

Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

Client Cache Cons

More clients = more memory, inconsistency

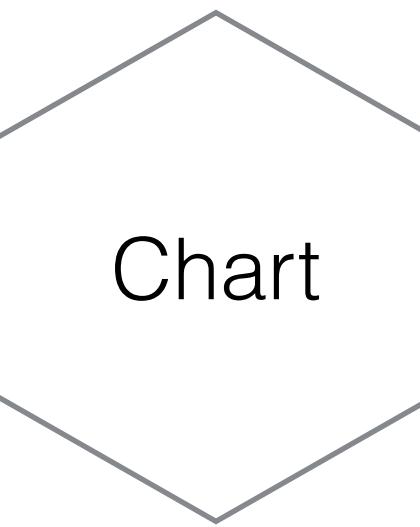
Invalidation

A DEDICATED SERVICE?

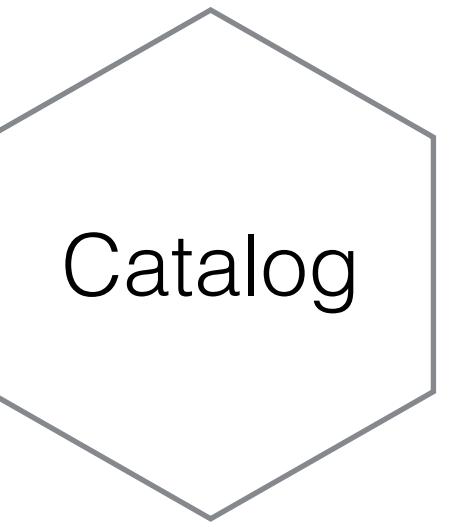
Catalog

Finance

A DEDICATED SERVICE?



Chart

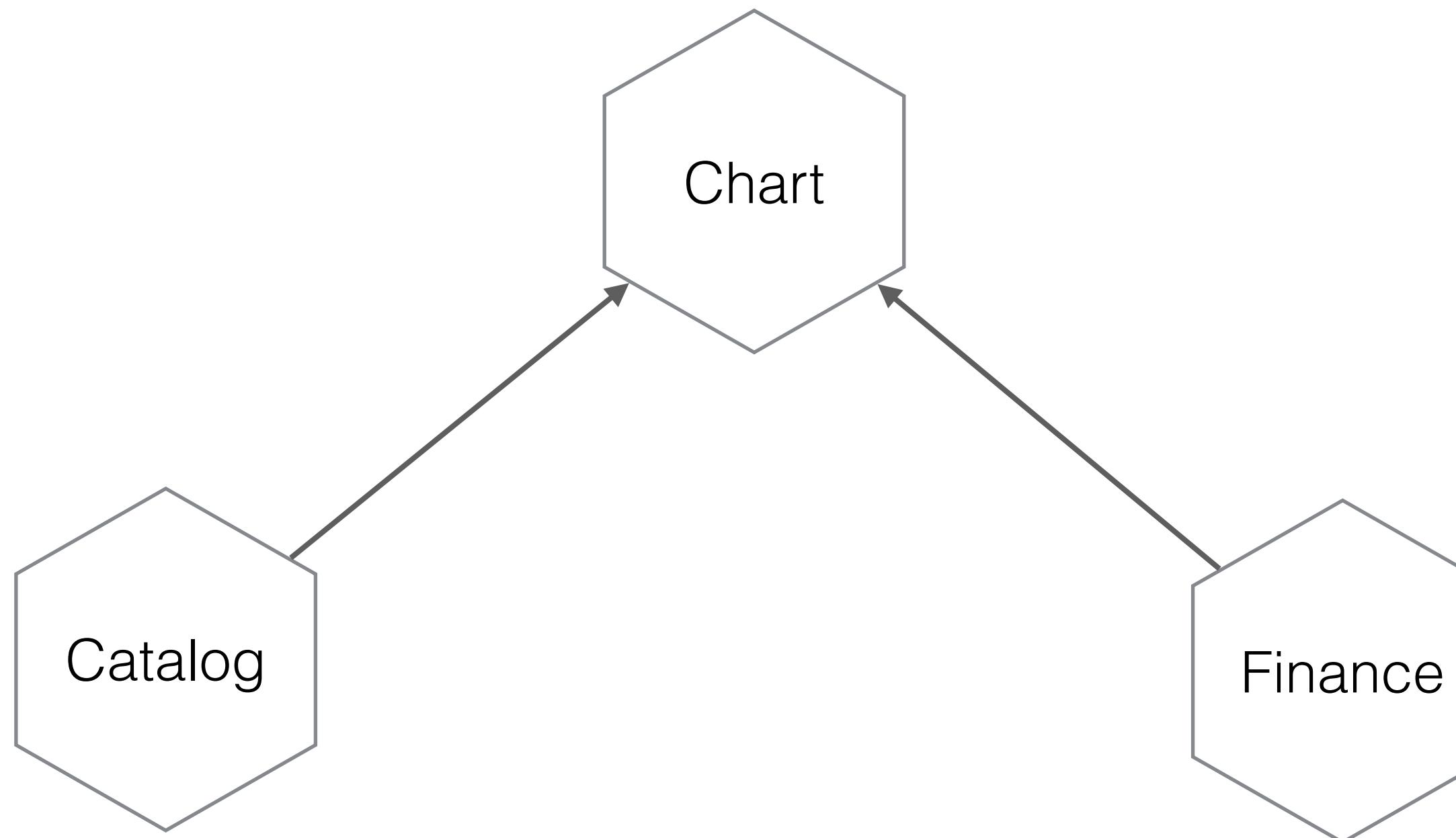


Catalog



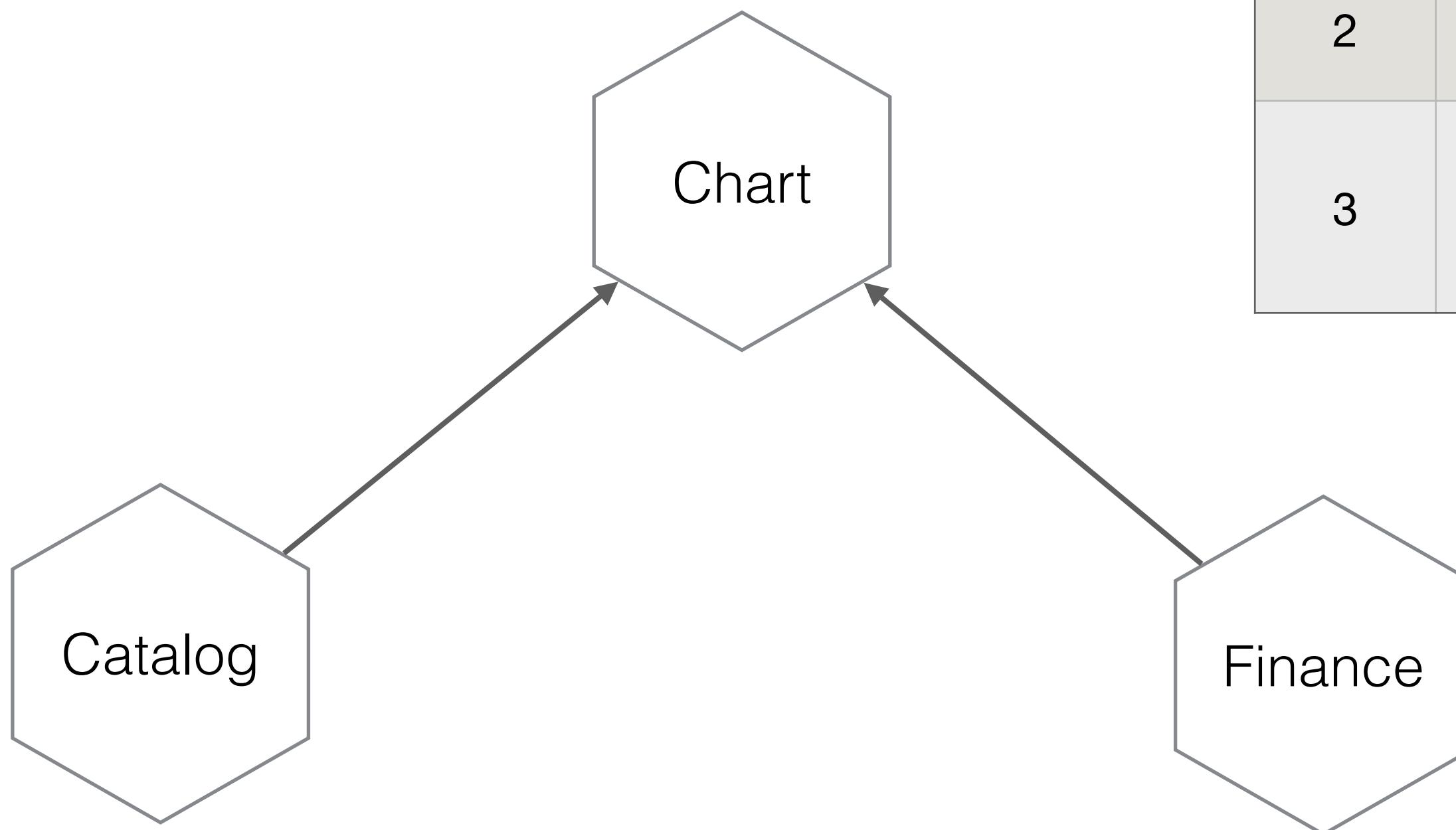
Finance

A DEDICATED SERVICE?



A DEDICATED SERVICE?

Best Sellers This Week!

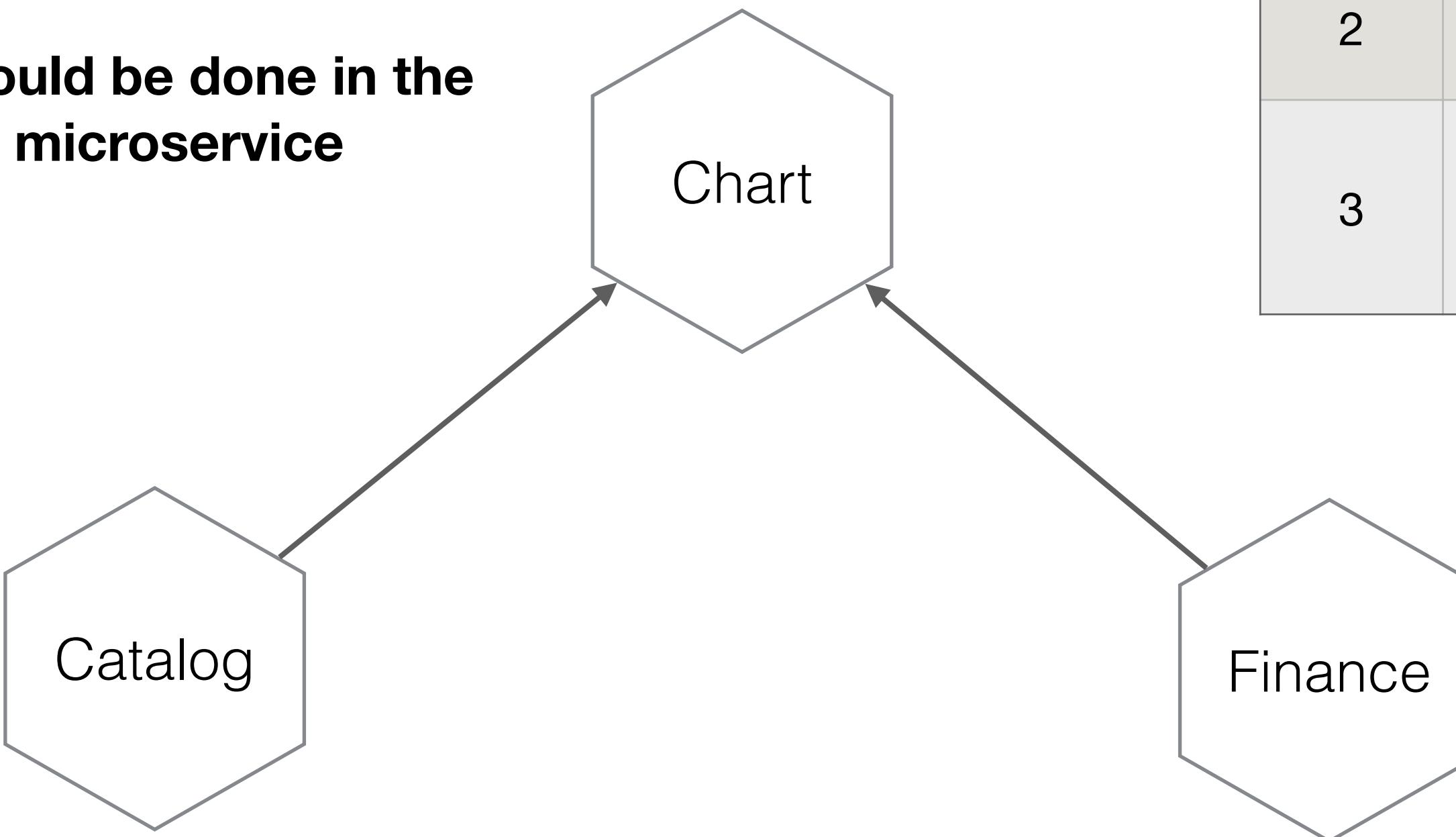


1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

A DEDICATED SERVICE?

Best Sellers This Week!

**Caching could be done in the
Chart microservice**

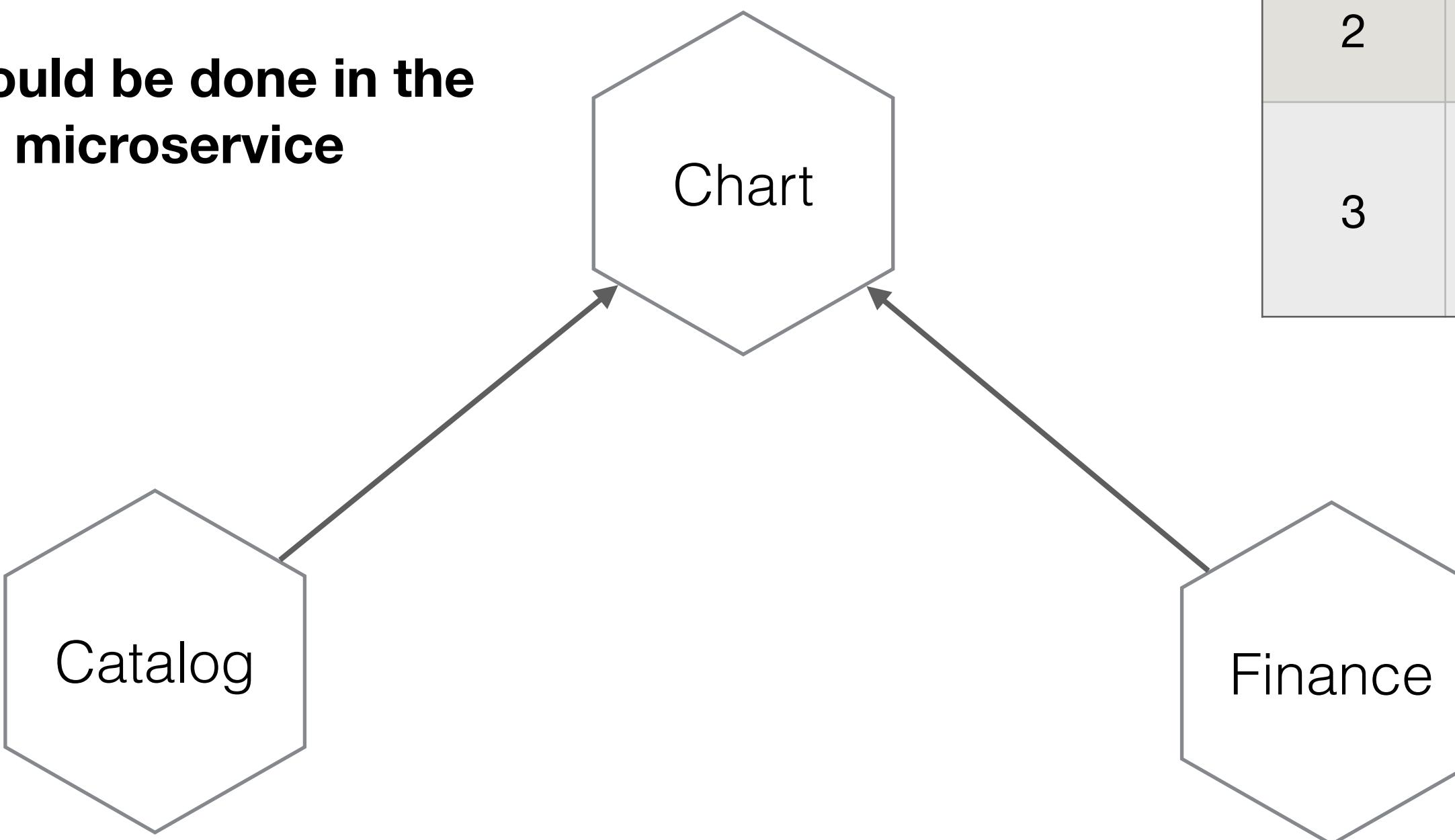


1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

A DEDICATED SERVICE?

