UPDATE PROPAGATION CONCEPT

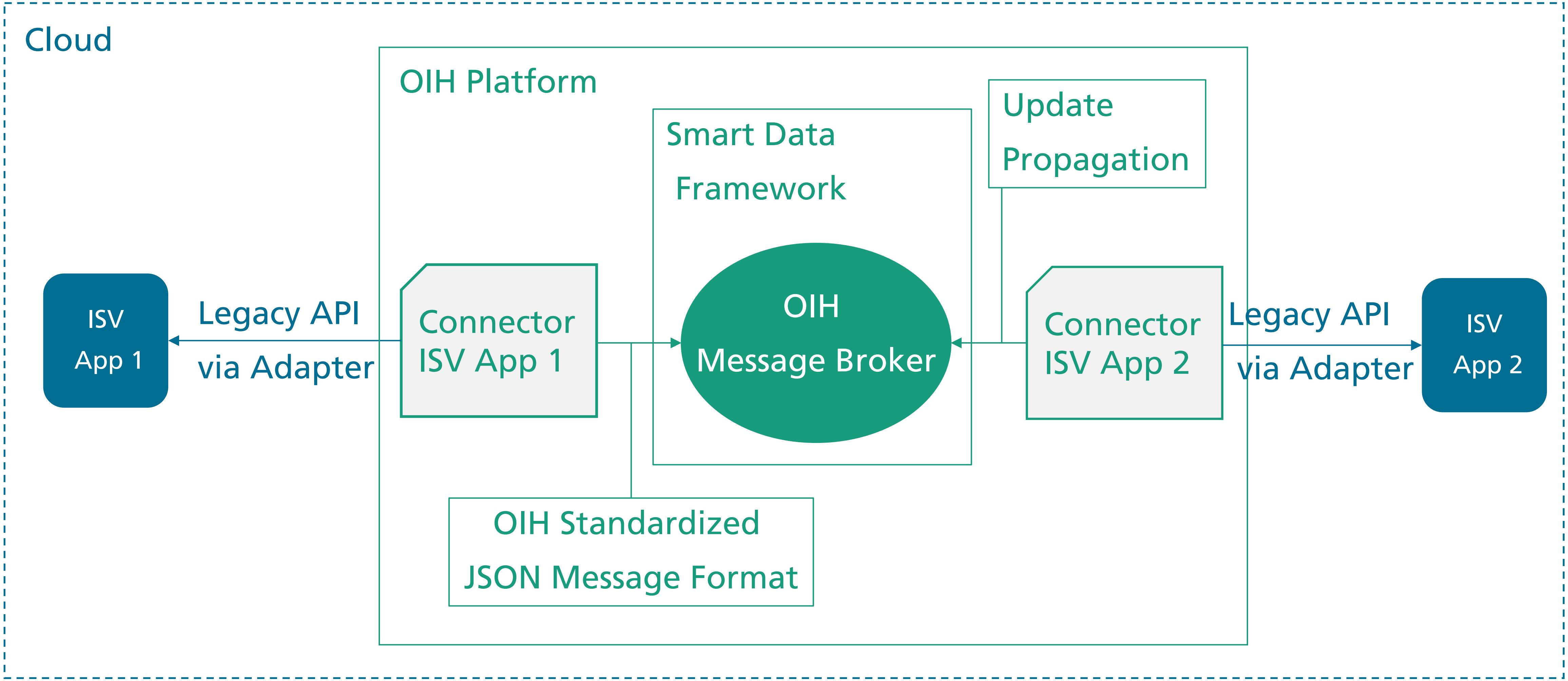
OIH Architecture Jour Fixe, 09.03.2018

Fraunhofer

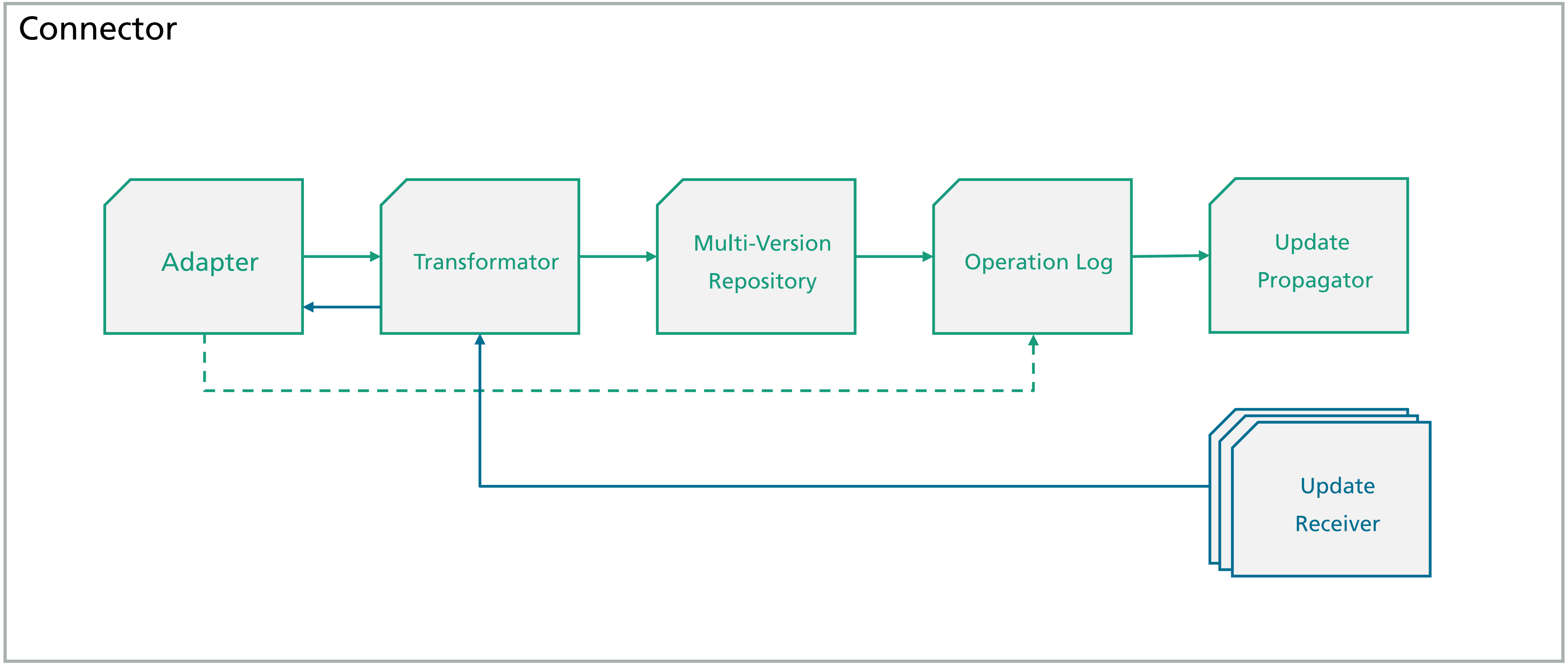
IESE

Building Blocks

