



MONOLITH TO MICROSERVICES

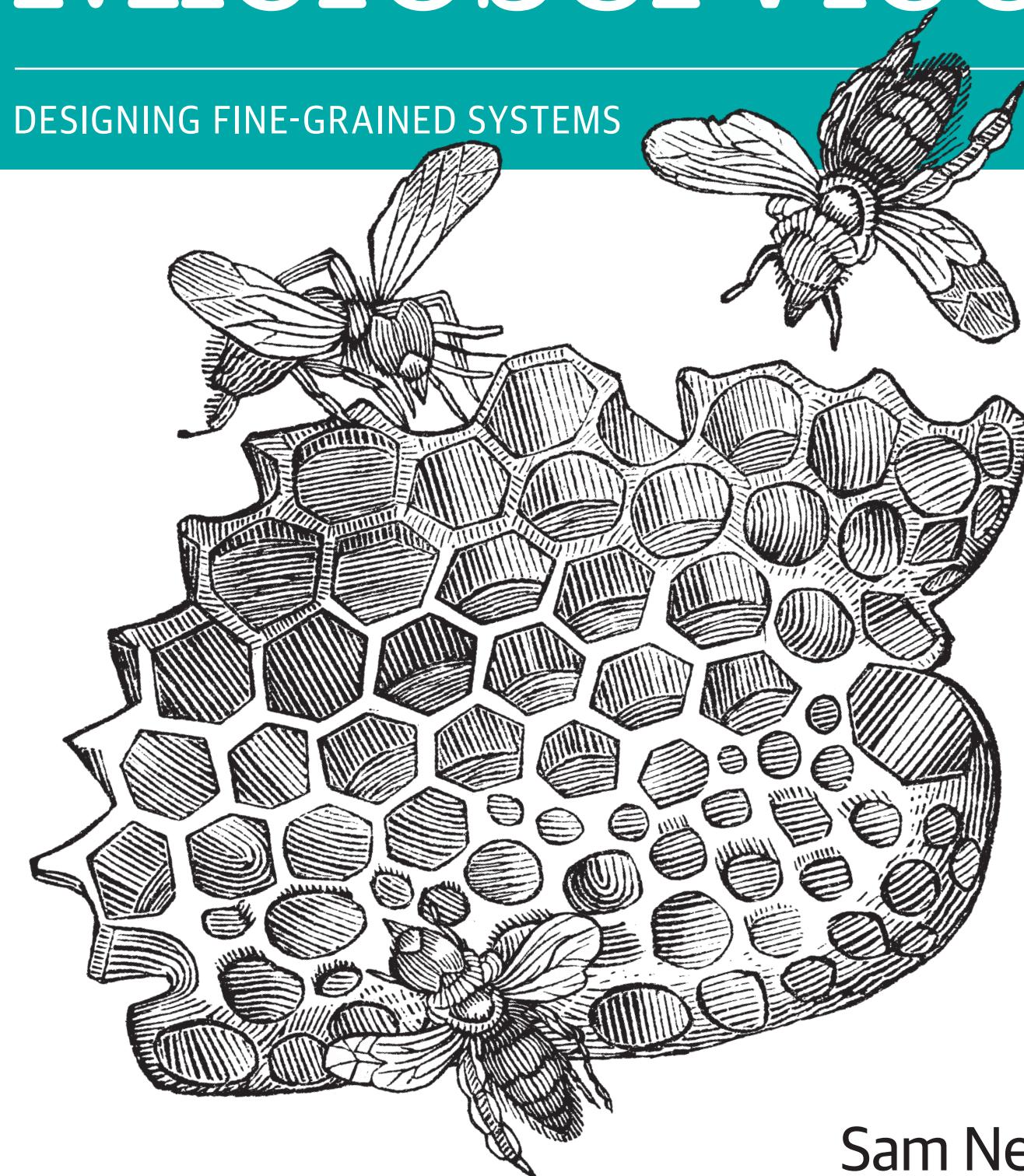
DATA DECOMPOSITION PATTERNS

Sam Newman

O'REILLY®

Building Microservices

DESIGNING FINE-GRAINED SYSTEMS



Sam Newman

**Sam
Newman
& Associates**



@samnewman

NEW BOOK!

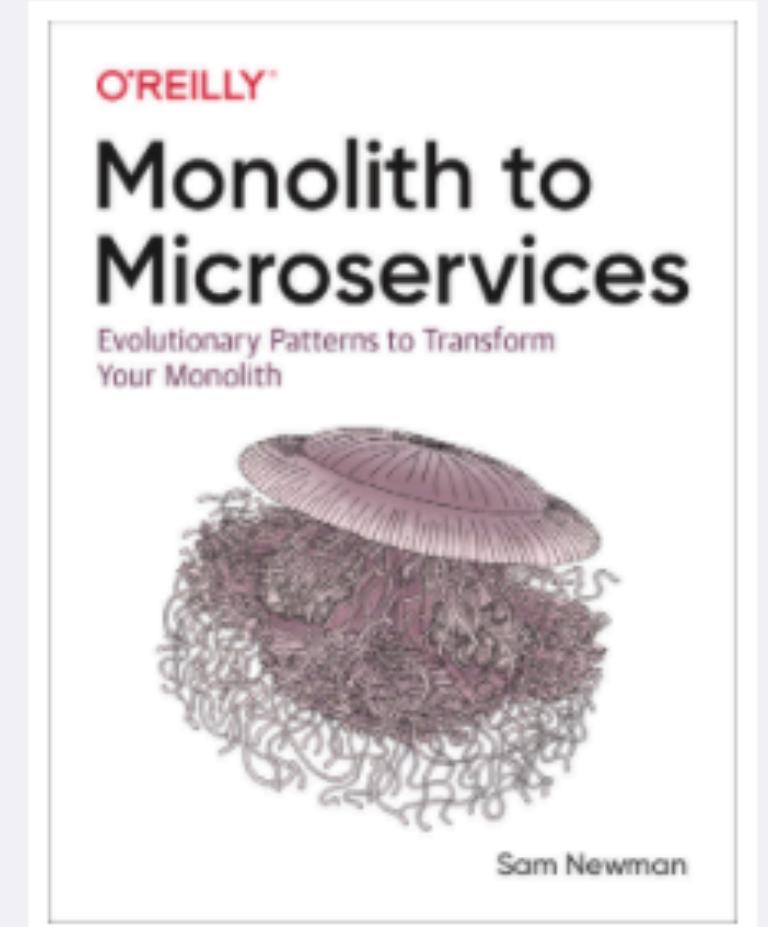
Monolith To Microservices.

Monolith To Microservices is a new book on system decomposition from O'Reilly

How do you detangle a monolithic system and migrate it to a microservices architecture? How do you do it while maintaining business-as-usual? As a companion to [Building Microservices](#), this new book details multiple approaches for helping you transition from existing monolithic systems to microservice architectures. This book is ideal if you're looking to evolve your current systems, rather than just rewriting everything from scratch.

How To Get The Book.

The book is now available, and you can order the dead-tree and Kindle versions over at Amazon. You can also read the book online on O'Reilly's online learning platform.



[Read on O'Reilly Learning Online](#)

[Order at Amazon.com](#)

[Order at Amazon.co.uk](#)

<https://samnewman.io/books/monolith-to-microservices/>

OTHER COURSES



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

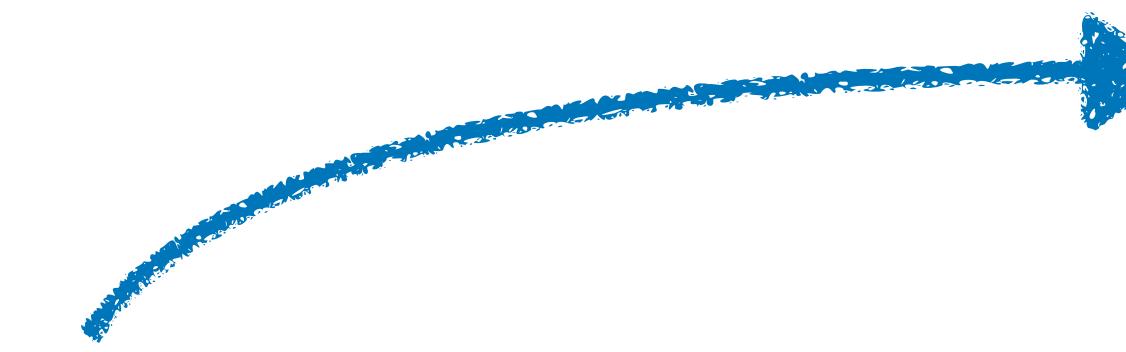
OTHER COURSES



MICROSERVICE FUNDAMENTALS

Sam Newman

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



**MONOLITH TO MICROSERVICES
APPLICATION DECOMPOSITION PATTERNS**

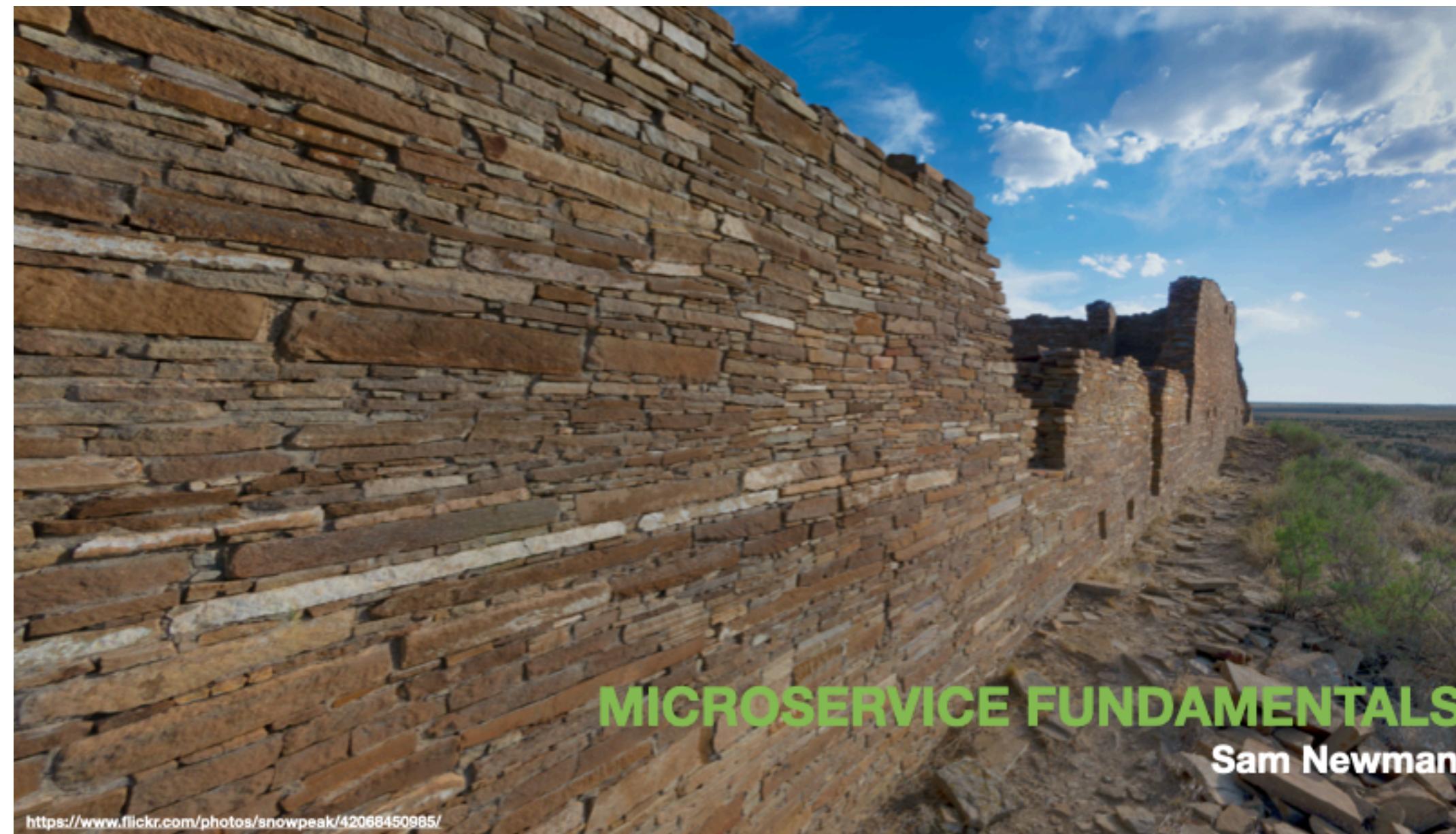
Sam Newman

<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

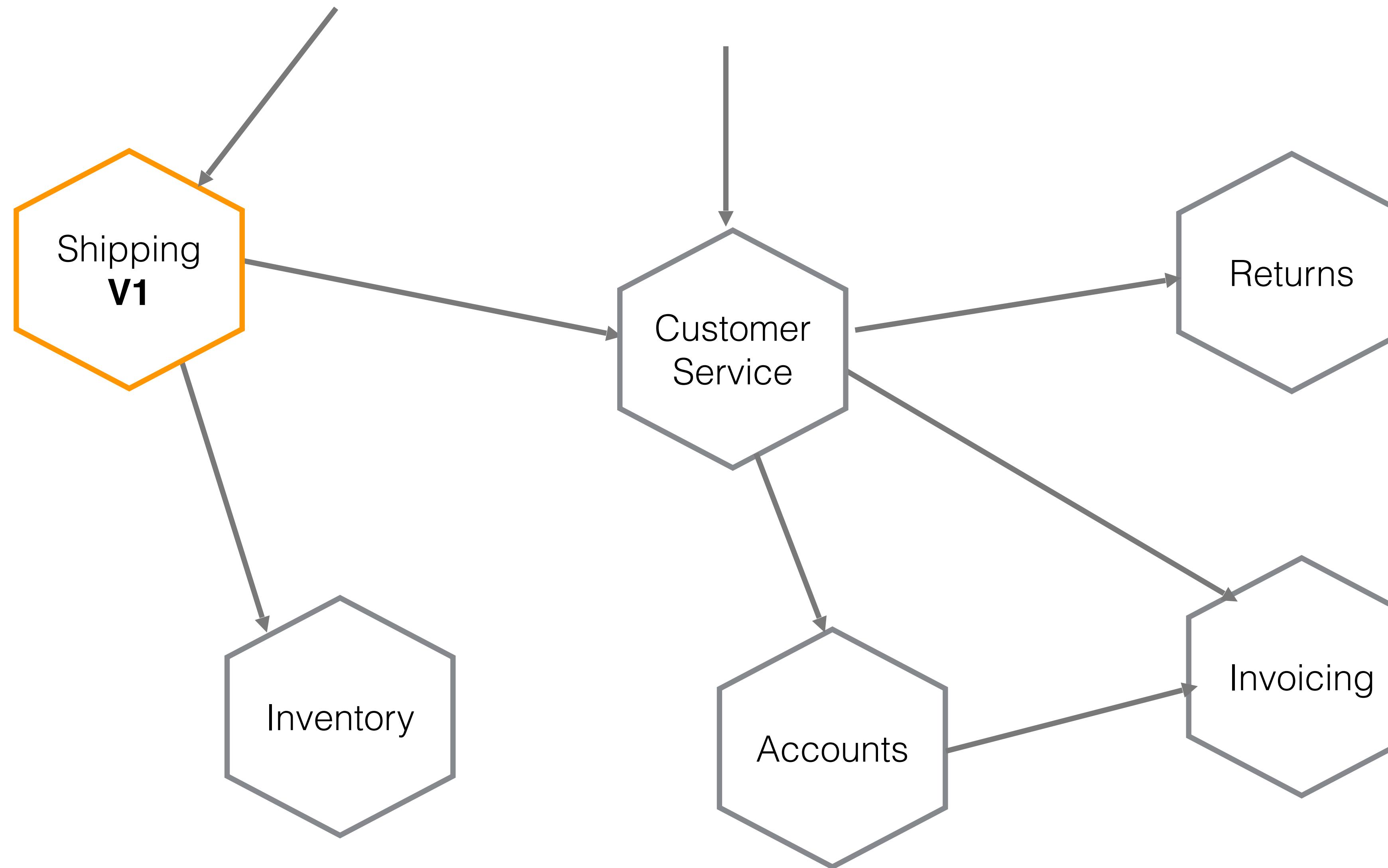
Just started using them

1-2 years experience

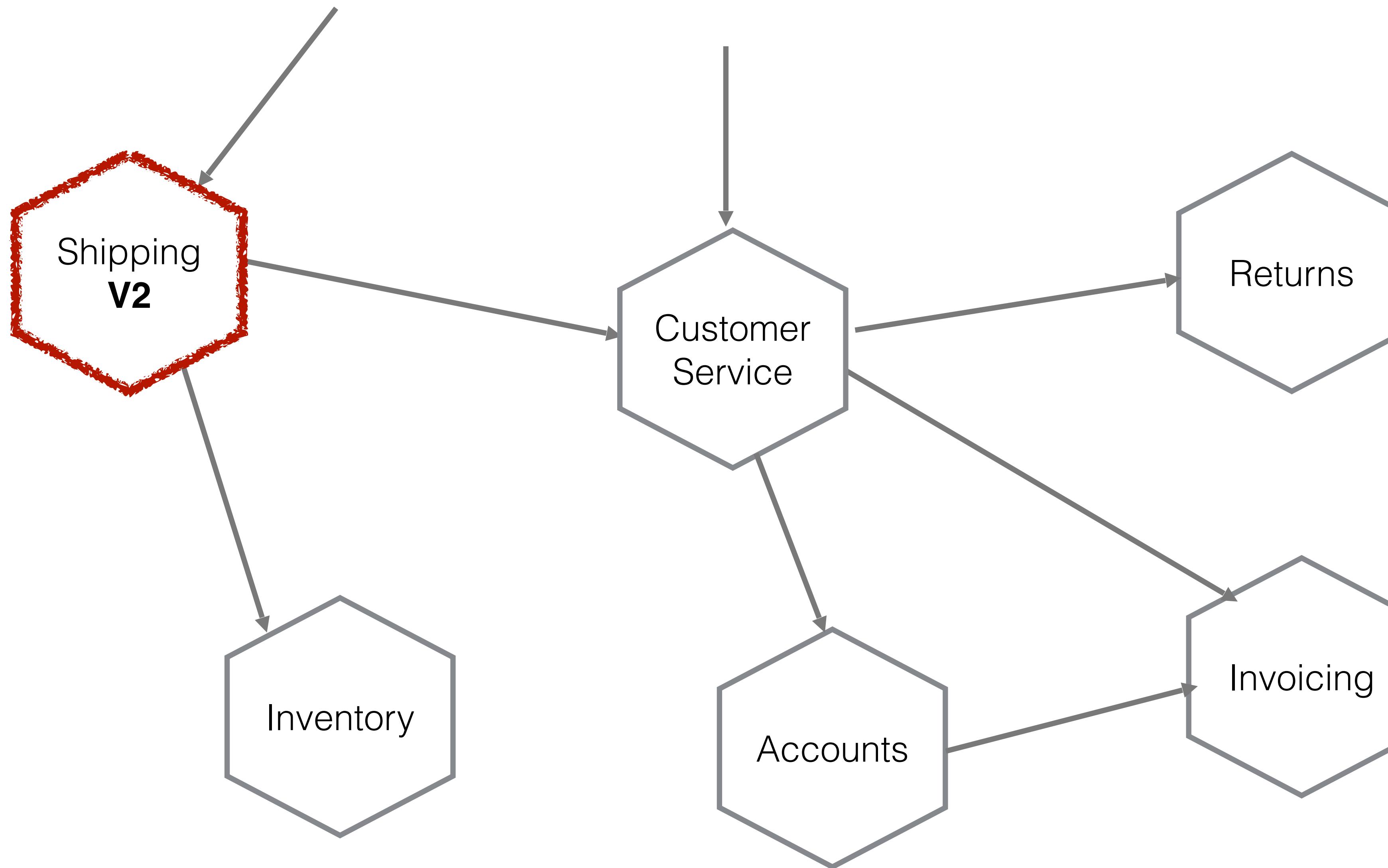
2+ years experience

What are microservices?

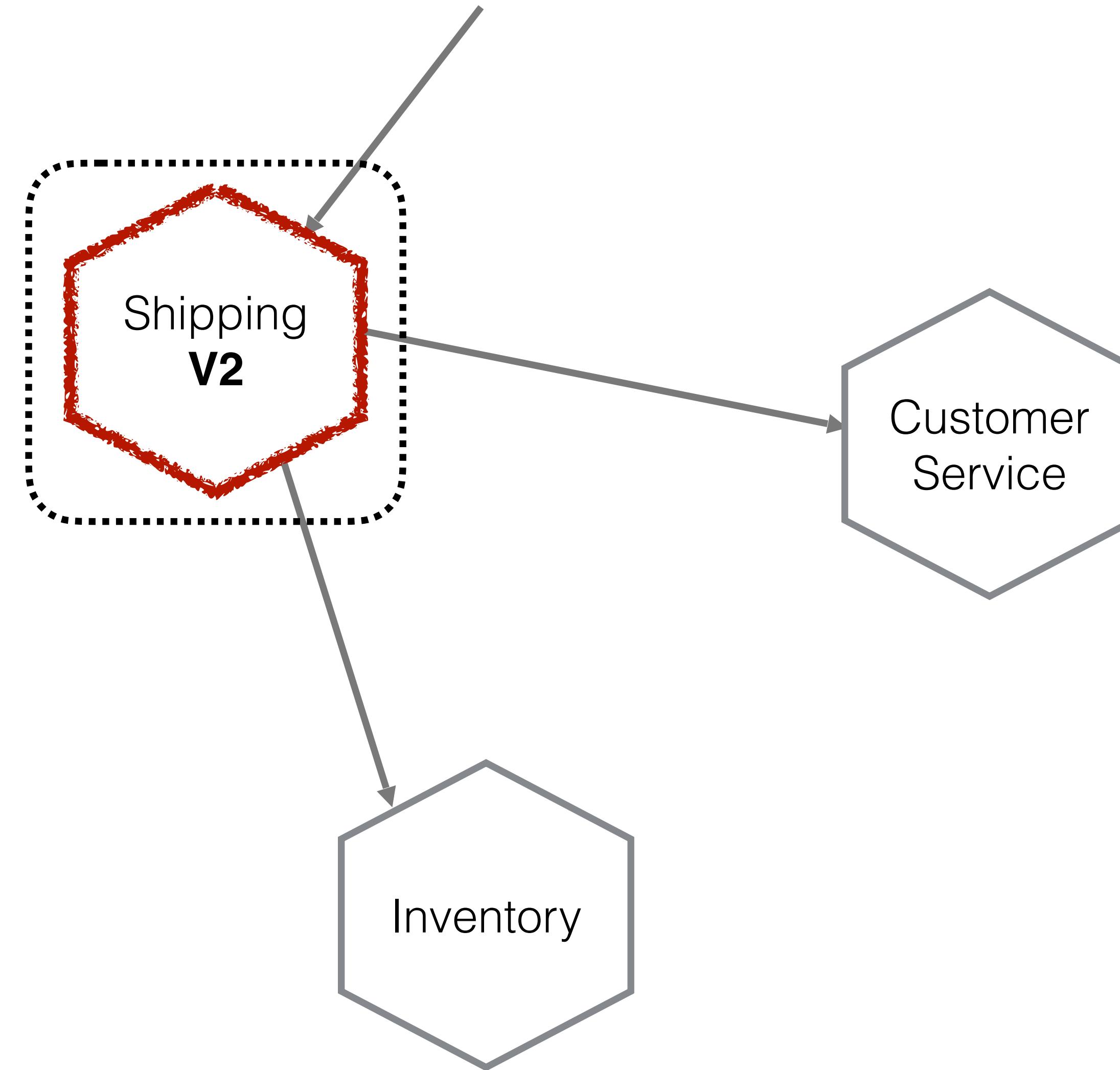
INDEPENDENT DEPLOYABILITY



INDEPENDENT DEPLOYABILITY

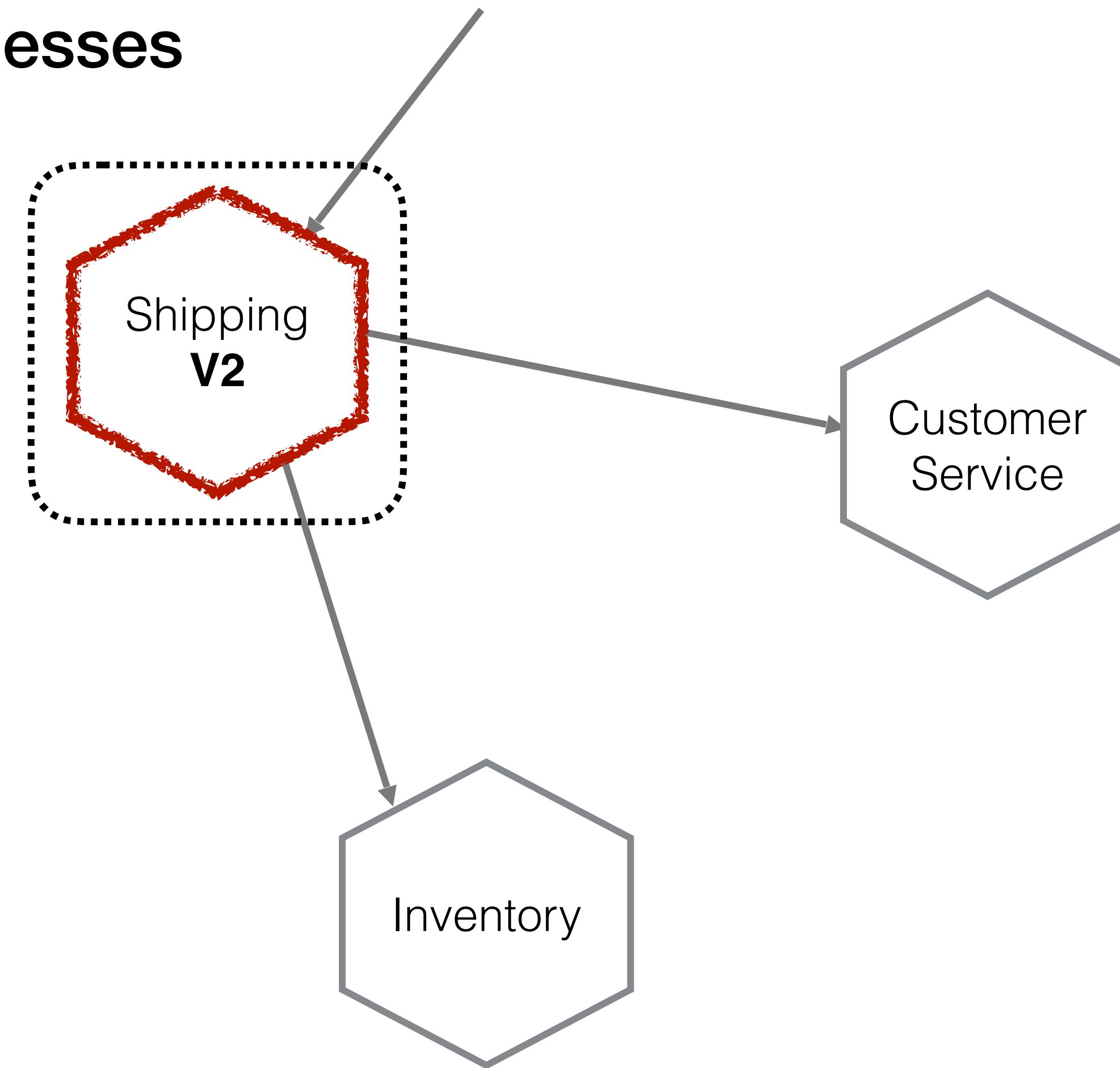


ANATOMY OF A MICROSERVICE



ANATOMY OF A MICROSERVICE

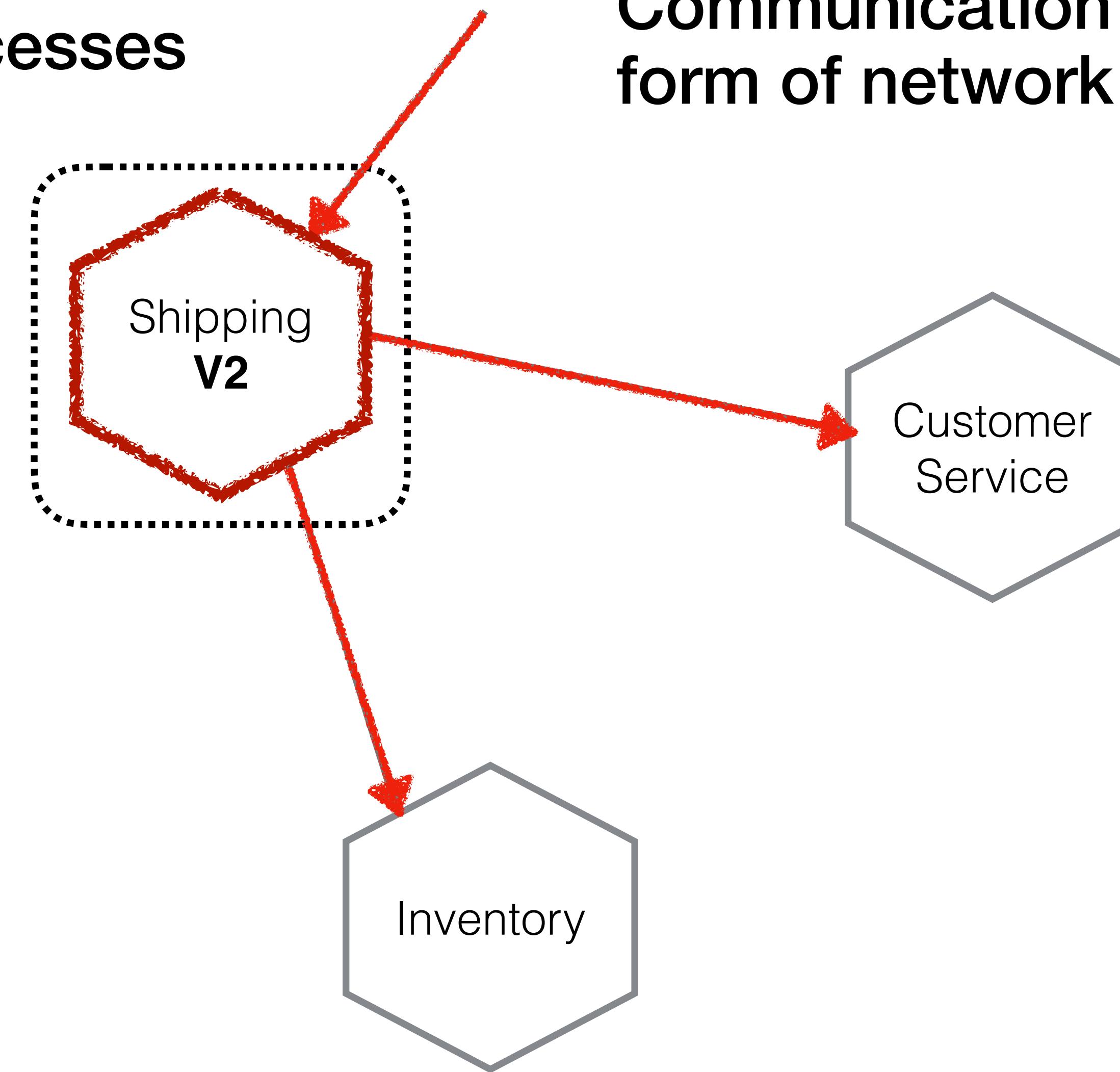
Microservice instances are typically separate processes



ANATOMY OF A MICROSERVICE

Microservice instances are typically separate processes

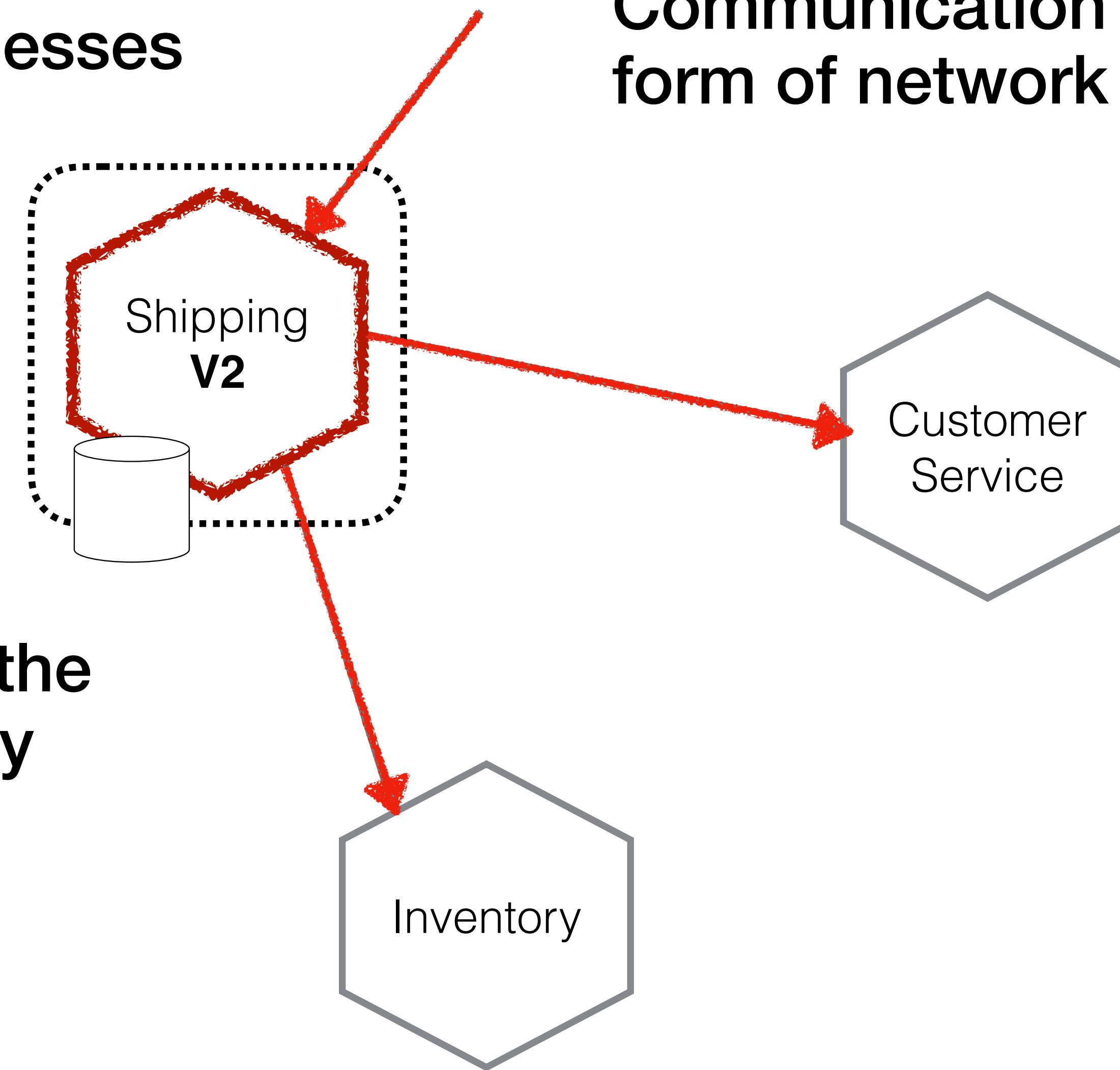
Communication is via some form of network call



ANATOMY OF A MICROSERVICE

Microservice instances are typically separate processes

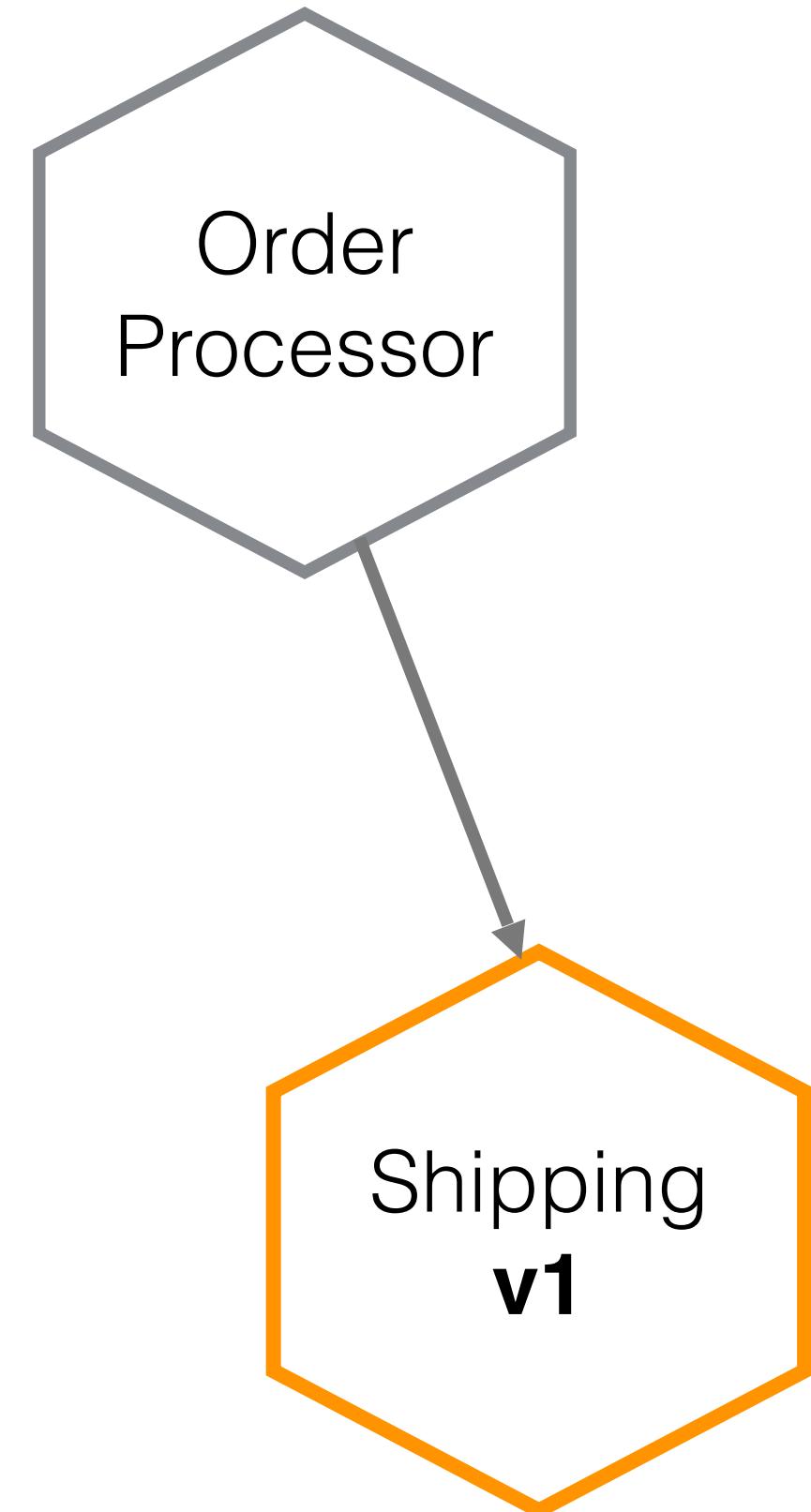
Communication is via some form of network call



Data is hidden inside the microservice boundary

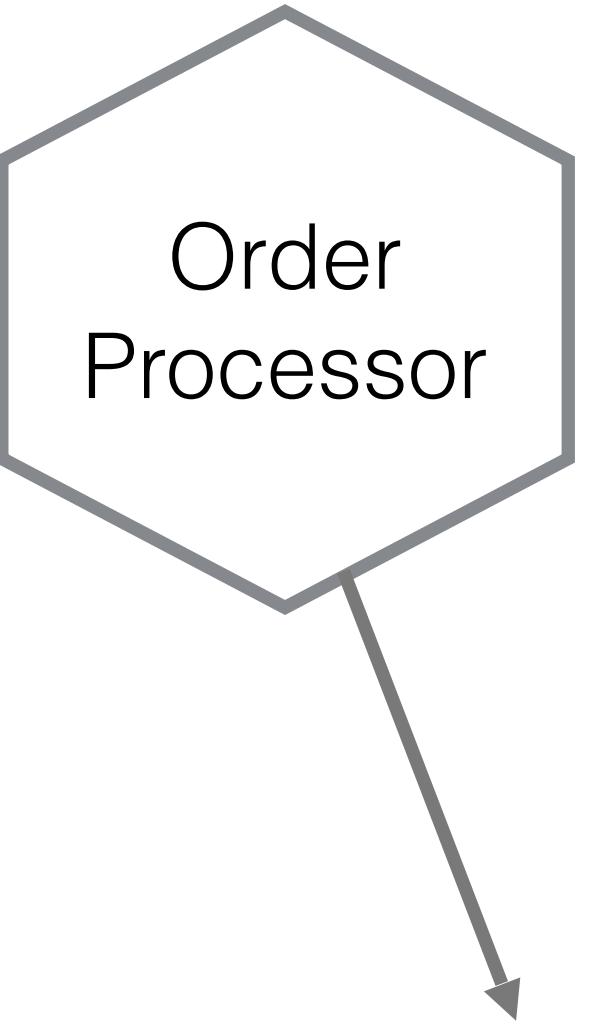
Why separate database?

**Independent deployability
requires interface stability**



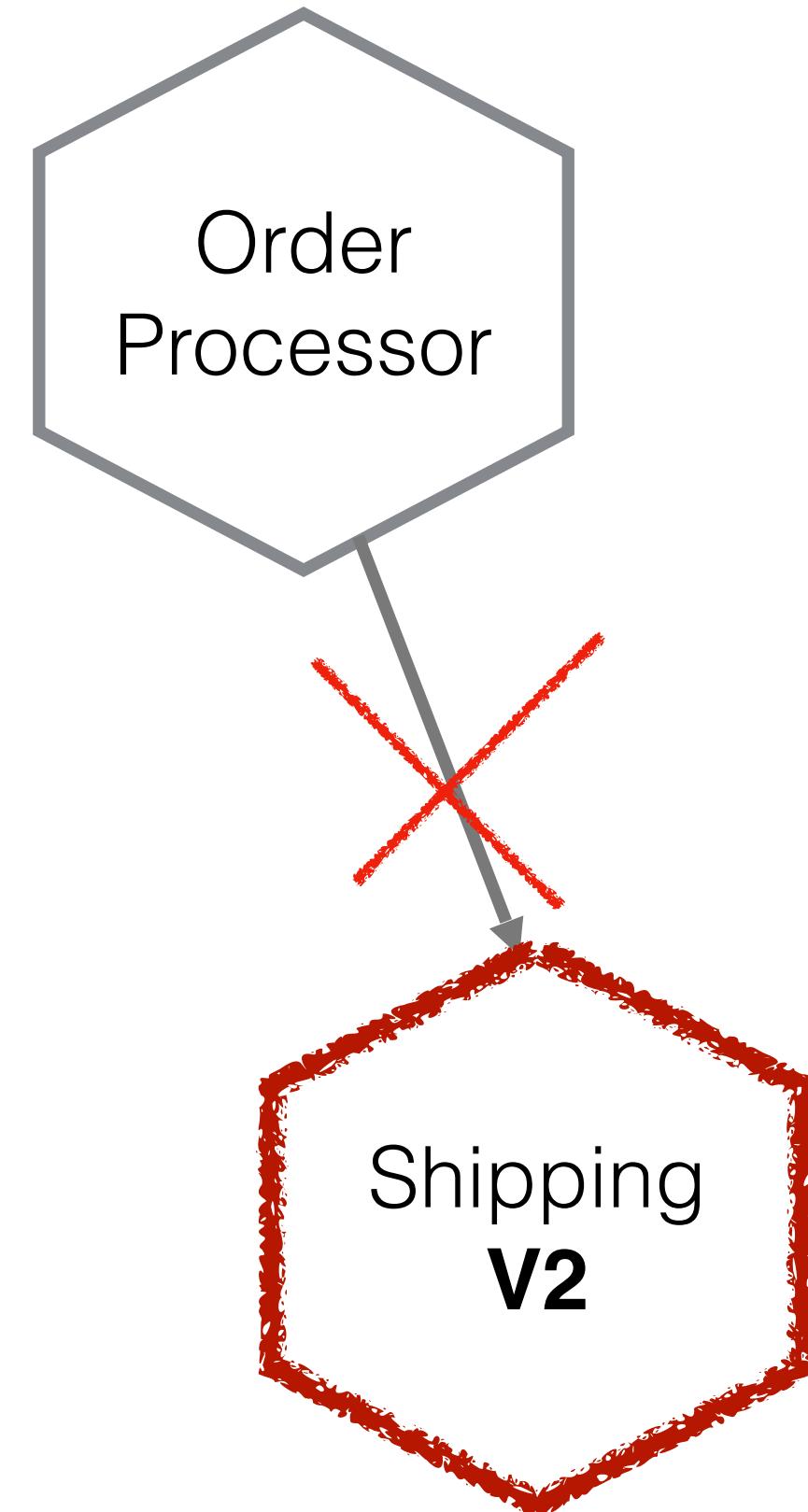
**Maintaining backwards
compatibility is key**

**Independent deployability
requires interface stability**



**Maintaining backwards
compatibility is key**

**Independent deployability
requires interface stability**



**Maintaining backwards
compatibility is key**

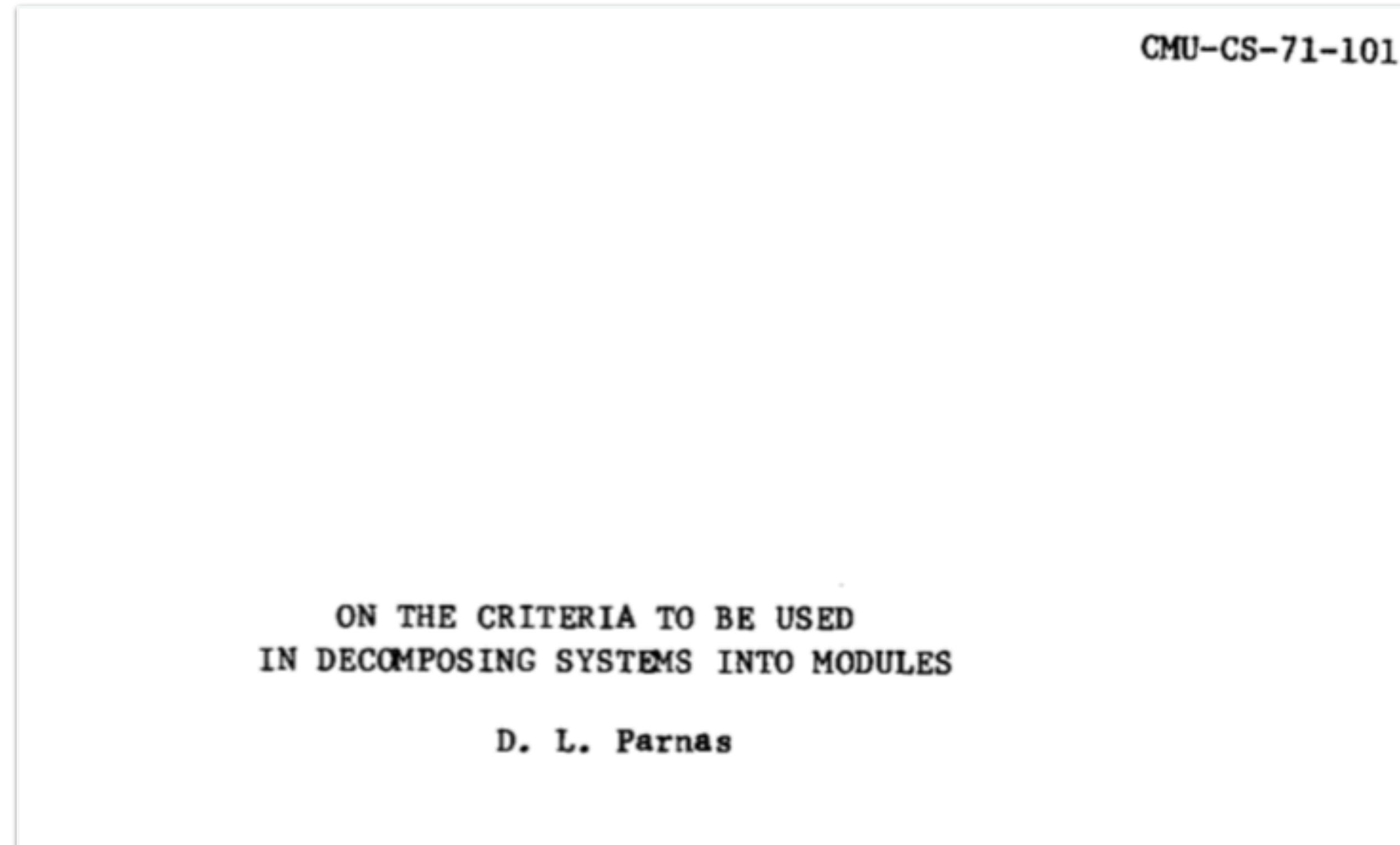
POLL: HOW MANY OF YOU HAVE ALL YOUR DATA IN A SINGLE DATABASE?

Yes

No

Not Sure!

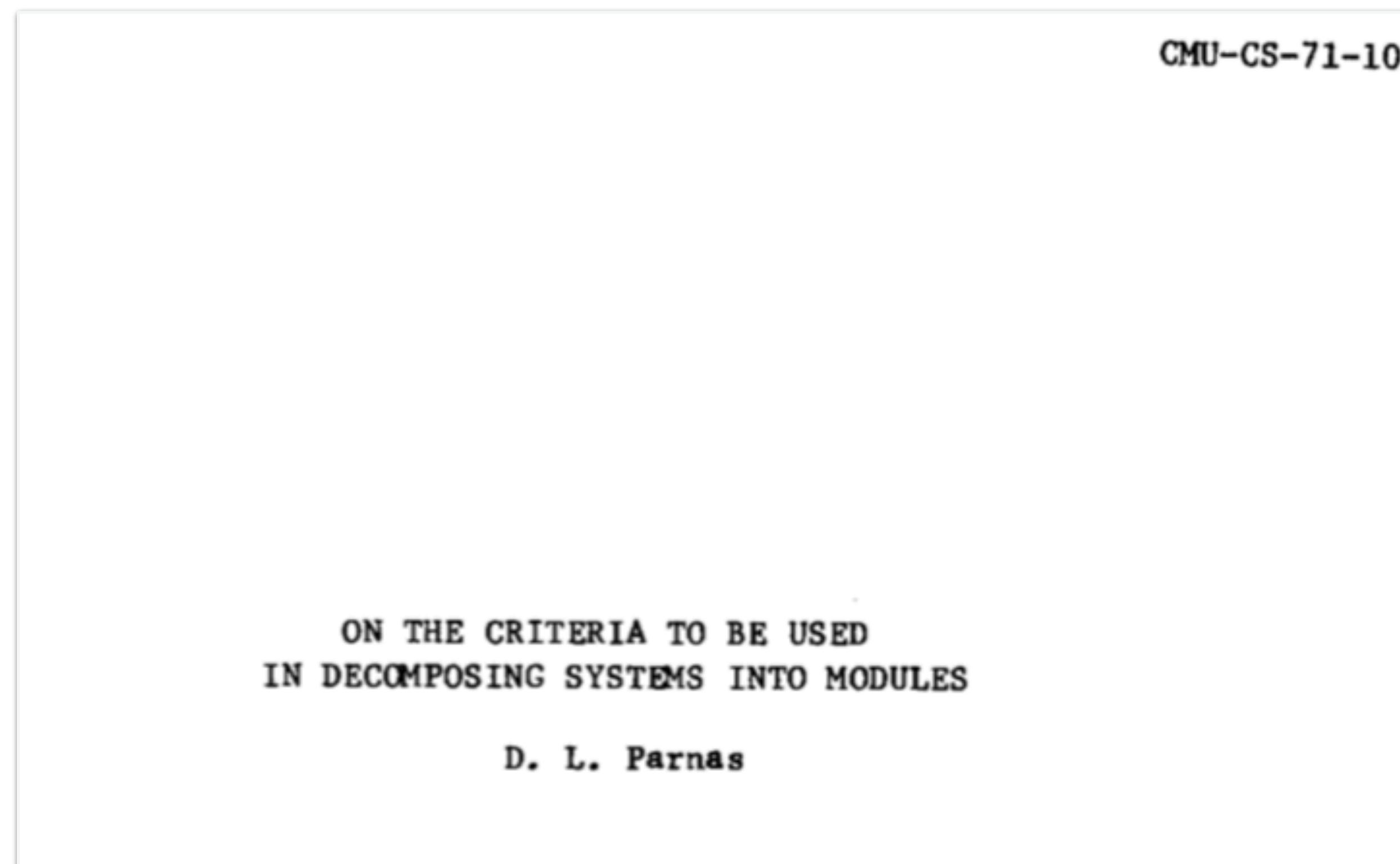
INFORMATION HIDING



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

@samnewman

INFORMATION HIDING

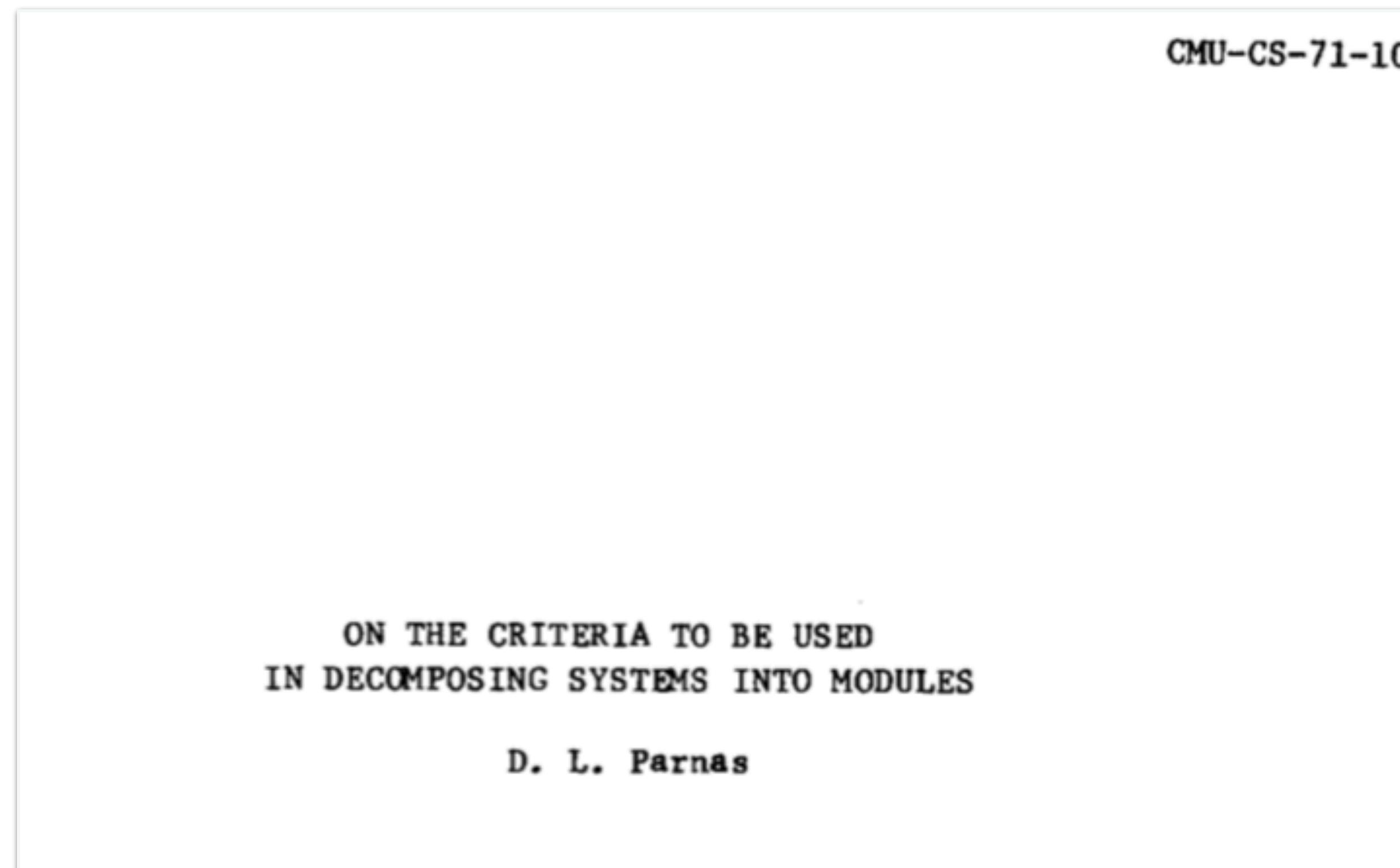


Published in 1971

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

@samnewman

INFORMATION HIDING



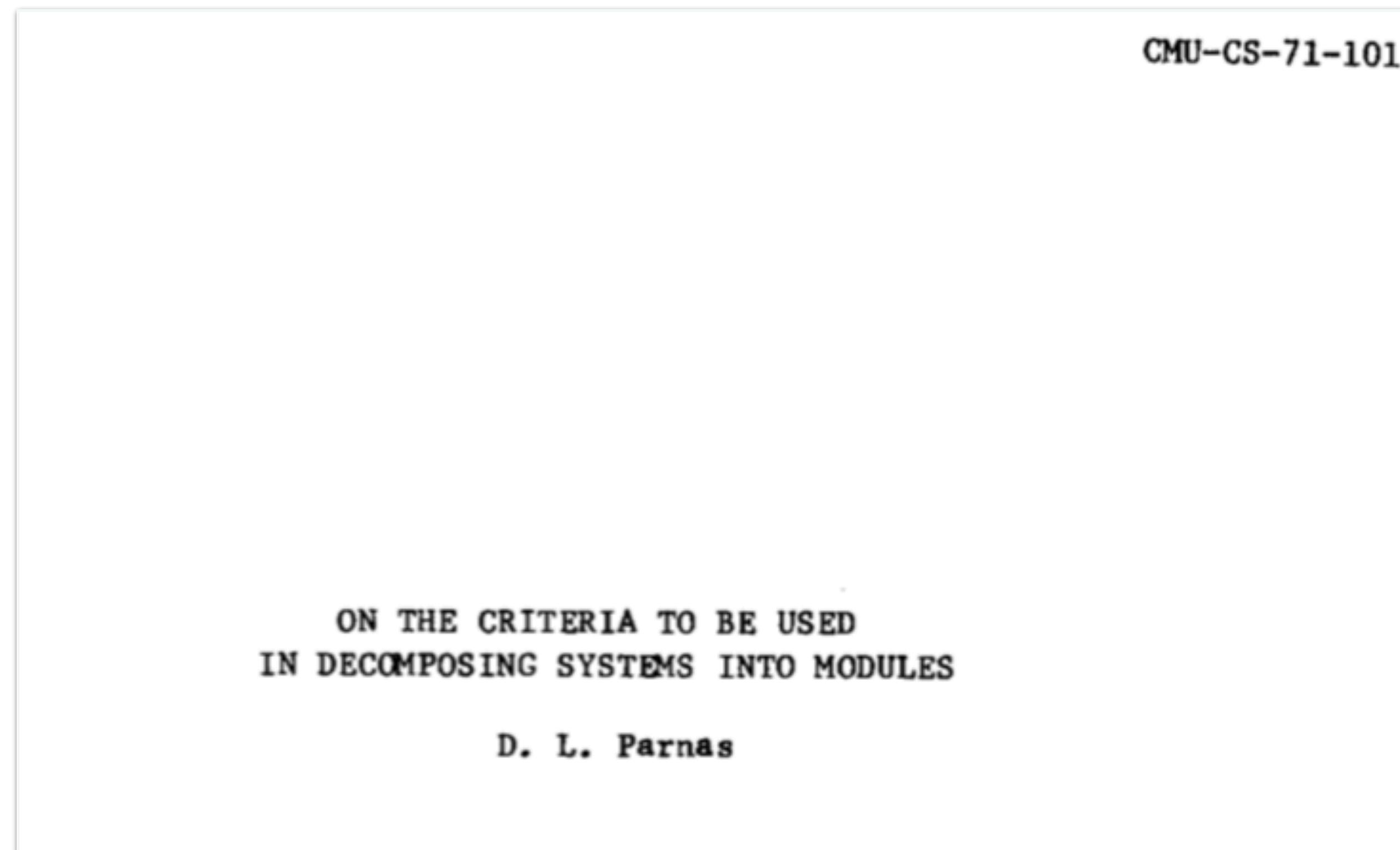
Published in 1971

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

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

@samnewman

INFORMATION HIDING AND MICROSERVICES

the morning paper
a random walk through Computer Science research, by Adrian Colyer

[ABOUT](#) [ARCHIVES](#) [INFOQ QR EDITIONS](#) [SEARCH](#) [SUBSCRIBE](#) [TAGS](#) [PRIVACY](#)

On the criteria to be used in decomposing systems into modules

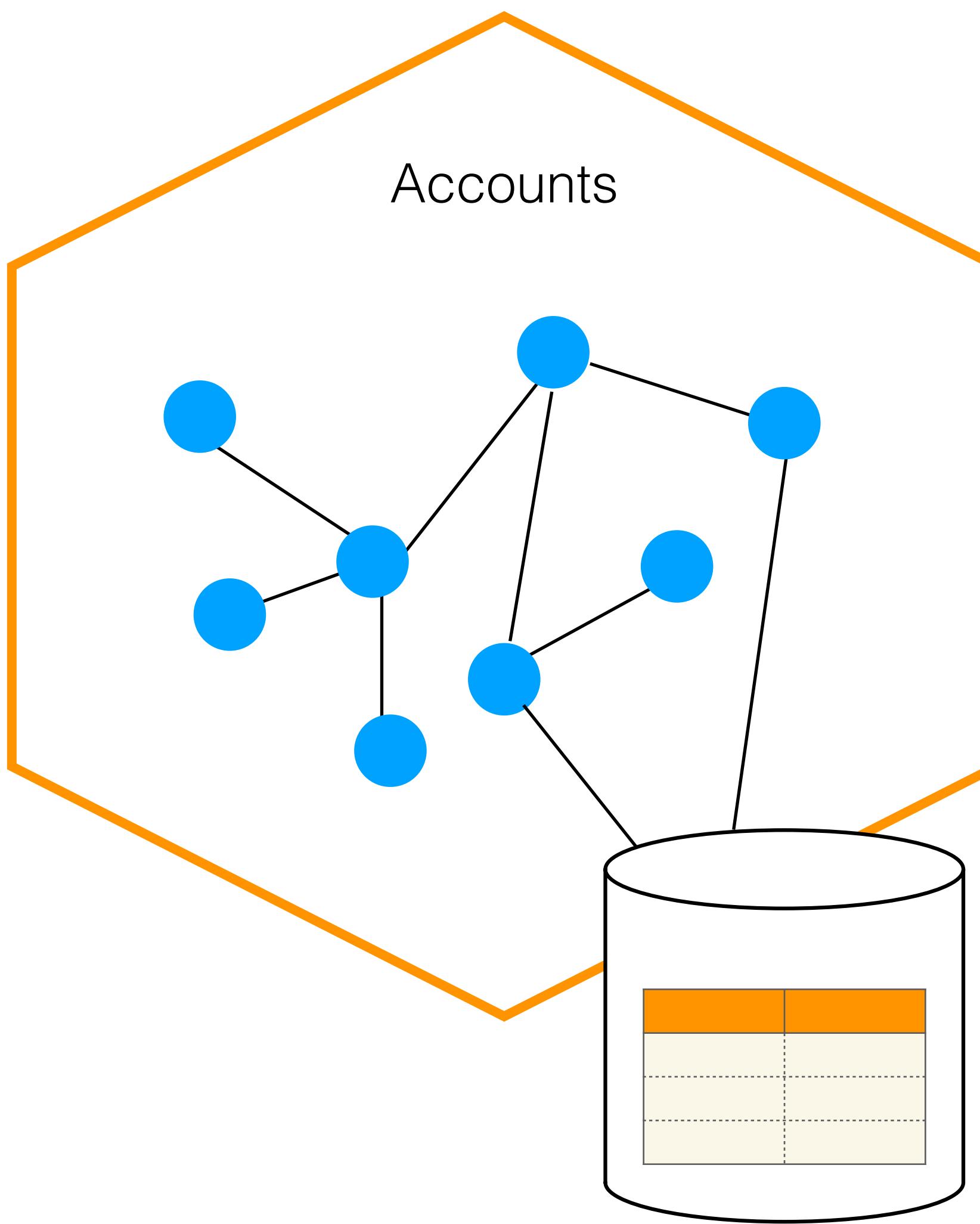
SEPTEMBER 5, 2016

[On the criteria to be used in decomposing systems into modules](#) *David L Parnas, 1971*

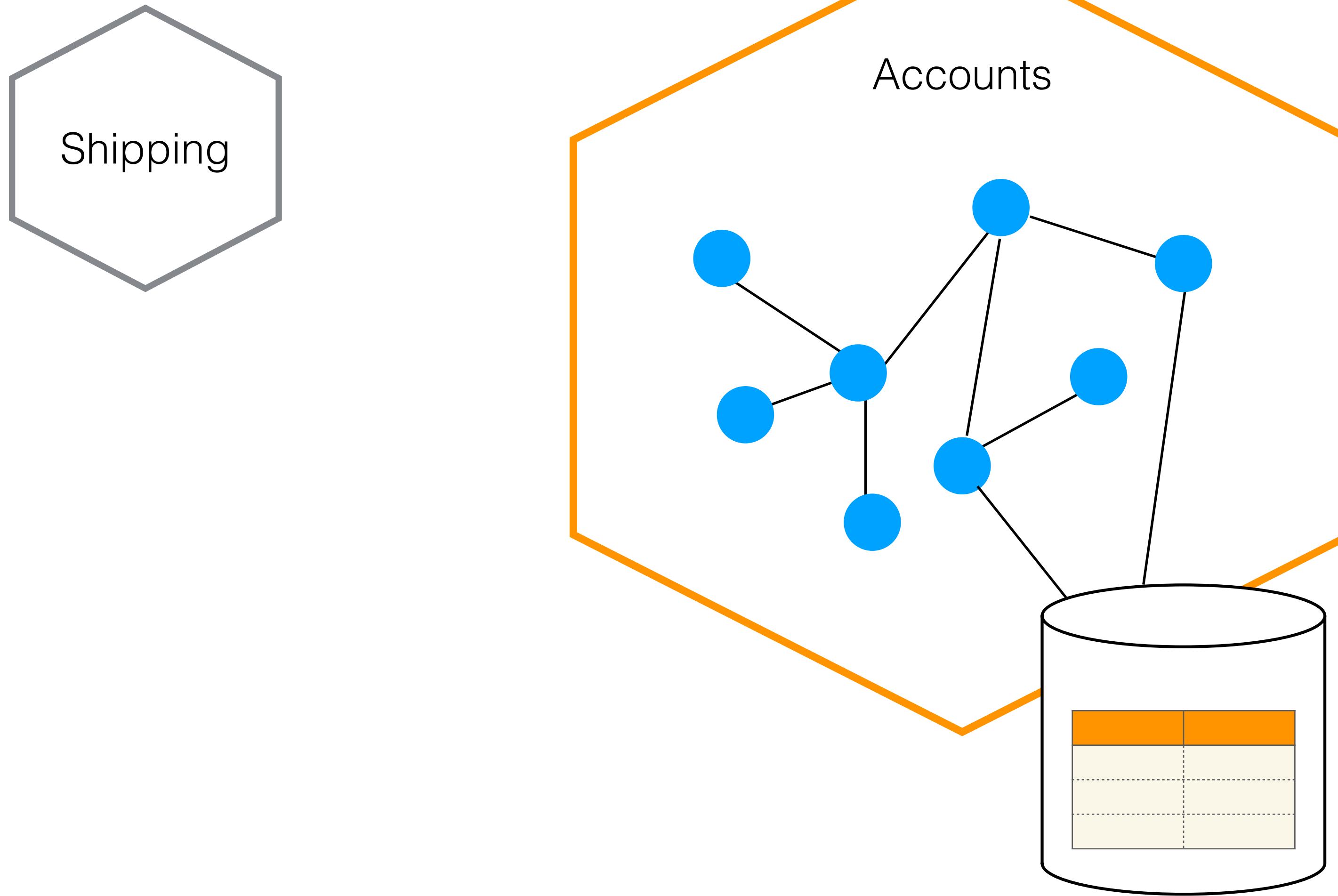
Welcome back to a new term of The Morning Paper! I thought I'd kick things off by revisiting a few of my favourite papers from when I very first started this exercise just over two years ago. At that time I wasn't posting blog summaries of the papers, so it's nice to go back and fill in that gap (blog posts started in October of 2014). Plus, revisiting some of the classics once every couple of years seems like a good idea – changing external circumstances can make them feel fresh again every time you read them.

<https://blog.acolyer.org/2016/09/05/on-the-criteria-to-be-used-in-decomposing-systems-into-modules/>

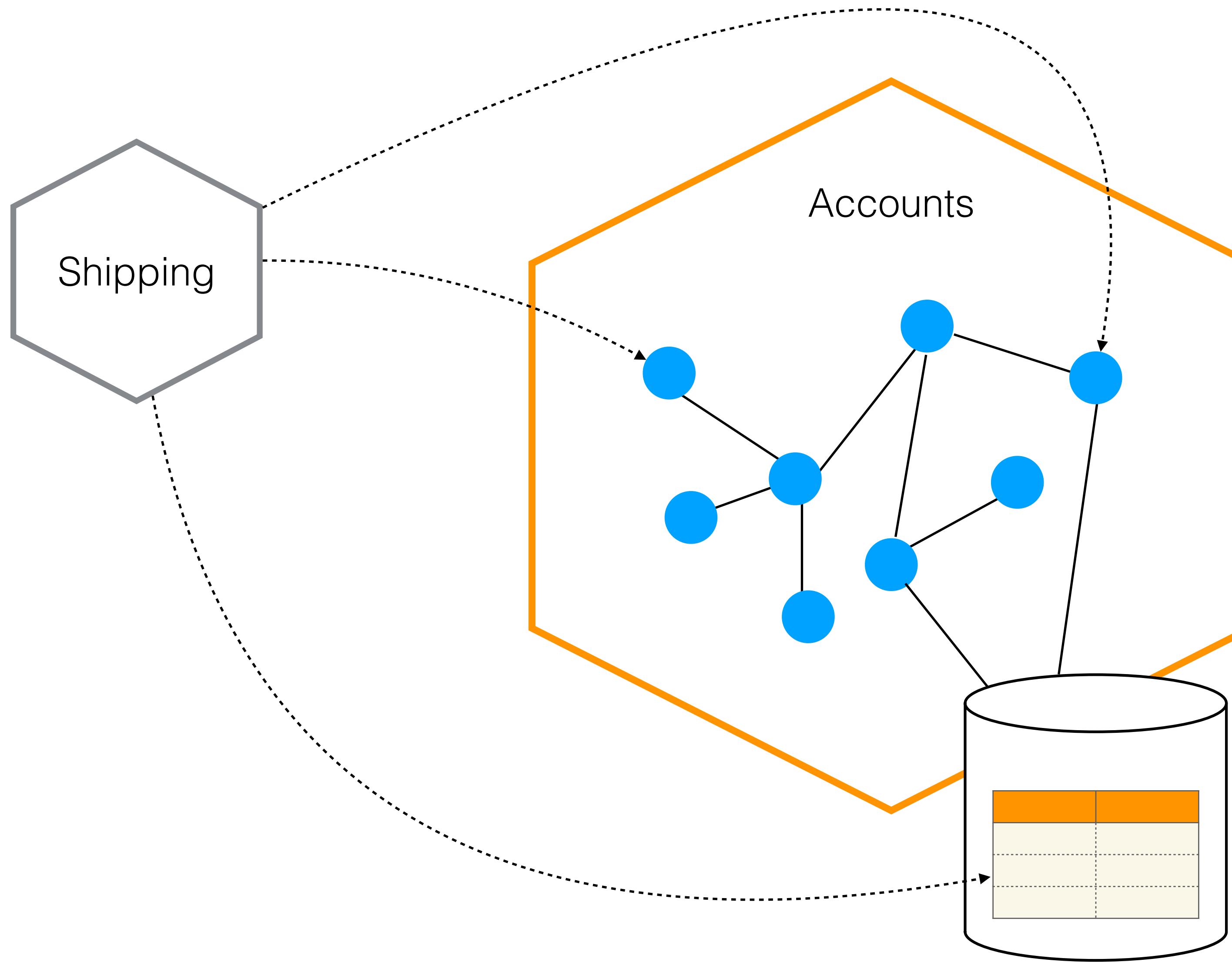
NOT HIDING?



NOT HIDING?

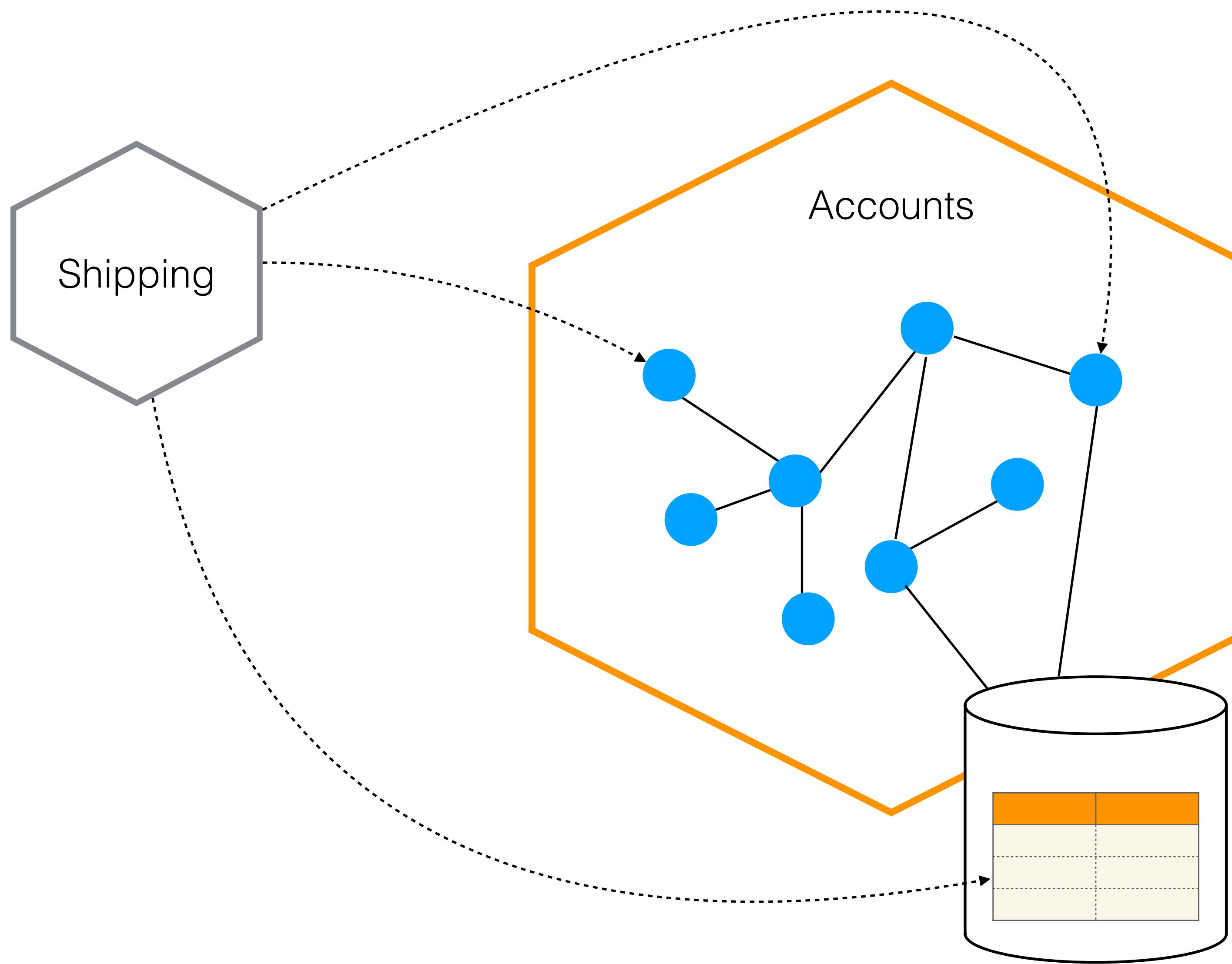


NOT HIDING?



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

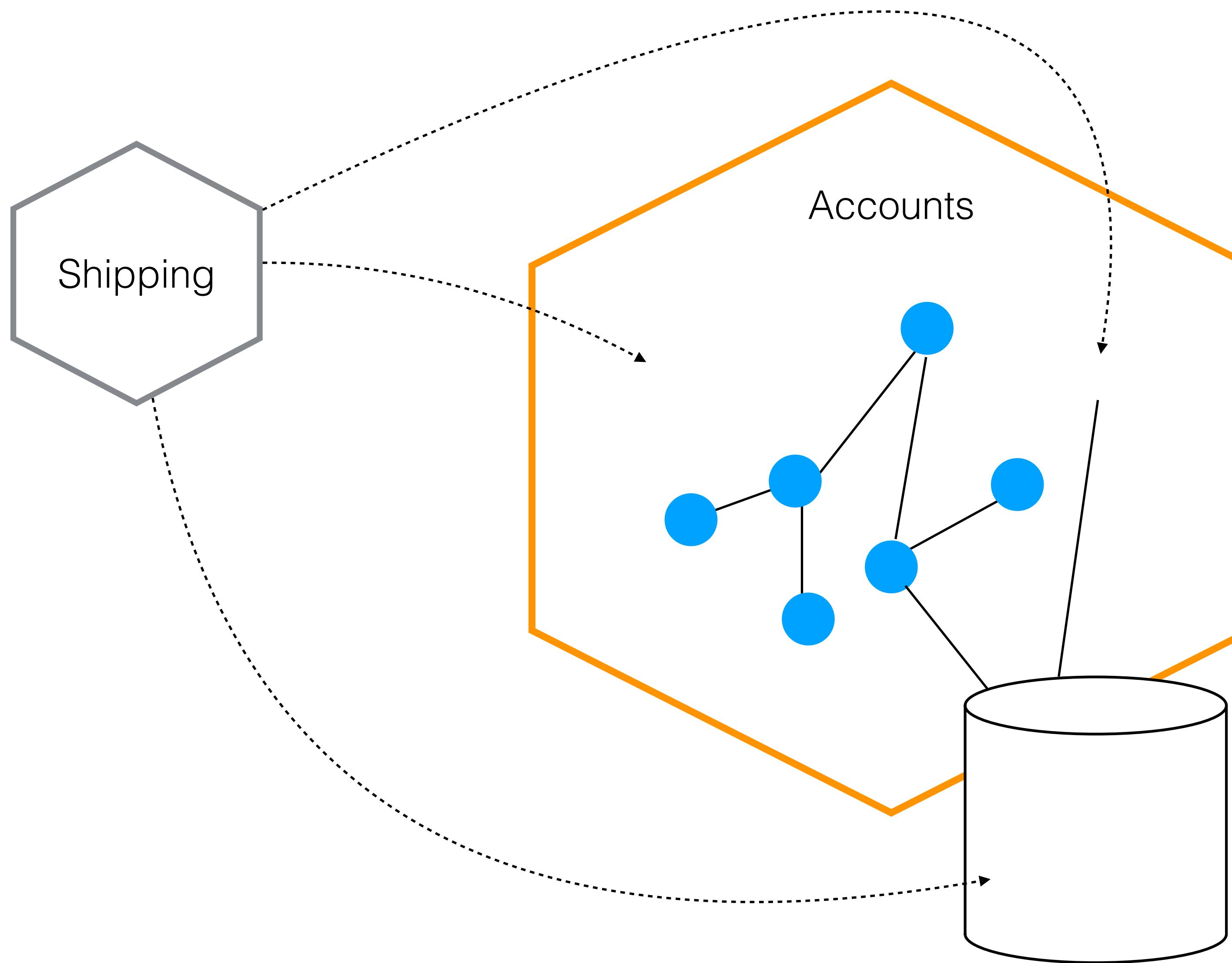
NOT HIDING?



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

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

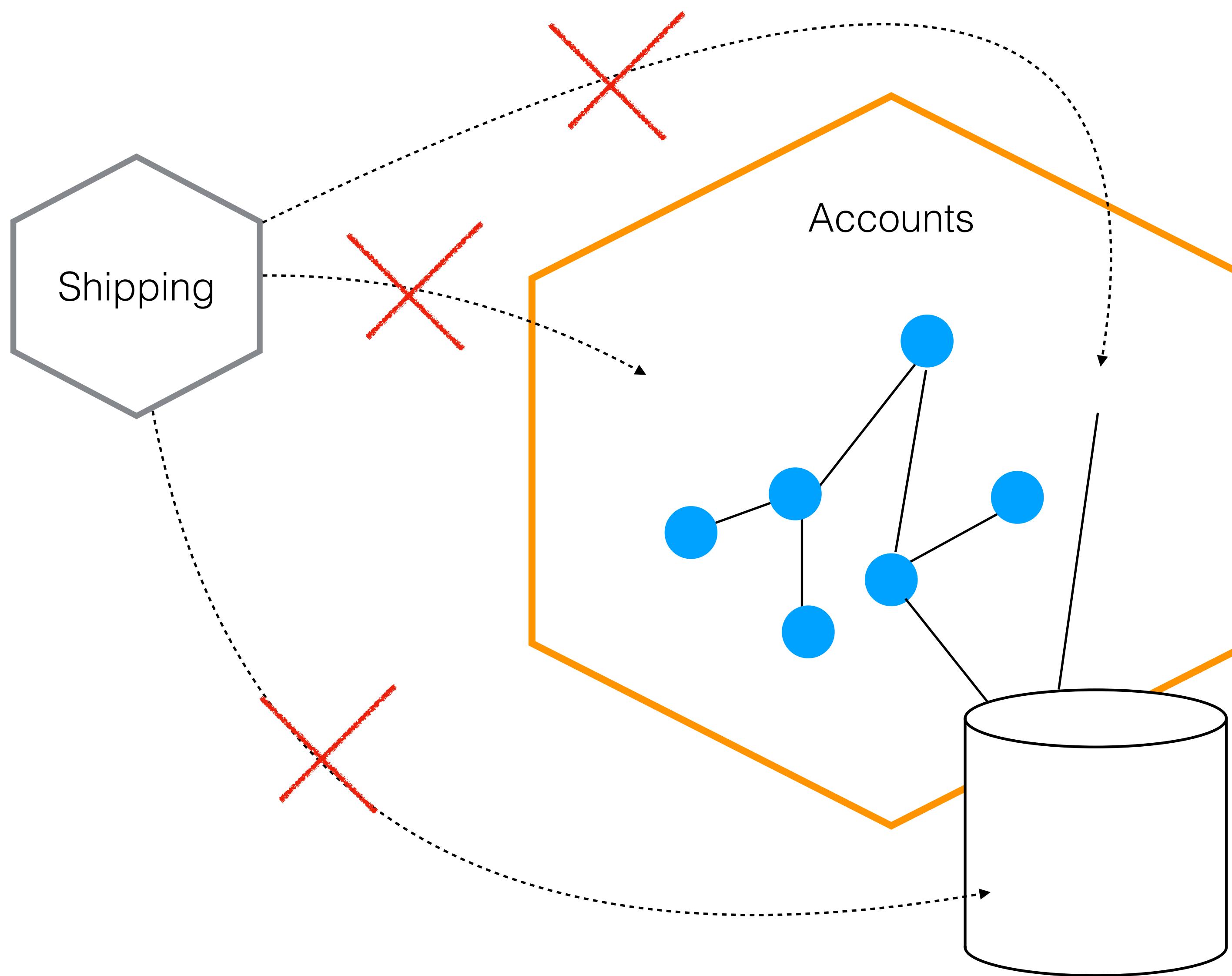
NOT HIDING?



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

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

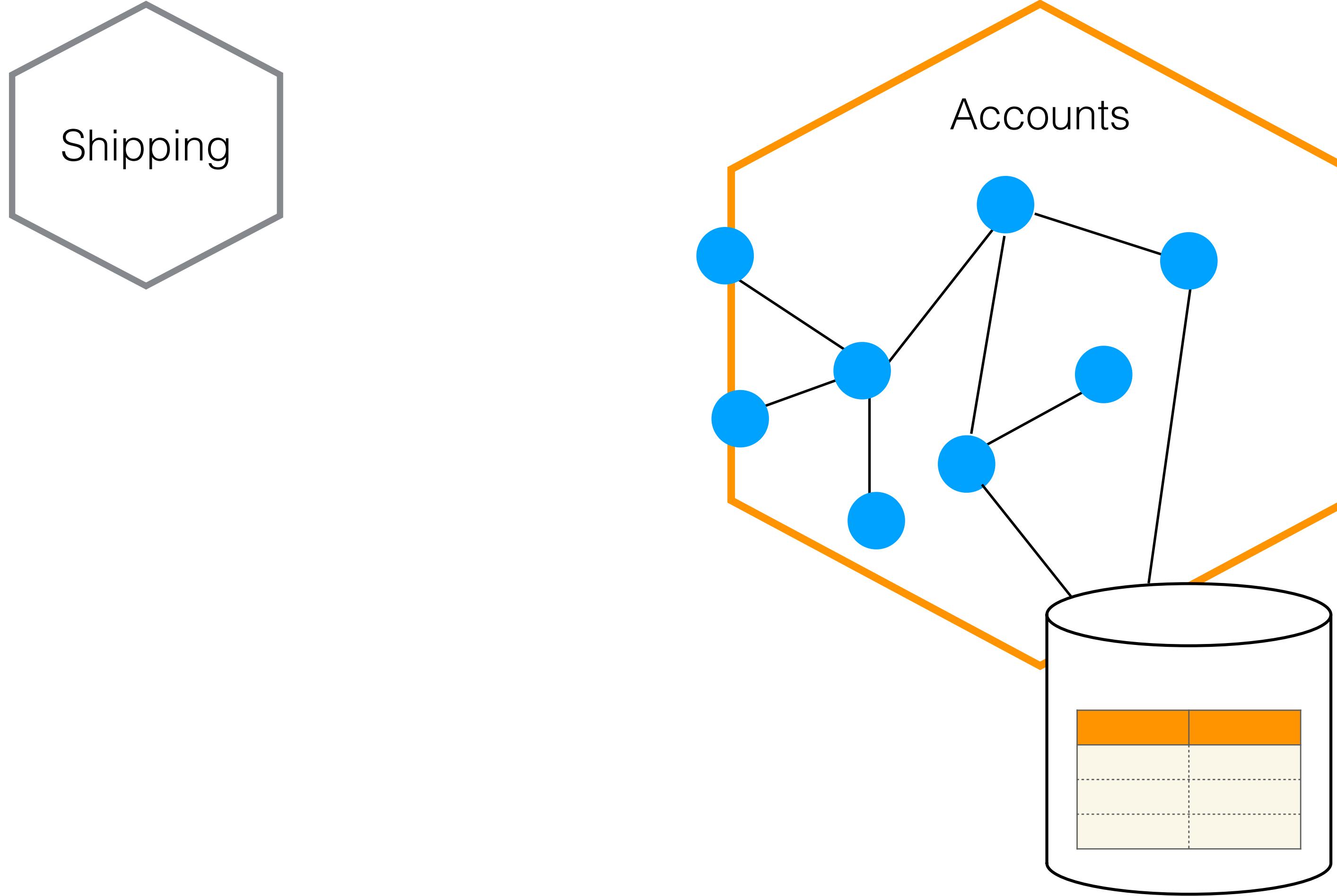
NOT HIDING?



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

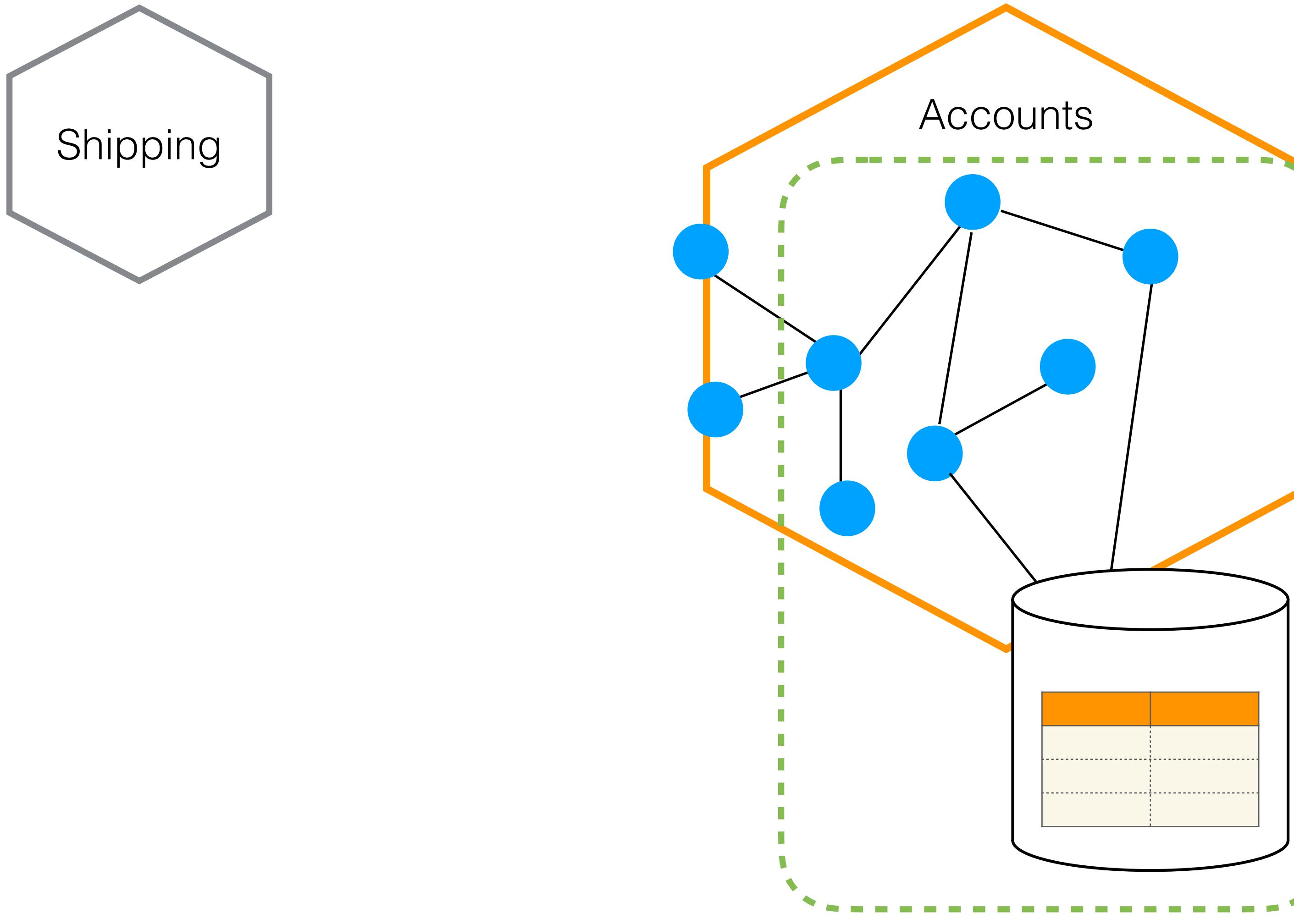
...then you can't change
this implementation without
breaking the consumer

HIDING!



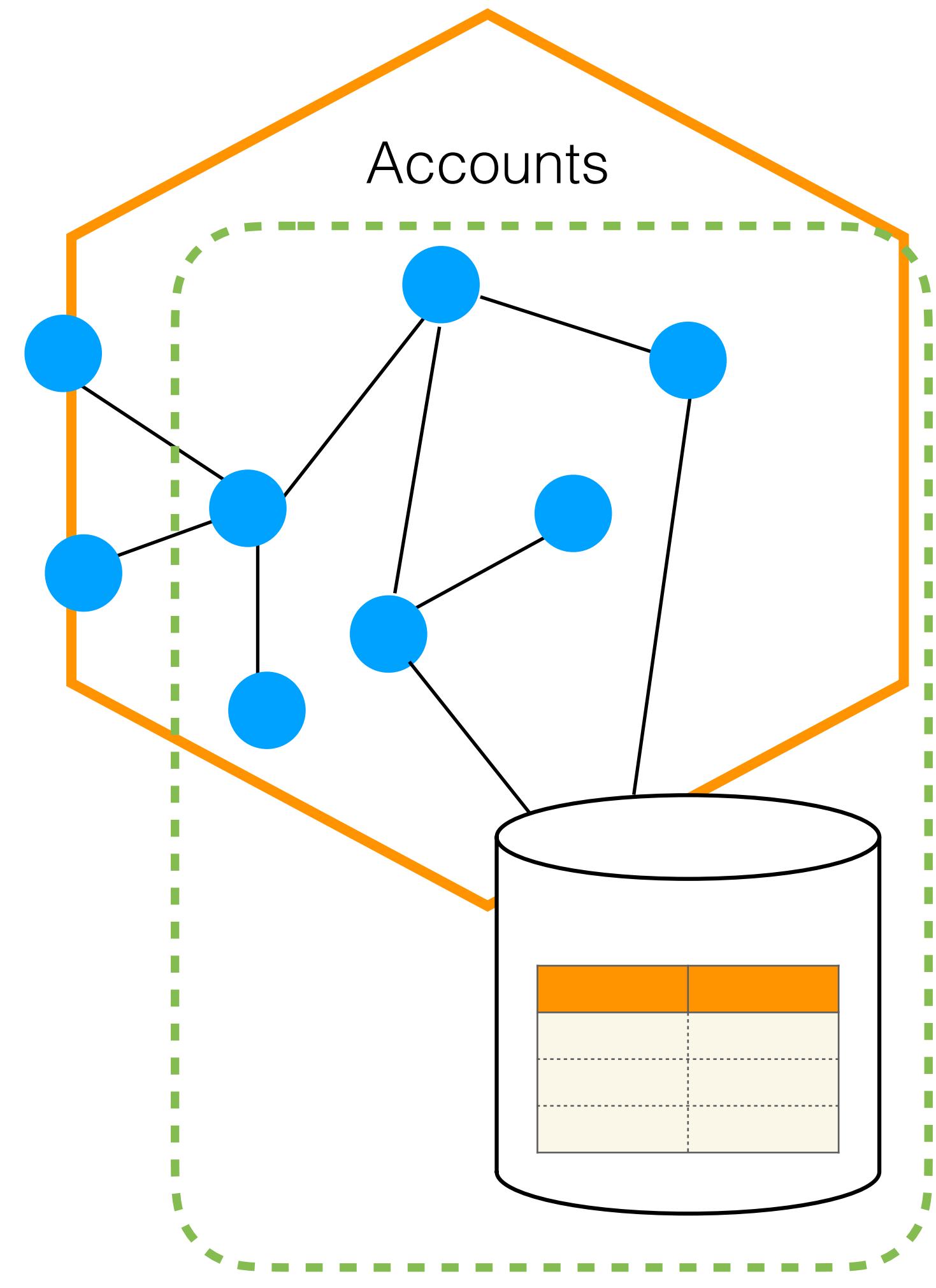
HIDING!

Hide your secrets!



Hidden

HIDING!

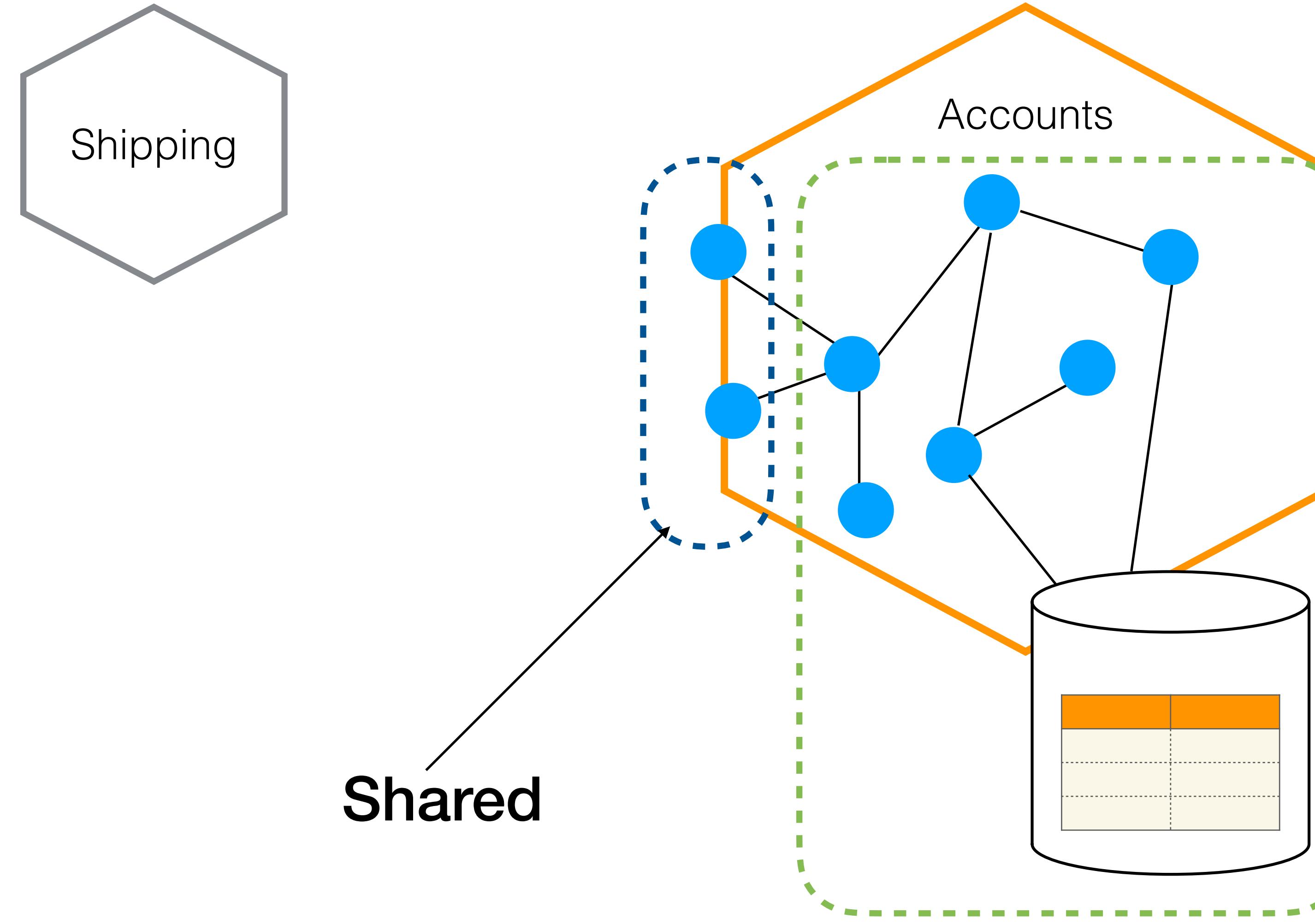


Hide your secrets!

Be explicit about what is shared, and what is hidden

Hidden

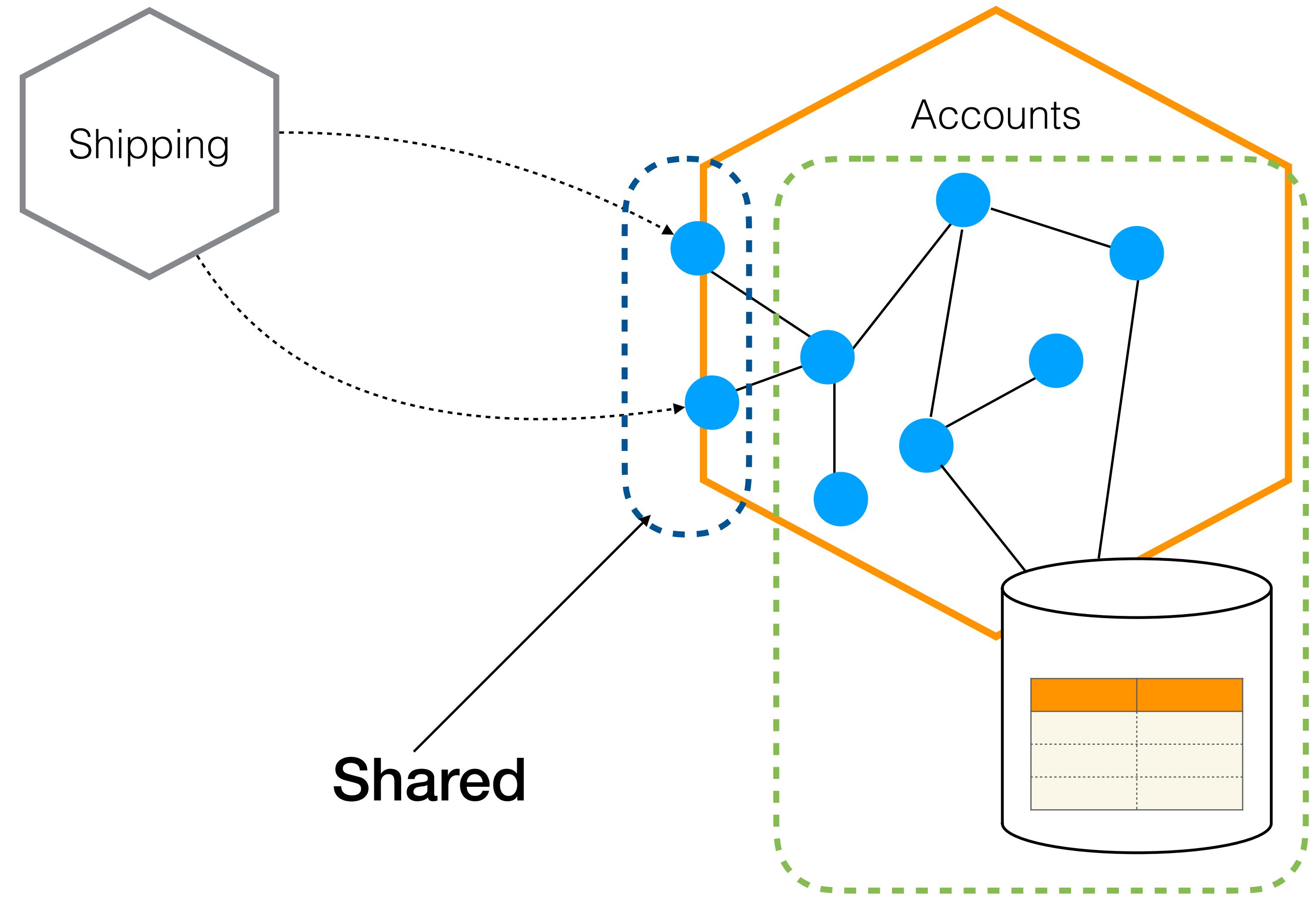
HIDING!



Hide your secrets!

Be explicit about what is shared, and what is hidden

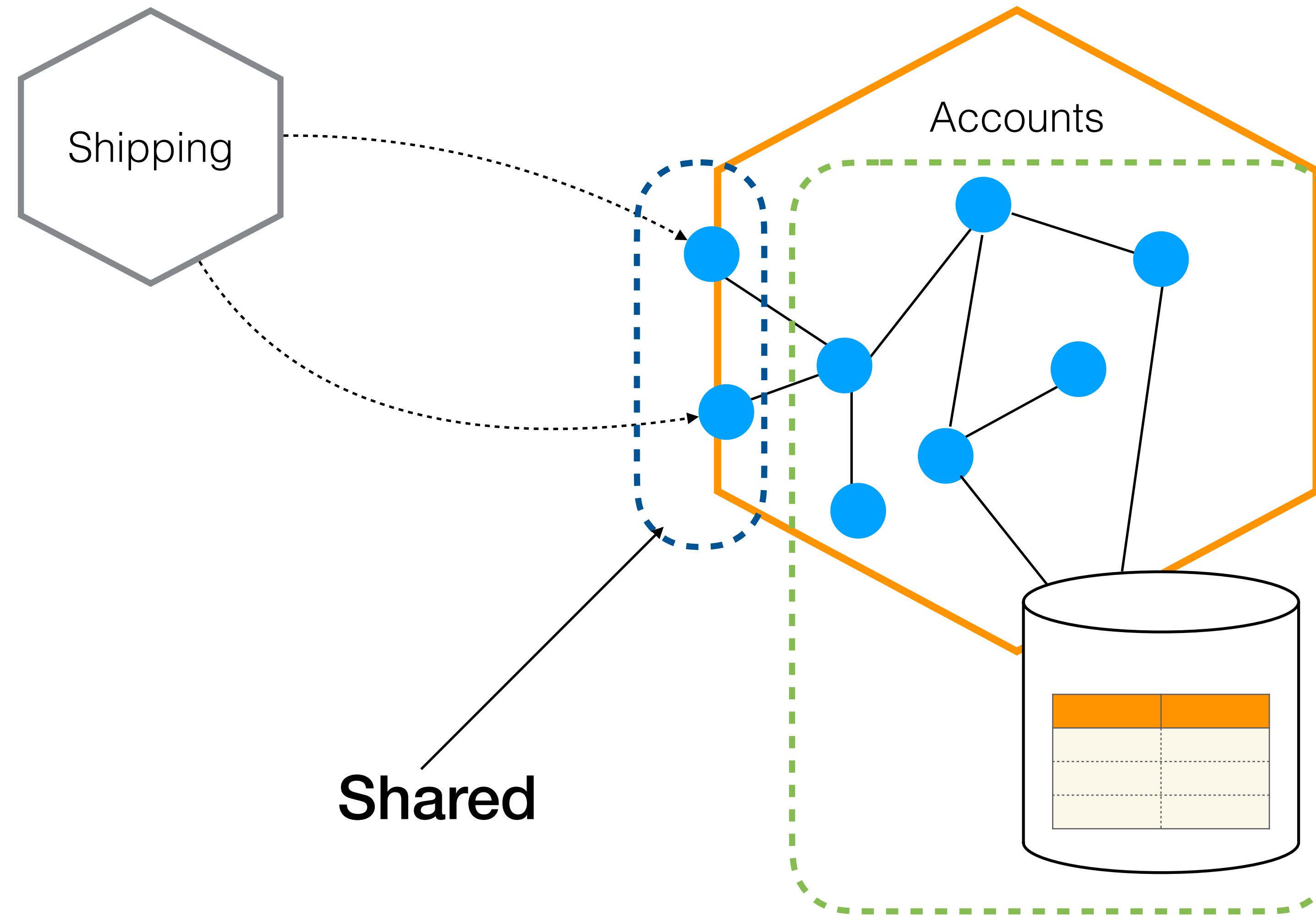
HIDING!



Hide your secrets!

Be explicit about what is shared, and what is hidden

HIDING!



Hide your secrets!

