READY: Completeness is in the Eye of the Beholder

B. Chandramouli, J. Gehrke, J. Goldstein, M. Hofmann,
D. Kossmann, J. Levandoski, R. Marroquin, W. Xin

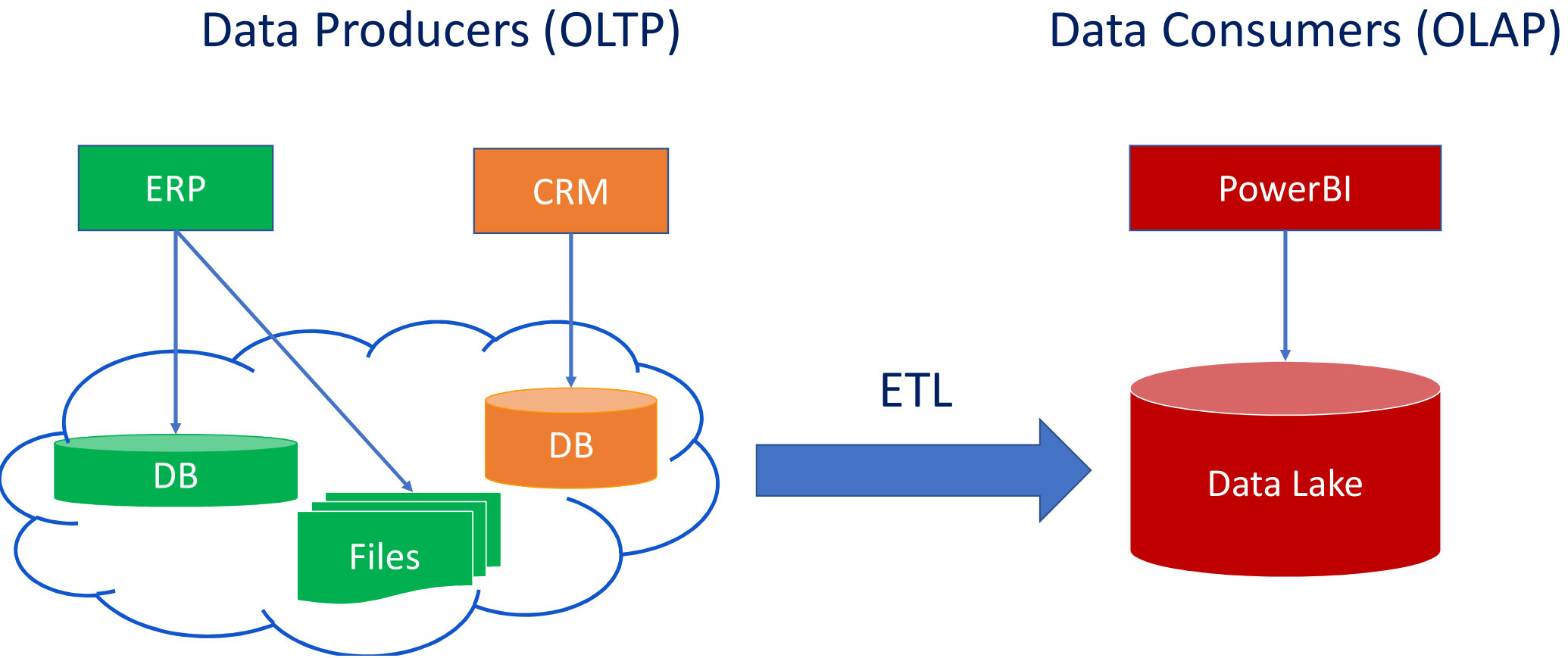
ETH Zurich, Facebook, Microsoft

Observations

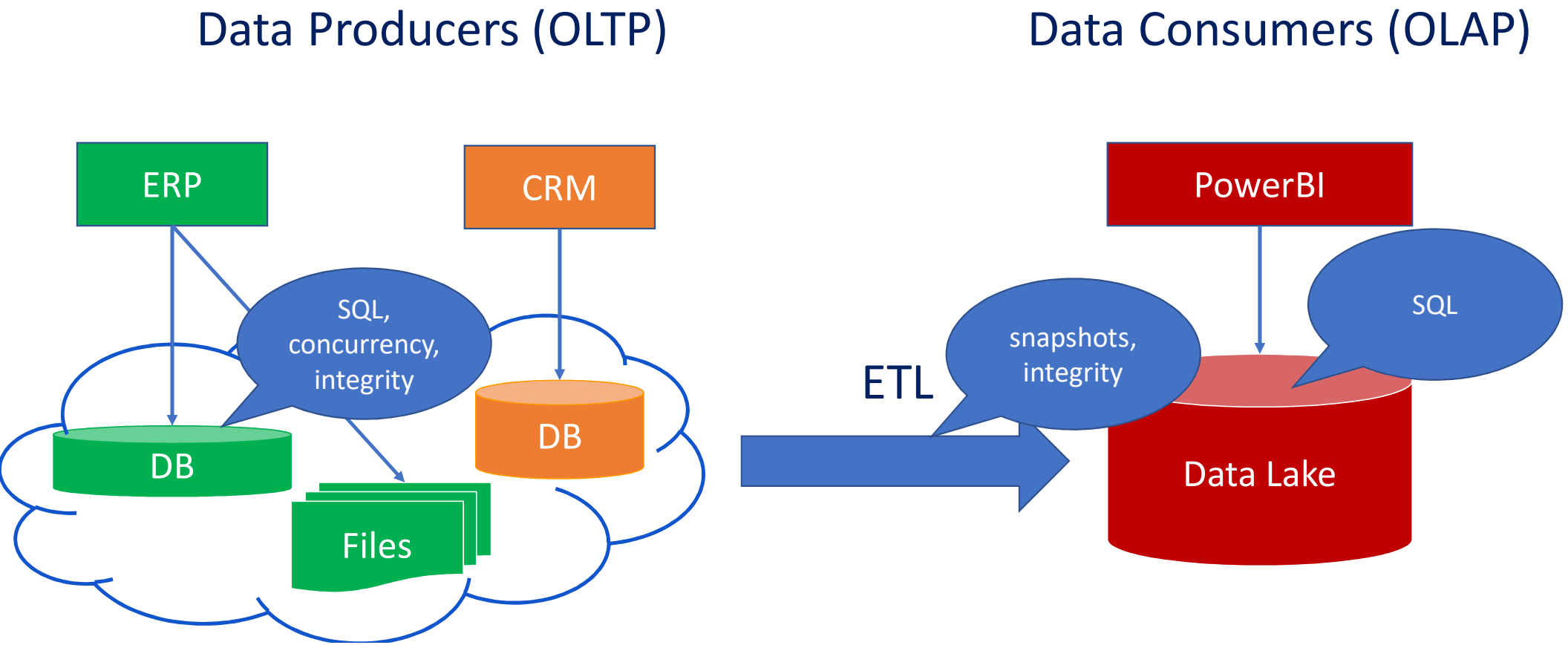
- Observation 1: We **produce** data **in silos** (OLTP databases)
  - rich update functionality (SQL)
  - transactions (concurrency & durability)
  - **integrity constraints in silo**
  - *principle: maximum control and isolation for high data quality*

- Observation 2: We **consume** data **across silos** (analytics in data lake)
  - rich query functionality (SQL)
  - snapshots across silos
  - **integrity constraints across silos**
  - *principle: the more data, the merrier*

