EVENTUAL CONSISTENCY

Du musst keine Angst haben… Oder doch?

Fraunhofer
IESE

24.06.2021

Susanne Braun

embarc Midsommar

Pat Helland

Database & Distributed Systems Guru

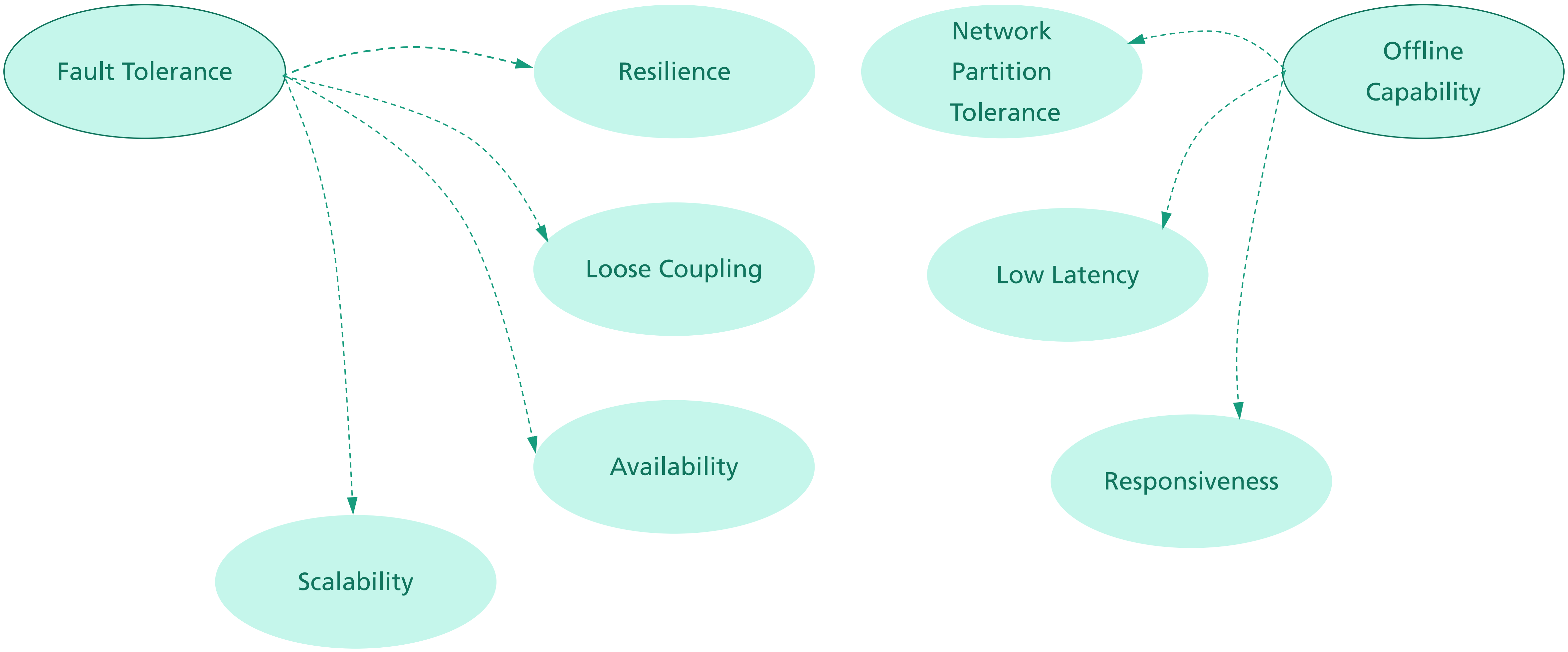
Architect of multiple transaction & database systems (e.g. DynamoDB)

Worked at Microsoft, Amazon, SalesForce, …

"There is an interesting connection between fault tolerance, offlineable systems, and the need for application-based eventual consistency."

Conference on Innovative Data Systems Research, 2009

“Correlating” Quality Attributes



Eric Brewer

Distributed Systems Researcher

Coined the **CAP theorem**, Contributed to Spanner

Prof. emeritus University of California, Berkeley, works now for Google

"But we forfeit **C** and **I** of ACID for availability, graceful degradation and performance."

ACM Symposium on Principles of Distributed Computing, 2000

ACID vs. BASE

This is about **Concurrency Control**!

Atomicity
Consistency
Isolation
Durability

Database is in a consistent state & all invariants are being met!

This is about **Convergence**!

Atomicity
~~**C**onsistency~~
~~**I**solation~~
Durability ?

ACID	BASE
Strong Consistency (in the sense of **one-copy-consistency**)	Eventual Consistency (stale data & approximate answers)
Isolation (in the sense of **one-copy-serializability**)	Availability (top priority)
Pessimistic Synchronization (global locks, synchronous update propagation)	Optimistic Synchronization (no locks, asynchronous update propagation)
Global Commits (2PC, majority consensus, …)	Independent Local Commits (conflict resolution, reconciliation, …)

Strong Consistency vs. Isolation

Make it appear as one system!

"**Strong Consistency** tries to mask the distributed nature of the system"

Make it appear I am the only user of the system!

"**Isolation** tries to mask the effects of concurrent execution"

Fraunhofer IESE

Douglas Terry

Distributed Systems Researcher

Coined the term **Eventual Consistency** in the 90ties

Former Prof. University of California, Berkeley, worked for Microsoft, Samsung, AWS

"A system providing eventual consistency guarantees that replicas would eventually **converge** to a mutually consistent state, i.e., to identical contents, if update activity ceased."

Int. Conference on Parallel and Distributed Information Systems, 1994

Douglas Terry

Distributed Systems Researcher

Coined the term **Eventual Consistency** in the 90ties

Former Prof. University of California, Berkeley, worked for Microsoft, Samsung, AWS

Pragmatic Definition

A system provides eventual consistency if:

(1) each update operation **is eventually received** by each replica

(2) **non-commutative** update operations are performed in the same order at each replica

(3) the outcome of a sequence of update operations is the same at each replica (**determinism**)

Replicated Data Management for Mobile Computing, 2008

Eventual Consistency

Remember:

The only guarantee you get:
convergence to identical state

Application needs to handle:

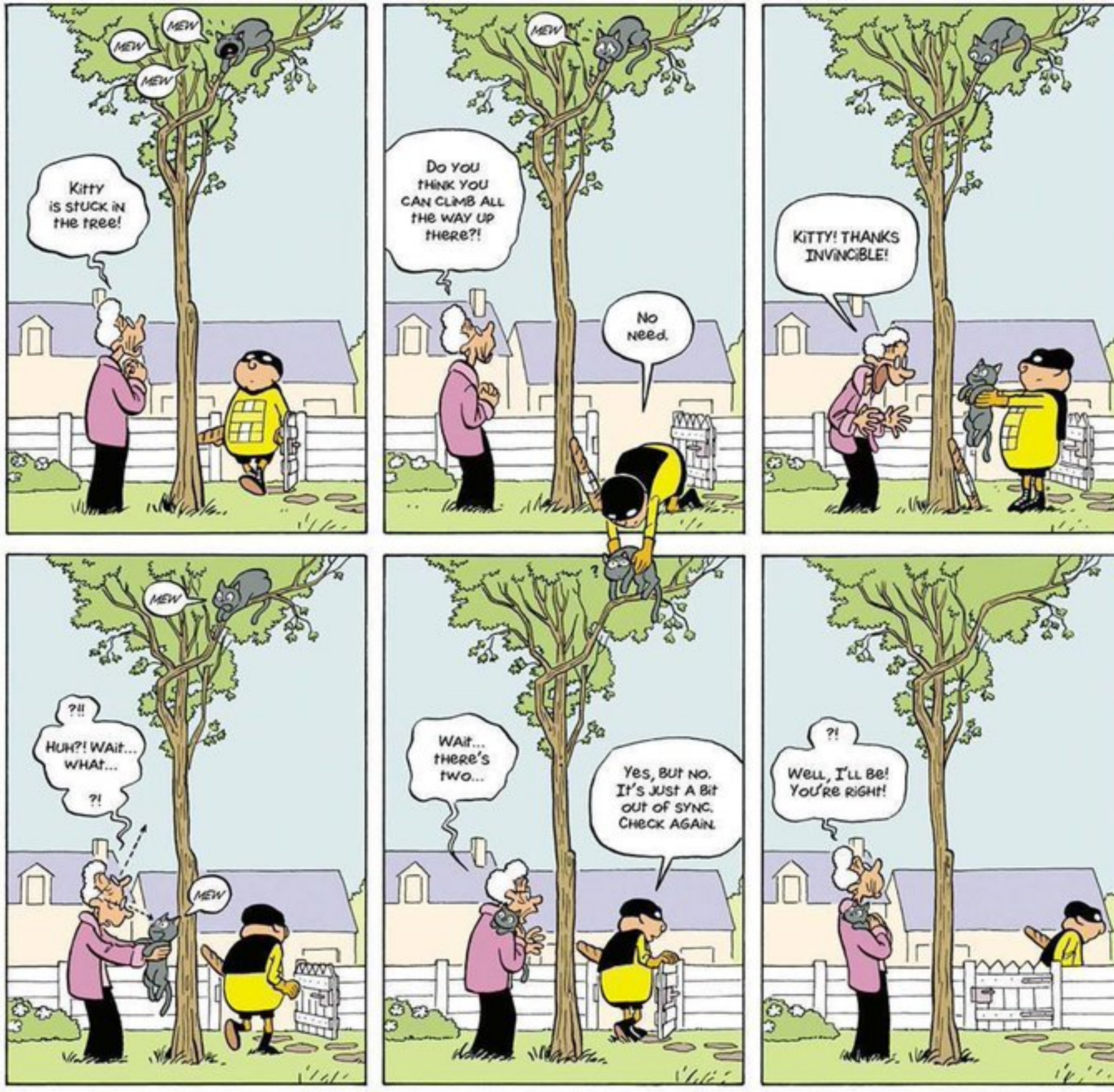
Outdated Data

Conflicts

Potential Concurrency Anomalies

Events / Operations coming out of order

Huge source of human error!