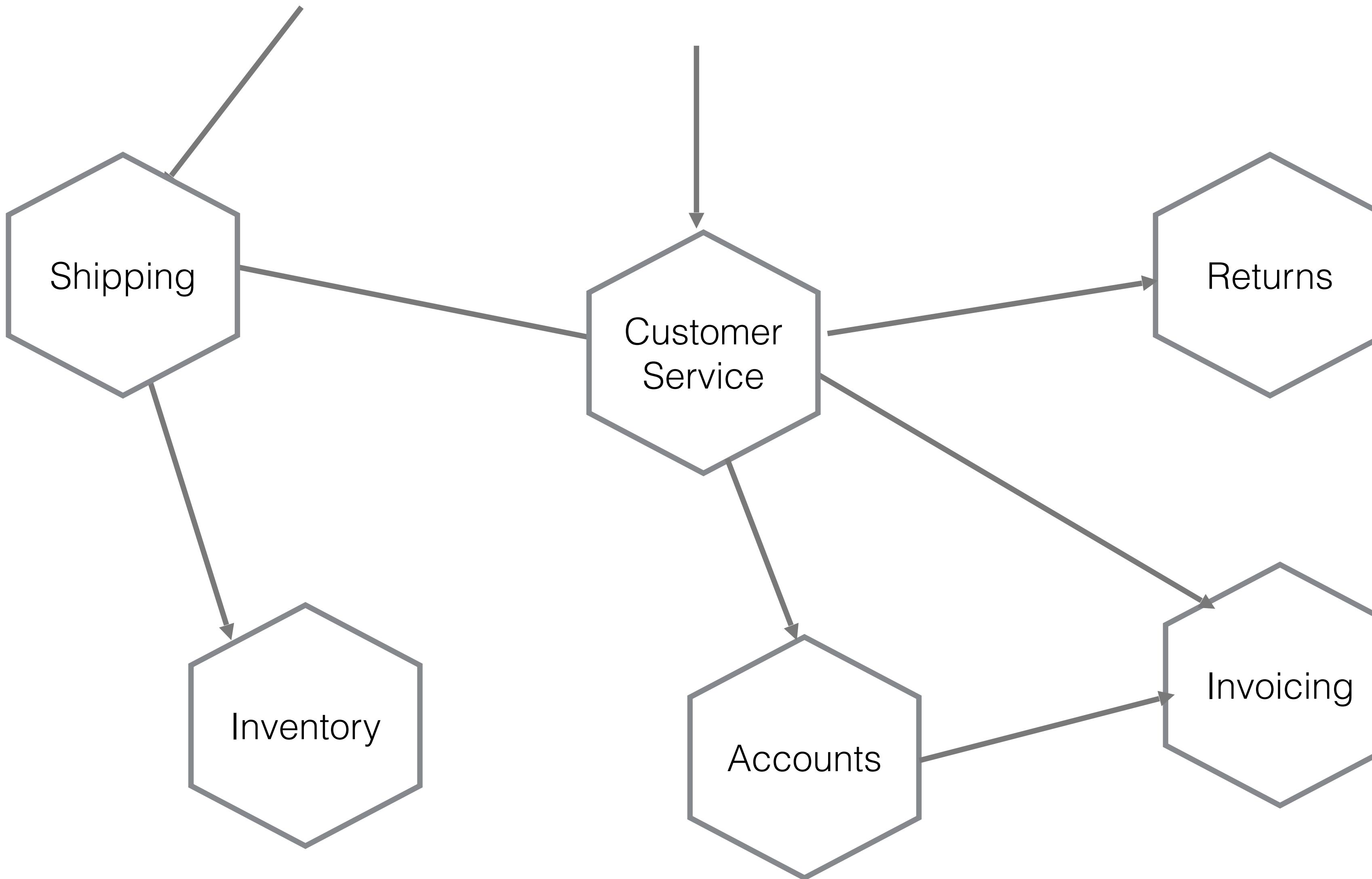
Be explicit about what is shared, and what is hidden

Hidden things can change, shared things can't

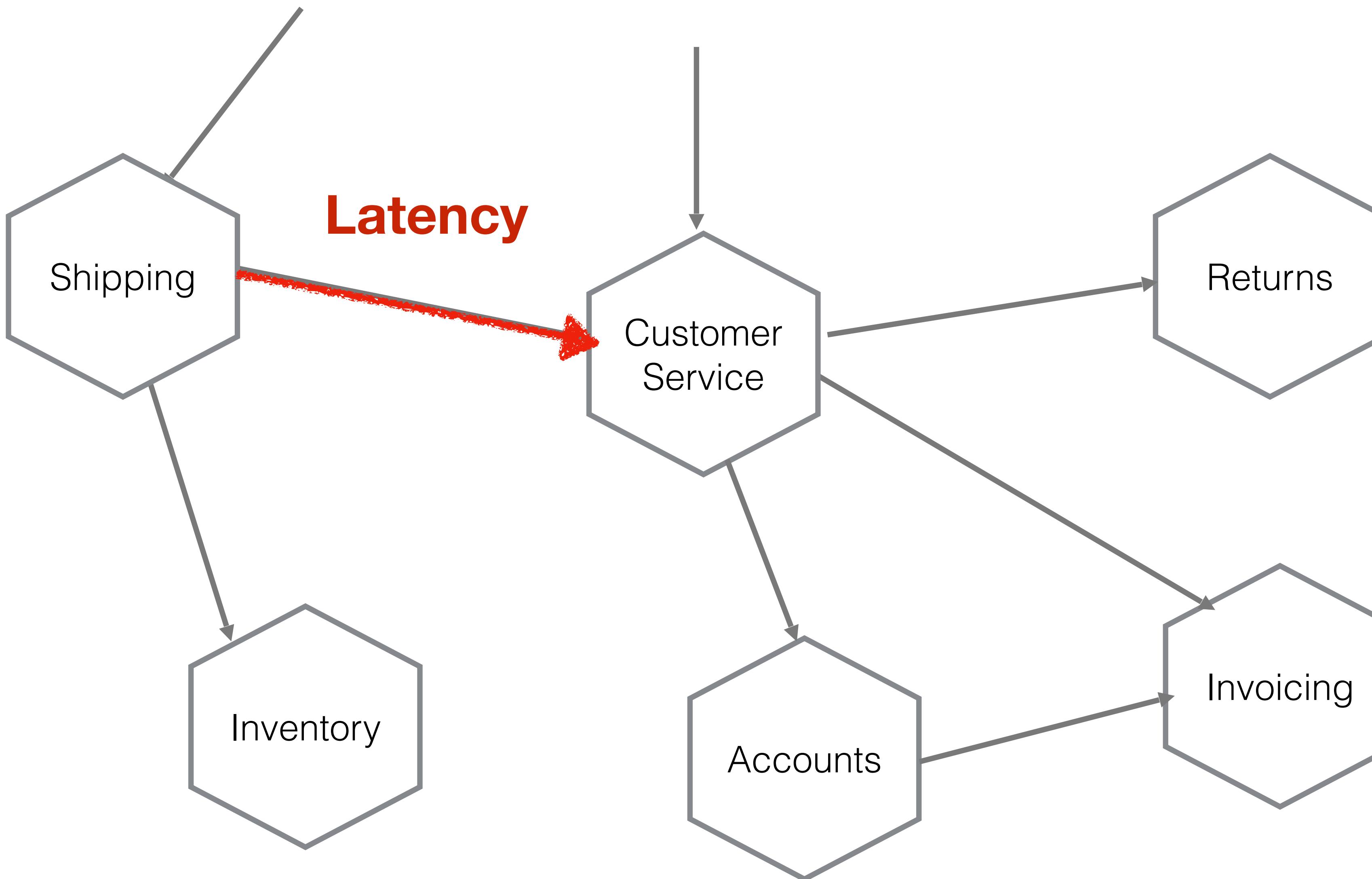
Hiding the data storage of a microservice is about hiding implementation detail

Information hiding makes independent deployability possible

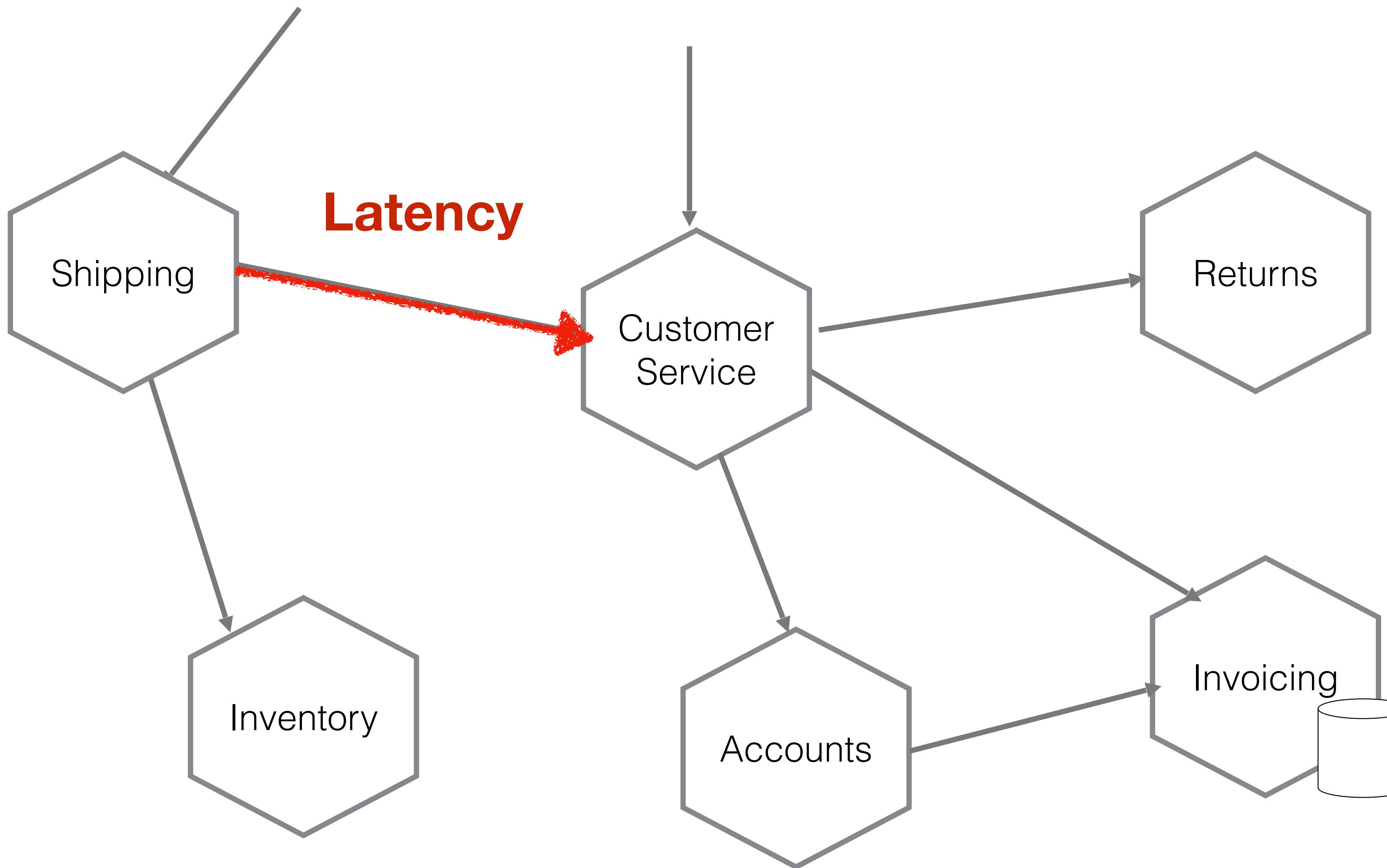
PROBLEMS WITH MICROSERVICES



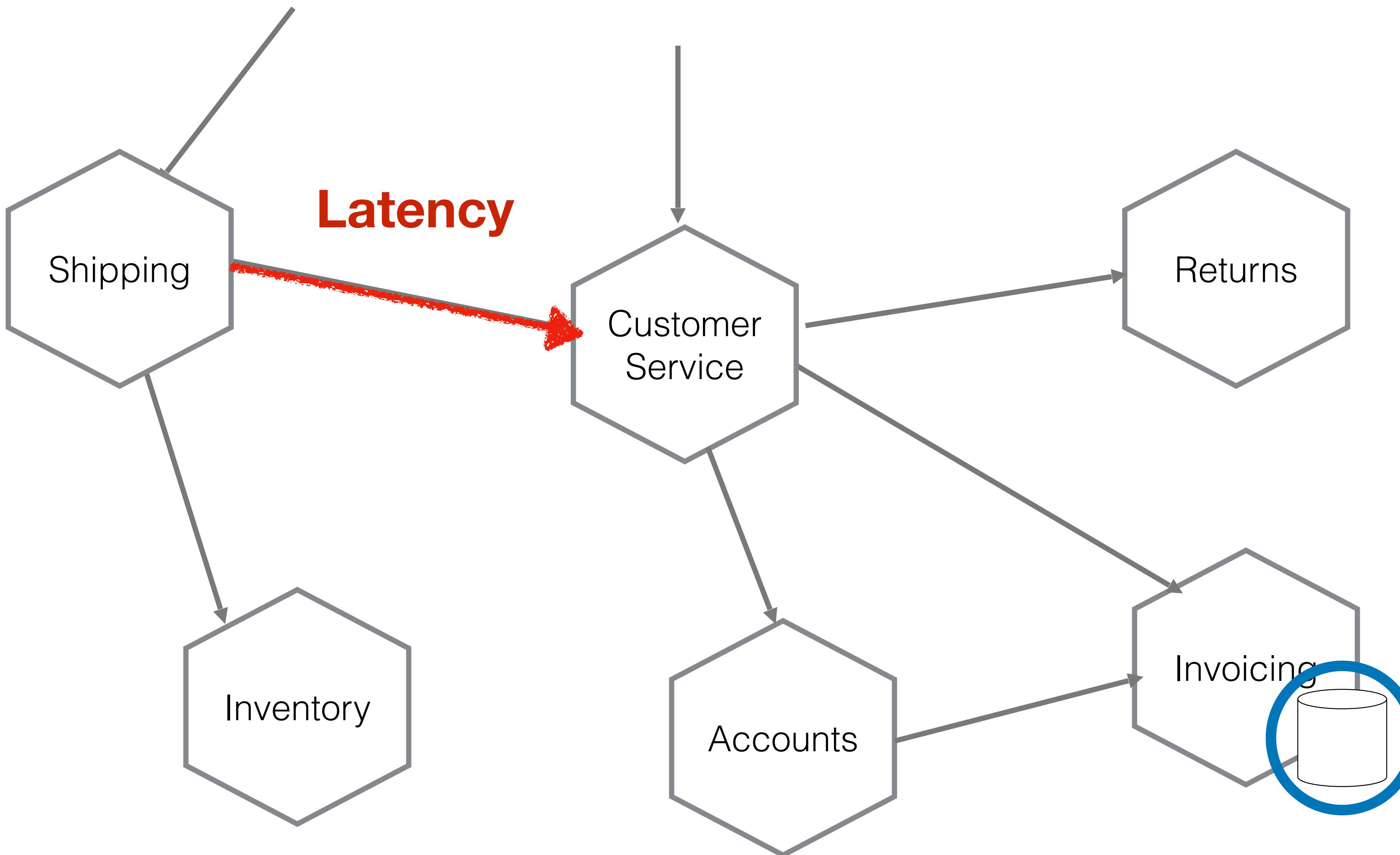
PROBLEMS WITH MICROSERVICES



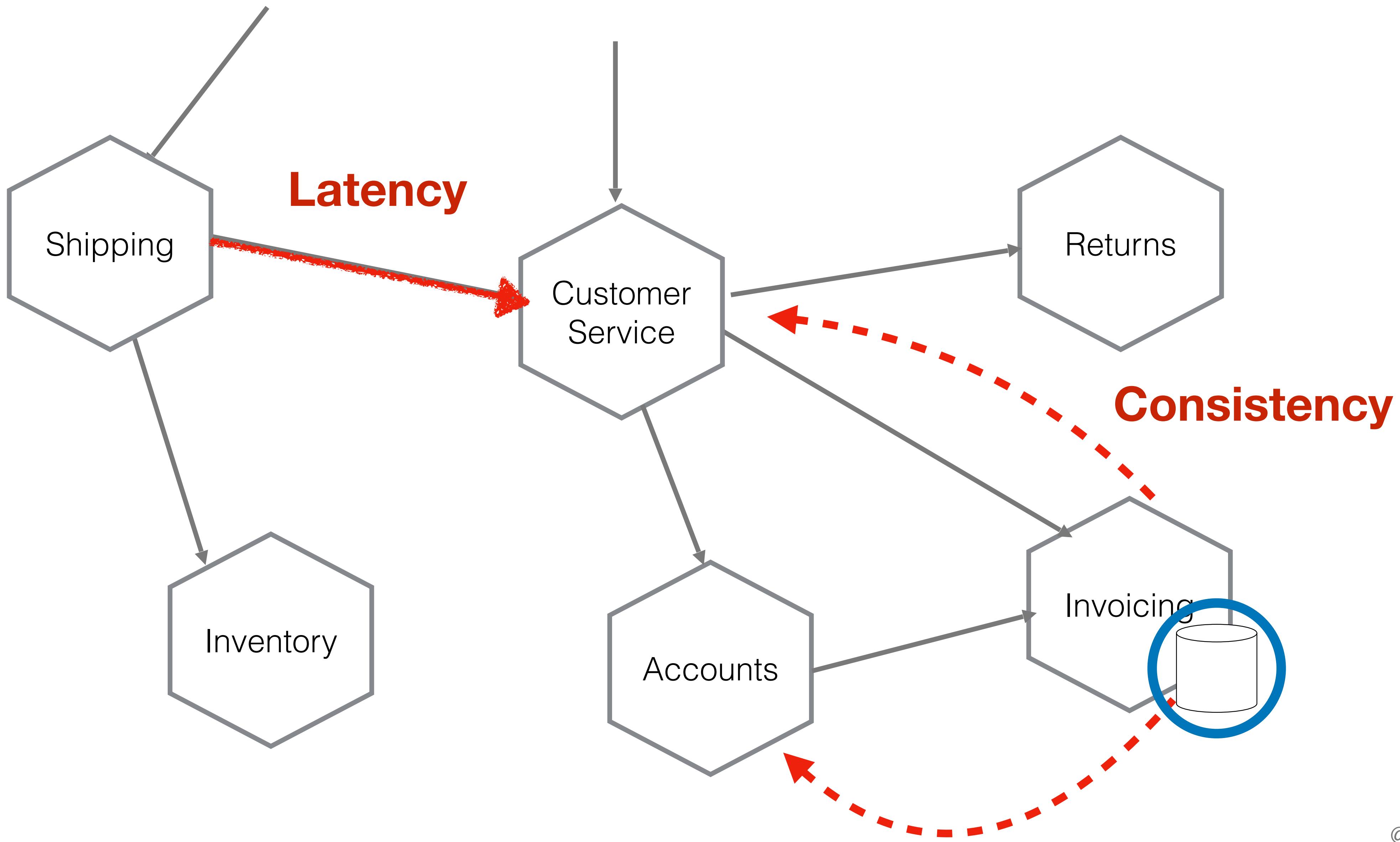
PROBLEMS WITH MICROSERVICES



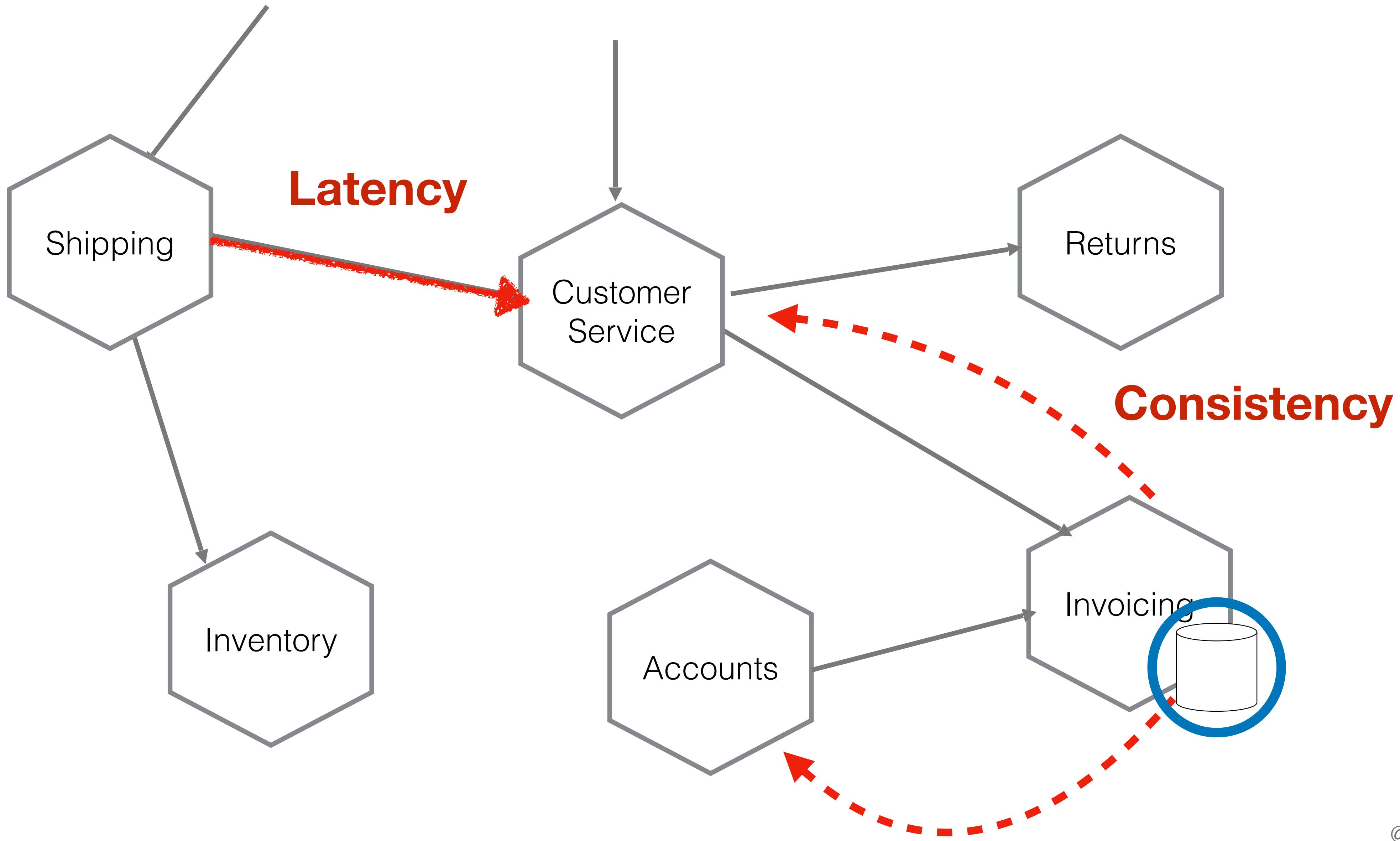
PROBLEMS WITH MICROSERVICES



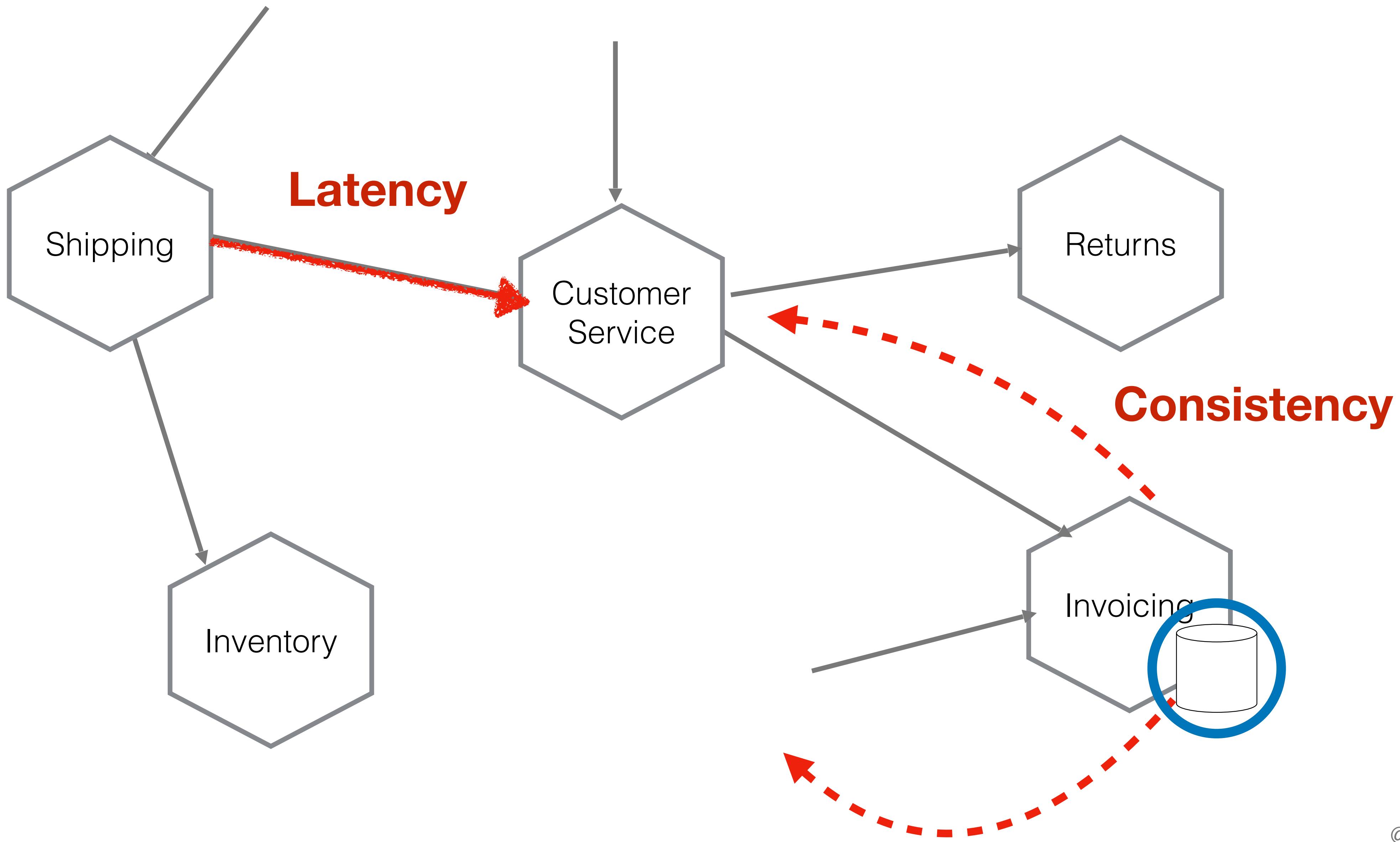
PROBLEMS WITH MICROSERVICES



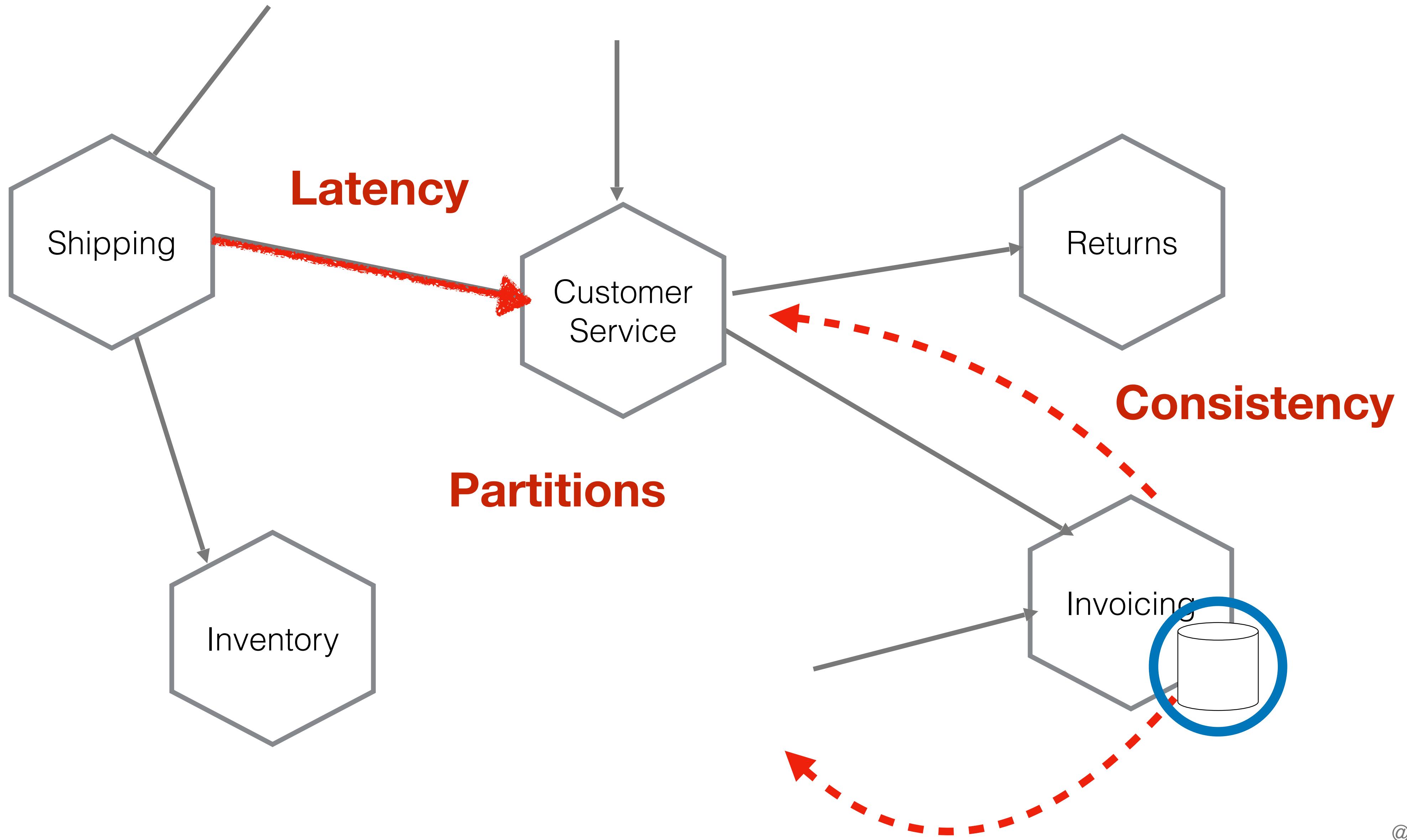
PROBLEMS WITH MICROSERVICES



PROBLEMS WITH MICROSERVICES



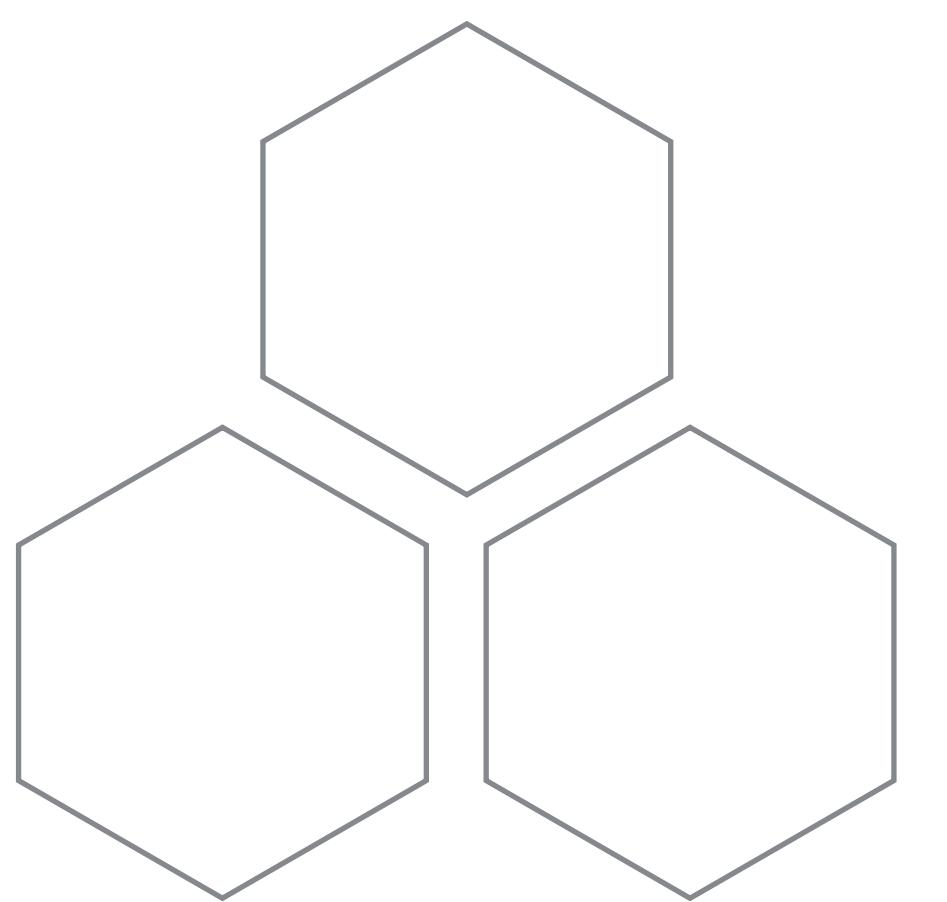
PROBLEMS WITH MICROSERVICES

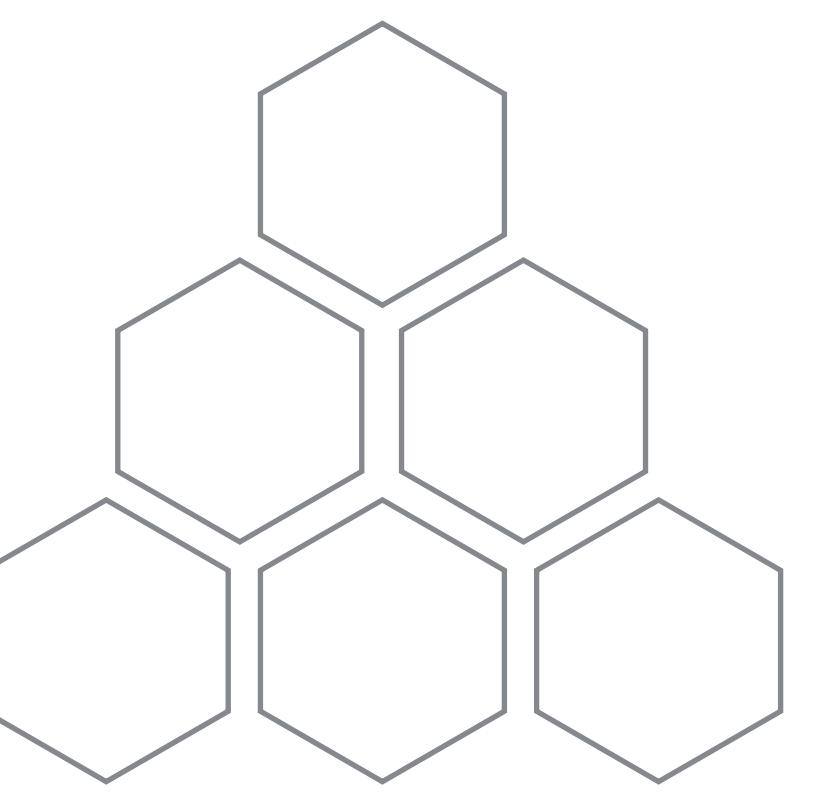
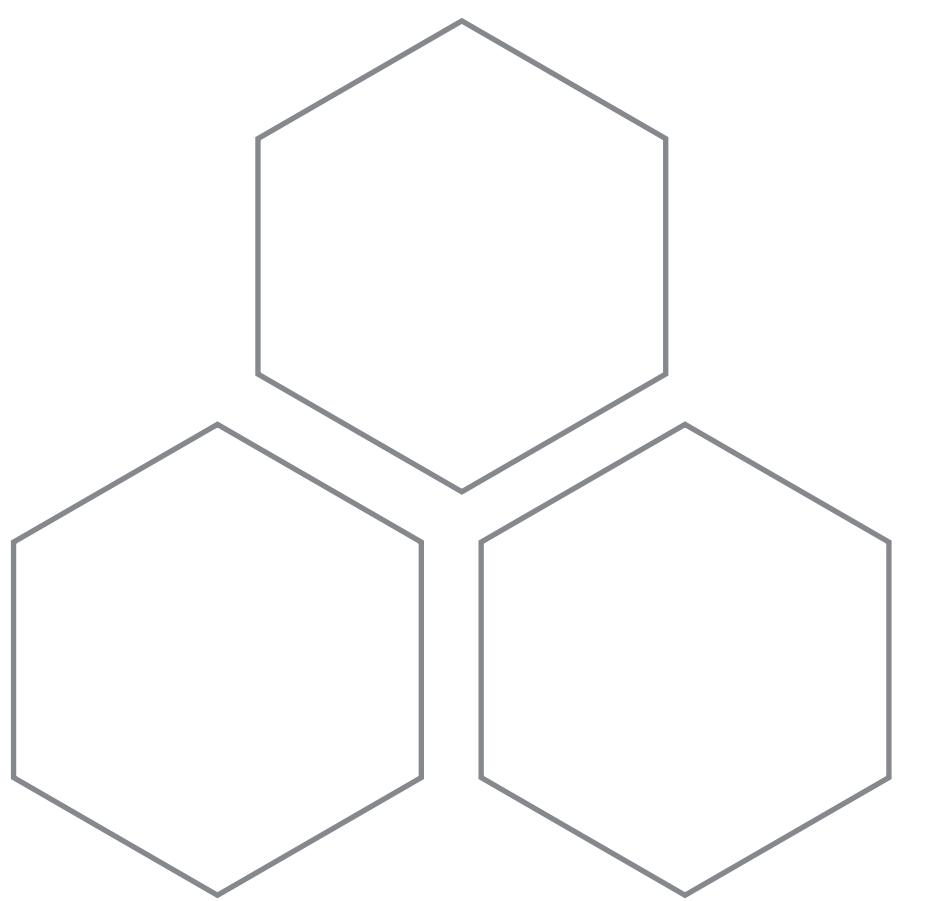


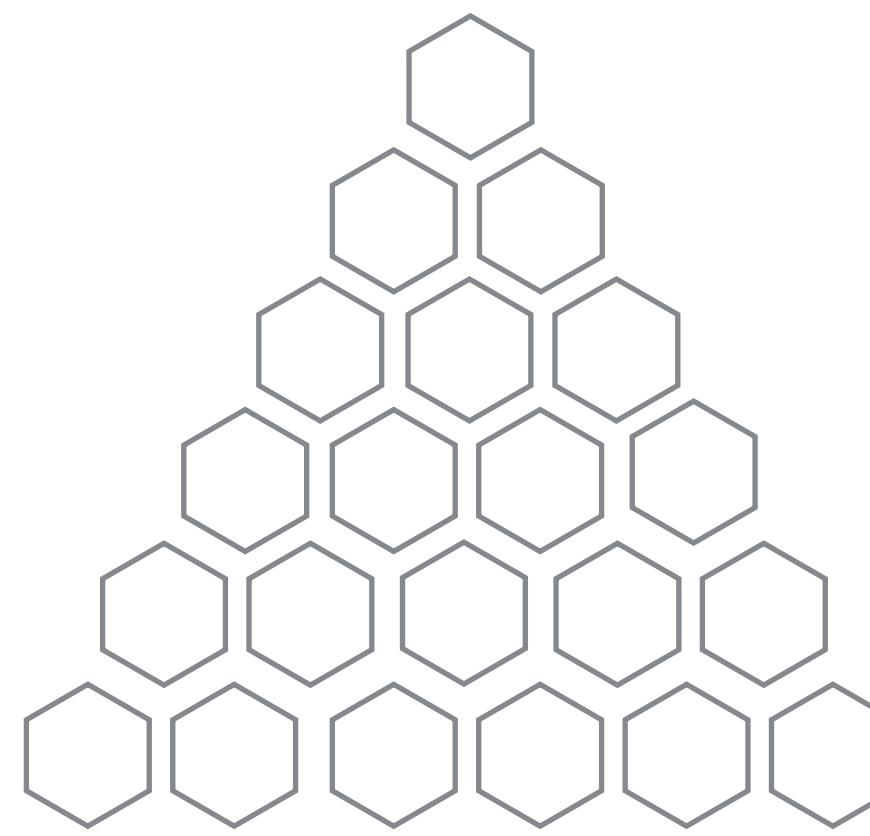
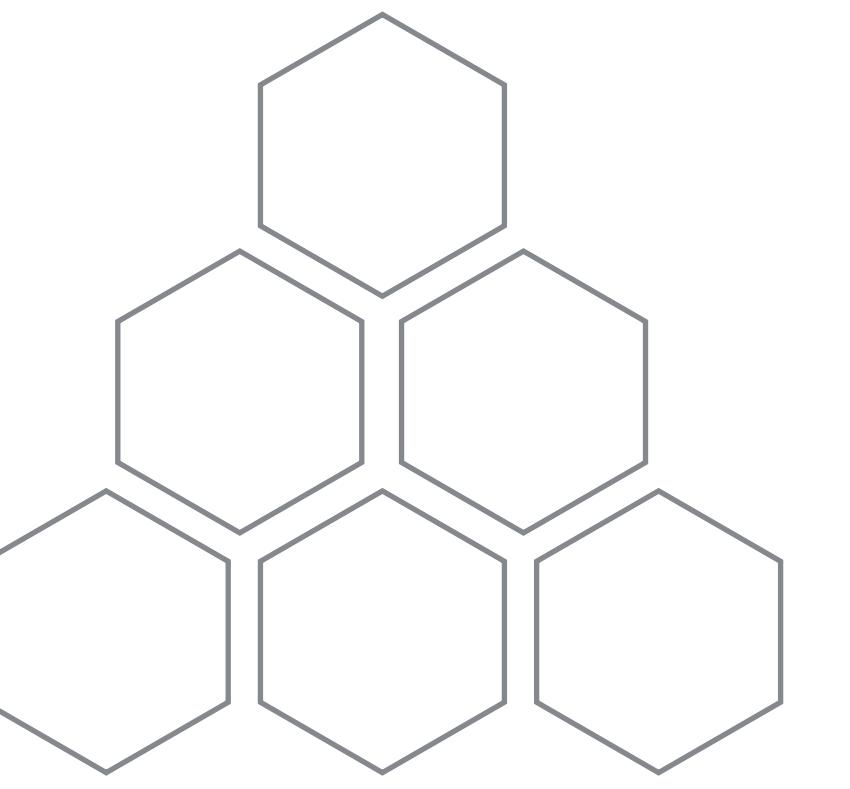
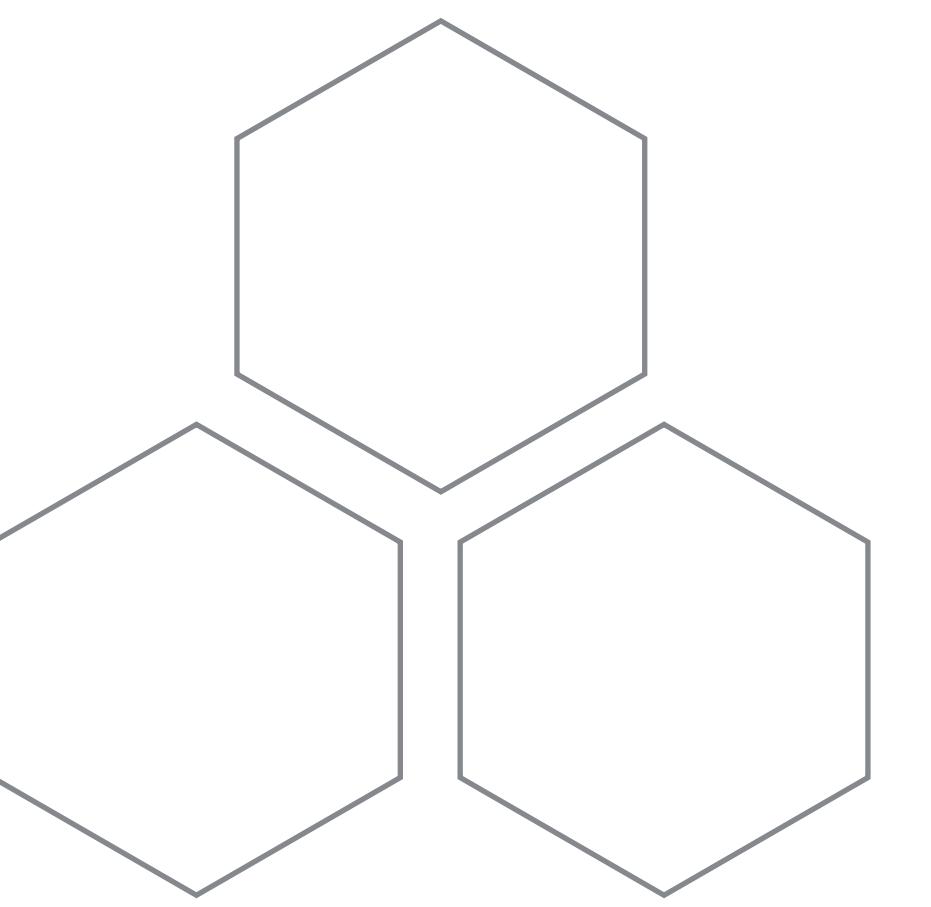
LOUDNESS

10



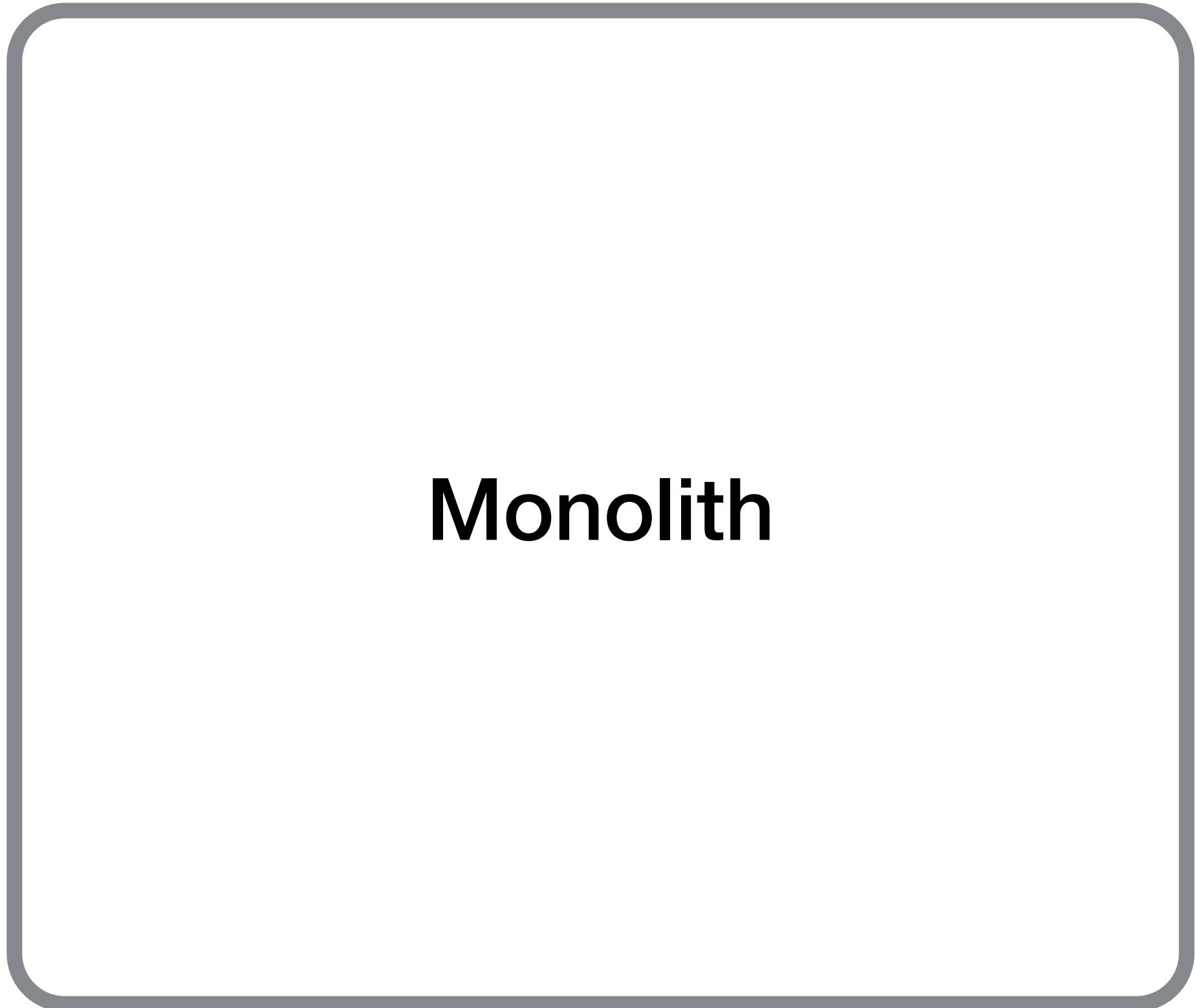






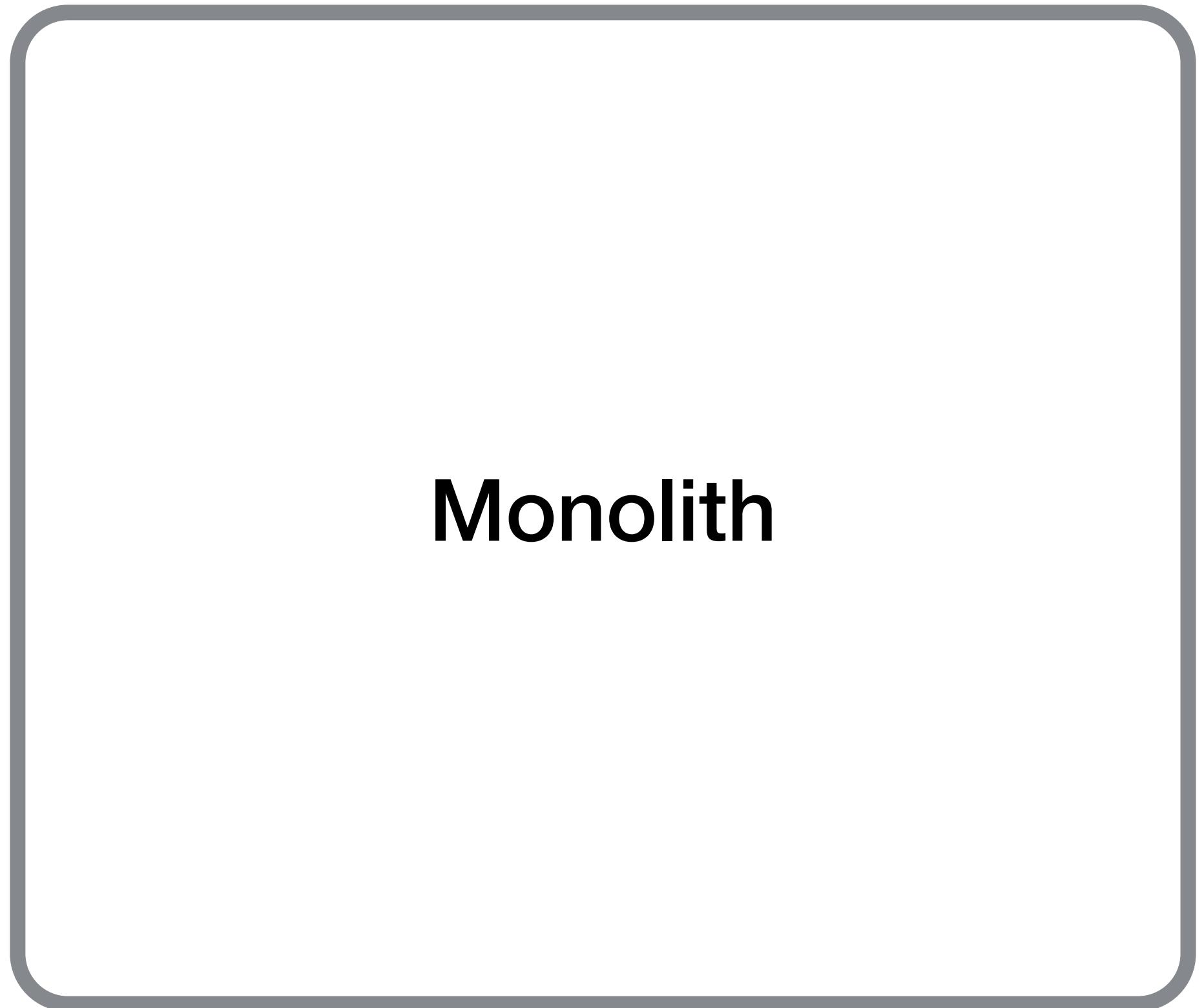
**You won't appreciate the true horror, pain and suffering of
microservices until you're running them in production**

INCREMENTAL DECOMPOSITION FTW

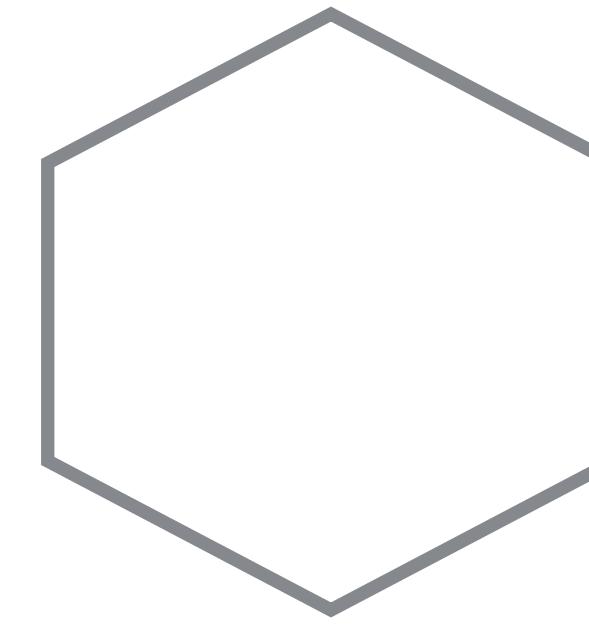


Monolith

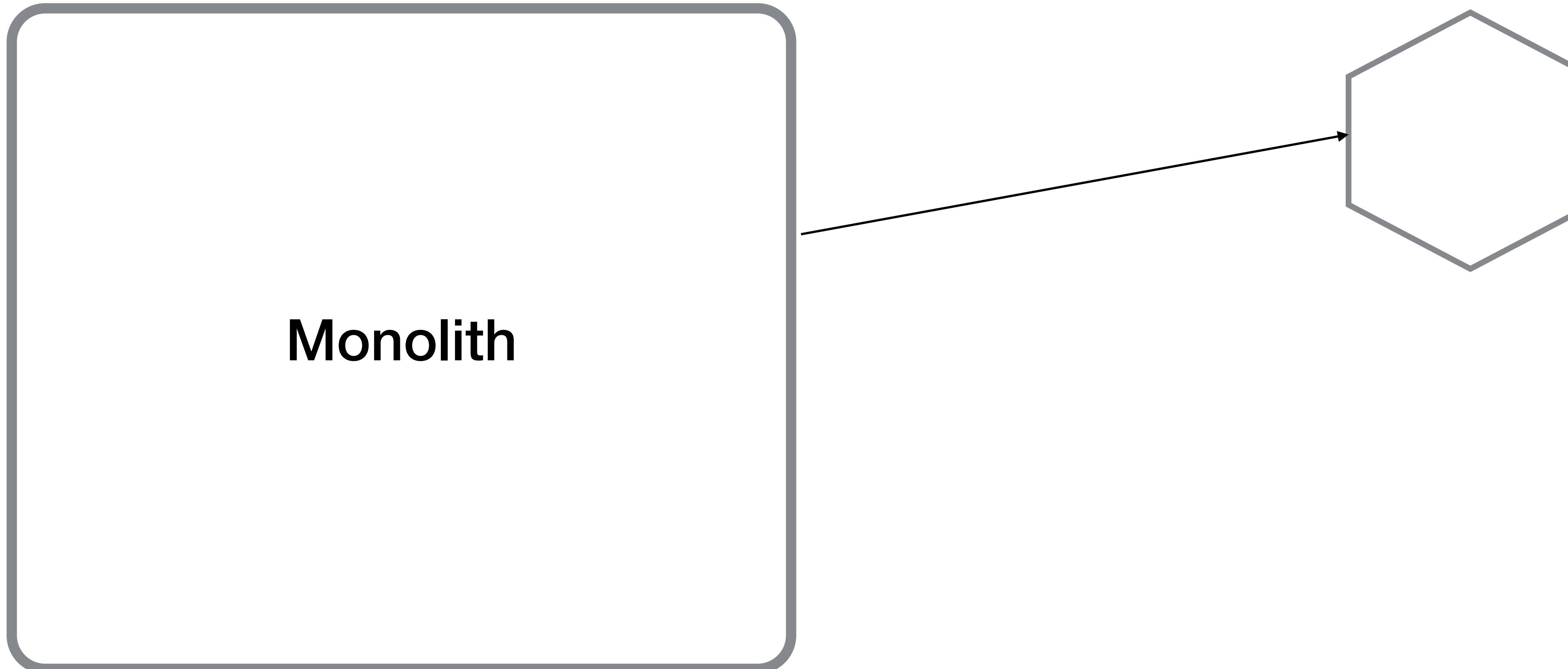
INCREMENTAL DECOMPOSITION FTW



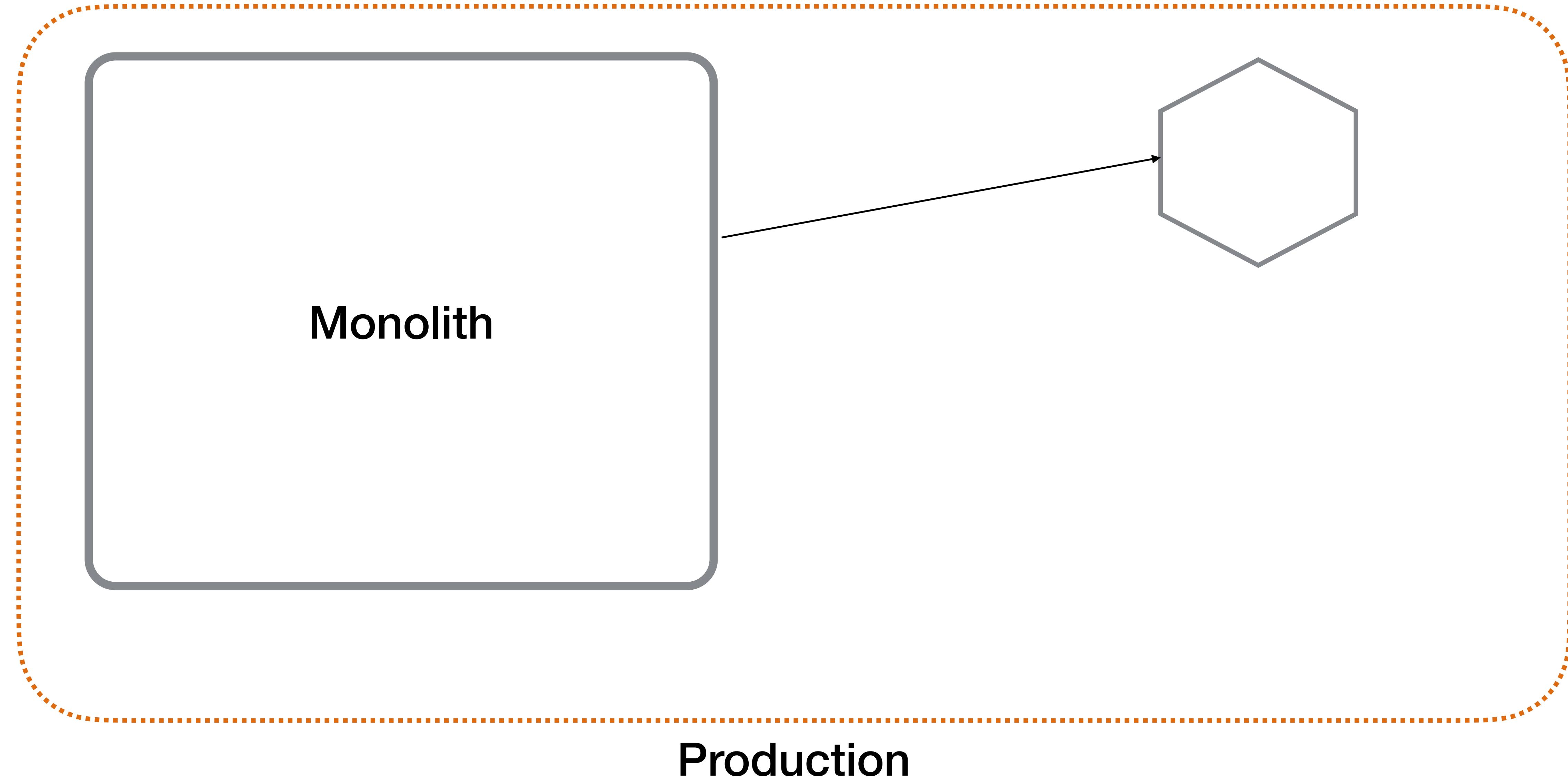
Monolith



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

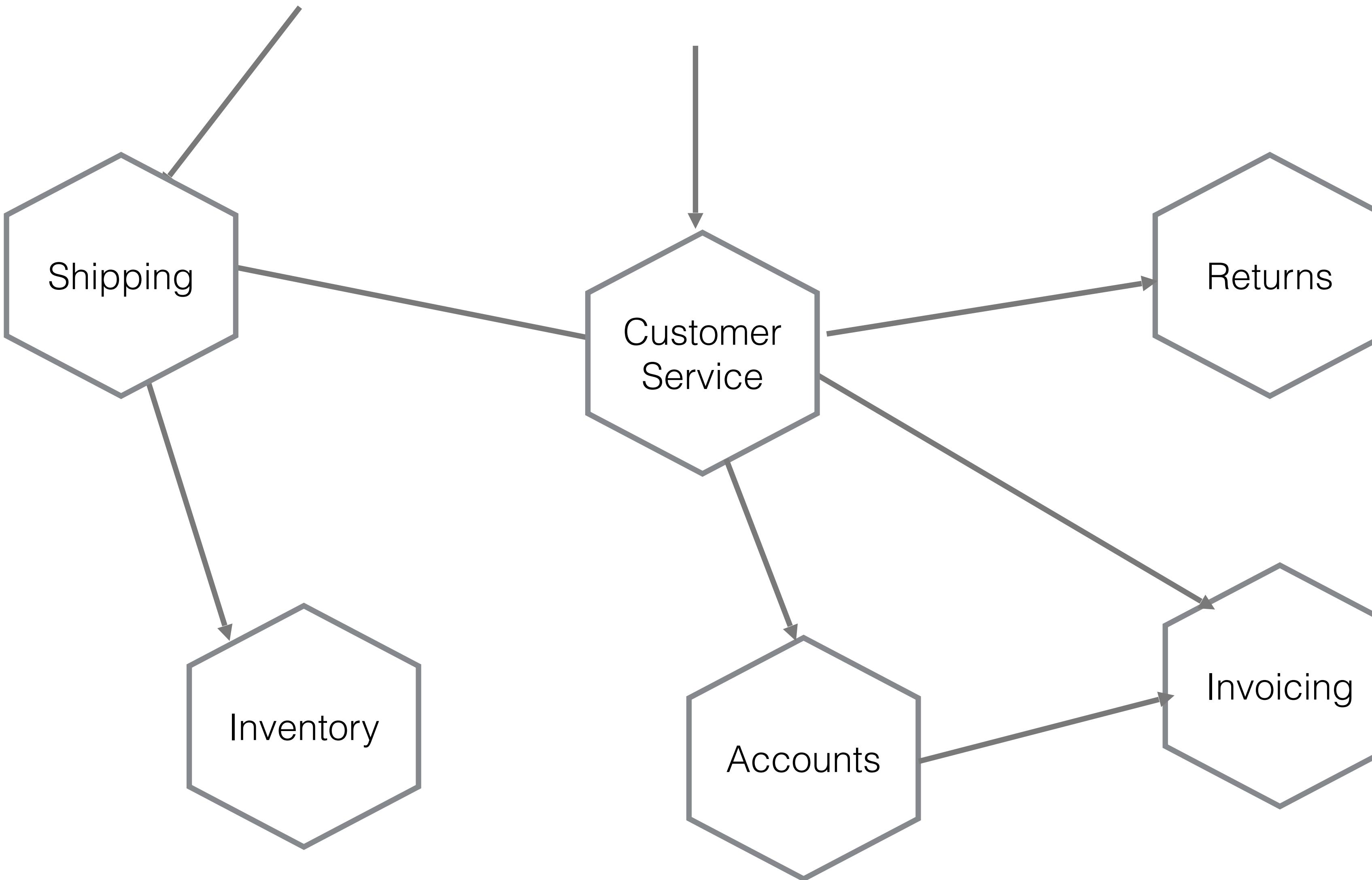
Introduction

Shared Data Patterns

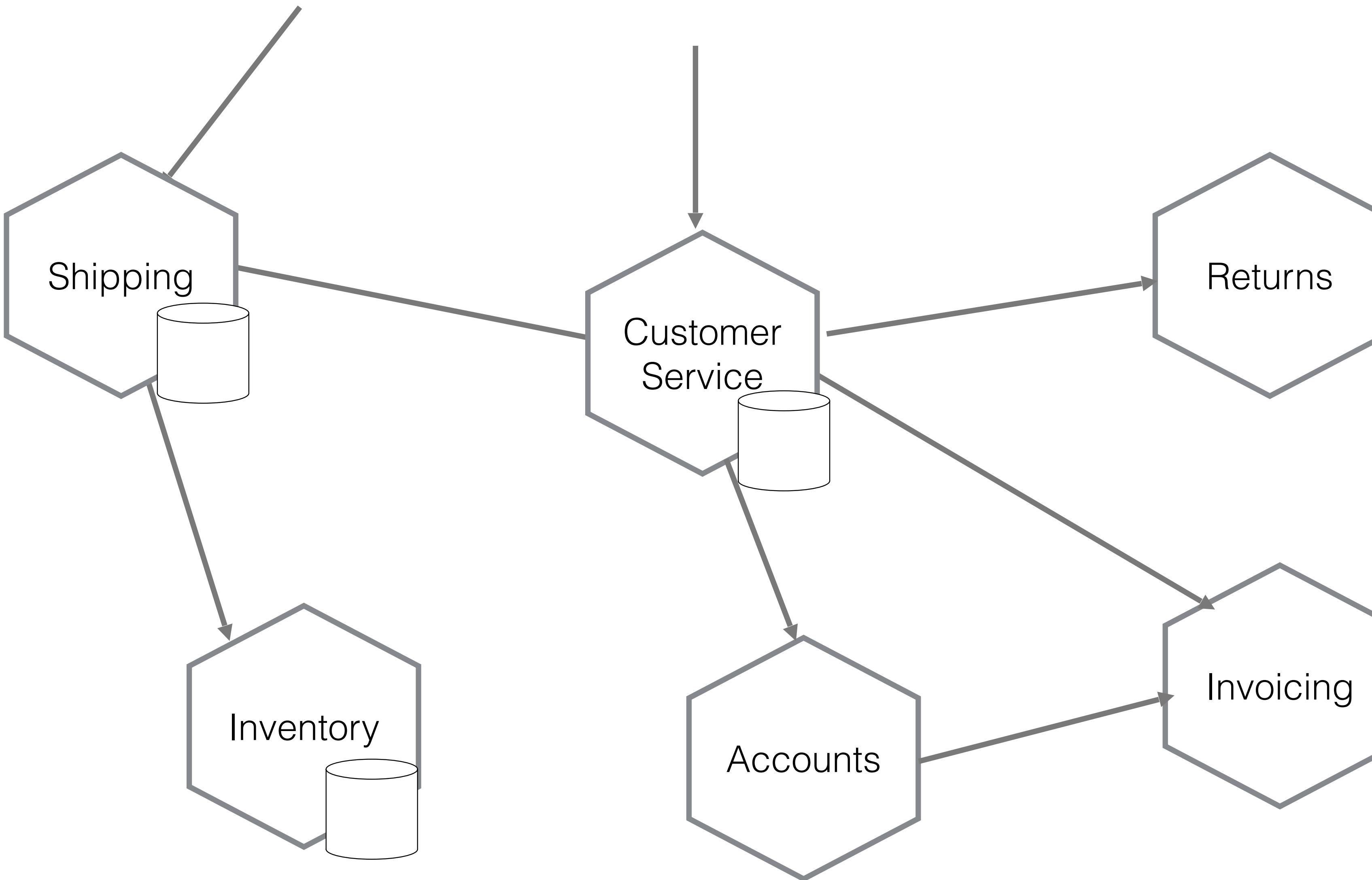
Migration Order

Decomposition Patterns

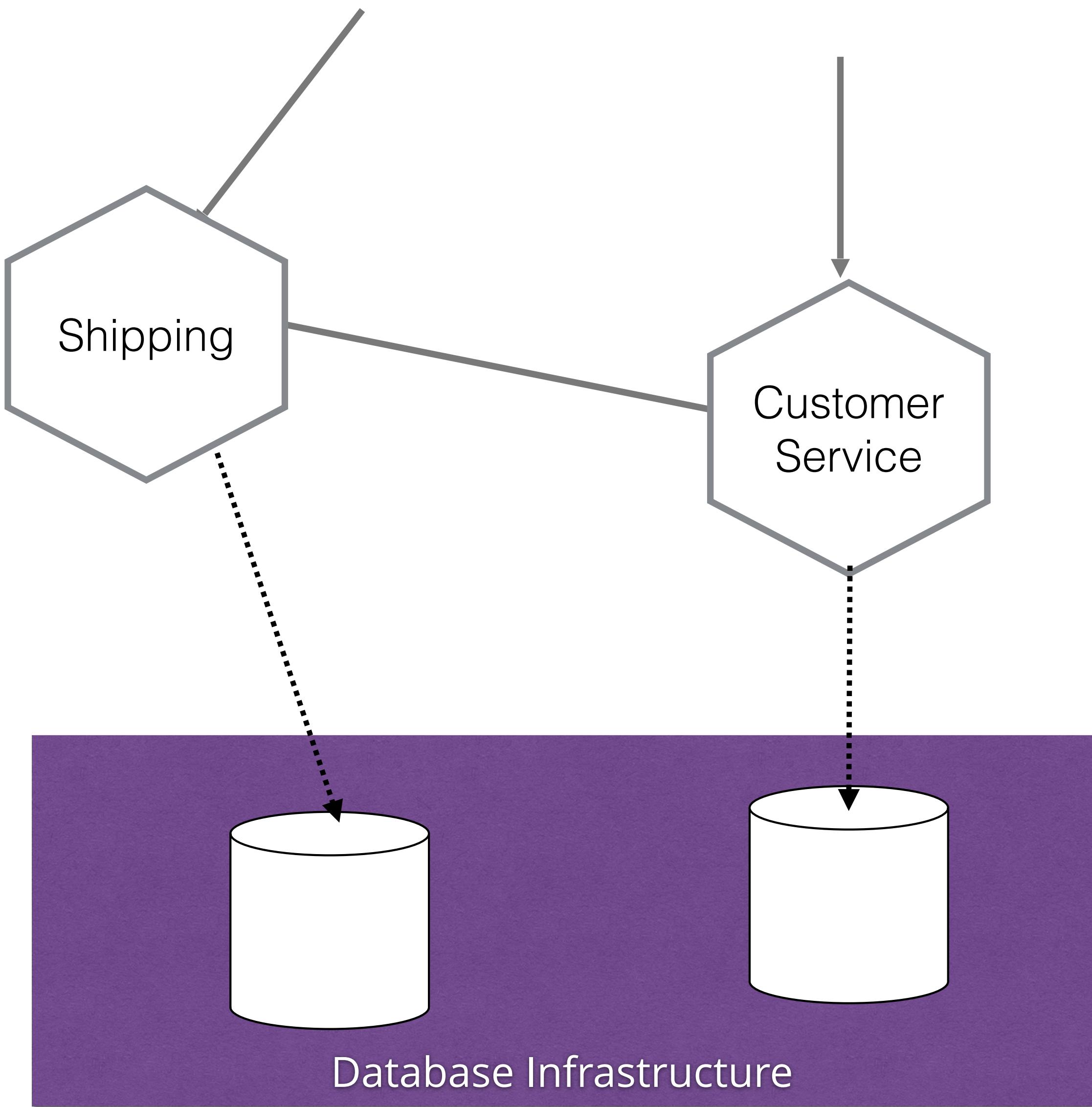
LANDSCAPE OF DATABASES



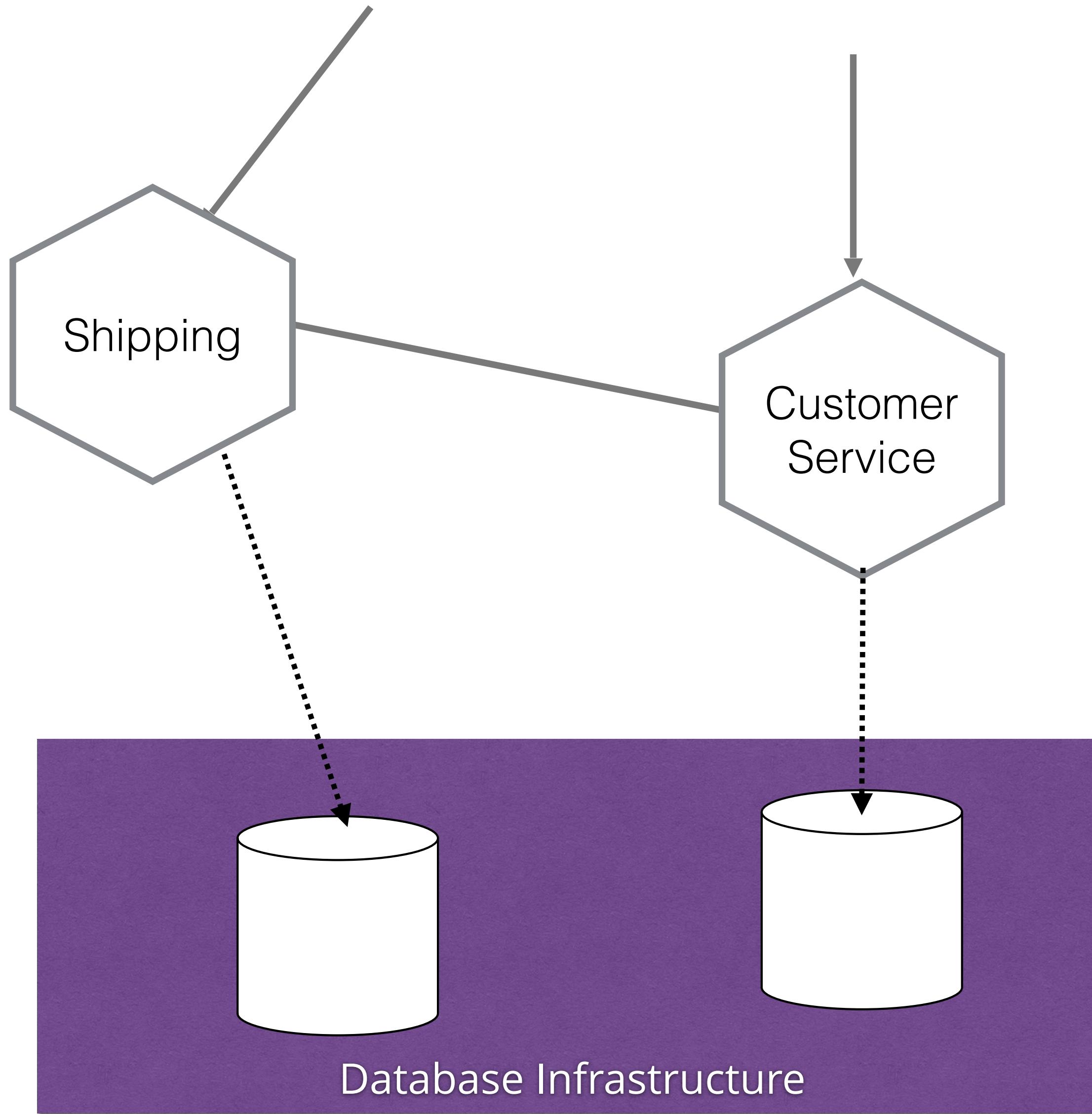
LANDSCAPE OF DATABASES



PATTERN: SHARED DB INFRASTRUCTURE

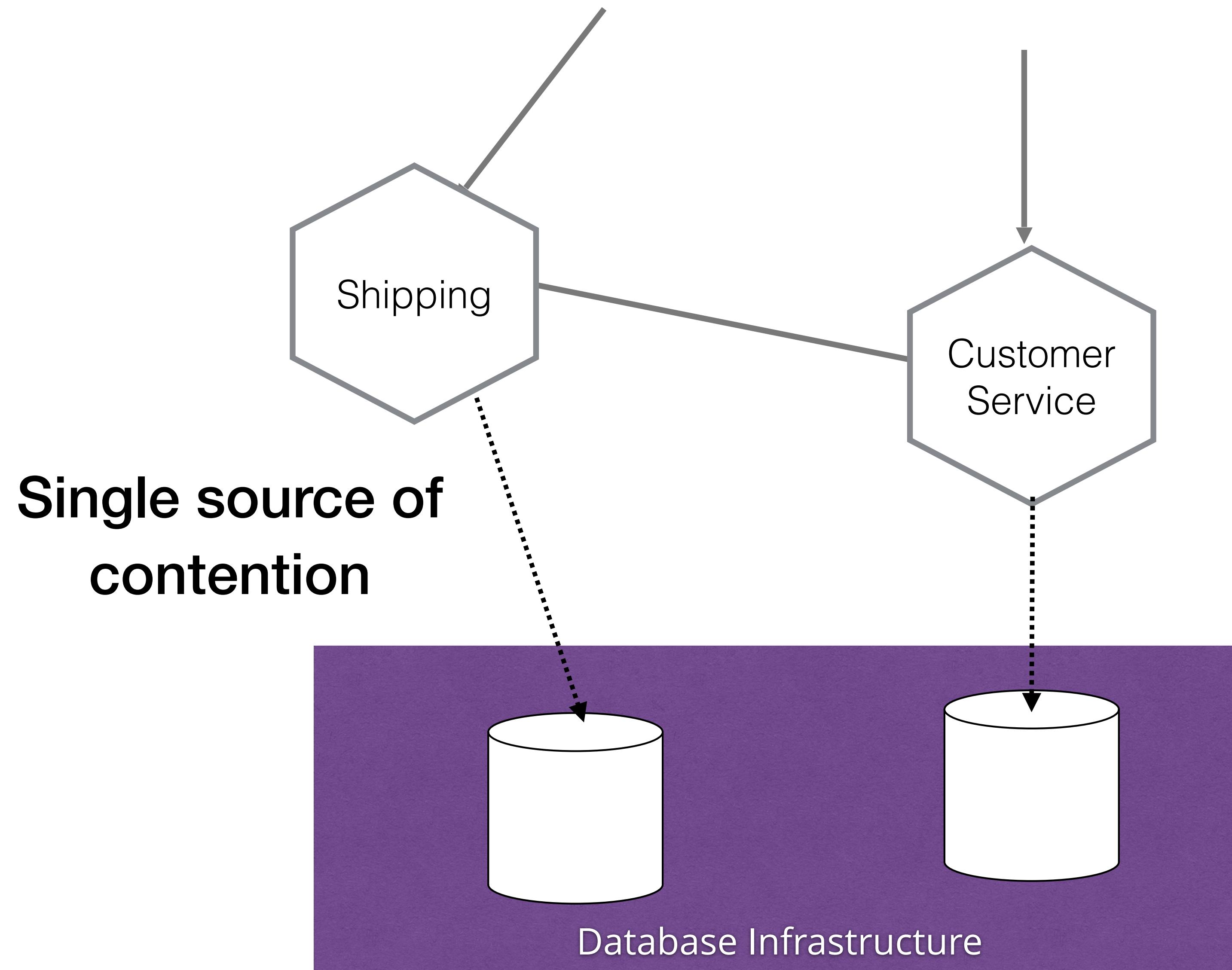


PATTERN: SHARED DB INFRASTRUCTURE



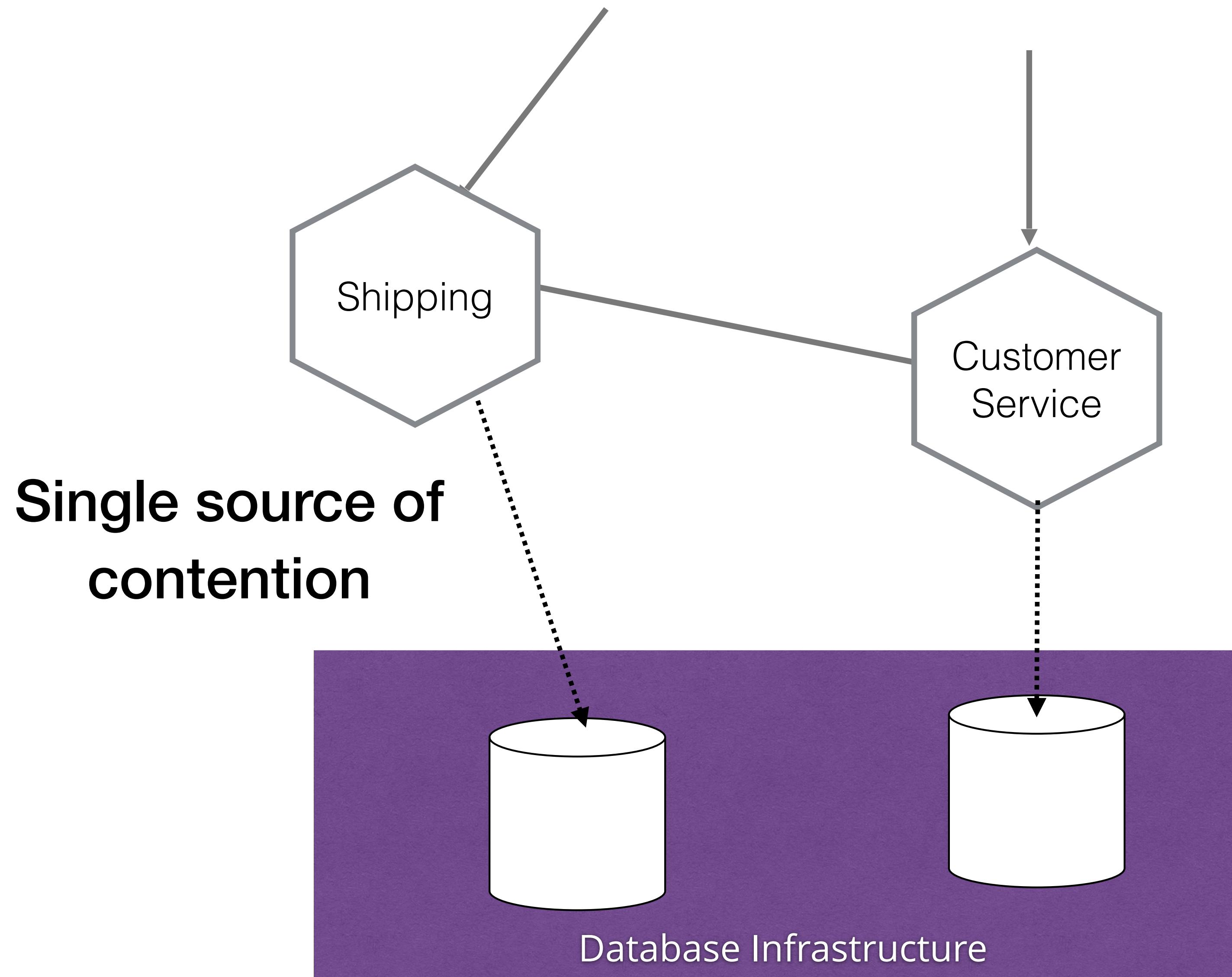
Reduce costs/overhead

PATTERN: SHARED DB INFRASTRUCTURE



Reduce costs/overhead

PATTERN: SHARED DB INFRASTRUCTURE

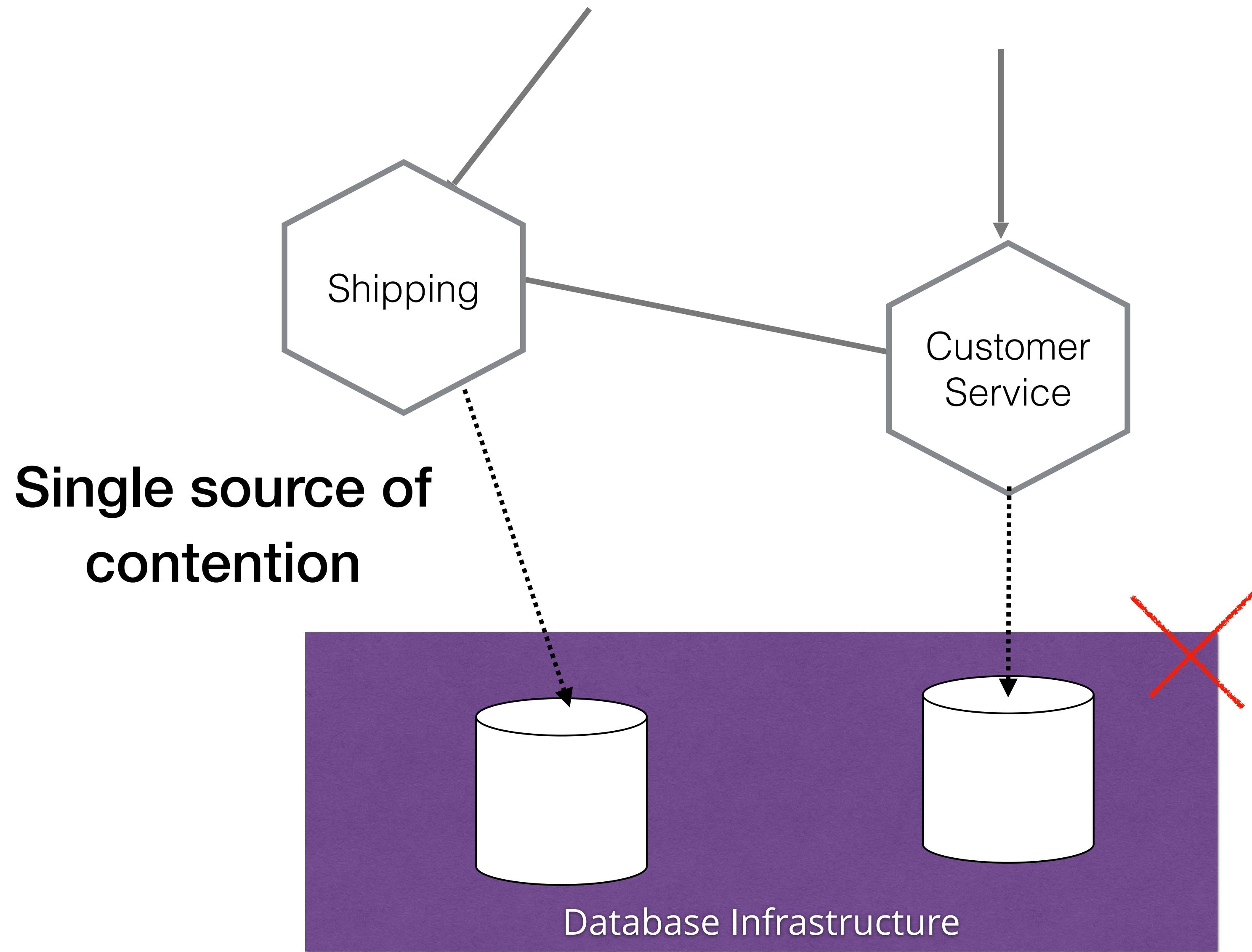


Single source of contention

Reduce costs/overhead

Single point of failure

PATTERN: SHARED DB INFRASTRUCTURE

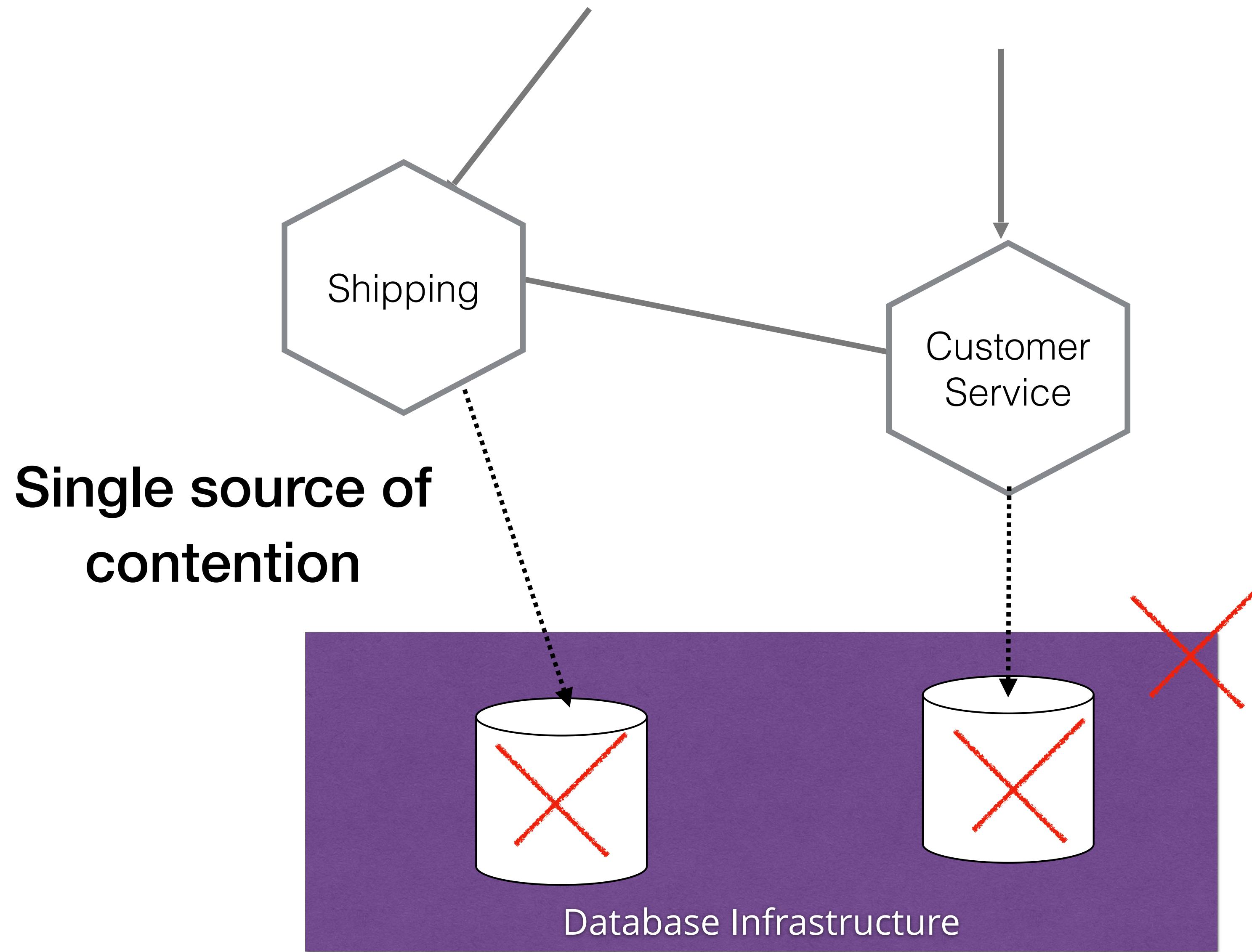


Single source of contention

Reduce costs/overhead

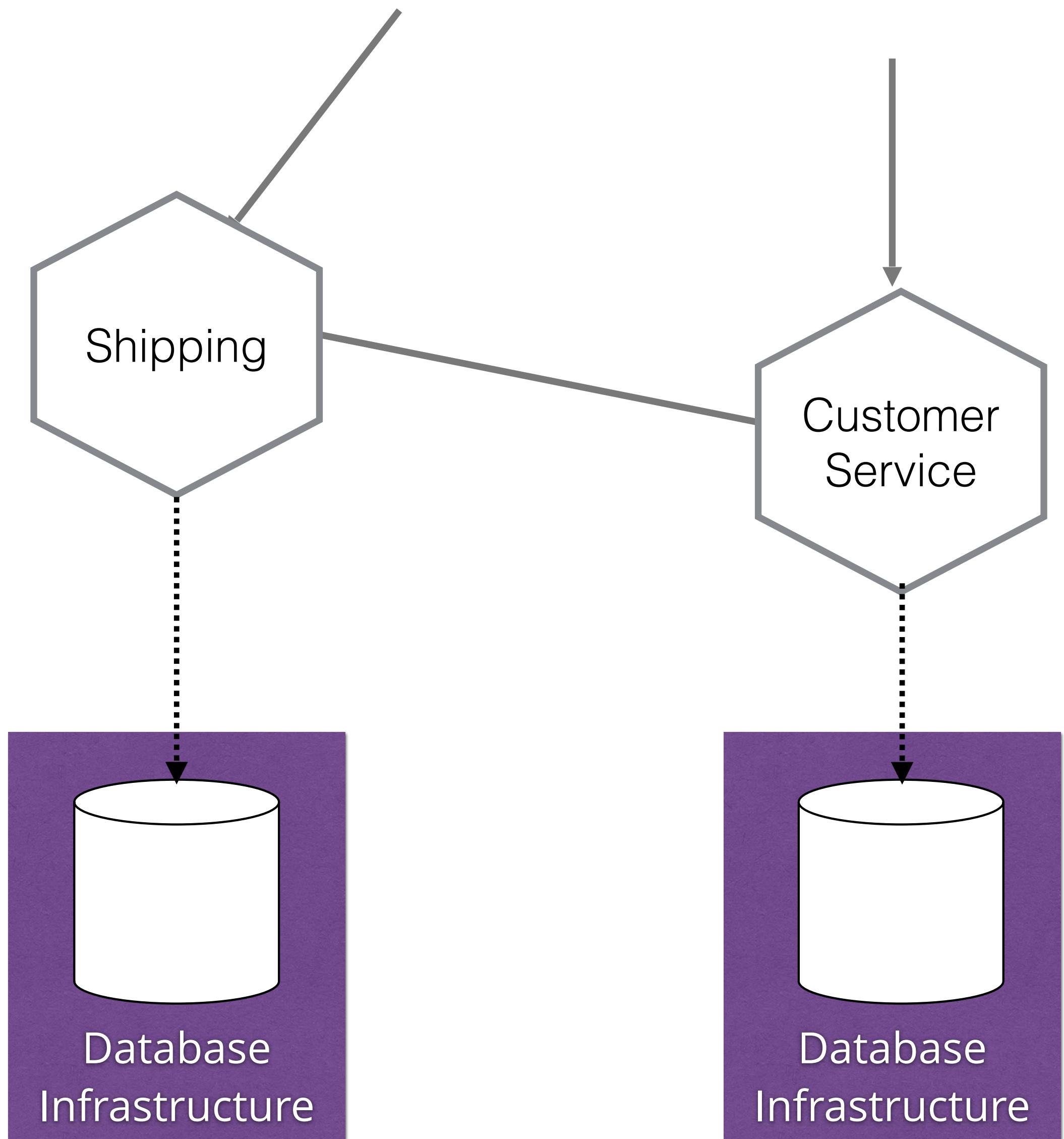
Single point of failure

PATTERN: SHARED DB INFRASTRUCTURE

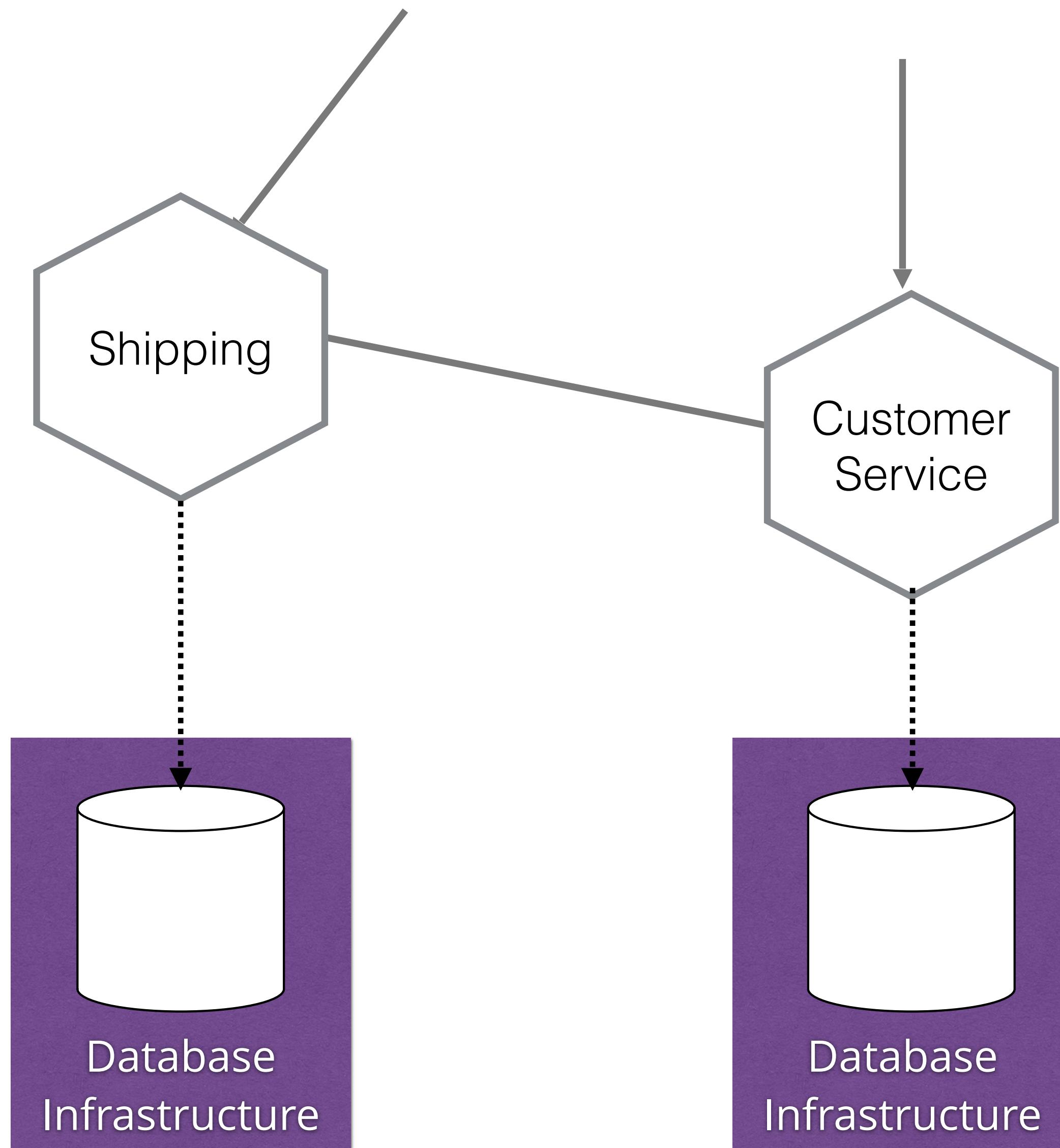


Reduce costs/overhead

PATTERN: ISOLATED DB INFRASTRUCTURE

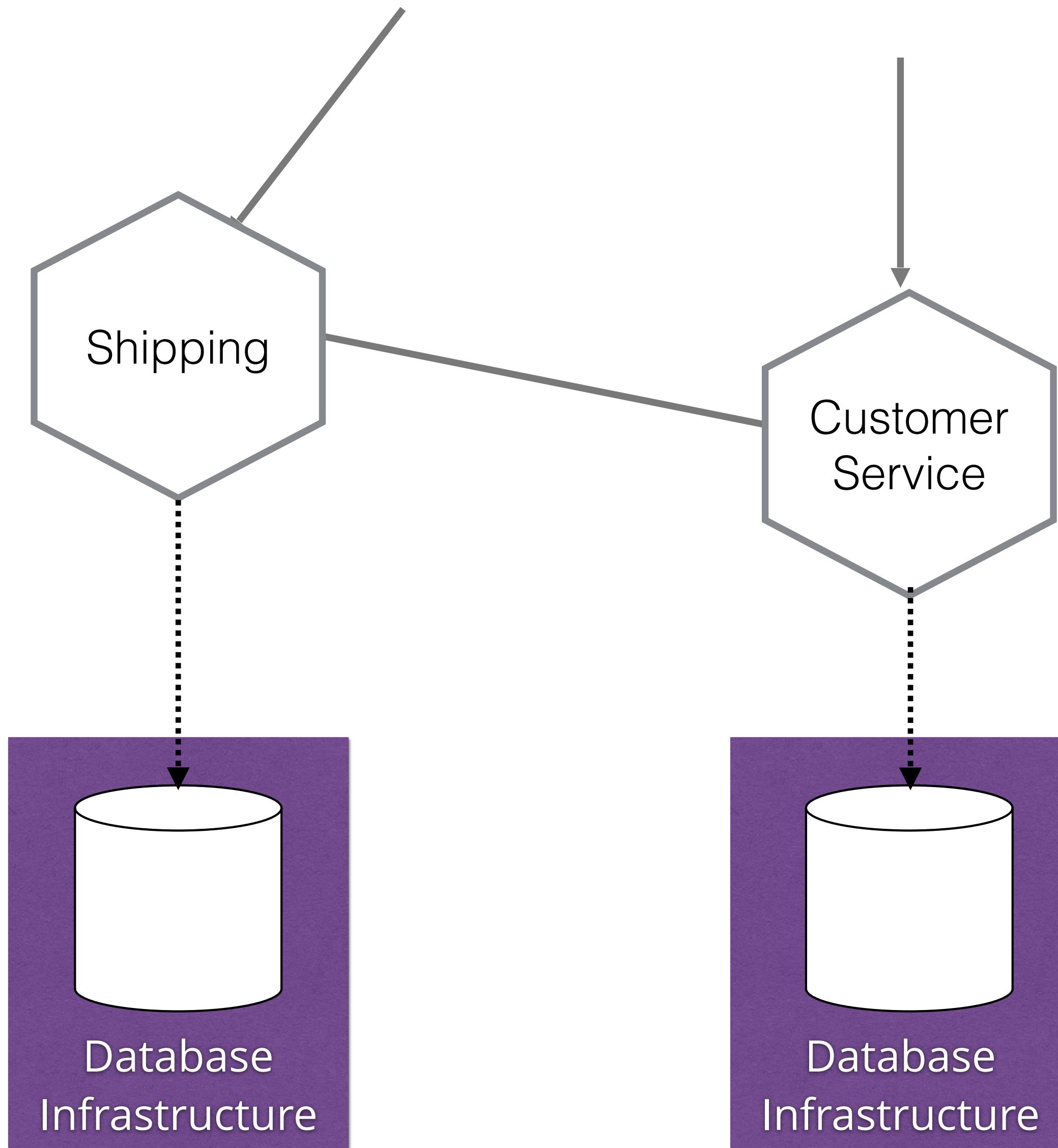


PATTERN: ISOLATED DB INFRASTRUCTURE



More DB infra to manage

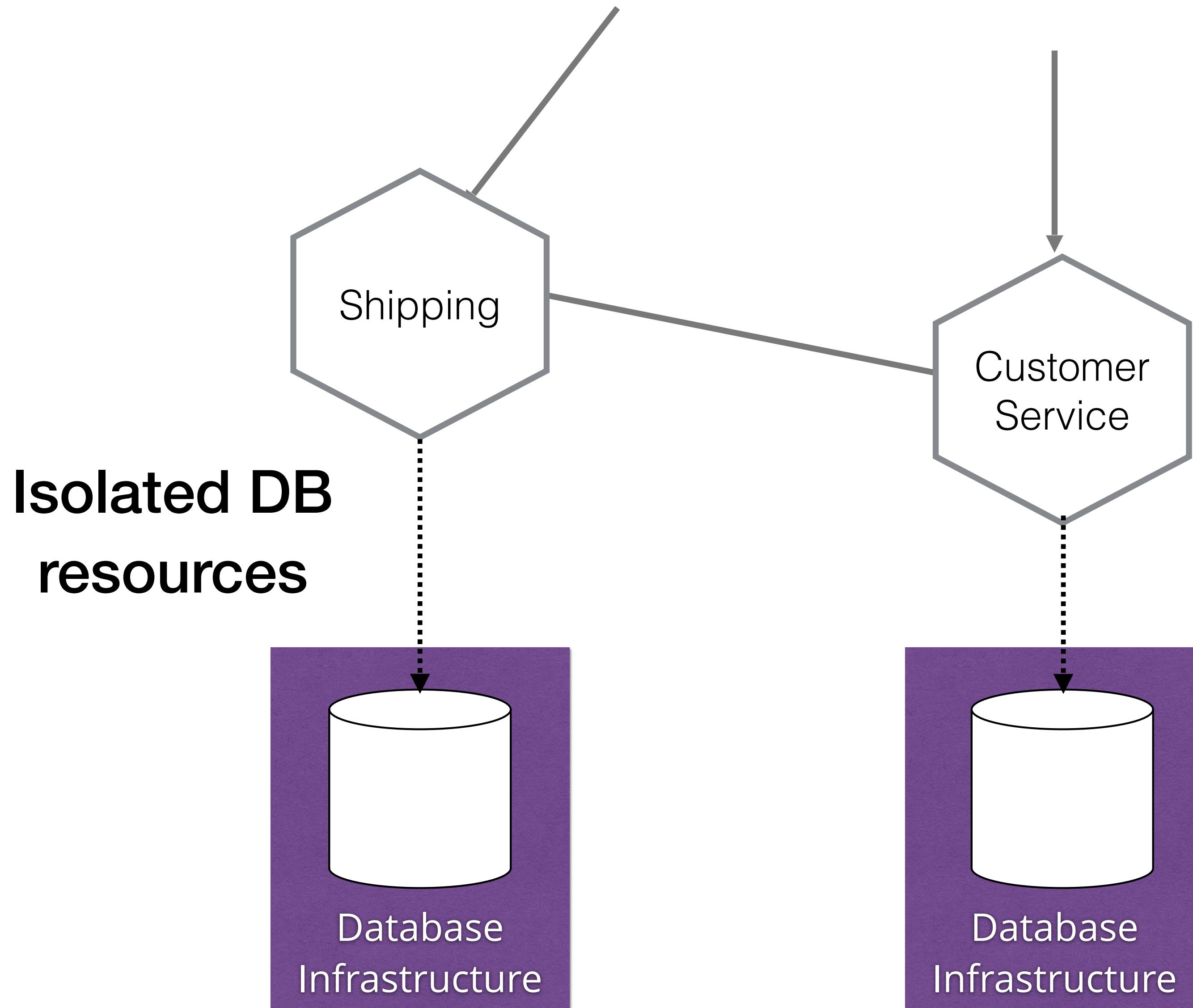
PATTERN: ISOLATED DB INFRASTRUCTURE



More DB infra to manage

Likely more expensive

PATTERN: ISOLATED DB INFRASTRUCTURE

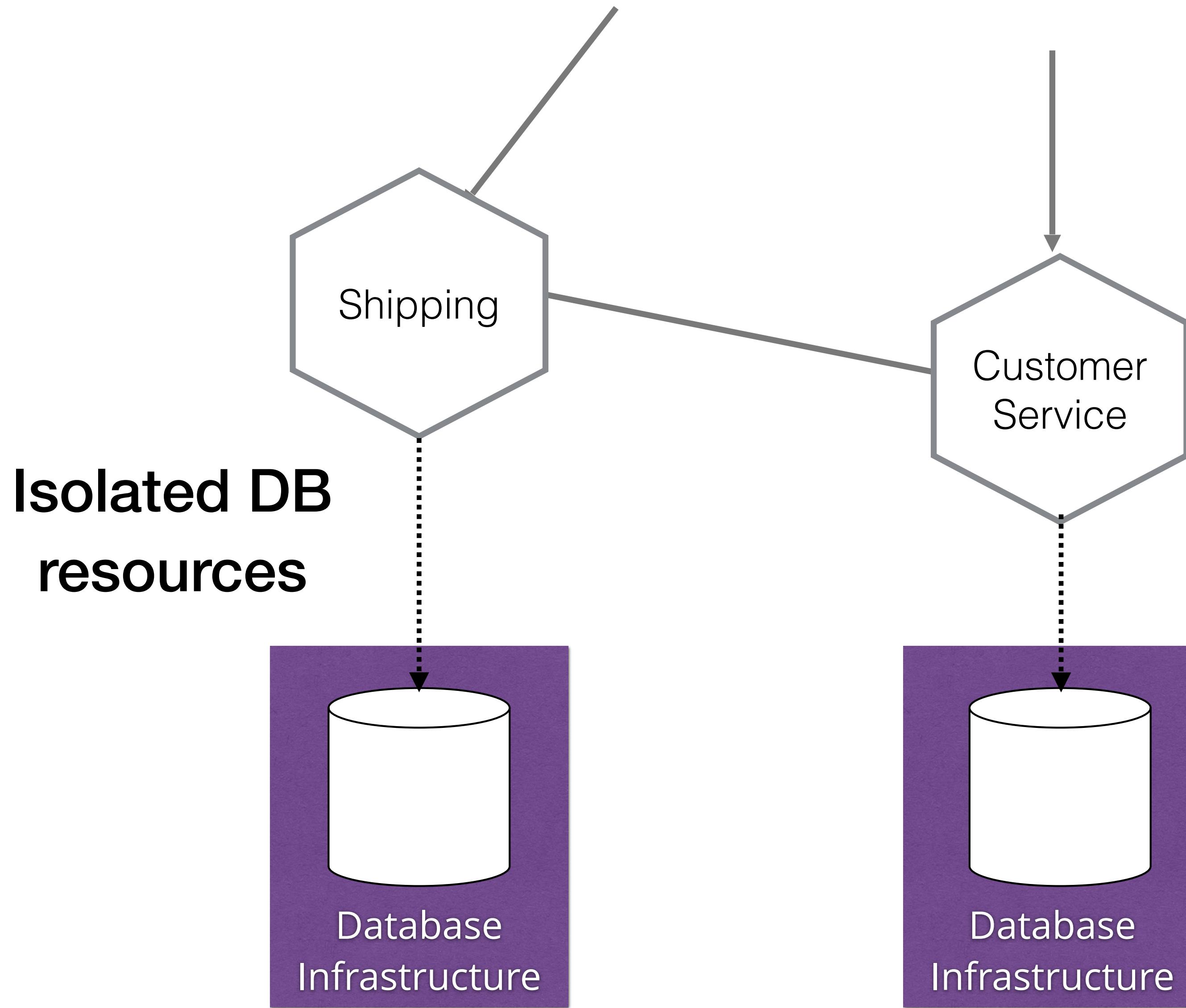


More DB infra to manage

Likely more expensive

Isolated DB
resources

PATTERN: ISOLATED DB INFRASTRUCTURE

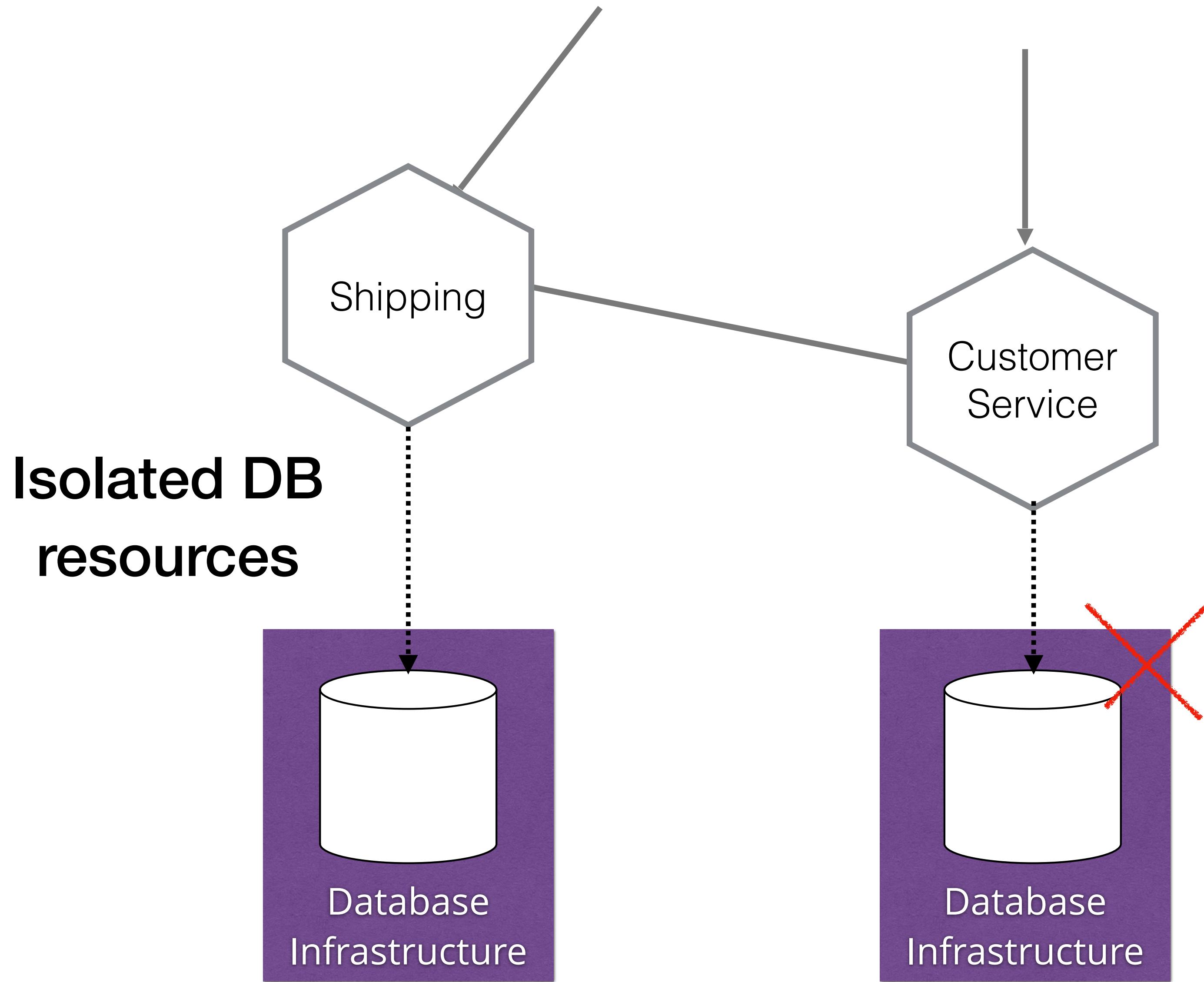


More DB infra to manage

Likely more expensive

Failure isolation

PATTERN: ISOLATED DB INFRASTRUCTURE

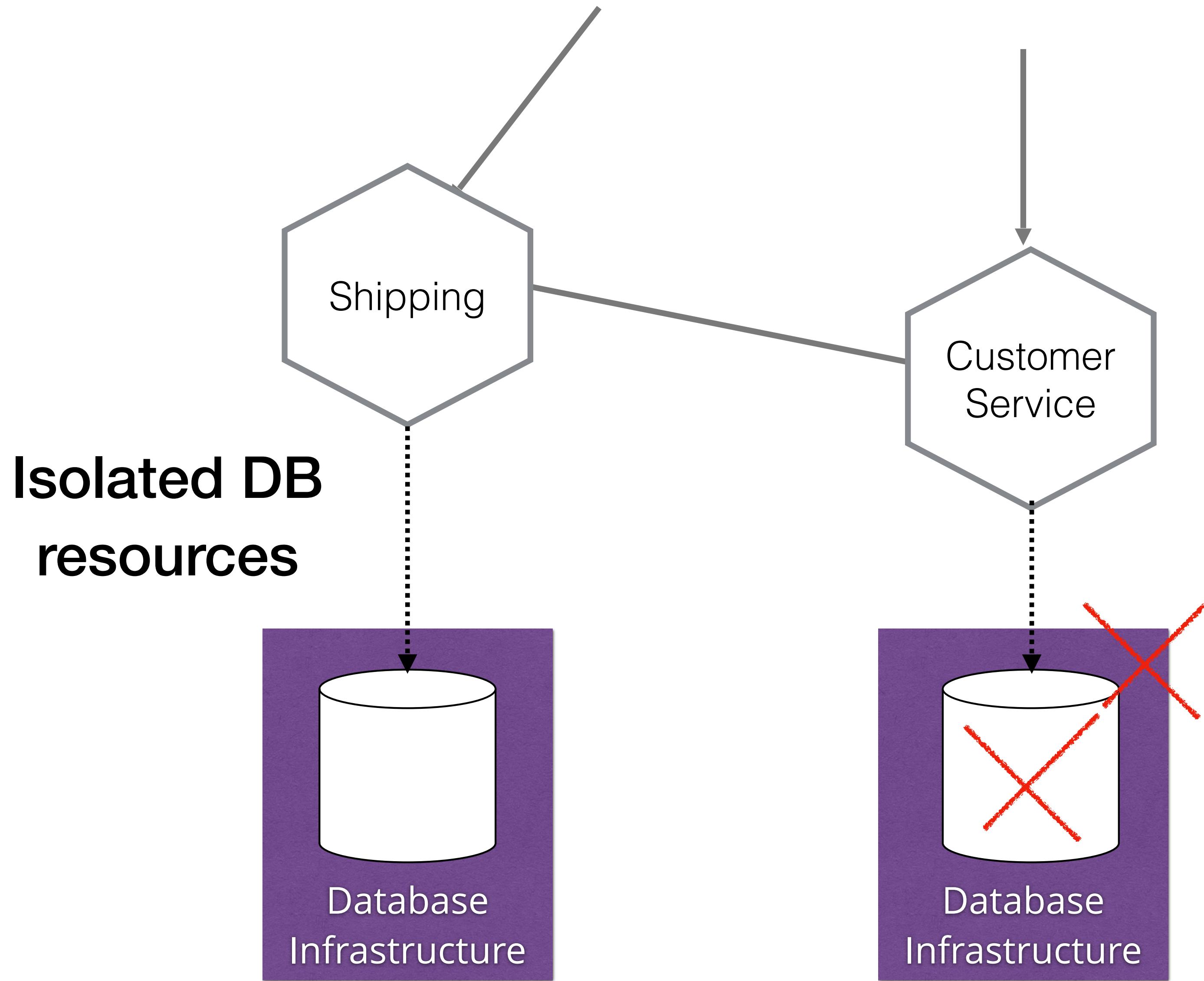


More DB infra to manage

Likely more expensive

Failure isolation

PATTERN: ISOLATED DB INFRASTRUCTURE



Isolated DB resources

More DB infra to manage

Likely more expensive

Failure isolation

IN GENERAL...

