Locally Self-Adjusting Tree Networks

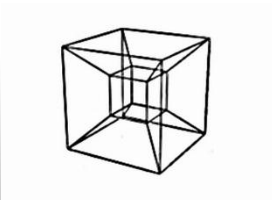
Chen Avin (BGU)
Bernhard Häupler (MIT)
Zvi Lotker (BGU)
Christian Scheideler (U. Paderborn)
Stefan Schmid (T-Labs)

From “Optimal” Networks to Self-Adjusting Networks

- Networks become more and more dynamic (e.g., flexible SDN control)
- Vision: go beyond classic “optimal” static networks
- Example (of this paper): Peer-to-peer

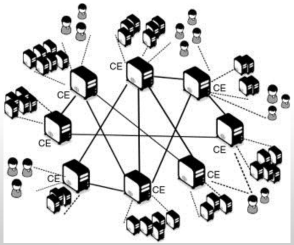
Chord, Pastry, SHELL

- Hypercubic
- Log diameter
- Log degree
- Log routing



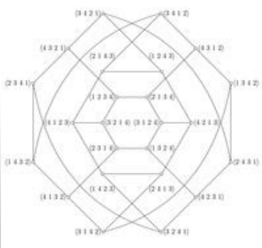
Koorde, ...

- Constant degree
- Log routing



Pancake

- Log/loglog degree and log/loglog routing



From “Optimal” Networks to Self-Adjusting Networks

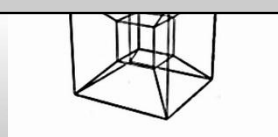
- Networks become more and more dynamic (e.g., flexible SDN control)
- Visi
- Exar

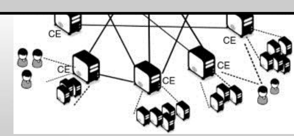
What if networks could self-adjust depending on communication pattern?

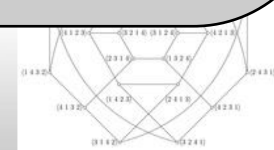
Chor

- Hyp
- Log
- Log
- Log

nd



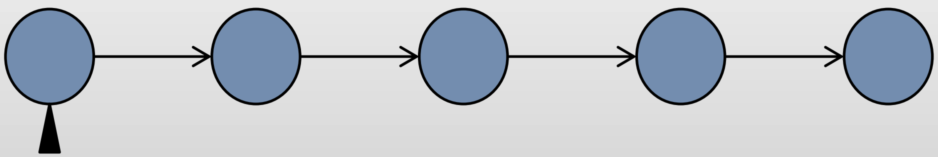




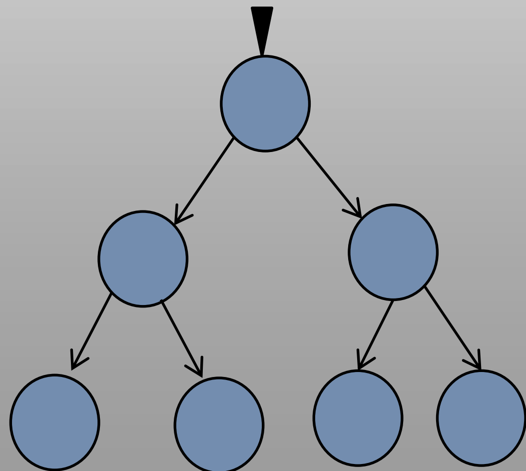
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees

- Linked list: move frequently accessed elements to front!

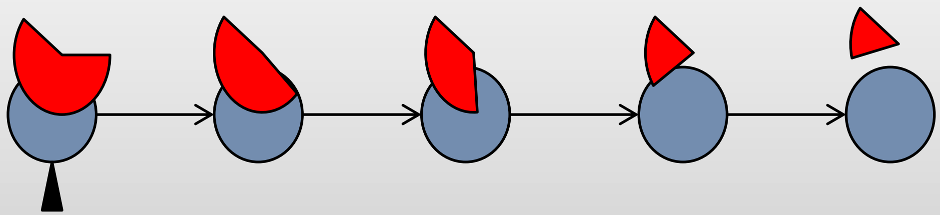


- Trees: move frequently accessed elements closer to root

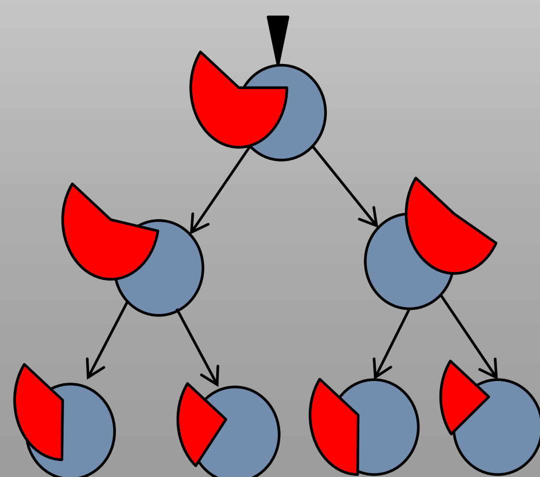


An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!



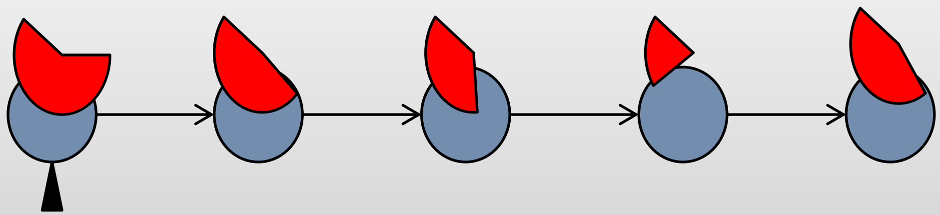
- Trees: move frequently accessed elements closer to root



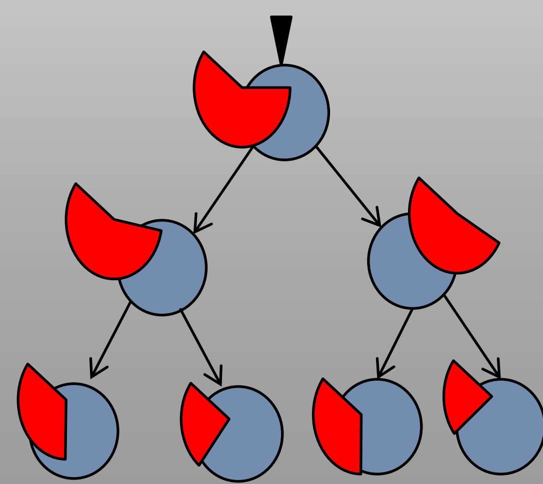
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees

- Linked list: move frequently accessed elements to front!



