Coordination Avoidance
in Distributed Databases

2015

PhD thesis by Peter David Bailis

presented by Slavik Derevyanko
2016/10/24
UNIVERSITY OF WATERLOO

Outline

- Coordination avoidance and Invariant Confluence principle
- Examples of Invariant Confluence principle application
- Read-atomic multi-partition transactions
- Conclusions

Transaction serializability

- Transactions - groups of multiple operations over multiple data items
- Traditional way to deal with concurrent transactions - serializability: a database providing serializability guarantees that the result of executing the transactions is equivalent to some serial execution of the transactions
    - Convenient for programmers - no need to reason about concurrency
    - Inconvenient for databases - serializability requires coordination
- Reasons to avoid coordination - allows for greater availability of a distributed database, lower access latency and greater scalability

Thesis statement

- Key question posed by Bailis - when is it necessary to use coordination to achieve conflict-free parallel execution, and when is it possible to forego coordination without compromising the safety of parallel transactions

- *Thesis Statement: Many semantic requirements of database-backed applications can be efficiently enforced without coordination, thus improving scalability, latency, and availability.*

Application side: data invariants

- Bailis proposal: instead of reasoning about data consistency on the level of read and write operations (transactions), **consider what coordination is actually required by a given application**
- Make applications define data invariants and explicitly specify what correctness means to these applications
- Example: “each employee record is linked to a department record”
- Consider if **application requires coordination**, by **taking into account both** application invariants **and** the nature of data transformation in transaction

Invariant confluence test

- Invariant confluence determines whether the result of executing operations on independent copies of data can be combined (or "merged") into a single, coherent (i.e., convergent) copy of database state.
- Given **a set of operations**, **a safety property** that we wish to maintain over all copies of database state, **and a merge function**, invariant confluence tells us **whether coordination-free execution is possible**.

Application of Invariant Confluence principle

UNIVERSITY OF WATERLOO

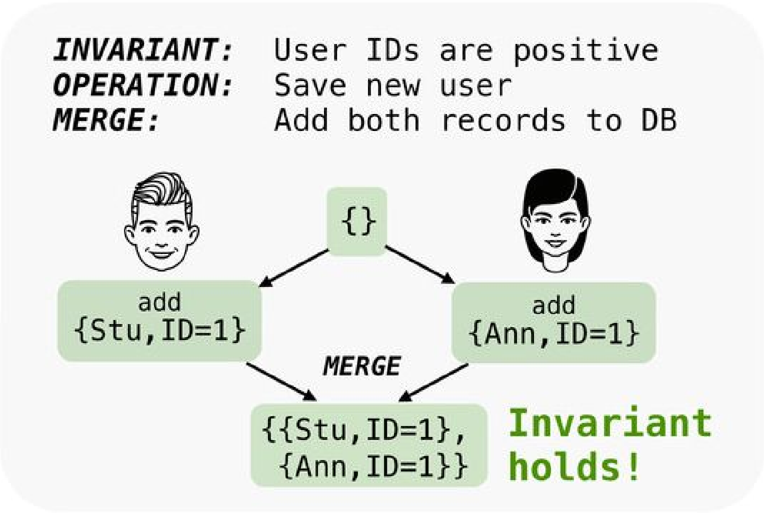
I-Confluence proof construction

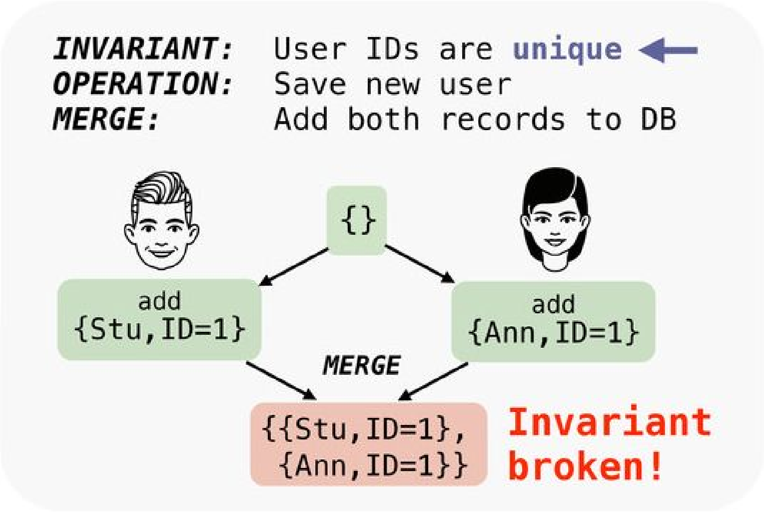
- To show a set of transactions **are not invariant confluent** with respect to an invariant I - **use proof by counterexample:** present two I-T-reachable states with a common ancestor that, when merged, are not I-valid.