When managing your own infrastructure
(traditional DC, private cloud), then **Shared**
DB Infrastructure is most common

IN GENERAL...

When managing your own infrastructure
(traditional DC, private cloud), then **Shared DB Infrastructure** is most common

Isolated DB Infrastructure is much easier to justify on public cloud

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

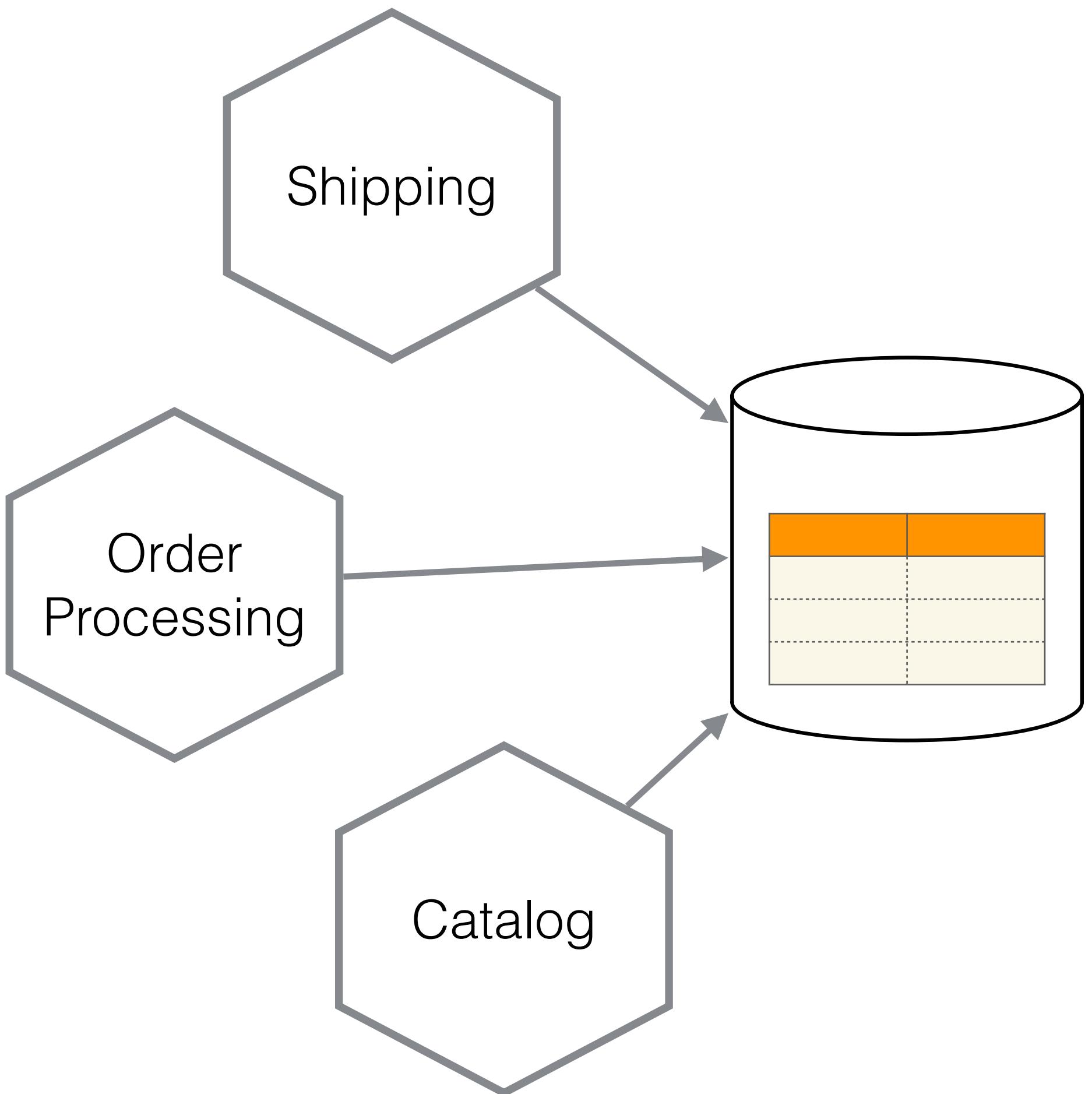
Relational (e.g. mysql, postgres, oracle)

Document (e.g. couchDB, mongo)

Column (e.g. Cassandra)

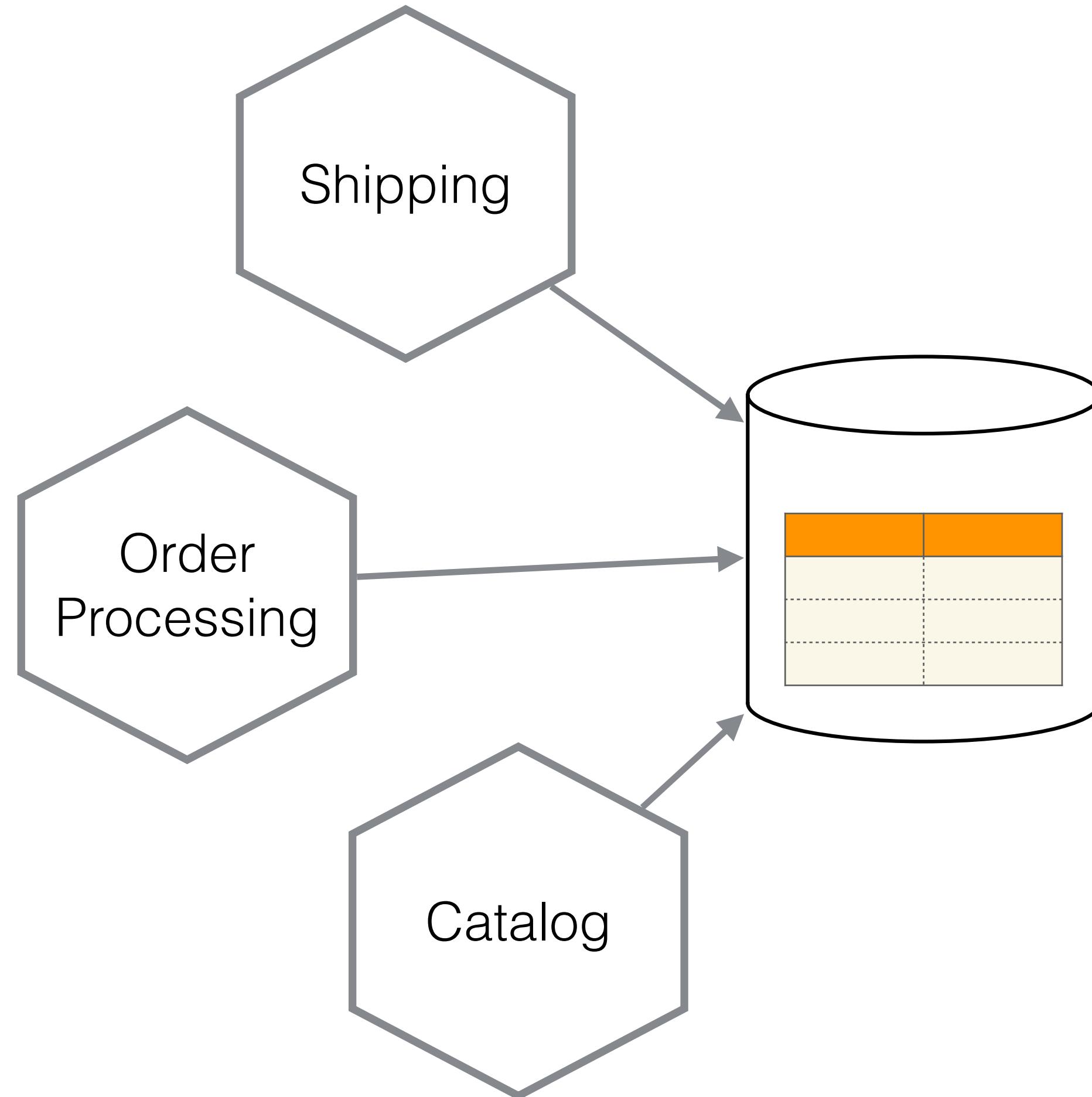
Graph (e.g. Neo)

PATTERN: SHARED DATABASE



Multiple services making use of the same database

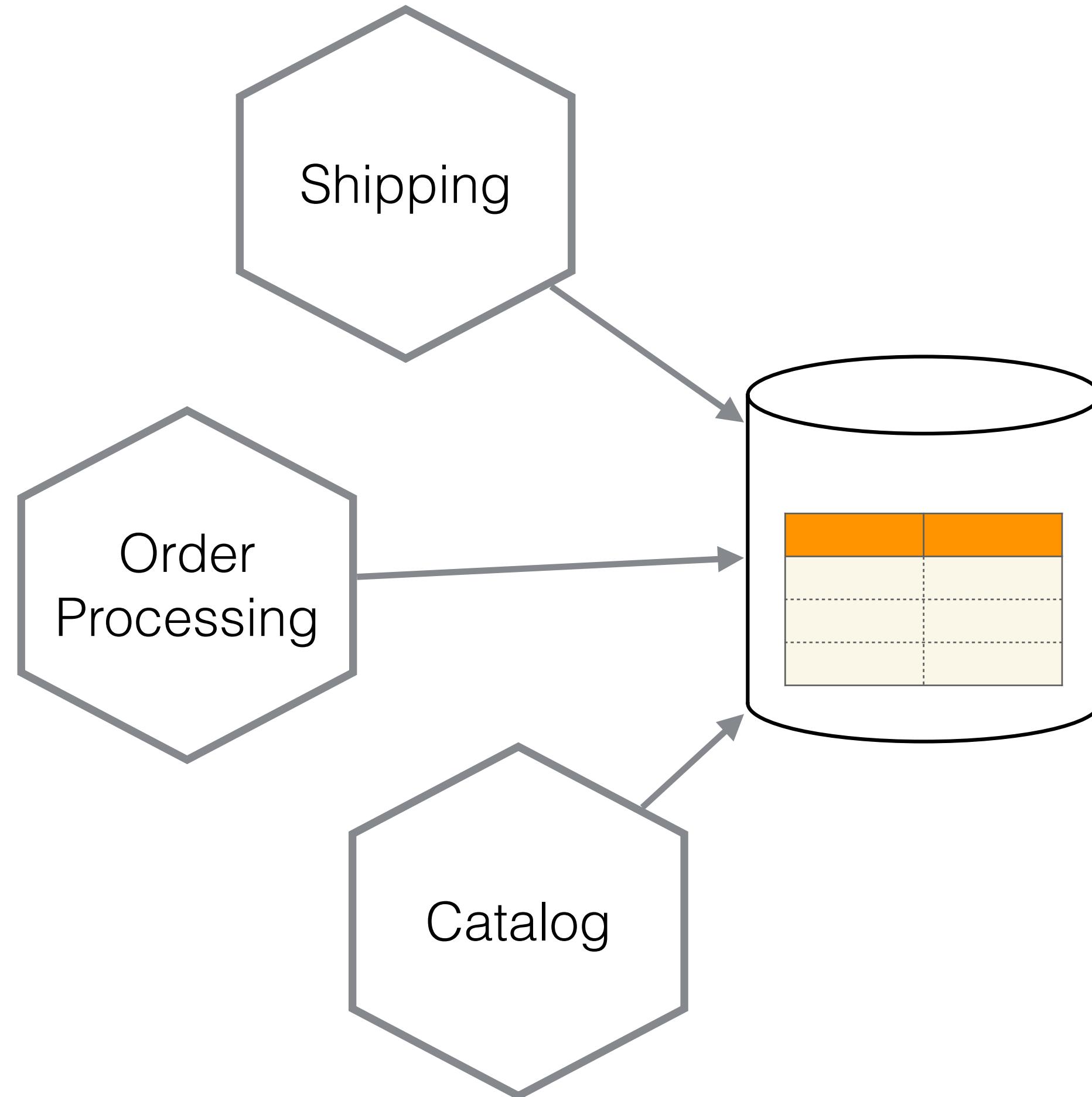
PATTERN: SHARED DATABASE



Multiple services making use of the same database

Changing the shared DB can be very disruptive

PATTERN: SHARED DATABASE

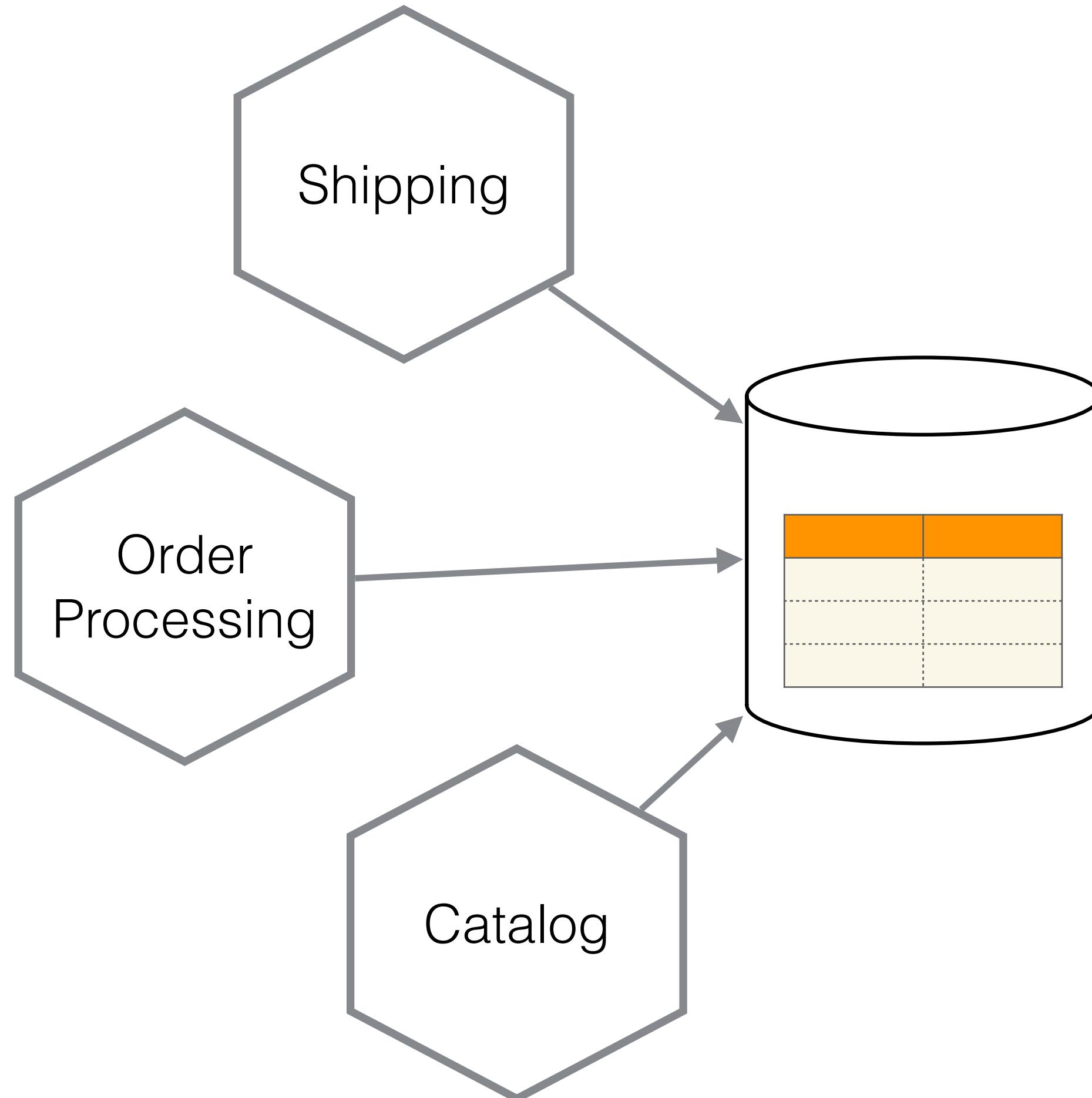


Multiple services making use of the same database

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?

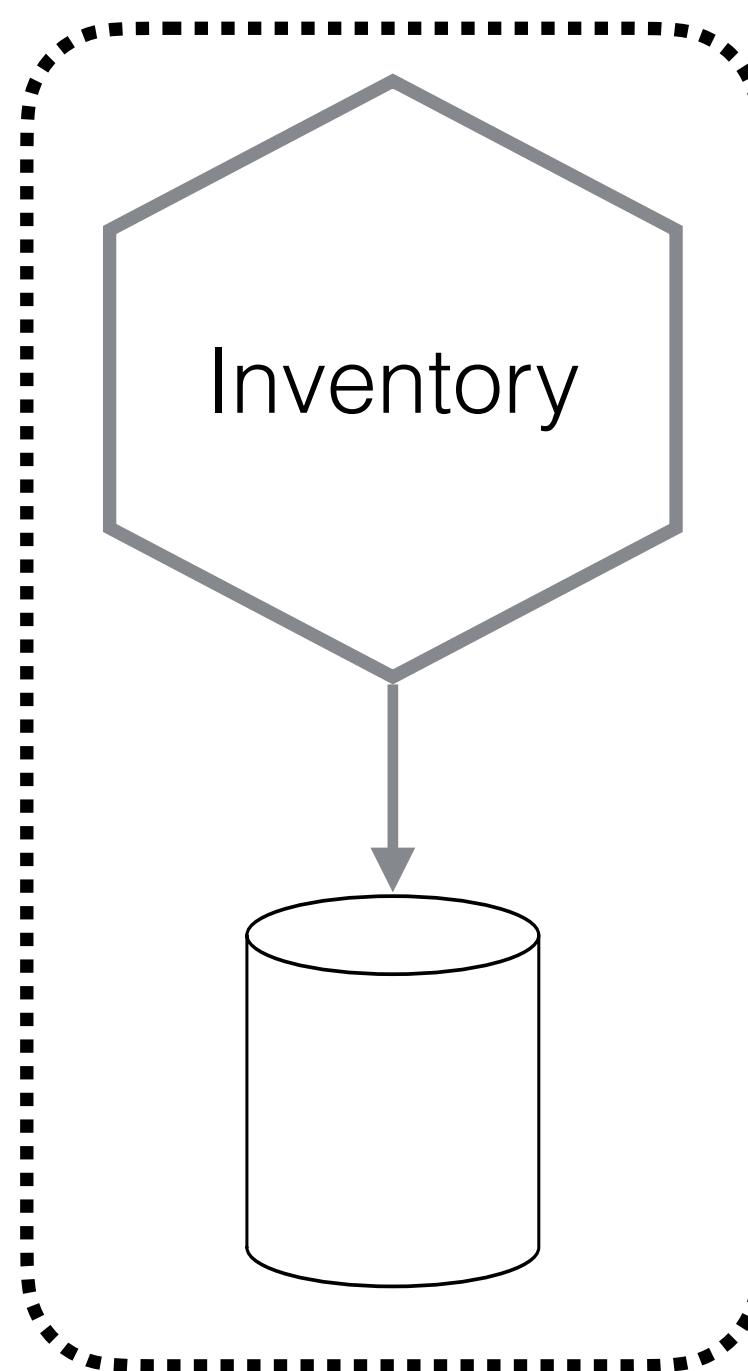
Static read-only reference data

WHEN TO USE A SHARED DATABASE?

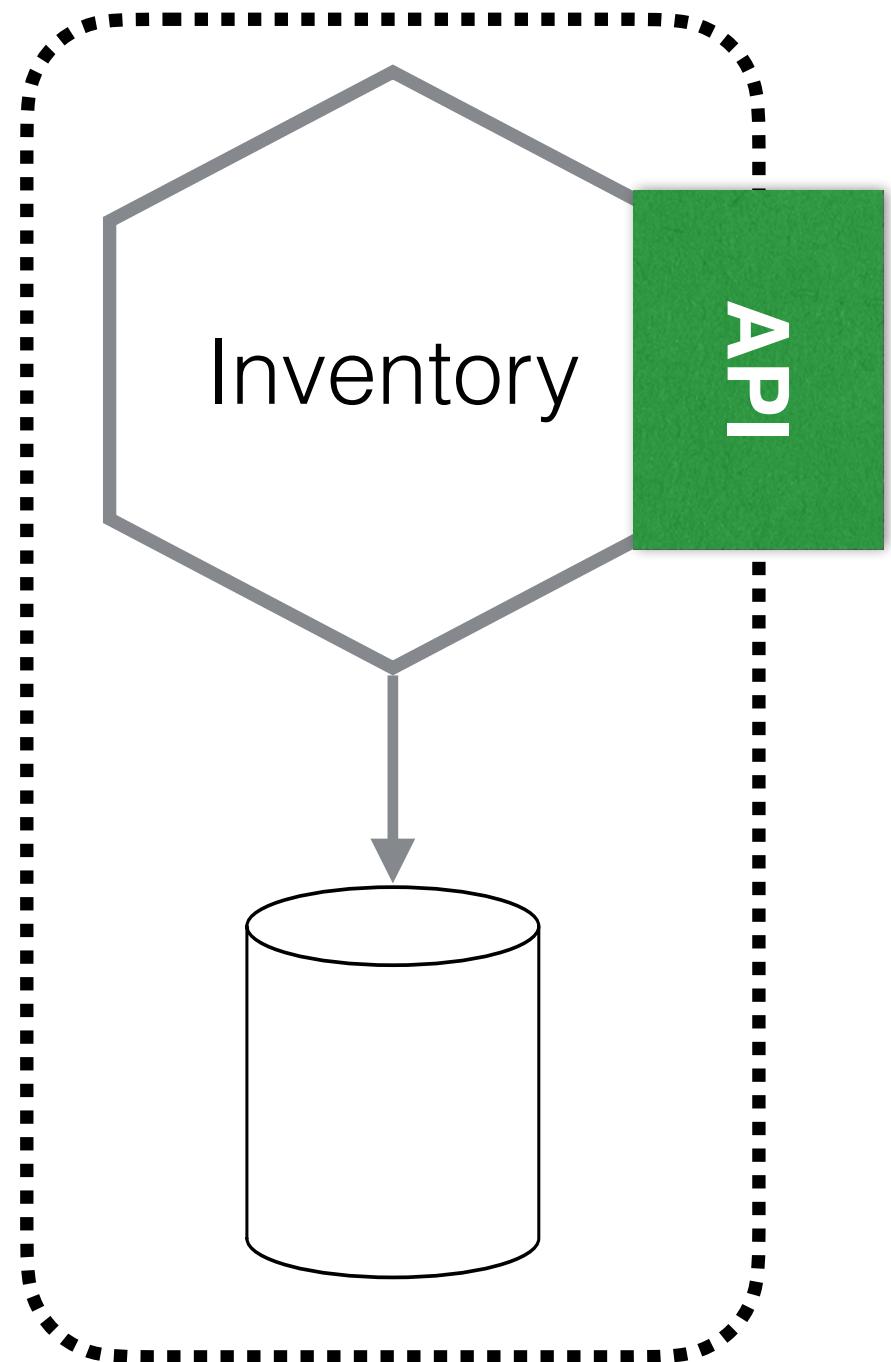
Static read-only reference data

As part of an incremental decomposition

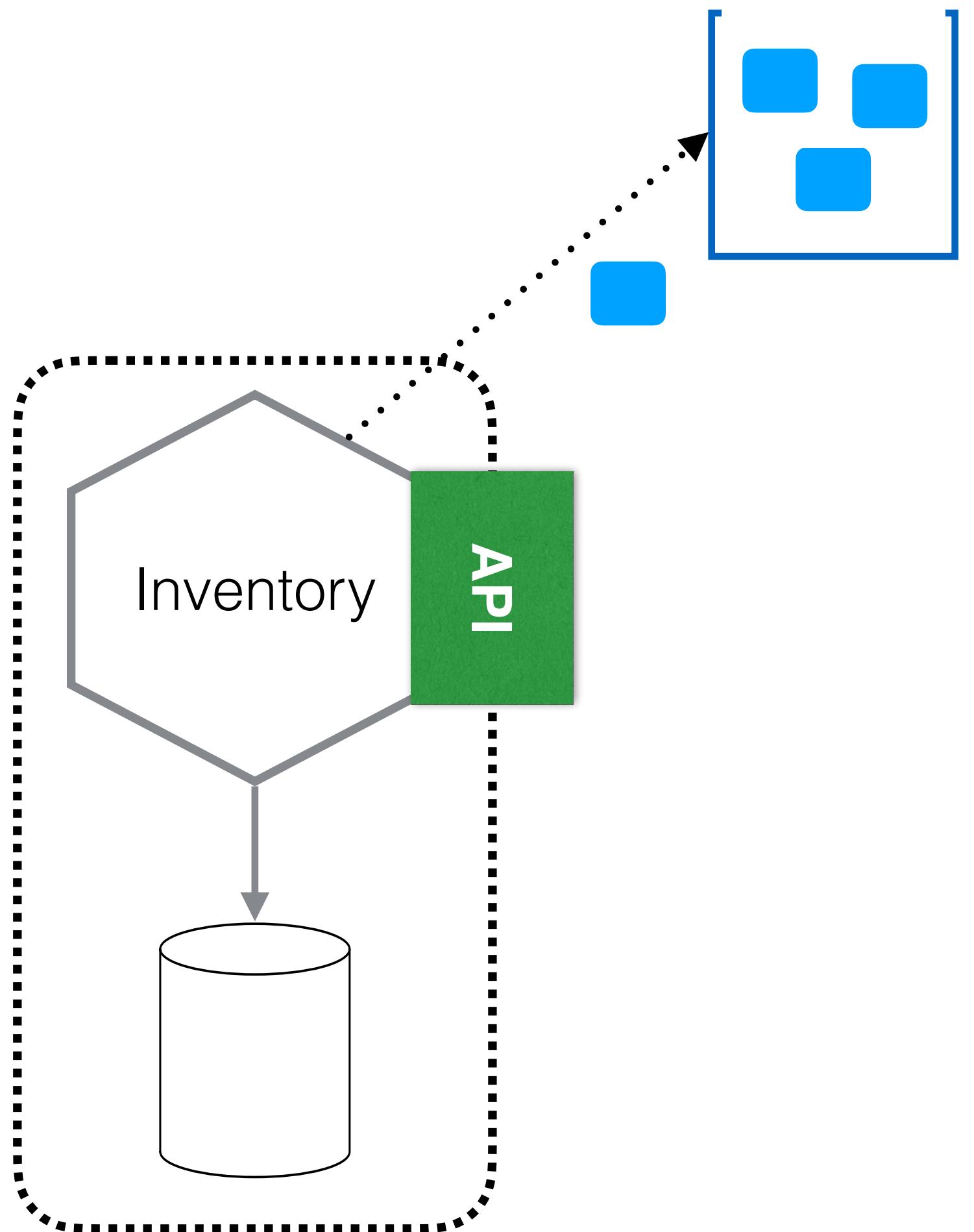
PATTERN: DATABASE AS AN ENDPOINT



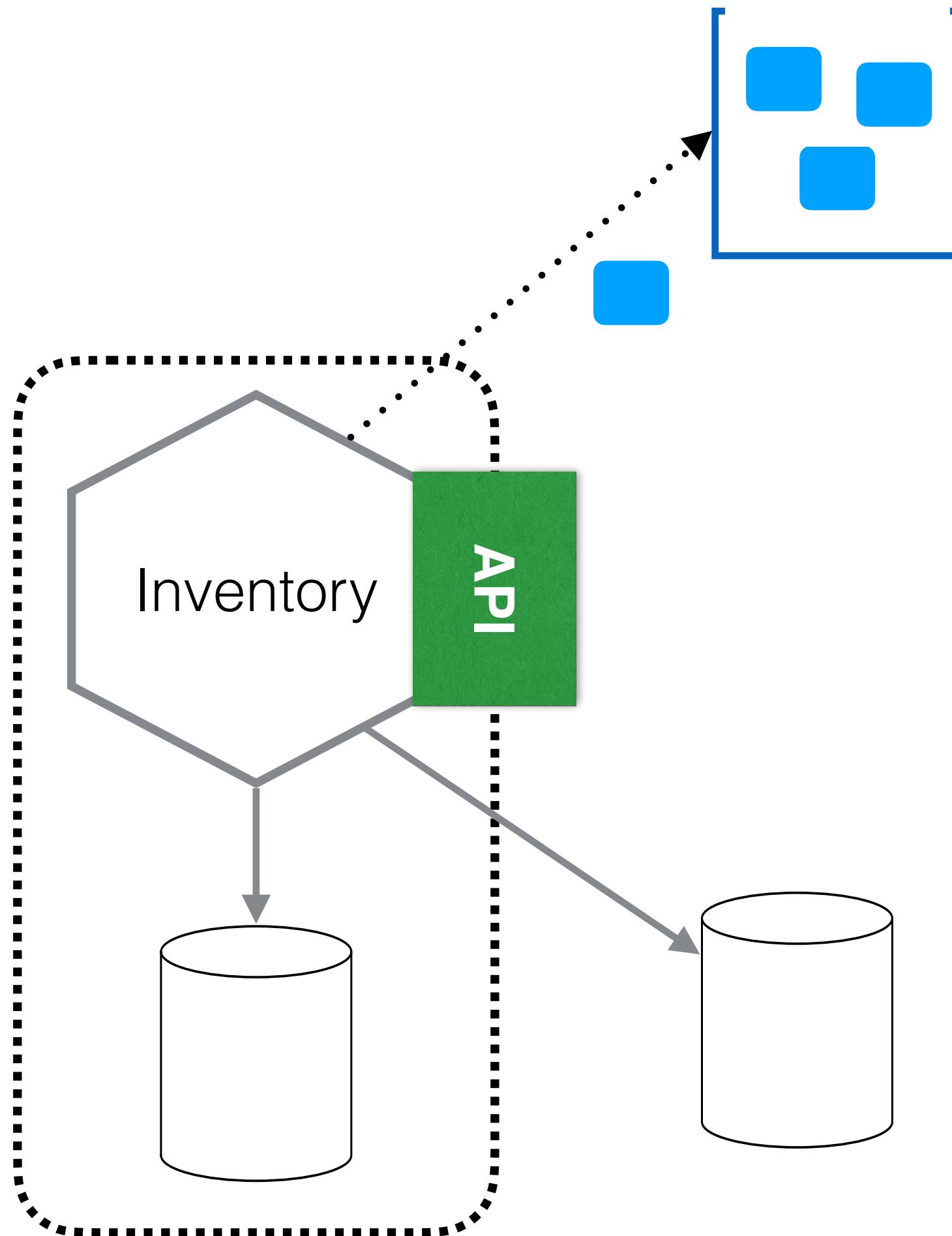
PATTERN: DATABASE AS AN ENDPOINT



PATTERN: DATABASE AS AN ENDPOINT

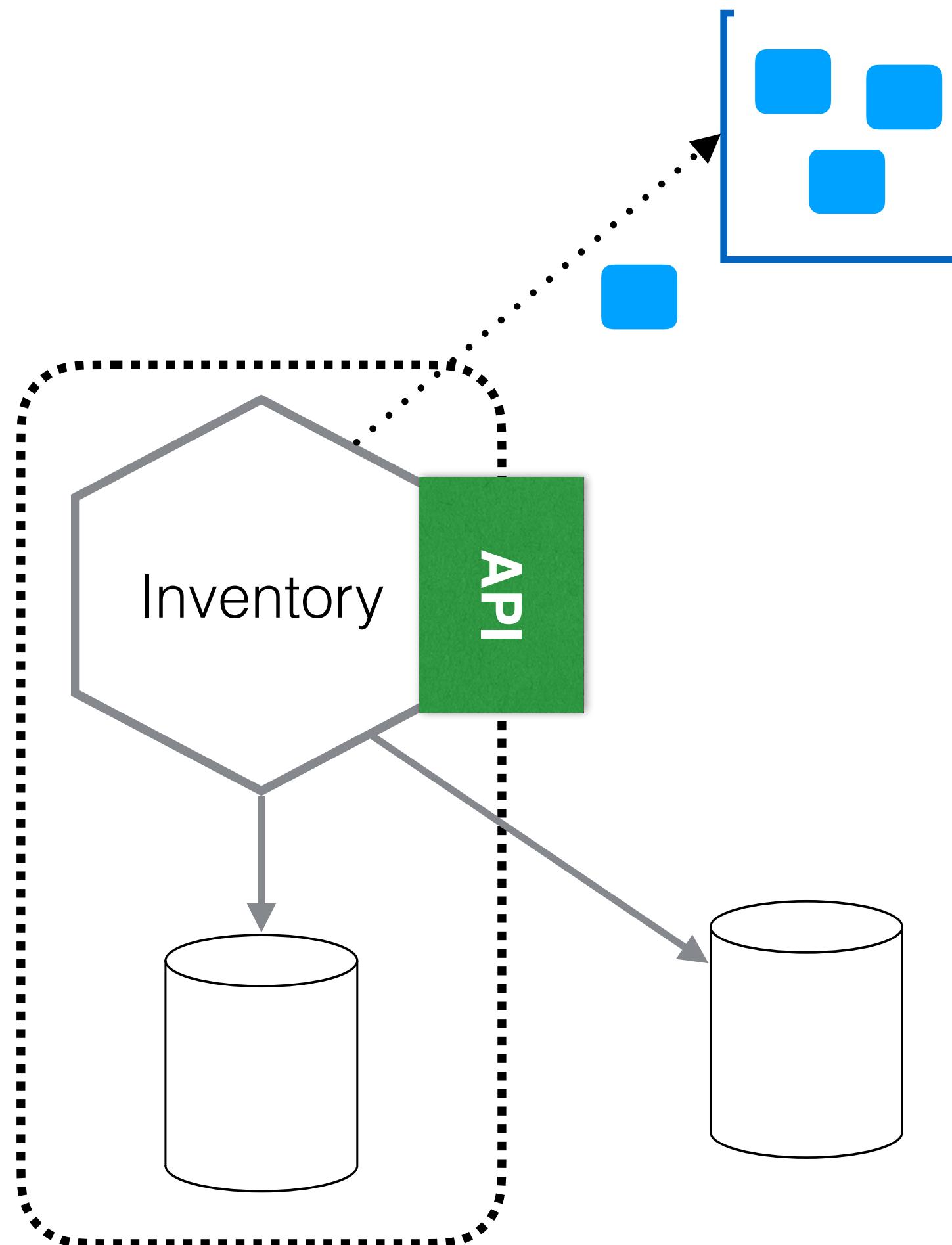


PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

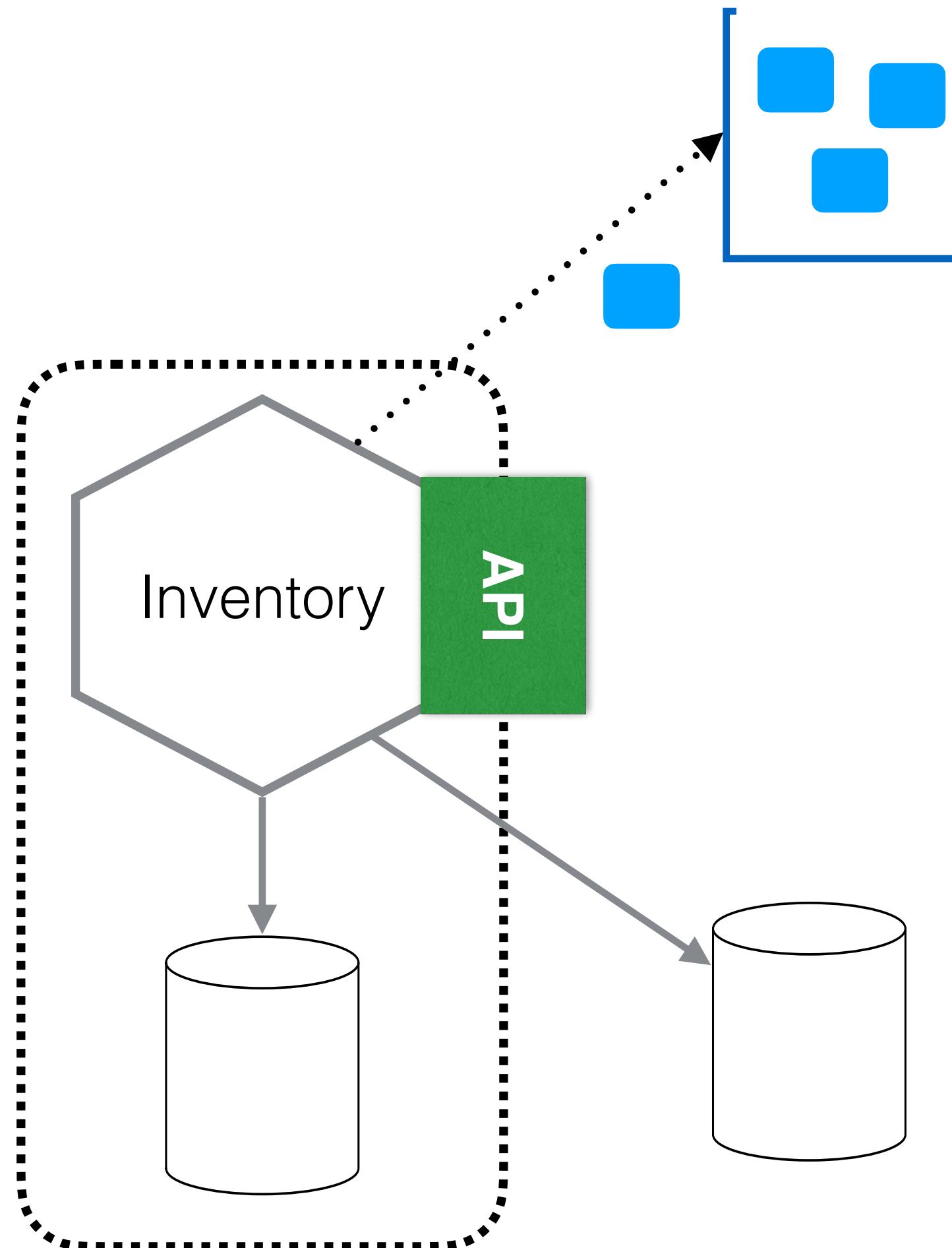
PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

Views are one way to implement this pattern

PATTERN: DATABASE AS AN ENDPOINT

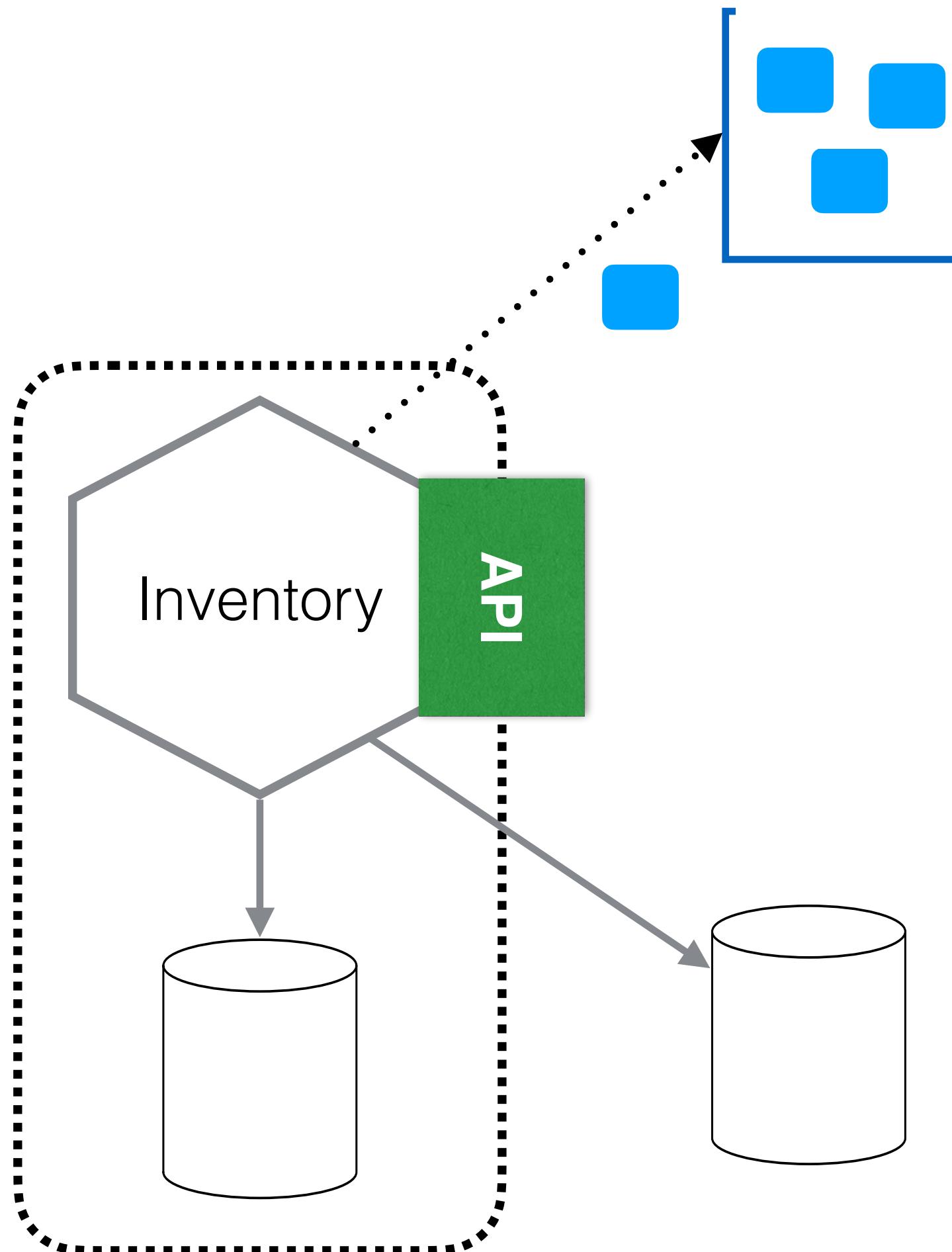


Expose a full DB as an endpoint

Views are one way to implement this pattern

Typically read-only

PATTERN: DATABASE AS AN ENDPOINT



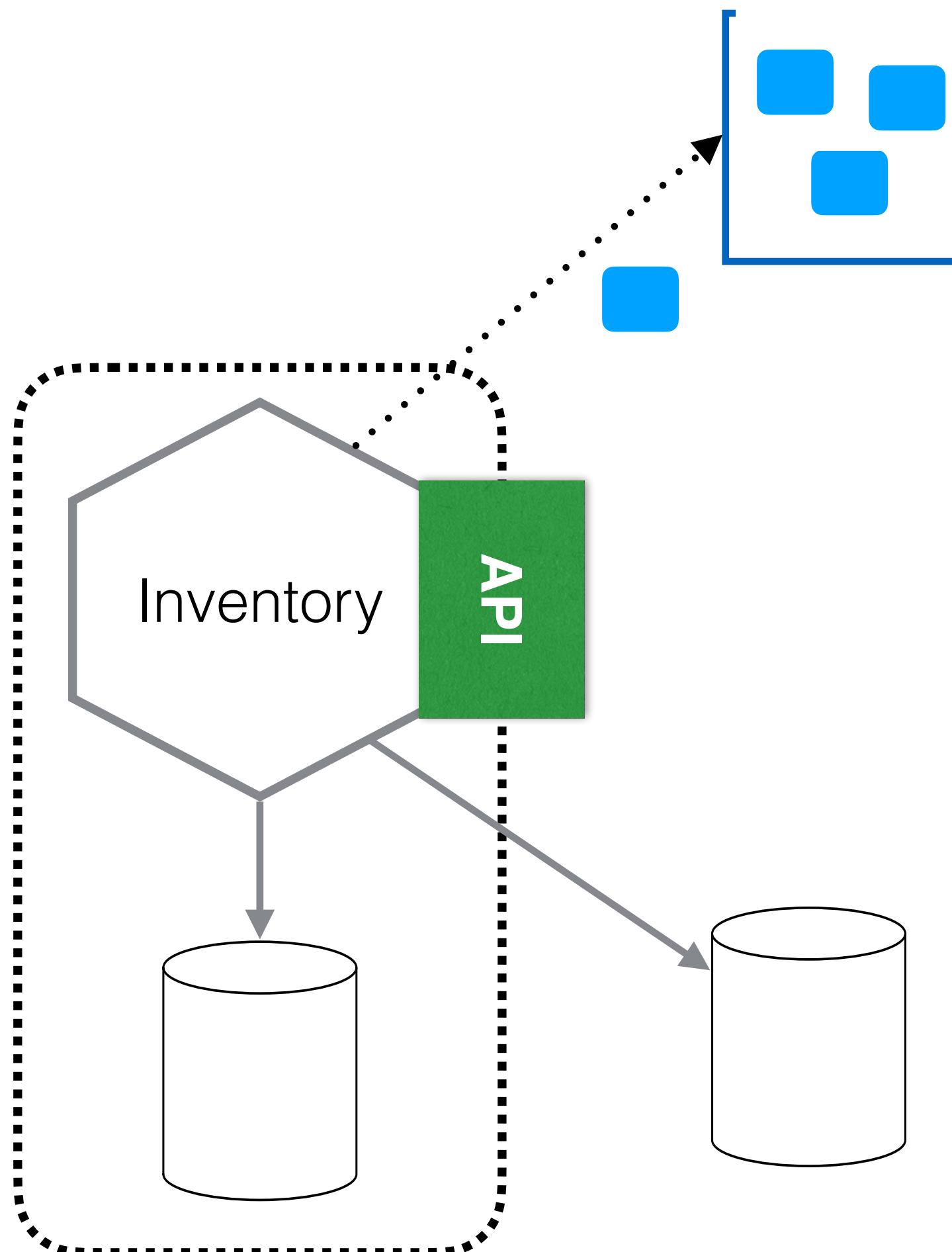
Expose a full DB as an endpoint

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

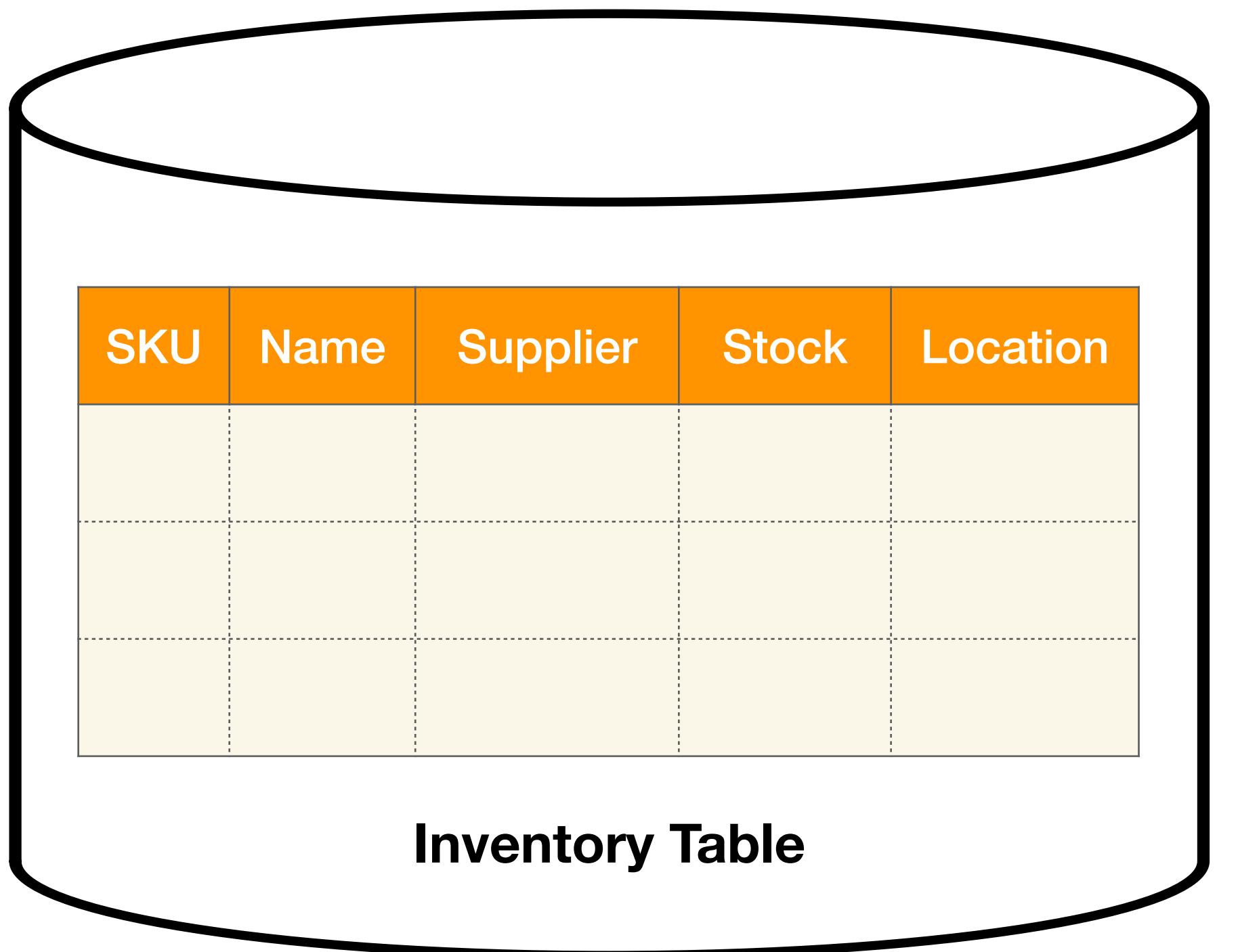
Views are one way to implement this pattern

Typically read-only

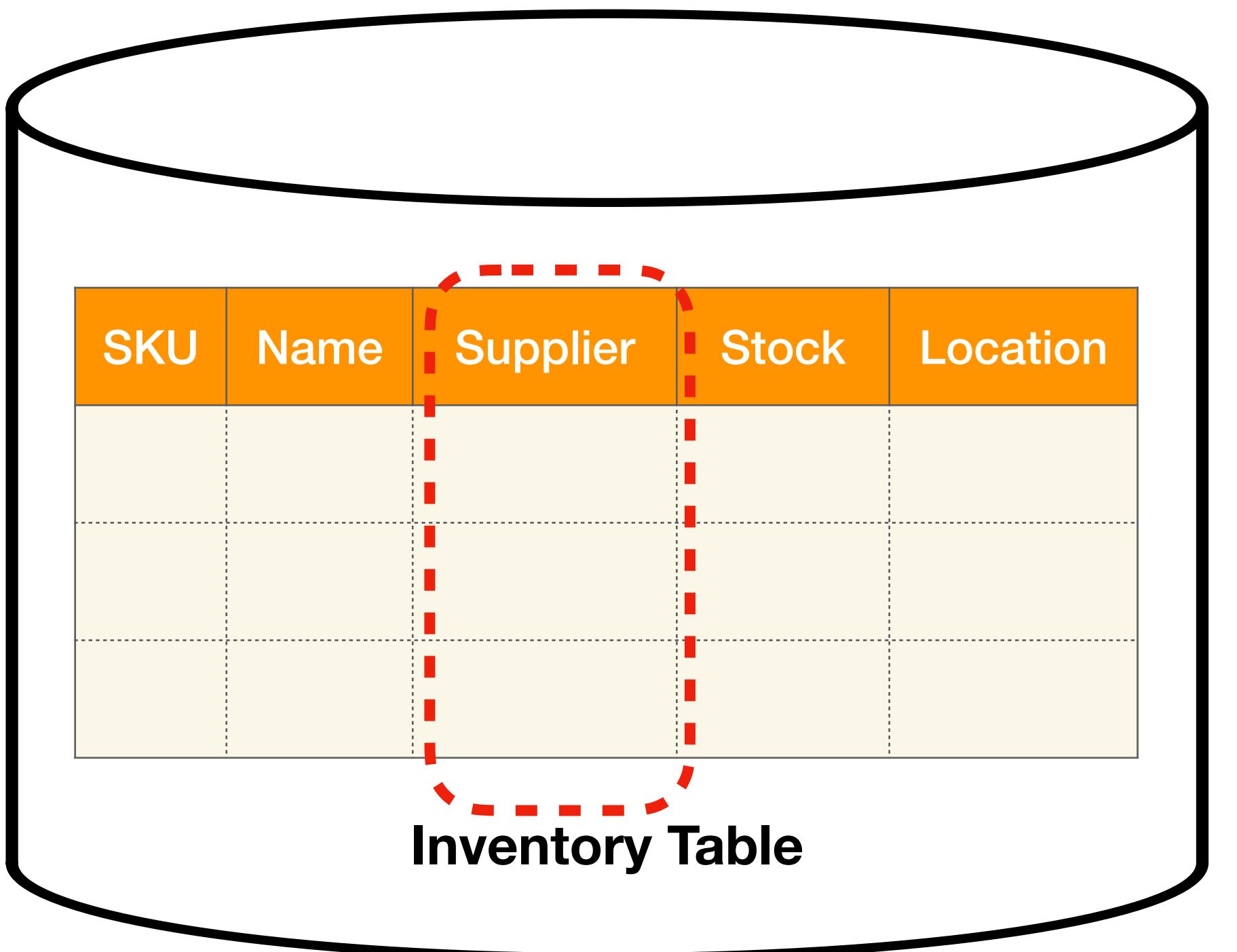
Really useful for legacy applications

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

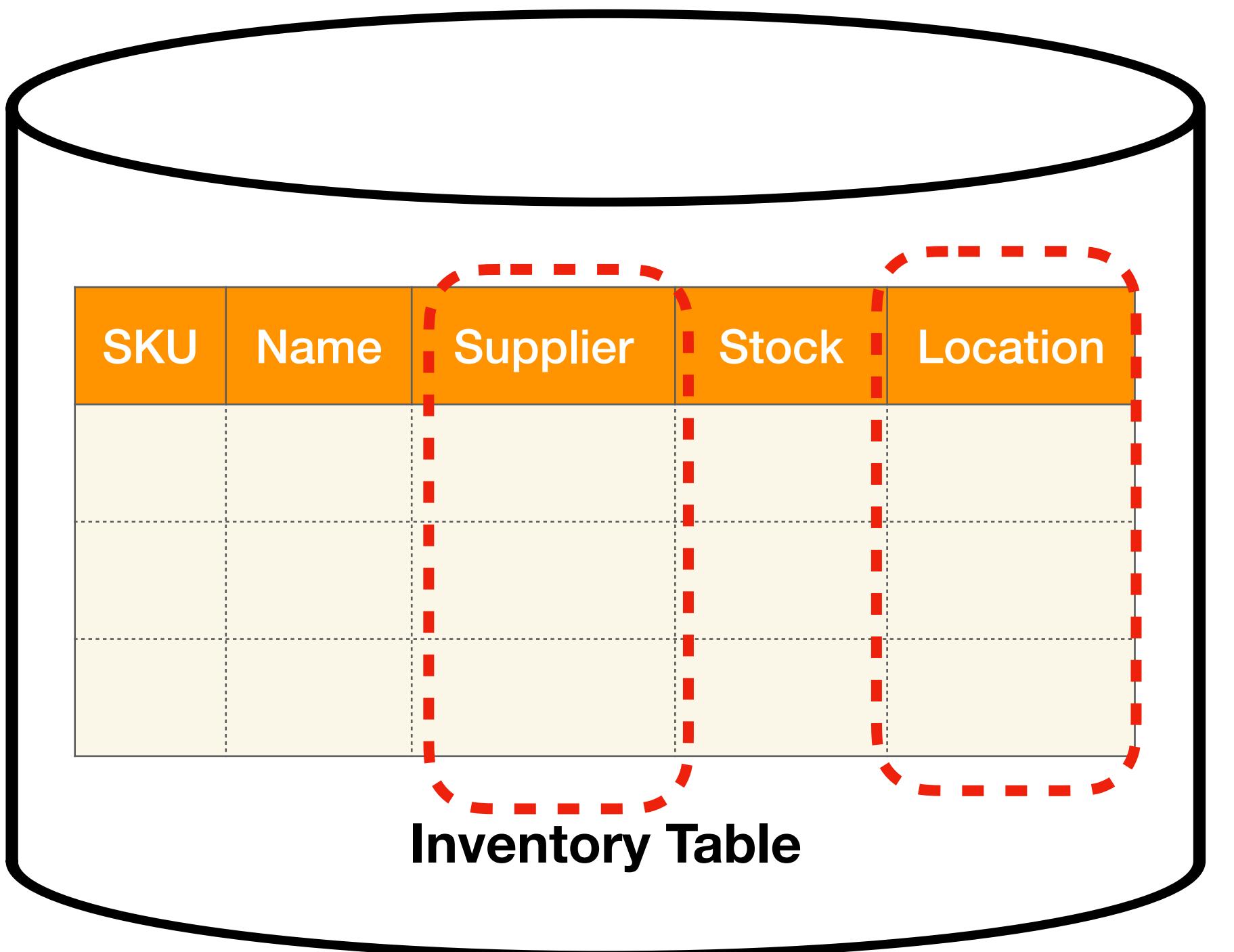
PATTERN: DATABASE VIEWS



PATTERN: DATABASE VIEWS

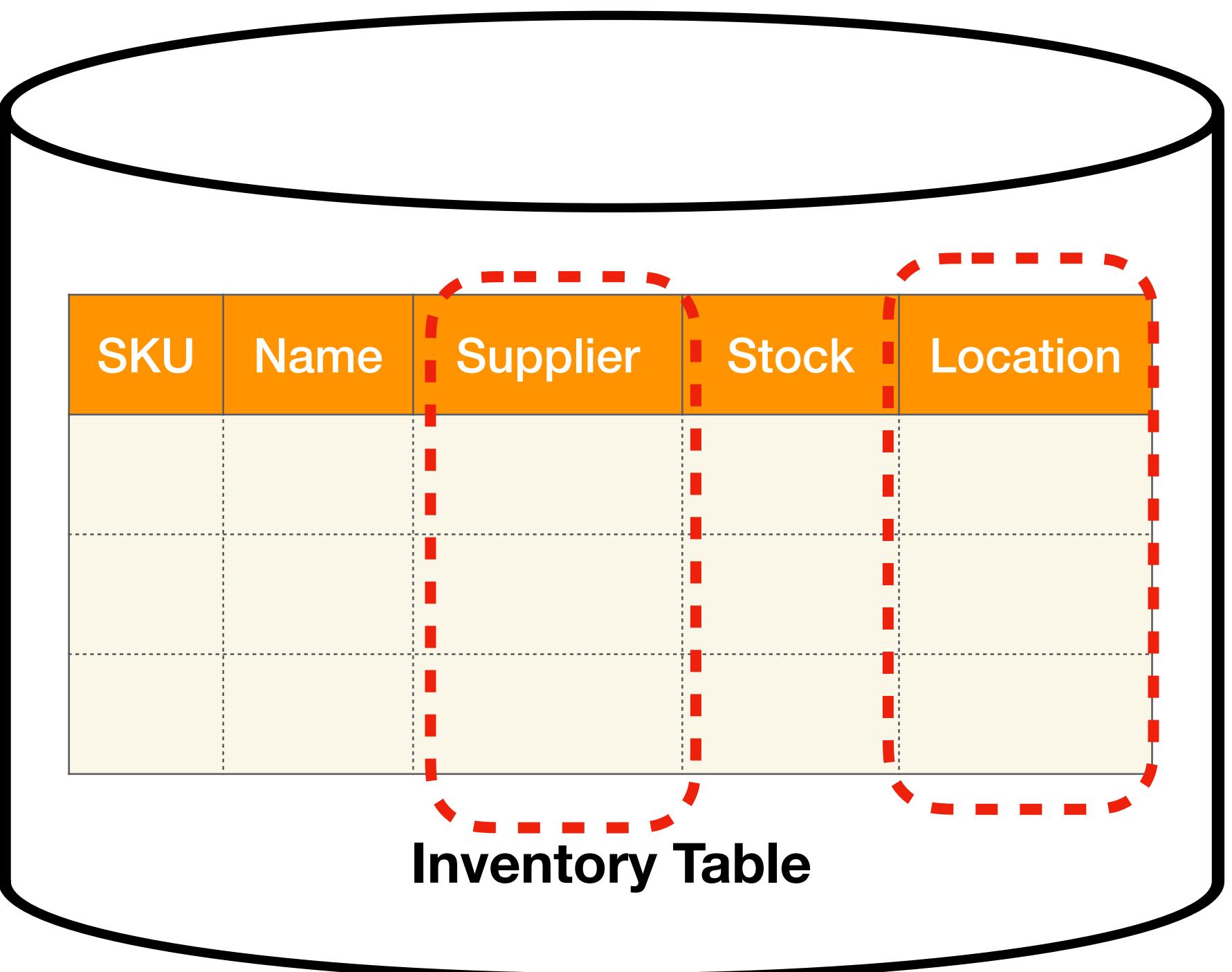


PATTERN: DATABASE VIEWS



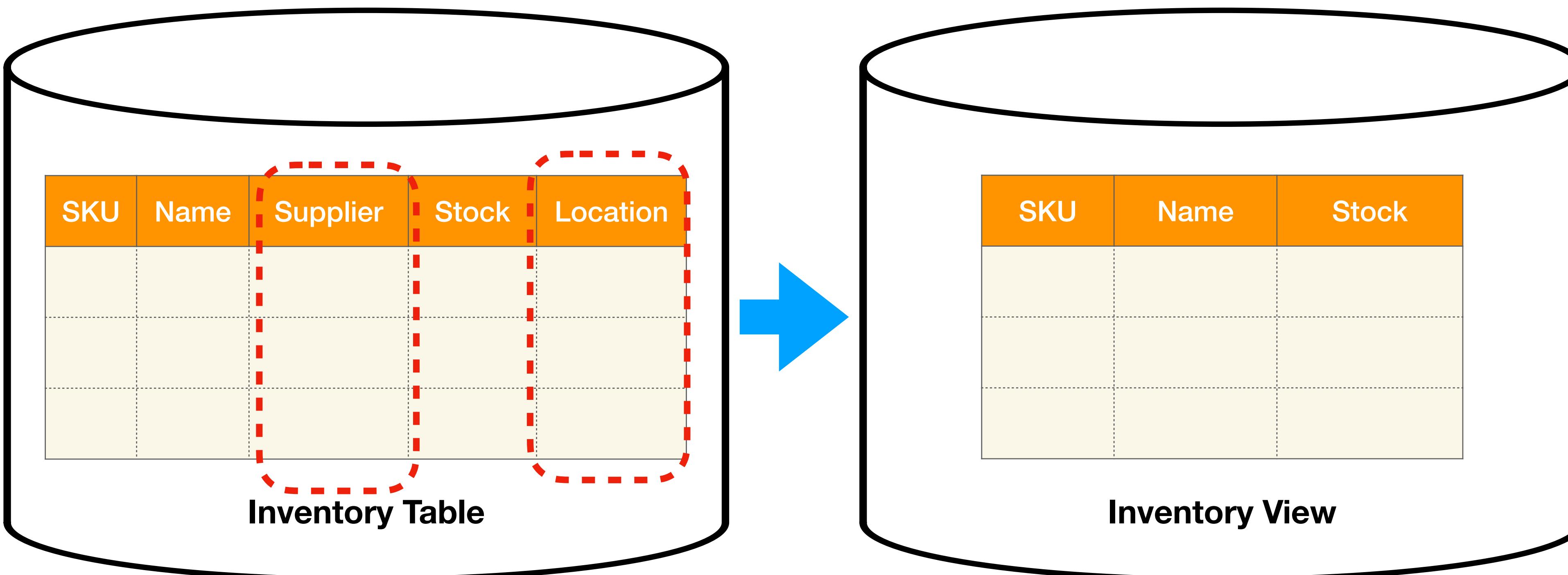
PATTERN: DATABASE VIEWS

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



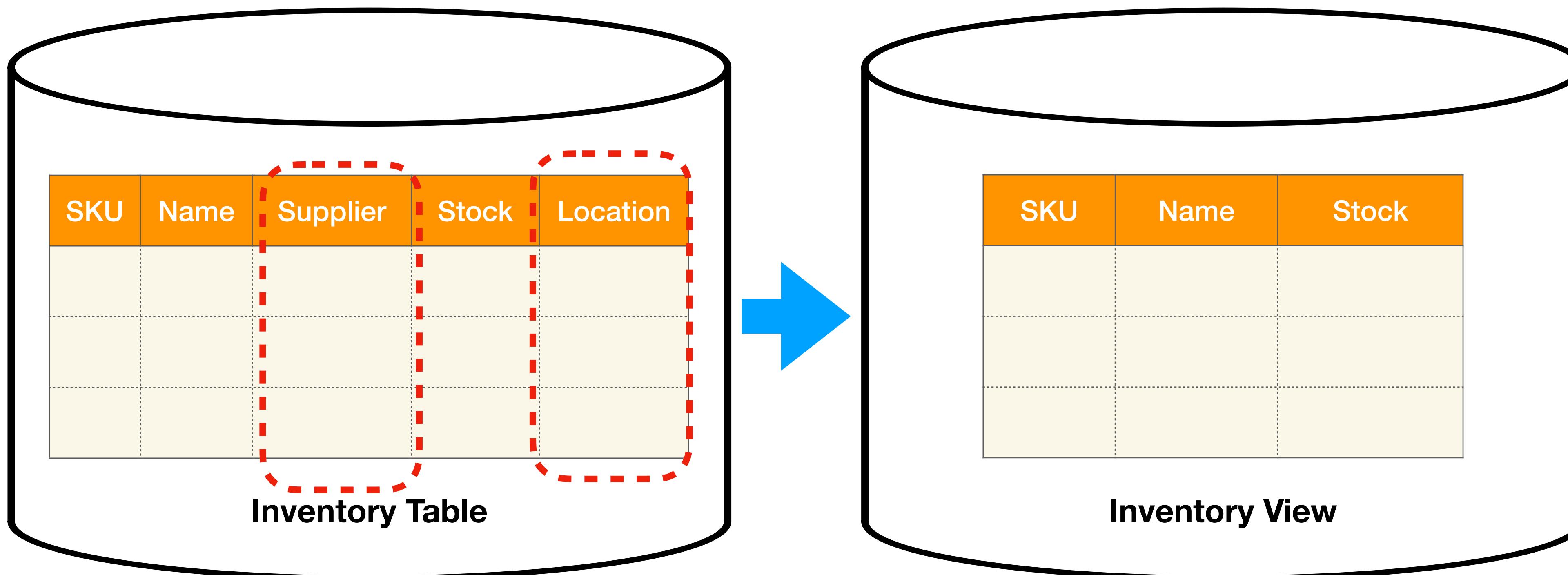
PATTERN: DATABASE VIEWS

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



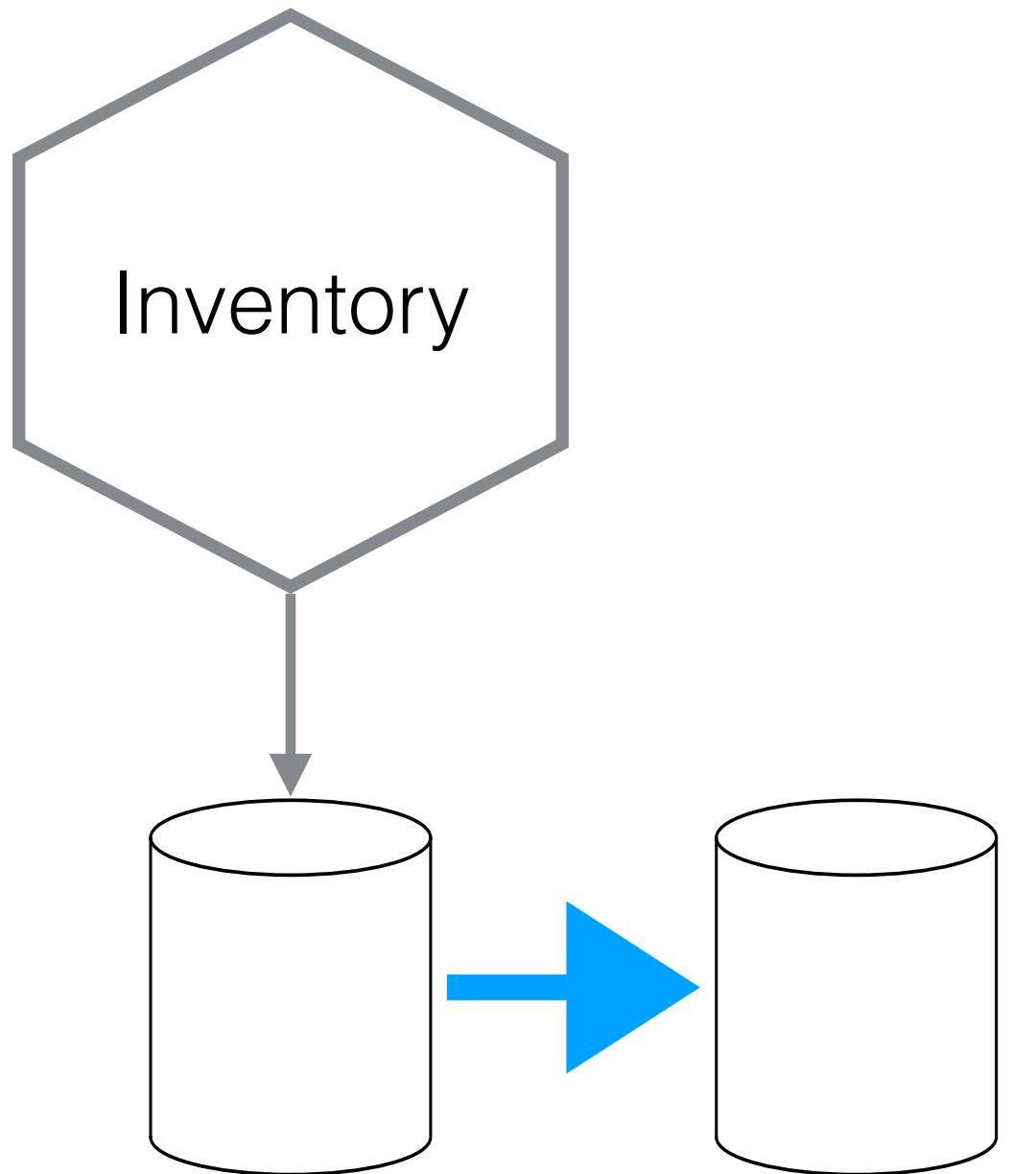
PATTERN: DATABASE VIEWS

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

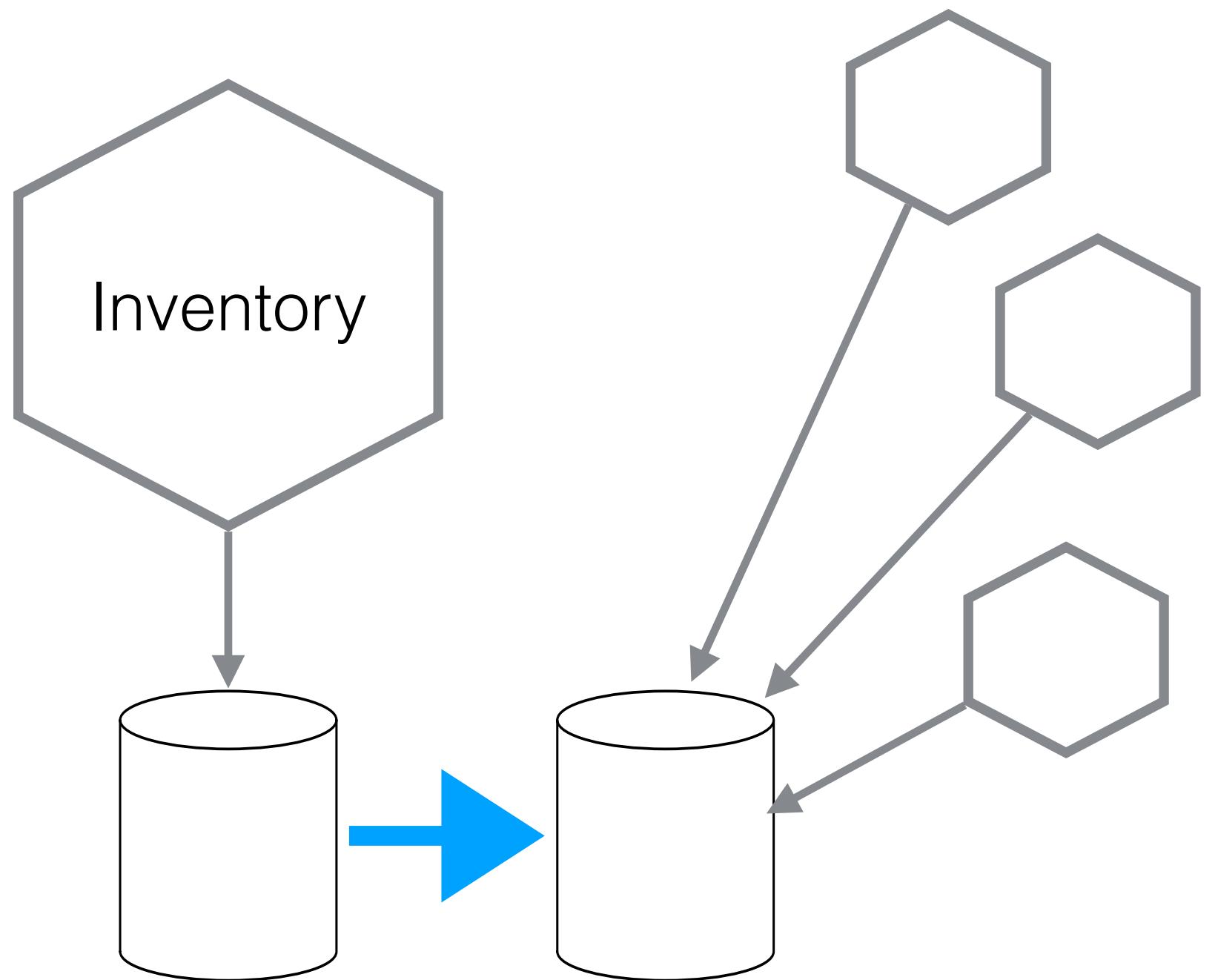


Allows for limited information hiding for direct DB access

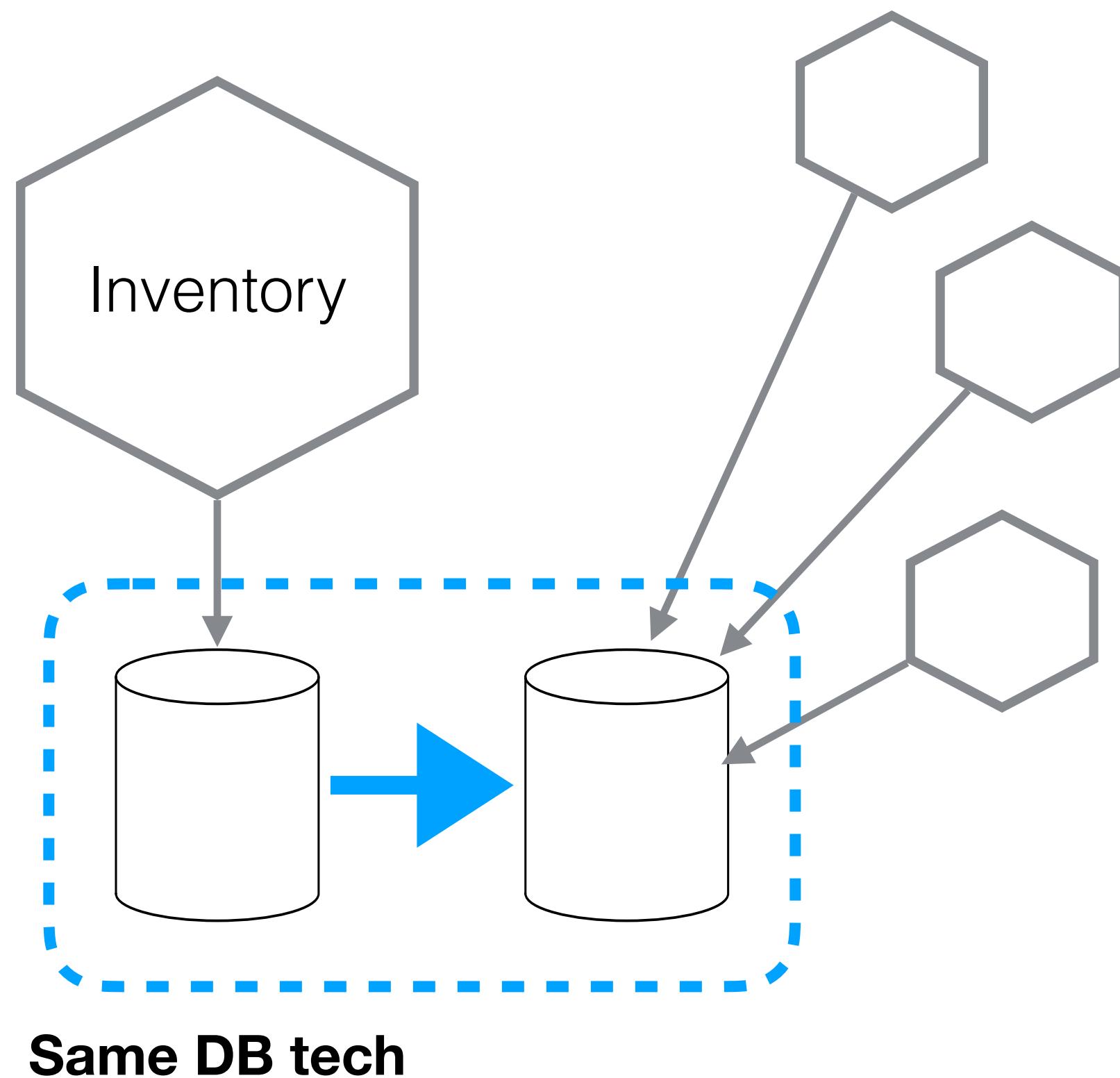
DATABASE VIEW - PROS AND CONS



DATABASE VIEW - PROS AND CONS

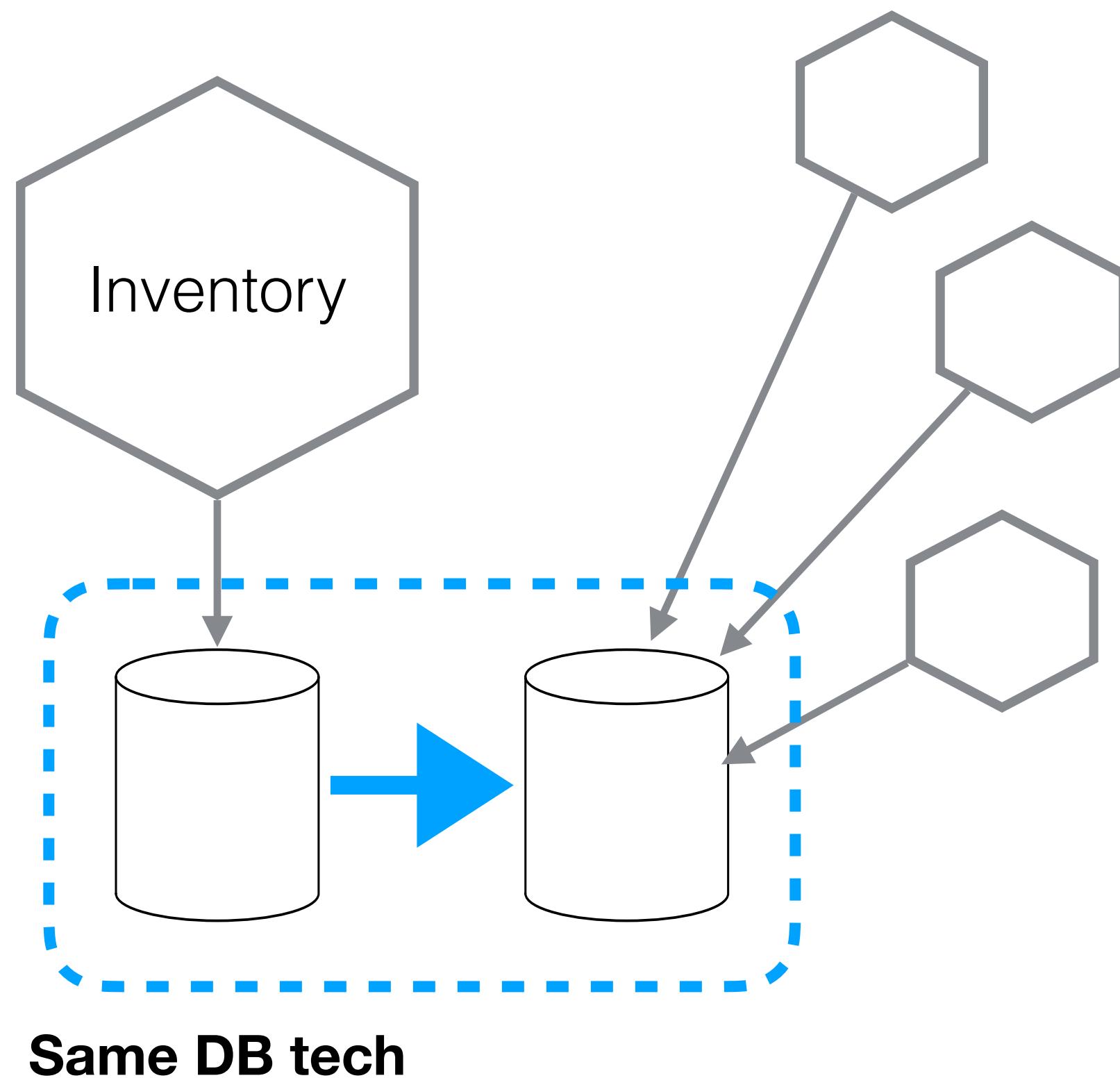


DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation

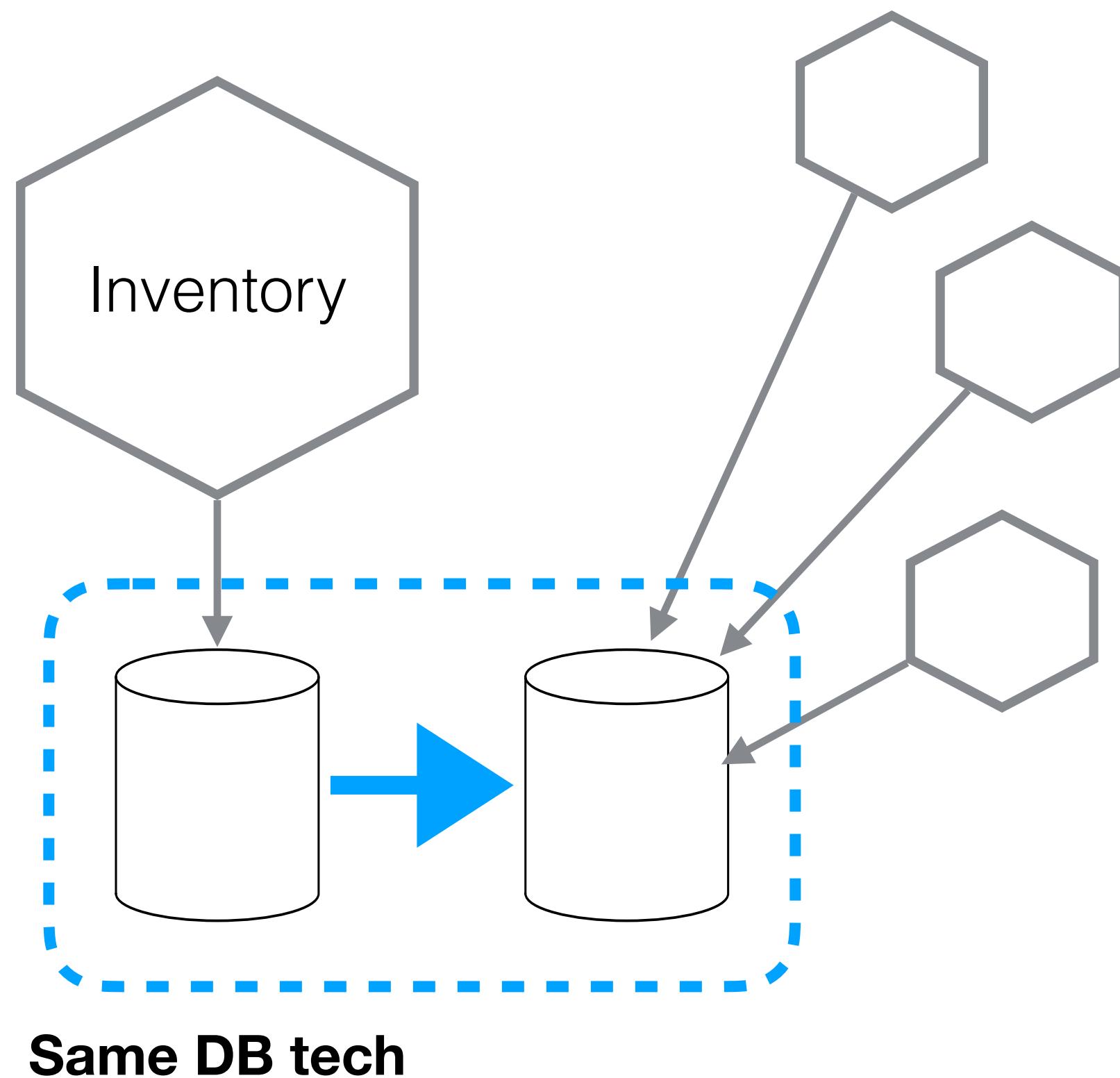
DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation

Doesn't solve writes

DATABASE VIEW - PROS AND CONS

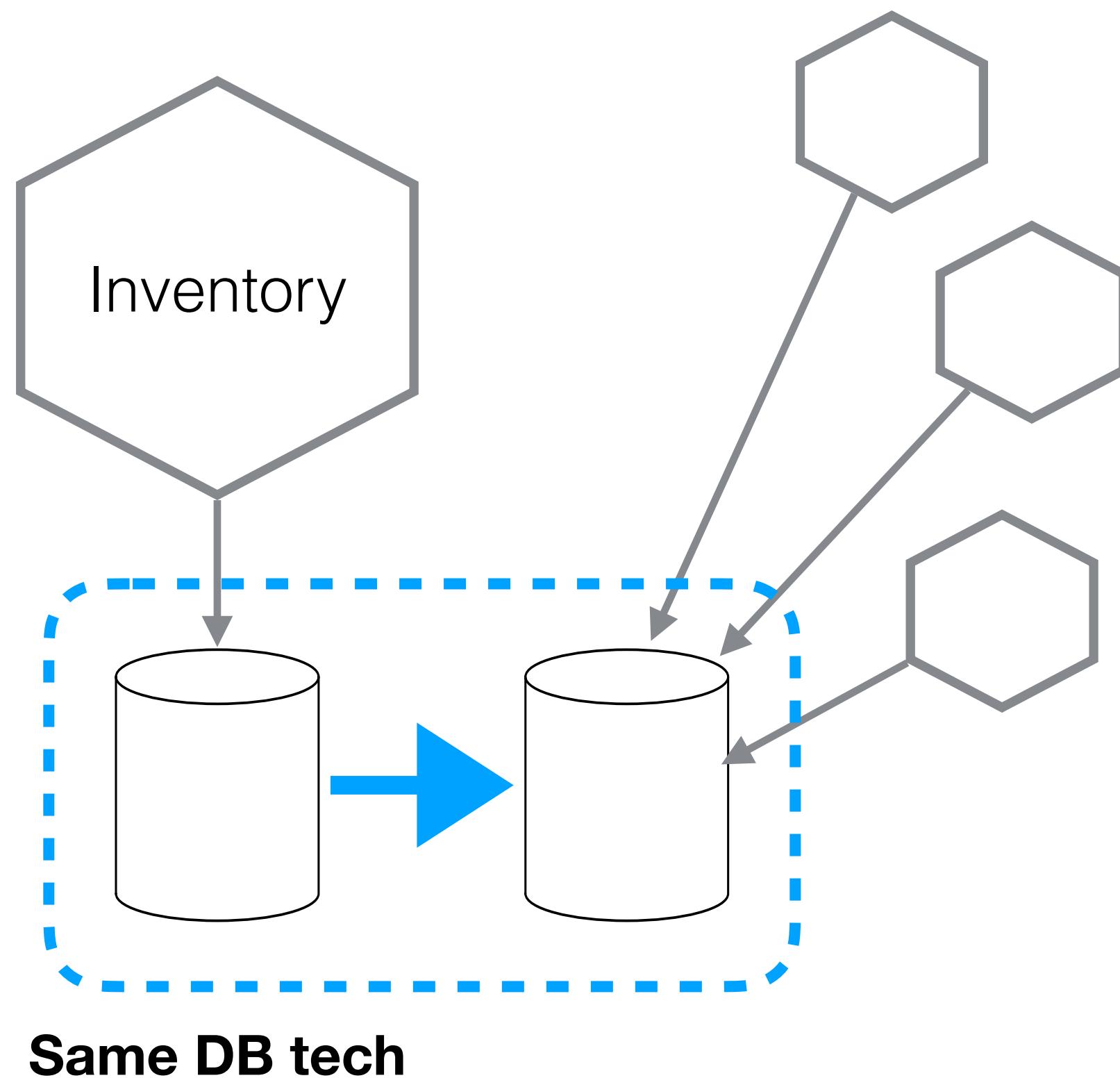


Tied to an underlying implementation

Doesn't solve writes

View mapping needs to be maintained if the source DB changes

DATABASE VIEW - PROS AND CONS



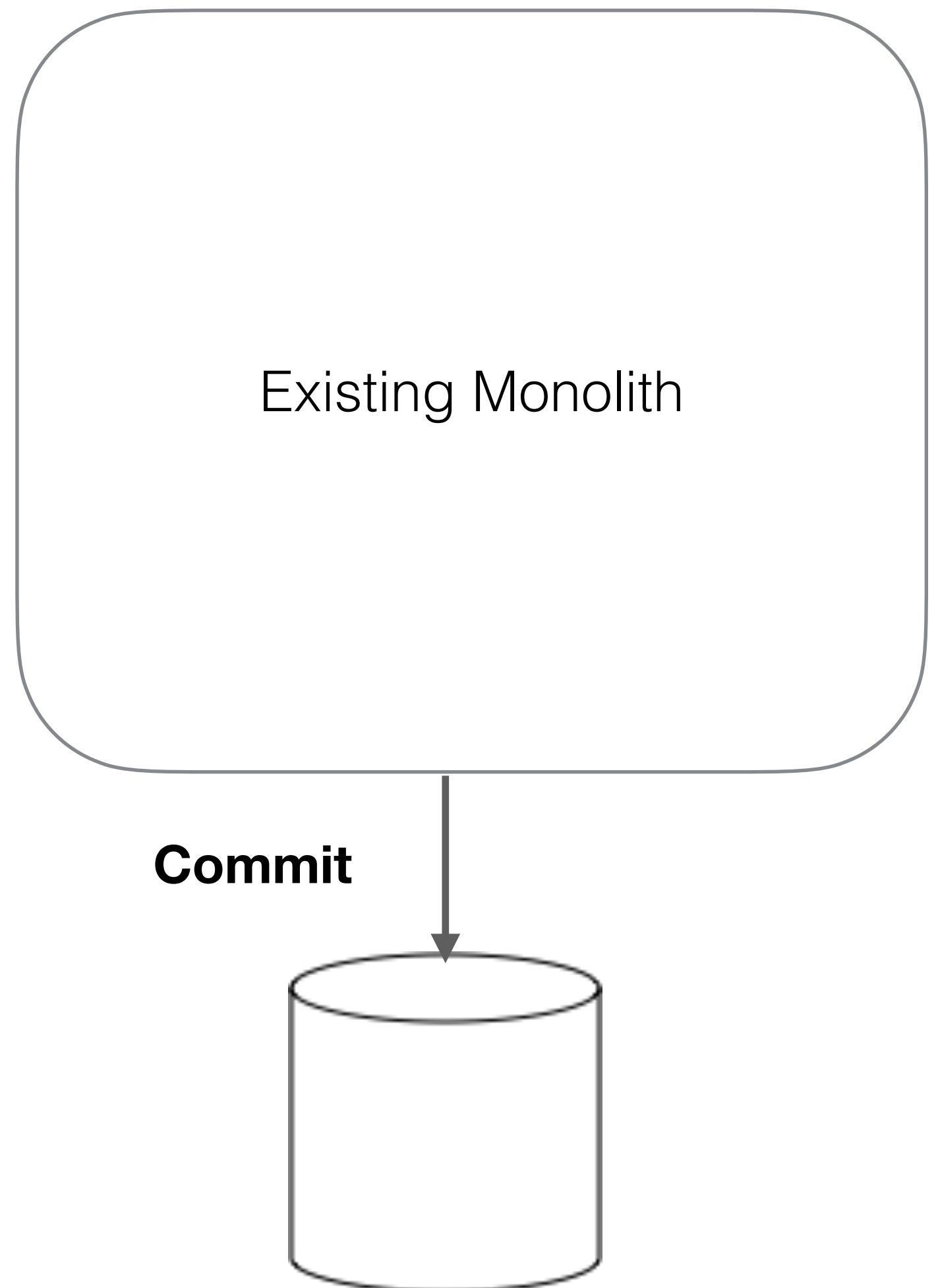
Tied to an underlying implementation

Doesn't solve writes

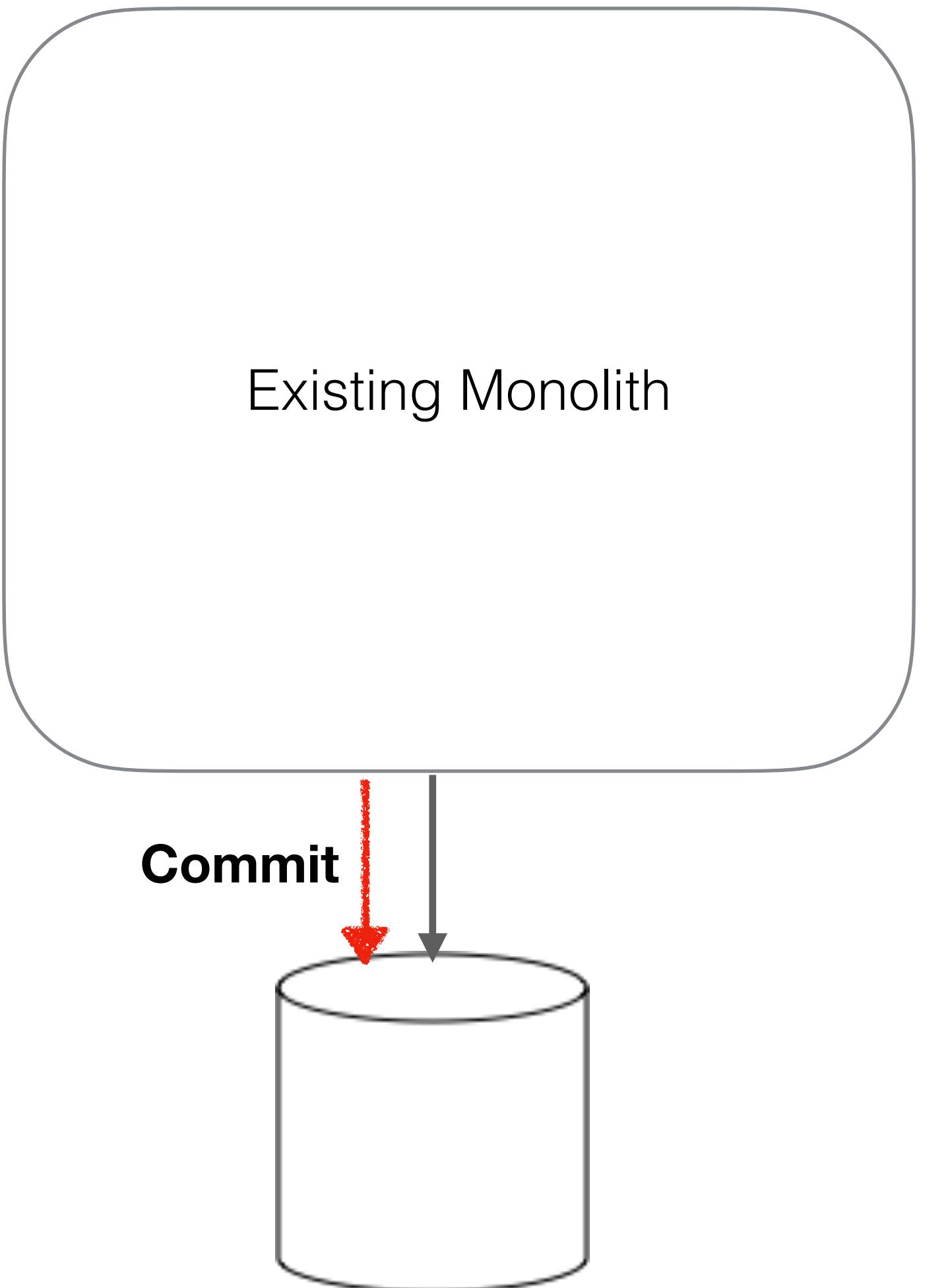
View mapping needs to be maintained if the source DB changes

Better than direct DB access

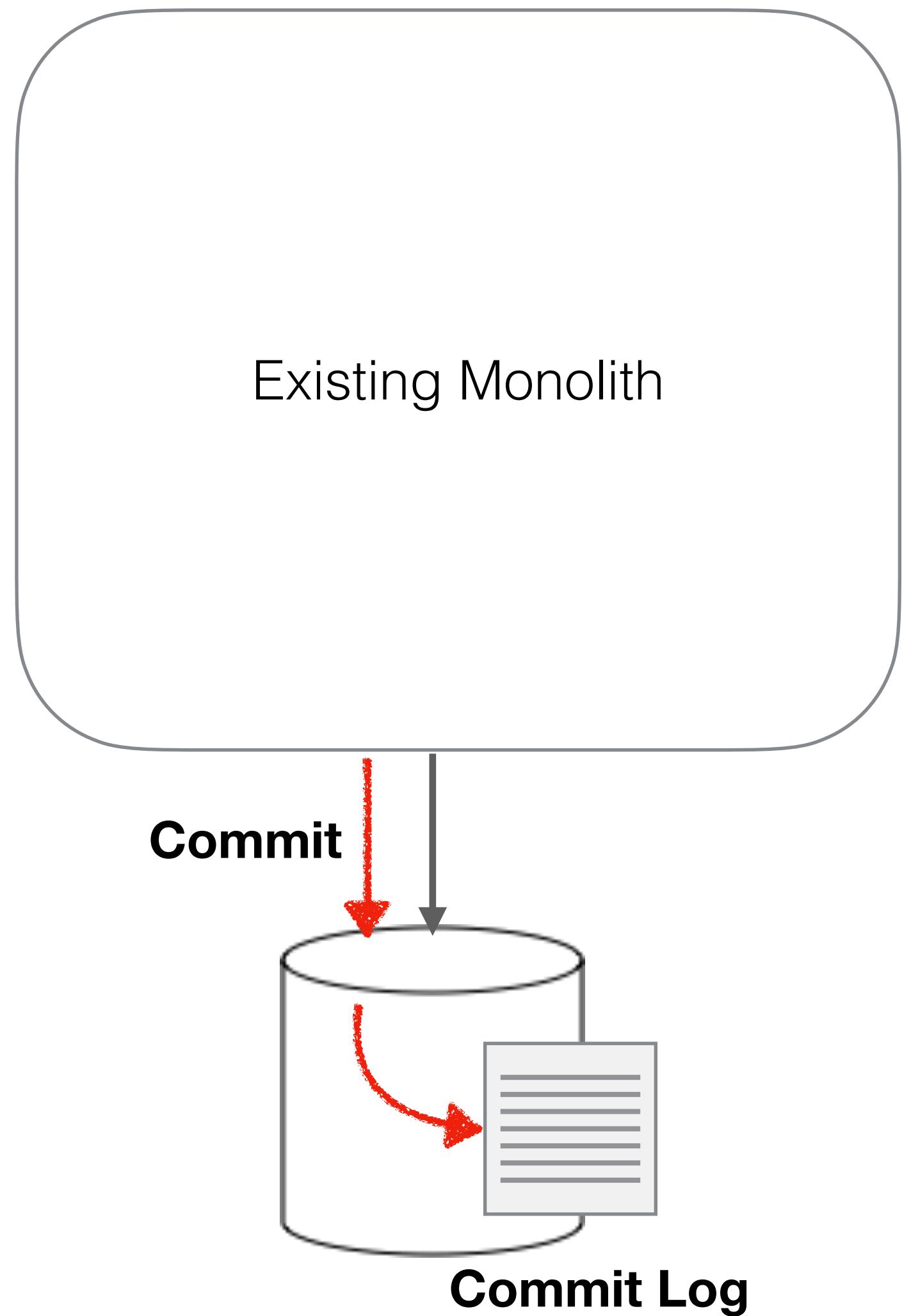
PATTERN: CHANGE DATA CAPTURE



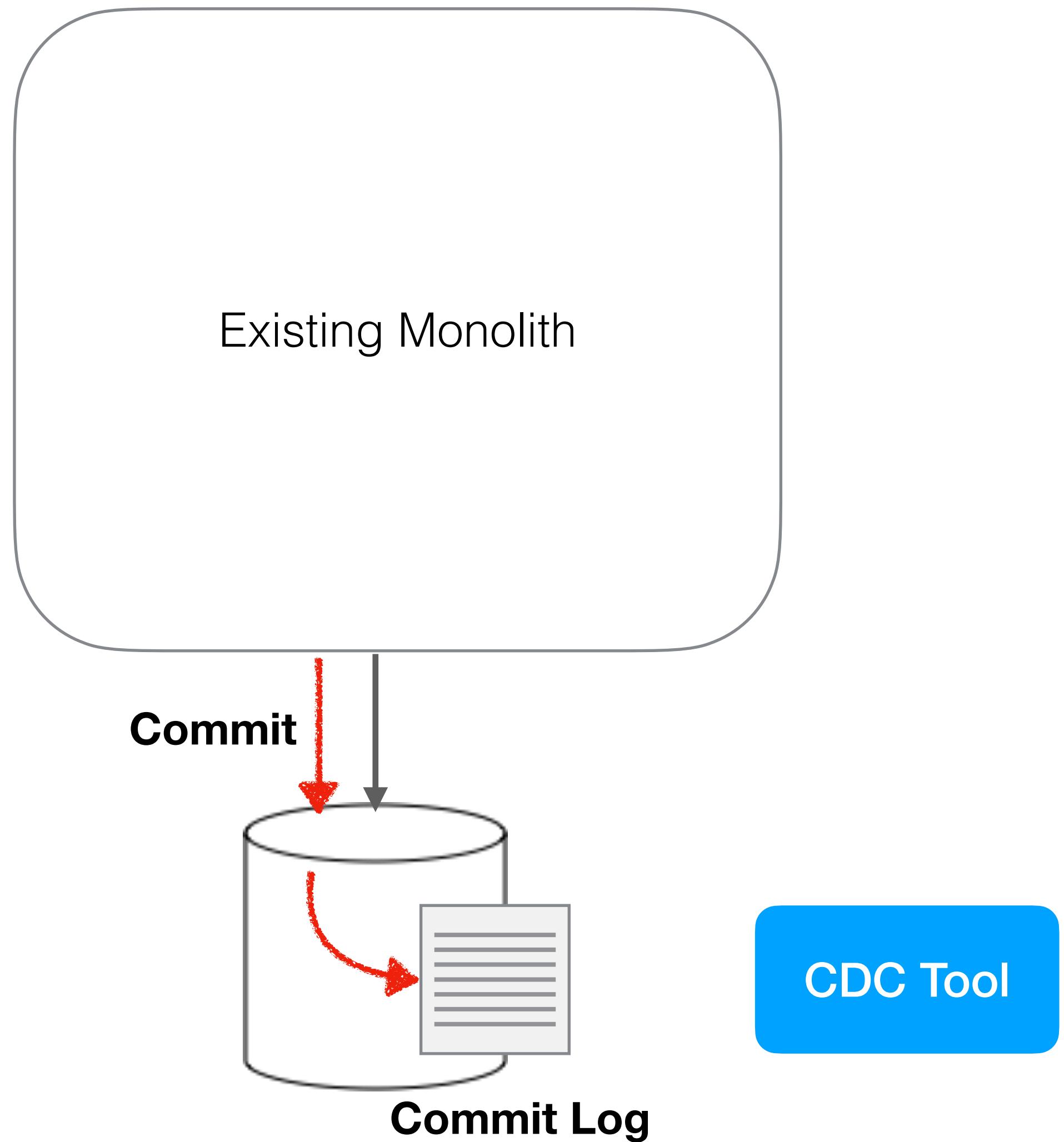
PATTERN: CHANGE DATA CAPTURE



PATTERN: CHANGE DATA CAPTURE

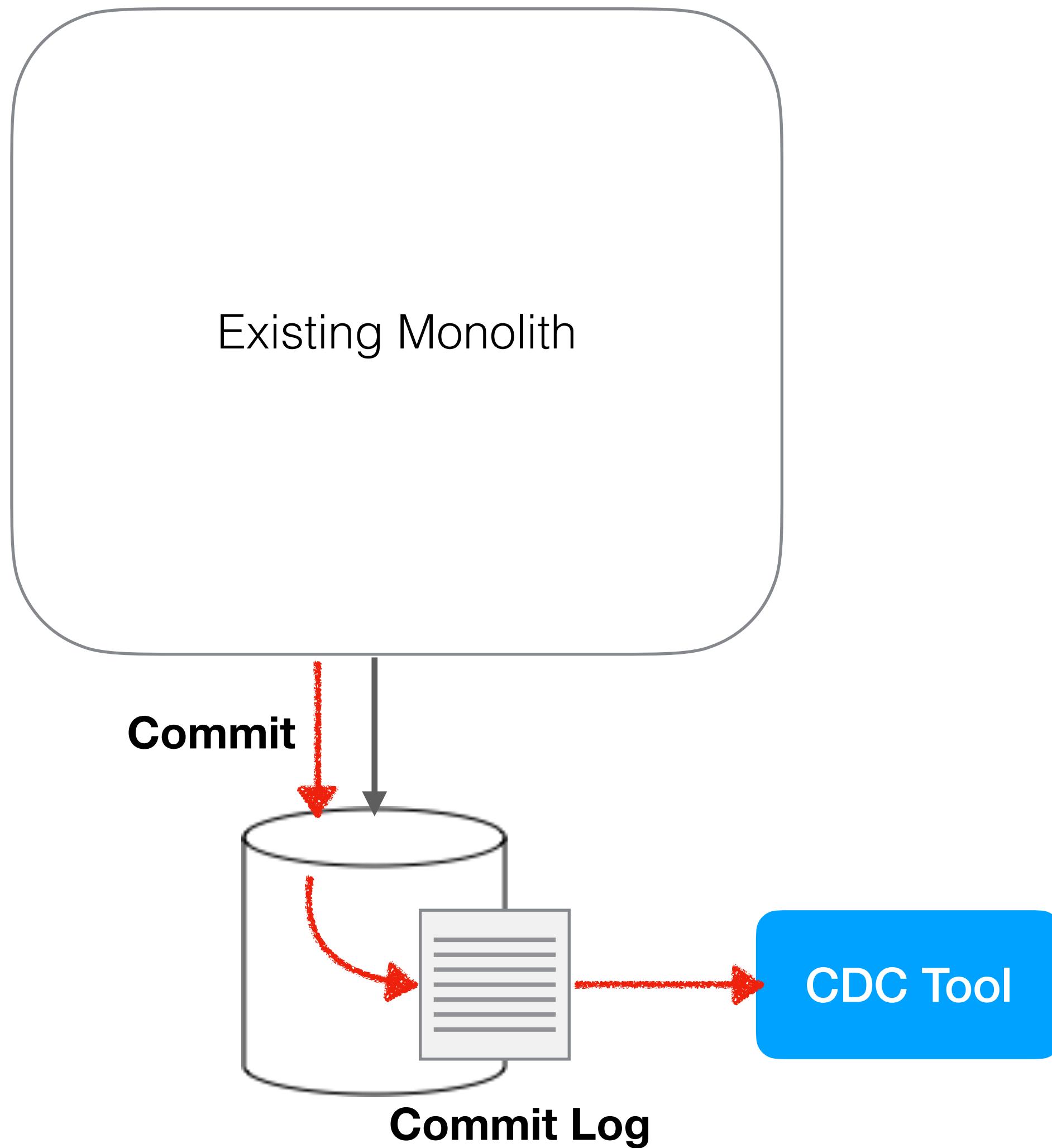


PATTERN: CHANGE DATA CAPTURE

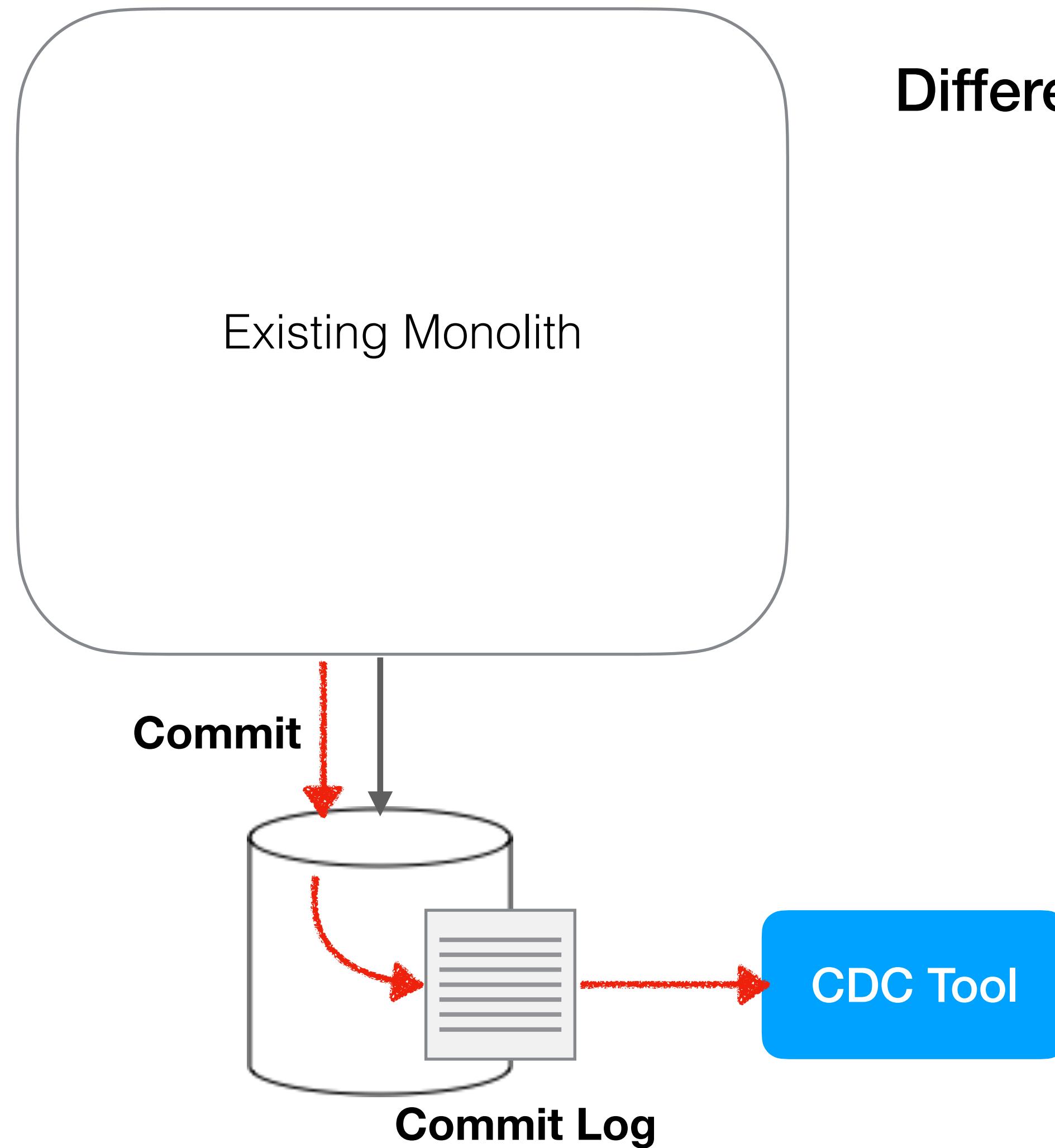


PATTERN: CHANGE DATA CAPTURE

Capture data as its inserted



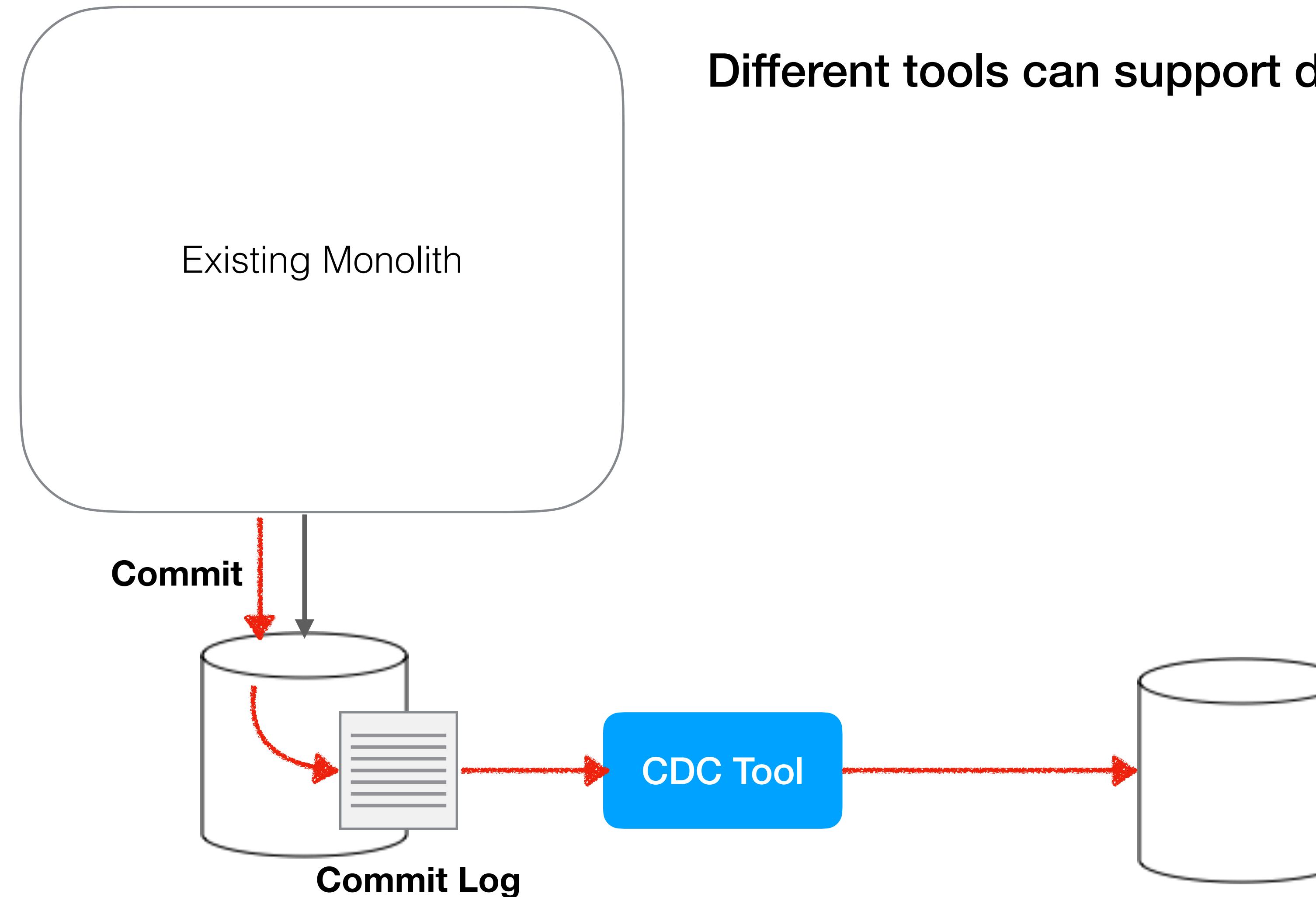
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

