COORDINATION AVOIDANCE IN DATABASE SYSTEMS

Peter Bailis
Stanford/MIT/Berkeley

Alan Fekete
University of Sydney

Mike Franklin
Ali Ghodsi
Joe Hellerstein
Ion Stoica
UC Berkeley

Slides of Peter Bailis' VLDB'15 talk

Serializable transactions are not as widely deployed as you might think…

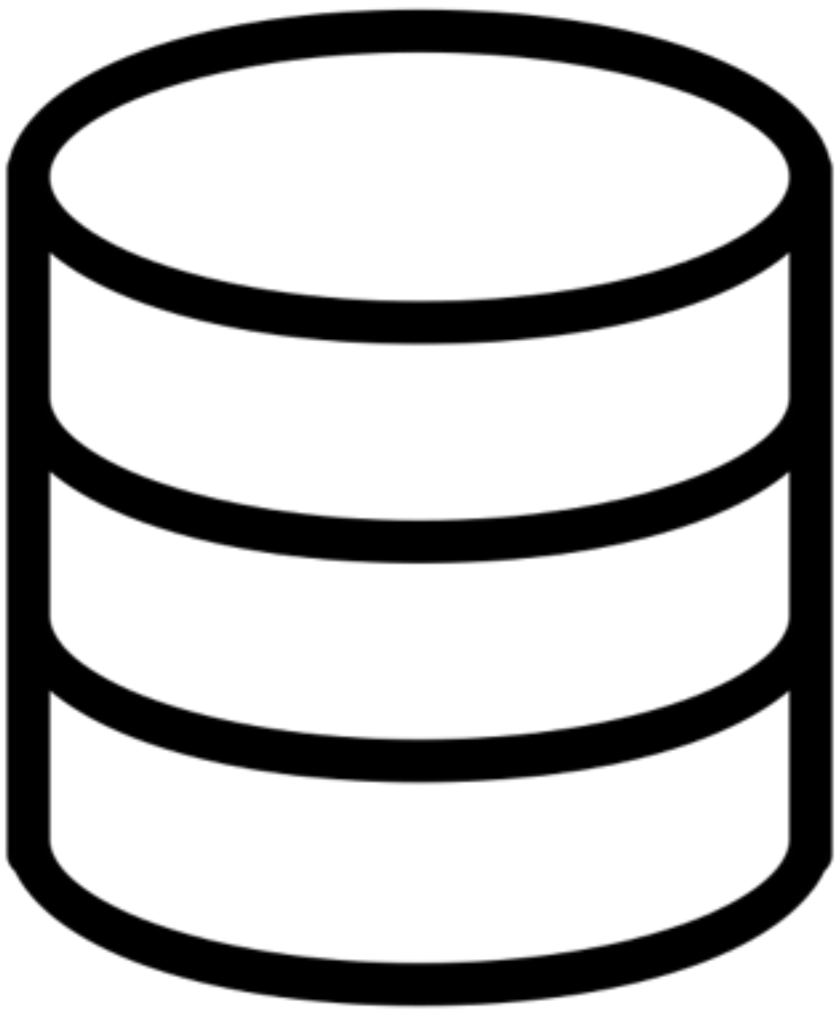
VLDB 2014	Default
Actian Ingres	NO
Aerospike	NO
Persistit	NO
Clustrix	NO
Greenplum	NO
IBM DB2	NO
IBM Informix	NO
MySQL	NO
MemSQL	NO
MS SQL Server	NO
NuoDB	NO
Oracle 11G	NO
Oracle BDB	YES
Oracle BDB JE	YES
PostgreSQL	NO
SAP Hana	NO
ScaleDB	NO
VoltDB	YES

Serializable transactions are not as widely deployed as you might think...

VLDB 2014	Default	Supported?
Actian Ingres	NO	YES
Aerospike	NO	NO
Persistit	NO	NO
Clustrix	NO	NO
Greenplum	NO	YES
IBM DB2	NO	YES
IBM Informix	NO	YES
MySQL	NO	YES
MemSQL	NO	NO
MS SQL Server	NO	YES
NuoDB	NO	NO
Oracle 11g	NO	NO
Oracle BDB	YES	YES
Oracle BDB JE	YES	YES
PostgreSQL	NO	YES
SAP Hana	NO	NO
ScaleDB	NO	NO
VoltDB	YES	YES

WHY?

serializability: equivalence to some serial execution

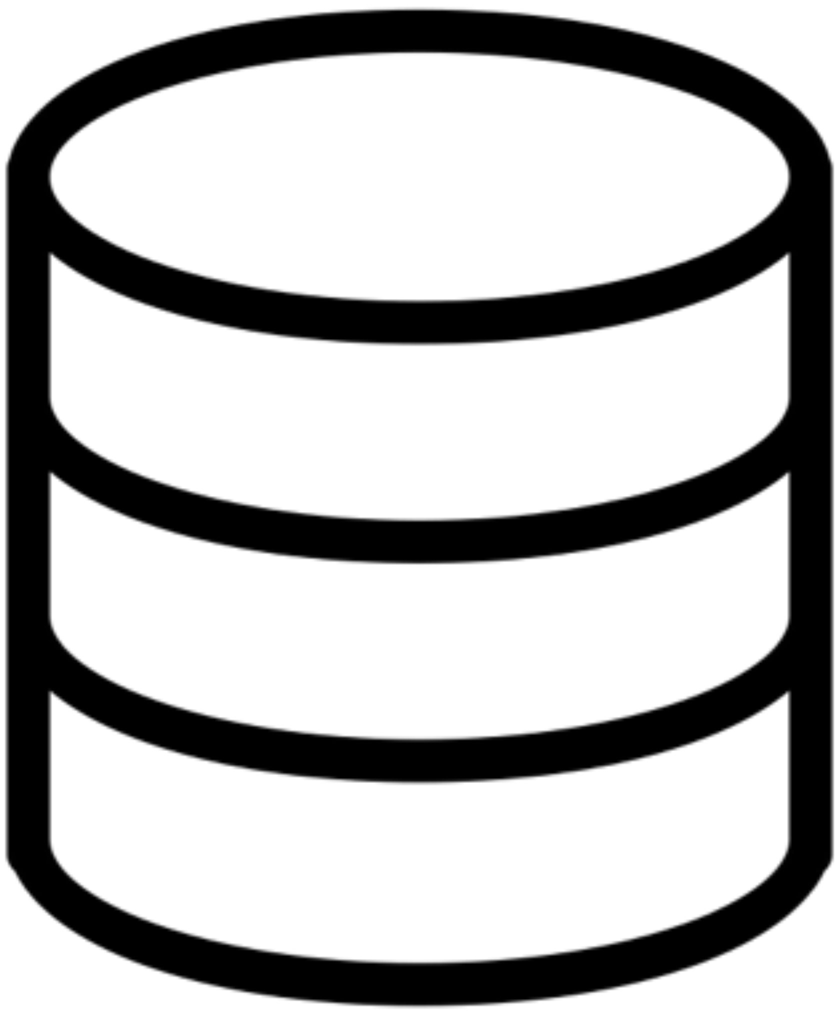


“post on timeline”

“accept friend request”

very general!

