Networks in the Age of Distributed Computation

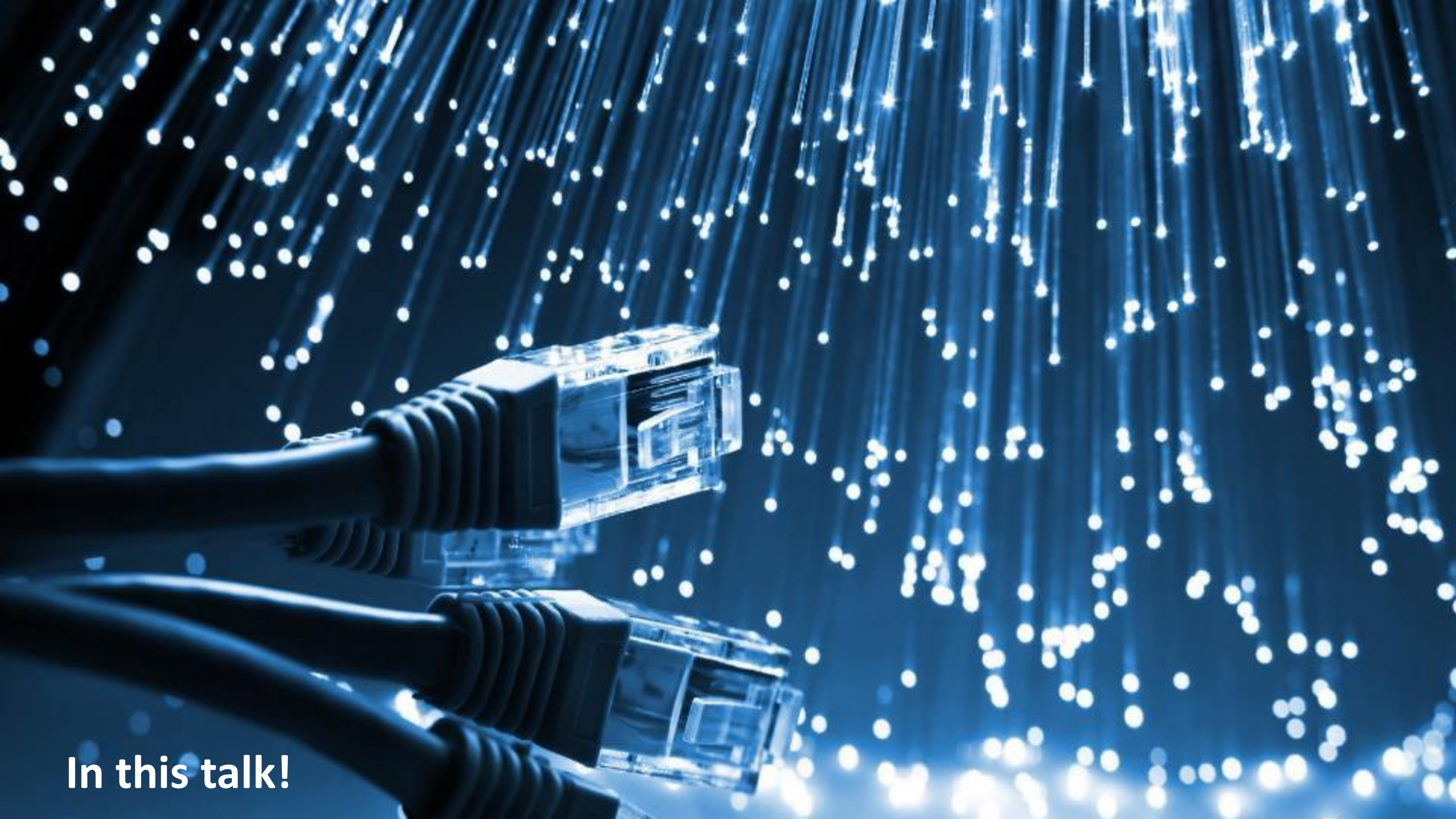
Stefan Schmid (TU Berlin)

Networks in the Age of Distributed Computation

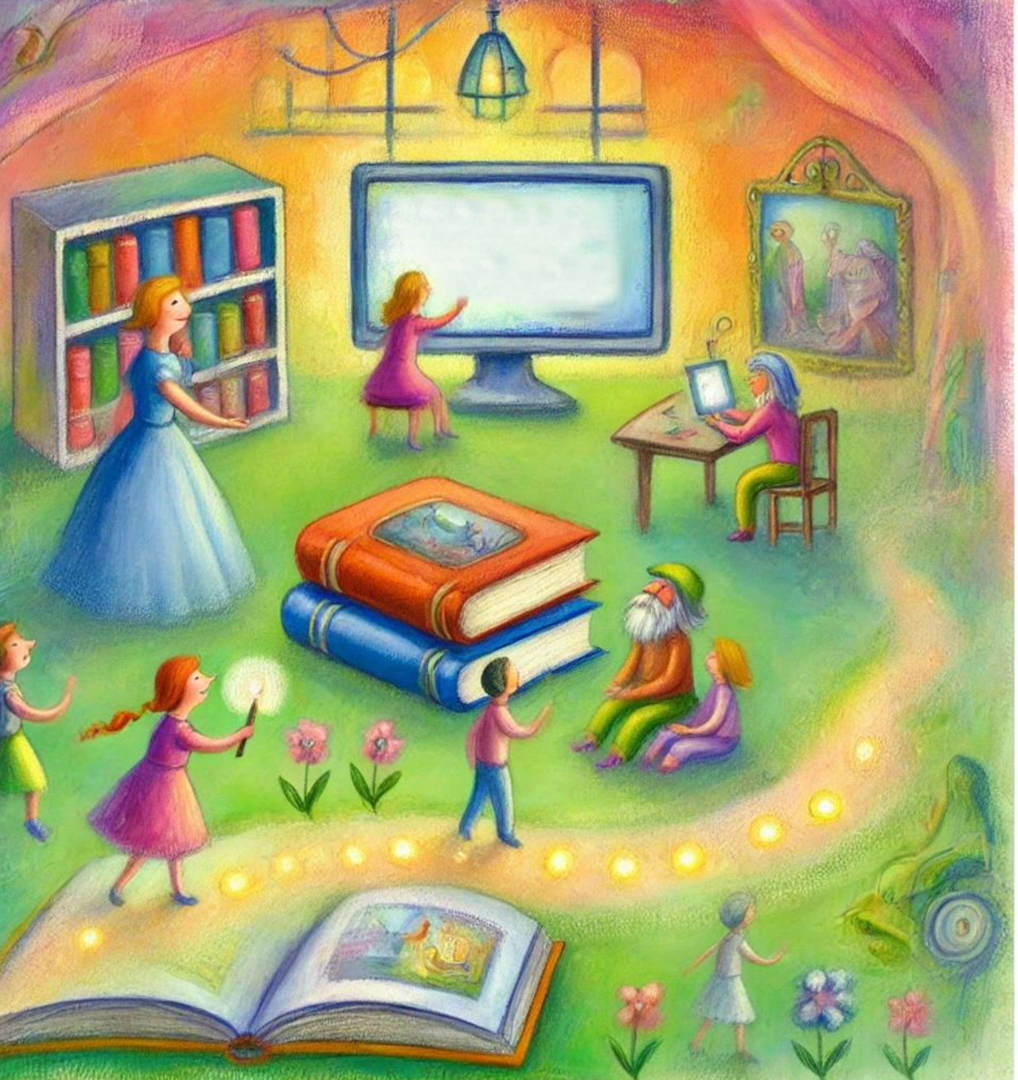
Stefan Schmid (TU Berlin)

See you at
DISC 2025 ☺

Not in this talk!



In this talk!



Wired networks?

- "***Cosy*** living room": well-understood and just works
- Passed ***test of time***
- Should and cannot be changed

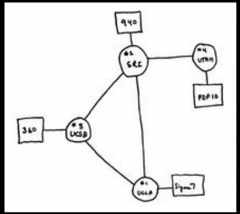
Wired networks!

- Place where ***fantastic innovations*** are happening ☺ On all layers.
- For ***performance*** and ***dependability***
- Still: specific and interesting ***contraints*** due to simple but fast hardware
- DISC bonus (compared to wireless): ***simple*** and discrete models ☺

Why do networks evolve?

The Internet 50 years ago. . .

When the Internet was designed…



… for a different purpose and context:

- Goal: connectivity between fixed locations / "super computers"
- For researchers : Simple applications like email and file transfer

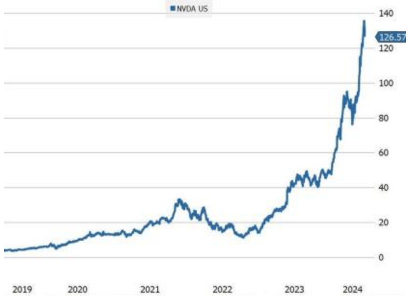
Now we live in a different era:
Age of Computation

Datacenters („hyperscale“)

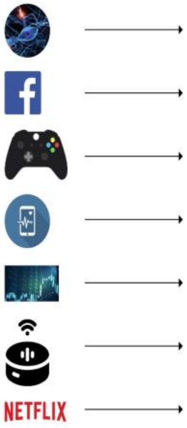
Data intensive applications requiring significant processing.

Age of Computation: Evidence

Nvidia: fastest growing company ever



Datacenters („hyperscale“)



Data intensive applications requiring significant processing.


Amazon buys nuclear-powered data center from Talen

Thu, Mar 7, 2024, 2:01PM | Nuclear News

Susquehanna nuclear plant in Salem Township, Penn., along with the data center in foreground. (Photo: Talen Energy)

Training even across ***multiple datacenters*** (and ***powerplants***)!

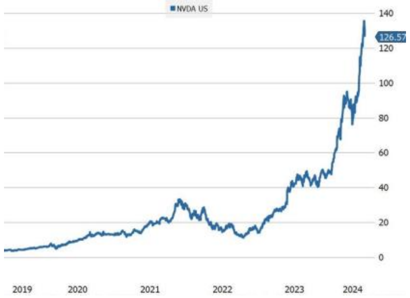
Age of Computation: Evidence

Datacenters („hyperscale“)

Data intensive applications requiring significant processing.

Energy consumption and probably also computation trends will likely stay. ***Kardashev Scale*** even classifies civilizations by their energy use!

Nvidia: fastest growing company ever



Amazon buys nuclear-powered data center from Talen

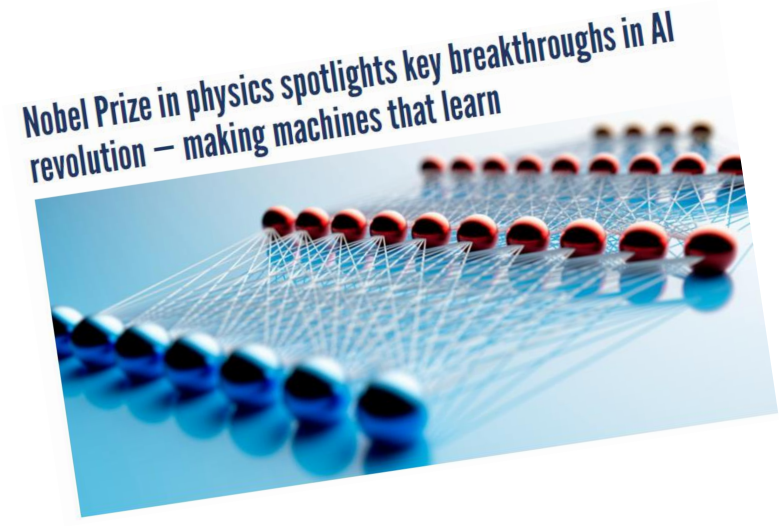
Thu, Mar 7, 2024, 2:01PM | Nuclear News

Susquehanna nuclear plant in Salem Township, Penn., along with the data center in foreground. (Photo: Talen Energy)

Training even across ***multiple datacenters*** (and ***powerplants***)!

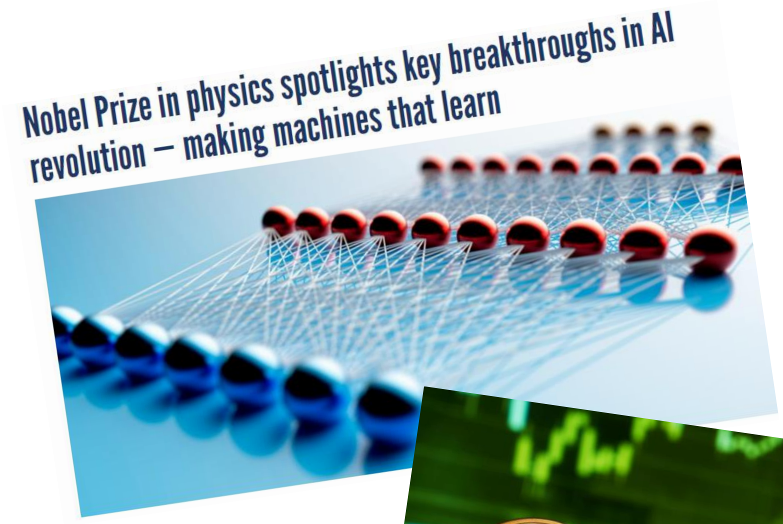
Age of Computation: More Evidence

Nobel Prizes in Physics and Chemistry…



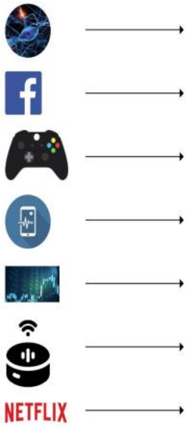
Age of Computation: More Evidence

... and soon also in Economics and Literature?!



Actually: Age of *Distributed* Computation

Datacenters („hyperscale“)



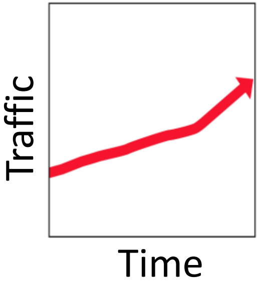
Distributed applications…

Actually: Age of *Distributed* Computation

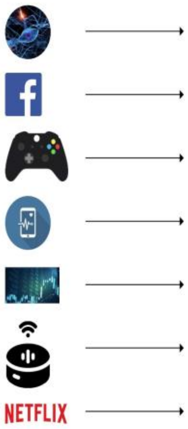
Datacenters („hyperscale“)

Distributed applications…

… require networks!



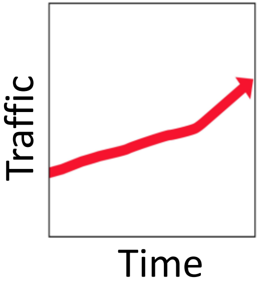
Actually: Age of *Distributed* Computation



Datacenters („hyperscale“)

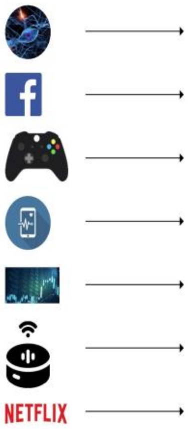
Distributed applications…

… require networks!



Networks are a critical infrastructure of digital society. Especially *to*, *from*, and *inside* datacenter networks!

Actually: Age of *Distributed* Computation

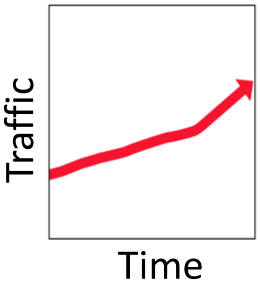


Datacenters („hyperscale“)

Networks are a critical infrastructure of digital society. Especially *to*, *from*, and *inside* datacenter networks!

Distributed applications…

… require networks!

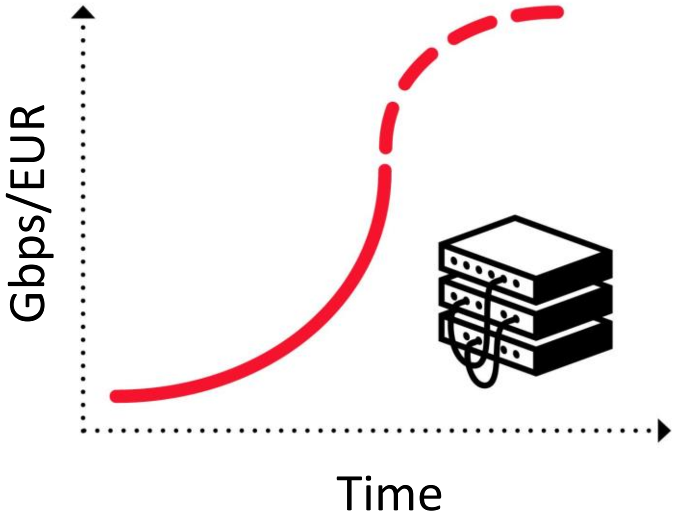


Challenge and Opportunity: Networks become larger and larger

- Also here: end of *Moore's Law in networking*
  - Transistor density rates stalling

- Hence: need more equipment, larger networks

- *Opportunity:* network itself forms large *distributed system*! With specialized but fast hardware.
  - E.g., in-network processing to ***speed up all-reduce***?



Example Innovation on Transport Layer: Congestion Control

- A highly ***decentralized problem***!

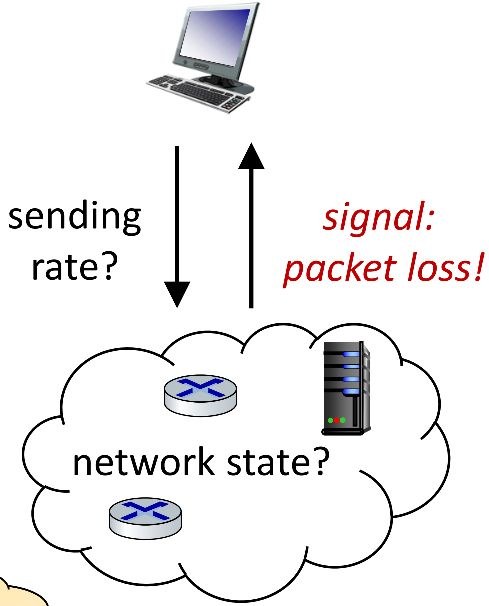
- How much packets dropped in Internet today?

Example Innovation on Transport Layer: Congestion Control

- A highly ***decentralized problem***!

- How much packets dropped in Internet today?
  - Not negligible.

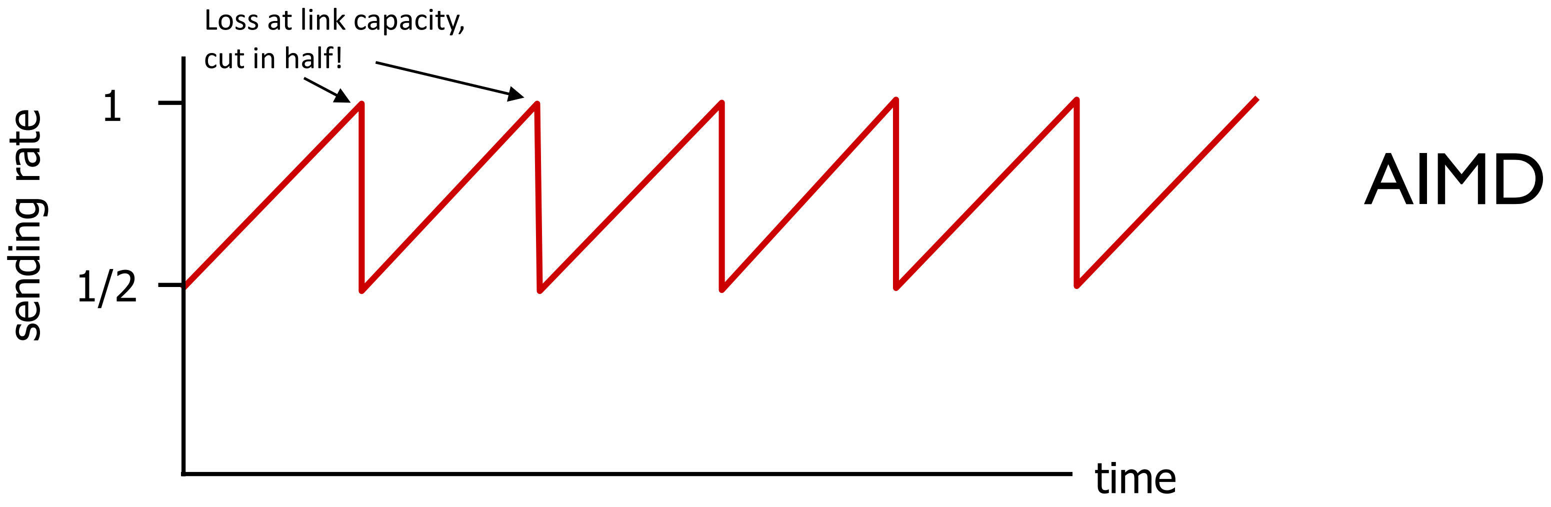
- Because of the way we ***control*** congestion!
  - A TCP sender cannot directly "see" traffic load in network…
  - … so ***opportunistically probes***: increases sending rate until loss
  - So ***TCP needs packet loss*** to determine their sending rate



How optimal is this?

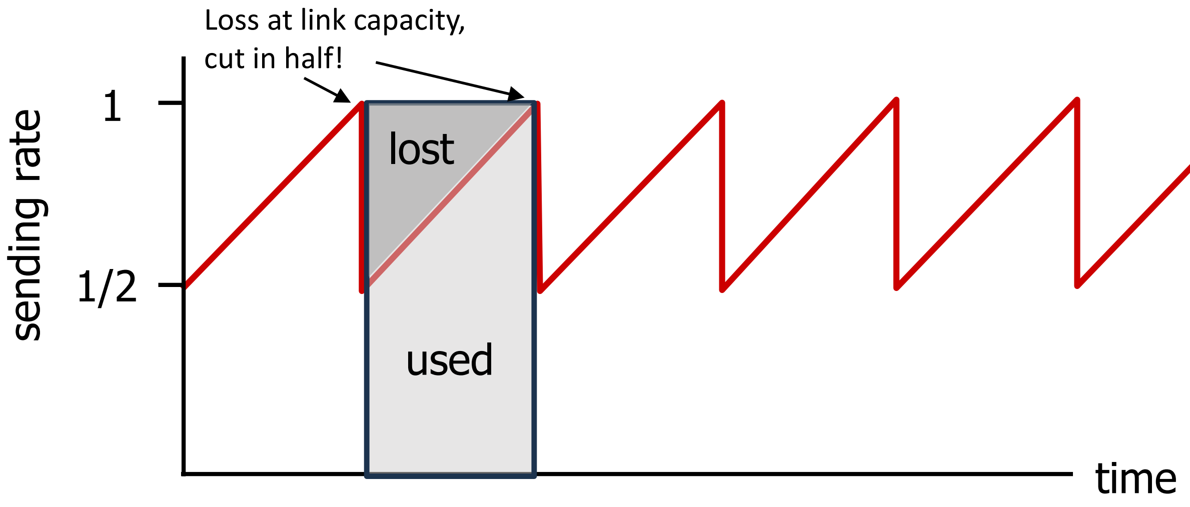
Example Innovation on Transport Layer: Congestion Control

- Well, huge success for decades: additive increase, multiplicative decrease (AIMD)
  - No congestion collapse since 1990s
  - Same mechanism since 30+ years, while ***traffic increased by factor 1 billion***!



Example Innovation on Transport Layer: Congestion Control

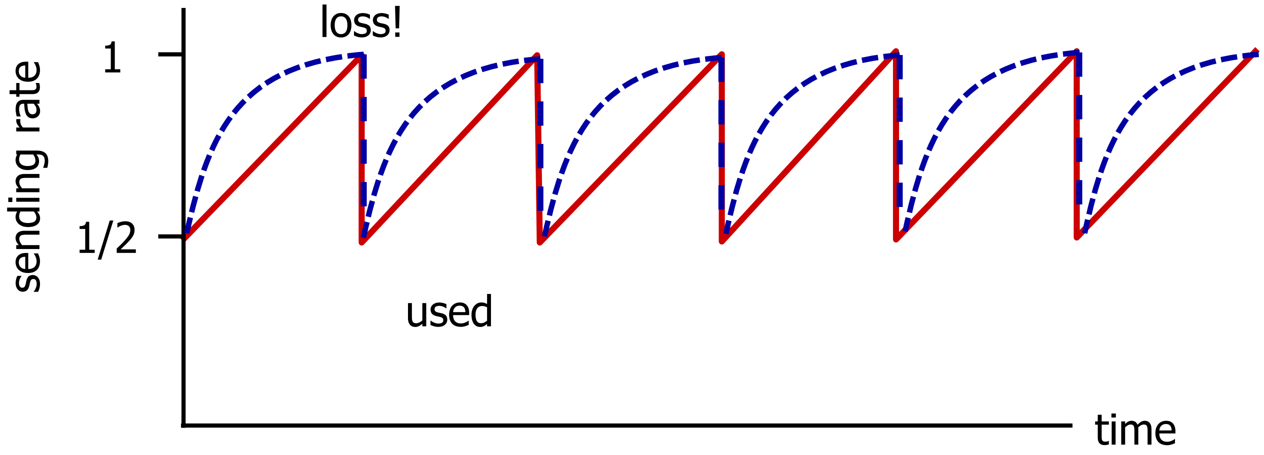
- Well, huge success for decades: additive increase, multiplicative decrease (AIMD)
  - No congestion collapse since 1990s
  - Same mechanism since 30+ years, while ***traffic increased by factor 1 billion***!



AIMD:
efficiency only ~75%

Example Innovation on Transport Layer: Congestion Control

- A little bit better: Linux‘ TCP CUBIC
  - Idea: increase sending rate faster until „near last packet loss“-rate

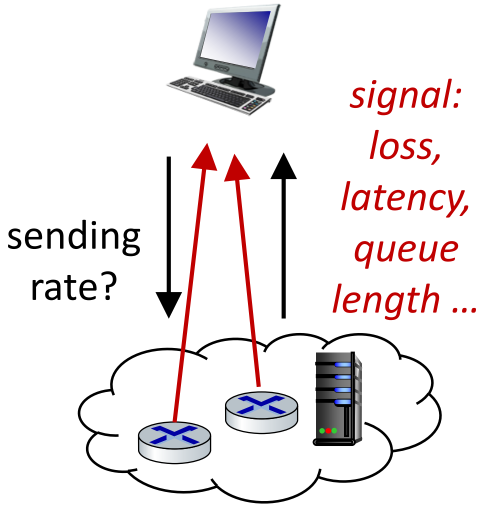


TCP CUBIC

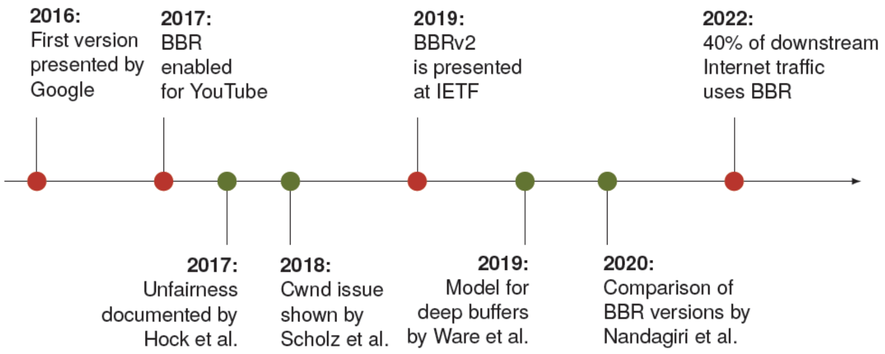
Can we do better? Significant efforts right now!

- Still: performance could be better
  - Google's BBR, QUIC, Netflix, ECN, etc.: additional ***signals*** about congestion (e.g., ***latency***)
  - Also: congestion control in ***datacenters*** (e.g., to handle ML workloads)

- ***Opportunity for DISC***: Many of these protocols have **no theoretical underpinnings**!
  - And indeed, have issues, e.g., regarding ***fairness***
  - Often hard to ***catch issues*** empirically and or in simulations!



Theory needed! Example BBR.



- BBR: relatively ***fast and large*** deployment
- But with ***fairness*** and other ***issues***
- ***Needed several adjustments*** *and new versions* still under development

Literature: Model-Based Insights on the Performance, Fairness, and Stability of BBR. Scherrer et al., ACM IMC 2022.

Example Innovation on Network Layer: Segment Routing („Valiant Routing“)

Example Innovation on Network Layer: Segment Routing („Valiant Routing“)

Which routes are taken by packets in today's communication networks?

How can I influence routes?

Example Innovation on Network Layer: Segment Routing („Valiant Routing“)

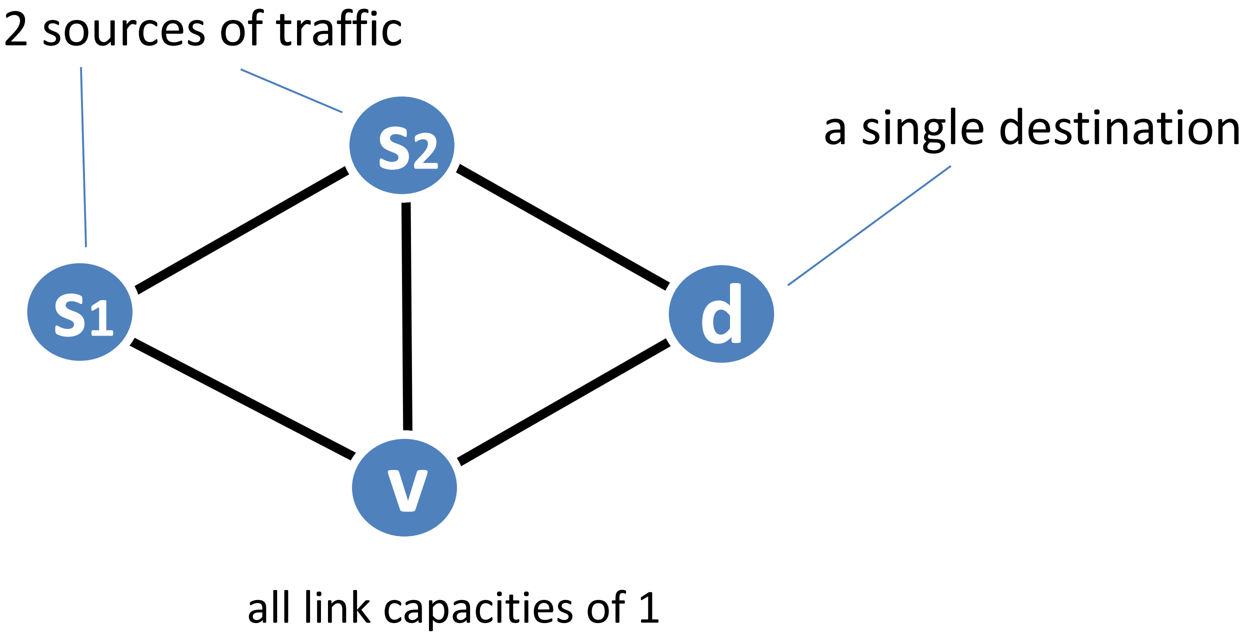
Which routes are taken by packets in today’s communication networks?

Shortest paths only!

