Reconfigurable Networks: Enablers, Algorithms, Complexity

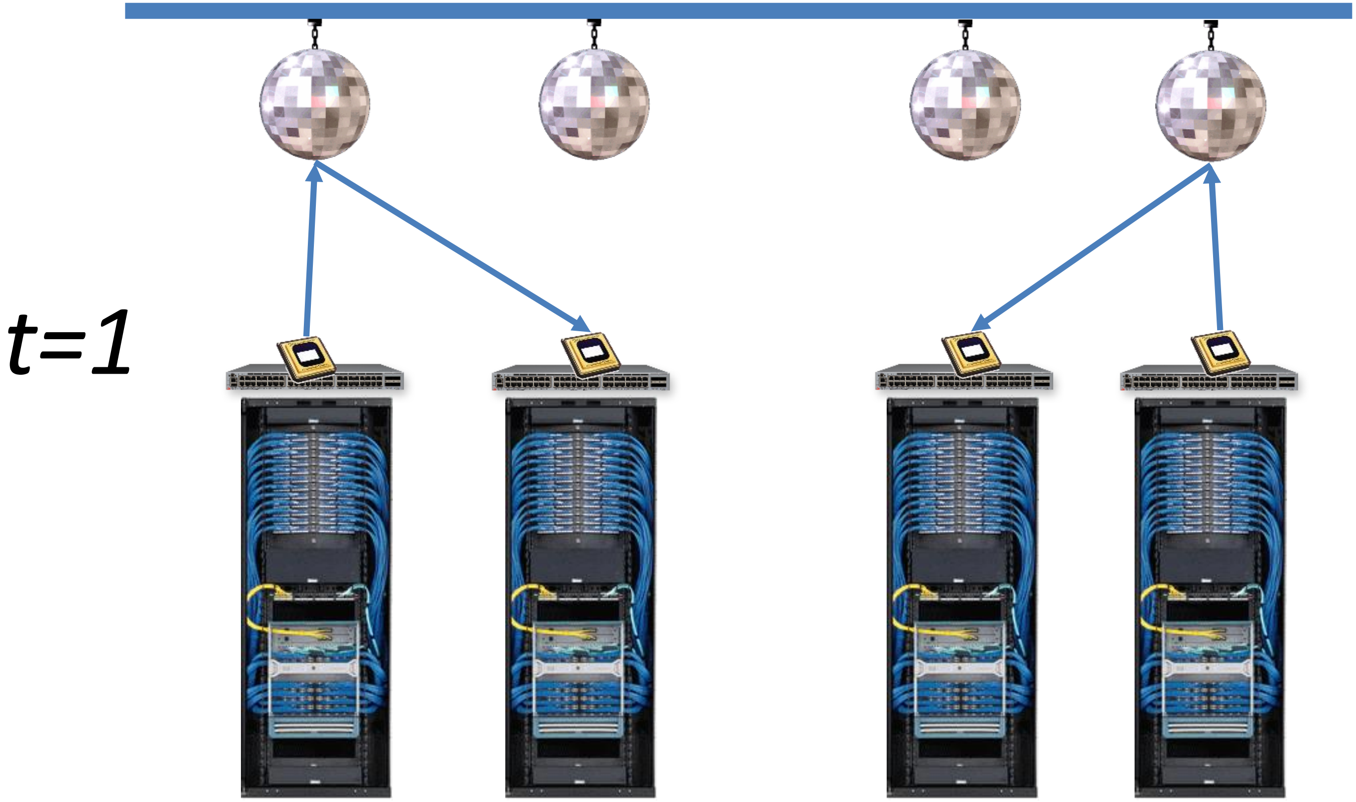
Ramakrishnan Durairajan, Klaus-Tycho Förster, Stefan Schmid

Tutorial @ ACM Sigmetrics 2019
Phoenix, Arizona, USA

A demand-aware network with fast reconfigurations + high fanout: ProjecToR

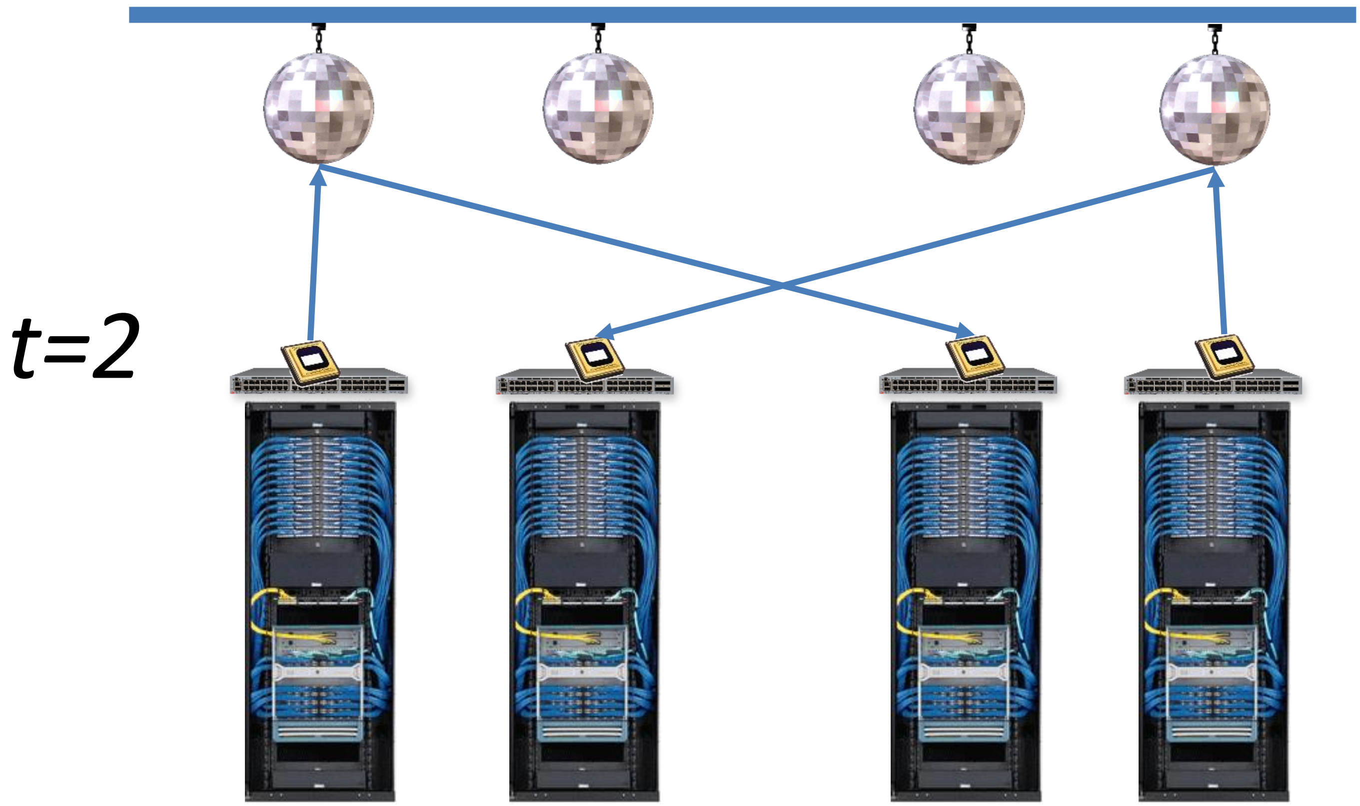
Manya Ghobadi et al.*
(* collaborators from U. Arizona!)
Kudos for some slides!

ProjecToR



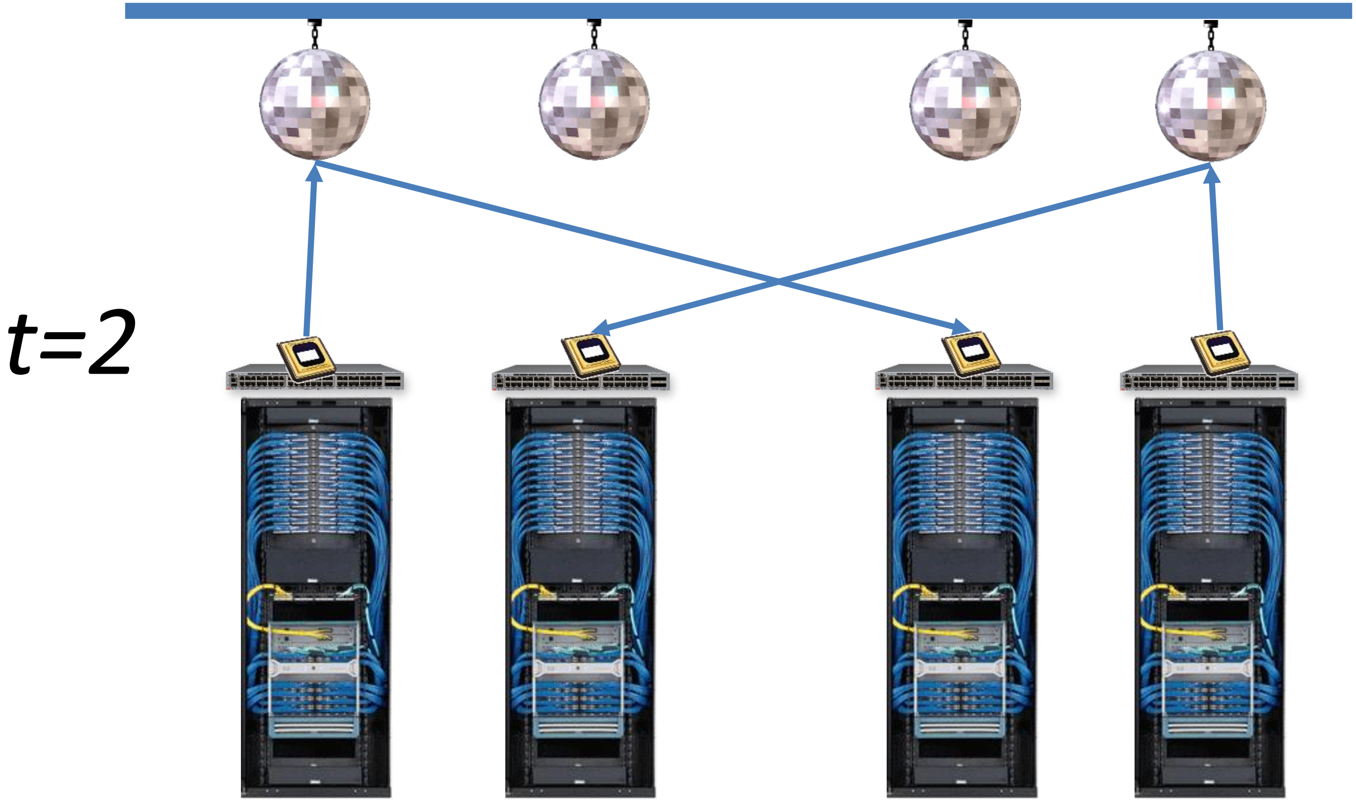
- Based on **free-space optics**
- Demand-aware

ProjecToR

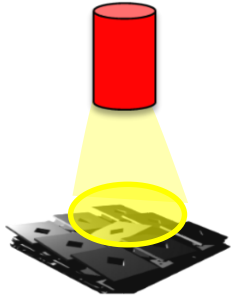


- Based on **free-space optics**
- Demand-aware

ProjecToR



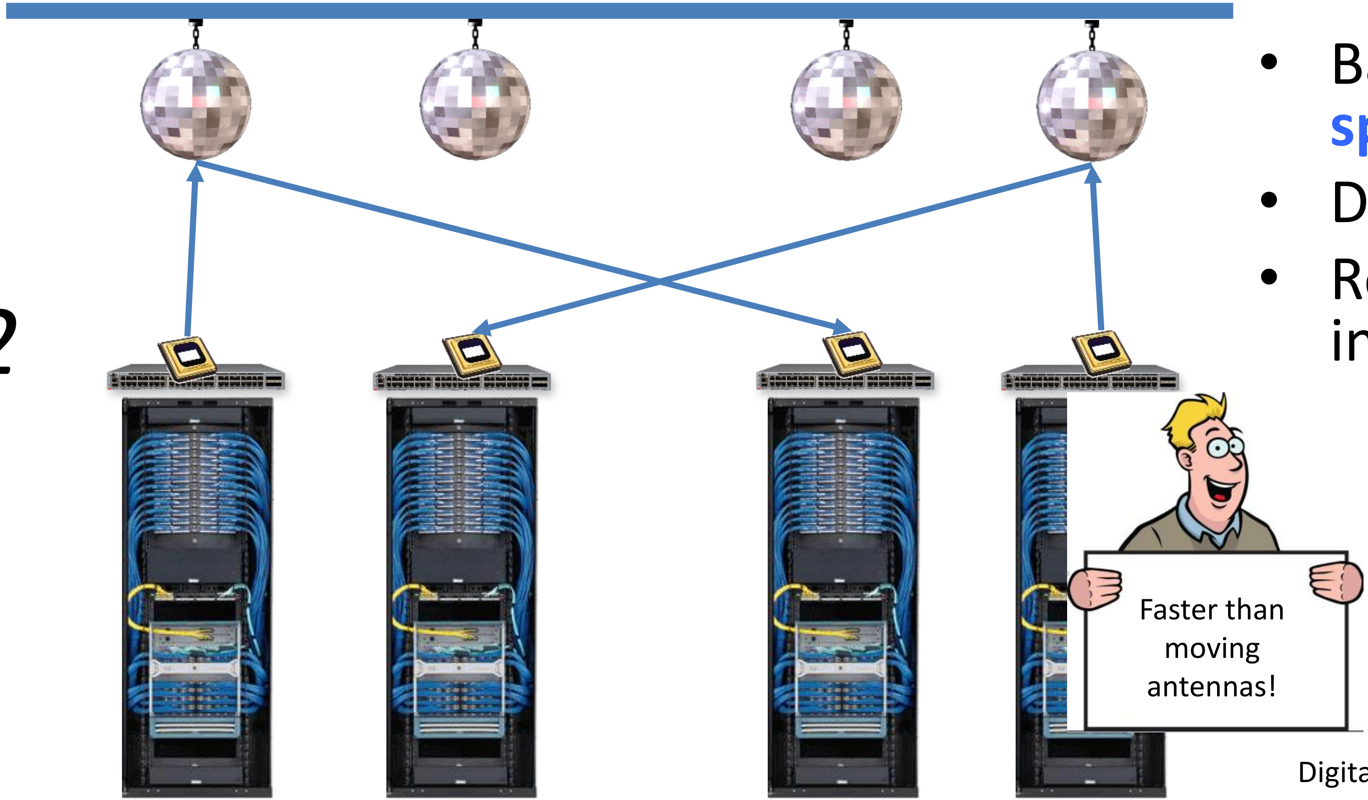
- Based on **free-space optics**
- Demand-aware
- Reconfiguration in ~10 µs:



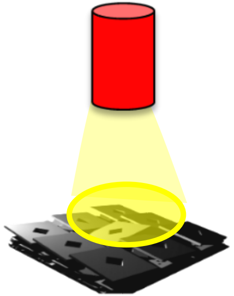
Digital Micromirror Devices (DMDs)

ProjecToR

t=2



- Based on **free-space optics**
- Demand-aware
- Reconfiguration in ~10 μs:



Digital Micromirror Devices (DMDs)

ProjecToR in More Details: Empirical Motivation

	A	B	C	D
A	0	3	3	3
B	3	0	3	3
C	3	3	0	3
D	3	3	3	0

Ideal demand matrix: uniform and static

	A	B	C	D
A	0	0 6	0 6	0
B	0	0	0	0 7
C	0 8	0	0 6	0 7
D	0	0 12	0 8	0

Non-ideal demand matrix: **skewed** and **dynamic**

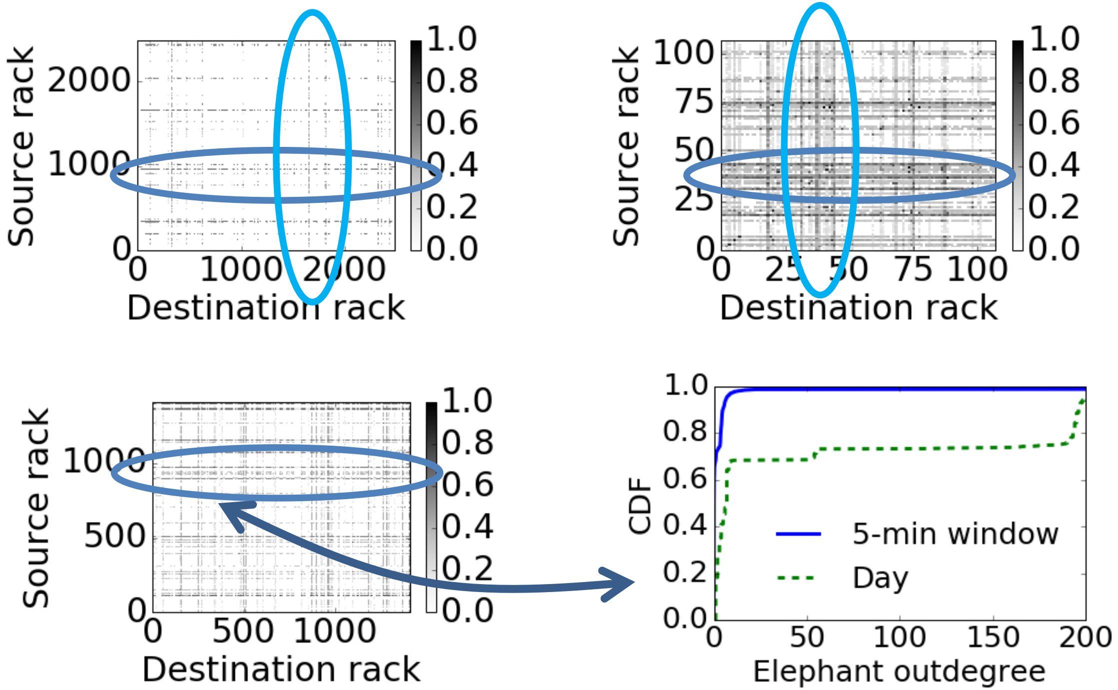
ProjecToR in More Details: Empirical Motivation

Observation 1:

- Many rack pairs exchange ***little traffic***
- Only some ***hot rack pairs*** are active

Observation 2:

- Some source racks send large amounts of traffic ***to many other racks***



Microsoft data: 200K servers across 4 production clusters, cluster sizes: 100 - 2500 racks.
Mix of workloads: MapReduce-type jobs, index builders, database and storage systems.

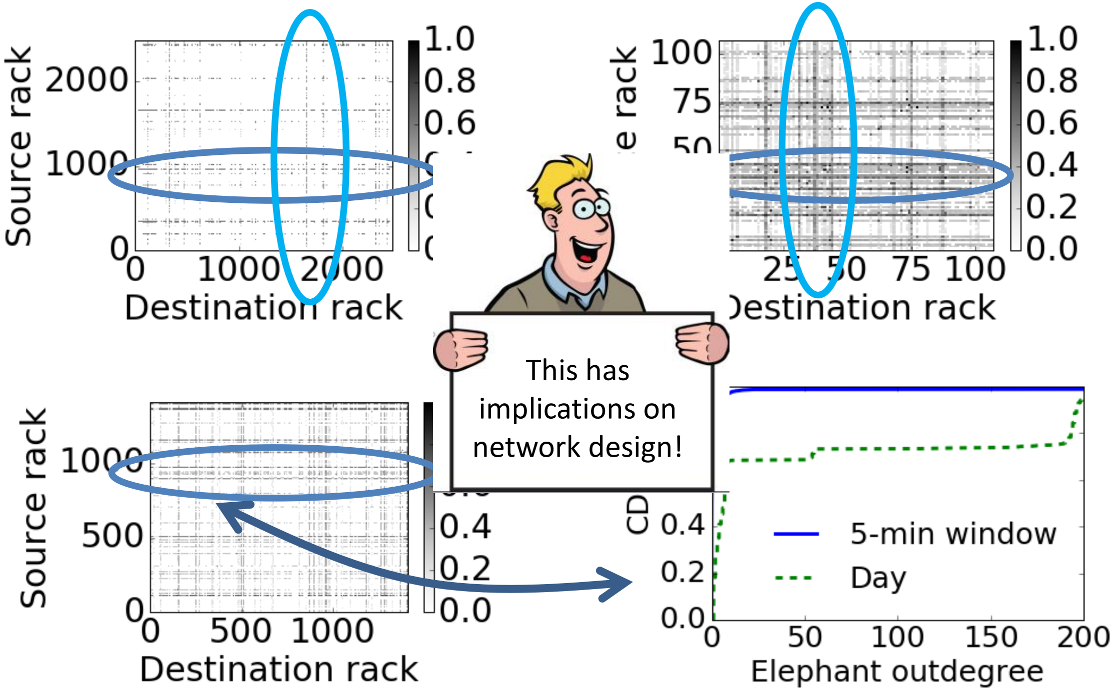
ProjecToR in More Details: Empirical Motivation

Observation 1:

- Many rack pairs exchange ***little traffic***
- Only some ***hot rack pairs*** are active

Observation 2:

- Some source racks send large amounts of traffic ***to many other racks***

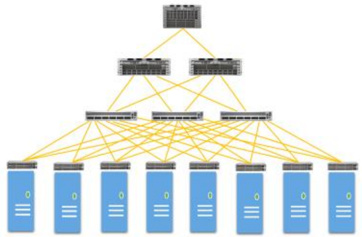


Microsoft data: 200K servers across 4 production clusters, cluster sizes: 100 - 2500 racks.
Mix of workloads: MapReduce-type jobs, index builders, database and storage systems.

ProjecToR in More Details: Empirical Motivation

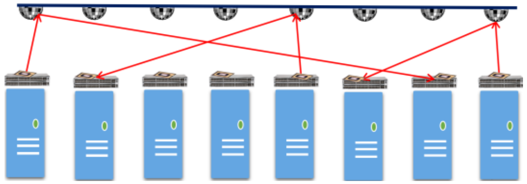
Implication for **static topology** with uniform capacity:

- ***Over-provisioned*** for most rack pairs
- ***Under-provisioned*** for few others

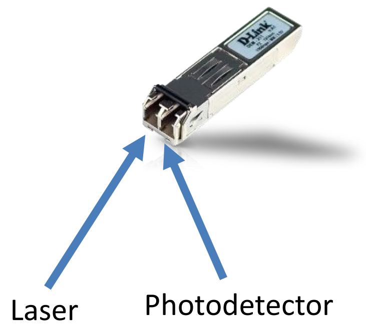


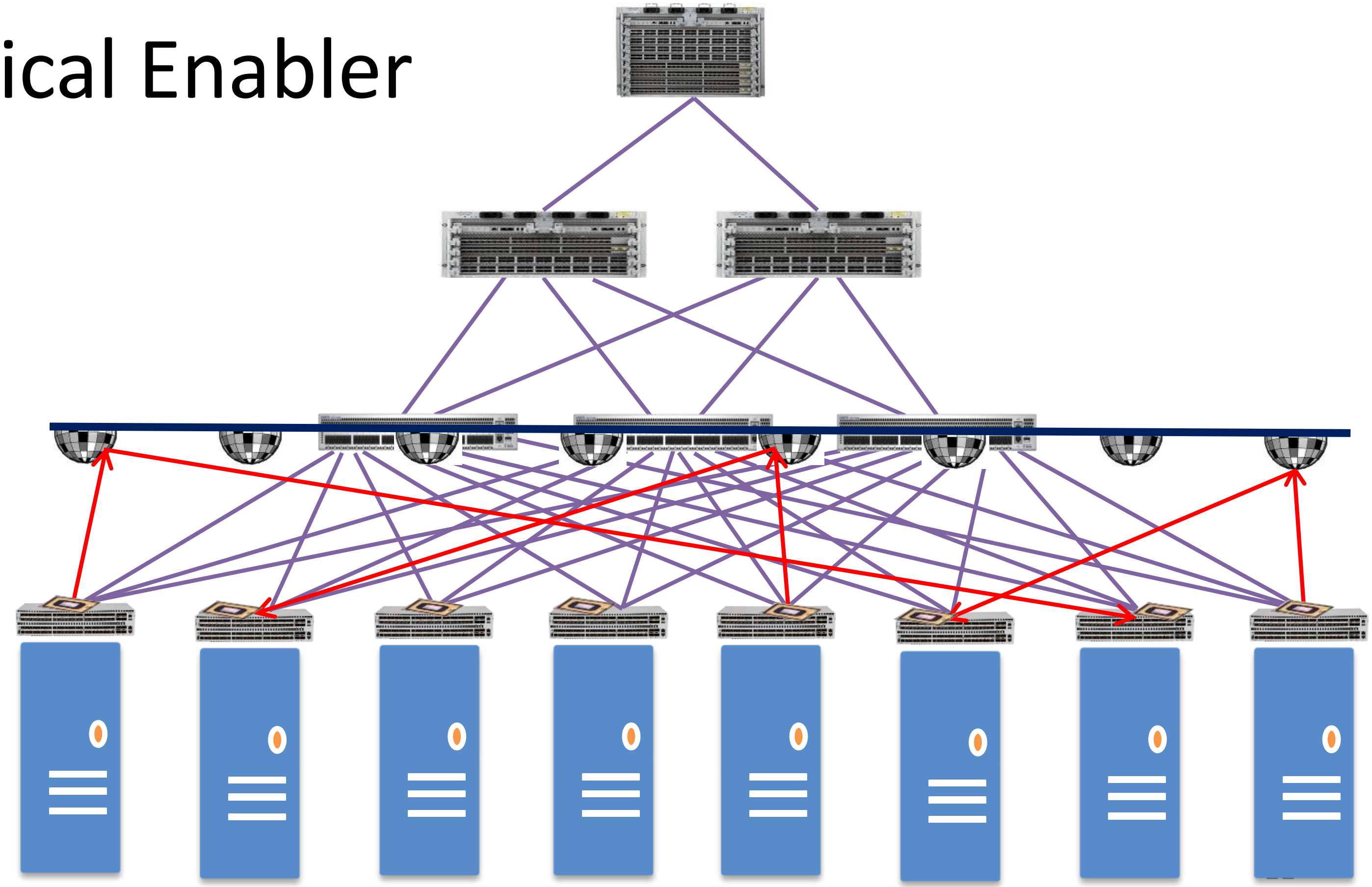
Implications for **dynamic topology**:

- Must be able to create direct links to lots of other racks (***high fan-out***)
- And switch quickly among destinations (***low switching time***)

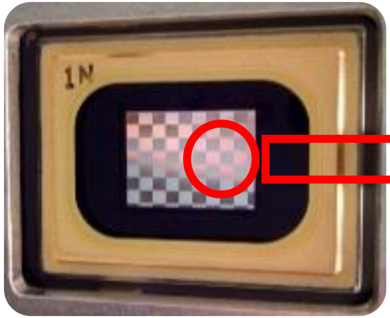


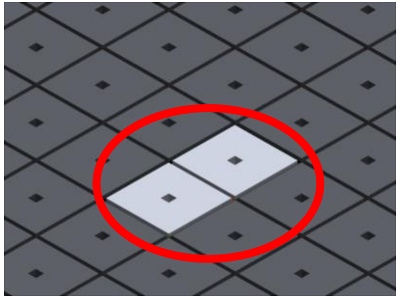
ProjecToR in More Details: Technological Enabler

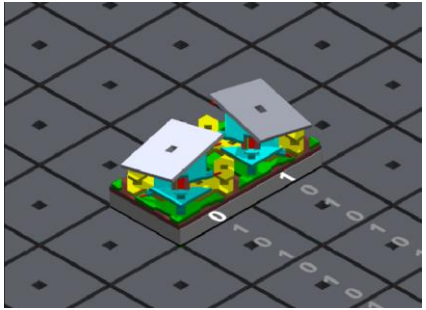




ProjecToR in More Details: DMDs







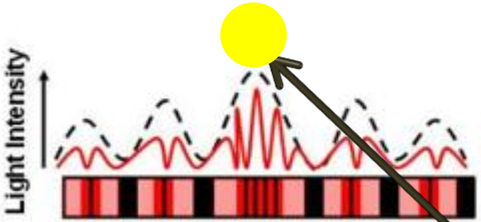
Texas Instruments

Array of micromirrors

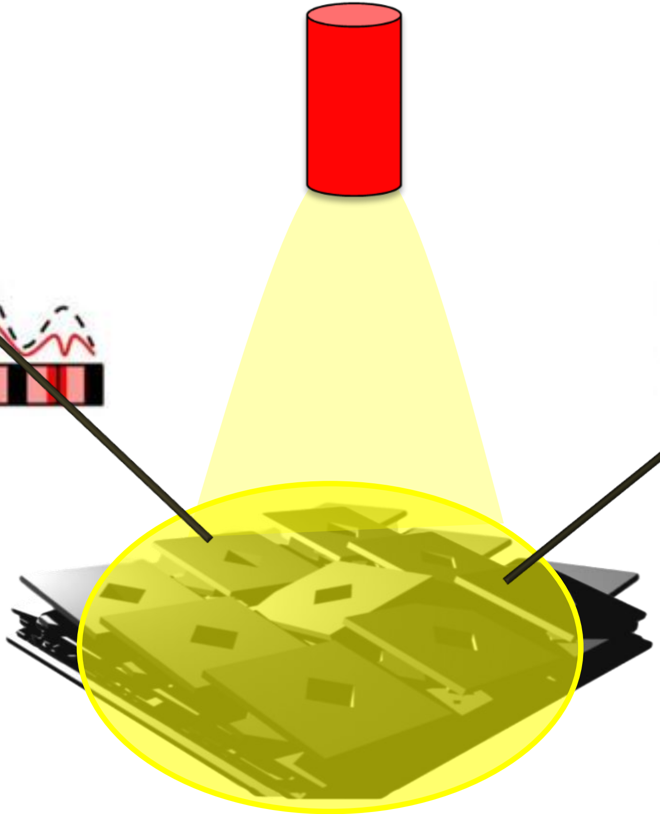
Memory cell

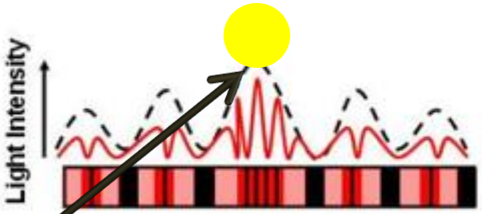
- Each micromirror can be turned on/off
- Essentially a **0/1-image**: e.g., array size 768 x 1024
- Direction of the diffracted light can be finely tuned