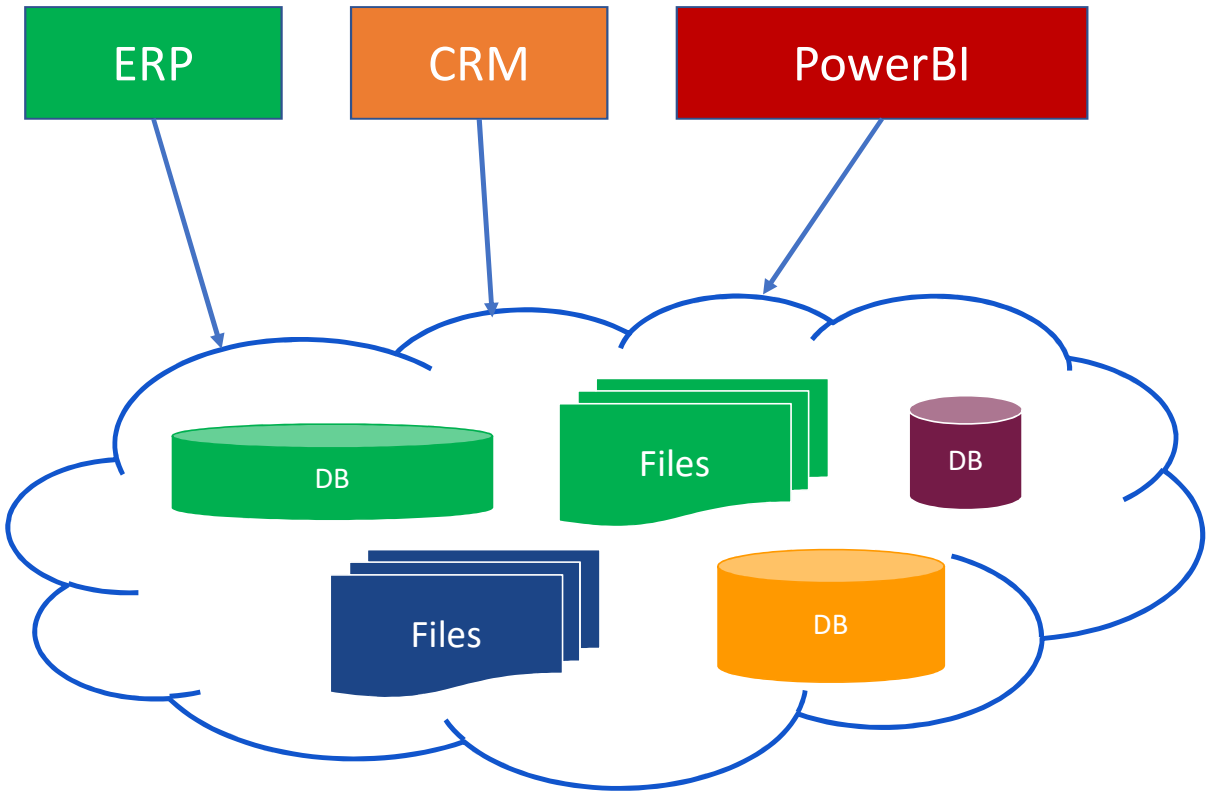
ETL: Implement Producer / Consumer Pipeline



ETL: Implement Producer / Consumer Pipeline

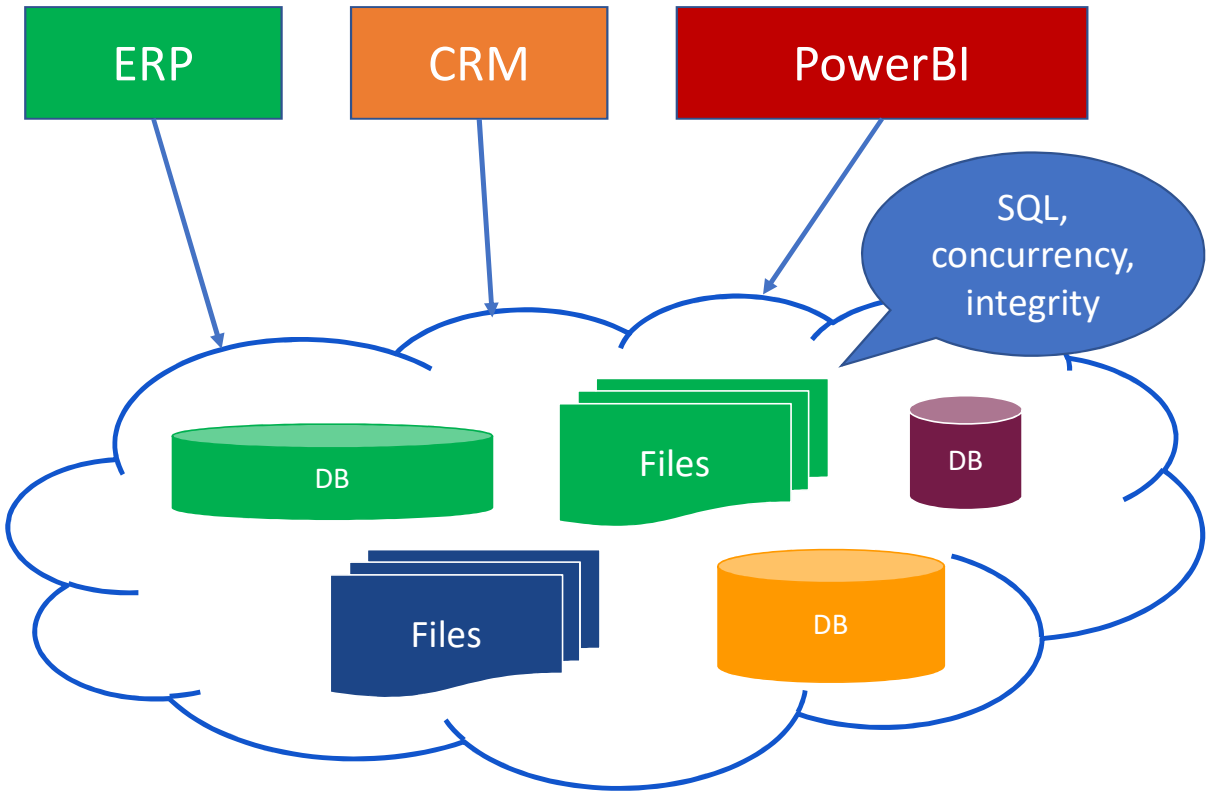


What if we produce data in the data lake…?



- One (logical) copy of data
  - lower cost to move data
  - higher freshness
- One (logical) system
  - higher agility & productivity
  - lower cost (optimization)

What if we produce data in the data lake…?

