Migrating Monolithic Applications to Microservices with CARGO

Vikram Nitin
Columbia University

Shubhi Asthana
IBM Research

Baishakhi Ray
Columbia University

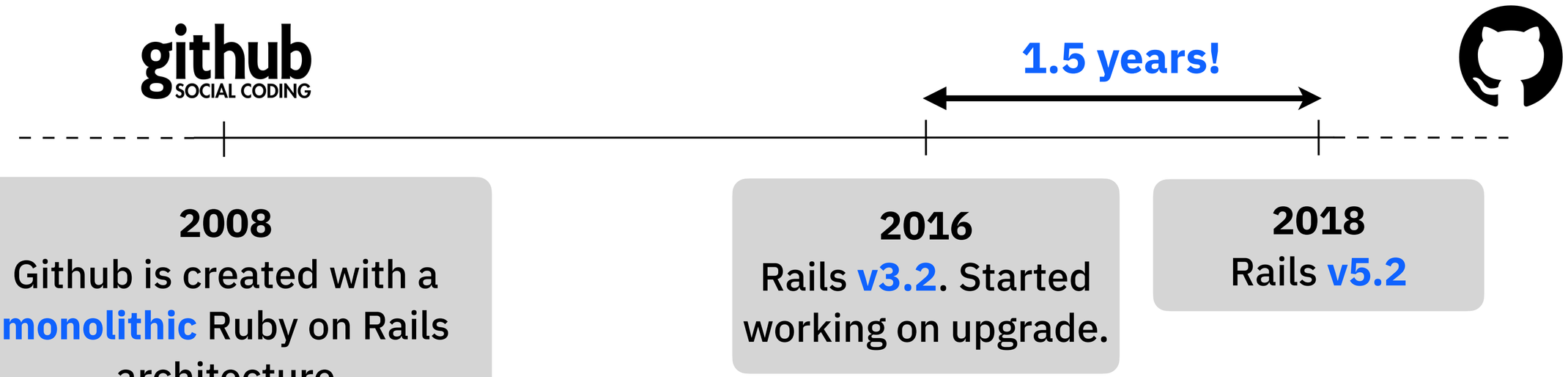
Rahul Krishna
IBM Research

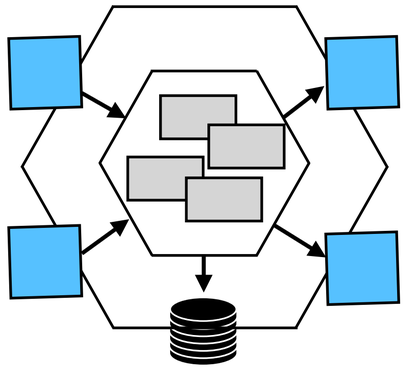
IBM Research

https://arxiv.org/abs/2207.11784

CS@CU

GitHub's Slow March toward Monolithic Hell





Why did it take that much effort do a version upgrade?

Monolithic code can be problematic!

"...having everyone doing development in the same monolithic code base is no longer the most efficient and optimal way to scale GitHub."

Sha Ma,
VP of Software Engineering at ***Github***

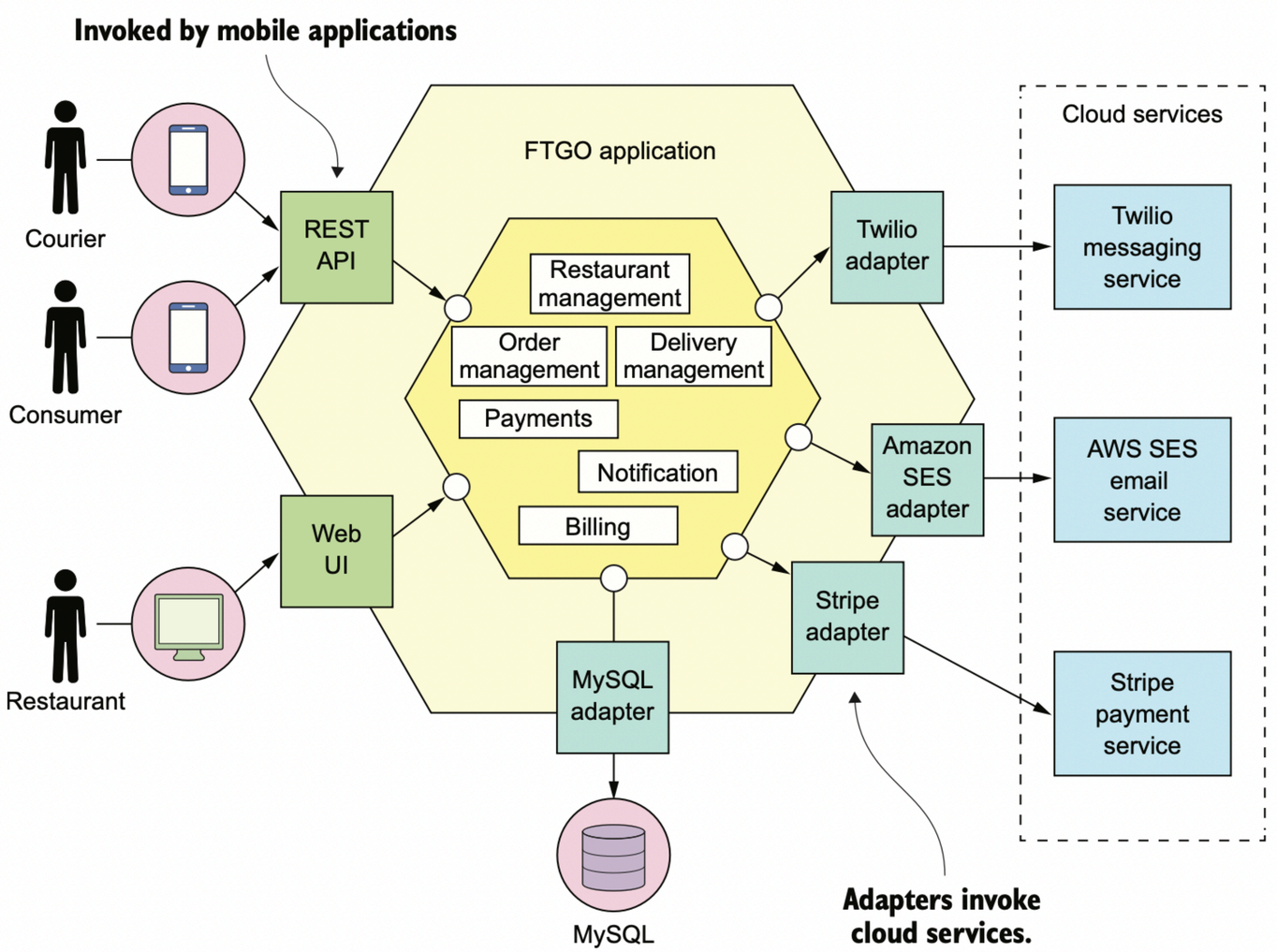
"... over time, as that project matures, as you add more developers on it, as it grows ... that monolith is going to add overhead into your process, and that software development lifecycle is going to begin to slow down."

Rob Brigham,
senior manager at ***Amazon AWS***

1. https://www.infoq.com/presentations/github-rails-monolith-microservices/
2. https://thenewstack.io/led-amazon-microservices-architecture/

Problems with Monolithic Architecture

Large, **complex code structure** is difficult for developers to comprehend

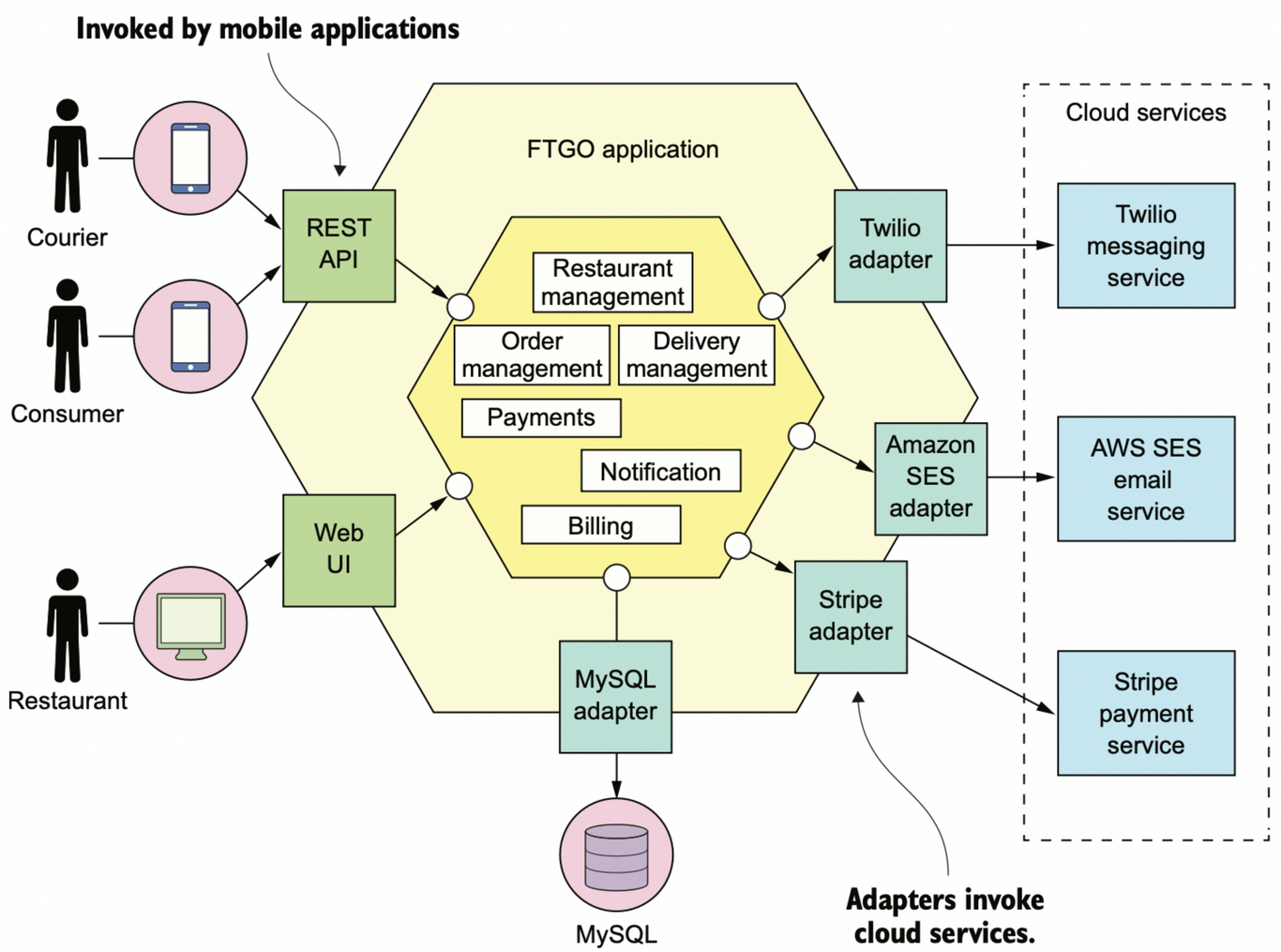


[1] Chris Richardson, "Microservice Patterns", 2018

Problems with Monolithic Architecture

Large, **complex code structure** is difficult for developers to comprehend

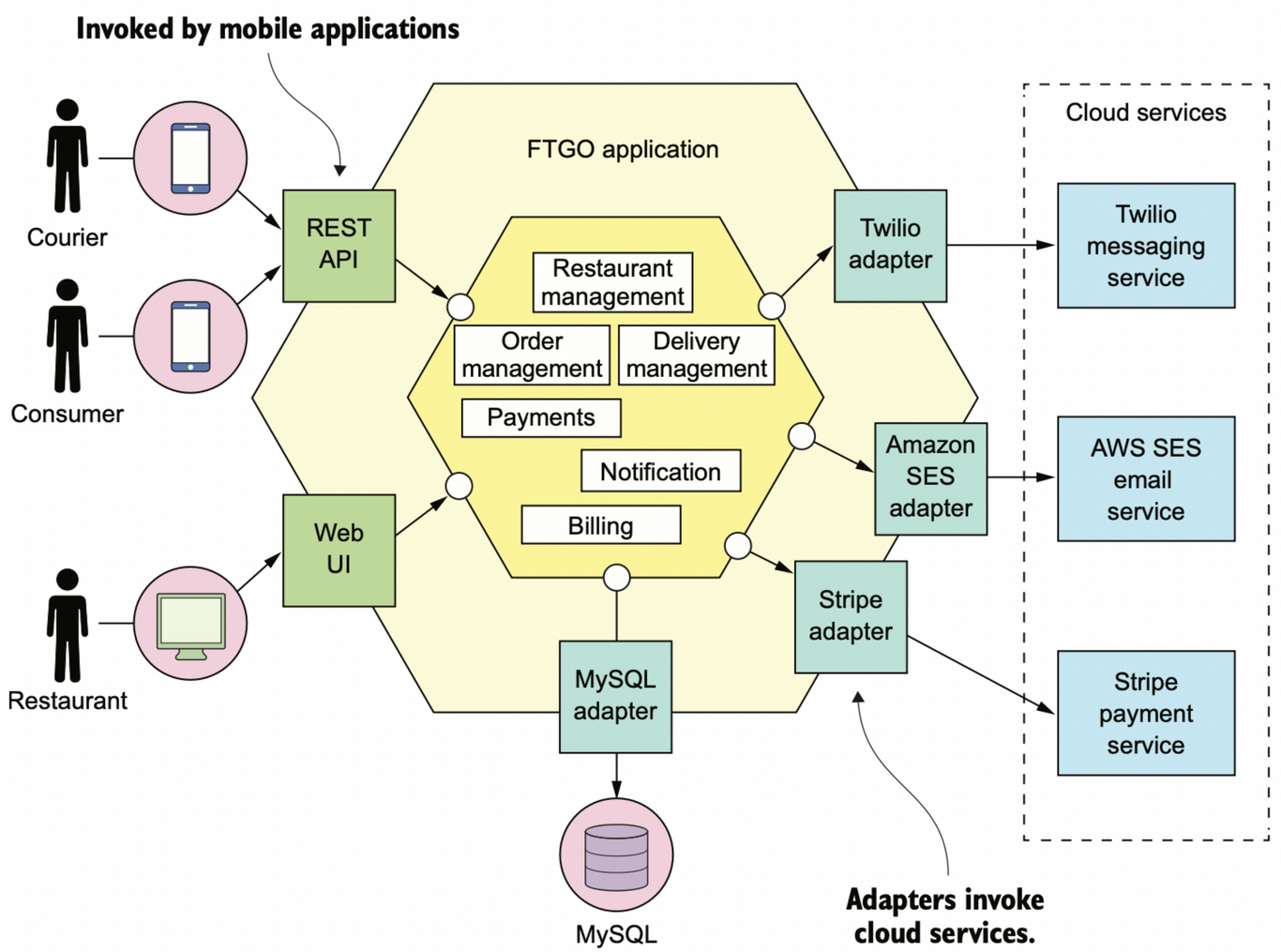
Development is slow because **build time** and start up time are high



[1] Chris Richardson, "Microservice Patterns", 2018

Problems with Monolithic Architecture

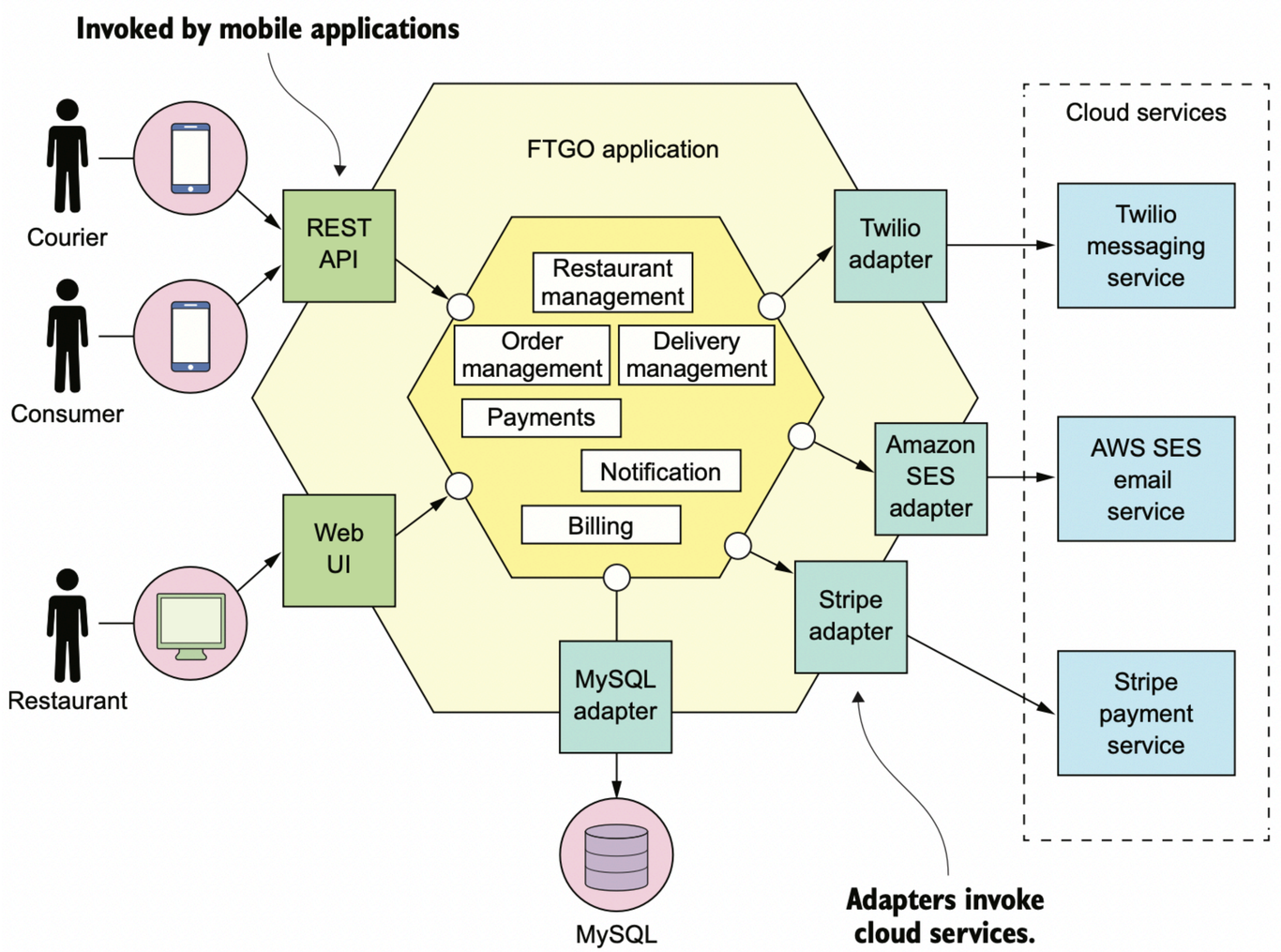
- Large, **complex code structure** is difficult for developers to comprehend
- Development is slow because **build time** and start up time are high
- **Deploying into production** is a long process, so continuous development is difficult



[1] Chris Richardson, "Microservice Patterns", 2018

Problems with Monolithic Architecture

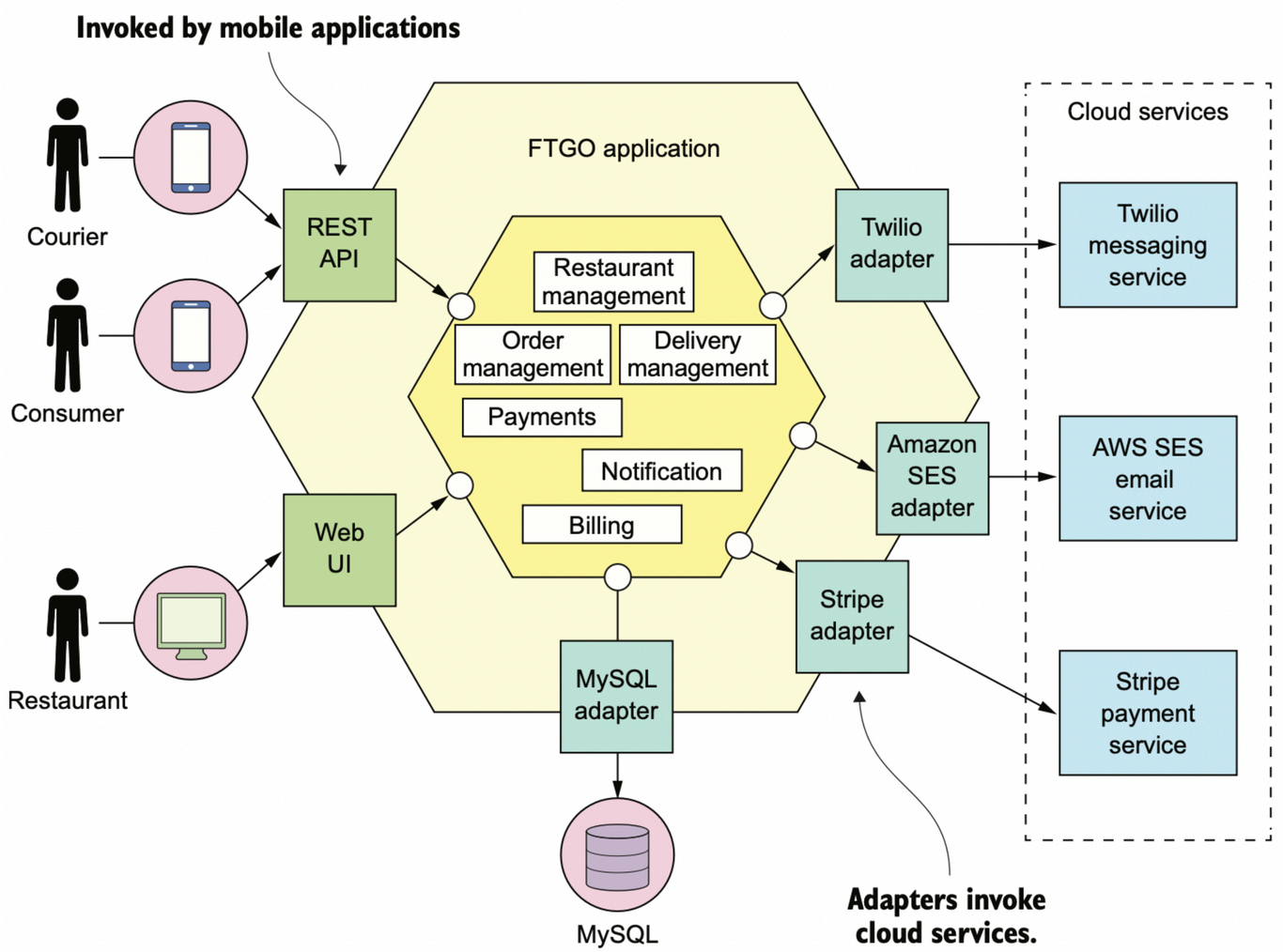
- Large, **complex code structure** is difficult for developers to comprehend
- Development is slow because **build time** and start up time are high
- **Deploying into production** is a long process, so continuous development is difficult
- **Scaling** to higher workloads is difficult