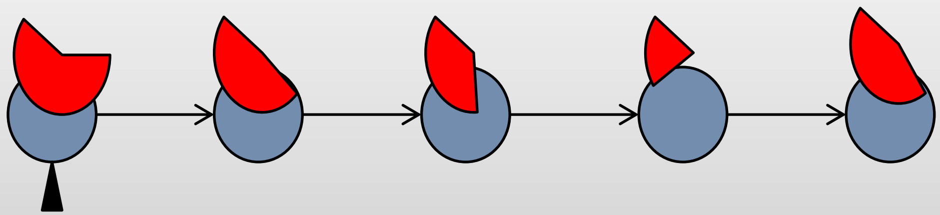
- Trees: move frequently accessed elements closer to root



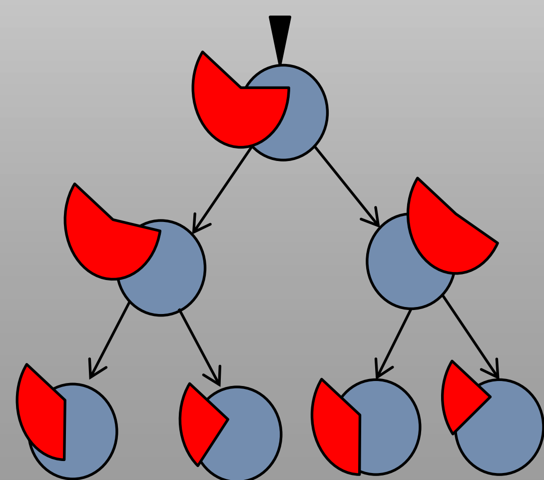
An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees

- Linked list: move frequently accessed elements to front!

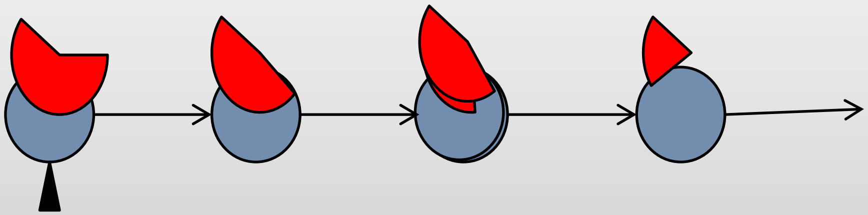


- Trees: move frequently accessed elements closer to root

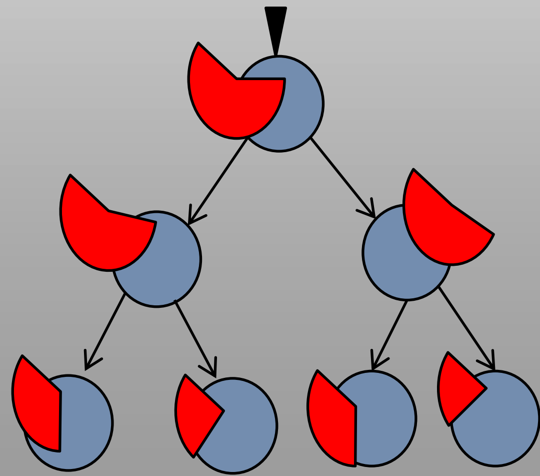


An Old Concept: Move-to-front, Splay Trees, …

- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!

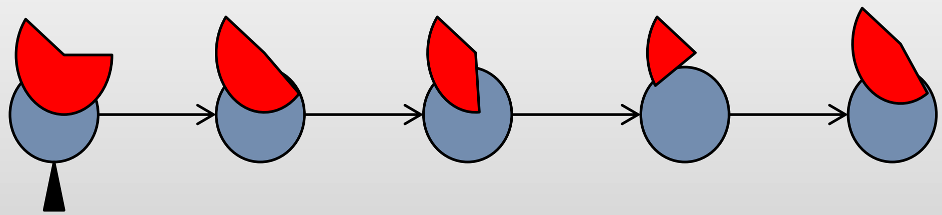


- Trees: move frequently accessed elements closer to root

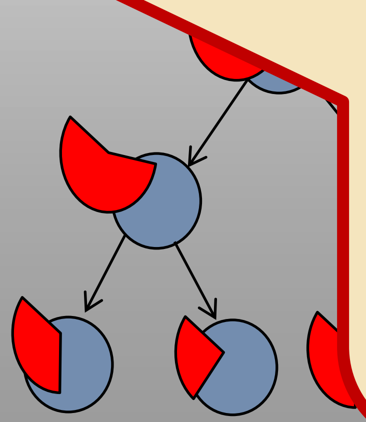


An Old Concept: Move-to-front, Splay Trees, …

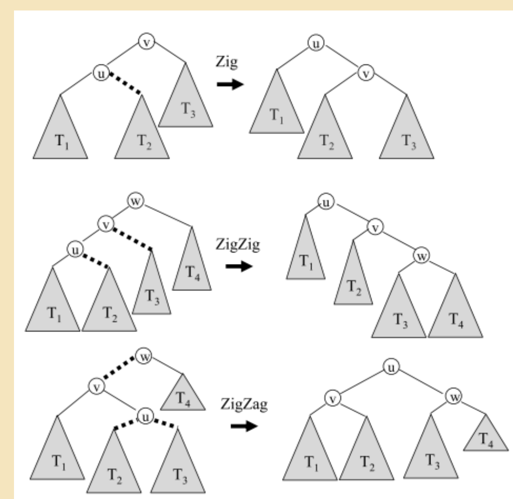
- Classic data structures: lists, trees
- Linked list: move frequently accessed elements to front!



- Trees: move frequently accessed el

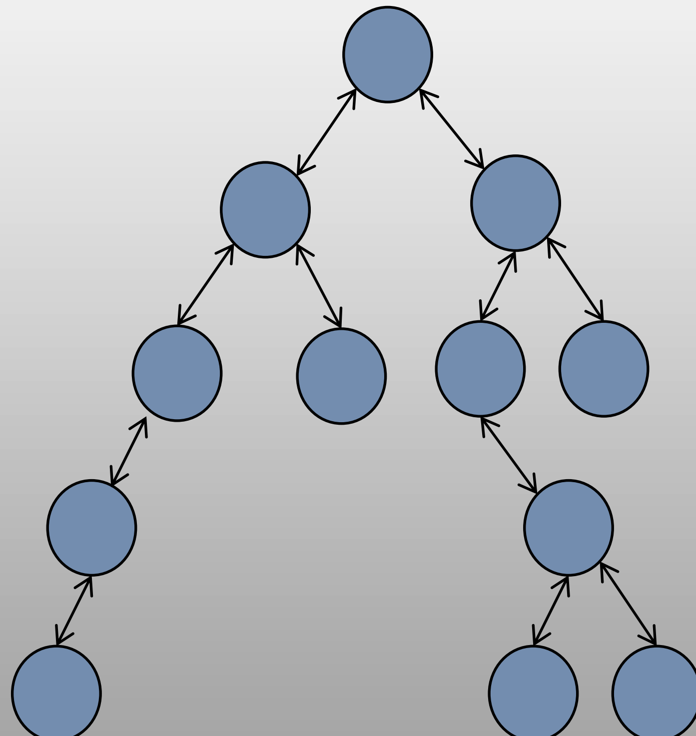


Splay Trees!



