# Transactions and Database Systems

Nick Walker, Jieliang Yin

# Database Management Systems (DBMS)

**Core Functions**

Protect the data store in the database

Provide correct and highly available access to the data

**Core Components**

Concurrency Control

Recovery

# Lots of jargon in this paper 📄

“Repeating history enables ARIES to employ a variant of the physiological logging (🙄) technique described earlier: it uses page-oriented REDO and a form of logical UNDO.”

# Transactions 📒

- Unit of work
- Either aborts or commits
- Abstraction that lets databases handle concurrent operations while maintaining some guarantees
- “ACID”
  - Atomic: all or nothing
  - Consistent: database integrity maintained
  - Isolation: one transaction doesn’t care about other concurrent transactions (concurrency)
  - Durability: once committed, will survive (fault tolerance)
- ACID ↔ Serializability

# SQL Example 👩‍💻

```
/* start a transaction */
BEGIN;

/* deduct 1000 from account 1 */
UPDATE accounts
SET balance = balance - 1000
WHERE id = 1;

/* add 1000 to account 2 */
UPDATE accounts
SET balance = balance + 1000
WHERE id = 2;

/* select the data from accounts */
SELECT id, name, balance
FROM accounts;

/* commit the transaction */
COMMIT;
```

# Concurrency: What's hard about this? 🚦🔒

**Q: Serializing transactions is slow, but how do we interleave execution correctly?**
A: Use locks.

**Q: What things can you reasonably lock? Tuples? Tables? Predicates (queries)?**
A: We'll have a multilevel locking system.

**Q: What if transactions get into a deadlock?**
A: Let's get into the deadlock detection business.

**Q: So we tried the locks, but then it wasn't much faster than serial execution…**
A: Only use *some* locks? 🤷

# Fault Tolerance: What's hard about this? 💣

**Q: What kinds of failure do you want to tolerate?**
A: Losing memory.

**Q: What do we need to do to restore the database after memory loss?**
A: We'll just *force* all transactions to be written to disk before committing, and persistently log what it would take to *undo* in case we need to abort some transactions after memory loss.

***Q:* We tried but it was really slow. What else can we do?**
A: Have everyone put the changes into a persistent log. If we lose memory, anything that wasn't committed just gets aborted, and anything committed can be *redone* via the log.

**Q: Now that the database can be stale on disk, we're running out of RAM holding chunks of it around before things get flushed to disk.**

A: ...

# ACID Transaction pitfalls 

- Works best with short transactions
- Not designed to be run with clients that can disconnect for a long time
- Doesn’t know about the application
- Convenient cudgel for recruiters
    - “What is ACID and how is it achieved?”

# Discussion 🗣️

1. ACID transactions: good or bad abstraction? Alternatives?
2. ACID transactions bundle a few concerns (Atomicity, Consistency, Isolation Durability). Is it still valuable to provide these properties together?
3. Today, we use cloud database services. Do they provide ACID? What challenges does the distributed case bring?
4. What are some other places where we'd like transaction semantics?

https://bit.ly/550-tx

intel® OPTANE™

PERSISTENT MEMORY

# Transactions: a history of research failure 🤦

“... it has become apparent over time that the **different characterizations [of relaxed consistency] in that paper were not specified to an equal degree of detail**. As pointed out in a recent paper by Berenson et al., the SQL-92 standard suffers  from a similar lack of specificity. Berenson et al. have attempted to clarify the issue but it is too early to determine if the have been successful.”

