Design Principles for Scaling Multi-core OLTP Under High Contention

Kun Ren, **Jose Faleiro**, Daniel Abadi
Yale University

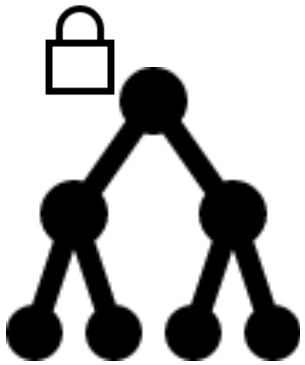
Conflicts: The scourge of database systems

- Logical conflicts
  - Due to data conflicts between transactions

```
T1:
Read(x);
```

```
T2:
Write(x);
```

- Physical conflicts
  - Due to contention on internal data-structures



Conflicts: The scourge of database systems

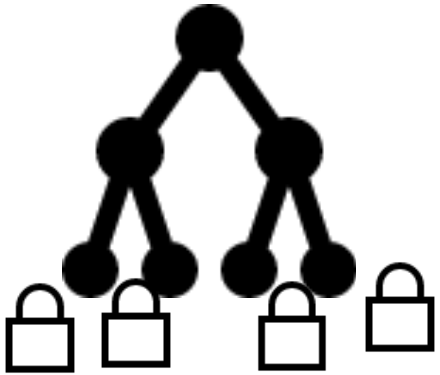
- Logical conflicts
  - Due to data conflicts between transactions

T_1: Read(x_0);

T_2: Write(x_2);

Addressed via new correctness criteria, exploiting semantics

- Physical conflicts
  - Due to contention on internal data-structures



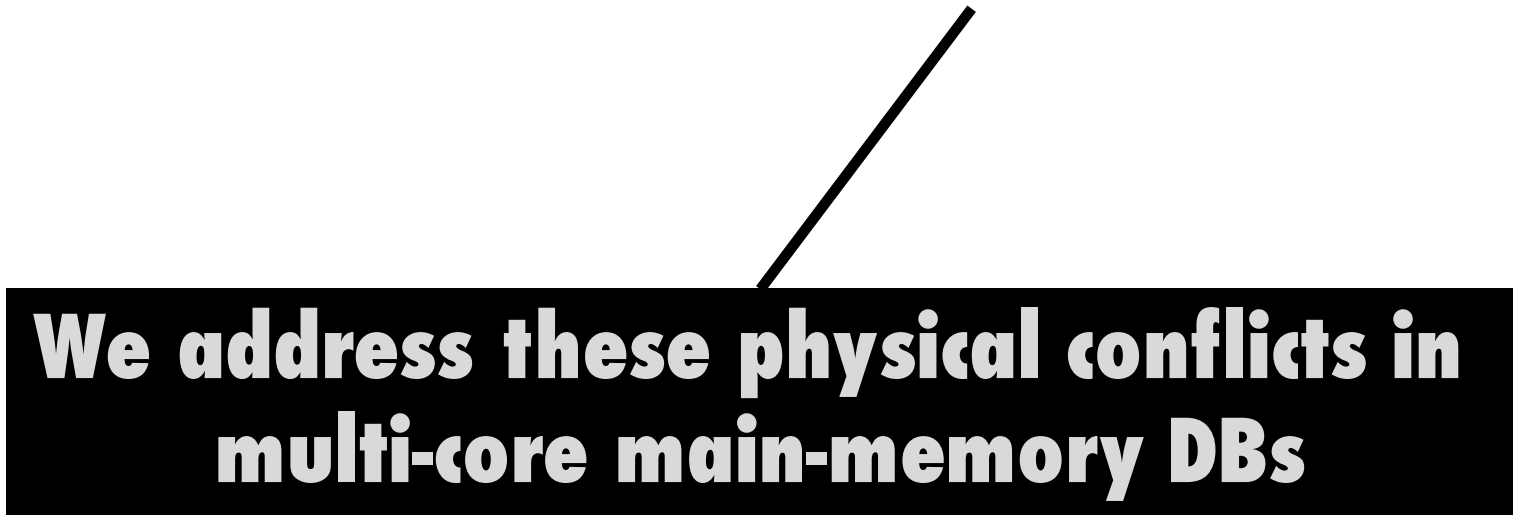
Addressed via new protocols, DB architectures

… but conflicts are inevitable

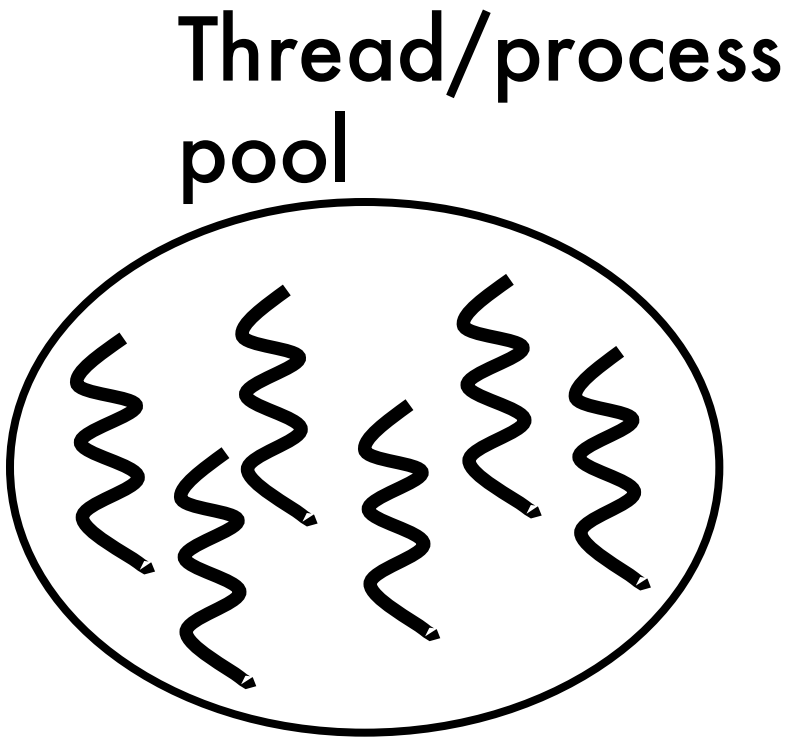
- Logical conflicts are application dependent
- Logical conflicts directly result in physical conflicts

… but conflicts are inevitable

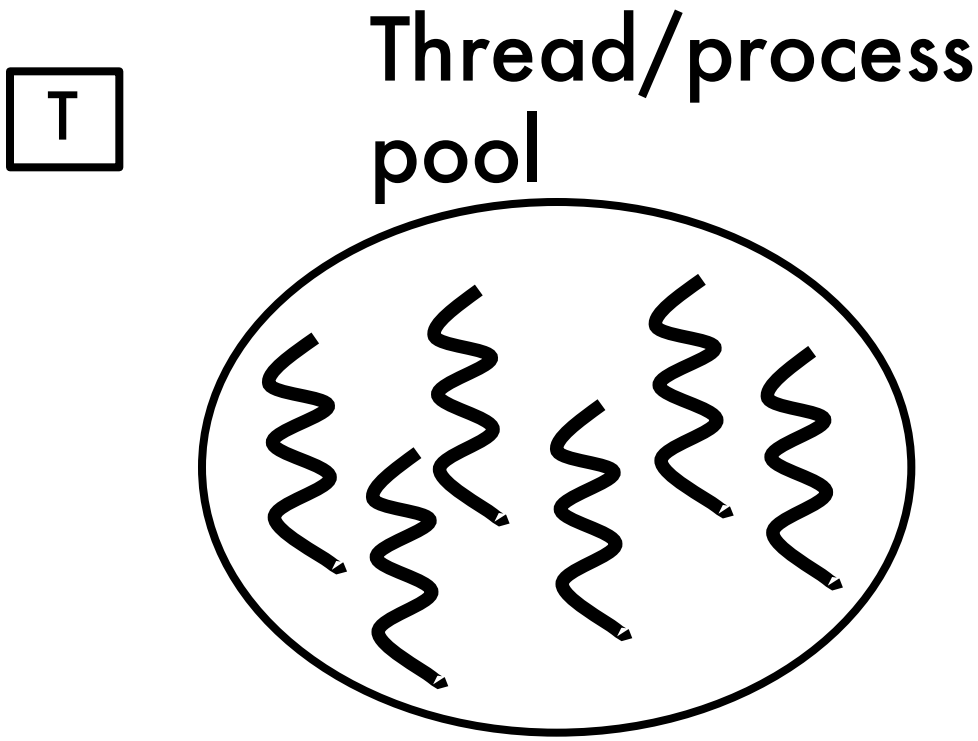
- Logical conflicts are application dependent
- Logical conflicts directly result in physical conflicts