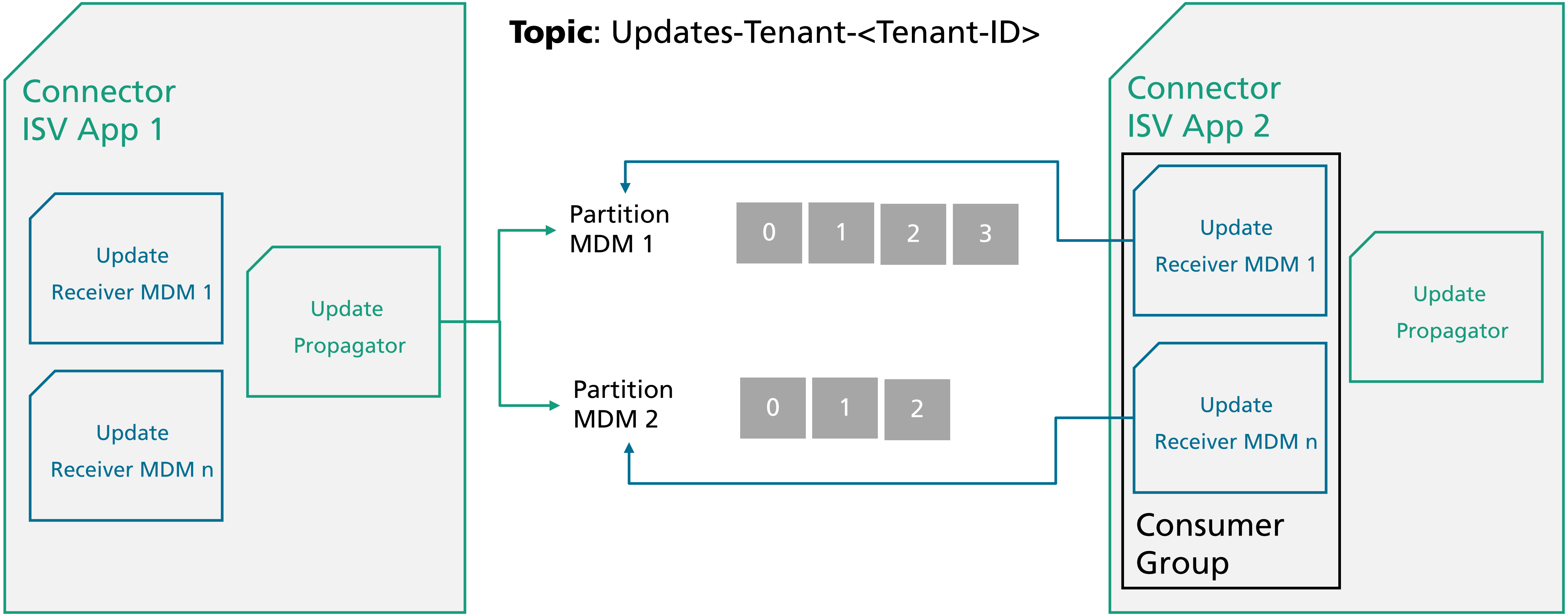
ISV App Ecosystem within one Cloud



Connector


Connector
Adapter
Transformator
Multi-Version Repository
Operation Log
Update Propagator
Update Receiver