- To show a set of transactions **are invariant confluent** with respect to an invariant I **use proof by contradiction**: show that if a state S is not I-valid, merging two I-T-reachable states S1 and S2 with a common ancestor state to produce S implies either one or both of S1 or S2 must not be I-valid.

Invariant confluence test example

- Key question: can invariants be violated by merging independent operations?





I-Confluence applied to common SQL operations

Invariant	Operation	invariant confluent?	Proof #
Attribute Equality	Any	Yes	1
Attribute Inequality	Any	Yes	2
Uniqueness	Choose specific value	No	3
Uniqueness	Choose some value	Yes	4
AUTO_INCREMENT	Insert	No	5
Foreign Key	Insert	Yes	6
Foreign Key	Delete	No	7
Foreign Key	Cascading Delete	Yes	8
Secondary Indexing	Update	Yes	9
Materialized Views	Update	Yes	10
>	Increment [Counter]	Yes	11
<	Increment [Counter]	No	12
>	Decrement [Counter]	No	13
<	Decrement [Counter]	Yes	14
[NOT] CONTAINS	Any [Set, List, Map]	Yes	15, 16
SIZE=	Mutation [Set, List, Map]	No	17

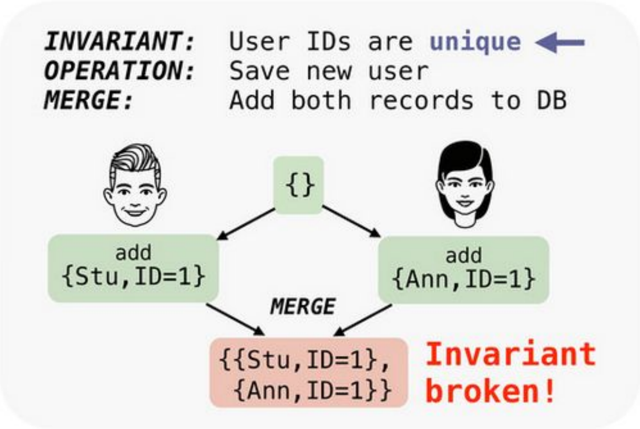
Table 6.1: Example SQL (top) and ADT invariant confluence along with references to formal proofs in Section 6.2.

I-Confluence example 1

Invariant	Operation	invariant confluent?	Proof #
Uniqueness	Choose specific value	No	3
Uniqueness	Choose some value	Yes	4