Eventual Consistency

Remember:

You do **not** get any isolation guarantees like ‘*Repeatable Read*’

Application needs to handle concurrency control:

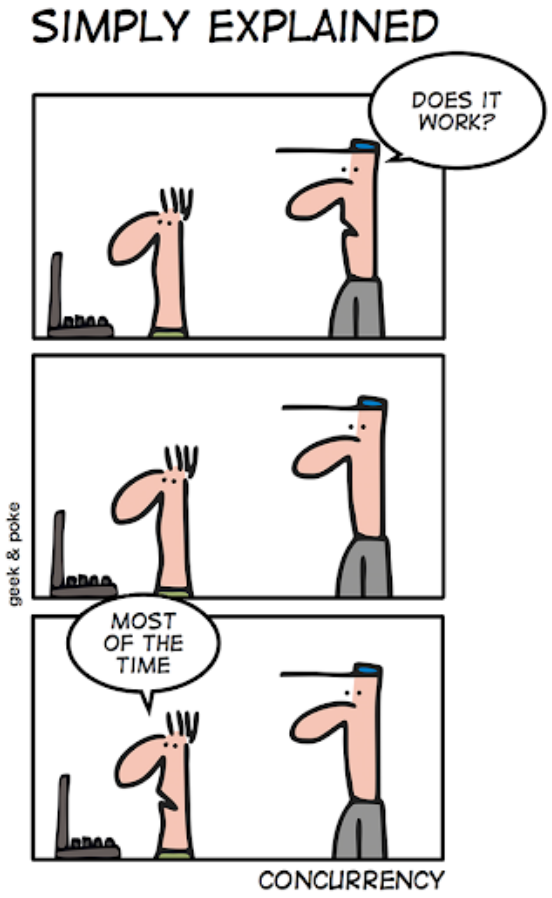
Hard to test

Issues emerge randomly in production

… are hard to reproduce

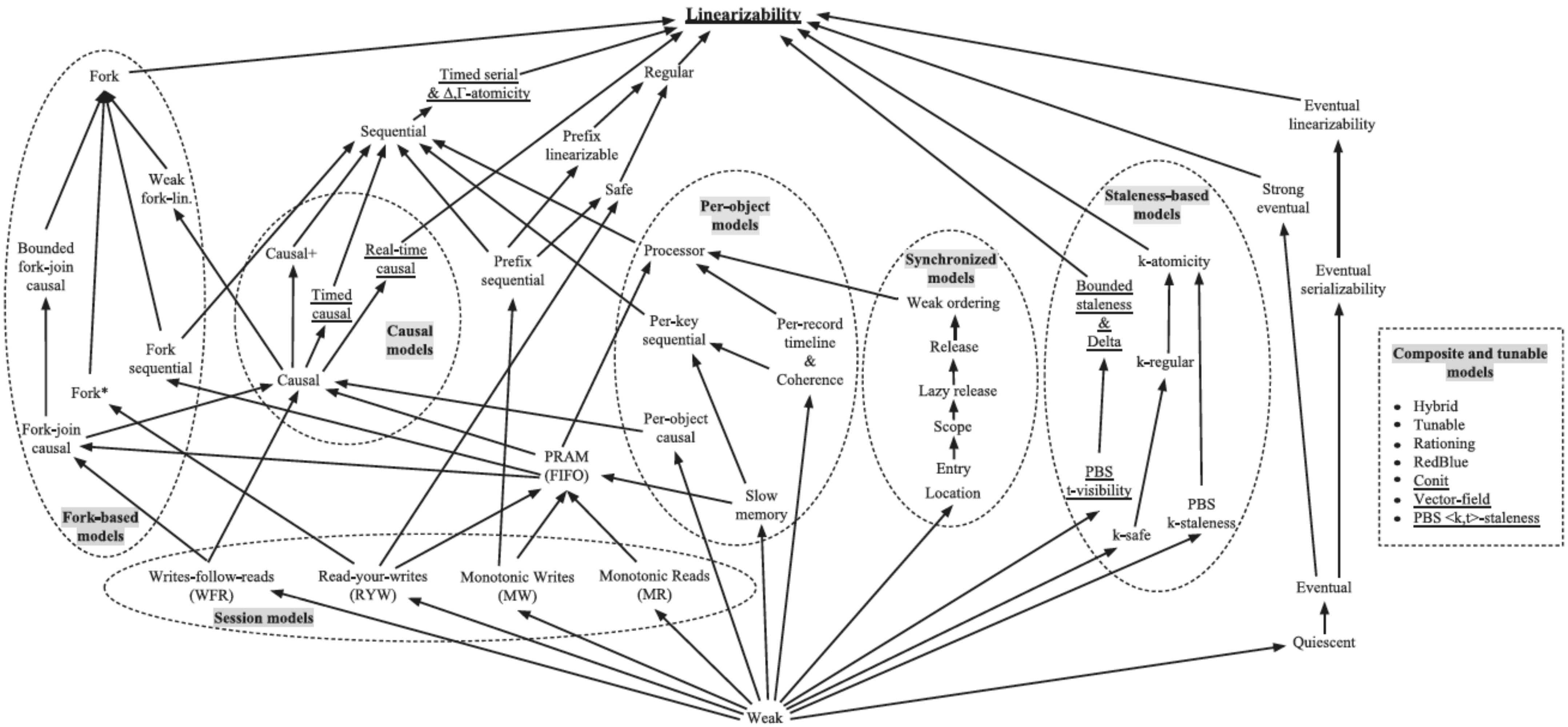
… are hard to debug

Huge source of human error!



Fraunhofer IESE

Consistency in Non-Transactional Distributed Storage Systems



Source: ACM Computing Surveys, Vol. 49, No. 1, Article 19, 2016

Pat Helland

Database & Distributed Systems Guru

Architect of multiple transaction & database systems (e.g. DynamoDB)

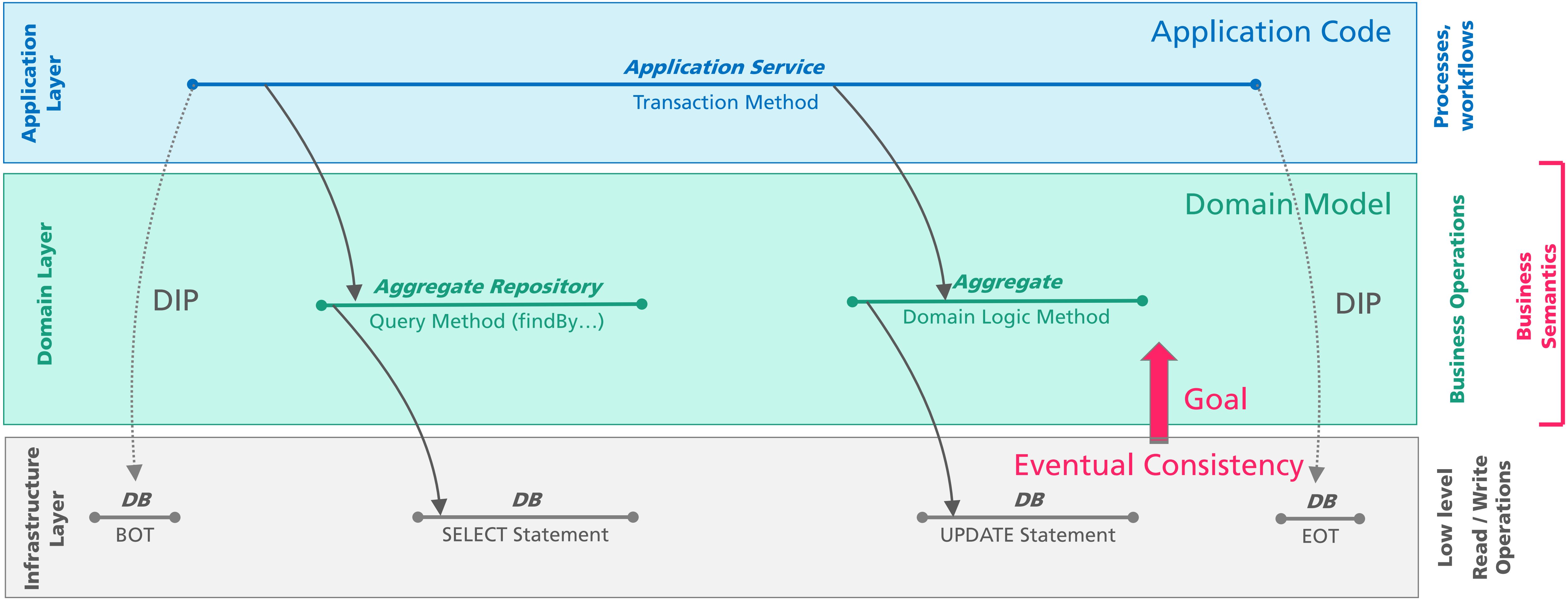
Worked at Microsoft, Amazon, SalesForce, ...

in terms of Reads and Writes

"... it is time for us to move past the examination of eventual consistency in terms of updates and storage systems. The real action comes when examining application-based operation semantics."