Kafka Setup



Fraunhofer IESE

Message Format

```
{
    "$schema": "http://www.openintegrationhub.de/draft-01/schema#",
    "description": "The OIH message format for update propagation.",
    "type": "object",
    "required": [
        "masterDataModelUuid",
        "masterDataModelVersion",

        "aggregateType",
        "aggregateUuid",

        "operationType",
        "operationTime",
        "operationOriginAppUuid"
    ],
    "properties": {
        "masterDataModelUuid": { "type": "string" },
        "masterDataModelName": { "type": "string" },
        "masterDataModelVersion": { "type": "string" },

        "aggregateType": { "type": "string" },
        "aggregateUuid": { "type": "string" },

        "operationType": {
            "type": "string",
            "enum": ["create", "update", "delete"]
        },
        "operationTime": { "type": "string", "format": "date-time"  },
        "operationOriginAppUuid": { "type": "string" },

        "securityUserUuid": { "type": "string" },
        "securityUserRole": { "type": "string" },

        "aggregate": {
            "type": "object"
            }
        }
    }
}
```

Message Example


```
{
    "masterDataModelUuid": "bc9c46fe-238b-11e8-b467-0ed5f89f718b",
    "masterDataModelName": "de.openintegrationhub.Contacts",
    "masterDataModelVersion": "1.0",

    "aggregateType": "Contact",
    "aggregateUuid": "bc9c46fe-238b-11e8-b467-0ed5f89f718b",

    "operationType": update",
    "operationTime": "2007-04-05T12:30-02:00",
    "operationOriginAppUuid": "com.snazzycontacts.SnazzyContacts",

    "securityUserUuid": "bc9c46fe-238b-11e8-b467-0ed5f89f718b",
    "securityUserRole": "some-role",

    "aggregate": {
        "firstName": "Susanne"
    }
}
```

Backup

Sync Basics

>

(Scientific) Classification of Data Replication Approaches

<table>
<tr><th></th><th></th><th colspan="2">Update Location</th><th></th></tr>
<tr><th></th><th></th><th>Primary Copy</th><th>Update Anywhere</th><th></th></tr>
<tr><th rowspan="4">Synchronization Time</th><th rowspan="3">eager / synchronous</th><td>+ simple synchronization</td><td>+ flexible</td><td></td></tr>
<tr><td colspan="2">+ strong consistency
- potentially long response times</td><td></td></tr>
<tr><td>- inflexible</td><td>- complex synchronization
- bad scalability</td><td></td></tr>
<tr><th>lazy / asynchronous</th><td>+ simple synchronization
+ usually performant
- stale data
- inflexible</td><td>+ flexible
+ always performant
- stale data
- inconsistencies
- conflict resolution</td><td>Optimistic Data Replication</td></tr>
</table>

Source:
- J. Grey et. al, The dangers of replication and a solution
- Verteiltes und Paralleles Datenmanagement, Rahm, Saake, Sattler, p. 287

Fraunhofer IESE

Challenges of Optimistic Replication

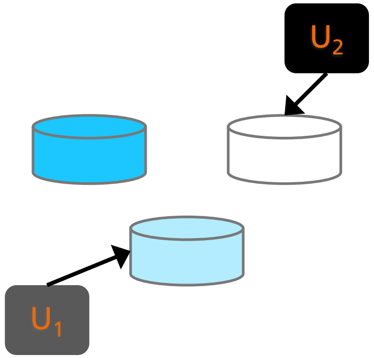
- CAP Theorem
- Inconsistencies / Eventual Consistency
- Conflict Detection & Conflict Resolution
- Concurrency Anomalies / Concurrency Control in Distributed System

ACID vs. BASE

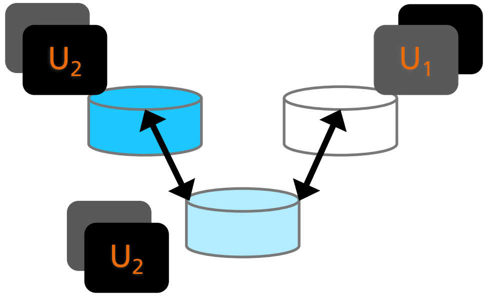
ACID	BASE
Strong consistency in the sense of one copy consistency	**Weak consistency** stale data and approximate answers OK
Isolation in the sense of one copy serializability	**Availability** first and highest priority
Pessimistic Synchronization global locks and synchronous propagation of updates	**Optimistic Synchronization** no locks, asynchronous propagation of updates, conflict resolution
Global Commit e.g. 2PC, majority consensus	**Independent local commits** conflict resolution, reconciliation