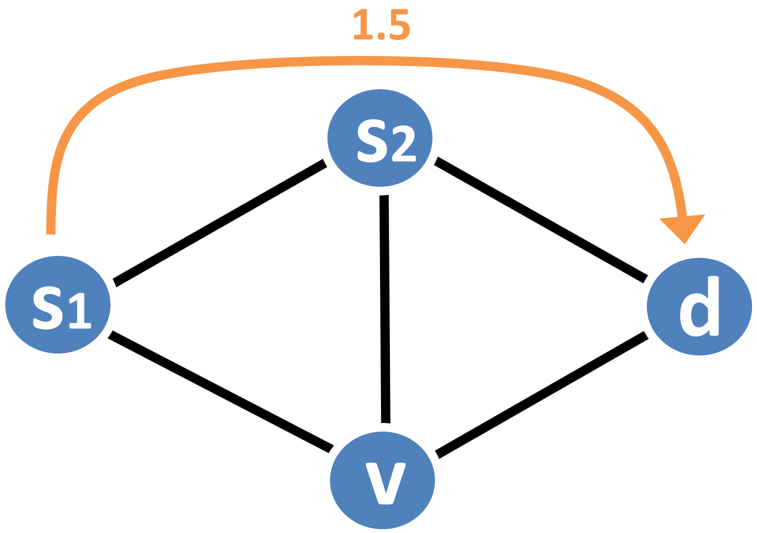
How can I influence routes?

Traffic engineering: change link weights!

Traffic Engineering



Traffic Engineering

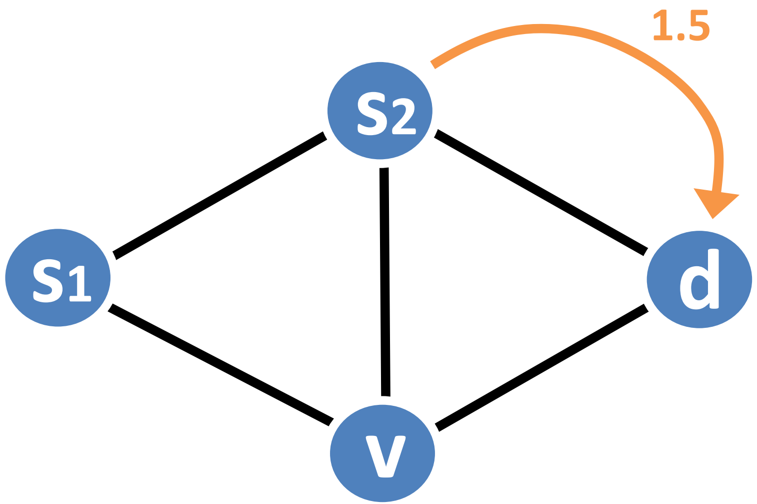


all link capacities of 1

Only two possible demand matrices:

1. **only $s_1 \rightarrow d = 1.5$**
2. only $s_2 \rightarrow d = 1.5$

Traffic Engineering

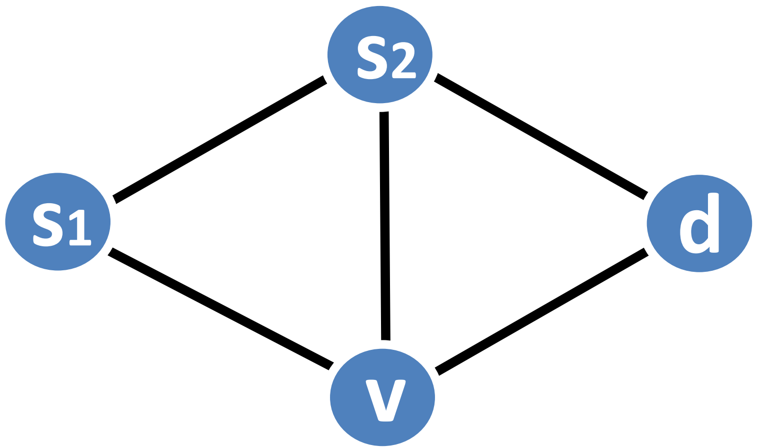


all link capacities of 1

Only two possible demand matrices:

1. only s1 → d = 1.5
2. **only s2 → d = 1.5**

Traffic Engineering



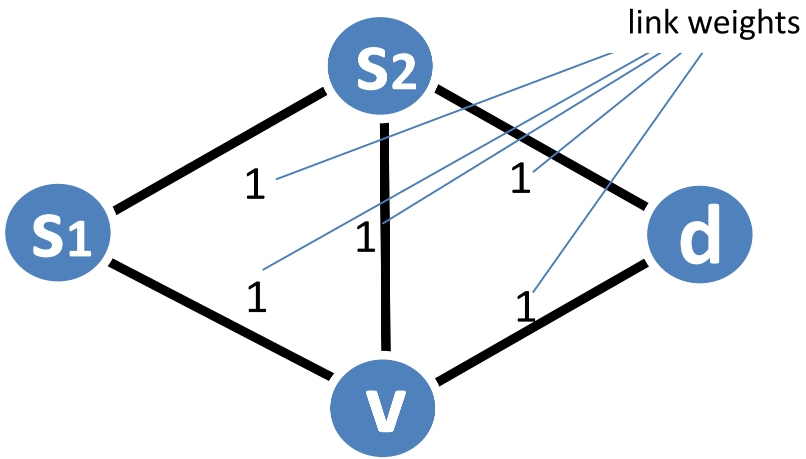
all link capacities of 1

Only two possible demand matrices:

1. only s1 → d = 1.5
2. only s2 → d = 1.5

How to set link weights to serve this traffic?
Without violating capacities and to minimize load.

Traffic Engineering



all link capacities of 1

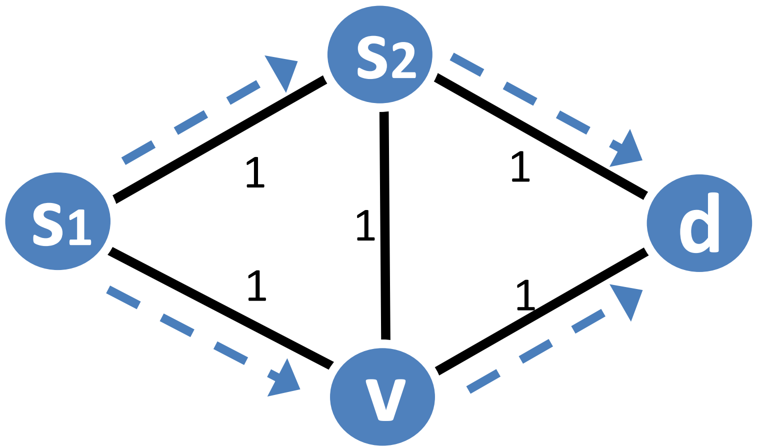
Only two possible demand matrices:

1. only s1→ d = 1.5
2. only s2→ d = 1.5

Traditional traffic engineering:

- operator sets link weights
- per-destination routing

Traffic Engineering



all link capacities of 1

 shortest path DAG

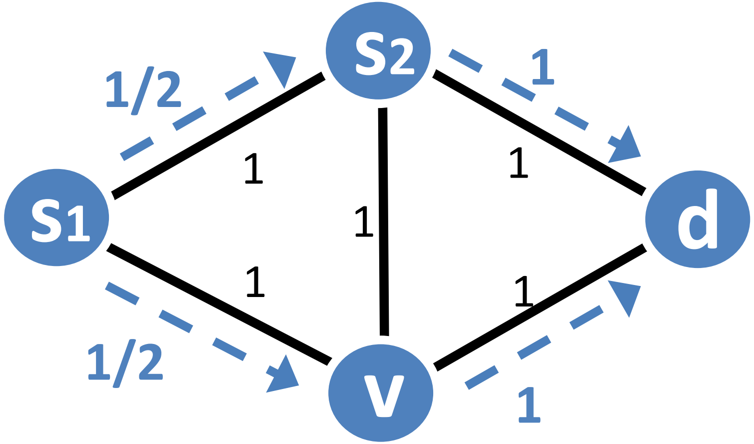
Only two possible demand matrices:

1. only $s_1 \rightarrow d = 1.5$
2. only $s_2 \rightarrow d = 1.5$

Traditional traffic engineering:

- operator sets link weights
- per-destination routing
- **shortest-path DAGs**

Traffic Engineering



all link capacities of 1


shortest path DAG

1/2 splitting ratio

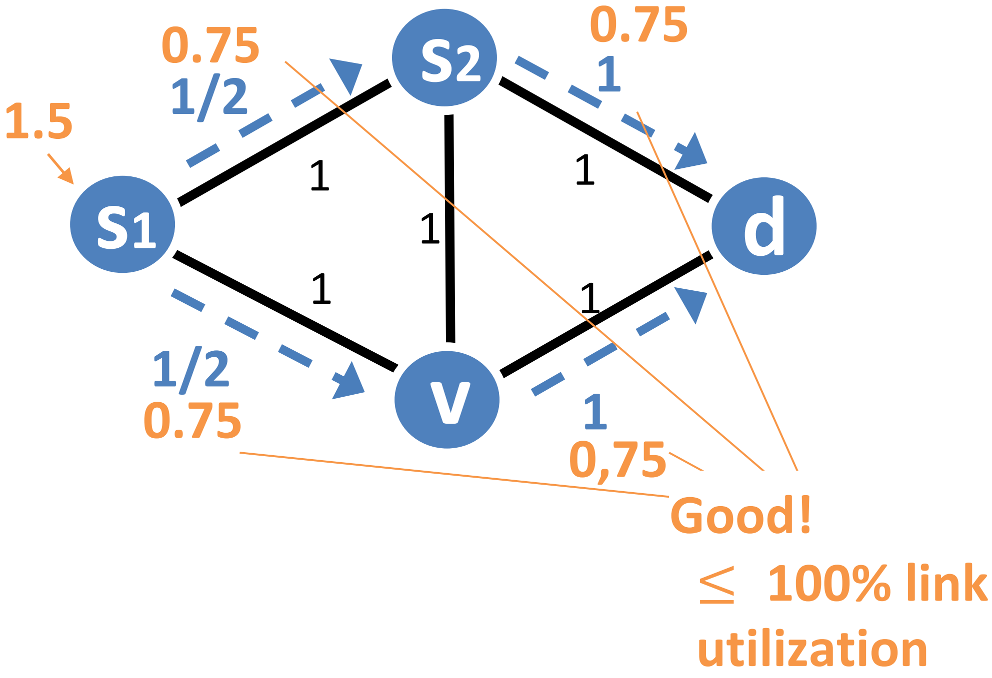
Only two possible demand matrices:

1. only $s_1 \rightarrow d = 1.5$
2. only $s_2 \rightarrow d = 1.5$

Traditional traffic engineering:

- operator sets link weights
- per-destination routing
- shortest paths DAGs
- **equal-split**

Traffic Engineering



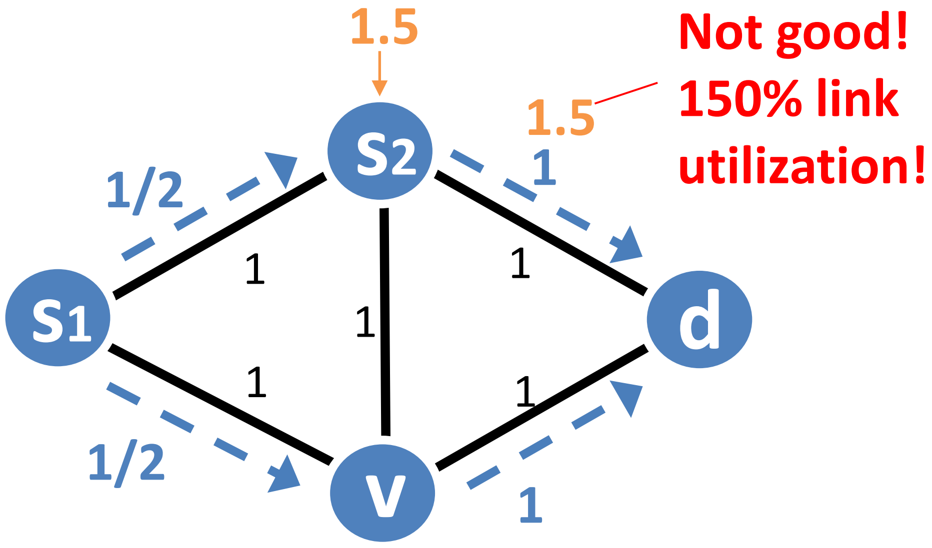
Only two possible demand matrices:

1. **only s1 → d = 1.5**
2. only s2 → d = 1.5

Traditional traffic engineering:

- operator sets link weights
- per-destination routing
- shortest paths DAGs
- equal-split

Traffic Engineering



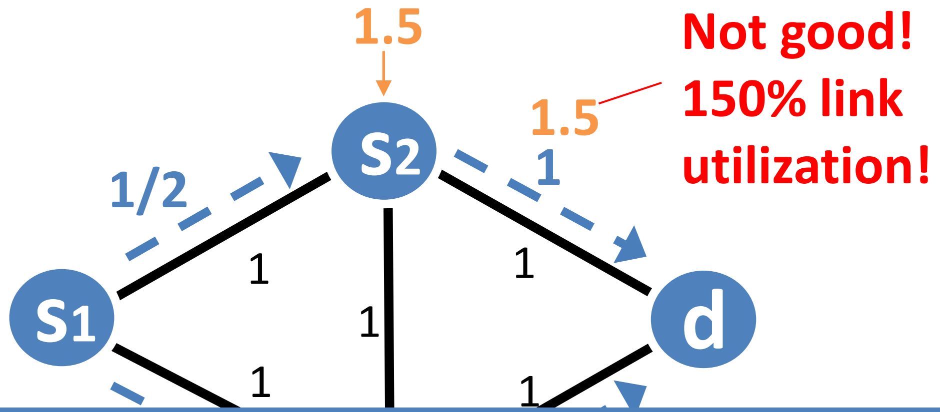
Only two possible demand matrices:

1. only $s_1 \rightarrow d = 1.5$
2. **only $s_2 \rightarrow d = 1.5$**

Traditional traffic engineering:

- operator sets link weights
- per-destination routing
- shortest paths DAGs
- equal-split

Traffic Engineering



Only two possible demand matrices:

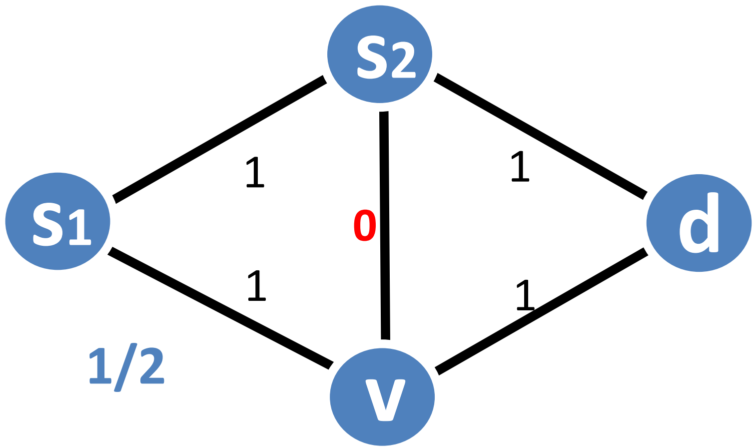
1. only s1 → d = 1.5
2. **only s2 → d = 1.5**

Traditional traffic engineering:

No link-weight assignment can attain
≤ 100% link utilization!
(for both demand matrices, although in principle enough capacity available!)

- equal-split

What about this?!



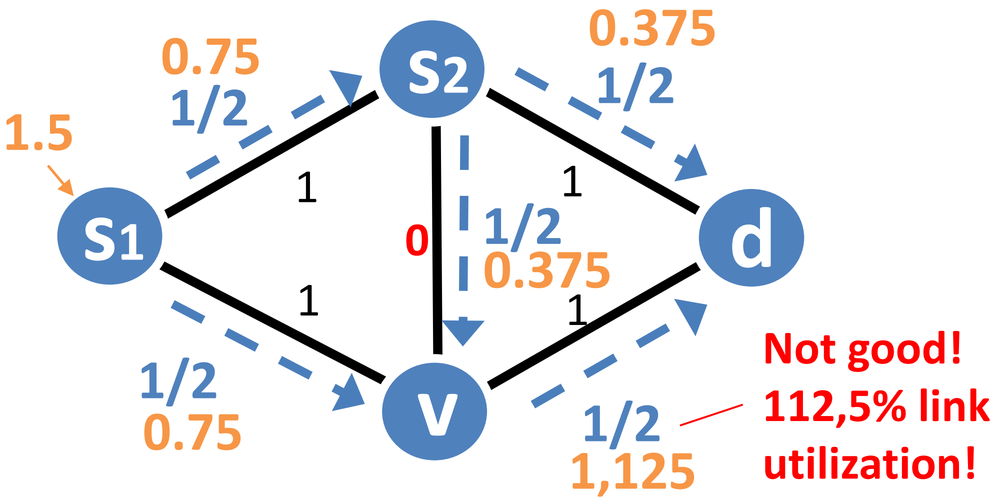
Only two possible demand matrices:

1. only s1 → d = 1.5
2. only s2 → d = 1.5

Traditional traffic engineering:

- operator sets link weights
- per-destination routing
- shortest paths DAGs
- equal-split

What about this?!



Careful: first flow now splits *twice*! Two more shortest paths later.

Only two possible demand matrices:

1. **only s1 → d = 1.5**
2. only s2 → d = 1.5

Traditional traffic engineering:

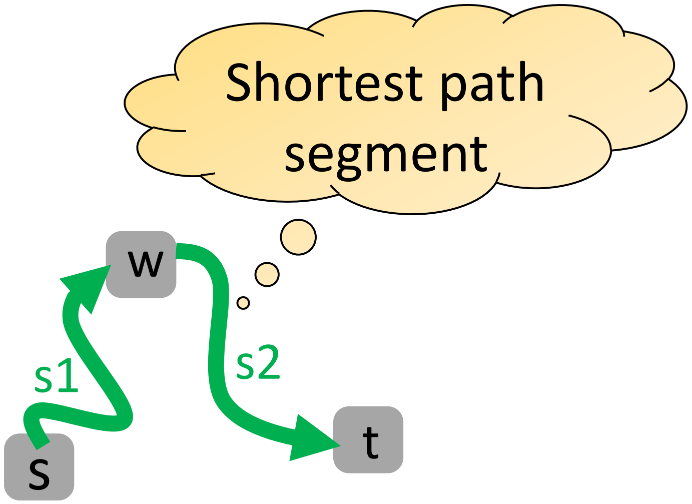
- operator sets link weights
- per-destination routing
- shortest paths DAGs
- equal-split

Powerful Extension: Segment Routing

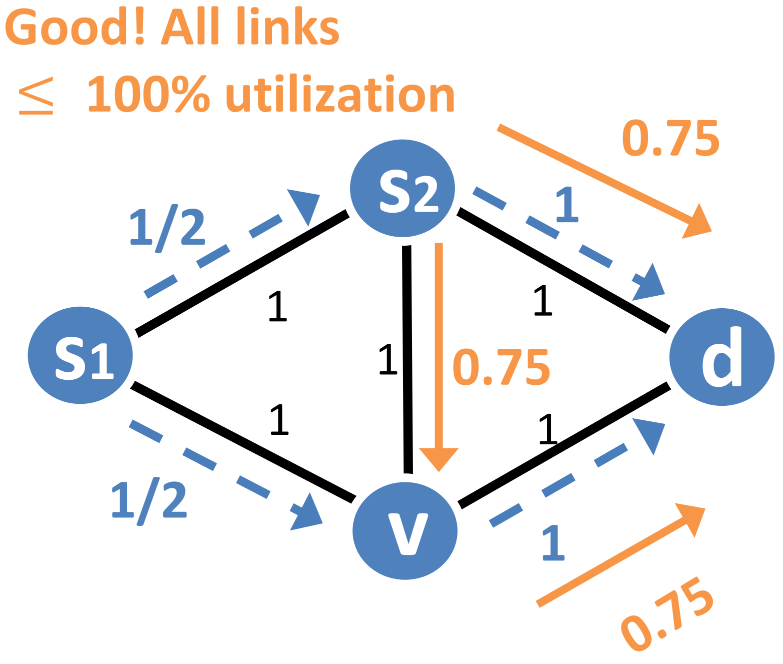
„Valiant Routing for IP Networks"

- Can define **waypoints** between source and destination
  - Like *Valiant routing*: important technique in oblivious routing (but random waypoint)

- Shortest paths on „**segments**" between waypoints (and source and destination)



Traffic Engineering with Segment Routing



Half of traffic from s_2 via waypoint v!

Only two possible demand matrices:

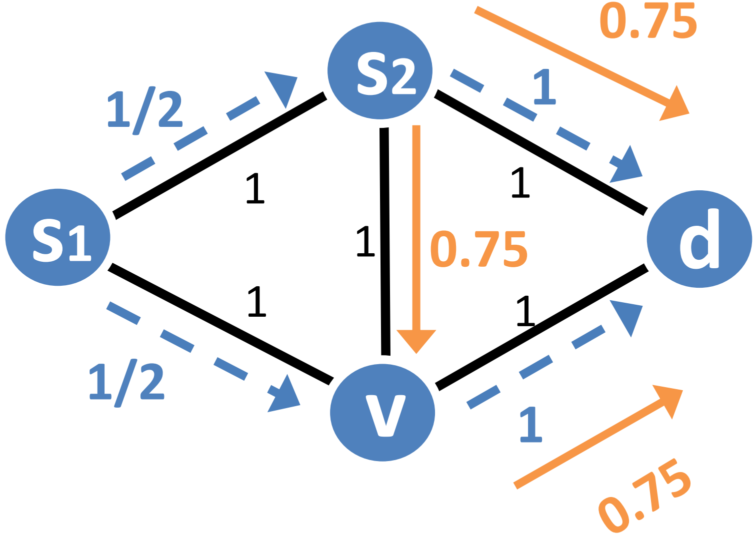
1. only s1 → d = 1.5
2. **only s2 → d = 1.5**

Segment Routing:

- Can push a **waypoint w** between source s2 and destination d
- Then: shortest path from *s to w*, and shortest path from *w to d*

Traffic Engineering with Segment Routing

Good! All links
≤ 100% utilization



Half of traffic from s_2 via waypoint v!

Only two possible demand matrices:

1. only $s_1 \rightarrow$ d = 1.5
2. **only $s_2 \rightarrow$ d = 1.5**

Segment Routing:

- Can push a **waypoint w** between source s2 and destination d
- Then: shortest path from *s to w*, and shortest path from *w to d*

Literature: Traffic Engineering with Joint Link Weight and Segment Optimization. Parham et al., ACM CoNEXT, 2021.

Example: Many more…

- New Ethernet versions
  - ***Automotive*** Ethernet
  - Ethernet for ***datacenters***
  - …
- Hollow-fiber: faster speed of light!
  - Cost(***latency***)>>>Cost(bandwidth)
- Optical and reconfigurable networks

Ultra Ethernet
Consortium

ENTERPRISE NETWORKING

Hollow Fiber: The New Option for Low Latency

Very low latency hollow fiber services target the financial services industry today and offer another option for serving latency-sensitive applications in the broader business market.