[1] Chris Richardson, "Microservice Patterns", 2018

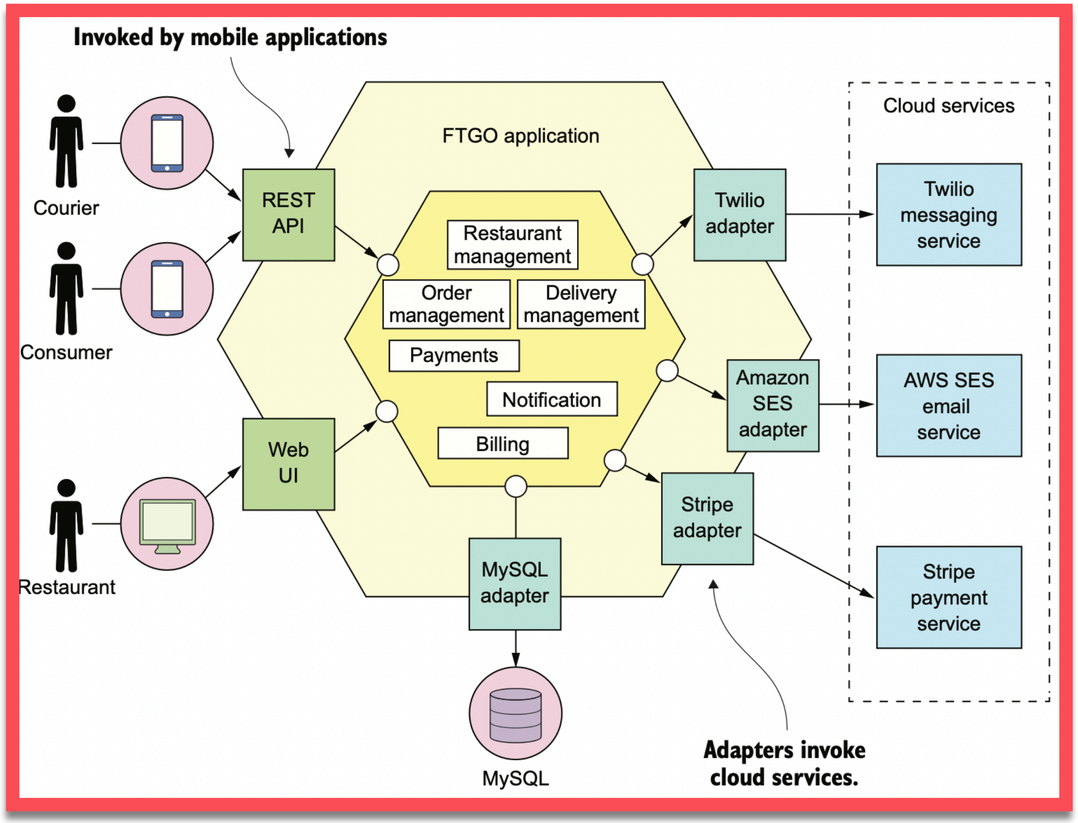
Problems with Monolithic Architecture

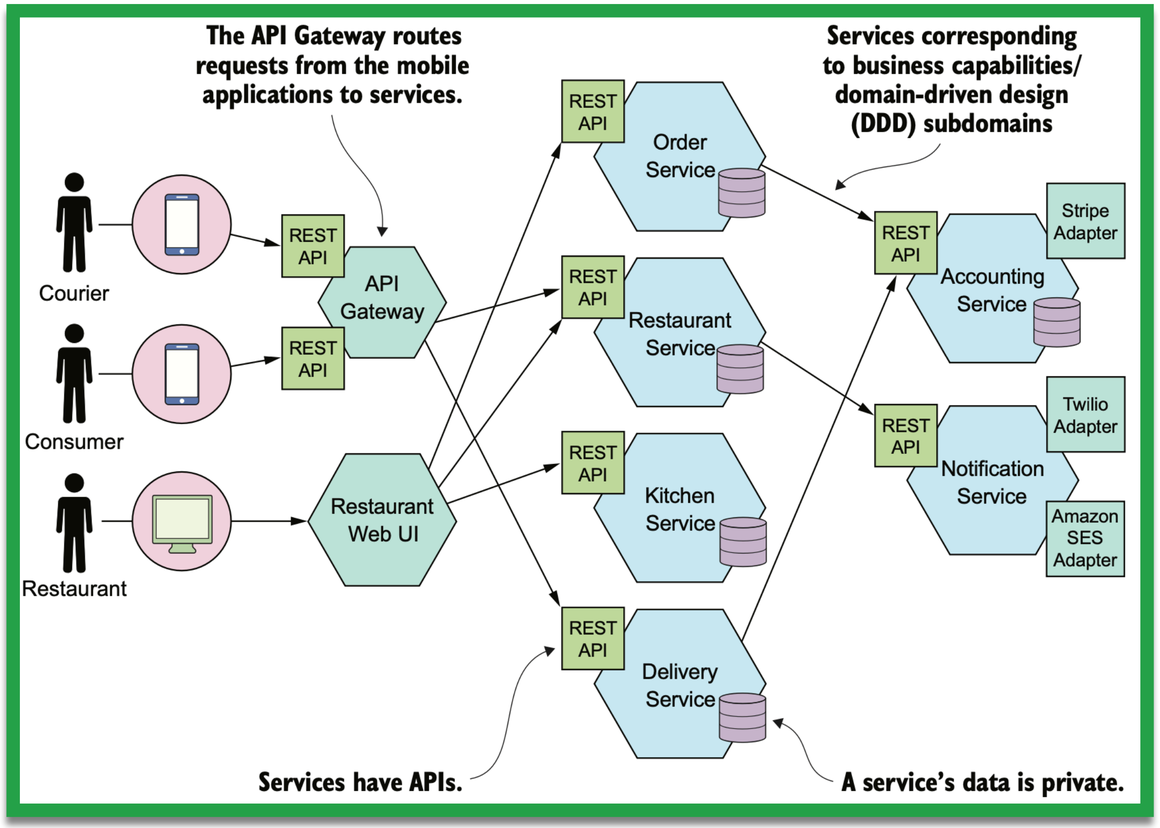
- Large, **complex code structure** is difficult for developers to comprehend
- Development is slow because **build time** and start up time are high
- **Deploying into production** is a long process, so continuous development is difficult
- **Scaling** to higher workloads is difficult
- Locked into **obsolete technology stack**. Difficult to adopt new frameworks



[1] Chris Richardson, "Microservice Patterns", 2018

From Monoliths to Microservices



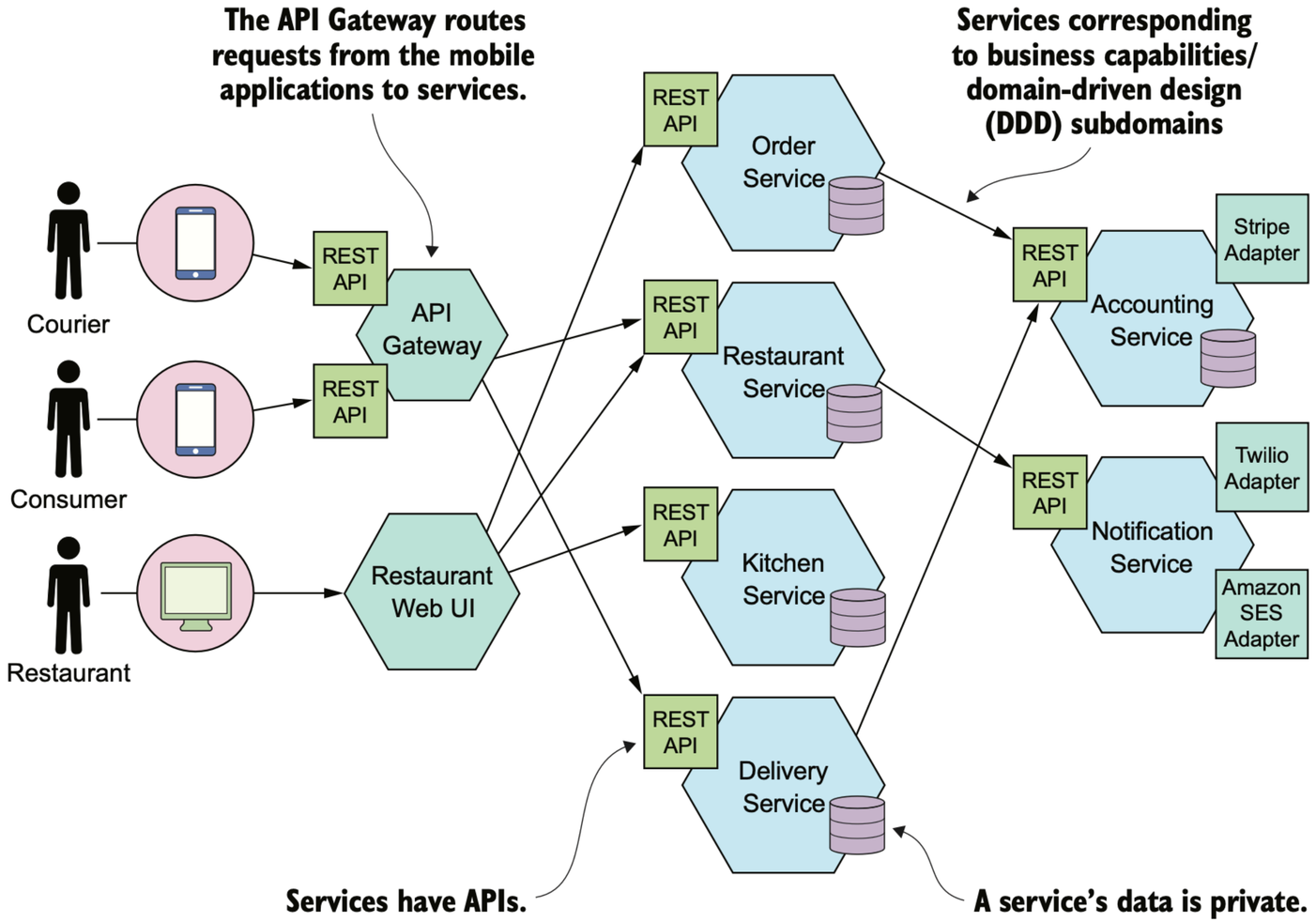


[1] Chris Richardson, "Microservice Patterns", 2018

The Microservice Architecture

Each microservice :

- Has code with **single functionality**

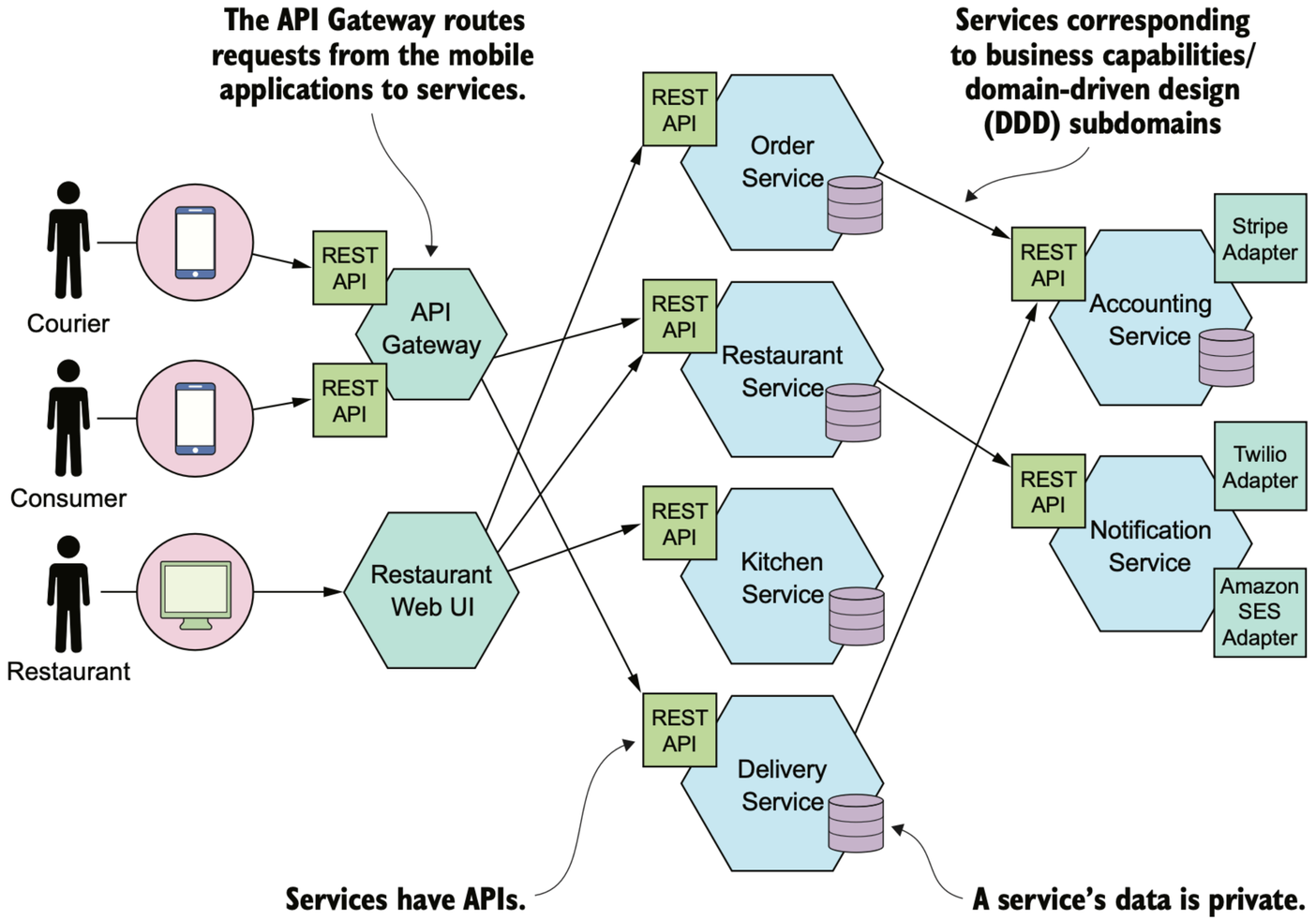


[1] Chris Richardson, "Microservice Patterns", 2018

The Microservice Architecture

Each microservice :

- Has code with **single functionality**
- Has a **private database** on which it performs transactions

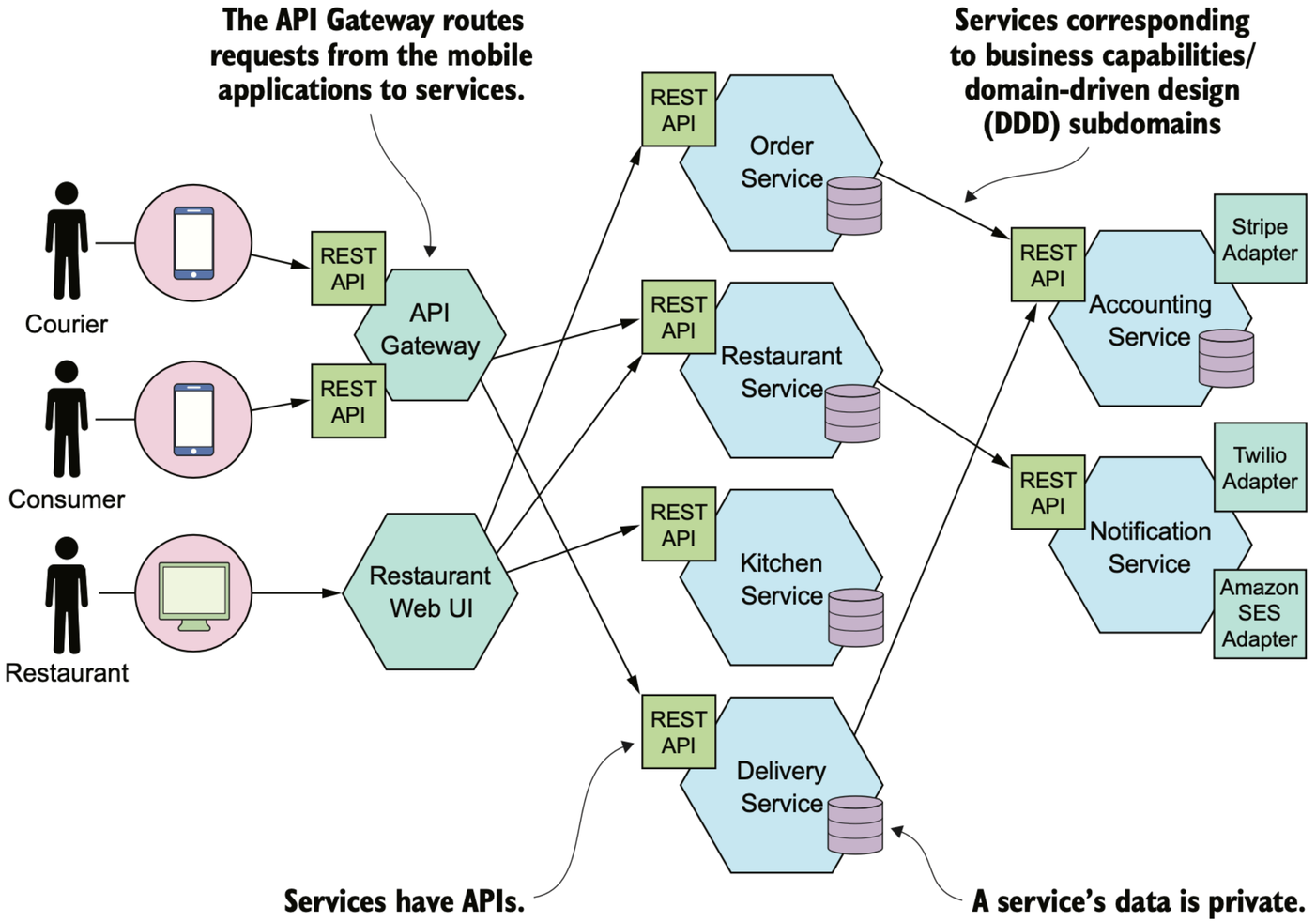


[1] Chris Richardson, "Microservice Patterns", 2018

The Microservice Architecture

Each microservice :

- Has code with **single functionality**
- Has a **private database** on which it performs transactions
- Communicates with the others either **asynchronously** (message bus, queues, etc.) or **synchronously** (REST API, gRPC, etc.)

