

MONOLITHS TO MICROSERVICES

Sam Newman

Splitting Out Services

Overview

Where to start

Modelling services

Incremental change

DB Integration & Refactoring





Where to start?

What makes a good service?

High Cohesion

Loose Coupling

Musik Web

Musik Web

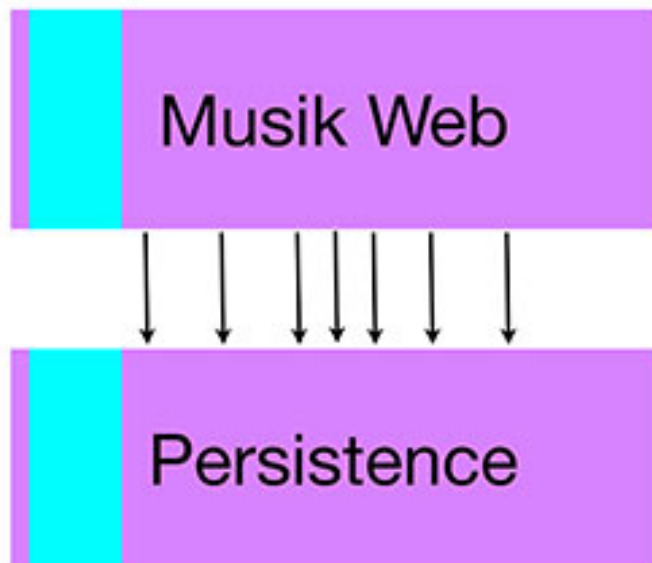


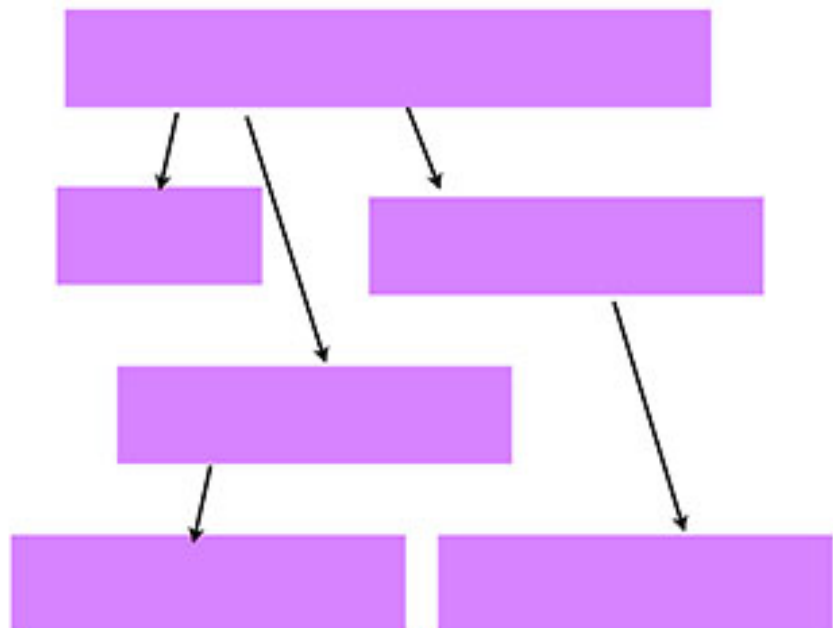
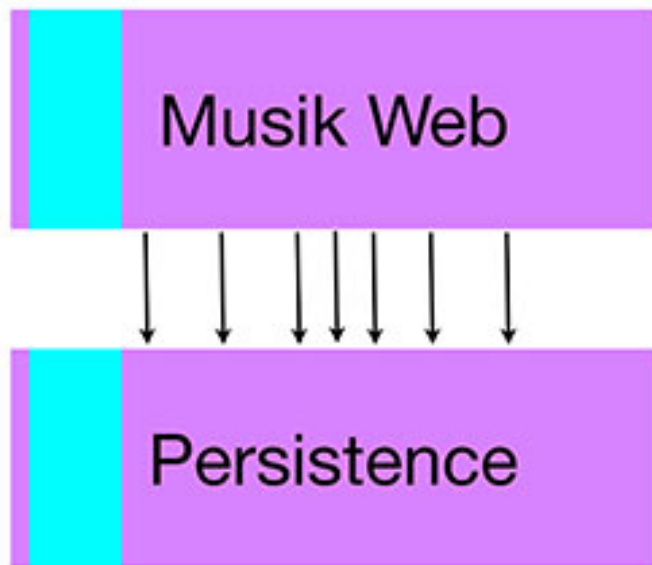
Persistence

```
graph TD; A[Musik Web] --> B[Persistence];
```

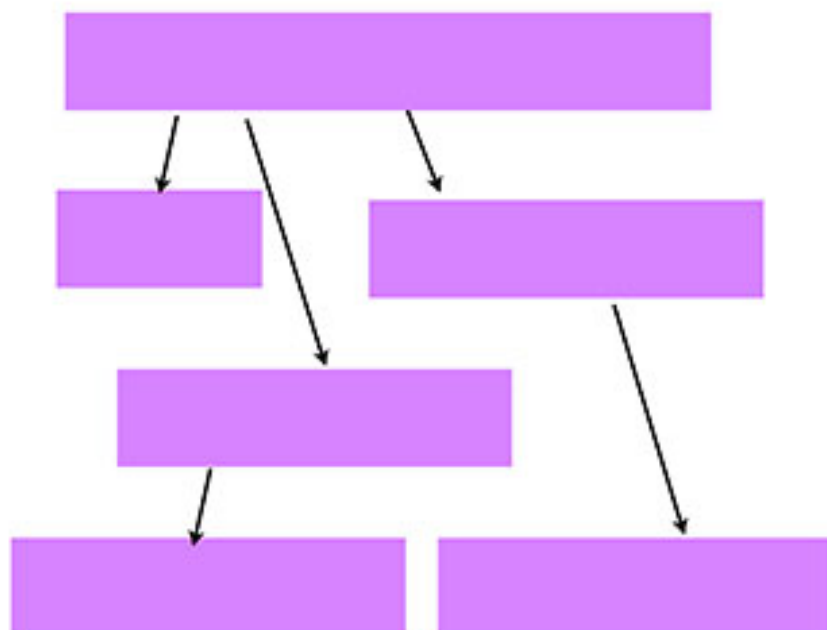
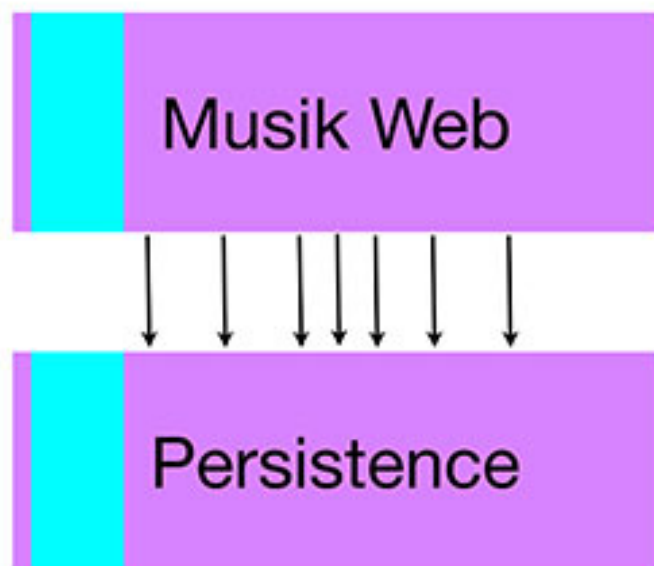
Musik Web

Persistence





ONION ARCHITECTURE



3 TIRED ARCHITECTURE

3 TIRED ARCHITECTURE

Presentation

3 TIRED ARCHITECTURE



Presentation

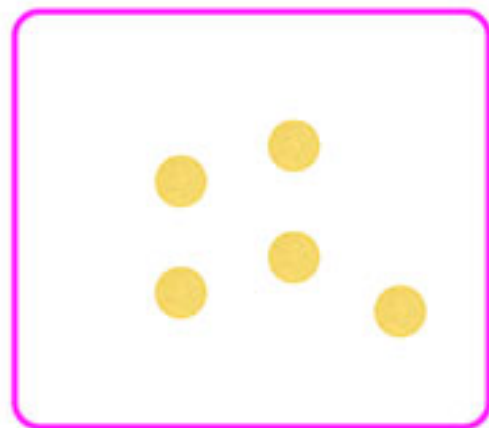
Business

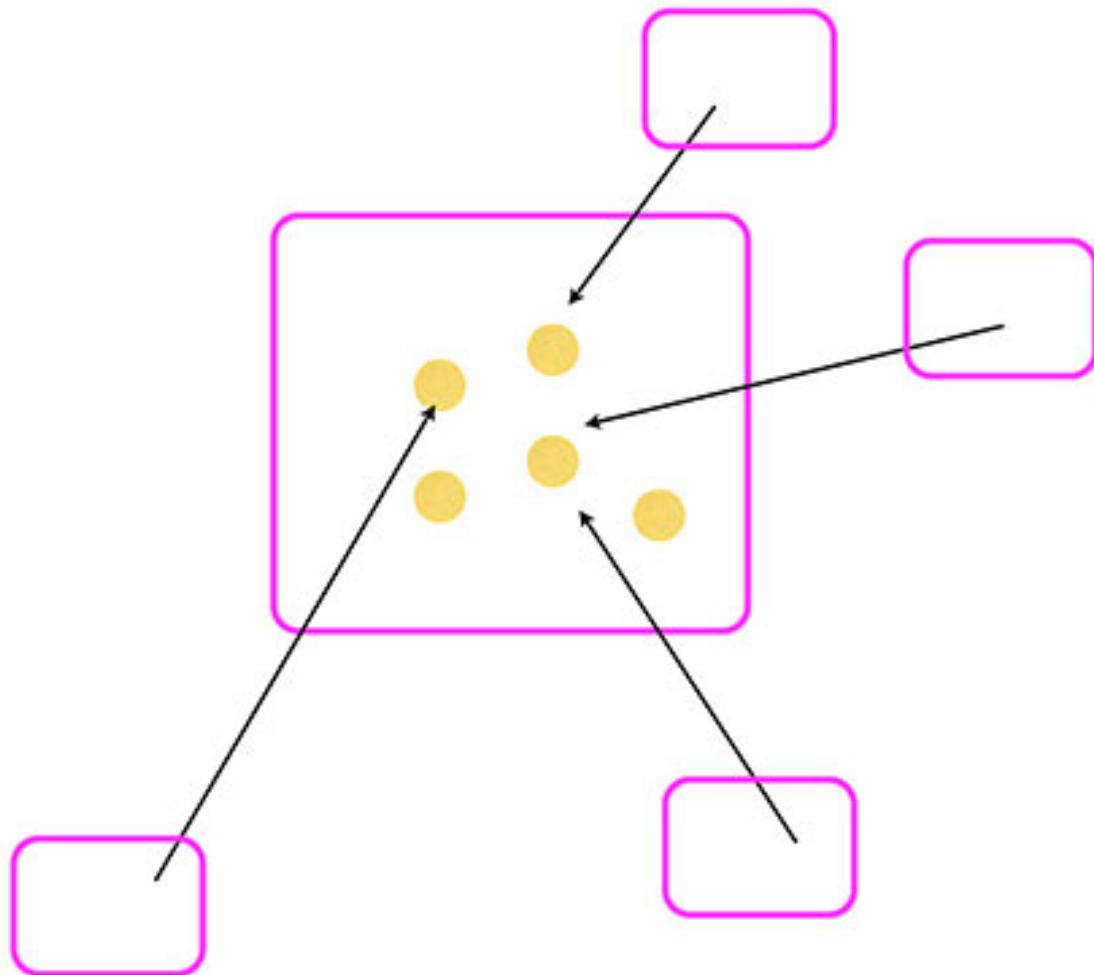
3 TIRED ARCHITECTURE

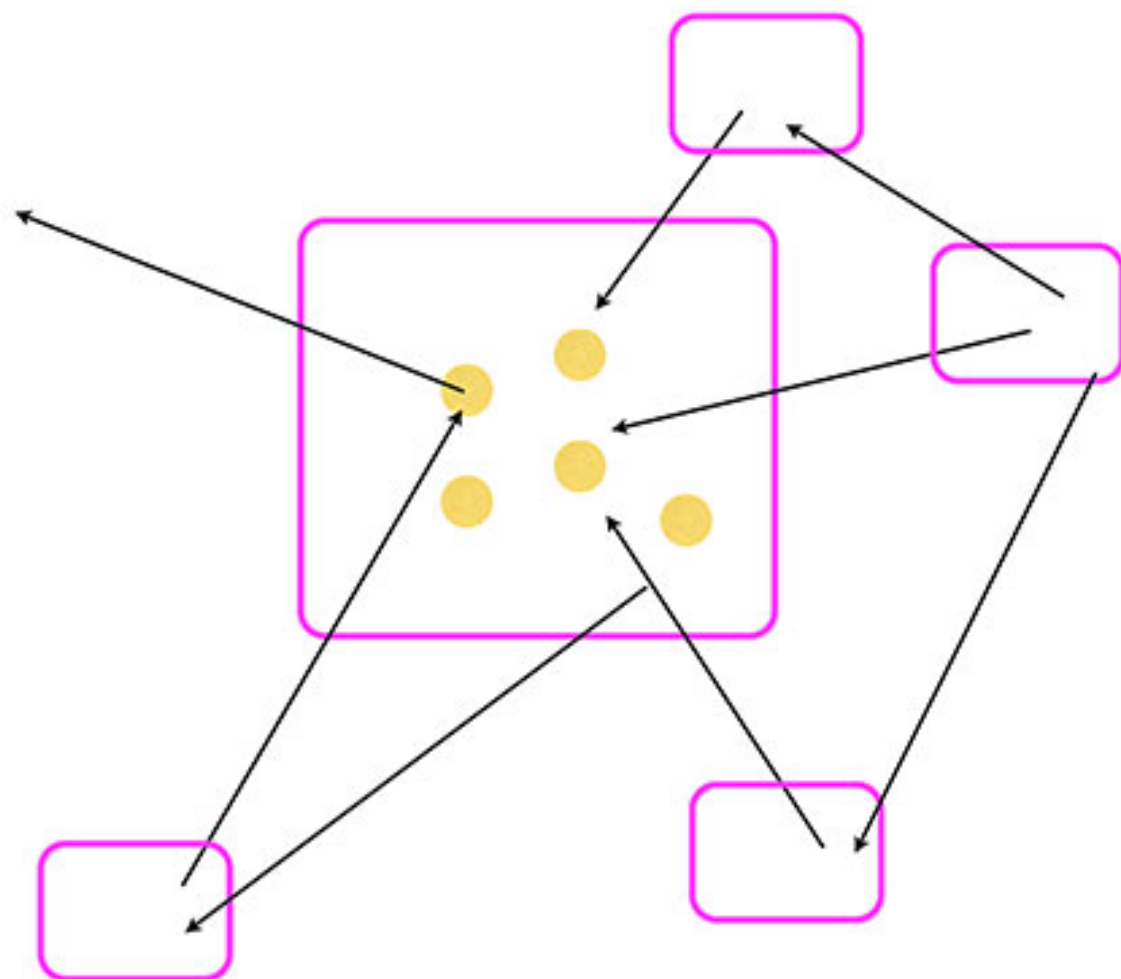
Presentation

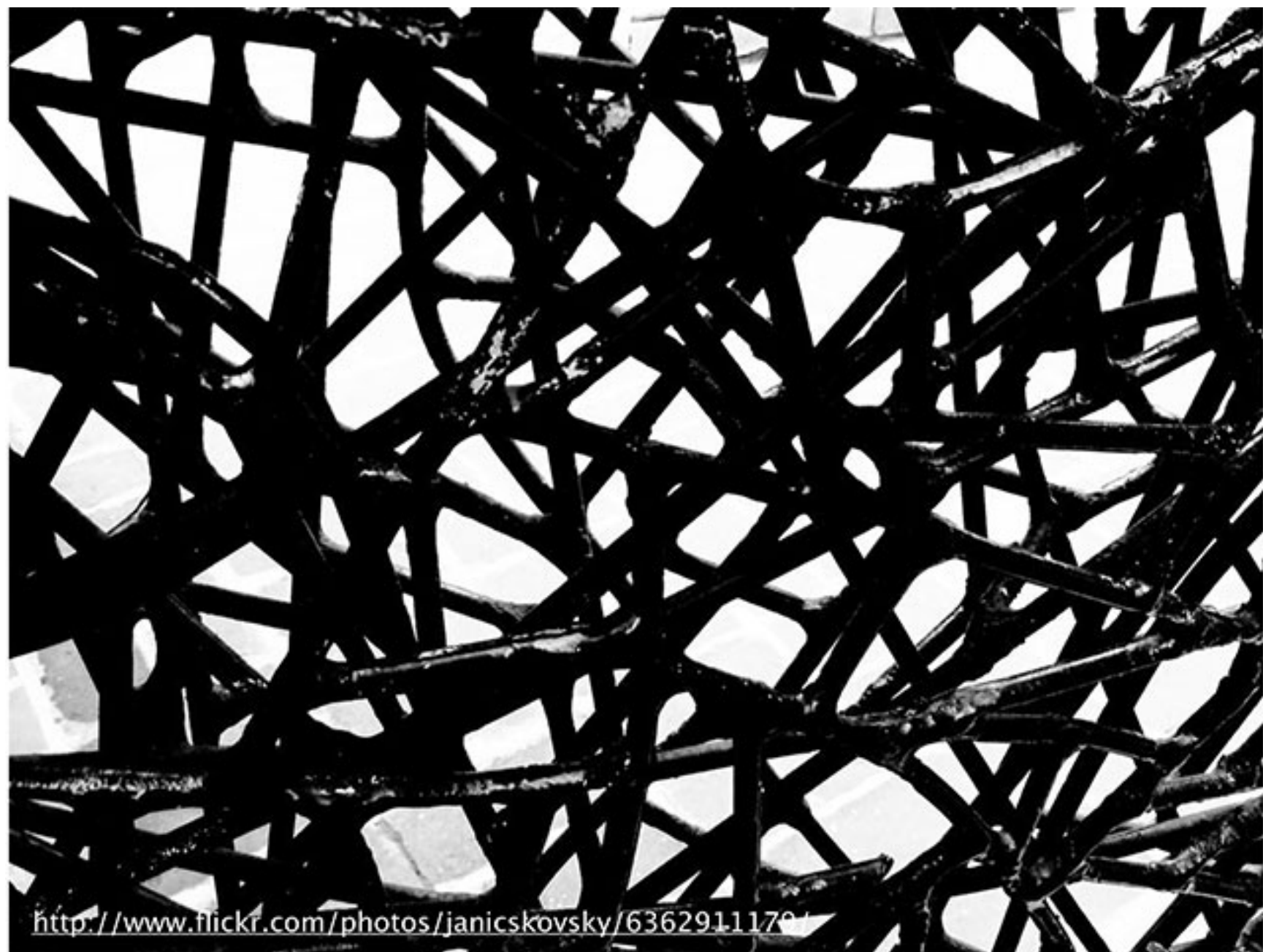
Business

Data Access









<http://www.flickr.com/photos/janicskovsky/6362911170/>

Domain-Driven

DESIGN

Tackling Complexity in the Heart of Software



Eric Evans

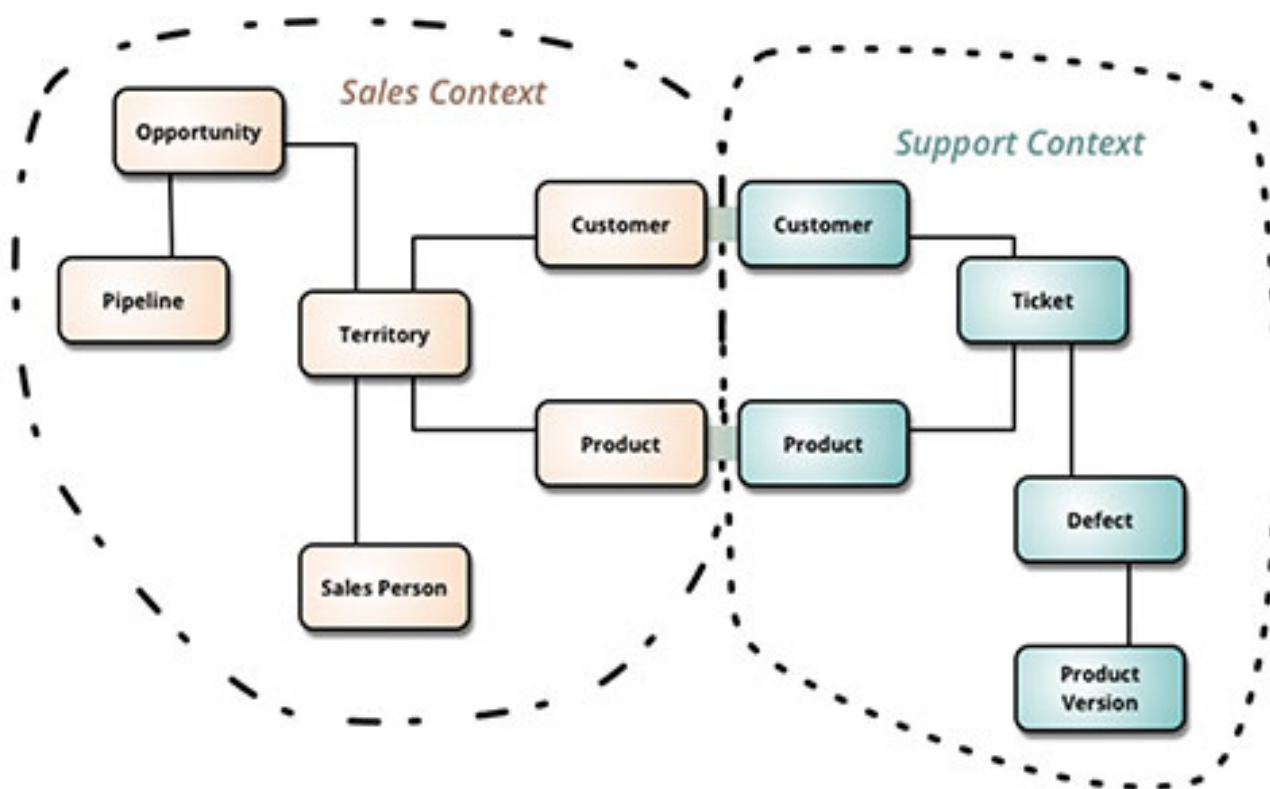
Foreword by Martin Fowler

Bounded Context

"The delimited applicability of a particular model. BOUNDING CONTEXTS gives team members a clear and shared understanding of what has to be consistent and what can develop independently."

“A specific responsibility enforced by explicit boundaries”

<http://www.sapiensworks.com/blog/post/2012/04/17/DDD-The-Bounded-Context-Explained.aspx>



<http://martinfowler.com/bliki/images/boundedContext/sketch.png>

Capabilities?



