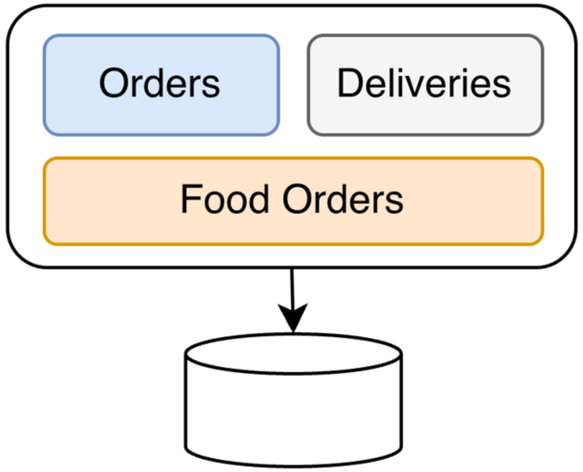
Aletheia: Automated Detection of Data Integrity Violations in Microservices

Mafalda Sofia Ferreira, João Ferreira Loff, João Garcia, Rodrigo Rodrigues
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

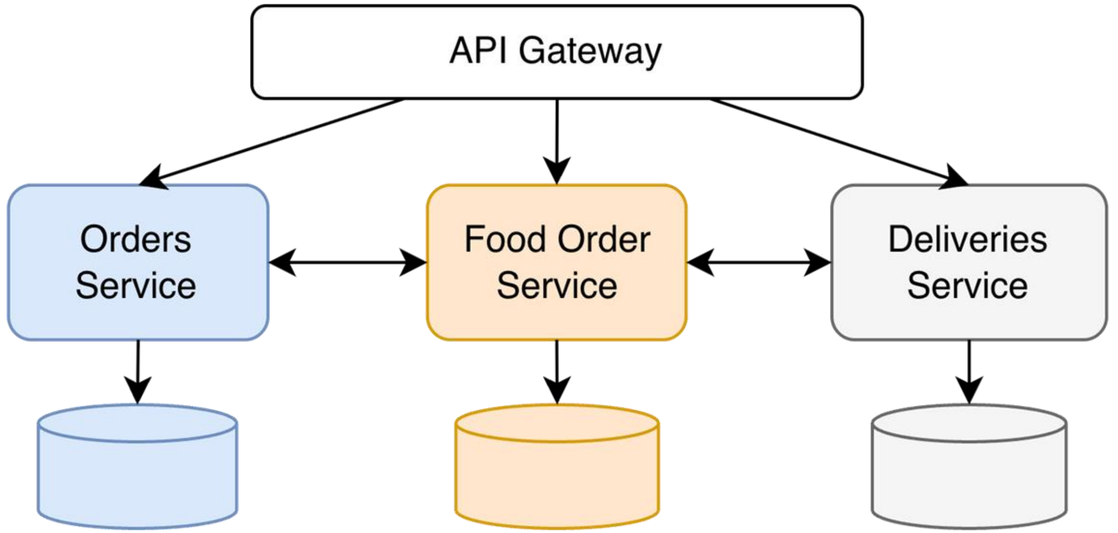
20th USENIX Symposium on Operating Systems Design and Implementation (OSDI '26)

From Monolithic Systems to Microservices

Monolithic Architecture



Microservice Architecture



From Monolithic Systems to Microservices

Out of 1502 respondents:

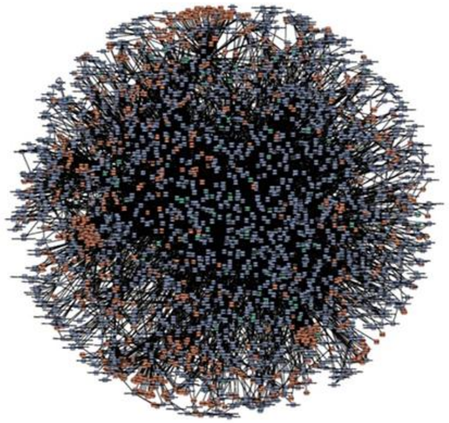
77% have **adopted** microservices

33% from organizations with **< 100 employees**

29% migrating majority of system to microservices

O'Reilly. Microservices Adoption in 2020. (2020, July 15) https://www.oreilly.com/radar/microservices-adoption-in-2020/

Microservices Death Stars:



amazon



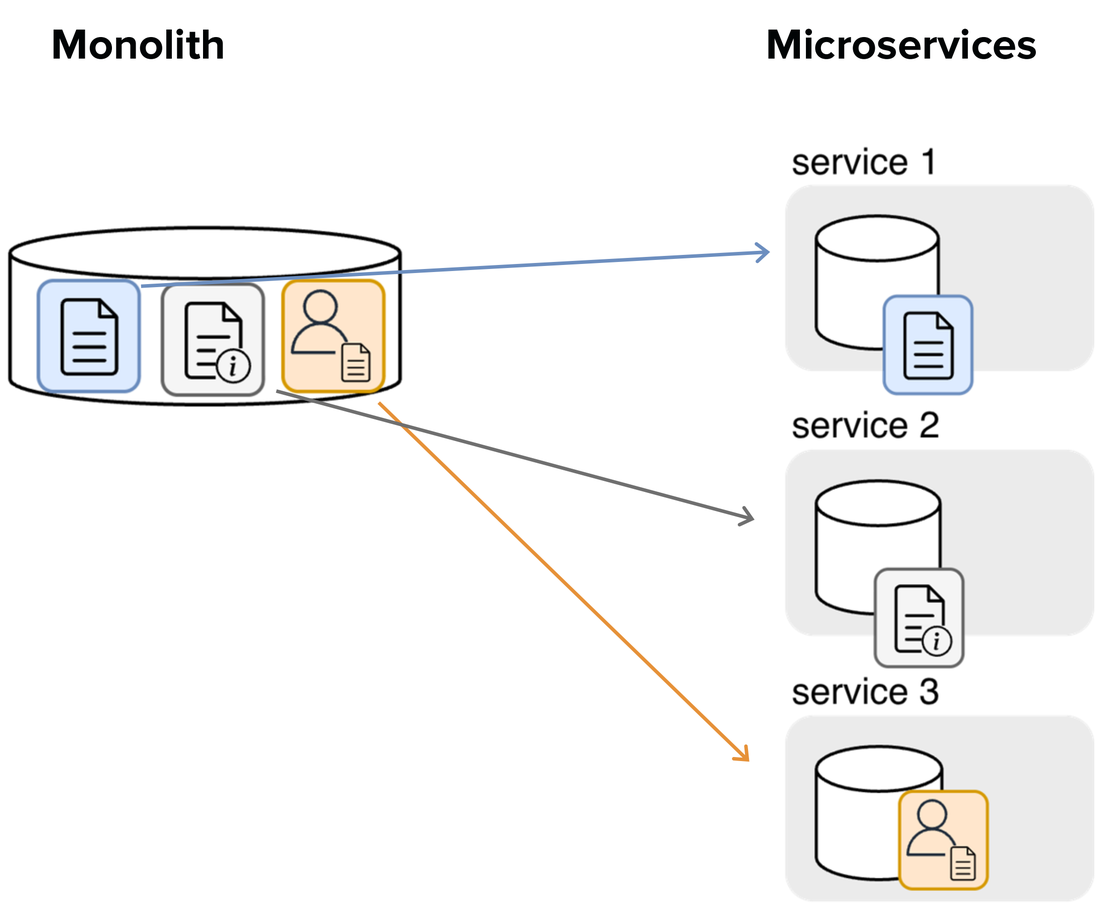
NETFLIX

Adrian Colyer. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. (2019, May 13)