The life of a transaction

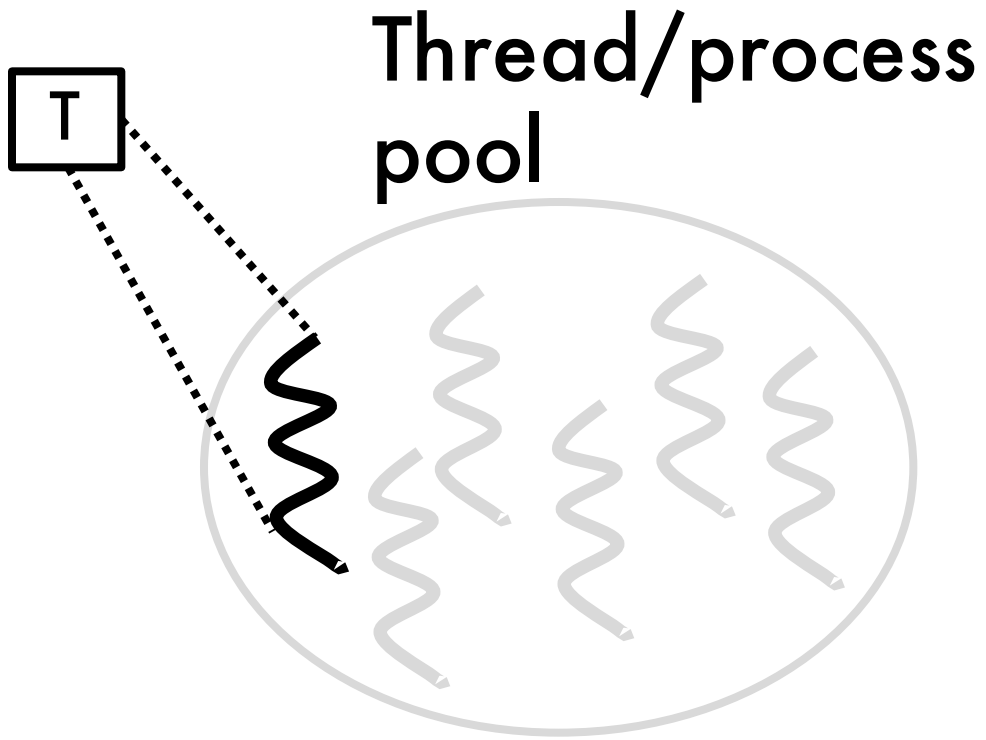


The life of a transaction



The life of a transaction

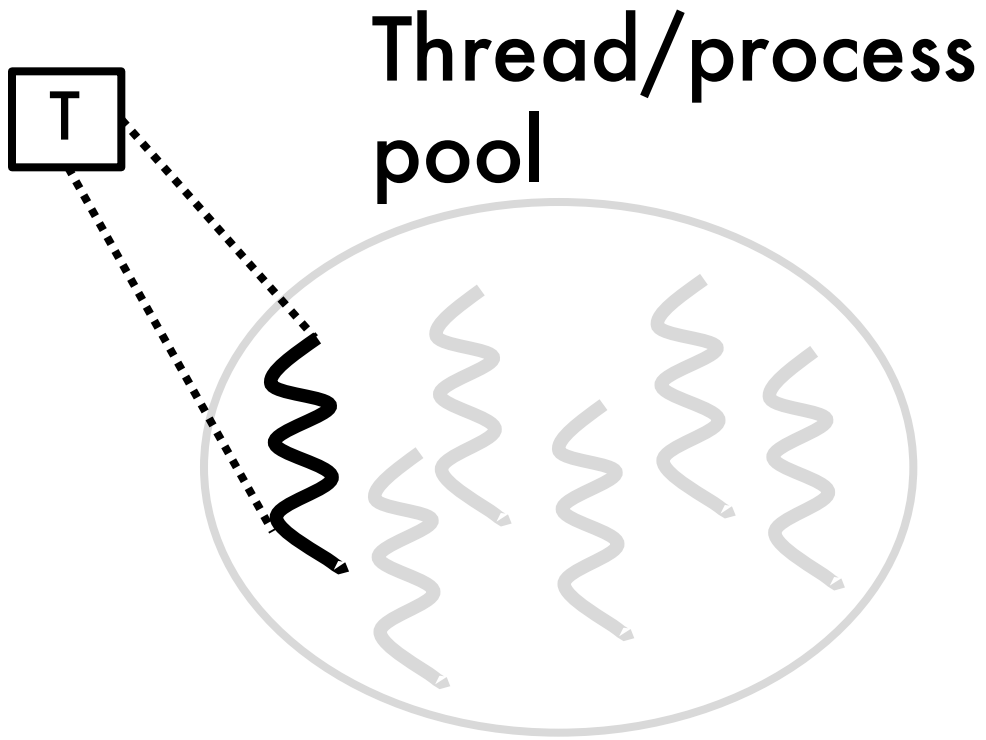
- Assign a transaction to an "execution context"



The life of a transaction

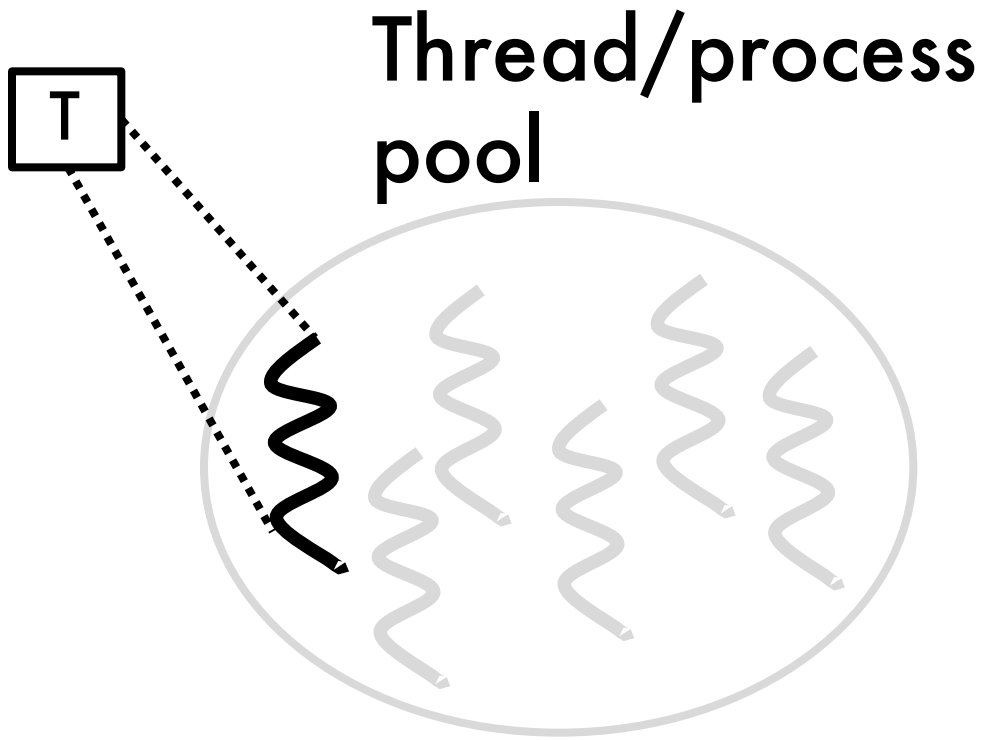
- Assign a transaction to an "execution context"

- Assigned context performs all actions required to execute the transaction
  - Concurrency control
  - Transaction logic
  - Logging

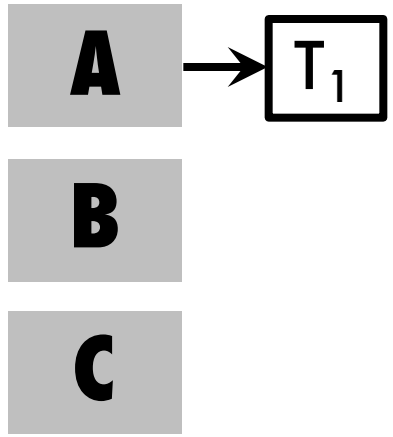


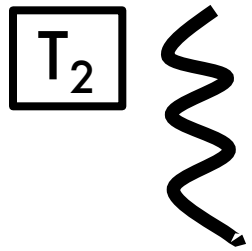
The life of a transaction

- Assign a transaction to an "execution context"
- Assigned context performs all actions required to execute the transaction
  - Concurrency control
  - Transaction logic
  - Logging
- Deal with conflicts via **shared** concurrency control meta-data



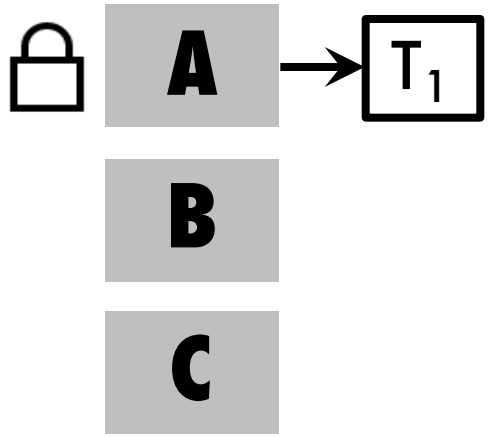
Example: Logical lock acquisition

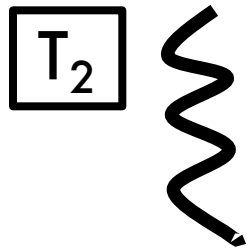




Example: Logical lock acquisition

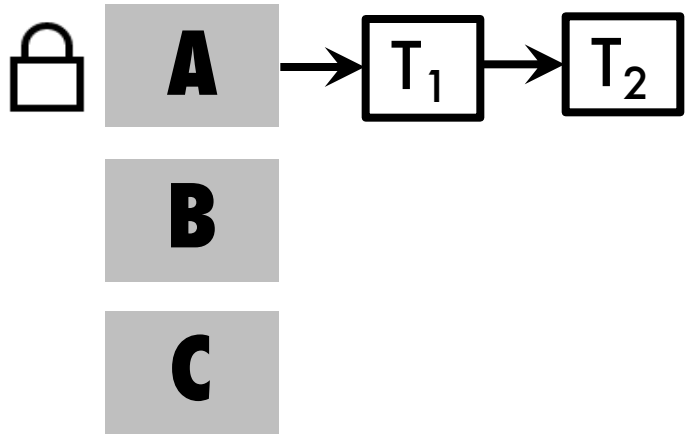
- Latch bucket

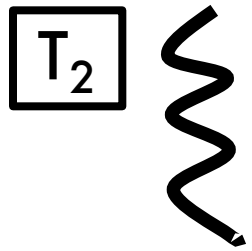




Example: Logical lock acquisition

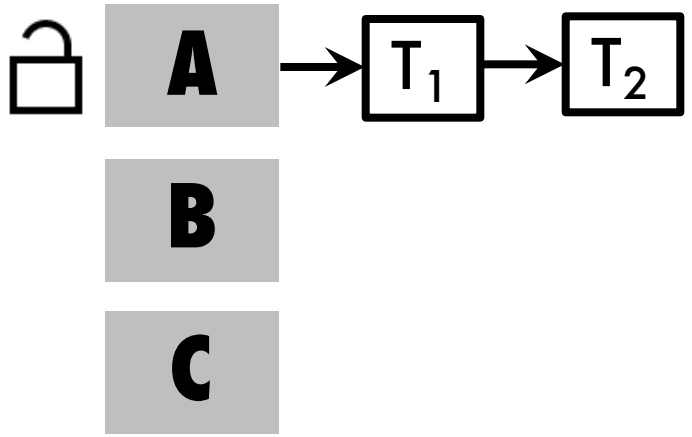
- Latch bucket
- Add lock request

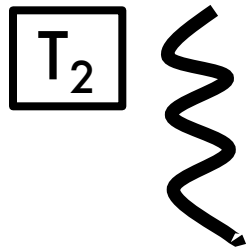




Example: Logical lock acquisition

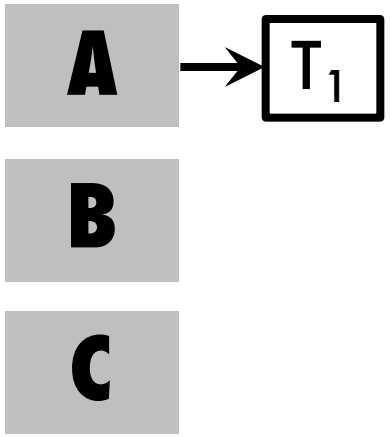
- Latch bucket
- Add lock request
- Unlatch bucket