ProjecToR in More Details:
DMDs to Redirect Light *Fast*



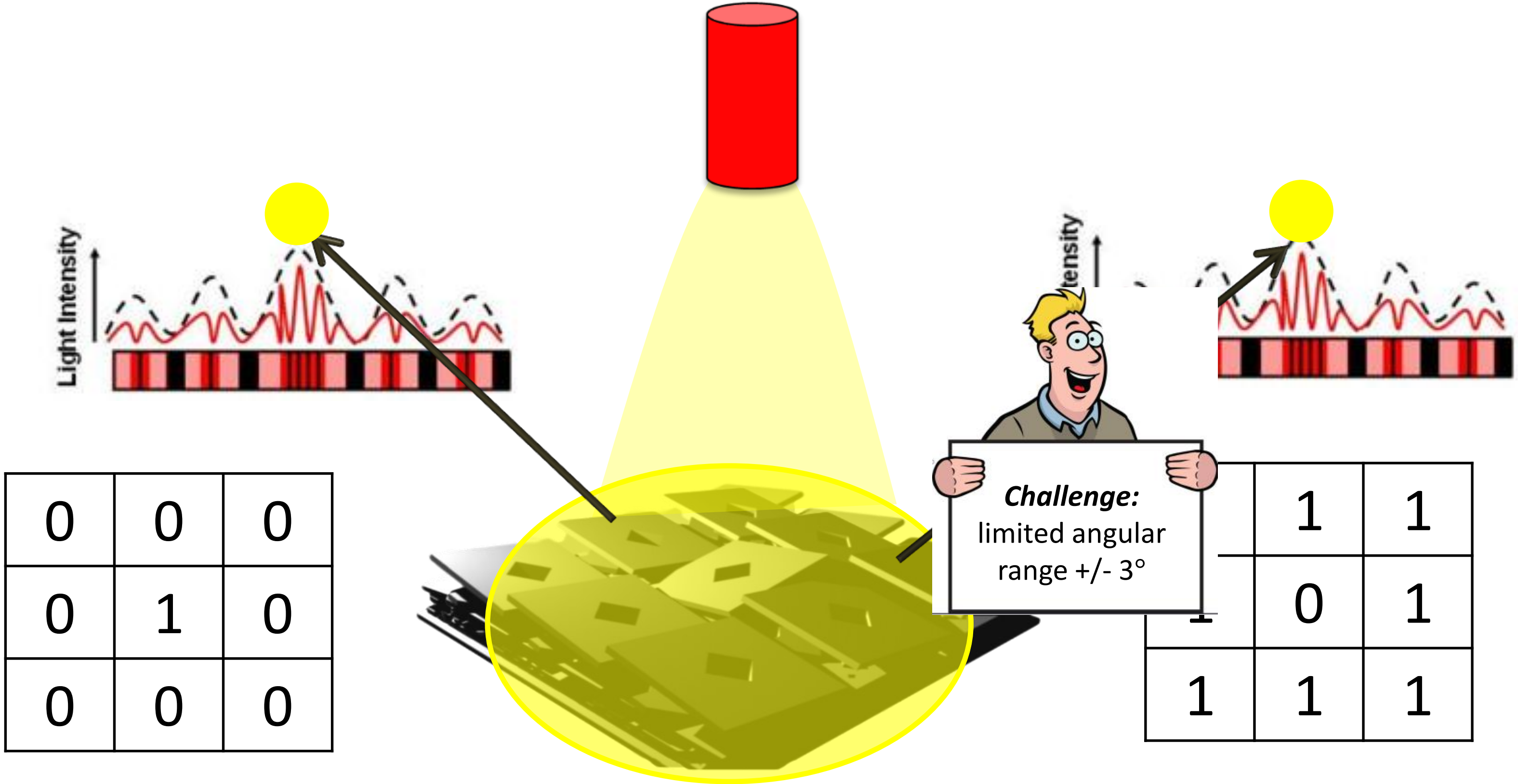
0	0	0
0	1	0
0	0	0





1	1	1
1	0	1
1	1	1

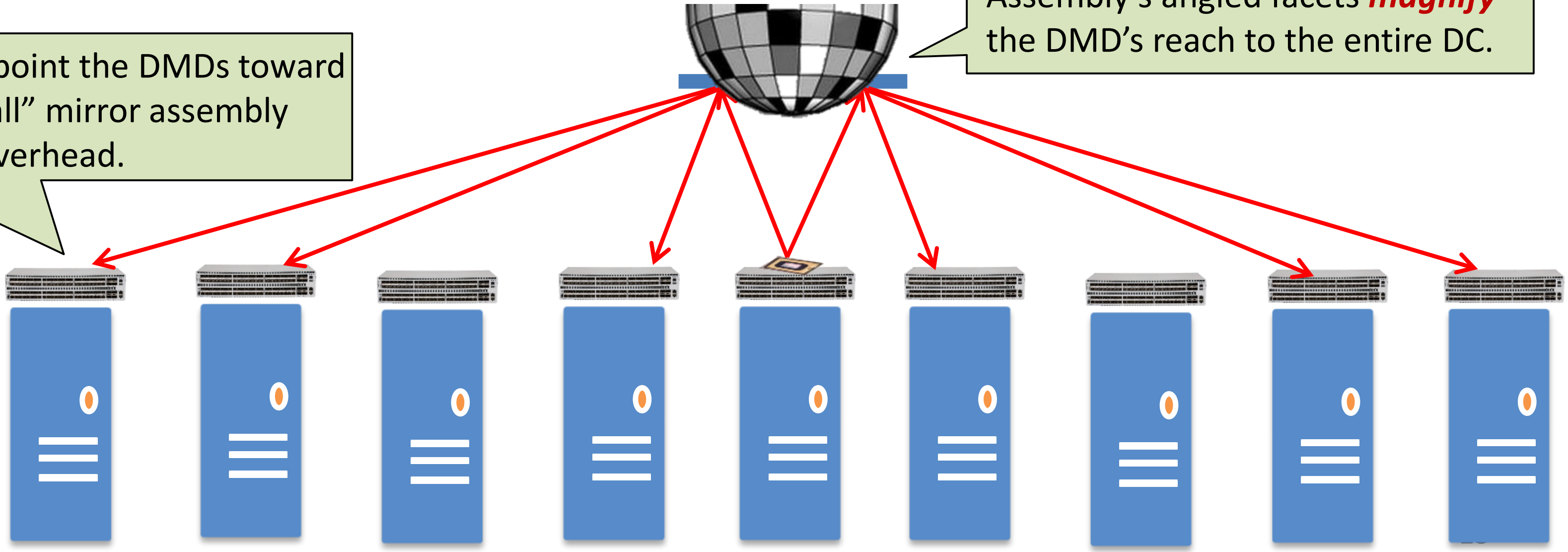
ProjecToR in More Details: DMDs to Redirect Light *Fast*



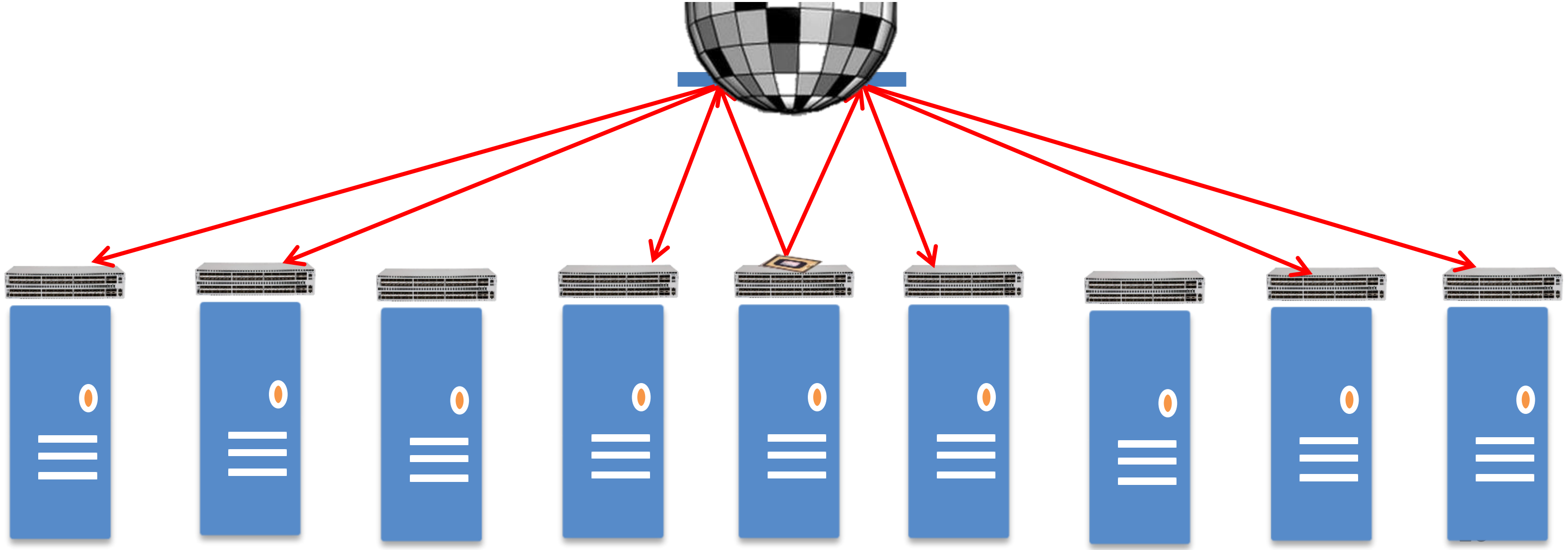
ProjecToR in More Details: Coupling DMDs with angled mirrors

Coupling: point the DMDs toward a “disco-ball” mirror assembly installed overhead.

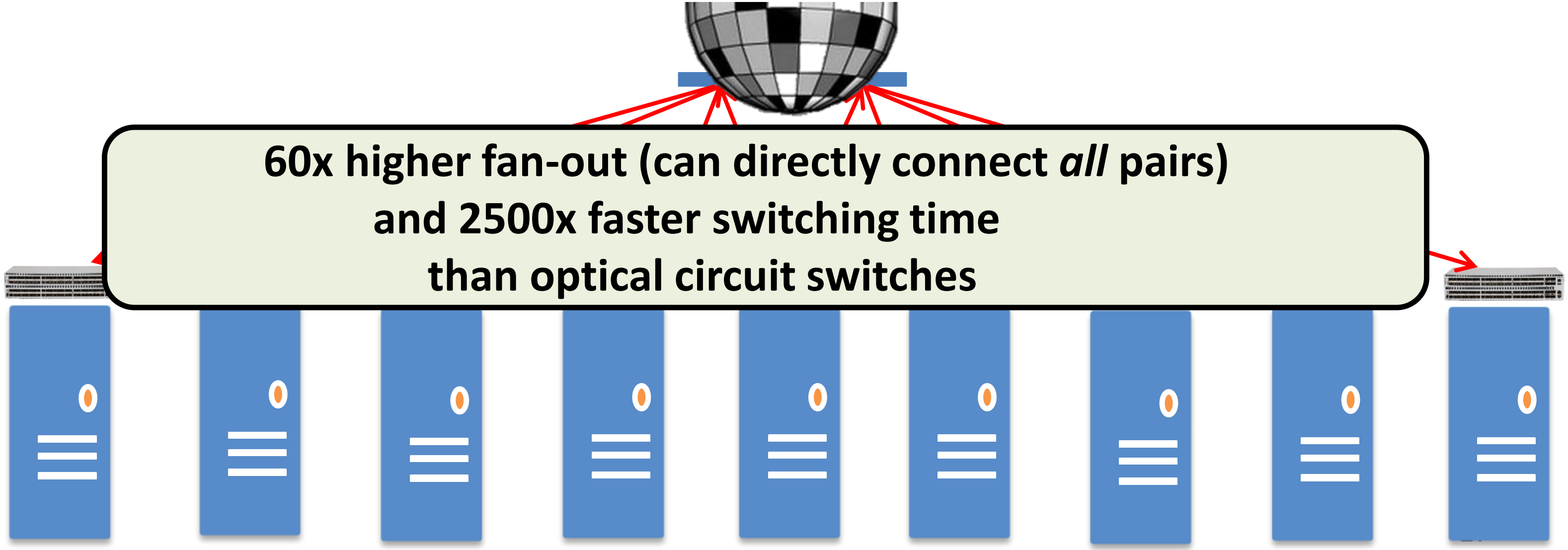
Assembly’s angled facets ***magnify*** the DMD’s reach to the entire DC.



ProjecToR in More Details: Coupling DMDs with angled mirrors

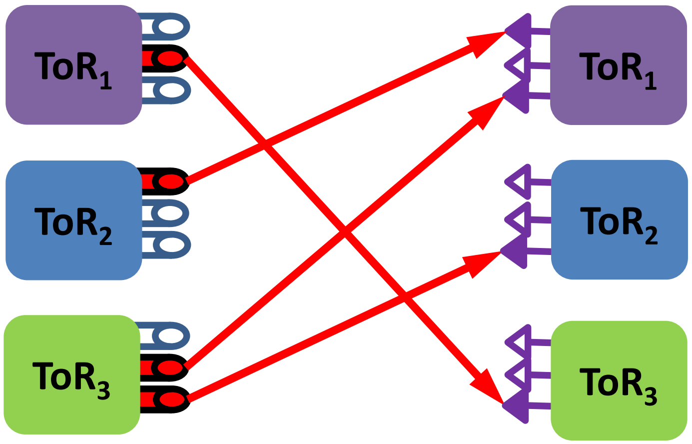


ProjecToR in More Details: Coupling DMDs with angled mirrors

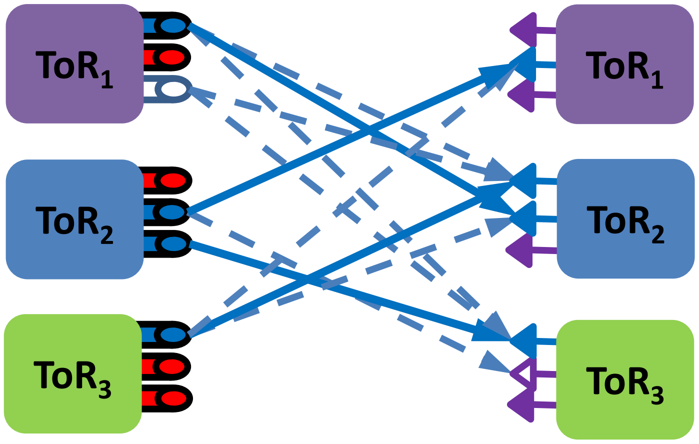


60x higher fan-out (can directly connect *all* pairs)
and 2500x faster switching time
than optical circuit switches

ProjecToR in More Details: 2-Topology Approach



+



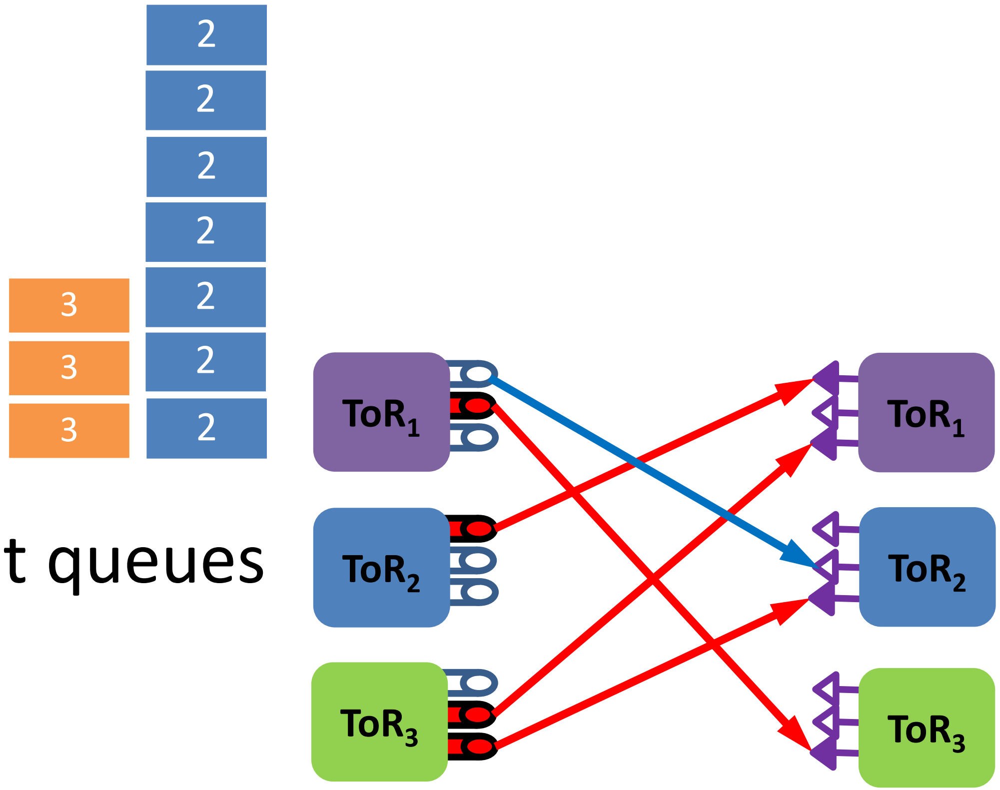
dedicated topology
(multihop, changes slowly)

k-shortest paths routing

opportunistic links
(singlehop, changes fast)

Decentralized stable matching

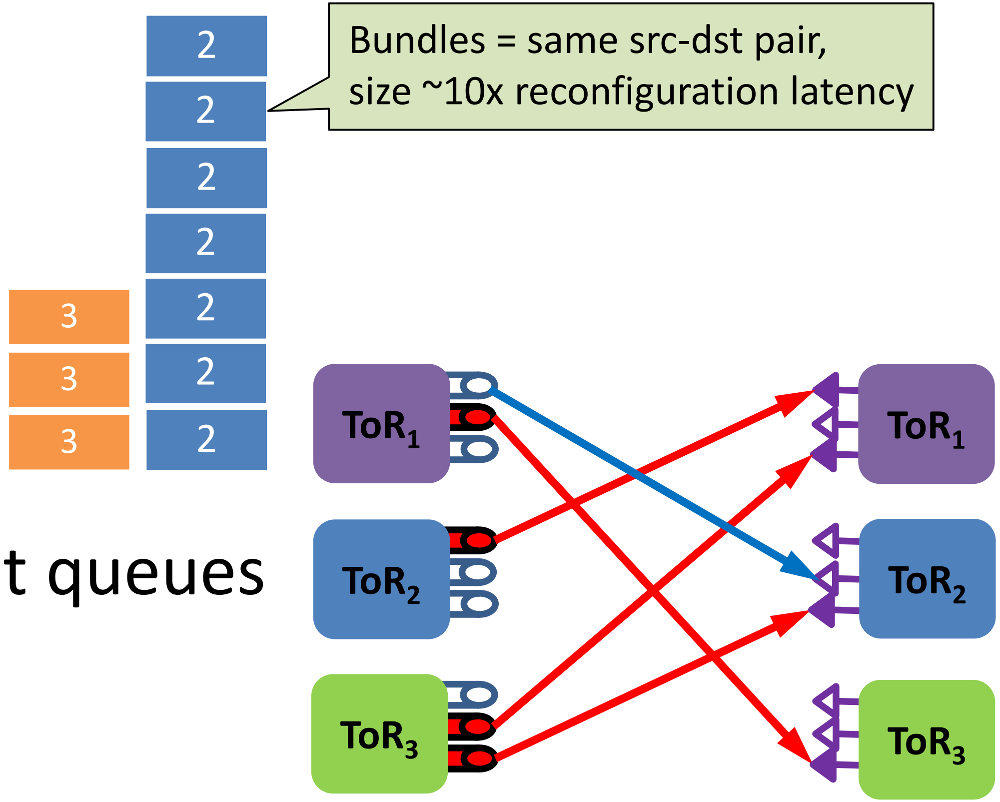
ProjecToR in More Details: Based on Virtual Output Queue Technique



Virtual output queues

ProjecToR in More Details: Based on Virtual Output Queue Technique

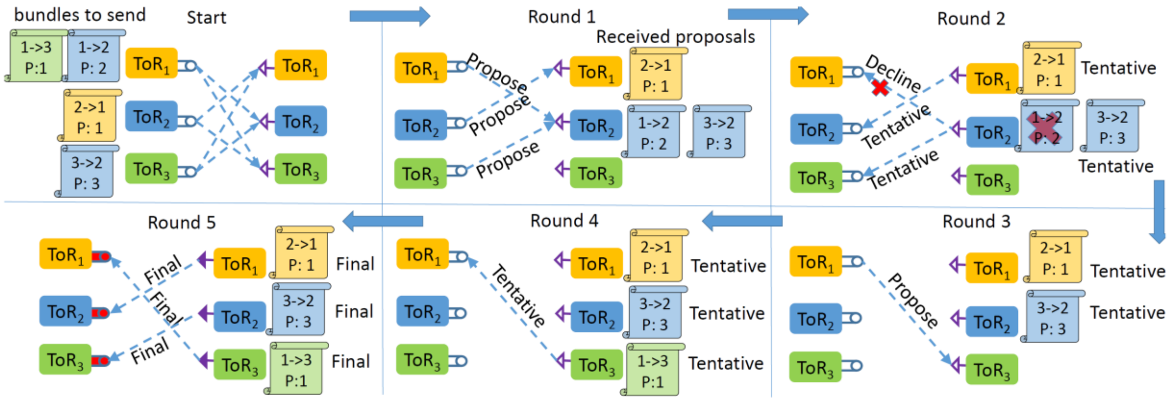
Bundles = same src-dst pair,
size ~10x reconfiguration latency



Virtual output queues

ProjecToR in More Details: Matching *with a Twist*

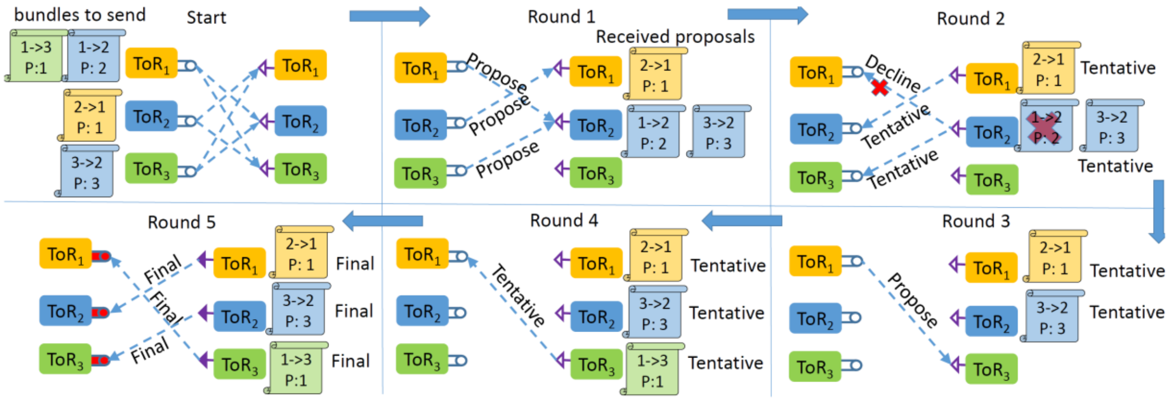
- Challenge: scheduling problem is **2-tiered**:
  - While ***traffic matrix*** is between ToRs
  - ***matching*** occurs between lasers and photodetectors: multiples of those per ToR!



ProjecToR in More Details: Matching *with a Twist*

- Challenge: scheduling problem is **2-tiered**:
  - While ***traffic matrix*** is between ToRs
  - sers and photodetectors: multiples of those per ToR!

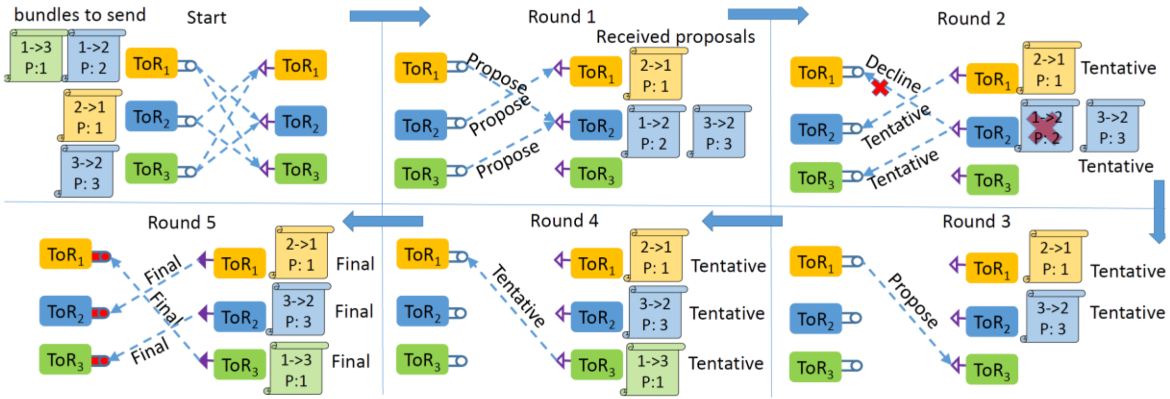
Each ToR has an ***interest map***: outstanding bundles and their priorities



ProjecToR in More Details: Matching *with a Twist*

- Challenge: scheduling problem is **2-tiered**:
  - While ***traffic matrix*** is between ToRs
  - ...tors: multiples of those per ToR!

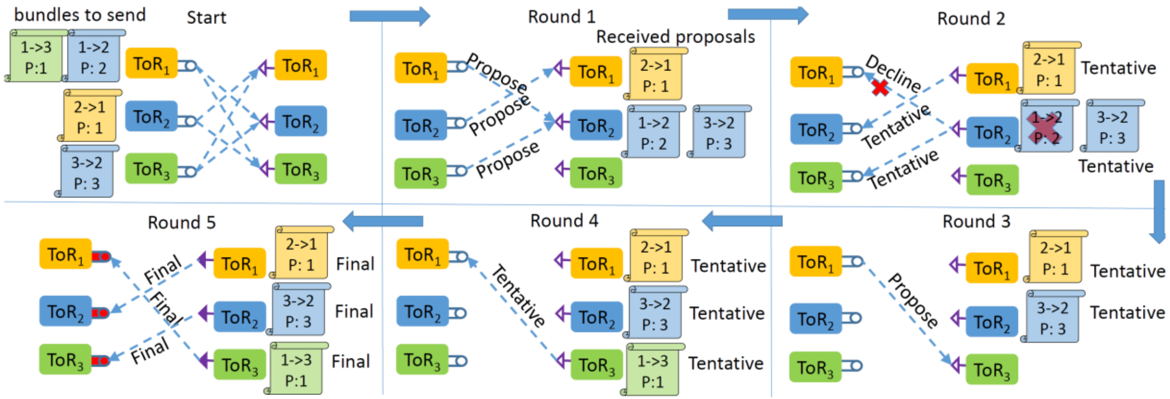
All ToRs ***send a proposal*** for their top priority bundle to the corresponding destination



ProjecToR in More Details: Matching *with a Twist*

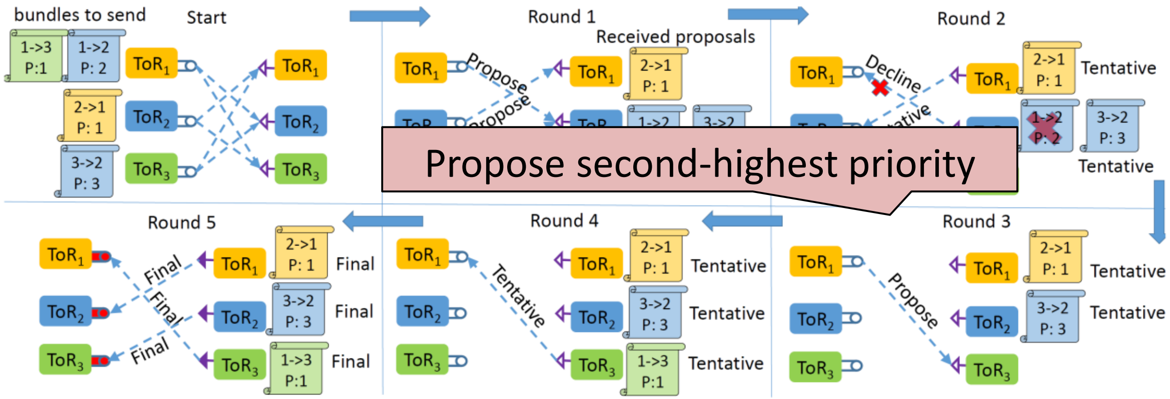
- Challenge: scheduling problem is **2-tiered**:
  - While ***traffic matrix*** is between ToRs
  - ***matching*** occurs between lasers and photodetectors: multiples of those per ToR!

Tentatively accept highest-priority proposal



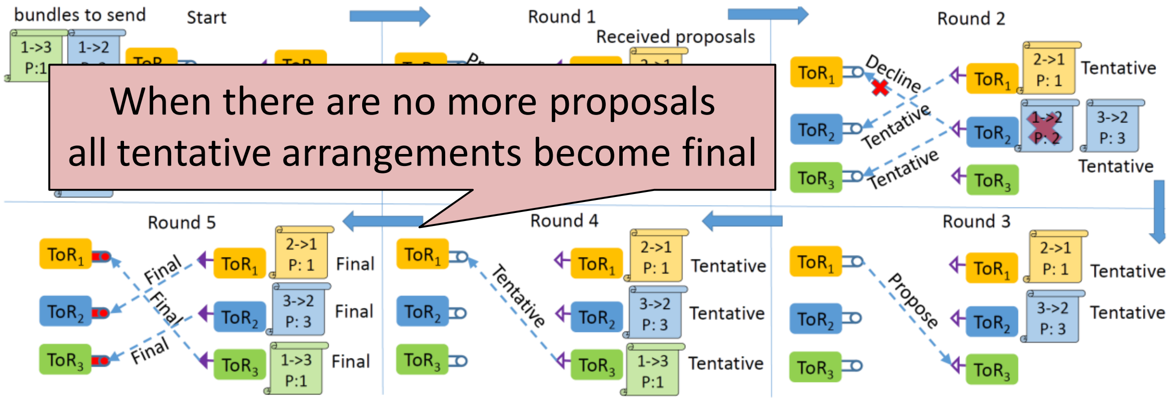
ProjecToR in More Details: Matching *with a Twist*

- Challenge: scheduling problem is **2-tiered**:
  - While ***traffic matrix*** is between ToRs
  - ***matching*** occurs between lasers and photodetectors: multiples of those per ToR!



ProjecToR in More Details: Matching *with a Twist*

- Challenge: scheduling problem is **2-tiered**:
  - While ***traffic matrix*** is between ToRs
  - ***matching*** occurs between lasers and photodetectors: multiples of those per ToR!



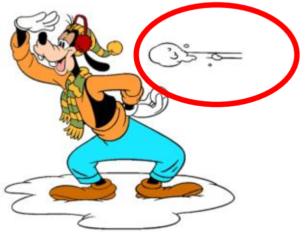
Such Fast Reconfigurability Enables Demand-Aware Networks (DANs)!

You: I invented a great new reconfigurable network which allows to self-adjust to the demand it serves!

Boss: Okay, so how much better is your demand-aware network really compared to demand-oblivious networks!?

You: hmm…

A Simple Answer

Demand-Oblivious Networks = 

The *SIGMETRICS* Answer

- It depends…

The *SIGMETRICS* Answer

As always in computer science! ☺

- It depends…
- … on the demand!

We need **metrics**!

Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures

Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures

A Simple Example

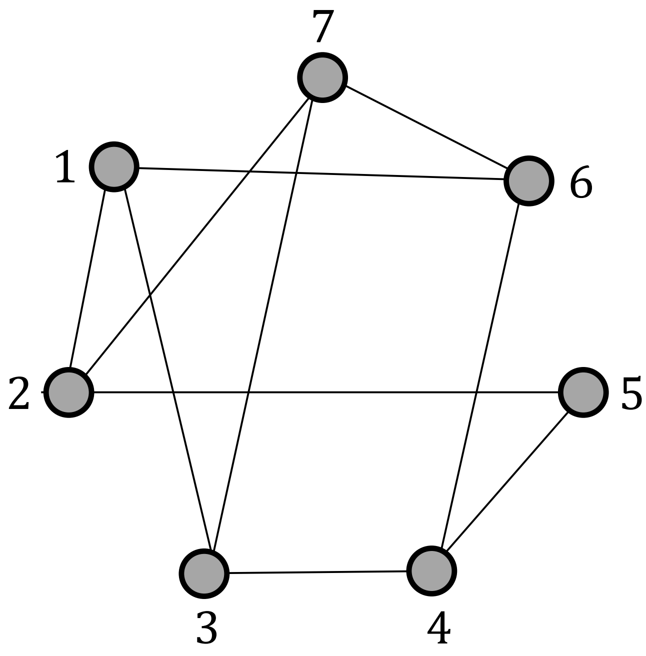
Input: Workload

Destinations

Sources	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

$\mathcal{D}$

Output: Constant-Degree DAN



N

Input: Workload

Destinations

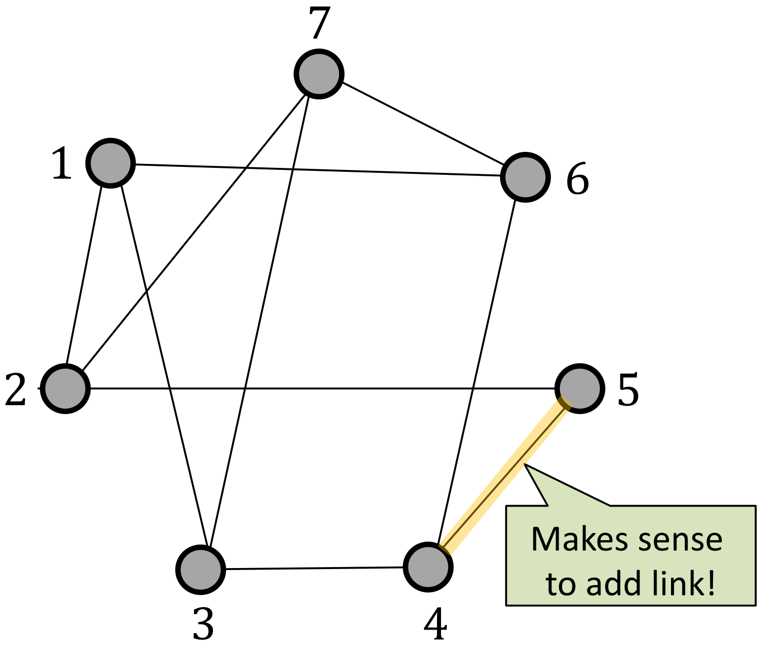
Sources

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	[illegible]	[illegible]	[illegible]
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Much from 4 to 5.

$\mathcal{D}$

Output: Constant-Degree DAN



N

Input: Workload

Output: Constant-Degree DAN

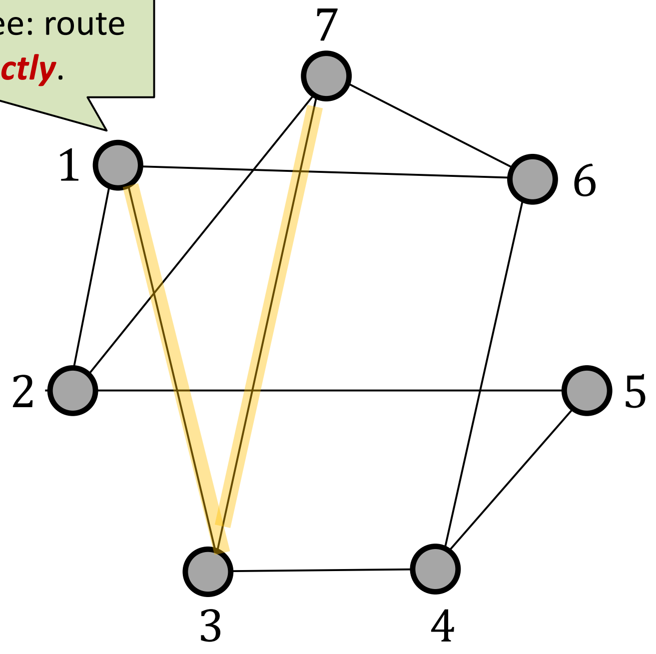
1 communicates to many.

Destinations

Sources	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

$\mathcal{D}$

Bounded degree: route to 7 ***indirectly***.



N

Input: Workload

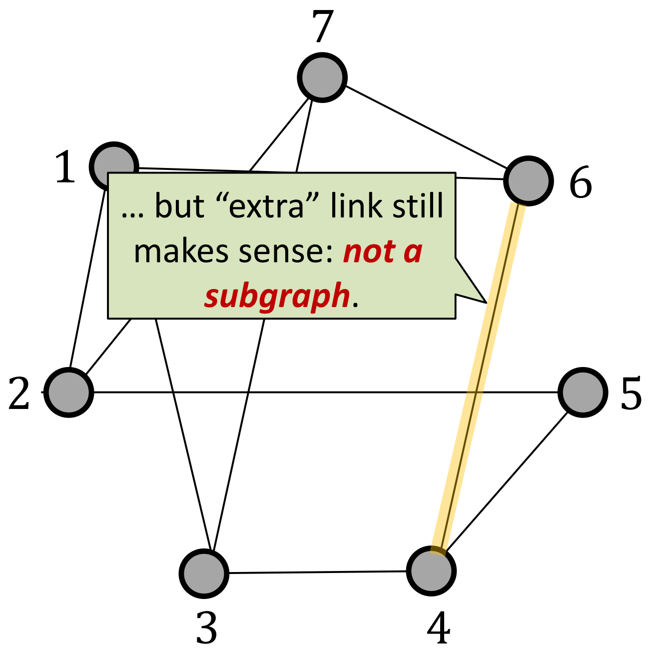
Destinations

Sources	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	[illegible]	[illegible]
6	$\frac{2}{65}$	0	0	0	0	0	[illegible] 65
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

4 and 6 don't communicate…

$\mathcal{D}$

Output: Constant-Degree DAN



N

Objective: Expected Route Length

DAN N of degree **Δ**

path length on *N*

$$\mathrm{ERL}(\mathcal{D},\mathrm{N}) = \sum_{(u,v)\in\mathcal{D}} p(u,v) \cdot d_N(u,v)$$

$\mathcal{D}[\mathbf{p(i,j)}]$: joint **distribution**

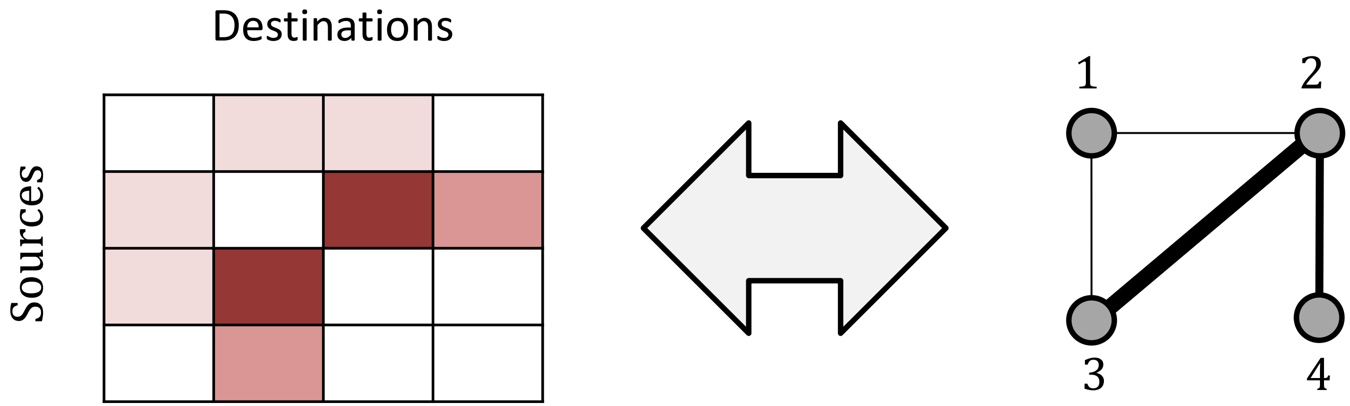
frequency

Remark

- Can represent demand matrix as a **demand graph**

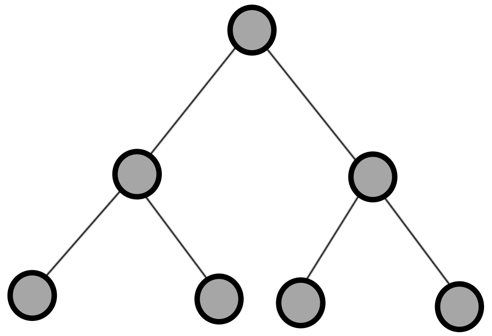
sparse distribution $\mathcal{D}$

sparse graph G($\mathcal{D}$)



Some Examples

- DANs of $\Delta = 3$:
  - E.g., complete binary **tree**
  - $d_N(u,v) \leq 2 \log n$
  - Can we do ***better*** than *log n*?

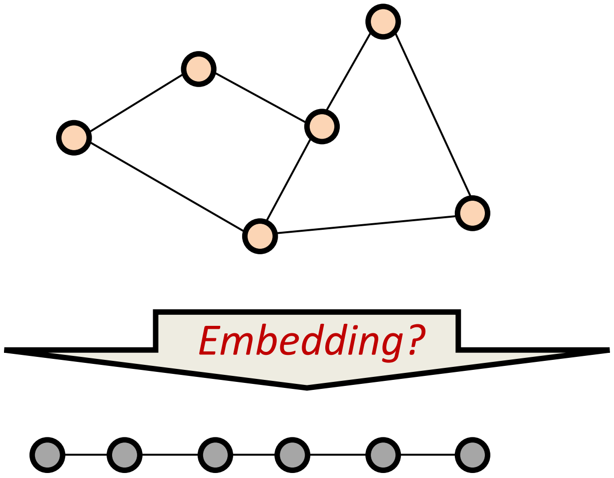


- DANs of $\Delta = 2$:
  - E.g., set of **lines** and **cycles**

Remark: Another Hardness Proof

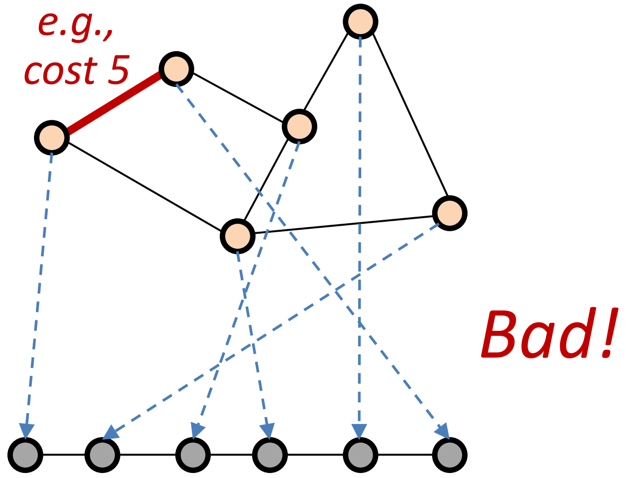
DAN design can be NP-hard

- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A "Virtual Network Embedding Problem", VNEP
  - *Minimize sum* of lengths of virtual edges



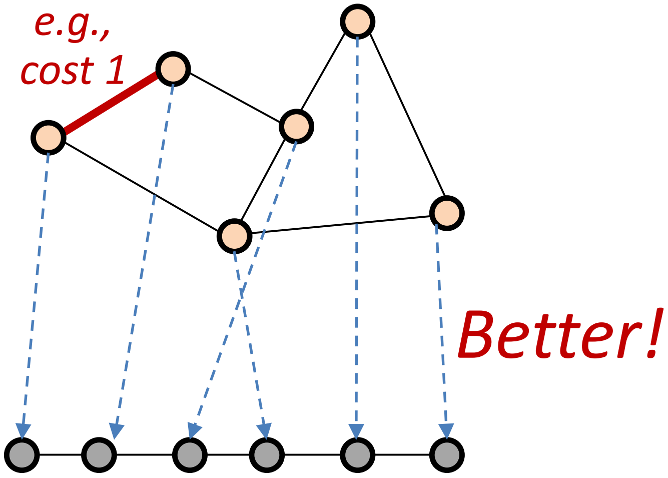
DAN design can be NP-hard

- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

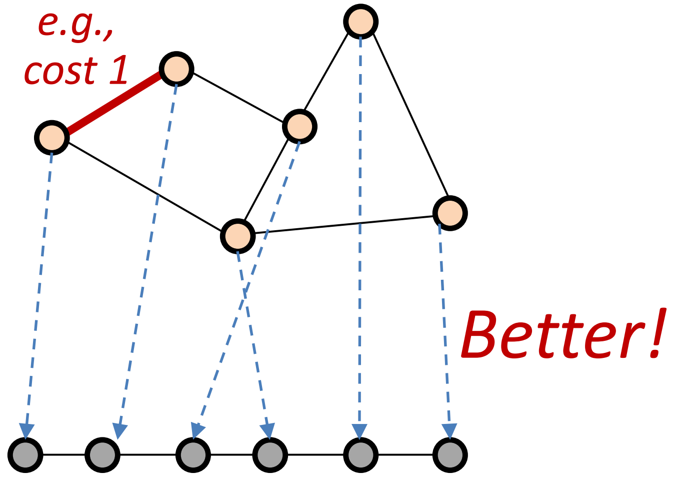
- **Example Δ = 2:** A Minimum Linear Arrangement (**MLA**) problem
  - A "Virtual Network Embedding Problem", VNEP
  - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

- **Example Δ = 2:** A Mini… inear Arrangement (MI… blem
  - A “Virtual N… bedding Problem”, VNEP
  - *Minir…* lengths of virtual edges

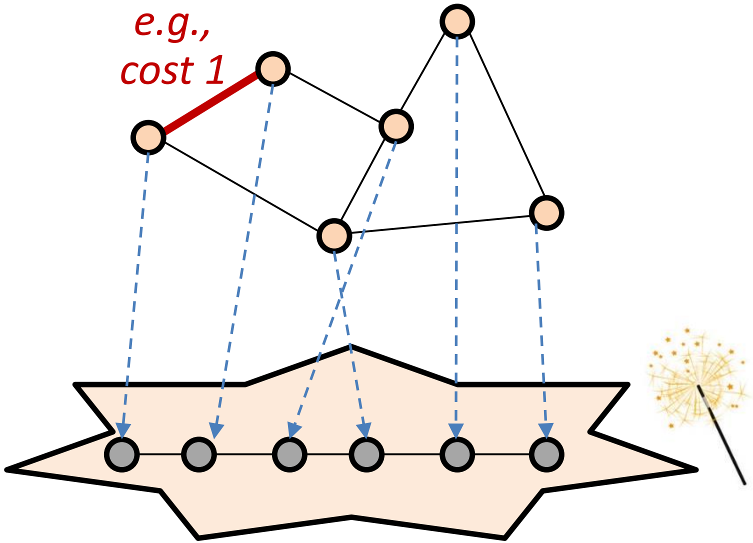
NP-hard, and so is DAN design.



DAN design can be NP-hard

- **Example Δ = 2:** A Mini[...]inear Arrangement (**MI**[...]blem
  - A "Virtual N[...]bedding Problem", VNEP
  - ***Minir***[...]lengths of virtual edges

NP-hard, and so is DAN design.



- But what about > 2? ***Embedding*** problem still hard, but we have an additional **degree of freedom**:

Do topological flexibilities make problem easier or harder?!

A new knob for optimization!

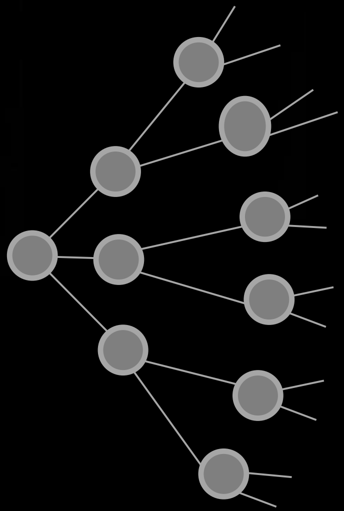
Rewinding the Clock: Degree-Diameter Tradeoff

Each network with n nodes and max degree $\Delta > 2$ must have a diameter of at least $\log(n)/\log(\Delta-1)-1$.

Example: constant Δ, $\log(n)$ diameter

Kudos to: Pedro Casas

Proof Idea



In k steps,
reach at most
$1 + \Sigma\, \Delta(\Delta - 1)^k$
nodes

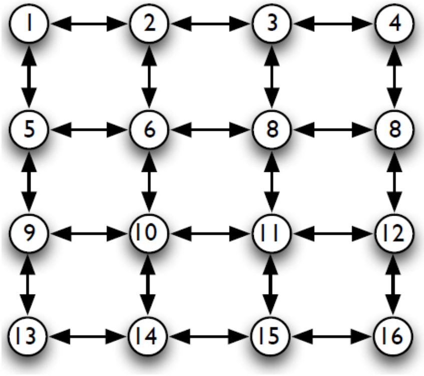
1 Δ Δ(Δ − 1) …

Kudos to: Pedro Casas

Is there a better tradeoff in DANs?

Sometimes, DANs can be much better!

Example 1: low-degree demand

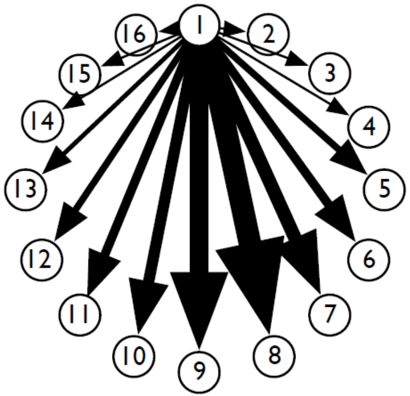


If **demand graph** is of degree Δ, it is trivial to design a **DAN** of degree Δ which achieves an ***expected route length of 1***.

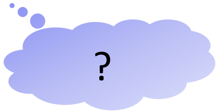
Just take DAN = demand graph!

Sometimes, DANs can be much better!

Example 2: skewed demand

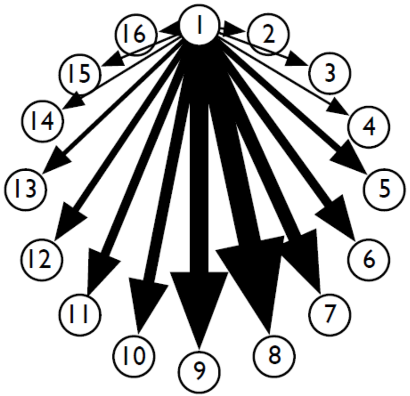


If **demand** is highly skewed, it is also possible to achieve an ***expected route length of 1*** in a constant-degree DAN.



Sometimes, DANs can be much better!

Example 2: skewed demand



If **demand** is highly skewed, it is also possible to achieve an ***expected route length of 1*** in a constant-degree DAN.

E.g., arrange neighbors of node 1 in a **Huffman** tree!

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. Chen Avin and Stefan Schmid. ACM **SIGCOMM CCR**, October 2018

So on what does it depend?

So on what does it depend?

We argue (but still don‘t know!): on the **“entropy” of the demand**!

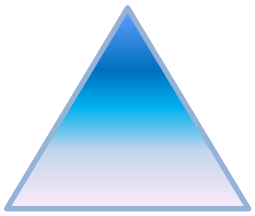
Intuition: Entropy Lower Bound

Lower Bound Idea: Leverage Coding or Datastructure

Destinations

Sources	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

- DAN just for a ***single (source) node 3***



- How good can this tree be? Cannot do better than Δ-ary **Huffman tree** for its destinations
- **Entropy** lower bound on ERL known for binary trees, e.g. ***Mehlhorn*** 1975

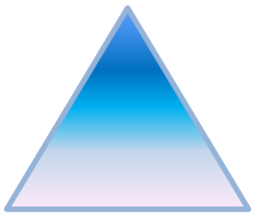
Lower Bound Idea: Leverage Coding or Datastructure

An optimal "**ego-tree**" for this source!

Destinations

Sources	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

- DAN just for a ***single (source) node 3***



- How good can this tree be? Cannot do better than Δ-ary **Huffman tree** for its destinations
- **Entropy** lower bound on ERL known for binary trees, e.g. ***Mehlhorn*** 1975

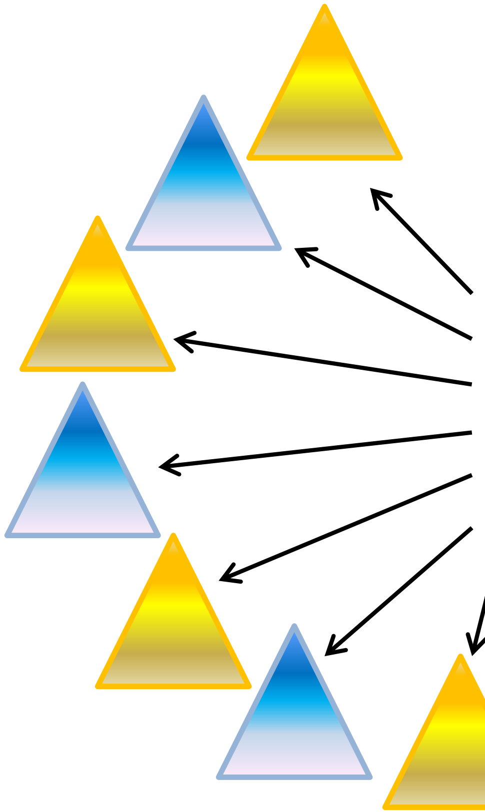
So: Entropy of the *Entire* Demand

sources destinations

- **Proof idea** (EPL=$\Omega(H_\Delta(Y|X))$):

entropy degree

- Compute **ego-tree** for each source node
- Take ***union*** of all **ego-trees**
- Violates ***degree restriction*** but valid lower bound



	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Entropy of the *Entire* Demand: Sources *and* Destinations

Do this in **both dimensions**:

EPL $\geq \Omega(\max\{H_\Delta(Y|X), H_\Delta(X|Y)\})$

$\Omega(H_\Delta(X|Y))$

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

$\Omega(H_\Delta(Y|X))$

$\mathcal{D}$

Entropy of the *Entire* Demand: Sources *and* Destinations

Do this in **both dimensions**:

EPL ≥ $\Omega(\max\{H_\Delta(Y|X), H_\Delta(X|Y)\})$

$\Omega(H_\Delta(X|Y))$

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

$\Omega(H_\Delta(Y|X))$

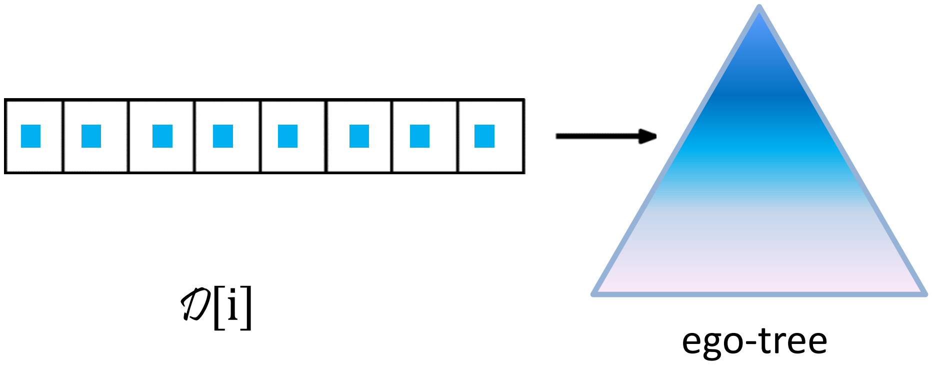
$\mathcal{D}$

Demand-Aware Network Designs of Bounded Degree. Chen Avin, Kaushik Mondal, and Stefan Schmid. **DISC**, 2017.

Achieving Entropy Limit: Algorithms

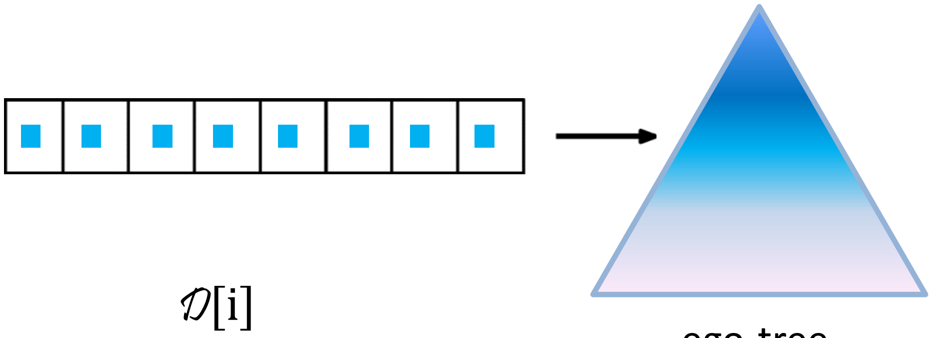
Ego-Trees Revisited

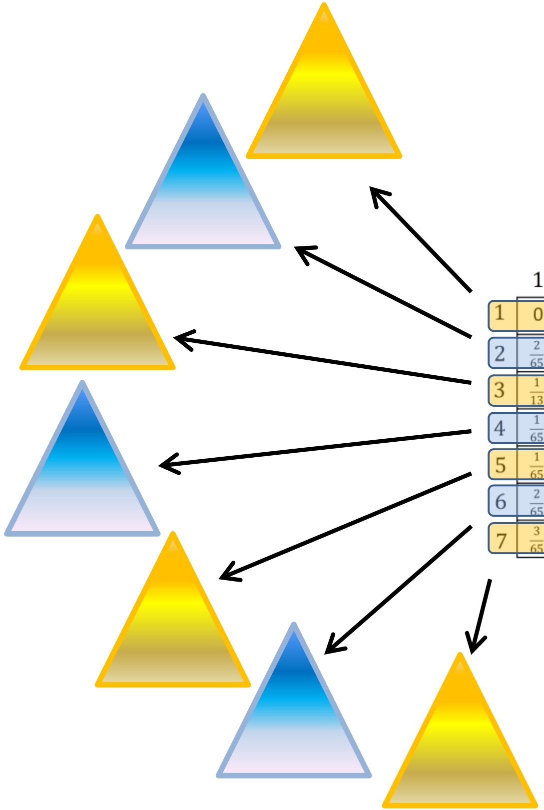
- ego-tree: optimal tree for a row (= given source)



Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



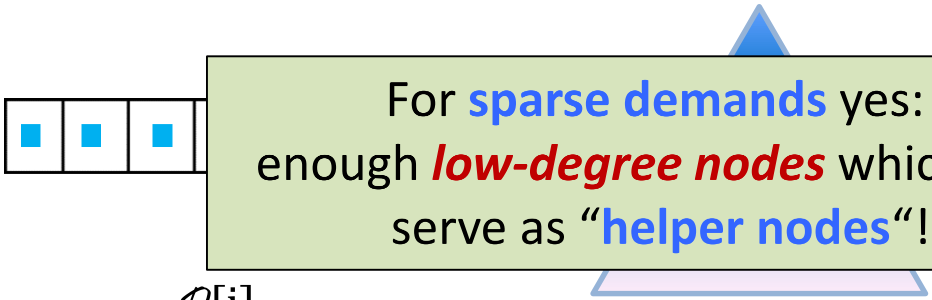


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

Can we merge the trees ***without distortion*** and ***keep degree low***?

Ego-Trees Revisited

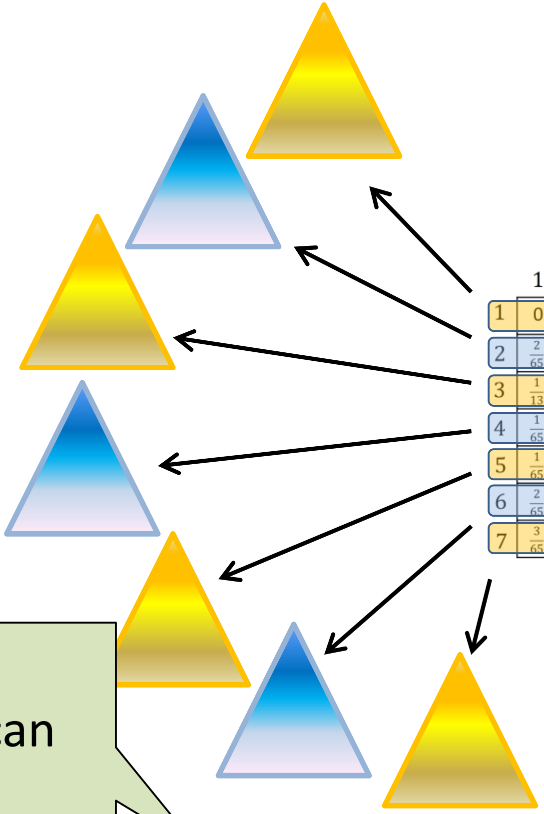
- ego-tree: optimal tree for a row (= given source)



For **sparse demands** yes:
enough ***low-degree nodes*** which can serve as “**helper nodes**“!

$\mathcal{D}[i]$

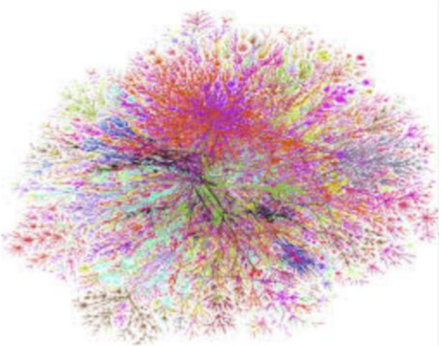
ego-tree



	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

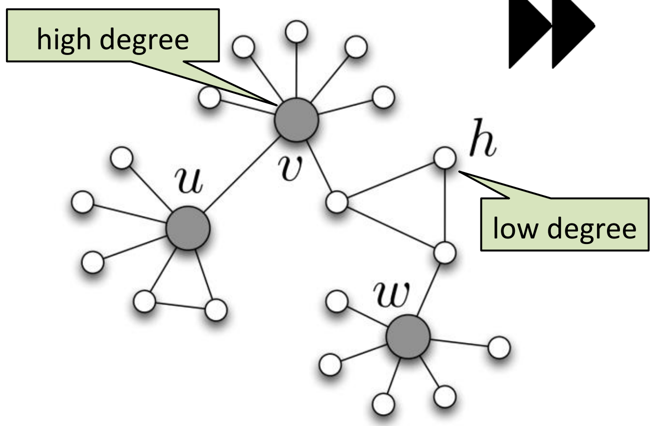
Can we merge the trees ***without distortion*** and ***keep degree low***?