strong transition to microservices

microservices are **complex** and **hard to debug**

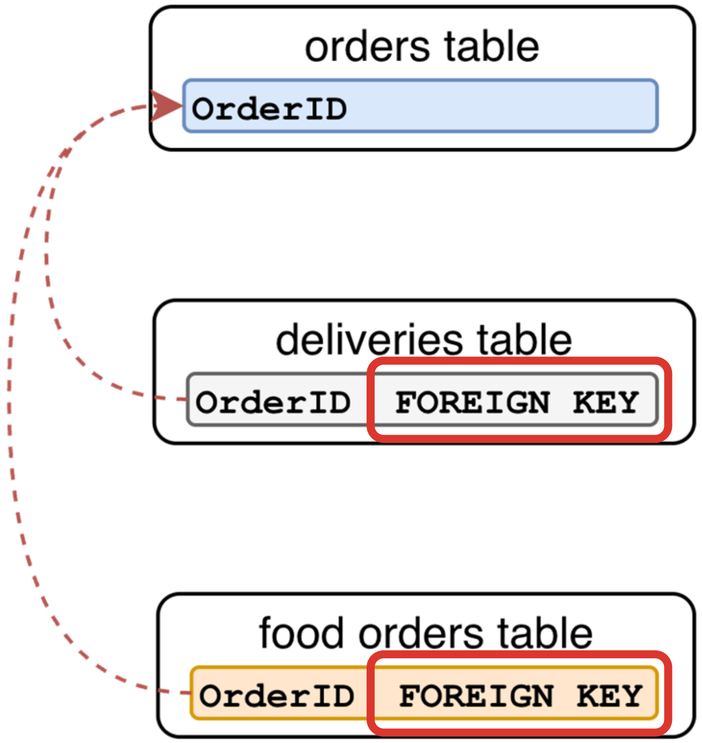
Data Integrity Challenges in Microservices



Decomposition effects on
data integrity?

Data Integrity Challenges in Microservices

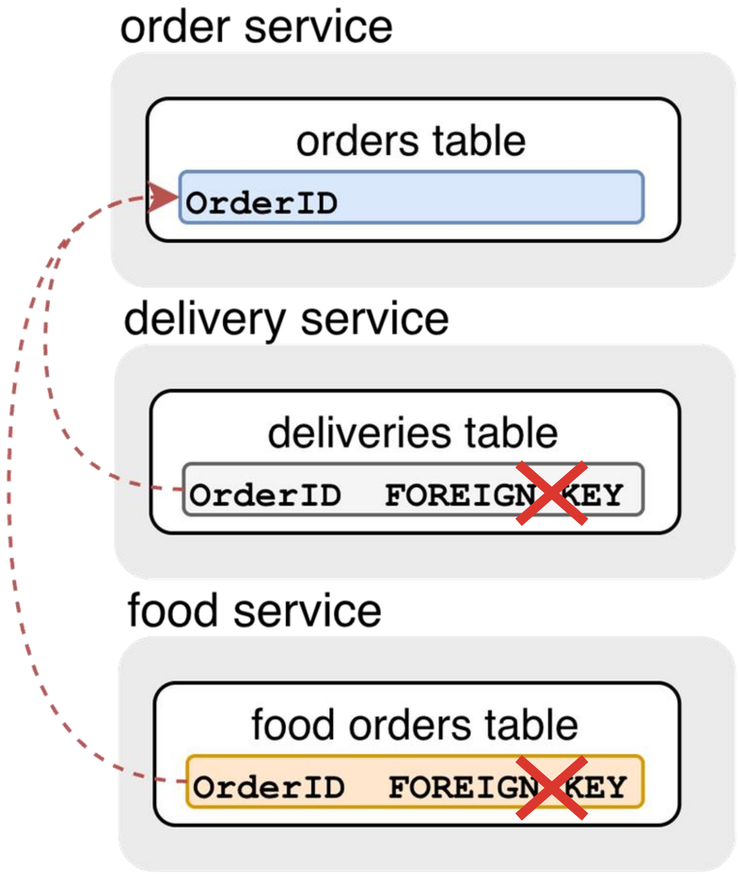
Foreign Keys



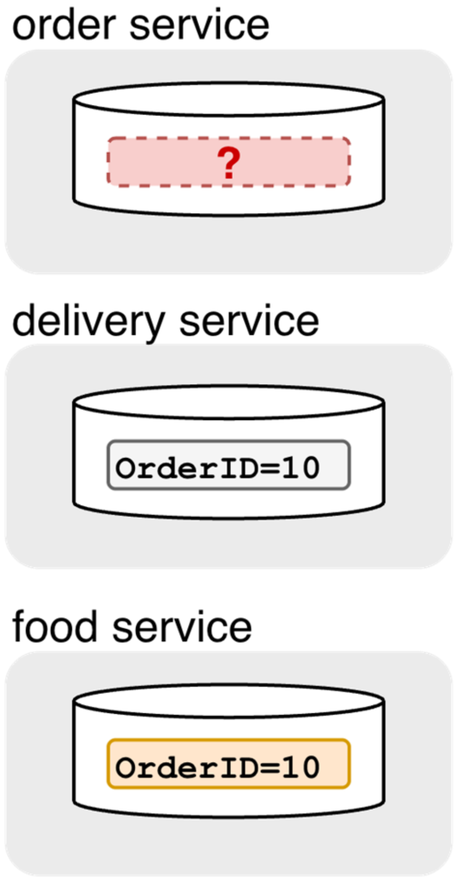
TrainTicket (ICSE ‘18)
Microservice benchmark for reservations of train tickets.

