CS 6530: Advanced Database Systems Fall 2024

Lecture 06
Concurrency control #2

Prashant Pandey

prashant.pandey@utah.edu

Acknowledgement: Slides taken from Prof. Andy Pavlo, CMU

SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

CONCURRENCY CONTROL

- The system assumes that a txn could stall at any time whenever it tries to access data that is not in memory.
- Execute other txns at the same time so that if one txn stalls then others can keep running.
  - Set locks to provide ACID guarantees for txns.
  - Locks are stored in a separate data structure to avoid being swapped to disk.

ACID guarantee

- **Atomicity** - each statement in a transaction (to read, write, update or delete data) is treated as a single unit. Either the entire statement is executed, or none of it is executed.
- **Consistency** - ensures that transactions only make changes to tables in predefined, predictable ways
- **Isolation** - when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don’t interfere with or affect one another.
- **Durability** - ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure.

STORAGE ACCESS LATENCIES

	L3	*DRAM*	*SSD*	*HDD*
Read Latency	~20 ns	60 ns	25,000 ns	10,000,000 ns
Write Latency	~20 ns	60 ns	300,000 ns	10,000,000 ns

CONCURRENCY CONTROL

- The protocol to allow txns to access a database in a multi-programmed fashion while preserving the illusion that each of them is executing alone on a dedicated system.
  - The goal is to have the effect of a group of txns on the database's state is equivalent to any serial execution of all txns.

- Provides **A**tomicity + **I**solation in ACID

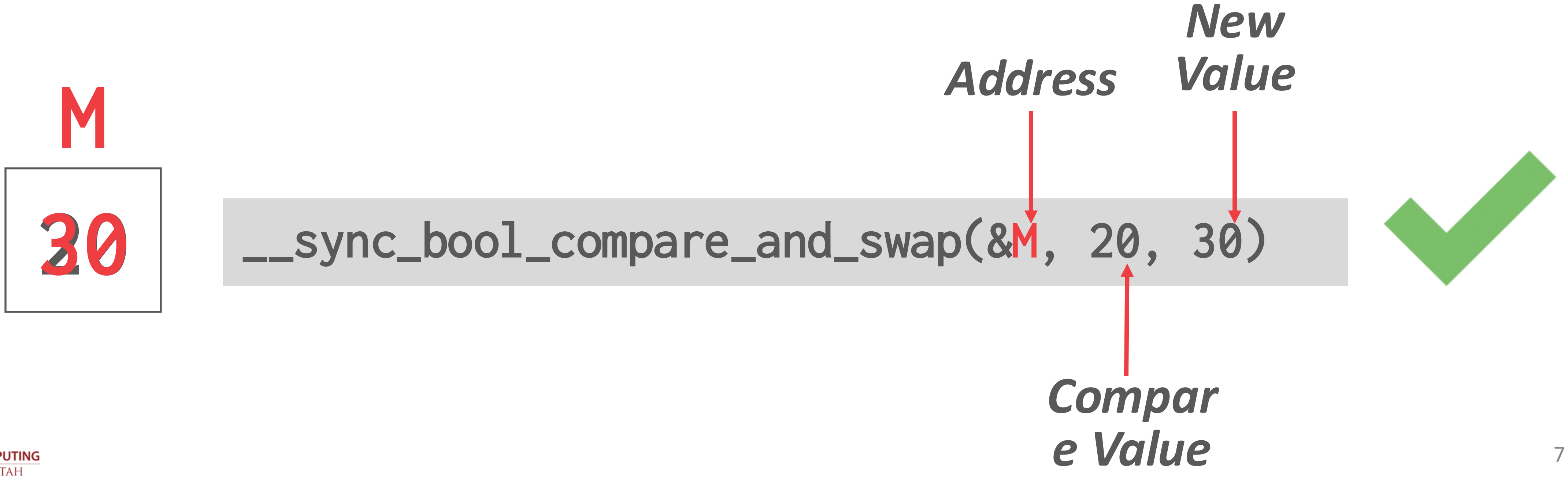
CONCURRENCY CONTROL

- For in-memory DBMSs, the cost of a txn acquiring a lock is the same as accessing data.
- New bottleneck is contention caused from txns trying access data at the same time.

- The DBMS can store locking information about each tuple together with its data.
  - This helps with CPU cache locality.
  - Mutexes are too slow. Need to use **compare-and-swap** (CAS) instructions.