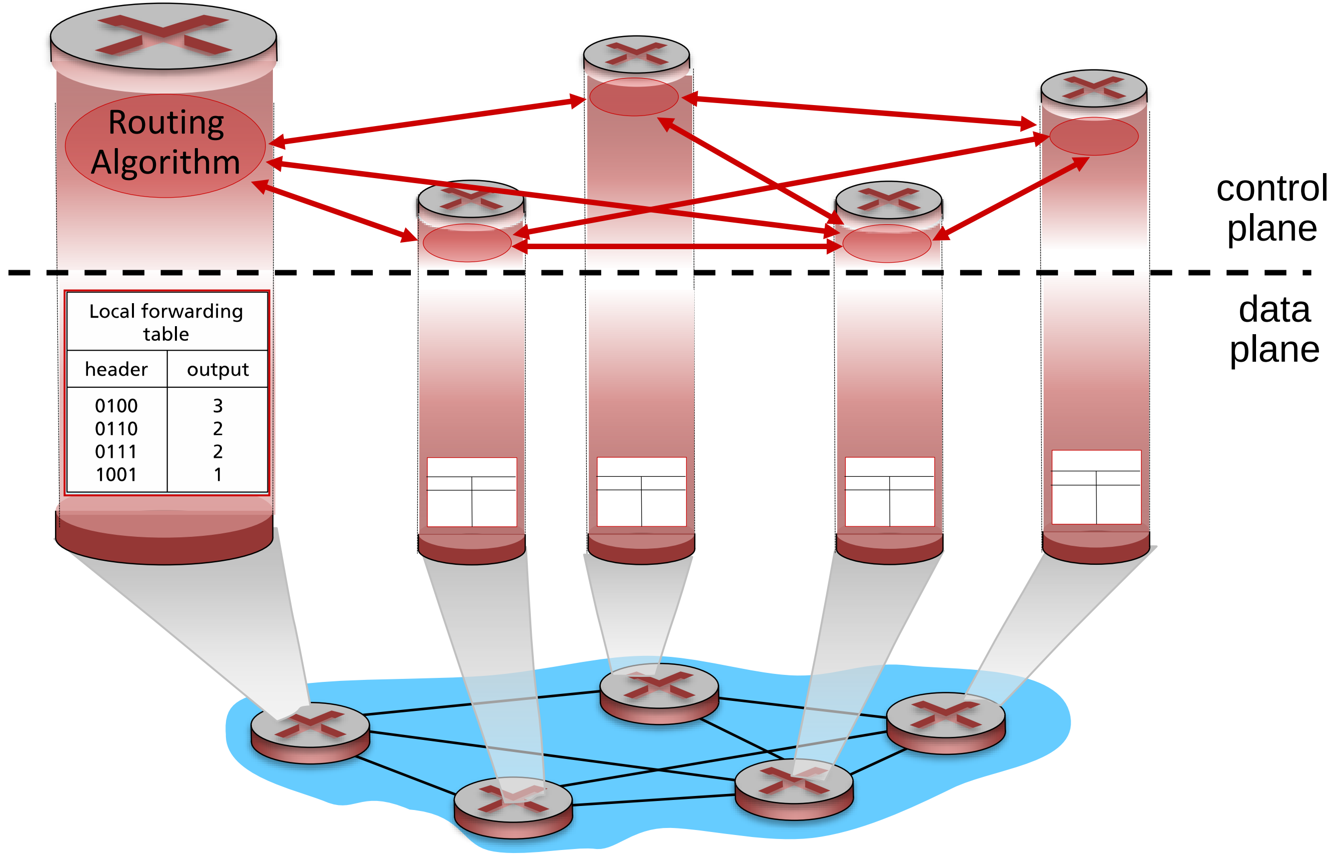
By *Michael Finneran*

December 18, 2020

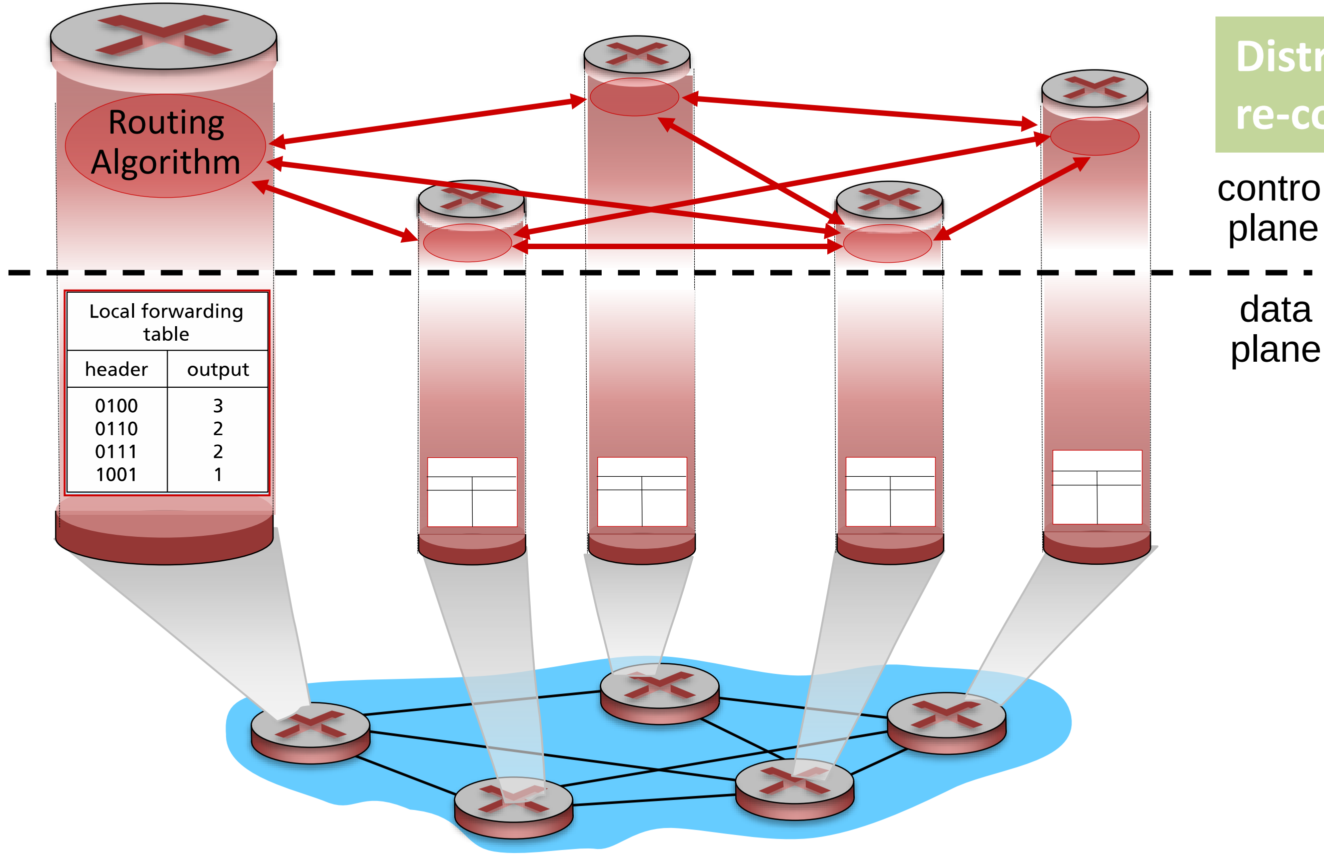
Roadmap: Two Examples

- Resilient routing
- Datacenter networks

Two Options to React to Link Failures



Two Options to React to Link Failures

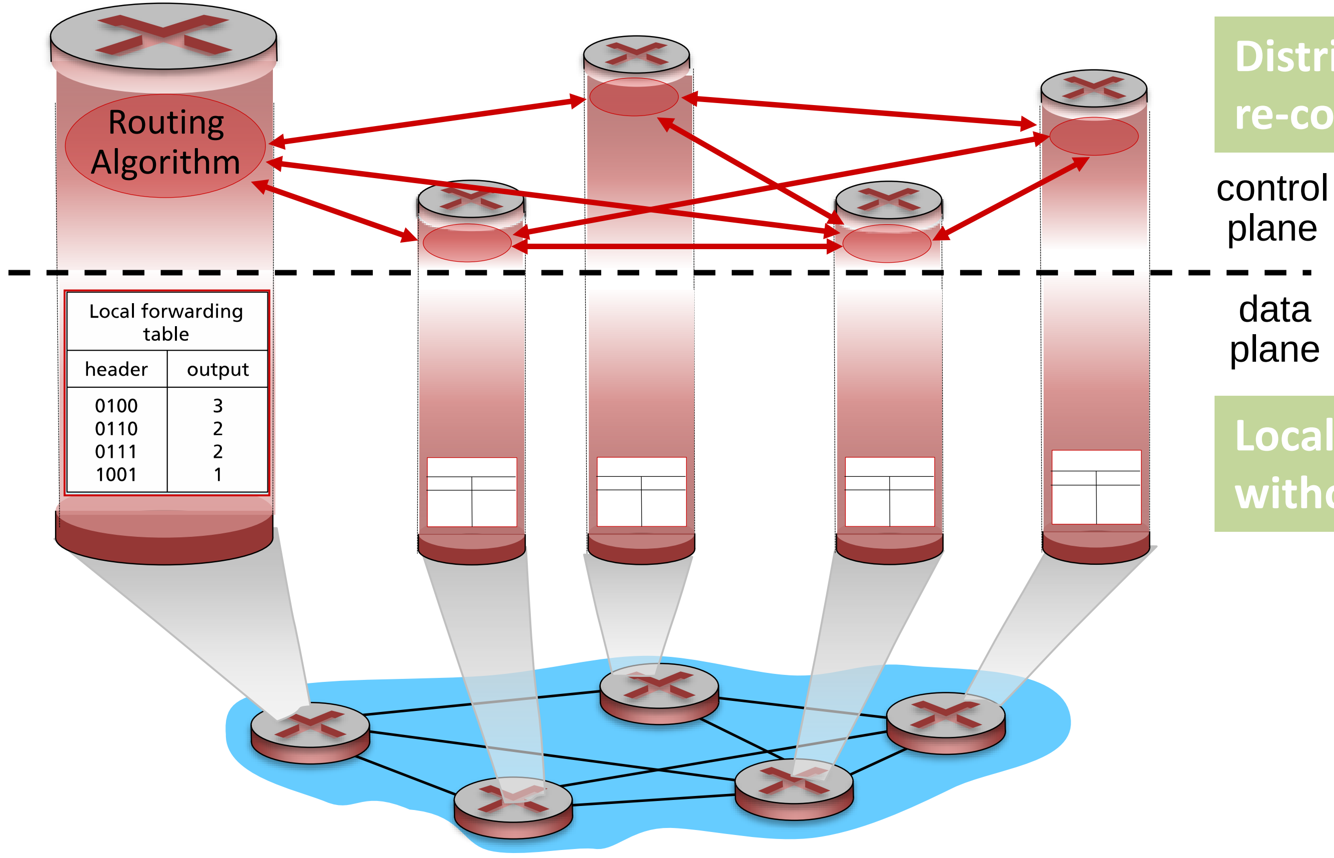


Distributed algorithms, e.g., re-compute shortest paths

control plane

data plane

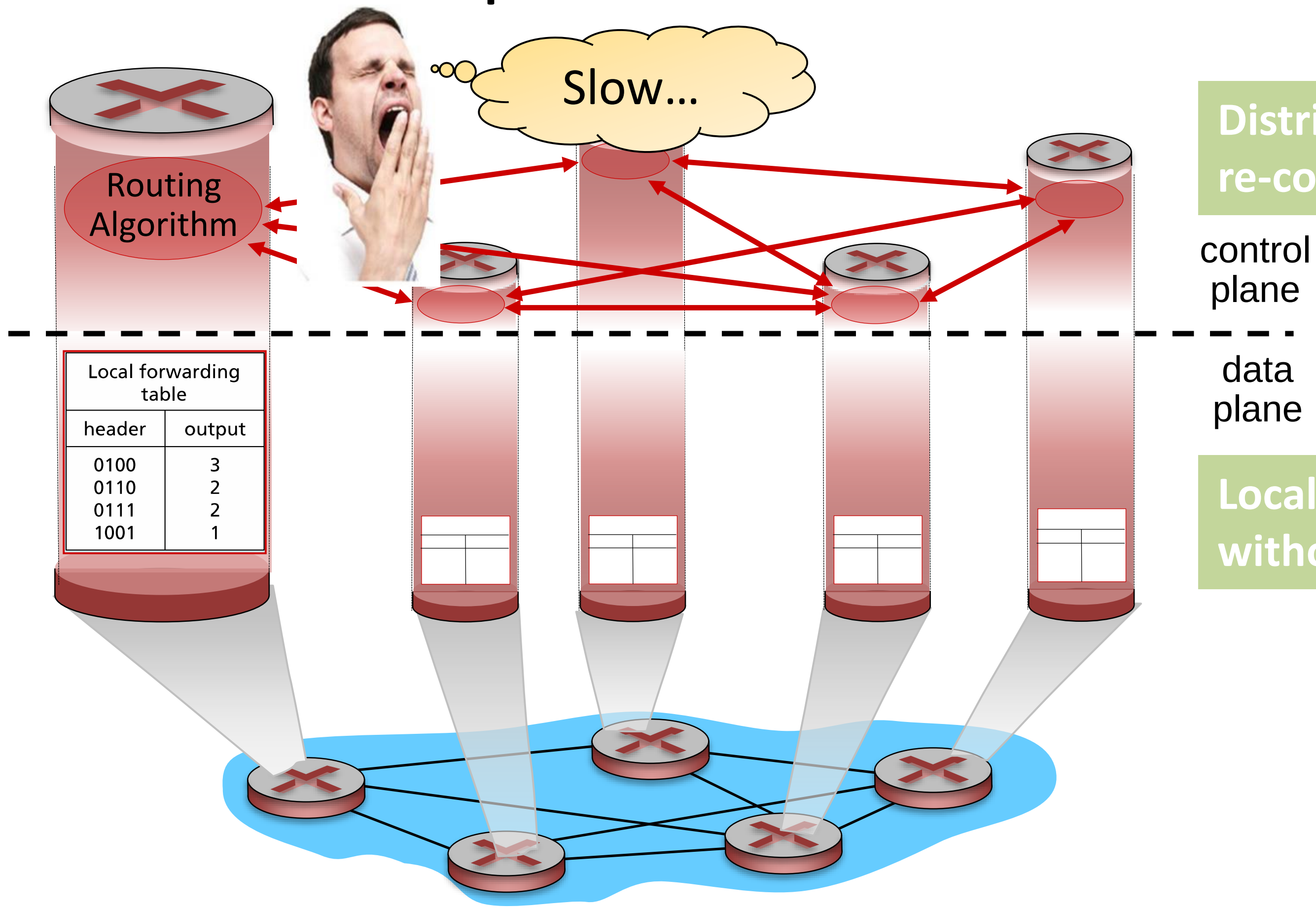
Two Options to React to Link Failures



Distributed algorithms, e.g., re-compute shortest paths

Local 'algorithms', typically without communication.

Two Options to React to Link Failures

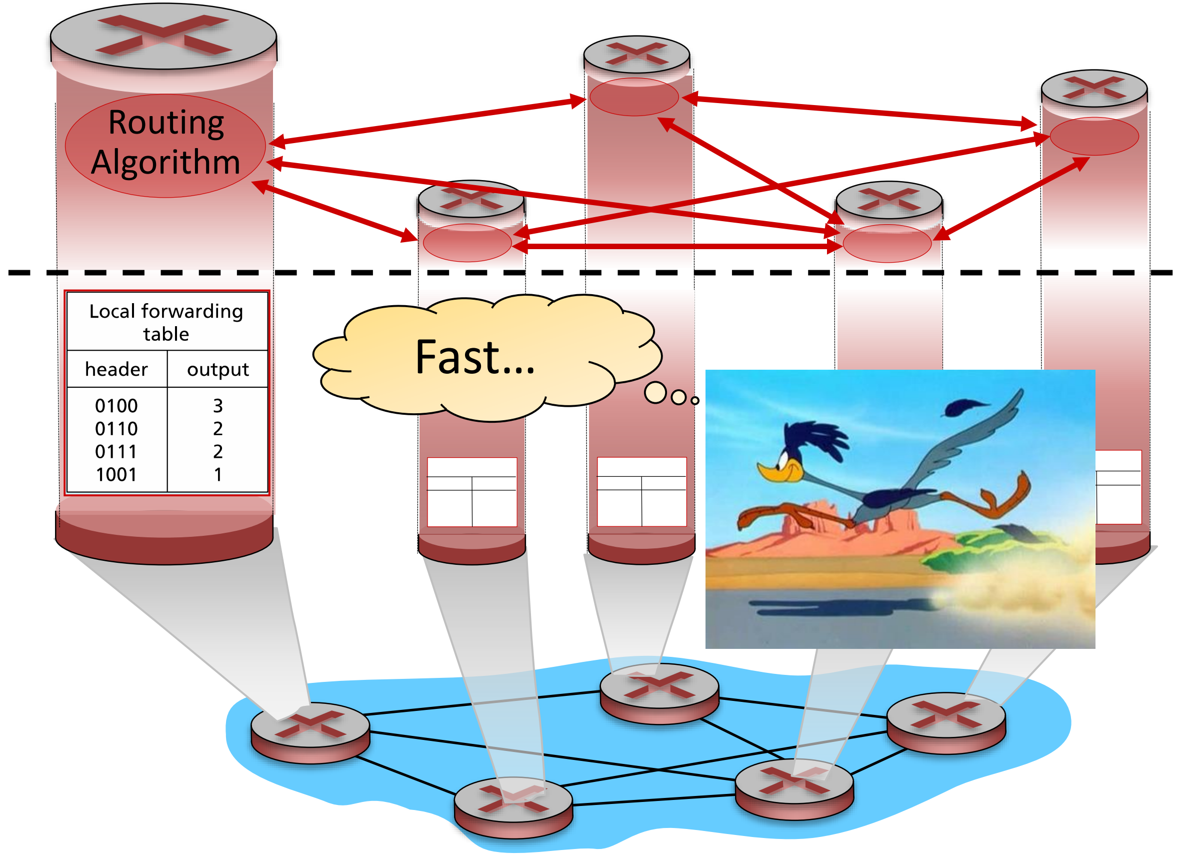


Distributed algorithms, e.g., re-compute shortest paths

Learn and re-converge!

Local 'algorithms', typically without communication.

Two Options to React to Link Failures



Distributed algorithms, e.g., re-compute shortest paths

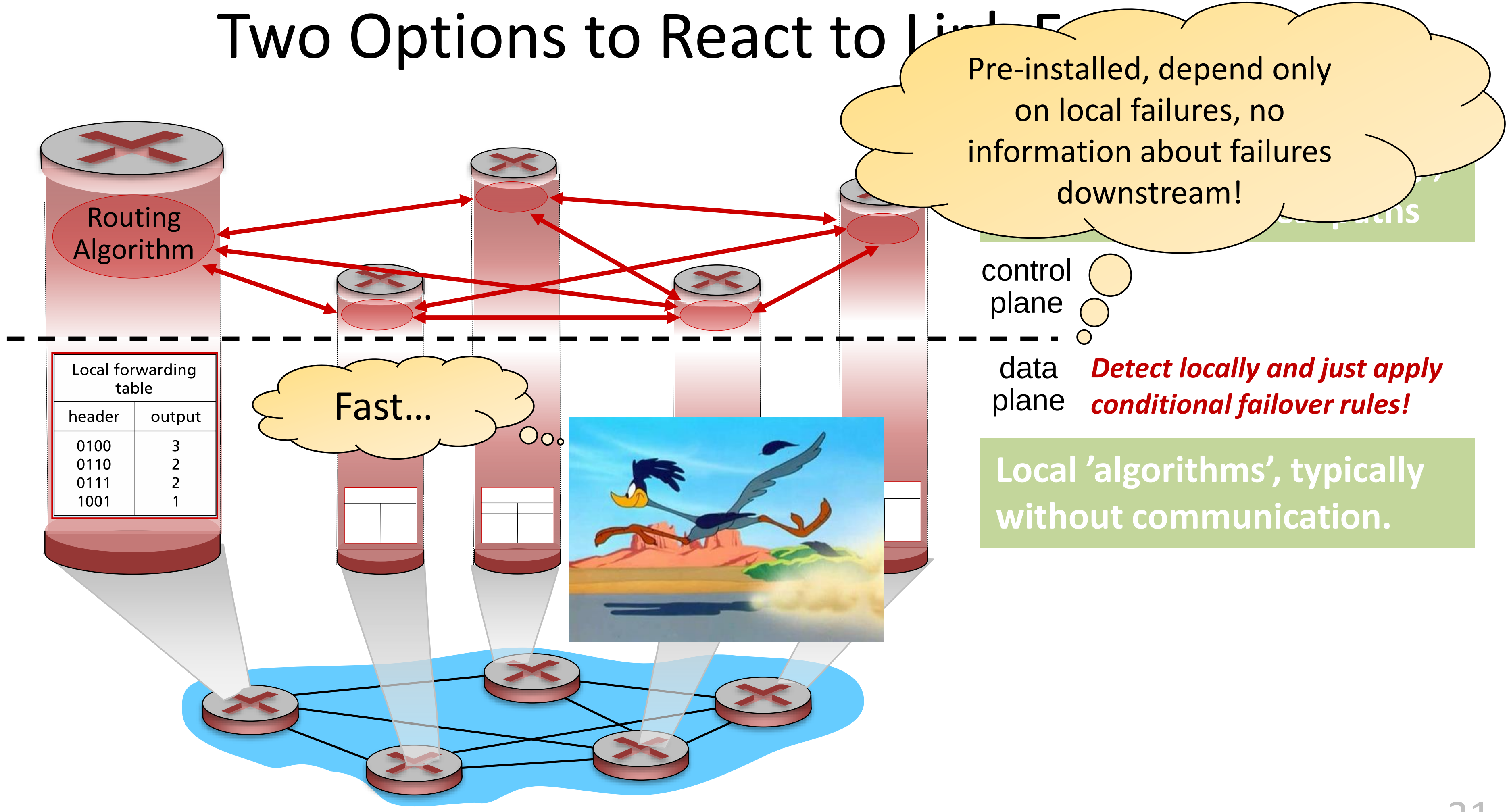
control plane

data plane

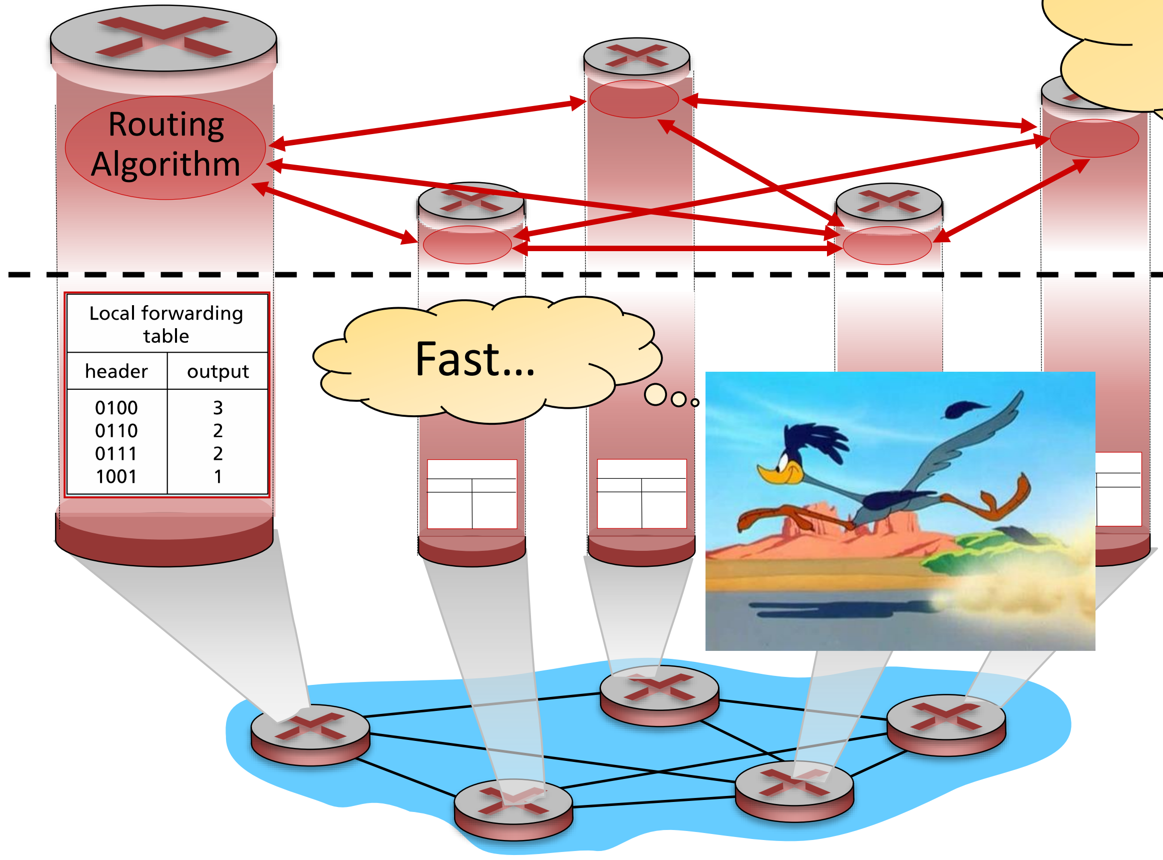
Detect locally and just apply conditional failover rules!

Local 'algorithms', typically without communication.

Two Options to React to Lin



Two Options to React to Li



Pre-installed, depend only on local failures, no information about failures downstream!

control plane

data plane

Detect locally and just apply conditional failover rules!

Local 'algorithms', typically without communication.

Algorithmic challenge: how to pre-install rules, so that later routes are connected even under multiple failures?

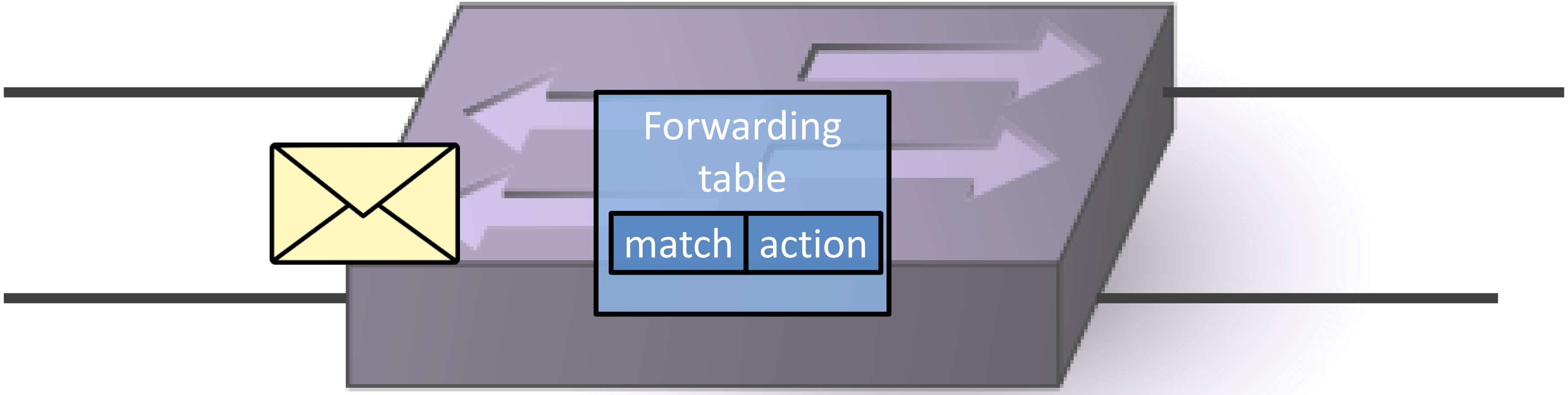
Why is slow bad? Packet drops until restored!

Video shot taken from “Lemmings”
designed and developed by DMA Design

How can a switch/router locally decide how to handle an arriving packet?

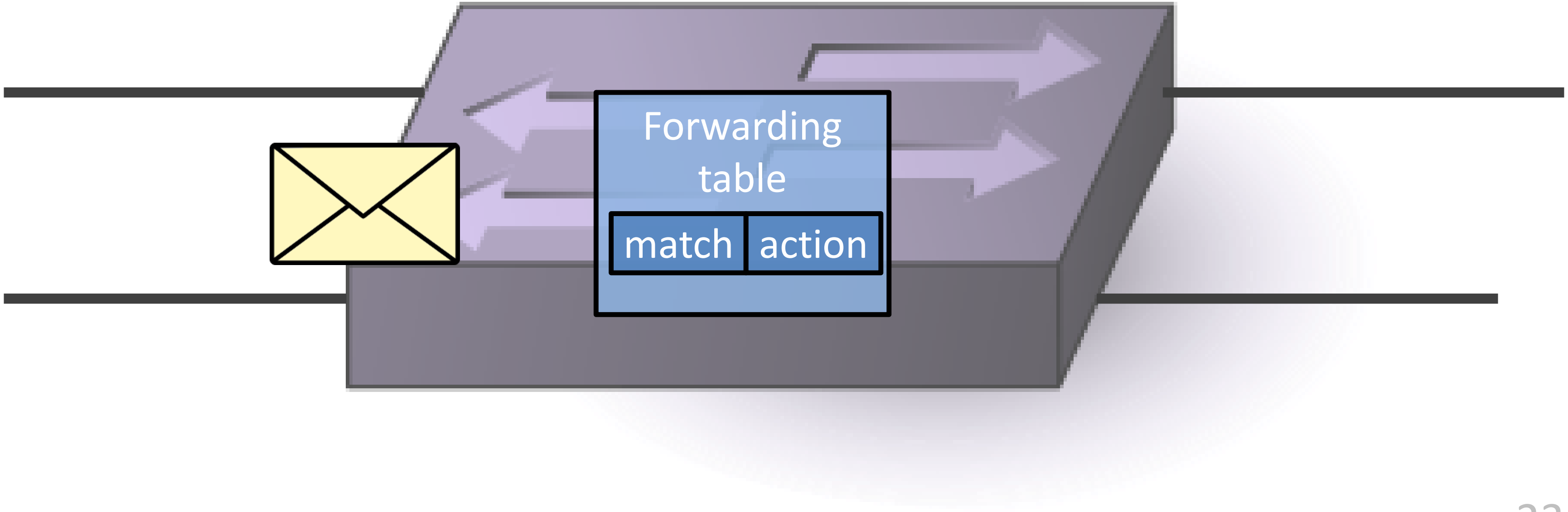
Credits: Marco Chiesa

Nodes Locally Store A Forwarding *Match -> Action Table*



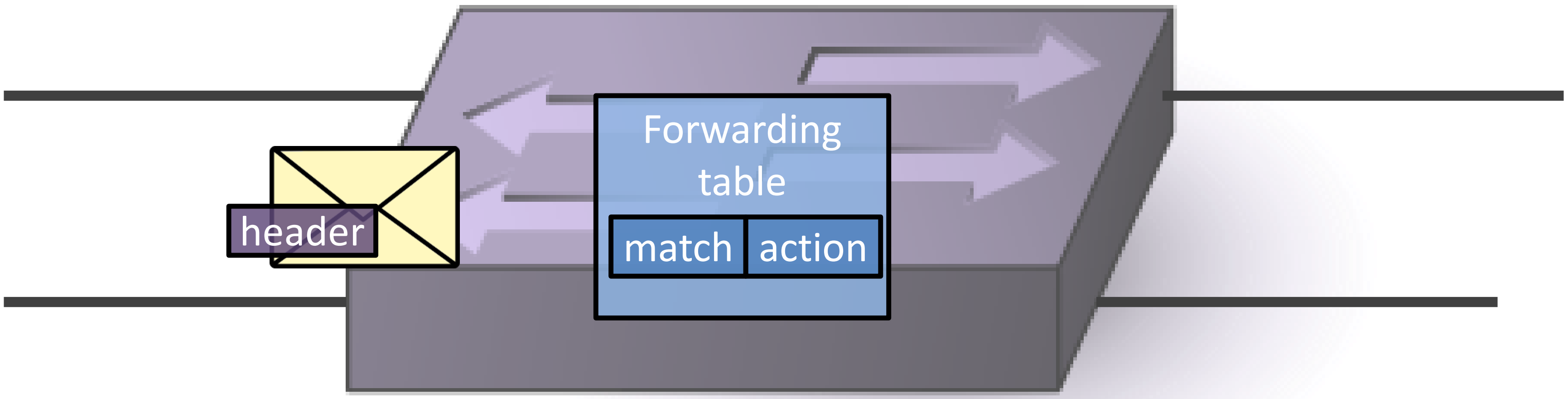
Credits: Marco Chiesa

And what information is locally available to decide how to handle an arriving packet?



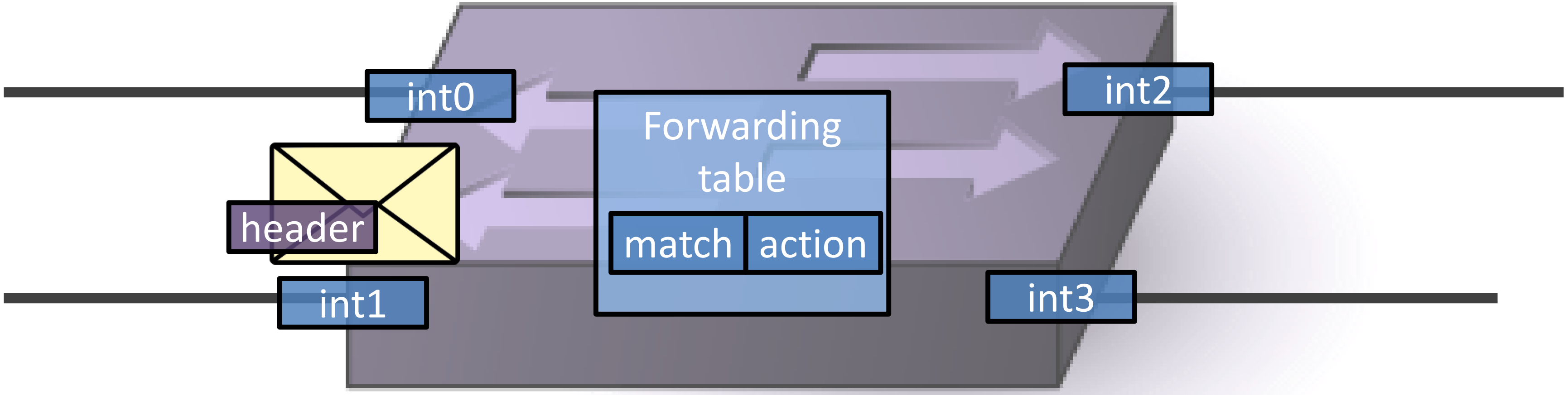
Credits: Marco Chiesa

Locally Available Information: The Packet Header (e.g., Source, *Destination*)



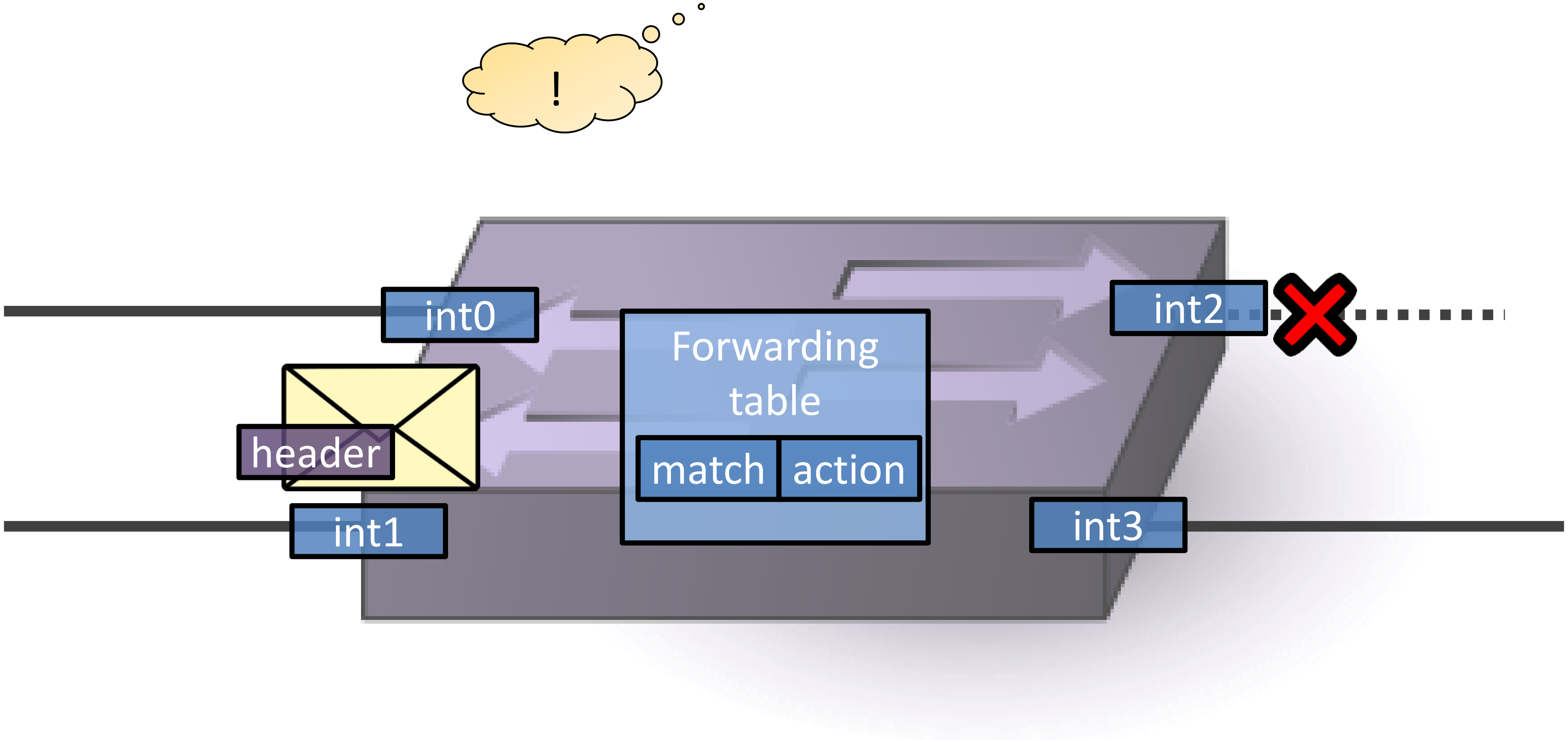
Credits: Marco Chiesa

Locally Available Information: The *Inport* of the Received Packet



Credits: Marco Chiesa

Locally Available Information: Which *Incident Failed Links*

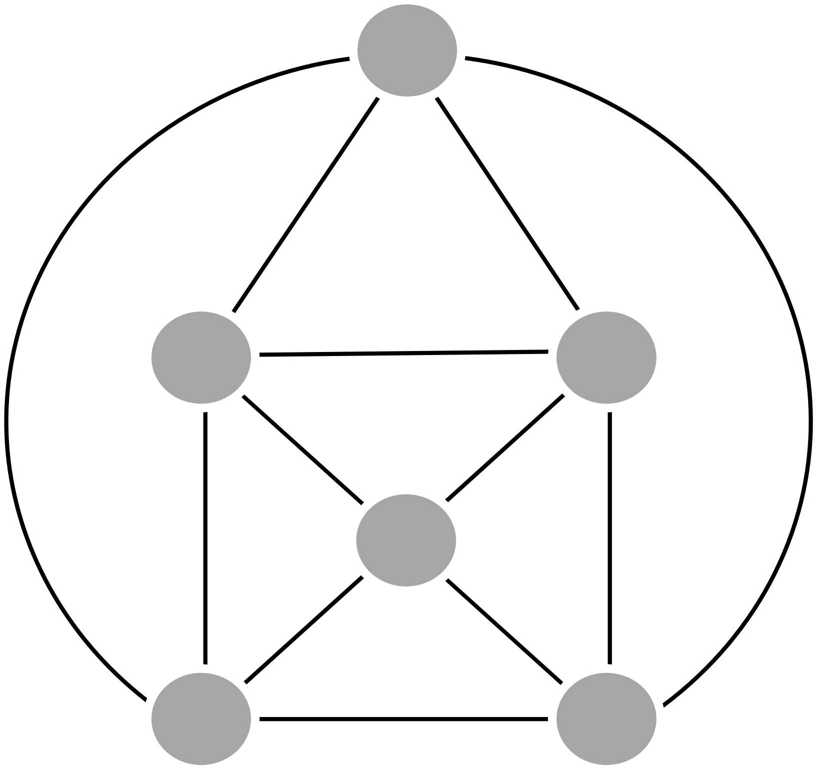


Credits: Marco Chiesa

Raises an Interesting Question

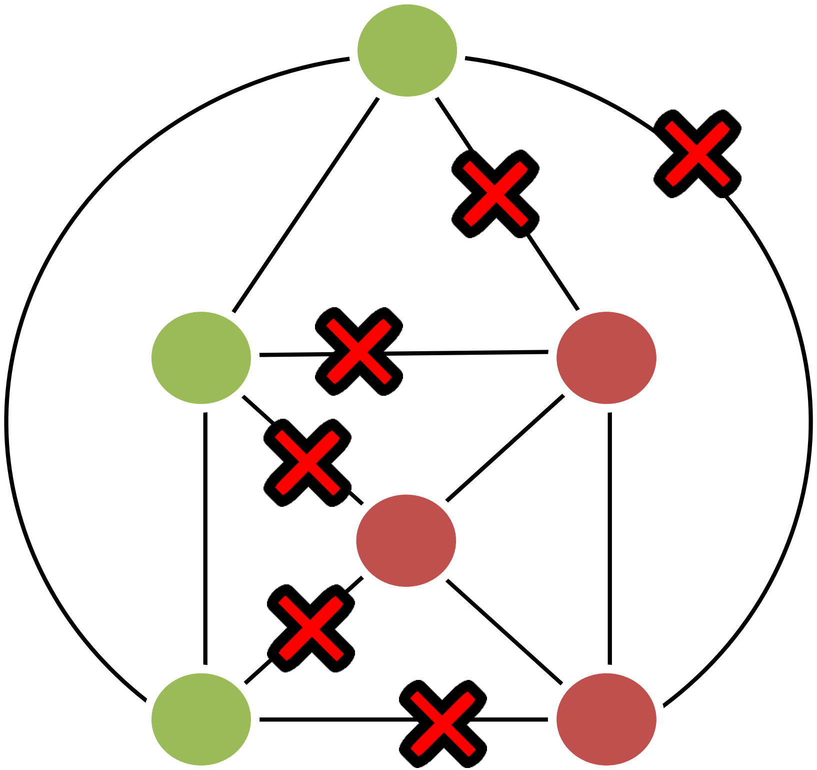
Can we *pre-install* local fast failover rules which ensure reachability under multiple failures? *In particular: How many failures* can be tolerated by static forwarding tables?

