Algorithmic Nuggets in Network Slicing and Resource Allocation

Stefan Schmid

Faculty of Computer Science
University of Vienna

IEEE ComSoc Autumn School
on Network Slicing and Data Driven Communication

Virtualization and Programmability:
It's a great time to be a networking researcher!

Rhone and Arve Rivers, Switzerland

Credits: George Varghese.

Teaching Goals: You will understand…

- … where flexibilities arise in current networks
- … how to exploit them algorithmically
- … requirements and challenges of performance isolation
- … where more research is urgently needed ☺

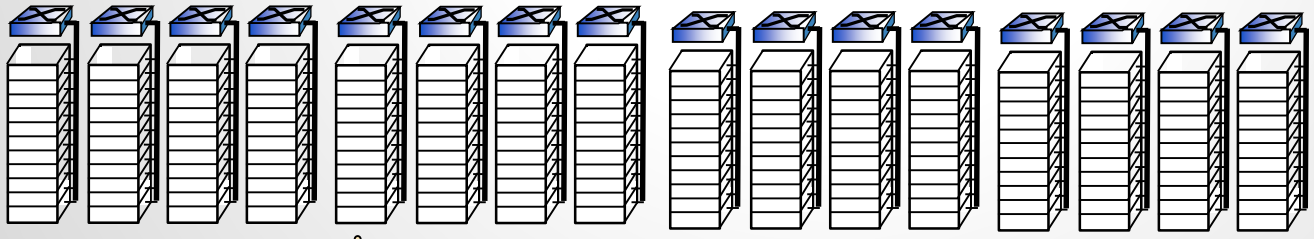
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - Network virtualization: today and tomorrow
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Toward self-adjusting networks
- Security aspects (if time permits)

Roadmap

- **A short recap:**
  - **Networking preliminaries**
  - Opportunities and challenges of SDNs
  - Network virtualization: today and tomorrow
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Toward self-adjusting networks
- Security aspects (if time permits)

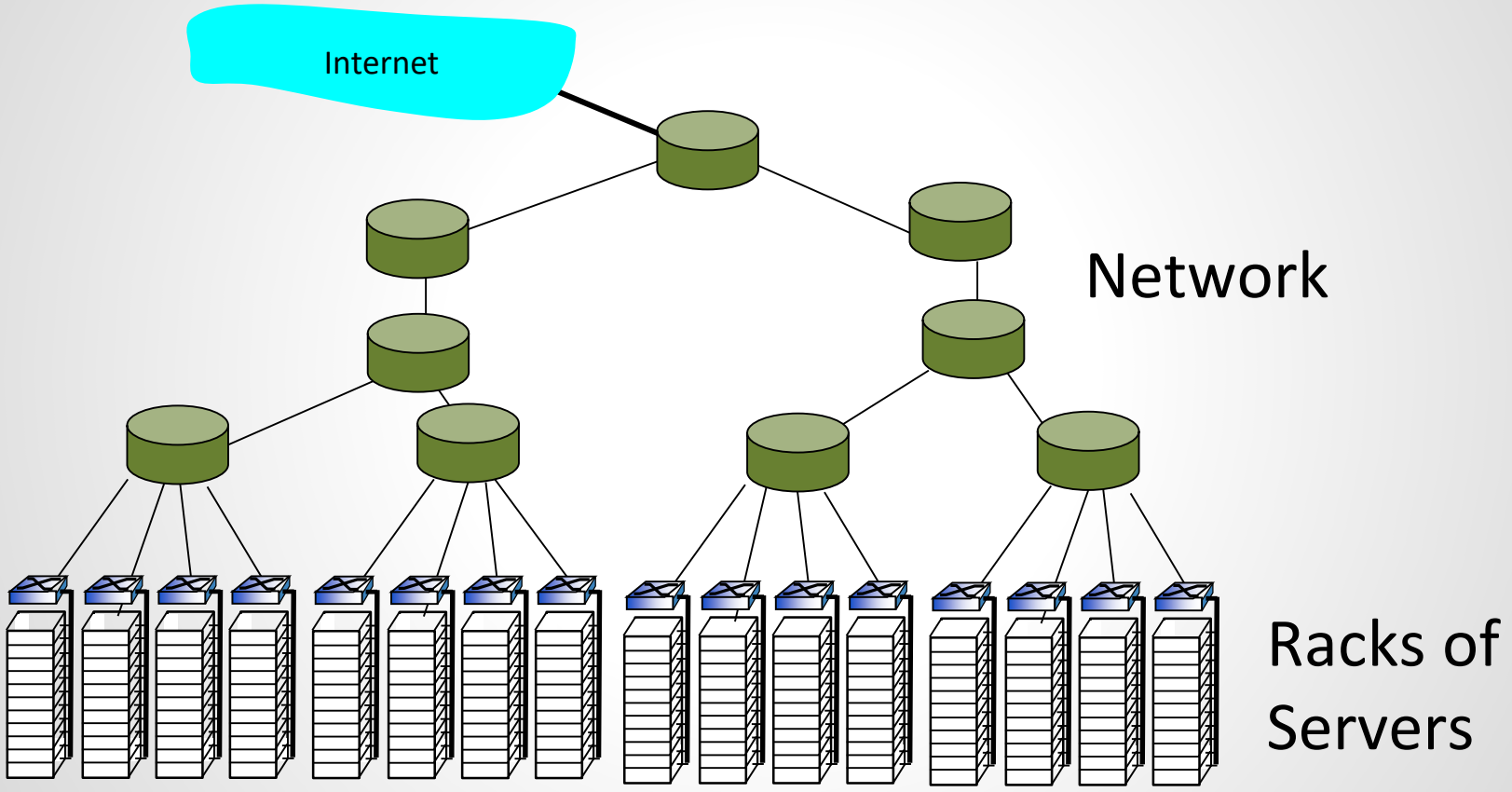
How to design a datacenter network?



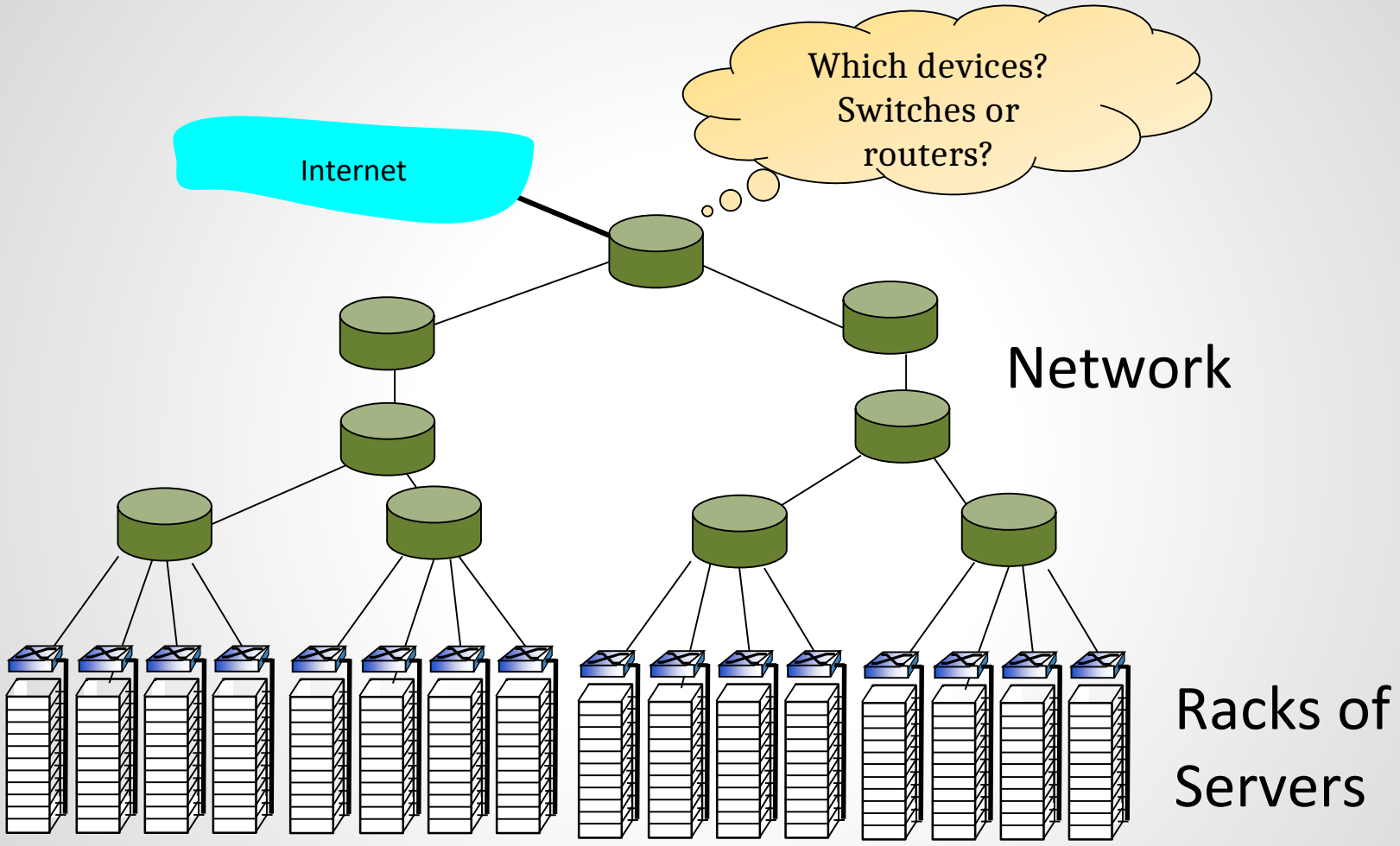
Racks of Servers

Many VMs!

How to design a datacenter network?



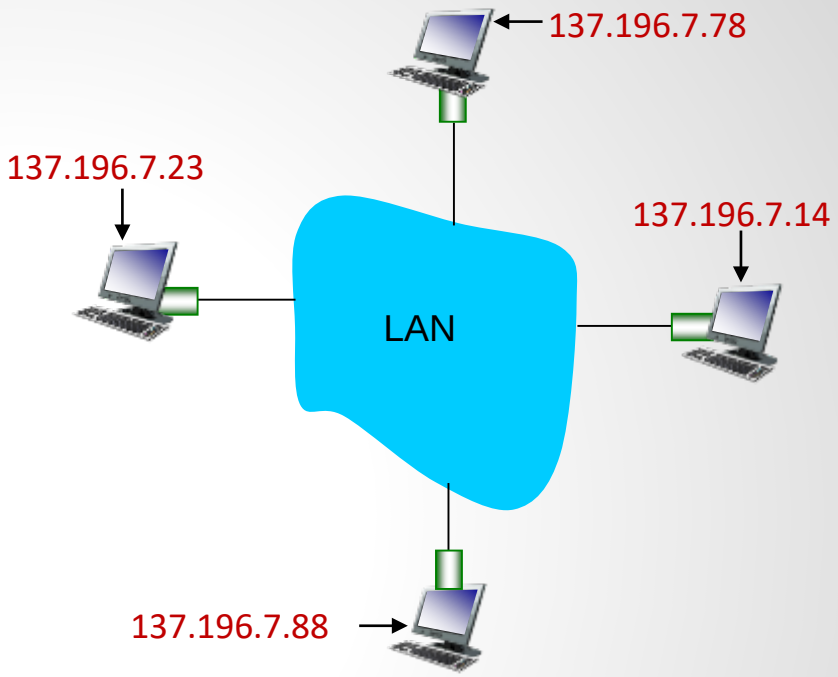
How to design a datacenter network?



To understand the tradeoff: Preliminaries

Network Layer:

- To transport datagrams between hosts…
- … routers execute a routing algorithm: compute shortest and policy-compliant paths…
- … based on hierarchical, location-dependent IP addresses!

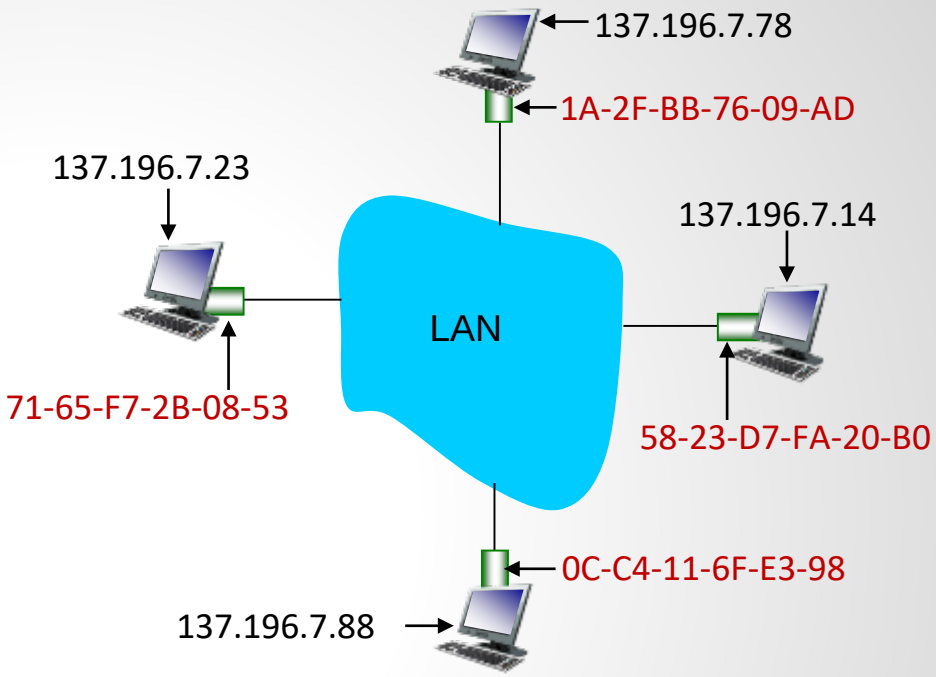


IP address like postal address: changes when moving to new LAN.

To understand the tradeoff: Preliminaries

Link Layer:

- Switches rely on flat, location-independent and globally unique MAC addresses to forward frames
- Address portable across LANs

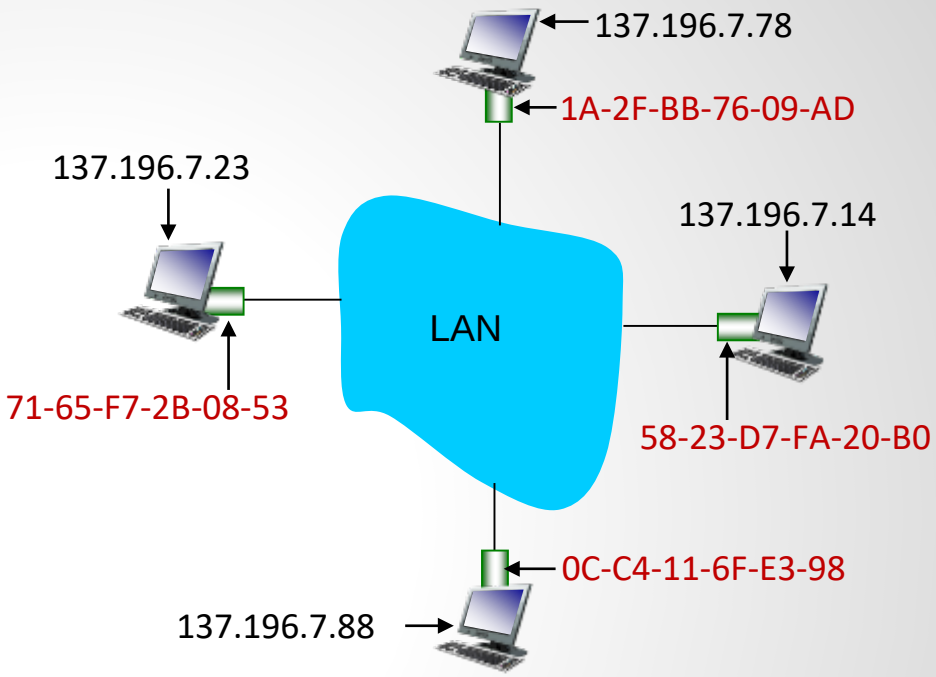


MAC address like Social Security number.

To understand the tradeoff: Preliminaries

Link Layer:

- Switches rely on flat, location-independent and globally unique MAC addresses to forward frames
- Address portable across LANs



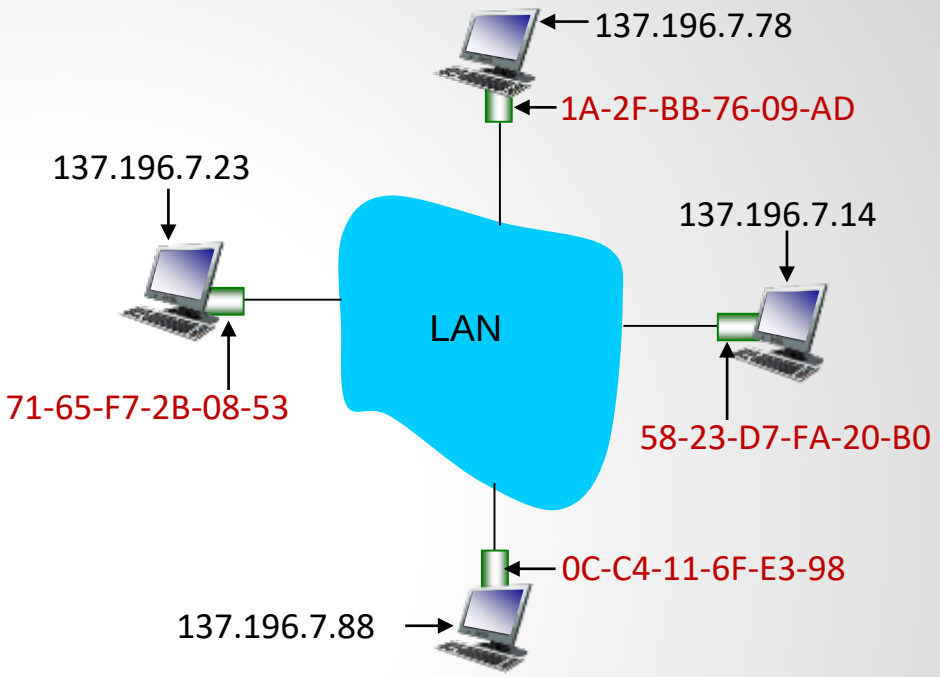
MAC address like Social Security number.

But how do switches in a LAN know to which ports they need to forward a packet?

To understand the tradeoff: Preliminaries

Link Layer:

- Switches rely on flat, location-independent and globally unique MAC addresses to forward frames
- Address portable across LANs



MAC address like Social Security number.

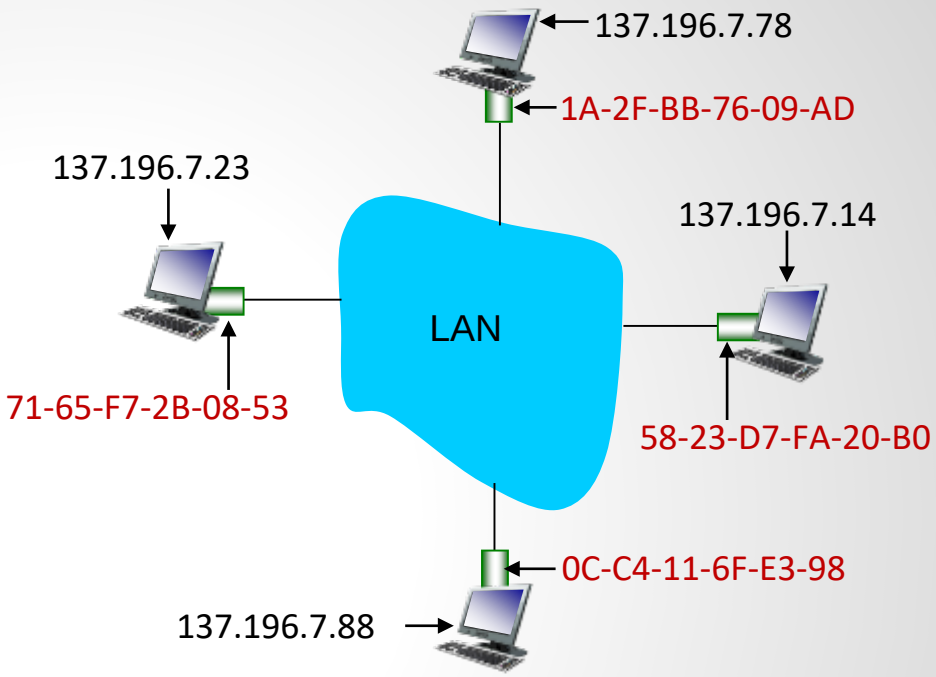
But how do switches in a LAN know to which ports they need to forward a packet?

And given an IP address (e.g., of gateway router) how can a host determine the required MAC destination address?

To understand the tradeoff: Preliminaries

Link Layer:

- Switches rely on flat, location-independent and globally unique MAC addresses to forward frames
- Address portable across LANs

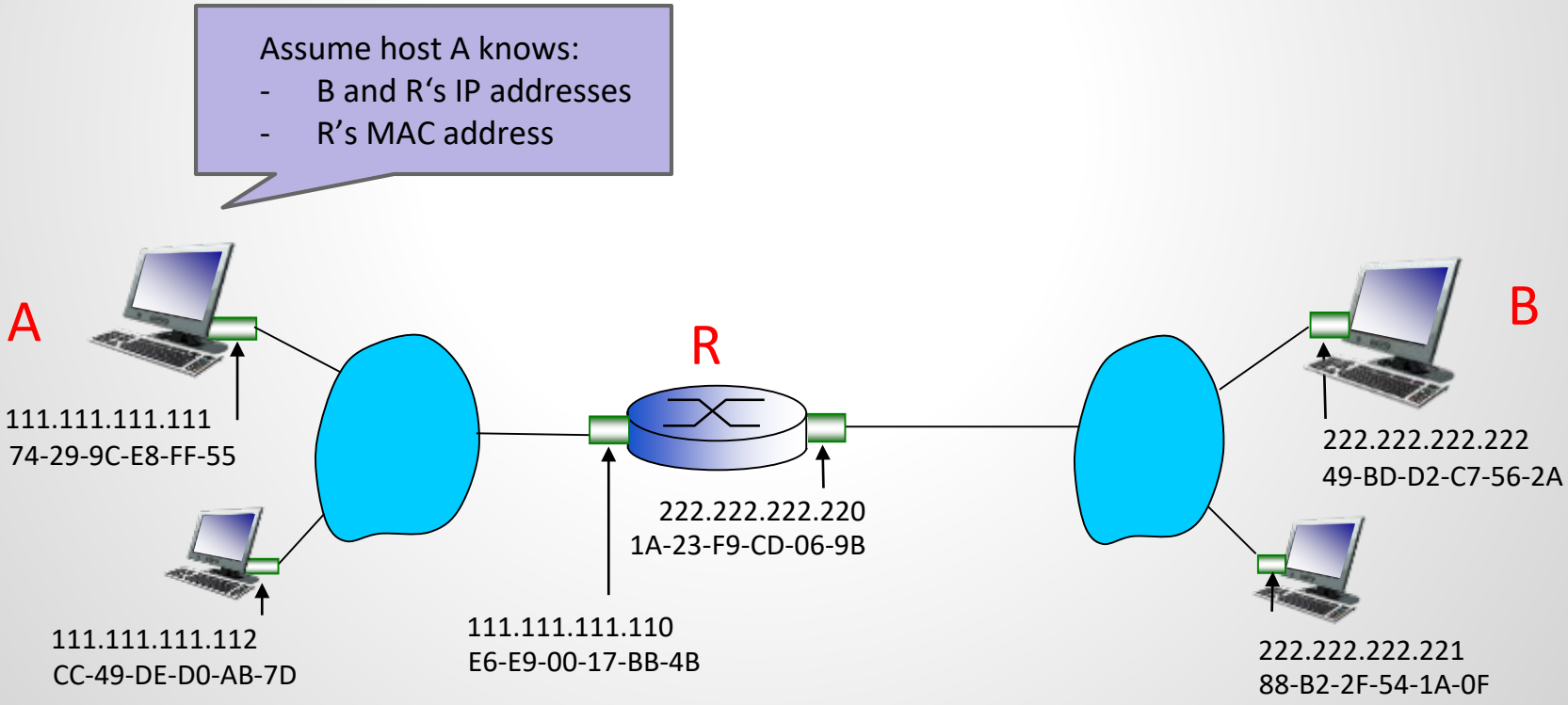


MAC address like Social Security number.

LANs are not about routing
but broadcasting and learning.
Layer-2 network = single broadcast domain.

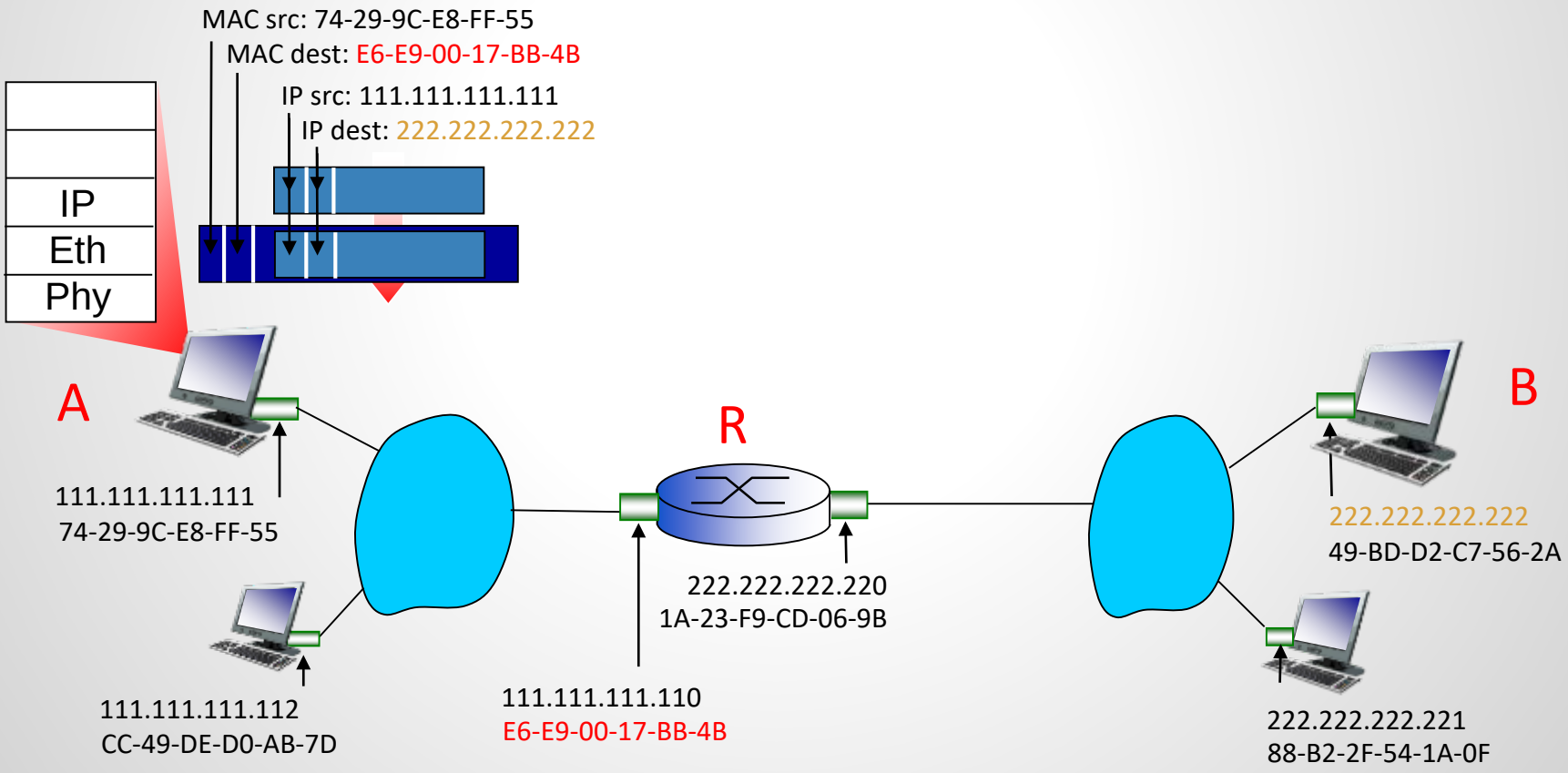
Recall: Routing to Another LAN

Assume: host A wants to send a packet to host B via router R:



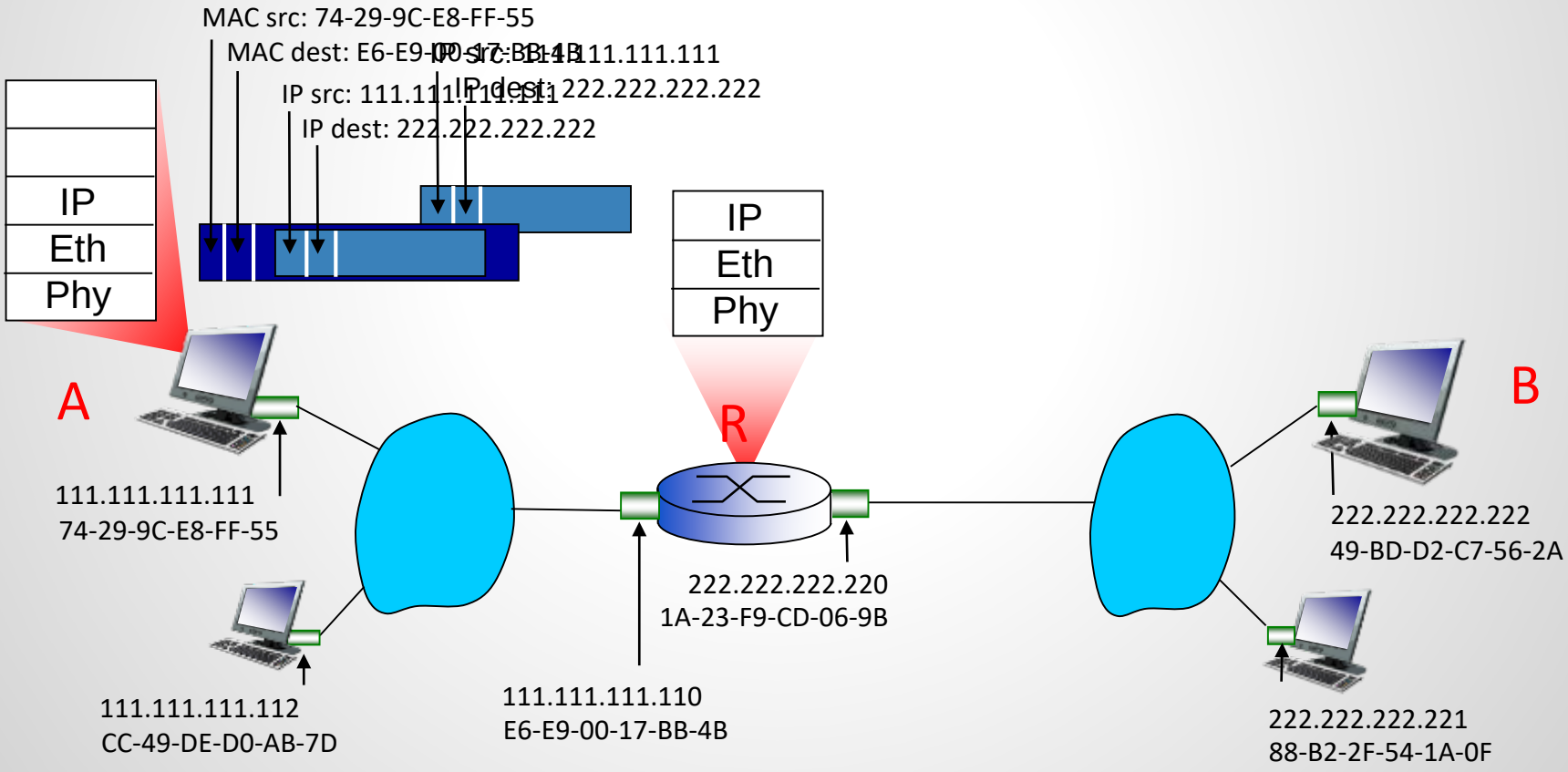
Recall: Routing to Another LAN

- Step 1: Host A creates IP datagram (IPsrc=A, IPdst=B)
- Step 2: A creates link-layer frame with MACdst=R (R's MAC address)



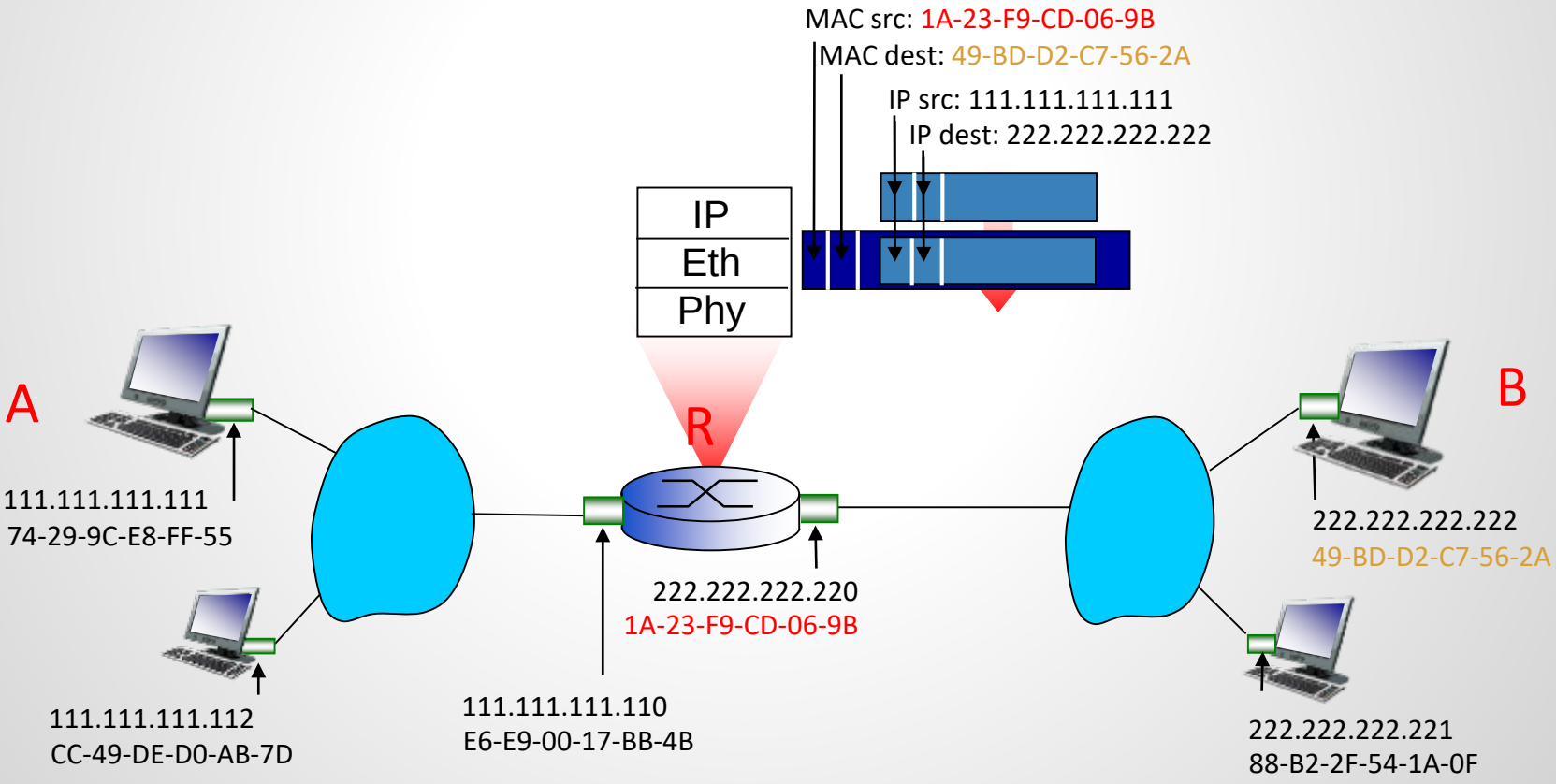
Recall: Routing to Another LAN

- Step 3: Frame sent from A to R
- Step 4: Frame received at R, passed up to IP



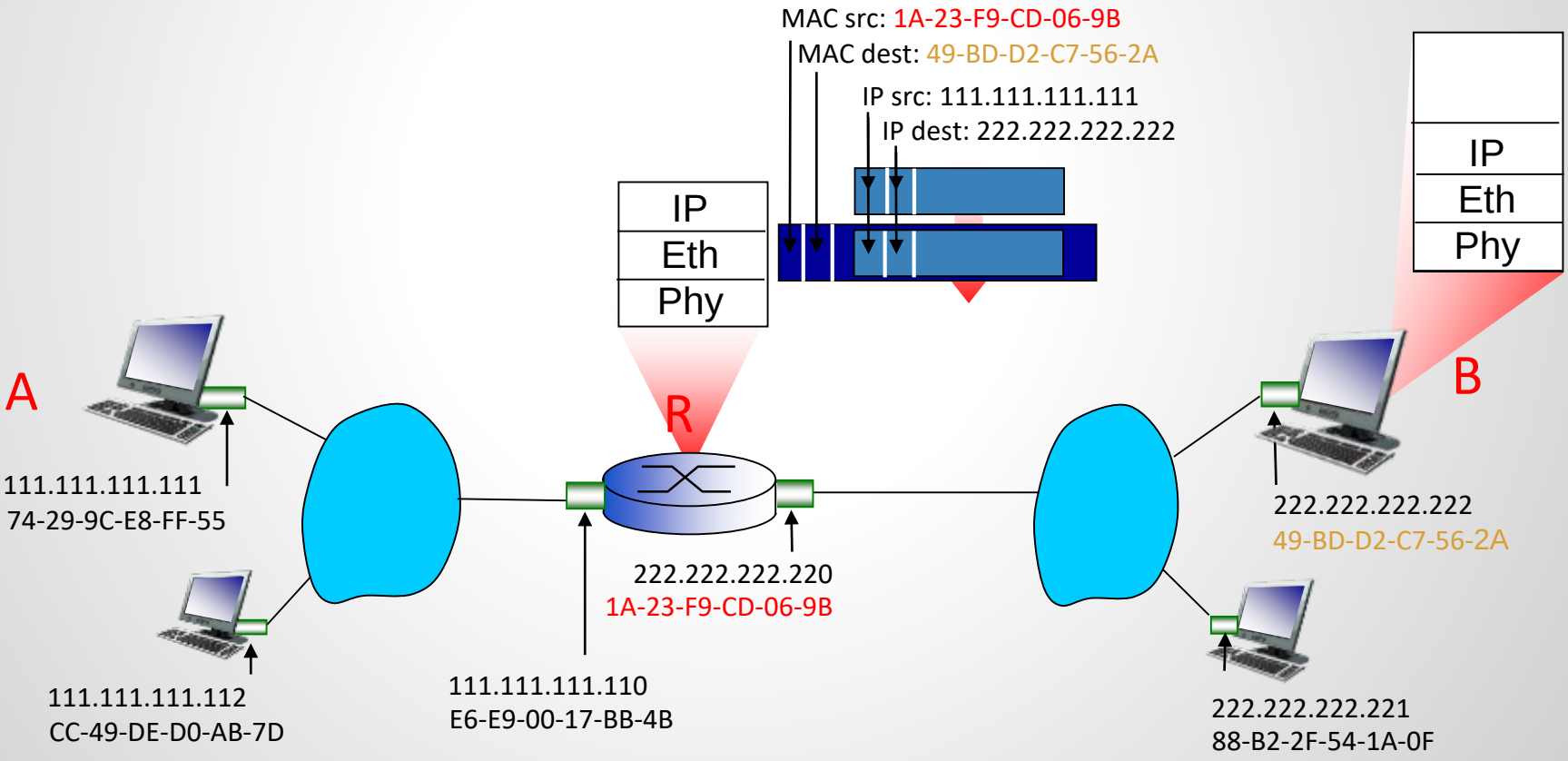
Recall: Routing to Another LAN

- Step 5: R forwards datagram with *same* IPsrc=A, IPdst=B
- Step 6: R creates link-layer frame with MACdst=B (B's MAC address), frame contains A-to-B IP datagram



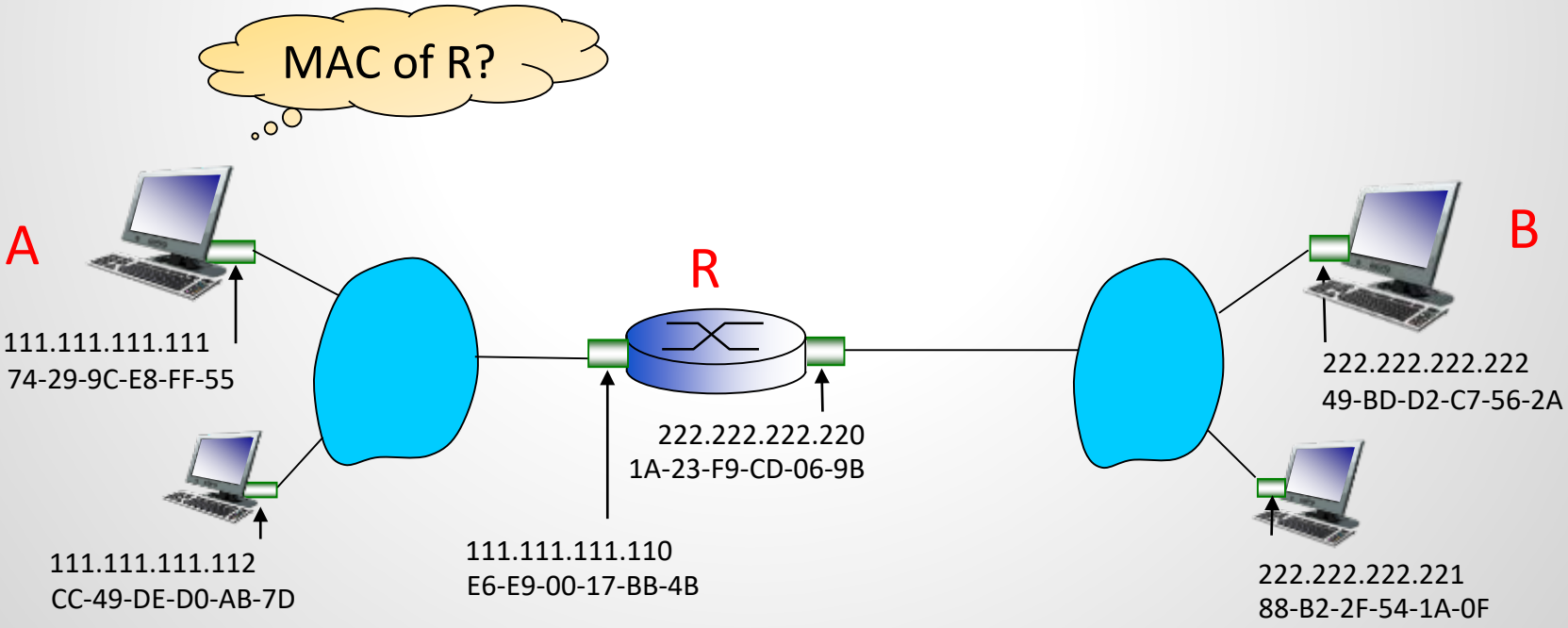
Recall: Routing to Another LAN

- Step 5: R forwards datagram with *same* IPsrc=A, IPdst=B
- Step 6: R creates link-layer frame with MACdst=B (B's MAC address), frame contains A-to-B IP datagram

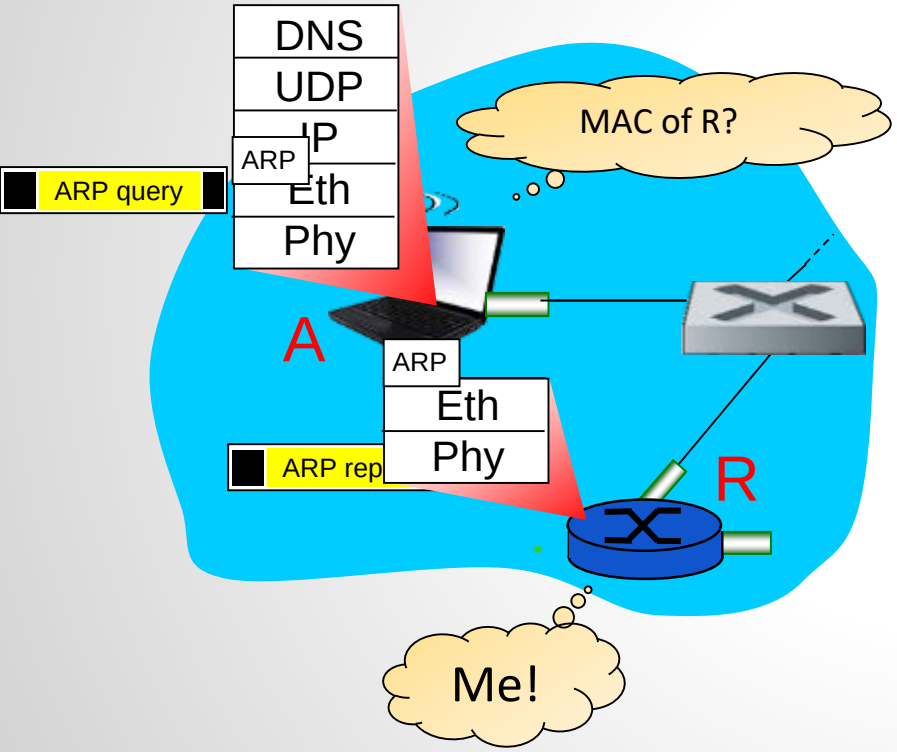


Recall: Routing to Another LAN

But what if host A does not know R's MAC address yet, but only the gateway R's IP address?



Given IP, find MAC!
The ARP Protocol



To learn R's MAC address:

Host A broadcasts ARP query containing R's IP address

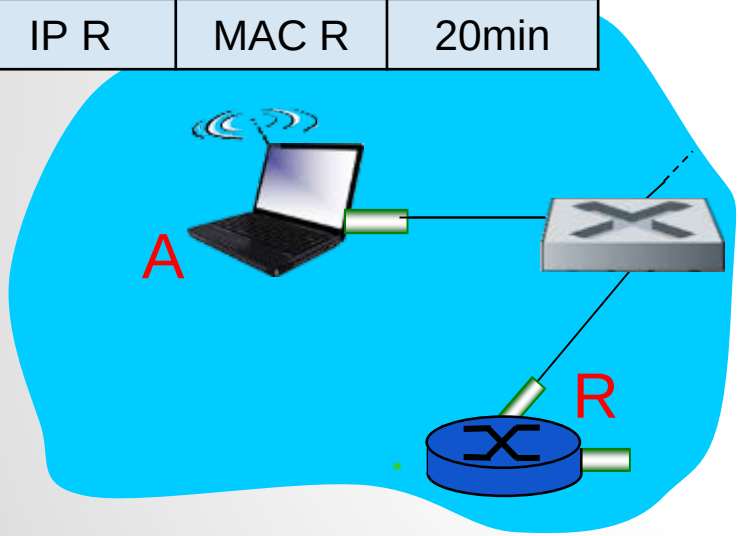
- ARP packet dstMAC = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

Router R receives ARP packet, replies to A with its (R's) MAC address

- Unicast to A's MAC address (why?)

Layer 2: About broadcasting and learning!

IP	MAC	TTL
IP R	MAC R	20min



The learned mappings are cached: each IP node (host, router) on LAN maintains ARP table:

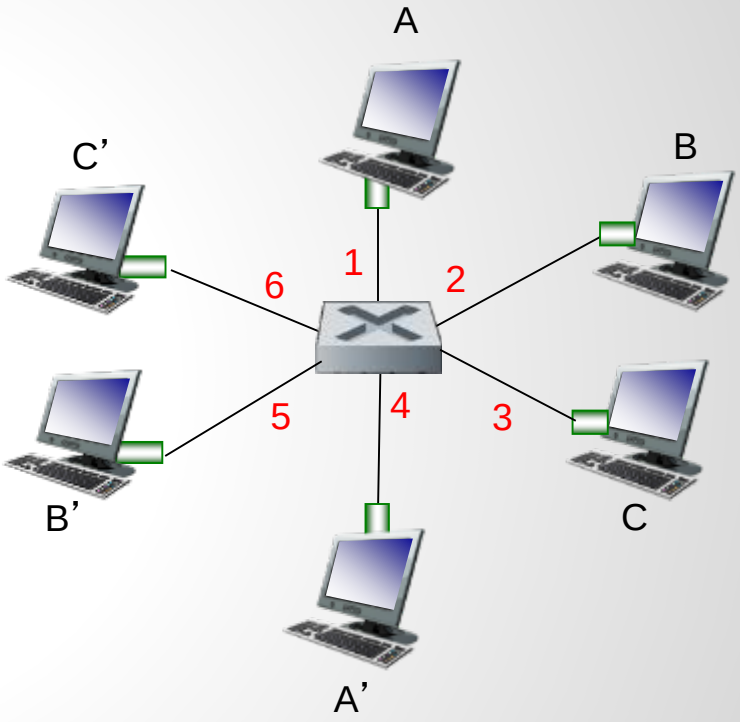
< IP address; MAC address; TTL>

Time-To-Live: avoid stale entries

ARP is "plug&play": nodes learn and create their ARP tables *without* intervention from administrator!

Also the switches broadcast and learn

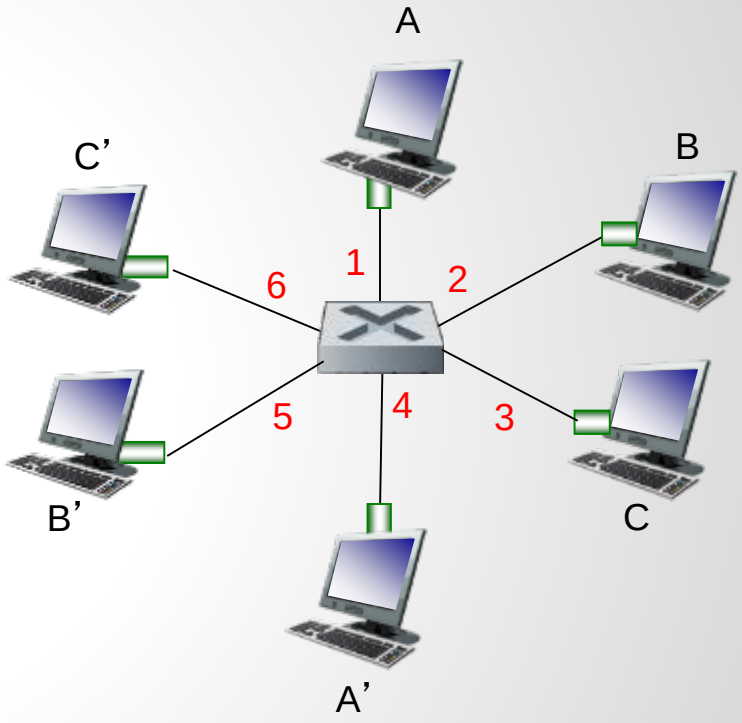
Another question: how does *a switch* know where to forward a frame? E.g., that A' is reachable via interface 4, B' reachable via interface 5?



switch with six interfaces (1,2,3,4,5,6)

Also the switches broadcast and learn

Another question: how does *a switch* know where to forward a frame? E.g., that A' is reachable via interface 4, B' reachable via interface 5?



switch with six interfaces (1,2,3,4,5,6)

MAC learning:

Each switch runs a MAC learning protocol and has a switch table:

Host's MAC address	Interface to reach host	TTL
A'	4	12min
B'	5	3min

But how can switches obtain this information?

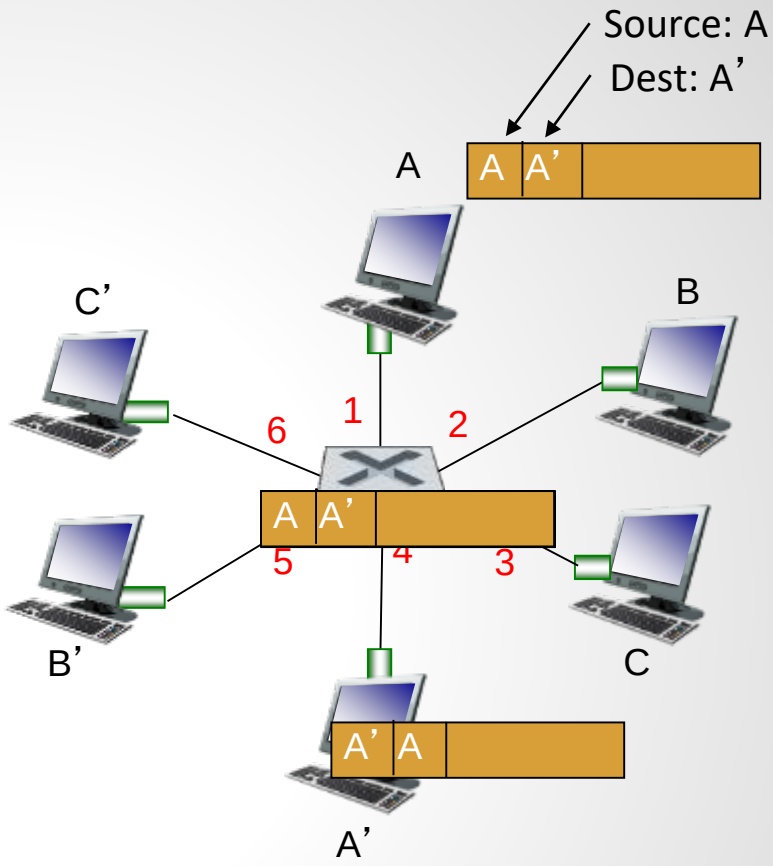
MAC Learning

- Destination A’ location *unknown*:

  flood but learn A

- Now destination A location known:

  send directly (and learn A’)



Host‘s MAC address	Interface to reach host	TTL
A	*1*	*10min*
A’	*4*	*10min*

Intermediate Conclusion

- Layer-2 networks are very flexible: location-independent addresses, plug&play, self-learning, etc.: devices (and virtual machines!) can move (migrate)

- But: Layer-2 networks do not scale: despite caching, LAN-wide broadcasts needed once in a while (ARP, MAC learning, DHCP, etc.)!

Intermediate Conclusion

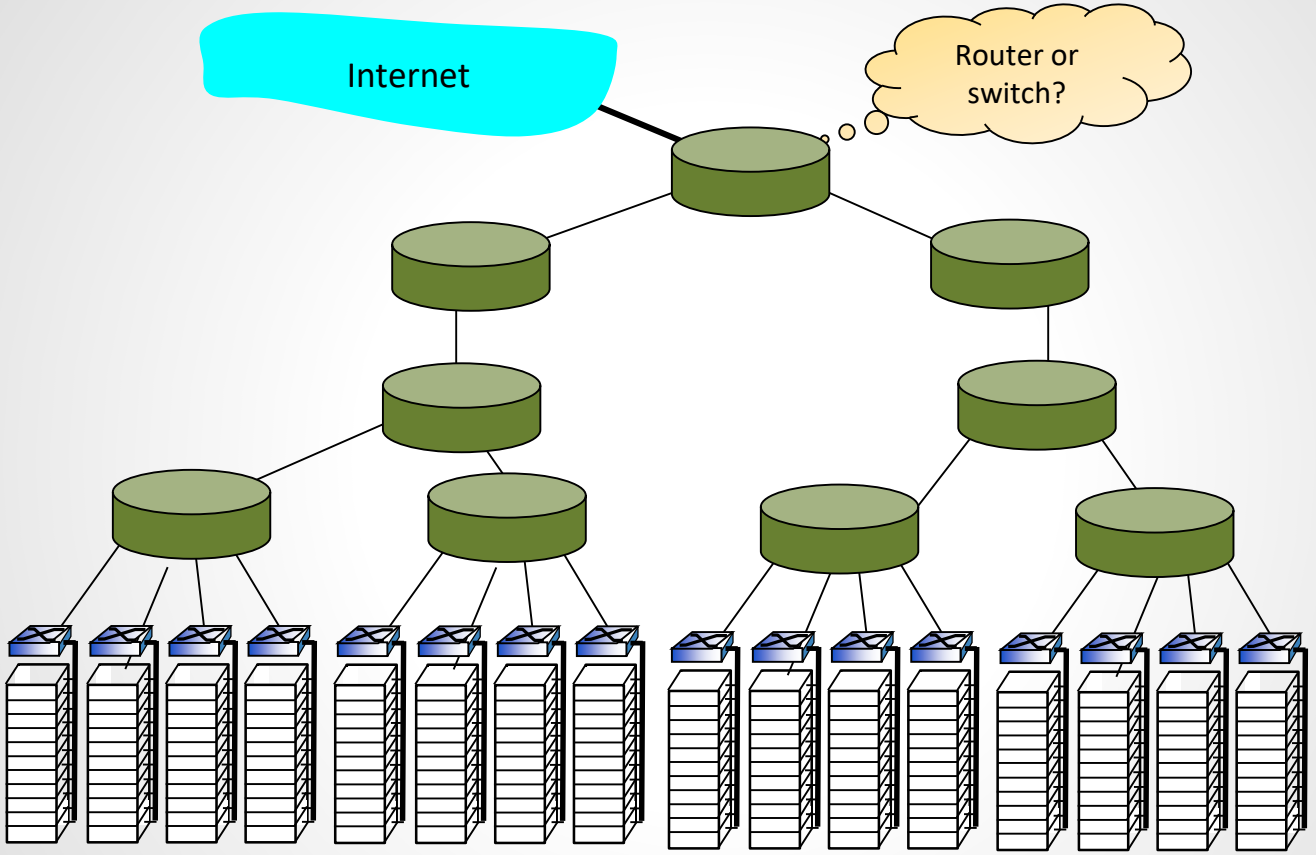
- Layer-2 networks are very flexible: location-independent addresses, plug&play, self-learning, etc.: devices (and virtual machines!) can move (migrate)

- But: Layer-2 networks do not scale: despite caching, LAN-wide broadcasts needed once in a while (ARP, MAC learning, DHCP, etc.)!

How large should a LAN be?

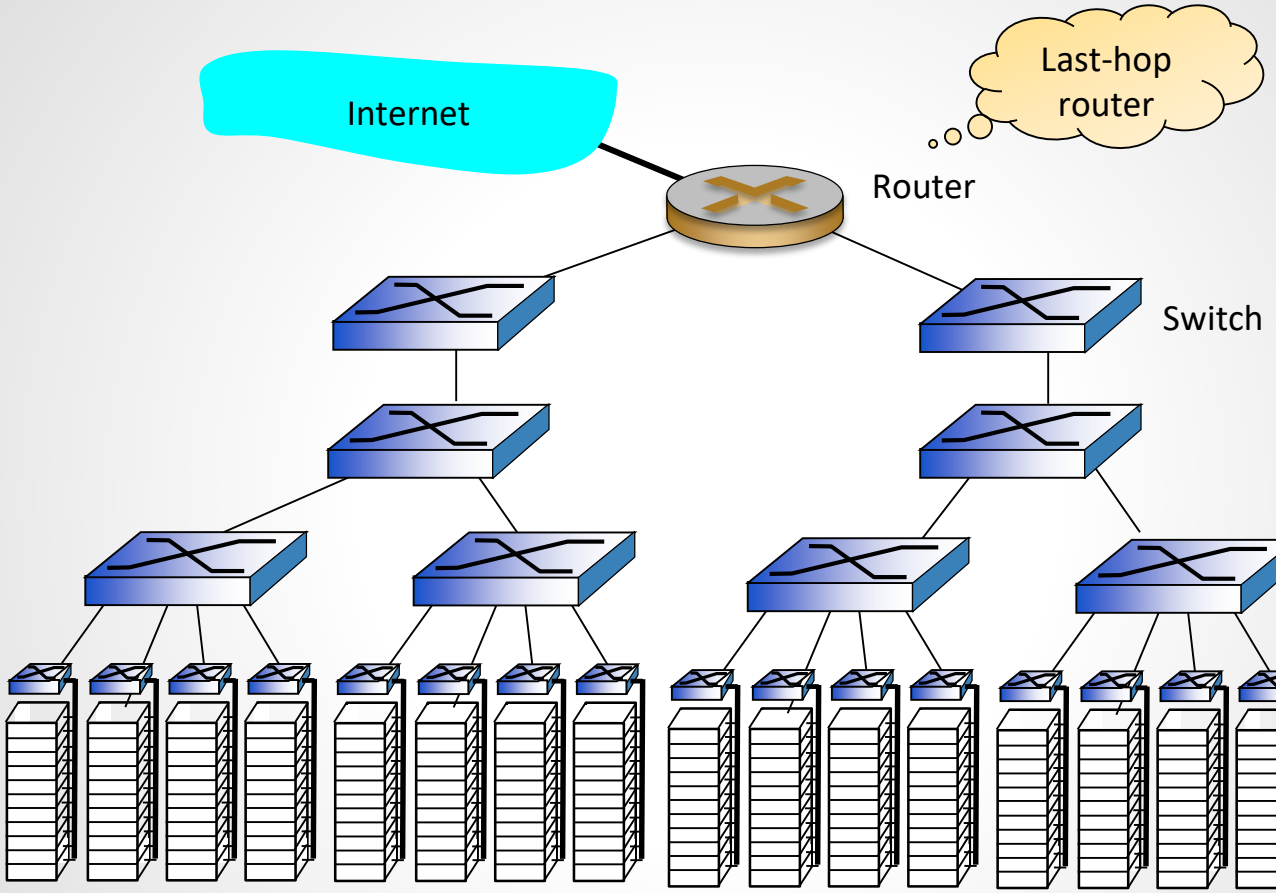
Flexibility vs Scalability tradeoff!

So how to design a datacenter?



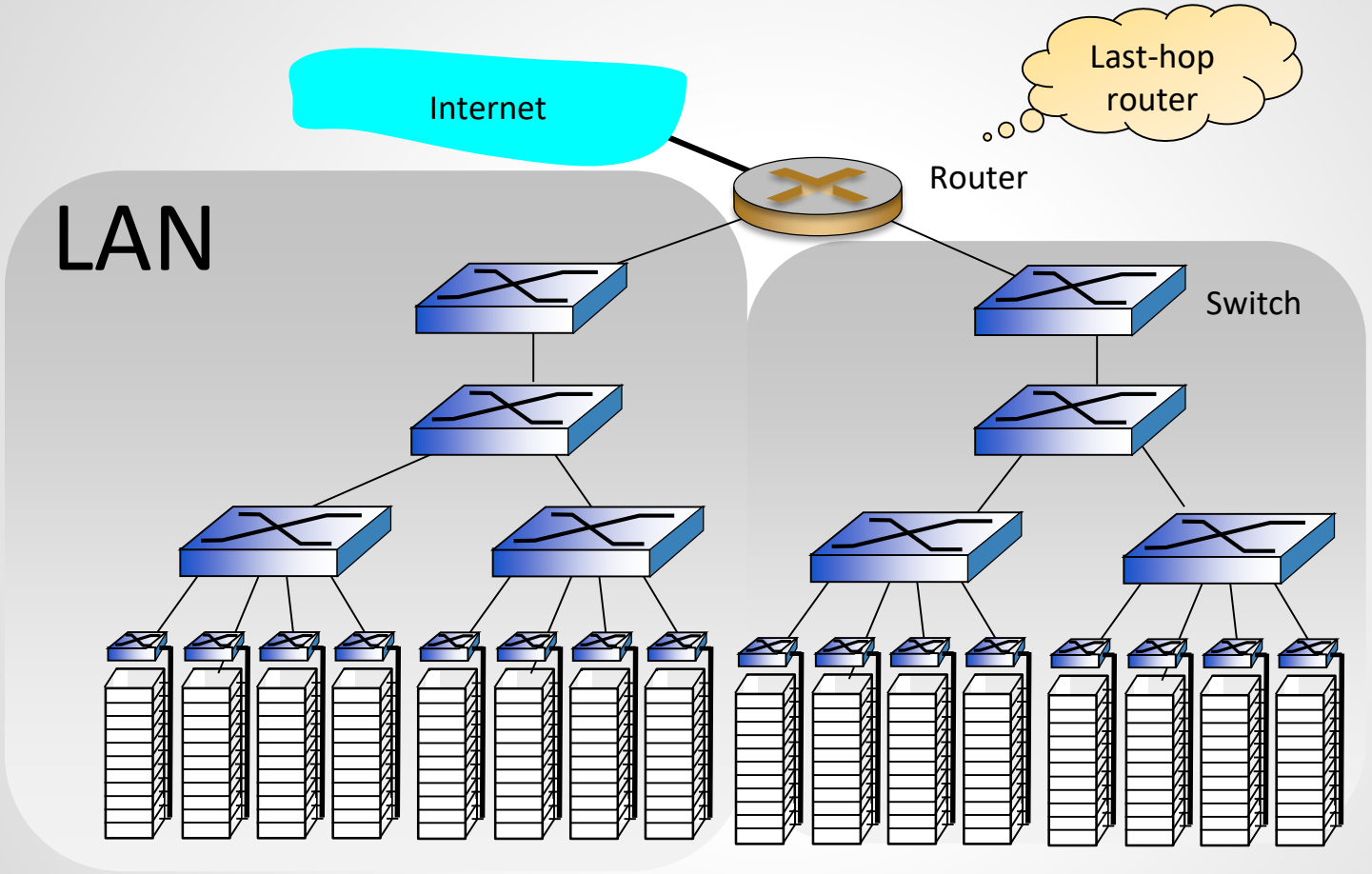
Racks of Servers

Proposal #1

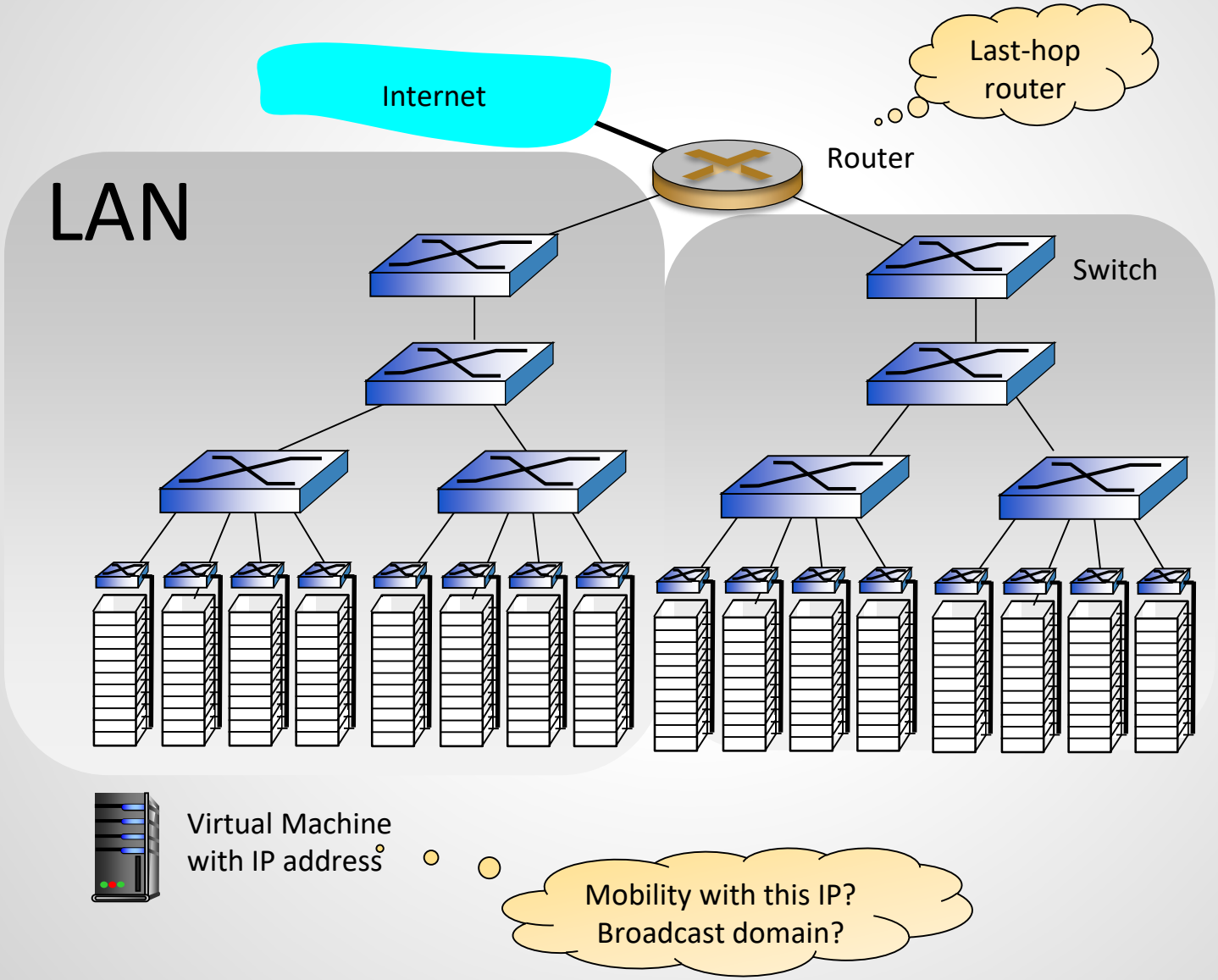


Where are LANs?

