Antoni Wolski

Who needs transactions any more?

Antoni Wolski, Ph.D.
AWO Consulting

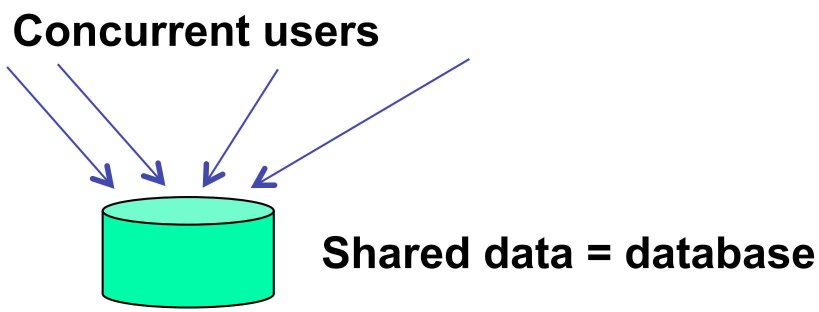
a.wolski@acm.org

Database Transactions Summit 2013
Haaga-Helia, Helsinki
2013-09-04

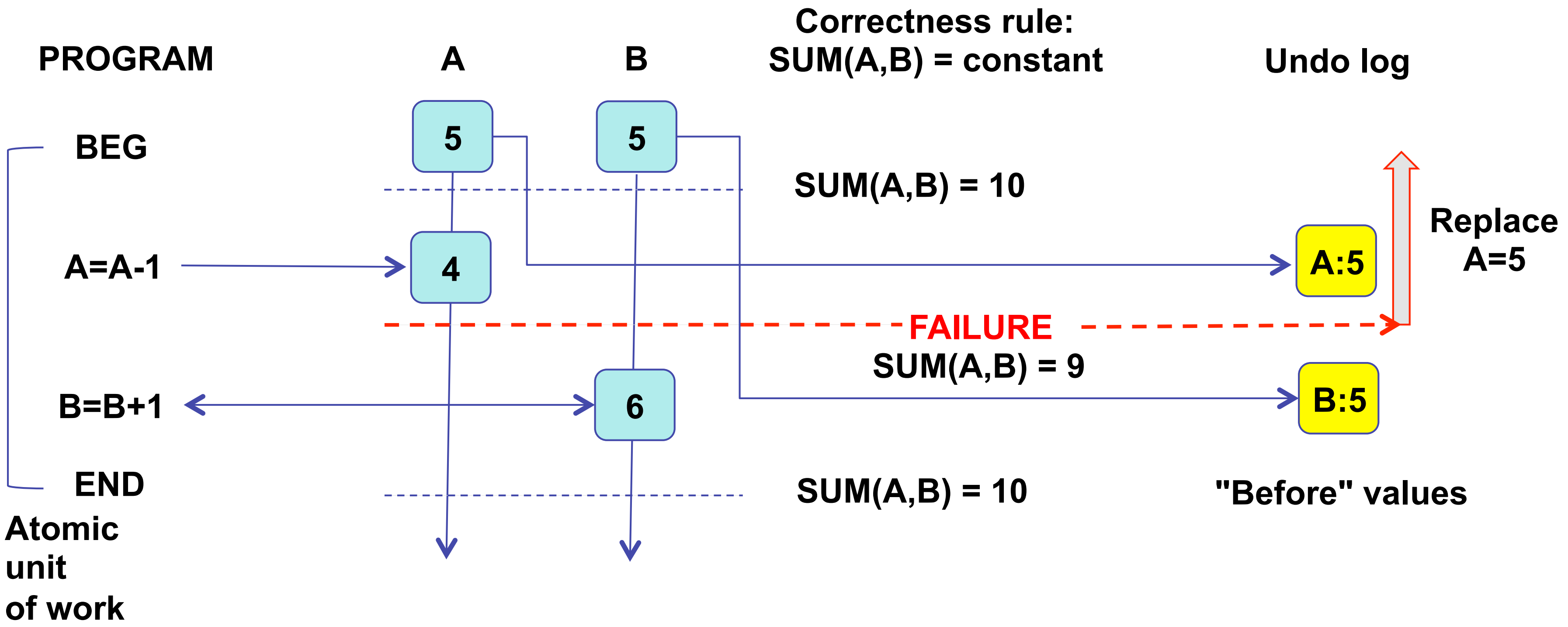
Transactions existed before they were invented

These questions have been bothering people since the first days of using shared data

- What to do when you fail in the middle of doing something?
- How to ensure that the result is correct?
- How to protect data from being messed up by concurrent apps?
- How to ensure that the results will not disappear upon a failure?

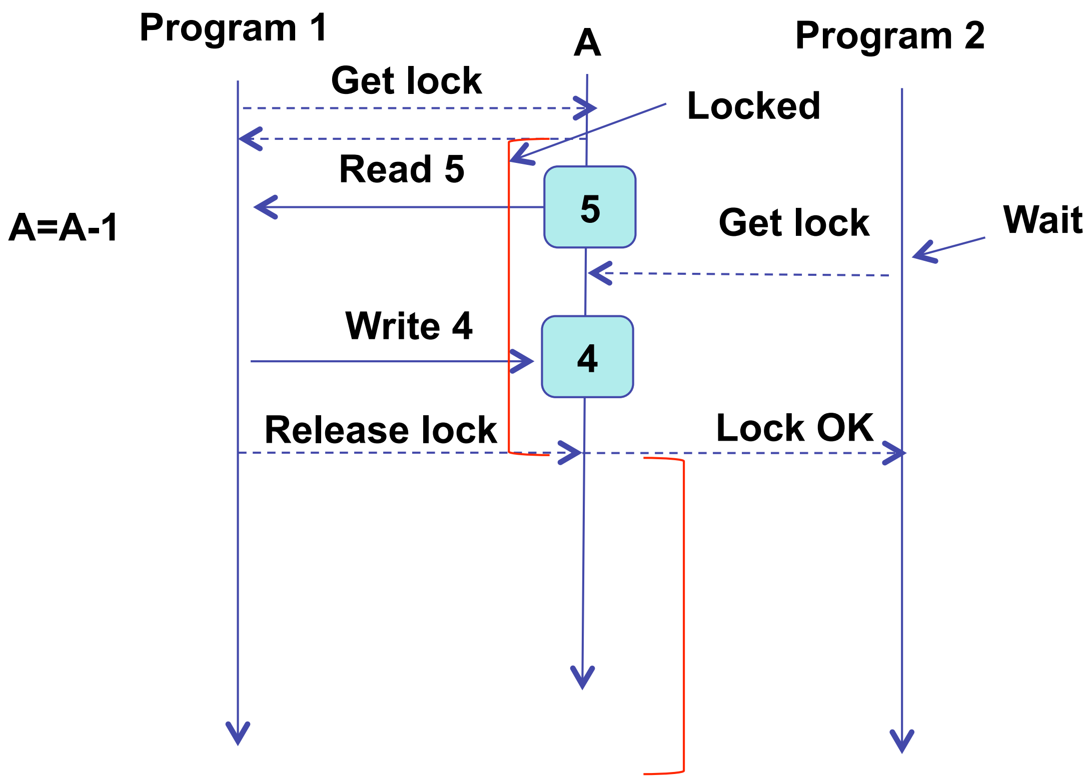


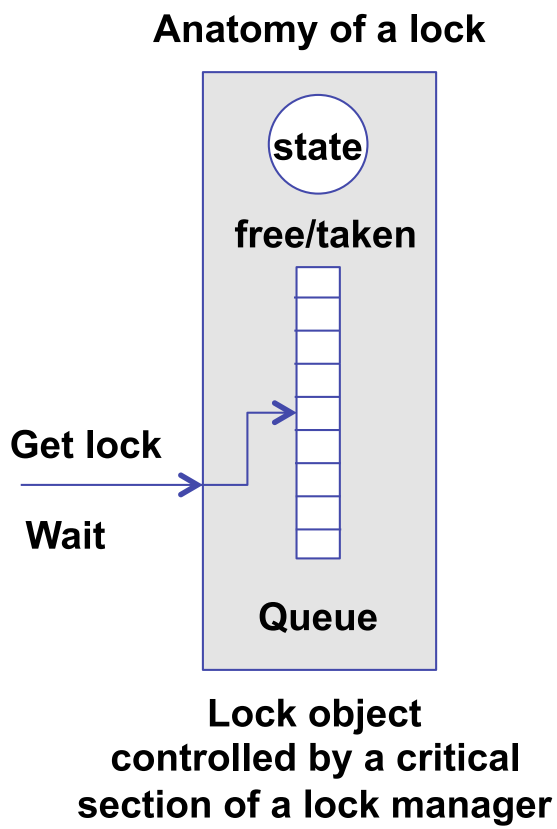
Example: Protect atomicity with the undo log



- If there is a failure inside an atomic unit of work, the partial results are removed, and the original values restored by using before images stored in the undo log.