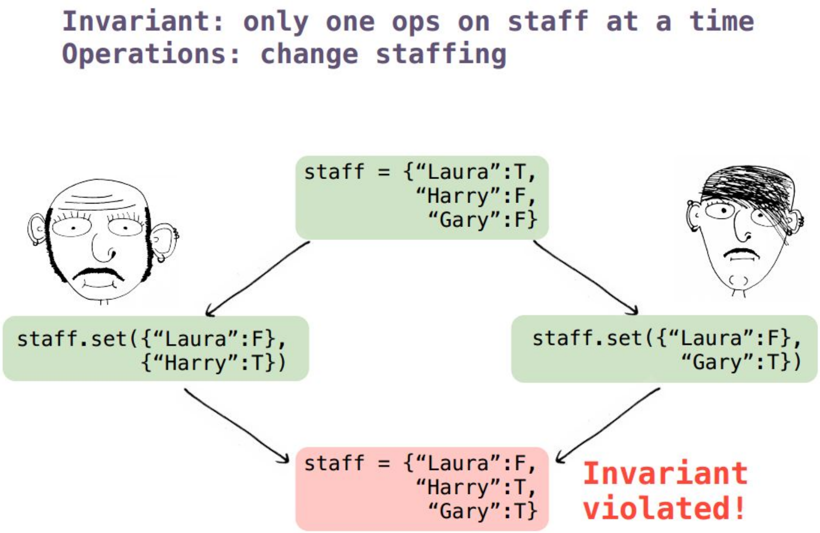
- Claim: **common uniqueness invariants aren't I-Confluent** (e.g., PRIMARY KEY and UNIQUE constraints).
- Example invariant: user IDs must be unique
- However, **reads and deletions are both invariant confluent** under uniqueness invariants: **reading and removing items cannot introduce duplicates**
- Case 2: the database chooses unique values on behalf of users. **If replicas assign unique IDs within their respective portion of the ID namespace, then merging locally valid states will also be globally valid**

Proof by counterexample:



I-Confluence example 2

- Claim 9: writing arbitrary values is not invariant confluent with respect to multi-item uniqueness constraints.
- Proof: by counterexample



I-Confluence example 3

Invariant	Operation	invariant confluent?	Proof #
Attribute Equality	Any	Yes	1
Attribute Inequality	Any	Yes	2

- Claim: Attributed equality invariants are i-confluent for any operations
- Example: **every user must have a last name assigned**, marking the LNAME column with a NOT NULL constraint
- **Proof by contradiction**: assume two database states S1 and S2 are each I-T-reachable under per-record inequality invariant I but that I(S1 U S2) is false. Then there must be a **r** ∈ S1 U S2 that violates I (i.e., r has the forbidden value) and such **r** must appear in S1, S2, or both. But, that would imply that one of S1 or S2 is not I-valid under I, a contradiction.

I-Confluence example 4

Invariant	Operation	invariant confluent?	Proof #
AUTO_INCREMENT	Insert	No	5

- Claim 11: Writing arbitrary values are not invariant confluent with respect to sequentiality constraints.
- Proof: by counterexample

Consider the following transactions:

$$T_{1s} := w(x_a = 1);\ \mathtt{commit}$$
$$T_{2s} := w(x_b = 3);\ \mathtt{commit}$$

and the sequentiality constraint on records:

$$I_s(D) = \{\max(r \in D) - \min(r \in D) = |D| + 1\} \vee \{|D| = 0\}$$

Now, I_s *holds over the empty database* ($I_s(\{\}) \rightarrow \mathtt{true}$)*, while inserting sequential new records into independent, empty replicas is also valid:*

$$T_{1s}(\{\}) = \{x_a = 1\},\ I_u(\{x_a = 1\}) \rightarrow \mathtt{true}$$
$$T_{2s}(\{\}) = \{x_b = 3\},\ I_u(\{x_b = 3\}) \rightarrow \mathtt{true}$$

However, merging these states results in invalid state:

$$I_s(\{x_a = 1\} \sqcup \{x_b = 3\} = \{x_a = 1, x_b = 3\}) \rightarrow \mathtt{false}$$

Therefore, $\{T_{1s}, T_{2s}\}$ *is not invariant confluent under* I_s.

UNIVERSITY OF WATERLOO

I-Confluence example 5

Invariant	Operation	invariant confluent?	Proof #
>	Increment [Counter]	Yes	11
<	Increment [Counter]	No	12
>	Decrement [Counter]	No	13
<	Decrement [Counter]	Yes	14

Claim 17 Counter ADT increments are invariant confluent with respect to greater-than constraints .