Data Integrity Challenges in Microservices

Foreign Keys



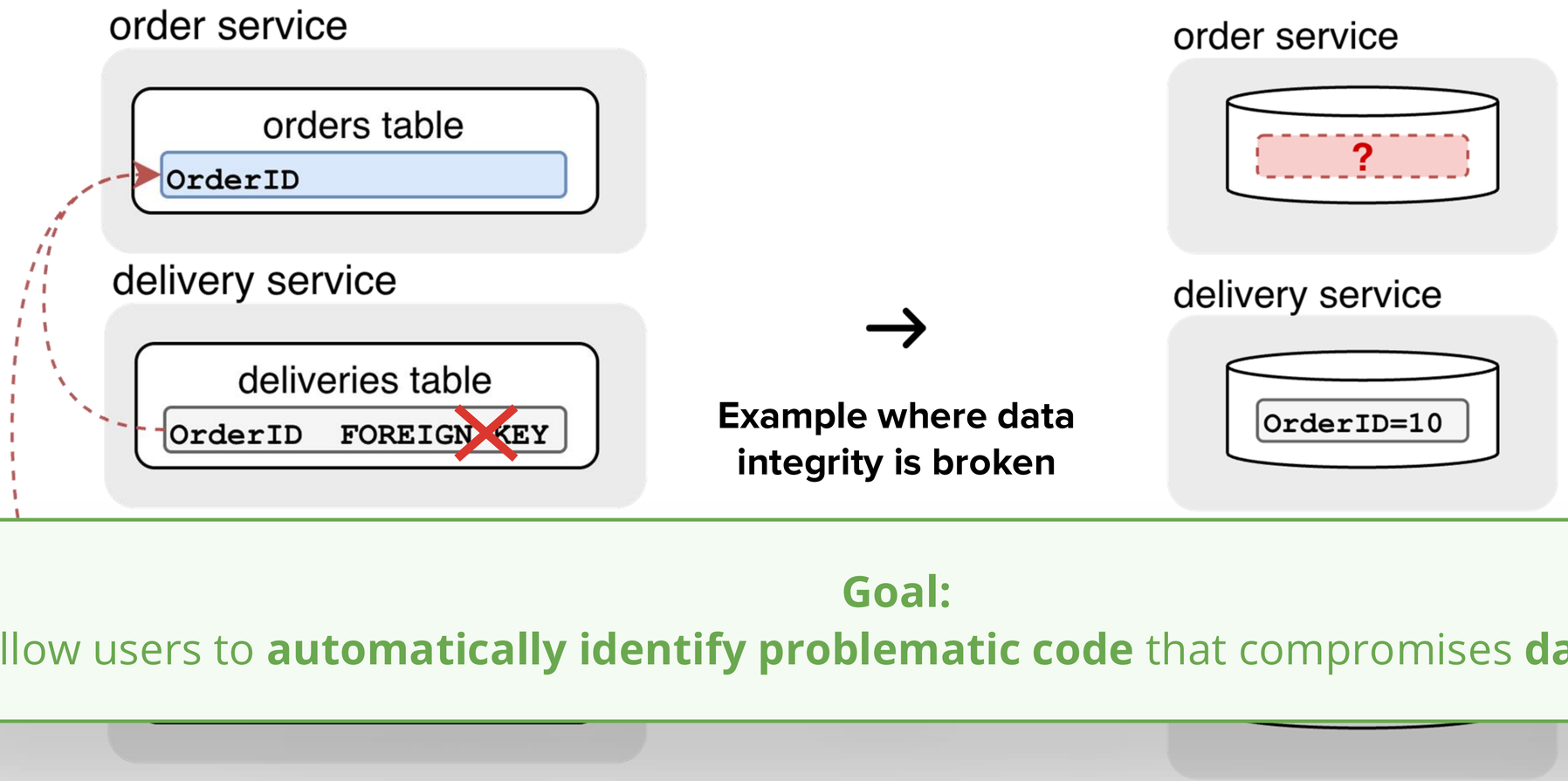
Example where data integrity is broken



TrainTicket (ICSE '18)
Microservice benchmark for reservations of train tickets.

Data Integrity Challenges in Microservices

Foreign Keys



Goal:
Allow users to **automatically identify problematic code** that compromises **data integrity**

TrainTicket (ICSE ‘18)
Microservice benchmark for reservations of train tickets.

Automating Data Integrity Analysis

Question 1

What can trigger **integrity violations** and **how** to **detect** it?

↓

1. Principled methodology based on relational algebra and the entity-relationship (ER) model

Question 2

Can we **automate** the identification process in microservice codebases?

↓

2. Automated detection of data integrity violations through static analysis

Formalized patterns of problematic operations across services

Model for Data Integrity in Microservices

1. Principled methodology based on relational algebra and the ER model

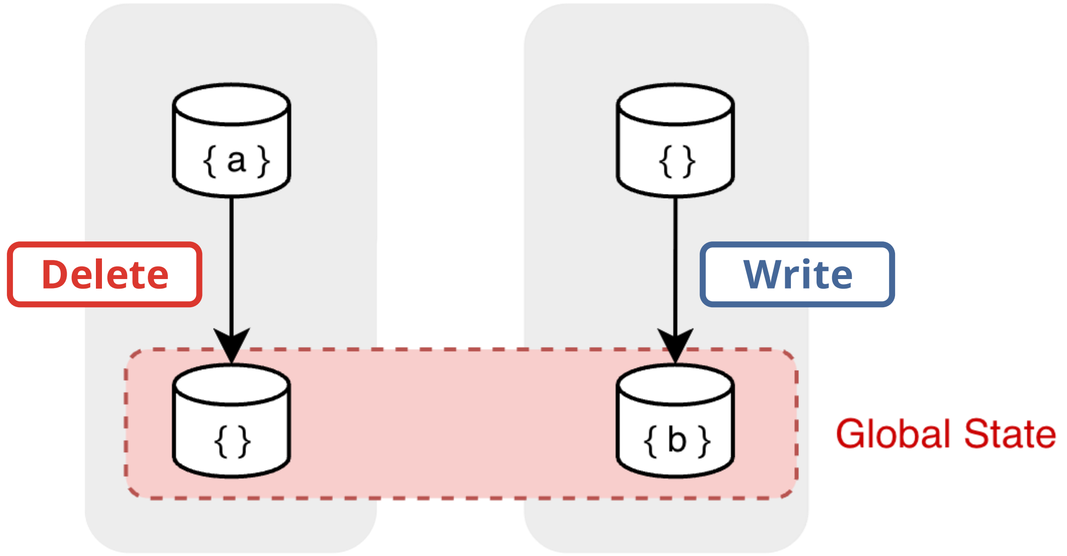
Data integrity constraints

Referential Integrity
(foreign key)

Entity Integrity
(primary key)

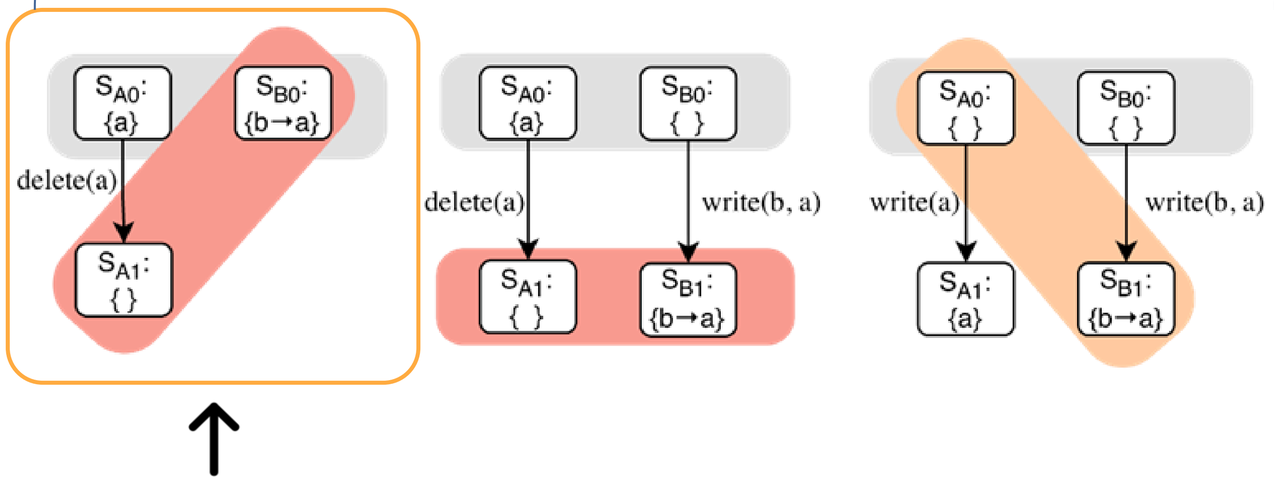
Uniqueness
(unique)

States of databases

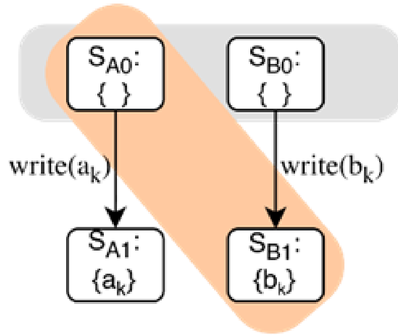


Model for Data Integrity in Microservices

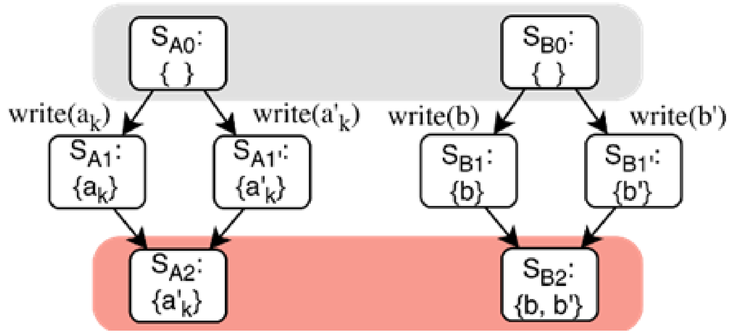
Referential Integrity
(Foreign Key)



Entity Integrity
(Primary Key)