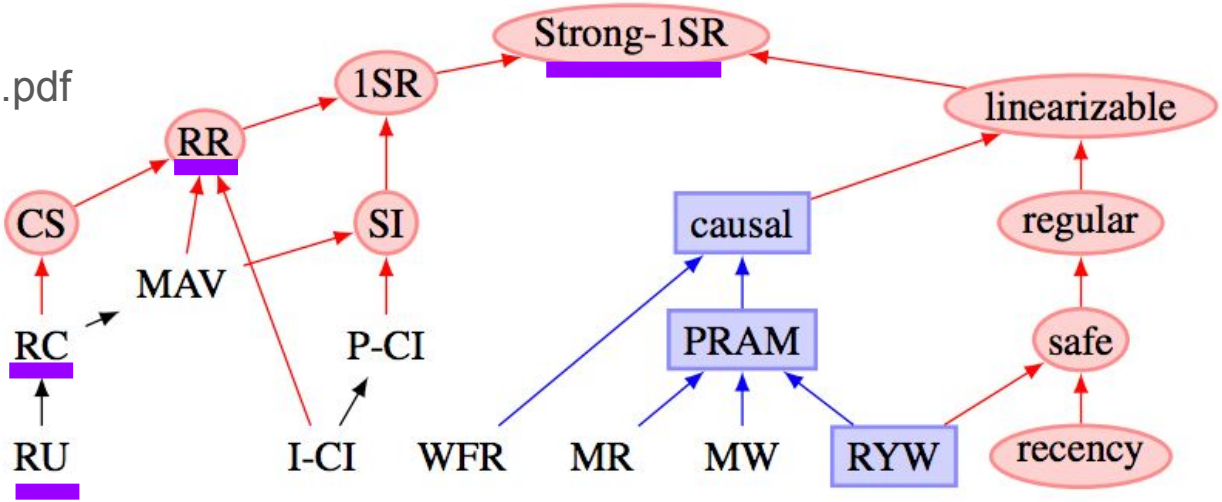
- The paper, in 1992

# Isolation in Practice

Turned out to be far more complicated than people thought:

“The current state of database software offers uncomfortable and unnecessary choices between availability and transactional semantics”

- Another paper, in 2013
https://arxiv.org/pdf/1302.0309.pdf



# Isolation in Practice 😧

“Despite the ubiquity of weak isolation, I haven’t found a database architect, researcher, or user who’s been able to offer an explanation of when, and, probably more importantly, *why* isolation models such as Read Committed are sufficient for correct execution. ... I don’t think we have any real understanding of how so many applications are seemingly (!?) okay running under them.”

http://www.bailis.org/blog/understanding-weak-isolation-is-a-serious-problem/

# Why might weak isolation work? 😬

1. For low-traffic and low-contention applications, it's possible that anomalies don't arise.
2. When anomalies do arise, it's possible that the read-write anomalies don't translate into application data corruption
3. It's possible data is actually (occasionally) corrupted, and apps just don't care

http://www.bailis.org/blog/understanding-weak-isolation-is-a-serious-problem/

# How relevant is ACID today? 🤔

- Flashback to last decade: NoSQL is hot
- “Come back to SQL, we have ACID”, they said
- People started to look carefully at “ACID compliance”

# Most common DBMSs not serializable by default 😳

http://www.bailis.org/blog/when-is-acid-acid-rarely/

| Database | Default Isolation | Maximum Isolation |
|---|---|---|
| Actian Ingres 10.0/10S | S | S |
| Aerospike | RC | RC |
| Akiban Persistit | SI | SI |
| Clustrix CLX 4100 | RR | ? |
| Greenplum 4.1 | RC | S |
| IBM DB2 10 for z/OS | CS | S |
| IBM Informix 11.50 | Depends | RR |
| MySQL 5.6 | RR | S |
| MemSQL 1b | RC | RC |
| MS SQL Server 2012 | RC | S |
| NuoDB | CR | CR |
| Oracle 11g | RC | SI |
| Oracle Berkeley DB | S | S |
| Oracle Berkeley DB JE | RR | S |
| Postgres 9.2.2 | RC | S |
| SAP HANA | RC | SI |
| ScaleDB 1.02 | RC | RC |
| VoltDB | S | S |
| **Legend** | *RC: read committed, RR: repeatable read, S: serializability, SI: snapshot isolation, CS: cursor stability, CR: consistent read* | |

