



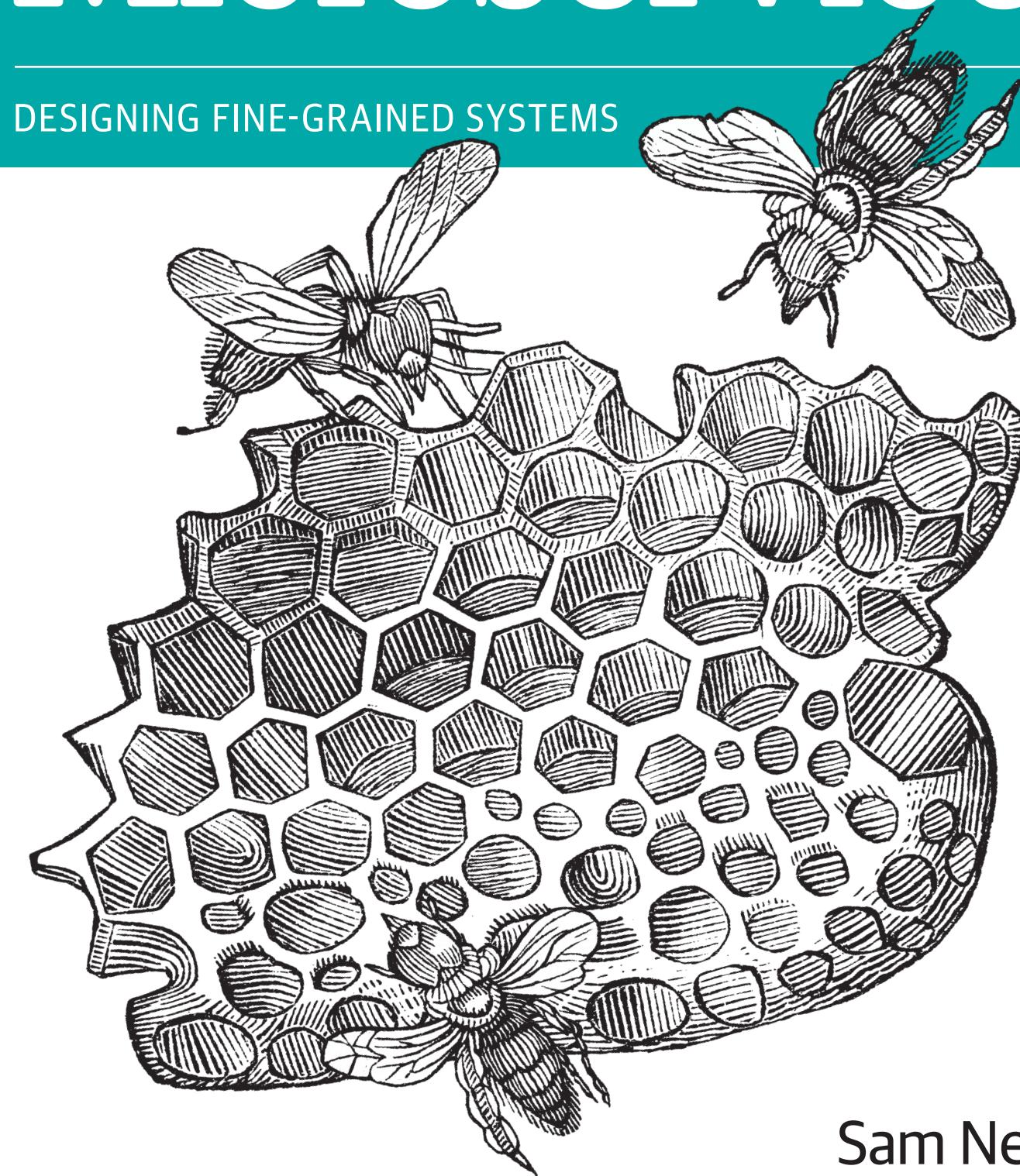
# HIDING THE LEAD COUPLING, COHESION AND MICROSERVICES

Sam Newman

O'REILLY®

# Building Microservices

DESIGNING FINE-GRAINED SYSTEMS



Sam Newman

**Sam  
Newman  
& Associates**



@samnewman

**NEW BOOK!**

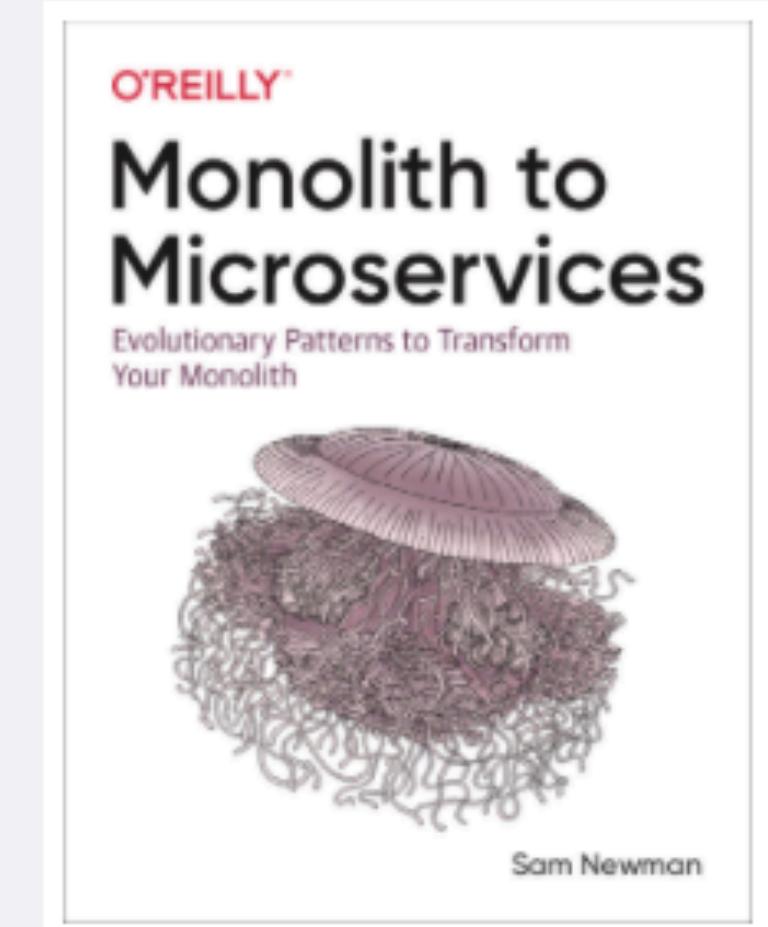
# Monolith To Microservices.

*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.



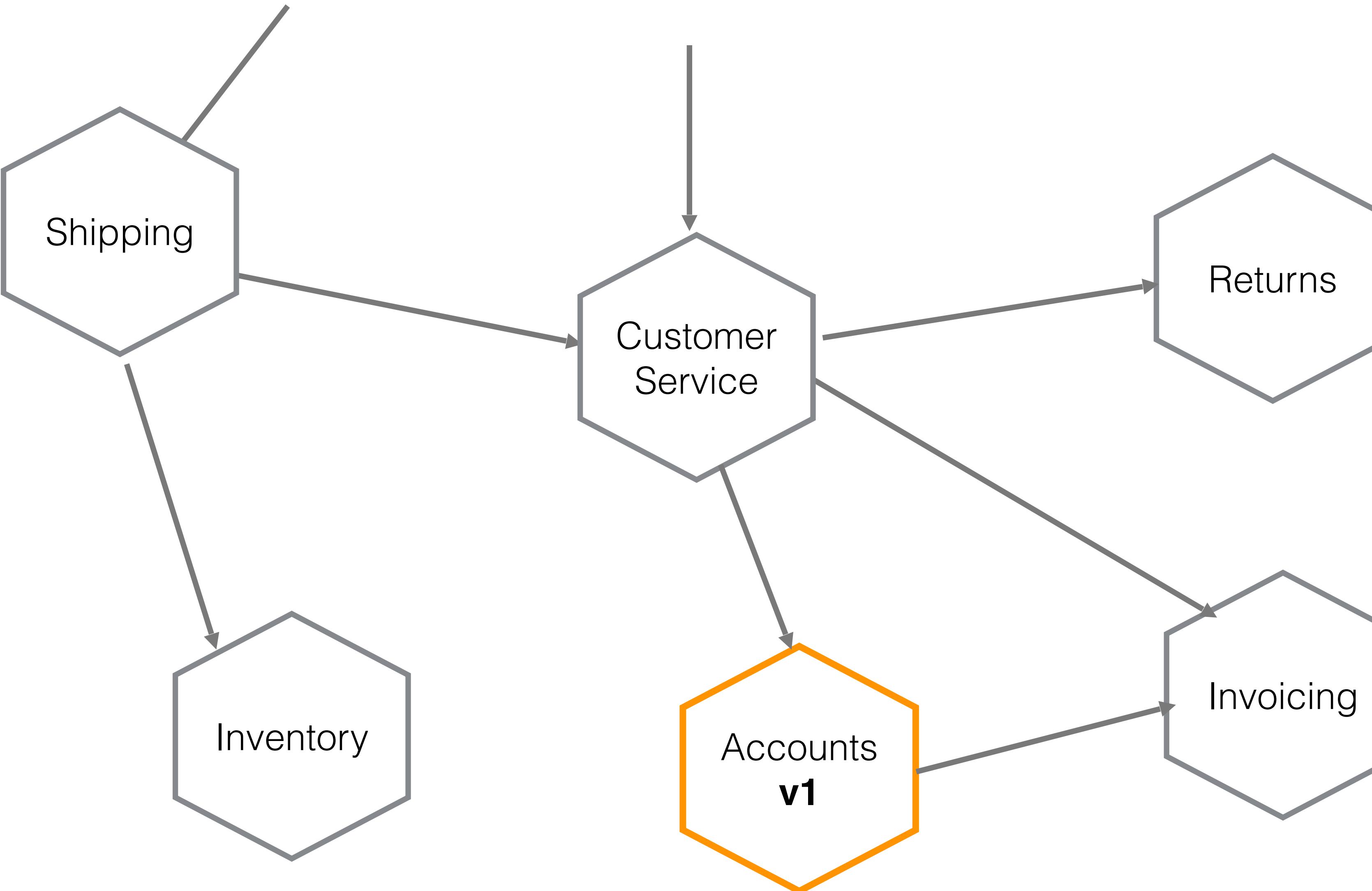
[Read on O'Reilly Learning Online](#)

[Order at Amazon.com](#)

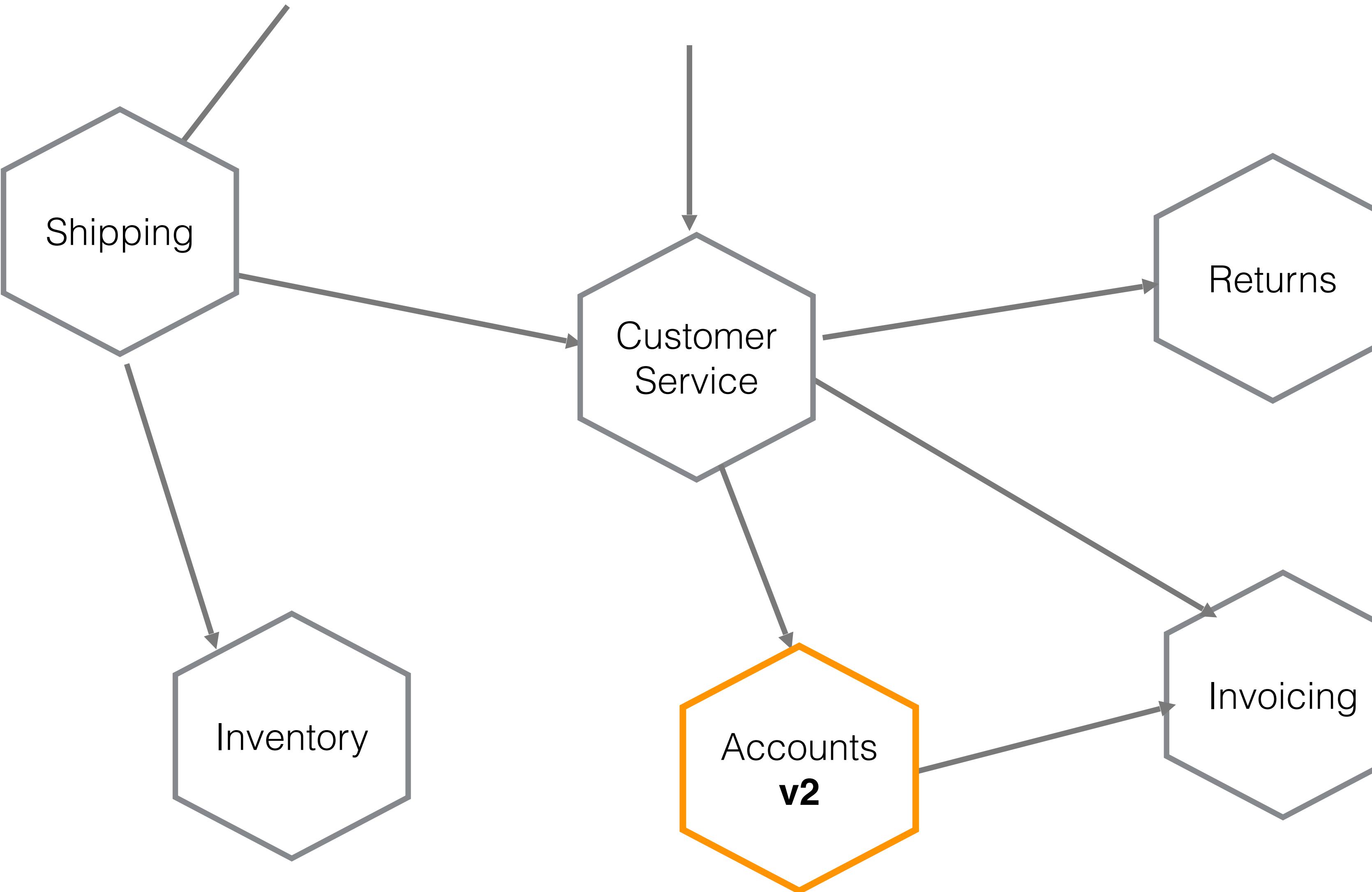
[Order at Amazon.co.uk](#)

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

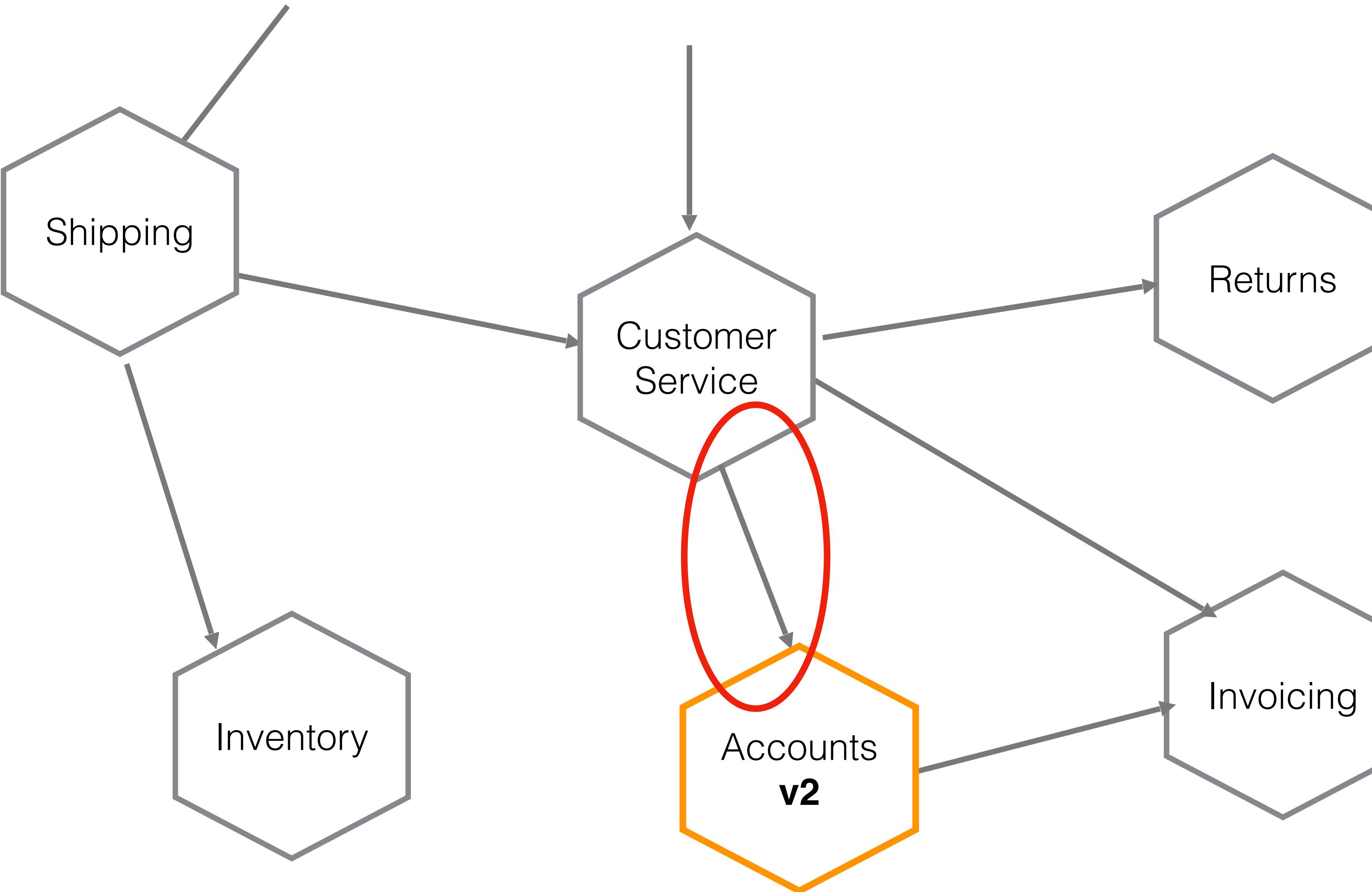
## INDEPENDENT DEPLOYABILITY



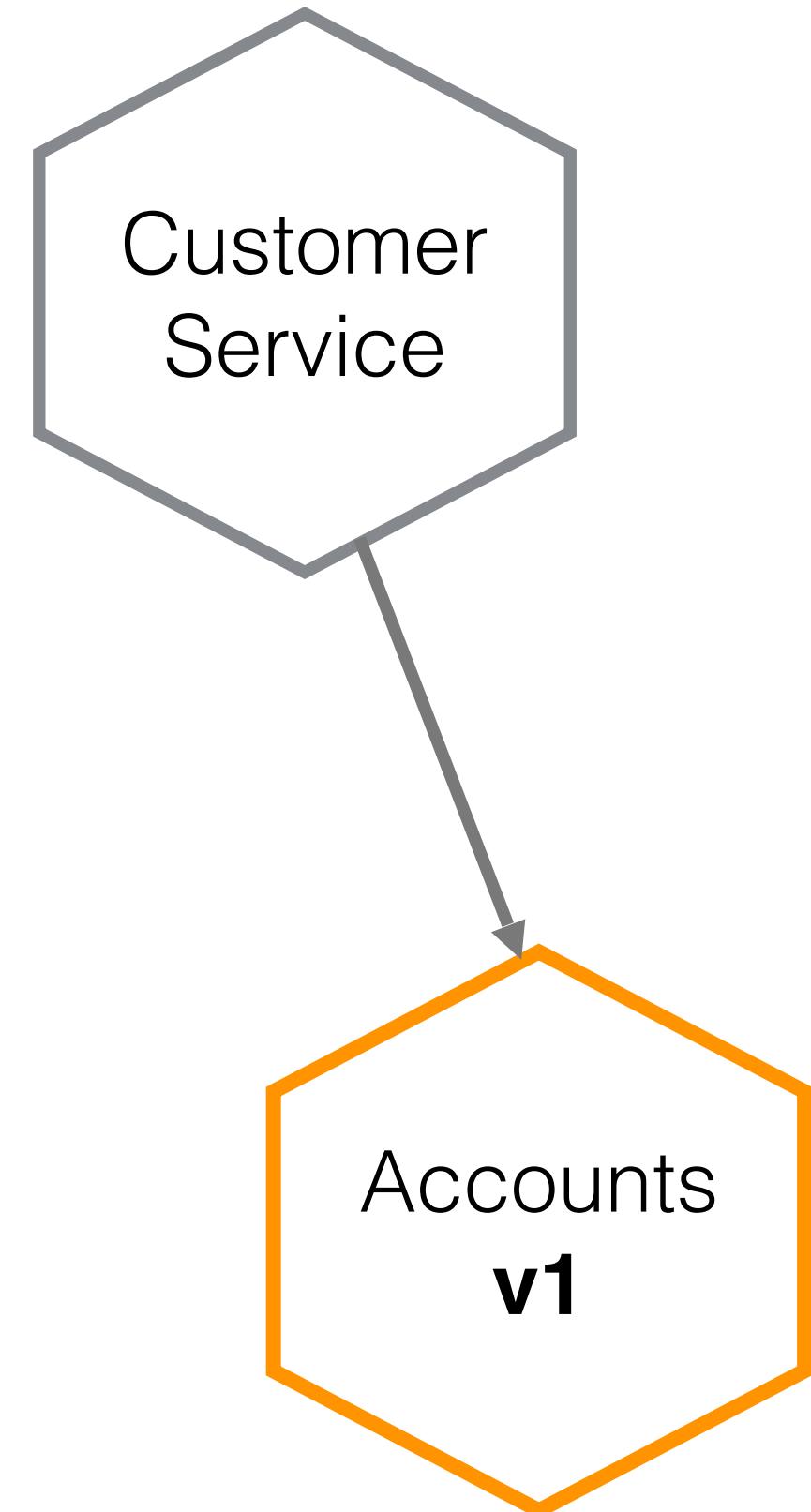
# INDEPENDENT DEPLOYABILITY



# INDEPENDENT DEPLOYABILITY

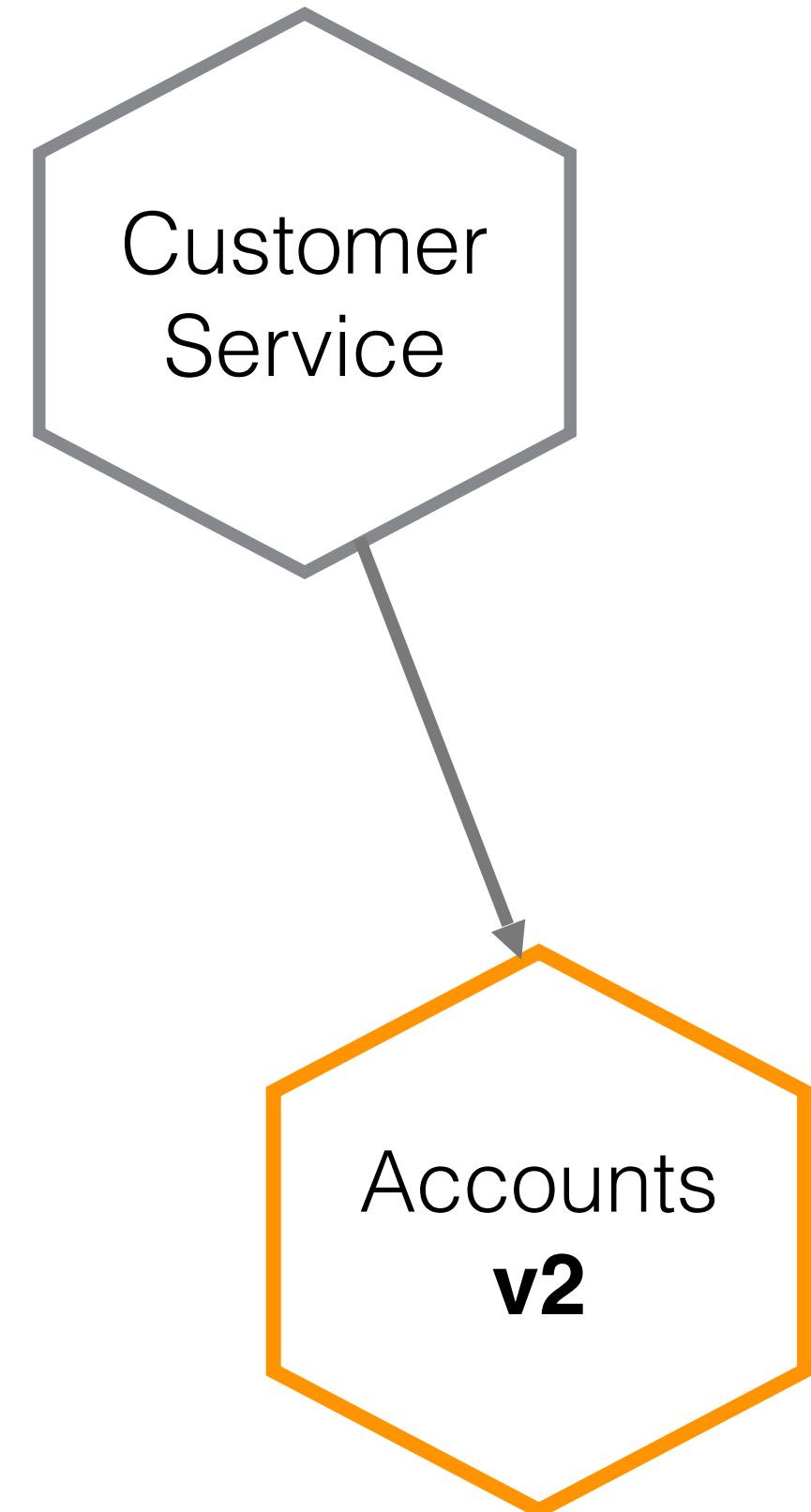


**Independent deployability  
requires interface stability**



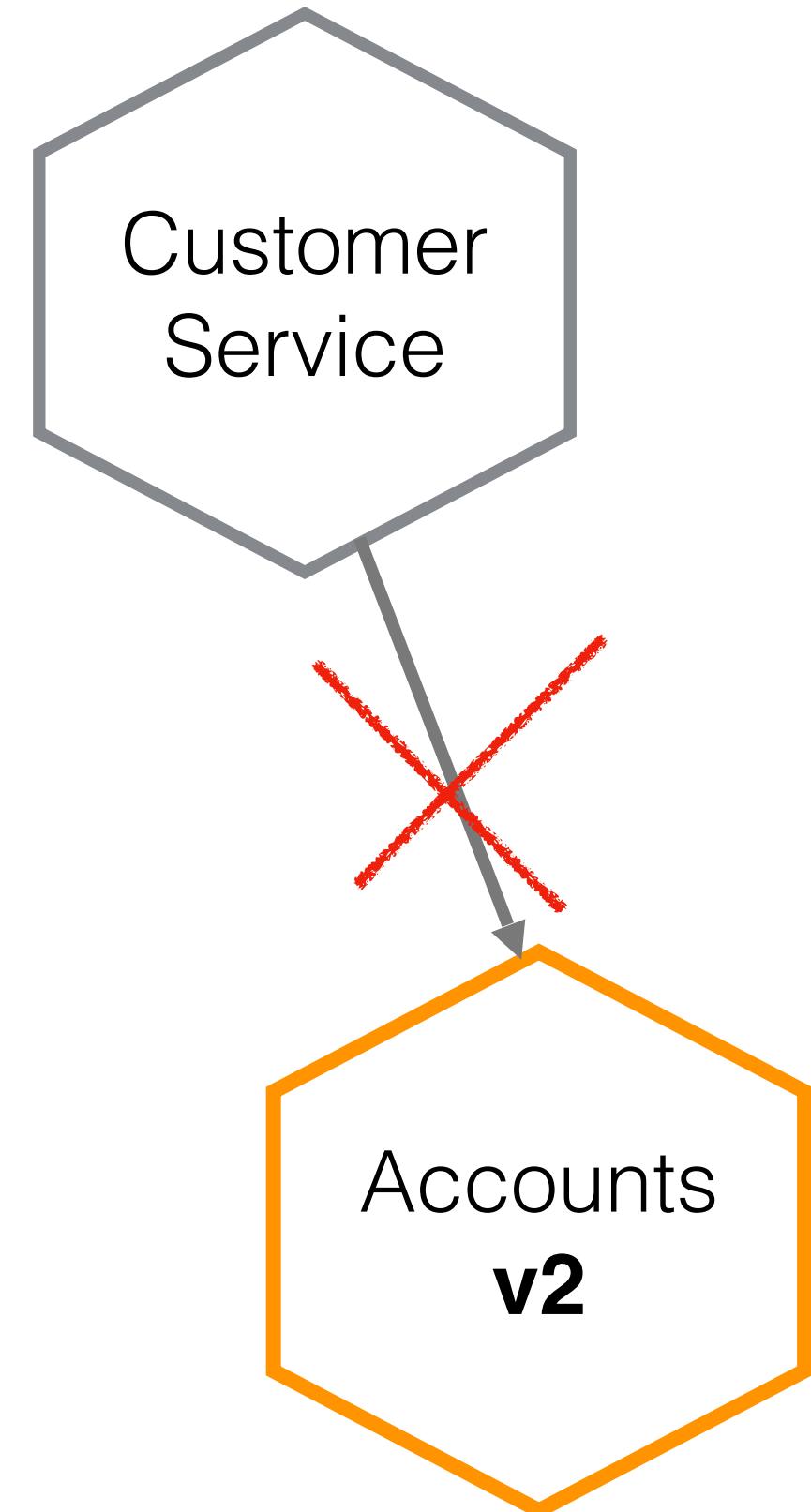
**Maintaining backwards  
compatibility is key**

**Independent deployability  
requires interface stability**



**Maintaining backwards  
compatibility is key**

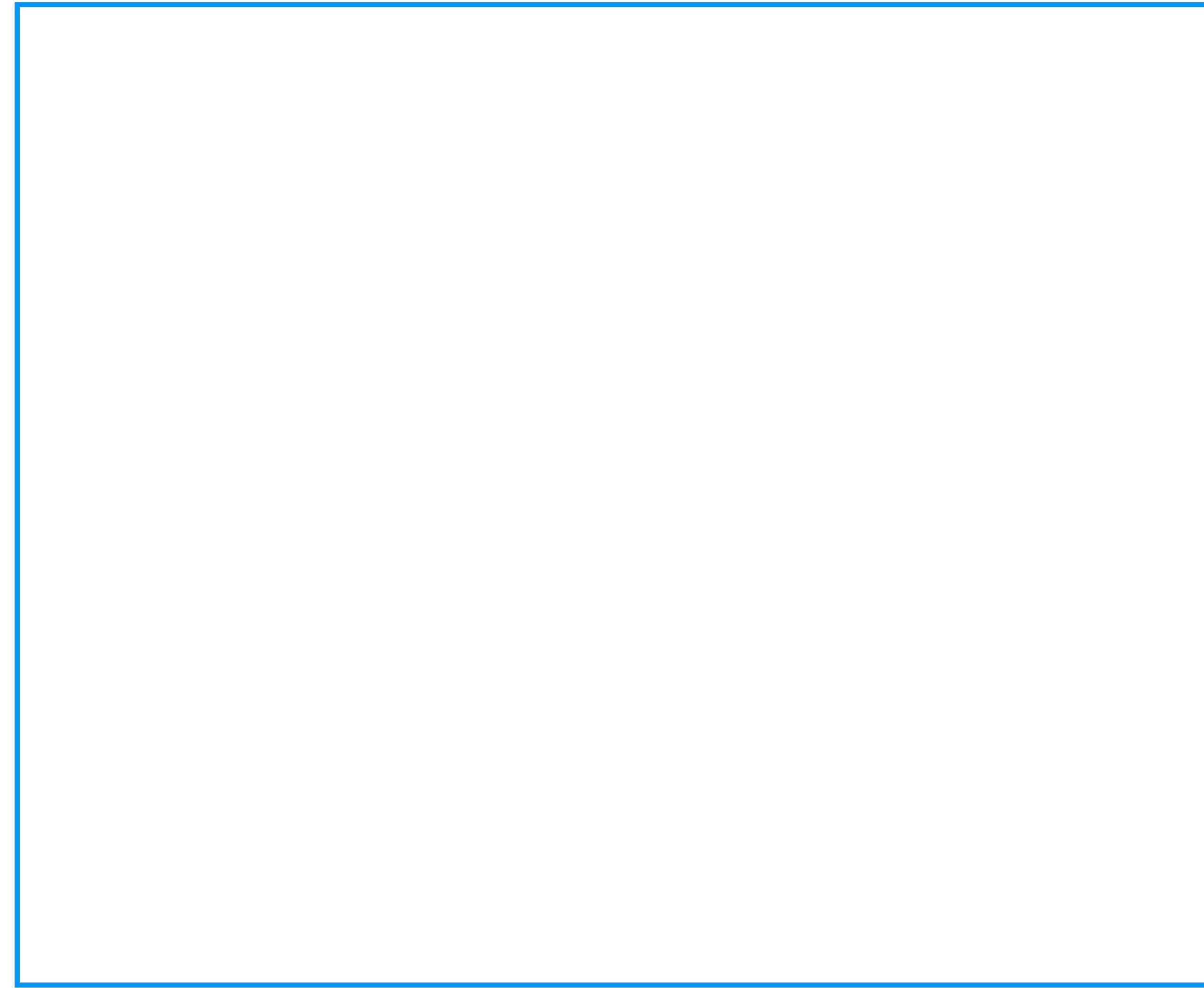
**Independent deployability  
requires interface stability**