Five Major Concepts of Optimistic Data Replication

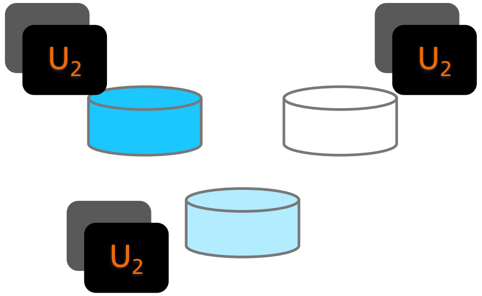
1. Independent update submission



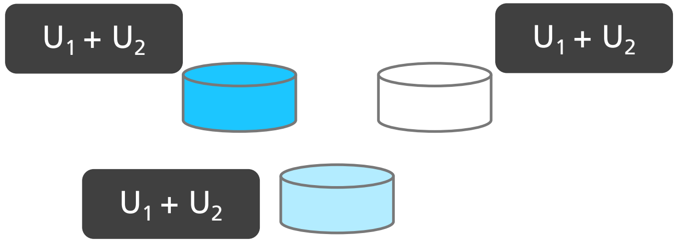
2. Asynchronous update propagation



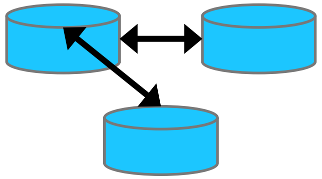
3. Update ordering



4. Reconciliation of concurrent writes

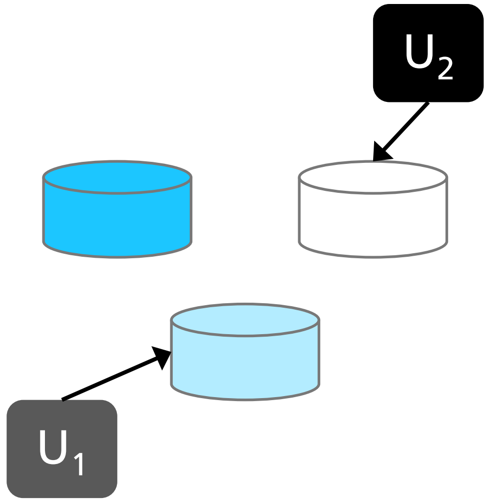


5. Consensus



Design Decisions

1. Independent Update Submission



- Design Decisions
  - **Change Tracking** (How to track updates that need to be propagated?)
  - **Sync State** (What state is maintained at a replica?)
  - **Metadata** (What metadata is stored and communicated about replicated items?)

DD Change Tracking

Design Alternatives:

- **Diff** approach: Changes are derived based on state-based comparison of data items each time synchronization is triggered (requires storage of “previous state” or a “Multiversion Repository”)
  - “minimally invasive” regarding existing / legacy systems
  - No semantics. Only “diff”, “previous state”, ”new state” can be propagated / used during conflict resolution
  - **Performance ?** Scalability ?
- **Logging** approach: Persistence layer or service layer is extended with a log for capturing changes
  - Aspect Oriented Programming (AOP): Use aspects for the monitoring of CRUD operations.
  - Model-Driven Development (MDD): CRUD Repositories are generated from (master data model). Repositories take care of either operation logging or setting of dirty flags / increase of version numbers.
  - MDD & Domain Driven Design (DD): CRUD Repositories and Services are generated from domain model. Execution of CRUD operations and high-level business operations can be tracked.

Legacy

Startups

DD Sync State at Replica

- Last synchronization time
- Global snapshot version of domain data
- Flag “outstanding changes / synchronization required”
- …

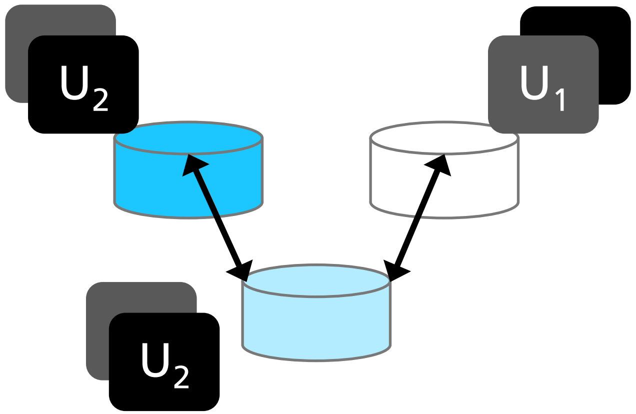
DD Metadata of Replicated Items

- Which Metadata is required for each replicated item?
  - Dirty Flag
  - Deleted Flag
  - Version Number
  - Version Vector
    - Happens-Before-Relationship
    - Concurrent changes
  - Source System (ID), Target Systems
  - Authoring entities (e.g. user id), Security Roles, ACLs
  - Master Data Model Version

DD Metadata of Replicated Items (cont.)