# CAP theorem 🪦😭

Distributed data stores can only provide two of the following three guarantees

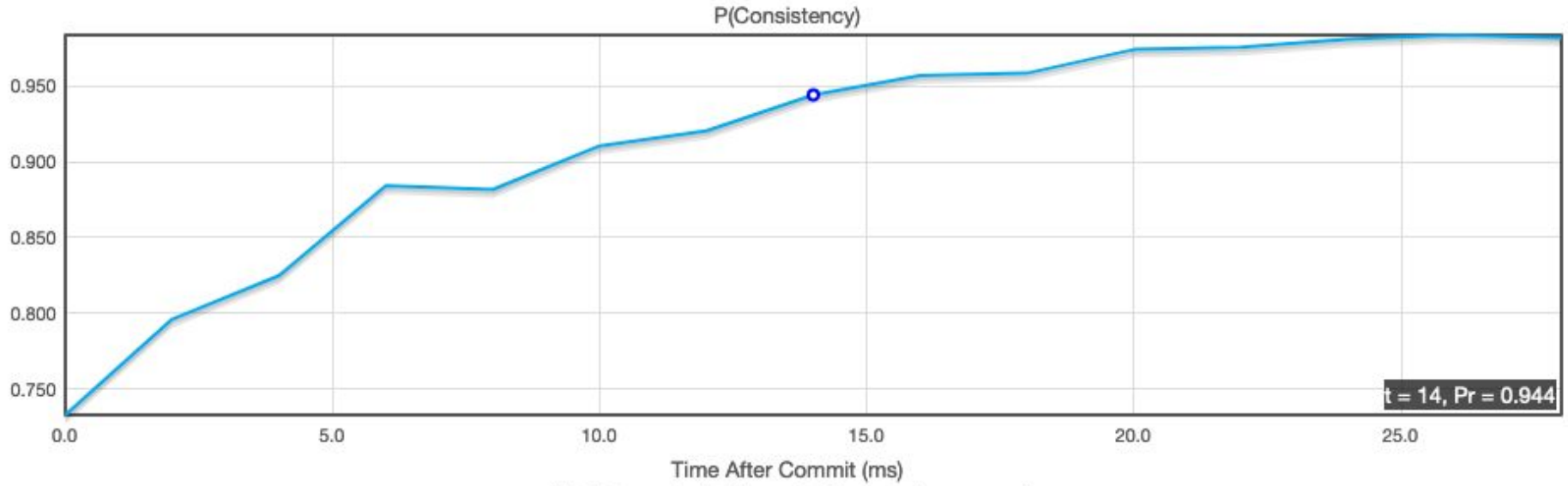
- **Consistency**: Every read receives the most recent write or an error.
- **Availability**: Every request receives a (non-error) response, without the guarantee that it contains the most recent write.
- **Partition tolerance**: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

# Eventual consistency (BASE 😉)

Distributed data stores can only provide two of the following three guarantees

- **Basically available:** reading and writing operations are available as much as possible but may not be consistent
- **Soft-state:** without consistency guarantees, after some amount of time, we only have some probability of knowing the state
- **Eventually consistent:** If we execute some writes and then the system functions long enough, we can know the state of the data; any further reads of that data item will return the same value

# How Eventual is Eventual Consistency? PBS in action under Dynamo-style quorums




http://pbs.cs.berkeley.edu/#demo

# RAMP-TAO: Layering Atomic Transactions on Facebook's Online TAO Data Store

"Facebook's graph store TAO, like many other distributed data stores, traditionally prioritizes availability, efficiency, and scalability over strong consistency or isolation guarantees to serve its large, read-dominant workloads. As product developers build diverse applications on top of this system, they increasingly seek transactional semantics..."

PVLDB 2021. doi:10.14778/3476311.3476379

# Transactional Memory 🧙

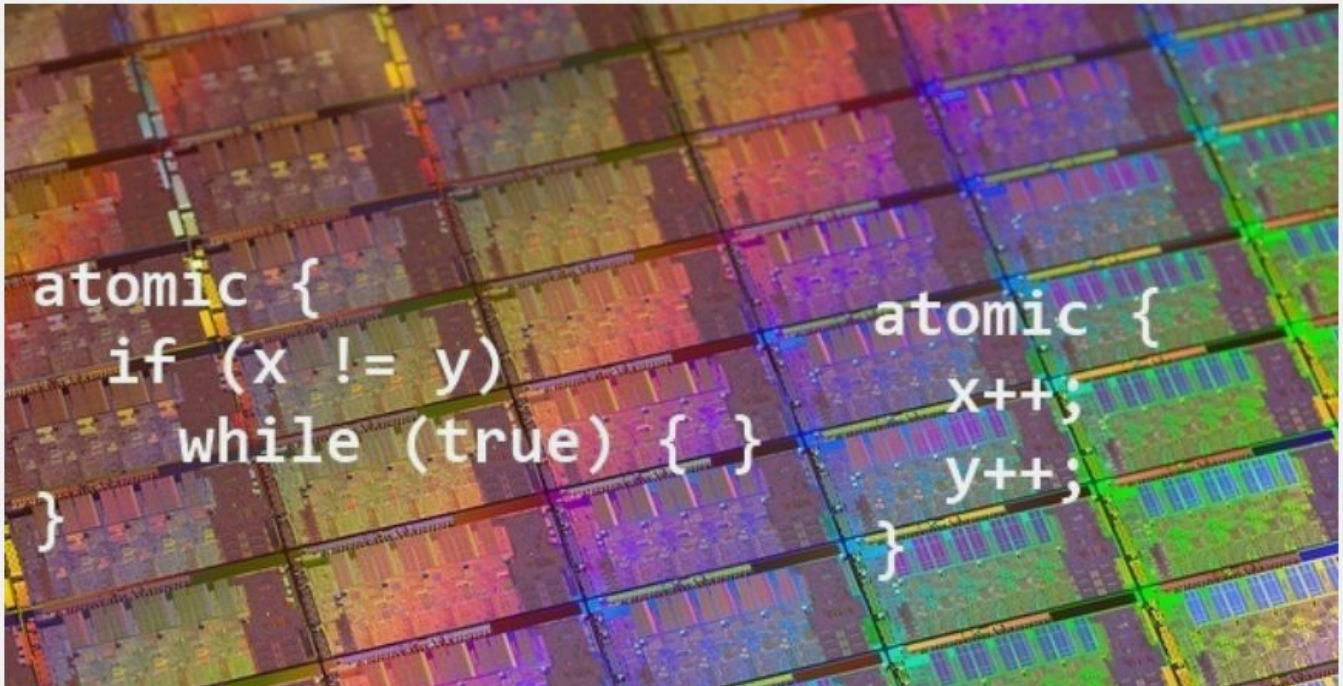
- Transaction semantics for batches of operations on shared memory
- Could be implemented in hardware or software
- Hasn’t seemed to pan out