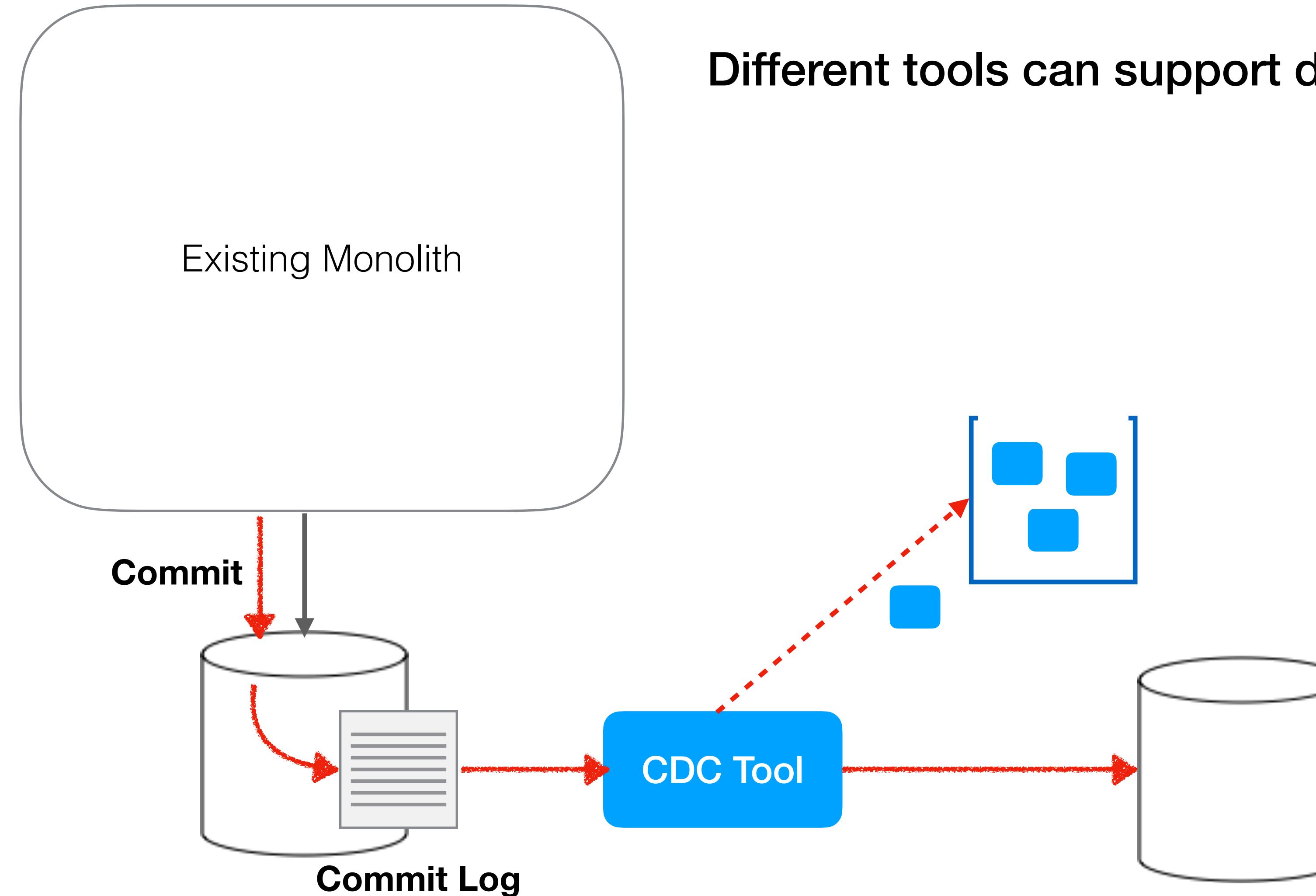
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

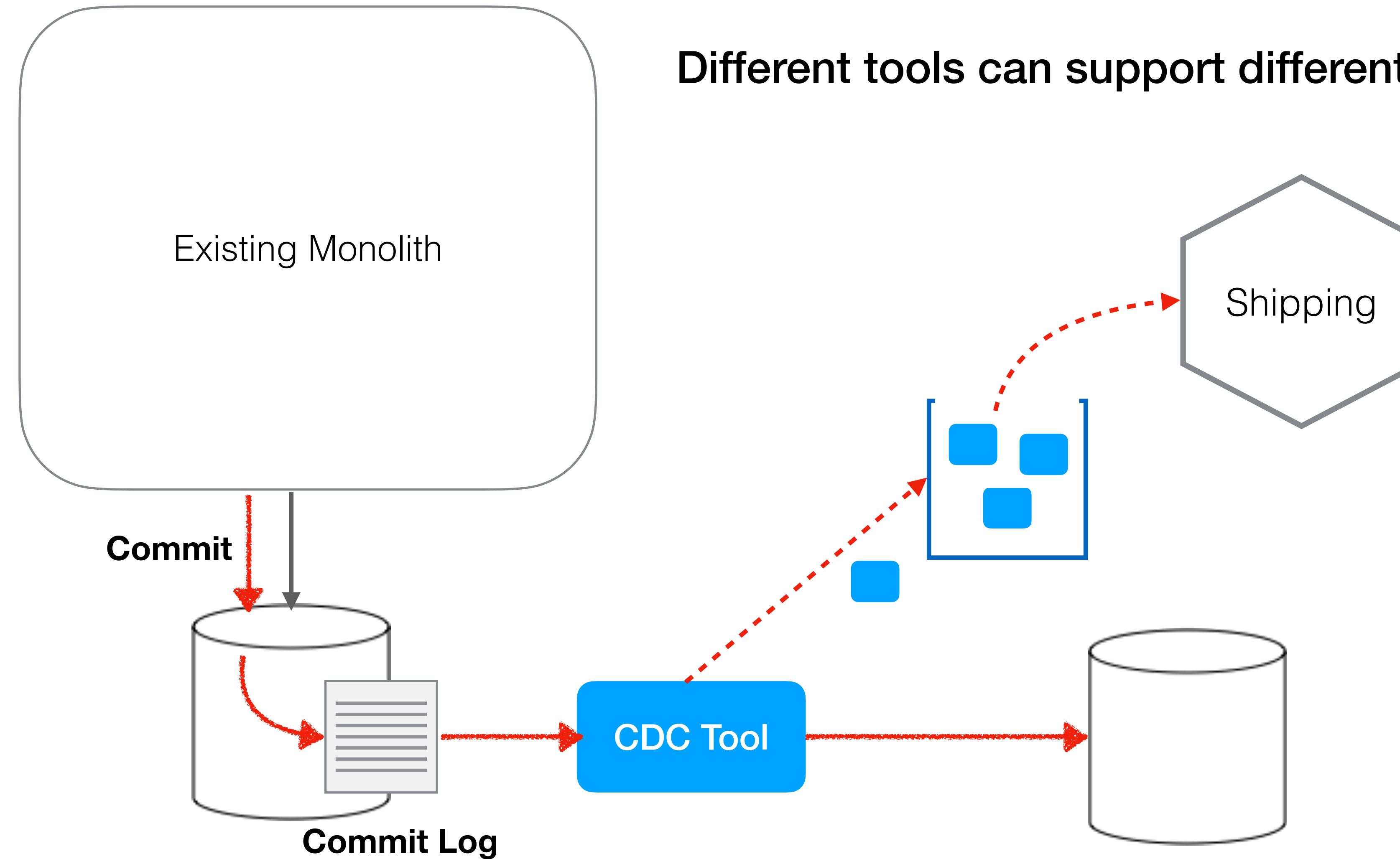
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

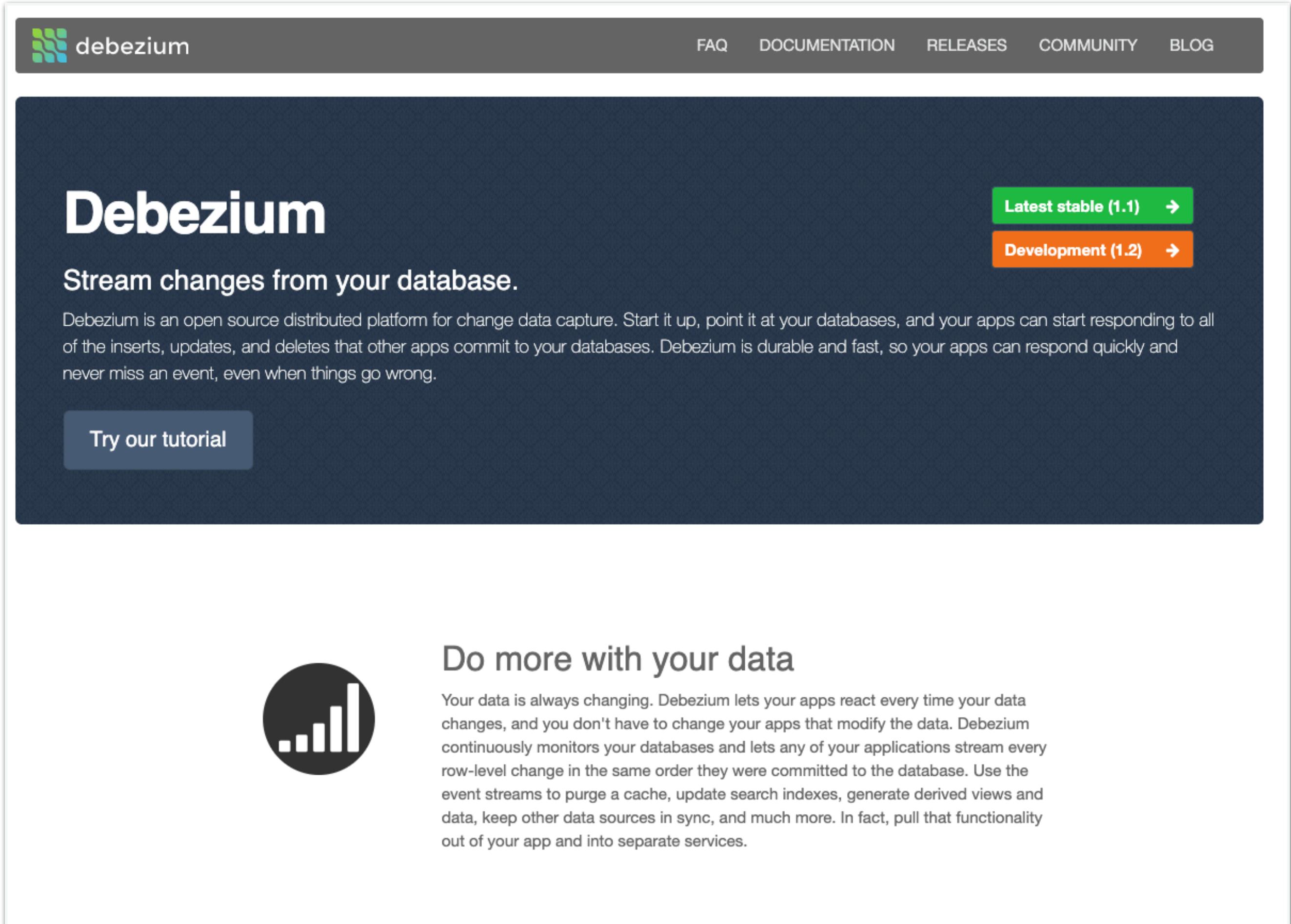
PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

DEBEZIUM, FOR FUN AND (HOPEFULLY) PROFIT



The screenshot shows the official Debezium website. At the top, there's a dark header bar with the Debezium logo on the left and navigation links for FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG. Below the header, the main content area has a dark background. On the left, the word "Debezium" is written in large white letters. To its right, there are two buttons: a green one labeled "Latest stable (1.1)" and an orange one labeled "Development (1.2)". Underneath these buttons, the text "Stream changes from your database." is displayed in white. A paragraph of text follows, explaining what Debezium does. At the bottom left of this section is a blue button with the text "Try our tutorial". In the bottom right corner of the main content area, there's a section titled "Do more with your data" with some descriptive text and a small icon of a signal tower.

Debezium

Stream changes from your database.

Debezium is an open source distributed platform for change data capture. Start it up, point it at your databases, and your apps can start responding to all of the inserts, updates, and deletes that other apps commit to your databases. Debezium is durable and fast, so your apps can respond quickly and never miss an event, even when things go wrong.

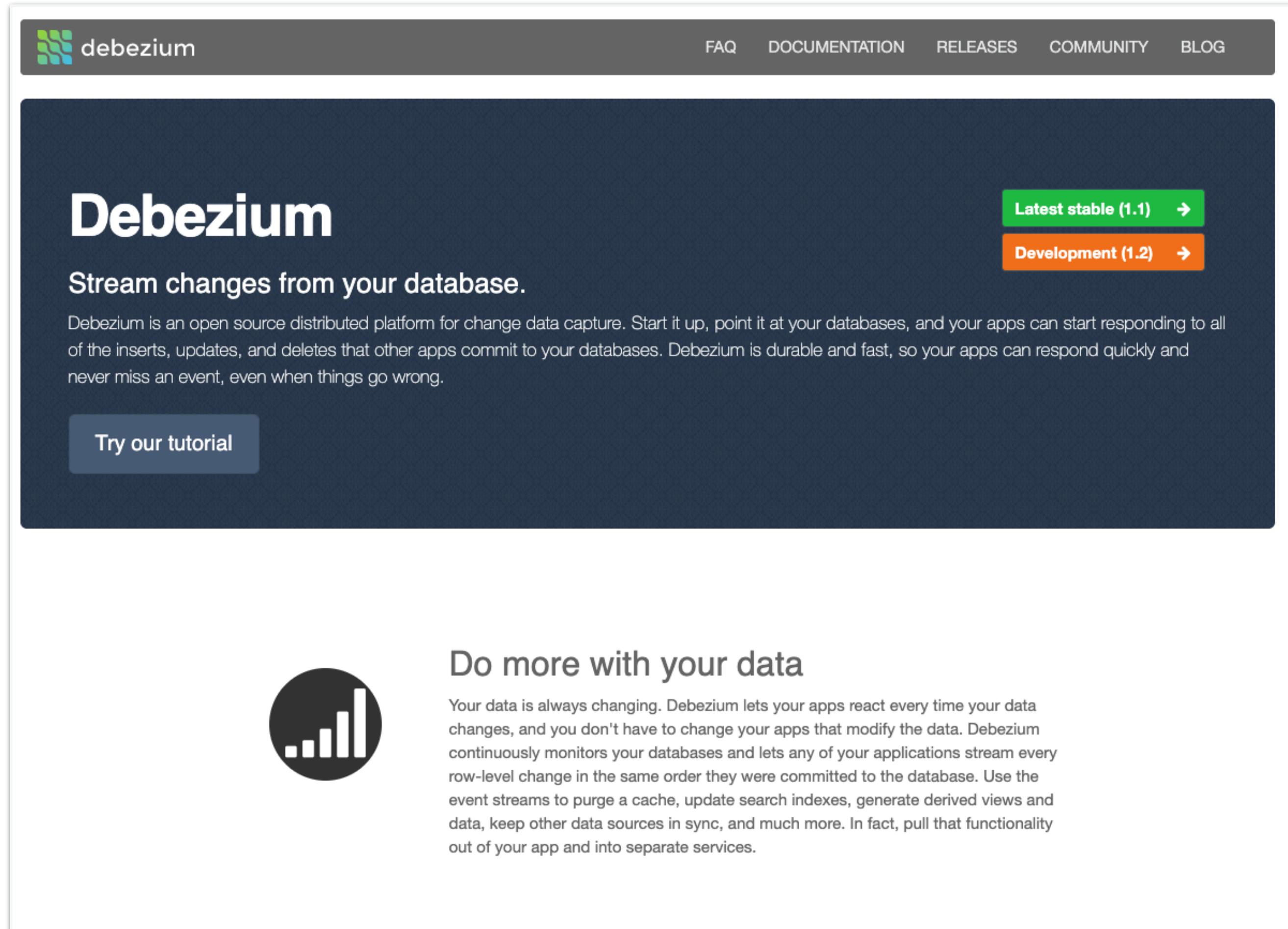
Try our tutorial

Do more with your data

Your data is always changing. Debezium lets your apps react every time your data changes, and you don't have to change your apps that modify the data. Debezium continuously monitors your databases and lets any of your applications stream every row-level change in the same order they were committed to the database. Use the event streams to purge a cache, update search indexes, generate derived views and data, keep other data sources in sync, and much more. In fact, pull that functionality out of your app and into separate services.

<https://debezium.io/>

DEBEZIUM, FOR FUN AND (HOPEFULLY) PROFIT



The screenshot shows the official Debezium website. At the top, there's a dark header bar with the Debezium logo on the left and navigation links for FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG on the right. Below the header is a large dark blue banner featuring the word "Debezium" in white. Underneath it, the tagline "Stream changes from your database." is displayed. To the right of the tagline are two buttons: a green one labeled "Latest stable (1.1)" and an orange one labeled "Development (1.2)". Below the banner, a call-to-action button says "Try our tutorial". In the bottom left corner of the main content area, there's a circular icon containing a bar chart or signal strength graphic. To its right, the text "Do more with your data" is followed by a detailed paragraph explaining how Debezium monitors databases for changes and streams them to applications.

Debezium

Stream changes from your database.

Debezium is an open source distributed platform for change data capture. Start it up, point it at your databases, and your apps can start responding to all of the inserts, updates, and deletes that other apps commit to your databases. Debezium is durable and fast, so your apps can respond quickly and never miss an event, even when things go wrong.

Try our tutorial

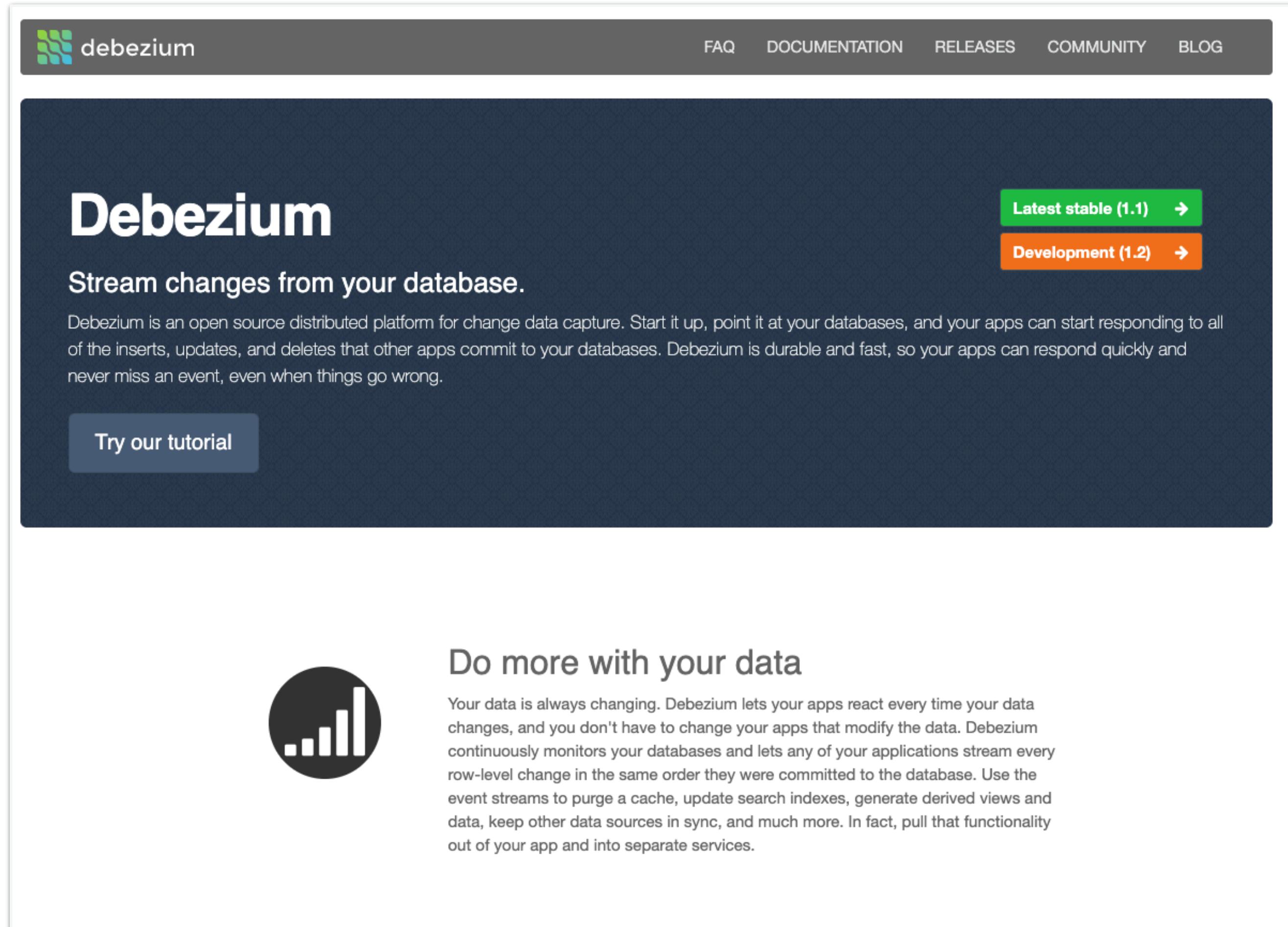
Do more with your data

Your data is always changing. Debezium lets your apps react every time your data changes, and you don't have to change your apps that modify the data. Debezium continuously monitors your databases and lets any of your applications stream every row-level change in the same order they were committed to the database. Use the event streams to purge a cache, update search indexes, generate derived views and data, keep other data sources in sync, and much more. In fact, pull that functionality out of your app and into separate services.

<https://debezium.io/>

Open source tool which implements the change data capture pattern for a number of databases

DEBEZIUM, FOR FUN AND (HOPEFULLY) PROFIT



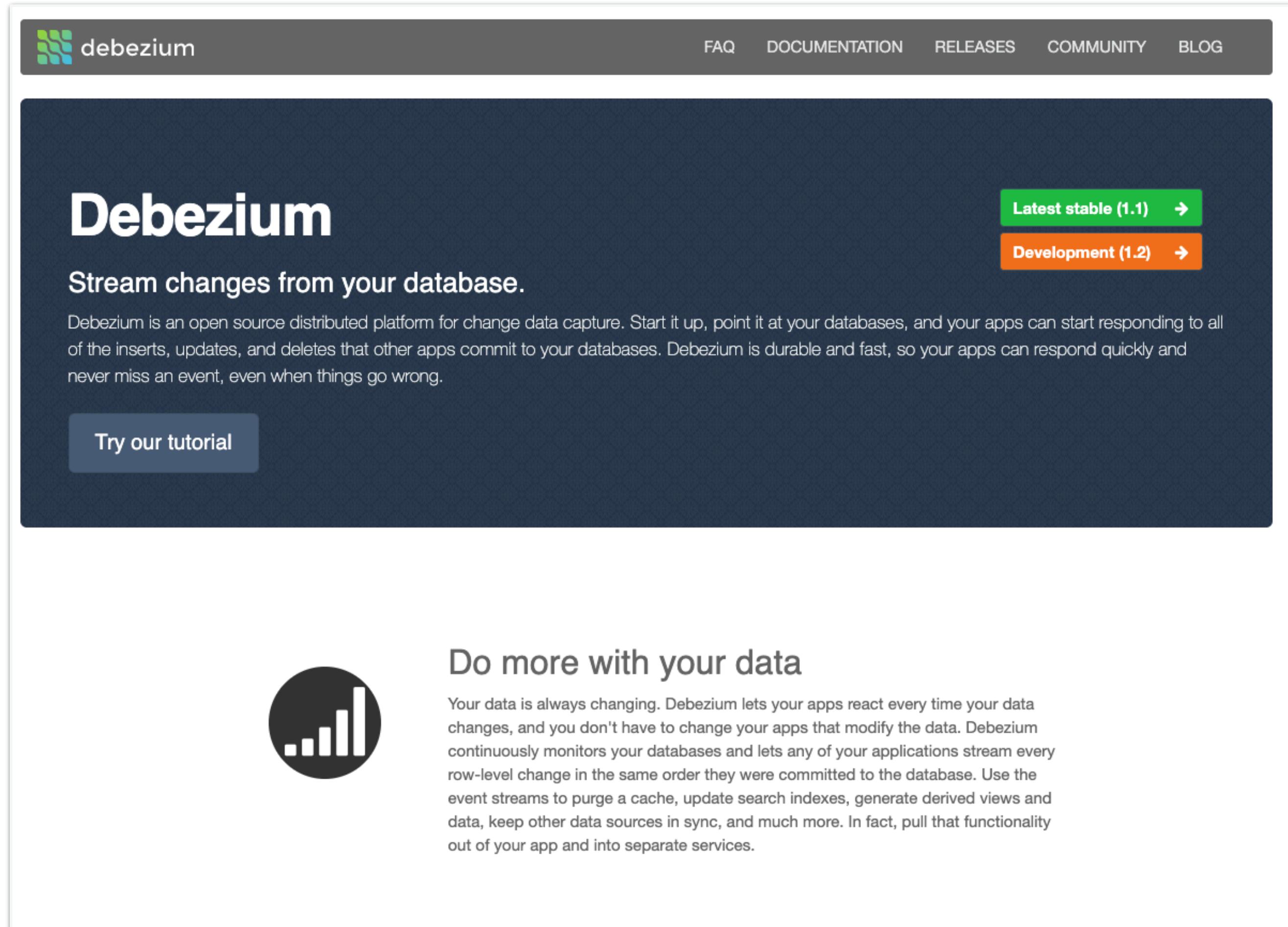
The screenshot shows the official Debezium website. At the top, there's a dark header bar with the Debezium logo on the left and navigation links for FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG on the right. Below the header, the main title "Debezium" is prominently displayed in large white font. A sub-headline "Stream changes from your database." follows. To the right of the headline are two buttons: "Latest stable (1.1)" in green and "Development (1.2)" in orange. Below the headline, a brief description explains that Debezium is an open source distributed platform for change data capture, monitoring databases for inserts, updates, and deletes. A blue button labeled "Try our tutorial" is located at the bottom left of the main content area. On the left side of the page, there's a sidebar with a circular icon containing a bar chart and the text "Do more with your data". Below this, a detailed paragraph explains how Debezium lets applications react to data changes without modifying the data source, using event streams for various operations like purging caches or generating derived views.

<https://debezium.io/>

Open source tool which implements the change data capture pattern for a number of databases

Sends captured data over kafka

DEBEZIUM, FOR FUN AND (HOPEFULLY) PROFIT



The screenshot shows the official Debezium website. At the top, there's a dark header bar with the Debezium logo on the left and navigation links for FAQ, DOCUMENTATION, RELEASES, COMMUNITY, and BLOG on the right. Below the header, the main title "Debezium" is prominently displayed in large white font. A sub-headline "Stream changes from your database." follows. A brief description explains that Debezium is an open source distributed platform for change data capture, monitoring databases for inserts, updates, and deletes. Two download buttons are visible: "Latest stable (1.1)" in green and "Development (1.2)" in orange. A blue button labeled "Try our tutorial" is located at the bottom left. On the right side of the main content area, there's a section titled "Do more with your data" featuring a circular icon with a bar chart. The text in this section discusses how Debezium lets apps react to data changes without modifying the data source, monitoring databases for row-level changes, and using event streams for various operations like purging caches or generating derived views.

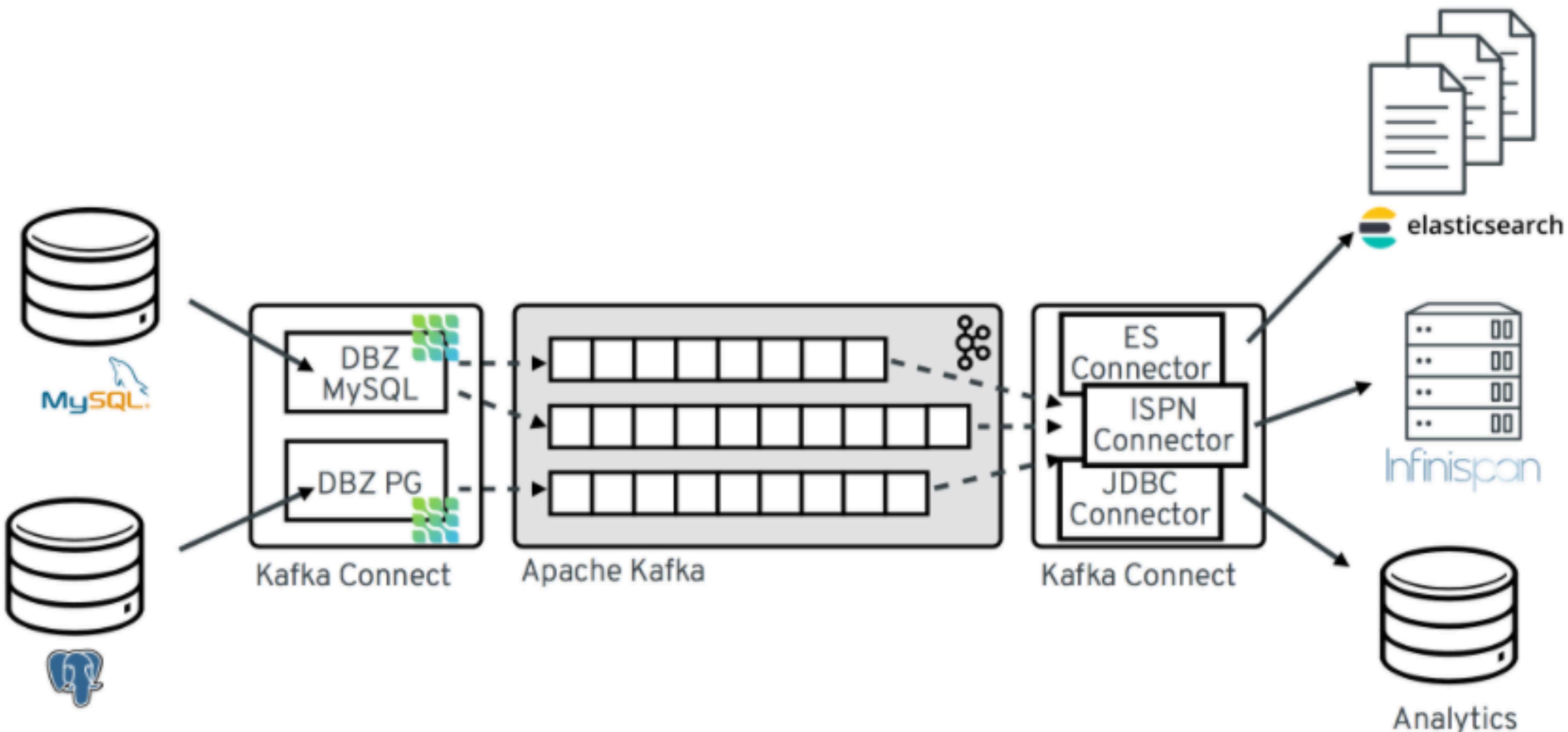
<https://debezium.io/>

Open source tool which implements the change data capture pattern for a number of databases

Sends captured data over kafka

Other tools are available!

DEBEZIUM IN USE



<https://debezium.io/documentation/reference/architecture.html>

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

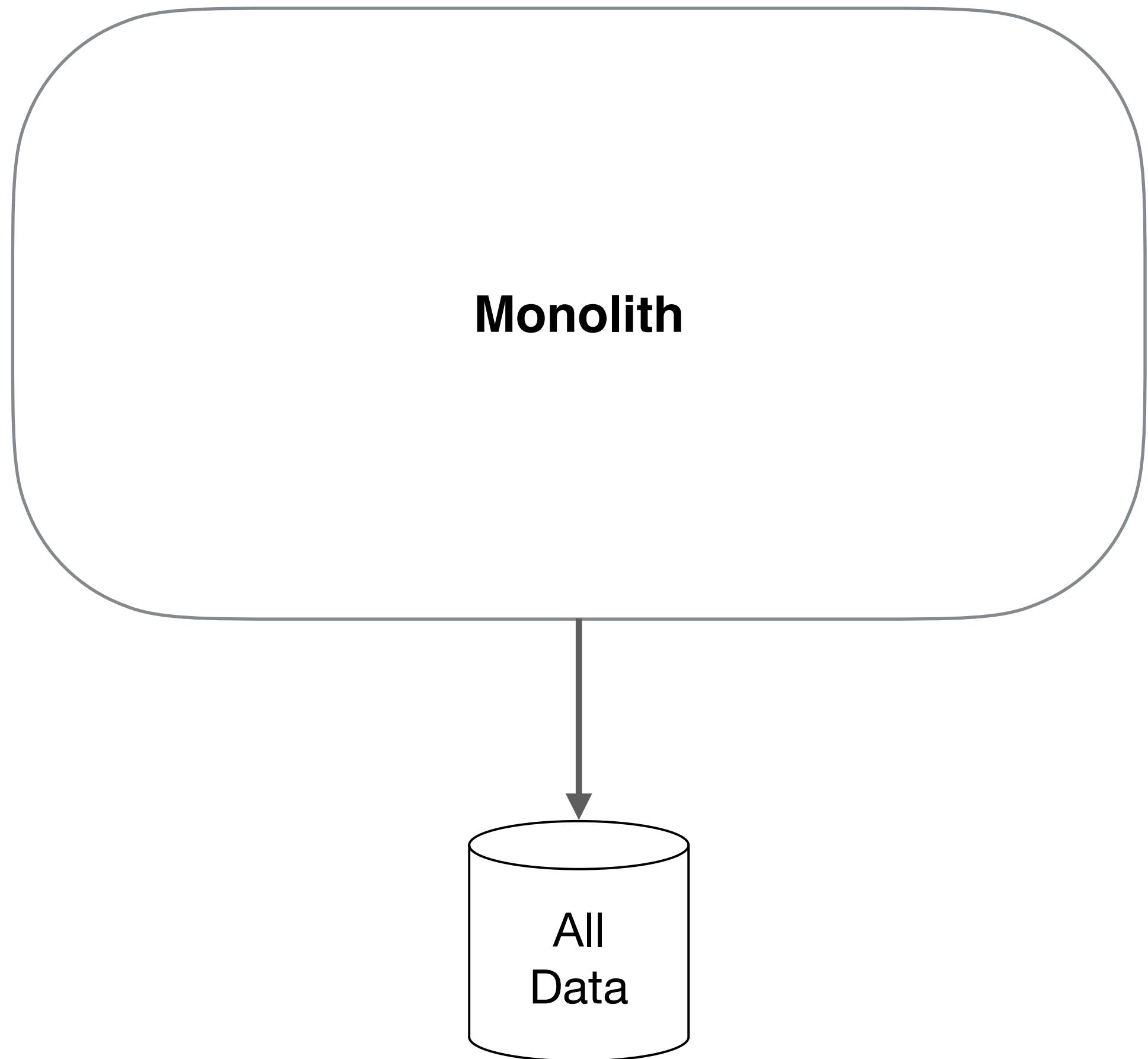
Shared Data Patterns

Migration Order

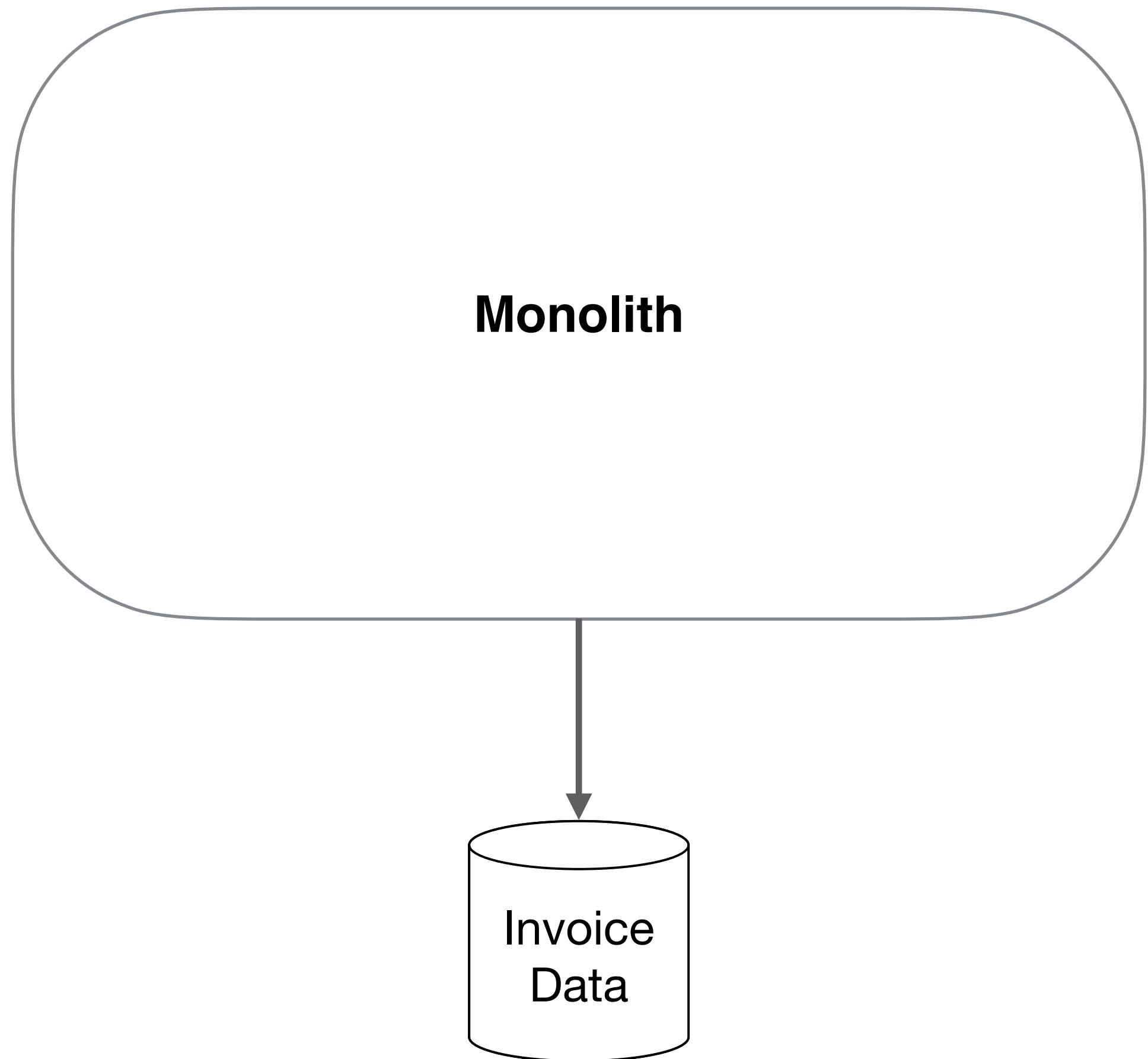
Decomposition Patterns

Extract data or app first?

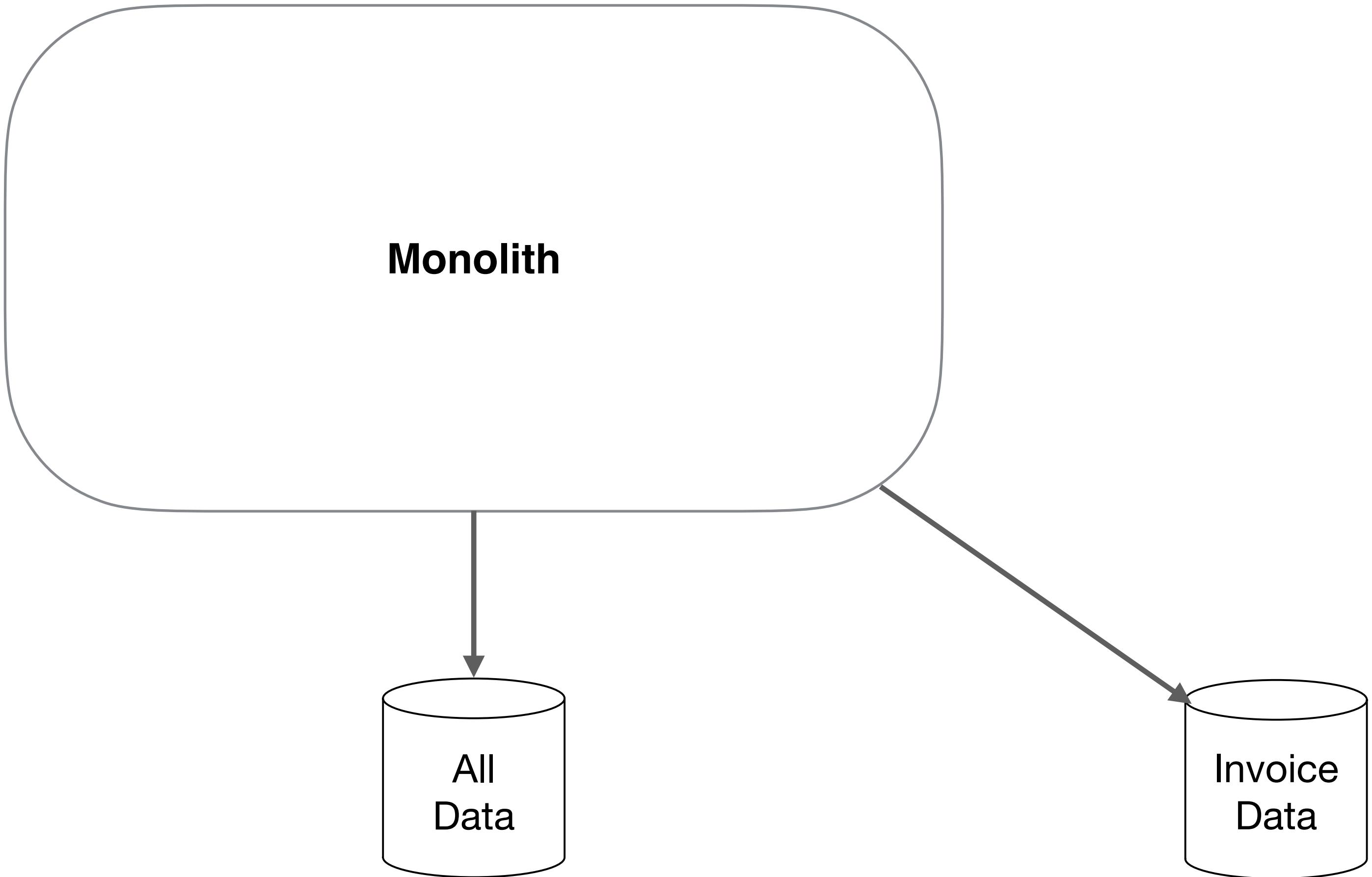
DATA FIRST?



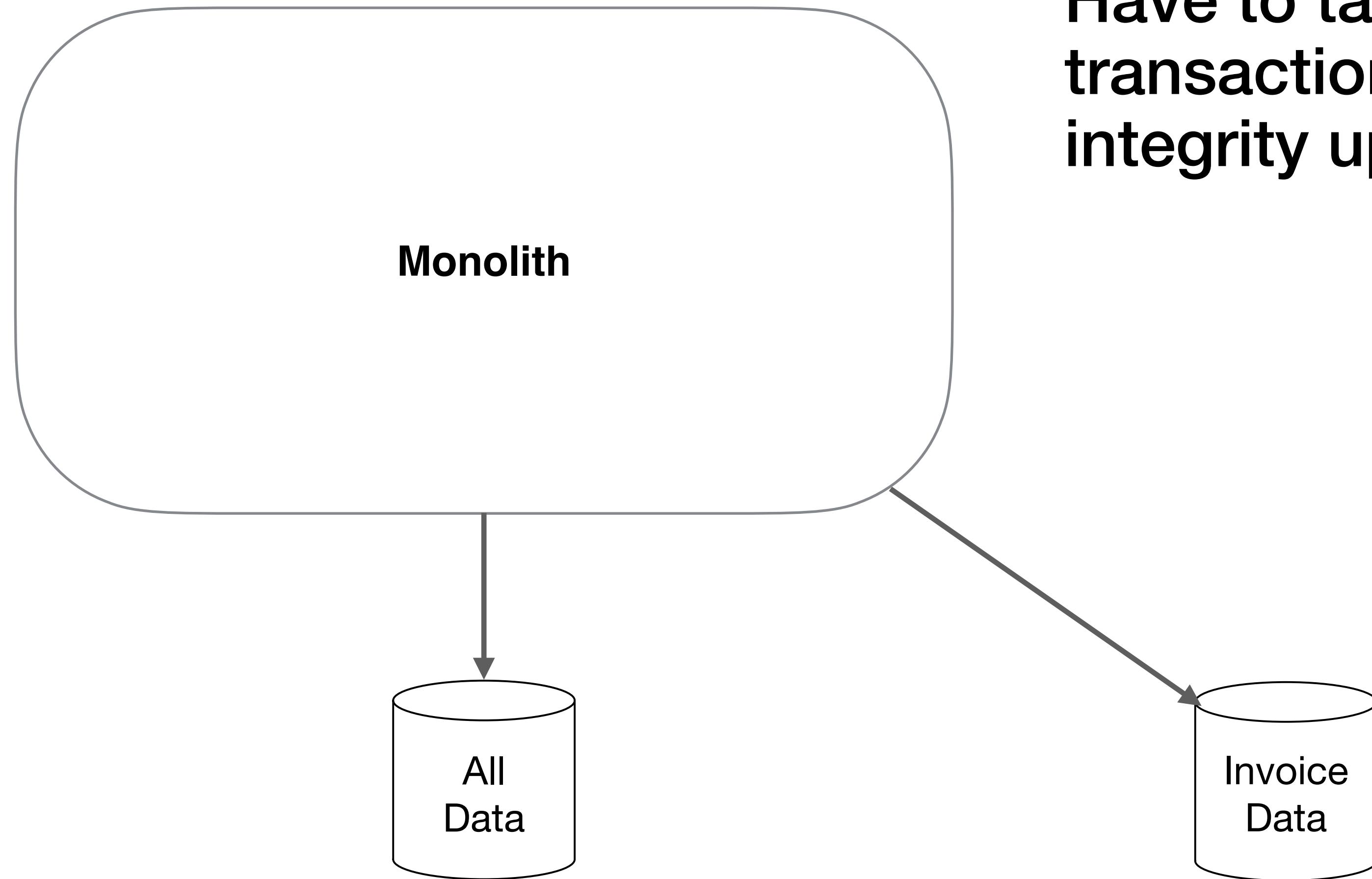
DATA FIRST?



DATA FIRST?

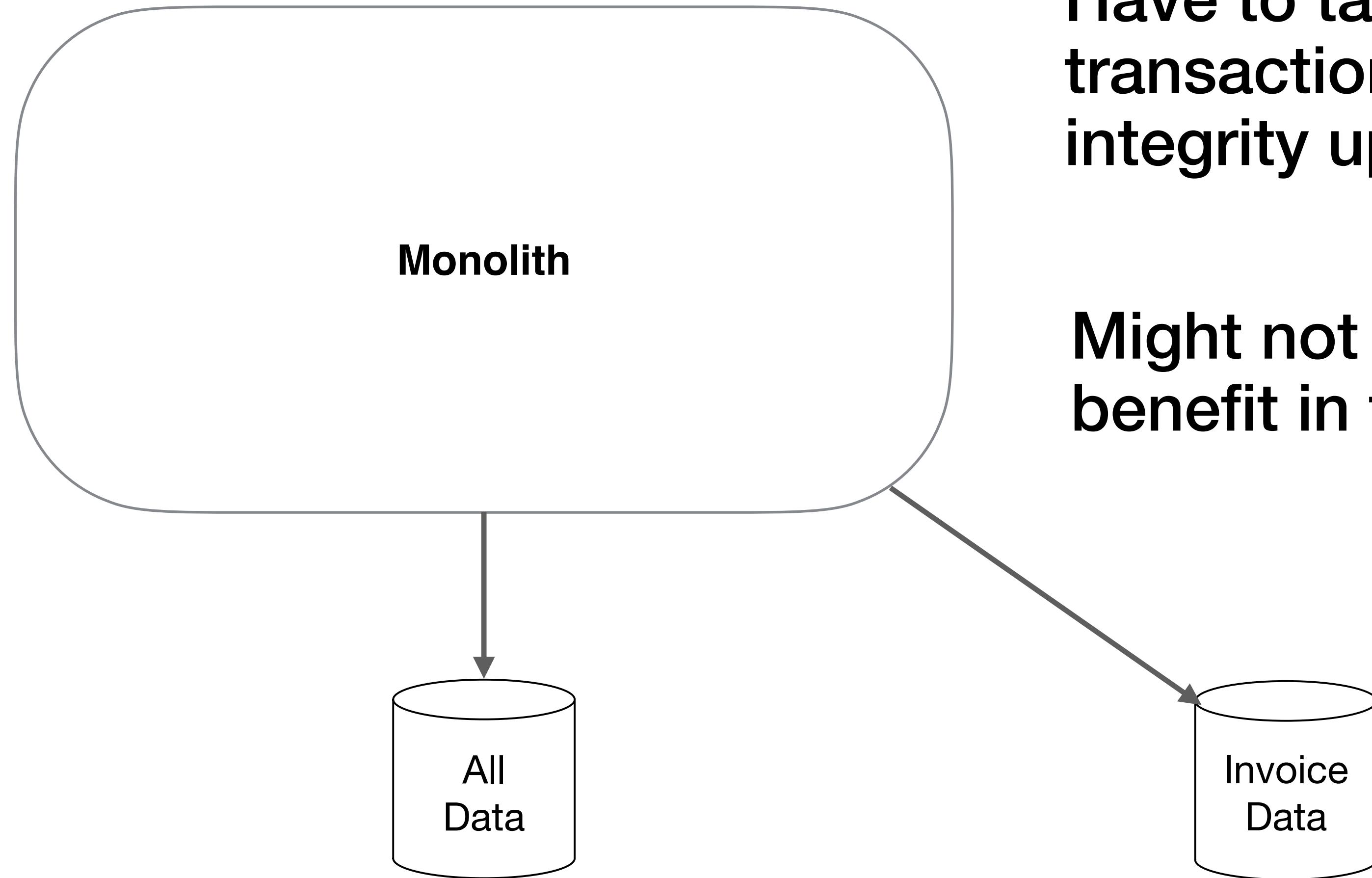


DATA FIRST?



Have to tackle issues around transactions, referential integrity up front

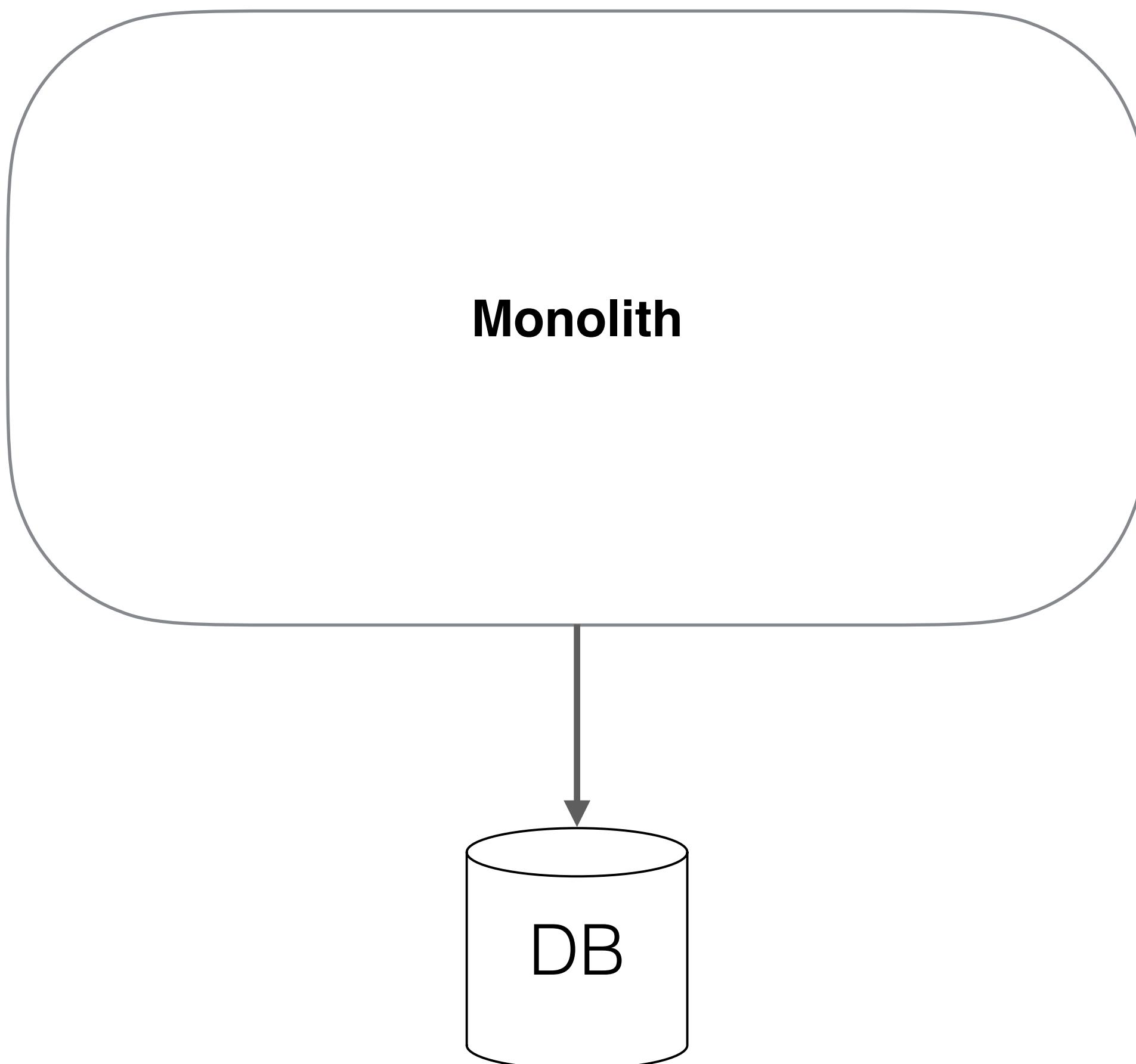
DATA FIRST?



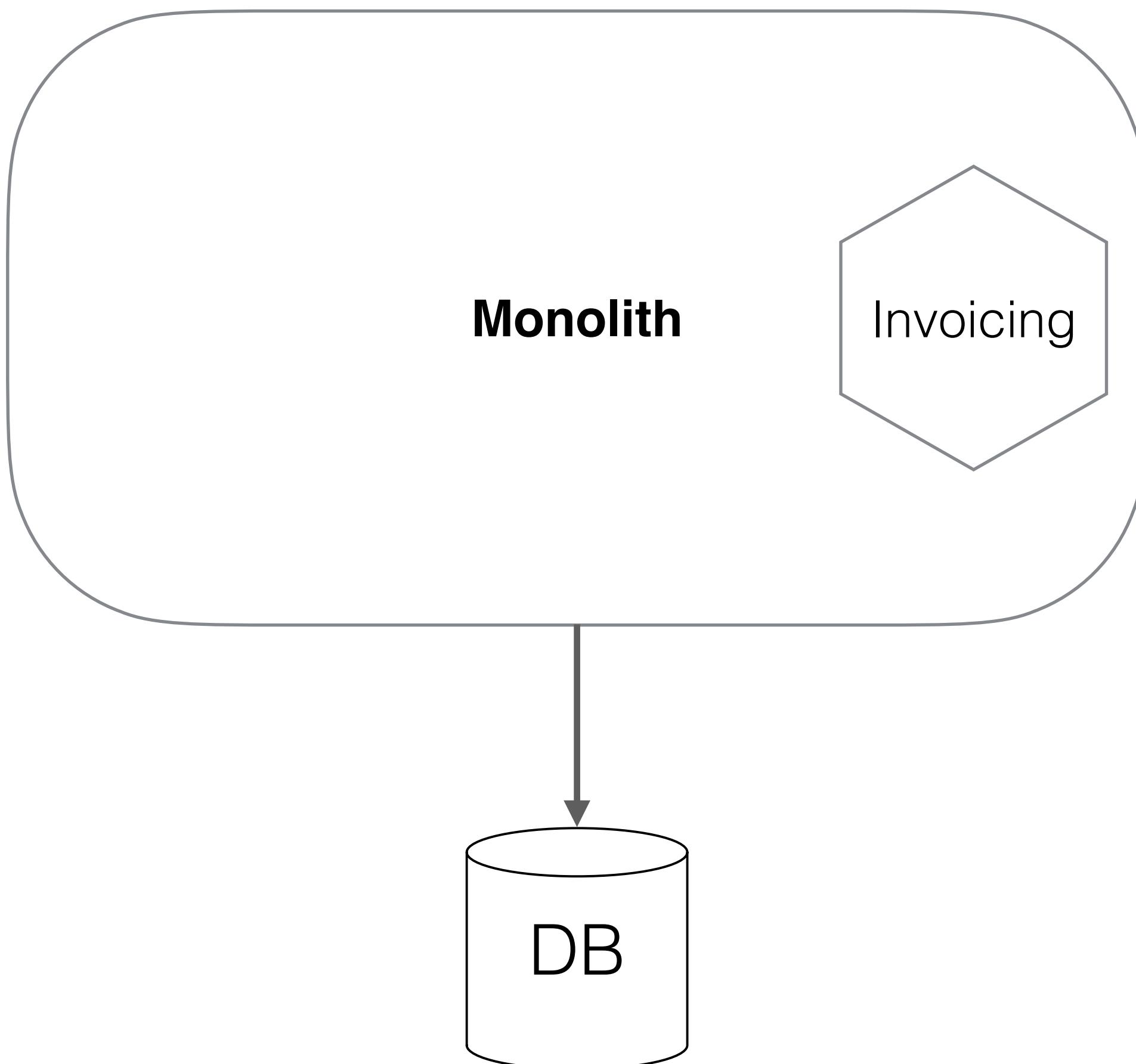
Have to tackle issues around transactions, referential integrity up front

Might not deliver much benefit in the short term

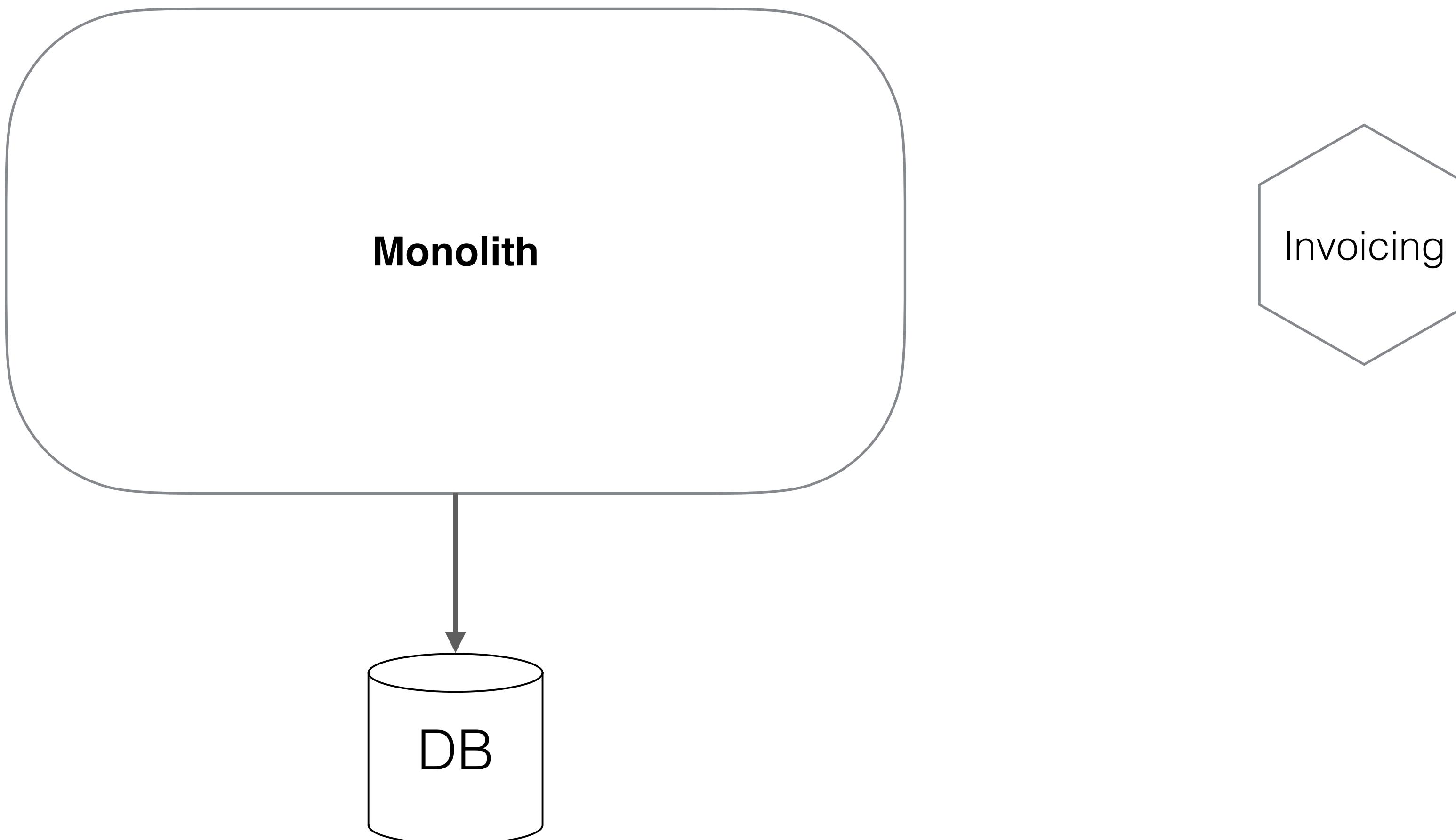
EXTRACTING CODE FIRST



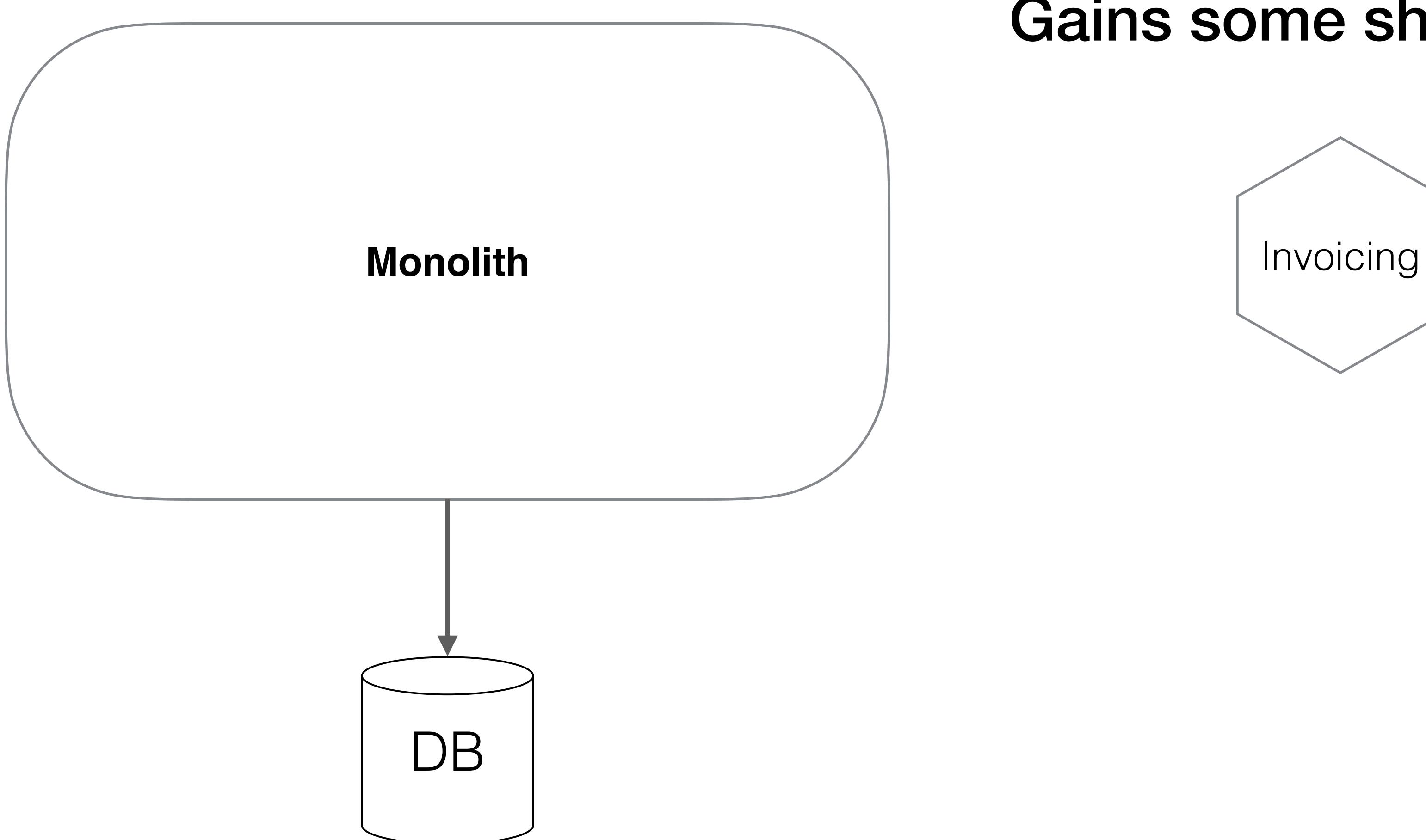
EXTRACTING CODE FIRST



EXTRACTING CODE FIRST

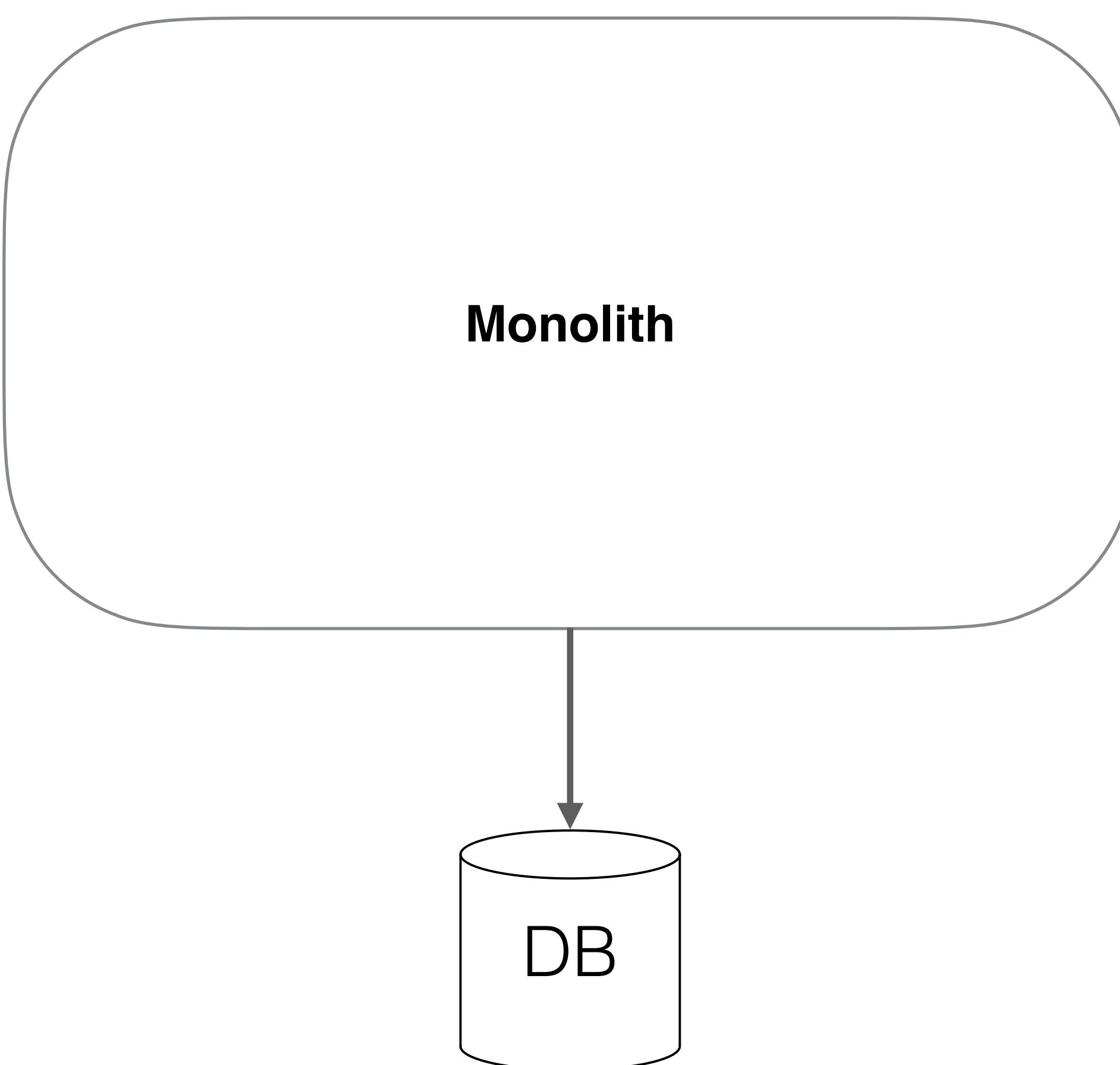


EXTRACTING CODE FIRST



Gains some short term benefit

EXTRACTING CODE FIRST

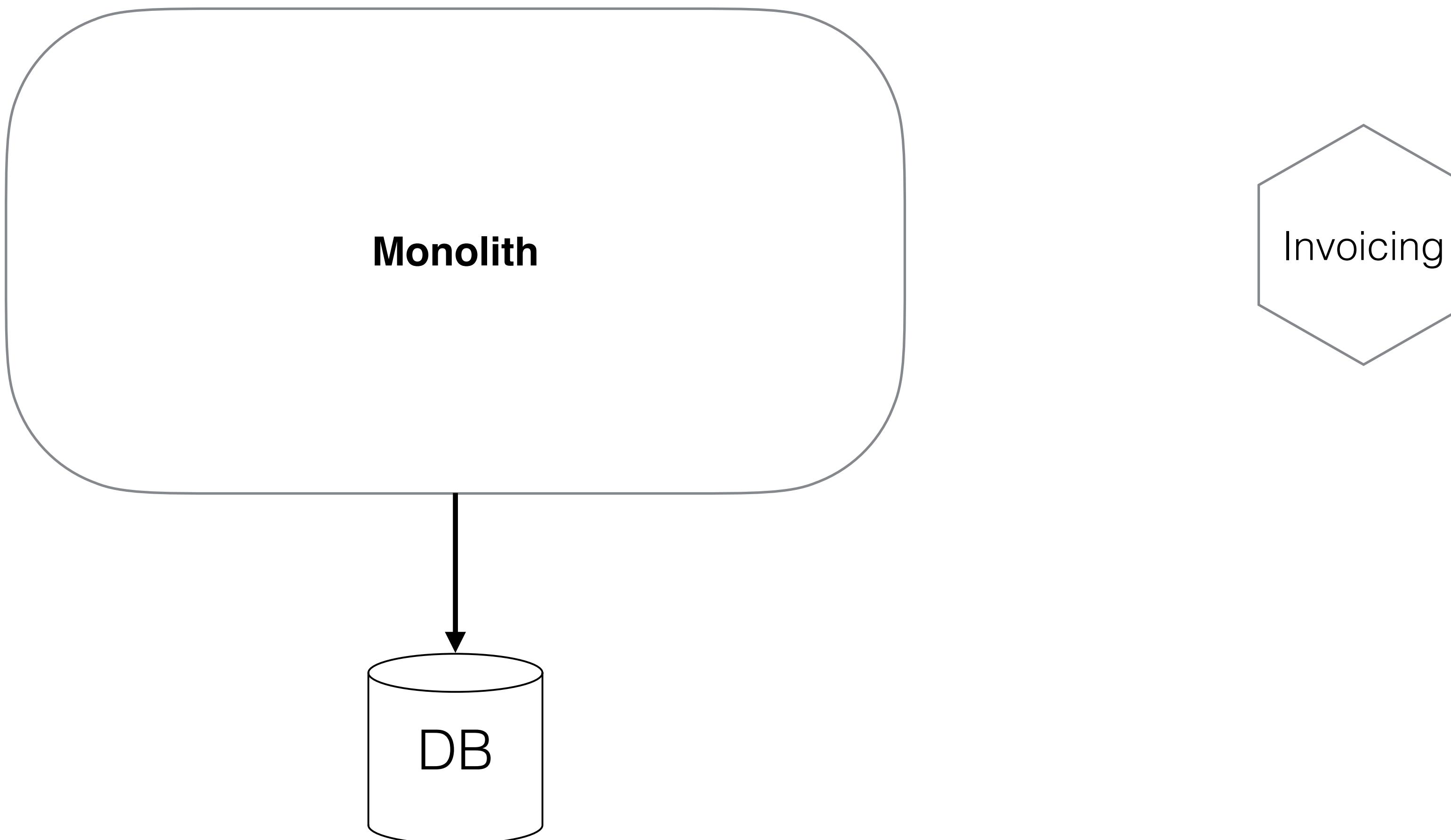


Gains some short term benefit

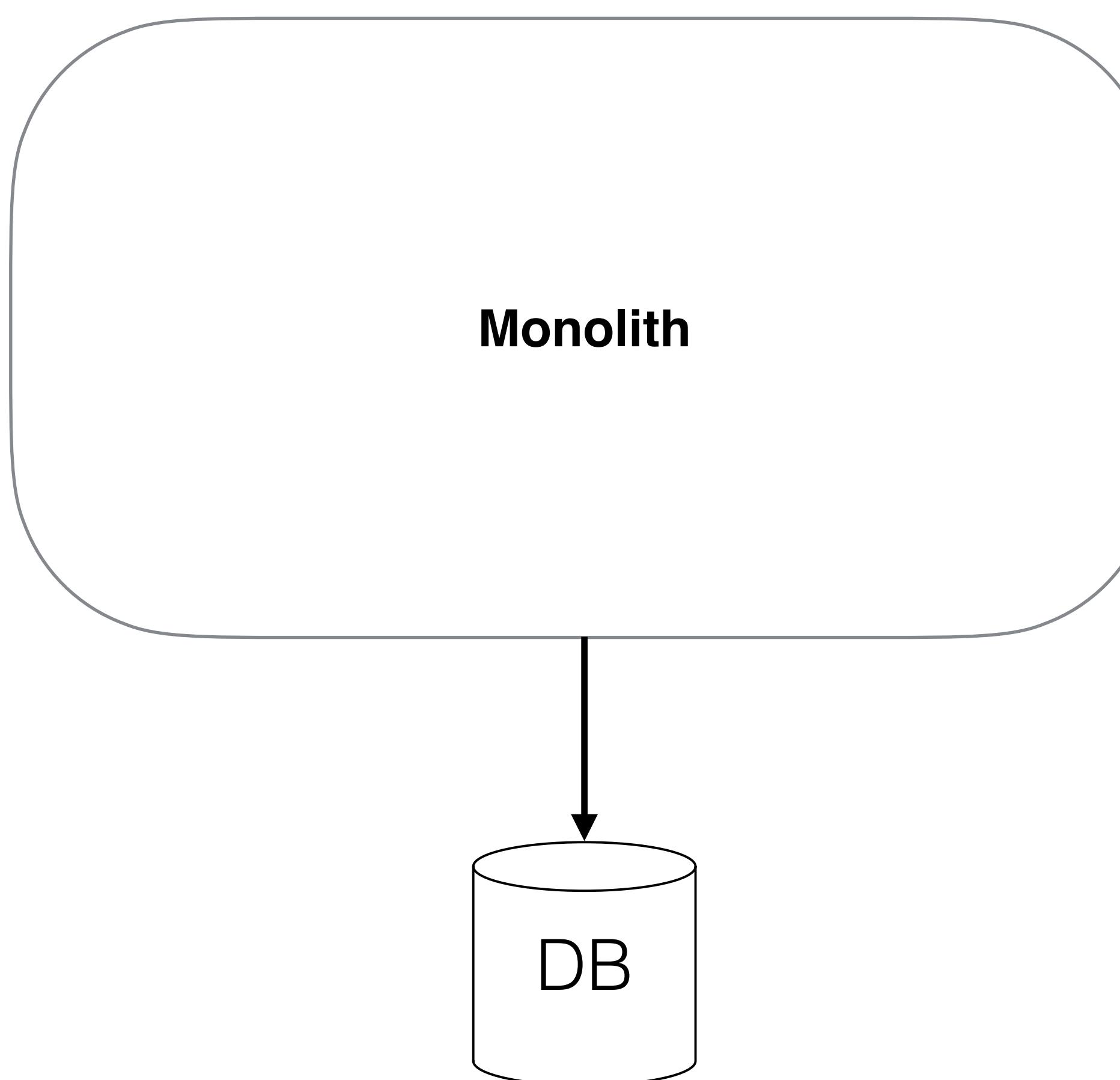


But what about data access?

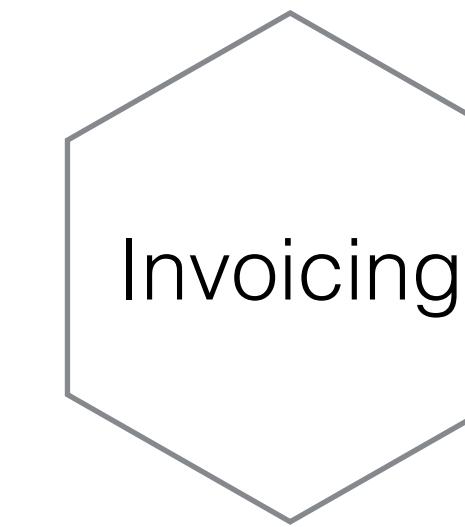
USE THE MONOLITH DB DIRECTLY



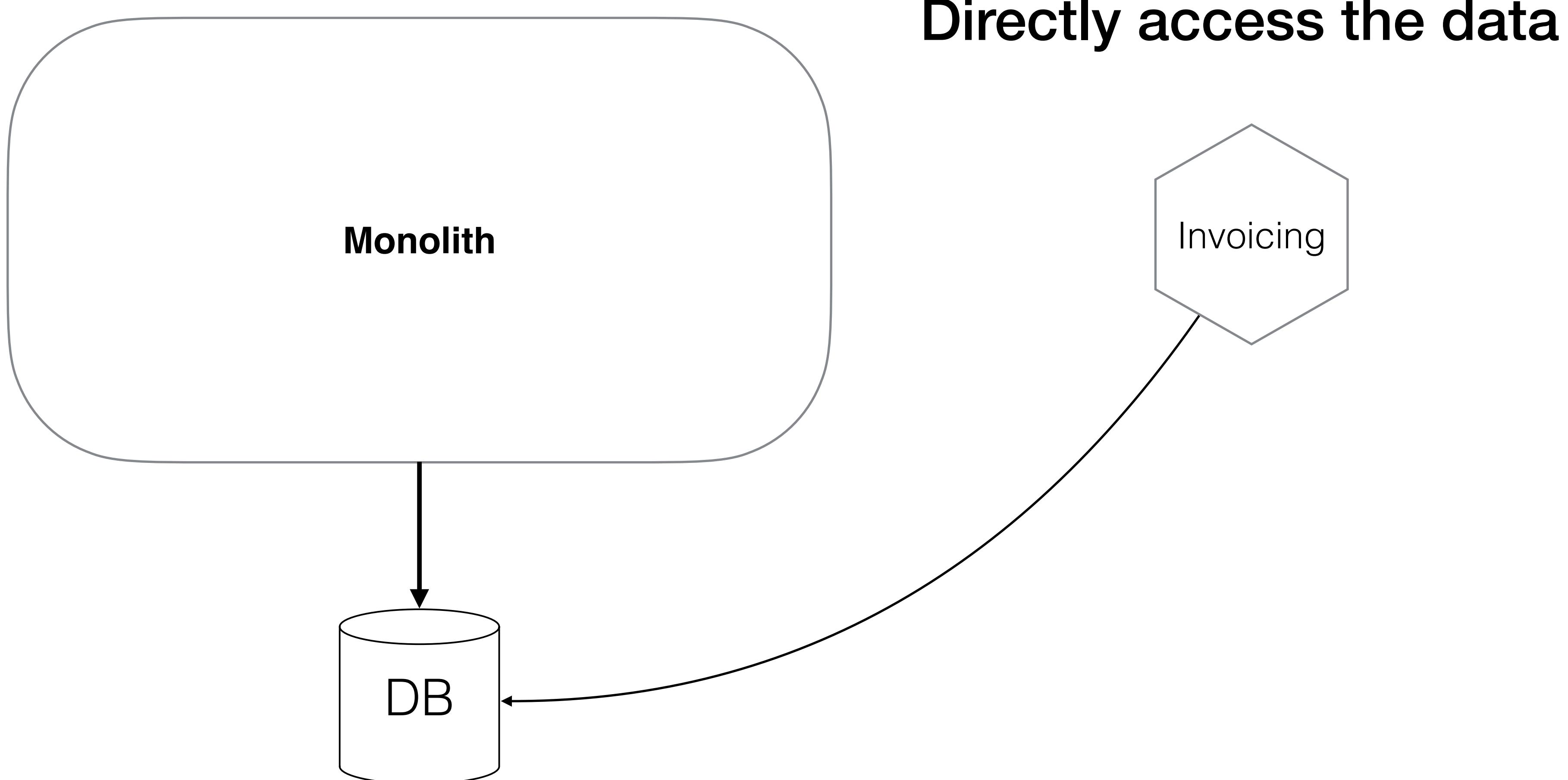
USE THE MONOLITH DB DIRECTLY



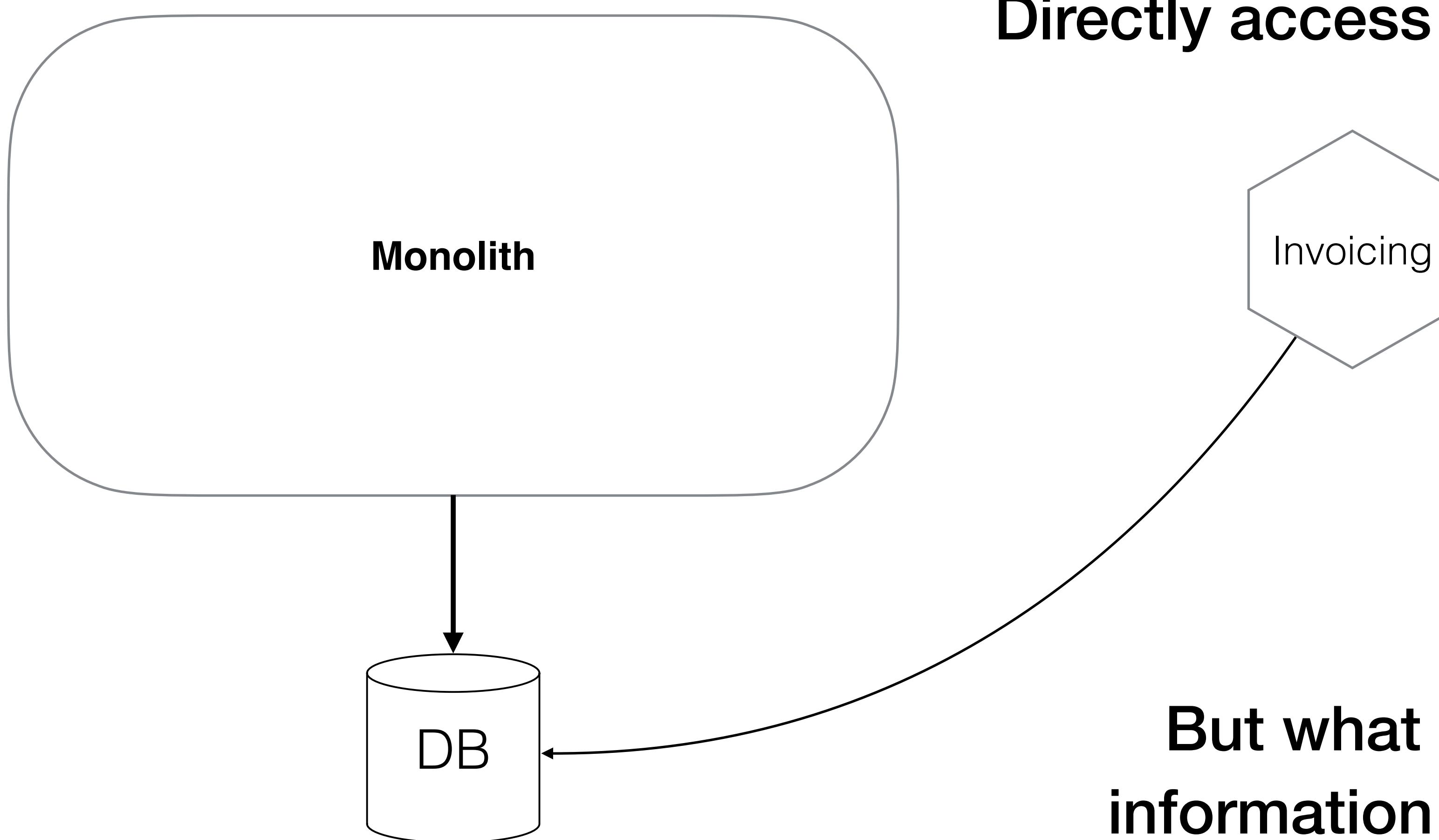
Directly access the data



USE THE MONOLITH DB DIRECTLY



USE THE MONOLITH DB DIRECTLY



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

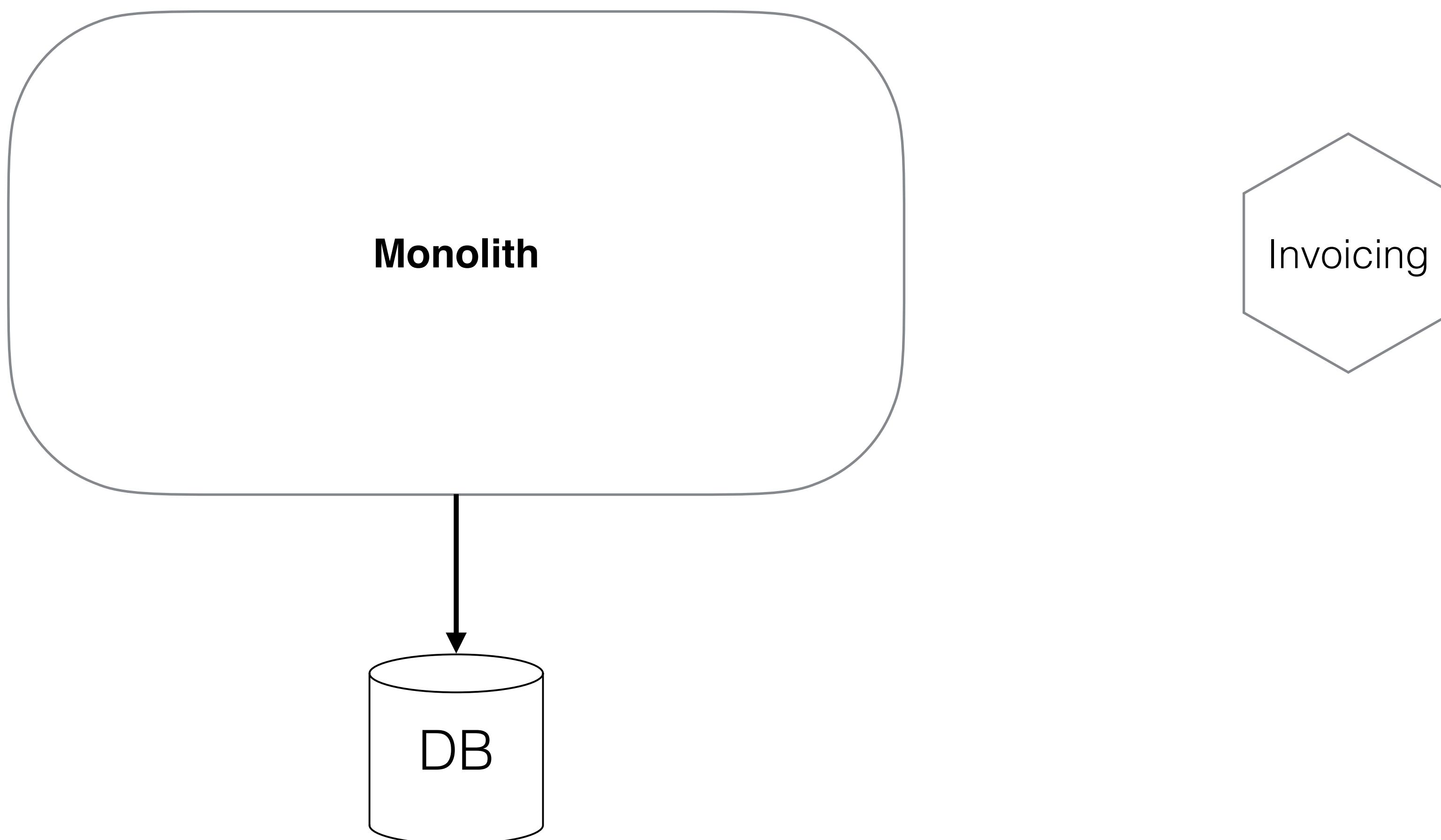
**OK for a short period of time as part of a
transition...**

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

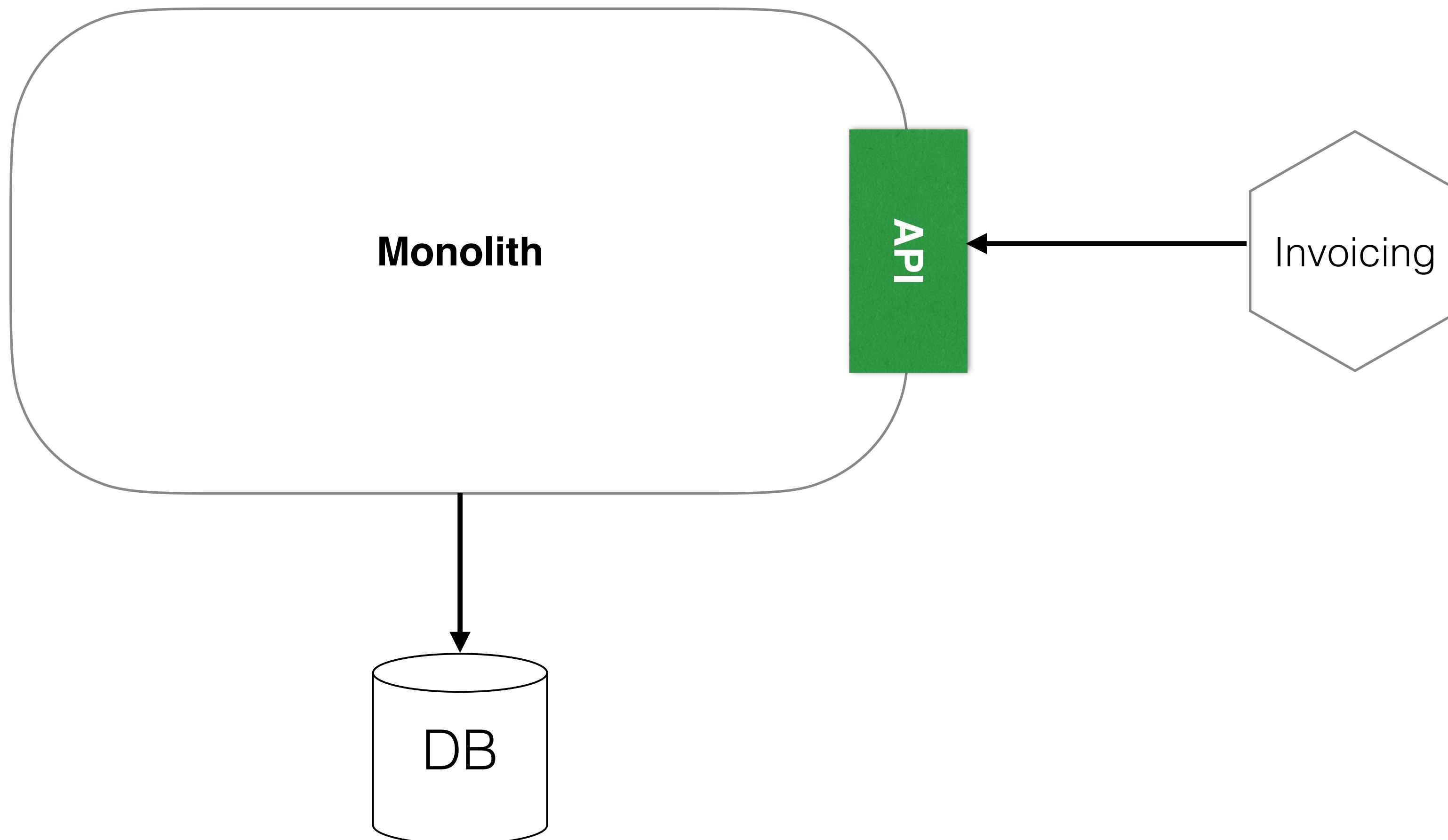
@samnewman

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

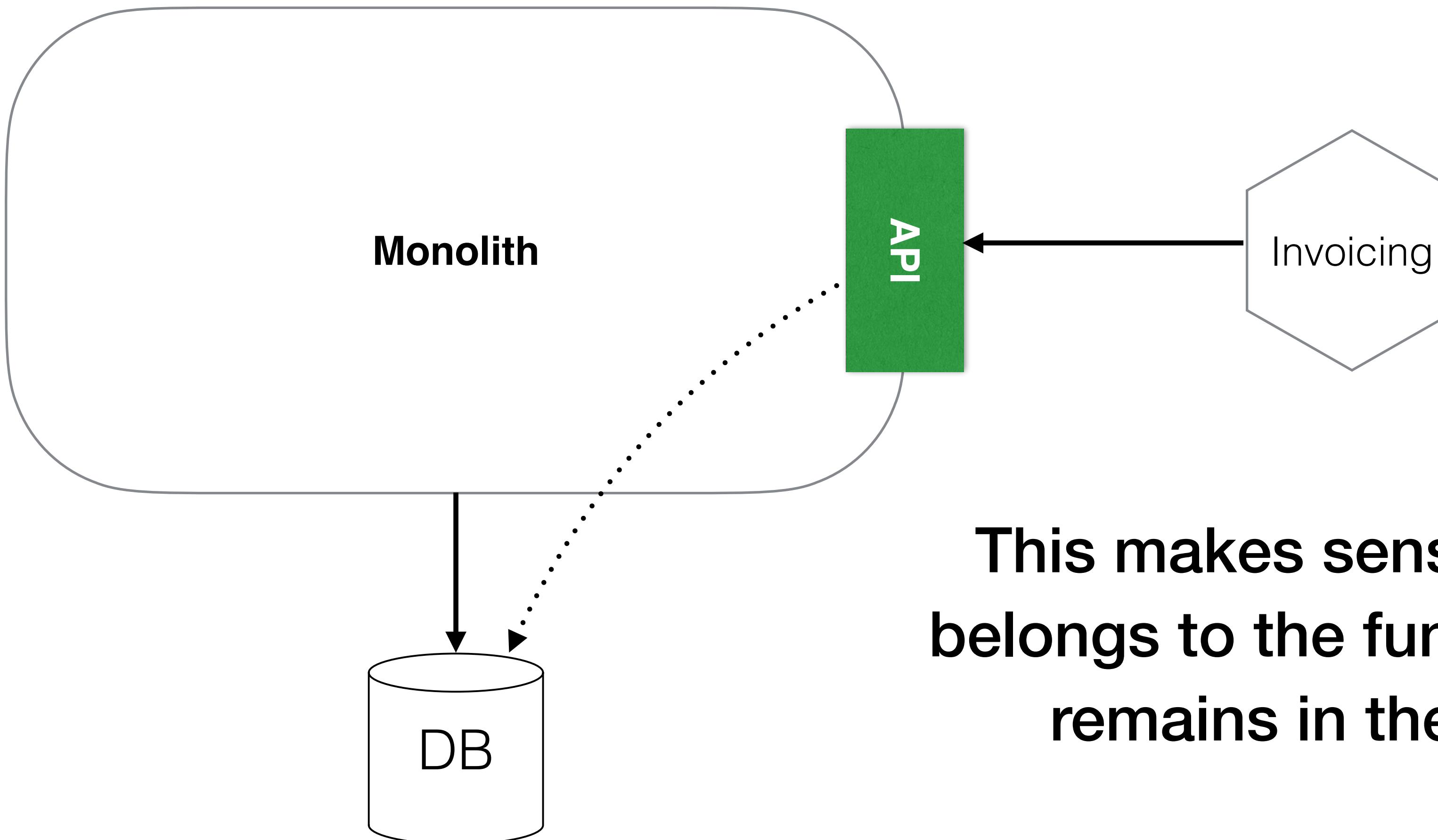
EXPOSE THE DATA VIA THE MONOLITH



EXPOSE THE DATA VIA THE MONOLITH

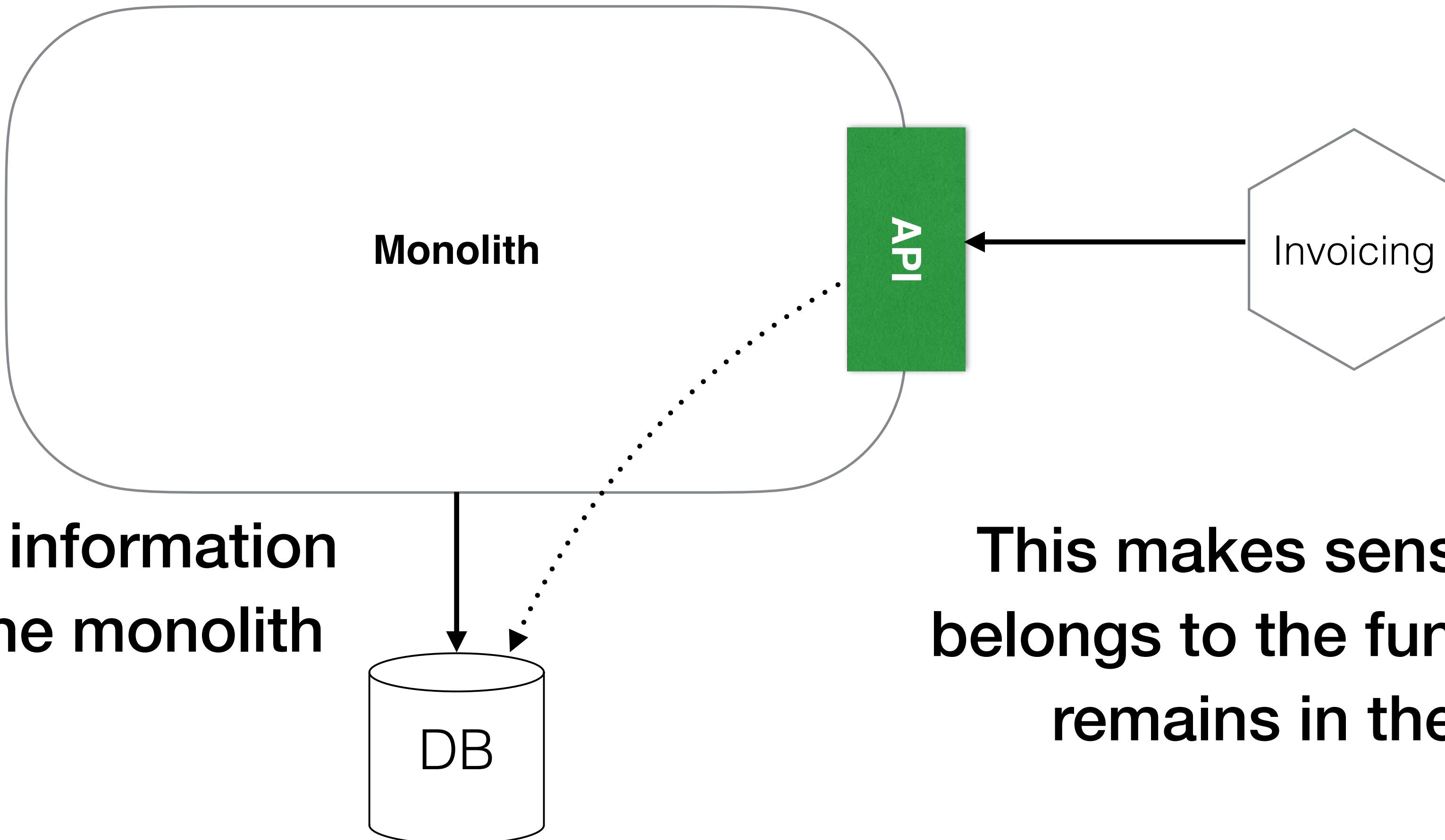


EXPOSE THE DATA VIA THE MONOLITH

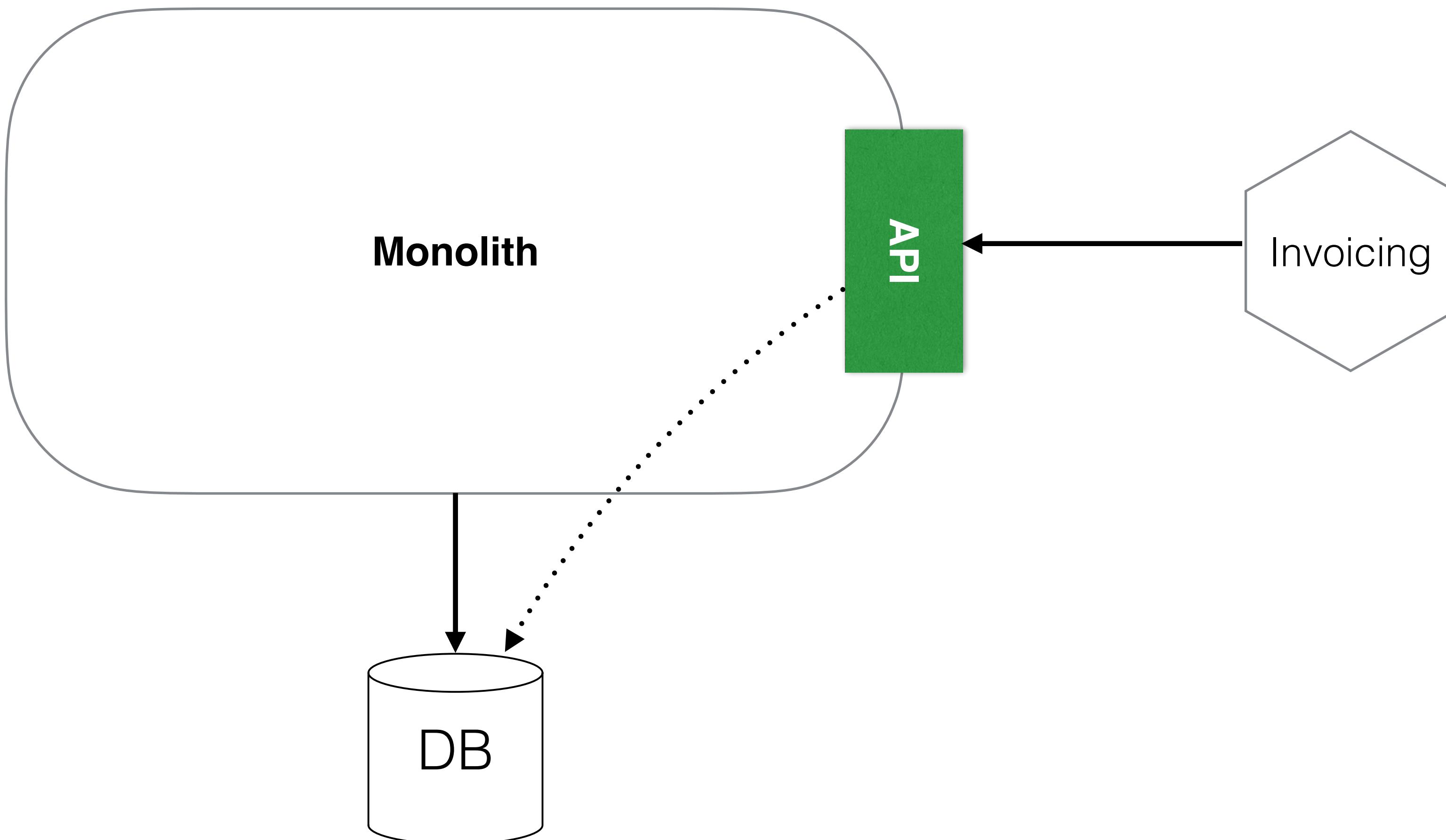


This makes sense if the data belongs to the functionality that remains in the monolith