- Storage of Metadata at Replica and/or Connector ?
- Unit metadata is attached to ?
  - Aggregate
  - Entity
  - Attribute

2. Asynchronous Update Propagation

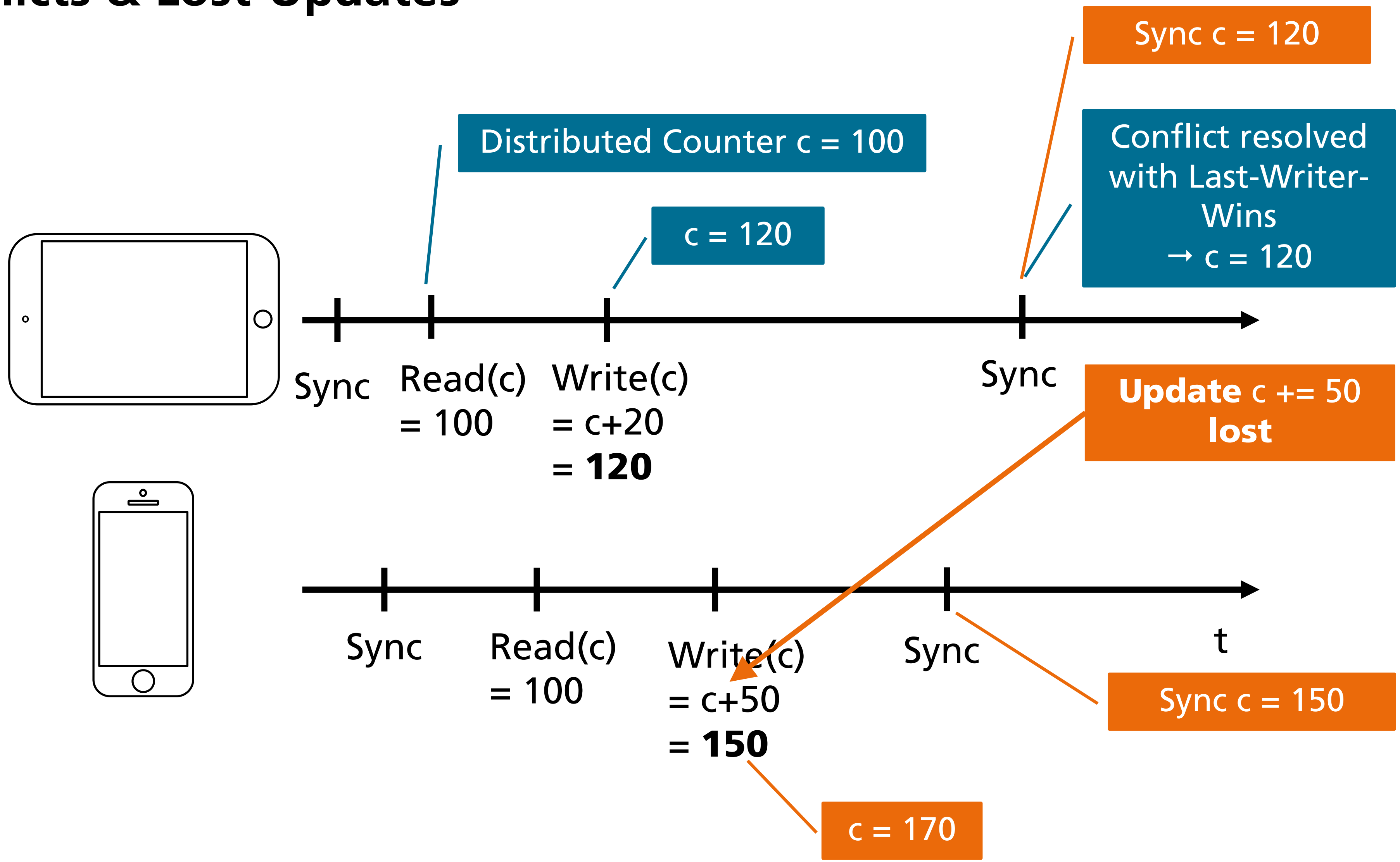


- Design Decisions
  - **Update Format** (Exchange of data items vs. exchange of update operations?)
  - **Communication** (What transport protocols are used?)
  - **Update Propagation Time / Intervals** (How to decide on the optimal point in time for update progpagation? Impact on scalability, data traffic and probability for conflicts)

DD Update Format

- **Design Alternatives**
  - **State-based** (current state of data is transferred)
    - State-of-the practice (e.g. Couchbase, Zumero, Relational DB replication products, …)
    - Concurrency anomalies like lost-updates, read skews and write skews possible
    - Easier to implement than operation-based approaches (in particular (eventual) consistency can be achieved much easier)

Write Conflicts & Lost-Updates



DD Update Format

- **Design Alternatives**
  - **Operation-based (**update operations are transferred)
    - More challenging to implement (existing research prototypes e.g. Bayou)
    - Exploitation of commutativity / compatibility of update operations to avoid concurrency anomalies like lost updates (see shared counter example)
    - Operational Transformation (Google Wave, Google Docs)
    - Conflict-Free-Replicated Data Types (CRDTs), research projects SyncFree, LightKone, AntidoteDB, Riak, …