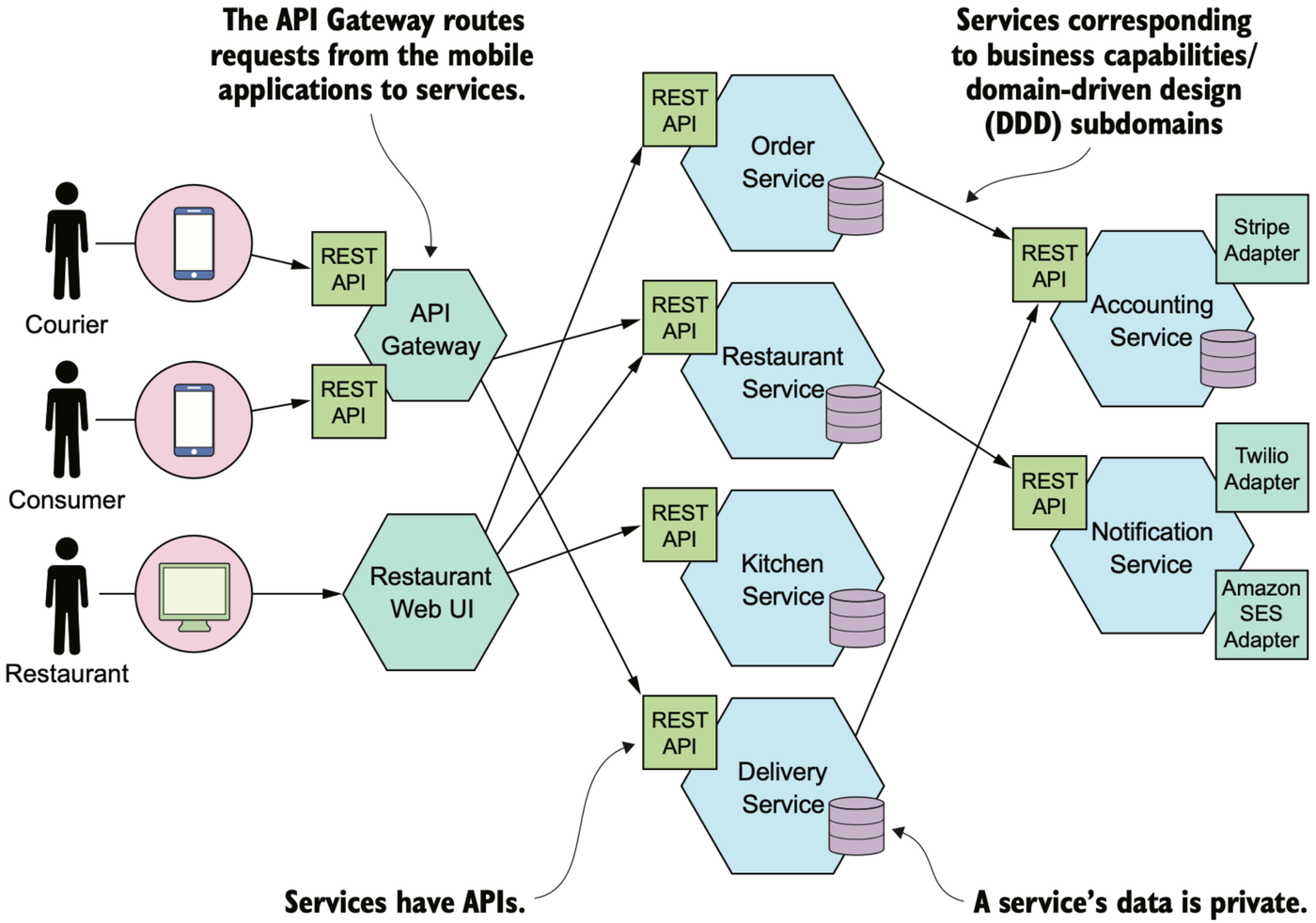


[1] Chris Richardson, "Microservice Patterns", 2018

The Microservice Architecture

Each microservice :

- Has code with **single functionality**
- Has a **private database** on which it performs transactions
- Communicates with the others either **asynchronously** (message bus, queues, etc.) or **synchronously** (REST API, gRPC, etc.)
- Is highly **cohesive** and loosely **coupled** with the other services

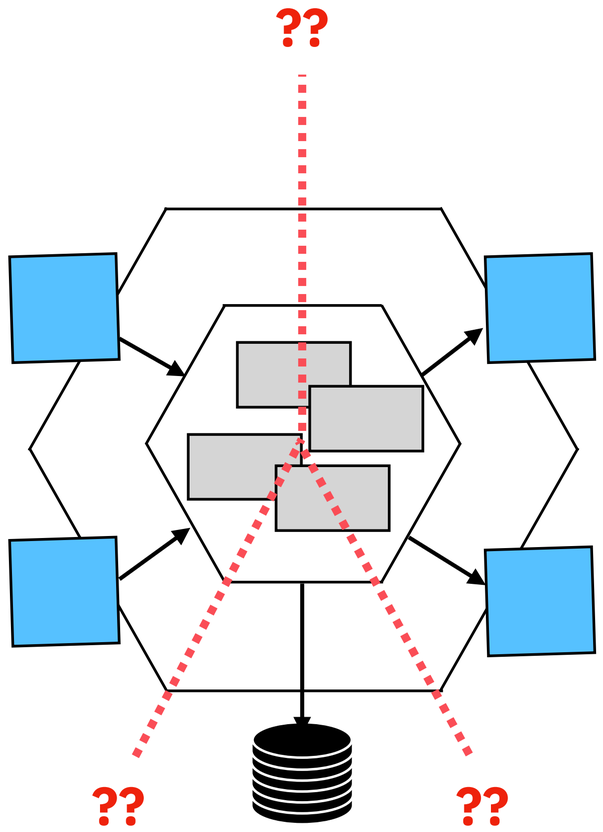


[1] Chris Richardson, "Microservice Patterns", 2018

Decomposing Monoliths into Microservices

This is a hard problem!

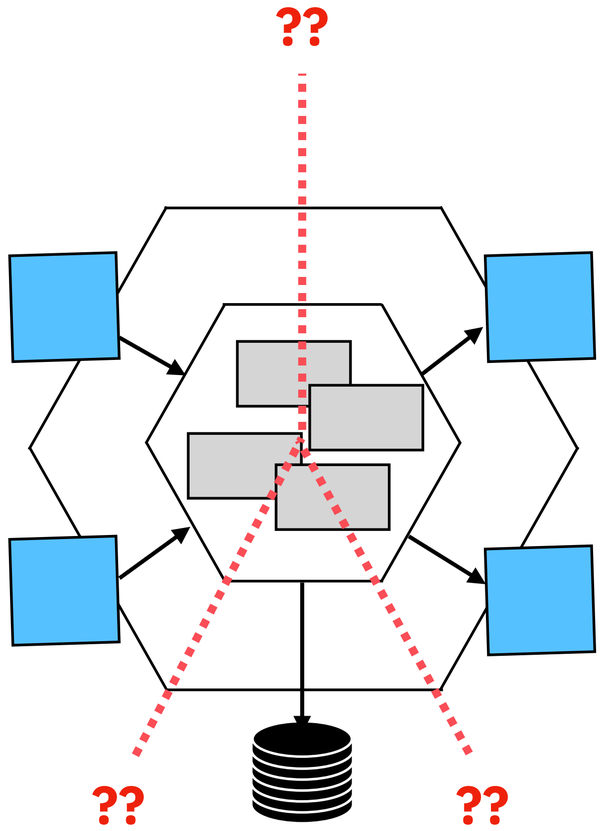
1. Identifying functional boundaries requires considerable **domain expertise**



Decomposing Monoliths into Microservices

This is a hard problem!

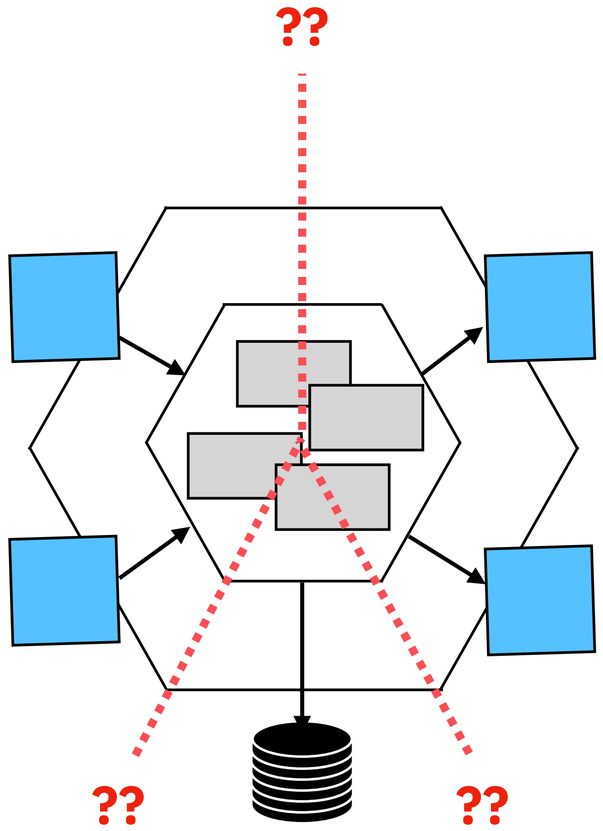
1. Identifying functional boundaries requires considerable **domain expertise**
2. Separating classes according to functionality is not sufficient:
   i. May lead to **Concurrency** issues
   ii. Maintaining **data consistency** is challenging



Decomposing Monoliths into Microservices

This is a hard problem!

1. Identifying functional boundaries requires considerable **domain expertise**
2. Separating classes according to functionality is not sufficient:
   i. May lead to **Concurrency** issues
   ii. Maintaining **data consistency** is challenging



Automated monolith decomposition tools

Automated Monolith Decomposition

Active area of Research

1	Escobar, D. et al.: Towards the understanding and evolution of monolithic applications as microservices. In: Proceedings of 42nd Latin American Computing Conference, CLEI. (2016)
2	Levcovitz, A. et al.: Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. In: 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM) .pp. 97–104 (2015)
3	Ahmadvand, M., Ibrahim, A.: Requirements reconciliation for scalable and secure microservice (de)composition. In: Proceedings - 2016 IEEE 24th International Requirements Engineering Conference Workshops, REW 2016. pp. 68–73 (2016)
4	Baresi, L. et al.: Microservices Identification Through Interface Analysis. In: ESOCC 2017: Service-Oriented and Cloud Computing. pp. 19–33 (2017)
5	Gysel, M. et al.: Service cutter: A systematic approach to service decomposition. In: Lecture Notes in Computer Science. pp. 185–200 (2016)
6	Mazlami, G. et al.: Extraction of Microservices from Monolithic Software Architectures. In: 2017 IEEE International Conference on Web Services (ICWS). pp. 524–531 (2017)
7	Mustafa, O., Gómez, J.M.: Optimizing economics of microservices by planning for granularity level Experience Report. (2017)
8	Hassan, S. et al.: Microservice Ambients: An Architectural Meta-Modelling Approach for

[1] Fritzsch, Jonas, et al. "From monolith to microservices: A classification of refactoring approaches."

Industrial tools

KONVEYOR

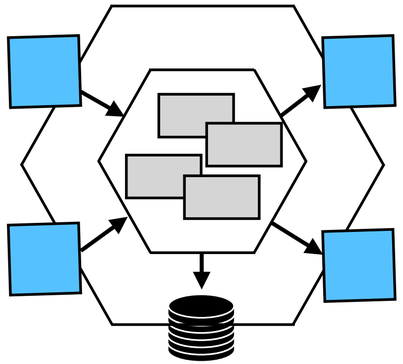
IBM

vFunction

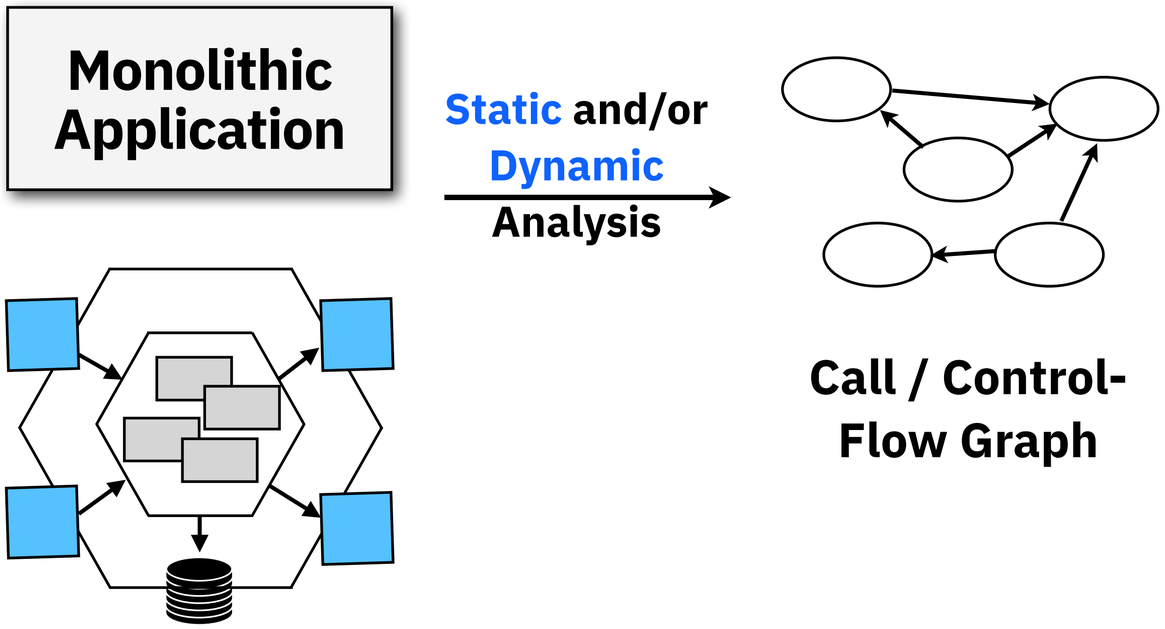
[1] https://www.ibm.com/cloud/mono2micro
[2] https://www.konveyor.io
[3] https://vfunction.com

Automated monolith decomposition tools

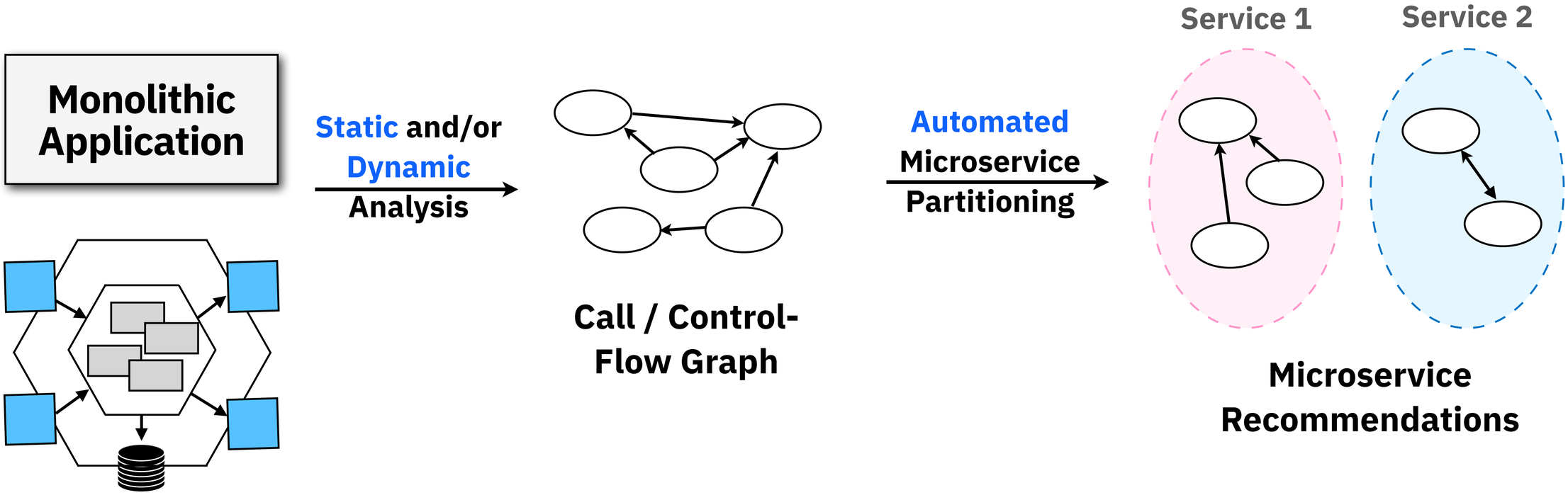
Monolithic Application



Automated monolith decomposition tools

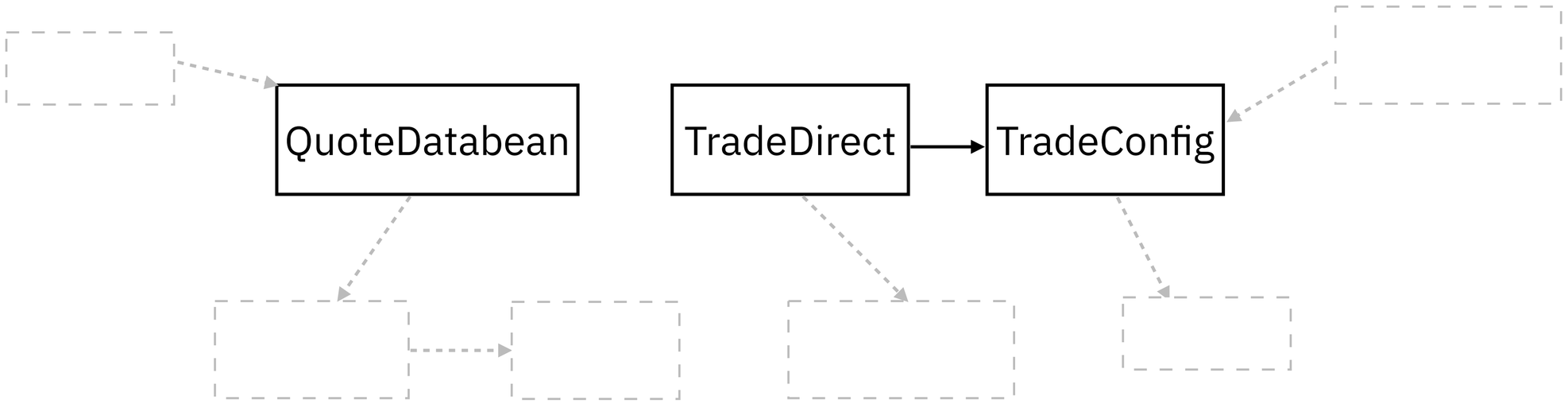


Automated monolith decomposition tools



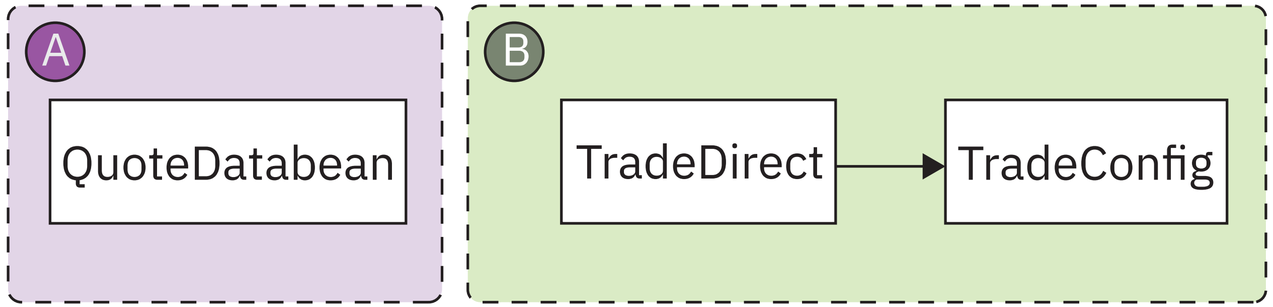
The Problem

A real example from a benchmark application, Daytrader. This is a portion of a **call-graph**, and there is a call edge between the classes `TradeDirect` and `TradeConfig`.



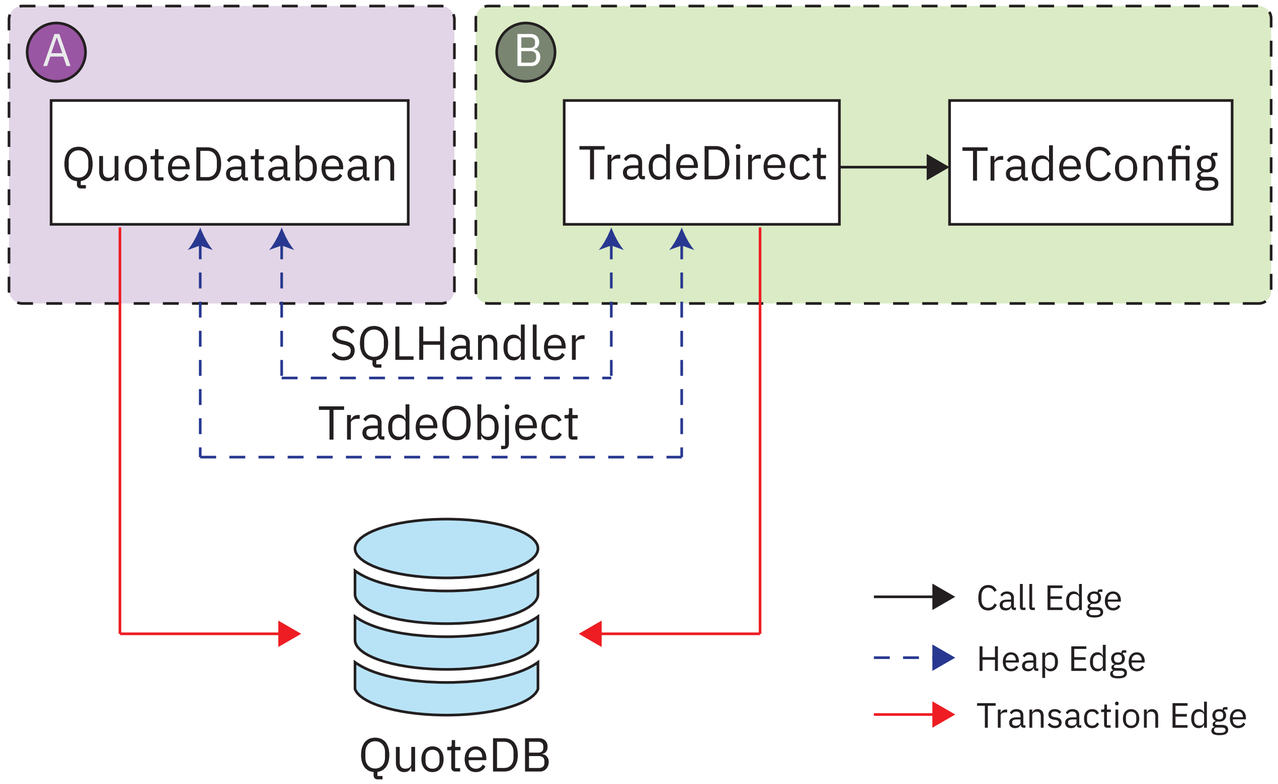
The Problem

In this commonly recommended partitioning, `QuoteDatabean` and `TradeDirect` lie in different partitions.