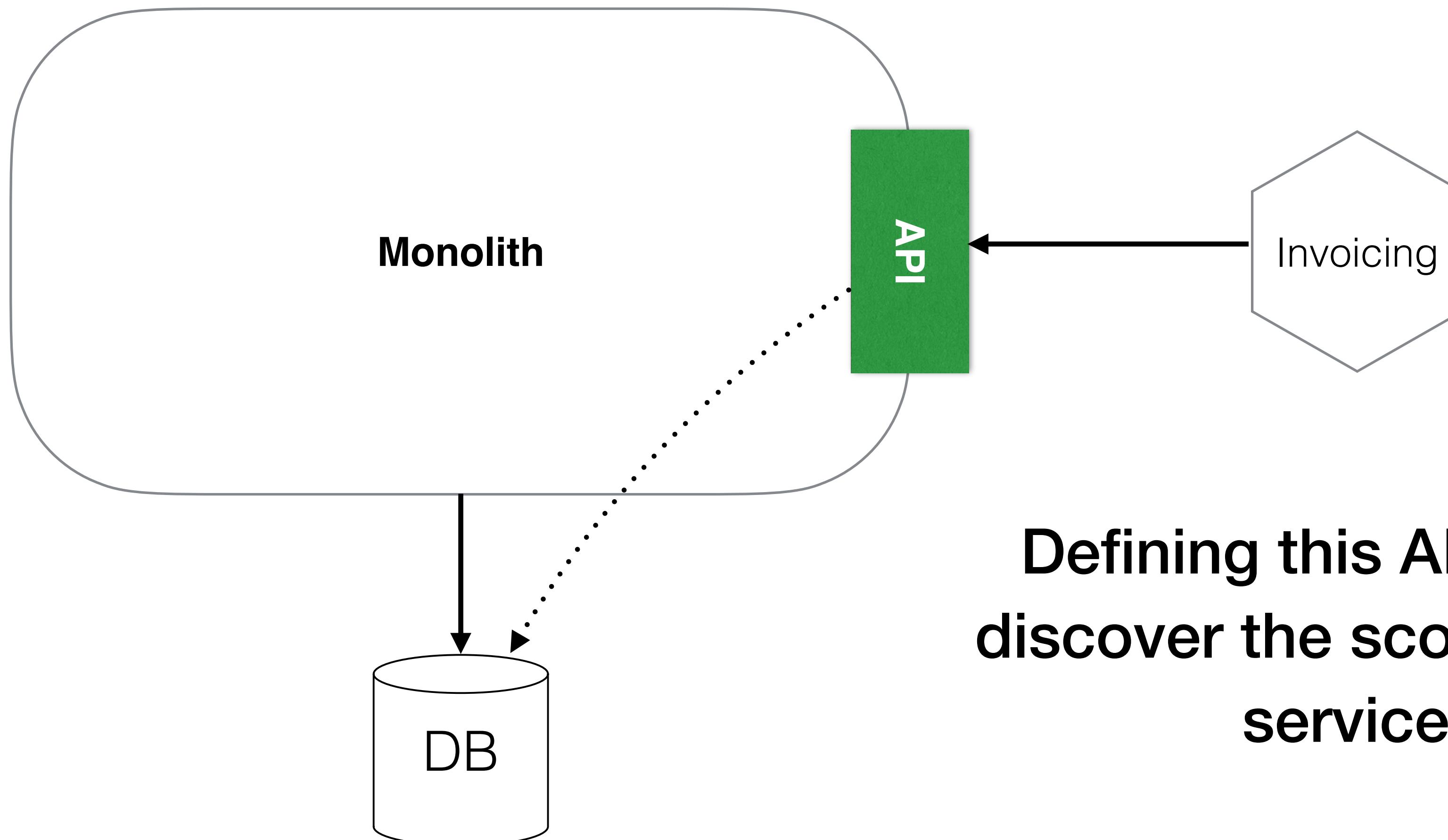
EXPOSE THE DATA VIA THE MONOLITH



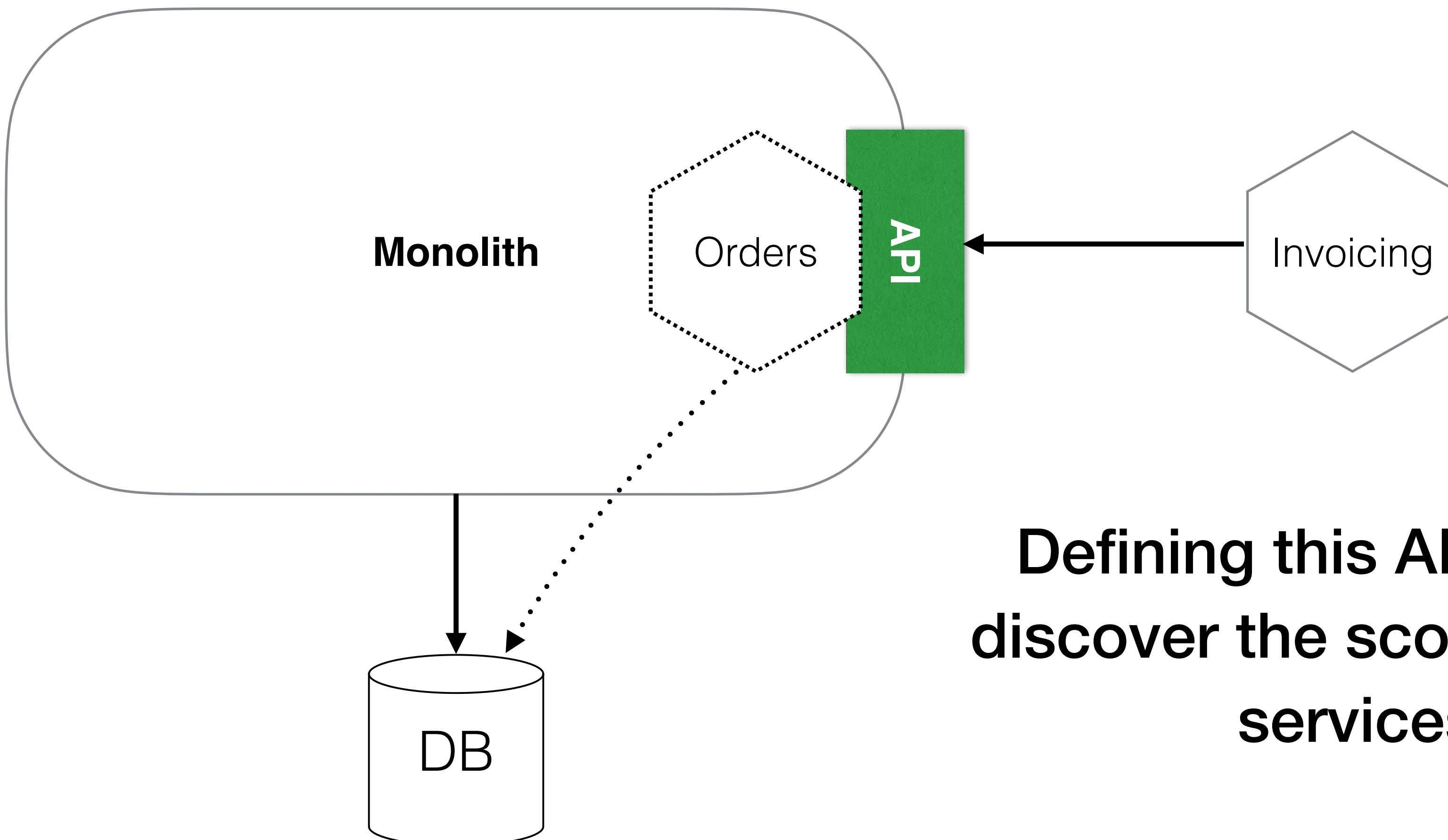
IDENTIFY FURTHER POTENTIAL MICROSERVICES



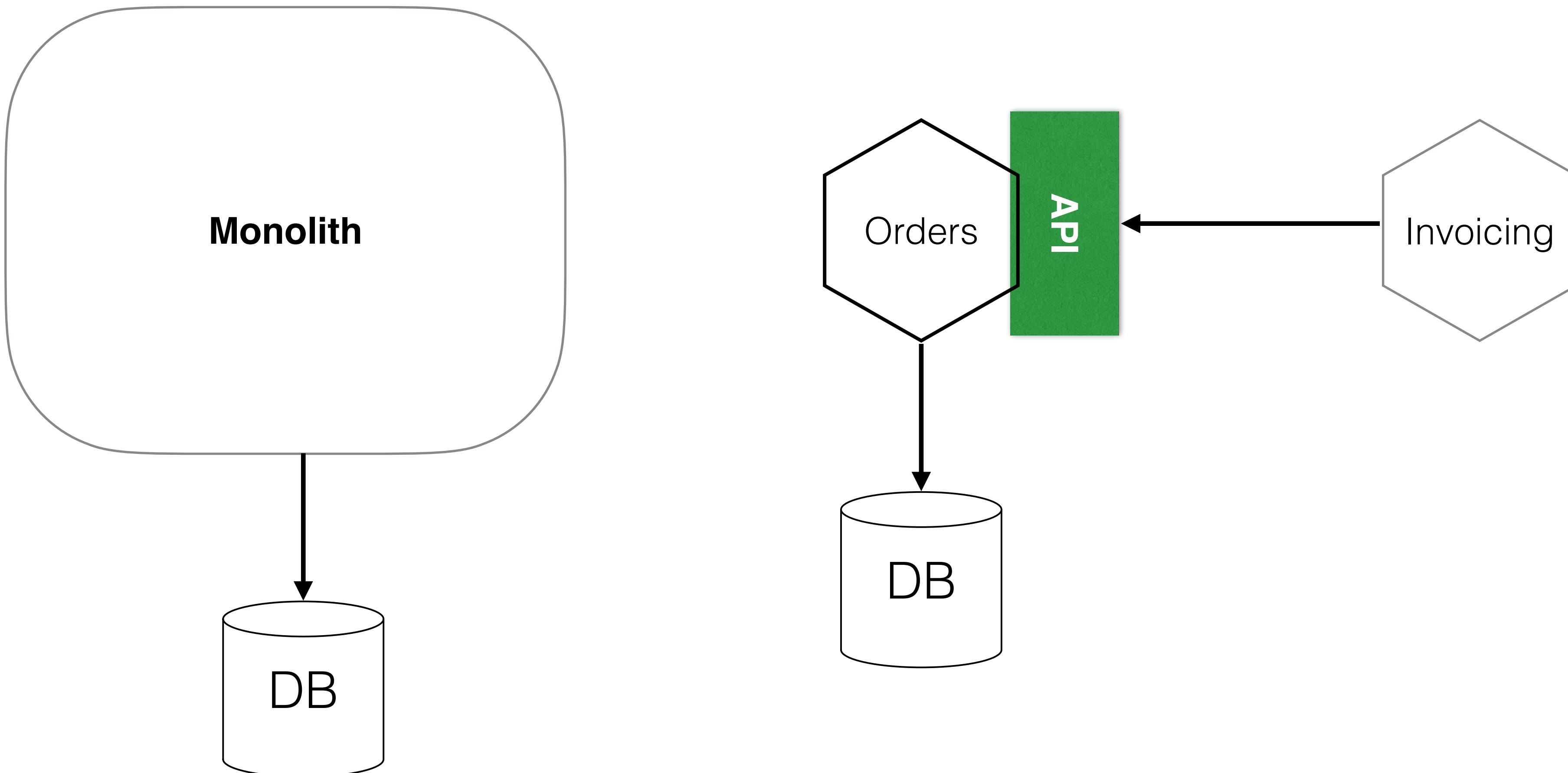
IDENTIFY FURTHER POTENTIAL MICROSERVICES



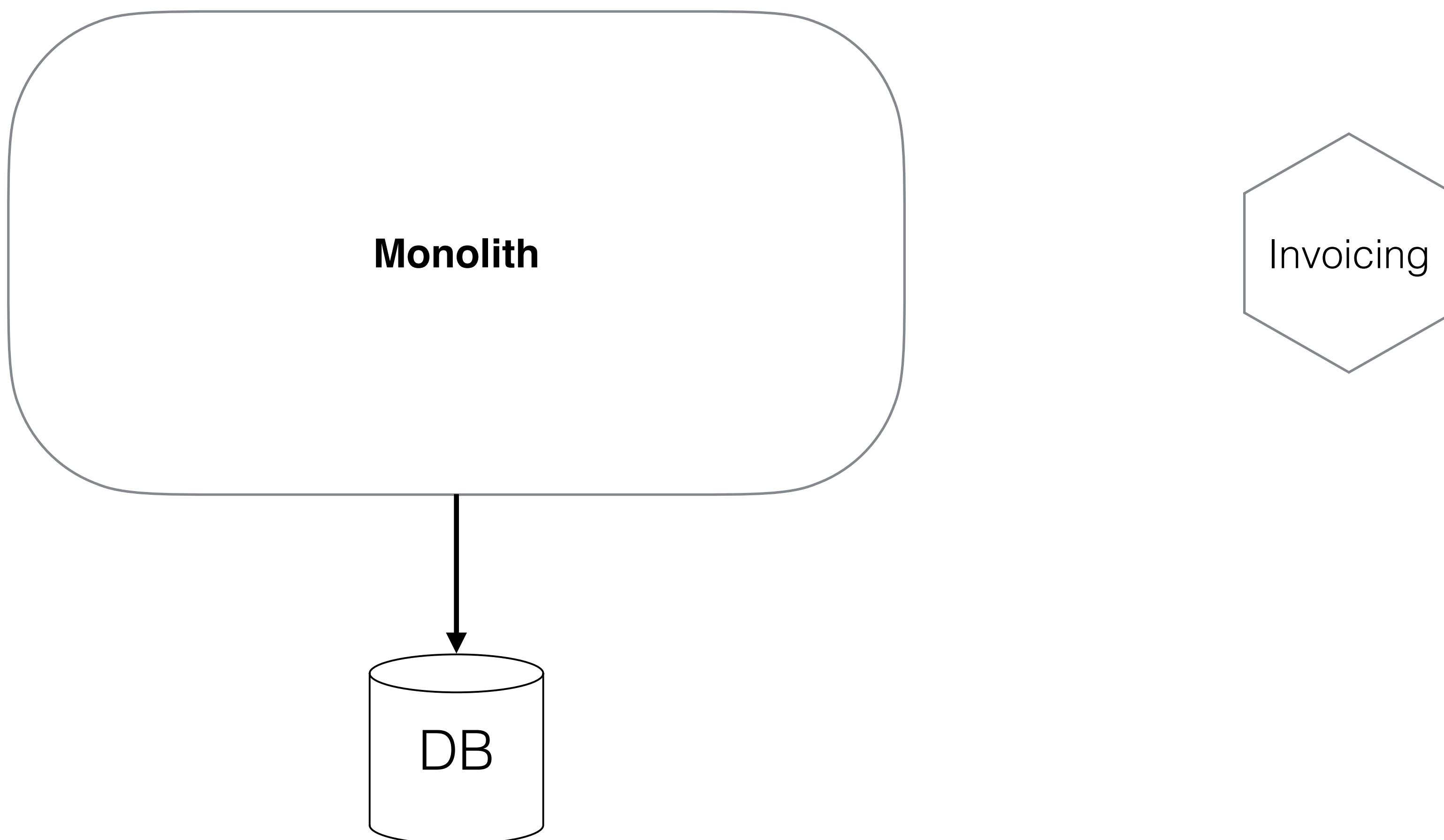
IDENTIFY FURTHER POTENTIAL MICROSERVICES



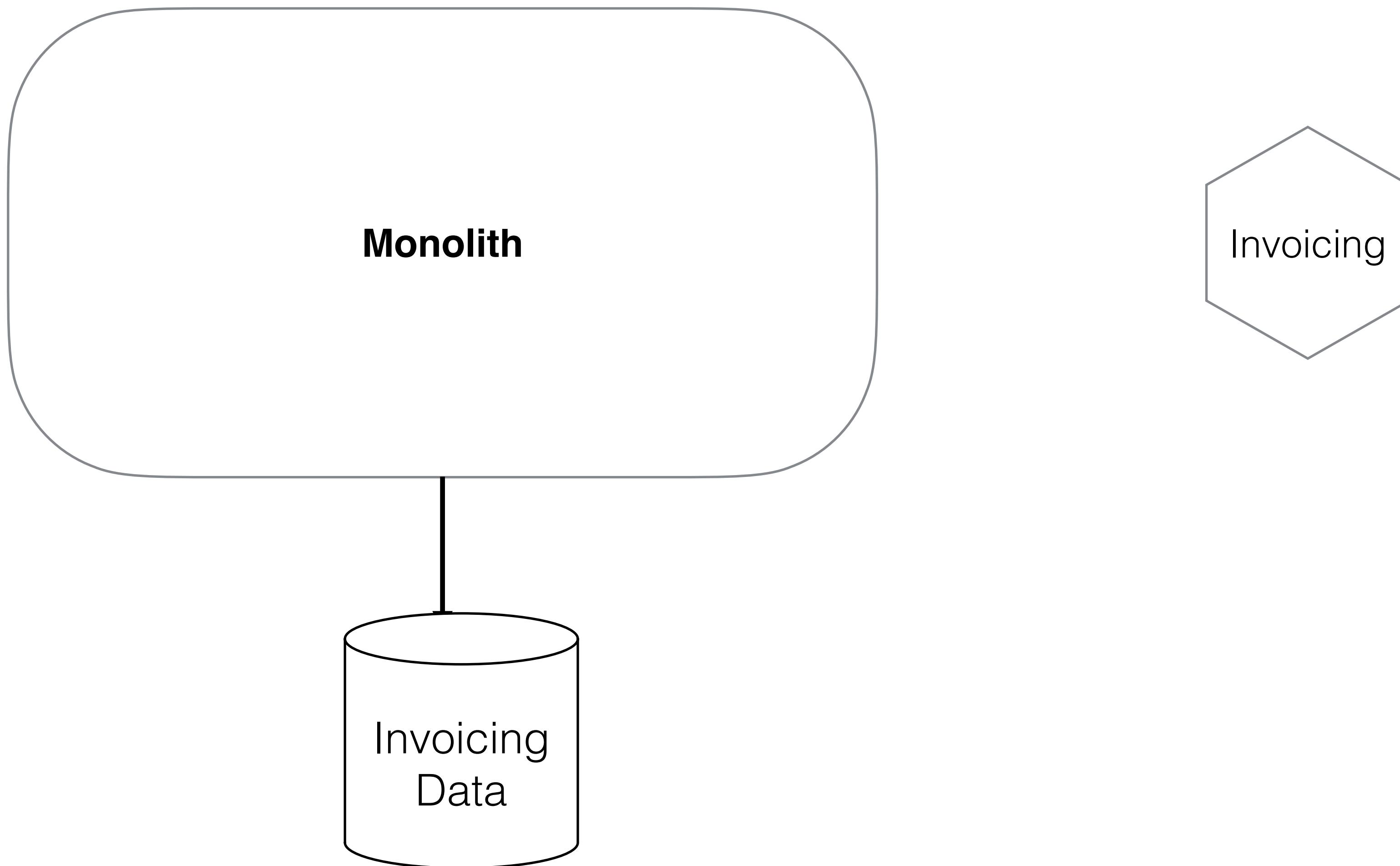
FINDING NEW MICROSERVICES?



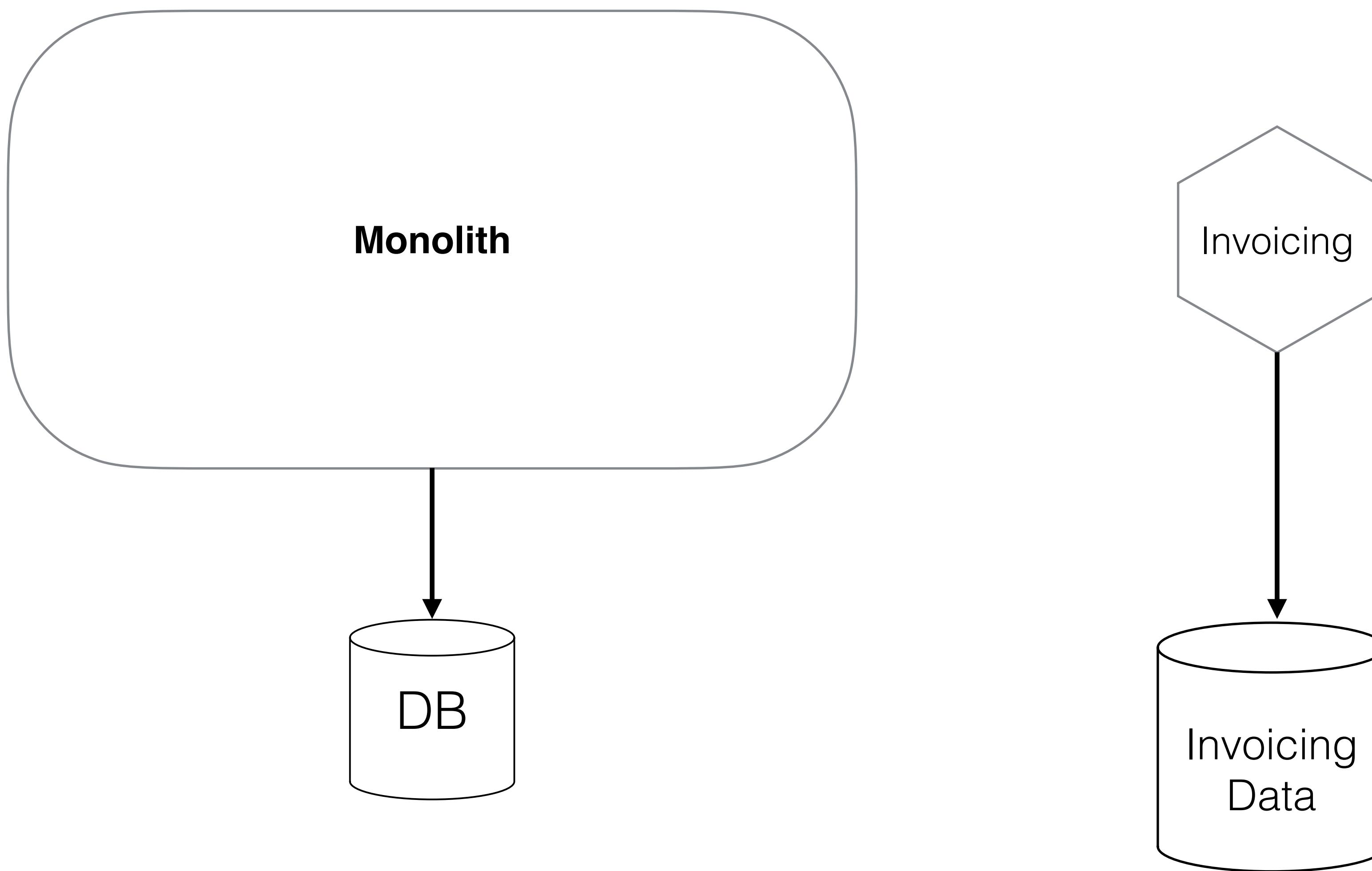
MOVE THE DATA!



MOVE THE DATA!

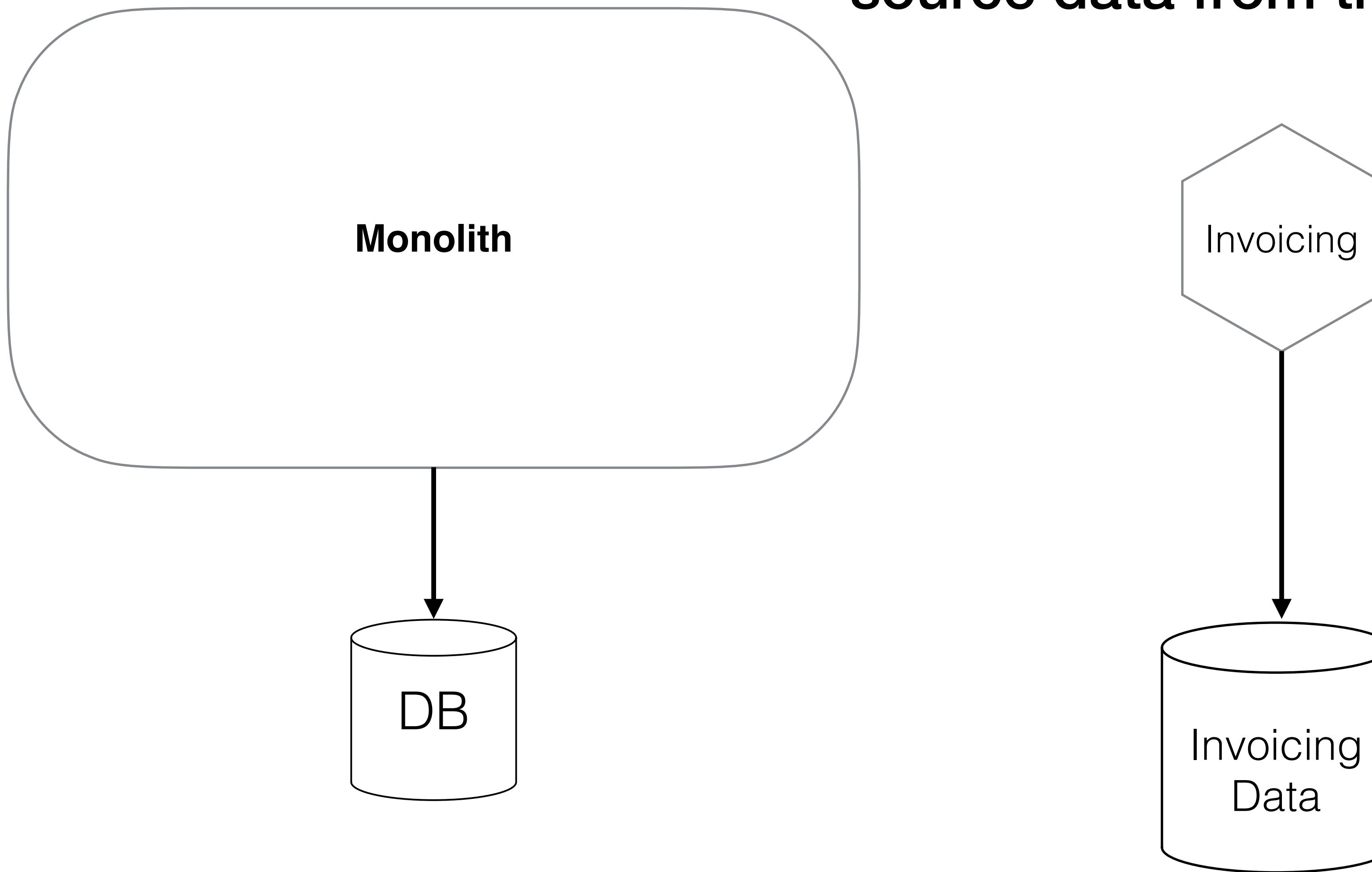


MOVE THE DATA!



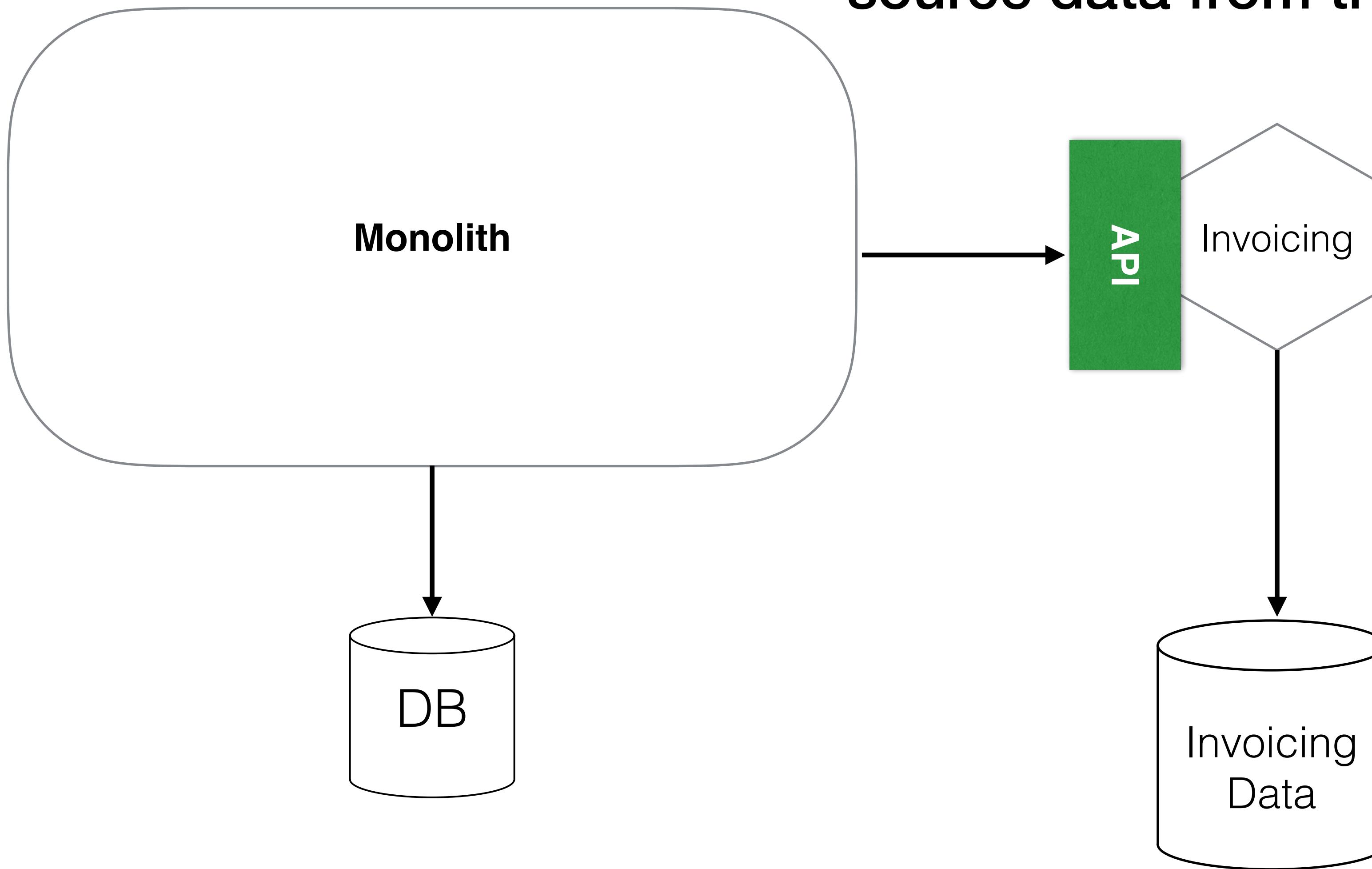
MOVE THE DATA!

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

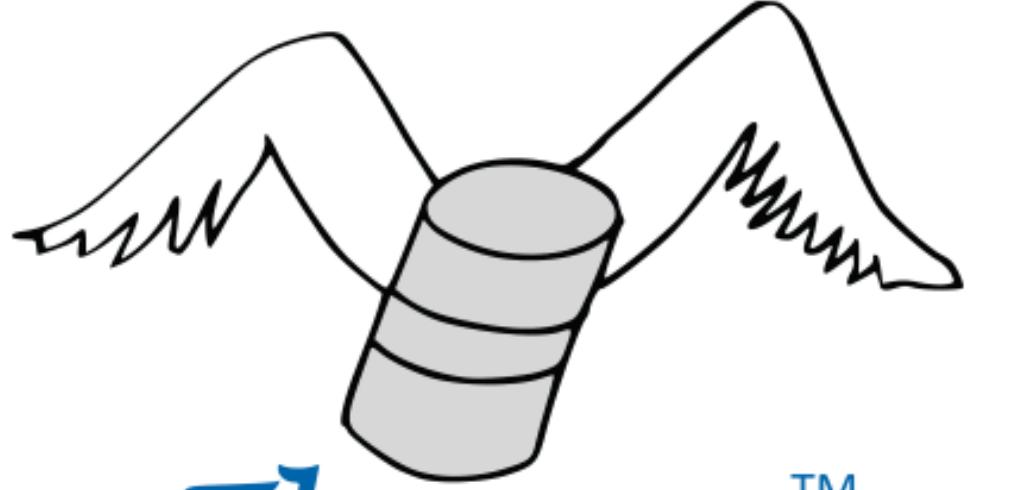
Shared Data Patterns

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 icon resting on stylized mountain peaks, with the word "Flyway" in blue script and "TM" in smaller letters, followed by "by Redgate". 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 says "Get Started with Flyway 6.4.4 →". At the bottom left, there are links for "Download - Release Notes - Apache v2 license" and "Works on: Windows, macOS, Linux, Docker, AWS Lambda, Google Cloud Functions". A "Stay updated with our newsletter" link is also present at the bottom.

Each DB refactoring is stored in version control

<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 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 a blue, lowercase, sans-serif font next to the cylinder, with a trademark symbol. 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 is a link to "Download - Release Notes - Apache v2 license" and a note that it "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". There is also a section for staying updated with the newsletter.

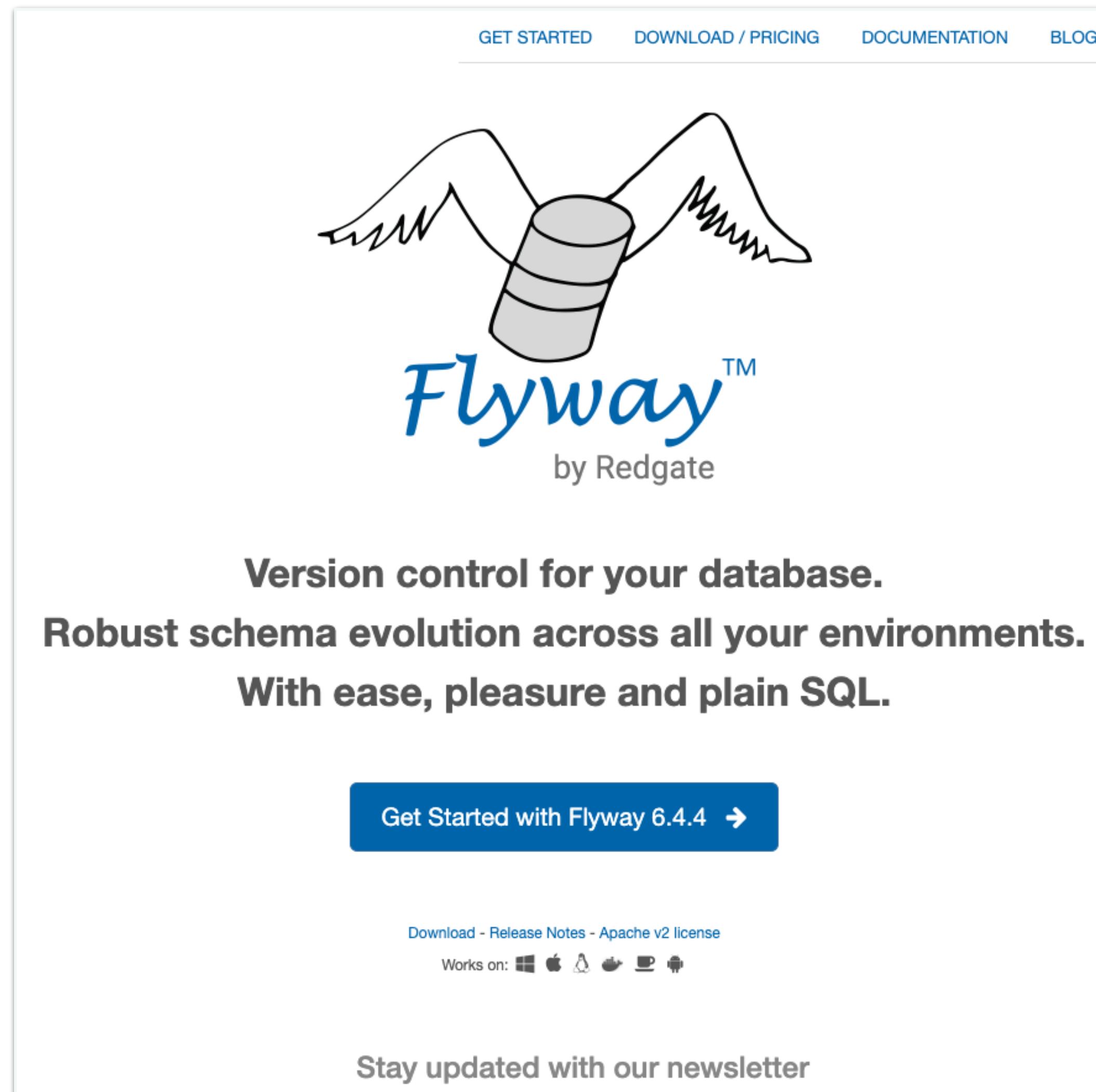
Each DB refactoring is stored in version control

Can be run in a safe, deterministic, idempotent manner

<https://flywaydb.org/>

@samnewman

DATA REFACTORING TOOLS



The screenshot shows the Flyway website homepage. At the top, there is a navigation bar with links: GET STARTED, DOWNLOAD / PRICING, DOCUMENTATION, and BLOG. Below the navigation is the Flyway logo, which features a grey cylinder icon resting on stylized mountain peaks, with the word "Flyway" in blue script and "TM" in small caps, followed by "by Redgate". 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 says "Get Started with Flyway 6.4.4 →". At the bottom of the page, there are links for "Download - Release Notes - Apache v2 license" and "Works on: Windows, macOS, Linux, Docker, AWS Lambda, Google Cloud Functions". 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/>

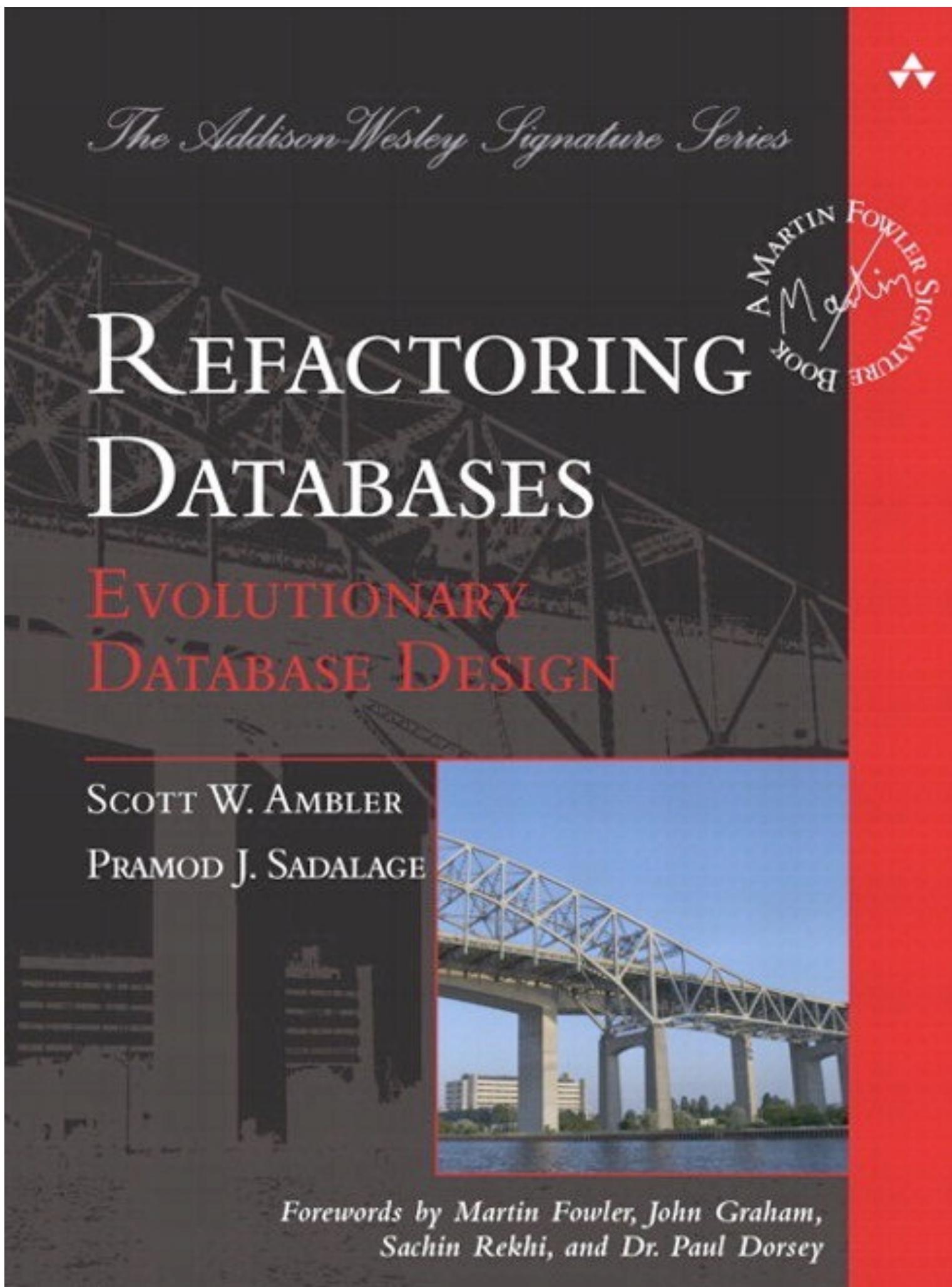
@samnewman

POLL: HAVE YOU USED A DB REFACTORING TOOL?

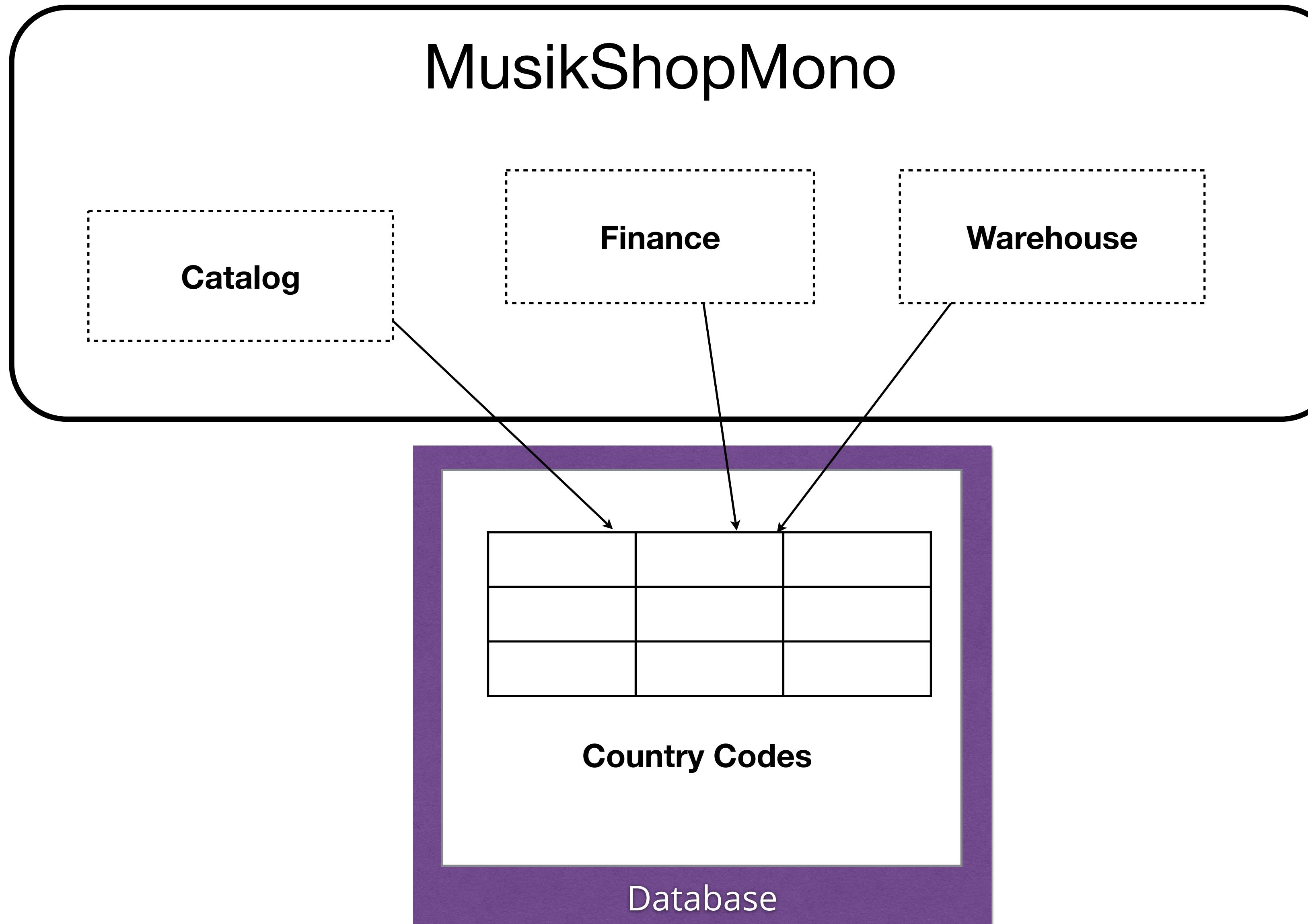
Yes

No

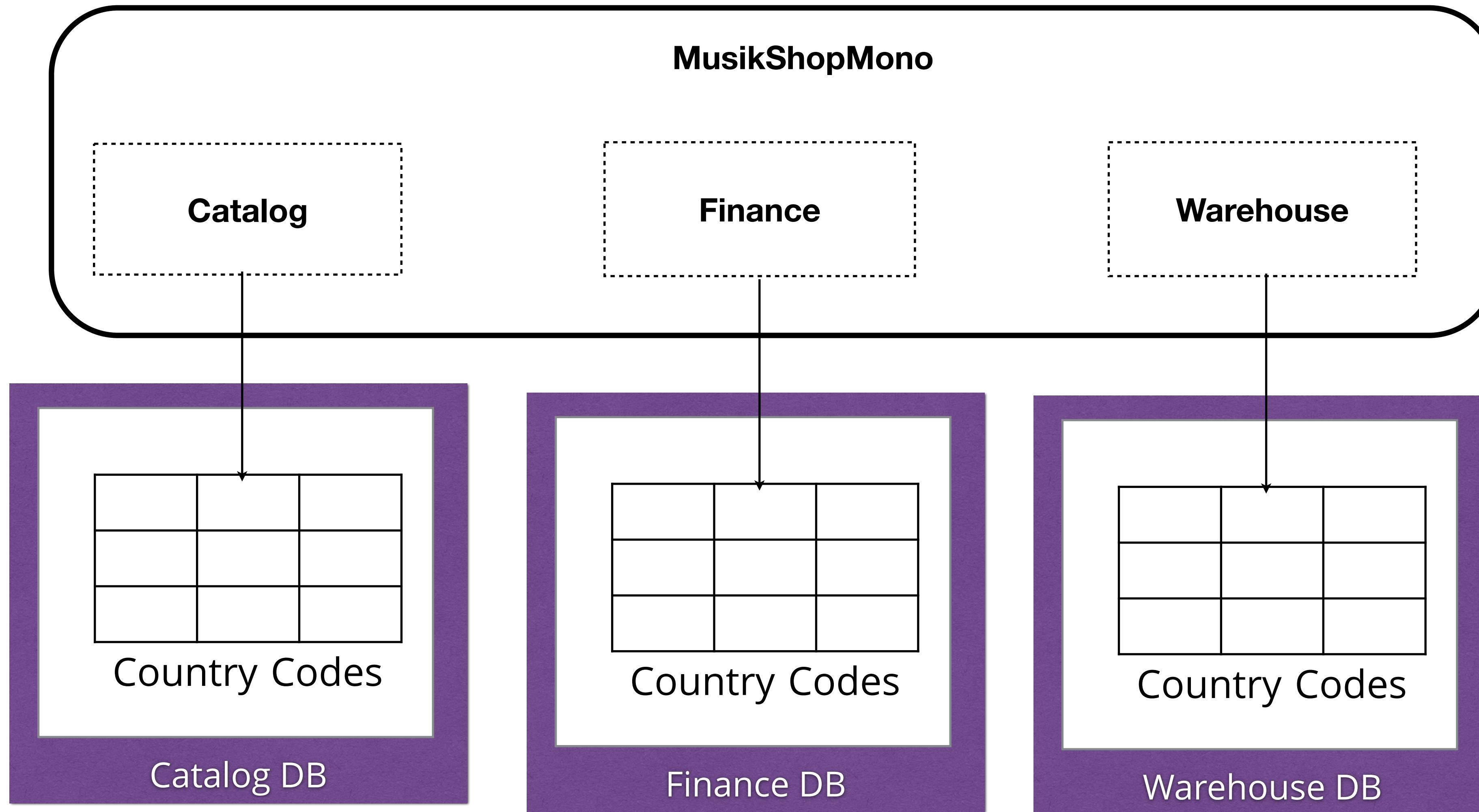
REFACTORING DATABASES - FURTHER READING



STATIC REFERENCE DATA



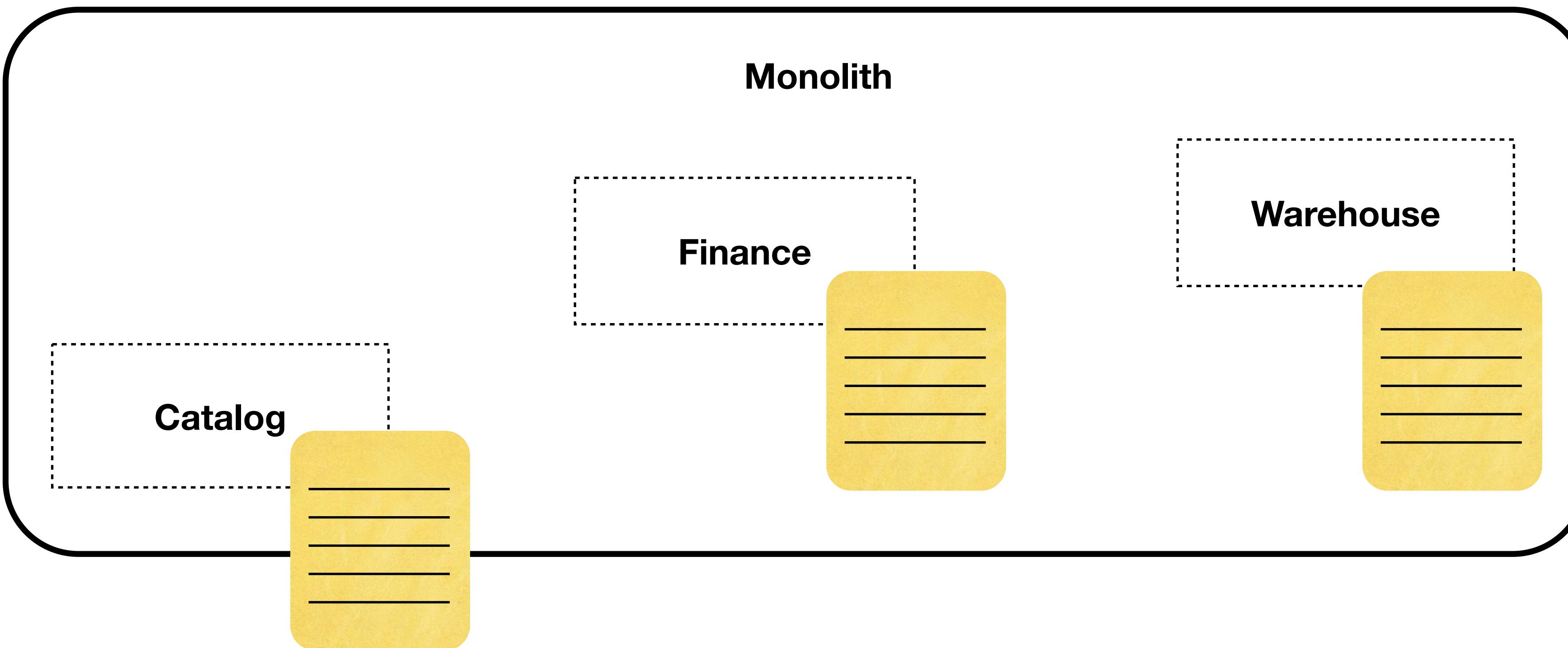
DUPLICATE THE DATA?



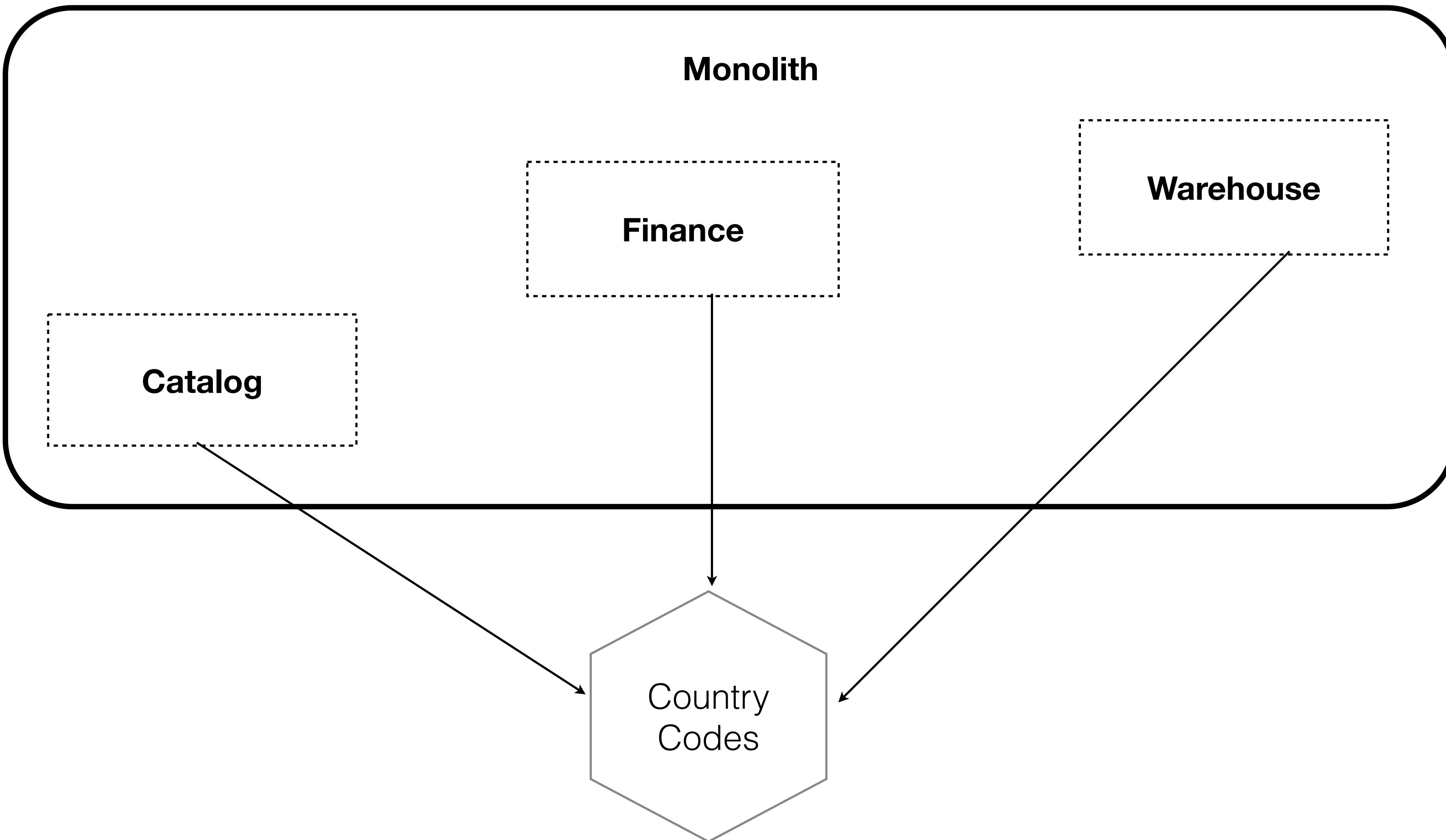
Duplication isn't always bad...

...but watch for inconsistency

MOVE DATA INTO CONFIGURATION, OR LIBRARIES



USE A SERVICE



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

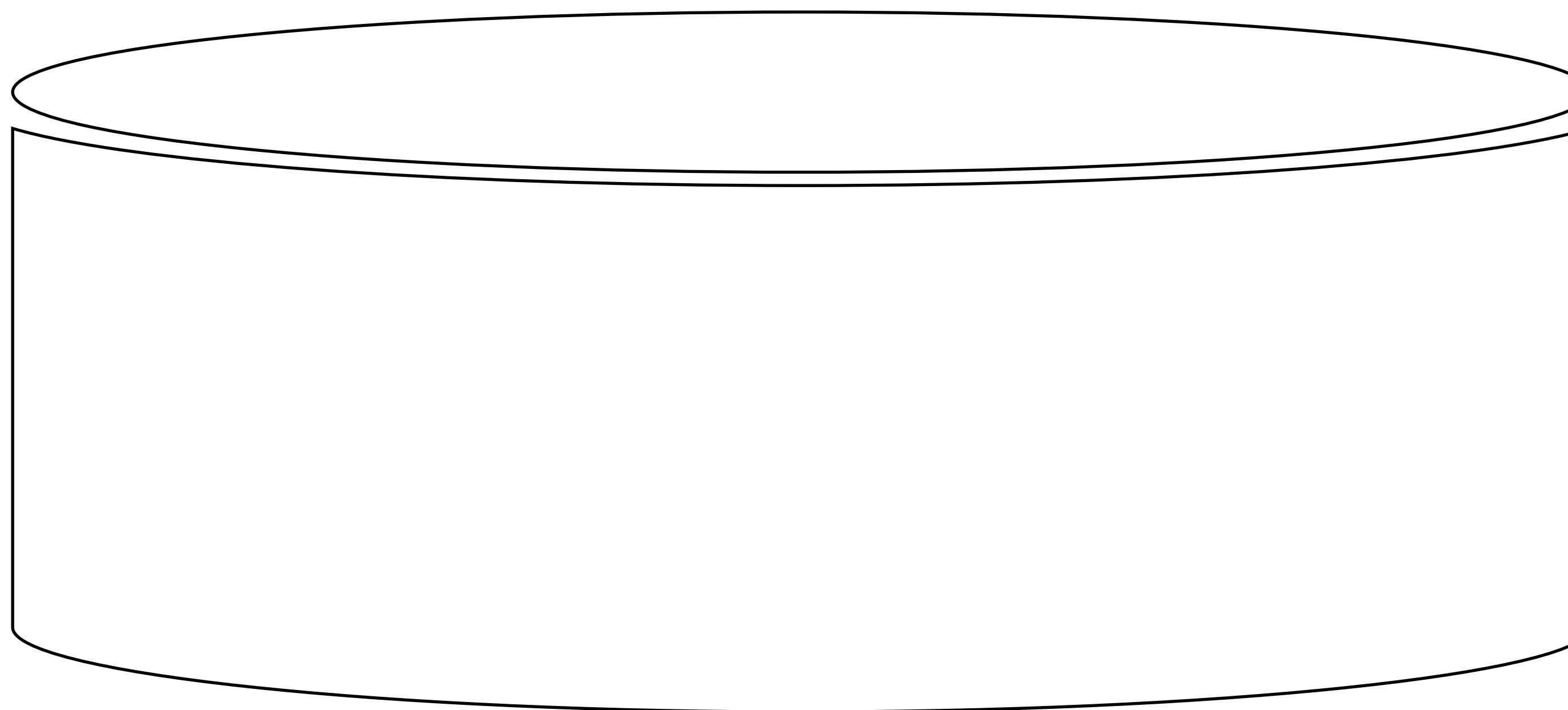
If you are ok with potentially having different sets of data

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

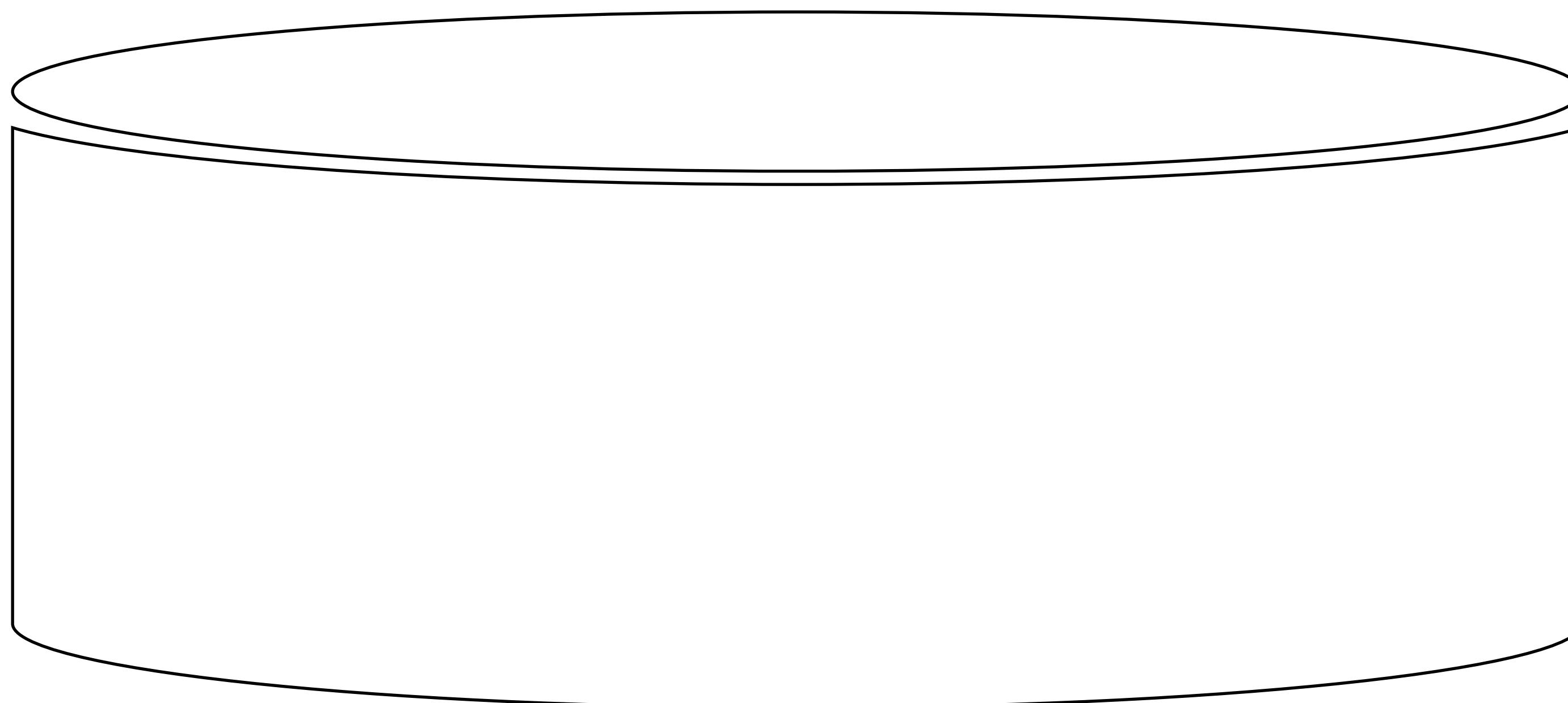
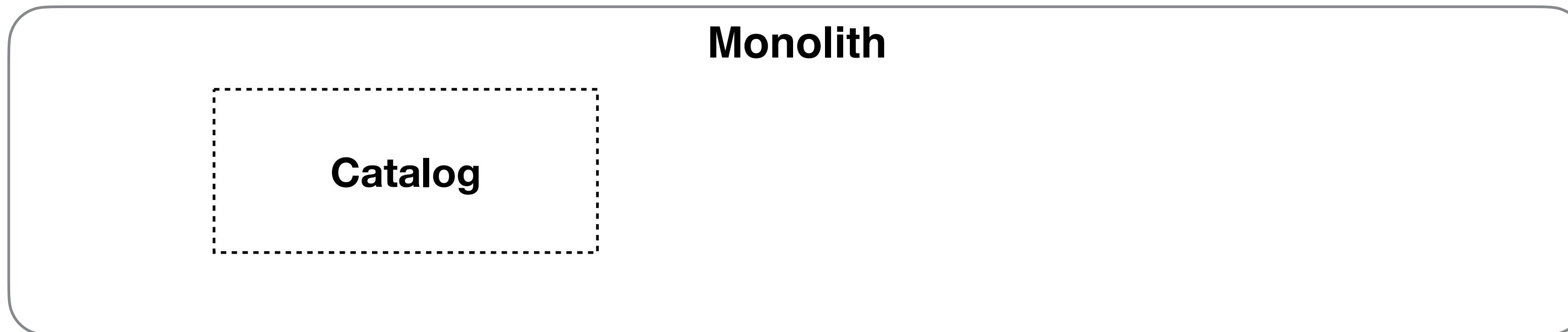
For larger datasets, copy data into each microservices' DB

TOP CHARTS!

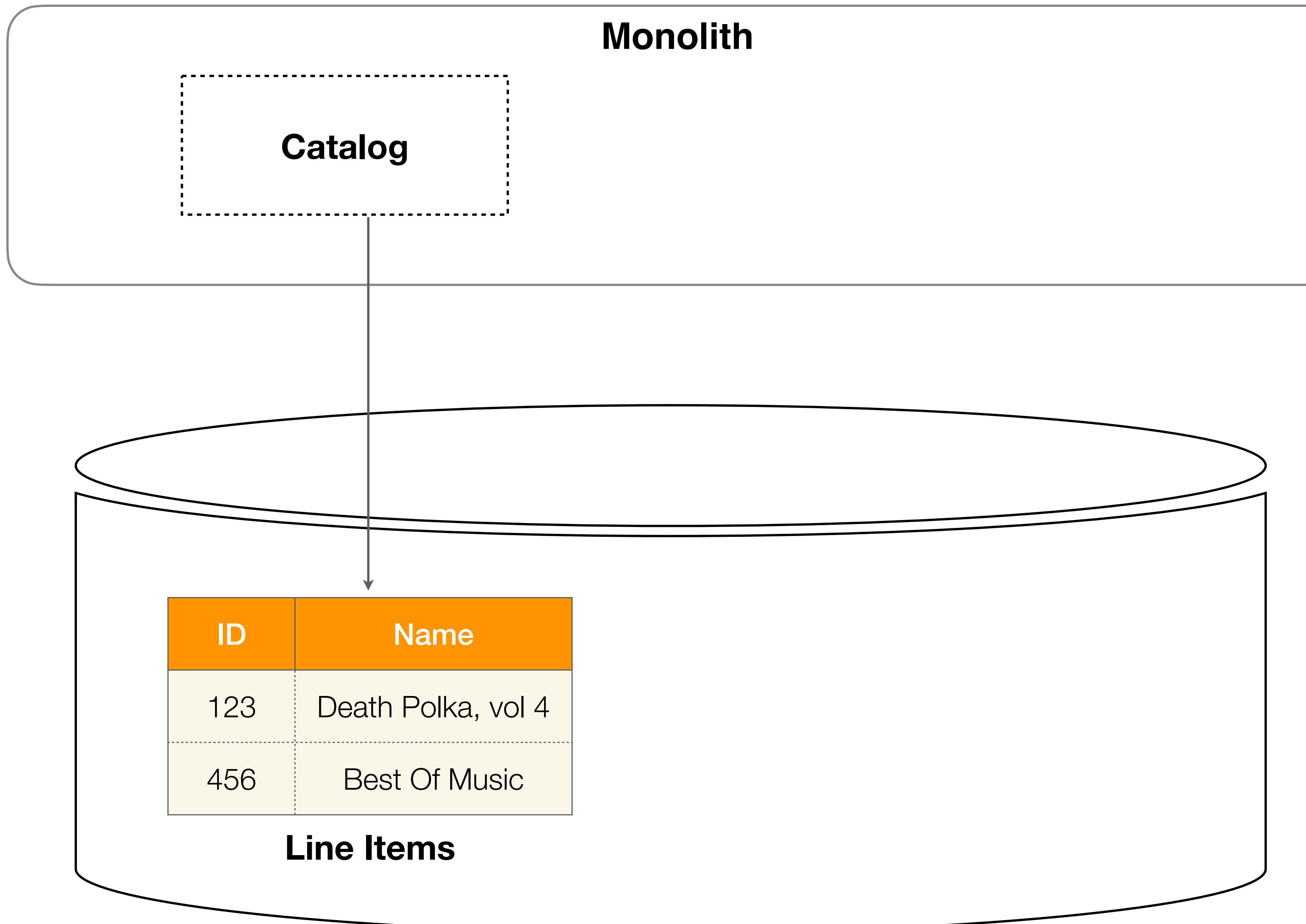
Monolith



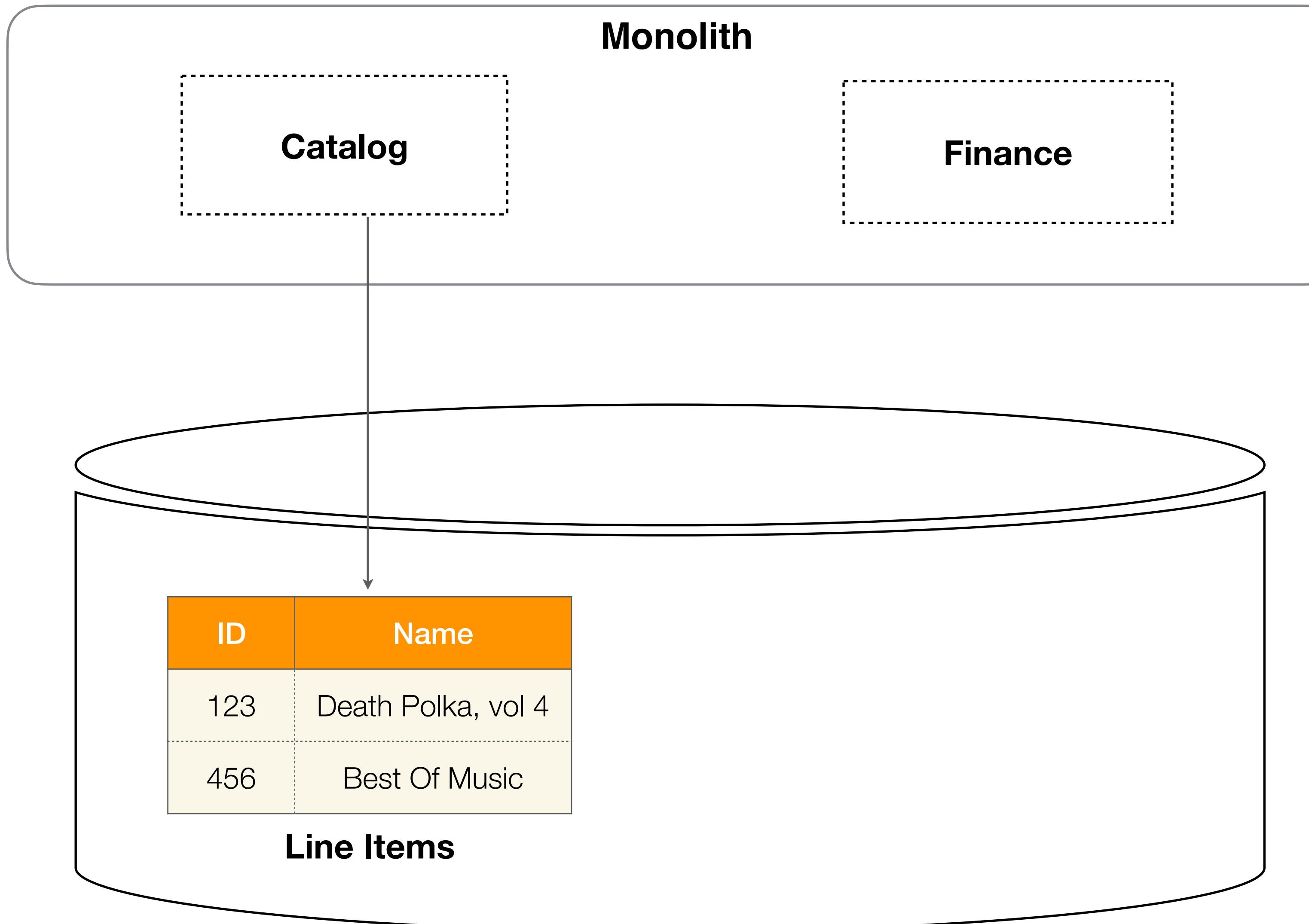
TOP CHARTS!



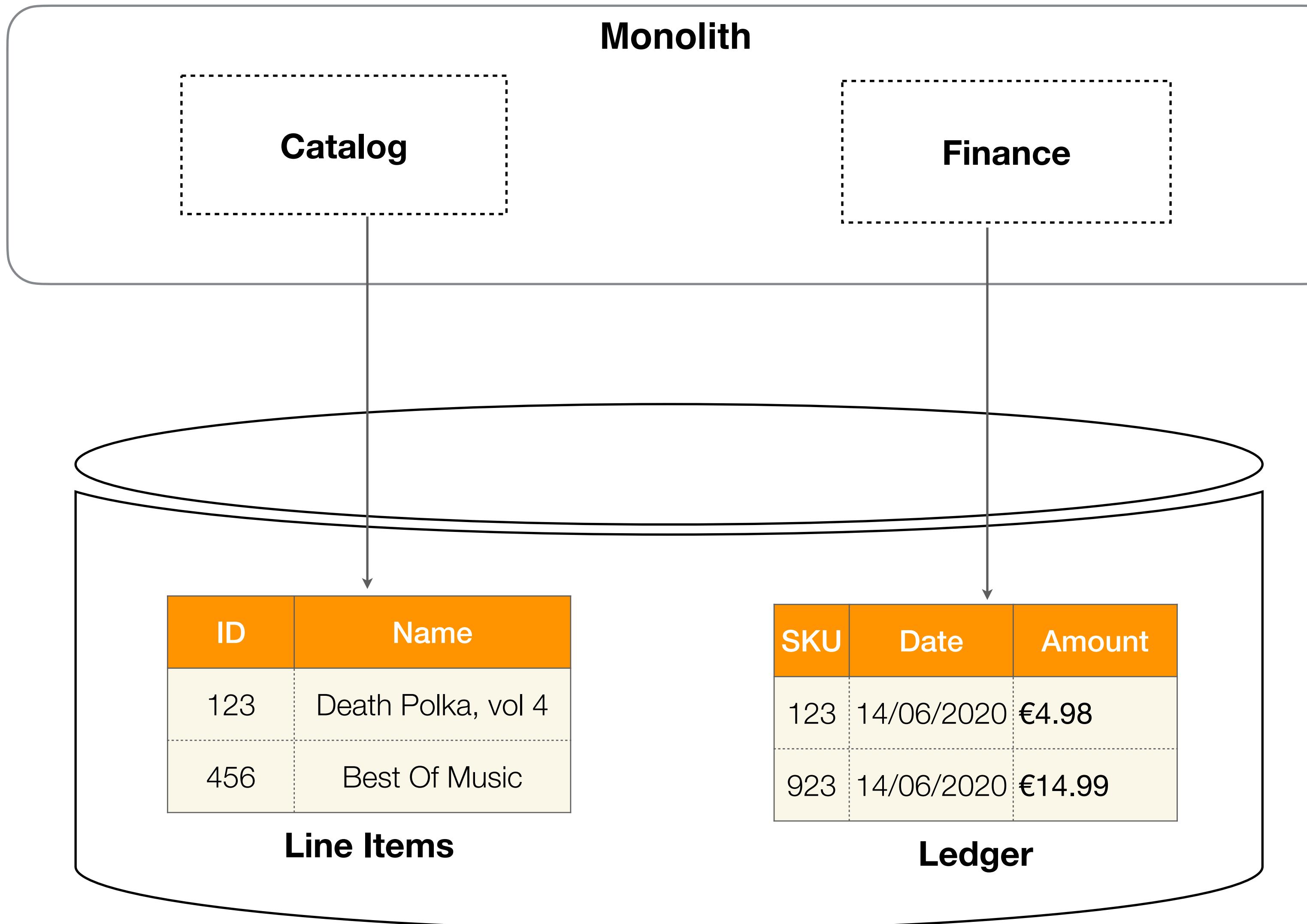
TOP CHARTS!



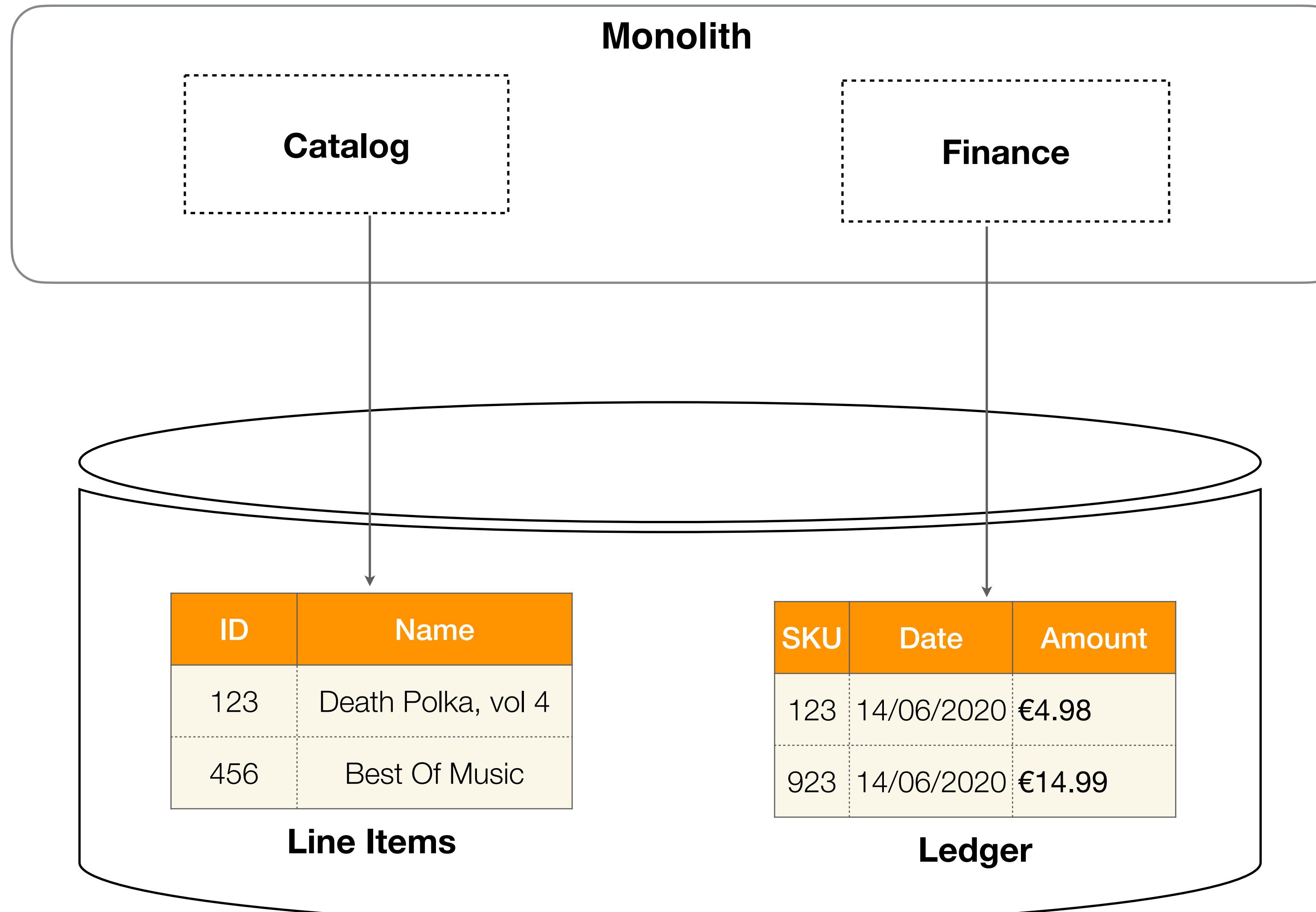
TOP CHARTS!



TOP CHARTS!



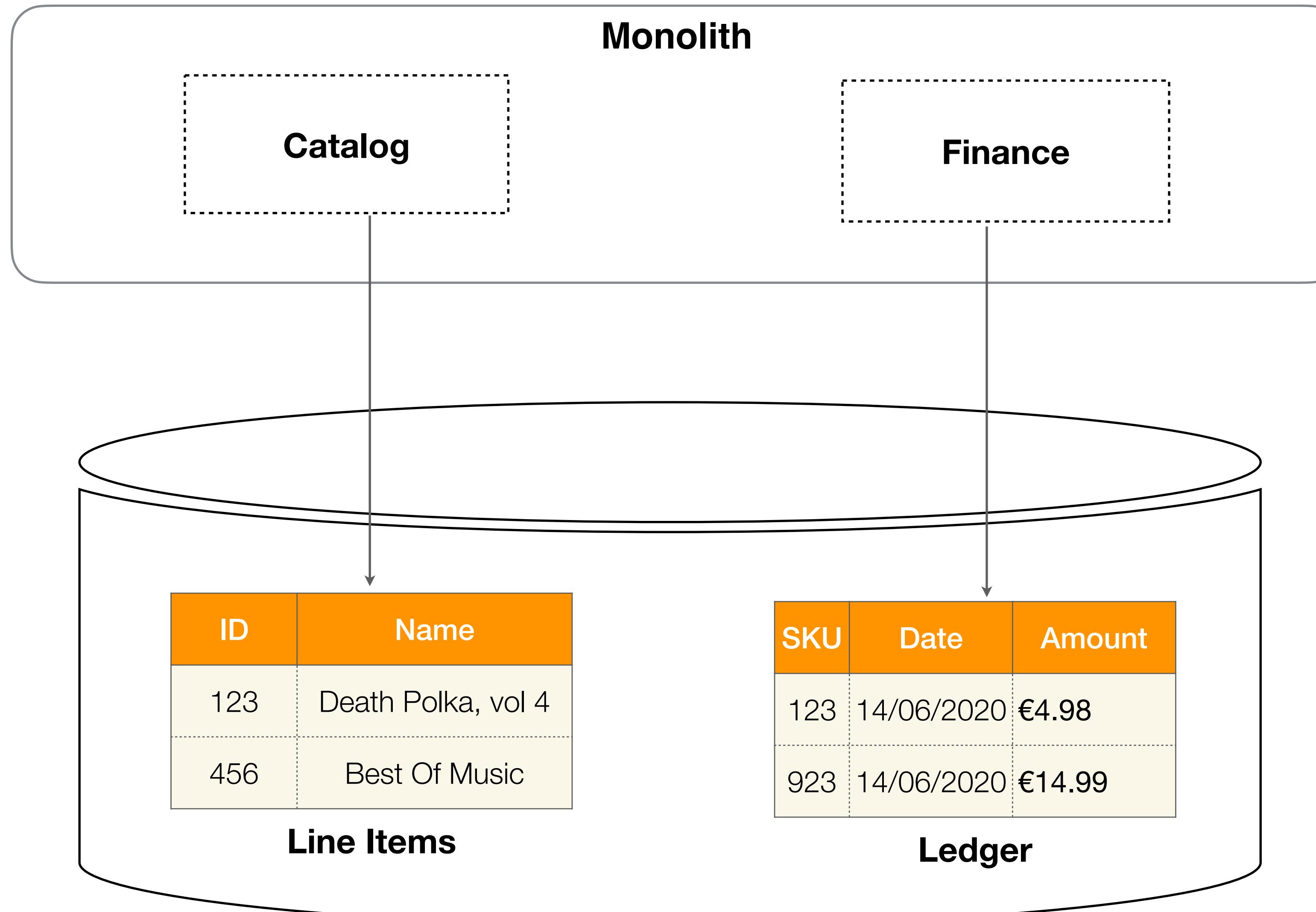
TOP CHARTS!



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!

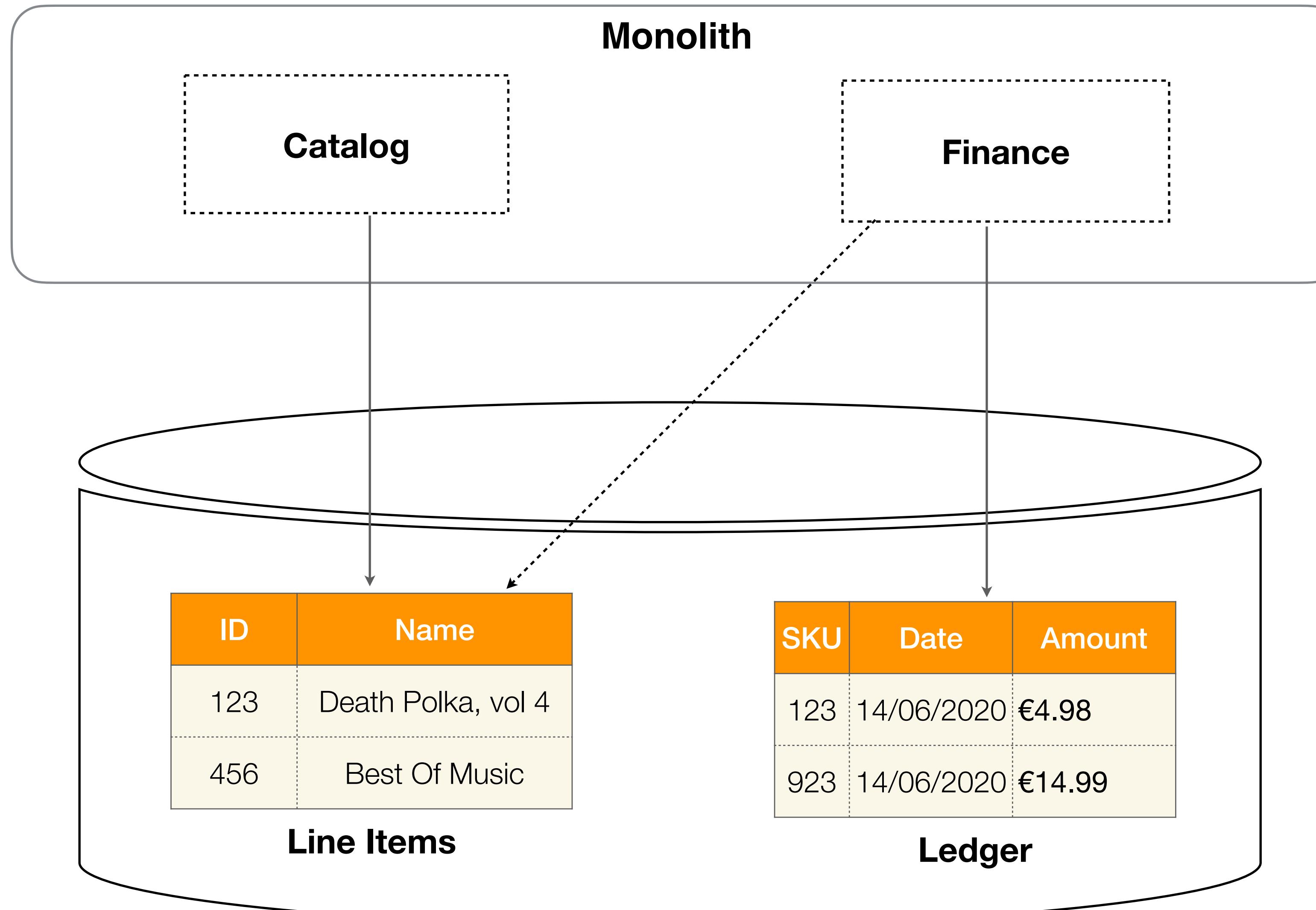


Best Sellers This Week!

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!

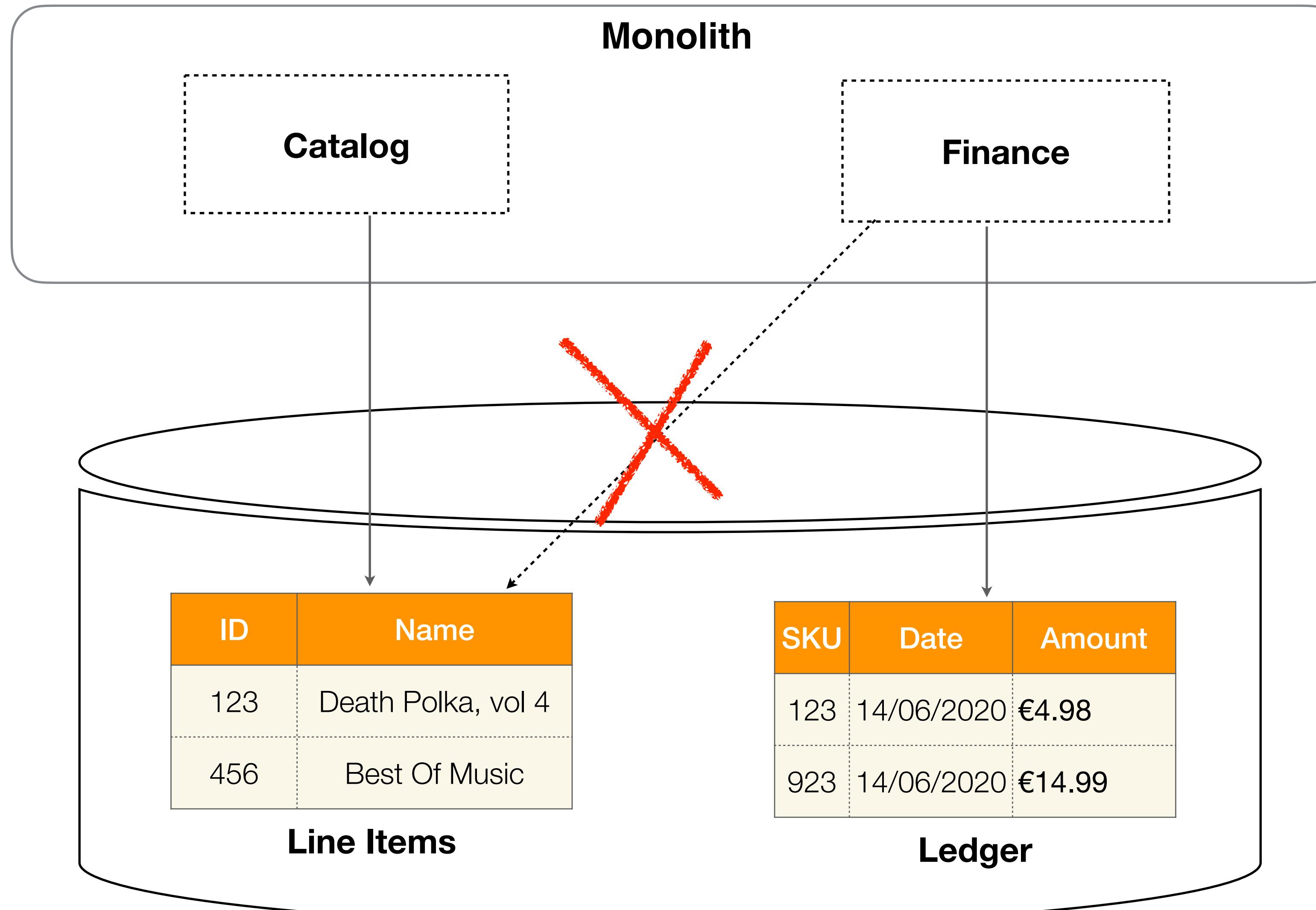


Best Sellers This Week!

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!

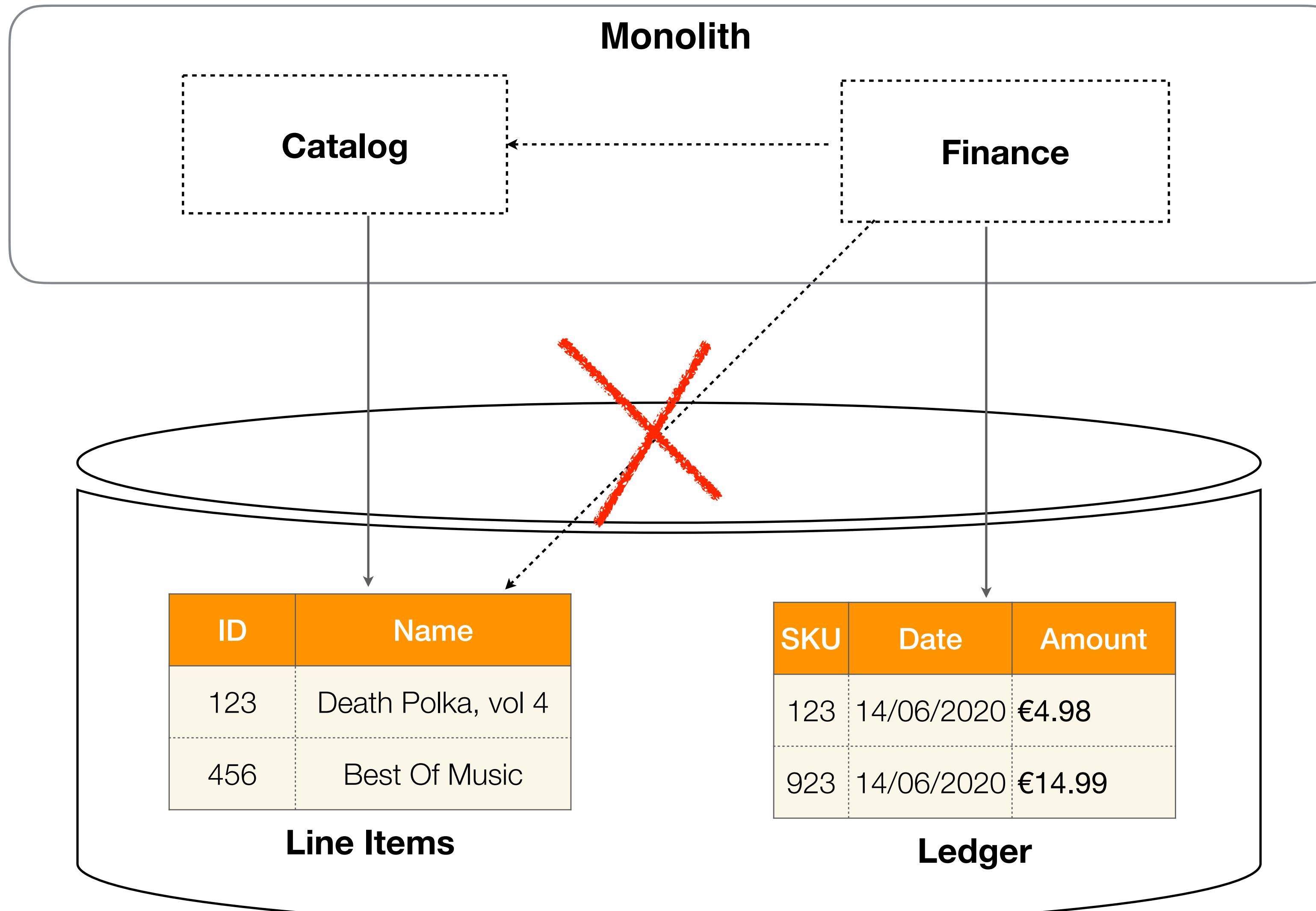


Best Sellers This Week!

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!

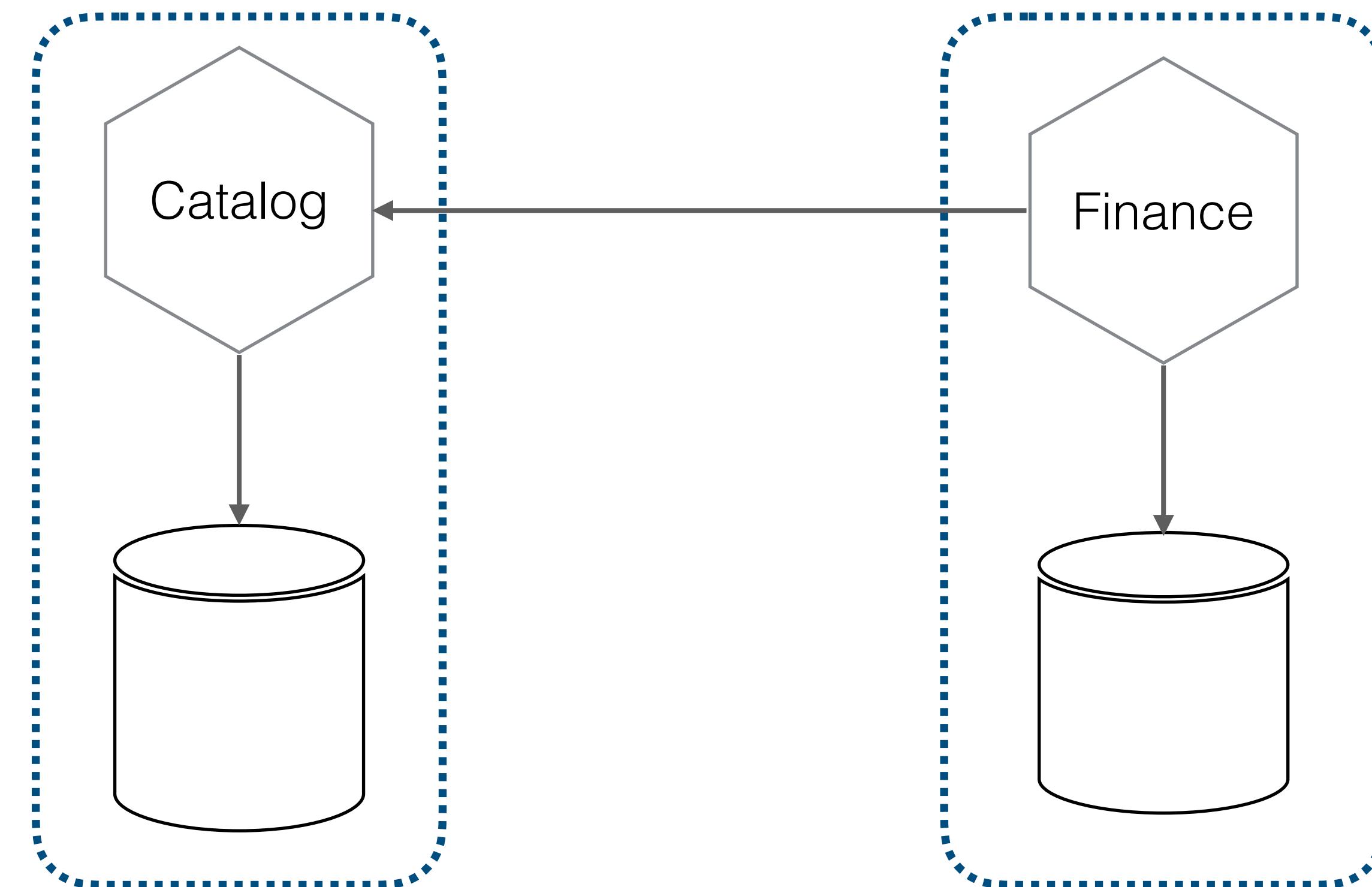


Best Sellers This Week!