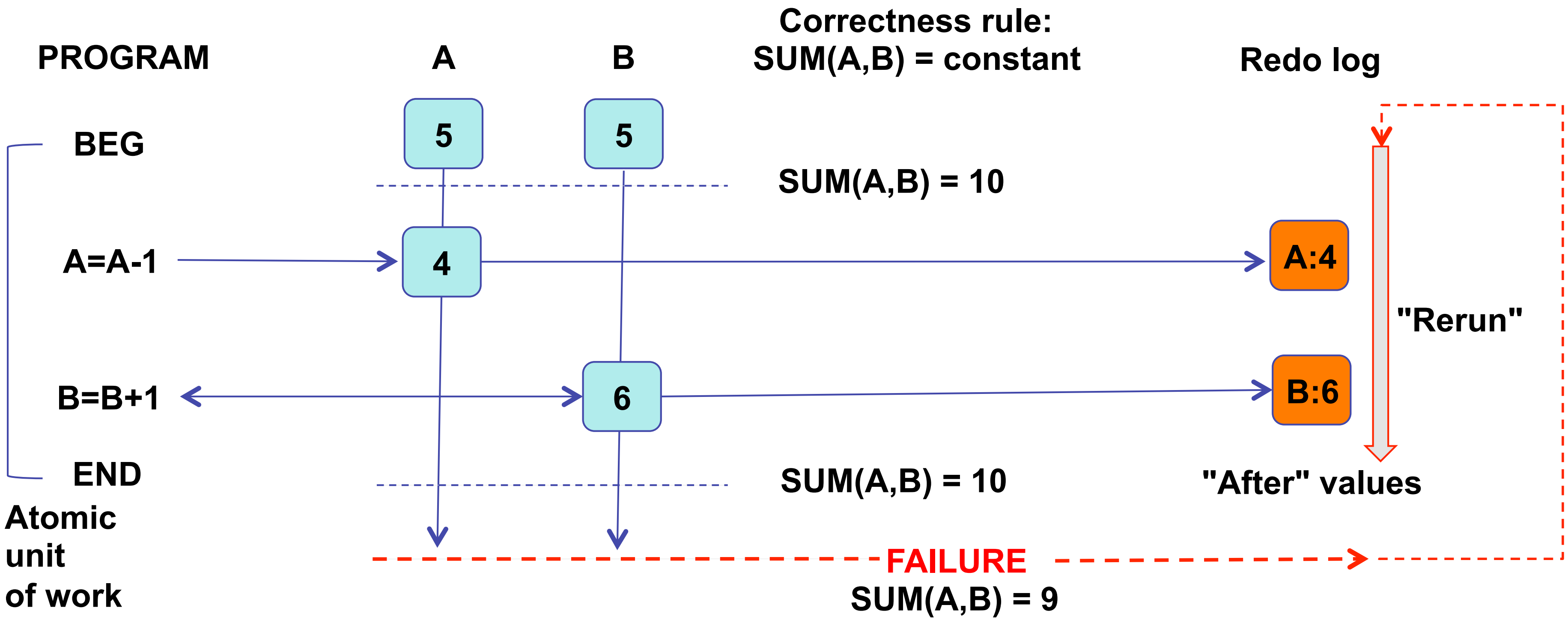
Example: Protect against update anomalies with locks





- Locks were invented in first data management systems in the 60's

Protect committed data with redo log

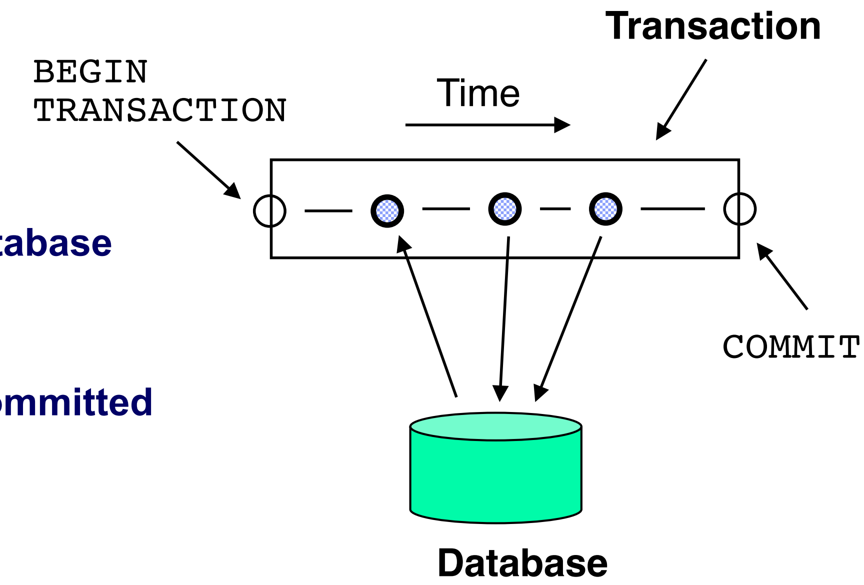


- If there is a failure immediately after the end of an atomic unit of work, there is no guarantee that the new state has been propagated to the disk.
- The latest state is however stored in the redo log and it can be "rerun".

The full package: an ACID transaction

Transaction (unit of work): a sequence of operations, having the following properties(ACID):

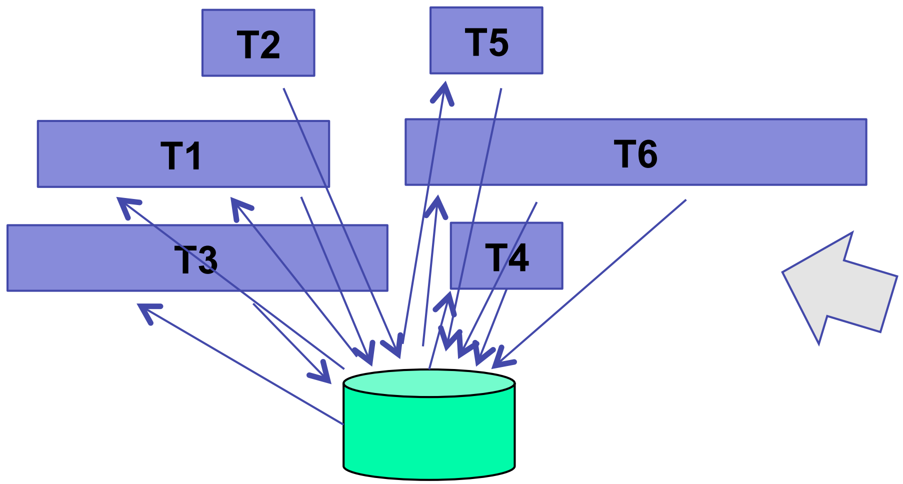
- Atomicity
  Either all or none
- Consistency
  The effect of a transaction is a consistent database state (in the presence of constraints)
- Isolation
  The changes are not seen before they are committed
- Durability
  The effects are immediately permanent



A system maintaining ACID properties produces serializable and recoverable transaction schedules.