So: How many failures can be tolerated by static forwarding tables?



Credits: Marco Chiesa

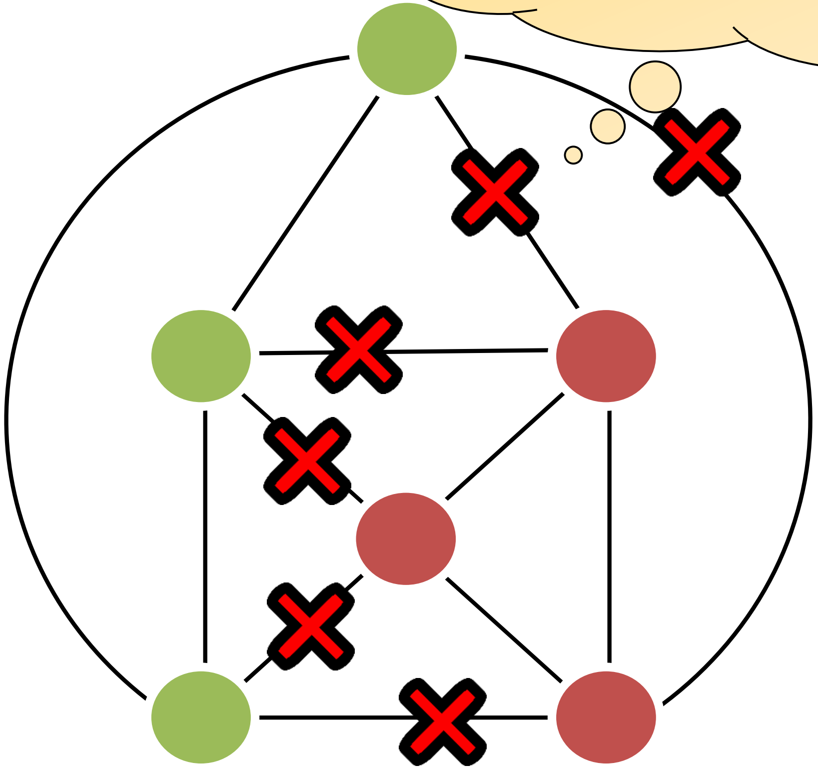
If we partition the network, there is not much to do



Credits: Marco Chiesa

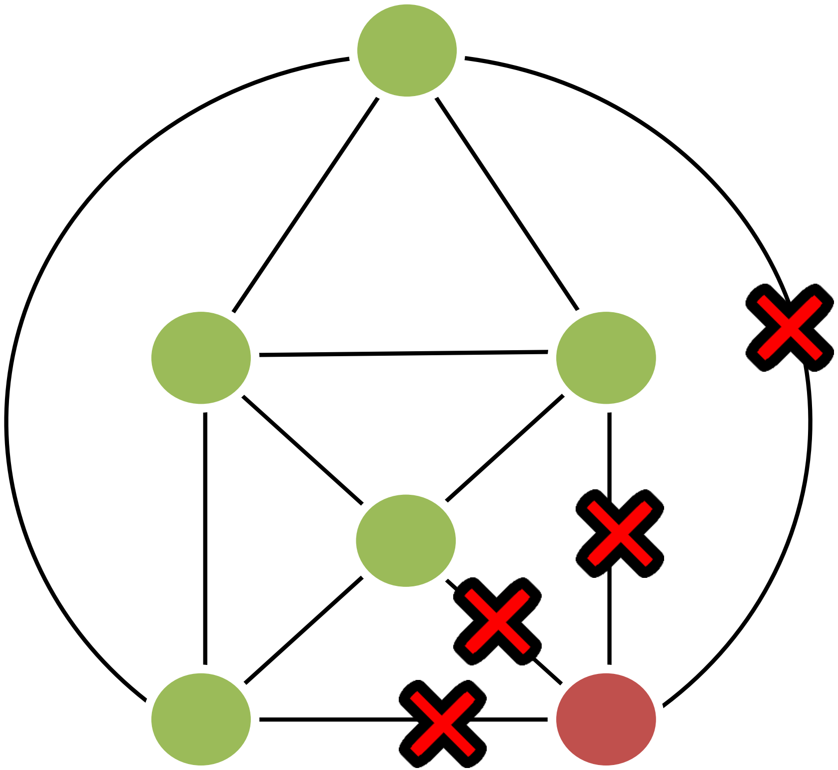
If we partition th…
there is not…

Clearly, topological connectivity is necessary. But also sufficient?



Credits: Marco Chiesa

Definition: *Connectivity k* of a network N: the minimum number of link deletions that partitions N



The connectivity of this network is *four*

Credits: Marco Chiesa

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

Perfect resilience

Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

Perfect resilience

Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

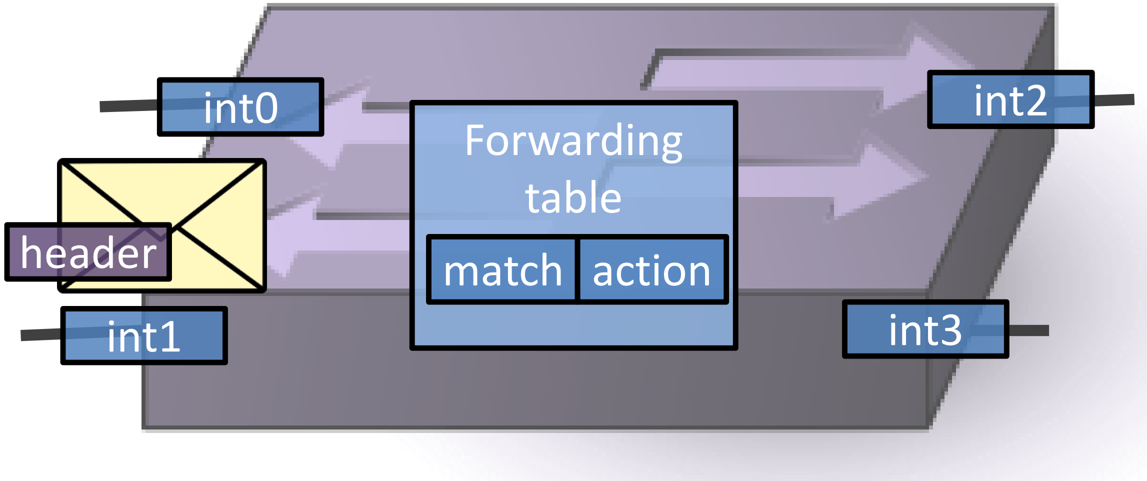
Perfect resilience

Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.

Spectrum of Models

Recall our switch model:



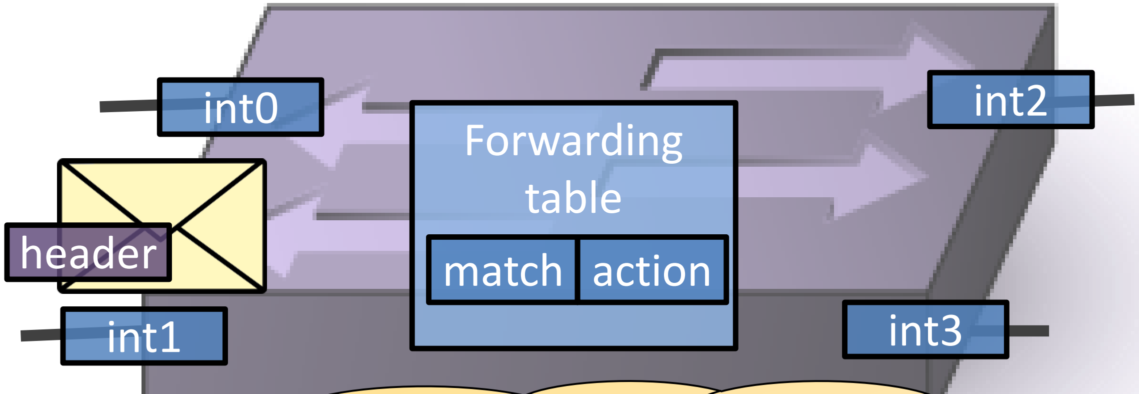
Achievable resilience depnds on *what can be matched*:

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting

Credits: Marco Chiesa

Spectrum of Models

Recall our switch model:



Can carry global information,
but often undesirable

Achievable resilience depnds on

Per-destination	**Per source**	**Incoming port**	**Probabilistic forwarding**	**Packet header rewriting**

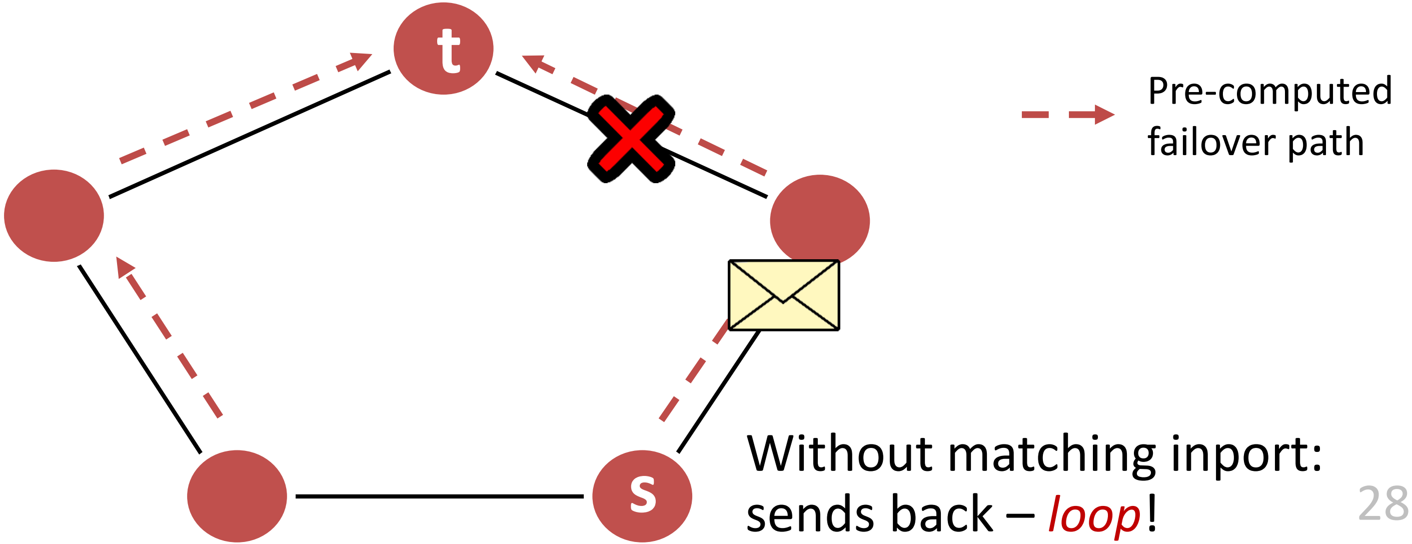
Credits: Marco Chiesa

Example: Which level of resiliency?

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X					

Per-destination routing *cannot cope* with *even one* link failure

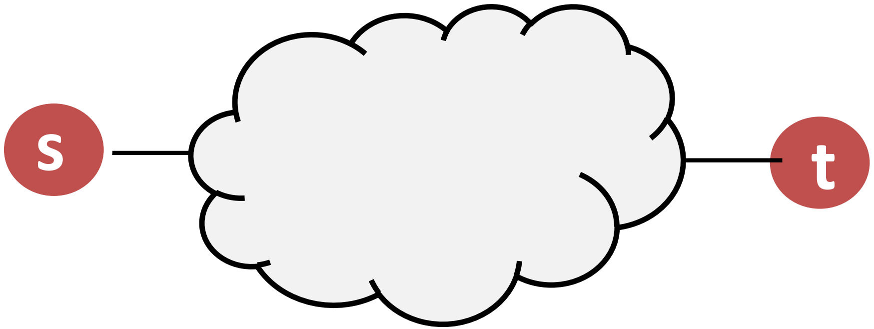
Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X					0



Without matching inport: sends back – *loop*!

Can we achieve k – 1 resiliency in k-connected graph here?

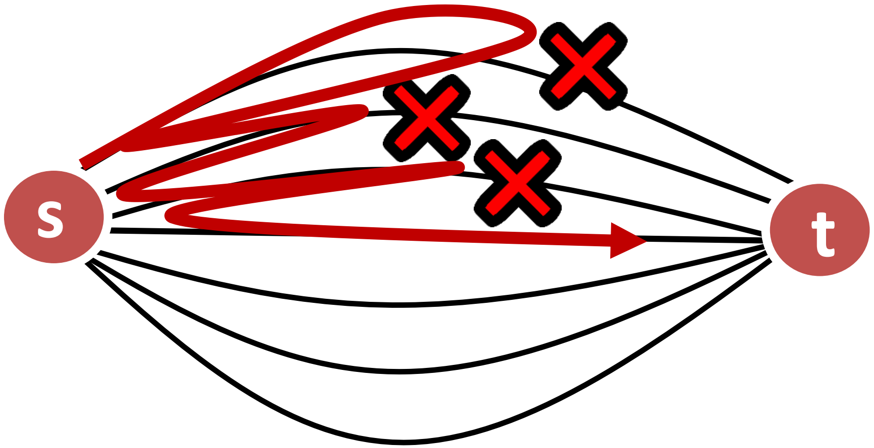
Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X	X	X			?



Credits: Marco Chiesa

Can we achieve k – 1 resiliency in k-connected graph here?

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X	X	X			Yes



k disjoint paths: try one after the other, routing *back to source* each time.

Credits: Marco Chiesa

Can we achieve k – 1 resiliency in k-connected graph here?

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X		X			?

What about this scenario?
Practically important. But open problem since many years…

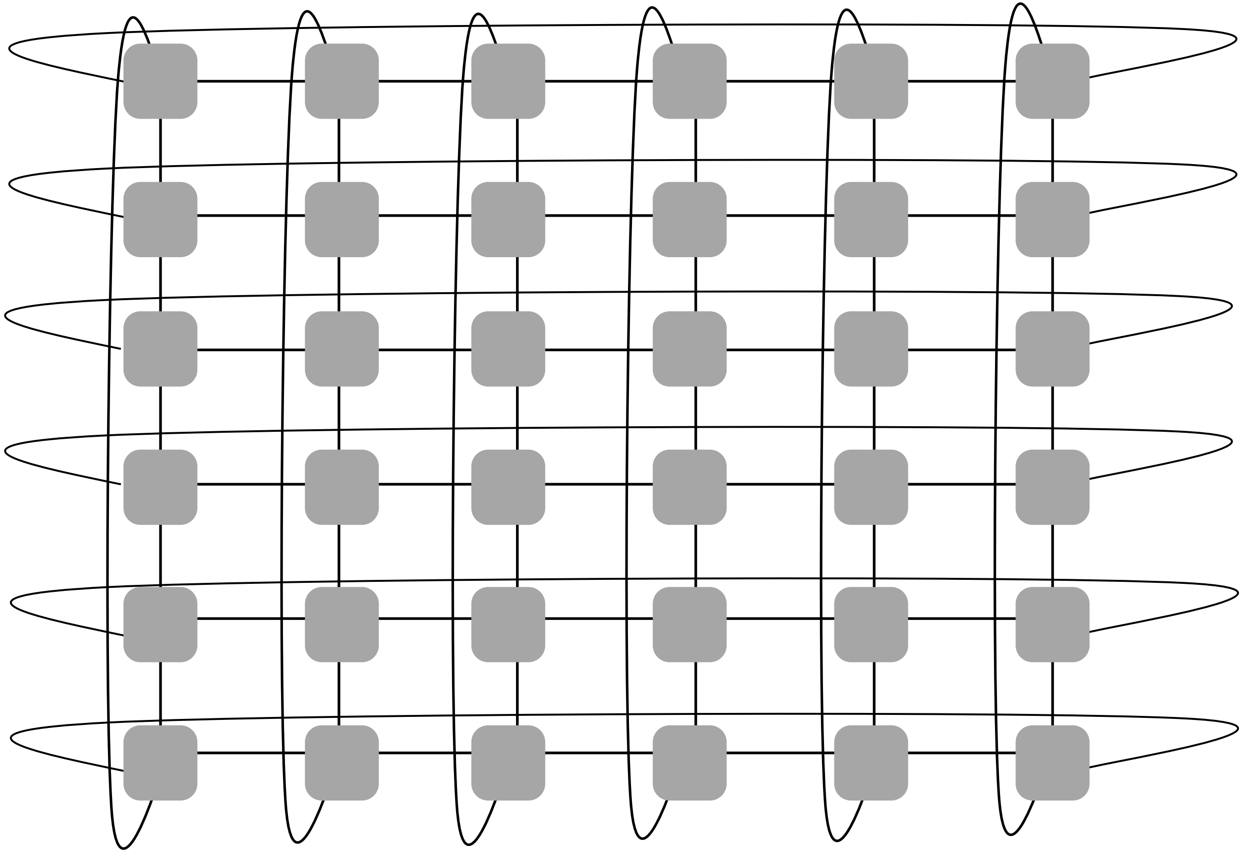
Can we achieve k – 1 resiliency in k-connected graph here?

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X		X			?

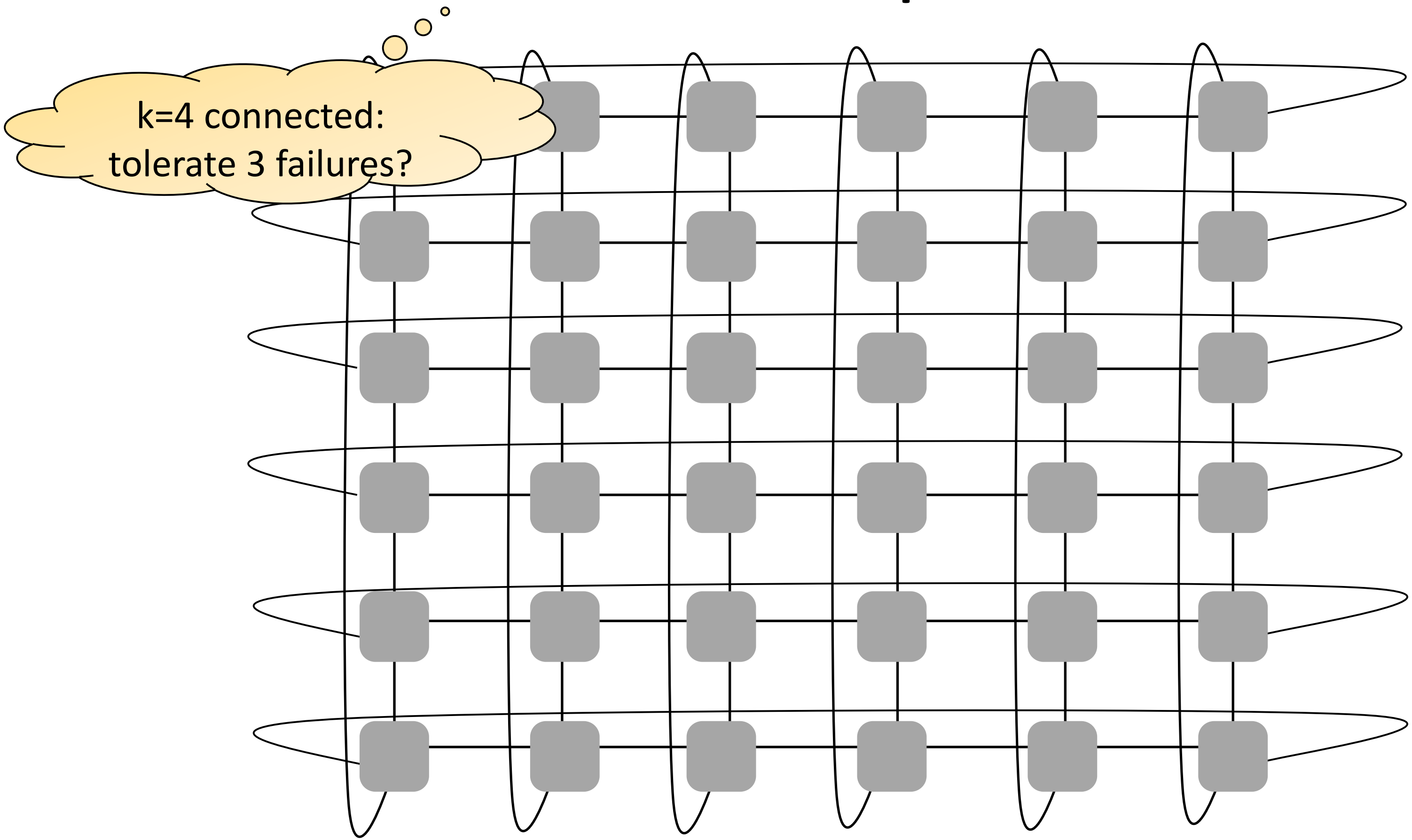
What about this scenario? Practically important. But open problem since many years…

For some special graphs we know: the answer is positive!

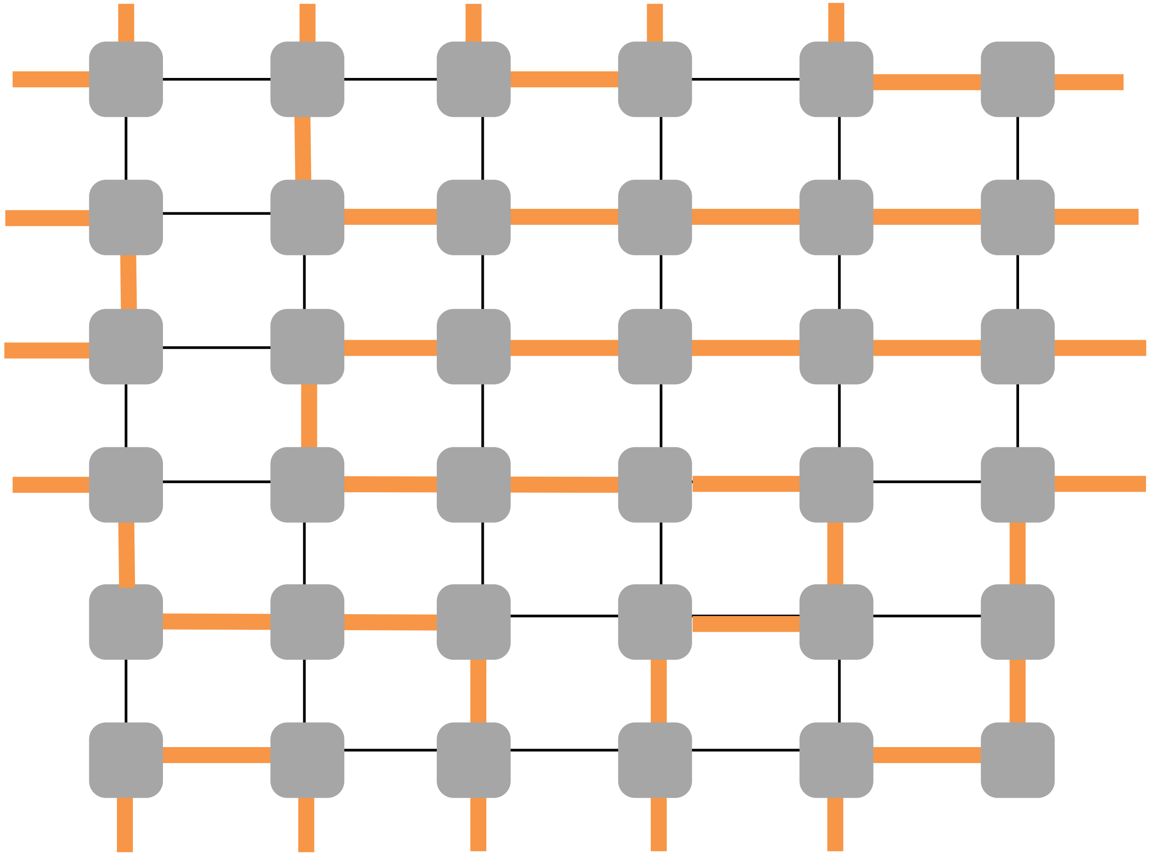
Ideal Resilience: Example 2-dim Torus?



Ideal Resilience: Example 2-dim Torus?