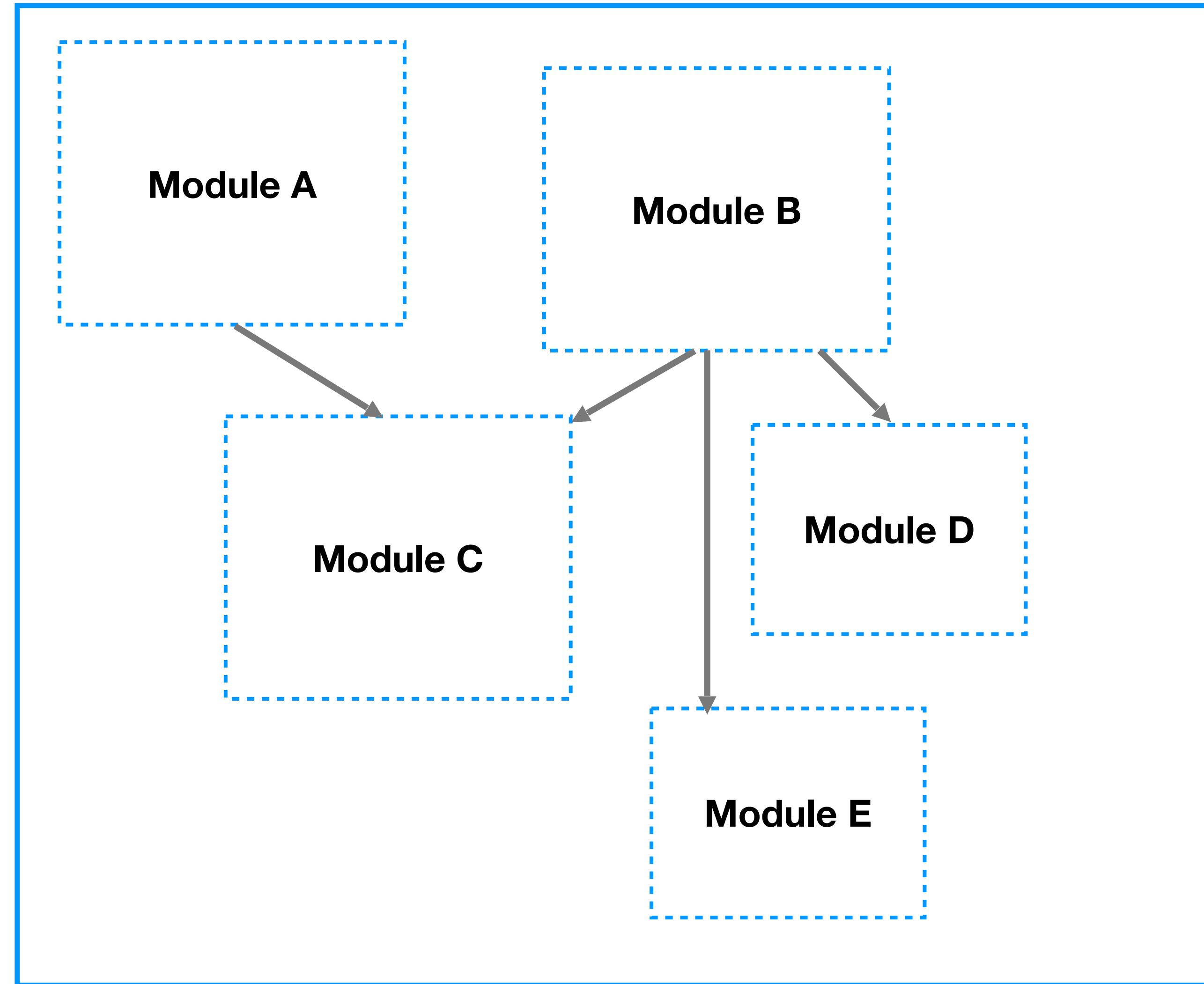
**Maintaining backwards  
compatibility is key**



# MODULES

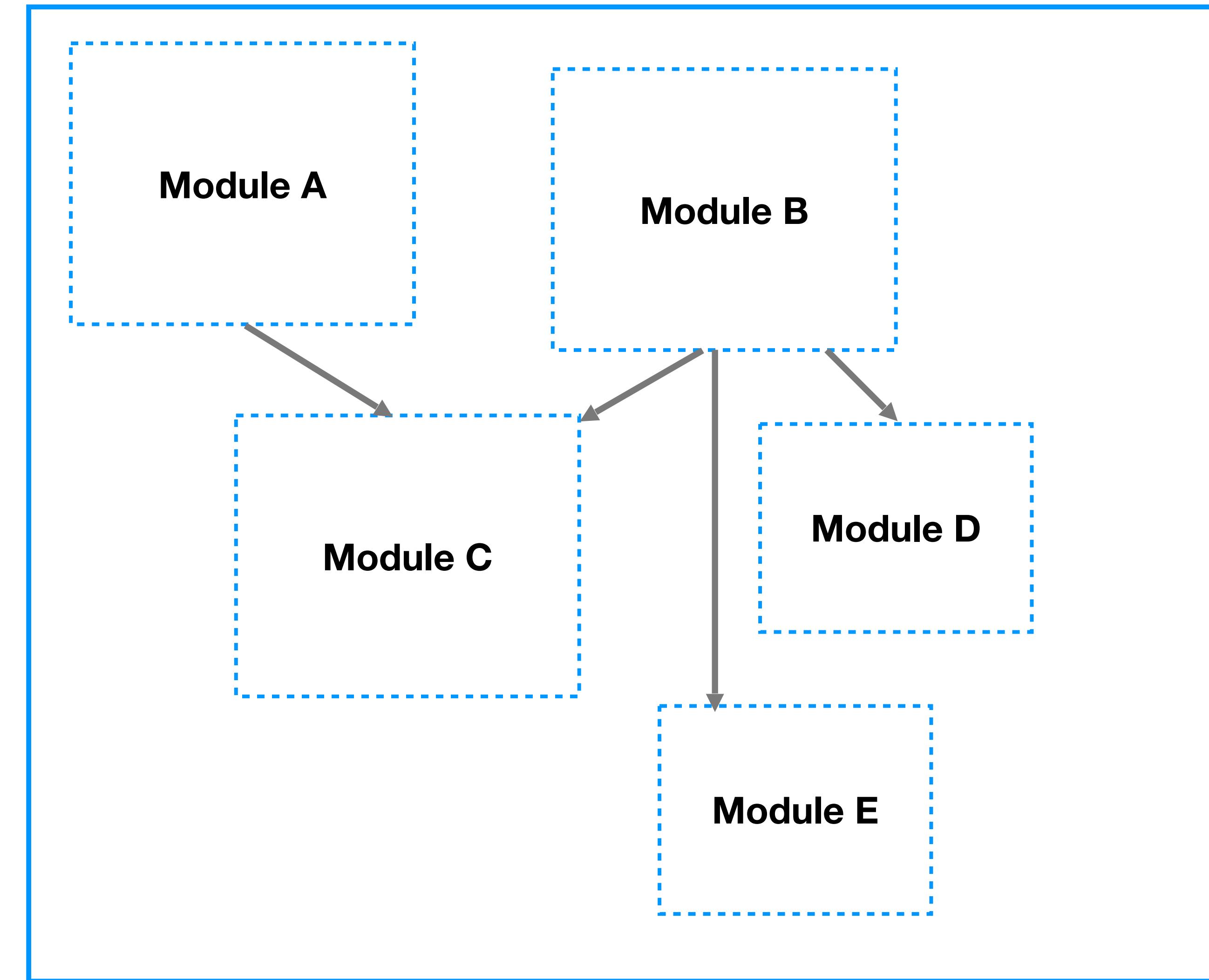


# MODULES



## MODULES

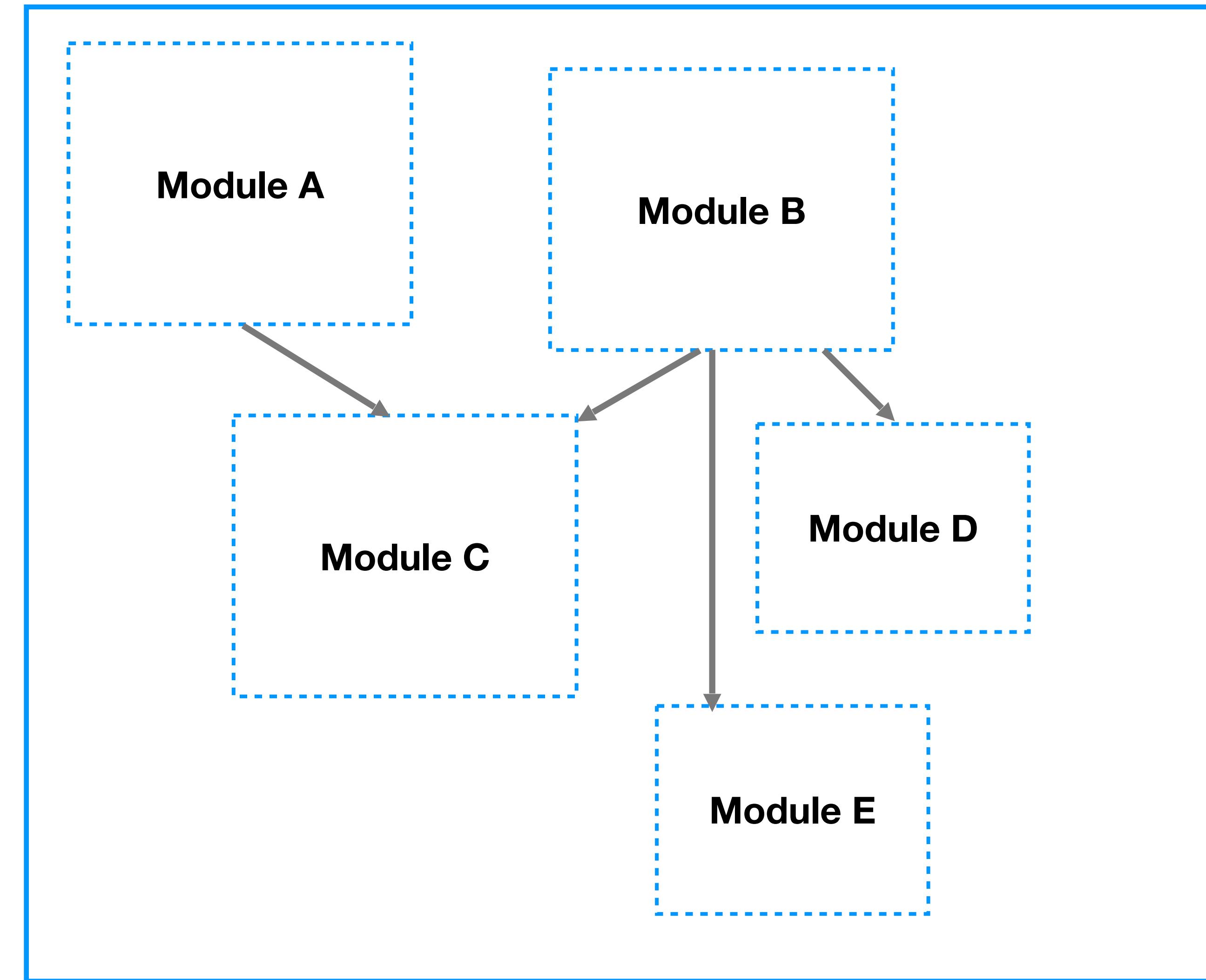
Allow for  
independent  
working



# MODULES

Allow for  
independent  
working

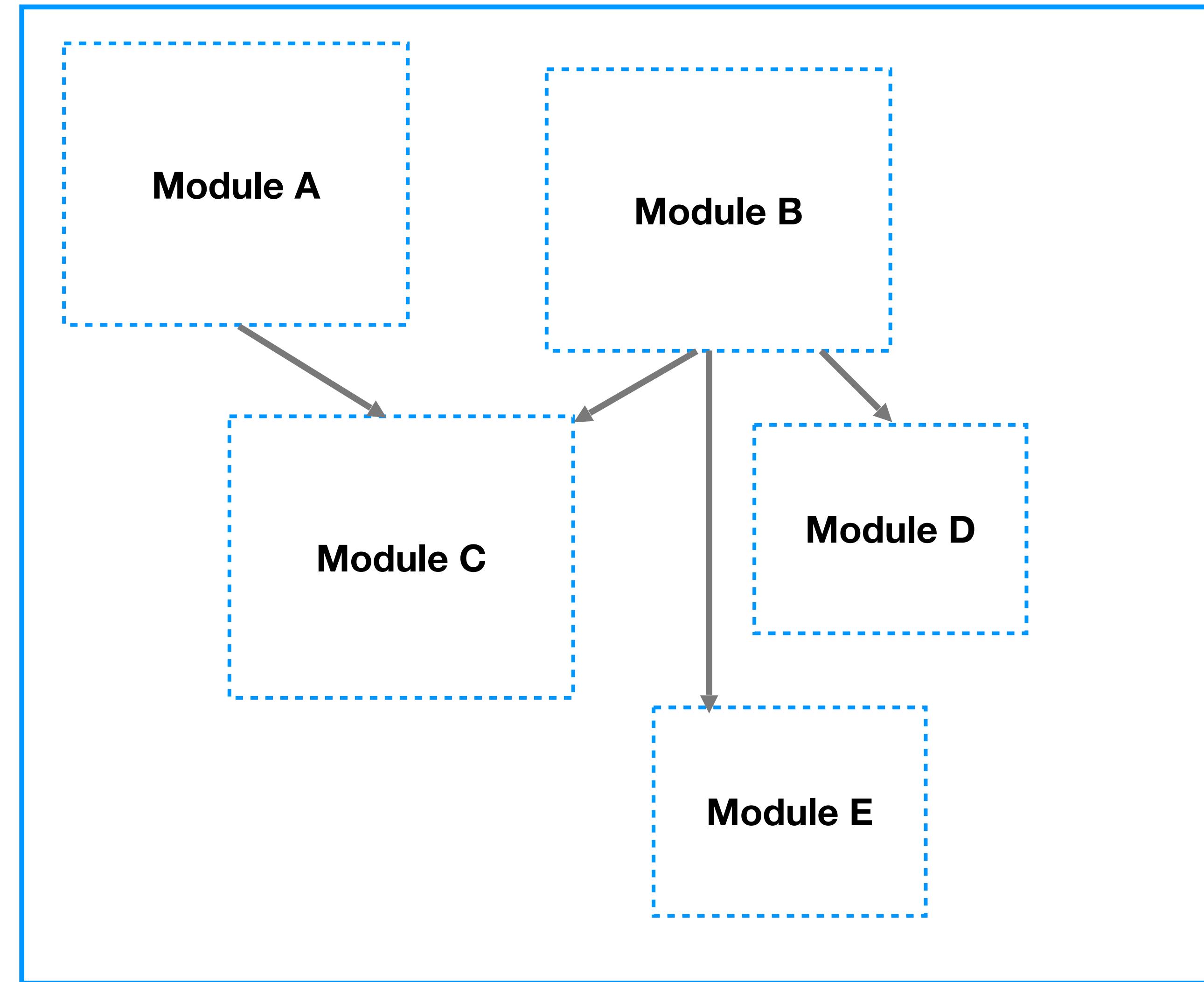
And reuse



# MODULES

Allow for independent working

And reuse

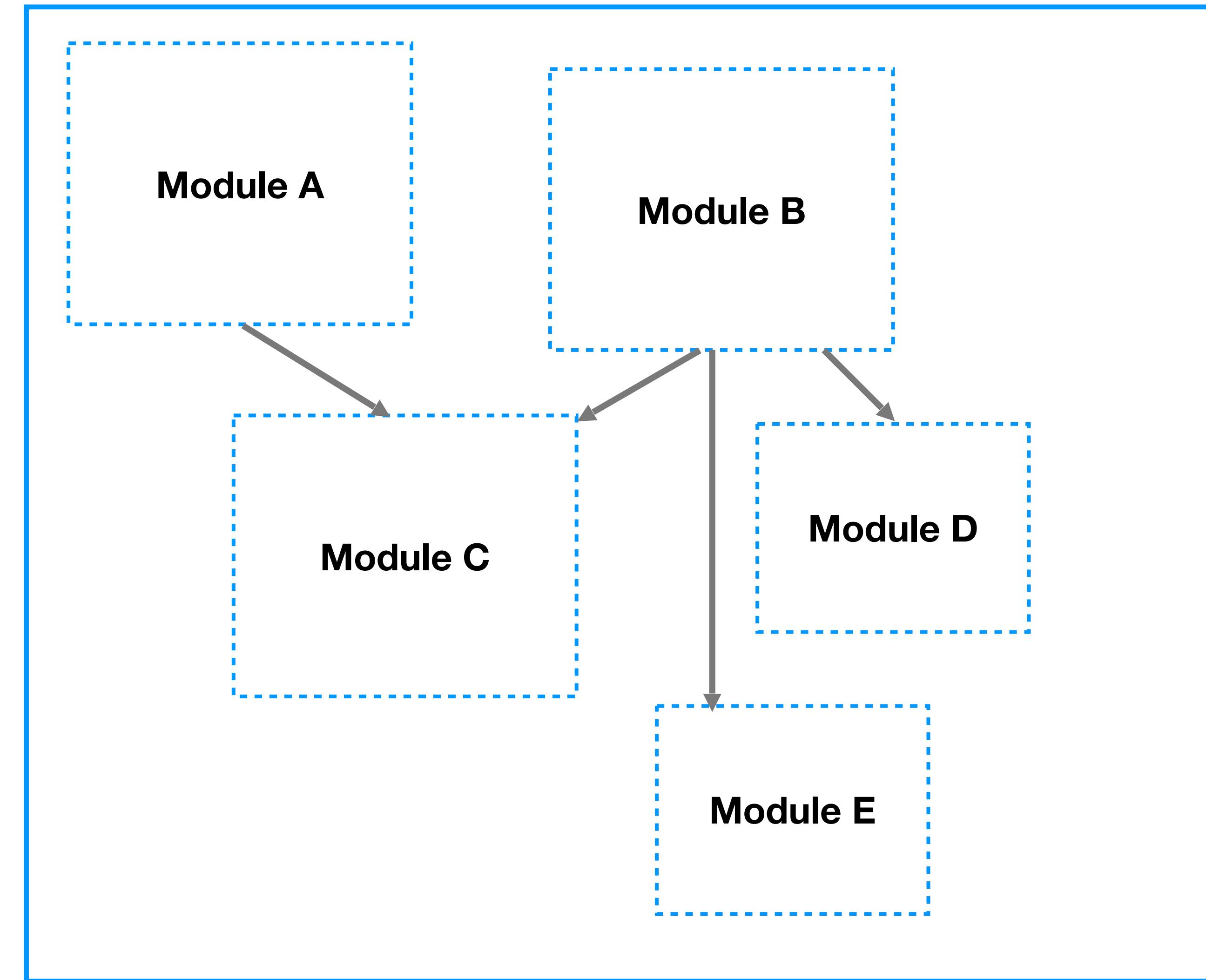


Namespaces,  
packages

# MODULES

Allow for independent working

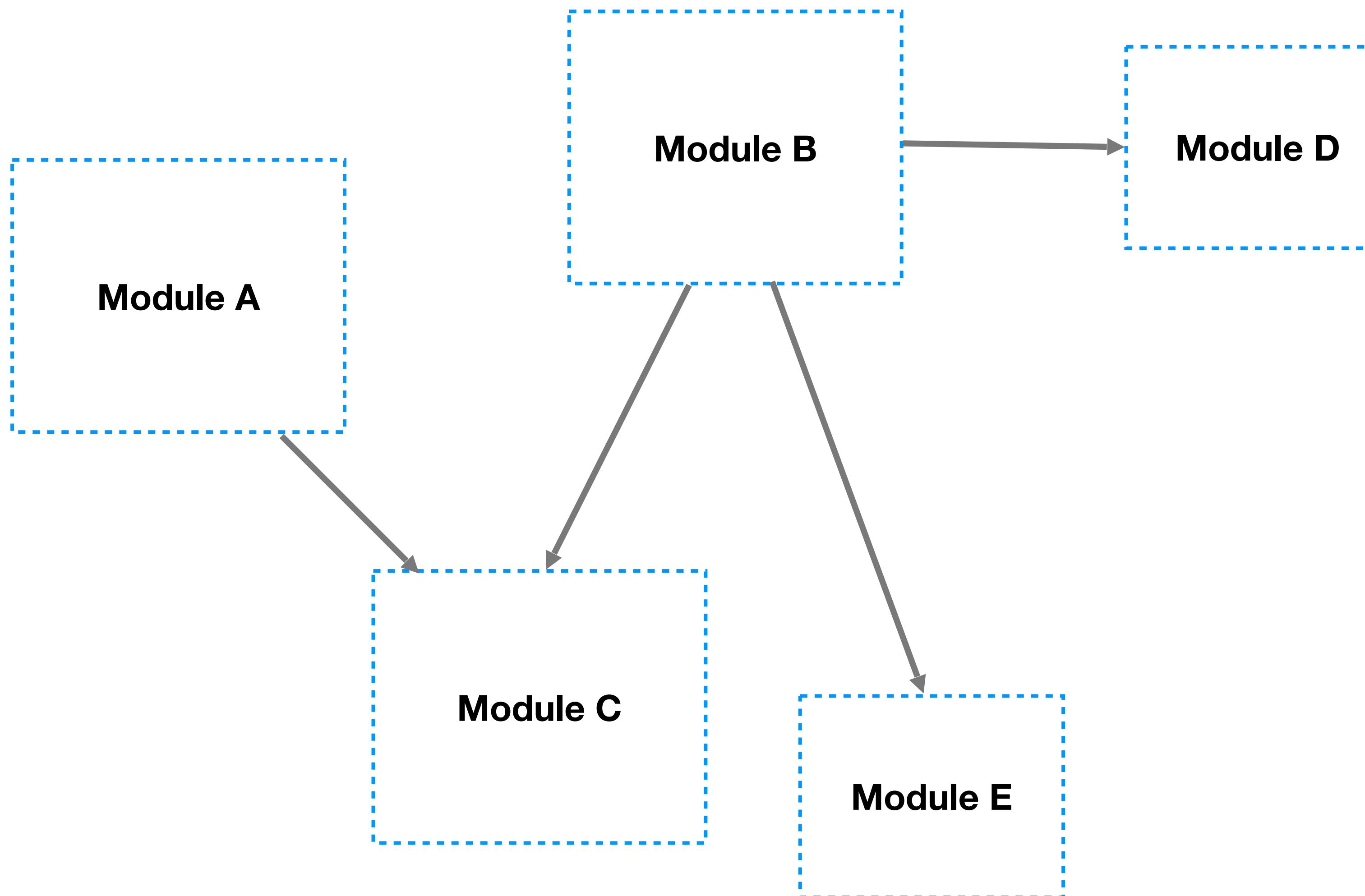
And reuse



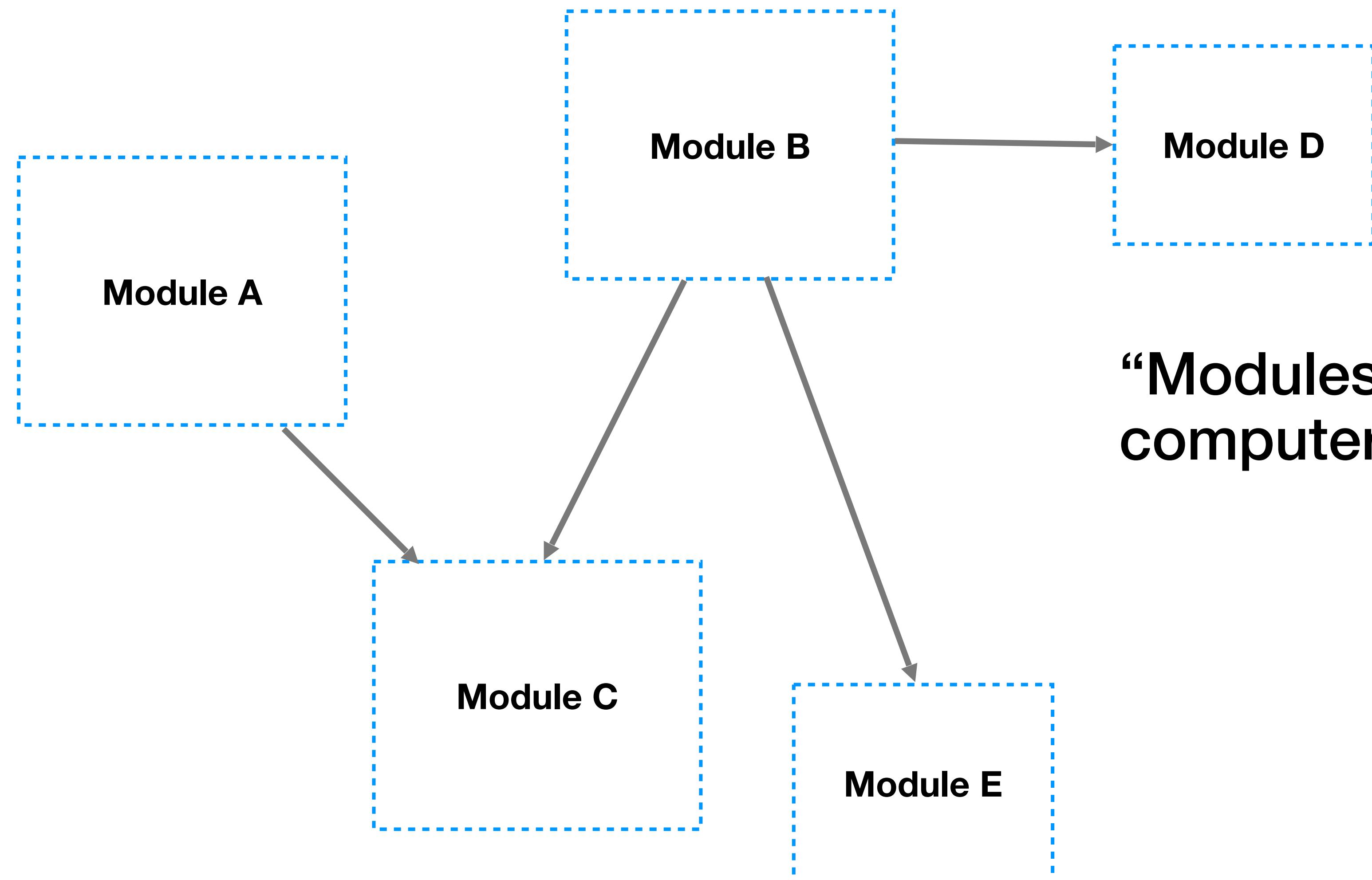
Namespaces,  
packages

NPMs, Gems,  
JARs etc

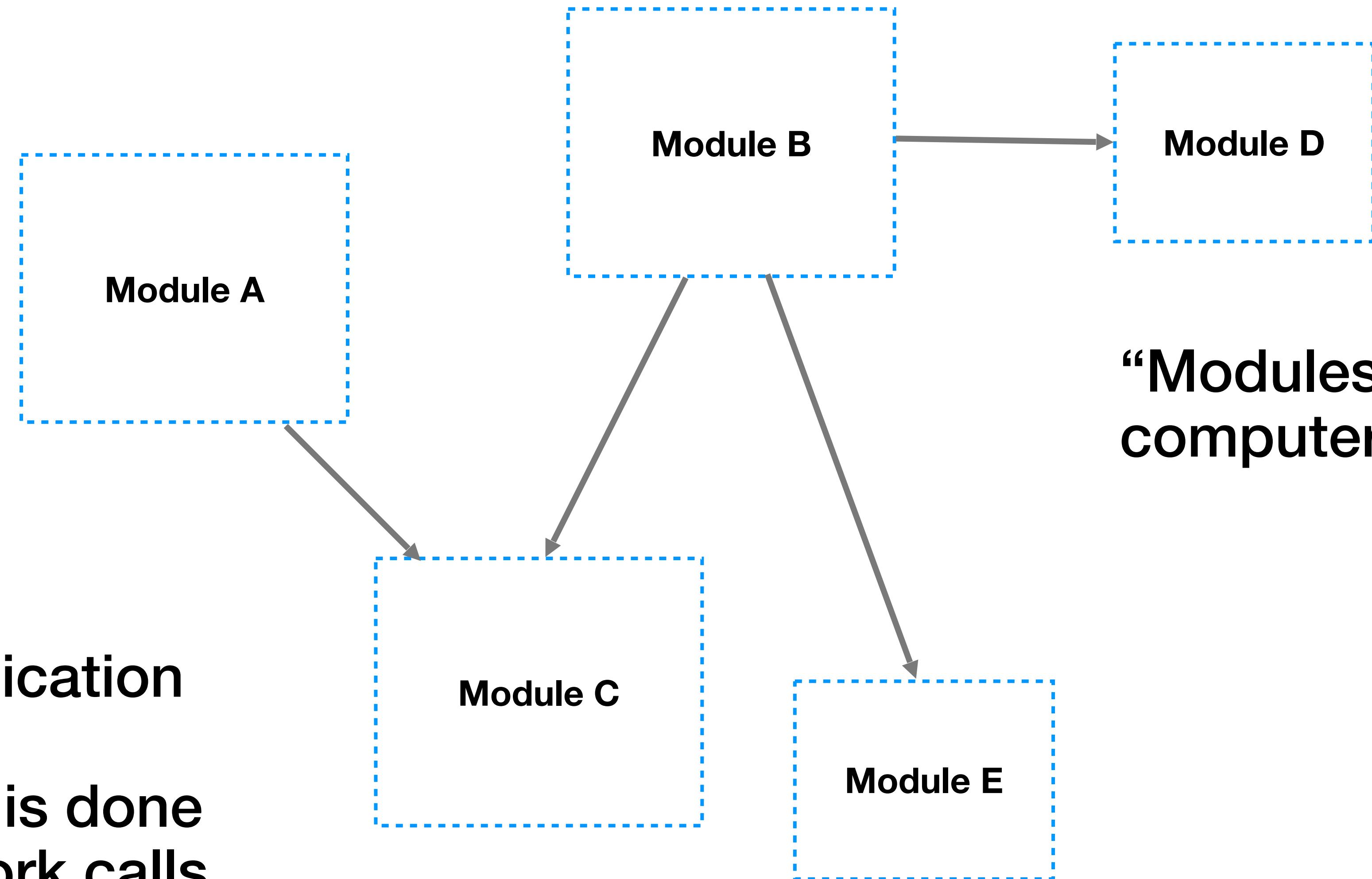
## AND MICROSERVICES?