COMPARE-AND-SWAP

- Atomic instruction that compares contents of a memory location M to a given value V
  - If values are equal, installs new given value V’ in M
  - Otherwise operation fails

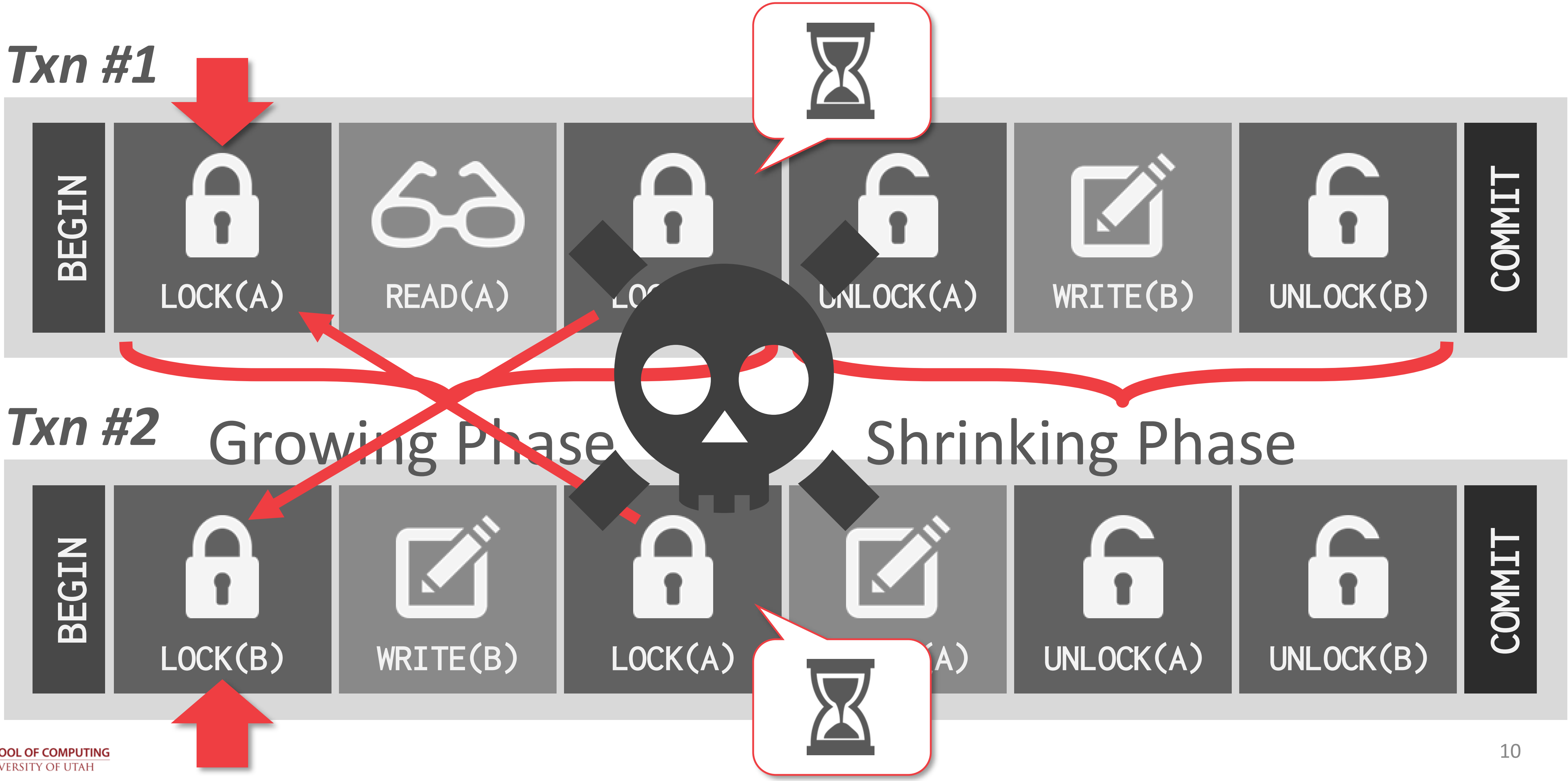


CONCURRENCY CONTROL SCHEMES

- **Two-Phase Locking (2PL)**
  - Assume txns will conflict so they must acquire locks on database objects before they are allowed to access them.