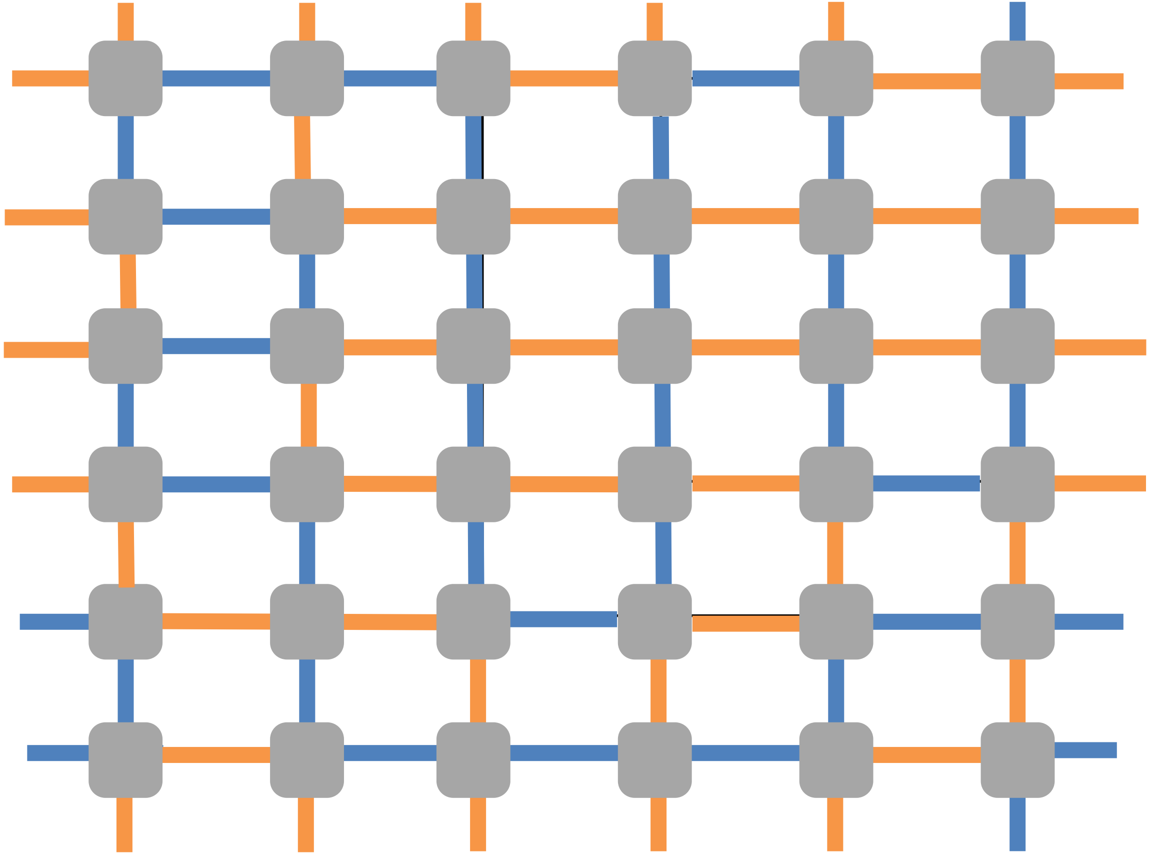
Idea: Decomposition into Hamilton Cycles



- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)

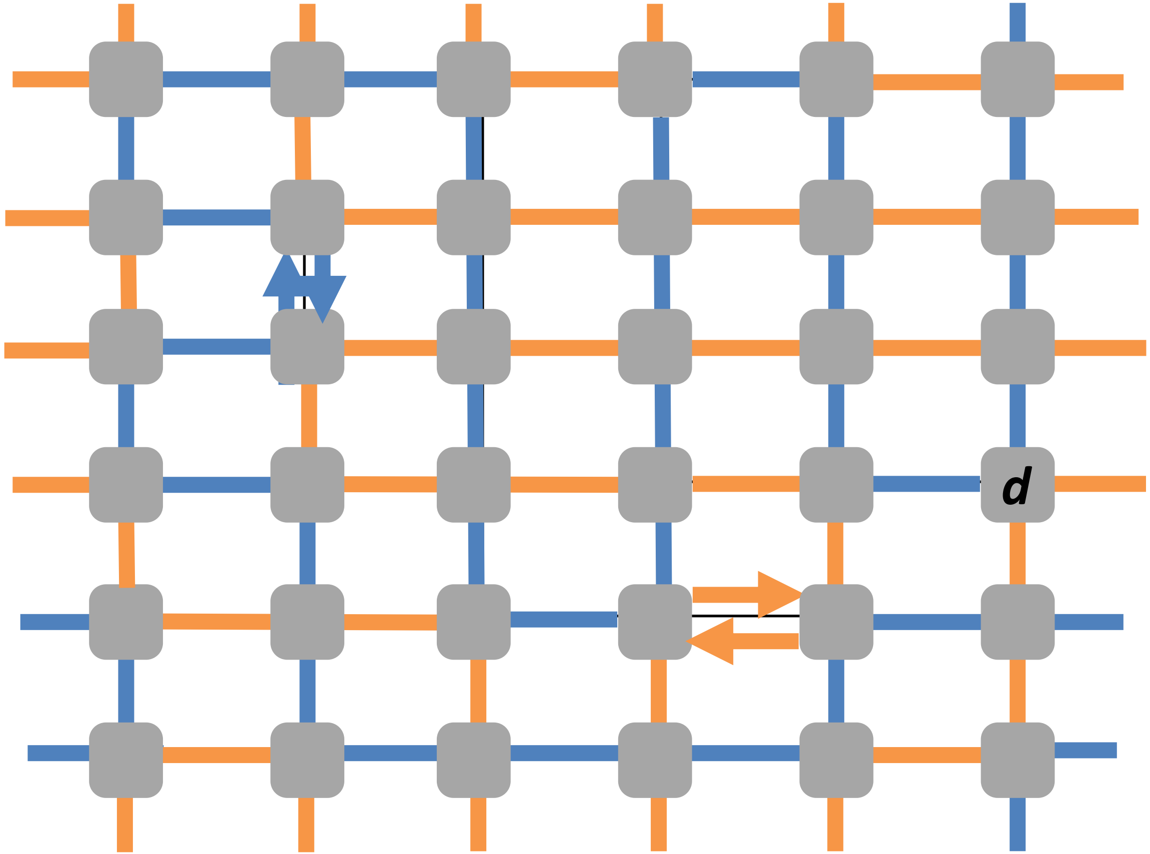
1st Hamilton cycle

Idea: Decomposition into Hamilton Cycles



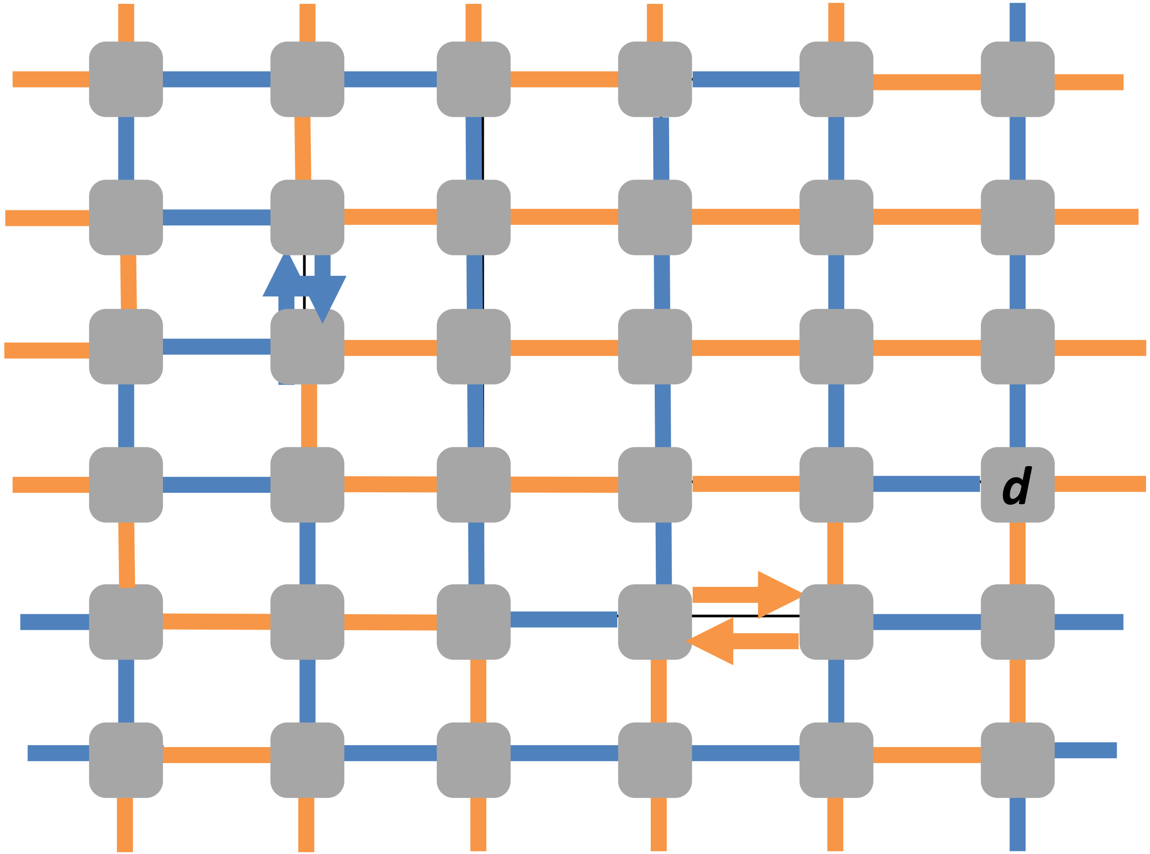
- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)

Idea: Decomposition into Hamilton Cycles



- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)
- Can route in both directions: *4-arc-disjoint* HCs

Idea: Decomposition into Hamilton Cycles



- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)
- Can route in both directions: *4-arc-disjoint* HCs

3-resilient routing to destination d:

- go along *1st directed HC*, if hit failure, *reverse* direction
- if again failure switch to *2nd HC*, if again failure *reverse direction*
- No more failures possible!

Ideal Resilience with Hamilton Cycles

Chiesa et al.: if k-connected graph has k arc disjoint Hamilton Cycles, k-1 resilient routing can be constructed!

Chiesa et al. **On the Resiliency of Static Forwarding Tables.** IEEE/ACM Transactions on Networking (ToN), 2017.

Ideal Resilience with Hamilton Cycles

Chiesa et al.: if k-connected graph has k arc disjoint Hamilton Cycles, k-1 resilient routing can be constructed!

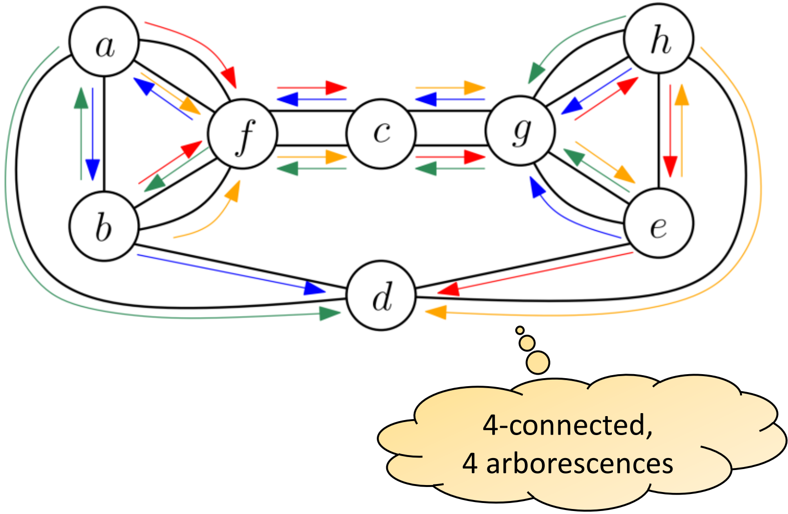
What about graphs which cannot be decomposed into Hamilton cycles?

Chiesa et al. **On the Resiliency of Static Forwarding Tables.** IEEE/ACM Transactions on Networking (ToN), 2017.

Ideal Resilience in General k-Connected Graphs

- Use directed trees (i.e. *arborescences*) instead of Hamilton cycles
  - *Arc-disjoint*, spanning, and *rooted* at destination

- Classic result: k-connectivity guarantees k-arborescence decomposition



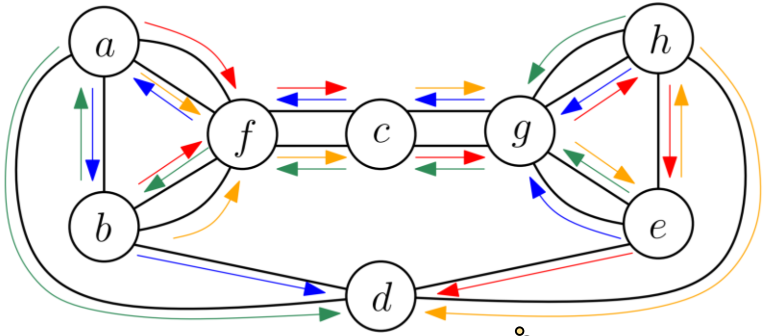
Basic idea:
- Idea: route towards root on one arborescence
- After failure: change arborescence (e.g. in circular fashion)
- Incoming port defines current arborescence
- After k-1 failures: At least one arborescence intact

J. Edmonds, **Edge-disjoint branchings**. Combinatorial Algorithms, 1972.

Ideal Resilience in General k-Connected Graphs

- Use directed trees (i.e. *arborescences*) instead of Hamilton cycles
  - *Arc-disjoint*, spanning, and *rooted* at destination

- Classic result: k-connectivity guarantees k-arborescence decomposition



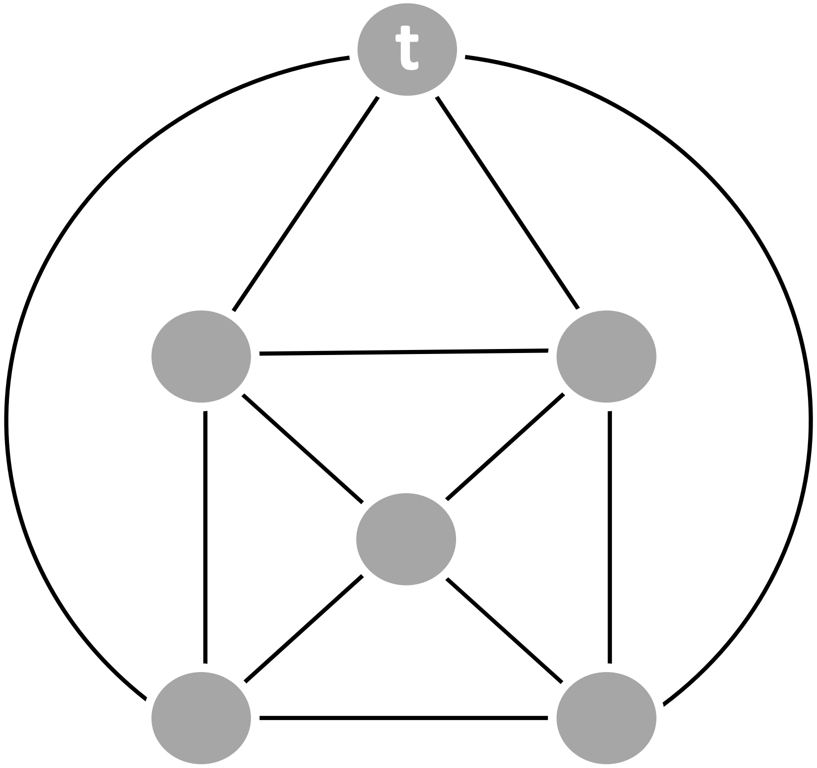
4-connected,
4 arborescences

Basic idea:
- Idea: route towards root on one arborescence
- After failure: change arborescence (e.g. in circular fashion)
- Incoming port defines current arborescence
- After k-1 failures: At least one arborescence intact

The challenge: how
to avoid earlier tree?

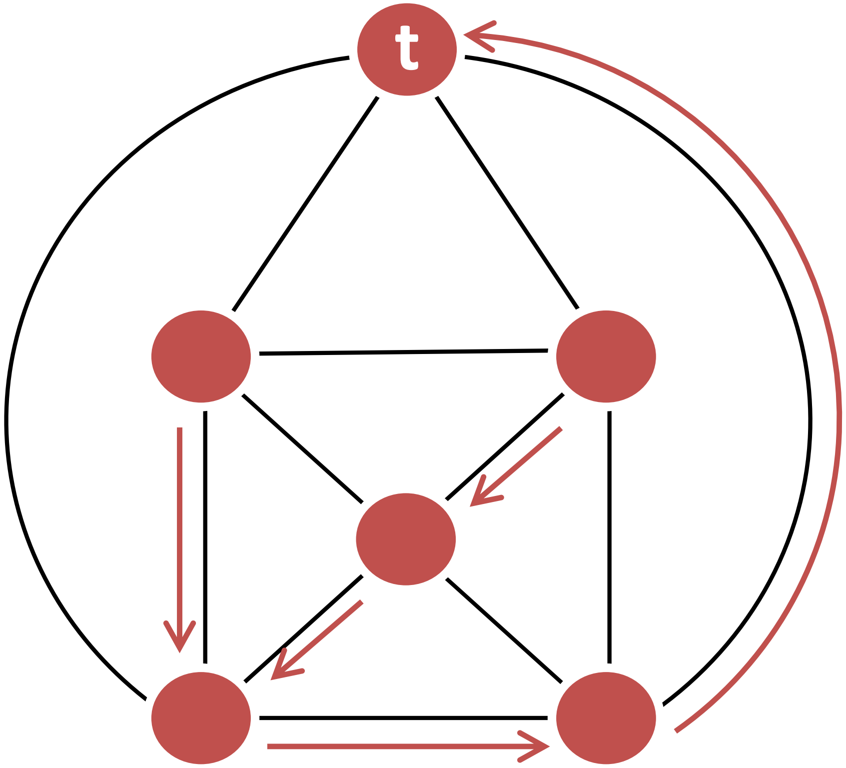
J. Edmonds, **Edge-disjoint branchings**. Combinatorial Algorithms, 1972.

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



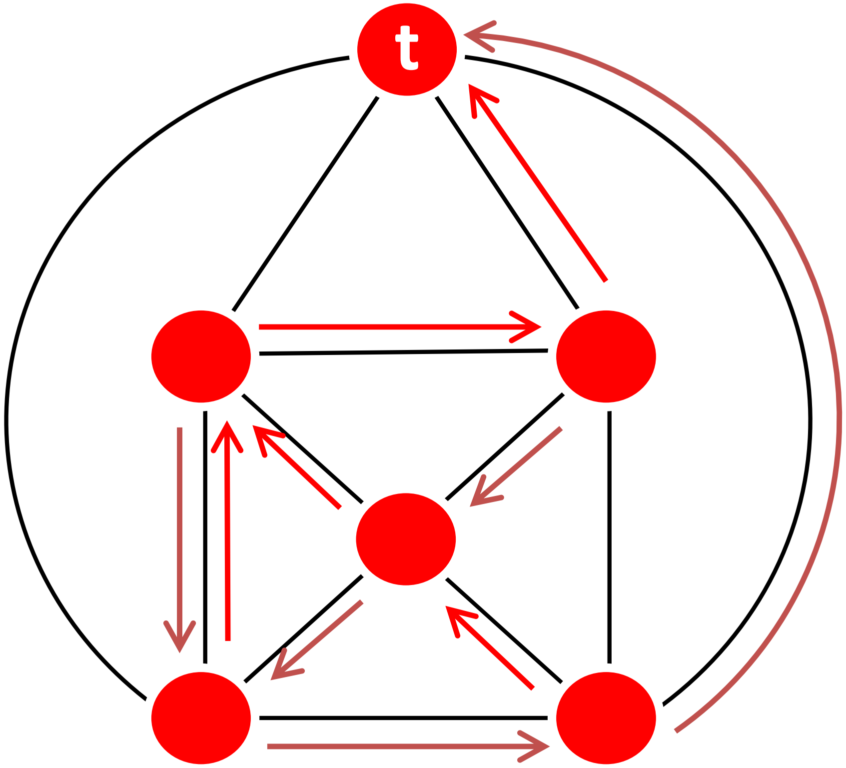
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



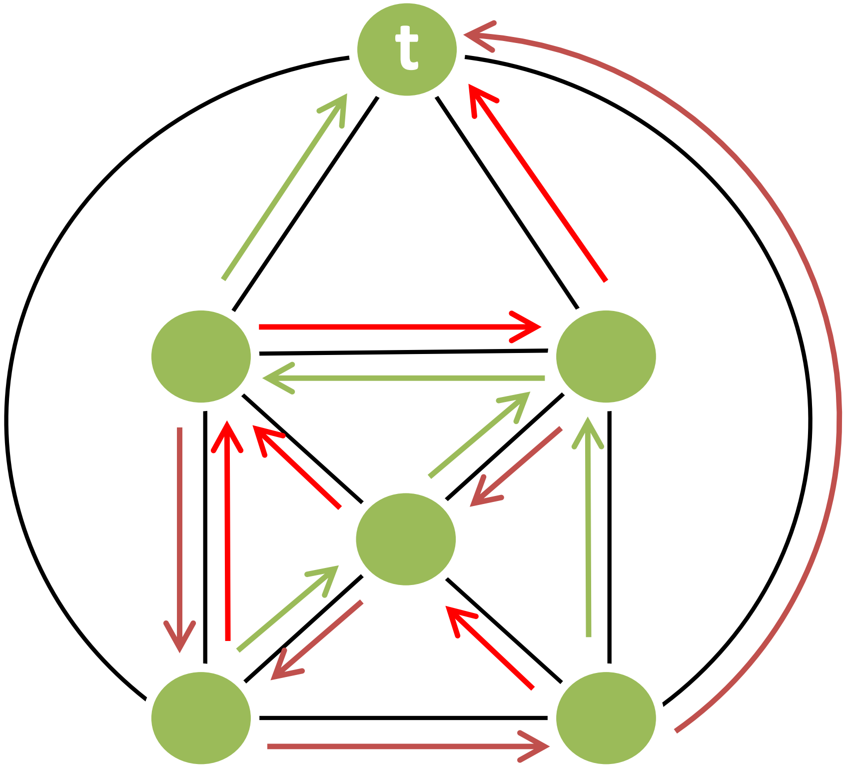
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



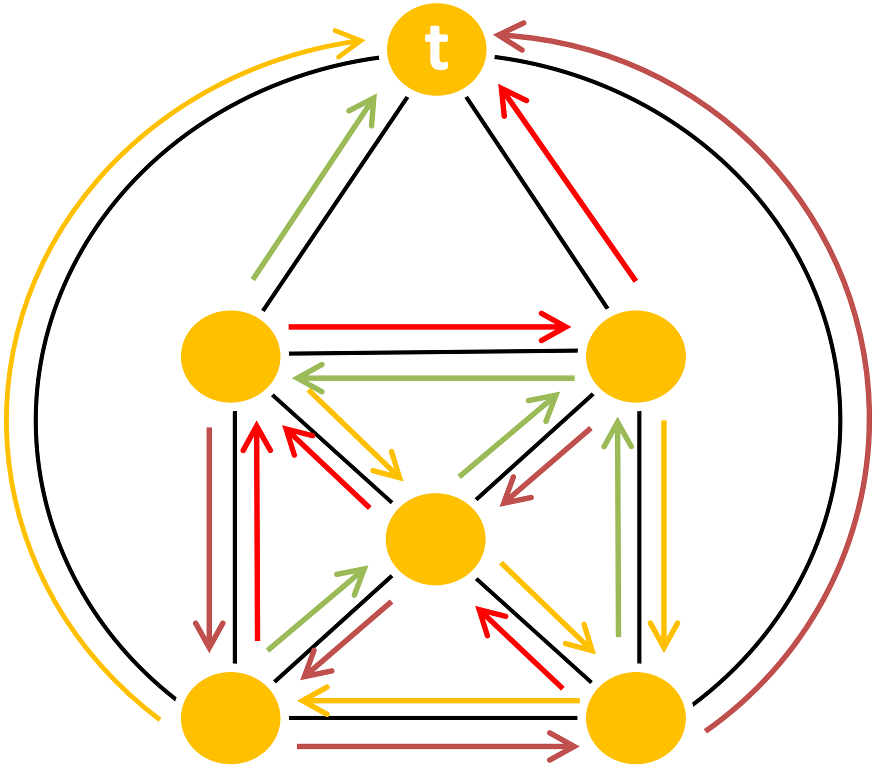
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



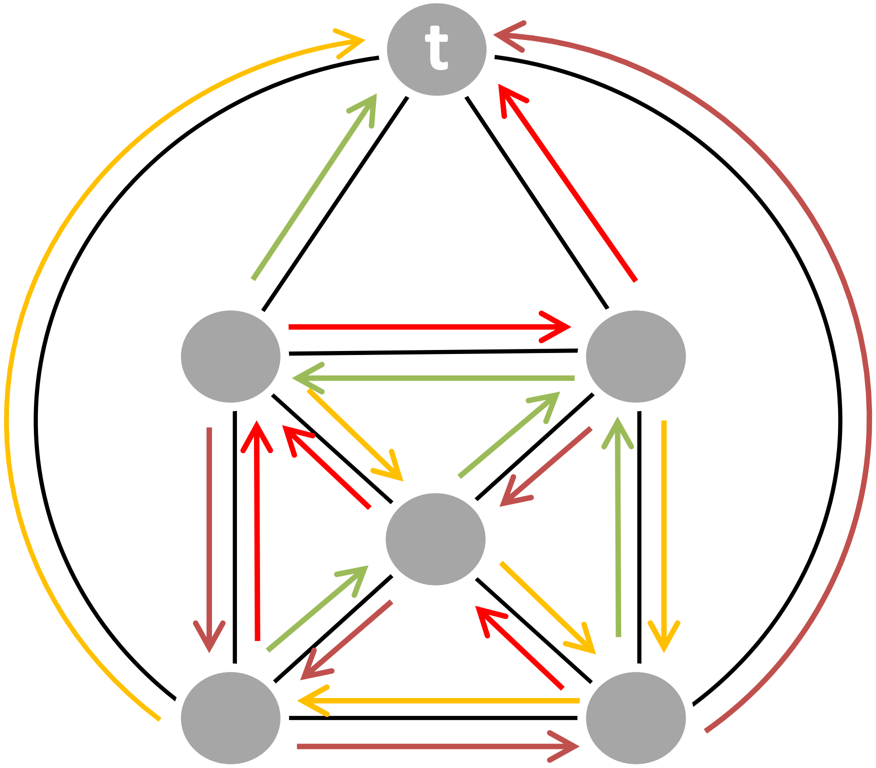
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



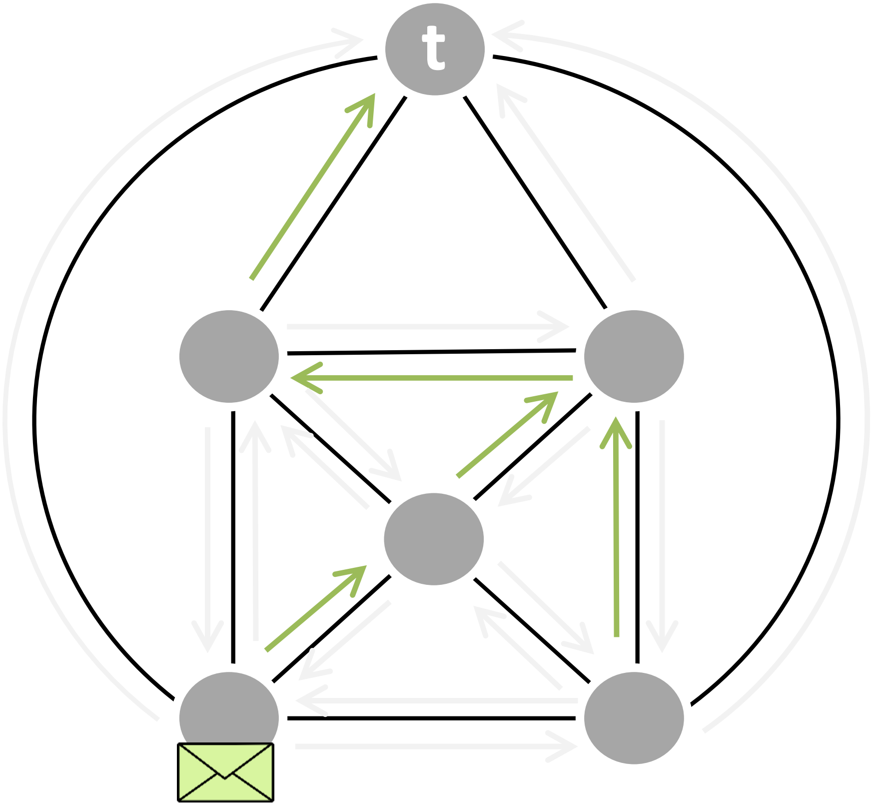
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



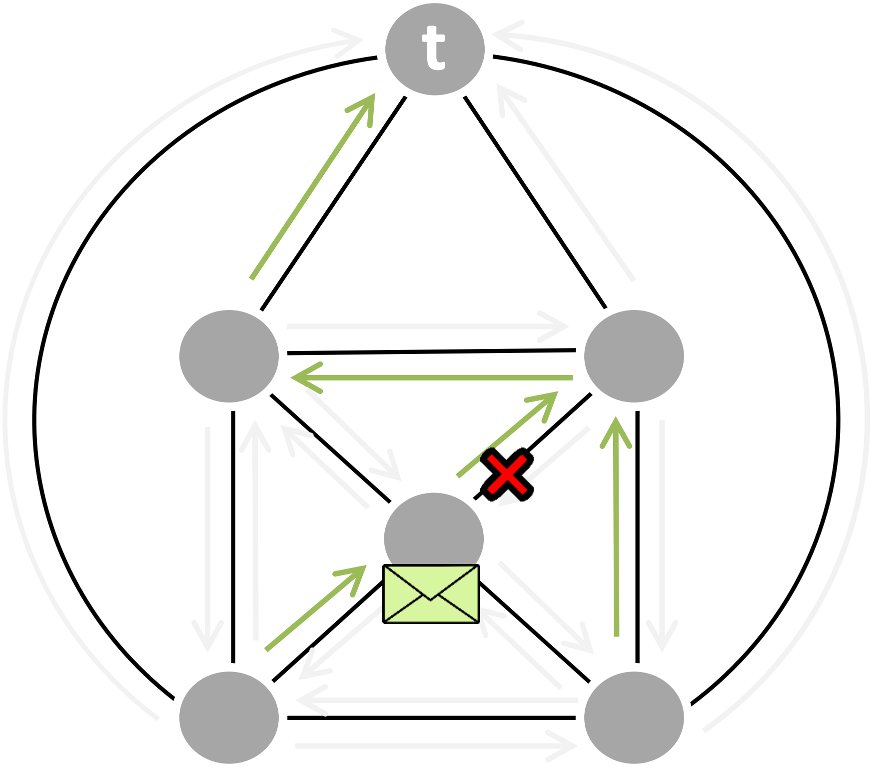
Credits: Marco Chiesa

General technique: routing along the same tree



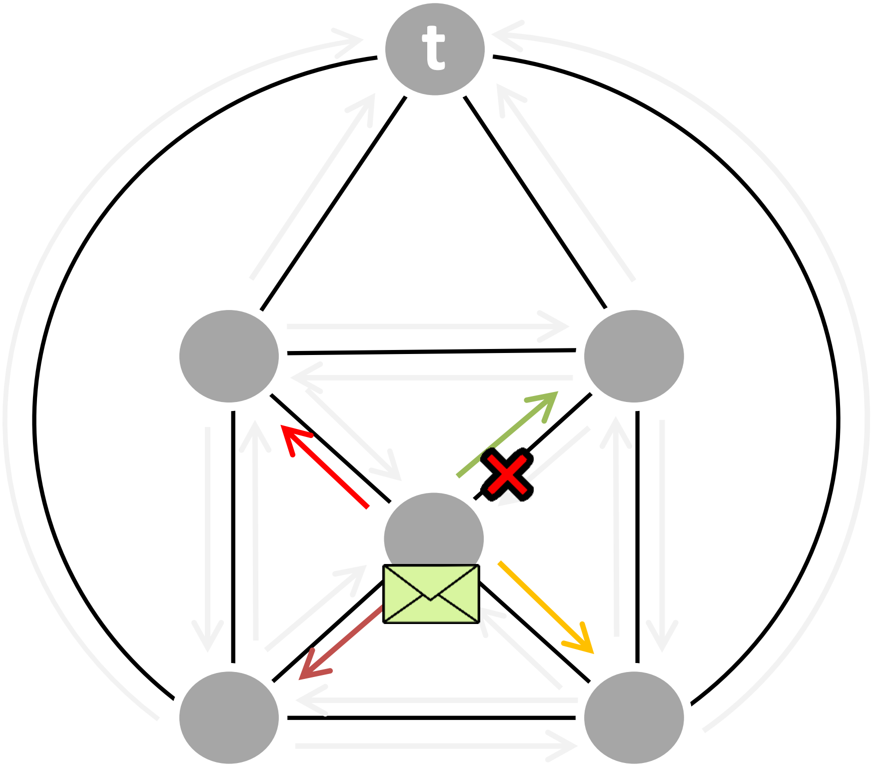
Credits: Marco Chiesa

When a failed link is hit…



Credits: Marco Chiesa

… how do we choose the next arborescence?



Credits: Marco Chiesa

But how do we choose the next arborescence?