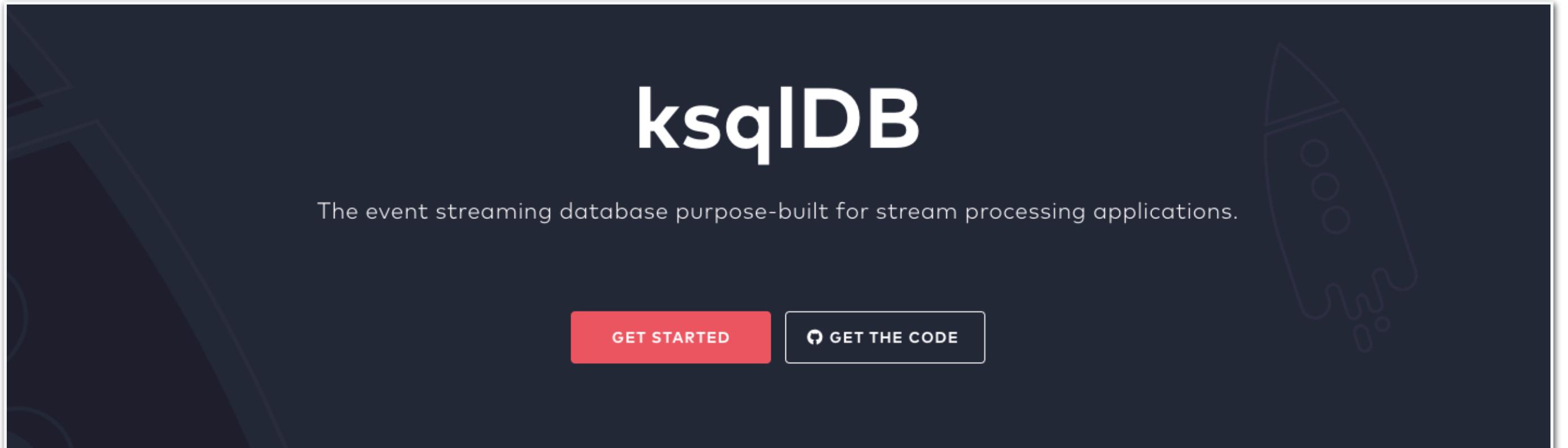
Best Sellers This Week!

**Caching could be done in the
Chart microservice**



1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

**Could also do a “live” query to the
downstream microservices**



The landing page for KSQLDB features a dark background with a faint rocket ship illustration. The title "ksqlDB" is prominently displayed in white. Below it is a subtitle: "The event streaming database purpose-built for stream processing applications." Two buttons are visible: a red "GET STARTED" button and a white "GET THE CODE" button with a GitHub icon.

Real, real-time

Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.

Kafka-native

Seamlessly leverage your existing [Apache Kafka®](#) infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.

What, not how

Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

Thousands of organizations love & trust ksqlDB

<https://ksqldb.io/>

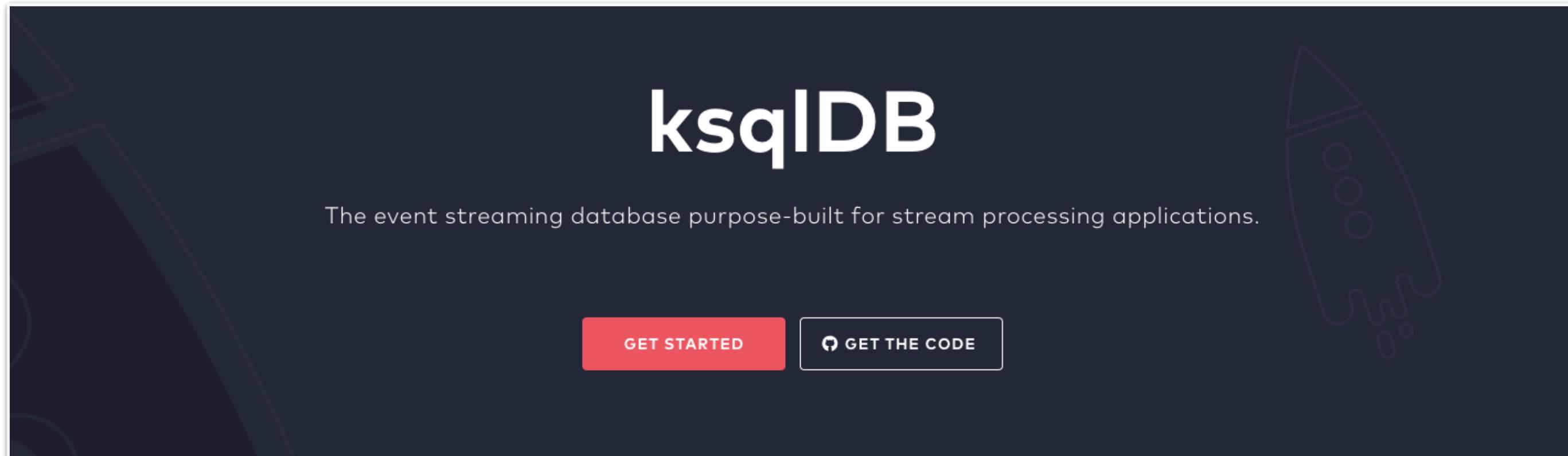
The screenshot shows the ksqlDB homepage with a dark background. At the top center is the word "ksqlDB" in large white letters. Below it is a subtitle: "The event streaming database purpose-built for stream processing applications." To the right of the subtitle is a faint illustration of a rocket ship. At the bottom of the main section are two buttons: "GET STARTED" (red) and "GET THE CODE" (white). The page then transitions into a light gray section containing three columns of text and logos.

Real, real-time	Kafka-native	What, not how
Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.	Seamlessly leverage your existing Apache Kafka® infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.	Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

Thousands of organizations love & trust ksqlDB

<https://ksqldb.io/>

Apply SQL-style queries
to streams being
emitted from Kafka



The screenshot shows the ksqlDB homepage. At the top, the word "ksqlDB" is written in a large, white, sans-serif font. Below it, a subtitle reads "The event streaming database purpose-built for stream processing applications." Two buttons are visible: a red "GET STARTED" button and a white "GET THE CODE" button. The main content area is divided into three sections: "Real, real-time", "Kafka-native", and "What, not how". Each section contains a brief description and a small icon. At the bottom, a light gray bar features the text "Thousands of organizations love & trust ksqlDB" and logos for Mailchimp, Bosch, Derivco, and pushowl.

Real, real-time

Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.

Kafka-native

Seamlessly leverage your existing [Apache Kafka®](#) infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.

What, not how

Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

Thousands of organizations love & trust ksqlDB

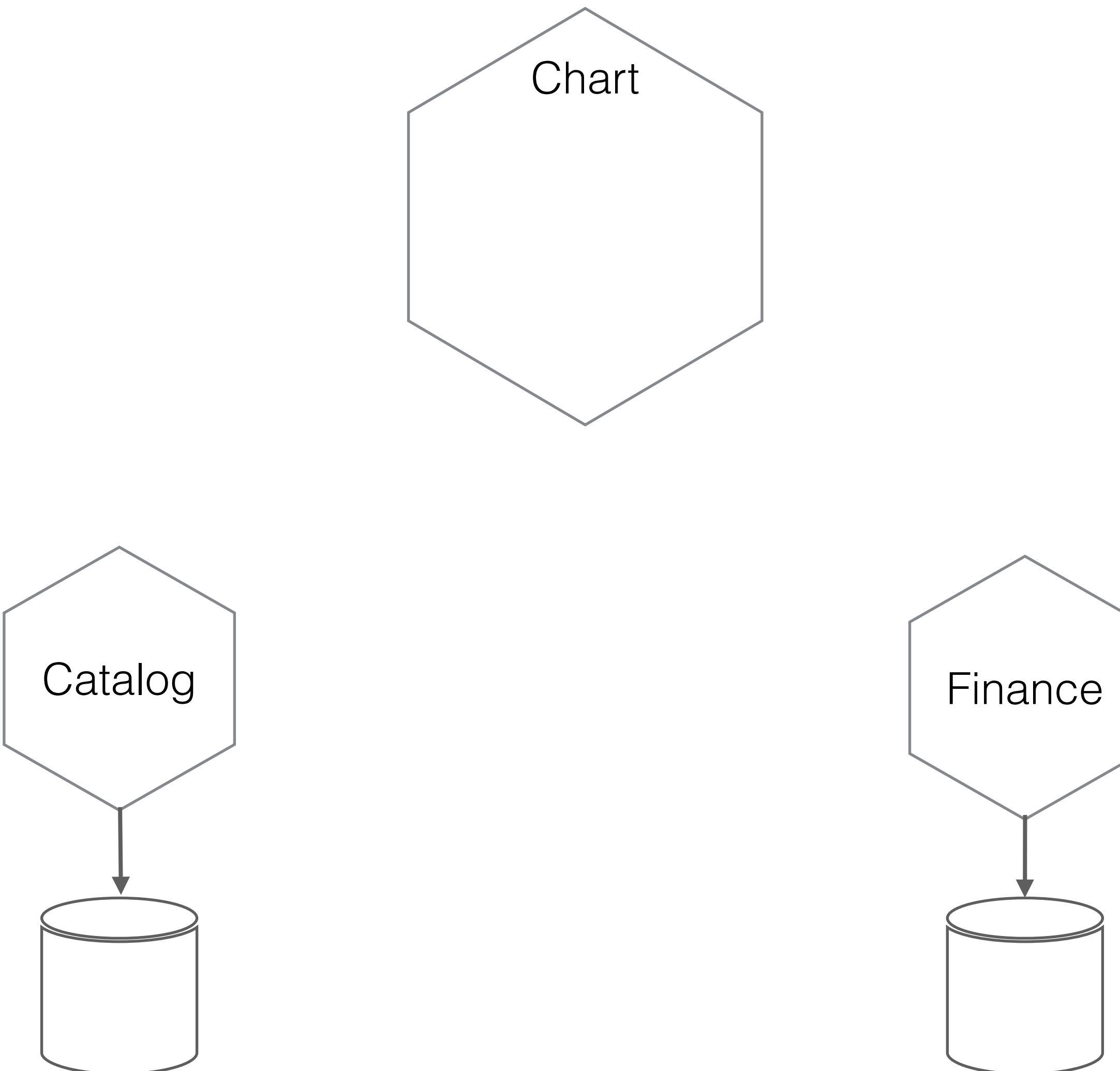
   

<https://ksqldb.io/>

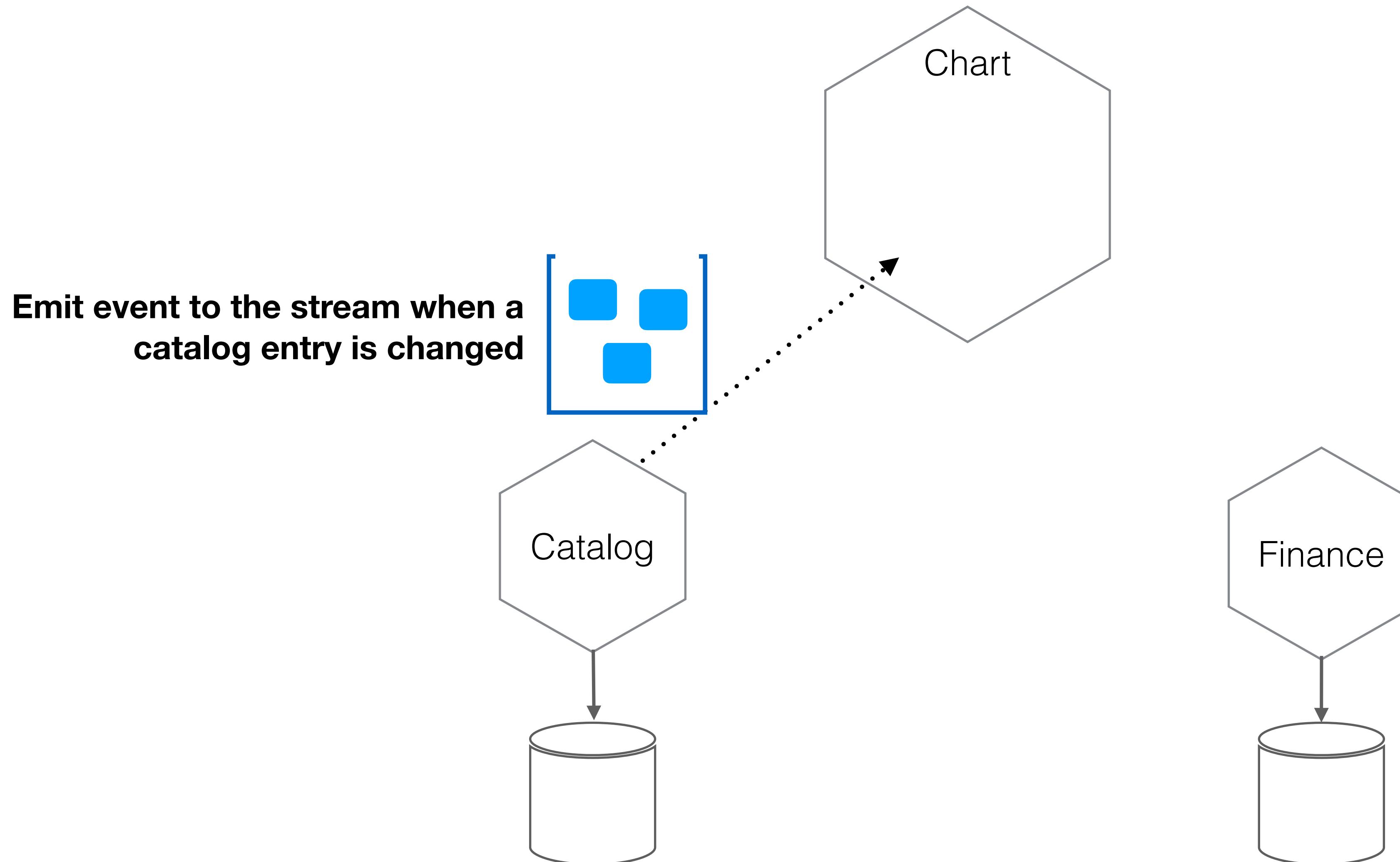
**Apply SQL-style queries
to streams being
emitted from Kafka**