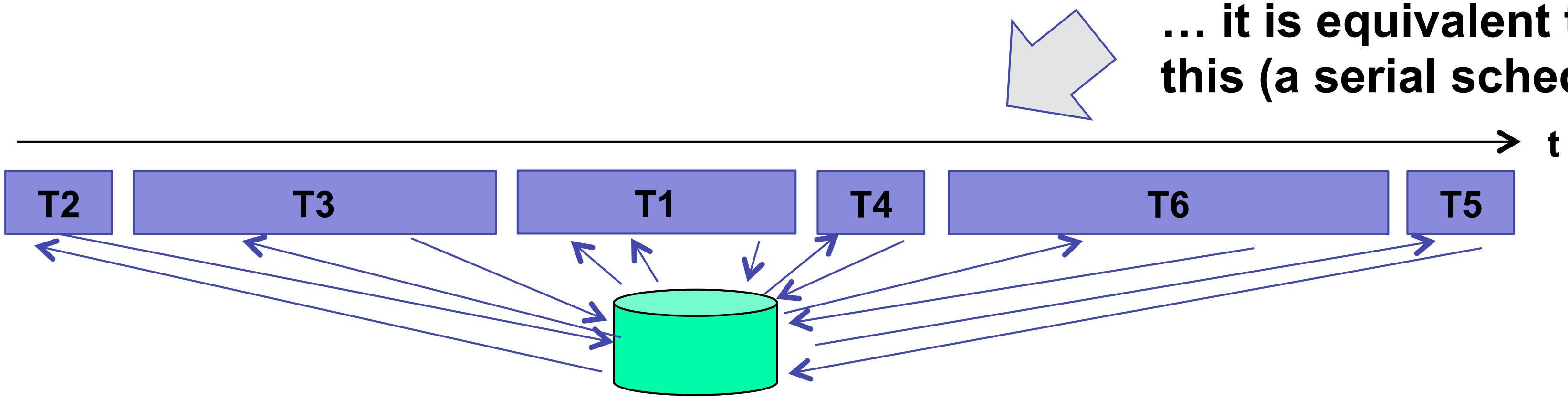
Understanding isolation

The goal is serializability

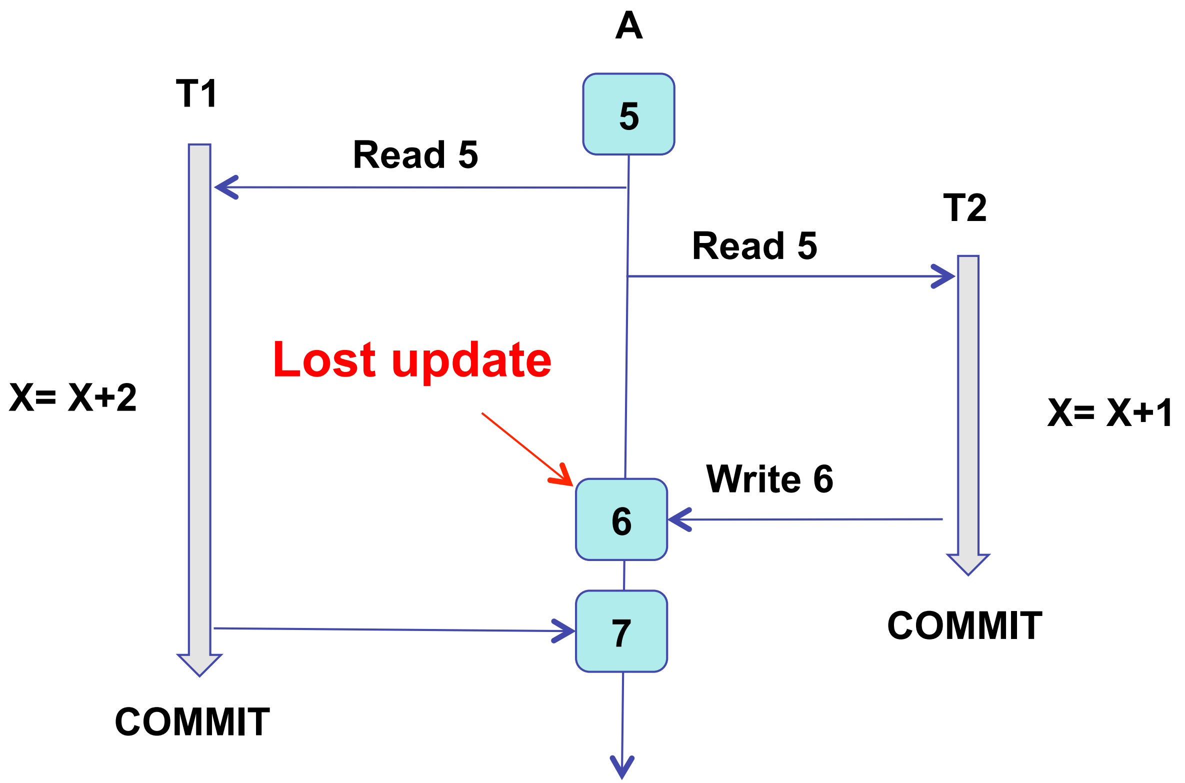


If this is a serializable schedule …

… it is equivalent to this (a serial schedule)

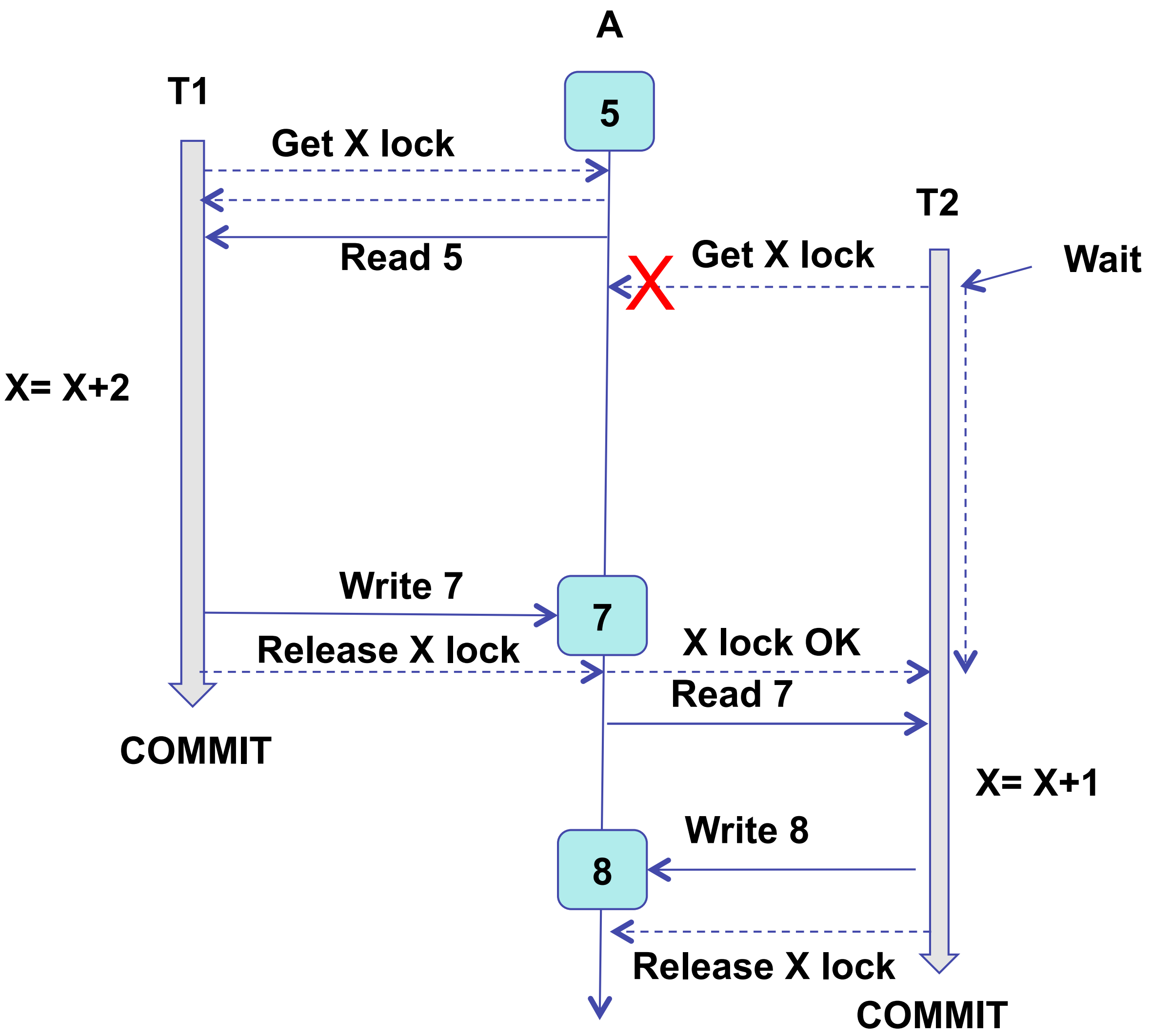


No isolation: the lost update anomaly



What is wrong?
Tell me the equivalent serial order of T1 and T2.

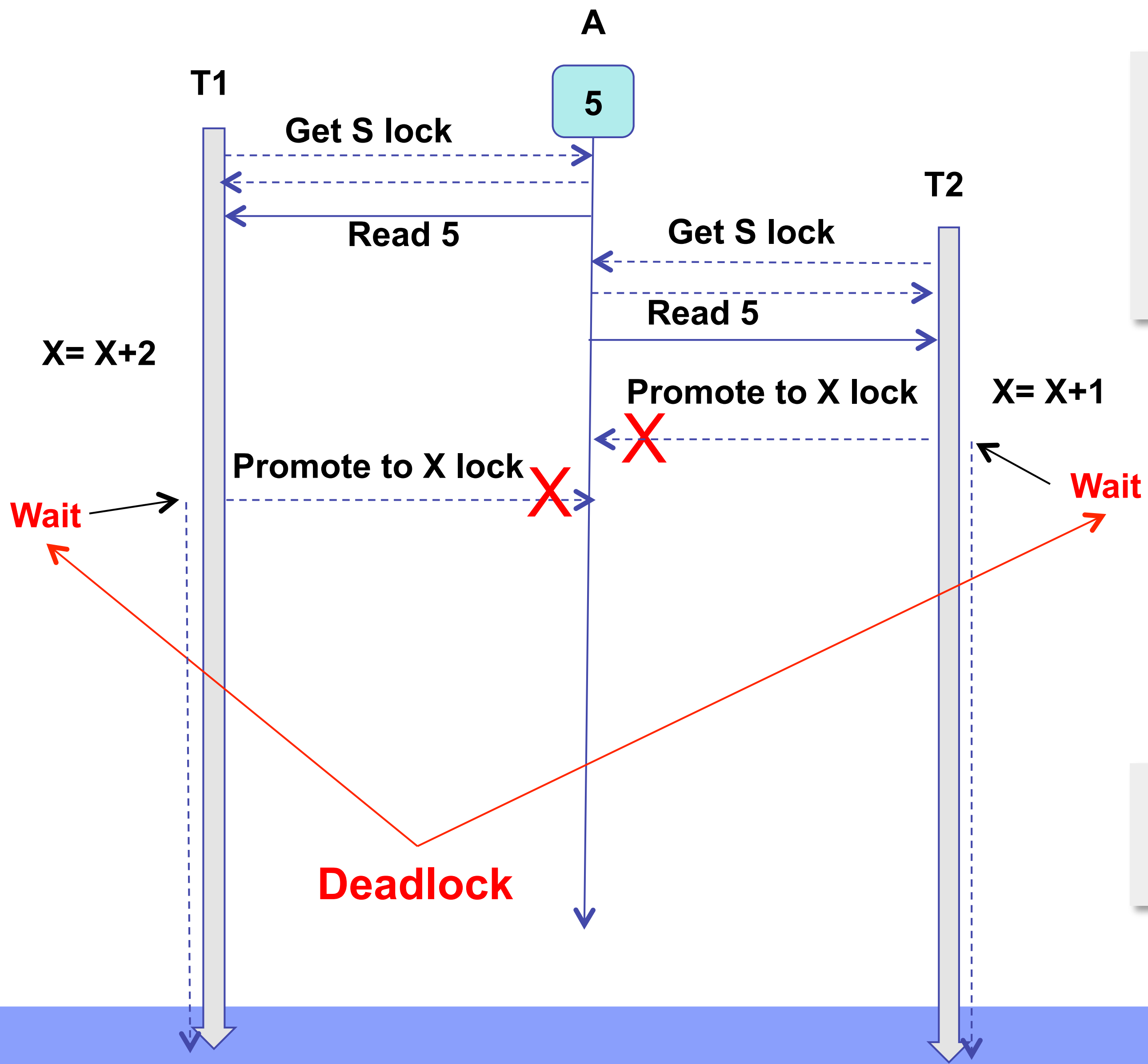
Isolation with locking: exclusive (X) locks