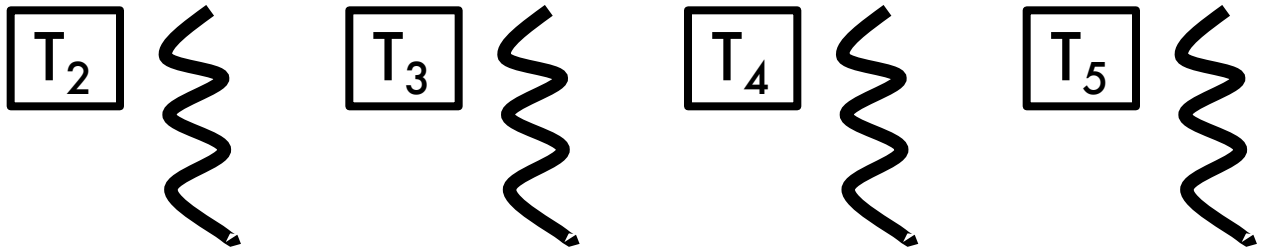




Example: Logical lock acquisition

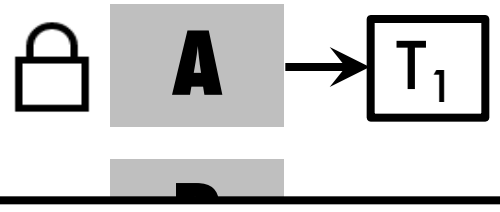
- Latch bucket
- Add lock request
- Unlatch bucket





Example: Logical lock acquisition

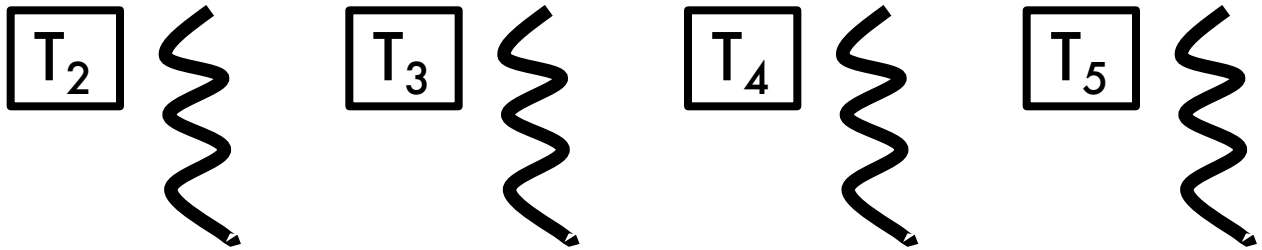
- **Latch bucket**
- Add lock request
- Unlatch bucket



