Distributed Transactional Contention Management as the Traveling Salesman Problem

Bo Zhang, Binoy Ravindran, Roberto Palmieri

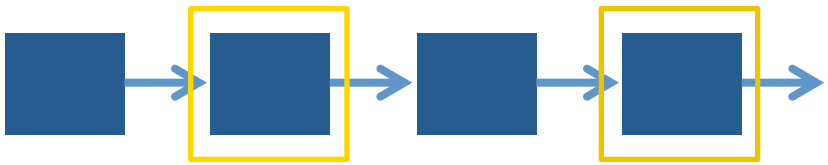
Virginia Tech

SIROCCO 2014

Virginia Tech

Lock-based concurrency control has serious drawbacks

- Coarse grained locking
  - Simple
  - But no concurrency



```
public boolean add(int item) {
  Node pred, curr;
  lock.lock();
  try {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
  } finally {
    lock.unlock();
  }
}
```

Fine-grained locking is better, but…

- Excellent performance
- Poor programmability

- Lock problems don't go away!
  - Deadlocks, livelocks, lock-convoying, priority inversion,….

- Most significant difficulty – composition

```
public boolean add(int item) {
  head.lock();
  Node pred = head;
  try {
    Node curr = pred.next;
    curr.lock();
    try {
      while (curr.val < item) {
        pred.unlock();
        pred = curr;
        curr = curr.next;
        curr.lock();
      }
      if (curr.key == key) {
        return false;
      }
       Node newNode = new Node(item);
       newNode.next = curr;
       pred.next = newNode;
       return true;
     } finally {
       curr.unlock();
     }
   } finally {
     pred.unlock();
   }
}
```

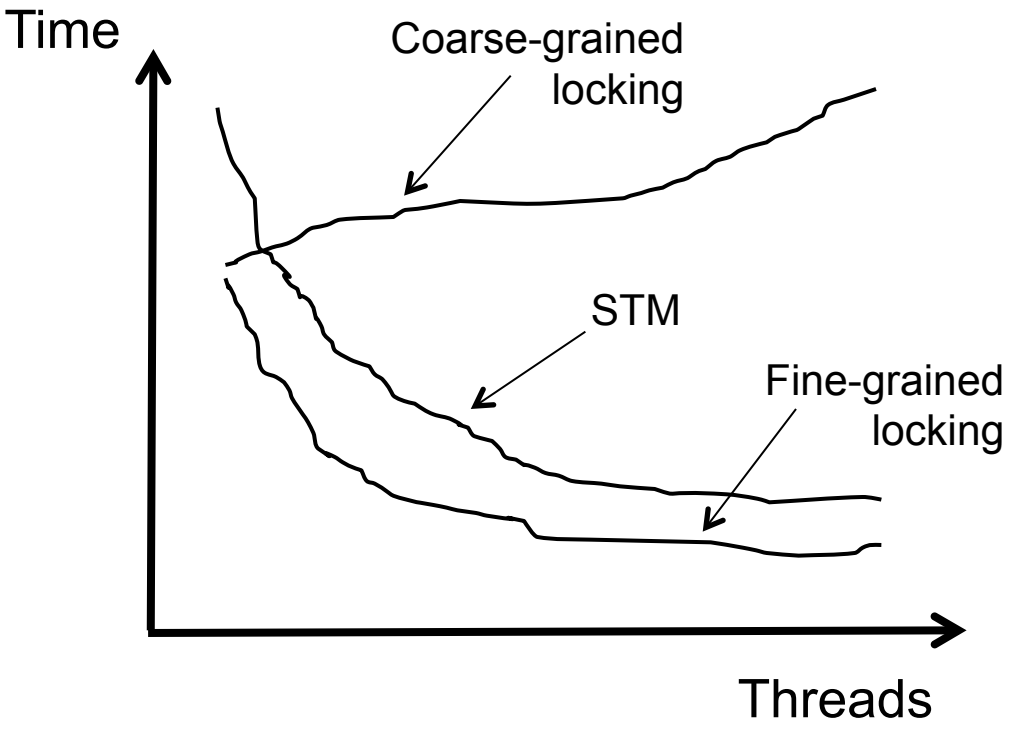
Transactional memory

- Like database transactions
- Easier to program
- Composable

- First HTM, then STM…now HyTM

```
public boolean add(int item) {
  Node pred, curr;
  atomic {
    pred = head;
    curr = pred.next;
    while (curr.val < item) {
      pred = curr;
      curr = curr.next;
    }
    if (item == curr.val) {
      return false;
    } else {
      Node node = new Node(item);
      node.next = curr;
      pred.next = node;
      return true;
    }
}
```