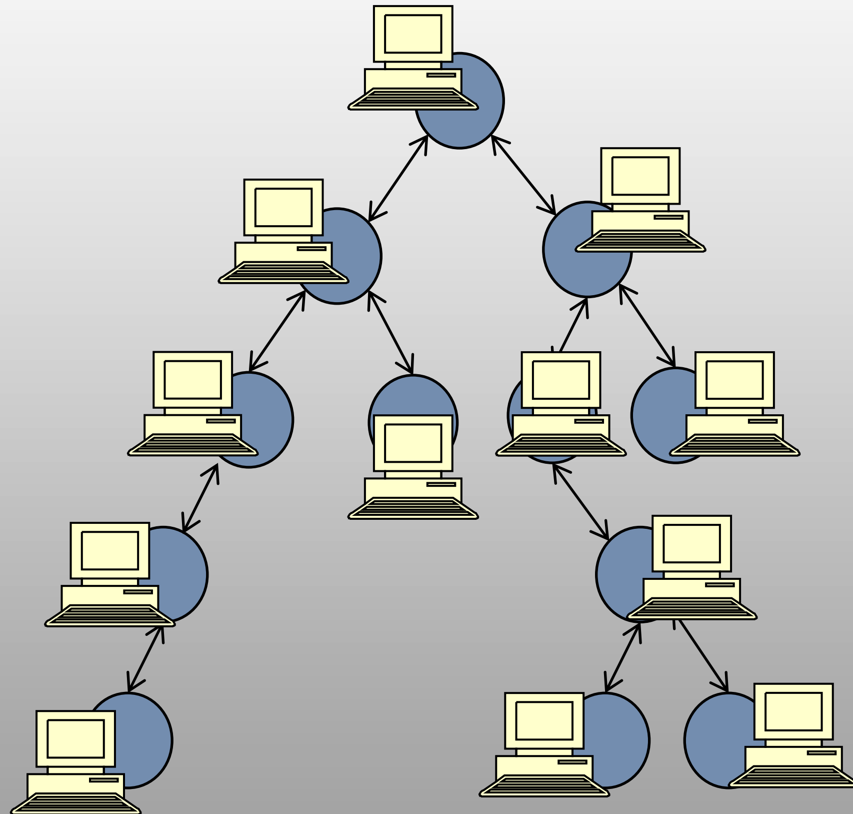
The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)



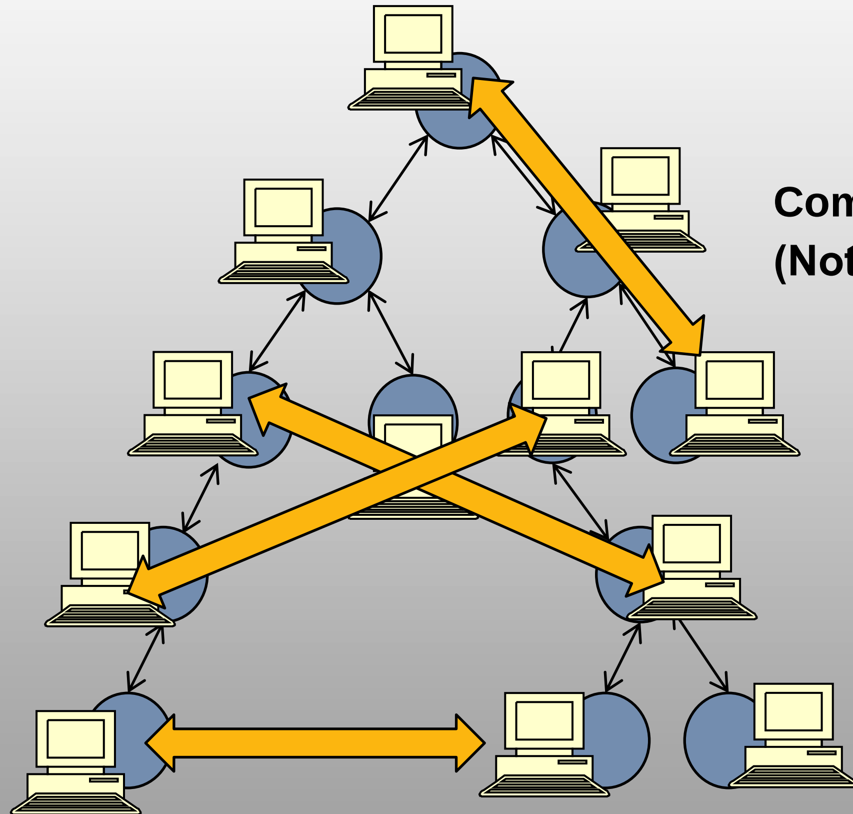
The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)



The Vision: Splay Networks (“Distributed Splay Trees”)

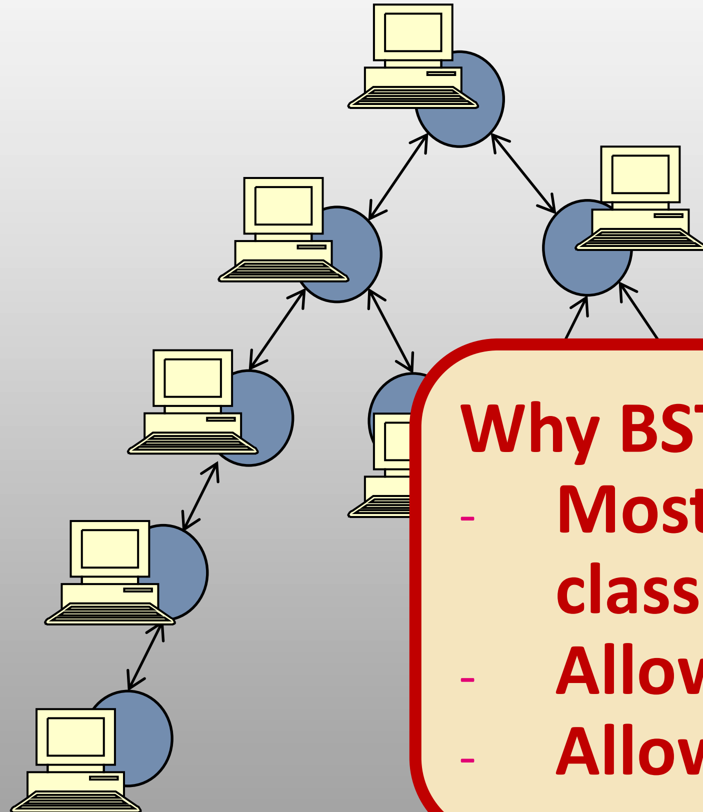
- Most simple self-adjusting tree network: Binary Search Tree (BST)



Communication between peer pairs!
(Not only lookups from root…)

The Vision: Splay Networks ("Distributed Splay Trees")

- Most simple self-adjusting tree network: Binary Search Tree (BST)



Why BST?!

- **Most simple generalization of classic data structure**
- **Allows for local routing!**
- **Allows for algebraic gossip**

Model: Self-Adjusting SplayNets

Input:

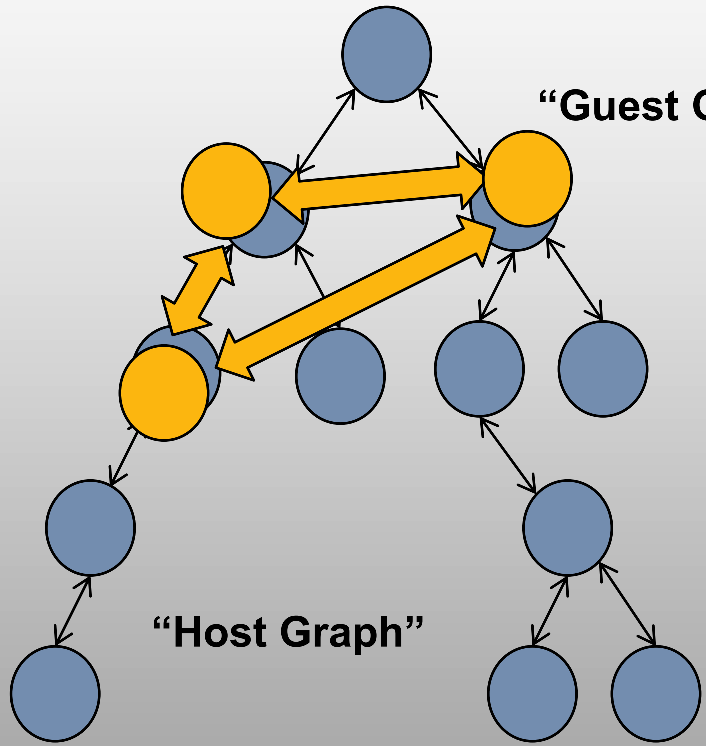
- communication pattern:
  (static or dynamic) graph