Fraunhofer IESE

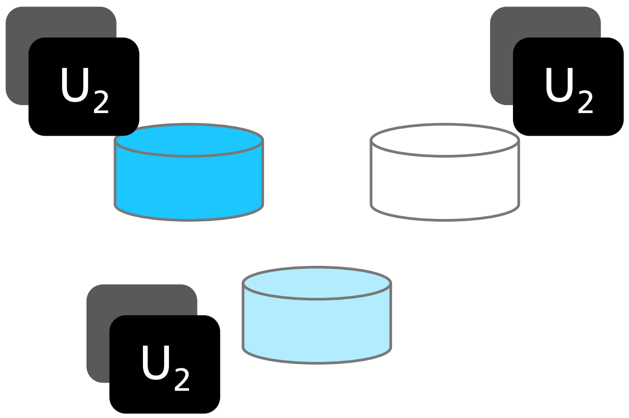
DD Communication

- **Proposal according to our last discussion at 08.02.2018**
  - Synchronous/Transactional delivery of updates to OIH platform (e.g. REST)
  - Asynchronous Propagation via some messaging middleware (e.g. RabbitMQ, Apache Kafka, …)

DD Synchronization Times / Intervals

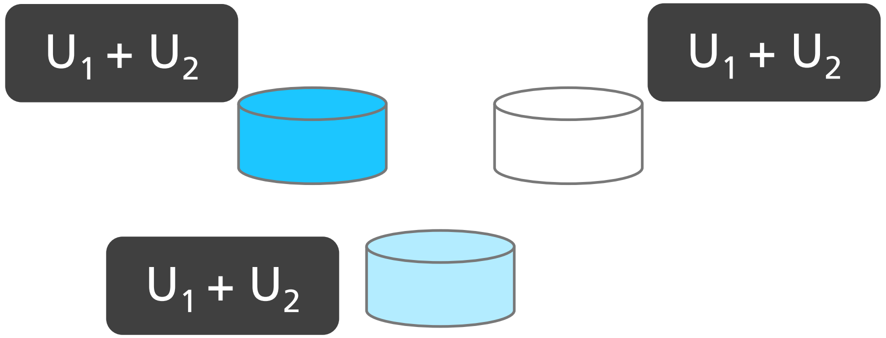
- Support for near-real-time propagation of updates ?
  - Support for prioritization of data streams?
- Configurable intervals ?

3. Update Ordering



- **Only required for operation-based approaches** in order to achieve eventual consistency
- Design Alternatives
  - Global ordering by central component / microservice (**Scalability?**)
  - Decentral **deterministic** ordering at each replica

4. Reconciliation of concurrent writes



- Reconciliation = Conflict Detection + Conflict Resolution
- Design Alternatives

Syntactic Reconcilation

vs.

Semantic Reconciliation

In Practice:

Syntactic conflict detection + syntactic conflict resolution

e.g. version numbers + last-writer-wins

Semantic resolution of conflicts on **Amazon shopping basket:**

Merge two versions -> deleted items might reappear

DD Reconciliation of concurrent writes

- **Design Alternatives**
  - Out-of-the-box **Syntactic Conflict Resolution** strategies
    - Last-Witer-Wins
    - First-Writer-Wins
    - Replica-ID-Wins (Leading Sysem / Primary Copy)
    - User-ID-Wins
    - Security-Role-Wins
  - Custom **Semantic Conflict Resolution** strategies
    - Callbacks:
      - Invoke custom conflict resolution handler in Connector

Lost-Updates possible

Complex implementation task

DD Reconciliation of concurrent writes

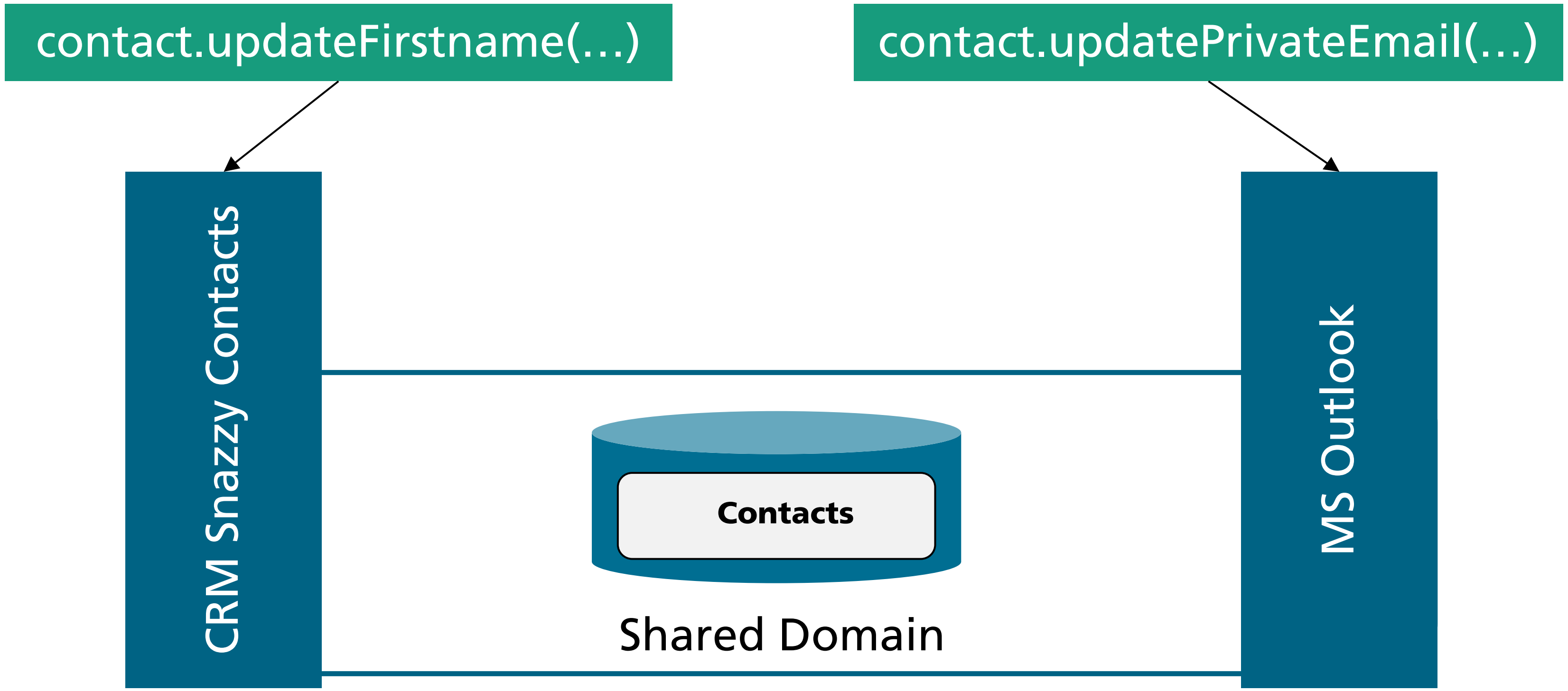
- **Design Alternatives**
  - Avoid conflicts by **application design** and **operation-based** approaches
    - Design business operations for maximum compatibility
      - E.g. **Event Sourcing + CQRS**
      - No conflicts, as everything is an insert operation into the write schema
      - Display values (read schema) calculated based on write schema
      - **-> Would require standardization of business operations on top of master data models in OIH**