## AND MICROSERVICES?

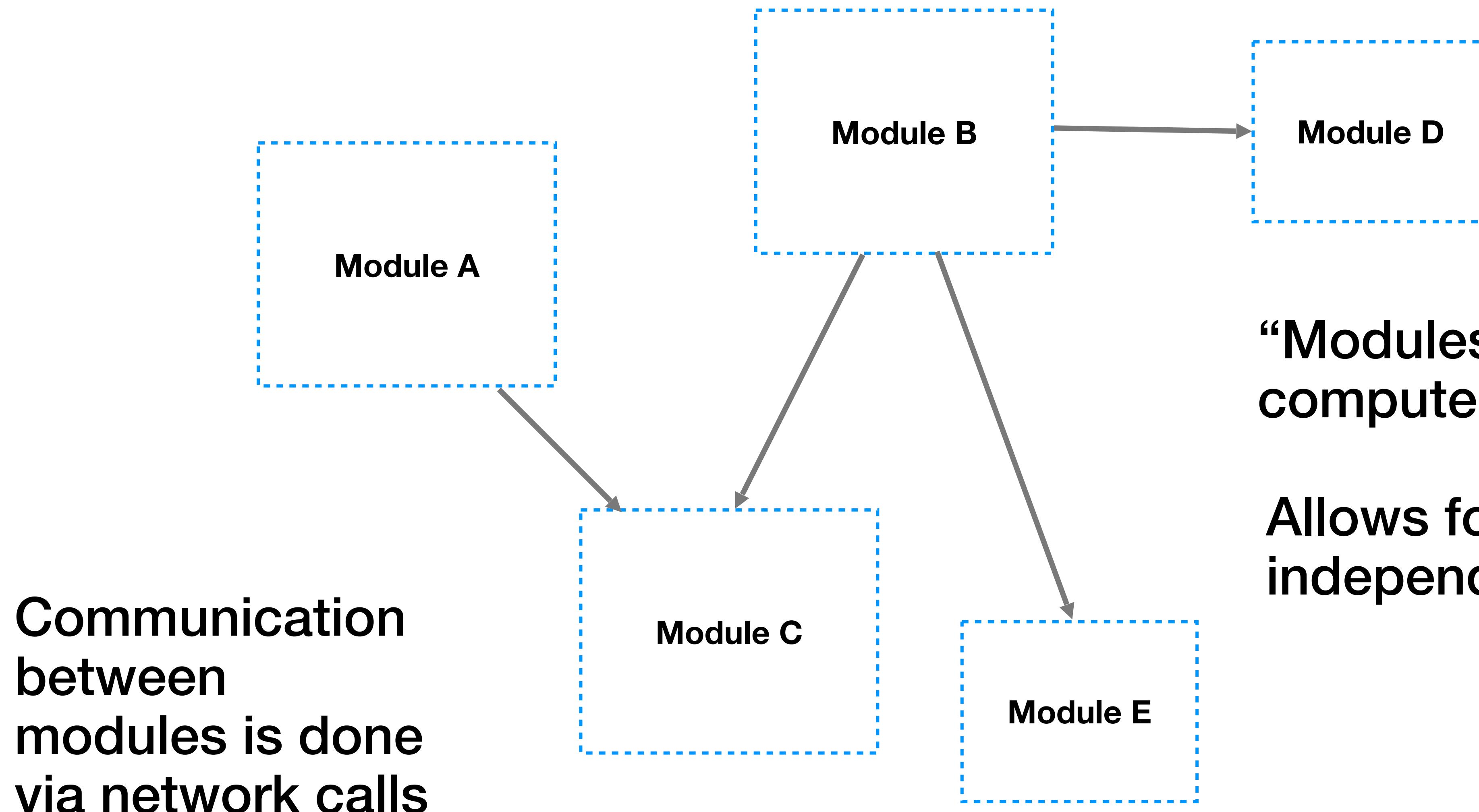


## AND MICROSERVICES?



Communication  
between  
modules is done  
via network calls

## AND MICROSERVICES?



Communication  
between  
modules is done  
via network calls

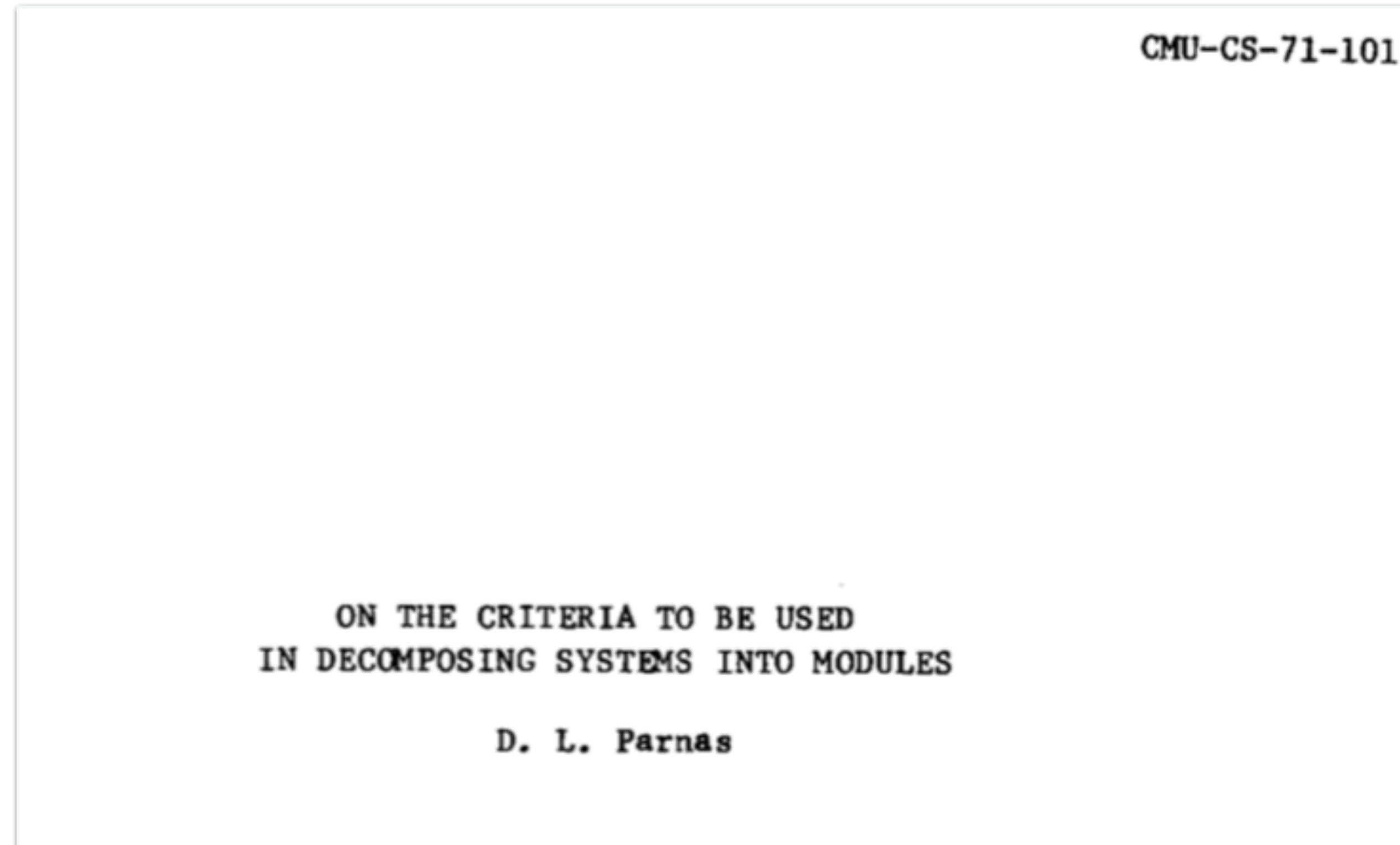
**“Modules” run on different computers**

**Allows for easier independent deployability**

**Microservices, when done right, are a form of modular architecture**



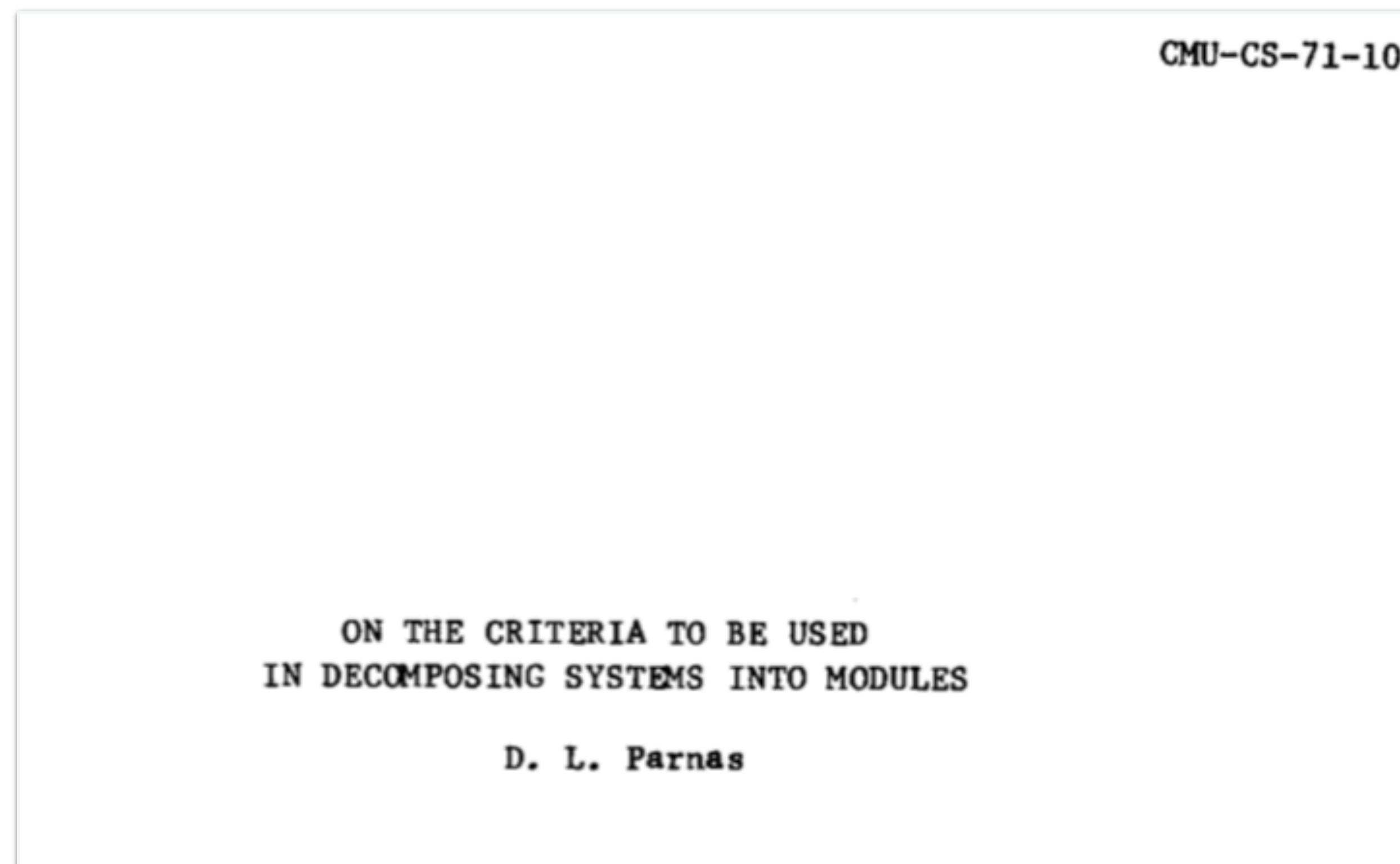
# INFORMATION HIDING



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

@samnewman

# INFORMATION HIDING

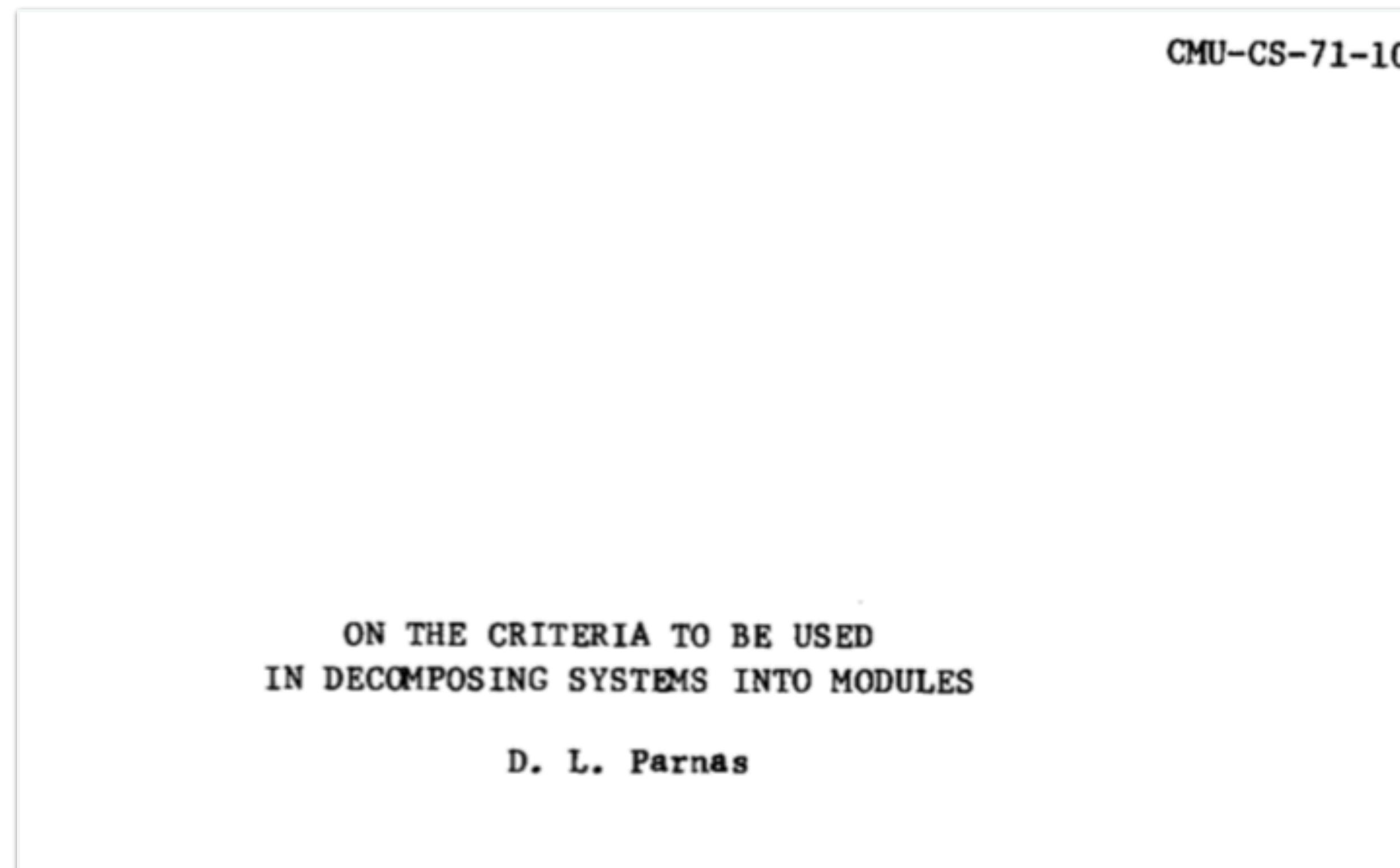


Published in 1971

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

@samnewman

## INFORMATION HIDING



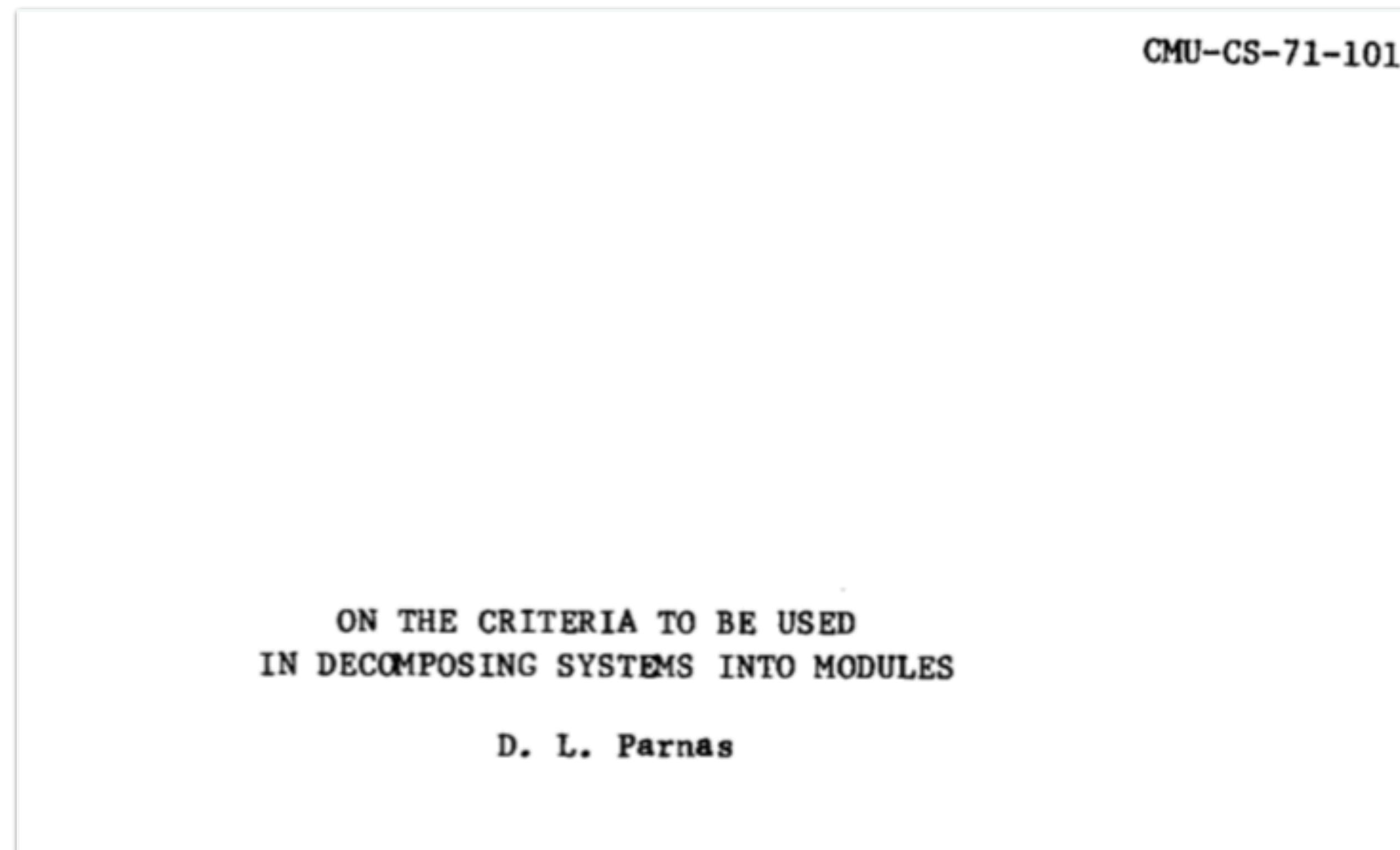
Published in 1971

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

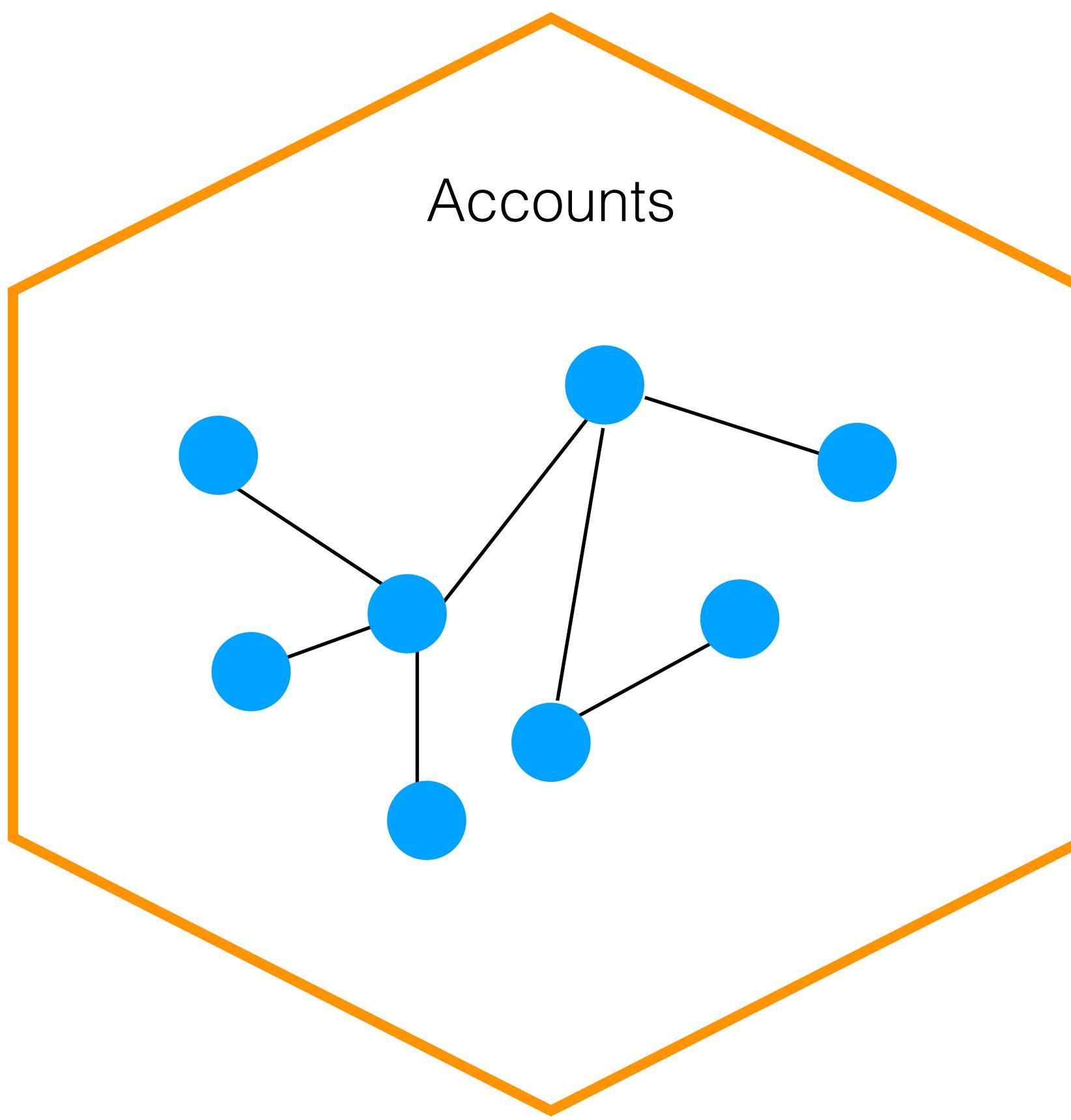
SEPTEMBER 5, 2016

[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.

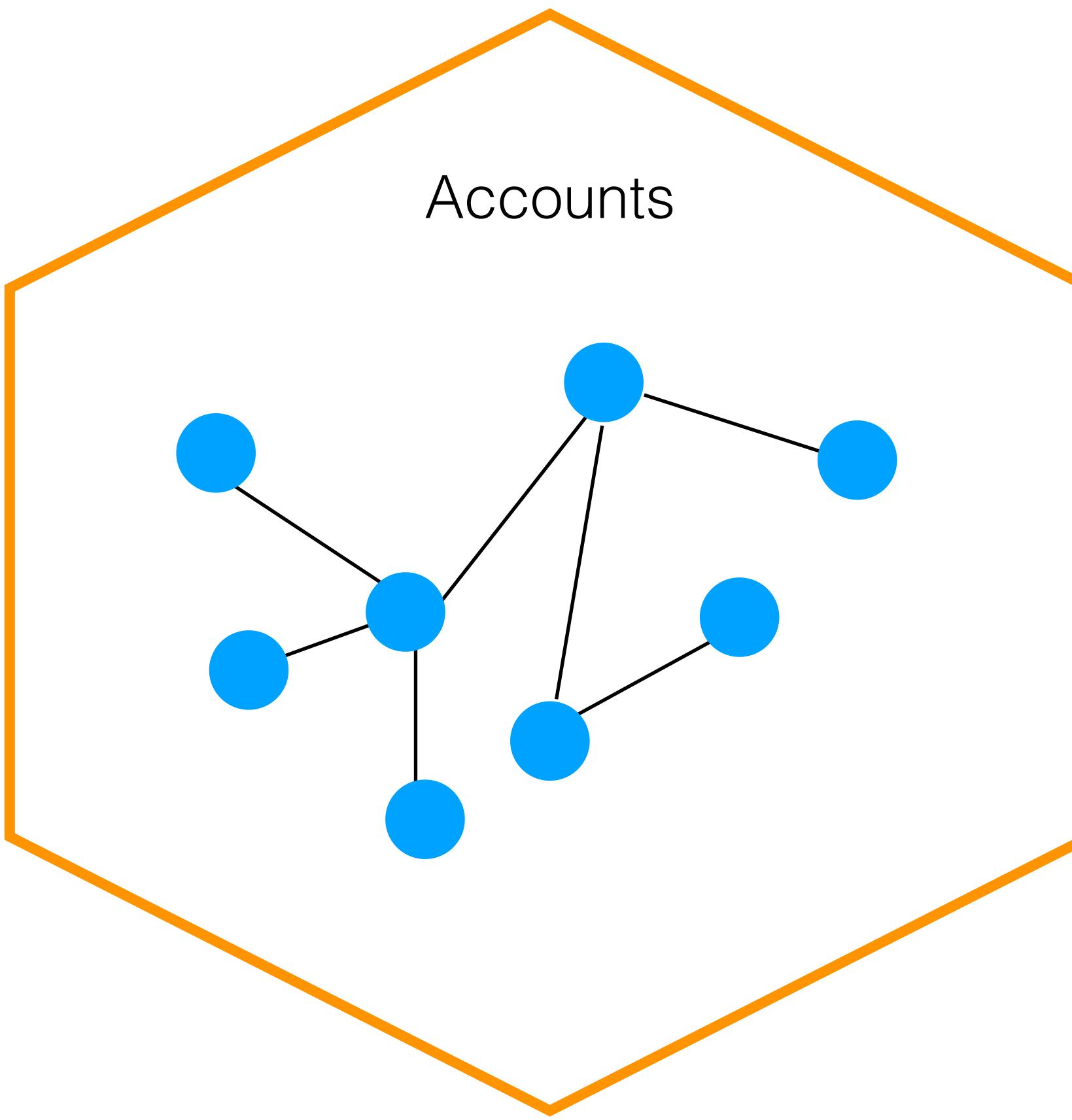
<https://blog.acolyer.org/2016/09/05/on-the-criteria-to-be-used-in-decomposing-systems-into-modules/>

# NOT HIDING?

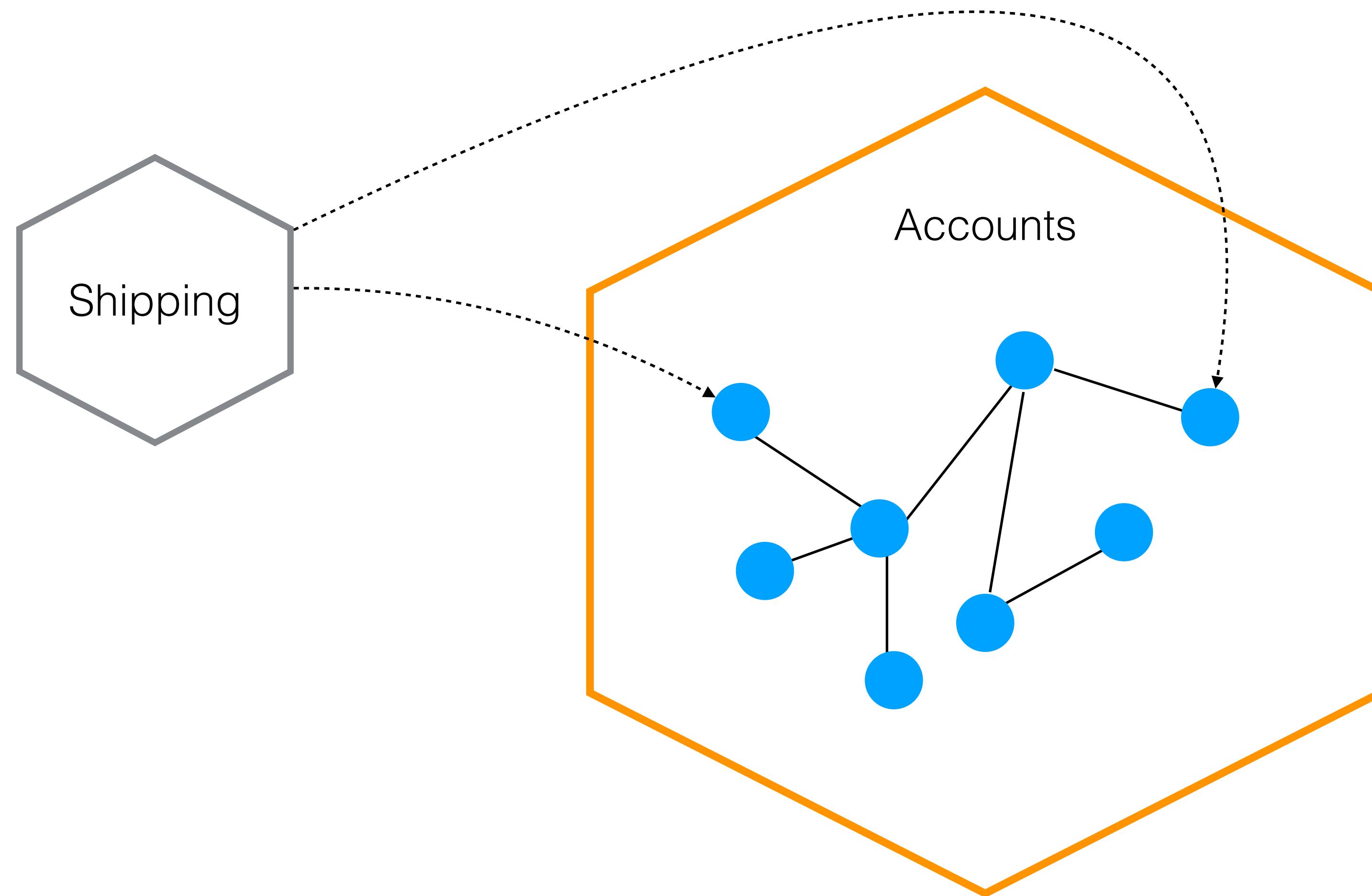


## NOT HIDING?

Shipping

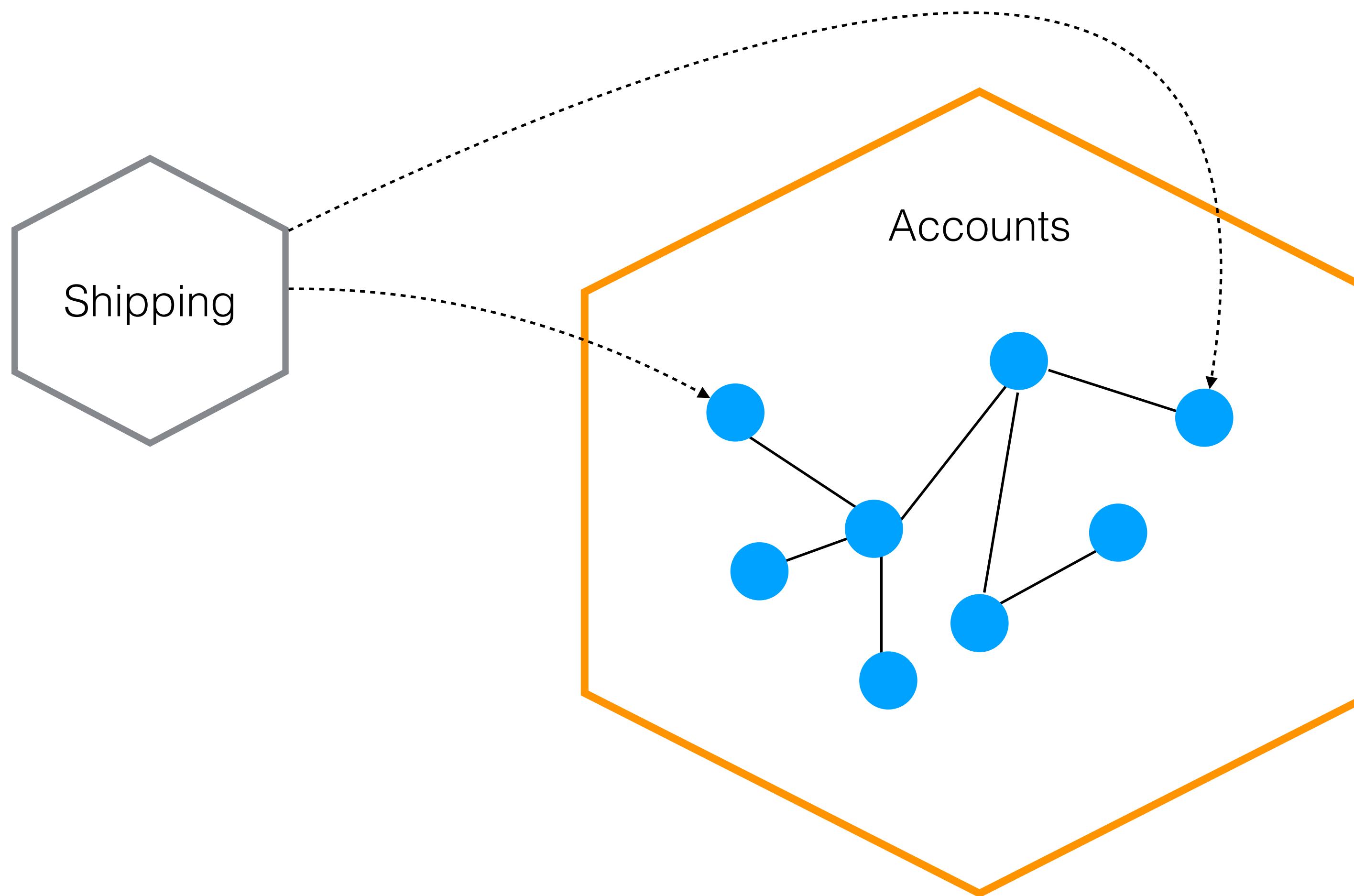


## NOT HIDING?



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

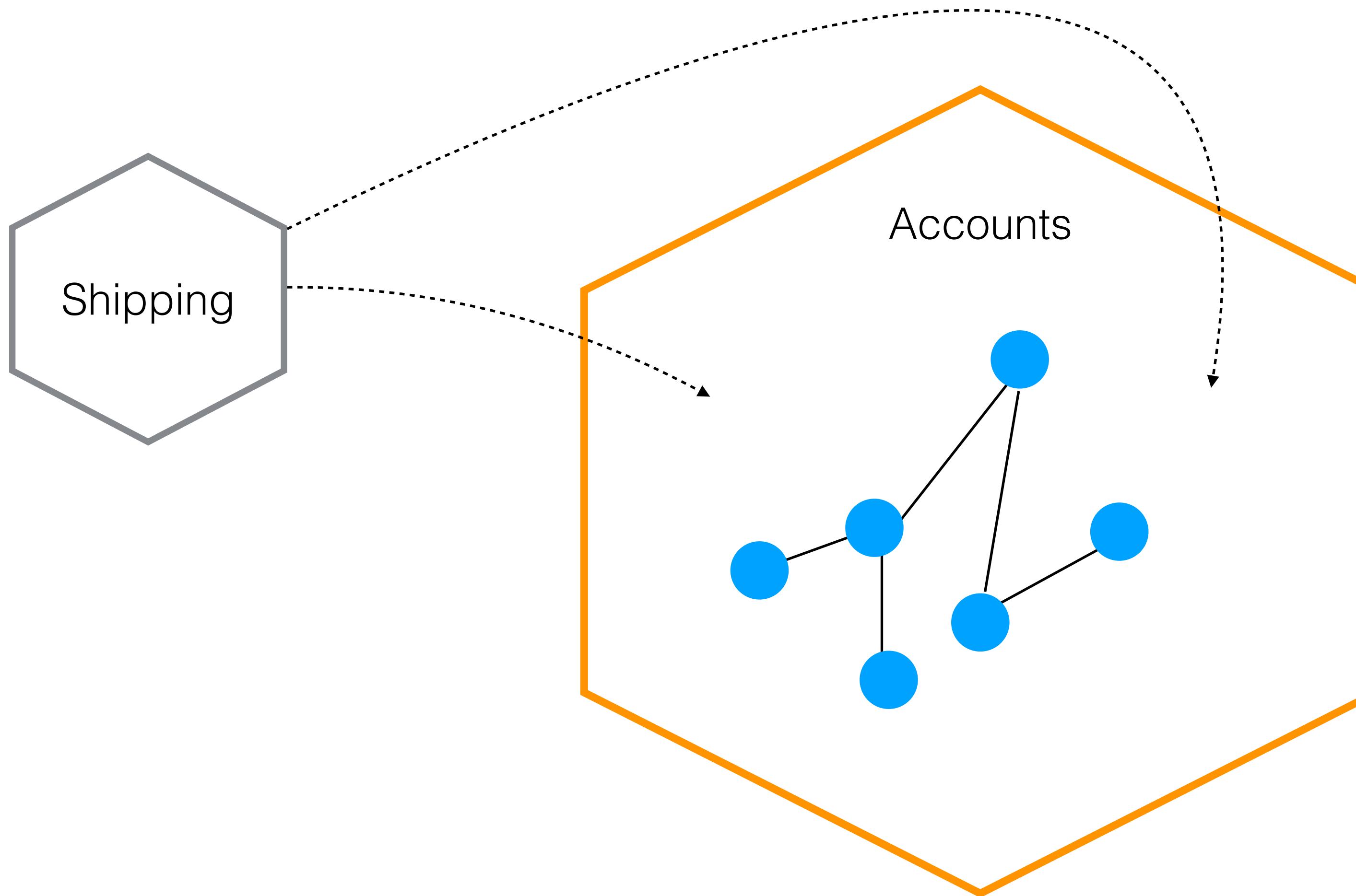
## NOT HIDING?