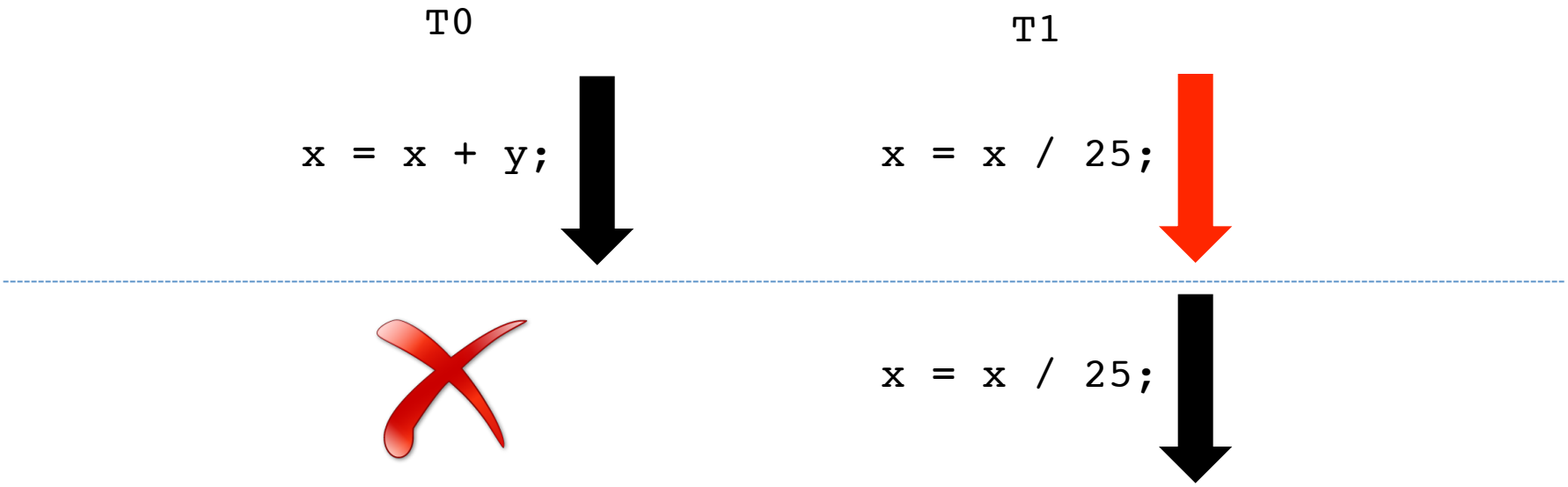
Optimistic execution yields performance gains at the simplicity of coarse-grain, but no silver bullet



- High data dependencies
- Irrevocable operations
- Interaction between transactions and non-transactions
- Conditional waiting
- ……

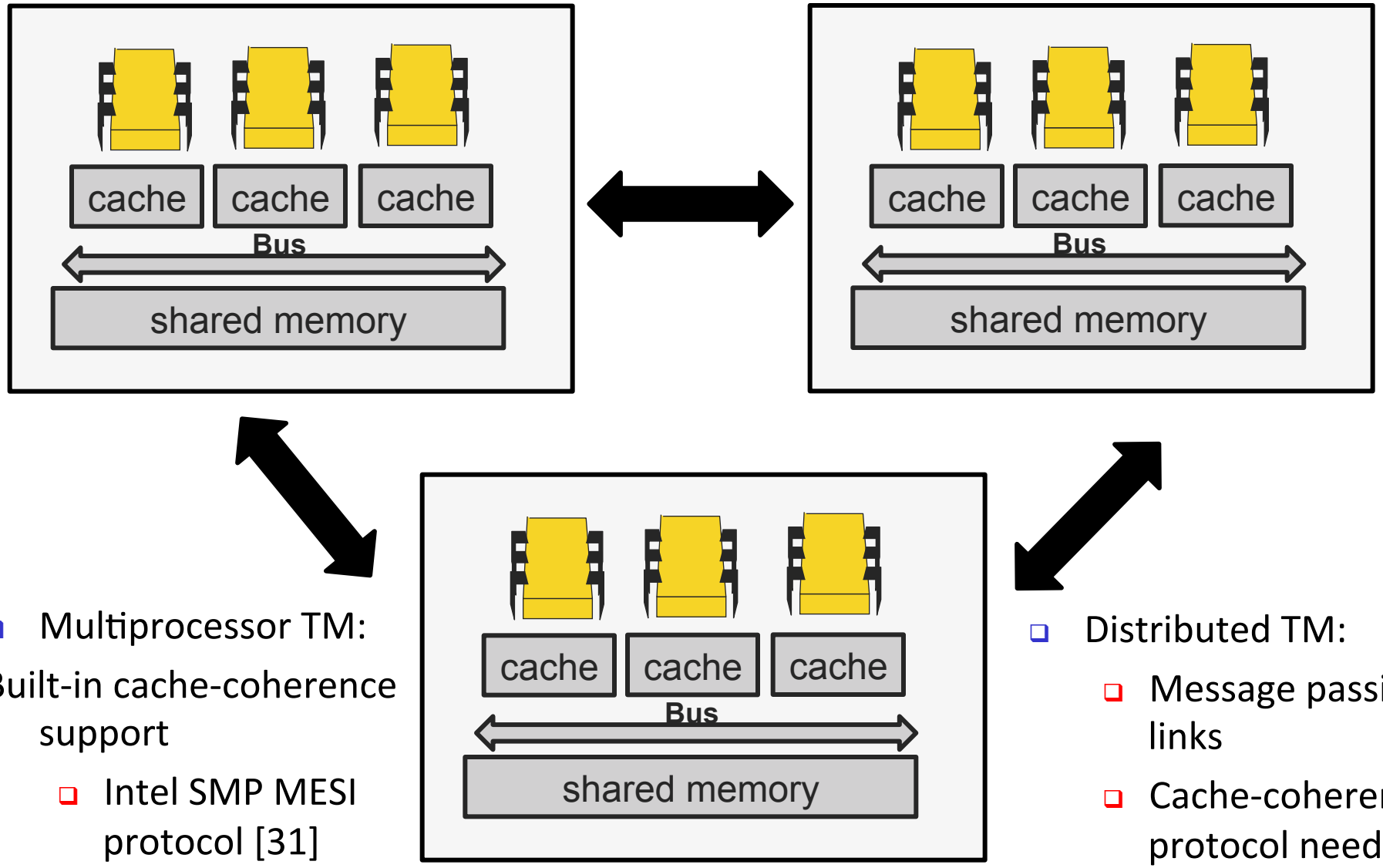
E.g., C/C++ Intel Run-Time System STM (B. Saha et. al. (2006). McRT-STM: A High Performance Software Transactional Memory. *ACM PPoPP*)