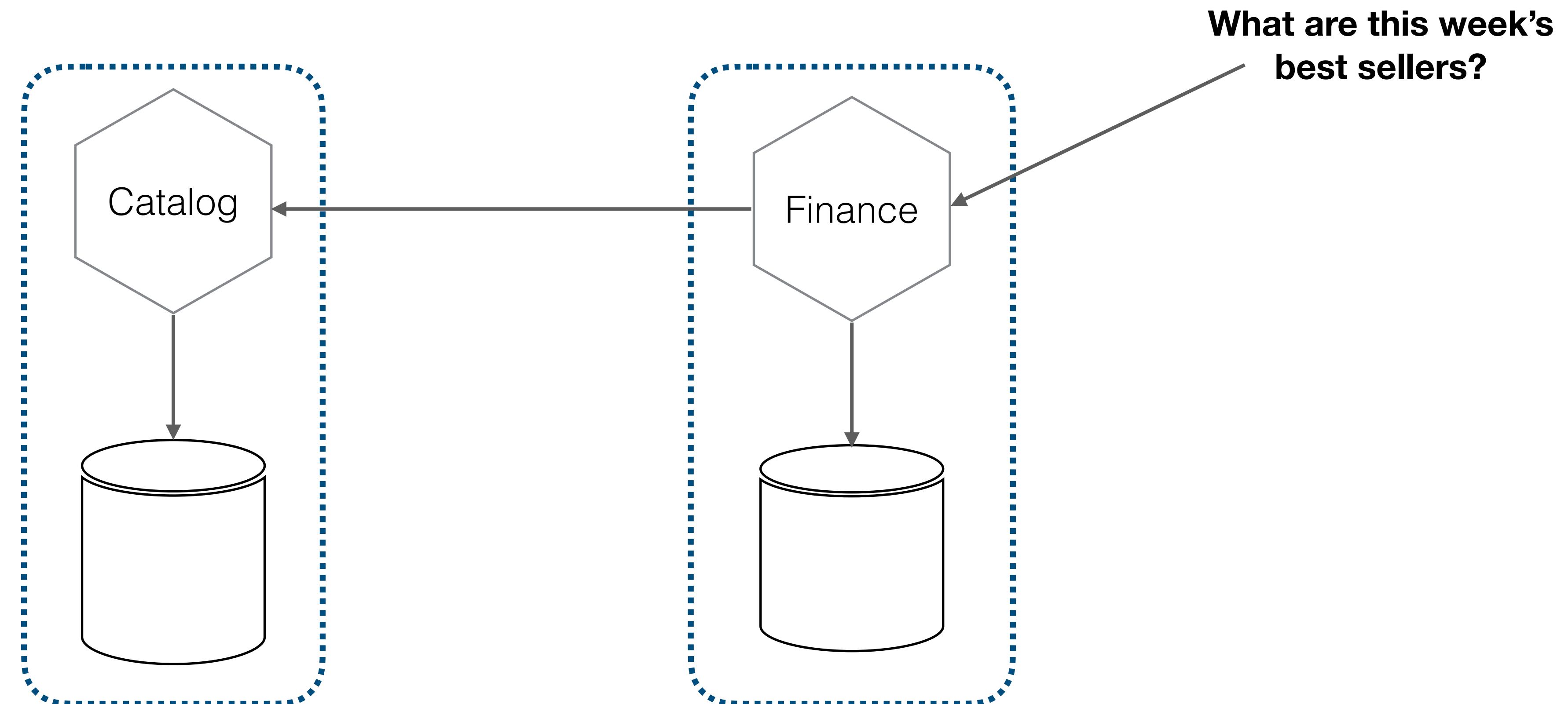
1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

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

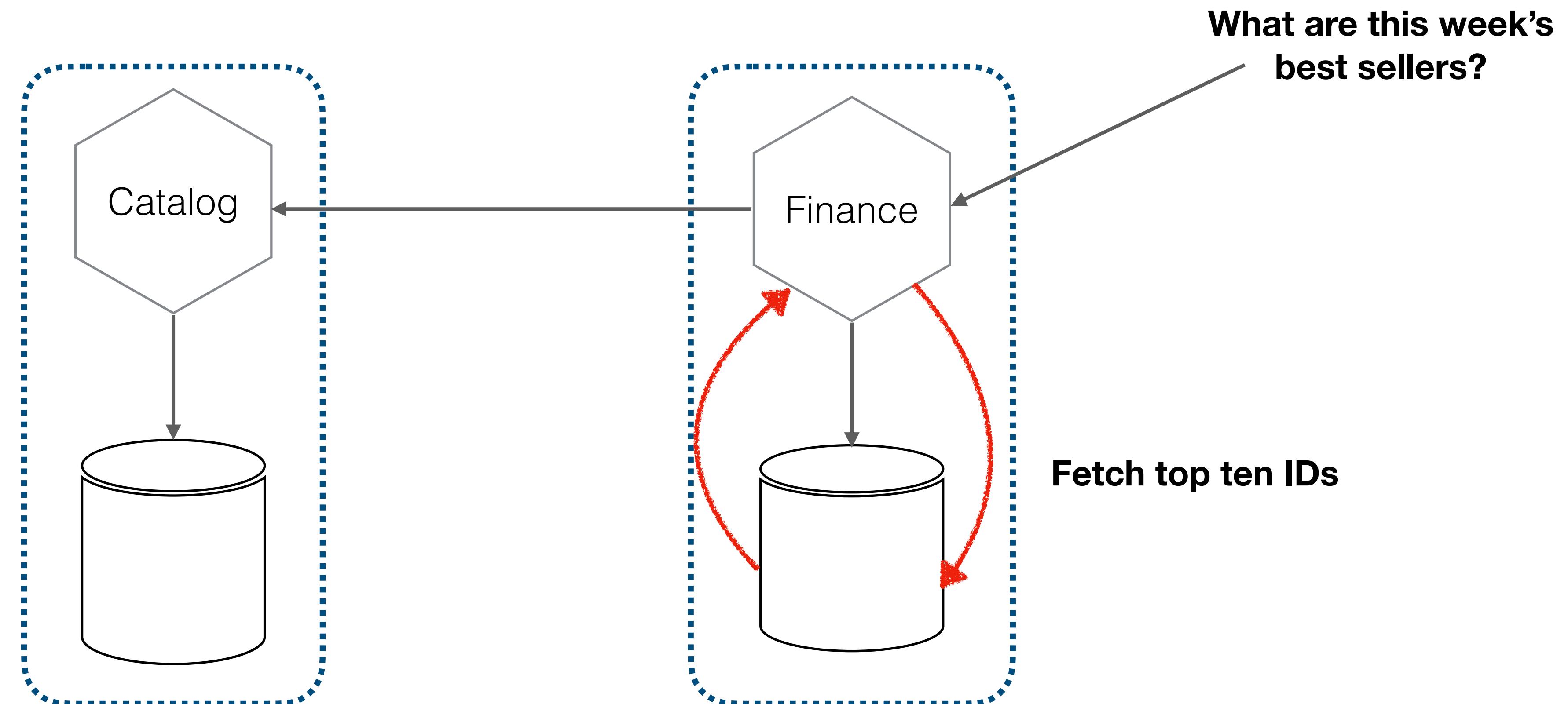
JOINS AT THE SERVICES TIER



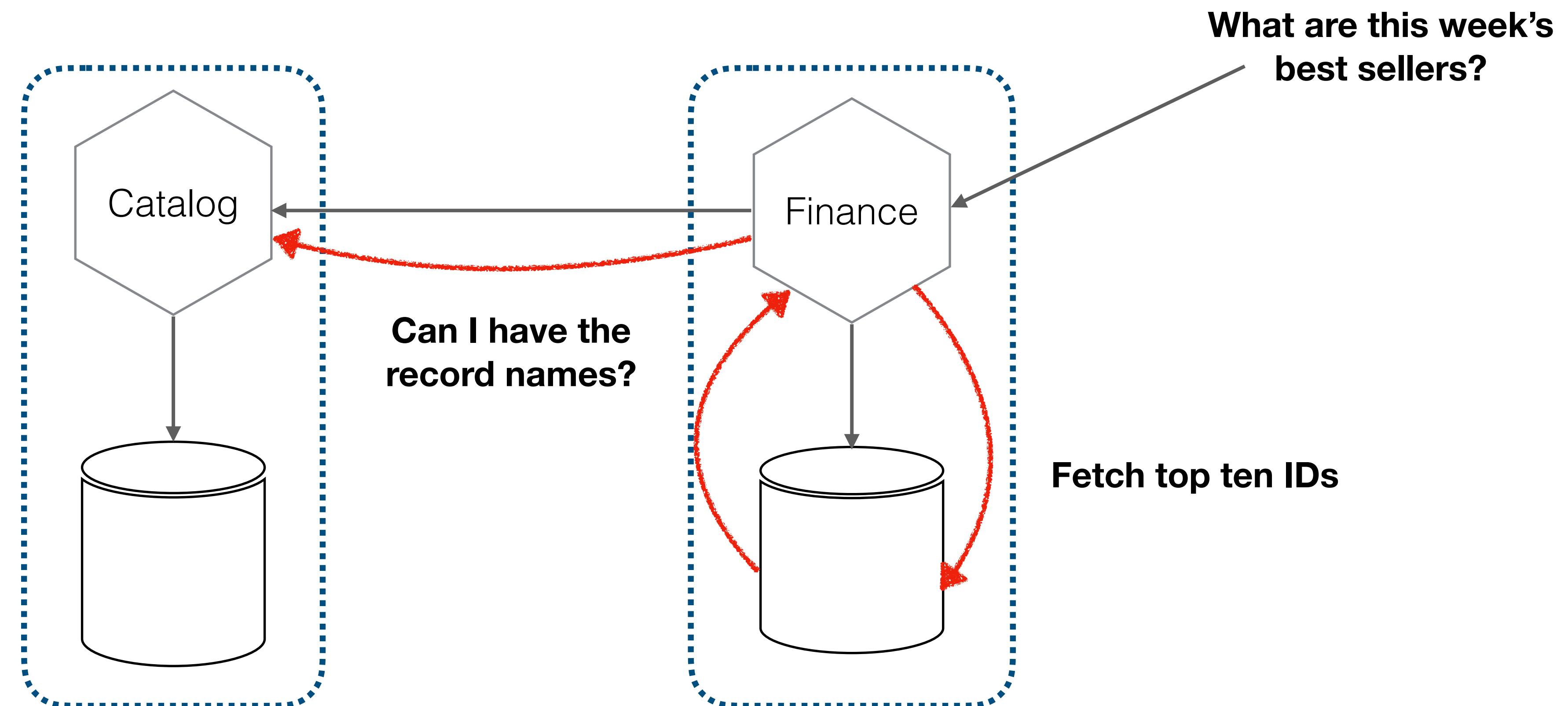
JOINS AT THE SERVICES TIER



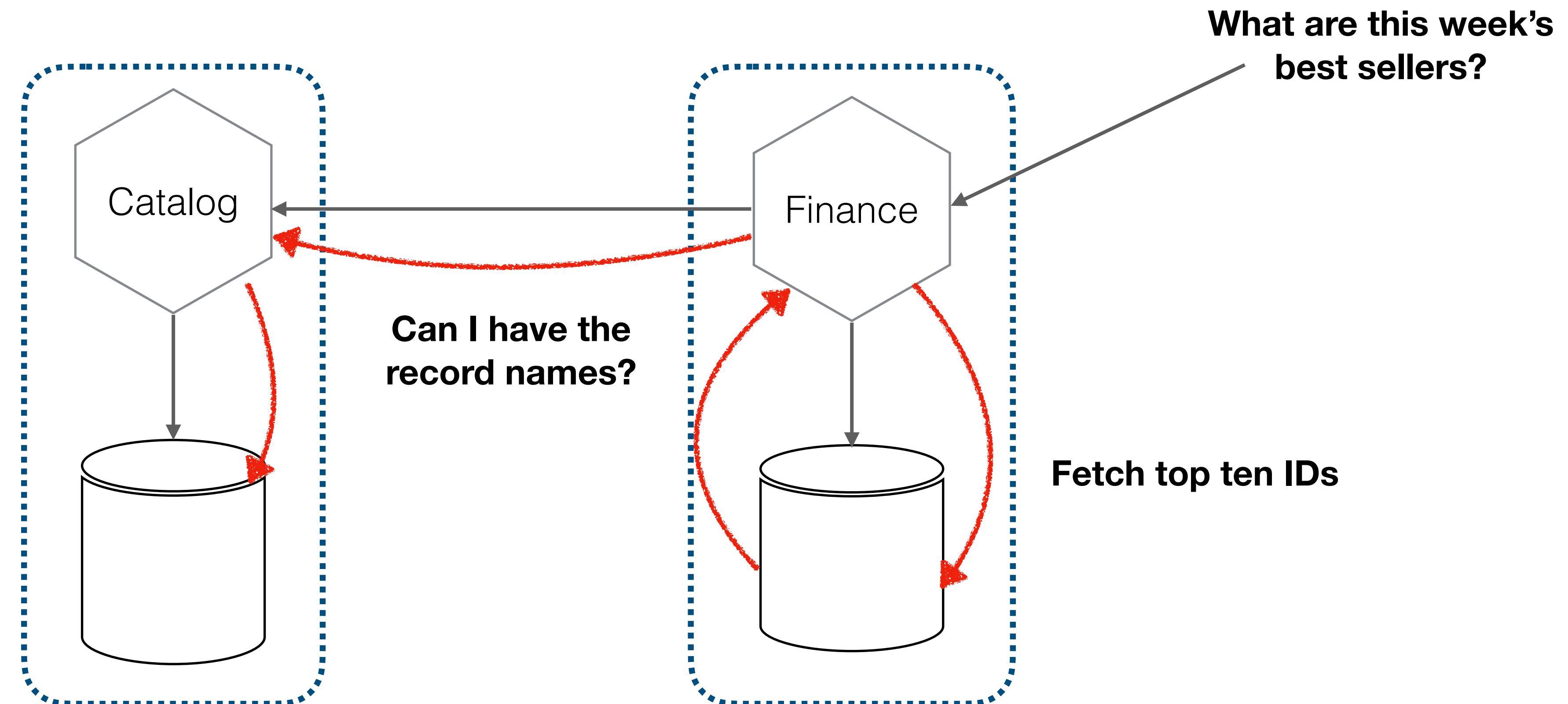
JOINS AT THE SERVICES TIER



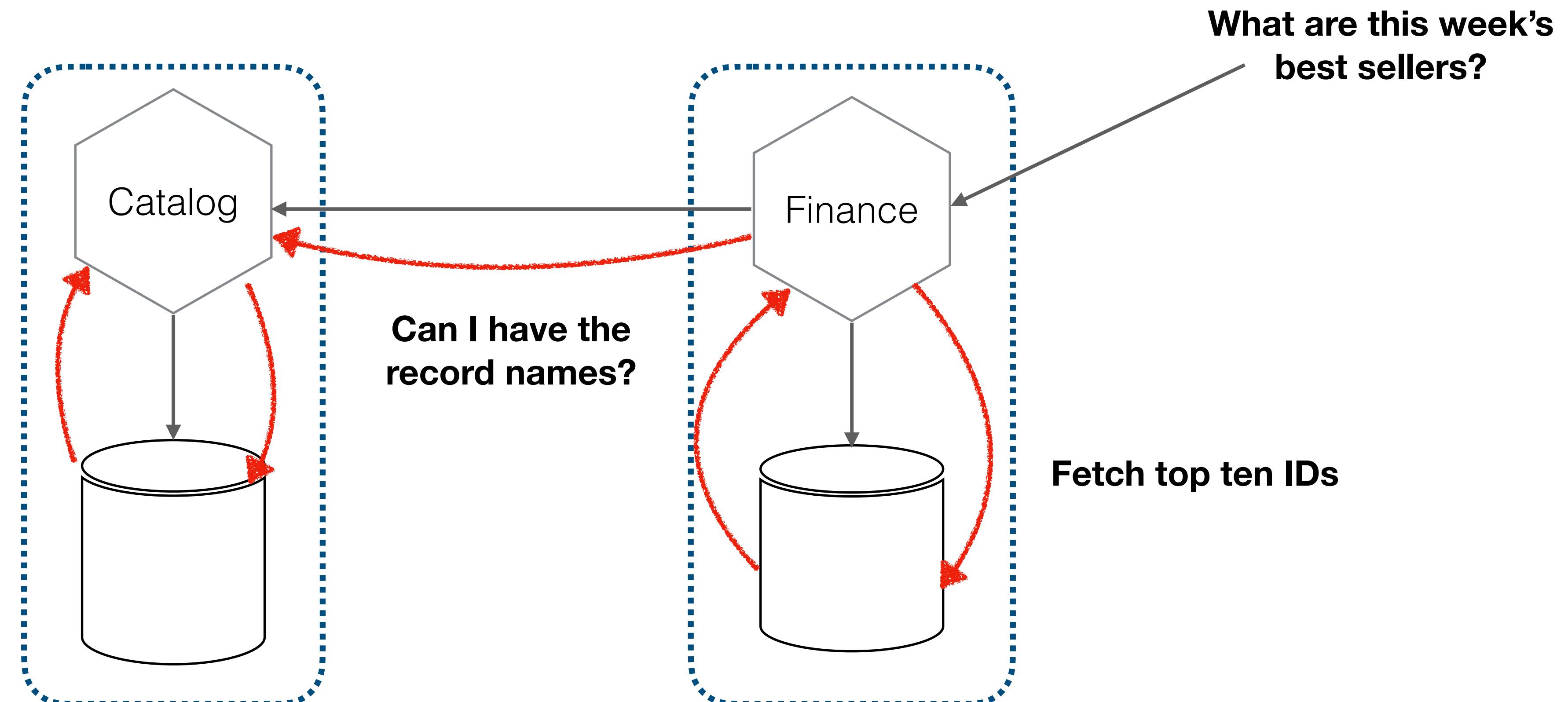
JOINS AT THE SERVICES TIER



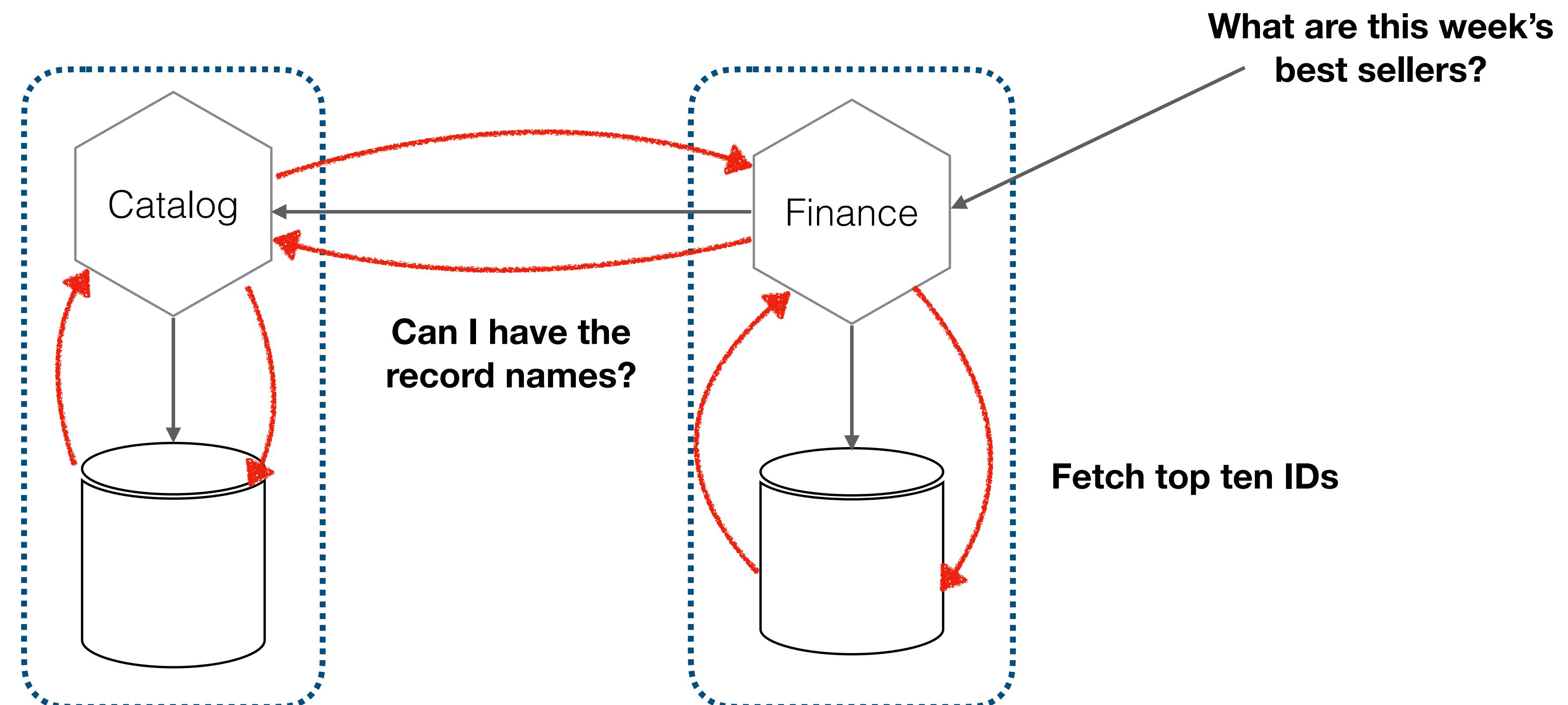
JOINS AT THE SERVICES TIER



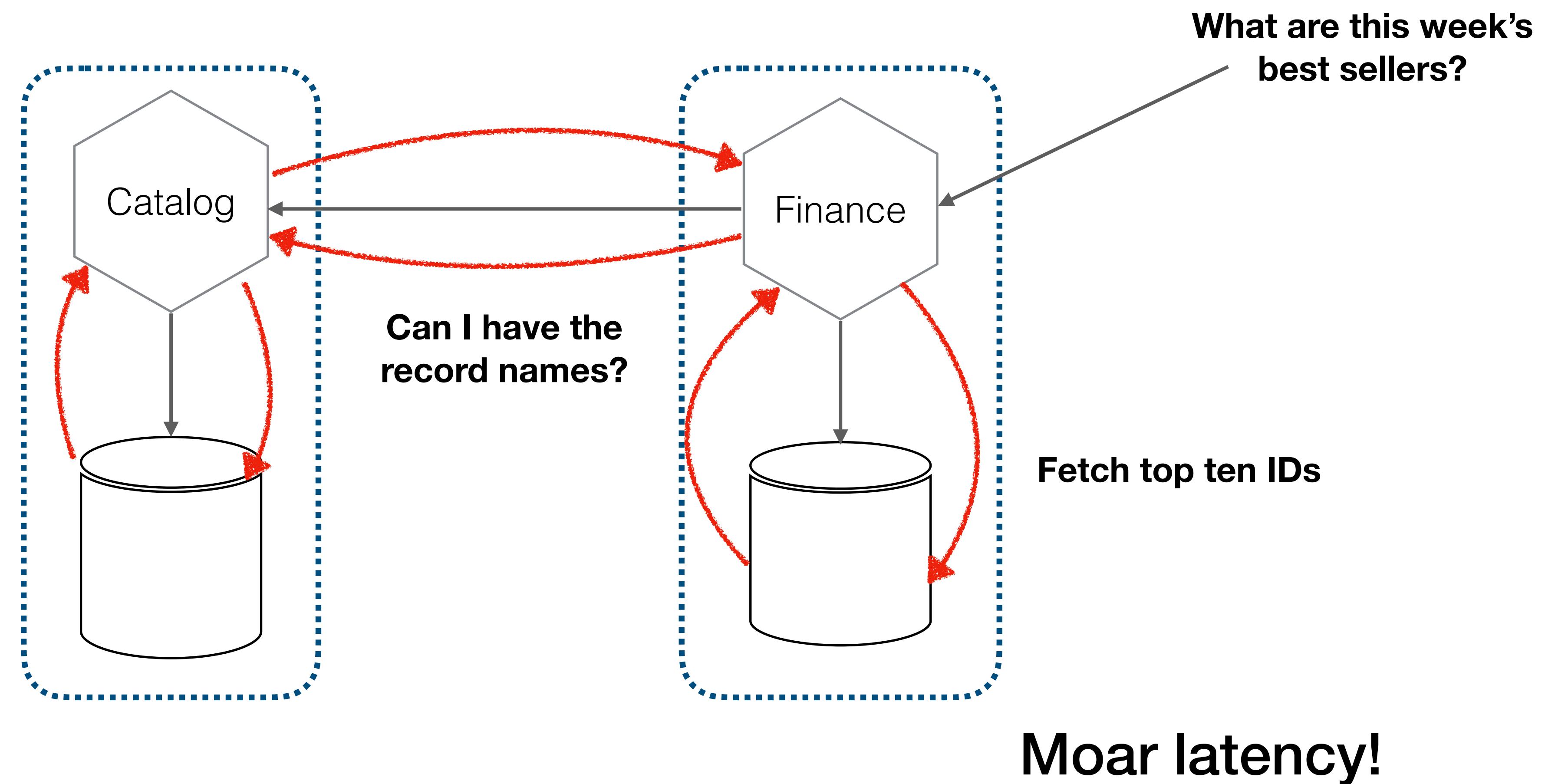
JOINS AT THE SERVICES TIER



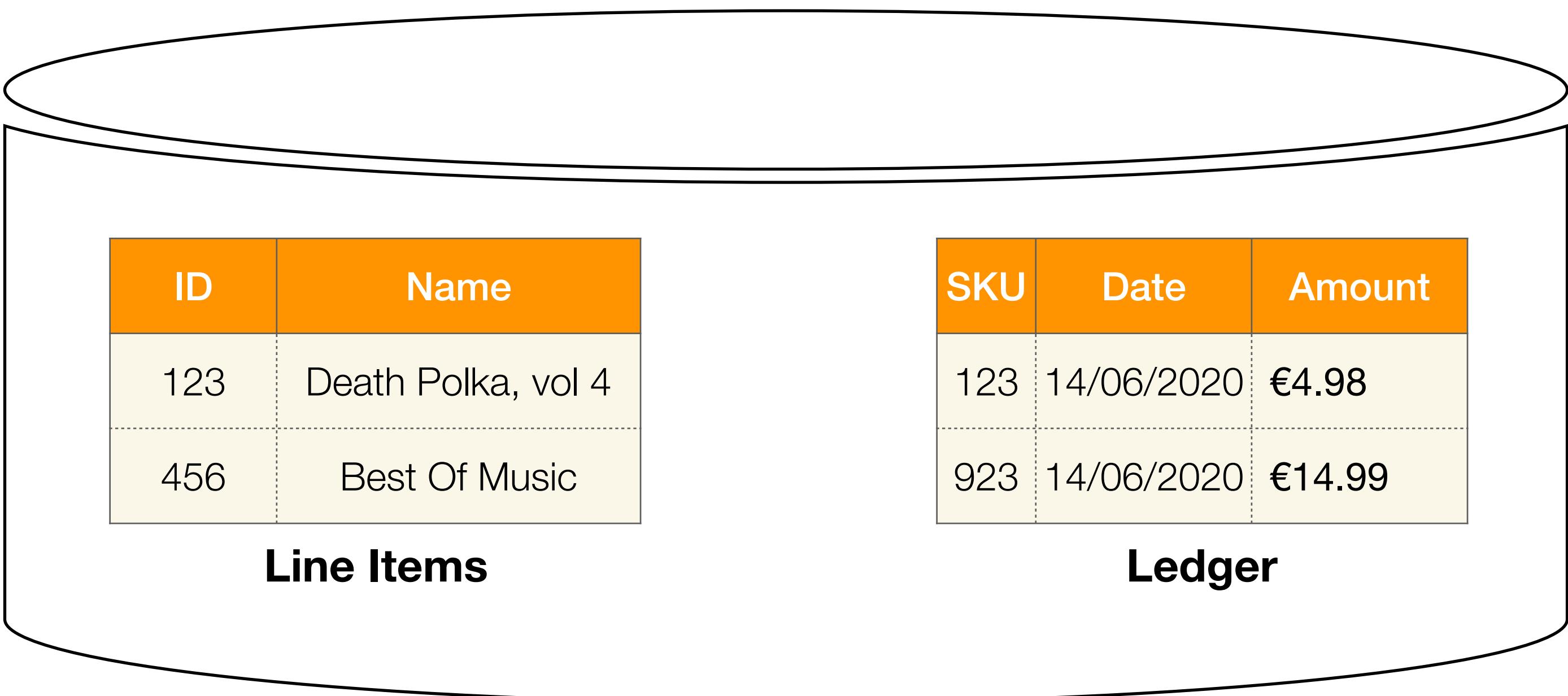
JOINS AT THE SERVICES TIER



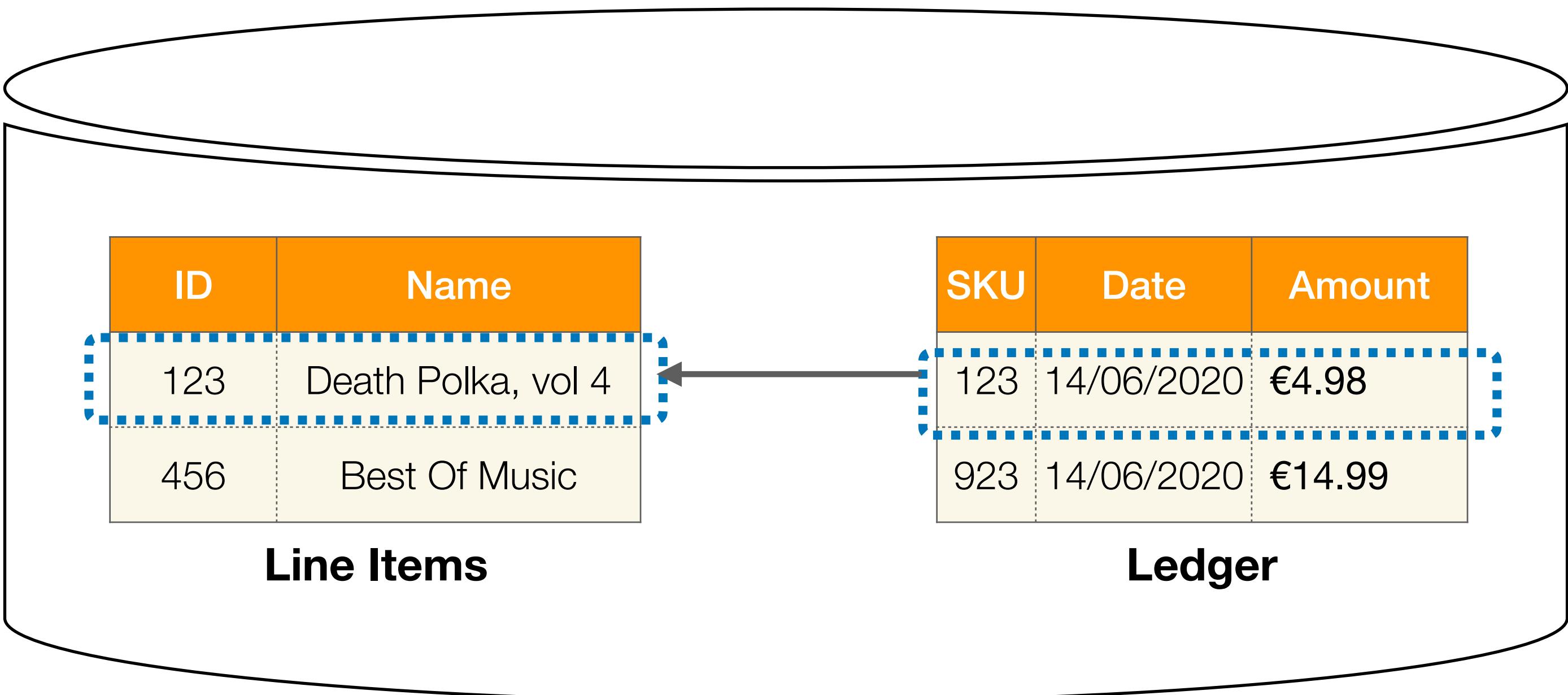
JOINS AT THE SERVICES TIER



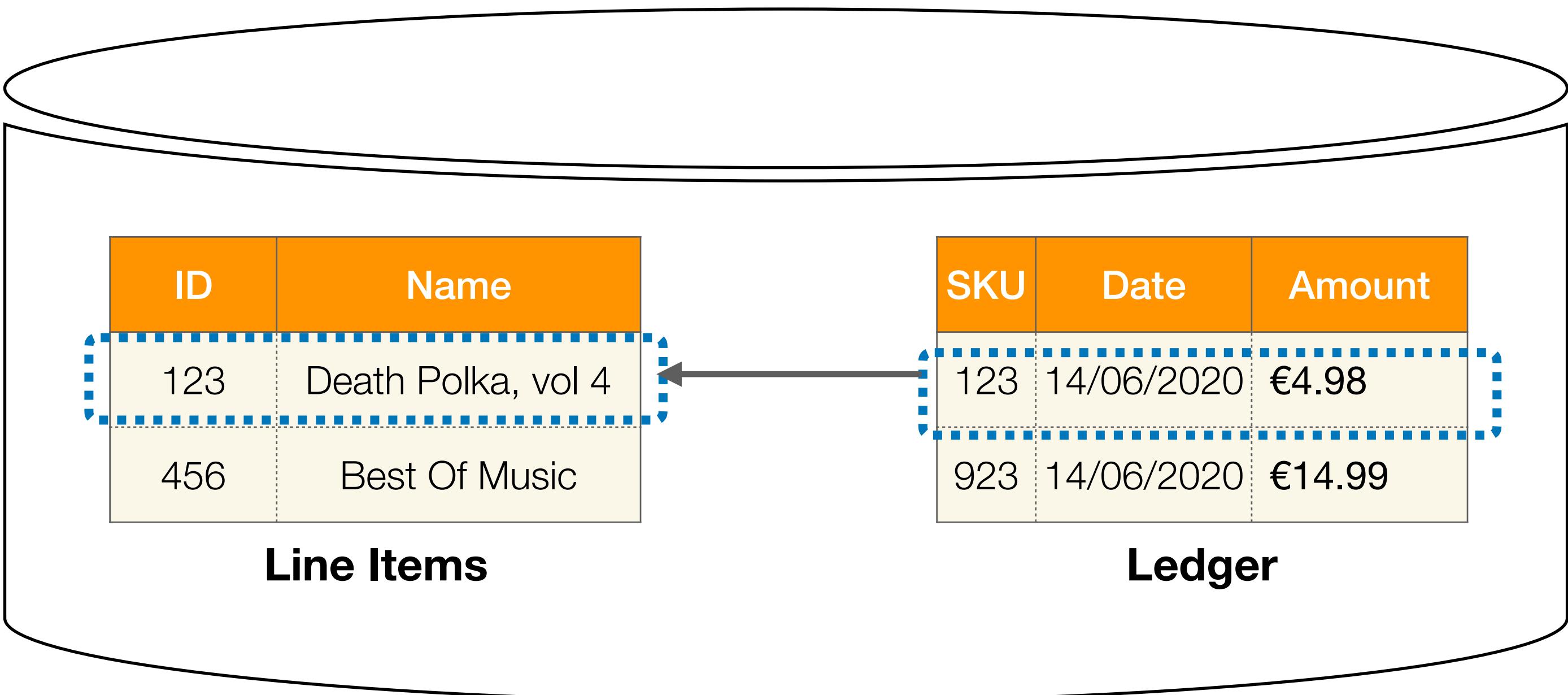
FOREIGN KEY RELATIONSHIPS



FOREIGN KEY RELATIONSHIPS

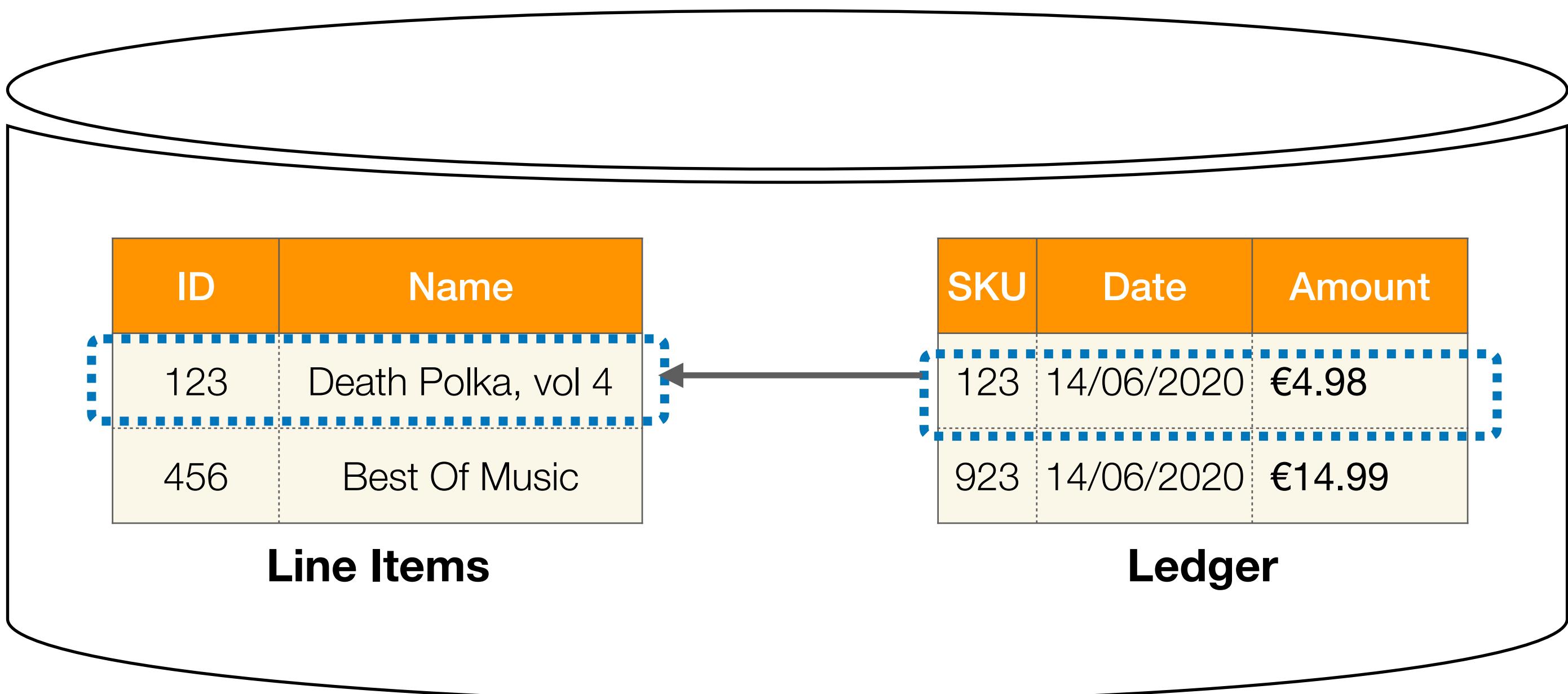


FOREIGN KEY RELATIONSHIPS



Enforce referential integrity

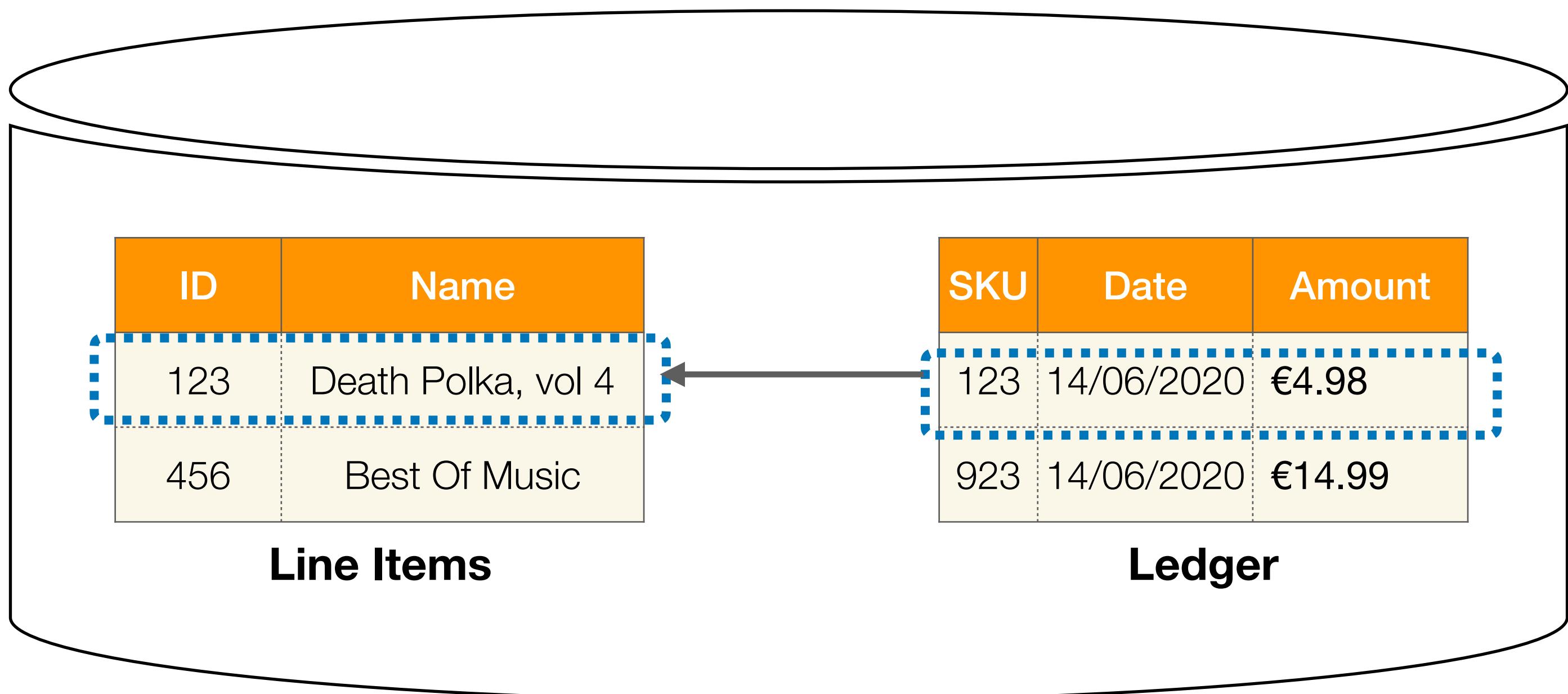
FOREIGN KEY RELATIONSHIPS



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

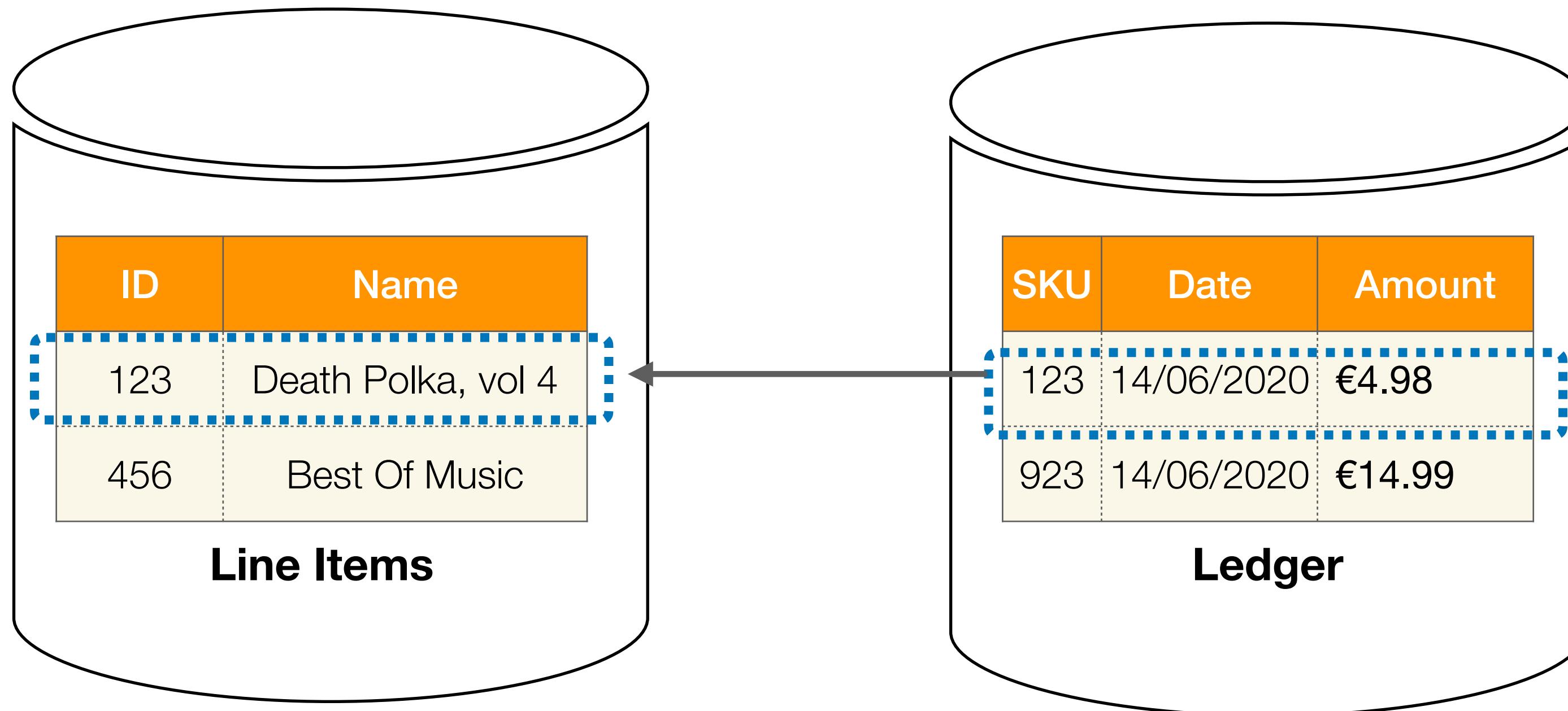
When facing the problems above, my team at SoundCloud started exploring alternatives that would allow for us to have simple, scalar values that were still rich enough to act as good identifiers across our hundreds of microservices. Reading through decades of industry work on the matter, we found something 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:

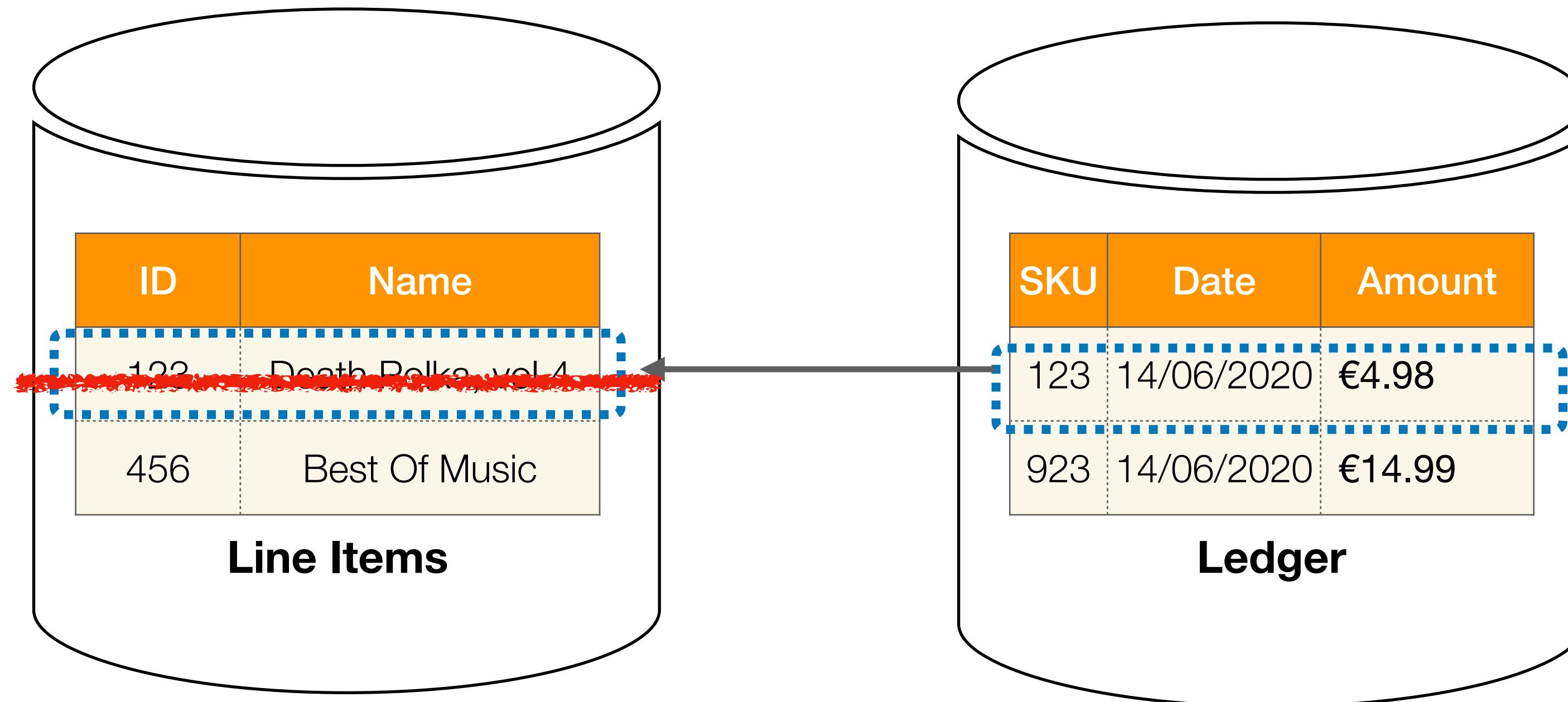
```
urn:tracks:123  
urn:users:123  
urn:comments:123  
urn:artwork:123  
urn:playlists:123
```

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

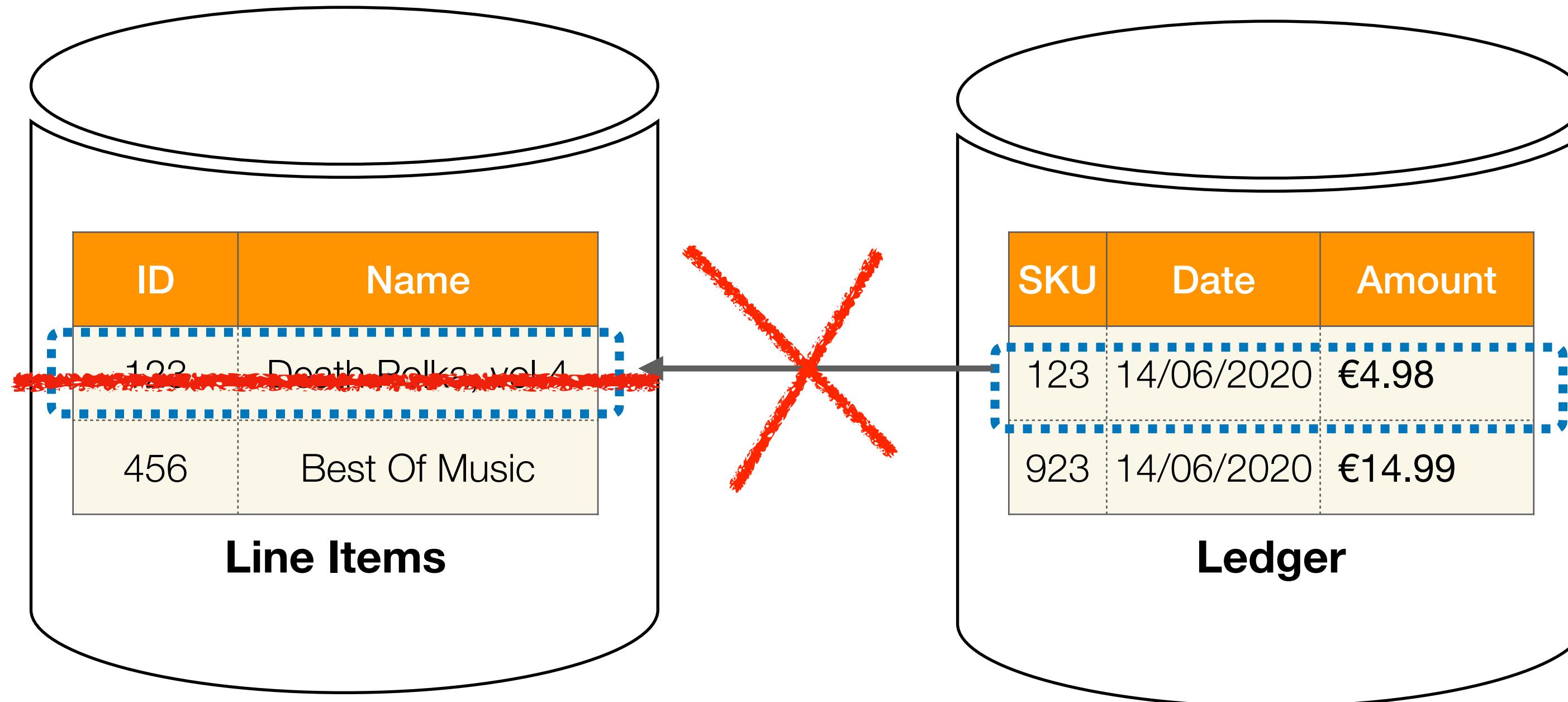
REFERENTIAL INTEGRITY ACROSS DATABASES?



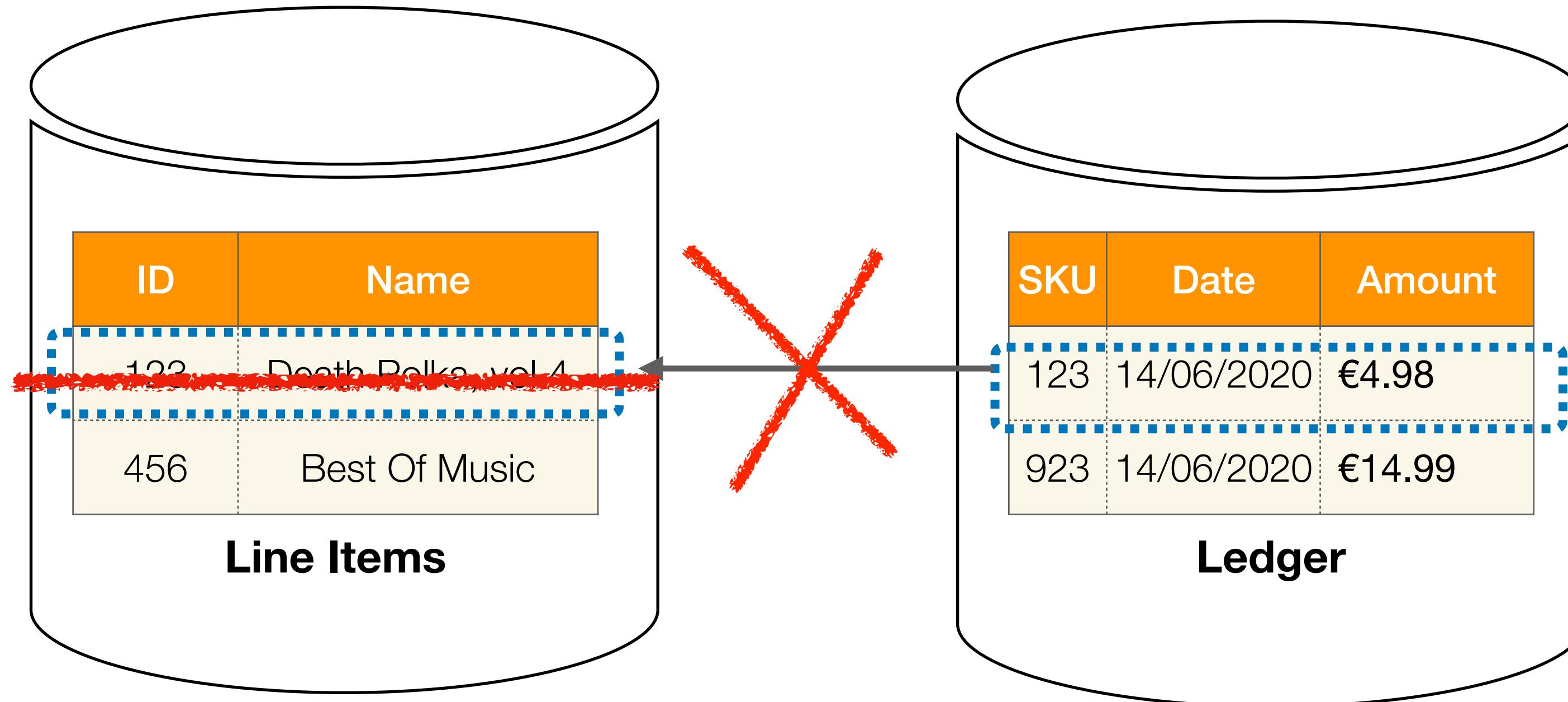
REFERENTIAL INTEGRITY ACROSS DATABASES?



REFERENTIAL INTEGRITY ACROSS DATABASES?

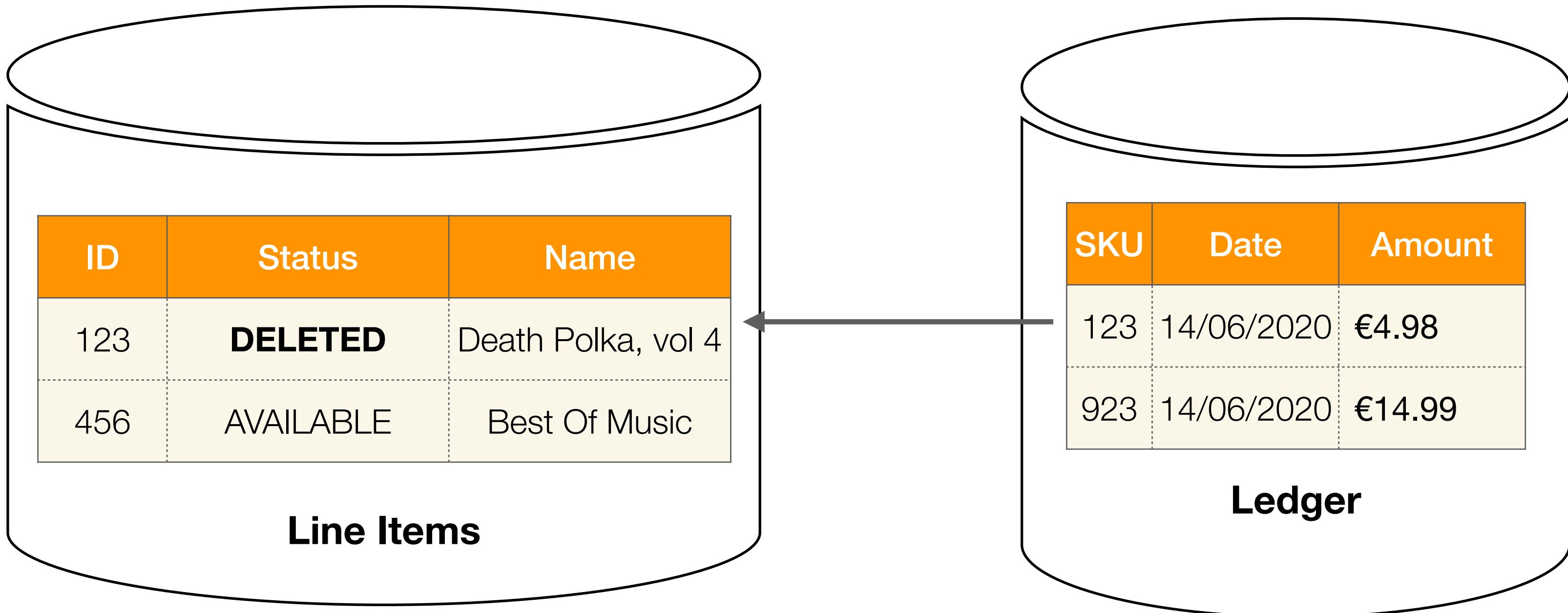


REFERENTIAL INTEGRITY ACROSS DATABASES?

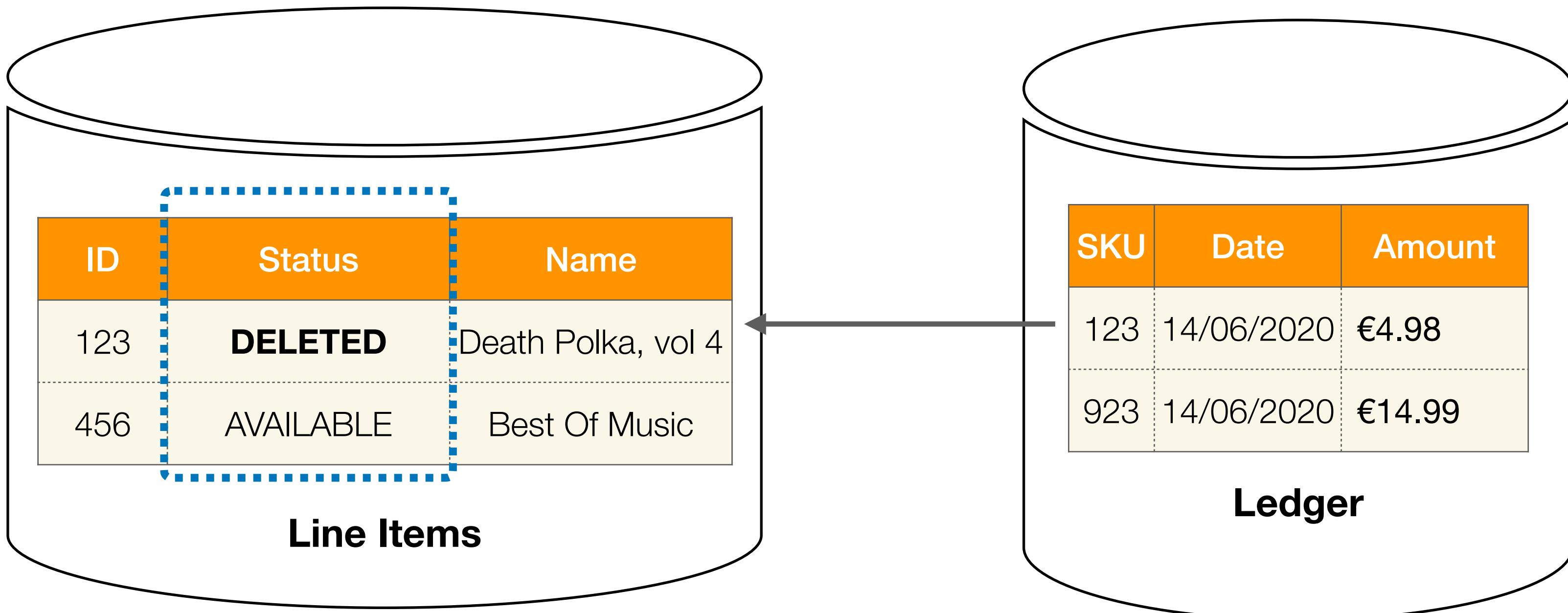


What are our options?

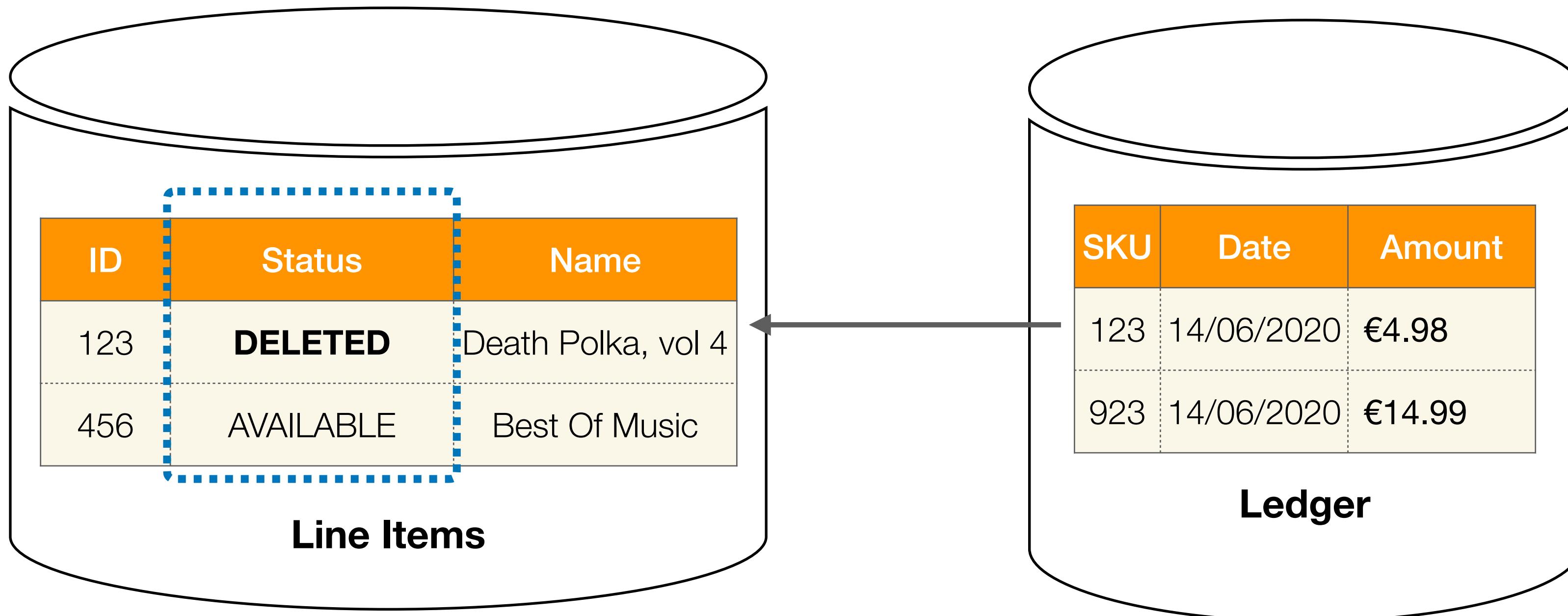
SOFT DELETE



SOFT DELETE



SOFT DELETE



Data not removed, allowing internal references to continue to work

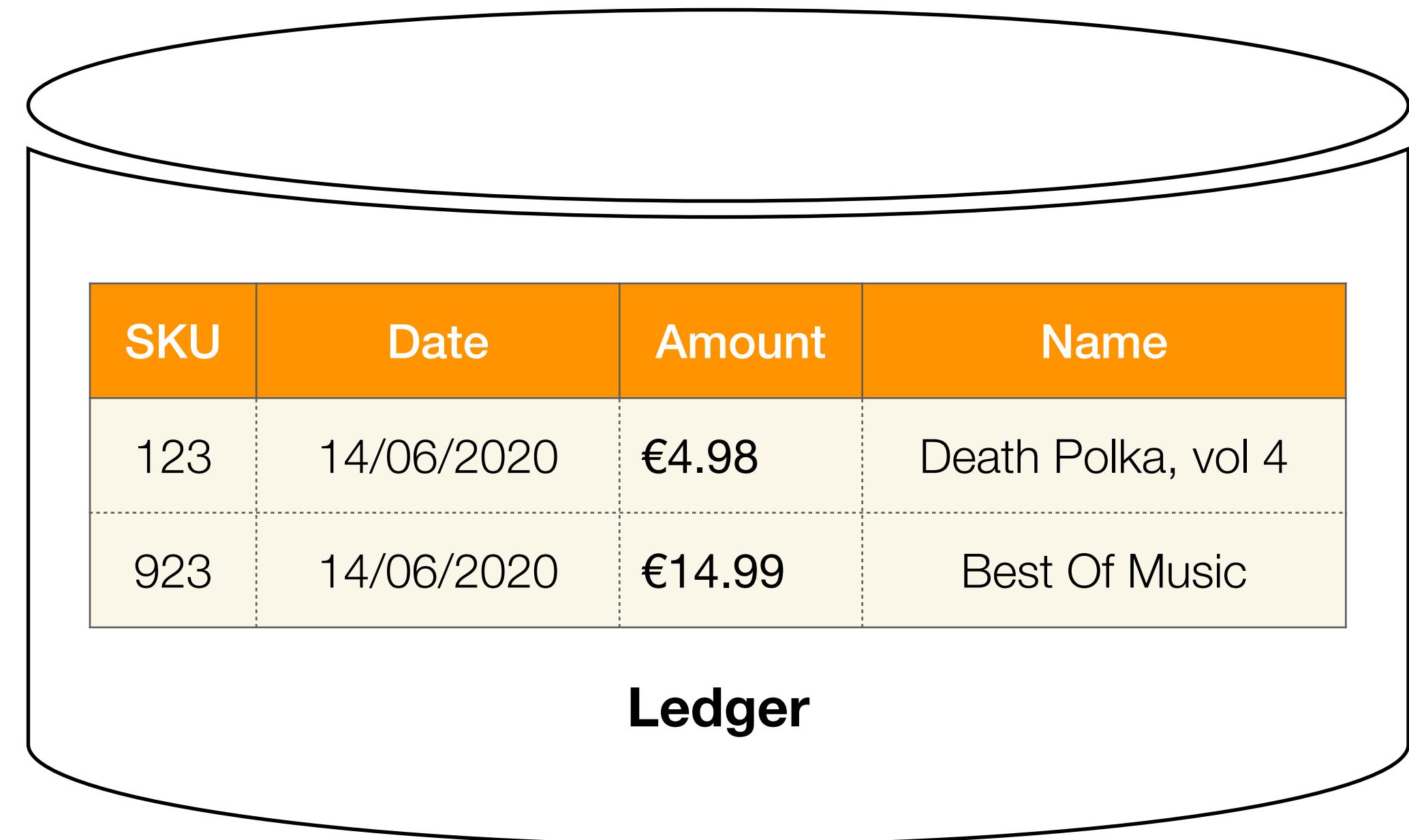
COPY DATA



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

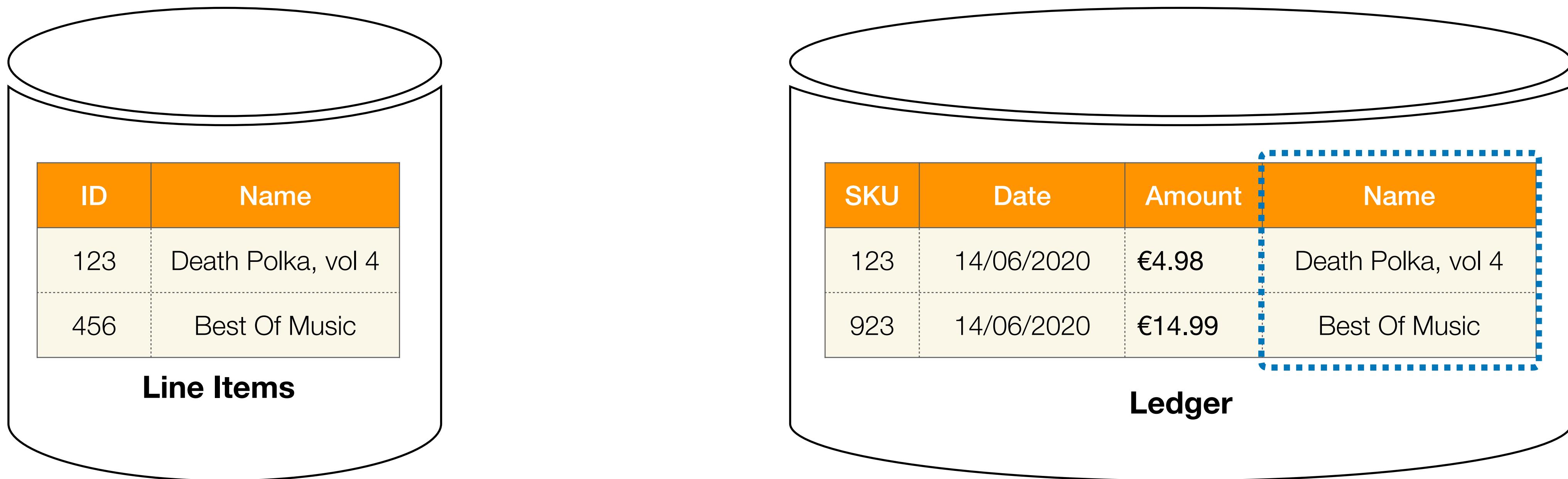
Line Items

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

Ledger

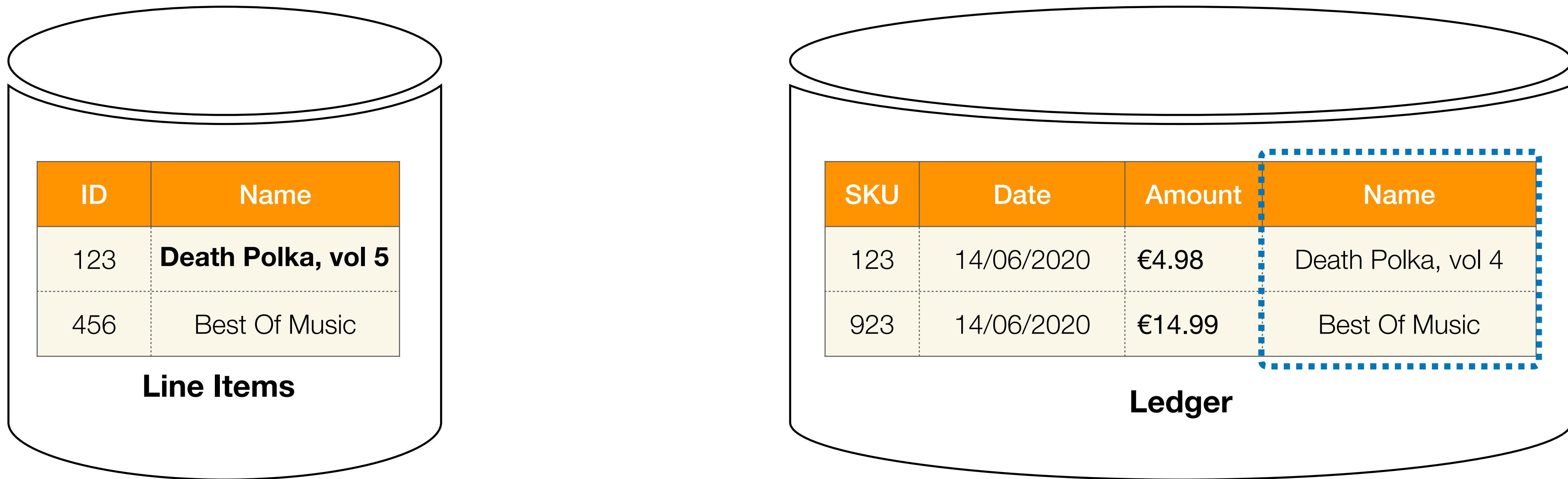
SKU	Date	Amount	Name
123	14/06/2020	€4.98	Death Polka, vol 4
923	14/06/2020	€14.99	Best Of Music

COPY DATA



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

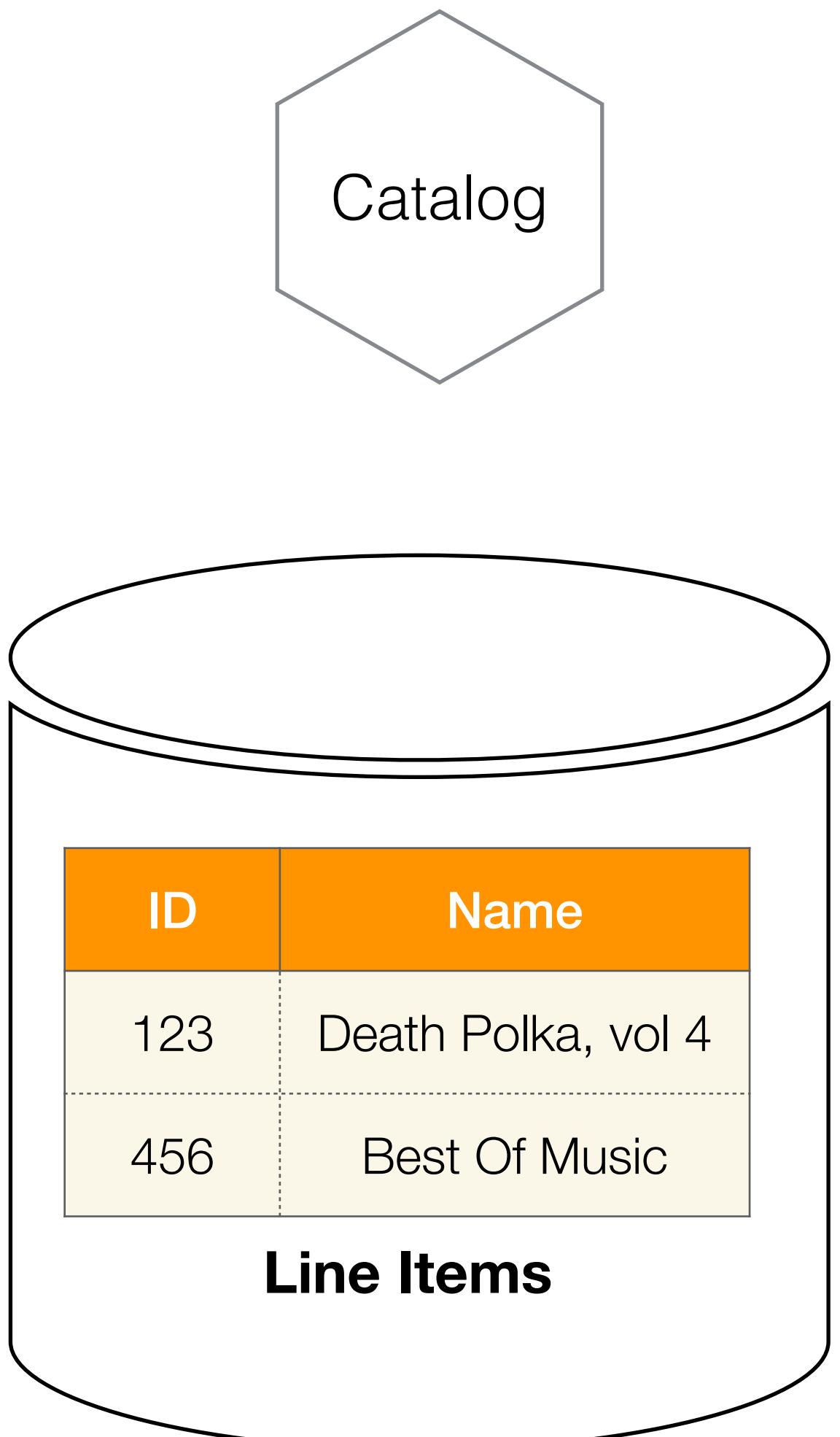
COPY DATA



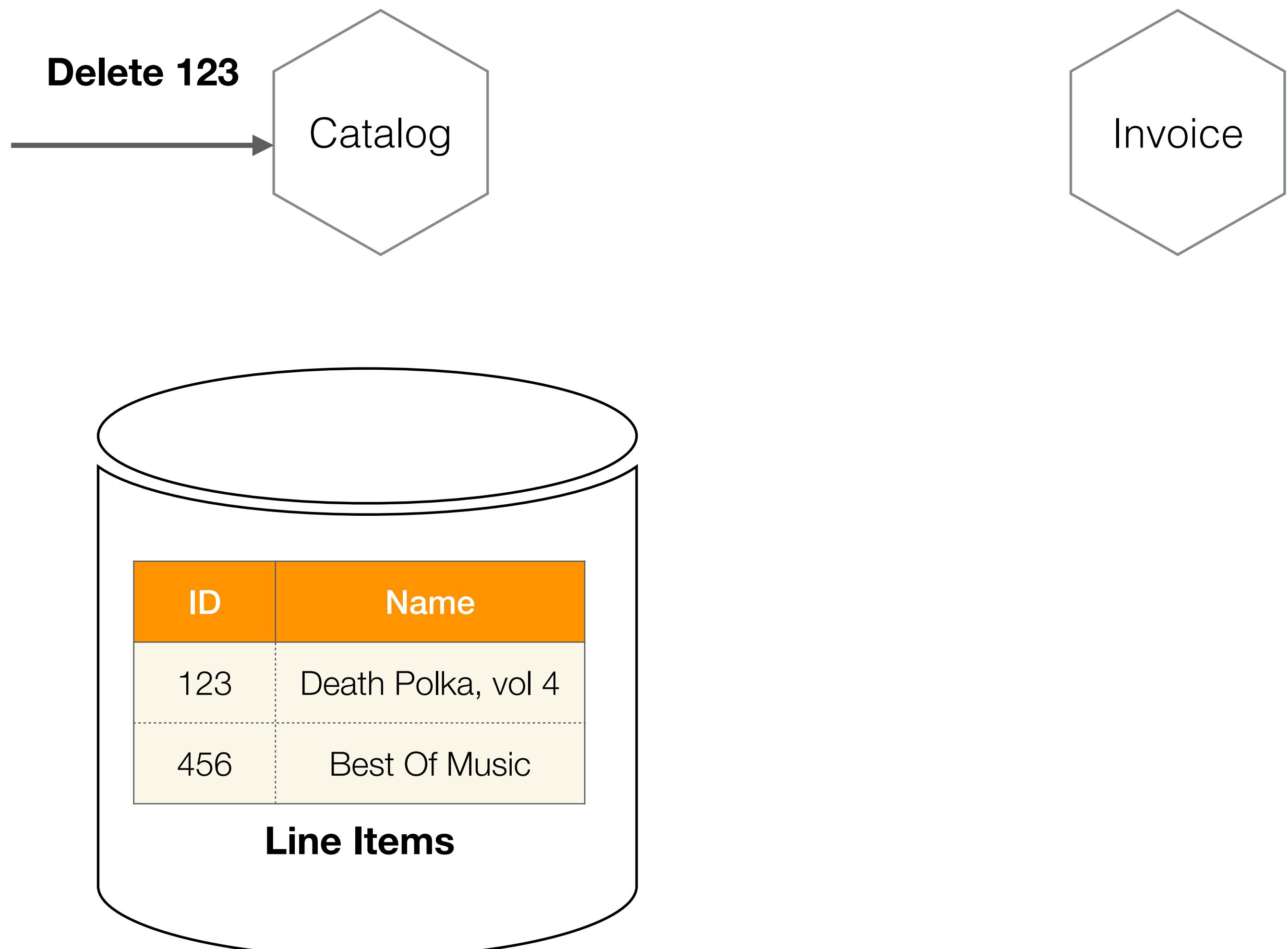
Avoids the need for the join if the name is the only thing needed

But what about if the source data changes?

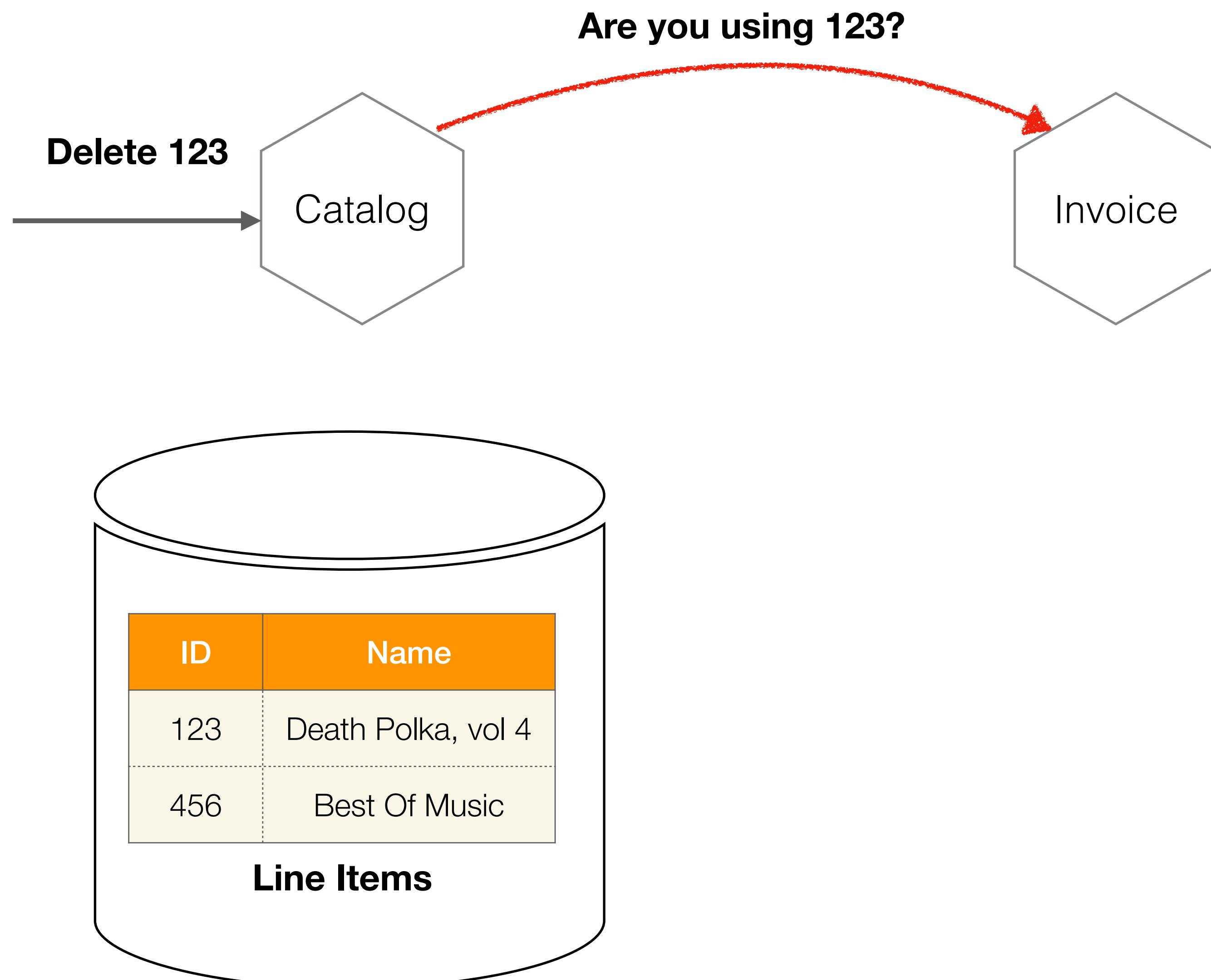
CHECK FOR USE BEFORE DELETE?



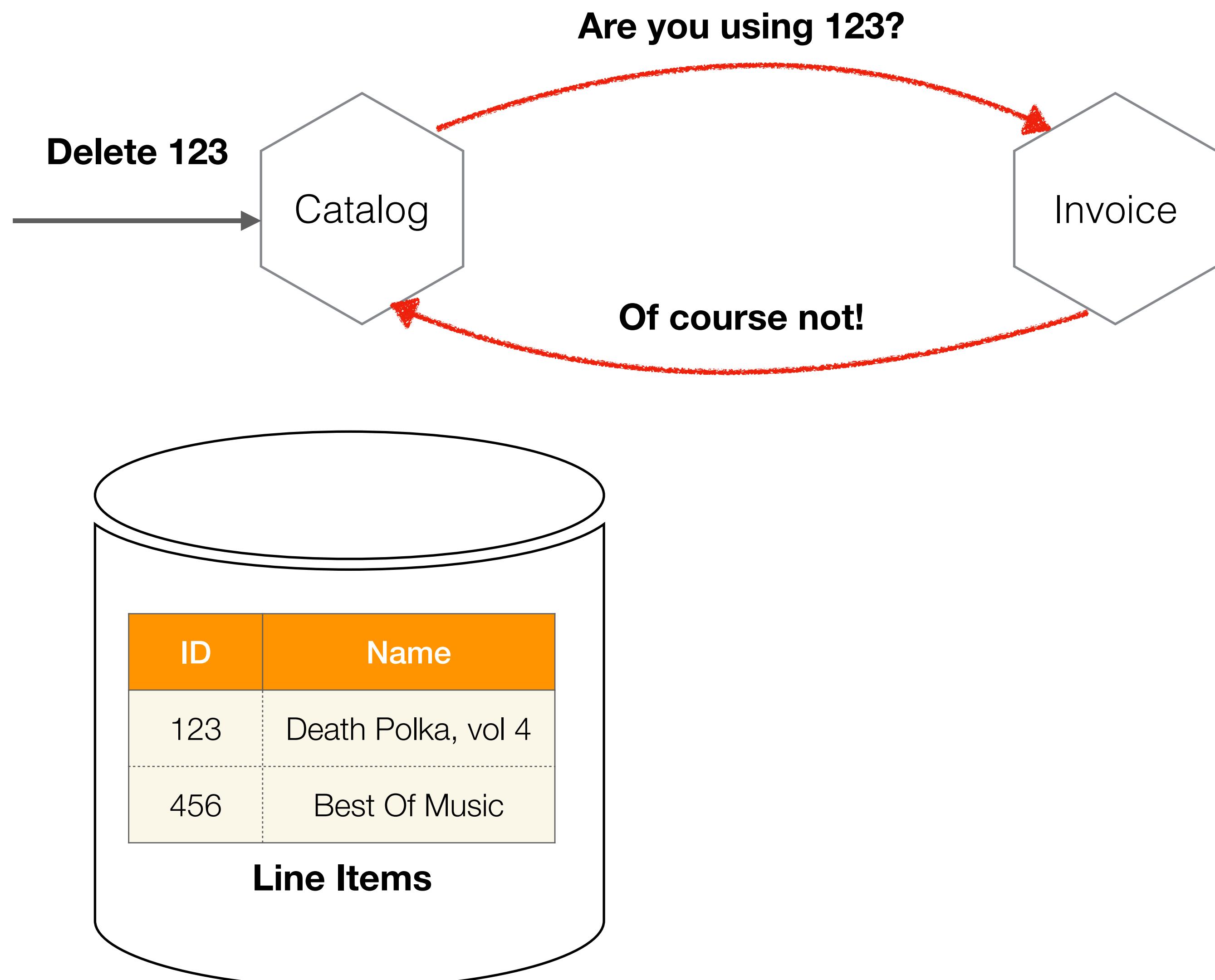
CHECK FOR USE BEFORE DELETE?



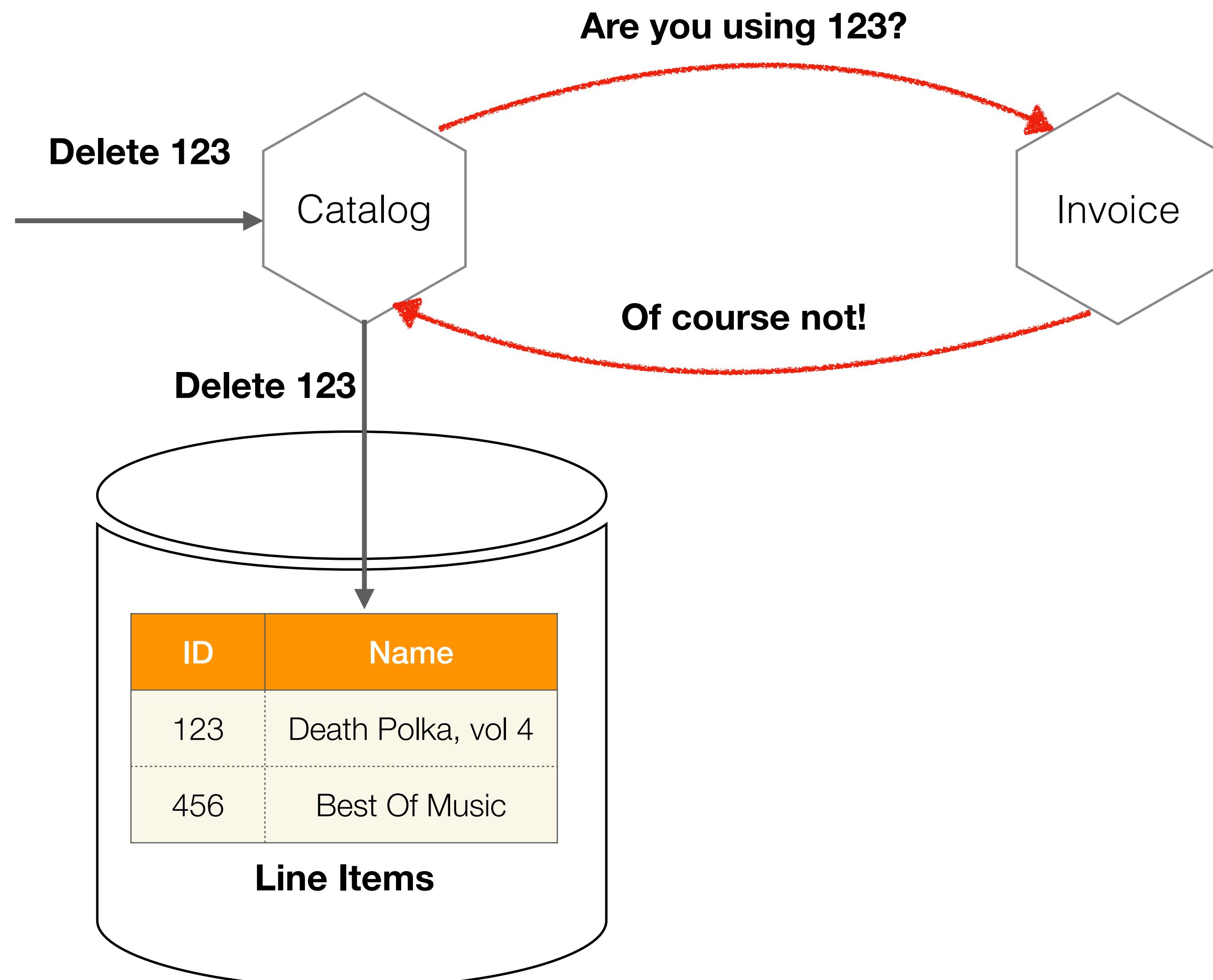
CHECK FOR USE BEFORE DELETE?



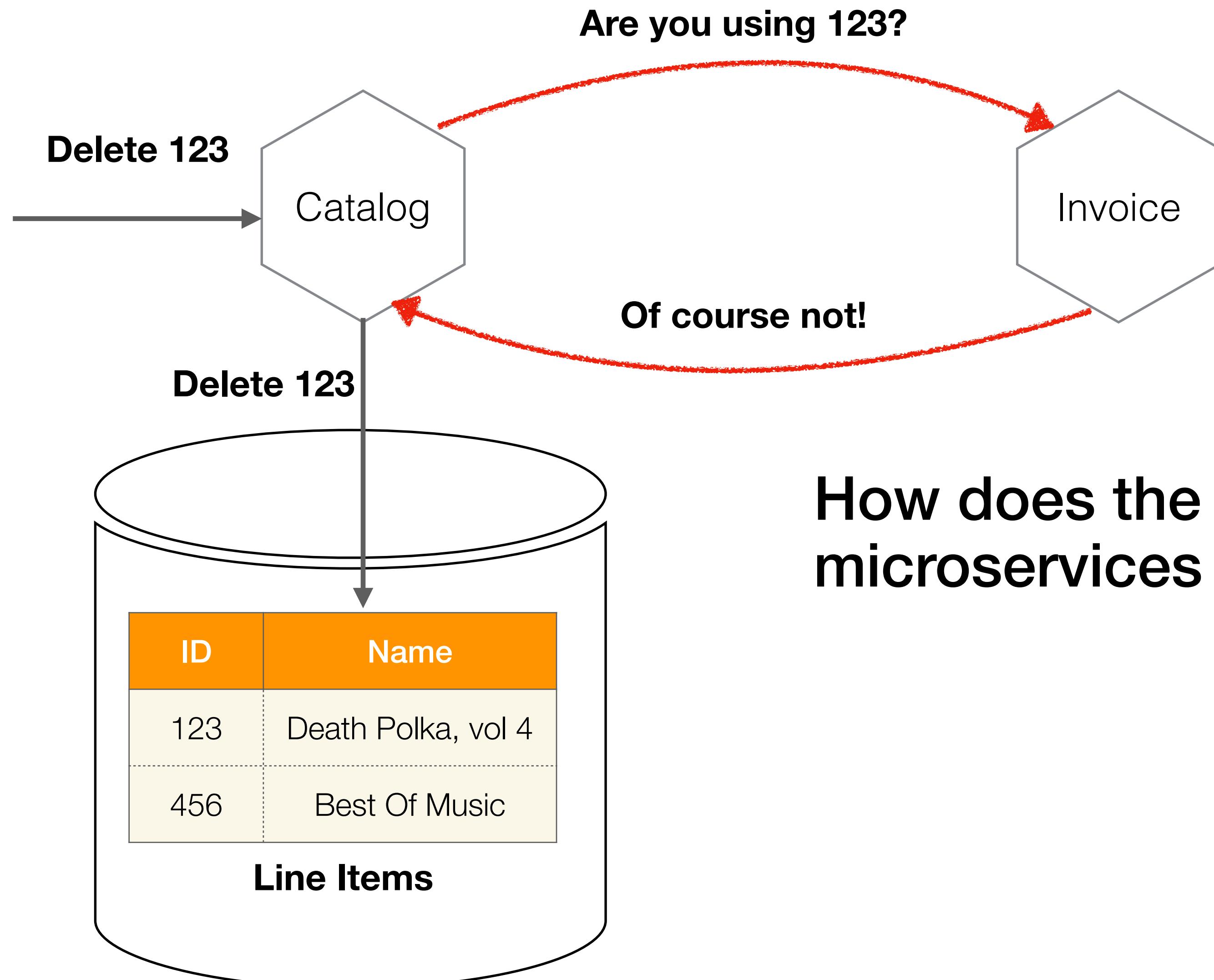
CHECK FOR USE BEFORE DELETE?



CHECK FOR USE BEFORE DELETE?

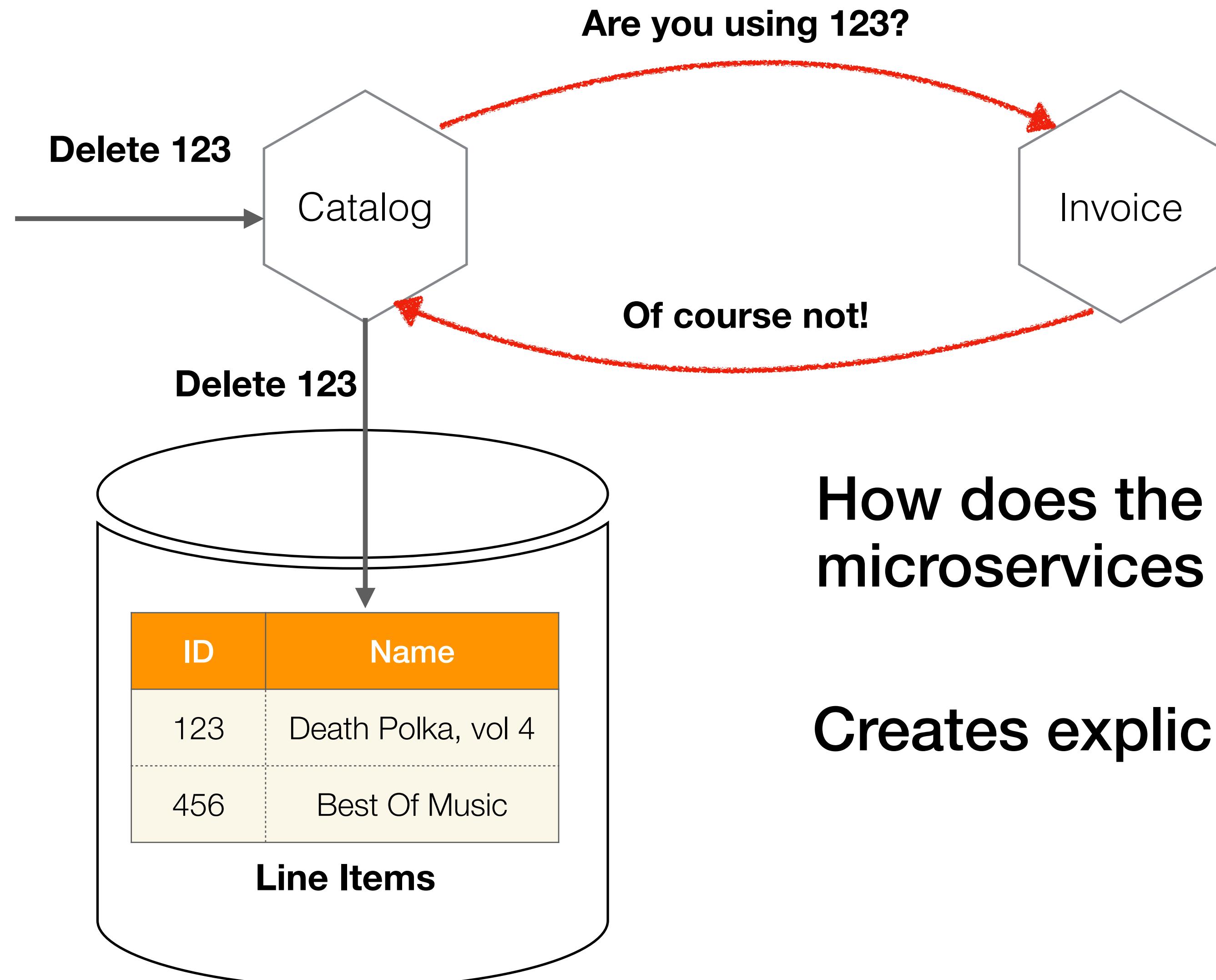


CHECK FOR USE BEFORE DELETE?



How does the catalog know which microservices to ask?

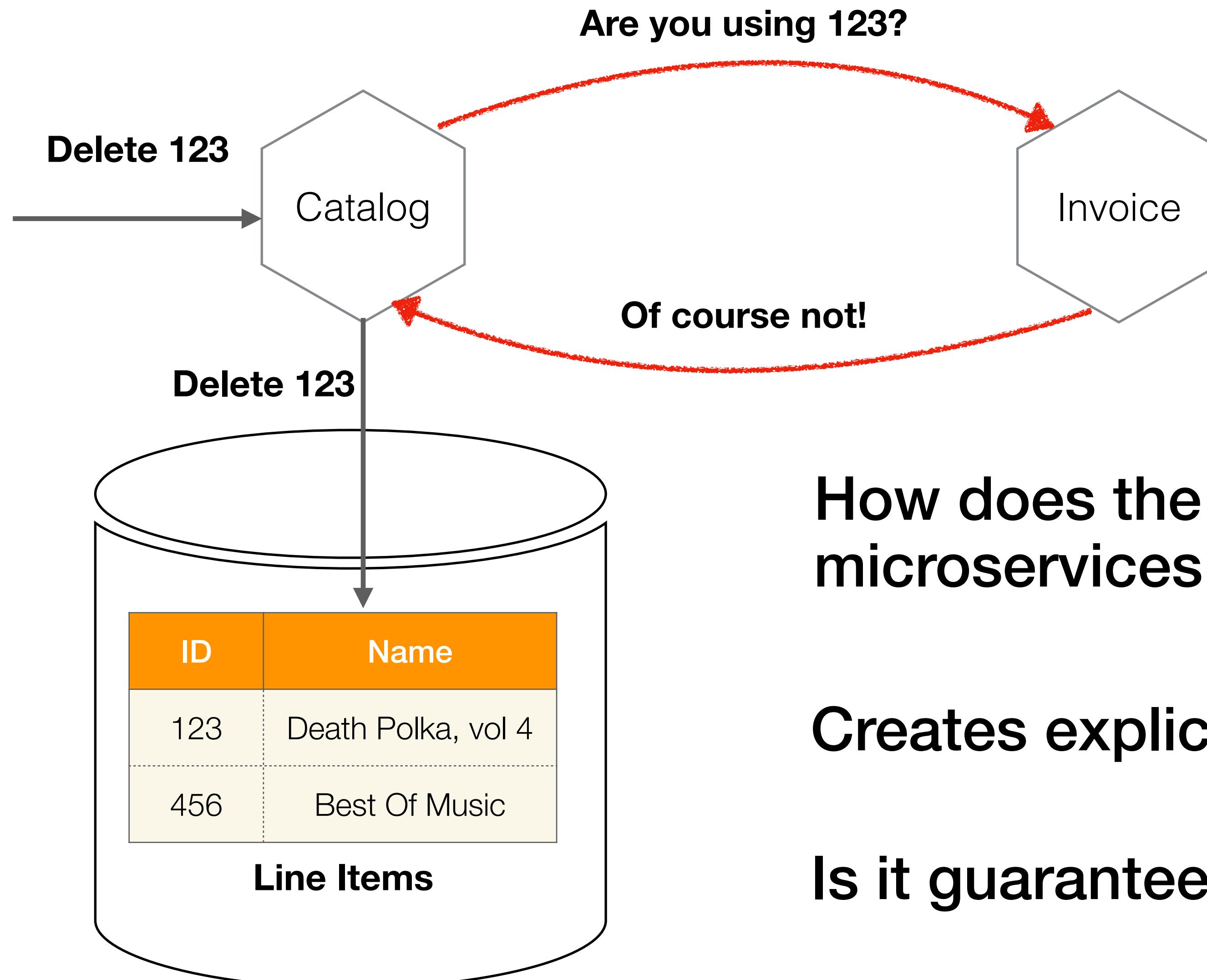
CHECK FOR USE BEFORE DELETE?



How does the catalog know which microservices to ask?

Creates explicit circular dependencies

CHECK FOR USE BEFORE DELETE?



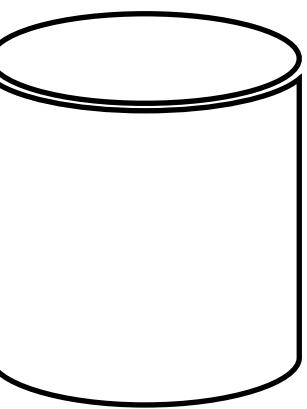
How does the catalog know which microservices to ask?

Creates explicit circular dependencies

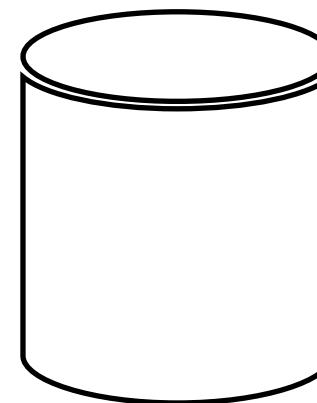
Is it guaranteed to work?

LOCKING REQUIRED?

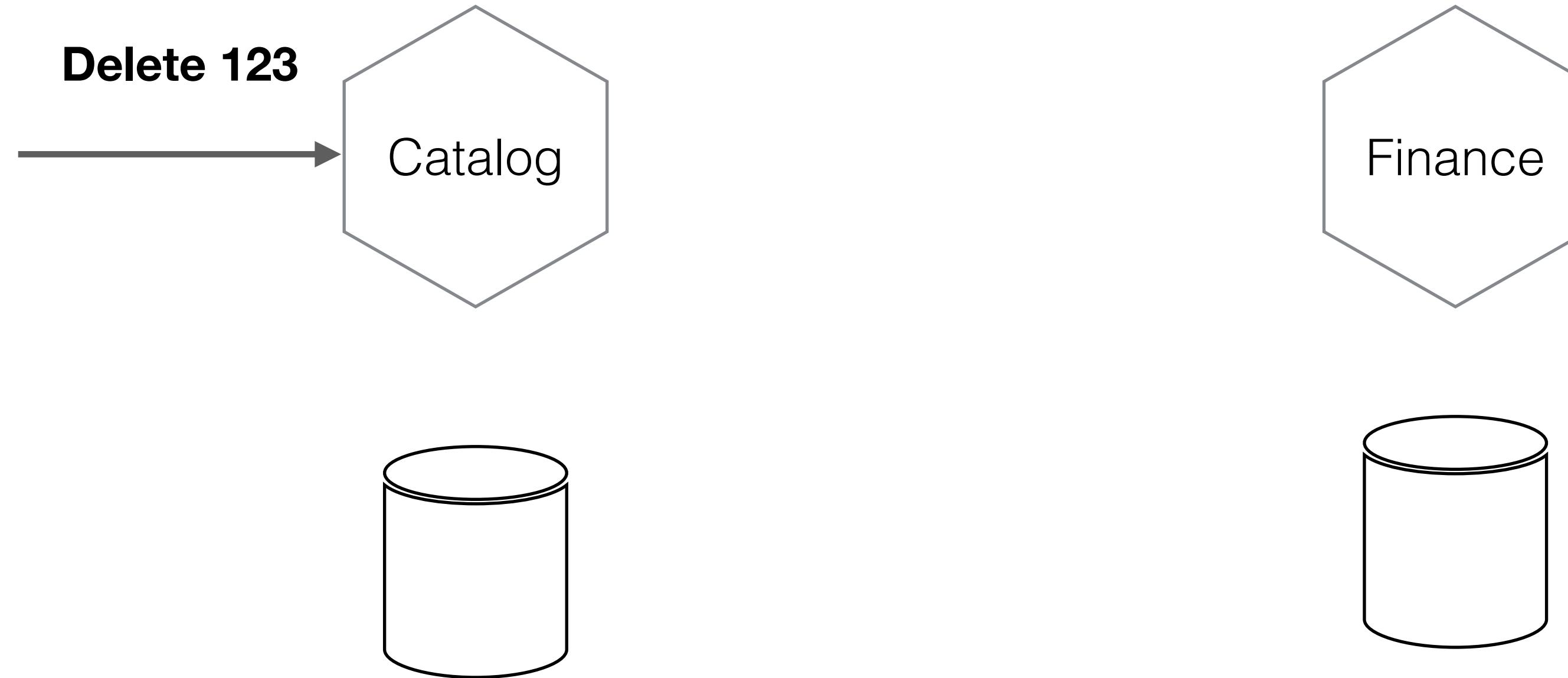
Catalog



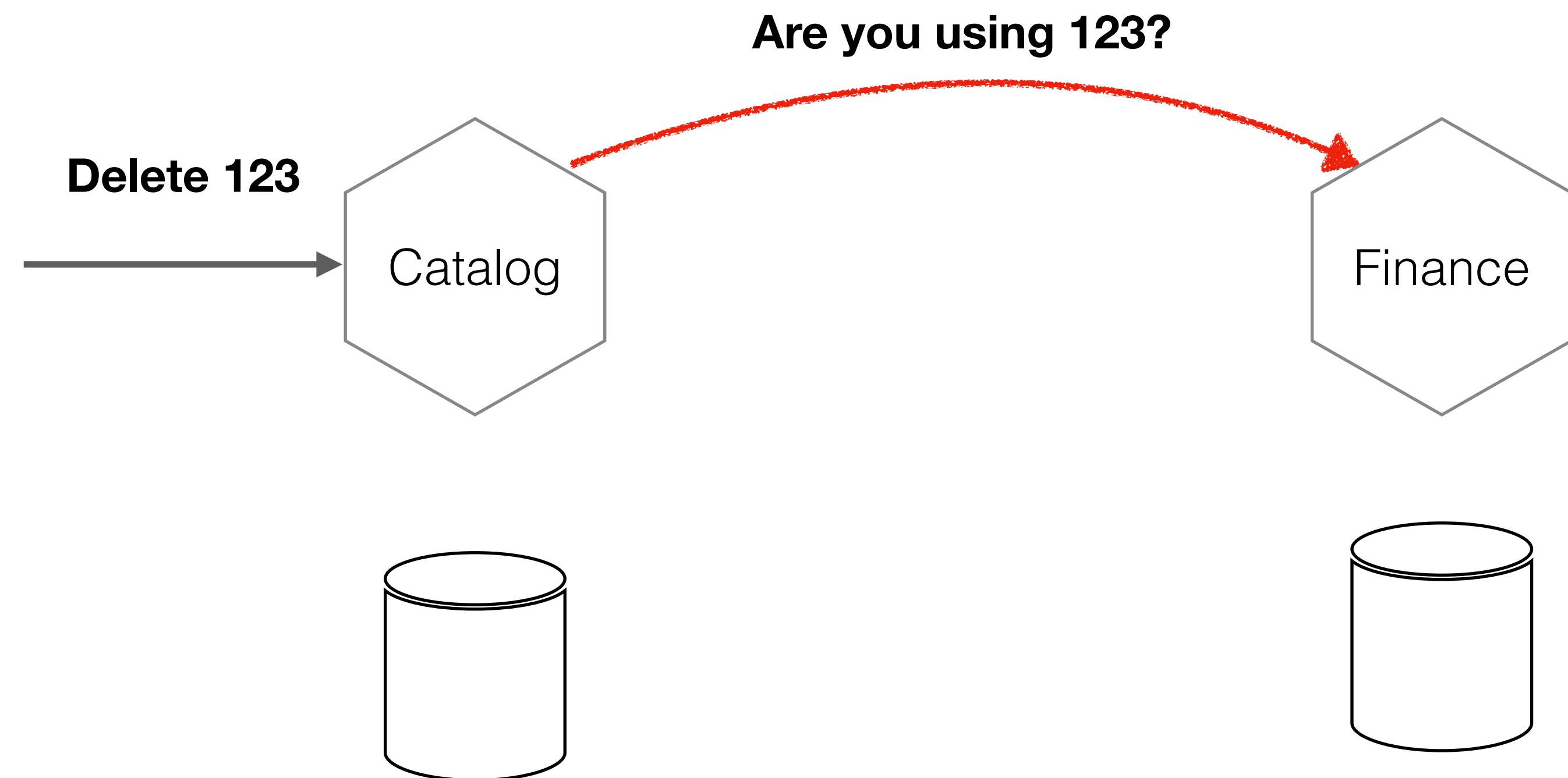
Finance



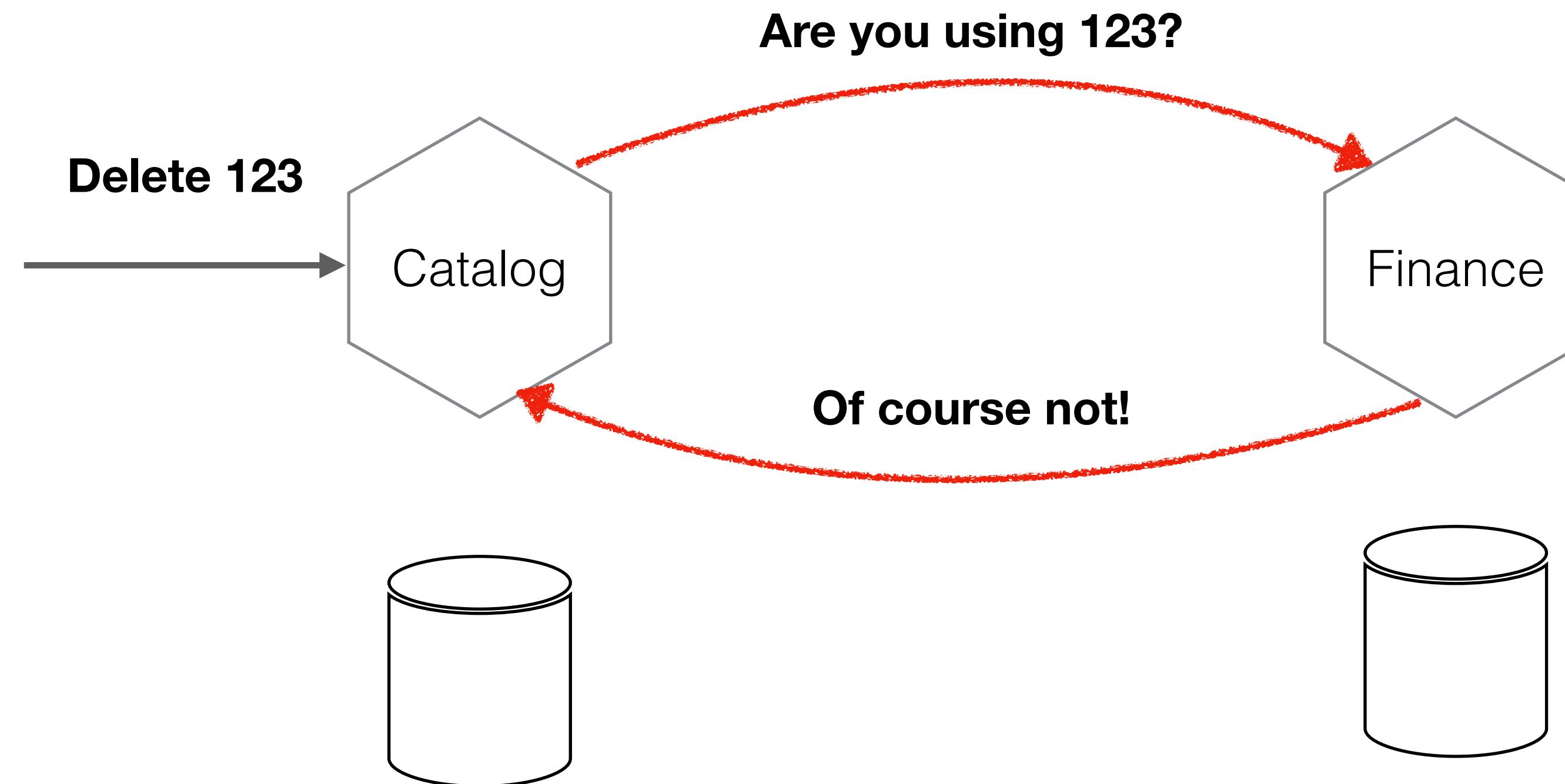
LOCKING REQUIRED?



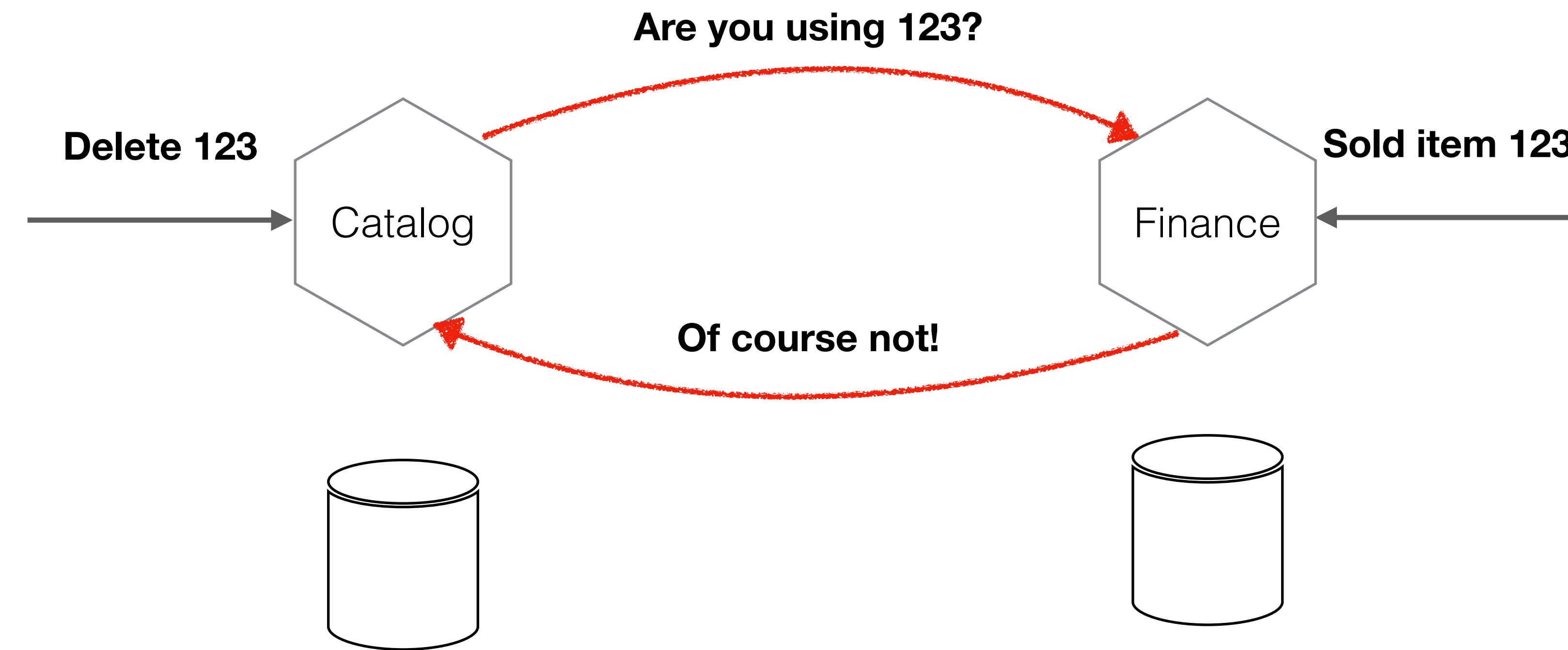
LOCKING REQUIRED?



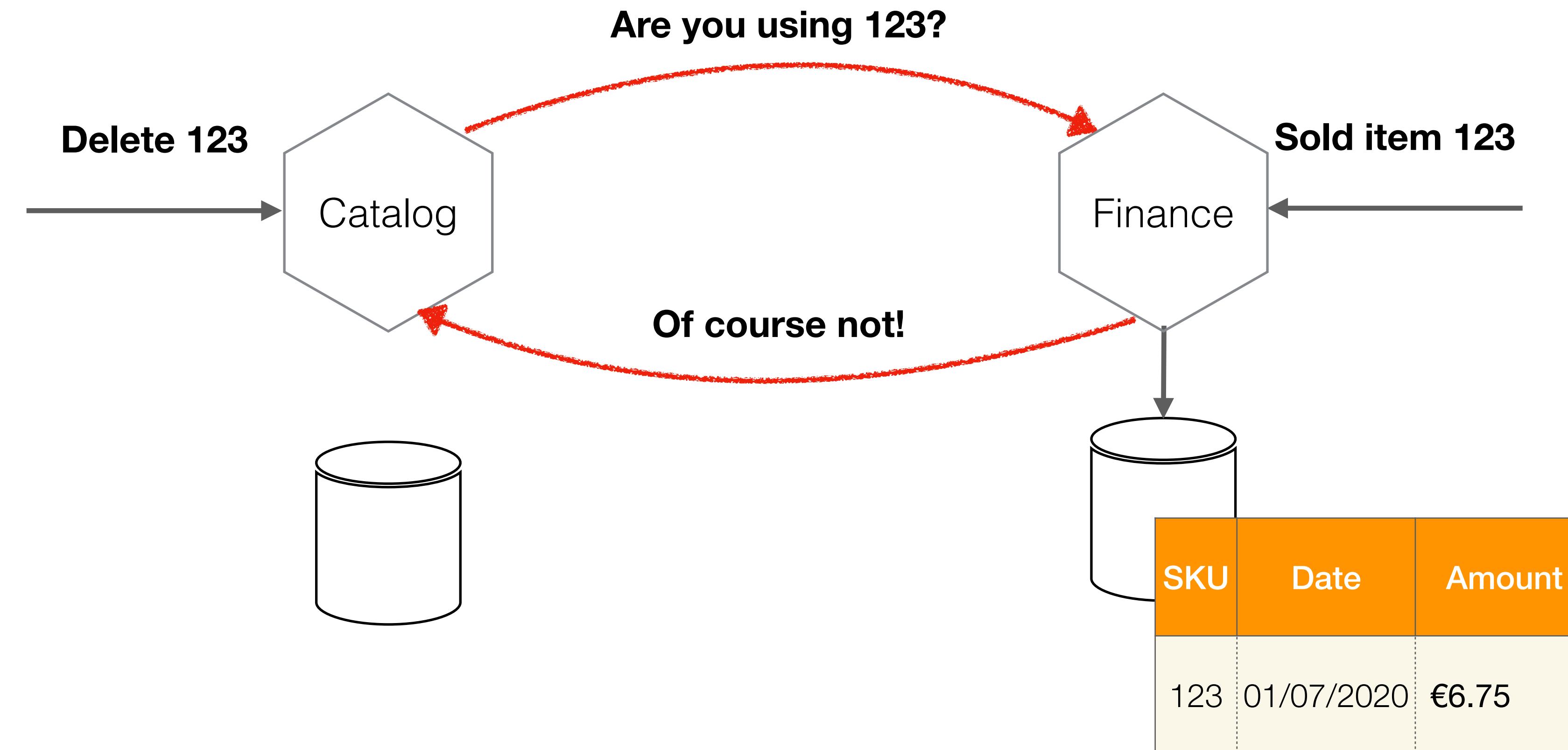
LOCKING REQUIRED?