Conference on Innovative Data Systems Research, 2009

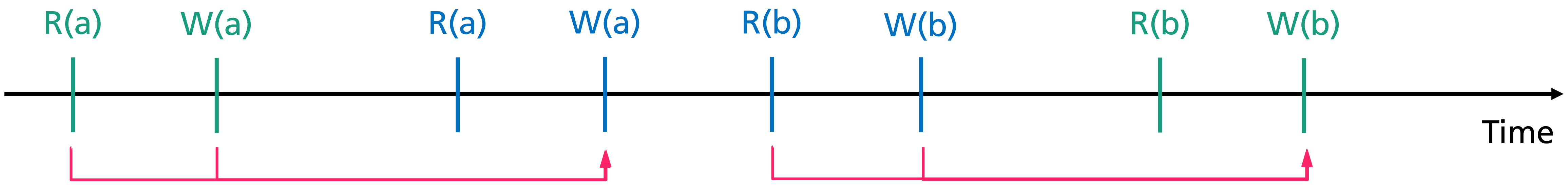
DDD Layered Architecture



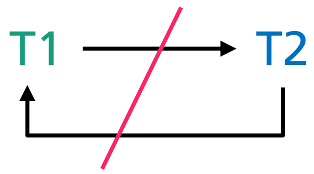
Recap Concurrency Control in Relational DBs

- A schedule of concurrent transactions is **conflict-serializable** iff the conflict graph is acyclic and compatible with the execution order of the conflicting operations

Transactions T1, T2:



Conflict graph:



!

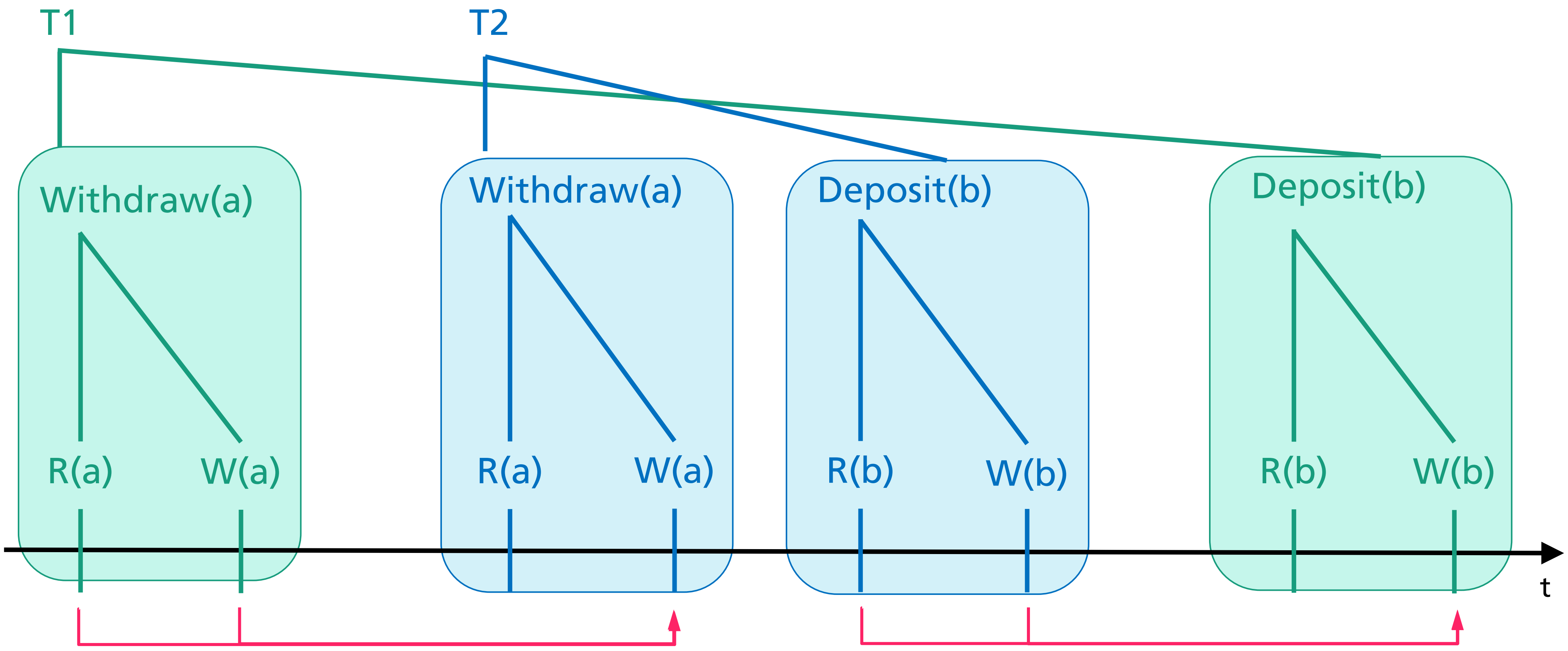
Conflict graph is cyclic

⇢ No conflict serializability

⇢ Schedule would be rejected

Fraunhofer IESE

Business Semantics - Banking



Fraunhofer IESE

Multilevel Transactions
(Weikum et al. 1992)

Goal: Increase number of operations that can run concurrently!

- **Exploit semantics of operations in level-specific conflict relations that reflect the commutativity / compatibility of operations**

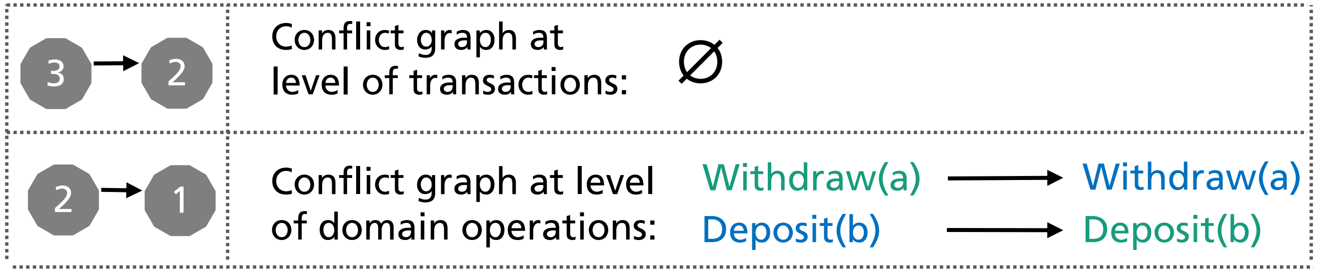
- Transactions are decomposed into operations and the operations again into sub-operations on multiple levels
  - **Transactions, Business operations, Low-level read and write operations**

- At each level a conflict relationship is defined
  - read-write conflicts and write-write conflicts on the same data item conflict at the lowest level
  - Non-commutative operations are conflicting on the level of business operations

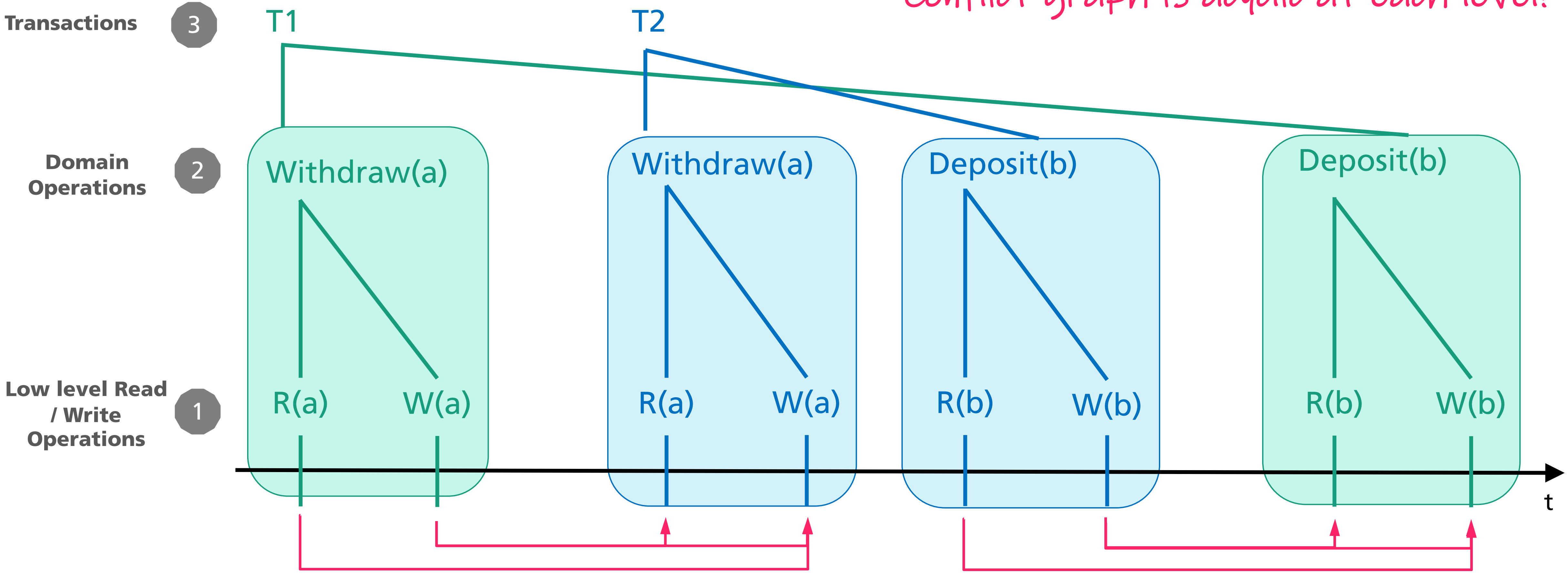
- If at each level the conflict serialization graph is acyclic then the multilevel schedule is in total **multilevel serializable**

In practice comparable to serializability!