From Trees to Networks

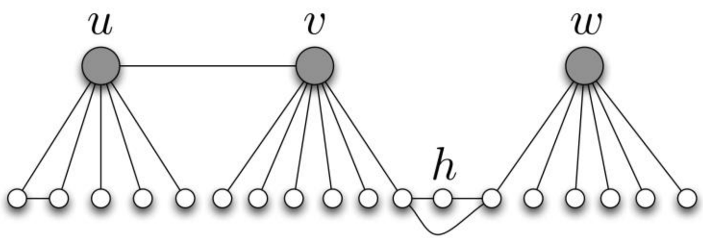


Idea: Degree Reduction

1 **Demand graph**

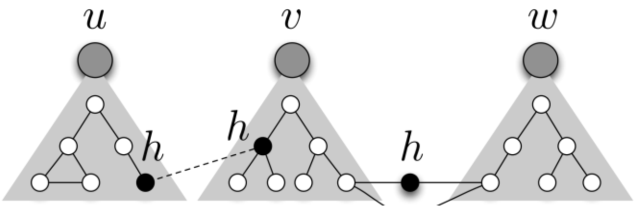


2 **Hierarchical representation**



Taking union of ego-trees results in ***high degree***:
u and *v* will appear in many ego-trees

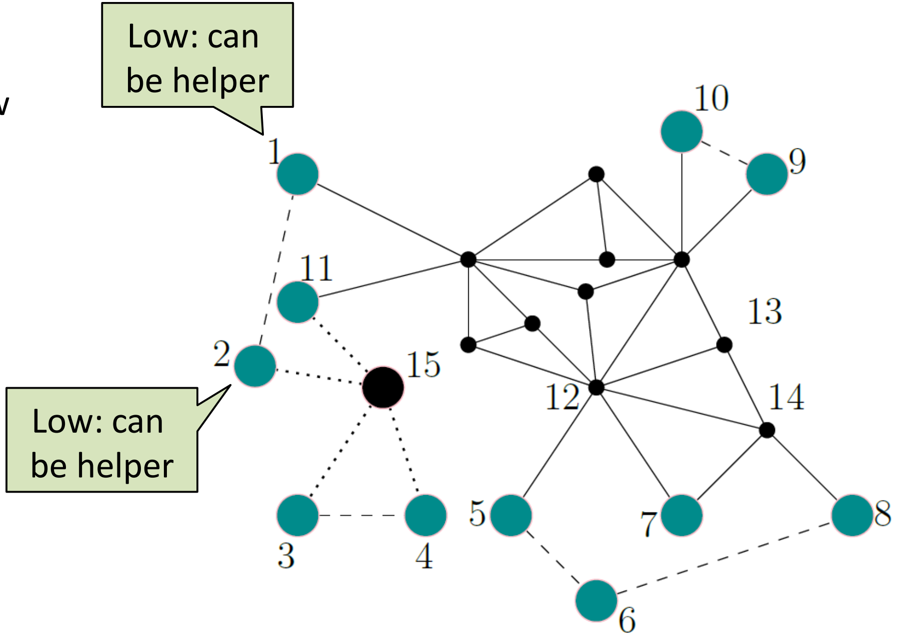
3 **Add low-degree nodes as helpers**



Node *h* ***helps edge (u, v)*** by participating in *ego-tree(u)* as a relay node toward *v* and *in ego-tree(v)* as a relay toward *u*

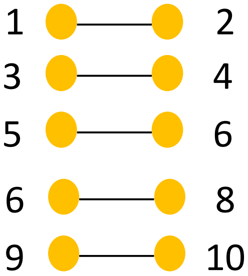
Algorithm: Degree Reduction

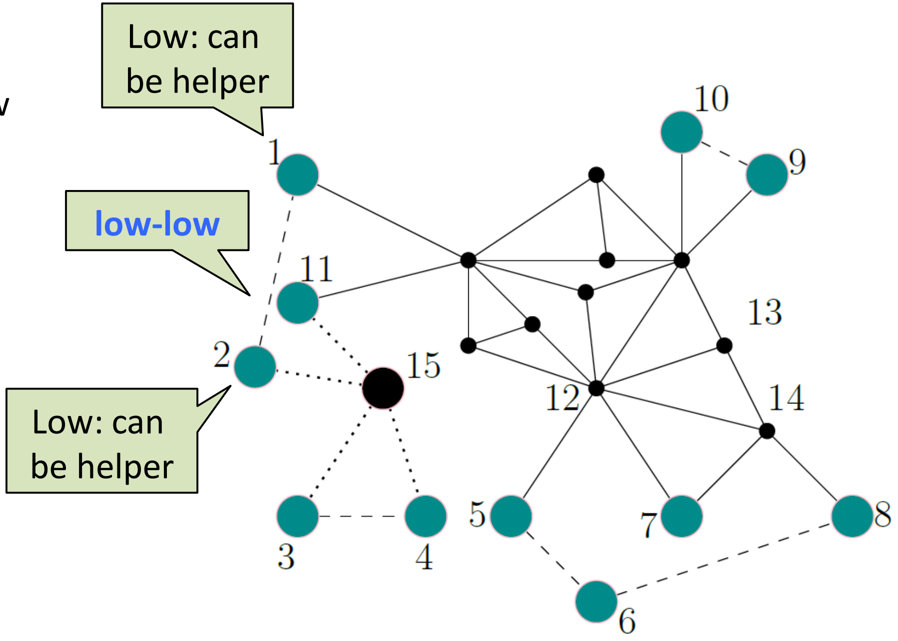
- Find low degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”



Algorithm: Degree Reduction

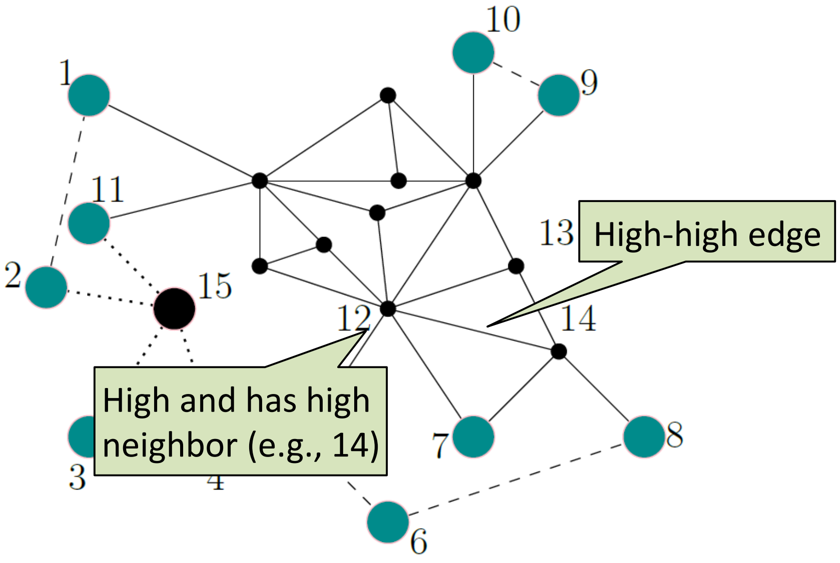
- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges into DAN and remove from demand





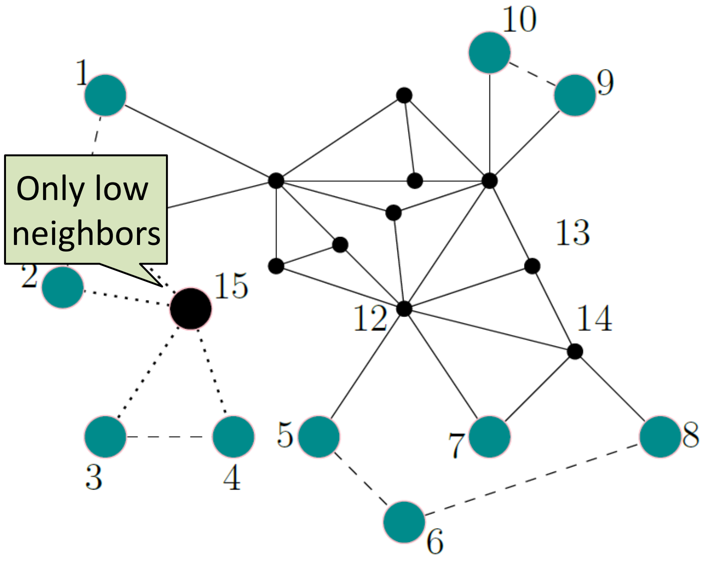
Algorithm: Degree Reduction

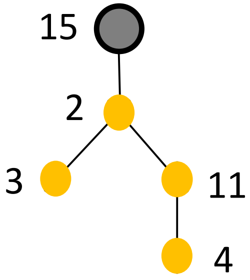
- Find low degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges into DAN and remove from demand
- Mark **high-high** edges
  - Put (any) **low degree** nodes in between (e.g., 1 or 2): one is enough so distance increased by **+1**



Algorithm: Degree Reduction

- Find low degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges into DAN and remove from demand
- Mark **high-high** edges
  - Put (any) **low degree** nodes in between (e.g., 1 or 2): one is enough so distance increased by **+1**
- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors



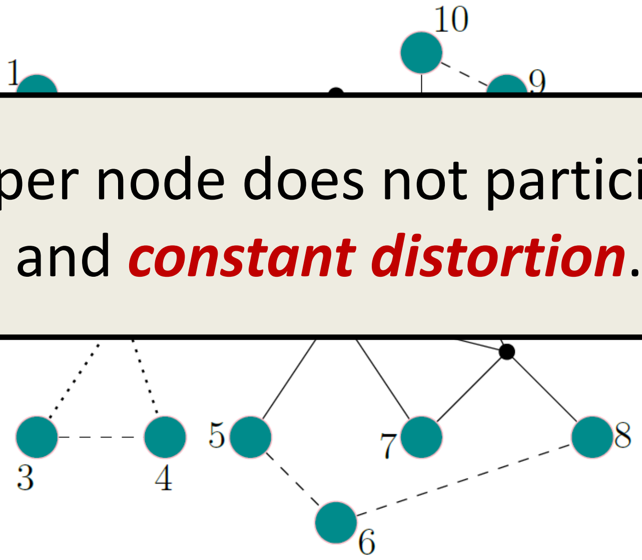


Algorithm: Degree Reduction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”

Theorem [Asymptotic Optimality]: Helper node does not participate in many trees, so ***constant degree***, and ***constant distortion***.

- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors

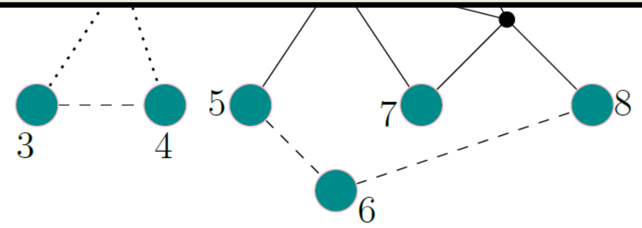


Algorithm: Degree Reduction

- Find **low** degree nodes
  – Half of the nodes of lowest degree: “below twice average degree”

Theorem [Asymptotic Optimality]: Helper node does not participate in many trees, so ***constant degree***, and ***constant distortion***.

- Now high degree nodes have only low degree neighbors: make **tree**
  – Create optimal **binary tree** with low degree neighbors



Demand-Aware Network Designs of Bounded Degree. Chen Avin, Kaushik Mondal, and Stefan Schmid. **DISC**, 2017.

DAN Design: Related to Spanners

Low-Distortion Spanners

- Classic problem: find sparse, distance-preserving (low-distortion) **spanner** of a graph

The „DAN“

The demand graph

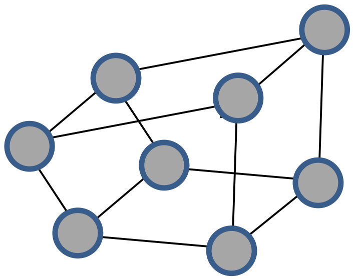
Low-Distortion Spanners

- Classic problem: find sparse, distance-preserving (low-distortion) **spanner** of a graph
- But:
  - Spanners aim at low distortion among ***all pairs***; in our case, we are only interested in the ***local distortion***, 1-hop communication neighbors
  - We allow **auxiliary edges** (not a subgraph): similar to **geometric spanners**
  - We require ***constant degree***

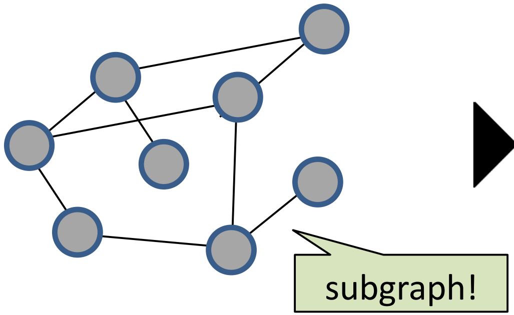
Yet: We can leverage the connection to spanners sometimes!

Theorem: If request distribution $\mathcal{D}$ is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant**, **sparse**) spanner for this request graph: then we can design a constant degree DAN providing an ***optimal ERL*** (i.e., O(H(X|Y)+H(Y|X)).

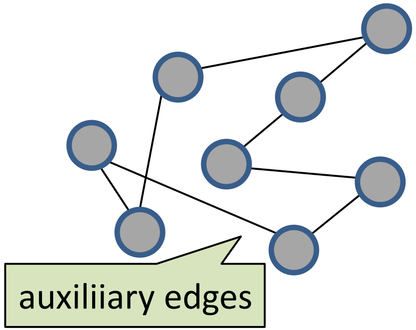
r-regular *and* ***uniform*** **demand***:*



Sparse, ***irregular (constant)*** **spanner***:*



Constant *degree* ***optimal*** **DAN** (ERL at most ***log r***):



Yet: We can leverage the connection to spanners sometimes!

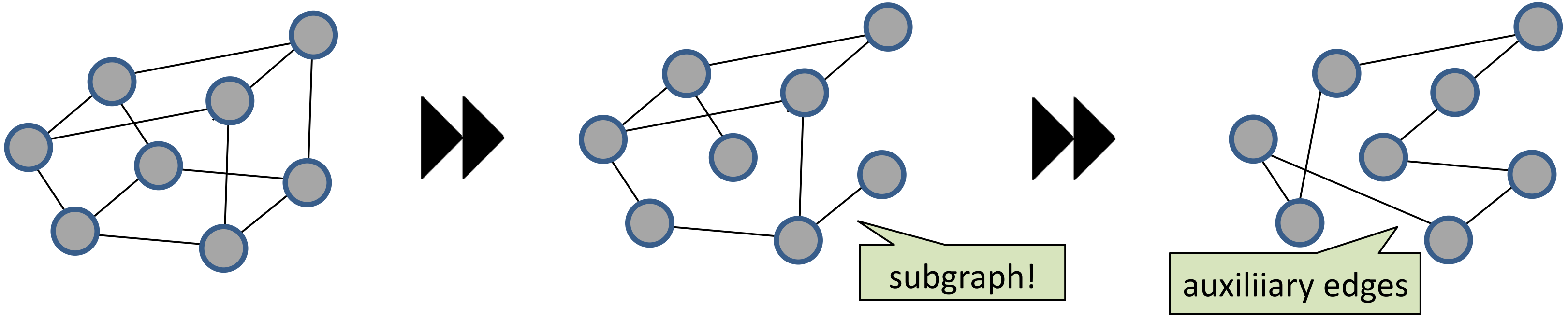
Theorem: If request distribution $\mathcal{D}$ is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant, sparse**) spanner for this request graph: then we can design a constant degree DAN p … |X)).

Simply using our degree reduction trick again: now for spanner!

r-regular *and* ***uniform*** **demand***:*

Sparse, ***irregular*** ***(constant)*** **spanner***:*

Constant *degree* ***optimal*** **DAN** (ERL at most ***log r***):



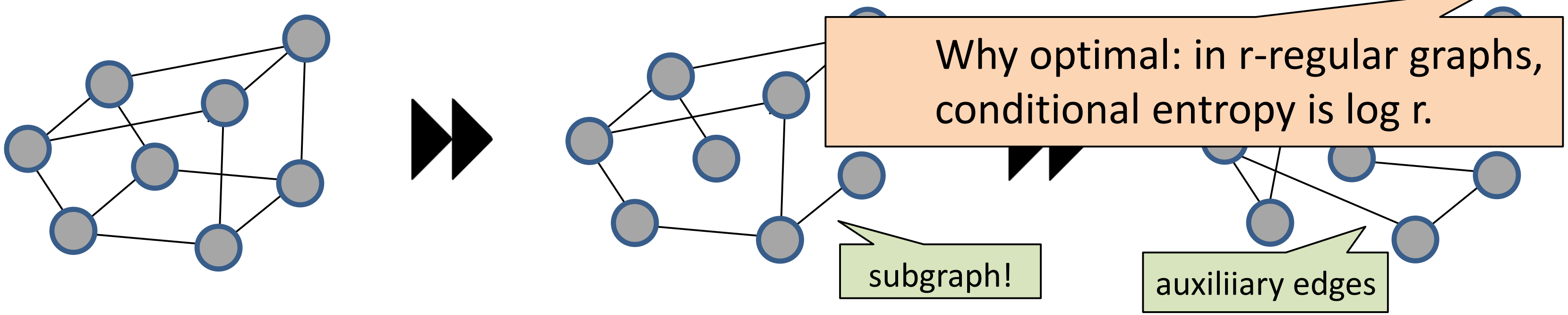
Yet: We can leverage the connection to spanners sometimes!

Theorem: If request distribution $\mathcal{D}$ is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant**, **sparse**) spanner for this request graph: then we can design a constant degree DAN providing an ***optimal EPL*** (i.e., O(H(X|Y)+H(Y|X)).

r-regular *and* ***uniform*** **demand***:*

Sparse, ***irregular*** ***(constant)*** **spanner***:*

Constant *degree* ***optimal*** **DAN** (ERL at most ***log r***):



Proof Idea

- **Degree reduction** again, this time ***from sparse spanner*** (before: from sparse demand graph)

Corollaries

- Optimal DAN designs for

  Has sparse 3-spanner.

  - **Hypercubes** (with n log n edges)
  - **Chordal** graphs

    Has sparse O(1)-spanner.
  - Trivial: graphs with polynomial degree (**dense graphs**)
  - Graphs of **locally bounded doubling dimension**

We also know some more algos, e.g., for BSTs.

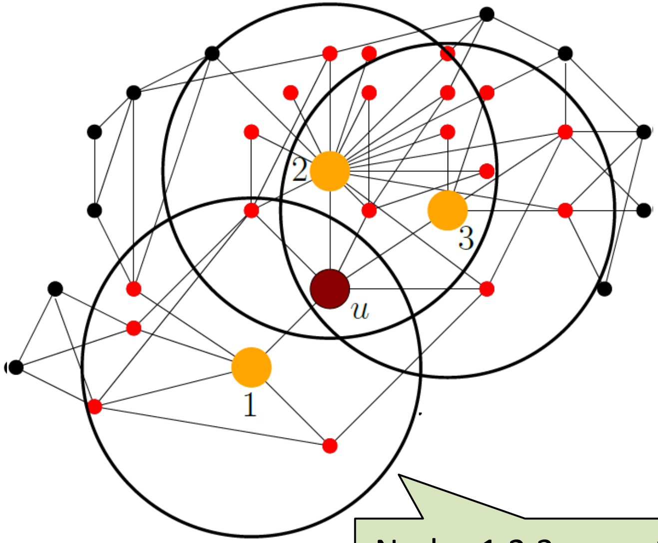
An Example: Demands of Locally-Bounded Doubling Dimension

- **LDD**: $G_{\mathcal{D}}$ has a **Locally-bounded Doubling Dimension** (LDD) iff all 2-hop neighbors are covered by 1-hop neighbors of just ***λ*** nodes
  - Note: care only about ***2-neighborhood***

We only consider 2 hops!

- Formally, $B(u, 2) \subseteq \bigcup_{i=1}^{\lambda} B(v_i, 1)$

- Challenge: can be of ***high degree***!



Nodes 1,2,3 cover 2-hop neighborhood of u.

DAN for Locally-Bounded Doubling Dimension

Lemma: There exists a sparse 9-(subgraph)spanner for LDD.

This ***implies optimal DAN***: still focus on regular and uniform!

Def. (ε-net): A subset V' of V is a **ε-net** for a graph $G = (V,E)$ if

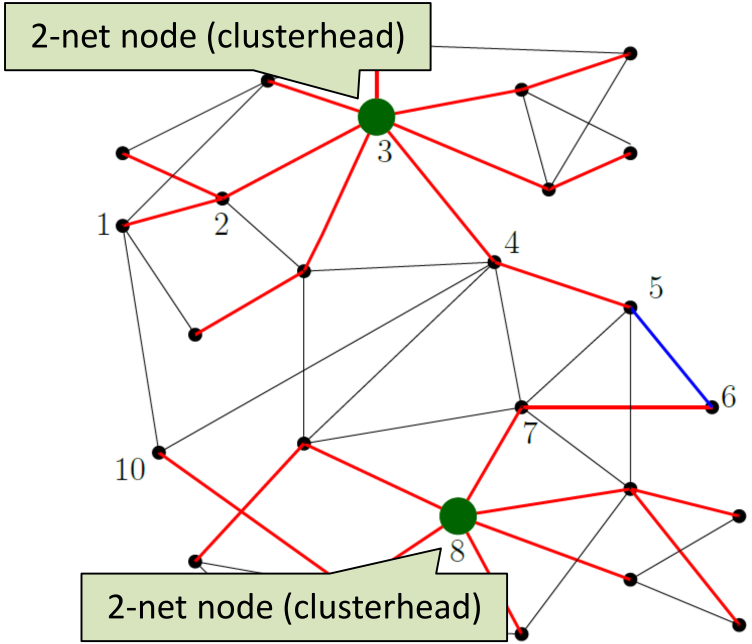
- V' sufficiently "independent": for every $u, v \in V'$, $d_G(u, v) > \boldsymbol{\varepsilon}$
- "dominating" V: for each $w \in V$, $\exists$ at least one $u \in V'$ such that, $d_G(u,w) \leq \boldsymbol{\varepsilon}$

9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

Easy: Select nodes into 2-net **one-by-one** in decreasing (remaining) degrees, **remove 2-neighborhood**. Iterate.



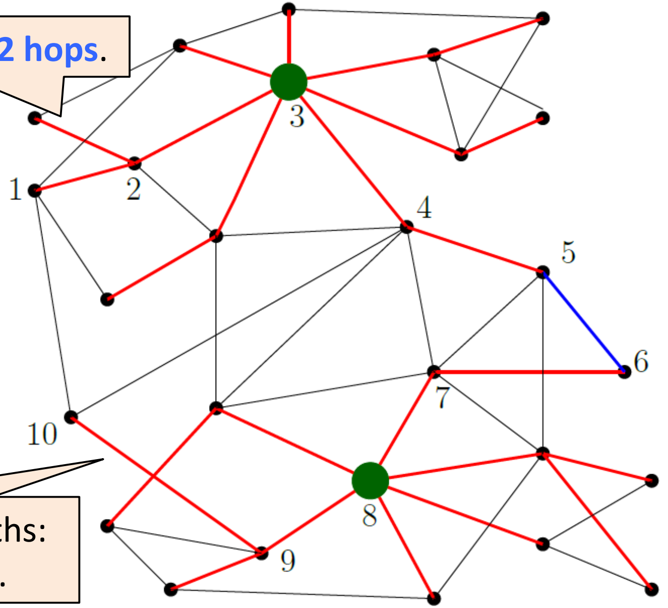
9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

2. Assign nodes to one of the closest 2-net nodes: tree

Assign: at most **2 hops**.



Union of these shortest paths: a **forest**. Add to **spanner**.

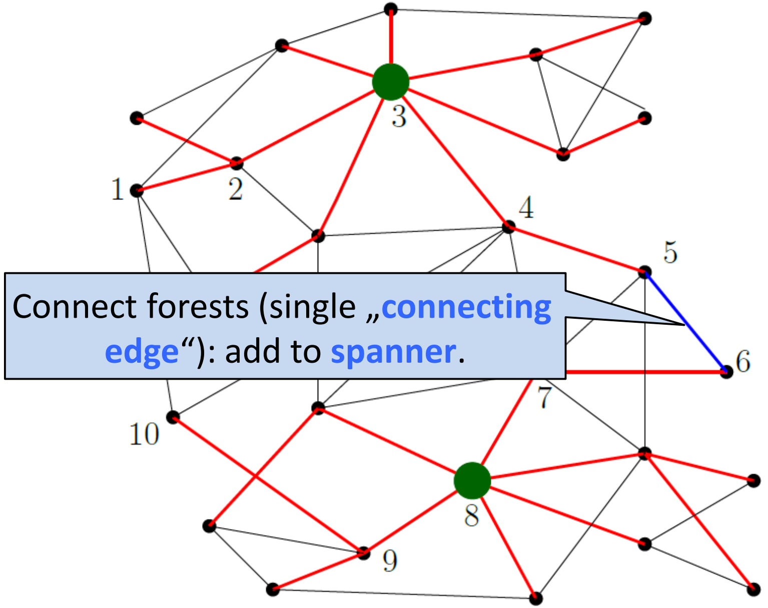
9-Spanner for LDD (= optimal DAN)

Simple algorithm:

1. Find a 2-net

2. Assign nodes to one of the closest 2-net nodes: tree

3. Join two clusters if there are edges in between



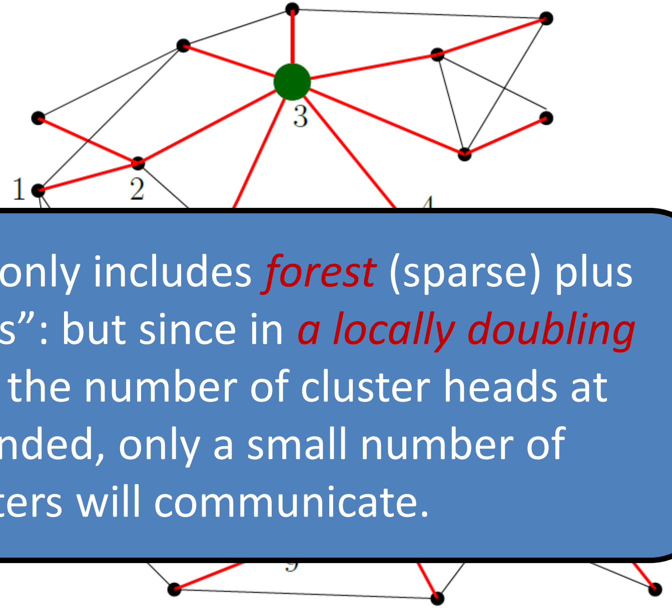
9-Spanner for LDD (= optimal DAN)

Distortion 9: *Short detour* via clusterheads: u,ch(u),x,y,ch(v),v

1. Find a 2-net

2. Assign nodes to one of the closest 2-net node

3. Join two clusters edges in between



Sparse: Spanner only includes *forest* (sparse) plus "connecting edges": but since in *a locally doubling dimension graph* the number of cluster heads at distance 5 is bounded, only a small number of neighboring clusters will communicate.

So: How *much* structure/entropy is there?

How to ***measure*** it?
And which ***types of structures***? E.g., **temporal** structure in addition to **non-temporal** structure?
More ***tricky***!

Often only intuitions in the literature…

“less than 1% of the rack pairs account for 80% of the total traffic”

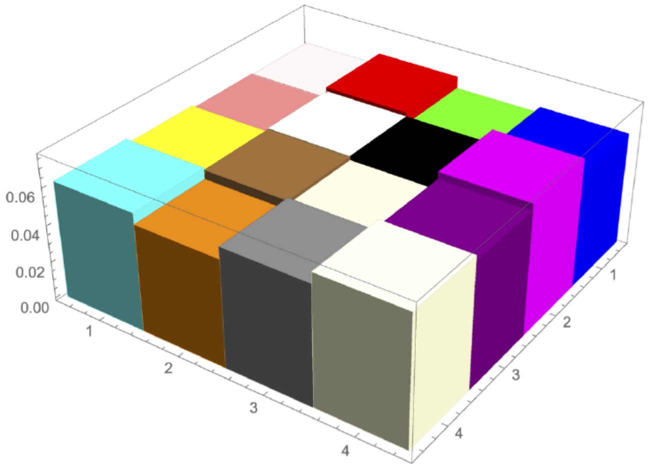
“only a few ToRs switches are hot and most of their traffic goes to a few other ToRs”

“over 90% bytes flow in elephant flows”

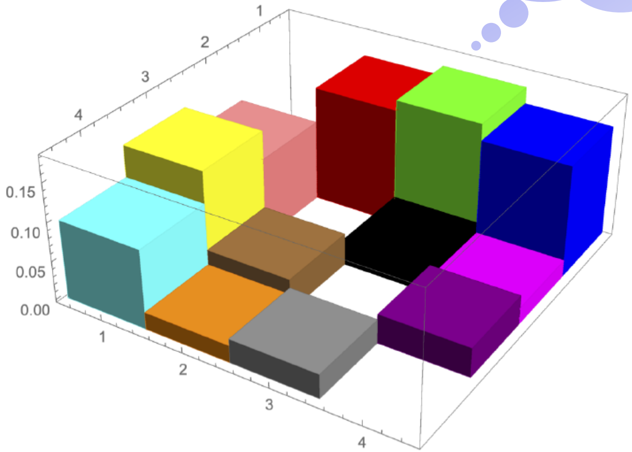
… and it *is* intuitive!

Non-temporal Structure

Color = comm. pair



VS



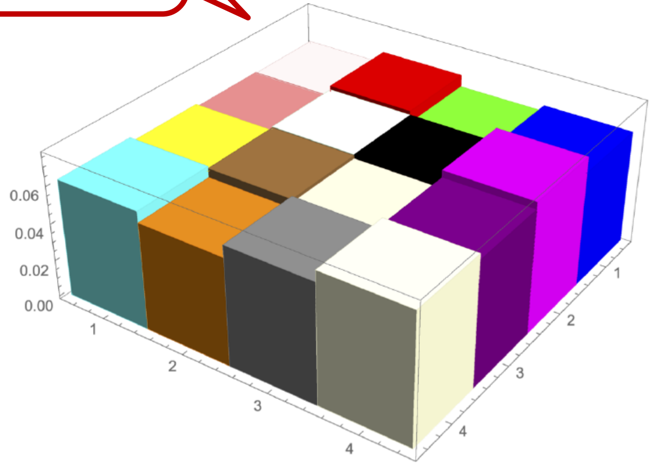
Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):

Which one has *more structure*?

… and it *is* intuitive!

Non-temporal Structure

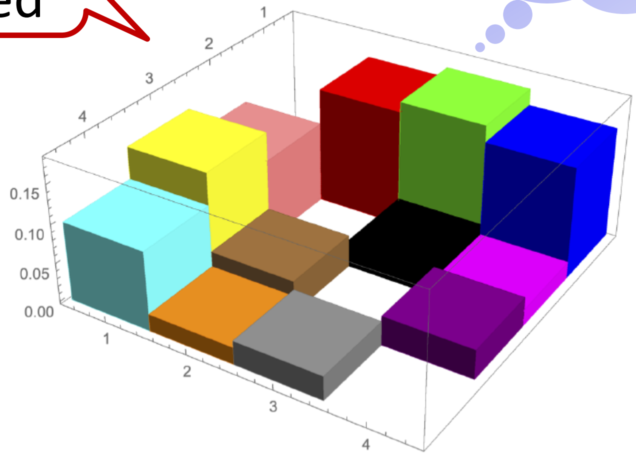
More uniform



VS

More skewed

Color = comm. pair

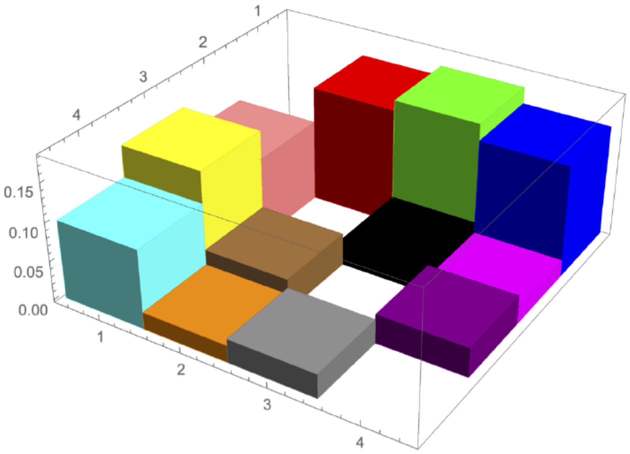


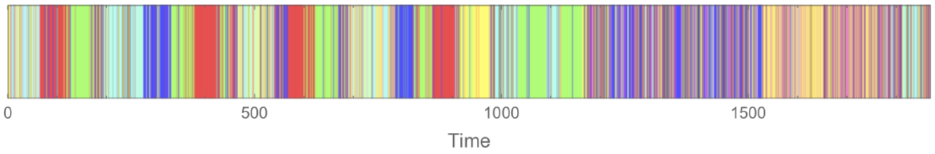
Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):

Which one has *more structure*?

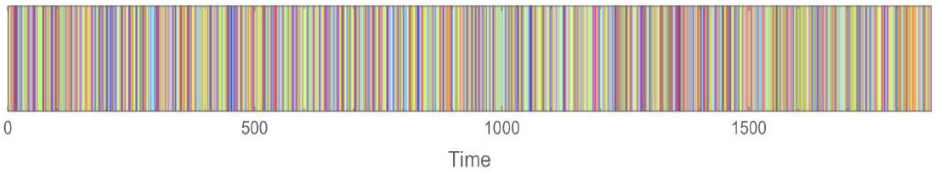
… and it *is* intuitive!