Output:

- sequence of network adjustments

Cost metric:

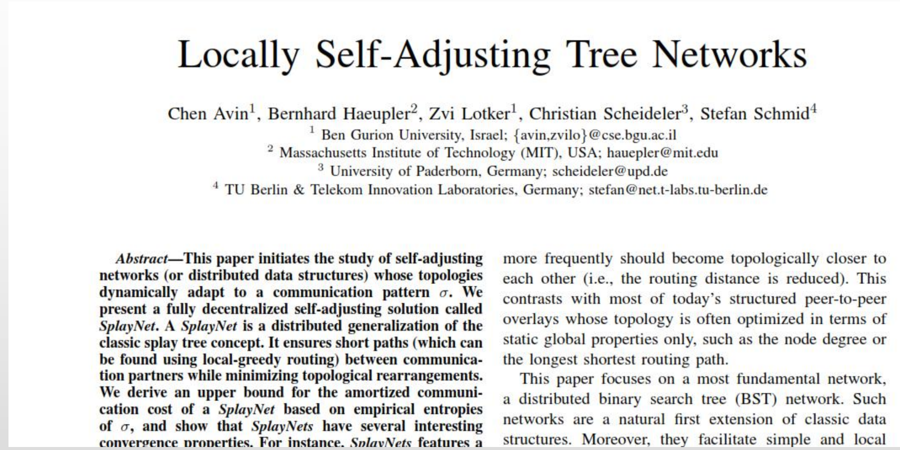
- expected path length
- # (local) network updates



Our Contribution

SplayNets

- "Online algorithm" for self-adjusting distributed trees
- Optimal offline algorithm (polynomial time, for large class of graphs!)



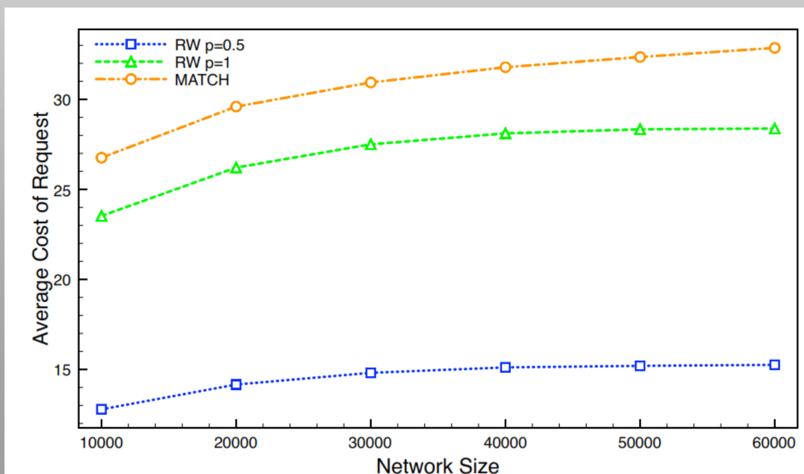
Locally Self-Adjusting Tree Networks

Chen Avin[1], Bernhard Haeupler[2], Zvi Lotker[1], Christian Scheideler[3], Stefan Schmid[4]
[1] Ben Gurion University, Israel; {avin,zvilo}@cse.bgu.ac.il
[2] Massachusetts Institute of Technology (MIT), USA; haeupler@mit.edu
[3] University of Paderborn, Germany; scheideler@upd.de
[4] TU Berlin & Telekom Innovation Laboratories, Germany; stefan@net.t-labs.tu-berlin.de

***Abstract*—This paper initiates the study of self-adjusting networks (or distributed data structures) whose topologies dynamically adapt to a communication pattern σ. We present a fully decentralized self-adjusting solution called *SplayNet*. A *SplayNet* is a distributed generalization of the classic splay tree concept. It ensures short paths (which can be found using local-greedy routing) between communication partners while minimizing topological rearrangements. We derive an upper bound for the amortized communication cost of a *SplayNet* based on empirical entropies of σ, and show that *SplayNets* have several interesting convergence properties. For instance, *SplayNets* features a**

more frequently should become topologically closer to each other (i.e., the routing distance is reduced). This contrasts with most of today's structured peer-to-peer overlays whose topology is often optimized in terms of static global properties only, such as the node degree or the longest shortest routing path.

This paper focuses on a most fundamental network, a distributed binary search tree (BST) network. Such networks are a natural first extension of classic data structures. Moreover, they facilitate simple and local



Performance evaluation:

- General bounds on amortized costs
- Lower bounds (empirical entropy)
- Analysis of convergence times for important static comm. patterns
- Optimality of online algorithm for special patterns (e.g., matchings)
- Simulation study (Facebook data)

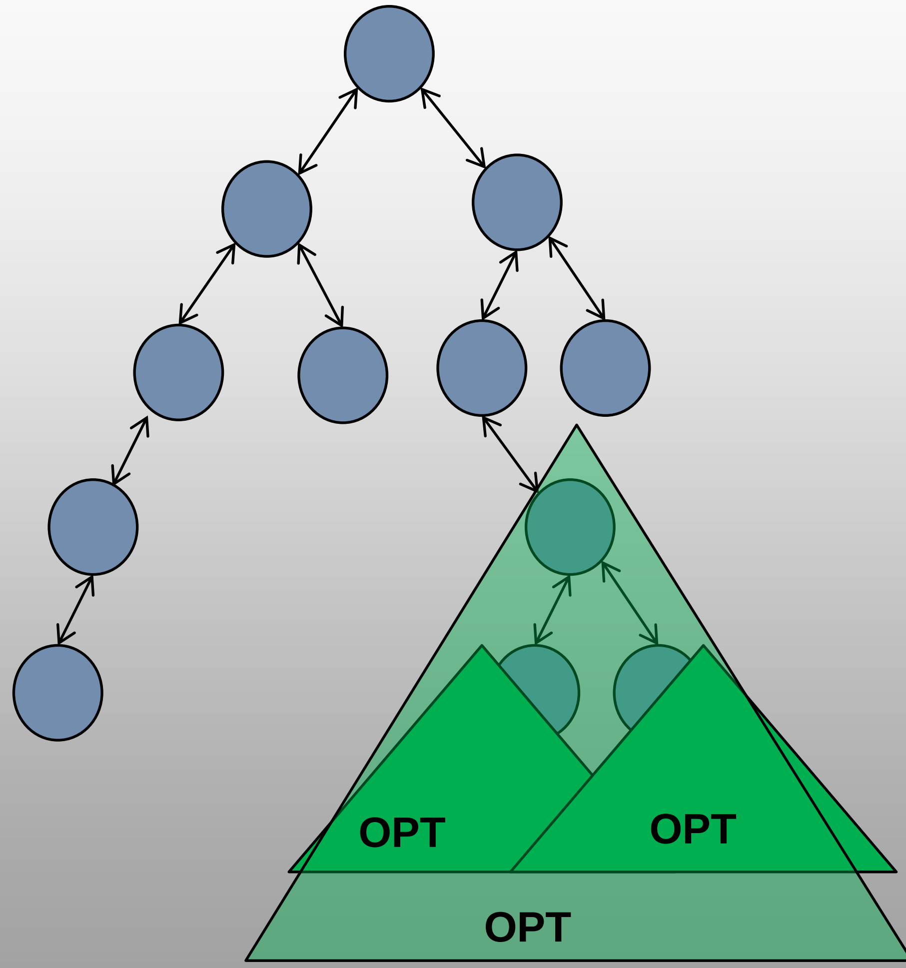
The Optimal Offline Solution

Dynamic program

- Binary search:
  decouple left from right!
- Polynomial time
  (unlike MLA!)
- So: solved M"BST"A