Multilevel Transactions Example
(Weikum et al. 1992)

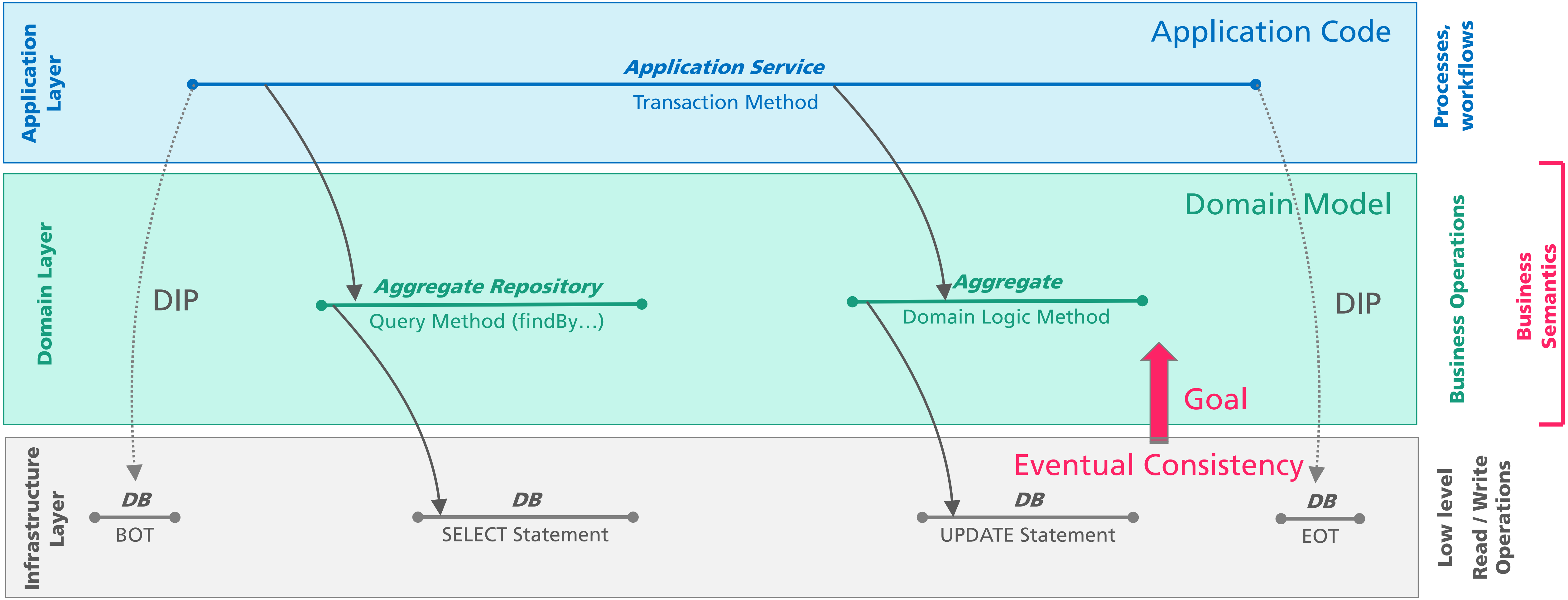


Conflict graph is acyclic at each level!



Fraunhofer IESE

DDD Layered Architecture



Fraunhofer IESE

Domain Operation Design

Pat Helland

Database & Distributed Systems Guru

Architect of multiple transaction & database systems (e.g. DynamoDB)

Worked at Microsoft, Amazon, SalesForce, …

A	Associative	(ab)c = a(bc)
C	Commutative	ab = ba
I	Idempotent	aa = a
D	Distributed	Operations executed out of order…

2.0

Conference on Innovative Data Systems Research, 2009

Commutative Operations

o.domainOperation1(..)

o.domainOperation2(..)

=

o.domainOperation2(..)

o.domainOperaton1(..)

'o' is some

Aggregate / Entity / Domain Service

Popular Examples in Scientific Publications

- Counters - Integer Addition
- Sets – Insert

- Banking – Withdraw
- Banking – Deposit

Fraunhofer IESE

Annette Bieniusa

CRDT Guru

Co-Creator of AntidoteDB

Worked at INRIA with Marc Shapiro, TU Kaiserslautern

INRIA ROCQUENCOURT

Conflict-free Replicated Data Types *

Marc Shapiro, INRIA & LIP6, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Carlos Baquero, Universidade do Minho, Portugal
Marek Zawirski, INRIA & UPMC, Paris, France

Thème COM — Systèmes communicants
Projet Regal

Rapport de recherche n° 7687 — Juillet 2011 — 18 pages

Abstract: Replicating data under Eventual Consistency (EC) allows any replica to accept updates without remote synchronisation. This ensures performance and scalability in large-scale distributed systems (e.g., clouds). However, published EC approaches are ad-hoc and error-prone. Under a formal Strong Eventual Consistency (SEC) model, we study sufficient conditions for convergence. A data type that satisfies these conditions is called a Conflict-free Replicated Data Type (CRDT). Replicas of any CRDT are guaranteed to converge in a self-stabilising manner, despite any number of failures. This paper formalises two popular approaches (state- and operation-based) and their relevant sufficient conditions. We study a number of useful CRDTs, such as sets with clean semantics, supporting both *add* and *remove* operations, and consider in depth the more complex Graph data type. CRDT types can be composed to develop large-scale distributed applications, and have interesting theoretical properties.

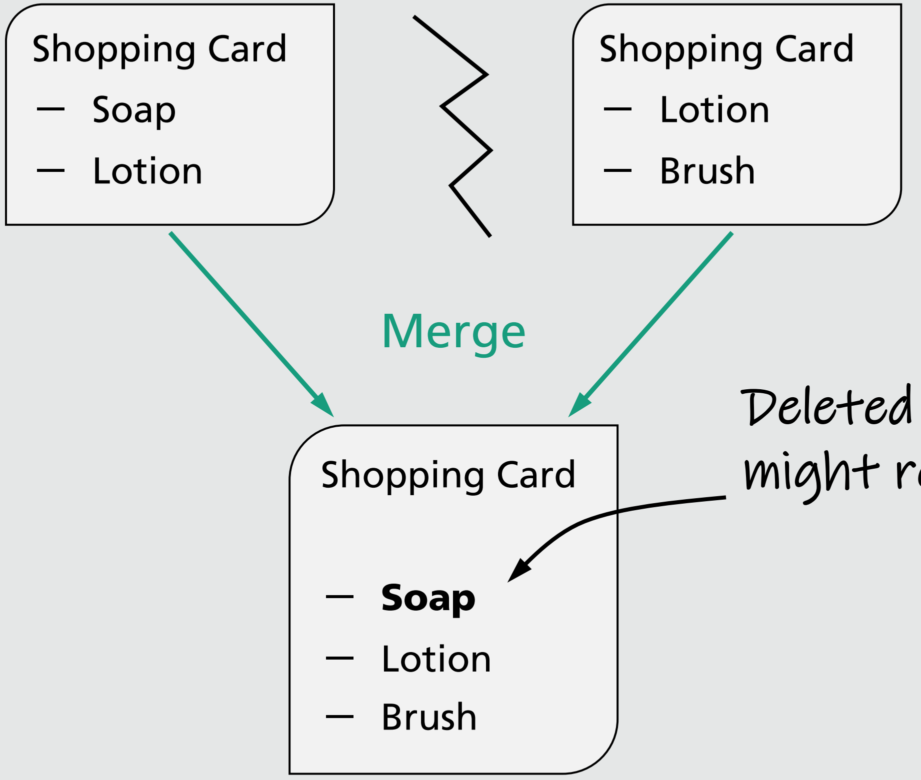
Conflict-Free Replicated Data Types (CRDTs)

Consider LUB as union of different object states produced at different replicas

CRDTs ship with <u>commutative merge operations</u> designed to be a <u>least upper bound (LUB)</u> of the conflicting versions.

CRDTs are grounded in algebraic theories of monotonic semilattices

Intuitive Example: Amazon's Shopping Card*



Deleted items might reappear

* Werner Vogels, Communications of the ACM, Volume 52, Issue 1, 2009

Beware of Domain Invariants

Best Practice

Model Domain Invariants explicitly!

Commutativity ?

Examples:

Banking – Withdraw

```
withdraw(amount) {
...
assert(balance > dispoLimit)
}
```