**Blends stream
processing with
standard SQL-use cases**

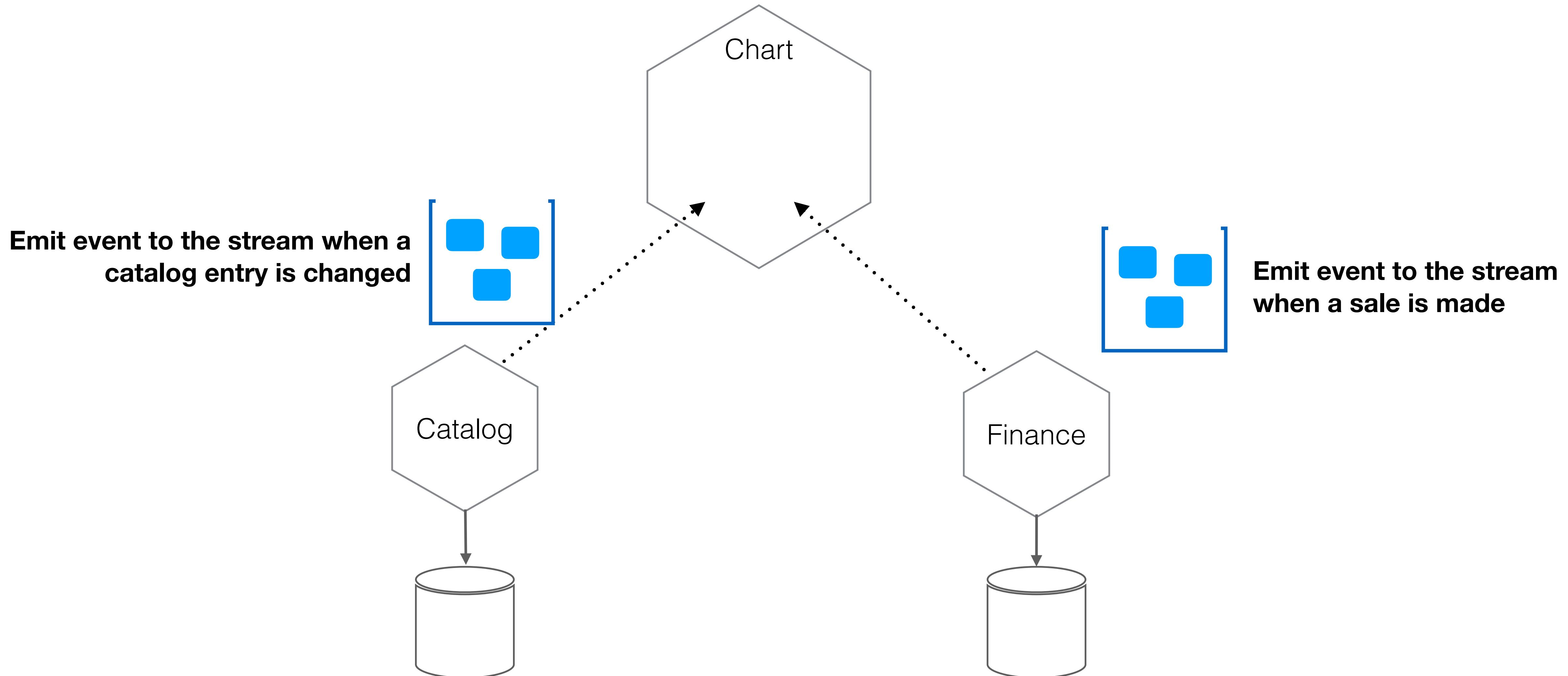
KSQLDB EXAMPLE ARCHITECTURE



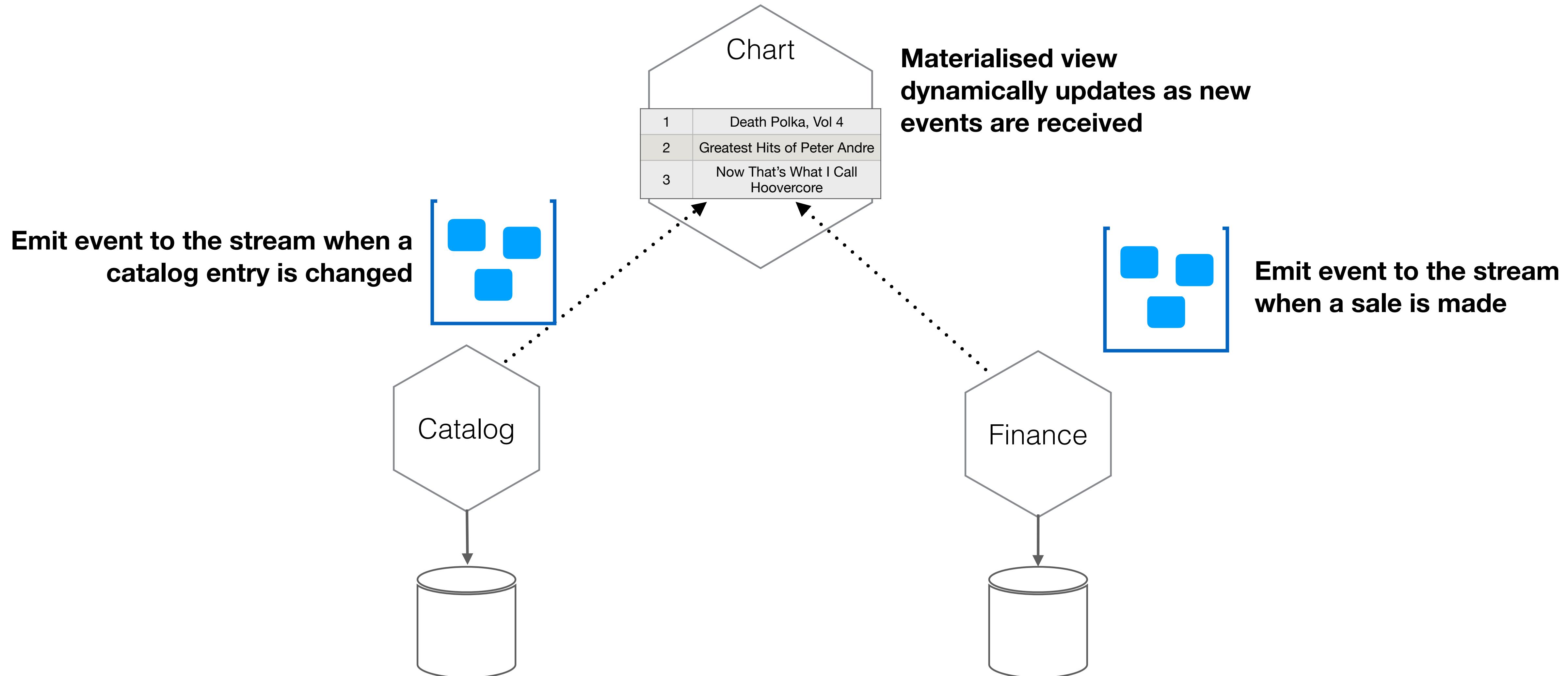
KSQLDB EXAMPLE ARCHITECTURE



KSQLDB EXAMPLE ARCHITECTURE



KSQLDB EXAMPLE ARCHITECTURE

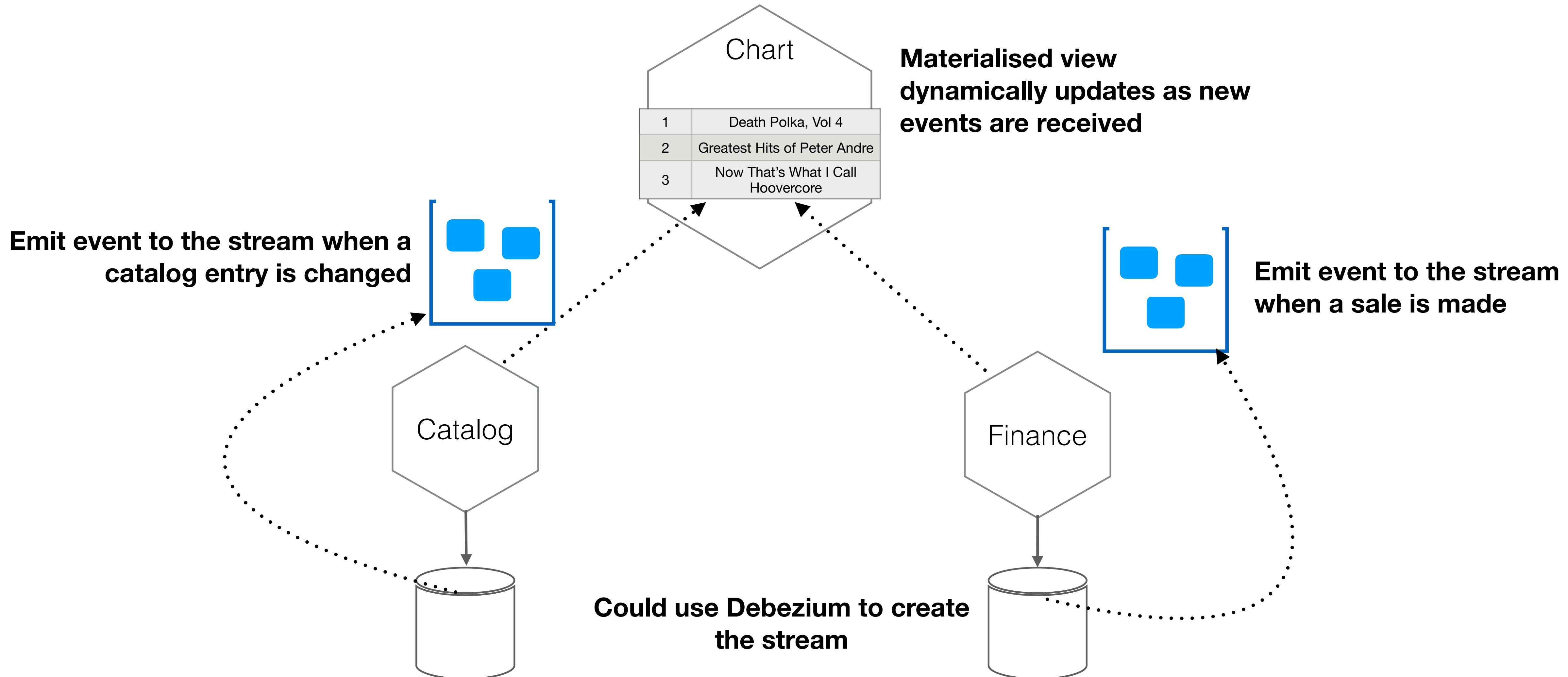


Emit event to the stream when a catalog entry is changed

Materialised view dynamically updates as new events are received

Emit event to the stream when a sale is made

KSQLDB EXAMPLE ARCHITECTURE



THE CACHING PRIME DIRECTIVE

THE CACHING PRIME DIRECTIVE

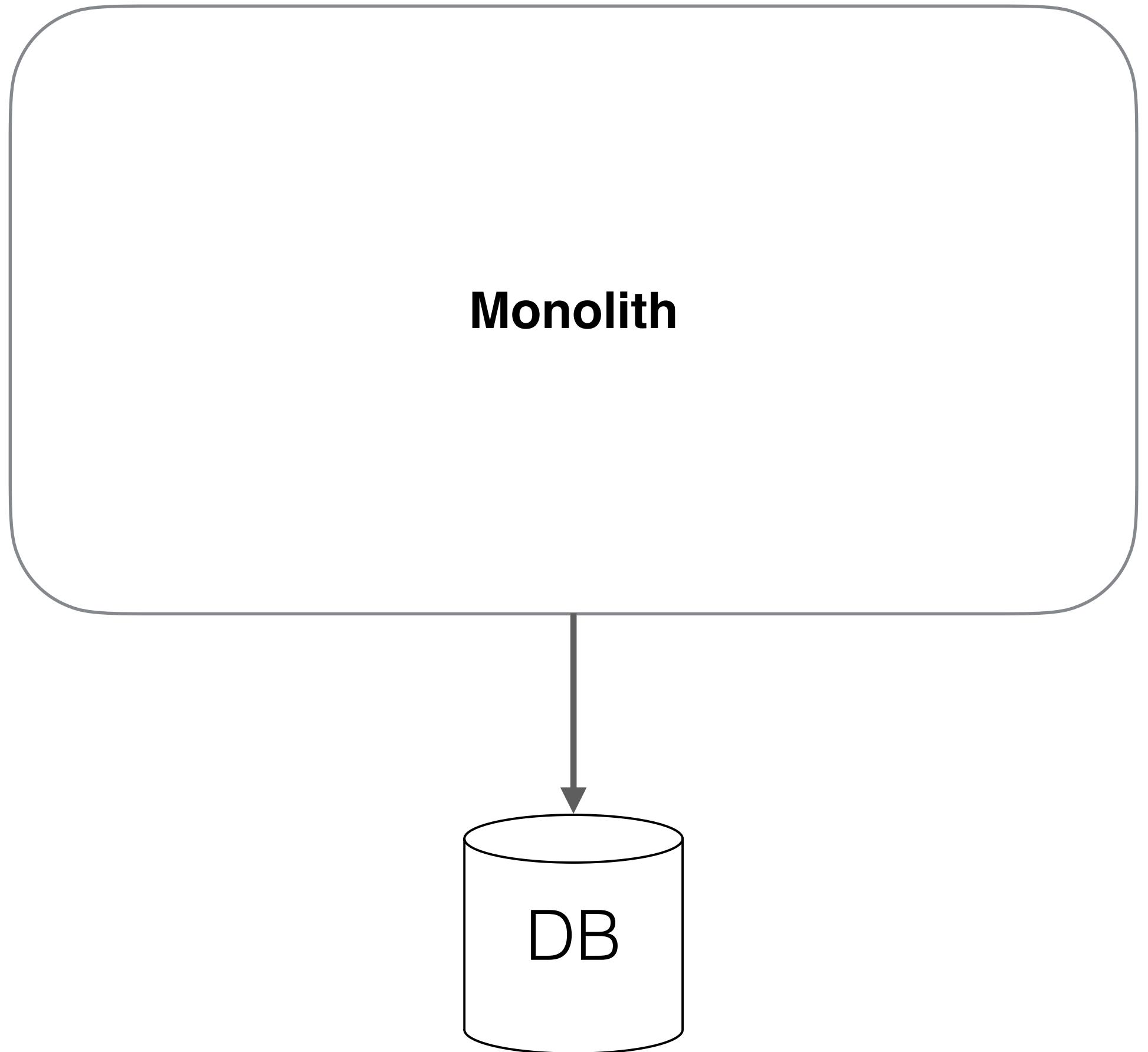
Cache in as few places as possible

THE CACHING PRIME DIRECTIVE

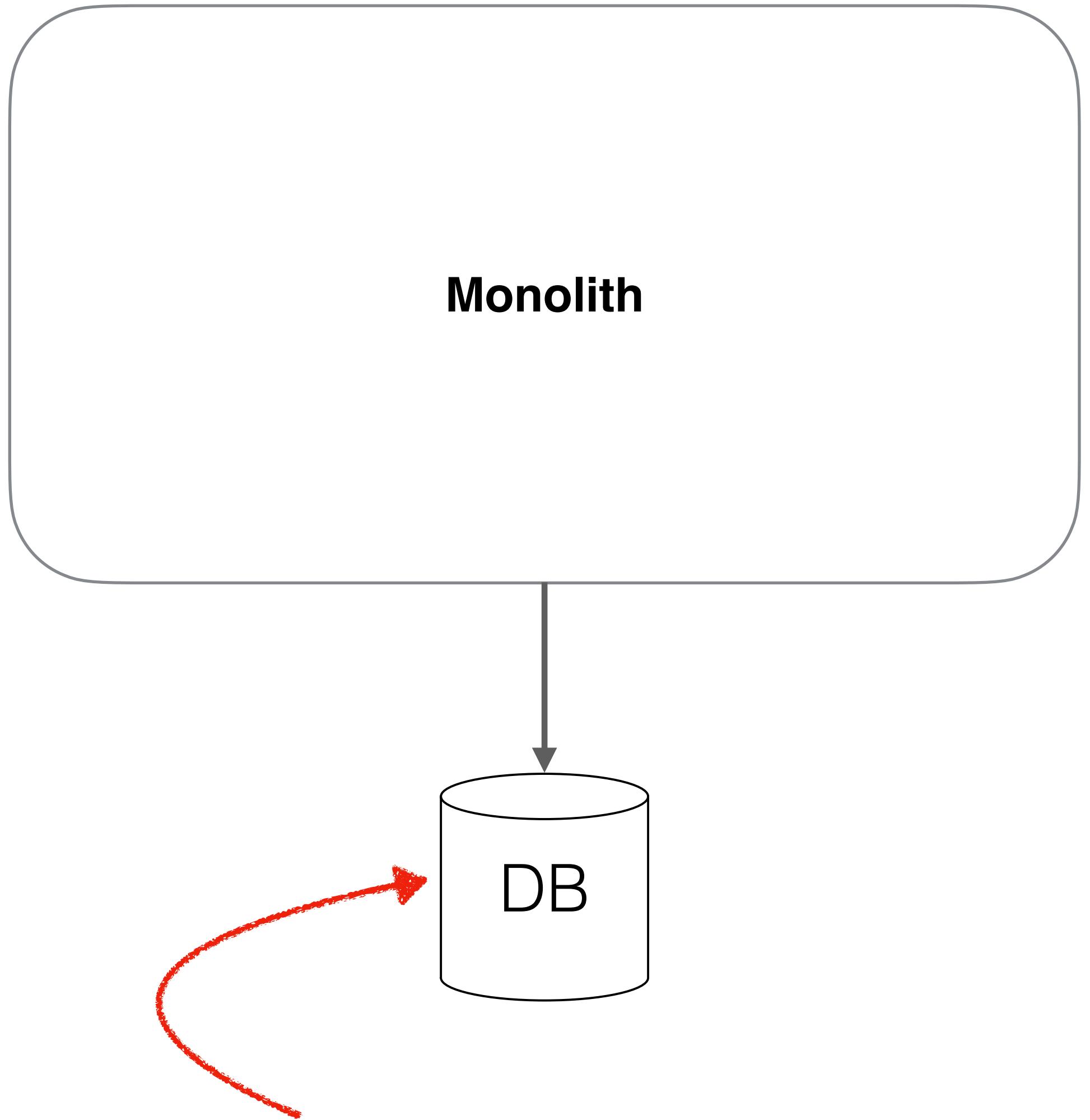
Cache in as few places as possible

Zero is the best number of caches

SINGLE SOURCE OF TRUTH?