serializability: equivalence to some serial execution



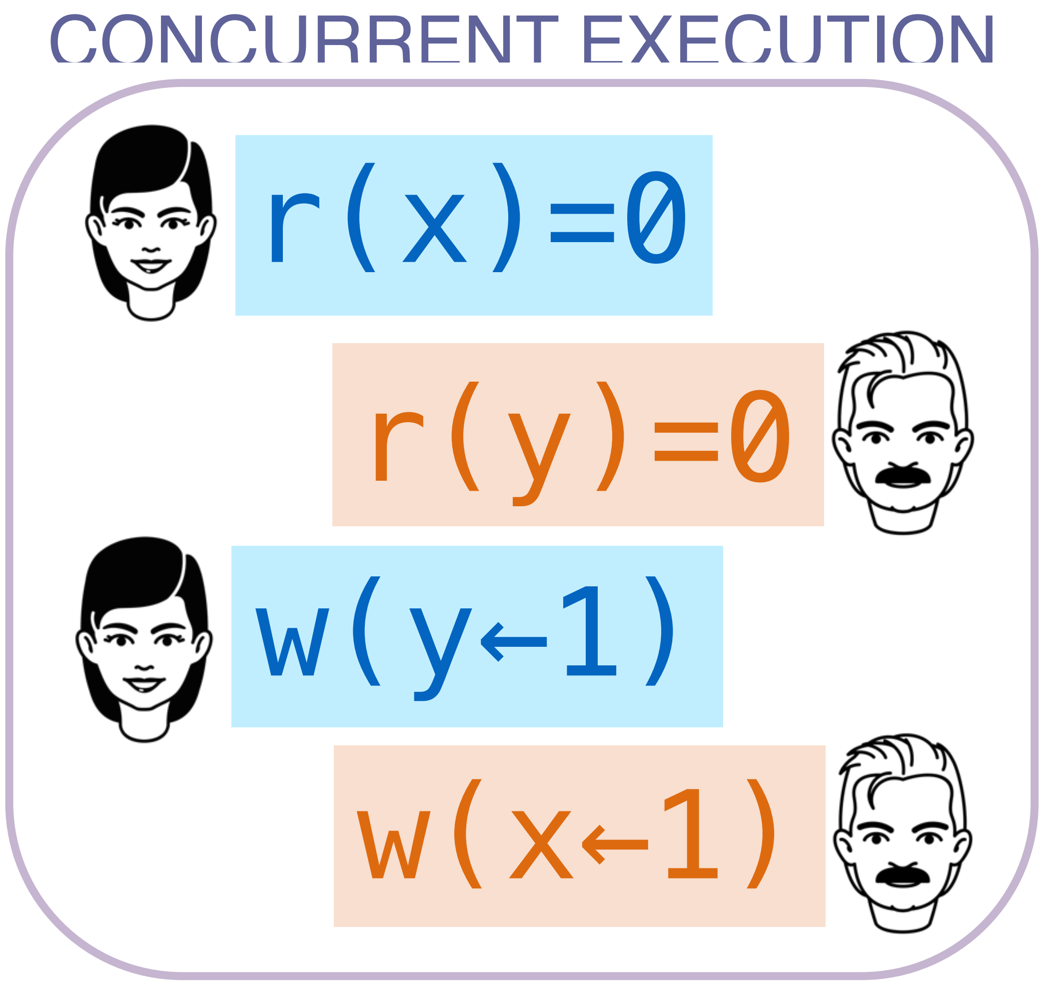
```
r(x)
w(y←1)
```

```
r(y)
w(x←1)
```

very general!

…but restricts concurrency

serializability: equivalence to some serial execution



IS NOT SERIALIZABLE!

Serializability requires Coordination

transactions cannot make progress independently

Serializability requires Coordination
transactions cannot make progress independently

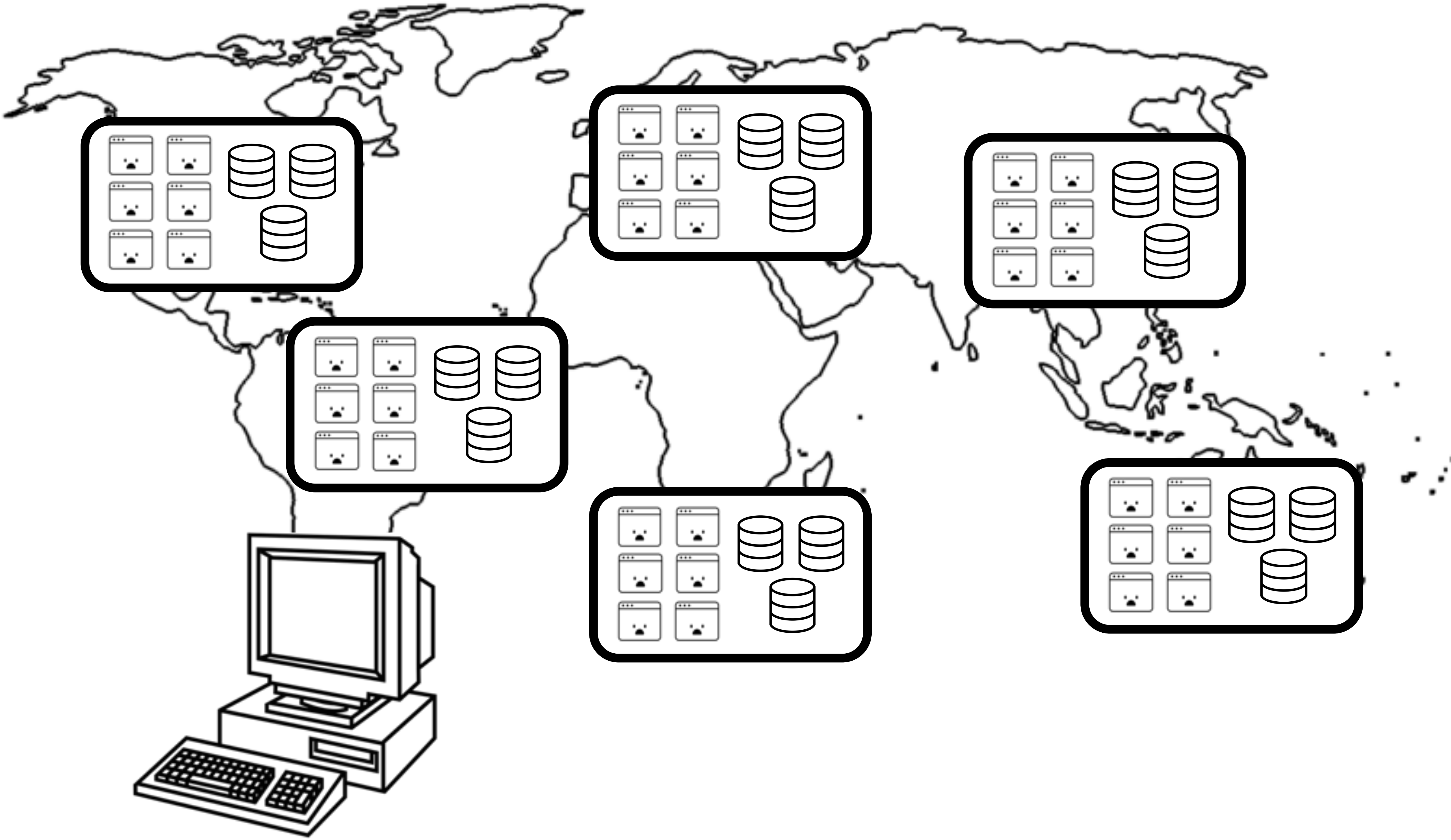
Two-Phase Locking *Multi-Version Concurrency Control*
Optimistic Concurrency Control *Pre-Scheduling*