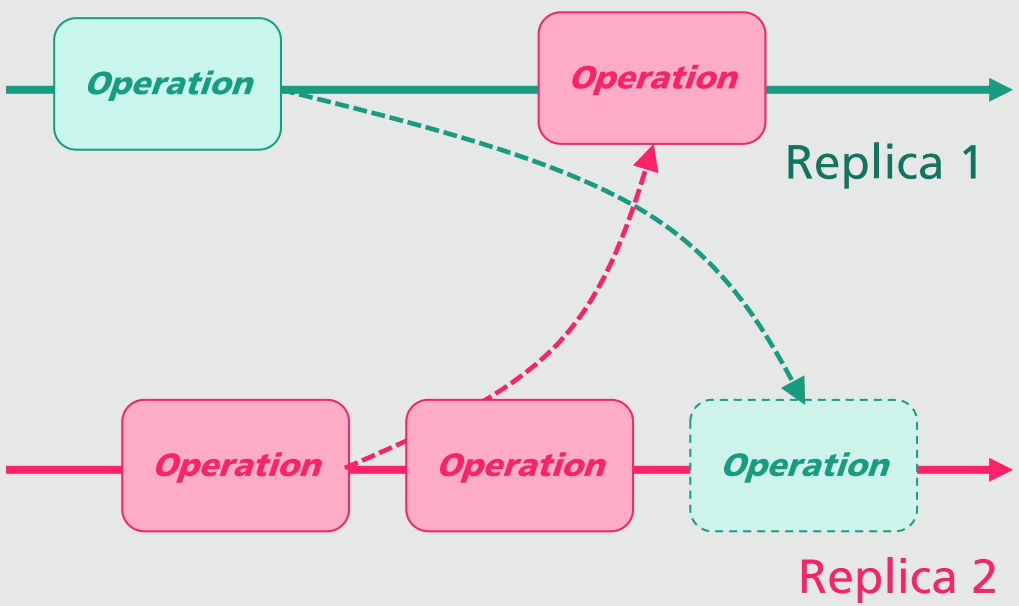

Fraunhofer IESE

“Distributed” Operations

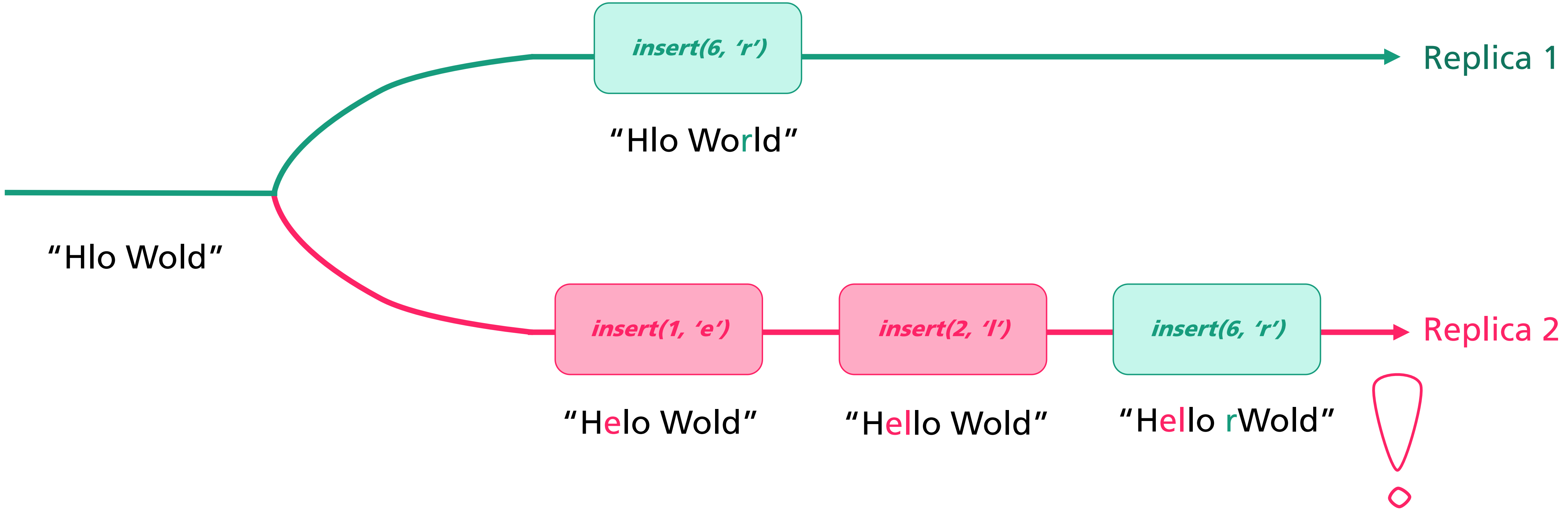
Concurrent operations can be executed in a different order on different replicas.

Domain Operations need the ability to produce intended updates if executed on different states on different replicas!

Example: Concurrent operations



Collaborative Text Editing



Fraunhofer IESE

Domain Data Design

Pat Helland

Database & Distributed Systems Guru

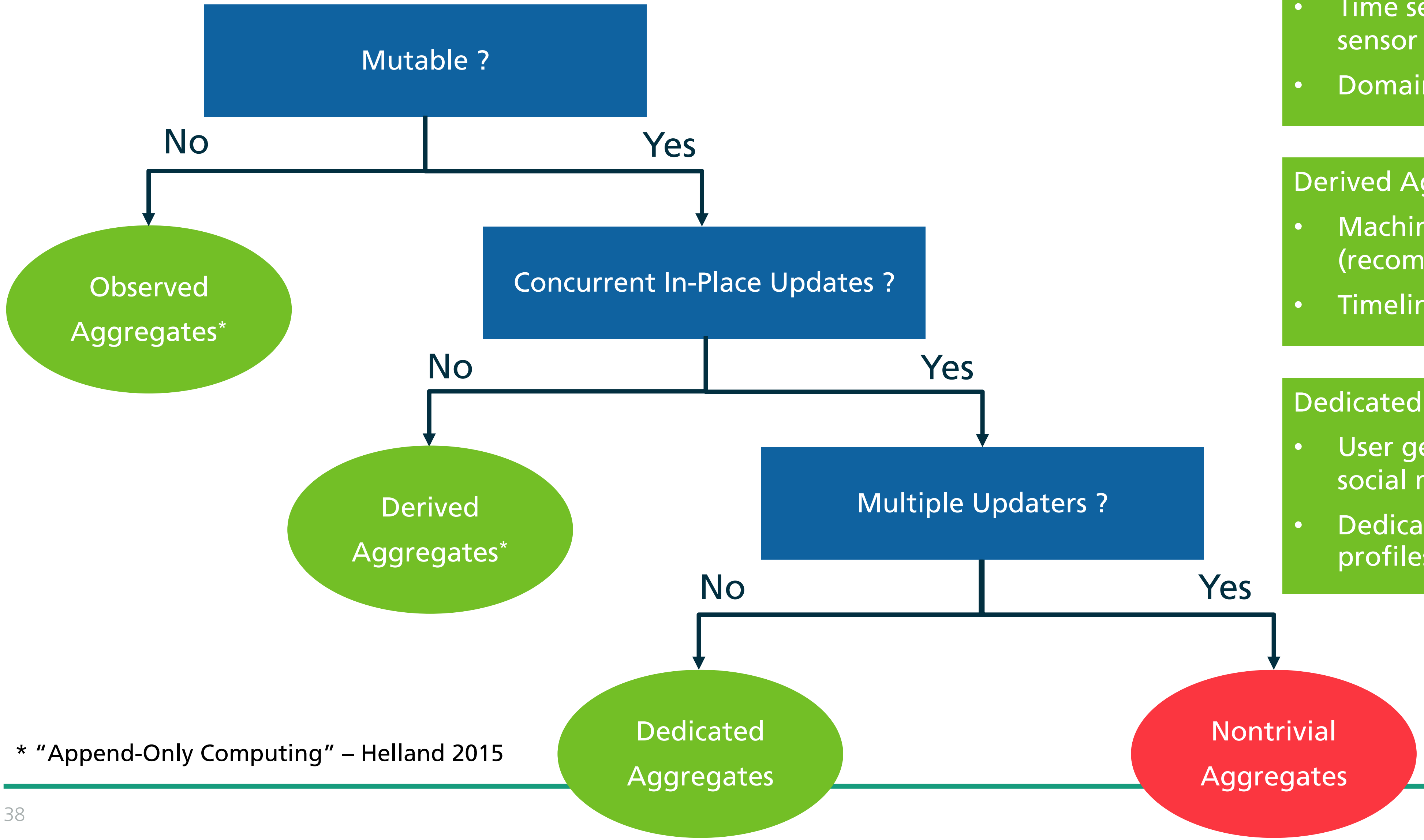
Architect of multiple transaction & database systems (e.g. DynamoDB)

Worked at Microsoft, Amazon, SalesForce, …

"Immutability Changes Everything"

ACM Queue, Volume 13, Issue 9, 2016

1st Level Classification of Replicated Aggregates



* "Append-Only Computing" – Helland 2015

Examples

Observed Aggregates:

- Time series data (machine sensor data, market data, …)
- Domain events

Derived Aggregates:

- Machine generated data (recommendations, …)
- Timeline or newsfeed data

Dedicated Aggregates:

- User generated data (reviews, social media posts, …)
- Dedicated master data (user profiles, account settings)

Fraunhofer IESE

Classification

2nd Level Classification of Nontrivial Aggregates

Nontrivial Aggregates	Update Frequency in Peak Times	Update Simultaneity in Peak Times	Concurrency Anomaly Probability
Collaboration Result Aggregates			
Activity Aggregates			
Reference Aggregates	low	improbable	low

Reference Aggregates Examples:

- Master data (CRM data, resources, products, …)
- Values (Valid currencies, product types, gender, …)
- Meta data (Tags, descrtiptive data of raw data, ..)