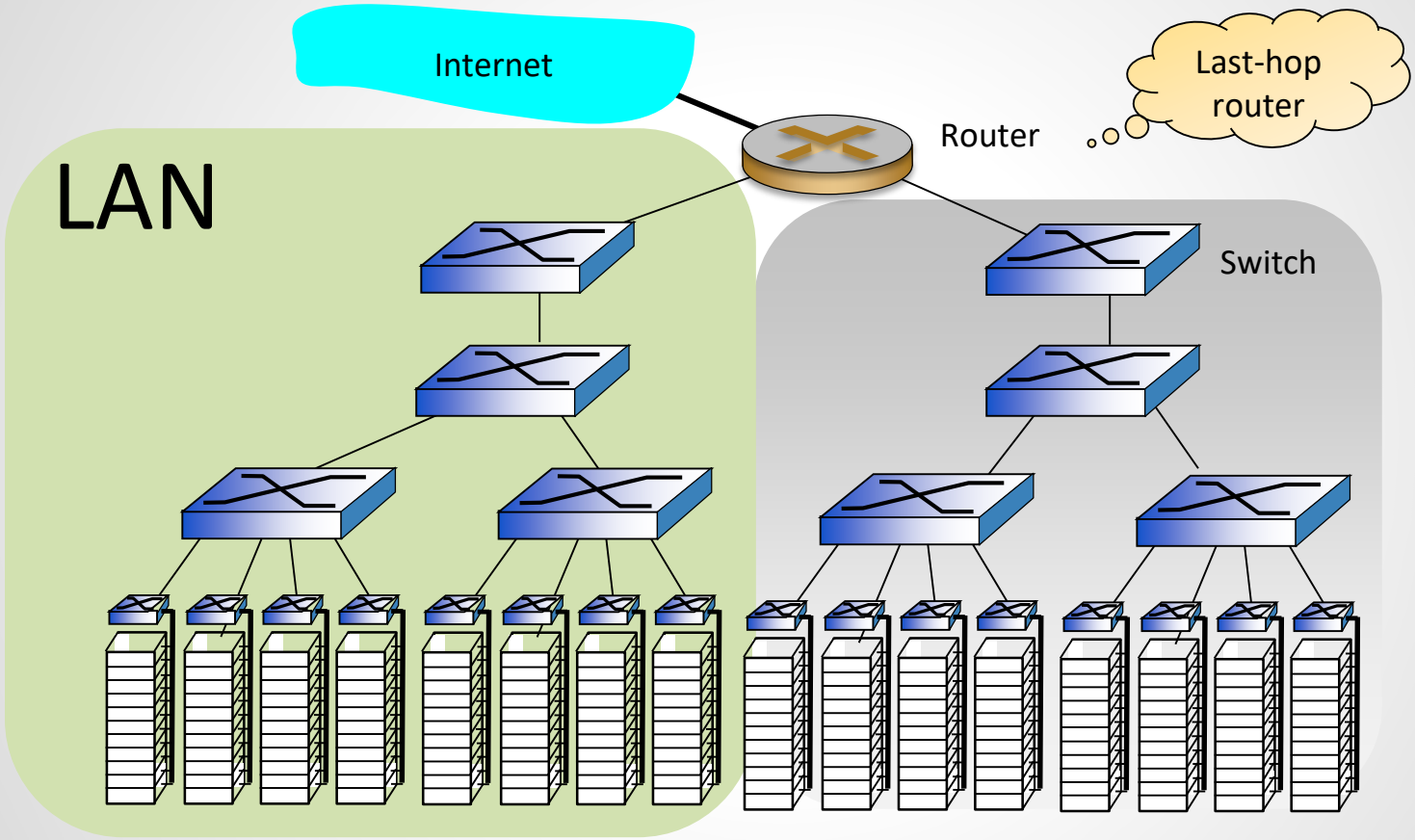
Proposal #1



Proposal #1


Last-hop router
Internet
Router
LAN
Switch
Virtual Machine with IP address
Mobility with this IP?
Broadcast domain?

Proposal #1

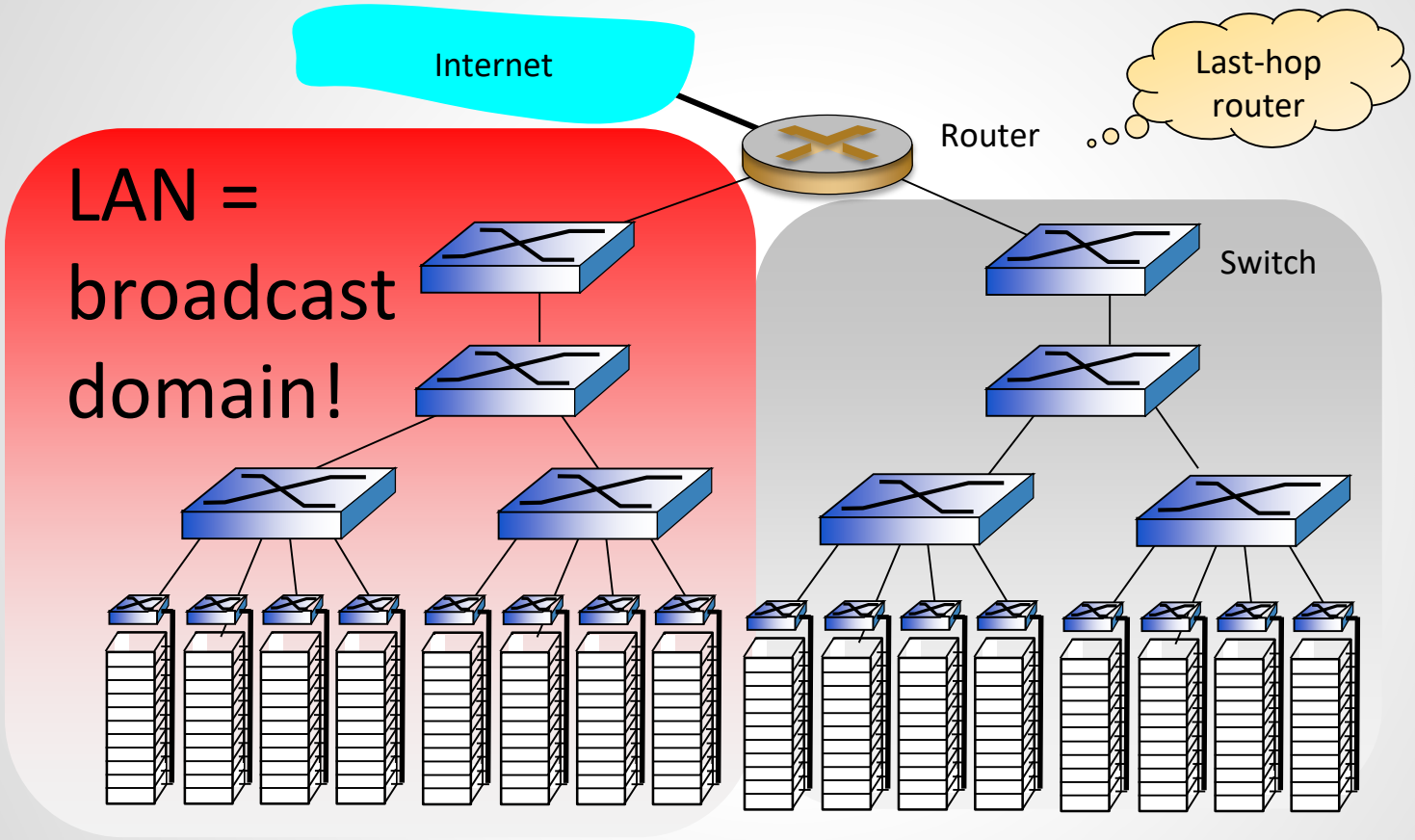


Virtual Machine with IP address

No need to change IP!

A large LAN: High mobility…

Proposal #1

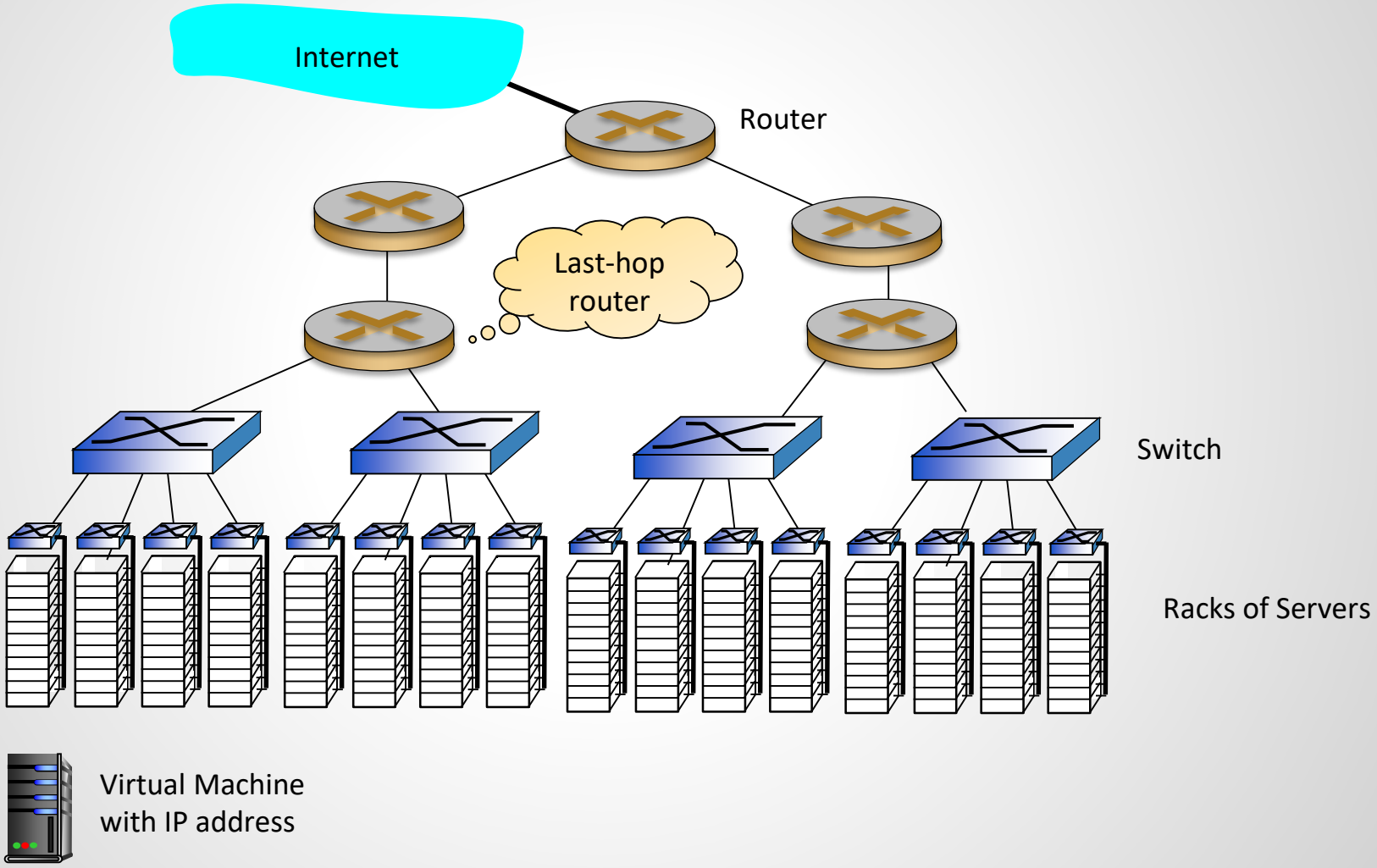


Virtual Machine with IP address

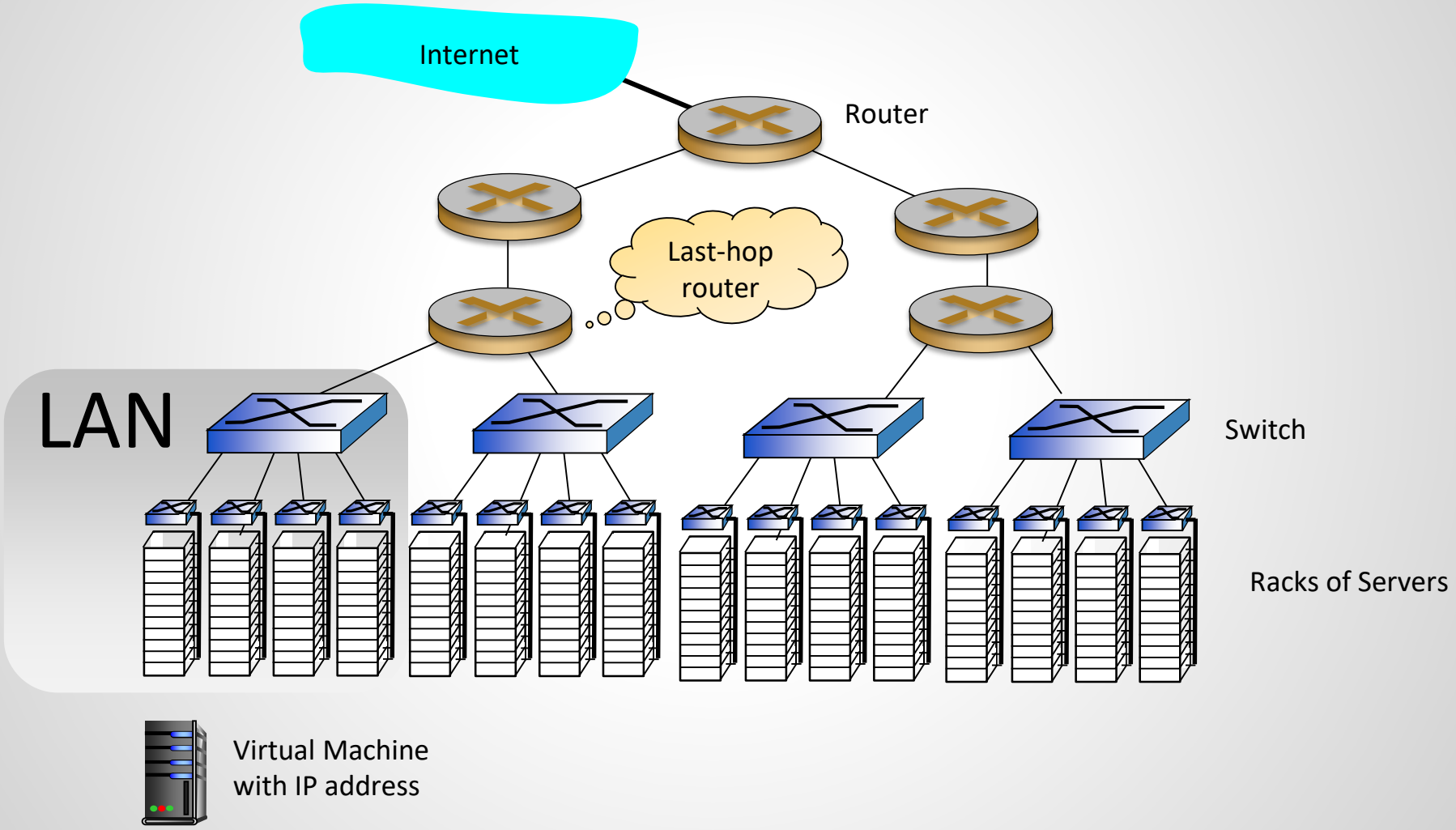
No need to change IP!

A large LAN: High mobility…
… but high overhead due to learning and broadcasting.

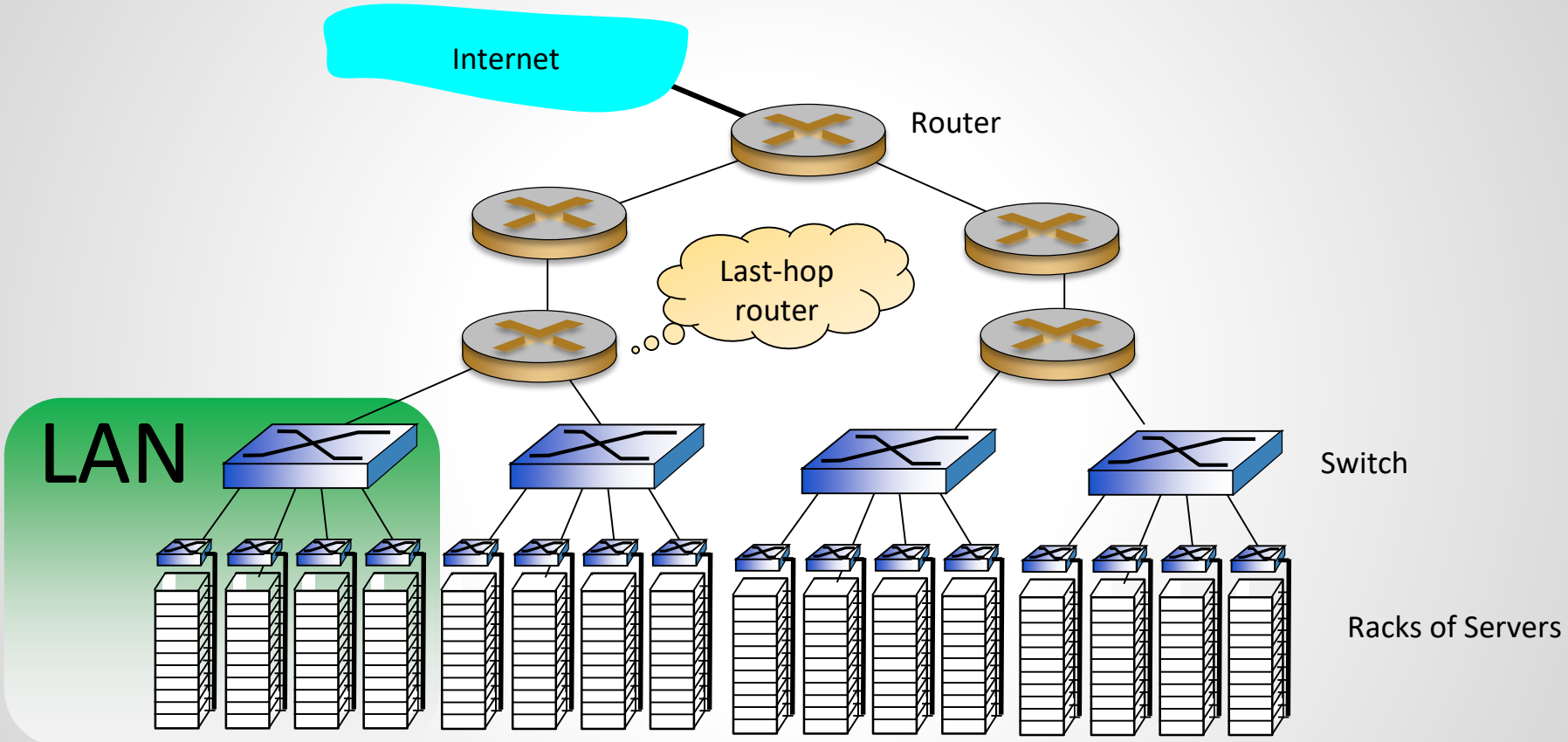
Proposal #2

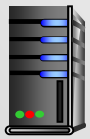

Internet
Router
Last-hop router
Switch
Racks of Servers
Virtual Machine with IP address

Proposal #2


Internet
Router
Last-hop router
LAN
Switch
Racks of Servers
Virtual Machine with IP address

Proposal #2



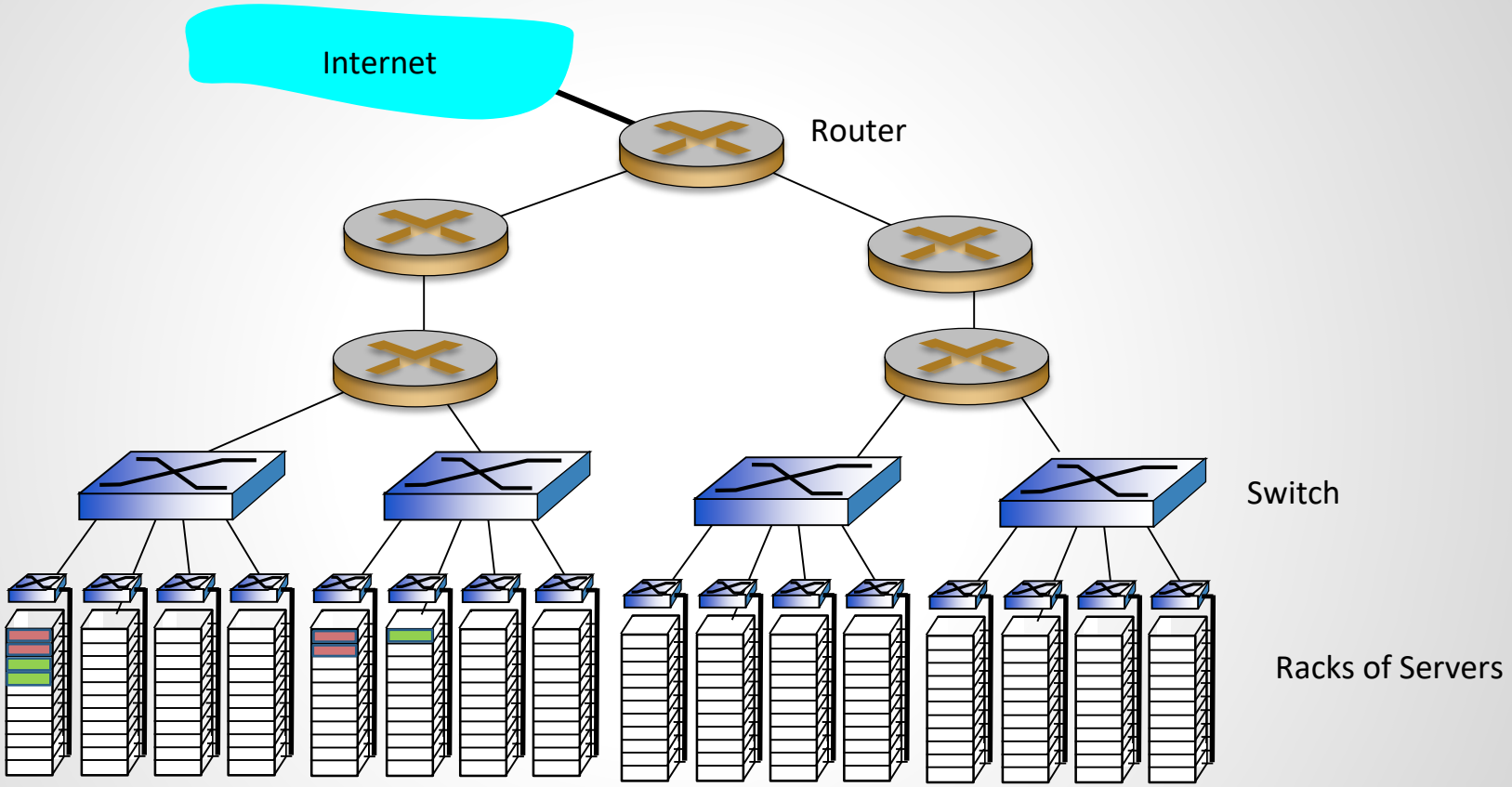


Virtual Machine with IP address

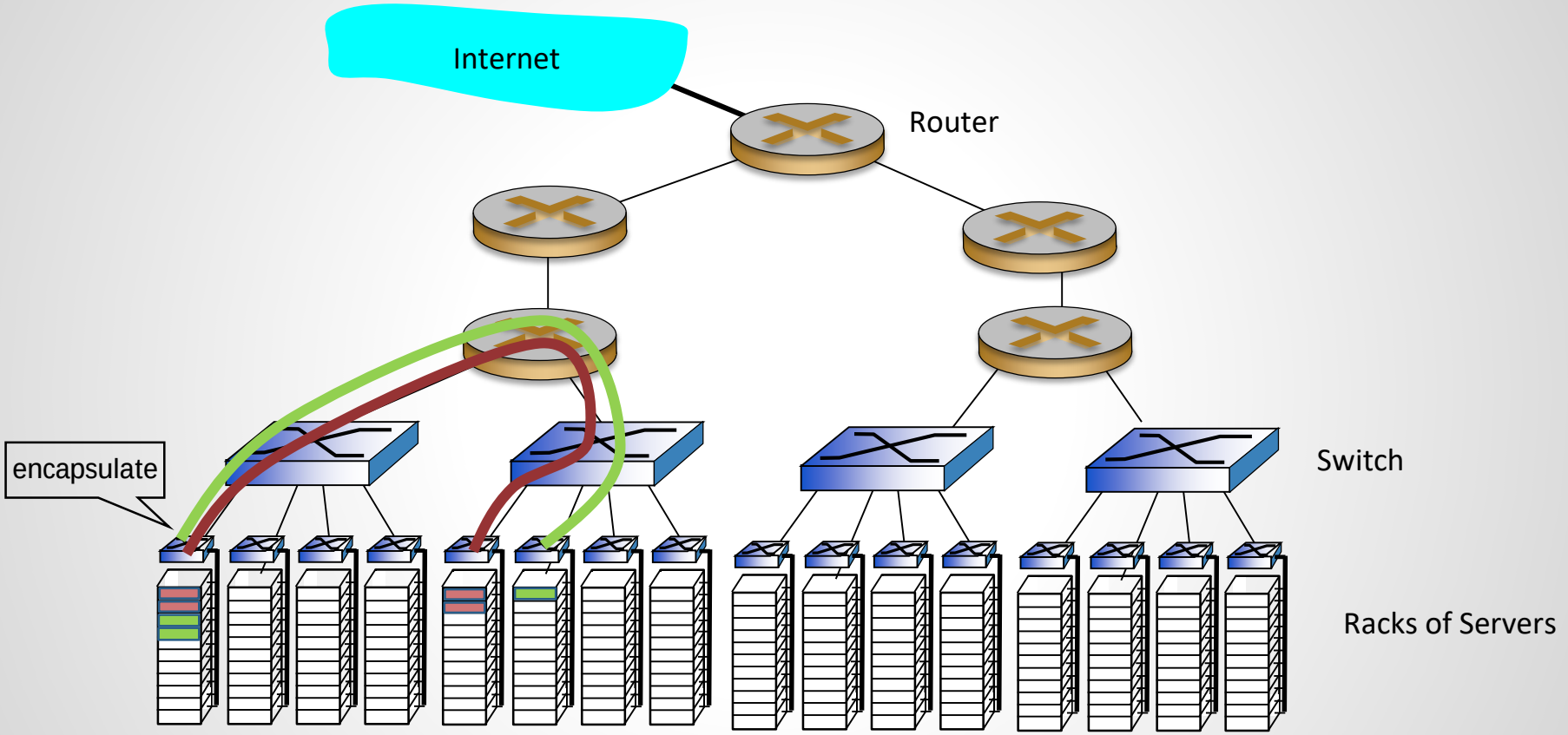
Mobility

A small LAN: A different mobility – overhead (scalability) tradeoff!

What about isolation of tenants?

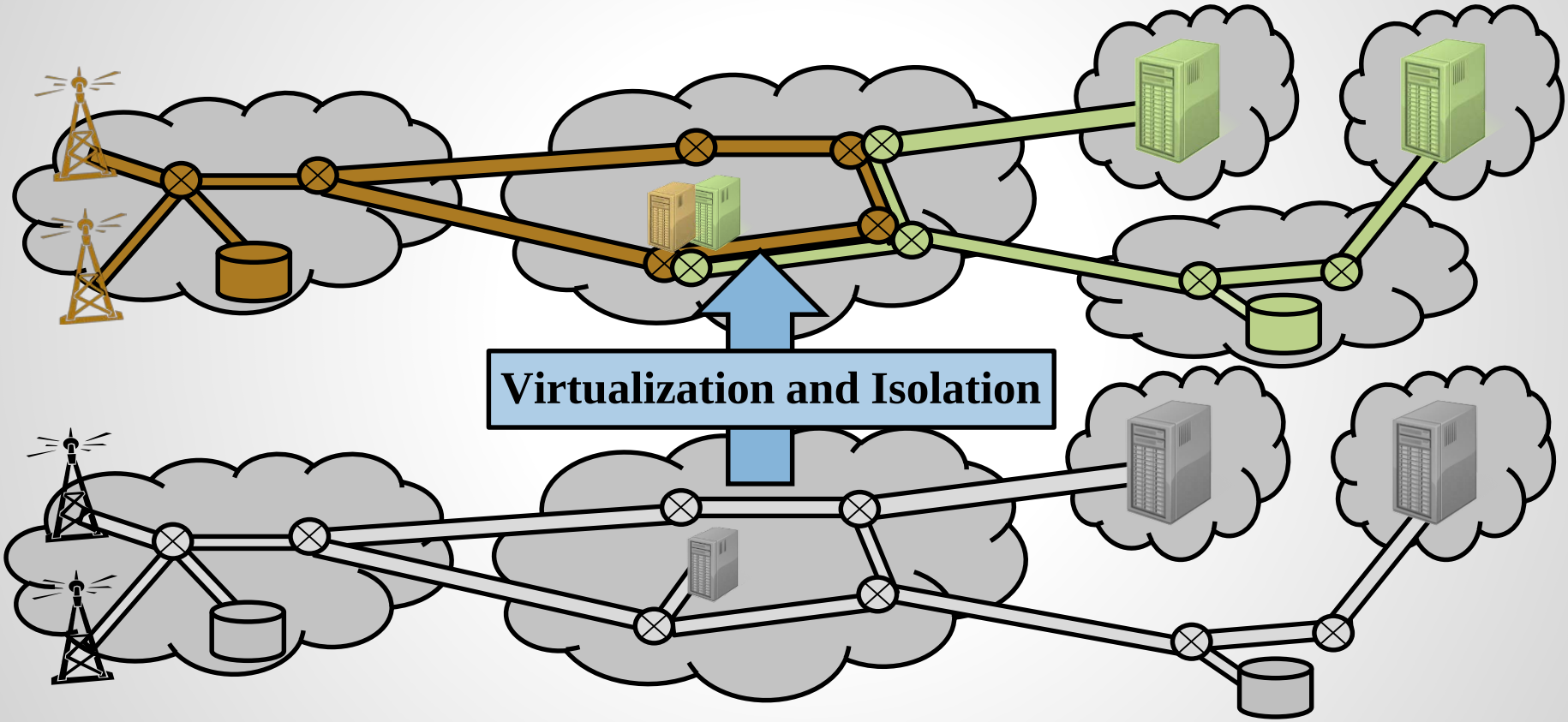


What about isolation of tenants?



Network virtualization: VLANs, VxLANs, tunneling, … *or SDN*!

Isolation: Required *on all levels*!



Roadmap

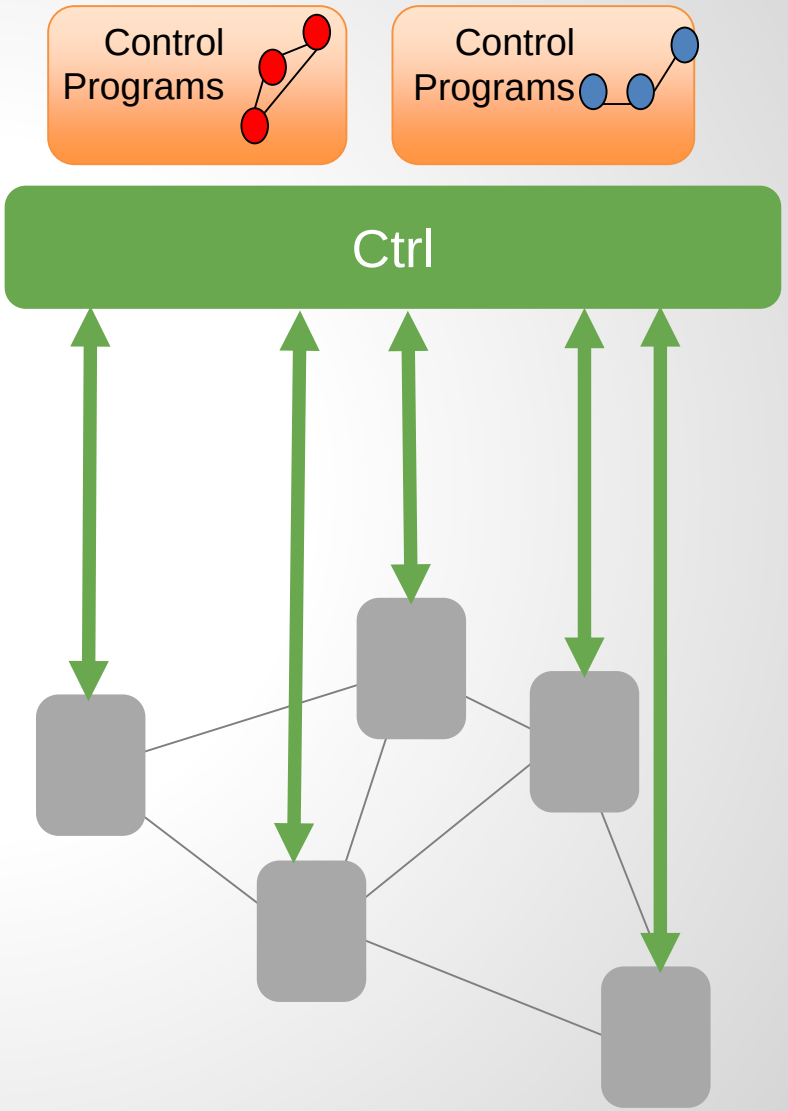
- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

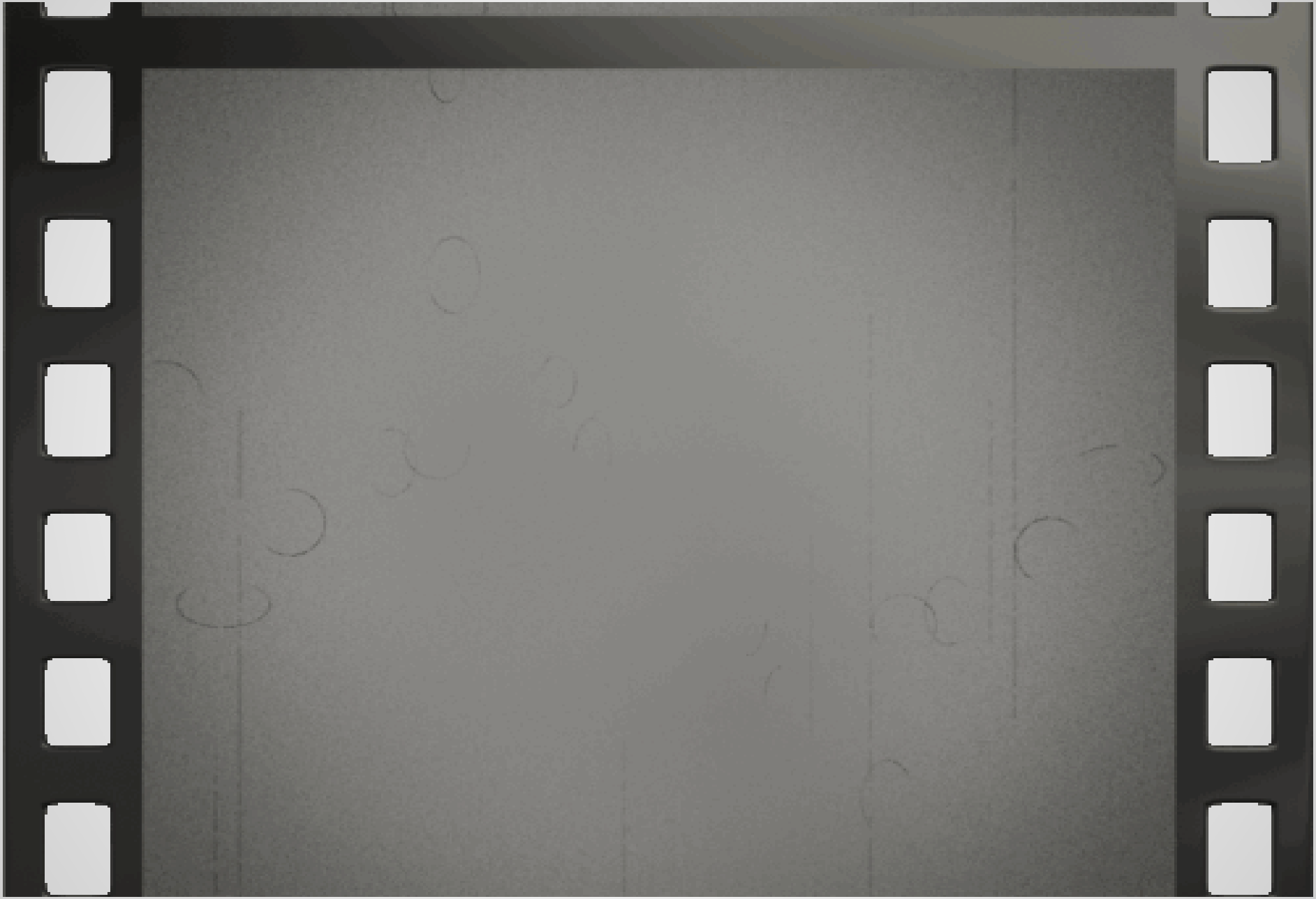
Roadmap

- A short recap:
  - Networking preliminaries
  - **Opportunities and challenges of SDNs**
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

SDN in a Nutshell

- Logically centralized control
- Innovation at speed of software development
- Direct control over paths
- Generalized match-action
- Example benefits:
  - Flexible network virtualization
  - Traffic engineering

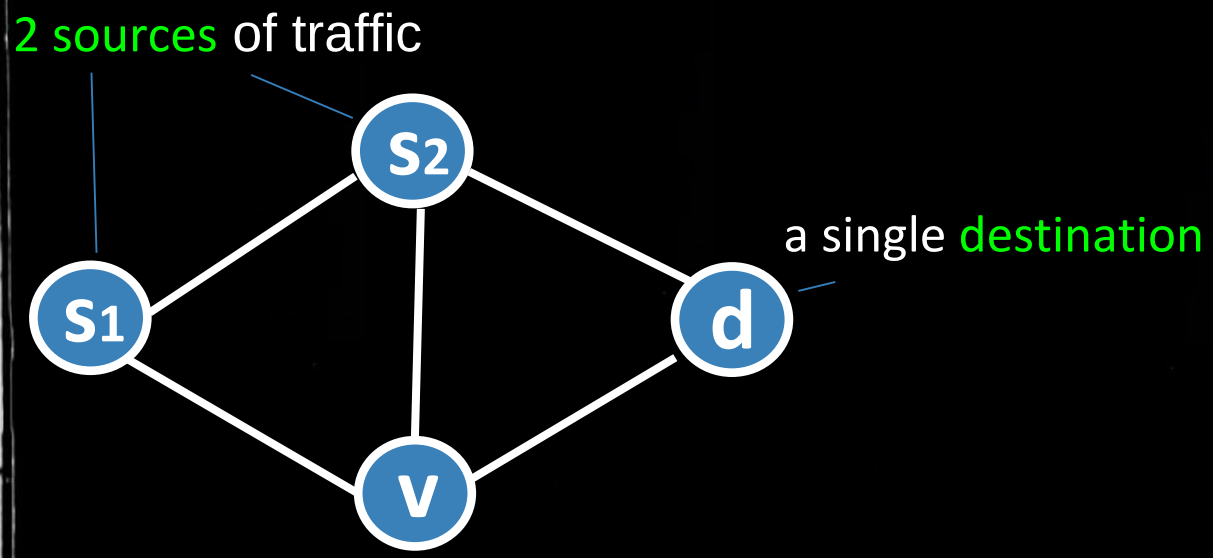




Slide credit:
Pedro Casas

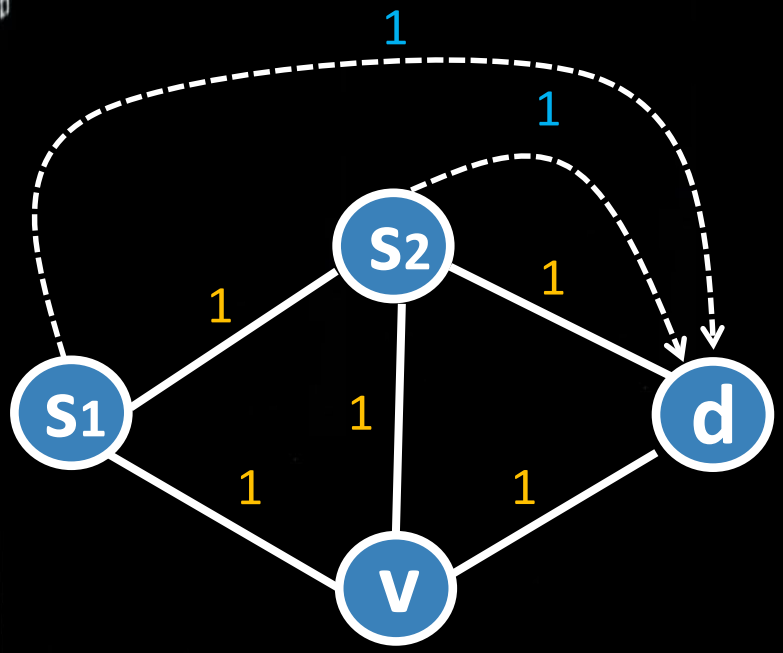
Traditional Traffic Engineering

Traditional Traffic Engineering



Kudos to: Pedro Casas

Traditional Traffic Engineering



all link capacities 1

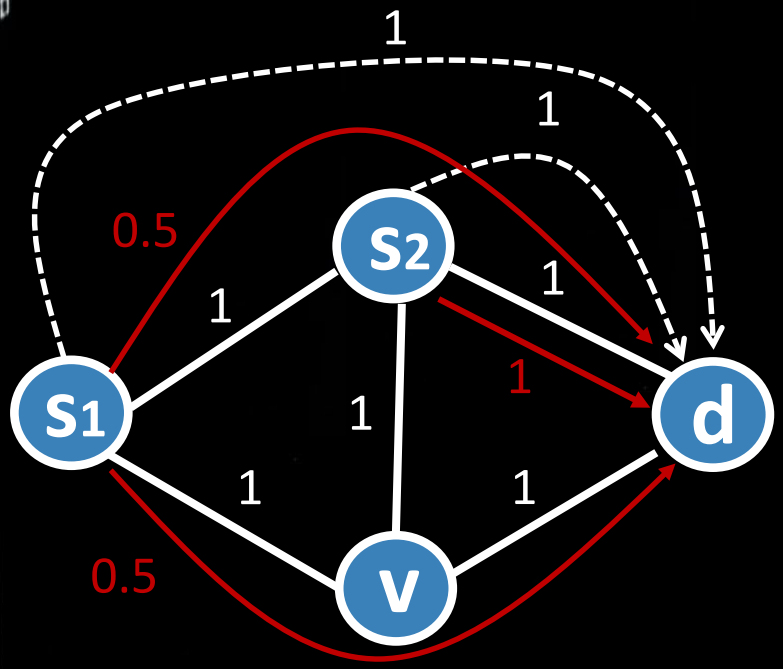
Only 2 possible demands:

1. s1 ▻ d = 1
2. s2 ▻ d = 1

- E.g., OSPF
- Indirectly: changing weights

Kudos to: Pedro Casas

Traditional Traffic Engineering



all link capacities 1

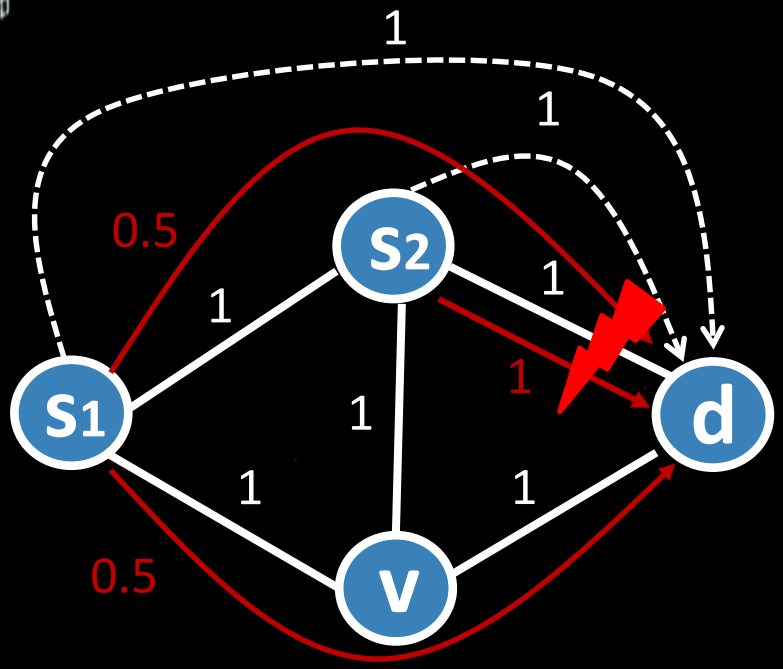
Only 2 possible demands:

1. s1 ▻ d = 1
2. s2 ▻ d = 1

- E.g., OSPF
- Indirectly: changing weights, equal splits

Kudos to: Pedro Casas

Traditional Traffic Engineering



all link capacities 1

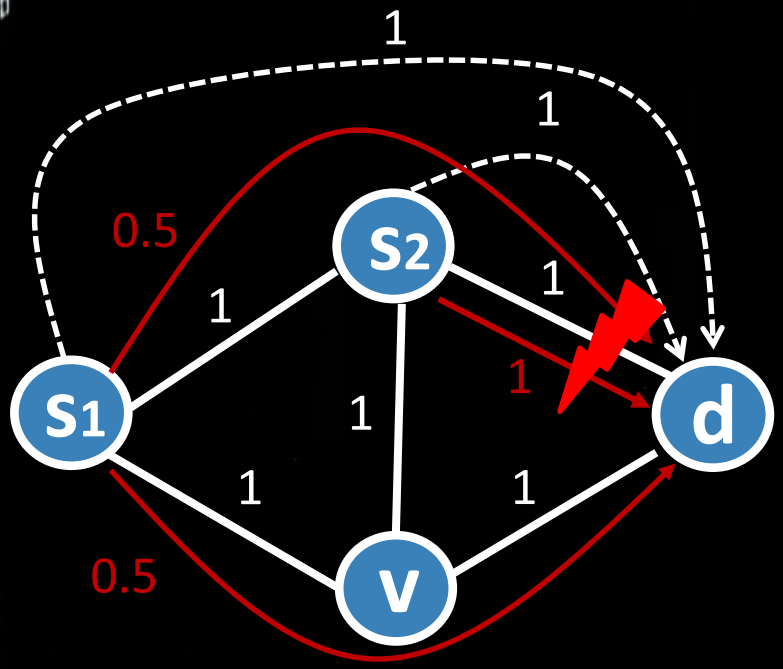
Only 2 possible demands:

1. s1 ▻ d = 1
2. s2 ▻ d = 1

- E.g., OSPF
- Indirectly: changing weights, equal splits

Kudos to: Pedro Casas

Traditional Traffic Engineering



all link capacities 1

Only 2 possible demands:

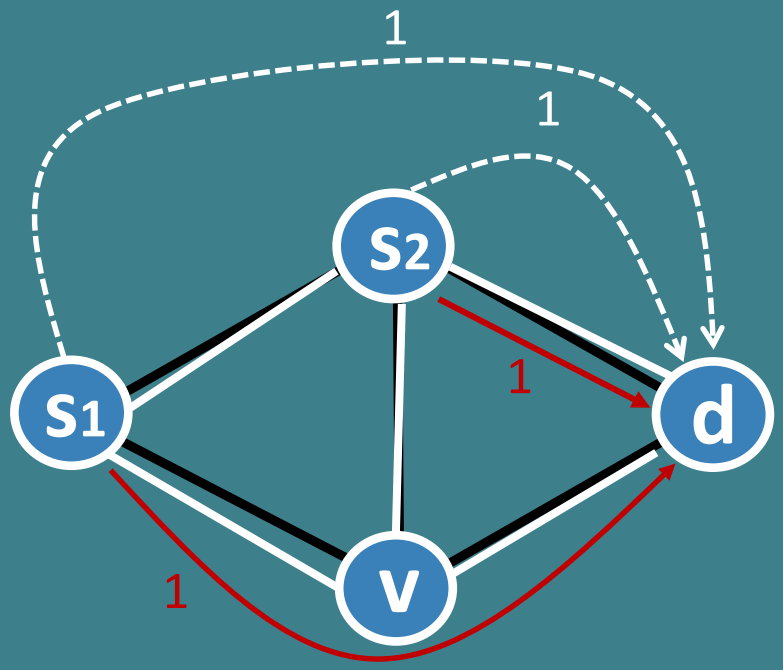
1. s1 ▻ d = 1
2. s2 ▻ d = 1

No link-weight assignment can attain ≤ 100% link utilization!

- E.g., OSPF
- Indirectly: changing weights, equal splits

Kudos to: Pedro Casas

Easy in SDN!



Only 2 possible demands:

1. s1 ▻ d = 1
2. s2 ▻ d = 1

SDN: Not necessarily simpler!

Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

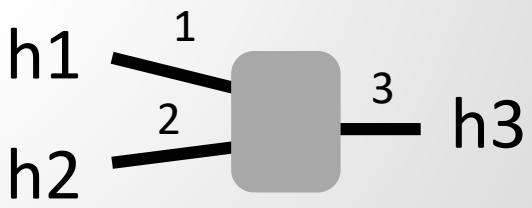
Credits: Jennifer Rexford

Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2:

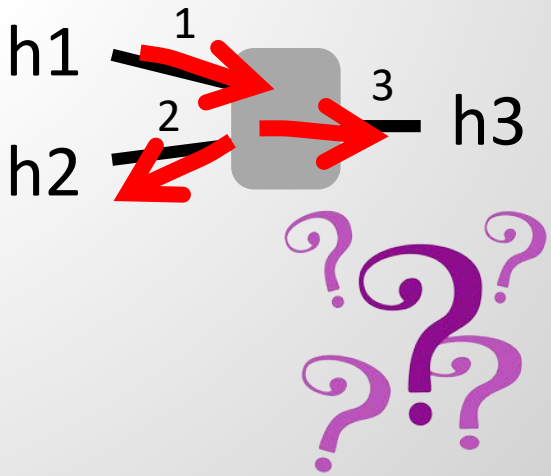


Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2: **flood**

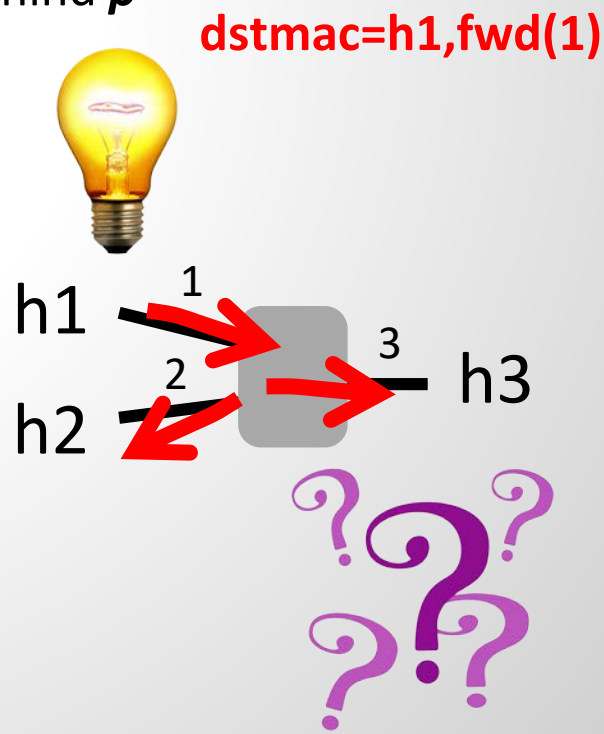


Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**

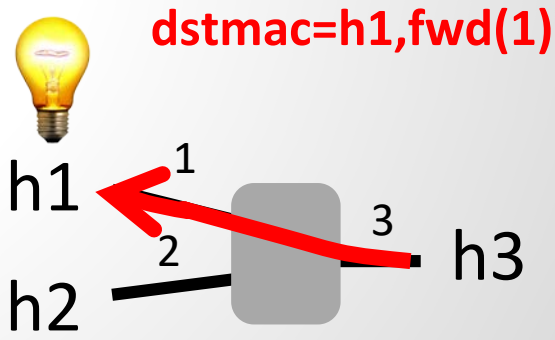


Recall: Networking 101

- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**
  - h3 sends to h1: **forward to p1**



Recall: Networking 101

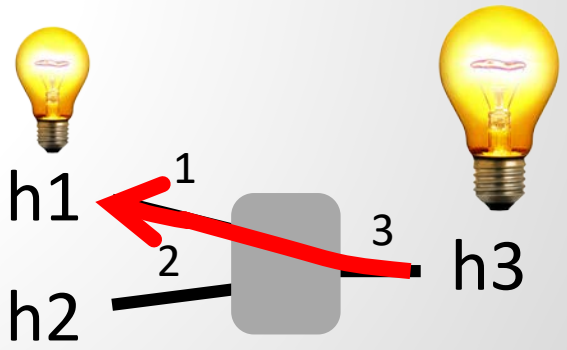
- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

dstmac=h1,fwd(1)

dstmac=h3,fwd(3)

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**
  - h3 sends to h1: **forward to p1, learn (h3,p3)**



Recall: Networking 101

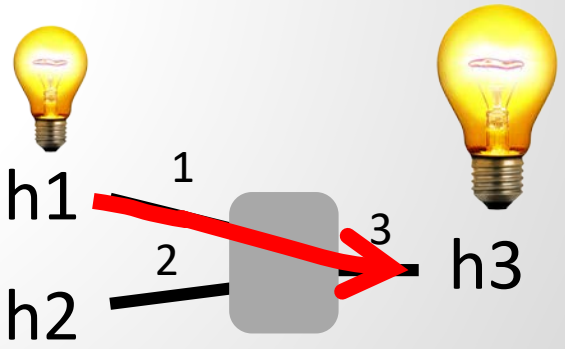
- Networking «Hello World»: MAC learning

- Principle: for packet (***src***,***dst***) arriving at port ***p***
  - If ***dst*** unknown: broadcast packets to all ports
    - Otherwise forward directly to known port
  - Also: if ***src*** unknown, switch learns: ***src*** is behind ***p***

dstmac=h1,fwd(1)

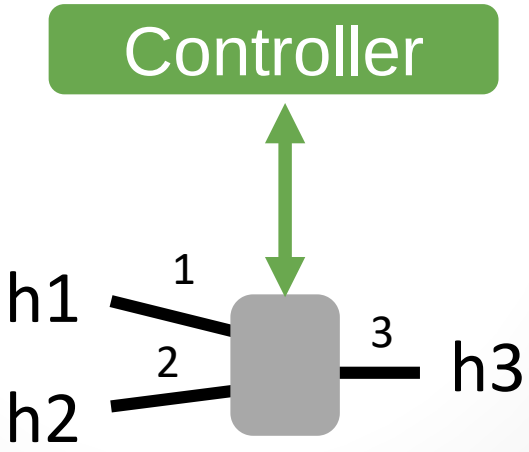
dstmac=h3,fwd(3)

- Example
  - h1 sends to h2: **flood, learn (h1,p1)**
  - h3 sends to h1: **forward to p1, learn (h3,p3)**
  - h1 sends to h3: **forward to p3**

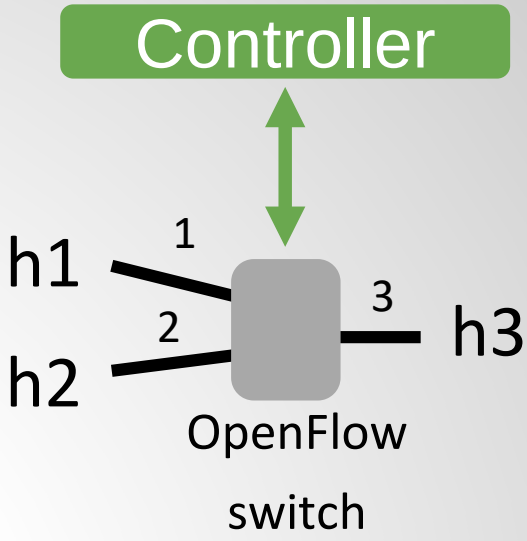


From Traditional Networks to SDN

How to implement this behavior in SDN?



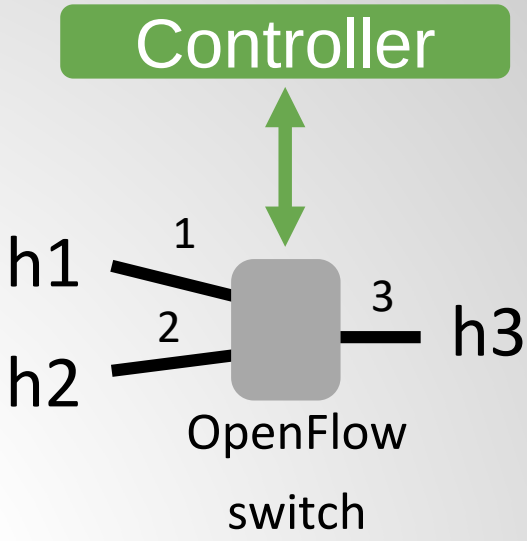
Example: SDN MAC Learning Done Wrong



- Initial table: Send everything to controller

Pattern	Action
*	send to controller

Example: SDN MAC Learning Done Wrong

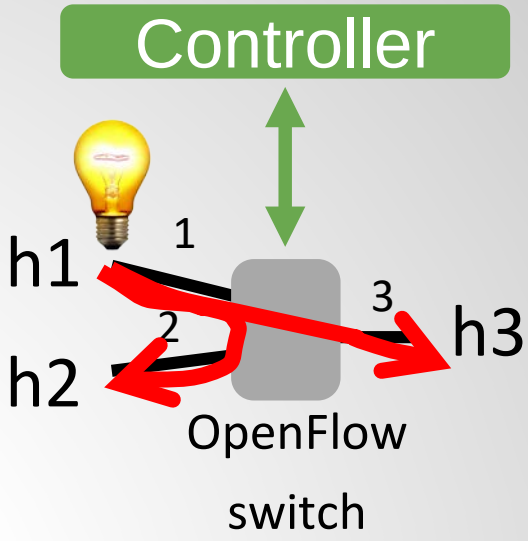


- Initial table: Send everything to controller

Pattern	Action
*	send to controller

- When h1 sends to h2:

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if destination unknown

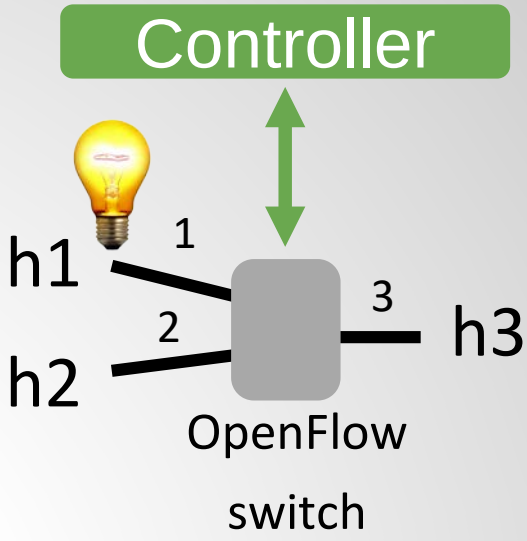
Pattern	Action
*	send to controller

h1 sends to h2

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

- When h1 sends to h2:
  - Controller learns that h1@p1, updates table, and floods

Example: SDN MAC Learning Done Wrong

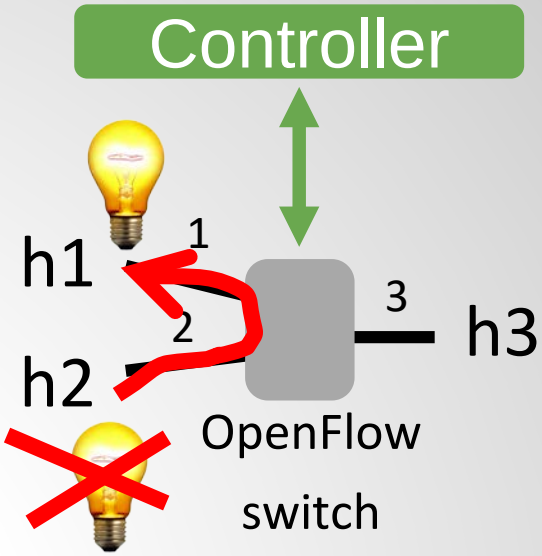


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

- Now assume h2 sends to h1:

Example: SDN MAC Learning Done Wrong

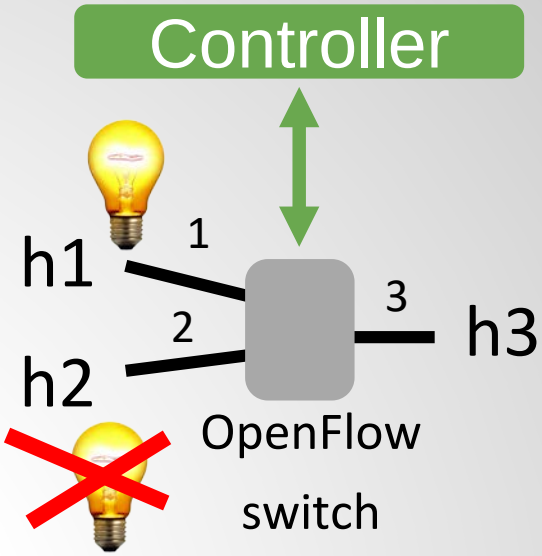


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

- Now assume h2 sends to h1:
  - *Switch* knows destination: message forwarded to h1
  - ***BUT***: No controller interaction, does not learn about h2: no new rule for h2

Example: SDN MAC Learning Done Wrong



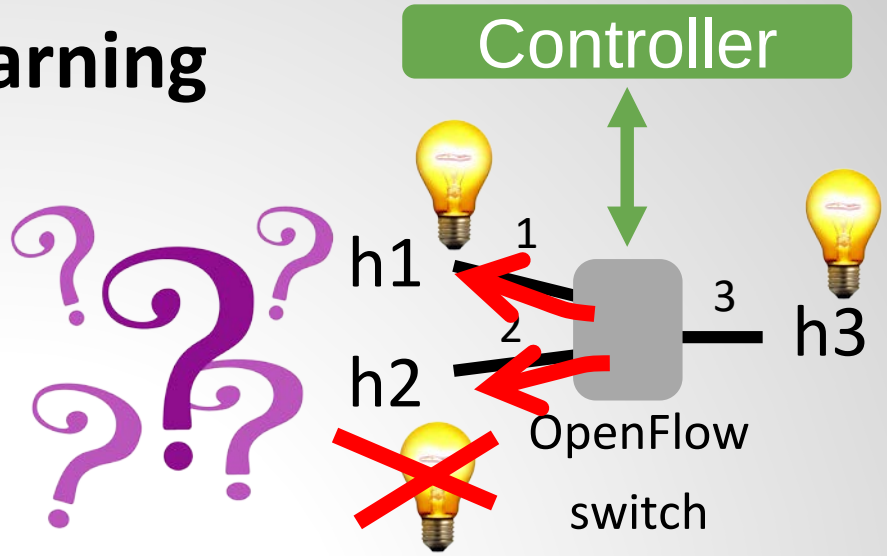
- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

h3 sends to h2

- Now, when h3 sends to h2:

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if destination unknown

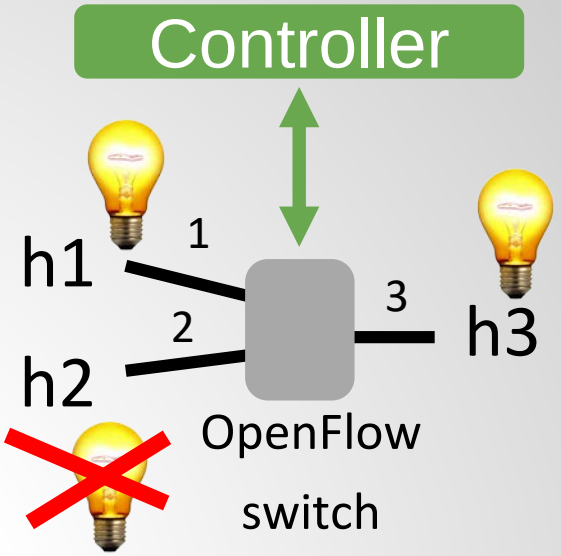
Pattern	Action
dstmac=h1	Forward(1)
*	send to controller

h3 sends to h2

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

- Now, when h3 sends to h2:
  - Dest unknown: goes to controller which learns about h3
  - And then floods

Example: SDN MAC Learning Done Wrong

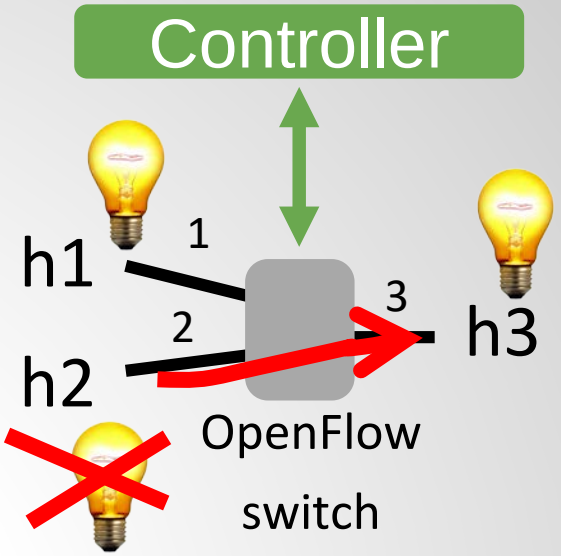


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

- Now, if h2 sends to h3 or h1:

Example: SDN MAC Learning Done Wrong

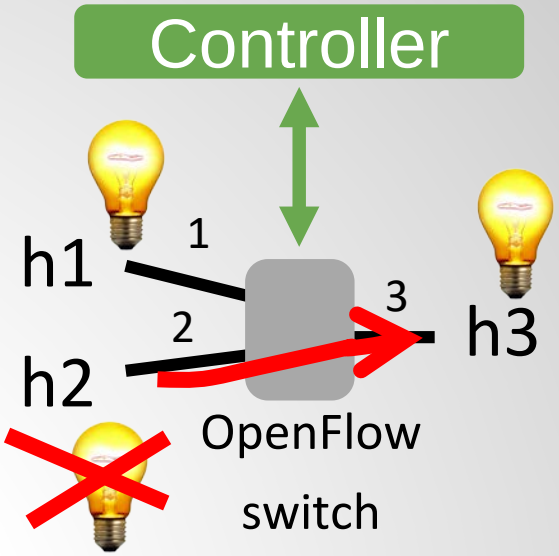


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

- Now, if h2 sends to h3 or h1:
  - Destinations known: controller does not learn about h2

Example: SDN MAC Learning Done Wrong

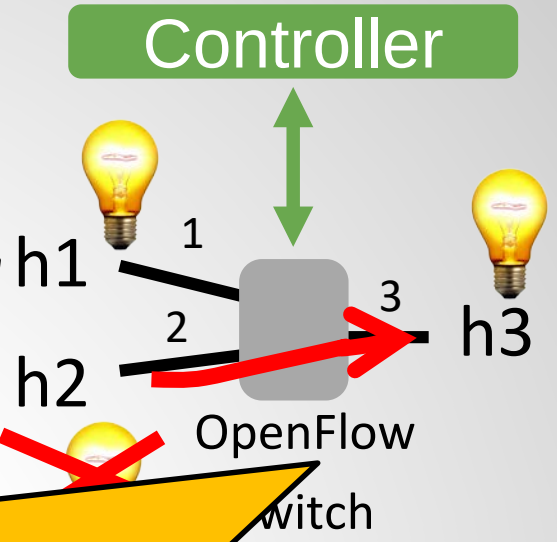


- Principle: only send to ctrl if destination unknown

Pattern	Action
dstmac=h3	Forward(3)
dstmac=h1	Forward(1)
*	send to controller

Ouch! Controller cannot learn about h2 anymore: whenever h2 is source, destination is known. All future requests to h2 will ***all be flooded***: inefficient!