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

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

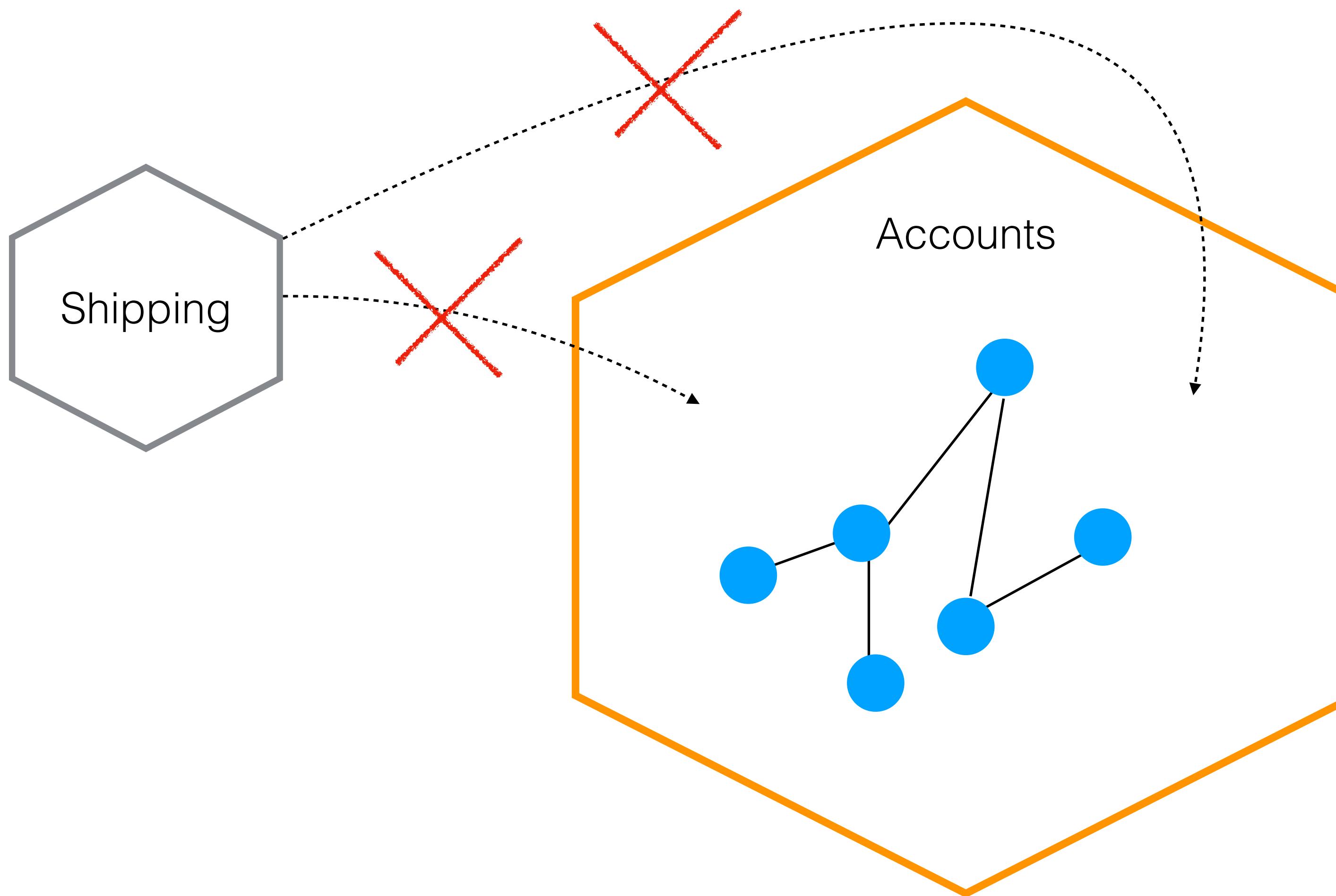
## NOT HIDING?



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

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

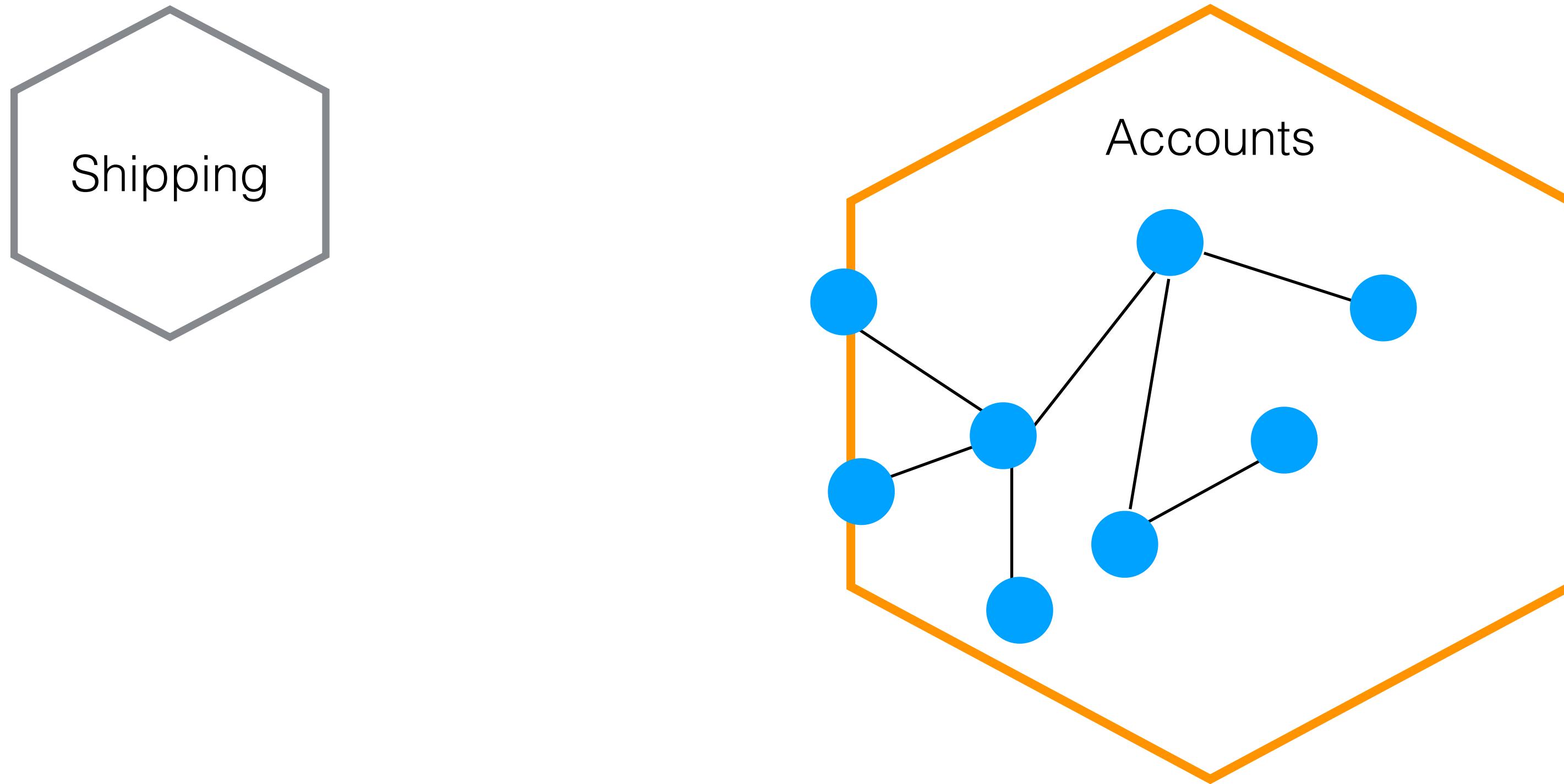
## NOT HIDING?



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

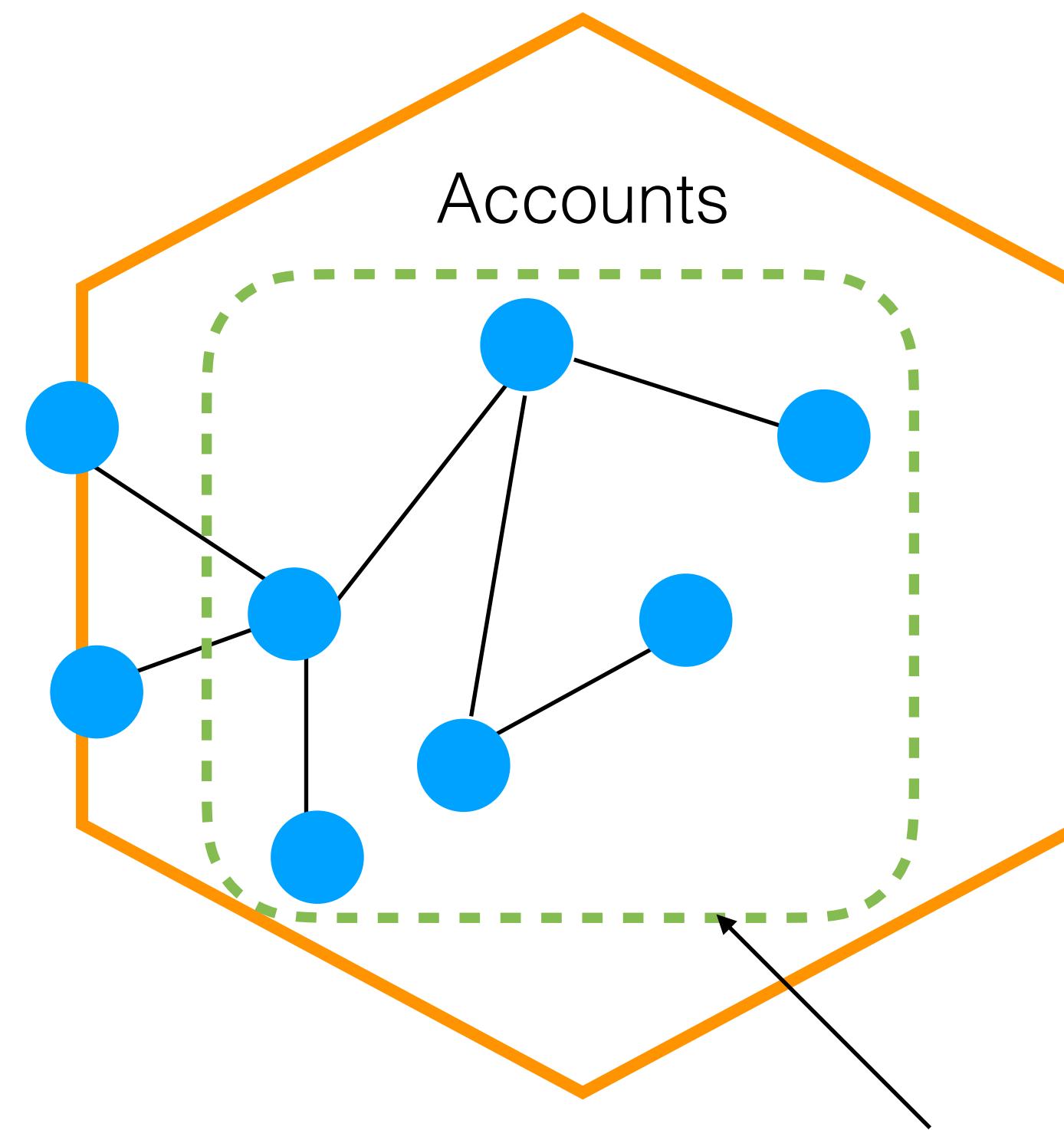
...then you can't change  
this implementation without  
breaking the consumer

# HIDING!



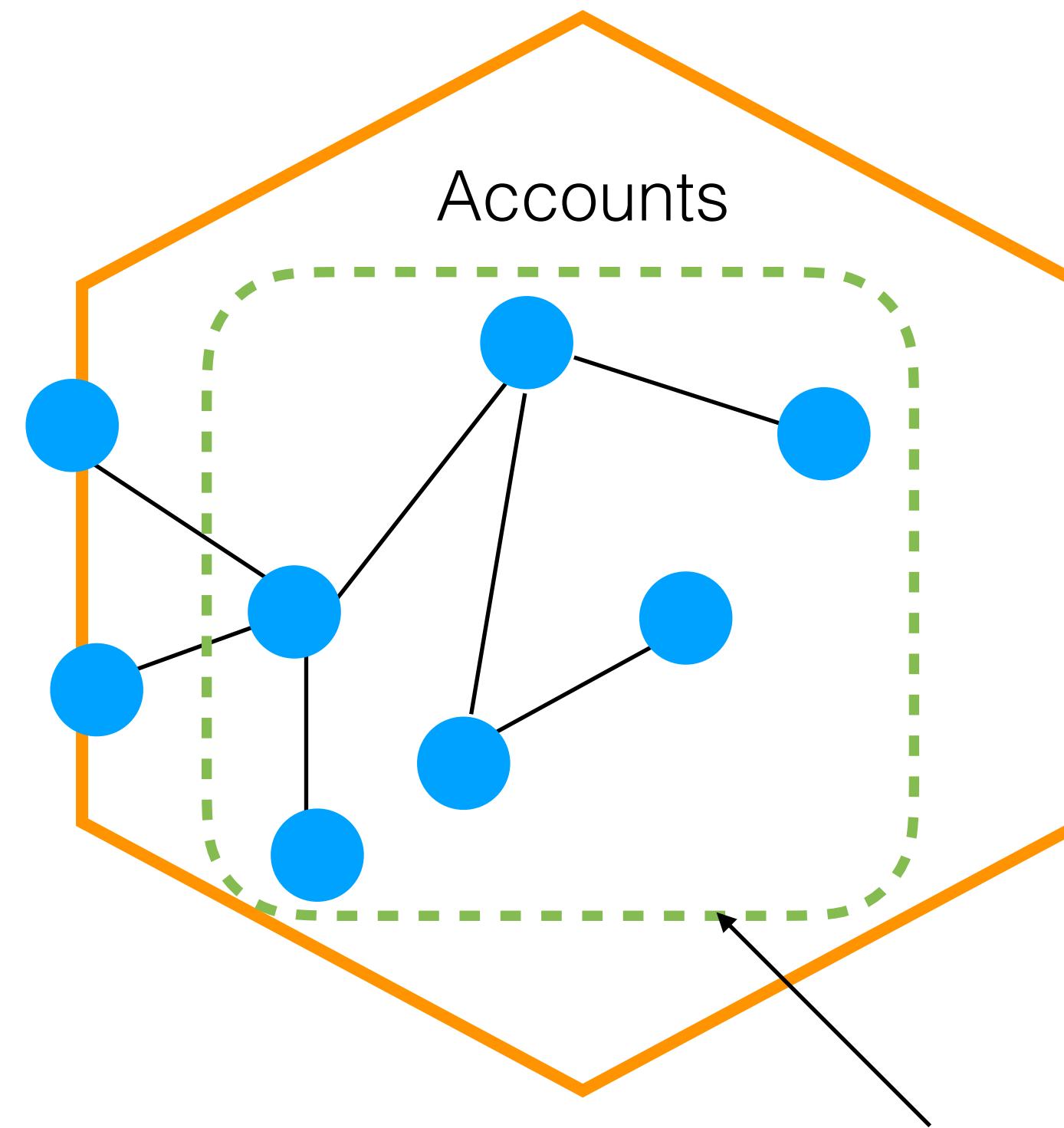
**HIDING!**

**Hide your secrets!**



**Hidden**

# HIDING!

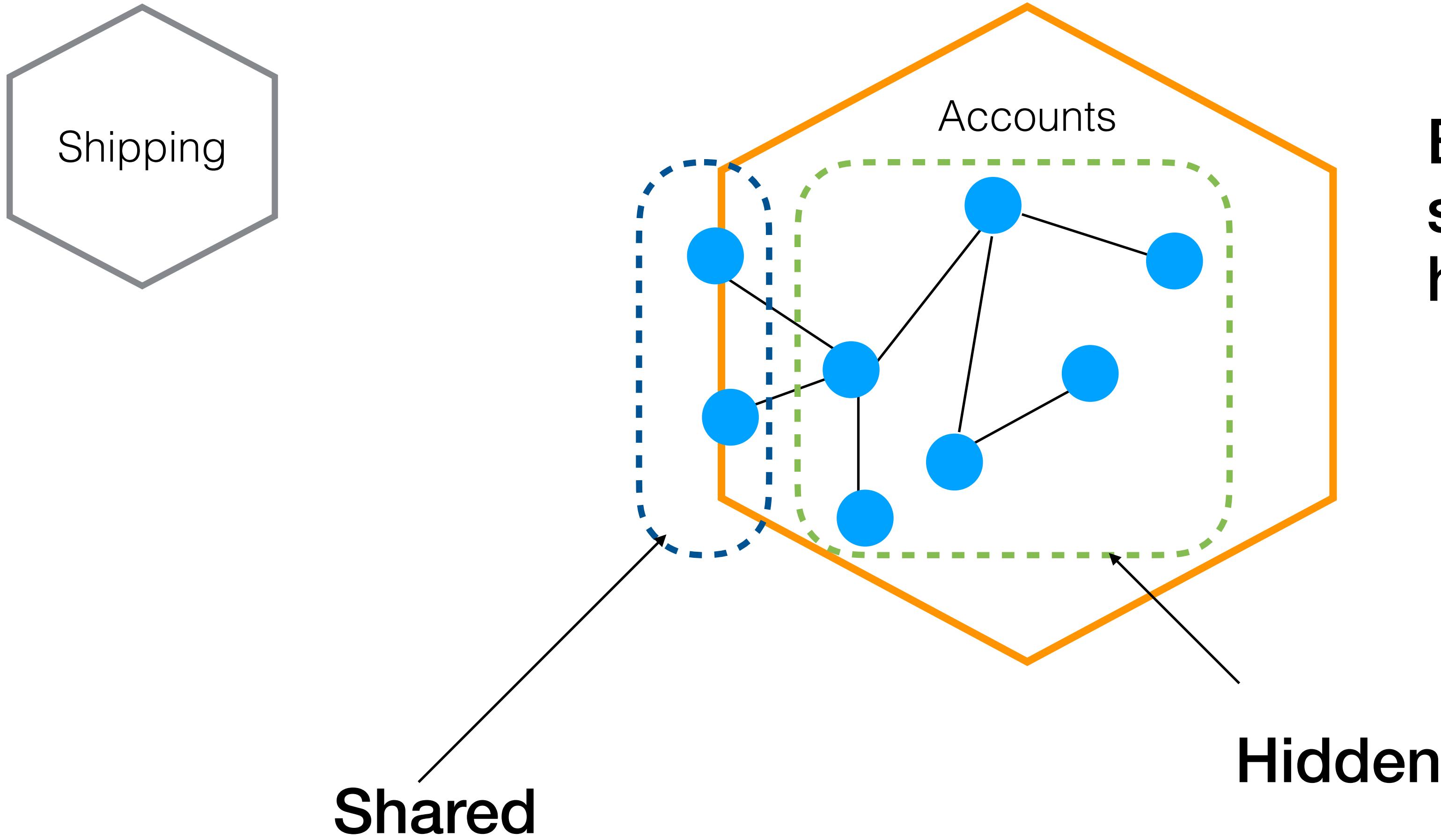


**Hide your secrets!**

**Be explicit about what is shared, and what is hidden**

**Hidden**

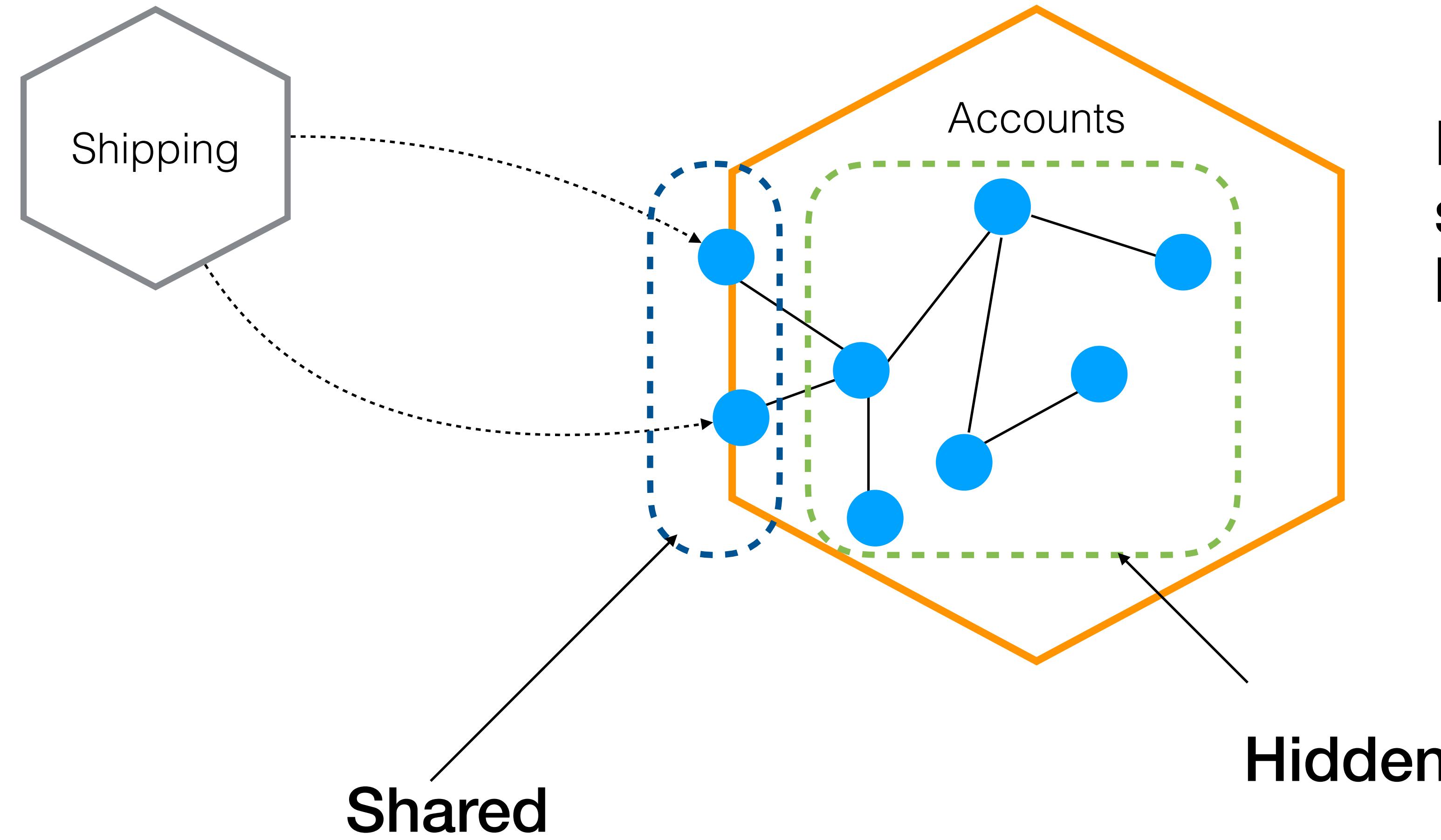
# HIDING!



# Hide your secrets!

**Be explicit about what is shared, and what is hidden**

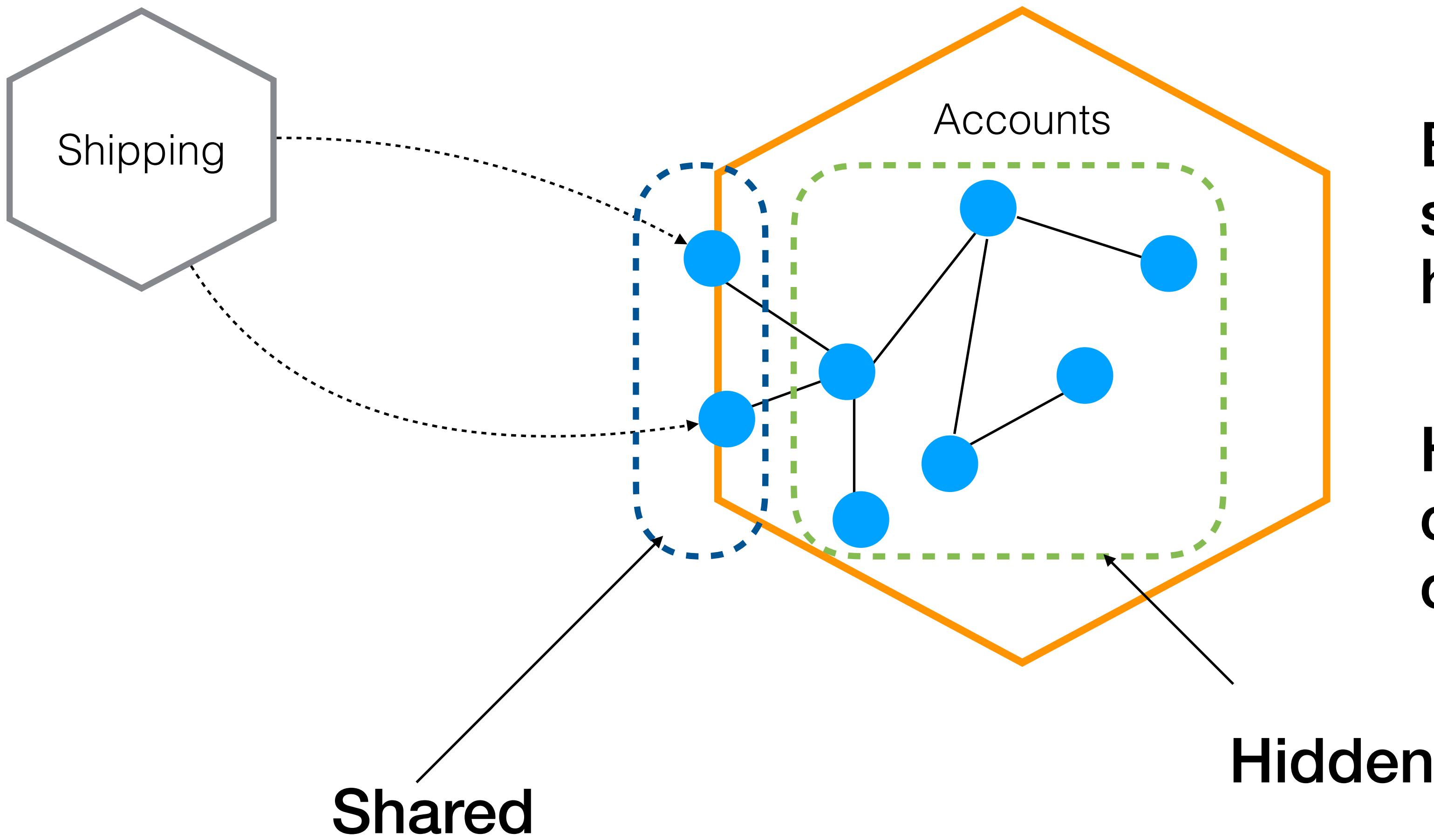
**HIDING!**



**Hide your secrets!**

**Be explicit about what is shared, and what is hidden**

# HIDING!



# Hide your secrets!

**Be explicit about what is shared, and what is hidden**

Hidden things can  
change, shared things  
can't

# Hidden

## ACCIDENTAL BREAKAGE

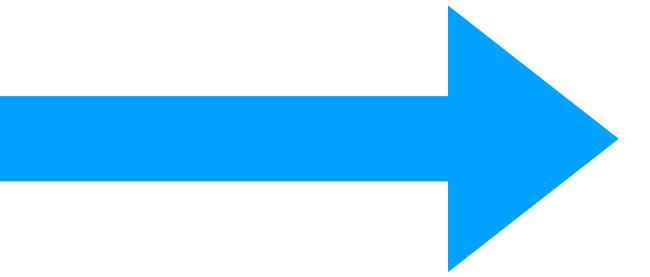
Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```

## ACCIDENTAL BREAKAGE

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```



## ACCIDENTAL BREAKAGE

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```



JSON

```
{  
    "id" : 123,  
    "name" : "sam",  
    "age" : 15  
}
```

## ACCIDENTAL BREAKAGE

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```



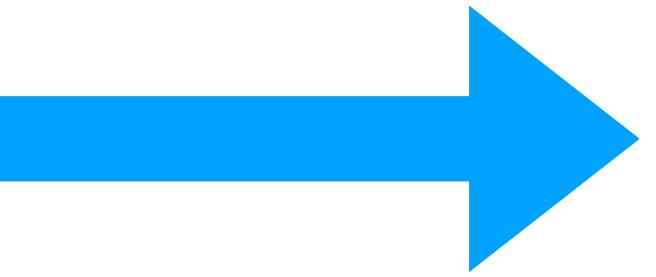
JSON

```
{  
    "id" : 123,  
    "name" : "sam",  
    "age" : 15  
}
```

## ACCIDENTAL BREAKAGE

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```



JSON

```
{  
    "id" : 123,  
    "name" : "sam",  
    "age" : 15  
}
```

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private LocalDate dob;  
    ...  
}
```

## ACCIDENTAL BREAKAGE

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```



JSON

```
{  
    "id" : 123,  
    "name" : "sam",  
    "age" : 15  
}
```

Code

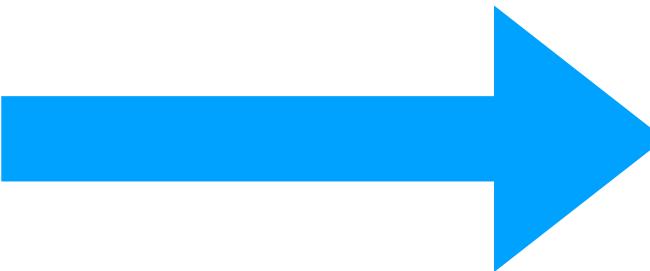
```
public class Customer {  
    private int id;  
    private String name;  
    private LocalDate dob;  
    ...  
}
```



## ACCIDENTAL BREAKAGE

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private int age;  
    ...  
}
```



JSON

```
{  
    "id" : 123,  
    "name" : "sam",  
    "age" : 15  
}
```

Code

```
public class Customer {  
    private int id;  
    private String name;  
    private LocalDate dob;  
    ...  
}
```



JSON

```
{  
    "id" : 123,  
    "name" : "sam",  
    "dob" : "15/02/1974"  
}
```

# DATA TRANSFER OBJECT

| P of EAA Catalog |

## Data Transfer Object

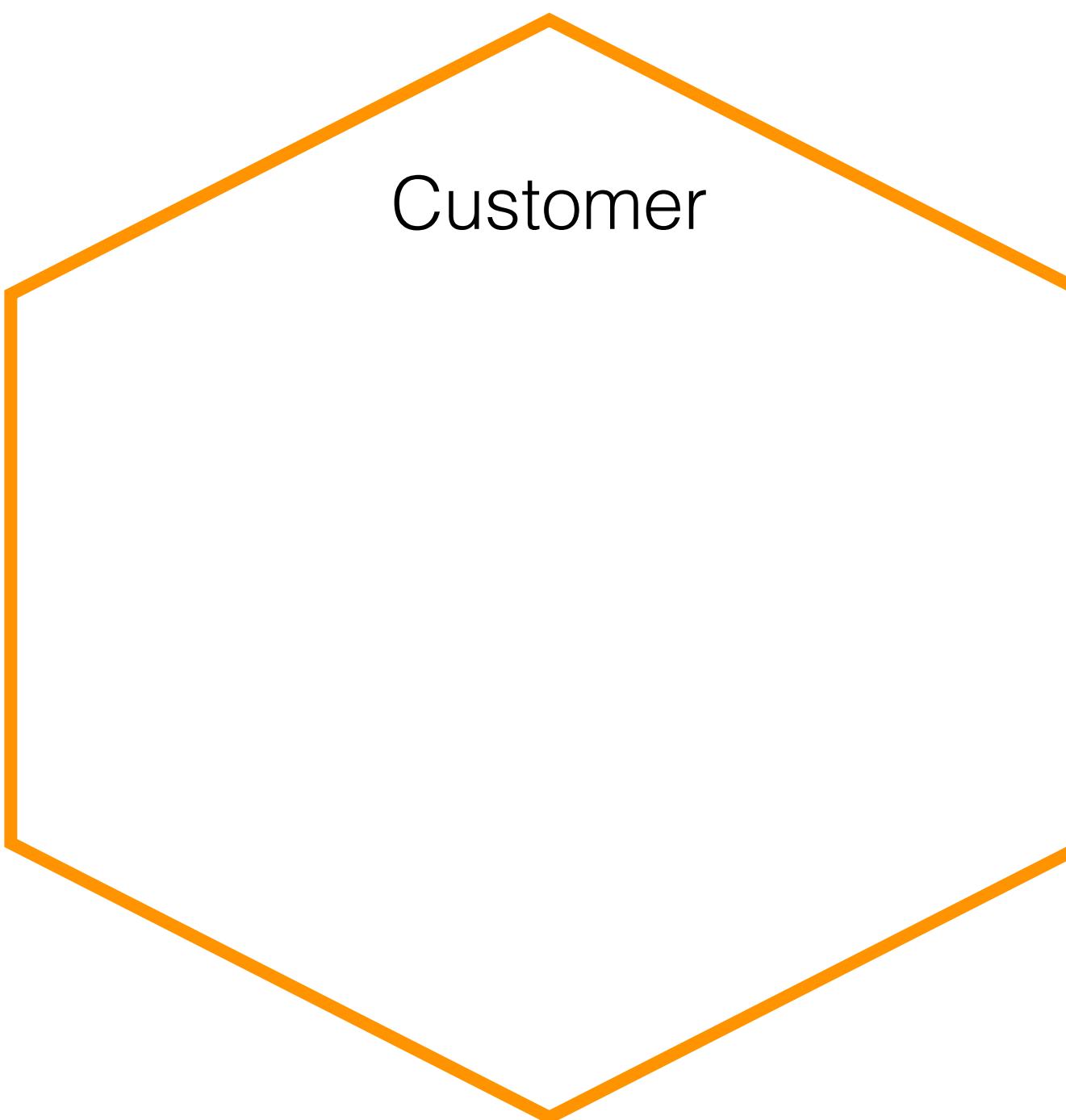
An object that carries data between processes in order to reduce the number of method calls.

For a full description see [P of EAA page 401](#)

```
classDiagram
    class AlbumDTO {
        title: String
        artist: String
        toXmlElement()
        readXml()
    }
    class Album {
        title: String
    }
    class Artist {
        name: String
    }
    class AlbumAssembler {
        --> AlbumDTO
        --> Album
    }
    Album --> Artist
```

When you're working with a remote interface, such as Remote Facade (388), each call to it is expensive. As a result you need to reduce the number of calls, and that means that you need to transfer more data with each call. One way to do this is to use lots of parameters. However, this is often awkward to program - indeed, it's often impossible with languages such as Java that return only a single value.

The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be serializable to go across the connection. Usually an assembler is used on the server side to transfer data between the DTO and any domain objects.