Blocking
Waiting
Aborts

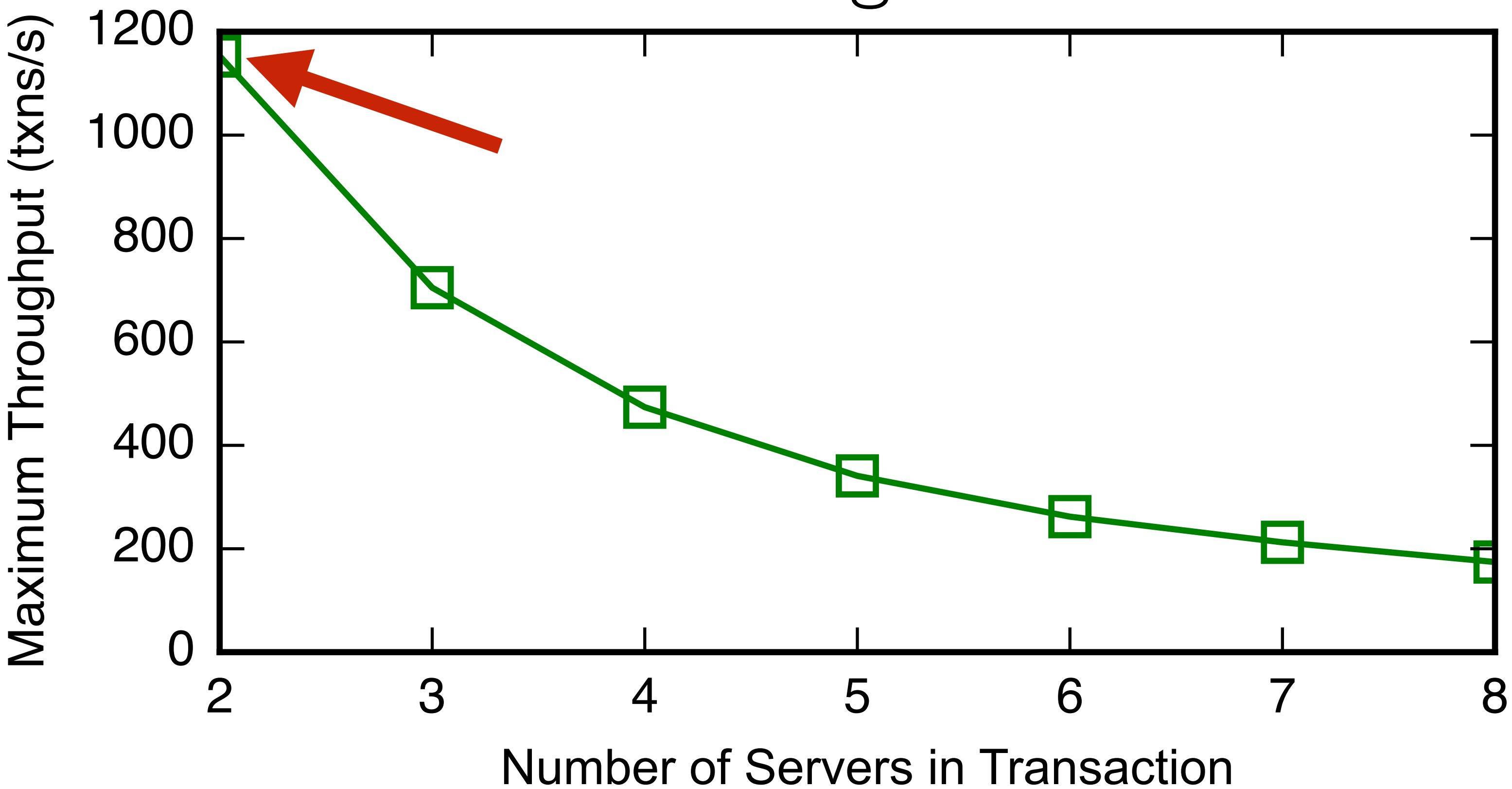
Costs of Coordination

Between Concurrent Transactions

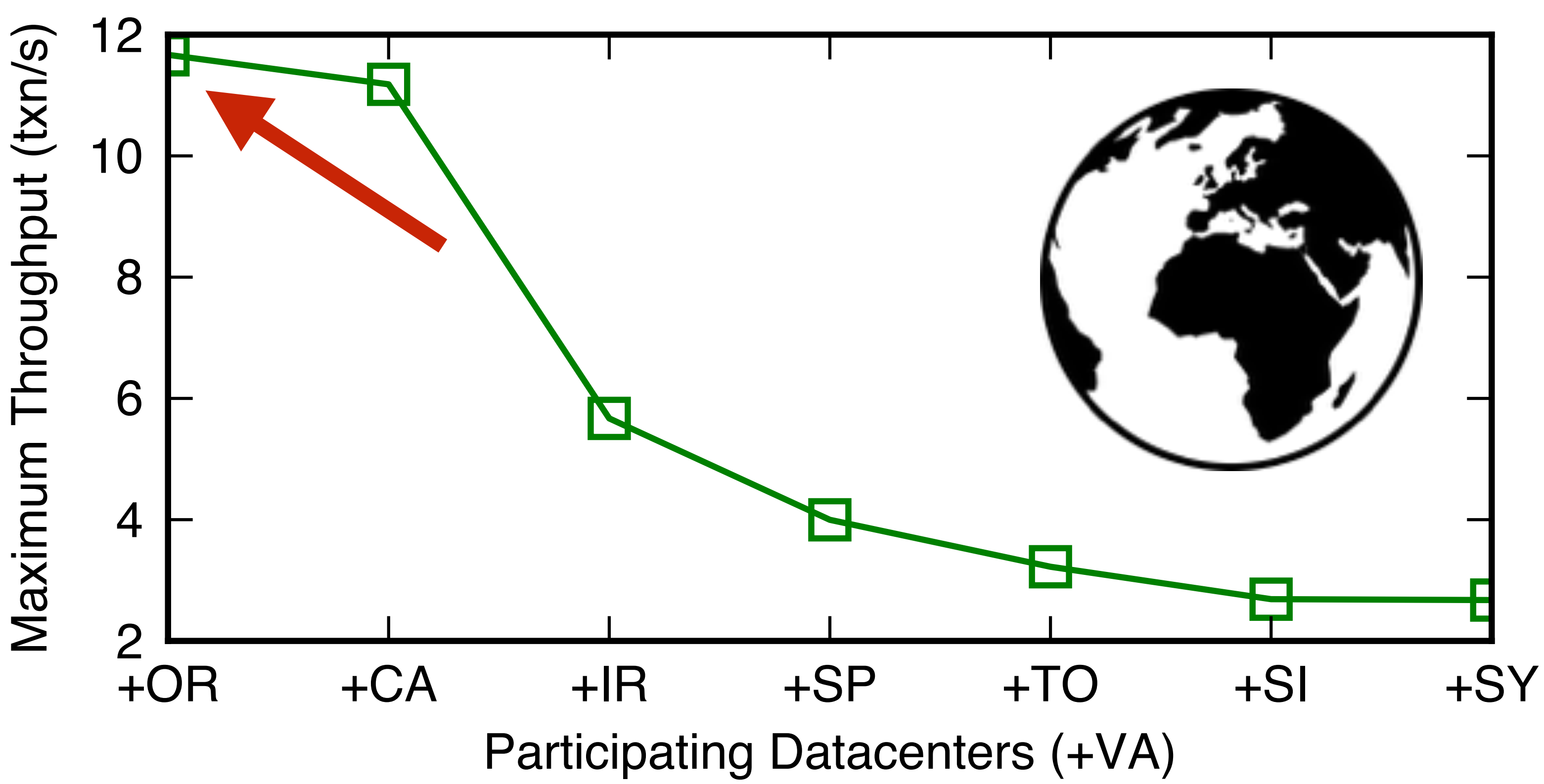
1. Decreased performance



For conflicting transactions



Local datacenter
(Amazon EC2)
Based on
[Bobtail, Xu et al., NSDI 13]



Multi-datacenter
(Amazon EC2)
Based on
[HAT, Bailis et al., VLDB 14]

Serializability requires Coordination

transactions cannot make progress independently

Costs of Coordination

Between Concurrent Transactions

1. Decreased performance

Serializability $\overset{*}{\Rightarrow}$ Consistency

Goal: maintain application-level correctness criteria

"usernames should be unique"

"account balances should remain positive"

"there should only be one administrator"

Consistency via Coordination

When is coordination strictly *necessary*?

When can we safely *avoid* coordination?

Serializability $\nLeftarrow$ Consistency

Goal: maintain application-level correctness criteria

"usernames should be unique"

"account balances should remain positive"

"there should only be one administrator"

An Optimality Theory of Concurrency Control for Databases

H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

C. H. Papadimitriou
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

1. Introduction

In many database applications it is desirable that the database system be time-shared among multiple users who access the database in an interactive way. In such a system the arriving requests for the execution of steps in different transactions from different users may by interleaved in any order. Assume that each transaction is correct in the sense that it preserves the consistency of the database when executed alone. The execution of many correct transactions in an interleaved order may, however, bring a consistent database state into an inconsistent one (see, e.g., [Eswaran et al. 76]). It is the task of the concurrency control mechanism of the database system, which is also called scheduler in this paper, to safeguard the database consistency by properly granting or rejecting the execution of arriving requests. A rejected request is scheduled for execution after some requests which arrive later have been scheduled for execution. That is, the concurrency control enforces database consistency by delaying the execution of some requests when this is necessary.

Although system consistency is the primary objective of concurrency control, there are certain other important considerations that must be taken into account in its design. For instance, one sure way to secure consistency would be to delay all other user requests until the first user logs out, then let the second user go, and so on. Since each individual transaction is correct, the execution of requests in this order will preserve consistency. Obviously, this straight-forward mechanism has a major deficiency: it may cause unnecessary delays for all but one user, and thus degrade the throughput and response time of the system. This scheduler, however, does have one important advantage. Namely, it requires no information about the transactions except for a user identification for each request. We see therefore that it is necessary to consider the performance of a scheduler and the information that it uses, in addition to its correctness.

Performance. We measure the performance of a scheduler by the set of request sequences which the scheduler can pass without any delay. We call this set the fixpoint set of the scheduler. The idea is that the richer this set is, the more likely that no delays will be imposed by the scheduler to the user requests. In fact, if the fixpoint set of a scheduler strictly includes that of another scheduler, then it can be argued that the former scheduler performs strictly better than the latter one as far as average delays are concerned. Further justification of this measure, as well as a discussion of its limitations appears in Section 6.

Information. The information used by a scheduler is the minimum knowledge about the database and the

SIGMOD 1979

Without knowledge of application, should use serializability

Serializability $\nLeftarrow$ Consistency

Goal: maintain application-level correctness criteria

“usernames should be unique”

“account balances should remain positive”

“there should only be one administrator”

Approach: use application-level criteria as basis for concurrency control

This paper's contributions:

1. *The I-confluence test (ICT):* Necessary, sufficient criterion for safe, coordination-free execution
2. ICT for SQL-based operations
3. Empirical speedups for real workloads

Semantic concurrency control is back, works, and is needed!