Only Startups / No Legacy

Fraunhofer IESE

Conflict Examples OIH

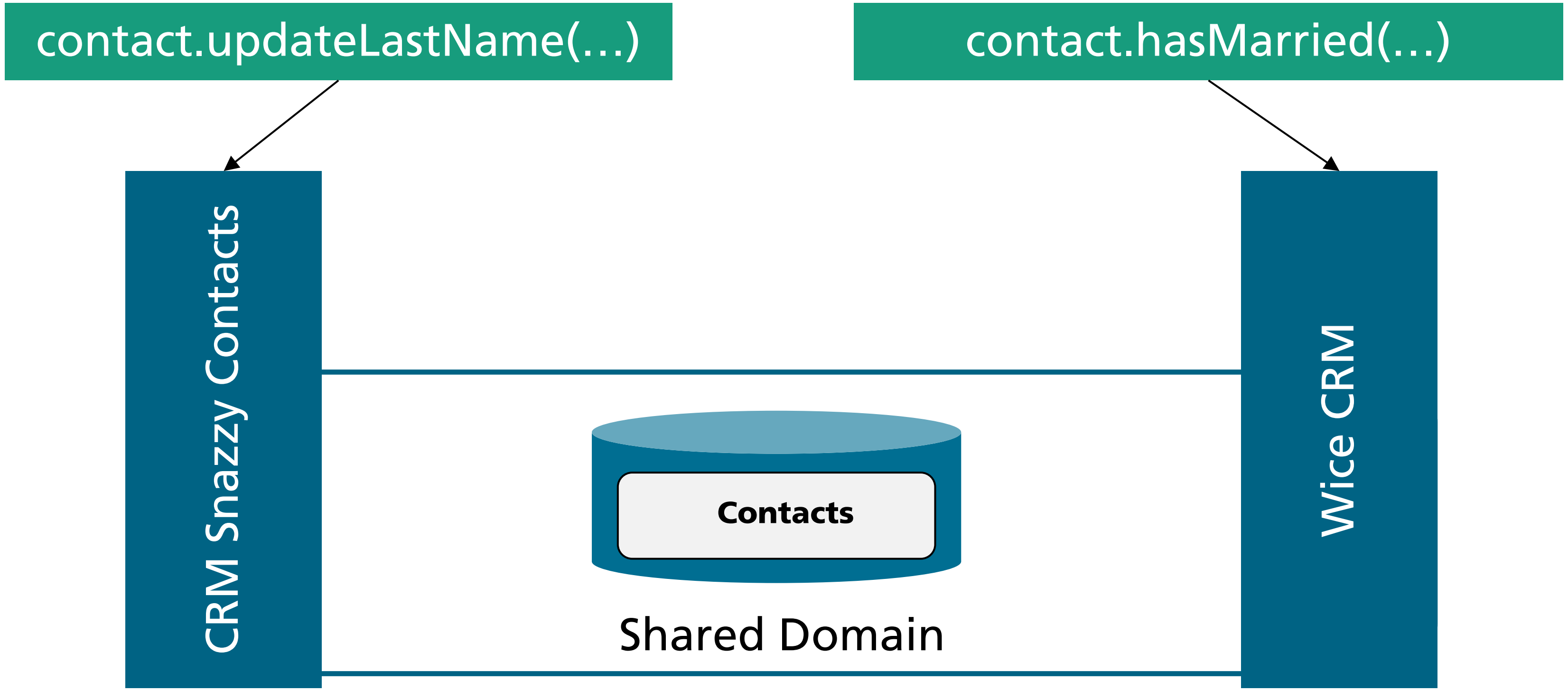
- Compatible Operations



- Execute in any order & preserve happens-before relationships

Conflict Examples OIH

- Equivalent operations: Syntactic resolution with winning operation based on timestamp and/or leading system