Several threads must acquire a single latch

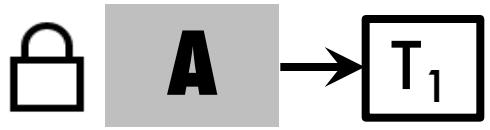
Synchronization overhead

Overhead increases with contention



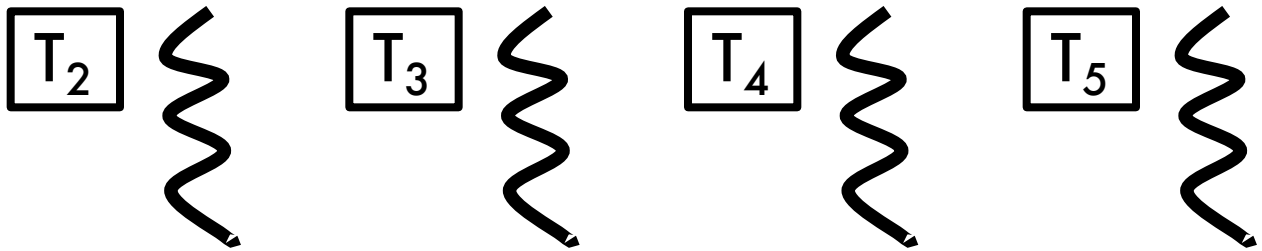
Example: Logical lock acquisition

- Latch bucket
- **Add lock request**
- Unlatch bucket



Lock list moves across cores

Coherence overhead



Example: Logical lock acquisition

- Latch bucket
- **Add lock request**
- Unlatch bucket

Lock list moves across cores

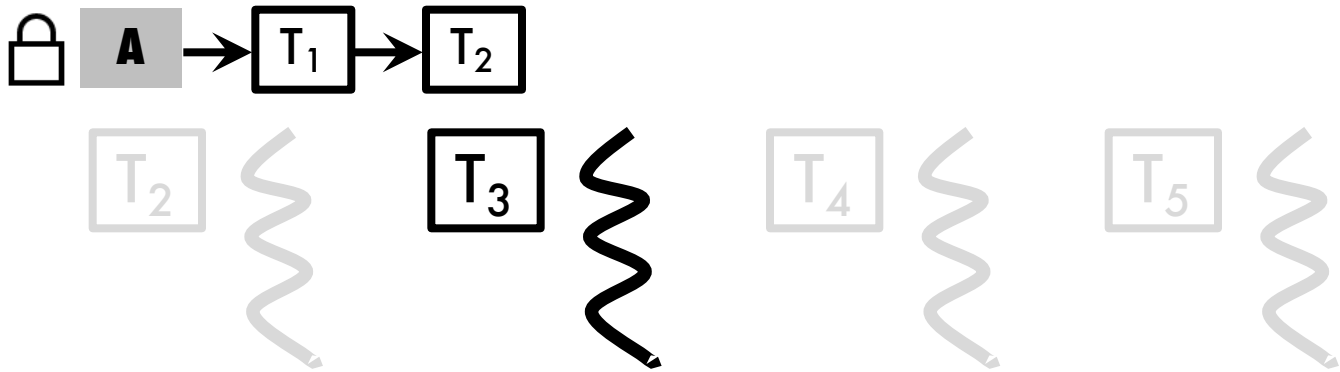
Coherence overhead

Example: Logical lock acquisition

- Latch bucket
- **Add lock request**
- Unlatch bucket

Lock list moves across cores

Coherence overhead

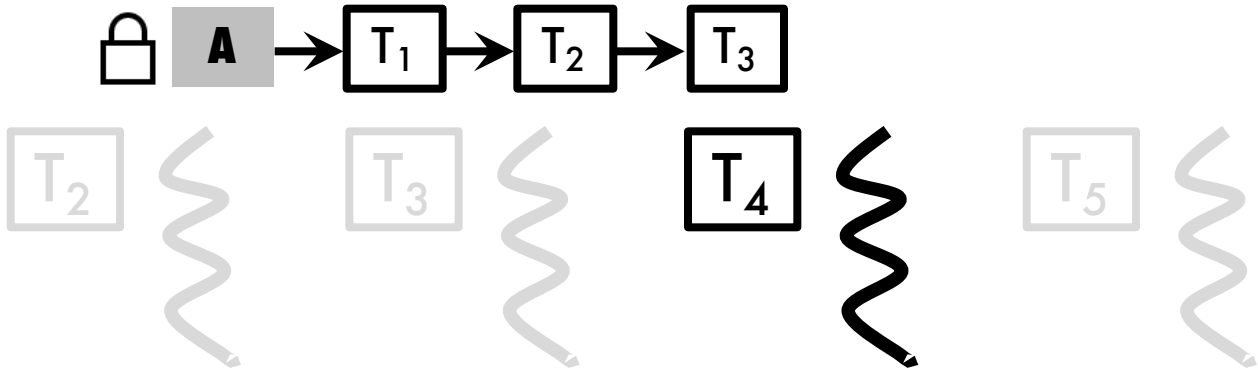


Example: Logical lock acquisition

- Latch bucket
- **Add lock request**
- Unlatch bucket

Lock list moves across cores

Coherence overhead

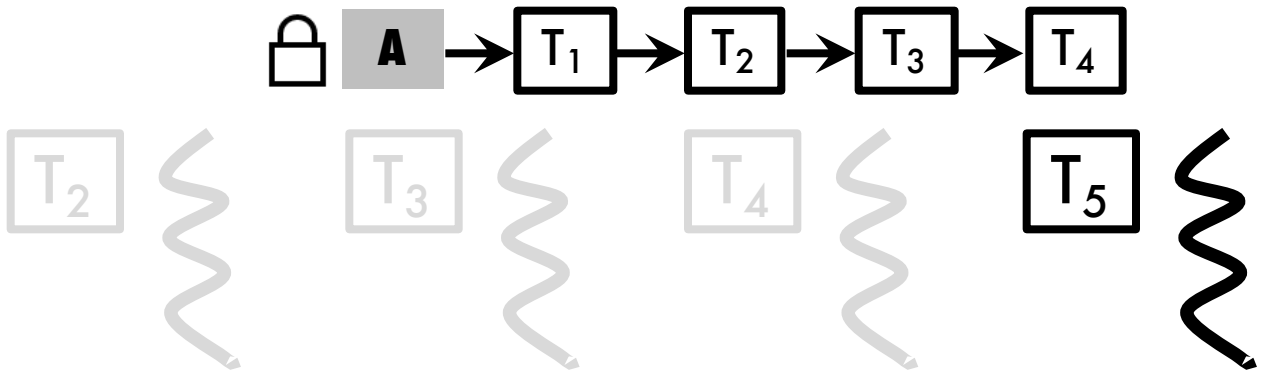


Example: Logical lock acquisition

- Latch bucket
- **Add lock request**
- Unlatch bucket

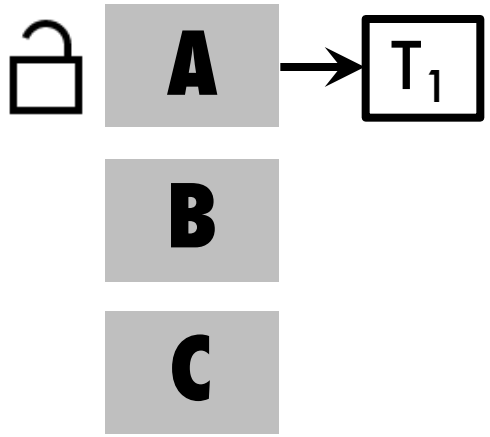
Lock list moves across cores

Coherence overhead

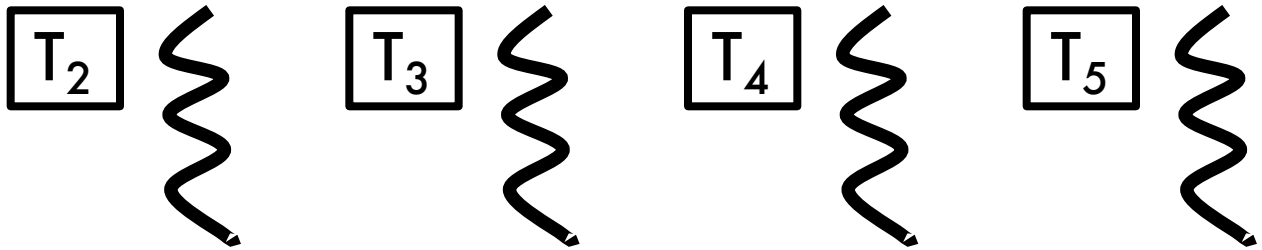


Example: Logical lock acquisition

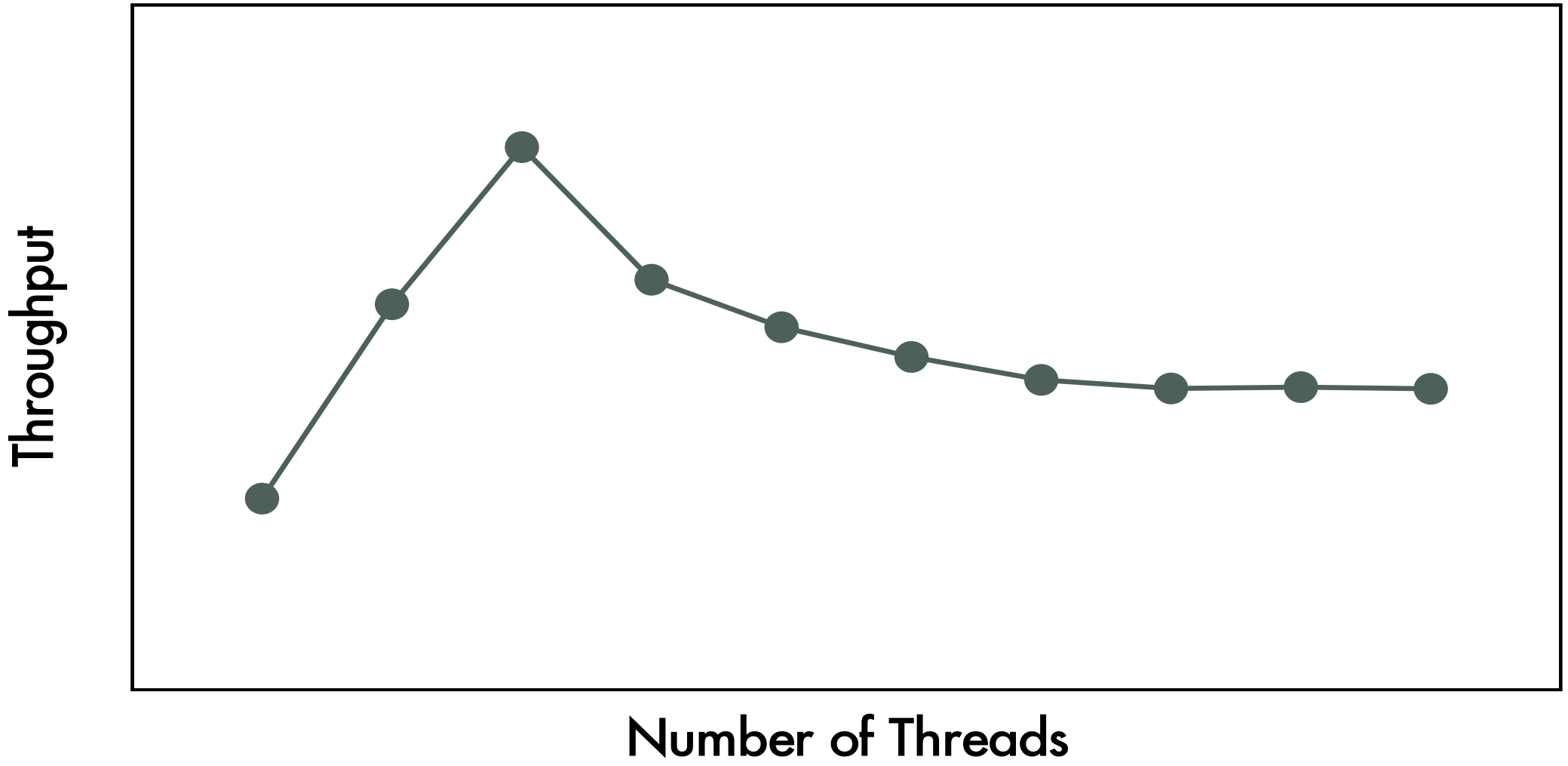
- Latch bucket
- Add lock request
- **Unlatch bucket**



More synchronization overhead



The result?



Dealing with contention on few cores

Dealing with contention on lots of cores

Observations

- Contention for lock list depends on workload, not implementation