Contention management. Which transaction to abort?



- Contention manager
  - Can cause too many aborts, e.g., when a long running transaction conflicts with shorter transactions
  - An aborted transaction may wait too long

From Multiprocessor to Distributed Systems (from STM to DTM)



- Multiprocessor TM:

Built-in cache-coherence support

  - Intel SMP MESI protocol [31]

- Distributed TM:
  - Message passing links
  - Cache-coherence protocol needed

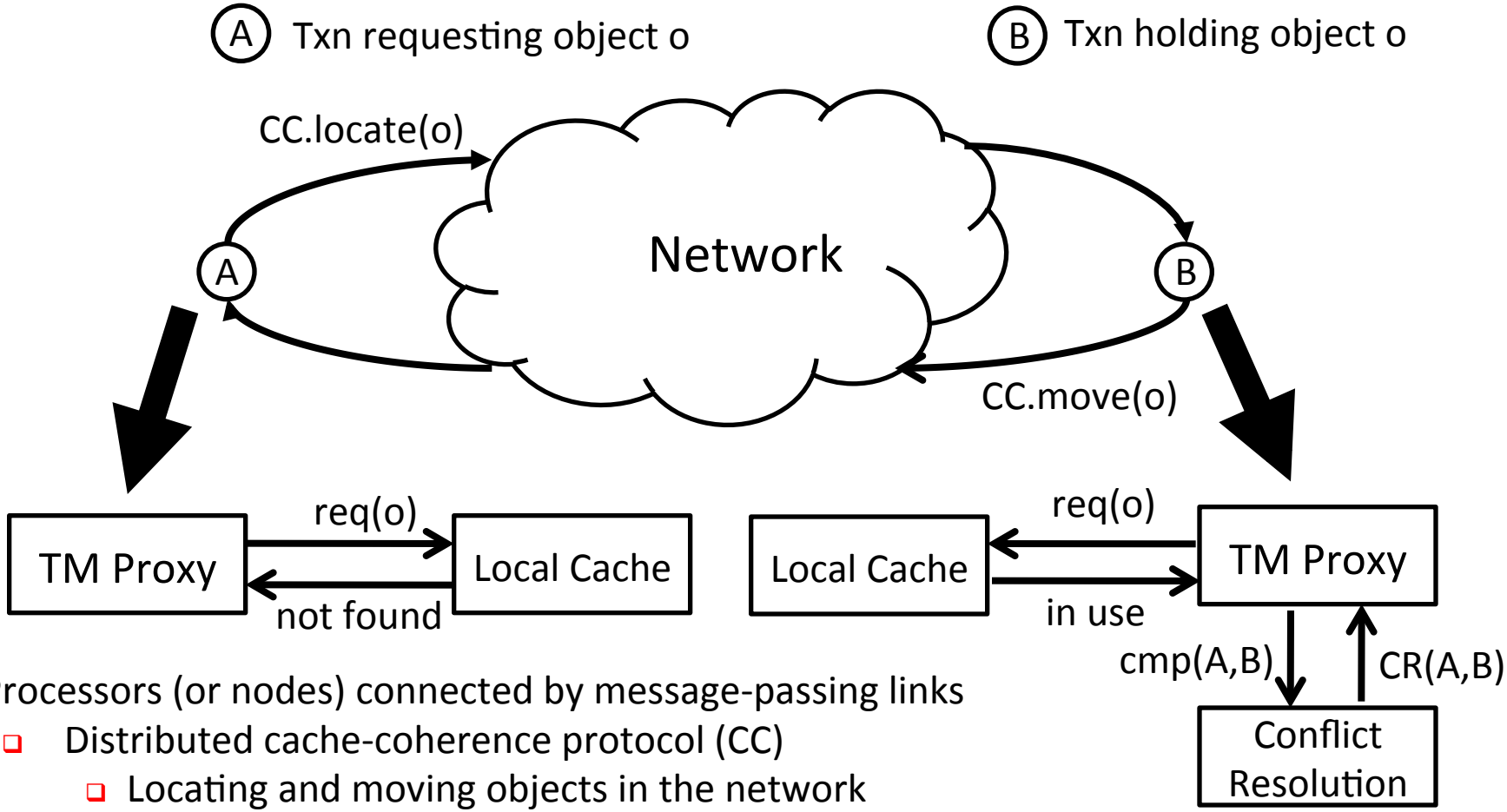
D-STM problem space

- Cache-coherence protocol
  - Locate and move objects in the network
  - Guarantee the consistency over multiple object copies
- Conflict resolution
  - Conservative approach
  - Non-conservative approach
  - Key property: guarantee progress
- Fault-tolerance
  - Network with node failures
  - Replication protocol: manage object replicas

Transaction execution models in DTM

- Control flow
  - Moving transactions, objects are held locally
  - Consistency: distributed commit protocol
  - Inherit the database transactional synchronization
- Data flow
  - Move the object to run all transactions locally
  - Synchronization: optimistic
  - Conflicts are resolved by conflict resolution strategy
  - No need for a distributed commit protocol
  - Easier to exploit locality

DTM, how it works



- Processors (or nodes) connected by message-passing links
  - Distributed cache-coherence protocol (CC)
    - Locating and moving objects in the network