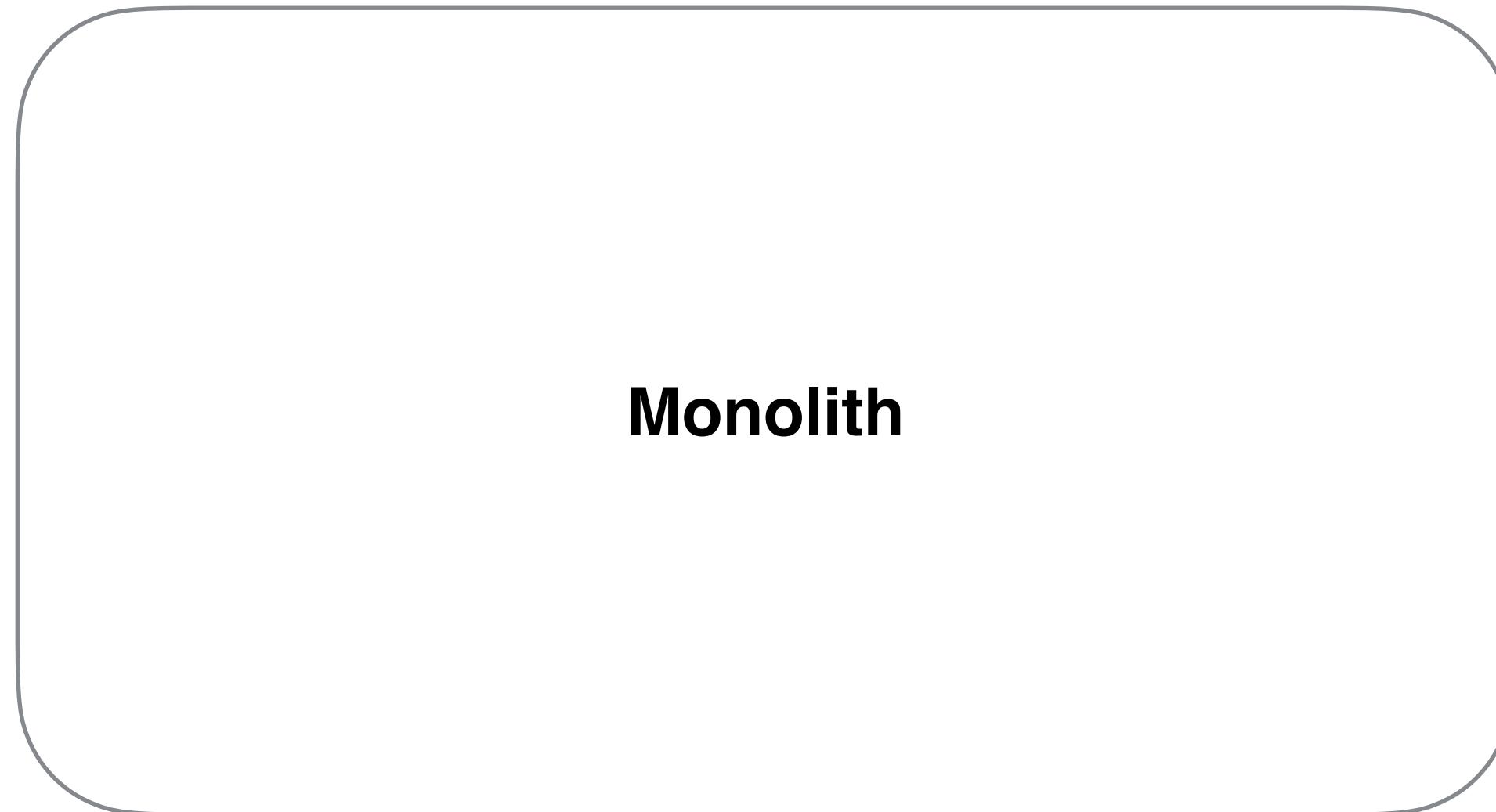


SINGLE SOURCE OF TRUTH?



This is the source of truth for invoicing data!

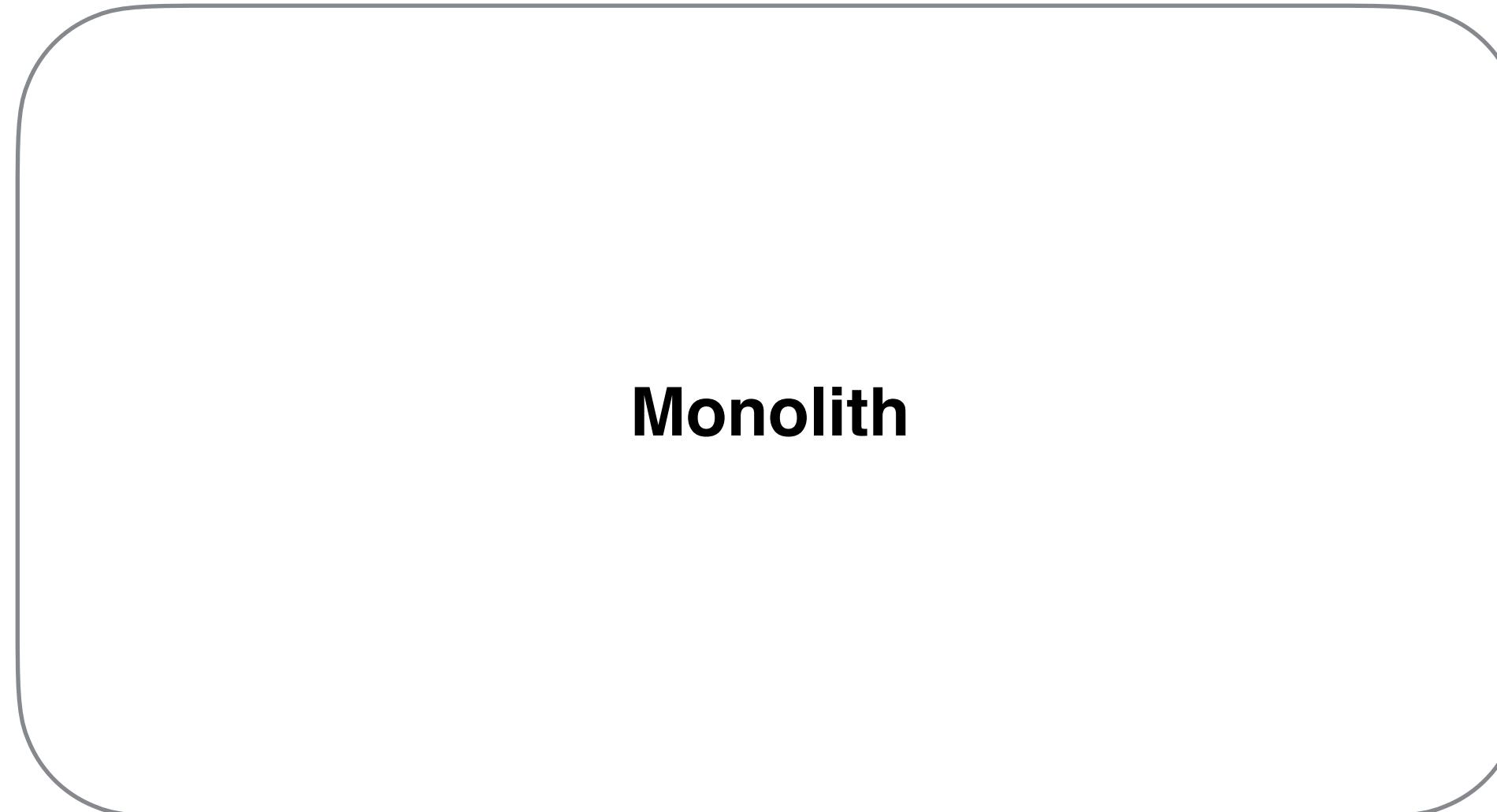
SINGLE SOURCE OF TRUTH?



Helps manage consistency of data

This is the source of truth for invoicing data!

SINGLE SOURCE OF TRUTH?



Monolith

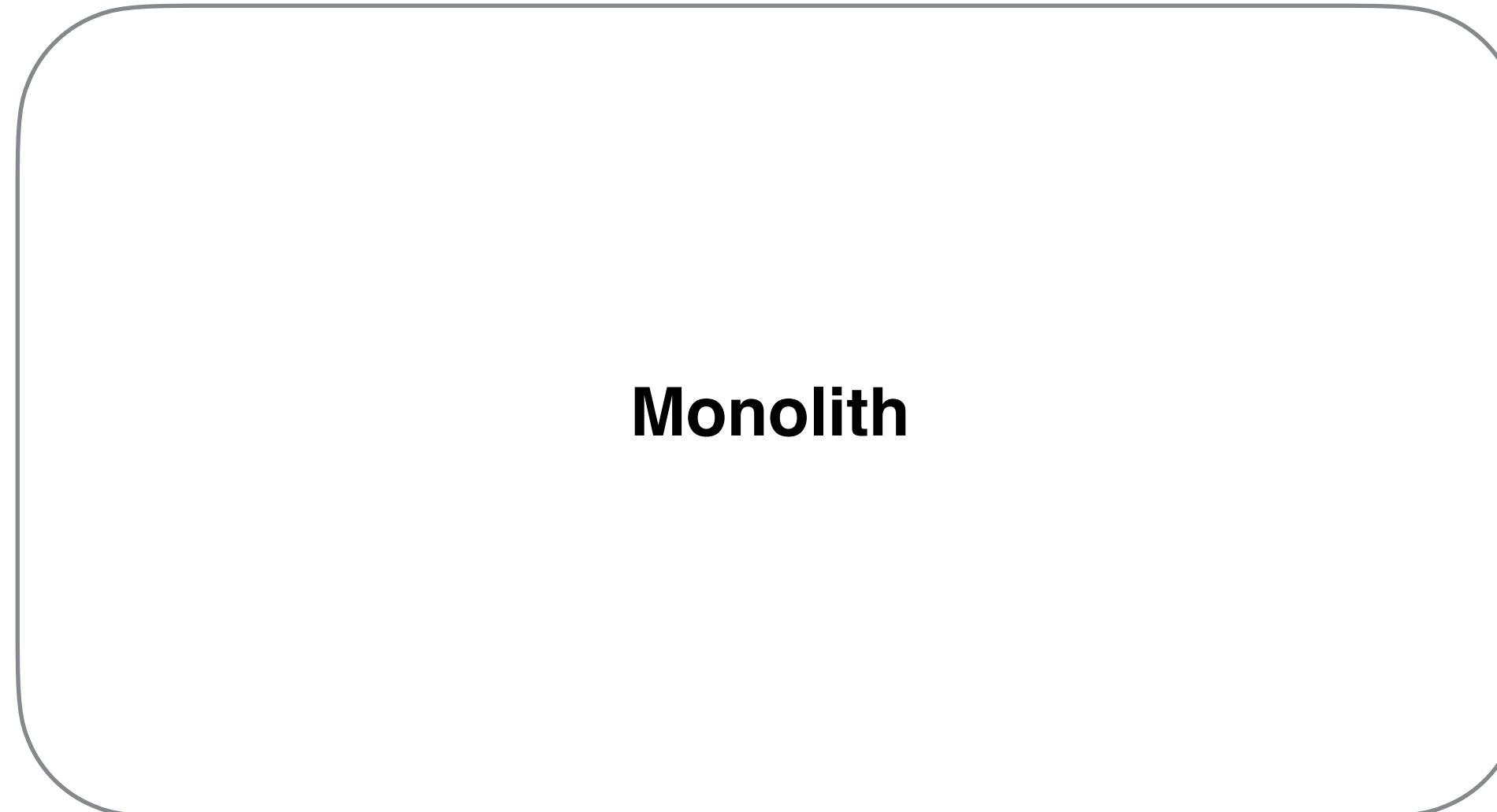
DB

This is the source of truth for invoicing data!

Helps manage consistency of data

Easier to control access

SINGLE SOURCE OF TRUTH?



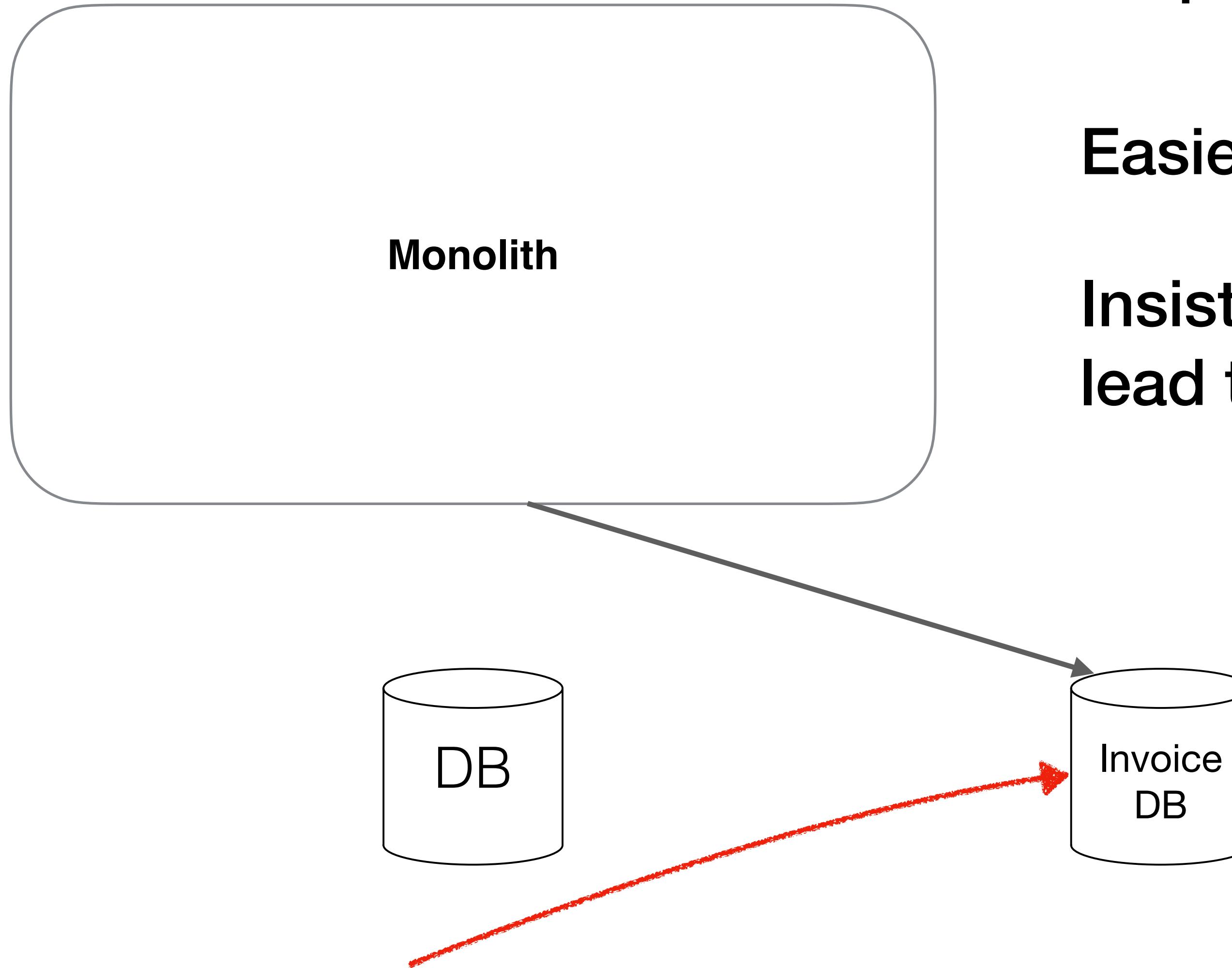
This is the source of truth for invoicing data!

Helps manage consistency of data

Easier to control access

Insisting on one source of truth can lead to migrations being “big bang”

SINGLE SOURCE OF TRUTH?