- Latches can be made as fine-grained as possible
  - E.g., bucket-level latches
- But if records are popular, fine-grained latching will not help

Every protocol has the same overheads

- Concurrency control protocols use object meta-data
  - Lock lists in locking
  - Timestamps in timestamp ordering, MVCC, OCC
- Object meta-data is accessible by any thread
  - E.g., threads update read and write timestamps in timestamp ordering
  - E.g., threads manipulate lock lists in 2PL
- Globally updatable shared meta-data is the problem
  - Synchronization, coherence overheads
  - No bound on threads contending for the same meta-data

Every protocol has the same overheads

- Concurrency control protocols use object meta-data
  - Lock lists in locking
  - Timestamps in timestam
- Object meta-data is
  - E.g., threads update re
  - E.g., threads manipulate lock lists in ZPL

Scalability anti-pattern

- Globally updatable shared meta-data is the problem
  - Synchronization, coherence overheads
  - No bound on threads contending for the same meta-data

Need a mechanism to bound contention on shared meta-data

Decouple concurrency control and execution

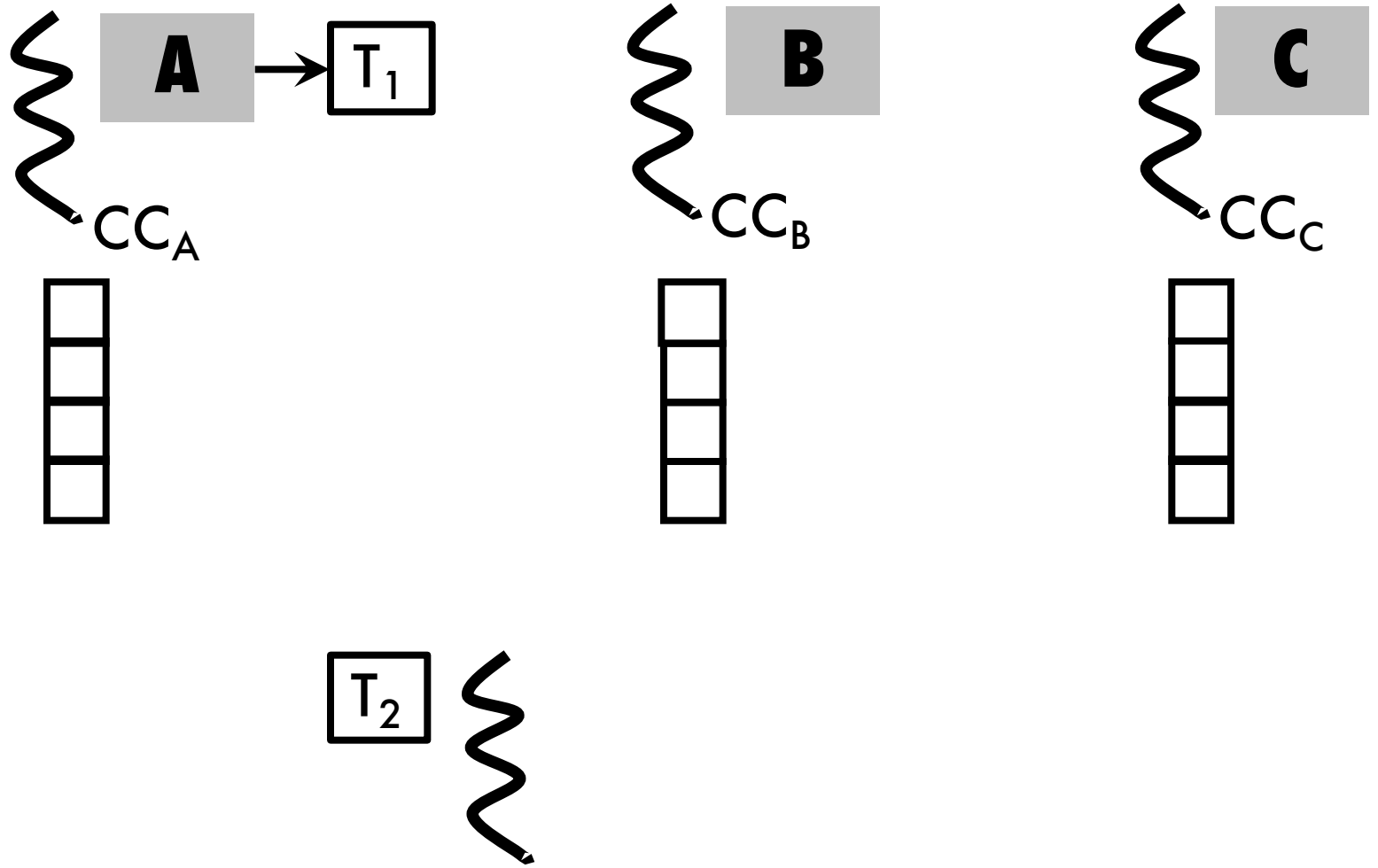
- Delegate concurrency control to a specific set of threads
- These threads are responsible for performing **only** concurrency control logic
- Access to concurrency control meta-data is mediated via concurrency control threads