<https://martinfowler.com/eaaCatalog/dataTransferObject.html>

# DATA TRANSFER OBJECT

| P of EAA Catalog |

## Data Transfer Object

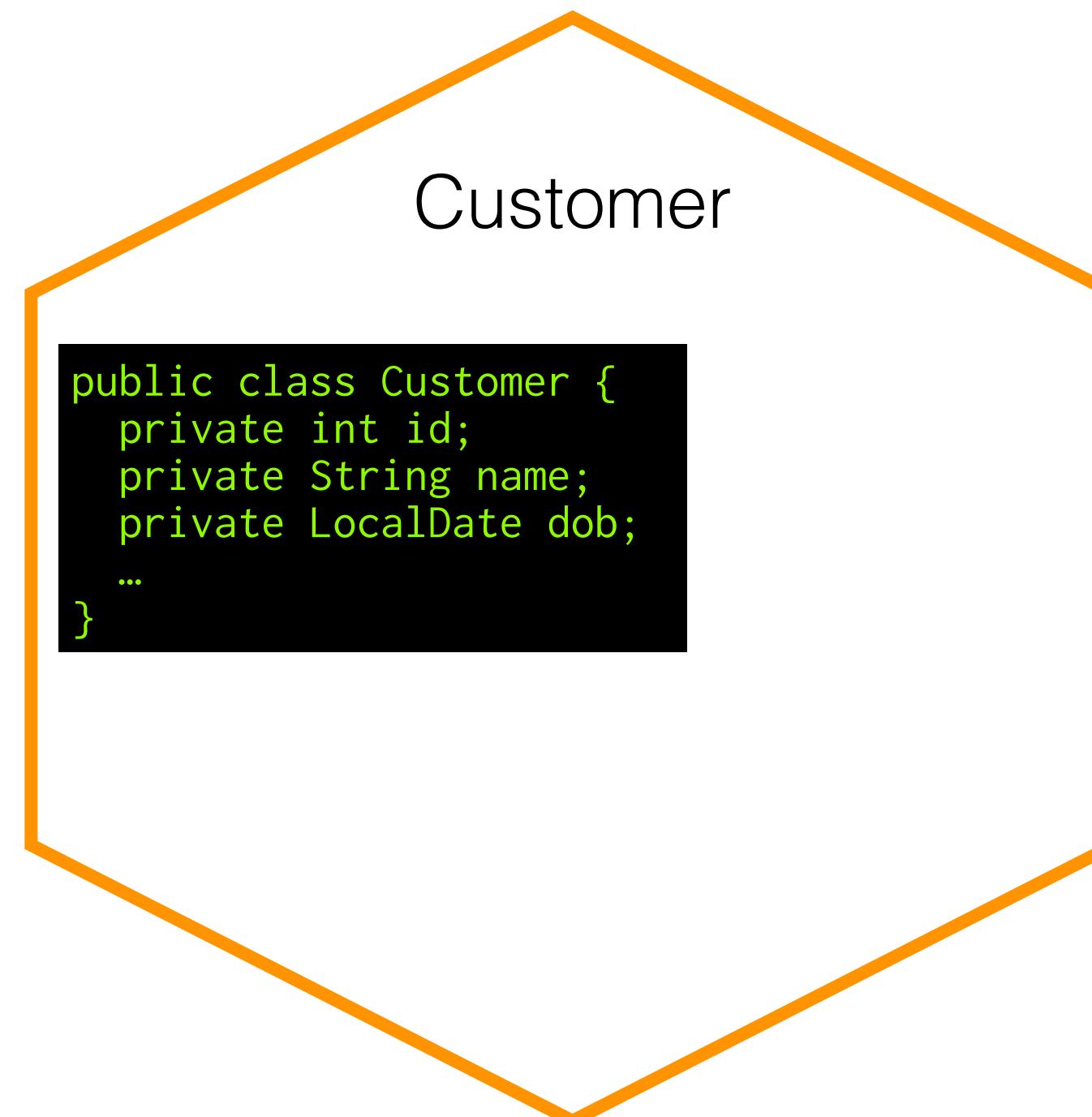
An object that carries data between processes in order to reduce the number of method calls.

For a full description see [P of EAA page 401](#)

```
classDiagram
    class AlbumDTO {
        title: String
        artist: String
    }
    class Album {
        title: String
    }
    class Artist {
        name: String
    }
    class AlbumAssembler {
        <<AlbumAssembler>>
    }
    AlbumAssembler "2" --> "1" Album
    AlbumAssembler "2" --> "1" Artist
    Album "1" --> Artist
```

When you're working with a remote interface, such as Remote Facade (388), each call to it is expensive. As a result you need to reduce the number of calls, and that means that you need to transfer more data with each call. One way to do this is to use lots of parameters. However, this is often awkward to program - indeed, it's often impossible with languages such as Java that return only a single value.

The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be serializable to go across the connection. Usually an assembler is used on the server side to transfer data between the DTO and any domain objects.



<https://martinfowler.com/eaaCatalog/dataTransferObject.html>

# DATA TRANSFER OBJECT

| P of EAA Catalog |

## Data Transfer Object

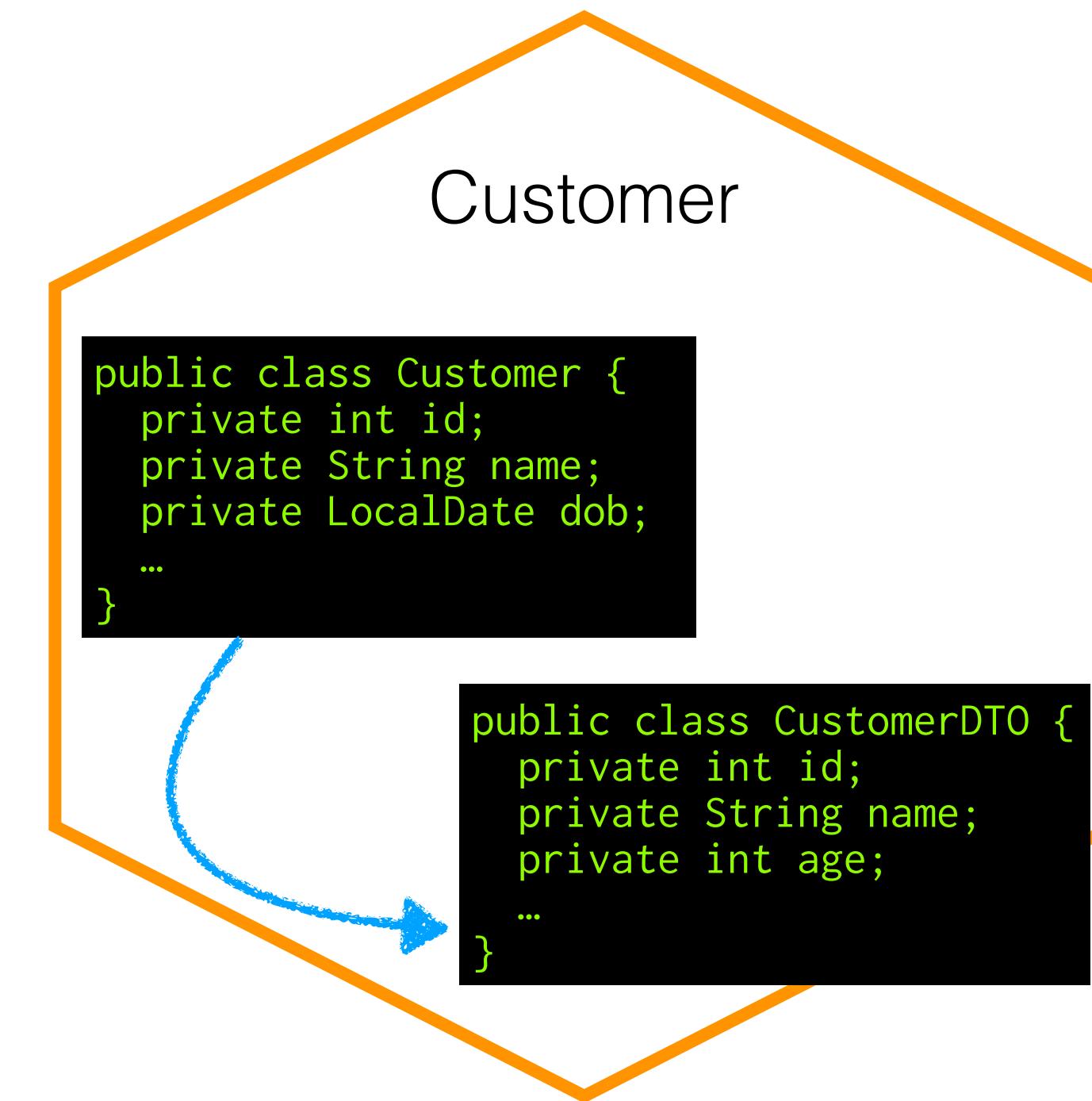
An object that carries data between processes in order to reduce the number of method calls.

For a full description see [P of EAA page 401](#)

```
classDiagram
    class AlbumDTO {
        title: String
        artist: String
        toXmlElement()
        readXml()
    }
    class Album {
        title: String
    }
    class Artist {
        name: String
    }
    class AlbumAssembler {
        --> AlbumDTO
        --> Album
        --> Artist
    }
    AlbumAssembler <--> AlbumDTO
    AlbumAssembler --> Album
    AlbumAssembler --> Artist
```

When you're working with a remote interface, such as Remote Facade (388), each call to it is expensive. As a result you need to reduce the number of calls, and that means that you need to transfer more data with each call. One way to do this is to use lots of parameters. However, this is often awkward to program - indeed, it's often impossible with languages such as Java that return only a single value.

The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be serializable to go across the connection. Usually an assembler is used on the server side to transfer data between the DTO and any domain objects.



<https://martinfowler.com/eaaCatalog/dataTransferObject.html>

# DATA TRANSFER OBJECT

| P of EAA Catalog |

## Data Transfer Object

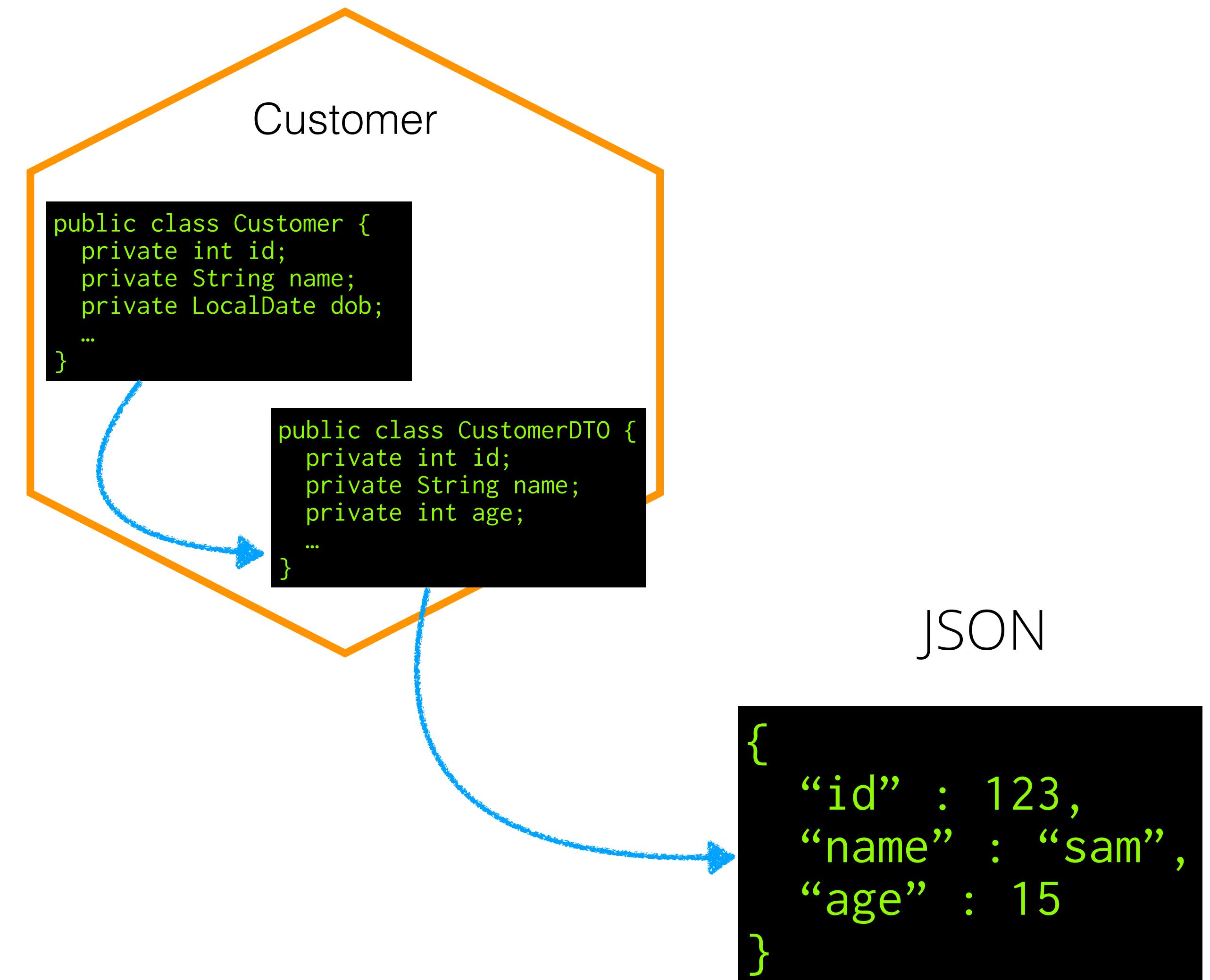
An object that carries data between processes in order to reduce the number of method calls.

For a full description see [P of EAA page 401](#)

```
classDiagram
    class AlbumDTO {
        title: String
        artist: String
        toXmlElement()
        readXml()
    }
    class Album {
        title: String
    }
    class Artist {
        name: String
    }
    class AlbumAssembler {
        --> AlbumDTO
        --> Album
        --> Artist
    }
    Album --> Artist
    Artist --> Artist
```

When you're working with a remote interface, such as Remote Facade (388), each call to it is expensive. As a result you need to reduce the number of calls, and that means that you need to transfer more data with each call. One way to do this is to use lots of parameters. However, this is often awkward to program - indeed, it's often impossible with languages such as Java that return only a single value.

The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be serializable to go across the connection. Usually an assembler is used on the server side to transfer data between the DTO and any domain objects.



<https://martinfowler.com/eaaCatalog/dataTransferObject.html>

# AND CONNECTIONS BETWEEN THEM?

## INFORMATION DISTRIBUTION ASPECTS OF DESIGN METHODOLOGY \*

D.L. PARNA

*Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA*

The role of documentation in the design and implementation of complex systems is explored, resulting in suggestions in sharp contrast with current practice. The concept of system structure is studied by examining the meaning of the phrase "connections between modules". It is shown that several system design goals (each suggesting a partial time ordering of the decisions) may be inconsistent. Some properties of programmers are discussed. System documentation which makes all information accessible to anyone working on the project, is discussed. The thesis that such information "broadcasting" is harmful, that it is helpful if most system information can be hidden from most programmers, is supported by use of the above mentioned considerations as well as by examples.

### 1. INTRODUCTION

Papers on design methodology assume (1) that the methods used in system design strongly affect the quality of the final product; and (2) by selecting an appropriate methodology we can avoid many of the problems previously encountered in constructing large systems.

Under the heading "Design Methodology" a number of separate topics can be distinguished:

1. The order in which design decisions are made [1, 2, 3, 6].
2. The characteristics of the final product (e.g., what constitutes "good structure" for a system) [4, 5, 6, 7].
3. Methods of detecting errors in design decisions shortly after they are made [1, 2, 3, 5, 8, 9].
4. Specification techniques [12, 13].

### 2. STRUCTURE DEFINED

The word "structure" is used to refer to a partial description of a system. A structure description shows the system divided into a set of *modules*, gives some characteristics of each module, and specifies some *connections* between the modules. Any given system admits many such descriptions. Since structure descriptions are not unique, our usage of "module" does not allow a precise definition parallel to that of "subroutine" in software or "card" in hardware. The definitions of those words delineate a class of object, but not the definition of "module". Nevertheless, "module" is useful in the same manner that "unit" is in military or economic discussions. We shall continue to use "module" without a precise definition. It refers to portions of a system indicated in a description of that system. Its precise definition is not

# AND CONNECTIONS BETWEEN THEM?

## INFORMATION DISTRIBUTION ASPECTS OF DESIGN METHODOLOGY \*

D.L. PARNA

*Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA*

The role of documentation in the design and implementation of complex systems is explored, resulting in suggestions in sharp contrast with current practice. The concept of system structure is studied by examining the meaning of the phrase "connections between modules". It is shown that several system design goals (each suggesting a partial time ordering of the decisions) may be inconsistent. Some properties of programmers are discussed. System documentation which makes all information accessible to anyone working on the project, is discussed. The thesis that such information "broadcasting" is harmful, that it is helpful if most system information can be hidden from most programmers, is supported by use of the above mentioned considerations as well as by examples.

### 1. INTRODUCTION

Papers on design methodology assume (1) that the methods used in system design strongly affect the quality of the final product; and (2) by selecting an appropriate methodology we can avoid many of the problems previously encountered in constructing large systems.

Under the heading "Design Methodology" a number of separate topics can be distinguished:

1. The order in which design decisions are made [1, 2, 3, 6].
2. The characteristics of the final product (e.g., what constitutes "good structure" for a system) [4, 5, 6, 7].
3. Methods of detecting errors in design decisions shortly after they are made [1, 2, 3, 5, 8, 9].
4. Specification techniques [12, 13].

### 2. STRUCTURE DEFINED

The word "structure" is used to refer to a partial description of a system. A structure description shows the system divided into a set of *modules*, gives some characteristics of each module, and specifies some *connections* between the modules. Any given system admits many such descriptions. Since structure descriptions are not unique, our usage of "module" does not allow a precise definition parallel to that of "subroutine" in software or "card" in hardware. The definitions of those words delineate a class of object, but not the definition of "module". Nevertheless, "module" is useful in the same manner that "unit" is in military or economic discussions. We shall continue to use "module" without a precise definition. It refers to portions of a system indicated in a description of that system. Its precise definition is not

**"The connections between modules are the assumptions which the modules make about each other."**

- David Parnas

# AND CONNECTIONS BETWEEN THEM?

## INFORMATION DISTRIBUTION ASPECTS OF DESIGN METHODOLOGY \*

D.L. PARNA

*Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA*

The role of documentation in the design and implementation of complex systems is explored, resulting in suggestions in sharp contrast with current practice. The concept of system structure is studied by examining the meaning of the phrase "connections between modules". It is shown that several system design goals (each suggesting a partial time ordering of the decisions) may be inconsistent. Some properties of programmers are discussed. System documentation which makes all information accessible to anyone working on the project, is discussed. The thesis that such information "broadcasting" is harmful, that it is helpful if most system information can be hidden from most programmers, is supported by use of the above mentioned considerations as well as by examples.

### 1. INTRODUCTION

Papers on design methodology assume (1) that the methods used in system design strongly affect the quality of the final product; and (2) by selecting an appropriate methodology we can avoid many of the problems previously encountered in constructing large systems.

Under the heading "Design Methodology" a number of separate topics can be distinguished:

1. The order in which design decisions are made [1, 2, 3, 6].
2. The characteristics of the final product (e.g., what constitutes "good structure" for a system) [4, 5, 6, 7].
3. Methods of detecting errors in design decisions shortly after they are made [1, 2, 3, 5, 8, 9].
4. Specification techniques [12, 13].

### 2. STRUCTURE DEFINED

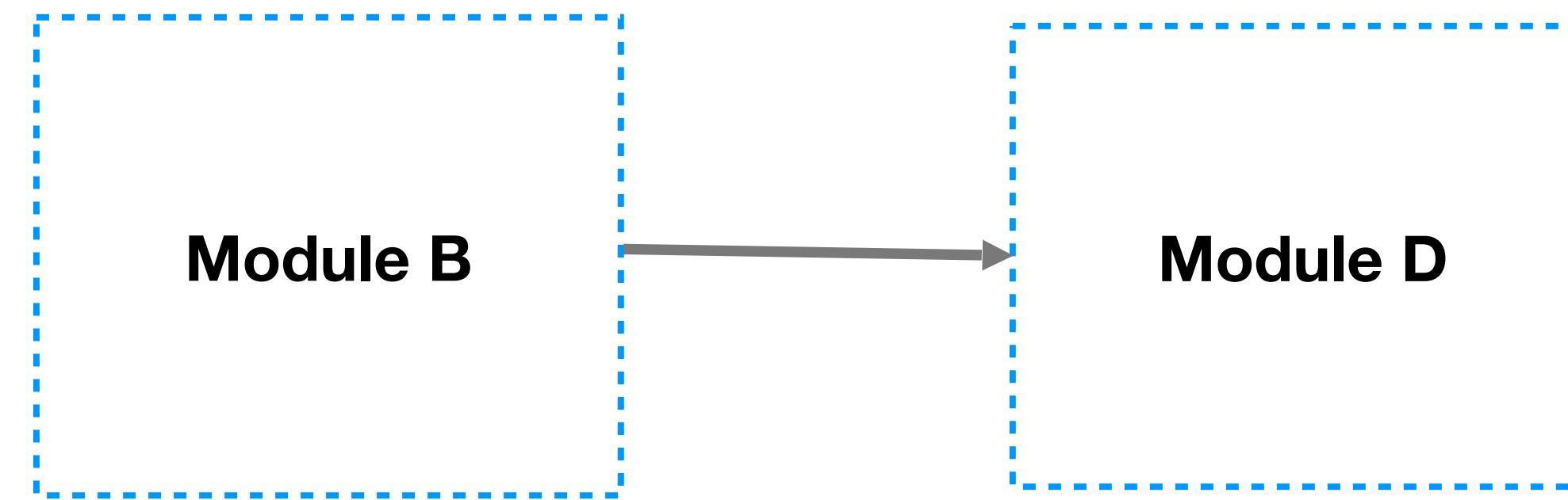
The word "structure" is used to refer to a partial description of a system. A structure description shows the system divided into a set of *modules*, gives some characteristics of each module, and specifies some *connections* between the modules. Any given system admits many such descriptions. Since structure descriptions are not unique, our usage of "module" does not allow a precise definition parallel to that of "subroutine" in software or "card" in hardware. The definitions of those words delineate a class of object, but not the definition of "module". Nevertheless, "module" is useful in the same manner that "unit" is in military or economic discussions. We shall continue to use "module" without a precise definition. It refers to portions of a system indicated in a description of that system. Its precise definition is not

**"The connections between modules are the assumptions which the modules make about each other."**

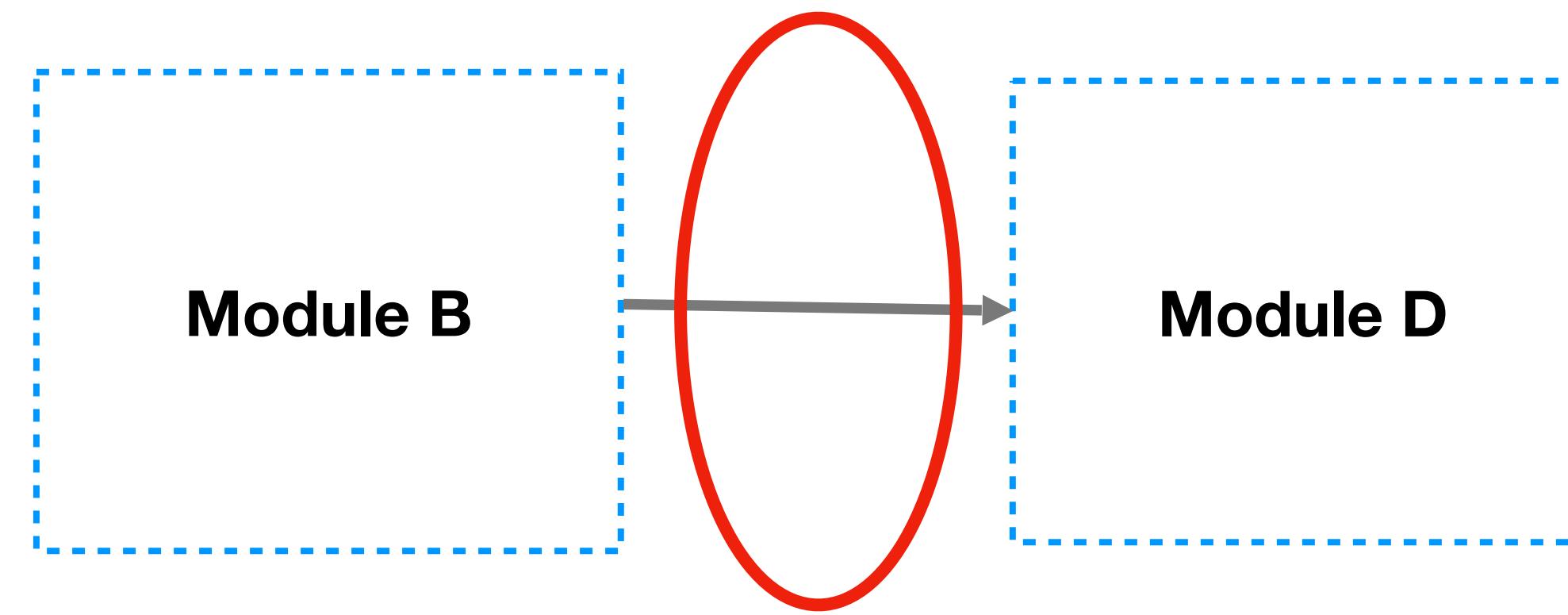
- David Parnas

**Information hiding is about reducing the assumptions one module has for another**

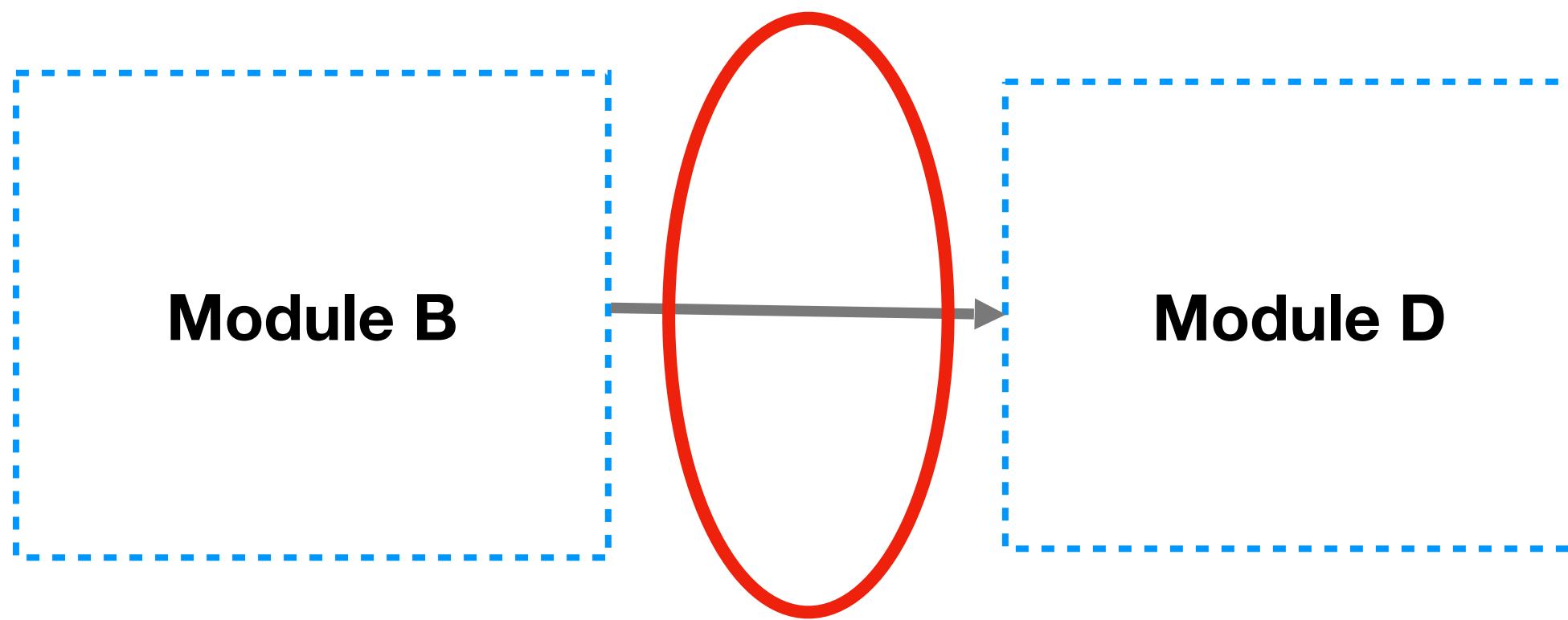
# EXPLICIT CONTRACTS?



## EXPLICIT CONTRACTS?

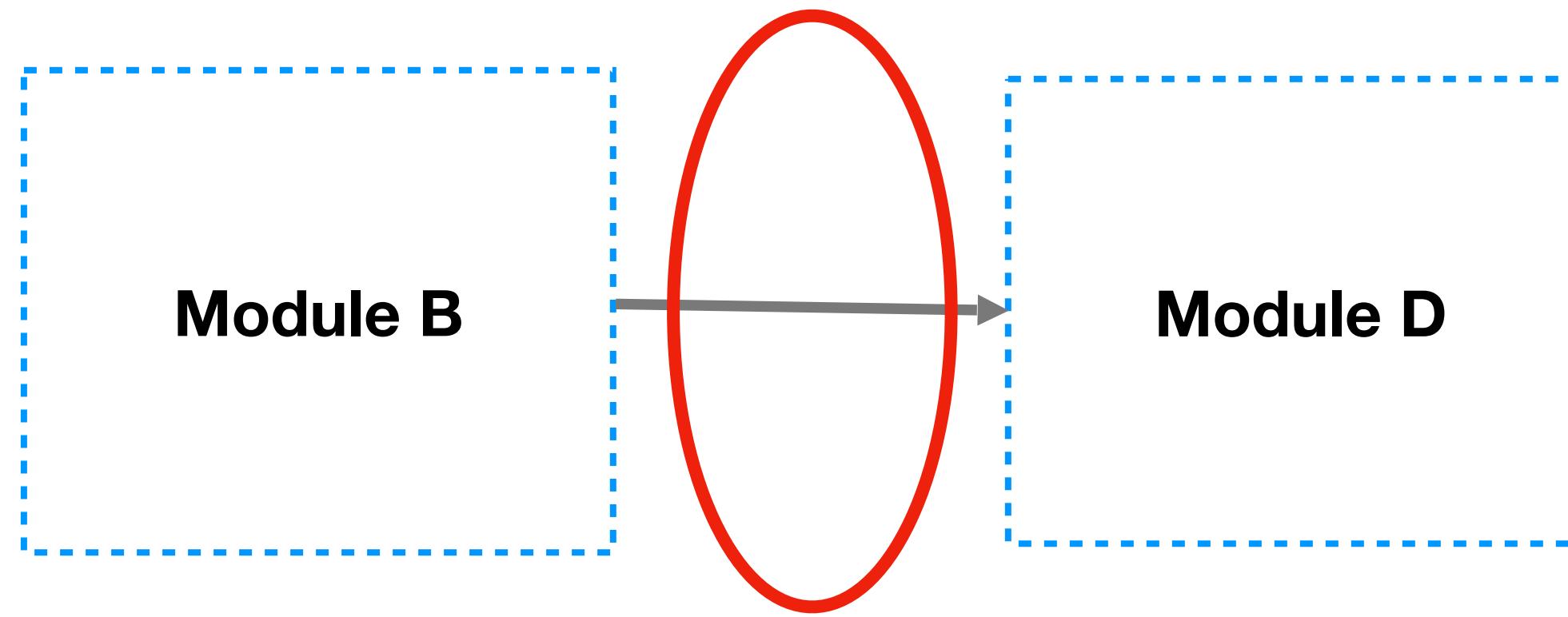


## EXPLICIT CONTRACTS?



Need an explicit understanding of  
the assumptions of consumers

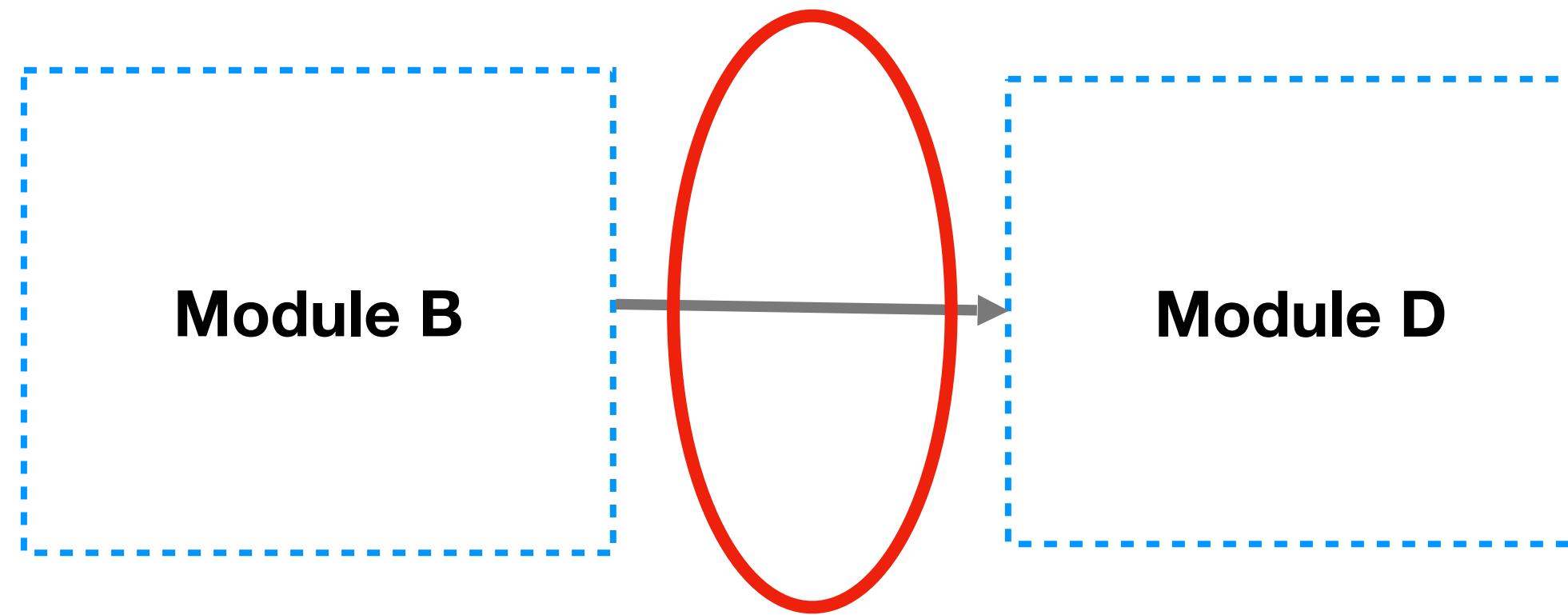
## EXPLICIT CONTRACTS?



