FEAT

Fair and Efficient Adversarial Transaction Ordering

Tien Tuan Anh Dinh — Deakin University
Dakai Kang — University of California, Davis; Sei Labs
Mohammad Sadoghi — University of California, Davis

ACM Symposium on Principles of Distributed Computing (PODC '26) · Egham, United Kingdom

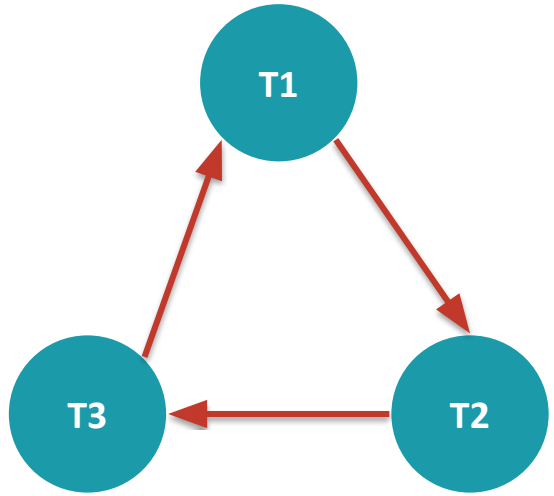
Transaction Ordering & MEV

- Transaction ordering is central to Byzantine consensus and blockchain systems.
- **Maximal Extractable Value (MEV) attacks:** Adversarial participants manipulate transaction ordering to extract profit.
- Uncontrolled MEV undermines both economic fairness and trust in the system.

Goal: generate a fair final transaction ordering based on the ordering preferences of a majority of the participants.

Why Fairness is Hard

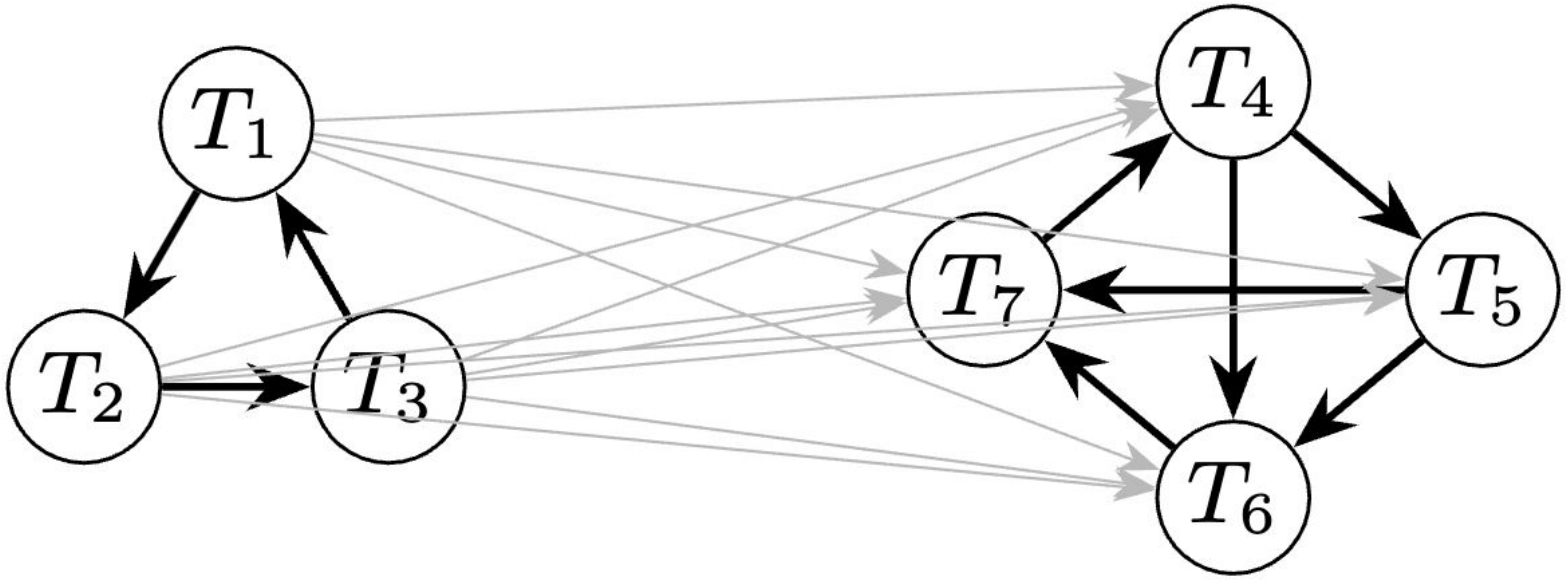
- Receive-Order-Fairness: if enough honest replicas see T before T', all honest replicas must output T **before** T'.
- Aeqitas [2020]: impossible in general — replica orderings can form Condorcet cycles.
- Batch-Order-Fairness (relaxation): partition the final ordering into batches;
- If enough honest replicas see T before T', all honest replicas must output T **no later than** T', i.e., T is in the same or an earlier batch than T'.



State of the Art: the Quadratic Bottleneck

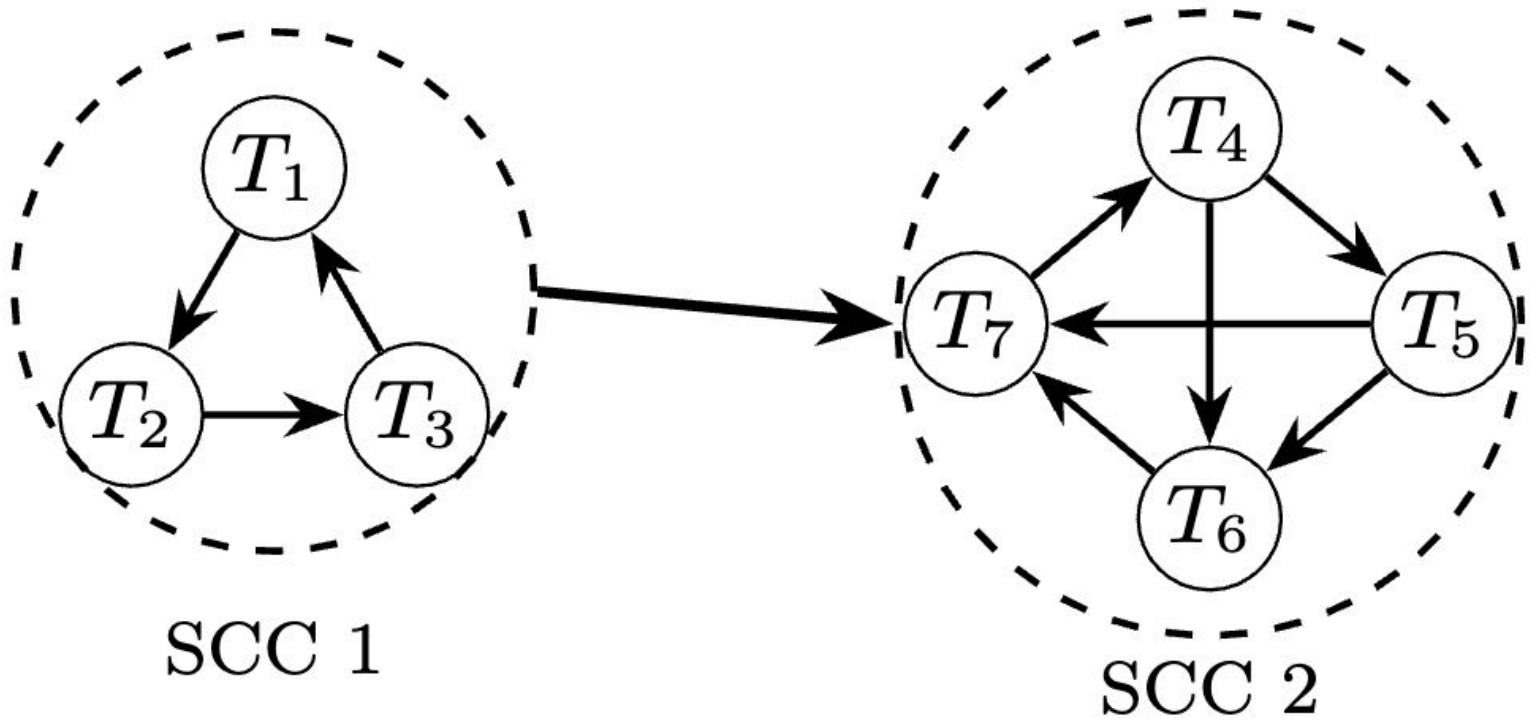
Step 1: Build a tournament graph of transactions.

- Nodes: Transactions
- Edges: Global pairwise ordering preference (derived from local ordering preferences of a majority of replicas.)



State of the Art: the Quadratic Bottleneck

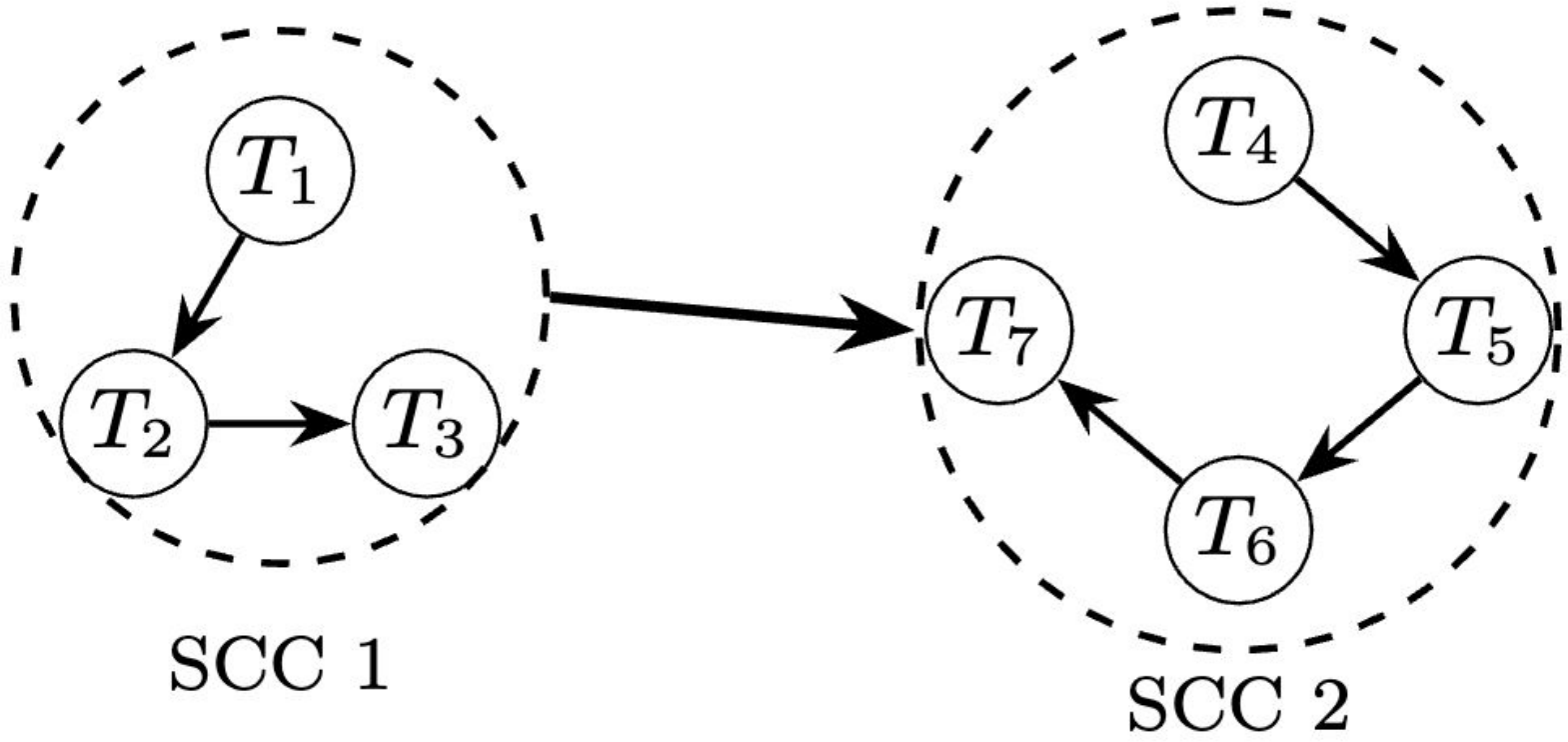
Step 2: Find Strongly Connected Components (SCC) and topologically sort them.



State of the Art: the Quadratic Bottleneck

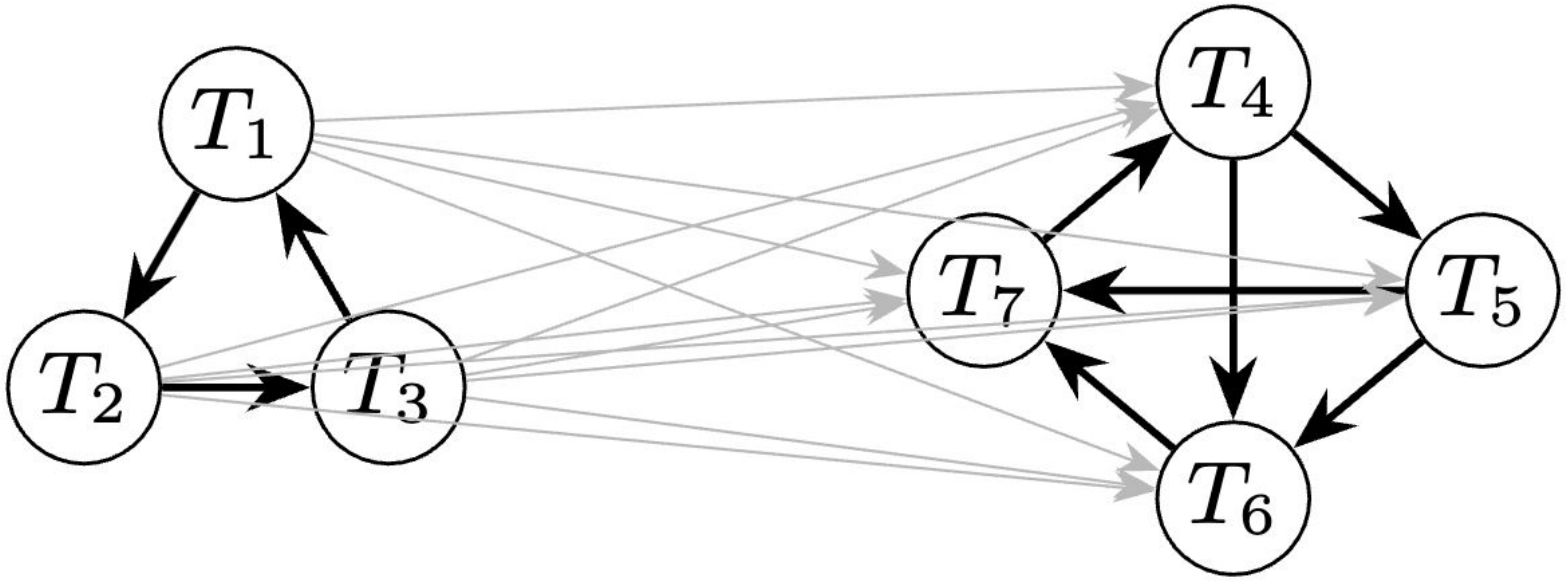
Step 3: Find a Hamilton Path within each SCC and generate the final ordering.

- **T1 → T2 → T3 → T4 → T5→ T6→ T7**

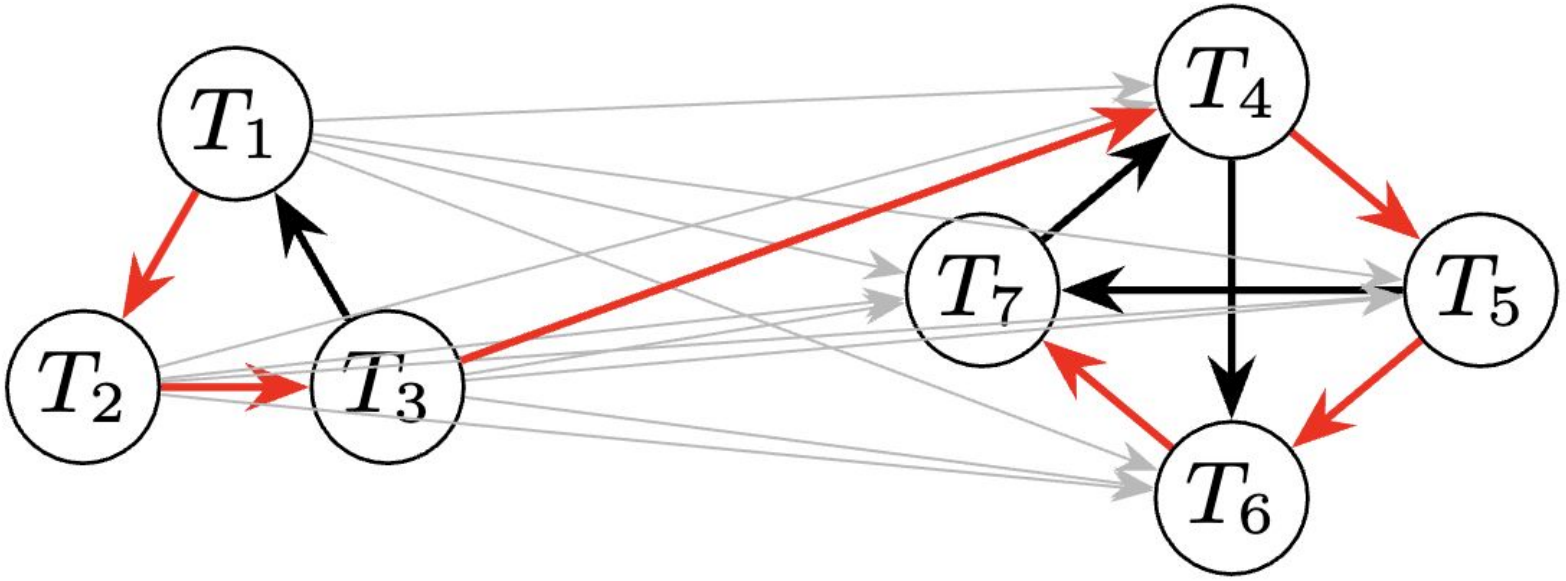


Is the Quadratic Cost Inherent?

Is the $\Theta(|T|^2)$ comparison complexity of building a tournament graph inherent to enforcing Batch-Order-Fairness.



Is the Quadratic Cost Inherent?



- Denoting ***cmp*** by the global ordering preference
- Adjacent-Comparator Consistency:

  for every transaction T and its successor transaction T', ***cmp***(T, T') = true, i.e., T⟶T'

- We observe that the final ordering satisfies *Adjacent-Comparator Consistency*

Adjacent-Comparator Consistency

Could we find a final ordering satisfying Adjacent-Comparator Consistency **directly without building a tournament graph?**

- If Adjacent-Comparator Consistency is satisfied, is Batch-Order-Fairness satisfied?
  It looks intuitive, and the answer is **yes** (proved in the paper)

- What is the complexity of finding such a transaction ordering satisfying Adjacent-Comparator Consistency?

Adjacent-Comparator Consistency

What is the **comparison complexity** of finding an ordering satisfying **Adjacent-Comparator Consistency**?

- we view it as a comparator-based sorting problem.
- The global ordering preference comparator ***cmp*** is *complete but non-transitive*.
- ***complete***: for any T and T', at least one of ***cmp*** (T, T') = True and ***cmp*** (T', T) = True holds
- ***non-transitive***:

  cmp (T1, T2) = True and ***cmp*** (T2, T3) = True;  ***cmp*** (T1, T3) = True

Adjacent-Comparator Consistency

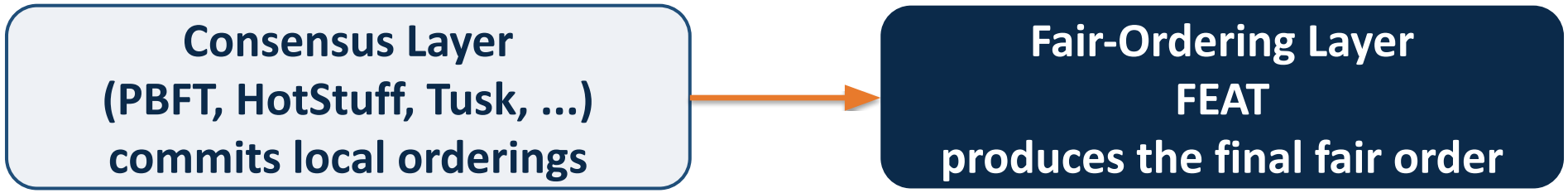
- If the comparator is ***transitive***, *the problem can be reduced from comparator-based sorting, whose comparison complexity bound is* $\Omega(|T|\log|T|)$.
- The worst-case lower bound of comparison complexity of finding such an ordering is $\boldsymbol{\Omega(|T|\log|T|)}$.

Can we design an algorithm that finds an ordering satisfying **Adjacent-Comparator Consistency** with **O(|T|log|T|) complexity?**

- If yes, we have an algorithm whose worst-case comparison complexity is tightly bounded by $\boldsymbol{\Theta(|T|\log|T|)}$

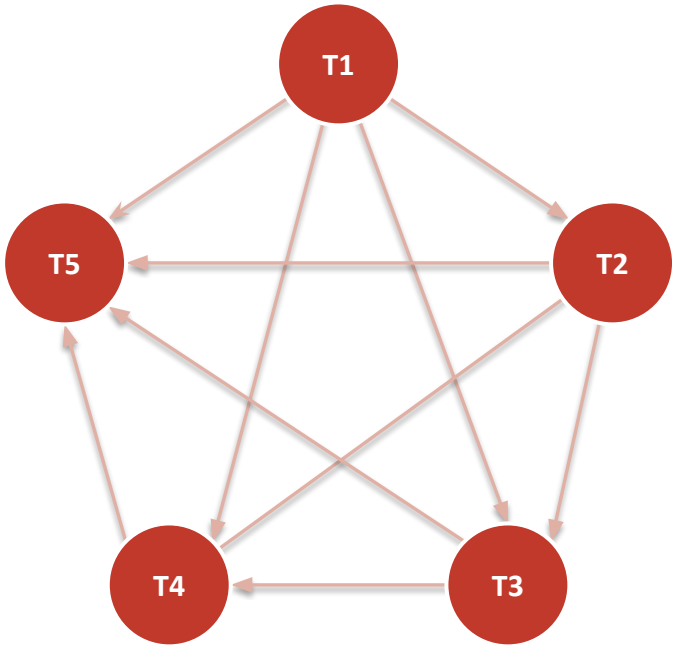
System Model

- n replicas, up to f Byzantine/adversarial; the rest are honest.
- $n > (2\gamma+1)f / (2\gamma-1)$, with $0.5 < \gamma \leq 1$ (reduces to the classic $n > 3f$ when $\gamma = 1$).
- γ is a parameter that controls fairness level of the final ordering.

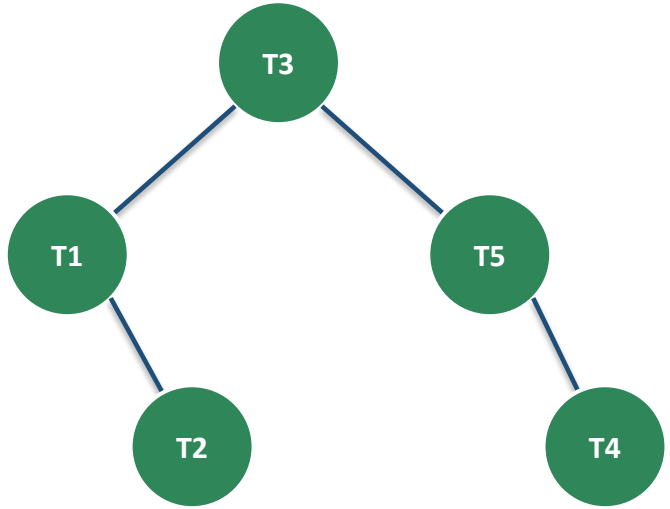


Key Idea: Tournament Graph → AVL Tree

Prior work: build the full graph



FEAT: insert into an AVL tree



FEAT: Transaction Classification

- Each transaction gets a score = number of replicas whose committed local ordering includes it.

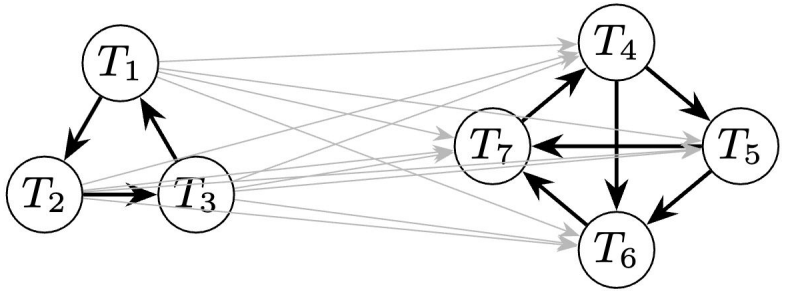
- Comparator Decidability Threshold $\tau = (n-f)/2$:
  cmp(T1, T2) = True if ***score(T1) > score(T2)*** *and* ***score(T1) >= τ***

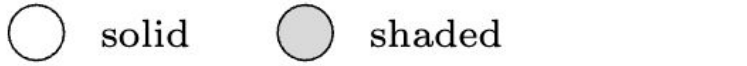
- Classify each transaction as blank / shaded / solid
  - solid (score ≥ 2τ) — decidable against every other transaction.
  - shaded (τ ≤ score < 2τ) — comparator with another shaded transaction may be undecidable yet.
  - blank (score < τ) — not yet eligible for ordering this round, will be retry in future rounds.

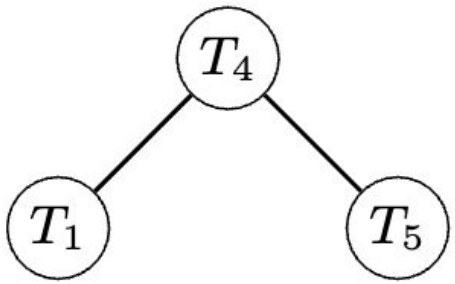
- for example, n = 7, f = 2, solid requires score >= 5, shaded requires score >= 2.5,

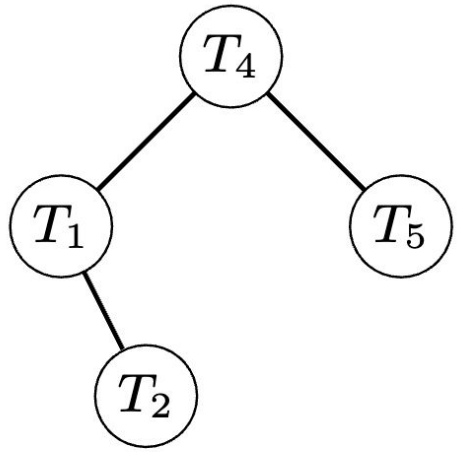
Example: 7 Transactions

- Assuming an Insertion order: T1, T4, T5, T2 (solid) then T7, T3, T6 (shaded).





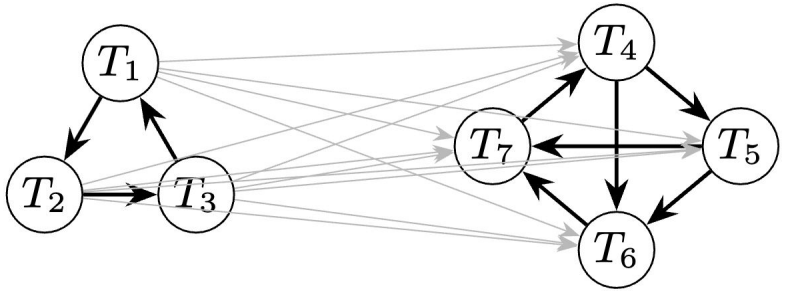


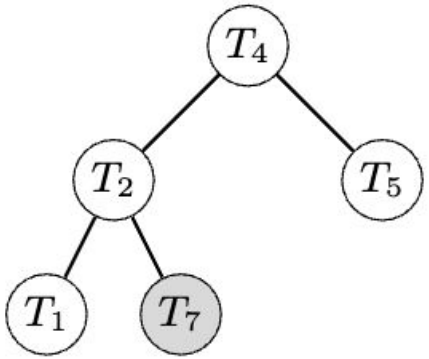


Step 1: Insert T_1 **Step 2: Insert T_4 (right of T_1)** **Step 3: Insert T_5; left-rotate at T_1** **Step 4: Insert T_2**

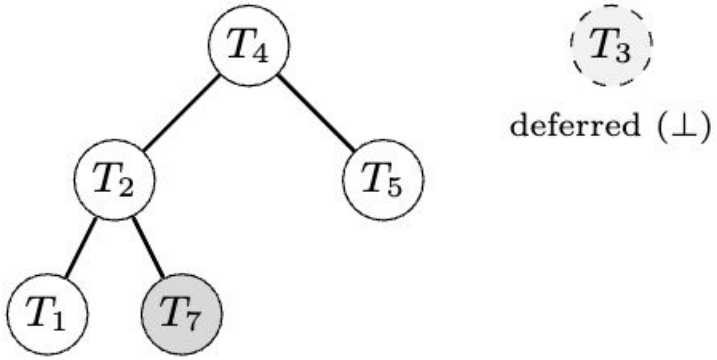
Example: 7 Transactions

- Assuming an Insertion order: T1, T4, T5, T2 (solid) then T7, T3, T6 (shaded).

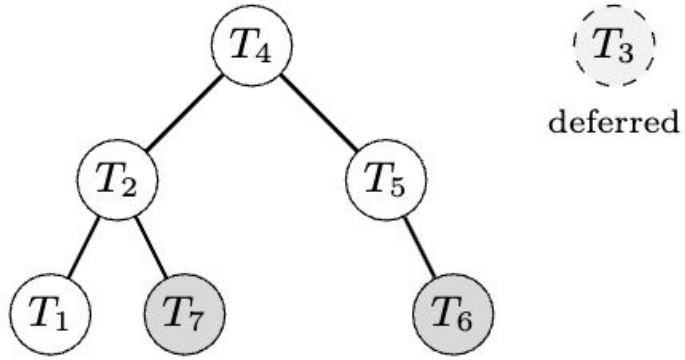




Step 5: Insert shaded T_7; left-rotate at T_1



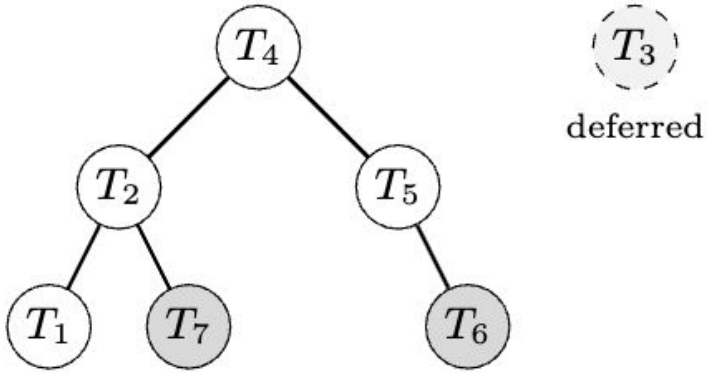
Step 6: Insert T_3; aborts at $\mathrm{cmp}(T_3, T_7) = \perp$



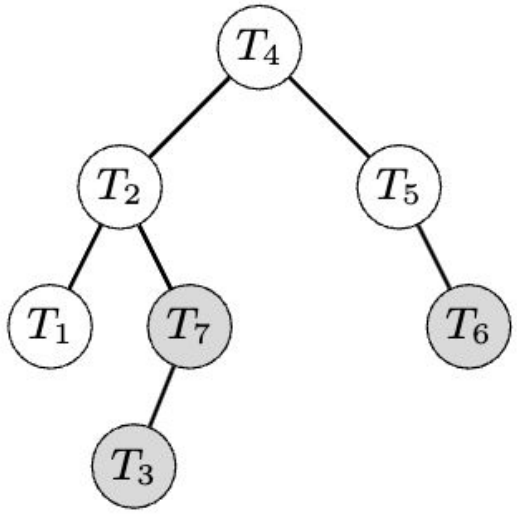
Step 7: Insert shaded T_6

Example

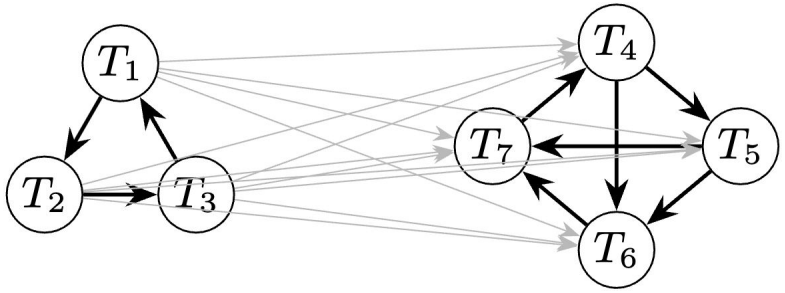
- Assuming an Insertion order: T1, T4, T5, T2 (solid) then T7, T3, T6 (shaded).



Step 7: Insert shaded T_6



Step 8: Round $r' > r$; retry T_3 as left child of T_7



- The final ordering is the **in-order traversal** of the AVL-Tree:

T1 → T2 → T3 → T7 → T4 → T5 → T6

FEAT: Correctness Analysis

- Is the Adjacent-Comparator Consistency satisfied?

Proof sketch:

(1) Assuming the in-order traversal of the AVL-Tree before inserting T satisfies it.

(2) Condition (1) holds true when the size of the AVL-Tree is 0 or 1.

(3) After inserting T, in the new in-order traversal, the two adjacent transactions of T, p and s, must be on the insertion path from root to T. (AVL-Tree property)

(4) The AVL-Tree rotation/rebalancing does not change the in-order traversal.

By induction, the Adjacent-Comparator Consistency is satisfied.

Thus, the Batch-Order-Fairness is satisfied.

FEAT: Complexity Analysis

(1) Each AVL-Tree insertion is O(log|T|)

(2) We assume each shaded transaction is retried for at most O(1) times

Thus, the comparison complexity is O(|T| log|T|)

Stitching Across Rounds

- Each round's AVL in-order traversal gives a round ordering RO_r.
- Within each round, Adjacent-Comparator Consistency holds, but it may not hold across rounds.
- We maintain a global TailList, and merge it with each RO_r
- Example:

  TailList: **T_A**→ **T_B** → **T_C**, RO_r: **T1** → **T2** → **T3** → **T7** → **T4** → **T5** → **T6**

  cmp(T_A, T_1) = True, cmp(T_B, T_1) = True, cmp(T_C, T_1) = False, cmp(T_C, T_2) = True.

  Merged Ordering: **T_A**→ **T_B** → **T1** → **T_C** → **T2** → **T3** → **T7** → **T4** → **T5** → **T6**

Ordering Finalization

- After stitching, we finalize transactions in the merged ordering until the last solid transaction in it, which is T4 in this example

Merge Ordering: **T_A**→ **T_B** → **T1** → **T_C** → **T2** → **T3** → **T7** → **T4** → **T5** → **T6**

- Then, transactions from T_A to T4 are finalized, and T5 → T6 becomes the new TailList

Intuitively, the stitching algorithm has O(|T|) complexity and holds the Adjacent-Comparator Consistency. (*proved in the paper*)

Takeaways

- We proved the worst-case comparison complexity for Batch-Order-Fairness: $\Omega(|T| \log |T|)$.
- FEAT achieves the matching upper bound with AVL-tree construction within each round and stitching across rounds.

Thank You — Questions?

Tien Tuan Anh Dinh — anh.dinh@deakin.edu.au
Dakai Kang — dakai@seinetwork.io
Mohammad Sadoghi — msadoghi@ucdavis.edu

Paper: doi.org/10.1145/3796701.3815905