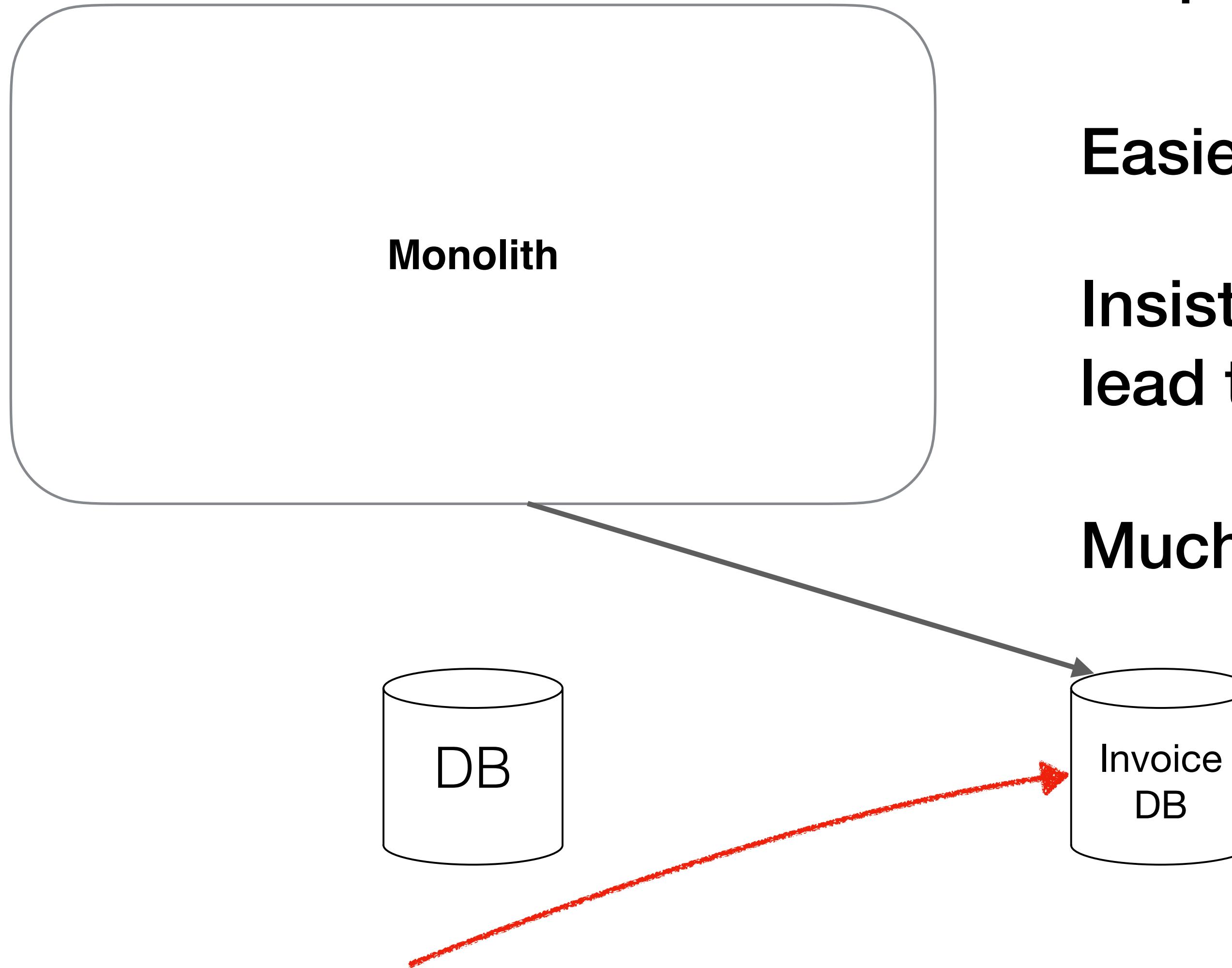
Helps manage consistency of data

Easier to control access

Insisting on one source of truth can lead to migrations being “big bang”

This is the source of truth for invoicing data!

SINGLE SOURCE OF TRUTH?



Helps manage consistency of data

Easier to control access

Insisting on one source of truth can lead to migrations being “big bang”

Much more risky switchover

This is the source of truth for invoicing data!