Need an explicit understanding of the assumptions of consumers

To help us understand what can change easily, and what cannot

## EXPLICIT CONTRACTS?



Need an explicit understanding of the assumptions of consumers

To help us understand what can change easily, and what cannot

Schemas can help!

# **Schemas FTW!**

# SCHEMAS

## JSON Schema

NEW DRAFT PUBLISHED!

The current version is [2019-09](#)!

**JSON Schema** is a vocabulary that allows you to **annotate** and **validate** JSON documents.

## Advantages

### JSON Schema

- Describes your existing data format(s).
- Provides clear human- and machine- readable documentation.
- Validates data which is useful for:
  - Automated testing.
  - Ensuring quality of client submitted data.

### JSON Hyper-Schema

- Make any JSON format a hypermedia format with no constraints on document structure
- Allows use of [URI Templates](#) with instance data
- Describe client data for use with links using JSON Schema.
- Recognizes collections and collection items.

## Project Status

16 September 2019: Draft 2019-09 (formerly known as draft-08) has been published!

The IETF document IDs are of the form `draft-handrews-*-02`. We are now using dates for meta-schemas, which are what implementations should use to determine behavior, so we will usually refer to [2019-09](#) (without the word "draft") on this web site.

<https://json-schema.org/>

```
syntax = "proto2";

package tutorial;

option java_package = "com.example.tutorial";
option java_outer_classname = "AddressBookProtos";

message Person {
    required string name = 1;
    required int32 id = 2;
    optional string email = 3;

    enum PhoneType {
        MOBILE = 0;
        HOME = 1;
        WORK = 2;
    }

    message PhoneNumber {
        required string number = 1;
        optional PhoneType type = 2 [default = HOME];
    }

    repeated PhoneNumber phones = 4;
}

message AddressBook {
    repeated Person people = 1;
}
```

<https://developers.google.com/protocol-buffers/>

@samnewman

# CHECKING SCHEMA COMPATIBILITY

## Protolock

 PASSED godoc reference docker build passing

Protocol Buffer companion tool. Track your .proto files and prevent changes to messages and services which impact API compatibility.

Source: [github.com/nilslice/protolock](https://github.com/nilslice/protolock)

### DEMO

#### Step 1: Define a .proto

Either use the demo `demo.proto` or write your own here:

```
demo.proto
syntax = "proto3";
option java_multiple_files = true;
option java_outer_classname = "DemoProto";
option java_package = "com.nilslice.demo";

enum ServingStatus {
    UNKNOWN = 0;
    SERVING = 1;
    NOT_SERVING = 2;
    SERVICE_UNKNOWN = 3;
}
ServingStatus status = 1;

service Health {
    rpc Check(HealthCheckRequest) returns (HealthCheckResponse);
    rpc Watch(HealthCheckRequest) returns (stream HealthCheckResponse);
}
```

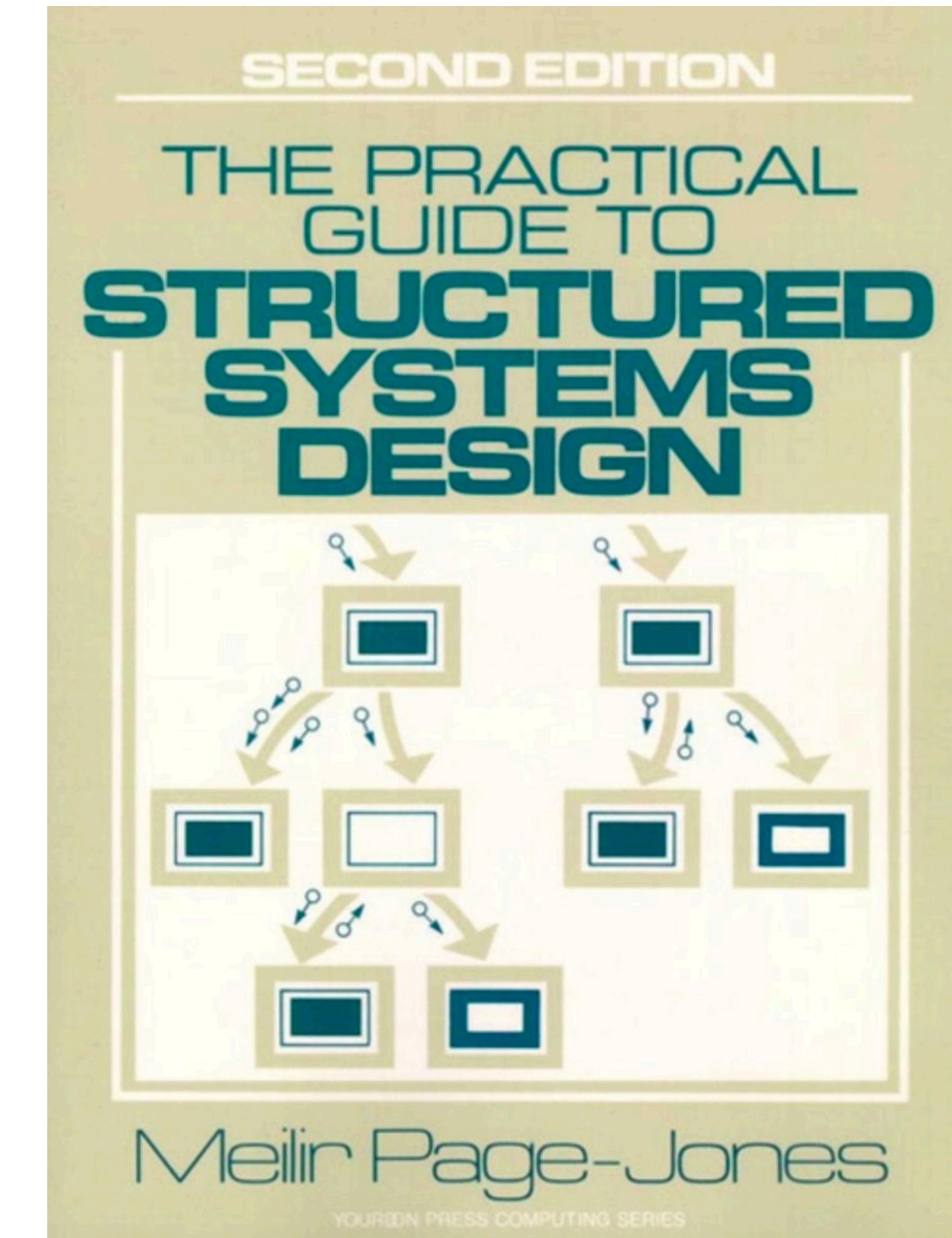
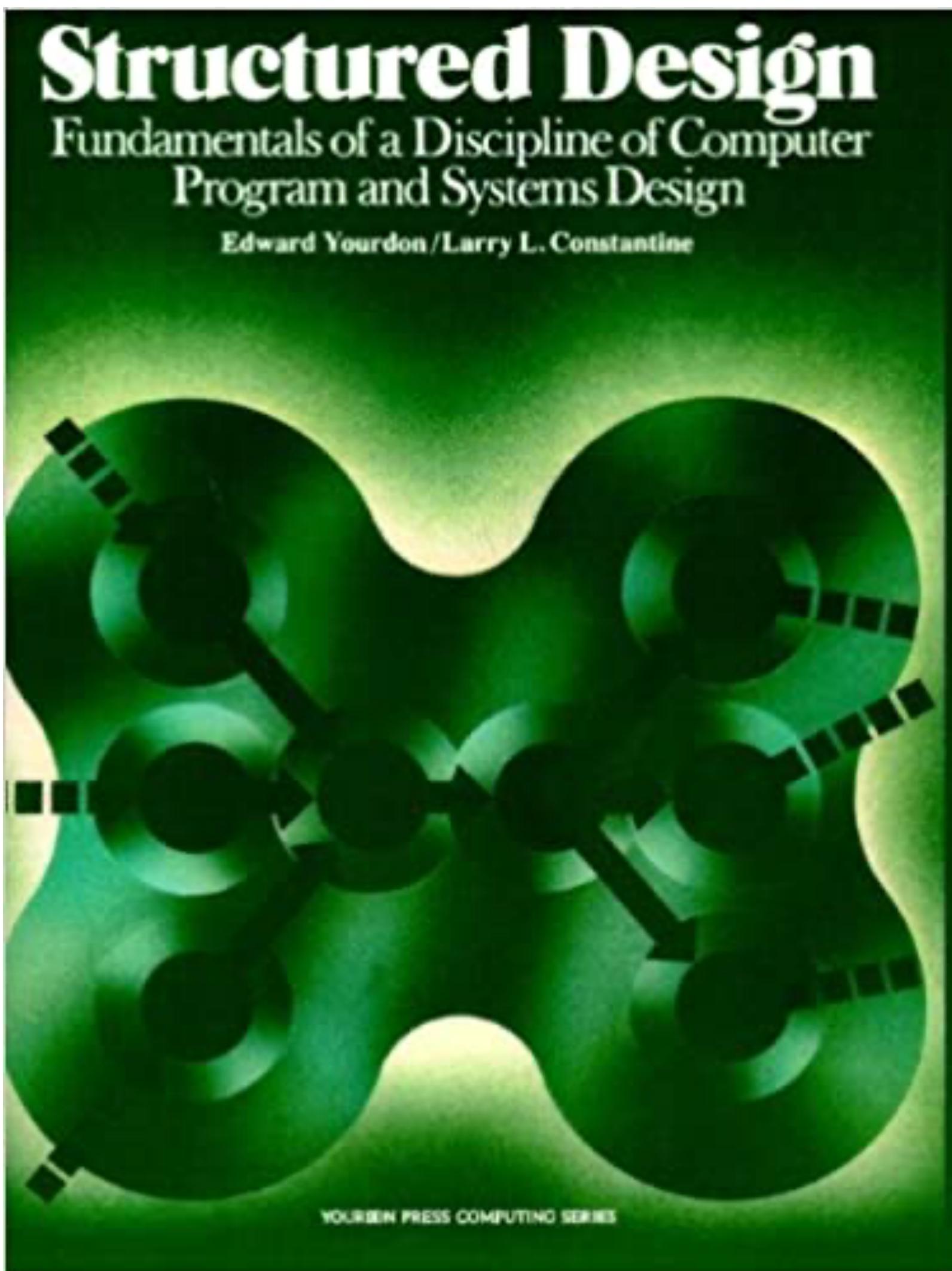
#### Step 2: Initialize a `proto.lock` file:

\$ protolock init (click to lock your .proto definition)