Register a new customer

Register a new customer

Oil Change

Register a new customer

Oil Change

Emission Test

Register a new customer

Oil Change

Emission Test

Replace Tire

Register a new customer

Oil Change

Emission Test

Replace Tire

Order parts

Register a new customer

Oil Change

Emission Test

Replace Tire

Order parts

Send an invoice

Register a new customer

Oil Change

Emission Test

Replace Tire

Order parts

Send an invoice

Make a repair

Register a new customer

Oil Change

Emission Test

Replace Tire

Contact a customer

Order parts

Send an invoice

Make a repair

Register a new customer

Contact a customer

Send an invoice

Oil Change

Replace Tire

Make a repair

Emission Test

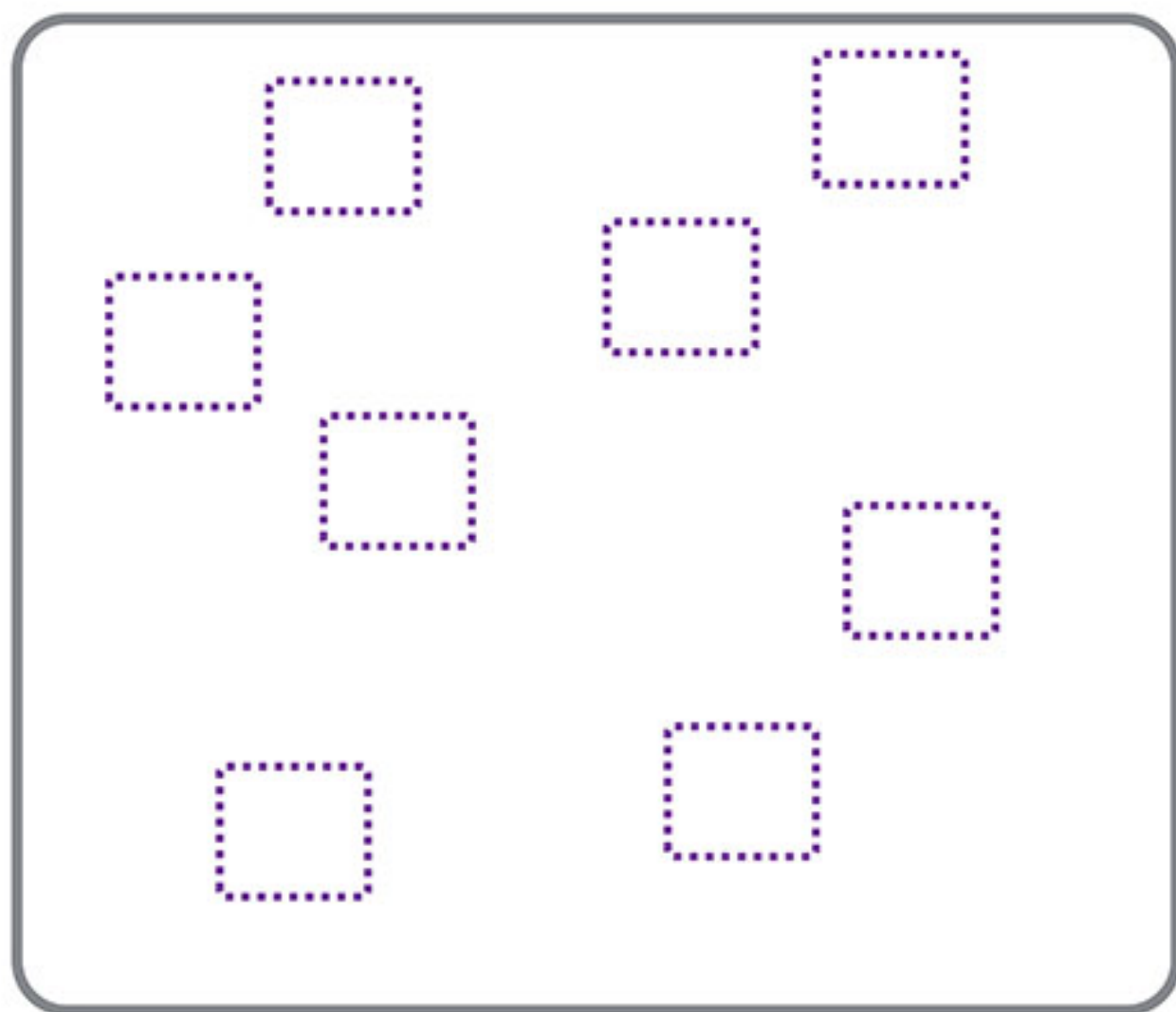
Order parts

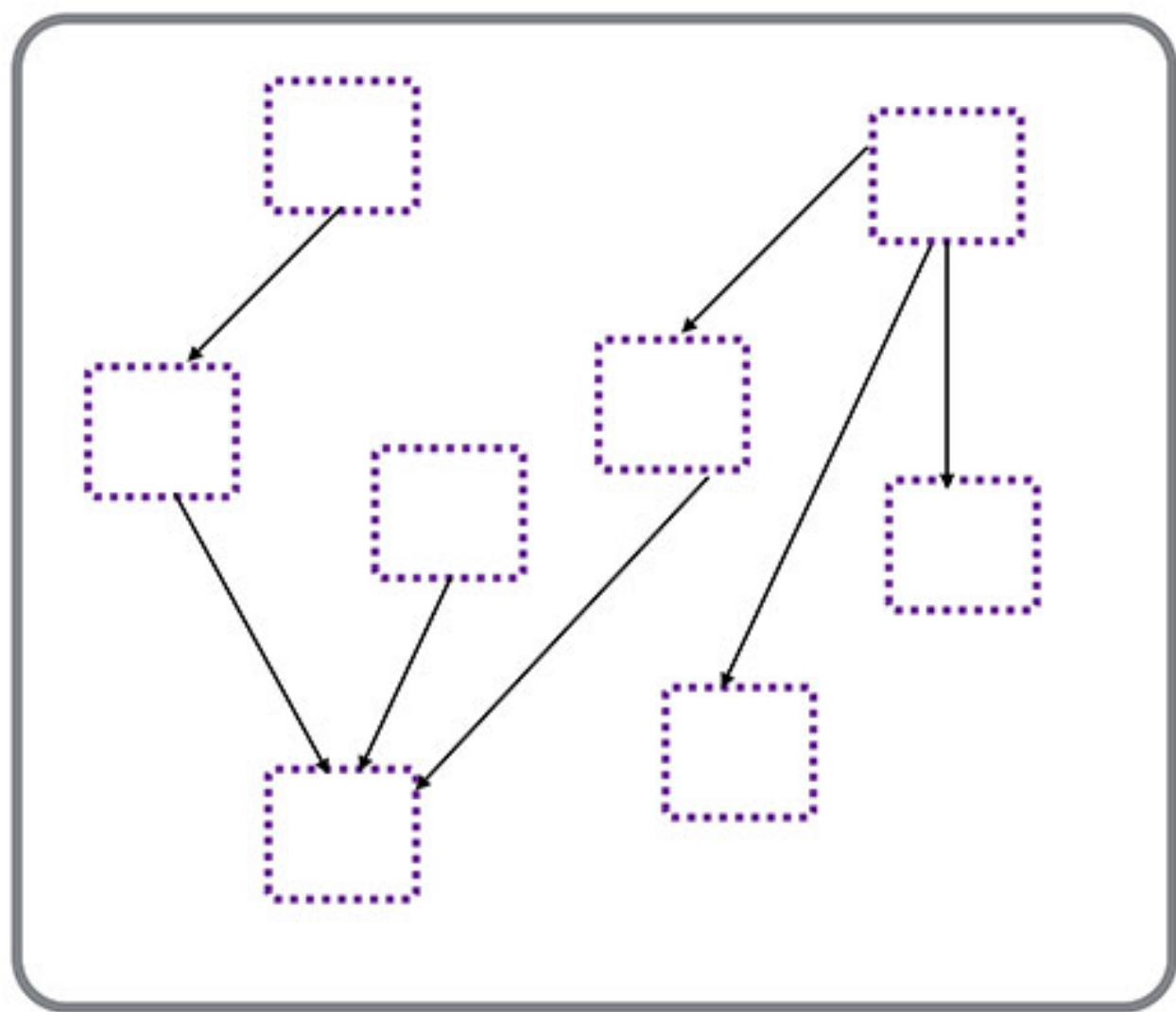
Customer Management

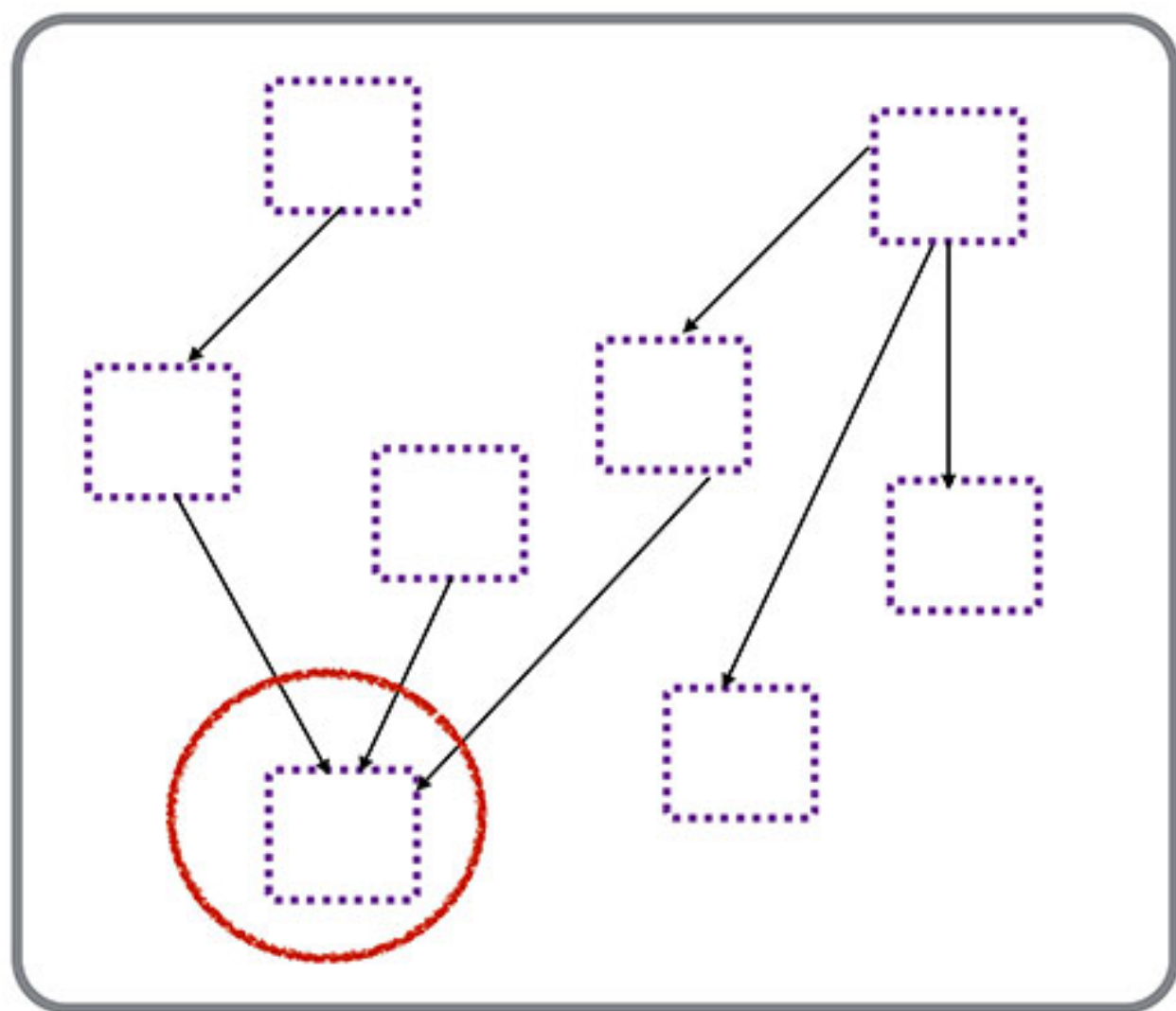
Maintenance

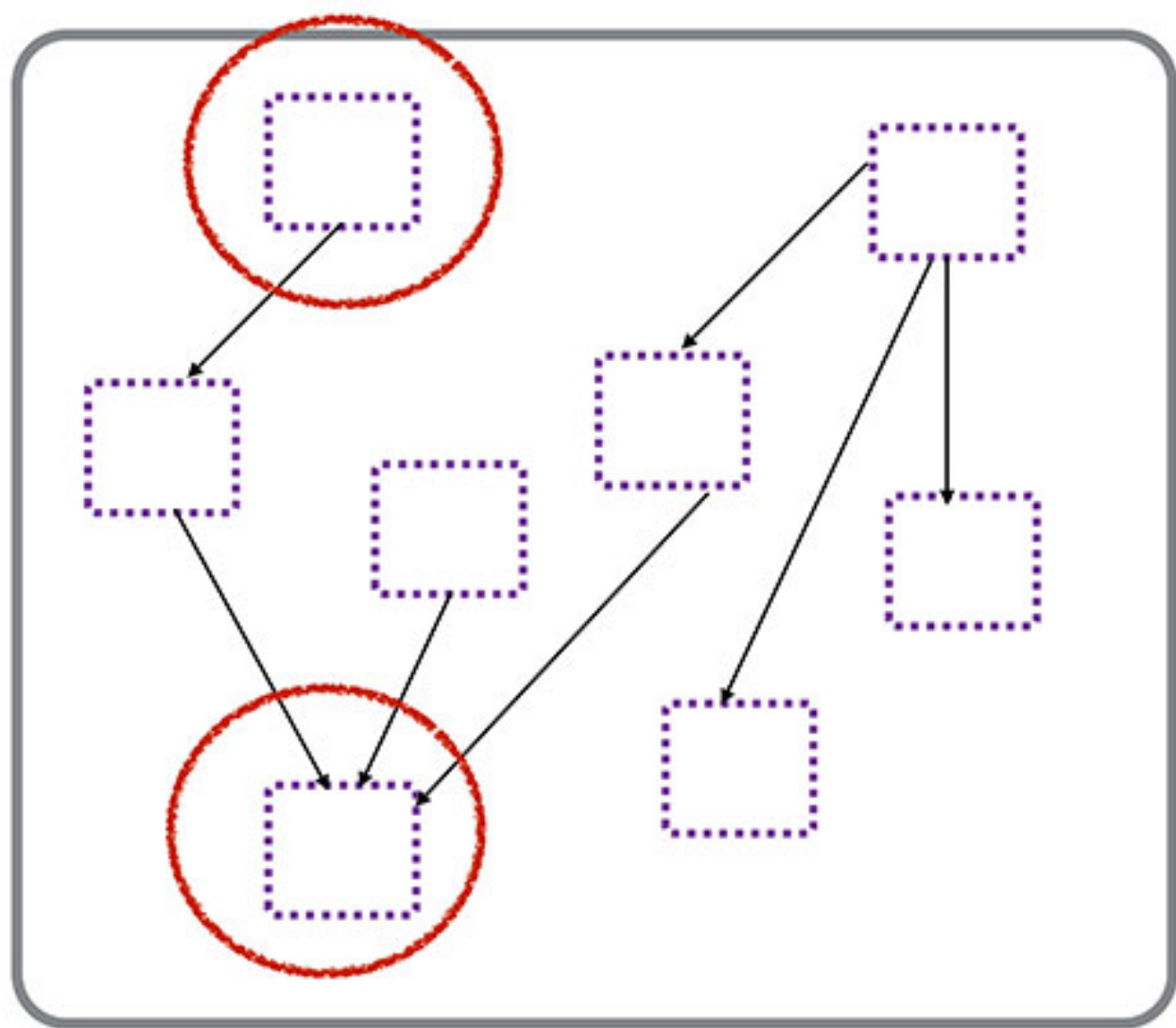
Inventory

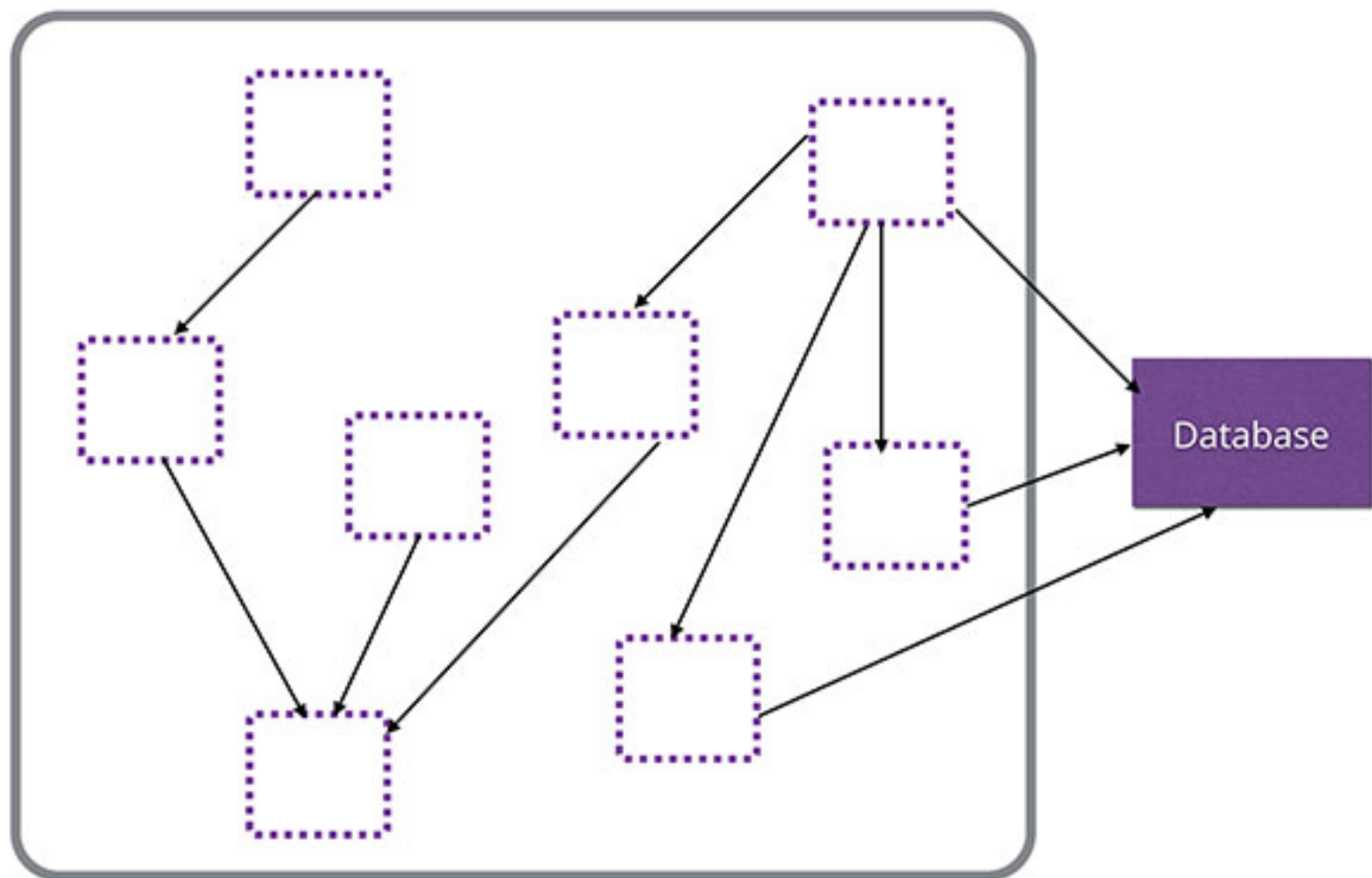
Remember The Goal!

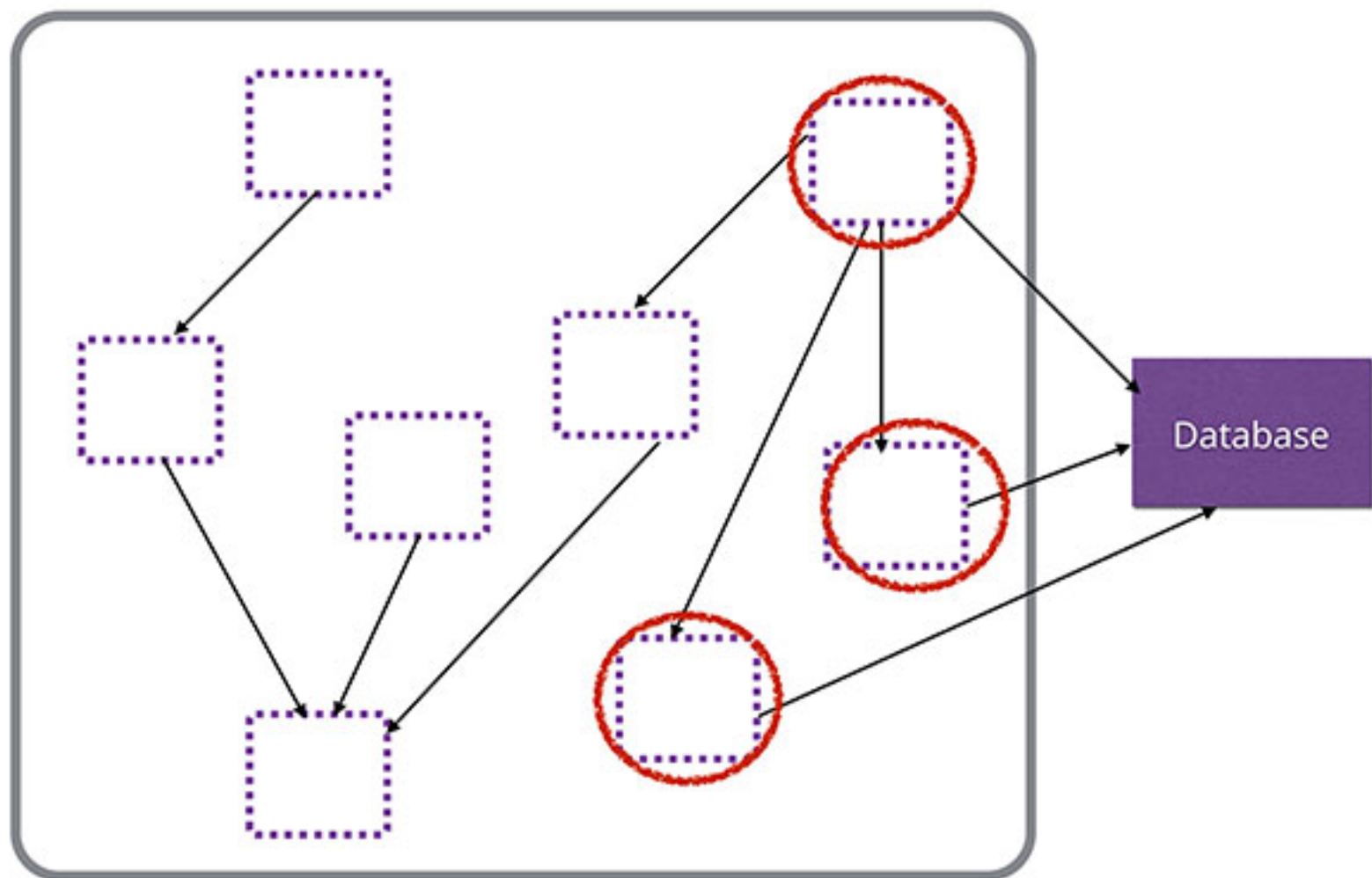


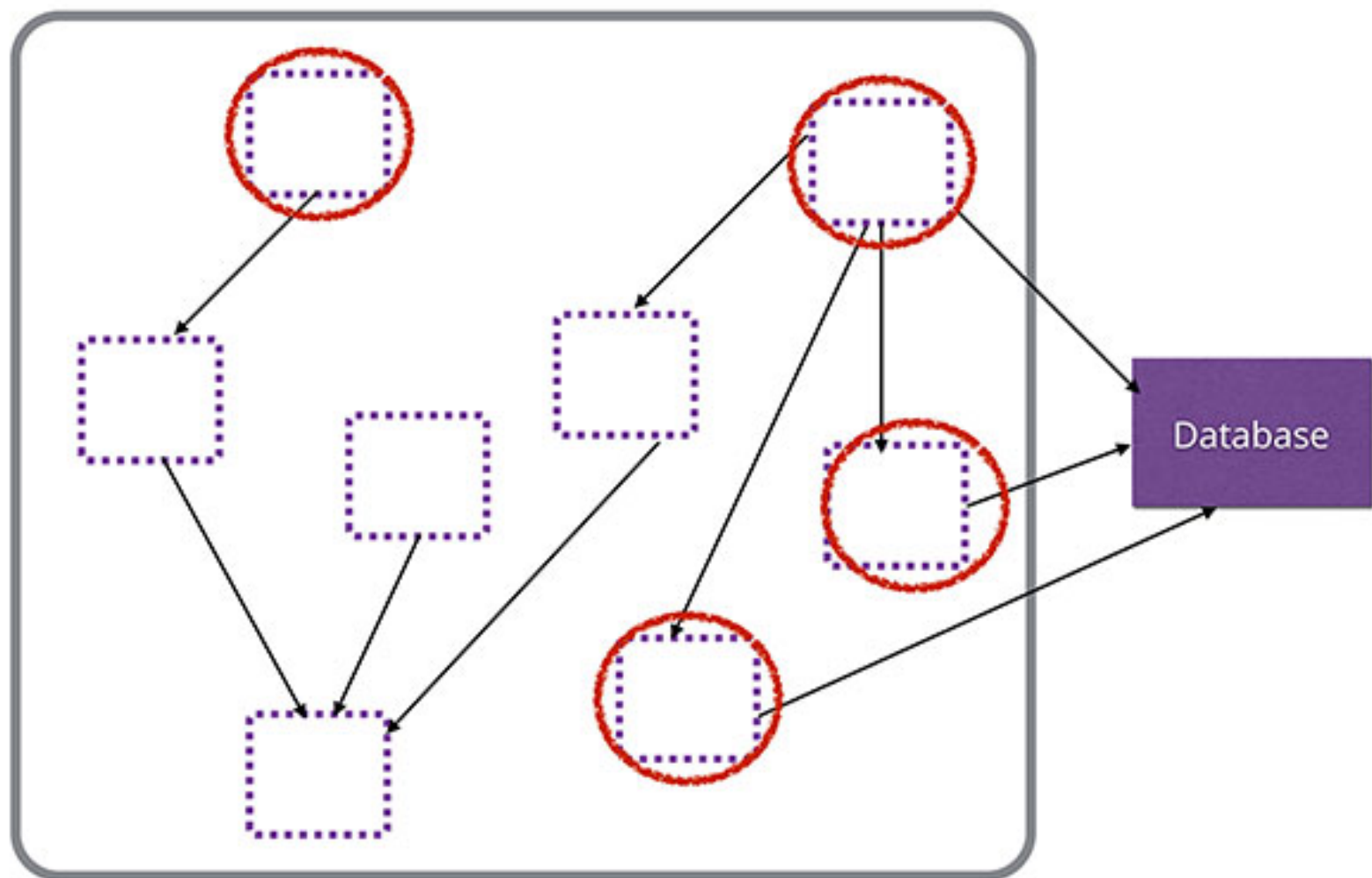












So all things being equal, look for:

So all things being equal, look for:

modules with few inbound
dependencies

So all things being equal, look for:

