

# **Is there a gap between DB theory and TM theory?**

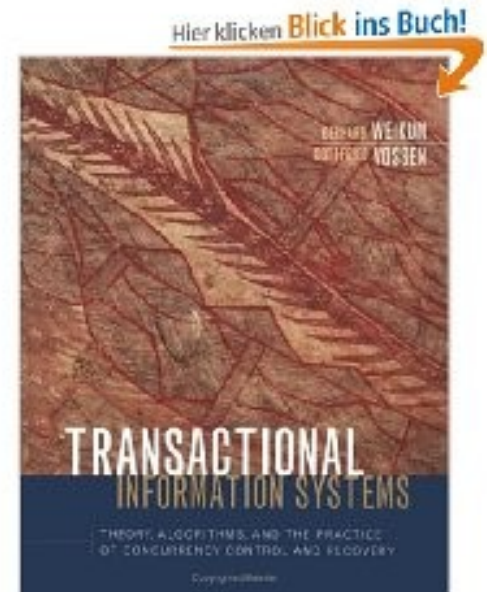
Torvald Riegel  
(TU Dresden, Germany)

# We don't want any gap to exist

- DB bigger than TM: more research, larger community
- DB theory has been in curricula
- DB background can help us connect to other areas:
  - On a technical level: connect to DBs, ...
  - More TM experts
- Especially important for TM theory:
  - We want many people to be able to understand TM theory
  - New names, new models, ... make this harder
  - We don't have enough time to invent from scratch

# DB theory

- Basic perspective: DB state matters most
- Implemented first, formalized later. (Sounds familiar?)
- Don't care much about variants that have no useful (DB) implementation
- First models from ancient times, a lot of refinements later  
→ recent models are better!
- Base for this talk:
  - Weikum and Vossen, 2001:  
Transactional Information Systems



# DB theory basics

- **Schedule(/history):**
  - Models “what is computed based on which input”
  - Consists of begin, operations, commit (for each txn)
  - Ops can be load/store (“page model”) or any other function (“object model”)
  - Partially or totally ordered
  - Inverse operations used to model roll-back
- **Scheduler** performs concurrency control
  - Decides whether schedules are allowed or not (and thus forcing txns to abort)

# Serializability: Equivalence to serial schedule

- View Serializability (VSR)
  - Txns must have same reads-from relation as in serial schedule
  - Testing this is NP-complete
- Conflict Serializability (CSR)
  - Data operations can conflict (eg, R/W, inc()/get())
  - Schedule must have same conflicts as serial schedule
  - Can be efficiently tested
  - CSR can (and should) be tested for prefixes of schedules too (including operations of active txns)
- Order-preserving CSR (OCSR):
  - CSR + non-overlapping txns keep RT order (sounds familiar?)
- Example result:  $VSR \supset CSR \supset OCSR$

# Multi-Version Serializability

- Extensions to multiple object versions: MCSR
  - Basically, multi-version schedule is in MCSR if it can be reordered/reduced to single-version CSR schedule with same conflicts
  - Shows confusing dual-use of schedules
    - Reasoning about correctness
    - Reasoning about ability/complexity of algorithms
    - We can use the “effective” single-version schedules to check STM correctness

# Misconceptions

- Write-buffering / lazy acquisition not allowed?
  - Schedules model how txns interact with shared state
  - Write-buffering STMs make writes visible before commit...
- Invisible reads are not covered?
  - Schedule models effective synchronization, not necessarily what the implementation does
  - E.g., invisible read = could have had a read lock. This can be easily modeled
- DB is about plain reads/writes only?
- Irrevocable txns cannot be modeled (with inverse operations)?

# DB theory basics: Recovery

- DB: make recovery after system crashes possible (and/or easier)
- TM: no system crashes, but:
  - When is it safe to return result of a txn?
  - When is it safe to use the result of a (uncommitted) txn?
- Additional restrictions on schedules (orthogonal to Serializ.):
  - Recoverability: Don't commit until inputs are committed
  - Avoid Cascading Aborts: Don't read uncommitted data
  - Strictness: Don't read/overwrite uncommitted data
  - Rigorousness: Strict + don't overwrite objects that are being read

# Three Aspects of TM Correctness

- **What do we compute, based on which inputs?**
  - Serializability!
- **Relation to real-time?**
  - Additional constraints on allowed inputs
  - DB theory is not very detailed here
  - However, we only need to cover begin+commit (RT invocation+response and begin/commit in schedule)
- **When is it safe to speculatively compute?**
  - Currently: never (e.g., TinySTM, Opacity) or always (e.g., JudoSTM)
  - DB theory: recoverability
  - Needs more research!

# A Rough “DB-based” TM Correctness Criterion

- What do we compute, based on which inputs?
  - (1) Multi-version CSR or (2) require reduction to single-version CSR
  - MVSR (aka 1-copy serializable) might not be necessary, we restrict it anyway for safety
  - Run CSR on prefixes of schedules!
- Real-time order:
  - (1) Do order-preserving MCSR
  - Or (2) restrict reduction/reordering to single-version schedule
- Safety: No speculative execution
  - Avoid Cascading Aborts or Strictness
- Opacity basically specifies this – but makes it look like there is a gap between DB and TM theory

# Example: Are DSTM and TinySTM correct?

- Both create strong two-phase locking (SS2PL) schedules:
  - Keep all locks until commit time
- SS2PL schedules are in COCSR
  - Txns are linearizable
- SS2PL schedules are rigorous
  - Safe, non-speculative execution
- TinySTM read-only is correct too (takes snapshot “in the past”):
  - Transformation to single-version schedule is SS2PL
  - Effective commit time earlier than commit request by user

# Multi-Level Concurrency Control

- Concurrency Control on different levels of abstraction
- Dates back to Open Nesting (80ies)
- Examples:
  - Two inc() methods that are implemented using TM do not have to be in conflict
  - Run DB txns on DB that uses TM
  - Transactional Boosting (datastructure-specific abstract locking + linearizable base objects)
  - Implementing txnal external actions
- Related DB theory is useful. E.g., Is deadlock possible with Boosting and/or Open Nesting?

# Conclusion

- Please have a look at recent DB theory, it's worth it
  - Clarify from/for TM perspective
  - Look for similarities and reuse, don't reject it at first sight
- There is no real gap between TM and DBs, unless we create it.