    - Maintaining consistency among multiple copies of an object
  - Conflict resolution module (CR)
    - Resolve conflicts among transactions
    - How to make the correct/optimal decision?

Design Goal

- Input
  - The distributed system: nodes communicate via message passing links
  - $\mathcal{T}$: a set of *n* transactions accessing *s* shared objects in a metric-space network of *m* nodes
  - A: conflict resolution strategy
  - C: cache-coherence protocol.
- Output
  - makespan(A,C): the total time needed to complete the set of transactions under (A,C).
- Goal: maximize the throughput by minimizing makespan(A,C) over all possible combinations of input (A,C).

Measures of Quality

- Compare with an optimal clairvoyant off-line scheduling algorithm OPT.
  - OPT has all transactions' knowledge in advance.
  - Each transaction is scheduled exactly once under OPT.

- Competitive ratio: evaluate the optimality of makespan(A,C)

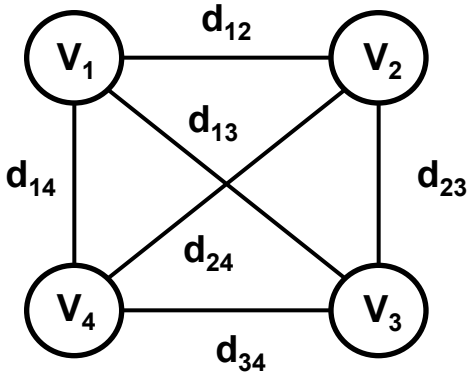
  - $CR(A,C)$ = max ($\mathbf{makespan}(A,C)$/($\mathbf{makespan}(OPT)$)

Problem statement

- We can consider the transaction scheduling problem for multiprocessor STM as a subset of the transaction scheduling problem for DTM. The two problems are equivalent as long as the communication cost can be ignored, compared with the local execution time duration.
- We model contention management as a non-clairvoyant scheduling problem
- If all transactions are conflicting each other, then a sequential schedule is the best solution.