See also:

- Related problem of
  phylogenetic trees



The Online SplayNets Algorithm

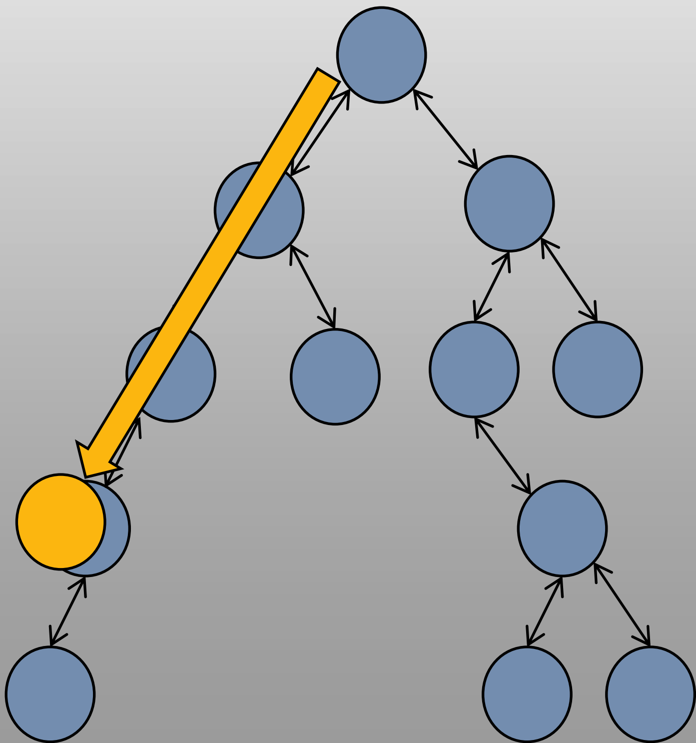
From Splay tree to SplayNet:

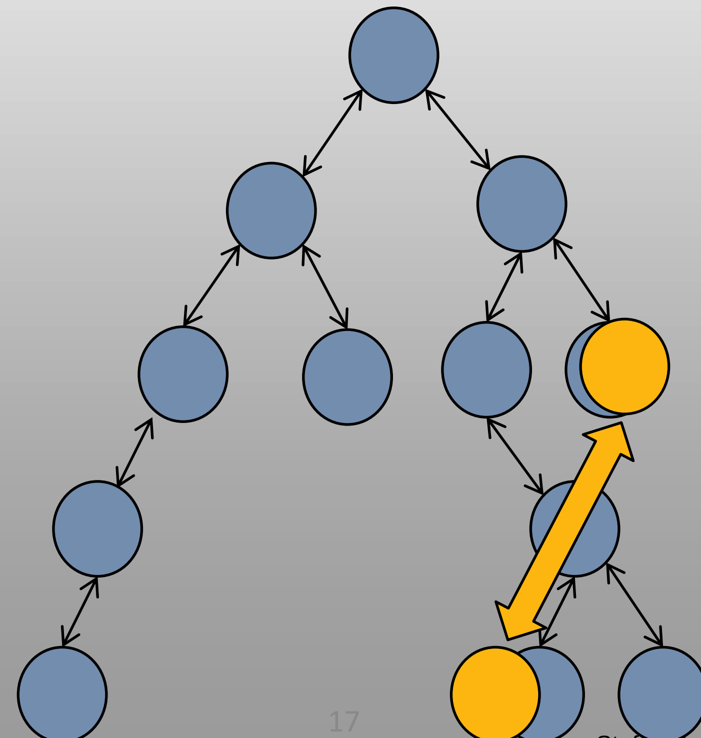
Algorithm 1 Splay Tree Algorithm $\texttt{ST}$

1: (* upon lookup (u) *)
2: **splay** u to root of T

Algorithm 2 Double Splay Algorithm $\texttt{DS}$

1: (* upon request (u, v) in T *)
2: $w := \alpha_T(u, v)$
3: $T' :=$ **splay** u to root of $T(w)$
4: **splay** v to the child of $T'(u)$