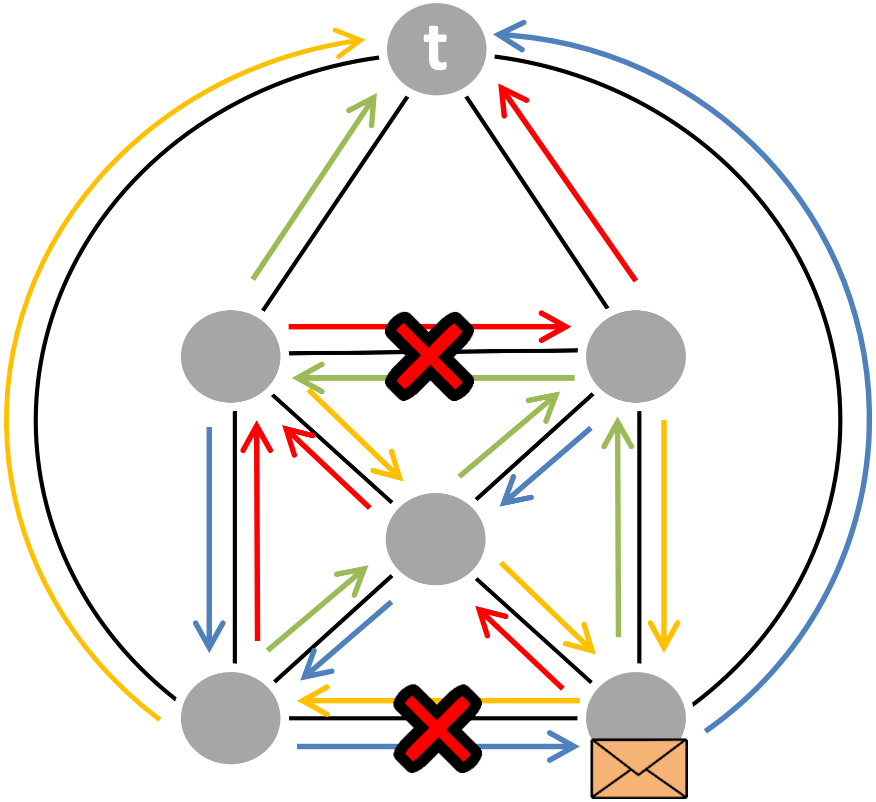
Circular-arborescence routing:

- compute an order of the arborescences
- switch to the next arborescence when hitting a failed link

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

1	2	3	4



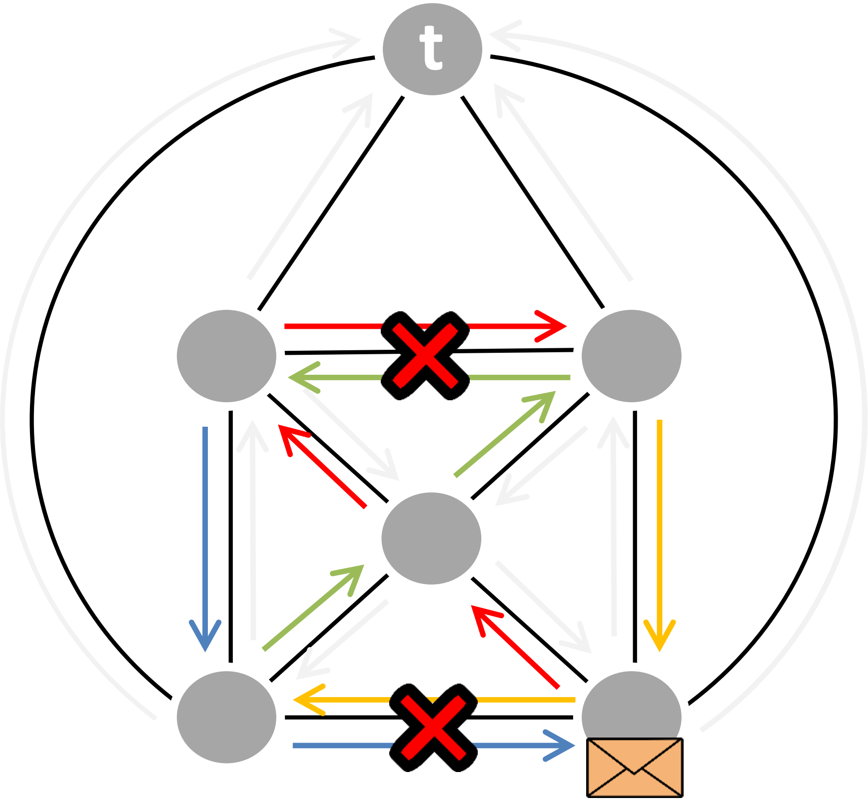
Intuition: each single failure may affect two arborescences

Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

Go along arborescence 1 to destination...



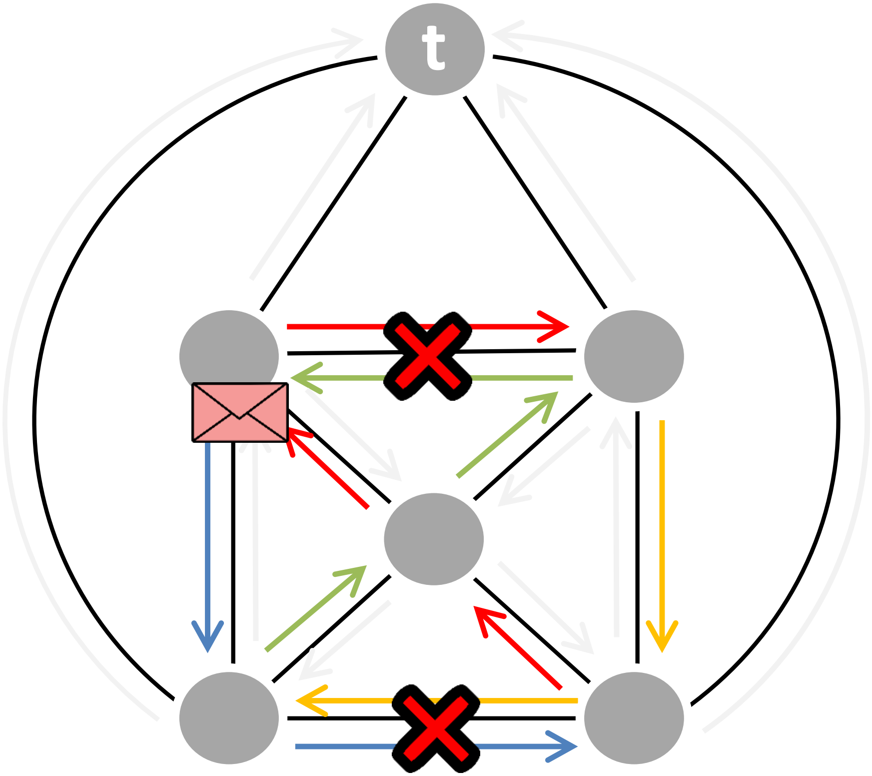
Intuition: each single failure may affect two arborescences

Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

Go along arborescence 2 to destination...



Intuition: each single failure may affect two arborescences

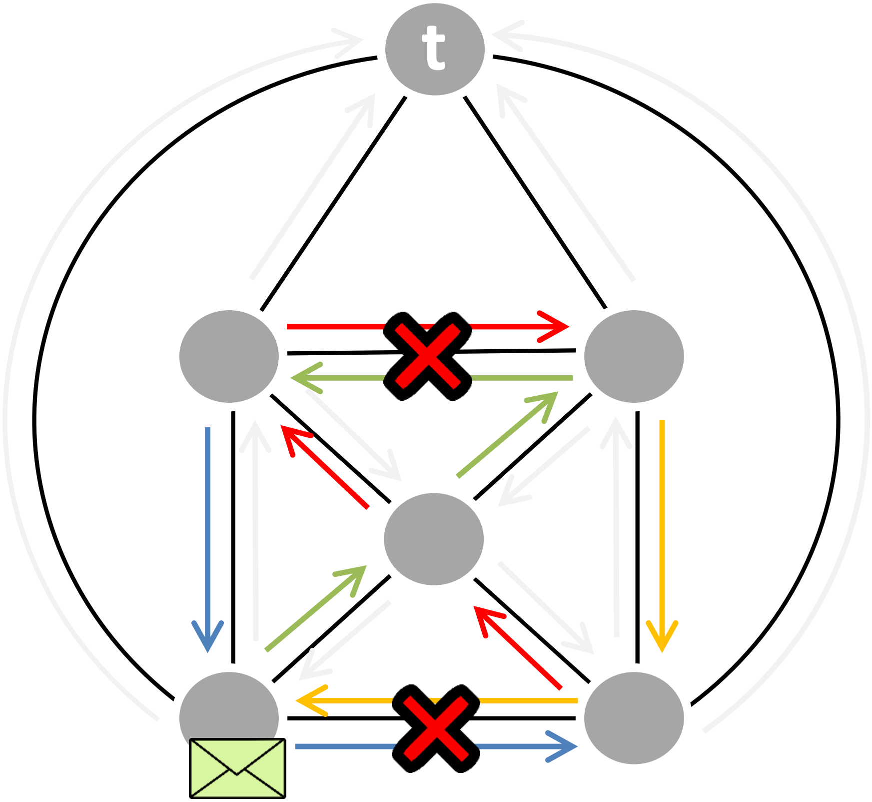
Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

1	2	3	4

Go along arborescence 3 to destination...



Intuition: each single failure may affect two arborescences

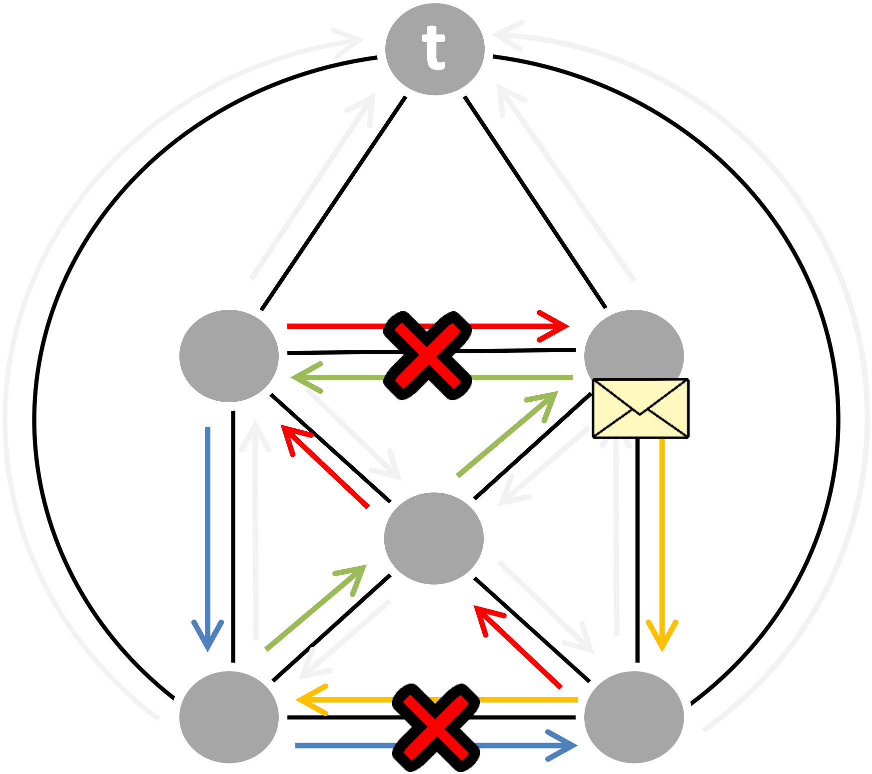
Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

1	2	3	4

Go along arborescence 4 to destination...

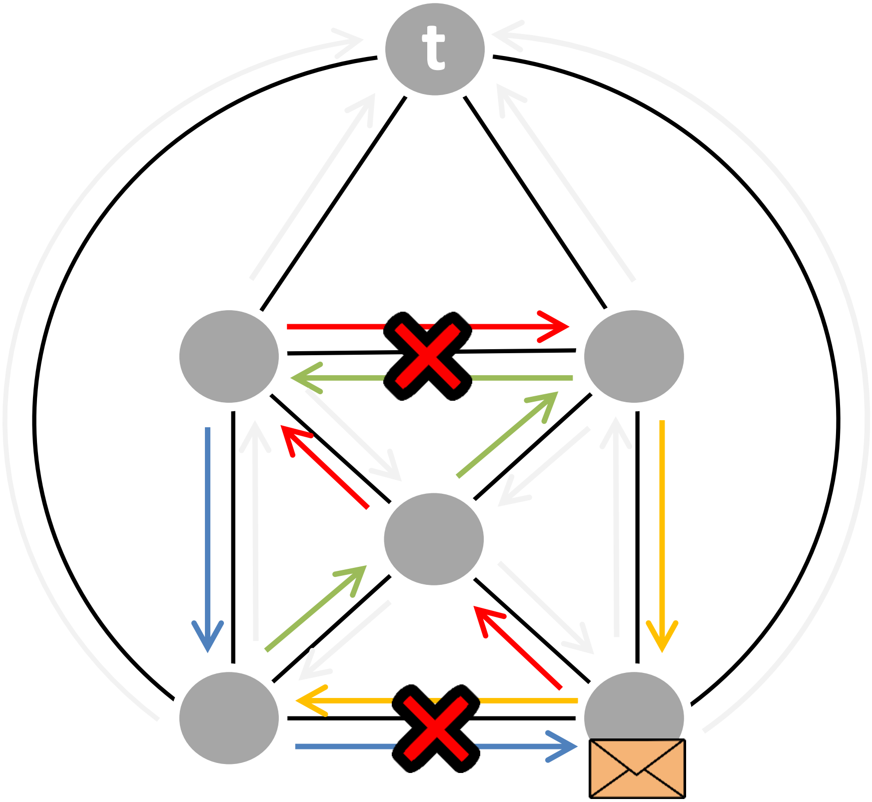


Intuition: each single failure may affect two arborescences

Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order



Intuition: each single failure may affect two arborescences

All k=4 arborescences used (2 failures disconnected affected all four): LOOP!

Credits: Marco Chiesa

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

Perfect resilience

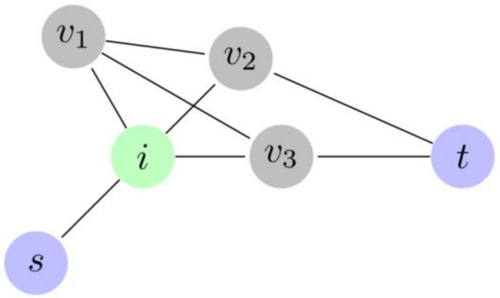
Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.

Resilience Criteria

Perfect resilience is impossible to achieve in general.

Already on simple planar graphs, proof by case distinction (and indistinguishability).



Related to several DISC problems but with twist!

- **Geometric routing**
  - E.g., a left-hand rule can be used in planar graphs
- **Local algorithms without communication**
  - E.g., Balanced Incomplete Block Design (*BIBD*) can be used to minimize congestion!
- **Graph exploration and connectivity problems**
  - E.g., Omer Reingold’s “undirected connectivity in log-space”

Many Open Questions…

- Big open question: **ideal resilience conjecture**
  - False? DISC experts!
- What if we can *rewrite* some header bits?
  - With log(n) bits it is easy: can remember all failures. What about less?
- What about fast rerouting in **Segment Routing** networks?
- What about *special graph classes*?
- Automated **synthesis** of tables (e.g., BDDs)

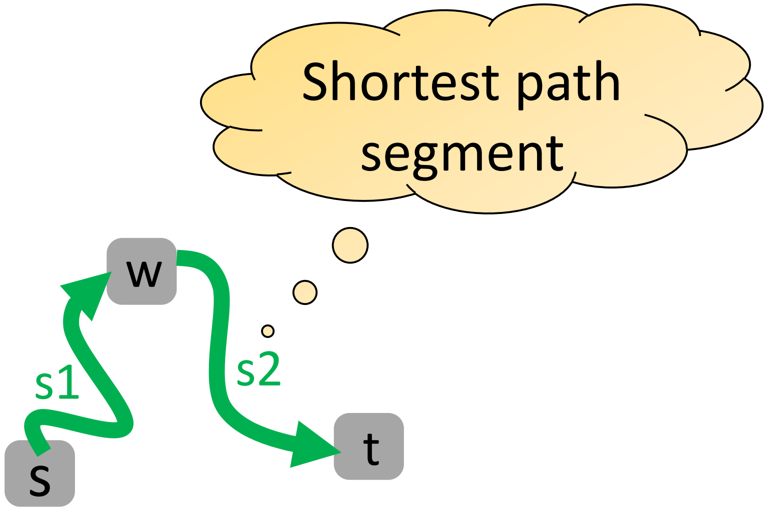
Many Open Questions…

- Big open question: **ideal resilience conjecture**
  - False? DISC experts!
- What if we can *rewrite* some header bits?
  - With log(n) bits it is easy: can remember all failures. What about less?
- What about fast rerouting in **Segment Routing** networks?
- What about *special graph classes*?
- Automated **synthesis** of tables (e.g., BDDs)

Literature:
On the Feasibility of Perfect Resilience with Local Fast Failover. Foerster et al., SIAM APOCS, 2021.
Randomized Local Fast Rerouting for Datacenter Networks with Almost Optimal Congestion. Bankhamer et al. DISC, 2021.

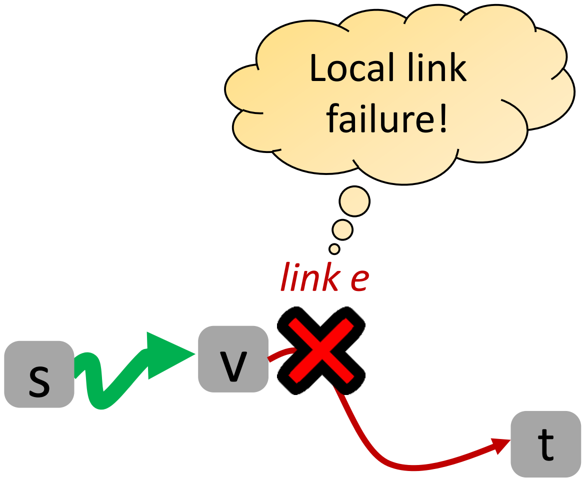
Local Reroute with Segment Routing?

- Recall segment routing: shortest path routing on segments
- Fast rerouting currently under standardization at IETF
  - Good time to have impact!



How to handle at least 1 failure?

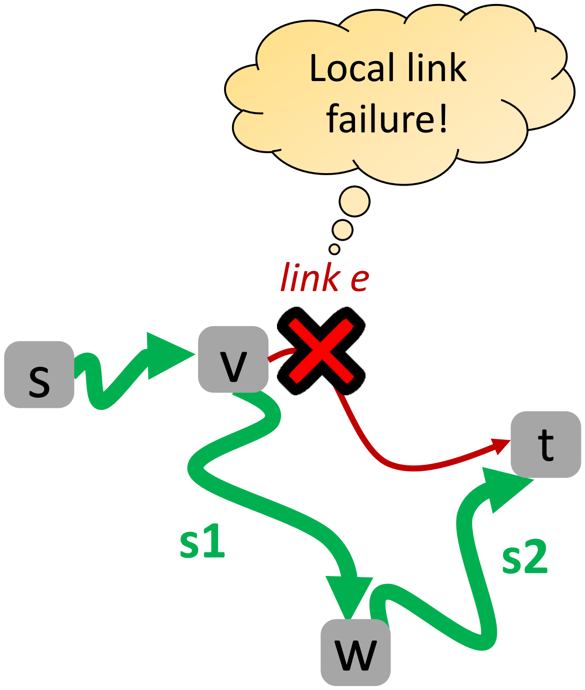
- When a *node v* on *route from s to t* *locally* detects failure on *link e*, it can *push a waypoint w*.



How to handle at least 1 failure?

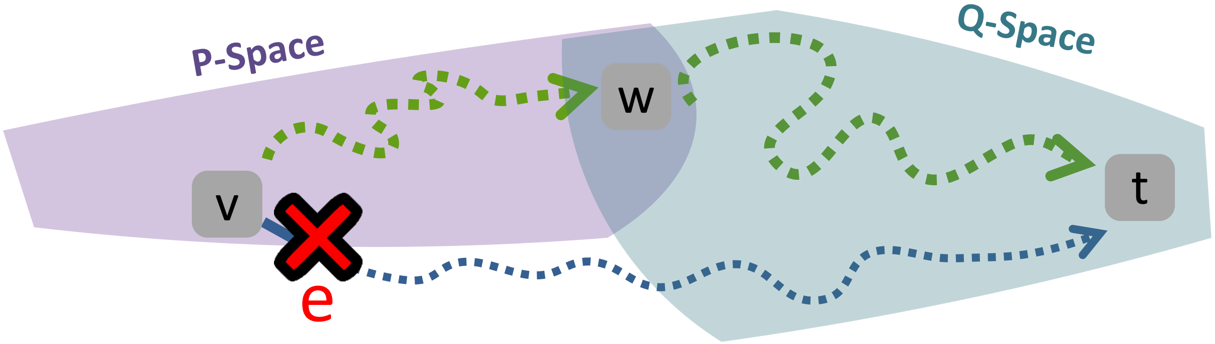
- When a *node v* on *route from s to t* *locally* detects failure on *link e*, it can *push a waypoint w*.

- **Rule:** v should push a w such that the *shortest path s1* (from v to w) and the *shortest path s2* (from w to t) does not include e again! So can route around.



A Local Solution

- We need two definitions:
  - **P-Space**: the nodes which *v* can reach on shortest paths without using e
  - **Q-Space**: the nodes which can reach *t* on shortest paths without using e



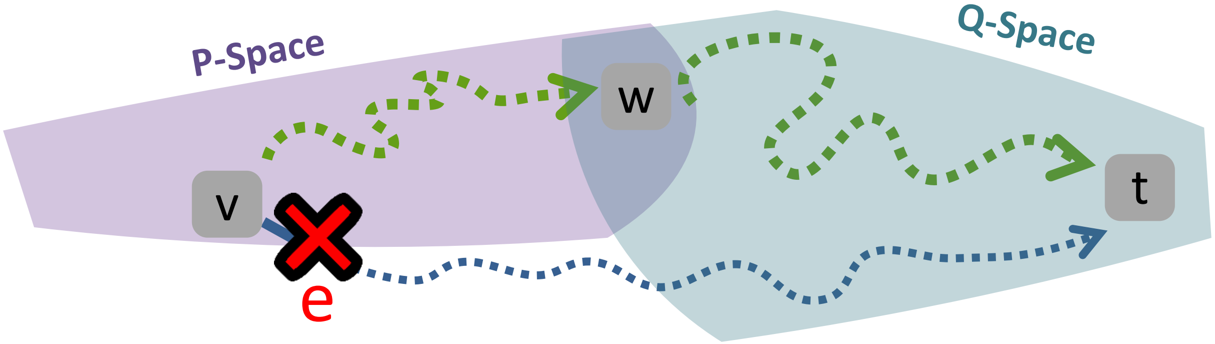
Then: choose *any waypoint w at intersection** for rerouting!

*If intersection empty, spaces must be adjacent and there is also a (different) solution.

A Local Solution

What about 2 failures?

- We need two definitions:
  - **P-Space**: the nodes which *v* can reach on shortest paths without using e
  - **Q-Space**: the nodes which can reach *t* on shortest paths without using e



Then: choose *any waypoint w at intersection** for rerouting!

*If intersection empty, spaces must be adjacent and there is also a (different) solution.

Literature: TI-MFA: Keep Calm and Reroute Segments Fast. Foerster et al., IEEE Global Internet Symposium (GI), 2018.

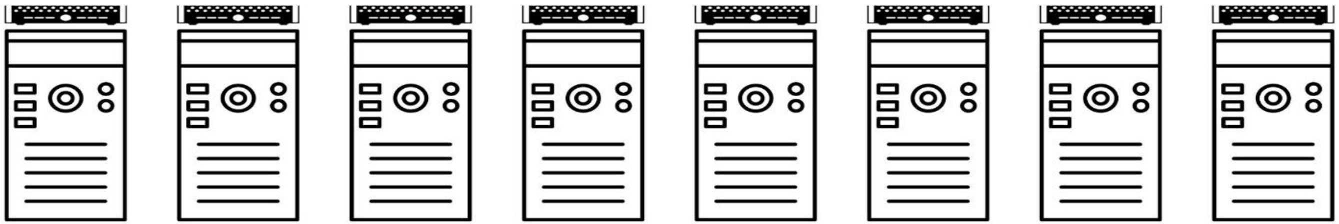
Roadmap: Two Examples

- Resilient routing

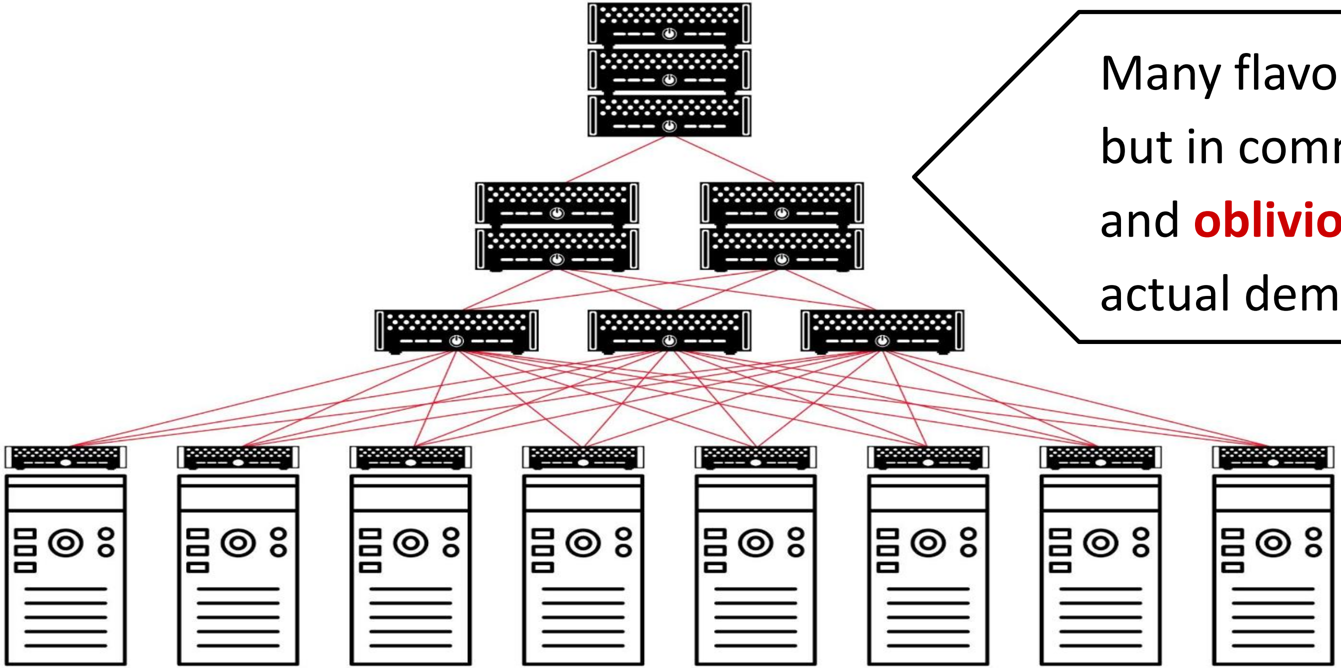
- Datacenter networks

Datacenter Networks

How to interconnect racks?



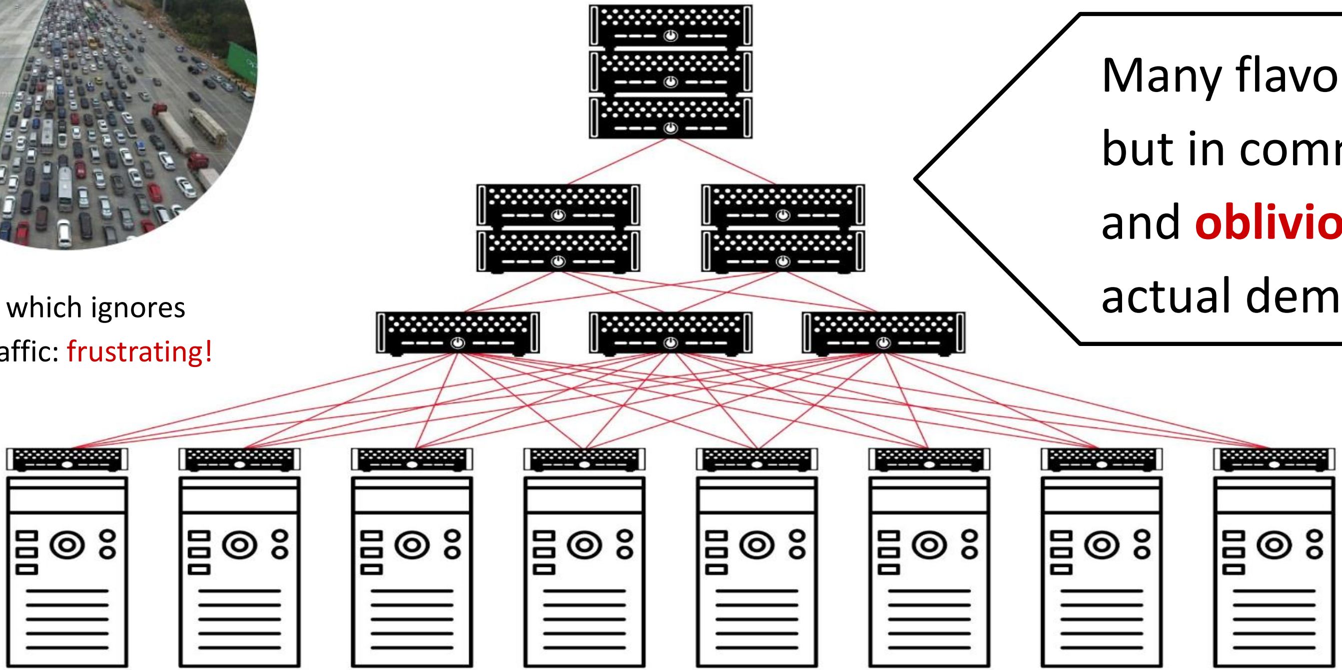
Datacenter Networks



Many flavors,
but in common: fixed
and **oblivious** to
actual demand.

Datacenter Networks

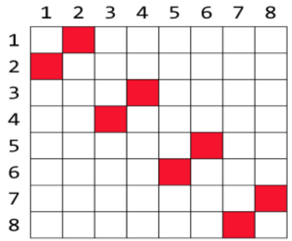
Highway which ignores actual traffic: frustrating!

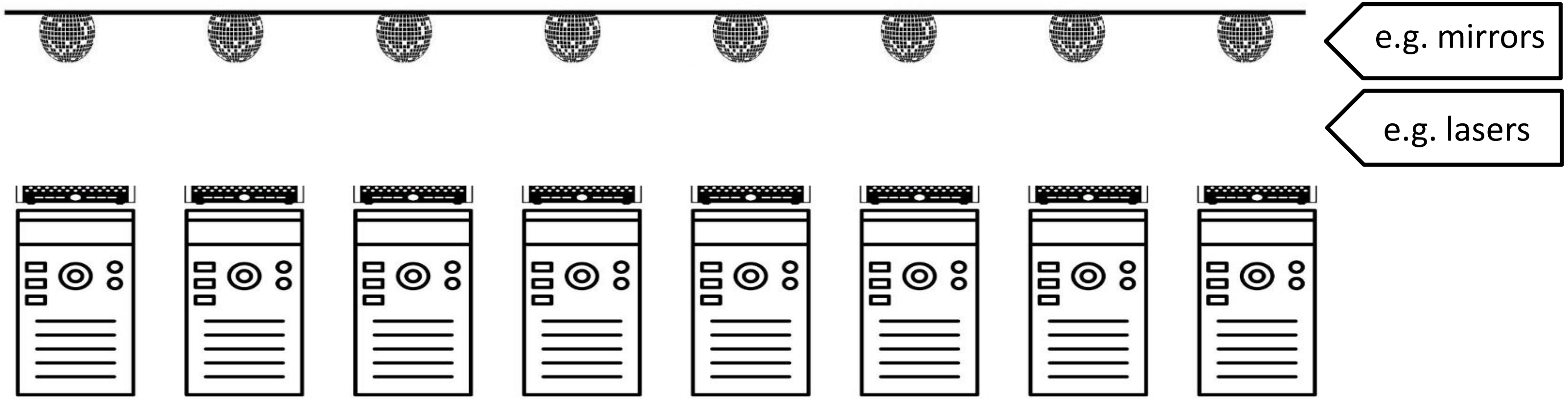


Many flavors,
but in common: fixed
and **oblivious** to
actual demand.