Temporal Structure





VS

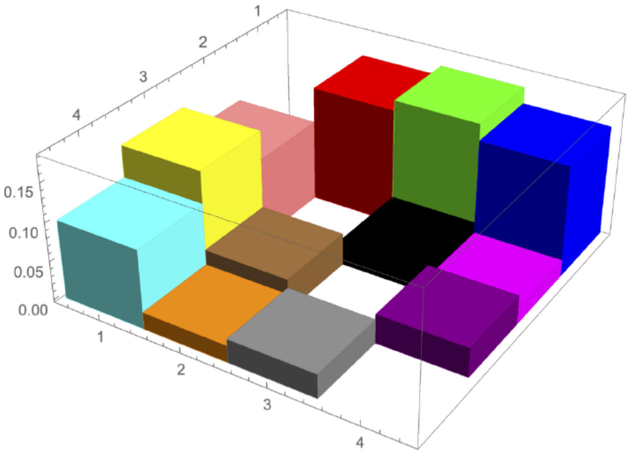


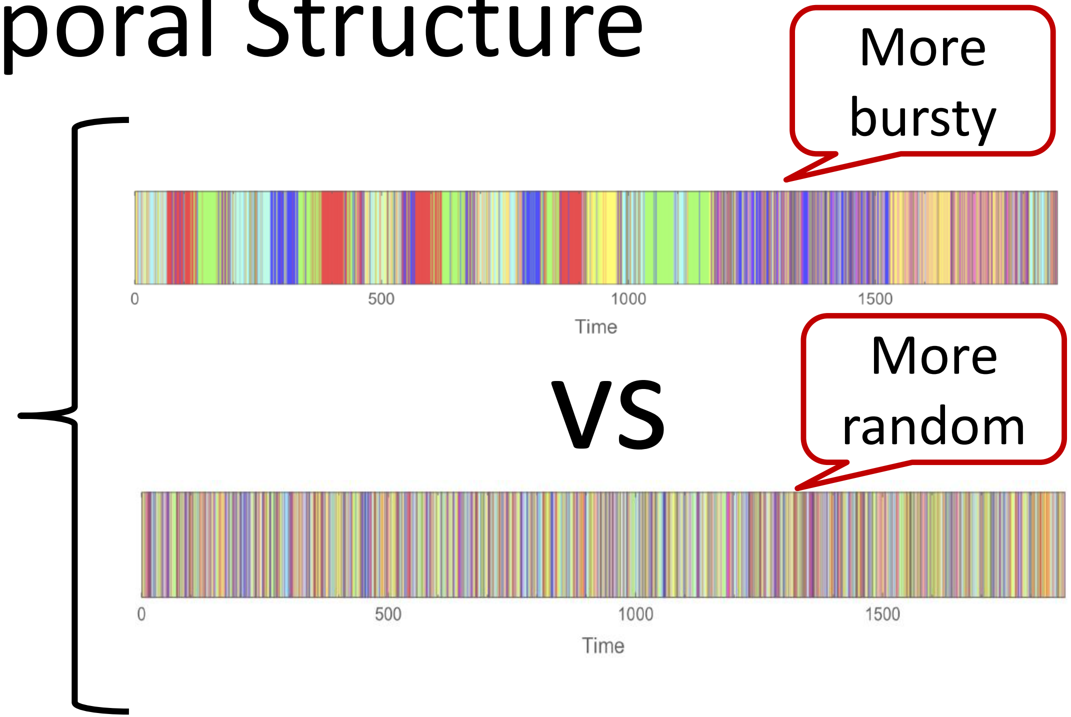
Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

… and it *is* intuitive!

Temporal Structure



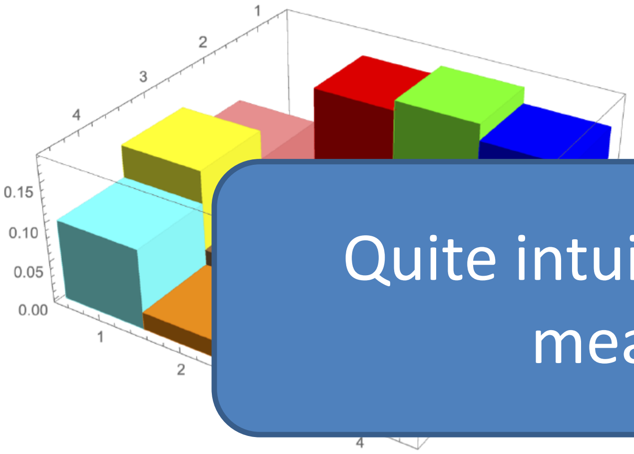


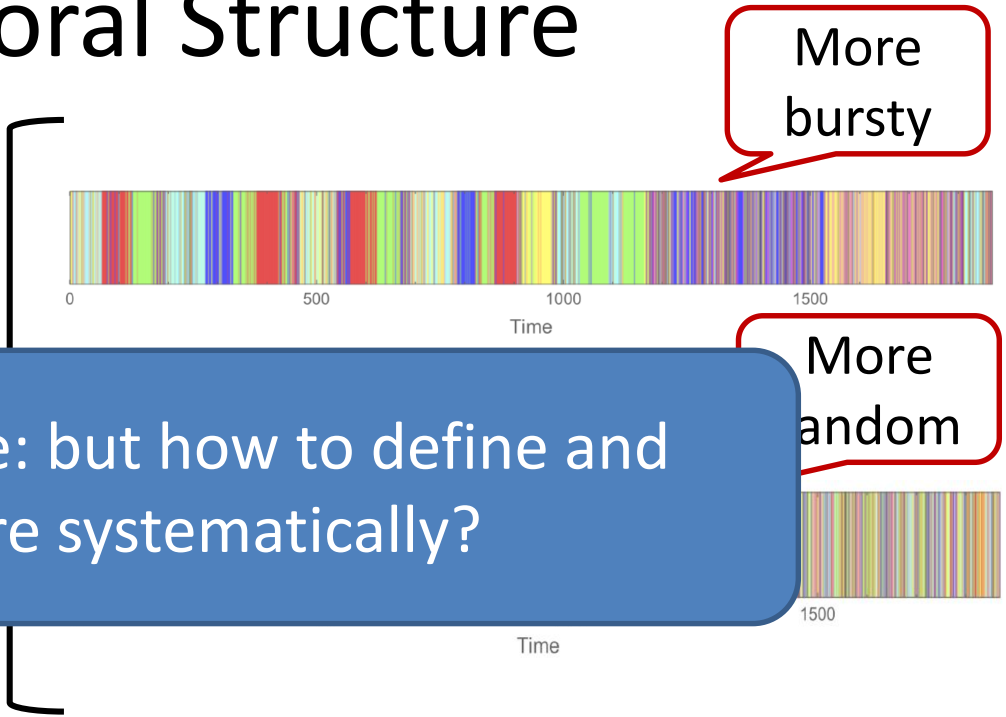
Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

… and it *is* intuitive!

Temporal Structure





Quite intuitive: but how to define and measure systematically?

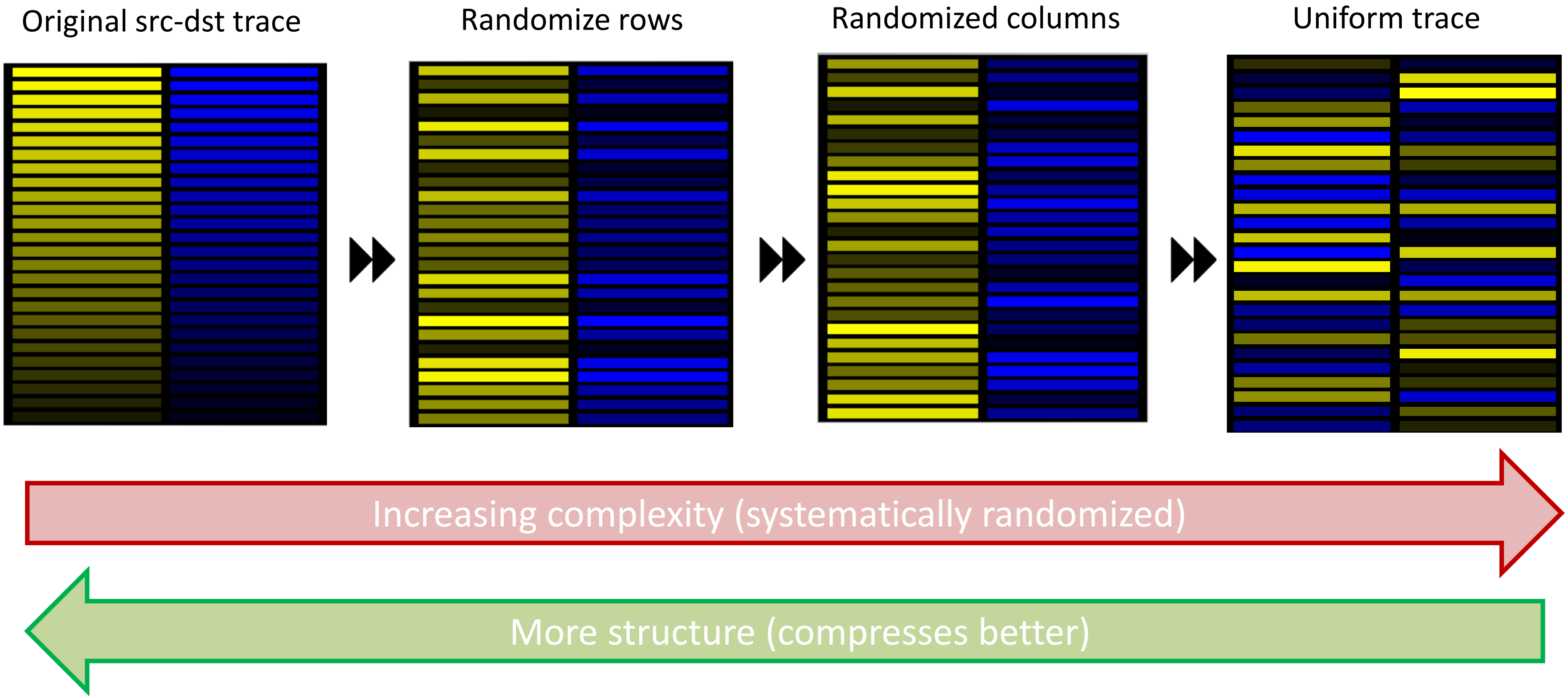
Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

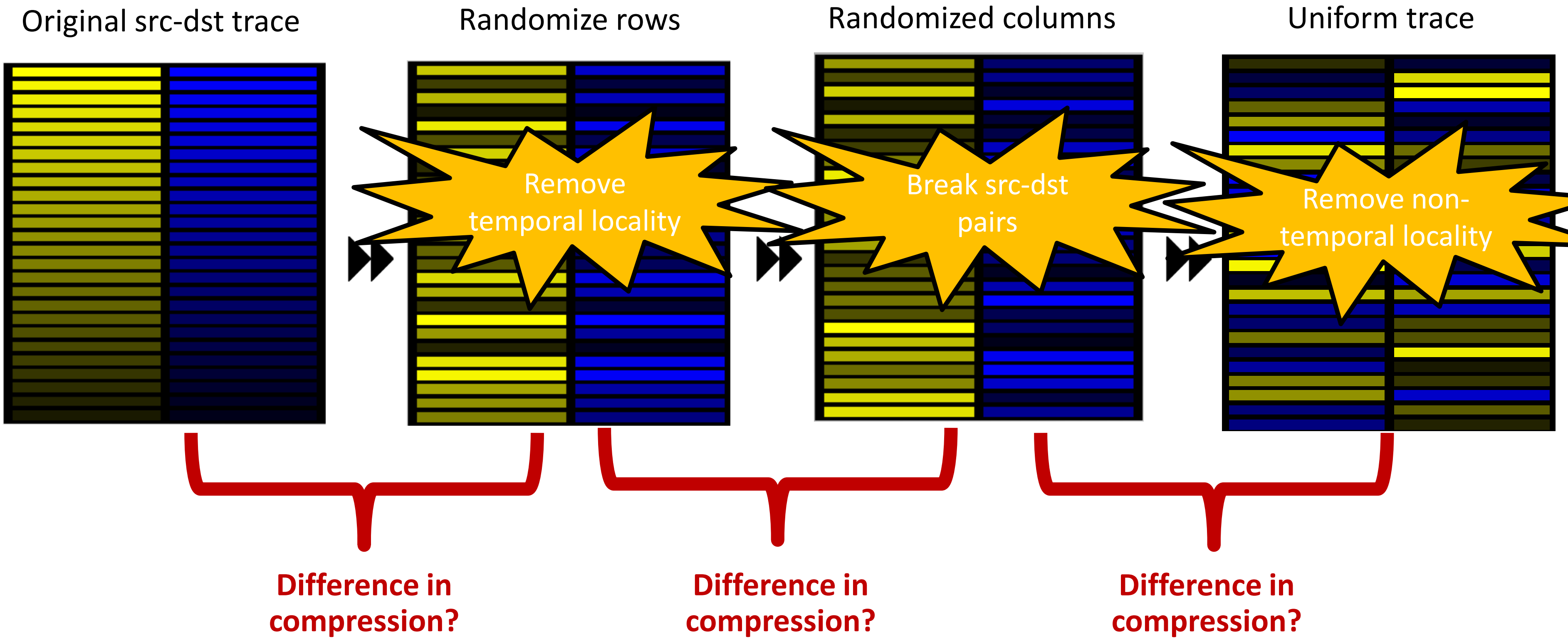
The Trace Complexity

- An **information-theoretic** approach: how can we ***measure the entropy*** (rate) of a traffic trace?
- Henceforth called the **trace complexity**
- Simple approximation: „**shuffle&compress**“
  - Remove structure by iterative *randomization*
  - Difference of compression *before and after* randomization: structure

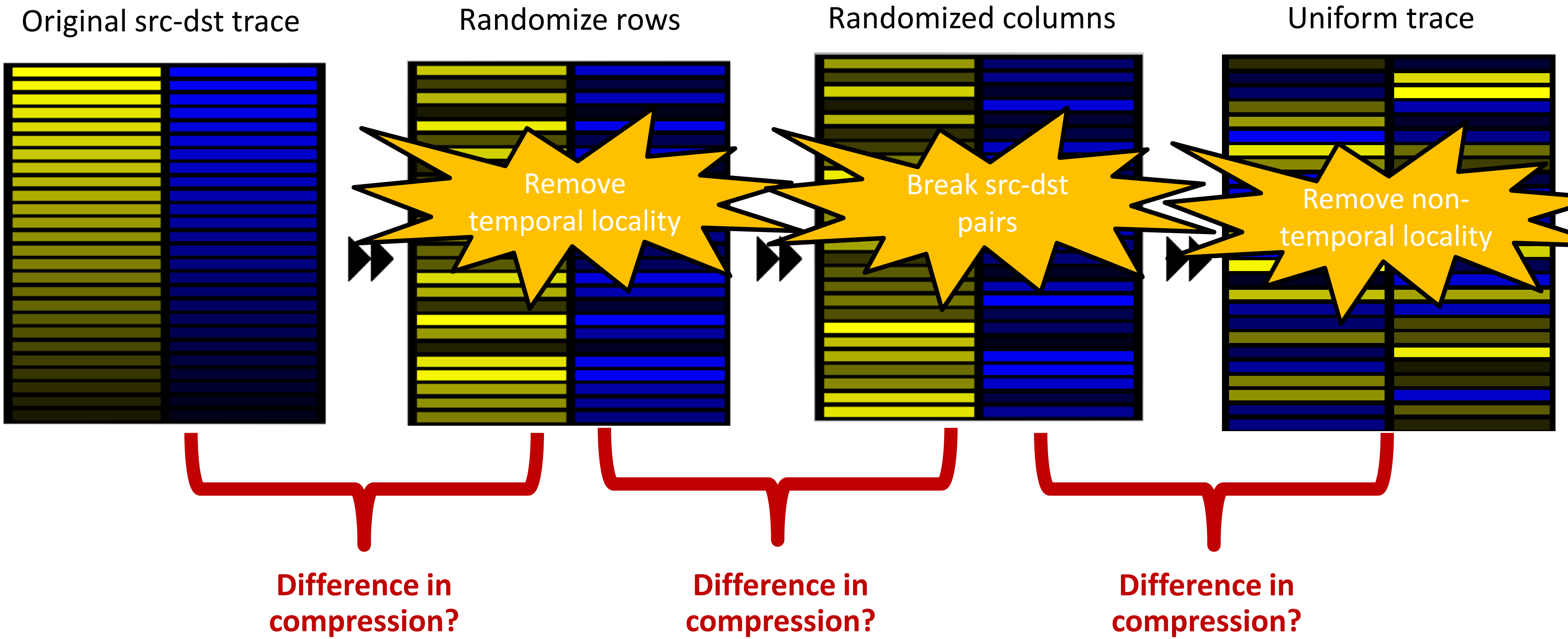
The Trace Complexity



The Trace Complexity

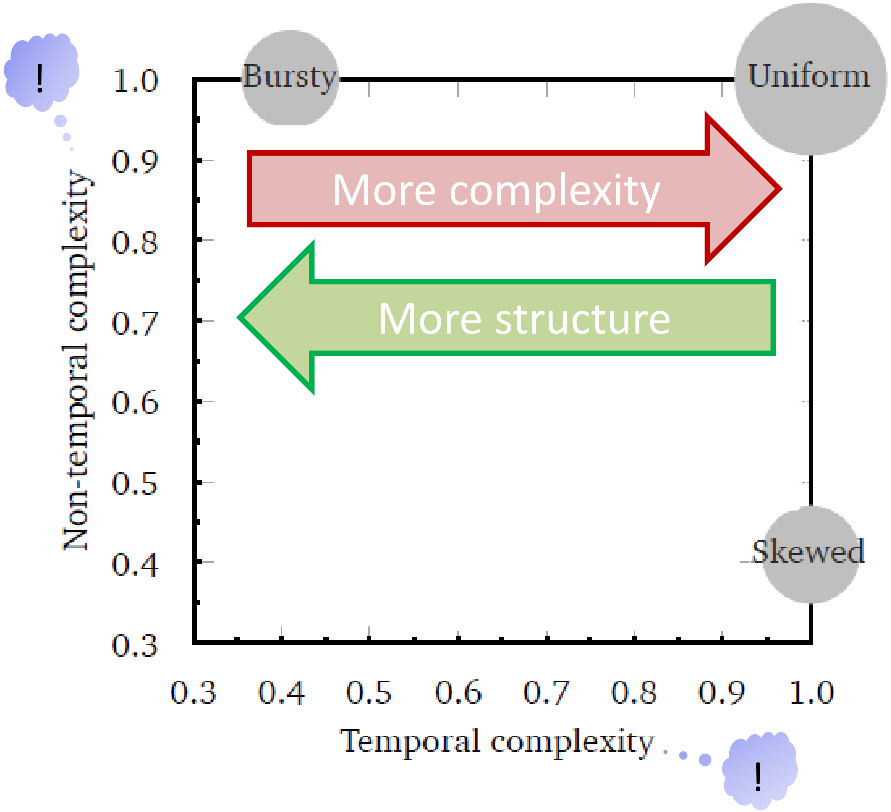


The Trace Complexity



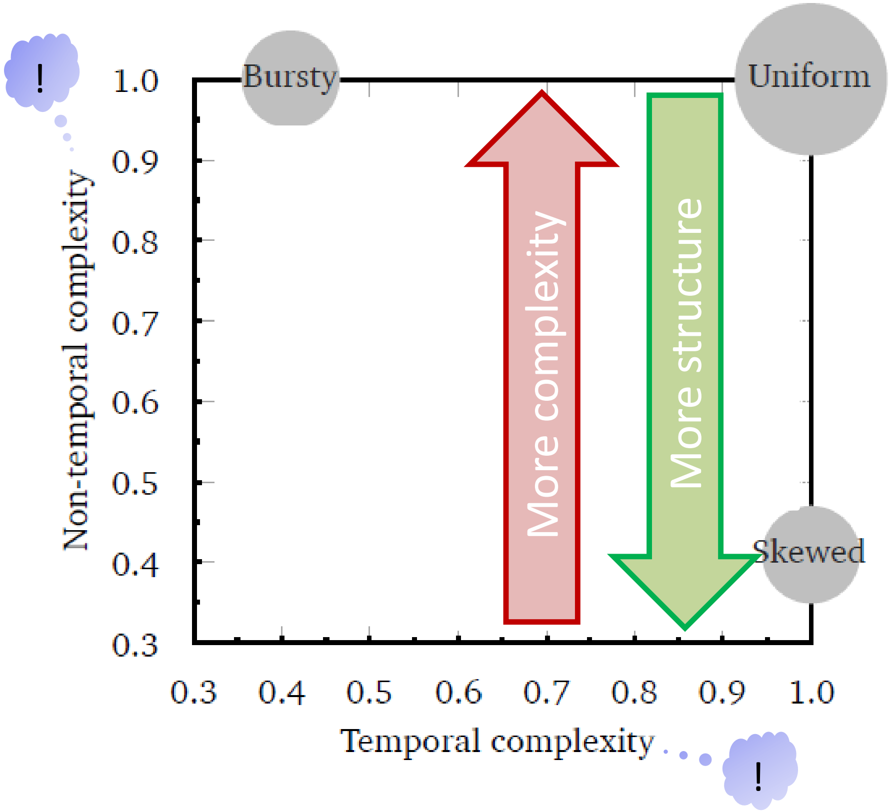
Can be used to define a „complexity map“!

The Complexity Map



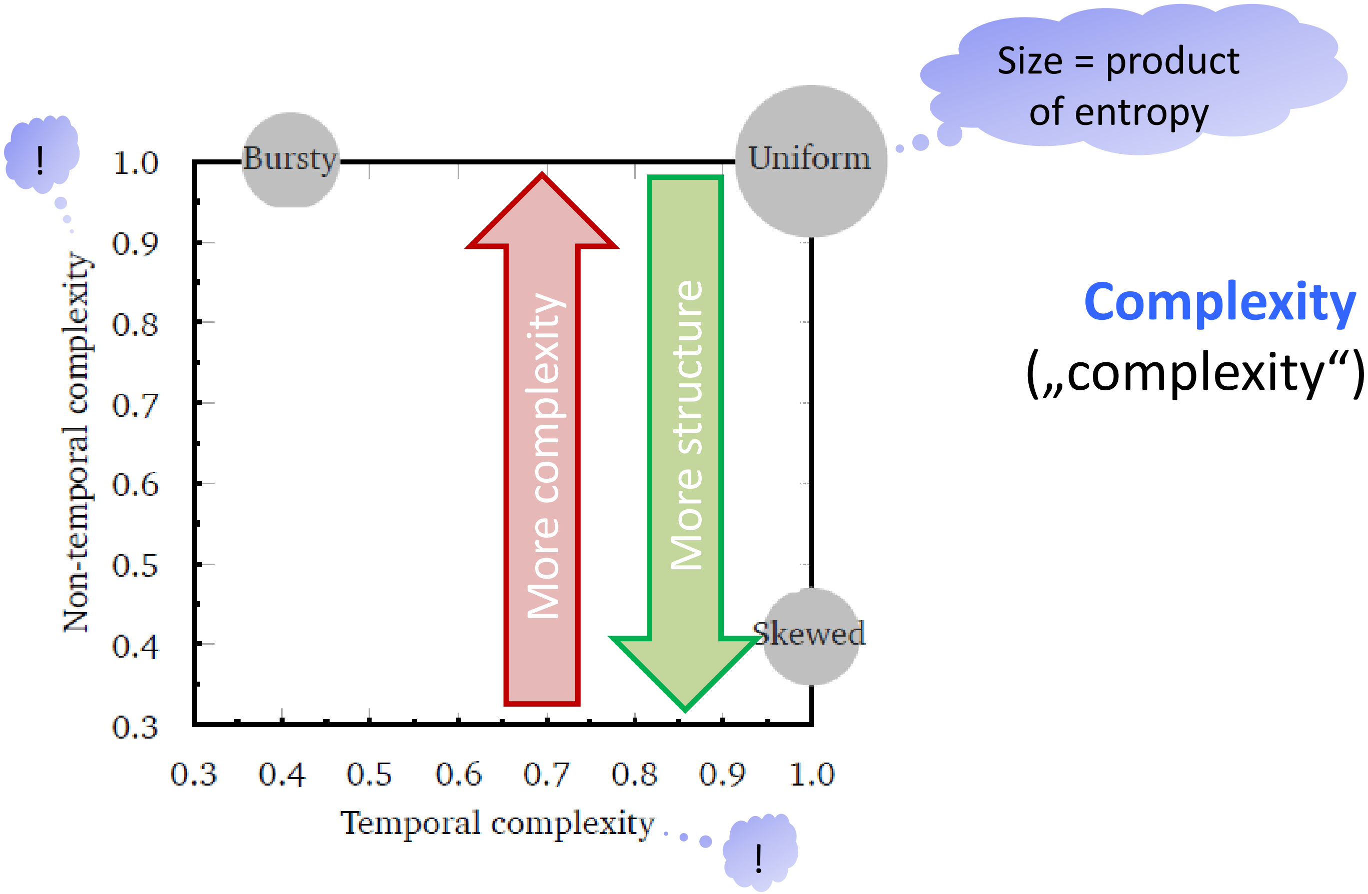
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map



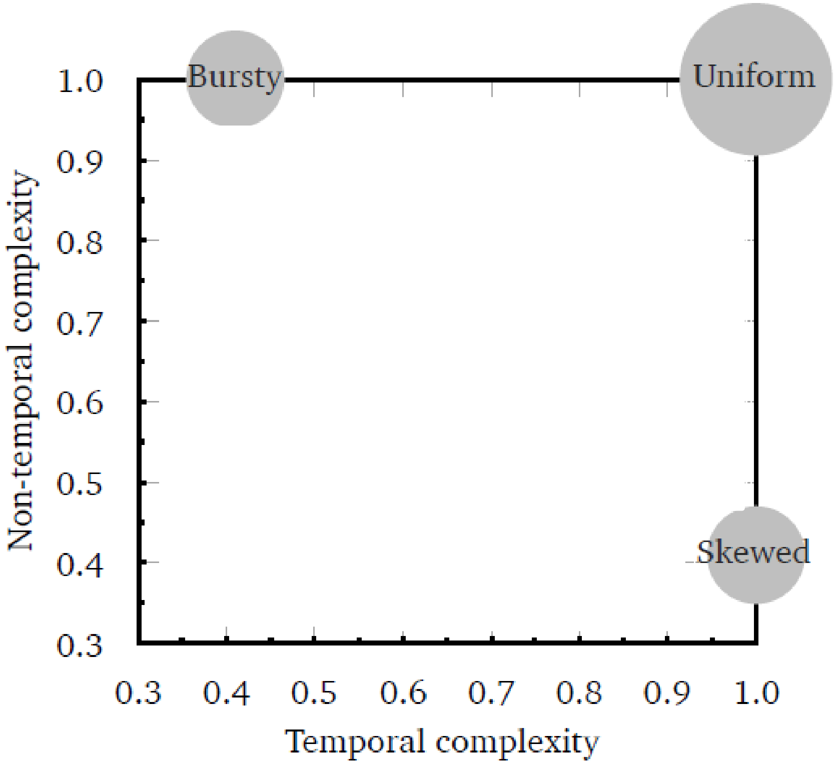
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map



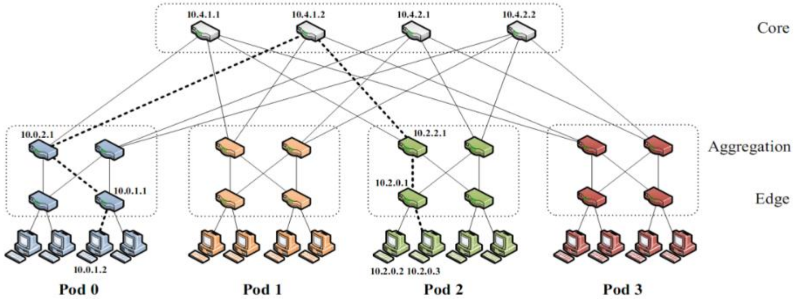
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map

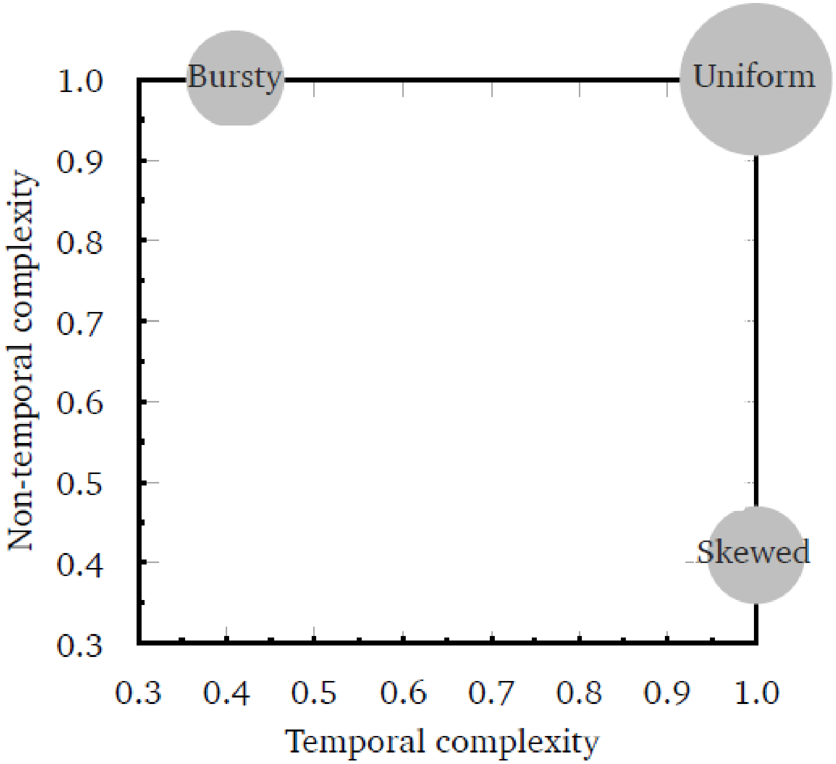


Uniform: Today's datacenters

- Traditional networks are optimized ***for the "worst-case"*** (**all-to-all** communication traffic)
- Example, fat-tree topologies: provide **full bisection bandwidth**

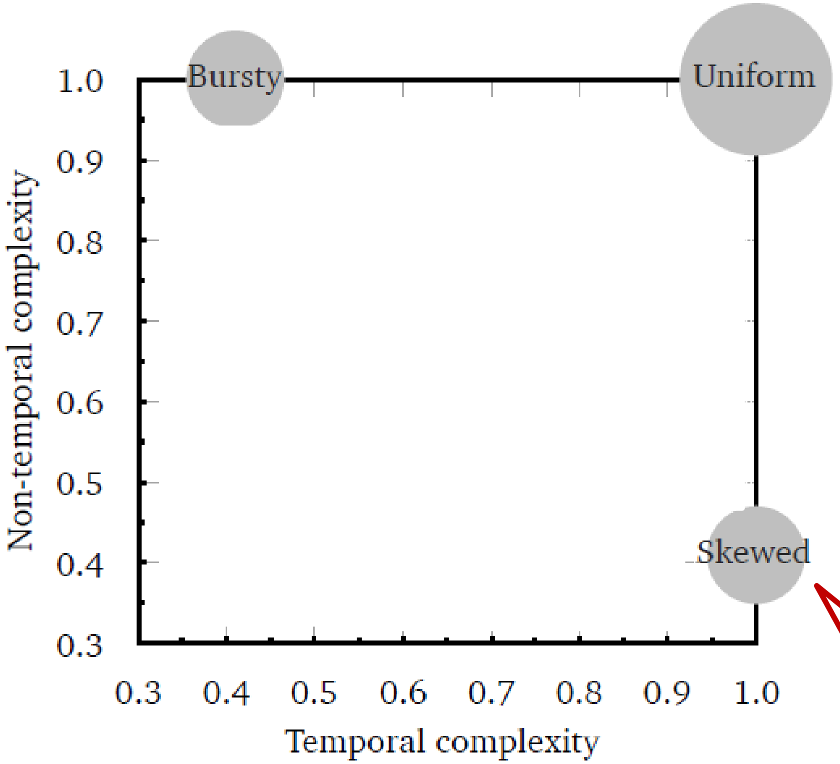


The Complexity Map



Good in the worst case ***but***: cannot leverage different **temporal** and **non-temporal** structures of traffic traces!