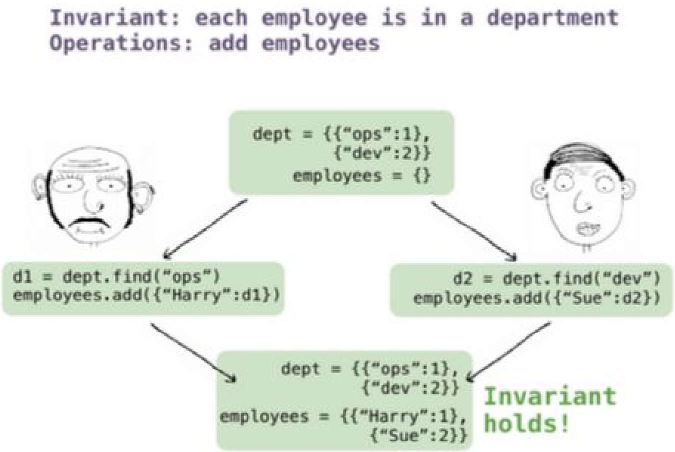
Claim 18 Counter ADT increments are not invariant confluent with respect to less-than constraints .

Claim 19 Counter ADT decrements are not invariant confluent with respect to greater-than constraints .

Claim 20 Counter ADT decrements are invariant confluent with respect to less-than constraints .

I-Confluence example 6: foreign keys

- Claim: **Insertions under foreign key constraints are invariant confluent**
- Proof by contradiction:
  - Invalid state: a record missing a corresponding record on the opposite side of the association
  - *S1* and *S2* - correct states before the merge (no invalid records)
  - *r* - invalid record in merged state S
  - As S1 and S2 are both valid, *r* must have a corresponding foreign key record (f) that “disappeared” during merge. Merge (in the current model) does not remove versions, so this is impossible.
- Arbitrary **deletion/modification** of records **is unsafe**: a user might be added to a department that was concurrently deleted



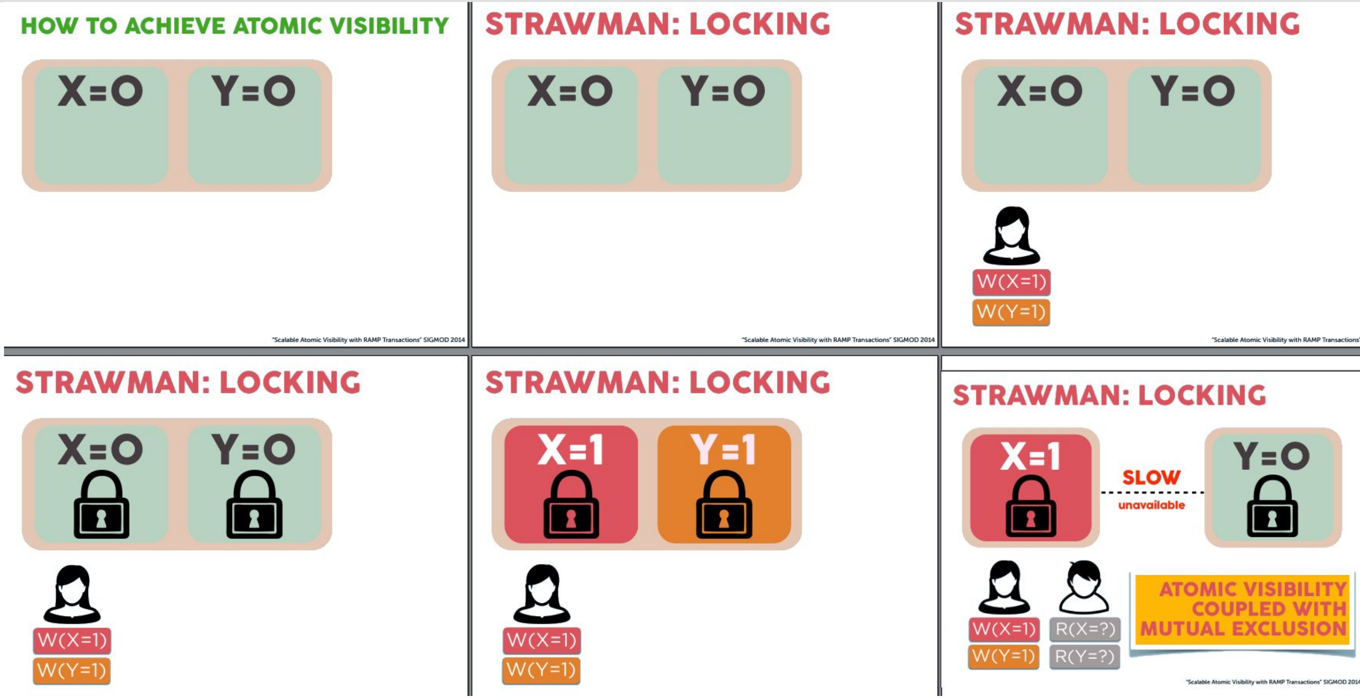
Read-atomic multi-partition transactions

