Principles of self-* communication networks: Data-driven optimization and verification

Stefan Schmid (TU Berlin)

AI in Networking
Summer School 2022

Self-* Networks: Why?

Self-* Networks: Why?

*… where *={configuring, repairing, optimizing, “driving”, …}*

Self-* Networks: Why?

*… where *={**configuring**, repairing, optimizing, "driving", …}*

Networks Are Complex: Even Tech-Savvy Companies Struggle

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm.

Service outage was due to a series of internal network events that corrupted router data tables.

Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems

Networks Are Complex: Even Tech-Savvy Companies Struggle

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was […] executed incorrectly […] more "stuck" volumes and added more requests to the re-mirroring storm.

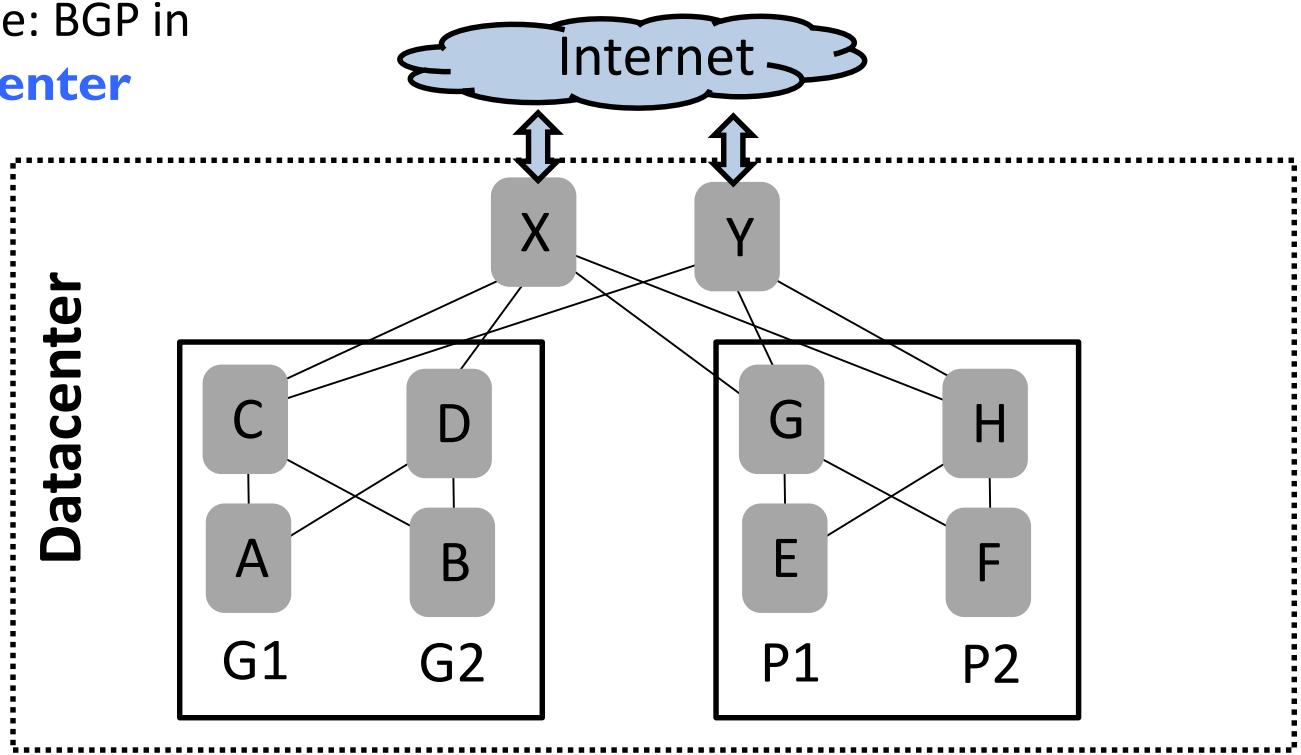
Service outage was due to a series of internal network events that corrupted router data tables.

Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems

Most of them: due to human errors.

Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

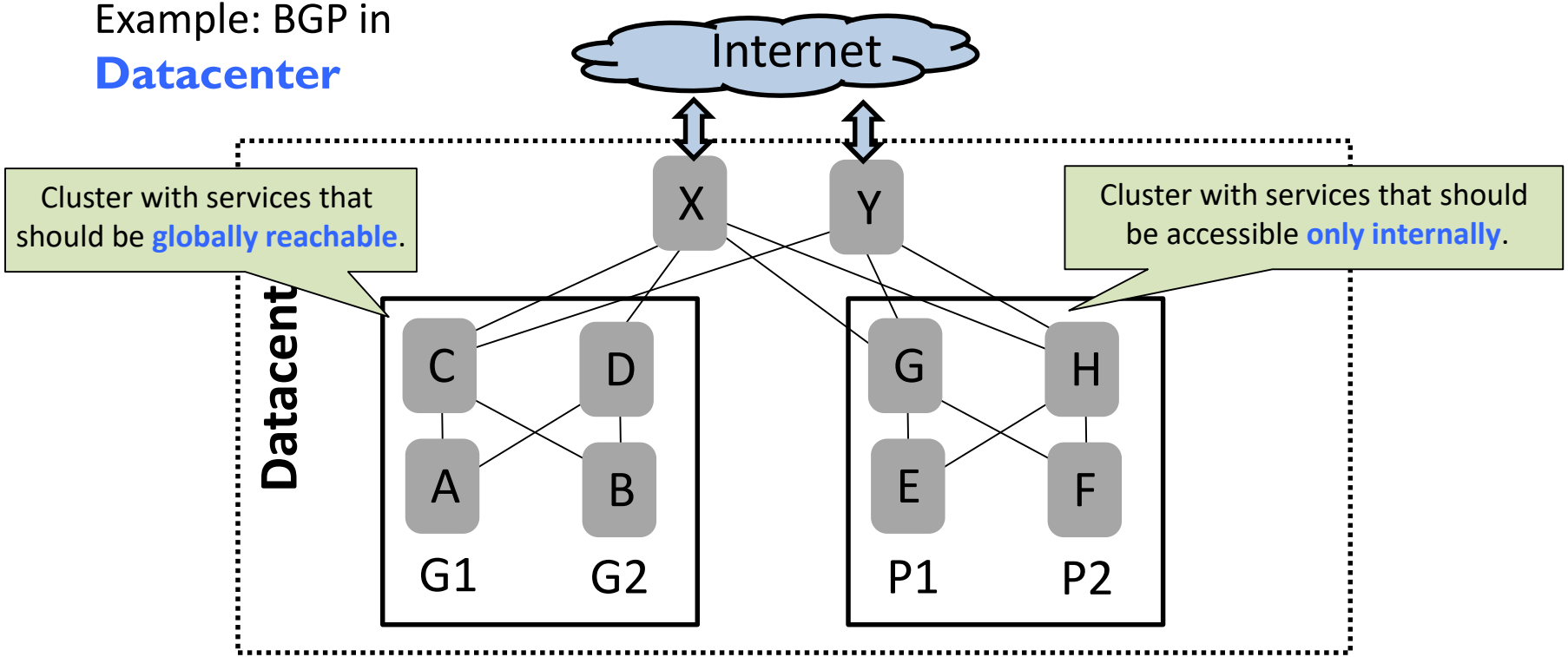
Example: BGP in **Datacenter**



Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

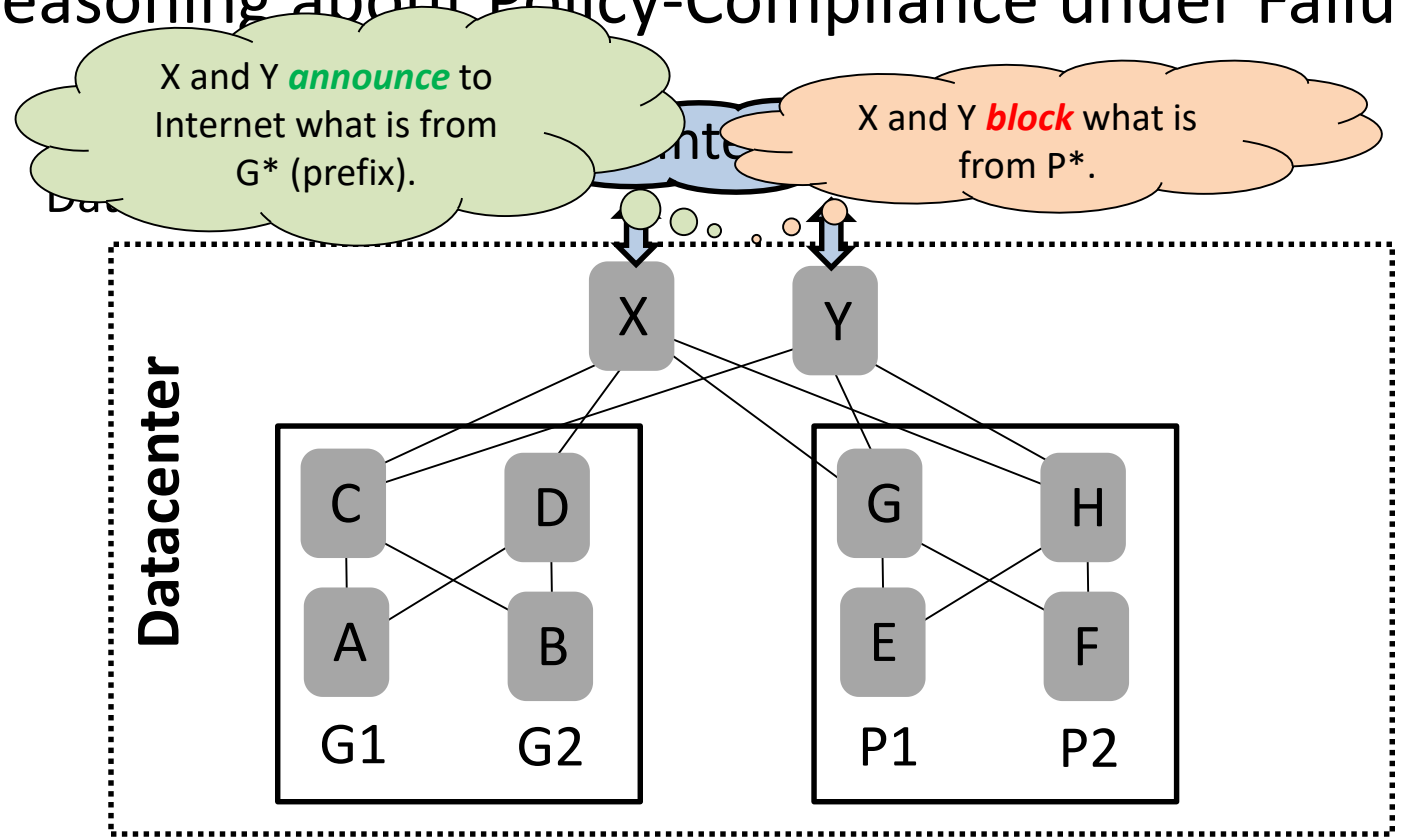
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures

Example: BGP in **Datacenter**



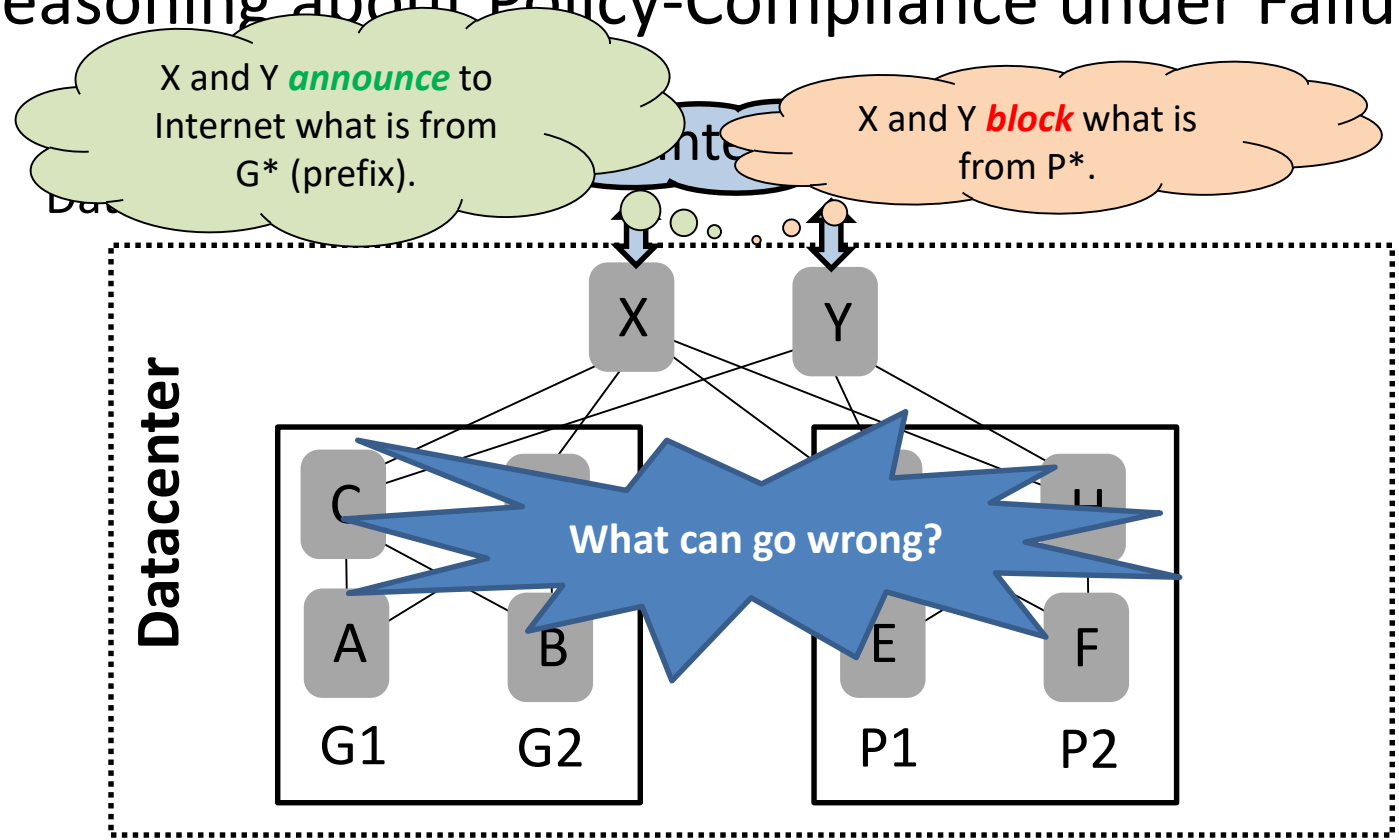
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



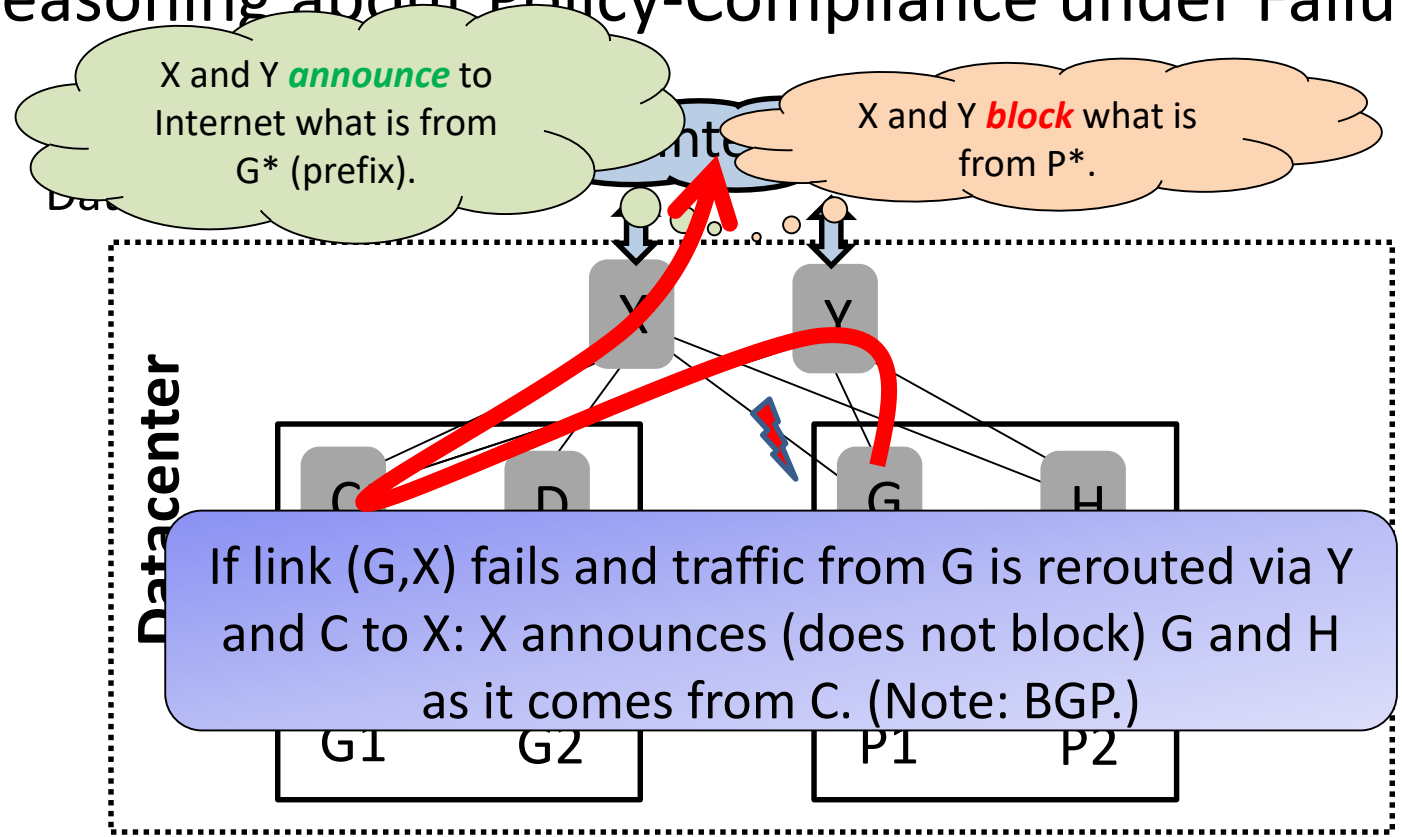
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

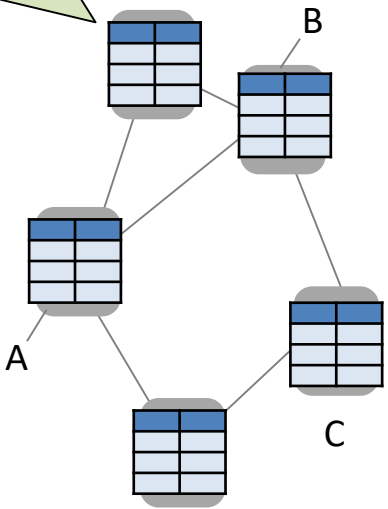
Particularly Challenging for Humans: Reasoning about Policy-Compliance under Failures



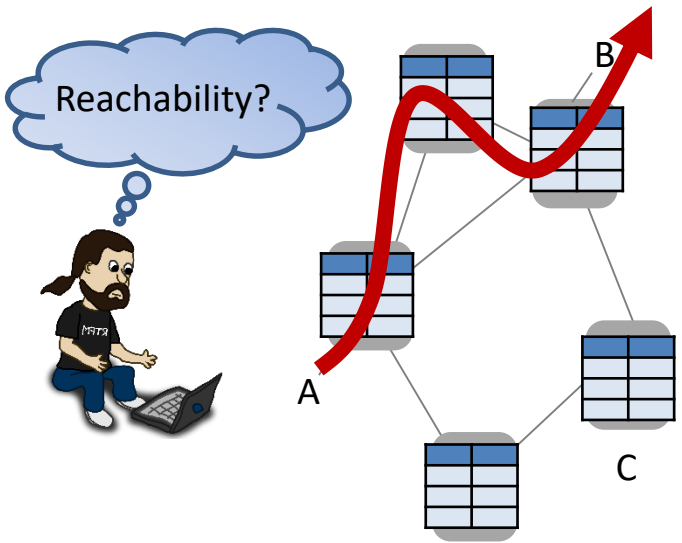
Credits: Beckett et al. (SIGCOMM 2016): Bridging Network-wide Objectives and Device-level Configurations.

Responsibilities of a Sysadmin

Routers and switches store list of forwarding rules, and conditional failover rules.



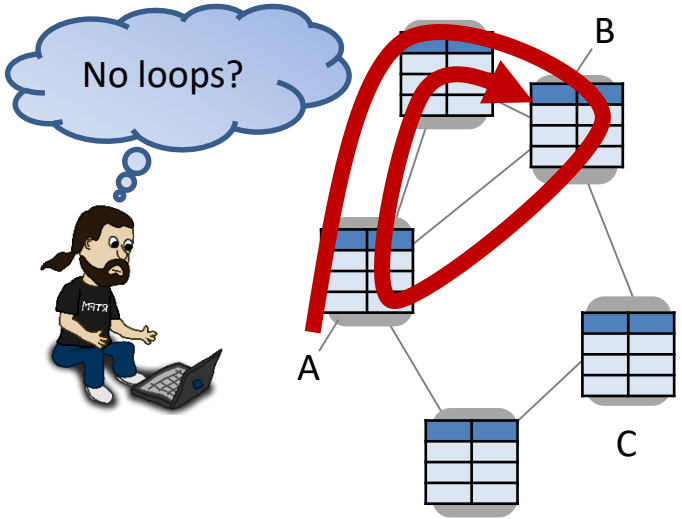
Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?

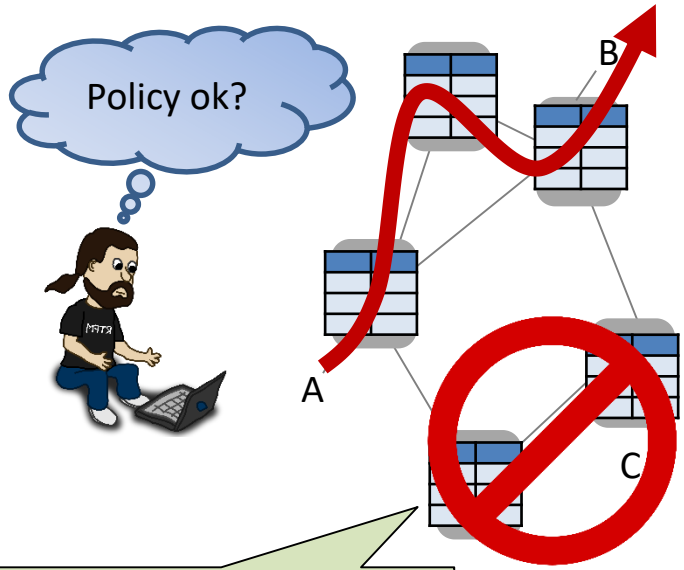
Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?

Responsibilities of a Sysadmin

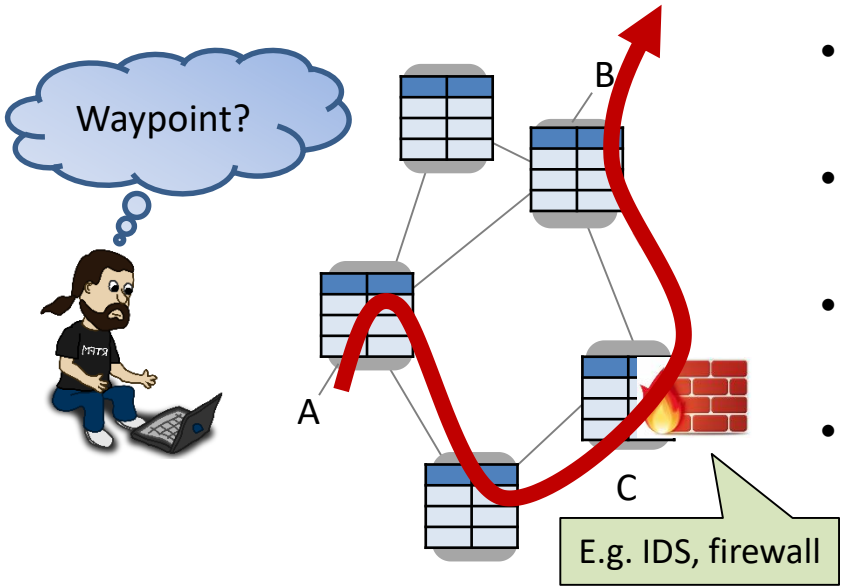


E.g. ***NORDUnet***: no traffic via Iceland (expensive!). Or no traffic through ***route reflectors***.

Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?

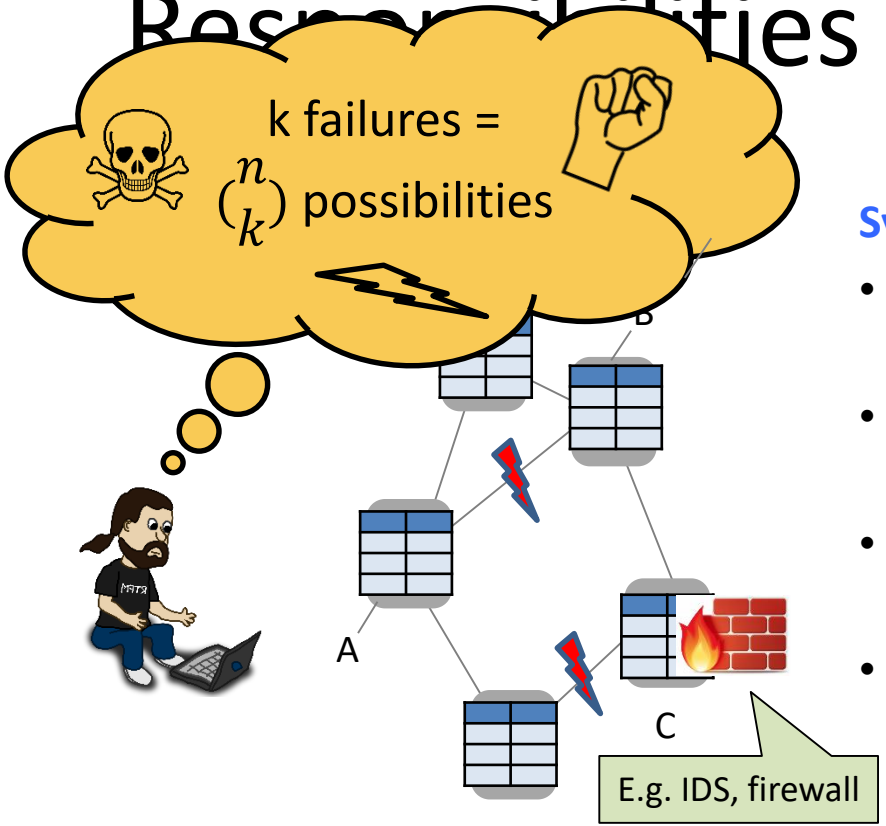
Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C?

Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C?

… and everything even under multiple failures?!

Can we automate such tests
or even self-repair?

Another Reason:
Flexibilities and Optimization Opportunities

Another Reason:
Flexibilities and Optimization Opportunities

*… where *={configuring, repairing, **optimizing**, “driving”, …}*

Flexibilities: Along 3 Dimensions

Somewhere in beautiful Germany…

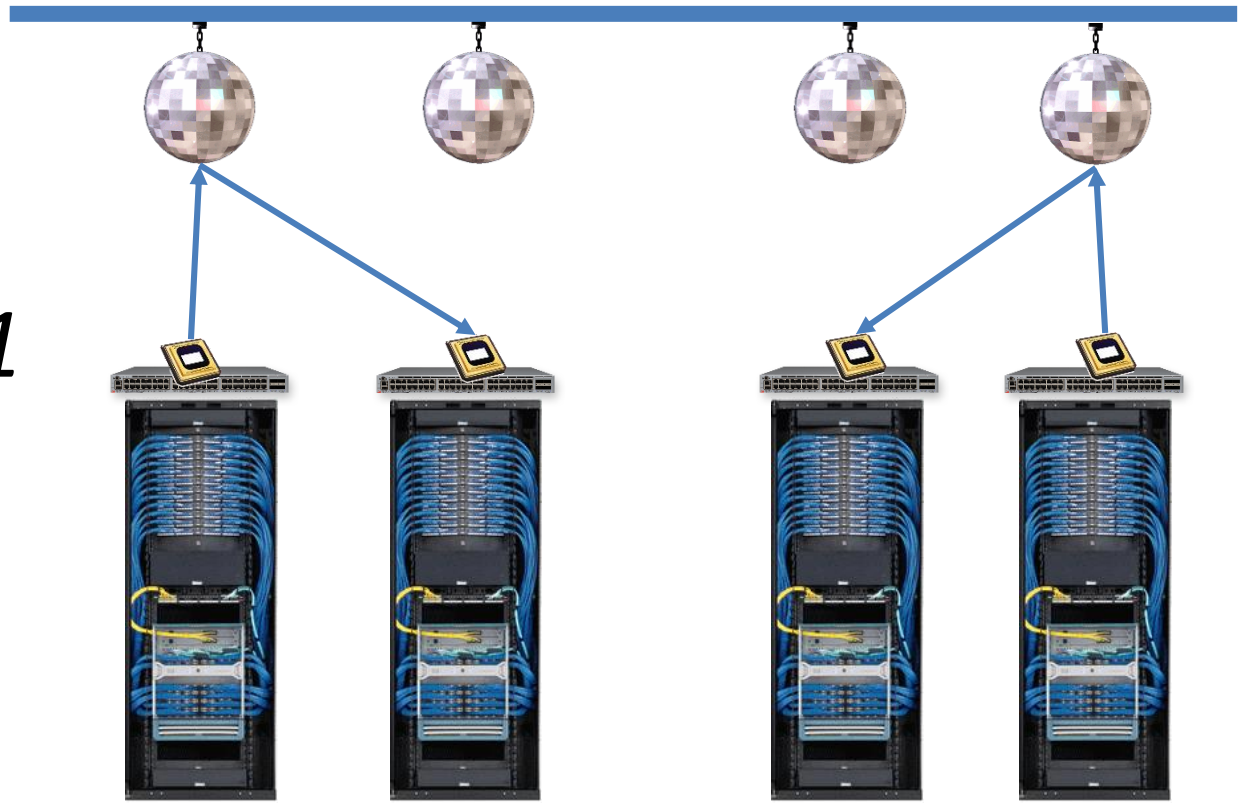
Flexibilities: Along 3 Dimensions

Somewhere in beautiful Germany…

Flexibilities: Along 3 Dimensions

ful Germany…

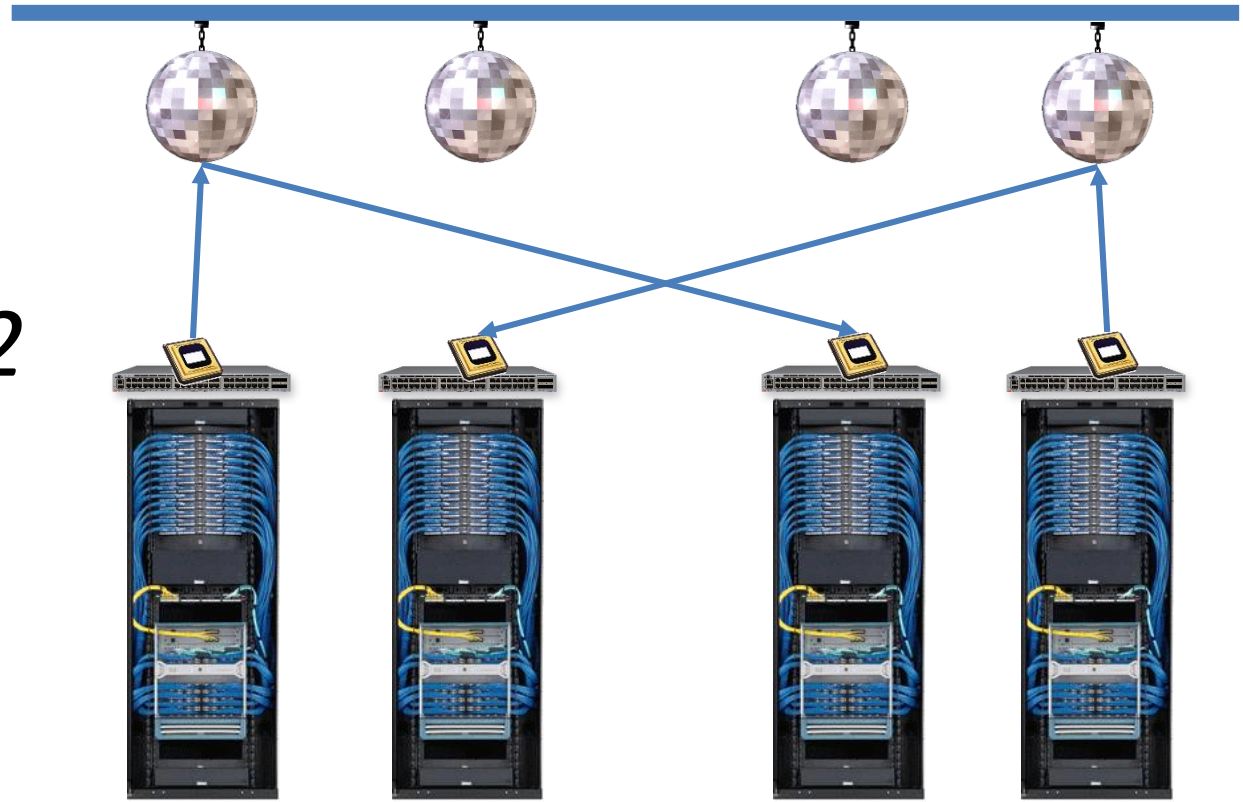
Example: ProjecToR



t=1

- Based on **free-space optics**

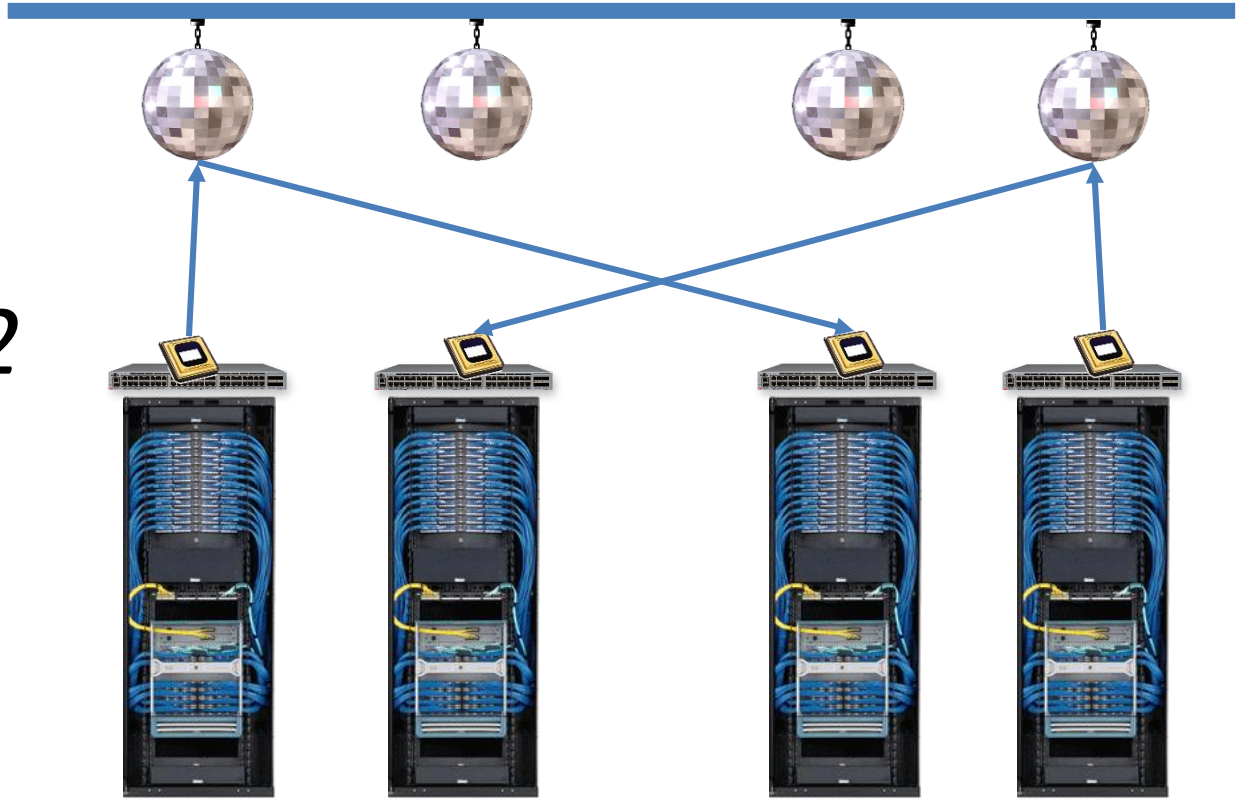
Example: ProjecToR



t=2

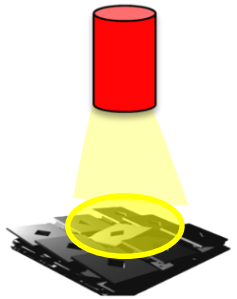
- Based on **free-space optics**

Example: ProjecToR



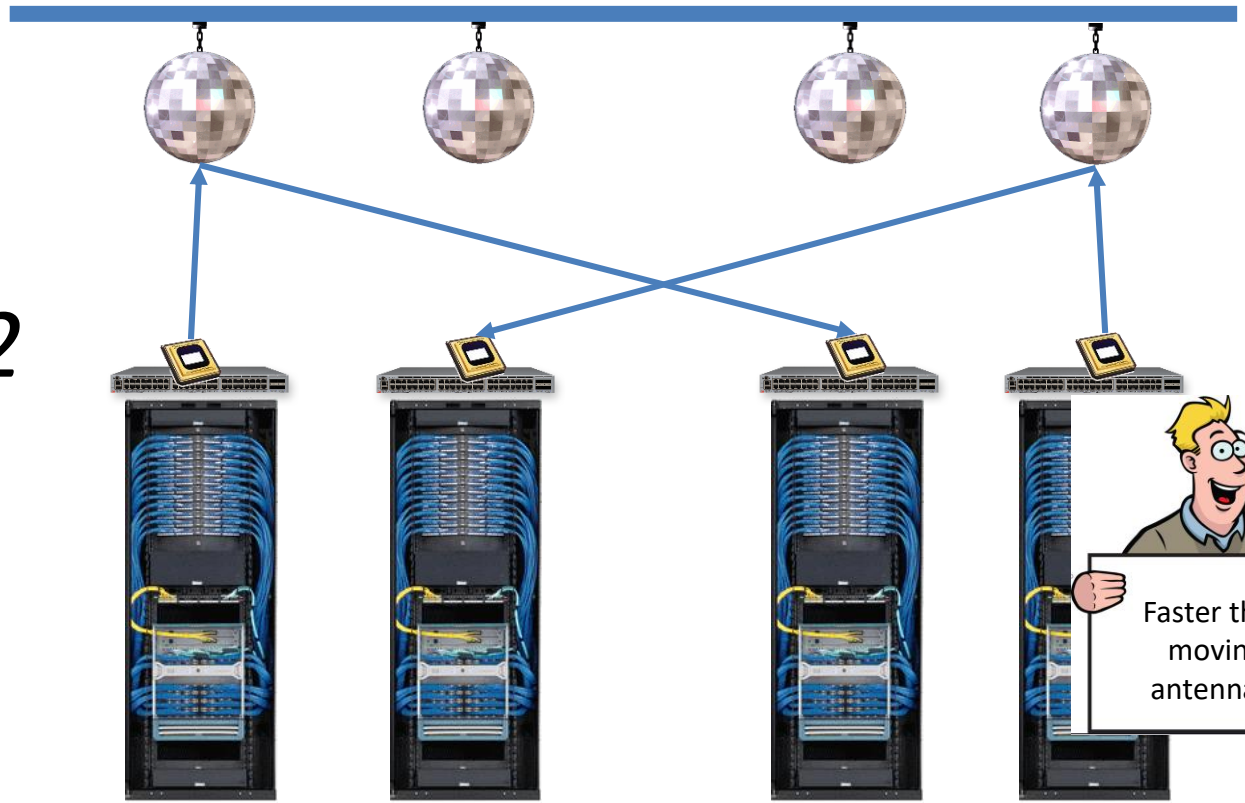
t=2

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



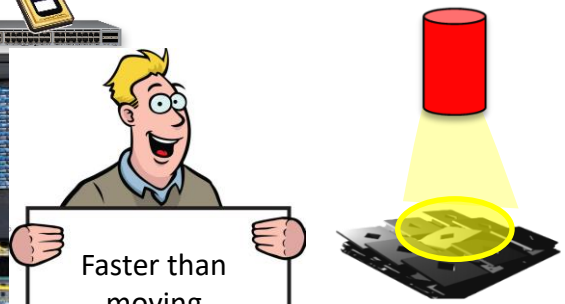
Digital Micromirror Devices (DMDs)

Example: ProjecToR



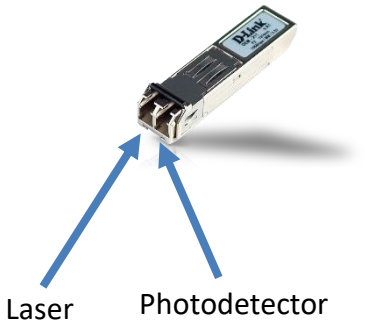
t=2

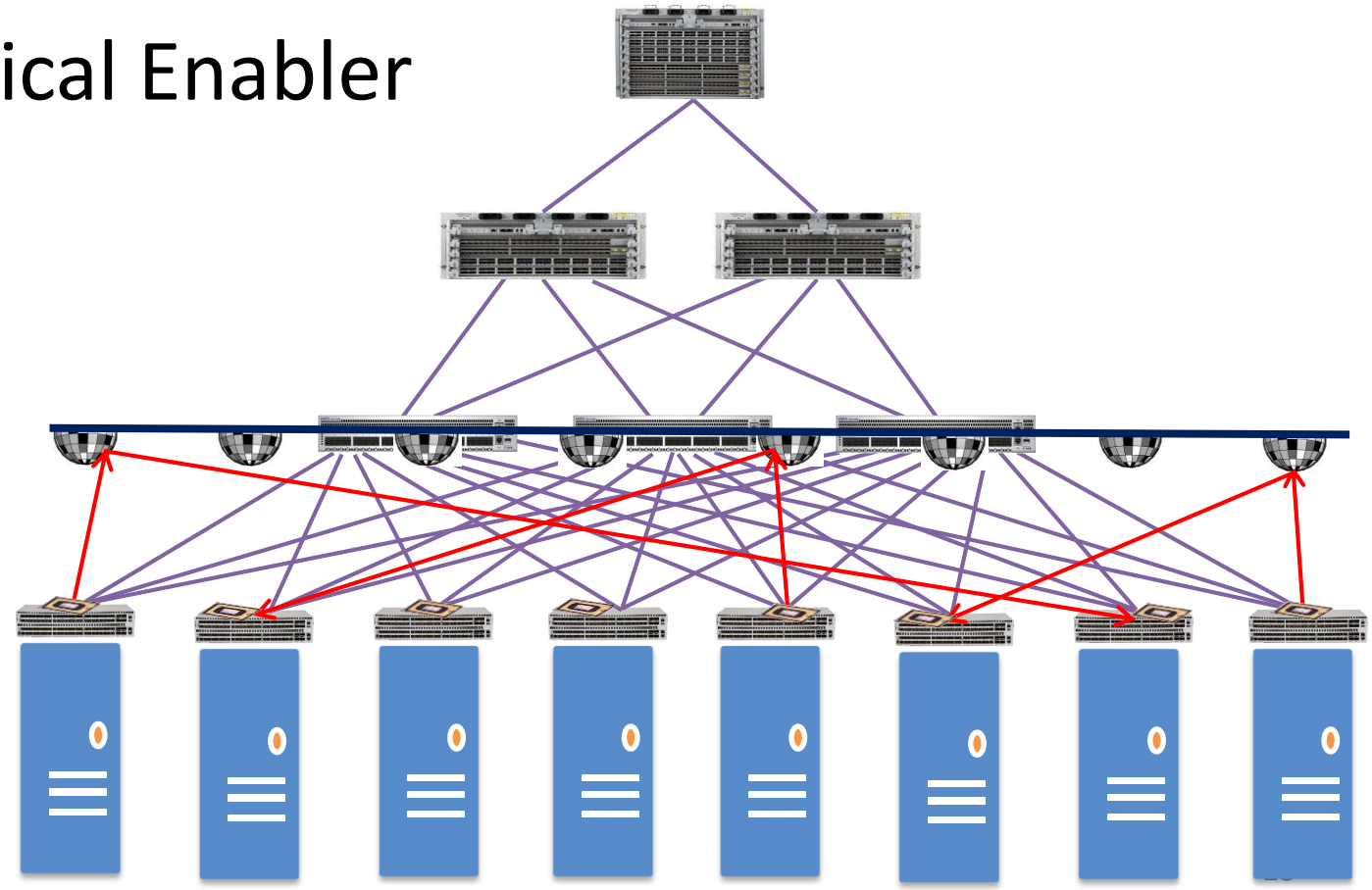
- Based on **free-space optics**
- Reconfiguration in ~10 µs:



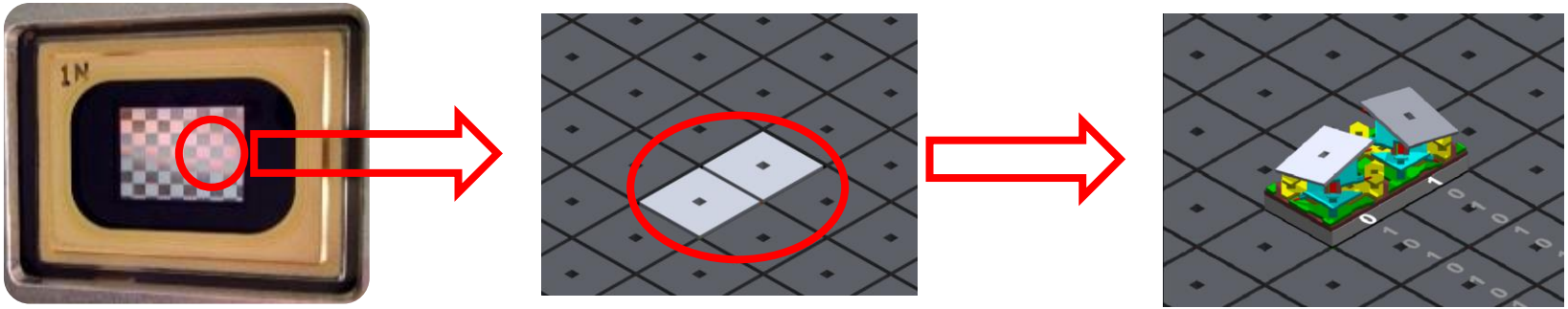
l Micromirror Devices (DMDs)