Isolation levels
READ COMMITTED
REPEATABLE READS
SERIALIZABLE
with
SELECT … FOR UPDATE

OK! The equivalent order is [T1, T2]

Isolation with locking: long shared (S) locks



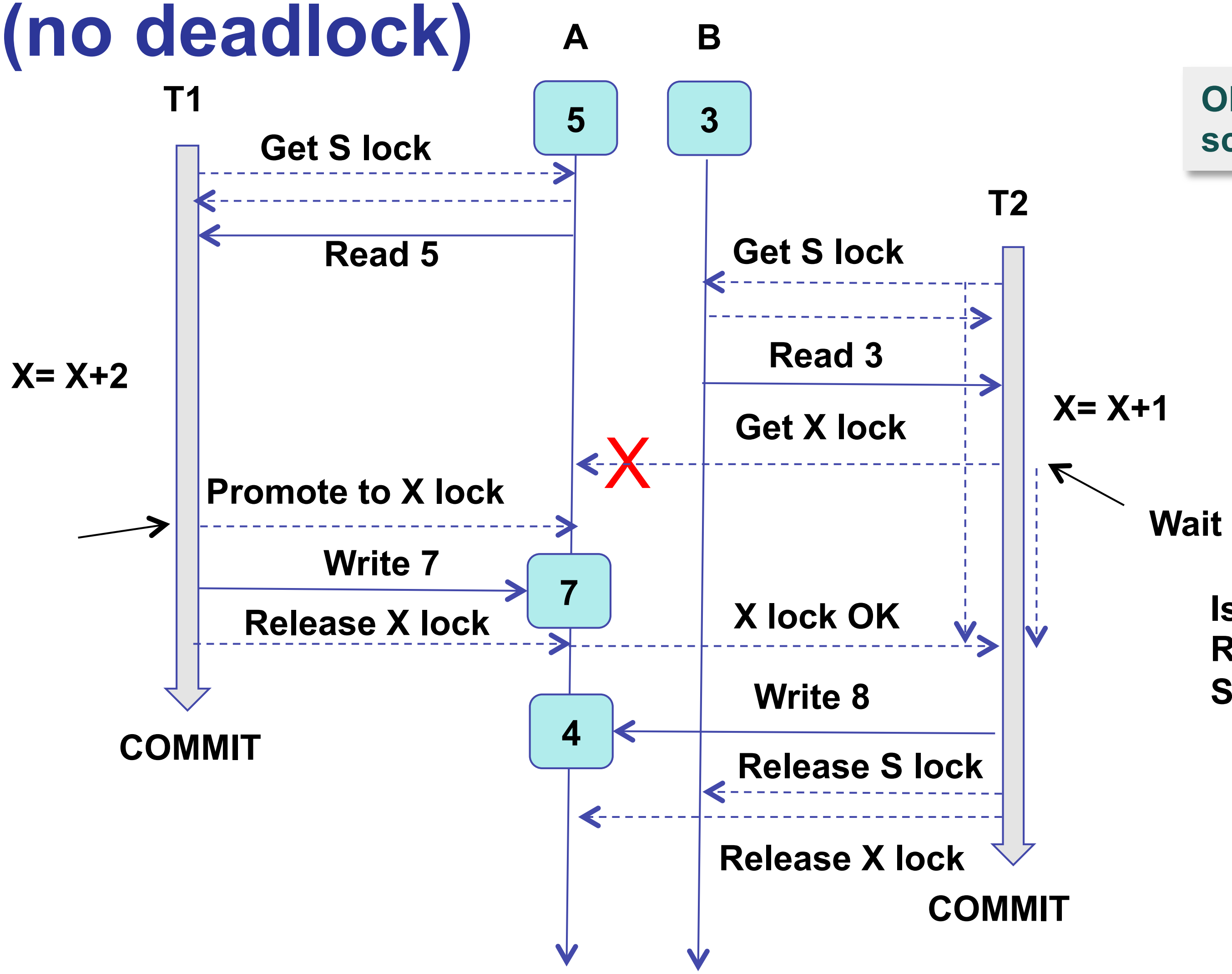
OK! A non-serializable schedule is blocked.

The deadlock is resolved by killing one of the transactions.

Isolation levels
REPEATABLE READ
SERIALIZABLE

We can only hope deadlocks will not appear

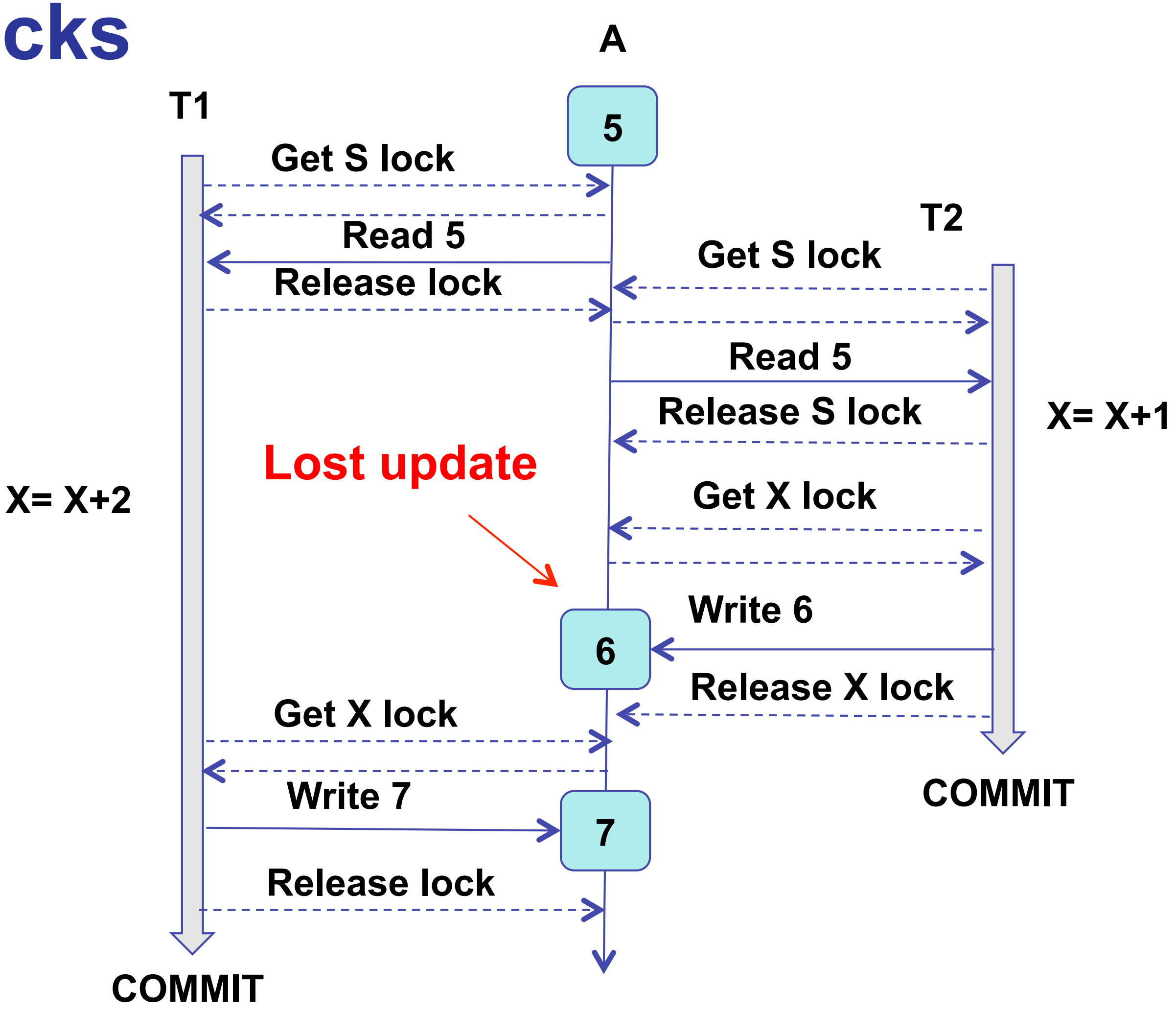
Isolation with locking: long shared (S) locks (no deadlock)



OK! The equivalent schedule is [T1, T2]

Isolation levels
REPEATABLE READ
SERIALIZABLE

Isolation with locking: short shared (S) locks



Isolation level
READ COMMITTED

The is no equivalent schedule of T1 and T2

BEWARE!