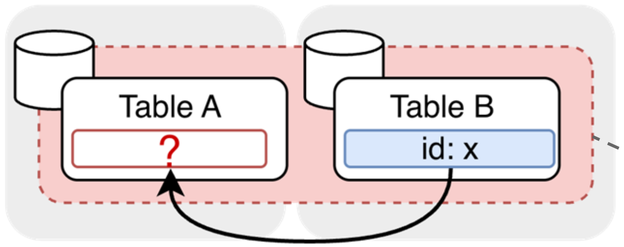
Uniqueness



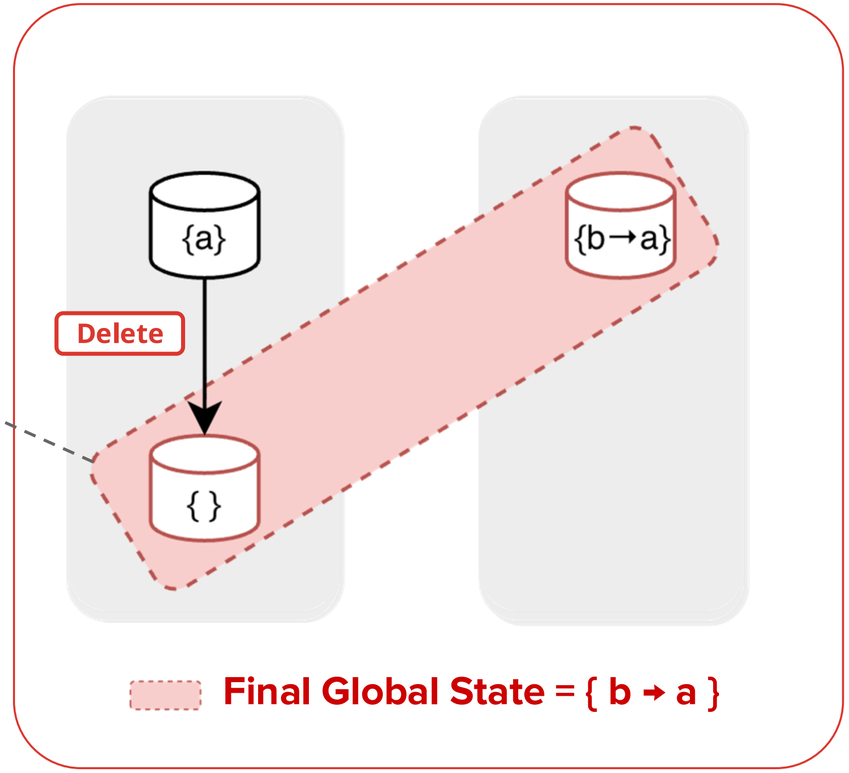
Model for Data Integrity in Microservices

Referential Integrity
Foreign key

Example Schema



cross-service
foreign key

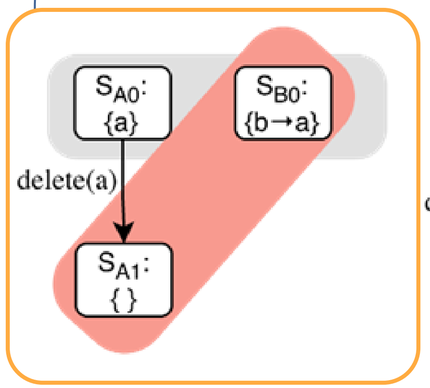


Pattern
(formalized in the paper)

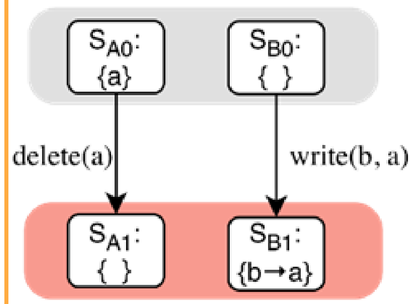
(RI-1)
Absence of Cascading Deletes

Model for Data Integrity in Microservices

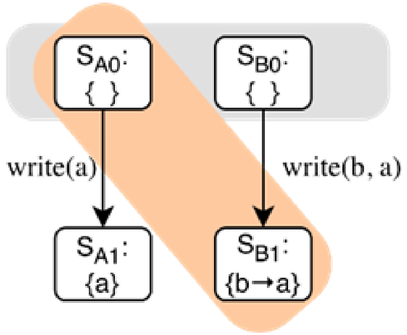
Referential Integrity
(Foreign Key)



(RI-1)
Absence of Cascading Deletes

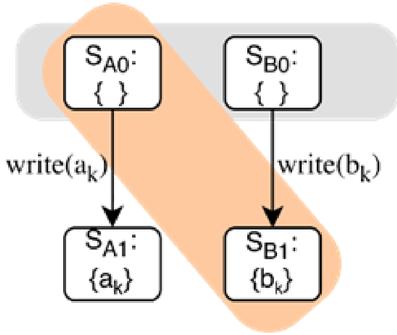


(RI-2)
Concurrent Operations



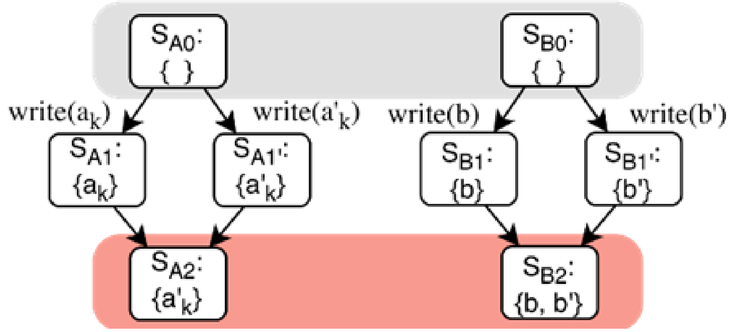
(RI-3)
Uncoordinated Replication

Entity Integrity
(Primary Key)



(EI-1)
Uncoordinated Replication

Uniqueness



(Un-1)
Conflicting Writes

Automating Data Integrity Analysis

Question 1

What can trigger **integrity violations** and **how** to **detect** it?

↓

1. Principled methodology based on relational algebra and the entity-relationship (ER) model

Question 2

Can we **automate** the identification process in microservice codebases?

↓

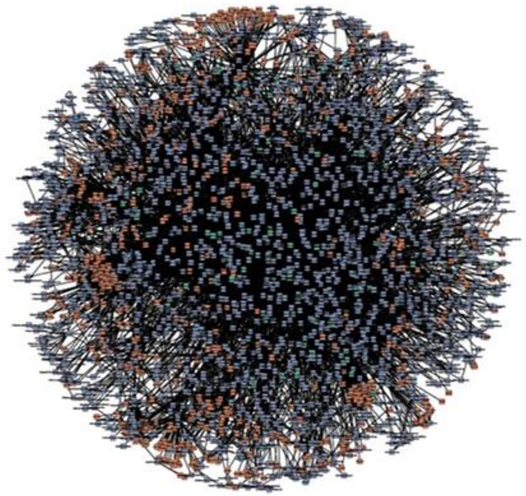
2. Automated detection of data integrity violations through static analysis

Formalized patterns of problematic operations across services

Aletheia

2. Automated detection of data integrity violations through static analysis

Microservices Death Stars:

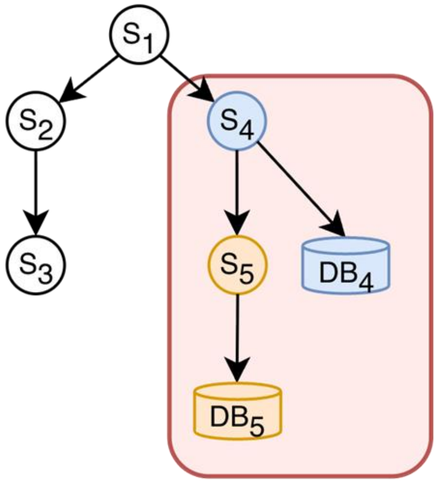


amazon



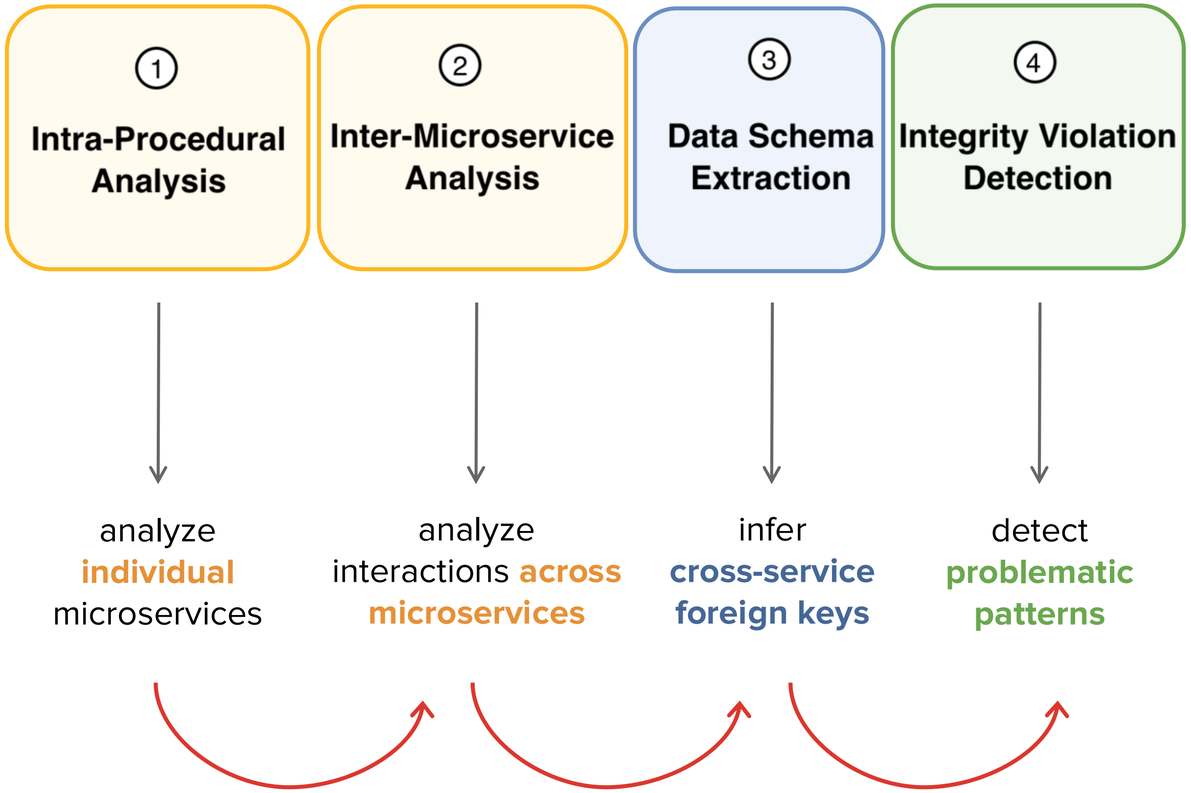
NETFLIX





problematic code pattern

Aletheia


1
Intra-Procedural Analysis
2
Inter-Microservice Analysis
3
Data Schema Extraction
4
Integrity Violation Detection
analyze individual microservices
analyze interactions across microservices
infer cross-service foreign keys
detect problematic patterns

Aletheia




```
func RegisterProduct (id, ...) {
        product_db.write(id, ...)
        inventory_service.AddStock(id, ...)
        ...
}
```

Snippet of Go code of a *product service* method.

Goal:
Determine, within each method, which **incoming** and **outgoing variables** **flow** into **database operations** and **microservice invocations**

Aletheia



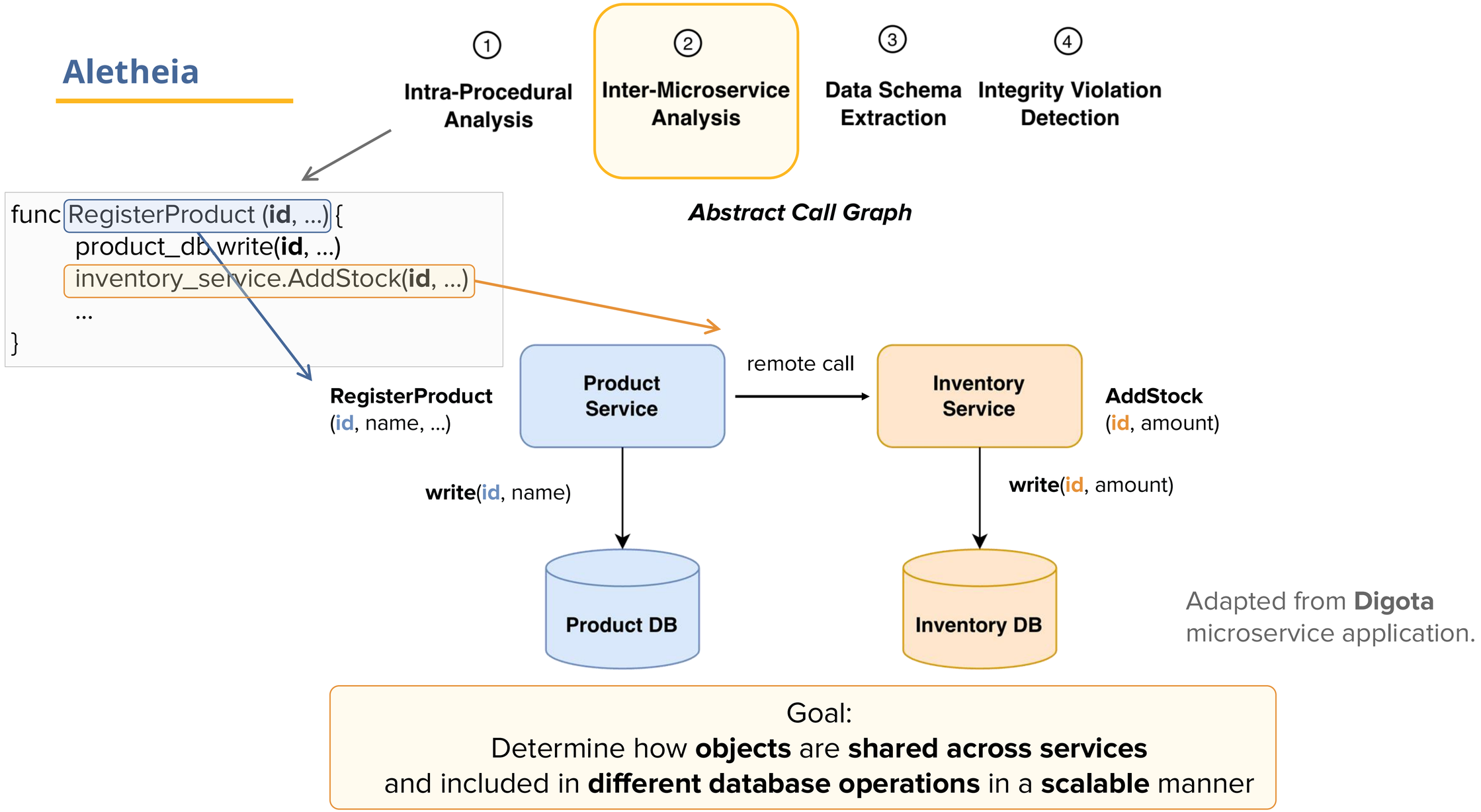

```
func RegisterProduct(id, ...) {
        product_db.write(id, ...)
        inventory_service.AddStock(id, ...)
        ...
}
```

product_db.product.id

Snippet of Go code of a *product service* method.