ProjecToR in More Details: Technological Enabler





ProjecToR in More Details: DMDs



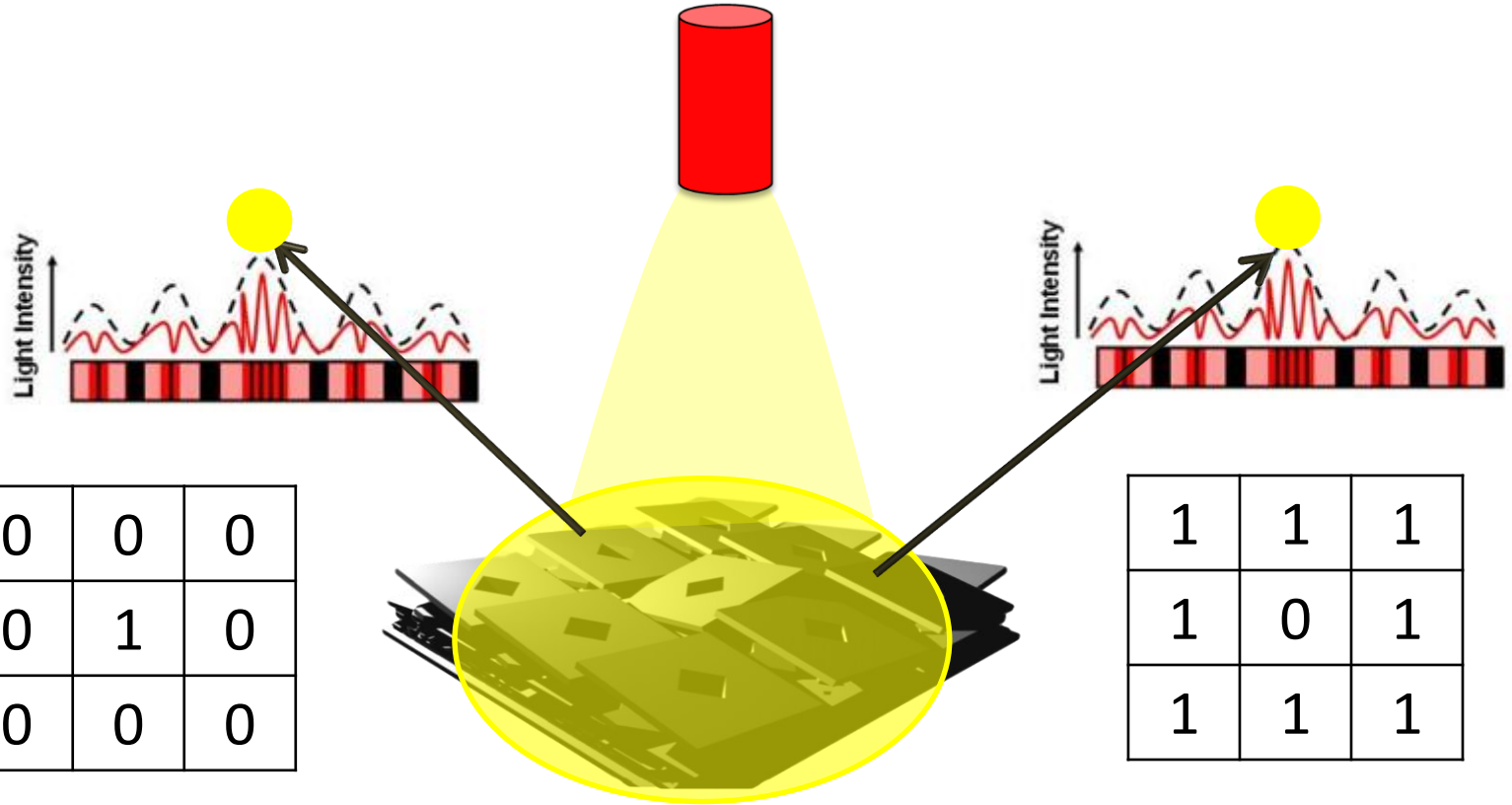
Texas Instruments

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

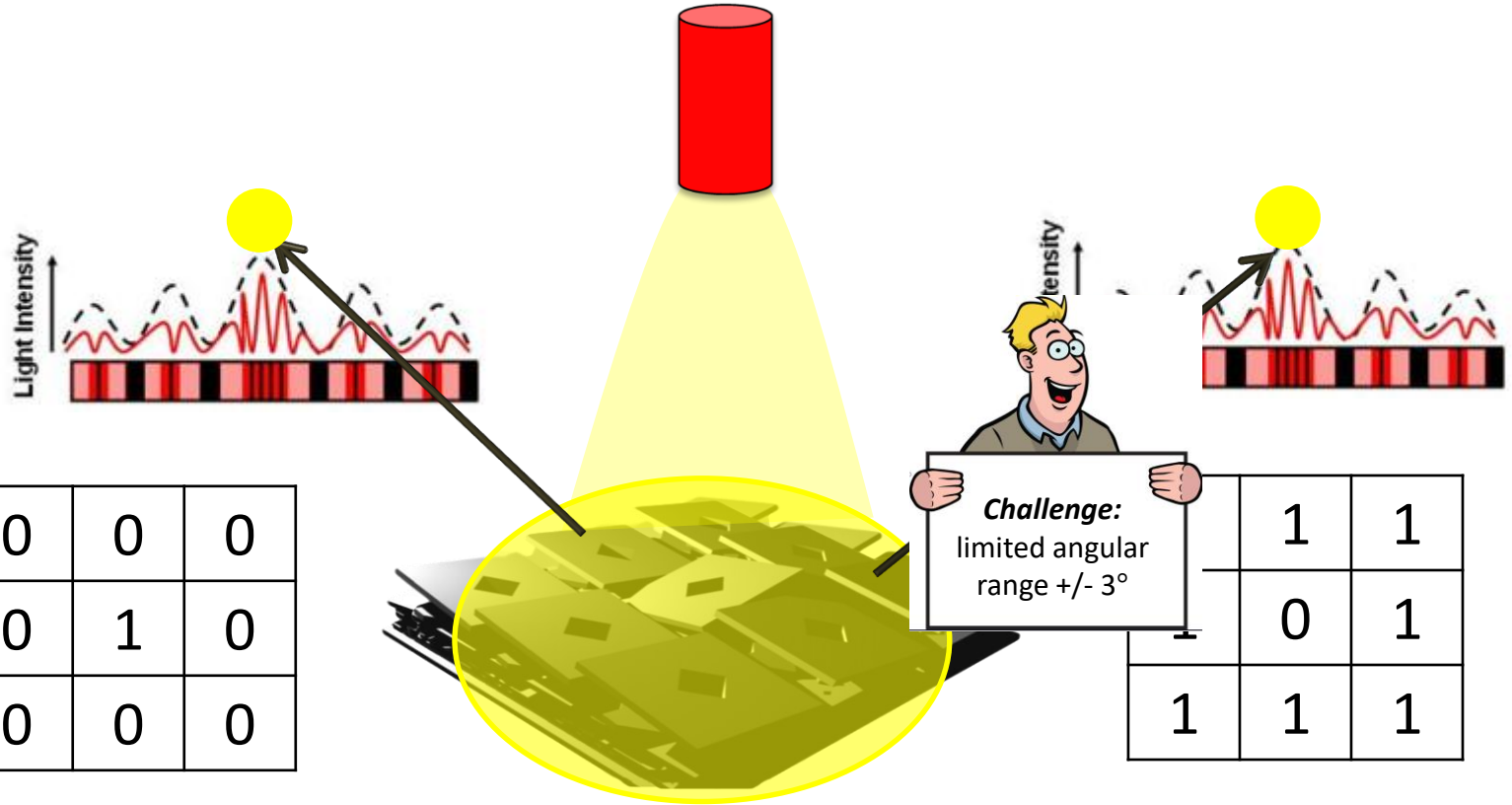
ProjecToR in More Details: DMDs to Redirect Light *Fast*



0	0	0
0	1	0
0	0	0

1	1	1
1	0	1
1	1	1

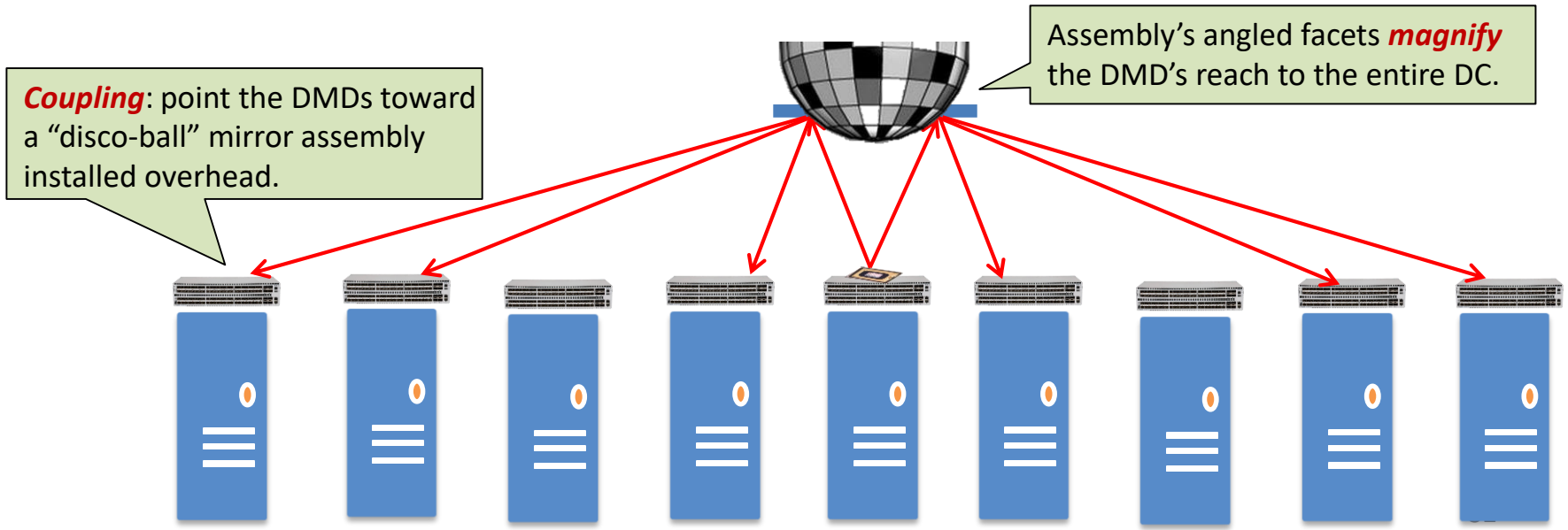
ProjecToR in More Details: DMDs to Redirect Light *Fast*



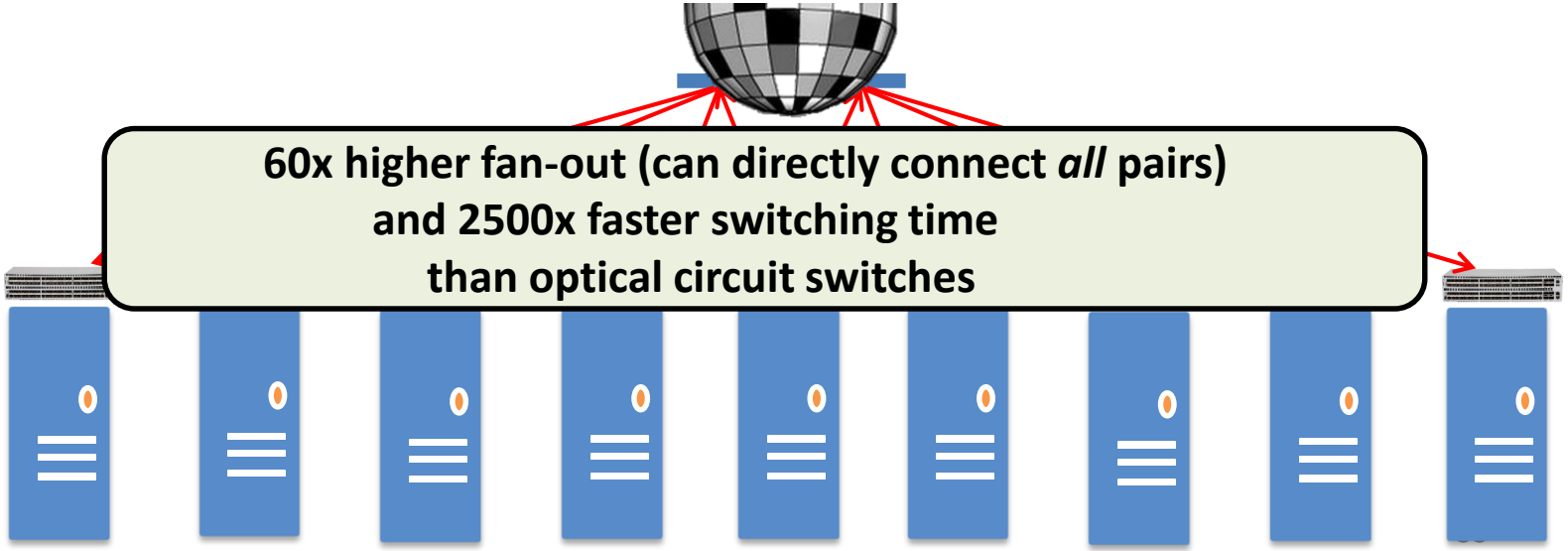
0	0	0
0	1	0
0	0	0

	1	1
	0	1
1	1	1

ProjecToR in More Details: Coupling DMDs with angled mirrors



ProjecToR in More Details: Coupling DMDs with angled mirrors



Empirical Motivation

Empirical Motivation: Structure

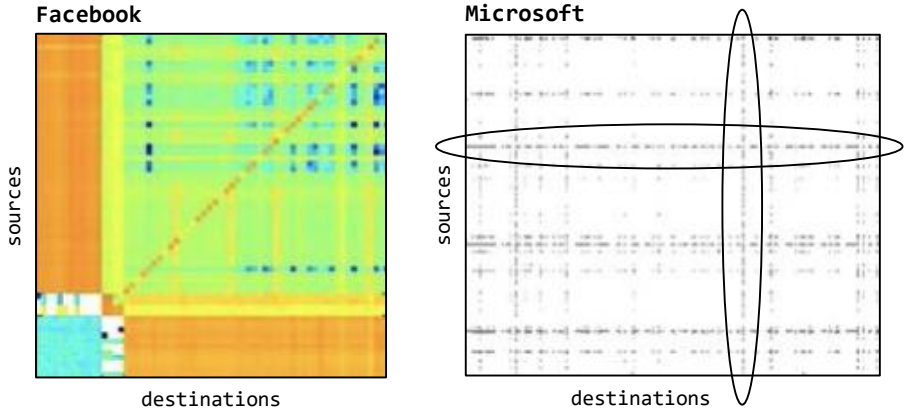
"less than 1% of the rack pairs account for 80% of the total traffic"

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

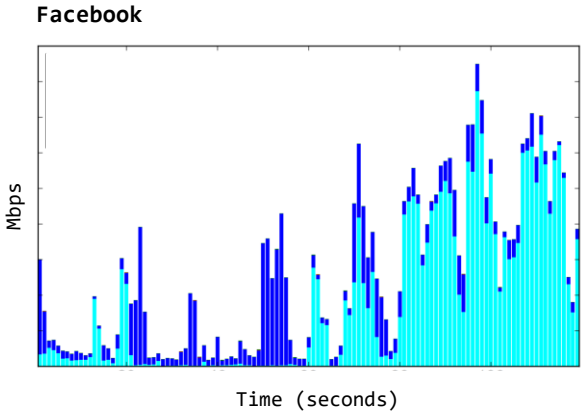
"over 90% bytes flow in elephant flows"

Spatial and Temporal Locality

traffic matrices sparse and skewed

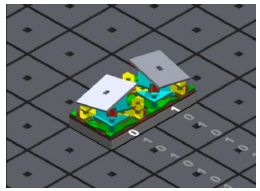


traffic bursty over time

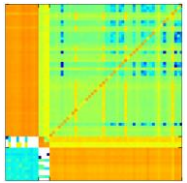


The Vision

Flexibility



Structure



Self-Adjusting Networks

Efficiency

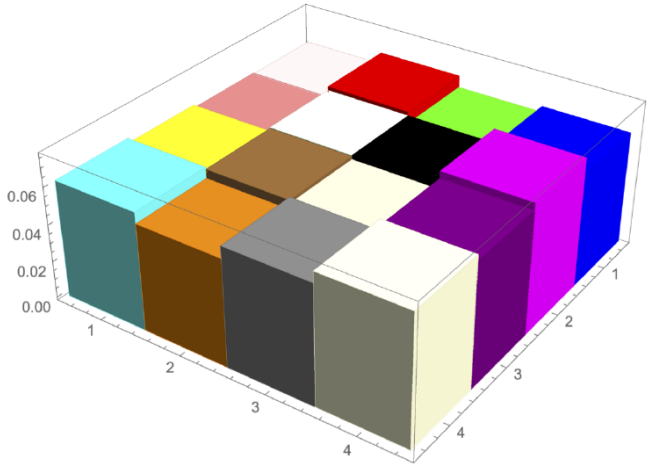
Roadmap

- How much structure is there in the data? A systematic approach.
- Exploiting structure in data: an example
- Self-repairing networks

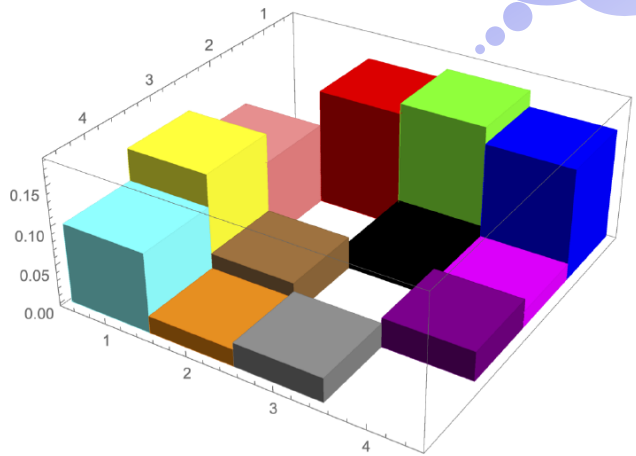
Roadmap

- **How much structure is there in the data? A systematic approach.**
- Exploiting structure in data: an example
- Self-repairing networks

Non-temporal Structure: It‘s Intuitive!



vs

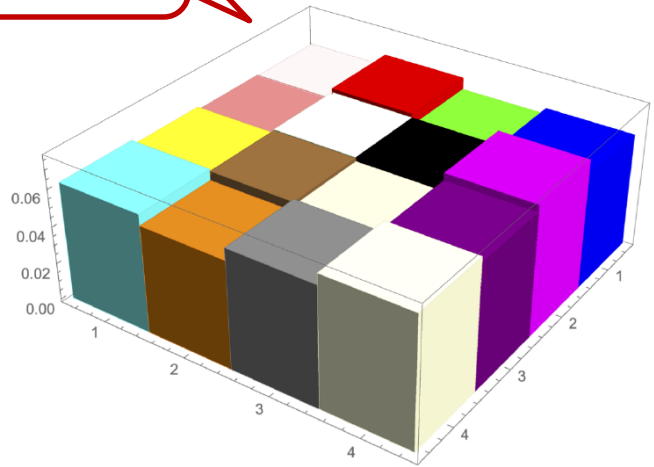


Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):

Which one has *more structure*?

Non-temporal Structure: It's Intuitive!

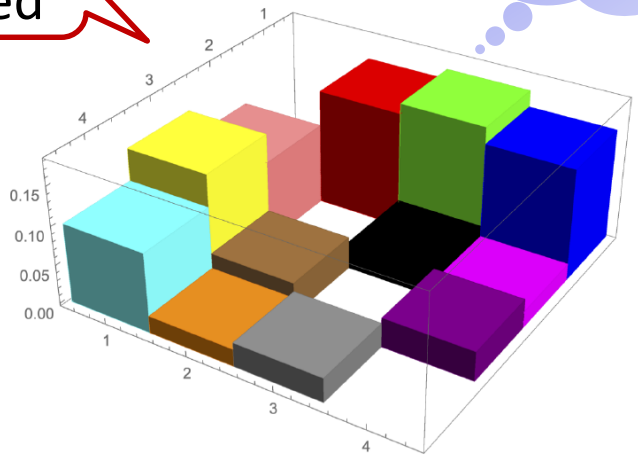
More uniform



vs

More skewed

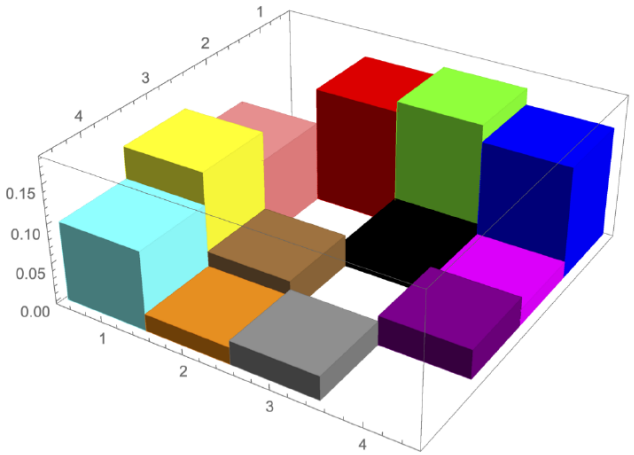
Color = comm. pair

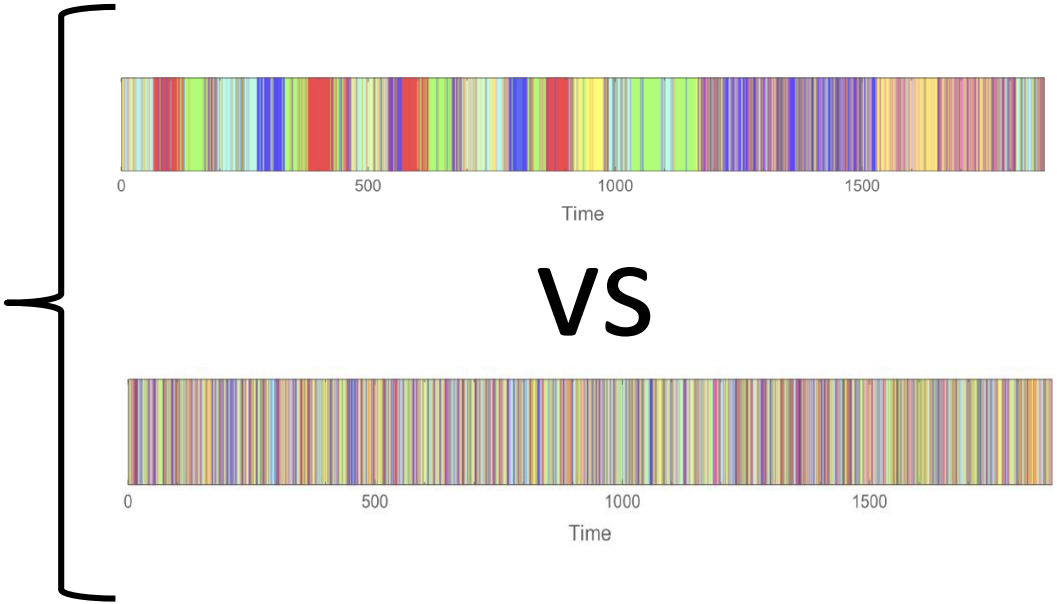


Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):

Which one has *more structure*?