The Complexity Map

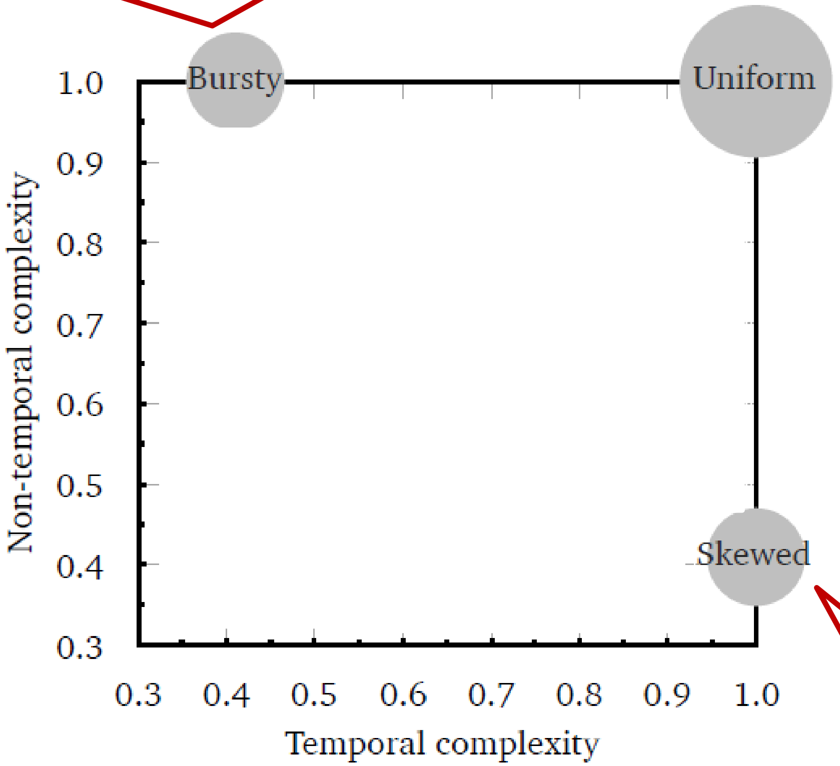


Good in the worst case ***but***: cannot leverage different **temporal** and **non-temporal** structures of traffic traces!

Non-temporal structure could be exploited already with ***static demand-aware networks***!

...plexity Map

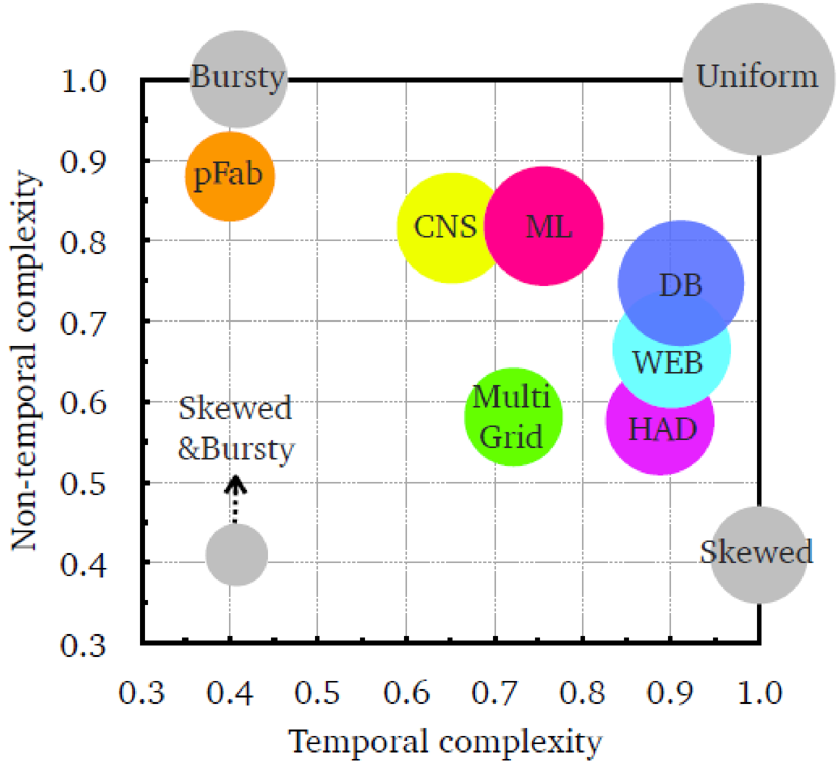
To exploit **temporal** structure, need ***adaptive demand-aware* (*"self-adjusting"*) *networks***.



Good in the worst case ***but***: cannot leverage different **temporal** and **non-temporal** structures of traffic traces!

Non-temporal structure could be exploited already with ***static demand-aware networks***!

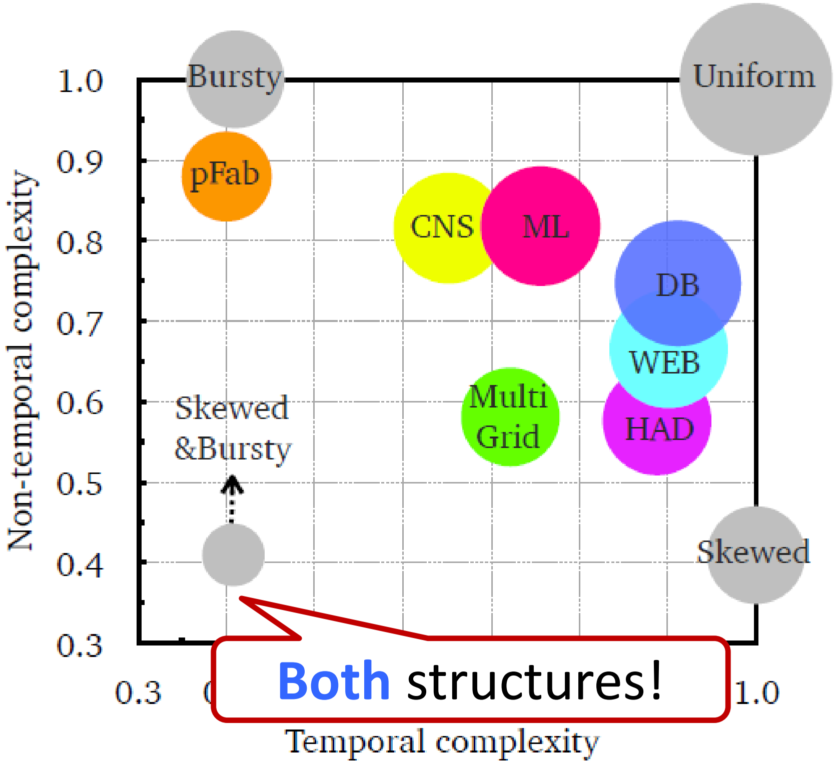
The Complexity Map



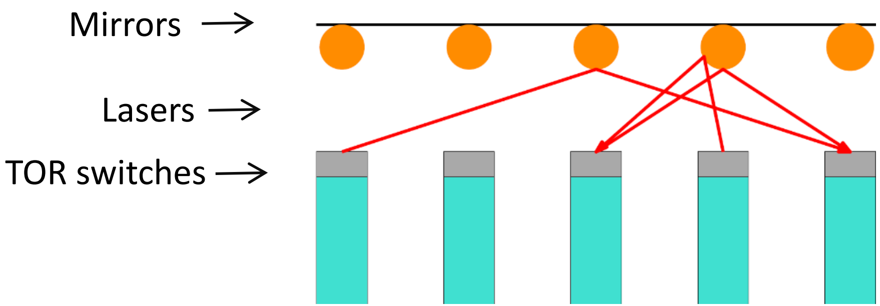
Observation: different applications feature quite significant (and different!) **temporal** and **non-temporal** structures.

- **Facebook** clusters: DB, WEB, HAD
- **HPC** workloads: CNS, Multigrid
- Distributed **Machine Learning** (ML)
- Synthetic traces like **pFabric**

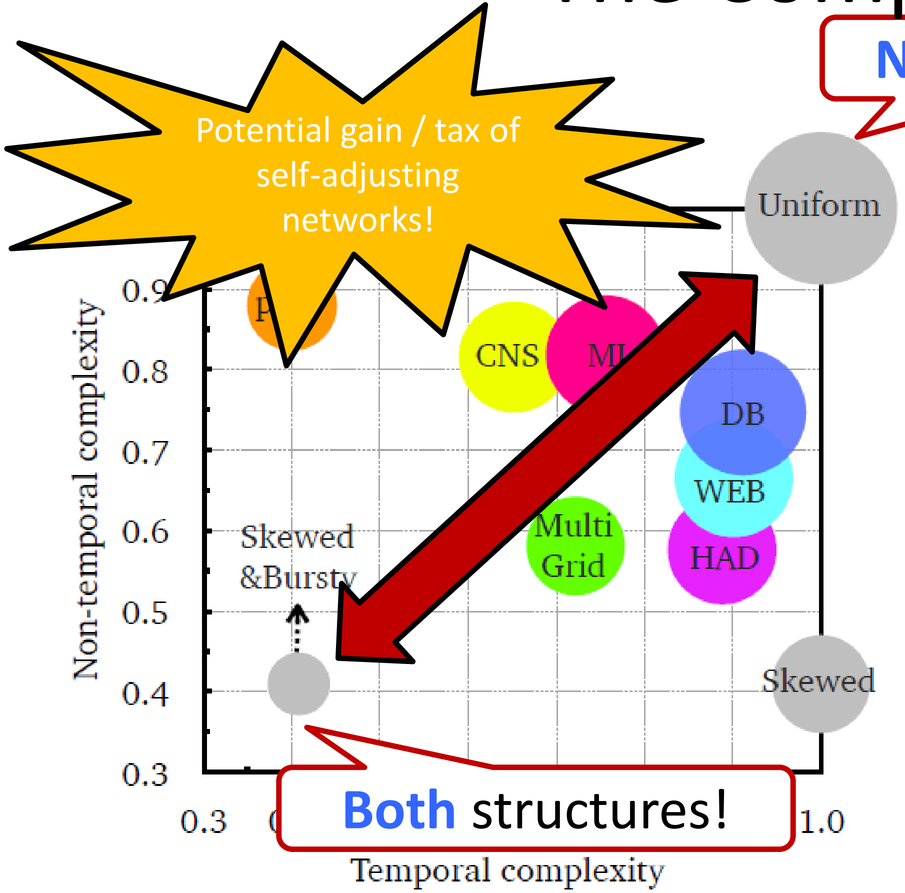
The Complexity Map



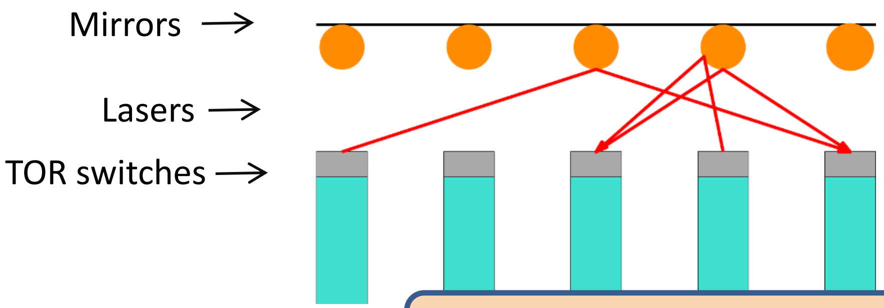
Goal: Design **self-adjusting networks** which leverage ***both*** dimensions of structure!



The Complexity Map



Goal: Design **self-adjusting networks** which leverage ***both*** dimensions of structure!



Measuring the Complexity of Packet Traces. Avin, Ghobadi, Griner, Schmid. **ArXiv** 2019.

But: How to design DANs which also leverage *temporal structure*?

Inspiration from **self-adjusting datastructures** again!

Roadmap

- Entropy: A metric for demand-aware networks?
  - Empirical motivation
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - A connection to self-adjusting datastructures

First: An Analogy

Static vs dynamic demand-aware networks!?

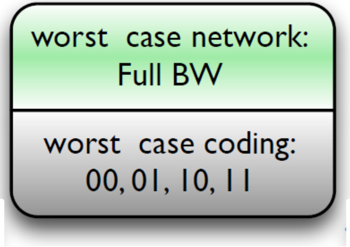
DANs vs **SANs**?

„Coming to Sigmetrics in Phoenix?“

An Analogy to Coding

00110101…

if demand ***arbitrary*** and ***unknown***



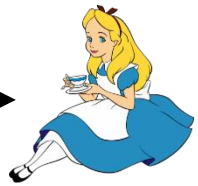
log diameter

log # bits / symbol

„Coming to Sigmetrics in Phoenix?“

An Analogy to Coding

01011…



if demand ***arbitrary*** and ***unknown***

worst case network: Full BW
worst case coding: 00, 01, 10, 11

log diameter

log # bits / symbol

 DAN!

if demand ***known*** and ***fixed***

entropy?

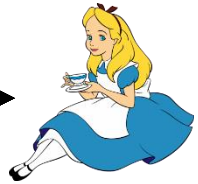
entropy / symbol

static Demand-Aware Nets
static Huffman: 1, 01, 001, 000

„Coming to Sigmetrics in Phoenix?“

An Analogy to Coding

011…



if demand ***arbitrary*** and ***unknown***

worst case network: Full BW
worst case coding: 00, 01, 10, 11

log diameter

log # bits / symbol

Dynamic DANs: Aka. **Self-Adjusting Networks** (SANs)!

DAN!

SAN!

if demand ***known*** and ***fixed***

entropy?

entropy / symbol

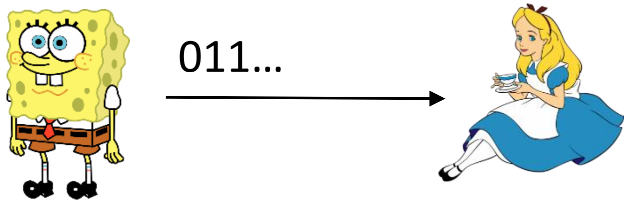
static Demand-Aware Nets
static Huffman: 1, 01, 001, 000

if demand ***unknown*** but ***reconfigurable***

dynamic Demand-Aware Nets
dynamic Huffman codes

An Analogy to Coding

„Coming to Sigmetrics in Phoenix?“



if demand ***arbitrary*** and ***unknown***

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

log diameter

log # bits / symbol

DAN!

SAN!

Dynamic DANs:
Aka. **Self-Adjusting Networks** (SANs)!

if demand ***known*** and ***fixed***

static
Demand-Aware Nets

static Huffman:
1, 01, 001, 000

Can exploit
spatial locality!

if demand ***unknown*** but ***reconfigurable***

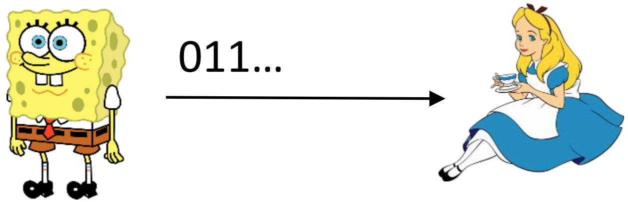
dynamic
Demand-Aware Nets

dynamic
Huffman codes

Additionally exploit
temporal locality!

„Coming to Sigmetrics in Phoenix?“

An Analogy to Coding



if demand ***arbitrary*** and ***unknown***

worst case network:
Full BW

log diameter

worst case coding:
00, 01, 10, 11

log # bits / symbol

Dynamic DANs:
Aka. **Self-Adjusting Networks** (SANs)!

DAN!

SAN!

if demand ***kno***

if demand ***unknown*** but ***r***

Aware Nets

static Huffman:
1, 01, 001, 000

dynamic
Demand

Huff codes

Can exploit
spat

„Cheating“: need to know demand!

Need online algorithms!

Additionally exploit
temporal locality!

Analogous to *Datastructures*: Oblivious…

- Traditional, **fixed** BSTs do not rely on any assumptions on the demand

- Optimize for the **worst-case**

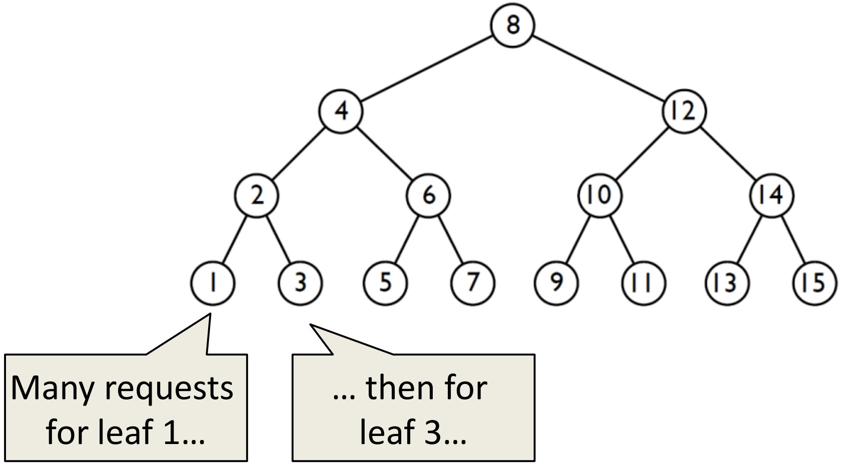
- Example **demand**:

  1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)

  many *many* *many* *many* *many*

- Items stored at ***O(log n)*** from the root, **uniformly** and **independently** of their frequency

Corresponds to *max possible demand*!



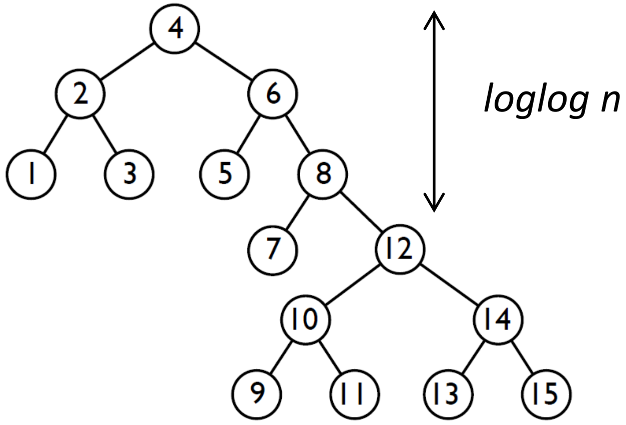
… Demand-Aware …

- **Demand-aware fixed** BSTs can take advantage of ***spatial locality*** of the demand

- E.g.: place frequently accessed elements close to the root

- E.g., **Knuth/Mehlhorn/Tarjan** trees

- Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)
  - Amortized cost ***O(loglog n)***

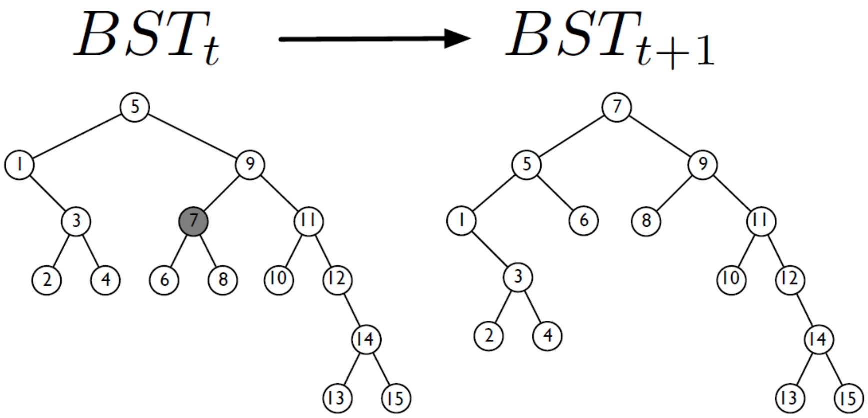


Amortized cost corresponds to ***empirical entropy of demand***!

… Self-Adjusting!

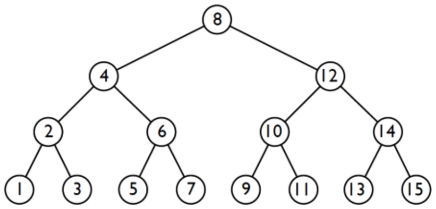
- **Demand-aware reconfigurable** BSTs can additionally take advantage of ***temporal locality***

- By moving accessed element to the root: amortized cost is ***constant***, i.e., O(1)
  - Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)



Datastructures

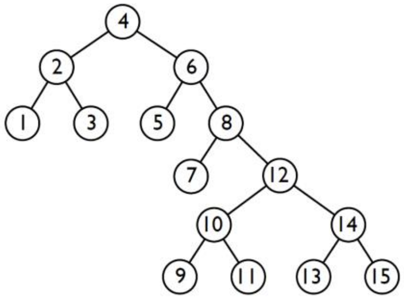
Oblivious



Lookup

O(log n)

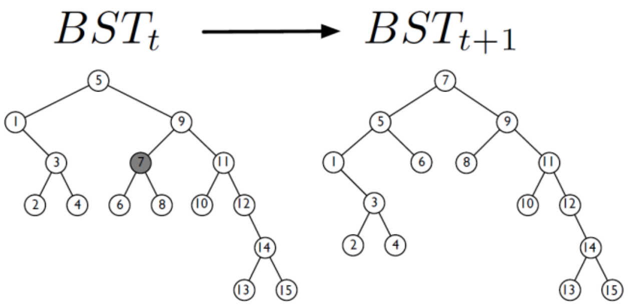
Demand-Aware



Exploit **spatial locality**:

empirical entropy O(loglog n)

Self-Adjusting

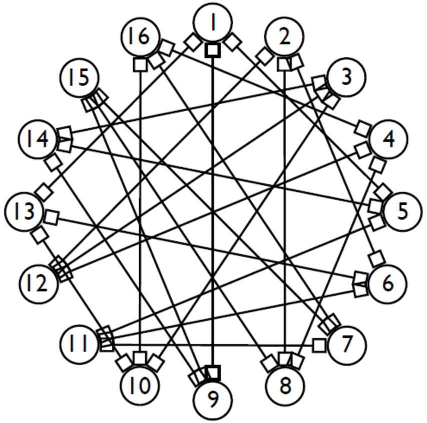


Exploit **temporal locality** as well:

O(1)

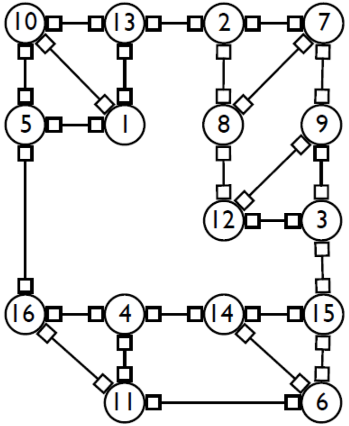
Analogously for Networks

Oblivious



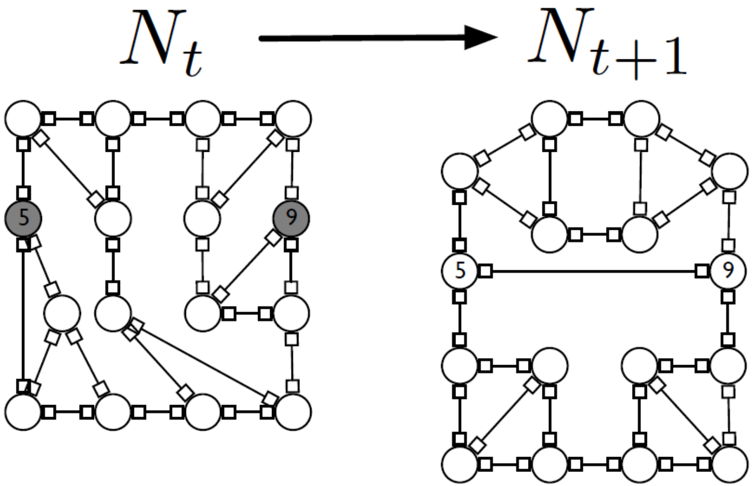
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



Exploit **spatial locality**

SAN



Exploit **temporal locality** as well

Avin, S.: Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. **SIGCOMM CCR** 2018.

Now: Design of Self-Adjusting Networks (SANs)

Inspiration from **self-adjusting datastructures** again!

What’s the model?

What’s the model?

Again: *it depends…* ☺

The Problem Input

A **sequence σ** = (u1,v1), (u2,v2), (u3,v3)….

chosen arbitrarily

Chosen i.i.d. from initially unknown fixed distribution

The Problem Input

A **sequence σ** = (u1,v1), (u2,v2), (u3,v3)….

chosen arbitrarily

Chosen i.i.d. from initially unknown fixed distribution

revealed online

given offline

The Problem Input

chosen arbitrarily

Chosen i.i.d. from initially unknown fixed distribution

A **sequence σ** = (u1,v1), (u2,v2), (u3,v3)….

revealed online

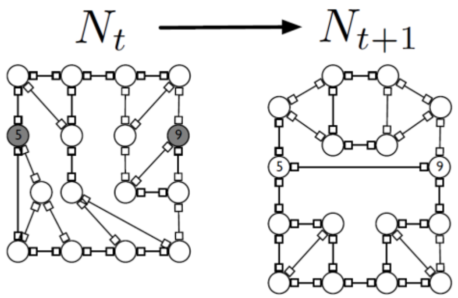
given offline

Other options: sequences of *snapshots*, generated according to *Markov process*, …

What's the objective? Metric?

Also here: *it depends…* ☺

A Cost-Benefit Tradeoff



Short routes

High reconfiguration cost

Tradeoff

Low reconfiguration cost

Long routes

Basic question:

How often to reconfigure?

A Metric

Entropy of the demand again…

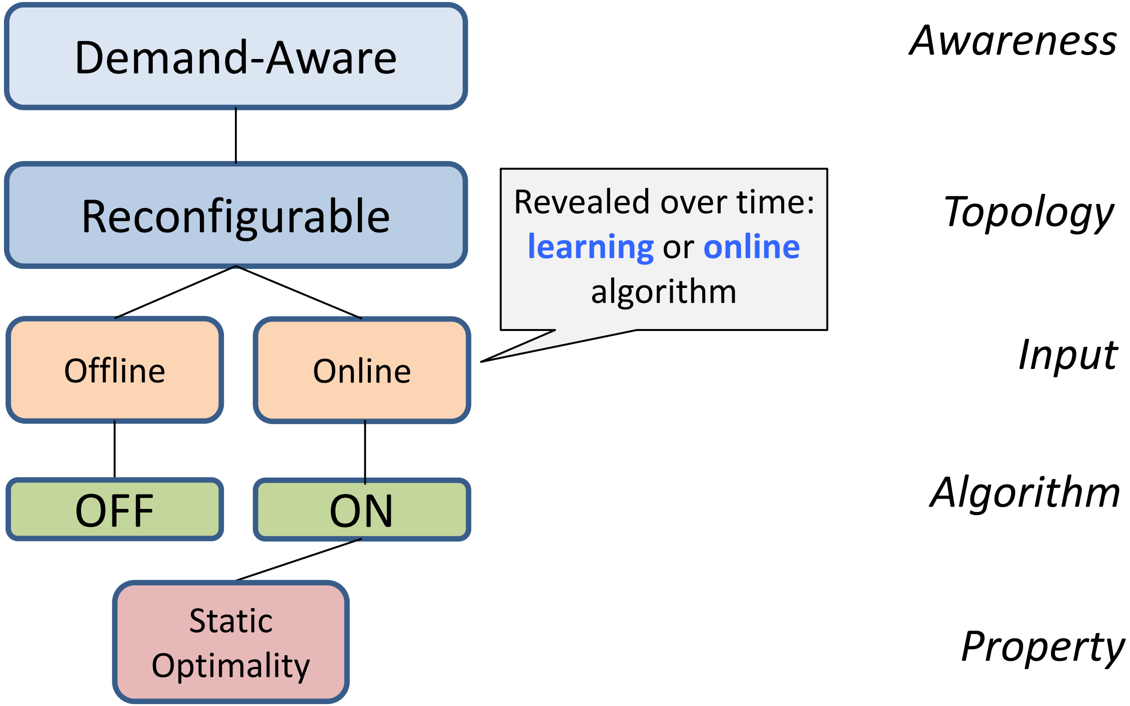
… but now **entropy rate** (entropy over time)!

A Taxonomy: Reconfigurable Networks

Static Optimality:

"***Not worse than static*** which knows demand ahead of time!"

ρ = Cost(ON)/Cost(STAT*) is constant.

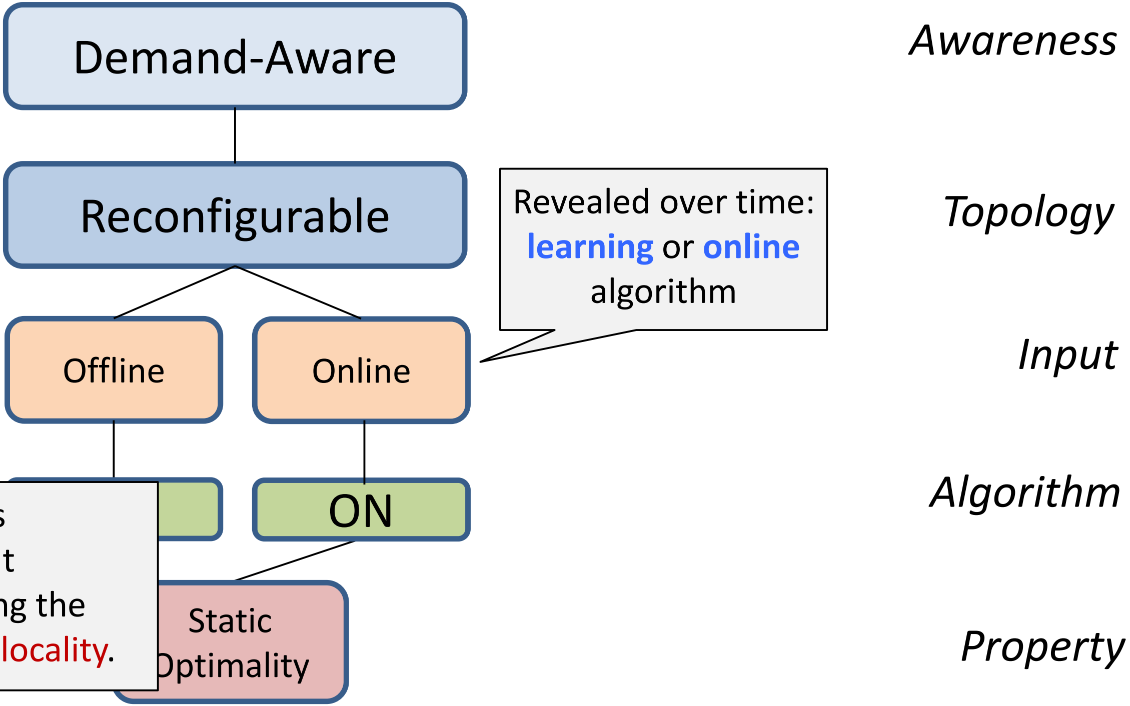


A Taxonomy: Reconfigurable Networks

Static Optimality:

"***Not worse than static*** which knows demand ahead of time!"