Goal:
Determine, within each method, which **incoming** and **outgoing variables** **flow** into **database operations** and **microservice invocations**

Aletheia



Adapted from **Digota** microservice application.

Goal:
Determine how **objects** are **shared across services**
and included in **different database operations** in a **scalable** manner

Aletheia

① Intra-Procedural Analysis

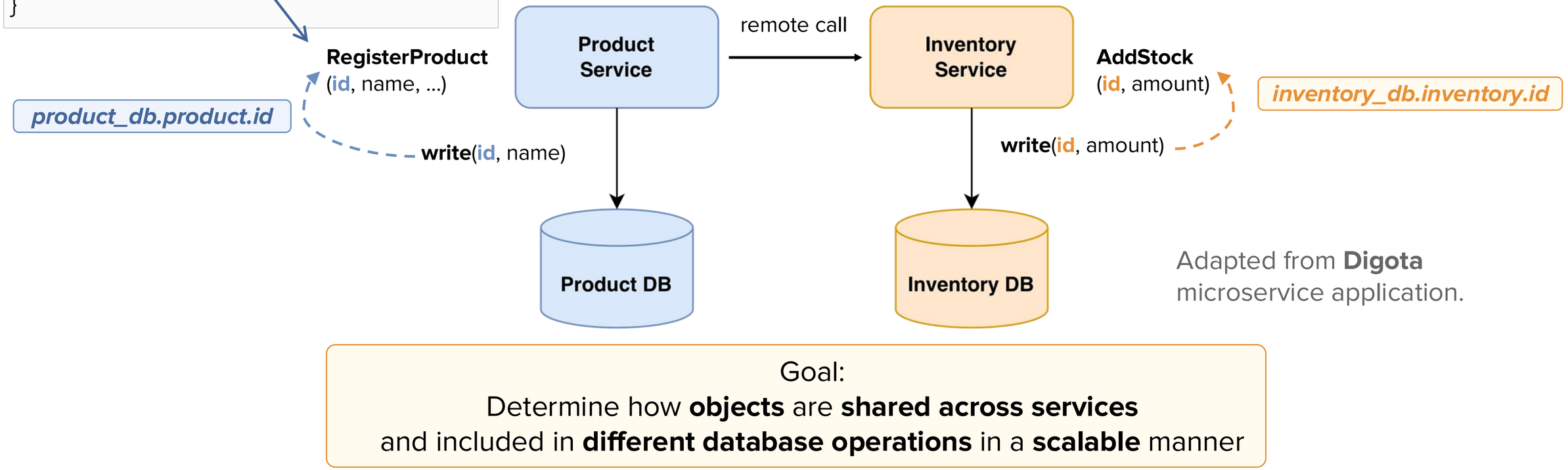
② Inter-Microservice Analysis

③ Data Schema Extraction

④ Integrity Violation Detection

```
func RegisterProduct (id, ...) {
    product_db.write(id, ...)
    inventory_service.AddStock(id, ...)
    ...
}
```

Abstract Call Graph



Adapted from **Digota** microservice application.

Goal:
Determine how **objects** are **shared across services**
and included in **different database operations** in a **scalable** manner

taints from **intra-procedural** analysis

new taints from **inter-microservice** analysis

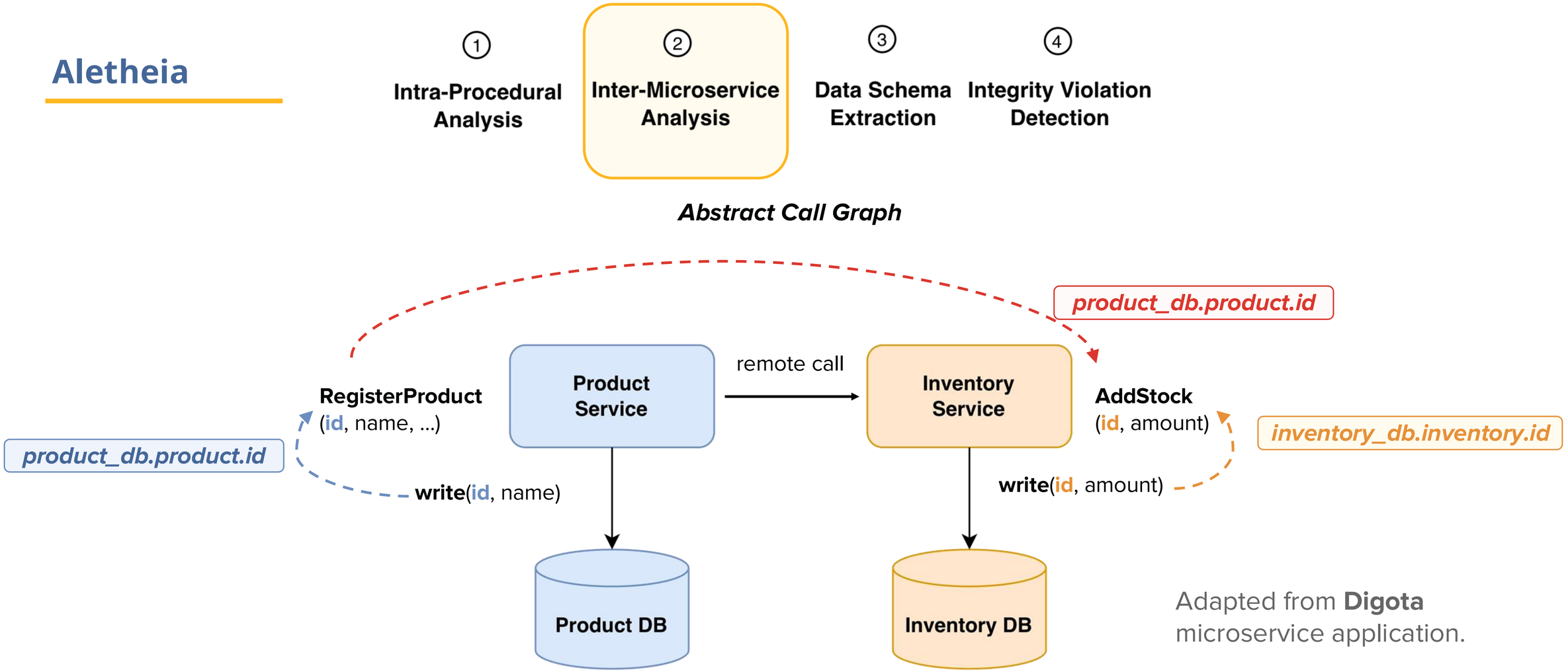
Aletheia

① Intra-Procedural Analysis

② Inter-Microservice Analysis

③ Data Schema Extraction

④ Integrity Violation Detection



Adapted from **Digota** microservice application.