An Alternative: Reconfigurable

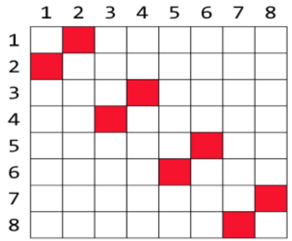
demand:



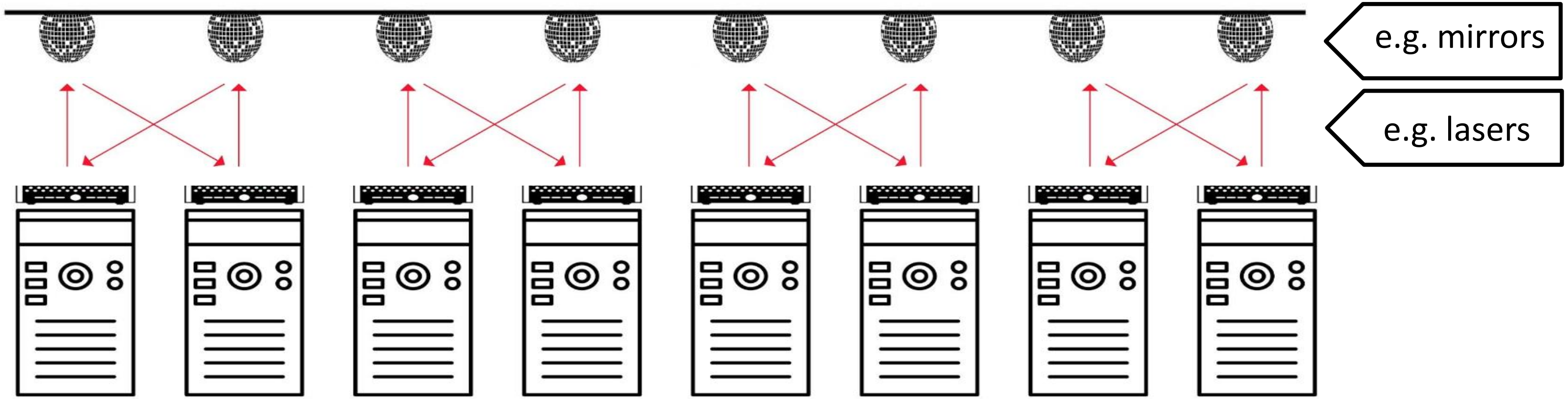


An Alternative: Reconfigurable

demand:

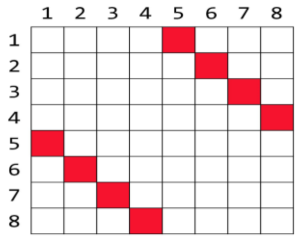


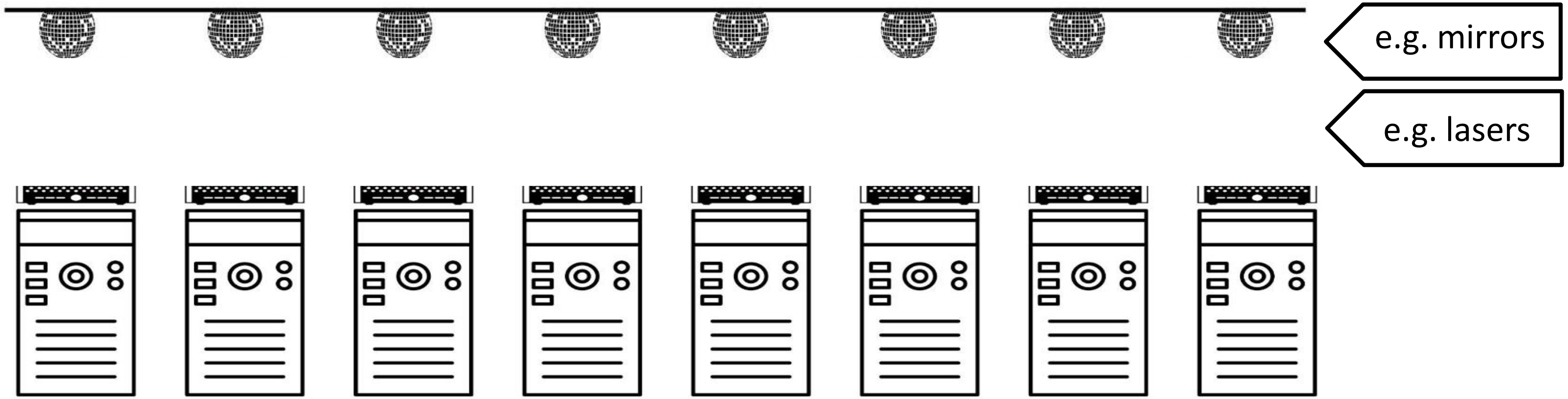
Matches demand!



An Alternative: Reconfigurable

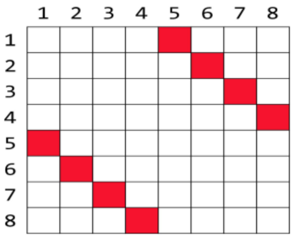
new demand:

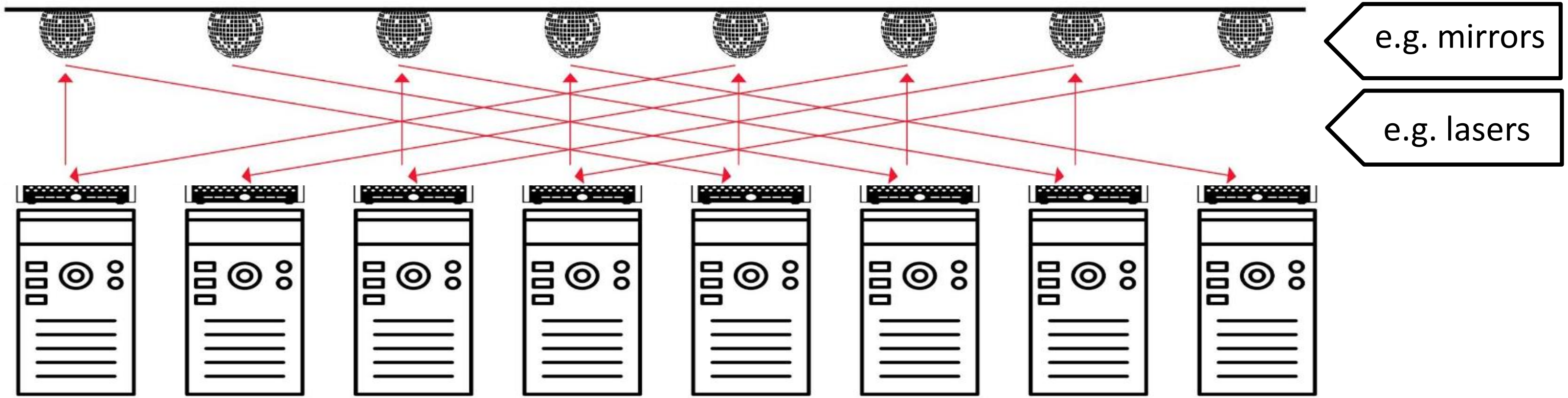




An Alternative: Reconfigurable

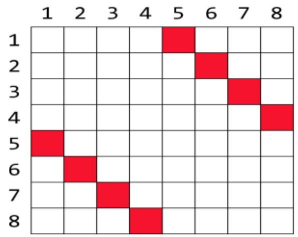
new demand:



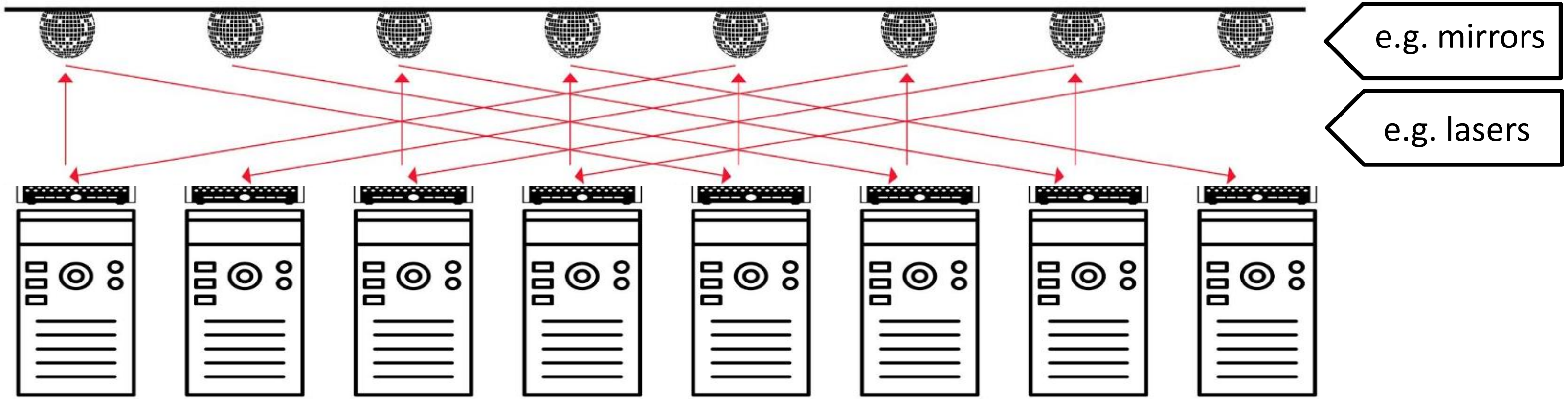


An Alternative: Reconfigurable

new demand:

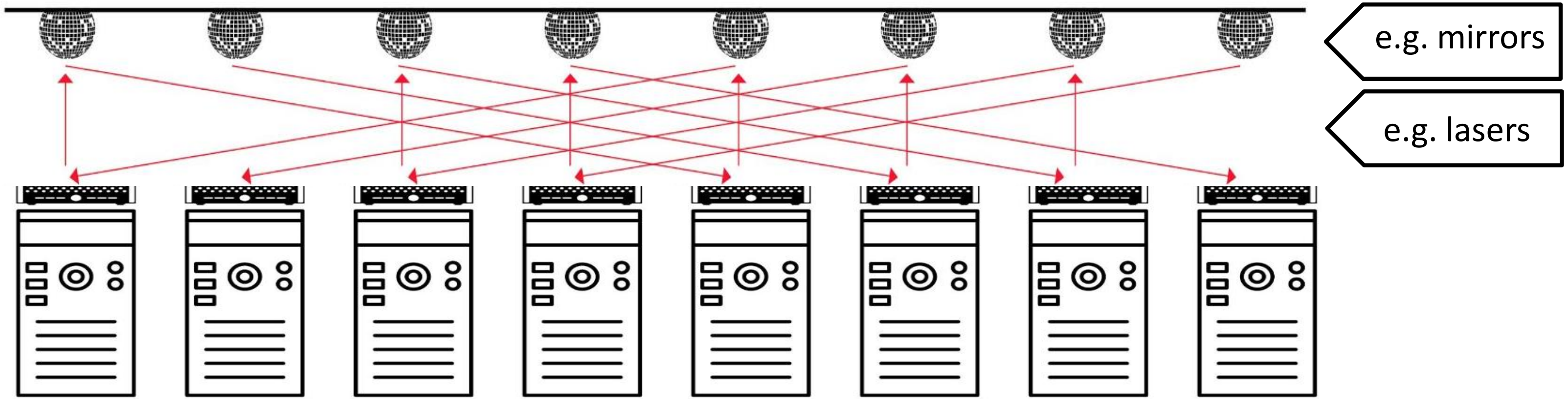


Matches demand!



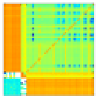
An Alternative: Reconfigurable

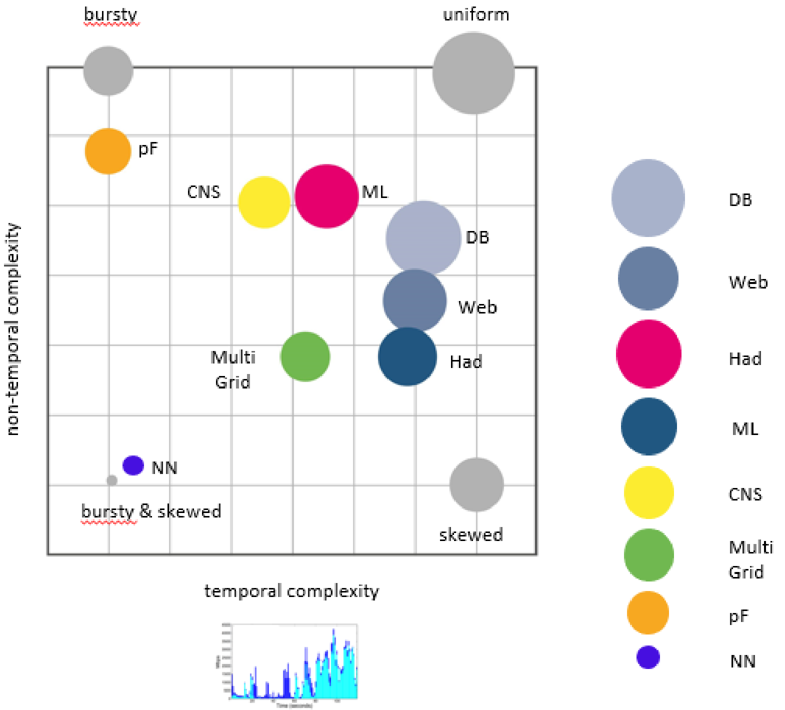
Self-adjusting networks: adapt in a demand-aware manner!



Empirical Motivation

- Workloads have much spatial and temporal structure
  - That is, low entropy
- Can be exploited for optimization

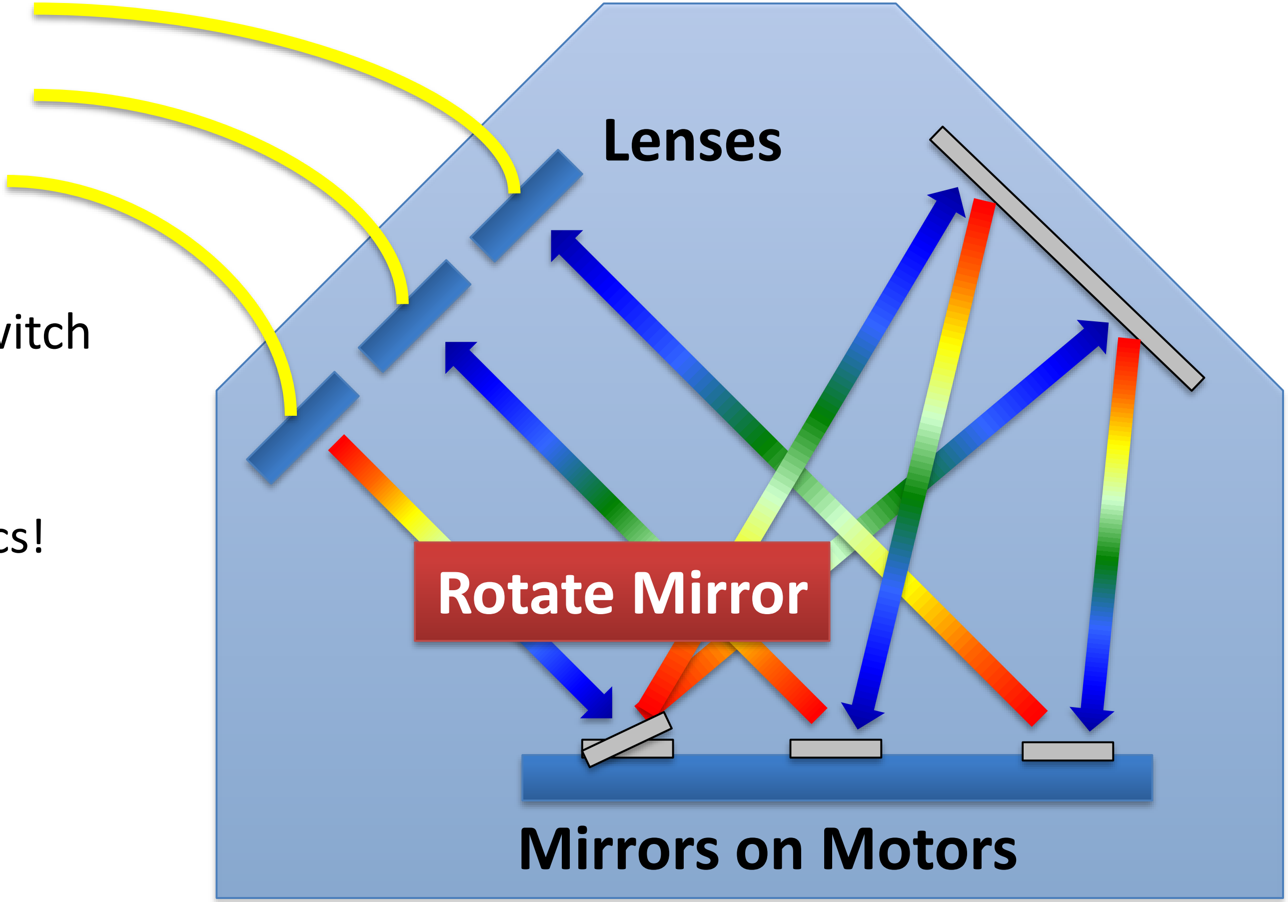




Literature: On the Complexity of Traffic Traces and Implications. Avin et al. ACM SIGMETRICS, 2020.

Enabler

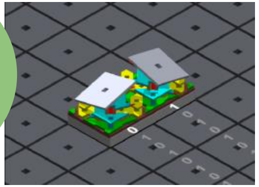
- Optical circuit switch
  - E.g., Google
- Adapt in microsecs!



Self-Adjusting Networks

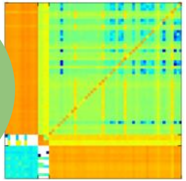
New!

Flexibility



More!

Structure

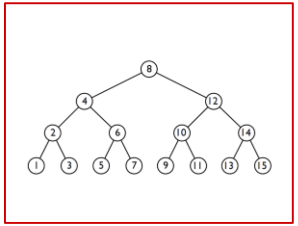


Self-Adjusting Networks

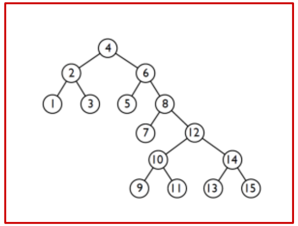
Efficiency

Connection to Datastructures & Coding

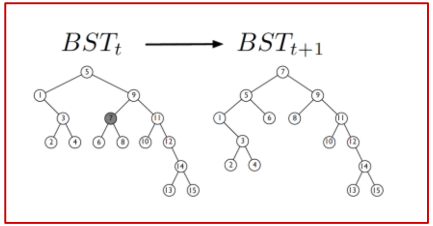
Traditional BST



Demand-aware BST

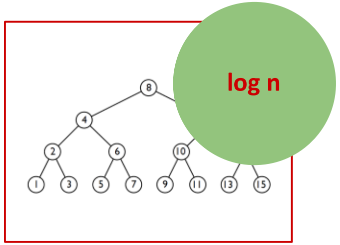


Self-adjusting BST

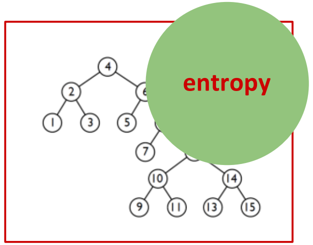


Connection to Datastructures & Coding

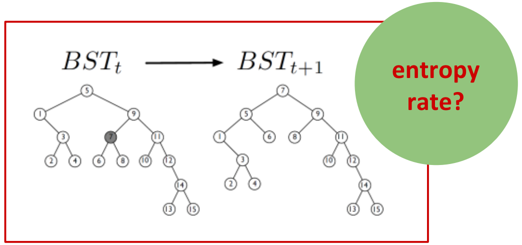
Traditional BST



Demand-aware BST

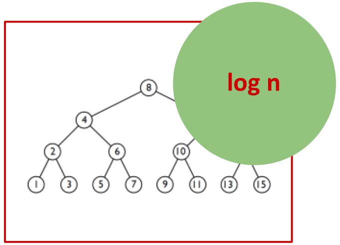


Self-adjusting BST

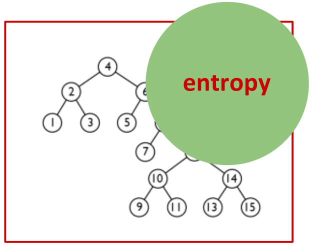


Connection to Datastructures & Coding

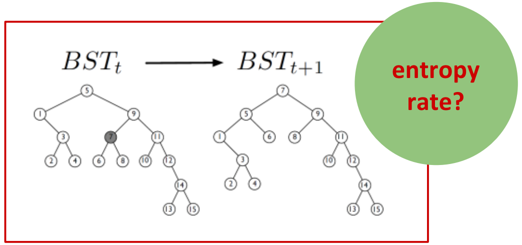
Traditional BST
(Worst-case coding)



Demand-aware BST
(Huffman coding)

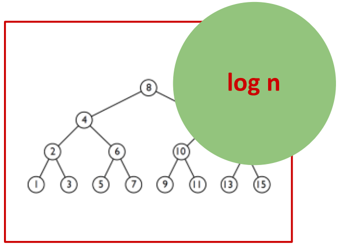


Self-adjusting BST
(Dynamic Huffman coding)

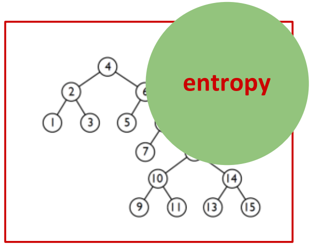


Connection to Datastructures & Coding

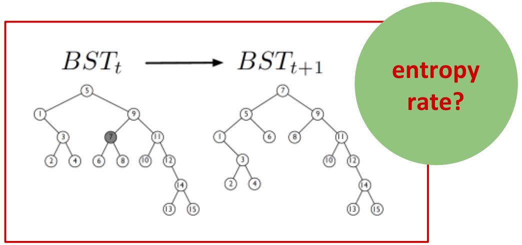
Traditional BST
(Worst-case coding)

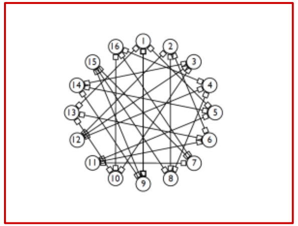


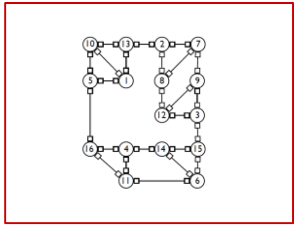
Demand-aware BST
(Huffman coding)

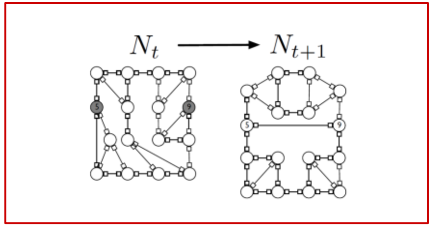


Self-adjusting BST
(Dynamic Huffman coding)









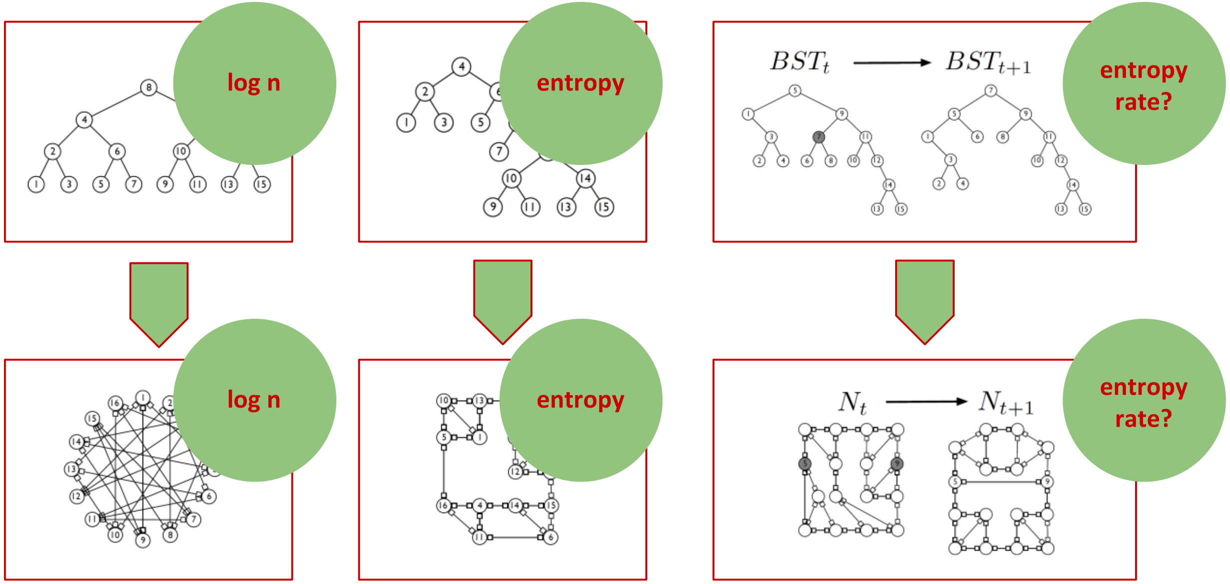
Reduced expected **route lengths**!

Connection to Datastructures & Coding

Traditional BST
(Worst-case coding)

Demand-aware BST
(Huffman coding)

Self-adjusting BST
(Dynamic Huffman coding)



Reduced expected **route lengths**!

More than an analogy!

Generalize methodology:

... and transfer entropy bounds and algorithms of data-structures to networks.

First result:

Demand-aware networks of asymptotically optimal route lengths.

First Deployments and a Challenge

- Google's ***demand-aware*** reconfigurable datacenter
- Key challenge according to ***Amin Vahdat***: scalable and ***distributed*** control


Systems

Jupiter evolving: Reflecting on Google's data center network transformation

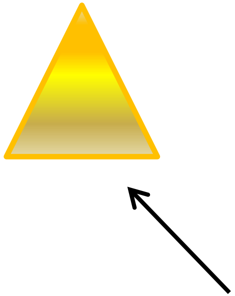
August 24, 2022

Amin Vahdat

VP & GM, Systems and Services Infrastructure

Example: Splay Networks

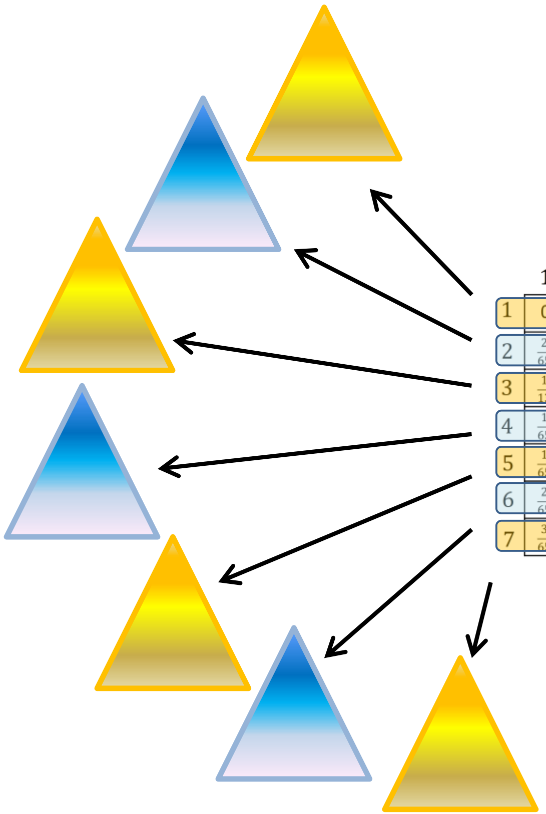
- Optimal static network for a source
  - Huffman tree or biased binary search tree



	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

Example: Splay Networks

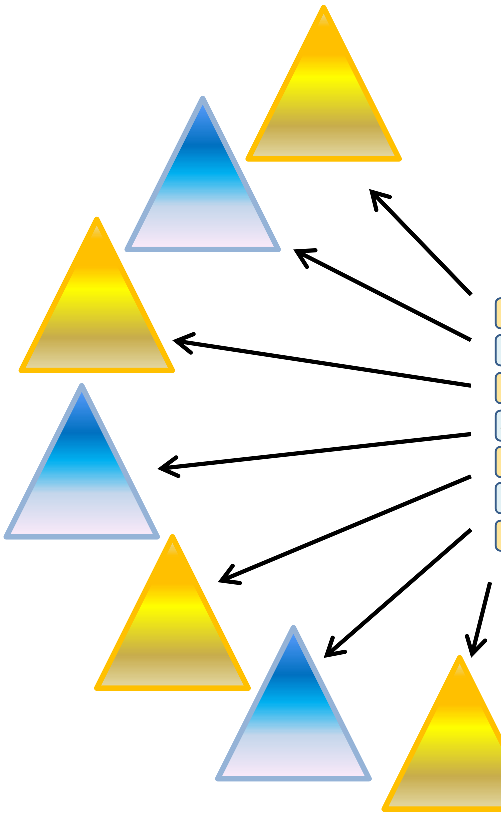
- Optimal static network for a source
  - Huffman tree or biased binary search tree
- For entire demand: take union
  - But reduce degree



	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

Example: Splay Networks

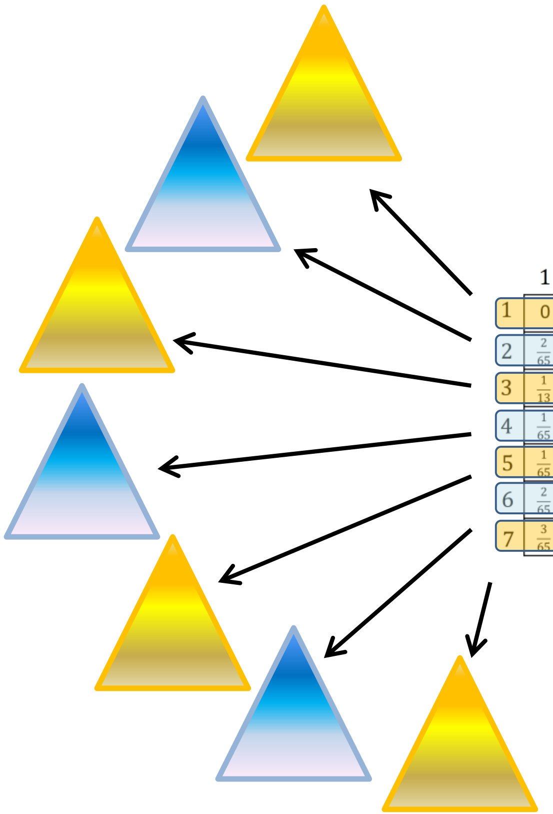
- Optimal static network for a source
  - Huffman tree or biased binary search tree
- For entire demand: take union
  - But reduce degree
- Dynamic: replace with splay tree



	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

Example: Splay Networks

- Optimal static network for a source
  - Huffman tree or biased binary search tree
- For entire demand: take union
  - But reduce degree
- Dynamic: replace with splay tree
- Distributed?
  - Distributed version of splay trees?