LOUDNESS

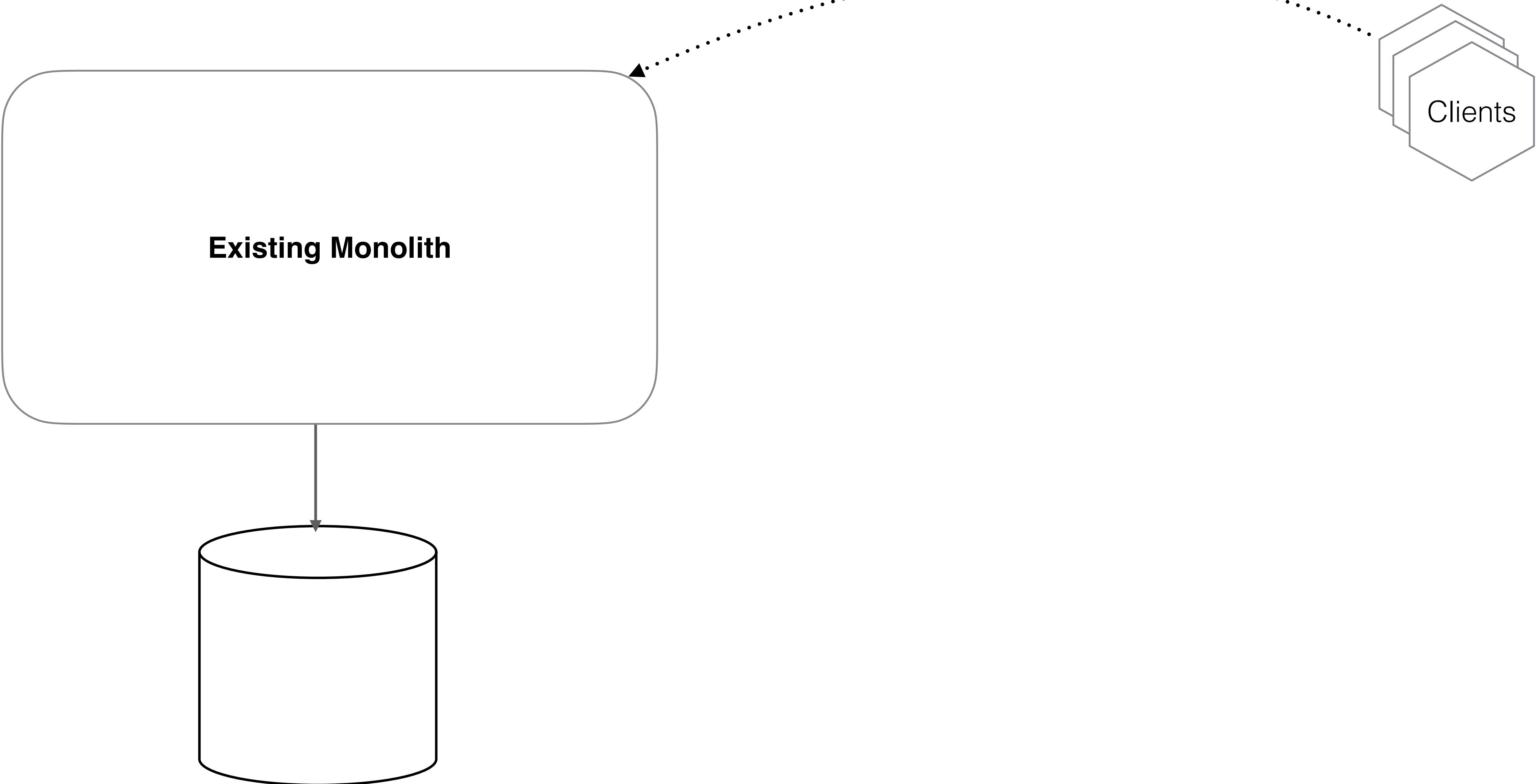
10



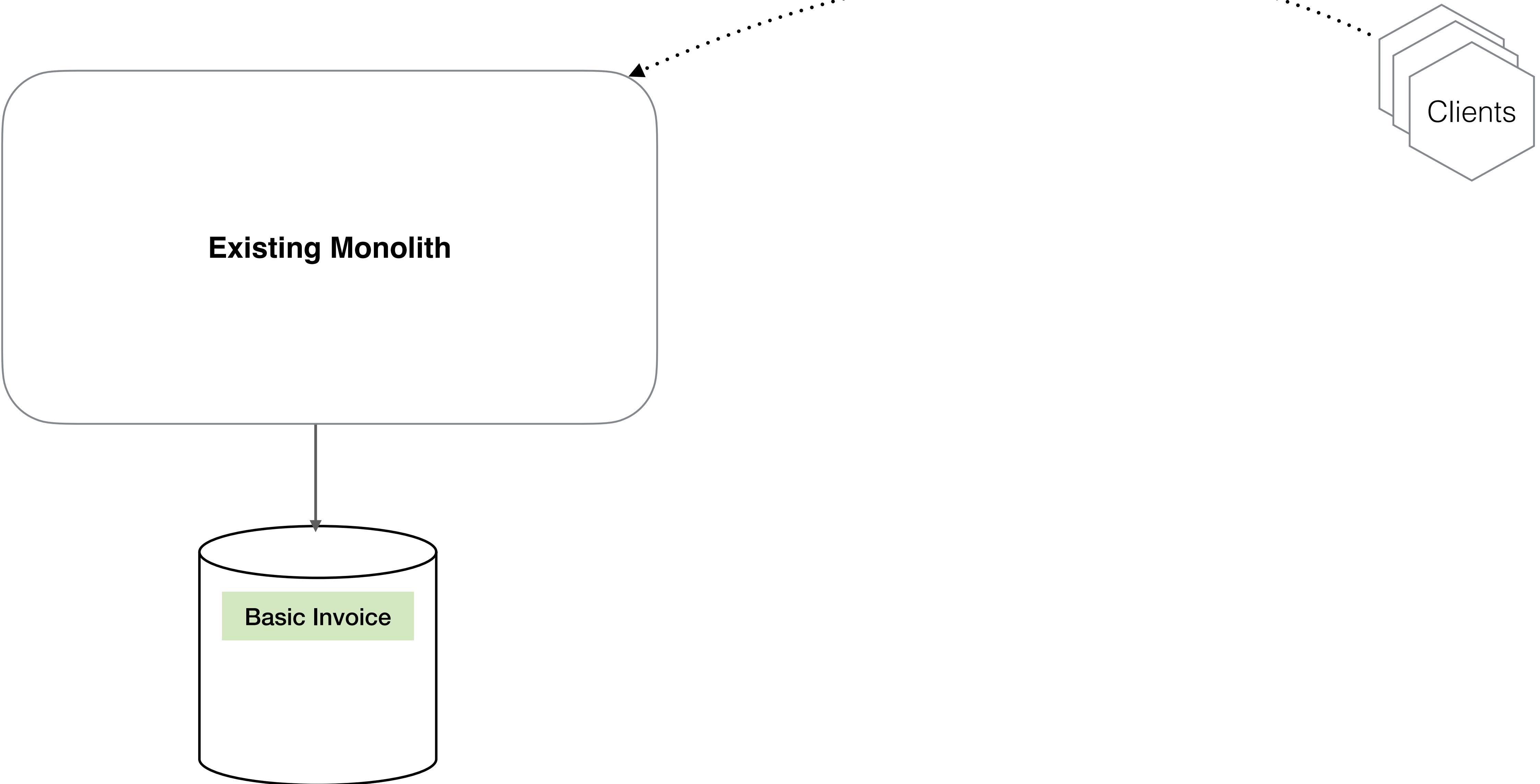


Square

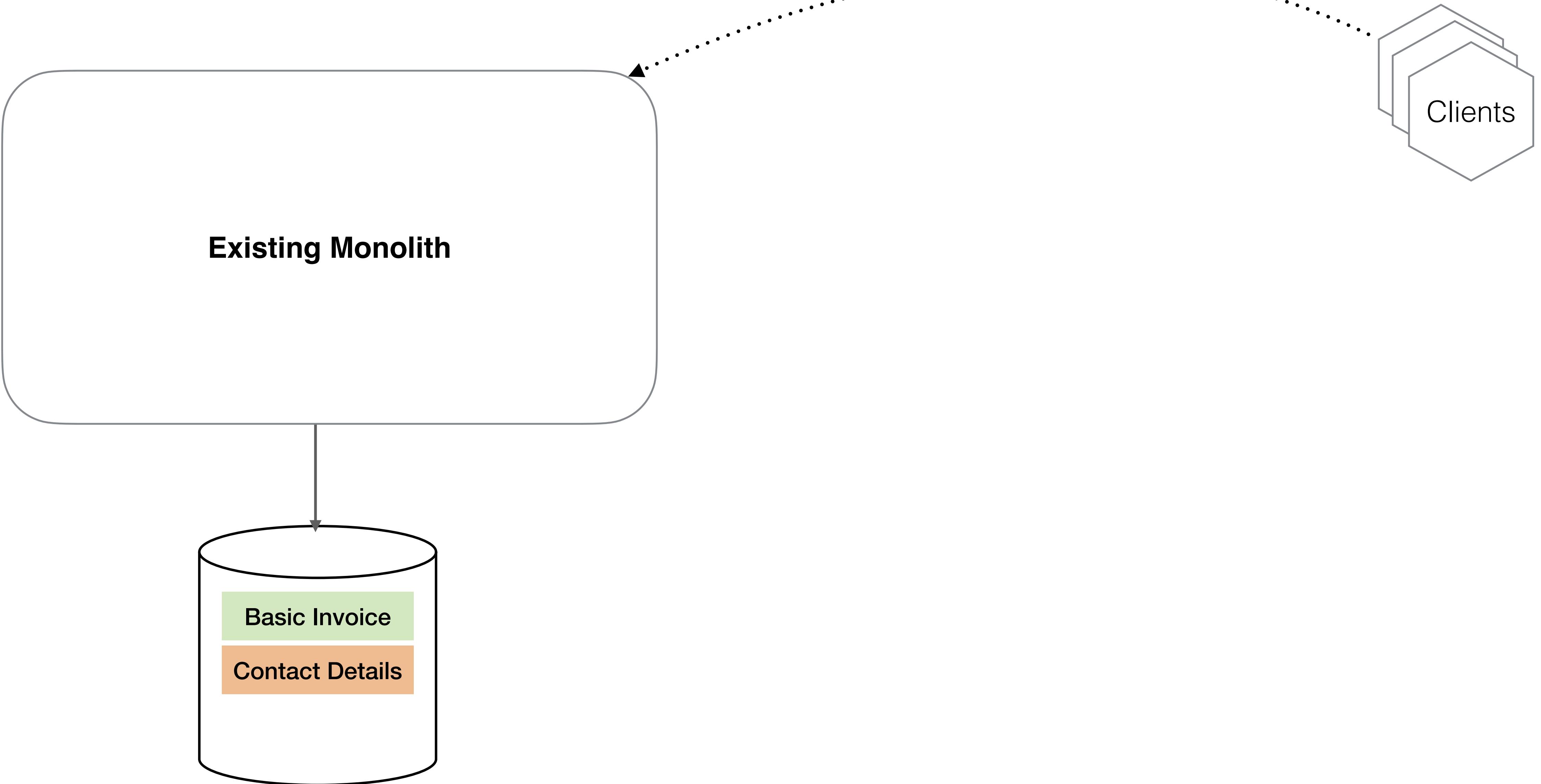
PATTERN: TRACER WRITE



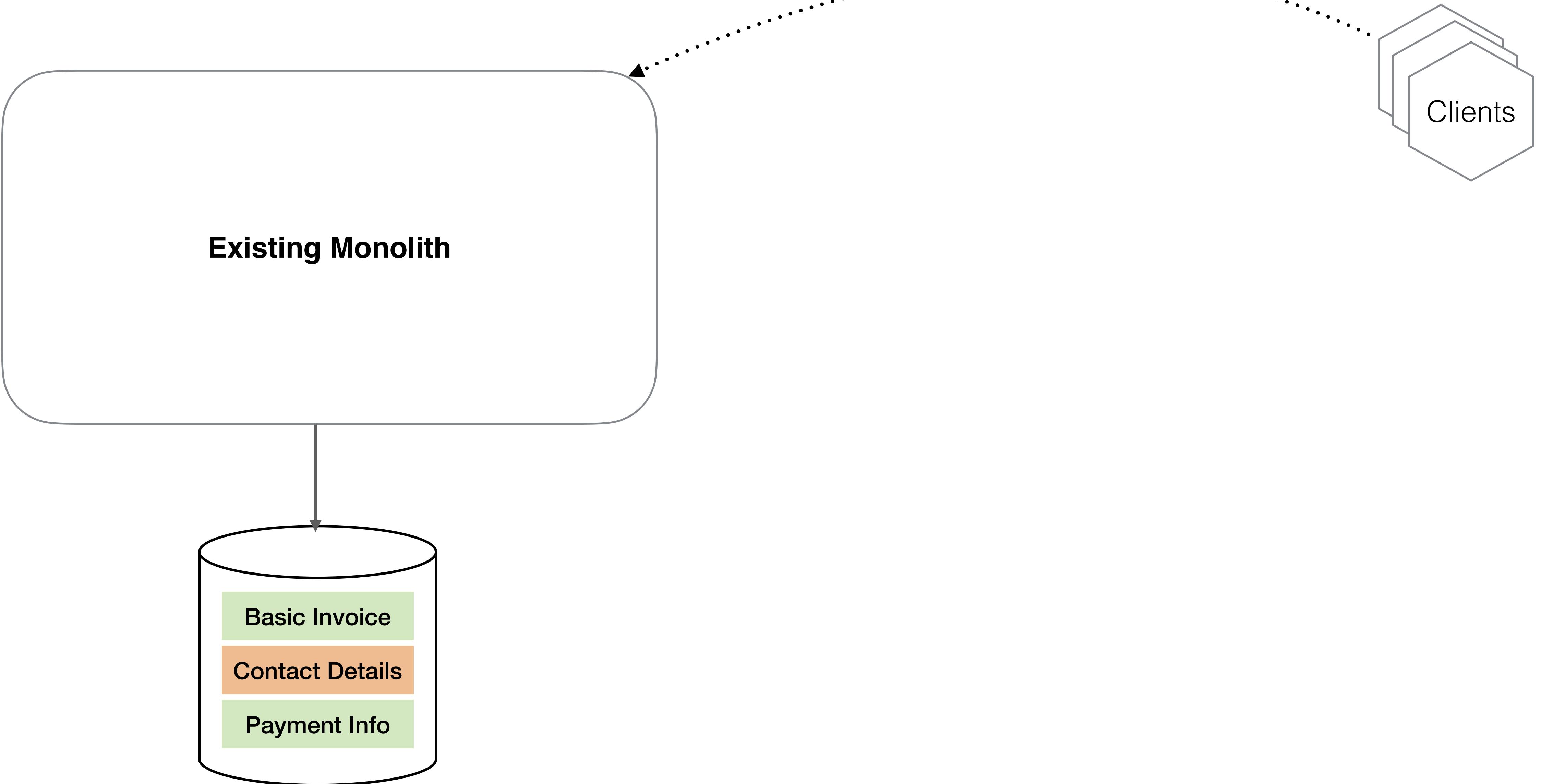
PATTERN: TRACER WRITE



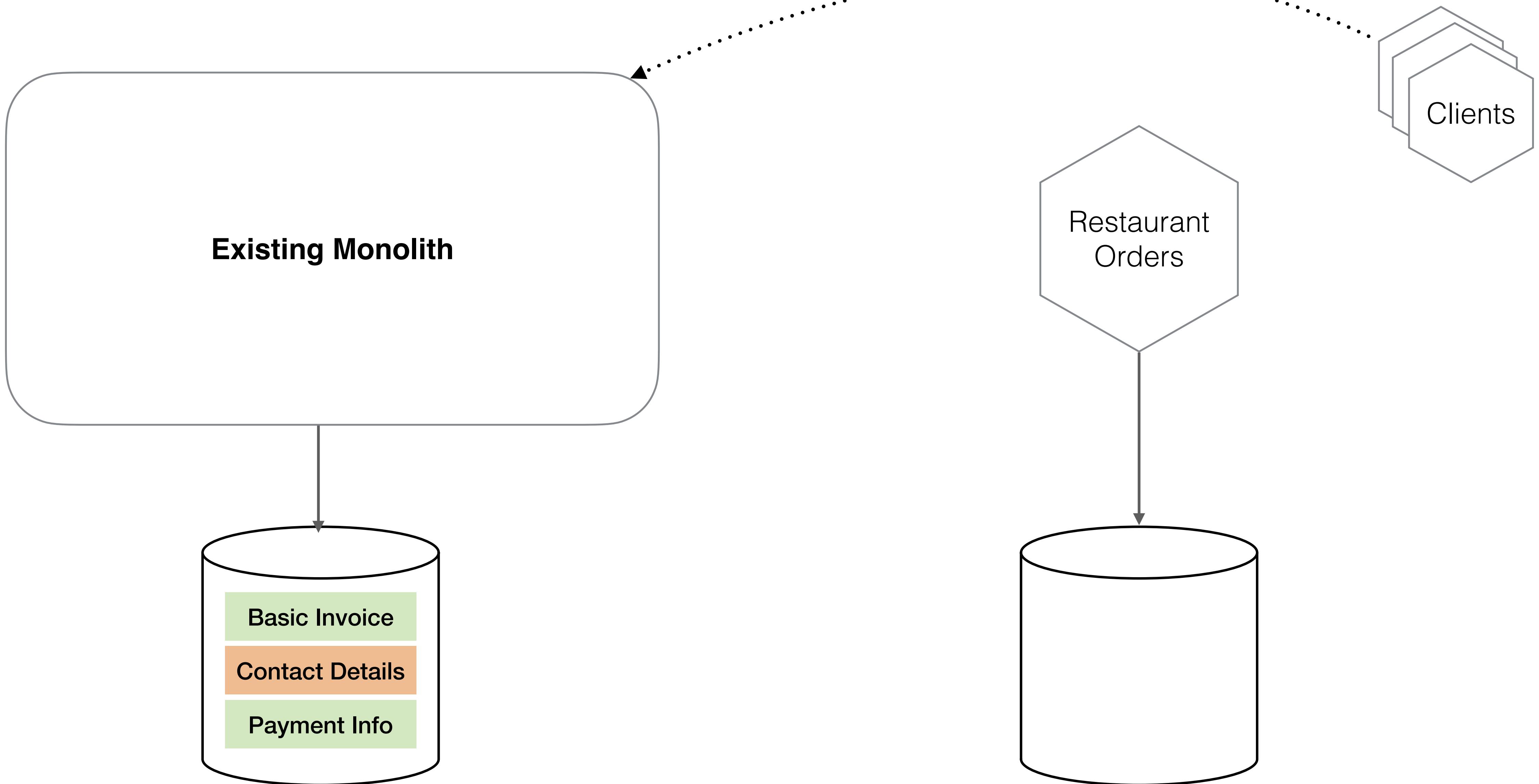
PATTERN: TRACER WRITE



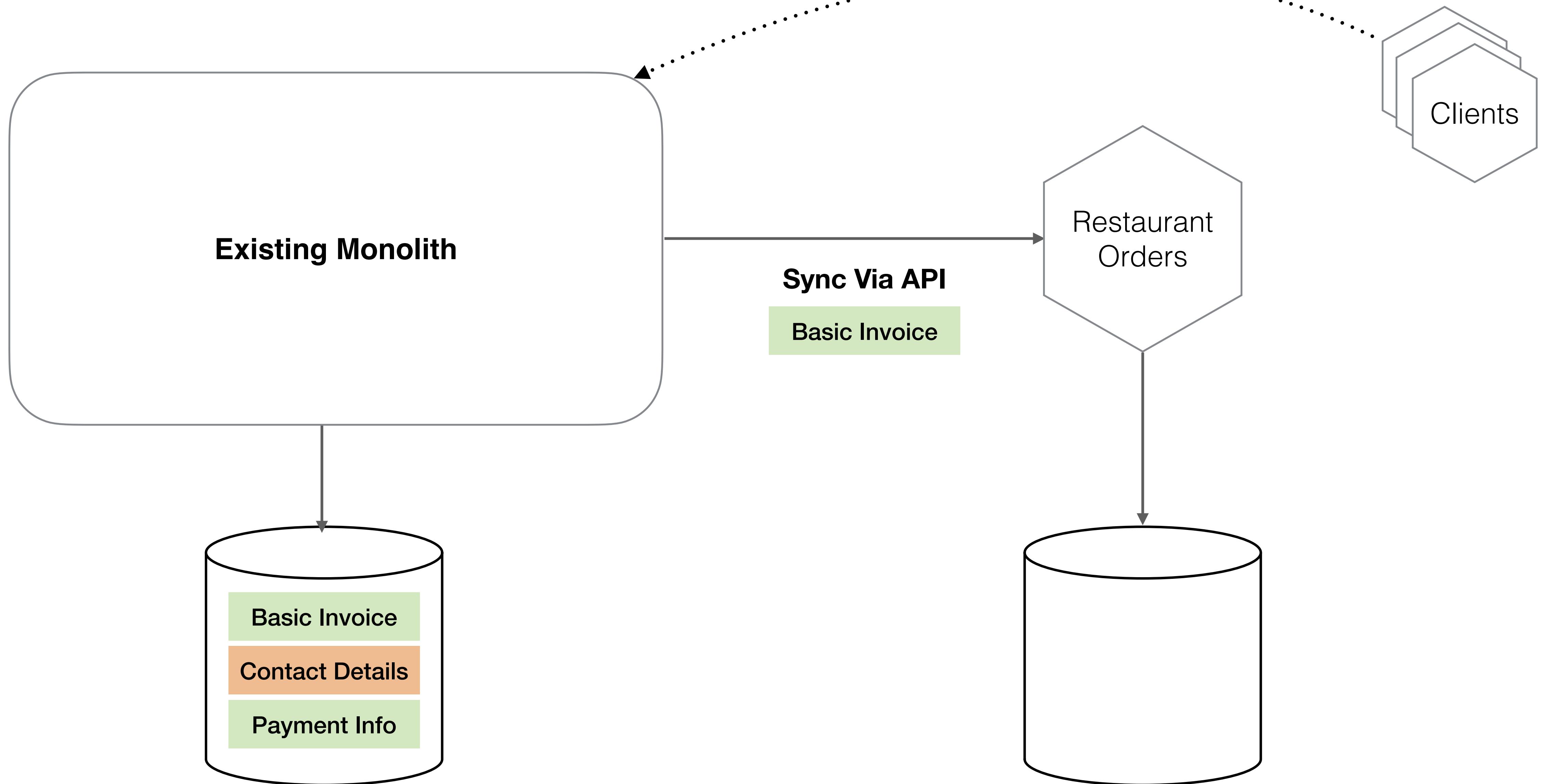
PATTERN: TRACER WRITE



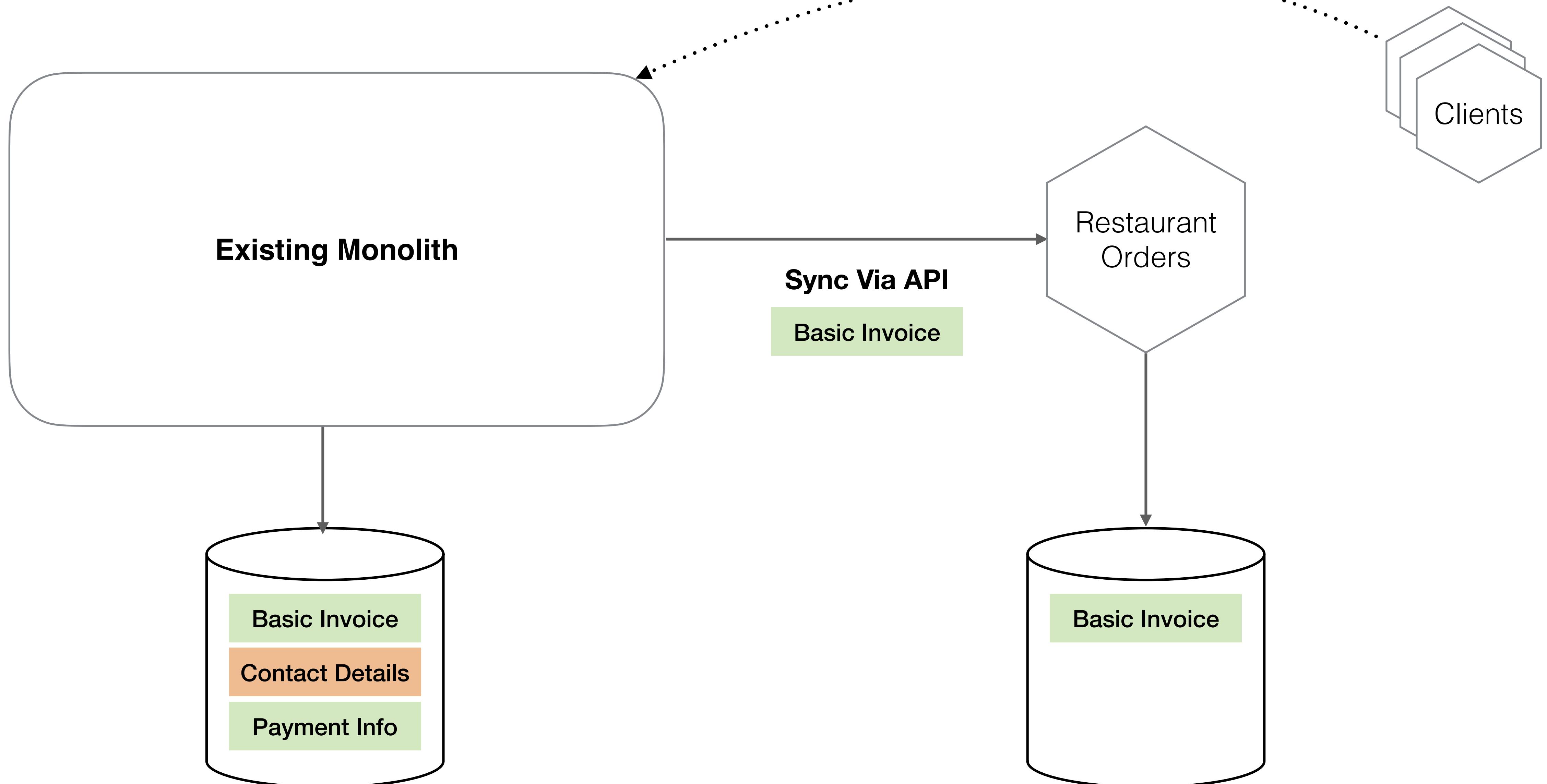
PATTERN: TRACER WRITE



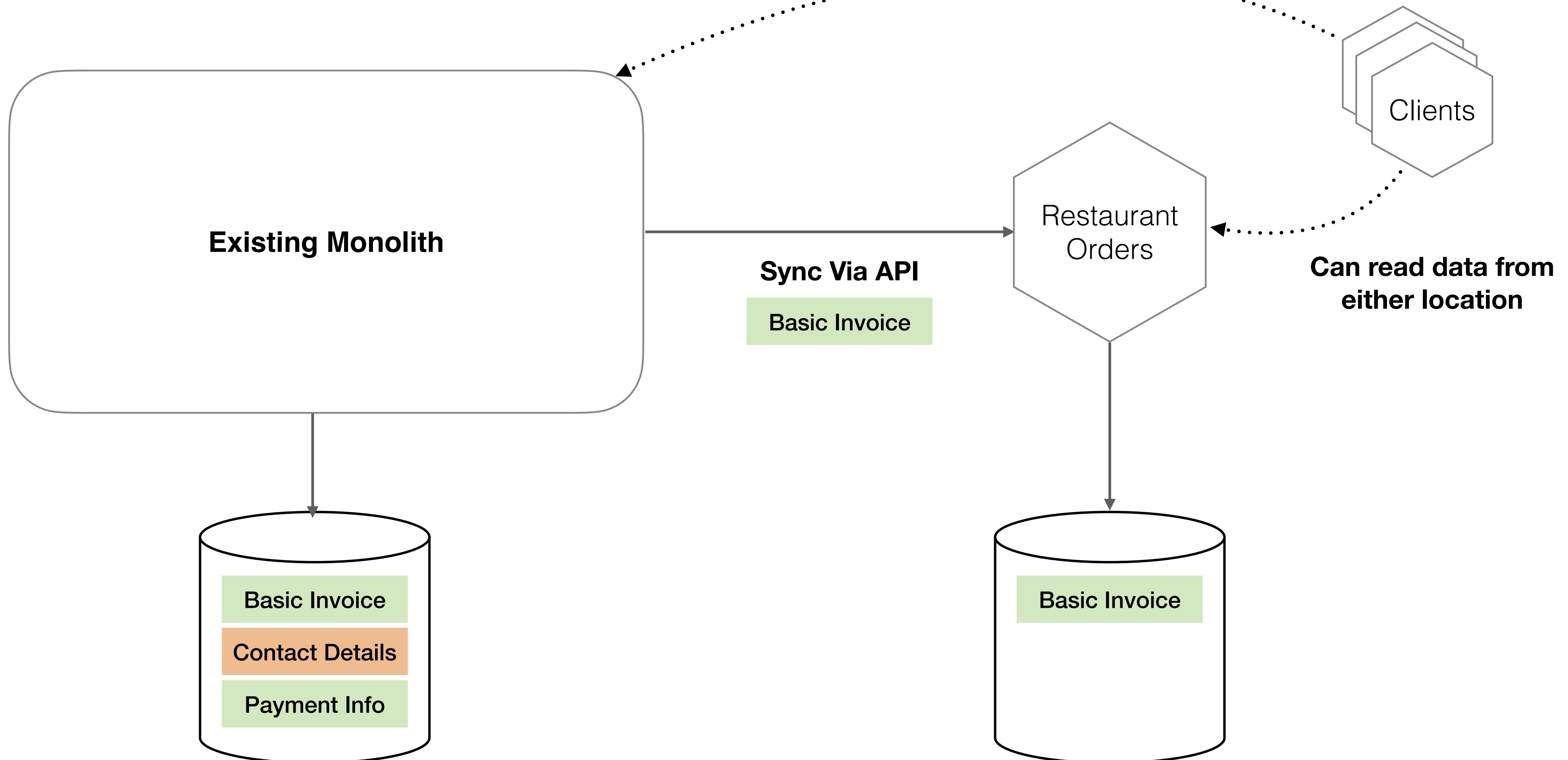
PATTERN: TRACER WRITE



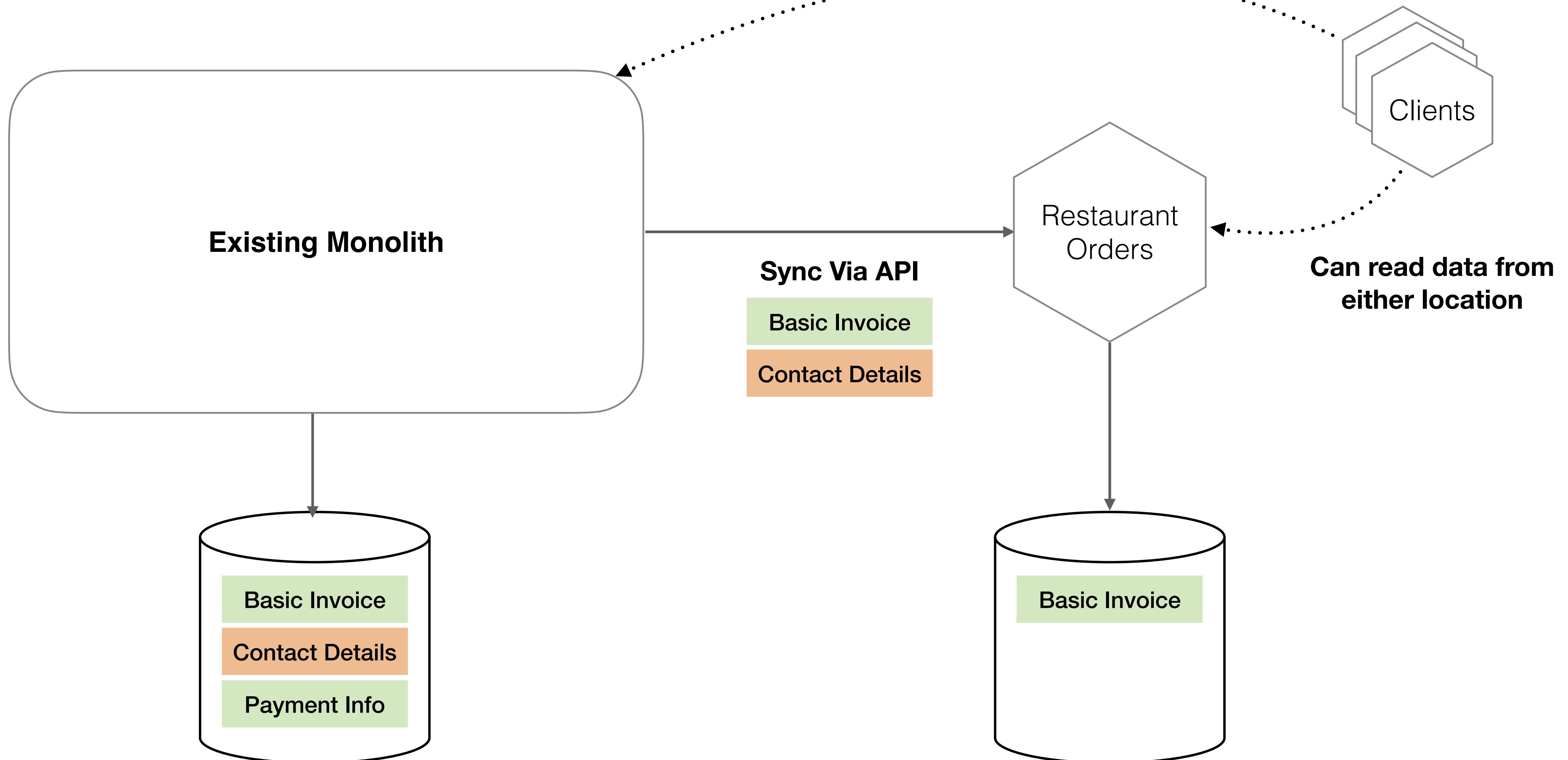
PATTERN: TRACER WRITE



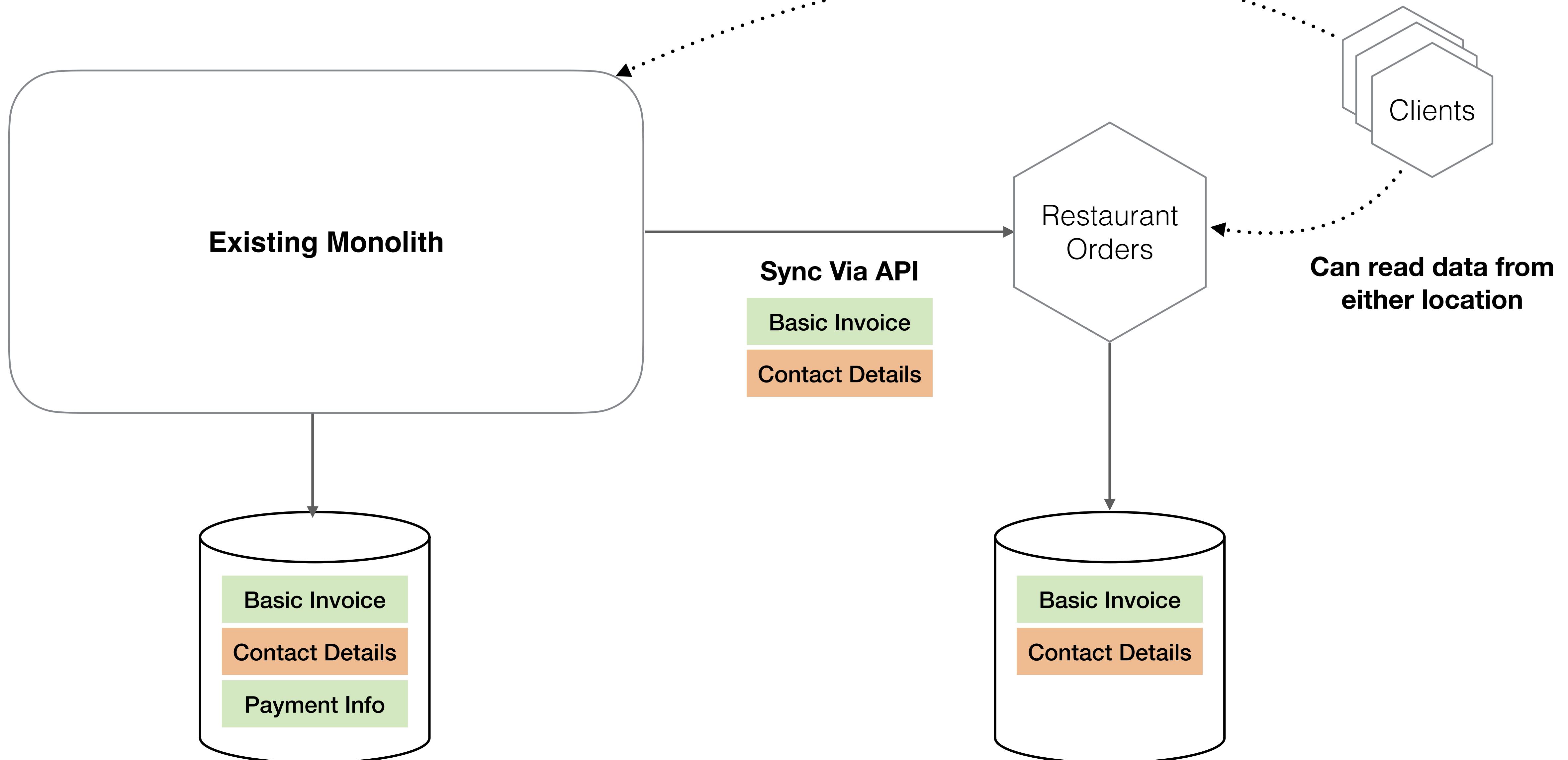
PATTERN: TRACER WRITE



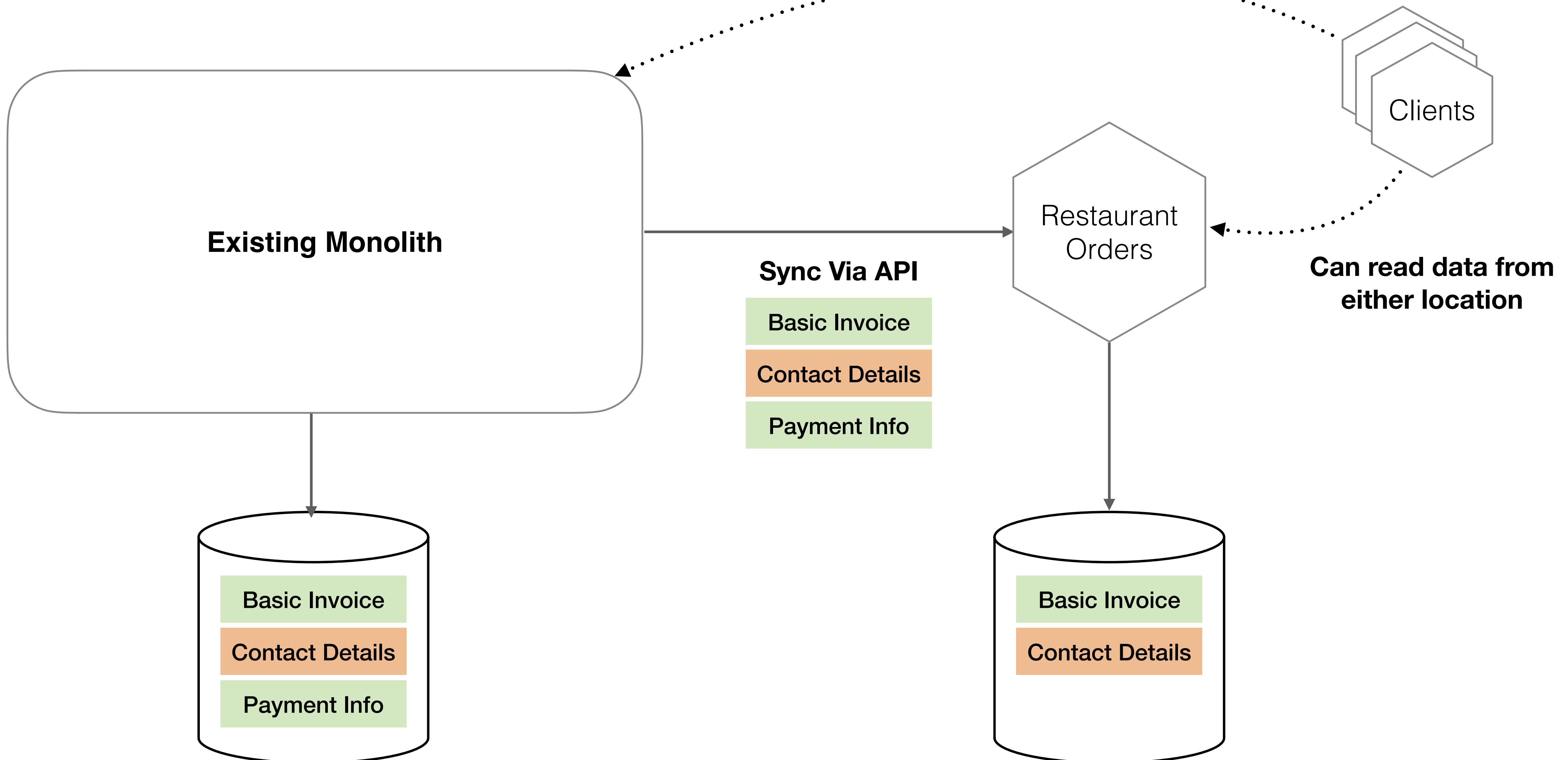
PATTERN: TRACER WRITE



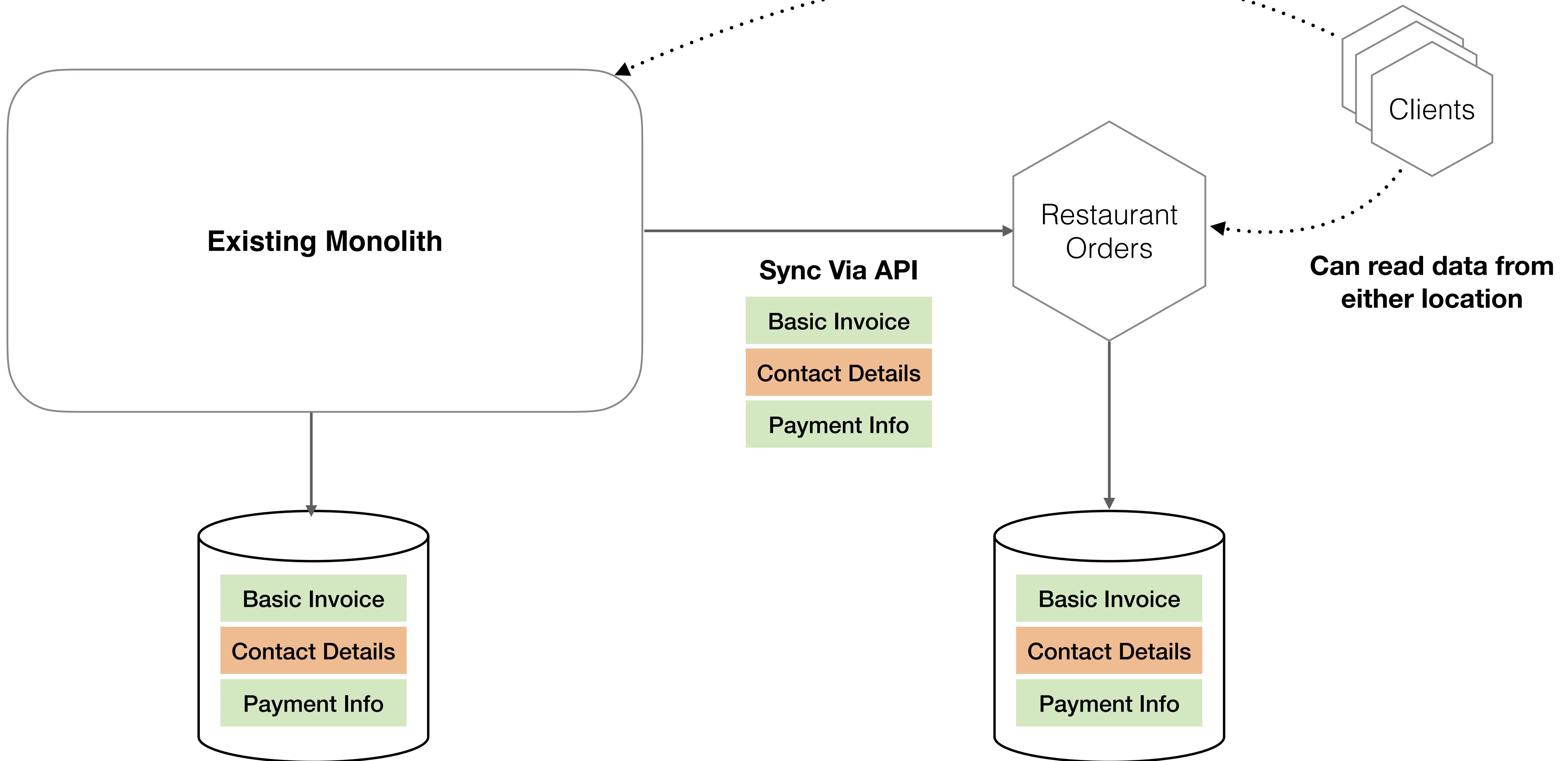
PATTERN: TRACER WRITE



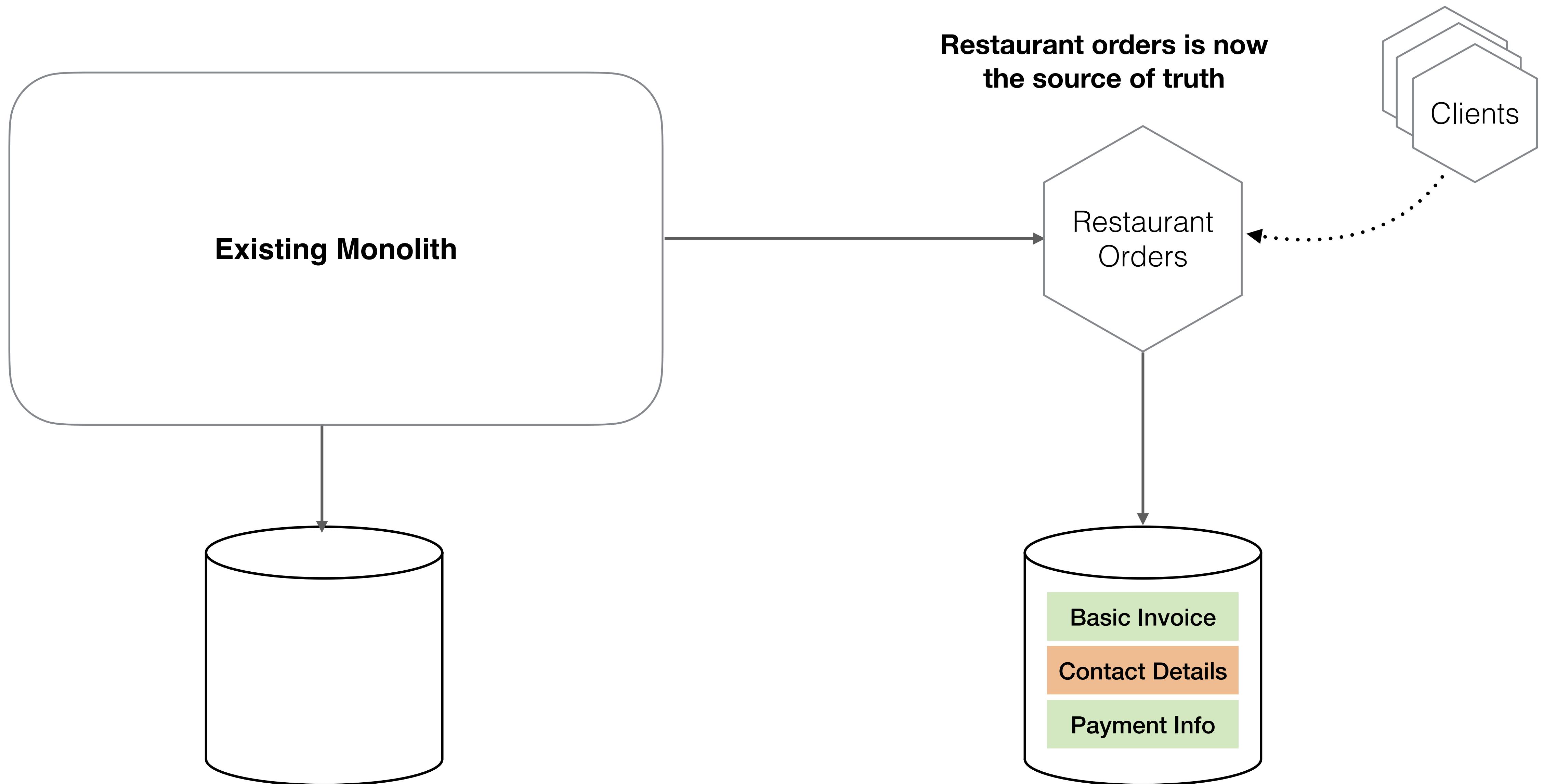
PATTERN: TRACER WRITE



PATTERN: TRACER WRITE



PATTERN: TRACER WRITE