The Problem

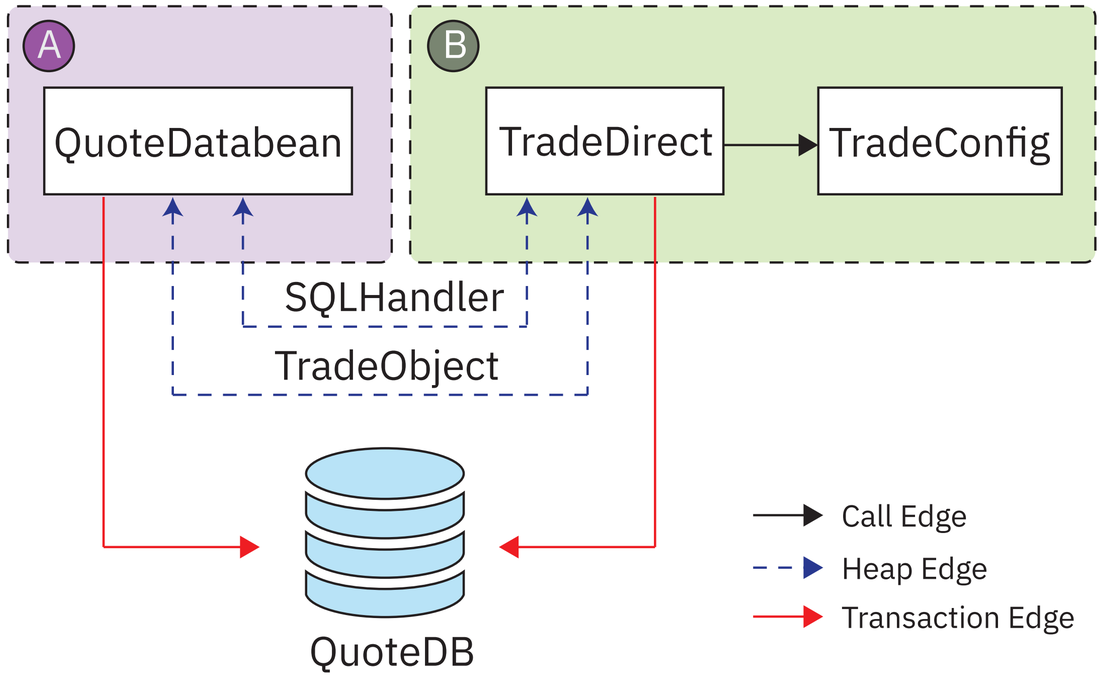
But if we look beyond the call graph, we find this...



The Problem

Some challenges in implementing this partitioning scheme :

1. **Distributed monolith :** The two classes `QuoteDatabean` and `TradeConfig` are tightly coupled (access shared heap objects)

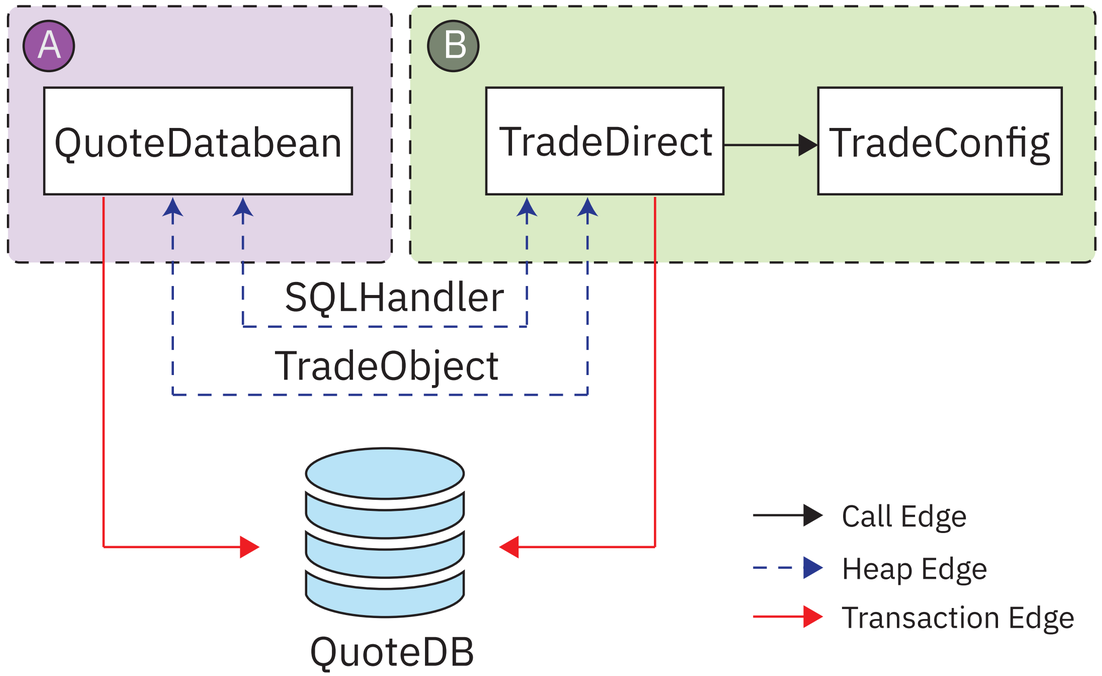


The Problem

Some challenges in implementing this partitioning scheme :

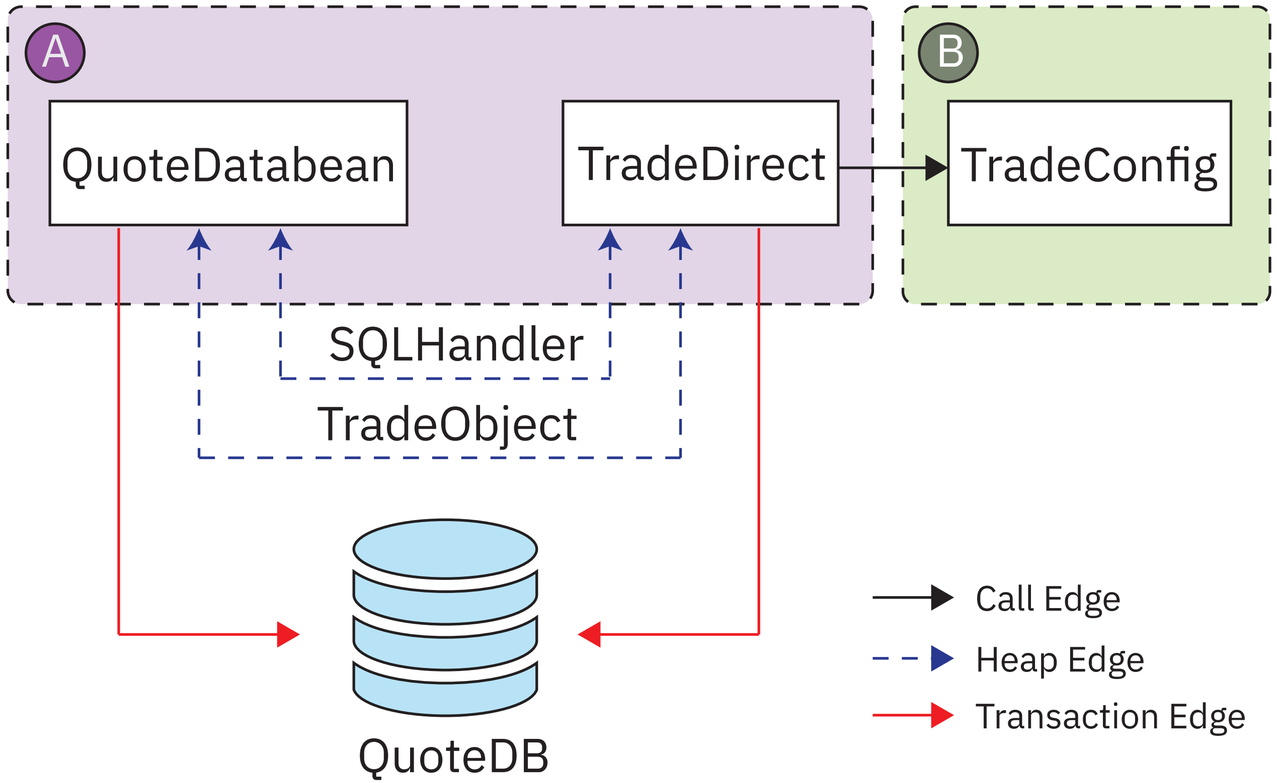
1. **Distributed monolith :** The two classes `QuoteDatabean` and `TradeConfig` are tightly coupled (access shared heap objects)

2. **Distributed transaction :** `QuoteDatabean` and `TradeConfig` write to the same DB.



The Problem

In reality, **QuoteDatabean** and **TradeDirect** are tightly coupled!
Our algorithm, CARGO, groups them in the same partition



How do we get better partitions?

Our algorithm CARGO, uses the following key ideas :

1. **More complete static analysis:** Capture many types of dependencies between classes and build a program dependency graph (PDG)

How do we get better partitions?

Our algorithm CARGO, uses the following key ideas :

1. **More complete static analysis:** Capture many types of dependencies between classes and build a program dependency graph (PDG)
2. **Explicitly model transactions:** Database transactions are added as edges in the PDG

How do we get better partitions?

Our algorithm CARGO, uses the following key ideas:

1. **More complete static analysis:** Capture many types of dependencies between classes and build a program dependency graph (PDG)
2. **Explicitly model transactions:** Database transactions are added as edges in the PDG.
3. **Detect communities in the PDG:** We present a novel community detection algorithm to assign partitions to nodes in the PDG.

Our System - Step 1

Extract a Context-sensitive Program Dependency Graph

Context-Sensitivity

```
1   void main(Object[] args) {
2     A a1 = new A();
3     Object v1 = a1.foo(new Object());
4
5     A a2 = new A();
6     Object v2 = a2.foo(new Object());
7   }
```

```
1   class A {
2     Object foo(Object v) {
3       B b = new B();
4       return b.bar(v);
5     }
6   }
```

```
1   class B {
2     Object bar(Object v) {
3       . . .
4     }
5   }
6
```

Context-Sensitivity

Context-insensitive Analysis

main()

```
1   void main(Object[] args) {
2     A a1 = new A(); // A/1
3     Object v1 = a1.foo(new Object());
4
5     A a2 = new A(); // A/2
6     Object v2 = a2.foo(new Object());
7   }
```

```
1   class A {
2     Object foo(Object v) {
3       B b = new B();
4       return b.bar(v);
5     }
6   }
```

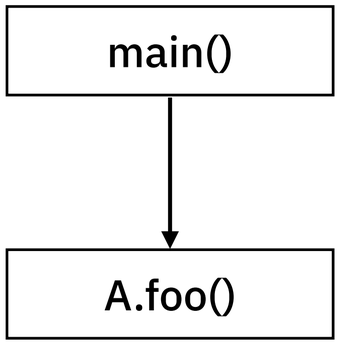
```
1   class B {
2     Object bar(Object v) {
3       . . .
4     }
5   }
6
```

Context-sensitive Analysis

[Φ, Φ] main()

Context-Sensitivity

Context-insensitive Analysis

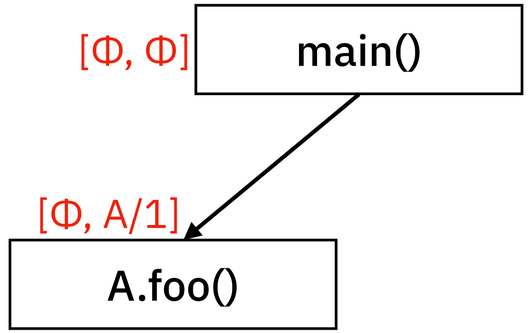



```
1   void main(Object[] args) {
2     A a1 = new A(); // A/1
3     Object v1 = a1.foo(new Object());
4
5     A a2 = new A(); // A/2
6     Object v2 = a2.foo(new Object());
7   }
```

```
1   class A {
2     Object foo(Object v) {
3       B b = new B();
4       return b.bar(v);
5     }
6   }
```

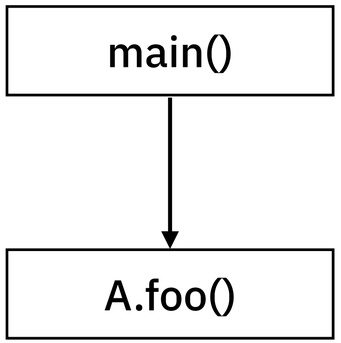
```
1   class B {
2     Object bar(Object v) {
3       . . .
4     }
5   }
6
```

Context-sensitive Analysis



Context-Sensitivity

Context-insensitive Analysis

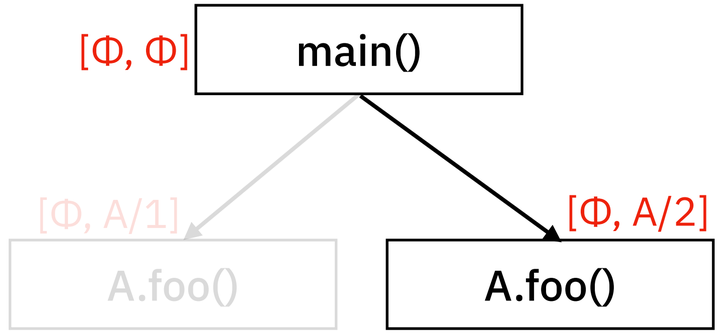



```
1   void main(Object[] args) {
2     A a1 = new A(); // A/1
3     Object v1 = a1.foo(new Object());
4
5     A a2 = new A(); // A/2
6     Object v2 = a2.foo(new Object());
7   }
```

```
1   class A {
2     Object foo(Object v) {
3       B b = new B();
4       return b.bar(v);
5     }
6   }
```

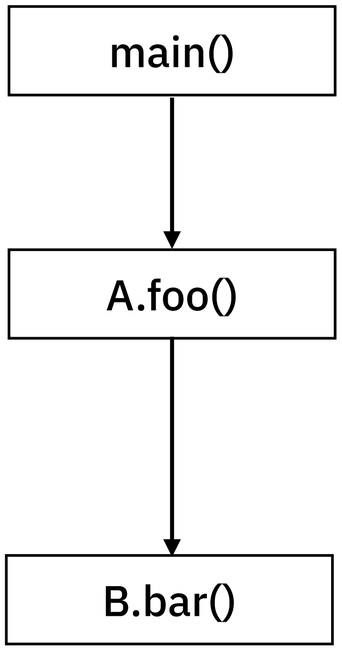
```
1   class B {
2     Object bar(Object v) {
3       . . .
4     }
5   }
6
```

Context-sensitive Analysis



Context-Sensitivity

Context-*insensitive* Analysis



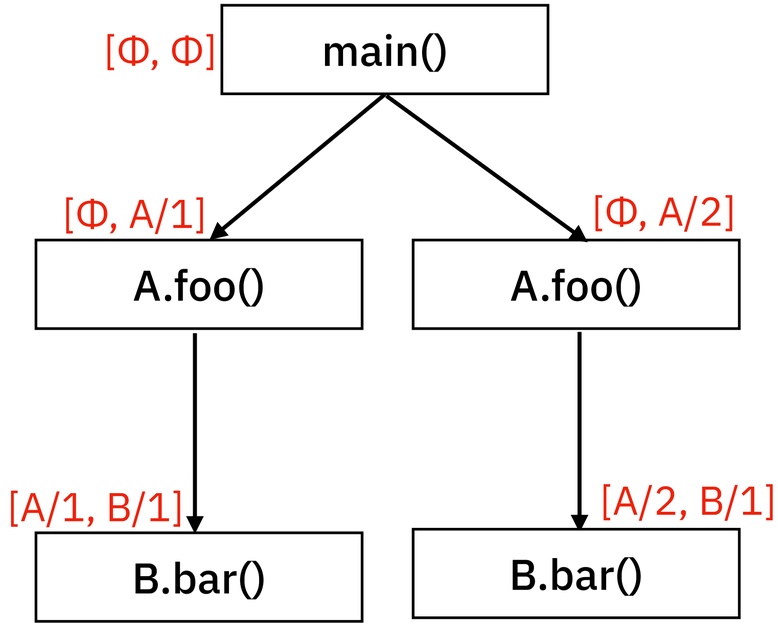

```java
void main(Object[] args) {
  A a1 = new A(); // A/1
  Object v1 = a1.foo(new Object());

  A a2 = new A(); // A/2
  Object v2 = a2.foo(new Object());
}
```

```java
class A {
  Object foo(Object v) {
    B b = new B(); // B/1
    return b.bar(v);
  }
}
```

```java
class B {
  Object bar(Object v) {
    . . .
  }
}
```

Context-*sensitive* Analysis

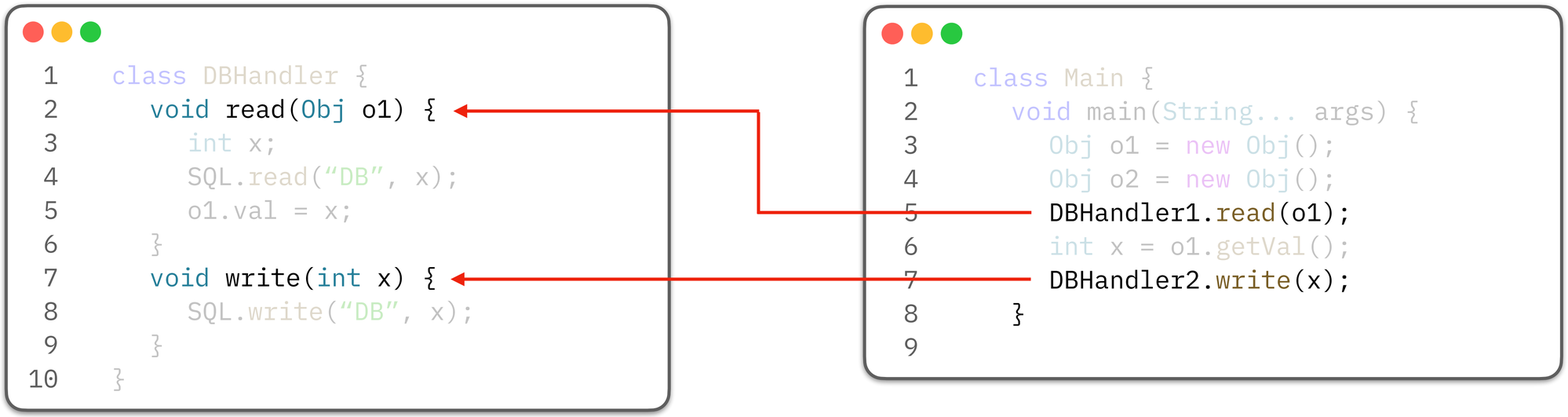


Building a Program Dependency Graph

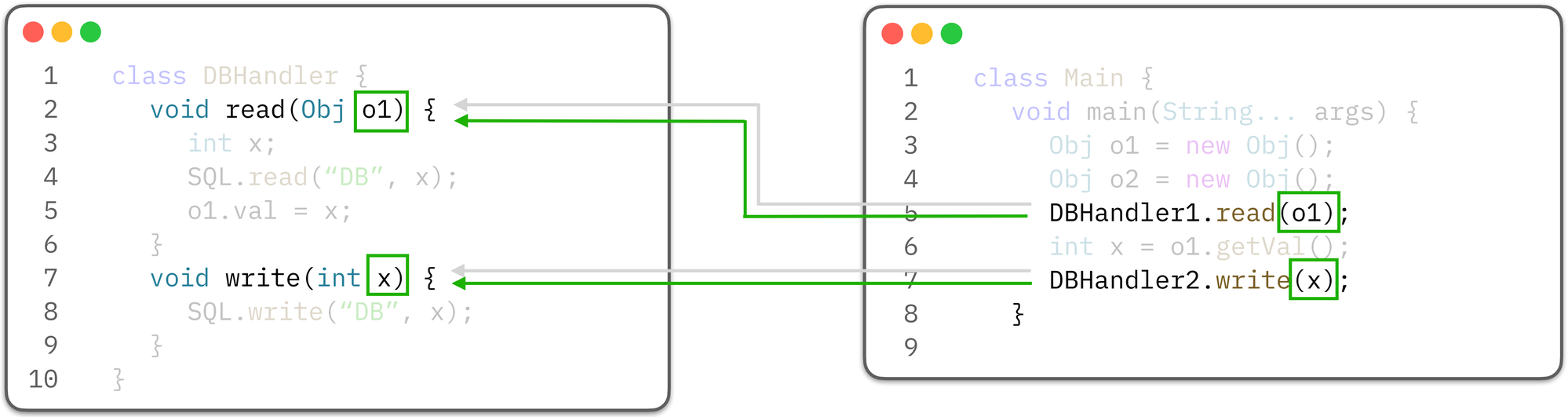
```
1    class DBHandler {
2       void read(Obj o1) {
3          int x;
4          SQL.read("DB", x);
5          o1.val = x;
6       }
7       void write(int x) {
8          SQL.write("DB", x);
9       }
10   }
```

```
1    class Main {
2       void main(String... args) {
3          Obj o1 = new Obj();
4          Obj o2 = new Obj();
5          DBHandler1.read(o1);
6          int x = o1.getVal();
7          DBHandler2.write(x);
8       }
9
```