<https://protolock.dev/>

**And of course testing...**

# STRUCTURED PROGRAMMING



# COUPLING AND COHESION

## COUPLING AND COHESION

# Coupling

# COUPLING AND COHESION

**Coupling**

**Cohesion**

# COUPLING AND COHESION

Low  
Coupling      Cohesion

# COUPLING AND COHESION

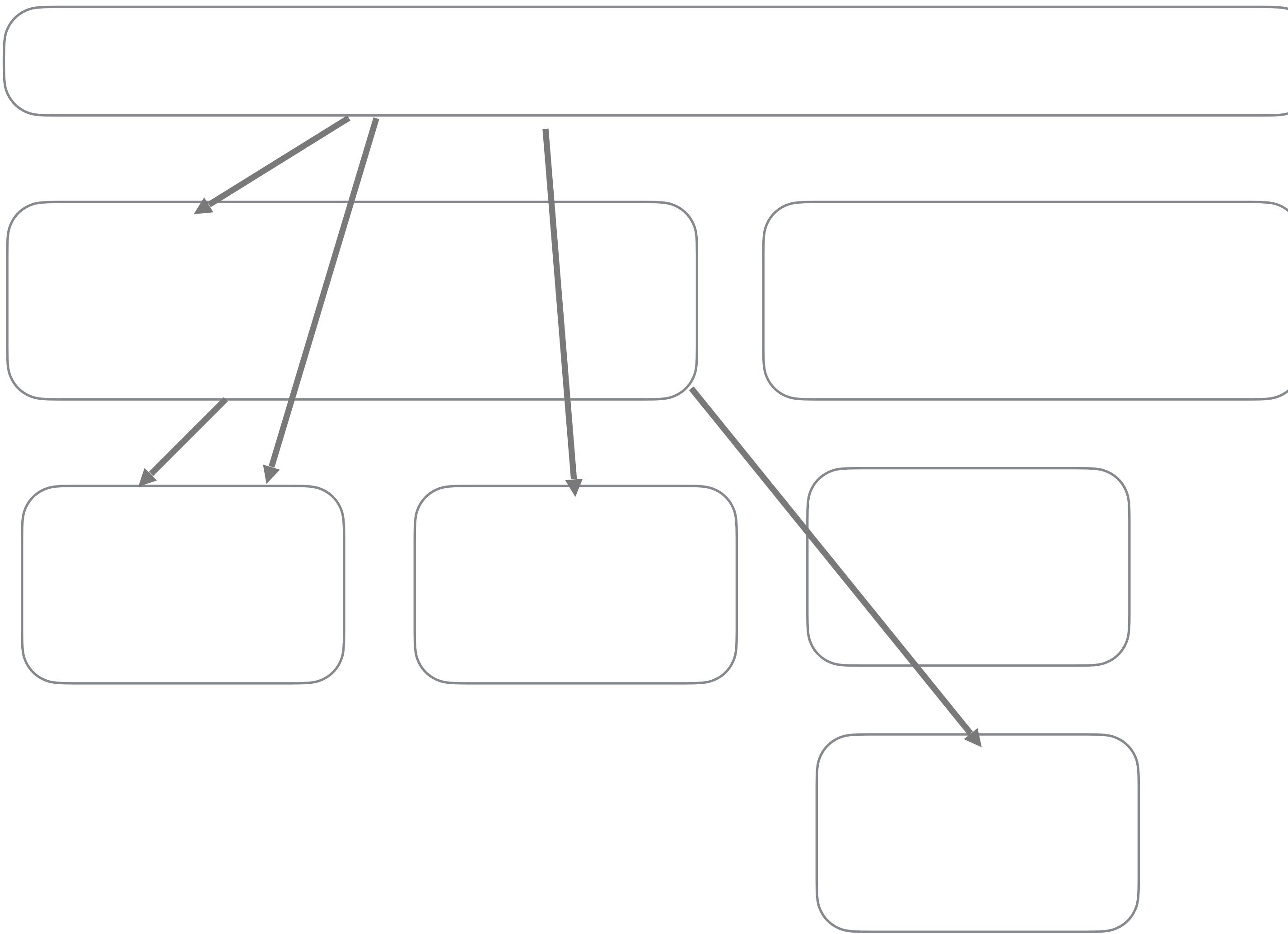
Low  
Coupling

Strong  
Cohesion

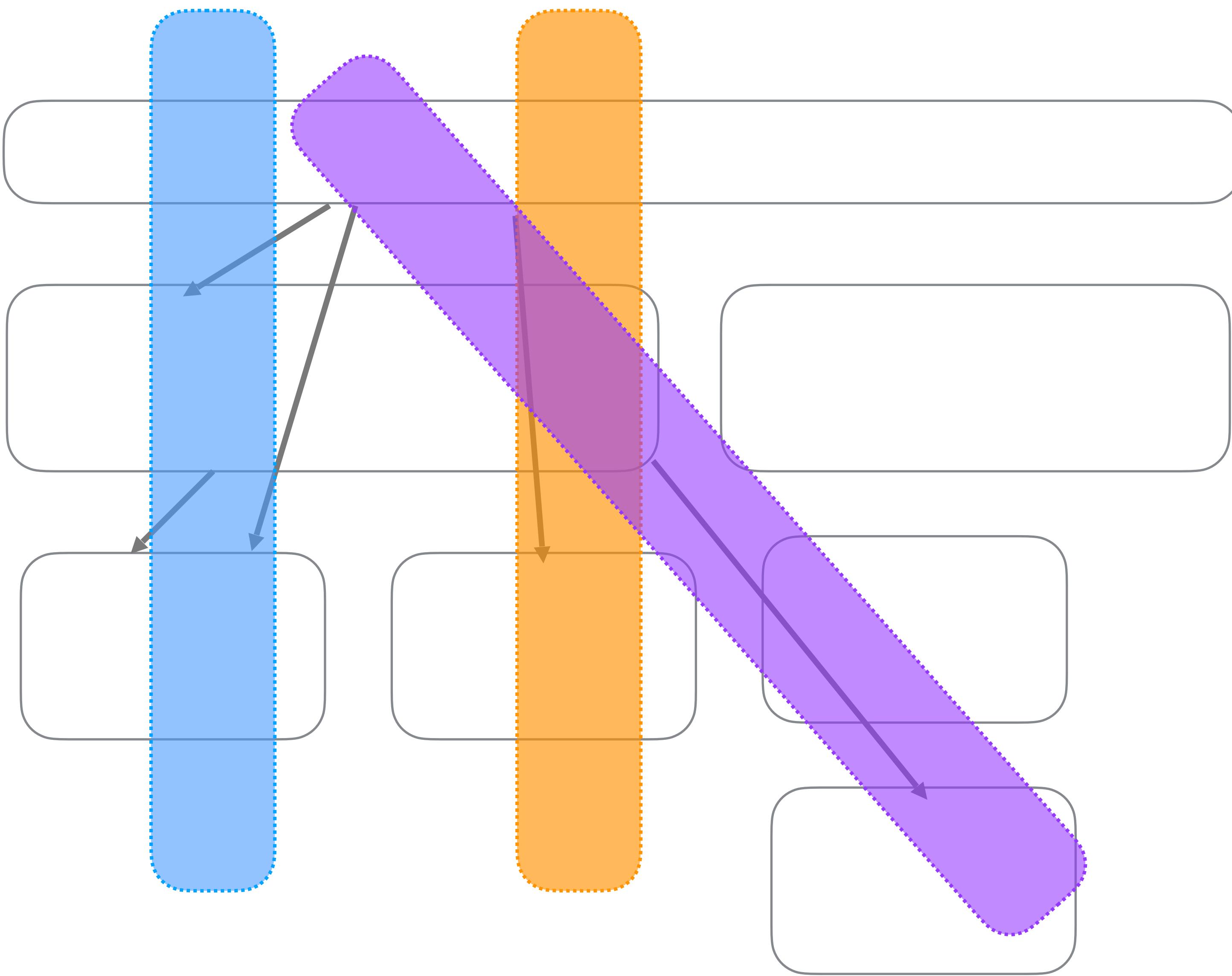
## COHESION

**The code that changes together, stays together**

## WEAK COHESION



## WEAK COHESION



## COUPLING

The degree to which changing one module requires a change in another

## CONSTANTINE'S LAW

***“A structure is stable if cohesion is strong and coupling is low.”***

***- Albert Endres and Dieter Rombach***

# TYPES OF COUPLING

## TYPES OF COUPLING

**Domain**

## TYPES OF COUPLING

**Domain**

**Tramp**

## TYPES OF COUPLING

**Domain**

**Tramp**

**Common**

## TYPES OF COUPLING

**Domain**

**Tramp**

**Common**

**Content**

## TYPES OF COUPLING

**Domain**

**Tramp**

**Common**

**Content**



## TYPES OF COUPLING

**Domain**

**Tramp**

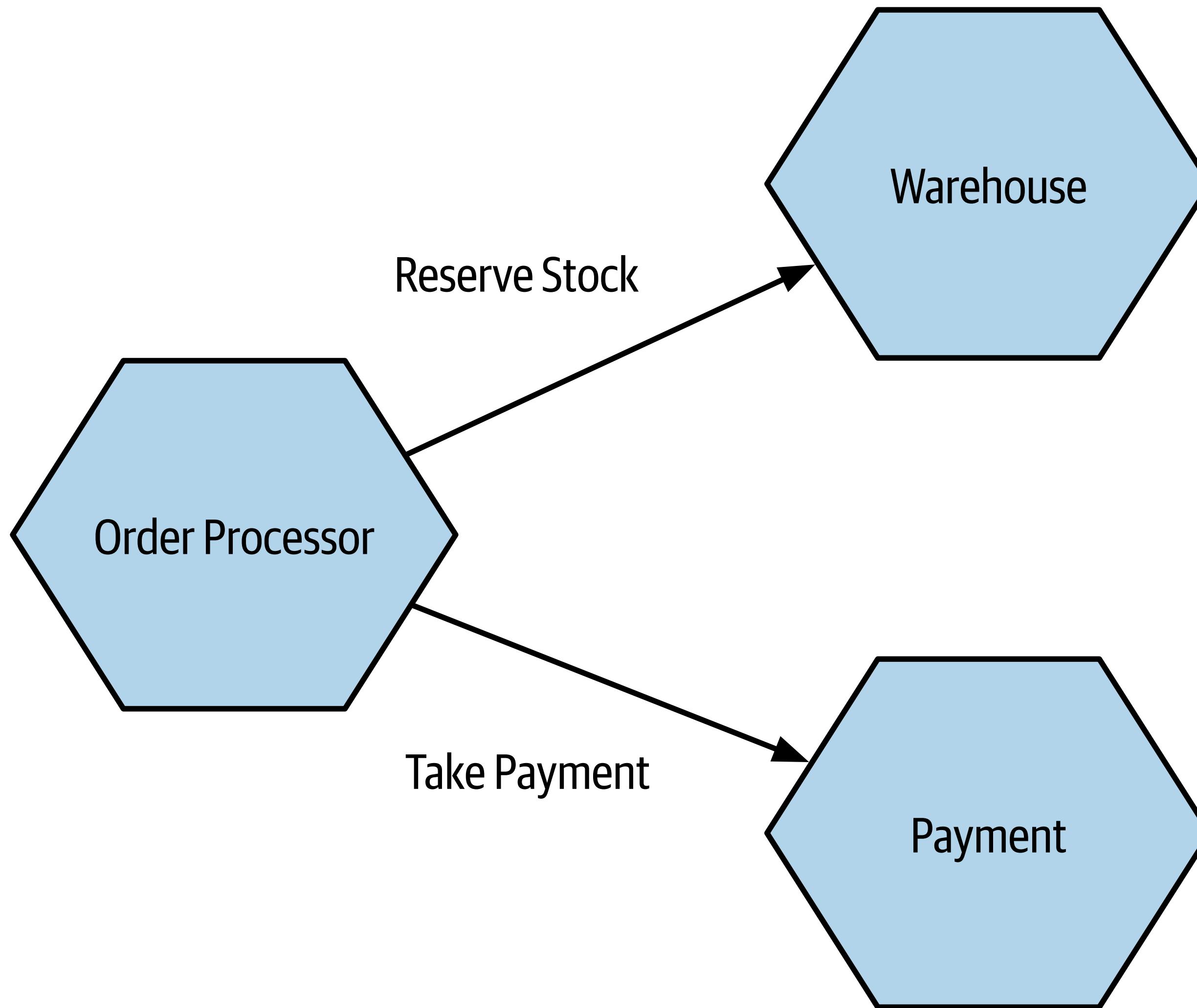
**Common**

**Content**



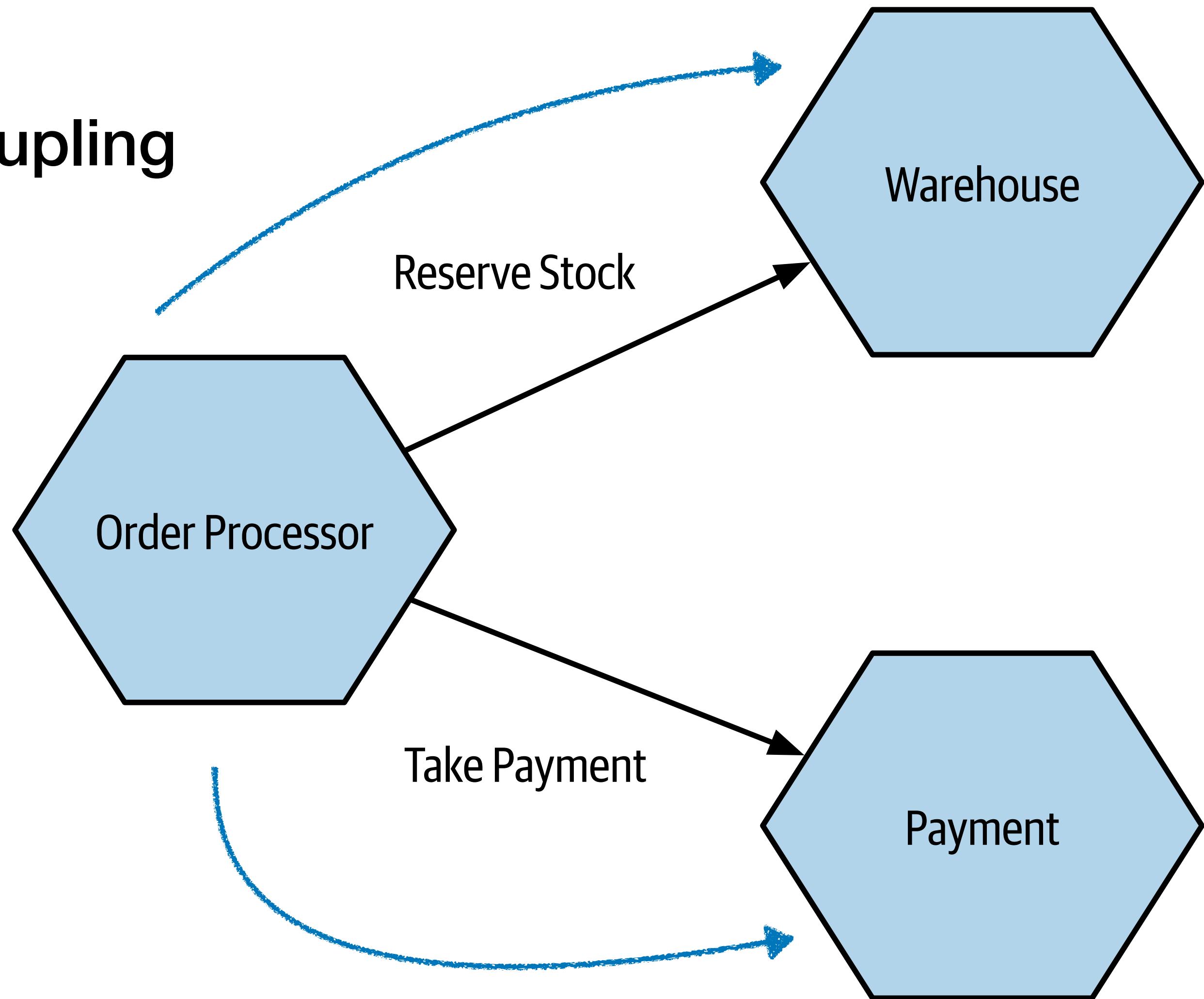
**Increasing coupling**

## DOMAIN COUPLING



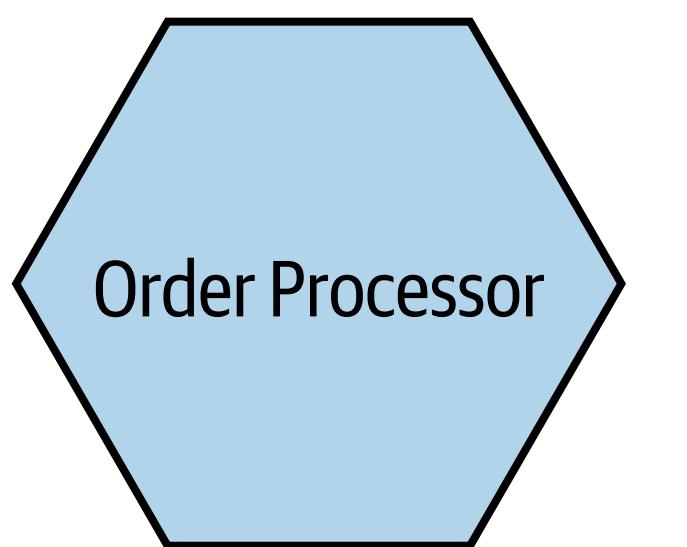
## DOMAIN COUPLING

### Domain Coupling

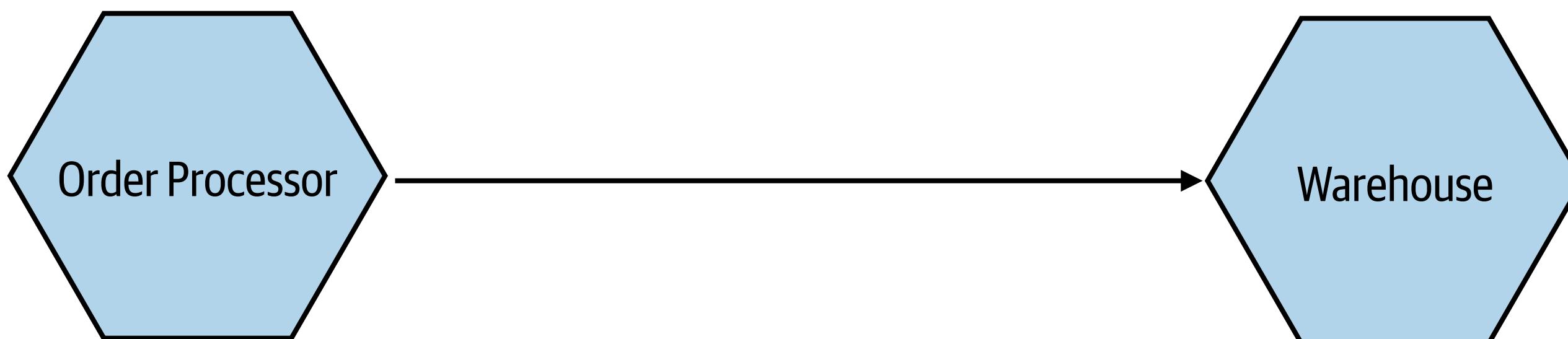


# TRAMP COUPLING

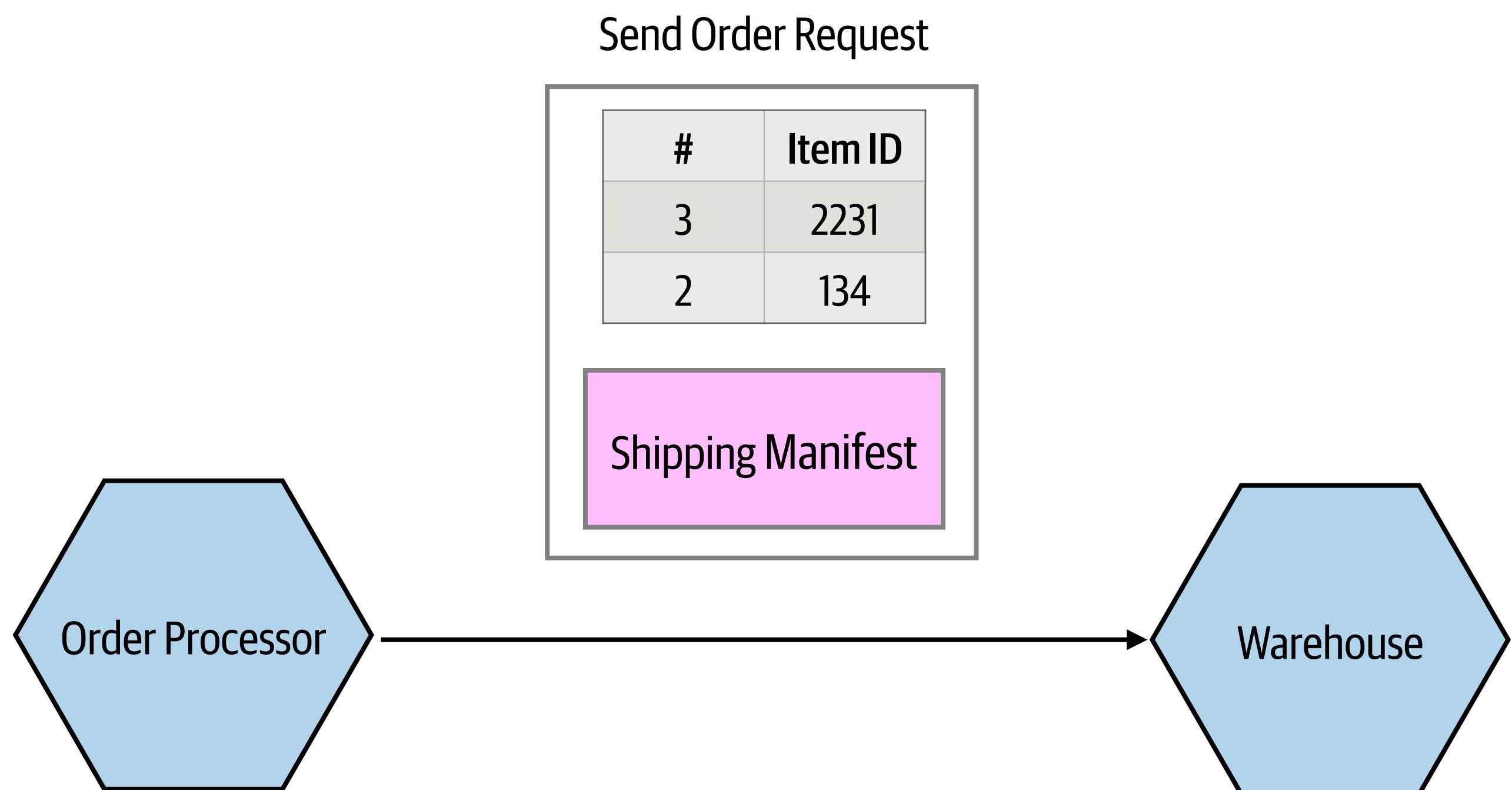
# TRAMP COUPLING



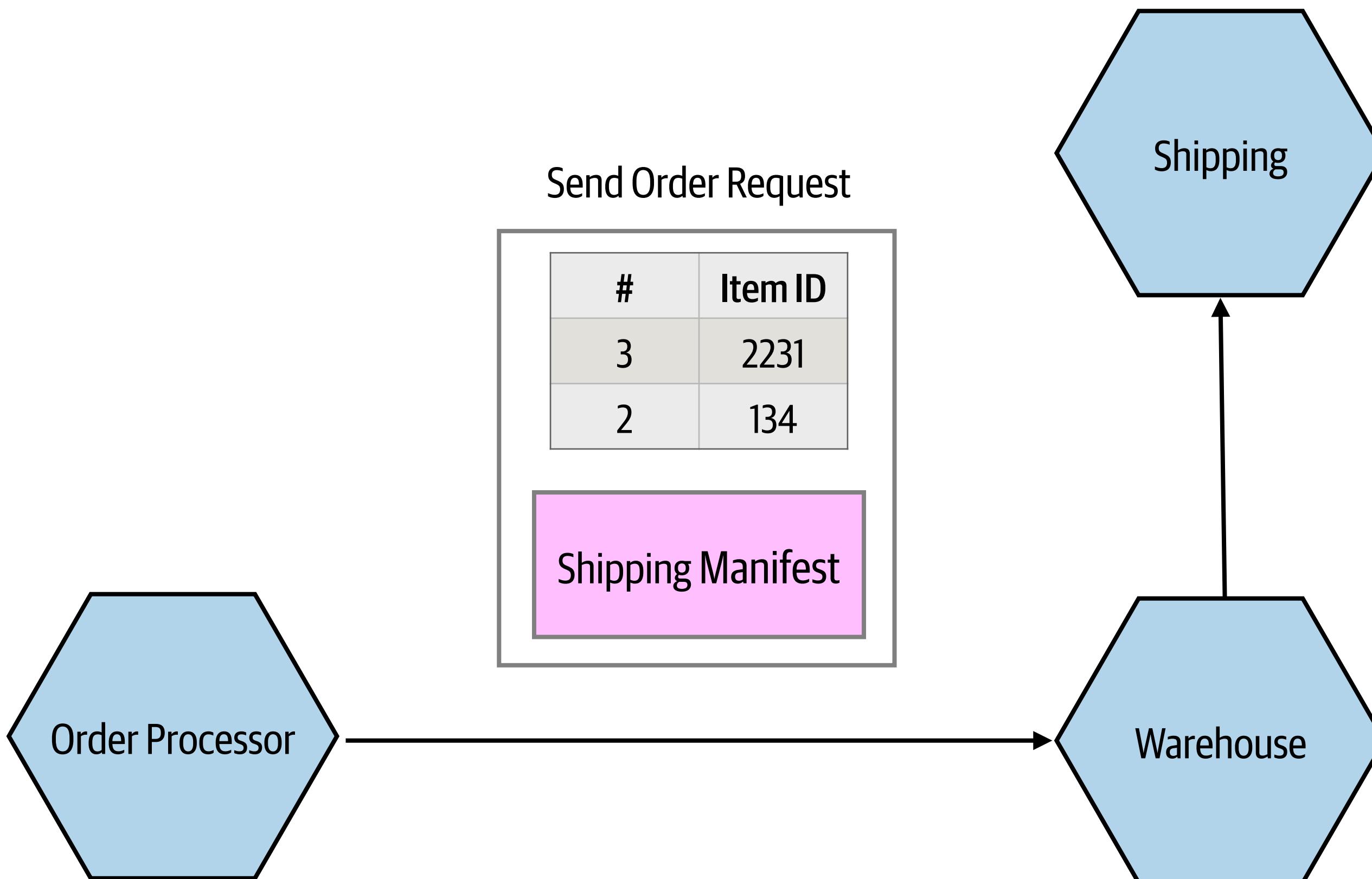
# TRAMP COUPLING



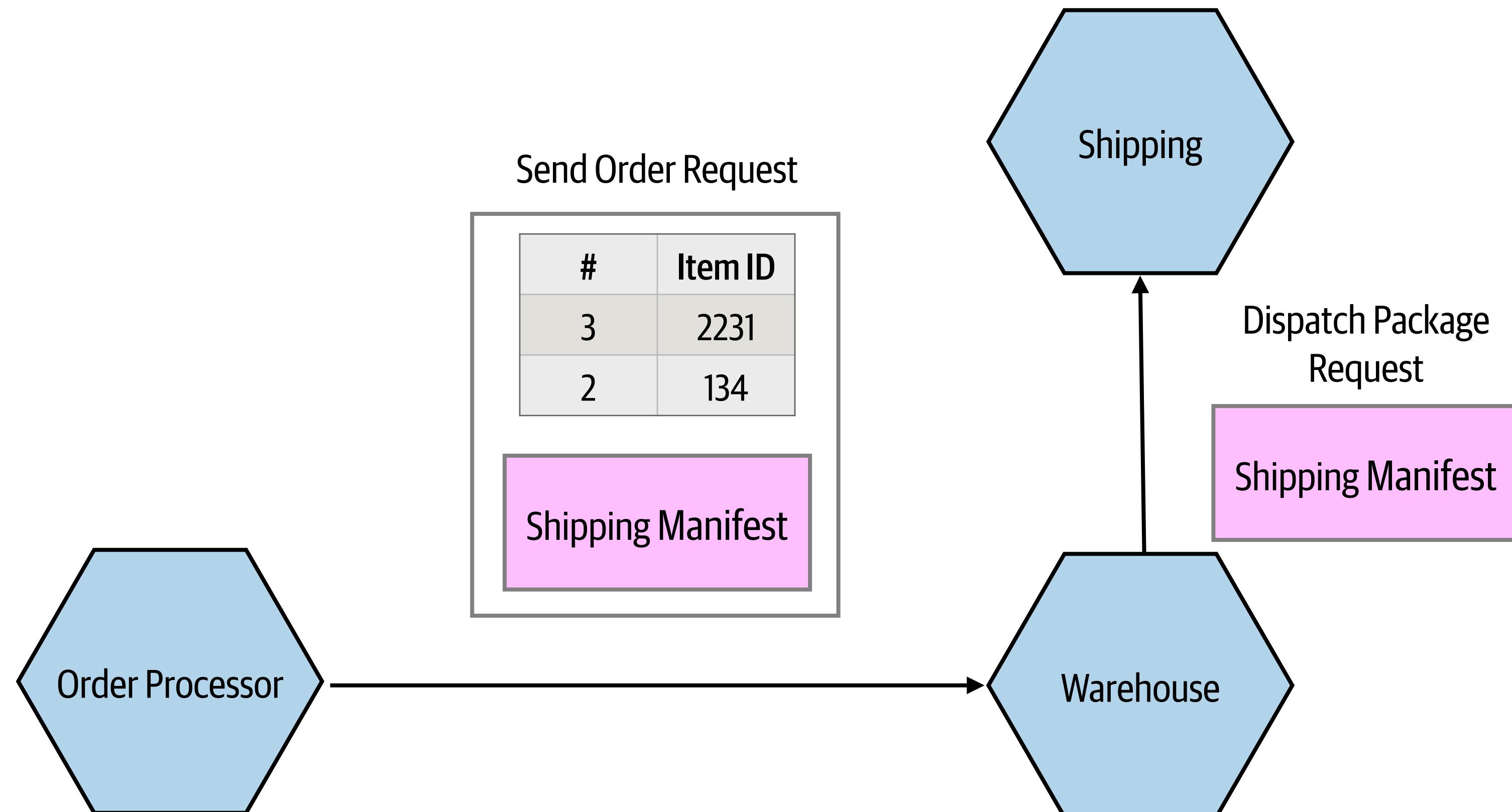
# TRAMP COUPLING



# TRAMP COUPLING

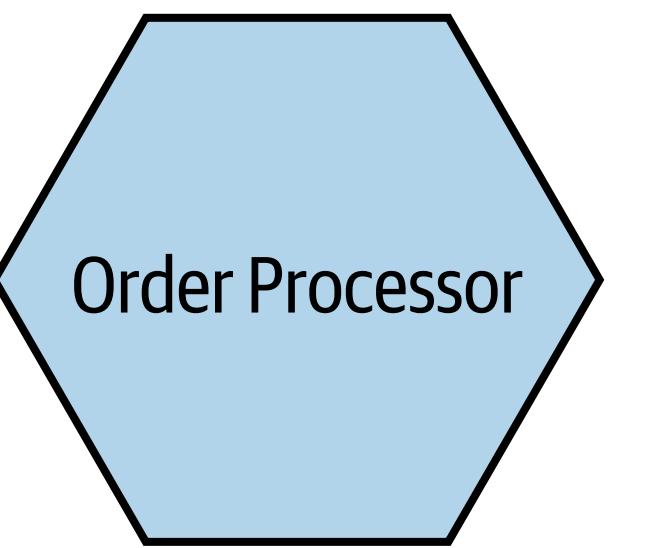


# TRAMP COUPLING

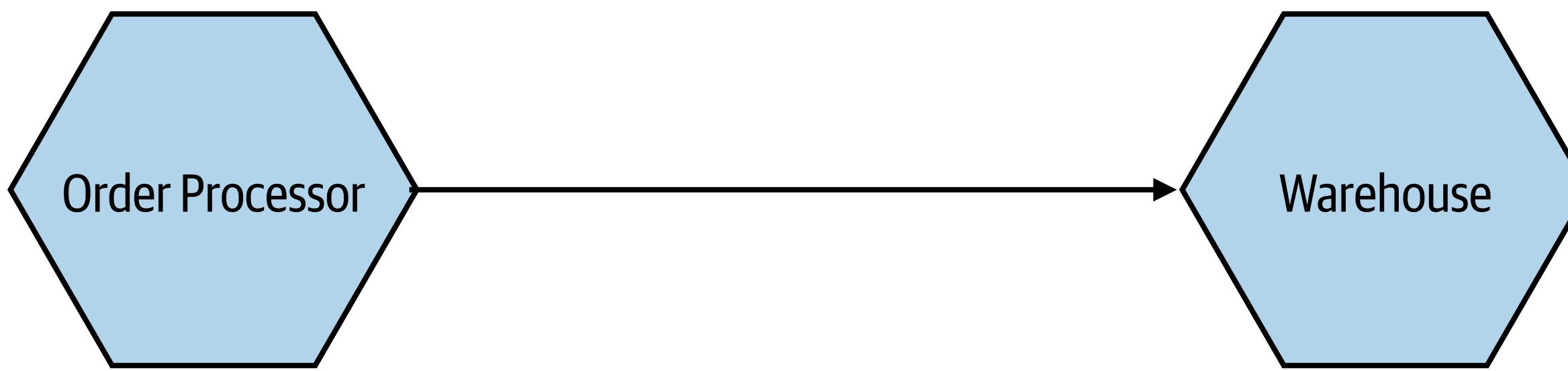


# AVOIDING TRAMP COUPLING

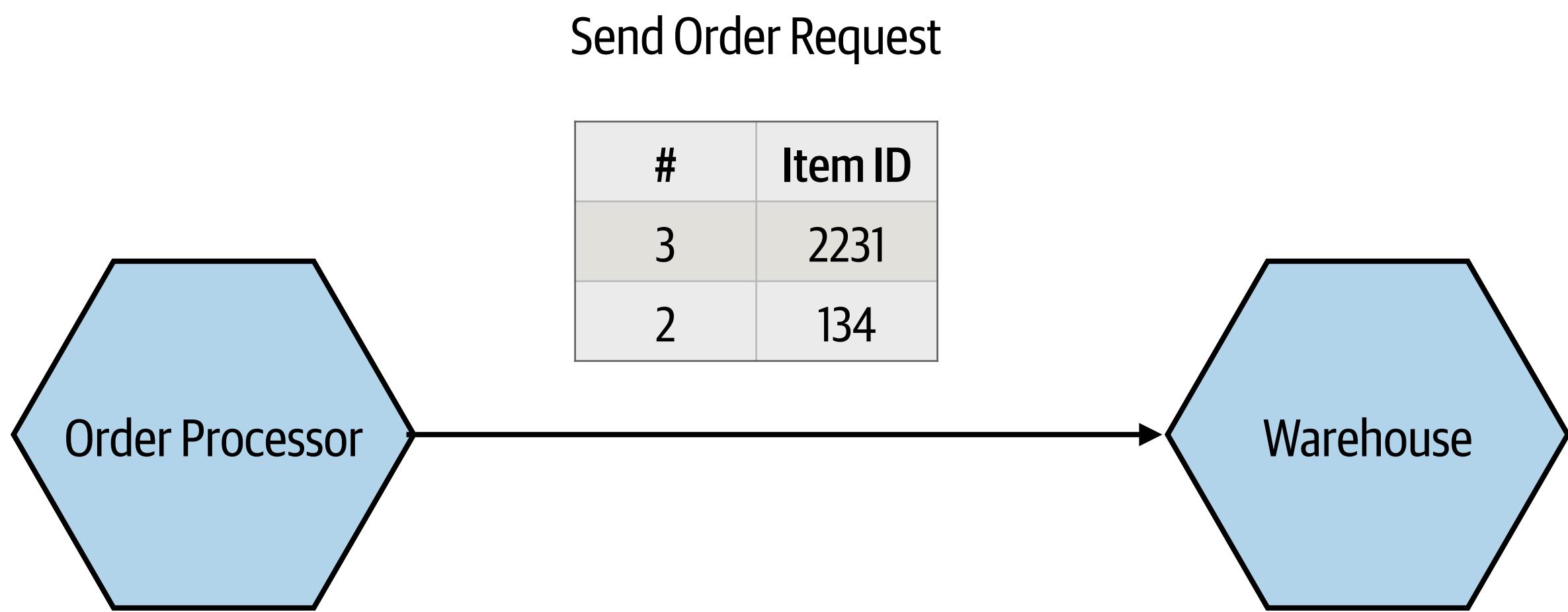
# AVOIDING TRAMP COUPLING



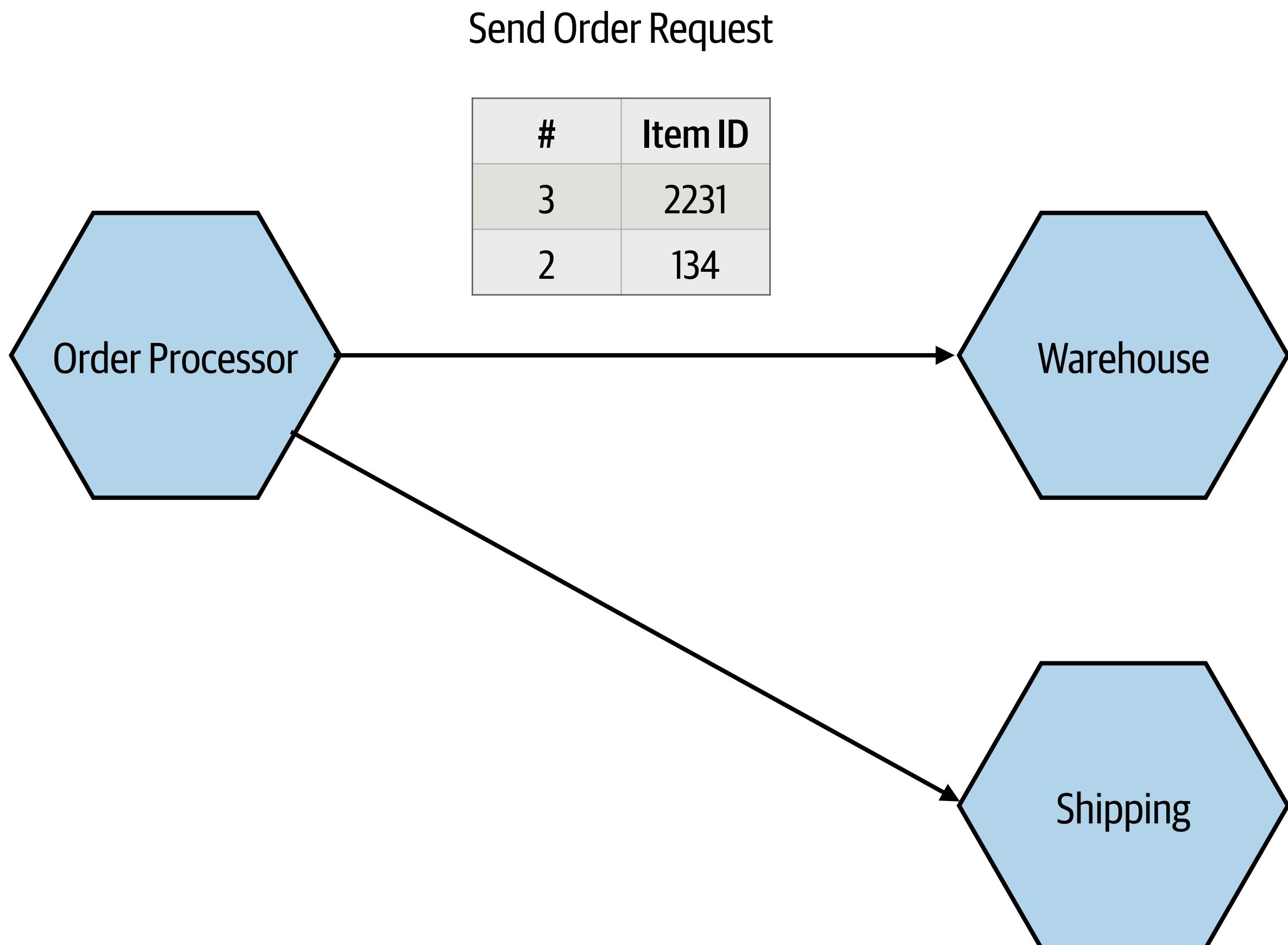
## AVOIDING TRAMP COUPLING



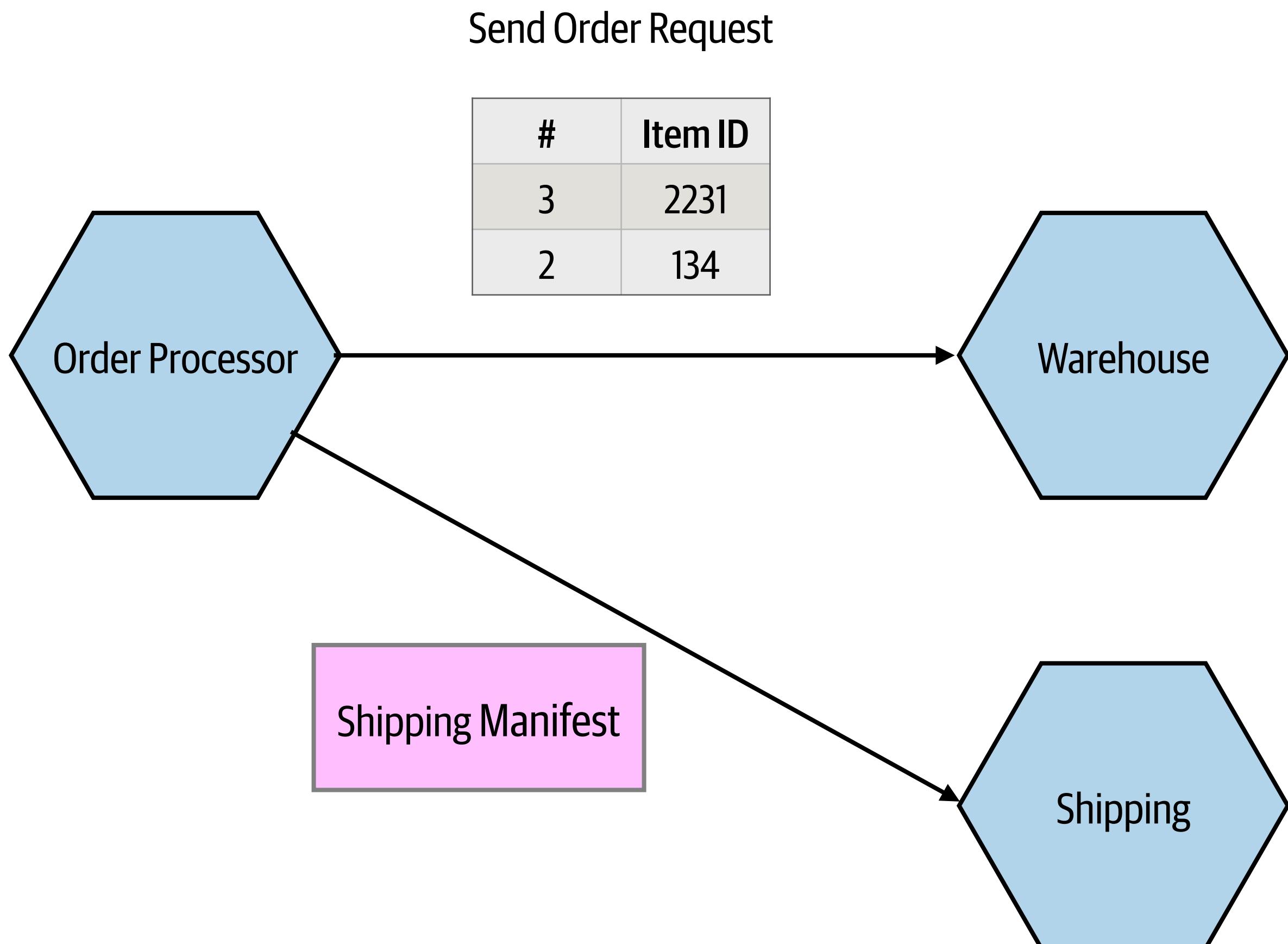
# AVOIDING TRAMP COUPLING



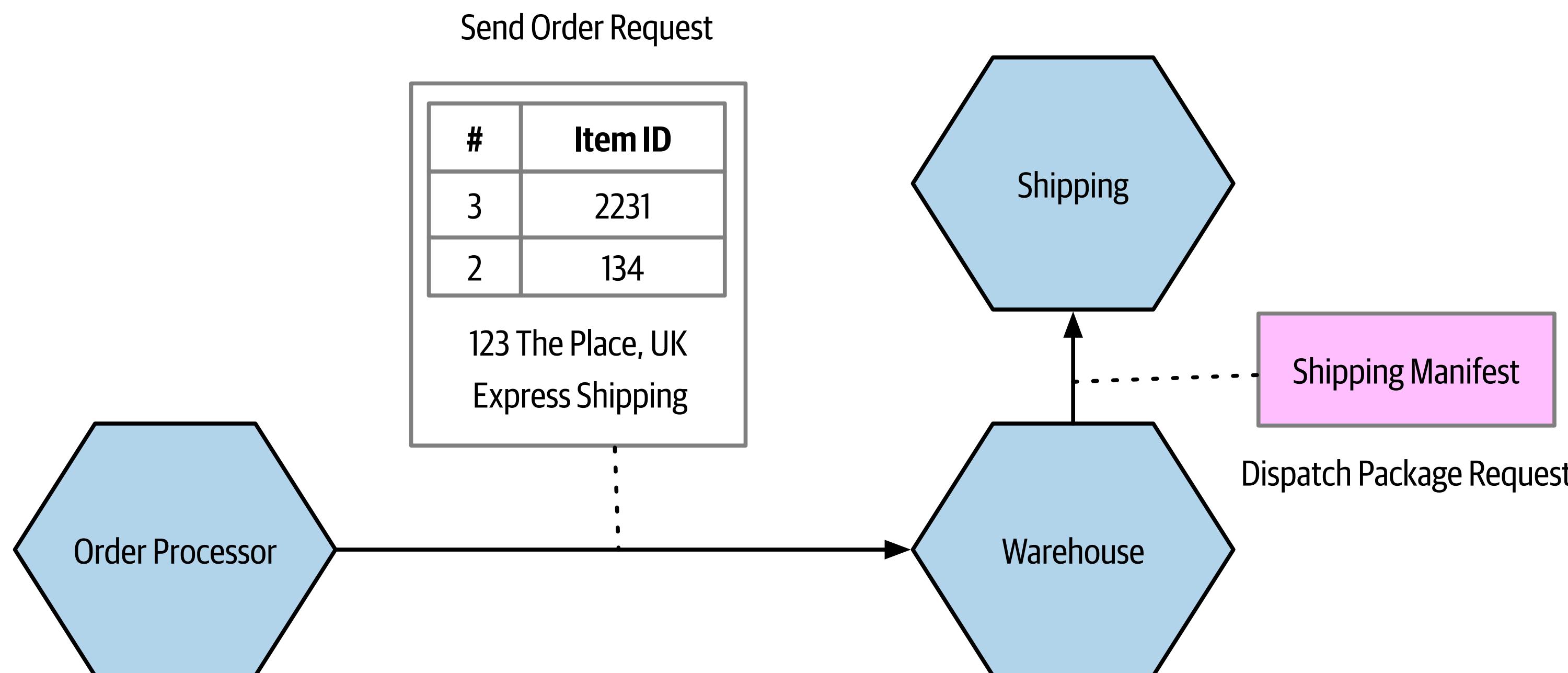
# AVOIDING TRAMP COUPLING



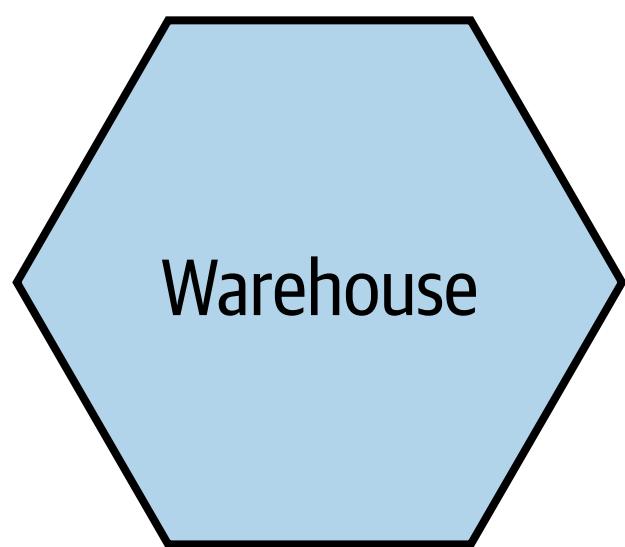
# AVOIDING TRAMP COUPLING



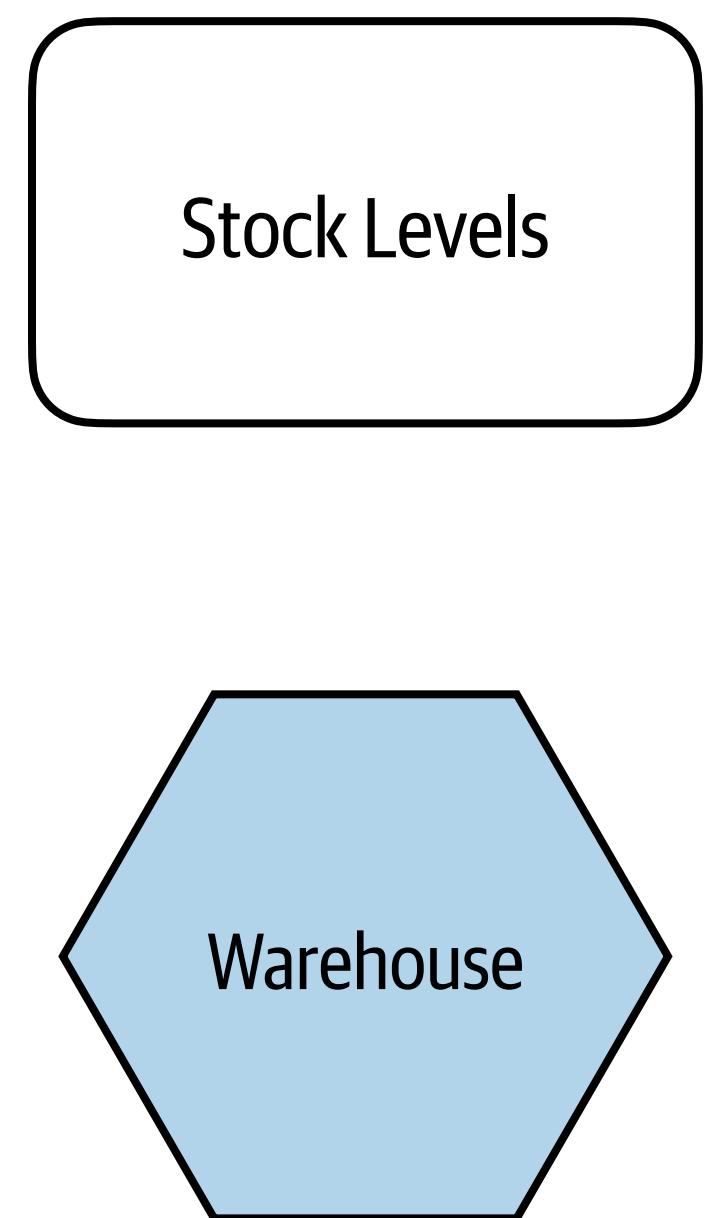
## AVOIDING TRAMP COUPLING (CONT)