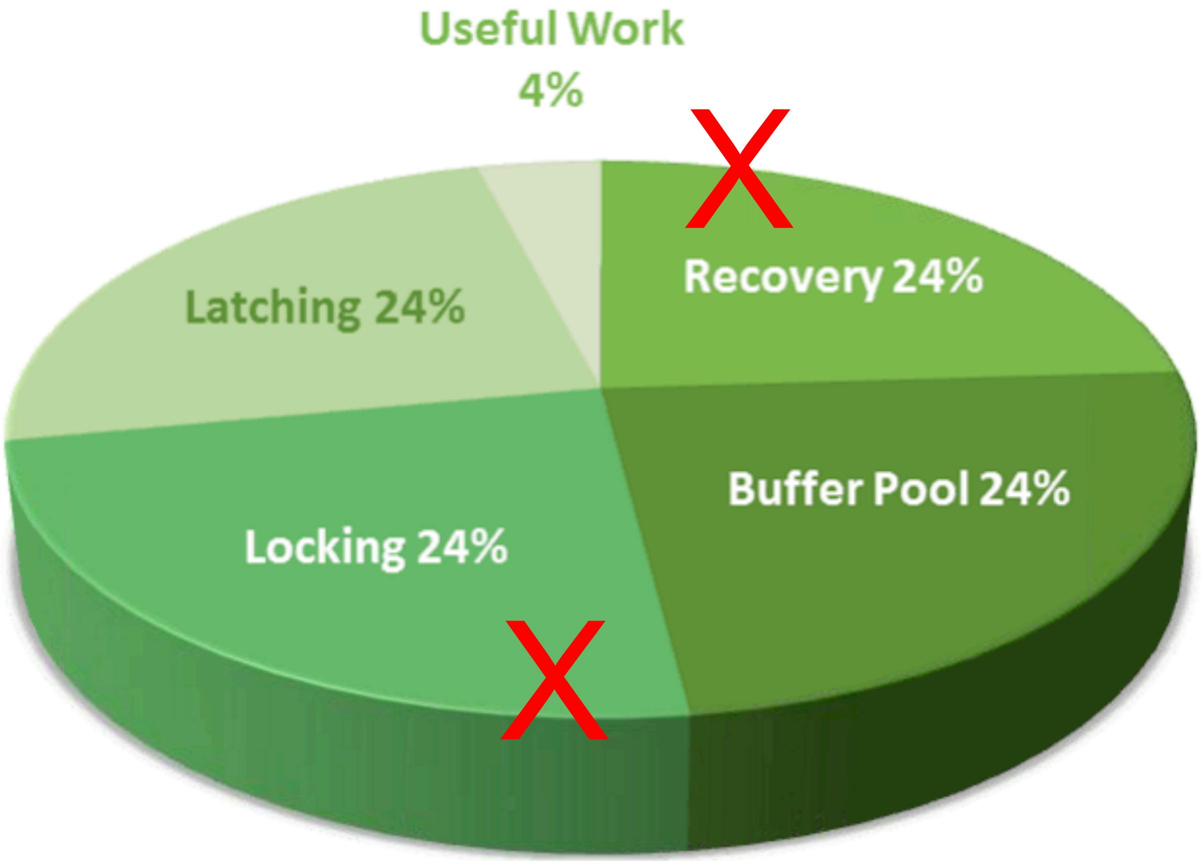
Isolation conclusions

- Beware of READ COMMITTED
  - good for systems with single writers
  - can you tolerate lost updates?
  - If not, use SELECT … FOR UPDATE, or UPDATE in place.
- If you can contain the deficiencies, READ COMMITTED is an efficient isolation level (the locks for the read-only items are short)
- READ COMMITTED with SELECT FOR UPDATE can produce serializable schedules if you read data items only once.
- REPEATABLE READ can produce serializable schedules if you ignore phantoms.
- SNAPSHOT isolation (if available) will prevent lost updates
- SERIALIZABLE isolation is conceptually best but heavy in operation

Isolation level scandal in U.K. in 1994

- In 1994, IT Week reported on a major clash between a British bank and a DBMS vendor (IBM).
- Because of the processing errors, the bank lost some of the asset transactions of its clients.
- The bank blame the vendor for an error in DBMS that "lost" the data.
- Later, it turned out the the bank used the CURSOR STABILITY isolation level (now: READ COMMITTED) without proper protection against lost updates.

Why everybody wants to escape the ACID straitjacket?



Source: M. Stonebraker, 2013

Is atomicity really needed?

- Atomicity is maintained with an undo log
- There is an overhead involved
- With atomicity, transactions last long, the locks stay longer → the concurrency is lower

Question:

- Can you replace multi-statement atomic transactions with single-statement transactions?

Decomposing transactions to smaller ones

How to replace multi-statement atomic transactions with a set of single-statement transactions (without losing atomicity)?

Supertransaction is a sequence of subtransactions.

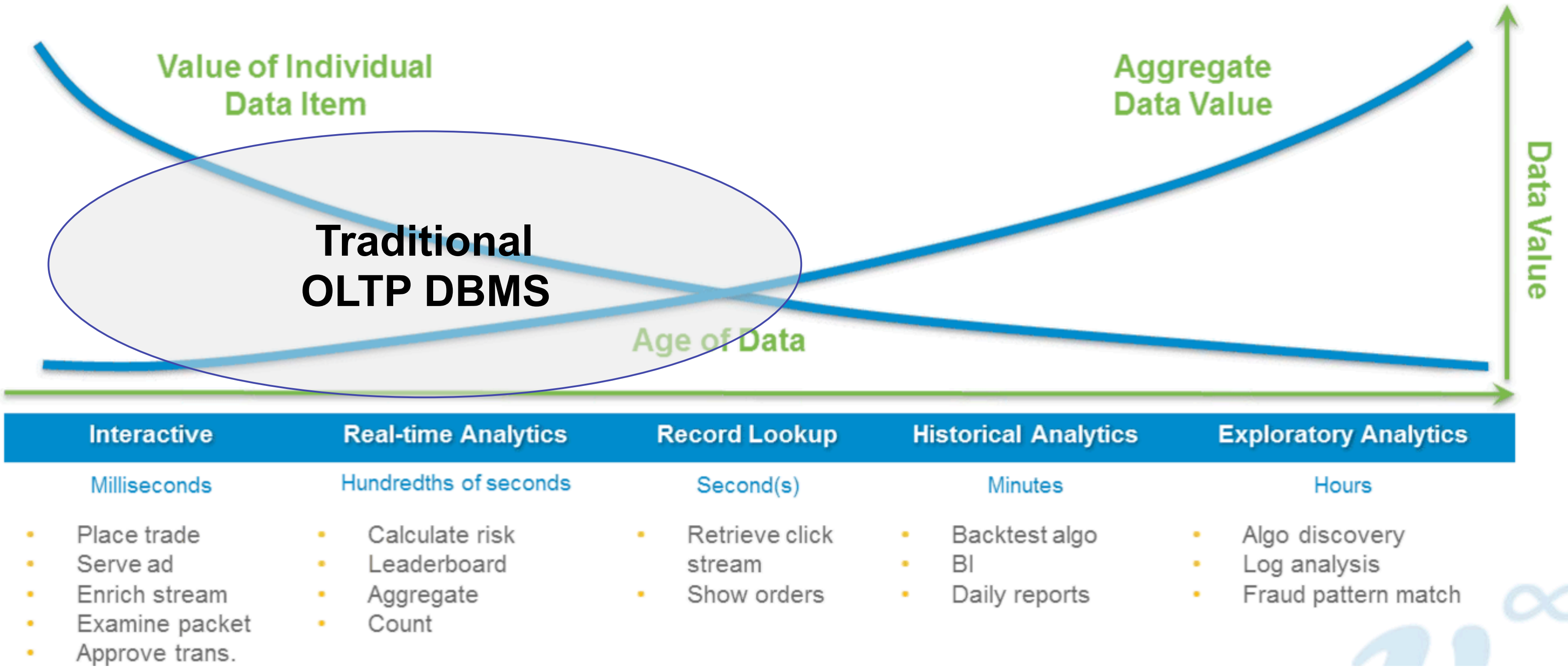
- Set the commit mode to AUTOCOMMIT
- For each of the statements, design a compensating statement, e.g. if it is INSERT, specify a corresponding DELETE.
- Execute your supertransactions this way:
  - In the first subtransaction(s), read all the data needed by the supertransaction (a read set), and store it for verification
  - In each next subtransaction, first check whether the input data is the same. If it is, execute the subtransaction, otherwise exit the supertransaction program block.
    - If everything is OK up to the last subtransaction, you are done.
- If there is a read set error or other subtransaction failure
  - For each successfully executed subtransaction, execute the compensating transaction.

Replace the undo log with compensating transactions

Problem: supertransactions are not serializable

Is durability really needed?

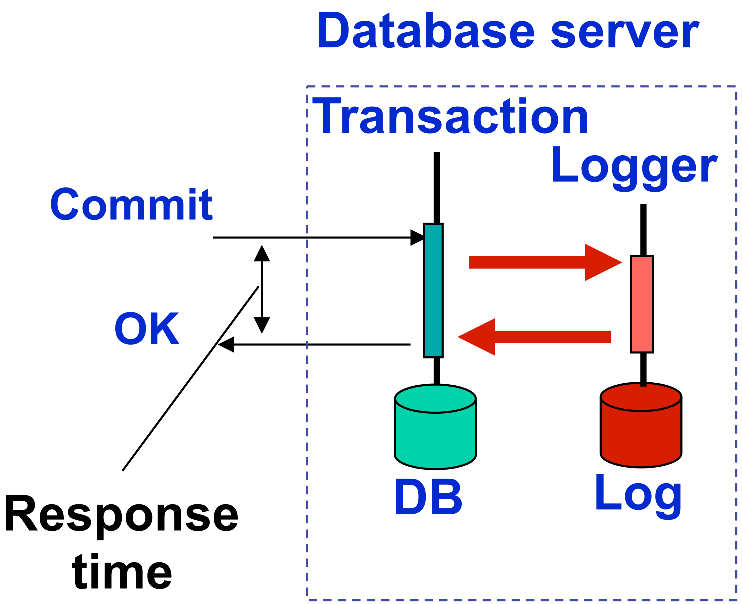
What is the value of a data item?