Example: SDN MAC Learning Done Wrong



- Principle: only send to ctrl if d[illegible]ination unkno[illegible]

How to efficiently detect such problems? And which rules to use to overcome them? An algorithmic problem!

Ouch! Controller c[illegible]ot learn[illegible]out h2 anymore: whenever h2 is sourc[illegible] destination is known. All future requests to h2 will ***all be flooded***: inefficient!

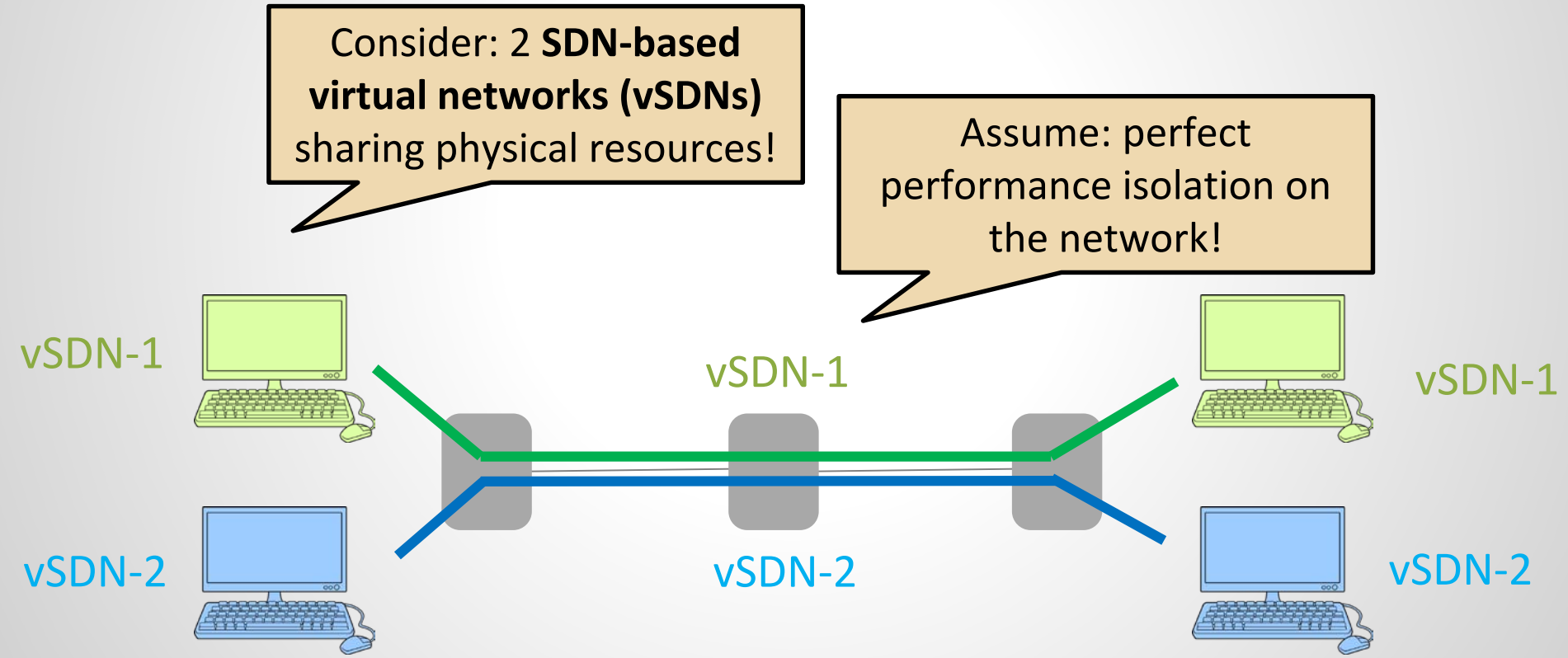
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

Roadmap

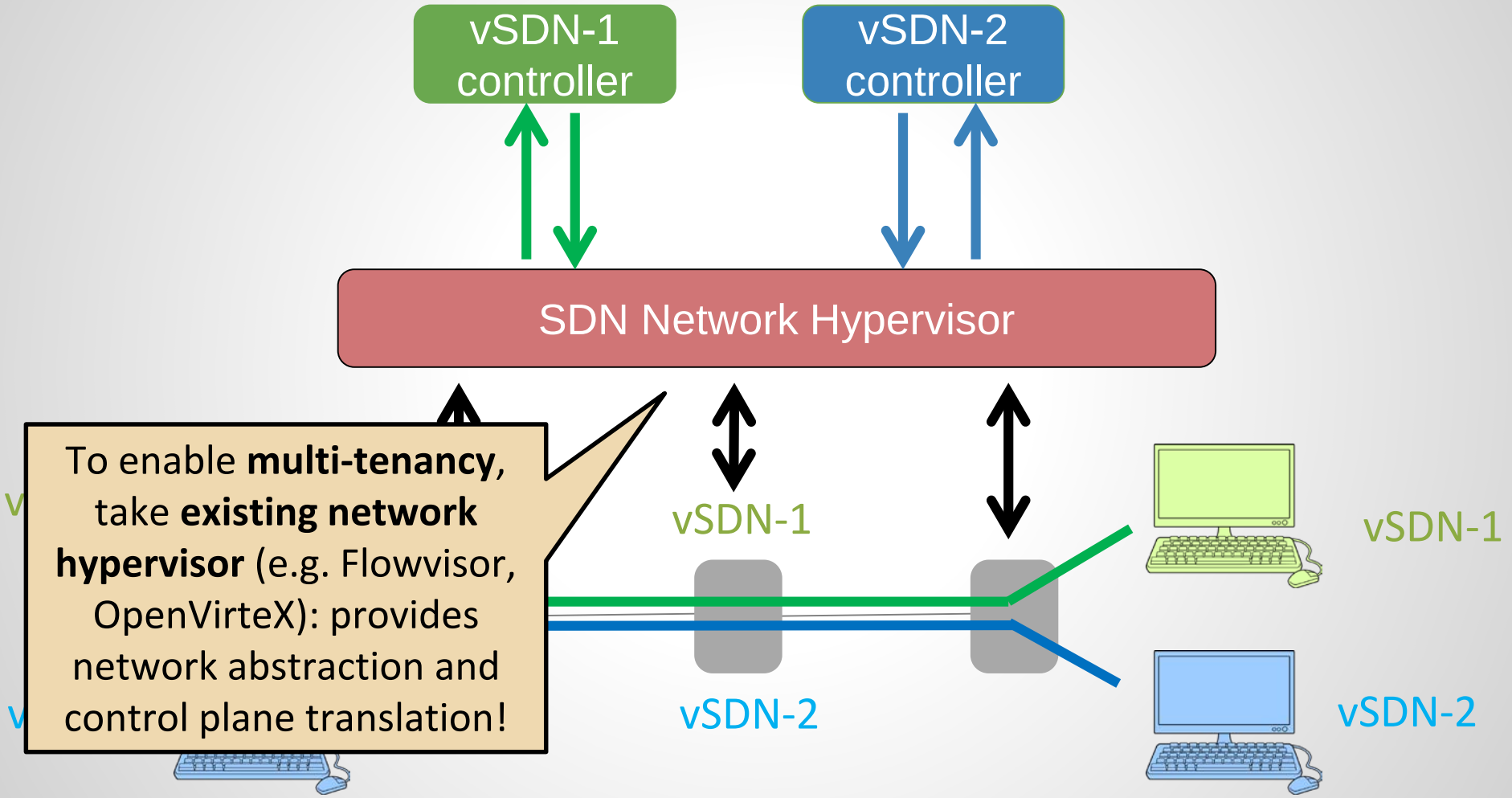
- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - **On predictable performance**
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

On Predictable Performance…



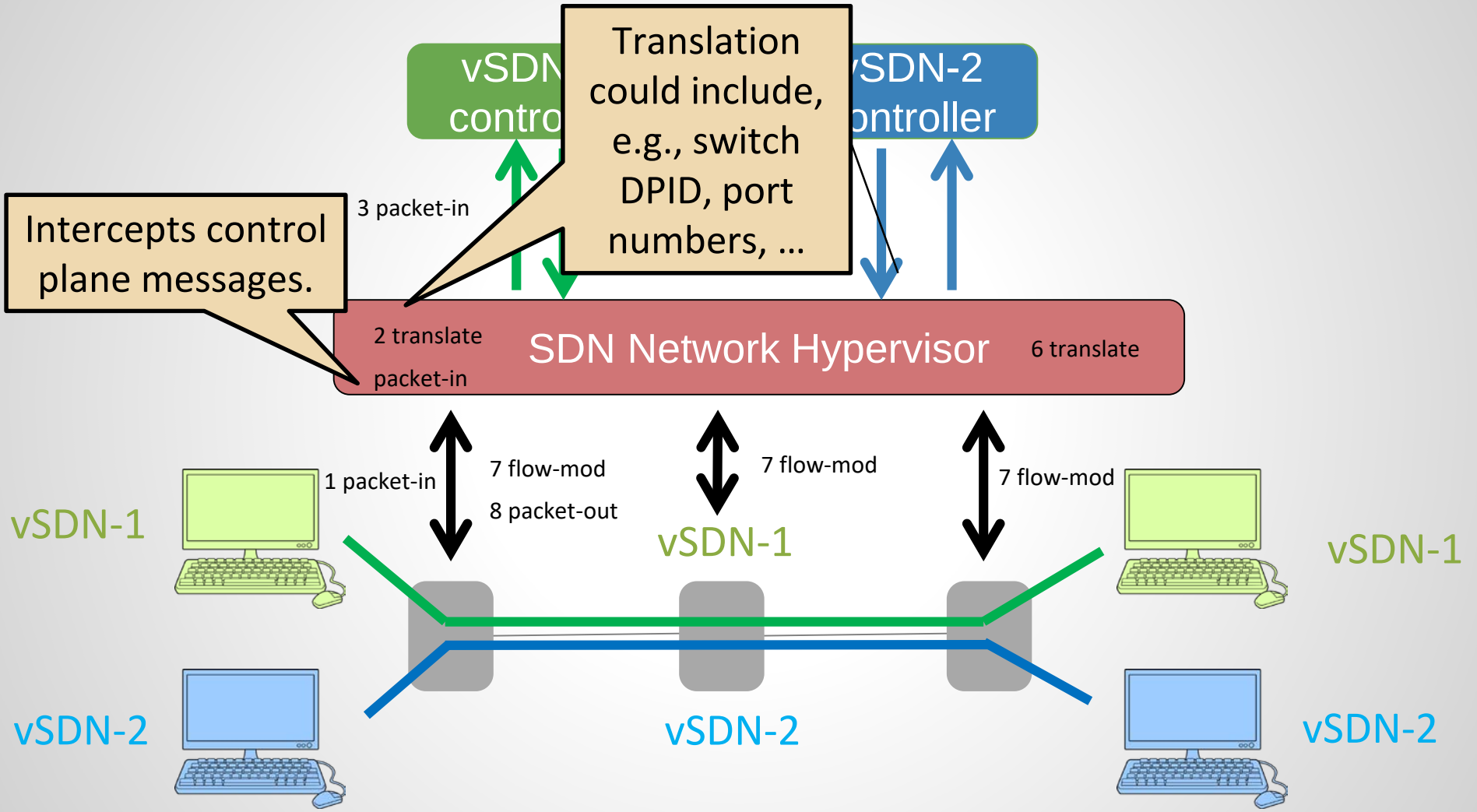
An Experiment: 2 vSDNs with bw guarantee!

On Predictable Performance…



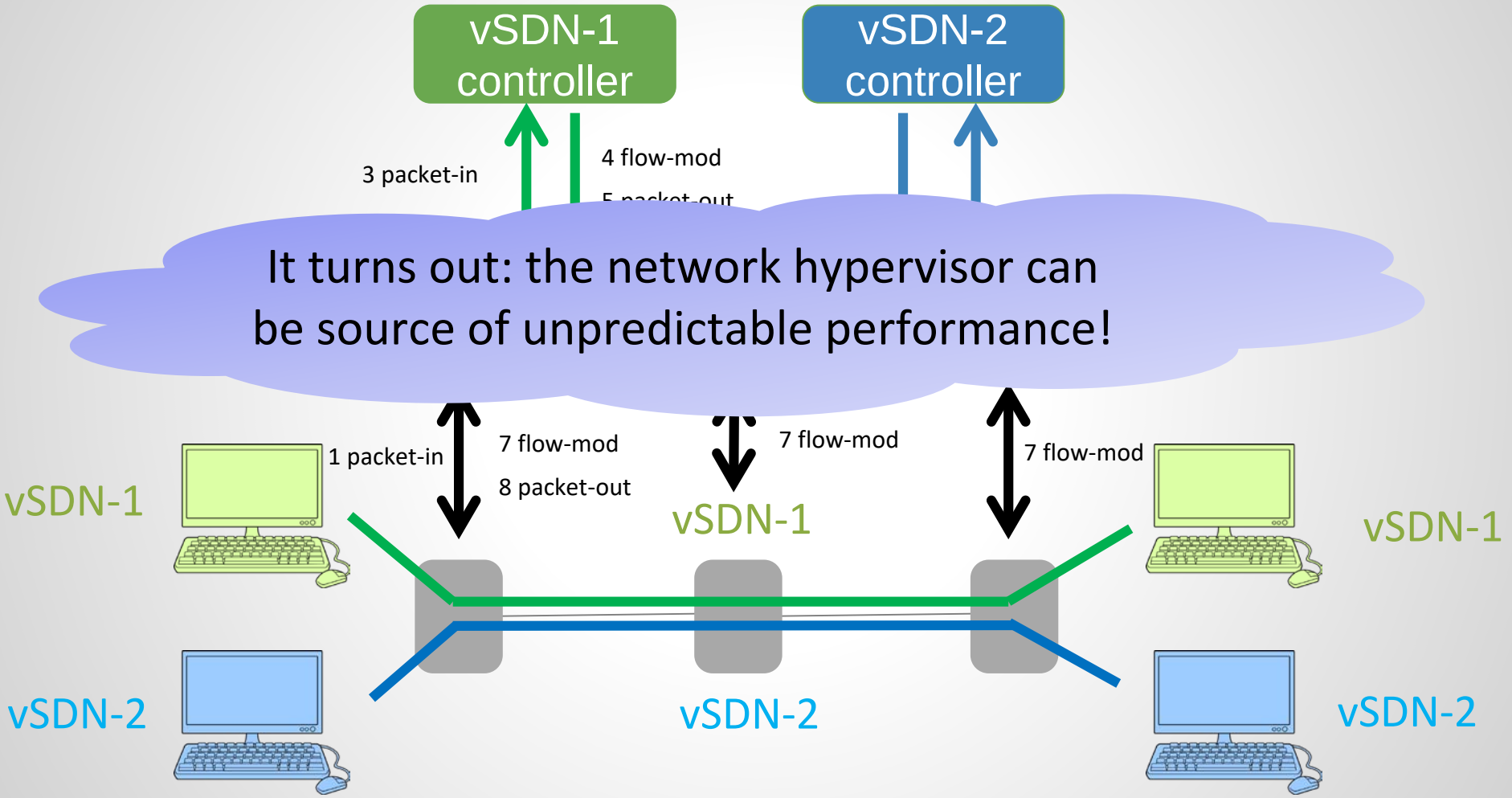
An Experiment: 2 vSDNs with bw guarantee!

On Predictable Performance…



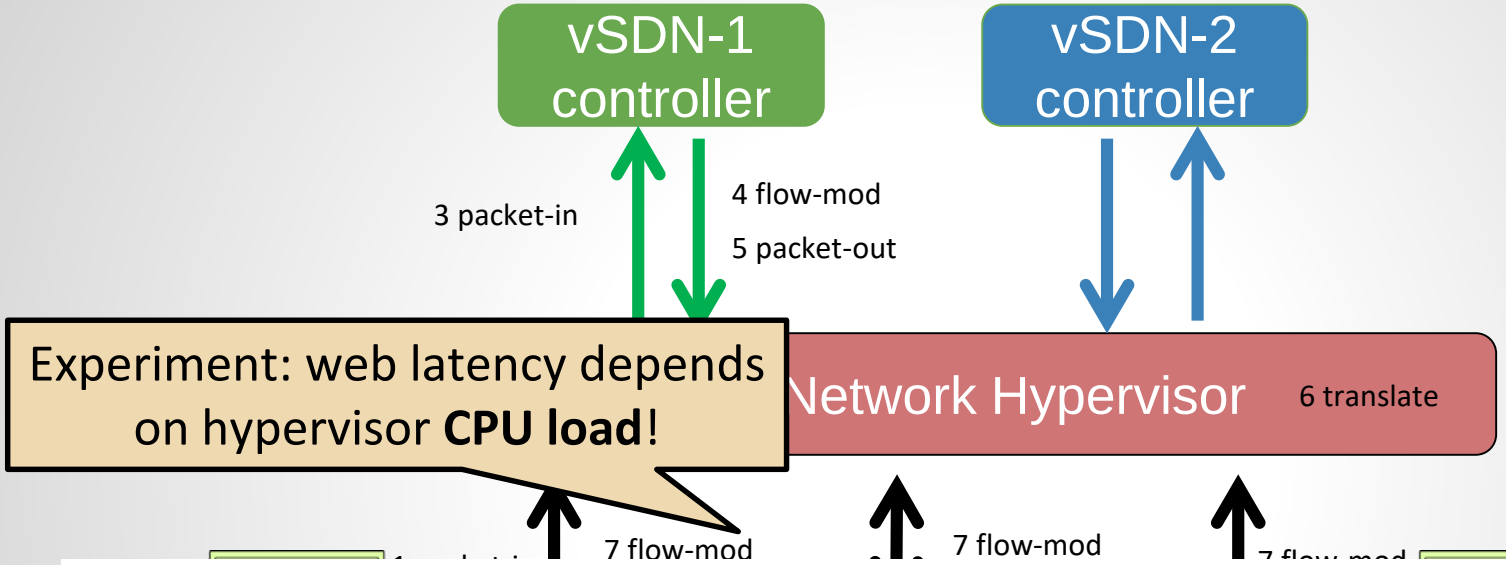
An Experiment: 2 vSDNs with bw guarantee!

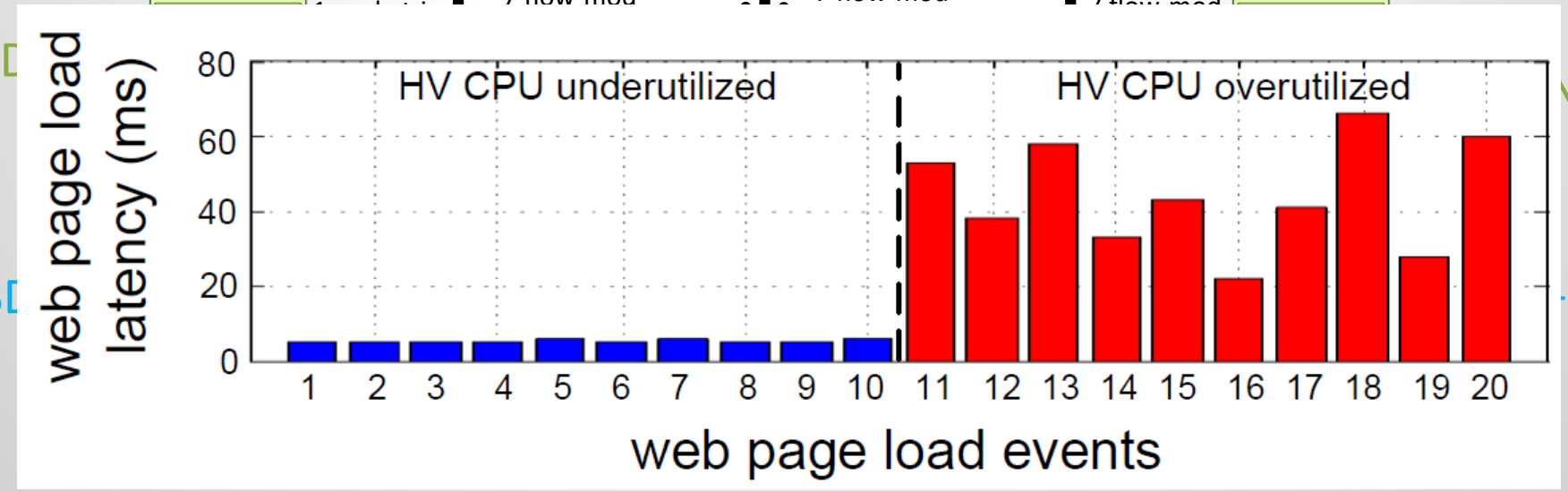
On Predictable Performance…



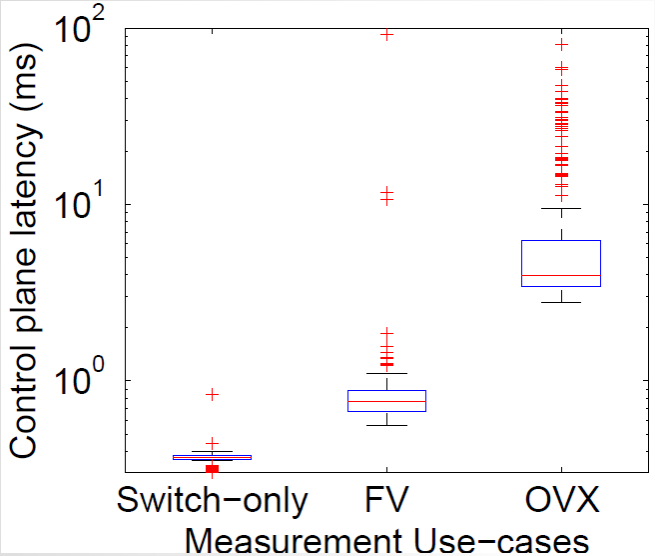
An Experiment: 2 vSDNs with bw guarantee!

On Predictable Performance…

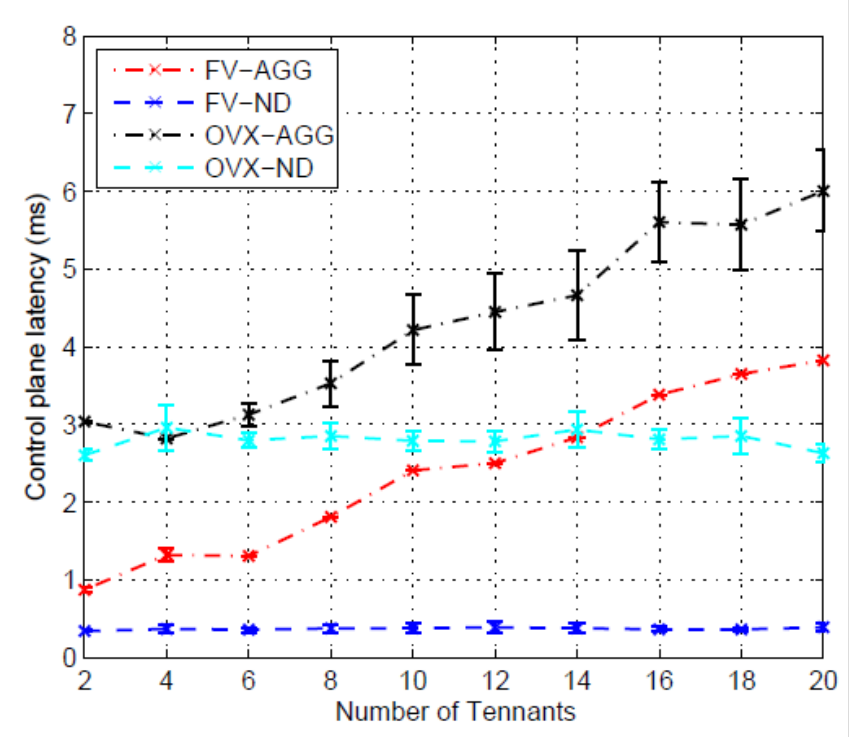




The Many Faces of Performance Interference



… number of tenants…



Performance also depends on hypervisor type…

(multithreaded or not, which version of Nagle's algorithm, etc.)

Further Reading

On The Impact of the Network Hypervisor on Virtual Network Performance
Andreas Blenk, Arsany Basta, Wolfgang Kellerer, and Stefan Schmid.
IFIP Networking, Warsaw, Poland, May 2019.

Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

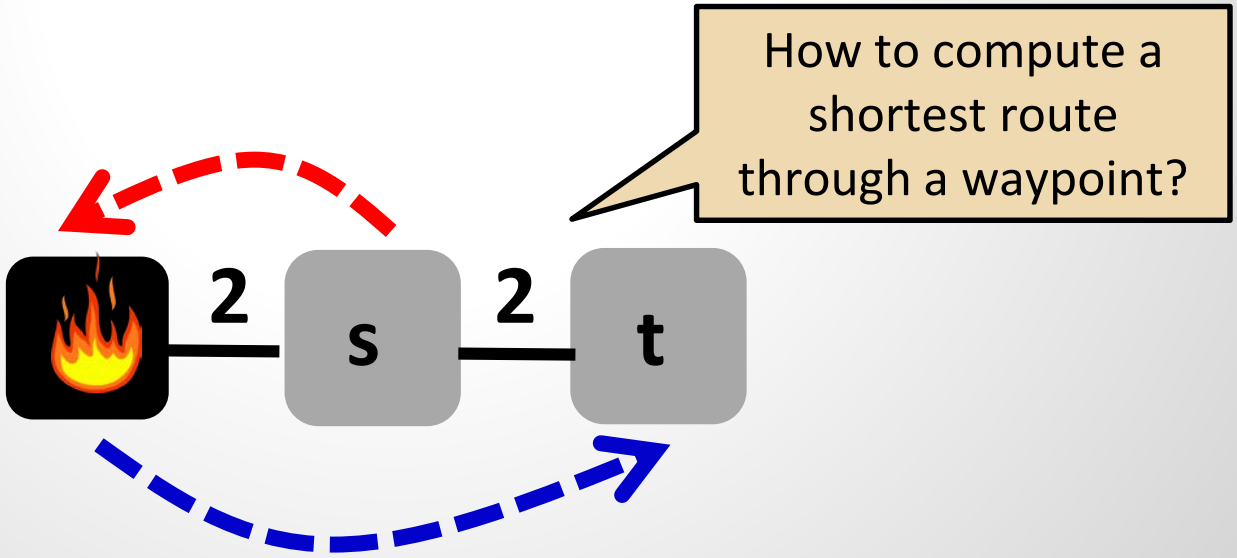
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- **Algorithms I: Flexible routing**
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

A Nugget: Flexible Traffic Steering

A novelty:

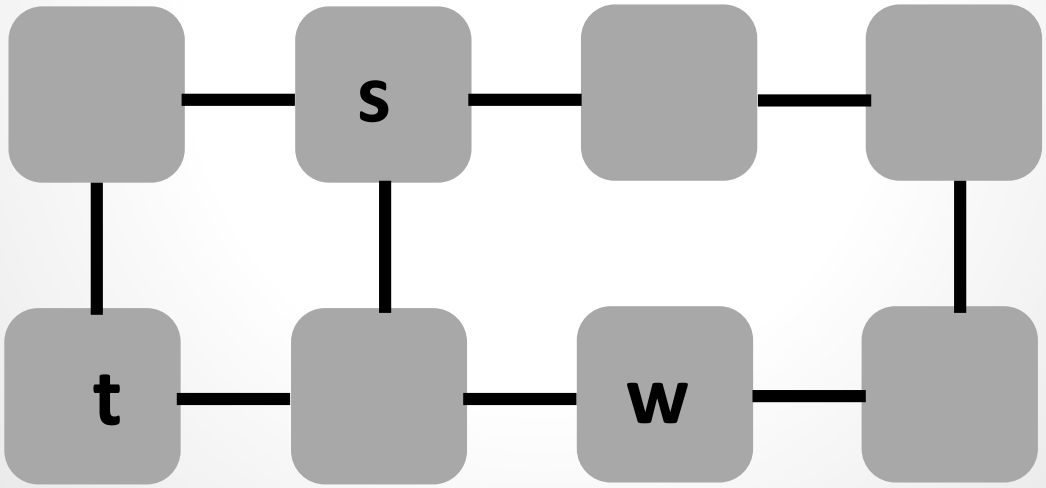
- ❑ Traditionally: routes form simple paths (e.g., shortest paths)
- ❑ Now: routing through middleboxes may require more general paths, *with loops*: a walk



Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

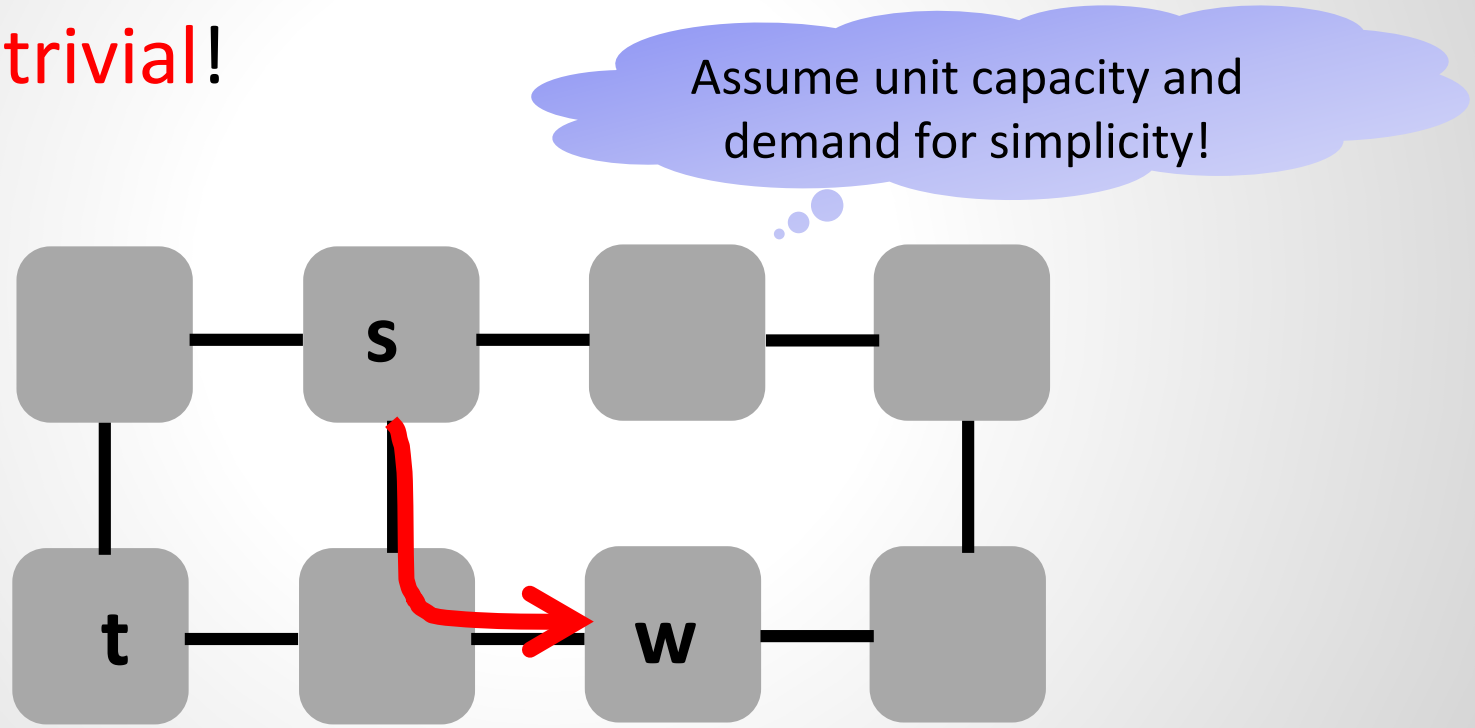
- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!



Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

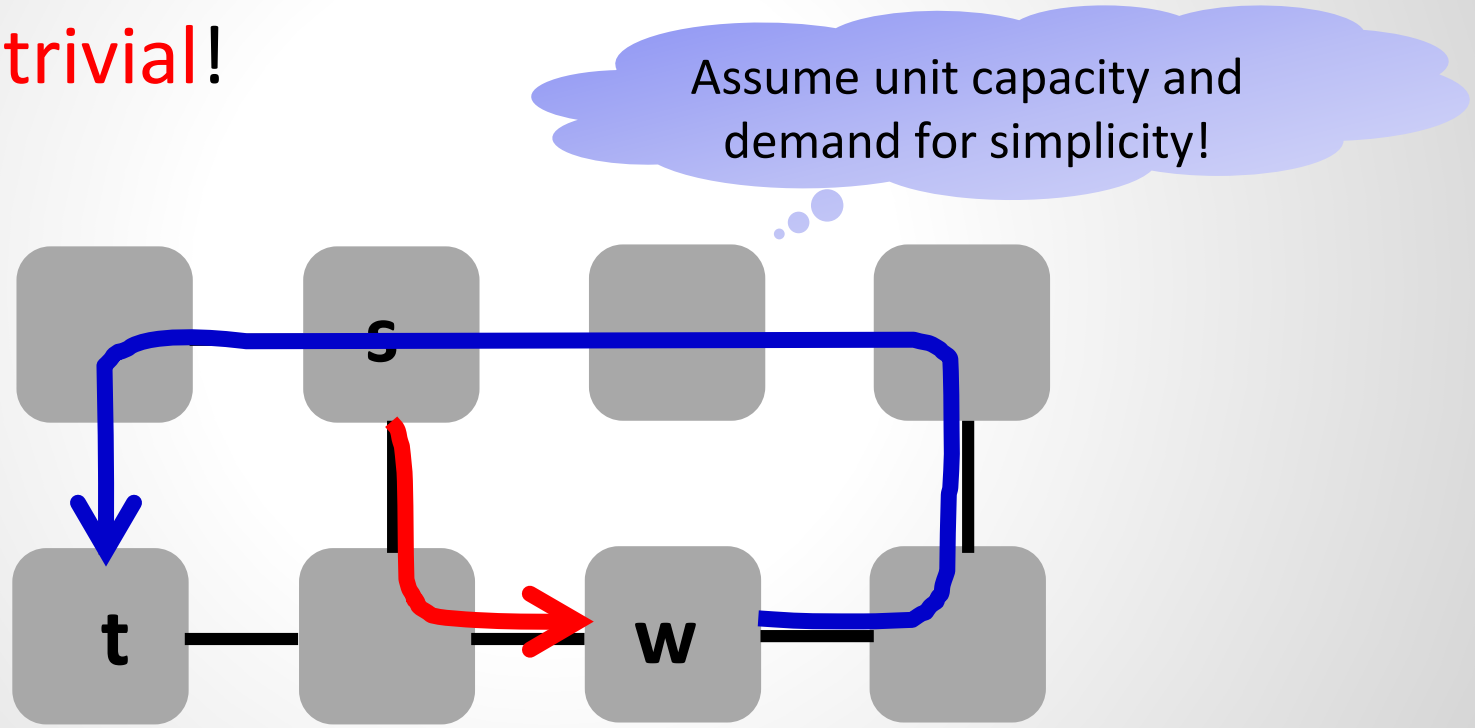
- Computing shortest routes through waypoints is non-trivial!



Greedy fails: **choose shortest path from s to w…**

Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

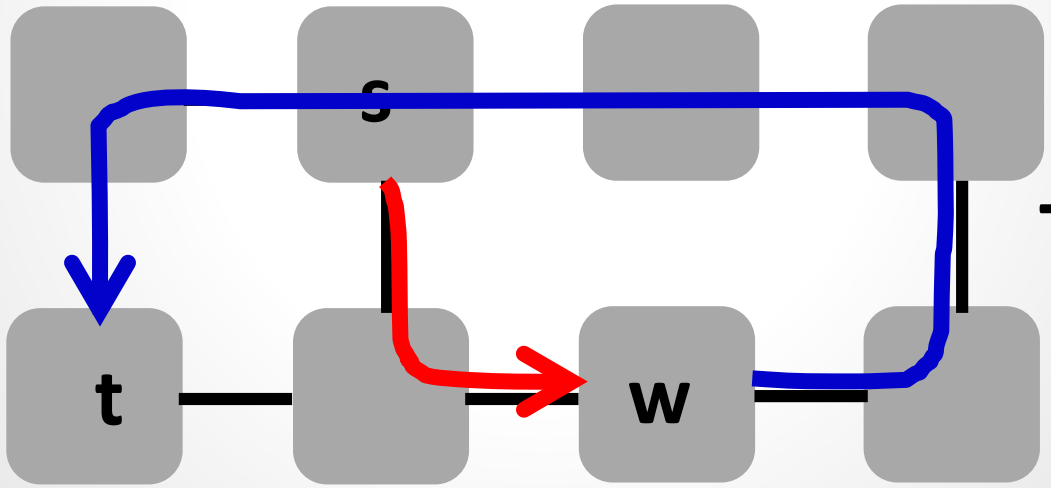


Greedy fails: **… now need long path from w to t**

Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

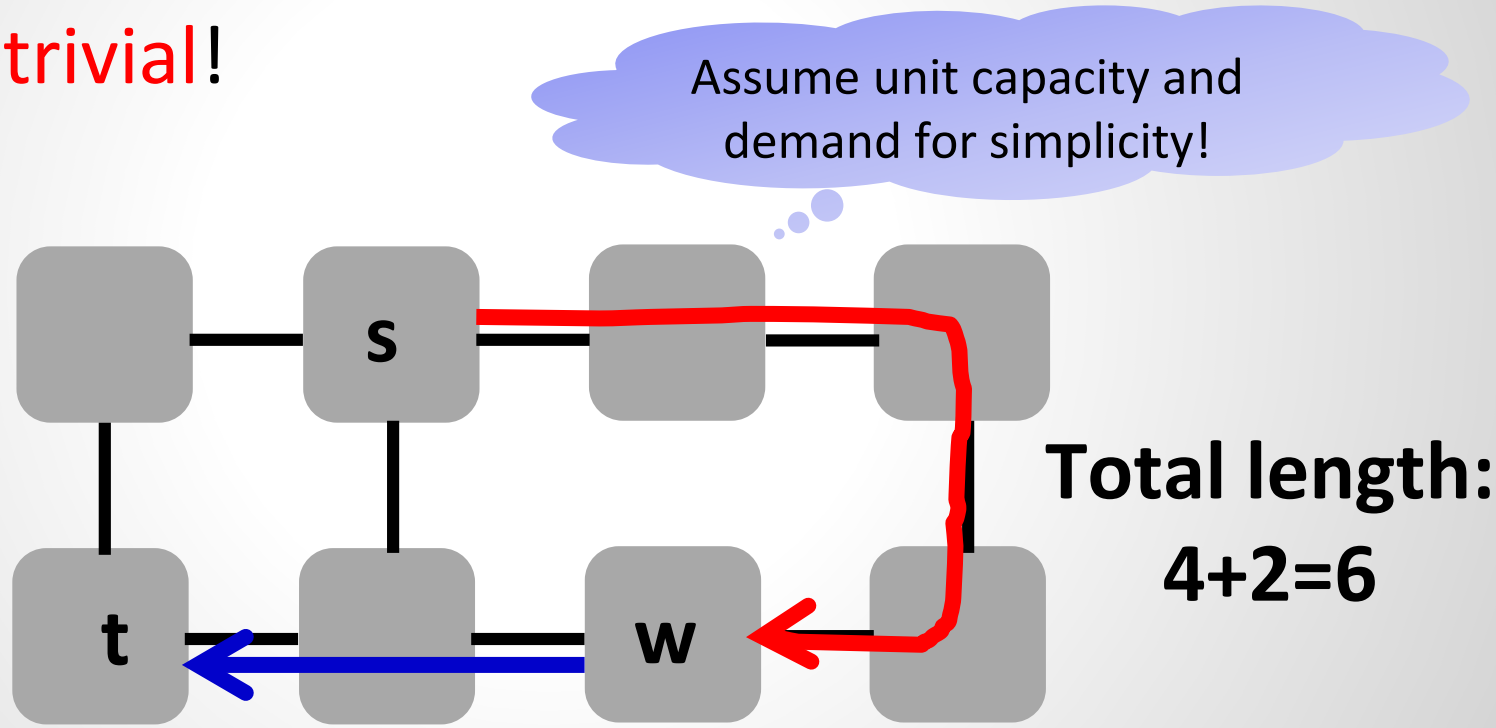


Total length: 2+6=8

Greedy fails: **… now need long path from w to t**

Comuting A Shortest Walk Through A *Single* Given Waypoint is Non-Trivial!

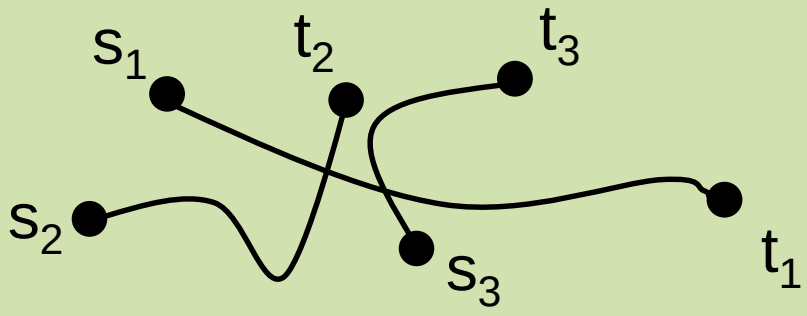
- Computing shortest routes through waypoints is non-trivial!



A *better solution*: *jointly* optimize the two segments!

Relationship to *Shortest Disjoint Paths*

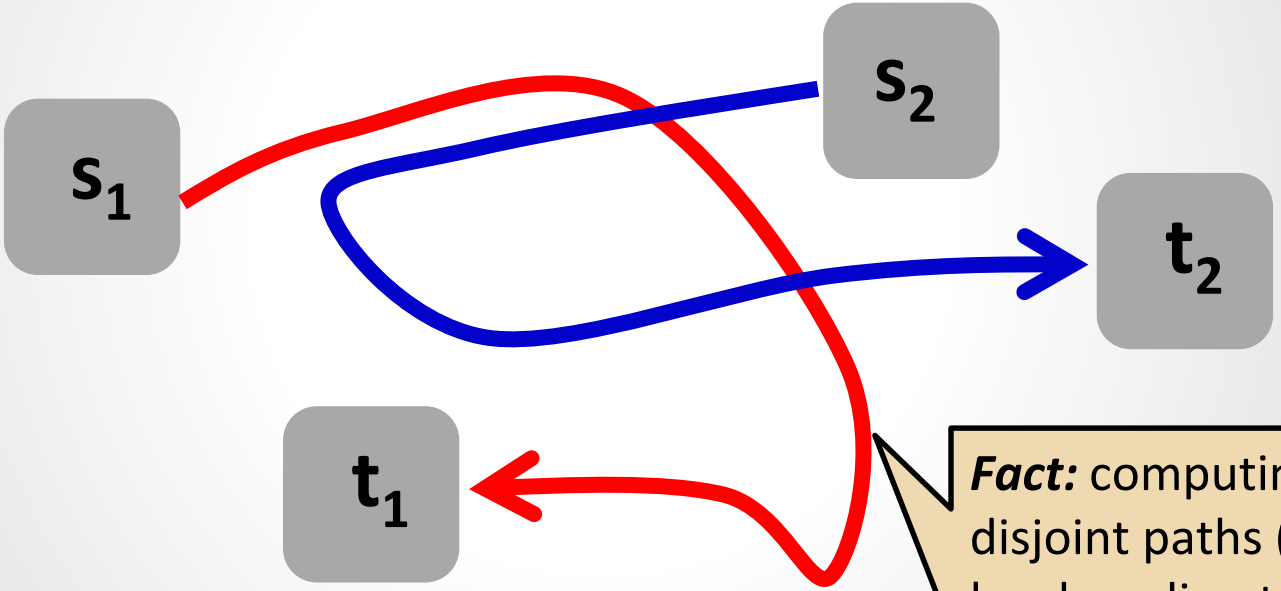
If capacities are 1, segments need to be edge-disjoint: A **disjoint paths problem**



- A well-known combinatorial problem!
- NP-hard on directed networks
- Feasibility in P on undirected networks for small (constant) number of flows
- Polytime randomized algorithm for 2 disjoint paths (recent result!)

NP-hard on *Directed* Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem

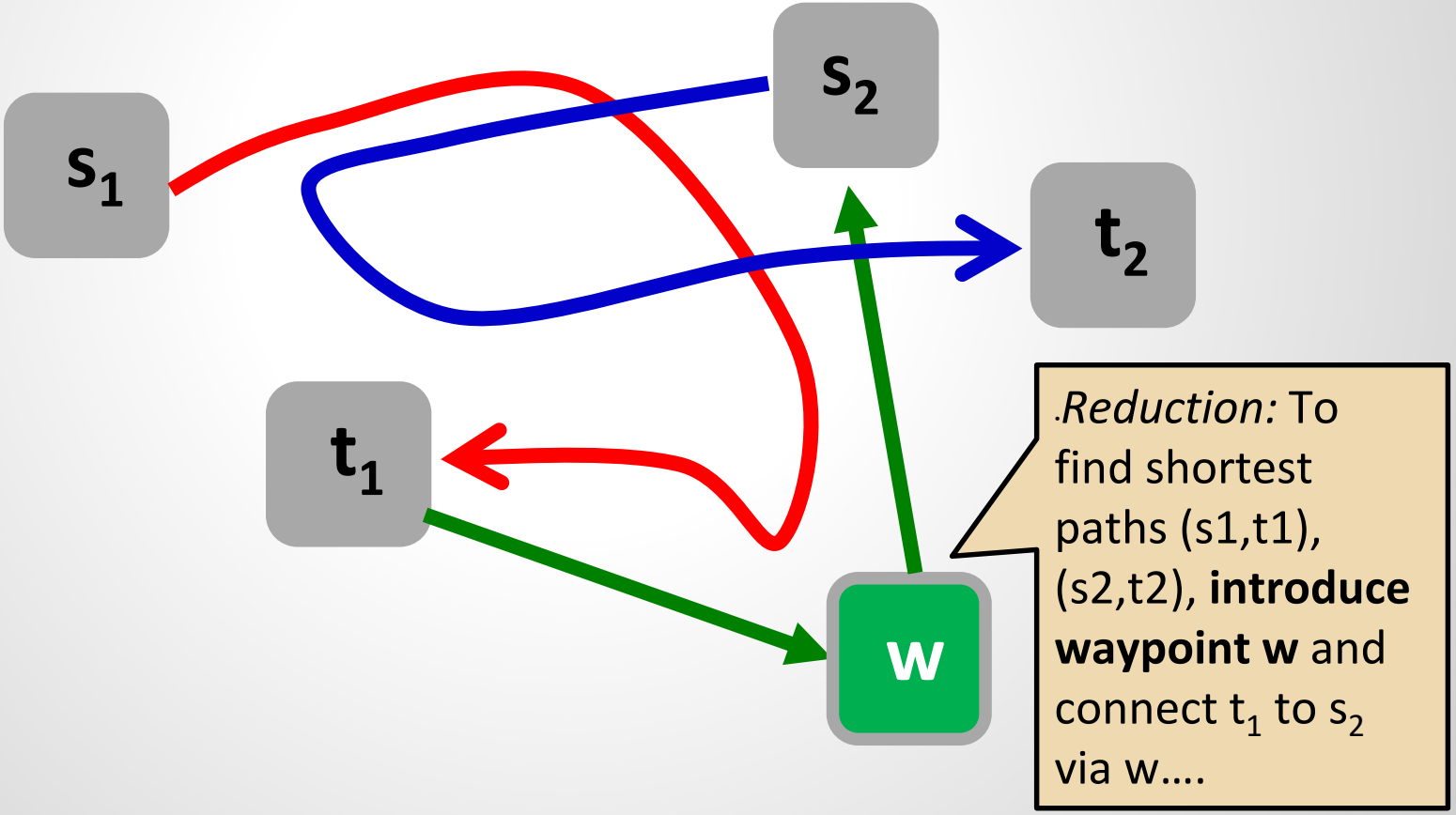


Fact: computing 2-disjoint paths (2DP) is NP-hard on directed graphs.

We show: If waypoint routing was in P, we could solve 2DP fast. ***Contradiction!***

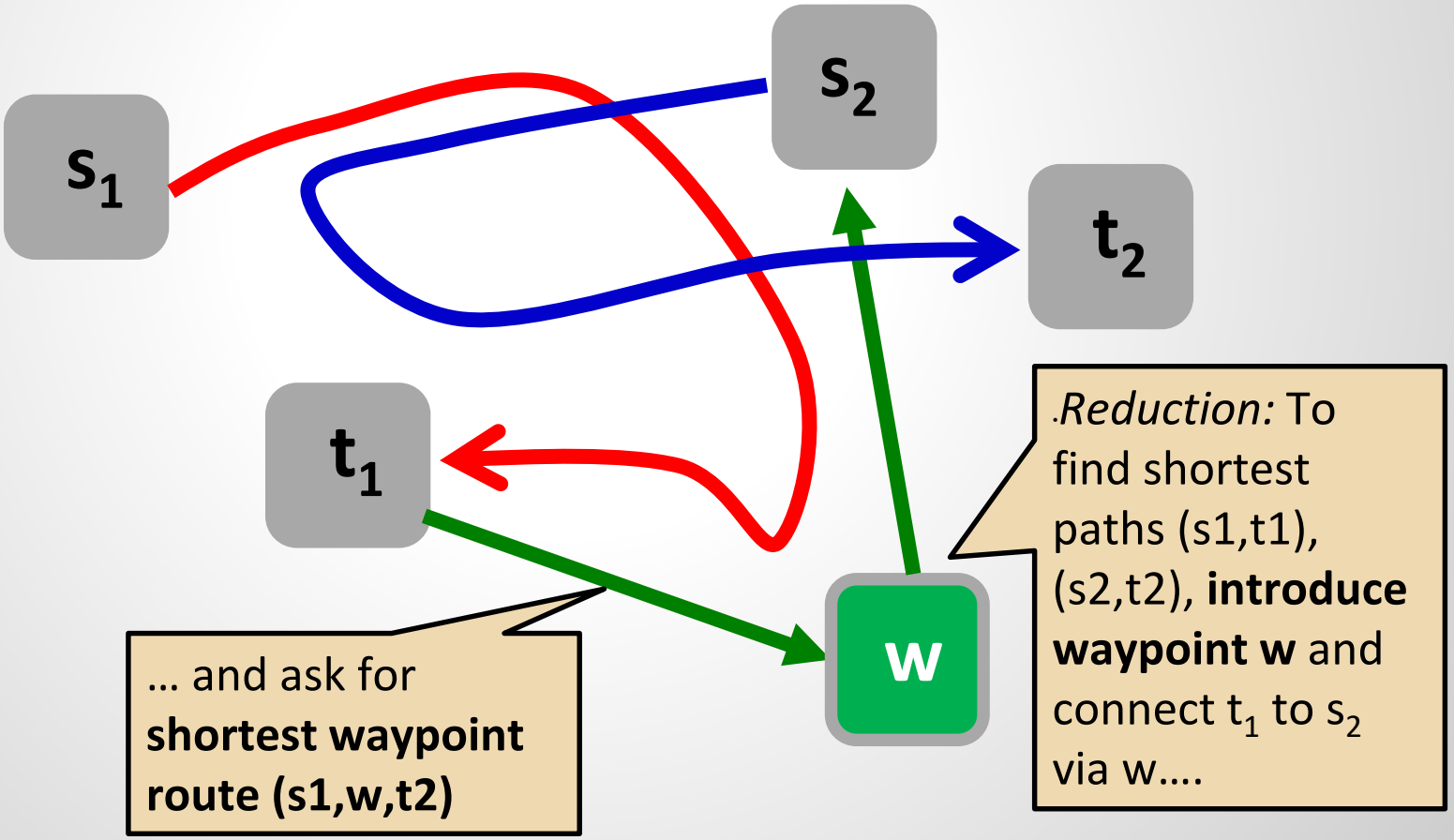
NP-hard on *Directed* Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem



NP-hard on *Directed* Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths $(s_1,t_1),(s_2,t_2)$ *to* shortest walk (s,w,t) problem

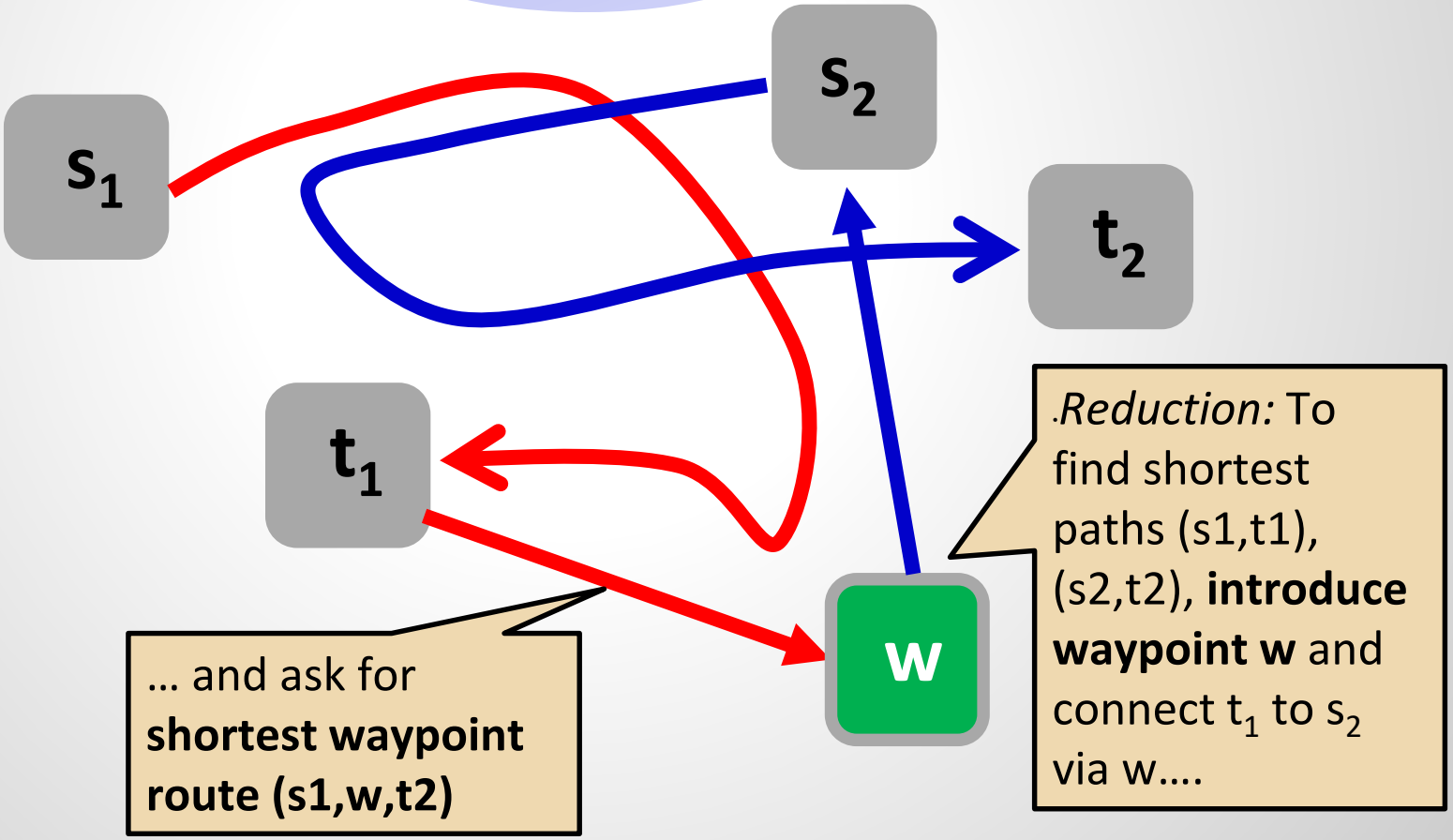


NP-hard on Direct

Reduc

to shortest w

The walk (s_1,w,t_2) walk defines a (s_1,t_1) and a (s_2,t_2) path pair before/after the waypoint! Solves original problem: **Contradiction!**



What about waypoint routes on *undirected* networks?

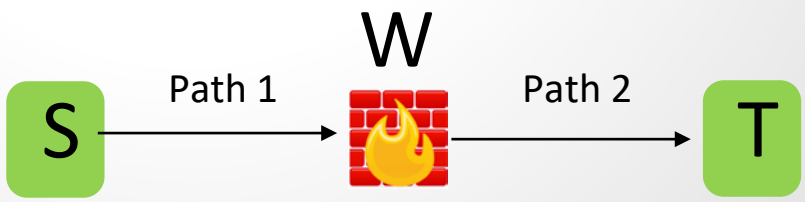
What about waypoint routes on *undirected* networks? (2)

Idea: Reduce it ***to*** disjoint paths problem!

- For a single waypoint, can even compute ***shortest route (walk)***!
  - Recall: there is a randomized polytime algorithm for 2 disjoint paths

Step 1: replace weights with parallel links

Step 2: compute 2 disjoint paths (A,W) and (W,B)



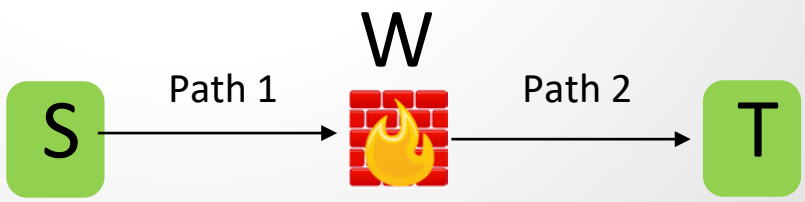
What about waypoint routes on *undirected* networks? (2)

Idea: Reduce it ***to*** disjoint paths problem!

- For a single waypoint, can even compute ***shortest route (walk)***!
  - Recall: there is a randomized polytime algorithm for 2 disjoint paths

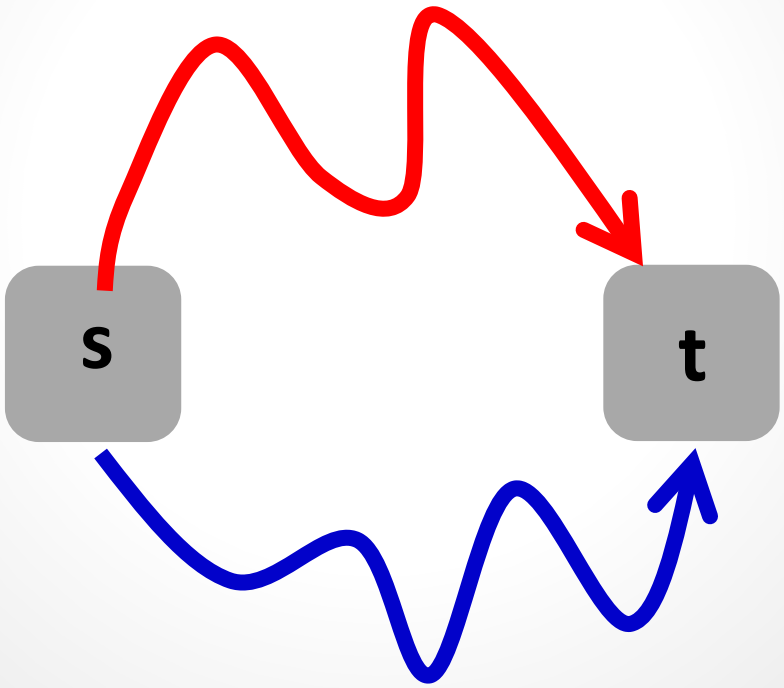
Good news: For a single waypoint, ***shortest*** paths can be computed even ***faster***!

Step 2: compute 2 disjoint paths (A,W) and (W,B)



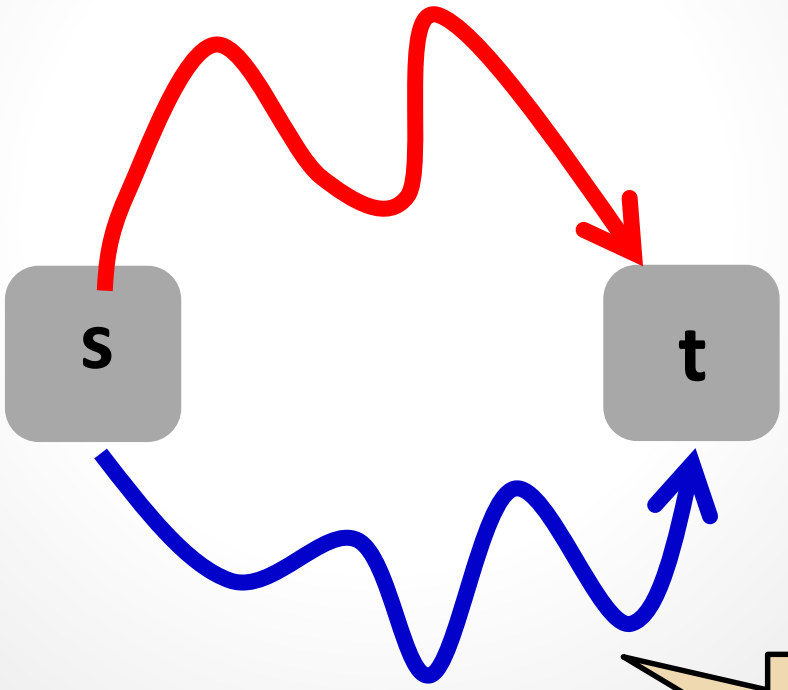
Walking Through a Waypoint on Steroids: Suurballe's Algorithm

- Suurballe's algorithm: finds two (edge-)disjoint shortest paths between same endpoints:



Walking Through a Waypoint on Steroids: Suurballe's Algorithm

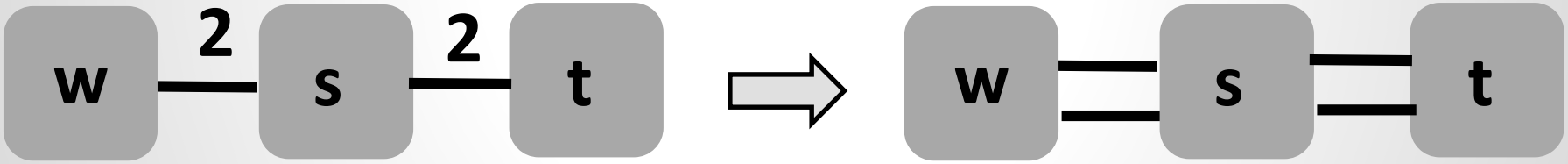
- Suurballe's algorithm: finds two (edge-)disjoint shortest paths between same endpoints:



How to compute a shortest (s,w,t) route with this algorithm??