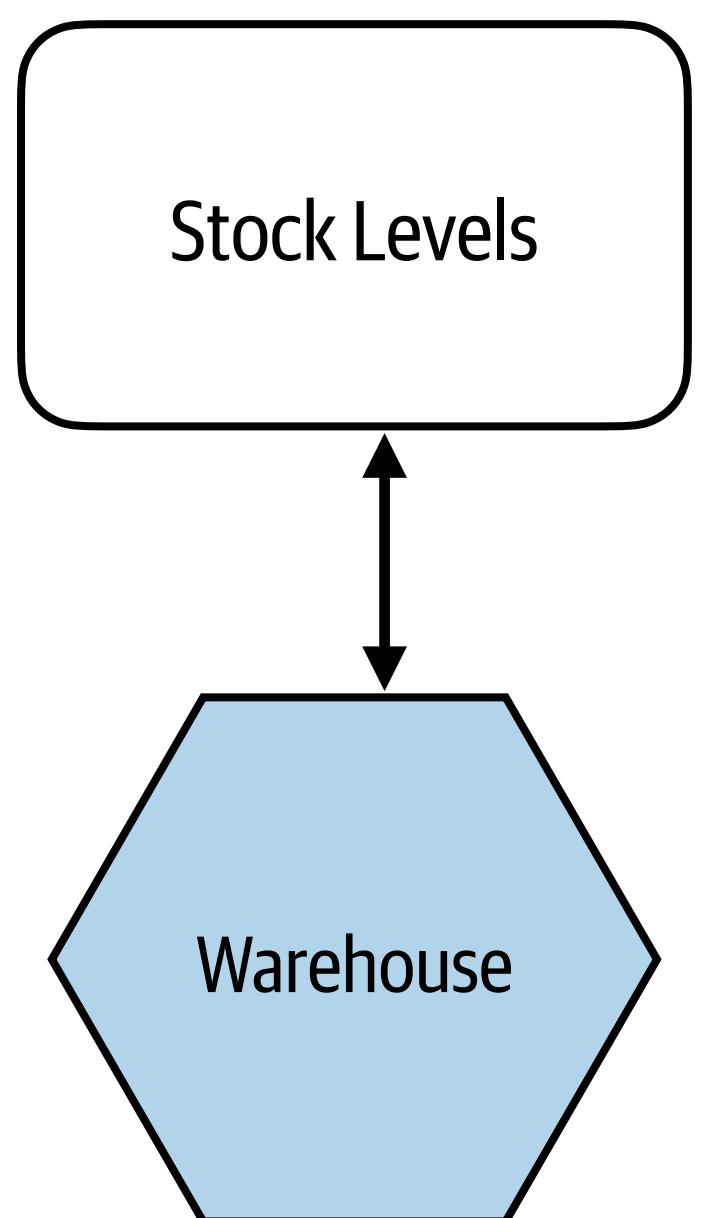
# COMMON COUPLING



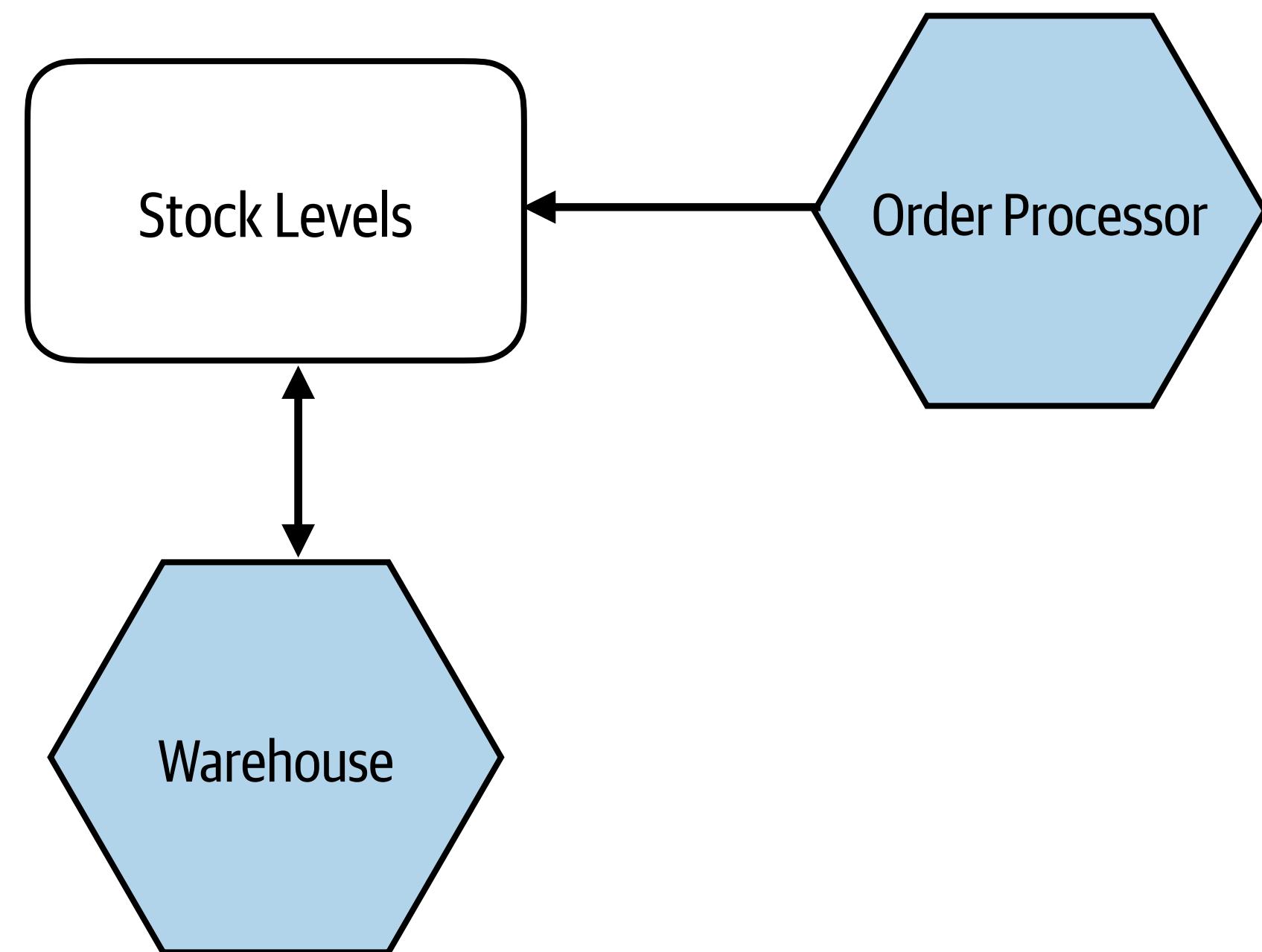
# COMMON COUPLING



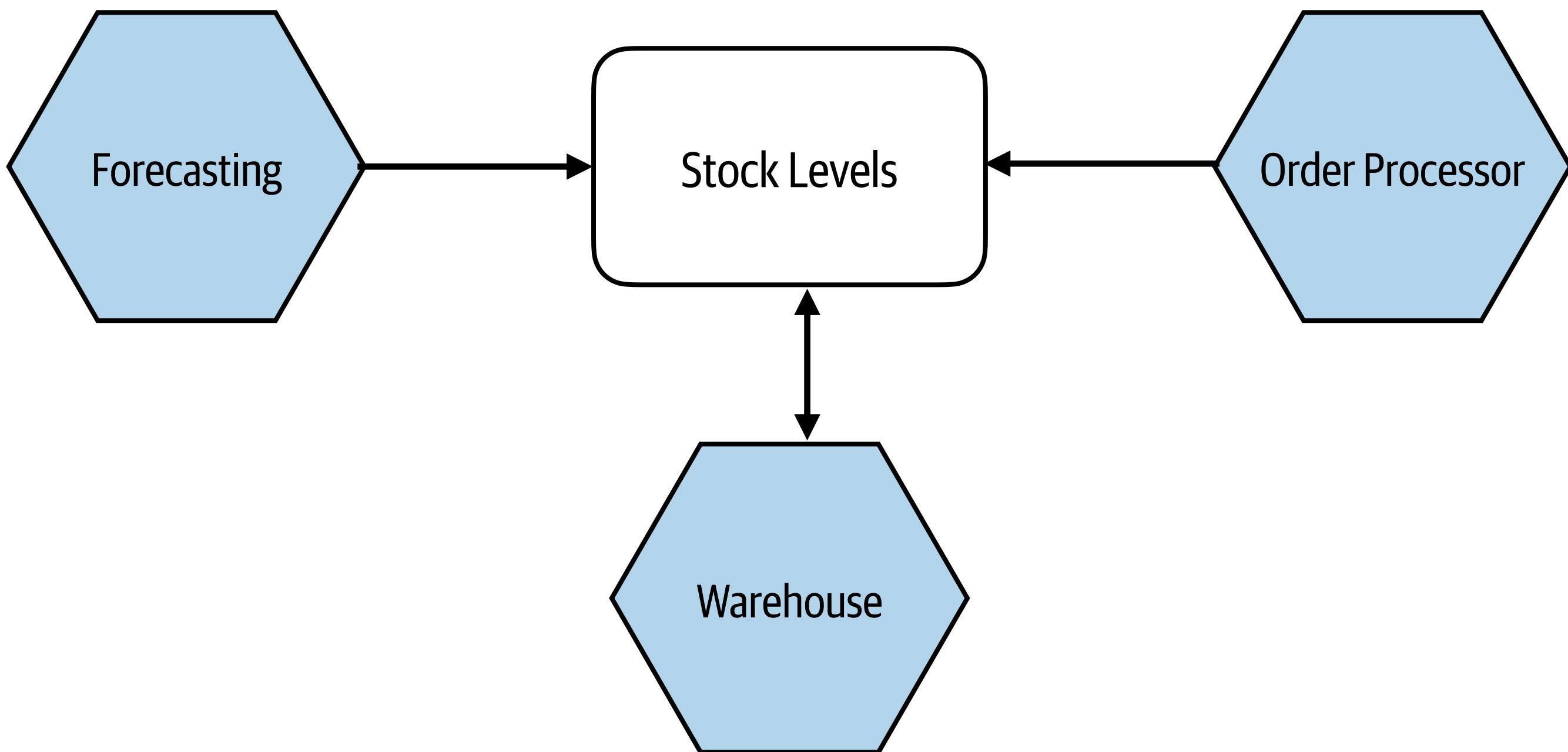
# COMMON COUPLING



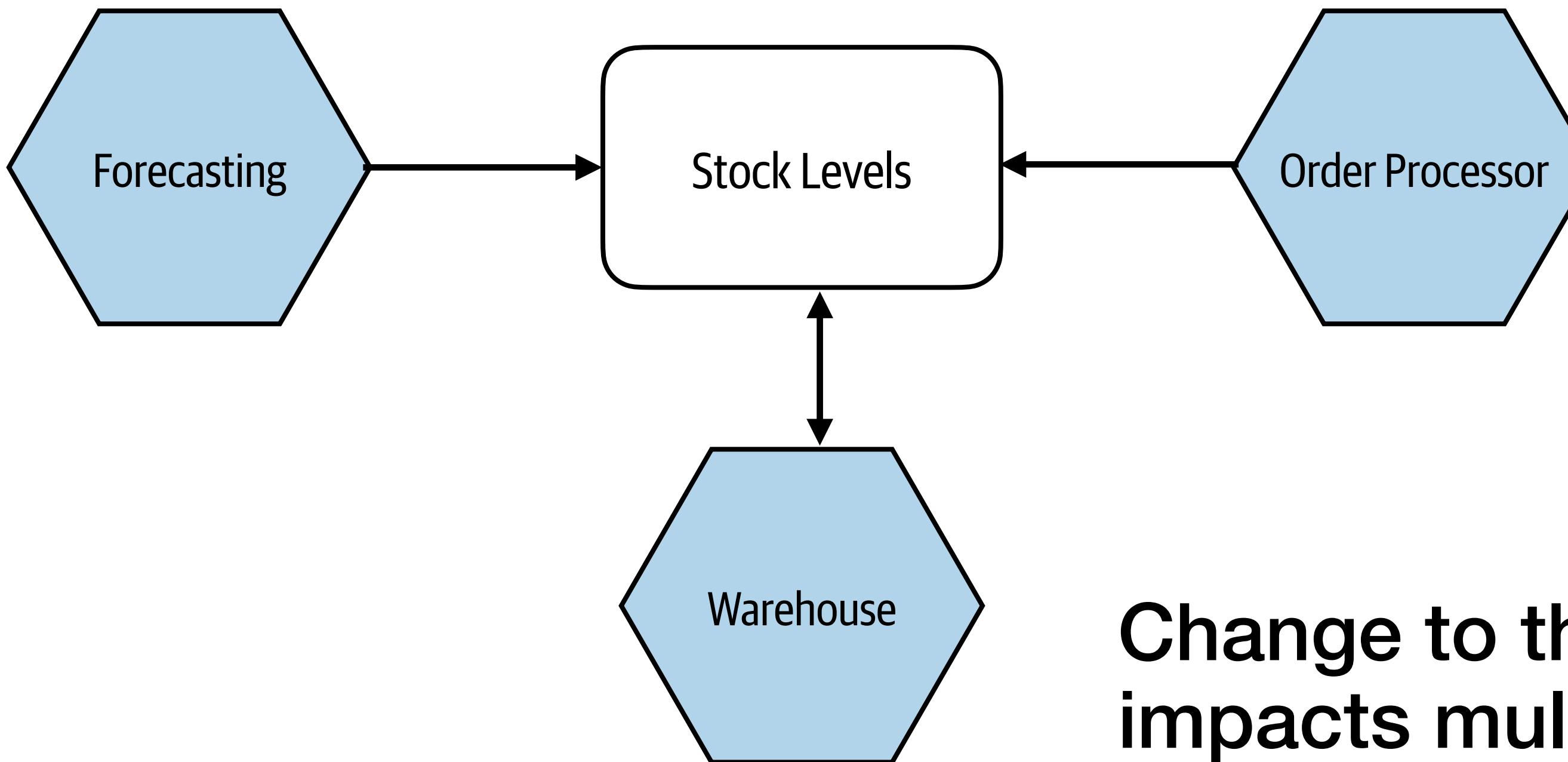
## COMMON COUPLING



## COMMON COUPLING

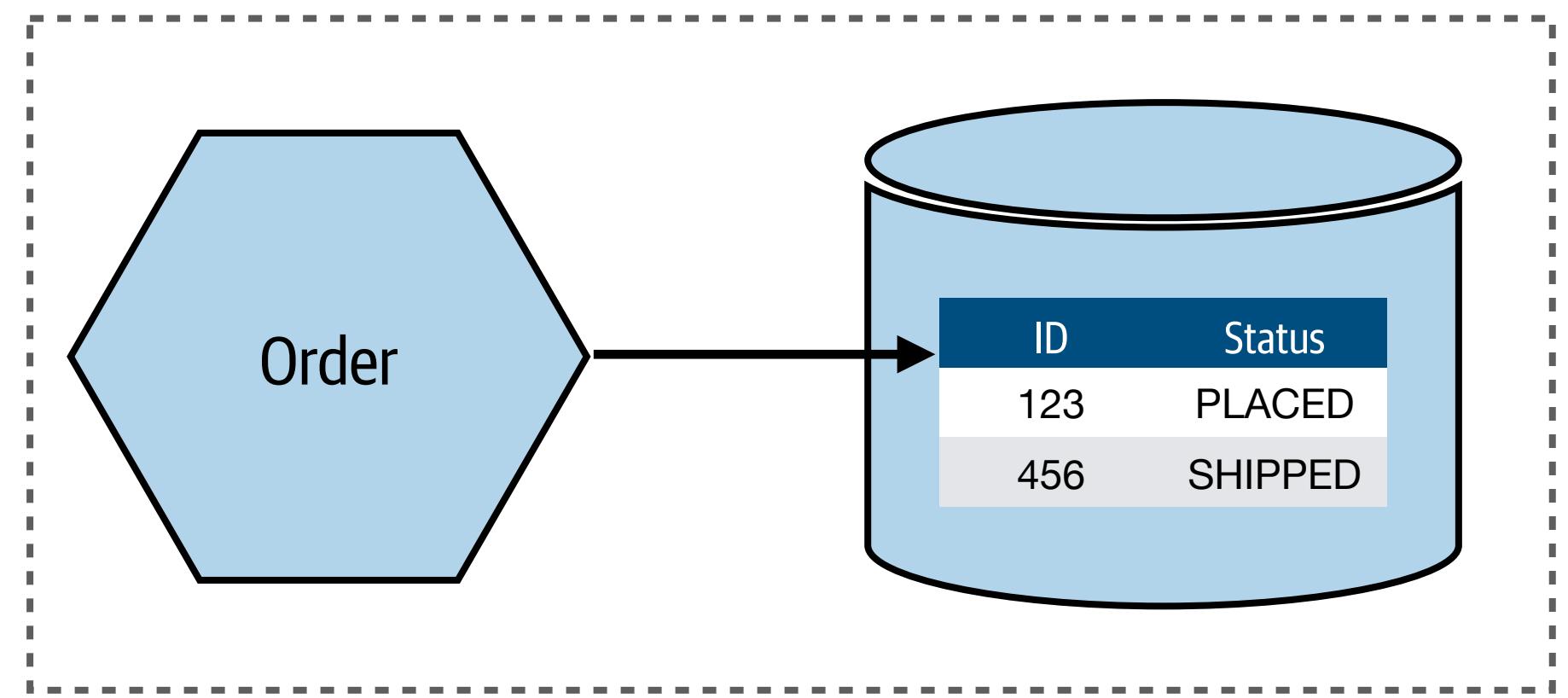


## COMMON COUPLING

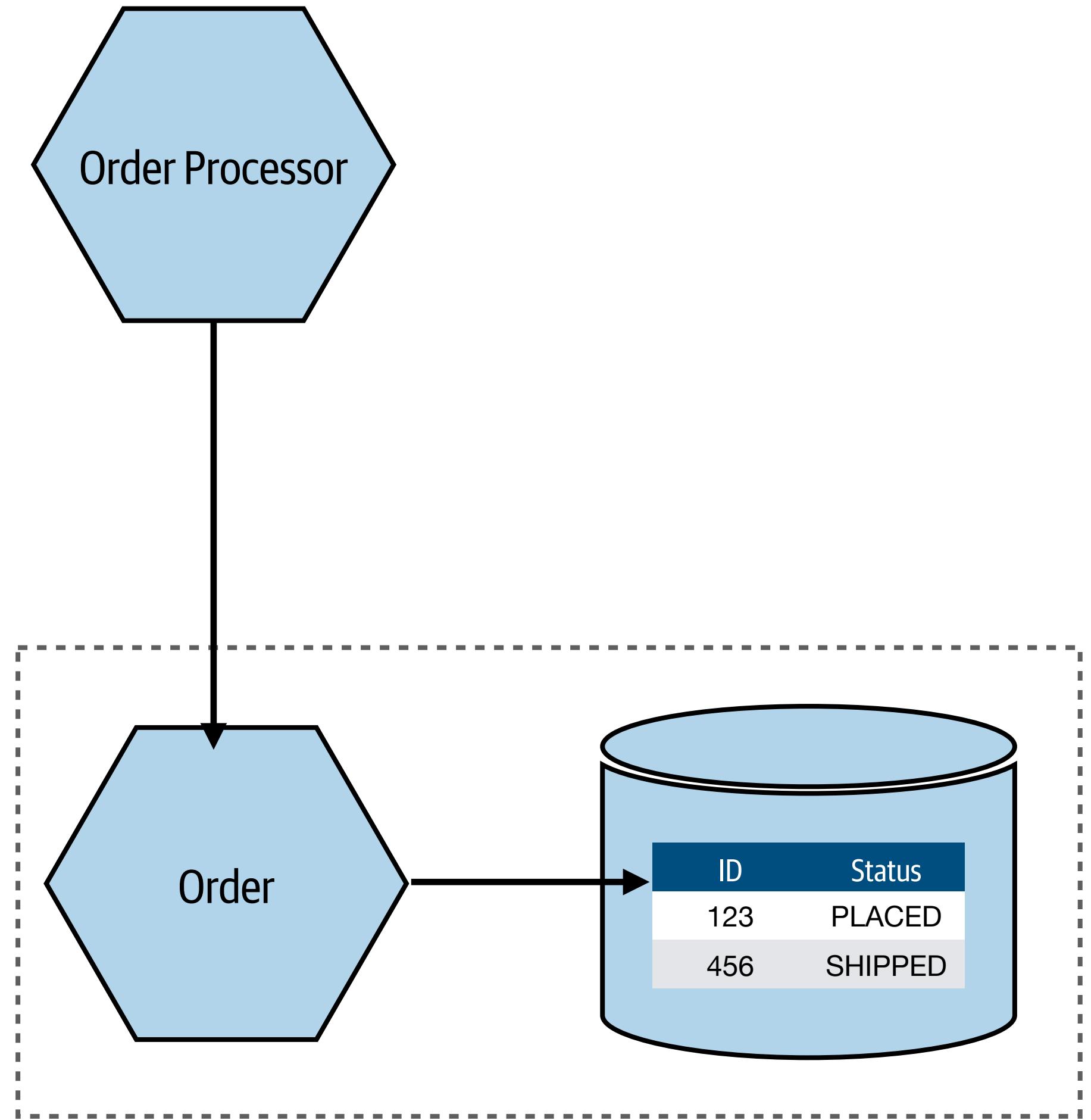


**Change to the common data impacts multiple microservices**

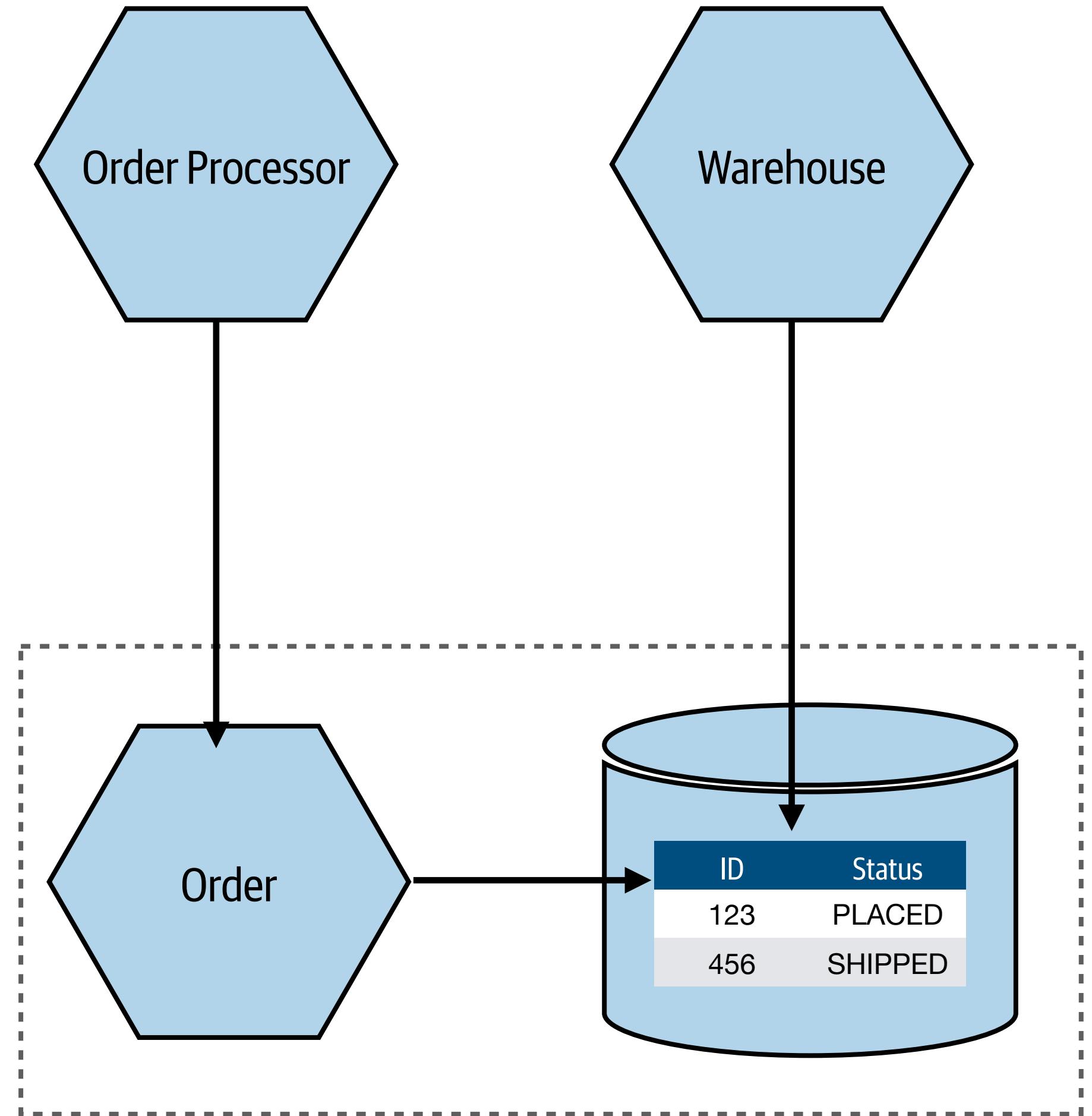
# CONTENT COUPLING



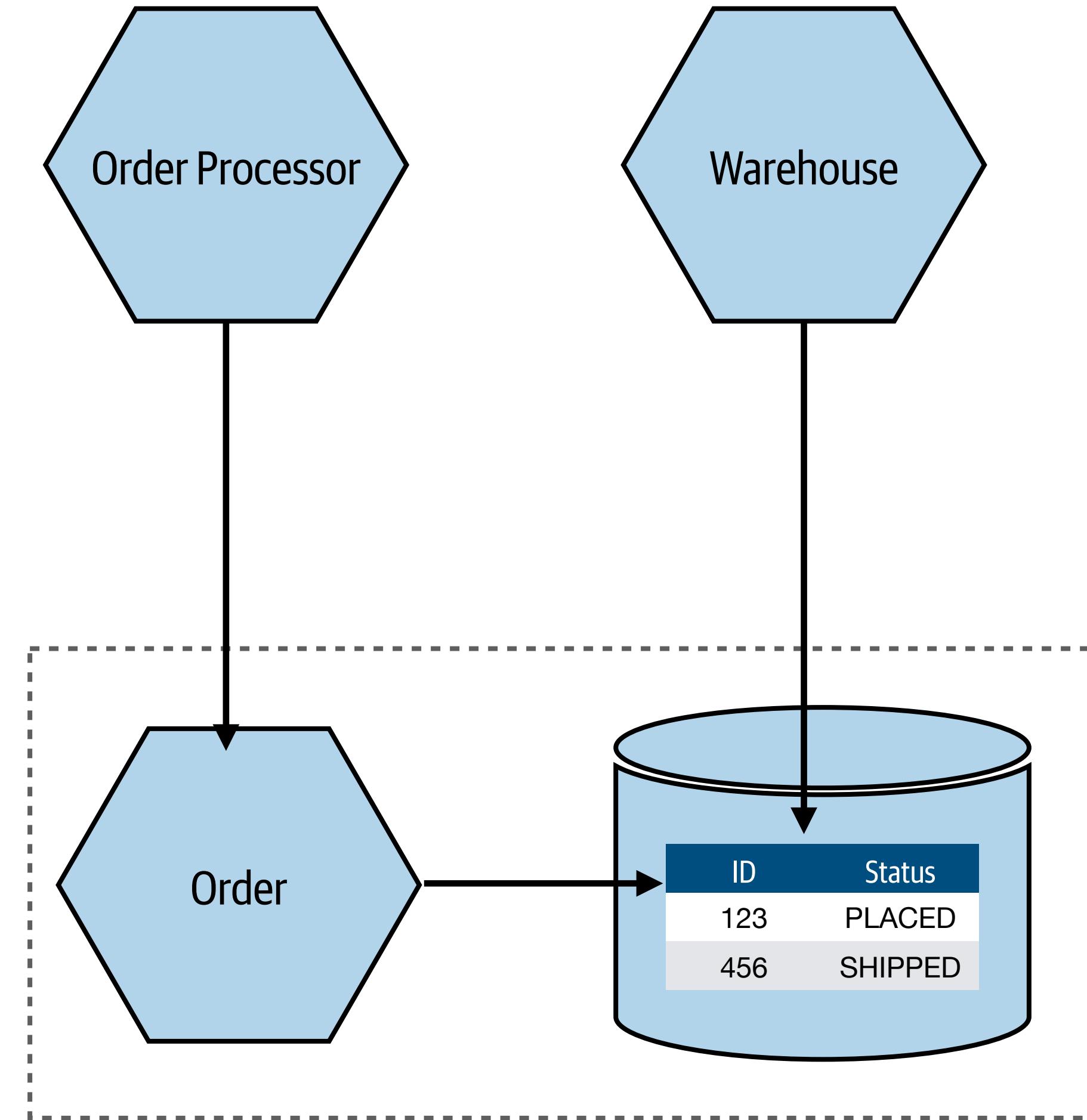
# CONTENT COUPLING



# CONTENT COUPLING

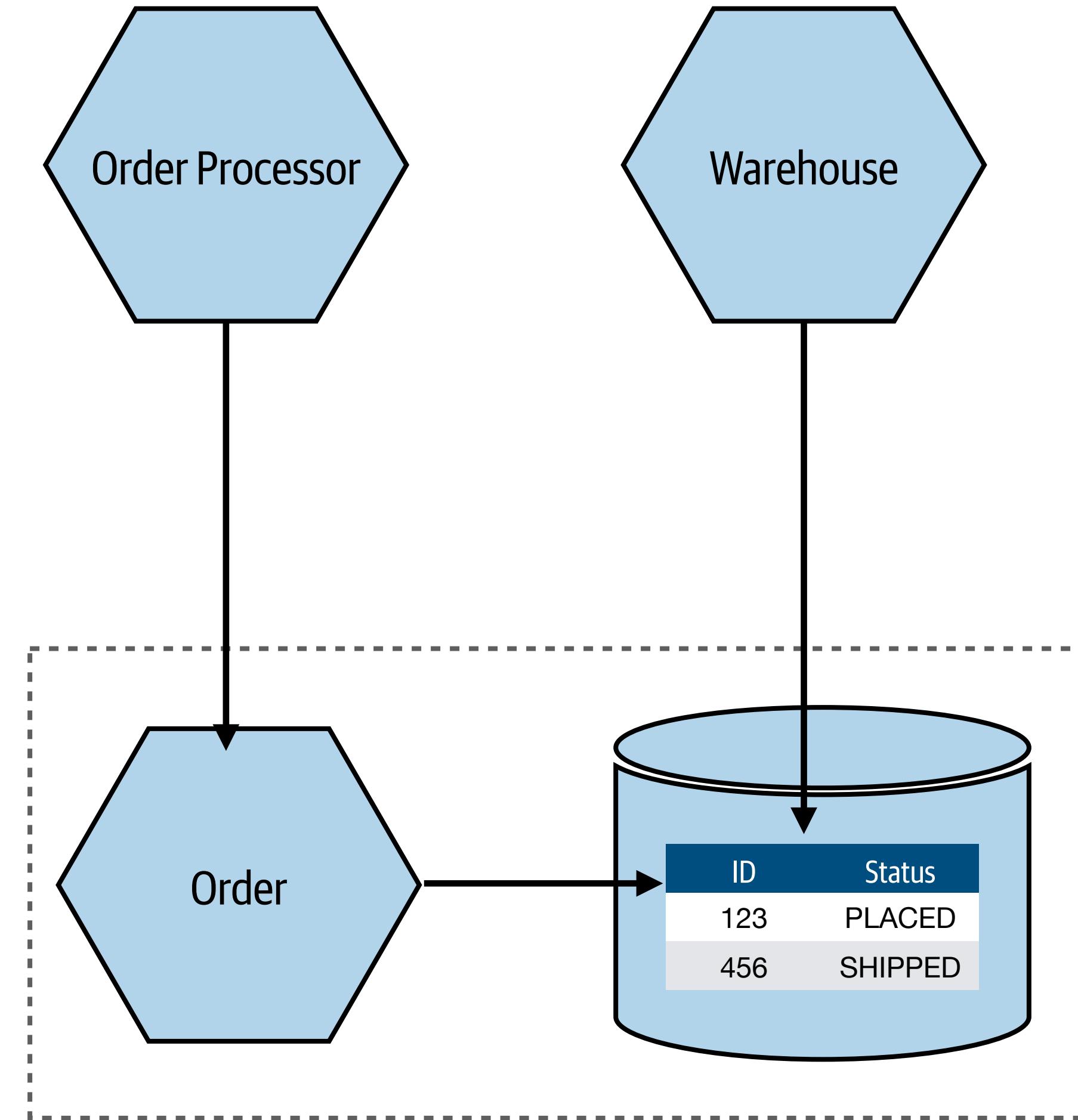


## CONTENT COUPLING



Information hiding bypassed

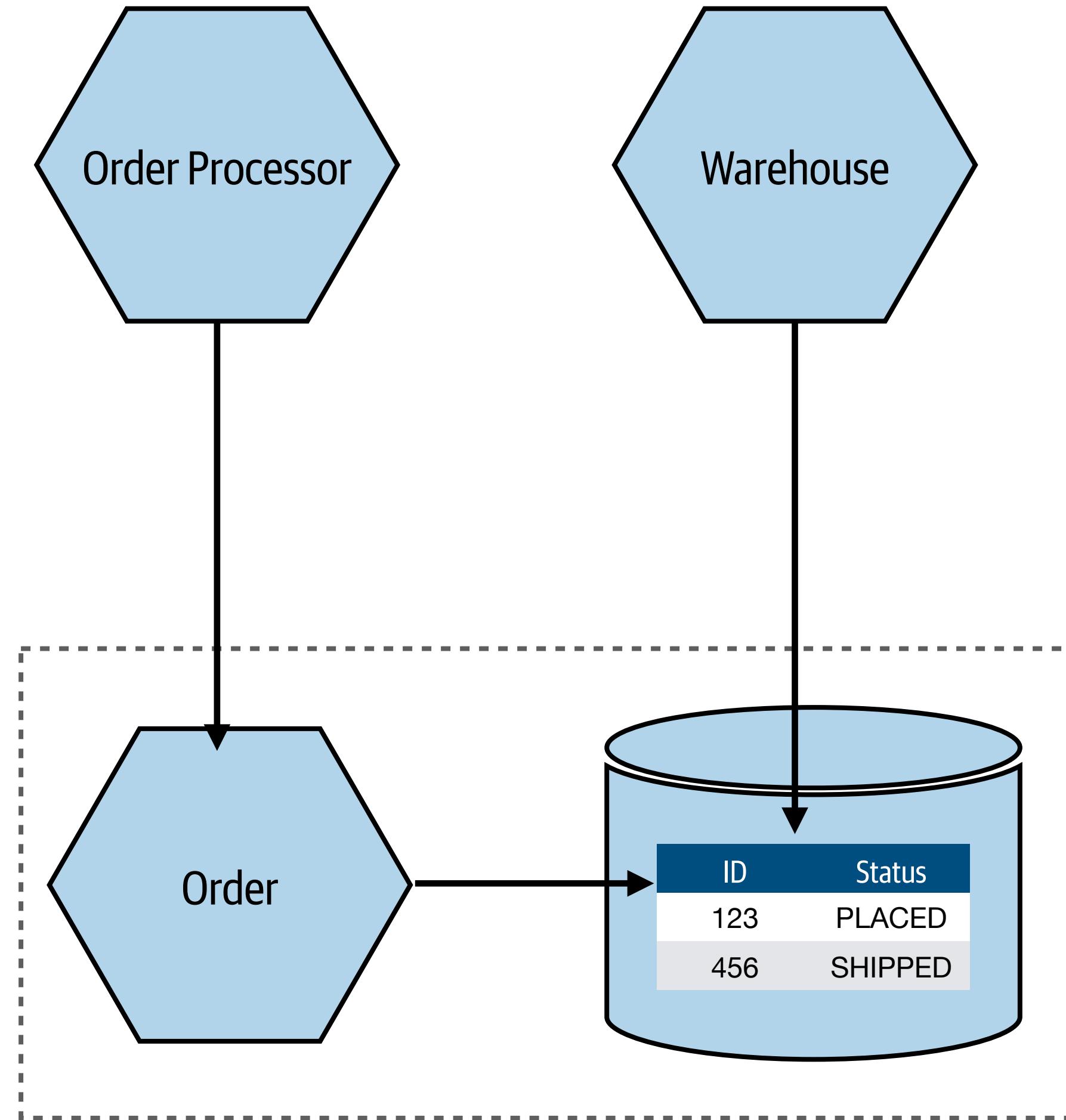
## CONTENT COUPLING



Information hiding bypassed

Unclear what can safely be changed in the Order service

## CONTENT COUPLING



Information hiding bypassed

Unclear what can safely be changed in the Order service

Aka Pathological Coupling

## TYPES OF COUPLING

**Domain**

**Tramp**

**Common**

**Content**



## TYPES OF COUPLING

**Domain**

**Tramp**

**Common**

**Content**



**Increasing coupling**

# IN SUMMARY

## IN SUMMARY

**Information hiding is super important**

## IN SUMMARY

**Information hiding is super important**

**Some coupling is worse than others**

## IN SUMMARY

**Information hiding is super important**

**Some coupling is worse than others**

**Information hiding, low coupling, strong cohesion = Independent Deployability**

# THANKS!

**Sam Newman & Associates**



Home    About    Offerings    Events    **Writing**    Contact

---

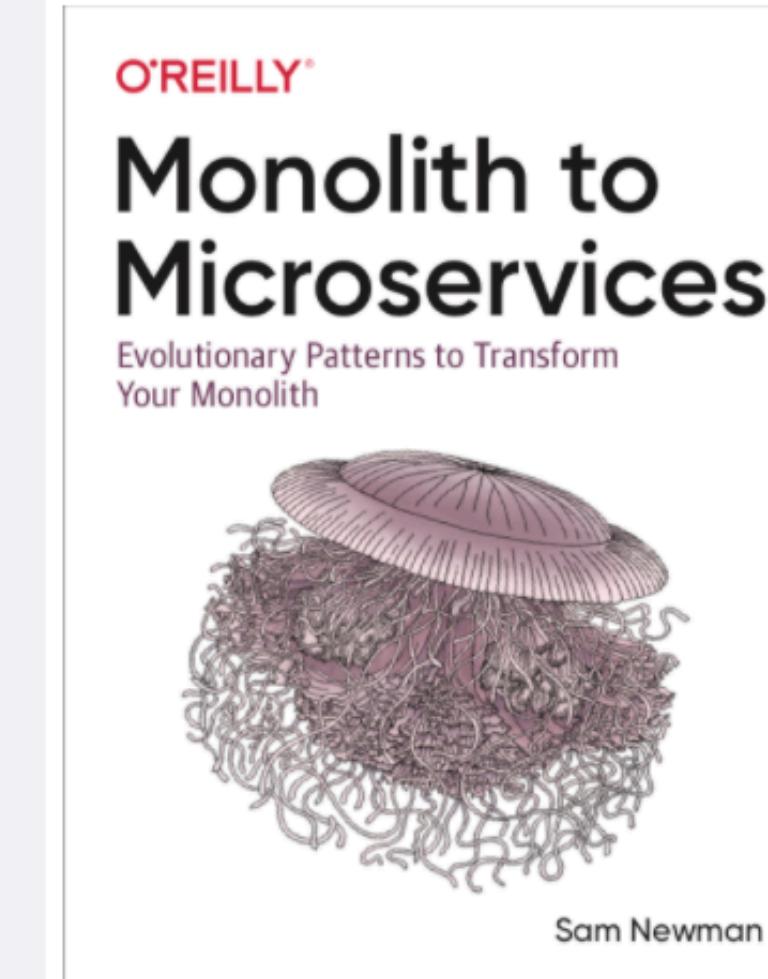
**Monolith To Microservices.**

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