Source: M. Stonebraker, 2013

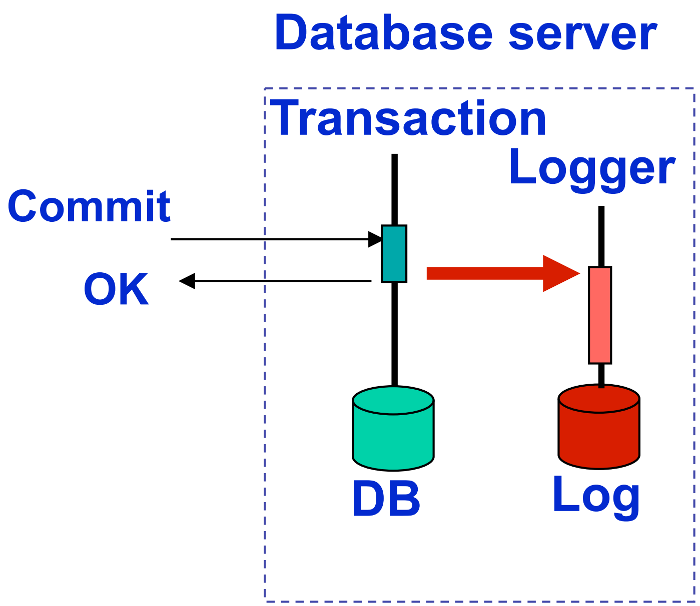
Strict and relaxed durability

Strict durability
Synchronous logging (write-ahead log, WAL)



Required for full ACID transactional behavior

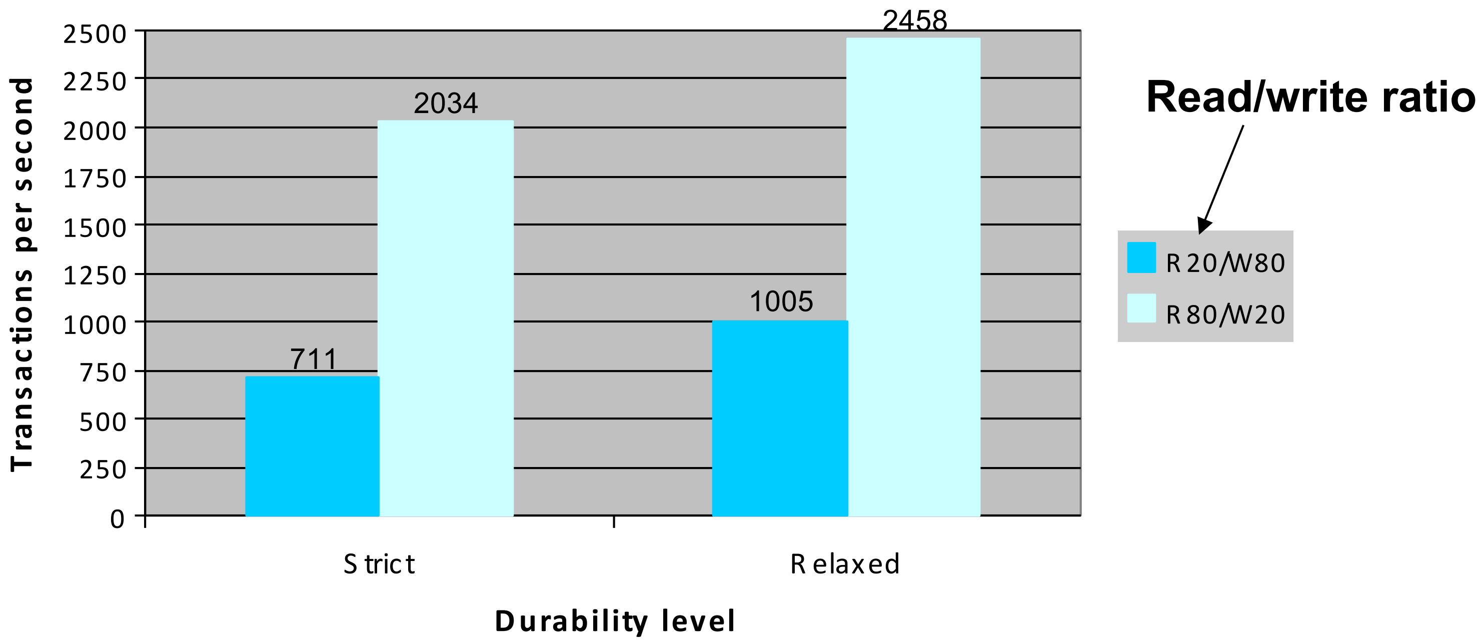
Relaxed durability
Asynchronous logging



This is often used because of the response time benefit

Impact of asynchrony of log writing on performance

The effect of relaxed durability level (asynchronous logging) on transaction throughp

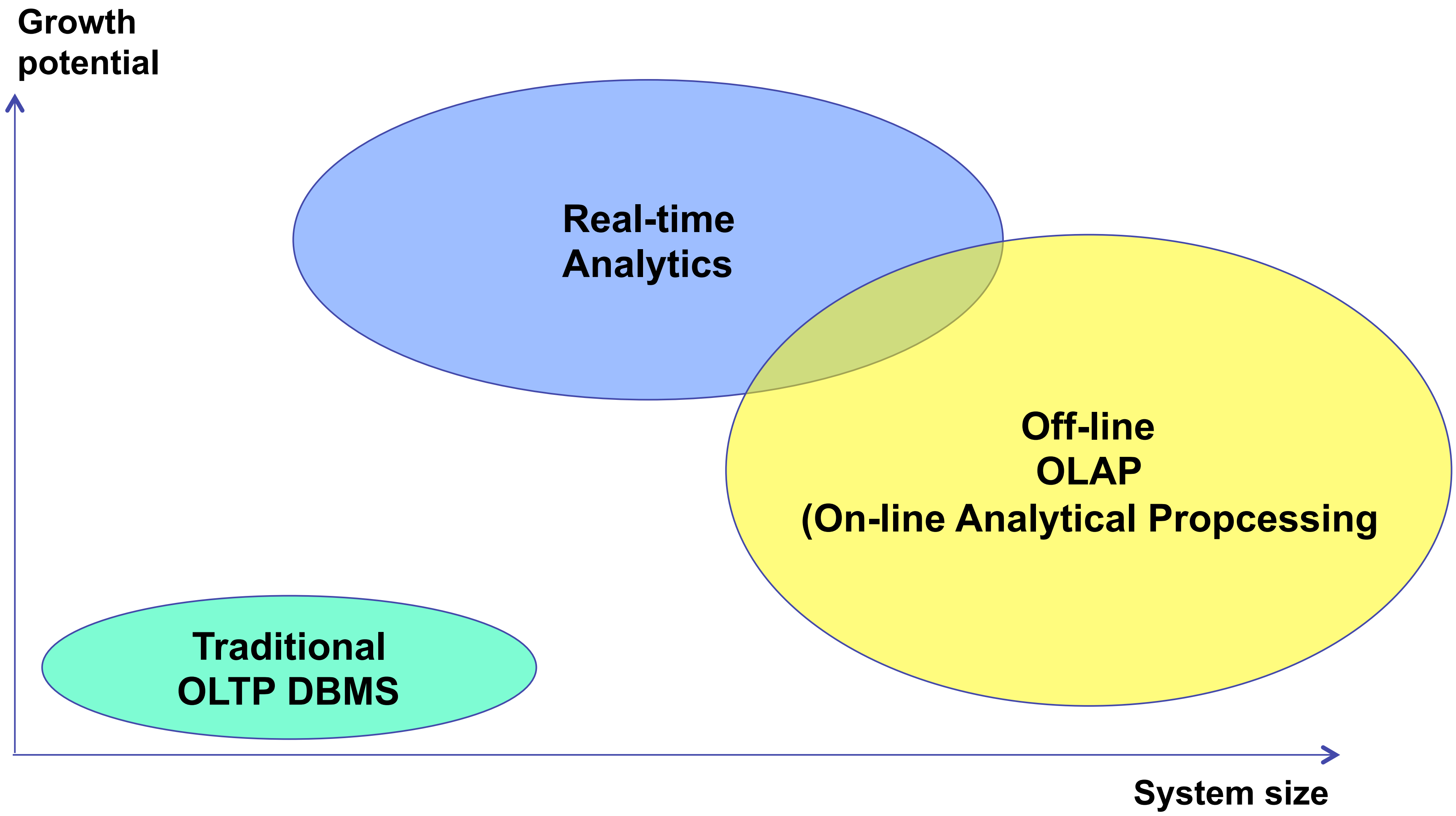


Risking transaction loss allows to increase throughput 20-40%.

When relaxed durability is OK?

- The quantified cost of of losing a few transactions is acceptable:
  - Example: Losing a few hundred billing records in a mobile network is OK (cost ca. few hundred euros)
- Results of single transactions have no value at all
  - In analytical processing the results are based on aggregates (AVG, SUM, MAX, MIN, statistical indicators, etc.)
- Can you do without a redo log?
  - How to restart? From checkpoint? Is that enough?
  - Some databases caontain only secondary data – can be recreated

Generally, how the data is used?