Temporal Structure

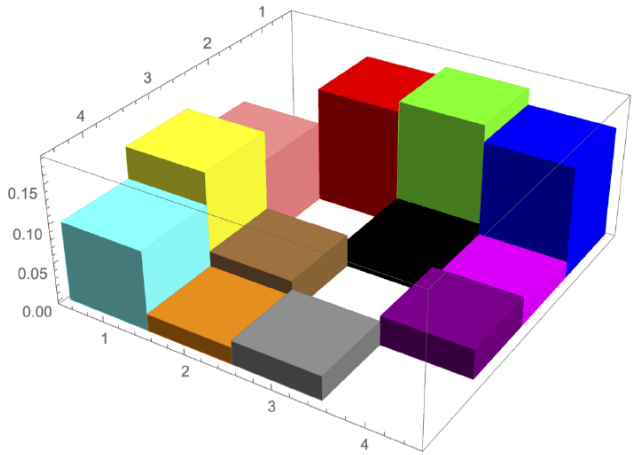


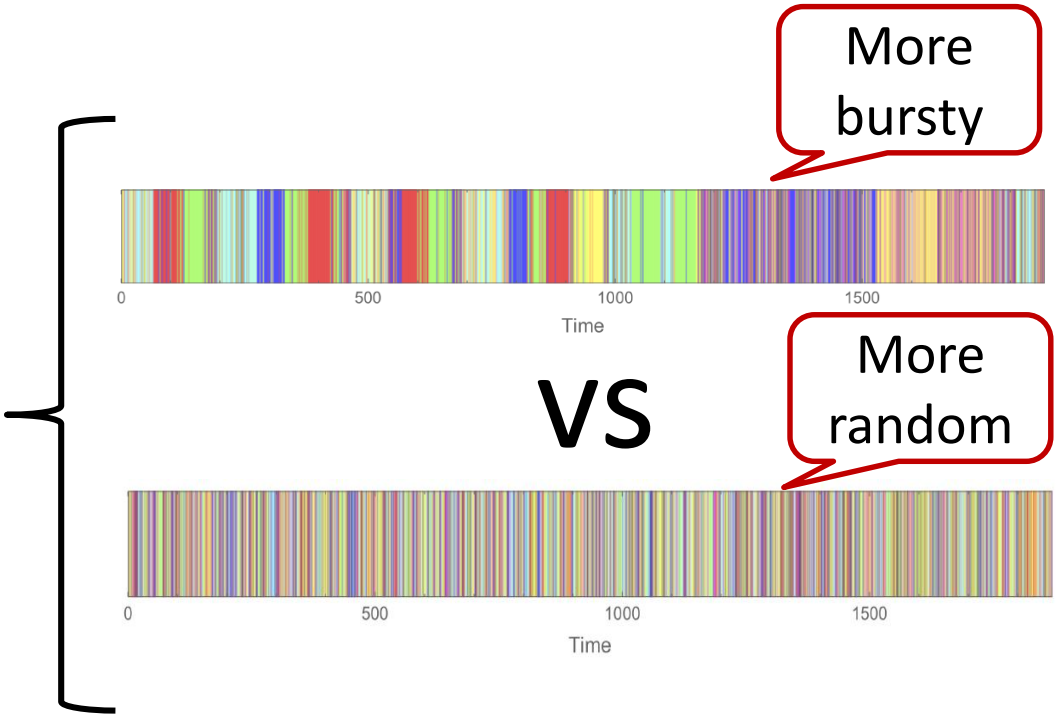


Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

Temporal Structure

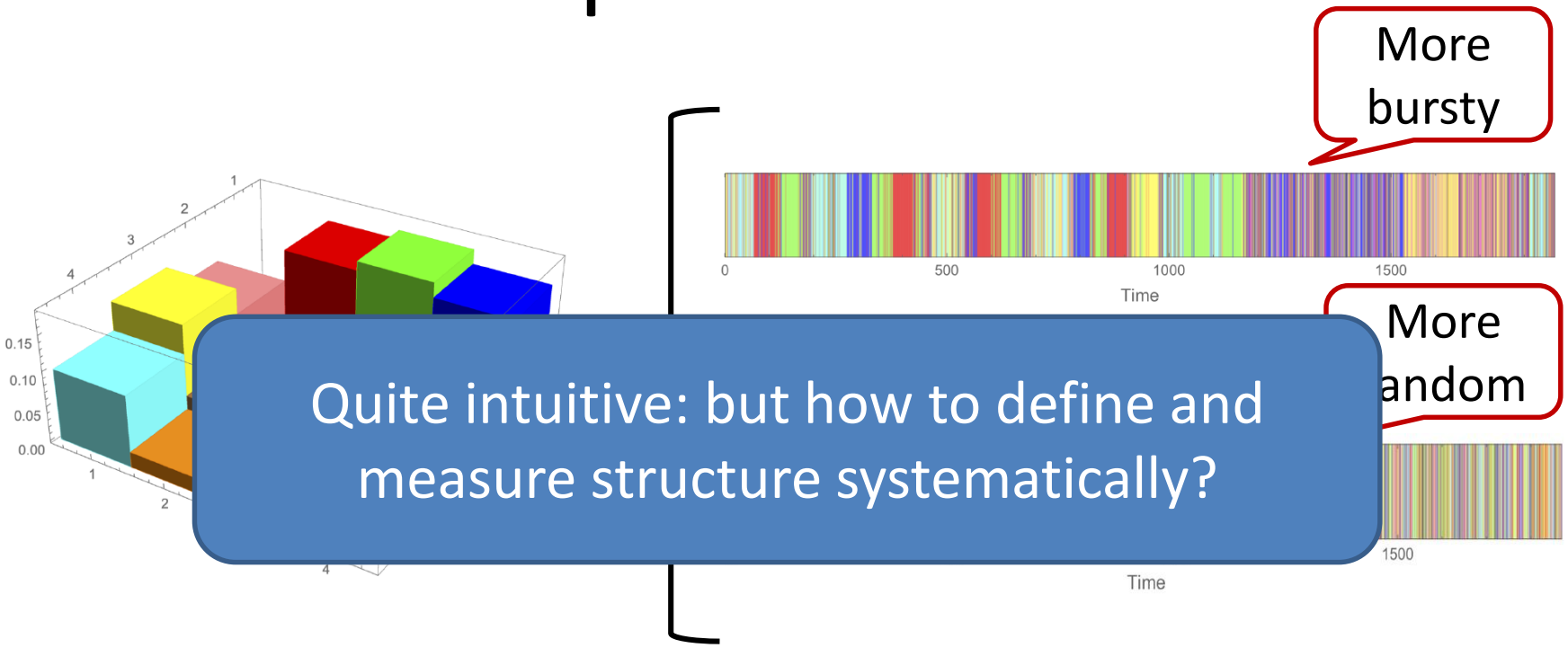




Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

Temporal Structure



Quite intuitive: but how to define and measure structure systematically?

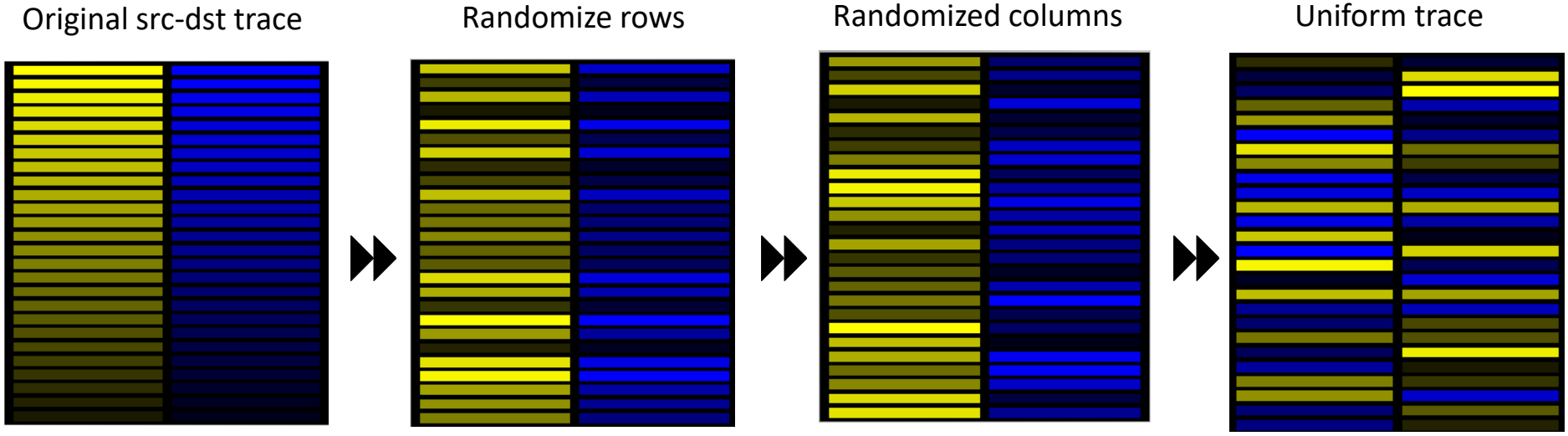
Two different ways to generate *same traffic matrix* (same non-temporal structure):

Which one has *more structure*?

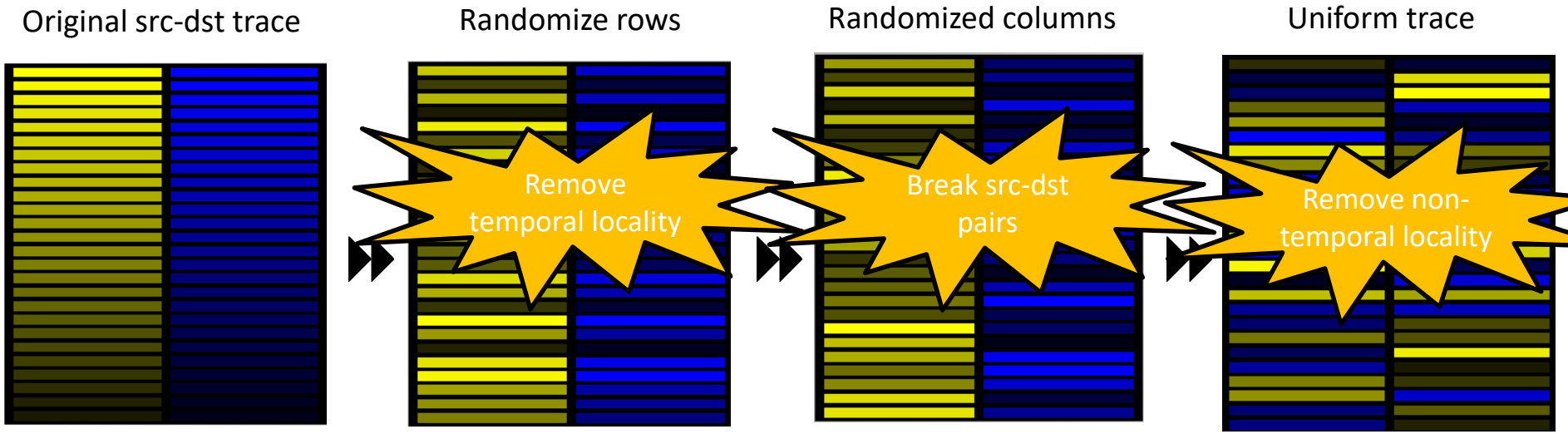
A Principled Approach: The Trace Complexity

- 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

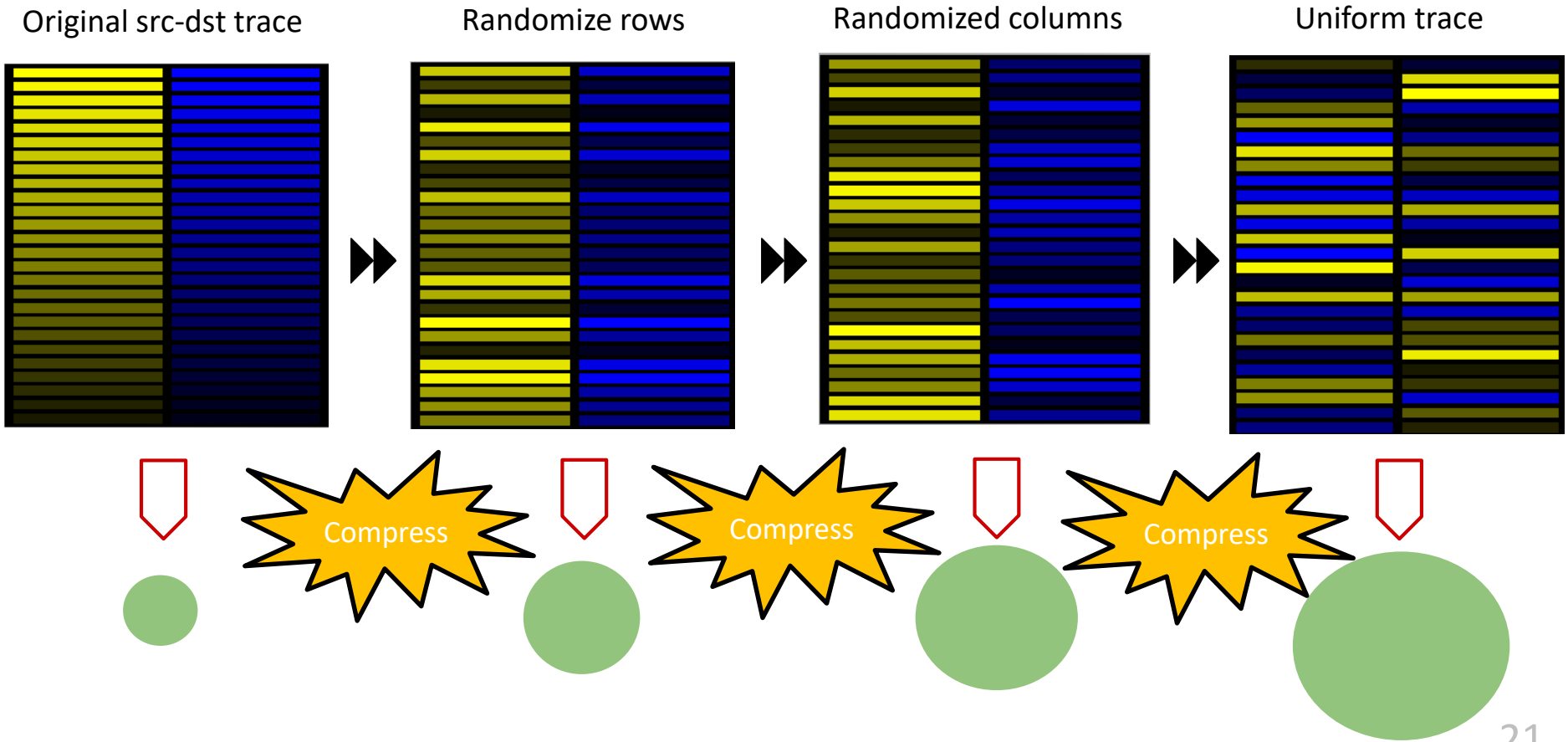
The Trace Complexity



The Trace Complexity



The Trace Complexity



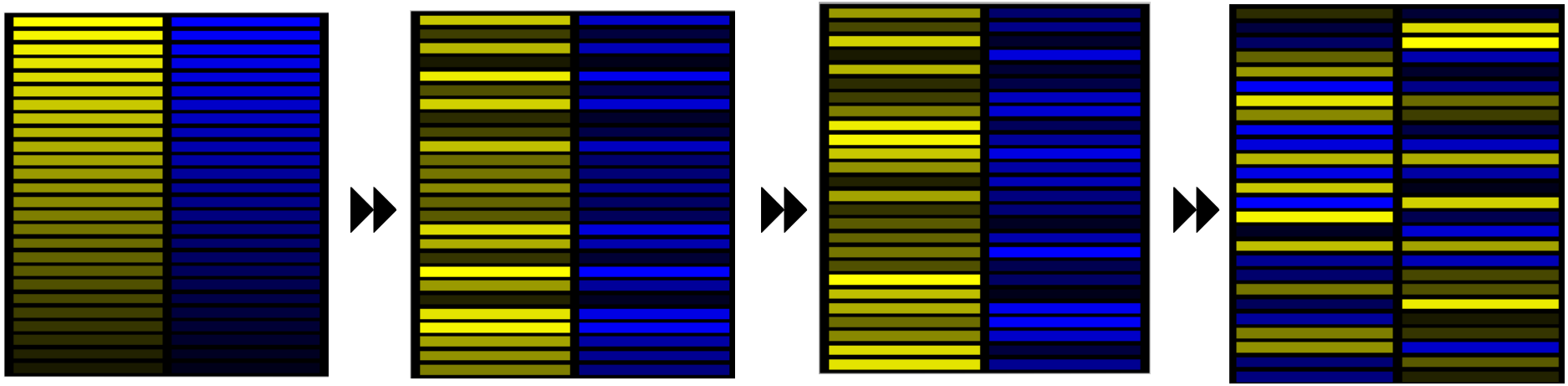
The Trace Complexity

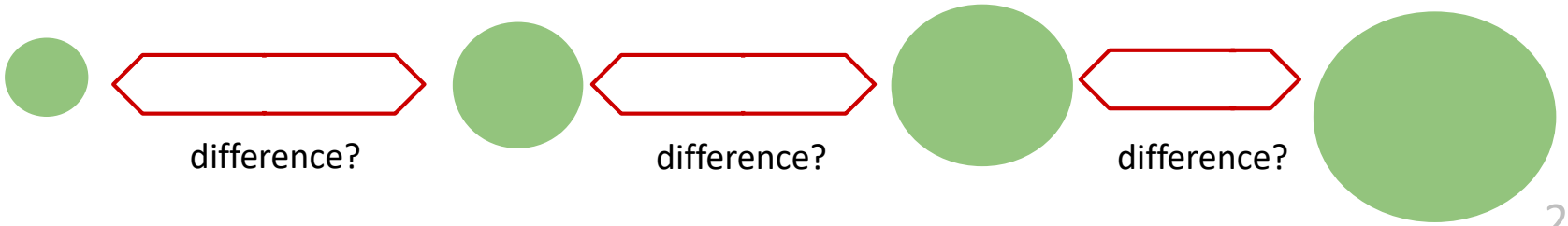
Original src-dst trace

Randomize rows

Randomized columns

Uniform trace





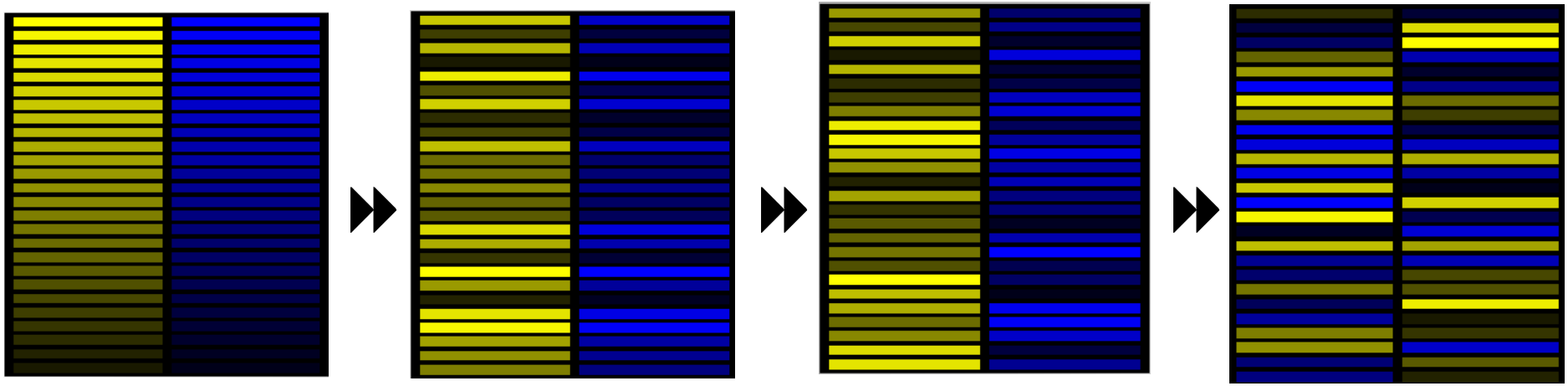
The Trace Complexity

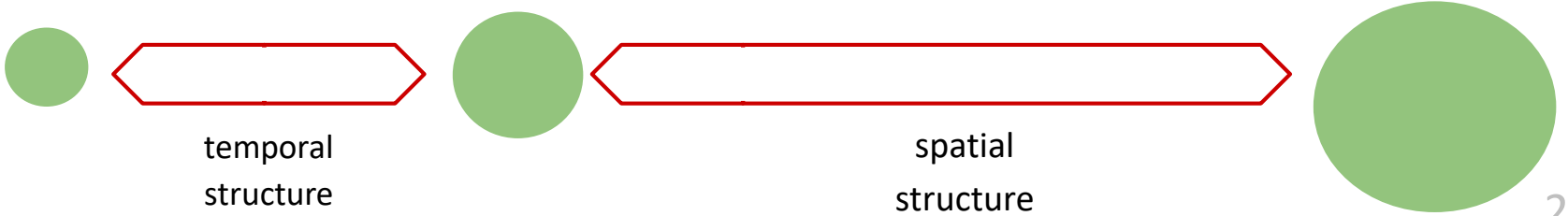
Original src-dst trace

Randomize rows

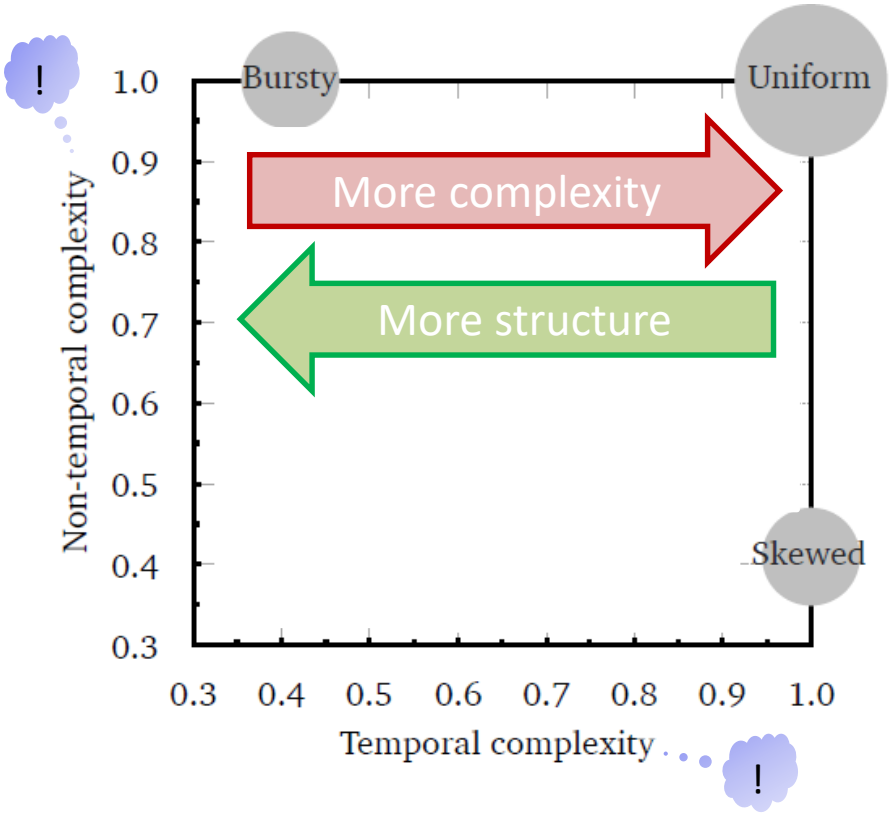
Randomized columns

Uniform trace



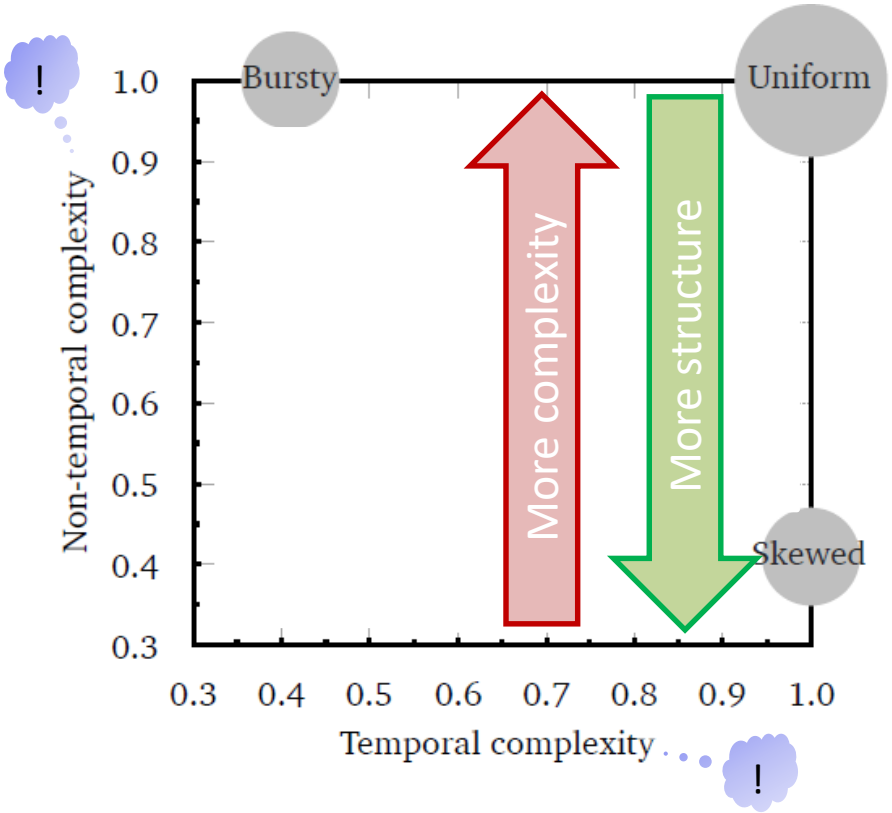


The Complexity Map



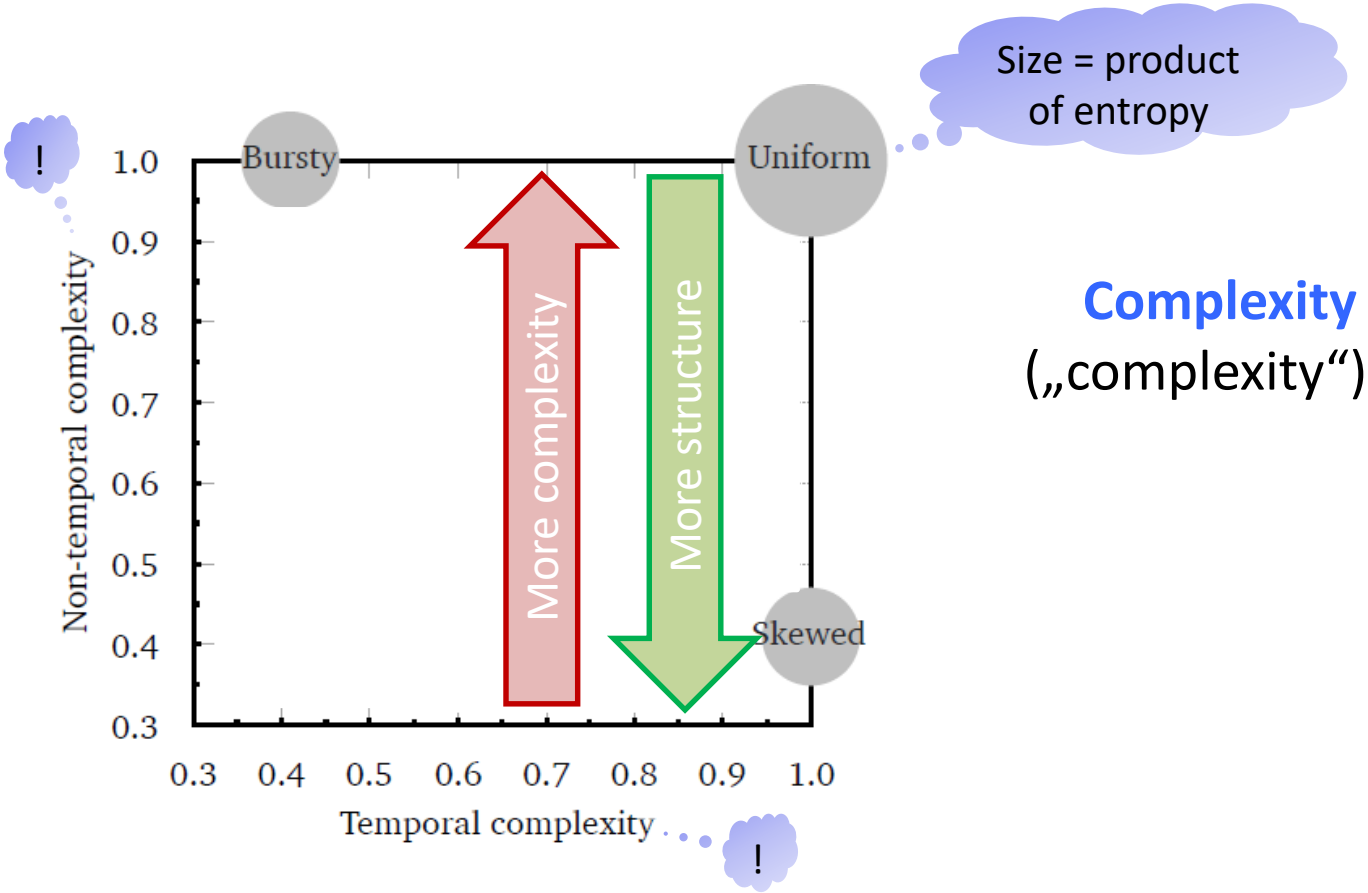
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map