modules with few inbound
dependencies

which can be stateless candidates

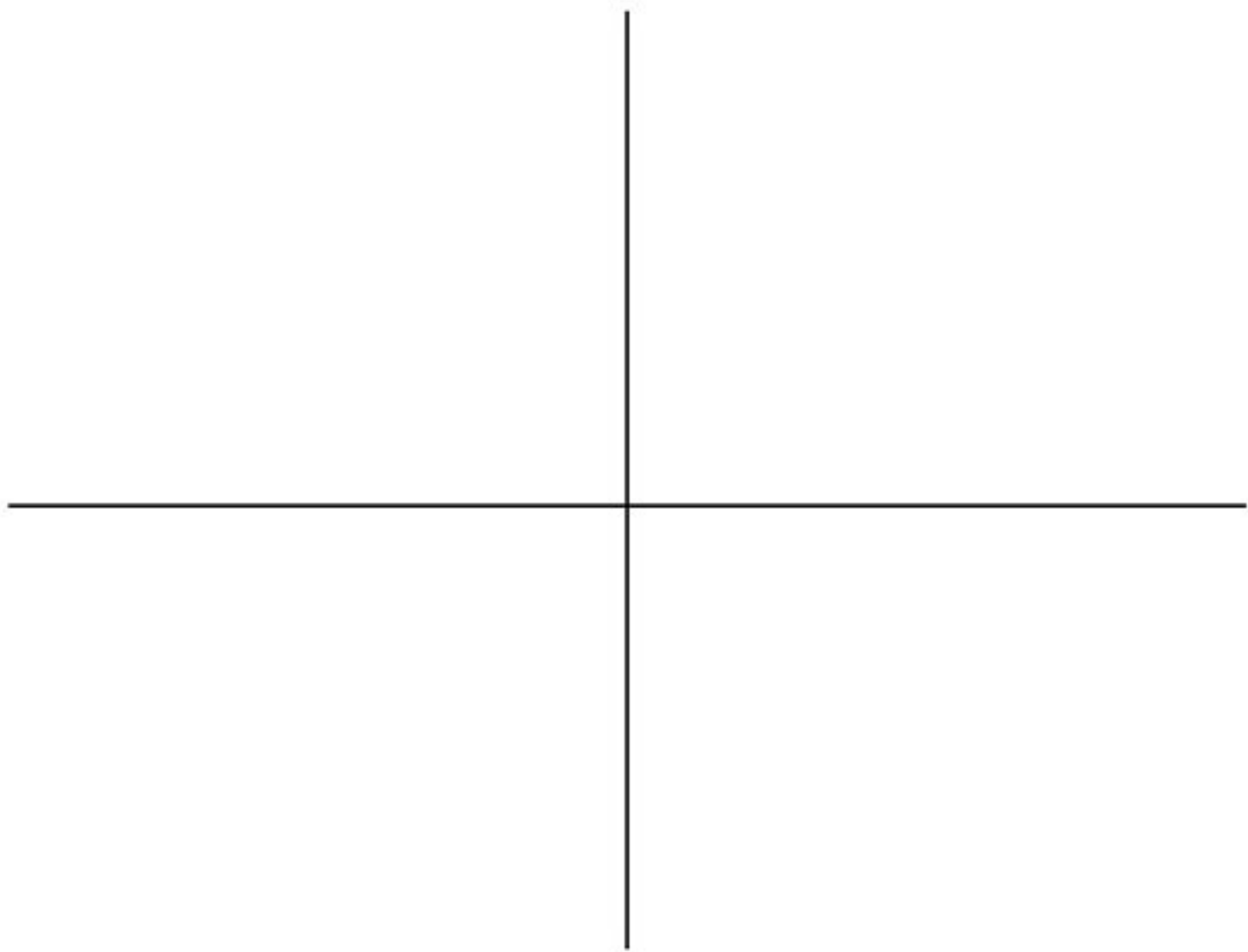
Ease Of Decomposition

VS

Benefits Of Decomposition

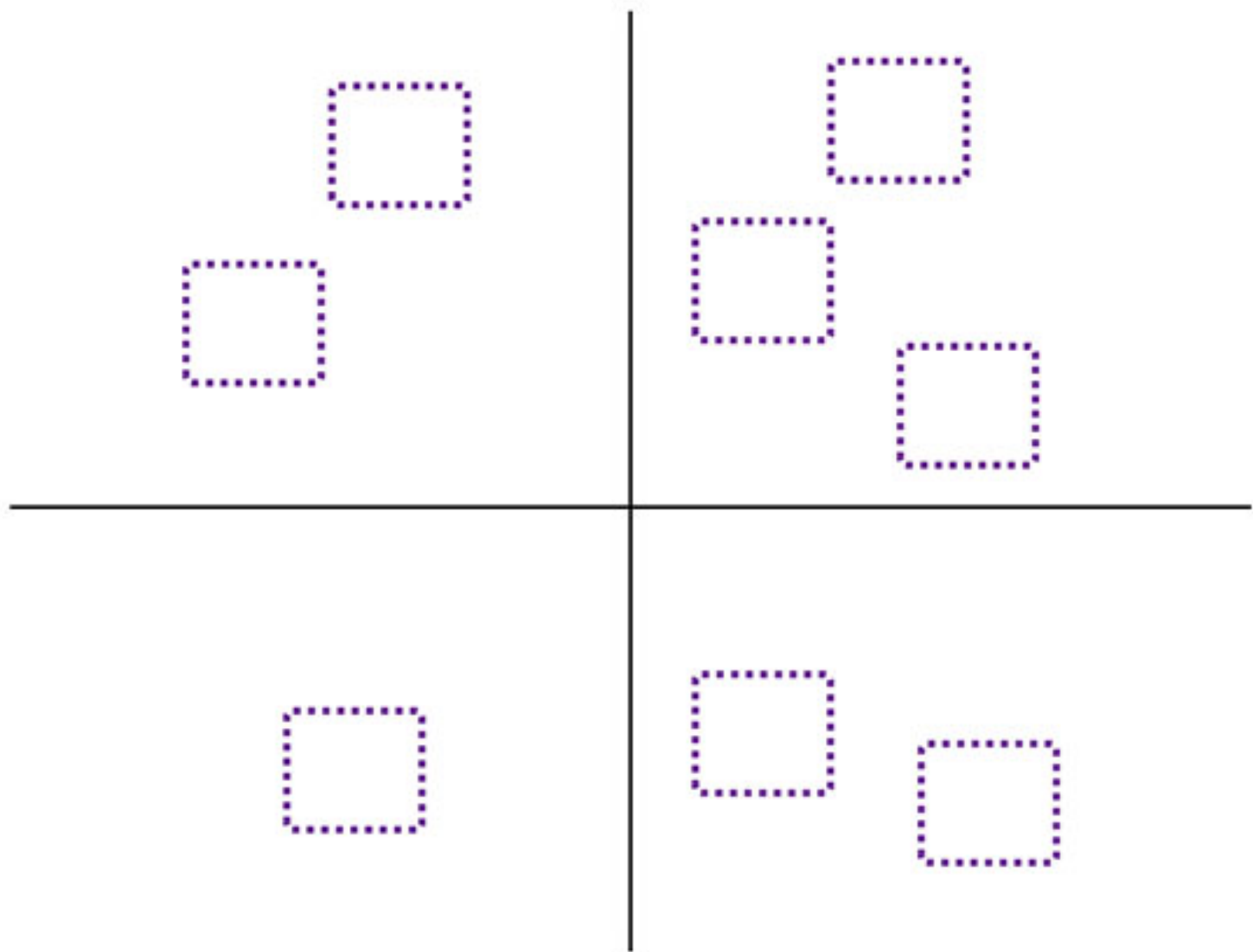
Benefit

Difficulty



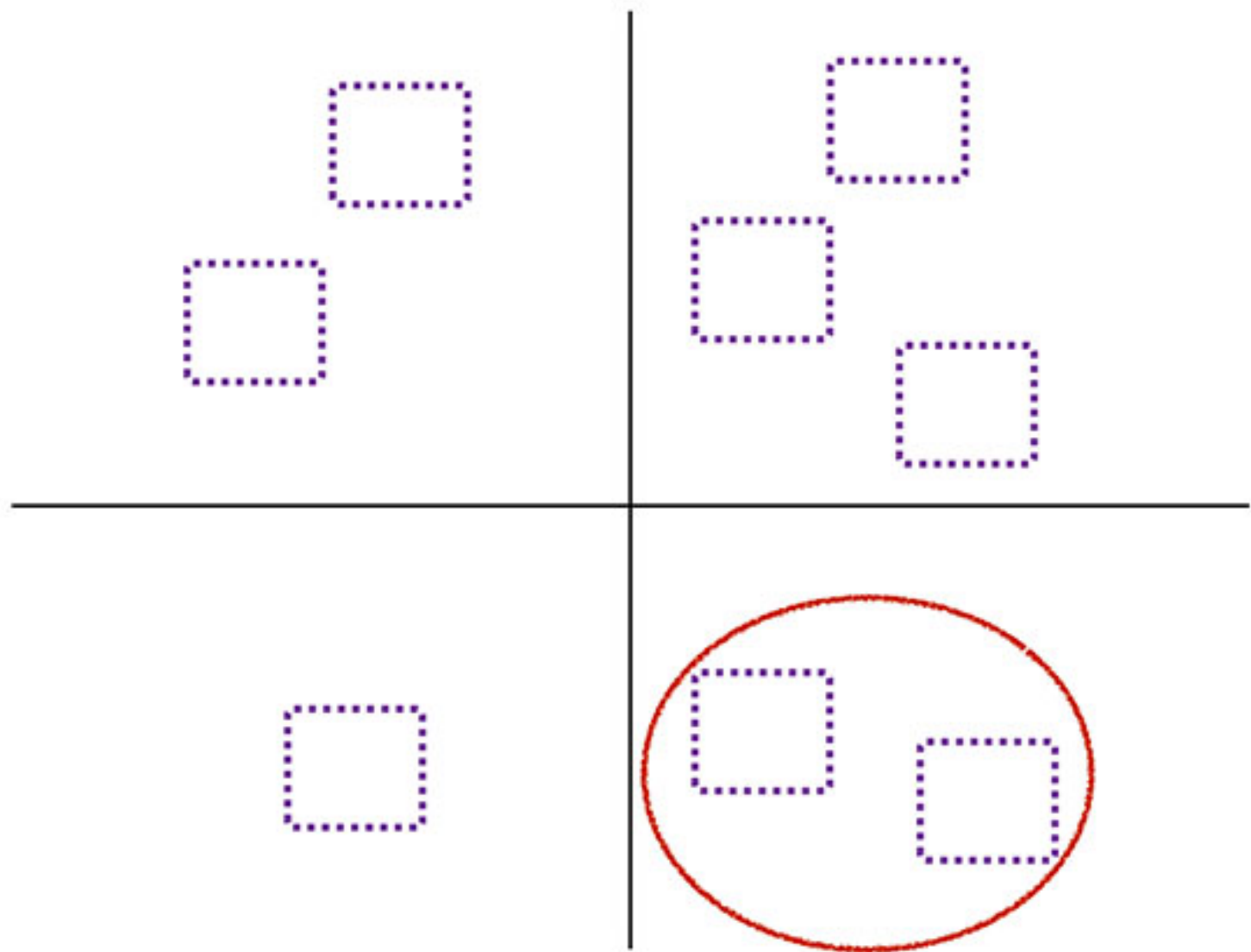
Benefit

Difficulty



Benefit

Difficulty

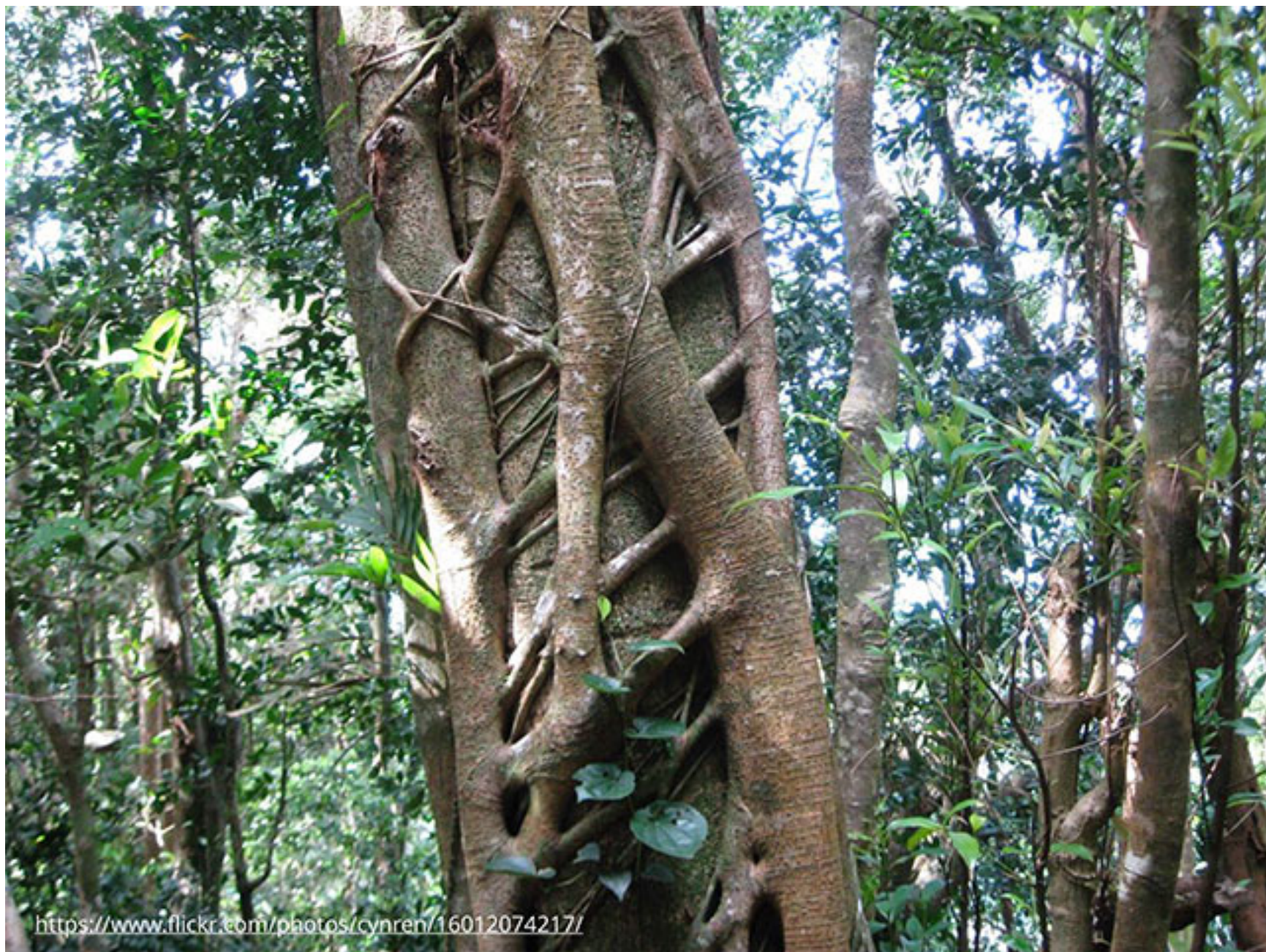


Incremental Change

Enter the strangler pattern

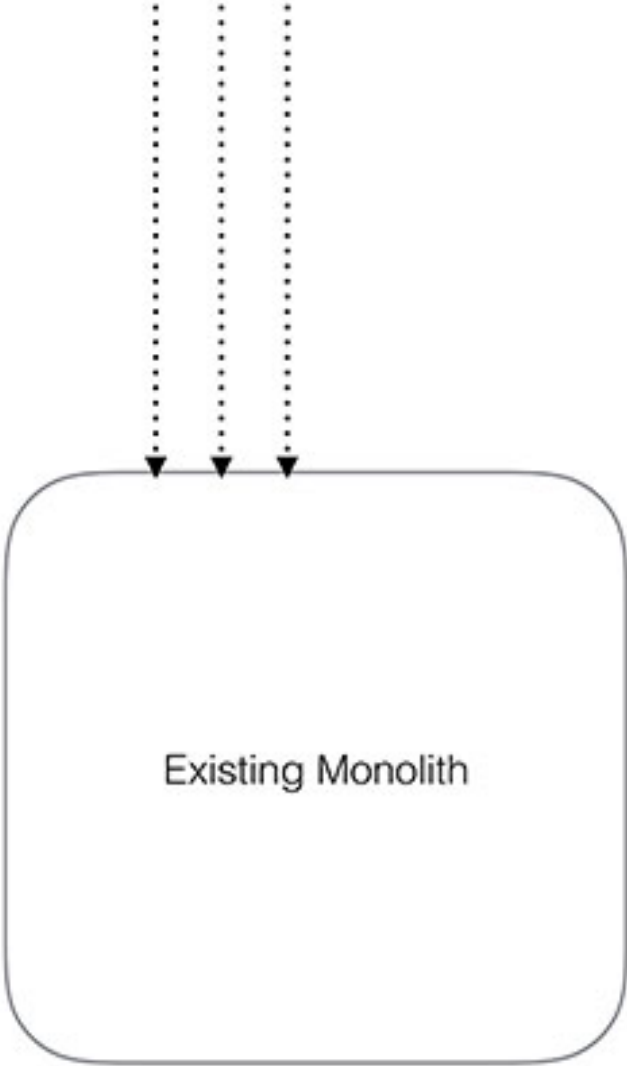
Enter the strangler pattern

Great pattern, terrible name



<https://www.flickr.com/photos/cynren/16012074217/>

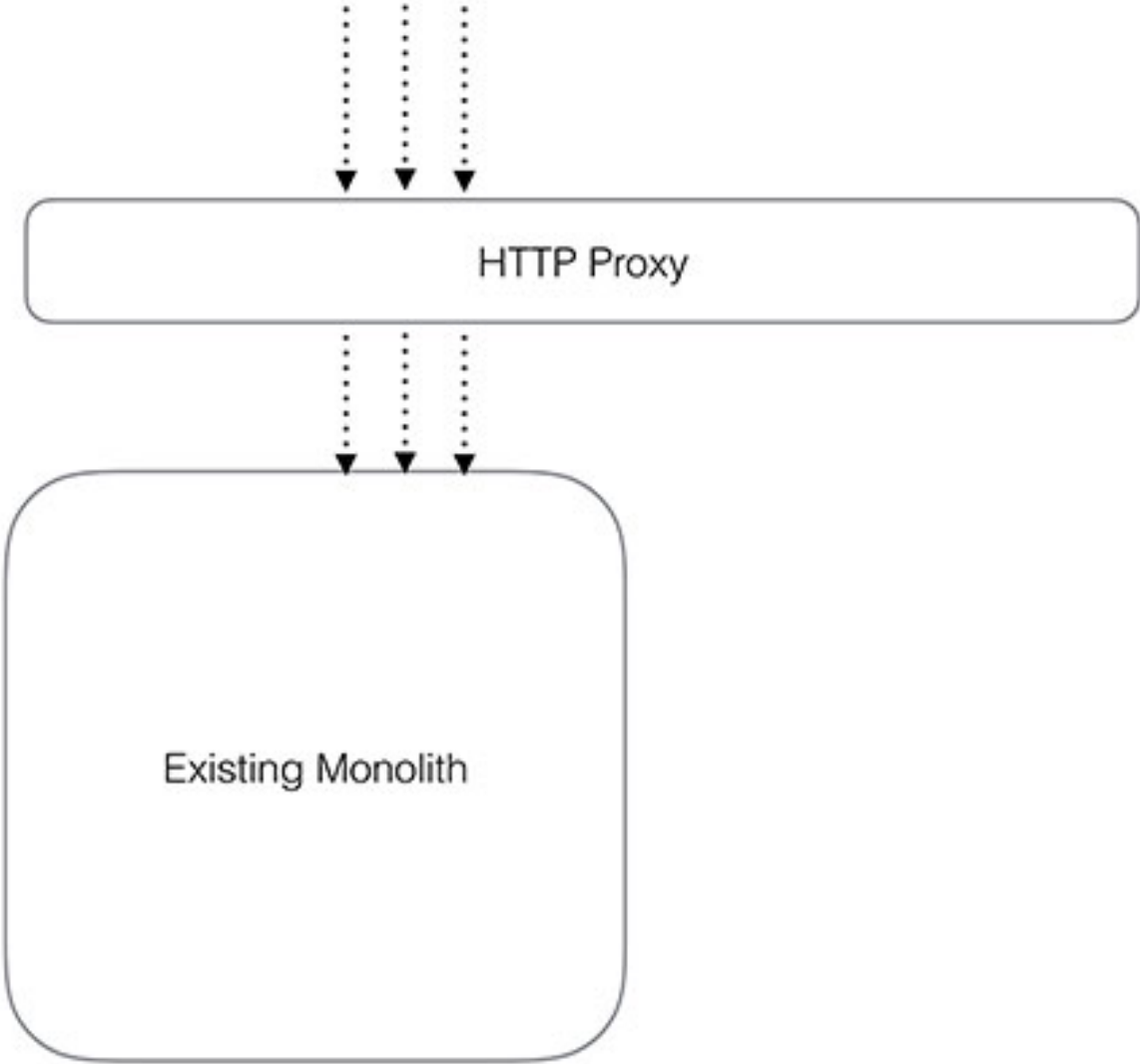
HTTP PROXY



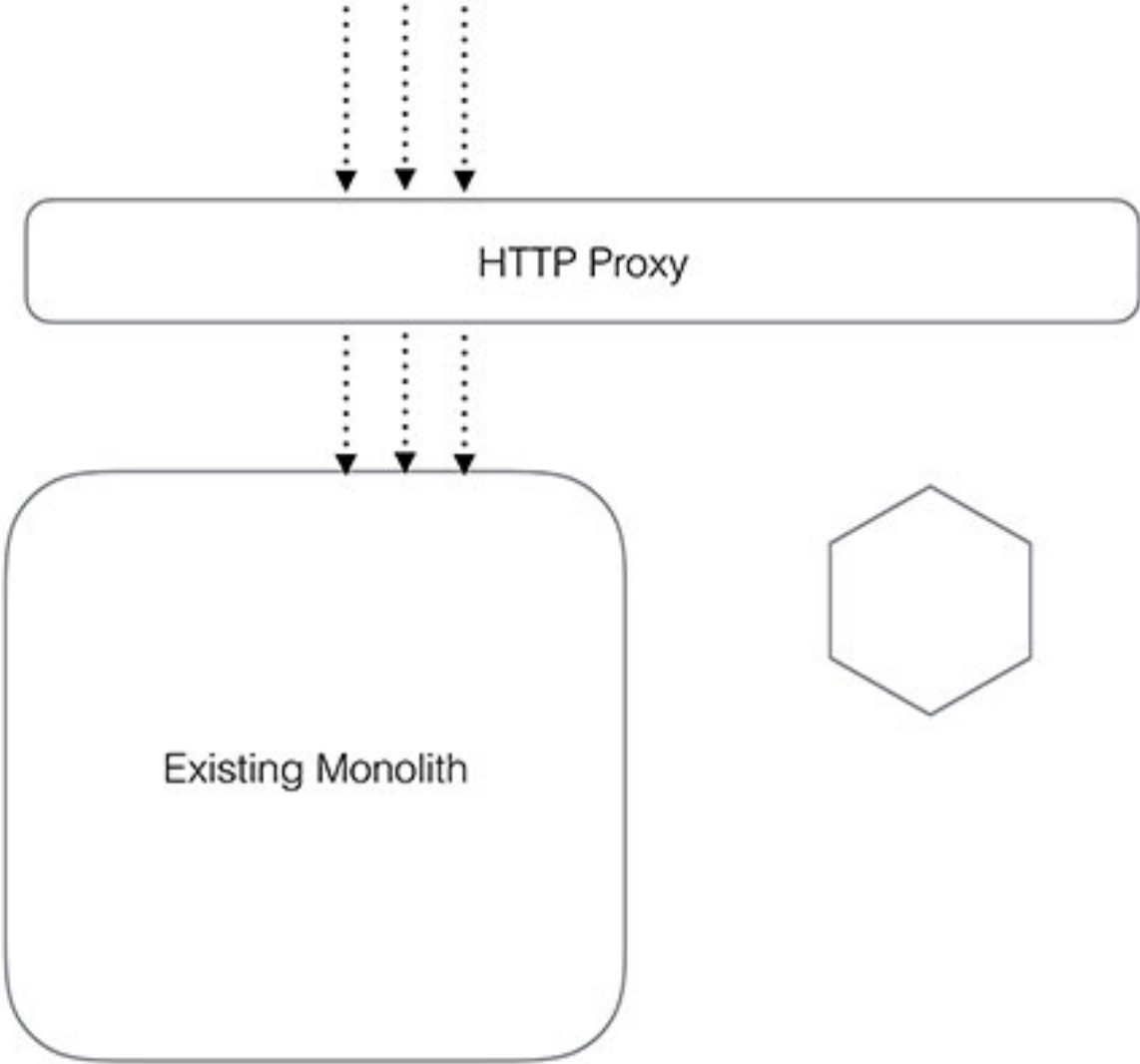
HTTP PROXY



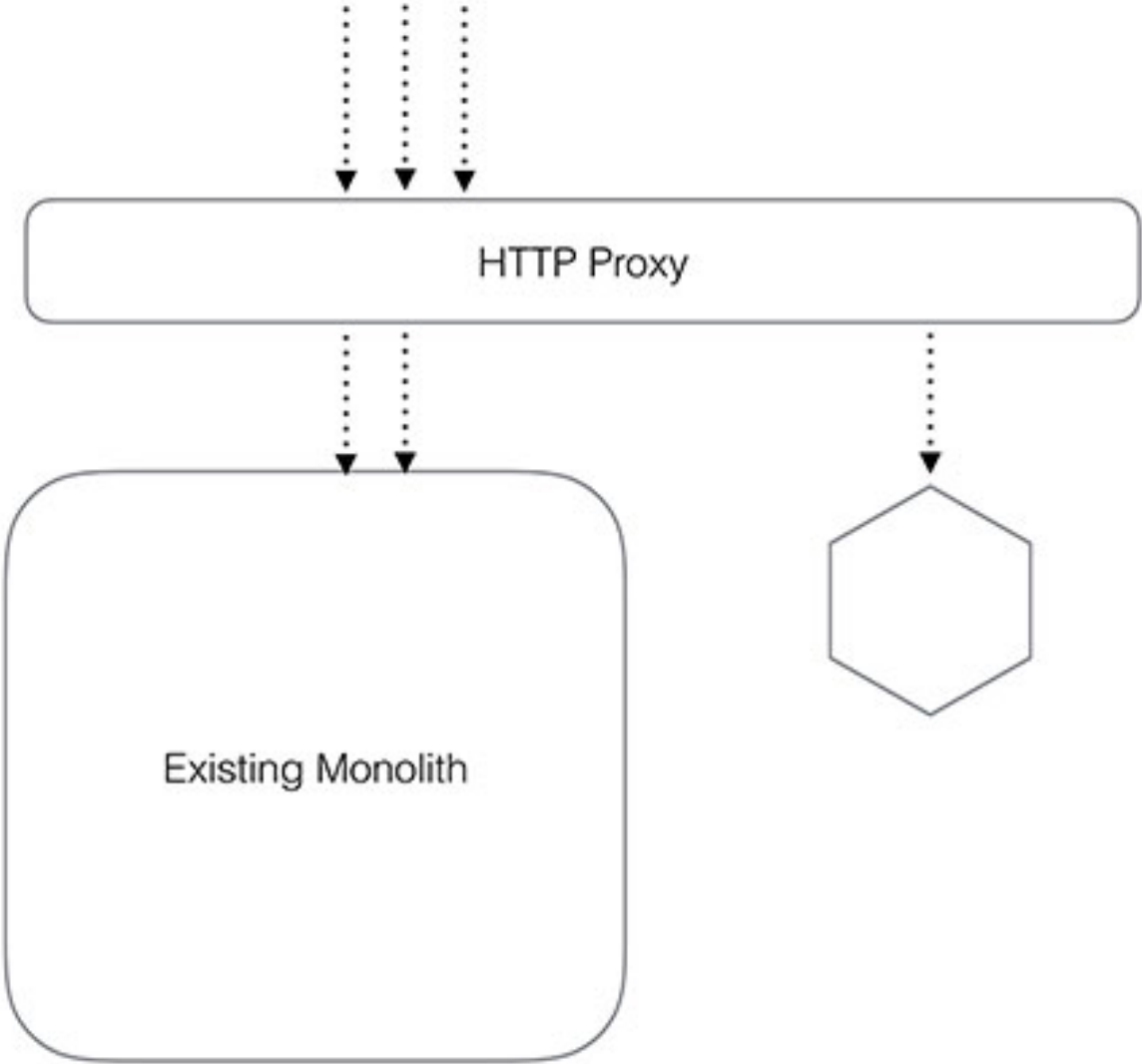
HTTP PROXY



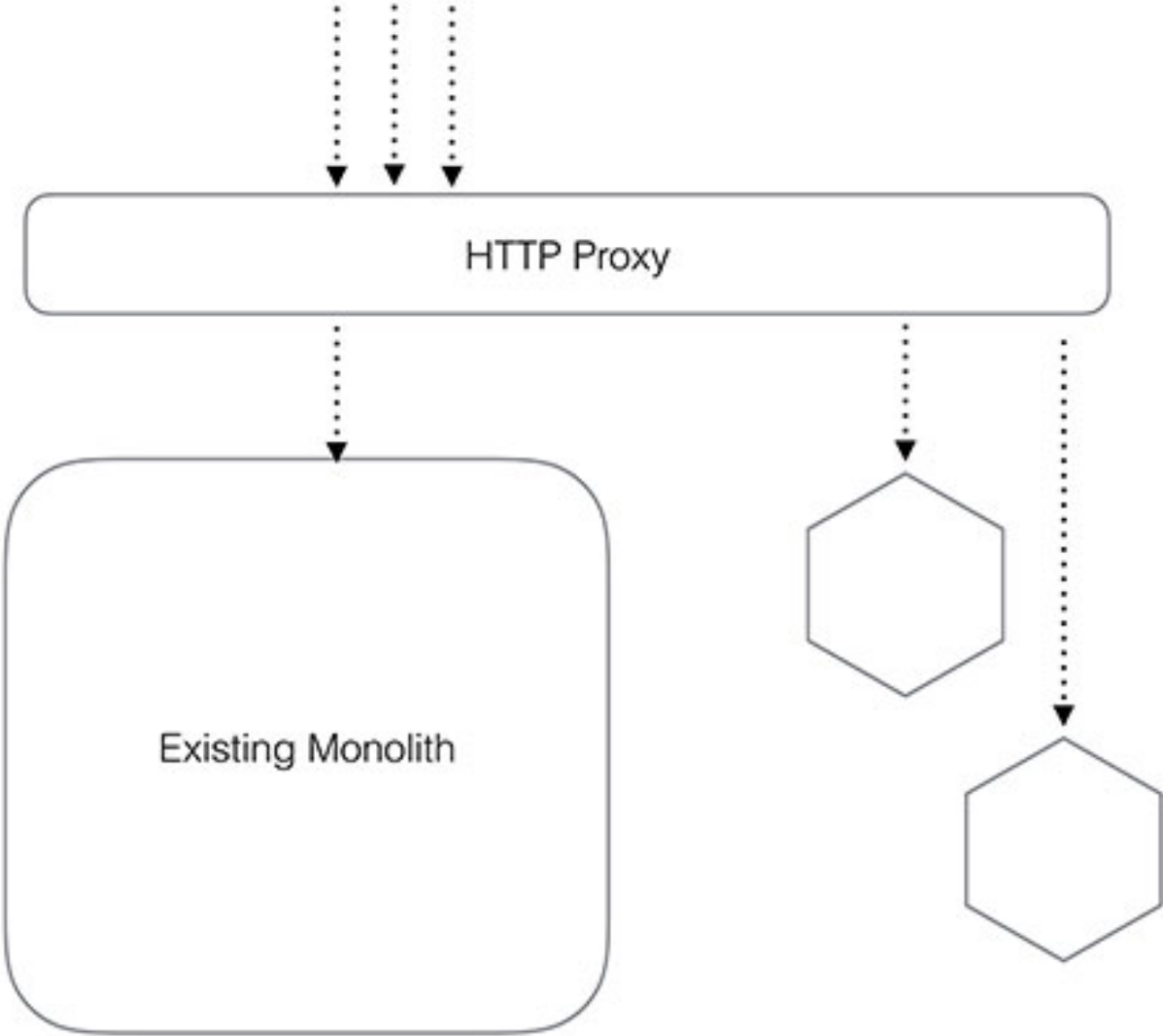
HTTP PROXY



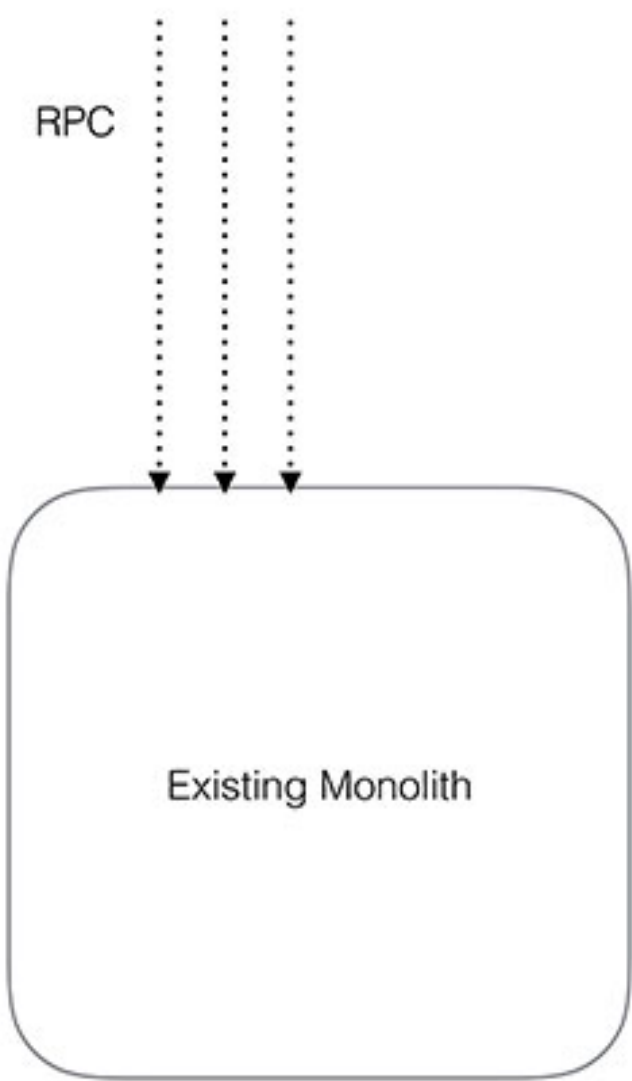
HTTP PROXY



HTTP PROXY



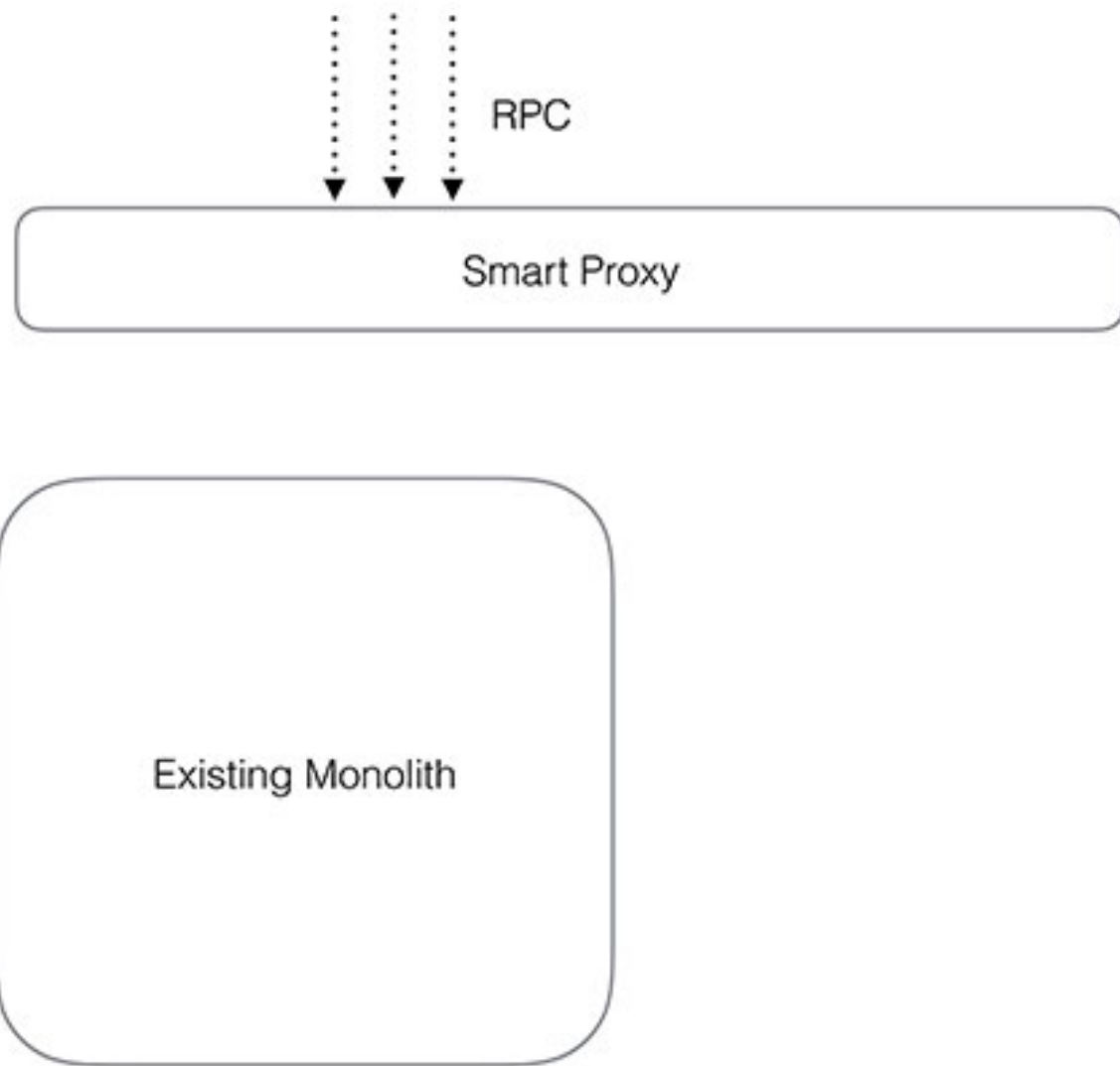
PROTOCOL CHANGING PROXY



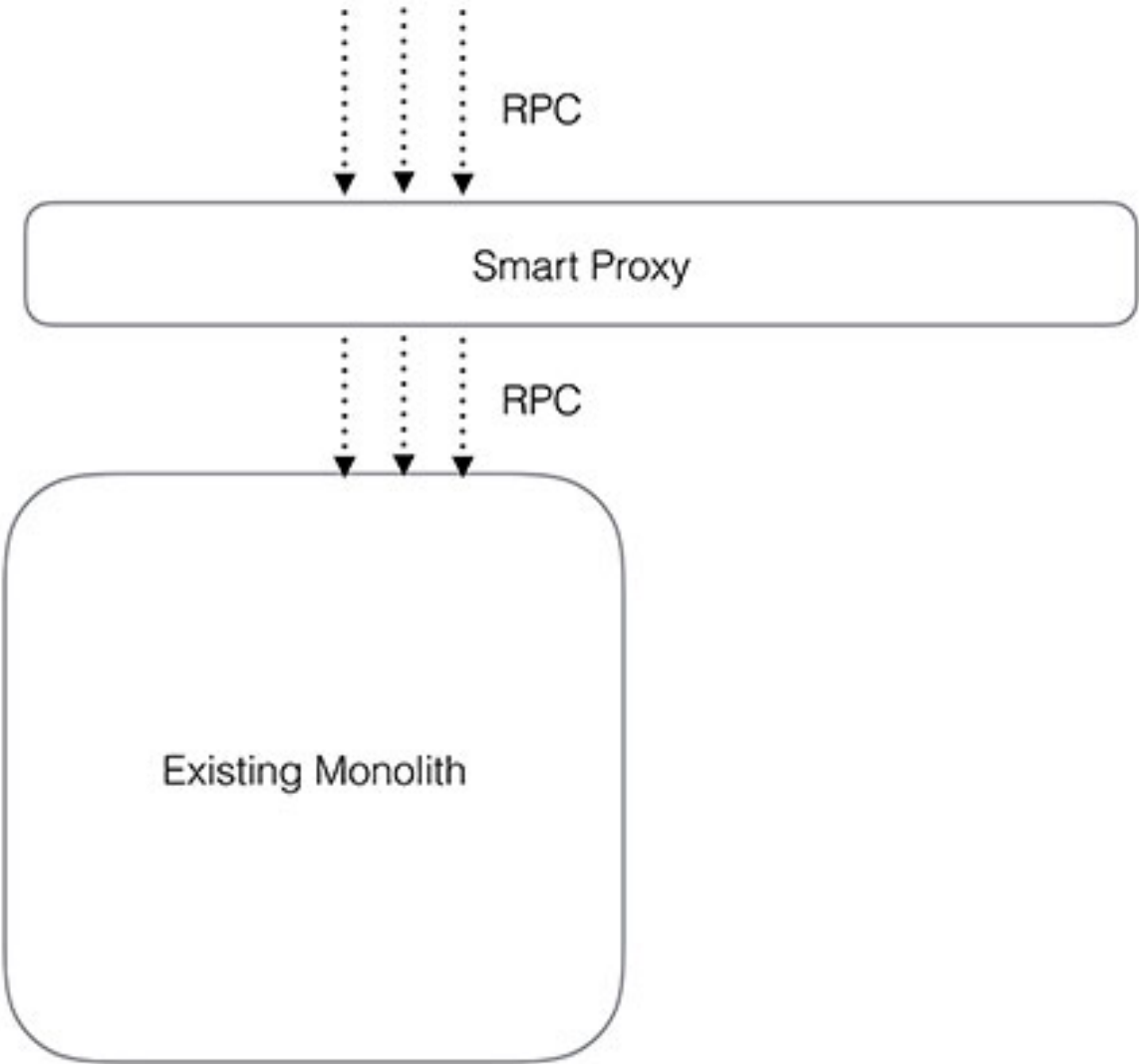
PROTOCOL CHANGING PROXY



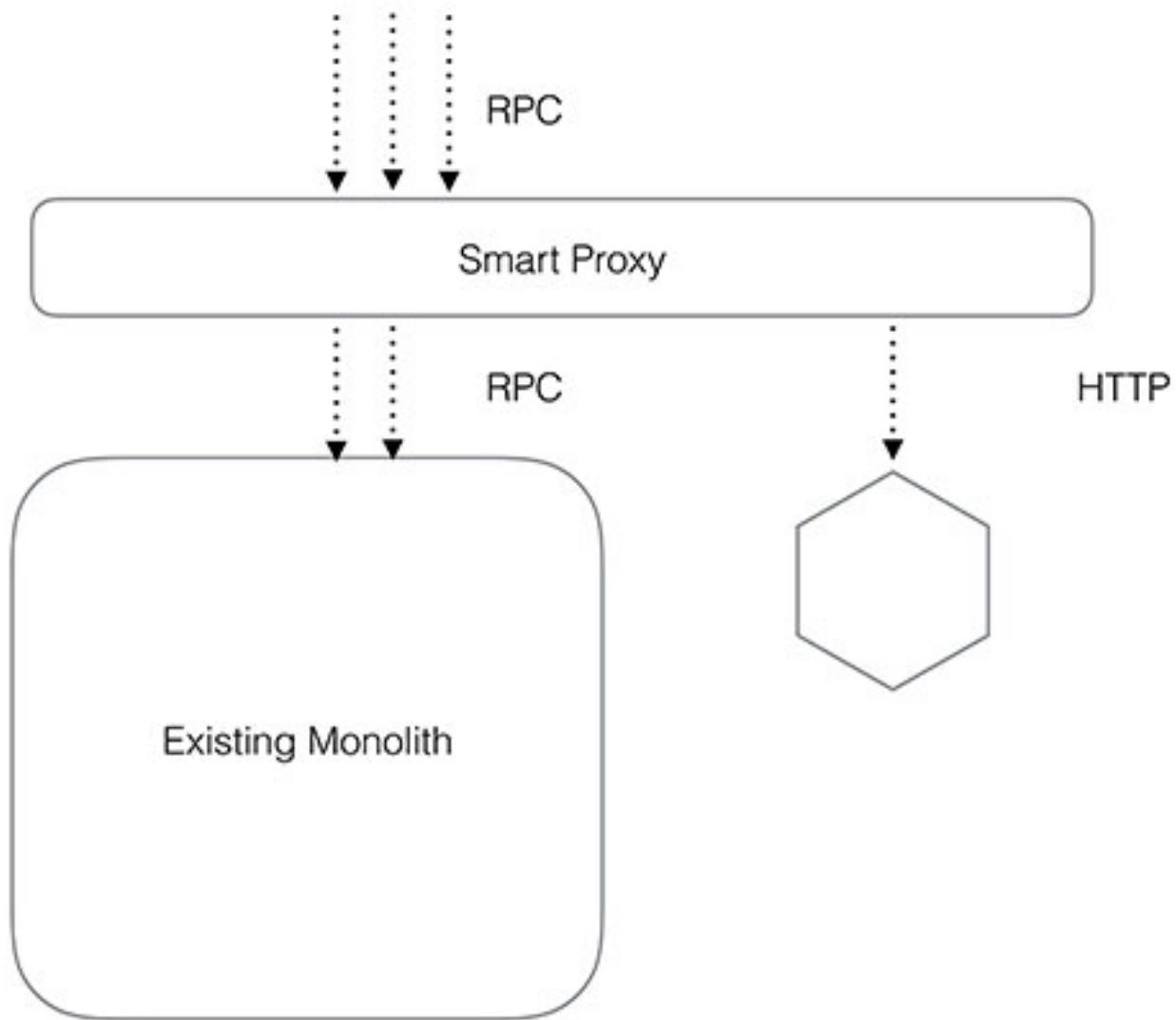
PROTOCOL CHANGING PROXY



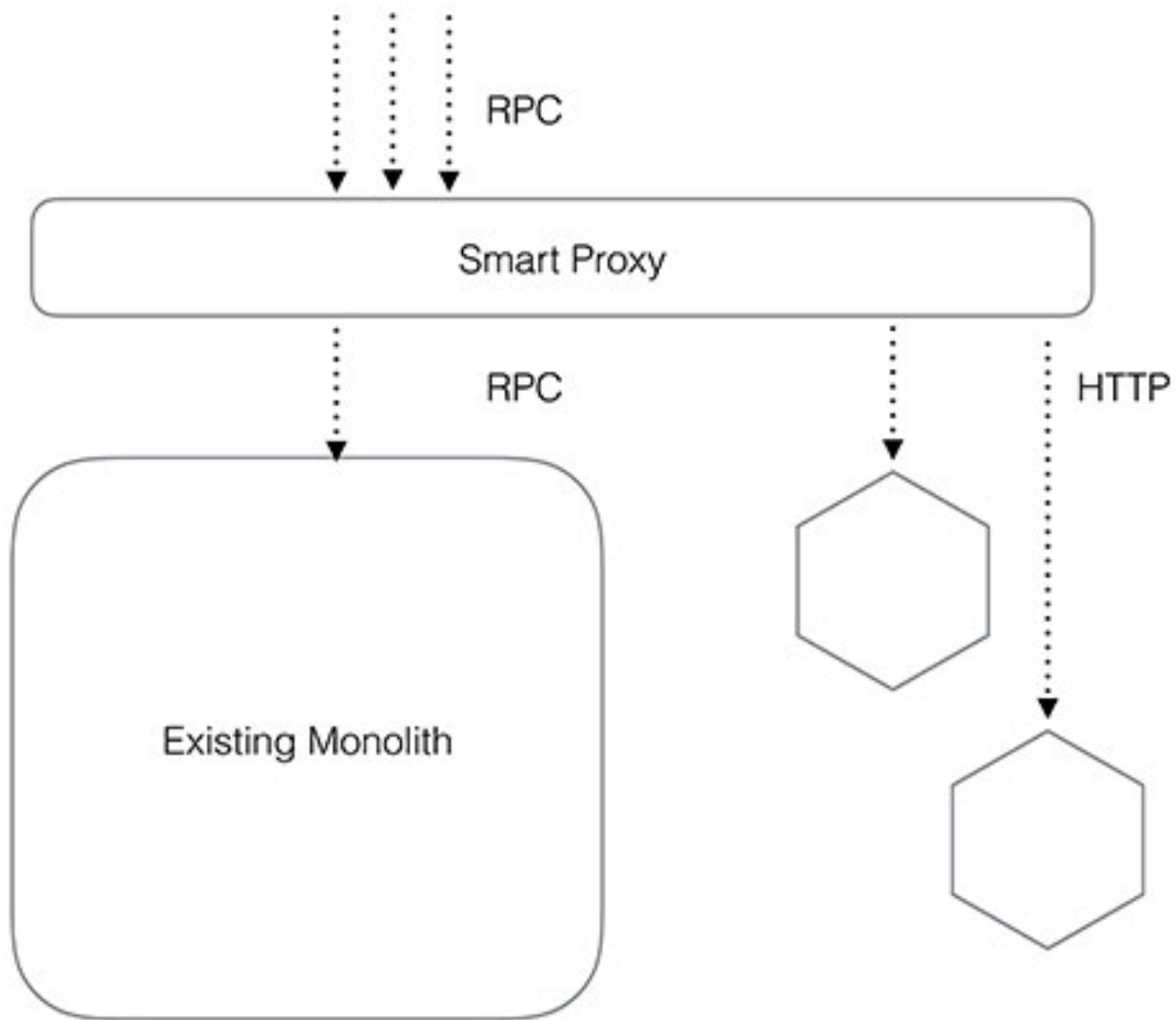
PROTOCOL CHANGING PROXY