Constraint	*Operation*
Equality, Inequality	Any
Generate unique ID	Any
Specify unique ID	Insert
>	Increment
>	Decrement
<	Decrement
<	Increment
Foreign Key	Insert
Foreign Key	Delete
Secondary Indexing	Any
Materialized Views	Any

Typical database constraints and operations

(SQL)

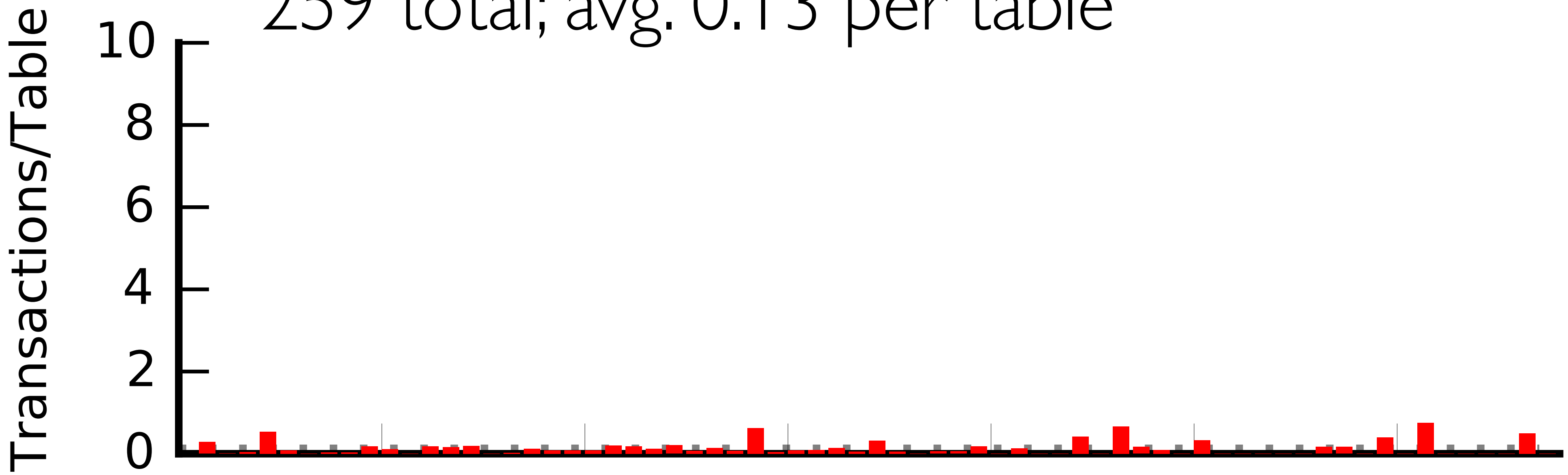
github
SOCIAL CODING

adopt-a-hydrant
alchemy_cms
amahi
bostonrb
boxroom
brevidy
browsercms
bucketwise
calagator
canvas-lms
carter
chiliproject
citizenry
comas
comfortable-
mexican-sofa
communityengine
copycopter-
server
danbooru
diaspora
discourse
enki
fat_free_crm
fedena
forem
fulcrum
gitlab-ci
gitlabhq
govsgo
heaven
inkwell
insoshi
jobsworth
juvia
kandan
linuxfr.org
lobsters
lovd-by-less
nimbleshop
obtvse
onebody
opal
opencongress
opengovernment
openproject
piggybak
publify
radiant
railscollab
redmine
refinerycms
ror_ecommerce
rucksack
saasy
salor-retail
selfstarter
sharetribe
skyline
spot-us
spree
sprintapp
squaresquash
sugar
teambox
tracks
tryshoppe
wallgig

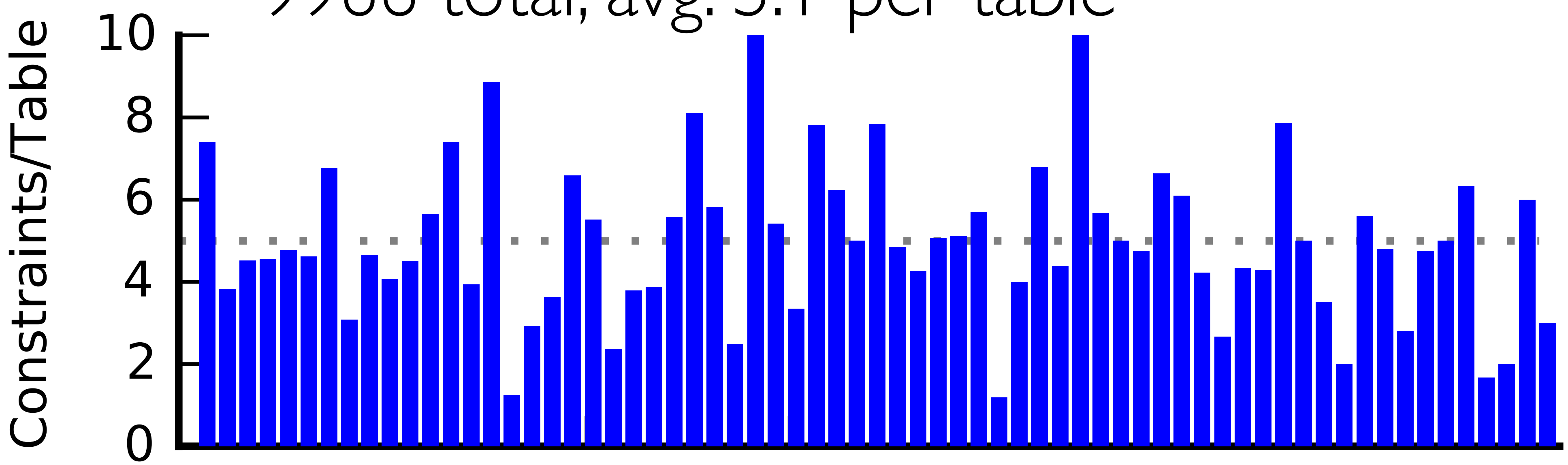
adopt-a-hydrant
alchemy_cms
amahi
bostonrb
boxroom
brevidy
browsercms
bucketwise
calagator
canvas-lms
carter
chiliproject
citizenry
comas
comfortable-mexican-sofa
communityengine
copycopter-server
danbooru
diaspora
discourse
enki
fat_free_crm
fedena
forem
fulcrum
gitlab-ci
gitlabhq
govsgo
heaven
inkwell
insoshi
jobsworth
juvia
kandan
linuxfr.org
lobsters
lovd-by-less
nimbleshop
obtvse
onebody
opal
opencongress
opengovernment
openproject
piggybak
publify
radiant
railscollab
redmine
refinerycms
ror_ecommerce
rucksack
saasy
salor-retail
selfstarter
sharetribe
skyline
spot-us
spree
sprintapp
squaresquash
sugar
teambox
tracks
tryshoppe
wallgig
zena