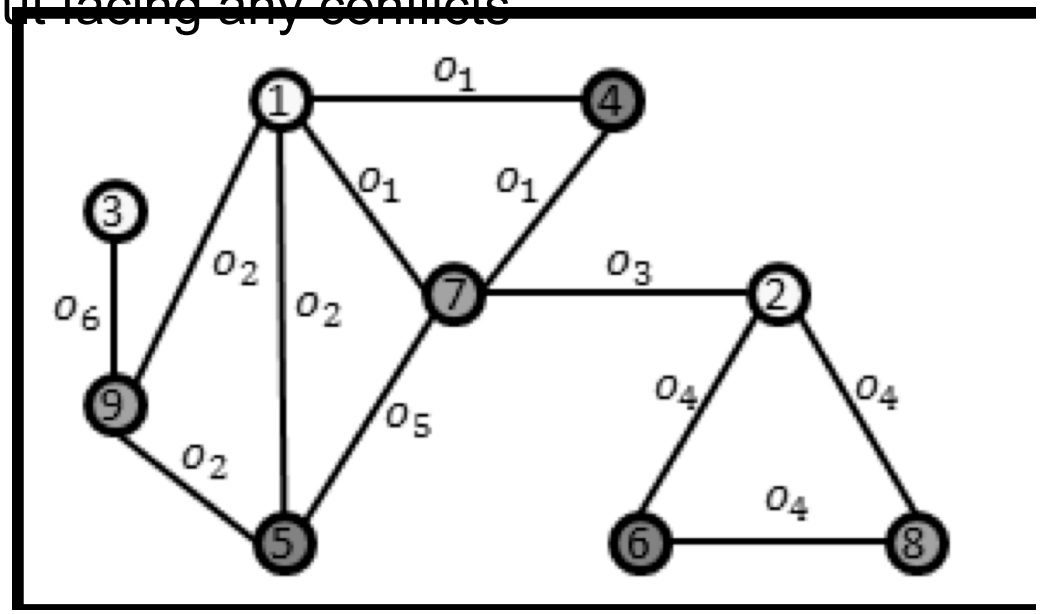
Towards the optimal: Cost Graph

- Cost Graph:
  - each node in the system is a vertex in the graph
  - each edge (v_i, v_j) represents the channel to move an object from node v_i to node v_j
  - each edge (v_i, v_j) is weighted with the cost (d_{ij}) for moving the object from node v_i to node v_j



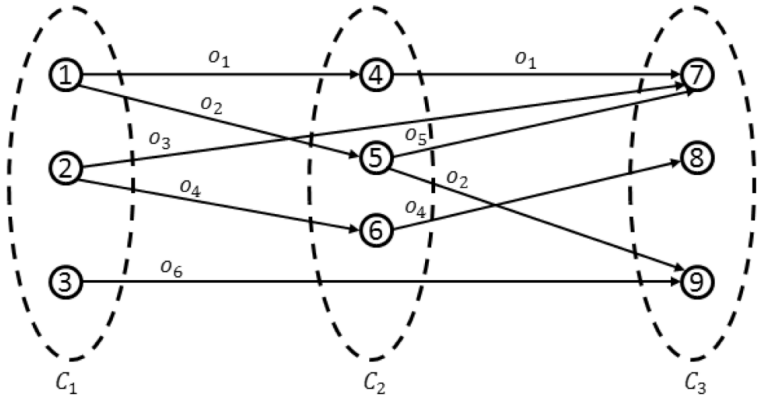
Towards the optimal: Conflict Graph

- Conflict Graph:
  - each transaction is represented as a numbered node
  - each edge is marked with the object which causes transactions to conflict
  - we can construct a coloring of the conflict graph
  - since transactions with the same color are not connected, every set Ci forms an independent set and can be executed in parallel without facing any conflicts



Towards the optimal: Ordering Conflict Graph

- An optimal offline schedule Opt determines a *k*-coloring of the conflict graph and an execution order (the ordering conflict graph) such that for any two sets C_i and C_j, where $i < j$, if T_1 and T_2 conflict, and T_1 is in C_1 and T_2 in T_2, then T_2 is postponed until T_1 commits
- There are *k*! ordering conflict graphs
- The ordering conflict graph is weighted:
  - Node's weight is the transaction's execution time
  - Arc's weight is the cost for moving the object from the source node to the destination node