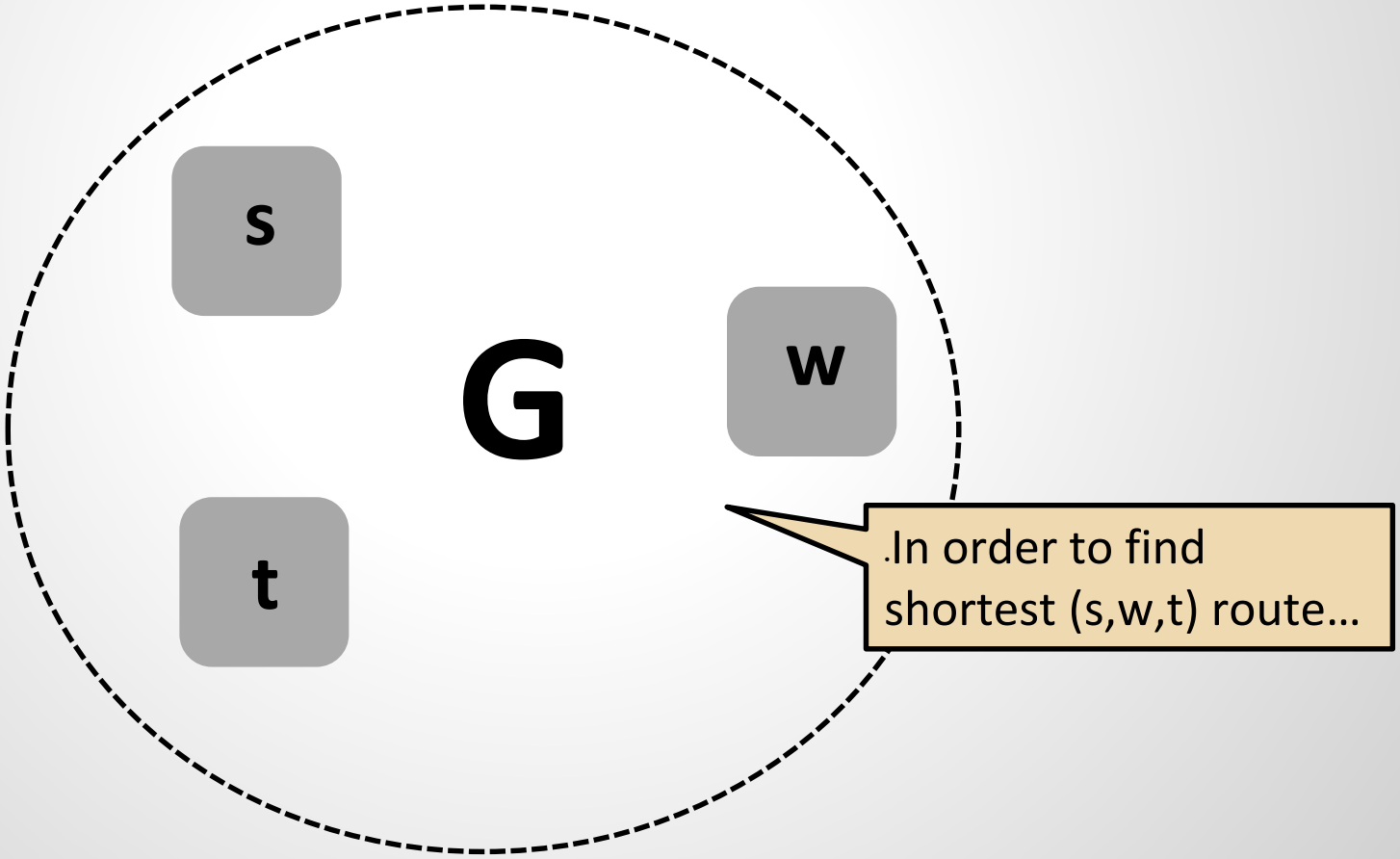
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 1:** replace capacities with parallel edges: paths will become edge-disjoint



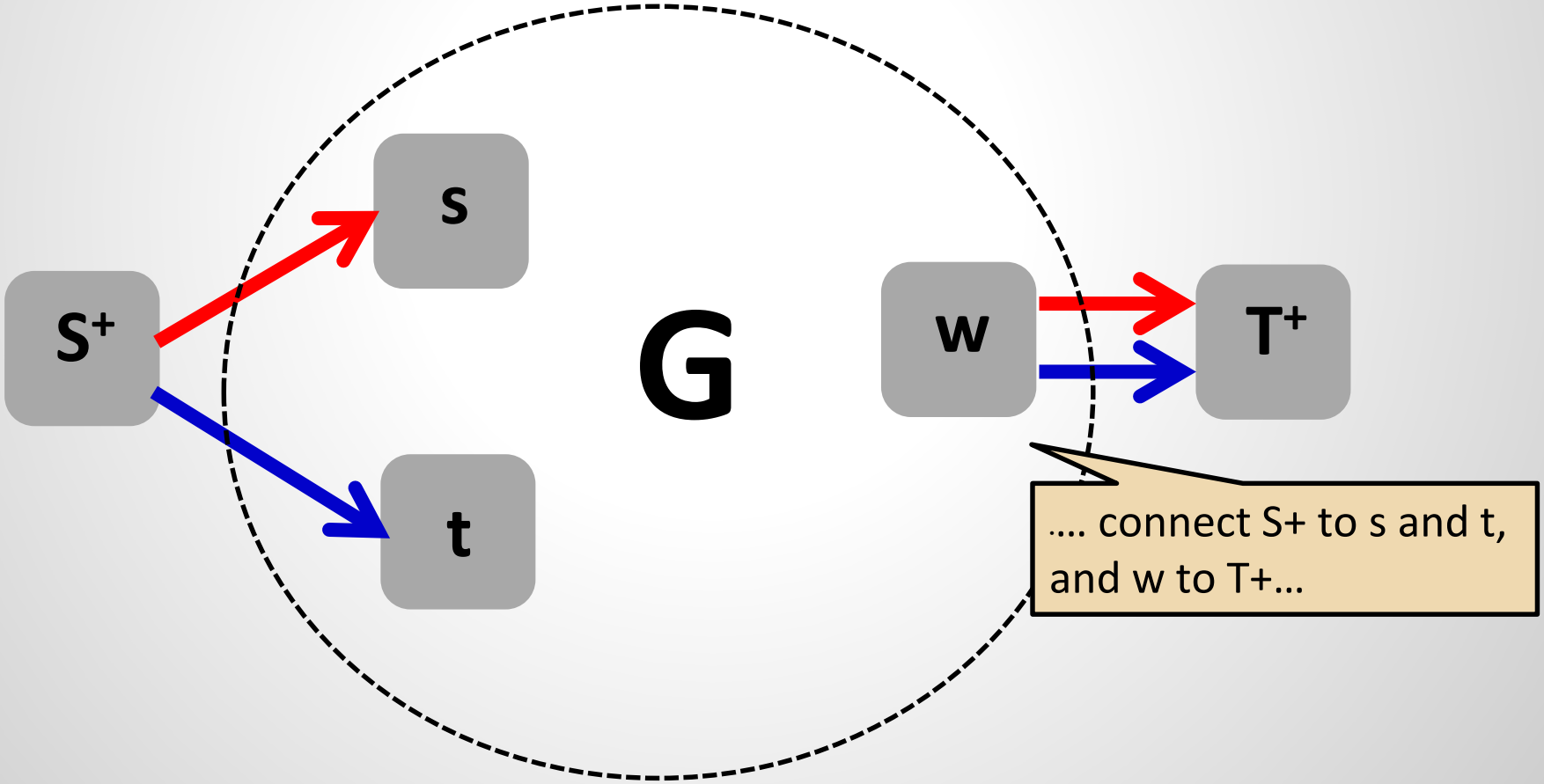
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



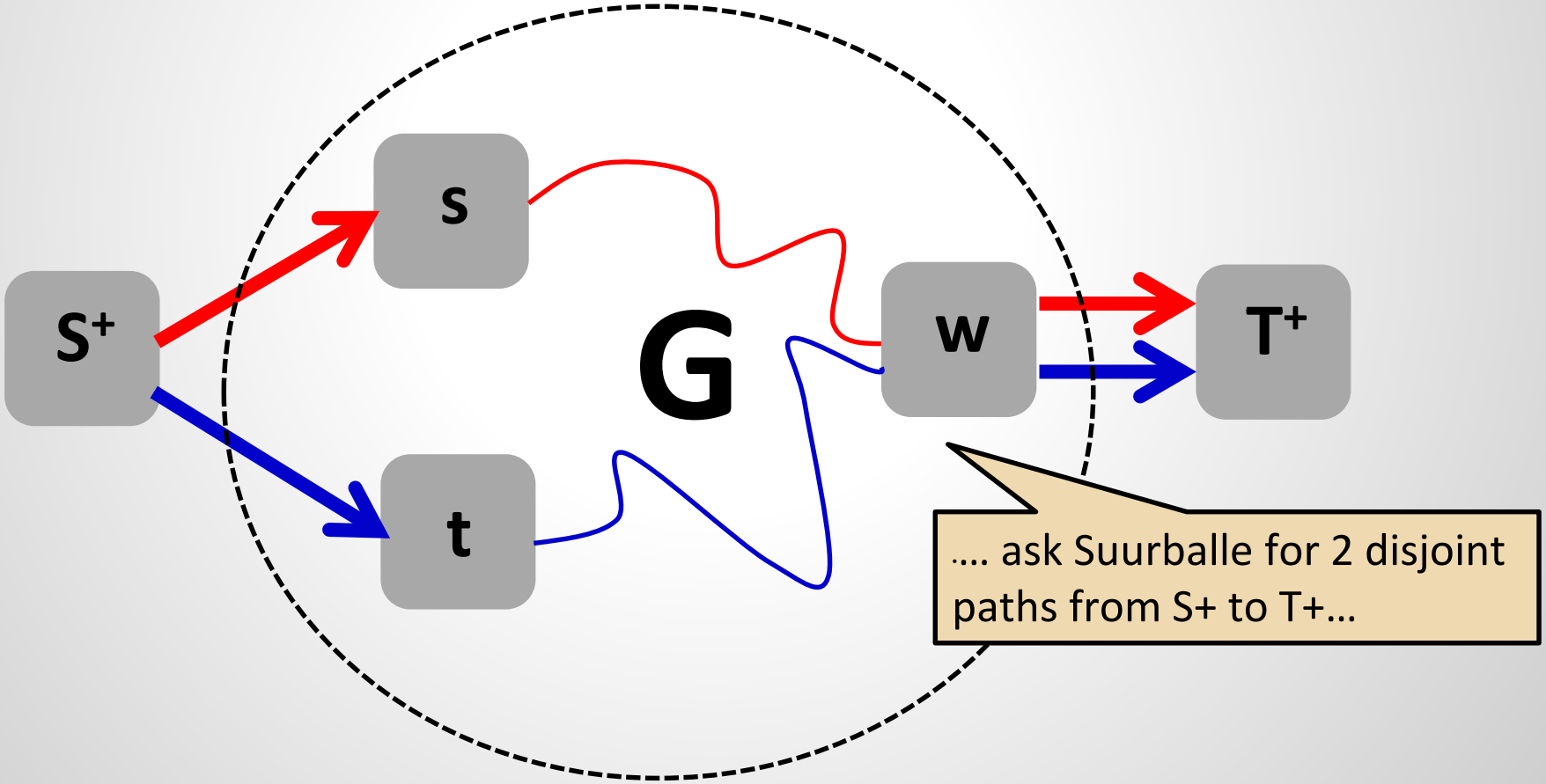
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



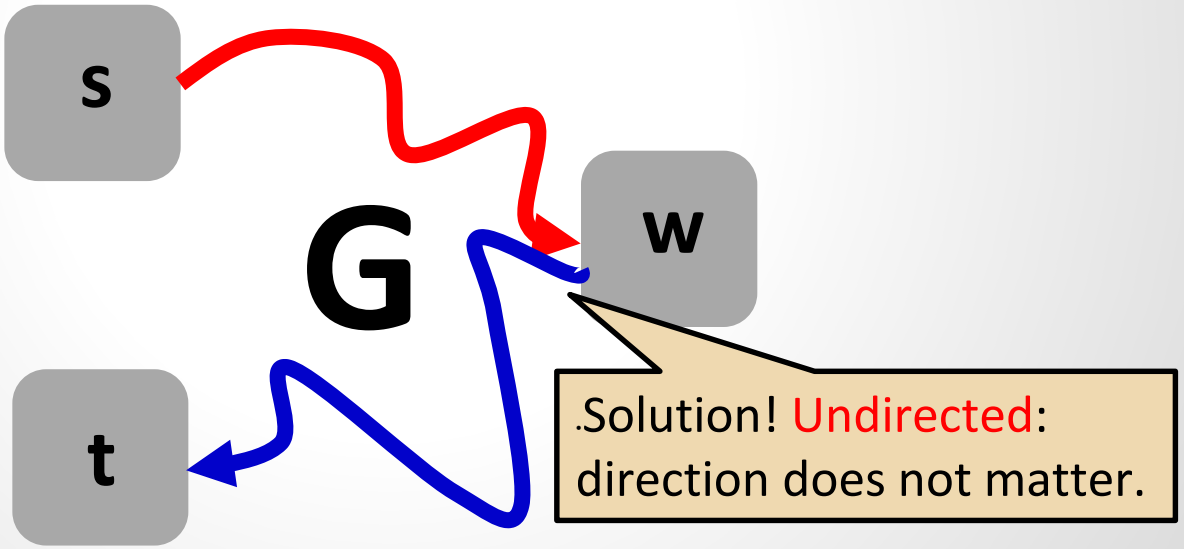
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



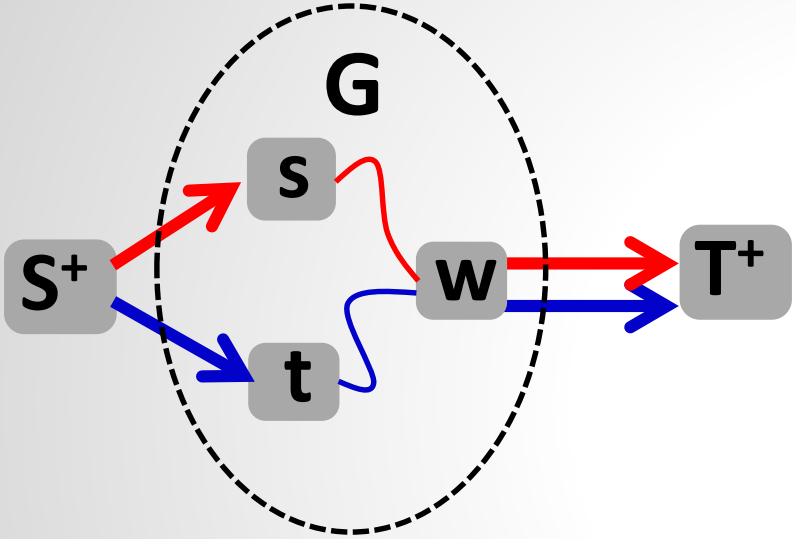
Walking Through a Waypoint on Steroids: Reduction to Suurballe's Algorithm

- **Step 2:** Reduction to Suurballe's algorithm:



Wait A Moment…!?

Can we not use Suurballe as well to solve 2 disjoint paths?

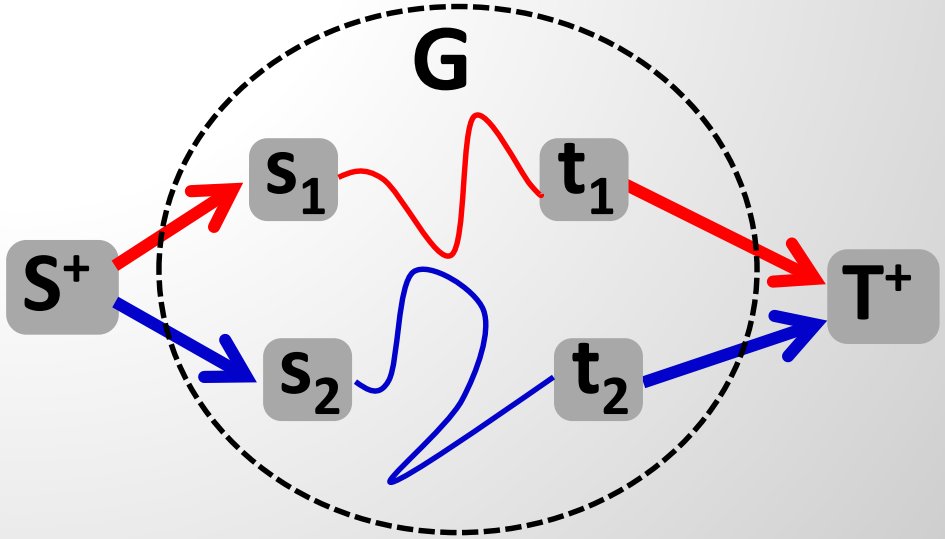


Reduction

Waypoint Routing ⇒ Suurballe

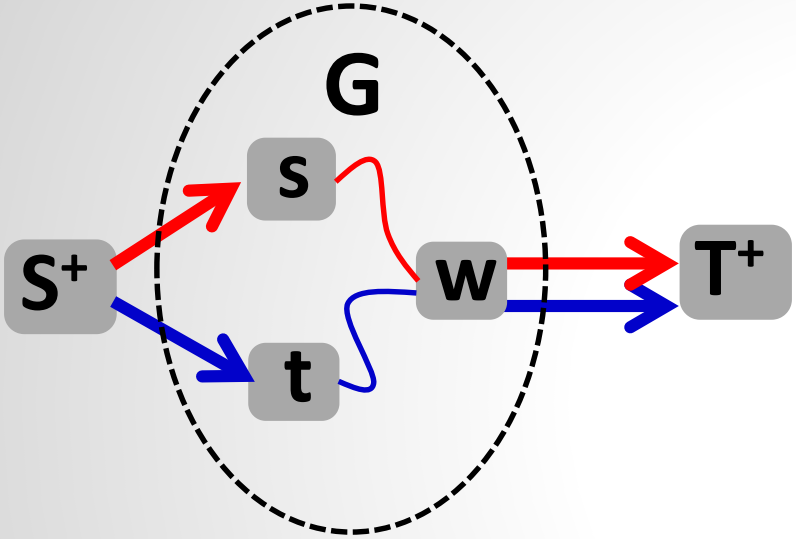
Reduction

2 Disjoint Paths ⇒ Suurballe



Wait A Moment…!?

No! Solves a much easier problem: 2 routes from $\{s_1,s_2\}$ to $\{t_1,t_2\}$.

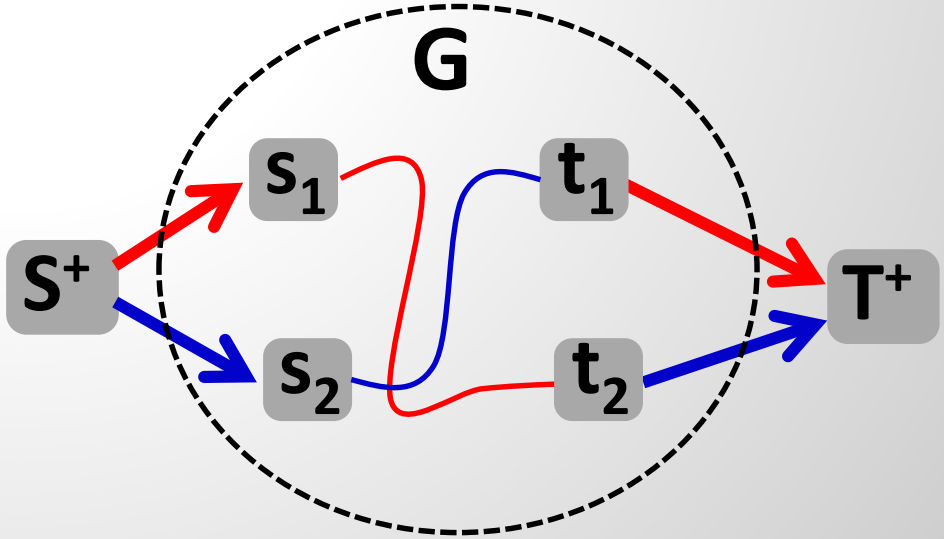


Reduction

Waypoint Routing ⇒ Suurballe

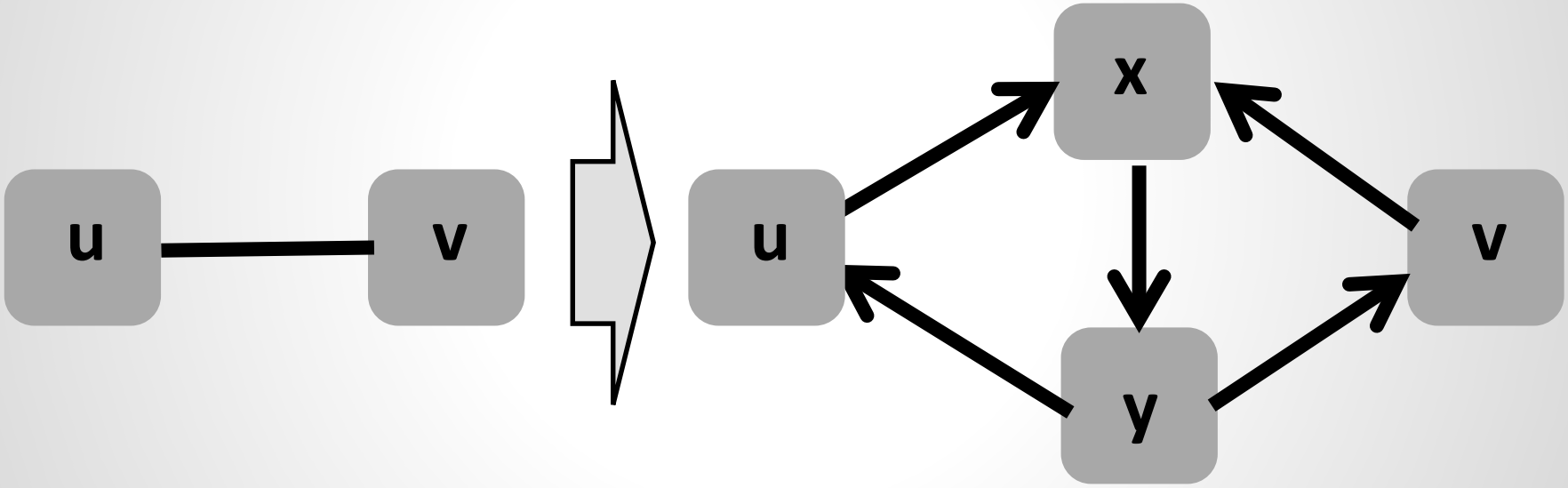
Reduction

2 Disjoint Paths ⇒ Suurballe



Remarks: Under the rug…

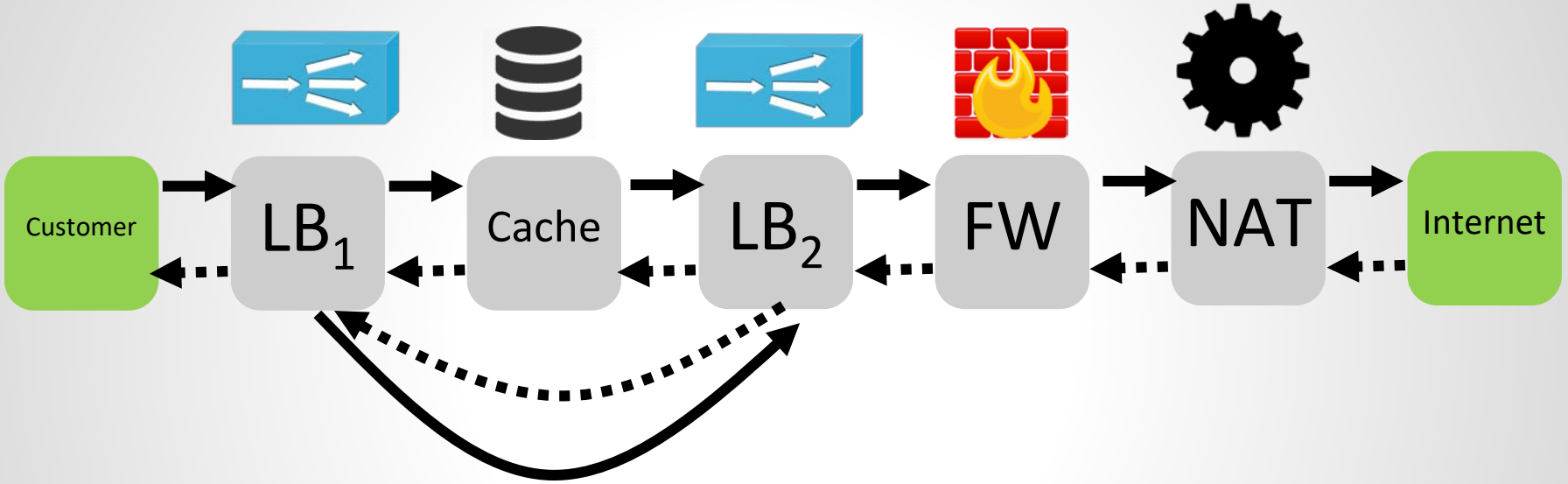
- **Remark 1:** Suurballe is actually for directed substrate graphs, so need gadget to transform problem in right form:



- **Remark 2:** Suurballe: for vertex disjoint
  - Suurballe & Tarjan: edge disjoint

Remark: There are complex requests…

IETF Draft:



- Service chain for mobile operators
- Load-balancers are used to route (parts of) the traffic through cache

Can be seen as a virtual network…

Credits: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06

Further Reading

Walking Through Waypoints
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid.
13th Latin American Theoretical Informatics Symposium (**LATIN**), Buenos Aires, Argentina, April 2018.

Charting the Algorithmic Complexity of Waypoint Routing
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), 2018.

Roadmap

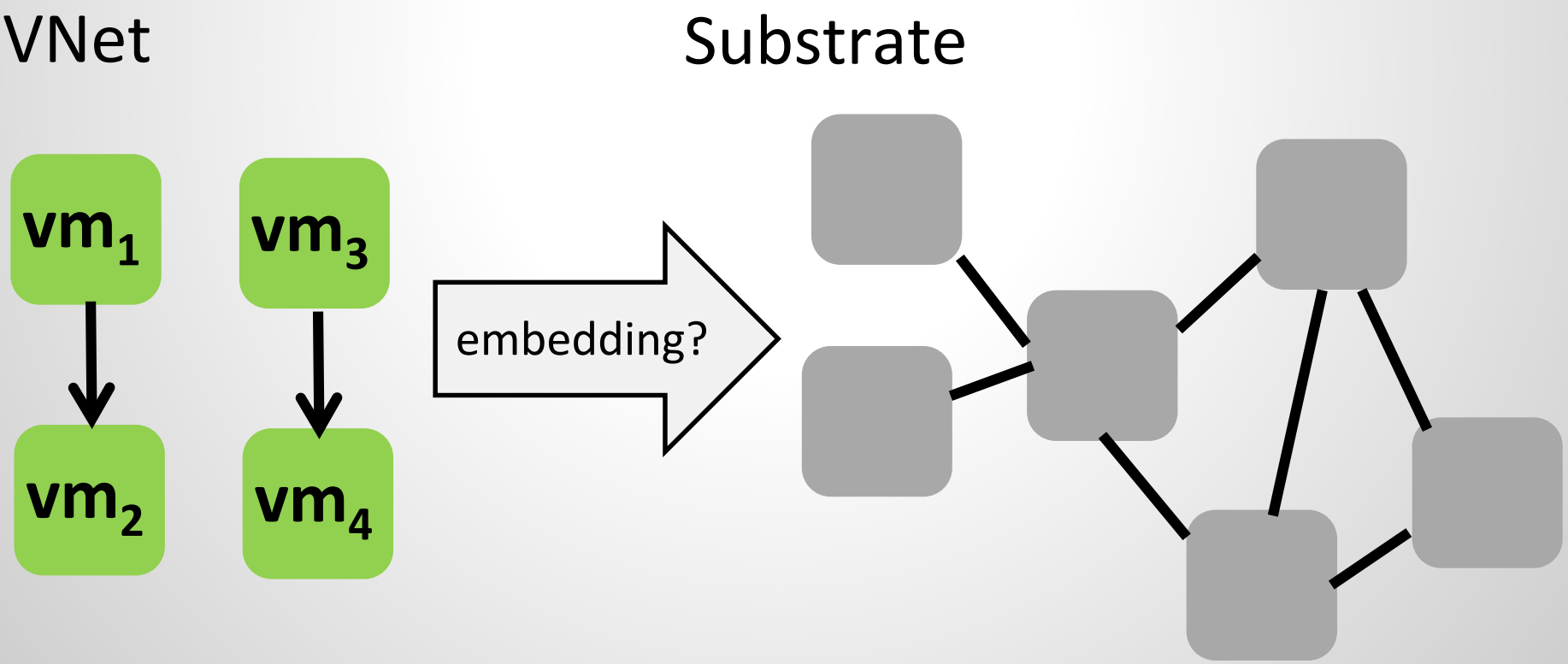
- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- **Algorithms II: Flexible embedding and slicing**
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

The Virtual Network Embedding Problem

- A fundamental resource allocation problem
- 2 dimensions of flexibility:
  - Mapping of virtual nodes (to physical nodes)
  - Mapping of virtual links (to paths)

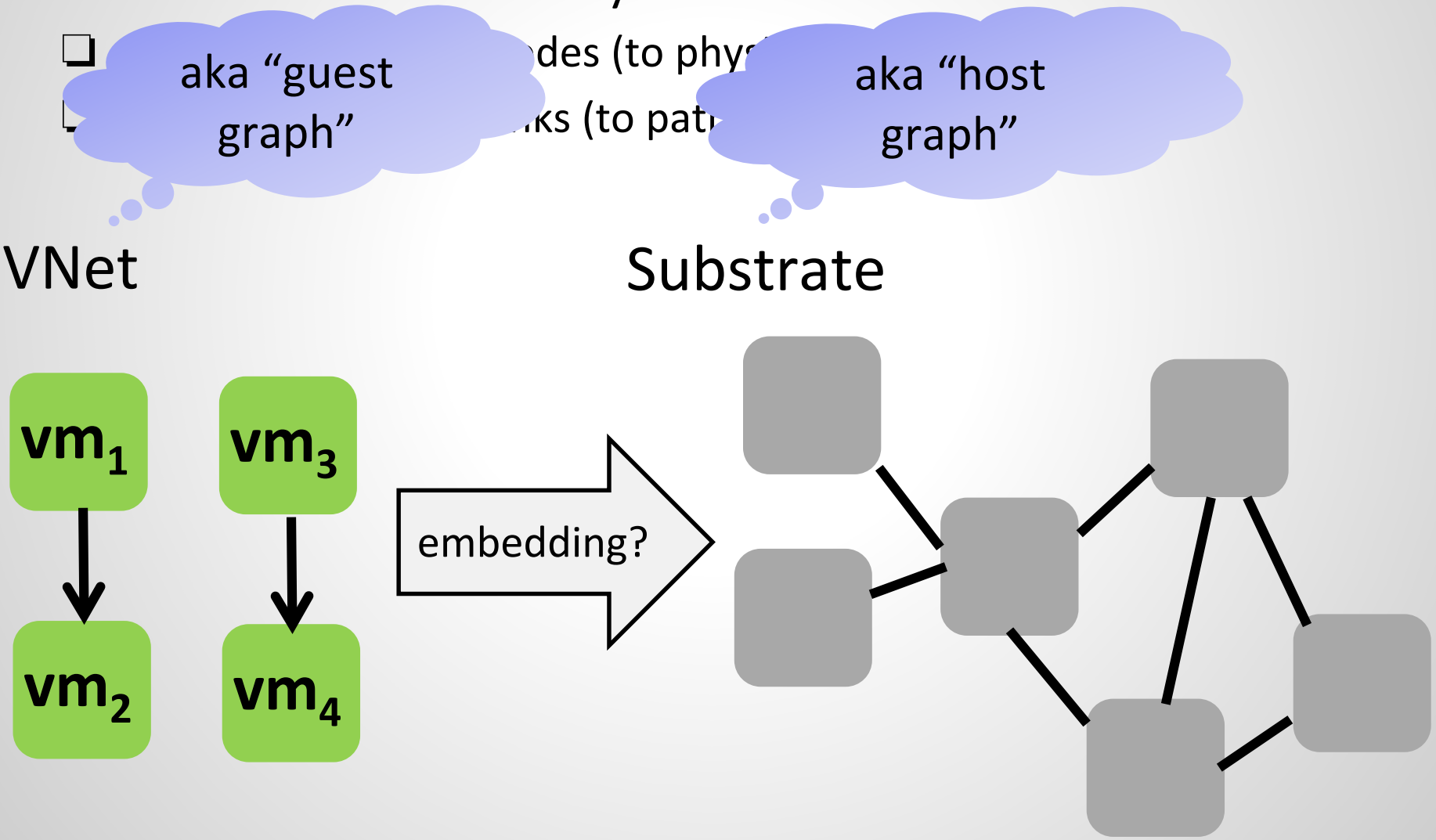


The Virtual Network Embedding Problem

- A fundamental resource allocation problem
- 2 dimensions of flexibility:
  - des (to phy
  - ks (to pat

aka "guest graph"

aka "host graph"

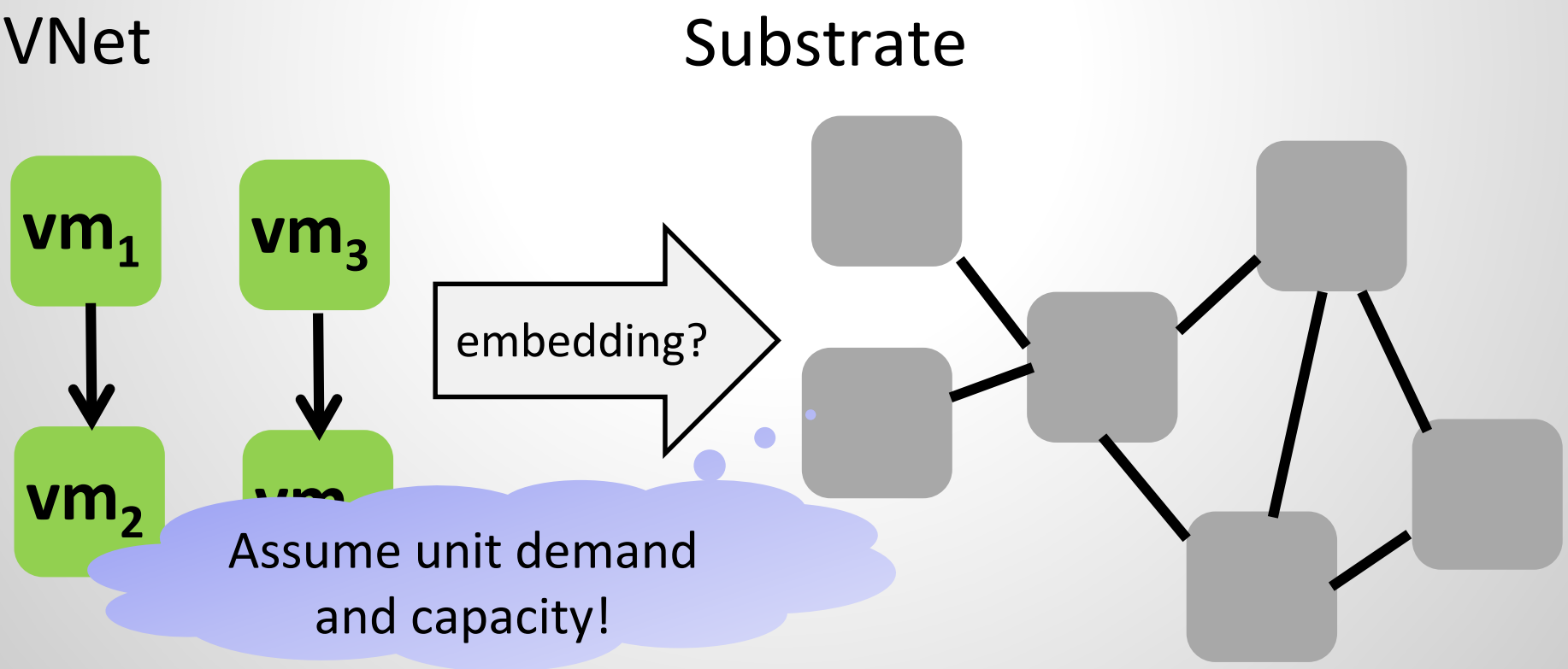


The Virtual Network Embedding Problem

- A fundamental resource allocation problem
- 2 dimensions of flexibility:
  - des (to phy
  - ks (to pat

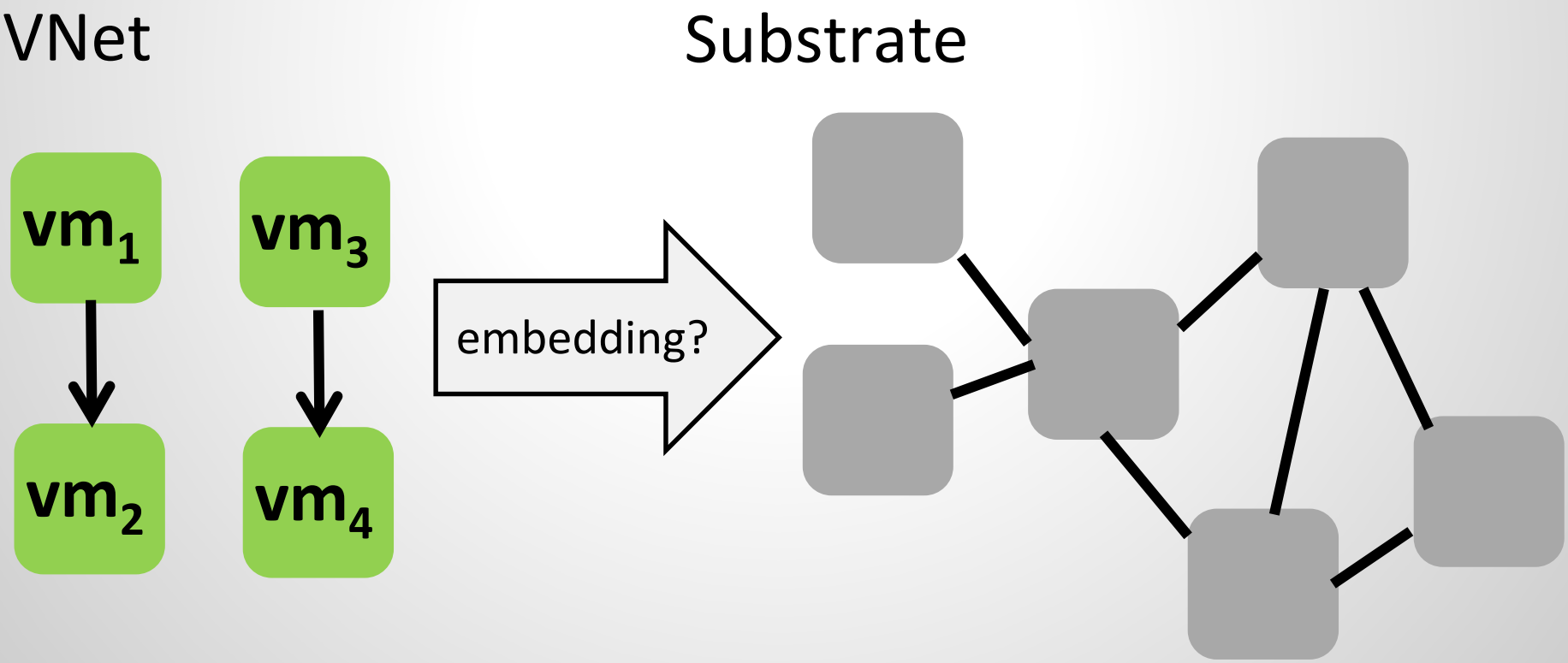
aka "guest graph"

aka "host graph"



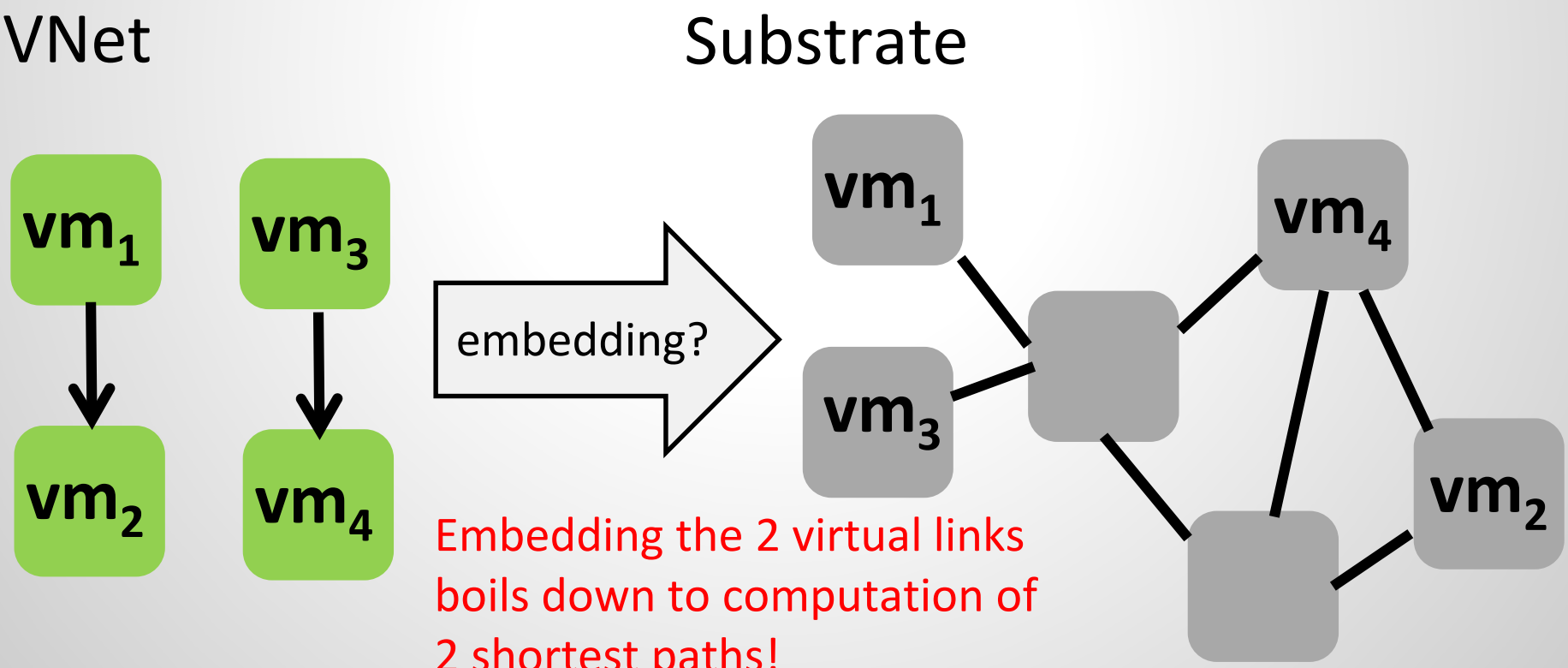
The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



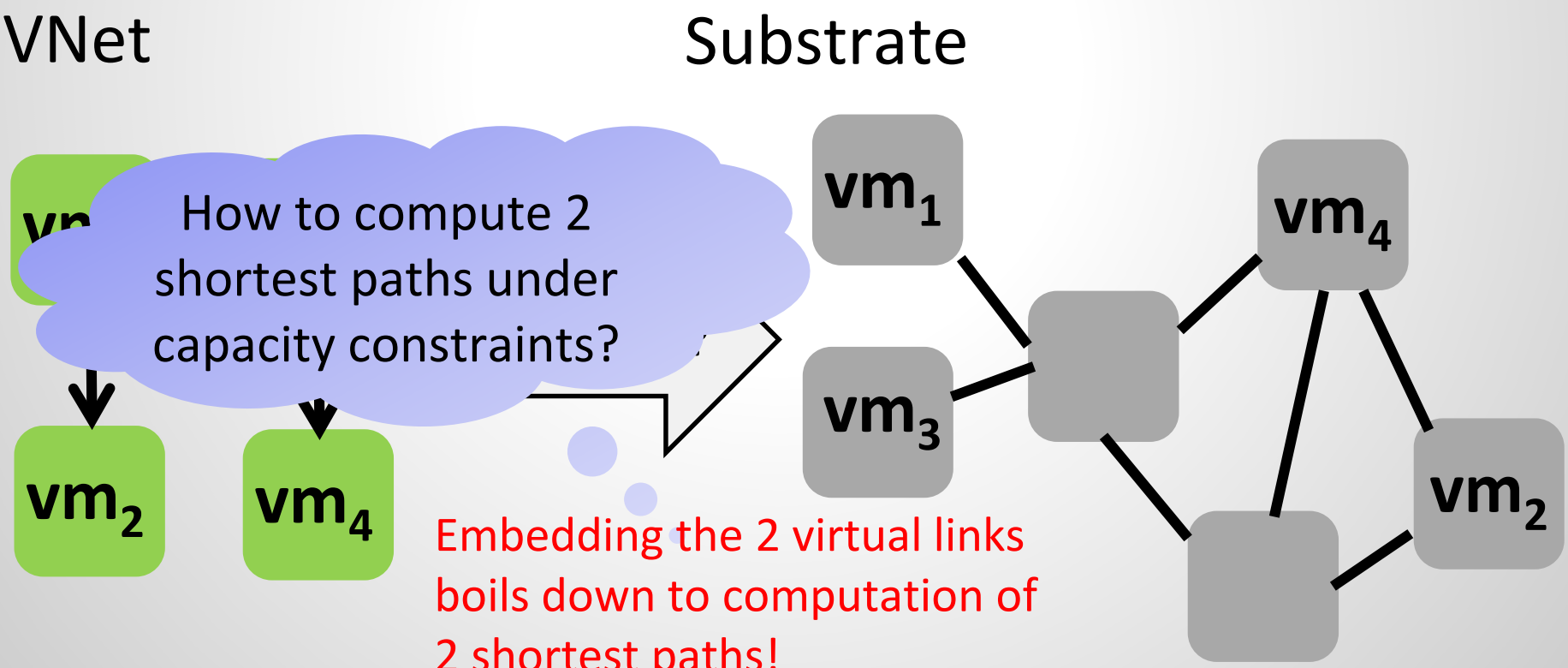
The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



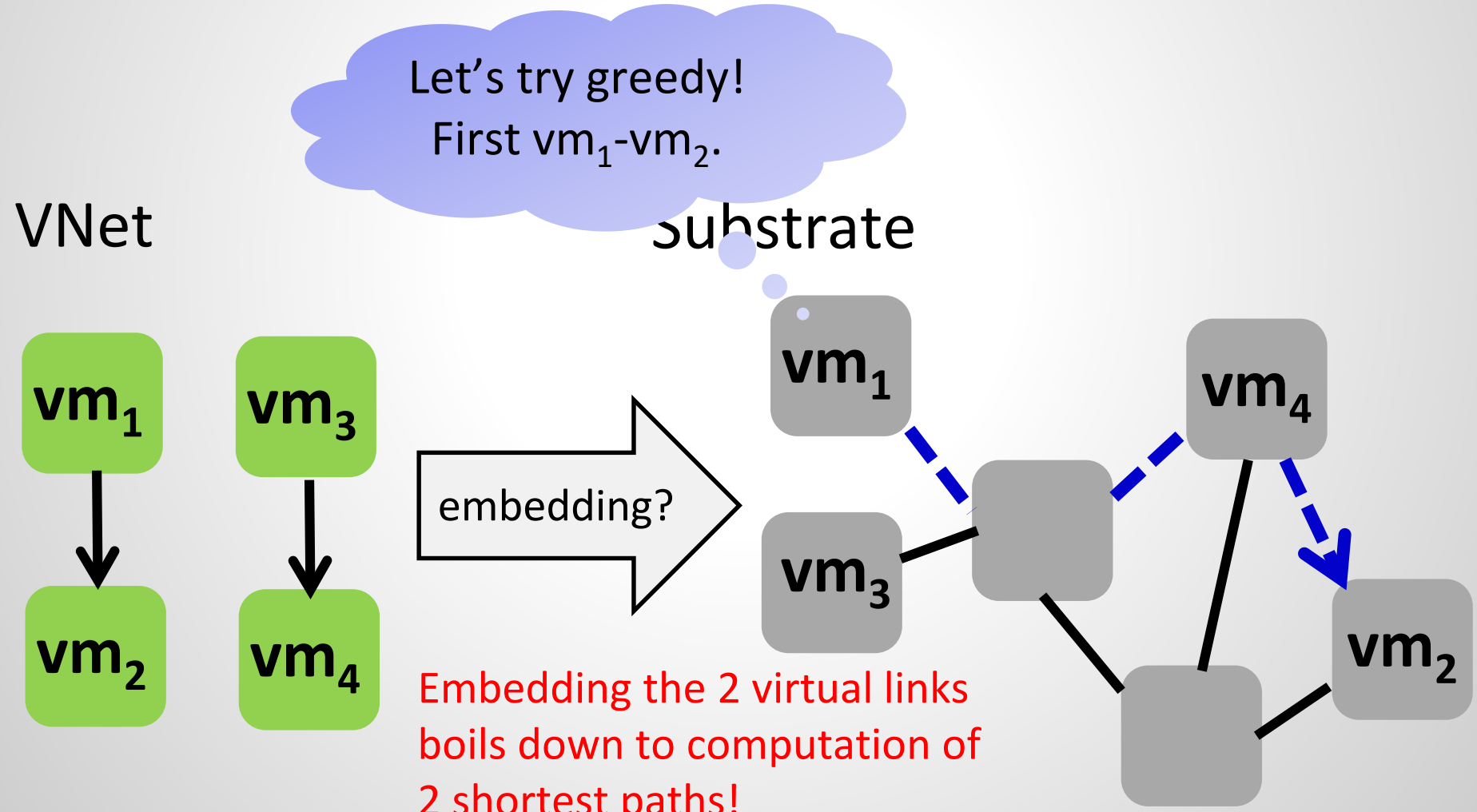
The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



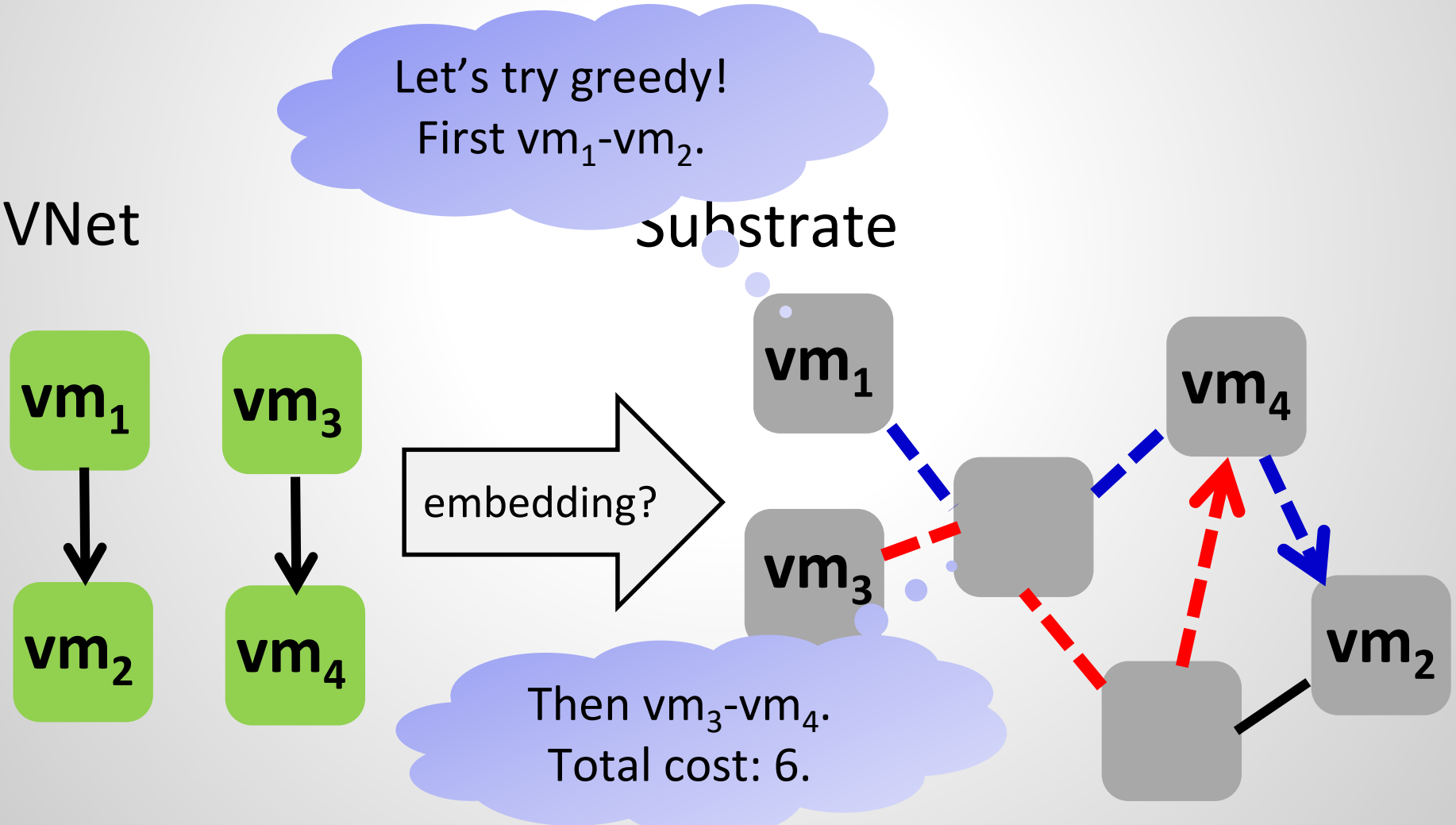
The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



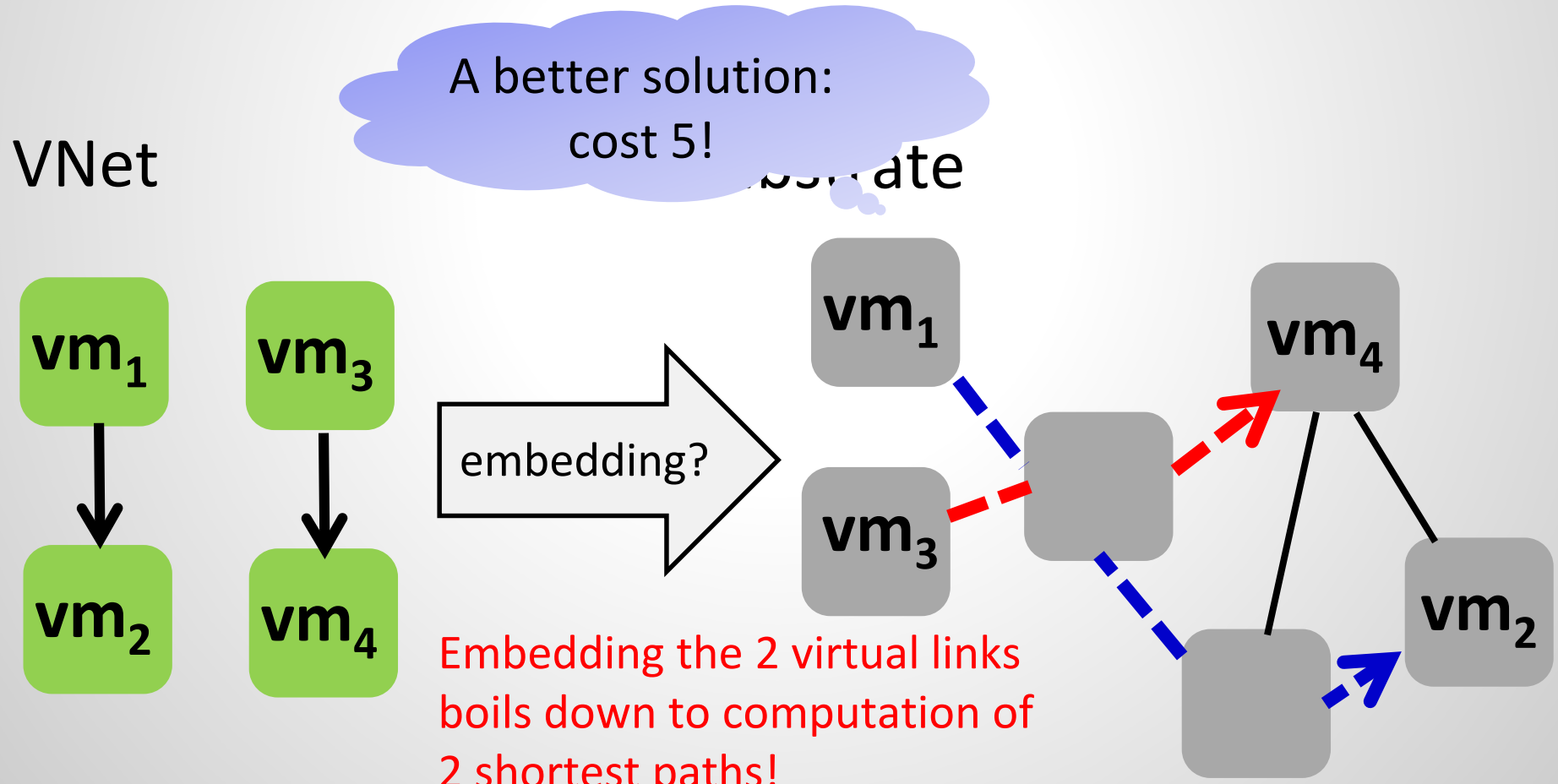
The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



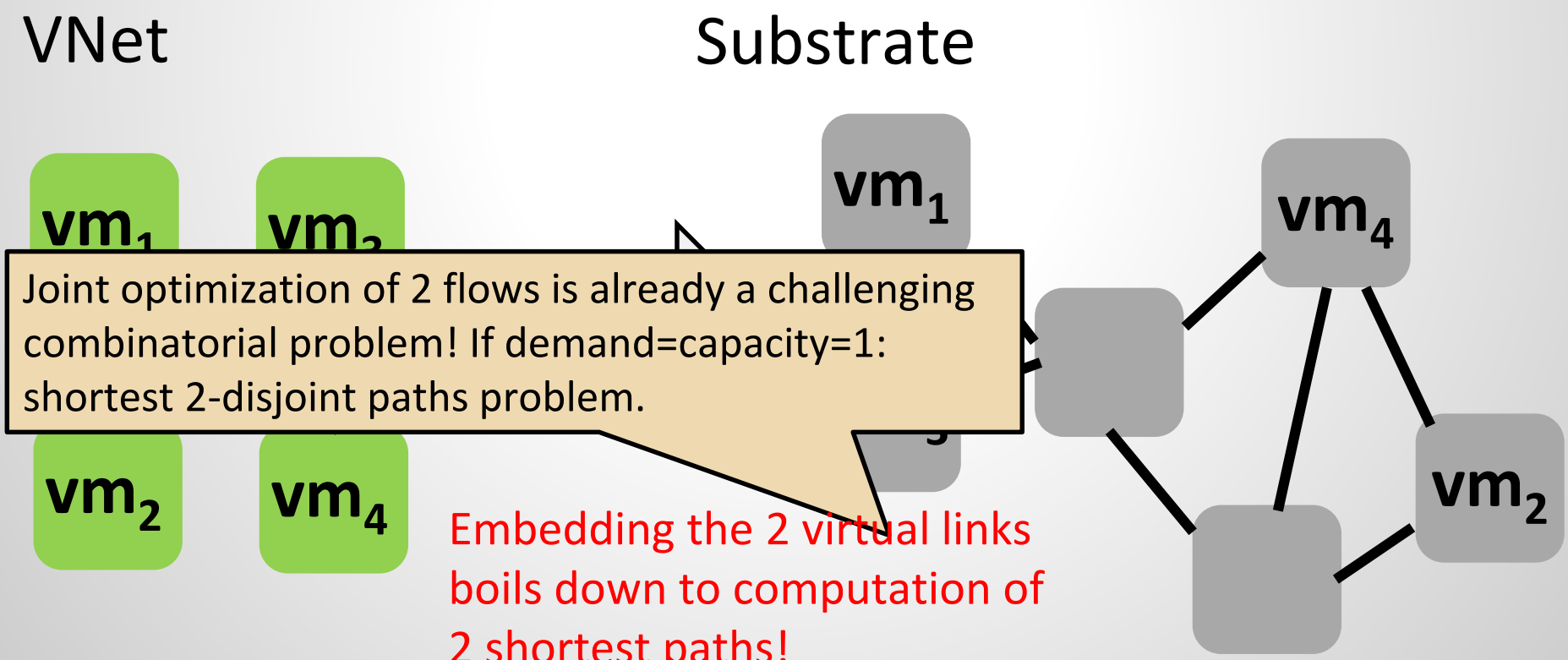
The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



The Virtual Network Embedding Problem

- Let's start simple: assume node mappings are given



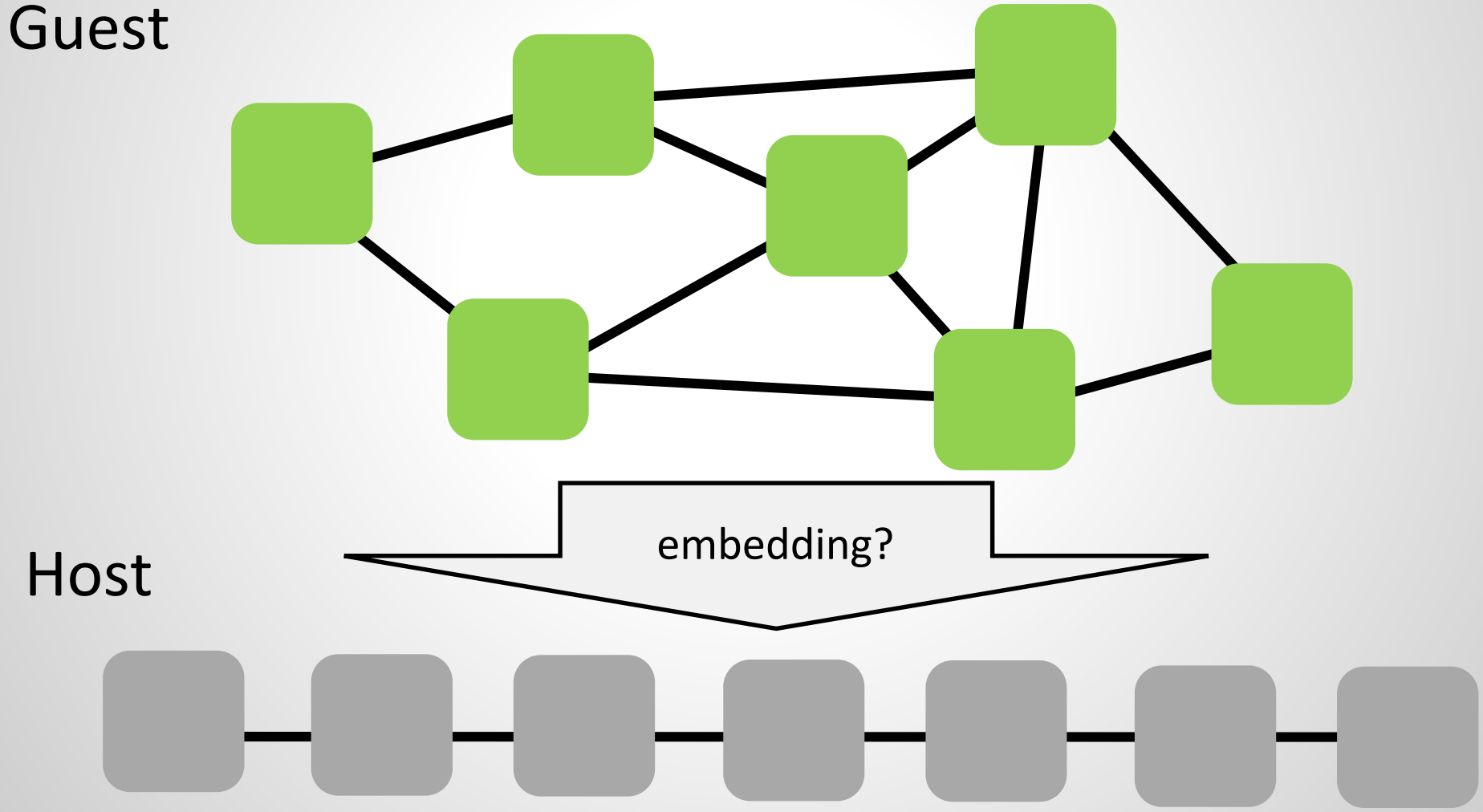
Therefore: Mapping Virtual Links is Challenging

Bad news: The Virtual Network Embedding Problem is hard
even if endpoints are already mapped and given.

But maybe at least
mapping nodes is simple?