67 projects 1.77M LoC 1957 tables

259 total; avg. 0.13 per table



9986 total; avg. 5.1 per table



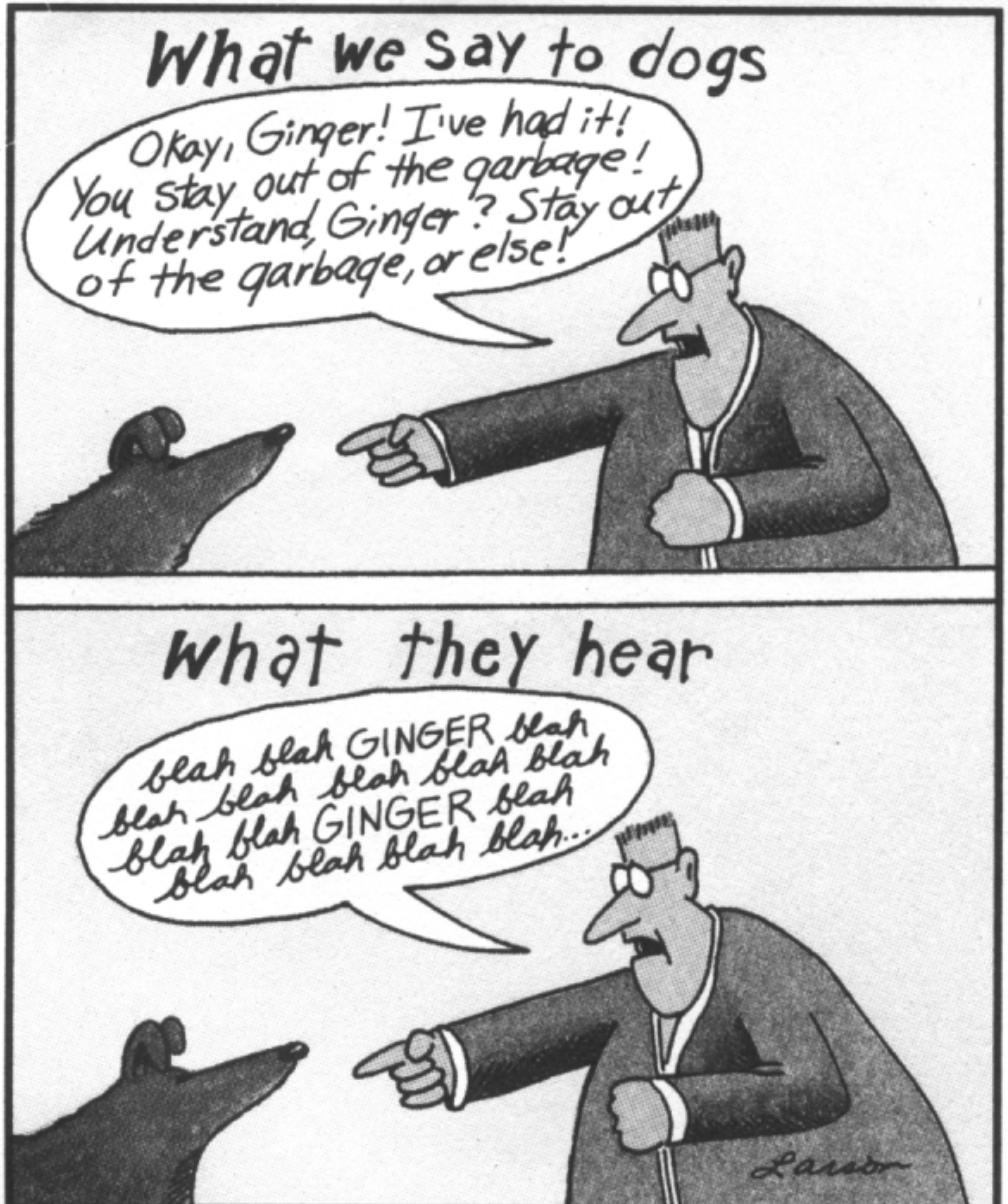
CONSTRAINTS MORE COMMON 37x

[SIGMOD 2015]

When is coordination strictly *necessary*?

When can we safely *avoid* coordination?

Key idea: Check if application constraints can be violated by "merging" independent operations



The Far Side,
Gary Larson

“post
on
timeline”

TODAY:
ENFORCEMENT
VIA

WHAT THE APPLICATION SAYS

WHAT THE DATABASE HEARS

read write write write read

read

write

write

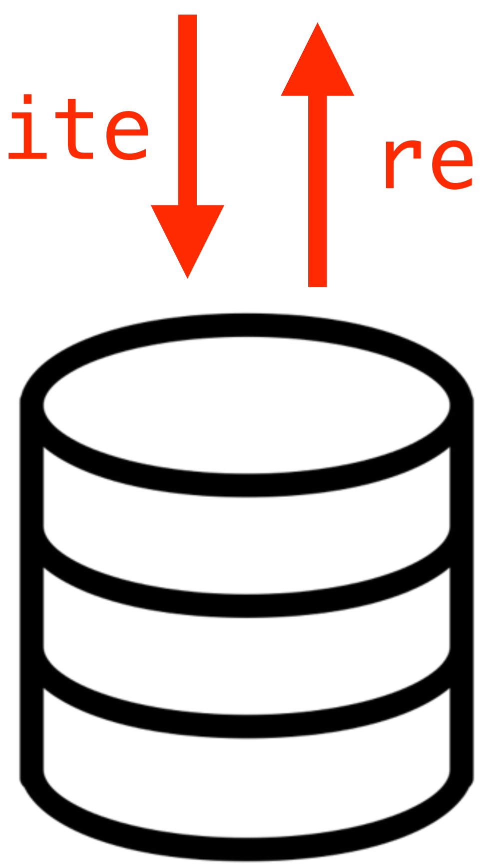
read

read write read

read

write write read

read read



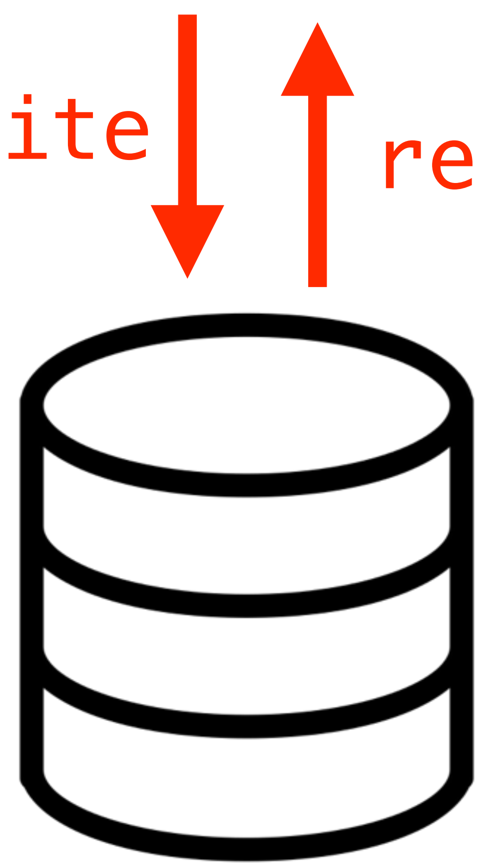
“no duplicate users”

CAN WE USE CONSTRAINTS TO AVOID

WHAT THE APPLICATION SAYS

WHAT THE DATABASE HEARS

read write write write



read

read

write

write

read

read write read

read

write write read

read read

“no duplicate users”

CAN WE USE CONSTRAINTS TO AVOID

WHAT THE APPLICATION SAYS

WHAT THE DATABASE HEARS

constraint

constraint

constraint

“no duplicate users”

constraint

constraint

constraint

Some Definitions

- Replica R belonging to database D is I-valid if I(R) = true
- A system is globally I-valid if all replicas are always I-valid
- A transaction set T is I-confluent if for all reachable states Di and Dj with a common ancestor, the result of merging Di and Dj (Di U Dj) is I-valid