**Restaurant orders is now
the source of truth**

POLL: DO YOU FEEL MORE CONFIDENT IN CHANGING HOW YOU ACCESS DATA?

Yes - we can change so many things!

Maybe - there are a few things here I can try

No - this has scared me off!

IN SUMMARY

IN SUMMARY

**Incremental decomposition of databases
isn't just possible, it's essential**

IN SUMMARY

**Incremental decomposition of databases
isn't just possible, it's essential**

**Break big changes into lots of
small changes**

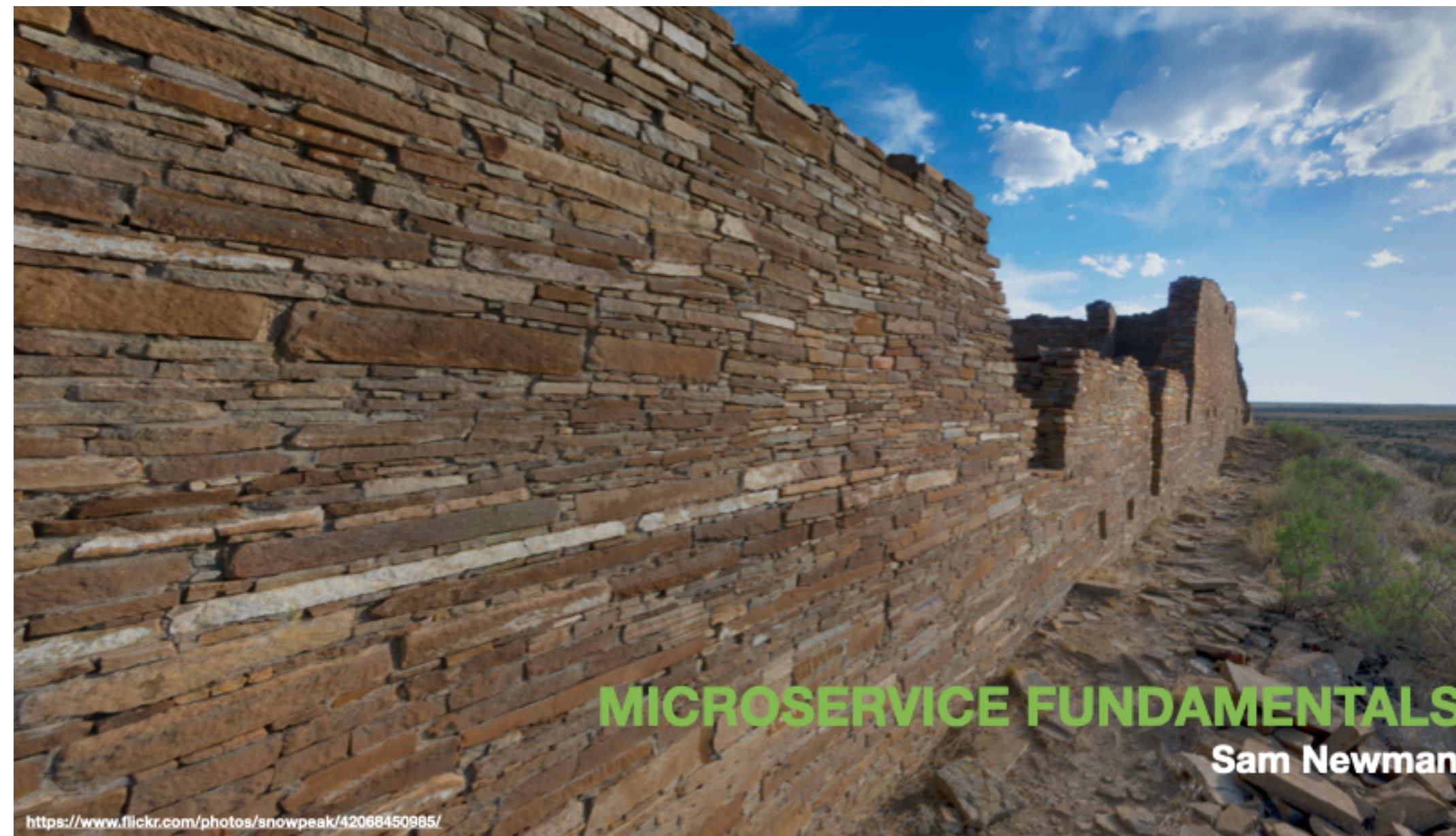
IN SUMMARY

**Incremental decomposition of databases
isn't just possible, it's essential**

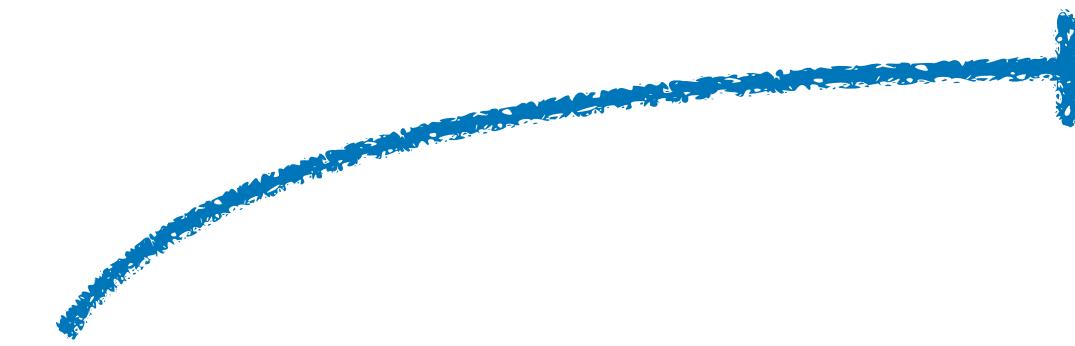
**Break big changes into lots of
small changes**

**The big shared database should be
avoided - and now you know how!**

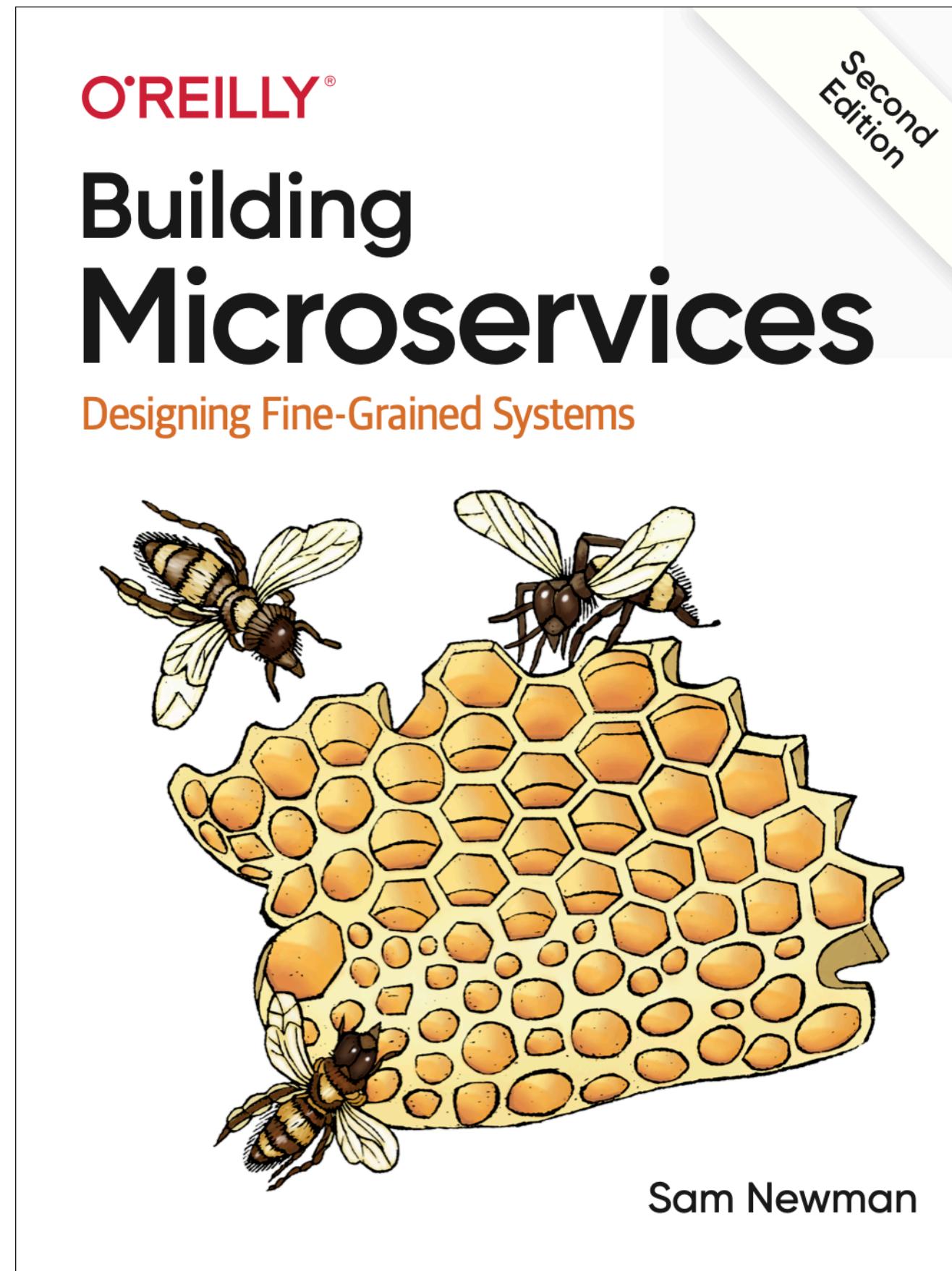
OTHER COURSES



<http://bit.ly/snewman-olt>



THANKS!