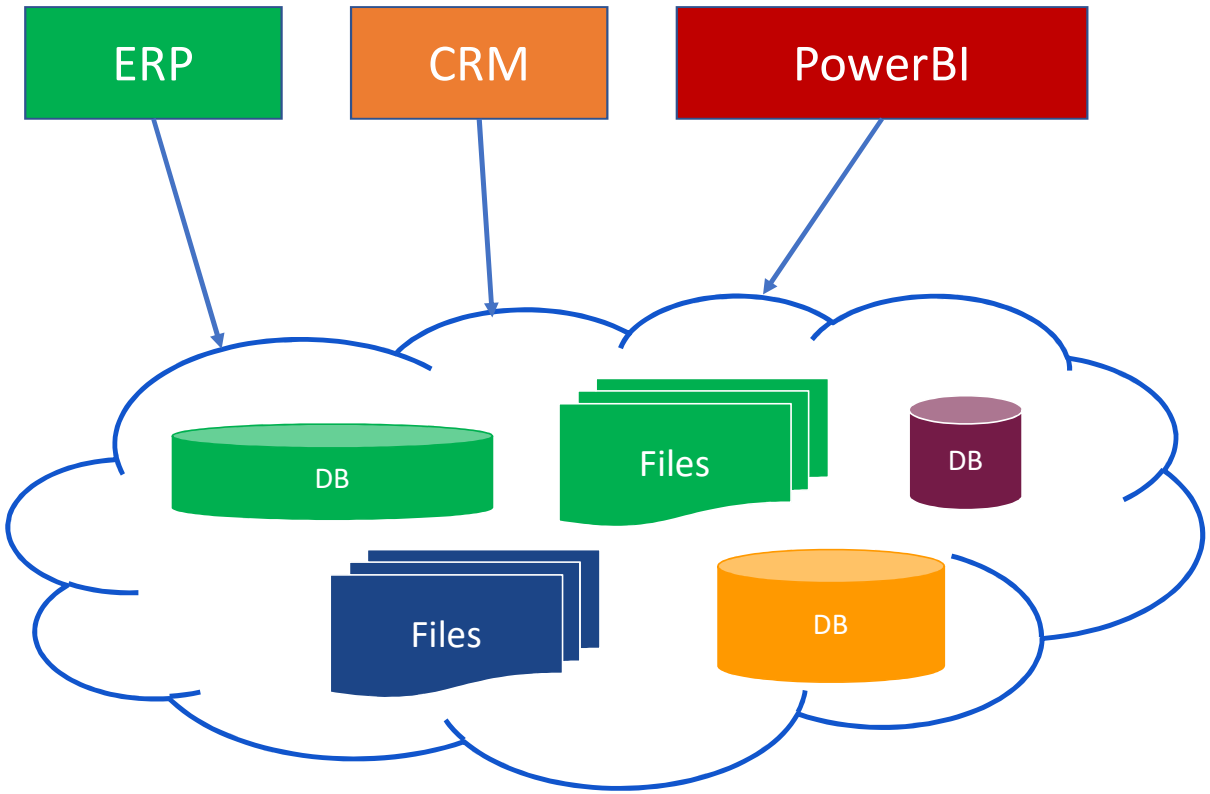


*We need to add **integrity constraints** to data lake!*

Agenda

- READY: Basic Concepts
- READY 1.0: Implementation (see paper)
- Experiments & Results

READY Goals and „CAP Theorem“



- Custom Integrity
  - every app defines its own set of integrity constraints
- Sharing
  - one (logical) copy of data
- Decoupling
  - No application blocked by constraints of another app
- Can only have two of three!

READY Goals

- Custom Integrity
  - every producer and every consumer defines own set of constraints
- Sharing
  - there is only one logical copy of the data
- Decoupling
  - producers are not blocked by consumers
  - consumers are not blocked by other consumers

- „CAP Theorem“: Easy to achieve two of these three goals
  - DW + ETL achieves „custom integrity“ and „decoupling“ goals. But, not „sharing“.
  - Data Lake achieves „sharing“ and „decoupling“. But, not „custom integrity“.

Example: Why is it difficult?

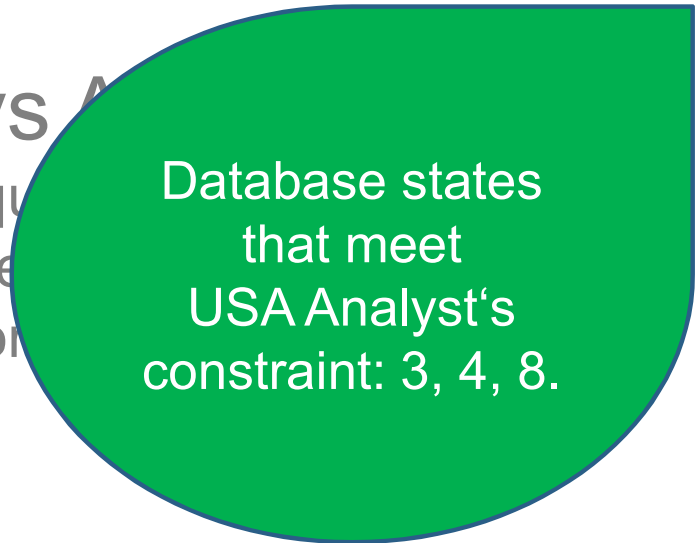
- USA Analyst
  - queries on USA orders
  - report only when all USA orders have shipped
- Toys Analyst
  - queries on Toys orders
  - report only when all Toys orders have shipped

Timeline of Order Process.

1. Enter(1, car, USA)
2. Enter(2, ball, Germany)
3. Ship(1)
4. Enter(3, ball, Germany)
5. Enter(4, car, USA)
6. Ship(2, 3)
7. Enter(5, ball, Germany)
8. Ship(4)

Example: Why is it difficult?

- USA Analyst
  - queries on USA orders
  - report only when all USA orders have shipped
- Toys A
  - qu
  - re
    or



Timeline of Order Proc.

1. Enter(1, car, USA)
2. Enter(2, ball, Germany)
3. **Ship(1)**
4. **Enter(3, ball, Germany)**
5. Enter(4, car, USA)
6. Ship(2, 3)
7. Enter(5, ball, Germany)
8. **Ship(4)**

Example: Why is it difficult?

- USA
- que
- rep
  orders

Database States that meet Toys Analyst's constraints: 1, 6

- Toys Analyst
- queries on Toys orders
- report only when all Toys orders have shipped

Timeline of Order Proc.

1. **Enter(1, car, USA)**
2. Enter(2, ball, Germany)
3. Ship(1)
4. Enter(3, ball, Germany)
5. Enter(4, car, USA)
6. **Ship(2, 3)**
7. Enter(5, ball, Germany)
8. Ship(4)

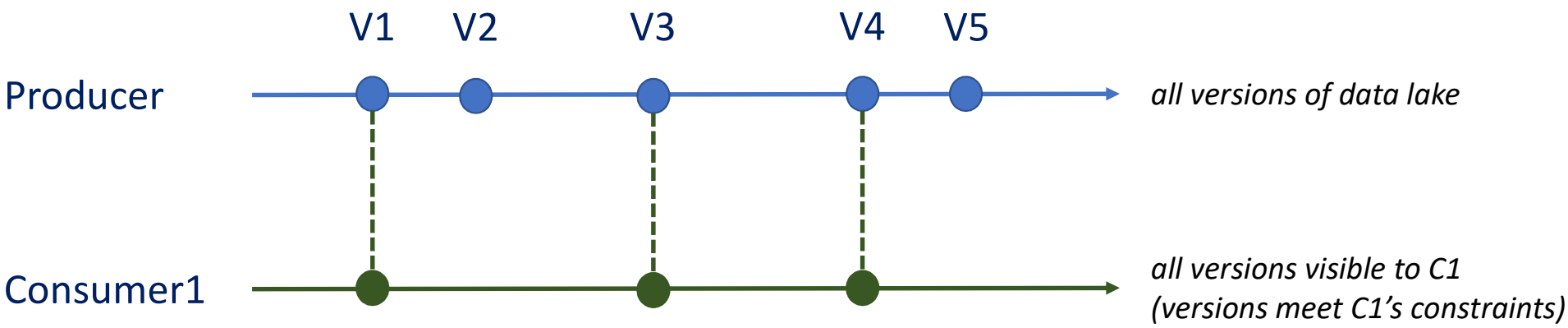
READY Approach

- Each update creates a new ***version*** of the ***data lake***
  - efficient implementation of update batches via „delta materialization“ (see paper)
- All applications run in a ***sandbox***
  - sandbox defines a set of integrity constraints
  - (sandbox also determines concurrency control policy)
- Consumers: sandbox controls which ***versions are visible***
  - non-compliant versions are not visible to consumer, but possibly to other consumers
  - query annotation determines which visible version to use (next, last, continuous)
- Producers: sandbox controls which ***versions are legal***
  - non-compliance results in abort of transactions, just as in regular RDBMS
  - In READY 1.0, there is only one producer sandbox