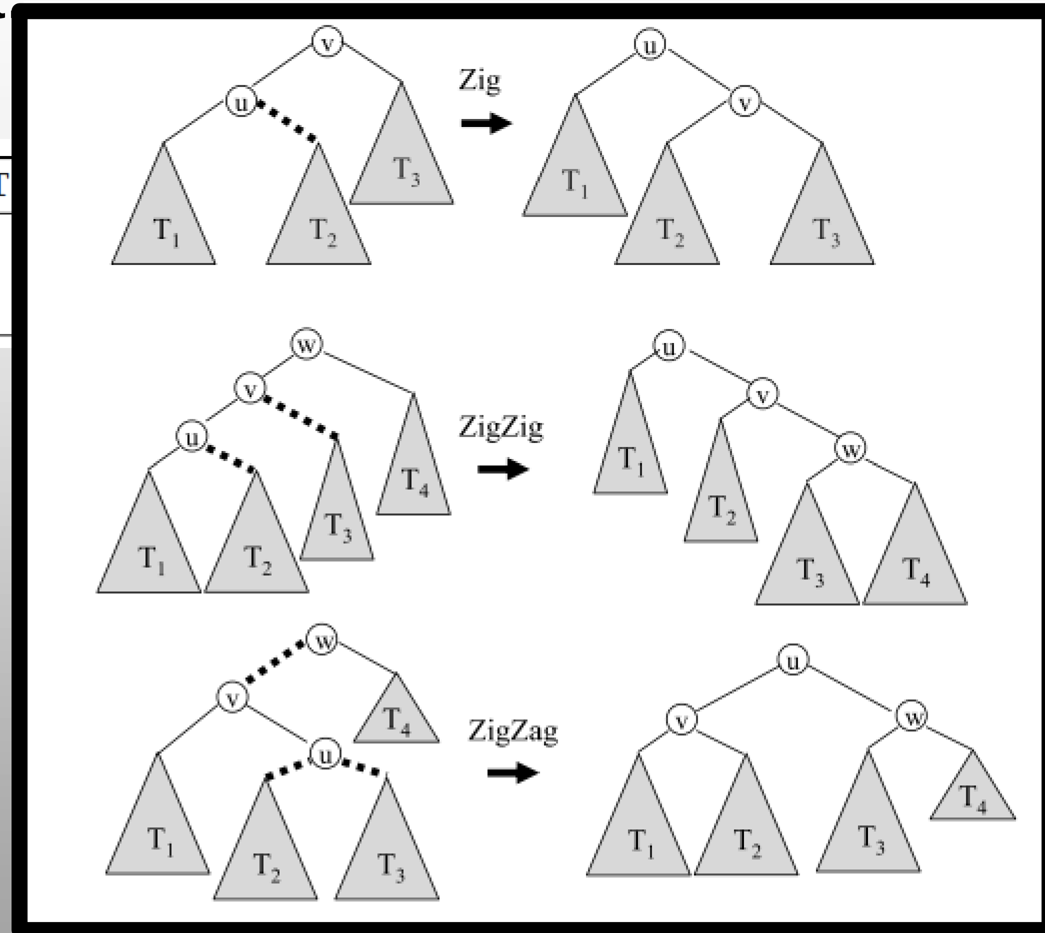
The Online SplayNets Algorithm

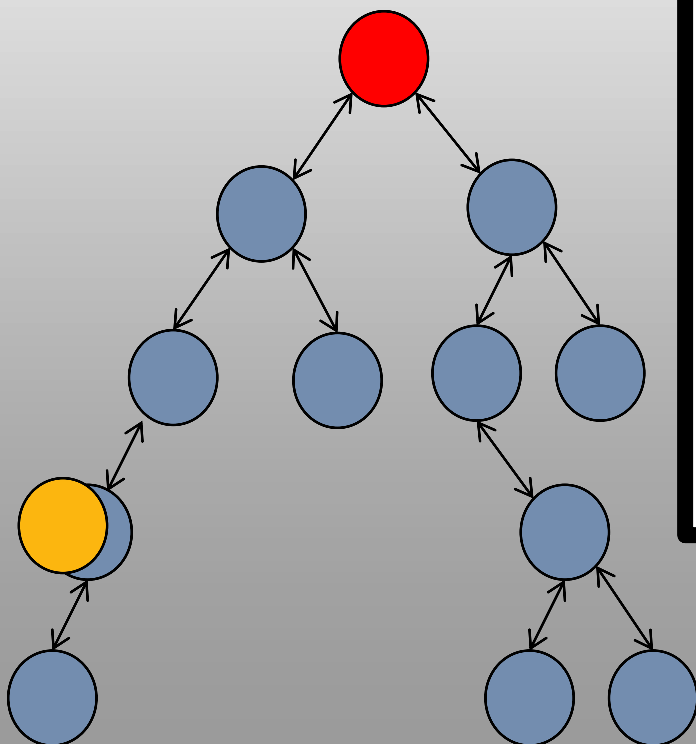
From Splay tree to SplayNet:

Algorithm 1 Splay Tree Algorithm ST

(* upon lookup (u) *)

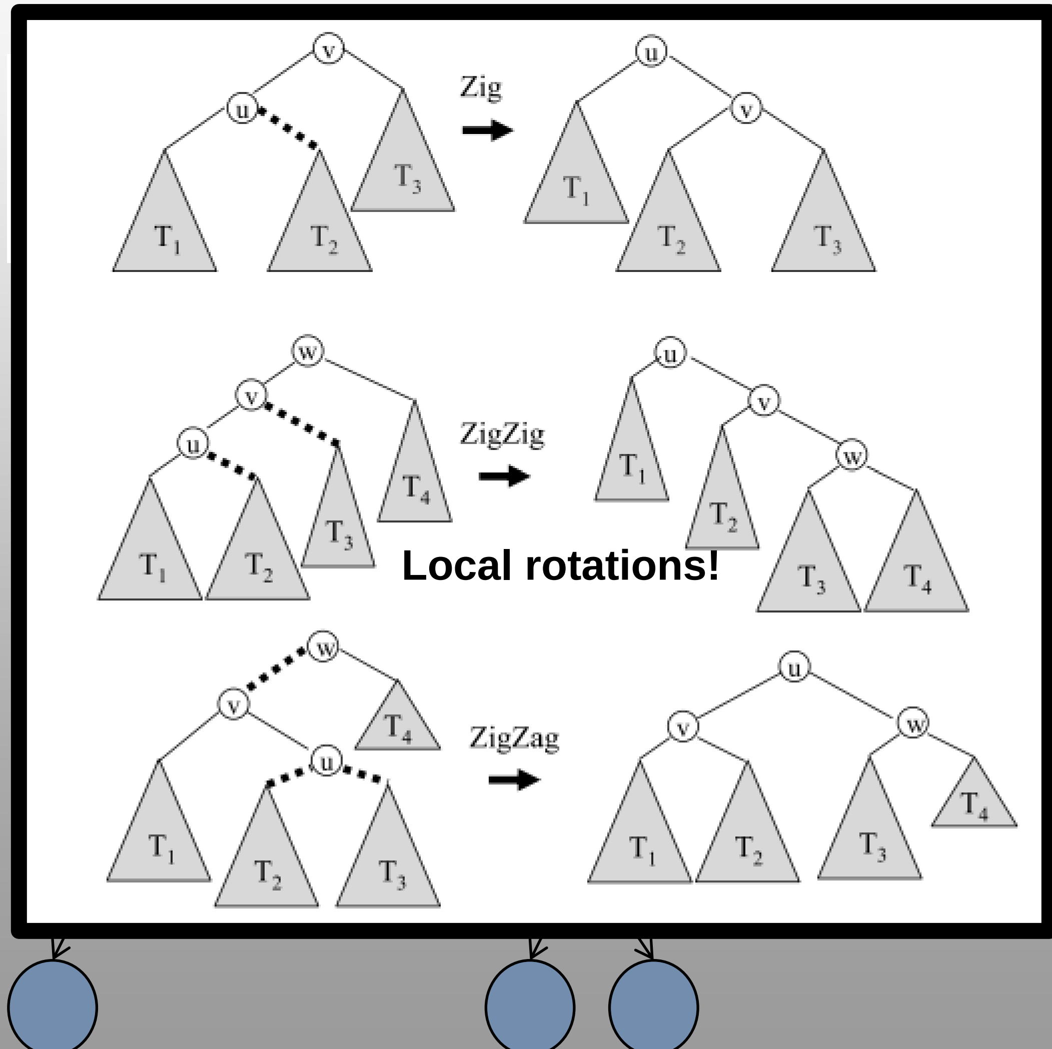
splay u to root of T





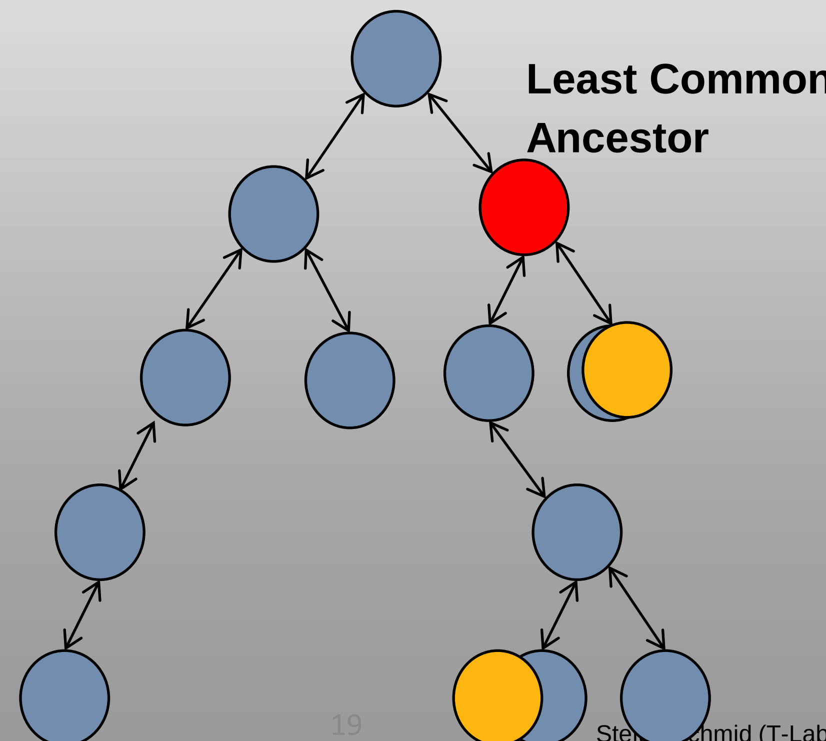
The Online SplayNets Algorithm

From Splay tree to SplayNet:



Algorithm 2 Double Splay Algorithm DS

1: (* upon request (u, v) in T *)
2: $w := \alpha_T(u, v)$
3: $T' :=$ **splay** u to root of $T(w)$
4: **splay** v to the child of $T'(u)$



Analysis: Basic Lower and Upper Bounds

Upper Bound

A-Cost < H(X) + H(Y)

where H(X) and H(Y) are empirical entropies of sources resp. destinations

Adaption of Tarjan&Sleator

Lower Bound

A-Cost > H(X|Y) + H(Y|X)

where H(|) are conditional entropies.