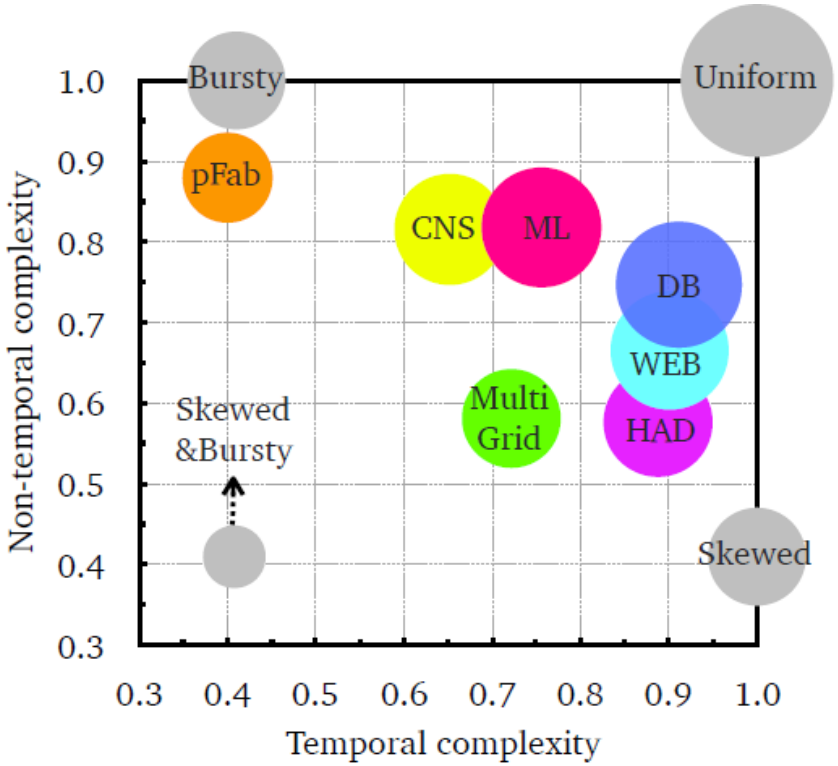
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map



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

The Complexity Map



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**

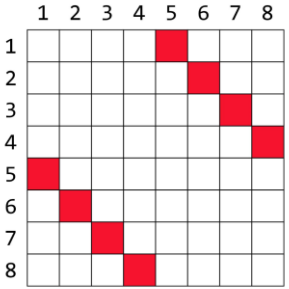
Roadmap

- How much structure is there in the data? A systematic approach.
- **Exploiting structure in data: an example**
- Self-repairing networks

A Simple Example

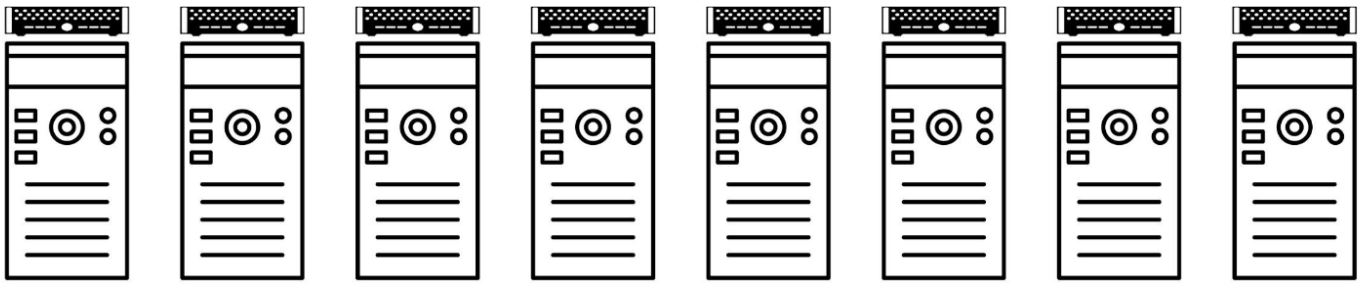
Demand-Aware Network Designs of Bounded Degree
Chen Avin, Kaushik Mondal, and Stefan Schmid.
31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

demand matrix:



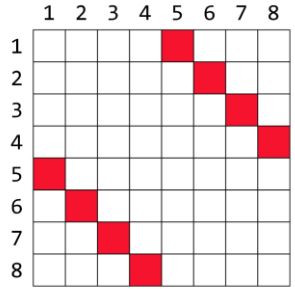
e.g., mirrors

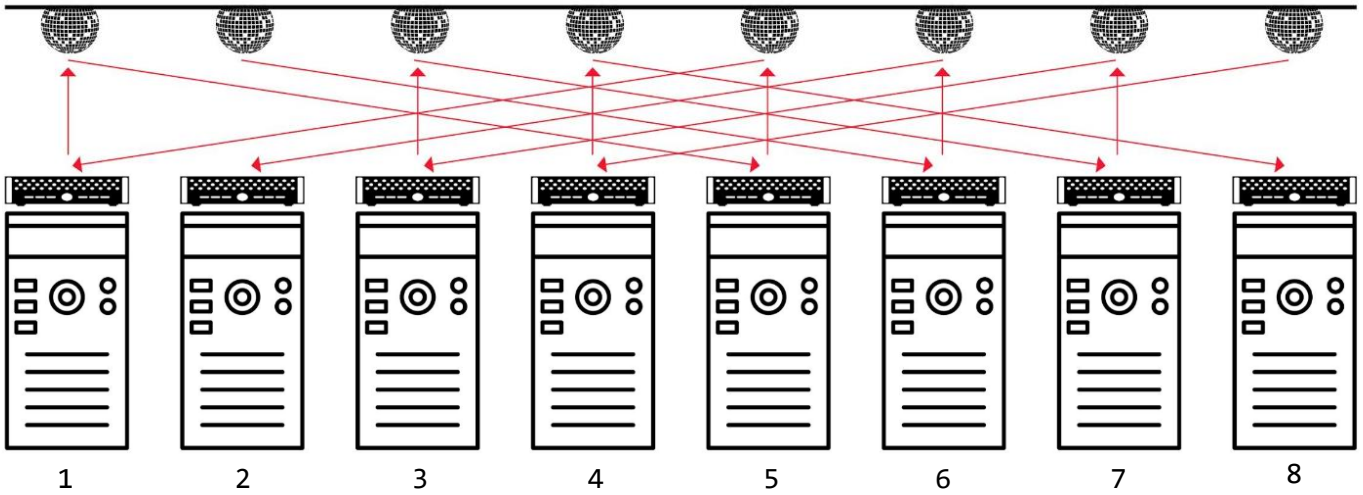
new flexible interconnect



Matches demand

demand matrix:

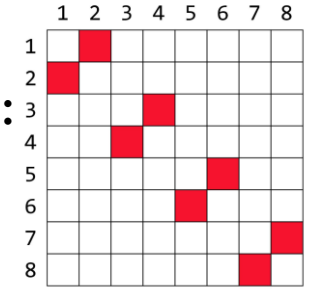




e.g., mirrors

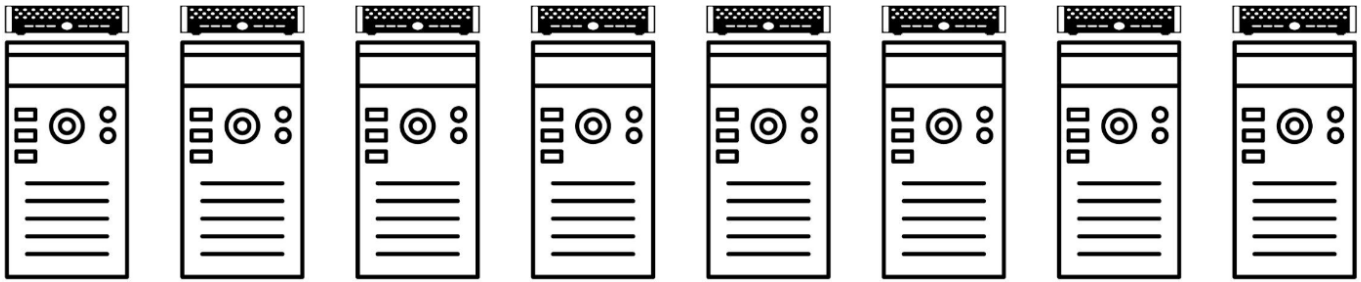
new flexible interconnect

new demand:



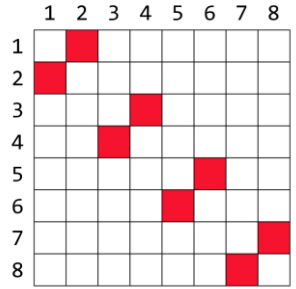
e.g., mirrors

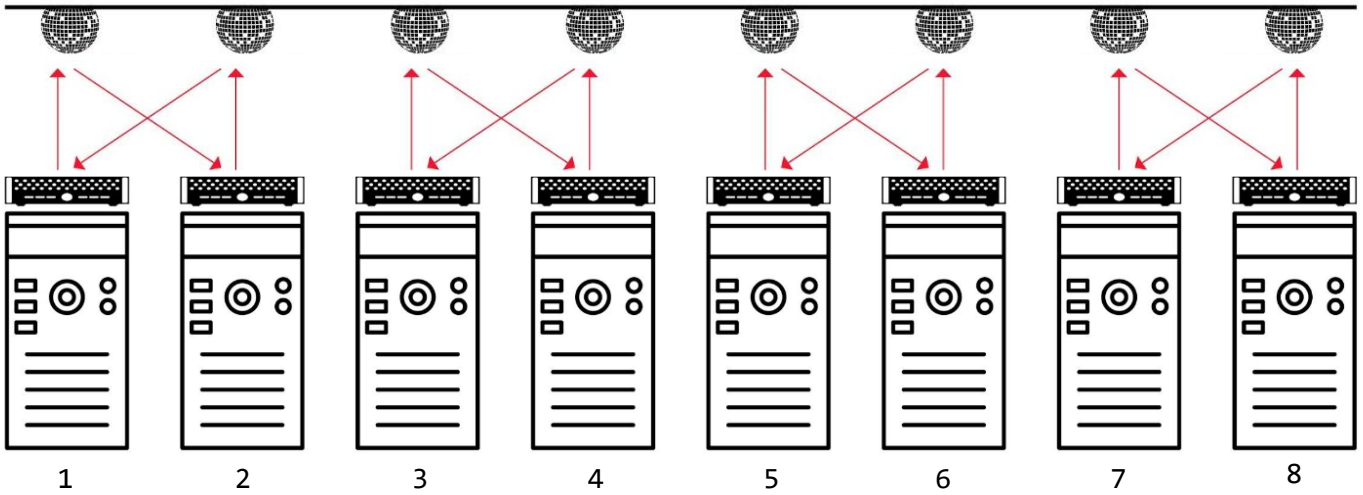
new flexible interconnect



Matches demand

new demand:





e.g., mirrors

new flexible interconnect

More Formally

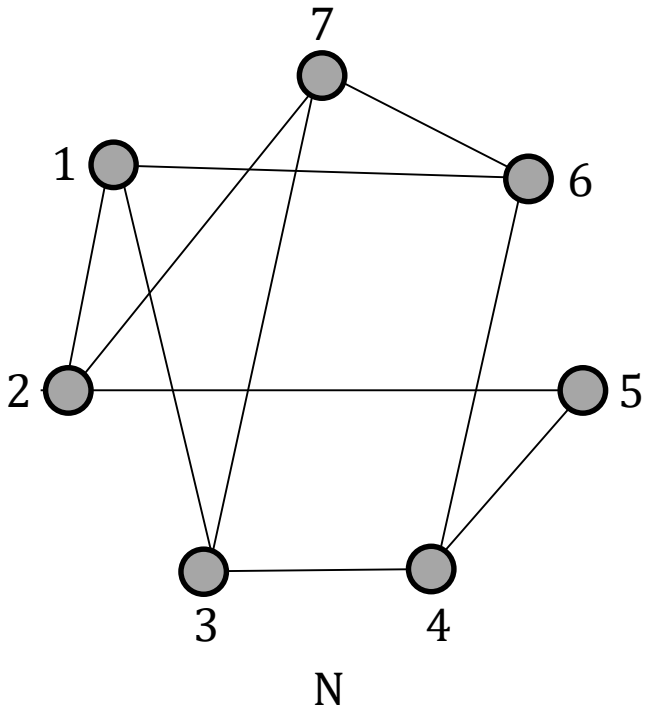
Input: Workload

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

D

Output: Constant-Degree DAN



N

Input: Workload

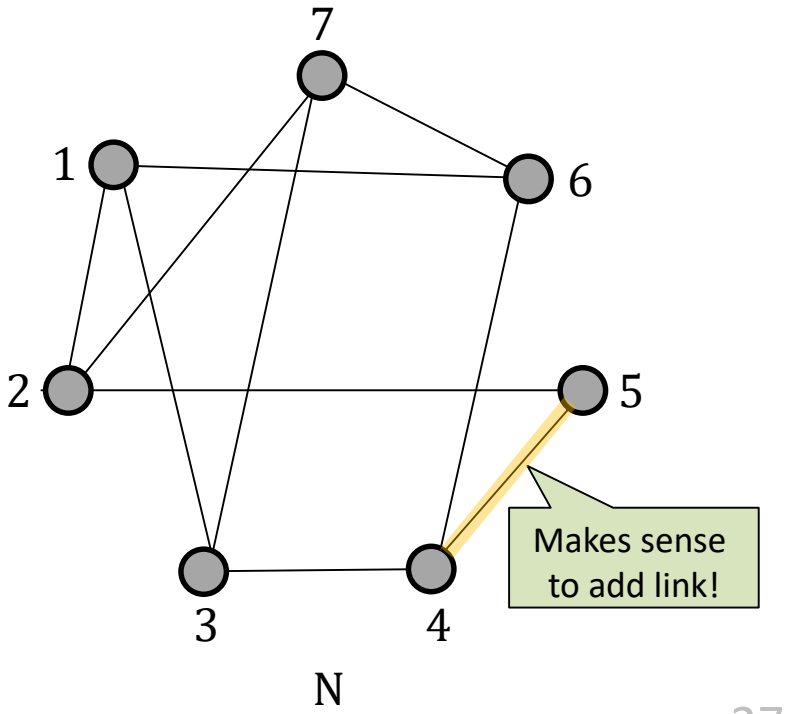
Destinations

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	$\frac{2}{65}$			
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.

D

Output: Constant-Degree DAN



Input: Workload

1 communicates to many.

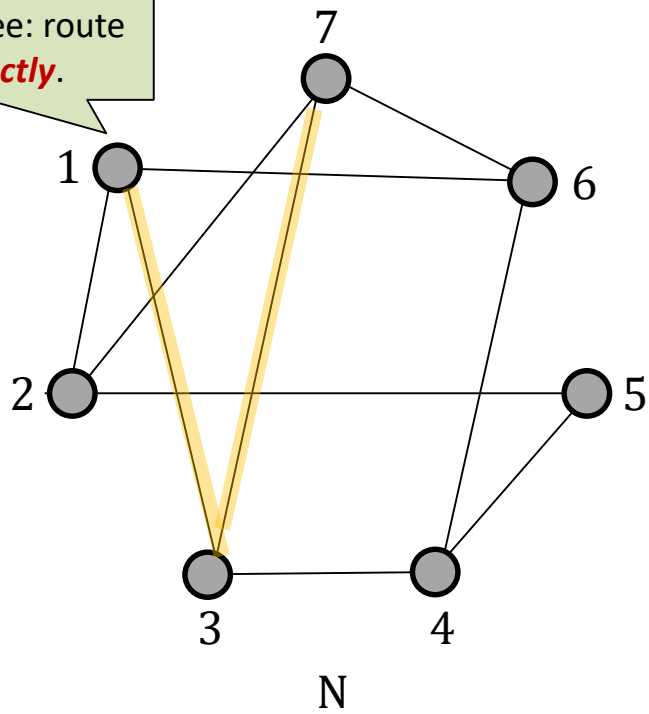
Destinations

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

D

Output: Constant-Degree DAN

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



Input: Workload

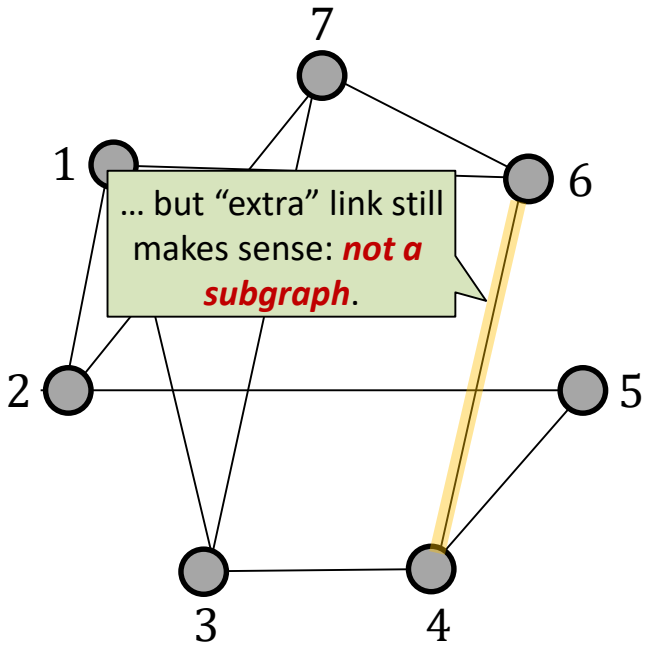
Destinations

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

D

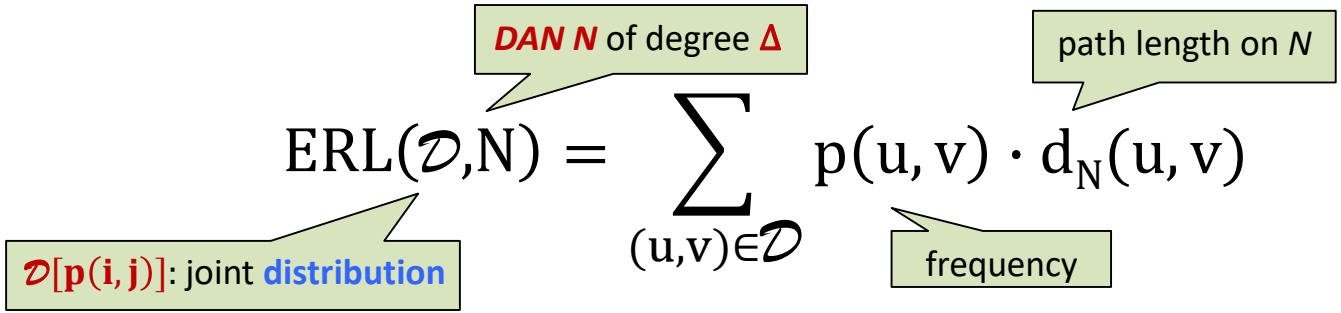
4 and 6 don't communicate…

Output: Constant-Degree DAN



N

Objective: Expected Route Length

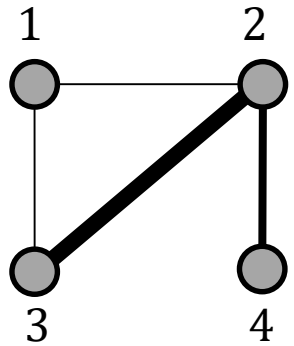


Remark

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

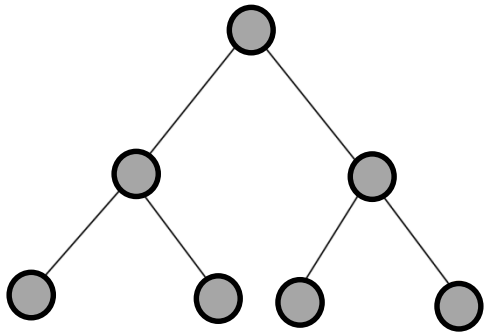
sparse distribution $\mathcal{D}$

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

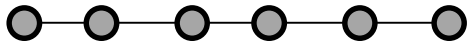


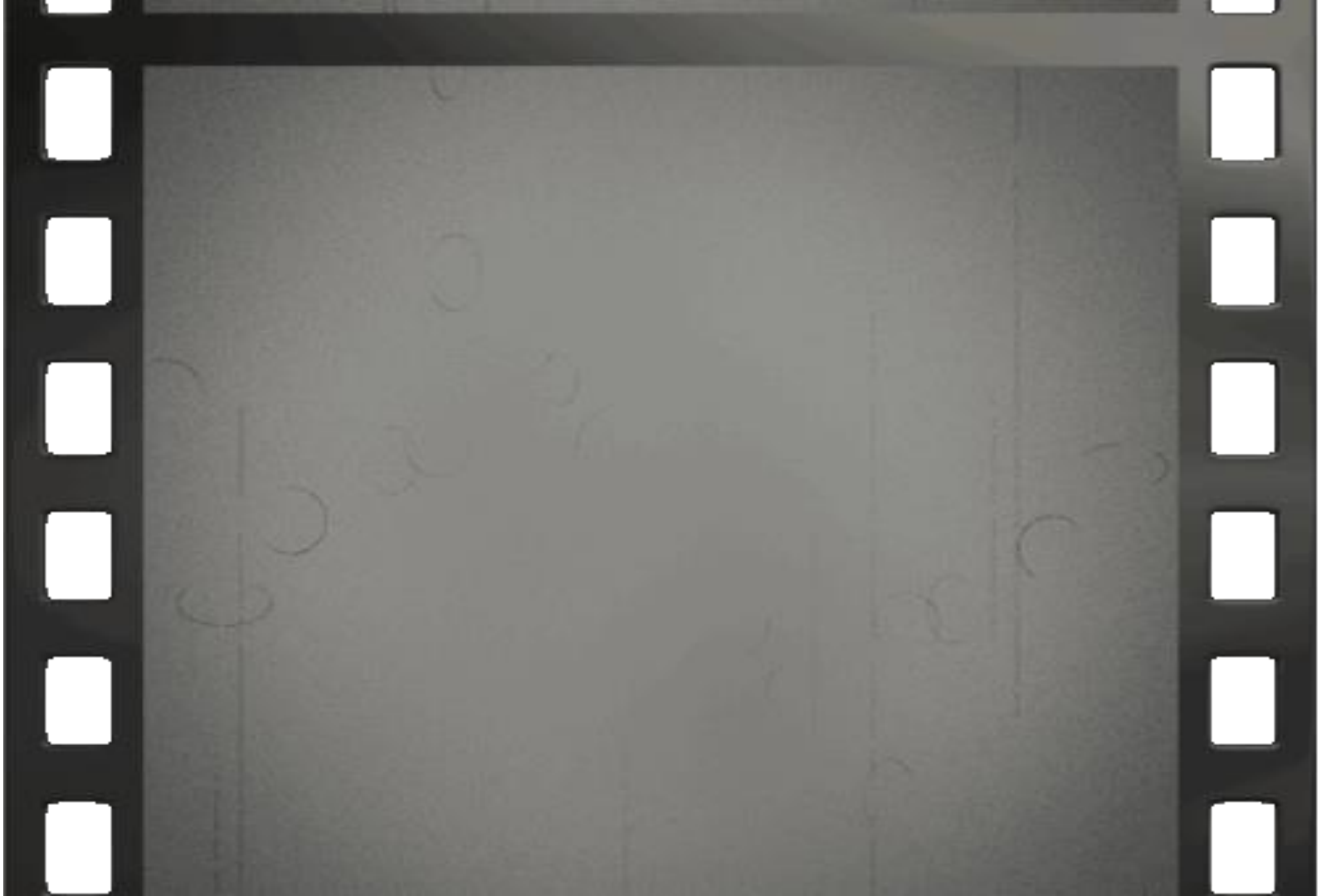
Some Examples

- 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**



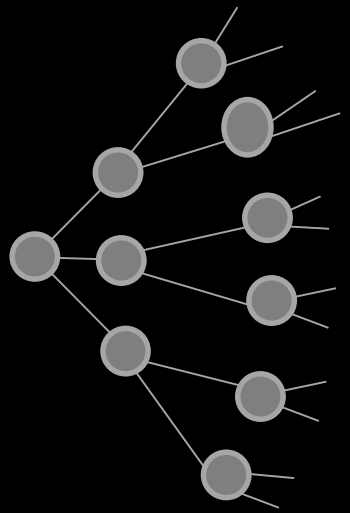


Rewinding the Clock: Degree-Diameter Tradeoff

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

Proof Idea



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

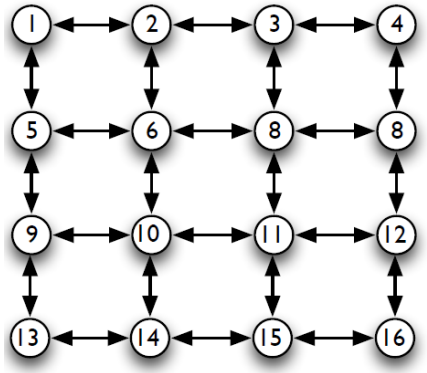
$1 \quad \Delta \quad \Delta(\Delta-1) \ldots$

Kudos to: Pedro Casas

Is there a better tradeoff in DANs?