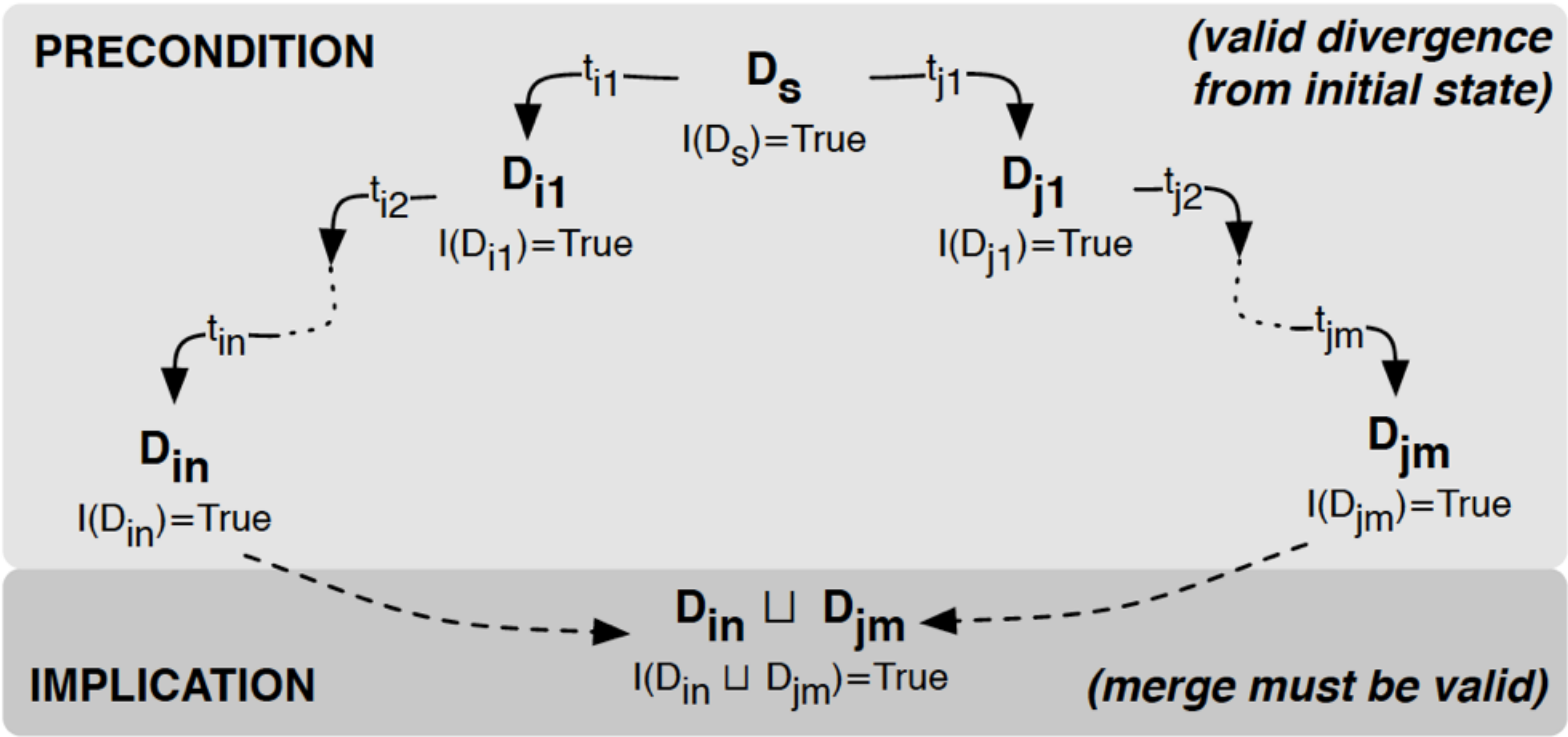


Figure 4: An $\mathcal{I}$-confluent execution illustrated via a diamond diagram. If a set of transactions T is $\mathcal{I}$-confluent, then all database states reachable by executing and merging transactions in T starting with a common ancestor (D_s) must be mergeable ($\sqcup$) into an I-valid database state.

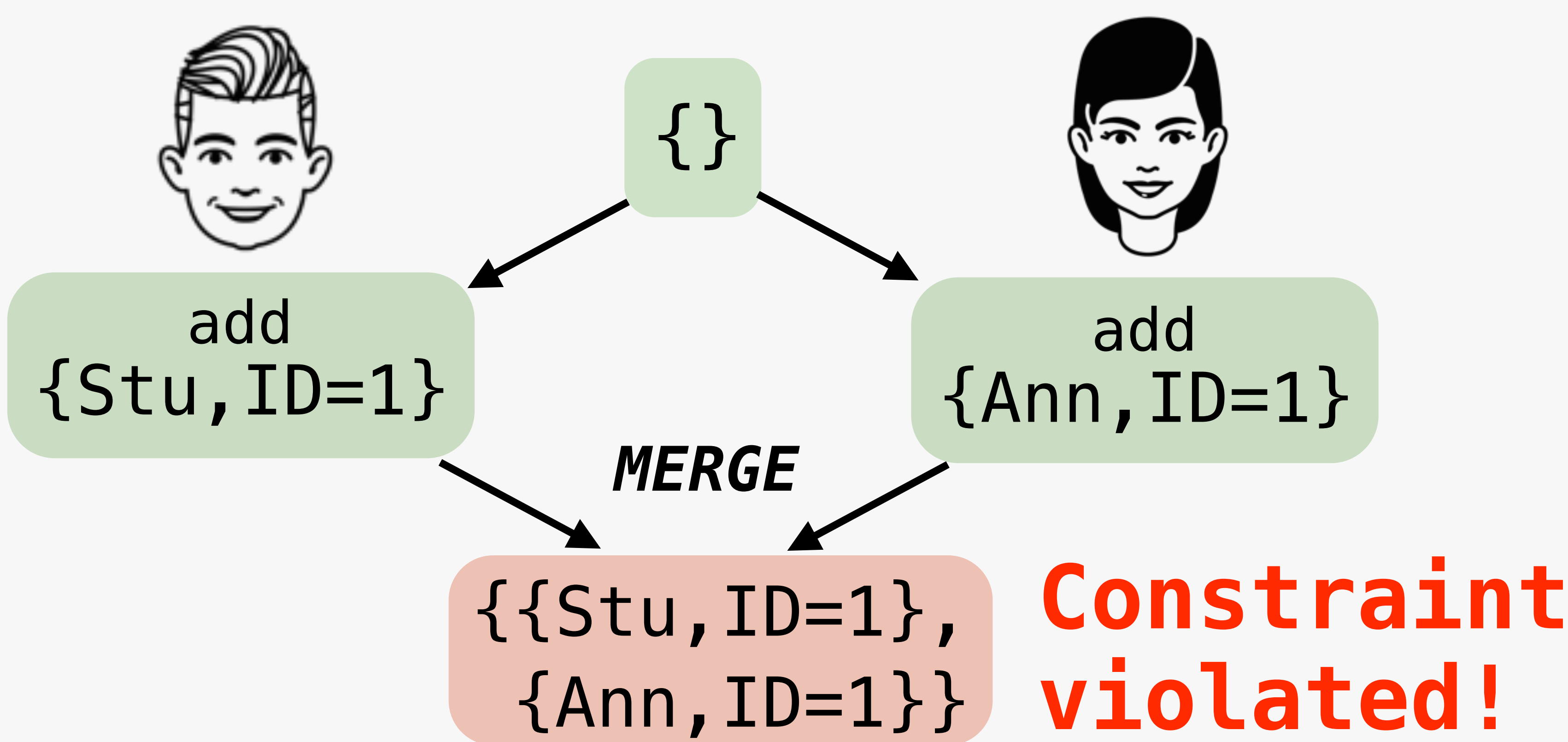
I-confluence Theorem

A globally I-valid system can execute a set of transactions T with coordination-freedom, transactional availability, convergence if and only if T is I-confluent with respect to I.

Key idea: Check if constraints can be violated by "merging" independent operations

ICT: Invariant Confluence Test

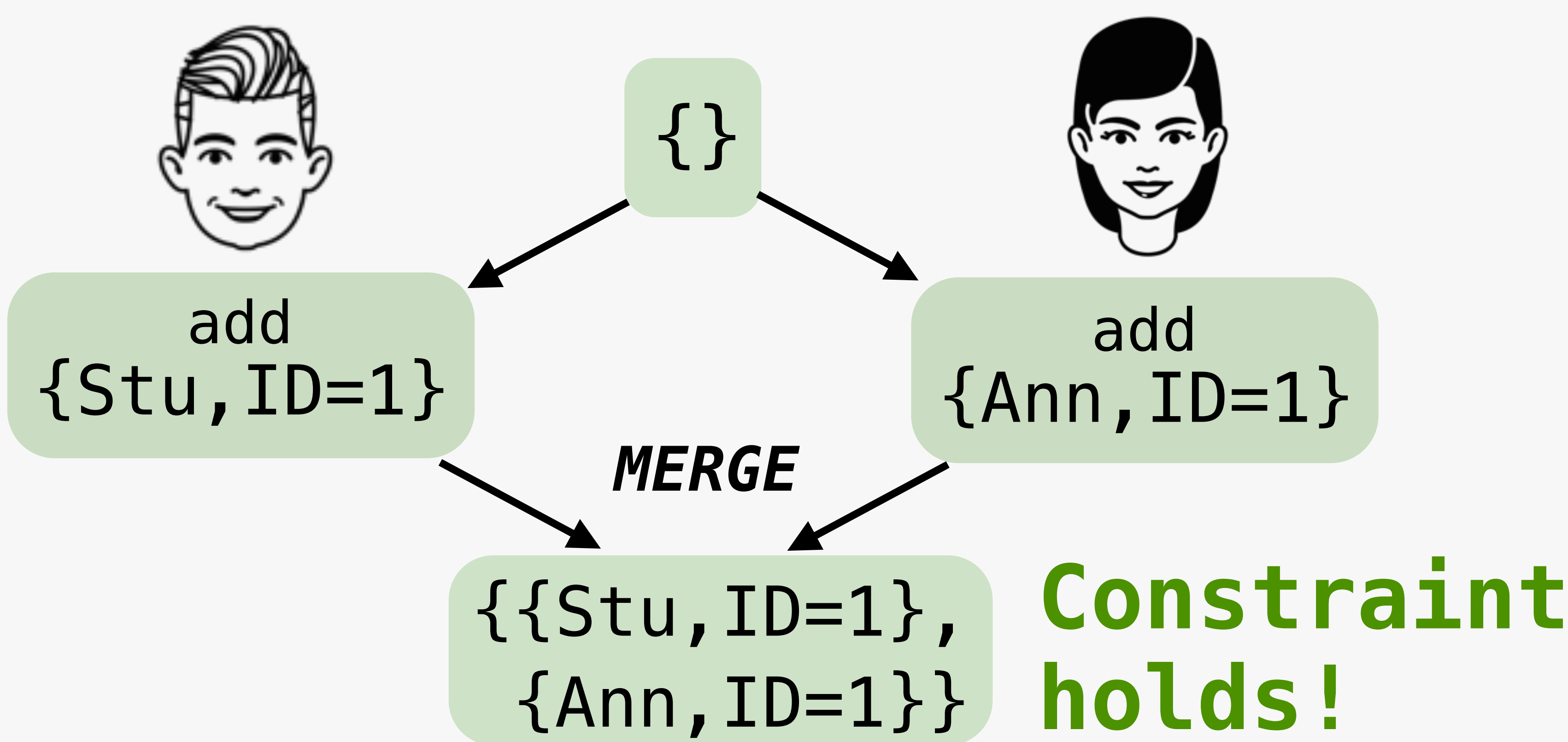
CONSTRAINT: User IDs are unique
OPERATION: Add users
MERGE: Set union