Communication via message-passing

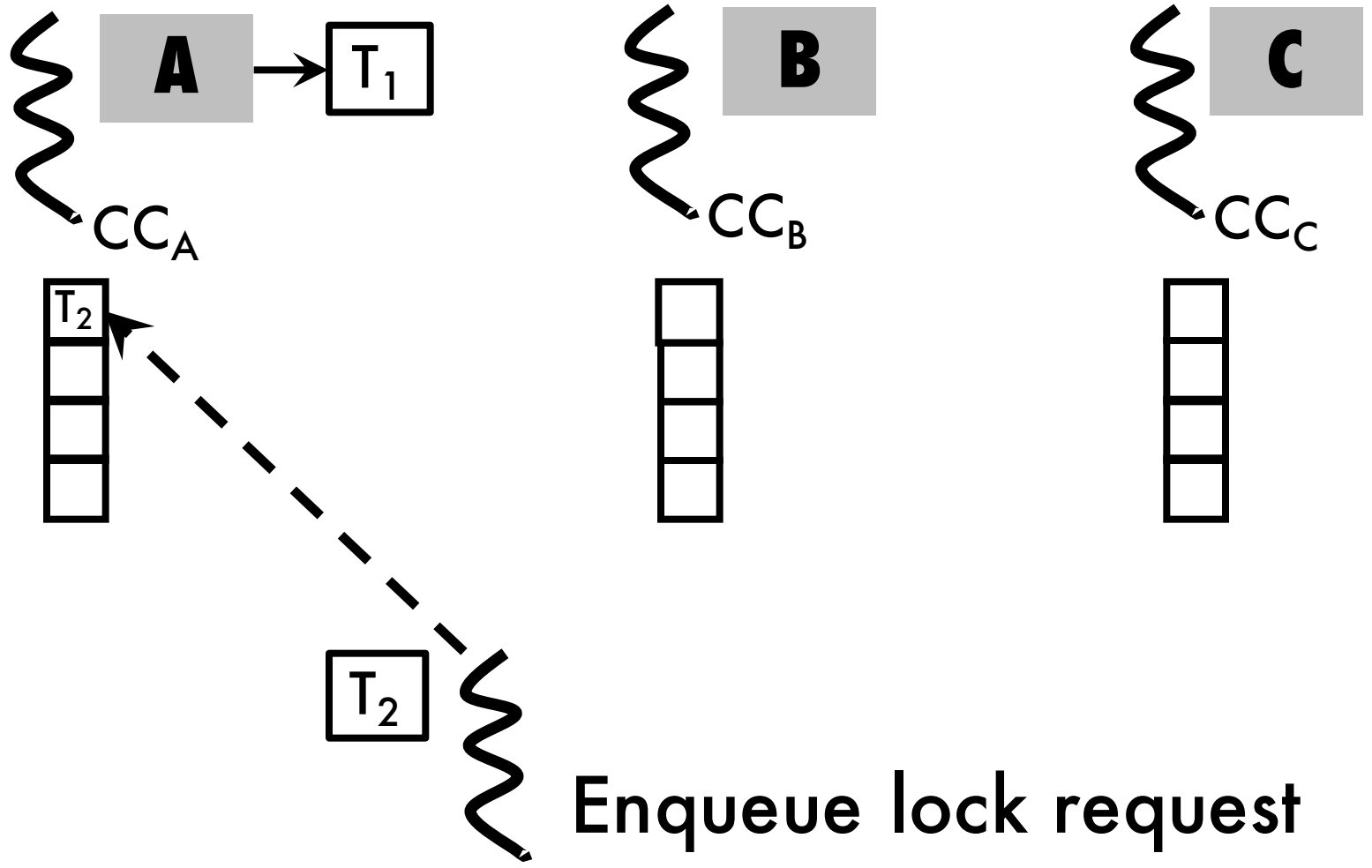
- No data sharing between concurrency control and execution threads

- Concurrency control and execution threads interact via explicit message-passing
  - Like RPC in distributed systems