taints from **intra-procedural** analysis

new taints from **inter-microservice** analysis

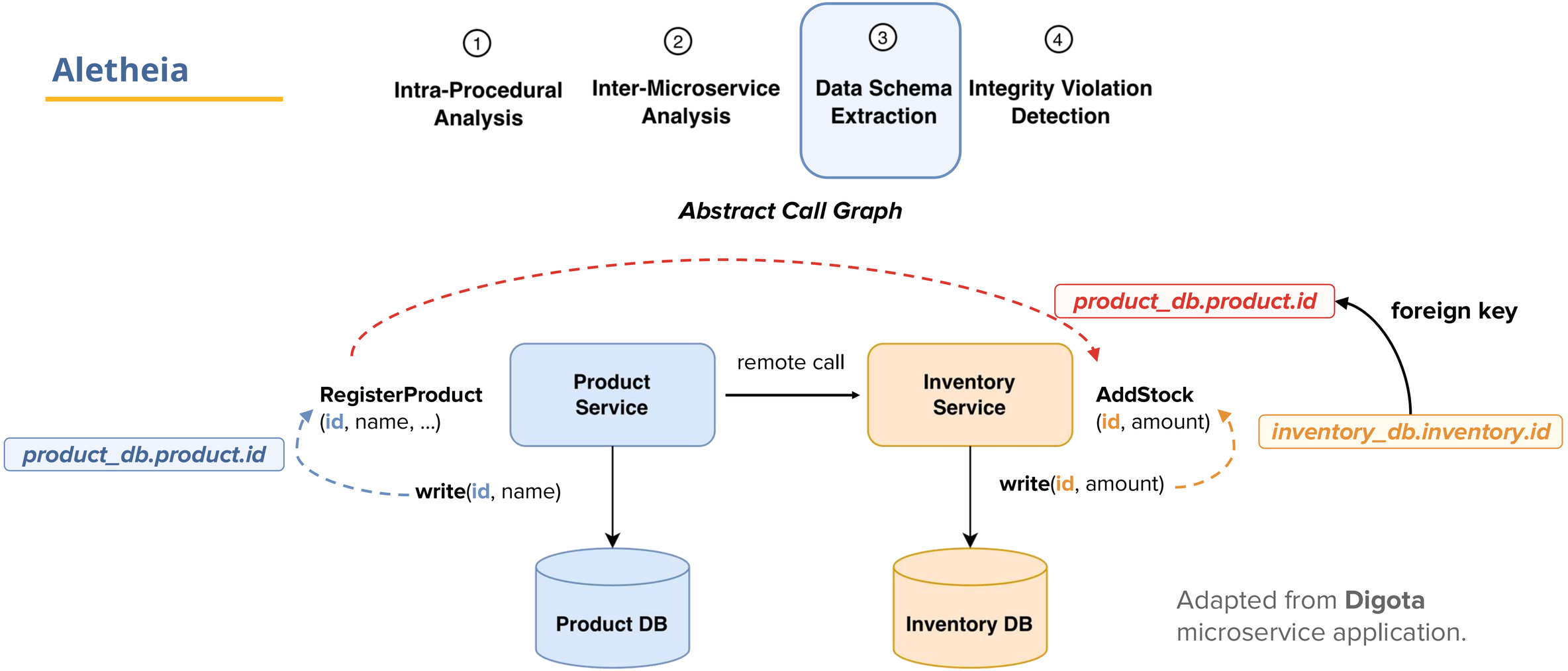
Aletheia

① Intra-Procedural Analysis

② Inter-Microservice Analysis

③ Data Schema Extraction

④ Integrity Violation Detection



Adapted from **Digota** microservice application.

taints from **intra-procedural** analysis

new taints from **inter-microservice** analysis

Aletheia

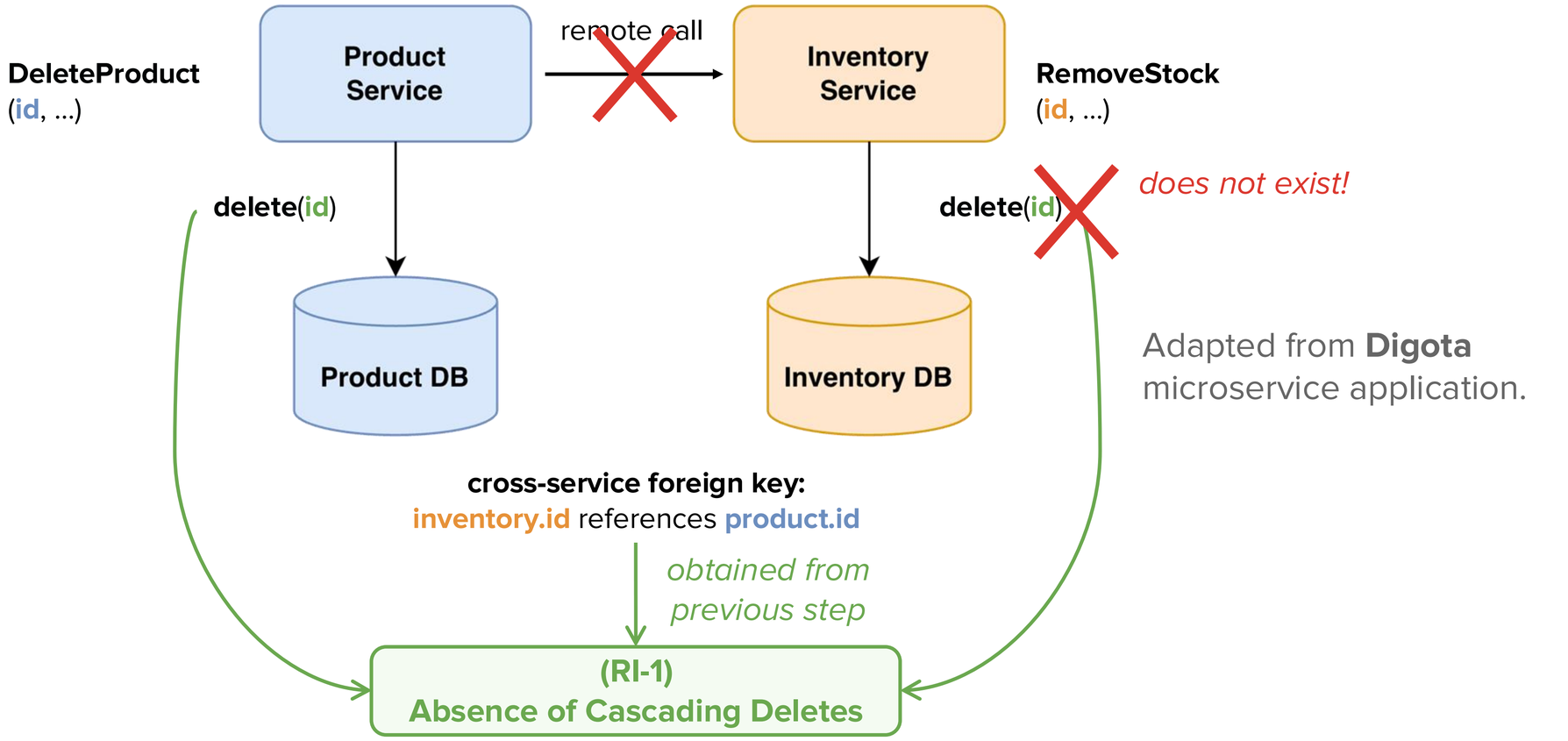
① Intra-Procedural Analysis

② Inter-Microservice Analysis

③ Data Schema Extraction

④ Integrity Violation Detection

Abstract Call Graph



Adapted from **Digota** microservice application.

Evaluation

Question 1

Can Aletheia **detect** integrity violations **effectively**?

Question 2

Which data integrity violations **were found** by Aletheia?

Question 3

How do application **size** and **complexity** impact **analysis time**?

Detailed in the paper

Evaluation

Question 1

Can Aletheia **detect** integrity violations **effectively**?

7 open-source microservice codebases – extended to support **Blueprint** (SOSP '23).

Applications include **DeathStarBench** and **TrainTicket** benchmarks.

Datastores:
- SQL, NoSQL, Caches, Queues

Evaluation | Q1: Can Aletheia detect integrity violations effectively?

	Application Characteristics		Detected Integrity Violations per Pattern					Performance Metrics		
Name	**#MS**	**#RPCs**	**RI-1**	**RI-2**	**RI-3**	**EI**	**Un**	**TP**	**FP**	**FN**
Digota	4	27	2	2	–	–	–	4	–	–
EShopMicroservices	5	33	2	6	–	–	–	8	–	2
Sockshop	8	49	3	3	–	–	–	–	6	–
PostNotification	3	6	–	–	1	–	–	1	–	–
DSB SocialNetwork	13	31	–	–	3	–	–	3	–	–
DSB MediaMicroservices	14	54	–	–	3	1	1	5	–	–
TrainTicket	31	170	14	17	4	–	–	29	6	20

Evaluation | Q1: Can Aletheia detect integrity violations effectively?