Key idea: Check if constraints can be violated by "merging" independent operations

ICT: Invariant Confluence Test

```
CONSTRAINT: User IDs are positive
OPERATION:  Add users
MERGE:      Set union
```



Key idea: Check if constraints can be violated by "merging" independent operations

ICT: Invariant Confluence Test

OUR CONTRIBUTION:

ICT $\Leftrightarrow$ safe, coordination-free execution possible

Theorem. A globally I-valid system can execute a set of transactions T with coordination-freedom, transactional availability, and convergence *if and only if* T are I-confluent with respect to I.

Generalizes classic partitioning-based indistinguishability arguments

Constraint	*Operation*	*OK?*
Equality, Inequality	Any	???
Generate unique ID	Any	???
Specify unique ID	Insert	???
>	Increment	???
>	Decrement	???
<	Decrement	???
<	Increment	???
Foreign Key	Insert	???
Foreign Key	Delete	???
Secondary Indexing	Any	???
Materialized Views	Any	???

Typical database constraints and operations (SQL)

Under set merge

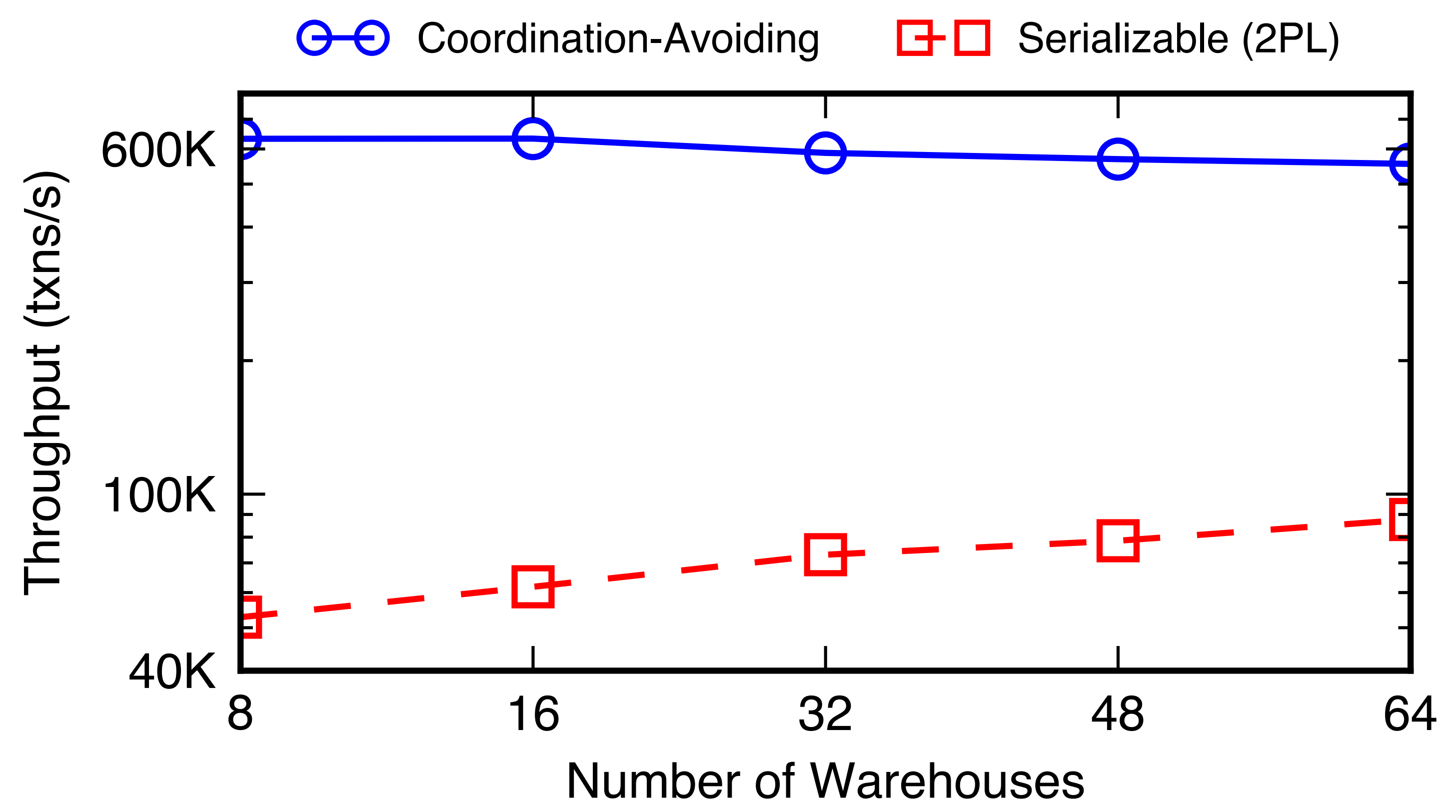
Constraint	*Operation*	*OK?*
Equality, Inequality	Any	Y
Generate unique ID	Any	Y
Specify unique ID	Insert	N
>	Increment	Y
>	Decrement	N
<	Decrement	Y
<	Increment	N
Foreign Key	Insert	Y
Foreign Key	Delete	Y*
Secondary Indexing	Any	Y
Materialized Views	Any	Y

Typical database constraints and operations (SQL)