Example: Logical lock acquisition

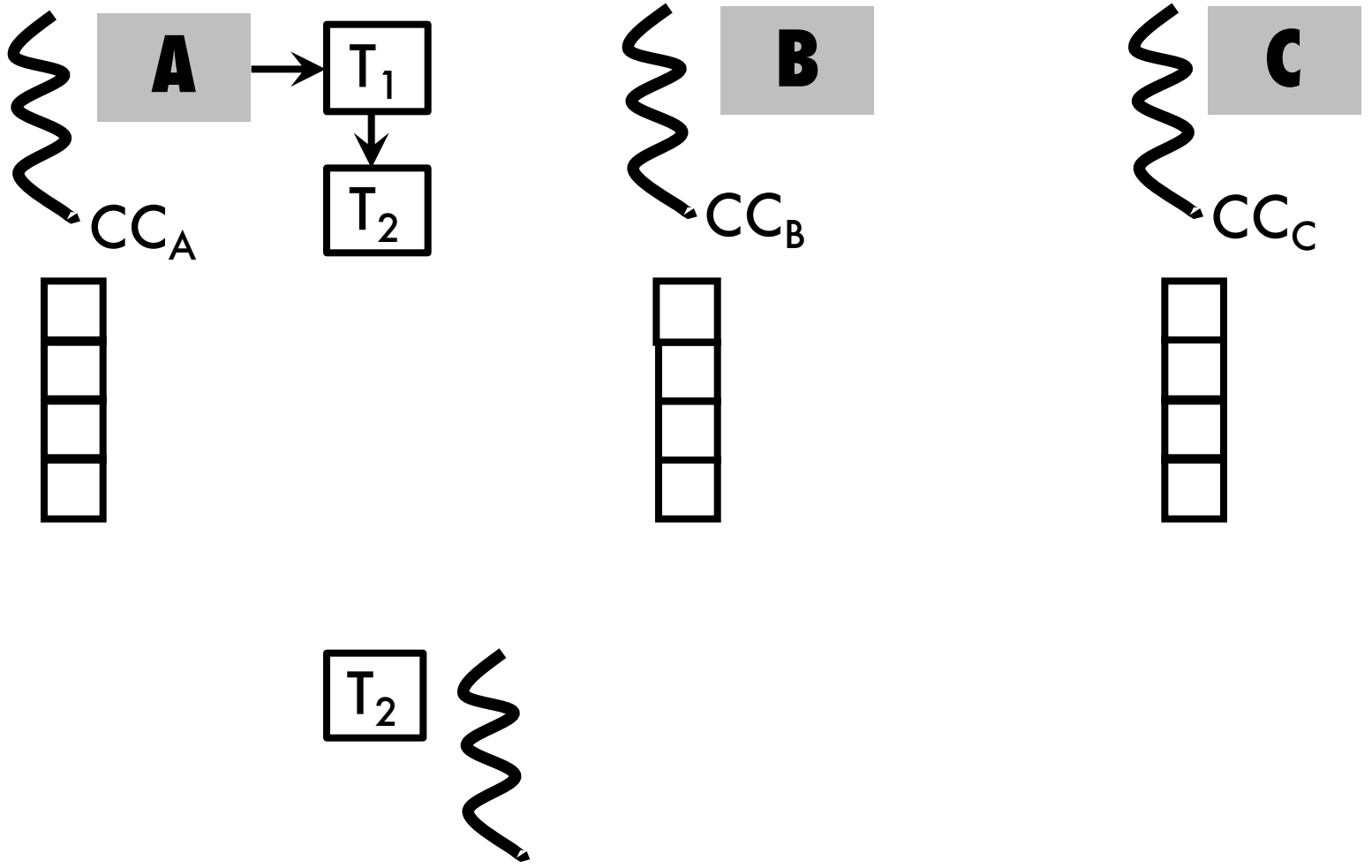


Example: Logical lock acquisition



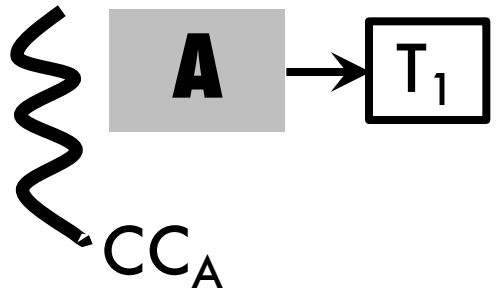
Example: Logical lock acquisition

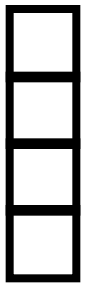
Add to lock list

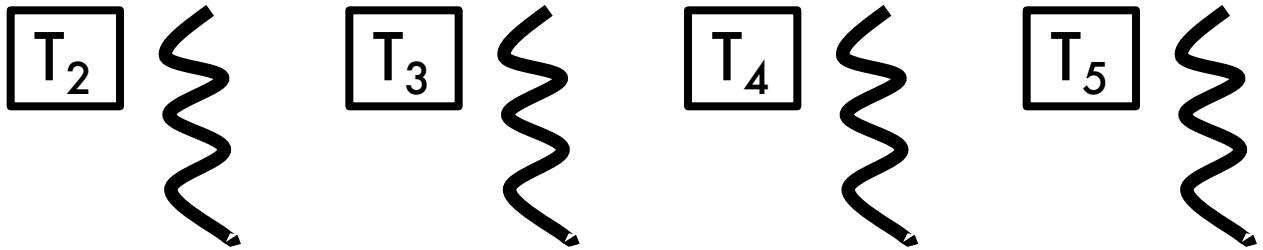


Example: Logical lock acquisition

- **Enqueue lock request**
- Acquire lock

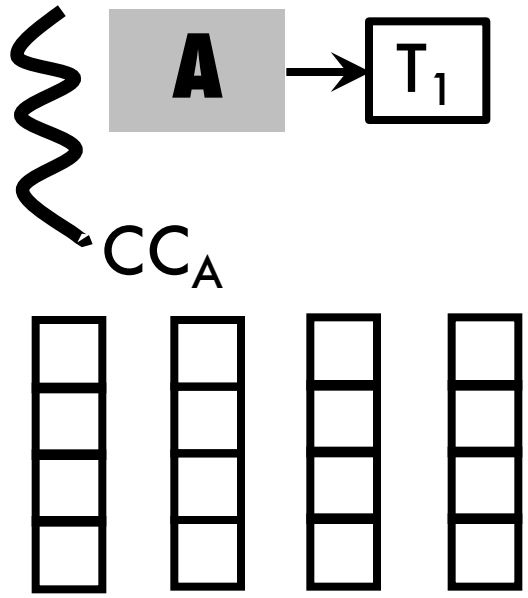






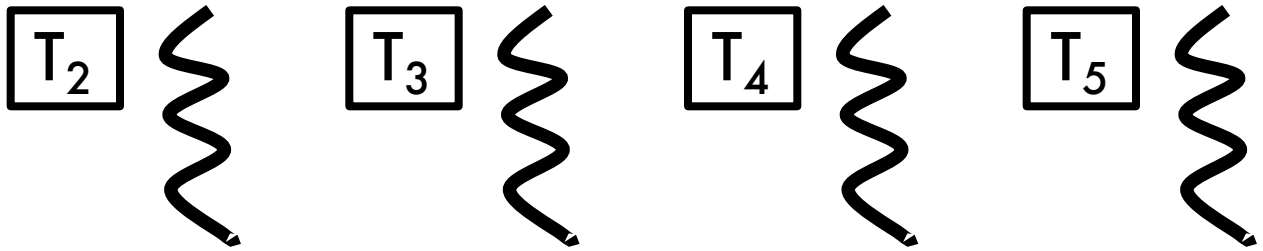
Example: Logical lock acquisition

- **Enqueue lock request**
- Acquire lock



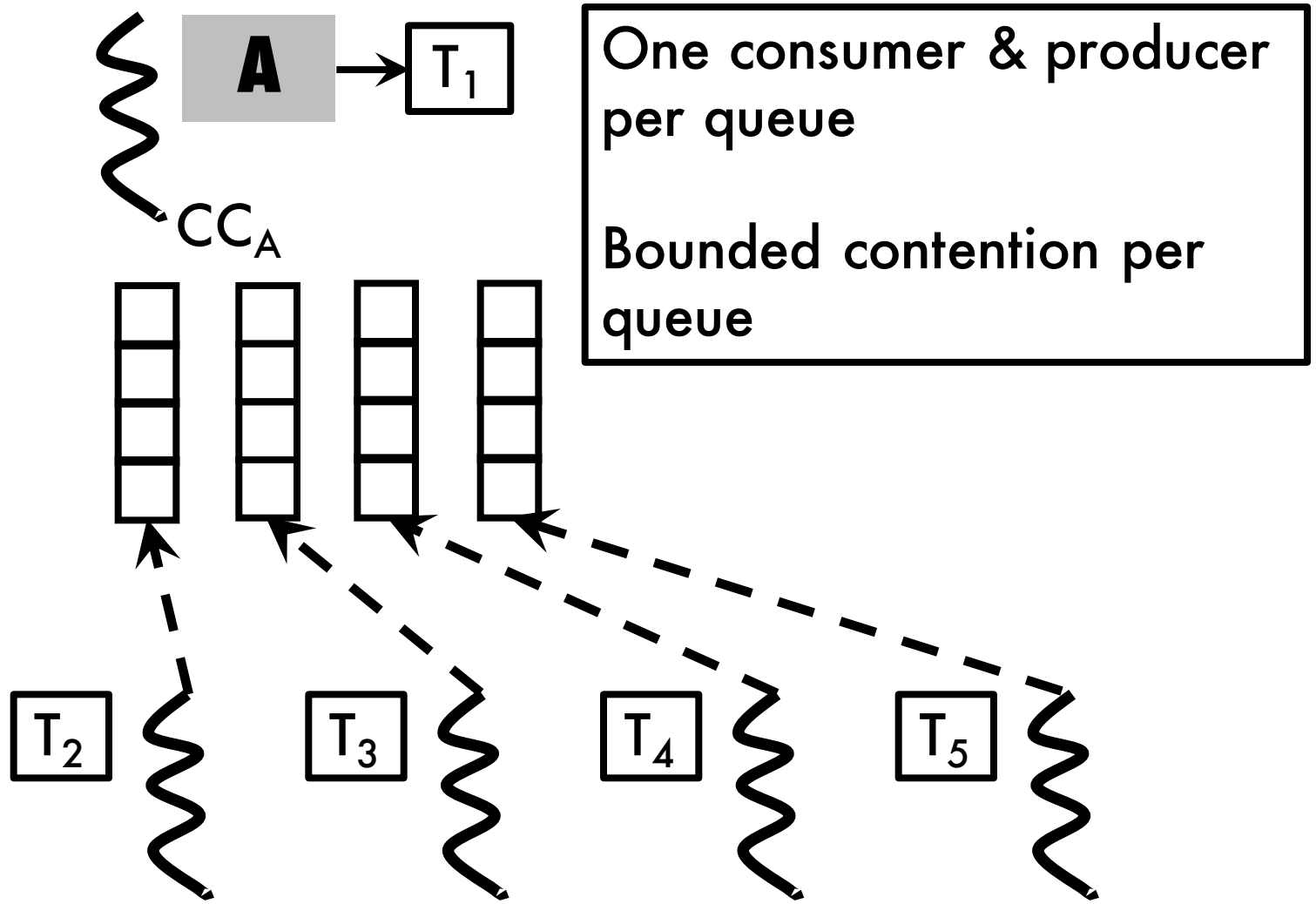
One consumer & producer per queue

Bounded contention per queue



Example: Logical lock acquisition

- **Enqueue lock request**
- Acquire lock



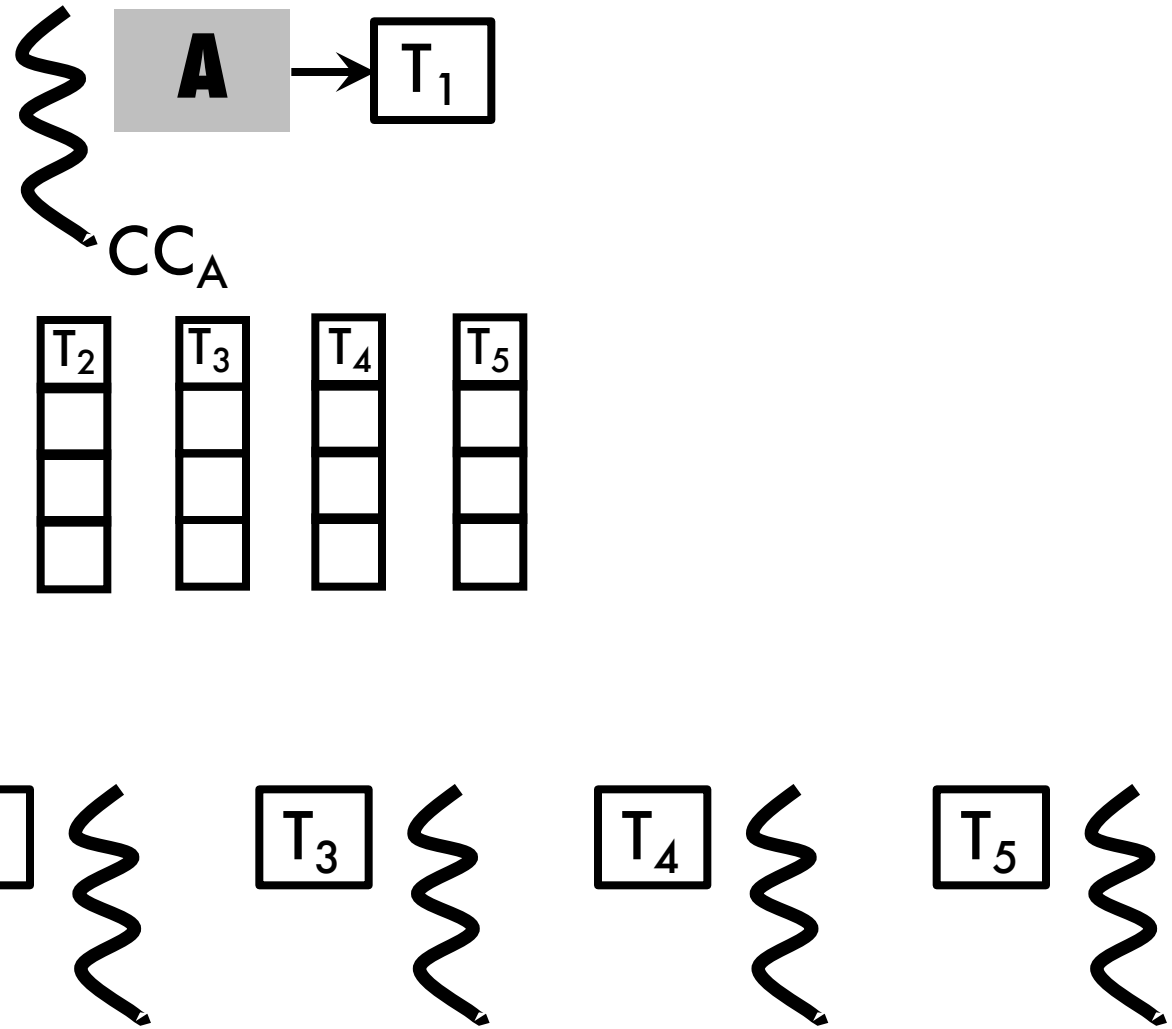
Example: Logical lock acquisition

- Enqueue lock request
- **Acquire lock**

One core manipulates lock list

List cannot "bounce" around cores

List likely remains cached under high contention



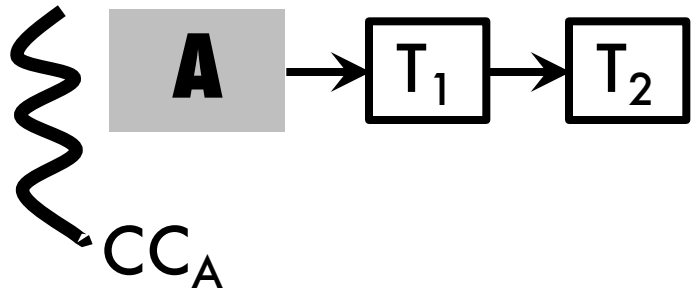
Example: Logical lock acquisition

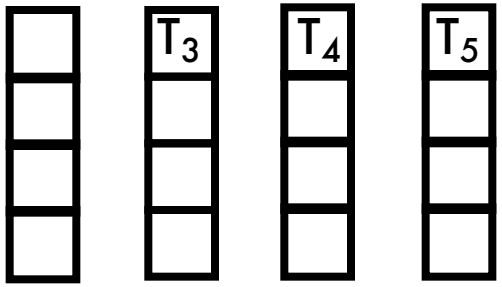
- Enqueue lock request
- **Acquire lock**