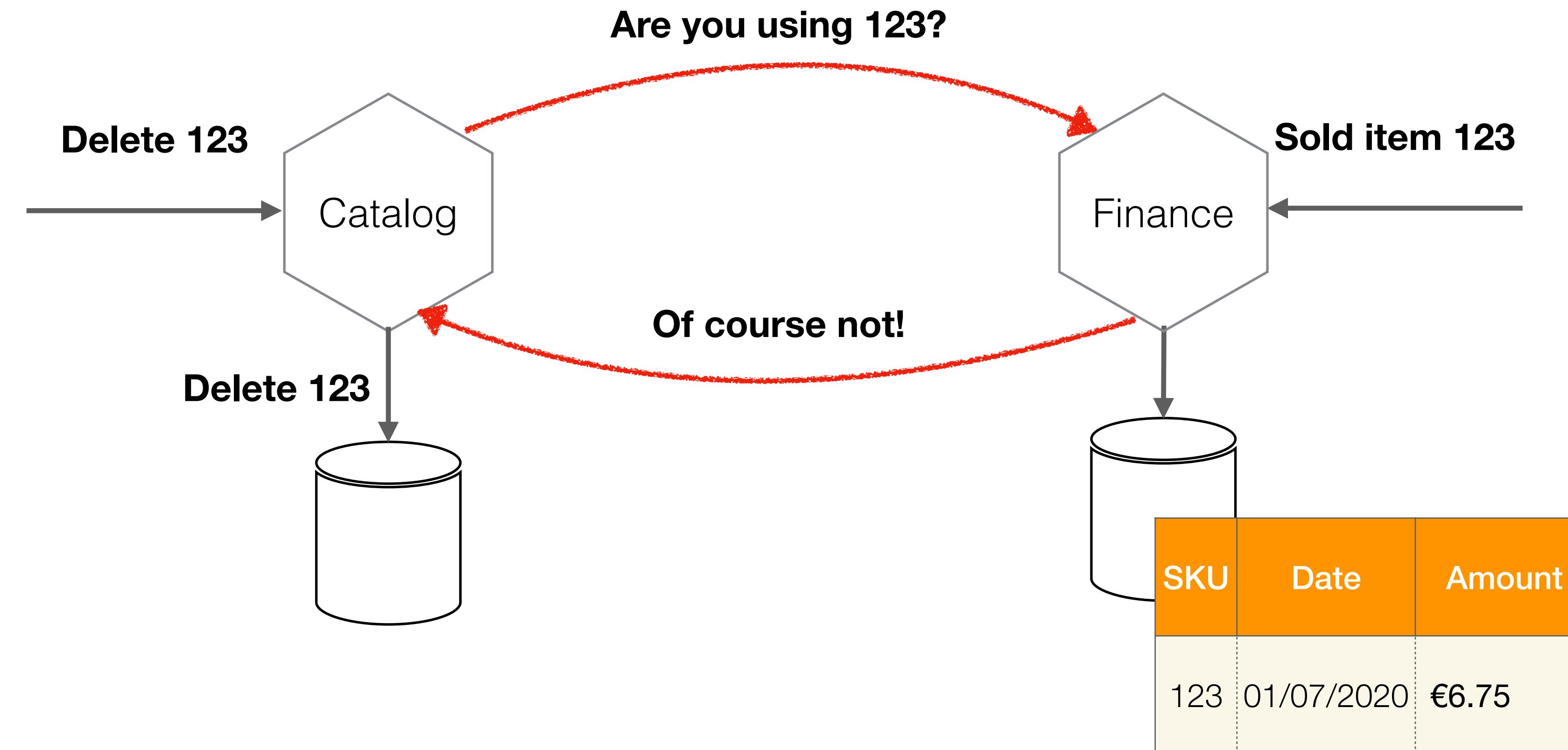
LOCKING REQUIRED?



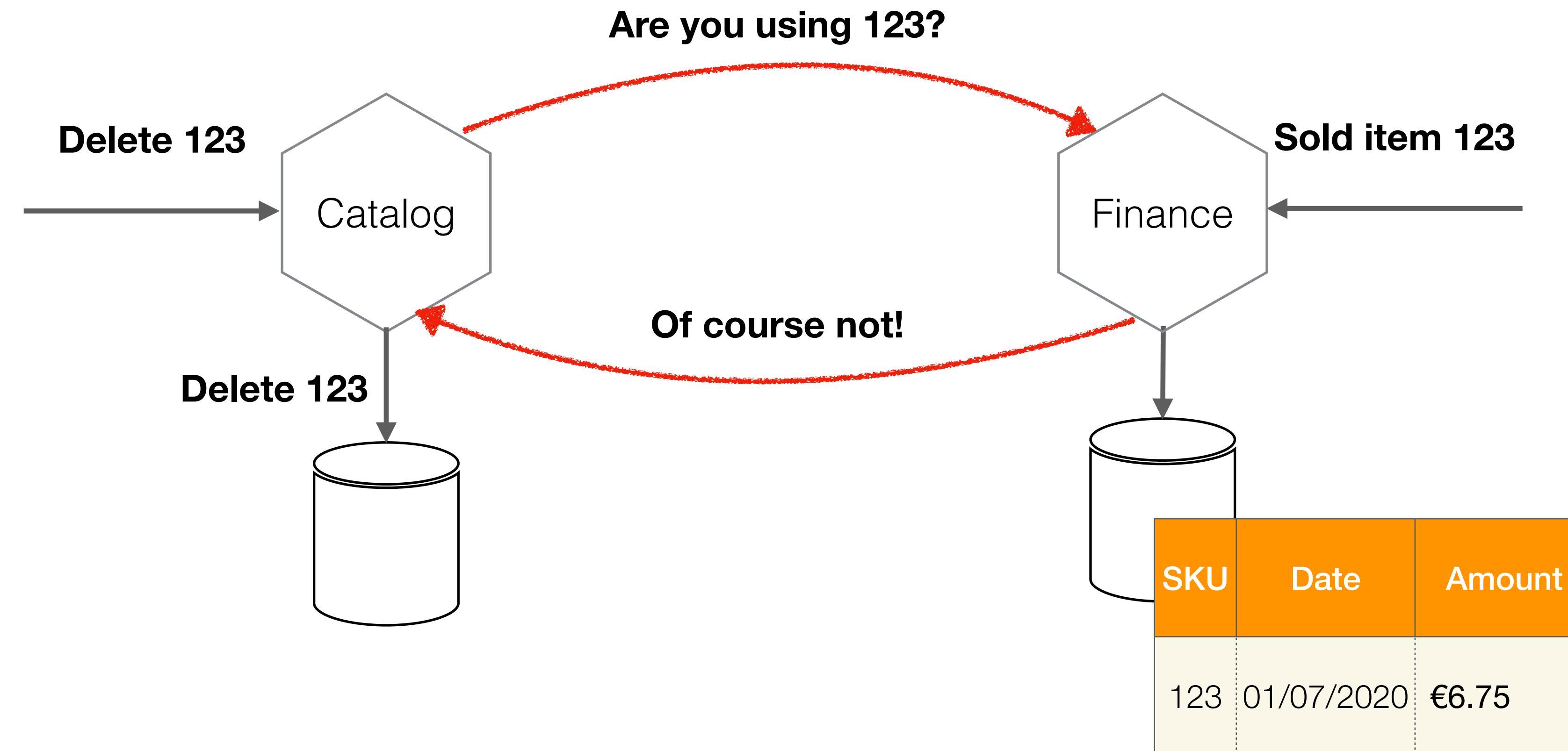
LOCKING REQUIRED?



LOCKING REQUIRED?

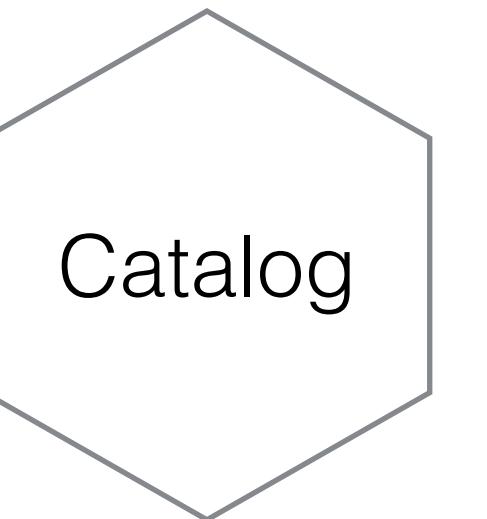


LOCKING REQUIRED?

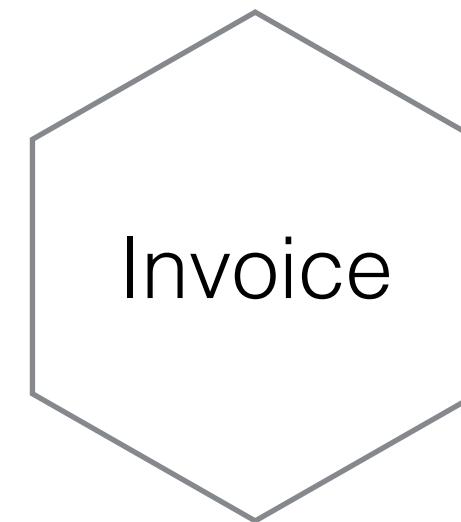


You need to lock the ledger table if you want to be certain that no references to 123 exist

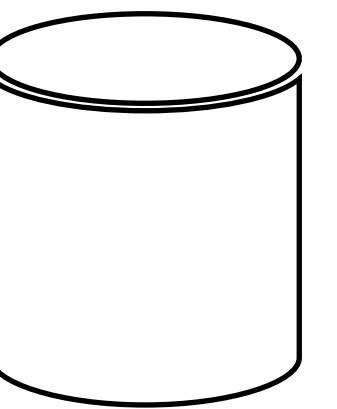
CASCADING DELETES?



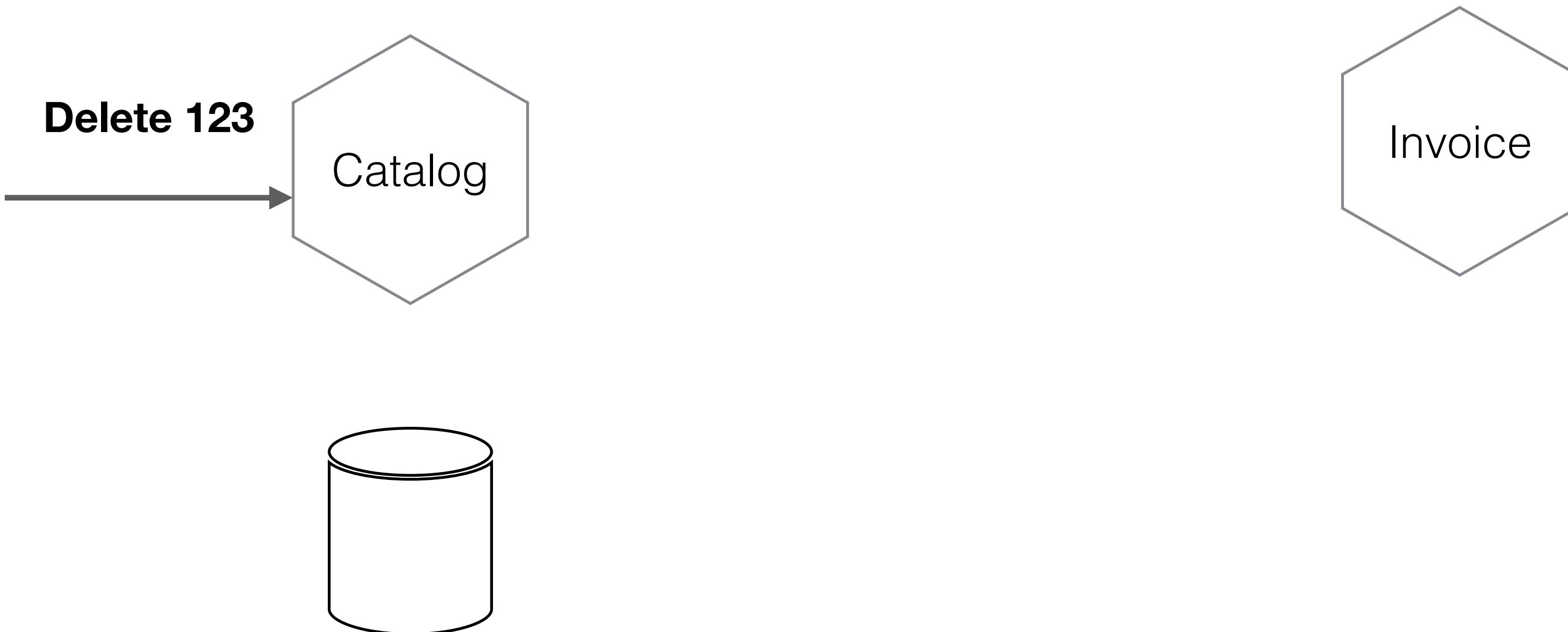
Catalog



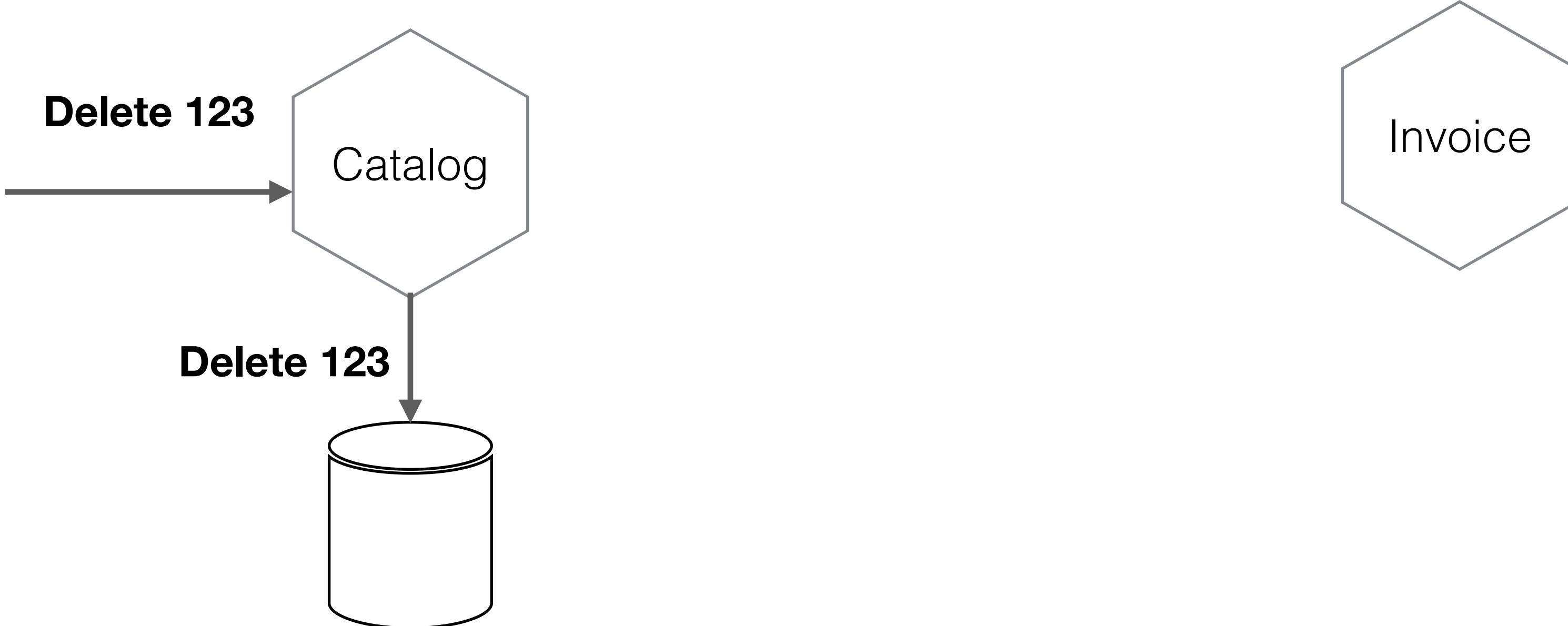
Invoice



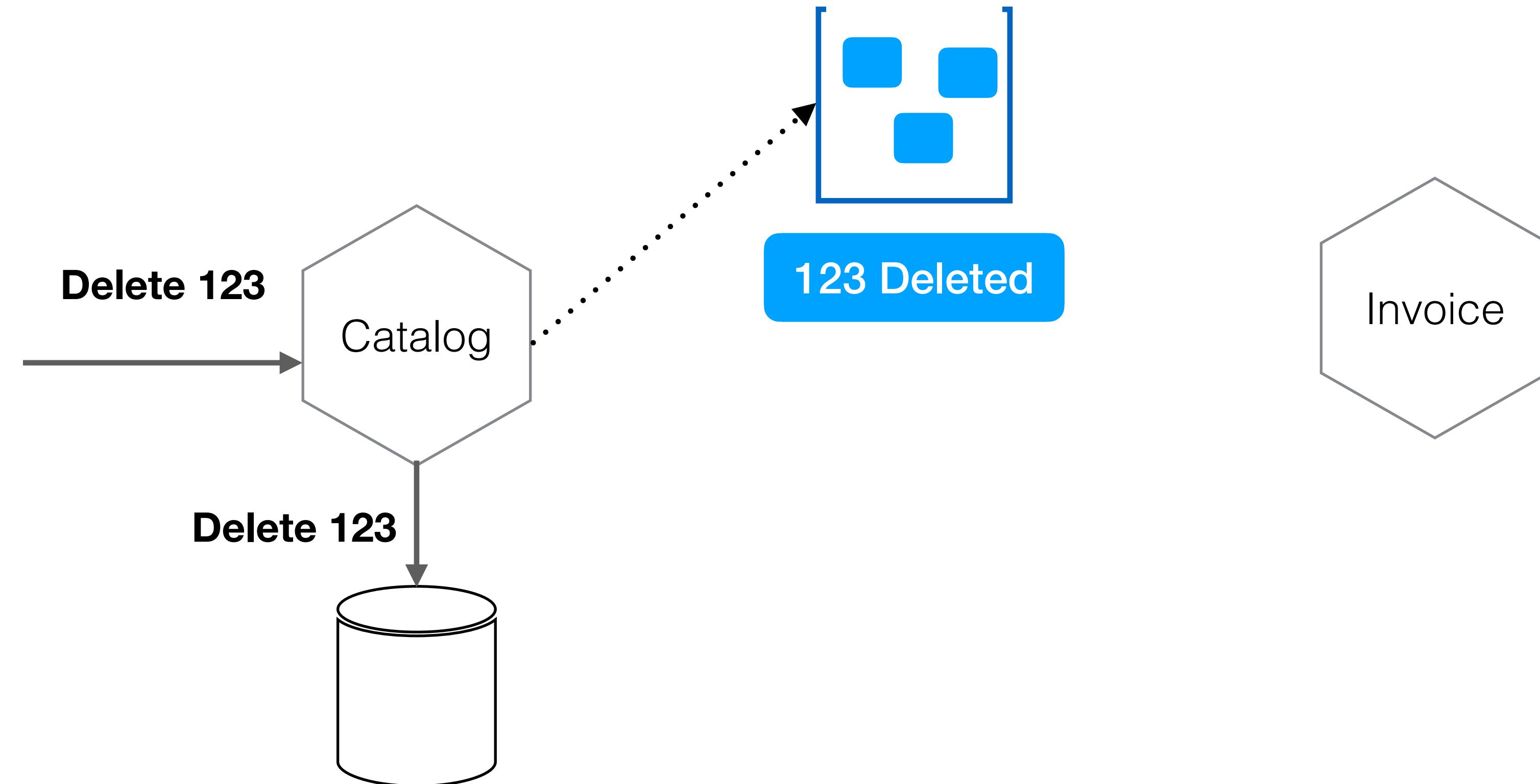
CASCADING DELETES?



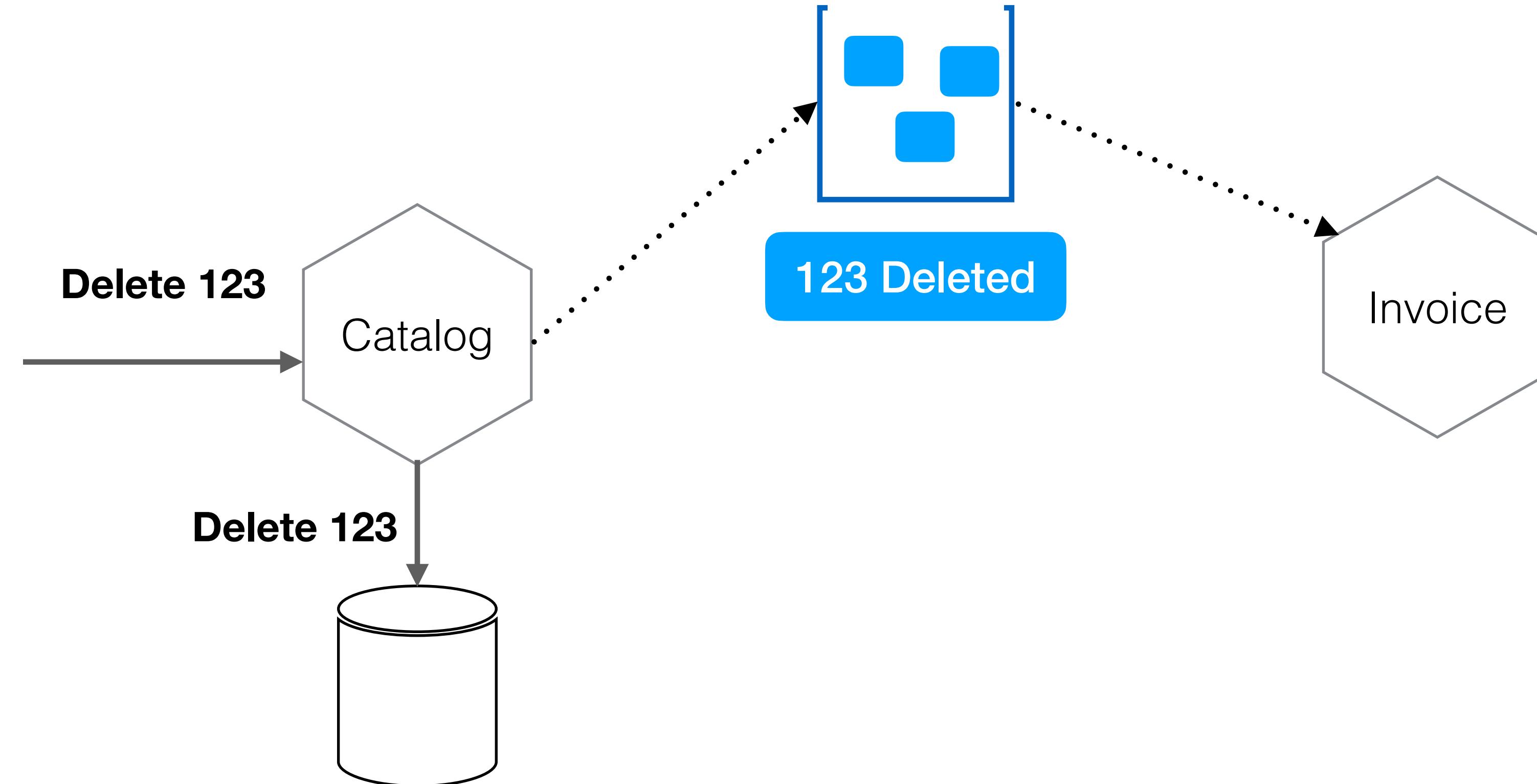
CASCADING DELETES?



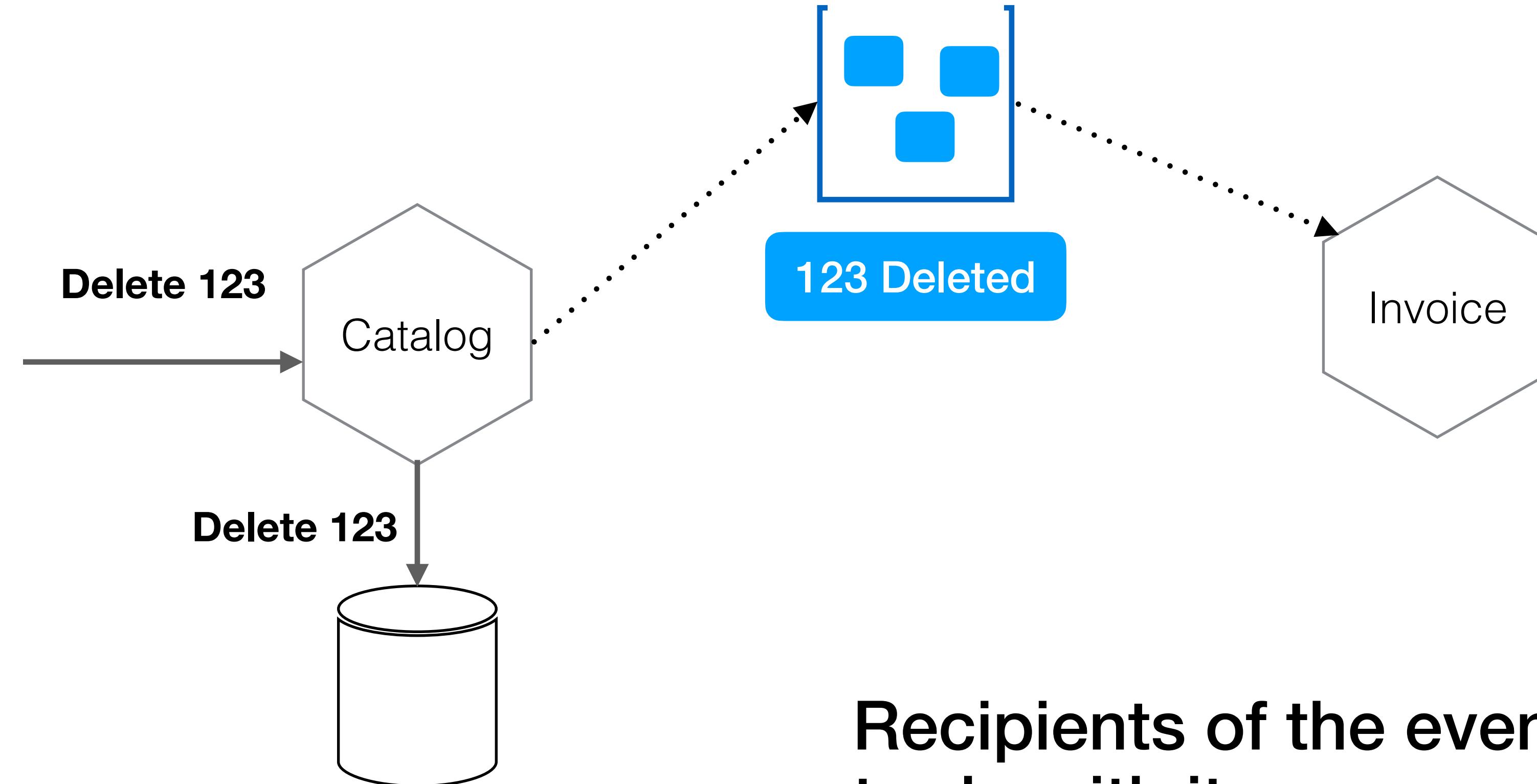
CASCADING DELETES?



CASCADING DELETES?

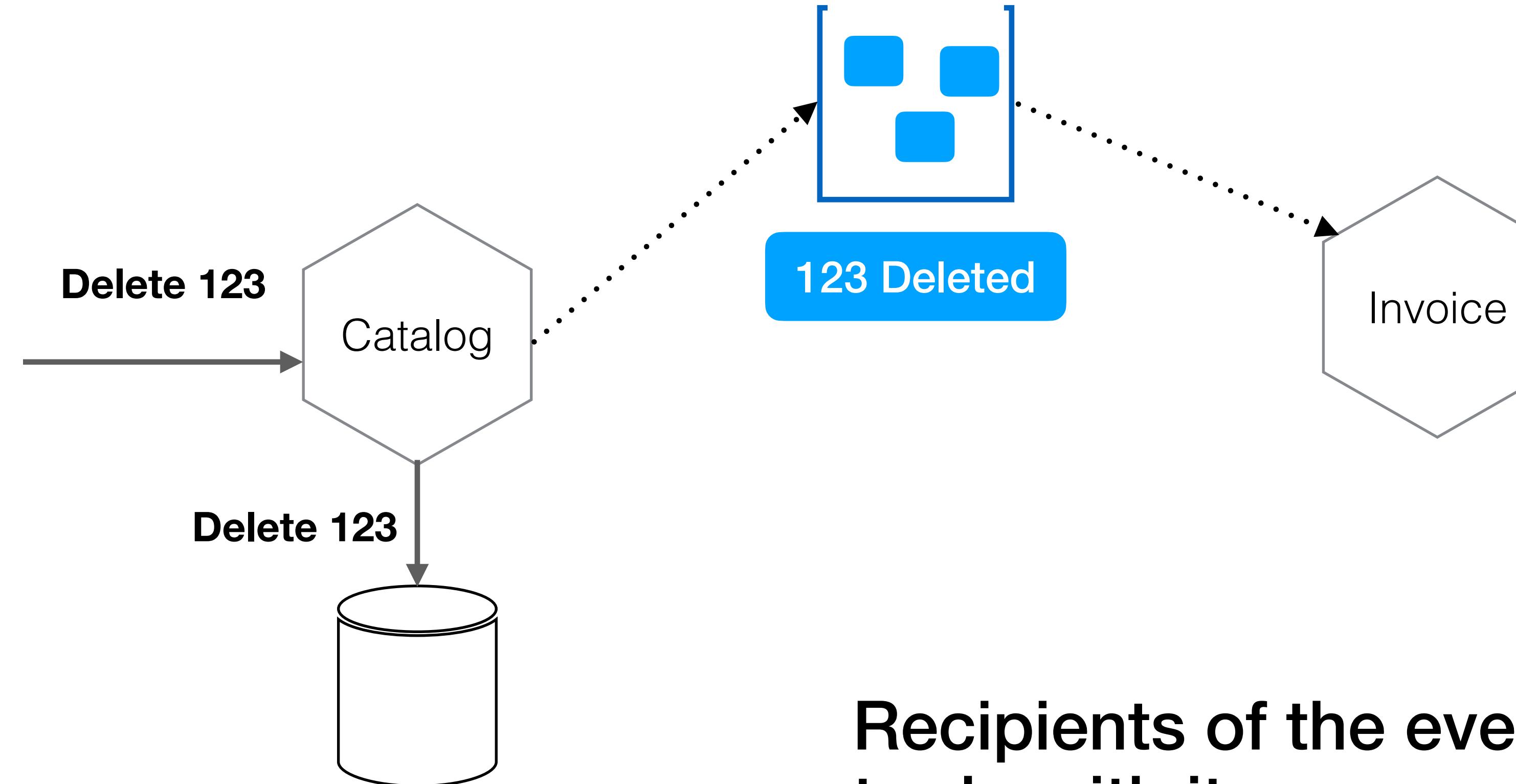


CASCADING DELETES?



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

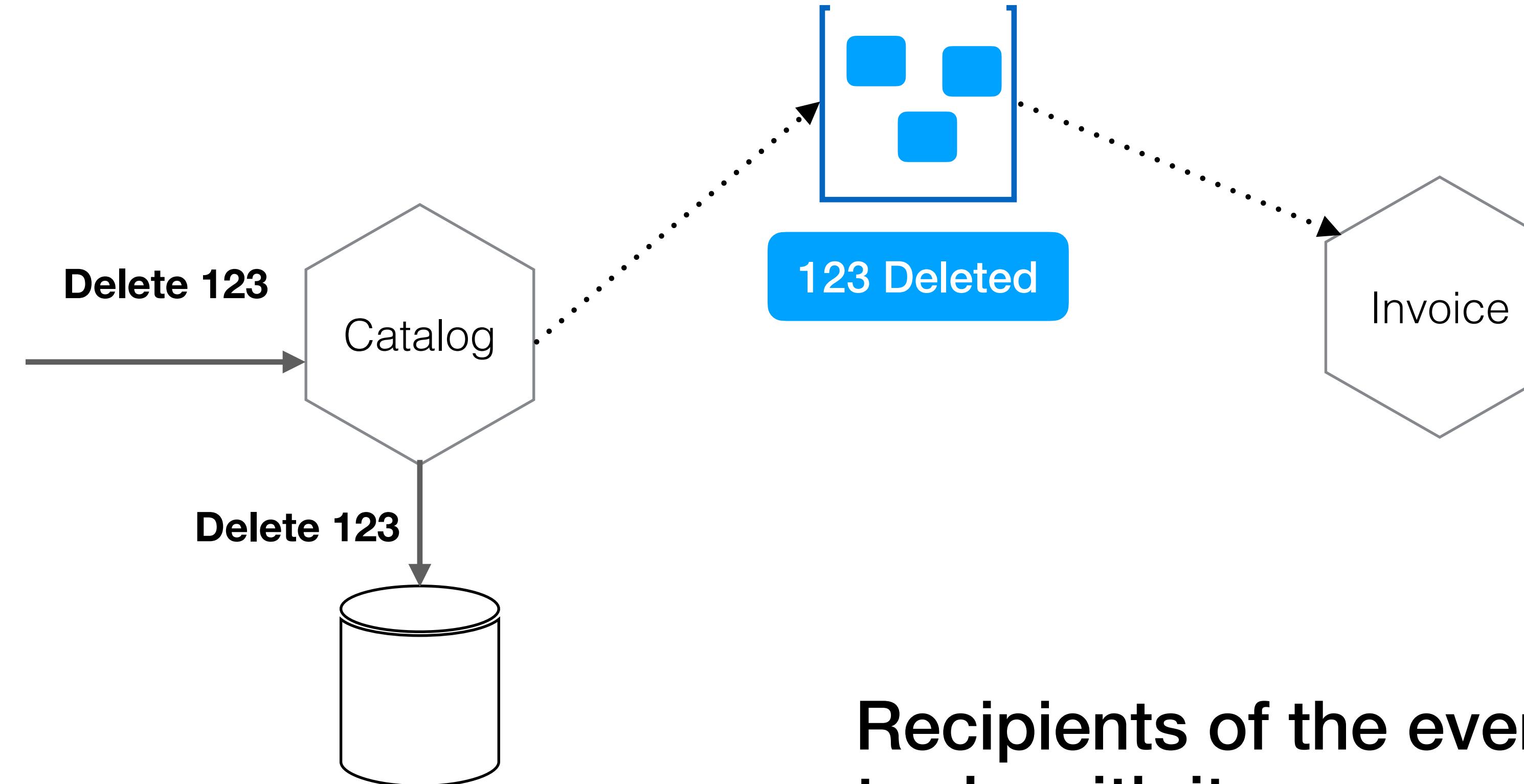
CASCADING DELETES?



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](#)

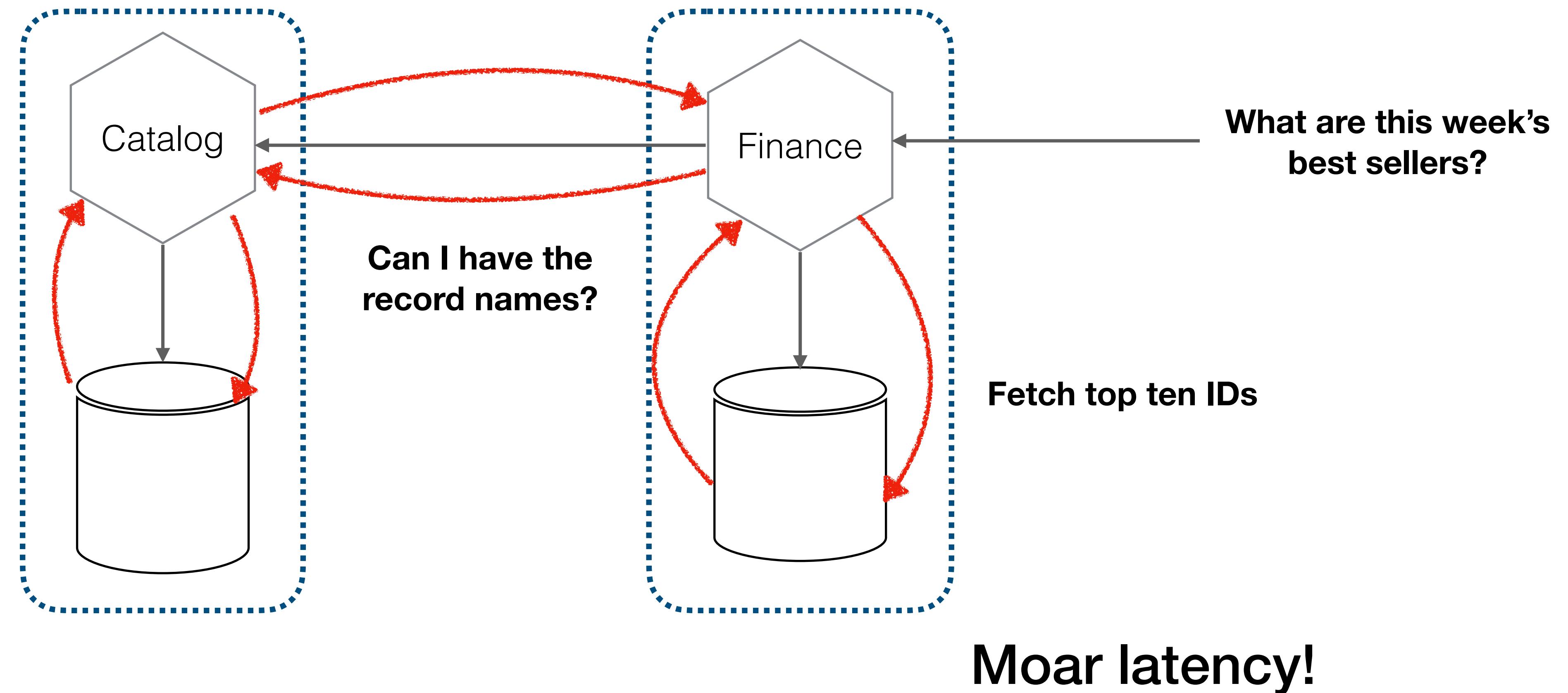
werner@allthingsdistributed.com

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

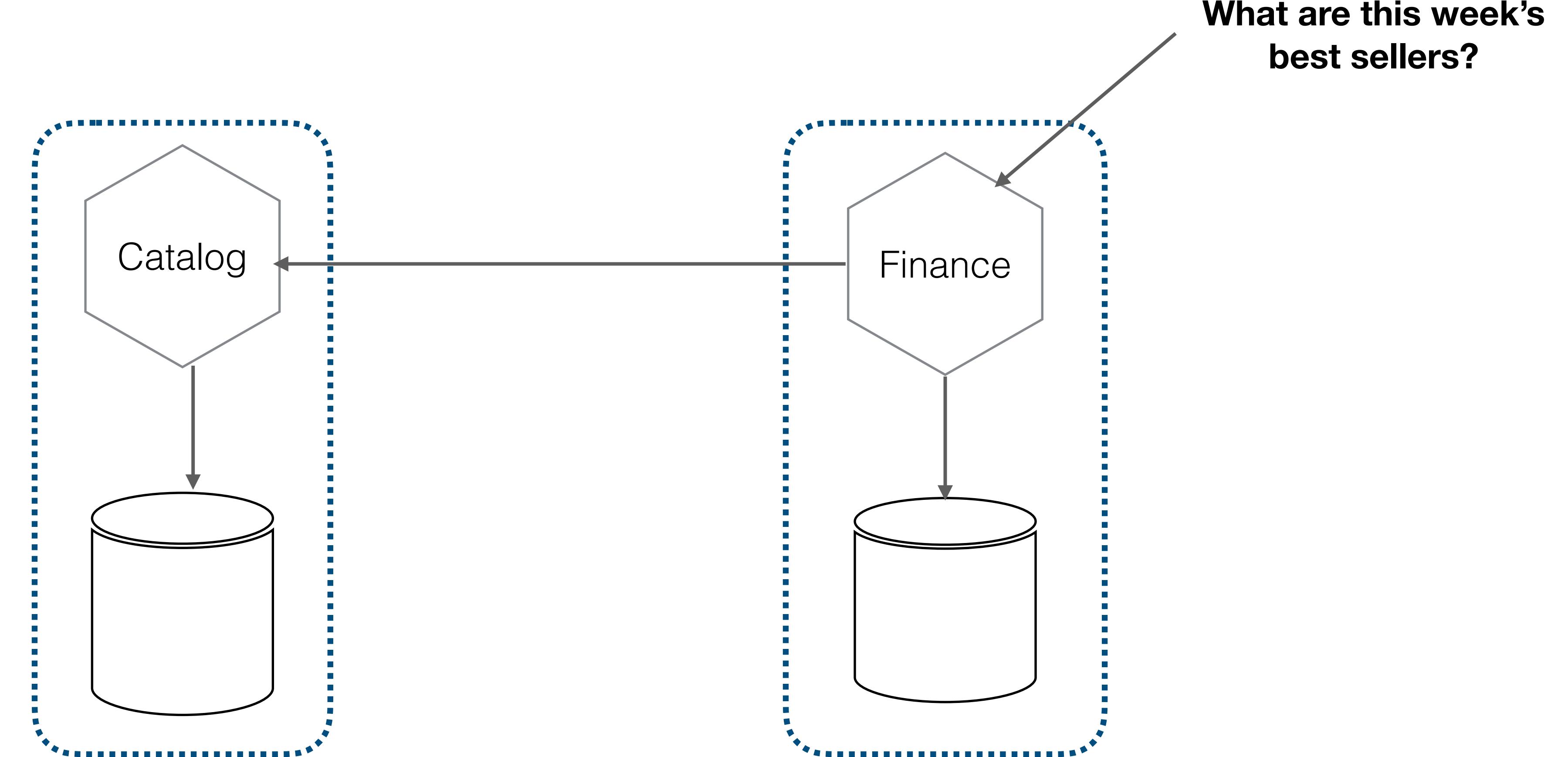
JOINS AT THE SERVICES TIER



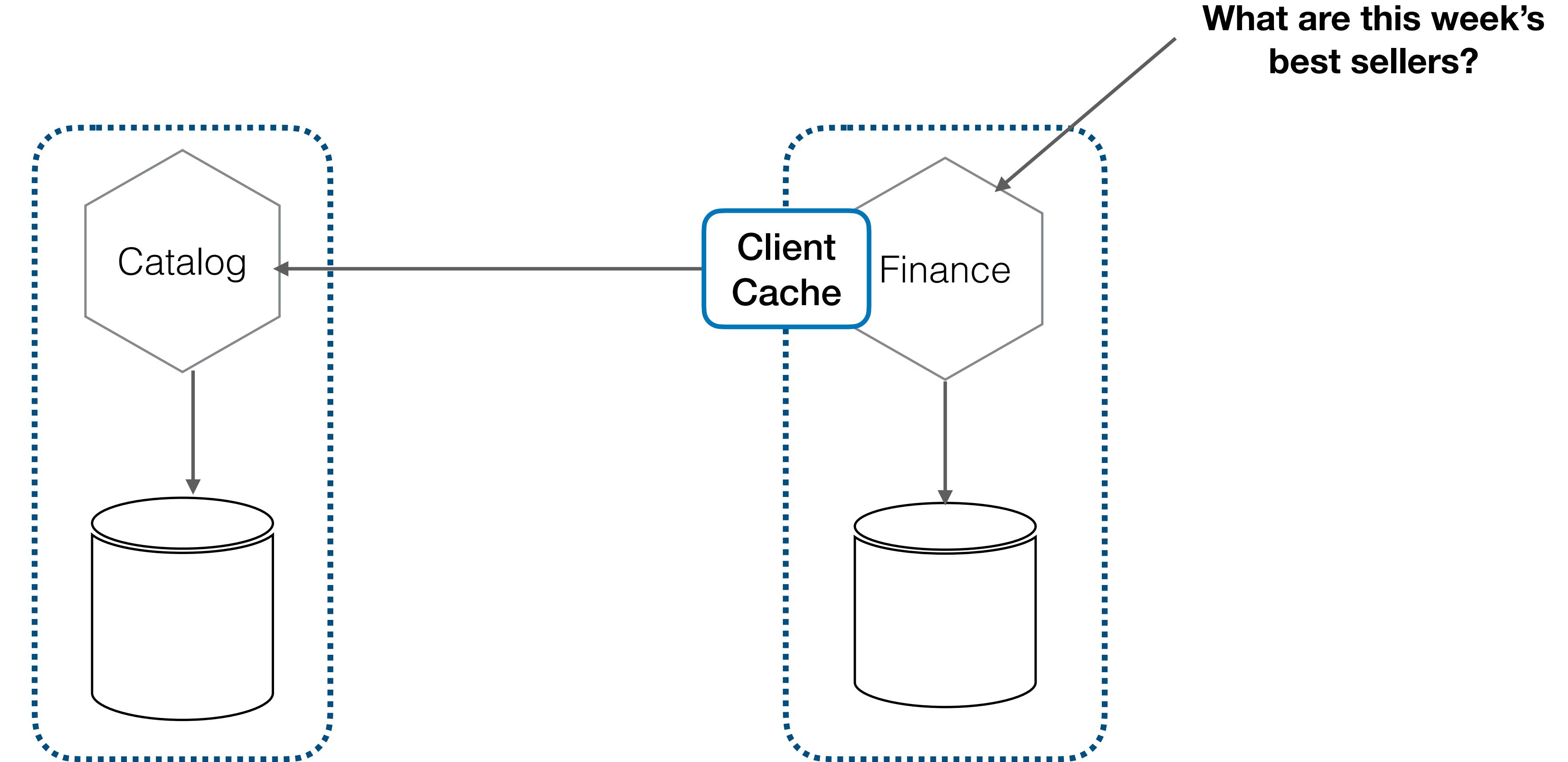
How can we make this join more efficient?

Caching!

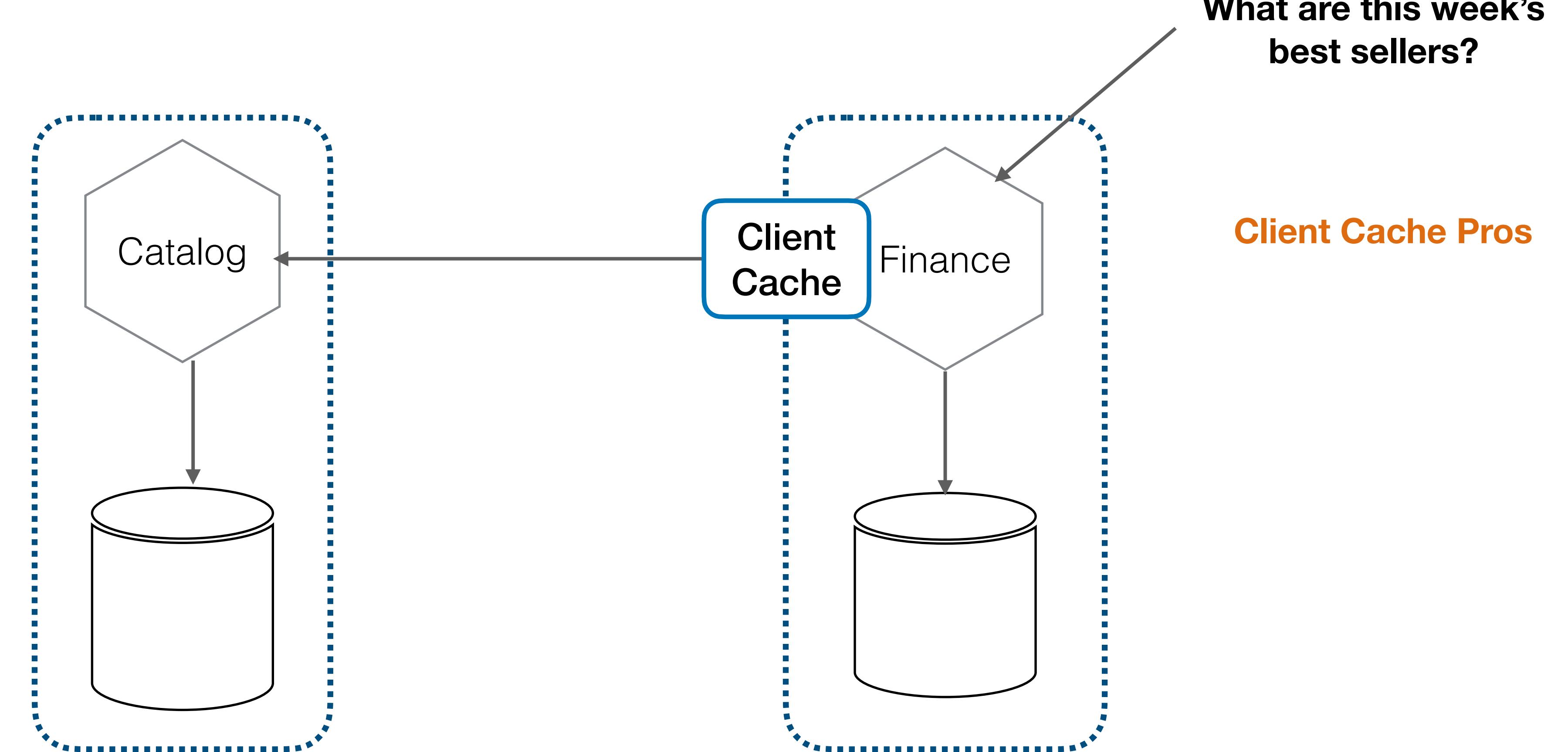
WHERE COULD WE CACHE?



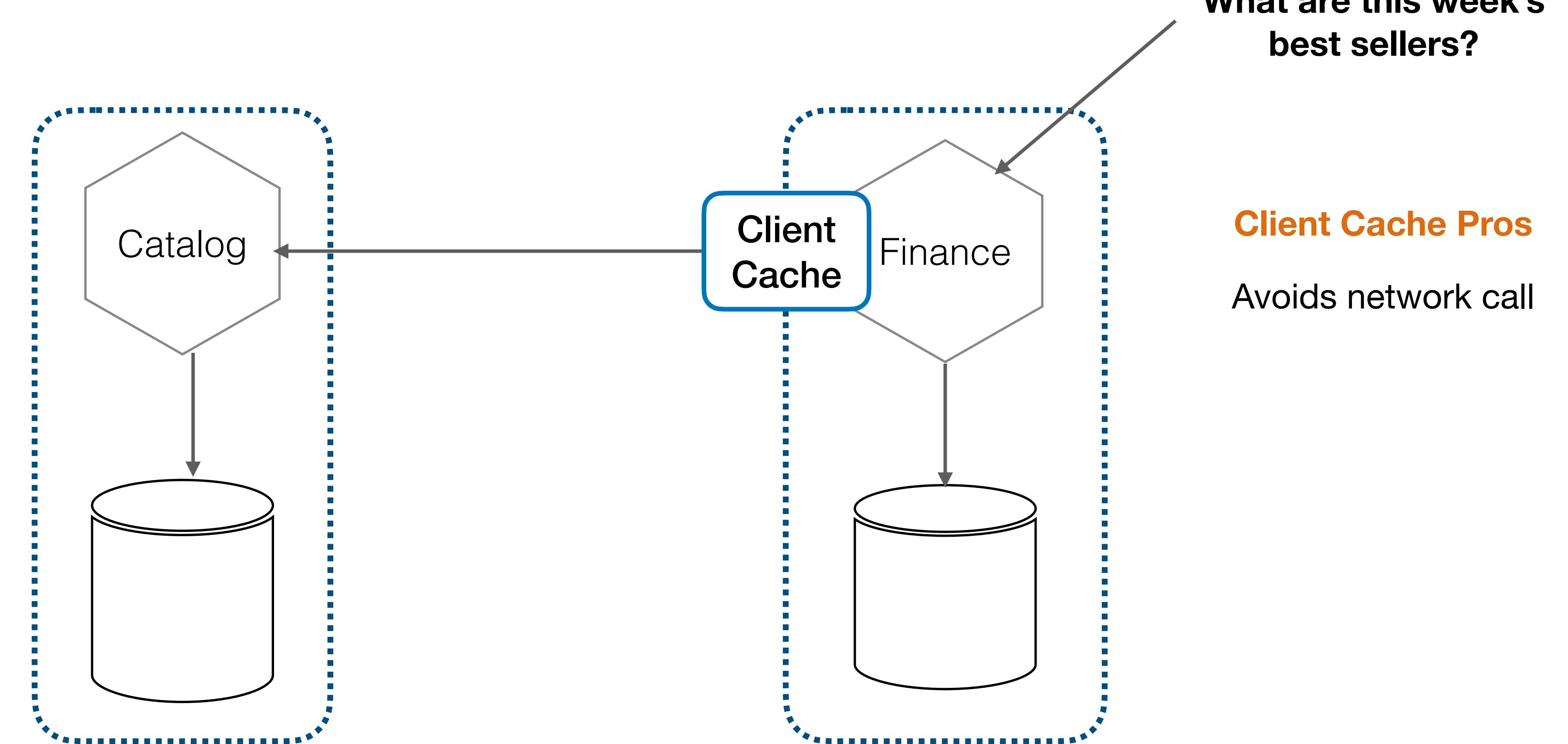
WHERE COULD WE CACHE?



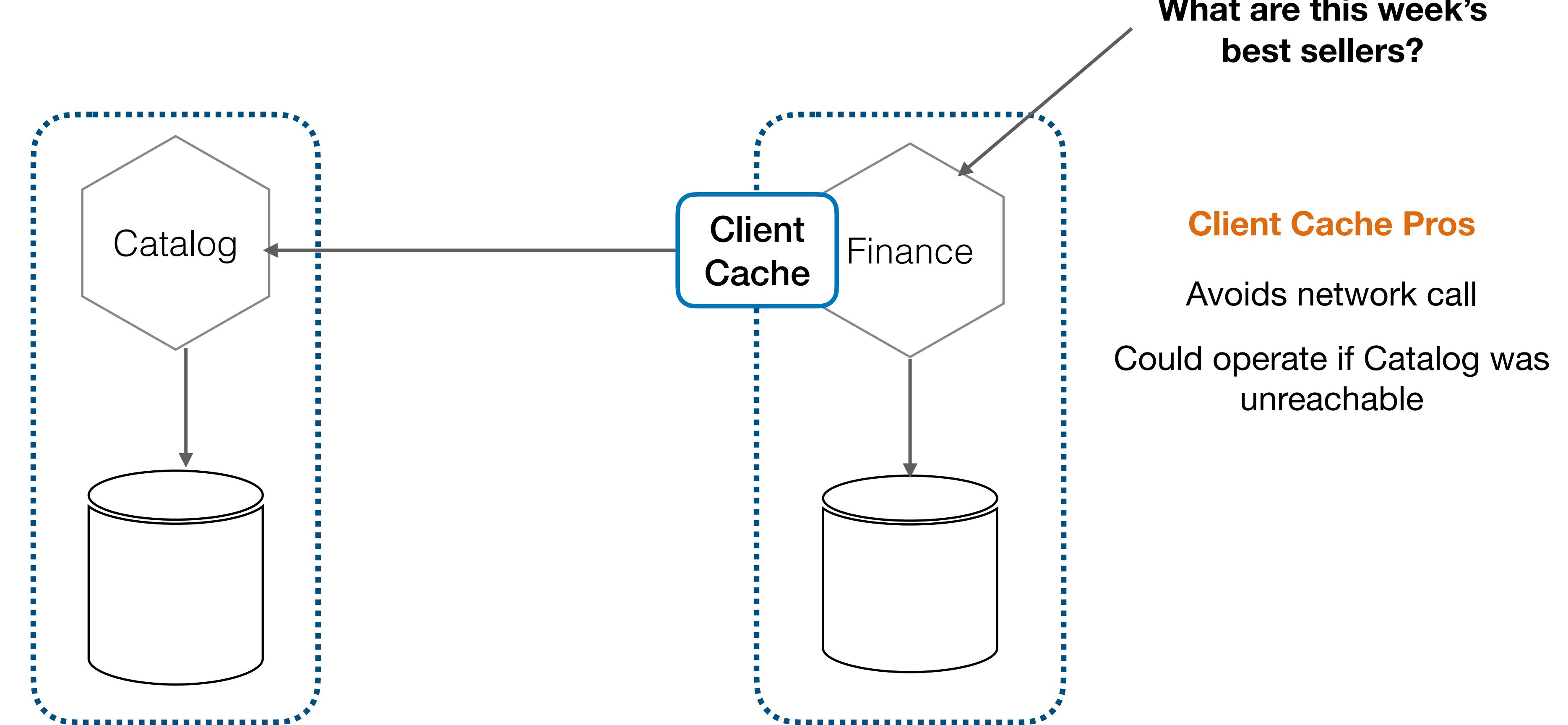
WHERE COULD WE CACHE?



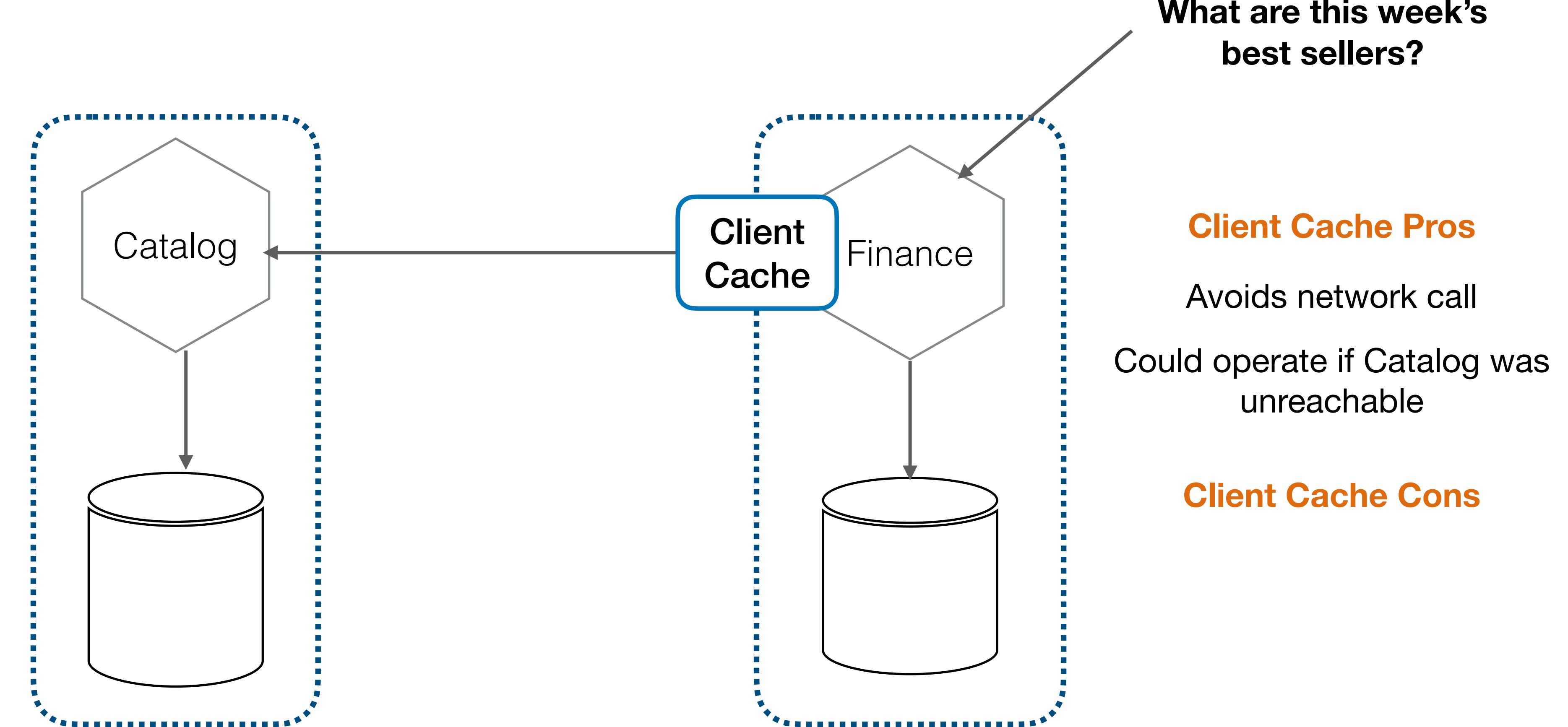
WHERE COULD WE CACHE?



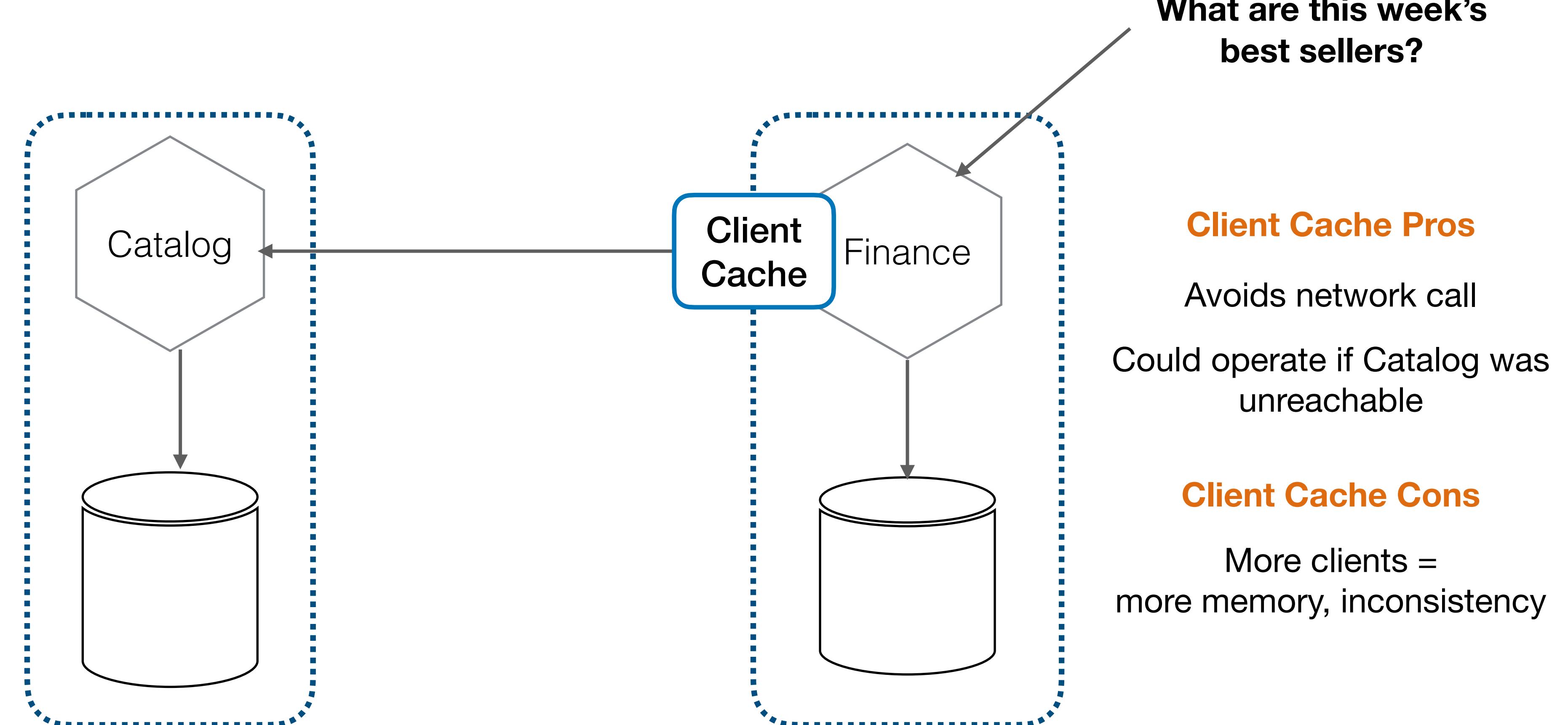
WHERE COULD WE CACHE?



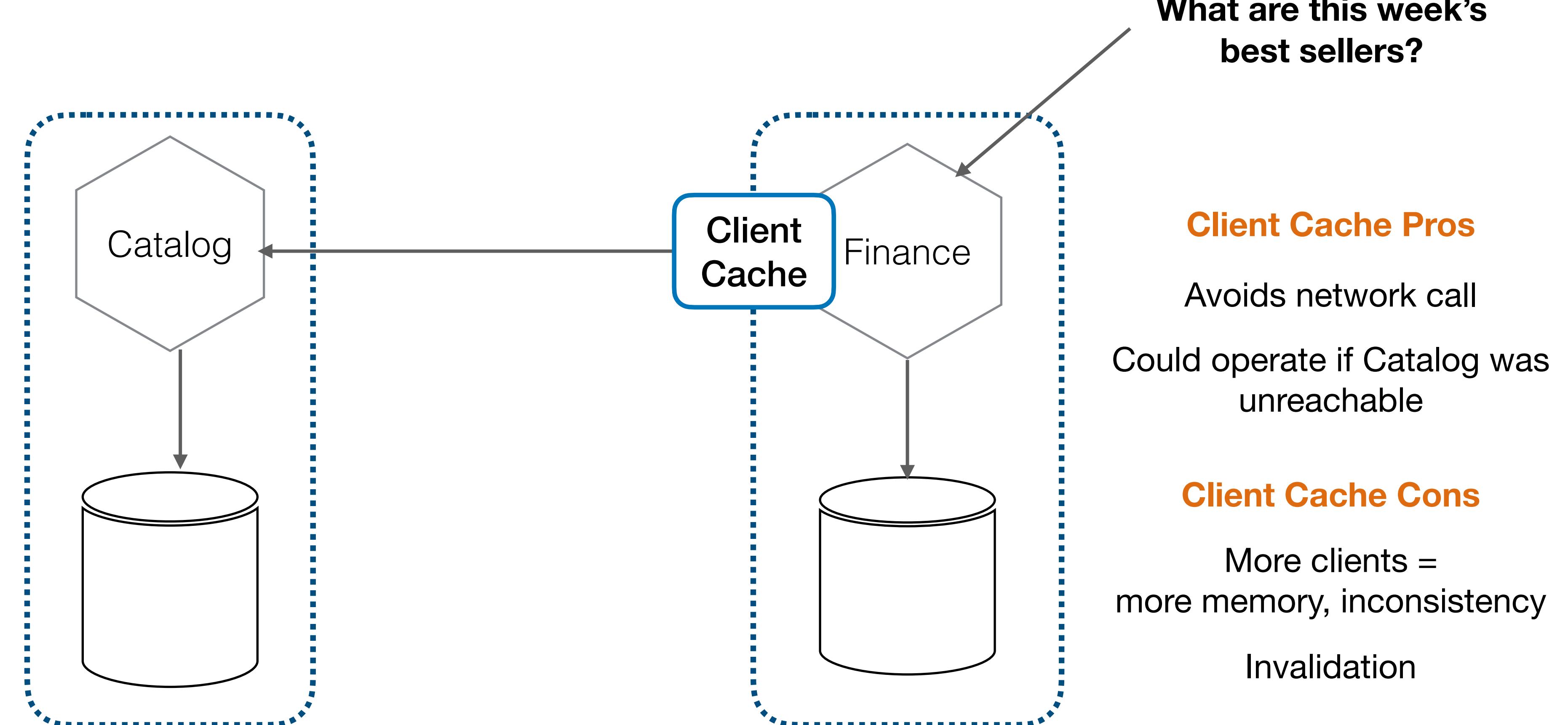
WHERE COULD WE CACHE?



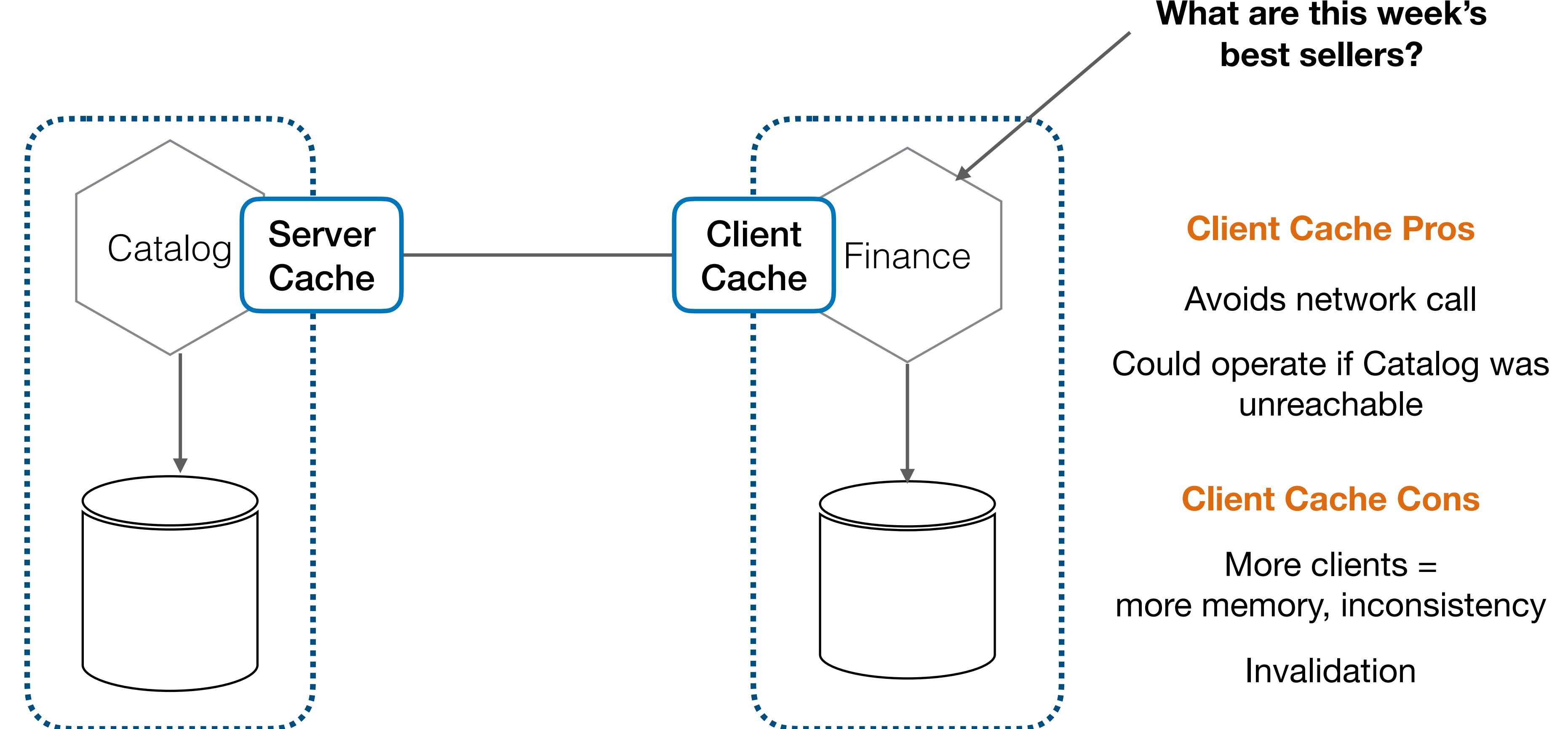
WHERE COULD WE CACHE?



WHERE COULD WE CACHE?

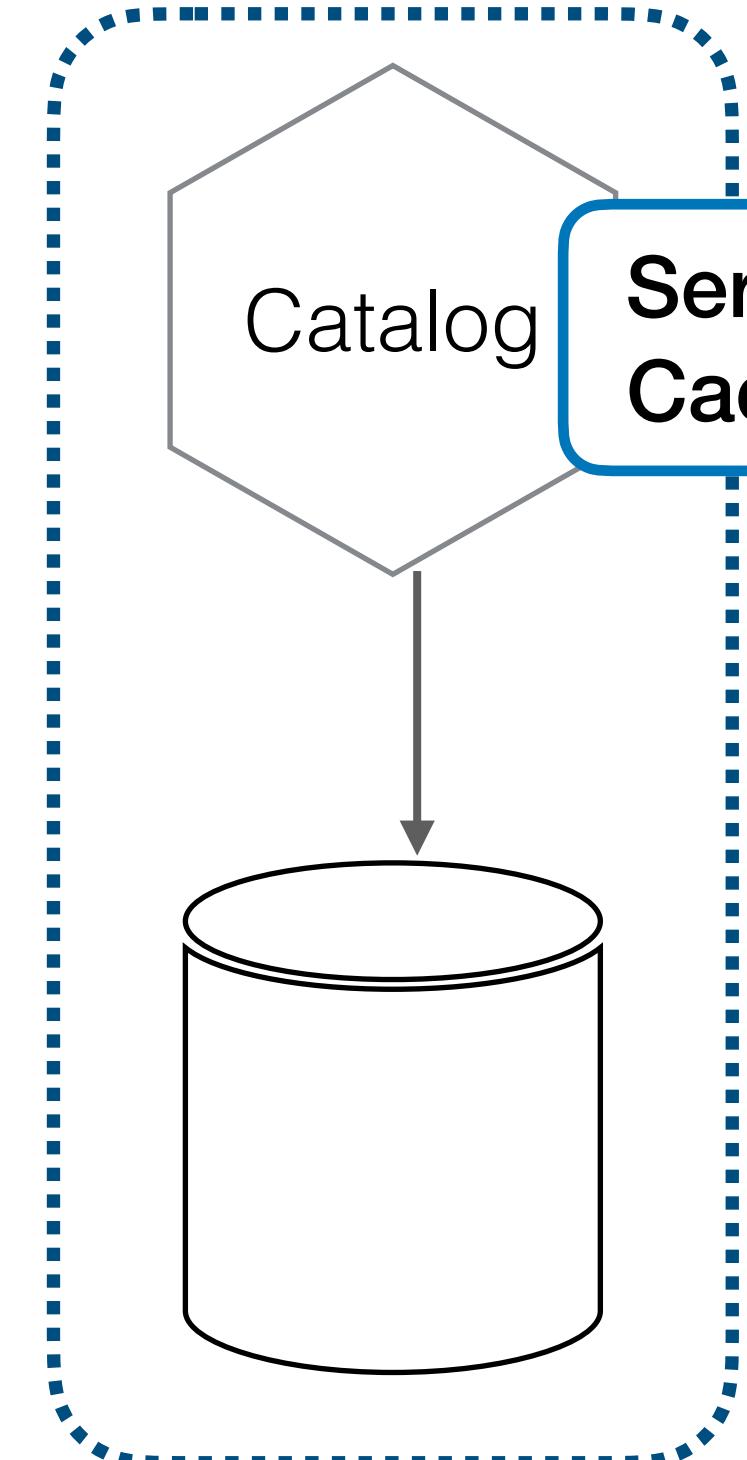


WHERE COULD WE CACHE?

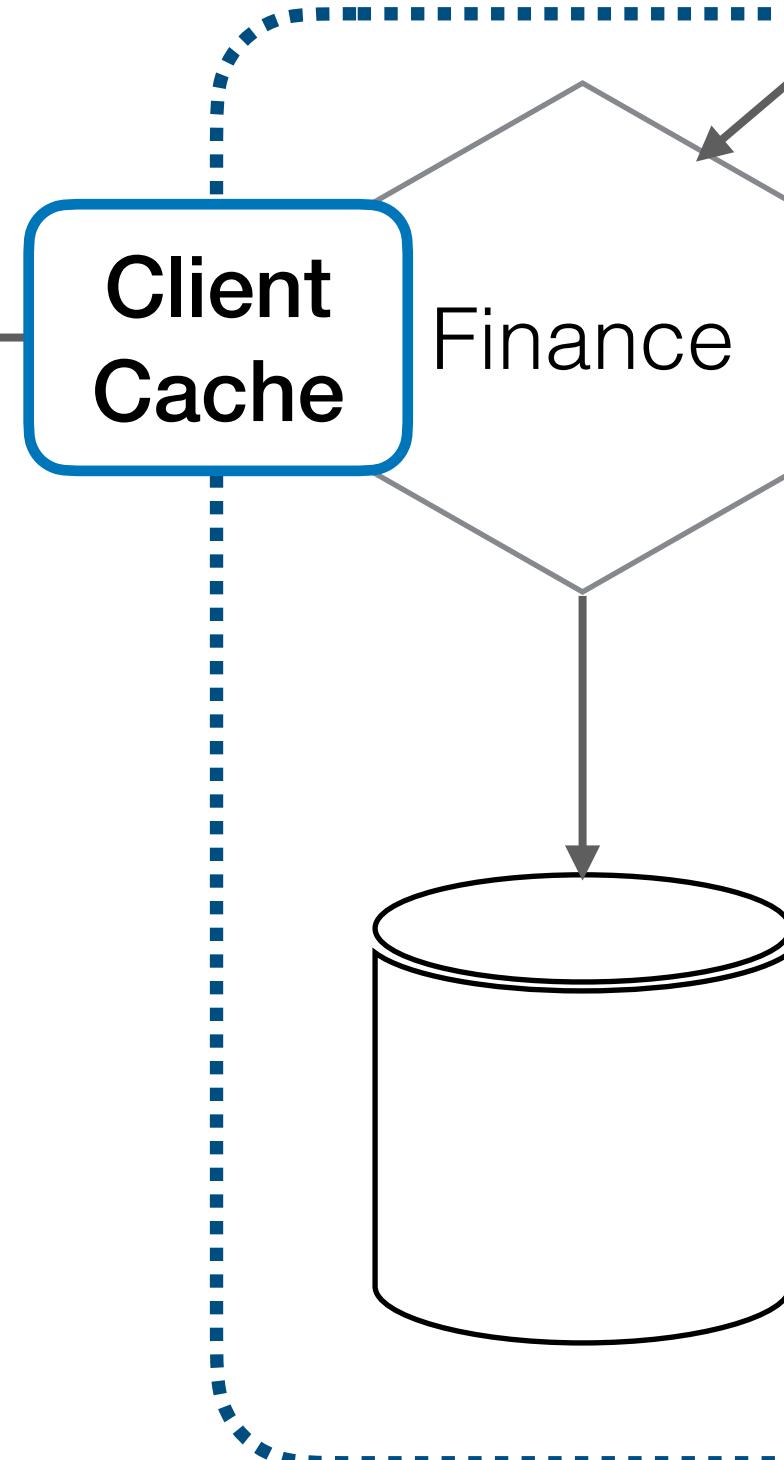


WHERE COULD WE CACHE?

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

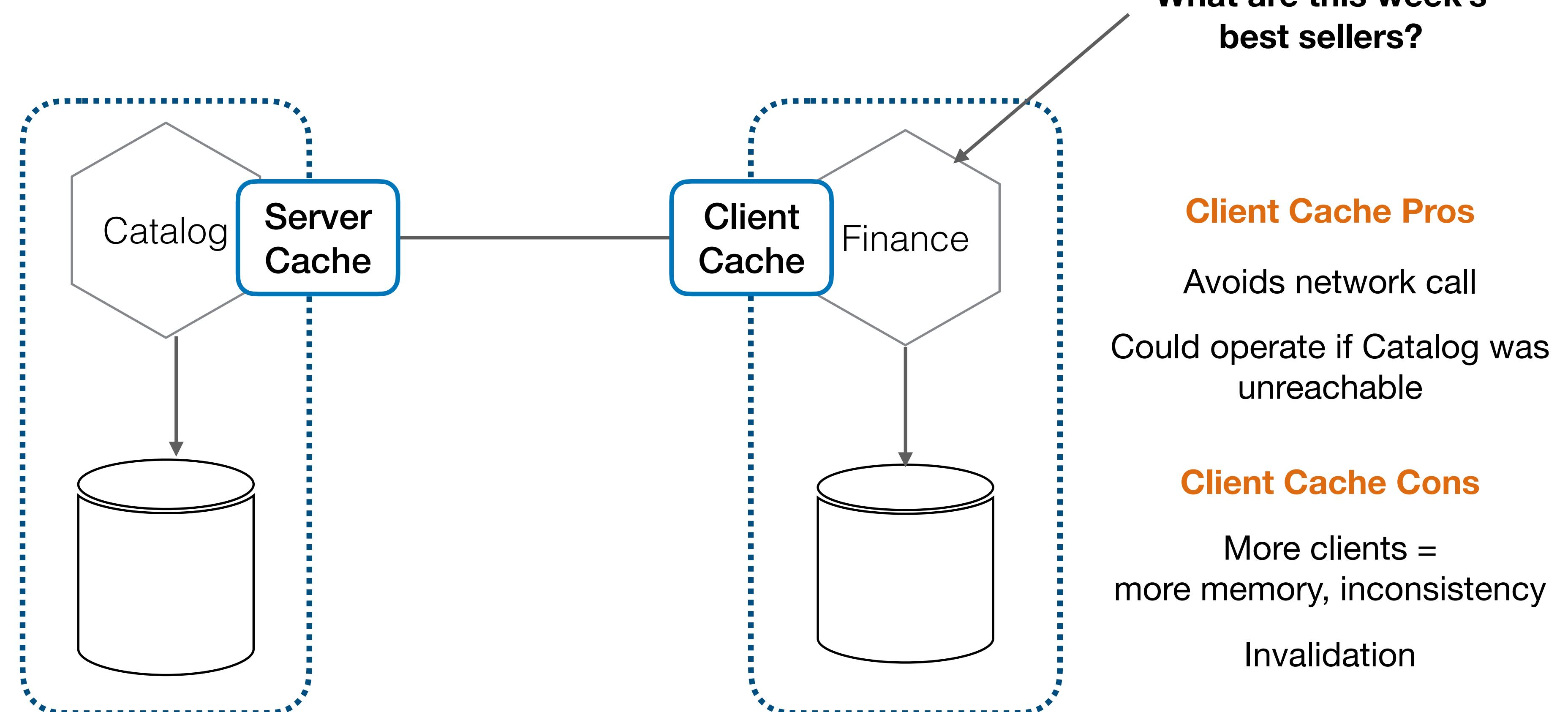
More clients = more memory, inconsistency

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

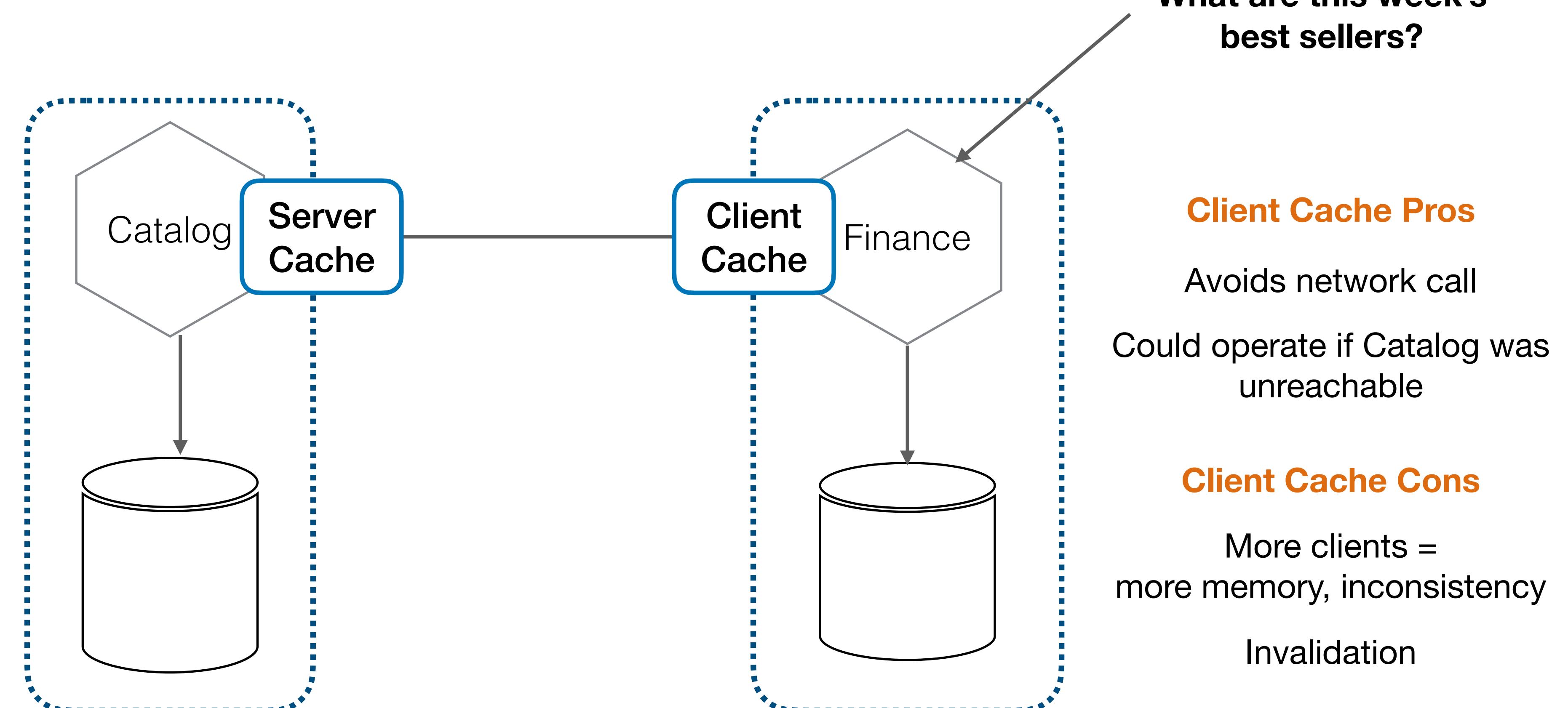
Invalidation

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

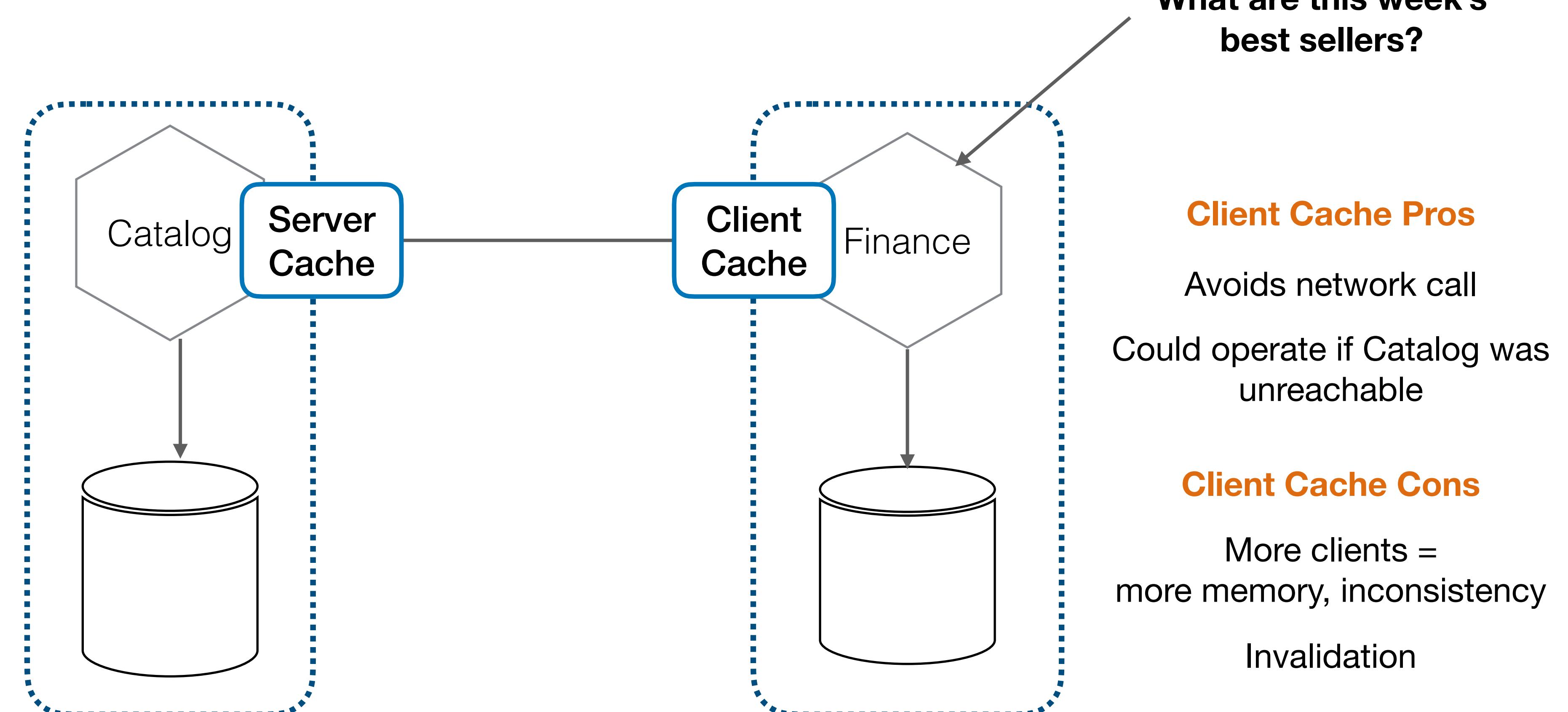
WHERE COULD WE CACHE?

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?

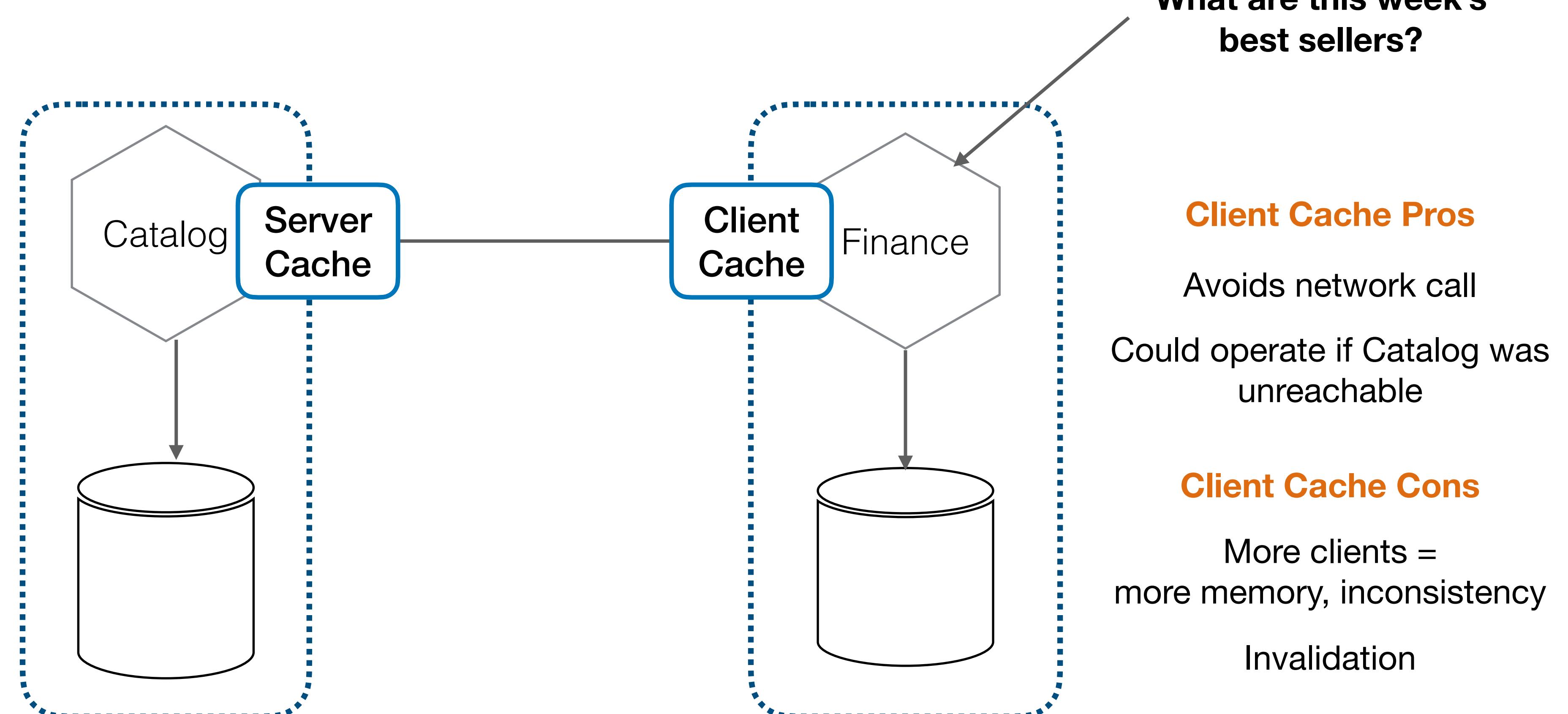
Server Cache Pros

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?

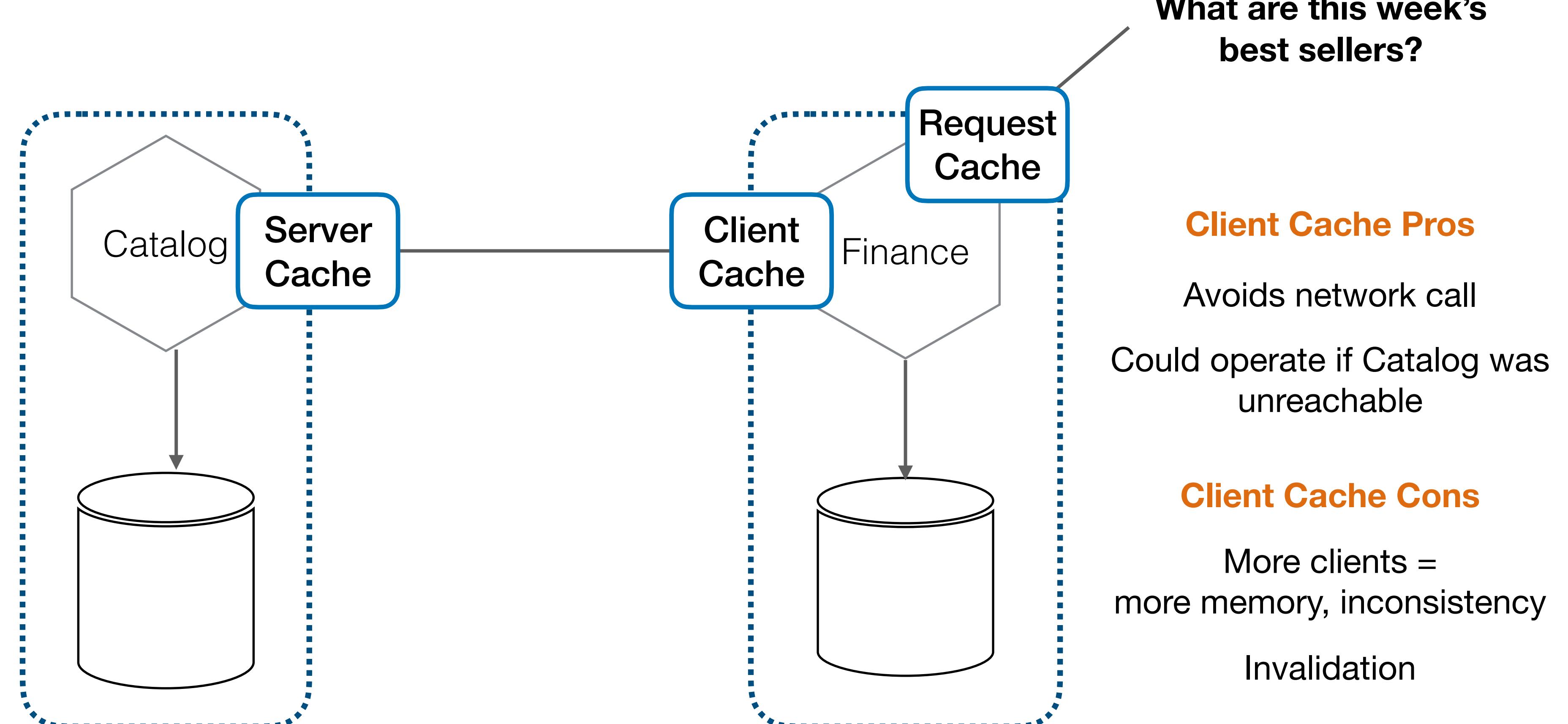
Server Cache Pros

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

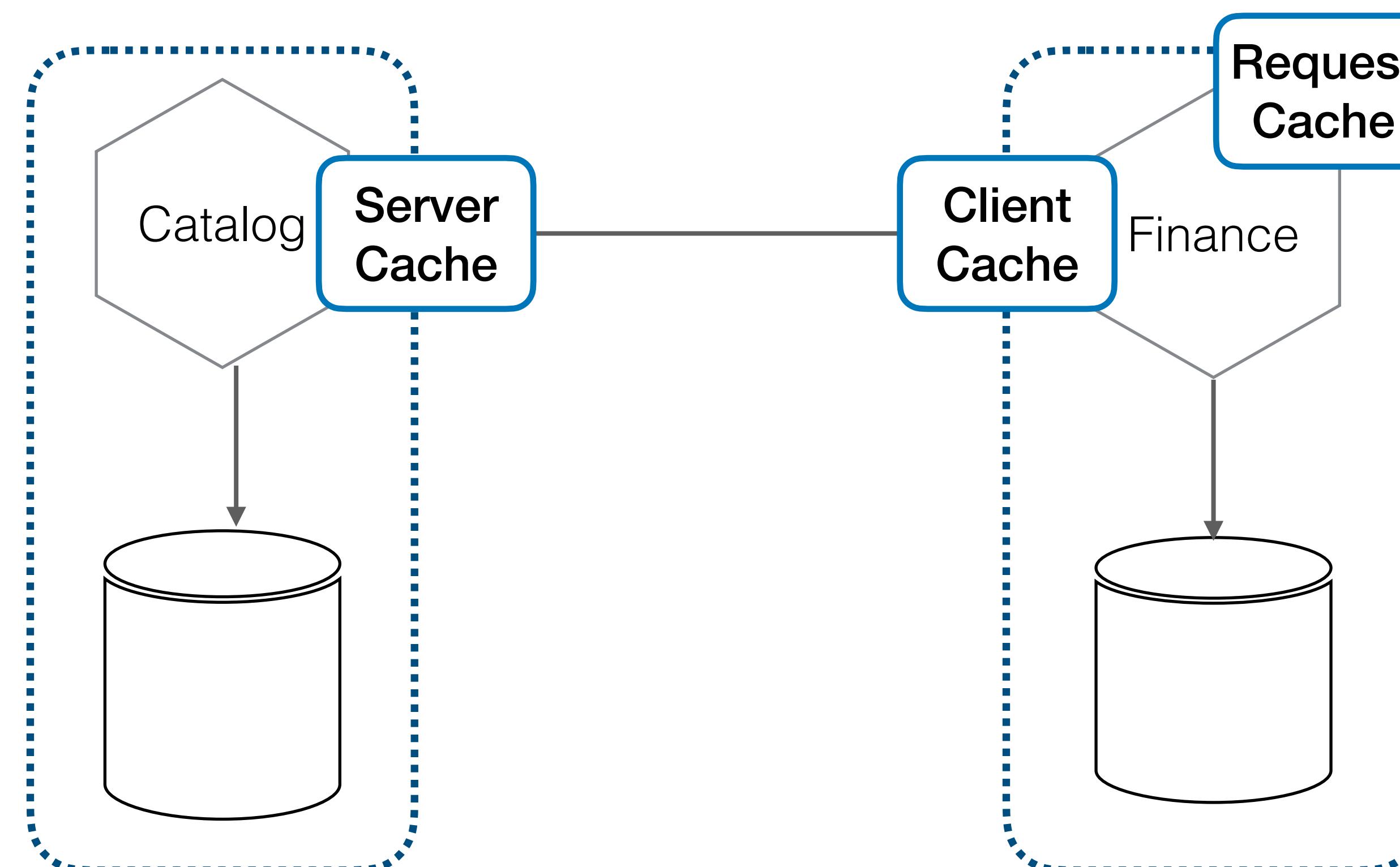
More clients = more memory, inconsistency

Invalidation

WHERE COULD WE CACHE?

Request Cache Pros

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?

Request Cache Pros

Super efficient

What are this week's best sellers?

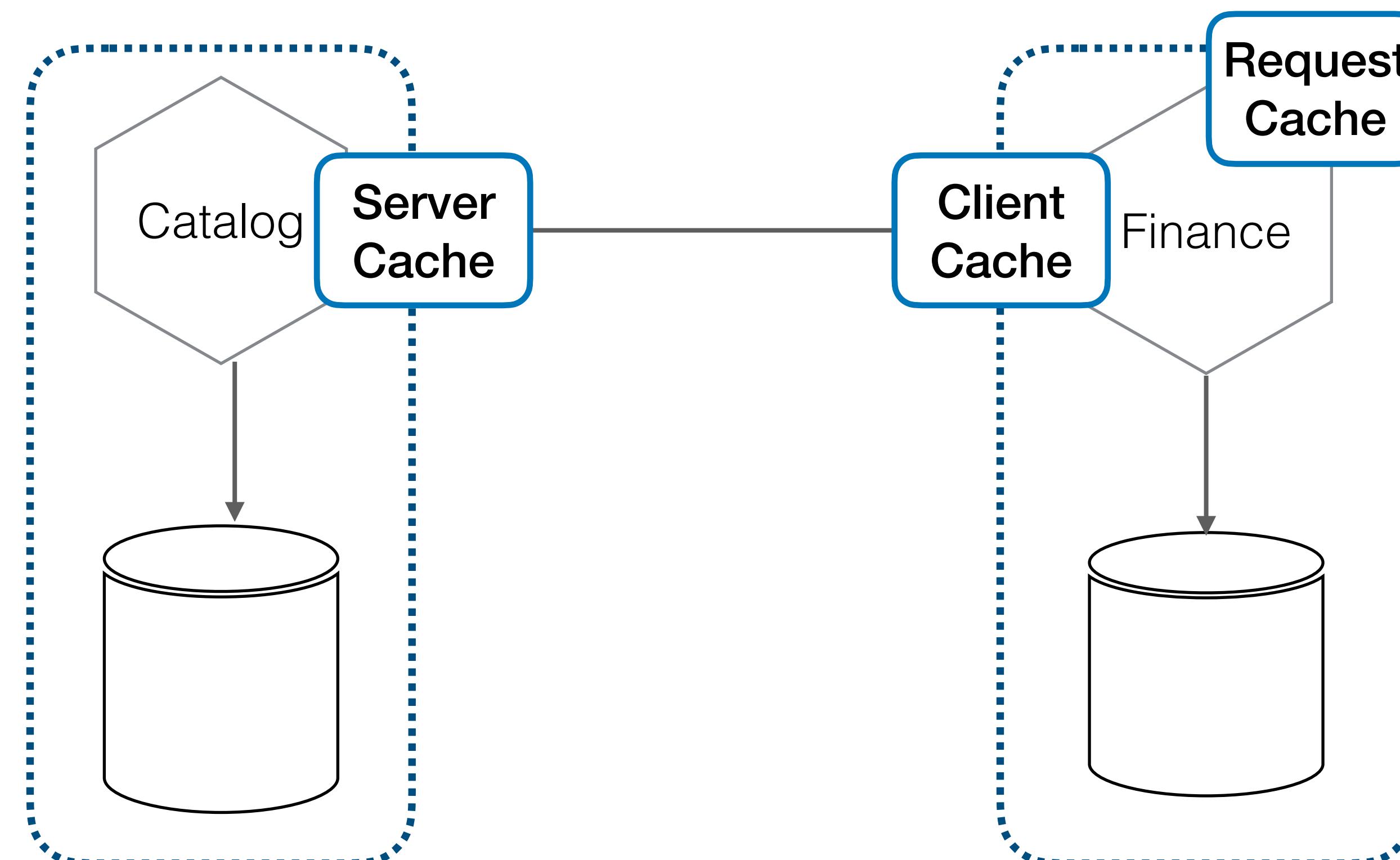
Server Cache Pros

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

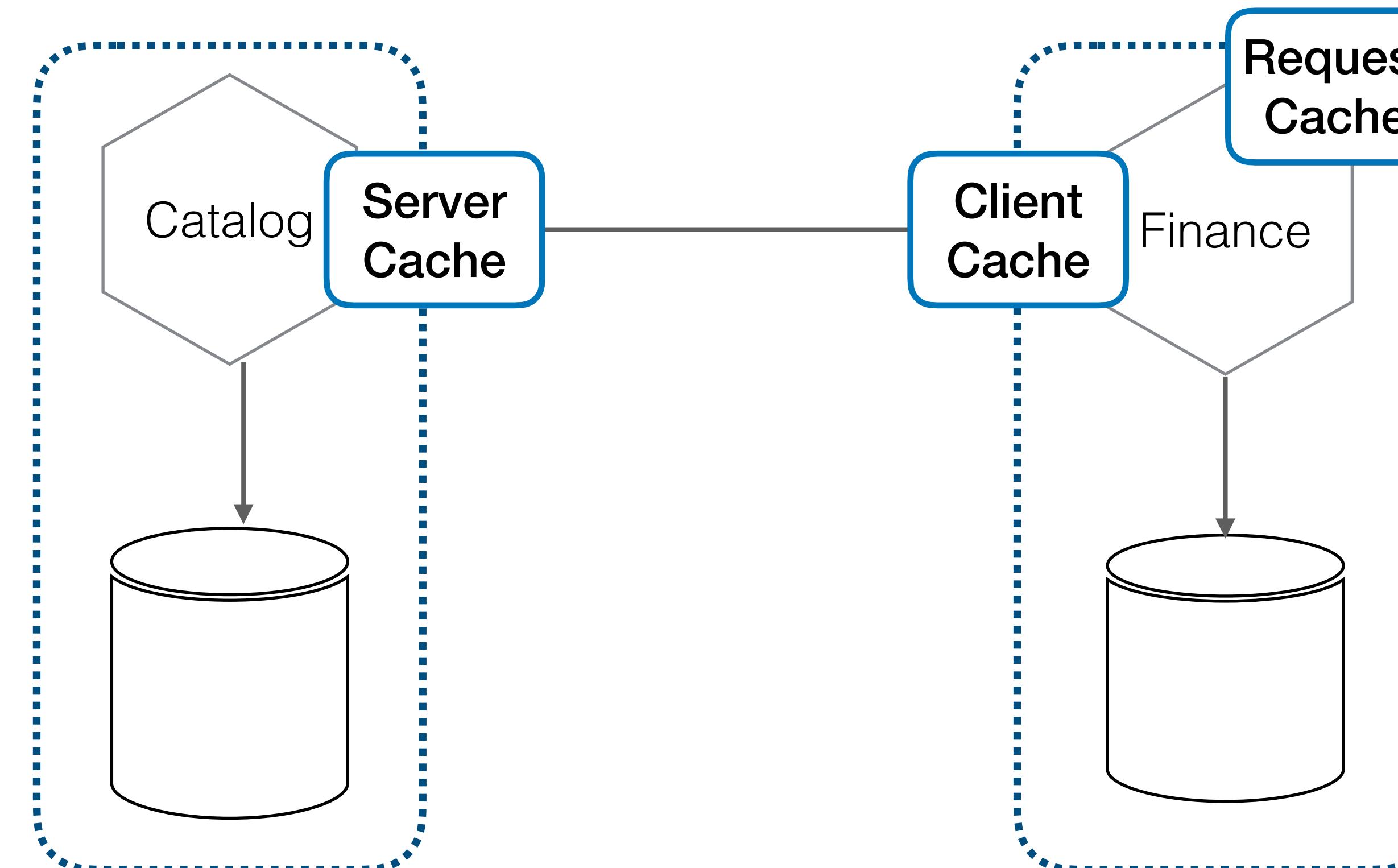
Call still needs to be made

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

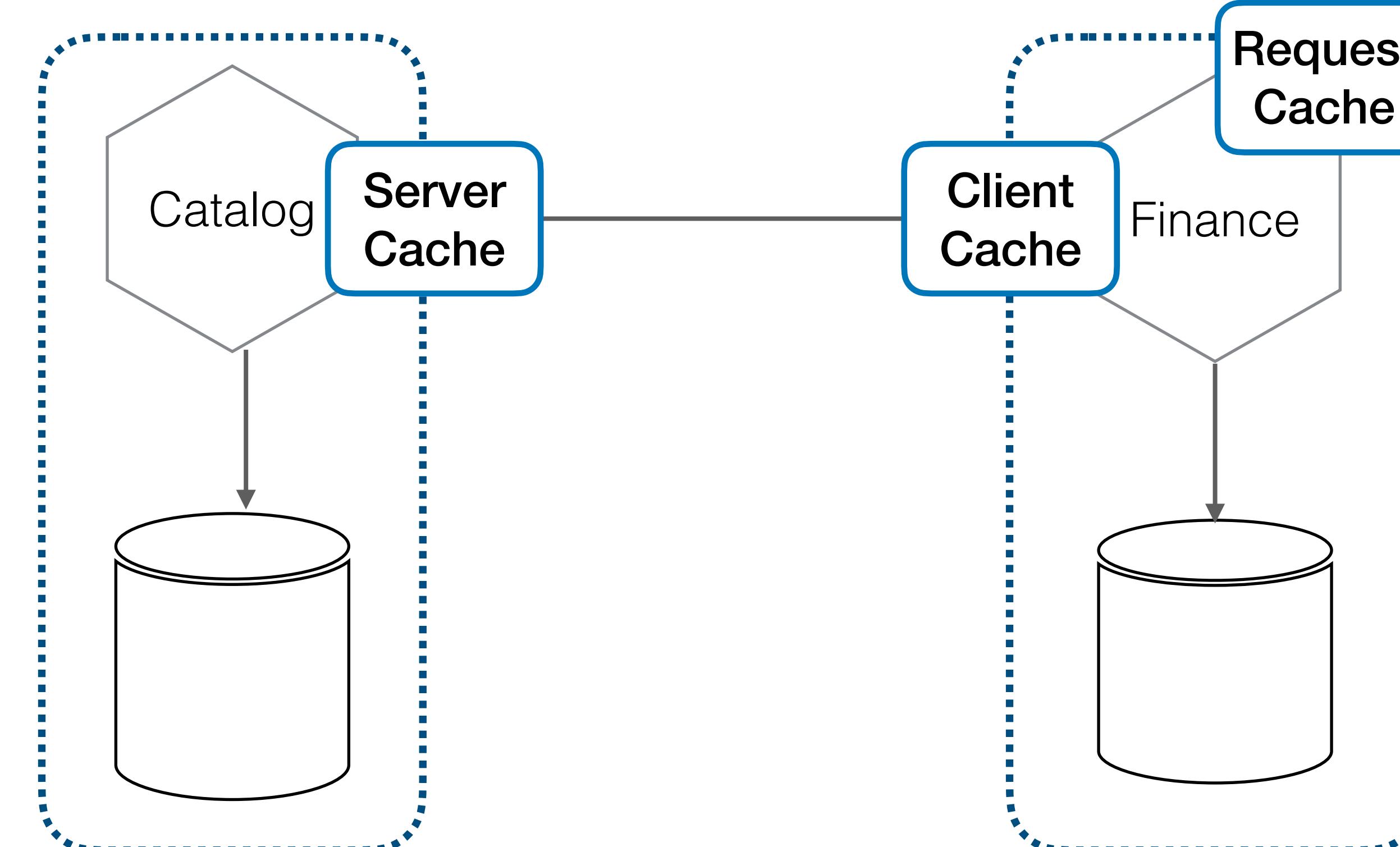
More clients = more memory, inconsistency

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

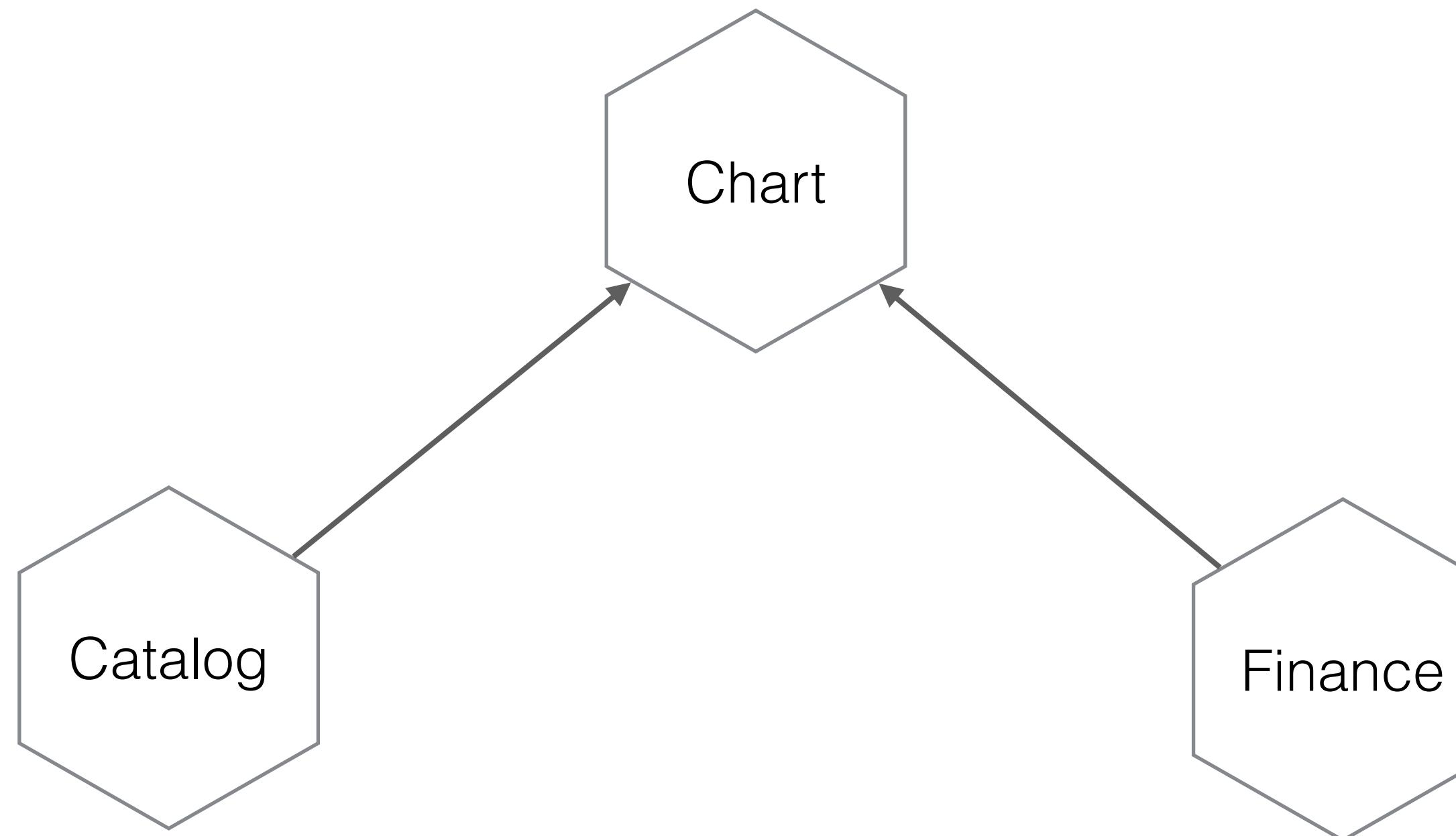
A DEDICATED SERVICE?