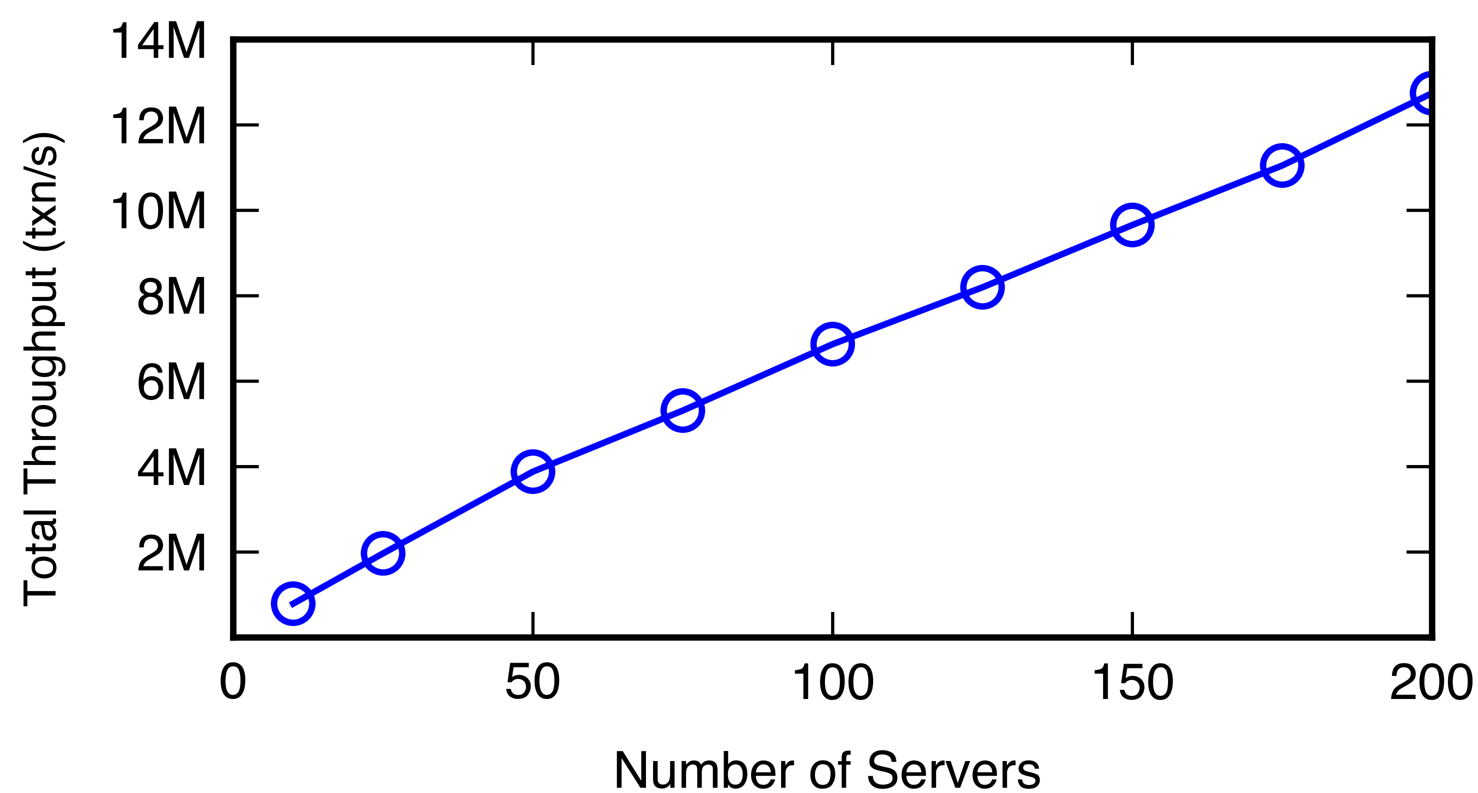
Under set merge

TPC-C 14/16 CONSTRAINTS PASS ICT

6-11x faster than ACID/serializability



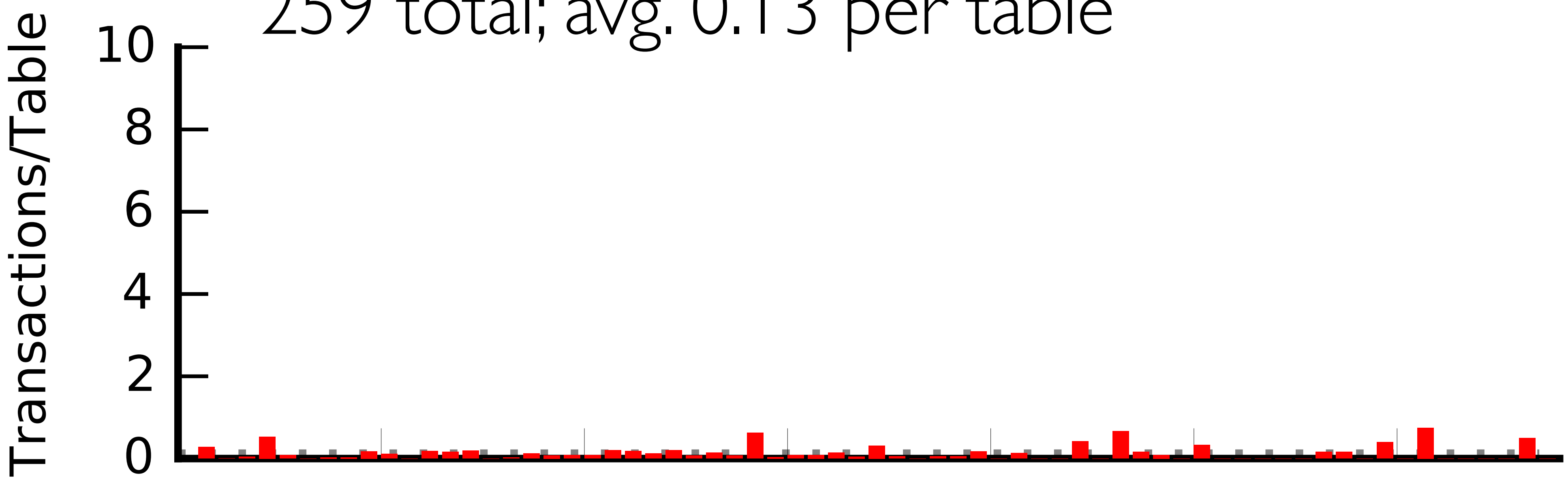
scale to over **25x** best listed result



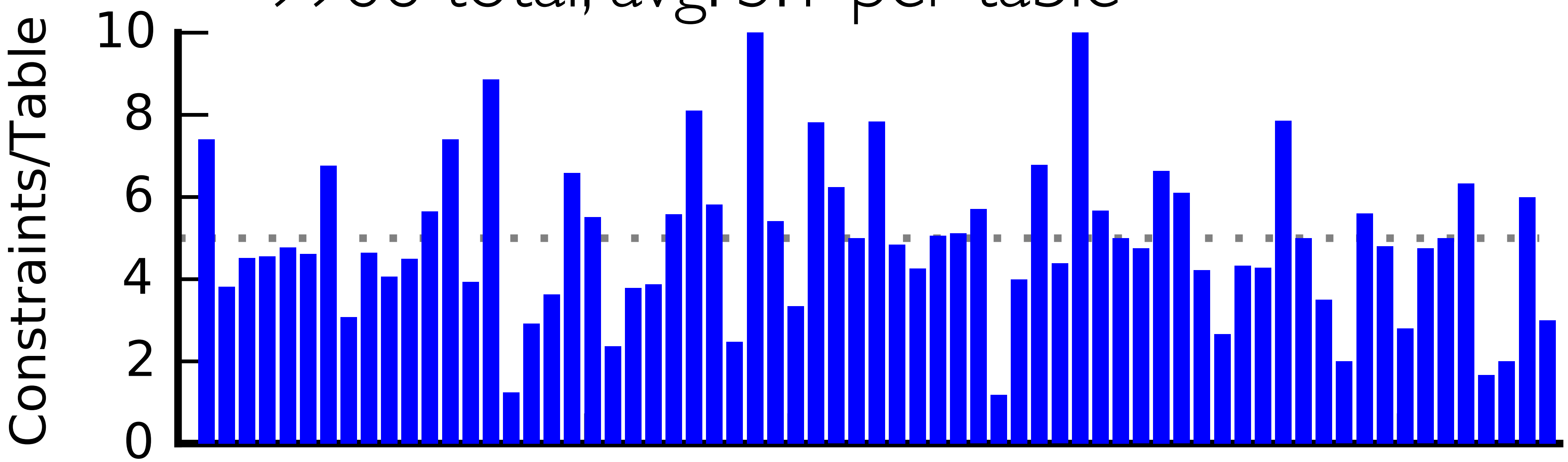
adopt-a-hydrant
alchemy_cms
amahi
bostonrb
boxroom
brevidy
browsercms
bucketwise
calagator
canvas-lms
carter
chiliproject
citizenry
comas
comfortable-mexican-sofa
communityengine
copycopter-server
danbooru
diaspora
discourse
enki
fat_free_crm
fedena
forem
fulcrum
gitlab-ci
gitlabhq
govsgo
heaven
inkwell
insoshi
jobsworth
juvia
kandan
linuxfr.org
lobsters
lovd-by-less
nimbleshop
obtvse
onebody
opal
opencongress
opengovernment
openproject
piggybak
publify
radiant
railscollab
redmine
refinerycms
ror_ecommerce
rucksack
saasy
salor-retail
selfstarter
sharetribe
skyline
spot-us
spree
sprintapp
squaresquash
sugar
teambox
tracks
tryshoppe
wallgig
zena

67 projects 1.77M LoC 1957 tables

259 total; avg. 0.13 per table



9986 total; avg. 5.1 per table



86.9% PASS ICT

[SIGMOD 2015]

Serializable transactions have largely failed

VLDB 2014	Default	Supported?
Actian Ingres	NO	YES
Aerospike	NO	NO
Persistit	NO	NO
Clustrix	NO	NO
Greenplum	NO	YES
IBM DB2	NO	YES
IBM Informix	NO	YES
MySQL	NO	YES
MemSQL	NO	NO
MS SQL Server	NO	YES
NuoDB	NO	NO
Oracle 11G	NO	NO
Oracle BDB	YES	YES
Oracle BDB JE	YES	YES
PostgreSQL	NO	YES
SAP Hana	NO	NO
ScaleDB	NO	NO
VoltDB	YES	YES

Conclusions

- Serializable transactions suffer from fundamental coordination overheads
- By reasoning about application correctness criteria instead of read-write traces, we can avoid coordination
- I-confluence is a necessary and sufficient condition for preserving criteria under coordination-free execution
- Real apps are I-confluent and allow huge speedups