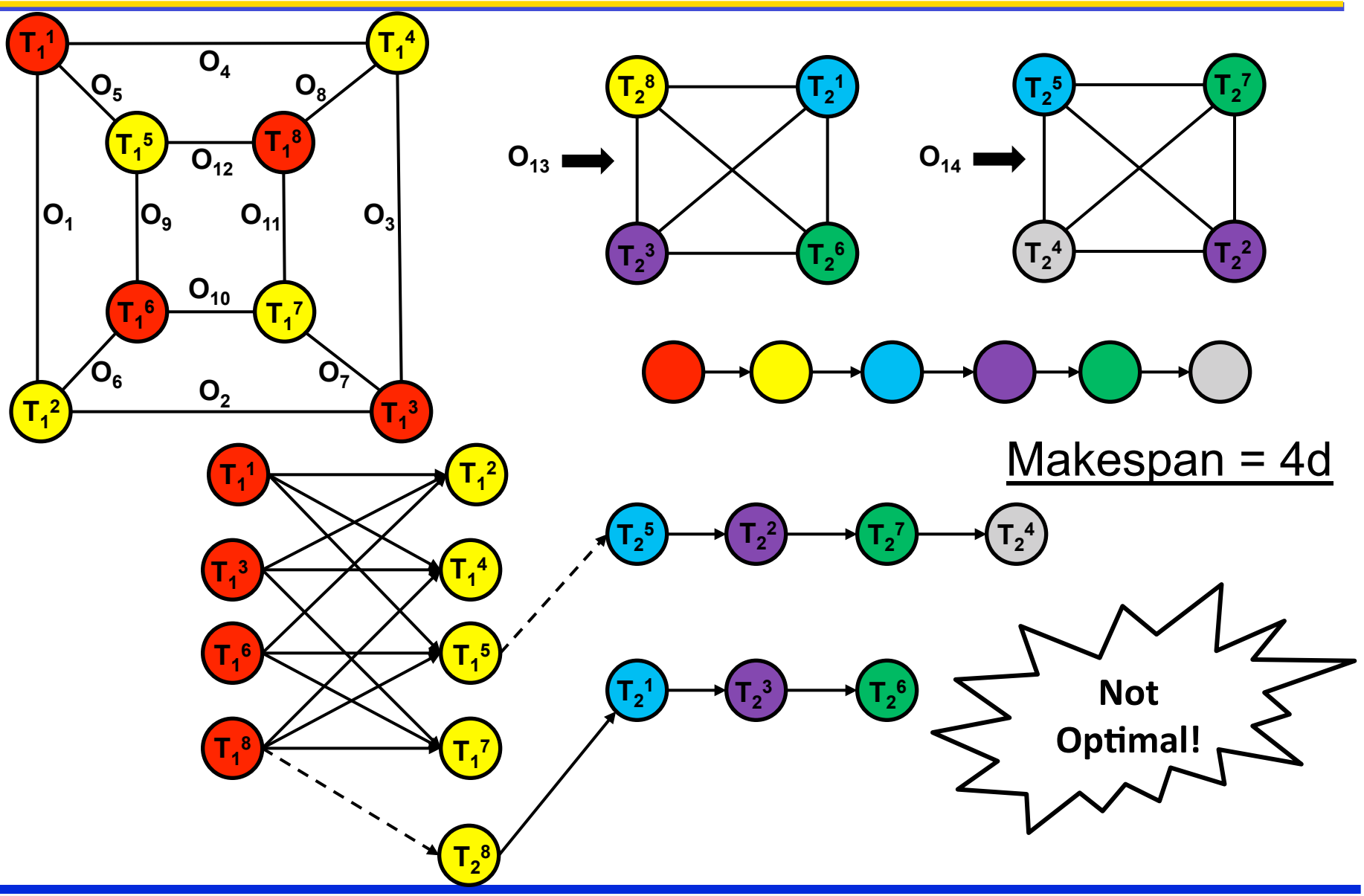
…but the optimal is too complex

- The commit time of a transaction T is determined by one of the weighted paths that ends at T
- The makespan is the weight of the longest weighted path in the ordering conflict graph
- The optimal is reached selecting the ordering conflict graph that minimizes the makespan
- Finding an optimal contention manager is (NP-) hard
  - If each node issues only one transaction and cost of moving objects is negligible, then the problem is equivalent to finding the chromatic number of the conflict graph
  - If the number of shared object is one, the problem is equivalent to finding the traveling salesman path (TSP) in the cost graph

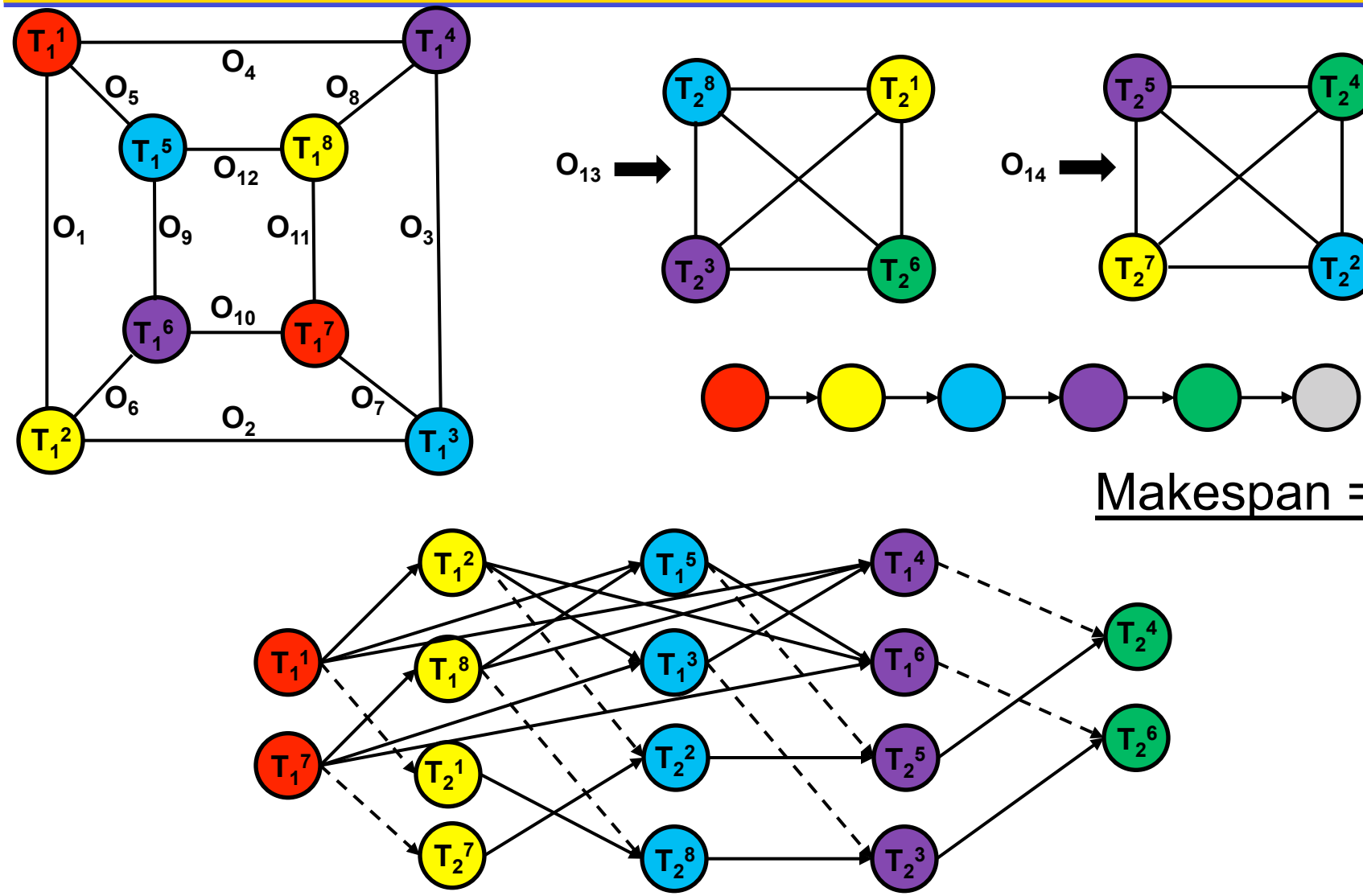
Also…

- When each node generates a sequence of transactions, it is not always optimal to schedule transactions according to the ordering conflict graph since the conflict graph evolves over time, an optimal schedule based on a static conflict graph may lose potential parallelism in the future.

Local optimality is not global optimality (2-coloring)



Local optimality is not global optimality (4-coloring)



Lower Bounds

- For STM, any online deterministic CM is *Ω(s)*-competitive, where *s* is the number of objects [Attiya et al. '06]
- For DTM, any online deterministic work conservative CM is

$$\Omega(\max[s, \frac{s^2}{\overline{D}}])$$

  optimal, where $\overline{D}$ is the normalized network diameter
- When the normalized network diameter is bounded (*D* is a constant), it can only provide a *Ω(s²)*-competitive ratio.

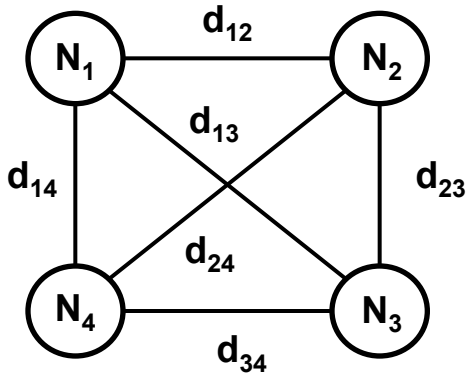
<u>Can we find an approximate optimal solution in reasonable time?</u>

CUTTING

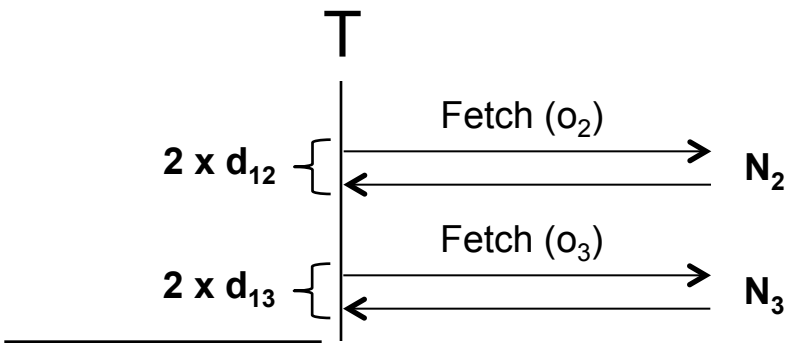
- CUTTING is a randomized scheduling algorithm based on partitioning the cost graph
- Assumptions:
  - Transaction T_i knows its required set of objects after it starts
  - We assume that the moving cost is bounded at *D*
- Input:
  - A set of transactions with their execution time
  - The conflict graph
  - The cost graph
  - An approximate TSP algorithm (ATSP)
- Output:
  - A schedule for executing transactions

Why TSP and why that assumption?