Sometimes, DANs can be much better!

Example 1: low-degree demand

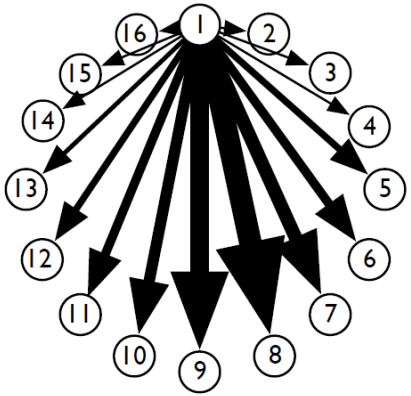


If **demand graph** is of degree Δ, it is trivial to design a **DAN** of degree Δ which achieves an *expected route length of 1*.

Just take DAN = demand graph!

Sometimes, DANs can be much better!

Example 2: skewed demand

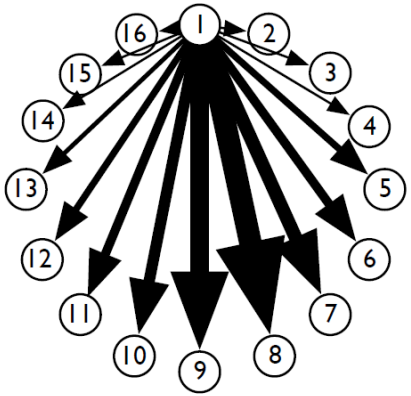


If **demand** is highly skewed, it is also possible to achieve an ***expected route length of O(1)*** in a constant-degree DAN.

?

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 O(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 fully know!): on the **“entropy” of the demand**!

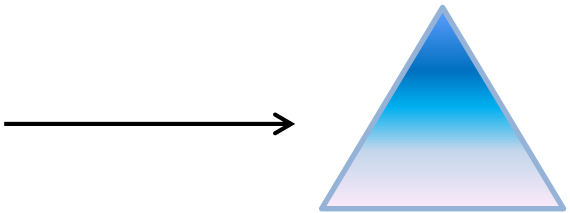
Intuition: Entropy Lower Bound

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

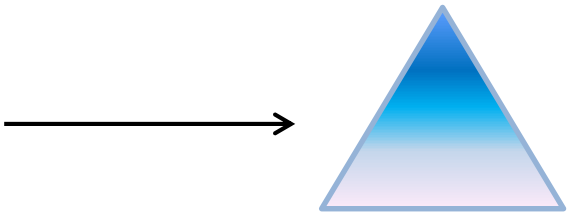
Lower Bound Idea: Leverage Coding or Datastructure

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***



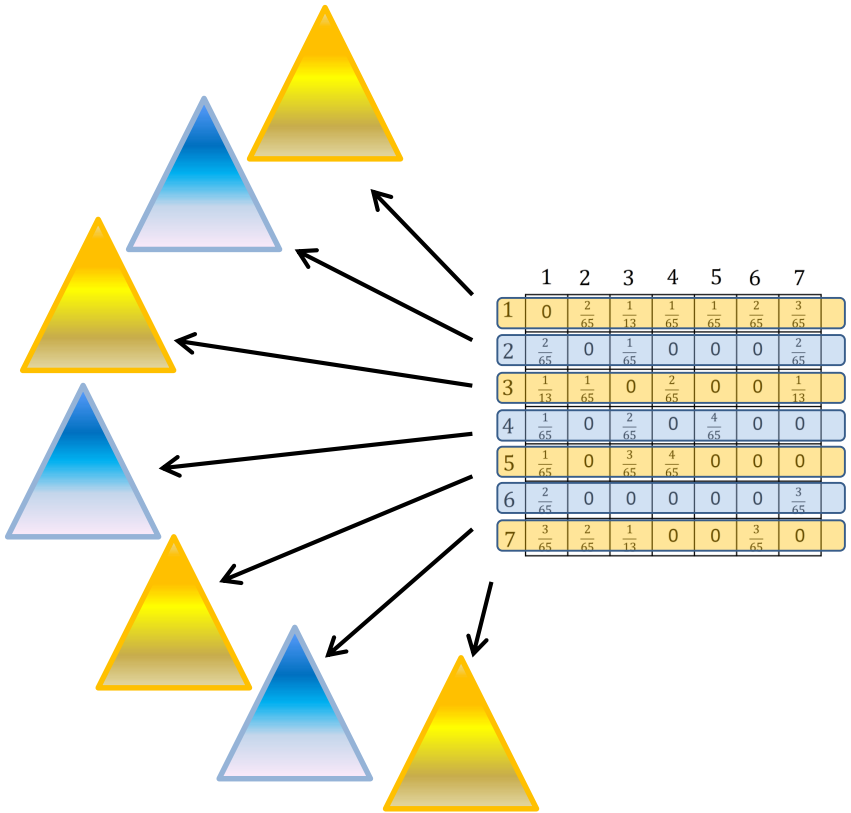
- How good can this tree be? Cannot do better than Δ-ary **Huffman tree** for its destinations
- **Entropy** lower bound on ERL known for binary trees, e.g. ***Mehlhorn*** 1975

So: Entropy of the *Entire* Demand

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

sources, destinations, entropy, degree

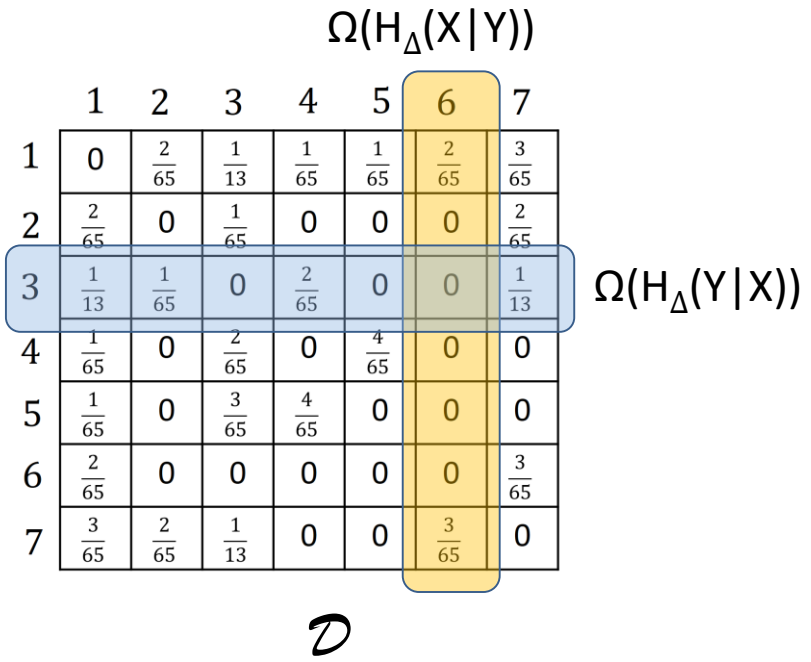
- Compute **ego-tree** for each source node
- Take ***union*** of all **ego-trees**
- Violates ***degree restriction*** but valid lower bound



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

Do this in **both dimensions**:

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

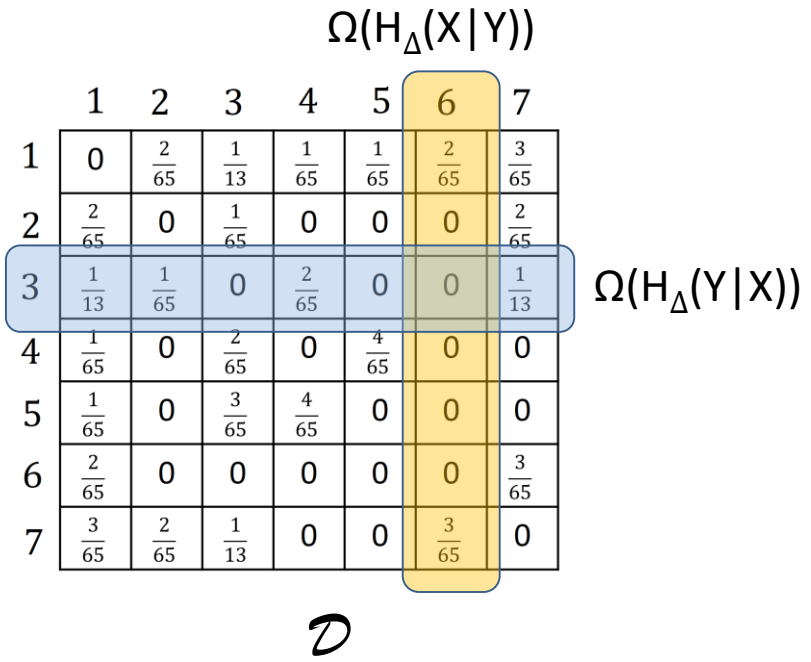


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$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

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



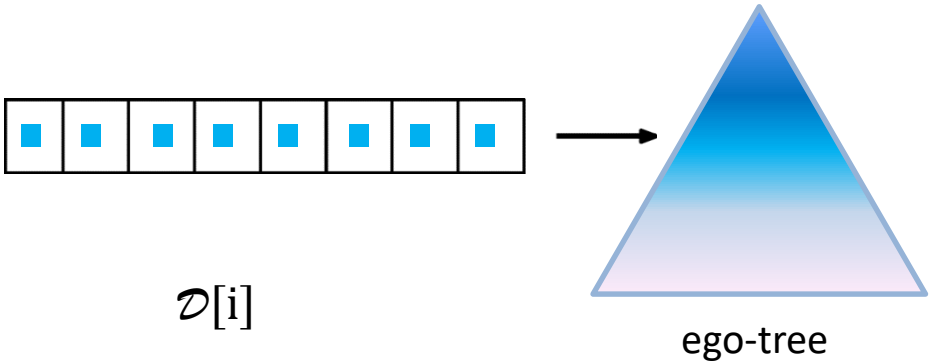
	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$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

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

Achieving Entropy Limit: Algorithms

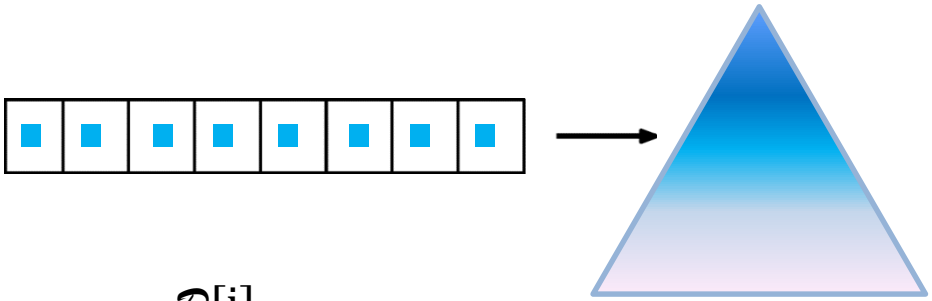
Ego-Trees Revisited

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



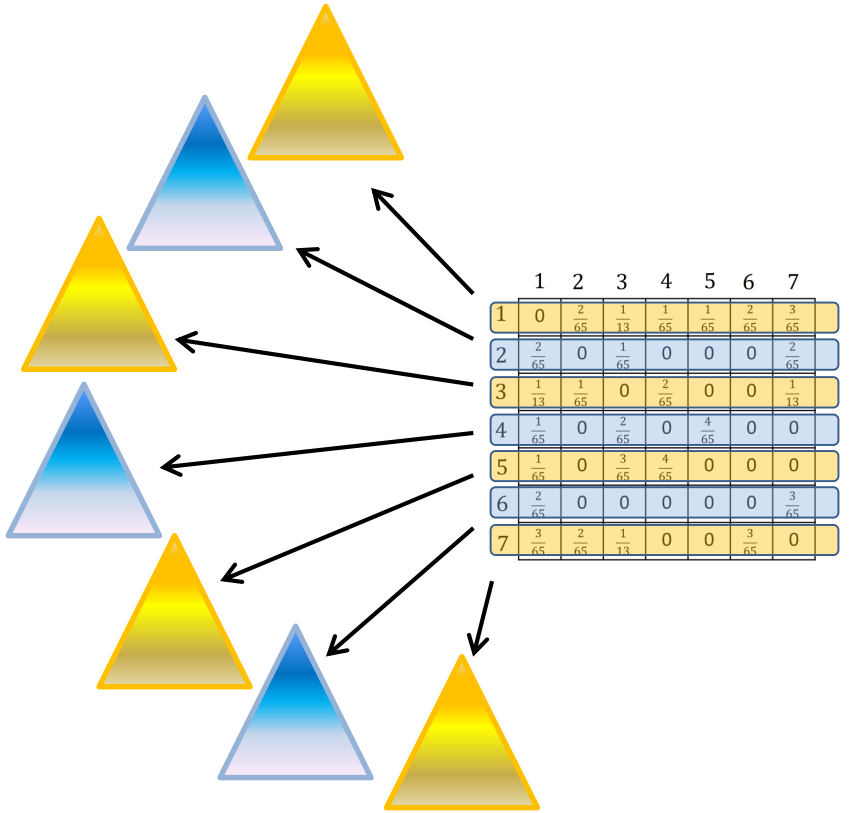
Ego-Trees Revisited

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



$\mathcal{D}[i]$

ego-tree

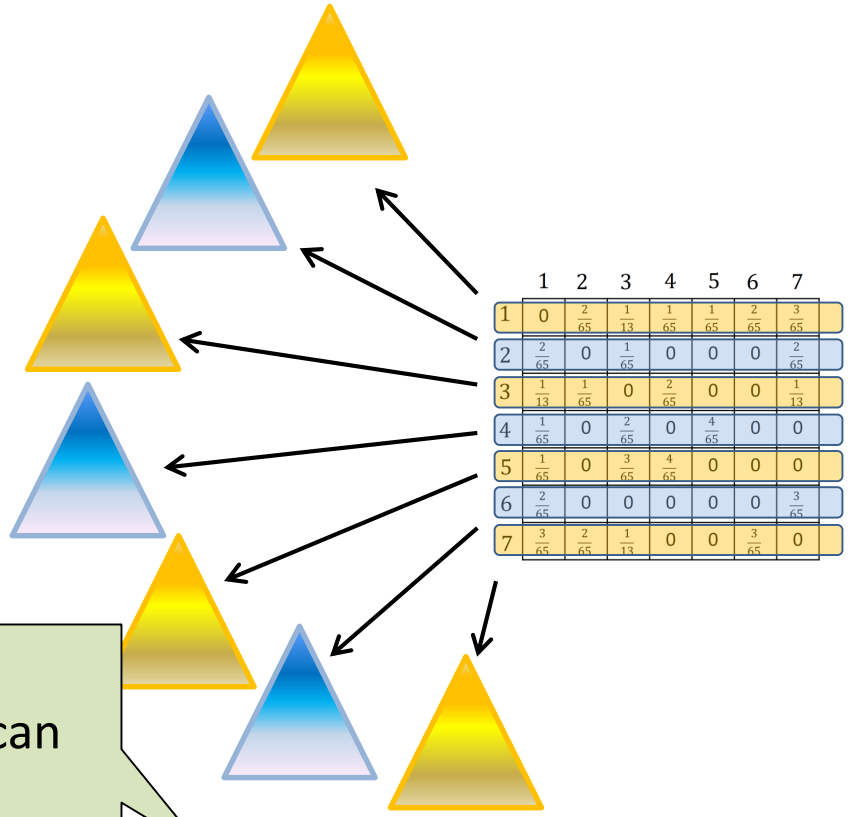


	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$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

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

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

An Analogy

Static vs dynamic demand-aware networks!?

DANs vs **SANs**?

„Coming to the Summer school?“

An Analogy to Coding

00110101…

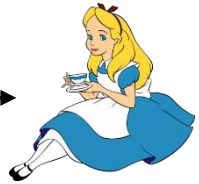
if demand ***arbitrary*** and ***unknown***

worst case network: Full BW	*log diameter*
worst case coding: 00, 01, 10, 11	*log # bits / symbol*

„Coming to the Summer school?“

An Analogy to Coding

01011…



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

DAN!

worst case network: Full BW	*log diameter*
worst case coding: 00, 01, 10, 11	*log # bits / symbol*

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

entropy?

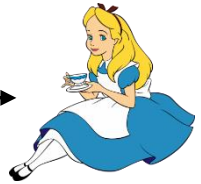
entropy / symbol

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

„Coming to the Summer school?“

An Analogy to Coding

011…



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 ***known*** and ***fixed***

entropy?	static Demand-Aware Nets
entropy / symbol	static Huffman: 1, 01, 001, 000

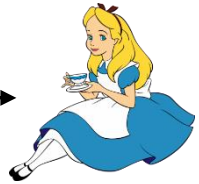
if demand ***unknown*** but ***reconfigurable***

dynamic Demand-Aware Nets
dynamic Huffman codes

„Coming to the Summer school?“

An Analogy to Coding

011…



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 ***known*** and ***fixed***

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

Can exploit ***spatial locality***!

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

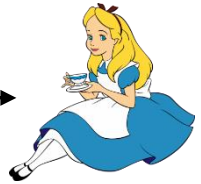
dynamic Demand-Aware Nets
dynamic Huffman codes

Additionally exploit ***temporal locality***!

„Coming to the Summer school?“

An Analogy to Coding

011…



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 ***know***

Can exploit ***spati***

Aware Nets

static Huffman: 1, 01, 001, 000

„Cheating“: need to know demand!

if demand ***unknown*** but ***re***

dynamic Demand

Huffman codes

Need online algorithms!

Additionally exploit ***temporal locality***!

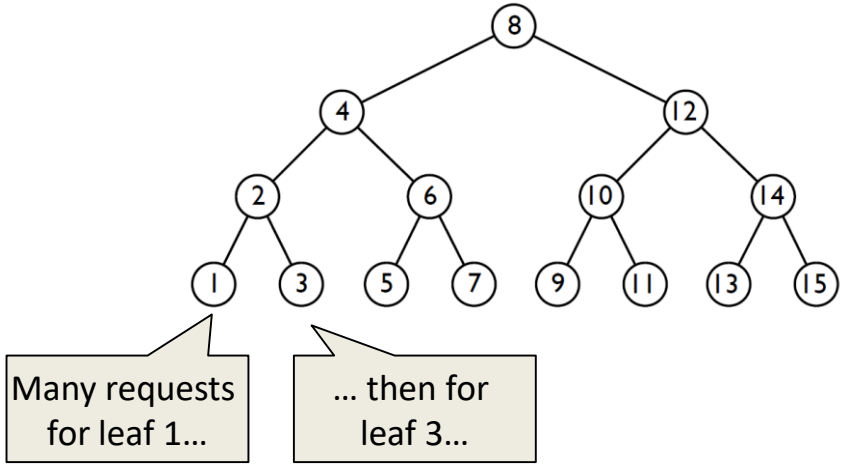
Analogous to *Datastructures*: Oblivious…

- 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*

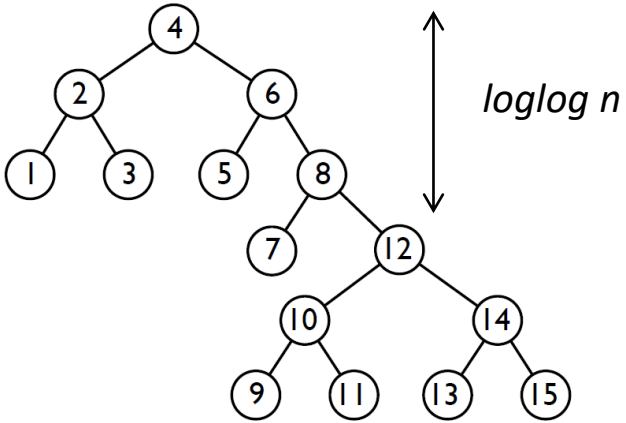
- Items stored at ***O(log n)*** from the root, **uniformly** and **independently** of their frequency



Corresponds to ***max possible demand***!

… Demand-Aware …

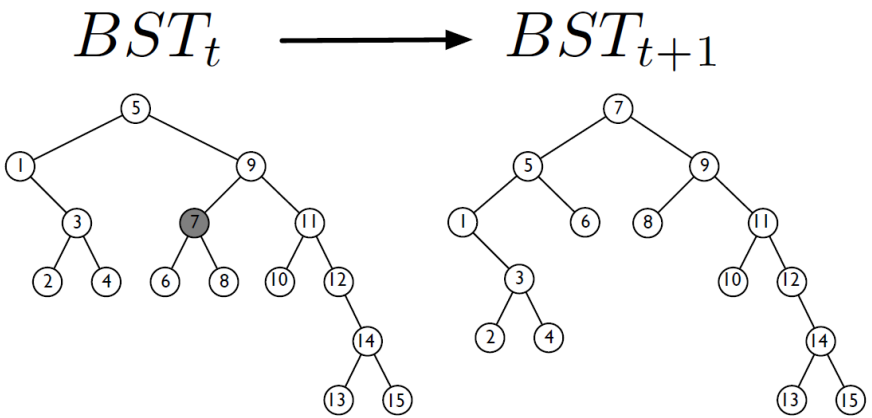
- **Demand-aware fixed** BSTs can take advantage of ***spatial locality*** of the demand
- E.g.: place frequently accessed elements close to the root
- E.g., **Knuth/Mehlhorn/Tarjan** trees
- Recall example **demand**: 1,…,1,3,…,3,5,…,5,7,…,7,…,log(n),…,log(n)
  - Amortized cost ***O(loglog n)***



… Self-Adjusting!