READY Approach: Temporal Data Lake

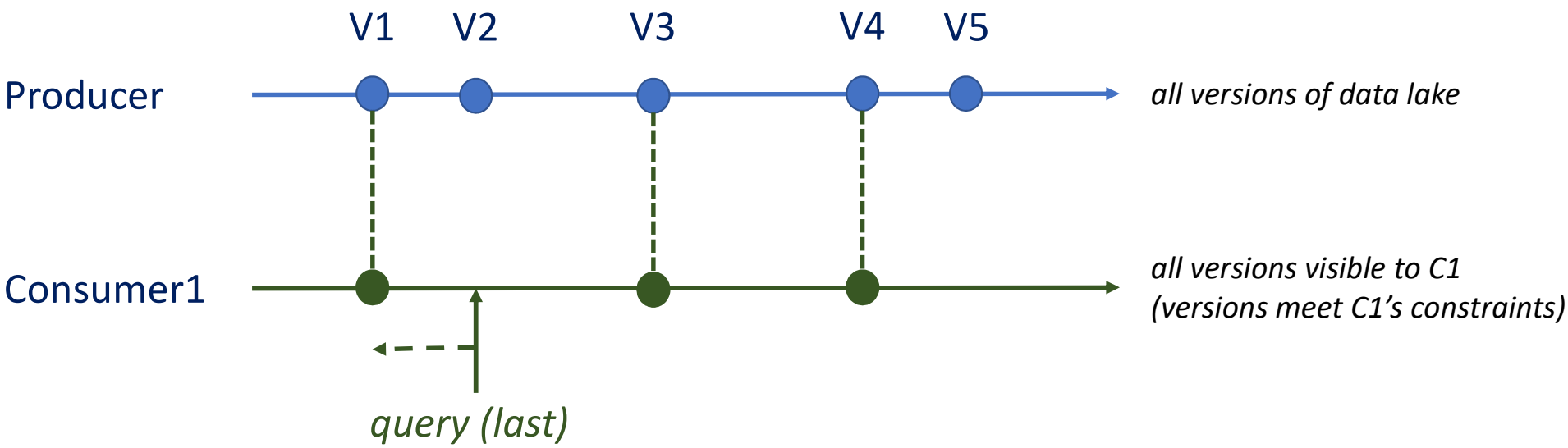
- Producer generates new versions independent of consumers
  - Only requirement: each version meets producers integrity constraints

READY Approach: Visibility of Versions



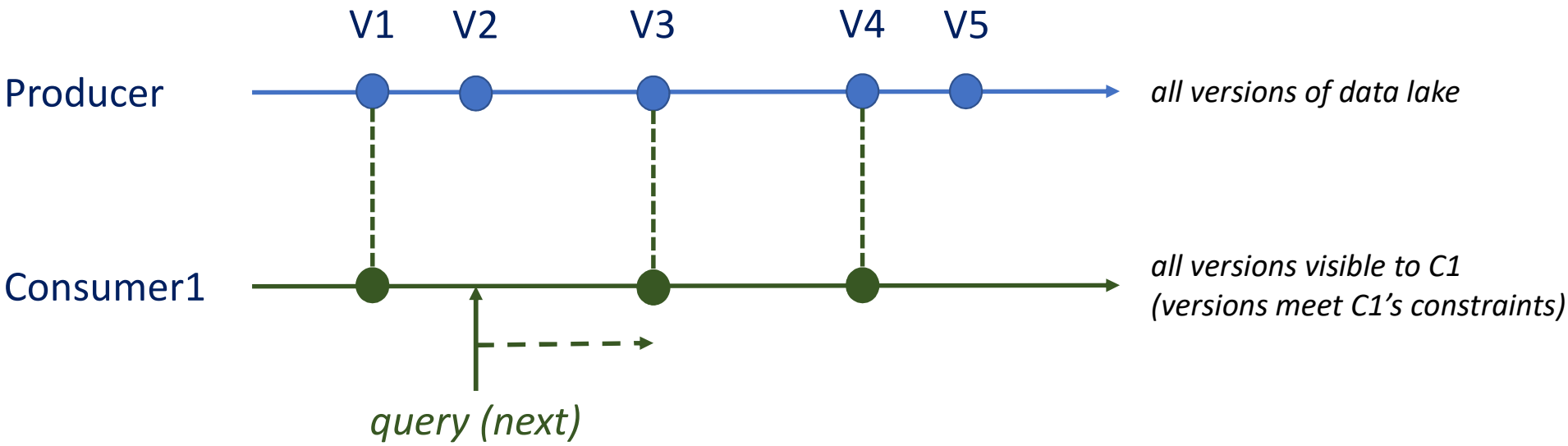
- Consumer1 only sees those versions that meets its constraints
  - does not block producer if producer creates version that is not compliant

READY Approach: Query Annotations



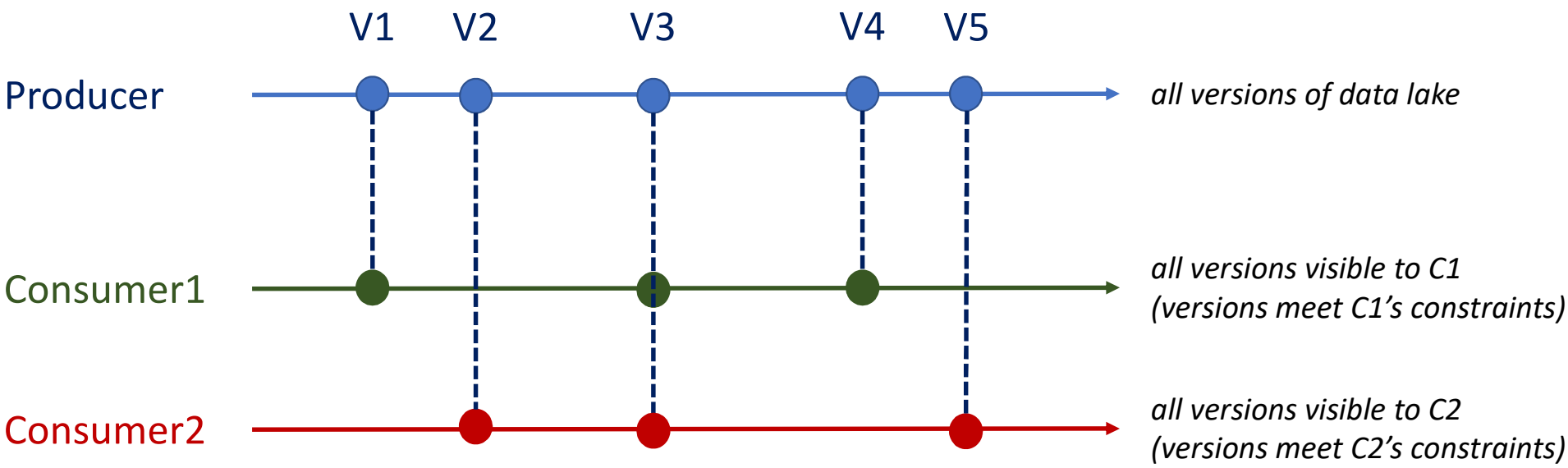
- Consumer1 only sees those versions that meets its constraints
  - *Last:* use latest visible version to process query

READY Approach: Query Annotations



- Consumer1 only sees those versions that meets its constraints
  - *Last:* use latest visible version to process query
  - *Next:* wait for next visible version to process query

READY Approach: Decoupling



- Consumer2 only sees those versions that meet its constraints
  - may or may not overlap with C1 or any other consumer

Related Work and Concepts

- Views
  - Pro: sandbox is like a view that filters the right version of a record
  - Con: sandboxes are updateable (producers run in sandboxes)
  - Con: simpler view definition
- Materialized Views, Incremental Maintenance & Streaming
  - a great way to implement sandboxes
- DataHub, Version Control Systems (git), Temporal Databases
  - right way to think about data lake and visibility of versions

Agenda

- READY: Basic Concepts
- **READY 1.0: Implementation**
- Experiments & Results

READY 1.0

- Sandboxes
  - Each transaction / query runs in a sandbox
  - Sandboxes Define:
    - ***which*** snapshots of data lake are visible
    - ***which*** business objects are visible

READY 1.0: Sandbox Syntax

```
CREATE SANDBOX sandboxName ( argname argtype )*
[ FOR UPDATES ]
[ WHEN predicate ]
[ WITH ( relationName: predicate )* ];
```

READY 1.0: Sandbox Example

```
CREATE SANDBOX noOpenOrderSandbox()
WHEN
 NOT EXISTS (SELECT * FROM Order o
          WHERE o.o_status = "Open")
```