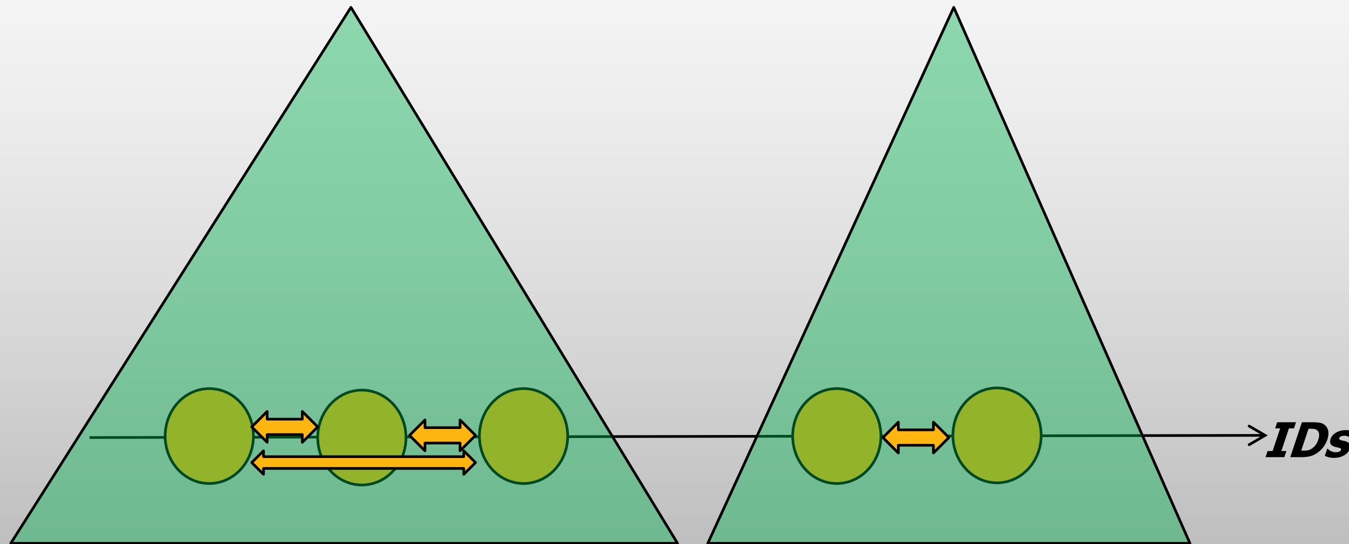
Assuming that each node is the root for “its tree”

Therefore, our algorithm is optimal, e.g., if communication pattern describes a product distribution!

Properties: Convergence

Cluster scenario:

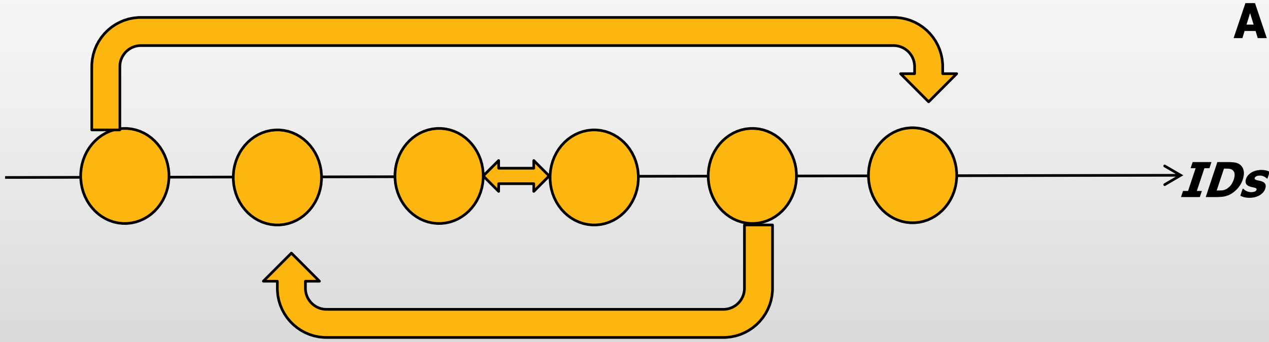
Nodes communicate within local clusters only!



Over time, nodes will form clusters in BST! No paths "outside".

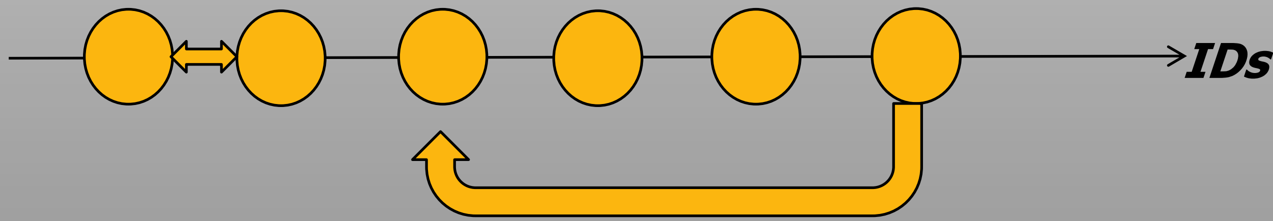
Properties: Optimal Solutions

Laminated scenario:



Will converge to optimum: Amortized costs 1.

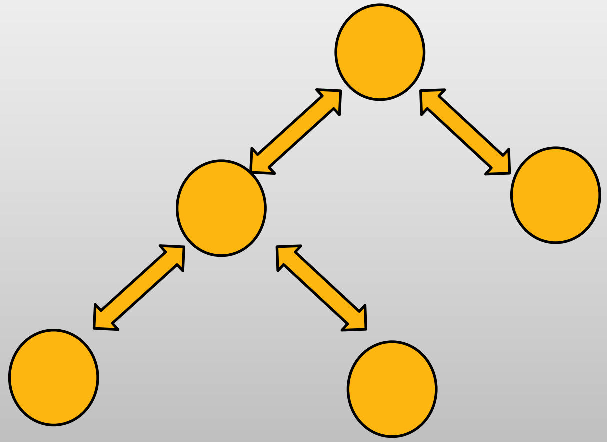
Non-crossing matching (= “no polygamy”) scenario:



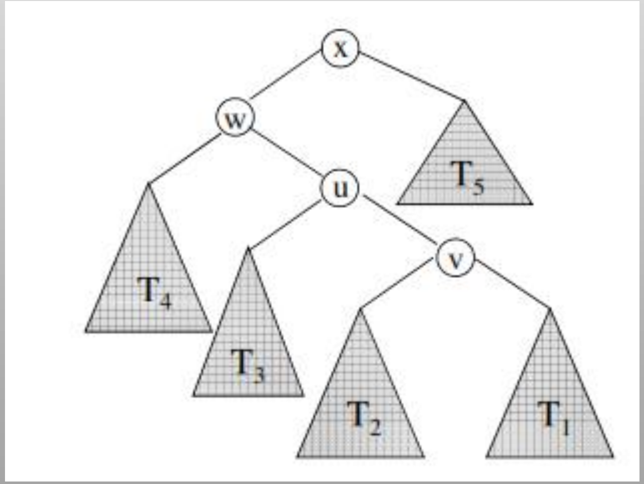
Will converge to optimum: Amortized costs 1.