- **Timestamp Ordering (T/O)**
  - Assume that conflicts are rare so txns do not need to first acquire locks on database objects and instead check for conflicts at commit time.

TWO-PHASE LOCKING



TWO-PHASE LOCKING

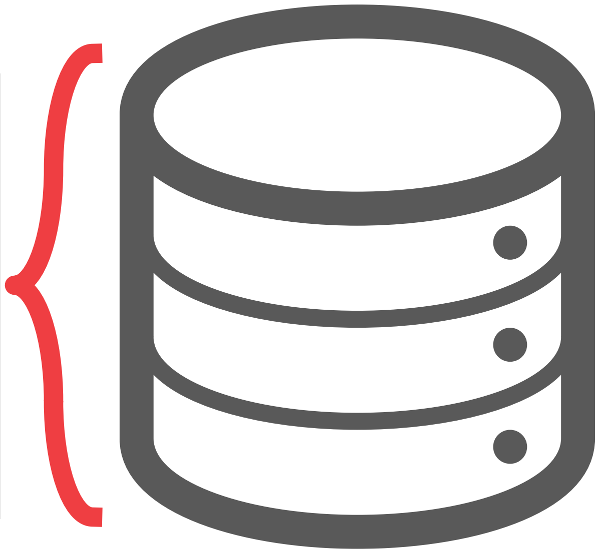
- **Deadlock Detection**
  - Each txn maintains a queue of the txns that hold the locks that it is waiting for.
  - A separate thread checks these queues for deadlocks.
  - If deadlock found, use a heuristic to decide what txn to kill in order to break deadlock.
- **Deadlock Prevention**
  - Check whether another txn already holds a lock when another txn requests it.
  - If lock is not available, the txn will either (1) wait, (2) commit suicide, or (3) kill the other txn.

TIMESTAMP ORDERING

- **Basic T/O**
  - Check for conflicts on each read/write.
  - Copy tuples on each access to ensure repeatable reads.
- **Optimistic Currency Control (OCC)**
  - Store all changes in private workspace.
  - Check for conflicts at commit time and then merge.