Fraunhofer IESE

2nd Level Classification of Nontrivial Aggregates

Nontrivial Aggregates	Update Frequency in Peak Times	Update Simultaneity in Peak Times	Concurrency Anomaly Probability
Collaboration Result Aggregates			
Activity Aggregates	high	probable	high
Reference Aggregates	low	improbable	low

Activity Aggregates Examples:

- State data of workflows, business processes, …
- Coordination data of joint activities (agricultural field operation, meeting, …)
- Task management data, Kanban board data, …

Fraunhofer IESE

Classification

2nd Level Classification of Nontrivial Aggregates

Nontrivial Aggregates	Update Frequency in Peak Times	Update Simultaneity in Peak Times	Concurrency Anomaly Probability
Collaboration Result Aggregates	very high	highly probable	very high
Activity Aggregates	high	probable	high
Reference Aggregates	low	improbable	low

Collaboration Result Aggregates Examples:

- Result data of collaborative knowledge work (CAD model, crop rotation plan, whiteboard diagram, …)
- Text data as result of collaborative authorship (manuals, scientific papers, meeting protocols, …)

Fraunhofer IESE

Best Practice

Concurrency Anomalies Impact Assessment

		Concurrency Anomaly Probability	Consequences of Data Corruption	Fixing Costs of Data Corruption
Nontrivial Aggregates	Collaboration Result Aggregates	very high	critical	very high
	Activity Aggregates	high	major	high
	Reference Aggregates	low	critical	very high
	Dedicated Aggregates	low	minor	moderate
Trivial Aggregates	Derived Aggregates	"Technical Immutability Border"	depends	moderate
	Observed Aggregates		critical	very high

A Classification of Replicated Data for the Design of Eventually Consistent Domain Models, S. Braun, S. Dessloch, ICSA 2020

Fraunhofer IESE

Estimation - Frequency of Classes in your Architecture Design

	Trad. Enterprise IS (ERP, CRM, Workflow Management)	**Social Media Apps** (Facebook, Twitter)	**Next: Data-Intensive Systems** (Smart Farming, Industrie 4.0)
Collaboration Result Aggregates	**30 %**		20%
Activity Aggregates	**30%**	1%	20%
Reference Aggregates	**30%**	4%	20%
Dedicated Aggregates	1 %	**50%**	20%
Derived Aggregates / Observed Aggregates	9 %	**45%** **Eventual Consistency is standard**	20%

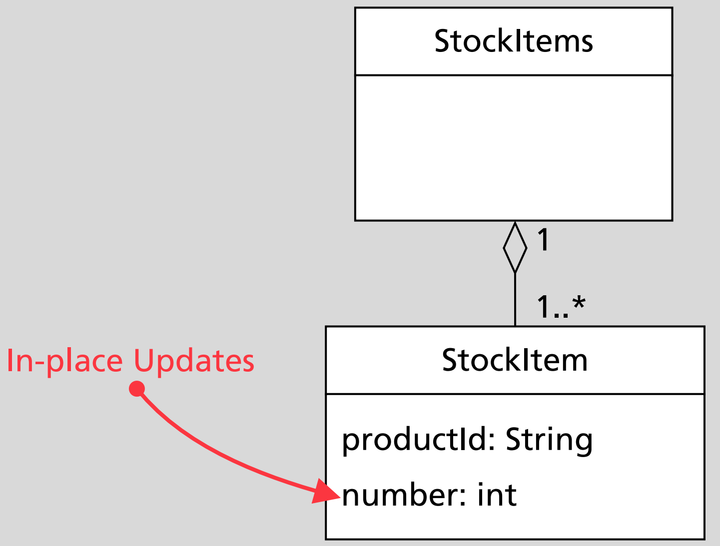
"Technical Immutability Border"

Fraunhofer IESE

Trivial Aggregates First

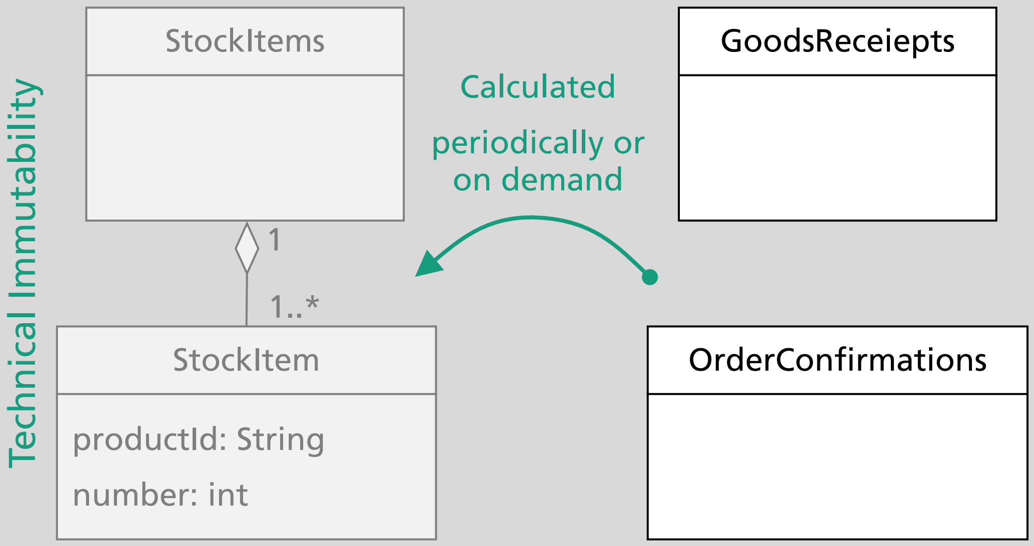
- **Whenever feasible, model aggregates as trivial aggregates***

Anti - Pattern



Nontrivial Activity Aggregate

Pattern



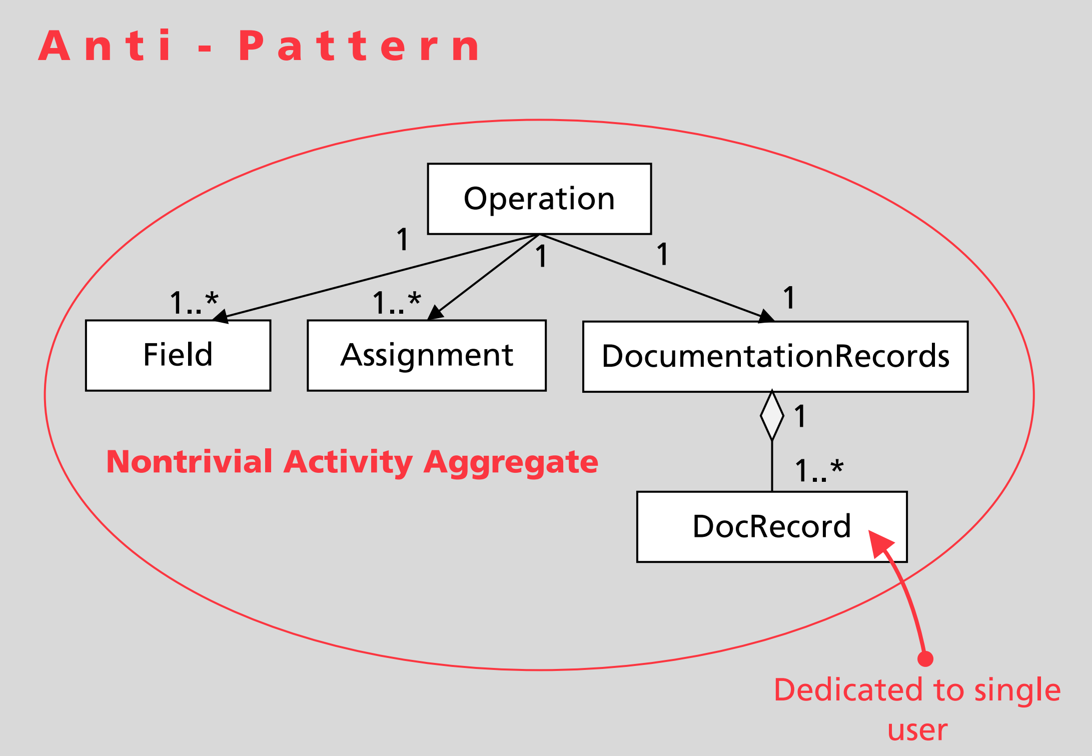
Derived Aggregate

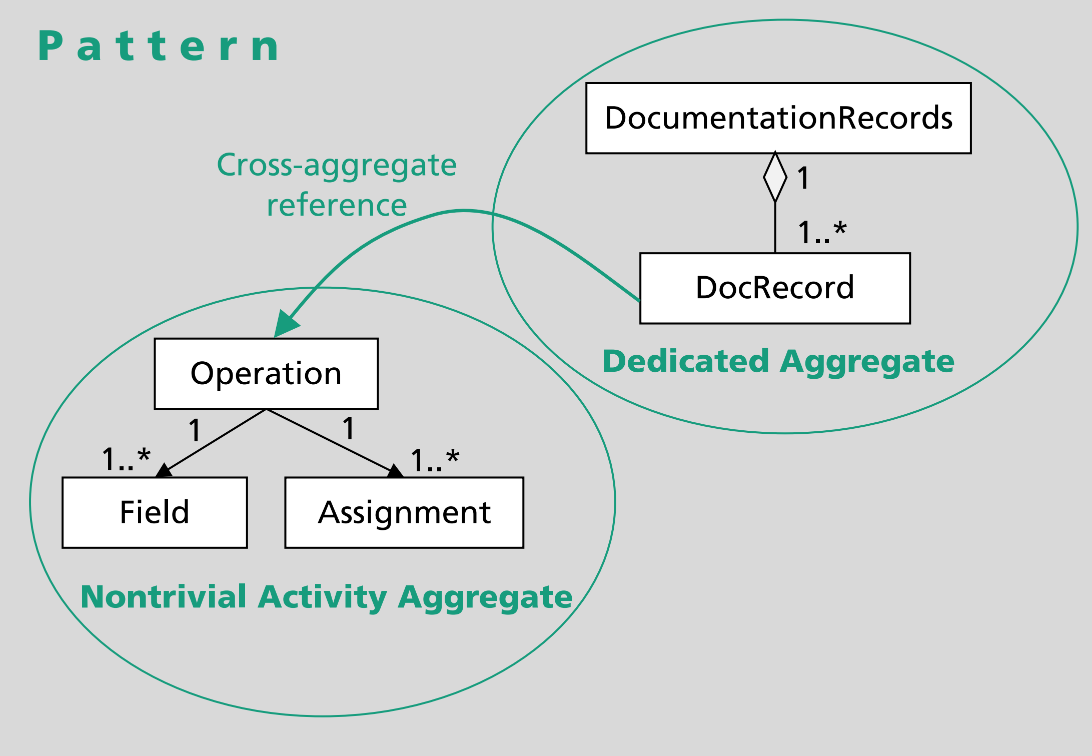
Observed Aggregates

* See also: Event Sourcing & CQRS (Command Query Responsibility Segregation)