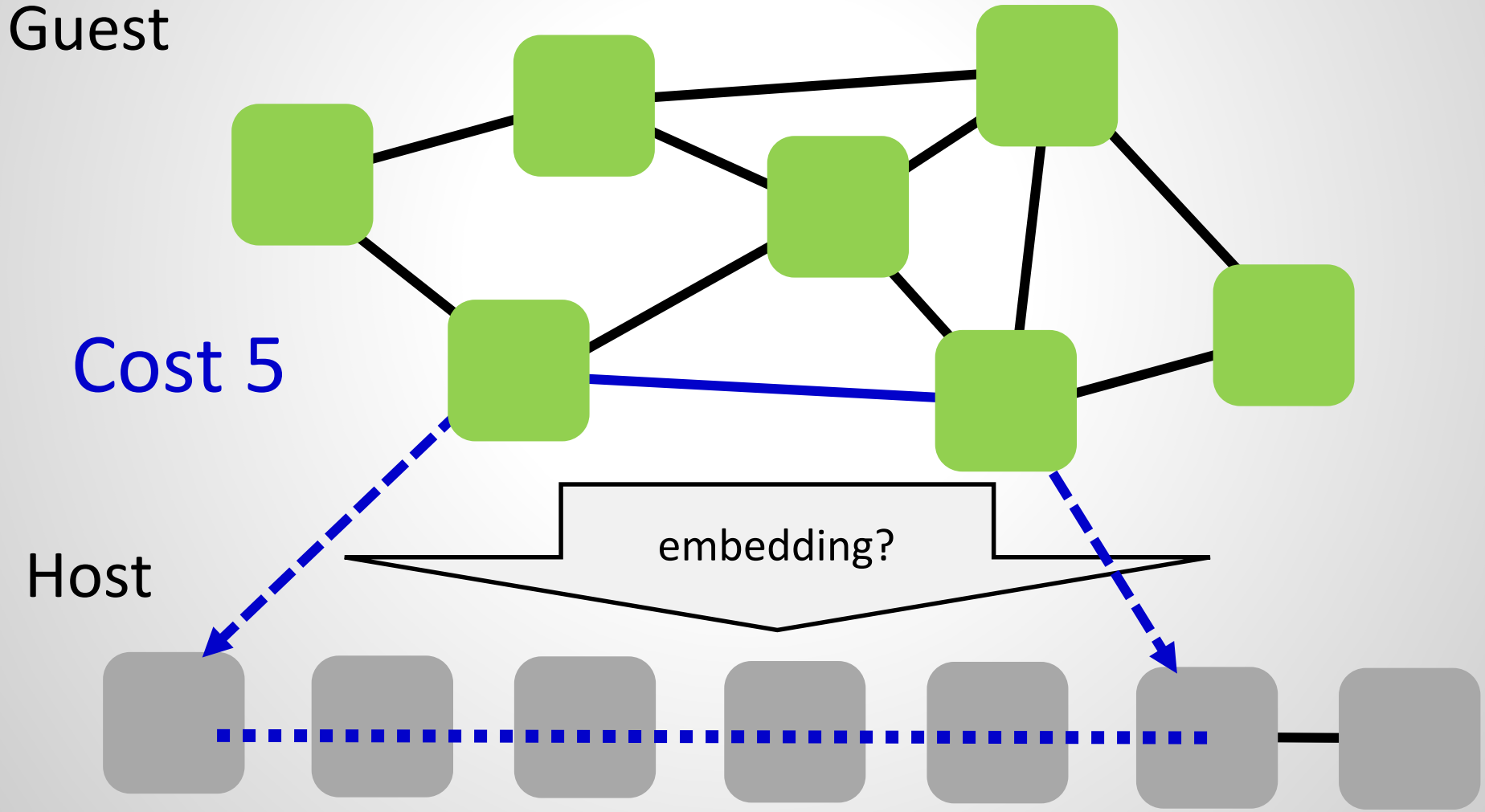
Mapping Virtual Nodes

- Let's start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line



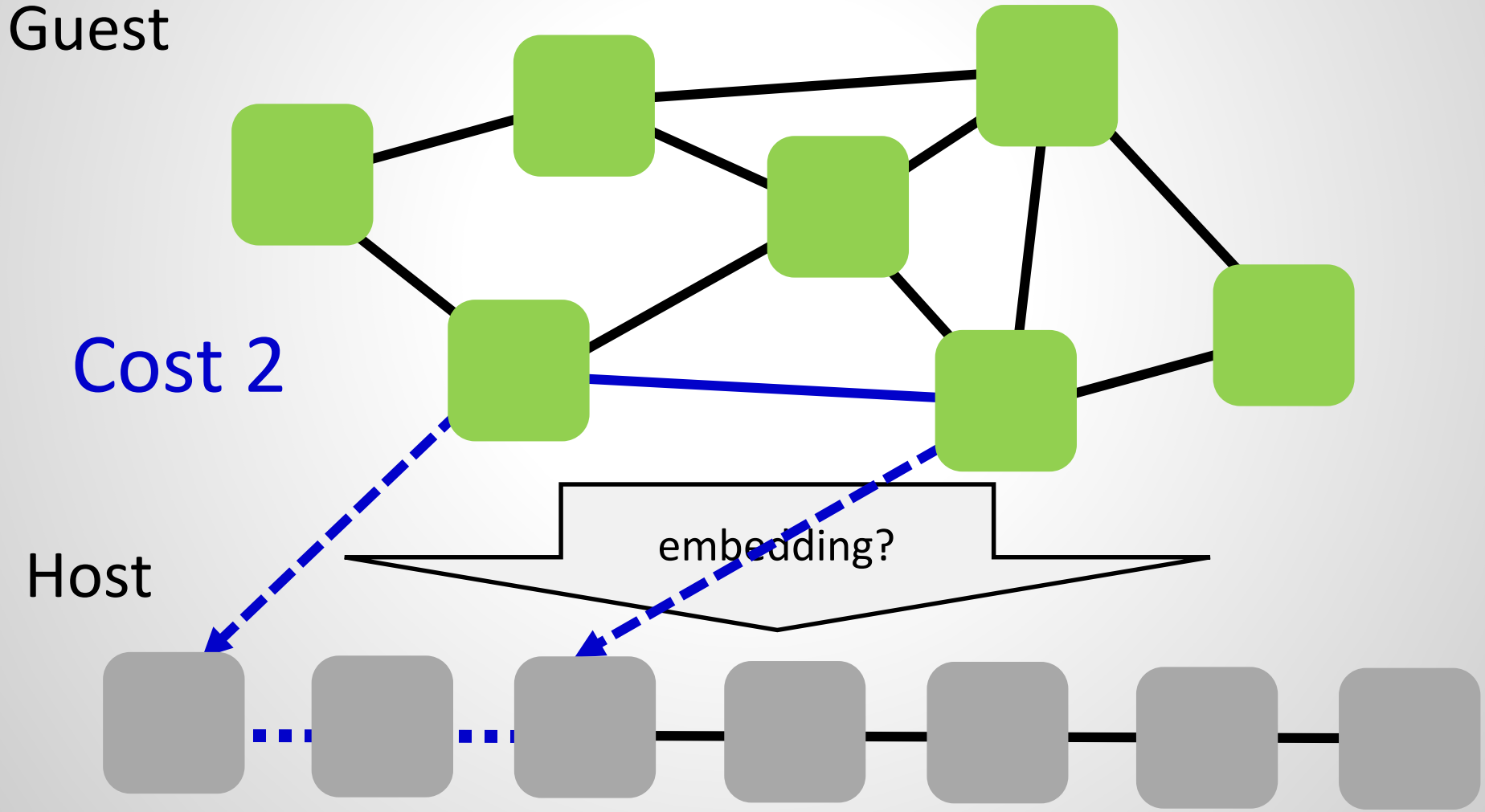
Mapping Virtual Nodes

- Let's start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line



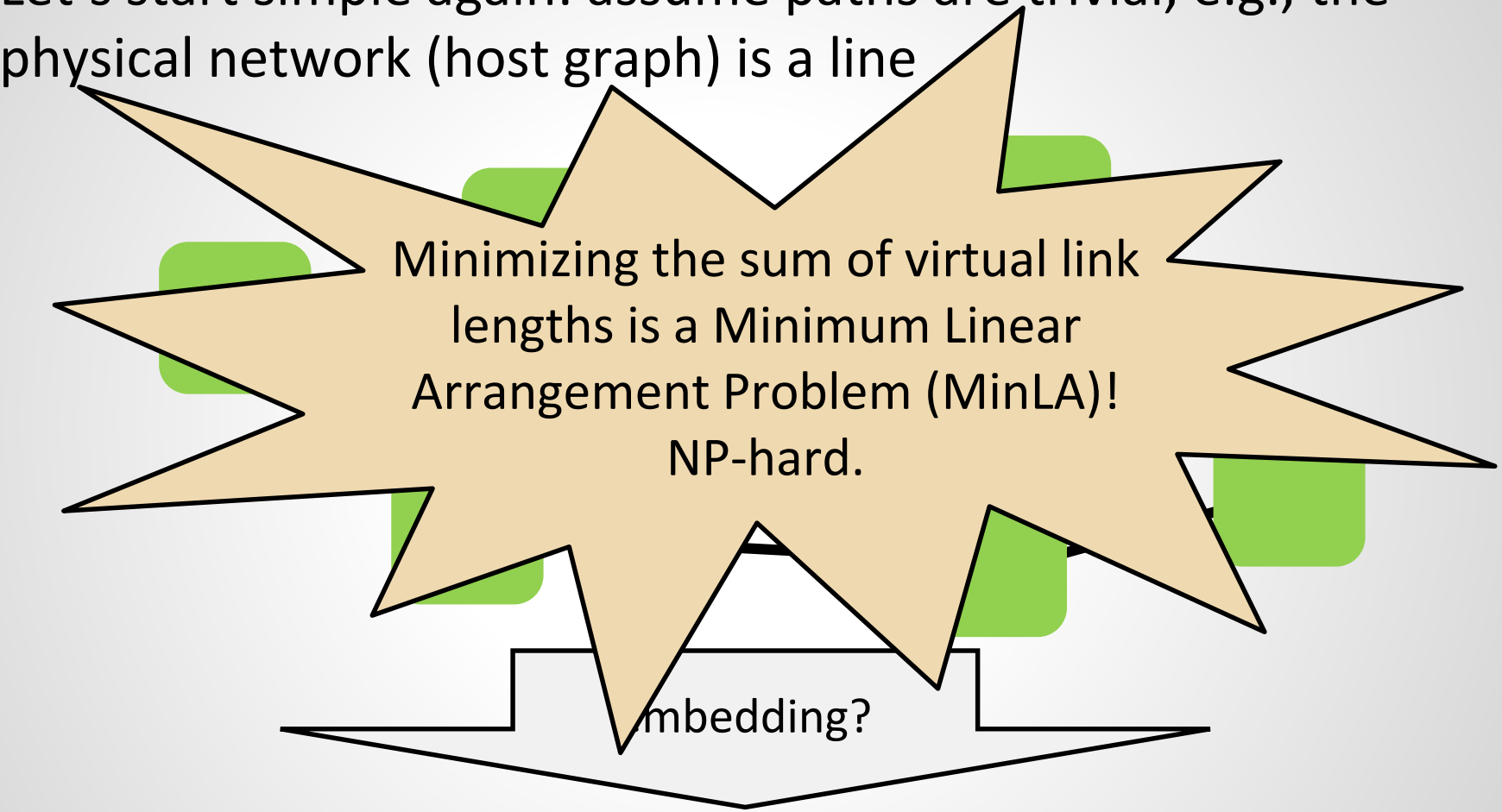
Mapping Virtual Nodes

- Let's start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line



Mapping Virtual Nodes

- Let's start simple again: assume paths are trivial, e.g., the physical network (host graph) is a line



Therefore: VNEP is Hard "in Both Dimensions"!

- We have seen examples that:
  - mapping virtual links is hard (even if nodes are given)
  - mapping virtual nodes is hard (even if links are trivial)

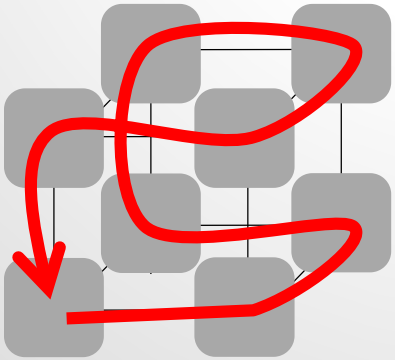
- Remark: the VNEP can also be seen as a generalization of the Subgraph Isomorphism Problem (SIP)

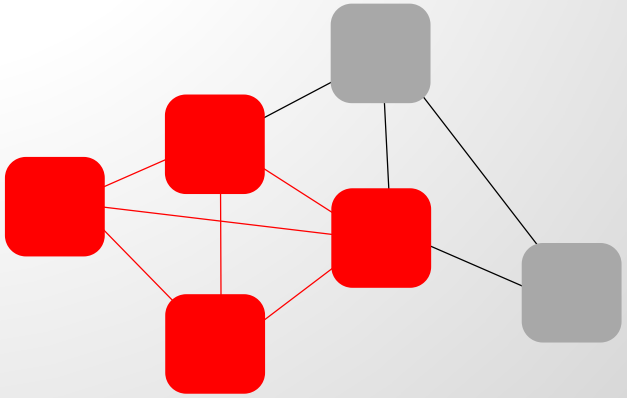
Known? Why is SIP NP-hard?

Therefore: VNEP is Hard "in Both Dimensions"!

- We have seen examples that:
  - mapping virtual links is hard (even if nodes are given)
  - mapping virtual nodes is hard (even if links are trivial)

- Remark: the VNEP can also be seen as a generalization of the Subgraph Isomorphism Problem (SIP)
  - The SIP problem: Given two graphs G,H, determine whether G contains a subgraph that is isomorphic to H?
  - NP-hard: "does G contain an n-node cycle?" is a Hamilton cycle problem (each node visited exactly once), a solution to "does G contain a k-clique?" solves maximum clique problem, etc.



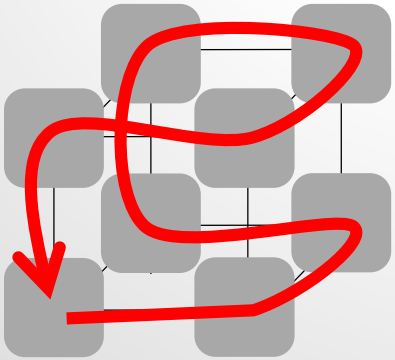


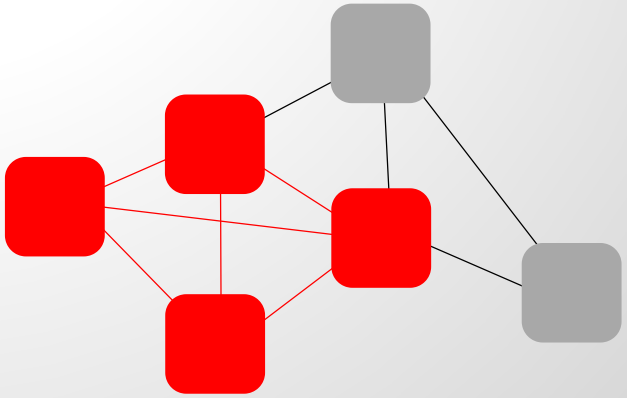
Therefore: VNEP is Hard "in Both Dimensions"!

- We have seen examples that:
  - mapping virtual links is hard (even if nodes are given)
  - mapping virtual nodes is hard (even if links are trivial)

- Remark: the VNEP can also be seen as a generalization of the Subgraph Isomorphism Problem (SIP)
  - The SIP problem: Given two graphs G,H, determine whether G contains a subgraph that is isomorphic to H?
  - N… …milton cycle problem (each node visited …?" solves maximum clique problem, etc.

So if SIP is hard, why is VNEP hard?





NP-Hardness: From SIP to VNEP

- Observe: VNEP is a generalization of SIP

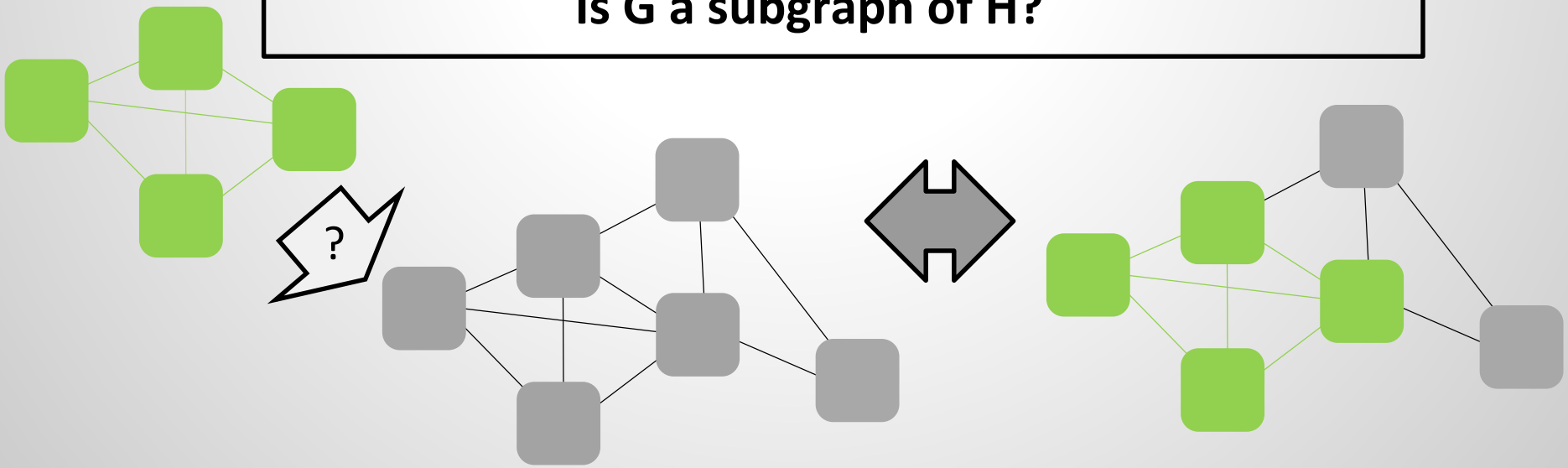
- For example:

Can VNet G=(V,E) be embedded in H at cost |E|?

(I.e., each virtual edge has length 1.)

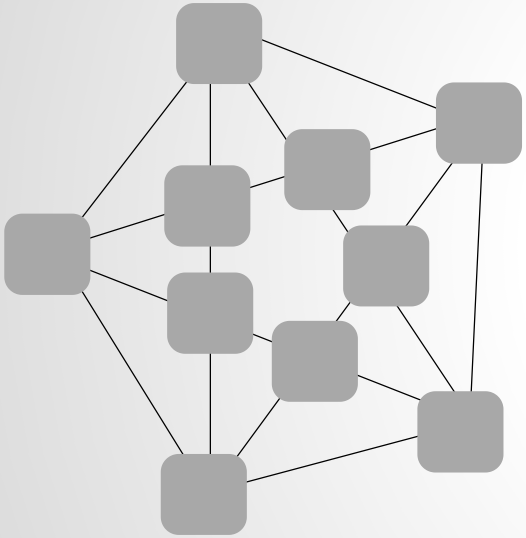
⇔

Is G a subgraph of H?

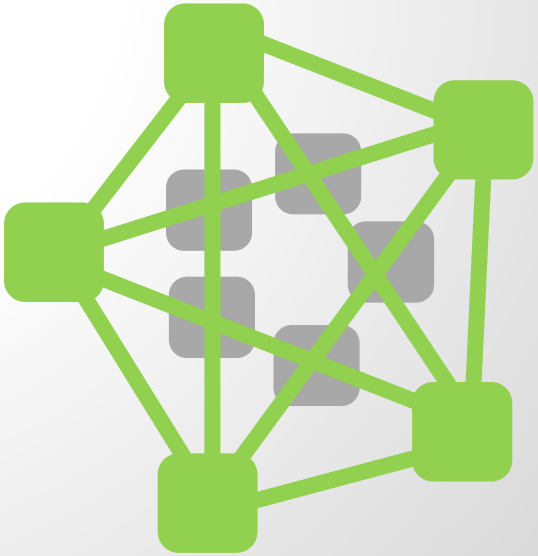


Remark: Graph Minors

Note: It is sometimes possible to embed a guest graph G on a host graph H, even though G is *not* a minor of H:



Assume planar host graph H:
K_5 and $K_{3,3}$ minor-free…



… but it is possible to embed non-planar guest graph $G=K_5$!

Can we at least formulate a “fast” MIP?

Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)
- Solved, e.g., with branch-and-bound search tree

Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize
  - Linear constraints (e.g., do not violate capacity constra

One that provides good relaxations!

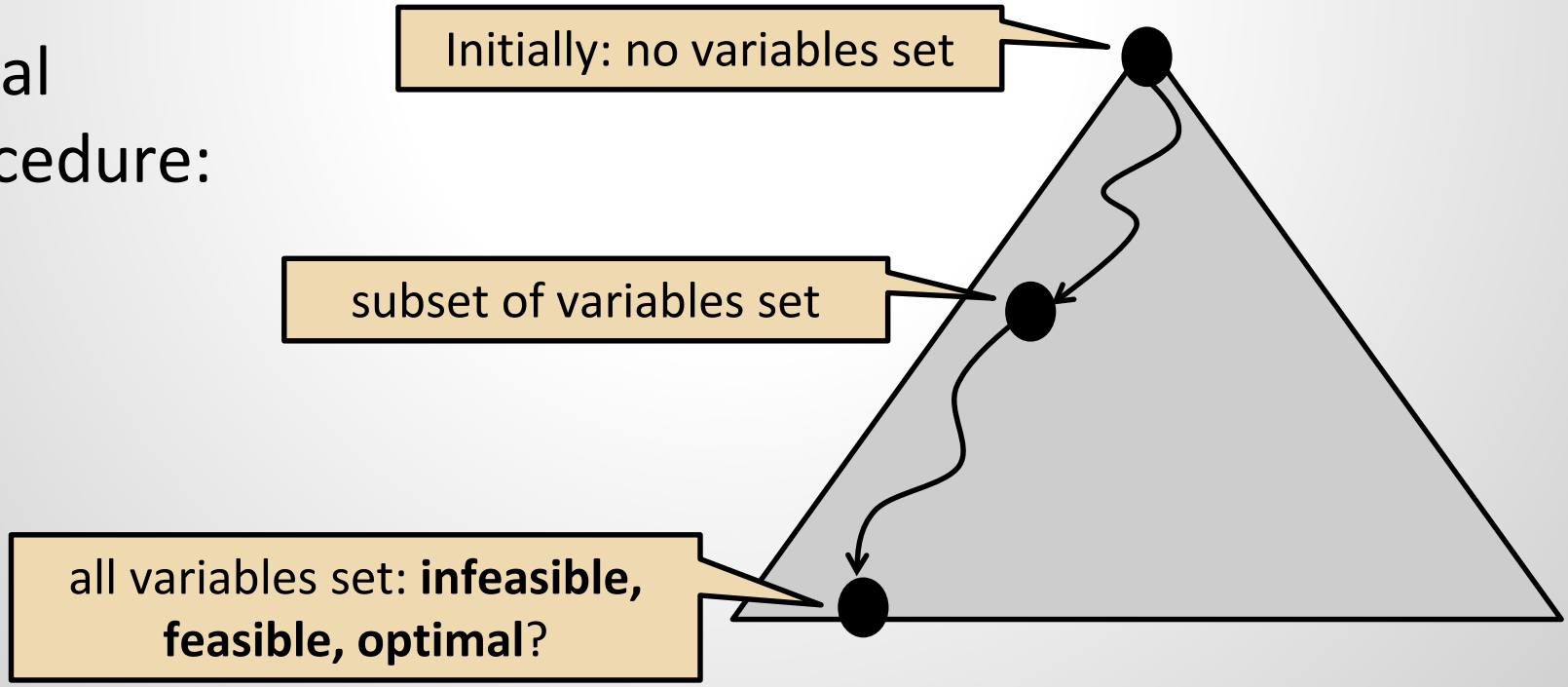
- Solved, e.g., with branch-and-bound search tree

Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual procedure:

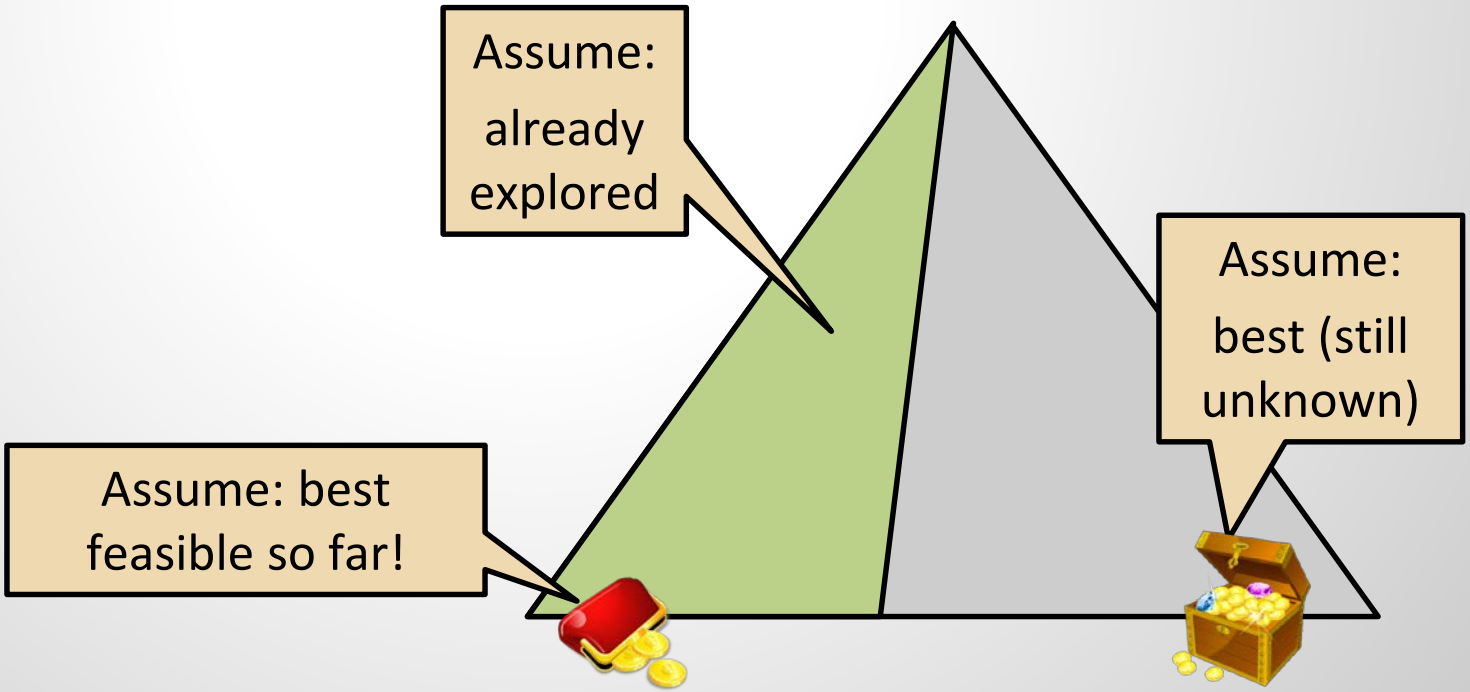


Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual procedure:

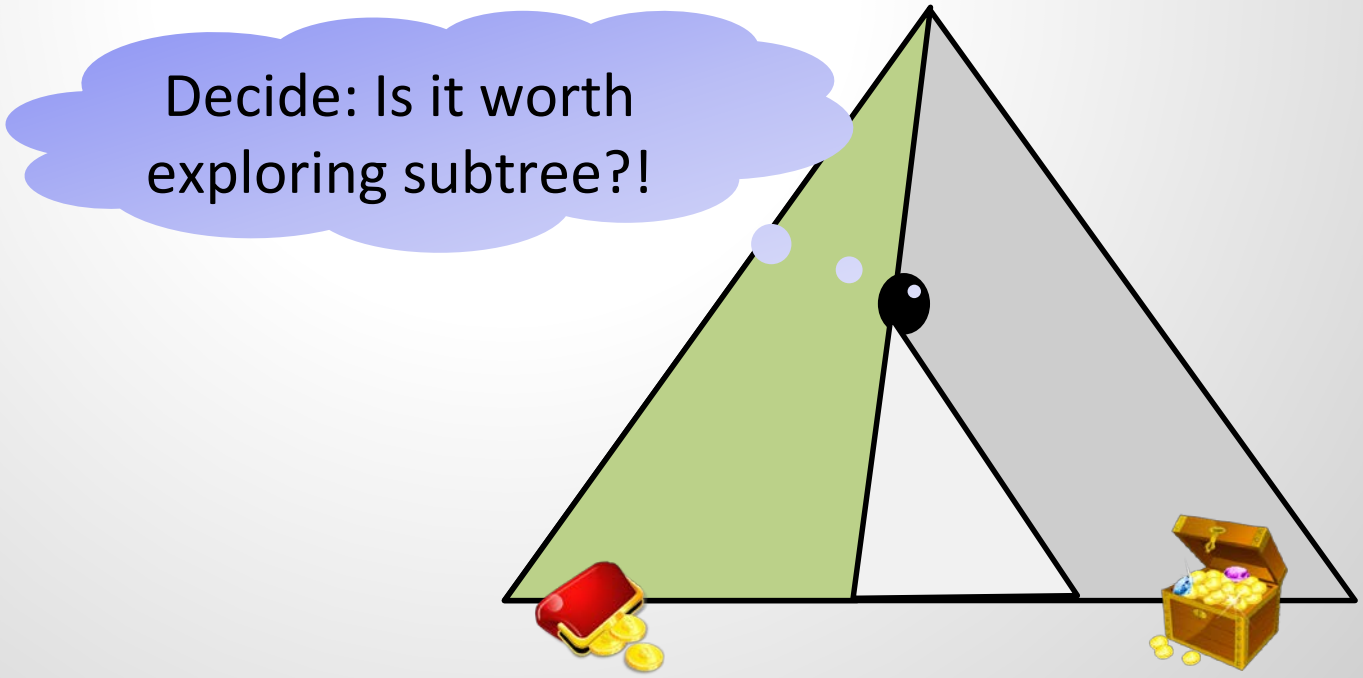


Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual procedure:

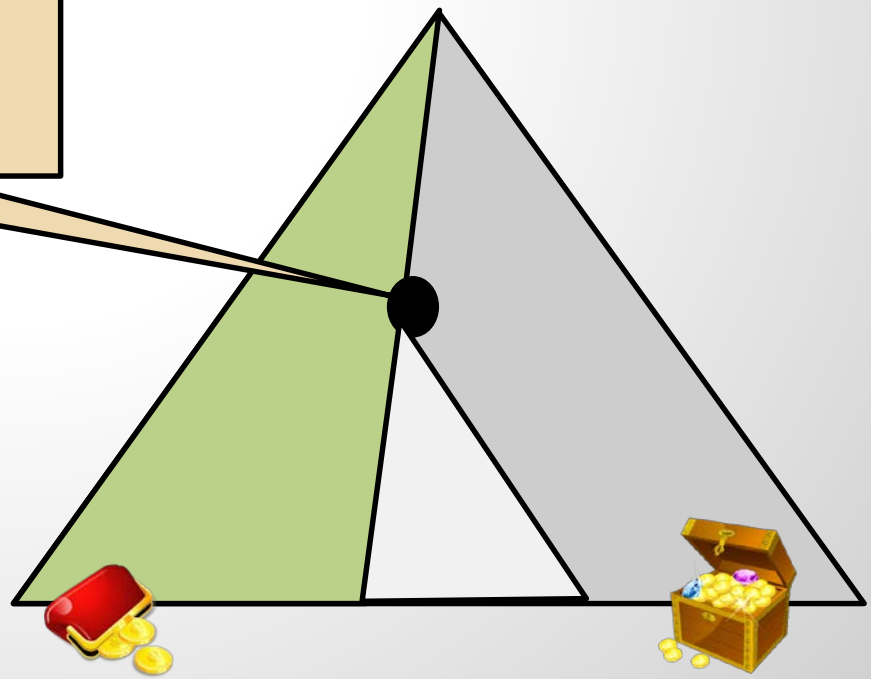


Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual trick: Relax! Solve LP (fast!), and if **relaxed solution** (more general!) **not better** then best solution so far: skip it!



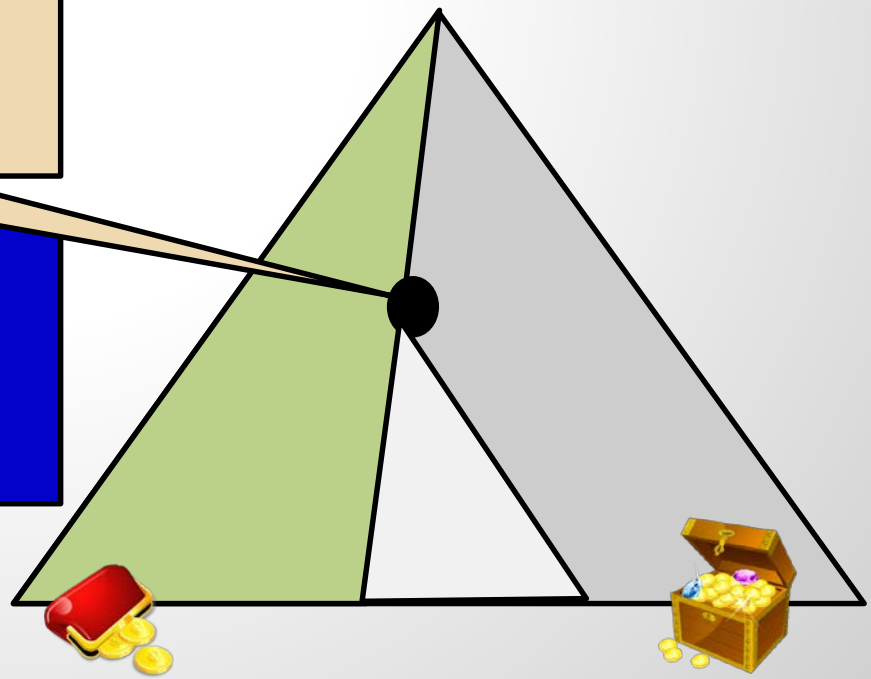
Can we at least formulate a “fast” MIP?

- Recall: Mixed Integer Program (MIP)
  - Linear objective function (e.g., minimize embedding footprint)
  - Linear constraints (e.g., do not violate capacity constraints)

- Solved, e.g., with branch-and-bound search tree

Usual trick: Relax! Solve LP (fast!), and if **relaxed solution** (more general!) **not better** then best solution so far: skip it!

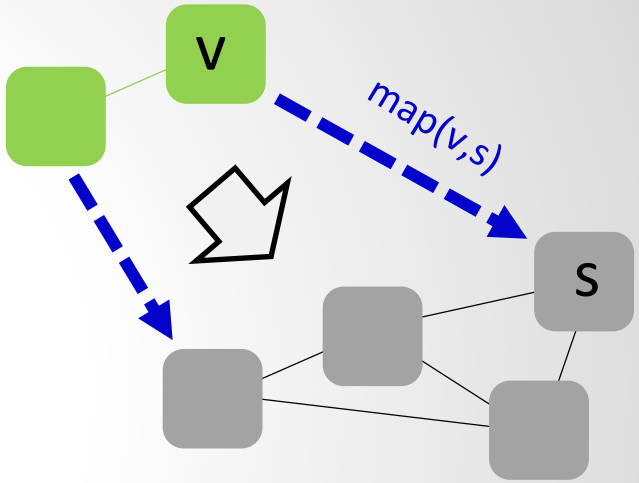
Bottomline: If MIP provides «good relaxations», large parts of the search space can be pruned.

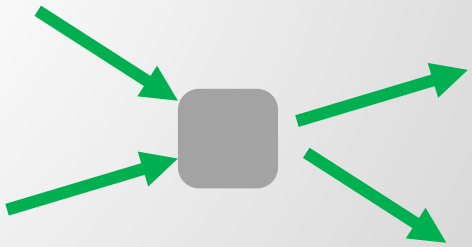


Can we at least formulate a “fast” MIP?

A typical MIP formulation:

- Introduce binary variables ***map(v,s)*** to map virtual nodes *v* to substrate node *s*
- Introduce flow variables for paths (say splittable for now)
- Ensure flow conservation: all flow entering a node must leave the node, unless it is the source or the destination





$$\Sigma_{u->v} f_{uv} = \Sigma_{v->w} f_{vw}$$

In Out

Can we at least formulate a “fast” MIP?

We get constraints like:

Assume bandwidth b requested from node s to node t.

In Out

$$\forall v: \sum_u f_{uv} - f_{vu} \geq map(s,v) * b - map(t,v) * \infty$$

What does this formula do and why is it correct?

Can we at least formulate a “fast” MIP?

We get constraints like:

Assume bandwidth b requested from node s to node t.

In Out

$$\forall v: \sum_u f_{uv} - f_{vu} \geq map(s,v) * b - map(t,v) * \infty$$

If map(s,v)=1, i.e., s mapped to v: so **flow starts at v**, and hence outgoing flow must be larger than incoming flow (plus b).

Can we at least formulate a "fast" MIP?

We get constraints like:

Assume bandwidth b requested from node s to node t.

In Out

$$\forall v: \sum_u f_{uv} - f_{vu} \geq map(s,v) * b - map(t,v) * \infty$$

If map(s,v)=0 and map(t,v)=0, i.e., **v is along the path** from s to t: then we have **flow conservation**: outgoing flow must equal incoming flow (here ≥, objective function will remove unnecessary flow).

Can we at least formulate a "fast" MIP?

We get constraints like:

Assume bandwidth b requested from node s to node t.

In Out

$$\forall v: \sum_u f_{uv} - f_{vu} \geq map(s,v) * b - map(t,v) * \infty$$

If map(t,v)=1, i.e., t mapped to v: so flow **terminates at node v**: so **no constraint**: minus infinity (but objective function will remove unnecessary flow).

Can we at least formulate a "fast" MIP?

We get constraints like:

Assume bandwidth b requested from node s to node t.

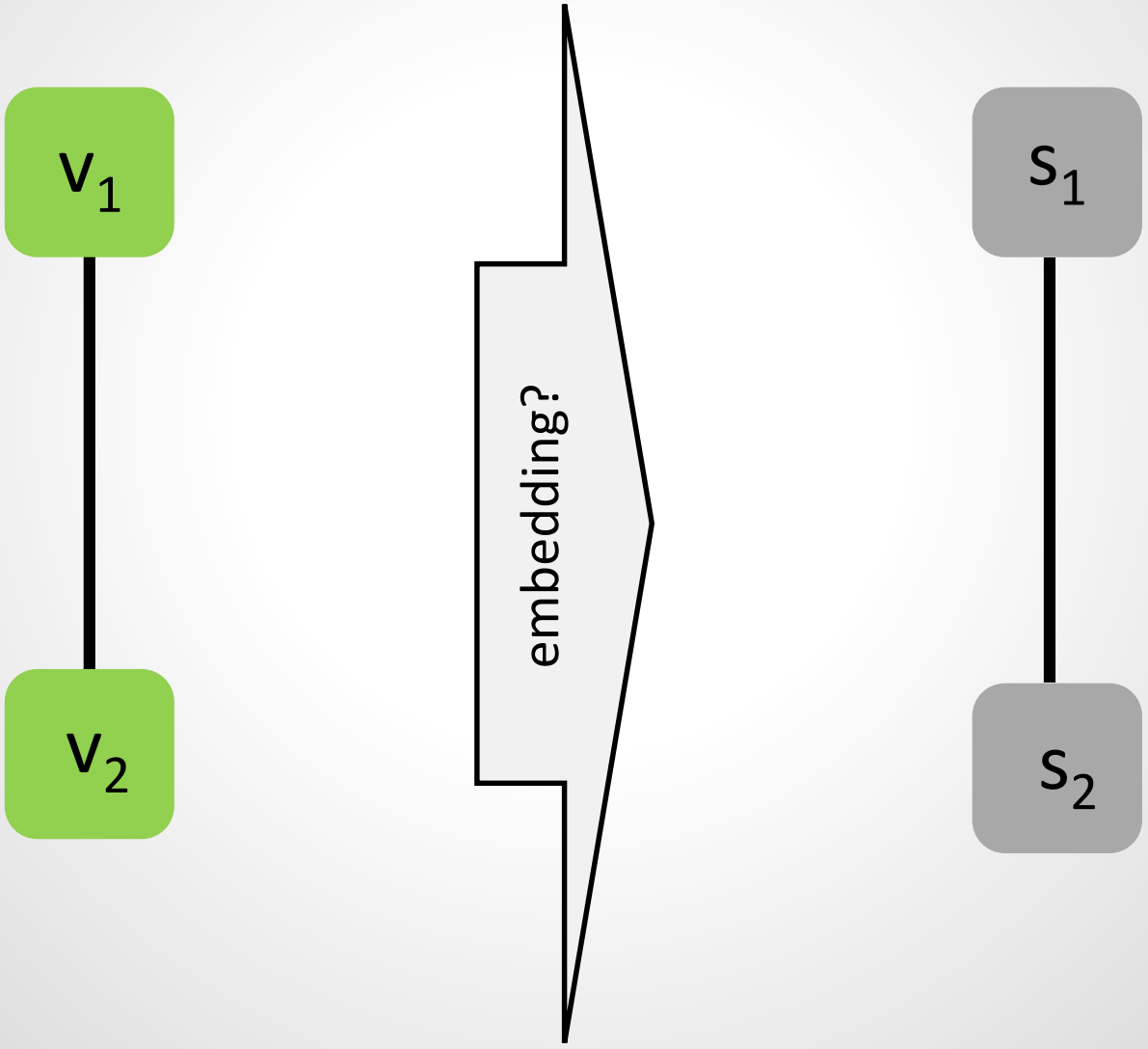
In Out

$$\forall v: \sum_u f_{uv} - f_{vu} \geq map(s,v) * b - map(t,v) * \infty$$

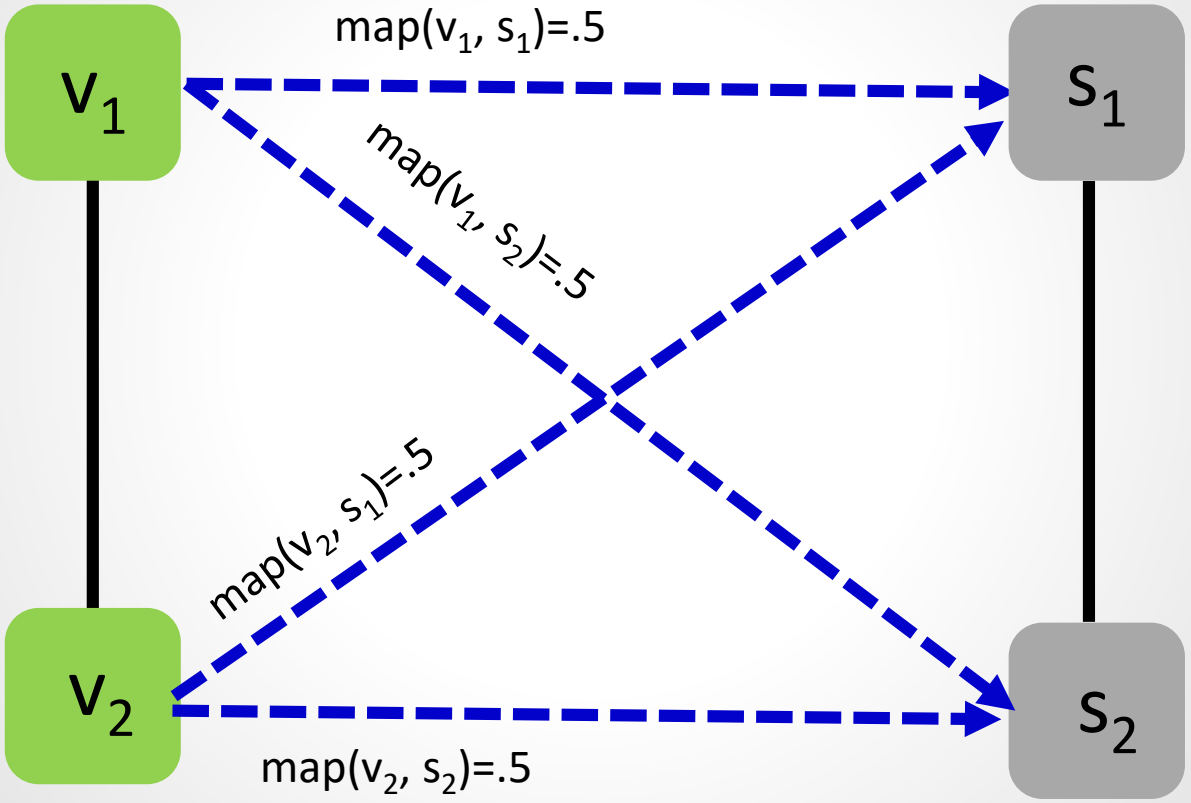
If map(t,v)=1, i.e., t mapped to v: so flow **terminates at node v**: so **no constraint**: minus infinity (but objective function will remove unnecessary flow).

Will such a MIP provide effective pruning?

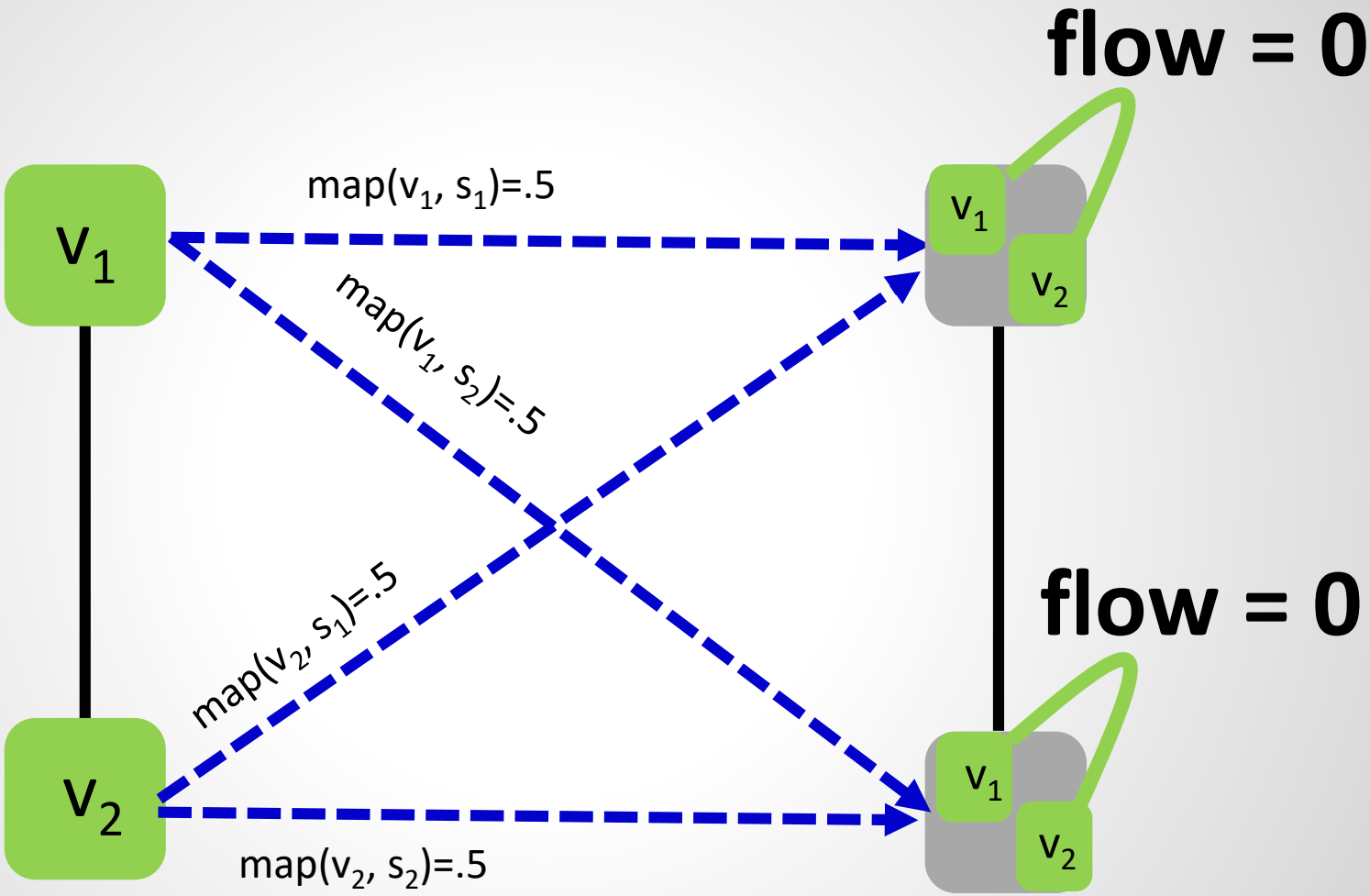
What will happen in this example?



What will happen in this example?



What will happen in this example?



Minimal flow = 0: fulfills flow conservation! Relaxation useless: does not provide any lower bound or indication of good mapping!

Remark: What about using randomized rounding?

An approximation approach which:

- … computes a solution to the linear relaxation of the IP,
- … decomposes this solution into convex combinations of elementary solutions, and
- … probabilistically chooses any of the elementary solutions based on their weight.

What about using randomized rounding?

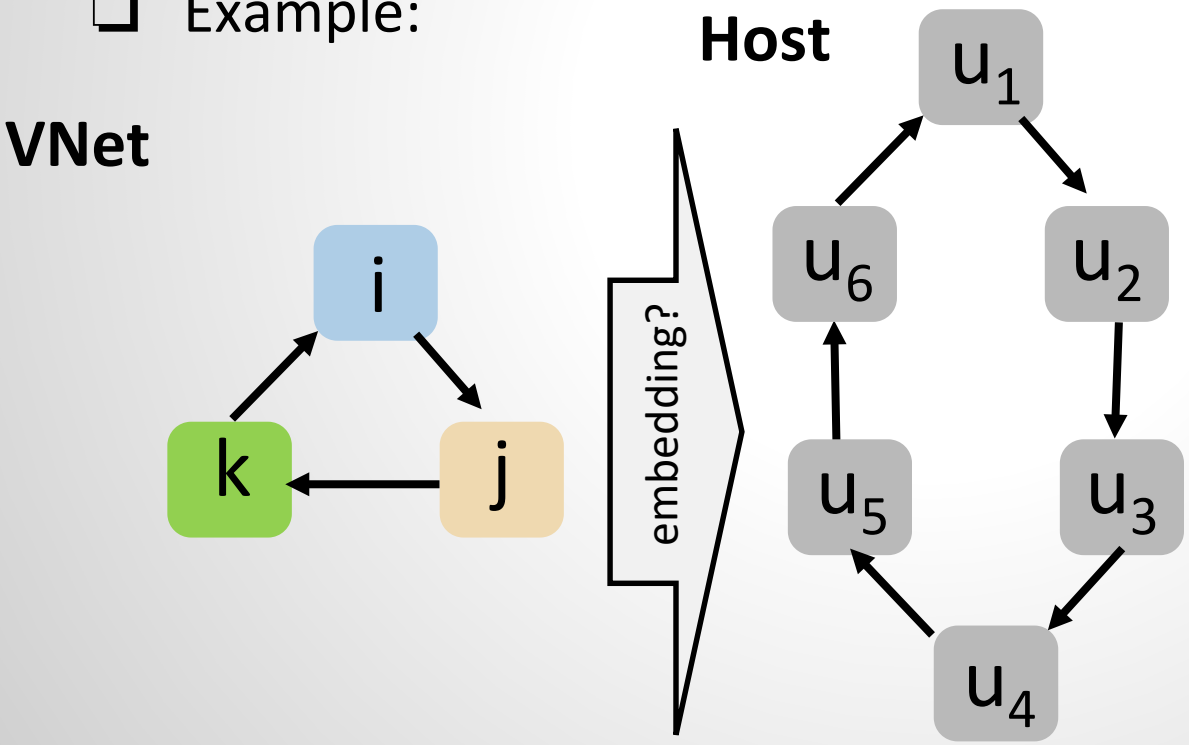
- Problem 1: relaxed solutions may not be very meaningful
  - see example for splittable flows before
- Problem 2: also for unsplittable flows, if using a standard Multi-Commodity Flow (MCF) formulation of VNEP, the integrality gap can be huge
  - Tree-like VNets are still ok
  - VNets with cycles: randomized rounding not applicable, since problem not decomposable

The linear solutions can be decomposed into convex combinations of valid mappings.

Non-Decomposability

- Relaxations of classic MCF formulation cannot be decomposed into convex combinations of valid mappings (so we need different formulations!)

- Example:

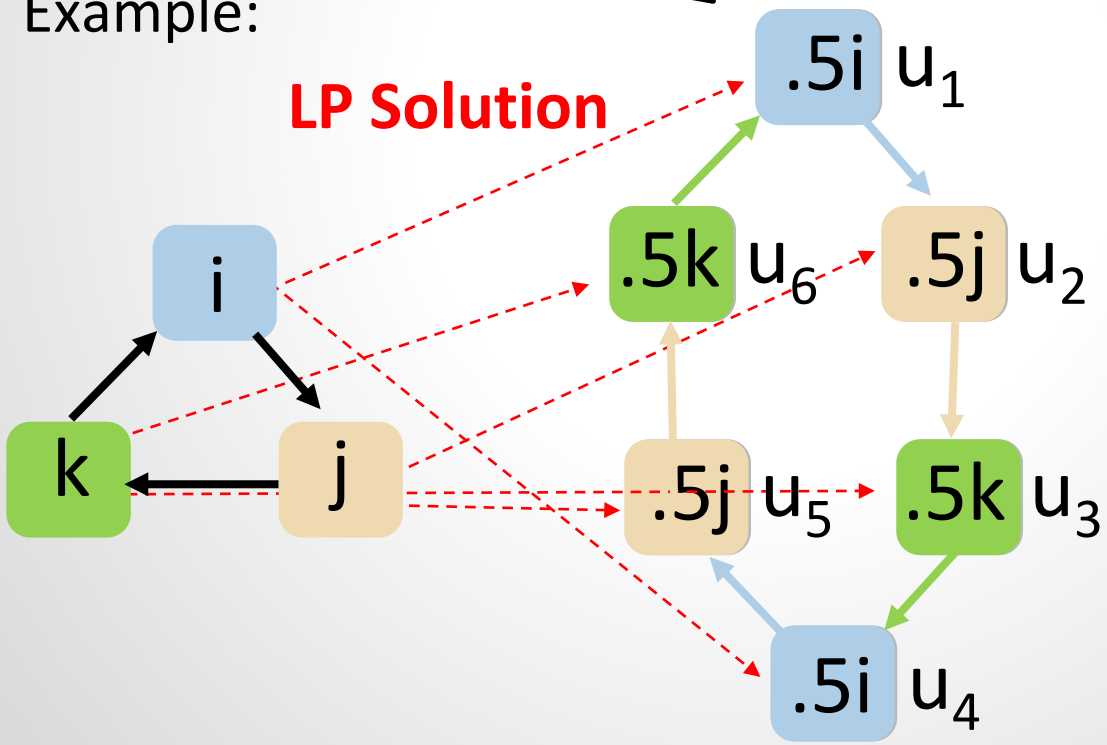


Non-Decomposability

- [redacted]rmulation cannot be decomposed [redacted] valid mappings (so we need

Valid LP solution: virtual node mappings sum to 1 and each virtual node connects to its neighboring node **with half a unit of flow**…

- Example:



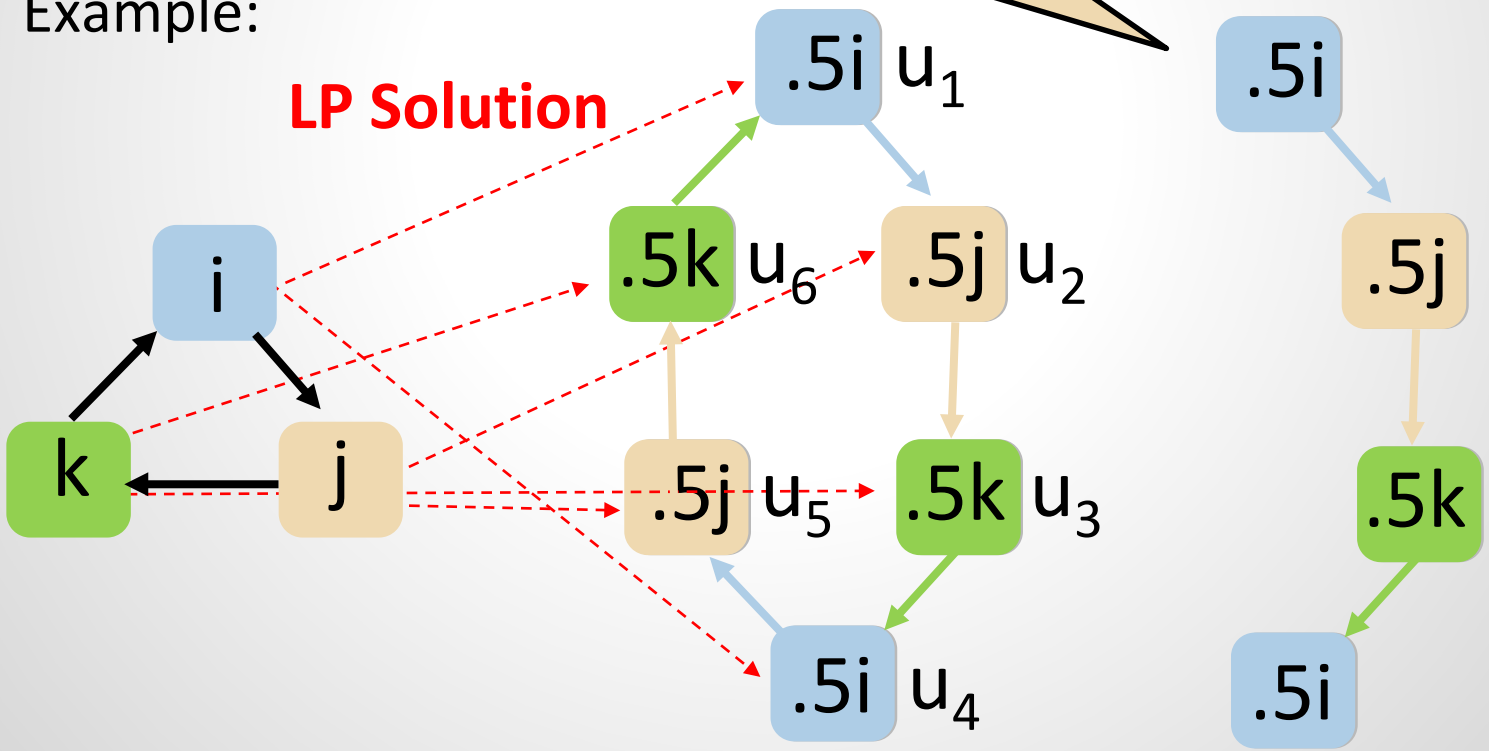
Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node i is mapped to u_1 and the only neighboring node that hosts j is u_2, so i must be fully mapped on u_1 and j on u_2. Similarly, k must be mapped on u_3. But flow of virtual edge (k,i) leaving u_3 only leads to u_4, so **i must be mapped on both u_1 and u_4**. This is **impossible**, even if **capacities are infinite**.

not be decomposed

s (so we need

Partial Decomposition

❑ Example:



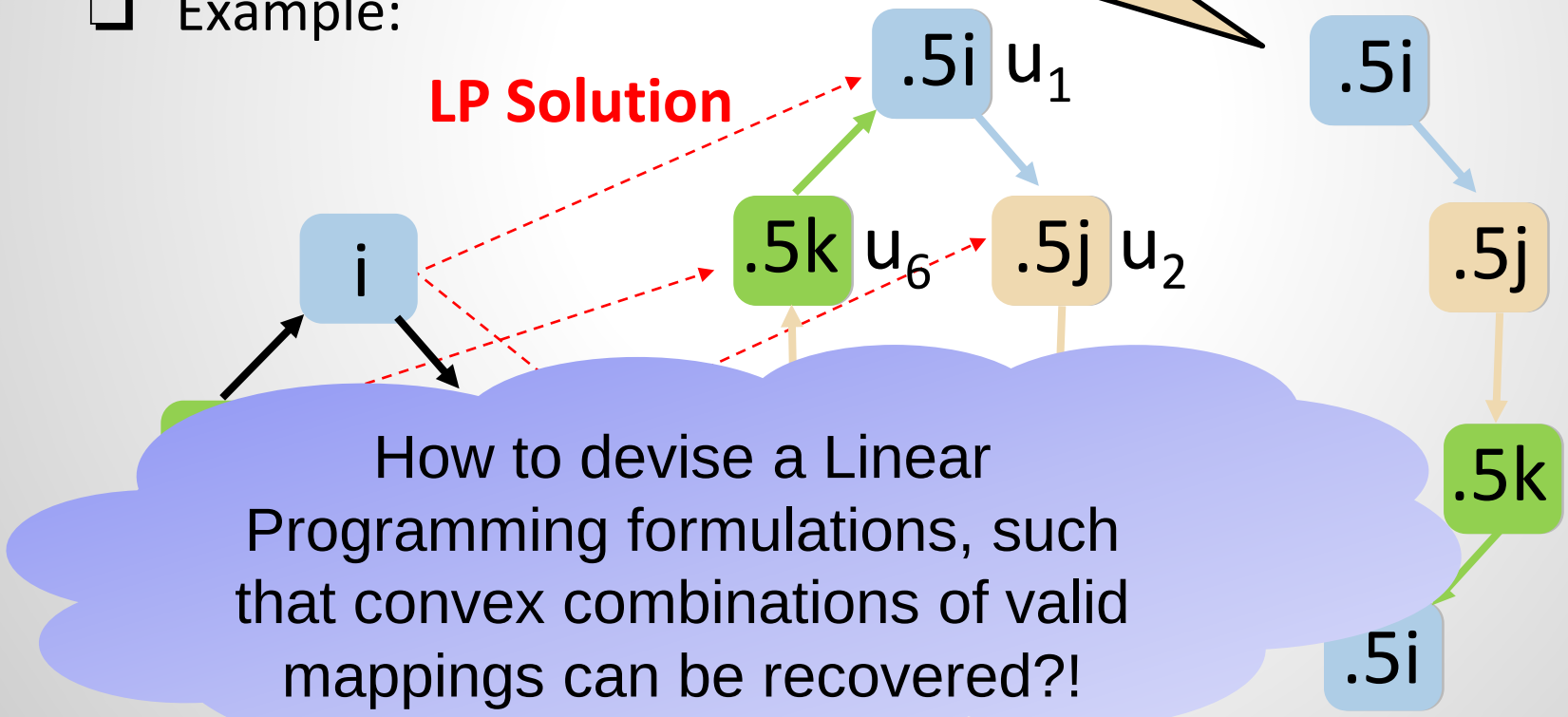
Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node i is mapped to u_1 and the only neighboring node that hosts j is u_2, so i must be fully mapped on u_1 and j on u_2. Similarly, k must be mapped on u_3. But flow of virtual edge (k,i) leaving u_3 only leads to u_4, so **i must be mapped on both u_1 and u_4**. This is **impossible**, even if **capacities are infinite**.

not be decomposed

s (so we need

Partial Decomposition

❑ Example:

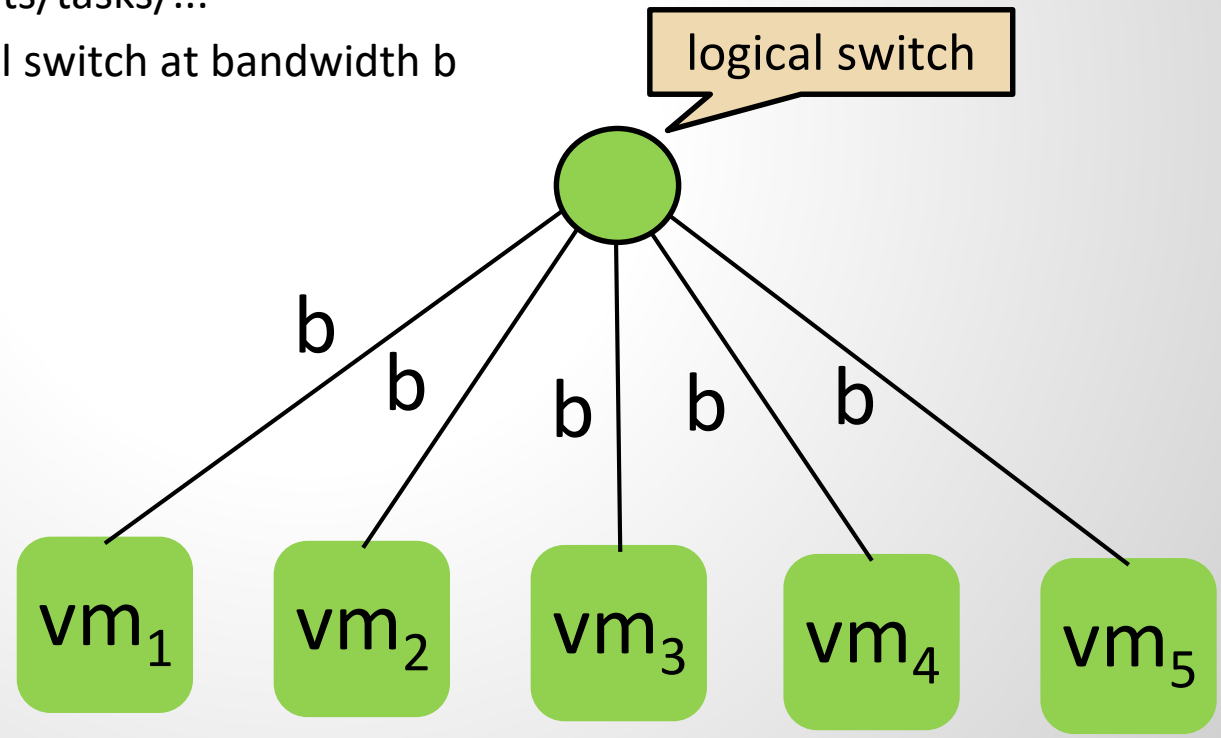



That's all Folks!

Wait a minute!
These problems need to be solved!
And they often can, even with guarantees.

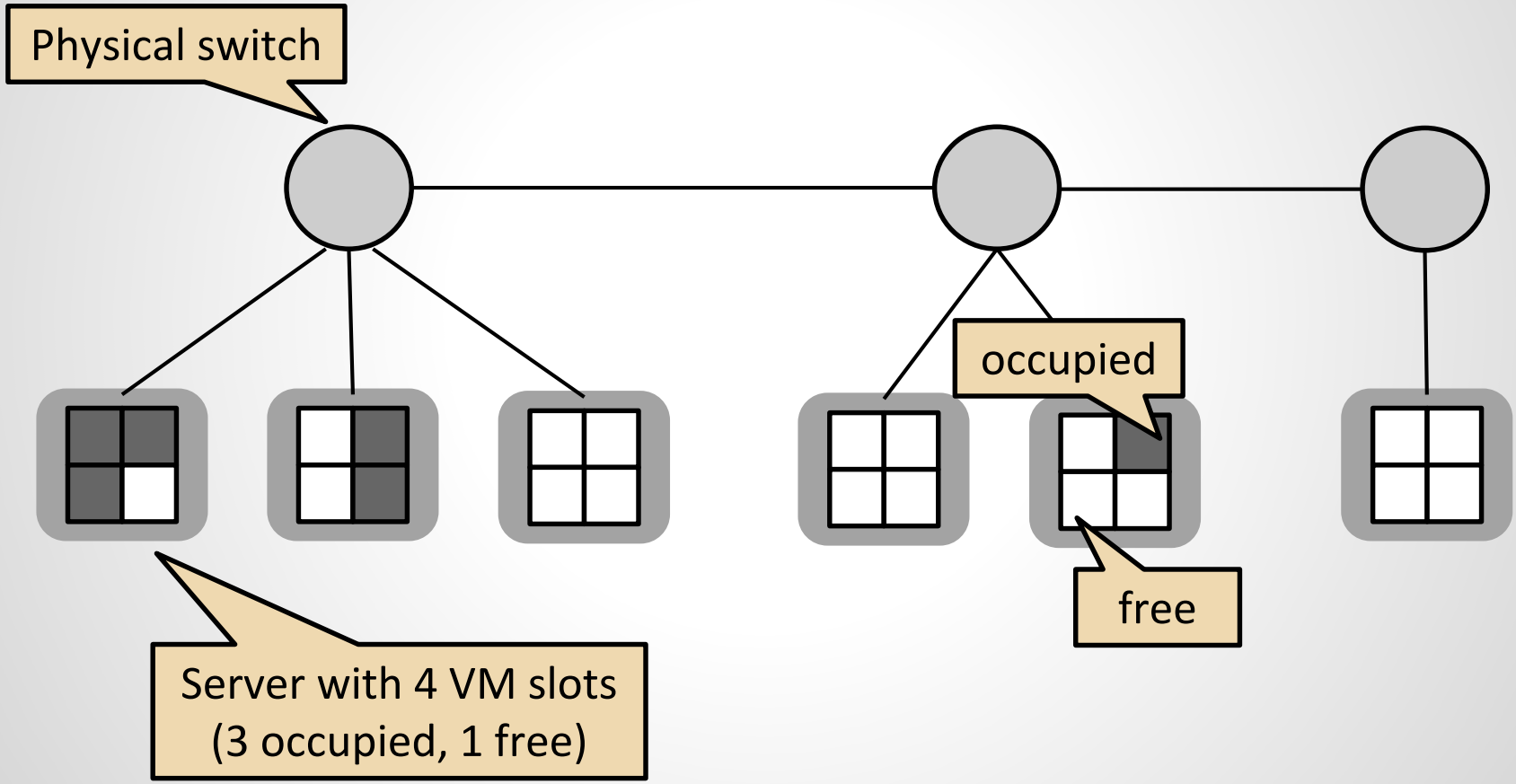
Theory vs Practice: In Practice There is Hope!