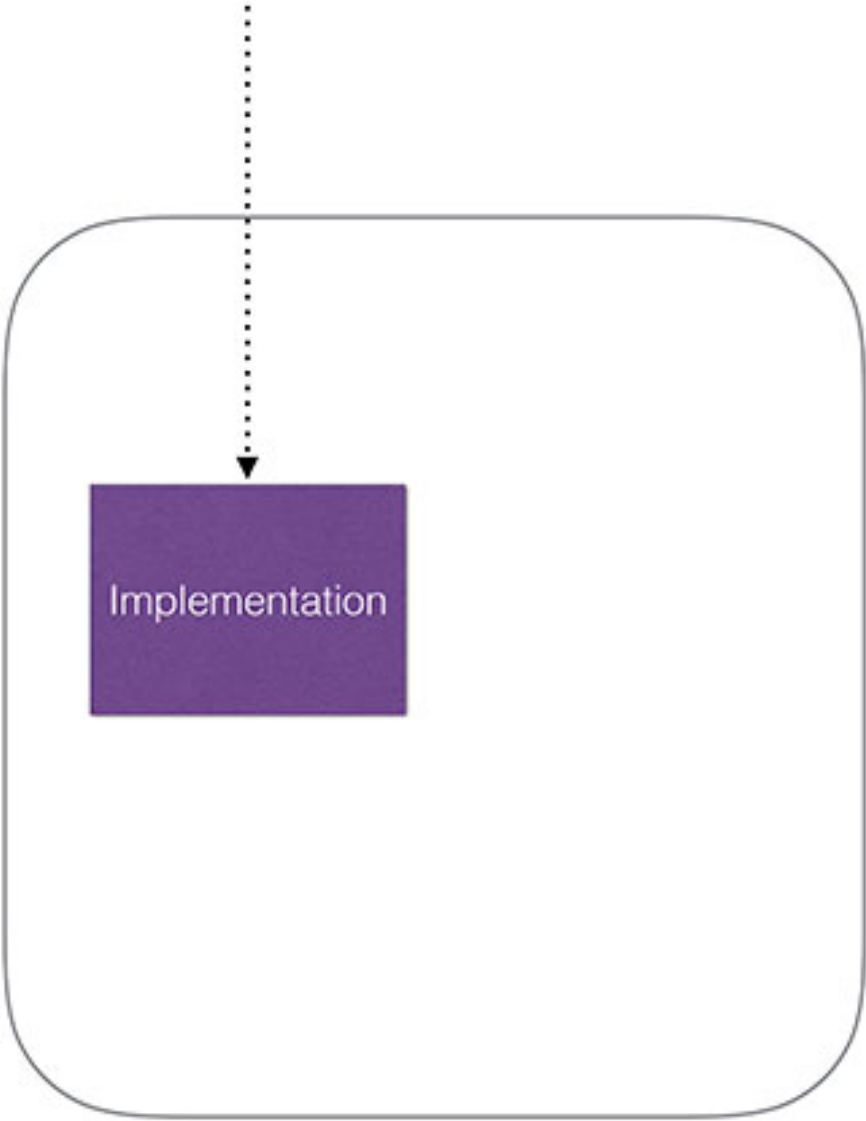
PROTOCOL CHANGING PROXY



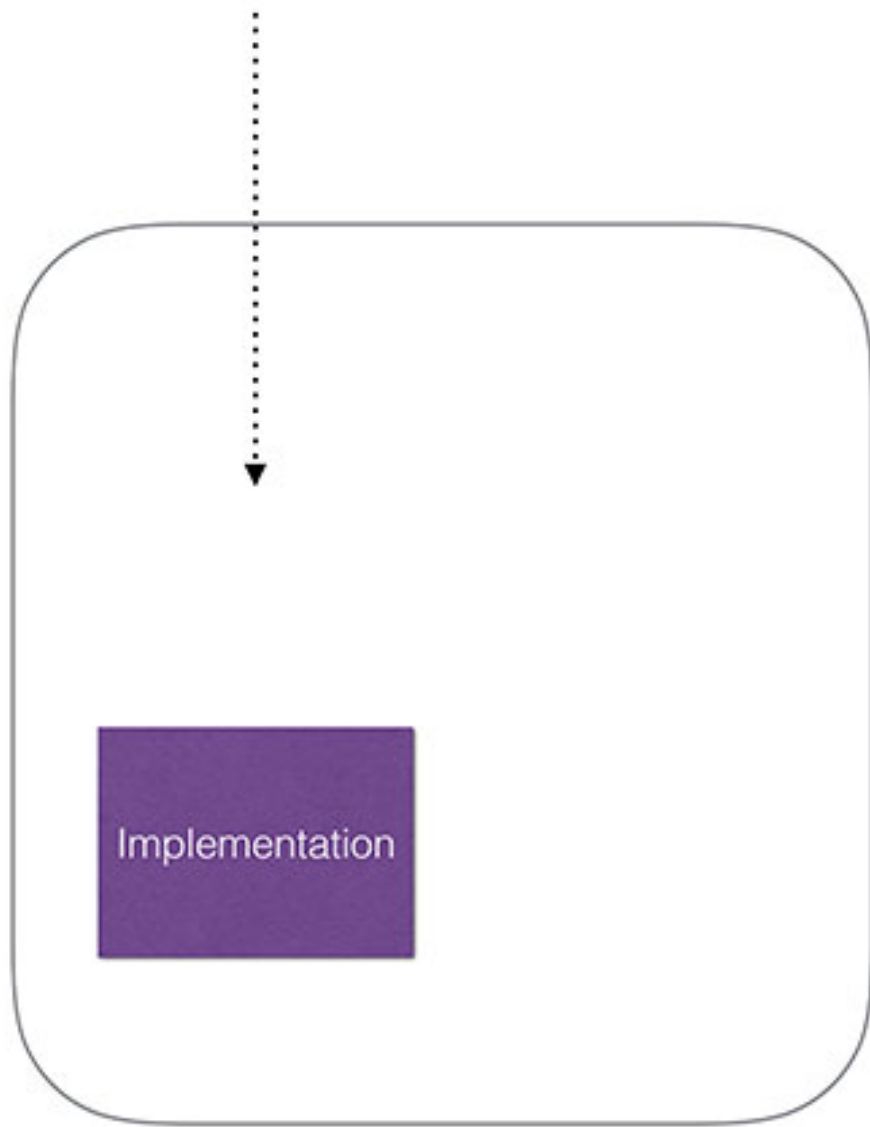
PROTOCOL CHANGING PROXY



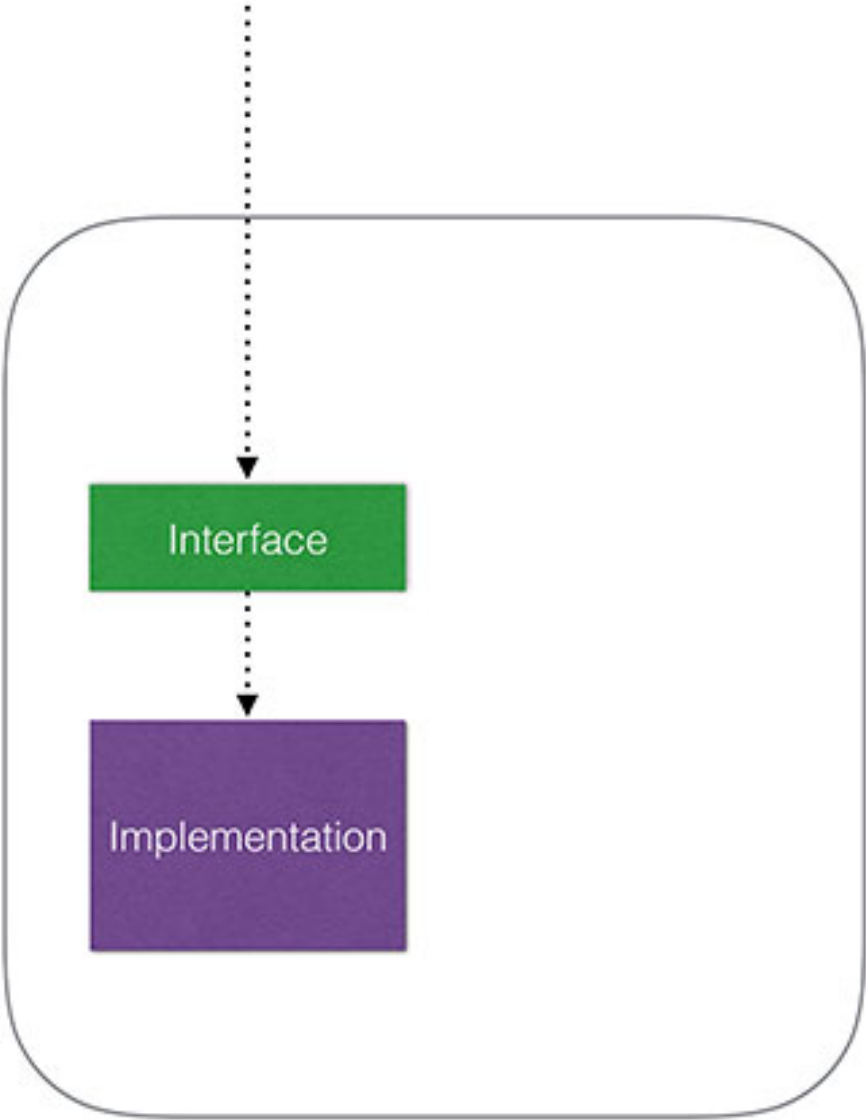
INTERNAL ABSTRACTION



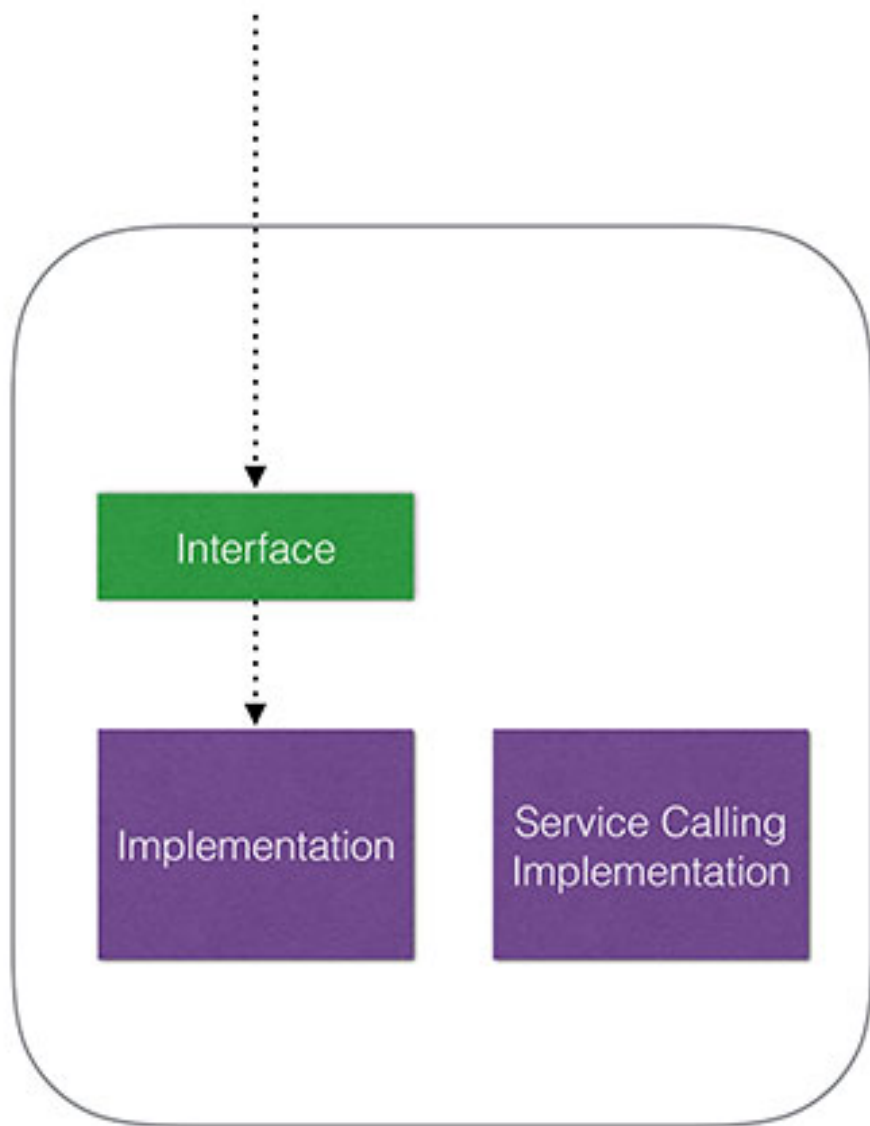
INTERNAL ABSTRACTION



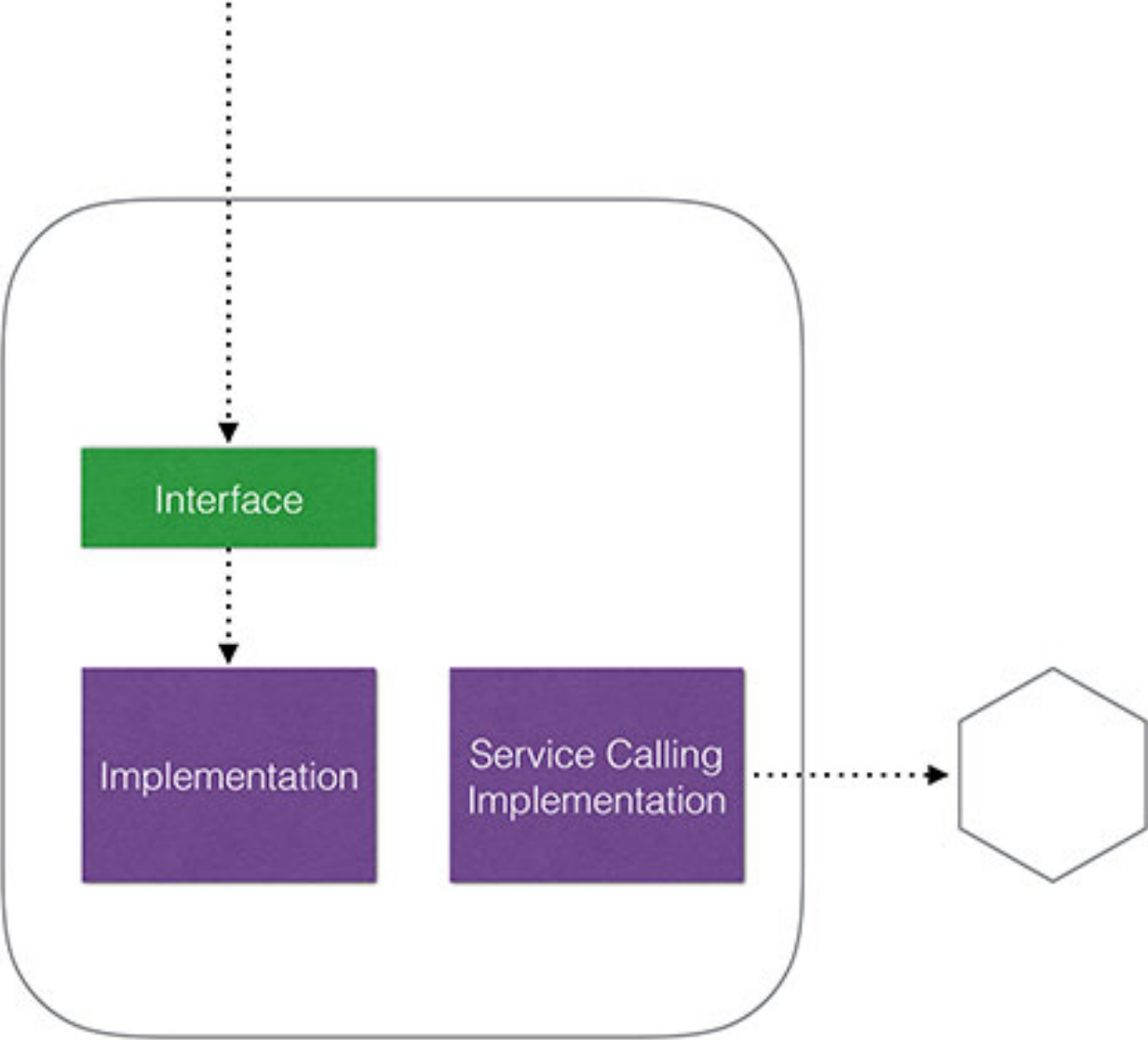
INTERNAL ABSTRACTION



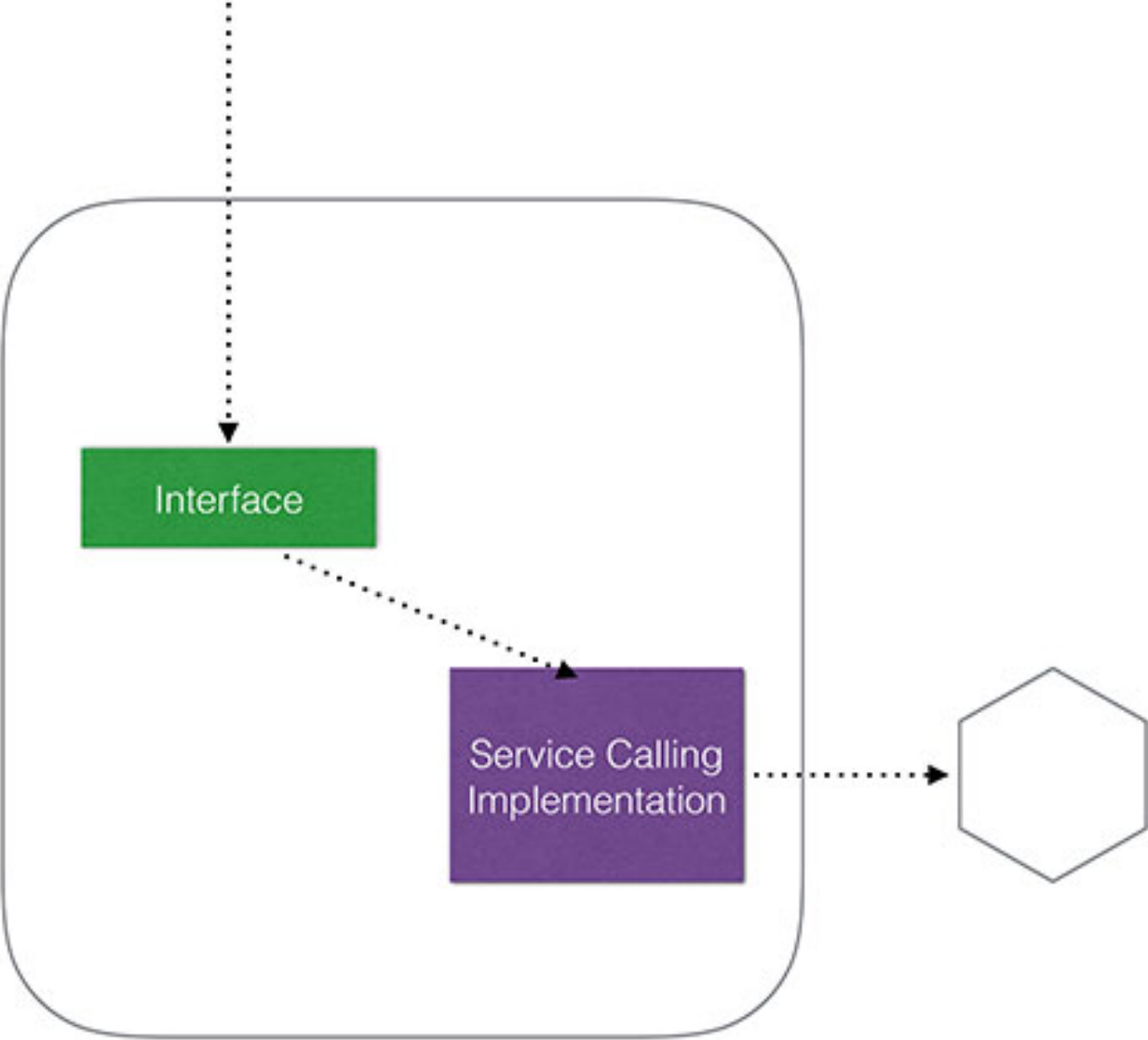
INTERNAL ABSTRACTION



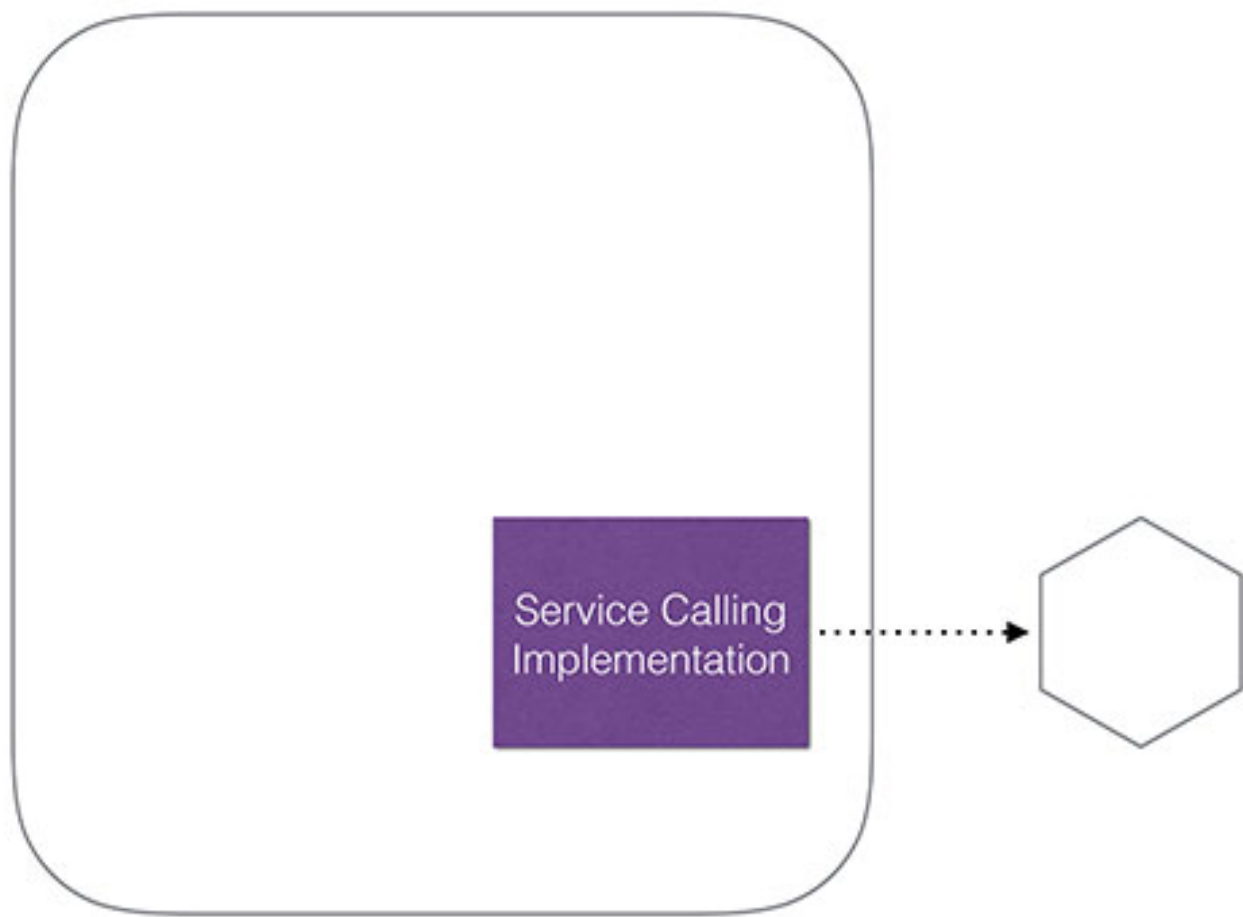
INTERNAL ABSTRACTION



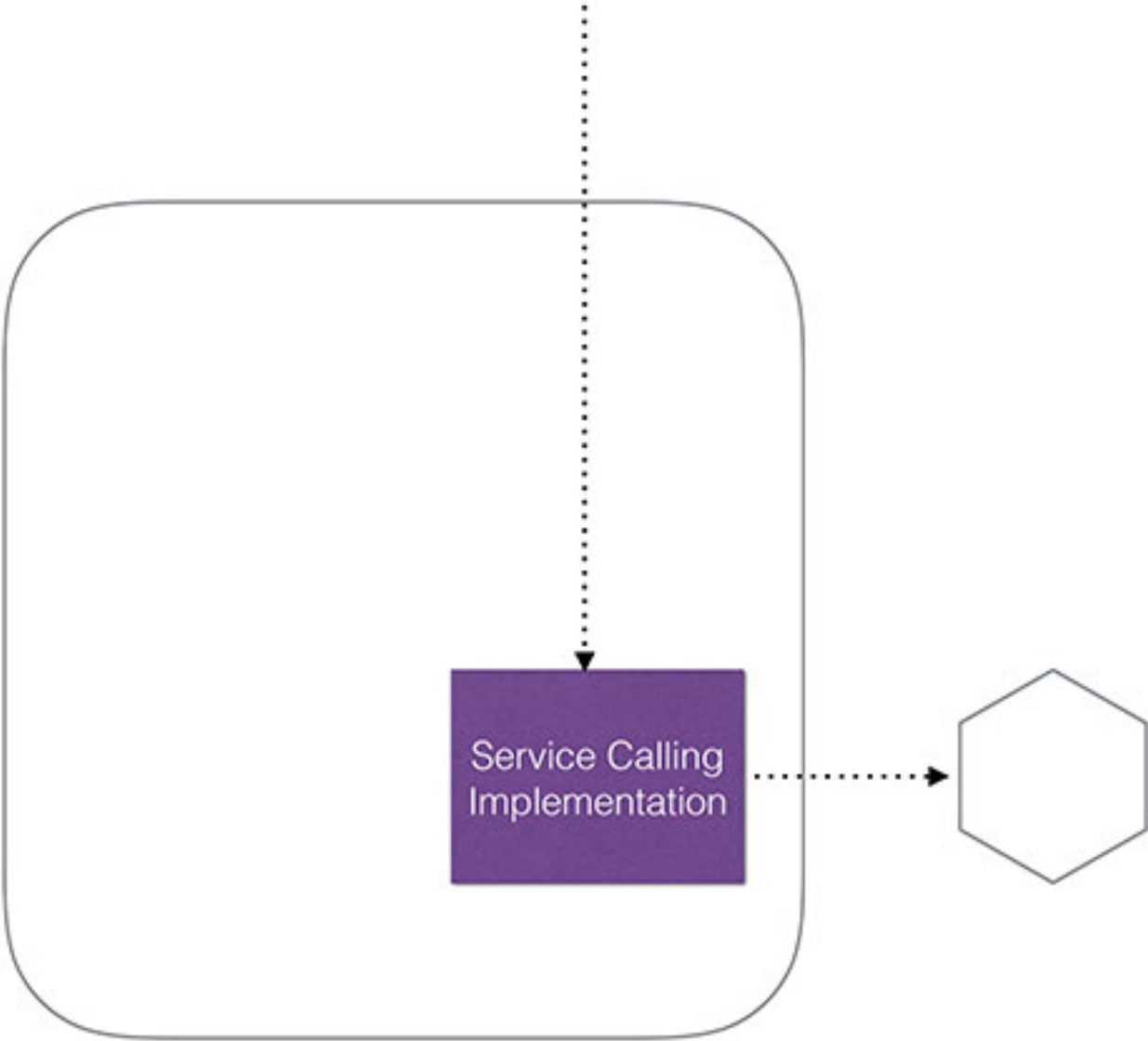
INTERNAL ABSTRACTION



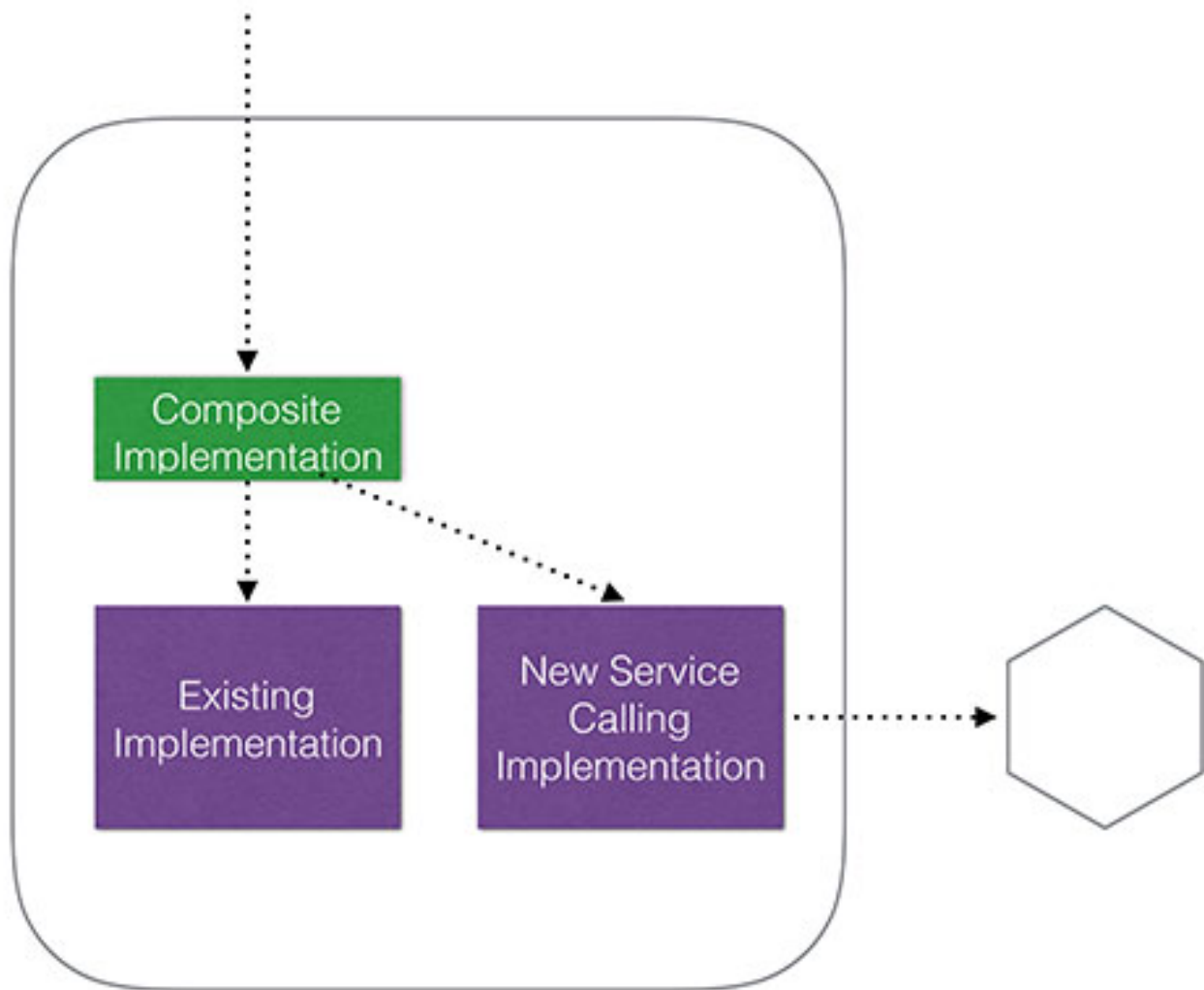
INTERNAL ABSTRACTION



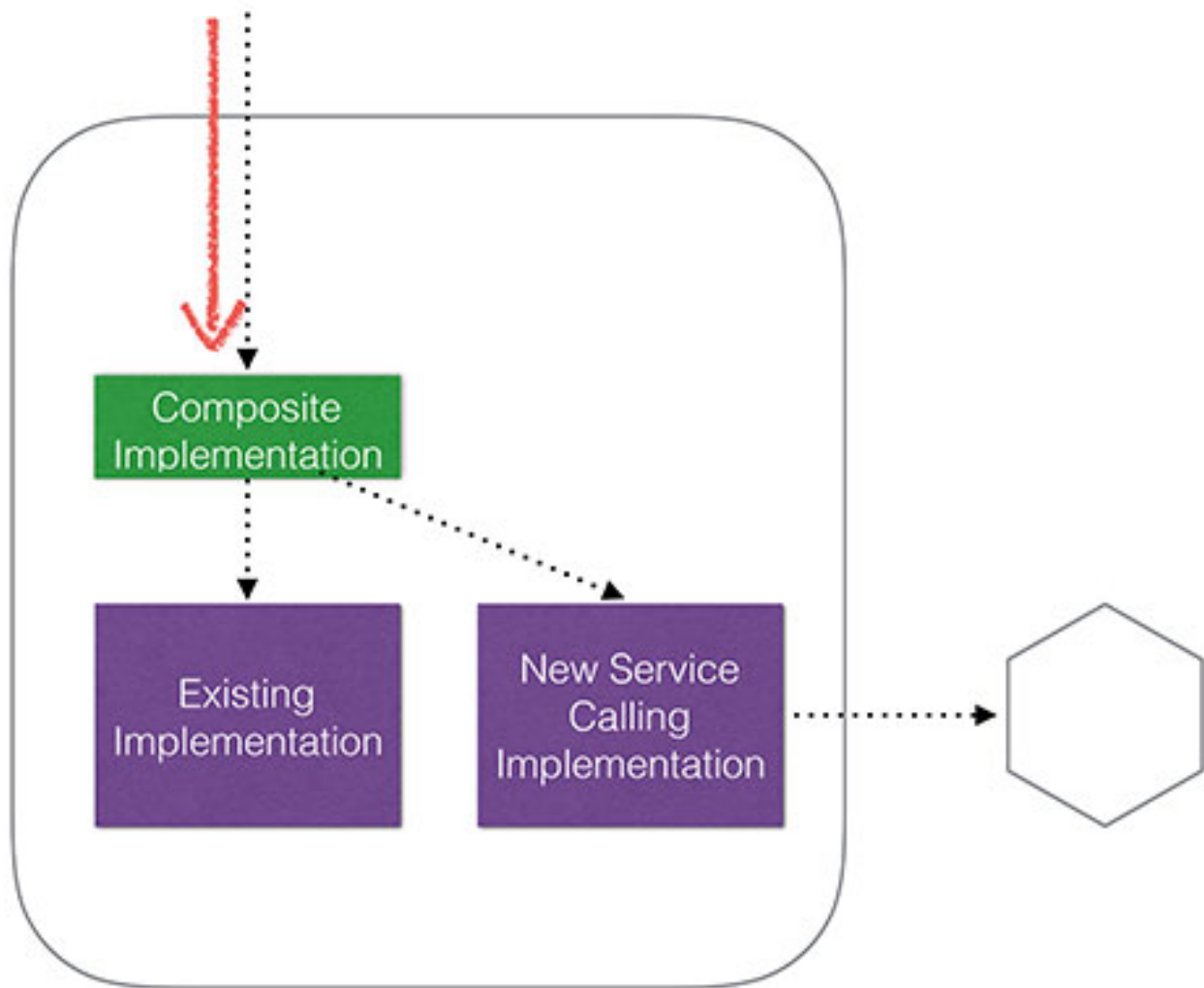
INTERNAL ABSTRACTION



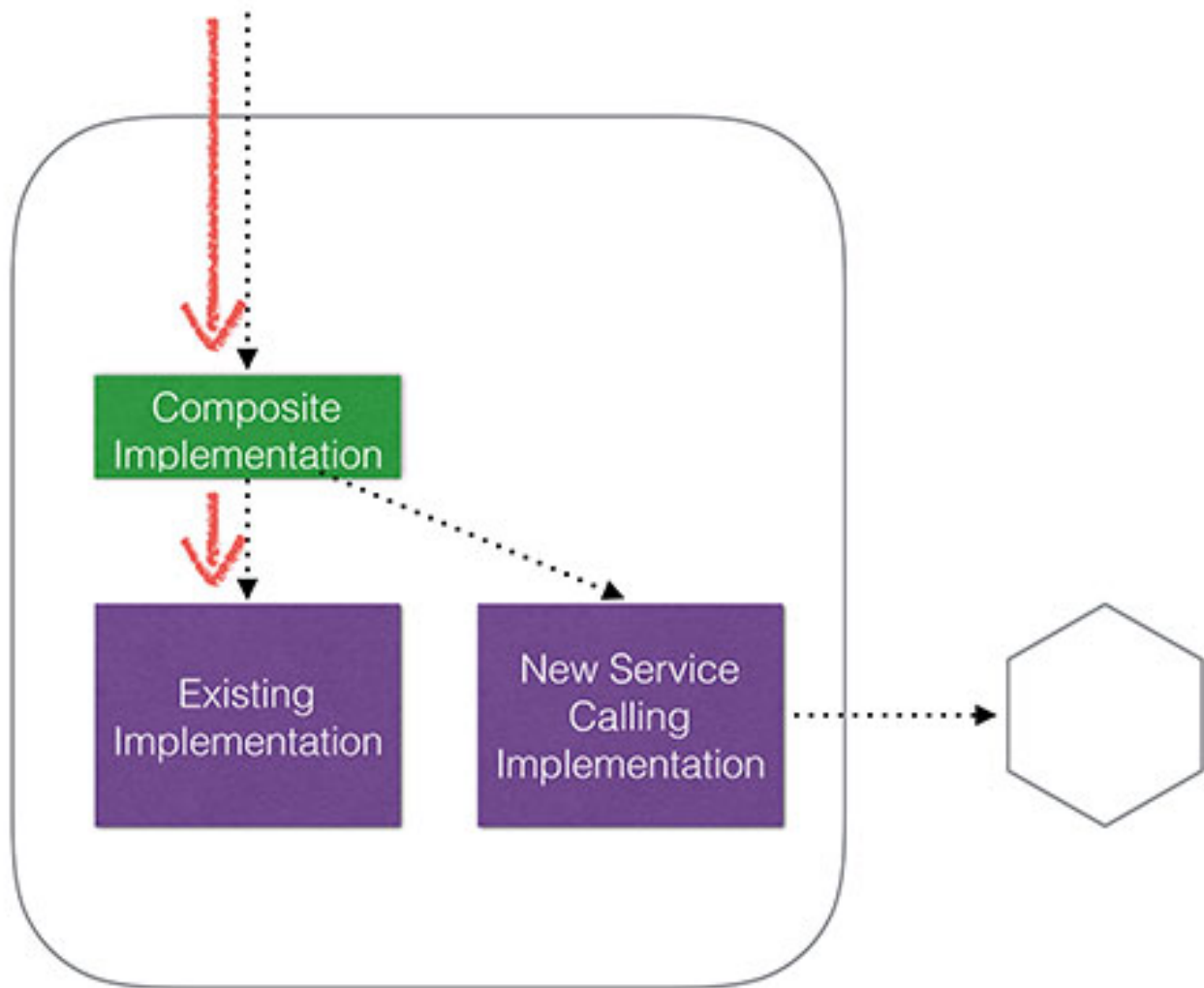
CO-EXISTING INTERNAL ABSTRACTIONS



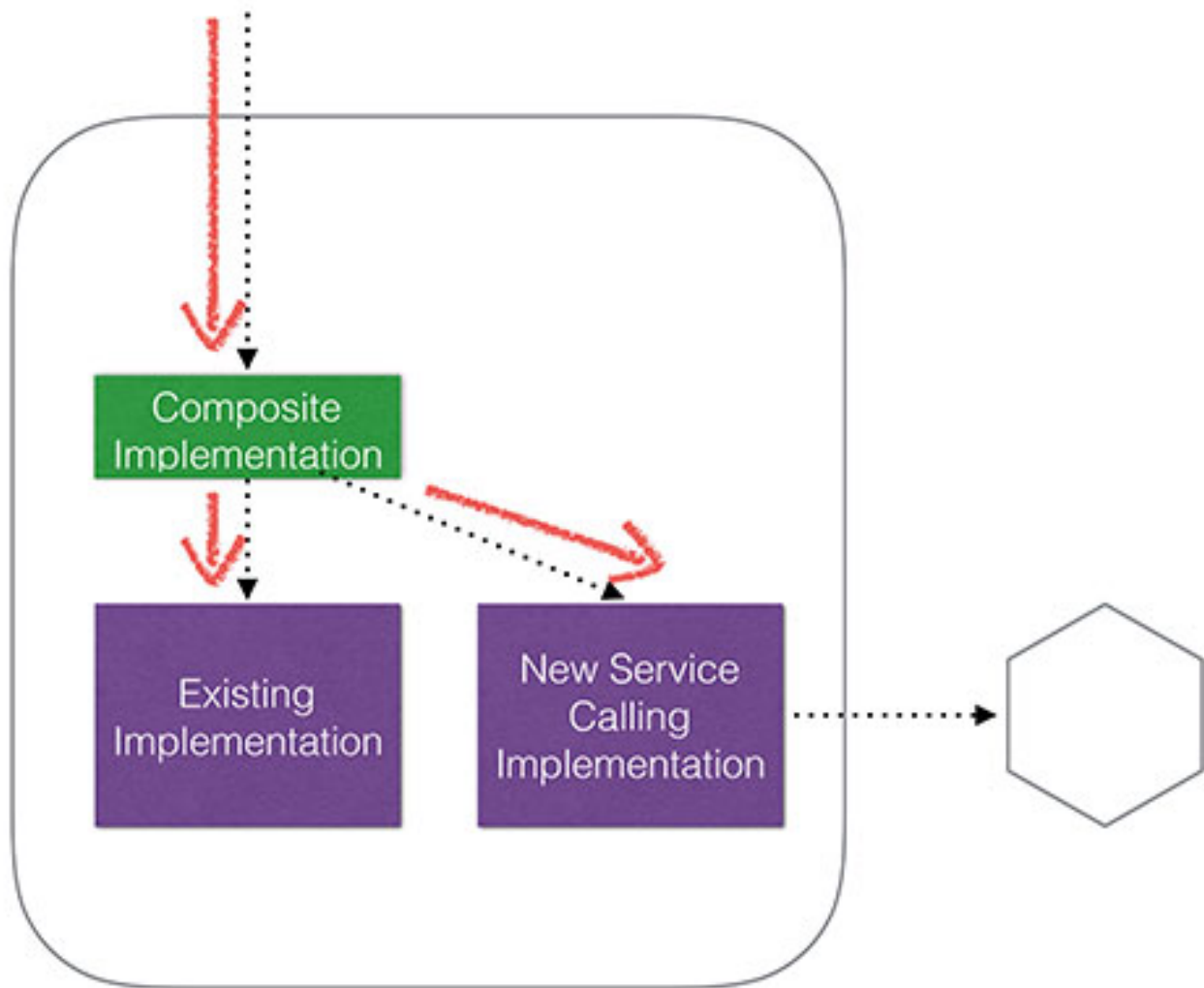
CO-EXISTING INTERNAL ABSTRACTIONS



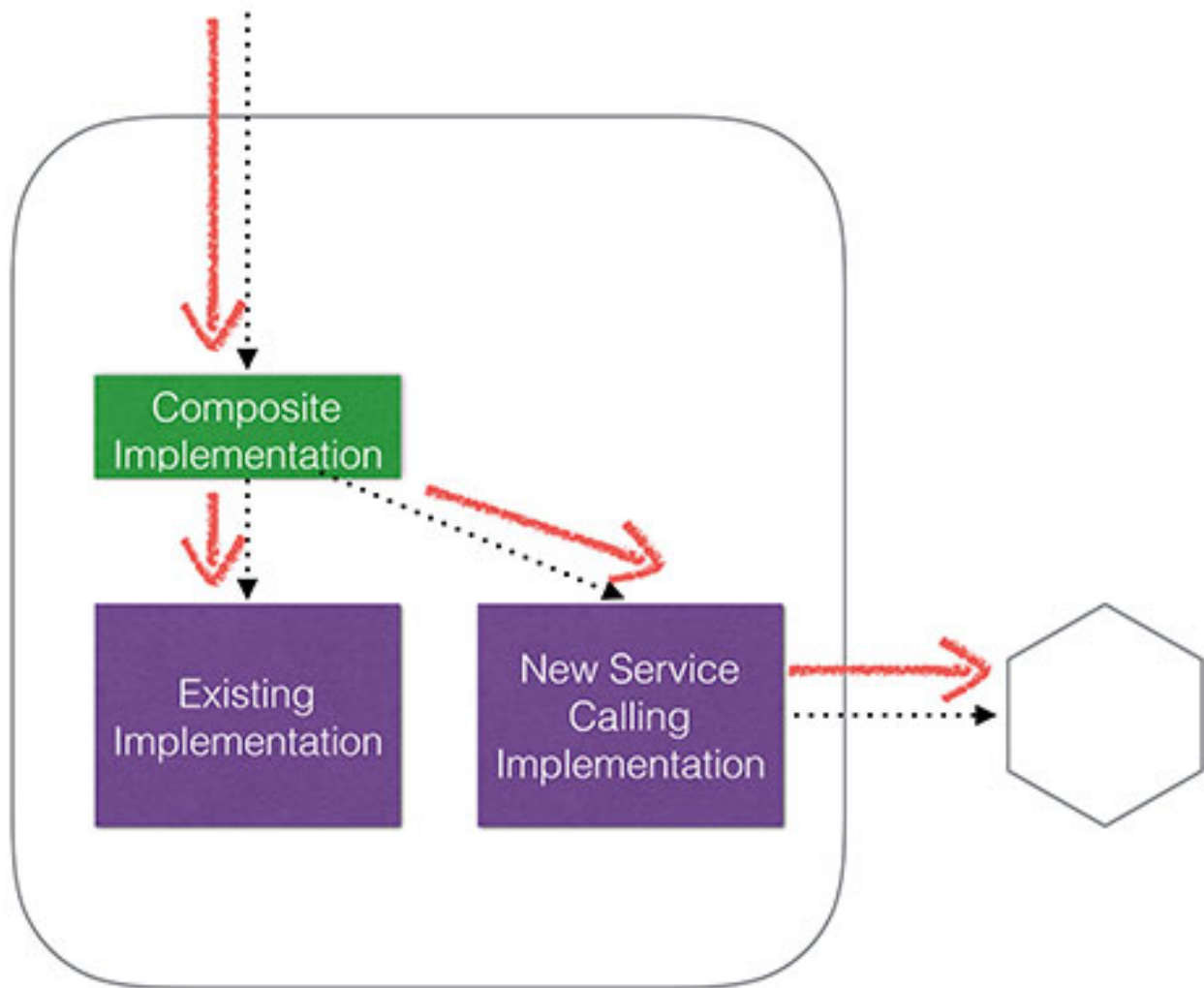
CO-EXISTING INTERNAL ABSTRACTIONS



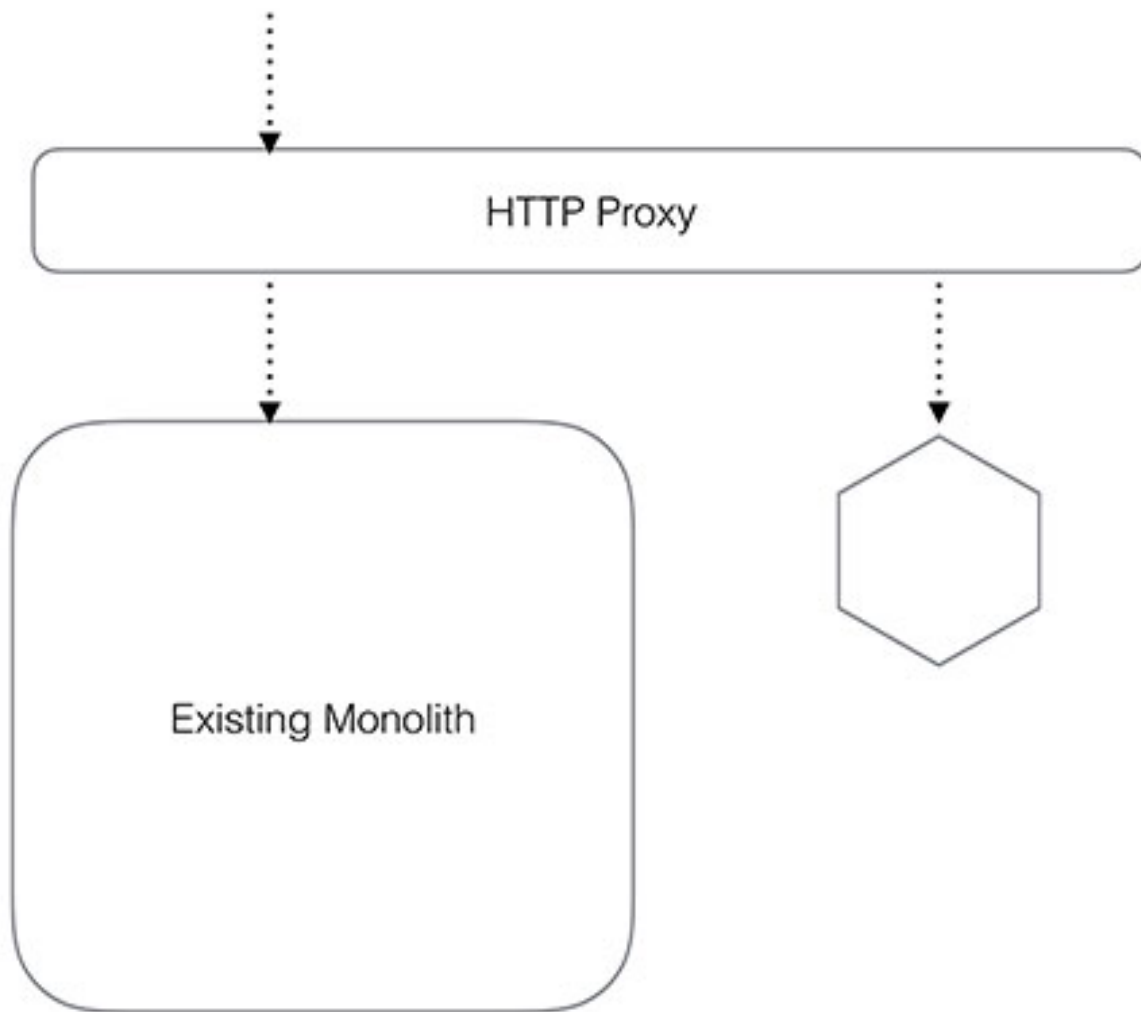
CO-EXISTING INTERNAL ABSTRACTIONS



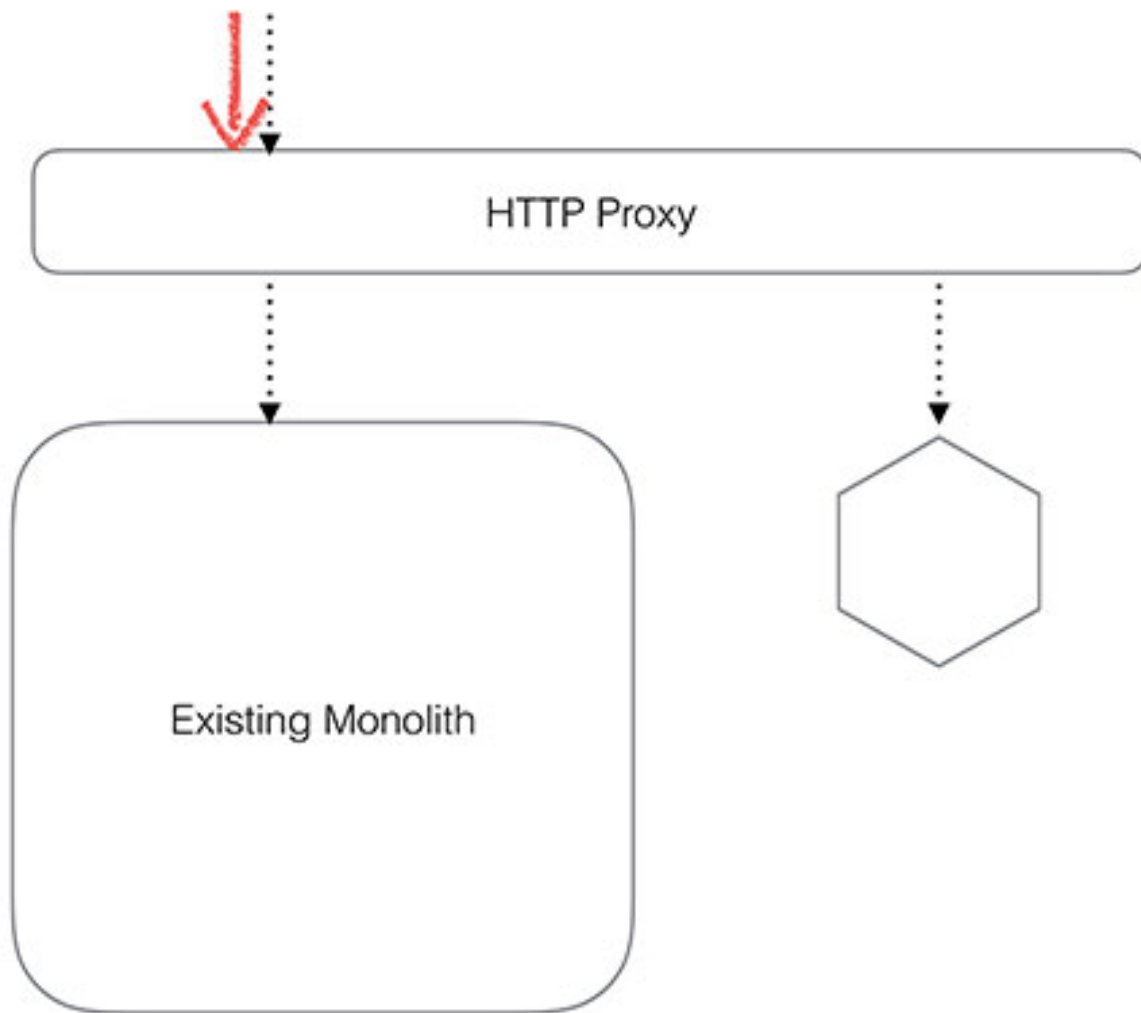
CO-EXISTING INTERNAL ABSTRACTIONS



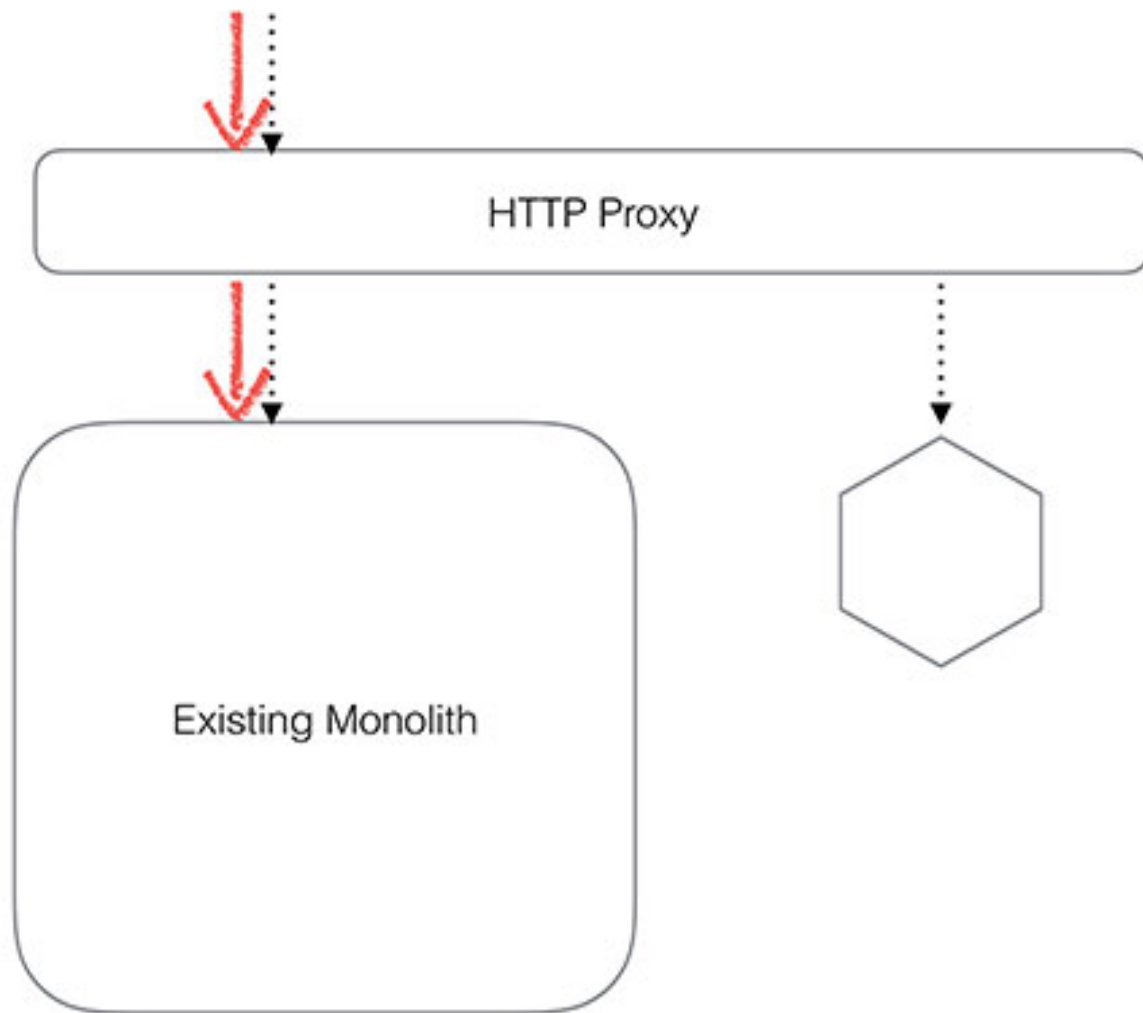
CO-EXISTING PROXY IMPLEMENTATIONS