UNIVERSITY OF WATERLOO

Foreign keys updates

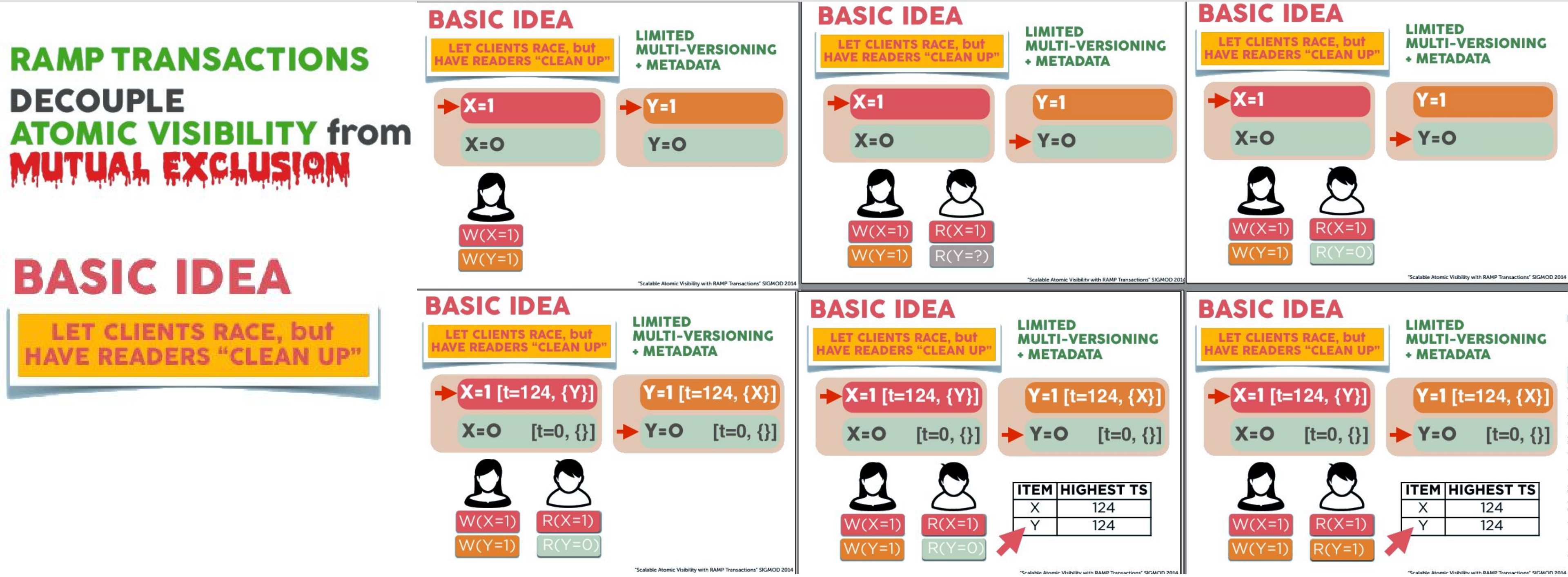
FOREIGN KEY DEPENDENCIES
NEED
ATOMIC VISIBILITY
SEE ALL OF A TXN'S UPDATES, OR NONE OF THEM