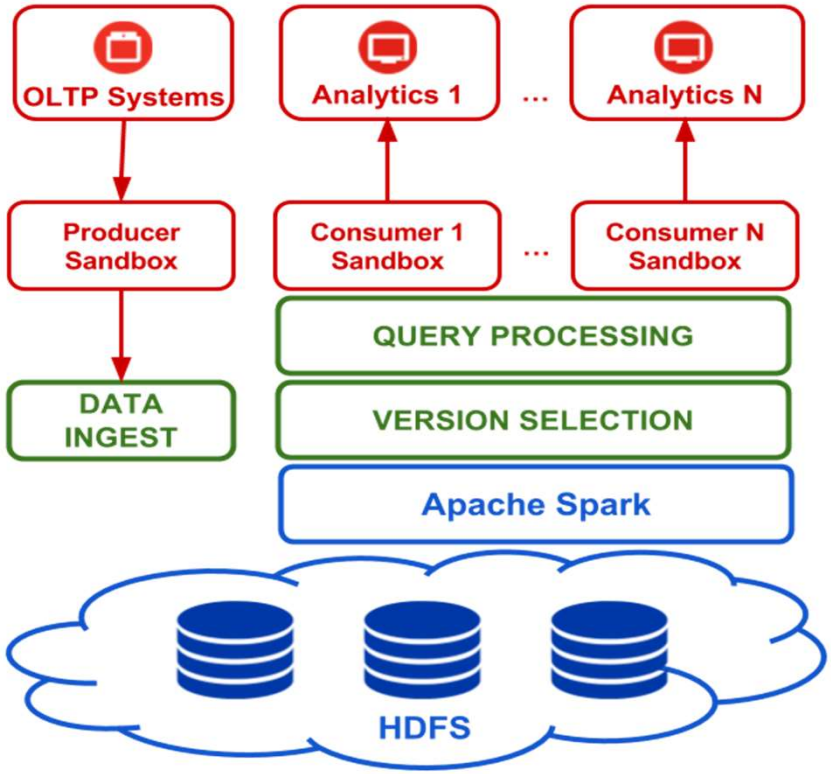
READY 1.0: Parameterized Sandbox

```
CREATE SANDBOX completeByNation(:nationId INT)
WHEN
 FORALL (SELECT o.status as s
           FROM Order o, Customer c
          WHERE o.o_custkey == c.c_custkey
             AND c.c_nationkey == :nationId)
 SATISFY s = "Verified"
```

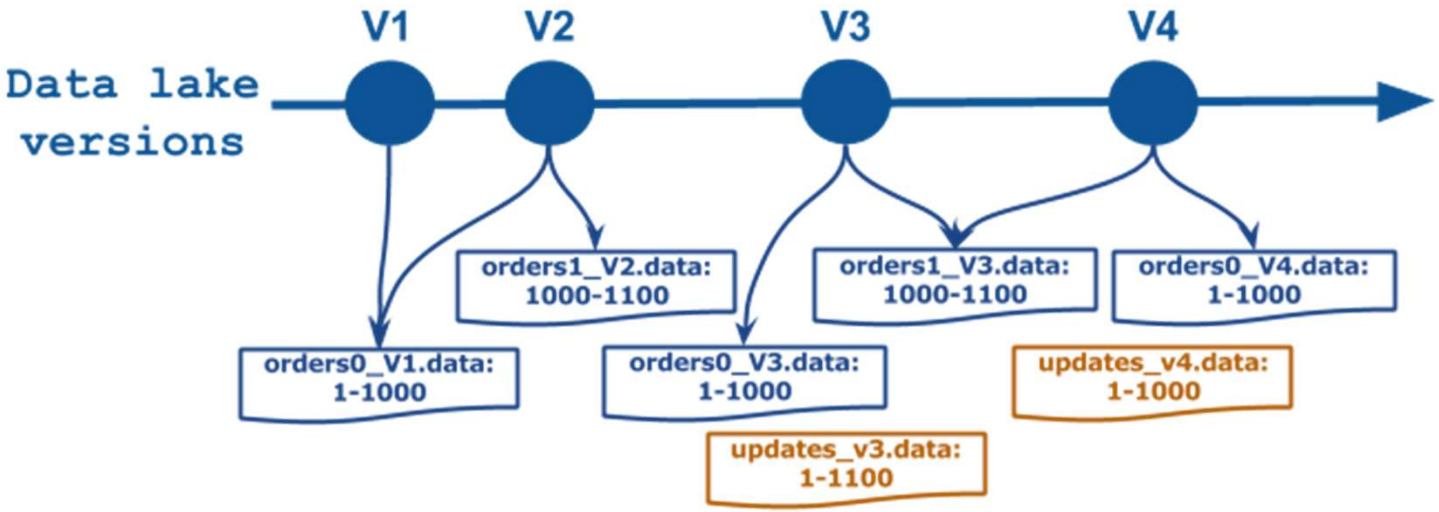
READY 1.0: Sandbox Usage

```
BEGIN USING completeByNation("Germany")
 NEXT;
    SELECT c.name, count(*)
       FROM Order o, Customer c
       WHERE o.o_custkey = c.c_custkey
        AND c.c_nationkey = "Germany"
       GROUP BY c.name
COMMIT;
```

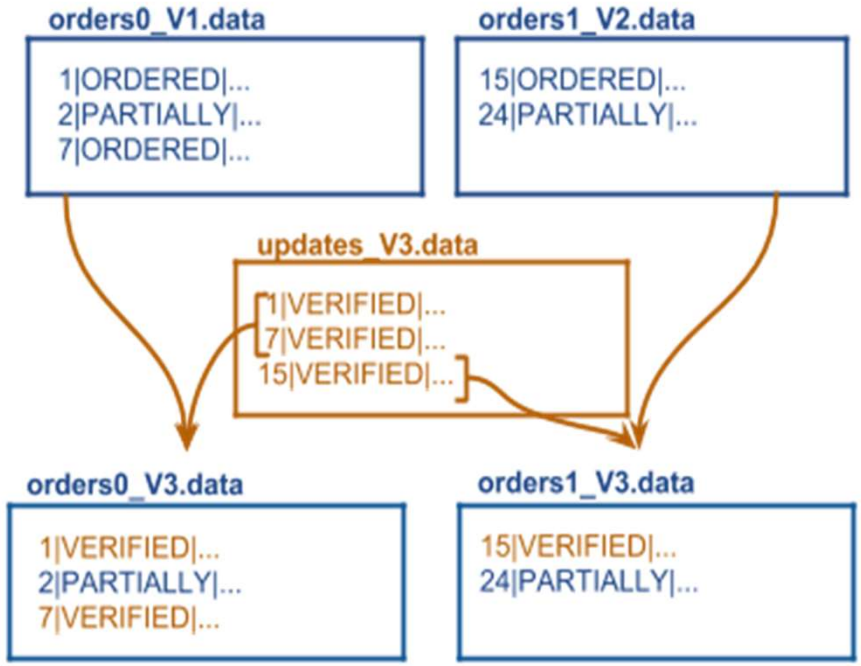
READY 1.0: System Overview



READY 1.0: Version Management



READY 1.0: Delta Materialization



READY 1.0: Version Selection (integrity checks)

- Batch processing whenever needed
- Incremental processing with every new version of data lake
  - Constraint checks can be expressed as tuple counting

```
Exists: count(S) > 0
FORALL: count(S) = count(p(S))
```