BASIC T/O

Record	Read Timestamp	Write Timestamp
A	10001	10005
B	10000	10001



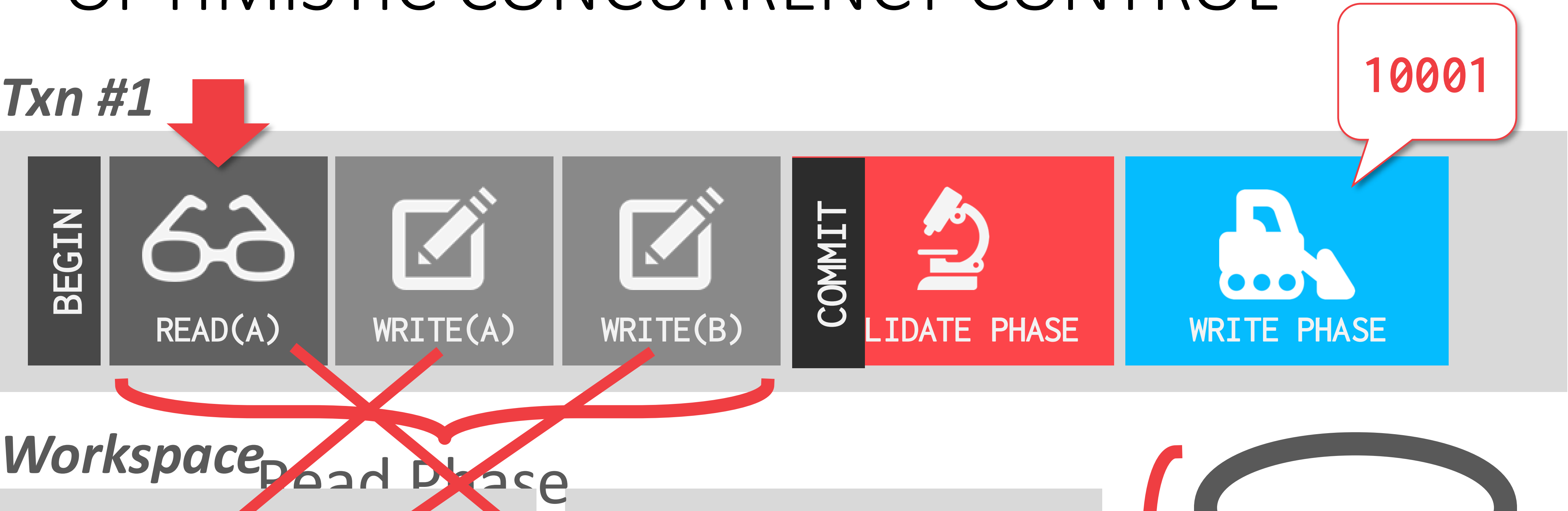
OPTIMISTIC CONCURRENCY CONTROL

- Timestamp-ordering scheme where txns copy data read/write into a private workspace that is not visible to other active txns.
- When a txn commits, the DBMS verifies that there are no conflicts.

- First proposed in 1981 at CMU by H.T. Kung.

ON OPTIMISTIC METHODS FOR CONCURRENCY CONTROL
ACM TRANSACTIONS ON DATABASE SYSTEMS 1981

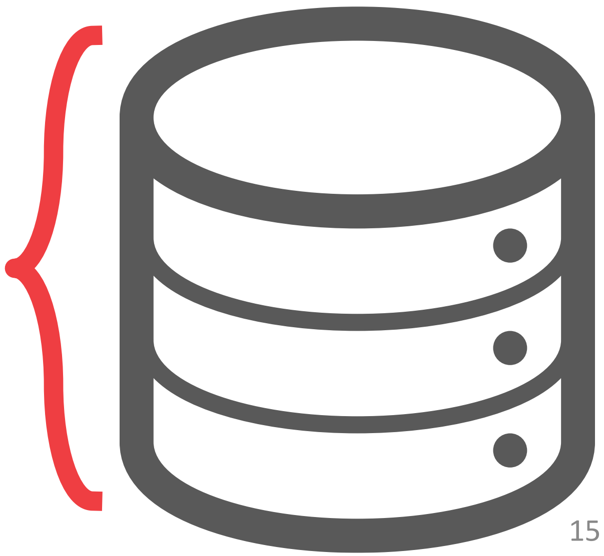
OPTIMISTIC CONCURRENCY CONTROL



Workspace

Record	Value	Write Timestamp
A	888	∞
B	999	∞

Record	Value	Write Timestamp
A	888	10001
B	999	10001



OBSERVATION

- When there is low contention, optimistic protocols perform better because the DBMS spends less time checking for conflicts.
- At high contention, the both classes of protocols degenerate to essentially the same serial execution.

CONCURRENCY CONTROL EVALUATION

- Compare in-memory concurrency control protocols at high levels of parallelism.
  - Single test-bed system.
  - Evaluate protocols using core counts beyond what is available on today's CPUs.

- Running in extreme environments exposes what are the main bottlenecks in the DBMS.

STARING INTO THE ABYSS: AN EVALUATION OF CONCURRENCY CONTROL WITH ONE THOUSAND CORES
VLDB 2014

1000-CORE CPU SIMULATOR

- DBx1000 Database System
  - In-memory DBMS with pluggable lock manager.
  - No network access, logging, or concurrent indexes.
  - All txns execute using stored procedures.

- MIT Graphite CPU Simulator
  - Single-socket, tile-based CPU.
  - Shared L2 cache for groups of cores.
  - Tiles communicate over 2D-mesh network.