I.e. in Facebook graph:
either we're both friends,
or neither of us is

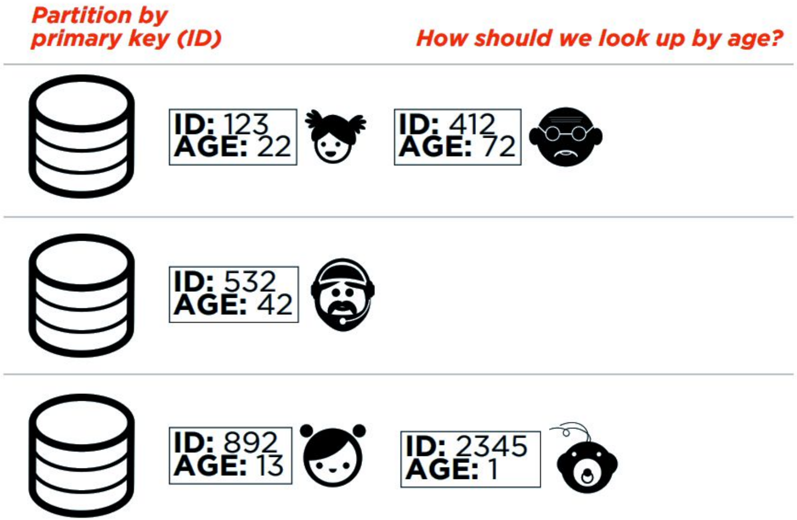


© Peter Bailis, "Coordination and the Art of Scaling", CloudantCON 2014

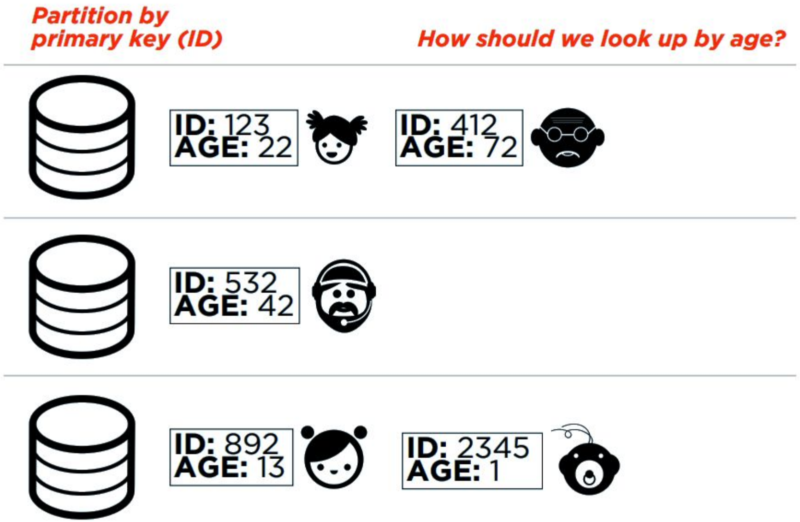
Read-atomic multi-partition transactions