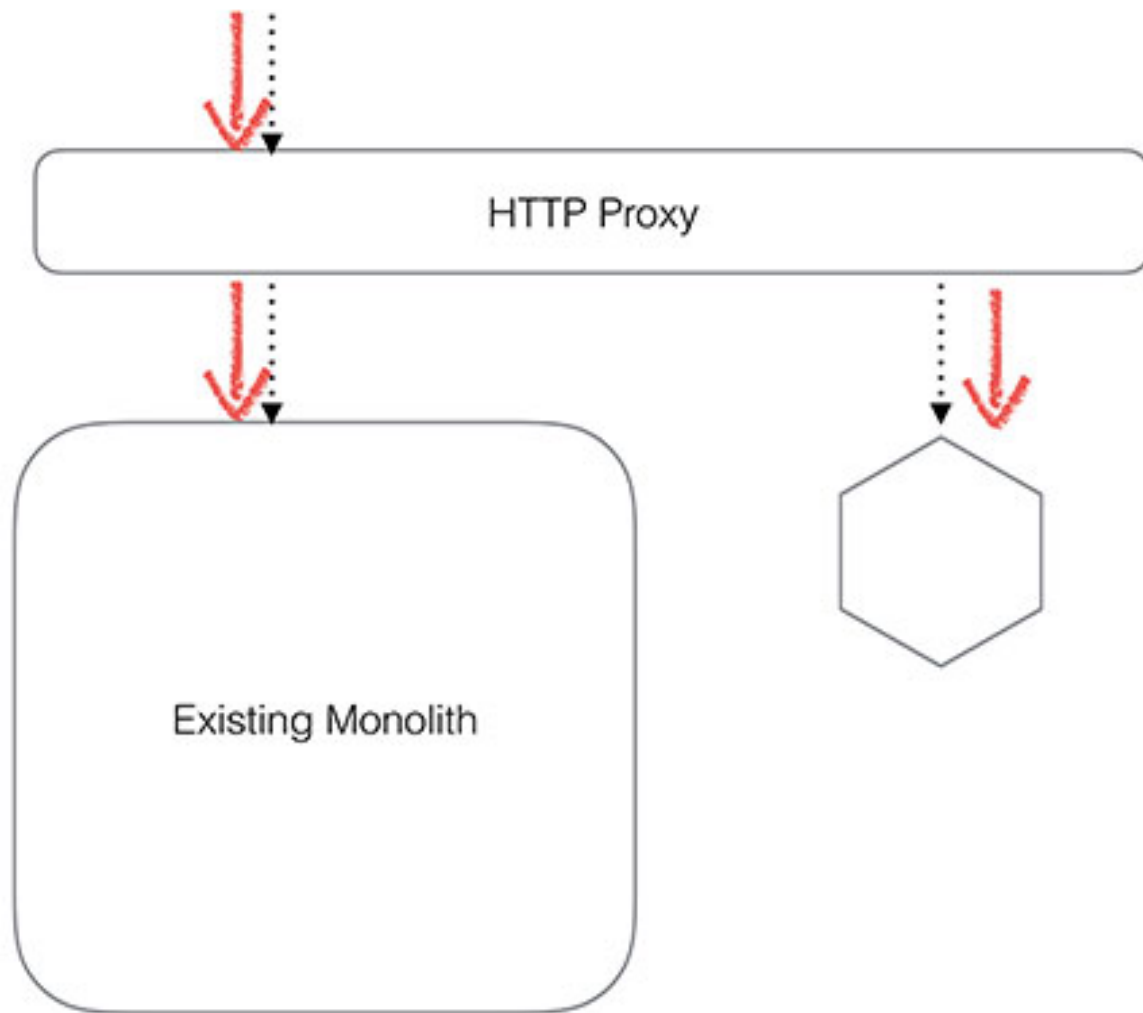
CO-EXISTING PROXY IMPLEMENTATIONS



CO-EXISTING PROXY IMPLEMENTATIONS



CO-EXISTING PROXY IMPLEMENTATIONS

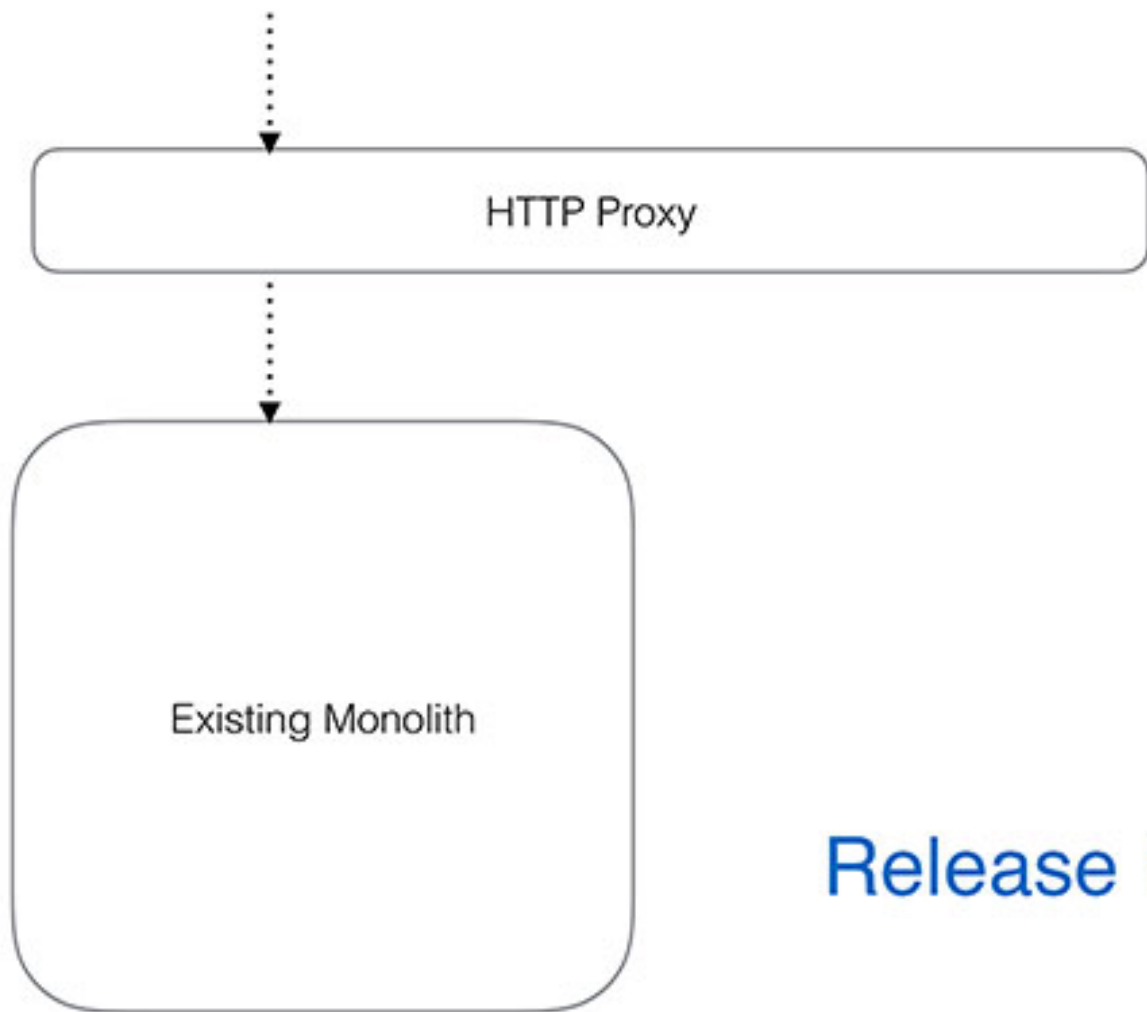


DARK LAUNCHING



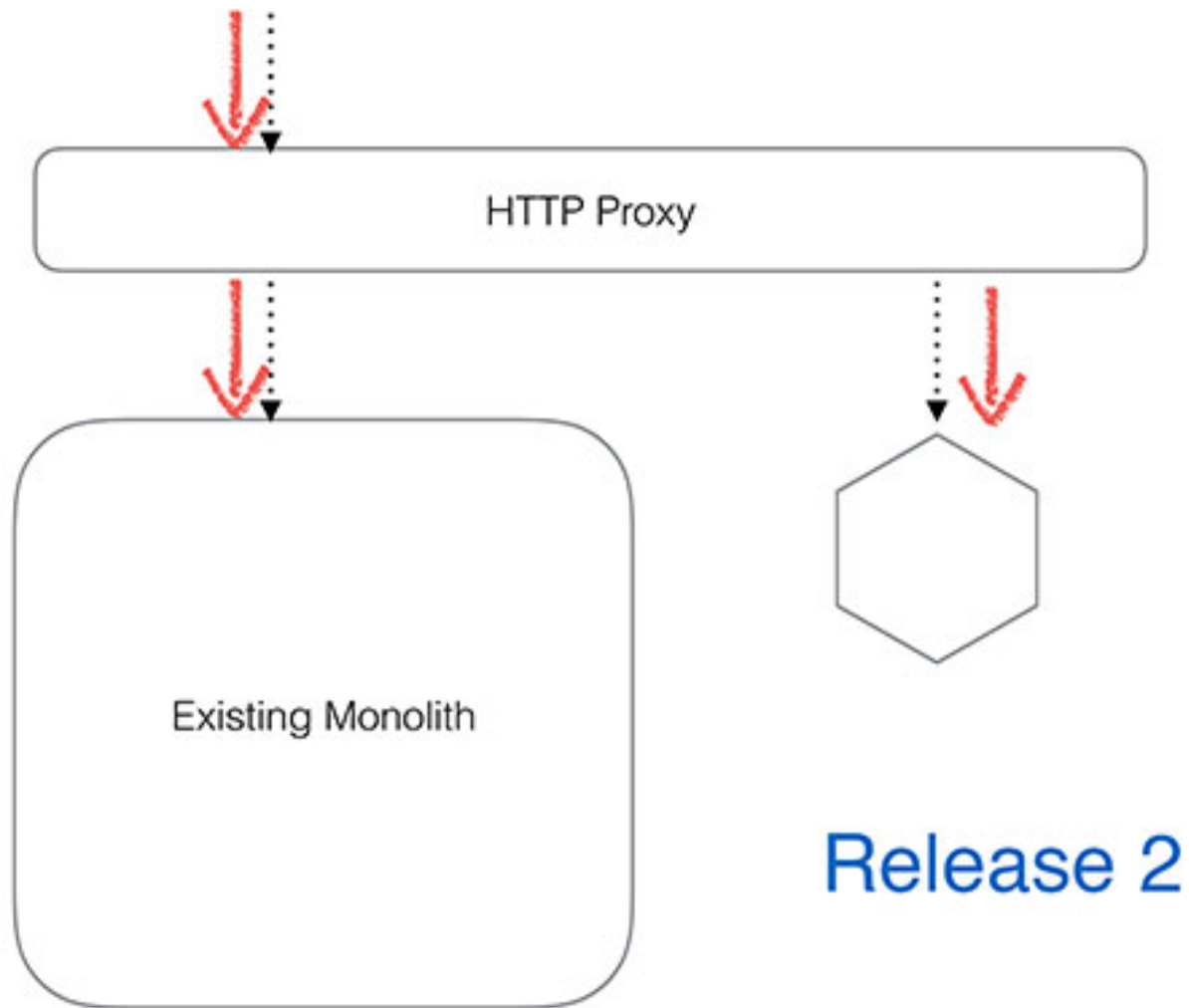
Existing Monolith

DARK LAUNCHING

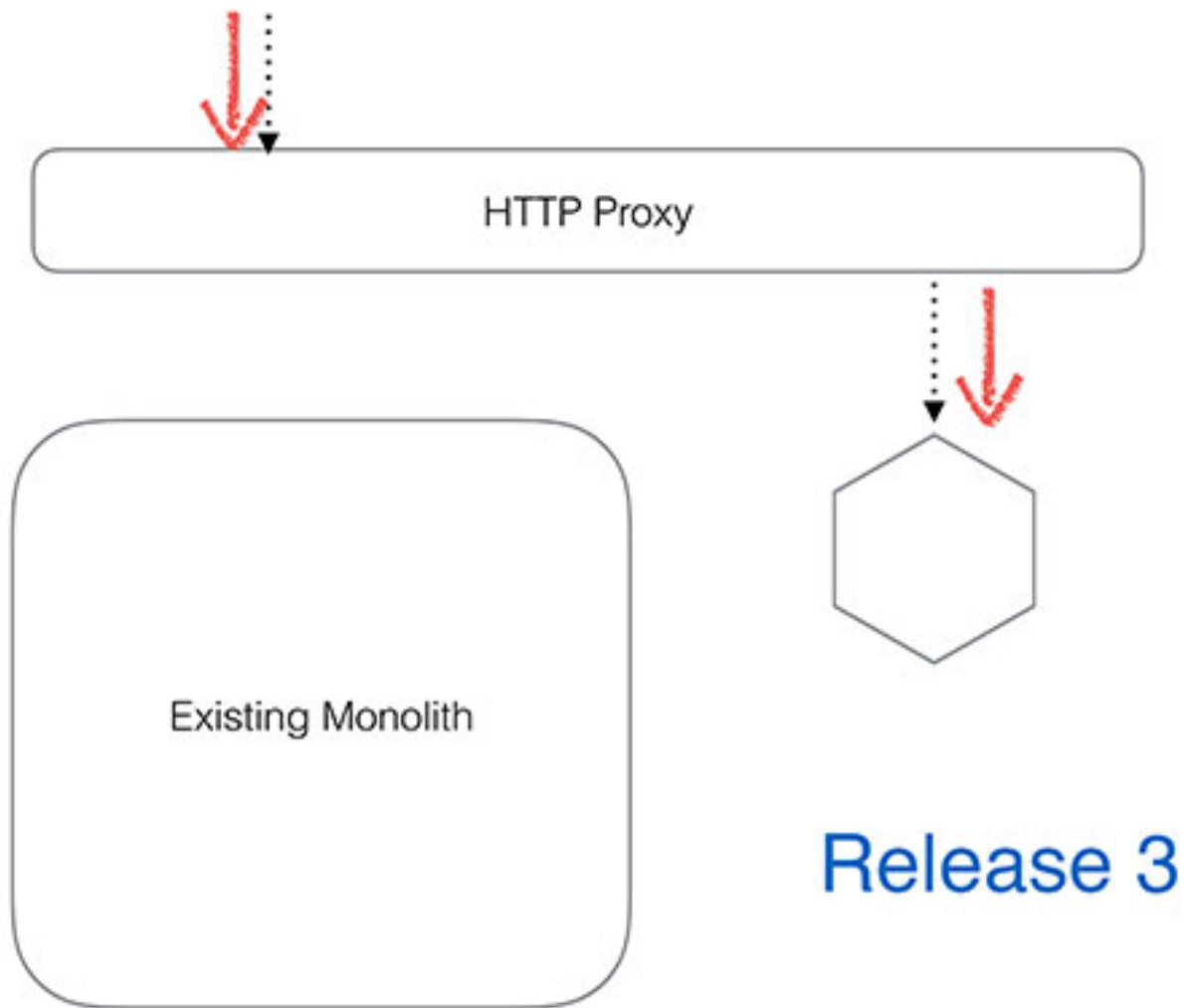


Release 1

DARK LAUNCHING



DARK LAUNCHING



Call both implementations & compare

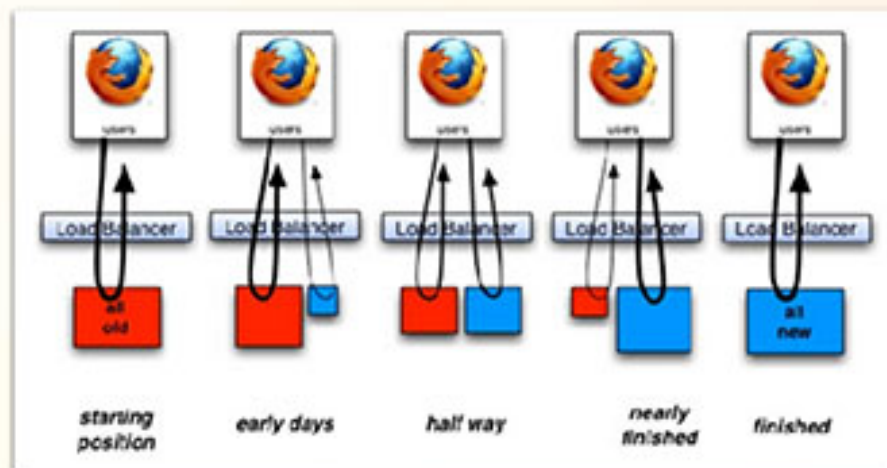
Divert proportion of traffic to test new
implementation (canary release)

Legacy Application Strangulation : Case Studies

Strangler Applications

Martin Fowler wrote an [article](#) titled "Strangler Application" in mid 2004.

Strangulation of a legacy or undesirable solution is a safe way to phase one thing out for something better, cheaper, or more expandable. You make something new that obsoletes a small percentage of something old, and put them live together. You do some more work in the same style, and go live again (rinse, repeat). Here's a view of that (for web-apps):