TARGET WORKLOAD

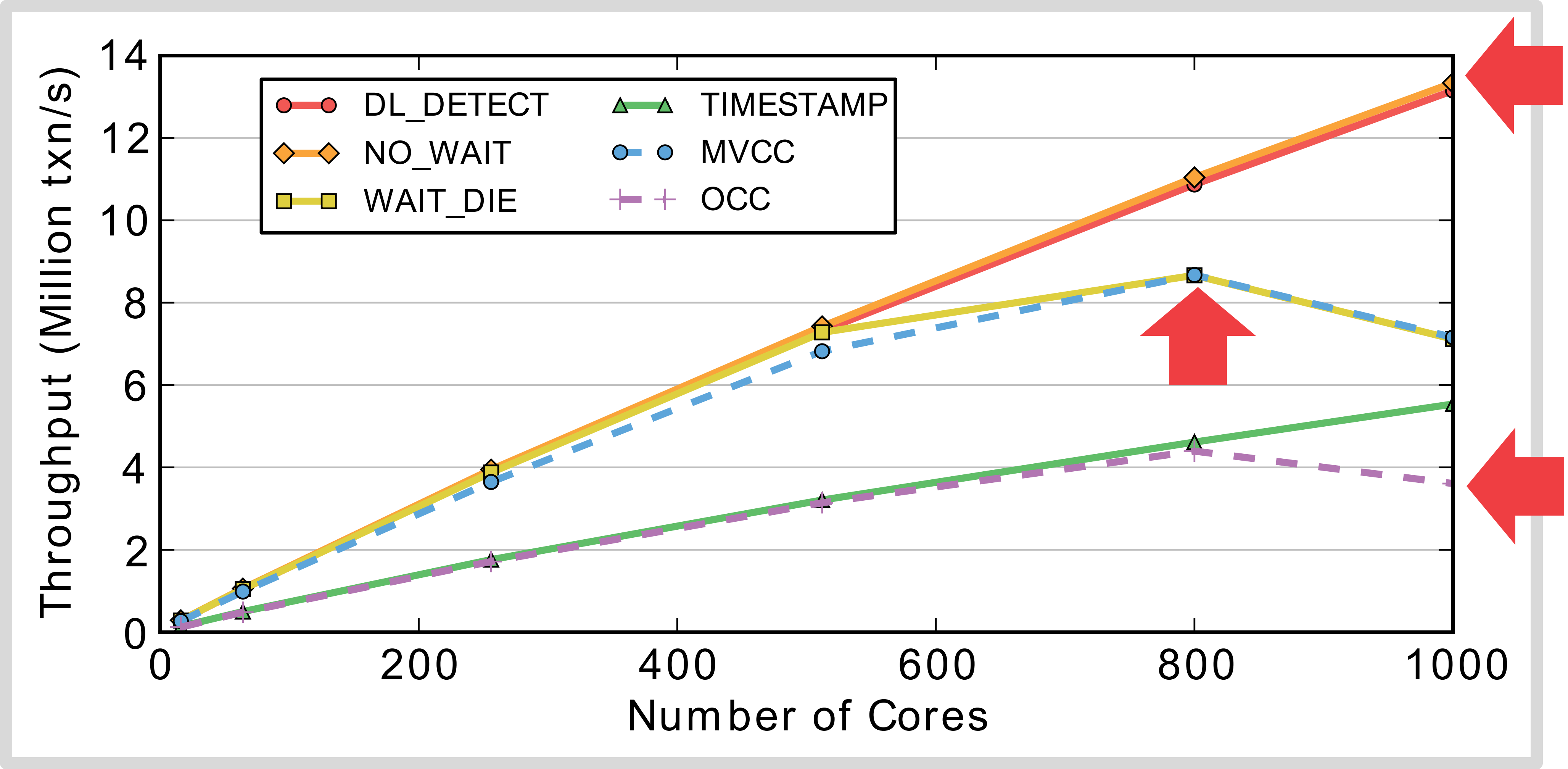
- Yahoo! Cloud Serving Benchmark (YCSB)
  - 20 million tuples
  - Each tuple is 1KB (total database is ~20GB)
- Each transactions reads/modifies 16 tuples.
- Varying skew in transaction access patterns.
- Serializable isolation level.

CONCURRENCY CONTROL SCHEMES

DL_DETECT	2PL w/ Deadlock Detection
NO_WAIT	2PL w/ Non-waiting Prevention
WAIT_DIE	2PL w/ Wait-and-Die Prevention