Big data

- **What**: data sets too large to be managed efficiently by DBMS
- **Where**: management of internet data (Google, Facebook), massive retail (Amazon), industrial measurement systems, meteorology, geology, satellite imaging, remote sensing, business intelligence, data warehousing, decision support systems.
- **Nature of data**: heterogeneous, semi-structured
- **Nature of metadata**: evolving schema
- **Data set sizes**: terabytes (10^{12}), petabytes (10^{15}), exabytes (10^{18}) and zettabytes (10^{21})
- **Needs**: fast access, scalability, high availability, eventual consistency
- **Known approaches**: key-values stores, MapReduce, distributed file systems (all have proprietary APIs – “NoSQL”)

Key-value store

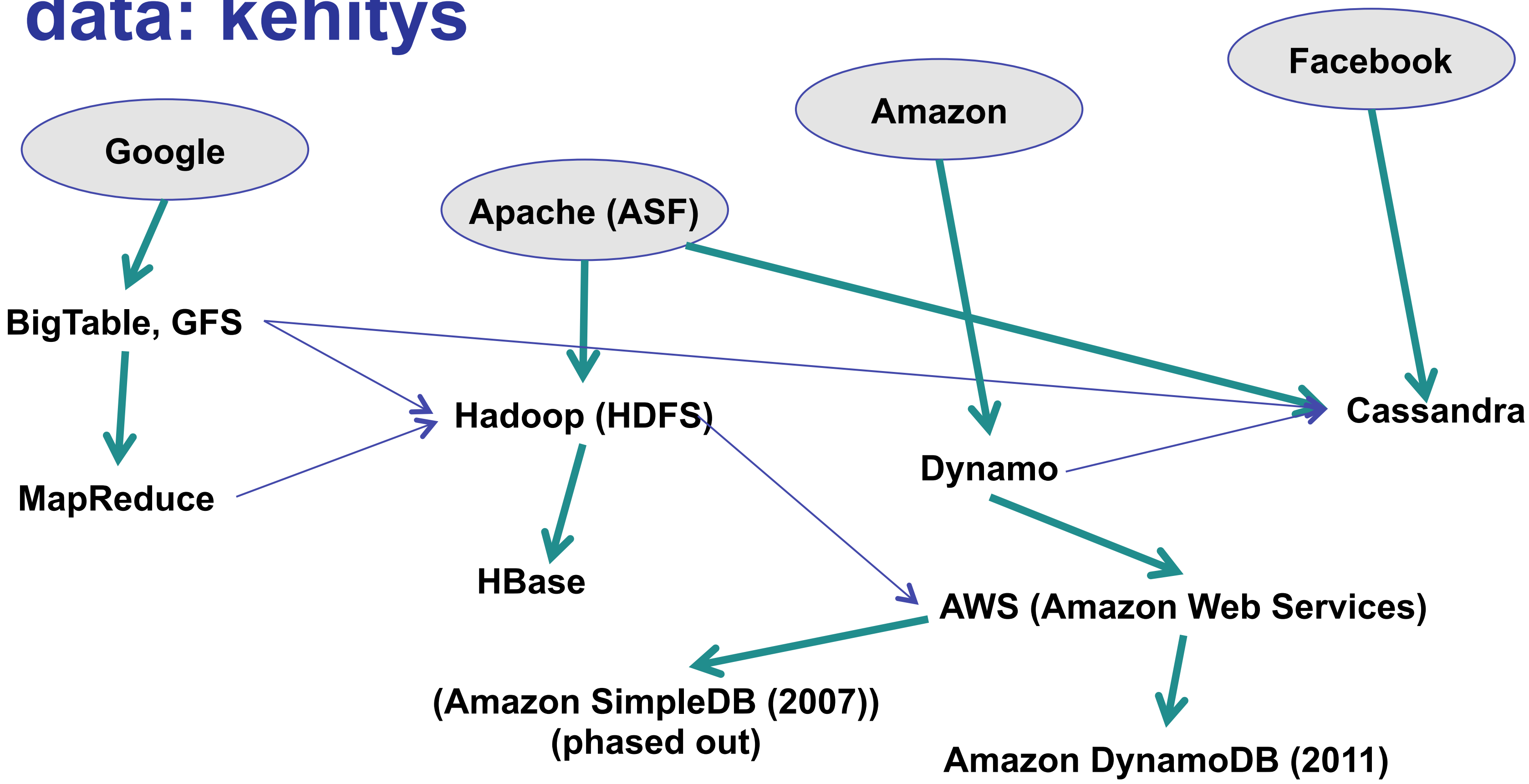
KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

Aug → 37.3

Value can be a BLOB or a complex structure

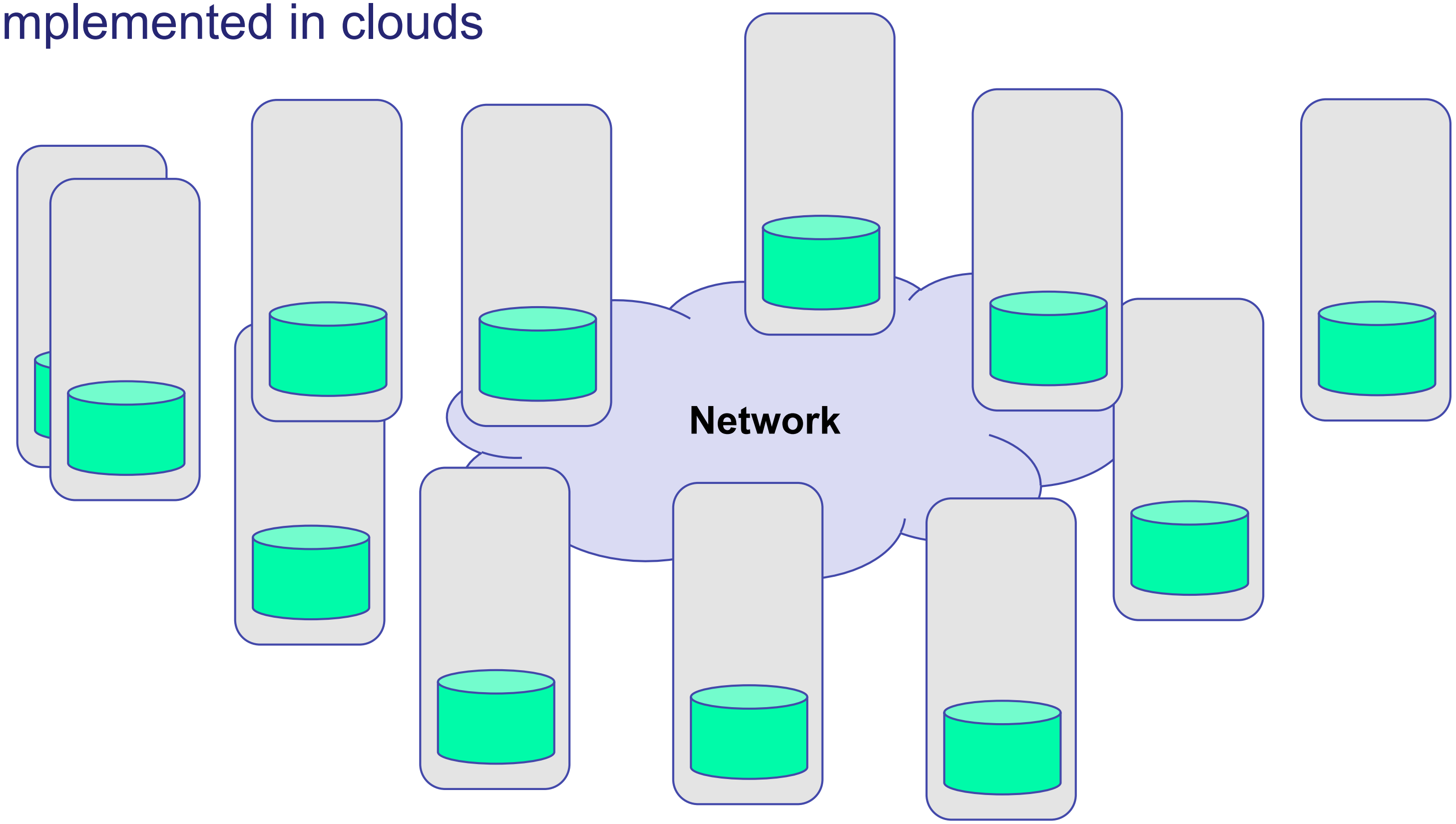
Key-value store is a two-domain relation

Big data: kehitys



A massive shared data system

- Loosely connected servers
- No synchronous protocols are possible (because of time constraints and performance
- Components (nodes) can fail, and the system can grow online
- Often implemented in clouds



CAP theorem

- CAP: three objectives: **C**onsistency, **A**vailability, **P**artitioning (P = resiliency to network partitioning)

(Eric Brewer, 2000)

- Theorem:

  Of the three objectives (C, A, P) only two can be met, at any single time, in a shared-data system.