You could migrate all functionality from an old technology solution to new one in a series of releases that focussed on nothing else. Some companies will do that as there is a lot of sense to getting your house in order before doing anything else. However people outside the developer team may see that as a non-productive period, that could lengthen at any time, if it were asked for at all. People paying for that will notice, and may object. I mean execs, the board, or shareholders looking at the balance sheet.

<http://paulhammant.com/2013/07/14/legacy-application-strangulation-case-studies/>

Databases

MusikShopMono



Database

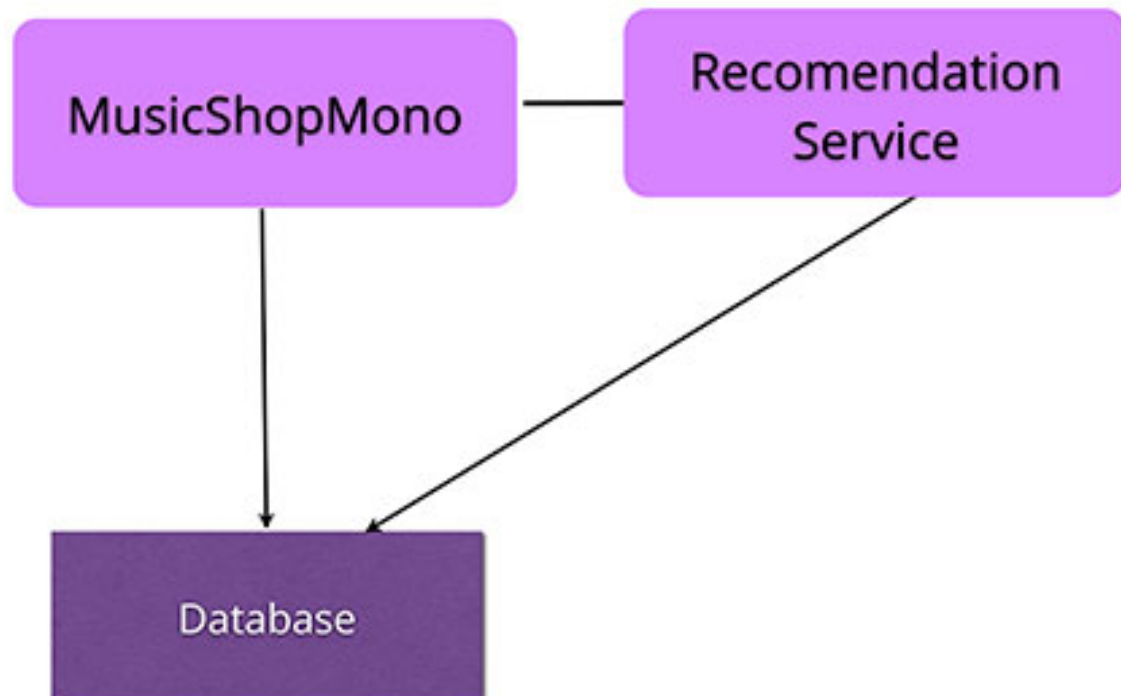
MusicShopMono

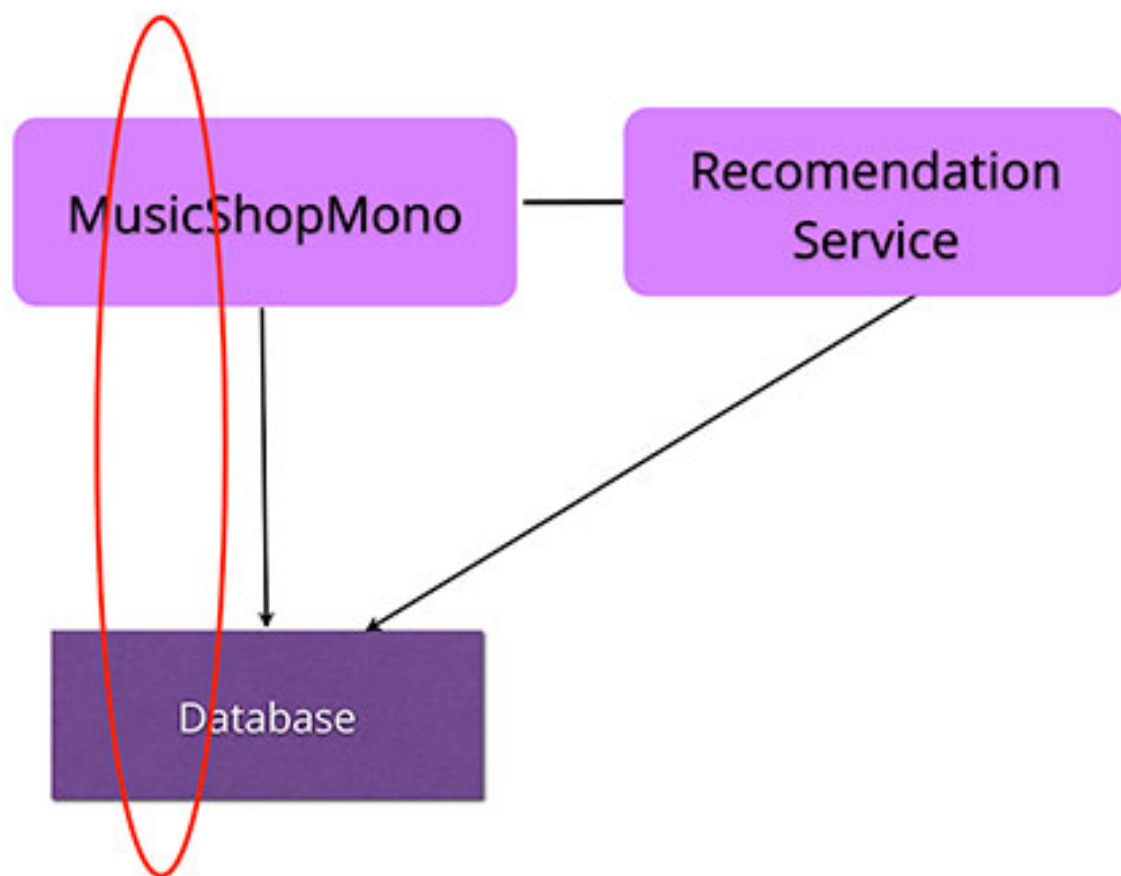


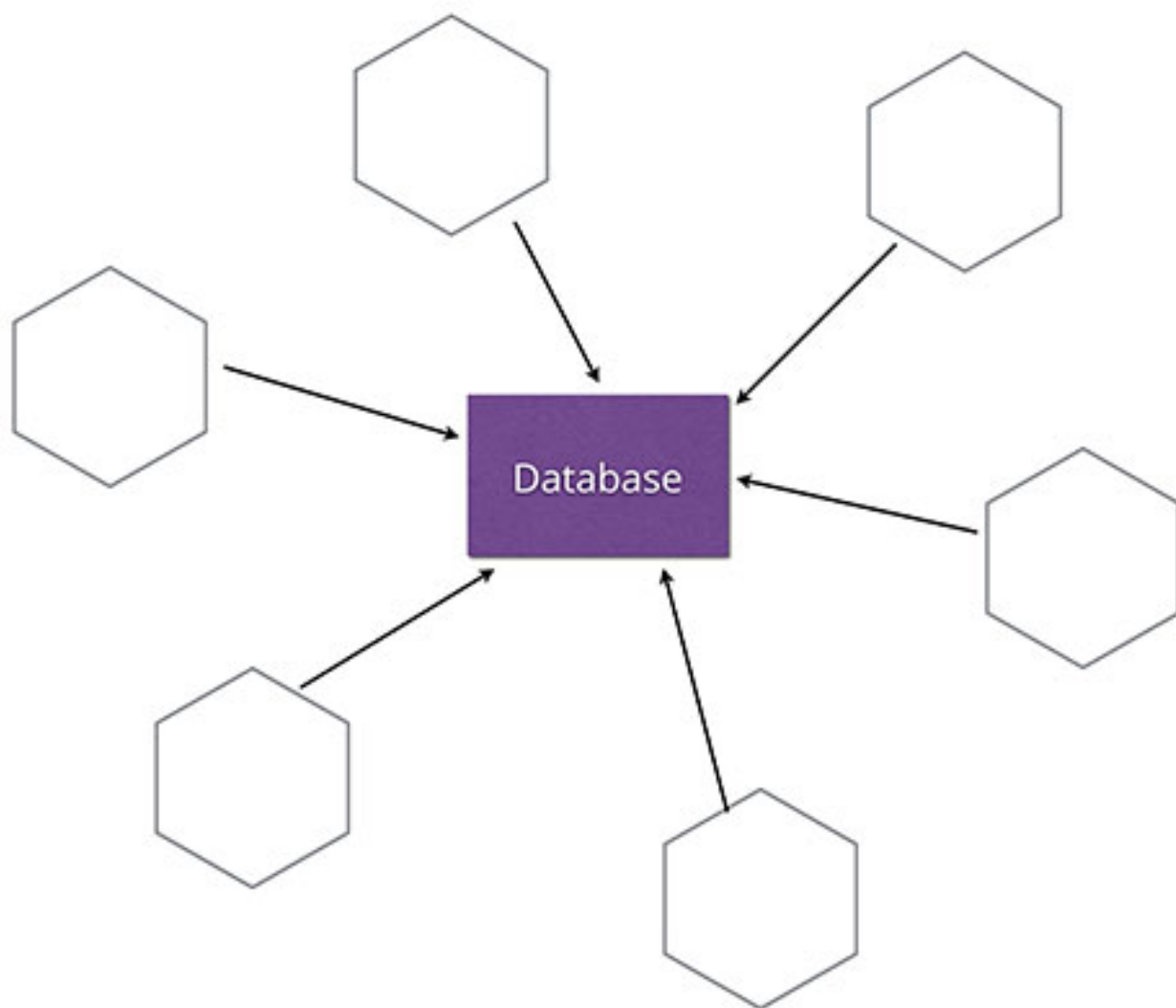
```
graph TD; MusicShopMono[MusicShopMono] --> Database[Database];
```

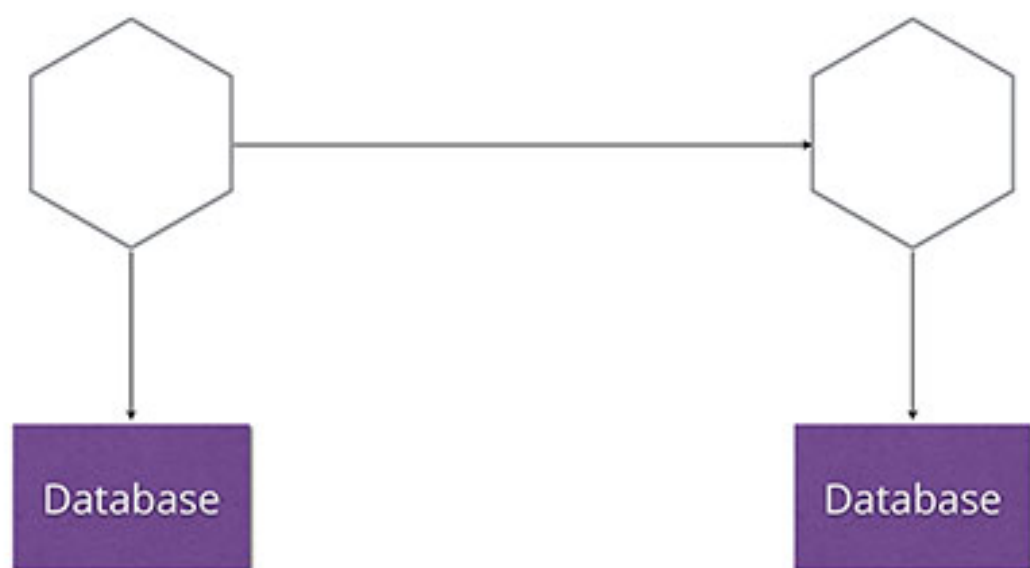
The diagram consists of two rectangular boxes. The top box is light purple with rounded corners and contains the text 'MusicShopMono'. The bottom box is a darker purple with sharp corners and contains the text 'Database'. A black arrow points vertically from the bottom center of the top box to the top center of the bottom box.