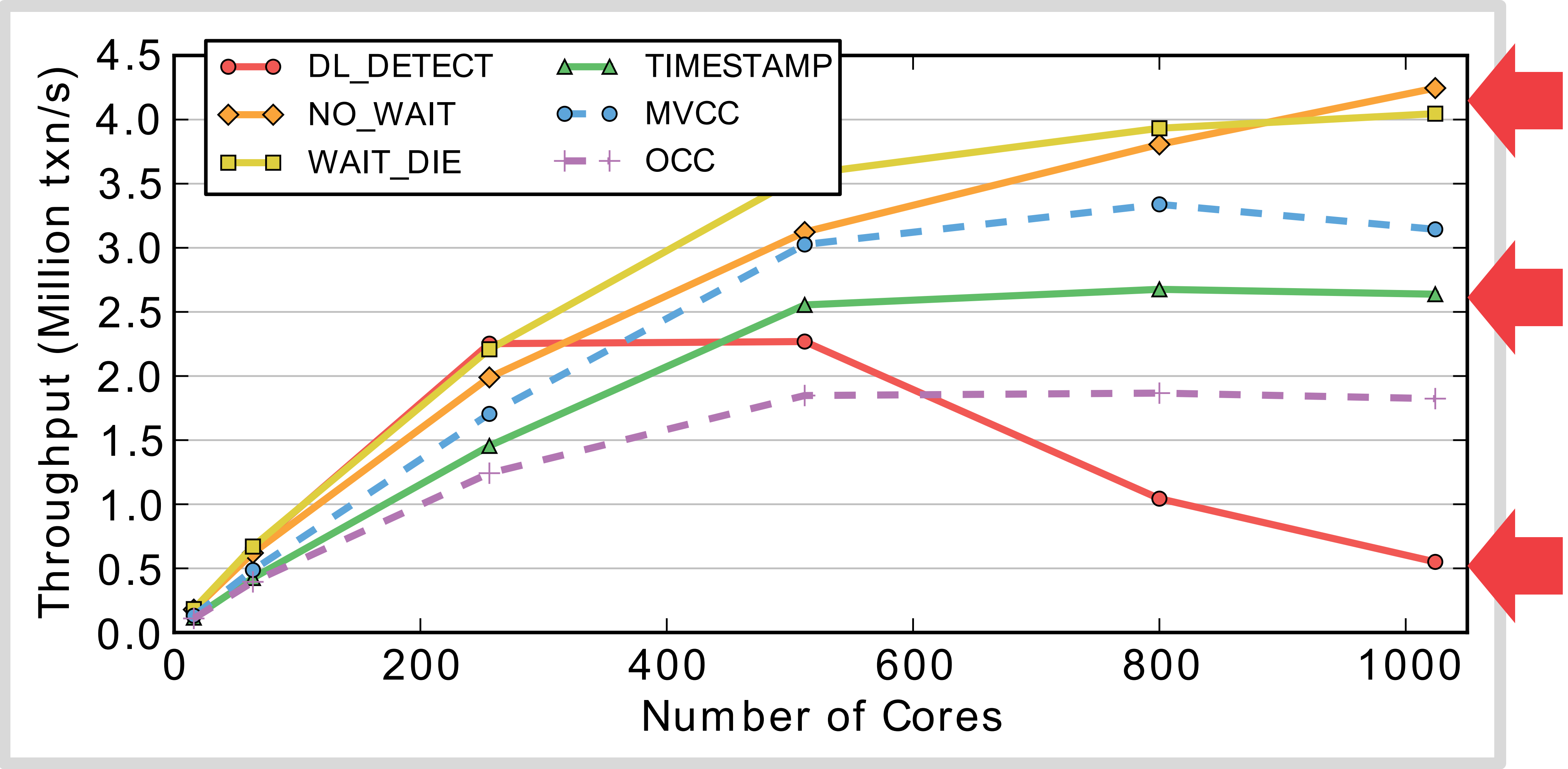
TIMESTAMP	Basic T/O Algorithm
MVCC	Multi-Version T/O
OCC	Optimistic Concurrency Control