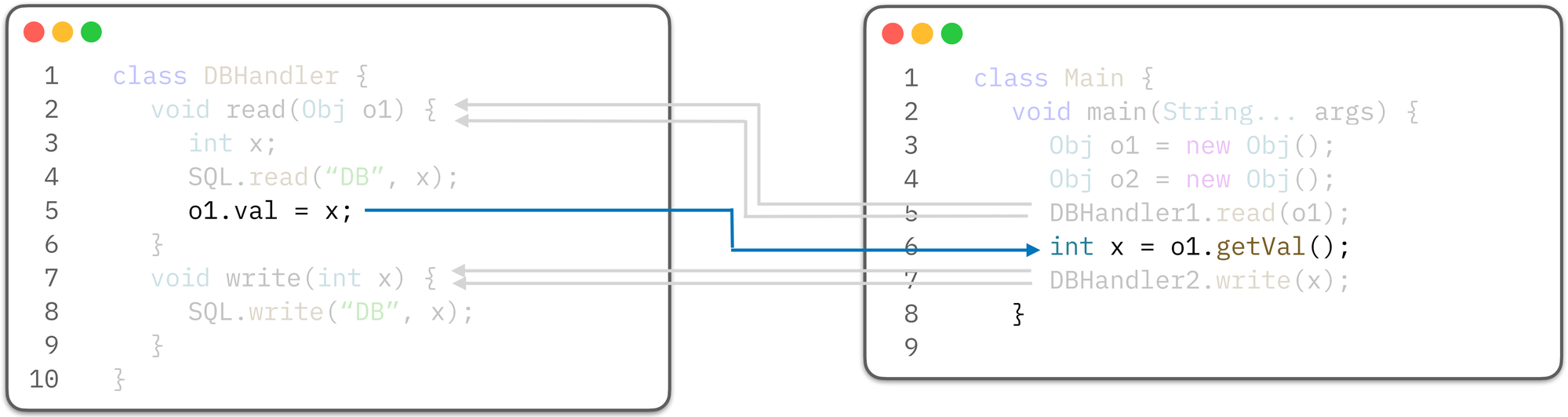
Building a Program Dependency Graph



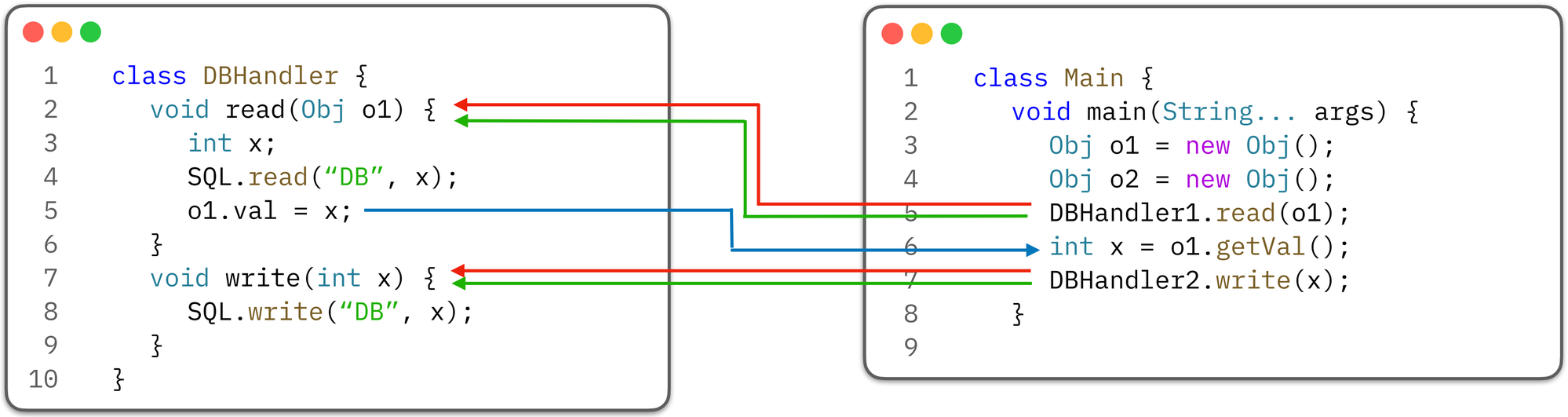
Building a Program Dependency Graph



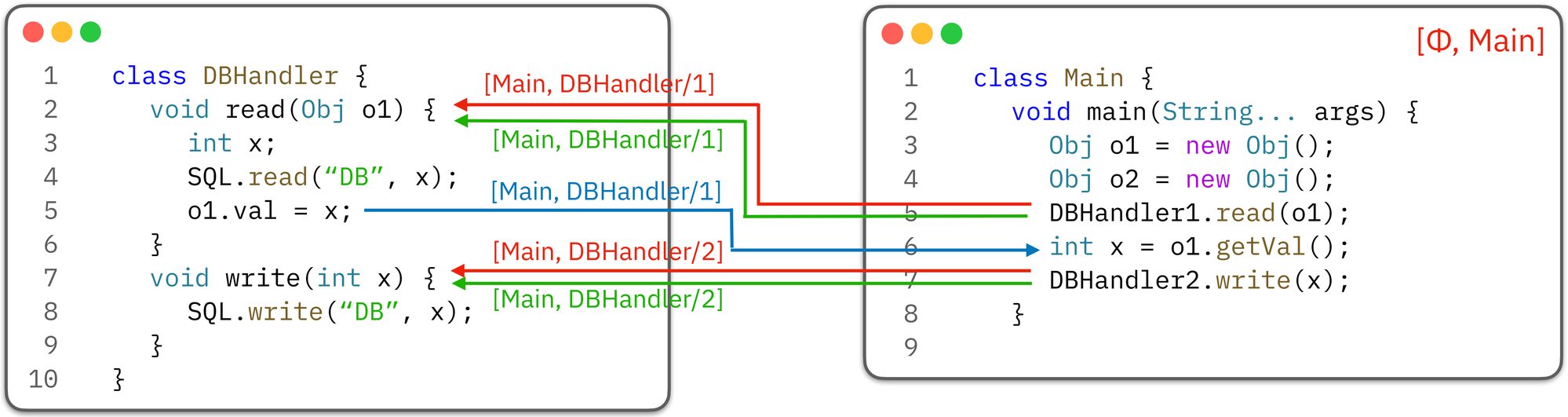
Building a Program Dependency Graph



Building a Program Dependency Graph



Building a Program Dependency Graph

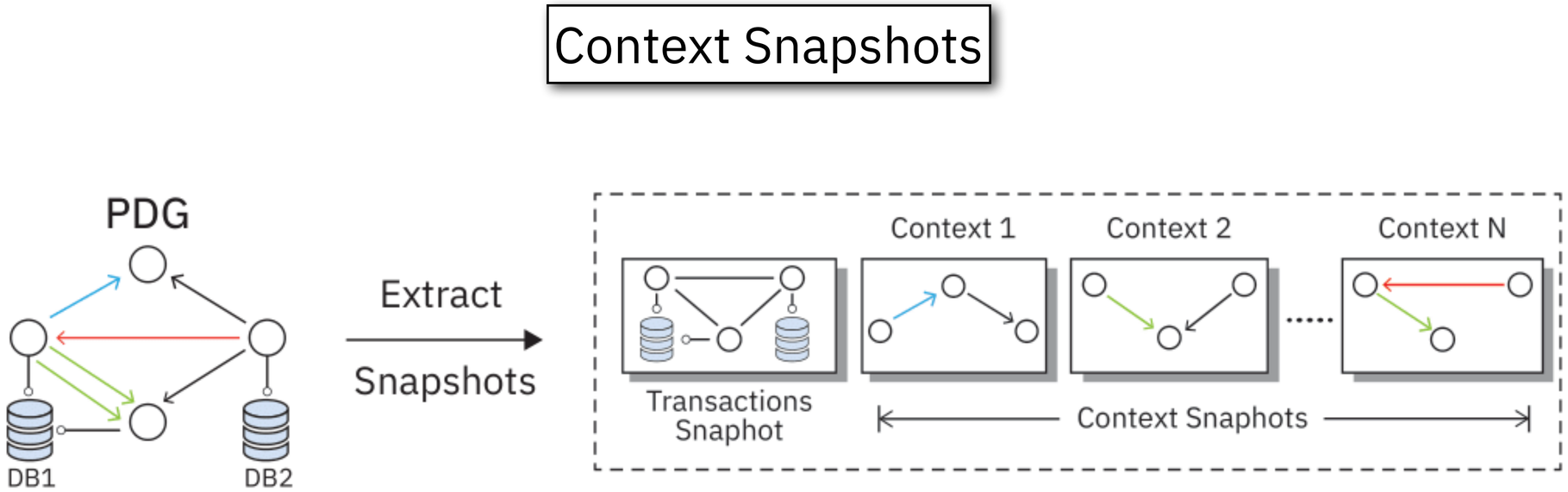


Call-return dependency

Dataflow dependency

Heap dependency

Our System - Step 2

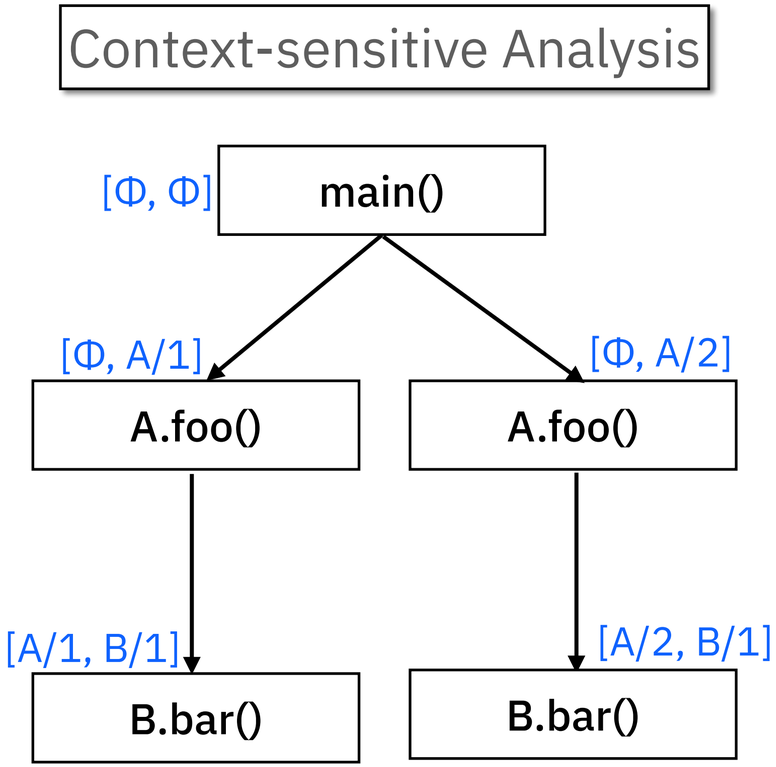


Context-sensitive Analysis

```
1   void main(Object[] args) {
2     A a1 = new A();
3     Object v1 = a1.foo(new Object());
4
5     A a2 = new A();
6     Object v2 = a2.foo(new Object());
7   }
```

```
1   class A {
2     Object foo(Object v) {
3       B b = new B();
4       return b.bar(v);
5     }
6   }
```

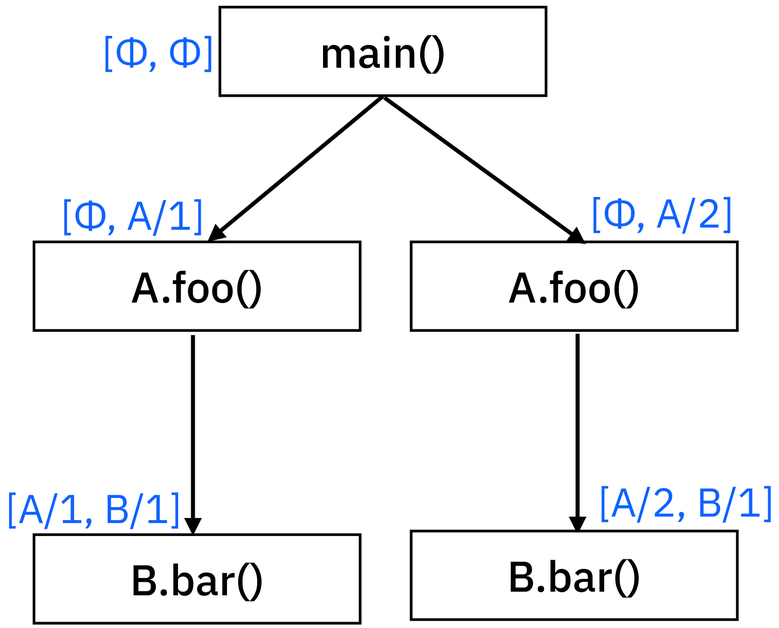
```
1   class B {
2     Object bar(Object v) {
3       . . .
4     }
5   }
6
```



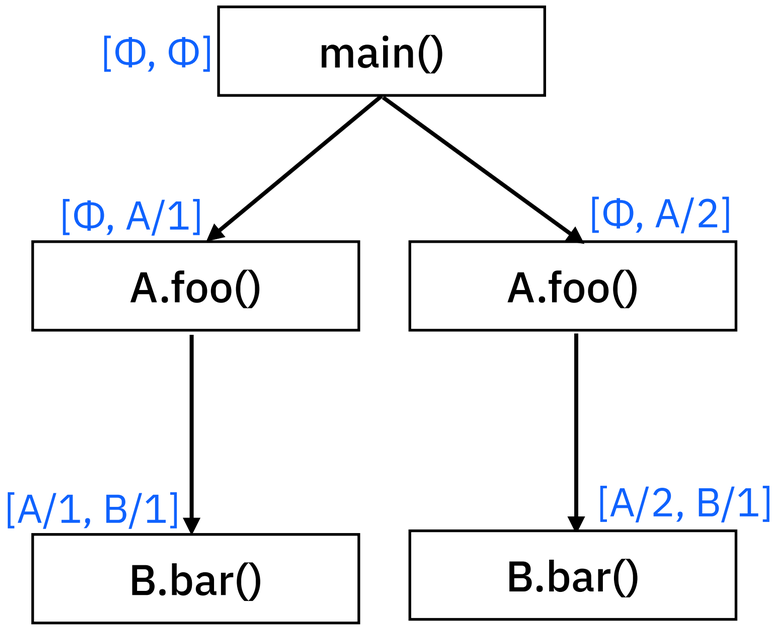
Context Snapshots

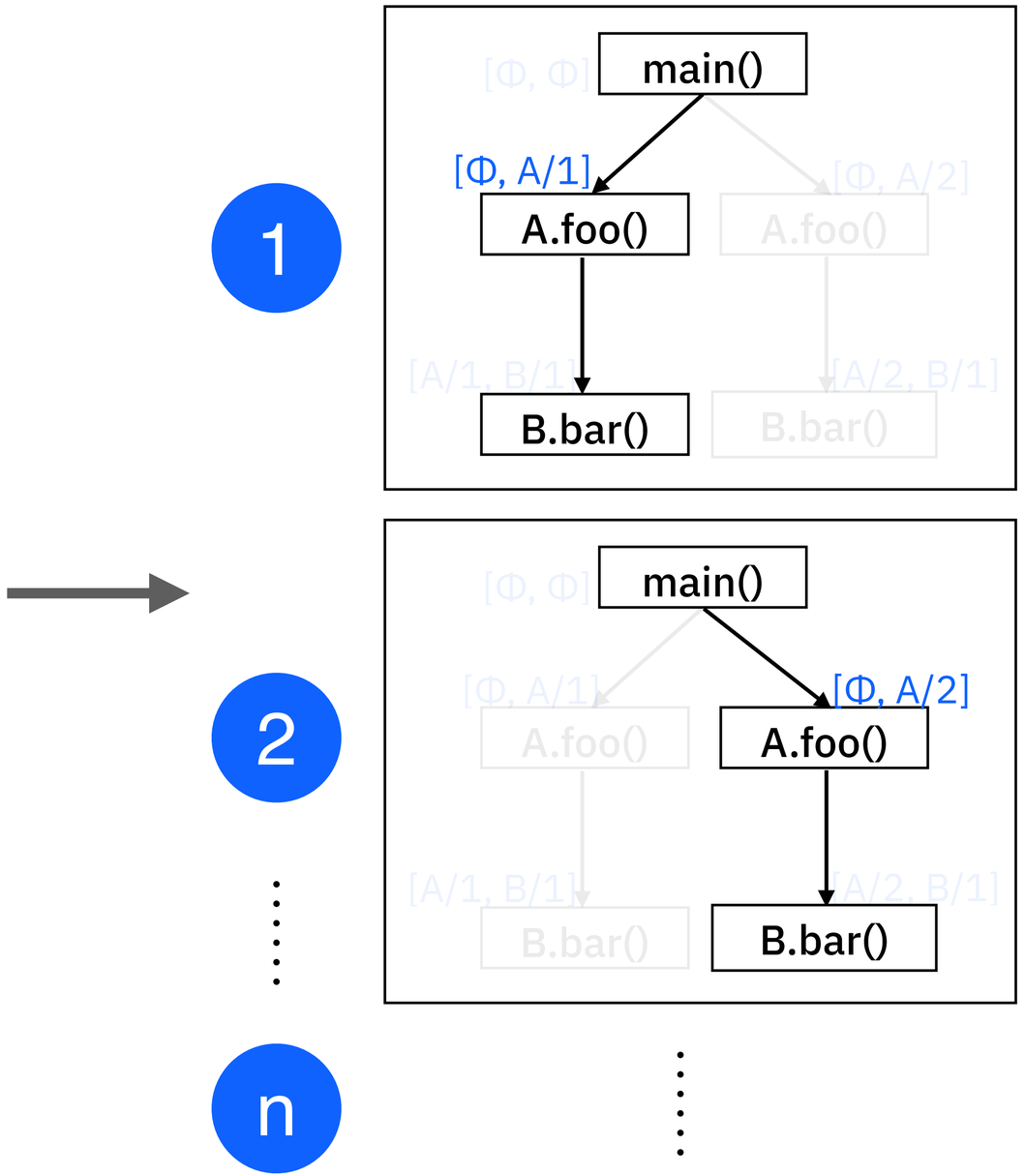
The context-sensitive PDG is a **superposition** of all possible contexts.

At any time, `A.foo()` and `B.bar()` can exist in *any one context* *but not* *both simultaneously*.