- From ACID to BASE

  ACID: Atomicity, Consistency, Isolation, Durability is too restrictive

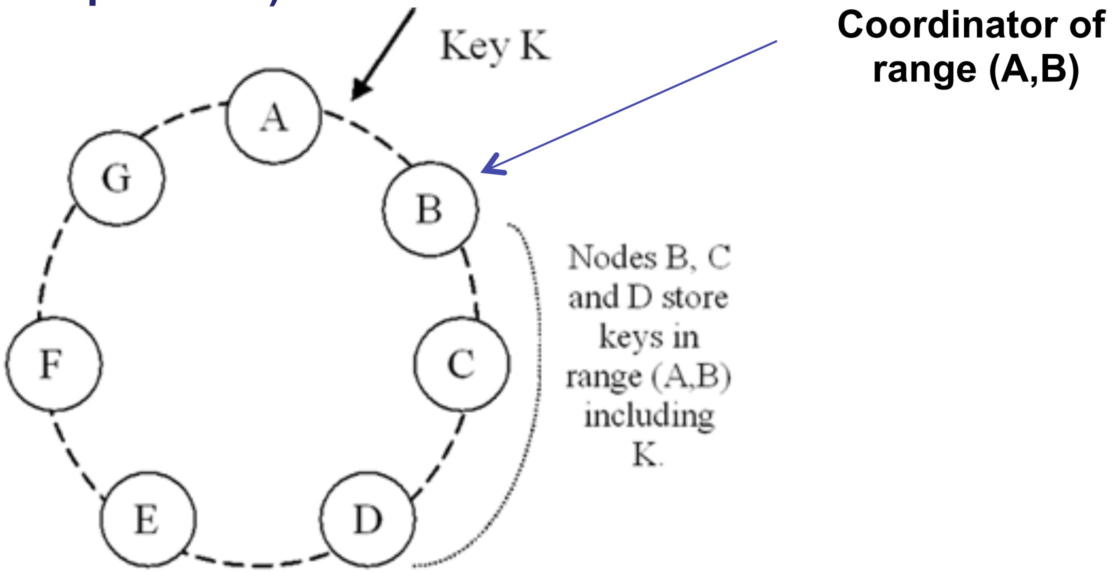
  The solution for big data is **BASE**:

  - **Ba**sically available
  - **S**oft-state ← = the current state in not consistent
  - **E**ventual consistency

Example: Amazon Dynamo

(Consistent Hashing with Replication)

Highly-available key-value store: the nodes can leave and join.

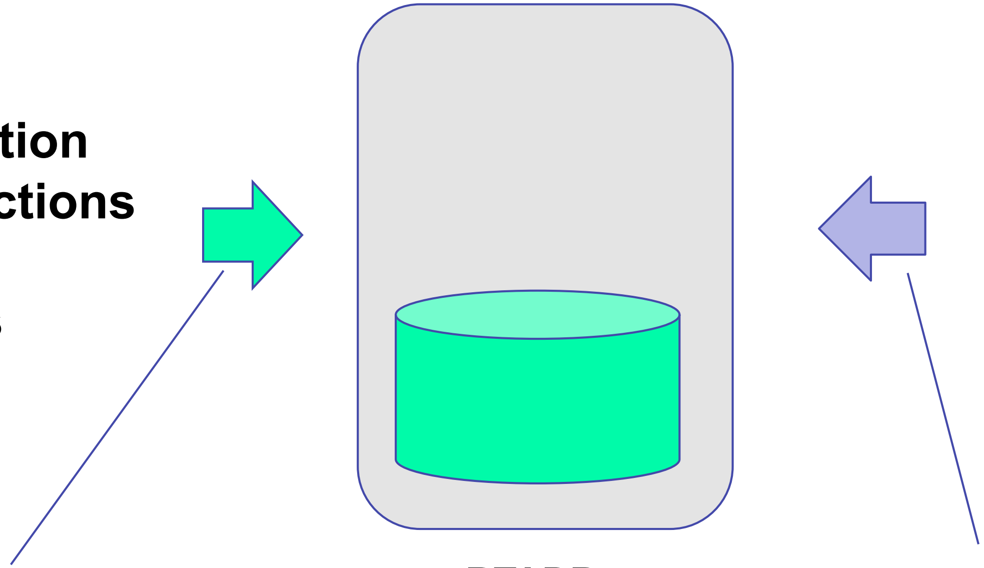


- Each key value has a coordinator node
- Coordinator node creates and manages replicas (here 3)
- A put() operation applies to a single node only
- All replicas can be updated: version based reconciliation (eventual consistency)
- Conflicts in branched versions initiate special processing (depending on the semantics of the data)
- Some operations are durable: synchronous replication to at least one node.

New challenge: real-time analytics database

Data acquistion transactions

Events



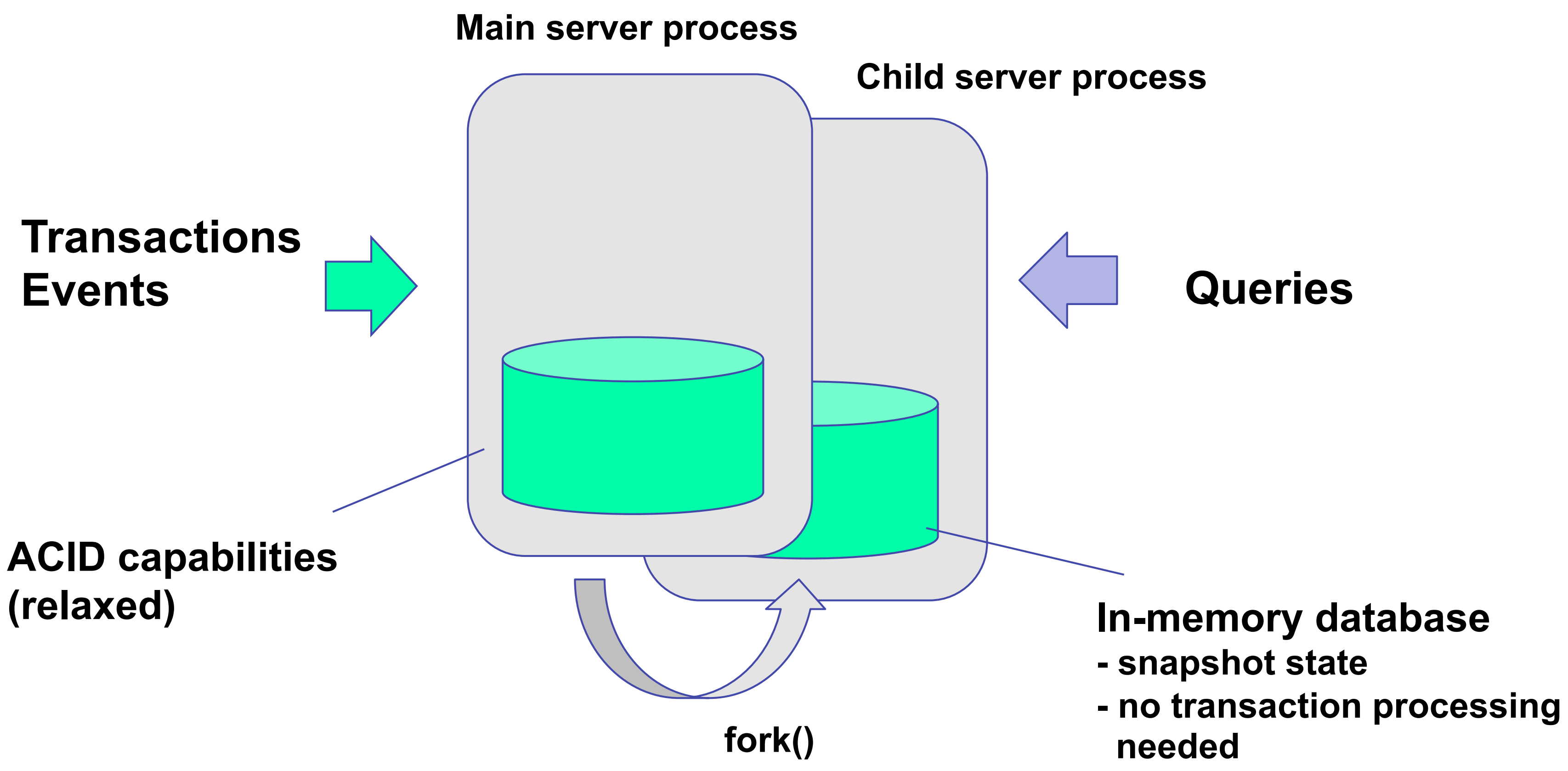
RTADB

Analytical queries

Requirements:
- **throughput**
- **limited atomicity (small transactions)**
- **isolation**
- **relaxed durability**

Requirements:
- **Throughput**
- **Result consistency**

RTADB can be solved – example: HyPer



Summary

- transaction concepts are the cornerstone of data processing
- you can relax the ACID capabilities when you understand them
- future data uses will incorporate both transactional and non-transactional processing

Bibliography

[BHG87] Philip Bernstein, Vassos Hadzilacos, Nathan Goodman. Concurrency control and recovery in database systems. Addison-Wesley Publishing Company, 1987.

[Ber95] Hal Berenson et al. A Critique of ANSI SQL Isolation Level. Proc. ACM SIGMOD 95, pp. 1-10, San Jose CA, June 1995.

[Bre00] Eric Brewer. Towards Robust Distributed Systems (kyenote talk). Proc. PODC 2000 (ACM Symposium on Principle of Distributed Computing). http://awoc.wolski.fi/dlib/big-data/Brewer_podc_keynote_2000.pdf

[GiLy02] Seth Gilbert, Nancy Lynch: Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services, ACM SIGACT News, 2002. http://awoc.wolski.fi/dlib/big-data/GiLy02-CAP.pdf

[GR92] Jim Gray and Adreas Reuter. Transaction Processing Systems, Concepts and Techniques. Morgan Kaufmann Publishers, 1992.

[Pri08] Dan Pritchet: BASE: An ACID Alternative. ACM Queue, May/June 2008. http://awoc.wolski.fi/dlib/big-data/Pritchett08-baseACID-acmqueue.pdf

[Strauch11] Strauch, C., Sites, U. L. S., & Kriha, W.: NoSQL databases. Lecture Notes, Stuttgart Media University, 2011. http://awoc.wolski.fi/dlib/big-data/strauch11-nosqldbs.pdf