READ-ONLY WORKLOAD

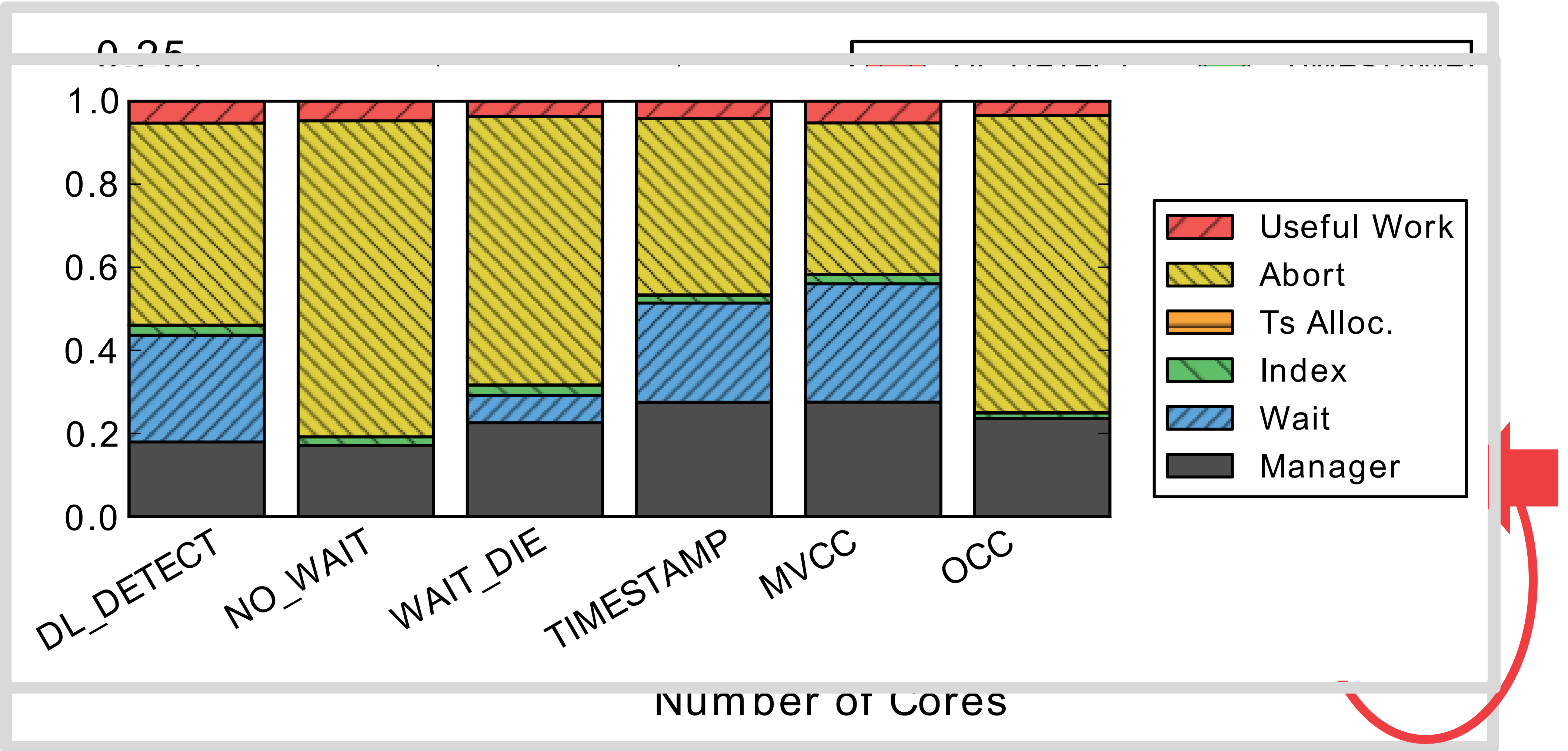


SCHOOL OF COMPUTING
UNIVERSITY OF UTAH

WRITE-INTENSIVE / MEDIUM-CONTENTION



WRITE-INTENSIVE / HIGH-CONTENTION



BOTTLENECKS

- **Lock Thrashing**
  - DL_DETECT, WAIT_DIE
- **Timestamp Allocation**
  - All T/O algorithms + WAIT_DIE
- **Memory Allocations**
  - OCC + MVCC