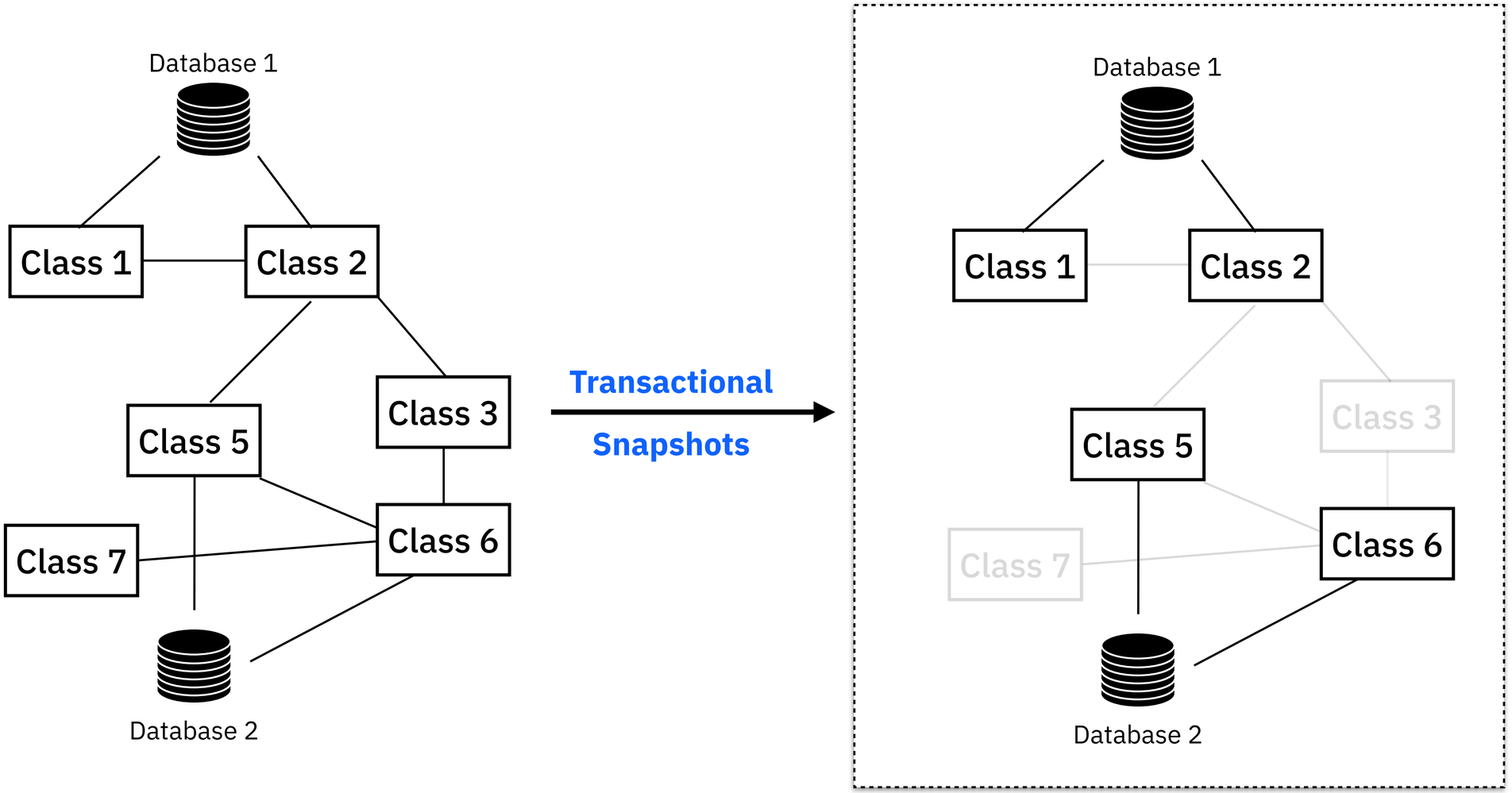


Context Snapshots





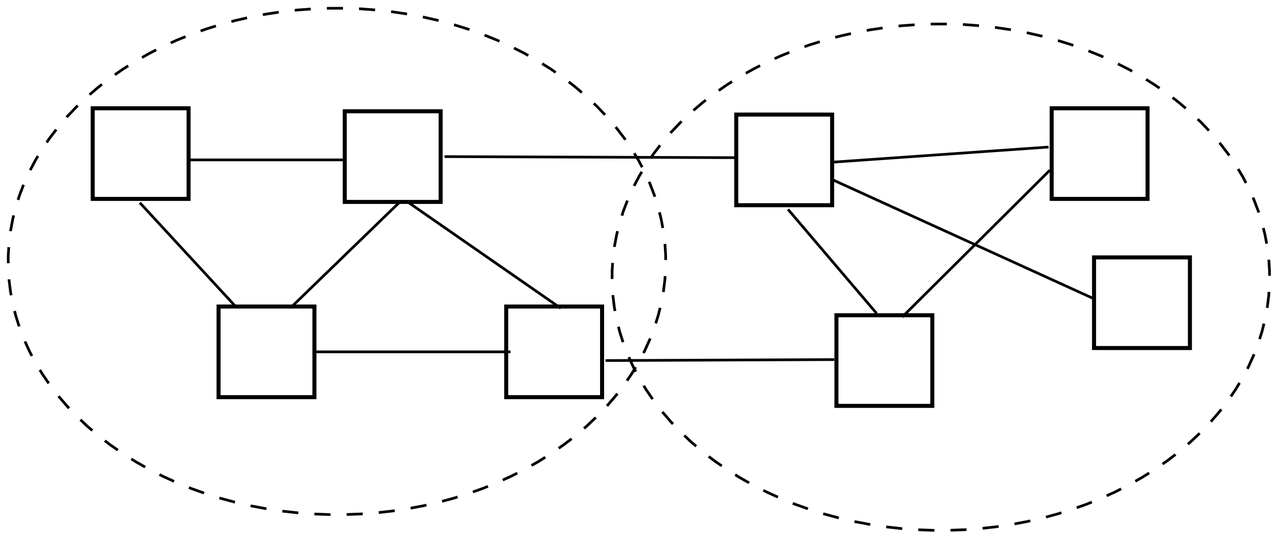
Transactional Snapshot



Our System - Step 3

Context-sensitive Label Propagation

Label propagation is an algorithm to detect communities of nodes in a graph

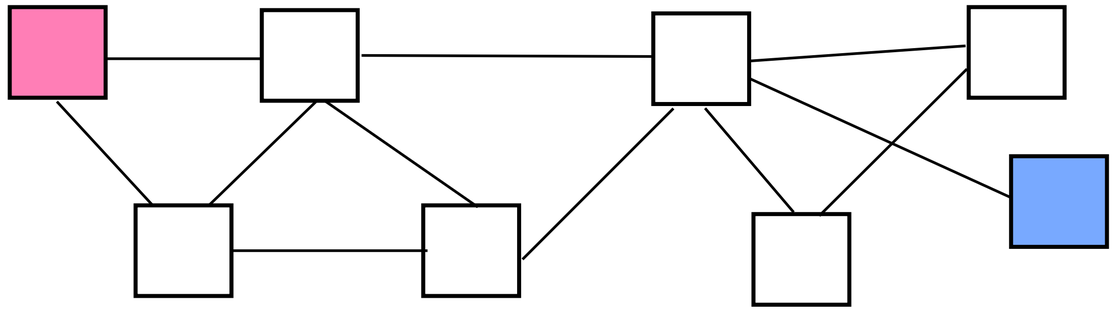


Label Propagation Algorithm

Initial State

We start with some **initial assignment** of labels to nodes

Magenta and **Blue** are the two categories of label

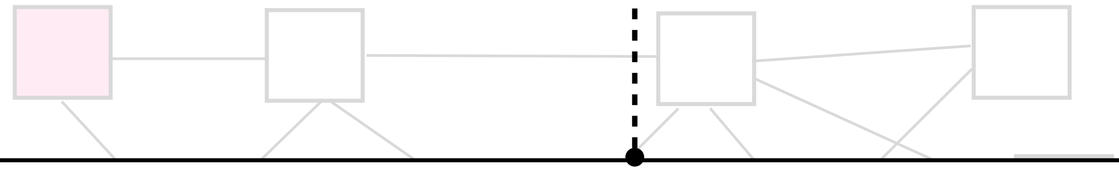


Label Propagation Algorithm

Initial State

We start with some **initial assignment** of labels to nodes

Magenta and **Blue** are the two categories of label



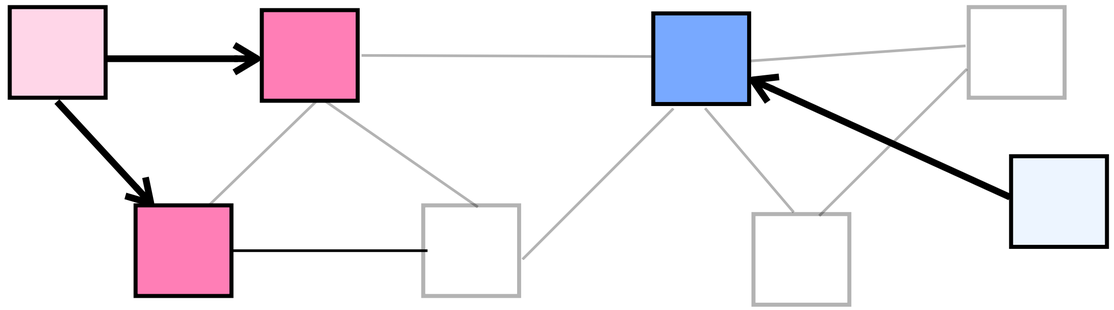
a) Random (**unsupervised**)

b) From the output of another algorithm (**semi-supervised**)

Label Propagation Algorithm

Pass 1

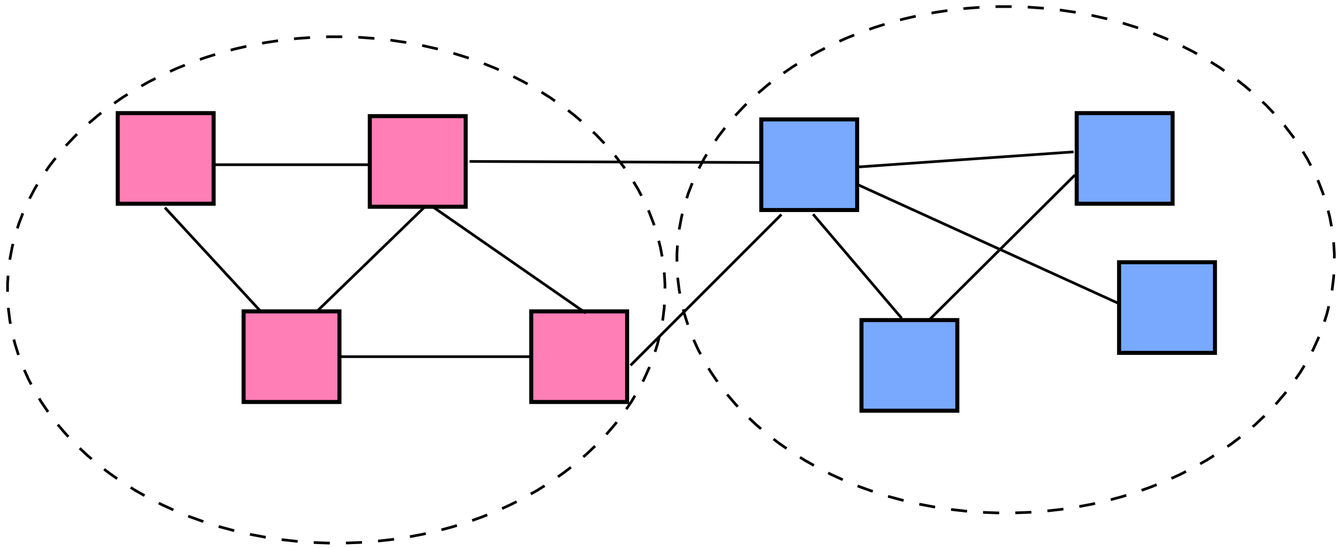
Each node is assigned the majority label of its neighbors



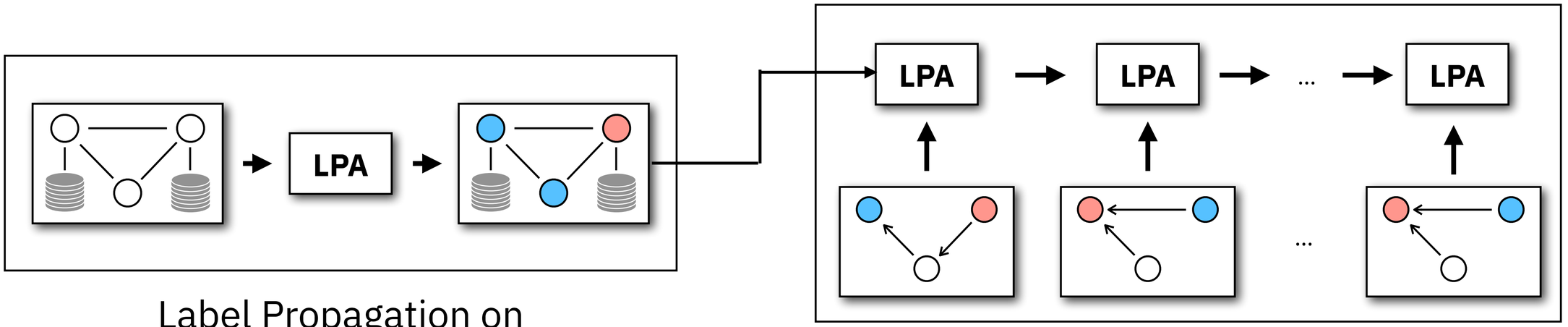
Label Propagation Algorithm

Final

Repeat until convergence (no more node updates)