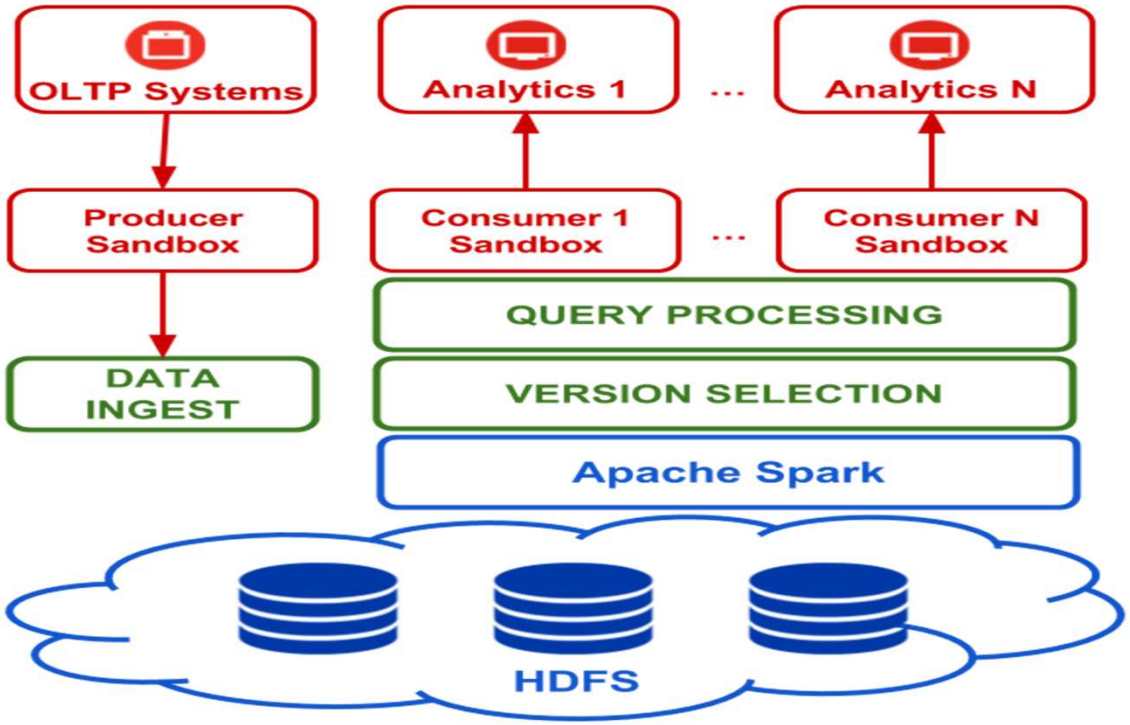
  - Transform constraint checking into:

```
count(post(S)) = count(pre(S)) + count(delta(S))
```

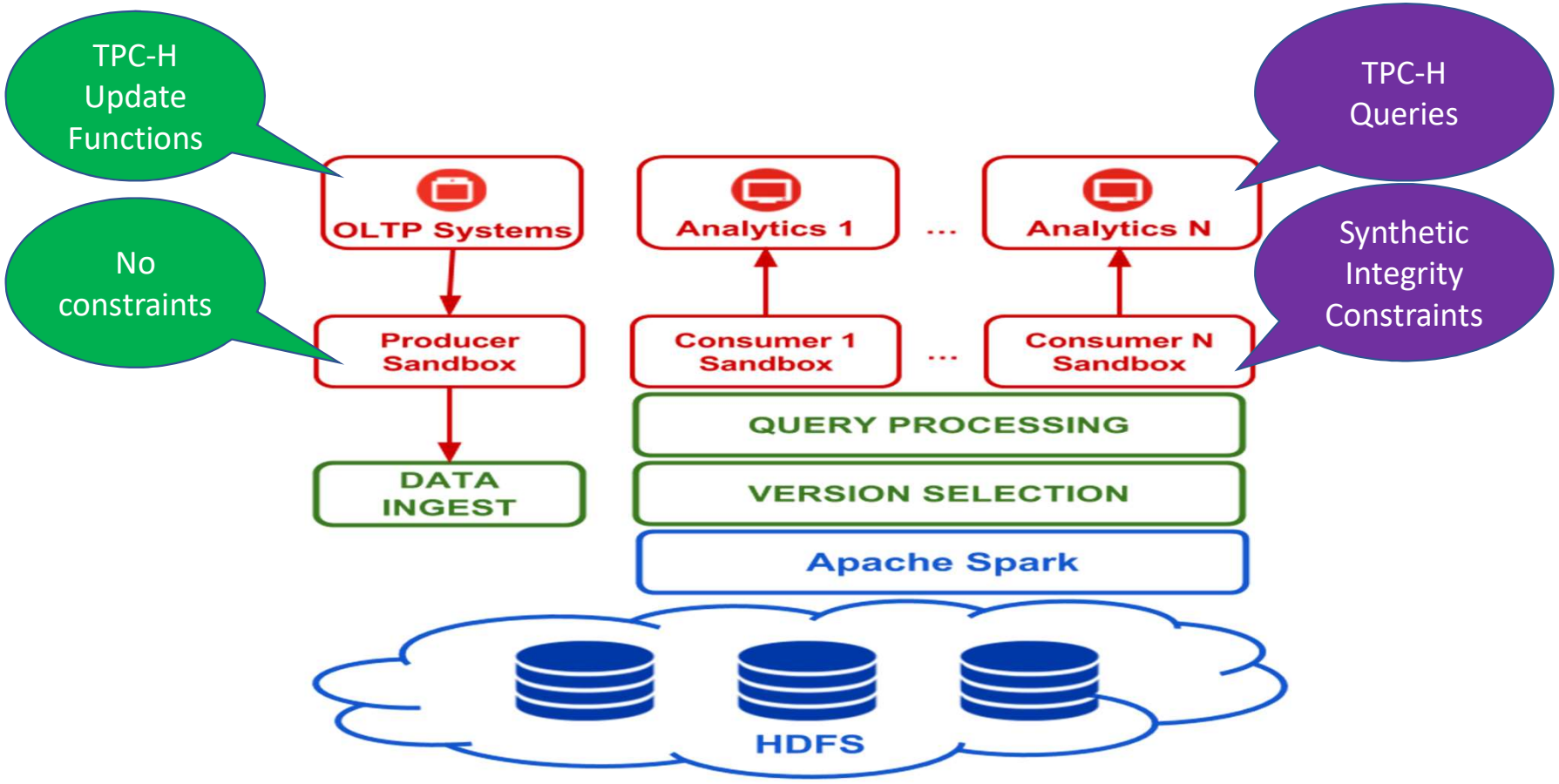
Agenda

- READY: Basic Concepts
- READY 1.0: Implementation (see paper)
- **Experiments & Results**

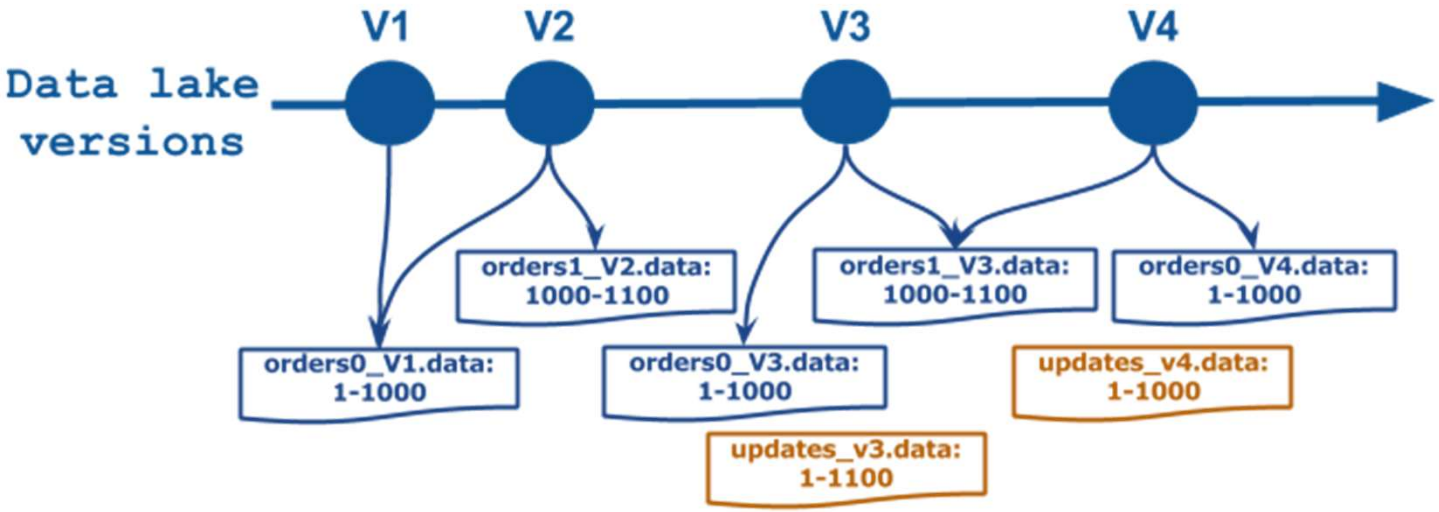
READY Prototype (Runtime System)