Dedicated Aggregates are Solitary

- **Design dedicated data as self-contained aggregate**

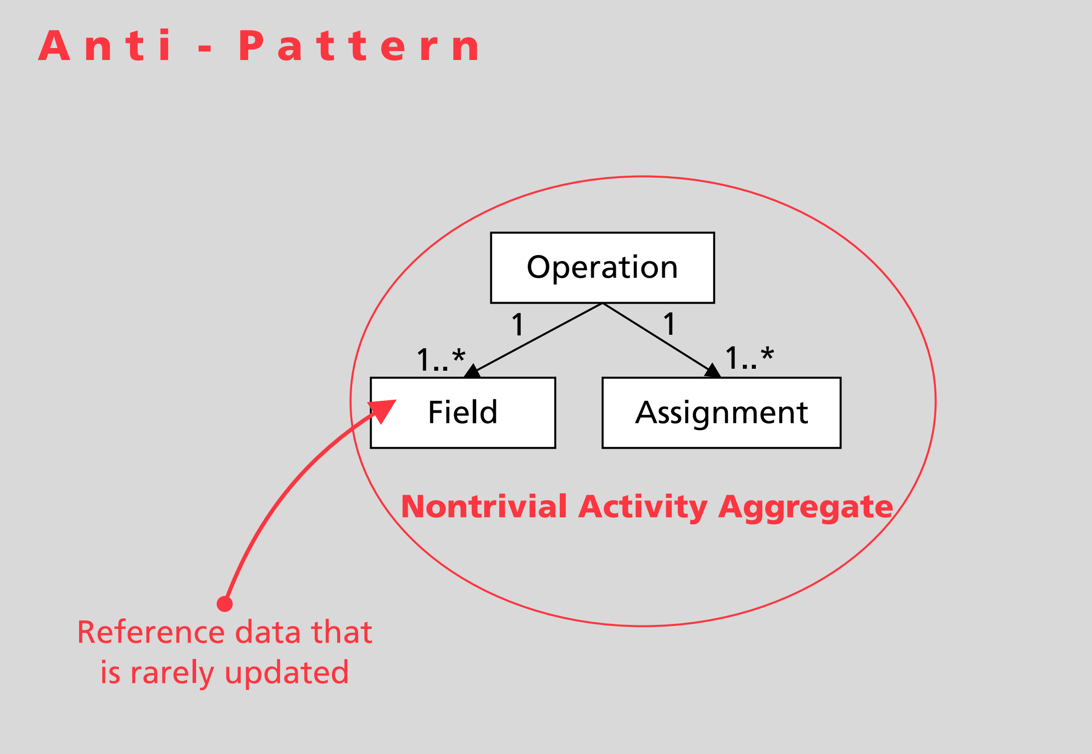


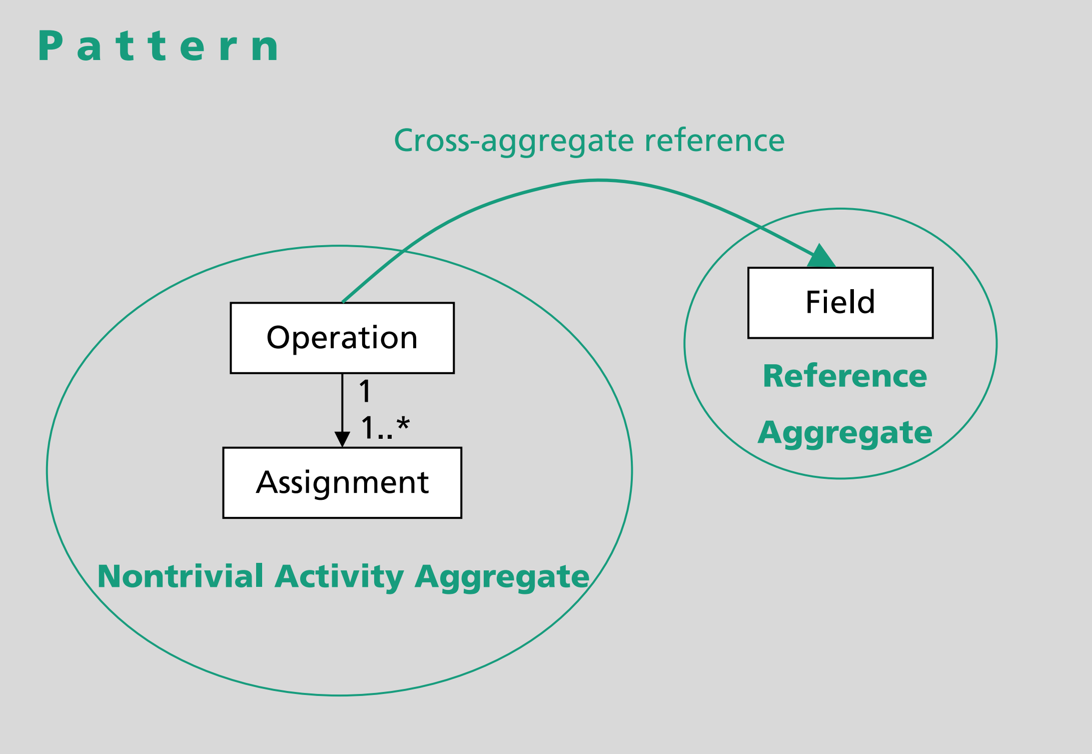


Fraunhofer IESE

Separation of Aggregate Classes

- **Whenever feasible, keep data of different classes in separate aggregates**





Derived Aggregates are idempotent

- **The calculation of the state of a derived aggregate should be idempotent & deterministic**

Best Practice

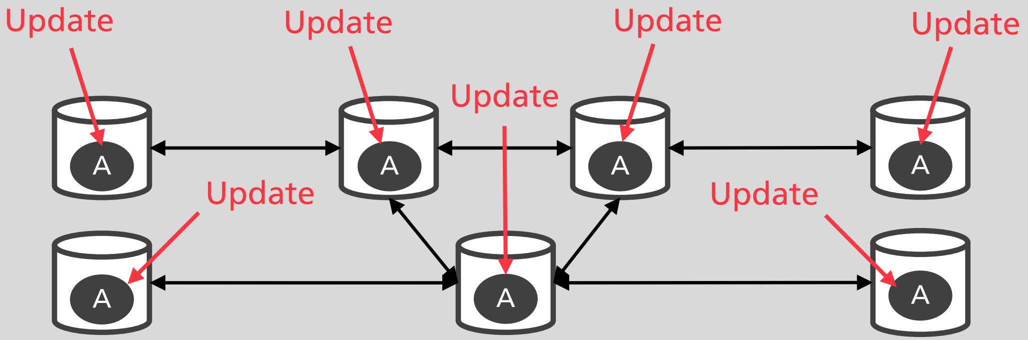
Fraunhofer IESE

Do not Forget the Master

- **Consider using <u>Primary Copy Replication</u>, if transactional guarantees are required**