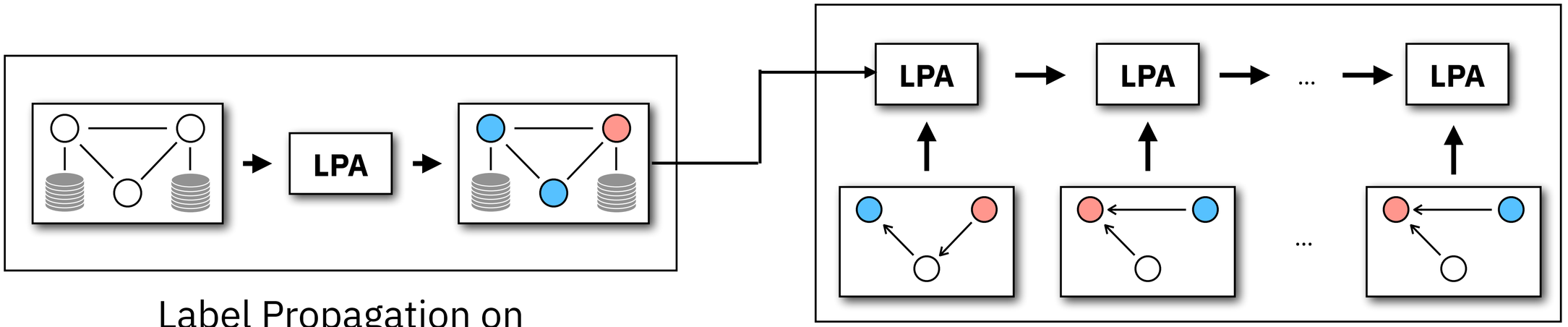
Context-sensitive Label Propagation



Label Propagation on
Transactional Snapshot

Label Propagation on
Context Snapshots

Context-sensitive Label Propagation

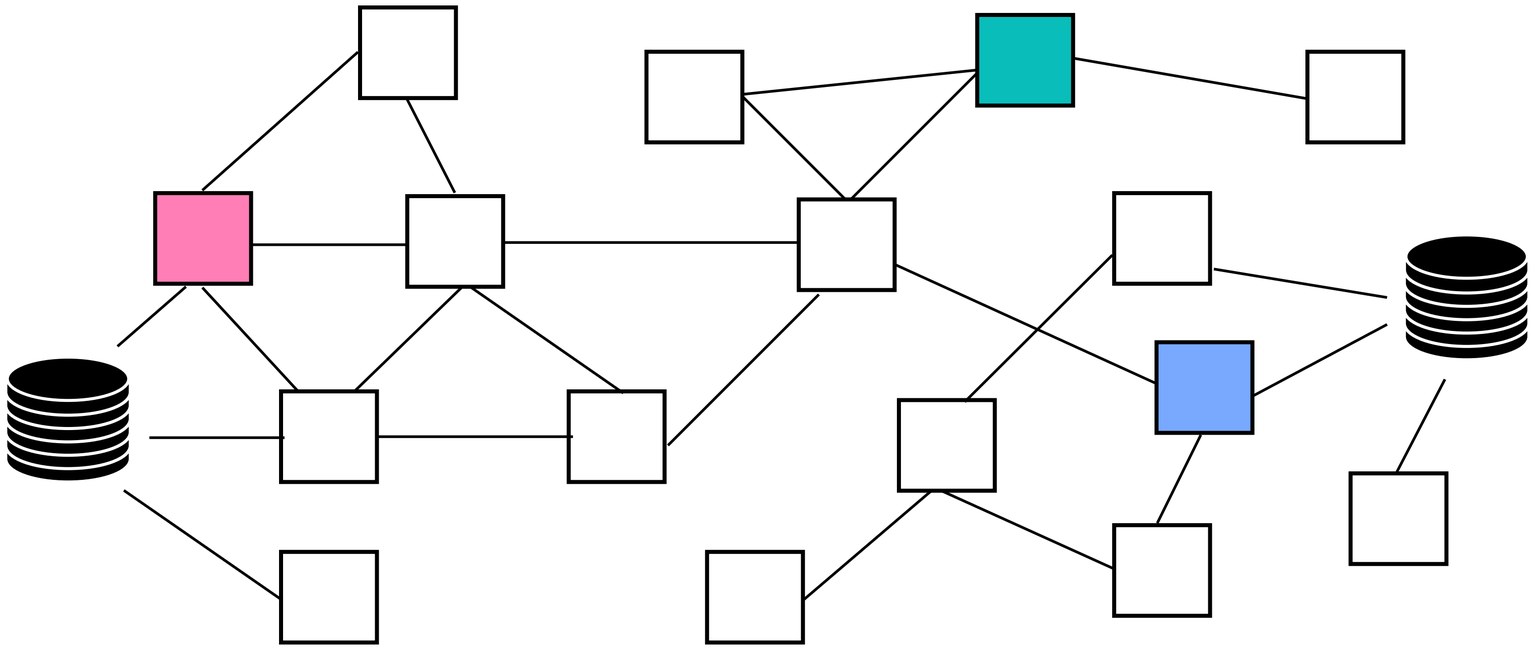


Label Propagation on
Transactional Snapshot

Label Propagation on
Context Snapshots

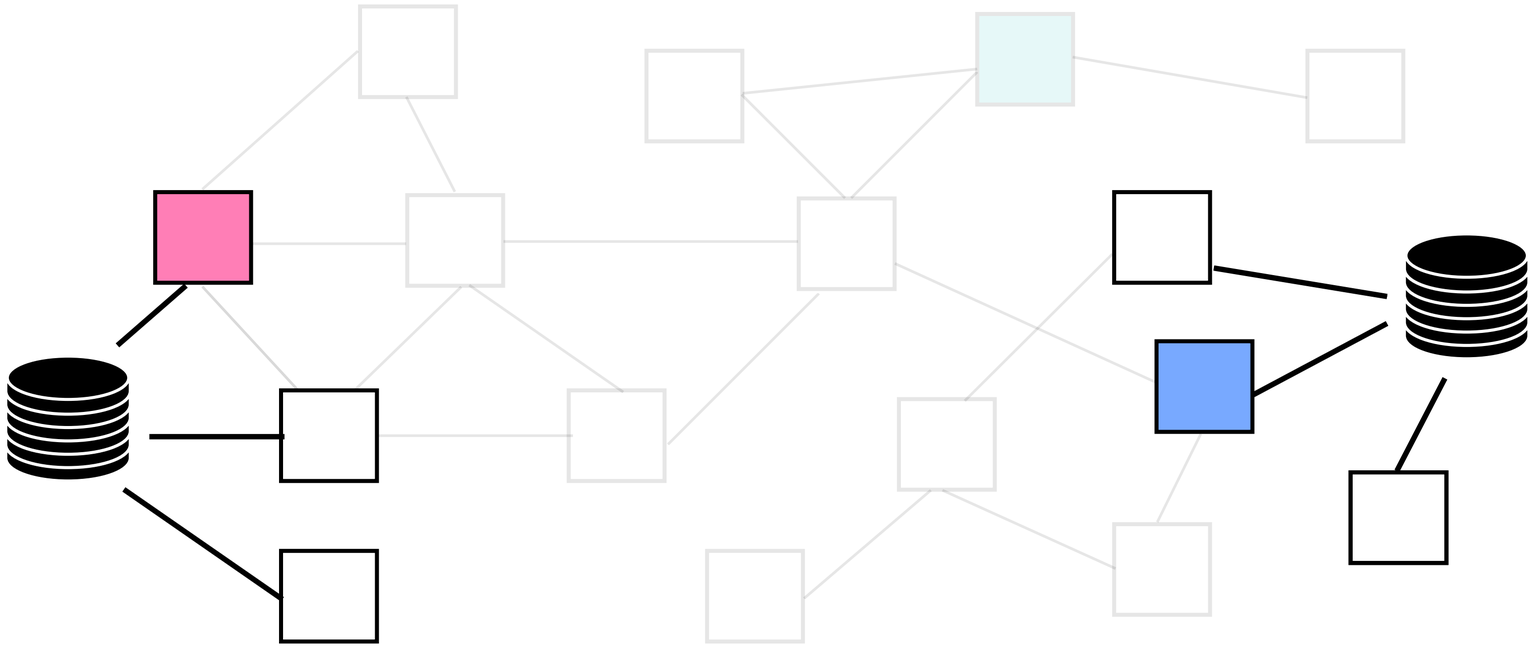
Context-sensitive Label Propagation

Initialize labels



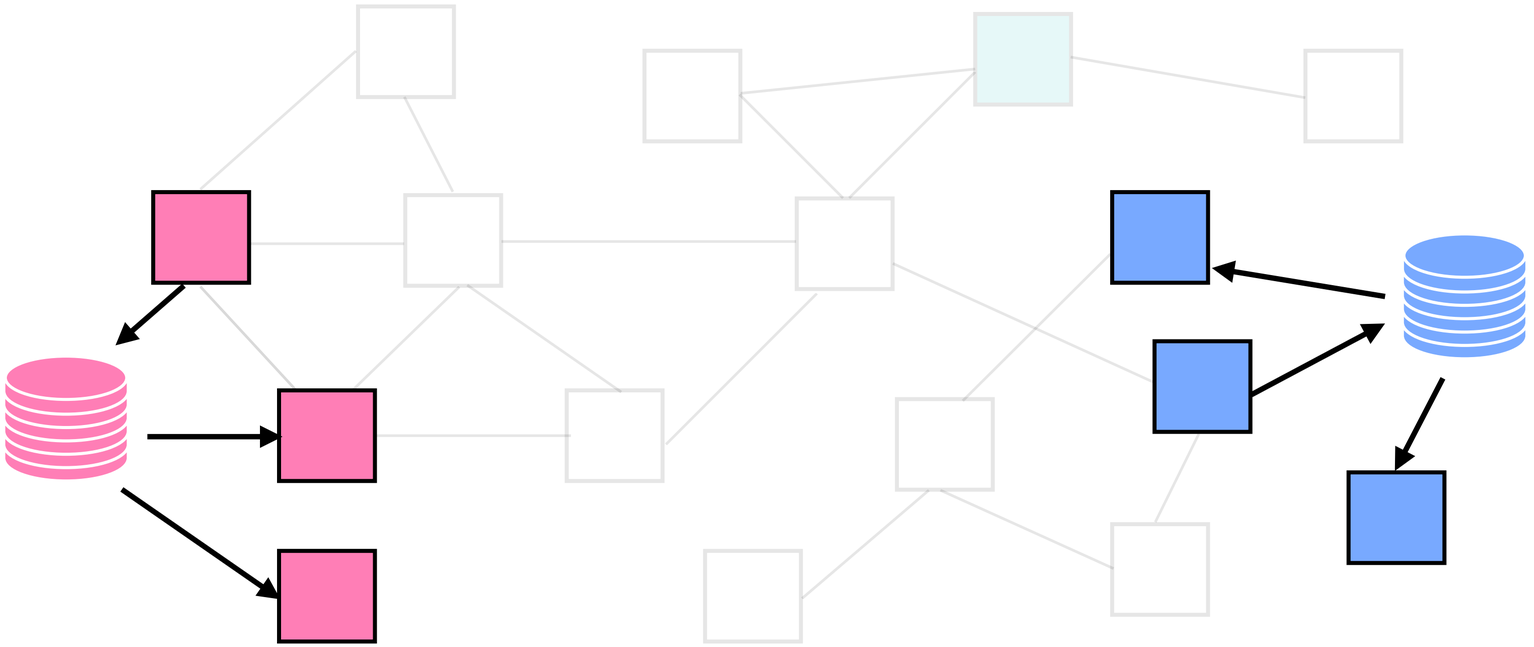
Context-sensitive Label Propagation

Transactional Snapshot



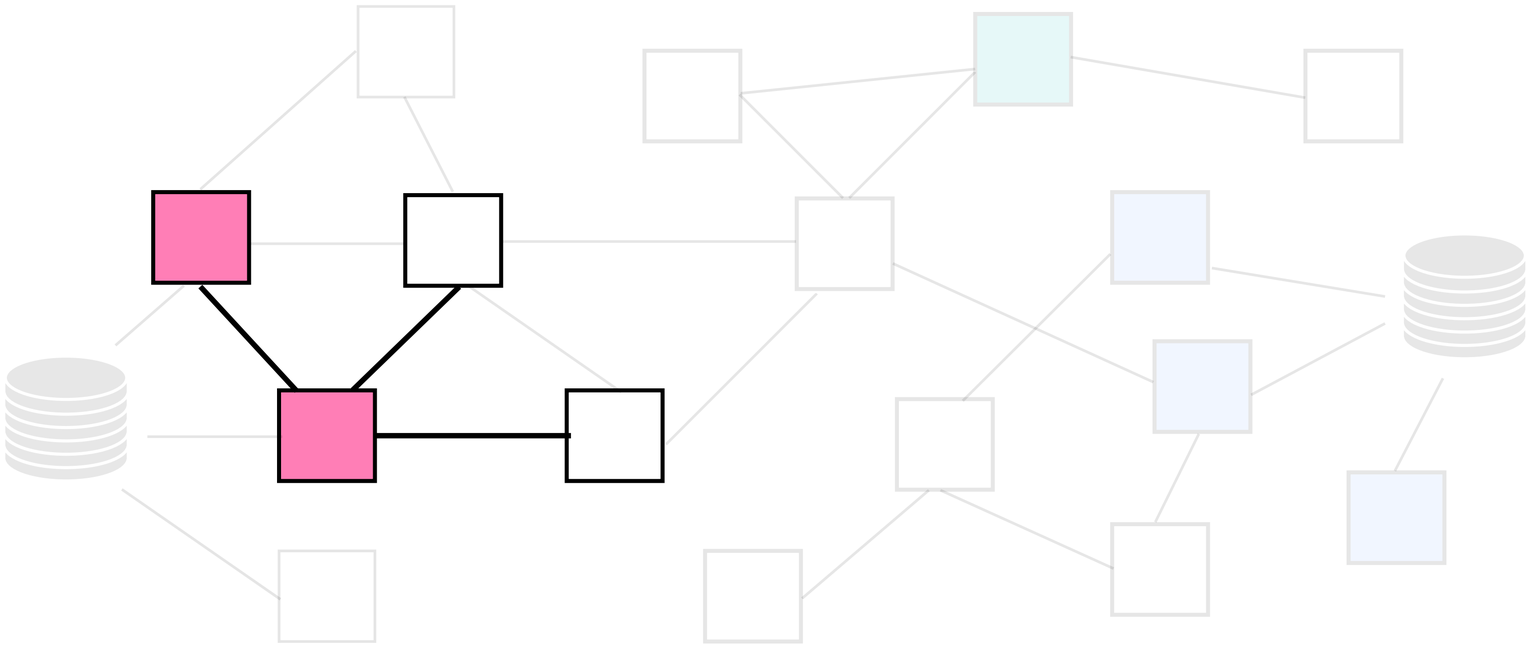
Context-sensitive Label Propagation

Propagate Labels



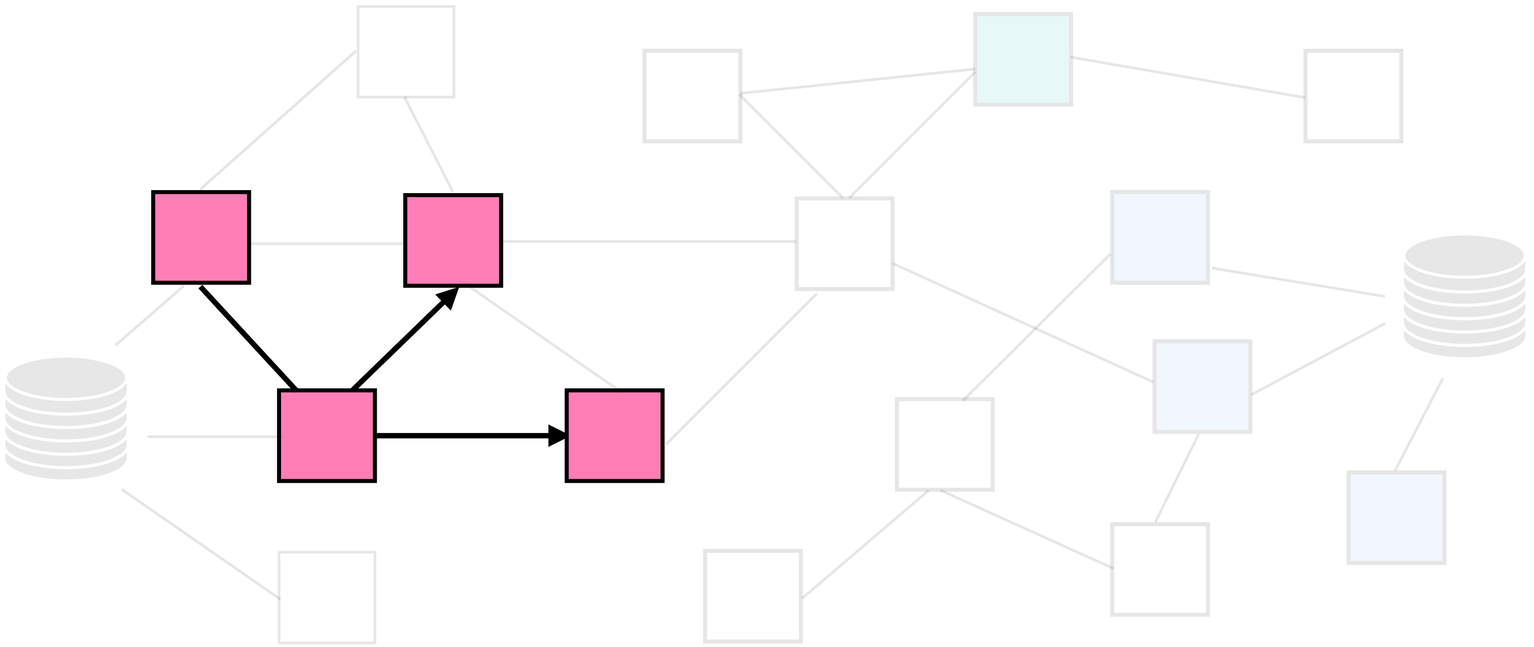
Context-sensitive Label Propagation

Context Snapshot 1



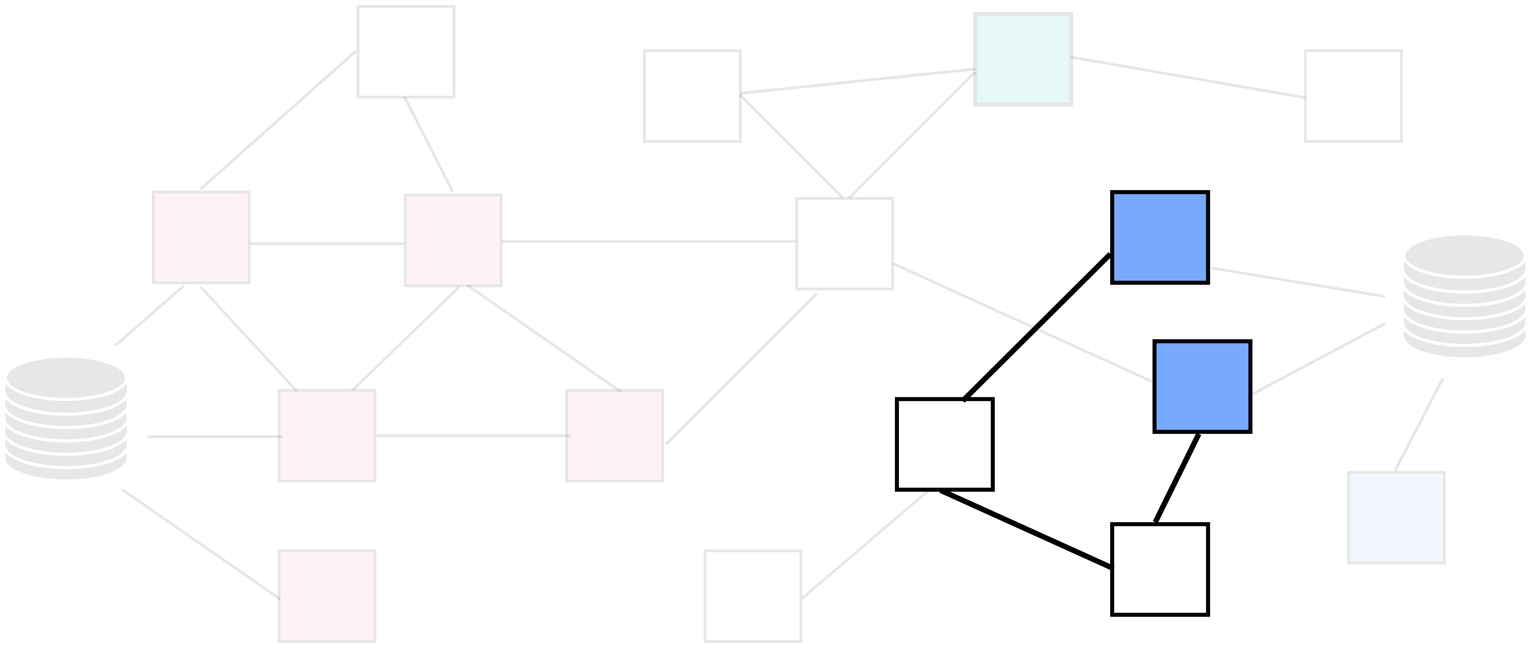
Context-sensitive Label Propagation

Propagate Labels



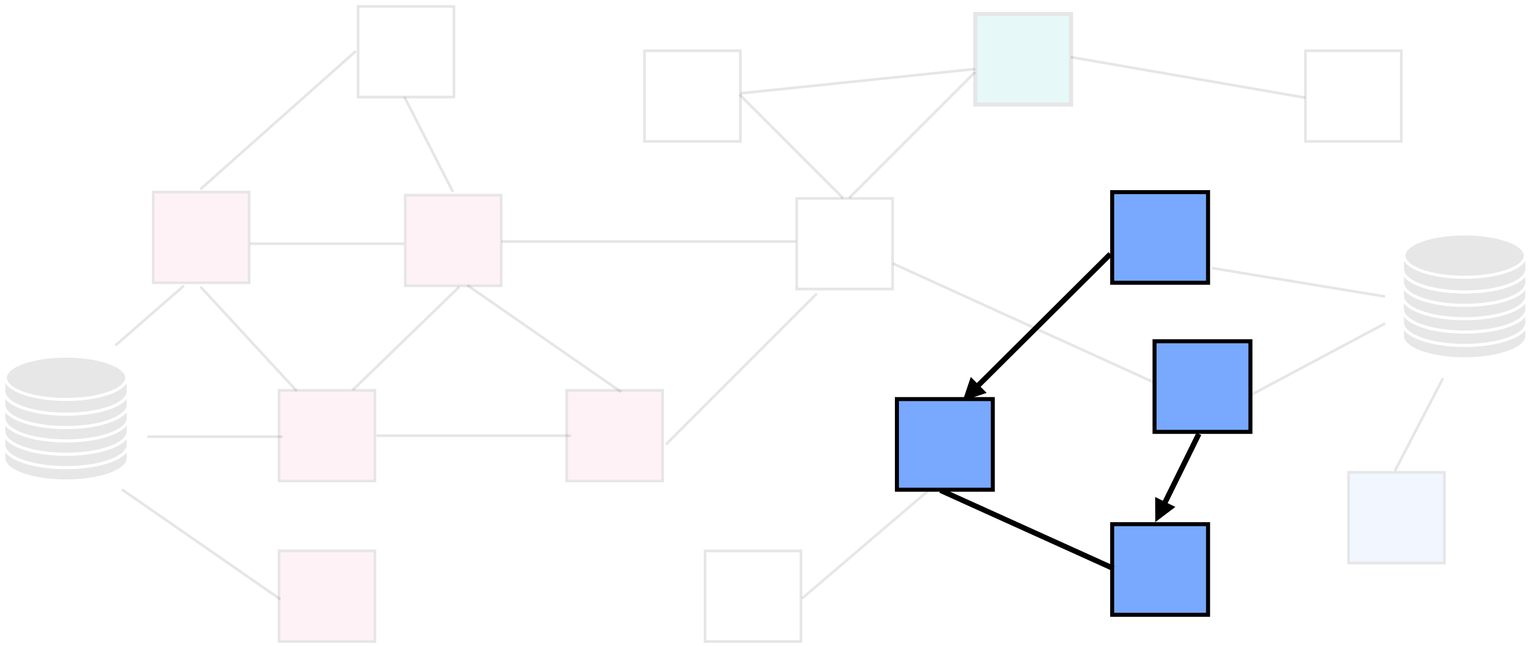
Context-sensitive Label Propagation

Context Snapshot 2



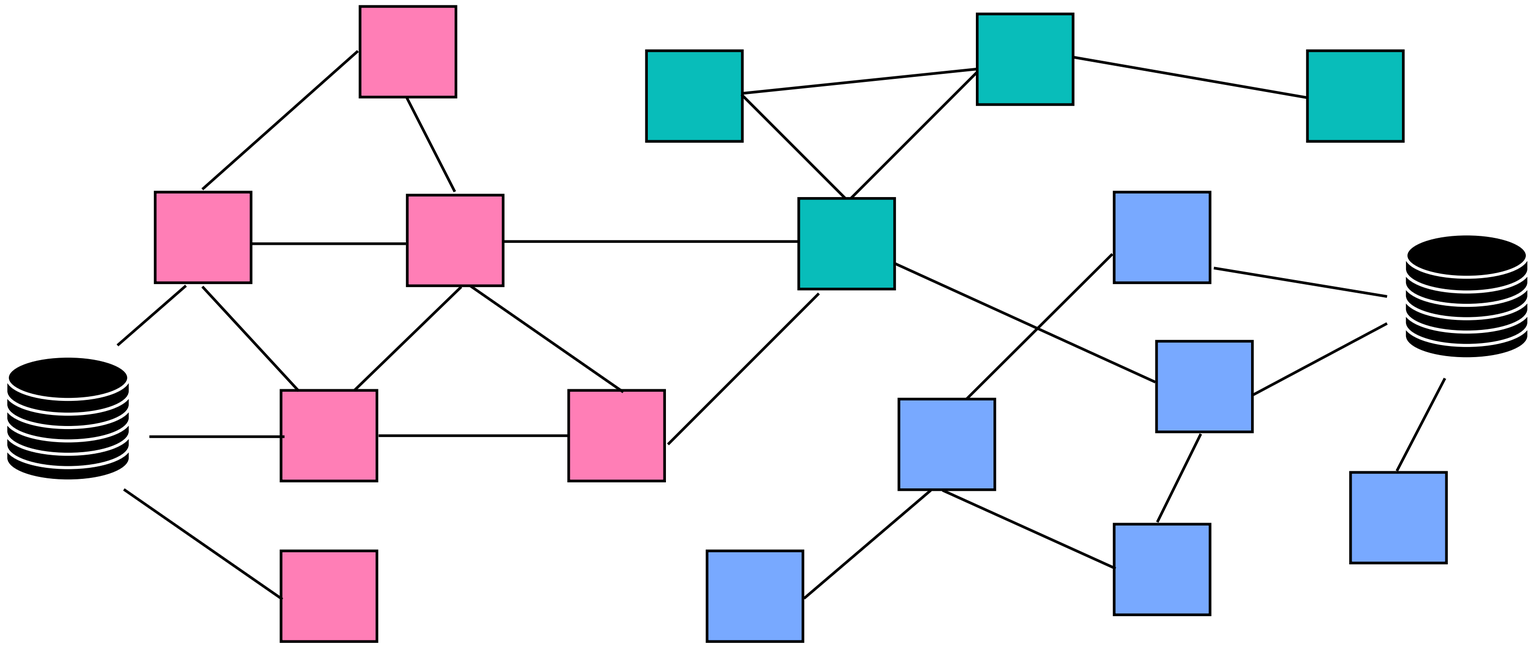
Context-sensitive Label Propagation

Propagate Labels



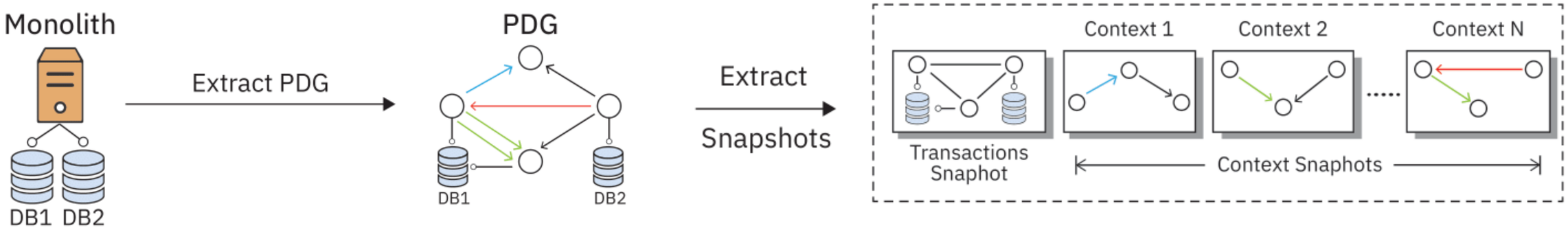
Context-sensitive Label Propagation

And so on for all context snapshots

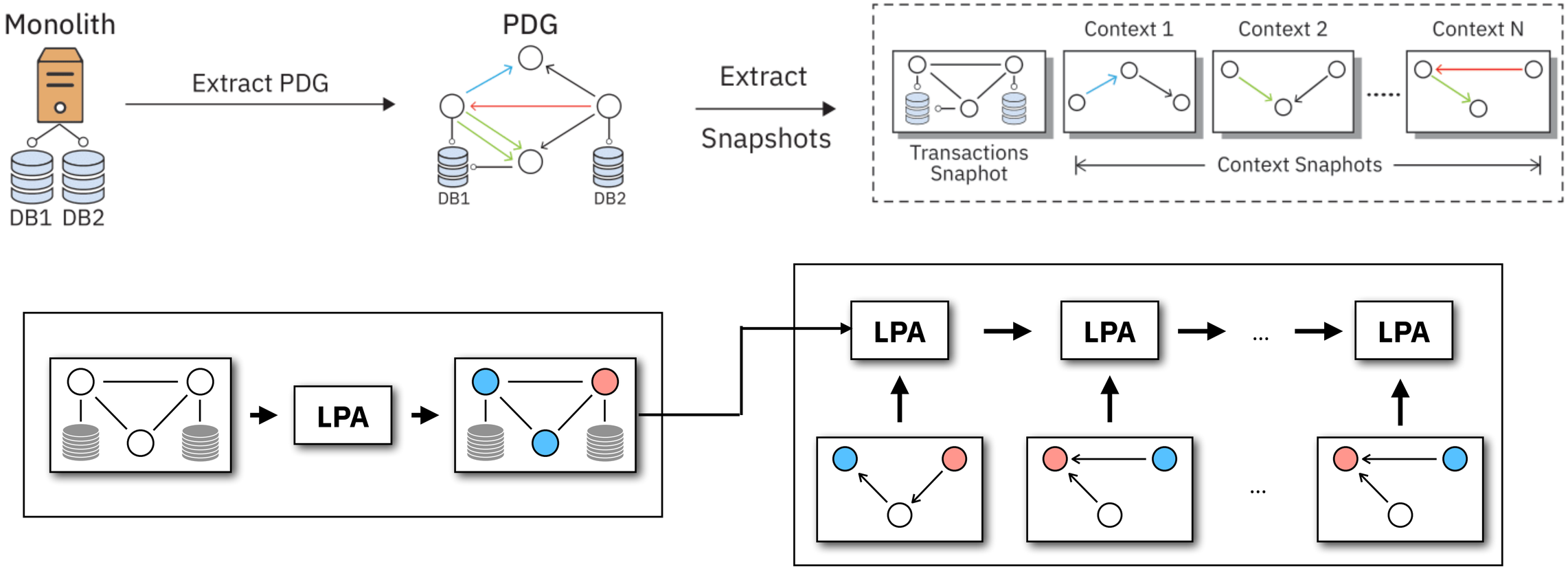


Our System - Overview

Our System - Overview



Our System - Overview



Evaluation Setup - Benchmark Applications

Application	Description	Java Framework	# Classes	# SQL
Daytrader	Trading application	Java EE 8, Websphere	109	6
Plants	Online plant shopping	Java EE 7, Websphere	33	.
AcmeAir	Website of a fictitious airline	Openliberty, Websphere eXtreme	66	.
JPetStore	Online pet supply store	Spring, Springboot	37	.
Proprietary1	Proprietary app	.	82	.