Name	#MS	#RPCs	RI (Ref. Integrity)	EI (Ent. Integrity)	Un (Uniqueness)	TP	FP	FN
Digota	4	27	4	–	–	4	–	–
EShopMicroservices	5	33	8	–	–	8	–	2
Sockshop	8	49	6	–	–	–	6	–
PostNotification	3	6	1	–	–	1	–	–
DSB SocialNetwork	13	31	3	–	–	3	–	–
DSB MediaMicroservices	14	54	3	1	1	5	–	–
TrainTicket	31	170	35	–	–	29	6	20

46 previously unreported bugs

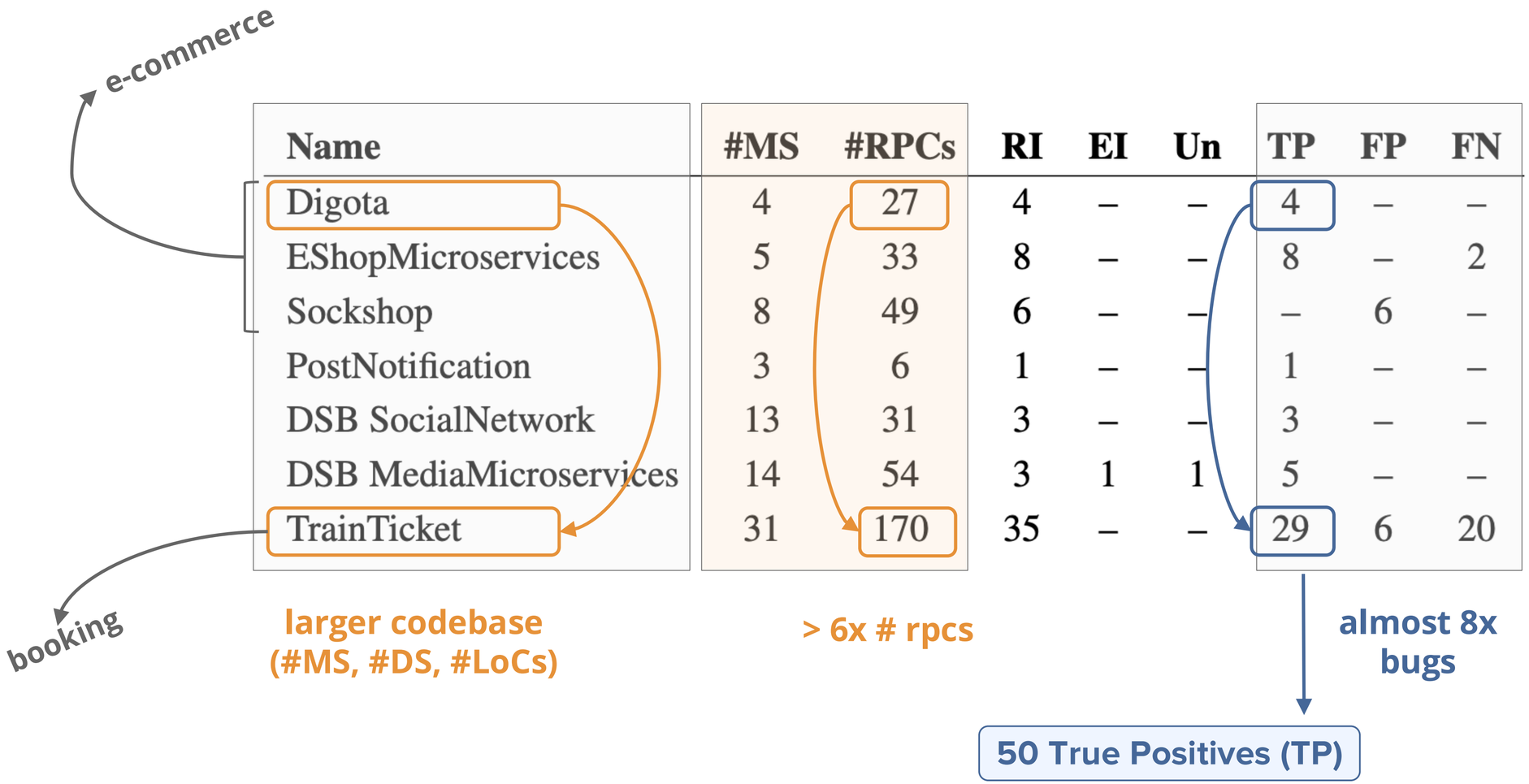
50 True Positives (TP)

12 False Positives (FP)

22 False Negatives (FN)

Precision: 81%
Recall: 69%

Evaluation | Q1: Can Aletheia detect integrity violations effectively?



Name	#MS	#RPCs	RI	EI	Un	TP	FP	FN
Digota	4	27	4	–	–	4	–	–
EShopMicroservices	5	33	8	–	–	8	–	2
Sockshop	8	49	6	–	–	–	6	–
PostNotification	3	6	1	–	–	1	–	–
DSB SocialNetwork	13	31	3	–	–	3	–	–
DSB MediaMicroservices	14	54	3	1	1	5	–	–
TrainTicket	31	170	35	–	–	29	6	20

larger codebase (#MS, #DS, #LoCs)

> 6x # rpcs

almost 8x bugs

50 True Positives (TP)

Evaluation | Q1: Can Aletheia detect integrity violations effectively?

e-commerce

booking

Ref. Integrity | Ent. Integrity | Uniqueness

Name	#MS	#RPCs	RI	EI	Un	TP	FP	FN
Digota	4	27	4	–	–	4	–	–
EShopMicroservices	5	33	8	–	–	8	–	2
Sockshop	8	49	6	–	–	–	6	–
PostNotification	3	6	1	–	–	1	–	–
DSB SocialNetwork	13	31	3	–	–	3	–	–
DSB MediaMicroservices	14	54	3	1	1	5	–	–
TrainTicket	31	170	35	–	–	29	6	20

Conservative approach - cannot determine whether deletions are intentionally not propagated

- 6 FP in SockShop *(e-commerce)*
- 3 FP in TrainTicket *(booking)*

RI-1 pattern
(Absence of Cascading Deletes)

12 False Positives (FP)

Evaluation | Q1: Can Aletheia detect integrity violations effectively?

e-commerce

booking

Name	#MS	#RPCs	RI (Ref. Integrity)	EI (Ent. Integrity)	Un (Uniqueness)	TP	FP	FN
Digota	4	27	4	–	–	4	–	–
EShopMicroservices	5	33	8	–	–	8	–	2
Sockshop	8	49	6	–	–	–	6	–
PostNotification	3	6	1	–	–	1	–	–
DSB SocialNetwork	13	31	3	–	–	3	–	–
DSB MediaMicroservices	14	54	3	1	1	5	–	–
TrainTicket	31	170	35	–	–	29	6	20

Cross-request implicit associations - associations that exist outside of microservice code

- 2 FN in EShopMicroservices *(e-commerce)*
- 12 FN in TrainTicket *(booking)*

RI-1 + RI-2 patterns

22 False Negatives (FN)

Aletheia: Automated Detection of Data Integrity Violations in Microservices

1.
Model for data integrity in microservices

Highlight the **problem of data integrity in microservices**.

Principled approach to reason about data integrity violations.

2.
Aletheia:
Static analysis tool

Automated detection of integrity violations through static analysis.

More information about Aletheia

github.com/aletheia-microservices

3.
Analysis of microservice codebases

Automatically identified **46 unknown integrity anomalies** in existing microservices.

Mafalda Sofia Ferreira, João Ferreira Loff, João Garcia, Rodrigo Rodrigues
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa