APM Seminar

Incremental FP-Growth

Josefine Harzmann, Sebastian Oergel

FP-Growth
revised

FP-Growth - Phase 1

- 2 phases: tree construction & frequent itemset mining

1	{#music, #jlo, #justin, #shaki, #us, i#britney, #interesting}
2	{#nice, #shaki, #jlo, #justin, #britney}
3	{#justin, #nice, #cute}
4	{#jlo, #nice, #us, #concert}
5	{#us, #music, #female, #britney, #jlo, #shaki, #justin}
6	{#shaki, #britney, #date, #justin, #us}

FP-Growth - Phase 1

- 2 phases: tree construction & frequent itemset mining
  - Phase 1
    - 1. counting frequencies of occurring items

1	{#music, #jlo, #justin, #shaki, #us, i#britney, #interesting}
2	{#nice, #shaki, #jlo, #justin, #britney}
3	{#justin, #nice, #cute}
4	{#jlo, #nice, #us, #concert}
5	{#us, #music, #female, #britney, #jlo, #shaki, #justin}
6	{#shaki, #britney, #date, #justin, #us}

FP-Growth - Phase 1

- 2 phases: tree construction & frequent itemset mining
  - Phase 1
    - 1. counting frequencies of occurring items
    - 2. filtering for frequent items & order items according to frequencies

1	{#music, #jlo, #justin, #shaki, #us, i#britney, #interesting}
2	{#nice, #shaki, #jlo, #justin, #britney}
3	{#justin, #nice, #cute}
4	{#jlo, #nice, #us, #concert}
5	{#us, #music, #female, #britney, #jlo, #shaki, #justin}
6	{#shaki, #britney, #date, #justin, #us}

FP-Growth - Phase 1

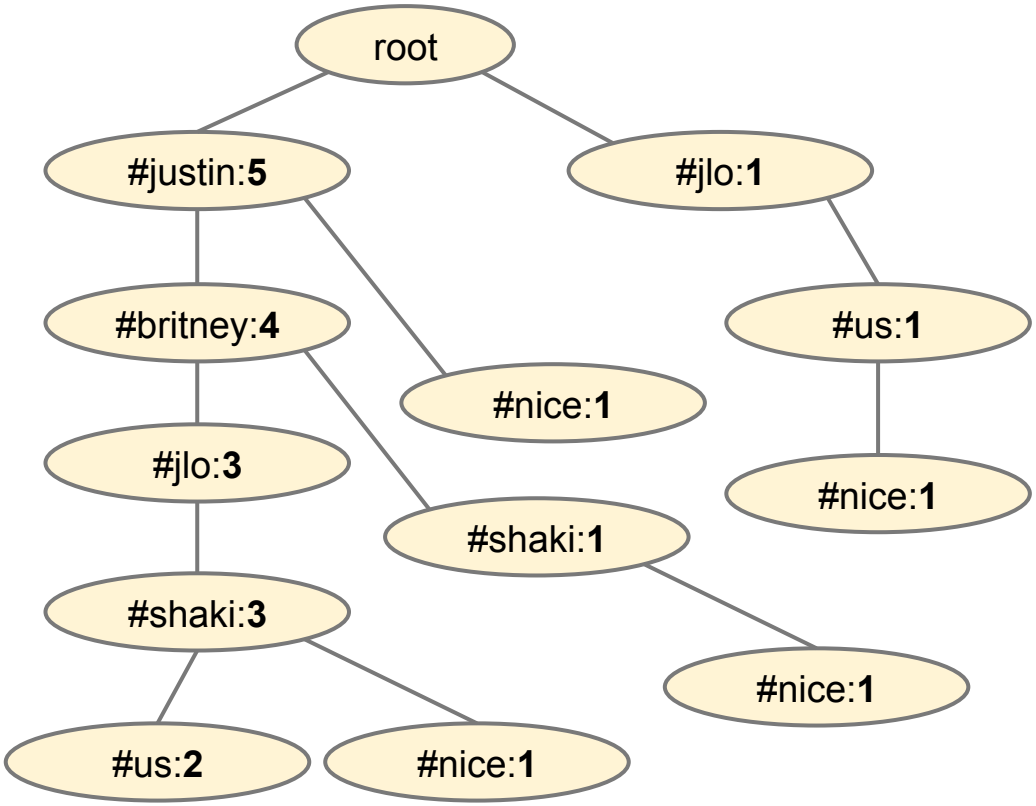
- 2 phases: tree construction & frequent itemset mining
  - Phase 1
    - 1. counting frequencies of occurring items
    - 2. filtering for frequent items & order items according to frequencies

1	{#music, #jlo, #justin, #shaki, #us, i#britney, #interesting}
2	{#nice, #shaki, #jlo, #justin, #britney}
3	{#justin, #nice, #cute}
4	{#jlo, #nice, #us, #concert}
5	{#us, #music, #female, #britney, #jlo, #shaki, #justin}
6	{#shaki, #britney, #date, #justin, #us}

1	{#justin, #britney, #jlo, #shaki, #us}
2	{#justin, #britney, #jlo, #shaki, #nice}
3	{#justin, #nice}
4	{#jlo, #us, #nice}
5	{#justin, #britney, #jlo, #shaki, #us}
6	{#justin, #britney, #shaki, #us}

FP-Growth - Phase 1

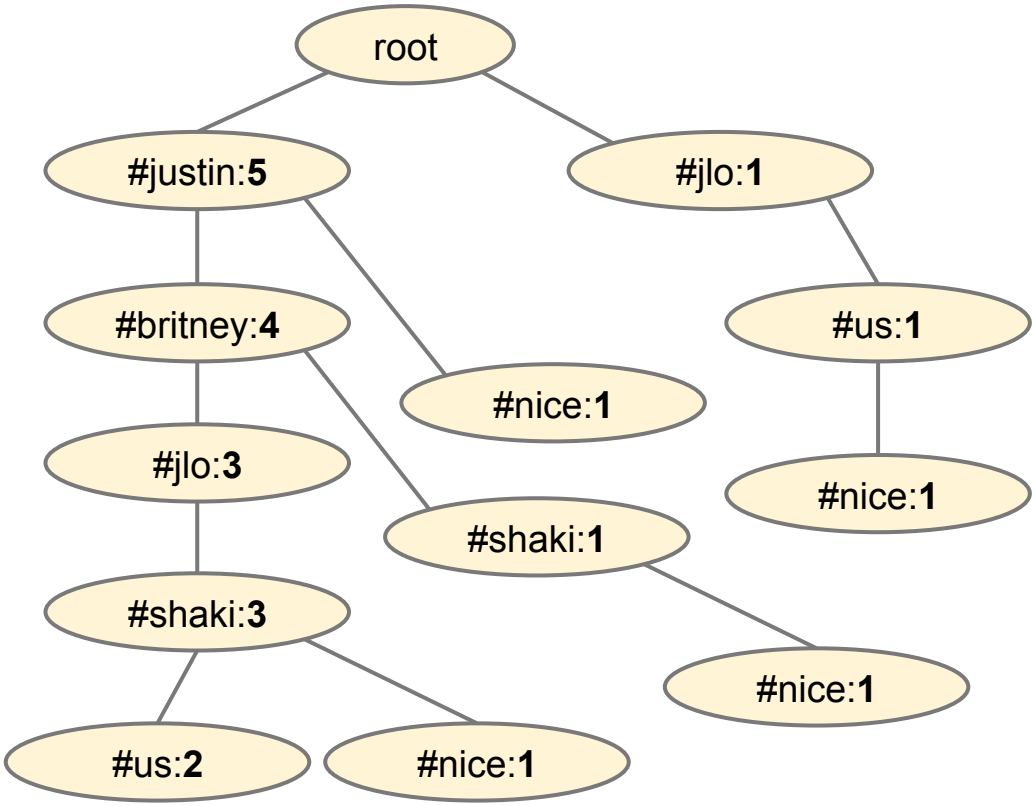
- 2 phases: tree construction & frequent itemset mining
  - Phase 1
    - 1. counting frequencies of occurring items
    - 2. filtering for frequent items & order items according to frequencies
    - Compact tree structure



1	{#justin, #britney, #jlo, #shaki, #us}
2	{#justin, #britney, #jlo, #shaki, #nice}
3	{#justin, #nice}
4	{#jlo, #us, #nice}
5	{#justin, #britney, #jlo, #shaki, #us}
6	{#justin, #britney, #shaki, #us}

FP-Growth - Phase 1

- 2 phases: tree construction & frequent itemset mining
  - Phase 1
    - 1. counting frequencies of occurring items
    - 2. filtering for frequent items & order items according to frequencies
    - Compact tree structure

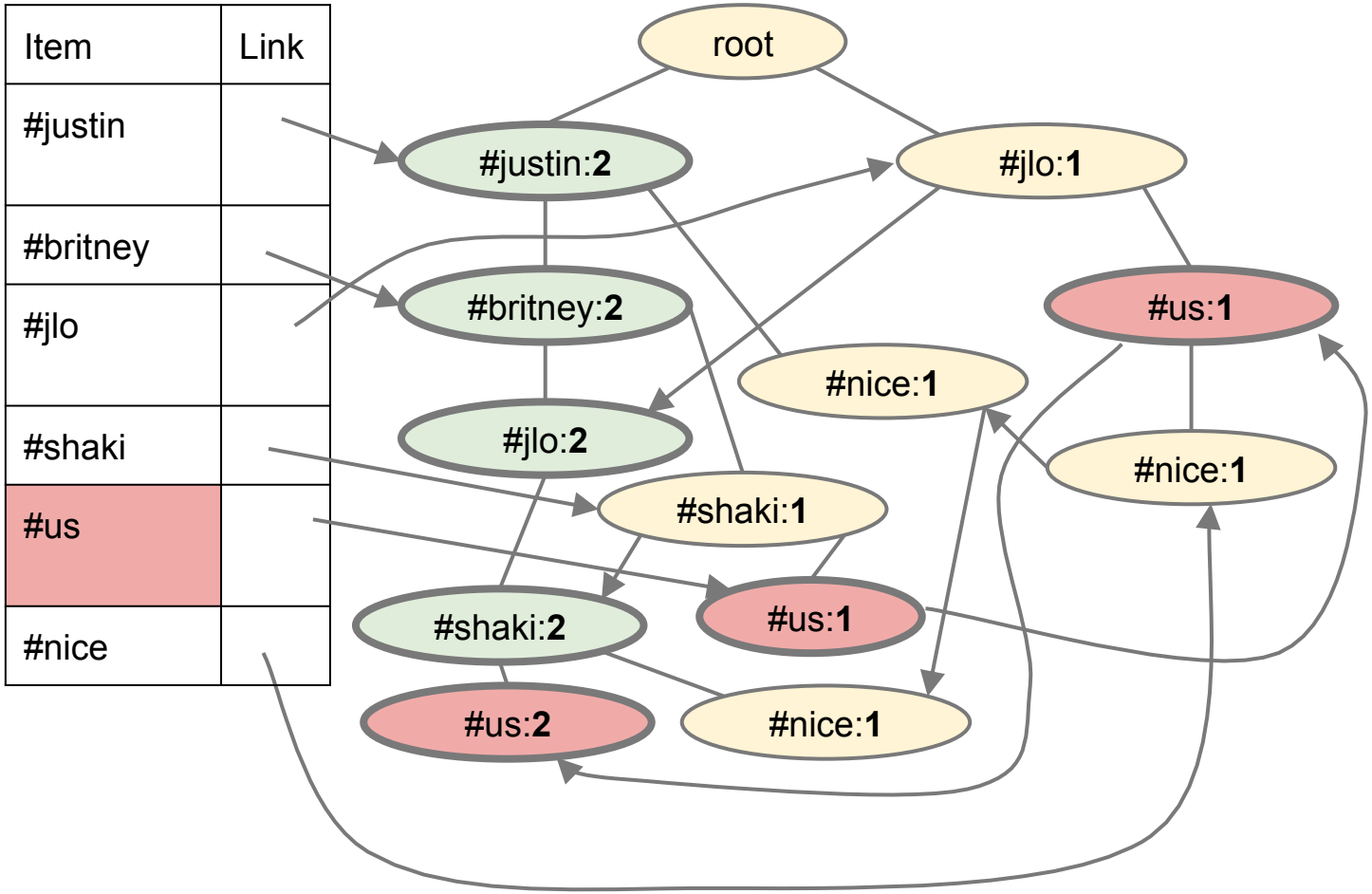


header-table

hashtag	(pointer)
#justin	
#britney	
#jlo	
#shaki	
#us	
#nice	

FP-Growth - Phase 2

- Iterate over generated header table (*index*)

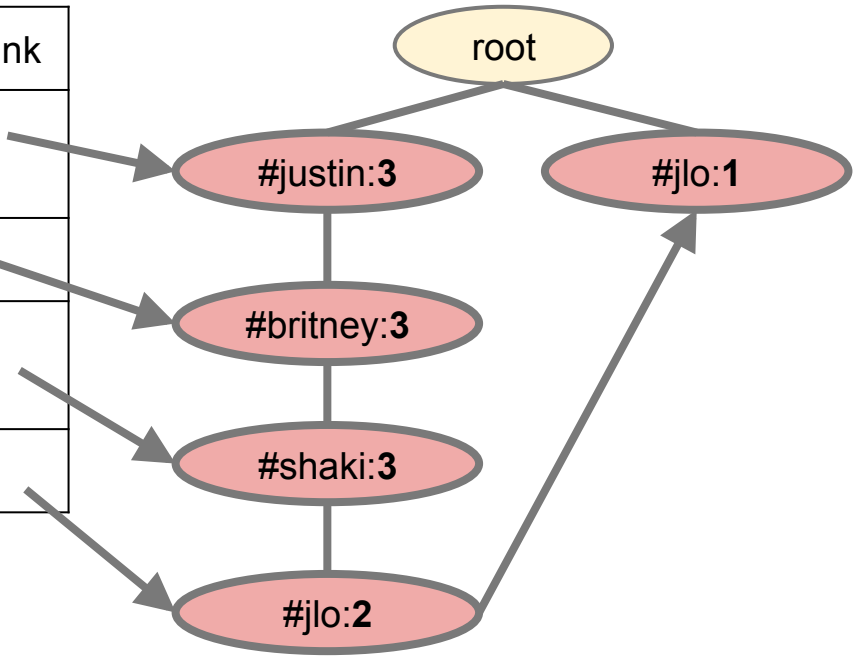


FP-Growth - Phase 2

- Iterate over generated header table (*index*)

1	{#jlo}
2	{#justin, #britney, #shaki}
3	{#justin, #britney, #shaki, #jlo}
4	{#justin, #britney, #shaki, #jlo}

Item	Link
#justin	
#britney	
#shaki	
#jlo	



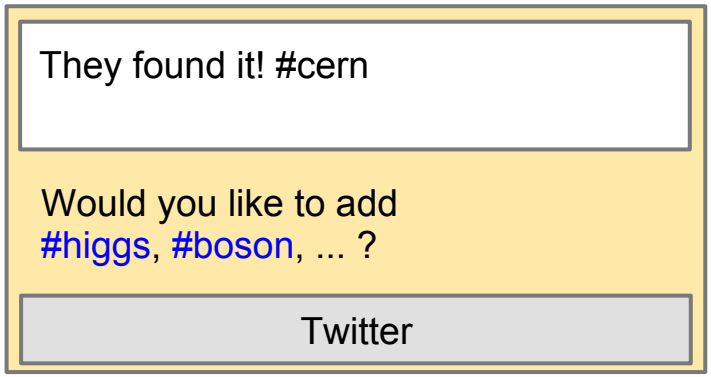
Use Case
Twitter

Idea

- Collection of Twitter tweets to find out patterns
  - Hashtag proposal at tweet input
  - Recent trends: beyond simple counting of hashtags

- June 2011:
  - 200 million tweets / day (http://blog.twitter.com/2011/06/200-million-tweets-per-day.html)

- Here:
  - Usage of the Twitter Streaming API
  - Limited by the bandwidth



Problems

- FP-Tree is not suited for adding new transactions
  - Not incremental

- Building the tree requires all transactions to be known
  - Frequency of each item has to known
  - Infrequent items are not included
  - Transactions have to be sorted by frequency of items
    => Usage of stream not possible

- When adding new transactions:
  - Infrequent items can become frequent
  - Items are likely to change their order (due to the frequency)

Extension
Incremental FP-Tree

Naive Implementation

- Main idea: add interface to add a transaction to existing tree
  - Save all transactions in tree
  - Save newly added transaction
  - Count transaction items
  - Remove infrequent items
  - Sort all transactions
  - Construct new FP-Tree

  => **Build new FP-Tree for every added transaction**

- Drawback: Tree creation is expensive
  - Takes a lot of time
  - Consumes a lot of memory

Incremental Approaches

- Specific approaches for adding new transactions to existing trees
  - Incremental approaches

- Many different approaches available
  - CanTree (Canonical Tree)
  - CATS & DB-Tree
  - PotFp-Tree
  - CP-Tree
  - ...

- Every approach has advantages, but also limitations

CP-Tree

- **C**ompact **P**attern tree: Tanbeer, Ahmed, Jeong, Lee **(2008)**

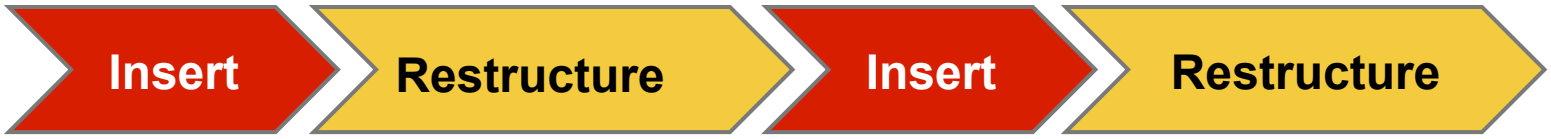
- Main ideas:
  - All items are stored in tree (not necessarily structured)
  - Infrequent items are not excluded for tree creation
  - 1 data scan is sufficient -> allows to handle a (potentially non-repeatable) stream of data instead of a fixed database

CP-Tree

- **C**ompact **P**attern tree: Tanbeer, Ahmed, Jeong, Lee **(2008)**

- Main ideas:
  - All items are stored in tree (not necessarily structured)
  - Infrequent items are not excluded for tree creation
  - 1 data scan is sufficient -> allows to handle a (potentially non-repeatable) stream of data instead of a fixed database

  - But: FP-Growth needs correctly structured (sorted) tree
  - 2 alternating phases for tree creation



  - FP-Growth needs to be slightly adapted
    - Has to consider only the frequent items

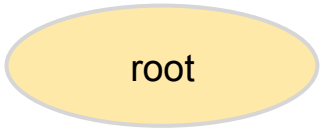
CP-Tree Example

- Example transactions
- 2 phases:
  - Insert transactions
  - Restructure tree
- Restructure after having inserted 3 transactions

1	{#britney, #pop, #justin, #nice, #jlo}
2	{#jlo, #shakira, #concert}
3	{#gaga, #justin, #jlo}
4	{#britney, #justin, #jlo}
...	...

CP-Tree - Insertion

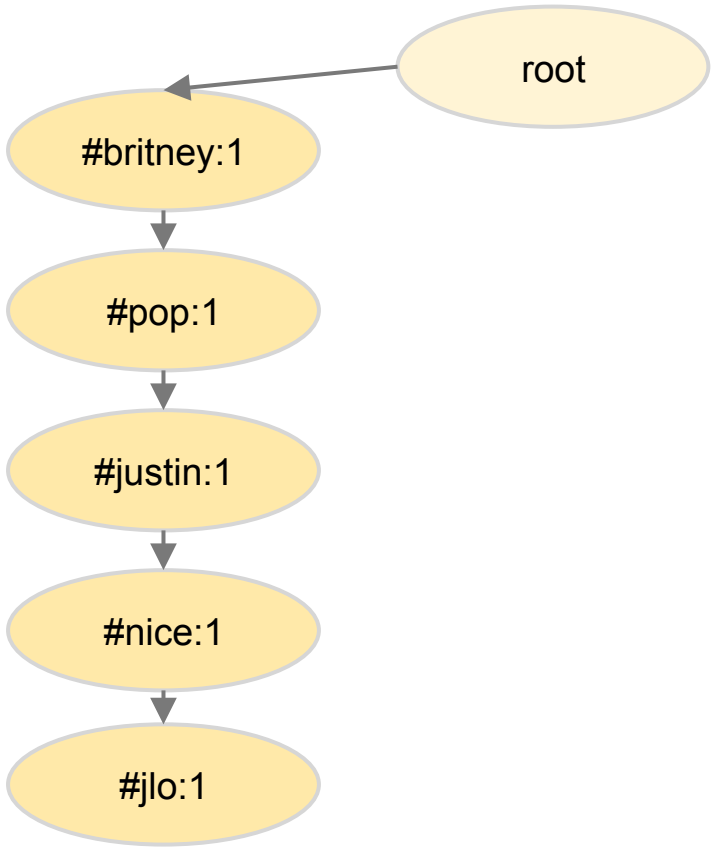
- Begin with root node



CP-Tree - Insertion

- Insert first transaction {#britney, #pop, #justin, #nice, #jlo}

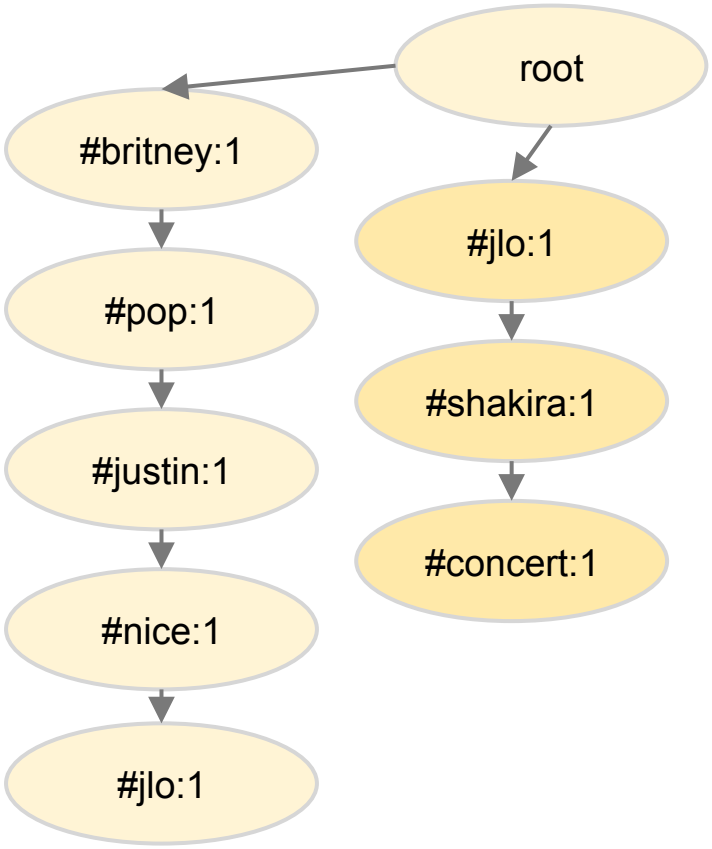
#britney	1
#pop	1
#justin	1
#nice	1
#jlo	1



CP-Tree - Insertion

- Insert second transaction {#jlo, #shakira, #concert}

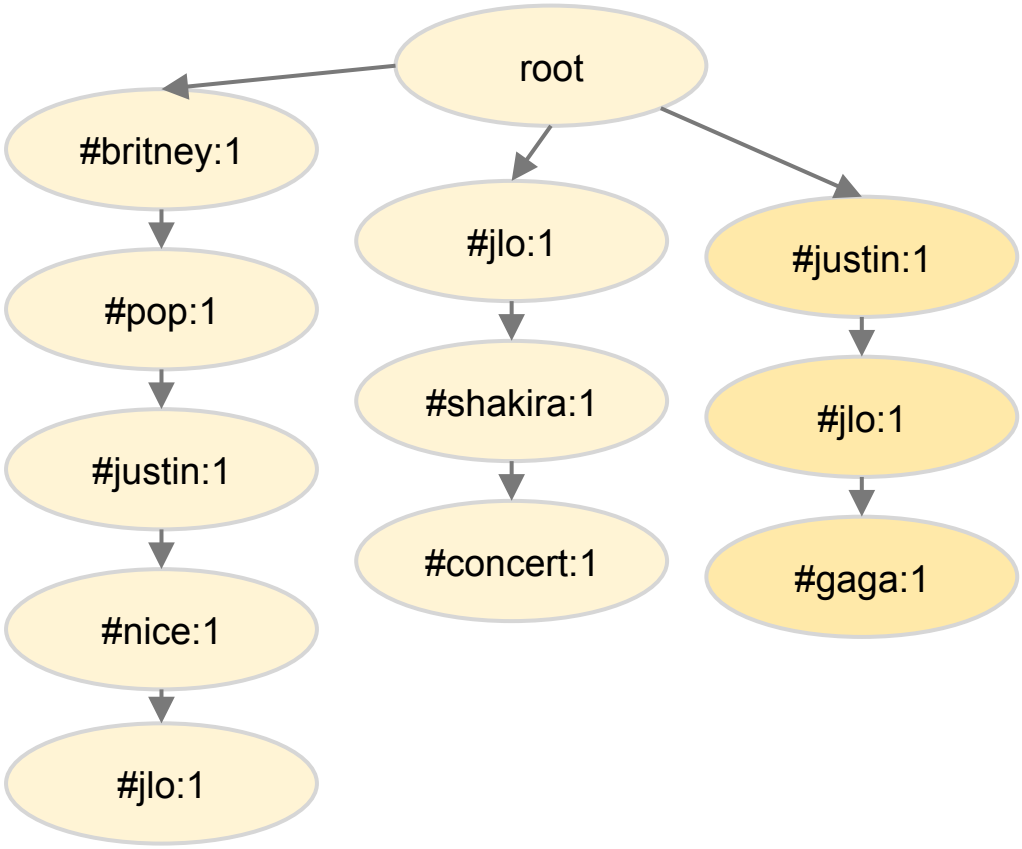
#britney	1
#pop	1
#justin	1
#nice	1
#jlo	2
#shakira	1
#concert	1



CP-Tree - Insertion

- Insert third transaction {#gaga, #justin, #jlo}

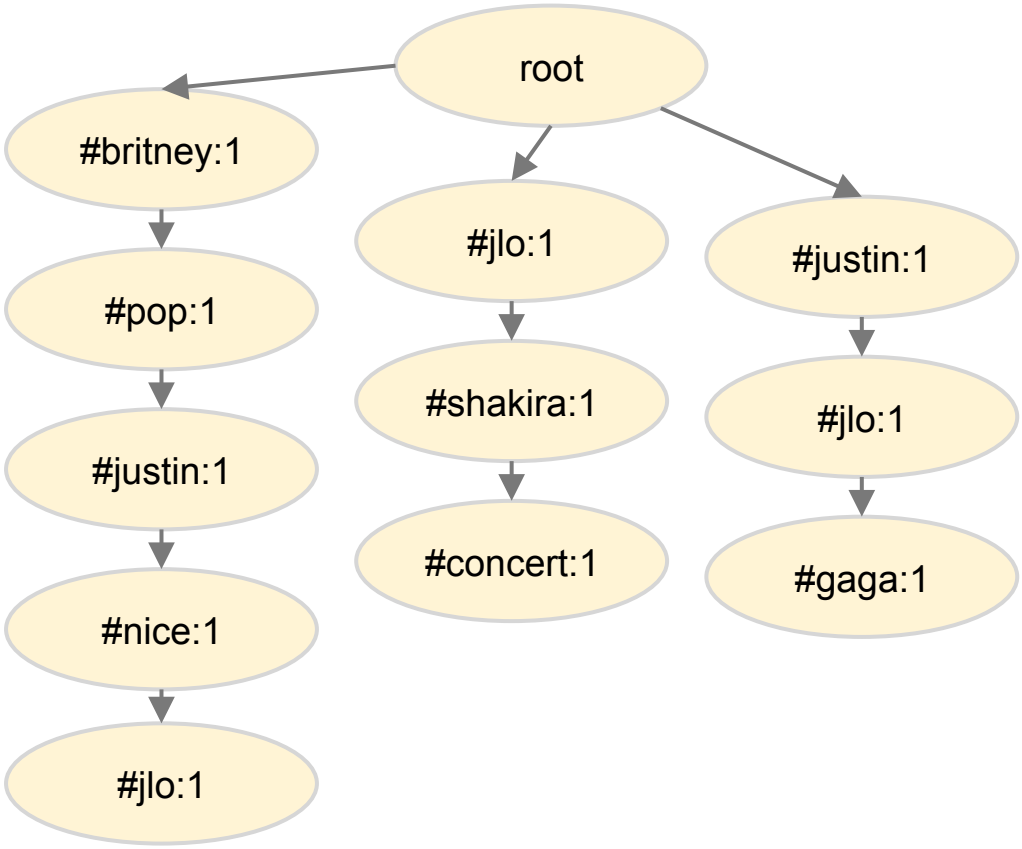
#britney	1
#pop	1
#justin	2
#nice	1
#jlo	3
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- Sort header table by frequency

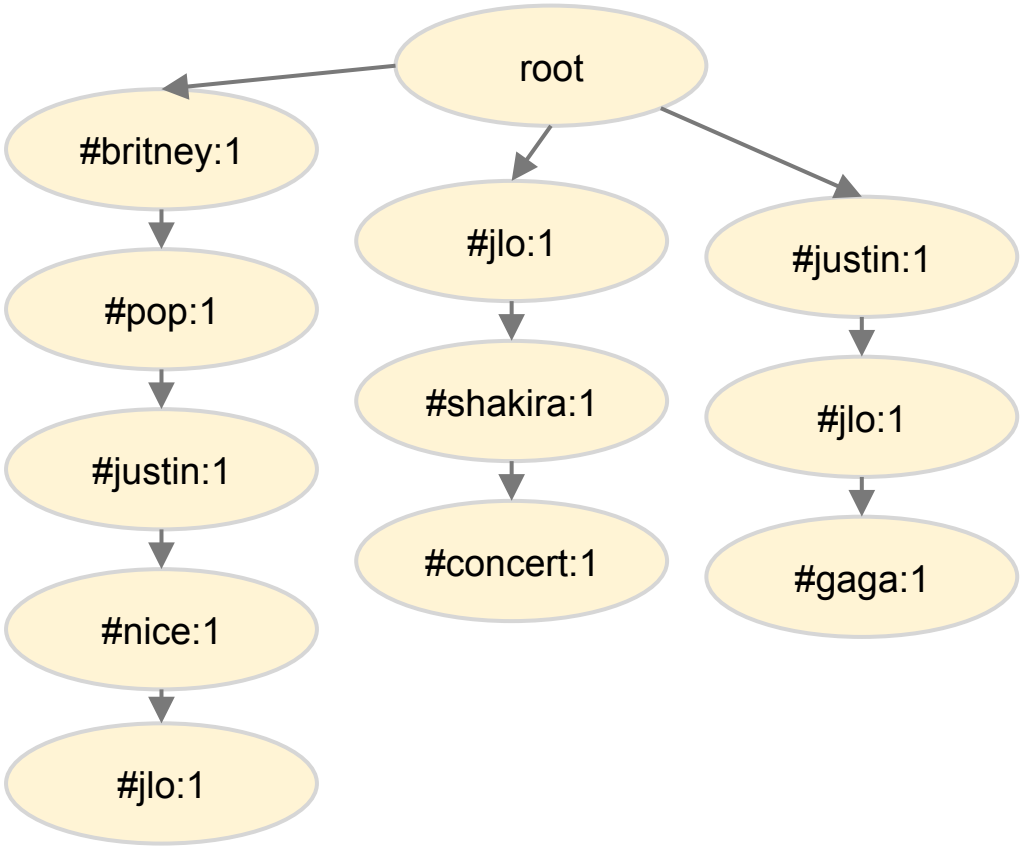
#britney	1
#pop	1
#justin	2
#nice	1
#jlo	3
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- Sort header table by frequency

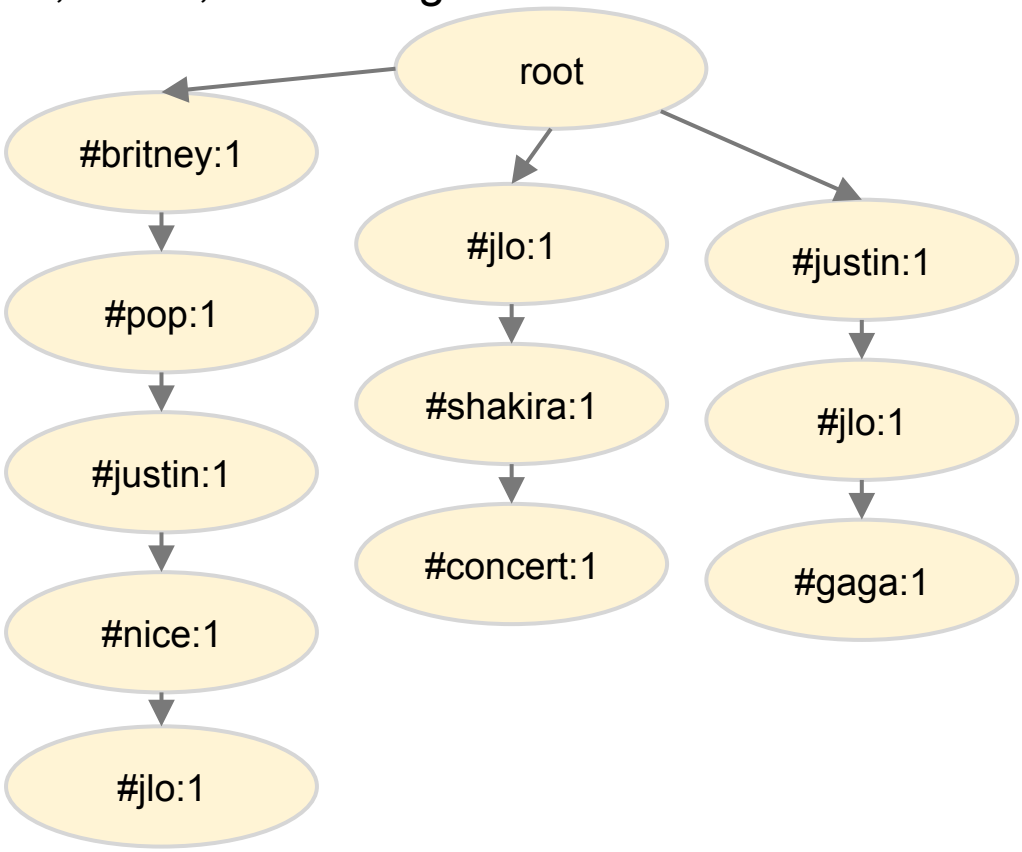
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

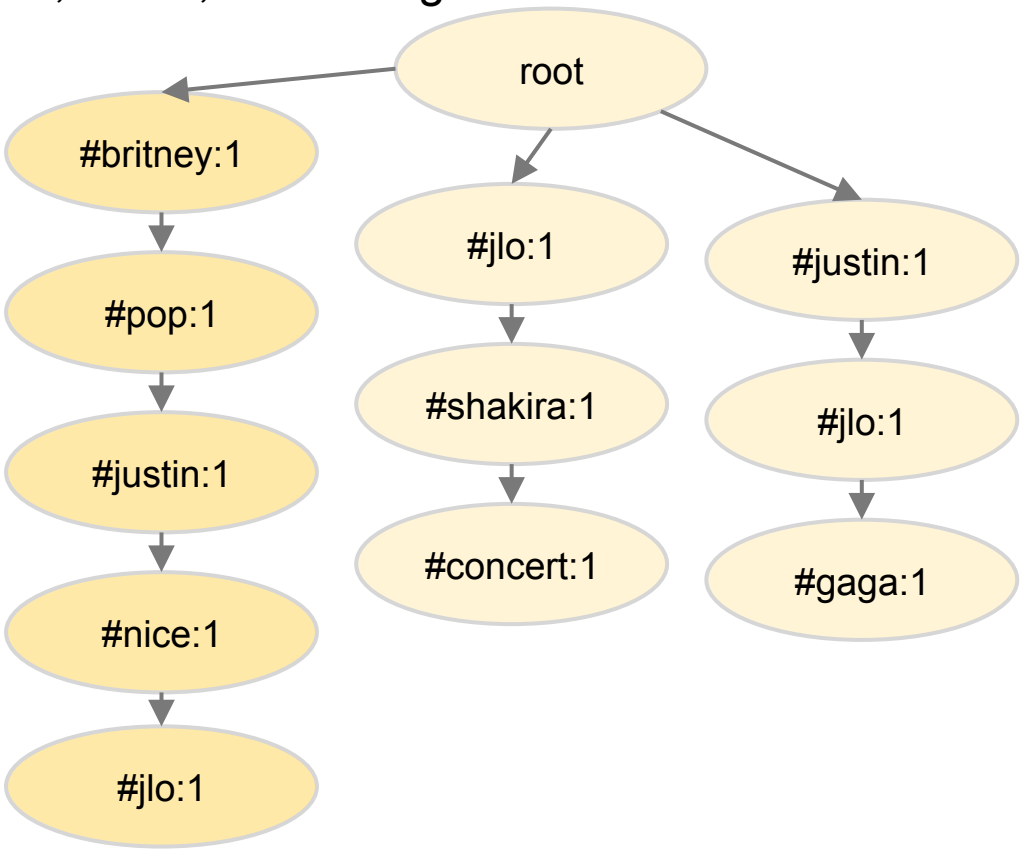
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1

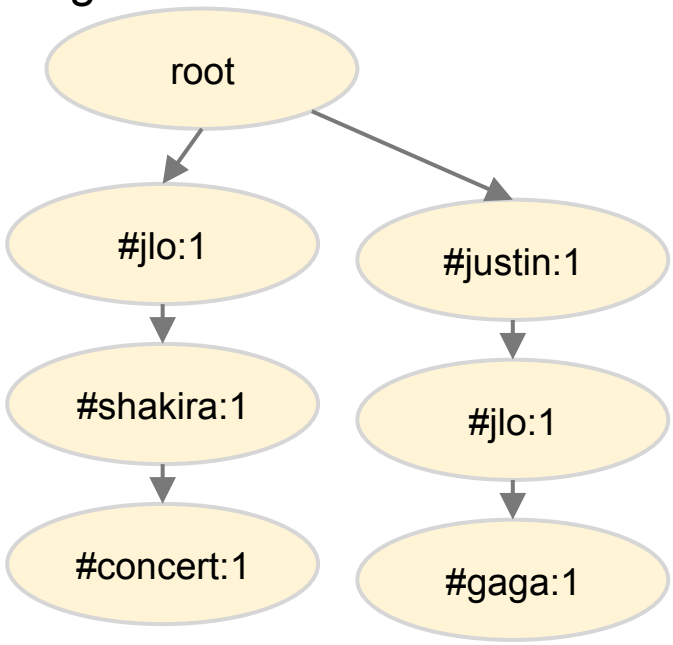


not sorted!

CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

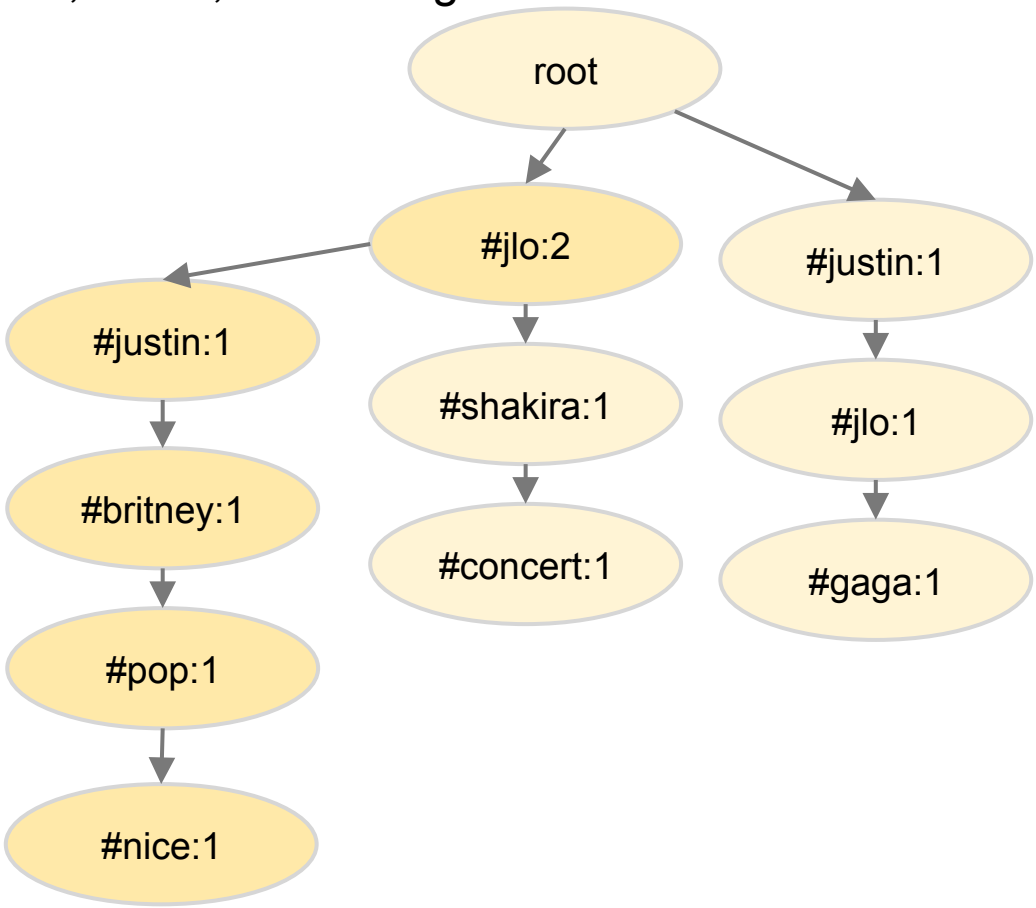
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

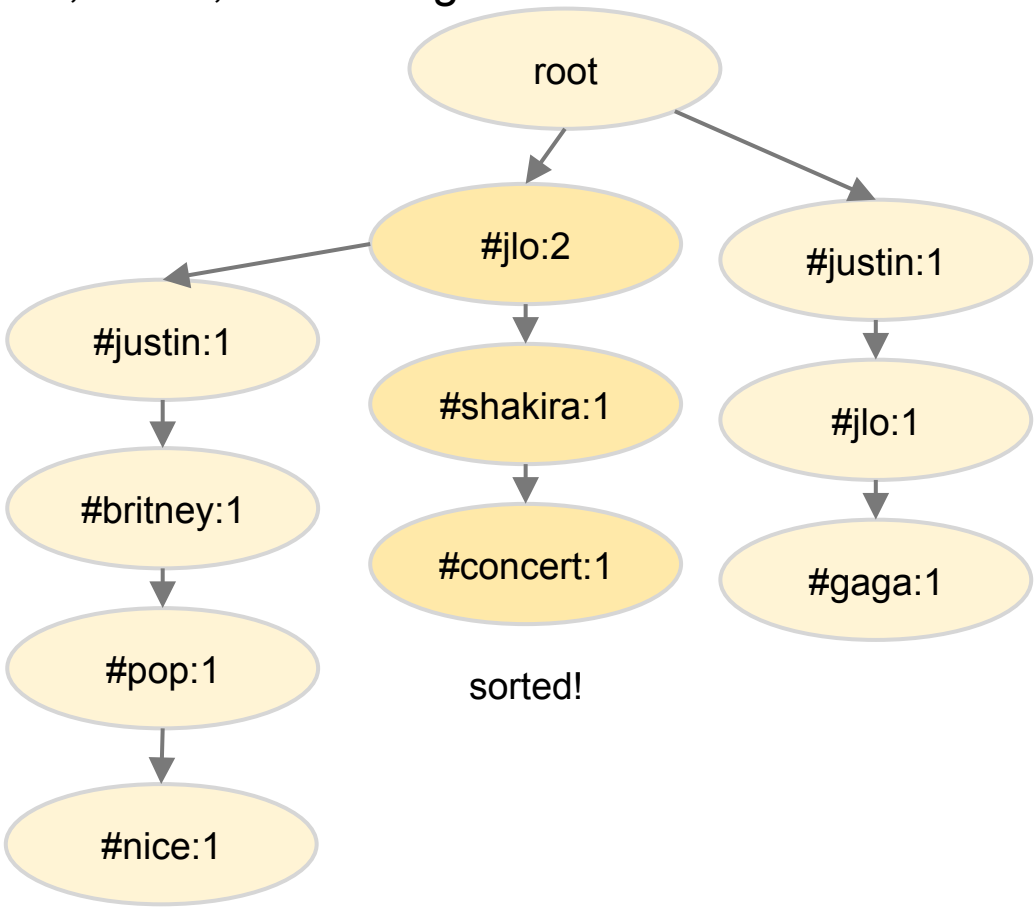
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

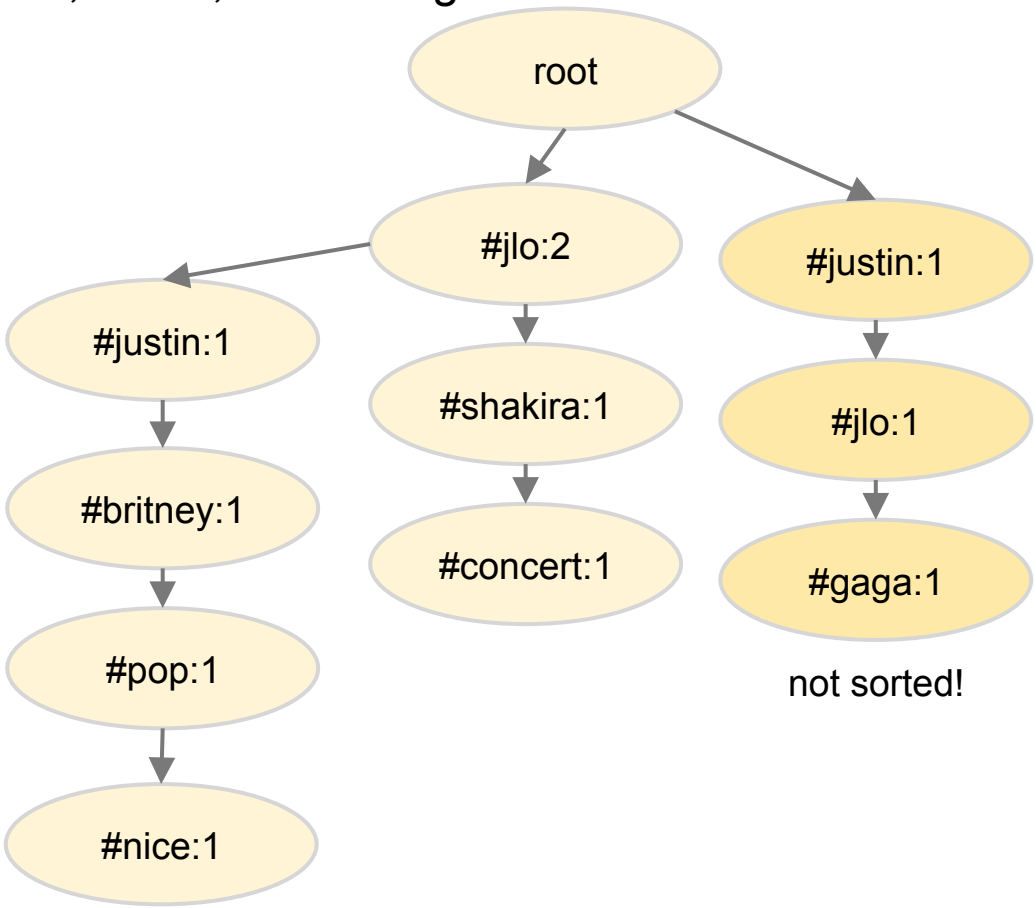
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

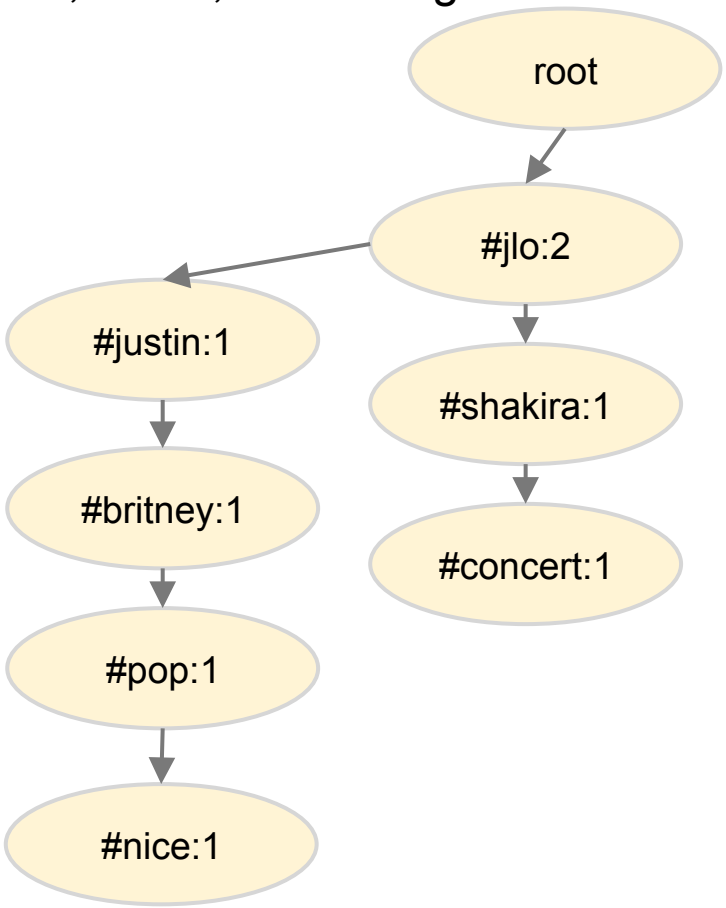
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

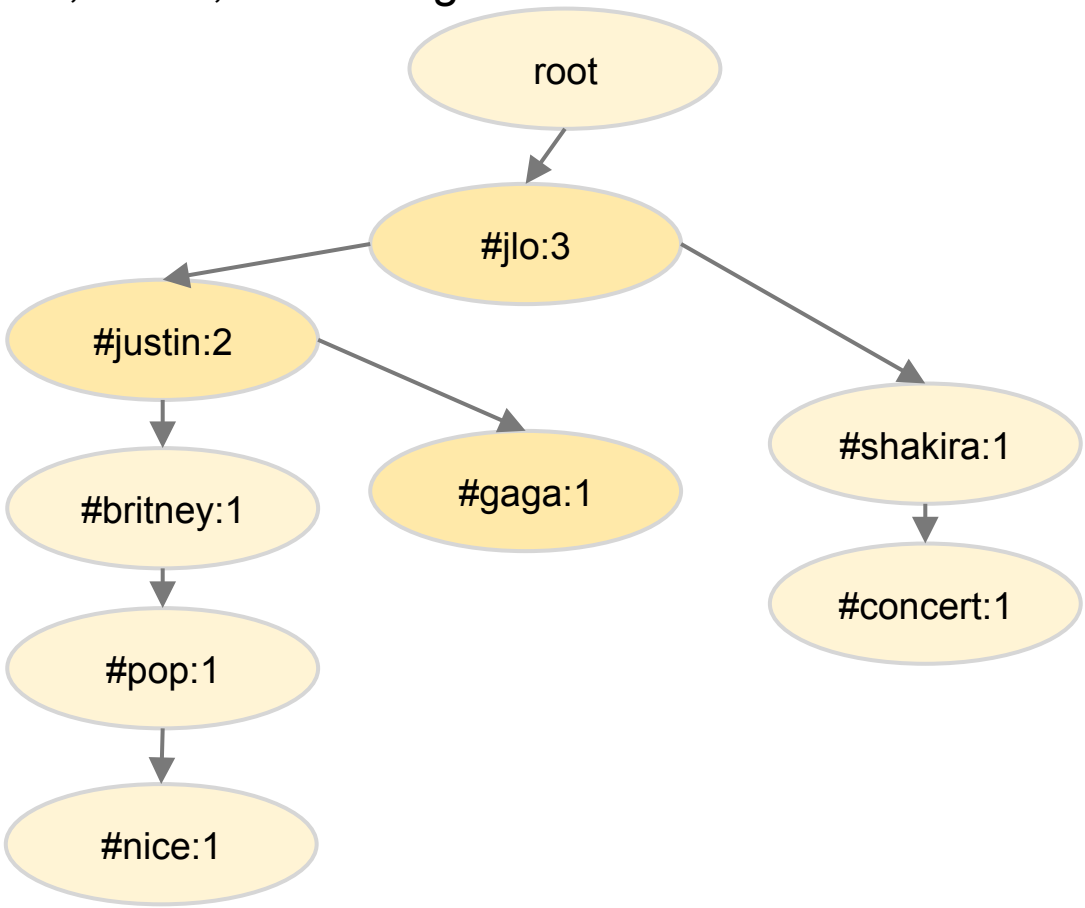
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

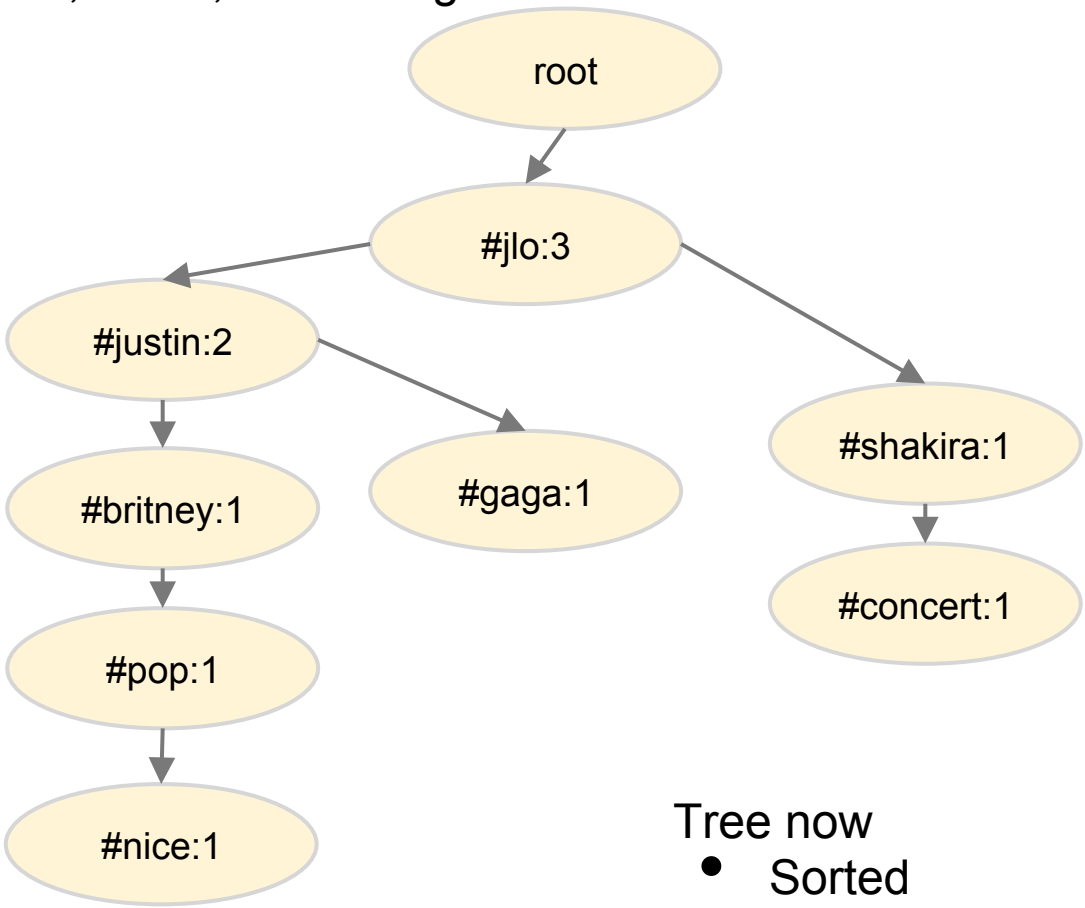
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Restructuring

- For each branch: check if it is sorted
  - if not: remove it, sort it, insert it again

#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1

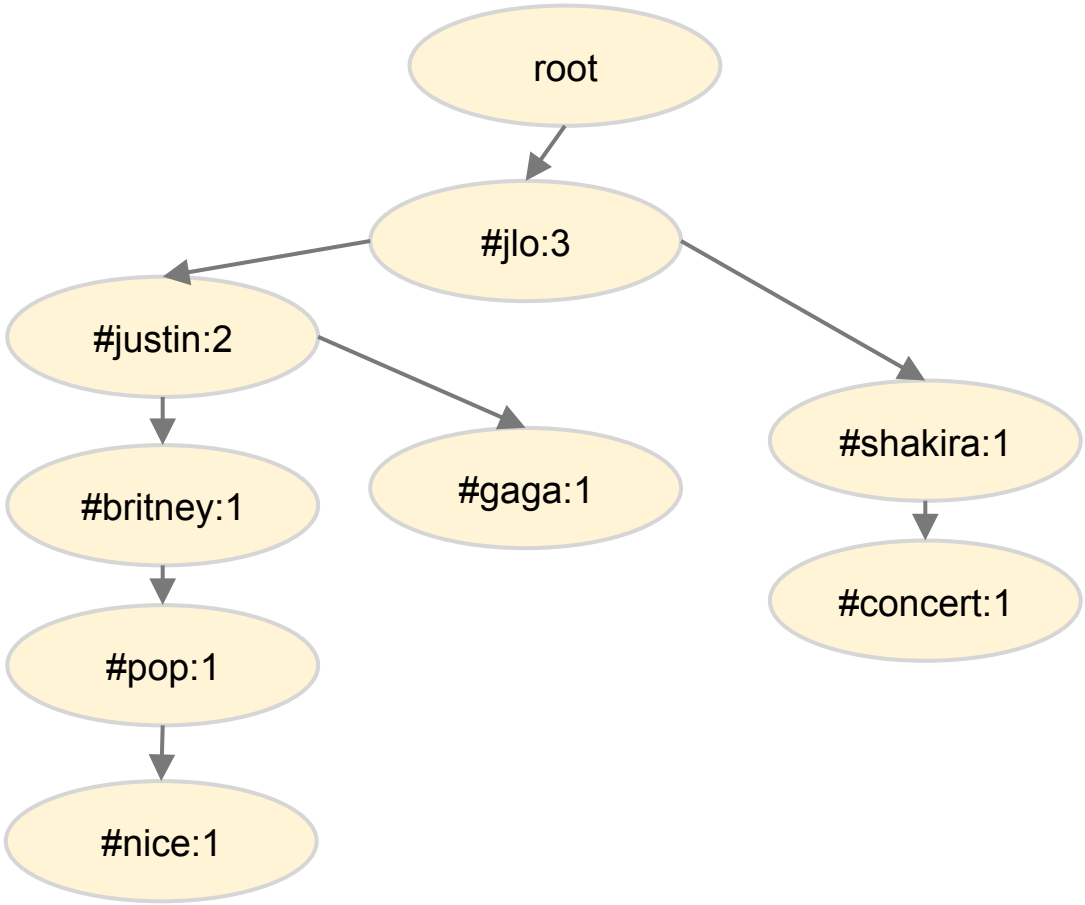


Tree now
- Sorted
- Compact

CP-Tree - Insertion

- Insert fourth transaction {#britney, #justin, #jlo}

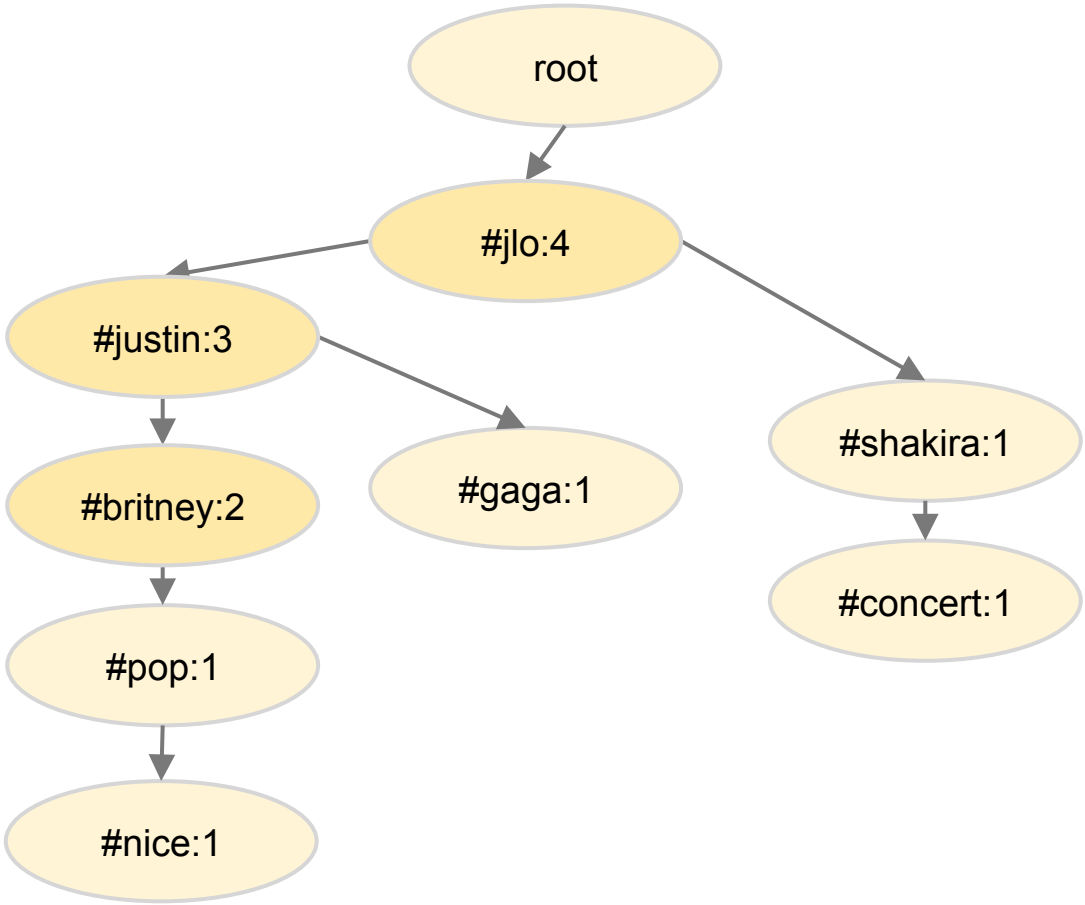
#jlo	3
#justin	2
#britney	1
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1



CP-Tree - Insertion

- Insert fourth transaction {#britney, #justin, #jlo}

#jlo	4
#justin	3
#britney	2
#pop	1
#nice	1
#shakira	1
#concert	1
#gaga	1

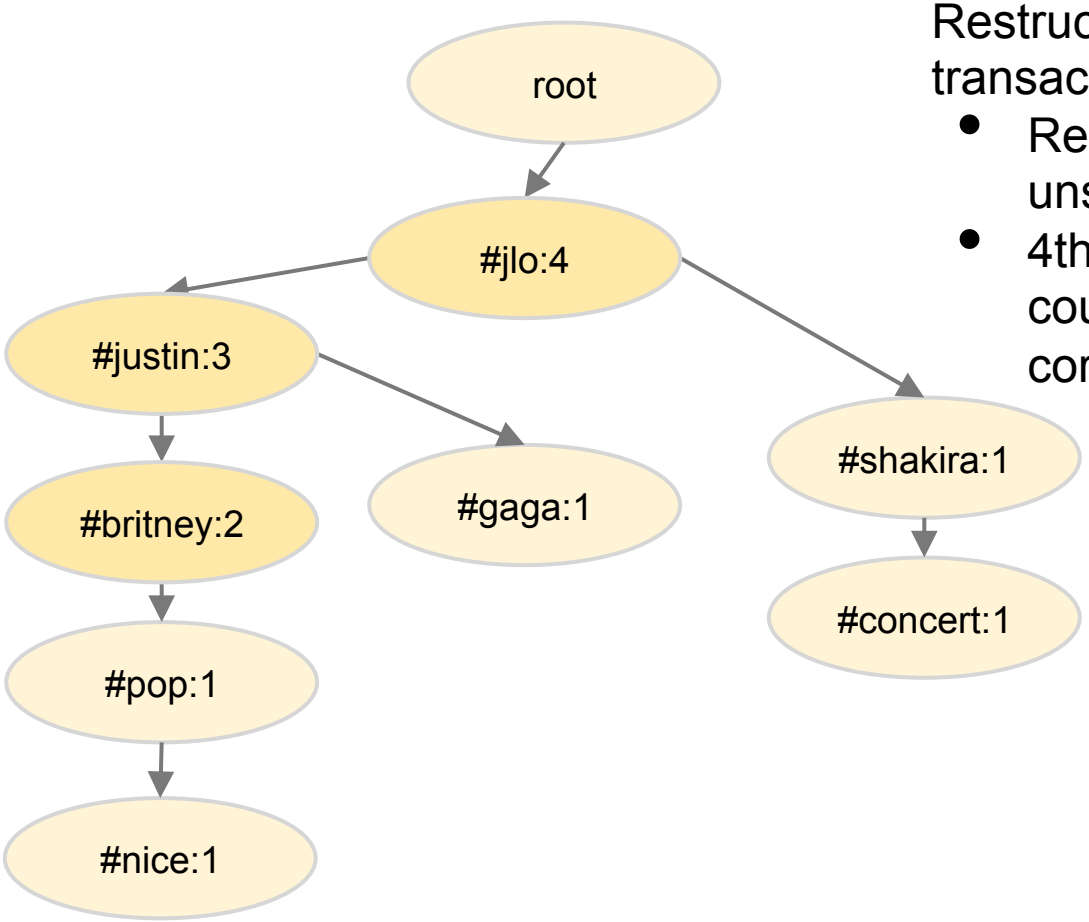


CP-Tree

- What if we didn't restructure after 3 transactions?
  - Tree less compact
  - Next restructurings would be more expensive

CP-Tree

- What if we didn't restructure after 3 transactions?
  - Tree less compact
  - Next restructurings would be more expensive

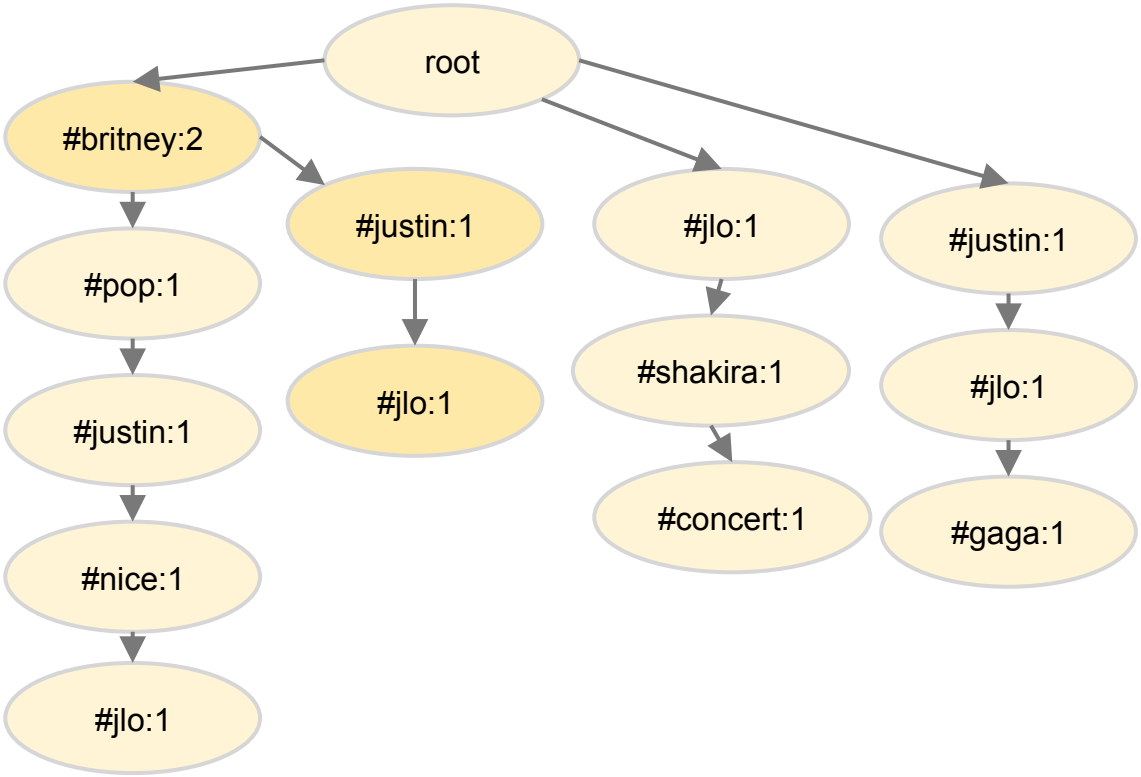


Restructuring after 3 transactions:

- Re-insertion of 2 unsorted paths
- 4th inserted path could make use of correct order

CP-Tree

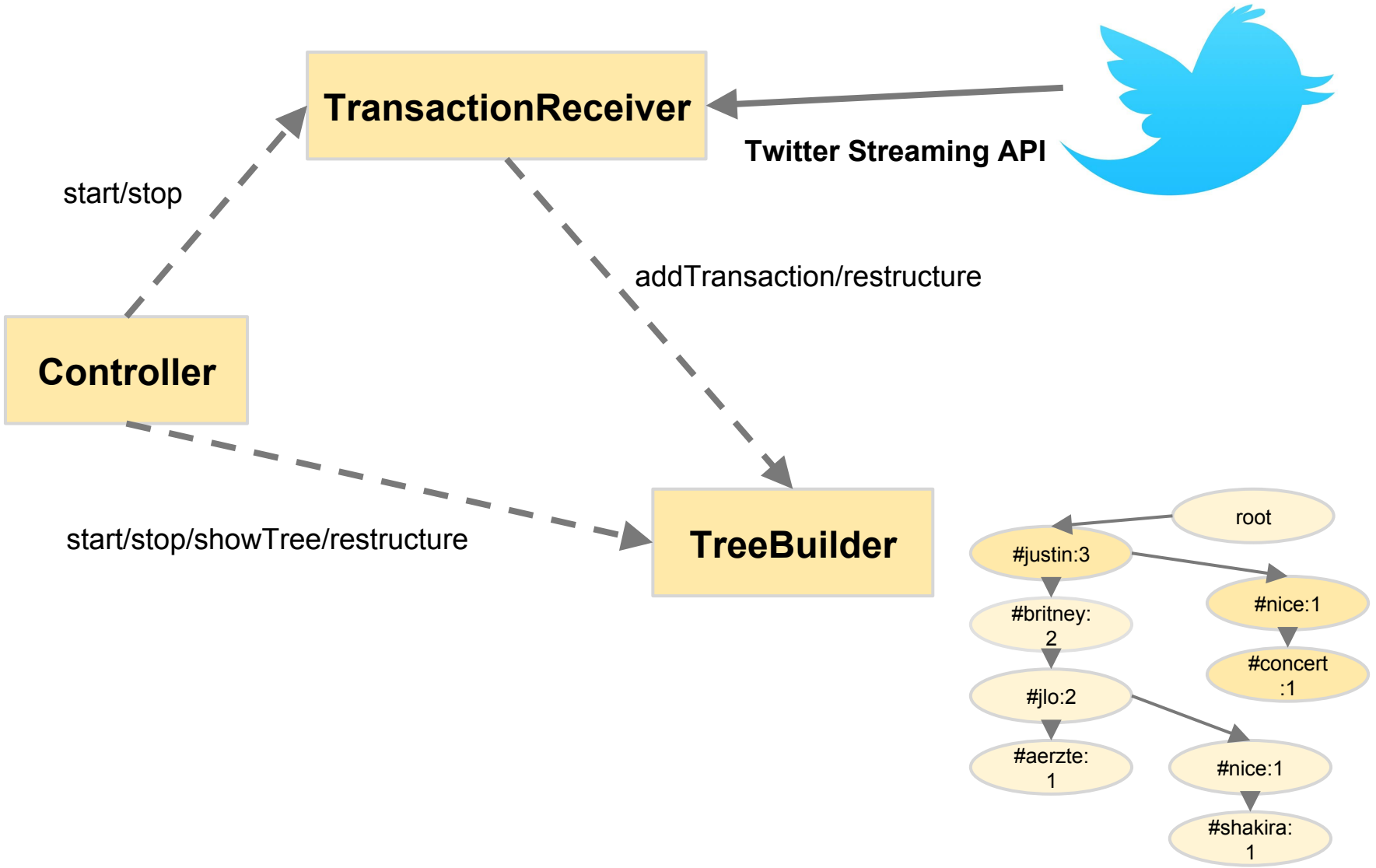
- What if we didn't restructure after 3 transactions?
  - Tree less compact
  - Next restructurings would be more expensive



Restructuring after 4 transactions:

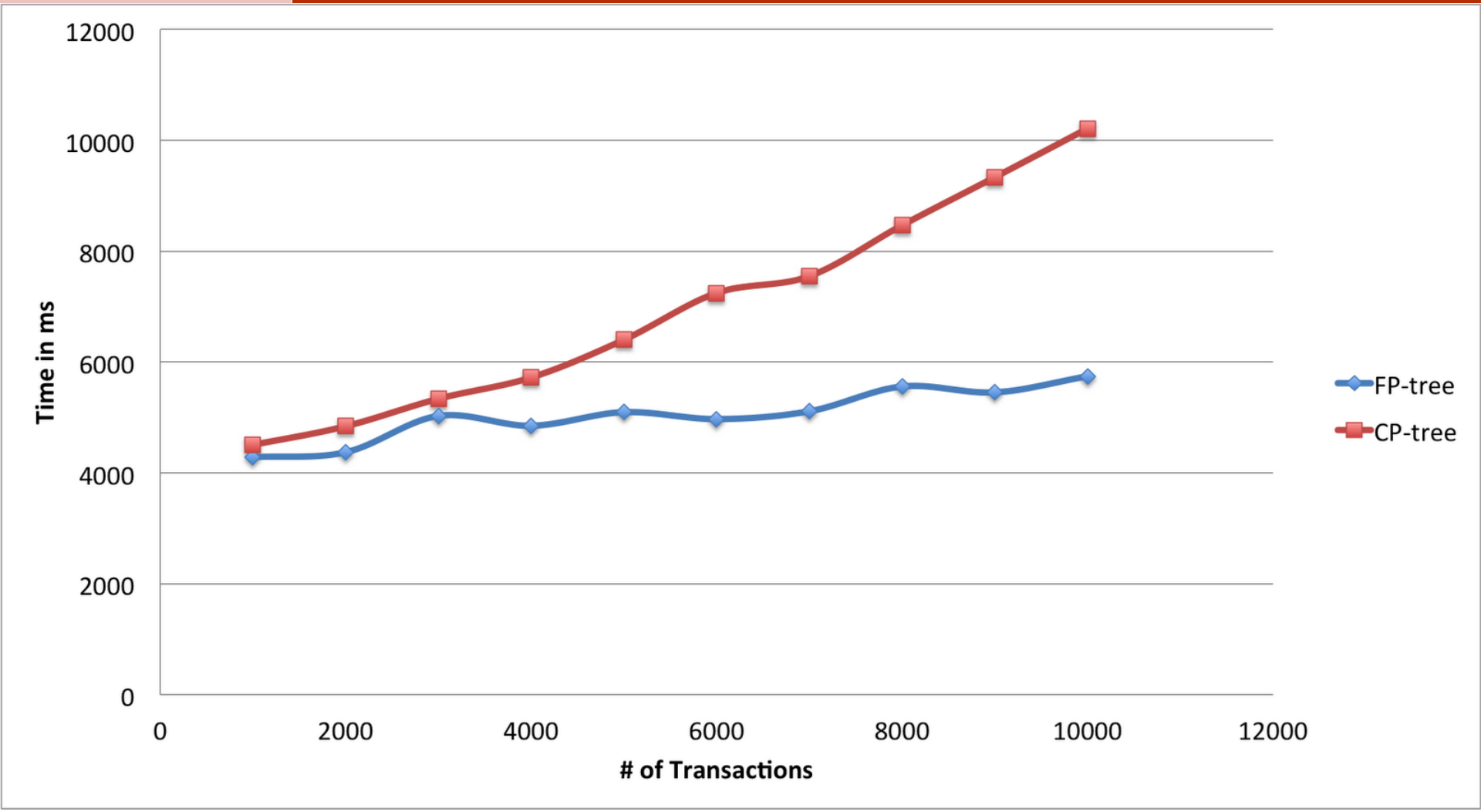
- Re-insertion of 3 unsorted paths
- 4th would not make use of the correct order -> would have to be sorted

CP-Tree - Implementation

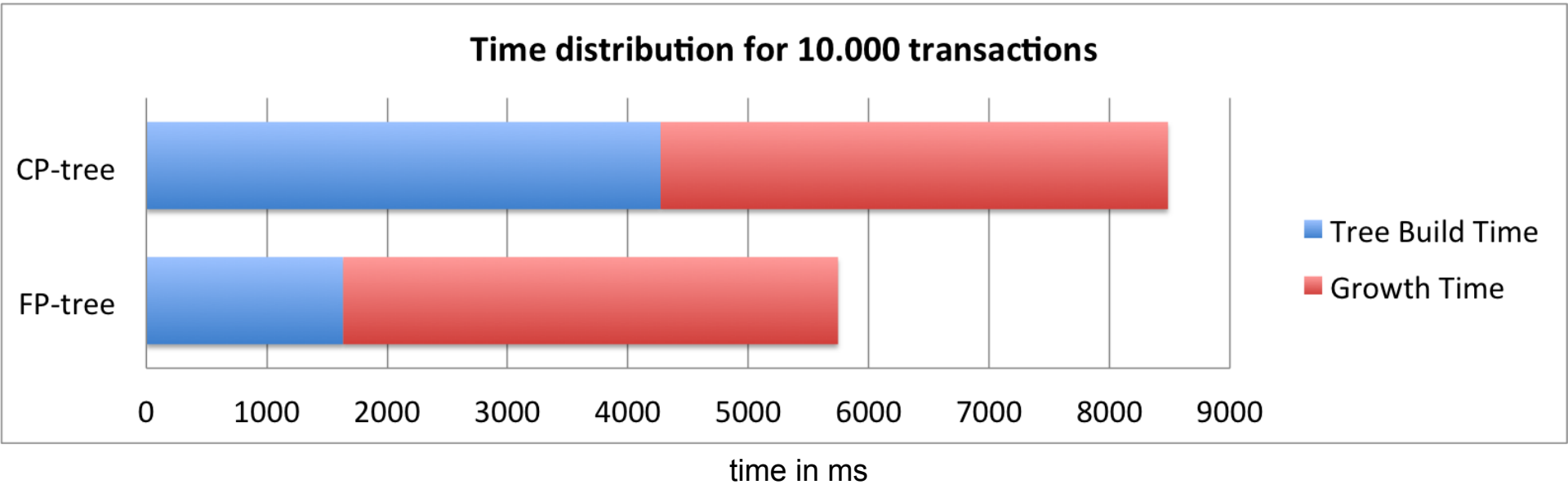


Experiments & Results

Comparison - FP-Tree & CP-Tree

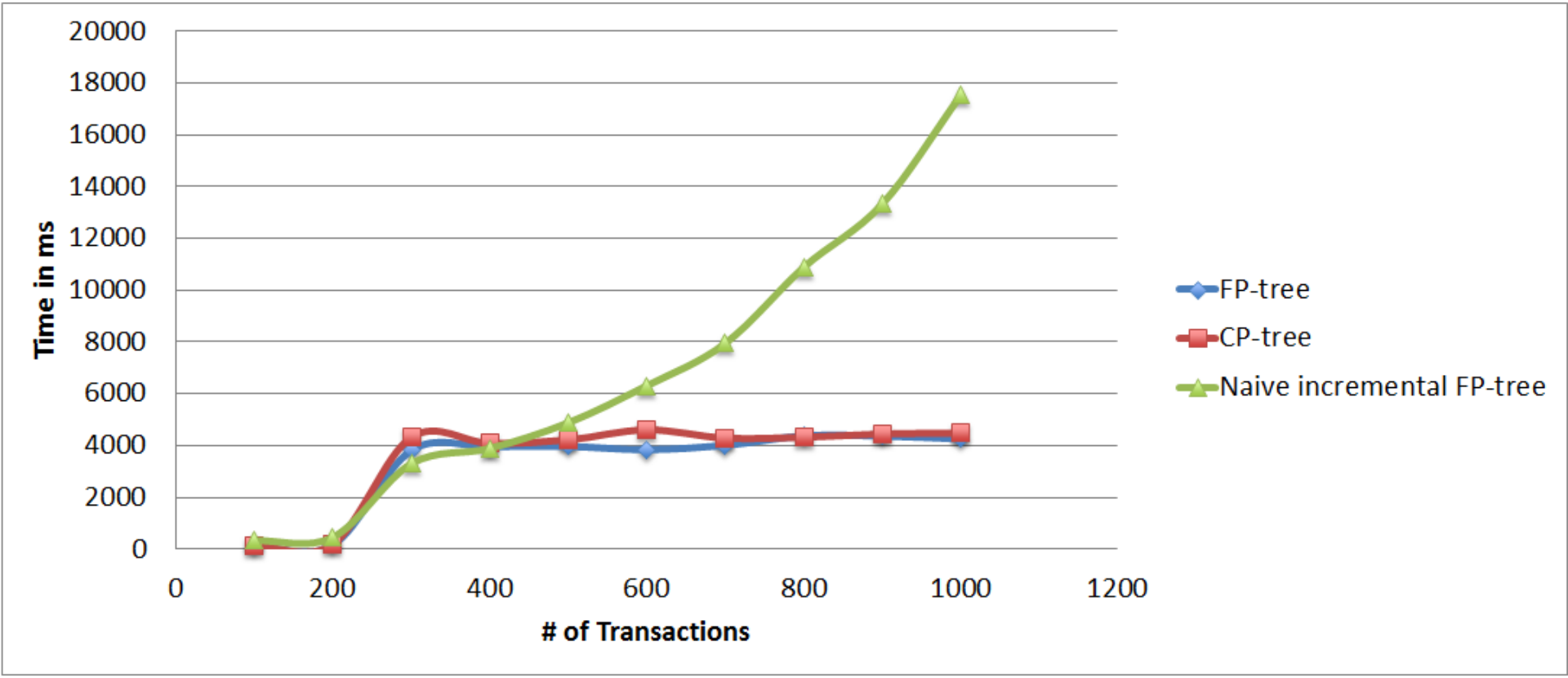


Comparison - FP-Tree & CP-Tree



- CP-Tree has a more expensive tree creation phase

Comparison - FP-Tree & CP-Tree & Naive



Patterns

Filtered Twitter data:

- ceramics, pottery -> porcelain
  - support: 0.00292, confidence: 0.99824
- supportgoodmusic, punchlines, teamtracmurda -> listen2me
  - support: 0.00265, confidence: 0.96449
- chai -> tea
  - support: 0.00431,confidence: 0.99762
- tea -> chai
  - support: 0.00431,confidence: 0.91449

Unfiltered Twitter data:

- Androidgames -> Android
  - support: 0.00390, confidence: 0.99597
- followback, followmejp, sougofollow -> followme
  - support: 0.00364, confidence: 0.97920
- xxx -> porn
  - support: 0.01357, confidence: 0.92230
- porn -> xxx
  - support: 0.01357, confidence: 0.74559

Patterns

Netflix movie data:

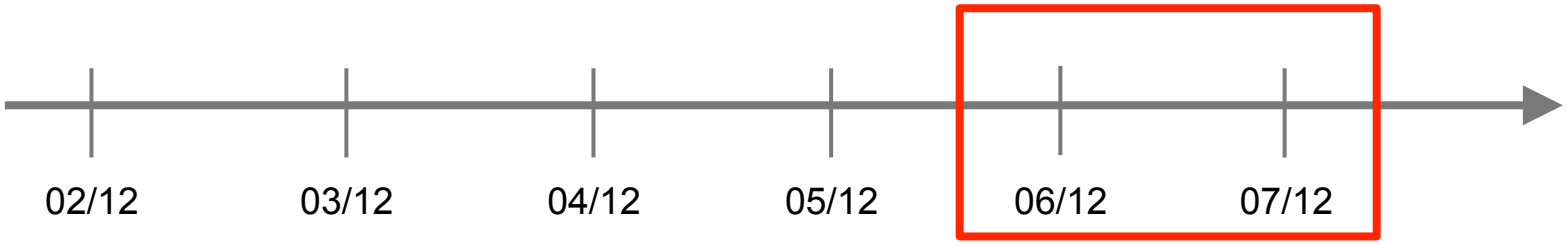
- LotR: The Fellowship of the Ring (Extended Edition) -> LotR: The Fellowship of the Ring
  - support: 0.1052, confidence: 0.84238
- LotR: The Fellowship of the Ring -> LotR: The Fellowship of the Ring (Extended Edition)
  - support: 0.1052, confidence: 0.51304
- Monsters, Shrek -> Finding Nemo
  - support: 0.0561 , confidence: 0.76849
- The Usual Suspects -> The Shawshenk Redemption (Die Verurteilten)
  - support: 0.0732 , confidence: 0.56686
- The Shawshenk Redemption -> The Usual Suspects
  - support: 0.0732 , confidence: 0.35495

CP-Tree - Extension: Sliding Time Window

- Problem:
  - (Not ending) data stream leads to continuously growing tree
  - Results also in longer runtime for restructuring

- Initial idea:
  - Collection of **newest Twitter tweets** to find out interesting and **up-to-date patterns**
  - Old transactions (tweets) can be deleted

- Assume sliding time window



Comparison: Without & With Sliding Window

- 3 alternatives, each ran > 30 minutes
- Checking # of frequent patterns (MFT = 5) every 10 minutes
- Checking top 3 frequent patterns

	No Removal	Removal after 10 minutes	Removal after 5 minutes
# patterns after 10 min	10	25	2
top 3 patterns after 10 min	TeamFollowBack (7) SOUGOFOLLOW (6) TFBJP (6)	TeamFollowBack (15) followme(11) aggr, okaeri (8)	TeamFollowBack (7) teamfollowback (5)
# patterns after 20 min	67	20	5
top 3 patterns after 20 min	TeamFollowBack (16) TFBJP (10) followme (10)	TFB (10) TeamFollowBack (9) followme (8)	TeamFollowBack (9) followme (8) xm, 4thofJuly, sirius (6)
# patterns after 30 min	348	19	2
top 3 patterns after 30 min	TeamFollowBack (30) autofollowback (21) TFBJP (20)	TeamFollowBack (16) TFB (8) aggr (7)	TeamFollowBack (7) MUSTFOLLOW (5)

Summary

	FP-Tree	CP-Tree
Phase 1 (building tree)	count items; filter, sort and insert items into tree	insert items into tree; restructure tree after a certain limit
Required data scans	2 (one for counting, one for sorting and inserting)	1 (one for inserting)
Requirements on data	need to be permanently stored in database (since 2 data scans are required)	can be streamed without storing in database
Tree elements	all frequent items at a certain moment	all items at a certain moment (or time window)
Tree size	fixed	changes over time (increasing or relatively stable w.r.t. a potential sliding time window)
Phase 2 (pattern mining)	traditional FP-Growth with all items (all items are frequent)	traditional FP-Growth only with frequent items
Suitability for incremental data	traditionally no approach available, naive approach (re-building tree)	suitable; intended to be used in an incremental way

References

- Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (SIGMOD '00). ACM, New York, NY, USA, 1-12.
- Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. 2008. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems* (RecSys '08). ACM, New York, NY, USA, 107-114.
- Xiu-Li Ma, Yun-Hai Tong, Shi-Wei Tang, and Dong-Qing Yang. 2004. Efficient incremental maintenance of frequent patterns with FP-tree. *J. Comput. Sci. Technol.* 19, 6 (November 2004), 876-884.
- Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. 2008. CP-Tree: A Tree Structure for Single-Pass Frequent Pattern Mining. In *Advances in Knowledge Discovery and Data Mining.* Springer, 1022--1027

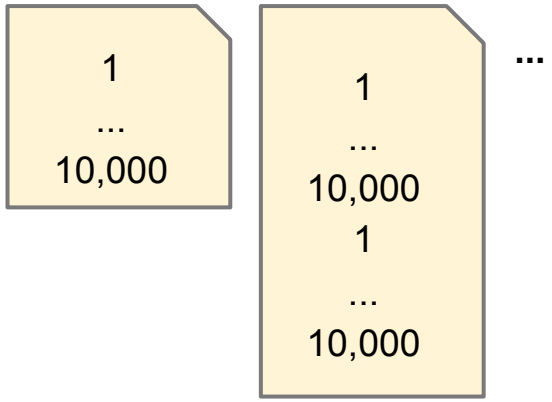
- Datasets:
  - Filtered Twitter data: Maximilian Jenders
  - Unfiltered Twitter data: Matthias Kohnen
  - Netflix data: Felix Leupold & Stefan George

Additional Slides

Comparison: Twitter vs. Netflix

- Comparison done with "traditional" FP-Growth
- Very different data characteristics
  - Twitter: short transactions, many different items (hashtags)
    => few frequent itemsets
  - Netflix: long transactions, limited set of items (movies)
    => lots of frequent itemsets

- Experiment setup
  - 3 datasets
    - 10,000 random Twitter transactions (~2 items / transactions)
    - 10,000 random Twitter transactions (~5 items / transactions)
    - 10,000 random Netflix transactions (~53 items / transactions)
    - Replicated it up to 50,000 transactions
      - Results in stable item / transaction ratio



Comparison: Twitter vs. Netflix

Time in ms	# of transactions (replicated)				
	10000	**20000**	**30000**	**40000**	**50000**
Twitter (normal)	461	546	643	658	709
Twitter (large)	371	501	620	796	920
Netflix	5224	46232	161119	343488	1095428

# frequent itemsets (MFT = 500)	# of transactions (replicated)				
	10000	**20000**	**30000**	**40000**	**50000**
Twitter (normal)	4	5	7	9	12
Twitter (large)	0	1	2	64	84
Netflix	816	19466	121406	476381	1412214

Comparison: Twitter vs. Netflix

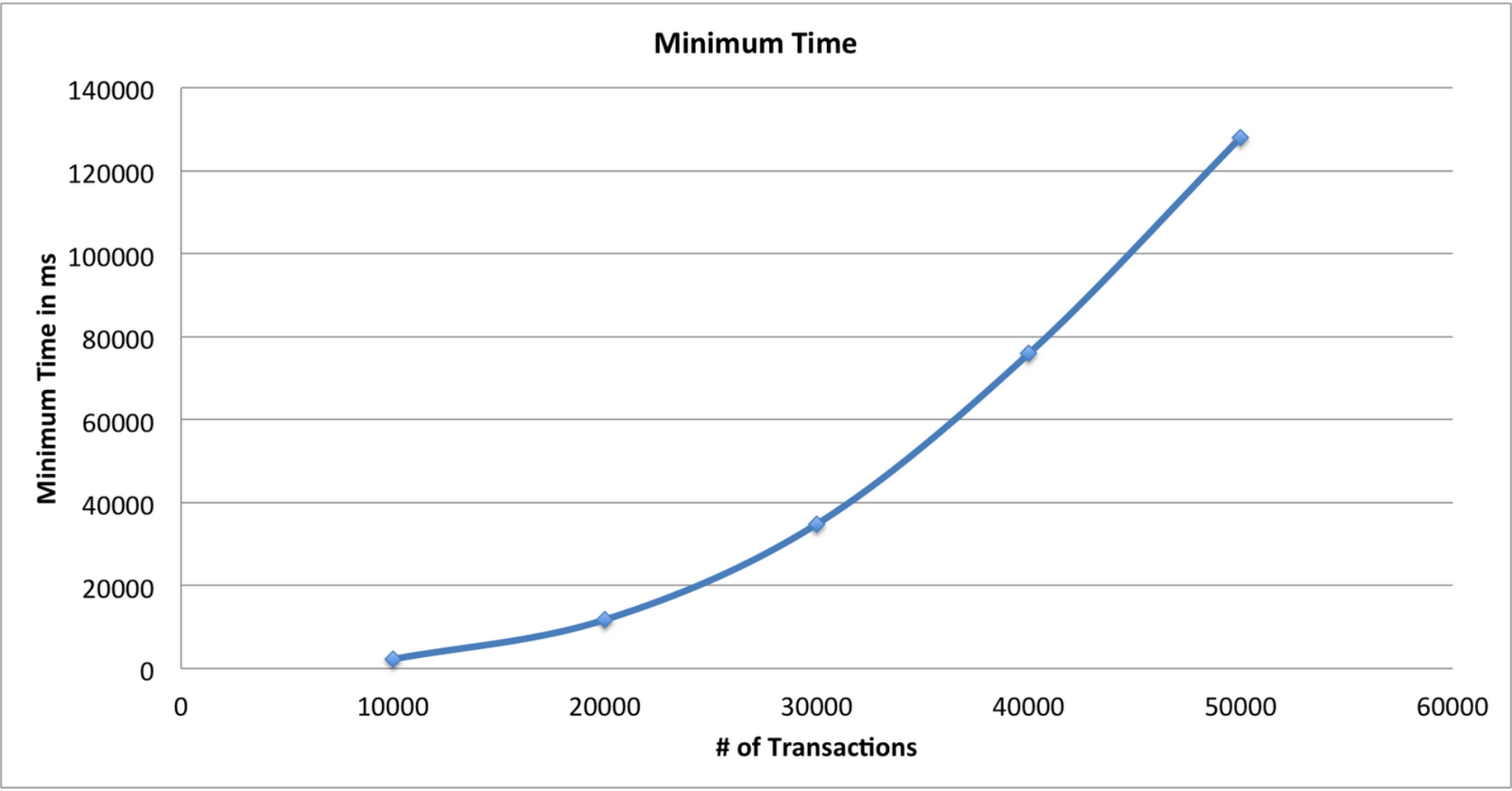
Time in ms	# of transactions (replicated)				
	10000	**20000**	**30000**	**40000**	**50000**
Twitter (normal)	461	546	643	658	709
Twitter (large)					0
Netflix					95428

# freque itemset (MFT = 5					**000**
Twitter (normal)					
Twitter (large)	0	1	2	64	84
Netflix	816	19466	121406	476381	1412214

- Netflix transactions need much more time
  - Limited set of movie titles
    => More frequent items
  - Longer transactions
    => Larger tree (# depth of tree)

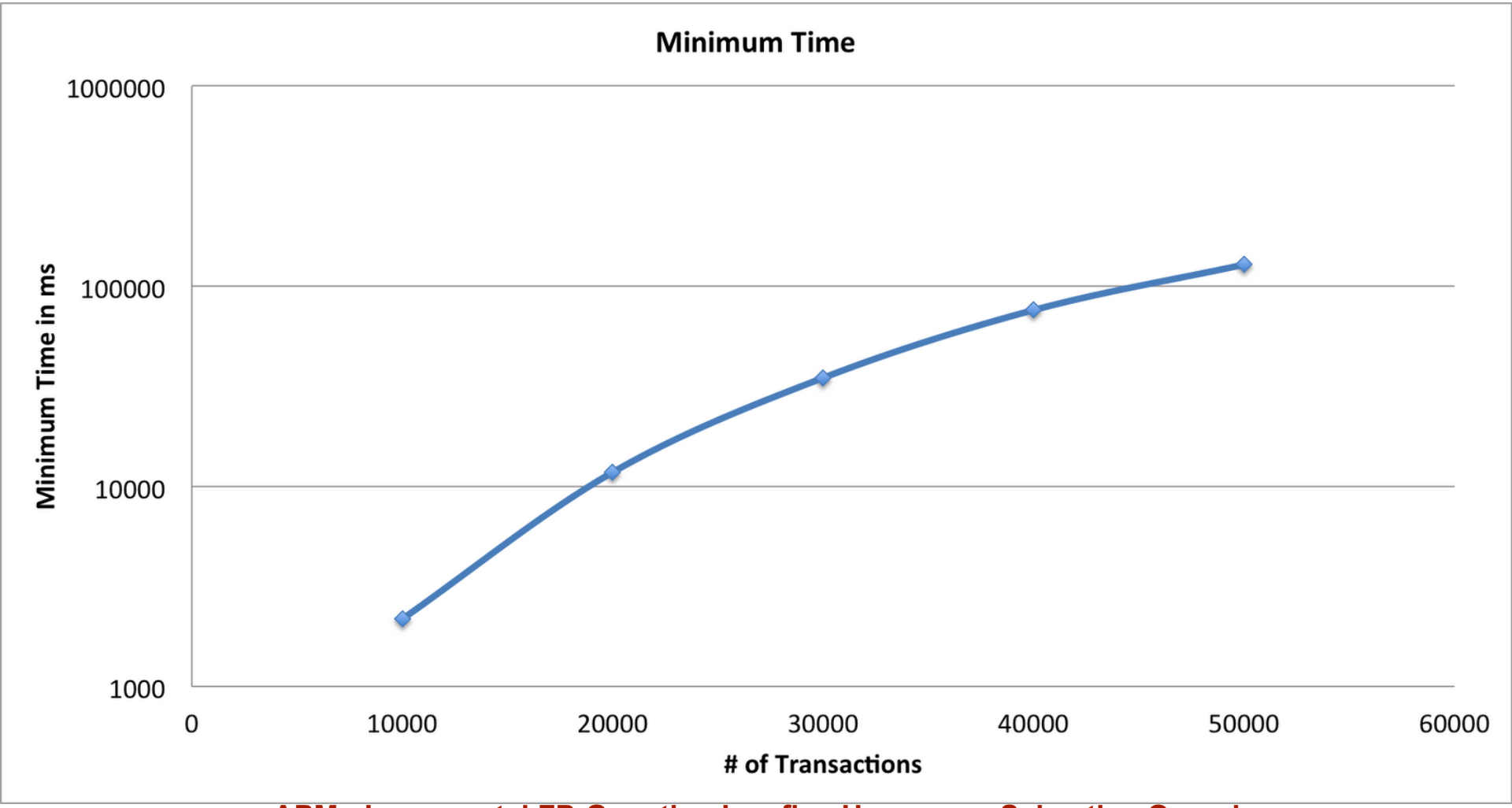
CP-Tree - Experiments

- Runtime for inserting N transactions

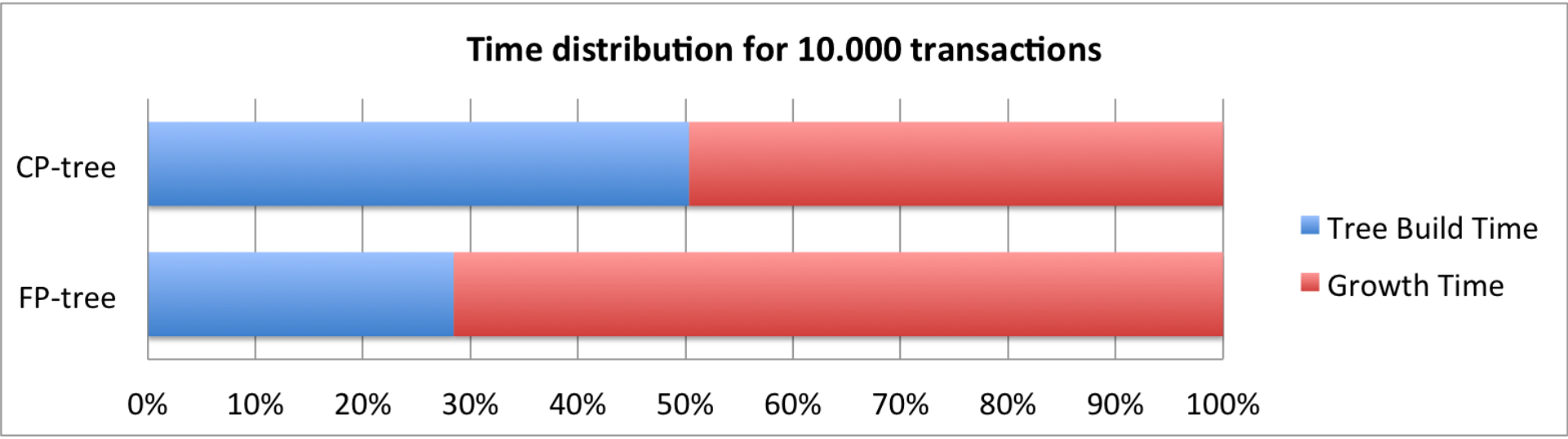


CP-Tree - Experiments

- Runtime for inserting N transactions (log scale)



Comparison - FP-Tree & CP-Tree



Insights

- Bandwidth was a bottleneck
  - Twitter Streaming API transmits a lot of metadata


```
{"completed_in":0.065,"max_id":
page=2&max_id=19158596613216665
since_id=191585966132166657&q=p
+0000","from_user":"JB5113","fr
Benicase","geo":null,"id":19158
{"result_type":"recent"},"profi
86039_9683939_6382765_n_normal.
50290563541040_568986039_968393
href=&quot;http:\/\/twitter.com
Android&lt;\/a&gt;","text":"The
pizza","to_user":null,"to_user_
+0000","from_user":"ClaireMBigg
Biggs","geo":null,"id":19158573
{"result_type":"recent"},"profi
age_url_https":"https:\/\/si0.t
href=&quot;http:\/\/twitter.com
iPhone&lt;\/a&gt;","text":"@San
Avoiding pizza at the same time
Cruel","to_user":"SanaAndTheCit
```


- Structure of data affect performance

- Uncontrolled parallelism should be avoided! (esp. async usage of actors)
  - Tree was not constructed correctly
  - Problem: constructed tree is shared object
    - Access would need to be synchronized

- Additional spam filter would be useful
  - Would also reduce the size of the tree

porn, sex (1696)
followmejp, sougofollow (948)
xxx, porn (930)
sougofollow, followme (898)
...