BIZ & IT —

# Transactional memory going mainstream with Intel Haswell

Transactional memory is a promising technique for making the development of ...

ARS STAFF - 2/8/2012, 6:10 PM



# Developer Perspective 😞

“It didn't live up to that since the **max transaction size is really easy to overflow** ... and general code has a nasty habit of writing metrics even in read cases ... which means constant transaction conflicts, which means slower code than just using a lock. Cliff Click then makes the argument that the **right move in the vast majority of cases ... is to rewrite the algorithm so that threads (or at least cores) communicate via messages and share nothing rather than trying to use transactional memory**”

Monocasa, HN, 2021 - https://news.ycombinator.com/item?id=27667027

# In Transactional Memory, No One Can Hear You Scream

*Attacking Intel's Transactional Synchronization Extensions*

June 26, 2019 / Markus Gaasedelen

*BAD GUYS RUIN EVERYTHING —*

# Researchers use Intel SGX to put malware beyond the reach of antivirus software

Processor protects malware from attempts to inspect and analyze it.

ARS STAFF - 2/12/2019, 12:54 PM

# Intel To Disable TSX By Default On More CPUs With New Microcode

Written by Michael Larabel in Intel on 28 June 2021 at 12:18 PM EDT. 33 Comments


Intel is going to be disabling Transactional Synchronization Extensions (TSX) by default for various Skylake through Coffee Lake processors with forthcoming microcode updates. Yes, this does mean performance implications for workloads benefiting from TSX. This change has seemingly not been talked about much at all publicly and I just happened to become aware of it when looking through new kernel patches.

# Database VS Blockchain

| | **Database** | **Blockchain** |
|---|---|---|
| Distribution | Centralized | Decentralized |
| Manipulation | Mutable | Immutable |
| Structure | Table | Time series |
| Access | divided and unequal | Unified and equal |
| Property | No | Yes |
| Performance | High | Low |

# Database + Blockchain

## **BigchainDB**: The database without single control, own and failure

**Decentralization**
No single point of control. No single point of failure. Decentralized control via a federation of voting nodes makes for a P2P network.

**Immutability**
More than just tamper-resistant. Once stored, data can’t be changed or deleted.

**Byzantine Fault Tolerant (BFT)**
Up to one third of the nodes in the network can be experiencing arbitrary faults and the rest of the network will still come to consensus on the next block.

**Query**
Write and run any MongoDB query to search the contents of all stored transactions, assets, metadata and blocks. Powered by MongoDB itself.

**Low Latency**
A global network takes about a second to come to consensus on a new block. In other words, transaction finality happens fast.

**Rich Permissioning**
Set permissions at transaction level to ensure a clear separation of duties and enforce selective access.

# Questions

1. What are other differences between blockchain and database?
2. Is it possible to merge blockchain with database?
3. Is there a winner? Blockchain or database, or the combination?