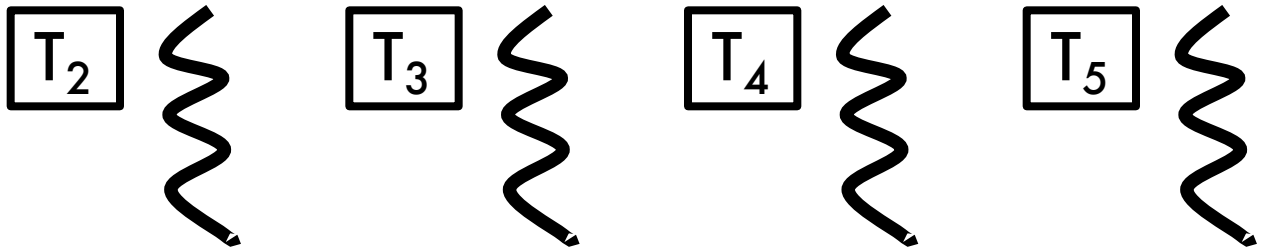
One core manipulates lock list

List cannot "bounce" around cores

List likely remains cached under high contention







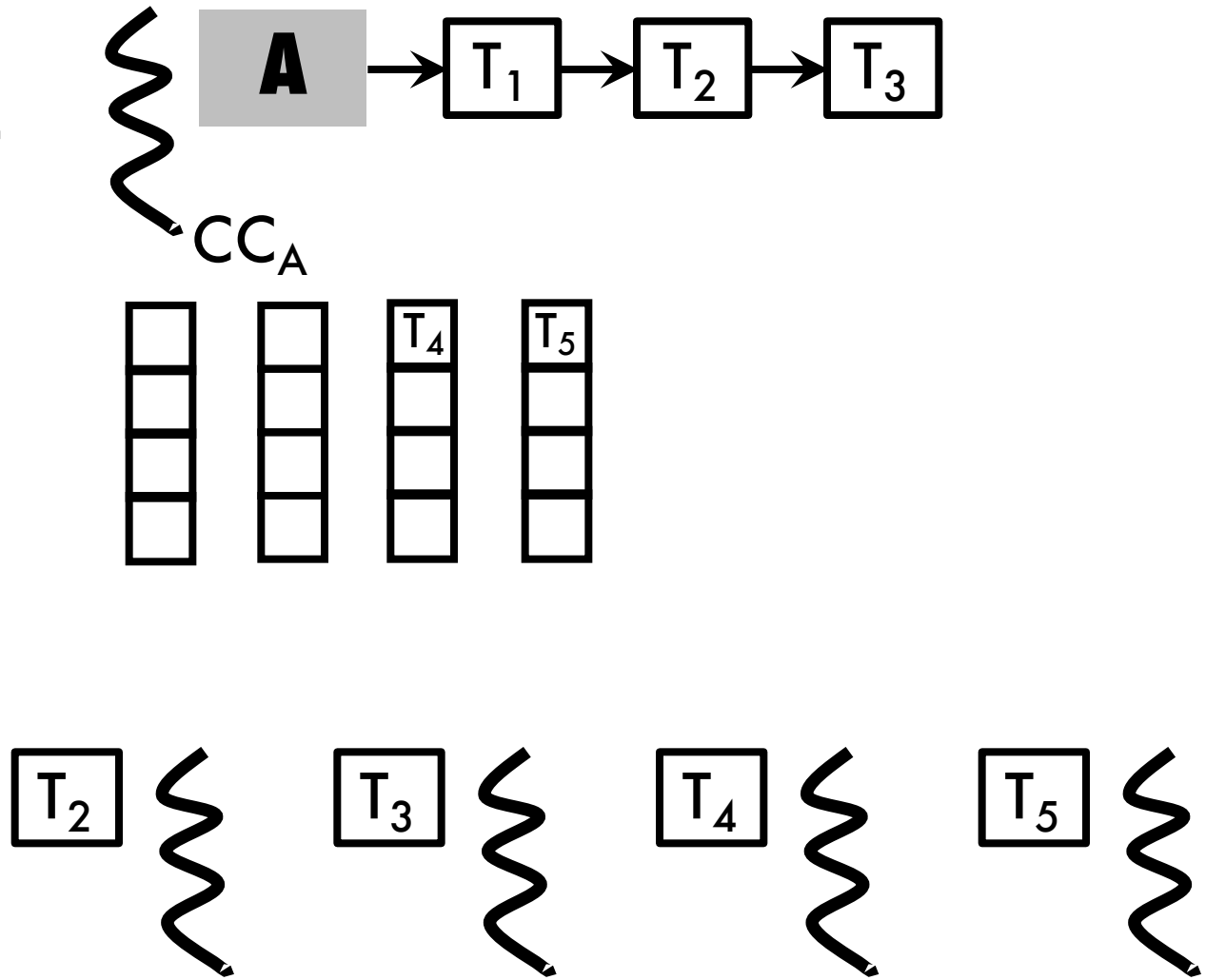
Example: Logical lock acquisition

- Enqueue lock request
- **Acquire lock**

One core manipulates lock list

List cannot "bounce" around cores

List likely remains cached under high contention



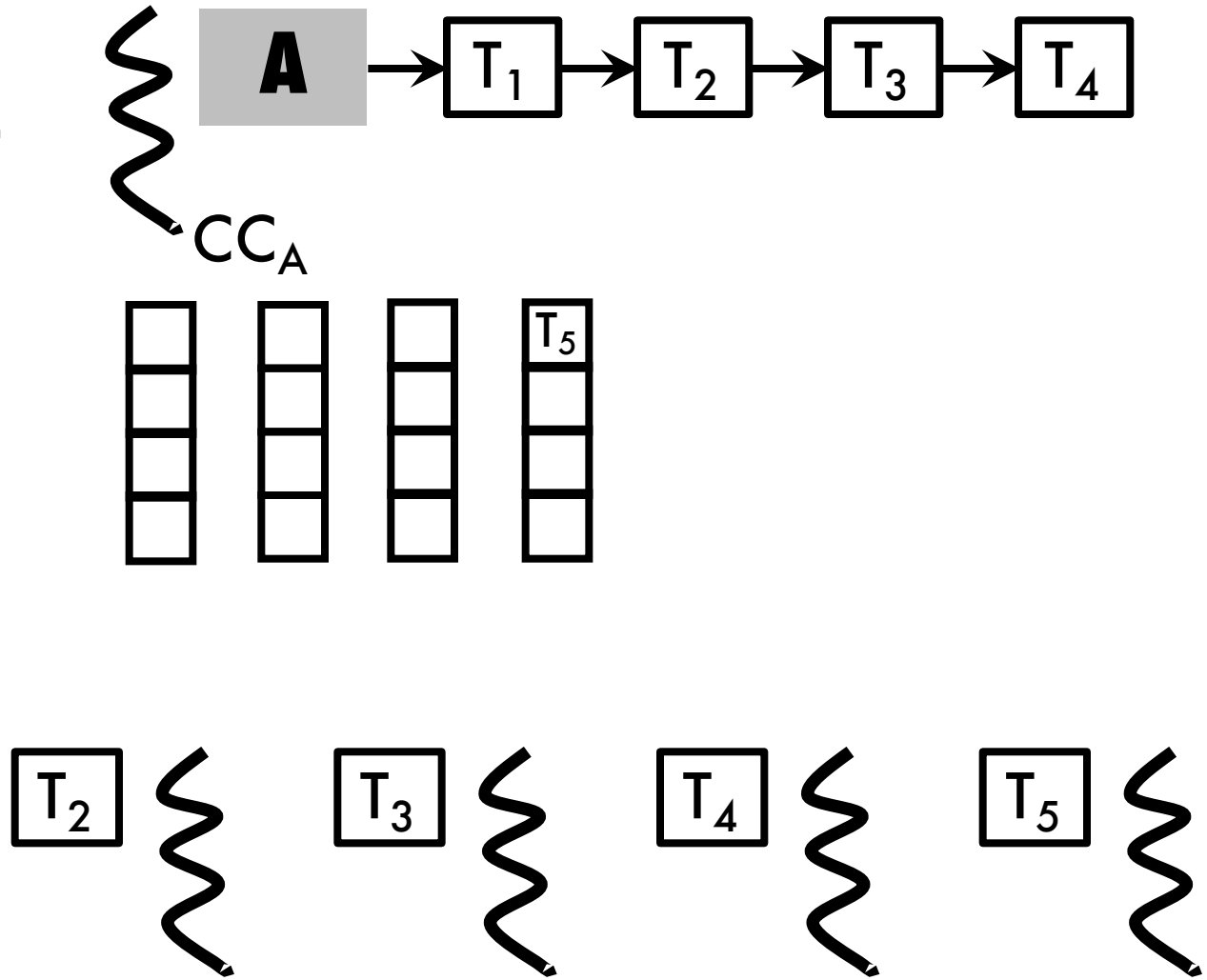
Example: Logical lock acquisition

- Enqueue lock request
- **Acquire lock**

One core manipulates lock list

List cannot "bounce" around cores

List likely remains cached under high contention



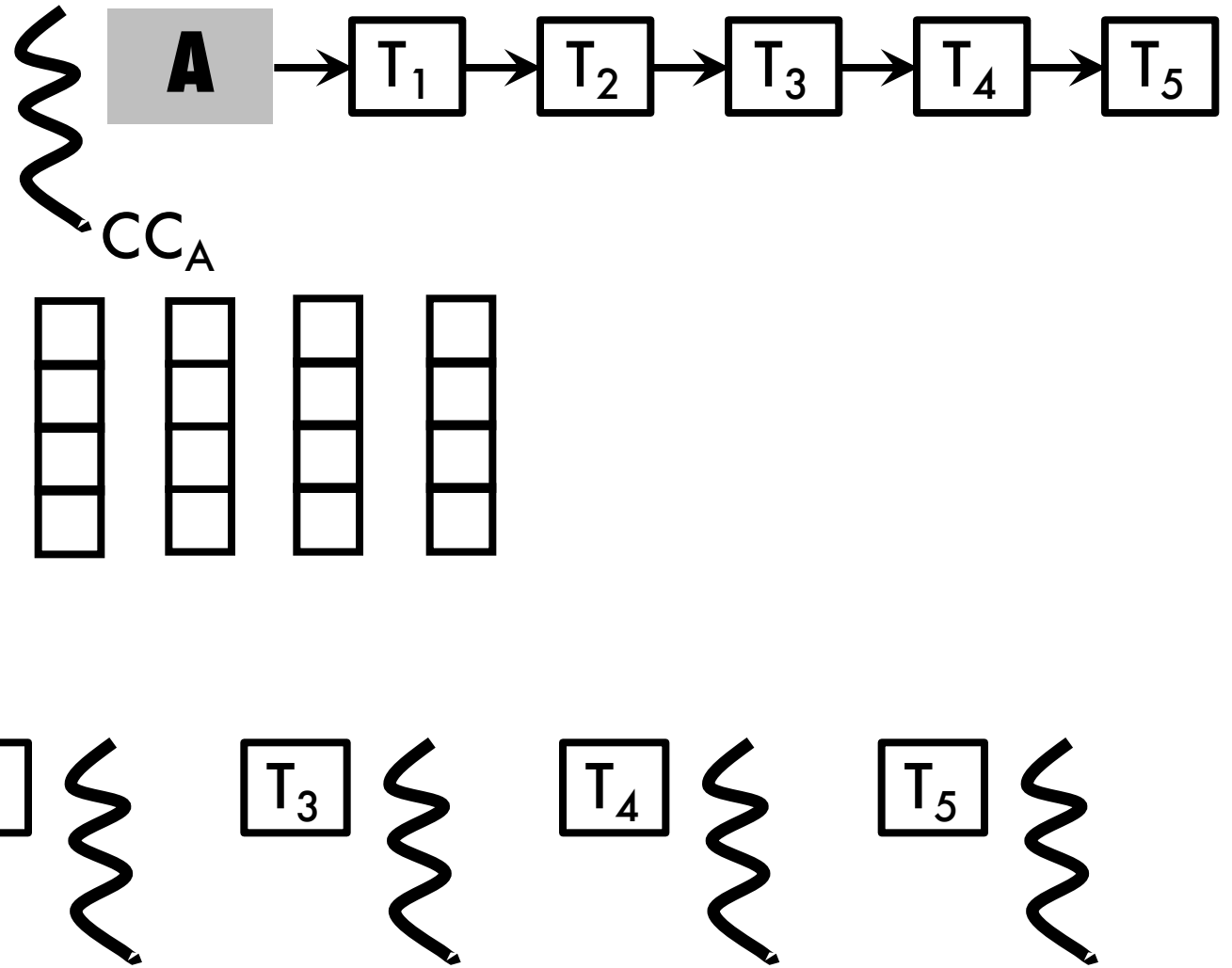
Example: Logical lock acquisition

- Enqueue lock request
- **Acquire lock**

One core manipulates lock list

List cannot "bounce" around cores

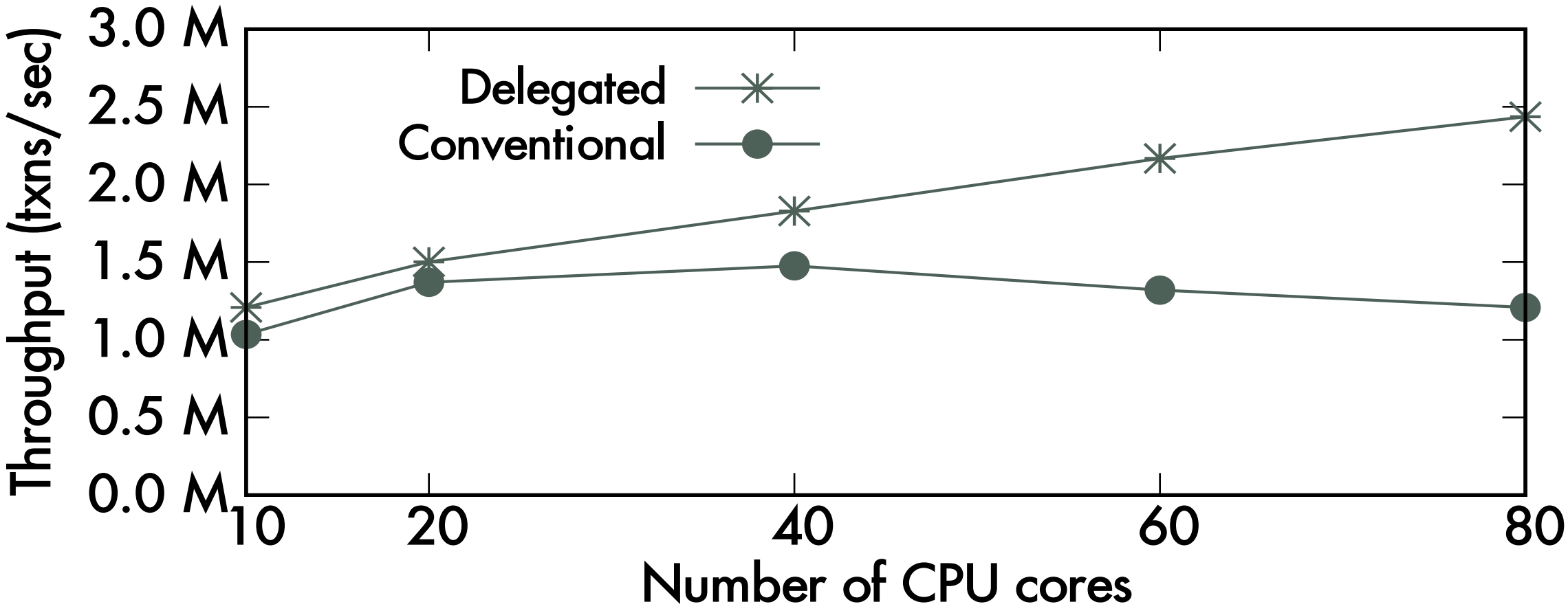
List likely remains cached under high contention



TPC-C NewOrder and Payment

- 16 Warehouses
- 80 core machine

TPC-C NewOrder and Payment



Observations

- Could be adapted to any concurrency control protocol
  - Indeed, to any multi-core DB **sub-system**
- Key idea: Delegate functionality to threads
  - E.g., concurrency control v.s. execution
  - Message-passing for communication
- Message-passing may be inevitable on heterogeneous hardware

Examples of delegating functionality

- Delegating functionality has been successfully used in a variety of domains

- Multi-core indexing – Physiological partitioning (PLP), PALM
- Distributed OCC validation – Hyder, Centiman
- Multi-core MVCC – Bohm, Lazy transactions

Conclusions

- DB implementations cannot circumvent workload conflicts
  - Workload conflicts result in data-structure contention
- Transaction to thread assignment causes unbounded data-structure contention
- Delegate **functionality** to threads to bound contention

If your DB is in this position…


KEEP
CALM
AND
DELEGATE