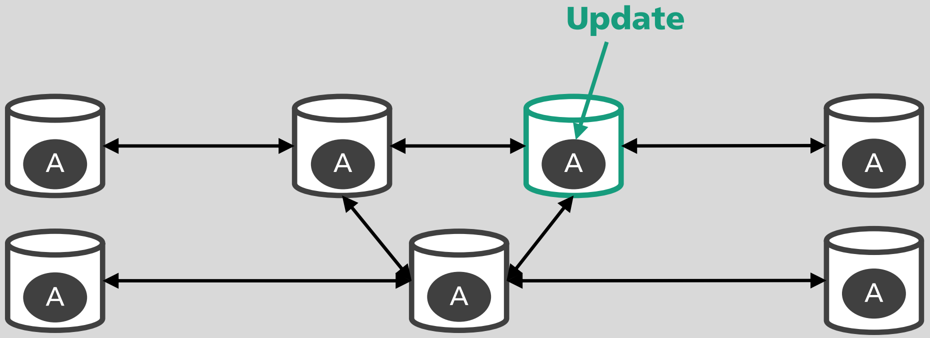
Anti - Pattern

Update Everywhere



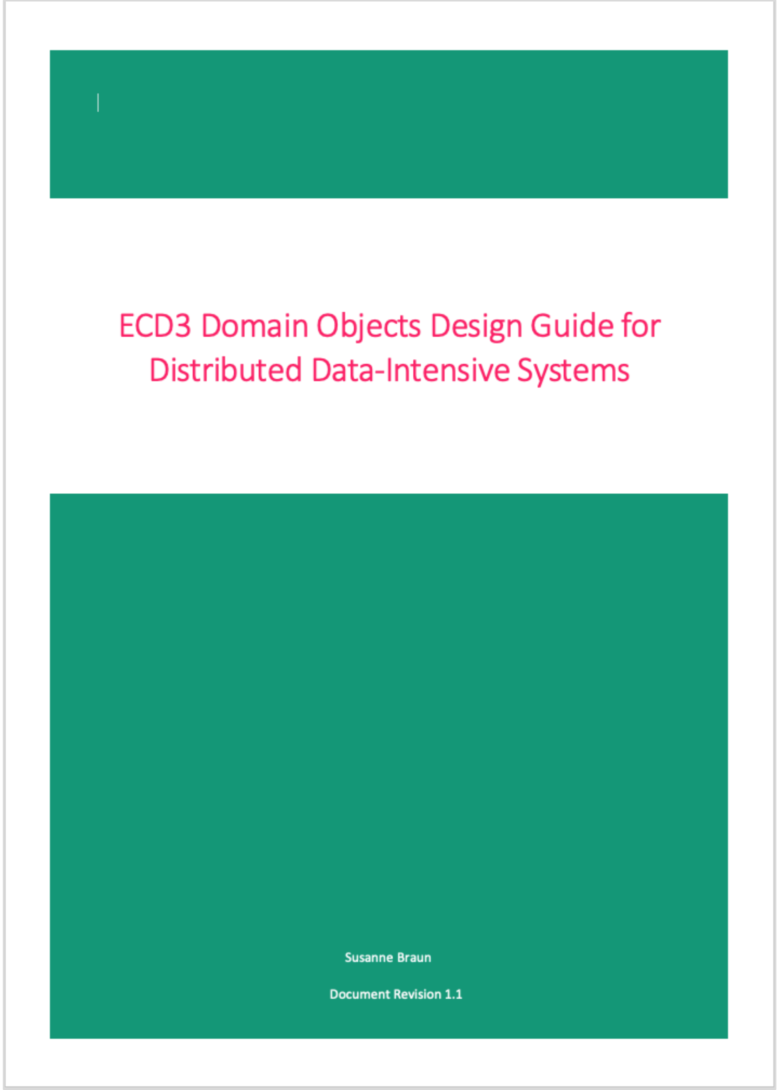
Pattern

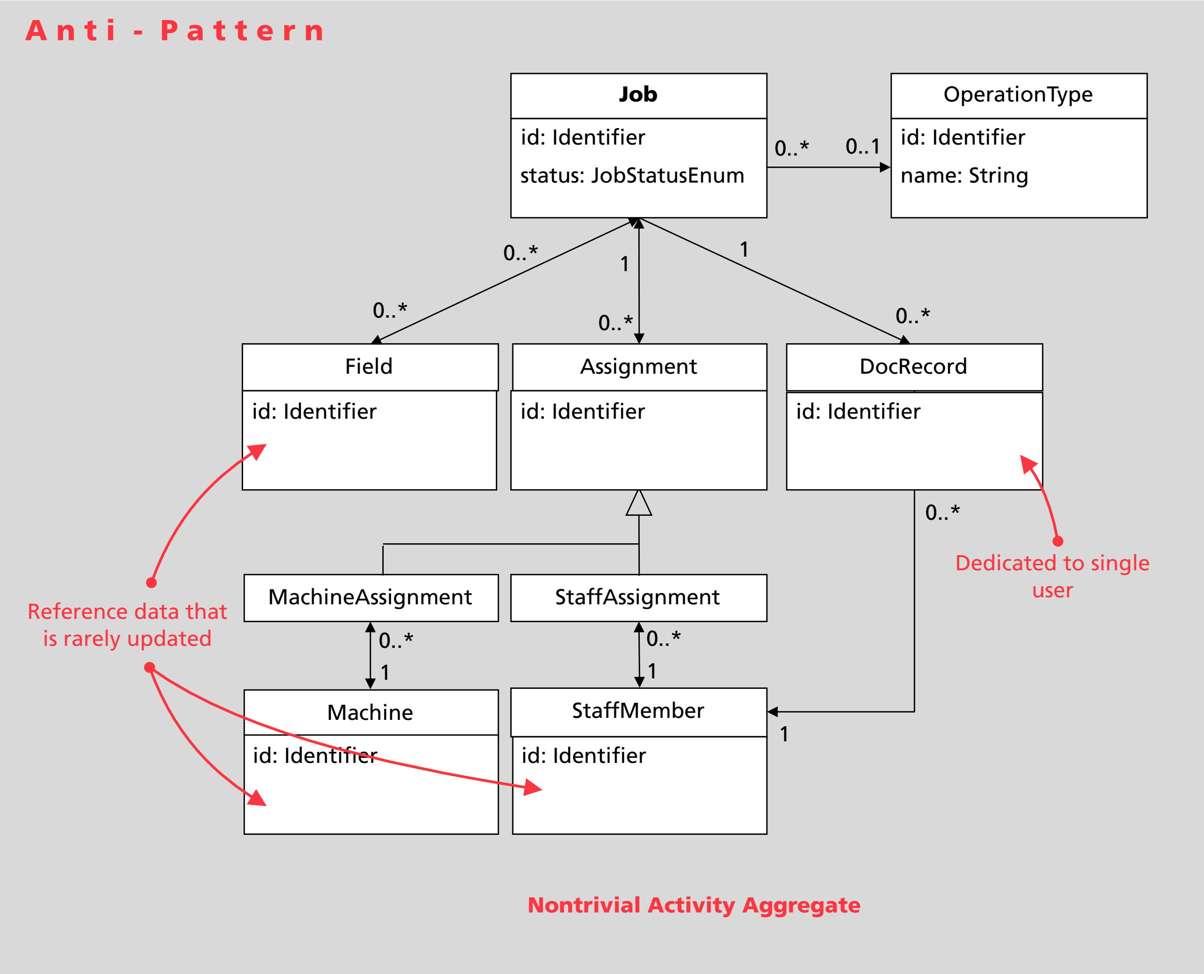
Primary Copy



Fraunhofer IESE

Extensive Guidance in the ECD3 Domain Objects Design Guide





Fraunhofer IESE

ECD3 Compatibility Relations

- To be published at the **8th Workshop on Principles and Practice of Consistency for Distributed Data**
- Of EuroSys 2021

Advanced Domain-Driven Design for Consistency in Distributed Data-Intensive Systems

Susanne Braun
Architecture Centric Engineering
Fraunhofer IESE
Kaiserslautern, Germany
susanne.braun@iese.fraunhofer.de

Annette Bieniusa
TU Kaiserslautern
Kaiserslautern, Germany
bieniusa@cs.uni-kl.de

Frank Elberzhager
Architecture Centric Engineering
Fraunhofer IESE
Kaiserslautern, Germany
frank.elberzhager@iese.fraunhofer.de

ABSTRACT

More and more data-intensive systems have emerged lately. Big Data, Artificial Intelligence, or cloud-native applications all require high scalability and availability. Data is no longer persisted in one central relational database with serialized and transactional access, but rather distributed and replicated among different nodes running only under eventual consistency. This poses a number of design challenges for software architects, as they cannot rely on a single system to mask the concurrency anomalies of concurrent access to distributed and replicated data. Based on three case studies, we developed a theory regarding how practitioners handle synchronization and consistency design challenges in distributed data-intensive applications. We also identified the "white spots" of missing design guidance needed by practitioners to handle the aforementioned challenges appropriately. We are currently evaluating our theory in the context of an action research study. In this study, we are also evaluating the novel design guidelines we are proposing in this regard, which, according to our theory, meet the needs of practitioners. Our design guidelines integrate with Domain-Driven Design, which is widely used in practice. Following the idea of multilevel serializability, we investigate the compatibility of business operations beyond commutativity. We provide concrete practical design guidance to achieve compatibility of non-commutative business operations. We also describe the basic infrastructure guarantees our design guidelines require from replication frameworks.

CCS CONCEPTS

• **Software and its engineering** → **Software design engineering; Software design tradeoffs;** • **Information systems** → **Distributed database transactions.**

KEYWORDS

domain-driven design, eventual consistency, data-intensive systems

ACM Reference Format:
Susanne Braun, Annette Bieniusa, and Frank Elberzhager. 2021. Advanced Domain-Driven Design for Consistency in Distributed Data-Intensive Systems. In *8th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC'21), April 26, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3447865.3457969

ACM ISBN 978-1-4503-8338-7/21/04…$15.00
https://doi.org/10.1145/3447865.3457969

1 INTRODUCTION

Distributed data-intensive systems pose new challenges for software architects and developers. To meet quality goals such as high availability and scalability, data is today no longer persisted in one central database, but is rather distributed and replicated across different nodes [26], often only being eventually consistent. Instead of relying on strong guarantees as given by ACID-compliant systems, developers basically have to design and build complex data synchronization schemes and also take care of concurrency control within the distributed system. This drastically increases the complexity of the systems software architects need to design. The implications are that software architects (and developers) need to have an in-depth understanding of the underlying concepts of traditional database transaction management, data replication, and distributed systems, and also need to be able to combine these. Unfortunately, in practice even senior staff often lacks a thorough understanding of these concepts. This has been confirmed by our observations from three medium to large case studies where **replicated data** was an enabler for achieving quality goals such as high availability and high scalability [8, 10, 11, 40]. We have described our observations in detail in a theory. This theory has already been accepted by an action research [49] study we are currently conducting. In this study, we are developing and evaluating novel design guidelines to help practitioners safely architect data-intensive systems that have heterogeneous consistency requirements. Our guidelines are an advancement of **Domain-Driven Design (DDD)** [19]. We therefore refer to them as ECD3 guidelines (ECD3 stands for "Eventually Consistent DDD"). To facilitate the design of domain models, we provide guidance for the design of domain objects (ECD3 Domain Objects Design Guide) and the design of domain operations (ECD3 Domain Operations Design Guide) [9].

In this paper, we extend our previous work on domain objects design [11] and provide an in-depth discussion of the ECD3 domain operation design criteria. We aim at increasing the number of domain operations that can run concurrently and free of conflicts on different replication nodes (short: replicas). Therefore, our guidelines take into account the compatibility and conflict relations of domain operations. We provide the following contributions:

- We propose novel criteria for the assessment of the compatibility relations of domain operations, which are easier to realize in practice than commutativity (as proposed in multilevel serializability [54]) (Section 4).

Future Work

- ECD³ – **E**ventually **C**onsistent **D**omain **D**riven **D**esign
  - Best Practices & Software Architecture Design Guidelines
  - Framework ☛ **Towards Multilevel Transactions**
- Action Research Study
- Workshops with Practitioners

EventuallyConsistentDDD/design-guidelines

We' re on Github!

Fraunhofer IESE

Susanne Braun

Software Developer & Architect

susanne.braun@iese.fraunhofer.de

@susannebraun

EventuallyConsistentDDD/design-guidelines

Fraunhofer IESE, Kaiserslautern

#Thanx
#StayHome