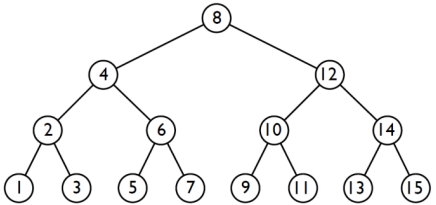
- **Demand-aware reconfigurable** BSTs can additionally take advantage of ***temporal locality***

- 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

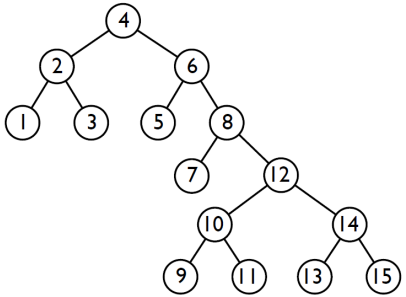
Oblivious



Lookup

O(log n)

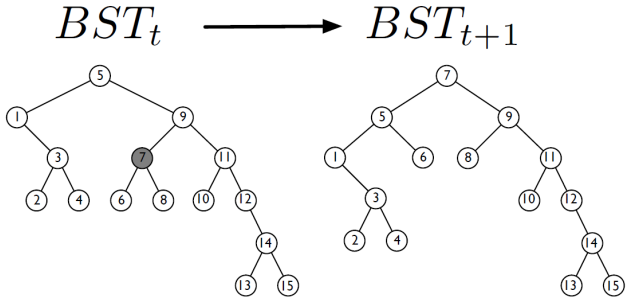
Demand-Aware



Exploit **spatial locality**:

empirical entropy O(loglog n)

Self-Adjusting

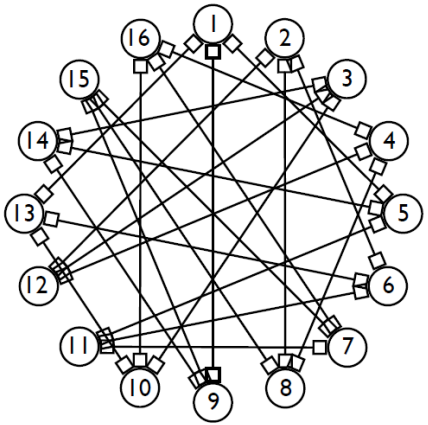


Exploit **temporal locality** as well:

O(1)

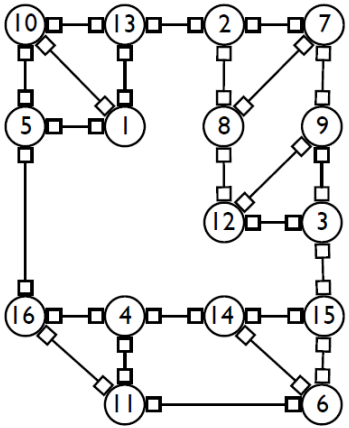
Analogously for Networks

Oblivious



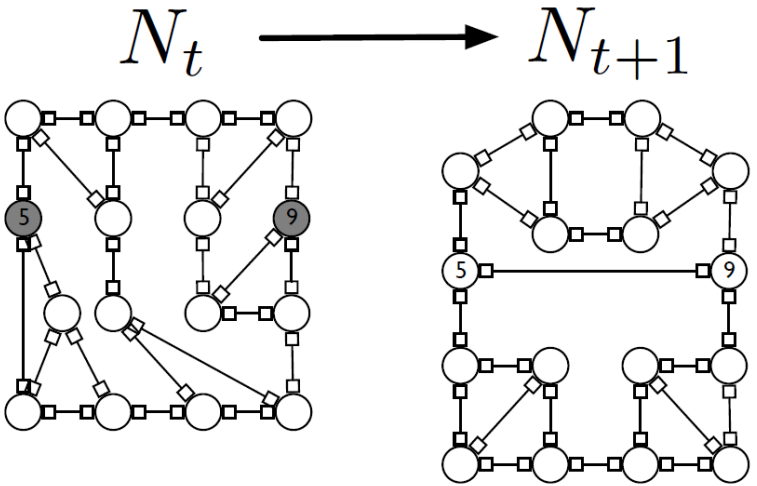
Const degree
(e.g., **expander**):
route lengths ***O(log n)***

DAN



Exploit **spatial locality**

SAN



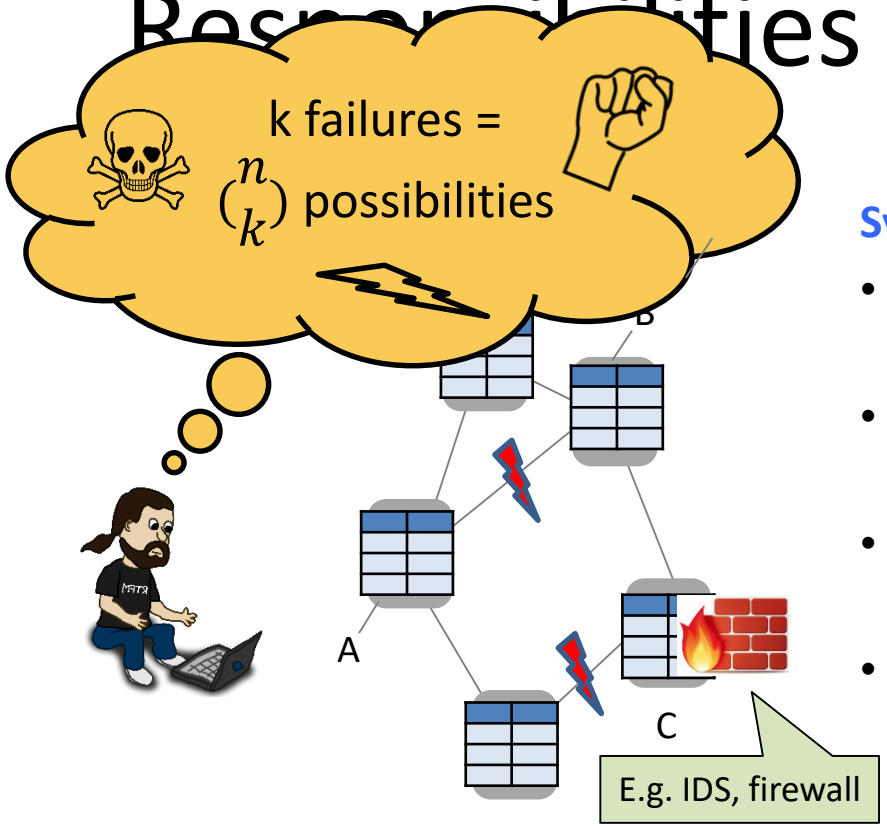
Exploit **temporal locality** as well

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

Roadmap

- How much structure is there in the data? A systematic approach.
- Exploiting structure in data: an example
- **Self-repairing networks**

Responsibilities of a Sysadmin



Sysadmin responsible for:

- **Reachability:** Can traffic from ingress port A reach egress port B?
- **Loop-freedom:** Are the routes implied by the forwarding rules loop-free?
- **Policy:** Is it ensured that traffic from A to B never goes via C?
- **Waypoint enforcement:** Is it ensured that traffic from A to B is always routed via a node C?

… and everything even under multiple failures?!

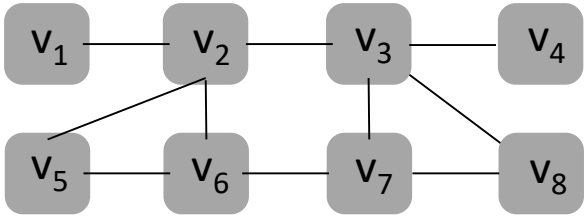
Can we automate such tests
or even self-repair?

Can we automate such tests or even self-repair?

Yes! Sometimes even ***fast***:
with **formal methods**
(enhanced with AI…).

How (MPLS) Networks Work

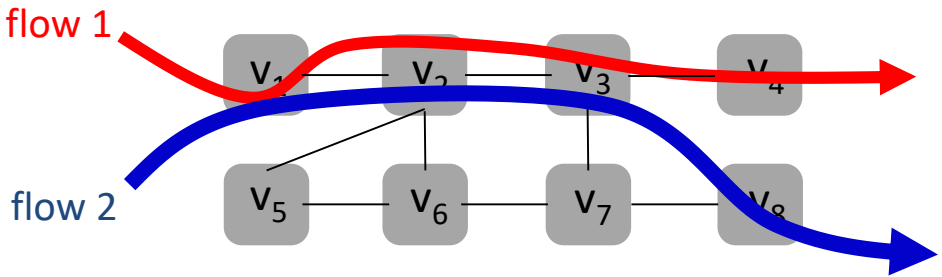
- Forwarding based on **top label** of label **stack**



Default routing of two flows

How (MPLS) Networks Work

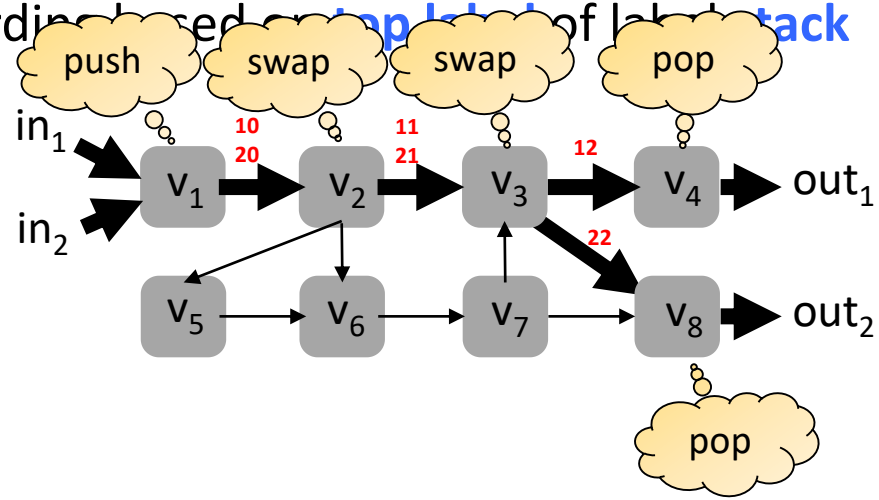
- Forwarding based on **top label** of label **stack**



Default routing of two flows

How (MPLS) Networks Work

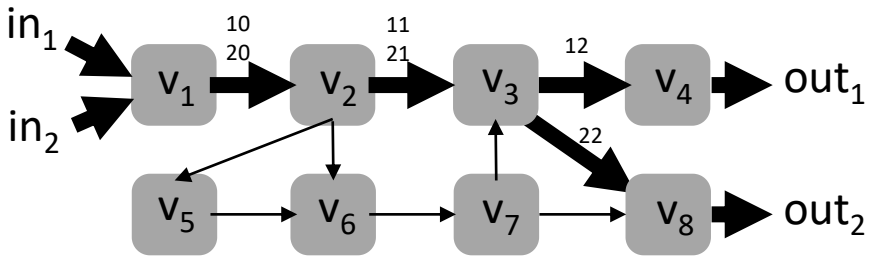
- Forwarding based on top label of label stack



Default routing of two flows

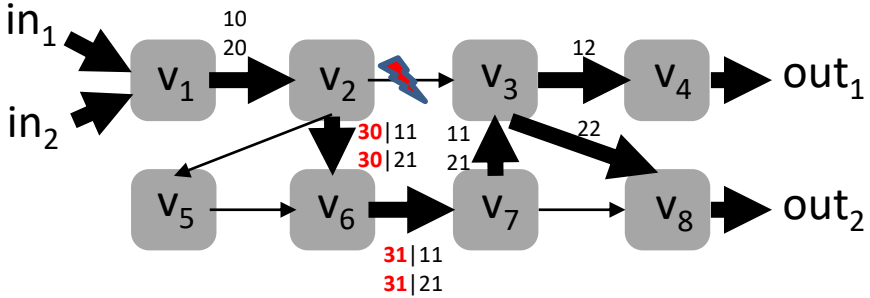
Fast Reroute Around *1 Failure*

- Forwarding based on **top label** of label **stack** (in packet header)



Default routing of two flows

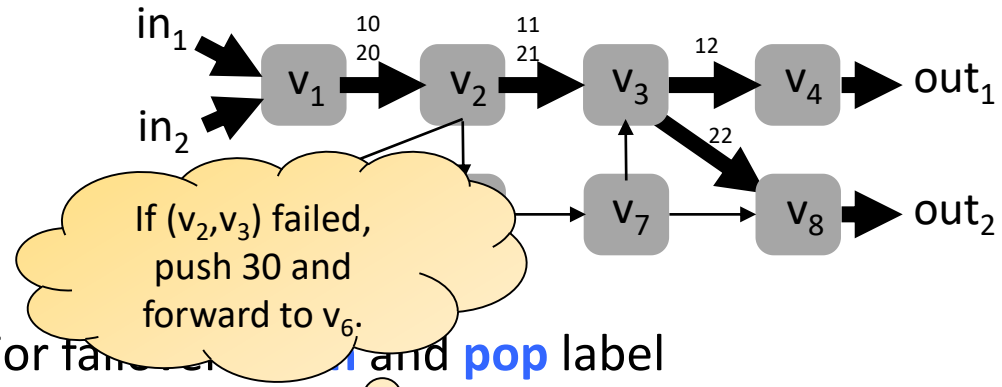
- For failover: **push** and **pop** label



One failure: push 30: route around (v_2,v_3)

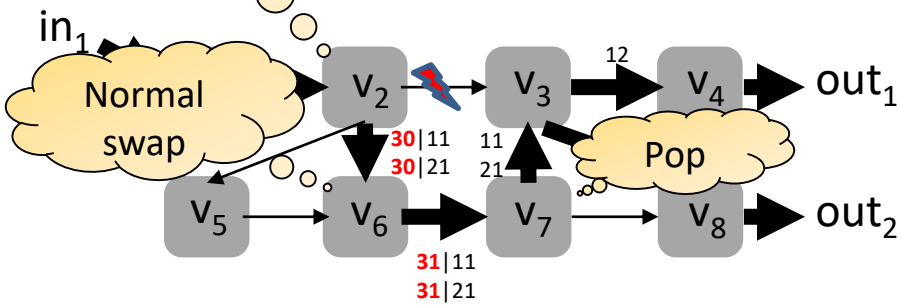
Fast Reroute Around *1 Failure*

- Forwarding based on **top label** of label **stack** (in packet header)



Default routing of two flows

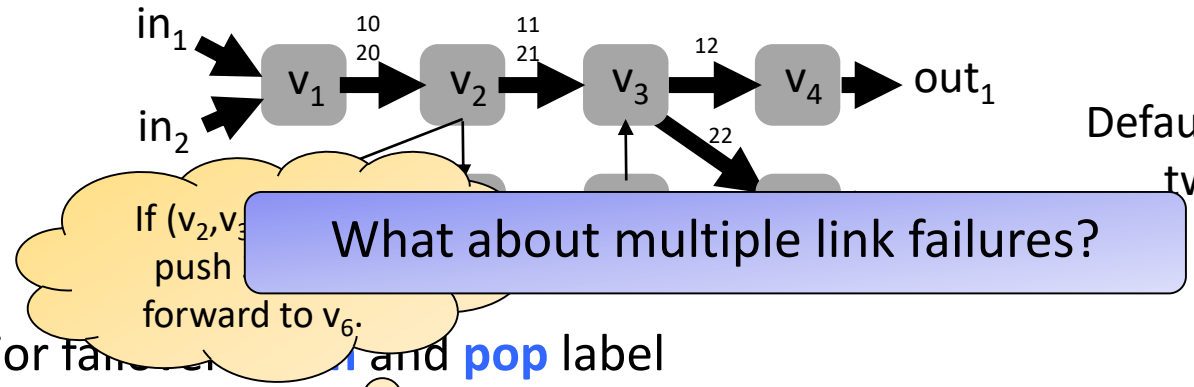
- For fai[illegible]l and **pop** label



One failure: push 30: route around (v_2,v_3)

Fast Reroute Around *1 Failure*

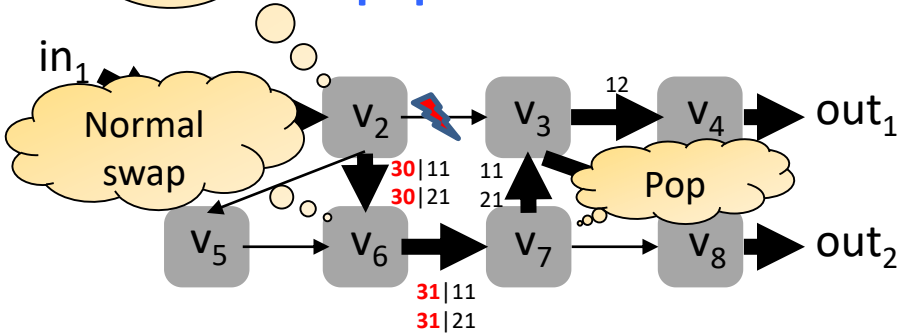
- Forwarding based on **top label** of label **stack** (in packet header)



What about multiple link failures?

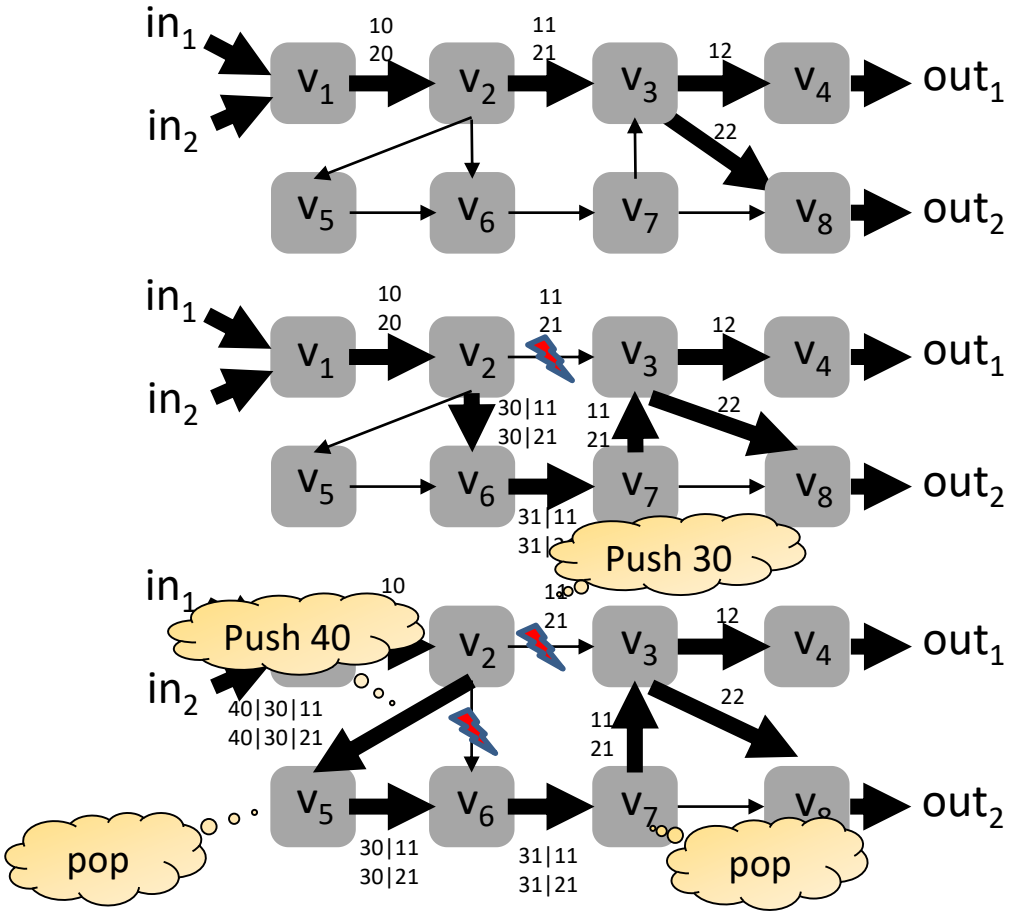
Default routing of two flows

- For fai[illegible] and **pop** label



One failure: push 30: route around (v_2,v_3)

2 Failures: Push *Recursively*



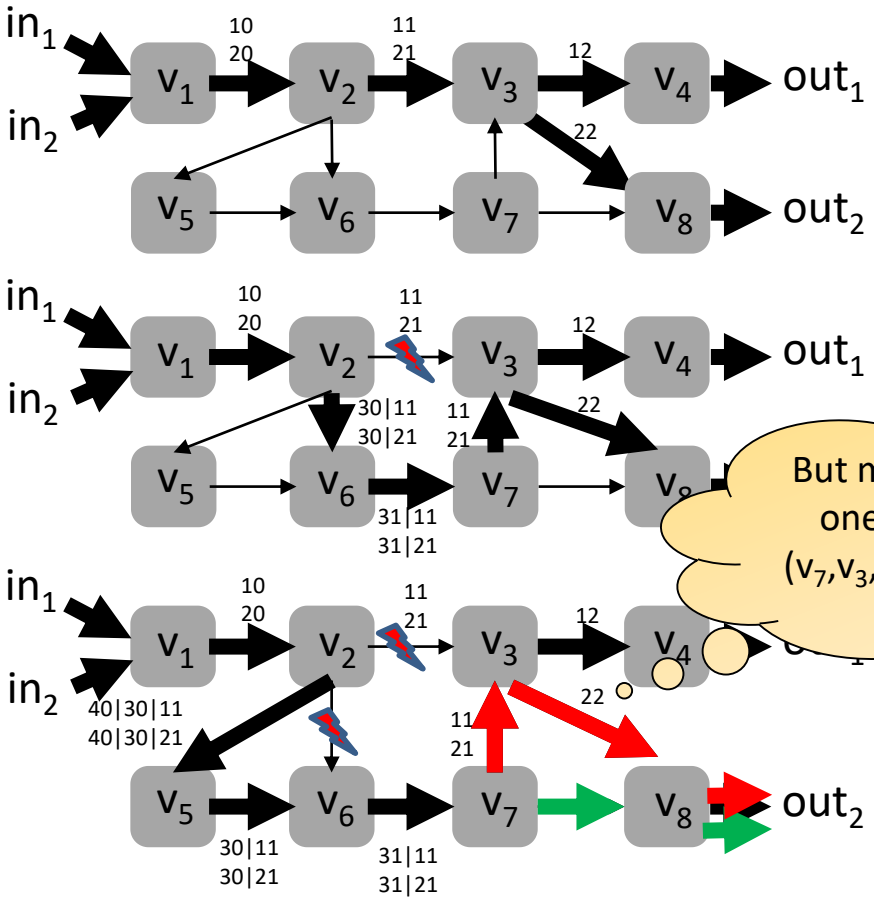
Original Routing

One failure: push 30: route around (v_2,v_3)

Two failures: first push 30: route around (v_2,v_3)

Push recursively 40: route around (v_2,v_6)

2 Failures: Push *Recursively*



Original Routing

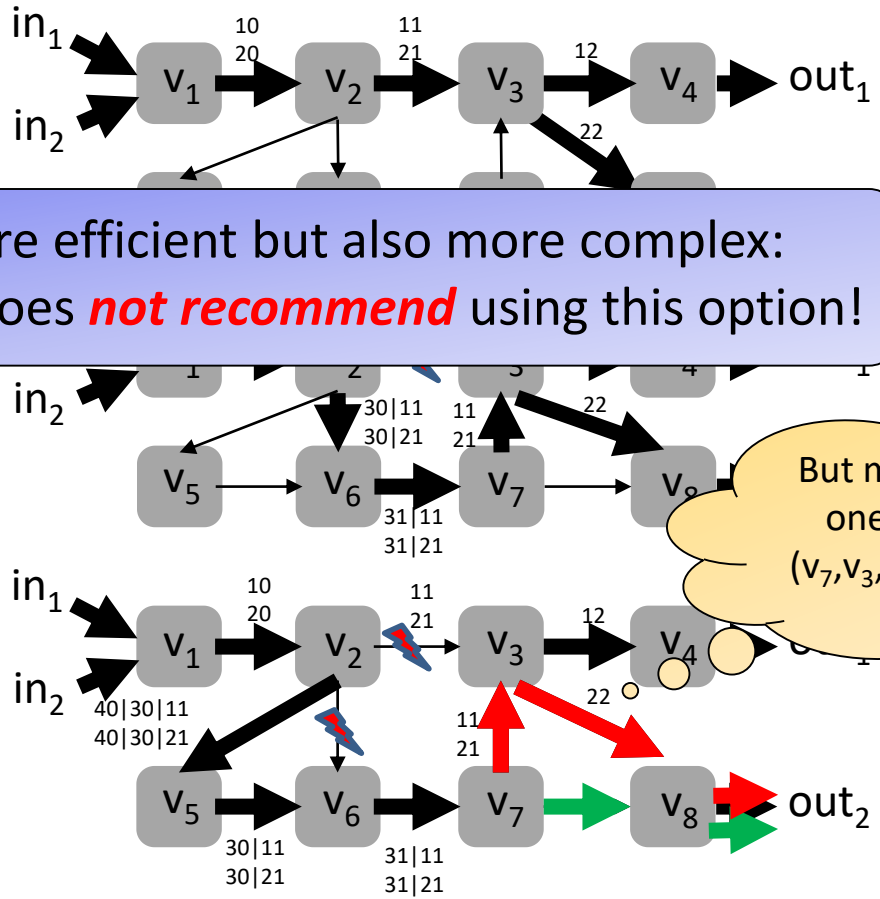
One failure: push 30:

But masking links one-by-one can be inefficient: (v_7,v_3,v_8) could be shortcut to (v_7,v_8).

route around (v_2,v_3)

Push recursively 40: route around (v_2,v_6)

2 Failures: Push *Recursively*



More efficient but also more complex:
Cisco does ***not recommend*** using this option!