- T invoked by N_1
- T_1 writes objects $\{o_2, o_3\}$
- N_2 stores o_2; N3 stores o_3

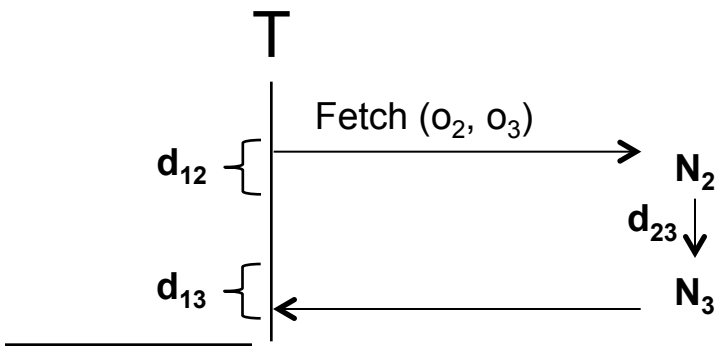


Without assumption



$(2 \times d_{12}) + (2 \times d_{13}) = \sim 4d$

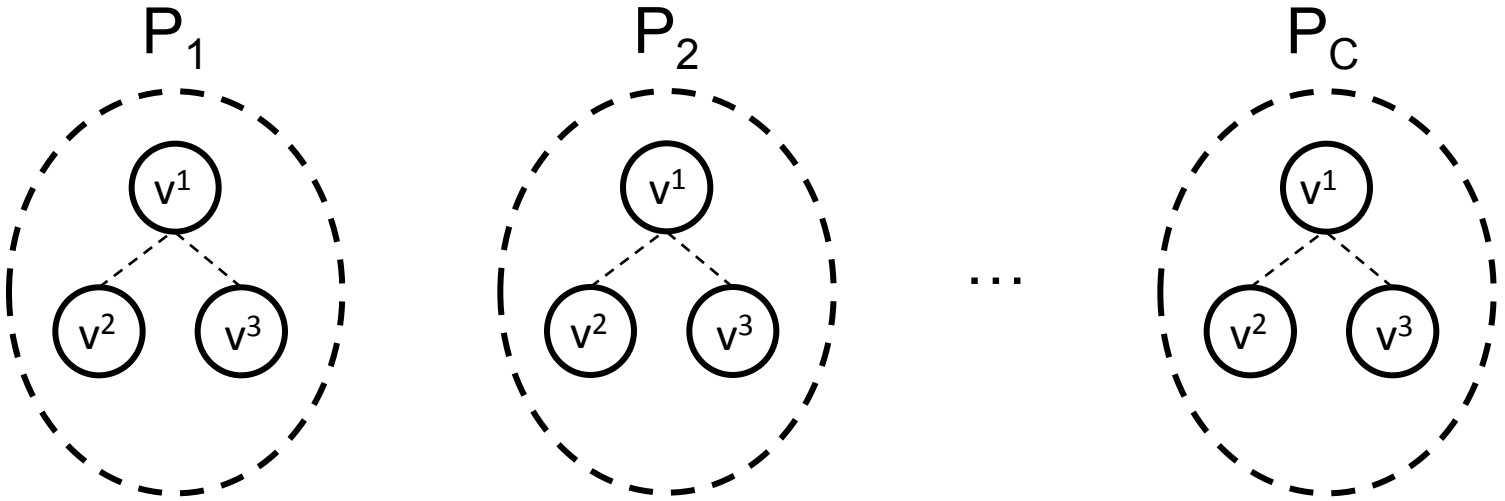
With assumption



$d_{12} + d_{23} + d_{13} = \sim 3d$

Cutting: how it works

- The cost graph is partitioned in C partitions such that for any pair of nodes (v_i, v_j) belonging to one partition, $d_{ij} \leq$ ATSP/C
- Within each partition:
  - Nodes are numbered with an integer from 1 to the size of the partition
  - A binary tree is built following nodes' numbers
- Each transaction randomly selects an integer that is used for deciding the transaction to abort after a conflict



Cutting: how it works

- Handling conflicts between two transactions:
  - Phase 1:
    - Within the same partition and one transaction is an ancestor of the other in the partition's binary tree, the node that precedes the other in the ATPS path aborts the other
    - Otherwise the transaction with the lesser partition number aborts the other
  - Phase 2:
    - Each transaction randomly selects an integer (π) when it starts or restarts. If one transaction is not an ancestor of the other, the transaction with the lower π proceeds and the other transaction aborts.
  - Whenever a transaction is aborted by a remote transaction, the requested object is moved to the remote transaction immediately.

Cutting: analysis

- The average case competitive ratio of Cutting is

$$O\left(s \cdot \phi_A \cdot \log^2 m \log^2 n\right)$$

for *s* objects shared by *n* transactions invoked by *m* nodes

- This is close to the multiprocessor bound of *O(s)*

Other results on Cutting

- A transaction T needs:

$$O\left(C \log^2 m \log n\right)$$

  trials from the moment it is invoked until it commits, on average

- The average response time of a transaction is:

$$O\left(C \log^2 m \log n \cdot (\tau + \frac{\text{ATSP}_A}{C})\right)$$

- The average-case competitive ratio of Cutting is

$$O\left(s \cdot \phi_A \cdot \log^2 m \log^2 n\right)$$

Thanks!

Questions?

Research project's web-site: www.hyflow.org