Evaluation Setup - Benchmark Applications

Application	Description	Java Framework	# Classes	# SQL
Daytrader	Trading application	Java EE 8, Websphere	109	6
Plants	Online plant shopping	Java EE 7, Websphere	33	.
AcmeAir	Website of a fictitious airline	Openliberty, Websphere eXtreme	66	.
JPetStore	Online pet supply store	Spring, Springboot	37	.
Proprietary1	Proprietary app	.	82	.

Evaluation Setup - Benchmark Applications

Application	Description	Java Framework	# Classes	# SQL
Daytrader	Trading application	Java EE 8, Websphere	109	6
Plants	Online plant shopping	Java EE 7, Websphere	33	.
AcmeAir	Website of a fictitious airline	Openliberty, Websphere eXtreme	66	.
JPetStore	Online pet supply store	Spring, Springboot	37	.
Proprietary1	Proprietary app	.	82	.

Evaluation Setup - Benchmark Applications

Application	Description	Java Framework	# Classes	# SQL
Daytrader	Trading application	Java EE 8, Websphere	109	6
Plants	Online plant shopping	Java EE 7, Websphere	33	.
AcmeAir	Website of a fictitious airline	Openliberty, Websphere eXtreme	66	.
JPetStore	Online pet supply store	Spring, Springboot	37	.
Proprietary1	Proprietary app	.	82	.

Evaluation Setup - Baseline Approaches

Approach	Summary
Mono2Micro*	Dynamic call traces and hierarchical clustering.
CoGCN	A Graph Neural Network and K-Means on a static call graph
FoSCI	Genetic Search-based algorithm on dynamic execution traces
MEM	A Minimum-Spanning Tree based Clustering Algorithm on a graph. Edit-history and semantics are used to define coupling

** Enterprise scale decomposition tool*

Evaluation Setup - “refining” partitions

CARGO can be used to **refine** the partitions produced by other approaches.

- Denoted by **“++”** suffix.

*E.g., **Mono2Micro++** denotes running CARGO with initial partition labels produced by Mono2Micro.*

Original Approach	Refined with CARGO
Mono2Micro	**Mono2Micro++**
CoGCN	**CoGCN++**
FoSCI	**FoSCI++**
MEM	**MEM++**

Research Questions

RQ-1 Effectiveness in remediating distributed transactions

RQ-2 Latency and Throughput improvements resulting from refined microservice partitions

RQ-3 Quality of microservice partition architectural metrics

Research Questions

RQ-1 Effectiveness in remediating distributed transactions

RQ-2 Latency and Throughput improvements resulting from refined microservice partitions

RQ-3 Quality of microservice partition architectural metrics

RQ-1 Distributed database transactions

To minimize distributed transactions, we would like, *to the extent possible*, for each database table to be accessed from one microservice partition only

RQ-1 Distributed database transactions

To minimize distributed transactions, we would like, *to the extent possible*, for each database table to be accessed from one microservice partition only

Evaluation

- Use Transactional Purity (TXP) to measure the tendency of a database table to be accessed by *multiple* microservices

RQ-1 Distributed database transactions

To minimize distributed transactions, we would like, *to the extent possible*, for each database table to be accessed from one microservice partition only

Evaluation

- Use Transactional Purity (TXP) to measure the tendency of a database table to be accessed by *multiple* microservices

$$TXP = 1 - \sum_{i=0}^{K} p_i \cdot log \left[\frac{1}{p_i} \right]$$

RQ-1 Distributed database transactions

To minimize distributed transactions, we would like, *to the extent possible*, for each database table to be accessed from one microservice partition only

Evaluation

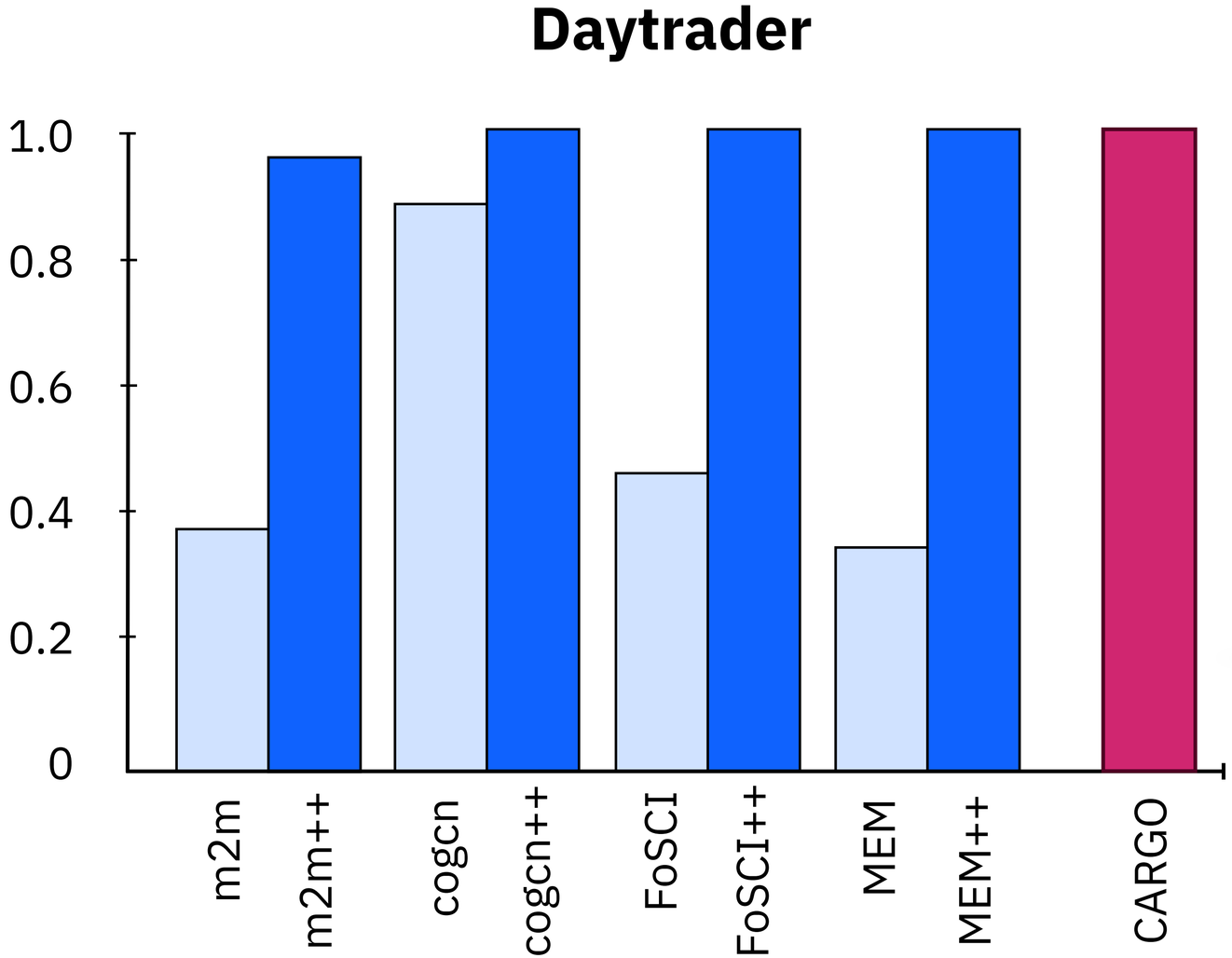
- Use Transactional Purity (TXP) to measure the tendency of a database table to be accessed by *multiple* microservices

Entropy

$$TXP = 1 - \sum_{i=0}^{K} p_i \cdot log \left[\frac{1}{p_i} \right]$$

- Lower purity indicates accesses from more microservices
- Higher purity indicates accesses from less microservices

RQ-1 Distributed database transactions



Summary

- For each of the 4 baselines, the refined partitions have higher transactional purity.
- FoSCI++ and MEM++ have transactional purity of 1.0, (i.e., no distributed transactions after repartitioning).
- CARGO (unsupervised) natively achieves transactional purity of 1.0

***++** implies refinement with CARGO*

Research Questions

RQ-1 Effectiveness in remediating distributed transactions

RQ-2 Latency and Throughput improvements resulting from refined microservice partitions

RQ-3 Quality of microservice partition architectural metrics

RQ-2 Runtime Performance Improvements

Minimizing distributed transactions can offer significant runtime benefits in terms of reduced latency and improved throughput.

Evaluation

- Deploy **two variants** of the applications:
  1. Application with original partitioning (Mono2Micro) [1]
  2. Application with partitions refined with CARGO (aka. Mono2Micro++) [2]

[1] https://github.com/vikramnitin9/tackle-data-gravity-insights/tree/main/RQ2/daytrader_apps/daytrader_cargo
[2] https://github.com/vikramnitin9/tackle-data-gravity-insights/tree/main/RQ2/daytrader_apps/daytrader_mono2micro

RQ-2 Runtime Performance Improvements

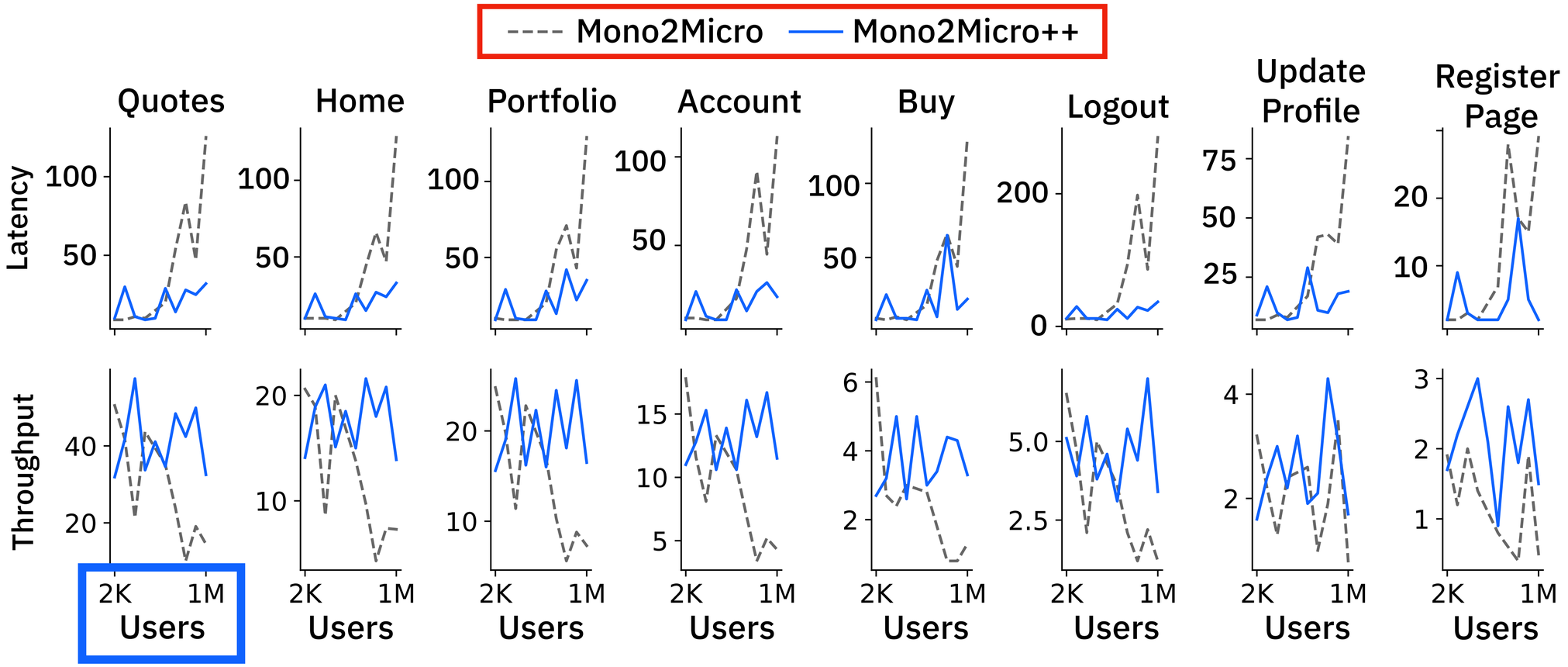
Minimizing distributed transactions can offer significant runtime benefits in terms of reduced latency and improved throughput.

Evaluation

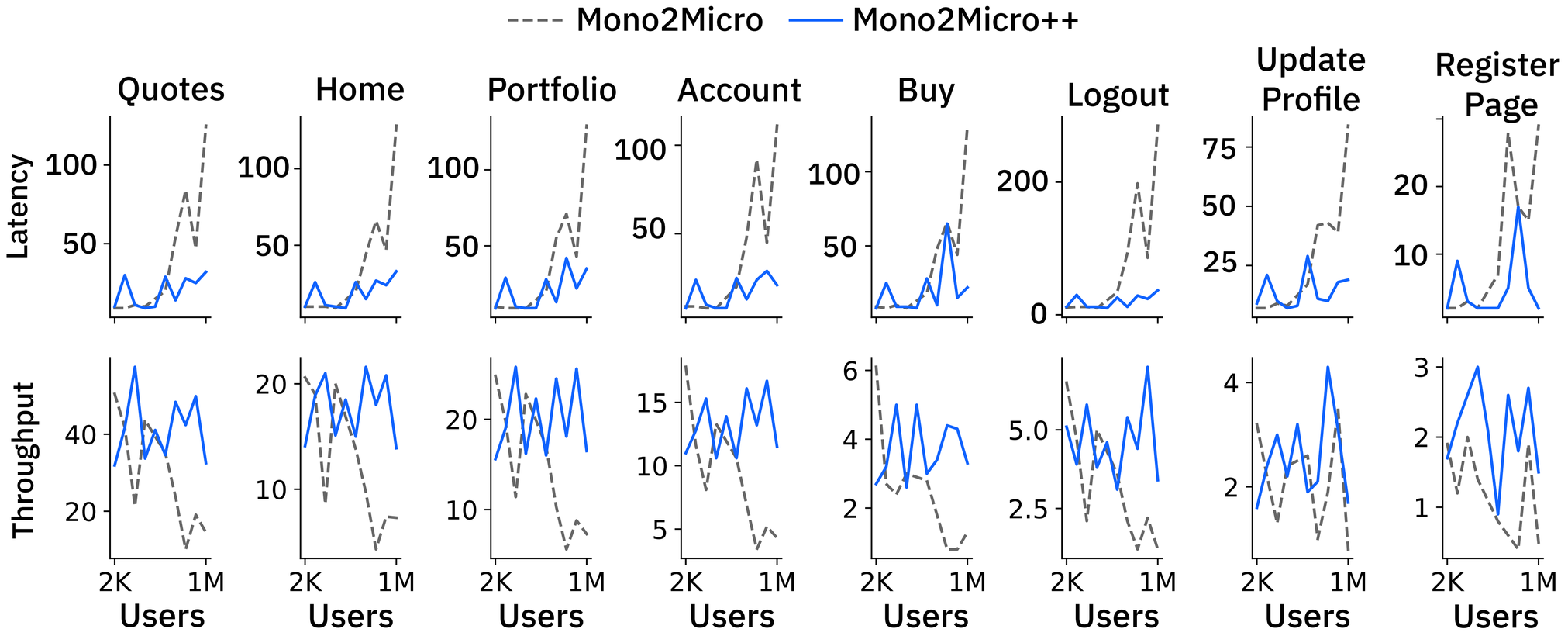
- Deploy **two variants** of the applications:
  1. Application with original partitioning (Mono2Micro) [1]
  2. Application with partitions refined with CARGO (aka. Mono2Micro++) [2]
- Compare two **runtime performance** metrics:
  - **Latency:** Time between reception and completion of a request (milliseconds)
  - **Throughput:** Number of successful requests honored per unit time (requests/second)

[1] https://github.com/vikramnitin9/tackle-data-gravity-insights/tree/main/RQ2/daytrader_apps/daytrader_cargo
[2] https://github.com/vikramnitin9/tackle-data-gravity-insights/tree/main/RQ2/daytrader_apps/daytrader_mono2micro

RQ-2 Runtime Performance Improvements



RQ-2 Runtime Performance Improvements



Significant improvements in latency and throughput. Repartitioned application has **11% lower latency** and **120% higher throughput** on average across use cases compare to the original applications.

Research Questions

RQ-1 Effectiveness in remediating distributed transactions

RQ-2 Latency and Throughput improvements resulting from refined microservice partitions

RQ-3 Quality of microservice partition architectural metrics

RQ-3 Partitions and their Architectural Quality

METRIC		DESCRIPTION
Coupling	▽	Average Coupling among partitions
Cohesion	△	Average cohesion within a partition
BCP	▽	Purity of Business use cases per partition.
ICP	▽	Inter-partition call volume

▽ Lower is better △ Higher is better

RQ-3 Partitions and their Architectural Quality

Coupling ▽

	Mono2Micro	Mono2Micro++	CARGO
Daytrader	0.78	0.02	0.01
Plants	0.31	0.04	0.05
AcmeAir	0.58	0.04	0.03
JPetstore	0.77	0.03	0.03
Proprietary	0.42	0.03	0.04
Win/Tie/Loss	5/0/0		

Cohesion △

	CoGCN	CoGCN++	CARGO
Daytrader	0.37	0.61	0.71
Plants	0.39	0.46	0.6*
AcmeAir	0.21	0.32	0.96
JPetstore	0.20	0.24	0.94
Proprietary	0.69	0.73	0.75*
Win/Tie/Loss	5/0/0		

**Mono2Micro++ performs slightly better than CARGO*

CARGO improves the partitioning quality (reduced coupling and increased cohesion) of other approaches and **works equally well in unsupervised mode.**

RQ-3 Partitions and their Architectural Quality

BCP ▽

	Mono2Micro	Mono2Micro++	CARGO
Daytrader	2.31	2.57	1.31
Plants	1.68	2.20	1.79
AcmeAir	1.29	1.48	1.75
JPetstore	2.25	2.35	2.87
Proprietary	1.53	1.23	1.55
Win/Tie/Loss	0/0/5		

CARGO performs poorly on BCP. The definition of BCP depends heavily on the quality of the generated business use cases, which Mono2Micro has access to but we do not.

Summary

1. Partitioning monolithic code can be challenging

Summary

1. Partitioning monolithic code can be challenging
2. Existing automated approaches miss key code and/or transactional dependencies

Summary

1. Partitioning monolithic code can be challenging
2. Existing automated approaches miss key code and/or transactional dependencies
3. We present CARGO, which uses **(a)** precise static analysis, **(b)** explicit modeling of database transactions, and **(c)** a novel community detection algorithm

Summary

1. Partitioning monolithic code can be challenging
2. Existing automated approaches miss key code and/or transactional dependencies
3. We present CARGO, which uses **(a)** precise static analysis, **(b)** explicit modeling of database transactions, and **(c)** a novel community detection algorithm
4. Compared to existing approaches, CARGO **(a)** reduces distributed transactions, **(b)** achieves better latency and throughput, **(c)** achieves better performance on architectural metrics

Future Directions

- Sometimes, it is not possible to avoid distributed transactions. In that case, we need mechanism to automatically refactor database transactions into distributed patterns like **SAGA**.
- Once we identify partitions, there is still a lot of work to **implement these partitions**.
  - Identify best mechanism for Inter-Process Communication (IPC), e.g., sync vs async
  - Automatically generate IPC code

Thank You!

https://arxiv.org/abs/2207.11784