- Guest graphs may not be general graphs, but e.g., virtual clusters: very simple and symmetric, used in context of batch processing
  - k VMs/compute-units/tasks/...
  - Connected to virtual switch at bandwidth b



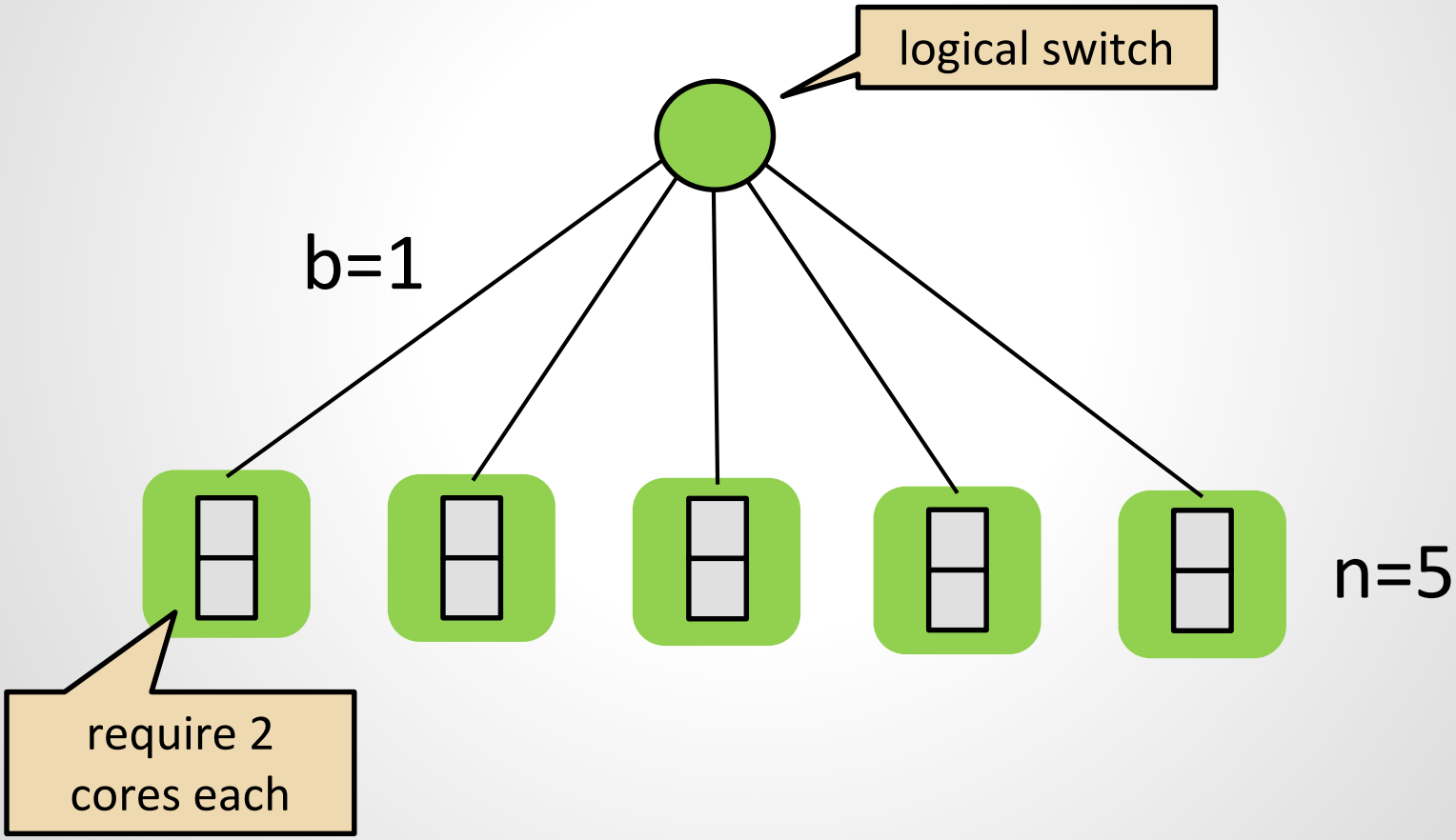
How to Embed a Virtual Cluster?

Consider host graph:

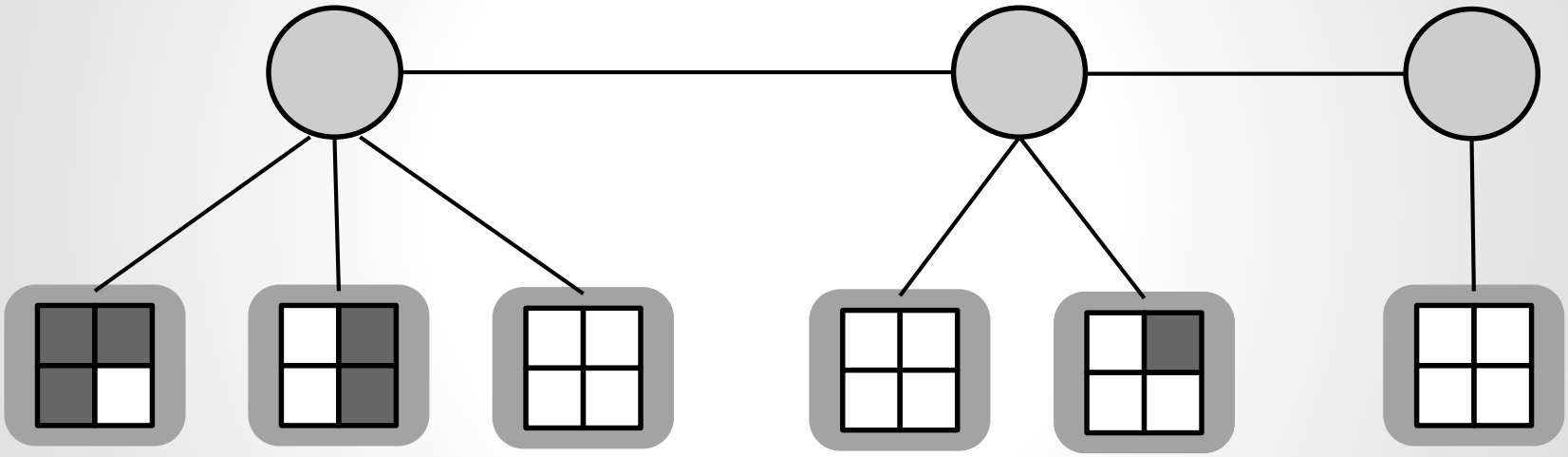


How to Embed a Virtual Cluster?

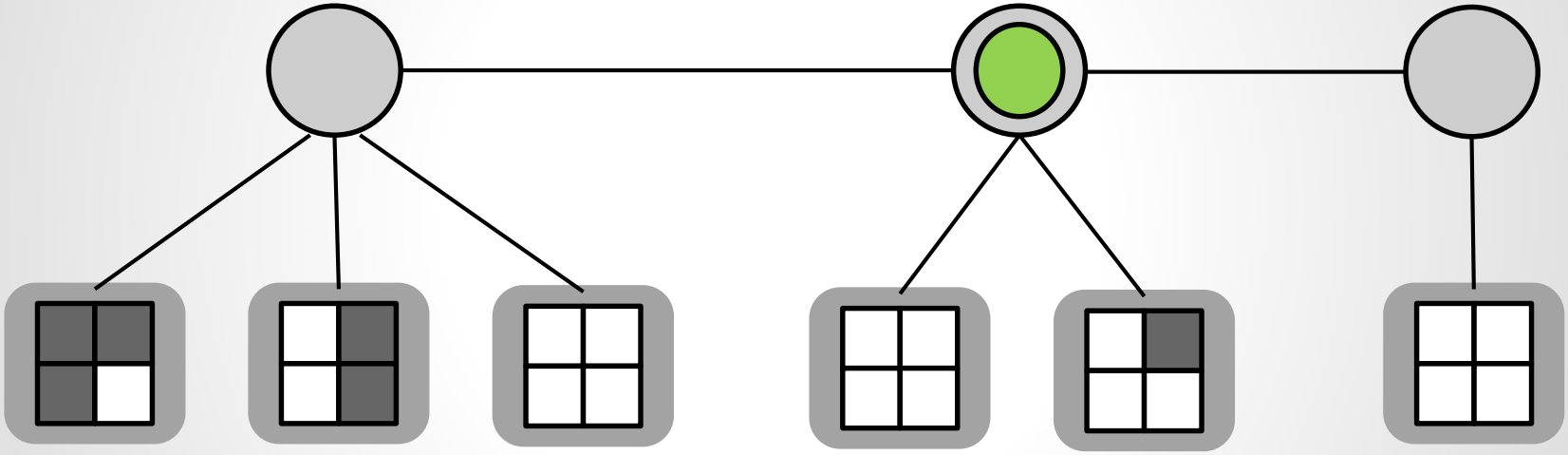
Consider guest graph:



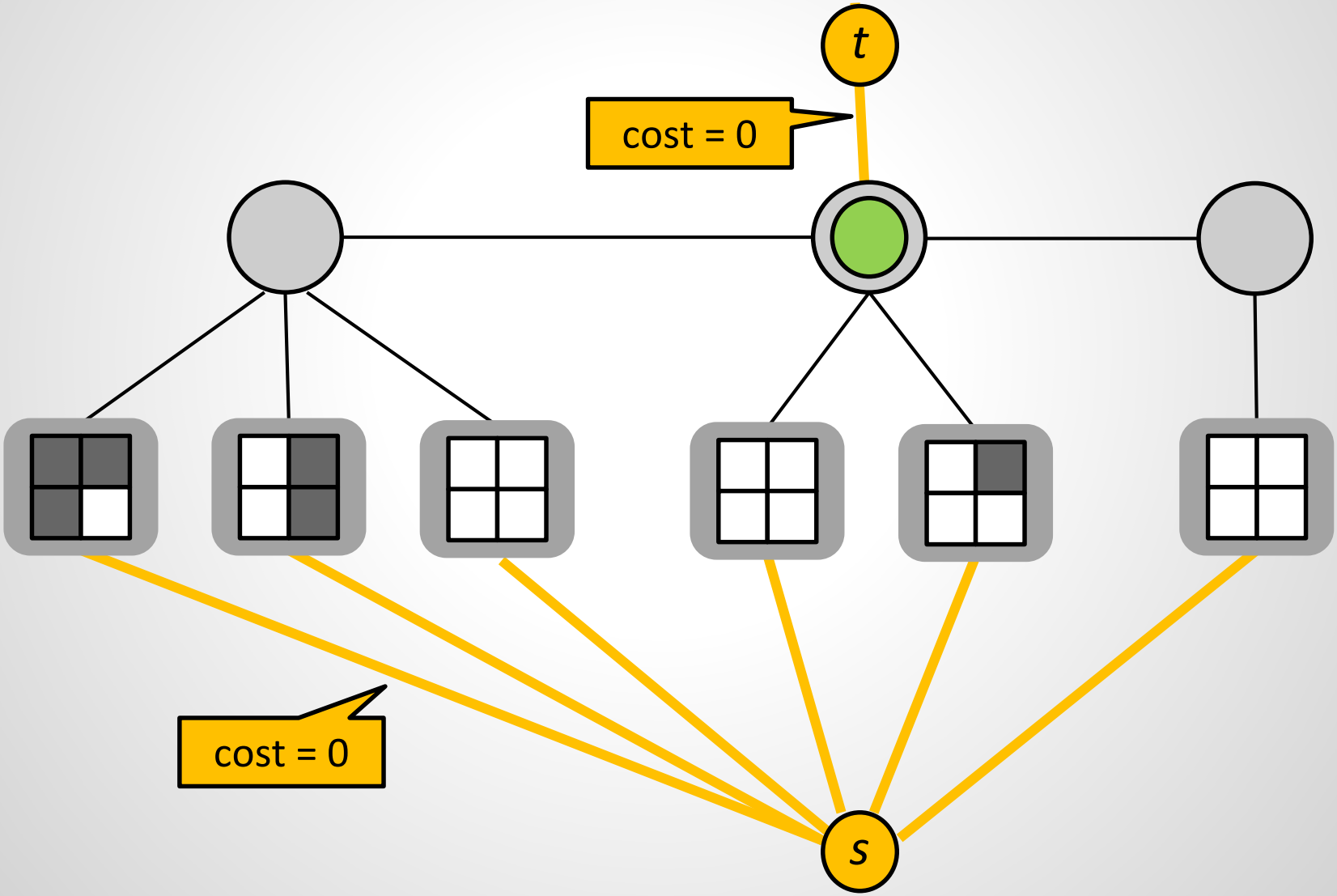
How to Embed a Virtual Cluster?



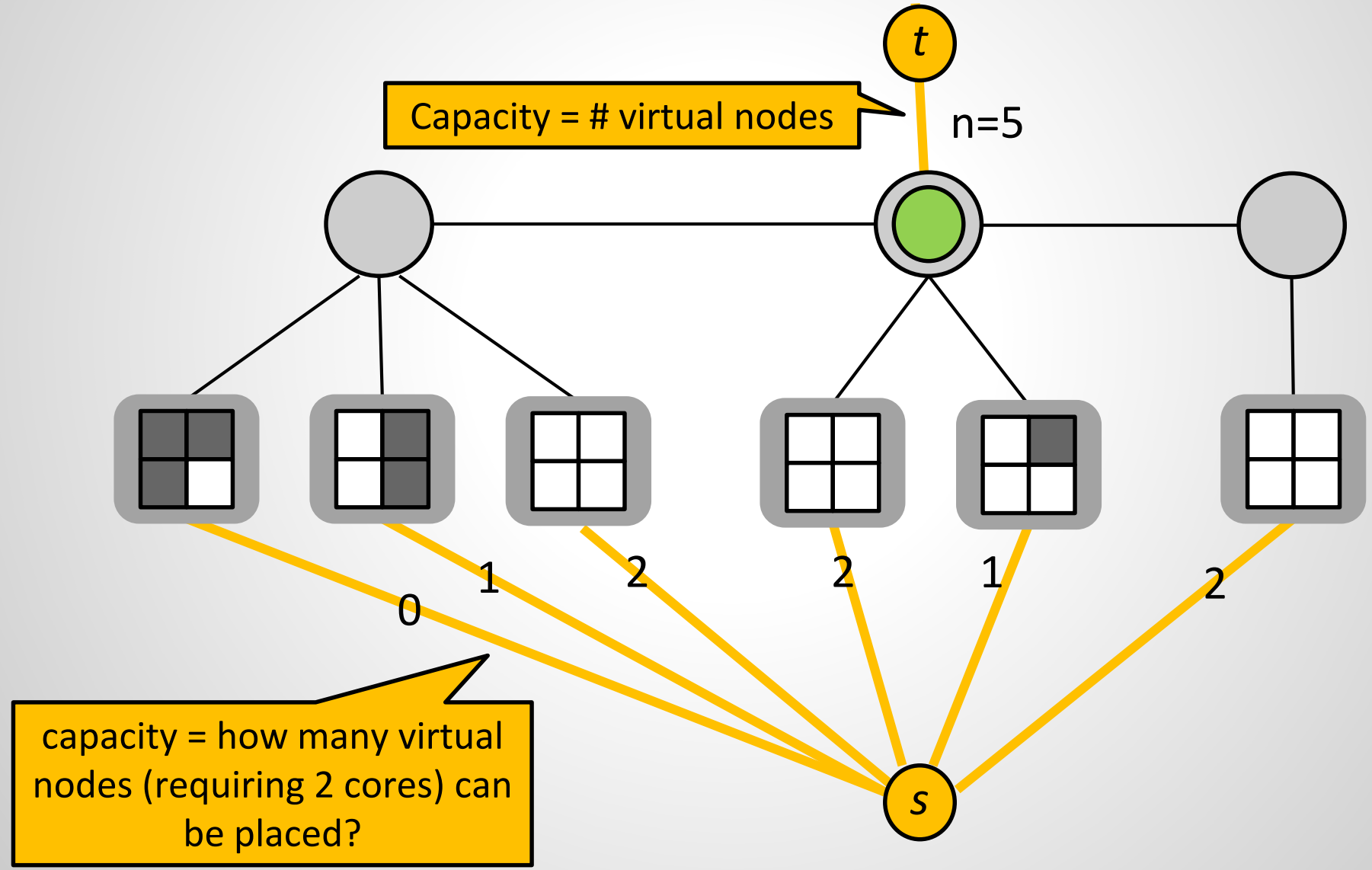
1. Place logical switch (try all options)



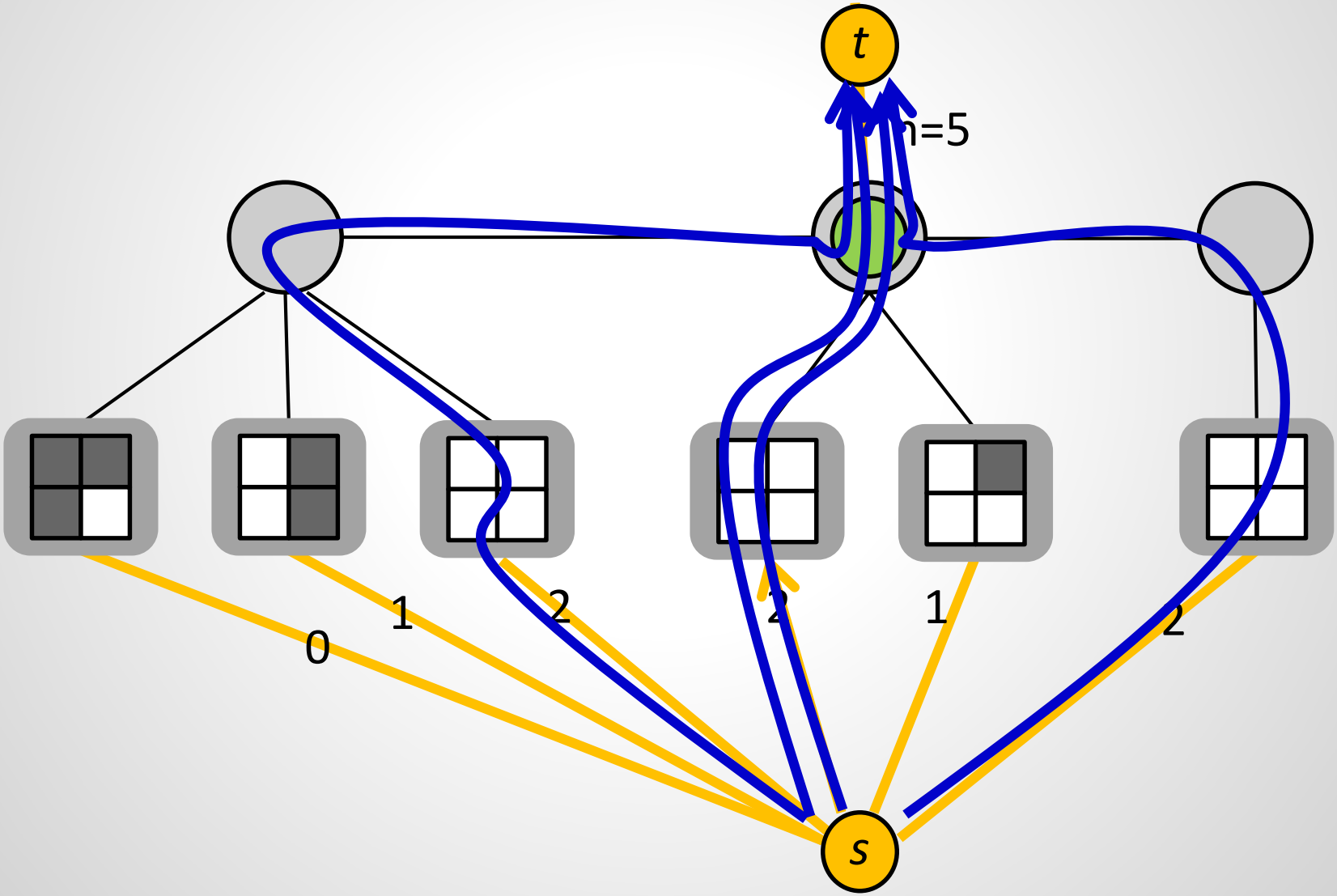
1. Place logical switch (try all options)
2. Extend network with artificial source s and sink t



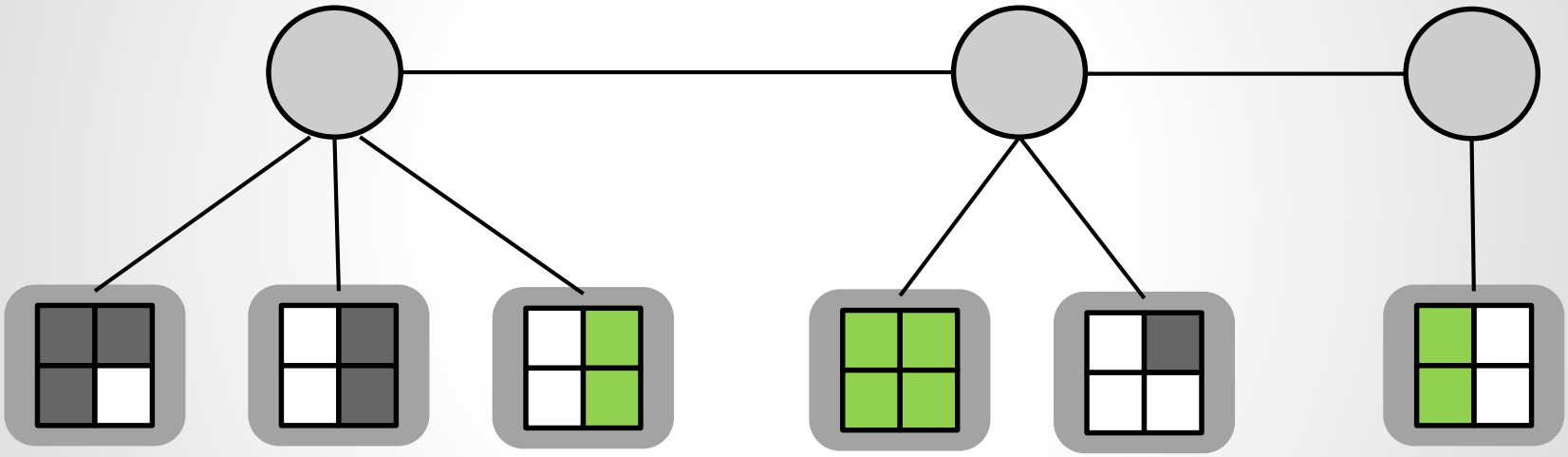
1. Place logical switch (try all options)
2. Extend network with artificial source s and sink t
3. Add capacities (recall that b=1, so each virtual node needs one unit of capacity)



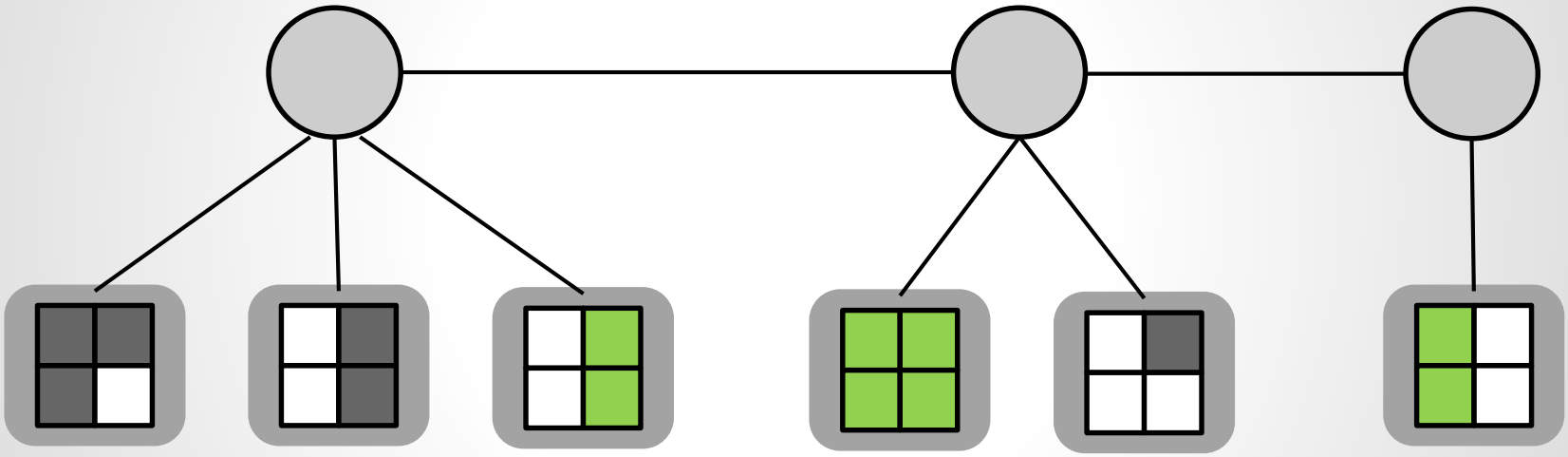
Then: Compute min-cost max flow of size n from s to t (e.g., successive shortest paths): due to capacity constraints at most size n.



Then: Compute min-cost max flow of size n from s to t (e.g., successive shortest paths): due to capacity constraints at most size n.
… and assign virtual nodes (and edges) accordingly!

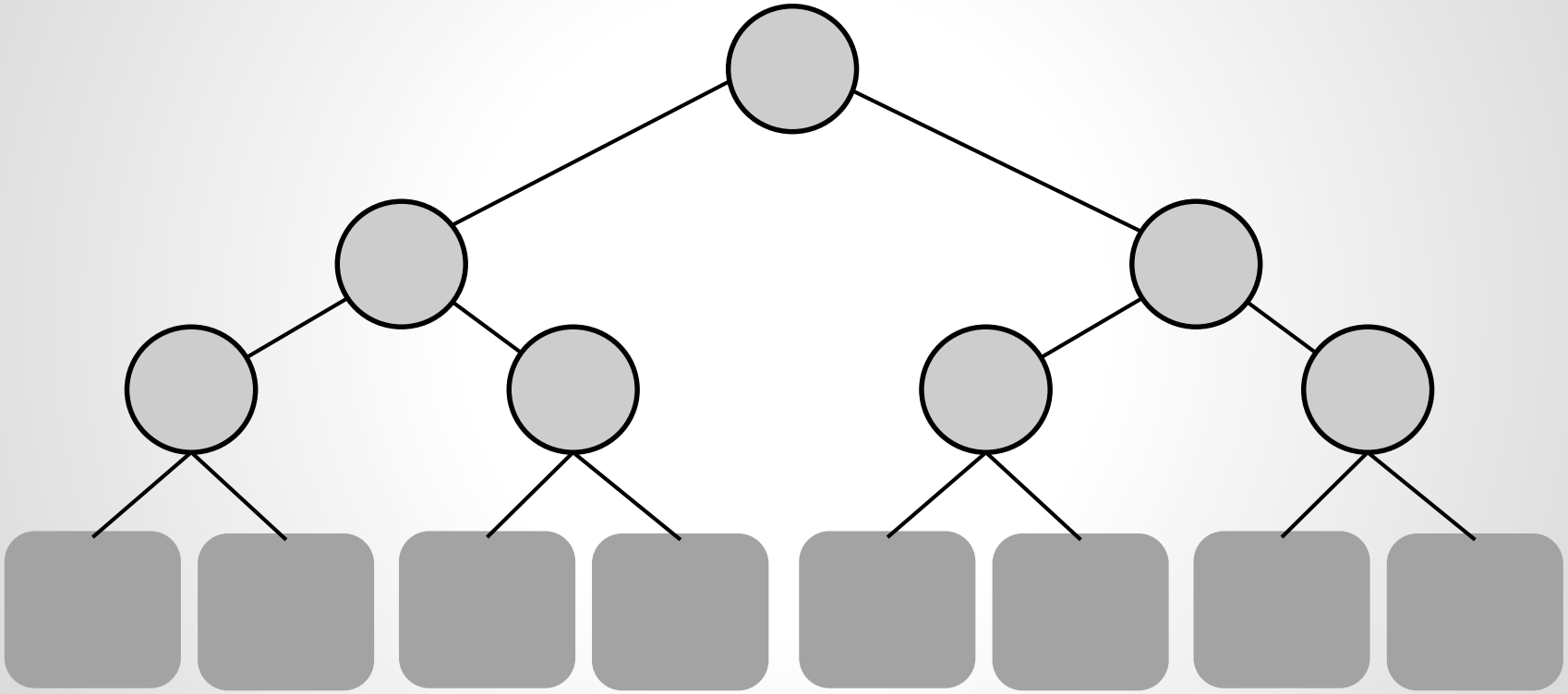


How to Embed a Virtual Cluster?



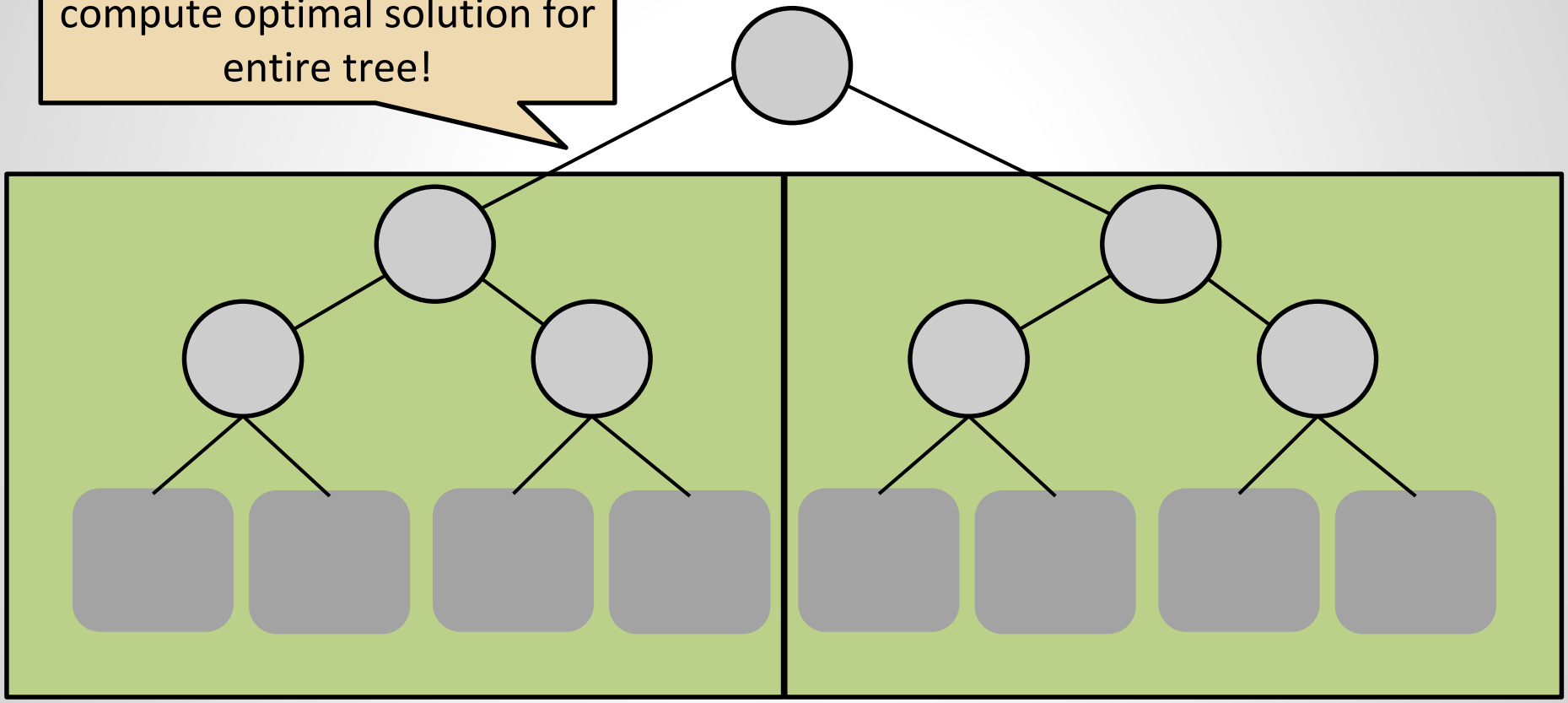
In fact: this physical network is even ***a tree***!
For trees with servers at leaves, even
simpler algorithms are possible. Ideas?

Dynamic Programming

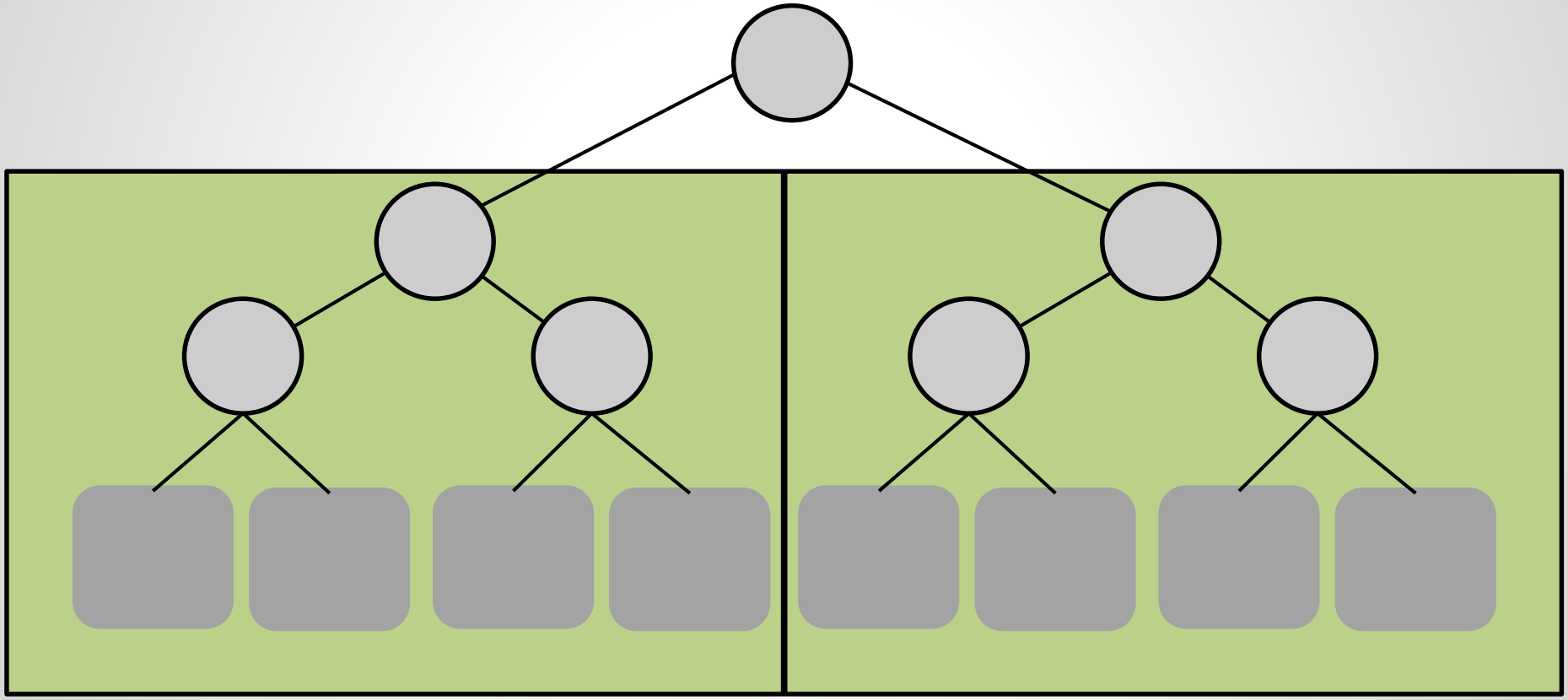


Dynamic Programming

Bottom-up programming: given optimal solution for subtrees, can quickly compute optimal solution for entire tree!



Dynamic Programming

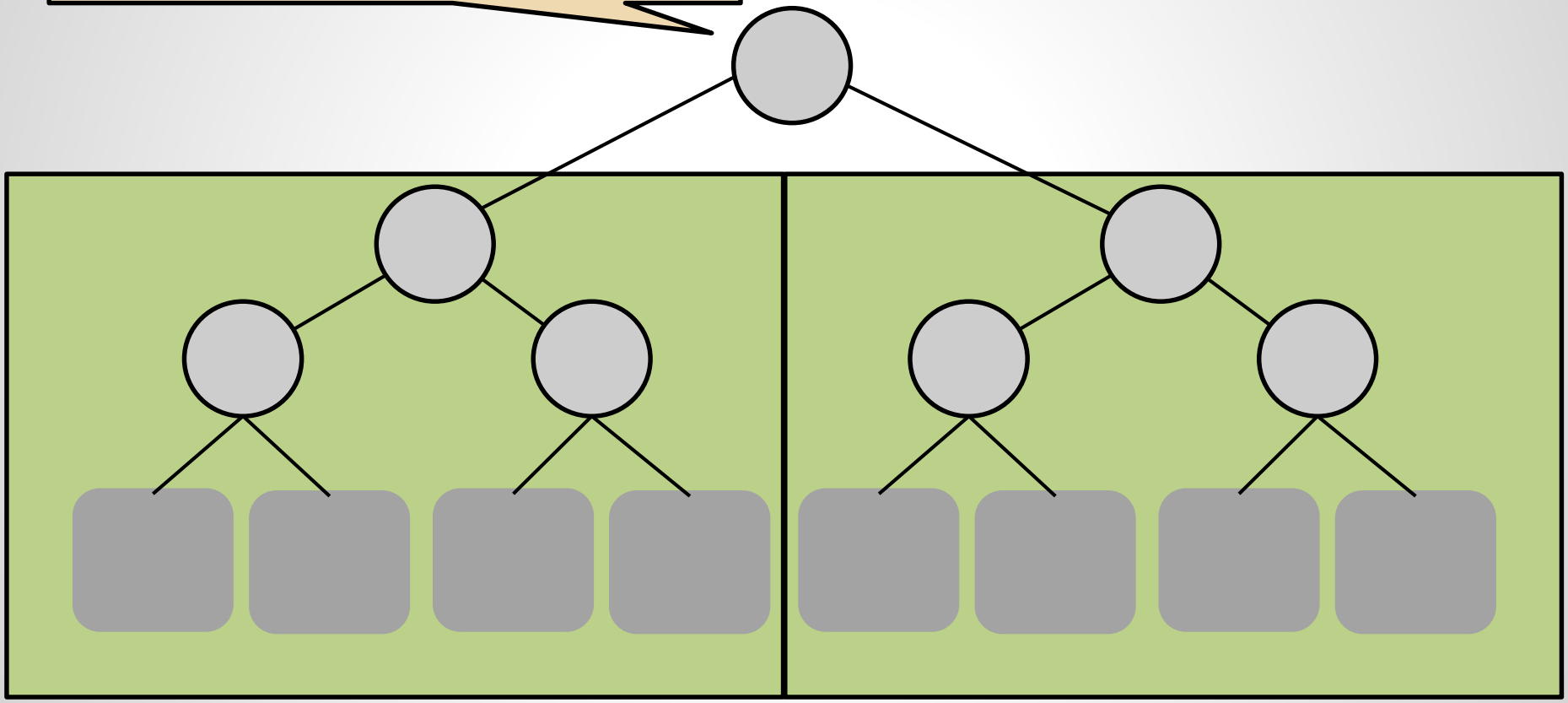


Given **optimal embedding** for x ∈ {0, ..., n} virtual nodes in left subtree…

Given **optimal embedding** for x ∈ {0, ..., n} virtual nodes in right subtree…

Dynamic Programming

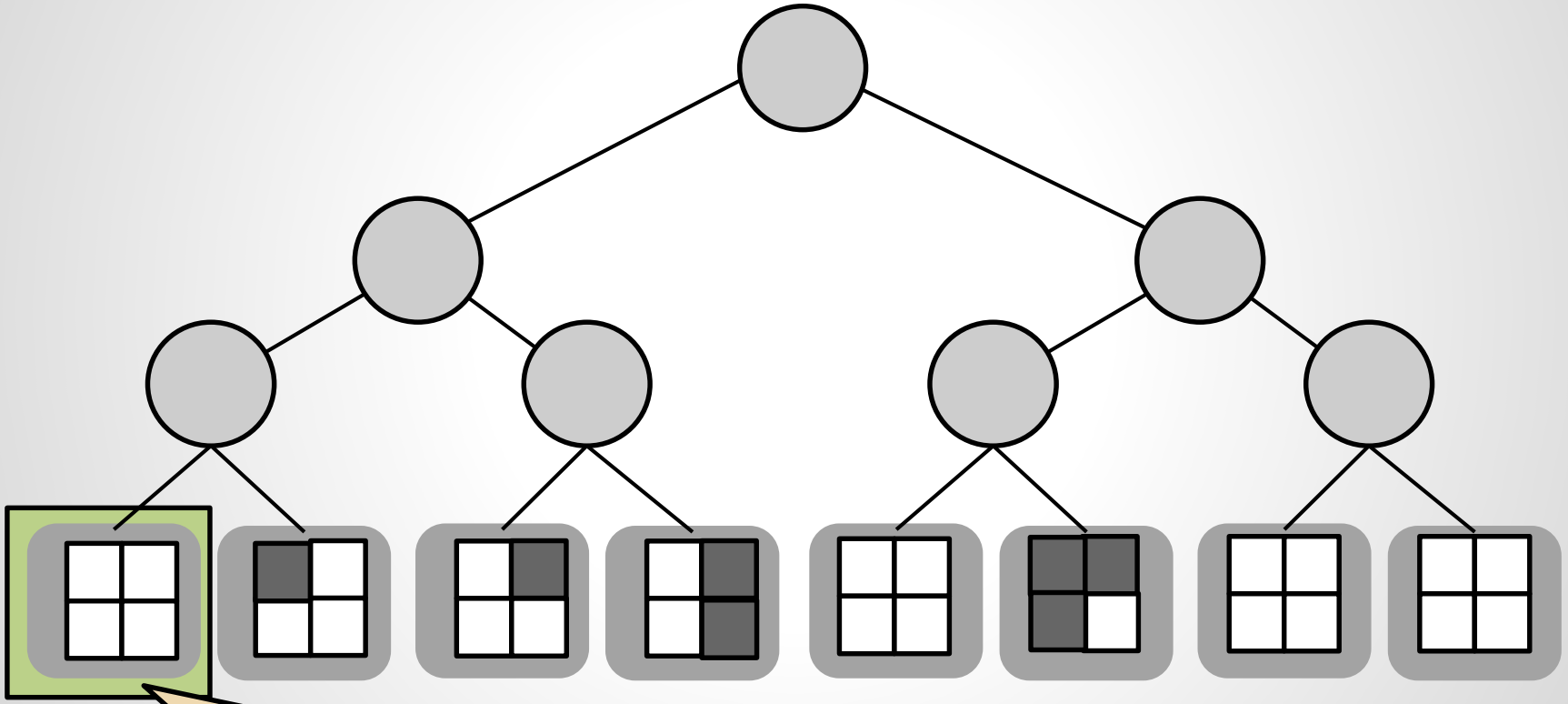
… can compute optimal embedding of $x \in \{0, ..., n\}$ virtual nodes in **entire subtree**!



Given **optimal embedding** for $x \in \{0, ..., n\}$ virtual nodes in left subtree…

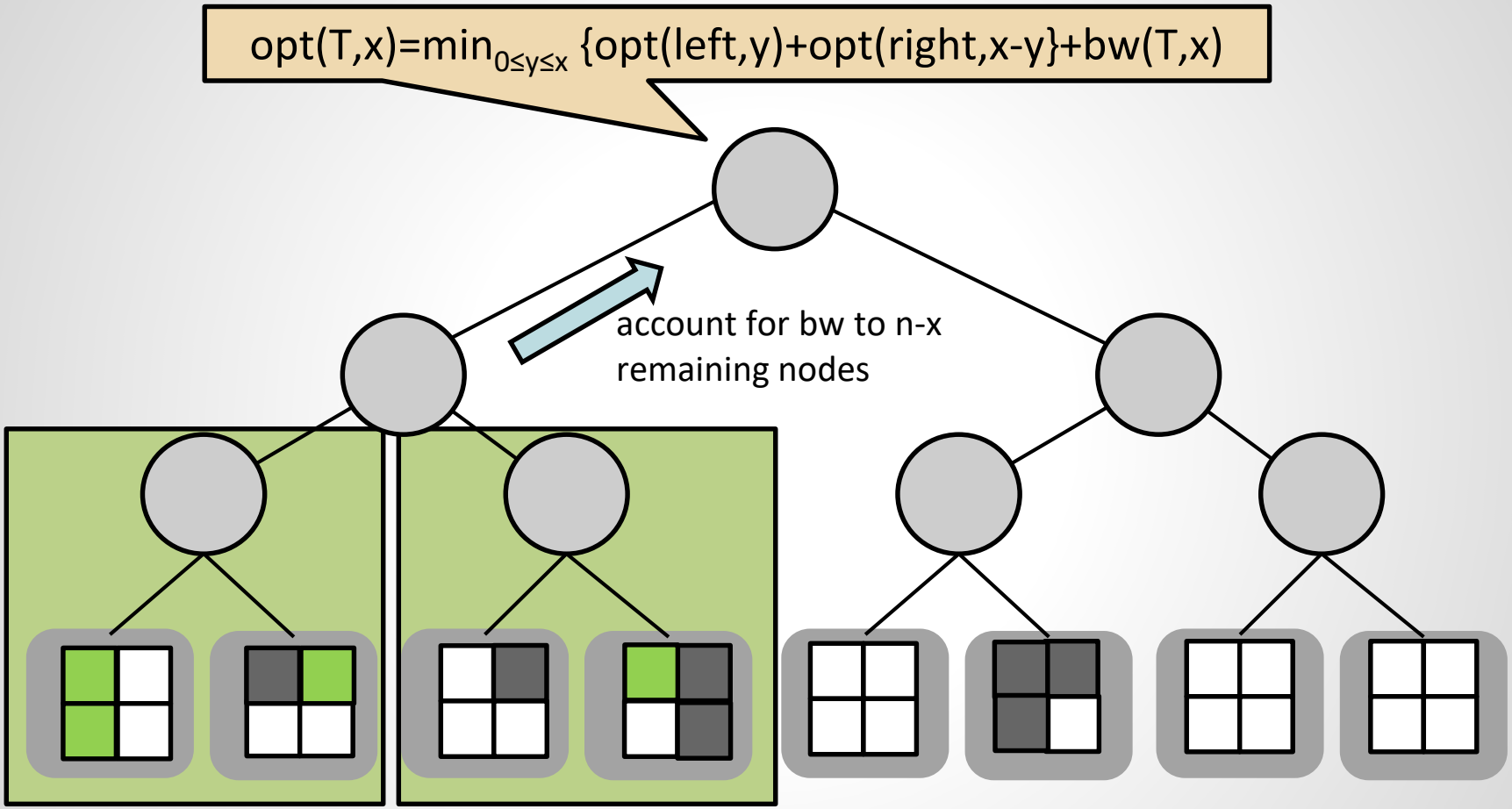
Given **optimal embedding** for $x \in \{0, ..., n\}$ virtual nodes in right subtree…

Dynamic Programming



Bottom-up «induction». Leaves easy: either x nodes fit server (cost 0) or not (cost ∞): opt[≤4] = 0 , opt[>4] = ∞

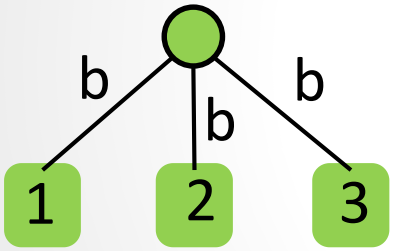
Dynamic Programming



To compute cost of embedding x nodes in T, place y nodes on the left, x-y on the right subtree, and compute cost due to links across root.

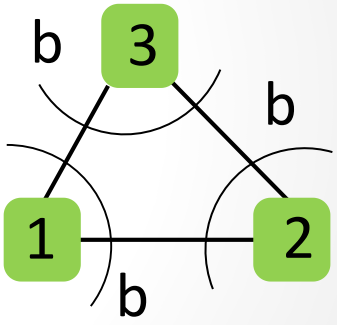
Remark on Virtual Cluster Abstraction

- Two interpretations:
  - Logical switch at unique location
  - Logical switch can be distributed («Hose model»)



Logical Switch Variant

vs

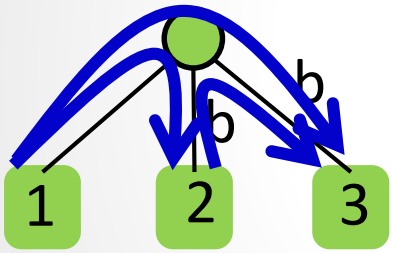


Hose Variant

Aggregated bw in/out node at most b.

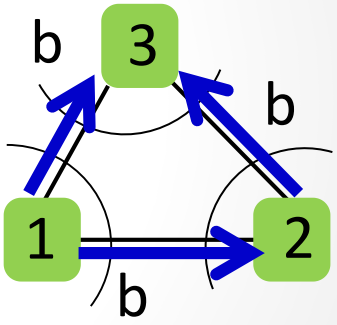
Remark on Virtual Cluster Abstraction

- Two interpretations:
  - Logical switch at unique location
  - Logical switch can be distributed («Hose model»)



Logical Switch Variant

vs



Hose Variant

Can serve the same communication patterns! (A **polytope of possible traffic matrices**.)

Example:

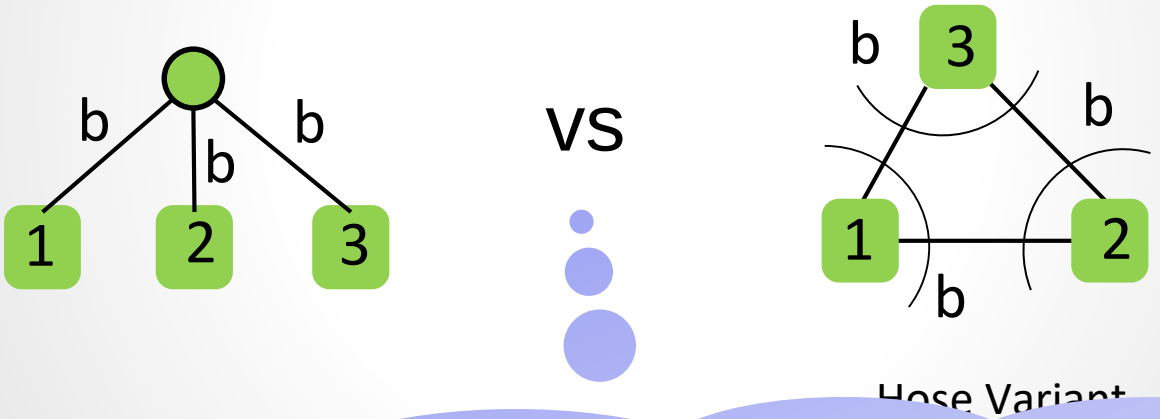
(1,2): b/2

(1,3): b/2

(2,3): b/2

Remark on Virtual Cluster Abstraction

- Two interpretations:
  - Logical switch at unique location
  - Logical switch can be distributed («Hose model»)

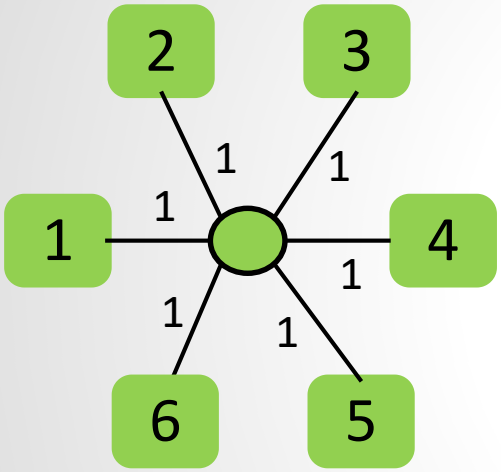


But **embedding costs** can be different if we do not insist on placing the logical switch explicitly! **Not on trees** though, and **not in uncapacitated networks**: without routing restrictions, optimal routing paths form a tree (**SymG=SymT** a.k.a. **VPN Conjecture**).

The Benefit of Hose Interpretation

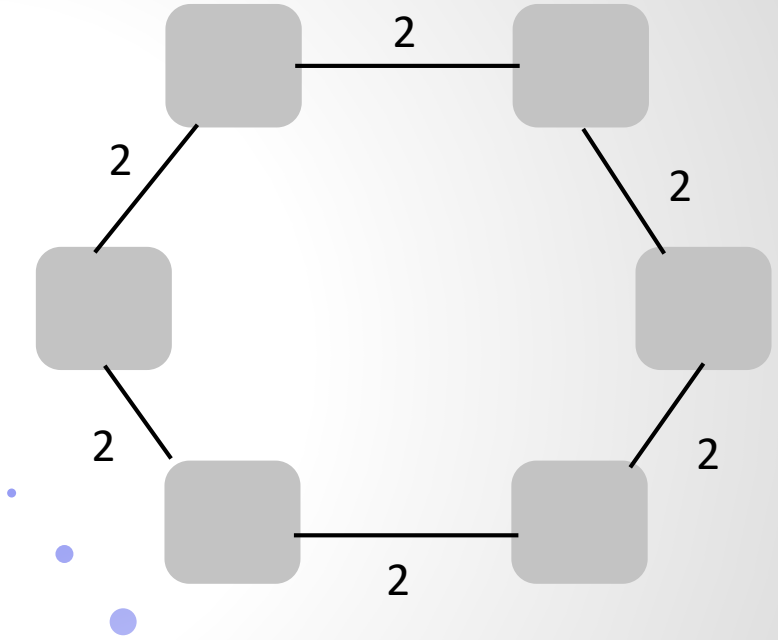
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

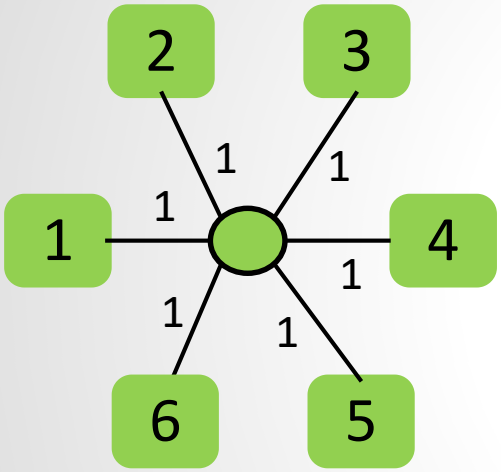
Host Graph: A Ring



How to embed as a star?

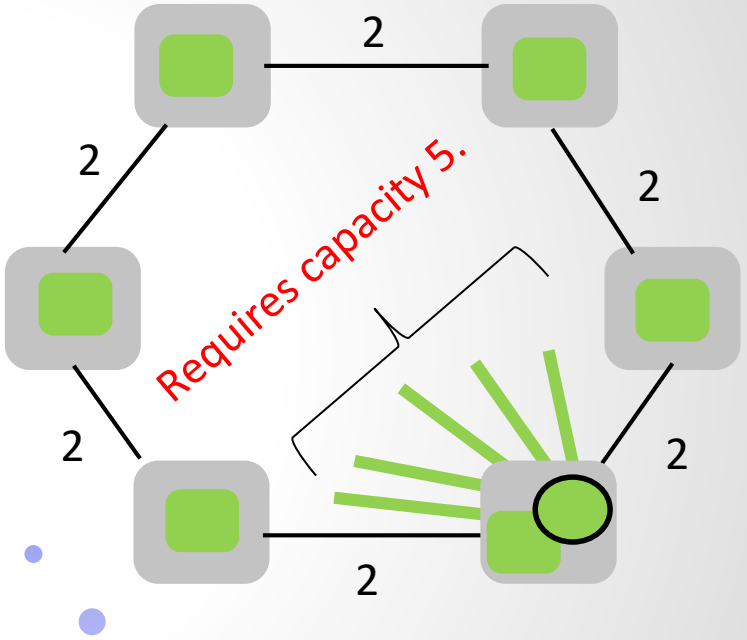
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

Host Graph: A Ring

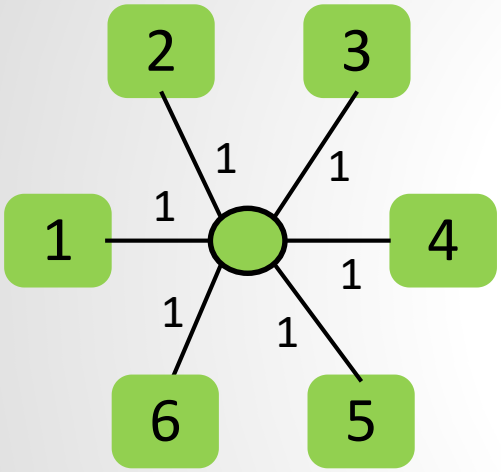


How to embed as a star?

Impossible: need at least 5 units of flow from/to node where ***star center is mapped***. However, capacity of incident links is only 4.

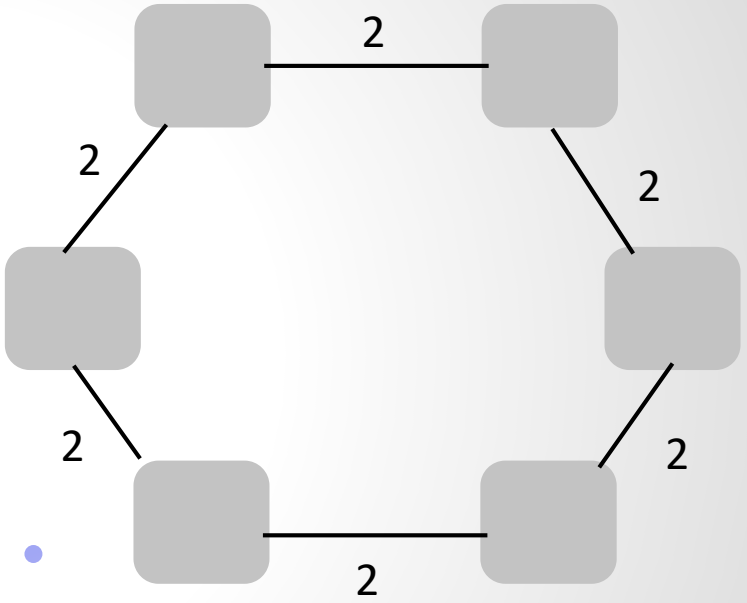
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

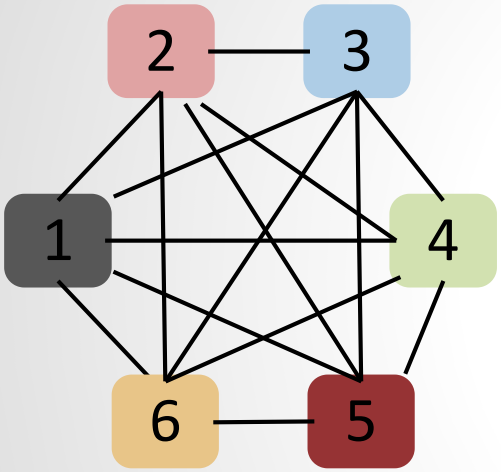
Host Graph: A Ring



How about hose?

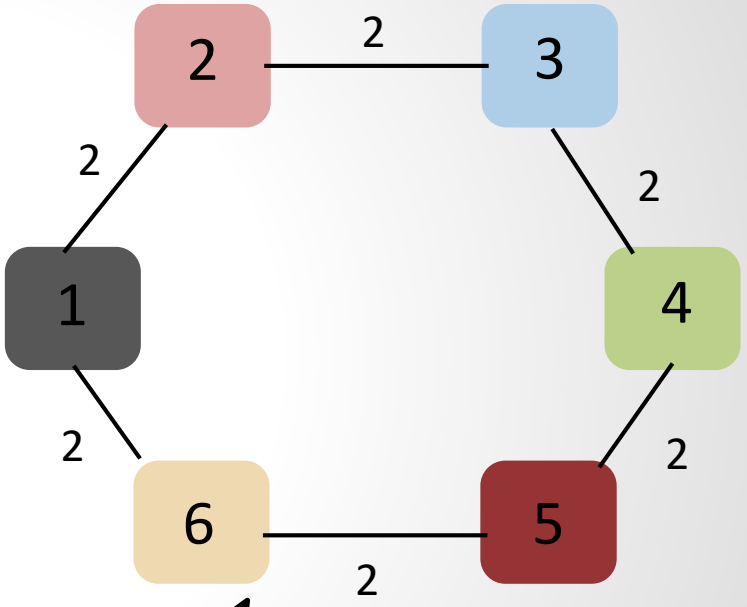
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

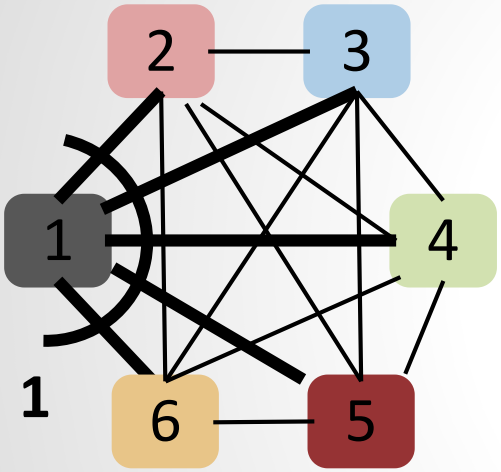
Host Graph: A Ring



Node mapping! Now: How to embed these virtual links?

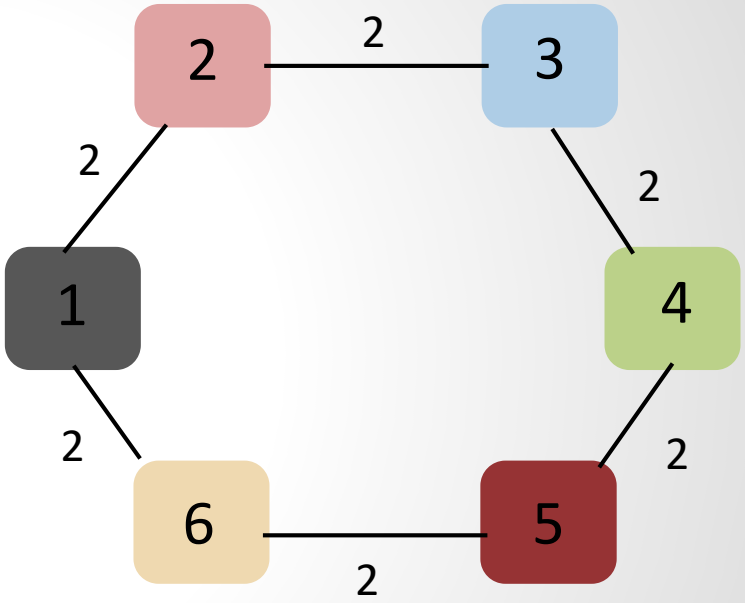
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

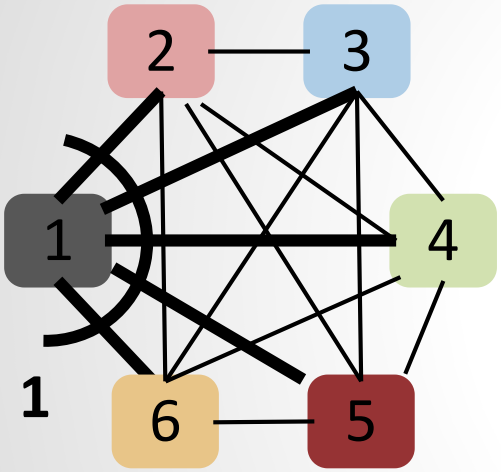
Host Graph: A Ring



Recall: hose has total demand at most 1.

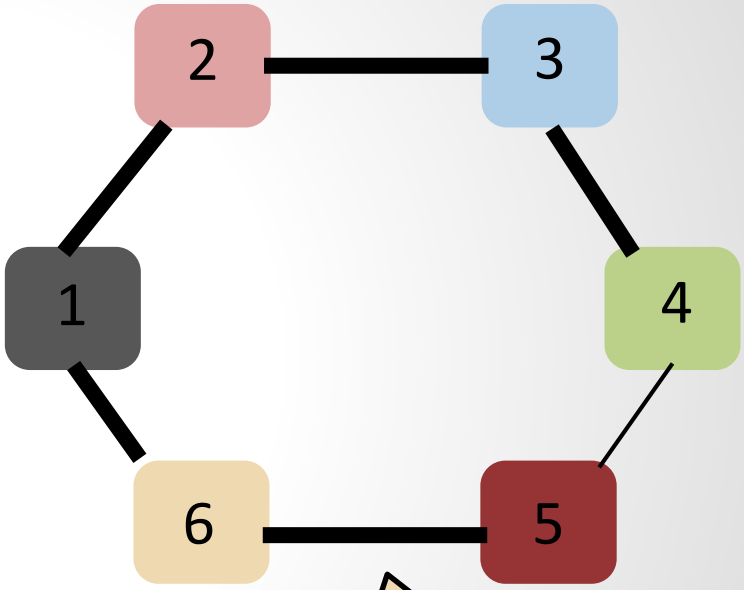
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

Host Graph: A Ring

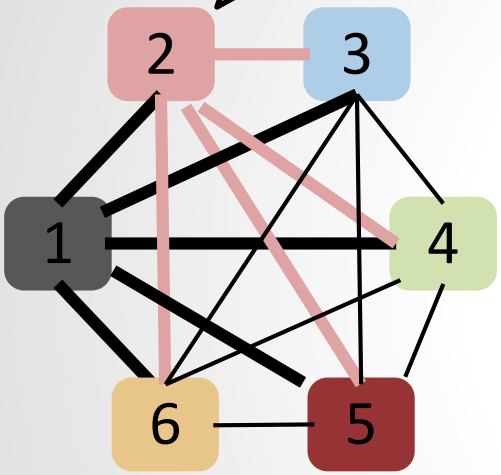


Recall: hose has total demand at most 1.

Virtual links from node 1 to {2,3,4,5,6} can be implemented along this route: fulfills capacity constraints under any traffic matrix fulfilling hose specification!

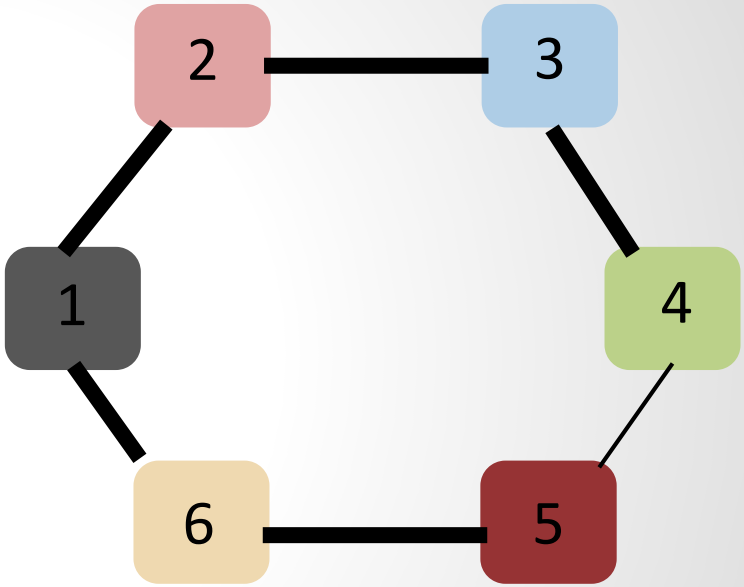
The Benefit of Hose Interpretation

Remaining virtual links to embed for virtual node 2.