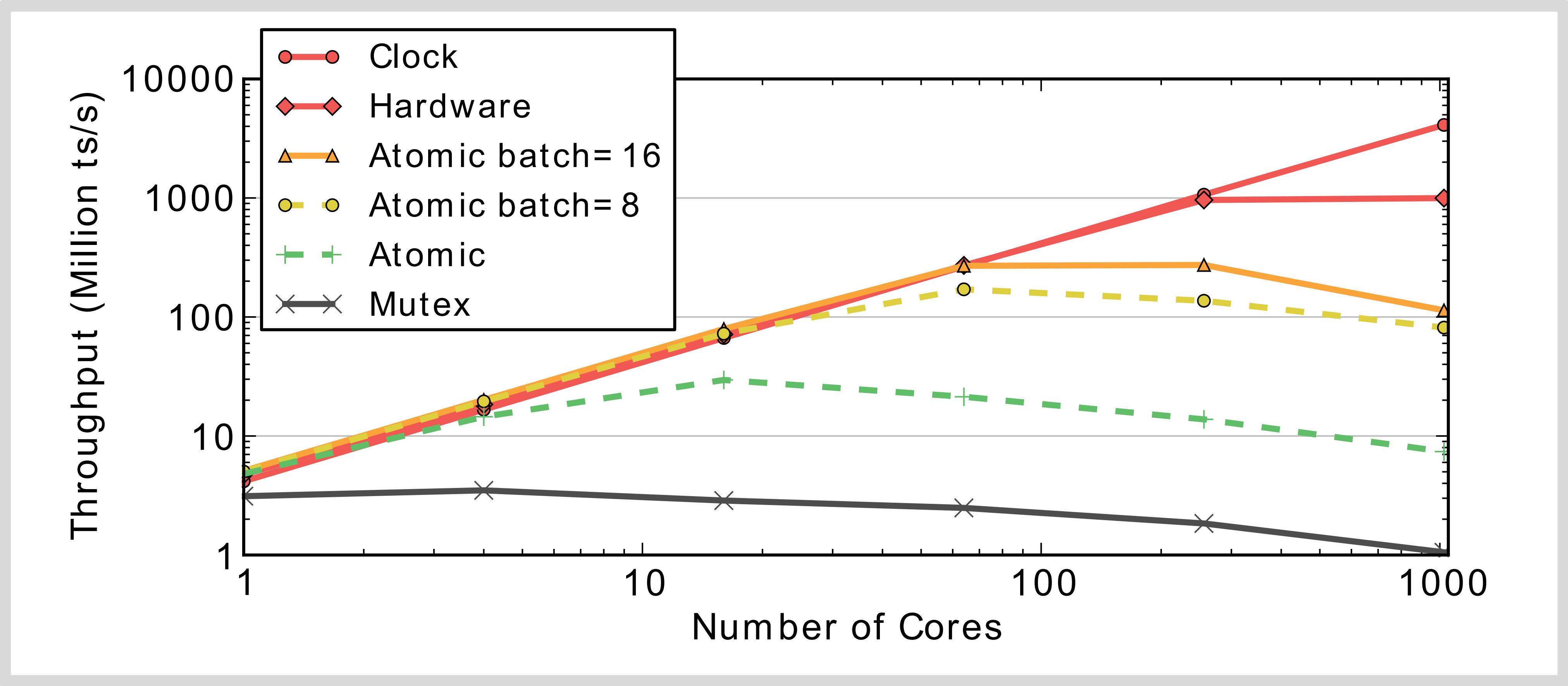
LOCK THRASHING

- Each txn waits longer to acquire locks, causing other txn to wait longer to acquire locks.
- Can measure this phenomenon by removing deadlock detection/prevention overhead.
  - Force txns to acquire locks in primary key order.
  - Deadlocks are not possible.

TIMESTAMP ALLOCATION

- **Mutex**
  - Worst option.
- **Atomic Addition**
  - Requires cache invalidation on write.
- **Batched Atomic Addition**
  - Needs a back-off mechanism to prevent fast burn.
- **Hardware Clock**
  - Not sure if it will exist in future CPUs.
- **Hardware Counter**
  - Not implemented in existing CPUs.

TIMESTAMP ALLOCATION



MEMORY ALLOCATIONS

- Copying data on every read/write access slows down the DBMS because of contention on the memory controller.
  - In-place updates and non-copying reads are not affected as much.

- Default libc `malloc` is slow. Never use it.
  - We will discuss this further later in the semester.

CONCURRENCY CONTROL

- Observation: The cost of a txn acquiring a lock is the same as accessing data.
- In-memory DBMS may want to detect conflicts between txns at a different granularity.
  - **Fine-grained locking** allows for better concurrency but requires more locks.
  - **Coarse-grained locking** requires fewer locks but limits the amount of concurrency.

Acknowledgement: Prof. Andy Pavlo, CMU

LARGER-THAN-MEMORY DATABASES

- DRAM is fast, but data is not accessed with the same frequency and in the same manner.
  - Hot Data: OLTP Operations
  - Cold Data: OLAP Queries

- We will study techniques for how to bring back disk-resident data without slowing down the entire system.

Acknowledgement: Prof. Andy Pavlo, CMU

NEXT CLASS

- Multi-Version Concurrency Control