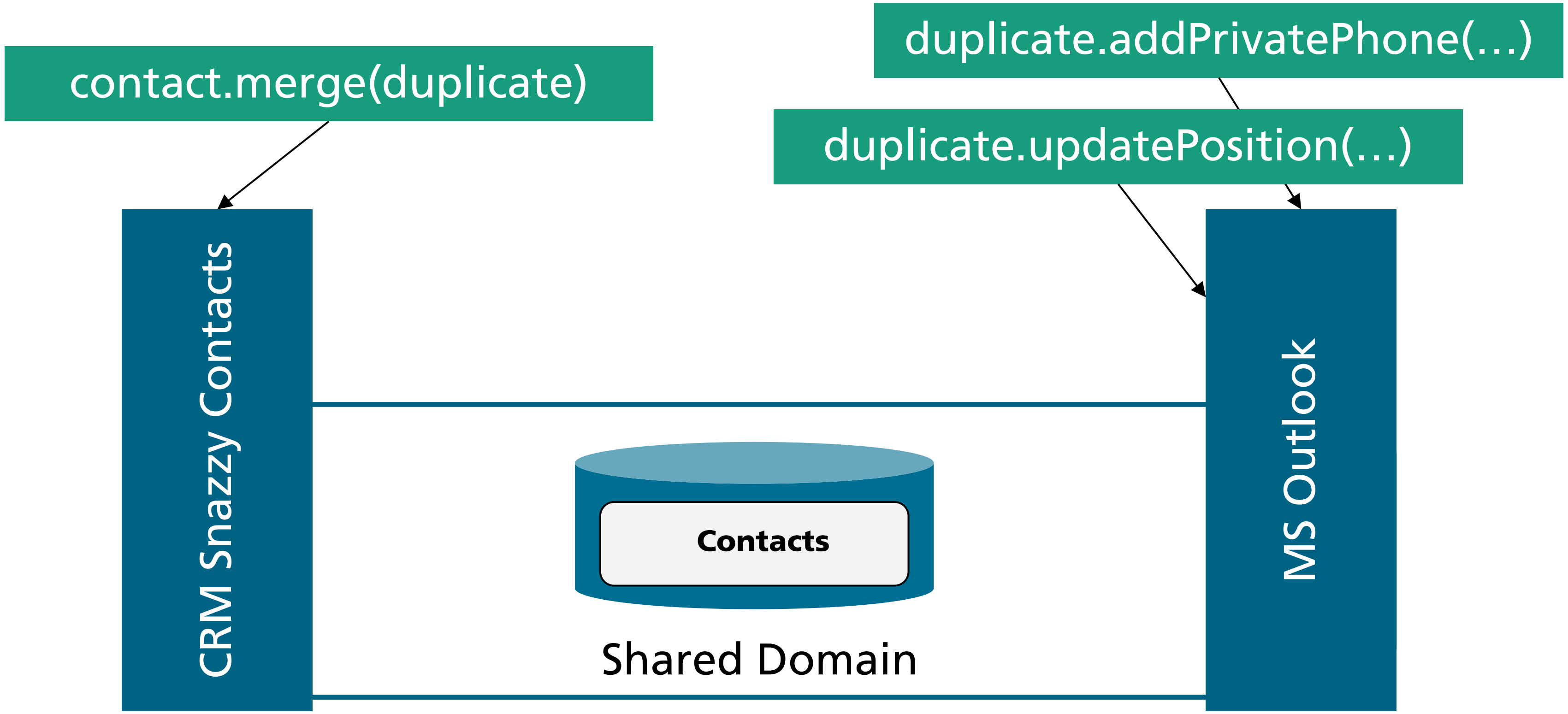


- All conflicting operations are executed. Winners are executed last. Happens-before relationships have to be preserved.

© Fraunhofer IESE

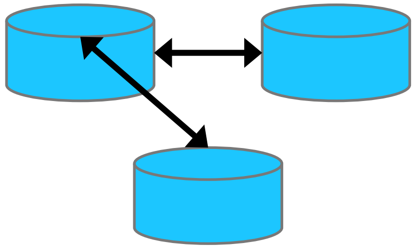
Conflict Examples OIH

- Incompatible operations: semantic resolution with conflict resolver



- Feasible implementation of conflict resolver: execute any updates on the duplicate first and then execute the merge

5. Consenus



- E.g. Paxos (might be required depending on other design decisions)
  - E.g. non-deterministic ordering of update operations

© Fraunhofer IESE

General Design Decisons

- **Overall Consistency Guarantees** (What are the desired Consistency Guarantees? How are they achieved?)

© Fraunhofer IESE