Catalog

Chart

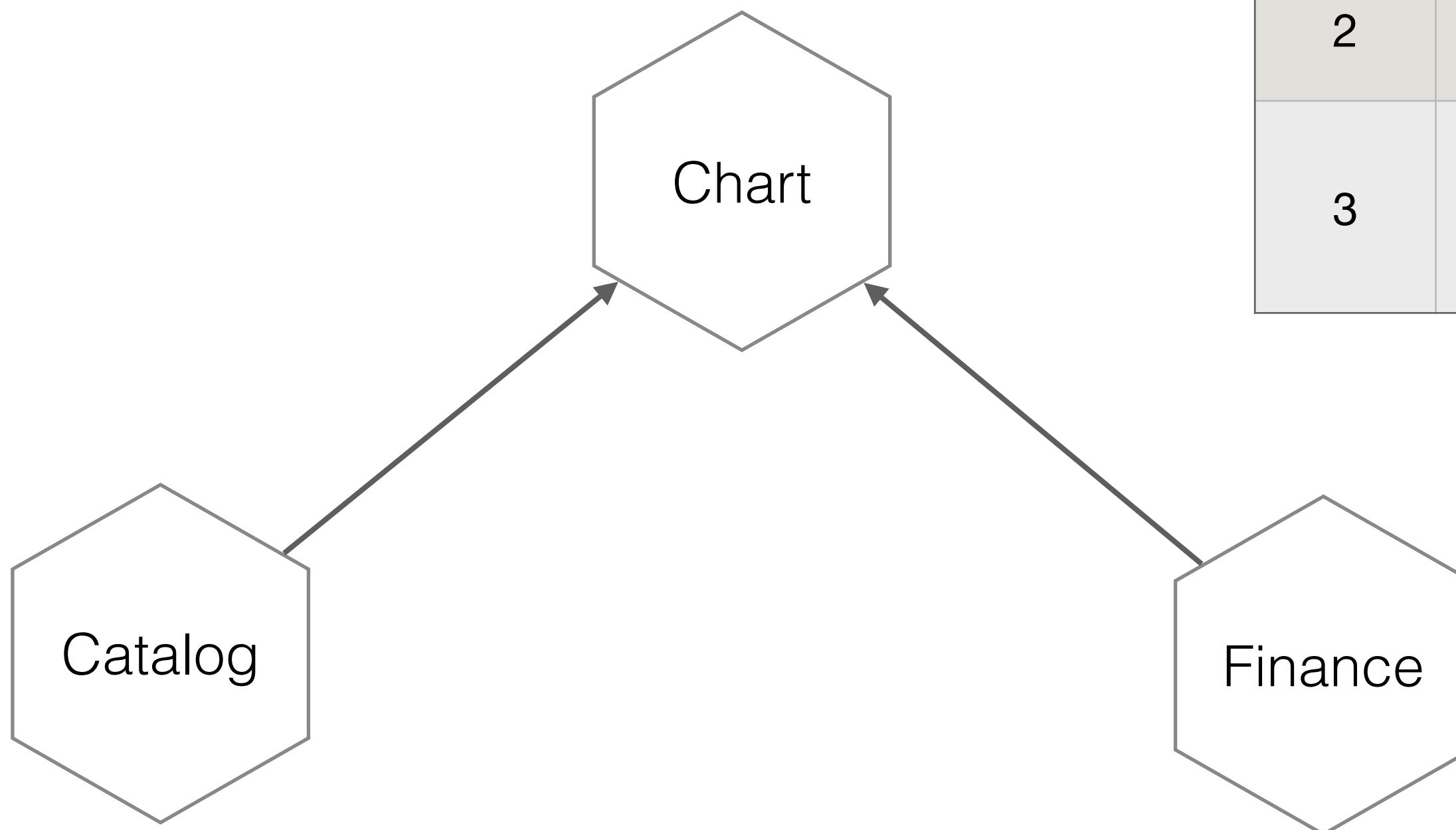
Finance

A DEDICATED SERVICE?



A DEDICATED SERVICE?

Best Sellers This Week!

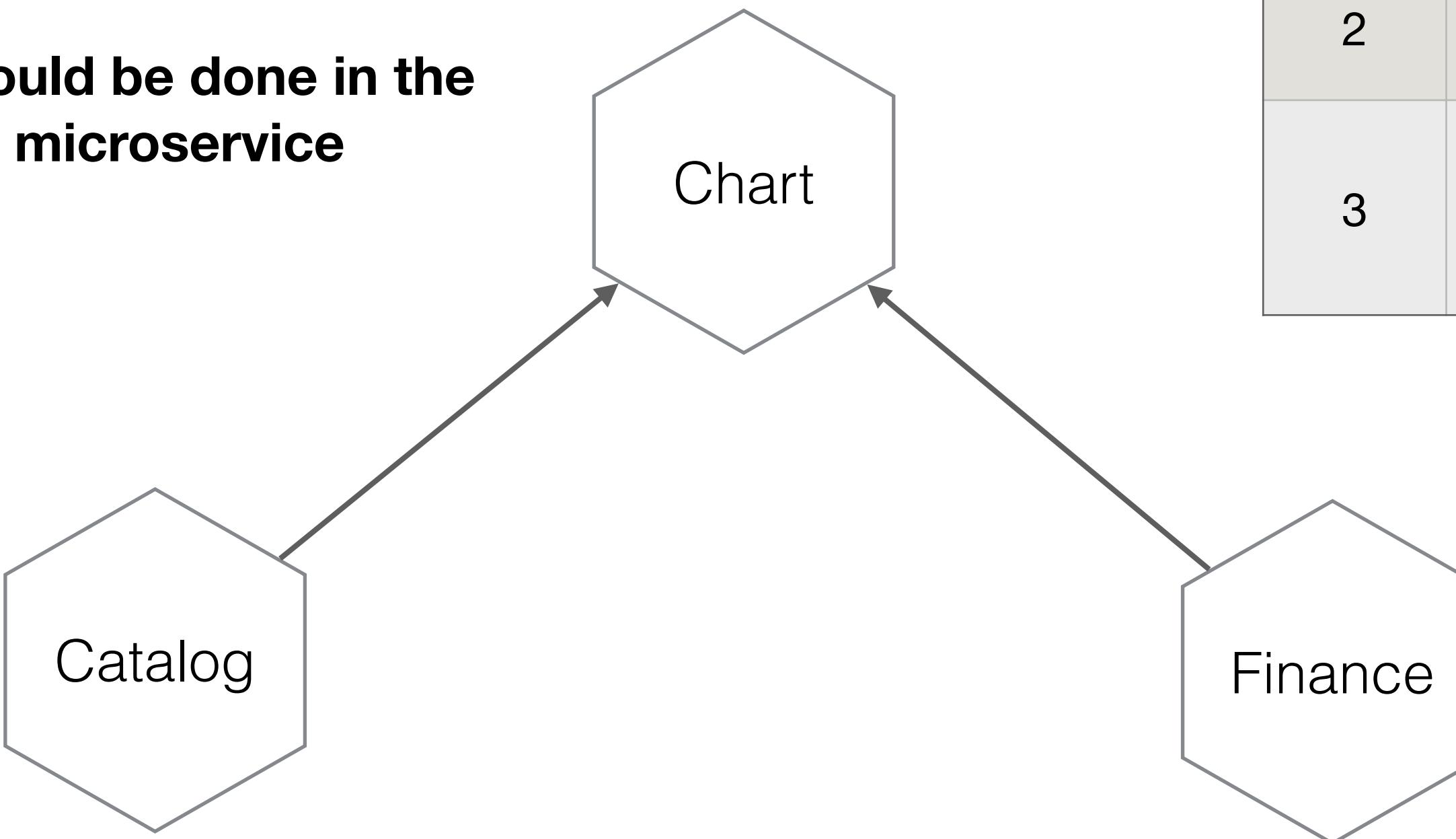


1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

A DEDICATED SERVICE?

Best Sellers This Week!

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

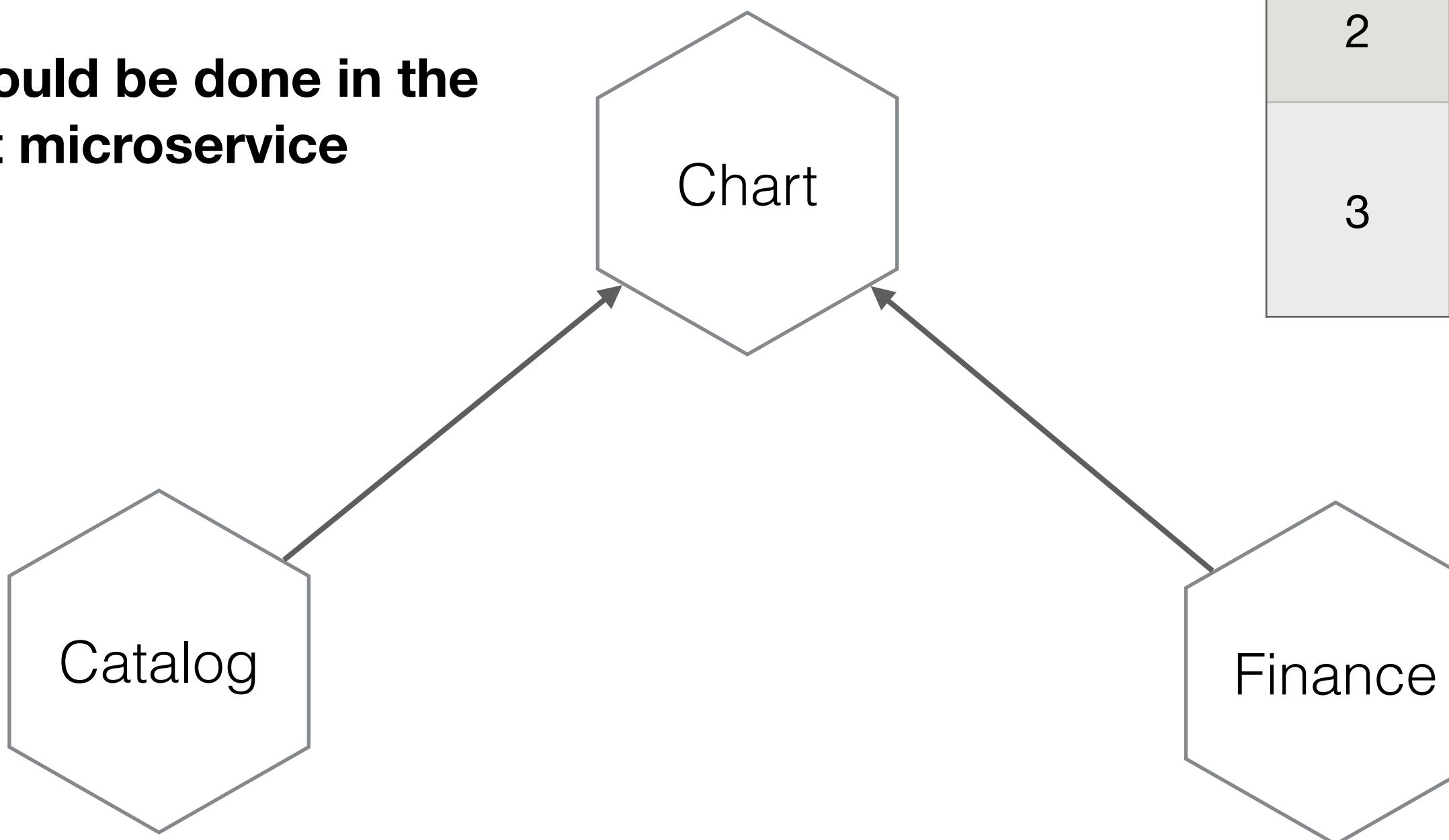


1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

A DEDICATED SERVICE?

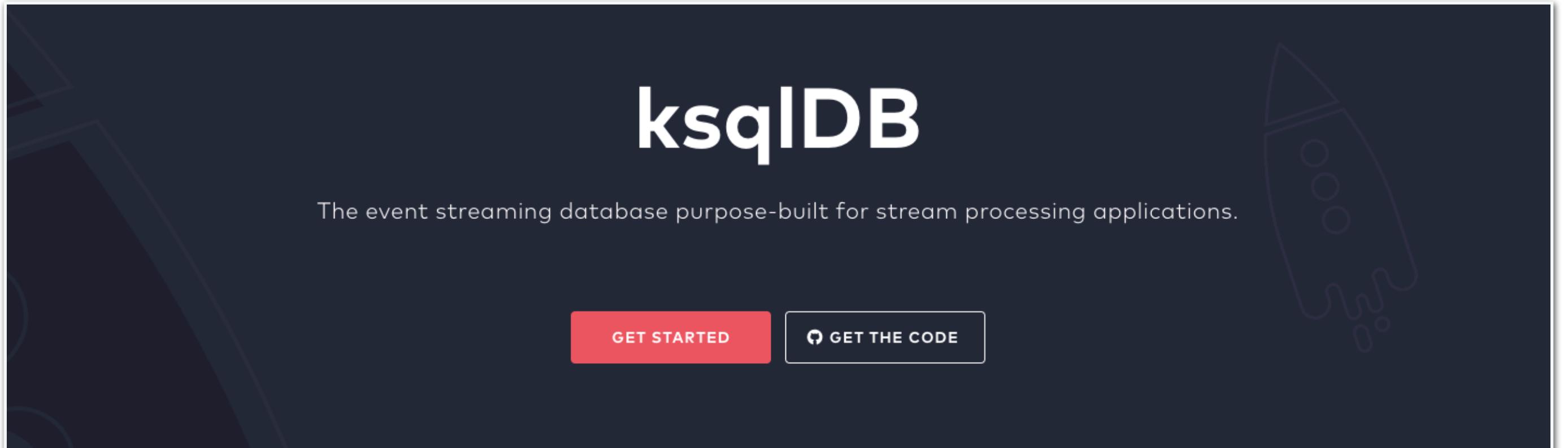
Best Sellers This Week!

**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.

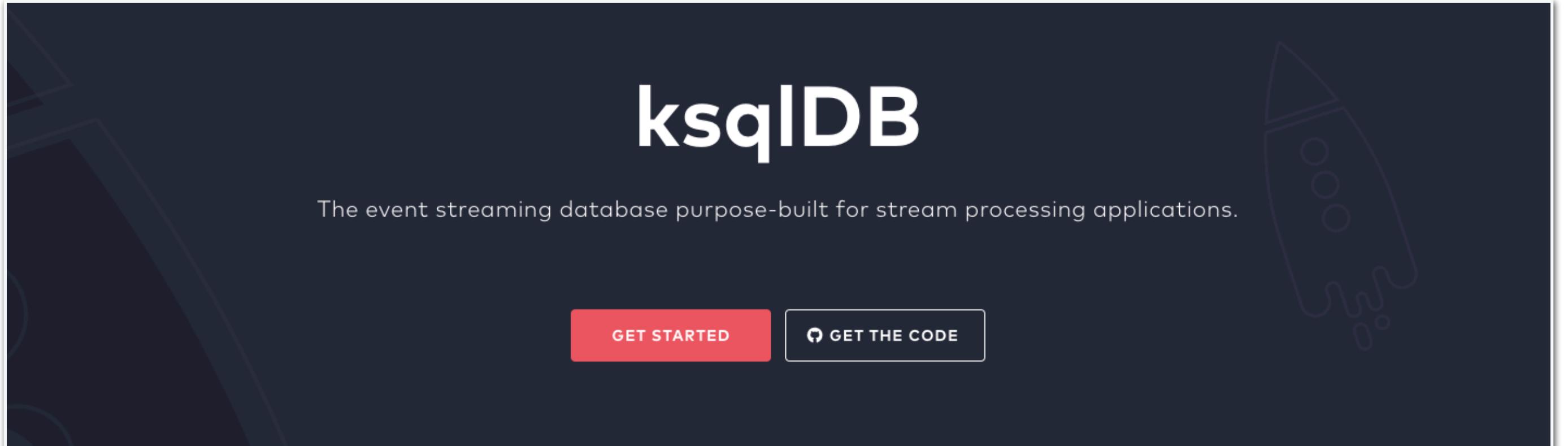
What, not how

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. 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 columns. The first column, titled "Real, real-time", describes building applications that respond immediately to events. The second column, titled "Kafka-native", describes leveraging existing Apache Kafka infrastructure. The third column, titled "What, not how", describes using a familiar SQL syntax. At the bottom, a section titled "Thousands of organizations love & trust ksqlDB" lists 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

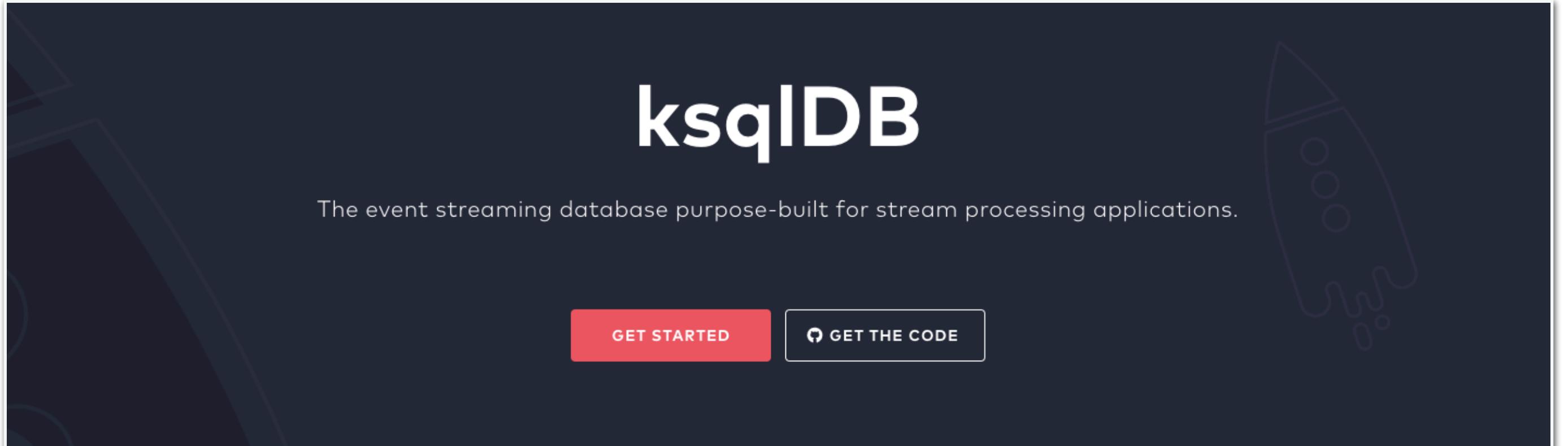
Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

Thousands of organizations love & trust ksqlDB

Apply SQL-style queries
to streams being
emitted from Kafka

<https://ksqldb.io/>



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 columns. The first column, titled "Real, real-time", describes building applications that respond immediately to events. The second column, titled "Kafka-native", discusses leveraging existing Apache Kafka infrastructure. The third column, titled "What, not how", explains using a familiar SQL syntax. At the bottom, a section titled "Thousands of organizations love & trust ksqlDB" lists 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

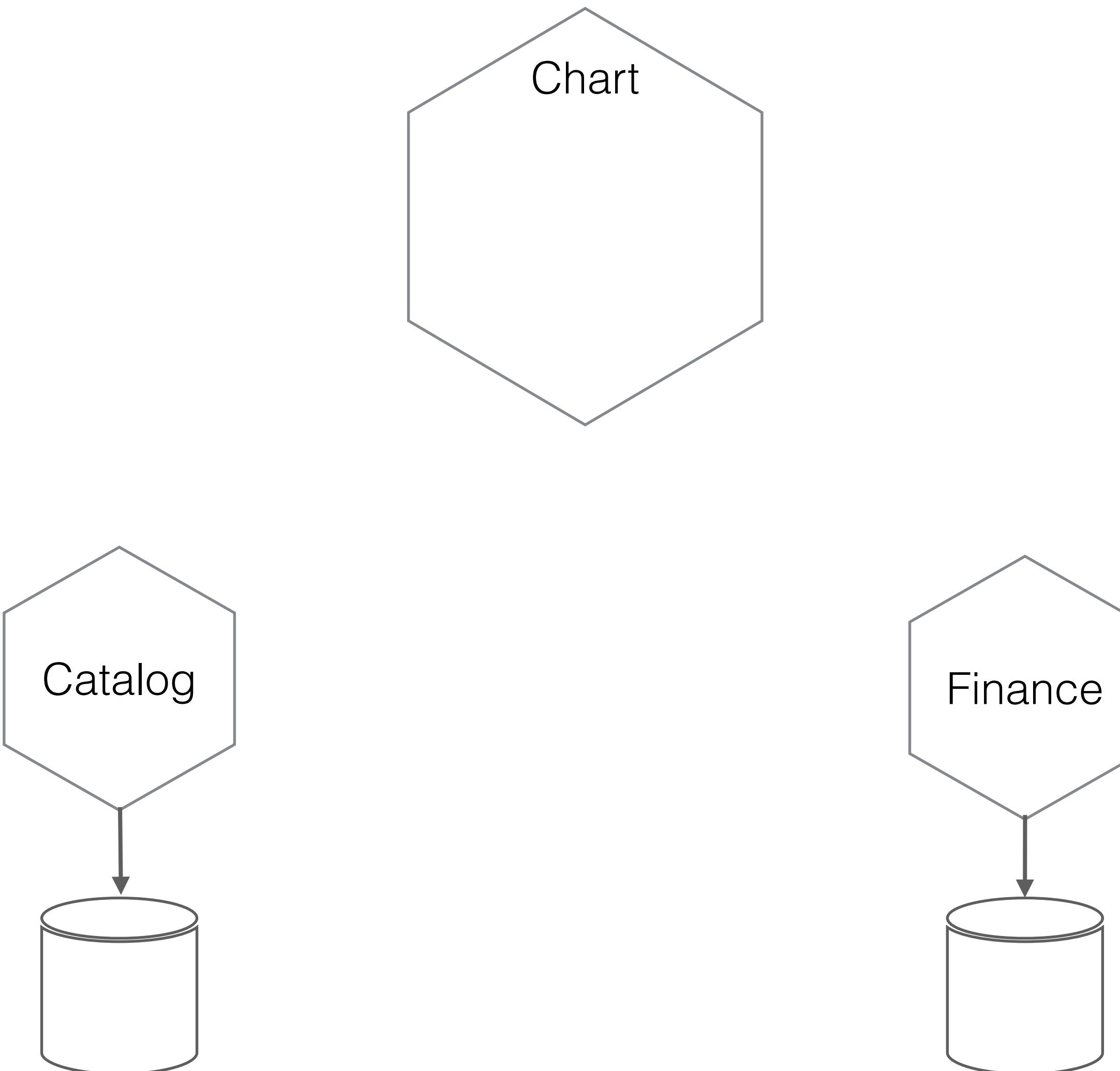
   

<https://ksqldb.io/>

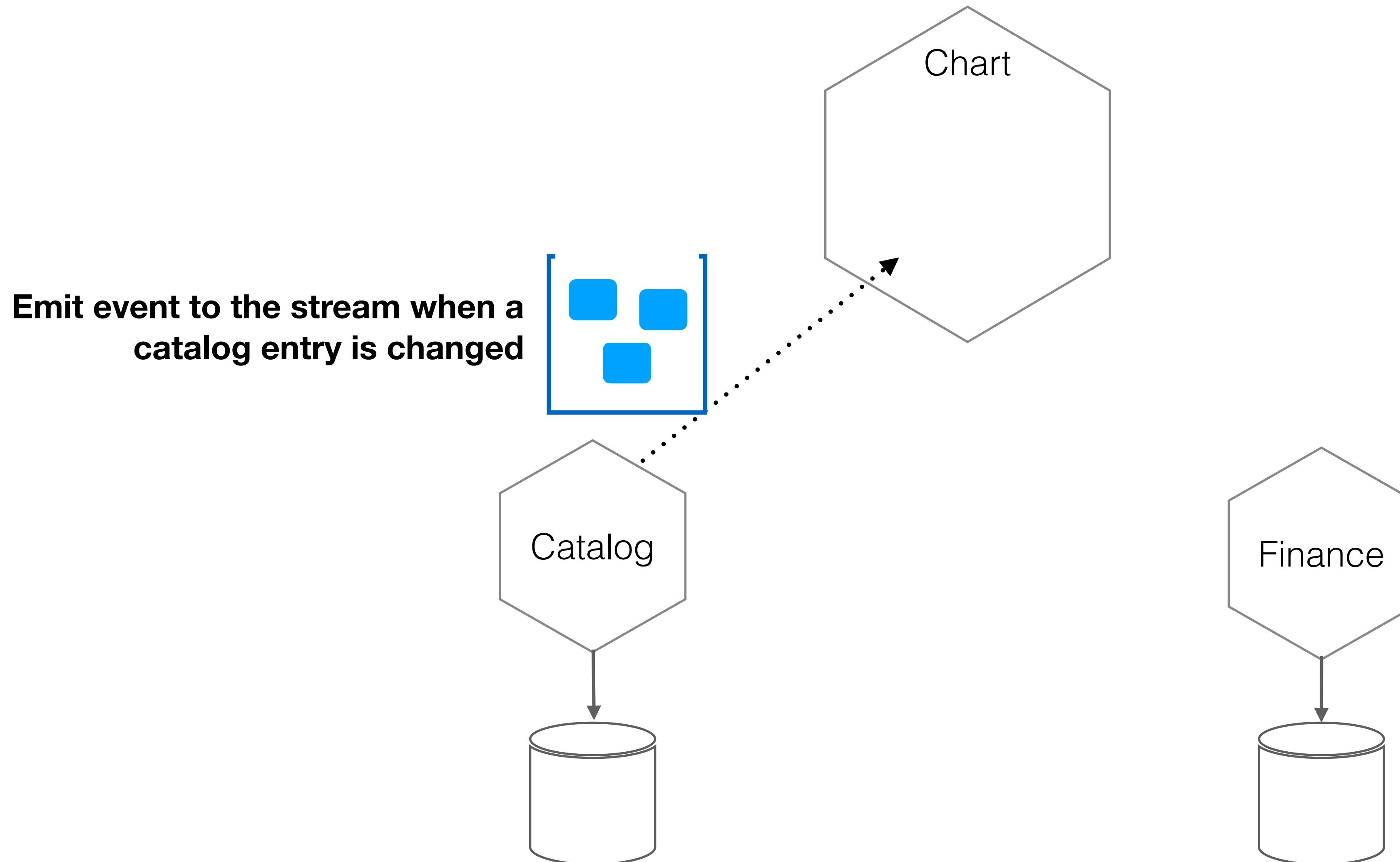
Apply SQL-style queries
to streams being
emitted from Kafka

Blends stream
processing with
standard SQL-use cases

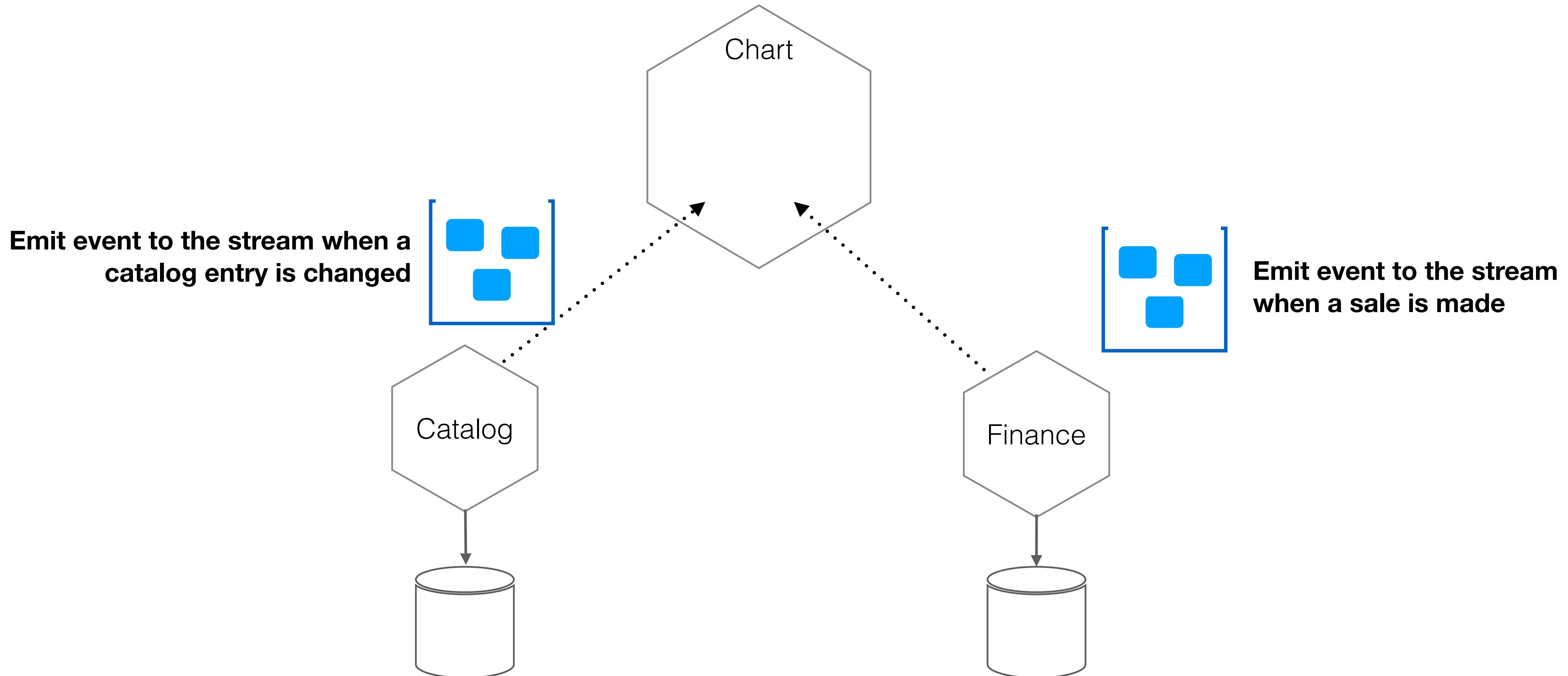
KSQLDB EXAMPLE ARCHITECTURE



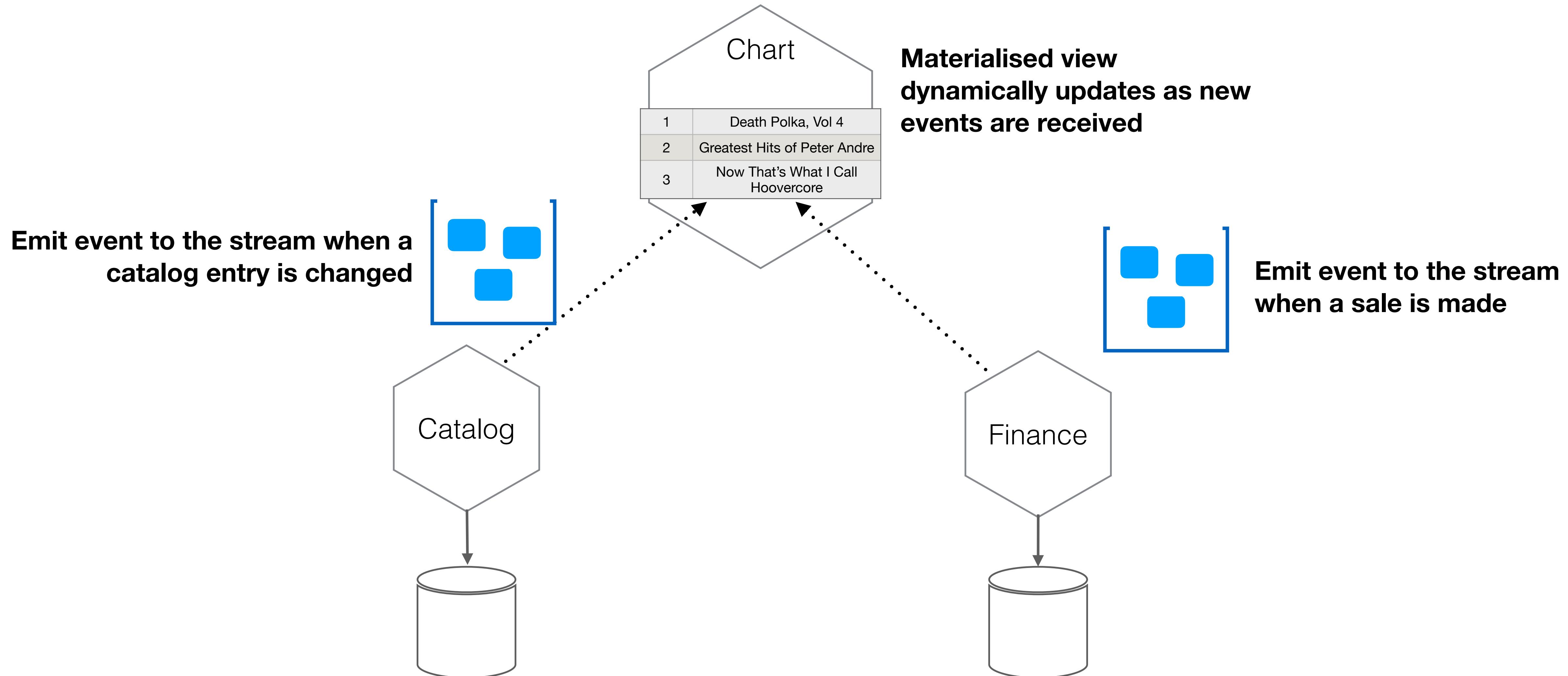
KSQLDB EXAMPLE ARCHITECTURE



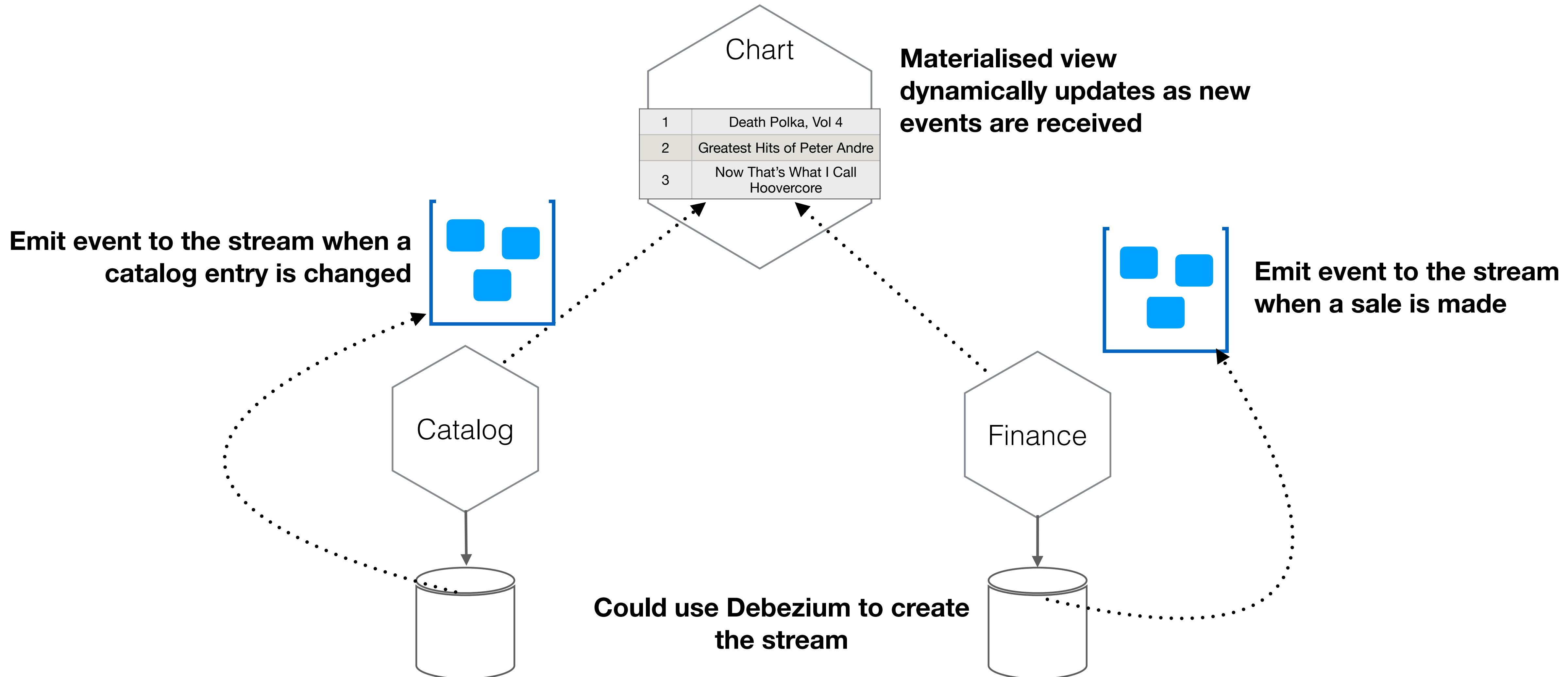
KSQLDB EXAMPLE ARCHITECTURE



KSQLDB EXAMPLE ARCHITECTURE



KSQLDB EXAMPLE ARCHITECTURE



THE CACHING PRIME DIRECTIVE

THE CACHING PRIME DIRECTIVE

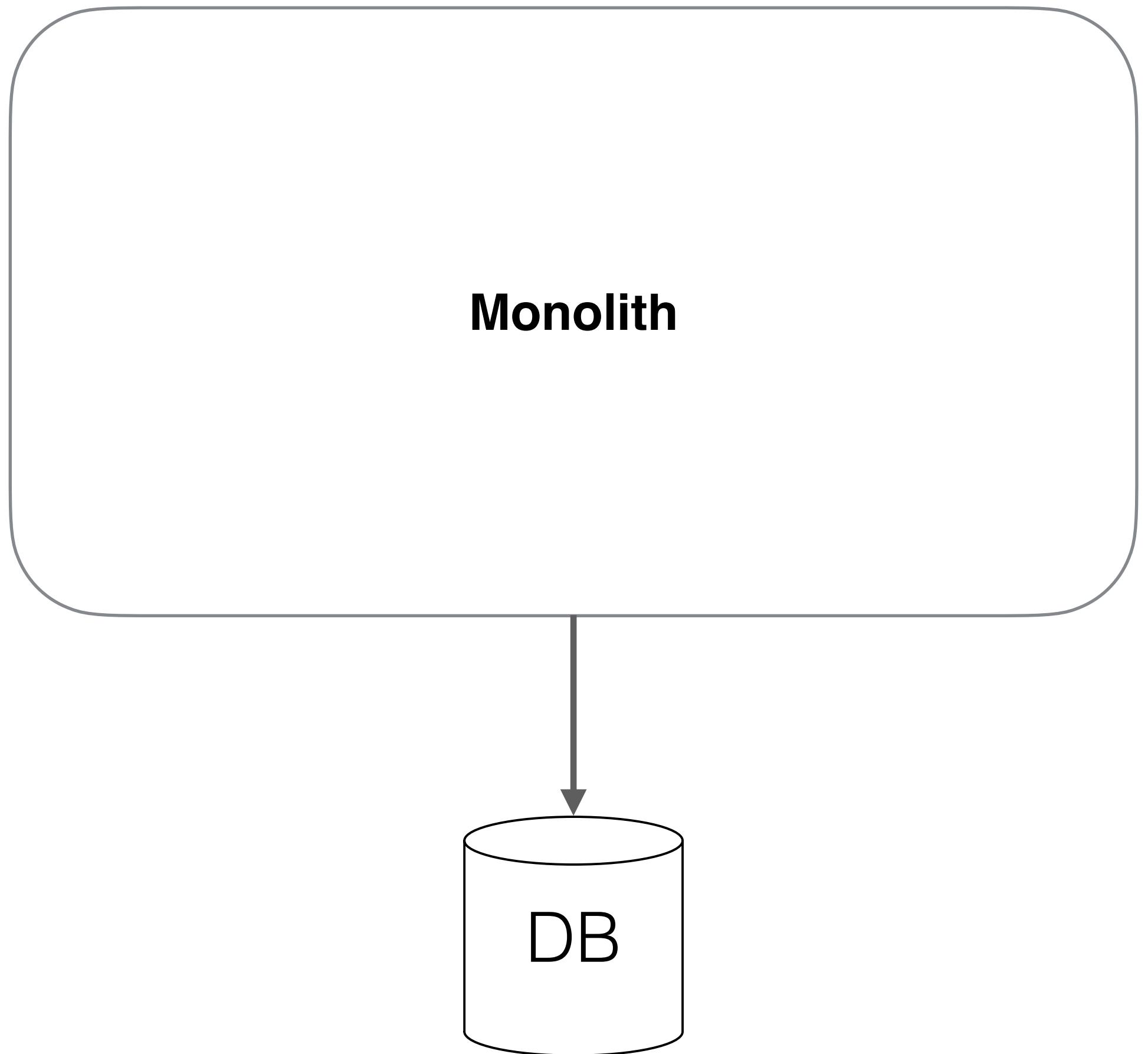
Cache in as few places as possible

THE CACHING PRIME DIRECTIVE

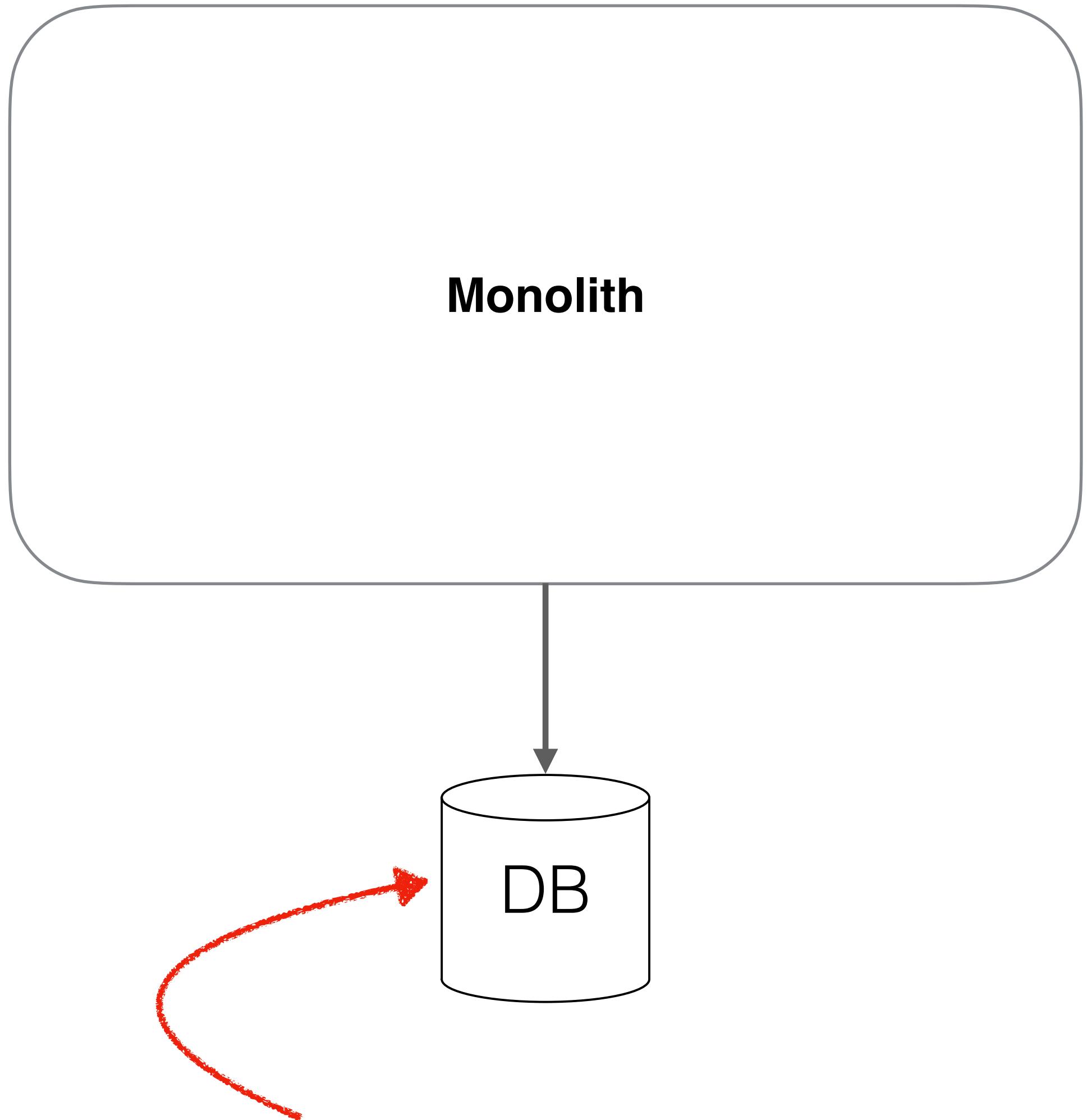
Cache in as few places as possible

Zero is the best number of caches

SINGLE SOURCE OF TRUTH?

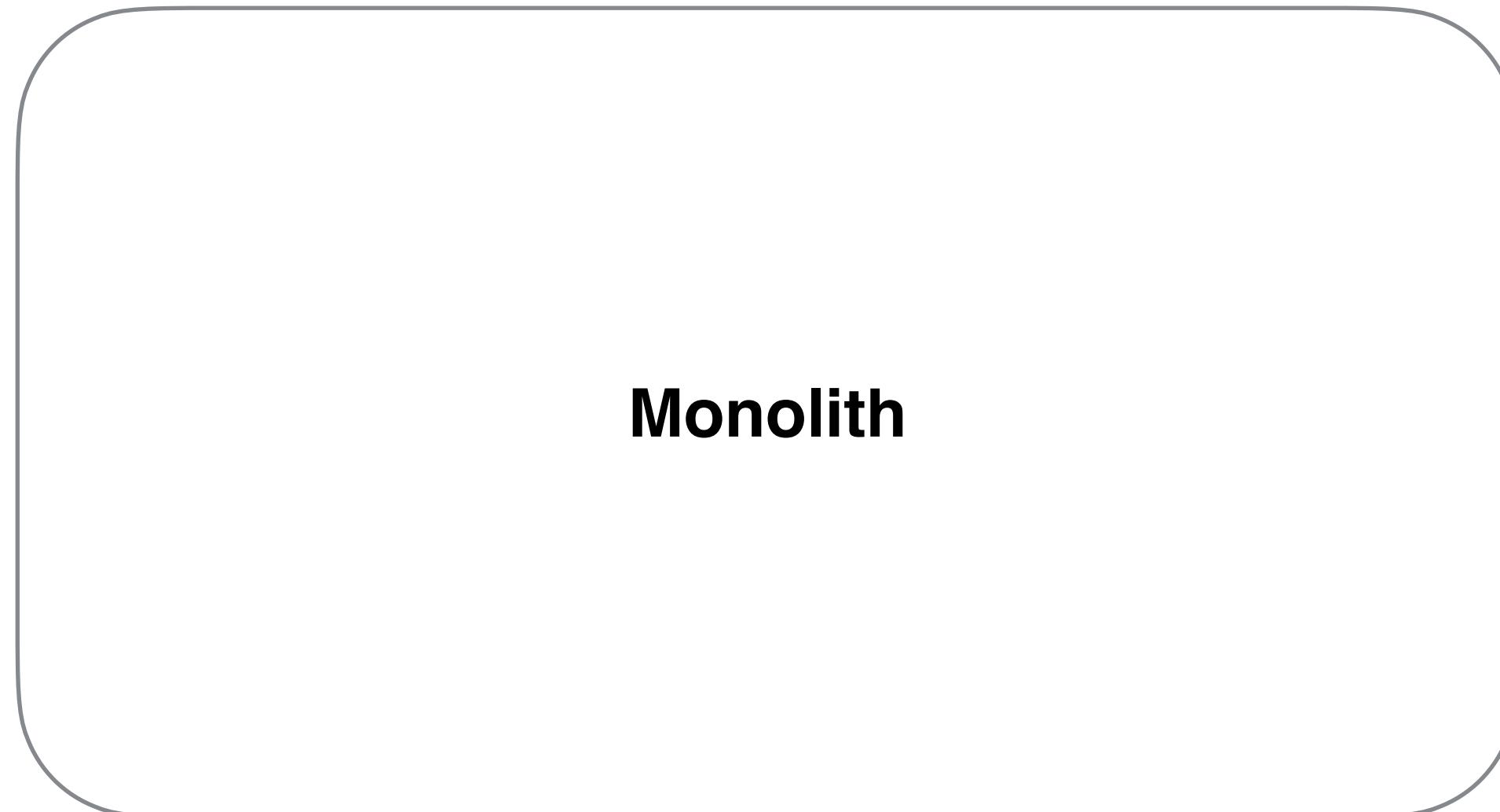


SINGLE SOURCE OF TRUTH?



This is the source of truth for invoicing data!

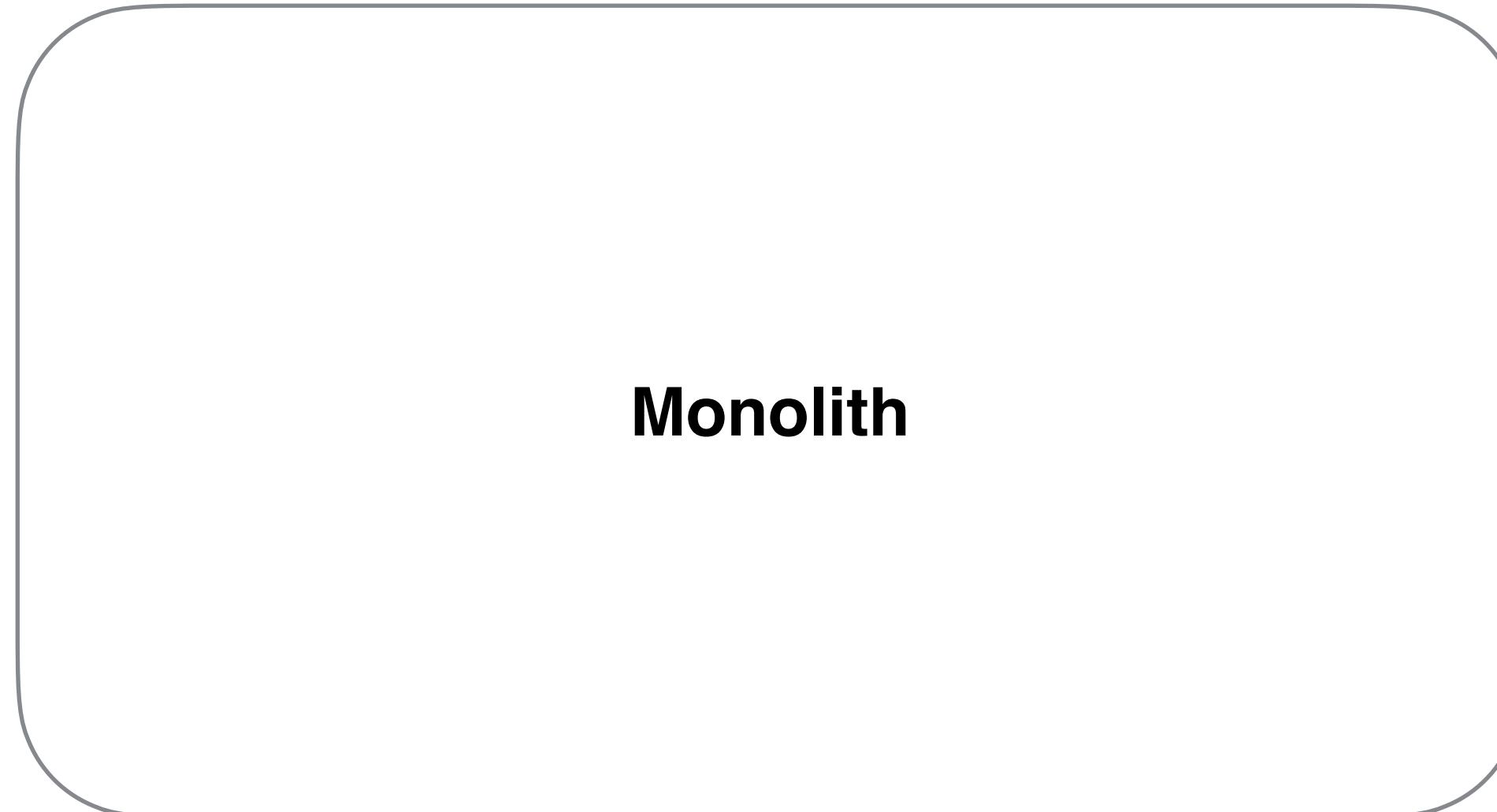
SINGLE SOURCE OF TRUTH?



Helps manage consistency of data

This is the source of truth for invoicing data!

SINGLE SOURCE OF TRUTH?



Monolith

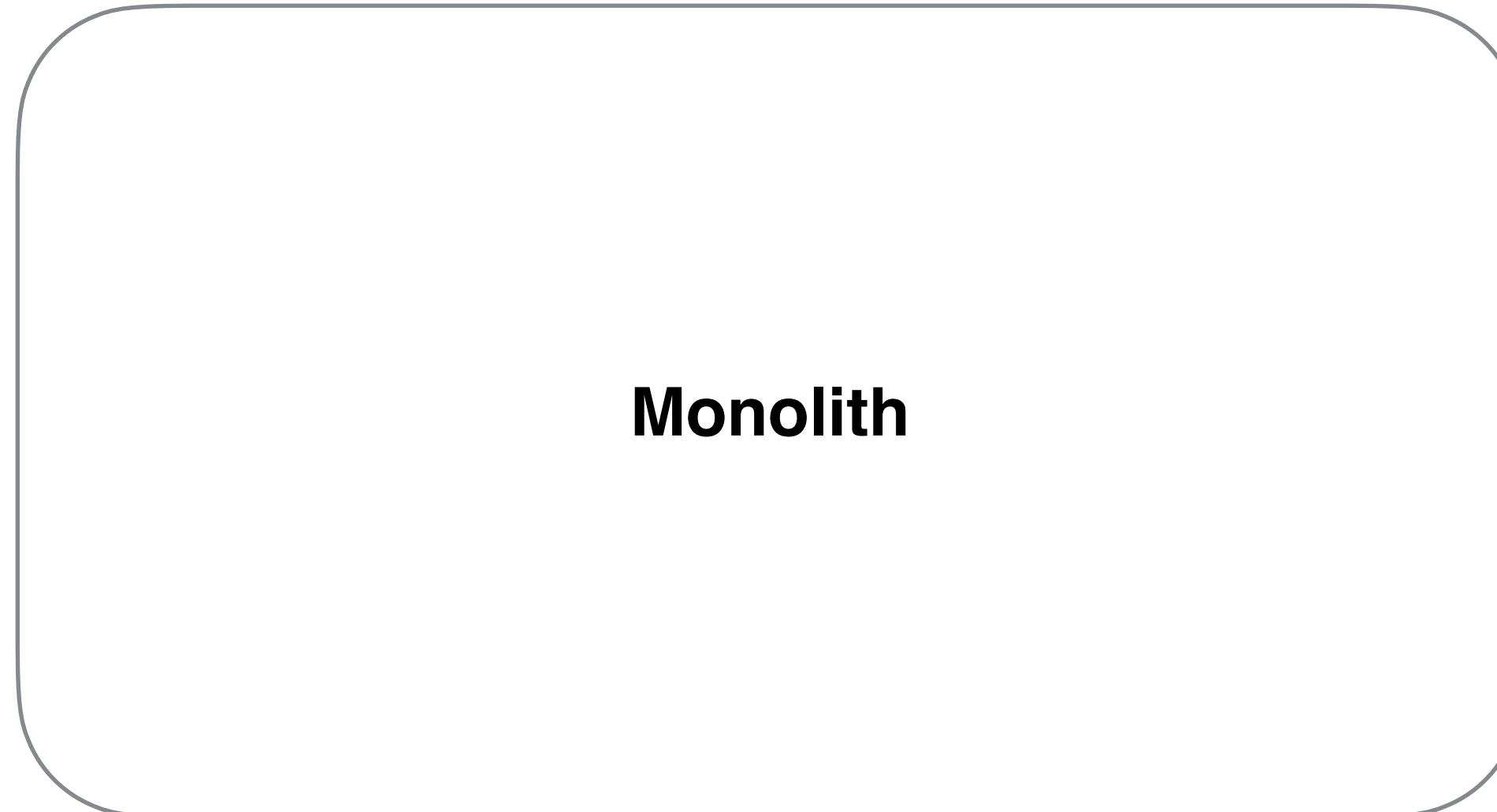
DB

This is the source of truth for invoicing data!

Helps manage consistency of data

Easier to control access

SINGLE SOURCE OF TRUTH?



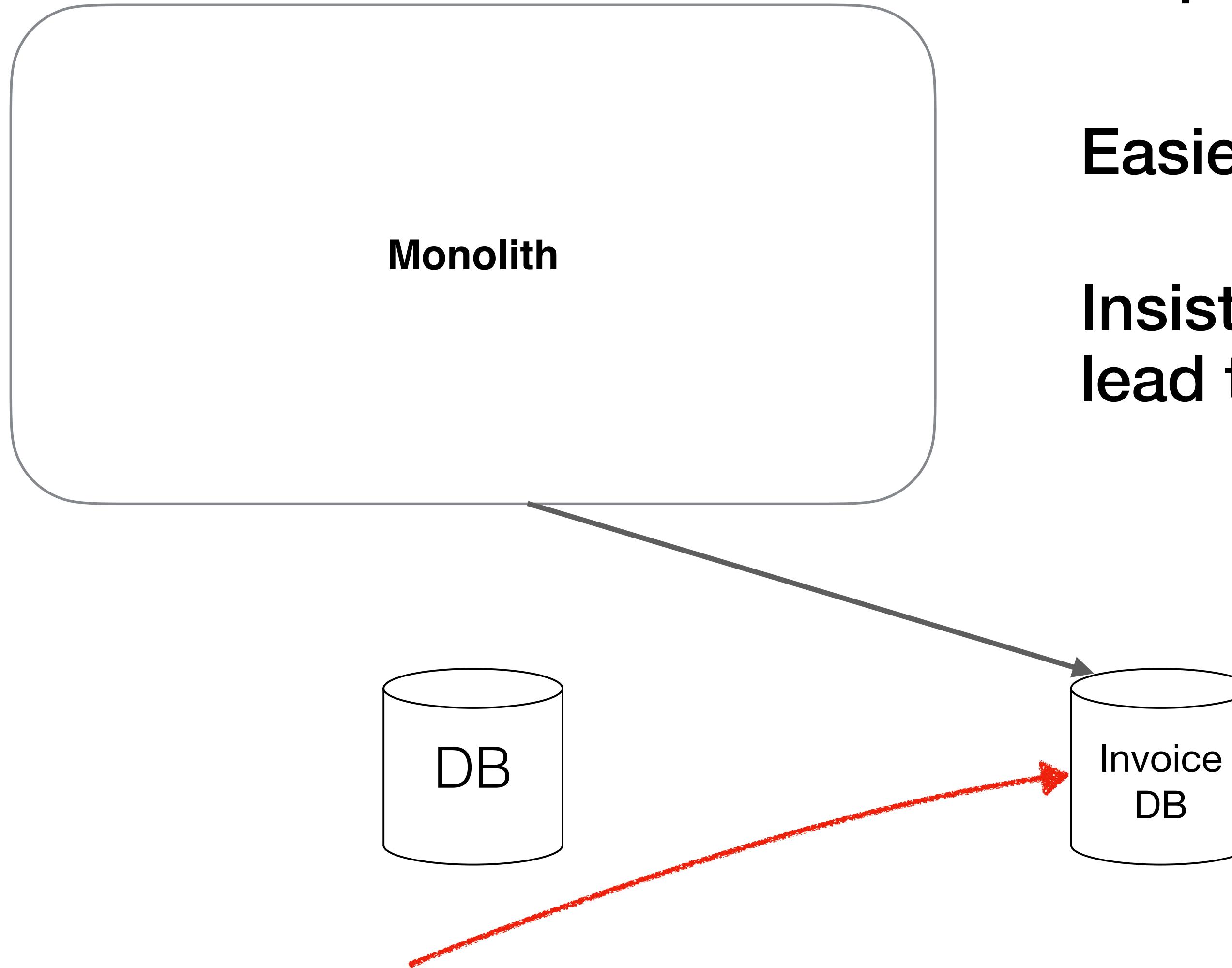
This is the source of truth for invoicing data!

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?



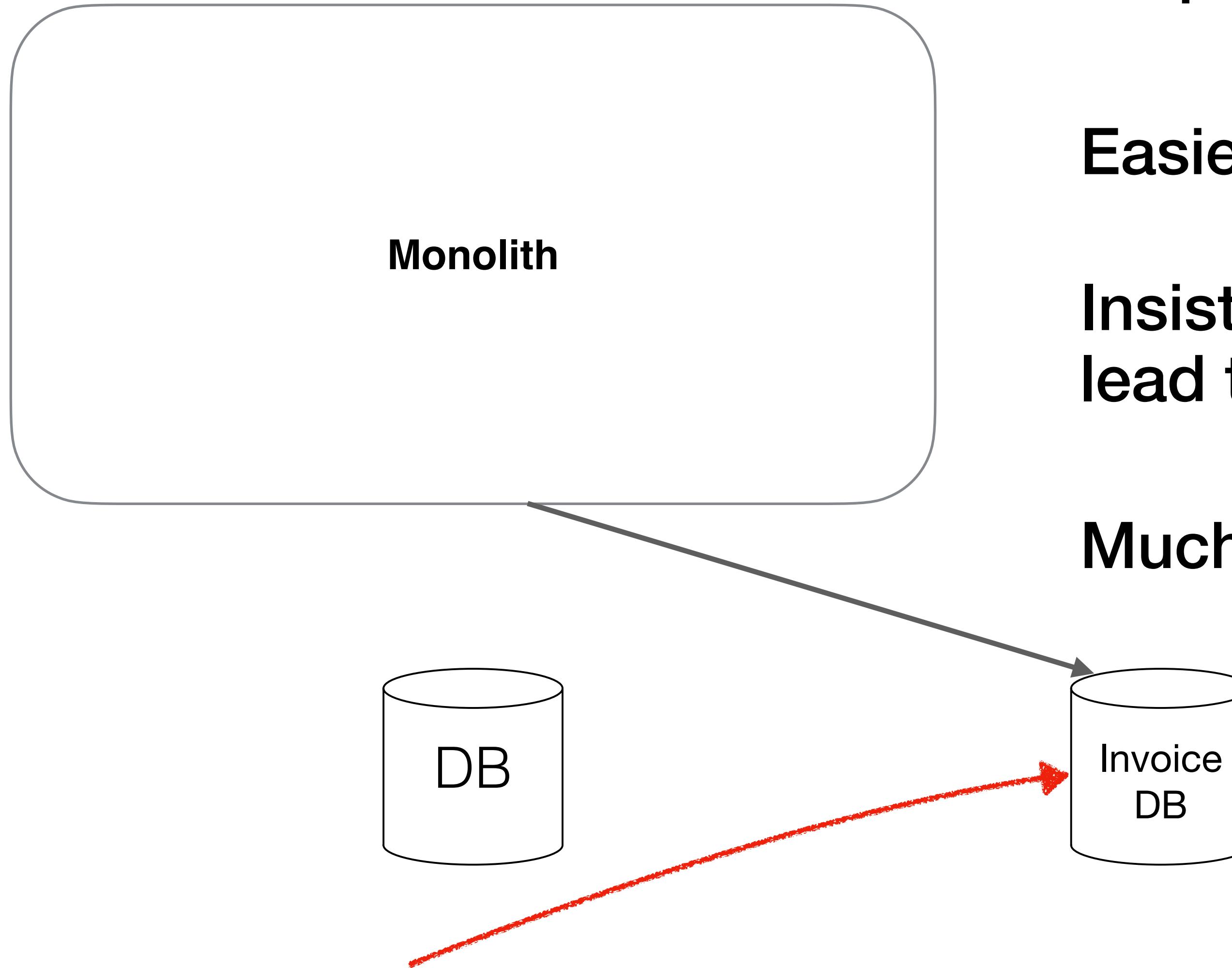
Helps manage consistency of data

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

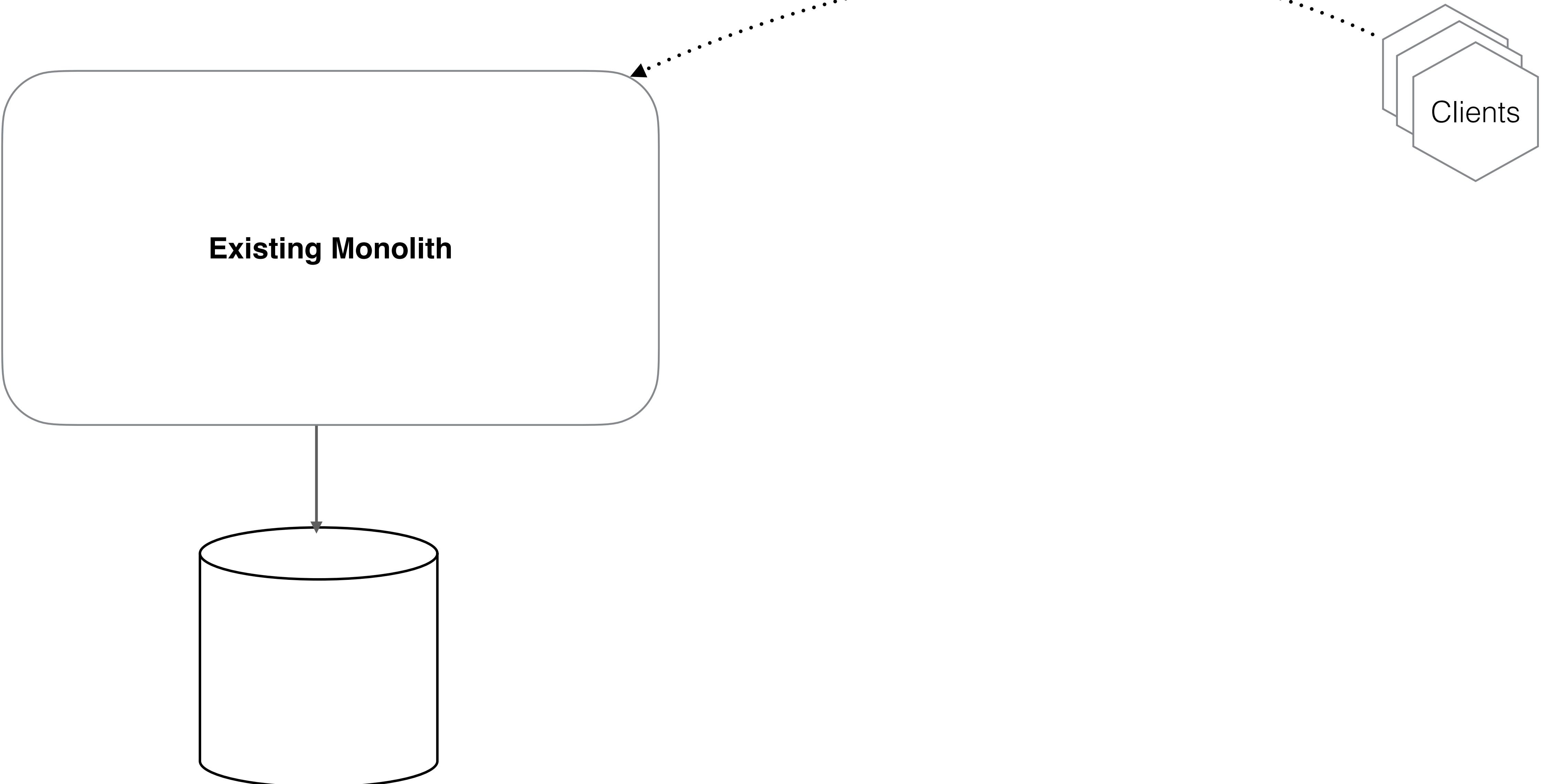
10



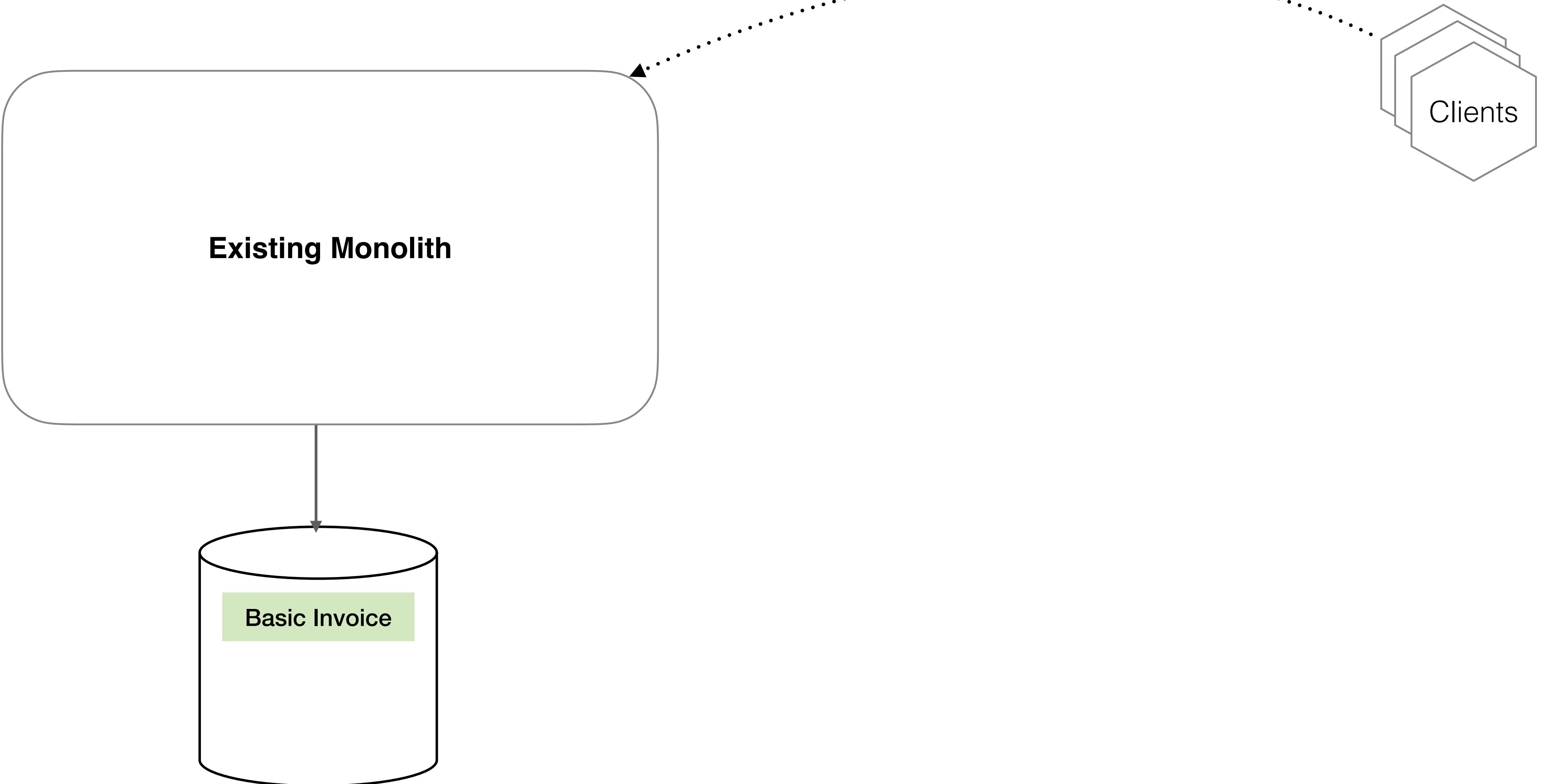


Square

PATTERN: TRACER WRITE



PATTERN: TRACER WRITE



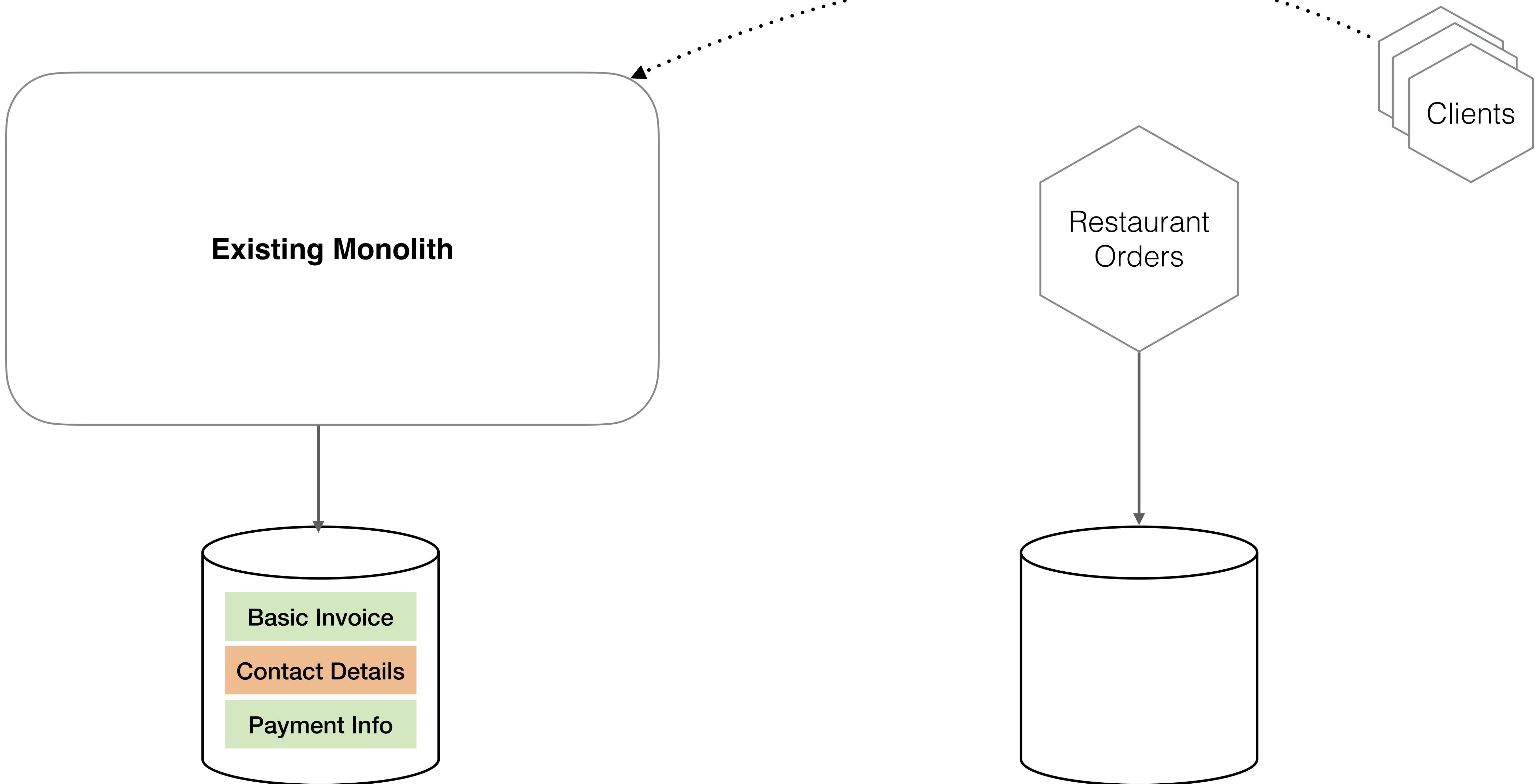
PATTERN: TRACER WRITE



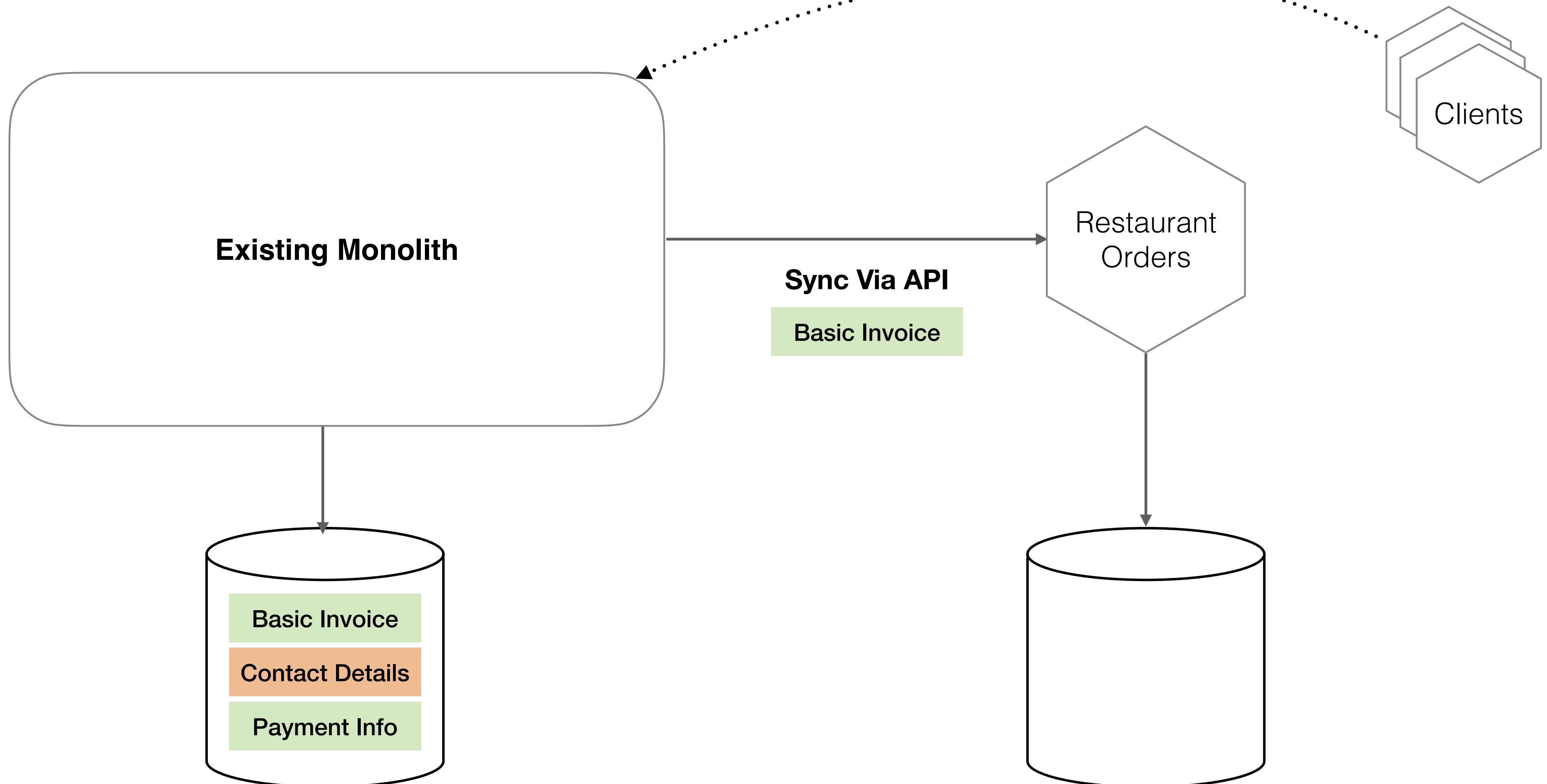
PATTERN: TRACER WRITE



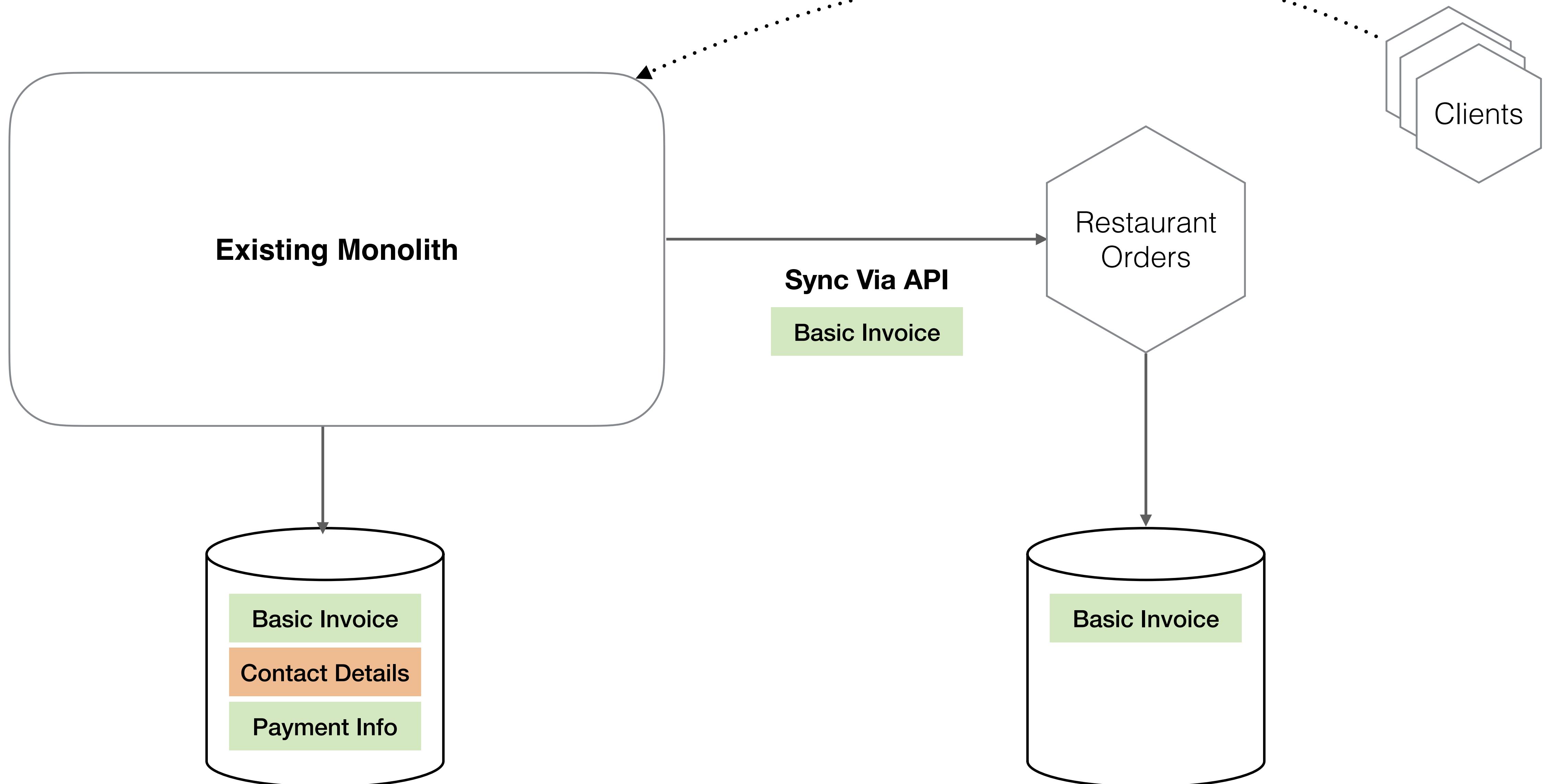
PATTERN: TRACER WRITE



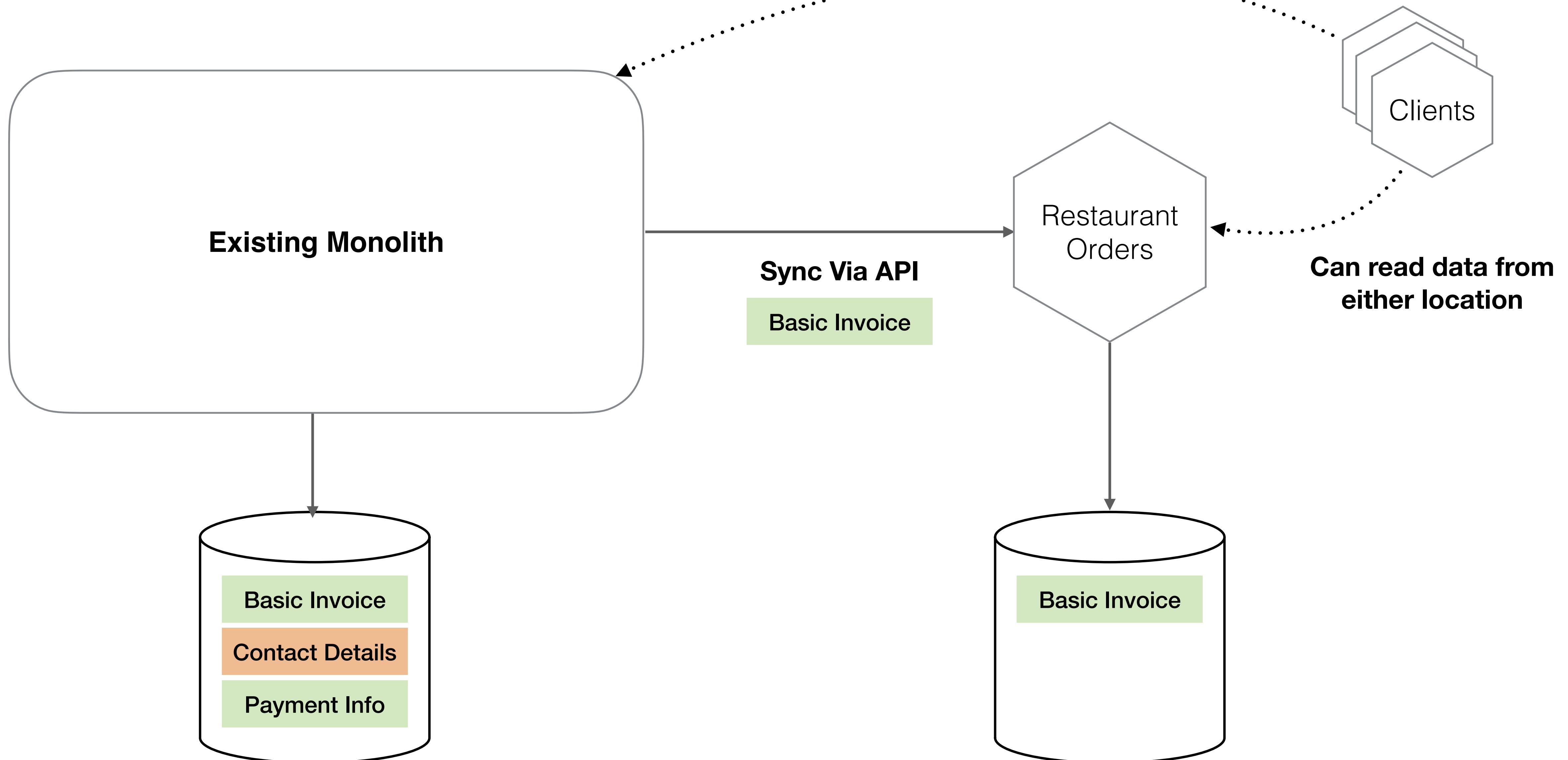
PATTERN: TRACER WRITE



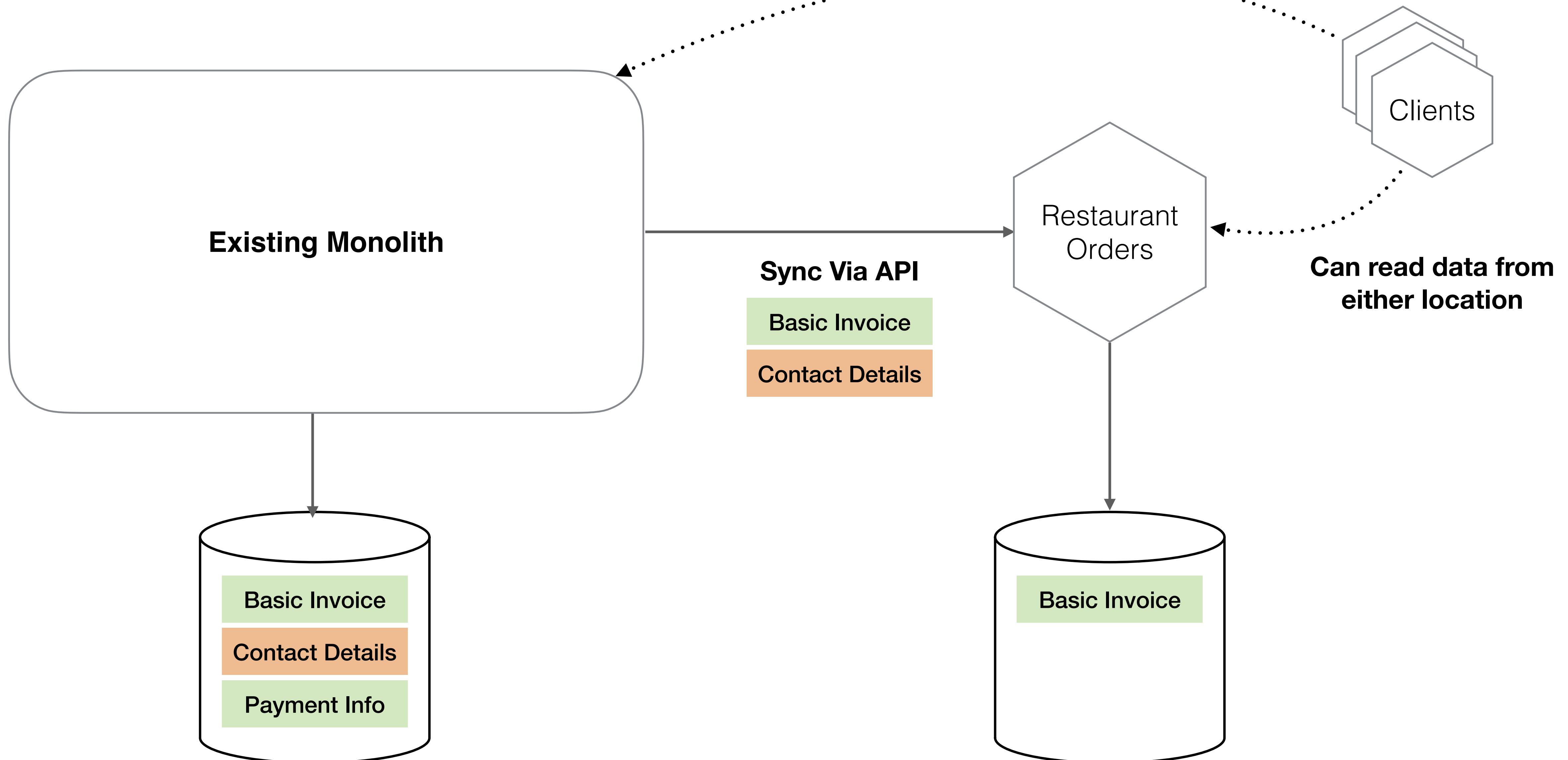
PATTERN: TRACER WRITE



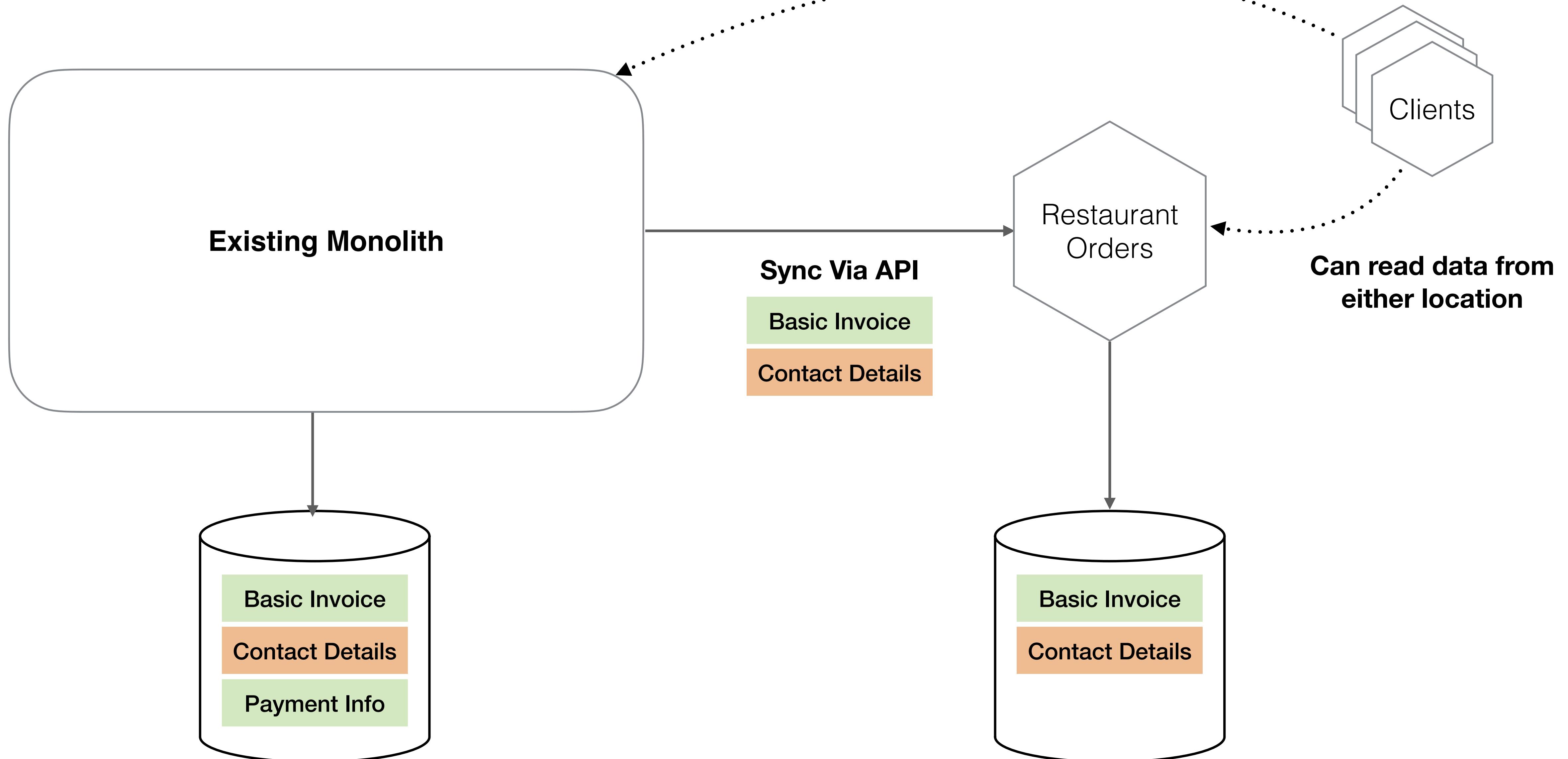
PATTERN: TRACER WRITE



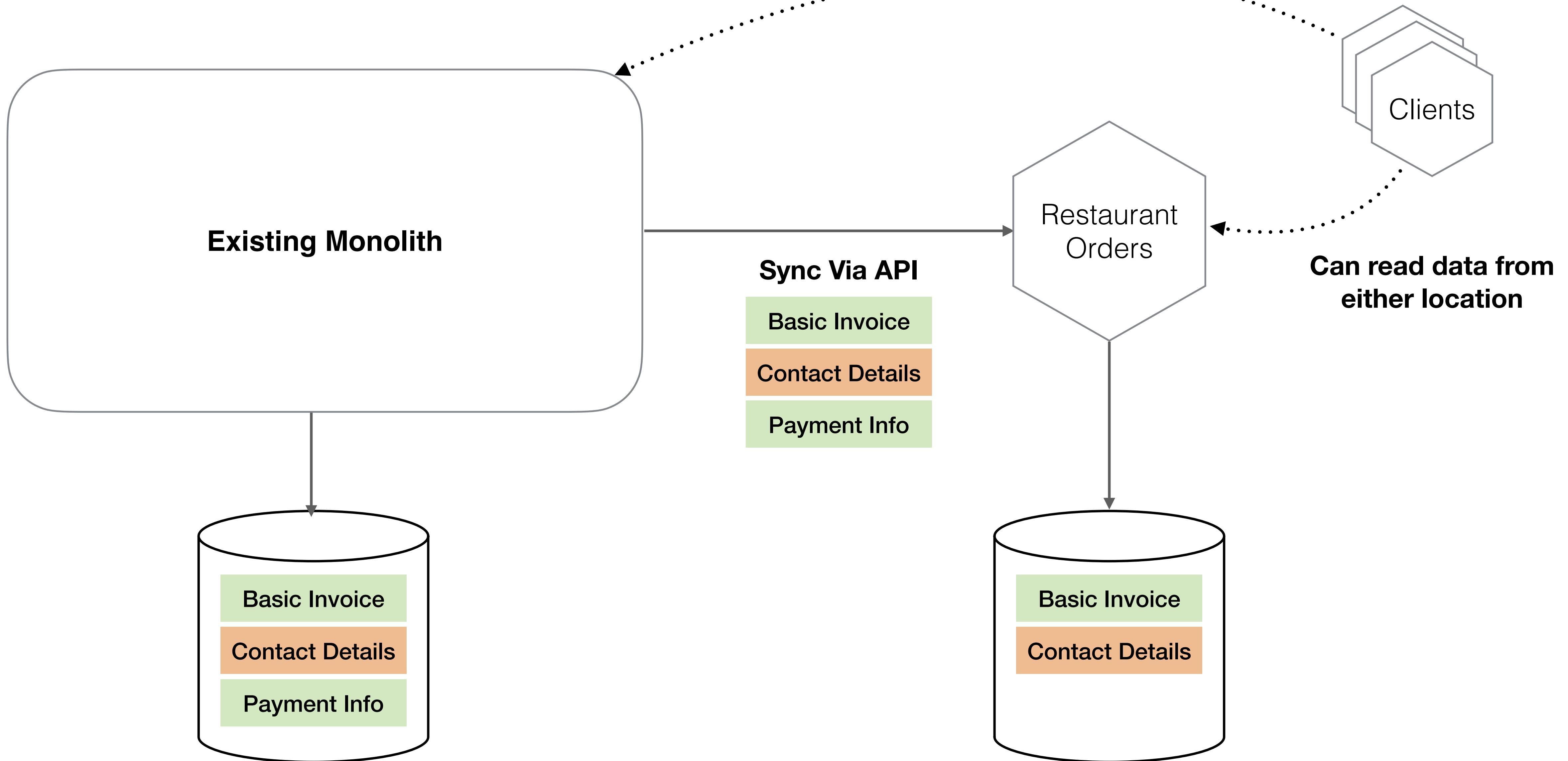
PATTERN: TRACER WRITE



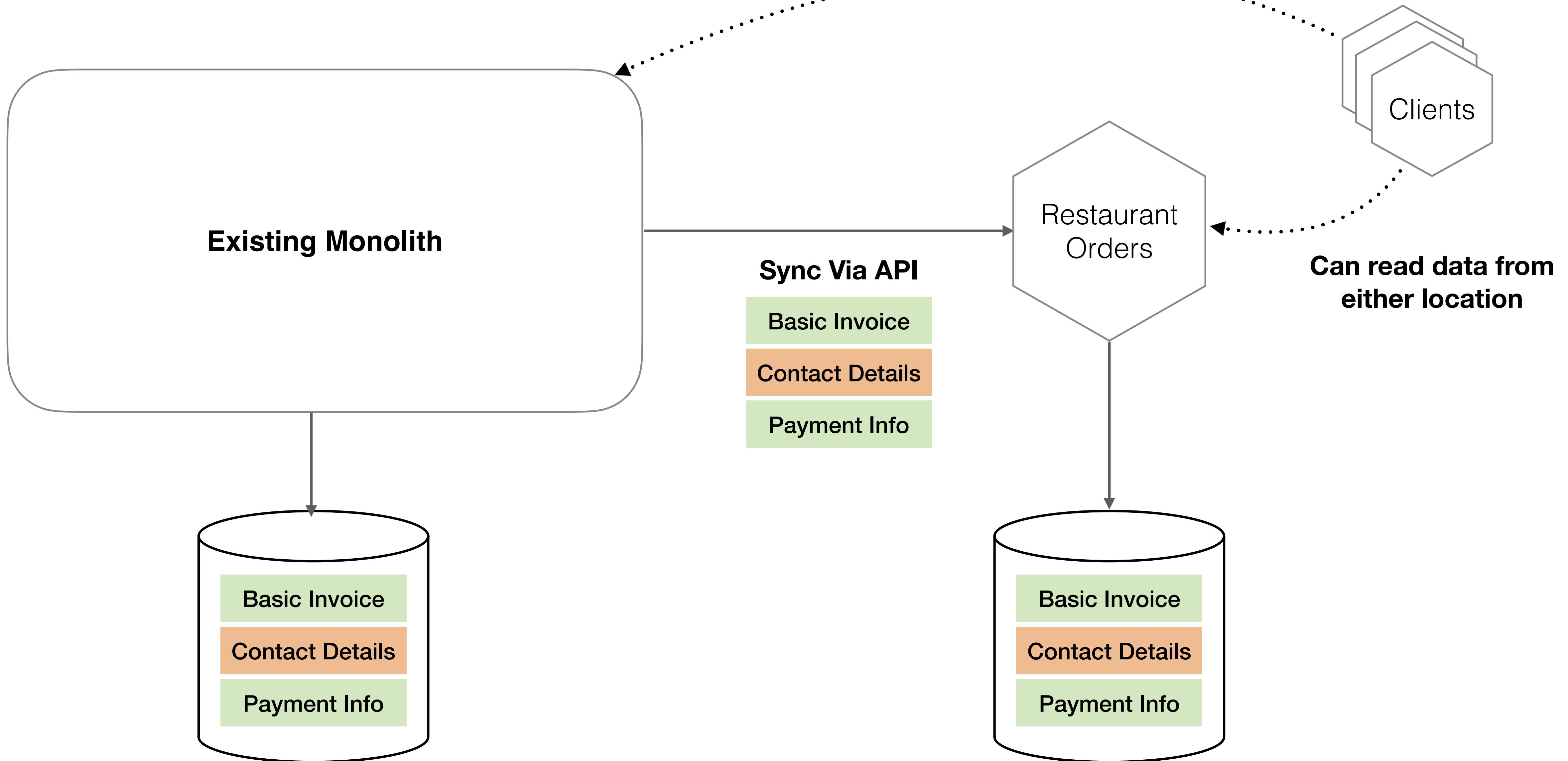
PATTERN: TRACER WRITE



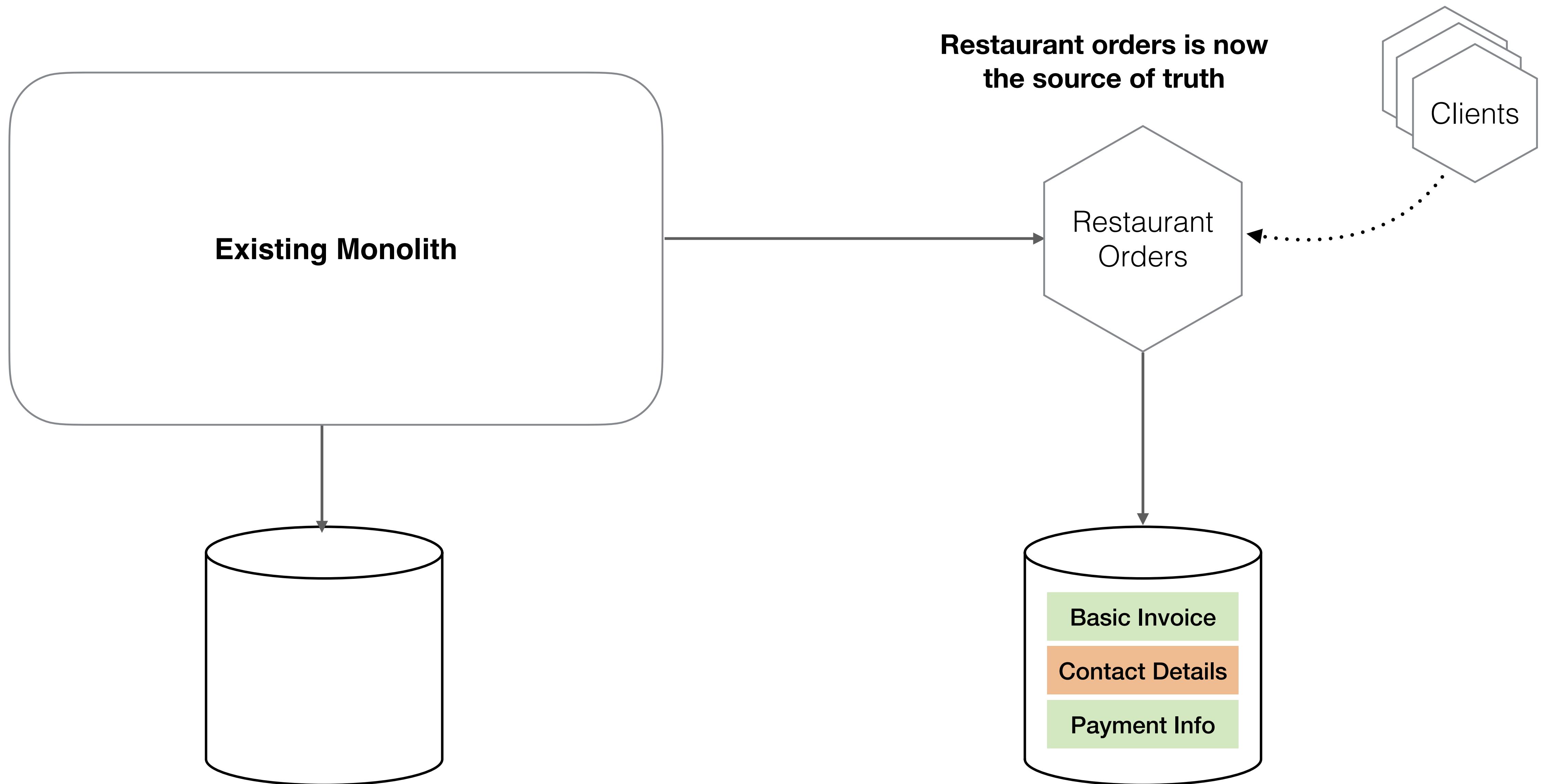
PATTERN: TRACER WRITE



PATTERN: TRACER WRITE



PATTERN: TRACER WRITE



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**

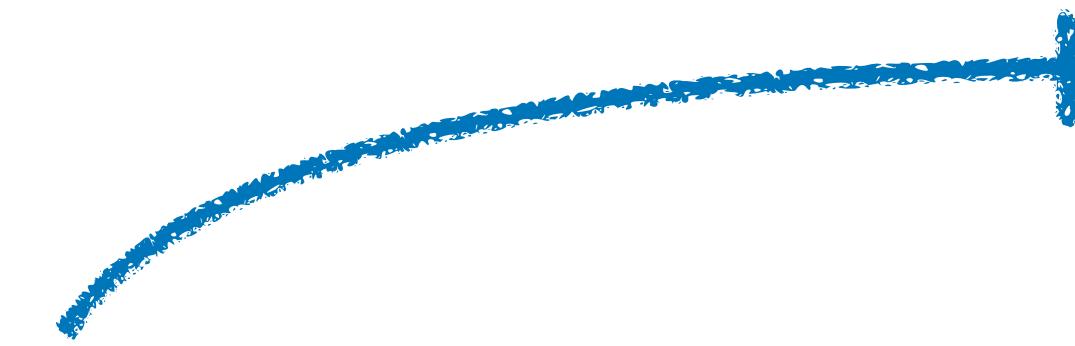
Break big changes into lots of small changes

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

OTHER COURSES



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



THANKS!

Sam Newman & Associates



Home About Offerings Events **Writing** Contact

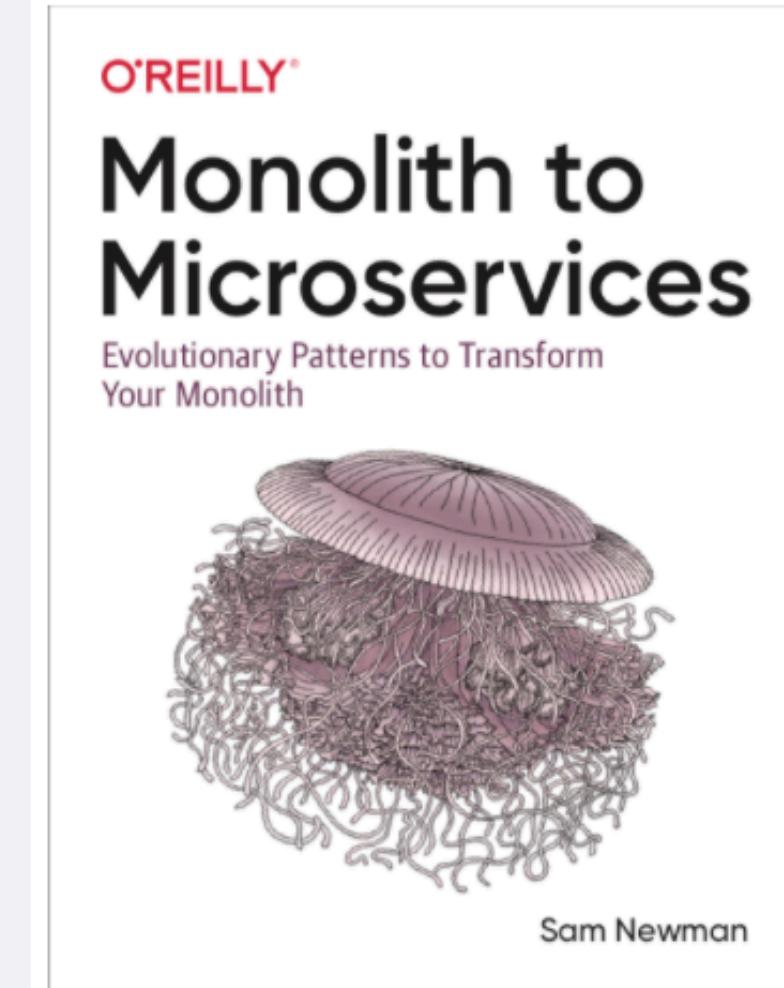
Monolith To Microservices.

Released on 2019-09-04 by O'Reilly

Monolith To Microservices is a new book on system decomposition from O'Reilly

If you're interested in migrating your existing systems to microservice architectures, or are struggling with services that are too big, then this book is for you

→ [Find Out More](#)



<https://samnewman.io/>

@samnewman