Properties: Optimal Solutions

Multicast scenario (BST): Example

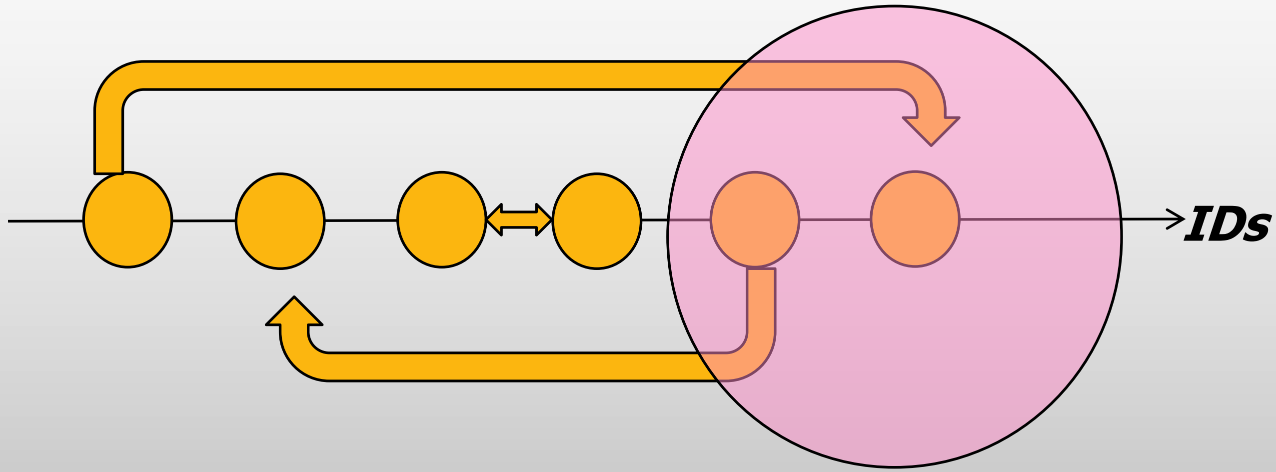


Invariant over “stable” subtrees (from right):

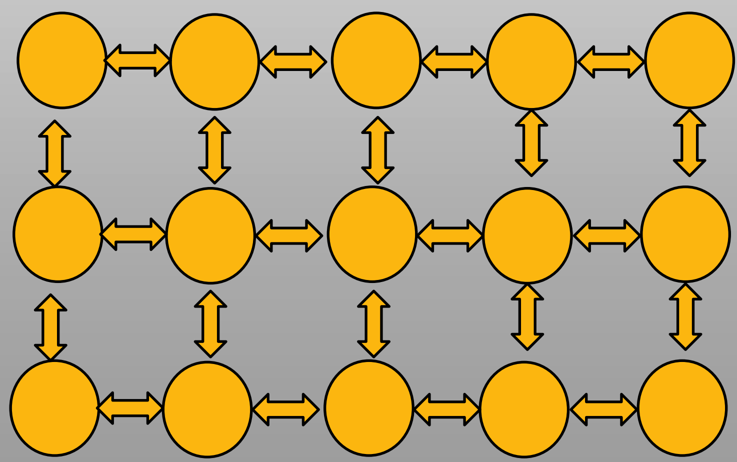


Improved Lower Bounds (and More Optimality)

Via interval cuts or conductance entropy:

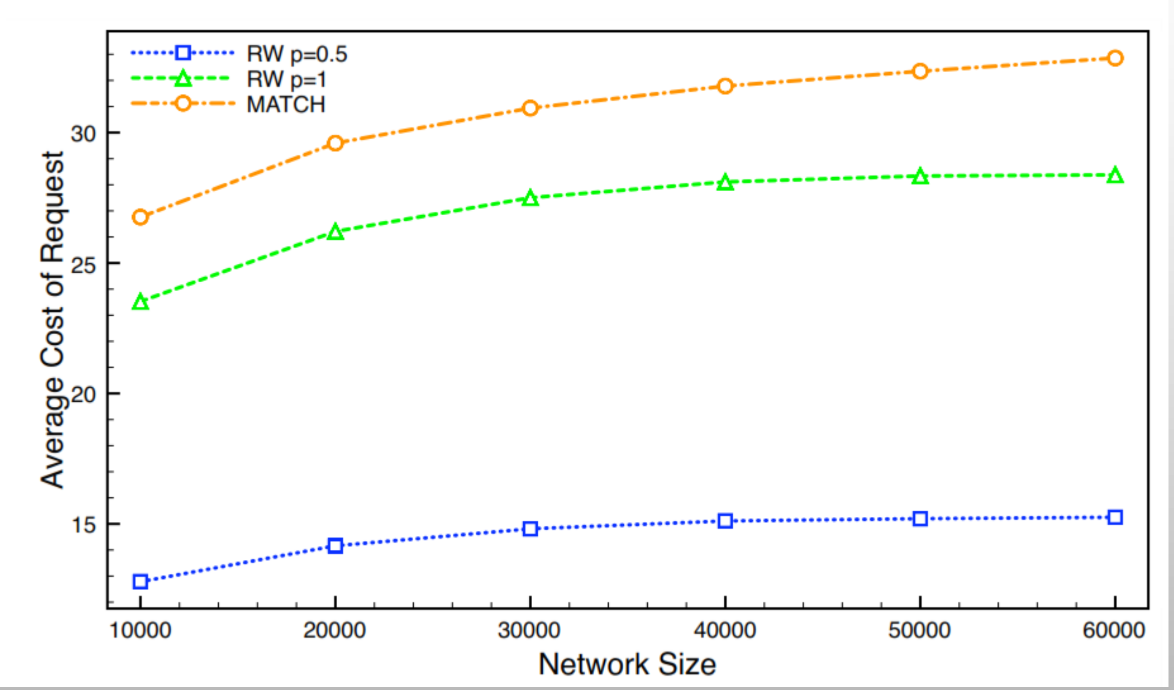


Grid:



Cut of interval: entropy yields amortized costs!

Simulation Results



- Facebook component with 63k nodes and 800k edges
- SplayNet exploit random walk locality, to less extent also matching

Conclusion

- Vision: self-adjusting networks

- Interesting generalization of Splay trees

- SplayNets
  - Formal analysis reveals nice properties
  - Amortized costs good: but tight?
  - Competitive ratio remains open

- Future work? Yes ☺

Thank you! Questions?

Algorithm 2 Double Splay Algorithm DS

1: (* upon request (u, v) in T *)
2: $w := \alpha_T(u, v)$
3: $T' :=$ **splay** u to root of $T(w)$
4: **splay** v to the child of $T'(u)$