ρ = Cost(ON)/Cost(STAT*) is constant.

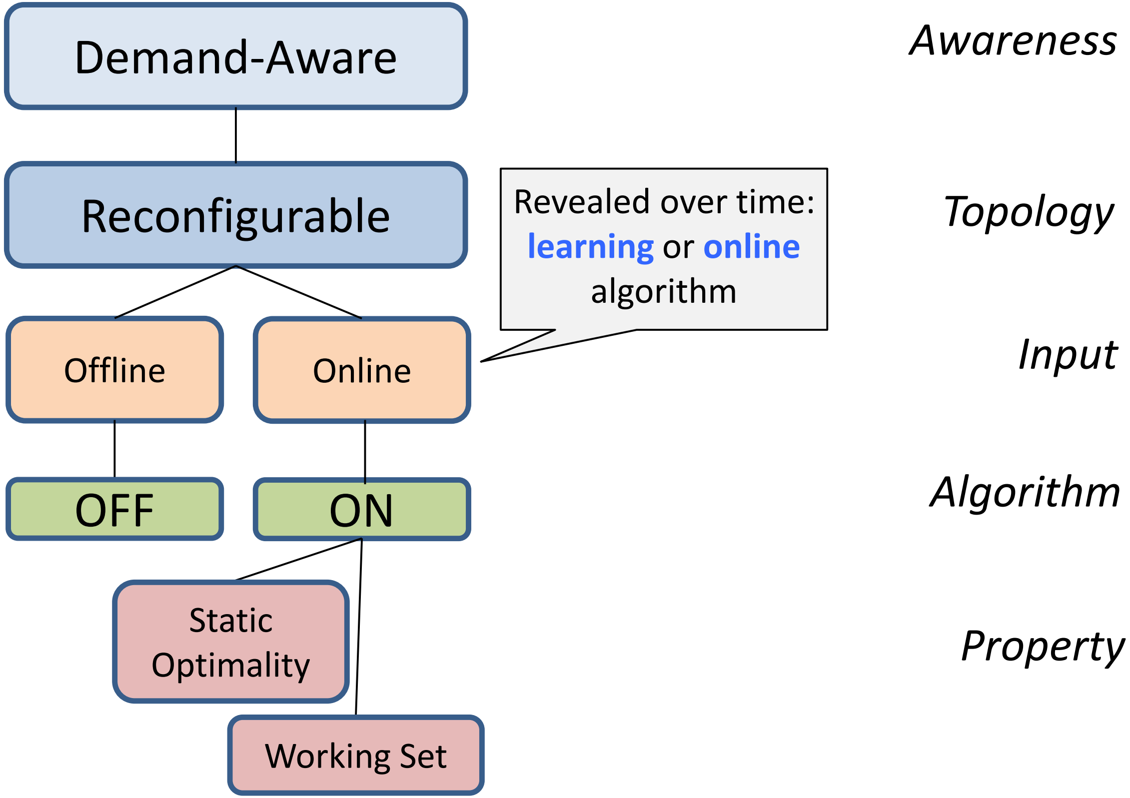


Note: may be ***<<1***. ON has advantage of adjusting, but the disadvantage of not knowing the workload. E.g. if much temporal locality.

A Taxonomy: Reconfigurable Networks

Working Set Property:

"***Topological distance*** between nodes ***proportional to how recently they communicated***!"



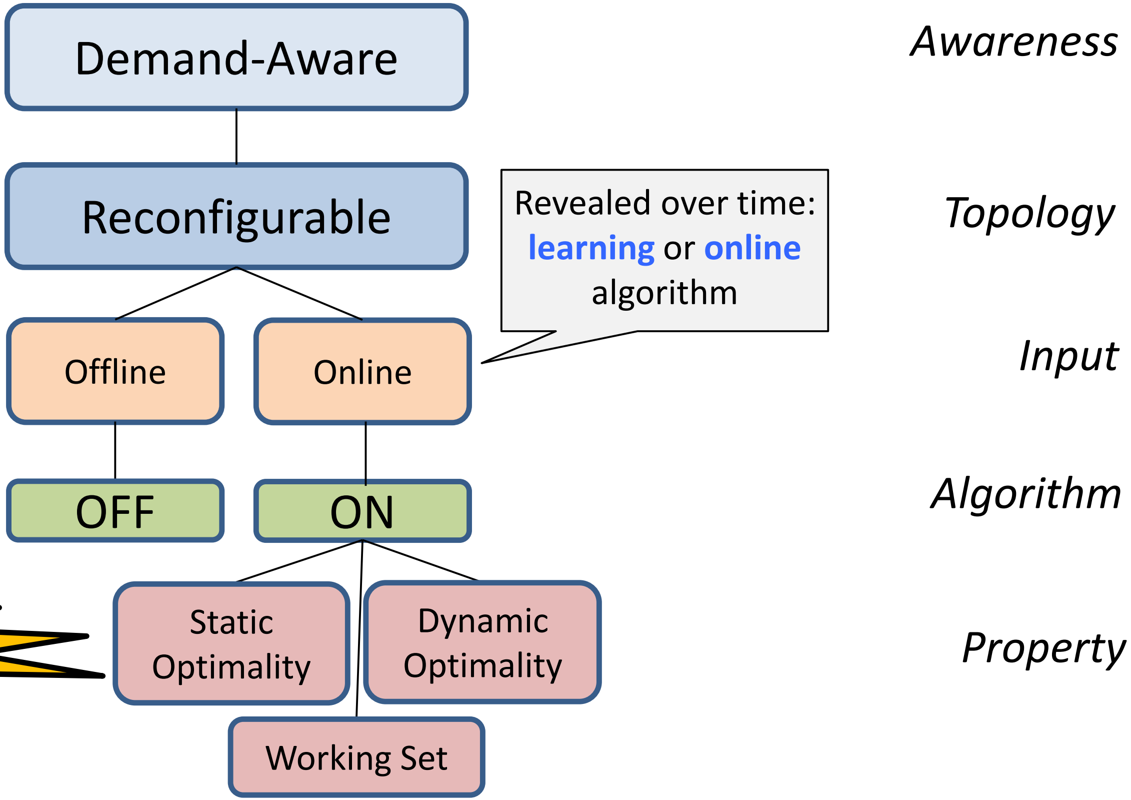
A Taxonomy: Reconfigurable Networks

Dynamic Optimality:

“***No worse than an offline*** algorithm which ***knows the sequence***!”

ρ = Cost(ON)/Cost(OFF*) is constant.

Always >=1.



Algorithms for Self-Adjusting Networks

Algorithms for Self-Adjusting Networks

Let us start with ***trees*** again:

Self-adjusting tree?

Algorithms for Self-Adjusting Networks

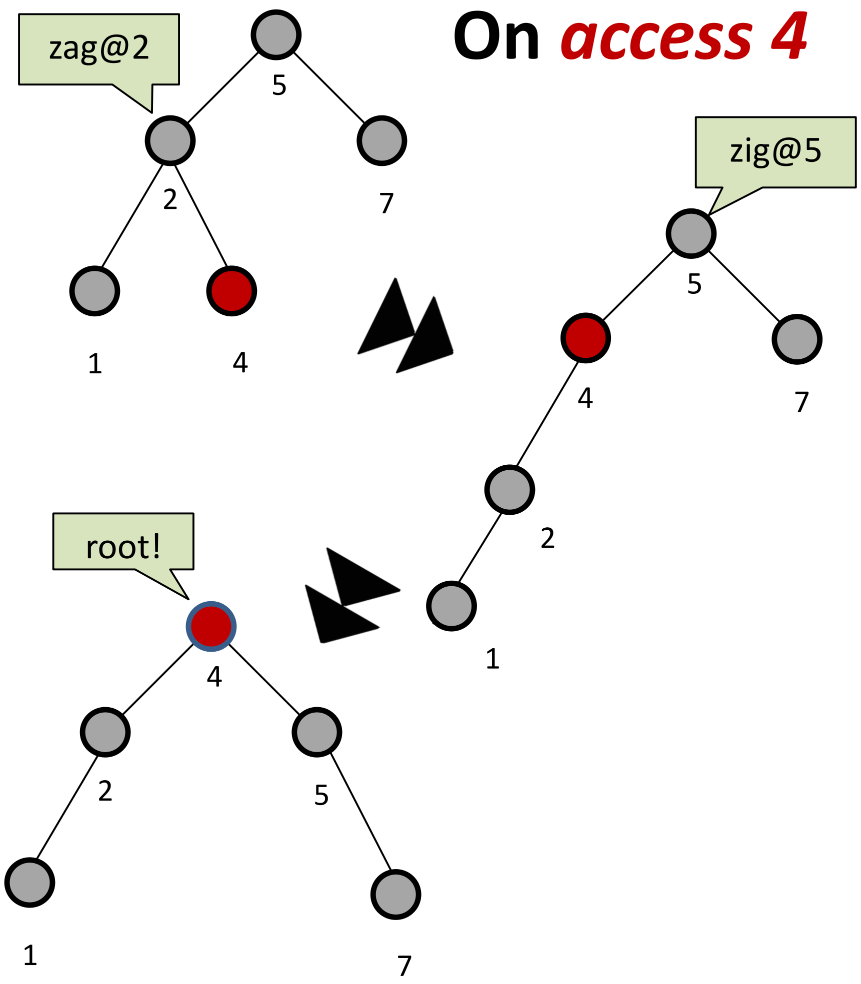
Let us start with ***trees*** again:

Self-adjusting tree?

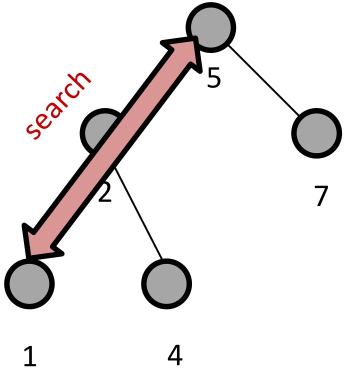
Use **self-adjusting BST**!

Recall: Splay Tree

- A Binary Search Tree (**BST**)
- Inspired by "**move-to-front**": move ***to root***!
- Self-adjustment: **zig**, **zigzig**, **zigzag**
  - Maintains ***search property***
- Many nice properties
  - **Static optimality**, **working set**, (static,dynamic) **fingers**, …

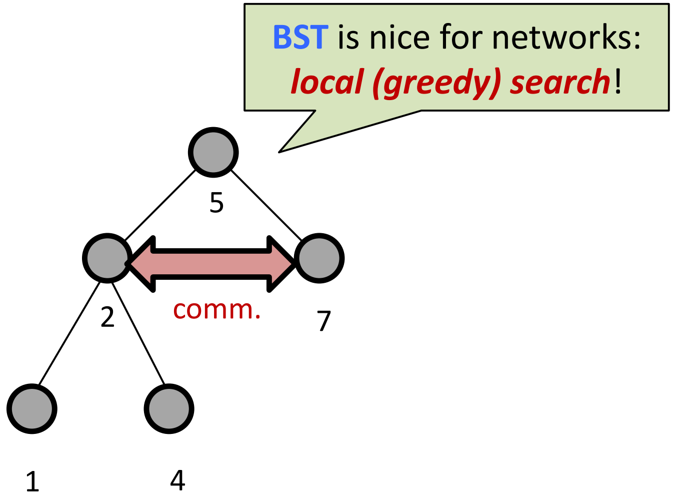


A Simple Idea:
Generalize Splay Tree To *SplayNet*



Splay Tree

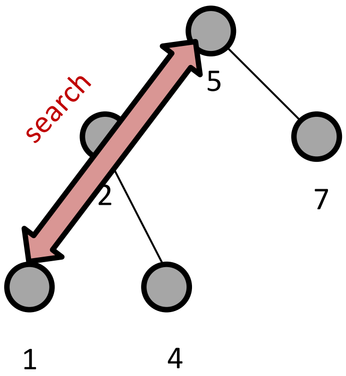
vs



SplayNet

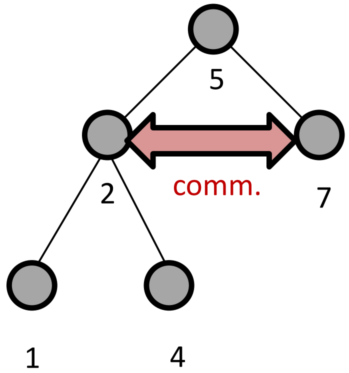
A Simple Idea:
Generalize Splay Tree To *SplayNet*

But how?



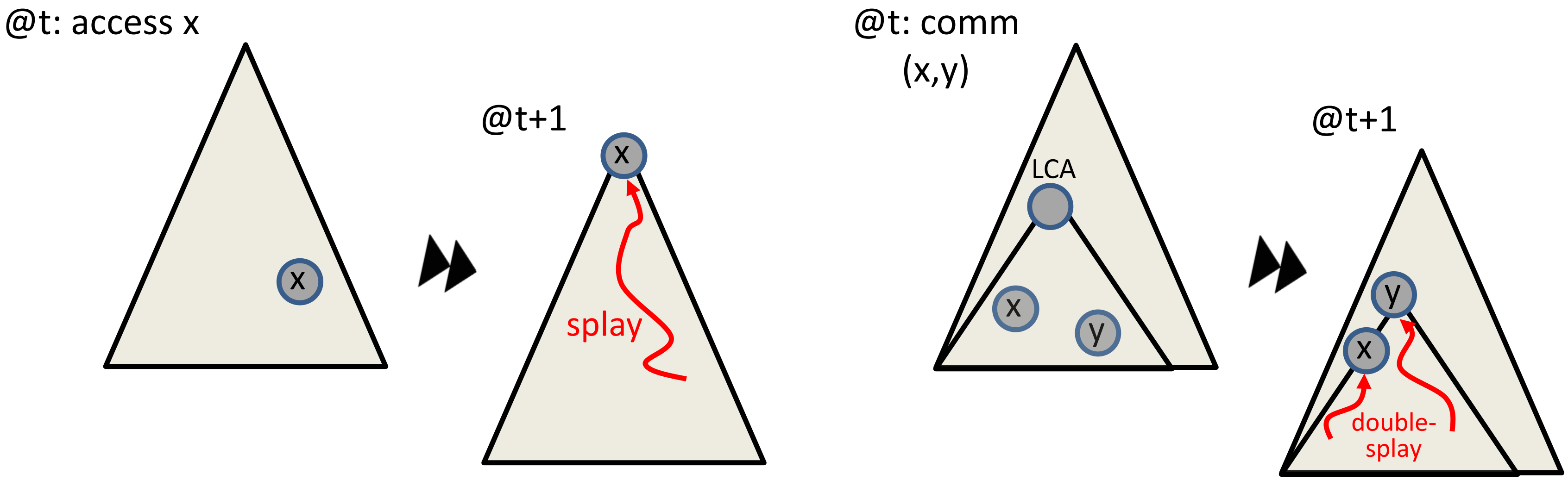
Splay Tree

VS



SplayNet

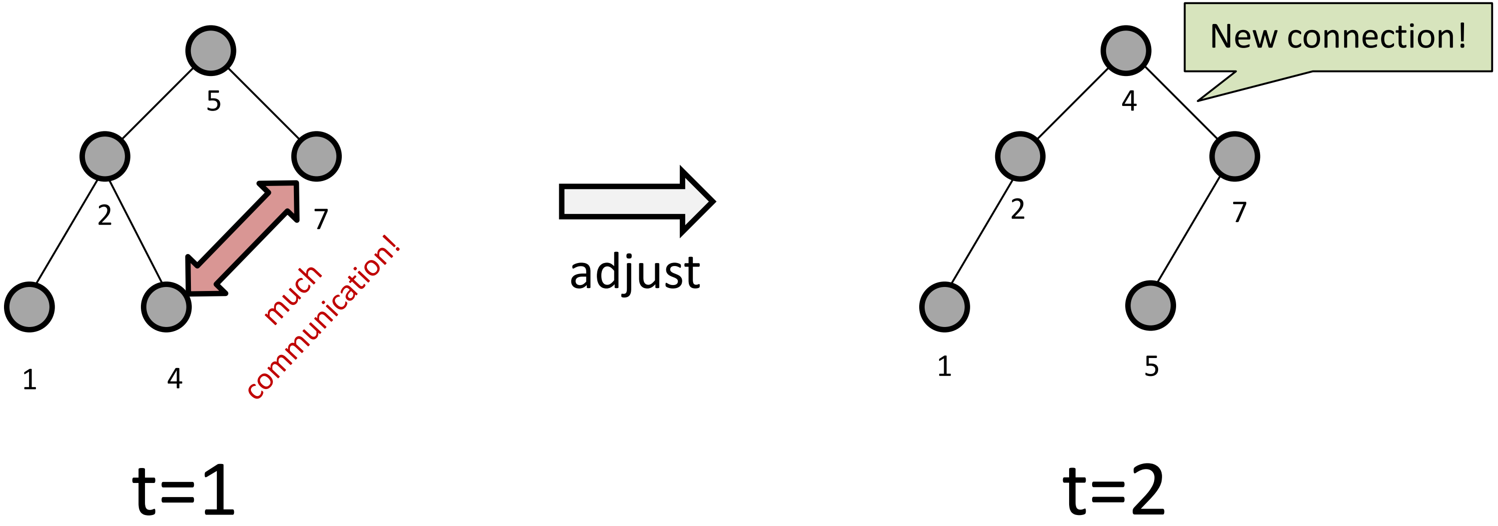
SplayNet: A Simple Idea



Splay Tree

SplayNet

Example

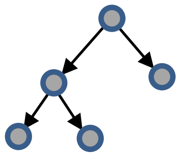


Challenges: How to **minimize reconfigurations**?
How to keep network **locally** routable?

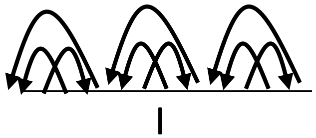
Properties of SplayNets

- **Statically optimal** if demand comes from a ***product distribution***
  - Product distribution: entropy equals conditional entropy, i.e., H(X)+H(Y)=H(X|Y)+H(X|Y)
- Converges to optimal static topology in
  - **Multicast scenario**: requests come from a binary tree as well
  - **Cluster scenario**: communication only within interval
  - **Laminated scenario** : communication is „non-crossing matching"

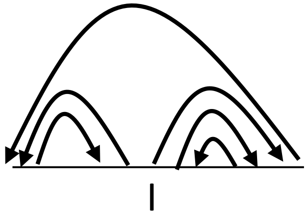
Multicast Scenario



Cluster Scenario



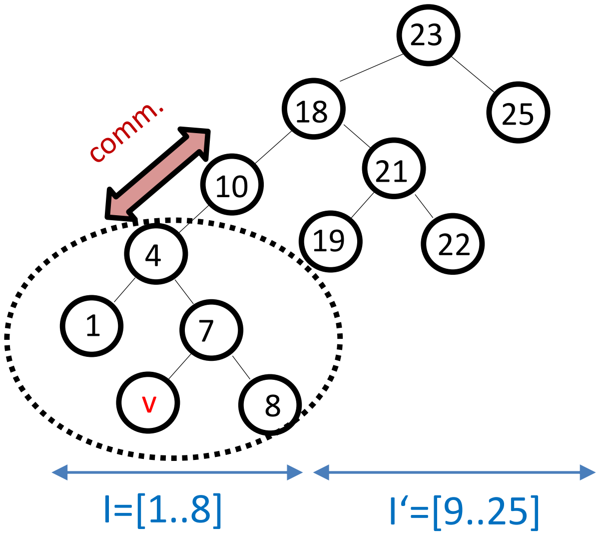
Laminated Scenario



Remark: Static SplayNet

Theorem: Optimal static SplayNet can be computed in polynomial-time (dynamic programming)

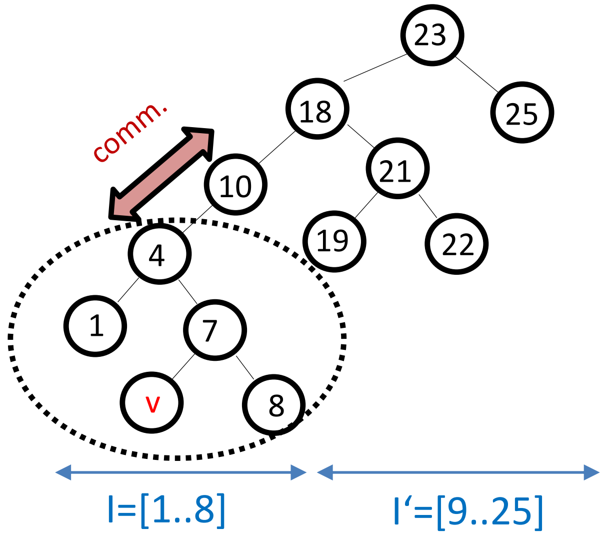
- **Unlike unordered tree?**



Remark: Static SplayNet

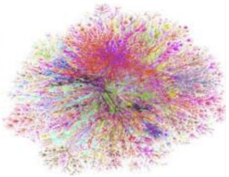
Theorem: Optimal static SplayNet can be computed in polynomial-time (dynamic programming)

- **Unlike unordered tree?**



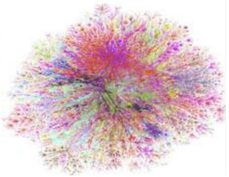
SplayNet: Towards Locally Self-Adjusting Networks. Schmid et al. IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016.

Algorithms for Self-Adjusting Networks II

From trees to networks!

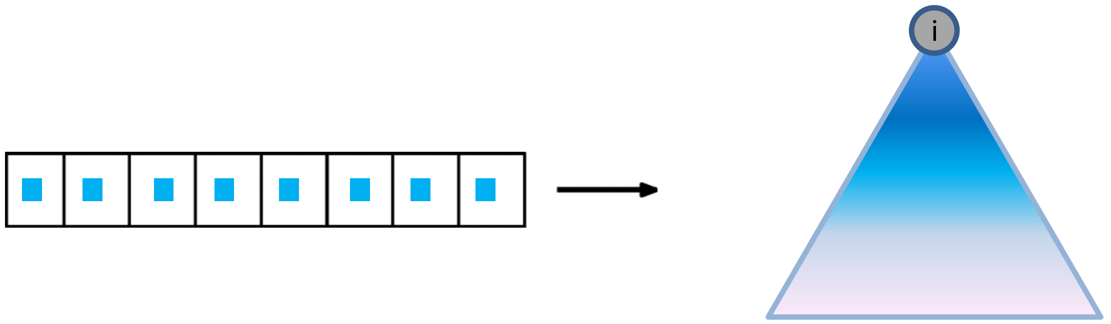
Algorithms for Self-Adjusting Networks II

From trees to networks!

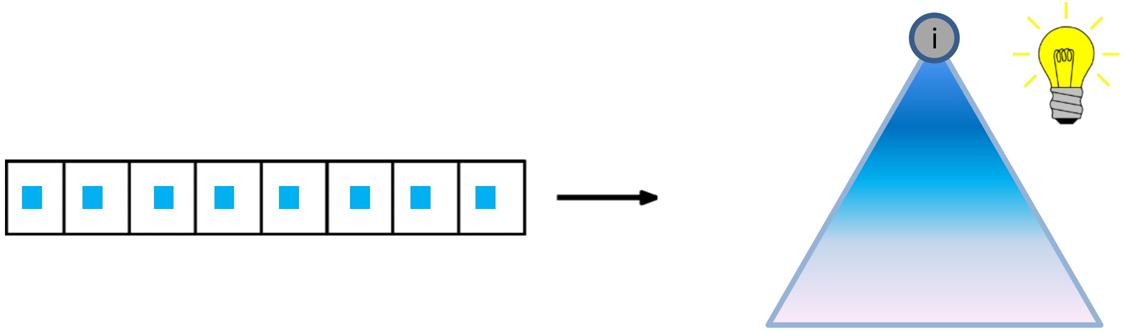
Ego-trees strike back!

Total Recall: Ego-Trees!



$\mathcal{D}[\mathrm{i}]$ $\qquad$ $\mathrm{Ego-Tree}$

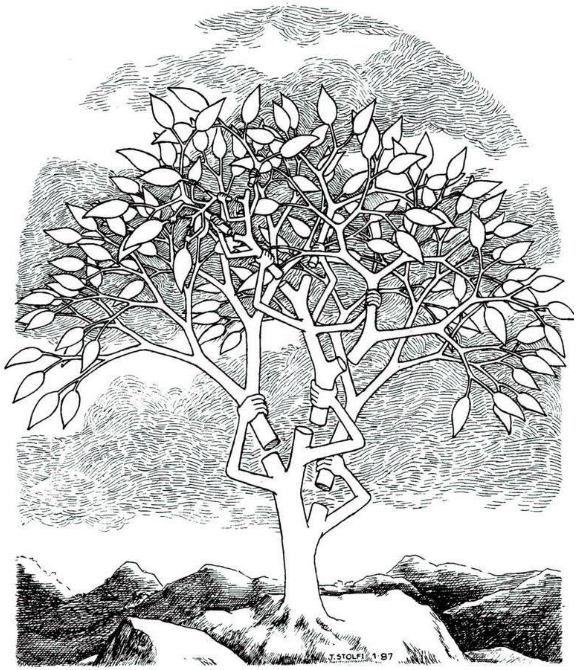
Total Recall: Ego-Trees!



$\mathcal{D}[i]$ $\qquad$ Ego−Tree

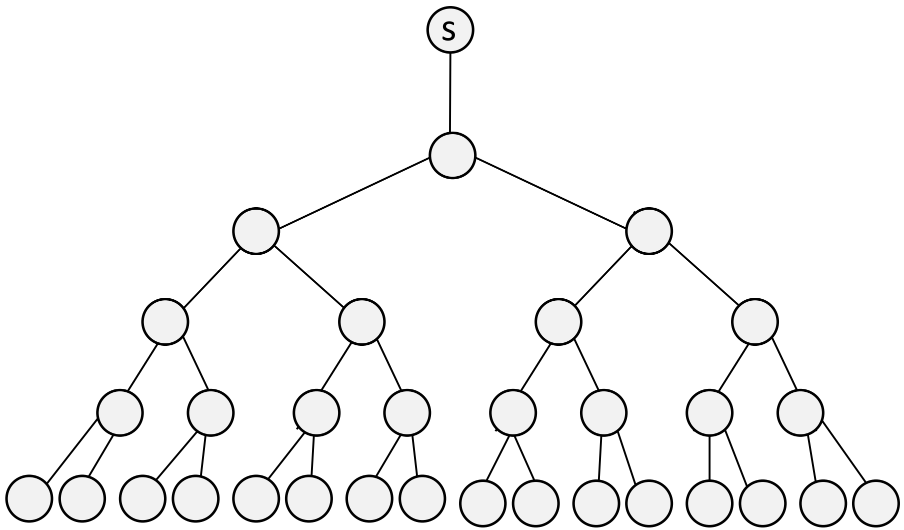
Idea: use our old approach but now let each node ***adjust its ego-tree***!

A Dynamic Ego-Tree: Splay Tree



An Alternative Dynamic Ego-Tree: Push-Down Tree

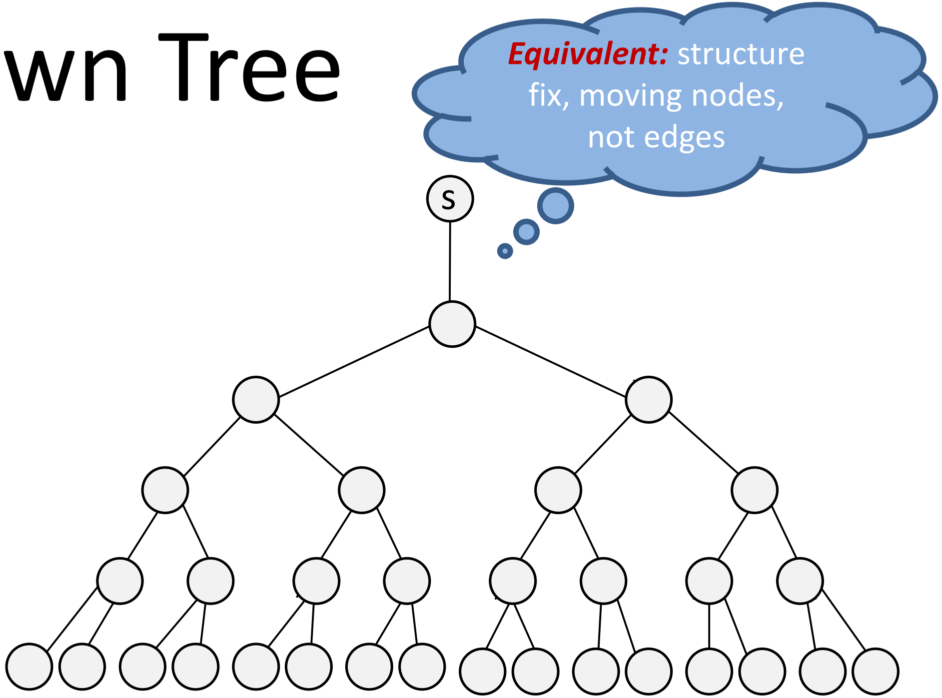
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



An Alternative Dynamic Ego-Tree: Push-Down Tree

Unordered!

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

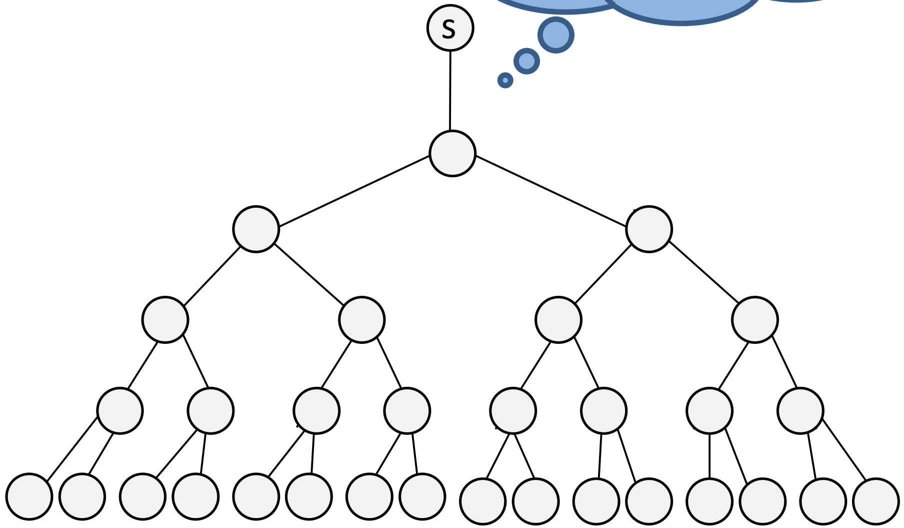


An Alternative Dynamic Ego-Tree: Push-Down Tree

Unordered!

Equivalent: structure fix, moving nodes, not edges

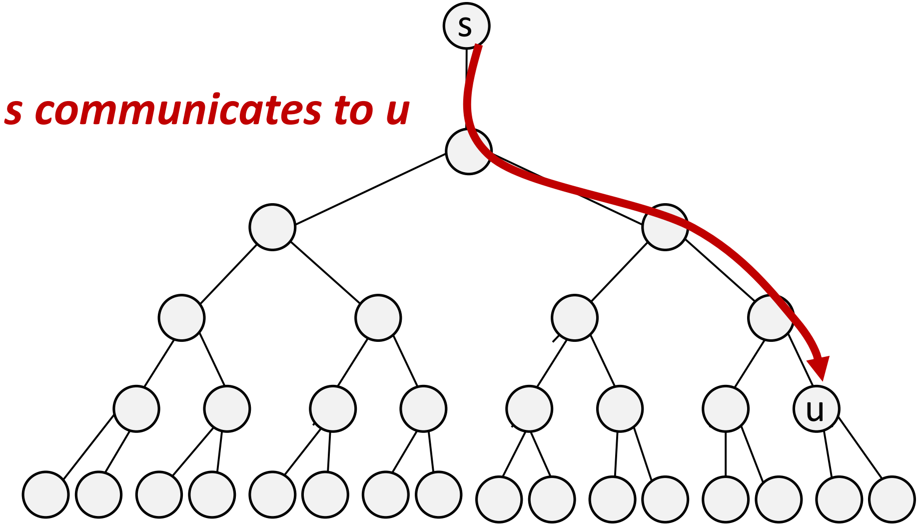
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



A useful dynamic property: **Most-Recently Used (MRU)**!
Similar to **Working Set Property**: more recent communication Partners closer to source.