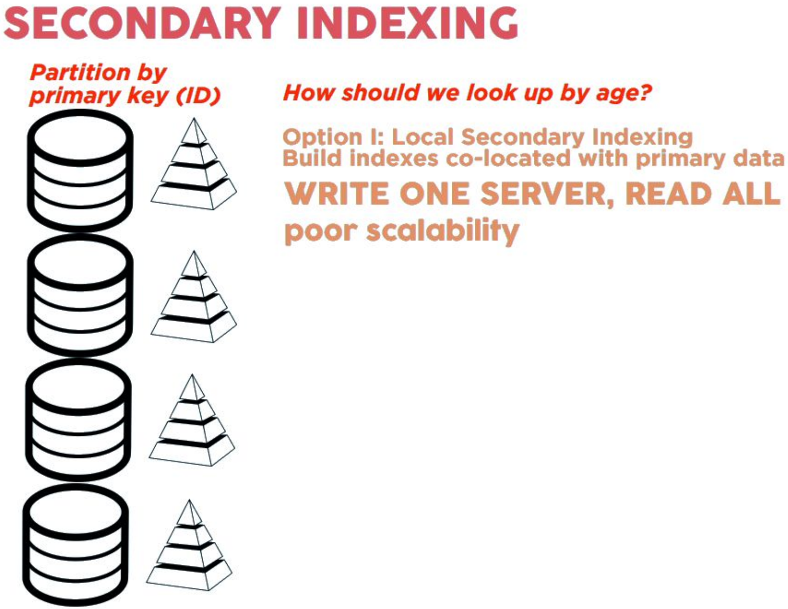


Another problem solved with RAMP transactions

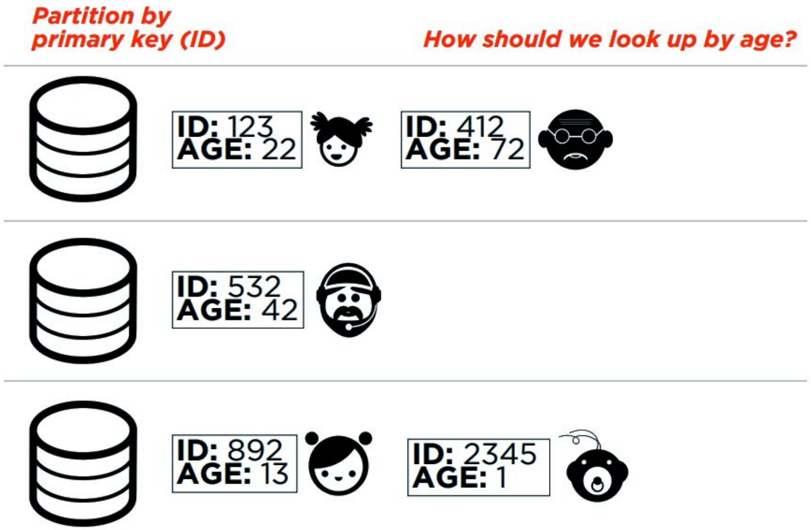


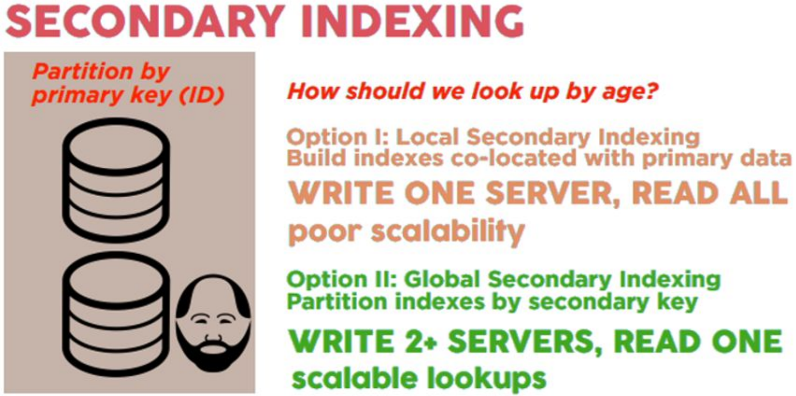
Secondary indexing

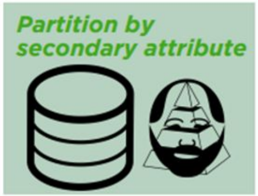




Secondary indexing







Real-world services employ either local secondary indexing (e.g., Espresso [38], Cassandra, and Google Megastore's local indexes [7]) or non-atomic (incorrect) global secondary indexing (e.g., Espresso and Megastore's global indexes, Yahoo! PNUTS's proposed secondary indexes [15]). The former is non-scalable but correct, while the latter is scalable but incorrect.

Conclusions

UNIVERSITY OF
WATERLOO

Conclusions

Traditional database systems suffer from coordination bottlenecks

By understanding application requirements, we can avoid coordination unless *necessary*

We can build systems that actually scale while providing correct behavior

Use of validations (DB constraints) in Rails web-apps:

Name	Occurrences	I-Confluent?
validates_presence_of	1762	Depends
validates_uniqueness_of	440	No
validates_length_of	438	Yes
validates_inclusion_of	201	Yes
validates_numericality_of	133	Yes
validates_associated	39	Depends
validates_email	34	Yes
validates_attachment_content_type	29	Yes
validates_attachment_size	29	Yes
validates_confirmation_of	19	Yes
Other	321	Mixed

Table 6.4: Use of and invariant confluence of built-in validations.

Thank you!

UNIVERSITY OF WATERLOO