Sam Newman. Home About Talks Podcast Writing Contact

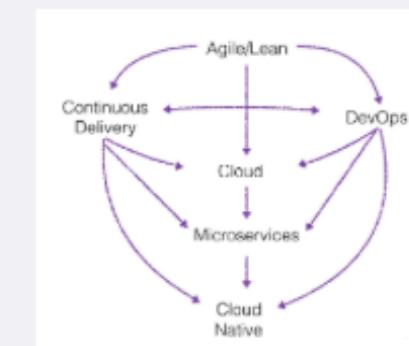
Talks & Workshops.

Here are a list of the talks I am currently presenting. On request, I can present different topics or even my older talks. If you want me to present these topics at your conference or company, then please contact me. You can also see where I'll be speaking next on my [events](#) page.

What Is This Cloud Native Thing Anyway? ■ 45min Talk

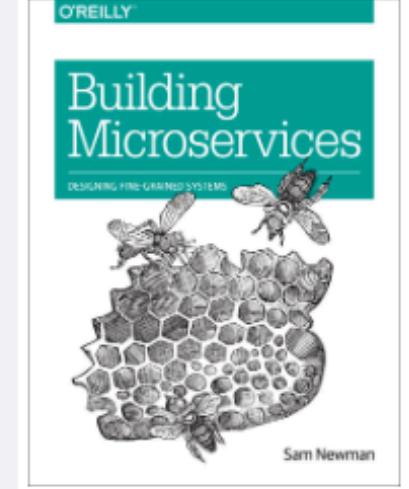
A talk exploring what the hell Cloud Native means

→ [Find Out More](#)



Feature Branches And Toggles In A Post-GitHub World. ■

1. Validate the integration
2. When the build breaks, fix it
3. Integrate daily

Book! ■ 
I have written a book called "Building Microservices", which is available now. Want to know more?
→ [Read on...](#)

Video! ■

@samnewman

<https://samnewman.io/>

@samnewman