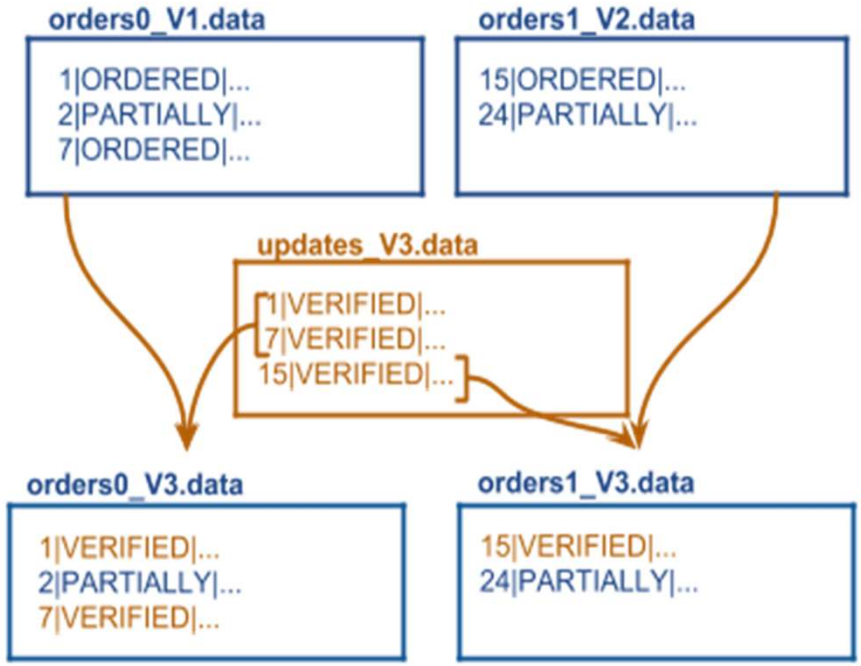
TPC-H on READY



READY 1.0: Version Management



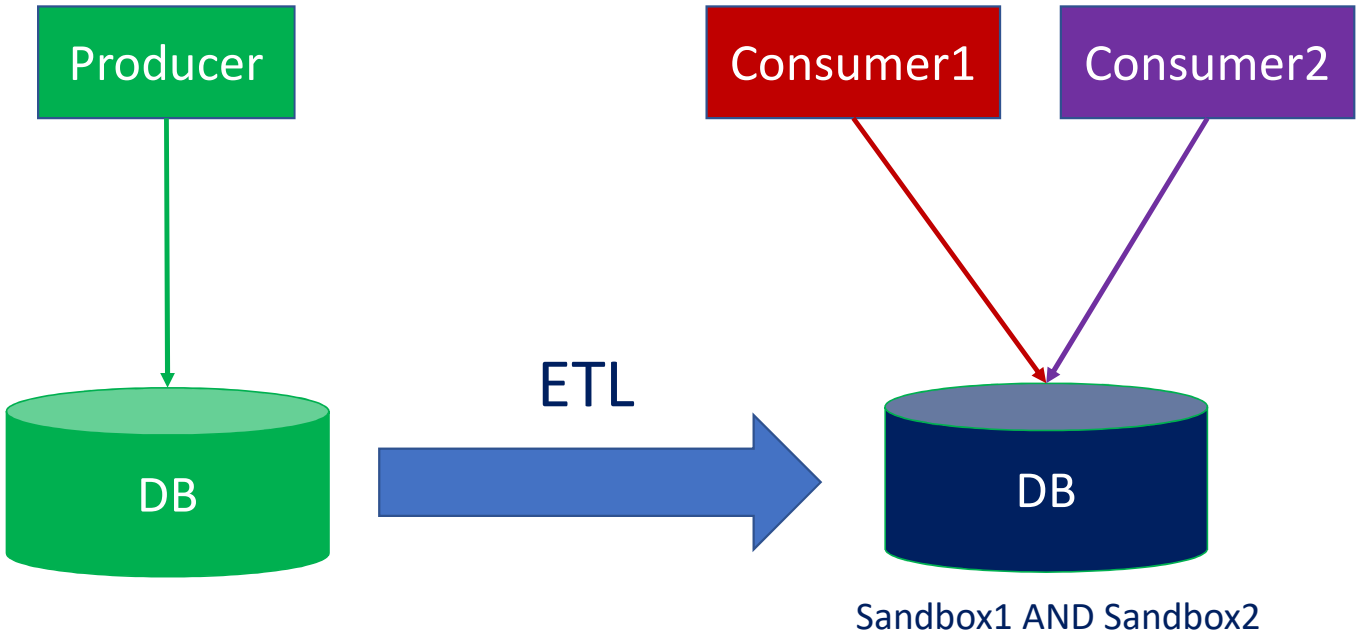
READY 1.0: Delta Materialization



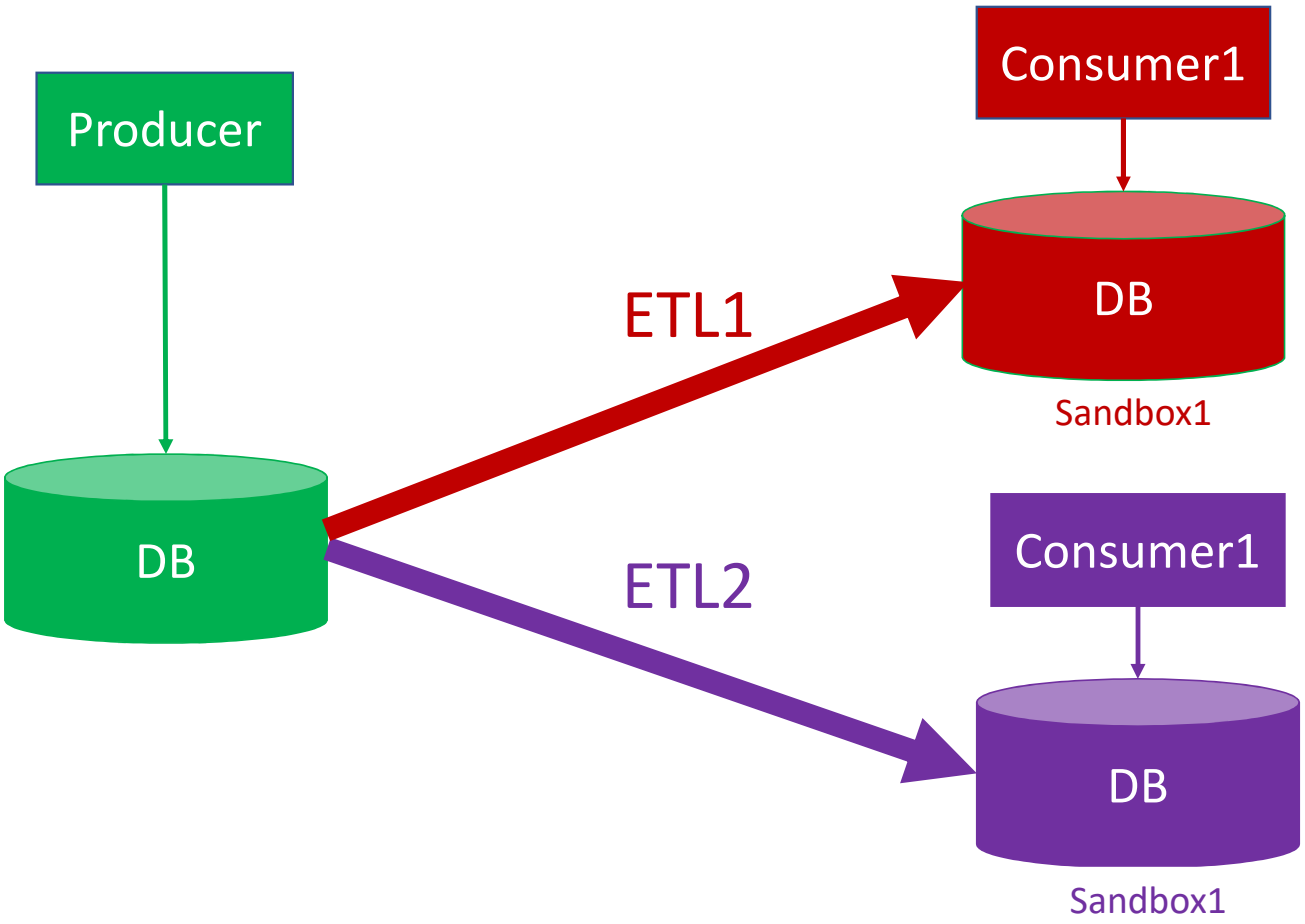
Experimental Set-Up and Goals

- Experiment 1: Measure Cost
  - study „sharing“ goal
  - vary number of applications (sandboxes) with synthetic integrity constraints
- Experiment 2: Measure Data Freshness
  - study „decoupling“ goal
  - vary number of applications (sandboxes) with synthetic integrity constraints
- Baselines for all experiments
  - Global: data warehouse in which all consumers run on single data mart
    - (conjunction of all sandboxes)
  - Personal: one data mart for each consumer