	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

Conclusion

- Wired networks: ***different*** from what you may think! And ***evolving***.
- Much control is ***distributed***
  - Congestion control, local fast re-routing, demand-aware networks
- A good moment to ***contribute***: on publications..
  - DISC expertise where other communities got stuck?
- … and in practice: have ***impact***, e.g., at standardizations at IETF, initiatives like Ultra Ethernet Consortium

Thank you!
Questions?

A Recent Survey

A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks
Marco Chiesa, Andrzej Kamisinski, Jacek Rak, Gabor Retvari, and Stefan Schmid.
IEEE Communications Surveys and Tutorials (**COMST**), 2021.

References

On the Price of Locality in Static Fast Rerouting
Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
52nd IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Baltimore, Maryland, USA, June 2022.

The Hazard Value: A Quantitative Network Connectivity Measure Accounting for Failures
Pieter Cuijpers, Stefan Schmid, Nicolas Schnepf, and Jiri Srba.
52nd IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Baltimore, Maryland, USA, June 2022.

On the Feasibility of Perfect Resilience with Local Fast Failover
Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
SIAM Symposium on Algorithmic Principles of Computer Systems (**APOCS**), Alexandria, Virginia, USA, January 2021.

Brief Announcement: What Can(not) Be Perfectly Rerouted Locally
Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
International Symposium on Distributed Computing (**DISC**), Freiburg, Germany, October 2020.

Improved Fast Rerouting Using Postprocessing
Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
IEEE Transactions on Dependable and Secure Computing (**TDSC**), 2020.

Resilient Capacity-Aware Routing
Stefan Schmid, Nicolas Schnepf and Jiri Srba.
27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**), Virtual Conference, March 2021.

AalWiNes: A Fast and Quantitative What-If Analysis Tool for MPLS Networks
Peter Gjøl Jensen, Morten Konggaard, Dan Kristiansen, Stefan Schmid, Bernhard Clemens Schrenk, and Jiri Srba.
16th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Barcelona, Spain, December 2020.

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.

More References

Randomized Local Fast Rerouting for Datacenter Networks with Almost Optimal Congestion
Gregor Bankhamer, Robert Elsässer, and Stefan Schmid..
International Symposium on Distributed Computing (**DISC**), Freiburg, Germany, October 2021.

Bonsai: Efficient Fast Failover Routing Using Small Arborescences
Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
49th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Portland, Oregon, USA, June 2019.

CASA: Congestion and Stretch Aware Static Fast Rerouting
Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan
38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

Load-Optimal Local Fast Rerouting for Dense Networks
Michael Borokhovich, Yvonne-Anne Pignolet, Gilles Tredan, and Stefan Schmid.
IEEE/ACM Transactions on Networking (**TON**), 2018.

PURR: A Primitive for Reconfigurable Fast Reroute
Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisinski, Georgios Nikolaidis, and Stefan Schmid.
15th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Orlando, Florida, USA, December 2019.
Artefact Evaluation: Available, Functional, Reusable.

On the Resiliency of Static Forwarding Tables
In IEEE/ACM Transactions on Networking (**ToN**), 2017
M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. Gurtov, A. Madry, M. Schapira, S. Shenker

Self-Adjusting Networks

Mars: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks
Vamsi Addanki, Chen Avin, and Stefan Schmid.
ACM **SIGMETRICS** and ACM Performance Evaluation Review (**PER**), Orlando, Florida, USA, June 2023.

Duo: A High-Throughput Reconfigurable Datacenter Network Using Local Routing and Control
Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid, and Chen Avin.
ACM **SIGMETRICS** and ACM Performance Evaluation Review (**PER**), Orlando, Florida, USA, June 2023.

Cerberus: The Power of Choices in Datacenter Topology Design (A Throughput Perspective)
Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin.
ACM **SIGMETRICS** and ACM Performance Evaluation Review (**PER**), Mumbai, India, June 2022.

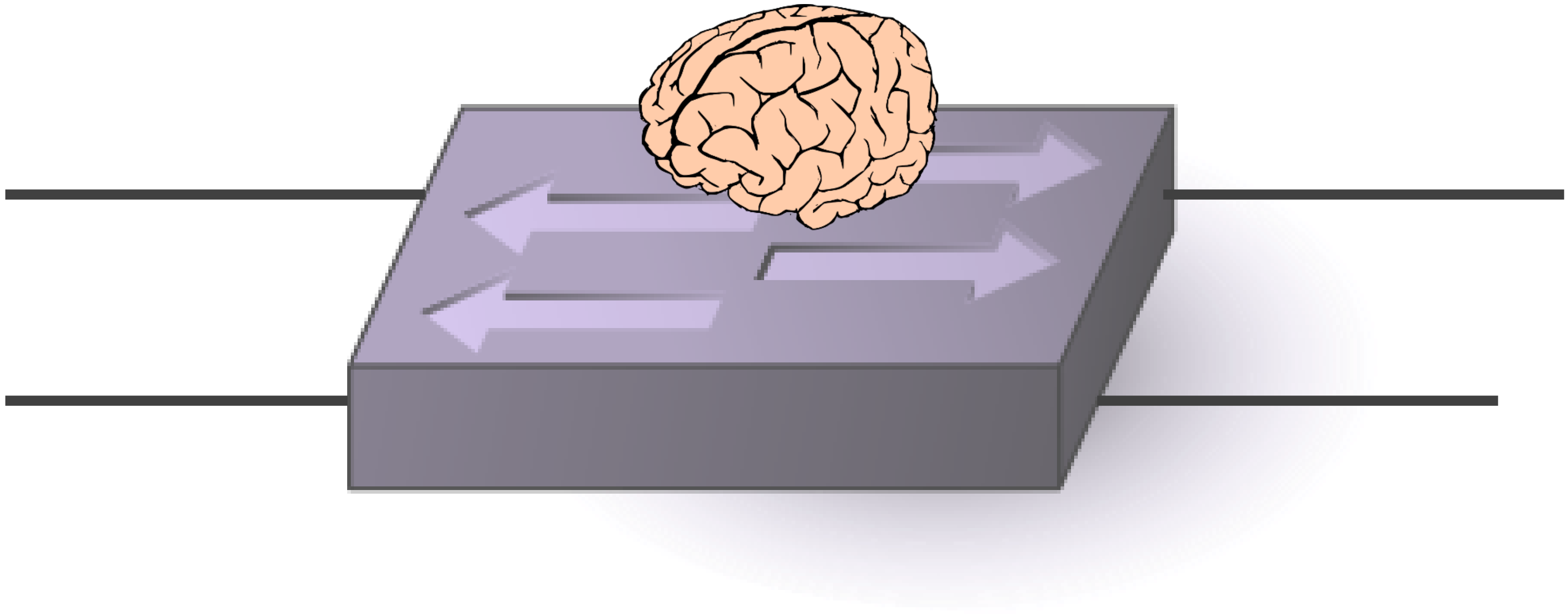
Demand-Aware Network Design with Minimal Congestion and Route Lengths
Chen Avin, Kaushik Mondal, and Stefan Schmid.
IEEE/ACM Transactions on Networking (**TON**), 2022.

On the Complexity of Traffic Traces and Implications
Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid.
ACM **SIGMETRICS** and ACM Performance Evaluation Review (**PER**), Boston, Massachusetts, USA, June 2020.

Backup Slides

Intelligent Routers: A Use Case

Intelligent Routers: A Use Case



Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

Intelligent Routers: A Use Case

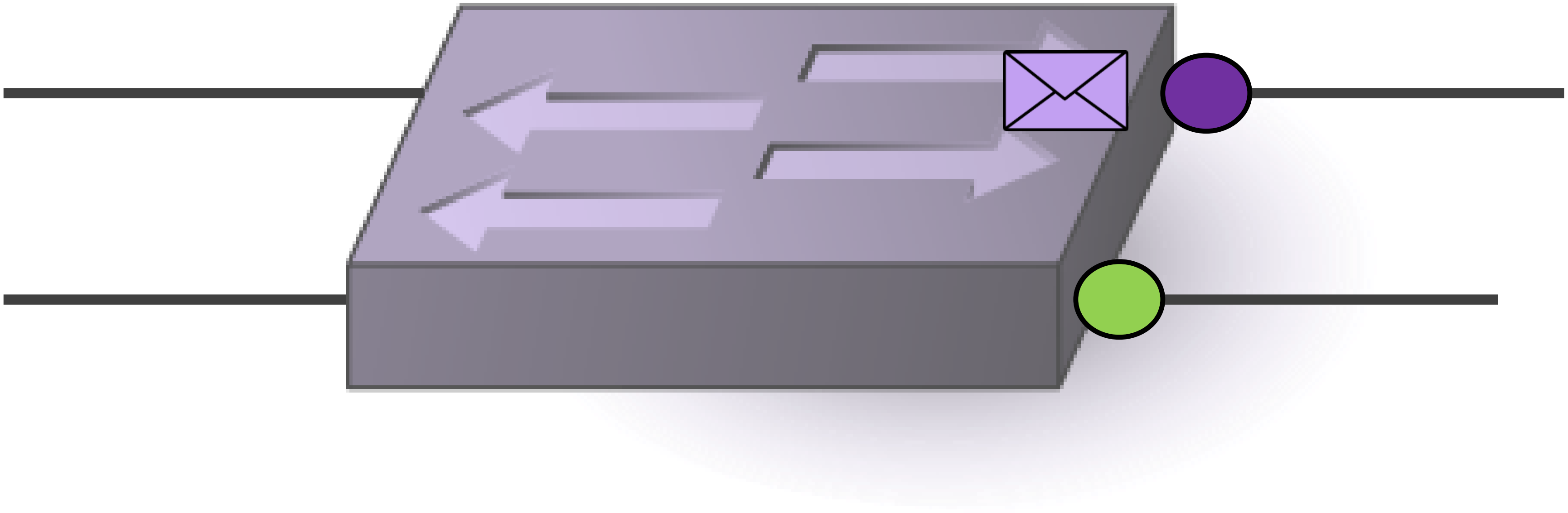
Assume: shared memory ***size 3***.

Scenario 1: assign buffer ***opportunistically***!

Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

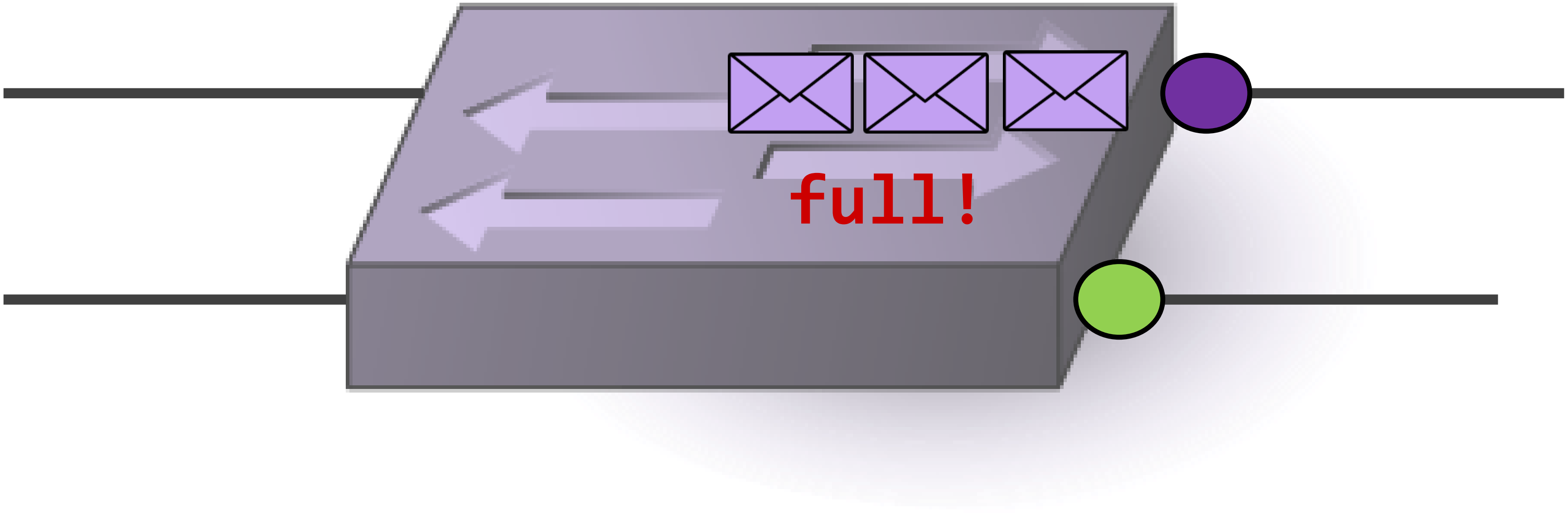
Scenario 1: assign buffer ***opportunistically***!



Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

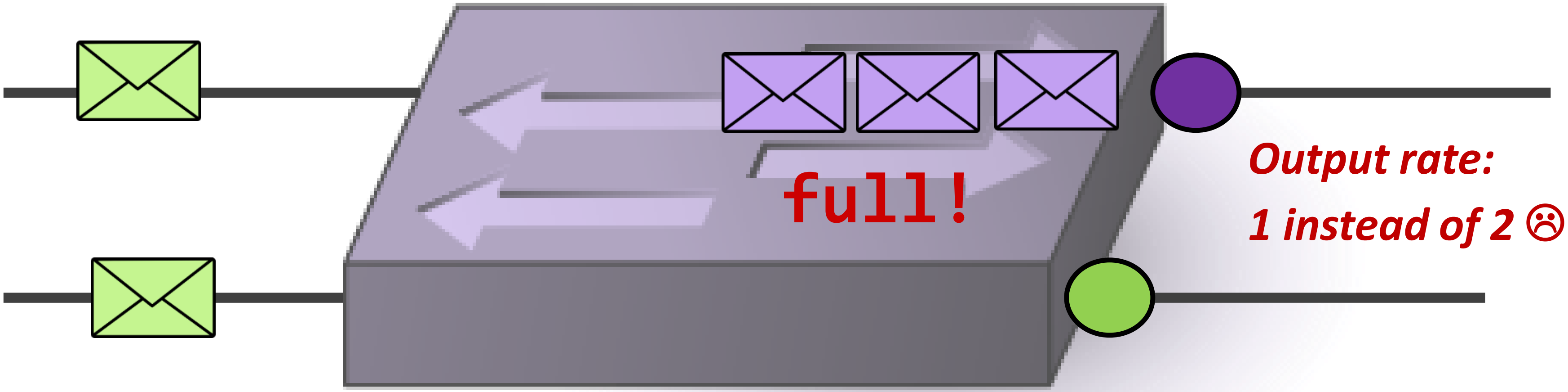
Scenario 1: assign buffer ***opportunistically***!



Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

Scenario 1: assign buffer ***opportunistically***!

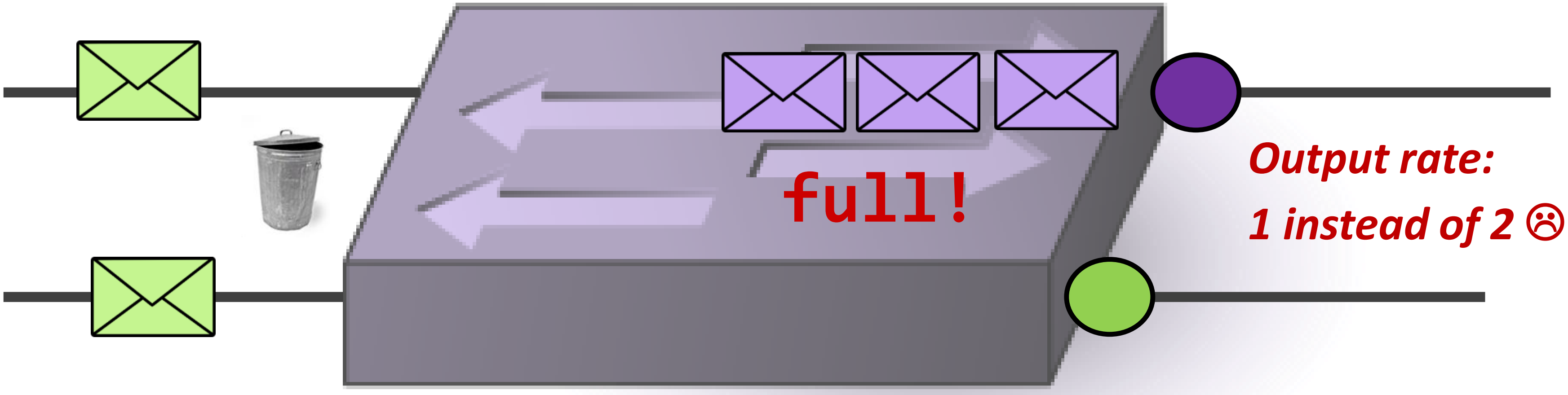


Suboptimal: green packets could be transmitted ***in parallel***, but there is no more space!

Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

Scenario 1: assign buffer ***opportunistically***!



Output rate:

1 instead of 2 ☹

Suboptimal: green packets could be transmitted ***in parallel***, but there is no more space!

Intelligent Routers: A Use Case

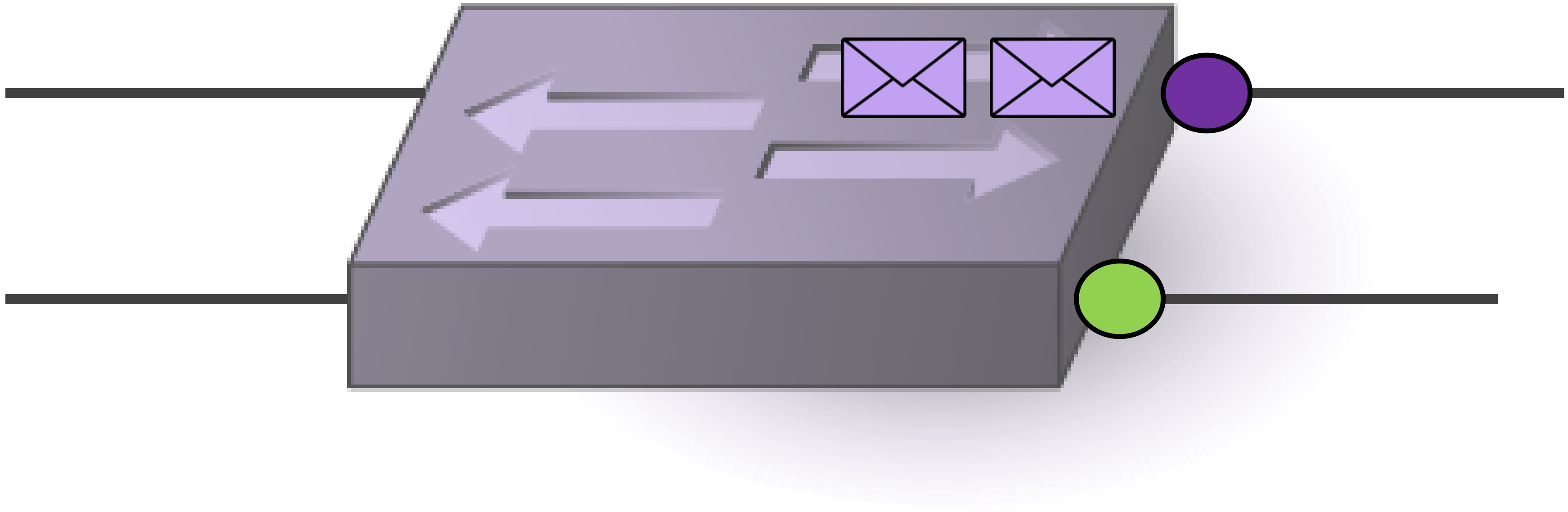
Assume: shared memory ***size 3***.

Scenario 2: assign buffer ***conservatively*** and ***keep space***.

Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

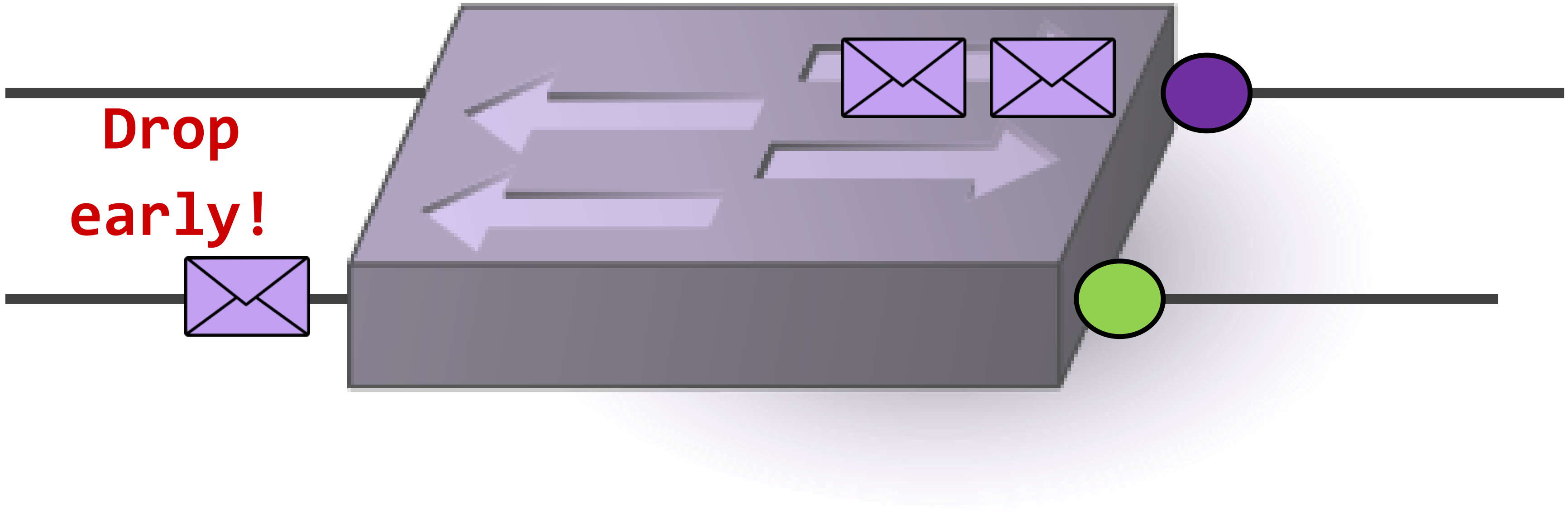
Scenario 2: assign buffer ***conservatively*** and ***keep space***.



Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

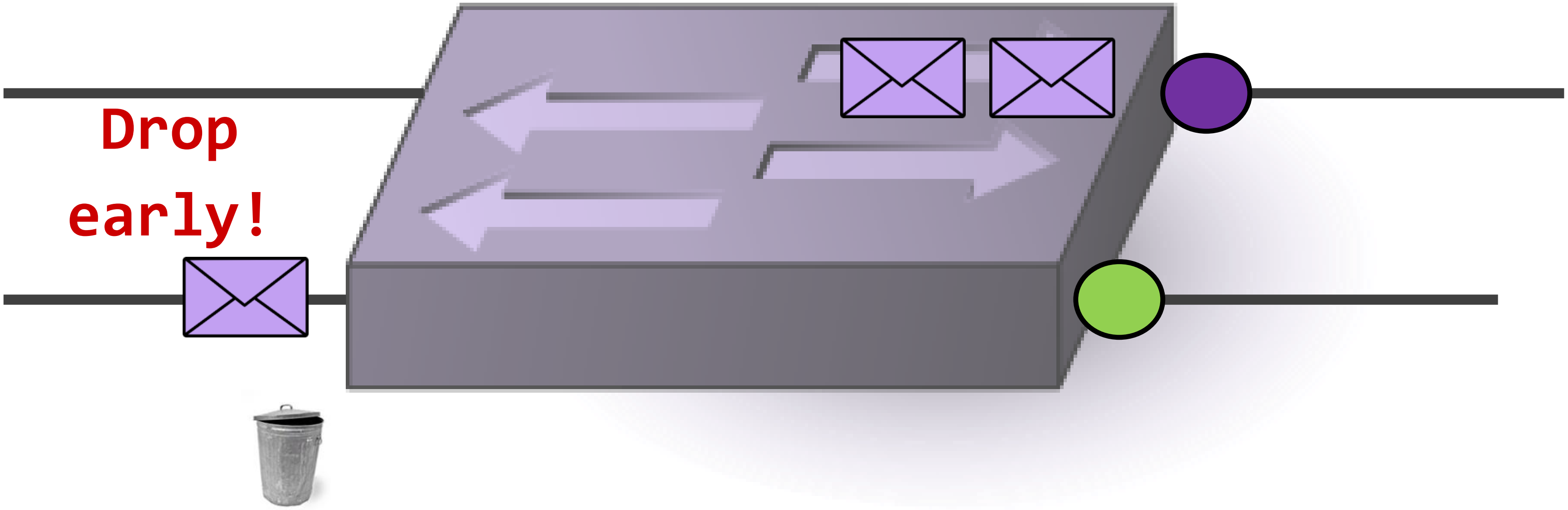
Scenario 2: assign buffer ***conservatively*** and ***keep space***.



Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

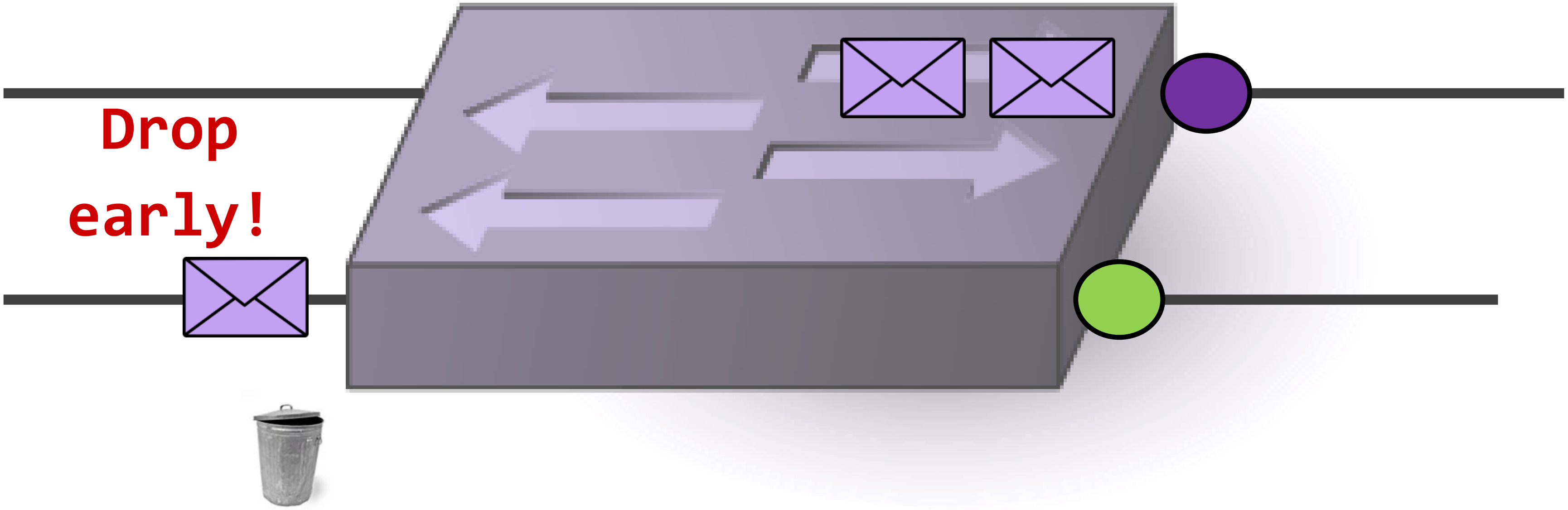
Scenario 2: assign buffer ***conservatively*** and ***keep space***.



Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

Scenario 2: assign buffer ***conservatively*** and ***keep space***.

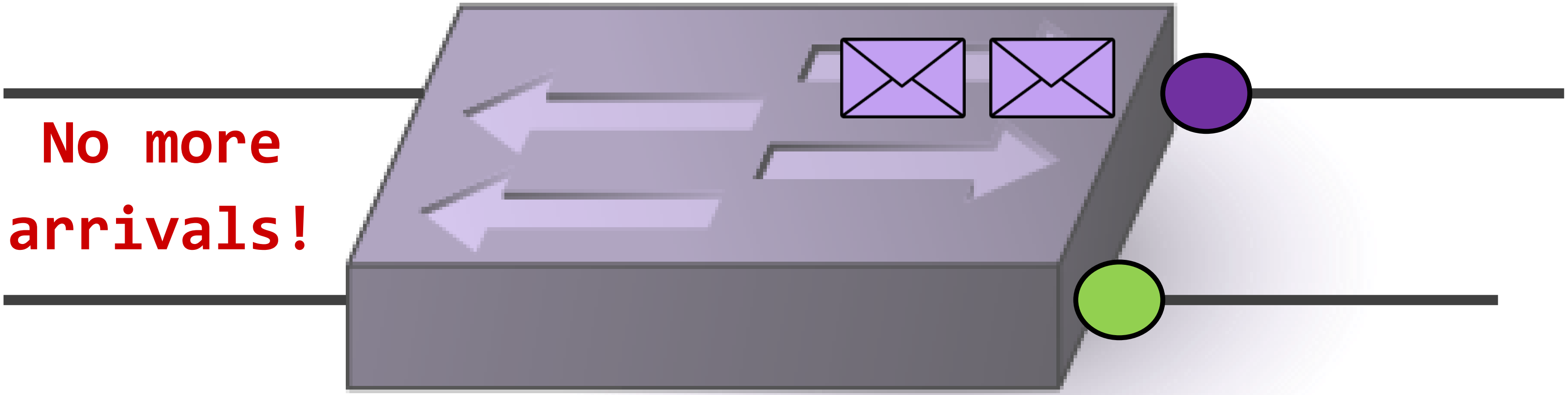


Intelligent Routers: A Use Case

Assume: shared memory ***size 3***.

Scenario 2: assign buffer ***conservatively*** and ***keep space***.

No more arrivals!



Suboptimal: drops were unnecessary, buffer not needed for green packets!

Credence

- Traffic at switch can be ***predicted*** fairly well
- AI/ML could significantly ***improve buffer management***…
- … and hence ***admission control and throughput***!
- Further reading:

Credence: Augmenting Datacenter Switch Buffer Sharing with ML Predictions
Vamsi Addanki, Maciej Pacut, and Stefan Schmid.
21st USENIX Symposium on Networked Systems Design and Implementation (**NSDI**), 2024.