Database

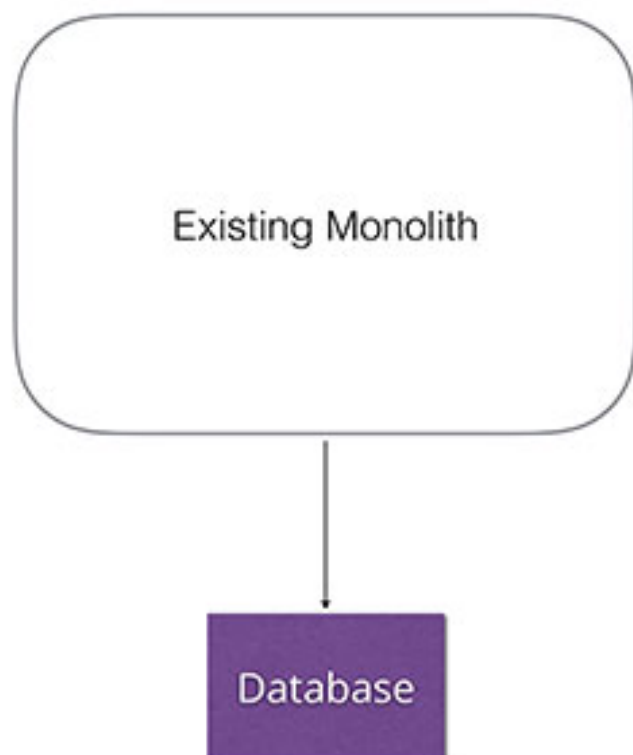




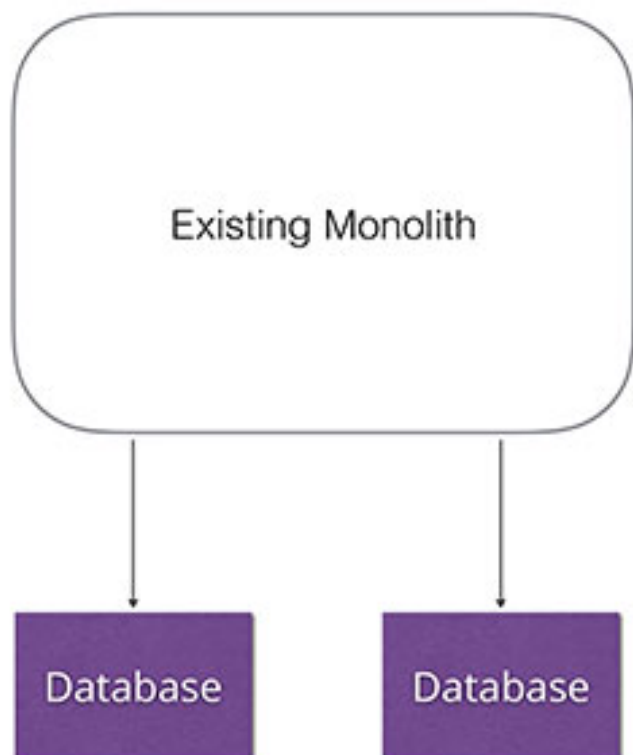




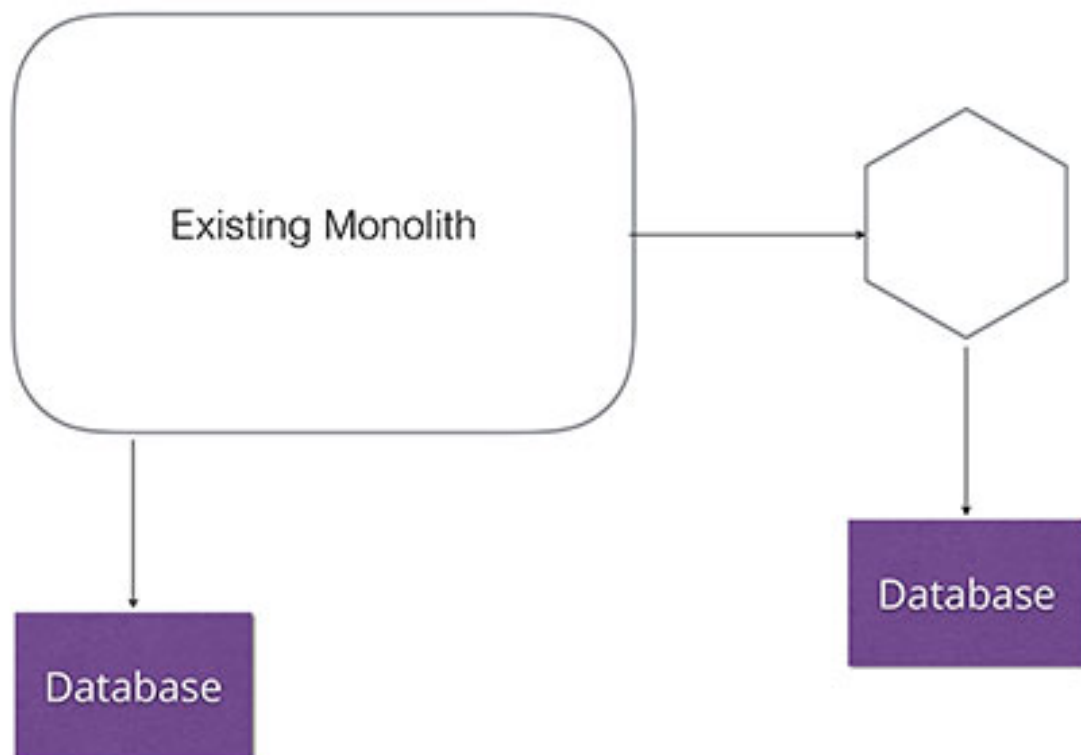
SPLIT DATABASES FIRST

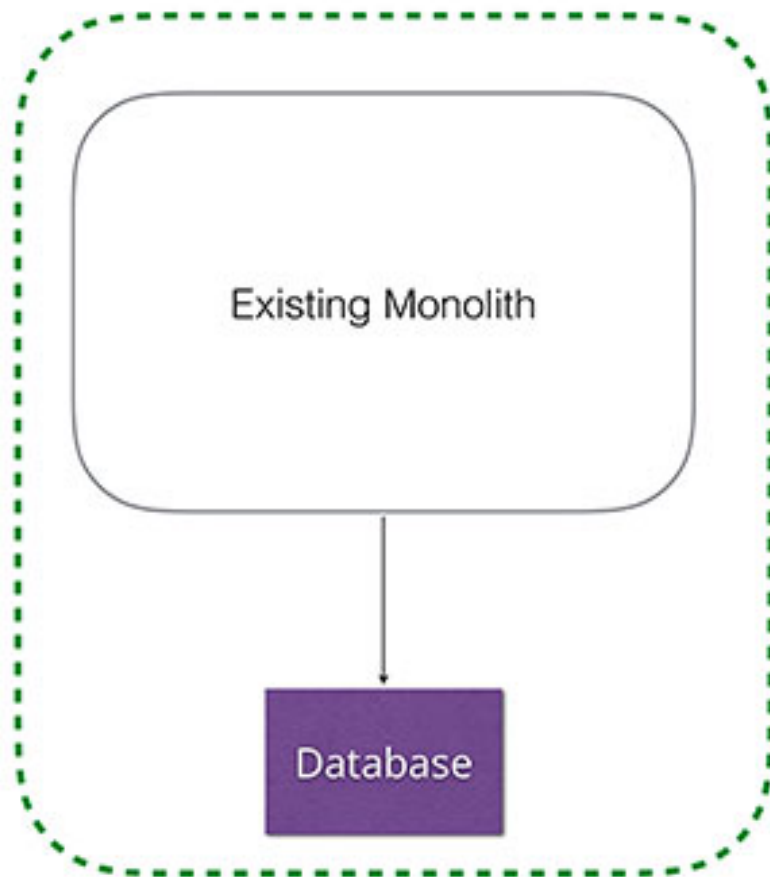


SPLIT DATABASES FIRST



SPLIT DATABASES FIRST

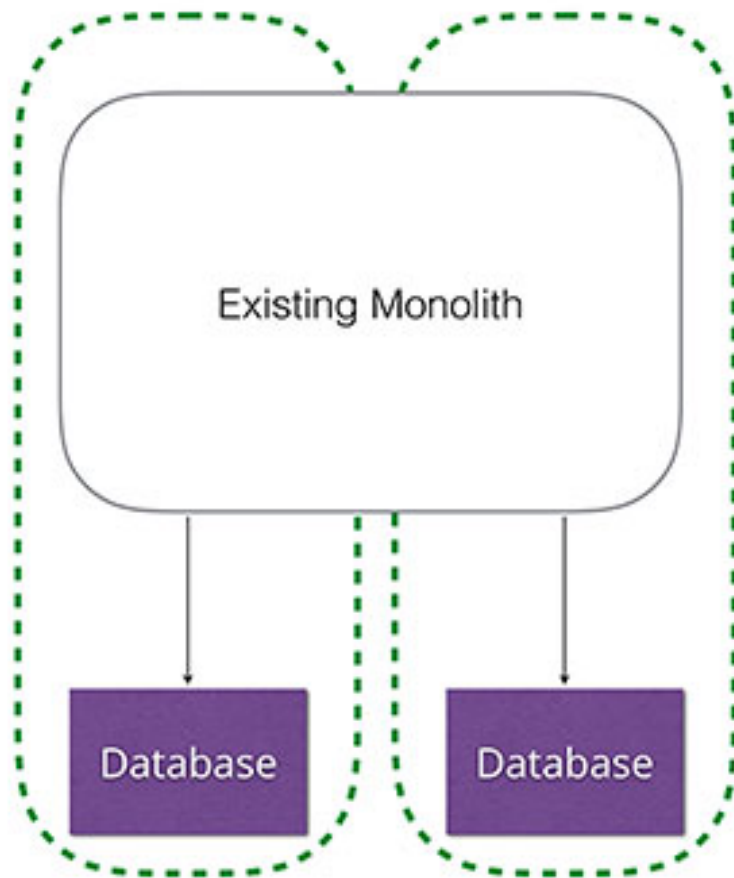




One Transactional
Boundary

Single Database Call

Joins done in the DB



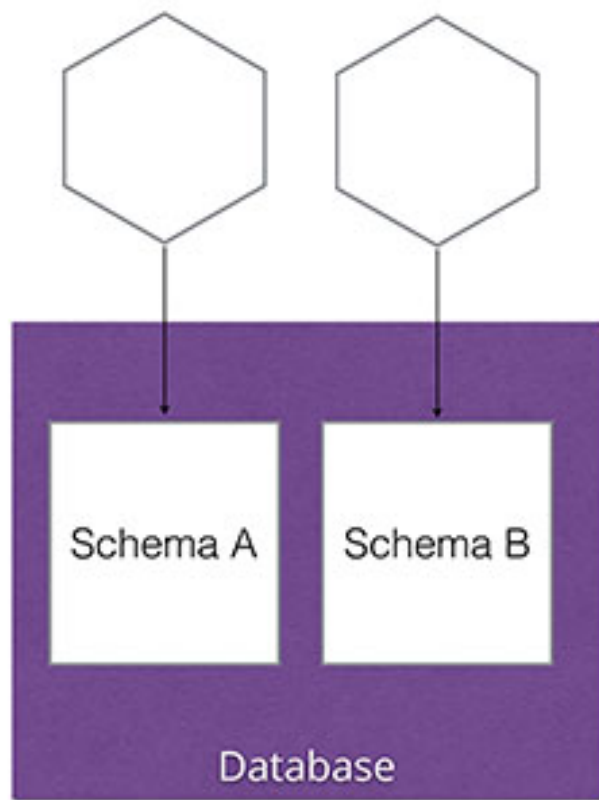
Broke transactional integrity

Two database calls

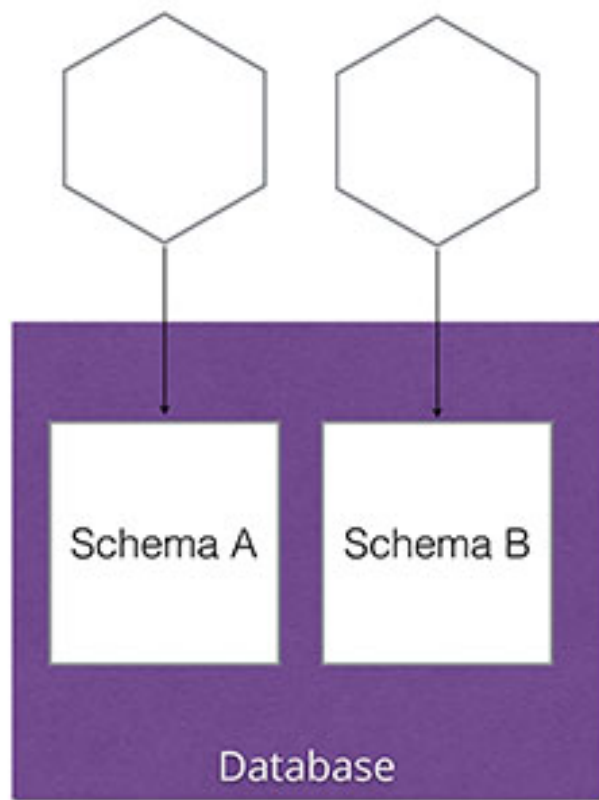
Some joins now done in application code

Understand the performance and transactional integrity issues early

SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT

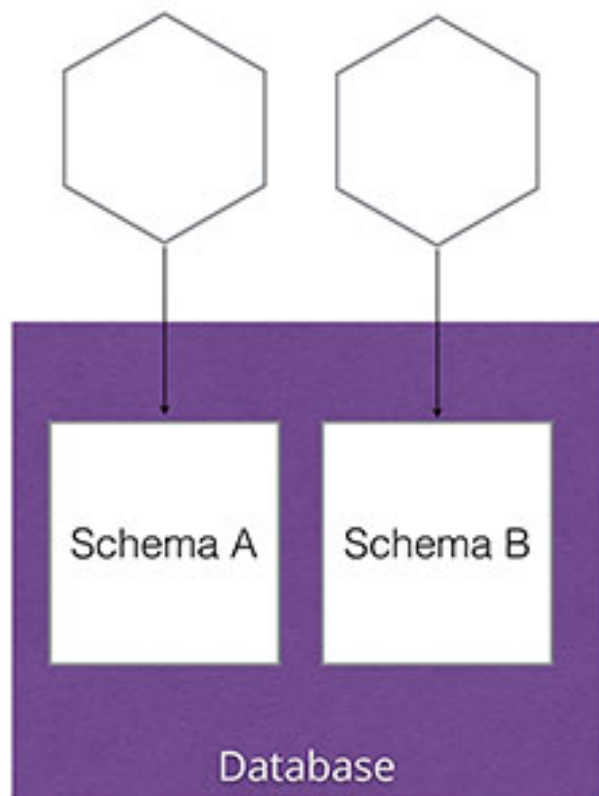


SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT



Logically decoupled

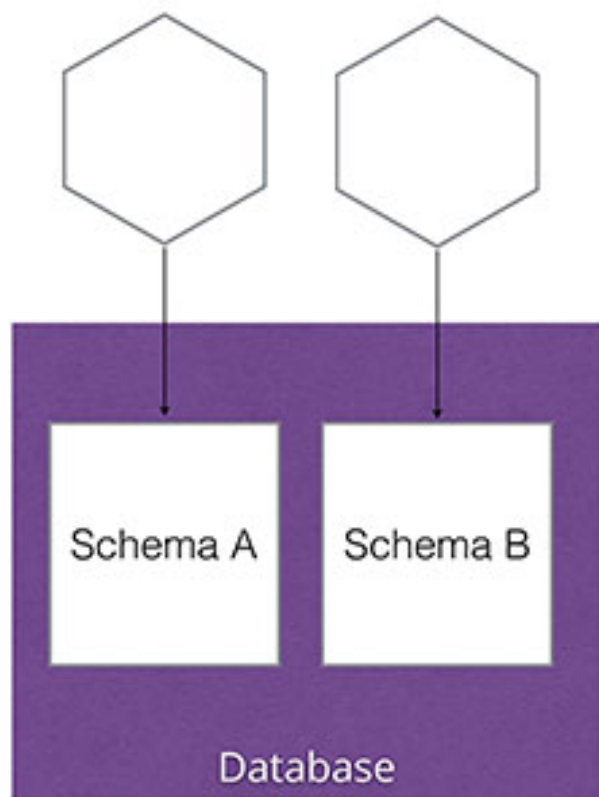
SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT



Logically decoupled

Little or no extra
infrastructure required

SEPARATE SCHEMAS, SAME UNDERLYING DB DEPLOYMENT



Logically decoupled

Little or no extra
infrastructure required

Potential single point of
failure

DB Refactoring Patterns

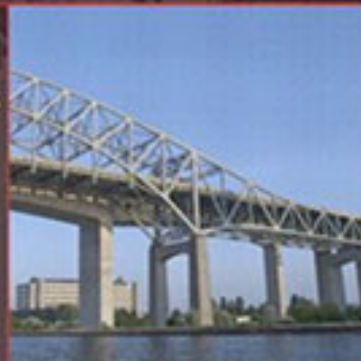
The Addison-Wesley Signature Series



REFACTORING DATABASES

EVOLUTIONARY
DATABASE DESIGN

SCOTT W. AMBLER
PRAMOD J. SADALAGE



*Forewords by Martin Fowler, John Graham,
Sachin Rekhi, and Dr. Paul Dorsey*

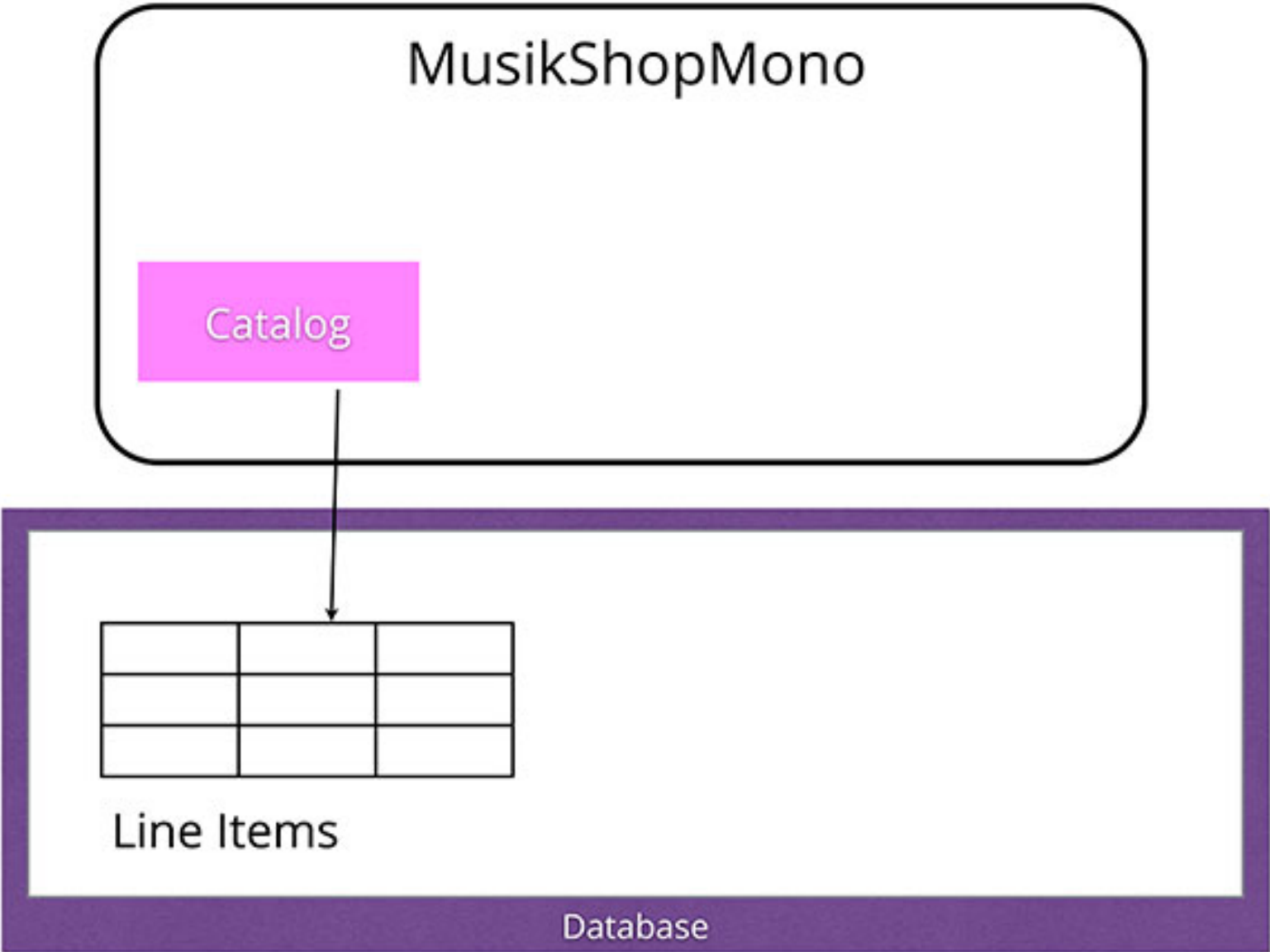
MusikShopMono

Catalog

Database

MusikShopMono

Catalog



The diagram illustrates the relationship between a Catalog and a Database. A pink box labeled 'Catalog' is located inside a rounded rectangle labeled 'MusikShopMono'. An arrow points from the 'Catalog' box to a table labeled 'Line Items'. This table is located inside a purple-bordered rectangle labeled 'Database'.

Line Items

Database

MusikShopMono

Catalog

Finance

The diagram illustrates the architecture of MusikShopMono. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Catalog' on the left and 'Finance' on the right. An arrow points from the 'Catalog' box down to a table within a larger purple rectangle labeled 'Database'. The table has three columns and three rows. Below the table is the text 'Line Items'.

Line Items

Database

MusikShopMono

Catalog

Finance

The diagram illustrates the architecture of MusikShopMono. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Catalog' and 'Finance'. Below this, a large purple rectangle labeled 'Database' at the bottom contains two tables. An arrow points from the 'Catalog' box to the 'Line Items' table, and another arrow points from the 'Finance' box to the 'Ledger' table. Both tables are 3x3 grids.

Line Items

Ledger

Database

MusikShopMono

Catalog

Finance

The diagram illustrates the architecture of MusikShopMono. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Catalog' on the left and 'Finance' on the right. A yellow notepad icon with five horizontal lines is positioned above the 'Finance' box. Arrows point from the 'Catalog' and 'Finance' boxes to two tables within a purple-bordered 'Database' container at the bottom. The 'Catalog' box points to a table labeled 'Line Items', and the 'Finance' box points to a table labeled 'Ledger'. Both tables are 3x3 grids.

Line Items

Ledger

Database

MusikShopMono

Catalog

Finance

The diagram illustrates the MusikShopMono system architecture. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Catalog' on the left and 'Finance' on the right. A yellow notepad icon is positioned above the 'Finance' box. Below this container is a large purple rectangle labeled 'Database' at its base. Inside the 'Database' container, there are two tables. The 'Catalog' box has a solid arrow pointing to the 'Line Items' table. The 'Finance' box has a solid arrow pointing to the 'Ledger' table. A dashed arrow points from the 'Finance' box to the 'Line Items' table. Both the 'Line Items' and 'Ledger' tables are 3x3 grids.