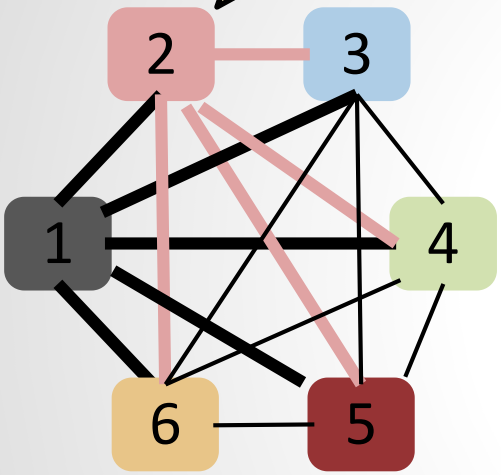
embedding?

Host Graph: A Ring



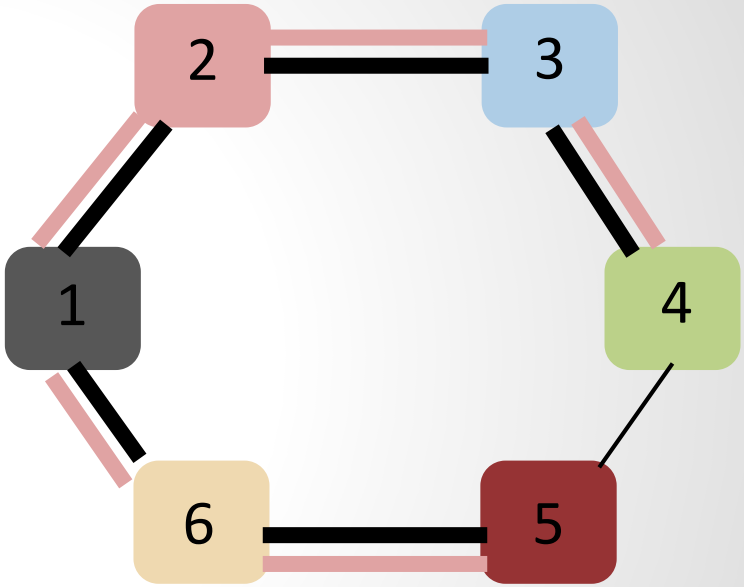
The Benefit of Hose Interpretation

Remaining virtual links to embed for virtual node 2.



embedding?

Host Graph: A Ring

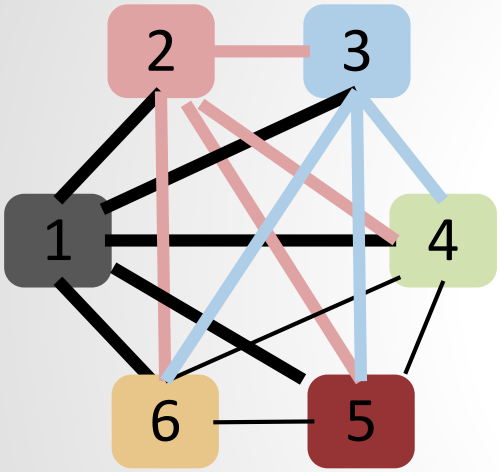


Can be implemented along this route: from node 2, reach nodes {3,4,5,6}.

The Benefit of Hose Interpretation

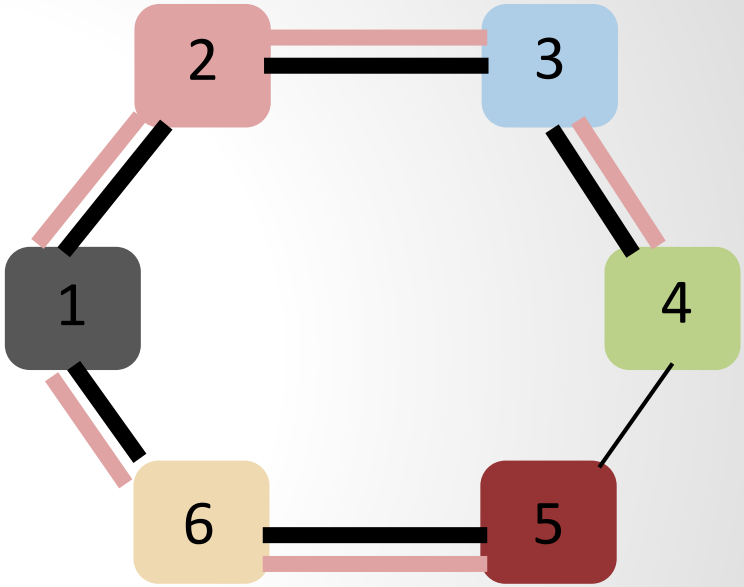
Remaining virtual links to embed for virtual node 3.

VNet: VC(n=6,b=1)



embedding?

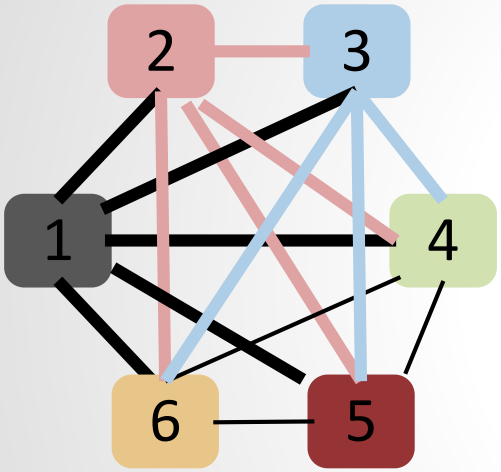
Host Graph: A Ring



The Benefit of Hose Interpretation

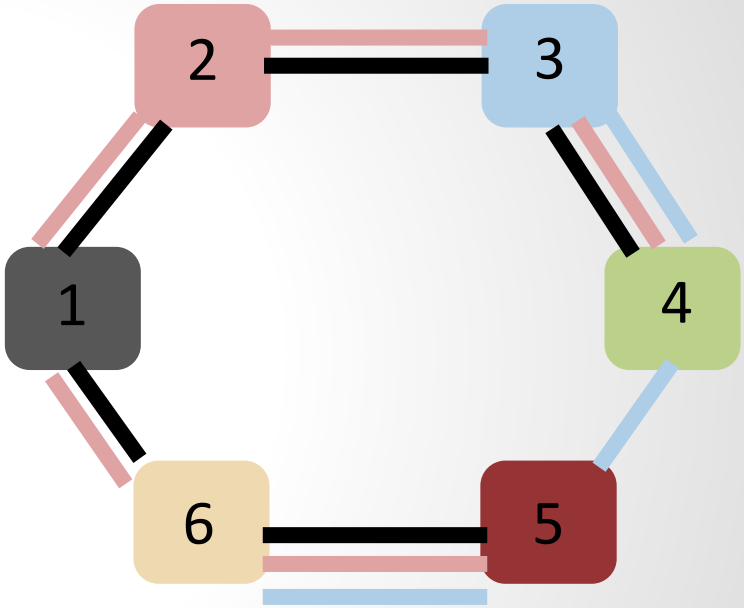
Remaining virtual links to embed for virtual node 3.

VNet: VC(n=6,b=1)



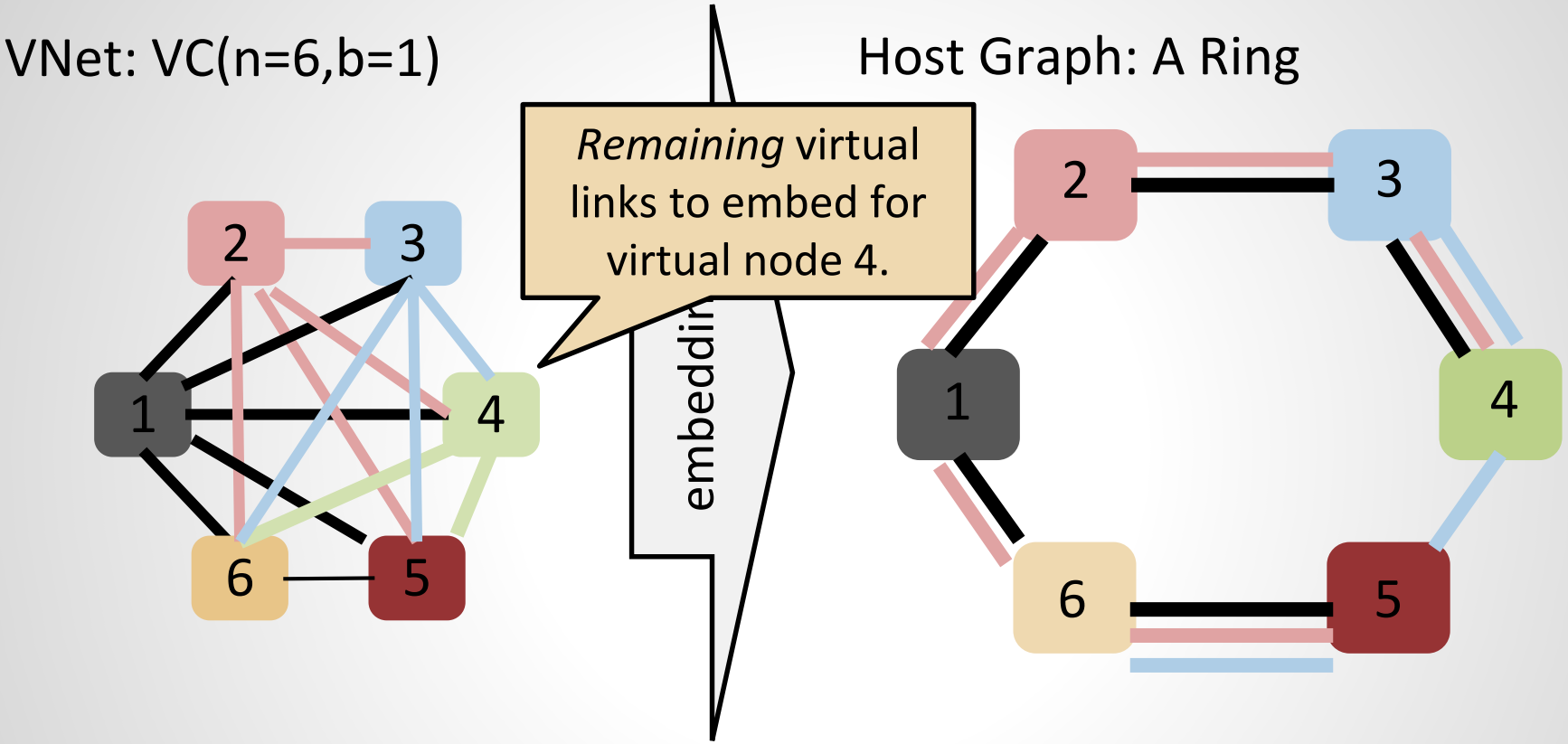
embedding?

Host Graph: A Ring

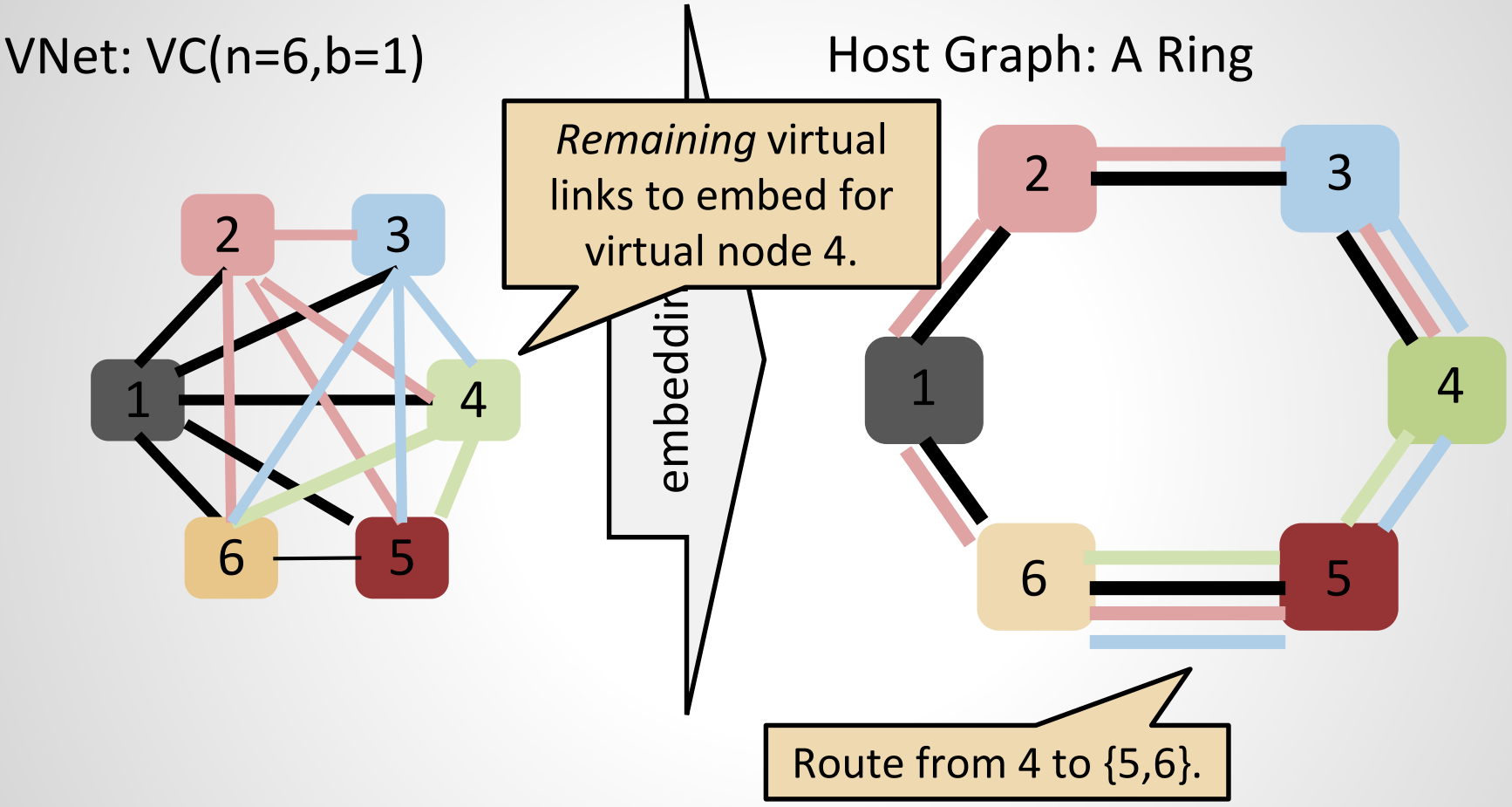


From 3, path reaches {4,5,6}.

The Benefit of Hose Interpretation

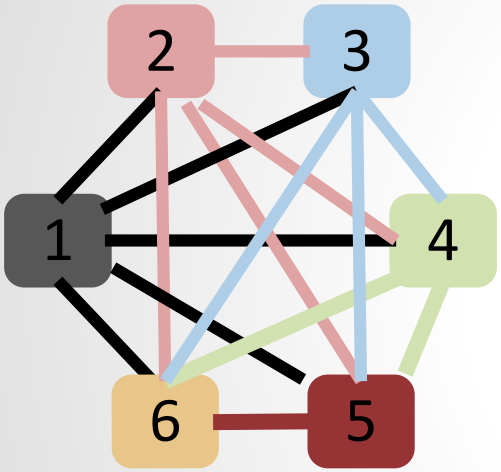


The Benefit of Hose Interpretation



The Benefit of Hose Interpretation

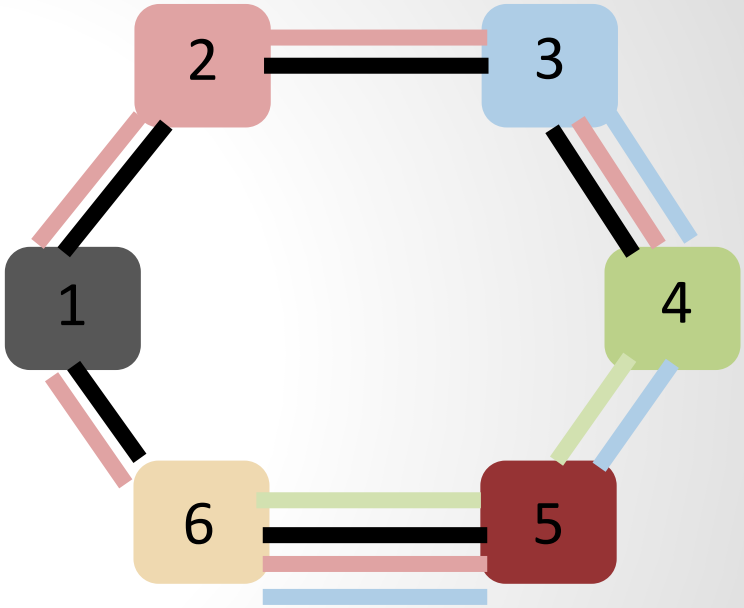
VNet: VC(n=6,b=1)



Remaining virtual link.

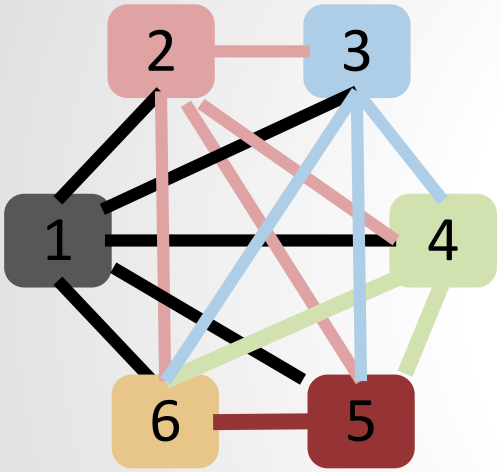
embedding?

Host Graph: A Ring



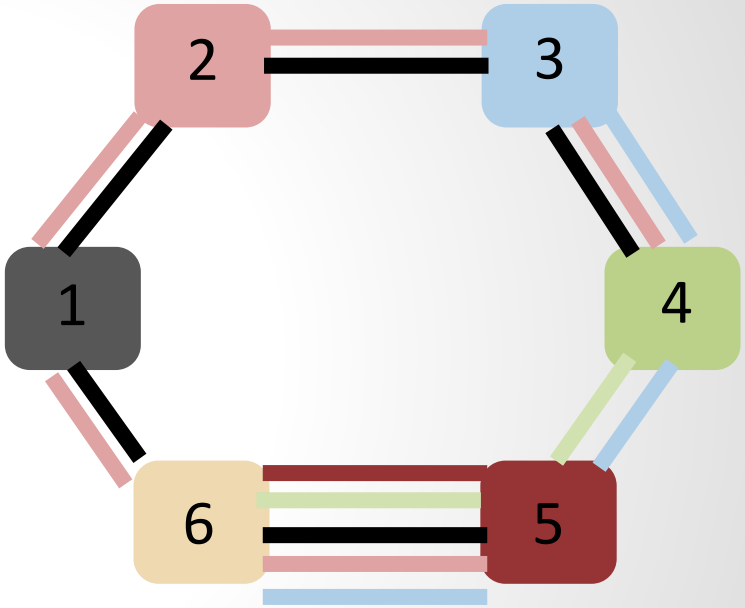
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

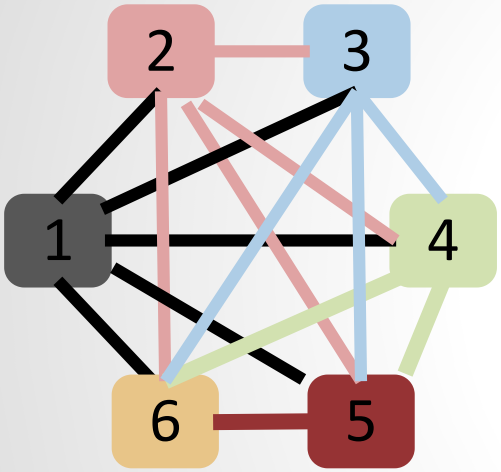
Host Graph: A Ring



All virtual links mapped to routes!

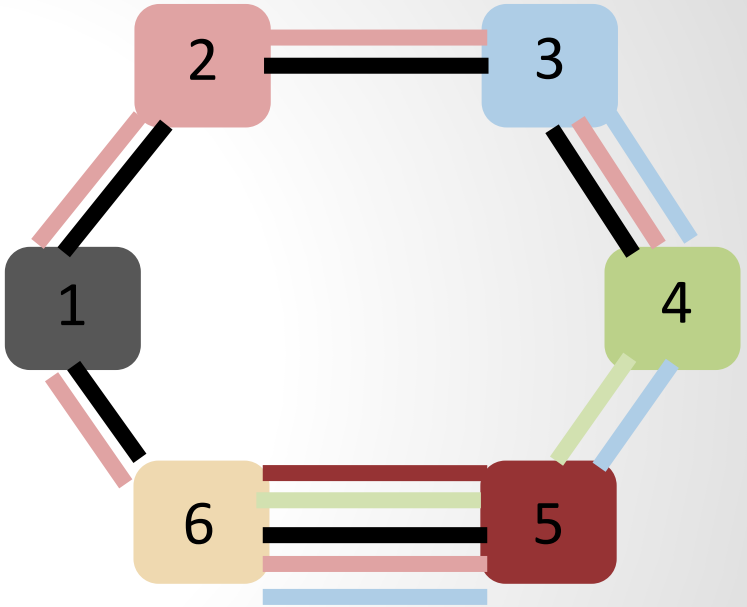
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

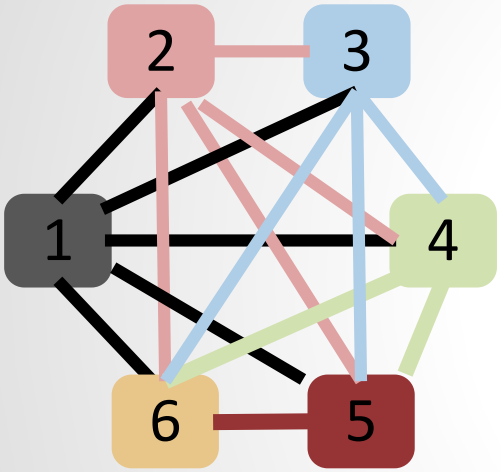
Host Graph: A Ring



But wait: 5 paths on link {5,6}!
Can demand really be satisfied given link capacity of 2?!

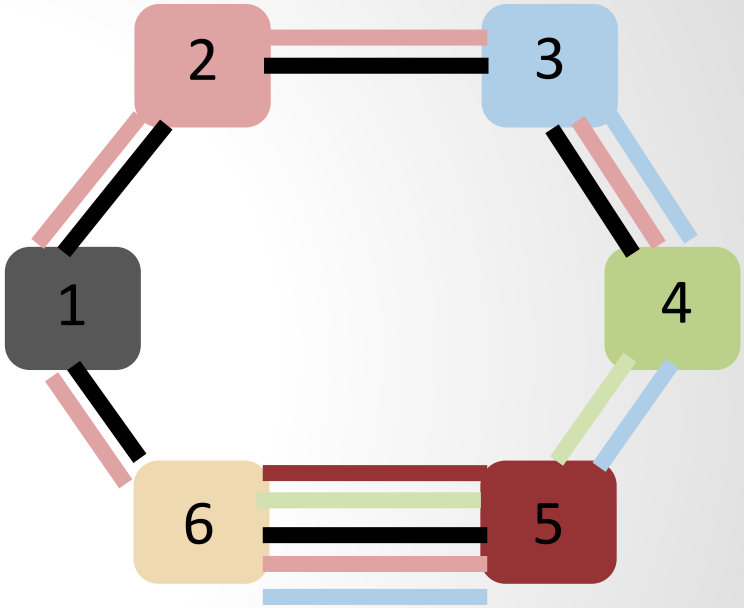
The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



embedding?

Host Graph: A Ring



Link {5,6} is used by routes: (1,5),(2,5),(3,6),(4,6),(5,6).

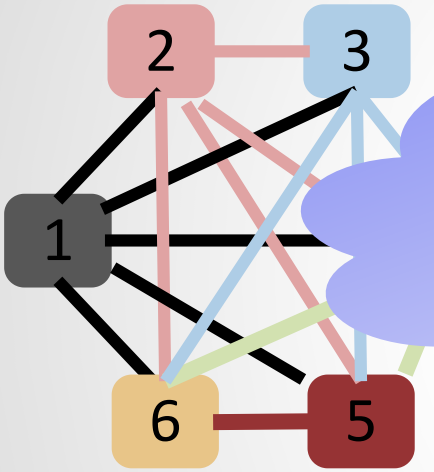
But by definition of the hose model, any traffic matrix M will respect:

$M_{1,5}+M_{2,5} \leq 1$ *and* $M_{3,6}+M_{4,6}+M_{5,6} \leq 1$.

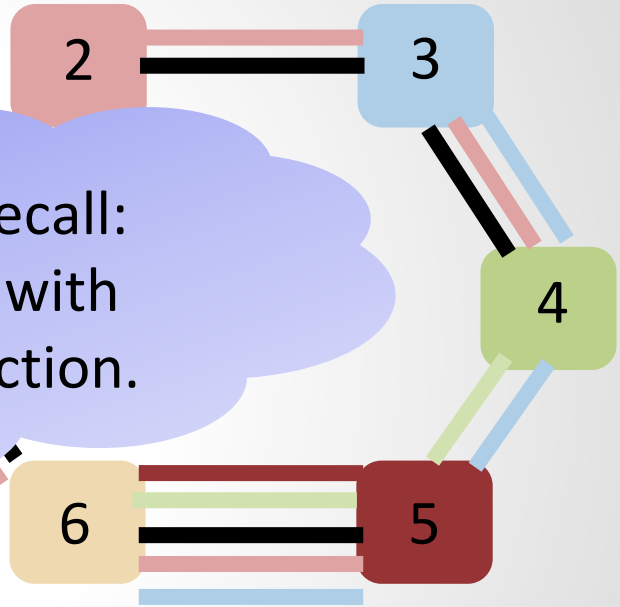
Hence $\Sigma M_{i,j} \leq 2$ *holds!*

The Benefit of Hose Interpretation

VNet: VC(n=6,b=1)



Host Graph: A Ring



Legal embedding! Recall: this was impossible with virtual switch abstraction.

Link {5,6} is used by routes: (1,5),(2,5),(3,6),(4,6),(5,6).

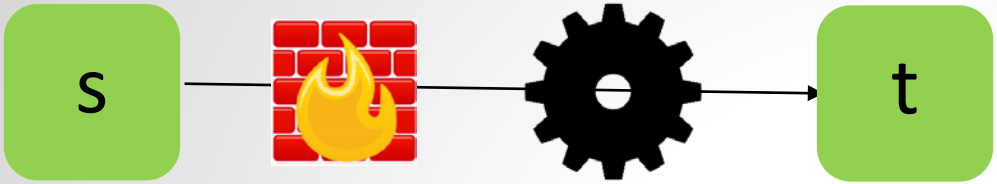
But by definition of the hose model, any traffic matrix M will respect:

$M_{1,5}+M_{2,5} \leq 1$ *and* $M_{3,6}+M_{4,6}+M_{5,6} \leq 1$.

Hence $\Sigma M_{i,j} \leq 2$ *holds!*

What about requests “with flexibilities”?

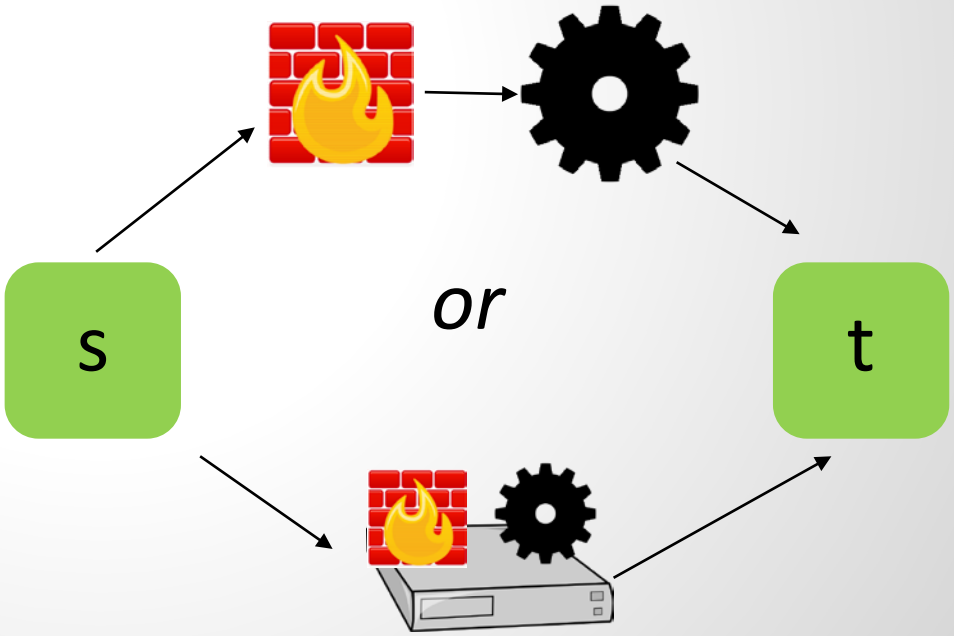
Requests can be more complex



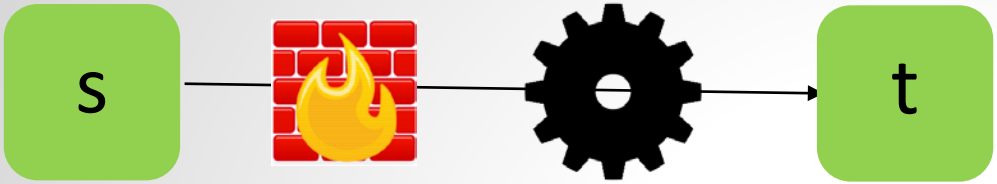
We have seen:

Already non-trivial!

But what if requests allow for alternatives and different decompositions?



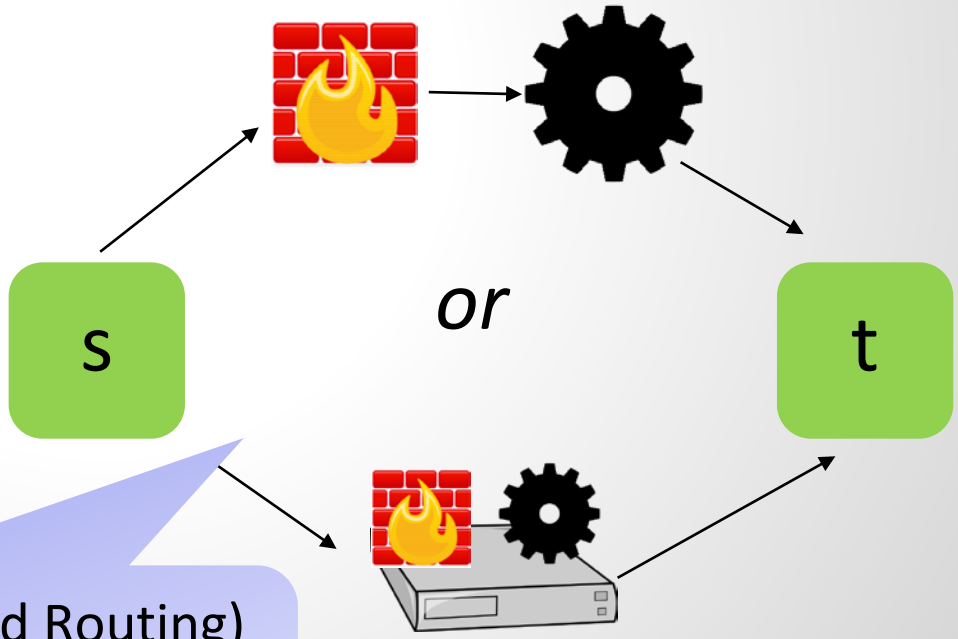
Requests can be more complex



We have seen:

Already non-trivial!

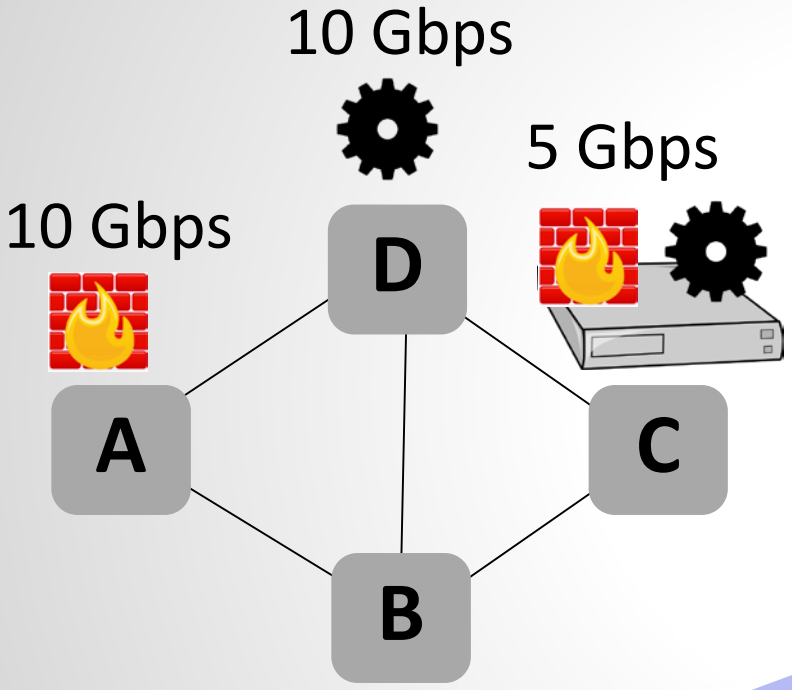
But what if requests allow for alternatives and different decompositions?



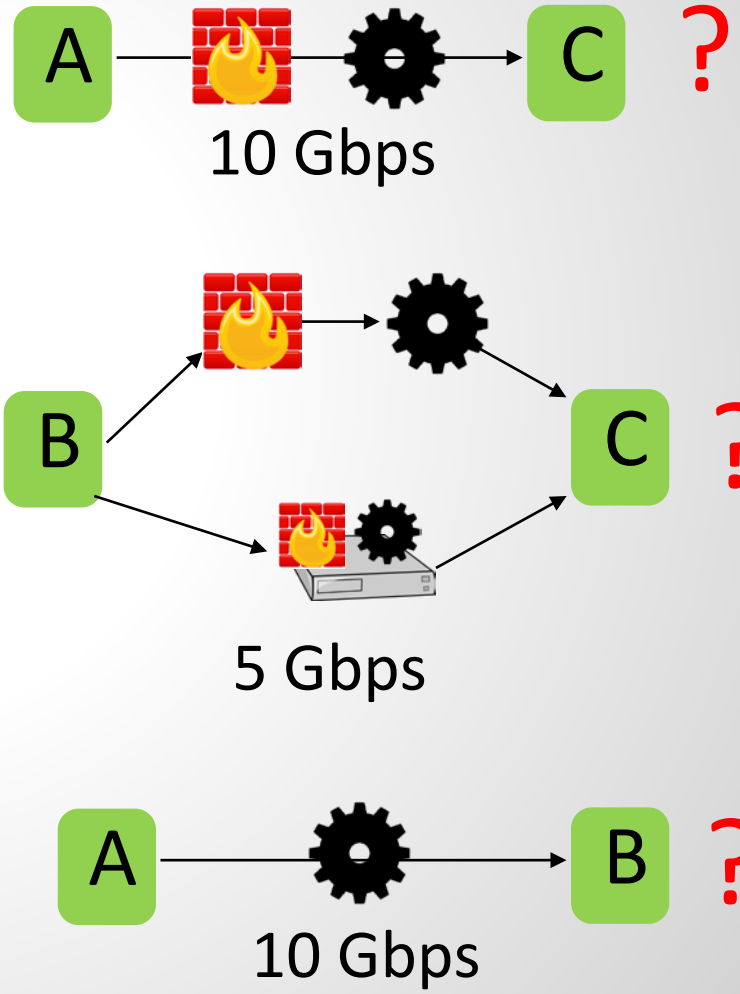
Known as PR (Processing and Routing) Graph: allows to model different choices and implementations!

Example: admission control and embedding

Substrate:



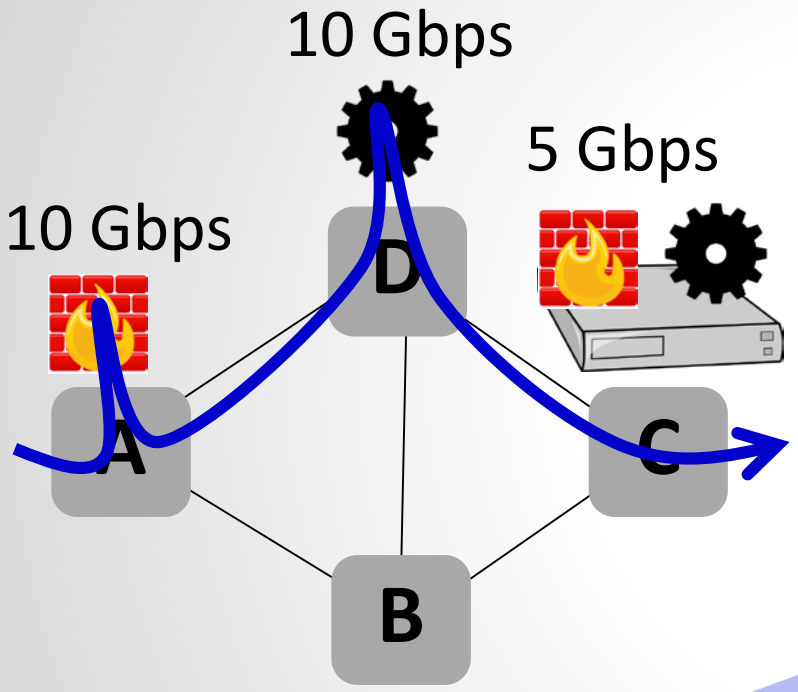
Requests:



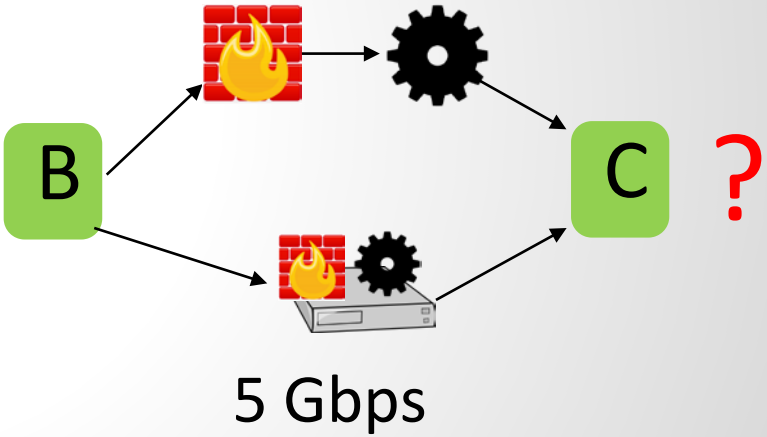
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



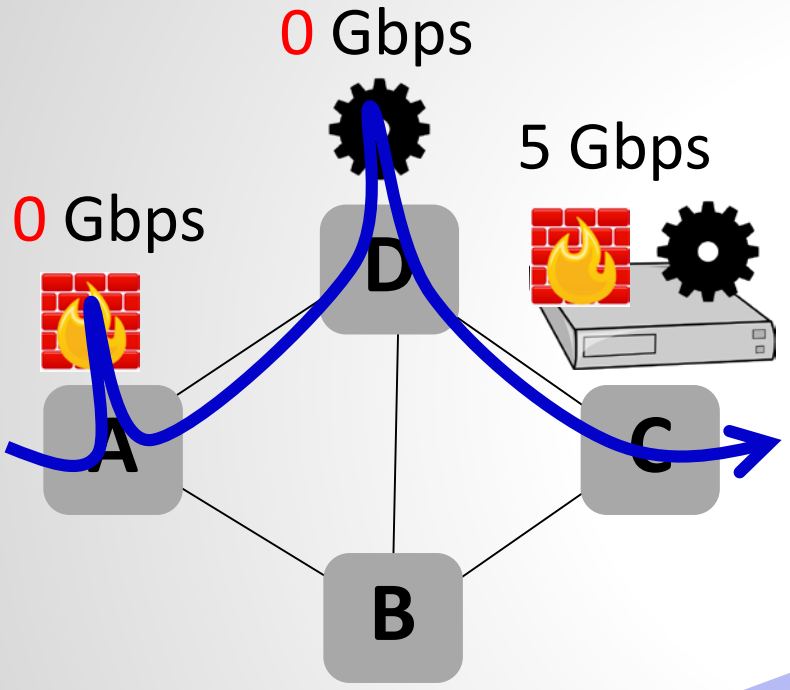
Requests:



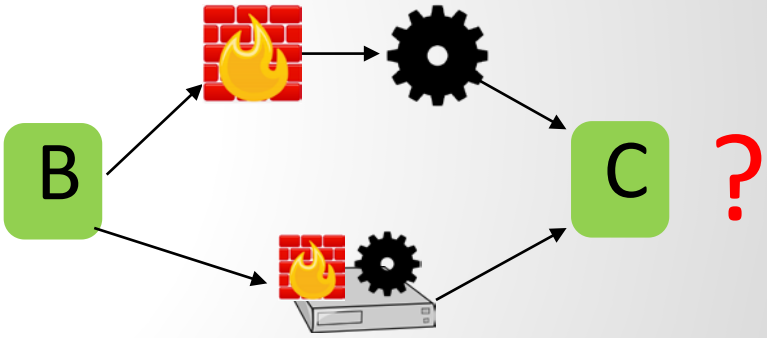
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



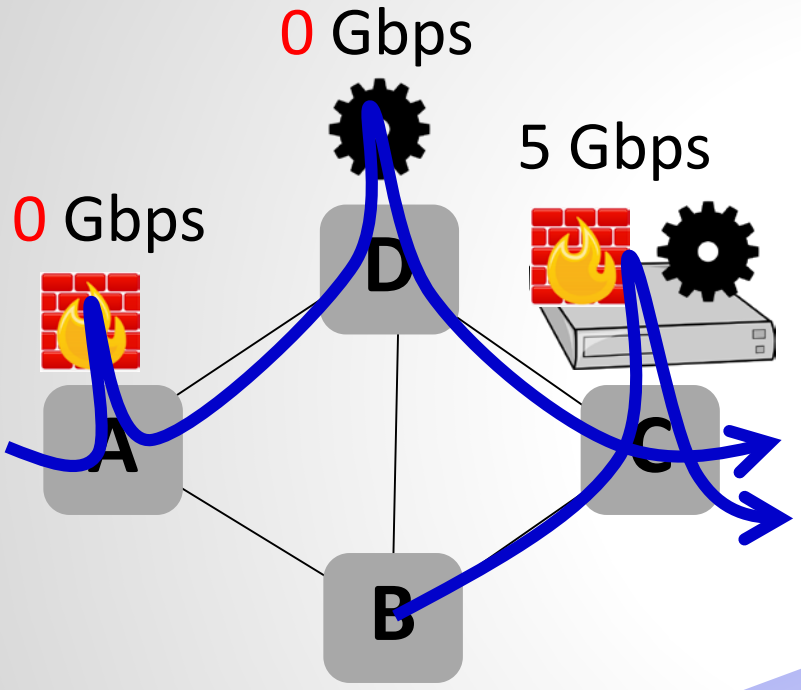
Requests:



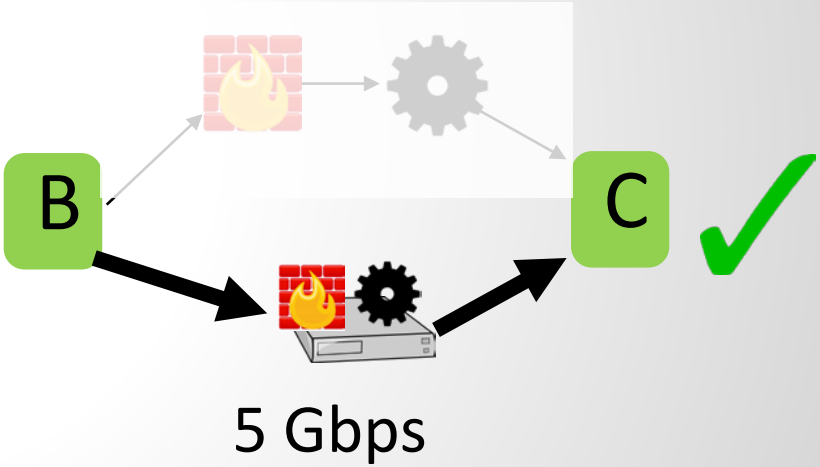
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



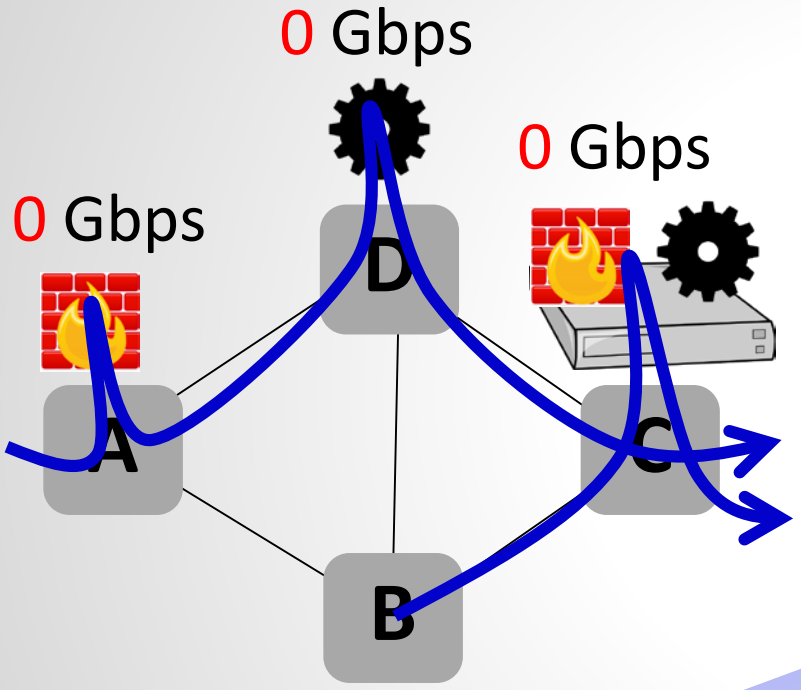
Requests:



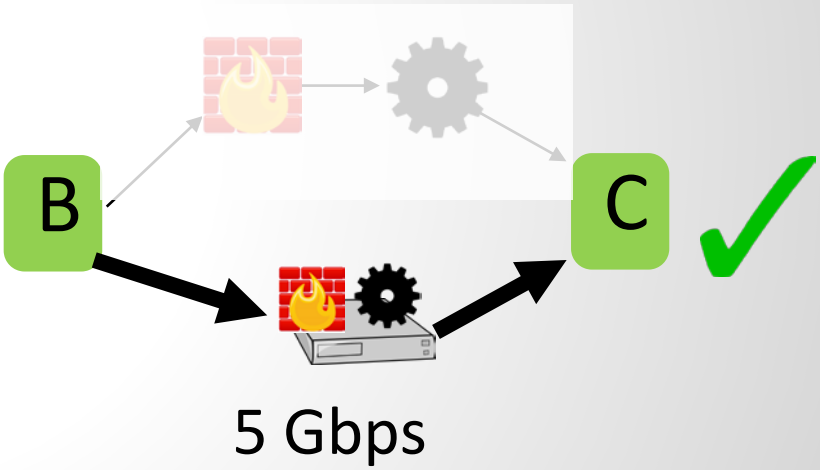
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



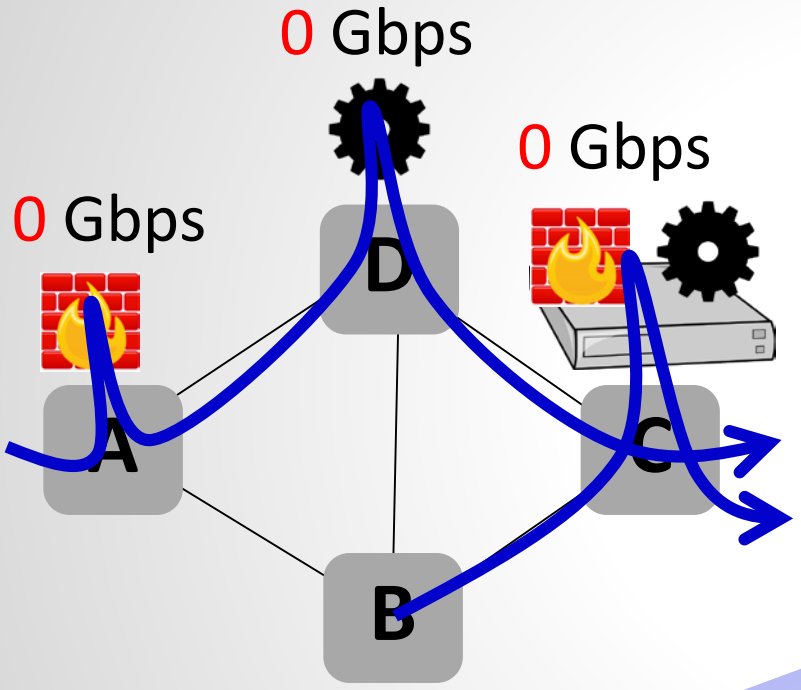
Requests:



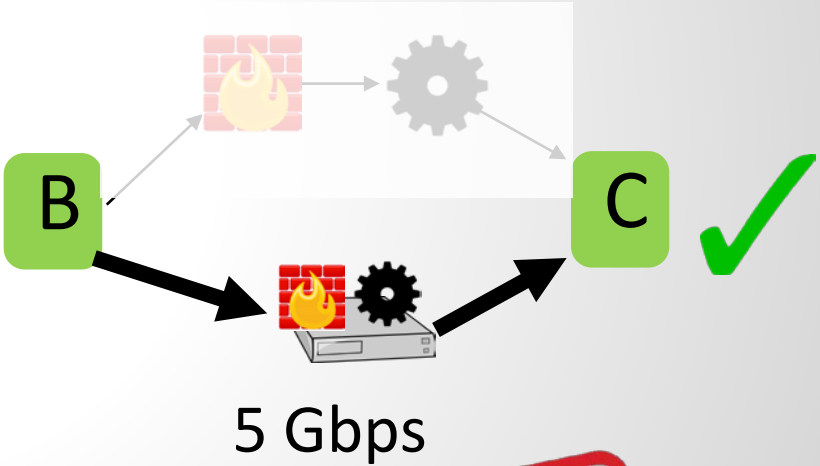
Which ones can be admitted and embedded?

Example: admission control and embedding

Substrate:



Requests:

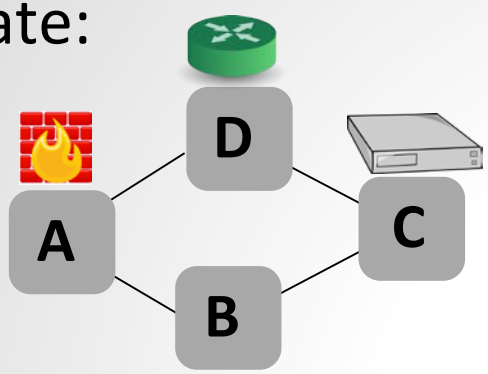


Which ones can be admitted and embedded?

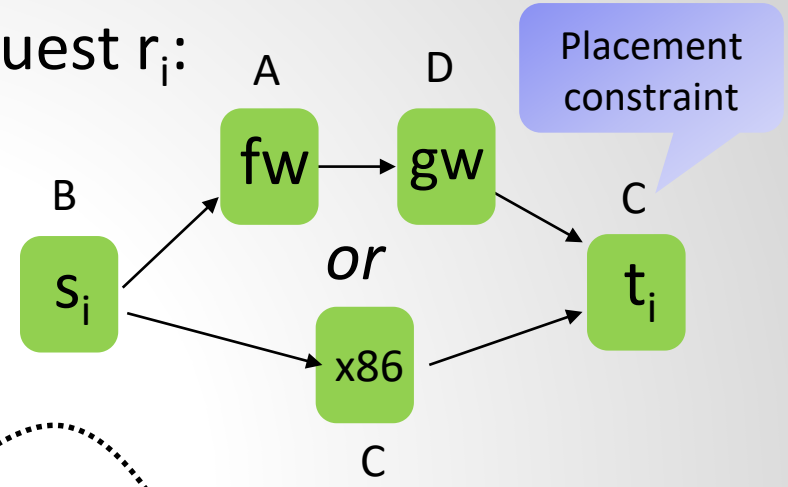
Approximations still possible for trees: Product graphs and randomized rounding

Approximations still possible for trees: Product graphs and randomized rounding

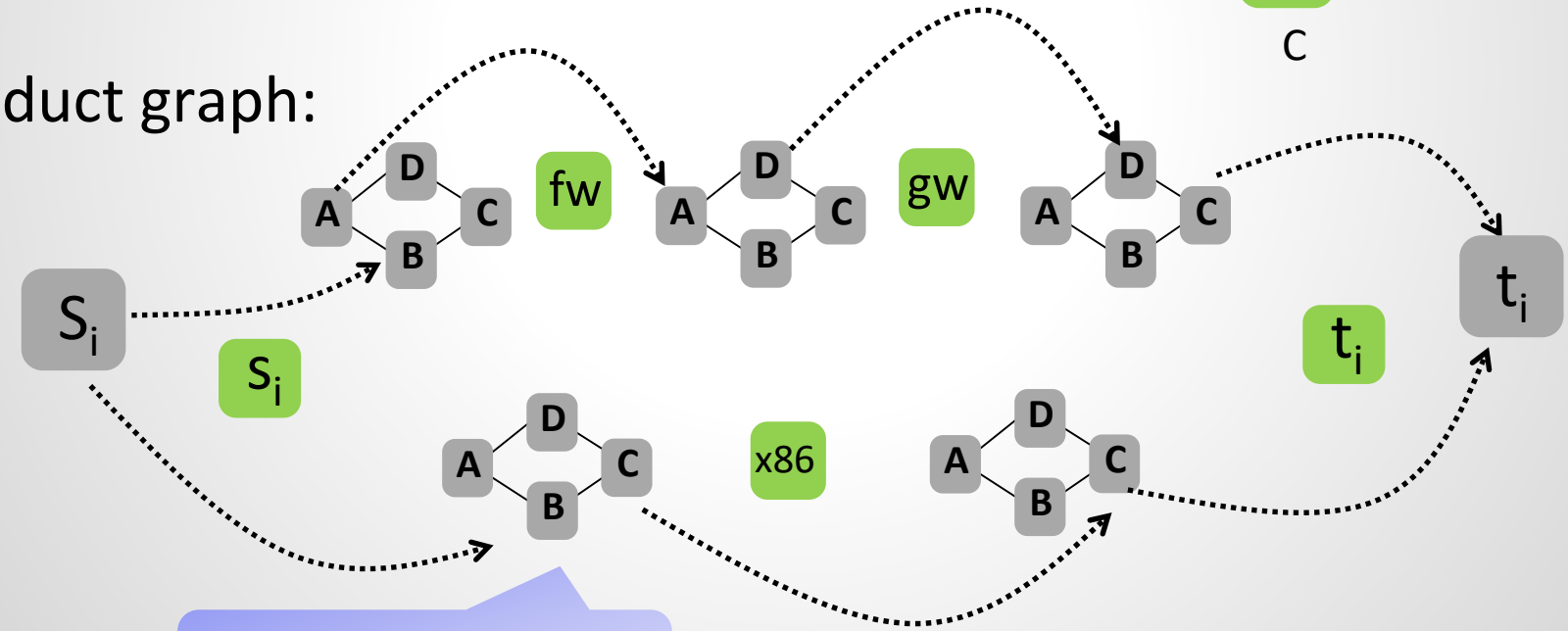
Substrate:



i^{th} request r_i:



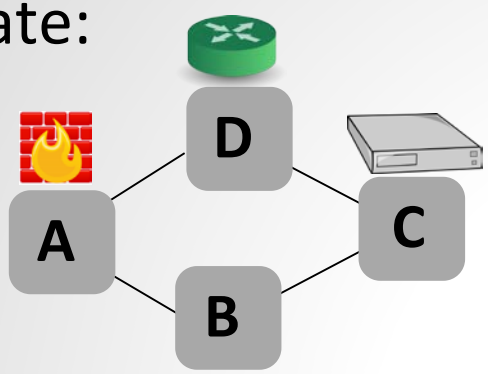
Product graph:



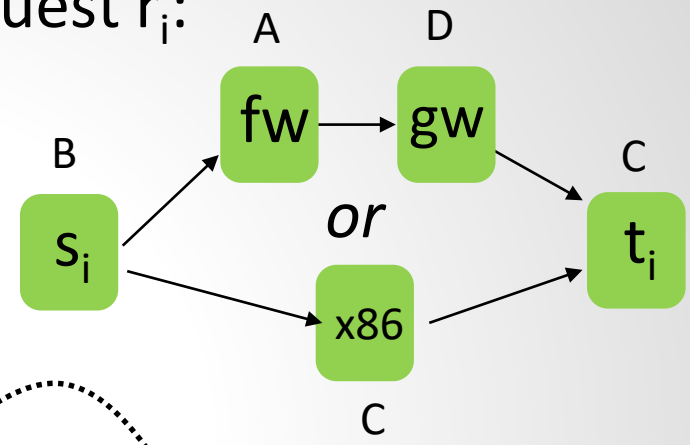
Copy substrate graph for each edge of chain

Approximations still possible for trees: Product graphs and randomized rounding

Substrate:

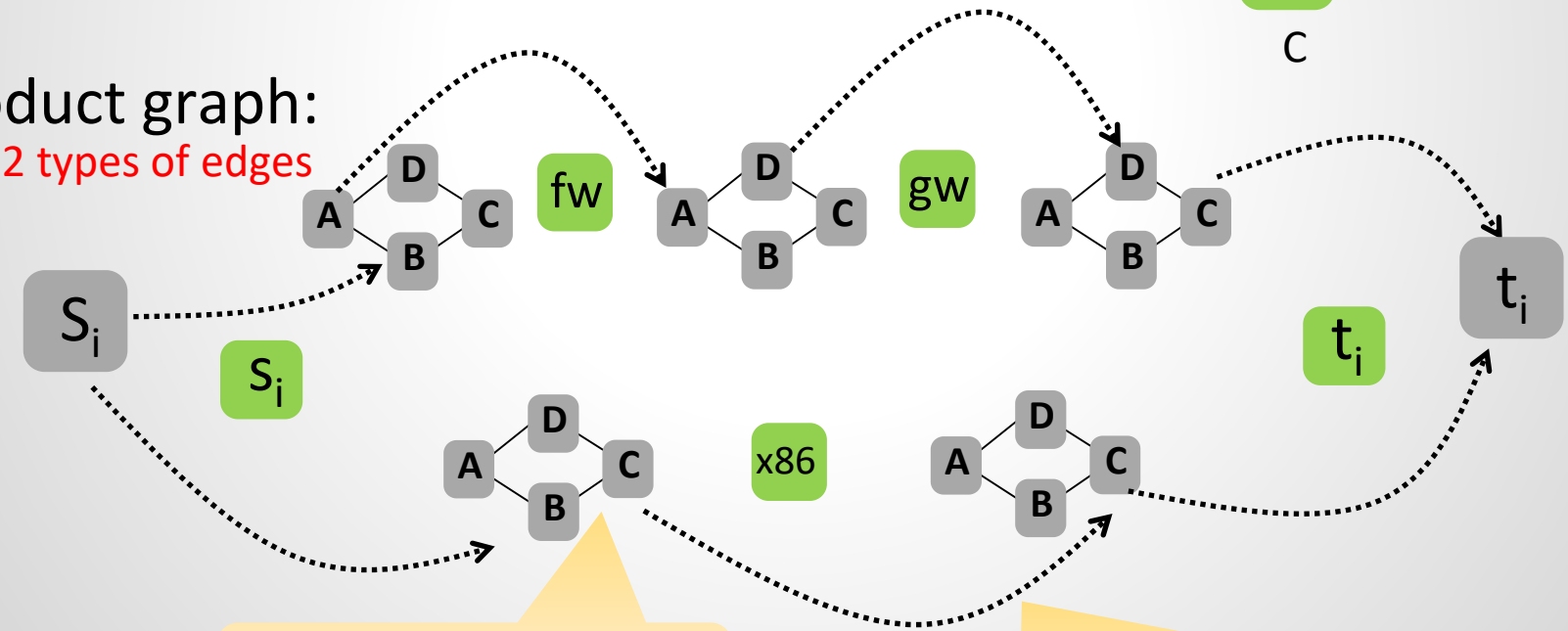


i^{th} request r_i:



Product graph:

with 2 types of edges

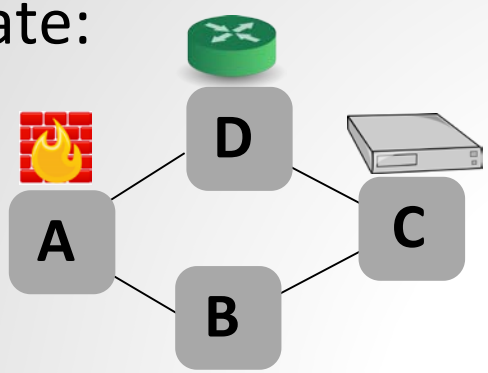


Routing edge: graph edge on same layer

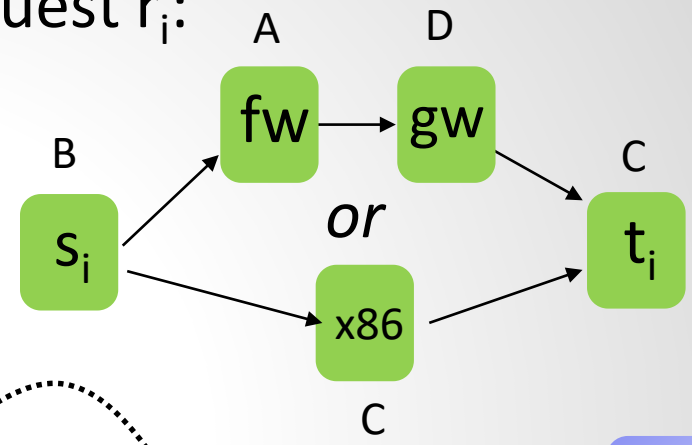
Processing edge: processing happens on C: connect C to C in next layer!