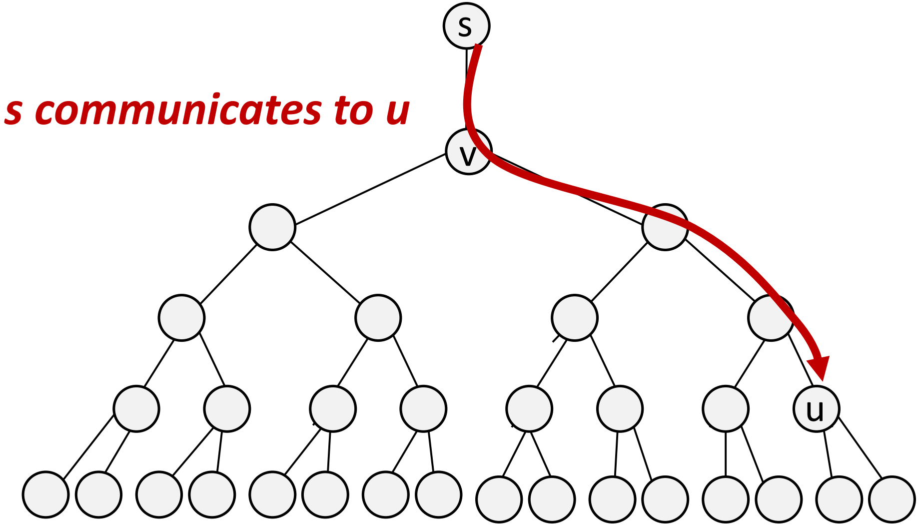
An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



An Alternative Dynamic Ego-Tree: Push-Down Tree

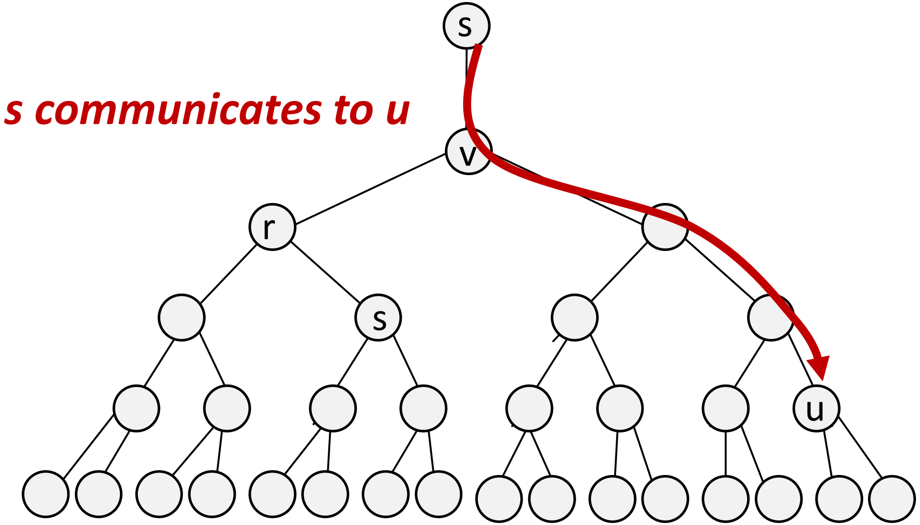
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

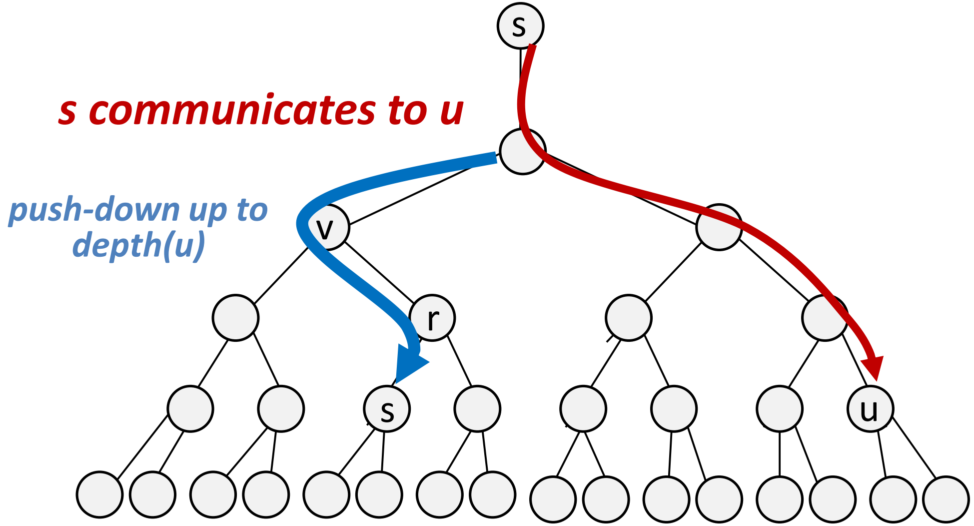


Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

Idea: Push v down, in a balanced manner, up to depth(u): left-right-left-right („rotate-push")

An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

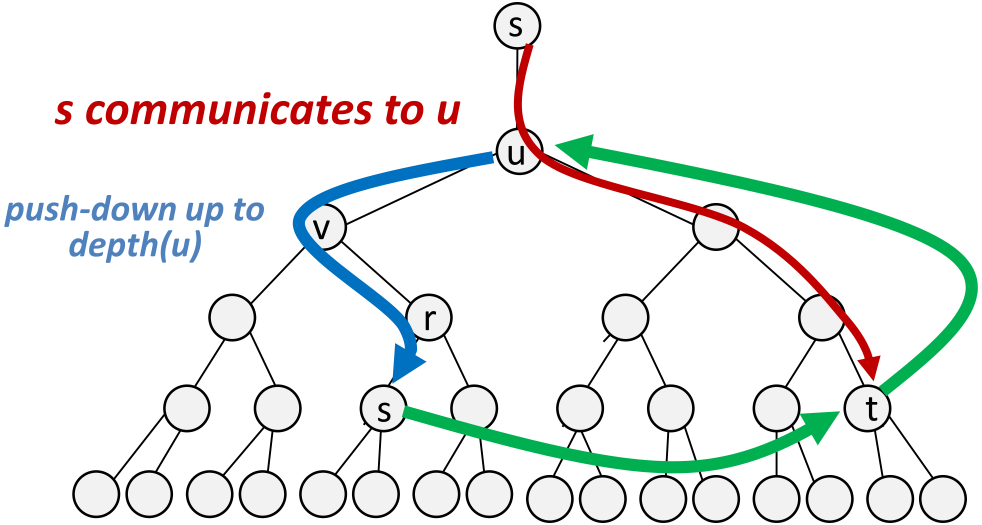


Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!

Idea: Push v down, in a balanced manner, up to depth(u): left-right-left-right („rotate-push")

An Alternative Dynamic Ego-Tree: Push-Down Tree

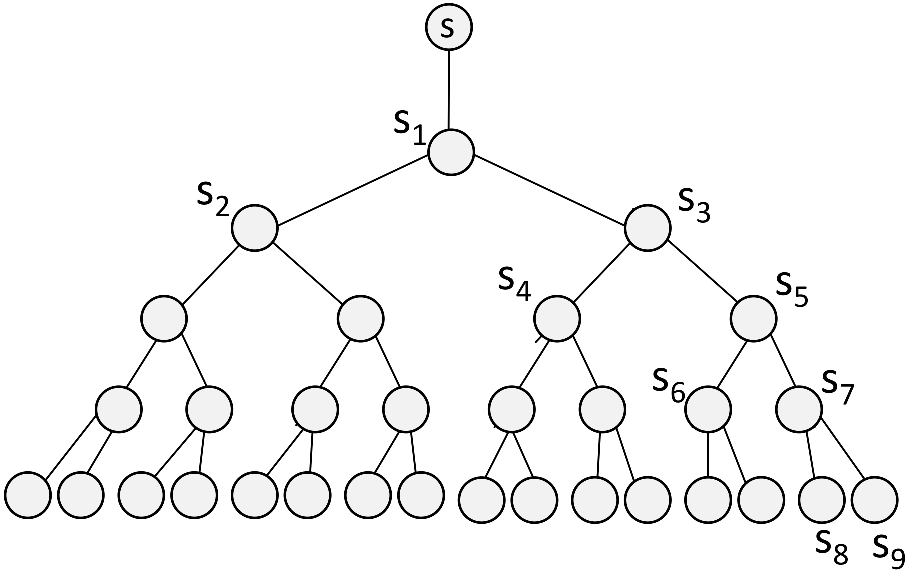
- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



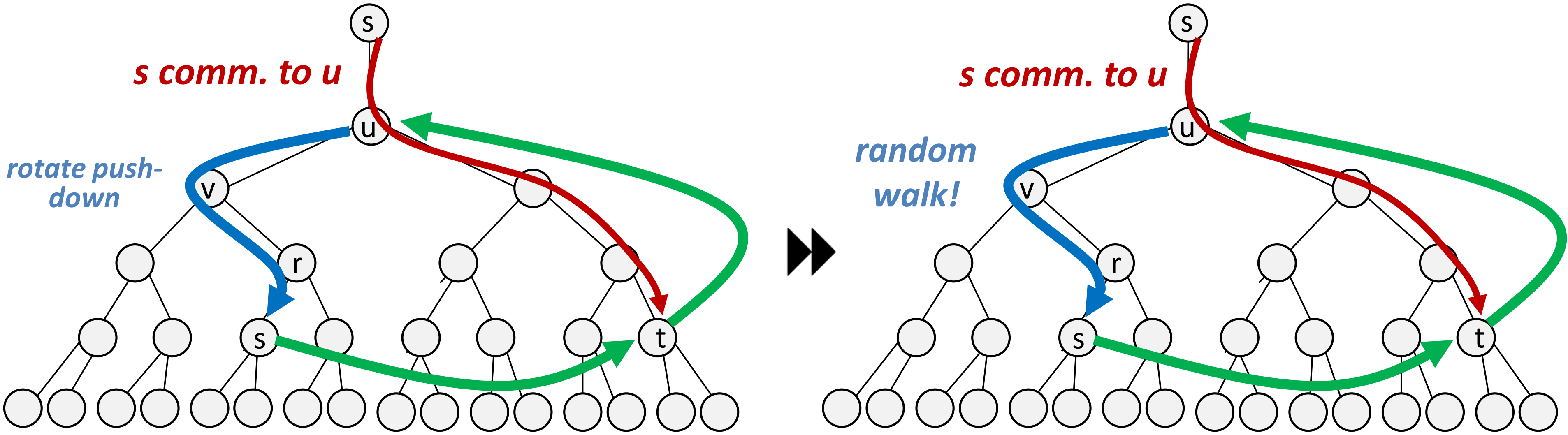
Then: promote u to available root, and t to u: at original depth!

Remarks

- Unfortunately, alternating push-down does ***not maintain MRU*** (working set) property
- Tree can ***degrade***, e.g.: sequence of requests from level 4,1,2,1,3,1,4,1

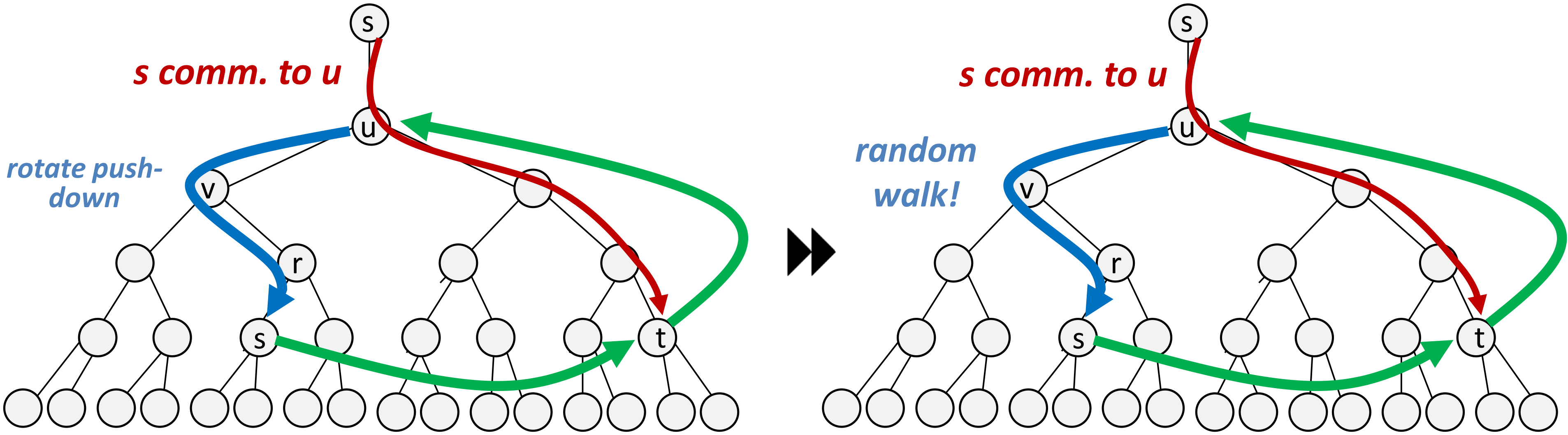


Solution: Random Walk



At least maintains approximate working set / MRU!

Solution: Random Walk



At least maint

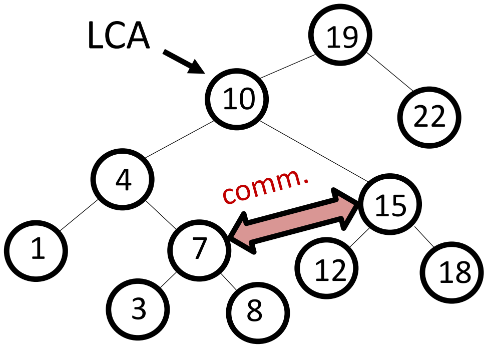
working

Push-Down Trees: Optimal Self-Adjusting Complete Trees
Chen Avin, Kaushik Mondal, and Stefan Schmid.
ArXiv Technical Report, July 2018.

Remark 1: Decentralized Algorithms

A “Simple” Decentralized Solution: Distributed SplayNet (*DiSplayNet*)

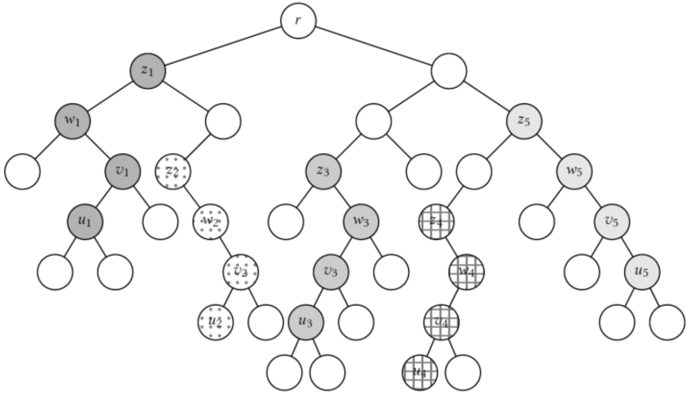
- SplayNet attractive: ordered BST supports **local routing**
  - Nodes ***maintain three ranges***: interval of left subtree, right subtree, upward
- If communicate (frequently): **double-splay** toward LCA
- Challenge: **concurrency**!
  - Access Lemma of splay trees no longer works: ***potential function*** does not „***telescope***“ anymore: a concurrently rising node may push down another rising node again



SplayNet

DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:



	1	2	3	4	5	6	7	8	...	$i-6$	$i-5$	$i-4$	$i-3$	$i-2$	$i-1$	i
σ_1	✓	✓	✓	✓	-	-	-	-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	-	...	✓	✓	-	-	-	-	-
σ_m	-	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-

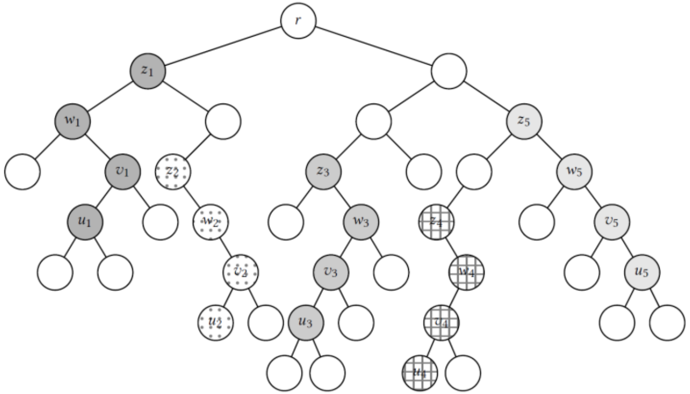
Sequential SplayNet: requests ***one after another***

	1	2	3	...	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	...	j	...	k
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

DiSplayNet: Analysis more challenging: potential function sum no longer **telescopic**. One request can "push-down" another.

DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:



Telescopic: max potential drop

	1	2	3	4	5	6	7	8	...	$i-6$	$i-5$	$i-4$	$i-3$	$i-2$	$i-1$	i
σ_1	✓	✓	✓	✓	[illegible]	[illegible]	[illegible]	-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	[illegible]	[illegible]	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	-	...	✓	✓	[illegible]	[illegible]	[illegible]	[illegible]	-
σ_m	-	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-

	1	2	3	...	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	...	j	...	k
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

Sequential SplayNet: requests ***one after another***

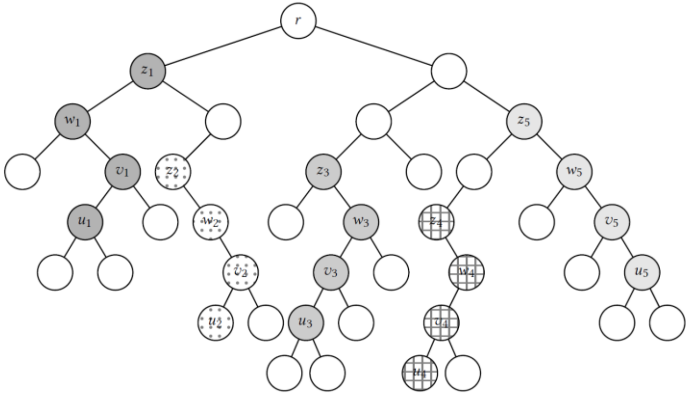
DiSplayNet: Analysis more challenging: potential function sum no longer **telescopic**. One request can “push-down” another.

DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzig,zigzag) are ***concurrent***
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:

Telescopic: max potential drop

	1	2	3	4	5	6	7	8	...	$i-6$	$i-5$	$i-4$	$i-3$	$i-2$	$i-1$	i
σ_1	✓	✓	✓	✓				-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...		...	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	-	...	✓	✓					-
σ_m	-	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-



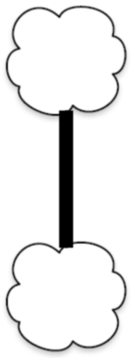
	1	2	3	...	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$	...	j	...	k
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

Sequential SplayNet: re

Distributed Self-Adjusting Tree Networks. Bruna Peres, Otavio Augusto de Oliveira Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid. IEEE **INFOCOM**, 2019.

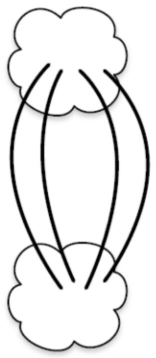
Remark 3: Accounting for Congestion

A Tradeoff?!



Short routes:
congestion

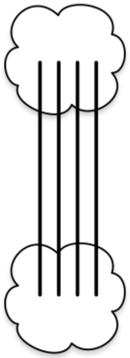
vs



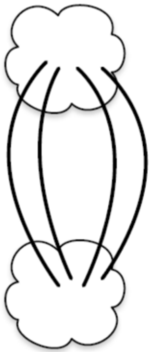
Low congestion:
long routes

A Tradeoff?!

Short routes:
congestion

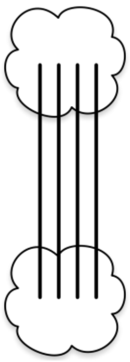


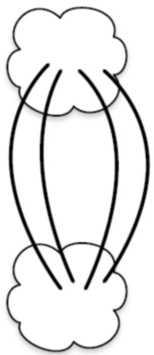
Or both?



Low congestion:
long routes

A Tradeoff?!





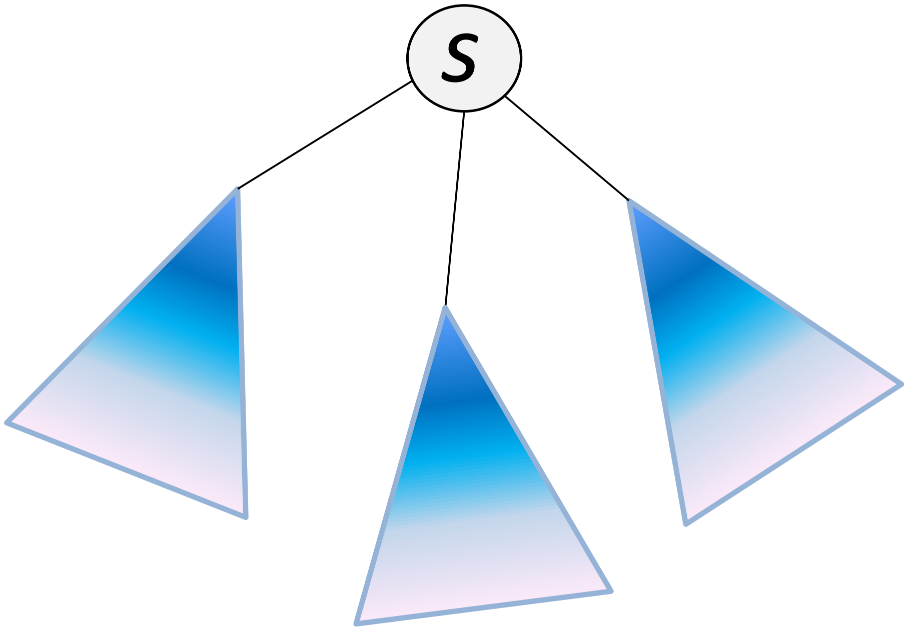
Short routes: congestion

Or both?

Low congestion: long routes

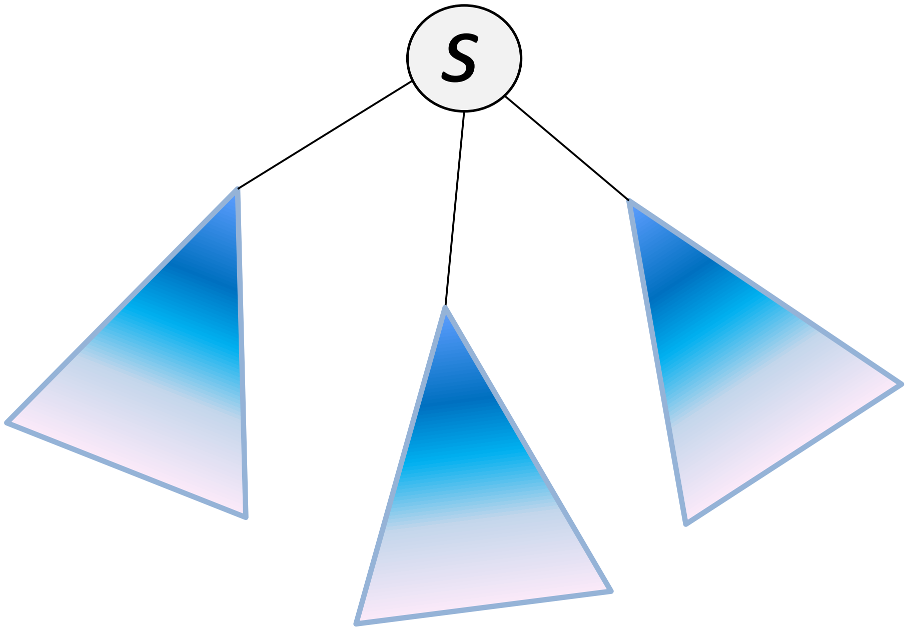
Ego-Tree++!

- Idea: place destination nodes greedily across subtrees s.t. ***congestion balanced***
- … while preserving distance
- Trees can have different sizes but ***similar mass***!
- Bicriteria guarantee



Ego-Tree++!

- Idea: place destination nodes greedily across subtrees s.t. ***congestion balanced***
- … while preserving distance
- Trees can have different sizes but ***similar mass***!
- Bicriteria guarantee



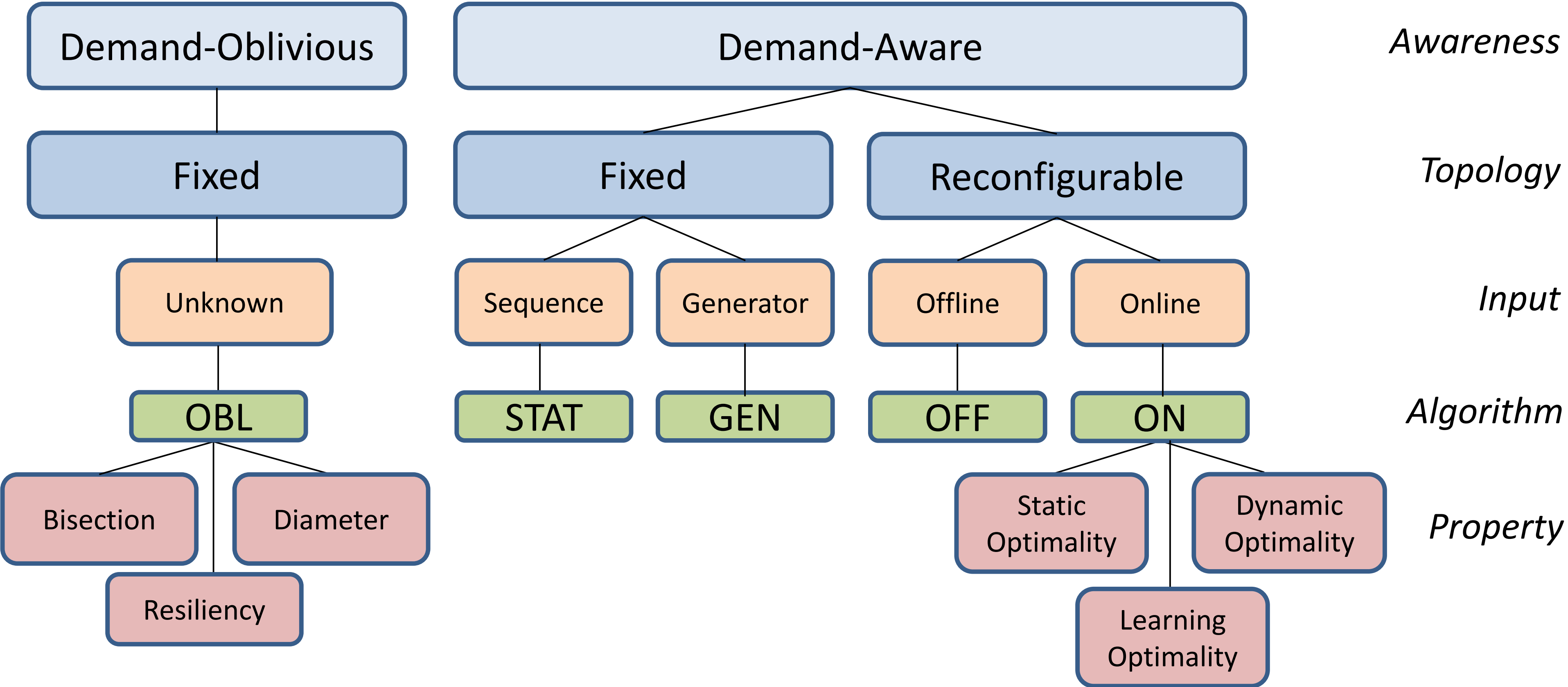
Demand-Aware Network Design with Minimal Congestion and Route Lengths. Chen Avin, Kaushik Mondal, and Stefan Schmid. IEEE **INFOCOM** 2019.

Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures

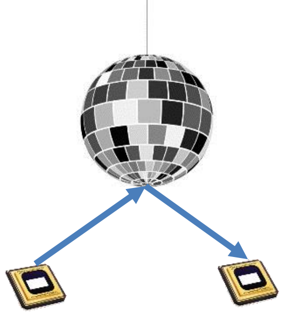
Uncharted Landscape!

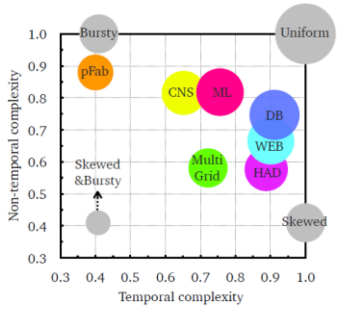
Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. **SIGCOMM CCR,** 2018.



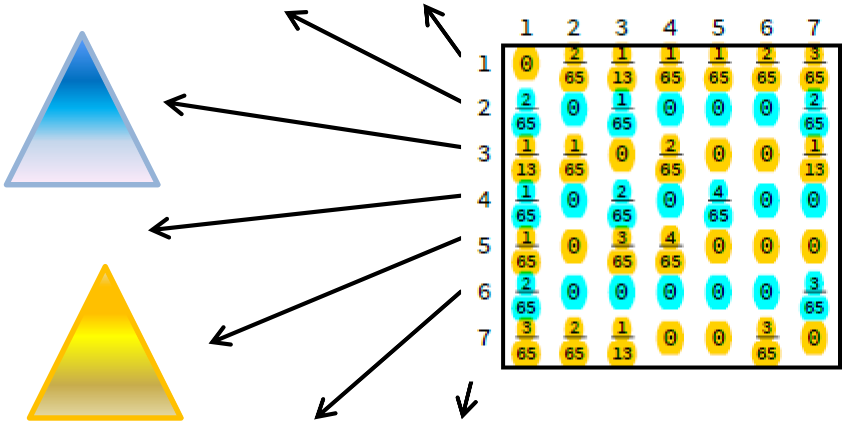
Big Open Questions

- Cross-layer aspects
- Metrics: just the beginning!
- We need more data
- Unifying theory
- How to convince operators?





Thank you! Questions?



Further Reading

Demand-Aware and Self-Adjusting Networks

Survey of Reconfigurable Data Center Networks: Enablers, Algorithms, Complexity
Klaus-Tycho Foerster and Stefan Schmid.
SIGACT News, June 2019.

Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks (Editorial)
Chen Avin and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), October 2018.

Demand-Aware Network Design with Minimal Congestion and Route Lengths
Chen Avin, Kaushik Mondal, and Stefan Schmid.
38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.
Documents: paper pdf, bibtex bib

Distributed Self-Adjusting Tree Networks
Bruna Peres, Otavio Augusto de Oliveira Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid.
38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks
Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anaïs Villedieu.
IFIP Networking, Warsaw, Poland, May 2019.

DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks
Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu.
IEEE/ACM International Symposium on Quality of Service (**IWQoS**), Phoenix, Arizona, USA, June 2019.

Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

SplayNet: Towards Locally Self-Adjusting Networks
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.
IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.

Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures
Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.
ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.