Original Routing

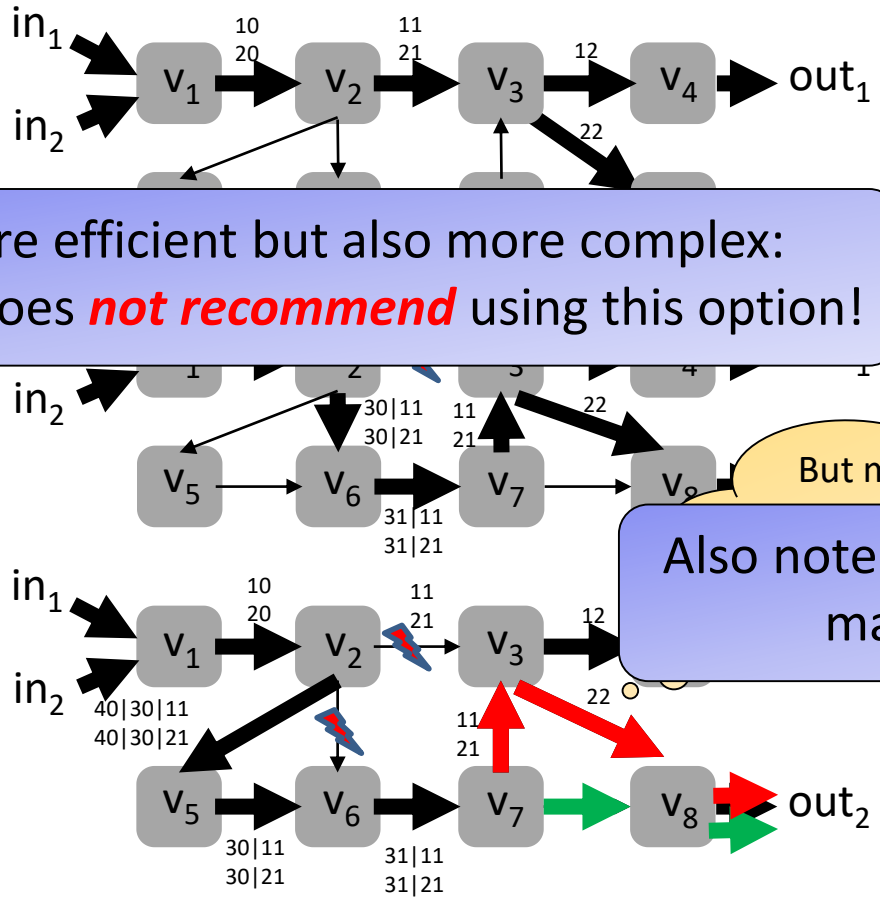
One failure: push 30:

But masking links one-by-one can be inefficient: (v_7,v_3,v_8) could be shortcut to (v_7,v_8).

…sh 30: route around (v_2,v_3)

Push recursively 40: route around (v_2,v_6)

2 Failures: Push *Recursively*



More efficient but also more complex:
Cisco does ***not recommend*** using this option!

Original Routing

One failure: push 30:

But masking links one-by-

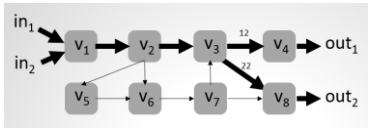
around (v_2,v_3)

Also note: due to push, ***header size*** may grow arbitrarily!

Push recursively 40:
route around (v_2,v_6)

Leveraging Automata-Theoretic Approach

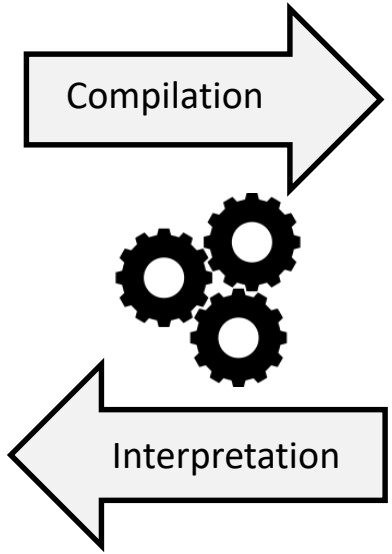
FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$

global FFT	Out-I	In-Label	Out-I	op
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$

MPLS **configurations**, Segment Routing etc.



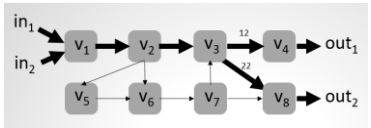
$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

Pushdown Automaton and **Prefix Rewriting Systems** Theory

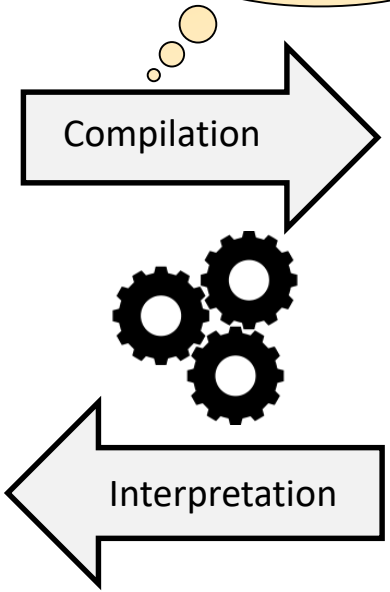
Leveraging Autom…pach

Use cases: Sysadmin ***issues queries*** to test certain properties, or do it on a ***regular basis*** automatically!

FT	In-I	In-Label	Out-I	op
τ_{v_1}	in_1	$\perp$	(v_1, v_2)	$push(10)$
	in_2	$\perp$	(v_1, v_2)	$push(20)$
τ_{v_2}	(v_1, v_2)	10	(v_2, v_3)	$swap(11)$
	(v_1, v_2)	20	(v_2, v_3)	$swap(21)$
τ_{v_3}	(v_2, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_2, v_3)	21	(v_3, v_8)	$swap(22)$
	(v_7, v_3)	11	(v_3, v_4)	$swap(12)$
	(v_7, v_3)	21	(v_3, v_8)	$swap(22)$
τ_{v_4}	(v_3, v_4)	12	out_1	pop
τ_{v_5}	(v_2, v_5)	40	(v_5, v_6)	pop
τ_{v_6}	(v_2, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	30	(v_6, v_7)	$swap(31)$
	(v_5, v_6)	61	(v_6, v_7)	$swap(62)$
	(v_5, v_6)	71	(v_6, v_7)	$swap(72)$
τ_{v_7}	(v_6, v_7)	31	(v_7, v_3)	pop
	(v_6, v_7)	62	(v_7, v_3)	$swap(11)$
	(v_6, v_7)	72	(v_7, v_8)	$swap(22)$
τ_{v_8}	(v_3, v_8)	22	out_2	pop
	(v_7, v_8)	22	out_2	pop



local FFT	Out-I	In-Label	Out-I	op
τ_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$push(30)$
	(v_2, v_3)	21	(v_2, v_6)	$push(30)$
	(v_2, v_6)	30	(v_2, v_5)	$push(40)$
global FFT	**Out-I**	**In-Label**	**Out-I**	**op**
τ'_{v_2}	(v_2, v_3)	11	(v_2, v_6)	$swap(61)$
	(v_2, v_3)	21	(v_2, v_6)	$swap(71)$
	(v_2, v_6)	61	(v_2, v_5)	$push(40)$
	(v_2, v_6)	71	(v_2, v_5)	$push(40)$



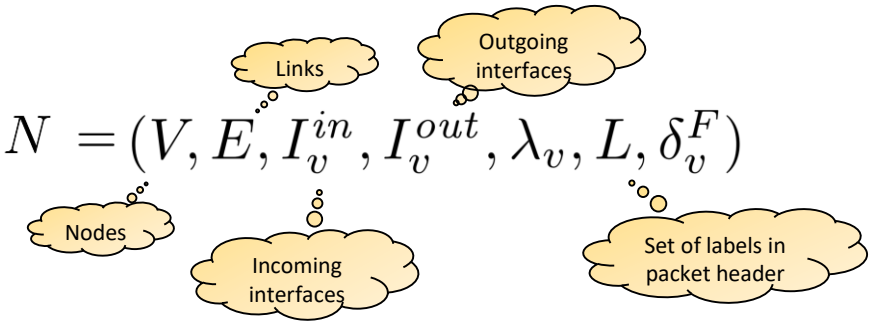
$$pX \Rightarrow qXX$$
$$pX \Rightarrow qYX$$
$$qY \Rightarrow rYY$$
$$rY \Rightarrow r$$
$$rX \Rightarrow pX$$

MPLS **configurations**, Segment Routing etc.

Pushdown Automaton and **Prefix Rewriting Systems** Theory

Network Model

- Network: a 7-tuple



Network Model

- Network: a 7-tuple

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$

Interface function

Interface function: maps outgoing interface to next hop node and incoming interface to previous hop node

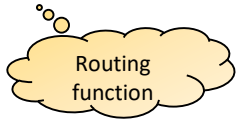
$$\lambda_v : I_v^{in} \cup I_v^{out} \rightarrow V$$

That is: $(\lambda_v(in), v) \in E$ and $(v, \lambda_v(out)) \in E$

Network Model

- Network: a 7-tuple

$$N = (V, E, I_v^{in}, I_v^{out}, \lambda_v, L, \delta_v^F)$$



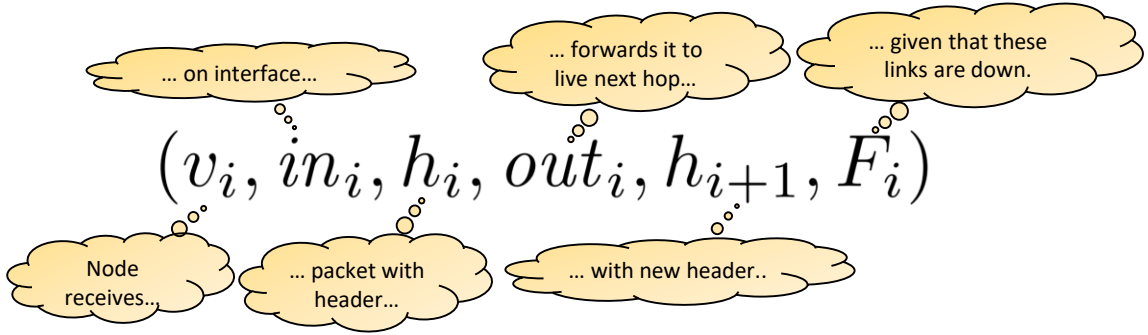
Routing function: for each set of failed links $F \subseteq E$, the routing function

$$\delta_v^F : I_v^{in} \times L^* \rightarrow 2^{(I^{out} \times L^*)}$$

defines, for all incoming interfaces and packet headers, outgoing interfaces together with modified headers.

Routing

Packet routing sequence can be represented using sequence of tuples:

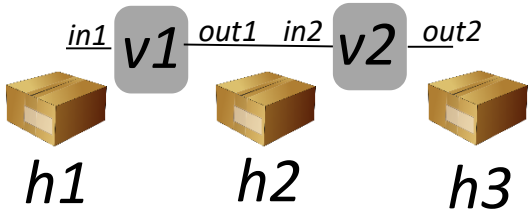


- Example: **routing** (in)finite sequence of tuples

$$(v_1, in_1, h_1, out_1, h_2, F_1),$$

$$(v_2, in_2, h_2, out_2, h_3, F_2),$$

$$\ldots$$



MPLS Network Model

- MPLS supports **three specific operations** on header sequences:

$$Op = \{swap(\ell) \mid \ell \in L\} \cup \{push(\ell) \mid \ell \in L\} \cup \{pop\}$$

- The **local routing table** can then be defined as

$$\tau_v : I_v^{in} \times (L \cup \{\bot\}) \hookrightarrow I_v^{out} \times Op$$

Interface + label

Maps to next hop and operation

- Local **link protection function** defines backup interface

$$\pi_v : I_v^{out} \times (L \cup \{\bot\}) \hookrightarrow I_v^{out} \times Op$$

protected

backup

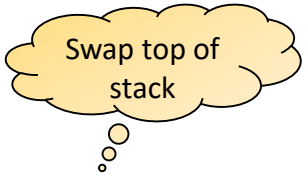
typically: push

Example Rules:
Regular Forwarding on Top-Most Label

Push:

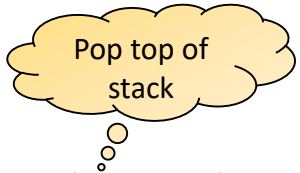
$$(v, in)\ell \to (v, out, 0)\ell'\ell \text{ if } \tau_v(in, \ell) = (out, push(\ell'))$$

Swap:



$$(v, in)\ell \to (v, out, 0)\ell' \text{ if } \tau_v(in, \ell) = (out, swap(\ell'))$$

Pop:



$$(v, in)\ell \to (v, out, 0) \text{ if } \tau_v(in, \ell) = (out, pop)$$

A Complex and Big Formal Language: Why Polynomial Time?

Arbitrary number k of failures: How can I avoid checking all $\binom{n}{k}$ many options?!

Classic Result in Automata Theory

- Classic result by **Büchi** 1964: the set of all reachable configurations of a pushdown automaton a is regular set
- Hence, we can operate only on Nondeterministic Finite Automata (NFAs) when reasoning about the pushdown automata
- The resulting **regular operations** are all polynomial time
  - Important result of **model checking**

Julius Richard Büchi

1924-1984

Swiss logician

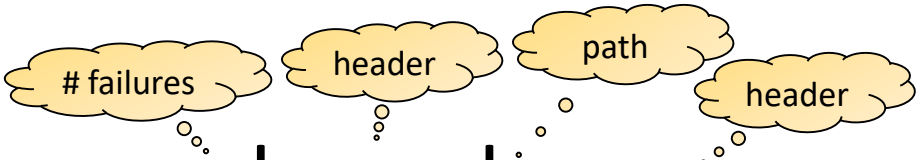
Tool and Query Language

Part 1: Parses query and constructs Push-Down System (PDS)

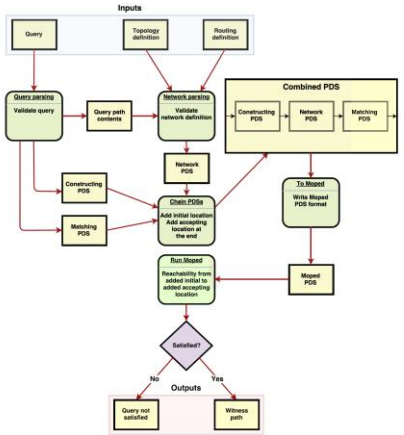
- In Python 3

Part 2: Reachability analysis of constructed PDS

- Using ***Moped*** tool



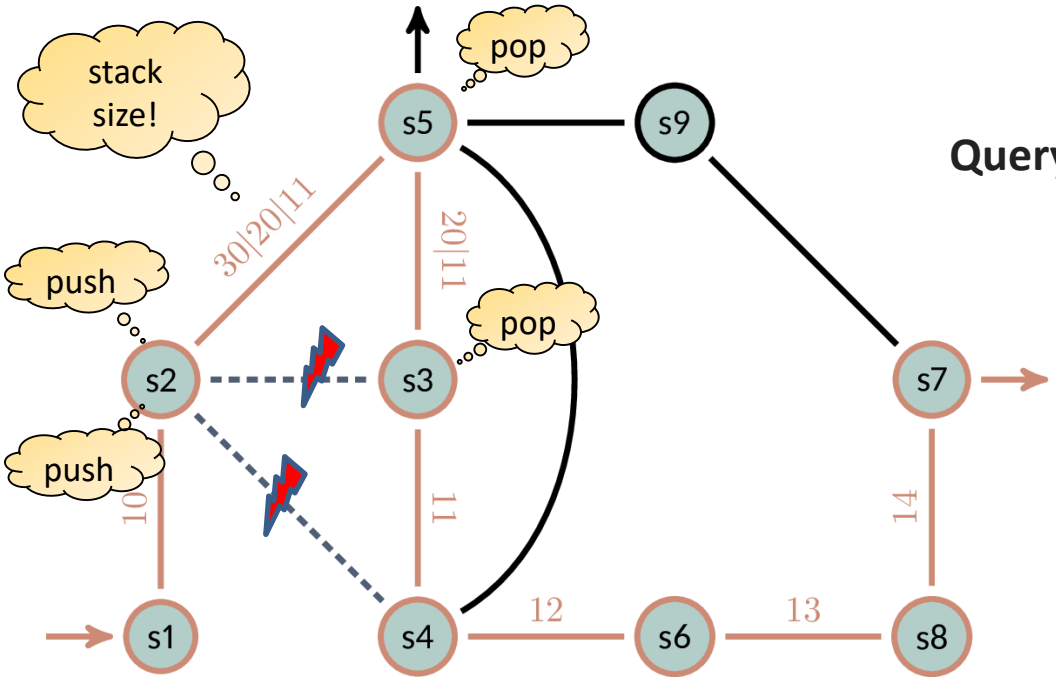
Regular query language



query processing flow

Example: Traversal Testing With 2 Failures

Traversal test with k=2: Can traffic starting with [] go through s5, under up to k=2 failures?



2 failures

Query: k=2 [] s1 >> s5 >> s7 []

YES!

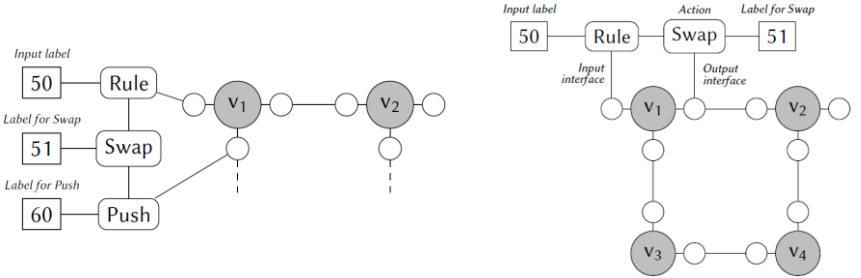
(Gives witness!)

Speeding things up with Deep Learning? And synthesis.

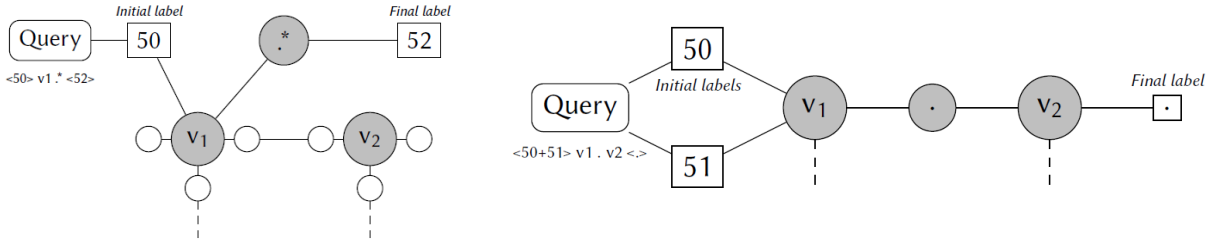
DeepMPLS: Fast Analysis of MPLS Configurations Using Deep Learning. Fabien Geyer and Stefan Schmid. **IFIP Networking**, Warsaw, Poland, May 2019.

Deep Learning for MPLS: DeepMPLS (s. talk by Fabien Geyer)

- Yes sometimes **without losing guarantees**
- Extend **graph-based neural networks**
- **Predict** counter-examples and **fixes**



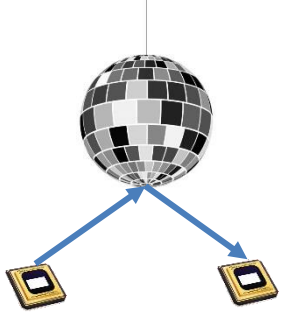
Network topologies and MPLS rules

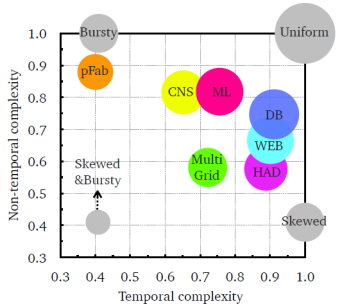


Network topologies and query

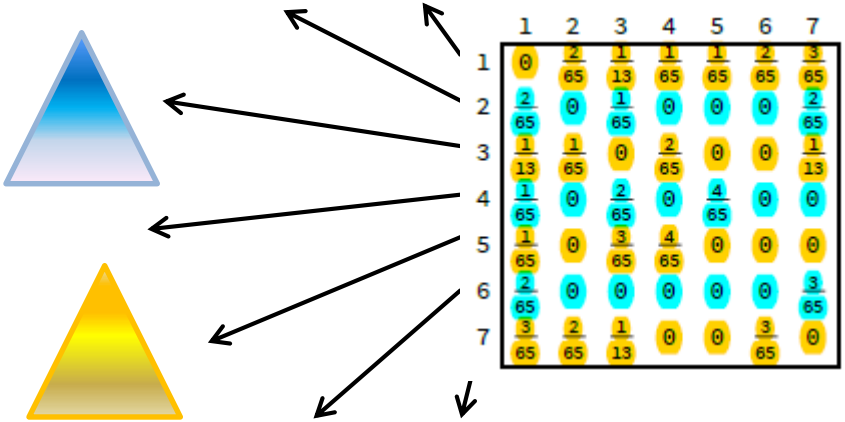
Challenges of Self-* Networks

- Can a self-* network realize its **limits**?
- E.g., when quality of **input data** is not good enough?
- When to hand over to human? Or **fall back** to „safe/oblivious mode“?
- Can we learn from self-driving **cars**?





Thank you! Questions?



Further Reading

Demand-aware 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.

Measuring the Complexity of Network Traffic Traces
Chen Griner, Chen Avin, Manya Ghobadi, and Stefan Schmid.
arXiv, 2019.

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.

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.

Further Reading

What-if analysis

P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures
Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen.
14th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Heraklion/Crete, Greece, December 2018.
Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks
Stefan Schmid and Jiri Srba.
37th IEEE Conference on Computer Communications (**INFOCOM**), Honolulu, Hawaii, USA, April 2018.

Secure sampling and dataplane

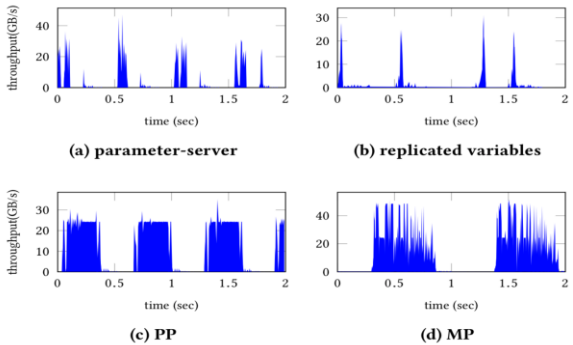
Preacher: Network Policy Checker for Adversarial Environments
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.
38th International Symposium on Reliable Distributed Systems (**SRDS**), Lyon, France, October 2019.
MTS: Bringing Multi-Tenancy to Virtual Switches
Kashyap Thimmaraju, Saad Hermak, Gabor Retvari, and Stefan Schmid.
USENIX Annual Technical Conference (**ATC**), Renton, Washington, USA, July 2019.
Taking Control of SDN-based Cloud Systems via the Data Plane (Best Paper Award)
Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid.
ACM Symposium on SDN Research (**SOSR**), Los Angeles, California, USA, March 2018.

Backup Slides

How Predictable is Traffic?

Even if reconfiguration fast, control plane (e.g., data collection) can become a bottleneck. However, many good examples:

- Machine learning applications
- Trend to disaggregation (specialized racks)
- Datacenter communication dominated by elephant flows
- Etc.



ML workload (GPU to GPU):
deep convolutional neural network

Predictable from their dataflow graph