Line Items

Ledger

Database

MusikShopMono

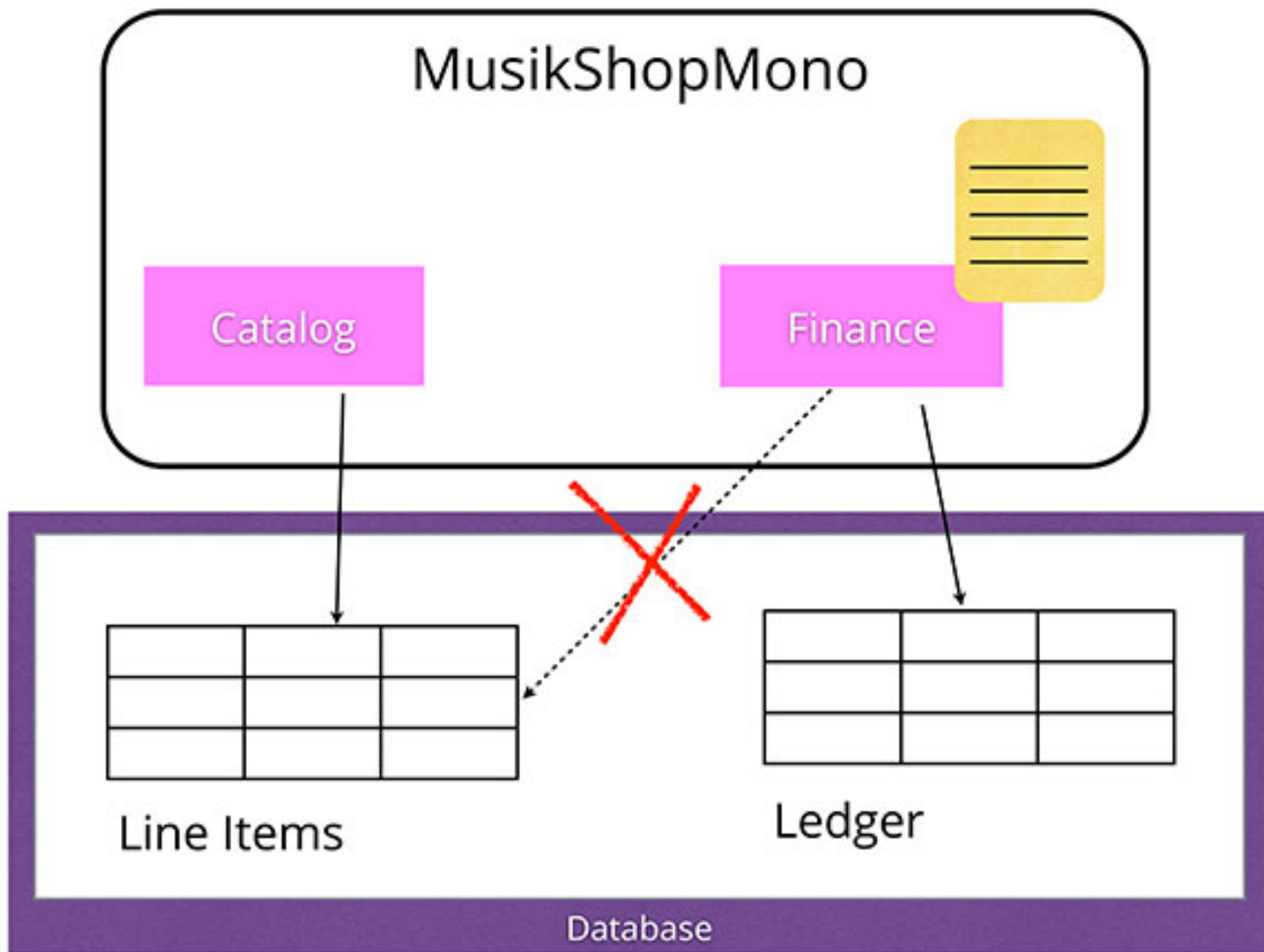
Catalog

Finance

Line Items

Ledger

Database



MusikShopMono

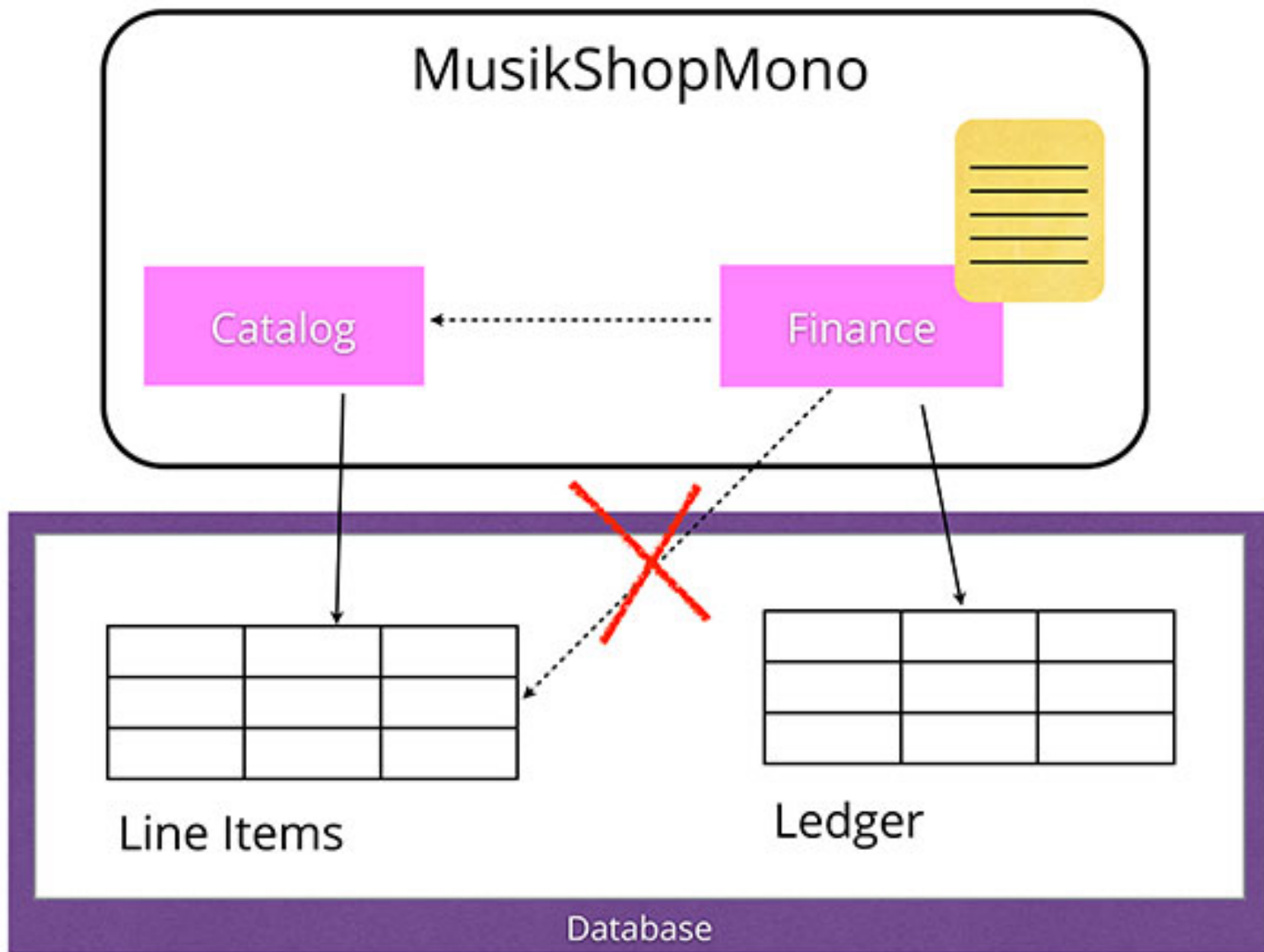
Catalog

Finance

Line Items

Ledger

Database



MusikShopMono

Catalog

Database

Database

MusikShopMono

Catalog

Finance

Database

Database

MusikShopMono

Catalog

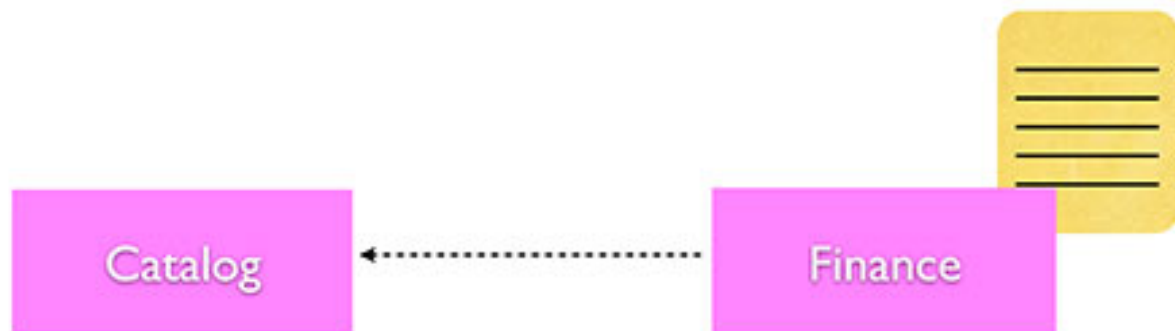
Finance



Database

Database

MusikShopMono



MusikShopMono

Catalog

Finance

The diagram illustrates the MusikShopMono system architecture. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Catalog' on the left and 'Finance' on the right. A dashed arrow points from 'Finance' to 'Catalog'. A yellow document icon with horizontal lines is positioned behind the 'Finance' box. A solid arrow points from the 'Catalog' box down to a table within a purple-bordered box labeled 'Database'. This box also contains the text 'Line Items'. To the right of this is another empty purple-bordered box labeled 'Database'.

Line Items

Database

Database

MusikShopMono

Catalog

Finance

The diagram illustrates the MusikShopMono system architecture. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Catalog' on the left and 'Finance' on the right. A dashed arrow points from 'Finance' to 'Catalog'. A yellow notepad icon is positioned to the right of the 'Finance' box. Below the 'MusikShopMono' container, there are two purple-bordered boxes representing databases. The left database box contains a table with 3 rows and 3 columns, labeled 'Line Items' below it and 'Database' at the bottom. An arrow points from the 'Catalog' box to this table. The right database box contains a similar table with 3 rows and 3 columns, labeled 'Ledger' below it and 'Database' at the bottom. An arrow points from the 'Finance' box to this table.

Line Items

Database

Ledger

Database

MusikShop System

Catalog Service

Finance Service

Catalog DB

Finance DB

MusikShop System

Catalog Service

Finance Service




Catalog DB

Finance DB

MusikShop System

Catalog Service

Finance Service



An arrow points from the Catalog Service box down to the Line Items table, indicating a query or data retrieval operation.

Line Items

Catalog DB

Finance DB

MusikShop System

Catalog Service

Finance Service

The diagram illustrates the MusikShop System architecture. At the top, the 'MusikShop System' is labeled. Below it, two services are shown: 'Catalog Service' on the left and 'Finance Service' on the right, both in purple rounded rectangles. A dashed arrow points from the 'Finance Service' to the 'Catalog Service'. Below each service is a database box with a purple border. The 'Catalog DB' box contains a table labeled 'Line Items' with 3 rows and 3 columns. The 'Finance DB' box contains a table labeled 'Ledger' with 3 rows and 3 columns. Solid arrows point from each service to its respective table.

Line Items

Catalog DB

Ledger

Finance DB



ID	Name	
123	Give Blood	

Line Items

Database

.....

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

.....

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

.....

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

ID		
123		

Line Items

Catalog DB

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

ID		
123		

Line Items

Catalog DB

SKU		
/catalog/item/123		

Ledger

Finance DB

ID	Name	
123	Give Blood	

Line Items

SKU		
123		

Ledger

Database

ID		
123		

Line Items

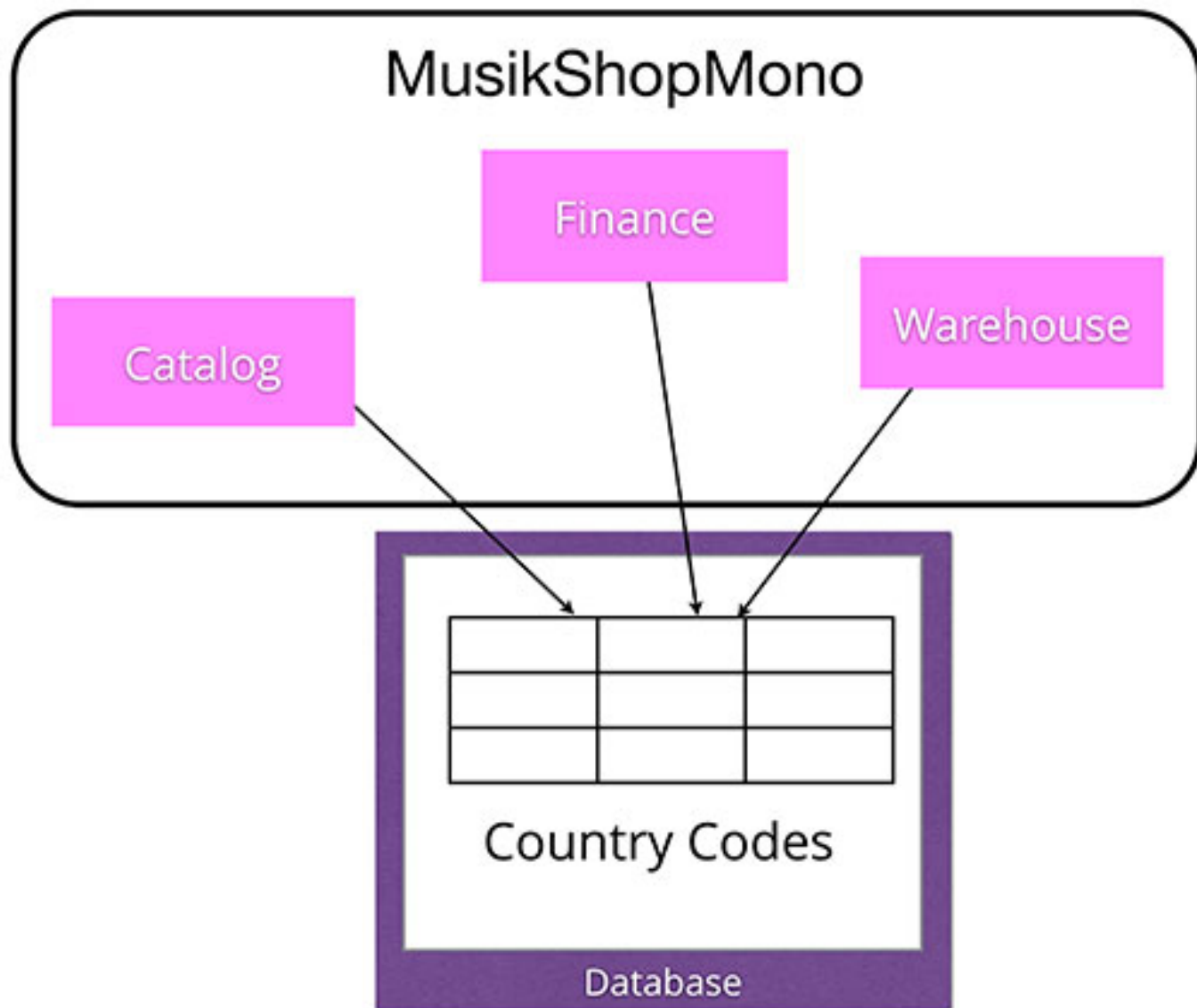
Catalog DB

SKU		
/catalog/item/123		

Ledger

Finance DB

Breaking foreign key relationships can introduce inconsistency in your system



MusikShopMono

Catalog

Finance

Warehouse

Country Codes

Catalog DB

Country Codes

Finance DB

Country Codes

Warehouse DB

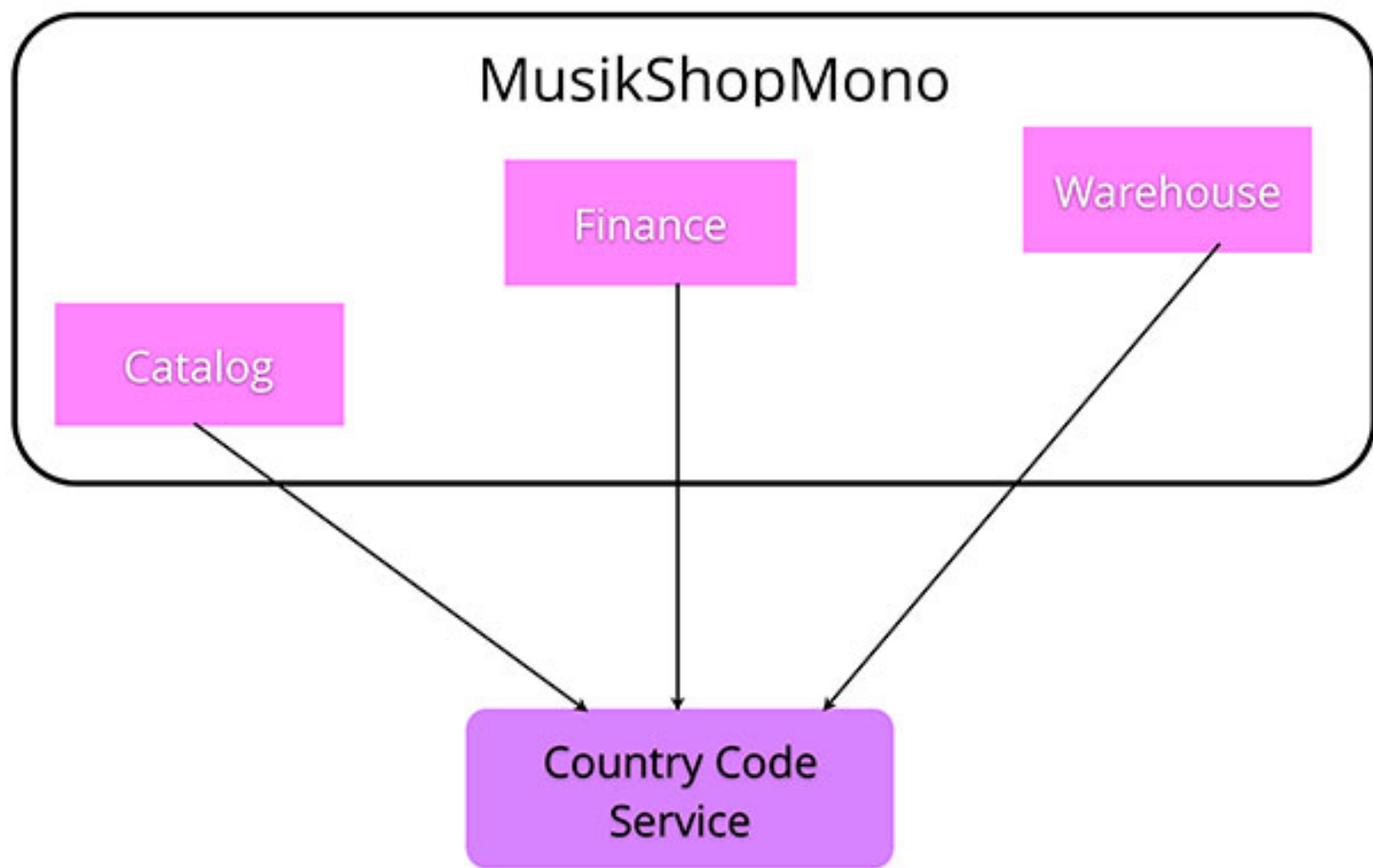
MusikShopMono

Catalog

Finance

Warehouse





MusikShopMono

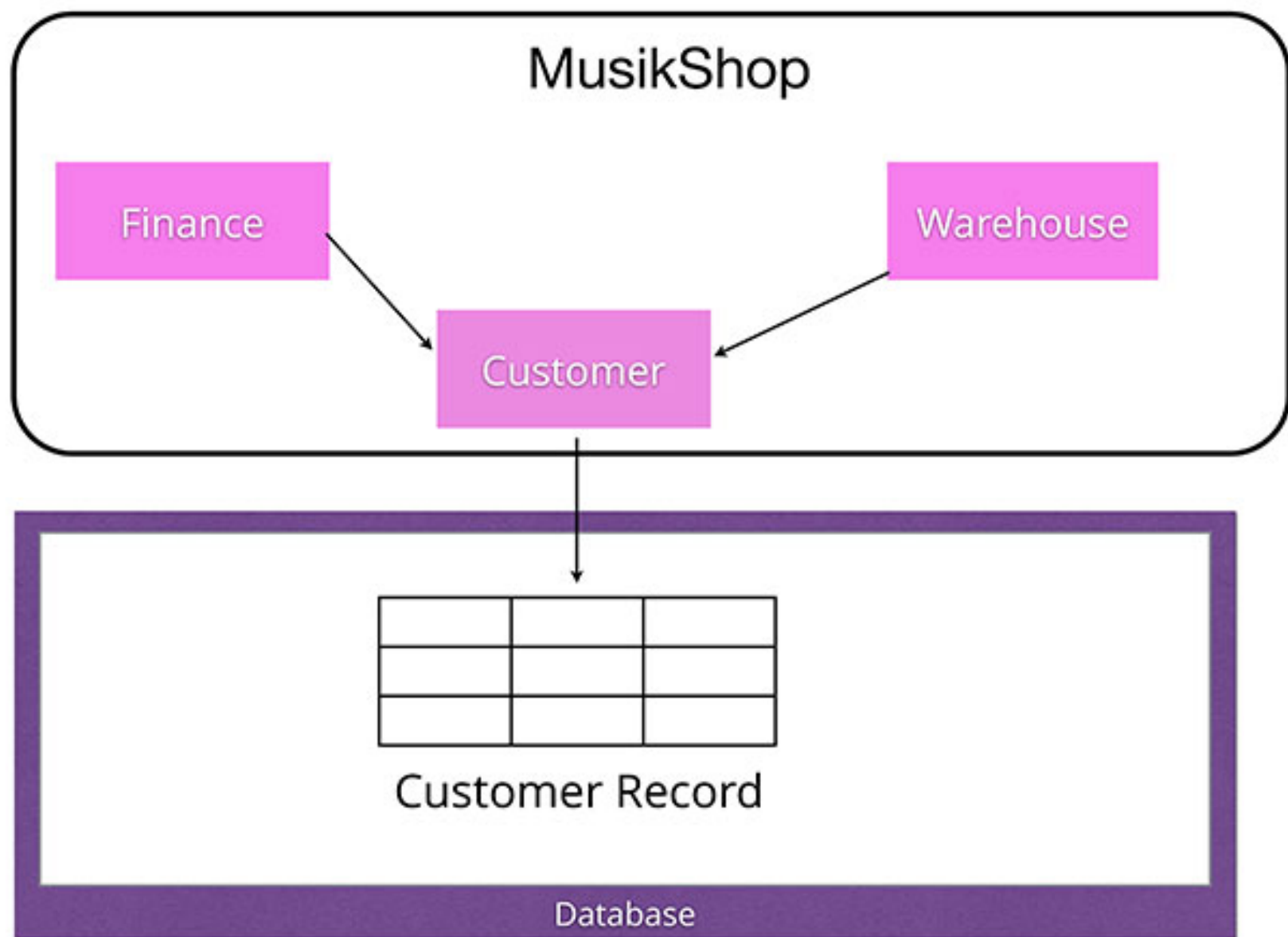
Finance

Warehouse

The diagram illustrates the MusikShopMono system architecture. At the top, a rounded rectangle labeled 'MusikShopMono' contains two pink boxes: 'Finance' and 'Warehouse'. Arrows from these boxes point down to a table labeled 'Customer Record' inside a larger purple rectangle labeled 'Database'. The table has 3 rows and 3 columns.

Customer Record

Database

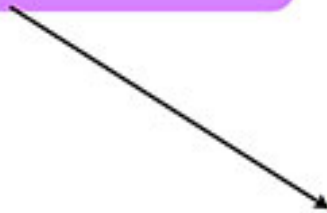


MusikShop System

Finance Service

Warehouse Service

Customer Service



MusikShop

Catalog

Warehouse

The diagram illustrates the data structure of MusikShop. It features a large rounded rectangle labeled 'MusikShop' at the top. Inside this rectangle are two pink boxes: 'Catalog' on the left and 'Warehouse' on the right. Below the 'MusikShop' rectangle is a purple-bordered box labeled 'Database'. Inside the 'Database' box is a white box labeled 'Item' which contains a 3x3 grid. Two arrows originate from the 'MusikShop' rectangle: one from the 'Catalog' box pointing to the top-left cell of the 'Item' grid, and another from the 'Warehouse' box pointing to the top-right cell of the 'Item' grid.

Item

Database

MusikShop

Catalog

Warehouse

The diagram illustrates the MusikShop architecture. At the top, a large rounded rectangle labeled 'MusikShop' contains two pink boxes: 'Catalog' on the left and 'Warehouse' on the right. Below this, a purple-bordered box labeled 'Database' contains a table with 3 rows and 3 columns. Arrows point from the 'Catalog' and 'Warehouse' boxes to the top-left and top-right cells of the table, respectively. To the right of the table is a text string 'Bee Gees Hits | \$4.99 | 45'.

Item

Database

Bee Gees Hits | \$4.99 | 45

MusikShop

Catalog

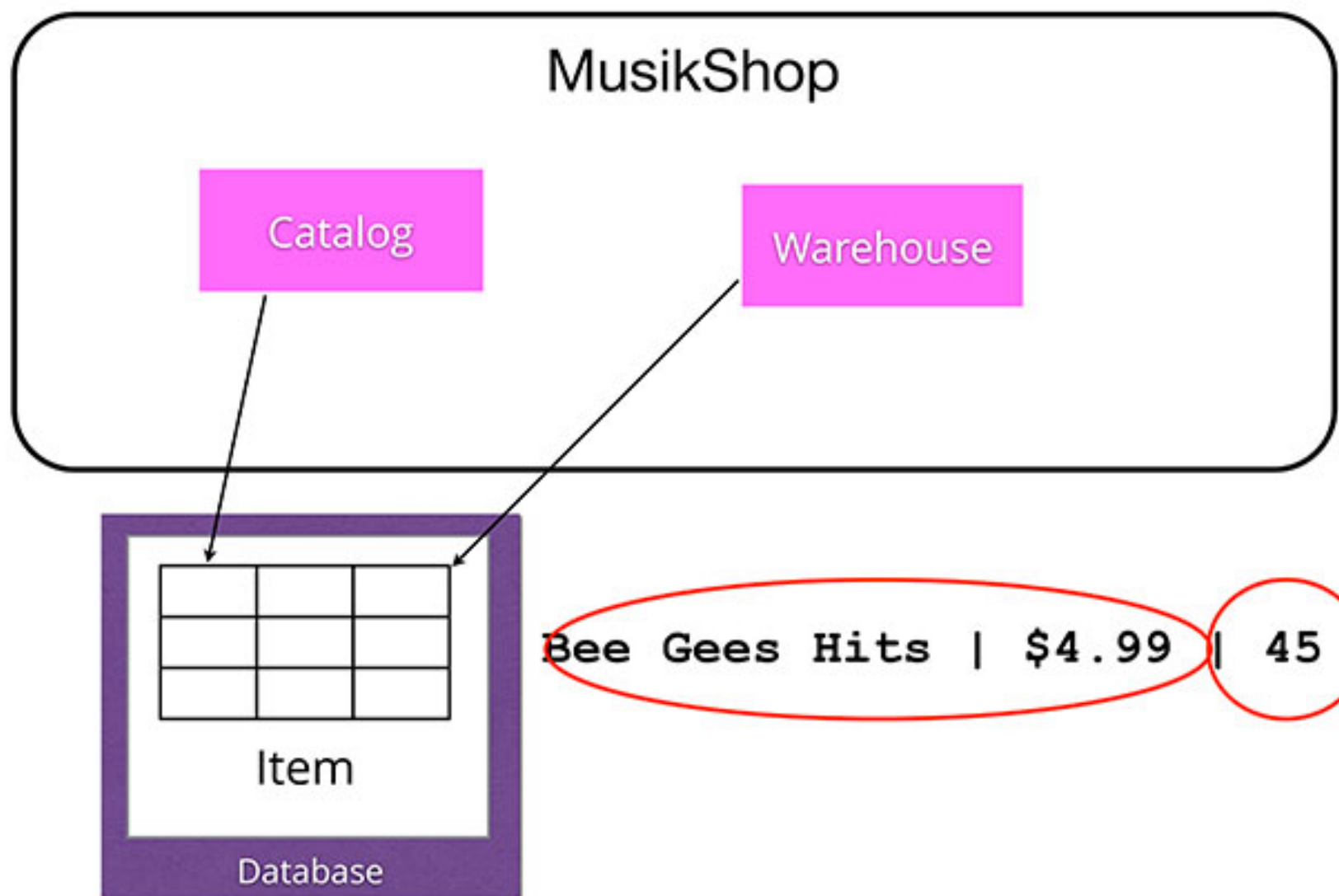
Warehouse

The diagram shows a MusikShop container with two components, Catalog and Warehouse, both pointing to a Database. The Database contains an Item table with 3 rows and 3 columns.

Item

Database

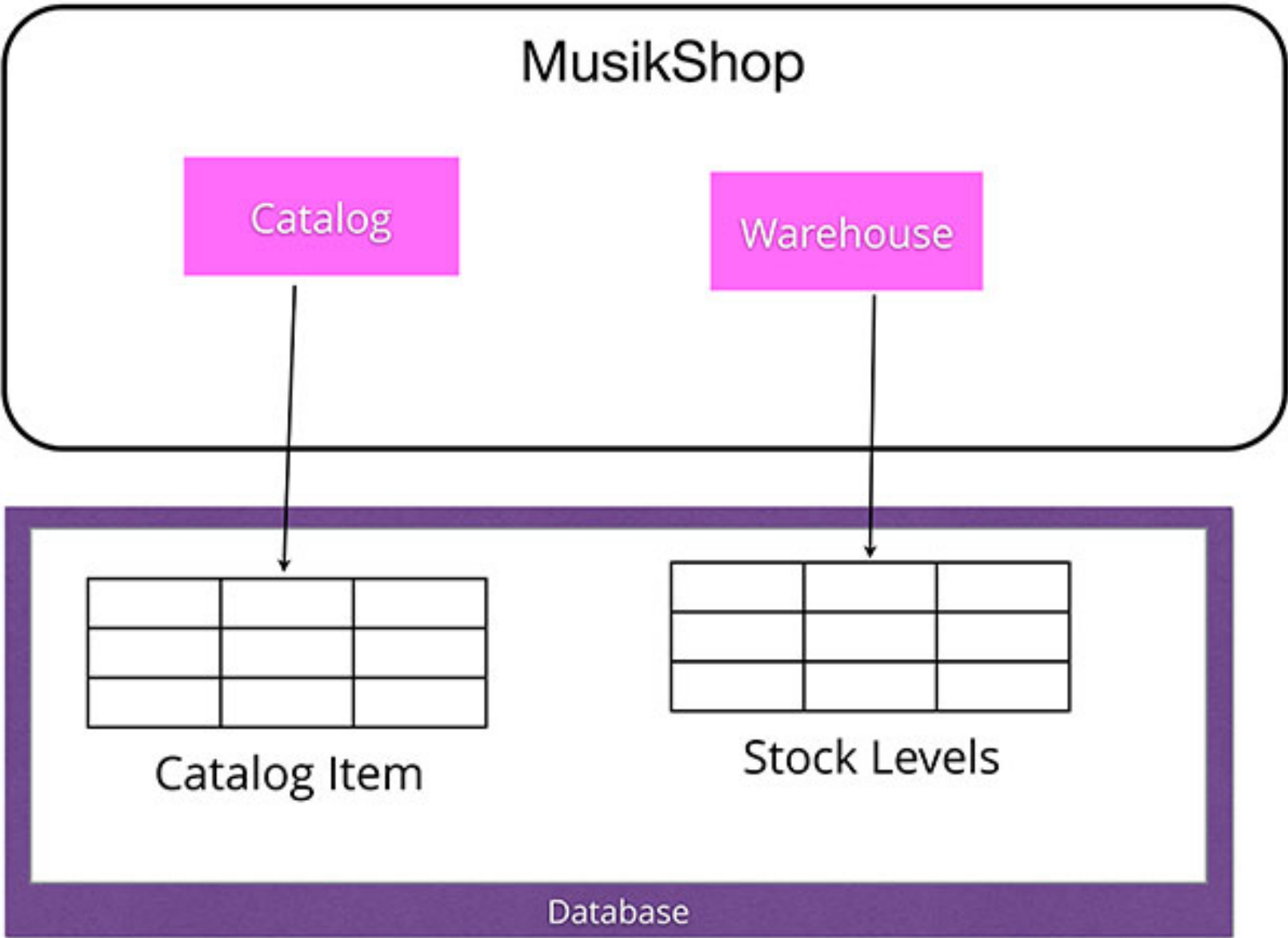
Bee Gees Hits | \$4.99 | 45



MusikShop

Catalog

Warehouse



Catalog Item

Stock Levels

Database

Summary

Where to start

Modelling services

Incremental change

DB Integration & Refactoring