Baseline 1: Global Data Warehouse

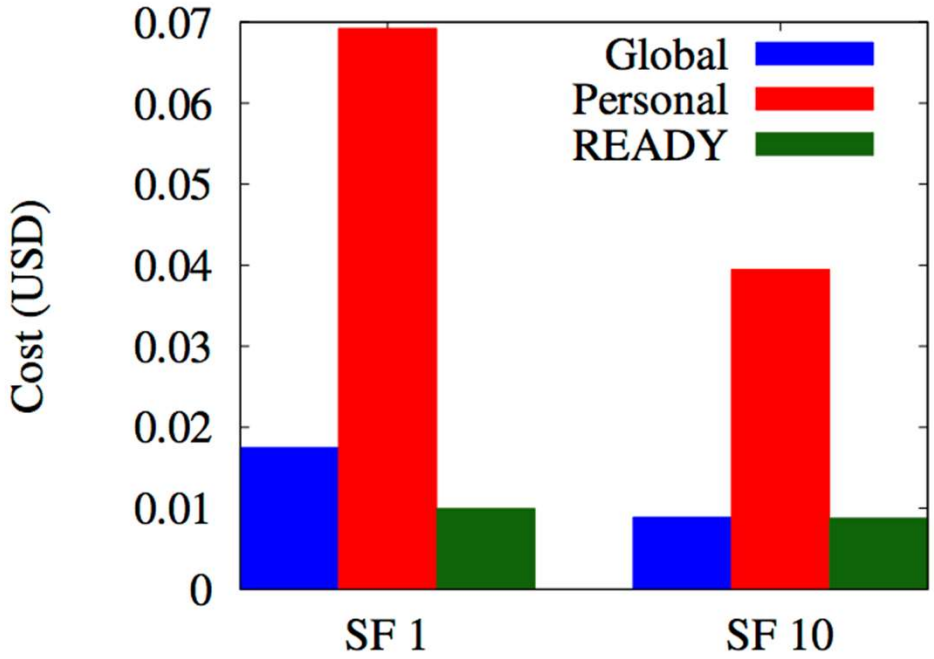


Baseline 2: Personal Data Warehouse



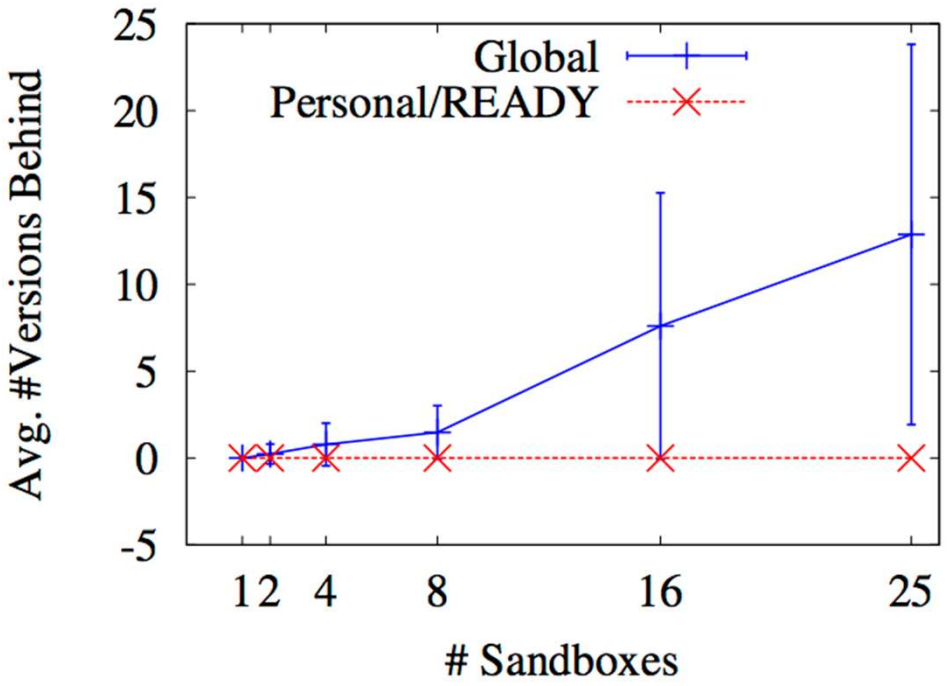
Exp 1: Cost of TPC-H Update Functions

- Cost of Delta Materialization
  - 8 Sandboxes, vary TPC-H Scaling Factor

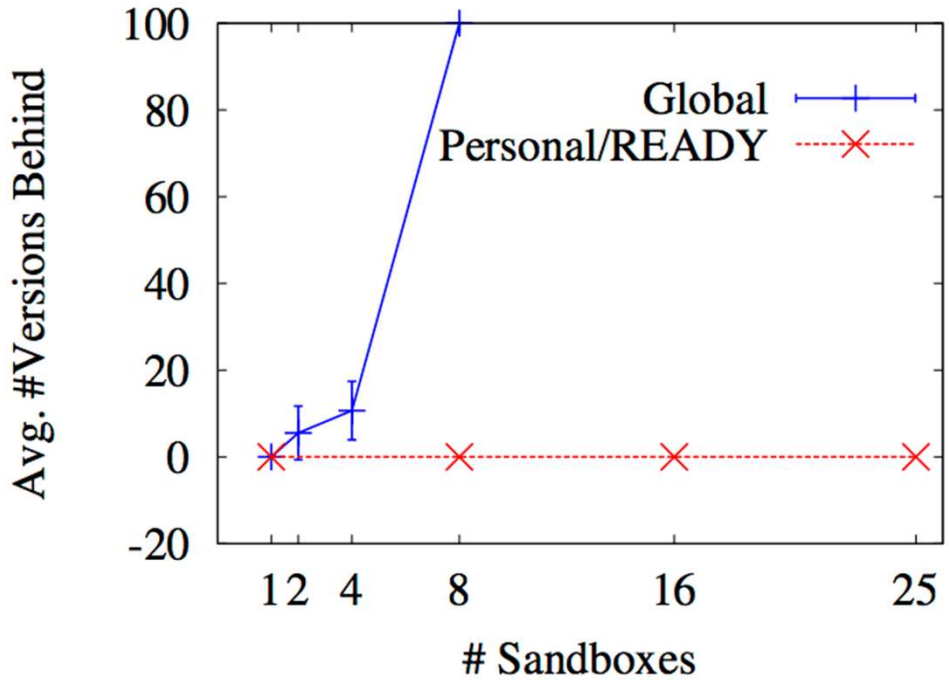


Exp 2: Data Freshness (vary SF, sandboxes)

SF1



SF10



Conclusion & Future Work

- Thought experiment: What if DBMS supports multiple sets of IC?
  - semantics make use of all classic DB concepts: snapshots, views, temporal
  - nice implementation on top of Spark possible
- Future Work
  - Generalize READY model: multiple producer sandboxes (think git & branching)
    - need a way to „merge“ branches