Approximations still possible for trees: Product graphs and randomized rounding

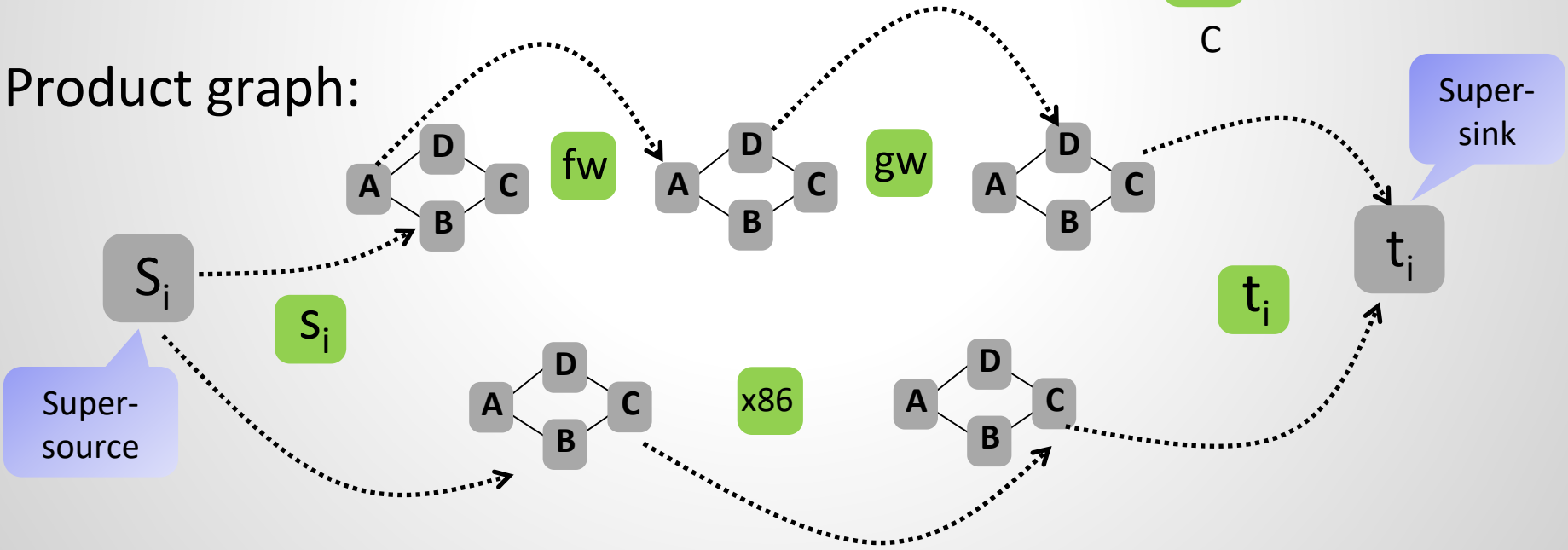
Substrate:



i^{th} request r_i:

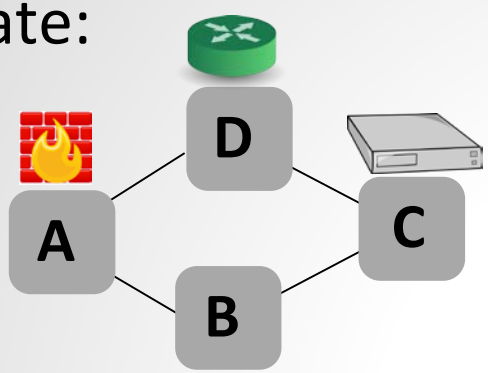


Product graph:

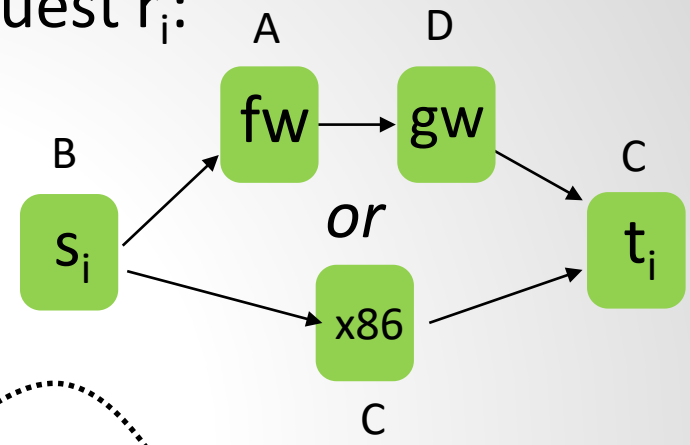


Approximations still possible for trees: Product graphs and randomized rounding

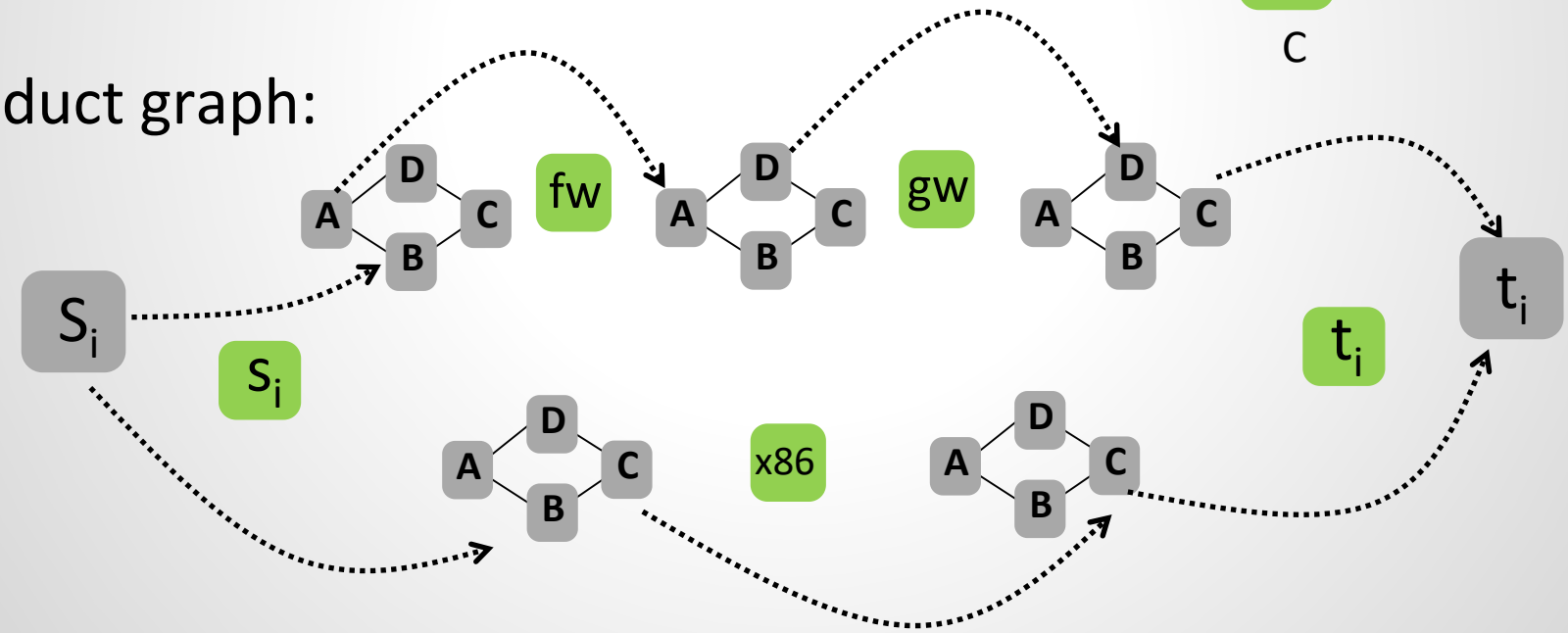
Substrate:



i^{th} request r_i:



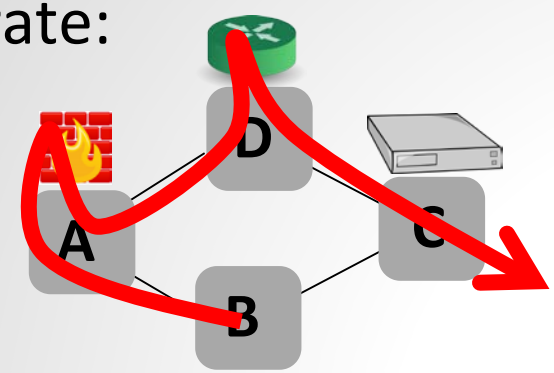
Product graph:



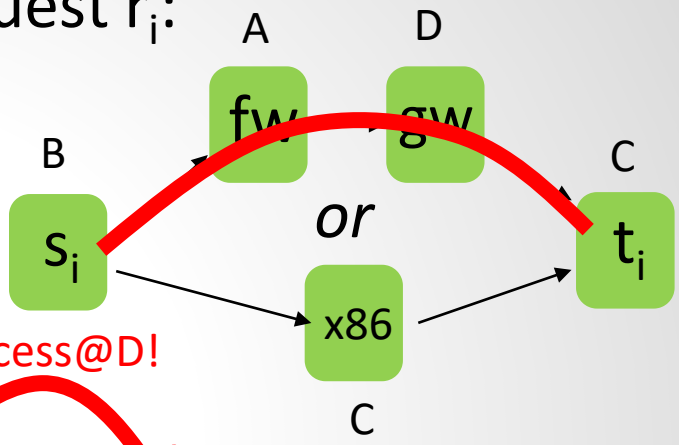
Any (s_i,t_i) flow presents a route of the request r_i!

Approximations still possible for trees: Product graphs and randomized rounding

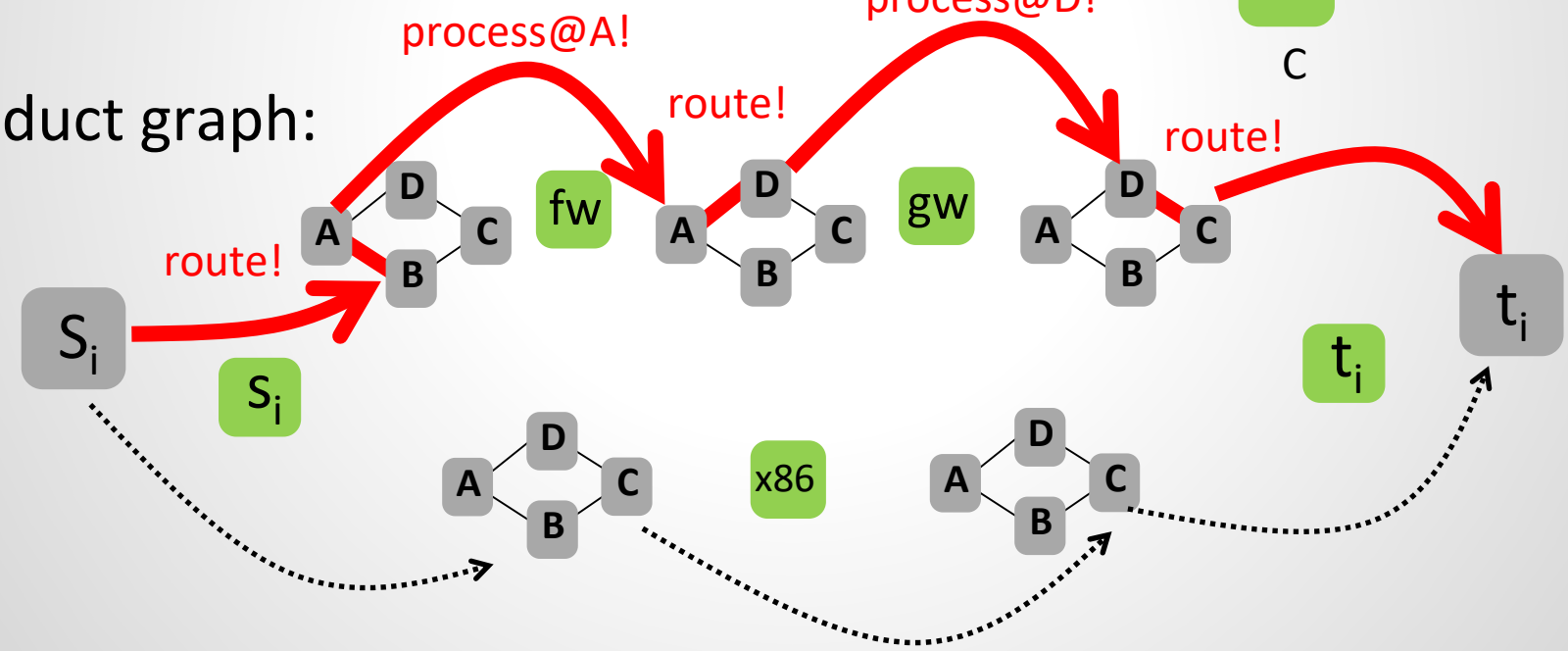
Substrate:



i^{th} request r_i:



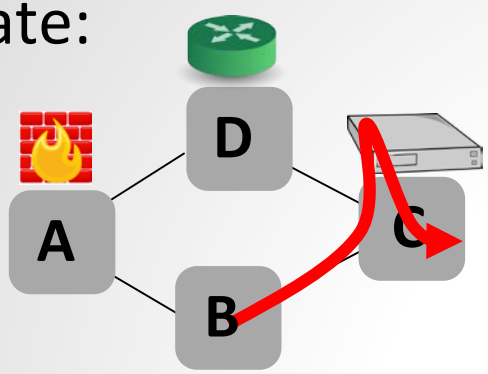
Product graph:



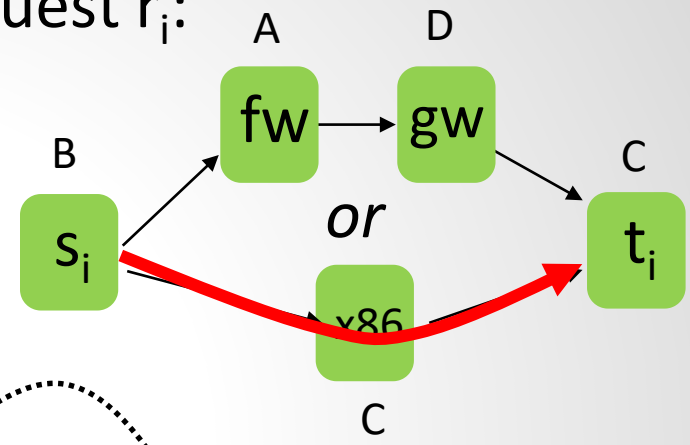
Any (s_i,t_i) flow presents a route of the request r_i!

Approximations still possible for trees: Product graphs and randomized rounding

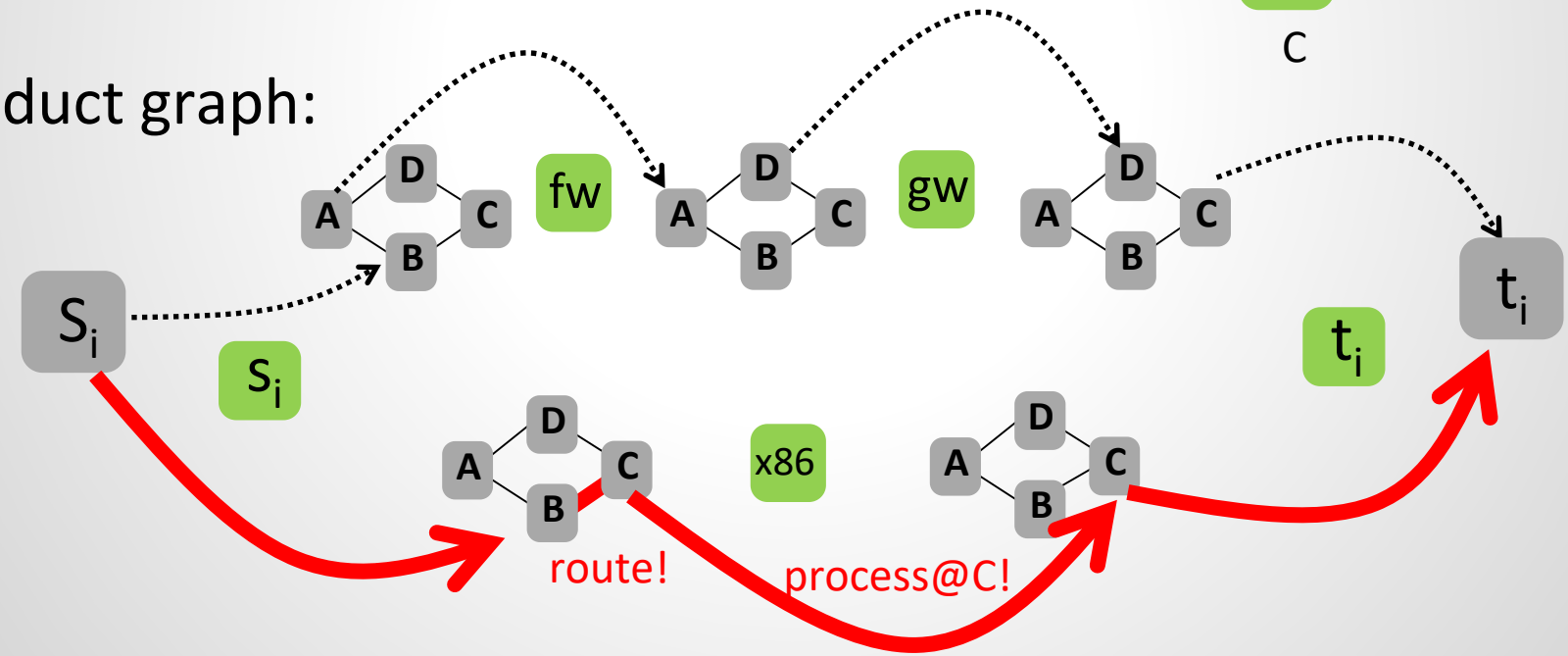
Substrate:



i^{th} request r_i:



Product graph:



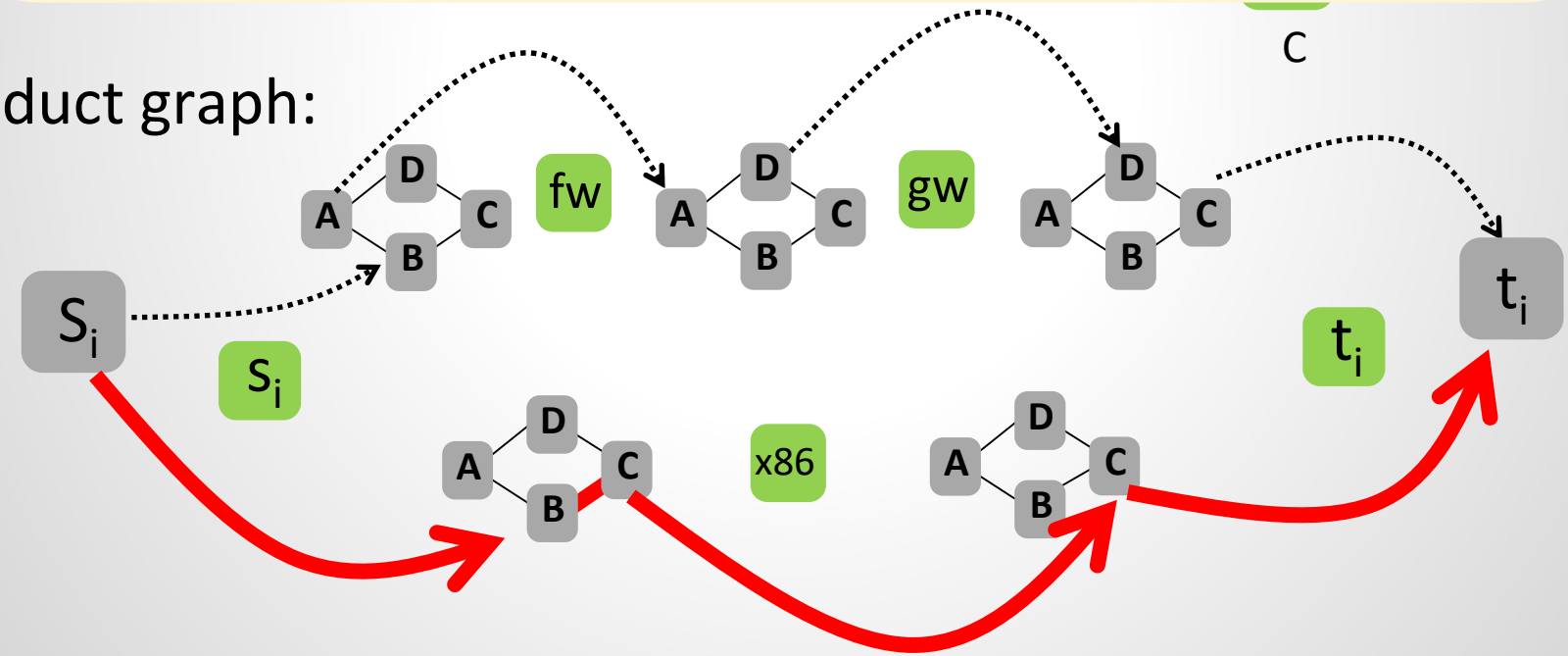
Any (s_i,t_i) flow presents a route of the request r_i!

Approximations still possible for trees:
Product graphs and randomized rounding

Su

This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g., randomized rounding).

Product graph:



Any (s_i,t_i) flow presents a route of the request r_i!

Approximations still possible for trees:
Product graphs and randomized rounding

Su

This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g., randomized rounding).

P

But note: cannot keep track of dependencies across stages (e.g., allocation on links or nodes): may yield oversubscription.

Any (s_i,t_i) flow presents a route of the request r_i!

What about requests which arrive over time?

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$	$\max B_j^T \cdot Y_j \;\; s.t.$
$Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$	$A_j \cdot Y_j \leq C$
$X, Z_j \geq \mathbf{0}$	$D_j \cdot Y_j \leq \mathbf{1}$
	$Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

Primal and Dual

The Buchbinder-Naor Approach

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$	$\max B_j^T \cdot Y_j \;\; s.t.$
$Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$	$A_j \cdot Y_j \leq C$
$X, Z_j \geq \mathbf{0}$	$D_j \cdot Y_j \leq \mathbf{1}$
	$Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Formulate the packing (dual) LP: Maximize profit

(Note: dynamic LP!)

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

Primal and Dual

The Buchbinder-Naor Approach

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

s.t. constraints

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

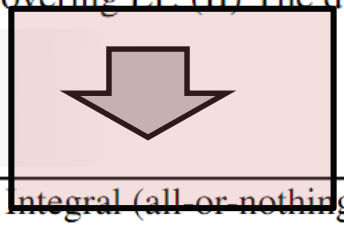
   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$	$\max B_j^T \cdot Y_j \;\; s.t.$
$Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$	$A_j \cdot Y_j \leq C$
$X, Z_j \geq \mathbf{0}$	$D_j \cdot Y_j \leq \mathbf{1}$
	$Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.



Buchbinder&Naor

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$	$\max B_j^T \cdot Y_j \;\; s.t.$
$Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$	$A_j \cdot Y_j \leq C$
$X, Z_j \geq \mathbf{0}$	$D_j \cdot Y_j \leq \mathbf{1}$
	$Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure) ← optimal embedding!
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$	$\max B_j^T \cdot Y_j \;\; s.t.$
$Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$	$A_j \cdot Y_j \leq C$
$X, Z_j \geq \mathbf{0}$	$D_j \cdot Y_j \leq \mathbf{1}$
	$Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. ~~$f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) \cdot f_{j,\ell} \in A_j\}$~~ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $g_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

Embedding cost vs profit?

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$	$\max B_j^T \cdot Y_j \;\; s.t.$
$Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$	$A_j \cdot Y_j \leq C$
$X, Z_j \geq \mathbf{0}$	$D_j \cdot Y_j \leq \mathbf{1}$
	$Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

If cheap: accept and update primal variables (always feasible solution)

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

Else reject

The Buchbinder-Naor Approach

Primal and Dual

(I)	(II)
$\min Z_j^T \cdot \mathbf{1} + X^T \cdot C \;\; s.t.$ $Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T$ $X, Z_j \geq \mathbf{0}$	$\max B_j^T \cdot Y_j \;\; s.t.$ $A_j \cdot Y_j \leq C$ $D_j \cdot Y_j \leq \mathbf{1}$ $Y_j \geq \mathbf{0}$

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j,\ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j,\ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row e : If $A_{e,(j,\ell)} \neq 0$ do

$$x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/}$$

   (c) $z_j \leftarrow b_j - \gamma(j,\ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

How to do this optimally?!

Further Reading

Virtual Network Embedding Approximations: Leveraging Randomized Rounding
Matthias Rost and Stefan Schmid.
IEEE/ACM Transactions on Networking (**TON**), 2019.

Parametrized Complexity of Virtual Network Embeddings: Dynamic & Linear Programming Approximations
Matthias Rost, Elias Döhne, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), January 2019.

Charting the Complexity Landscape of Virtual Network Embeddings
Matthias Rost and Stefan Schmid.
IFIP Networking, Zurich, Switzerland, May 2018.

Competitive and Deterministic Embeddings of Virtual Networks
Guy Even, Moti Medina, Gregor Schaffrath, and Stefan Schmid.
Journal Theoretical Computer Science (**TCS**), Elsevier, 2013.

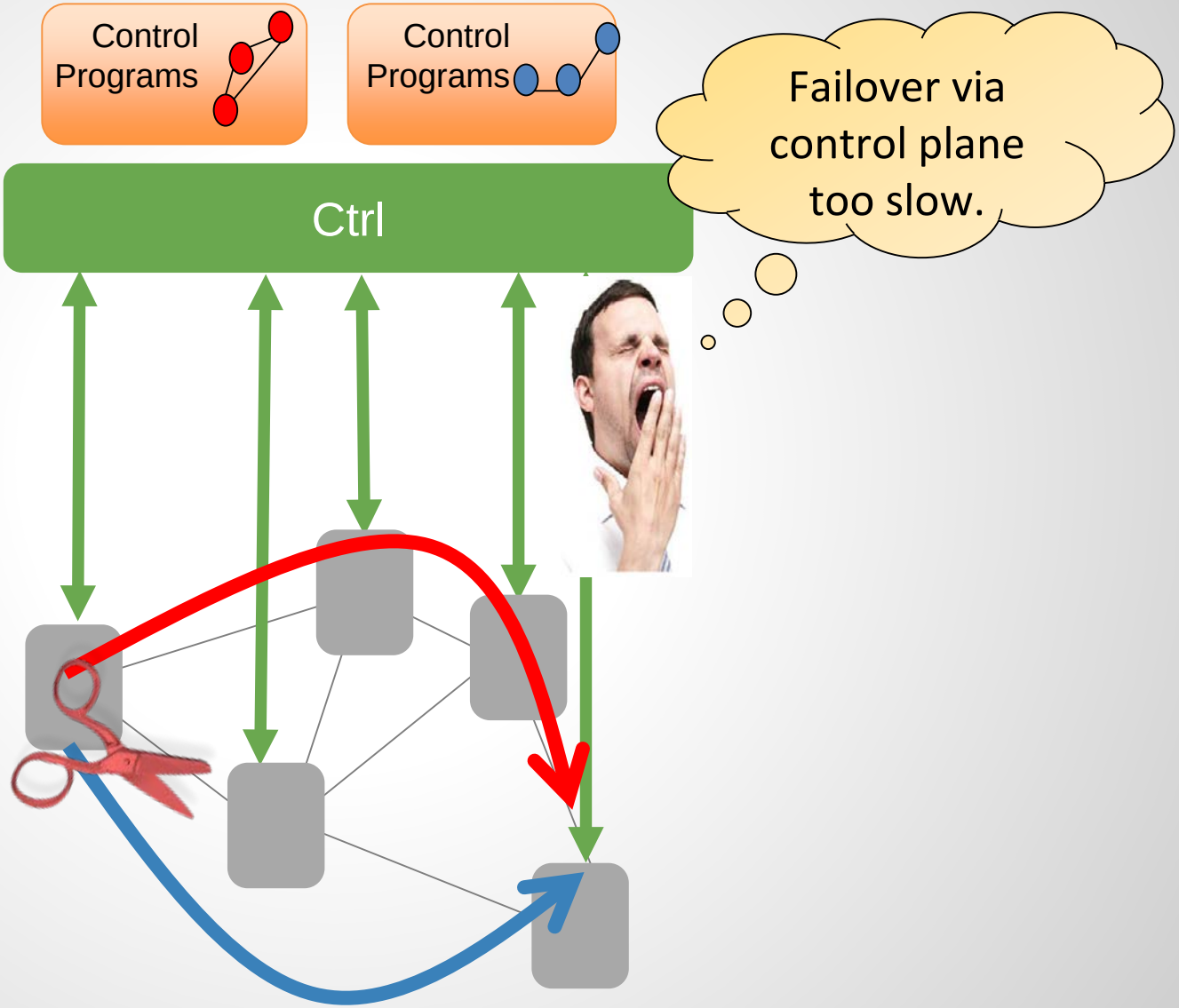
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

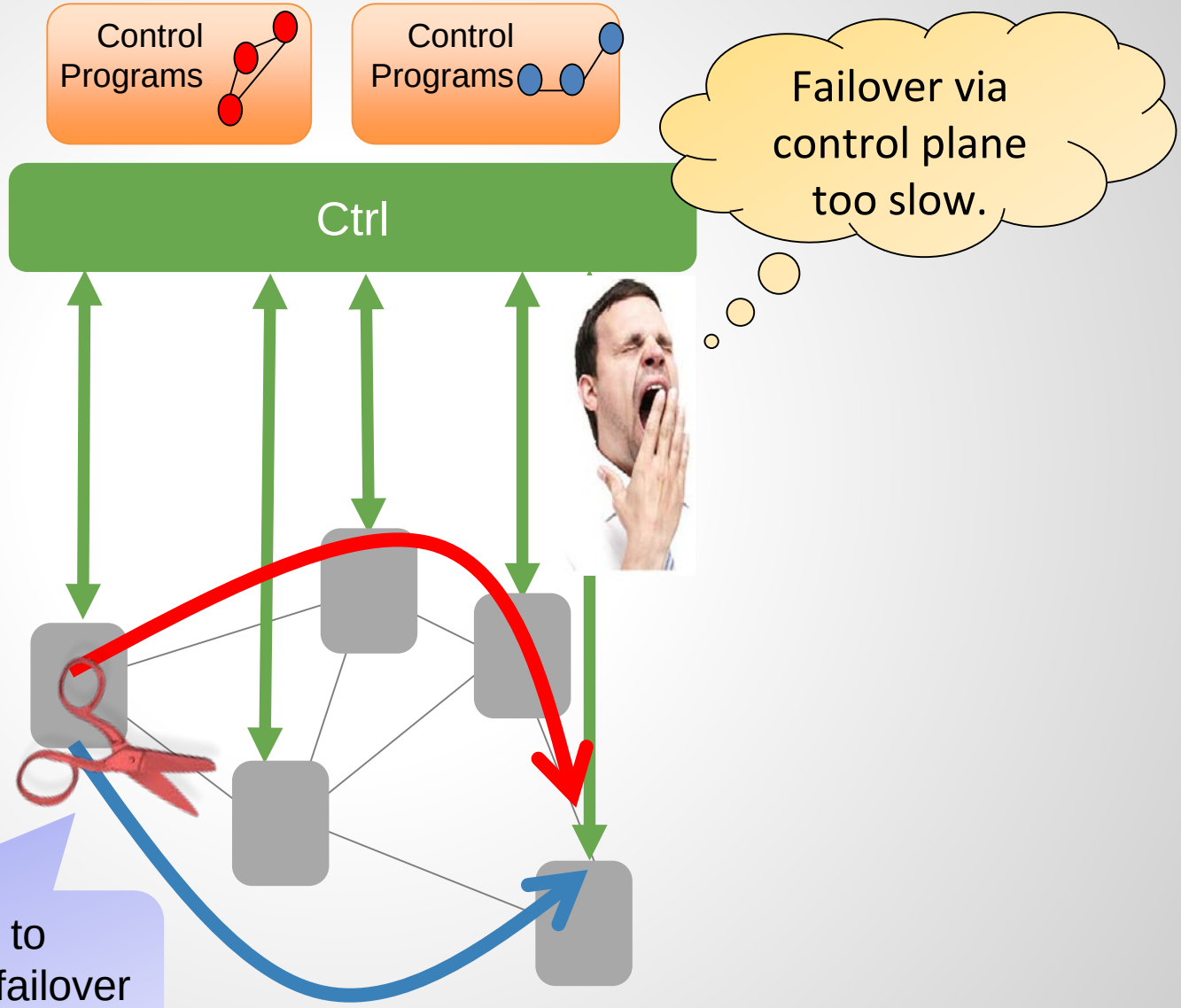
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- **Algorithms III: Flexible fast rerouting**
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

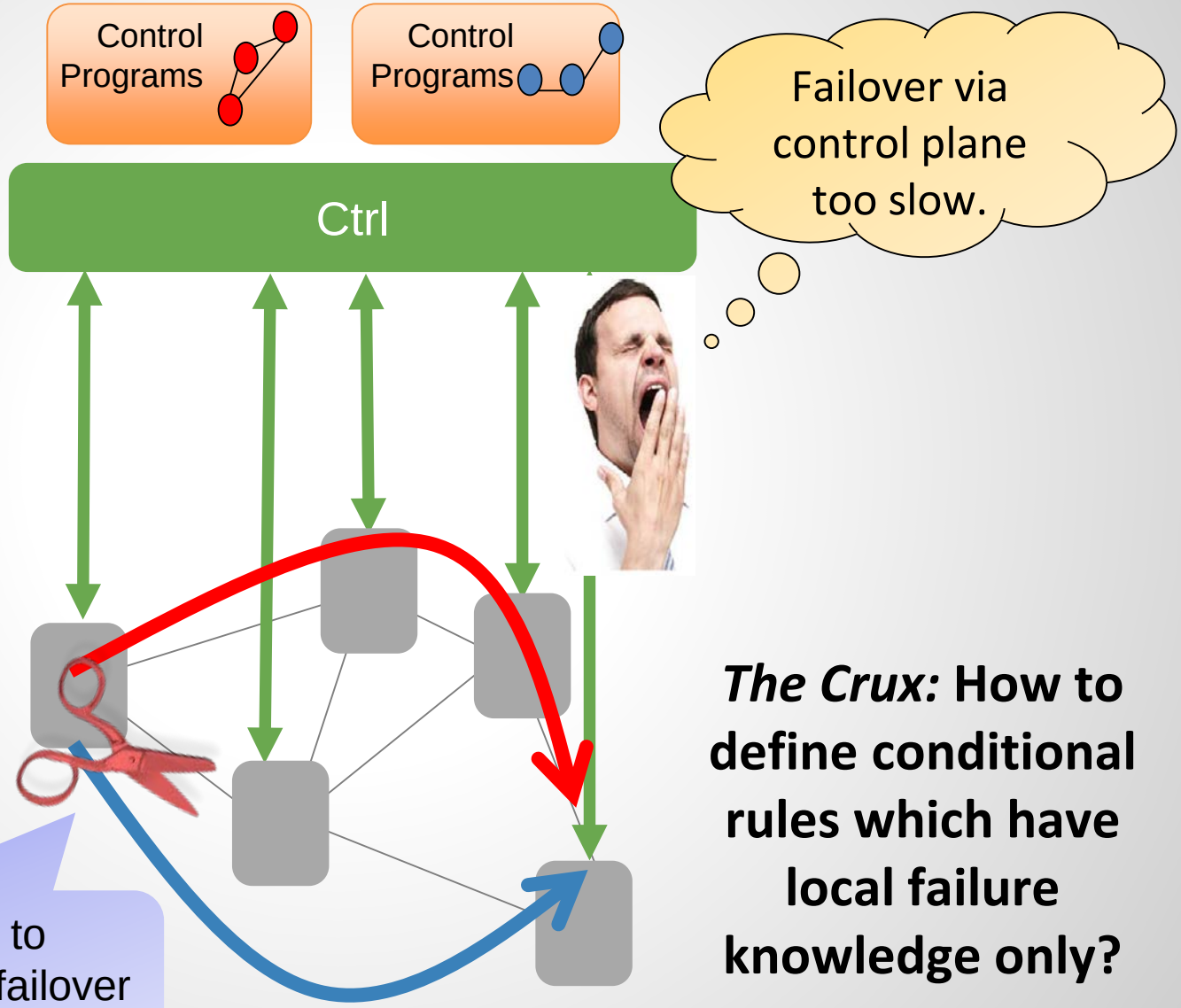
Motivation: Fast Rerouting



Motivation: Fast Rerouting



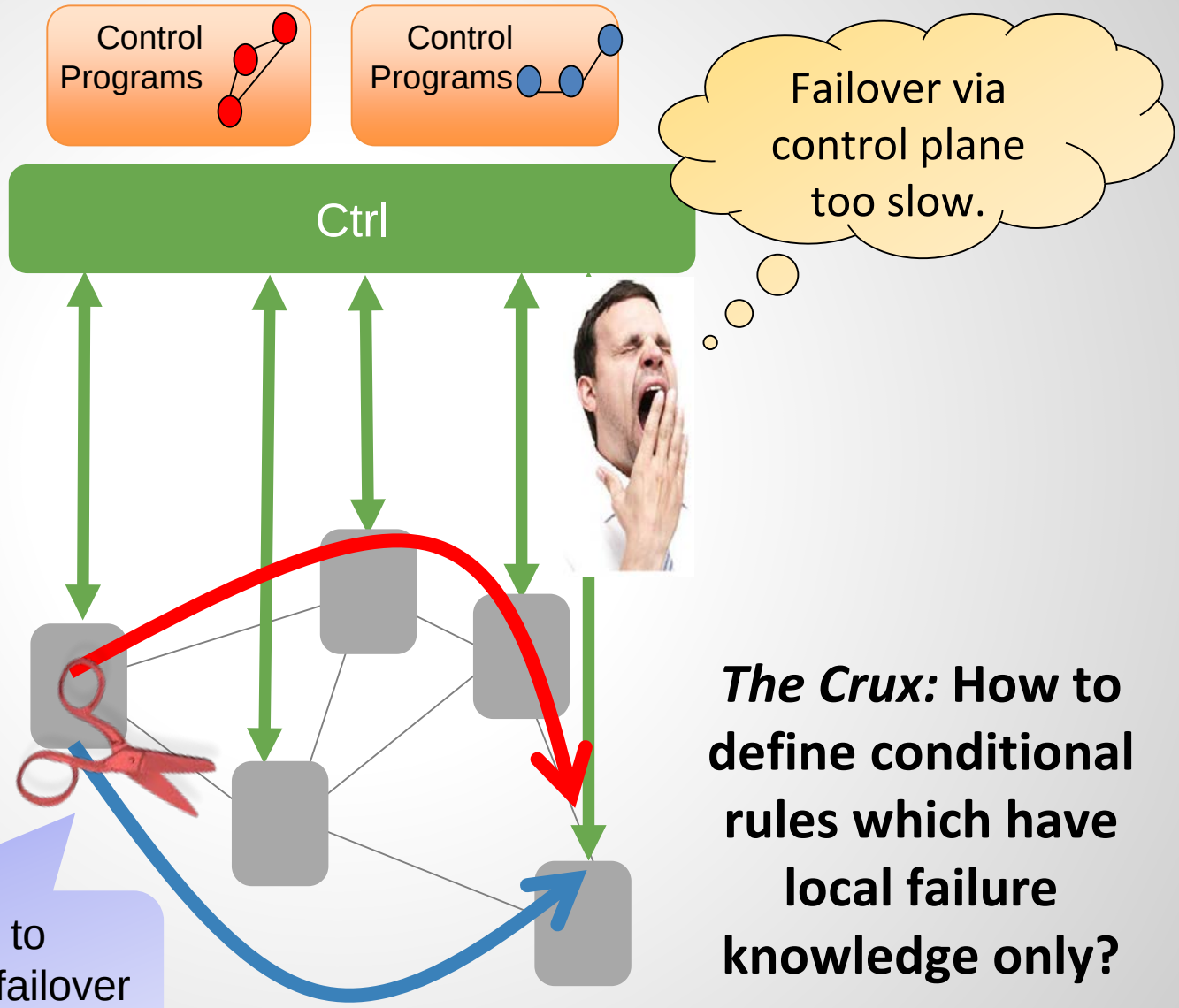
Motivation: Fast Rerouting



The Crux: **How to define conditional rules which have local failure knowledge only?**

Motivation: Fast Rerouting

Open problem: How many link failures can be tolerated in k-connected network without going through controller?



***The Crux:* How to define conditional rules which have local failure knowledge only?**

OpenFlow etc. allow to preconfigure conditional failover rules: 1st line of defense!

Solution: Use Arborescences (Chiesa et al.)

- Assume:
  - *k*-connected network *G*
  - destination *d*
  - *G* decomposed into *k* *d*-rooted arc-disjoint spanning arborescences

Known result: always exist in k-connected graphs (efficient)

Basic principle:

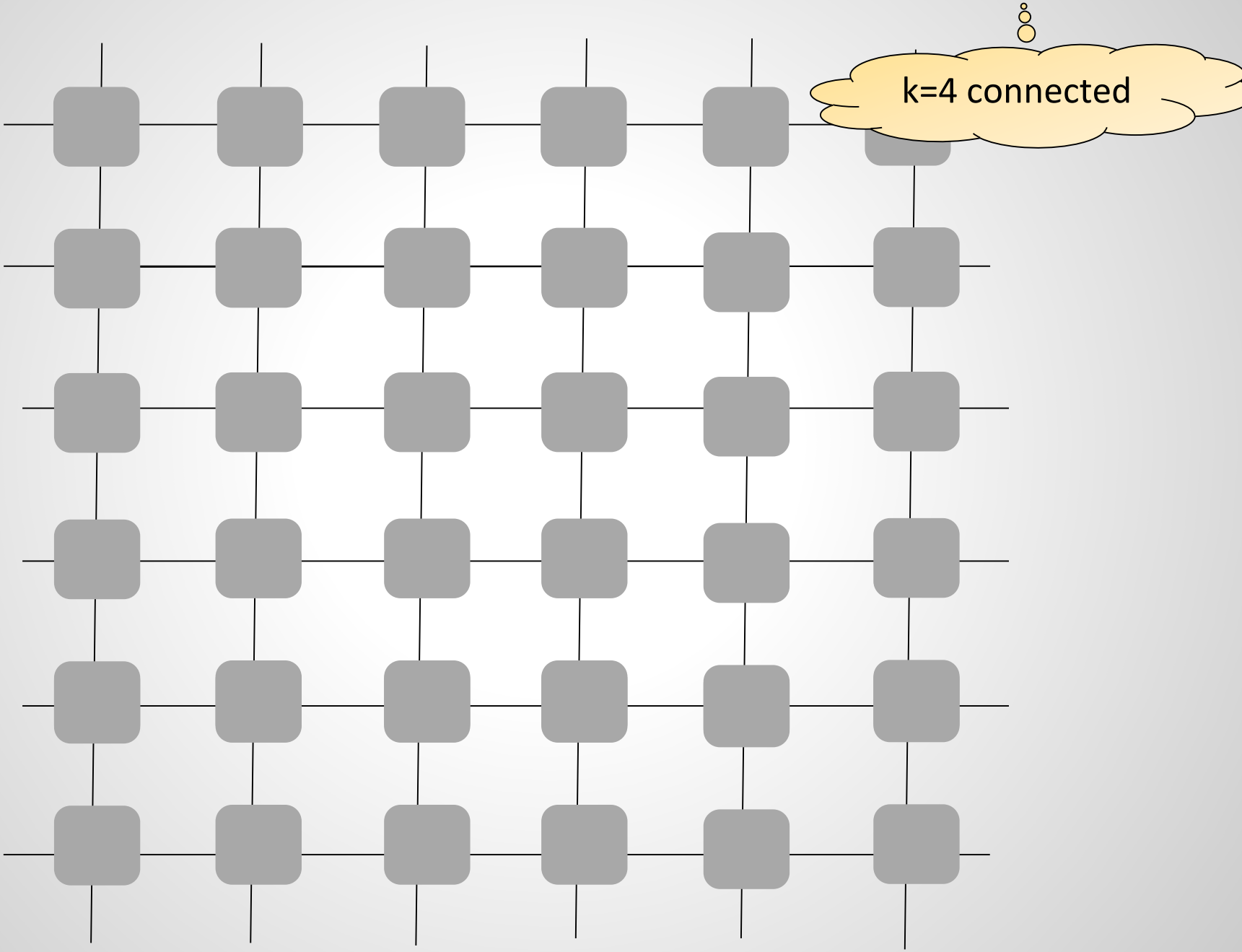
- Route along fixed arborescence ("directed spanning tree") towards the destination *d*
- If packet hits a failed edge at vertex *v*, reroute along a different arborescence

The Crux: which arborescence to choose next? Influences resiliency!

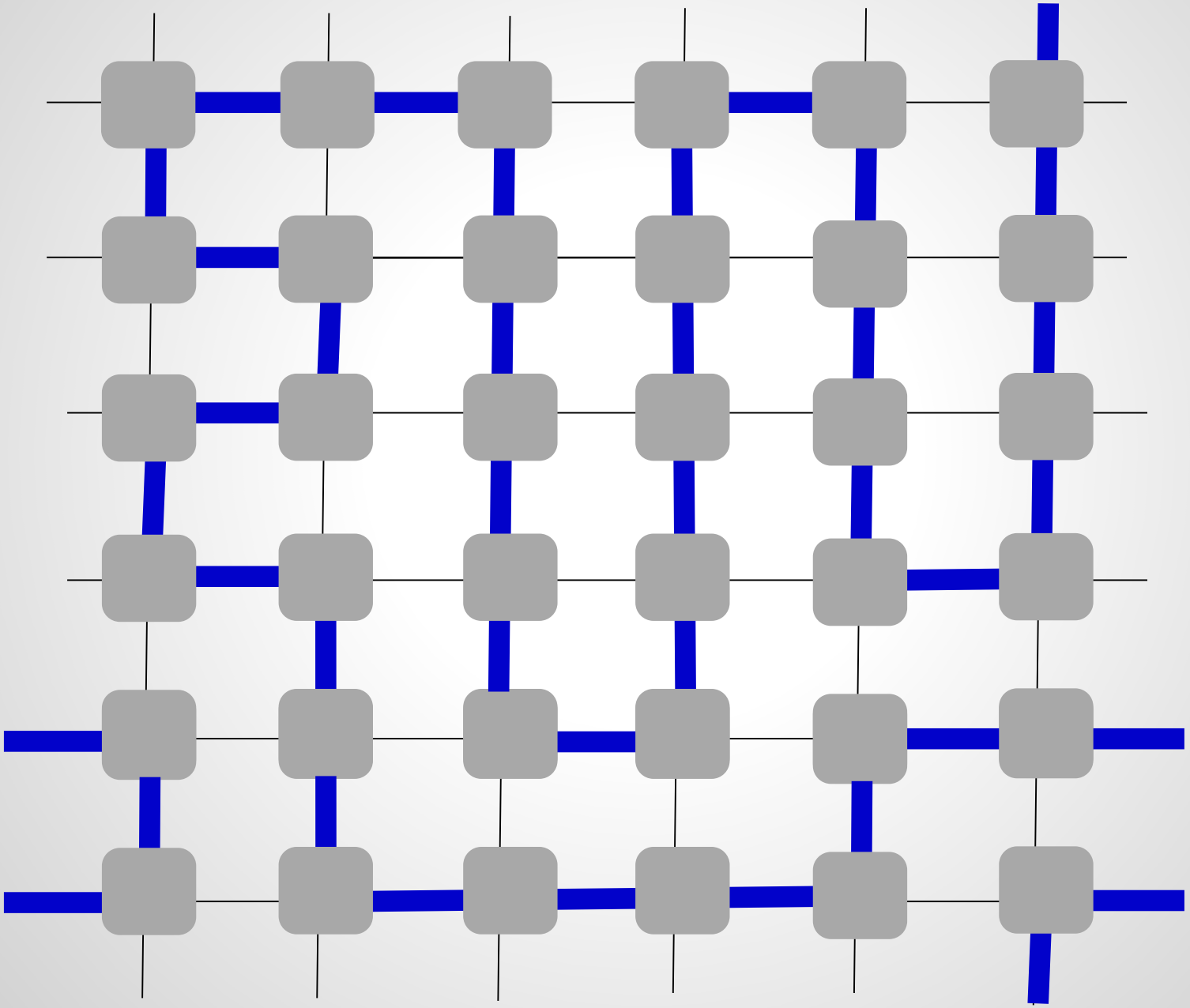
Simple Example: Hamilton Cycle

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

Example: 3-Resilient Routing Function for 2-dim Torus

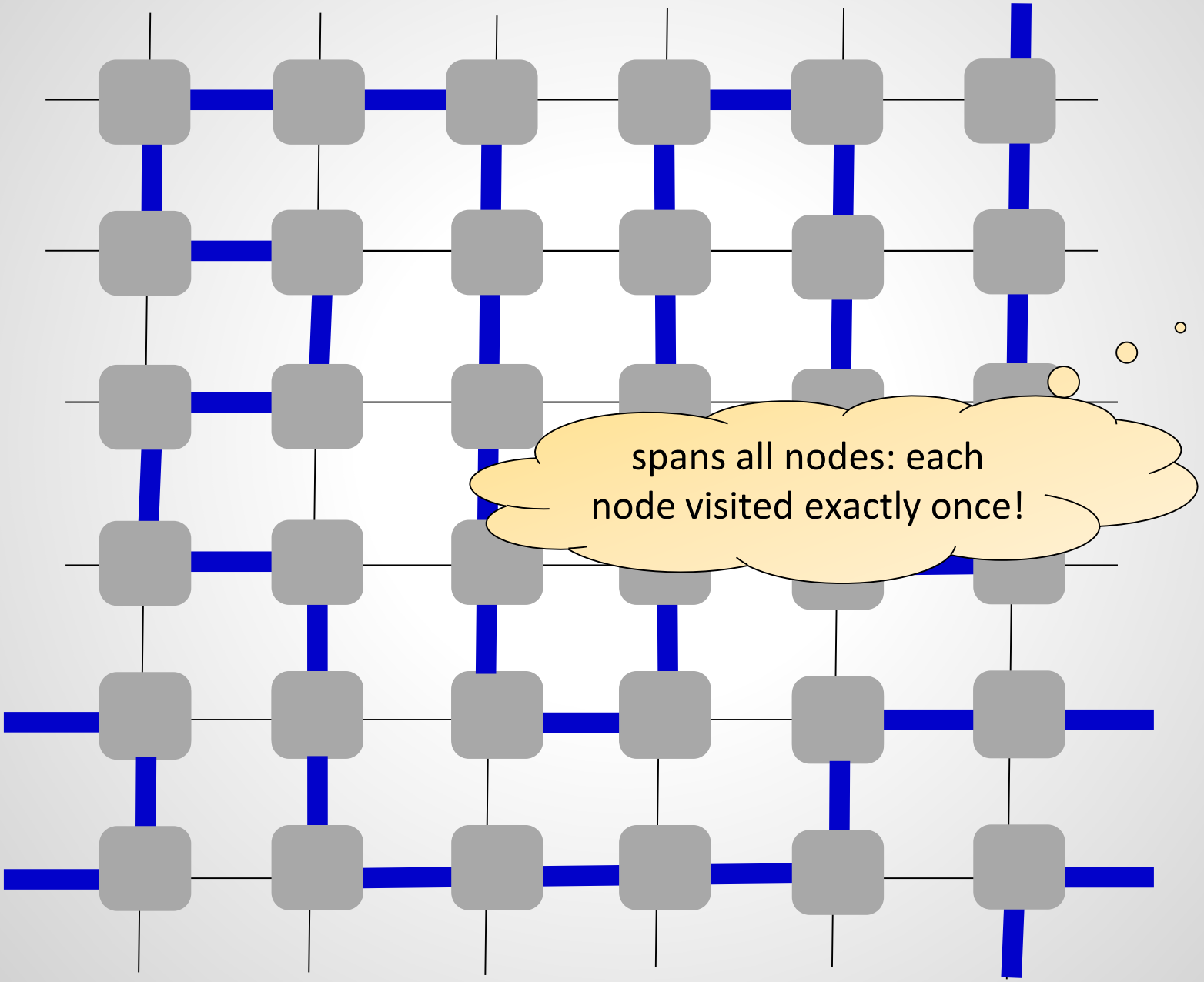


Example: 3-Resilient Routing Function for 2-dim Torus

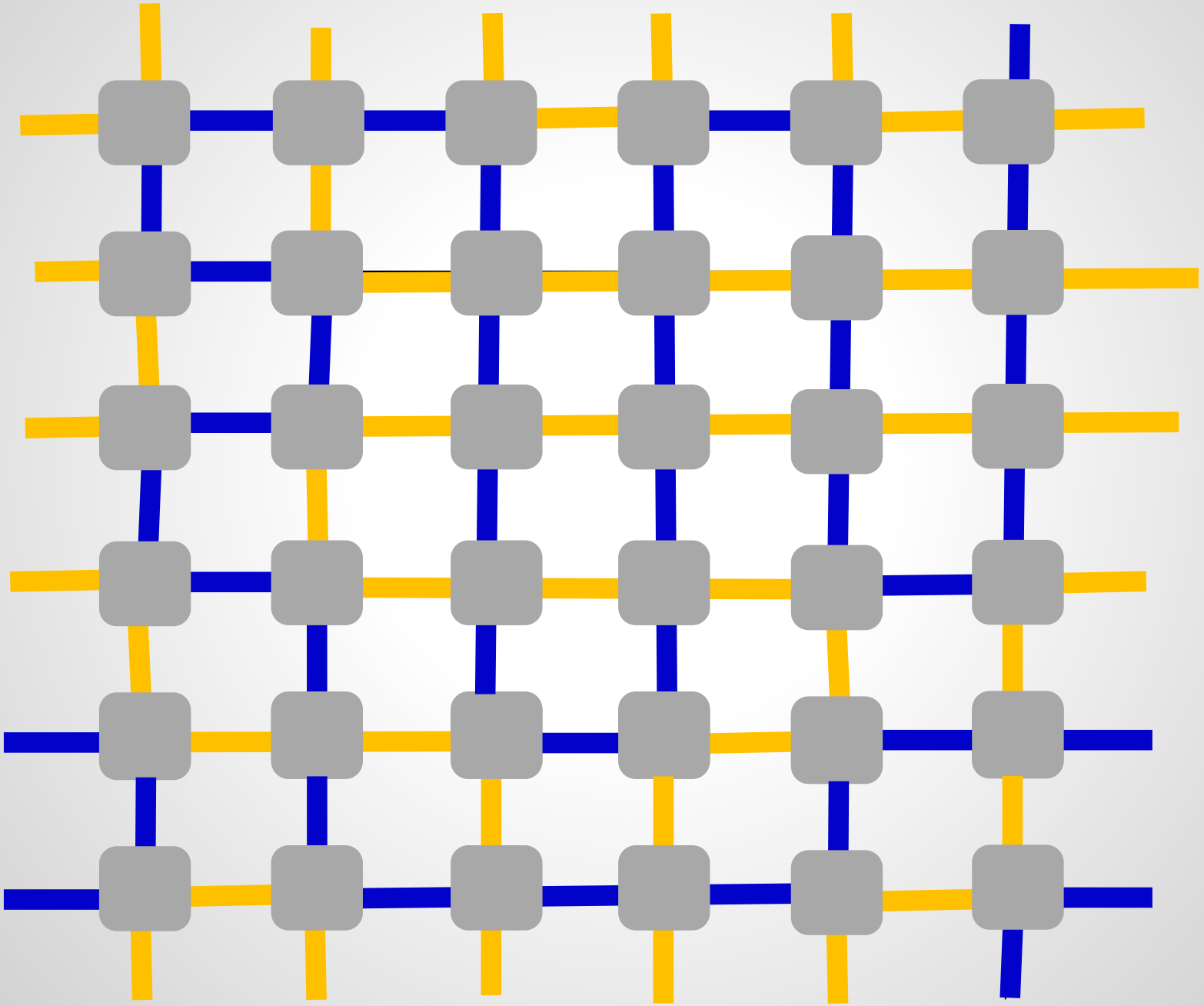


Edge-Disjoint Hamilton Cycle 1

Example: 3-Resilient Routing Function for 2-dim Torus

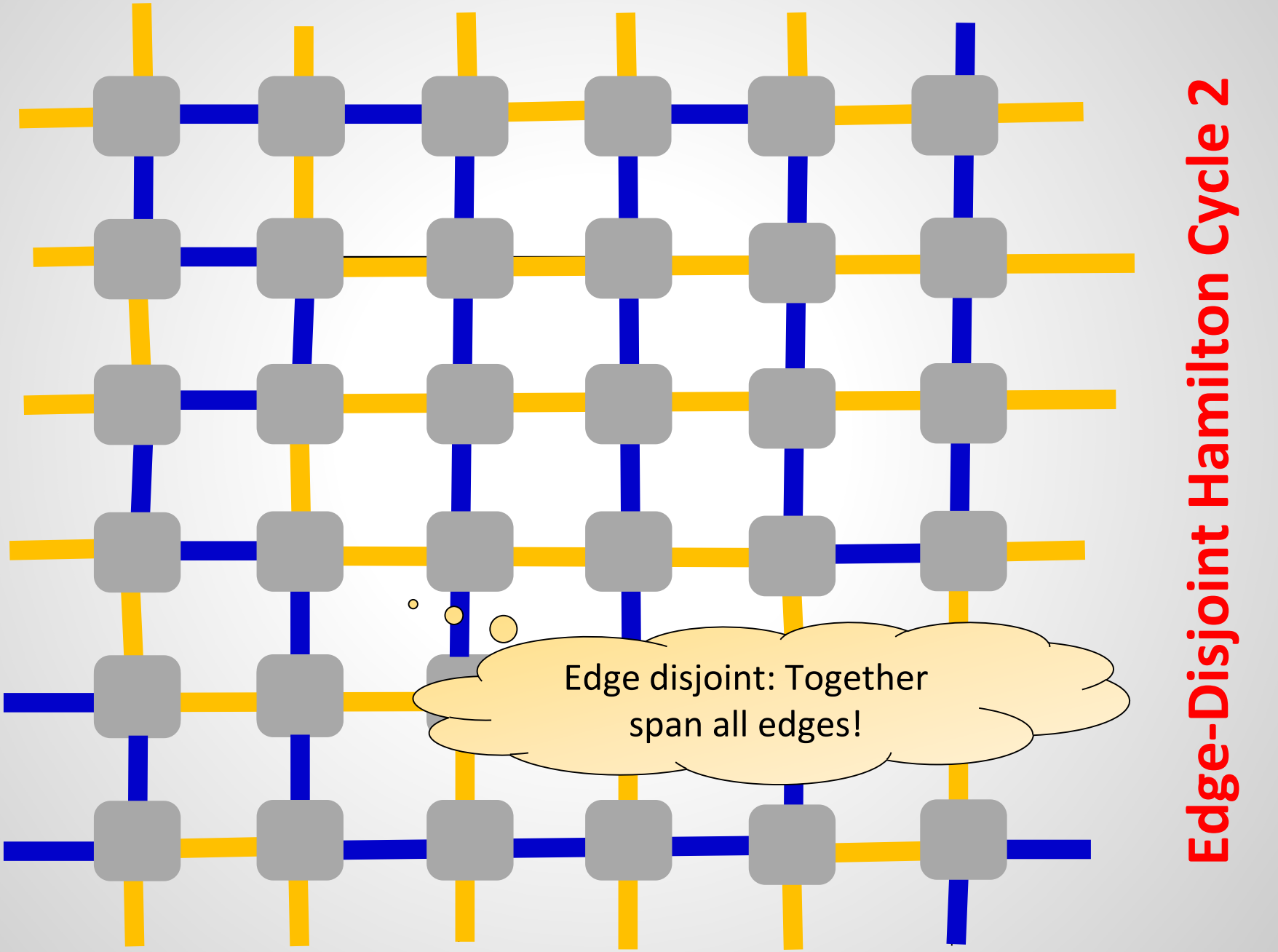


Example: 3-Resilient Routing Function for 2-dim Torus

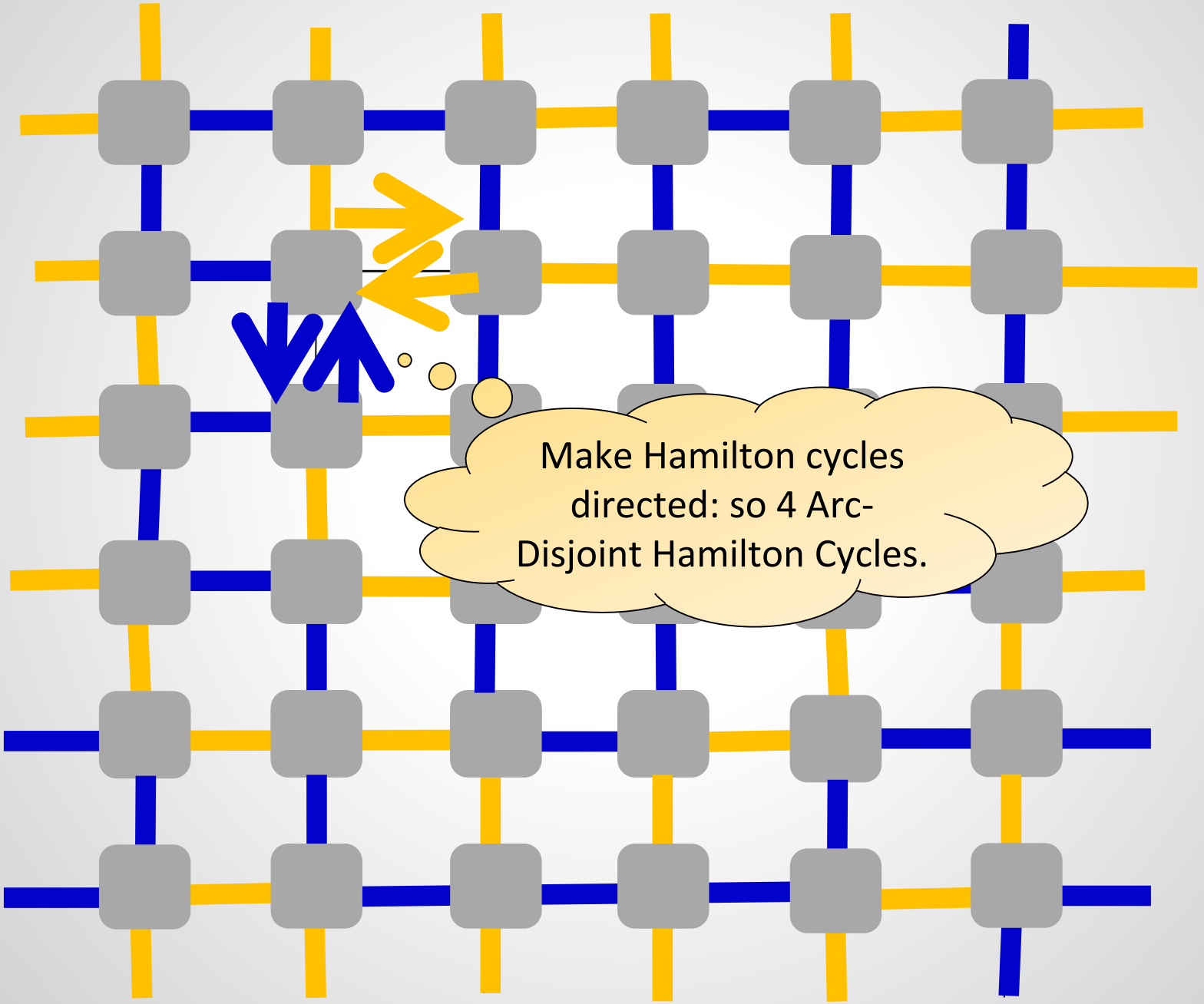


Edge-Disjoint Hamilton Cycle 2

Example: 3-Resilient Routing Function for 2-dim Torus

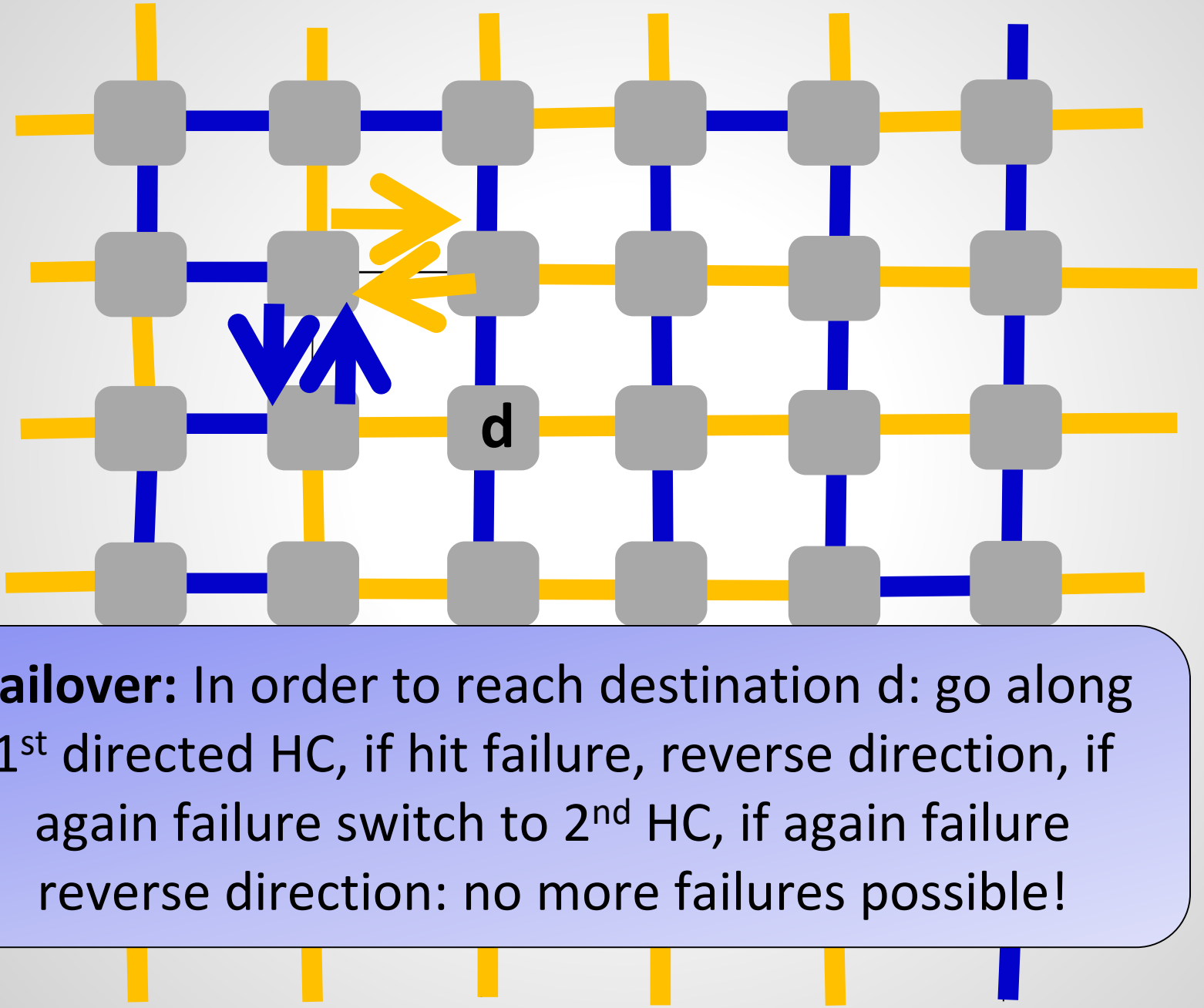


Example: 3-Resilient Routing Function for 2-dim Torus



4 Arc-Disjoint Arborescences

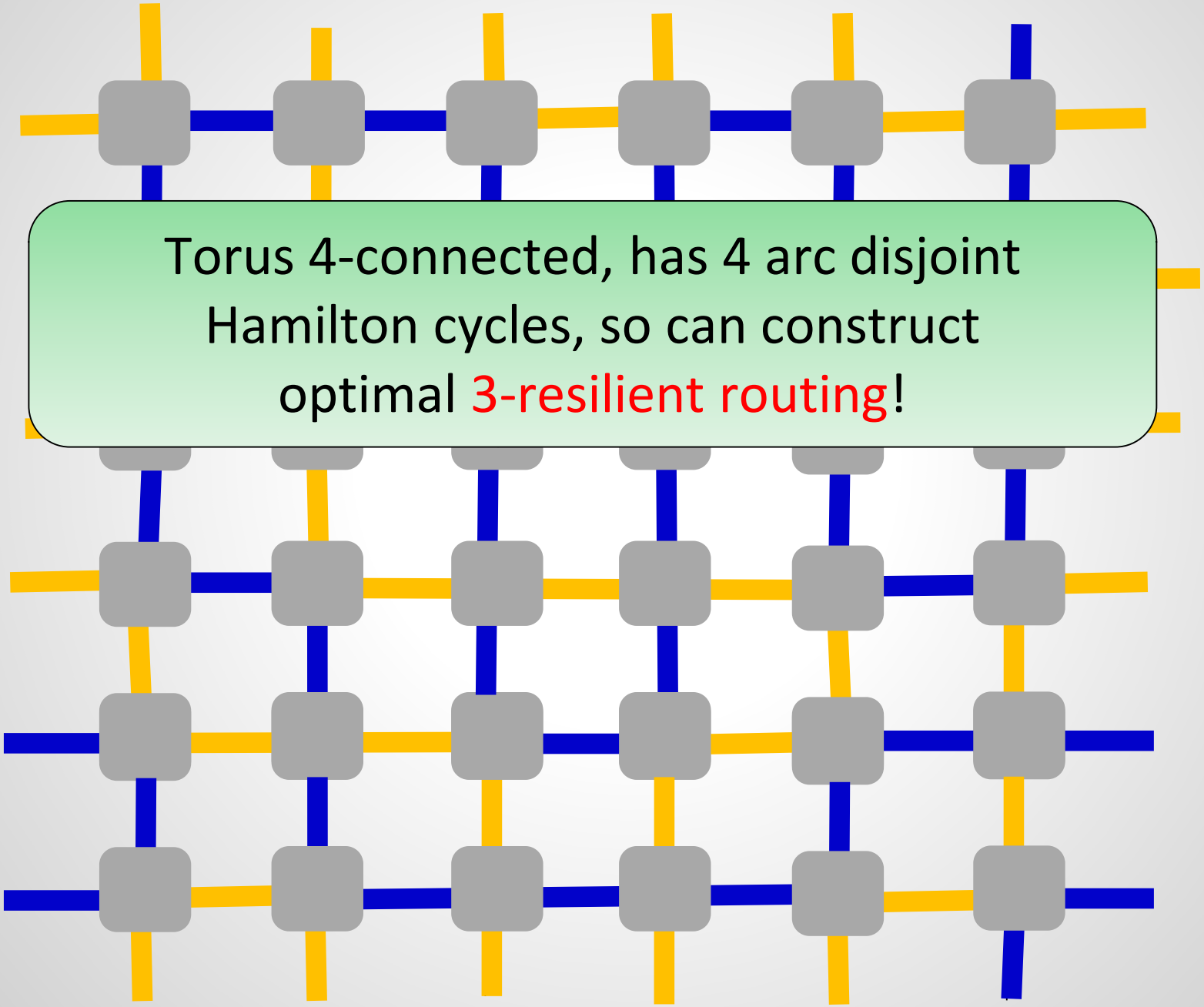
Example: 3-Resilient Routing Function for 2-dim Torus



Failover: In order to reach 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!

4 Arc-Disjoint Arborescences

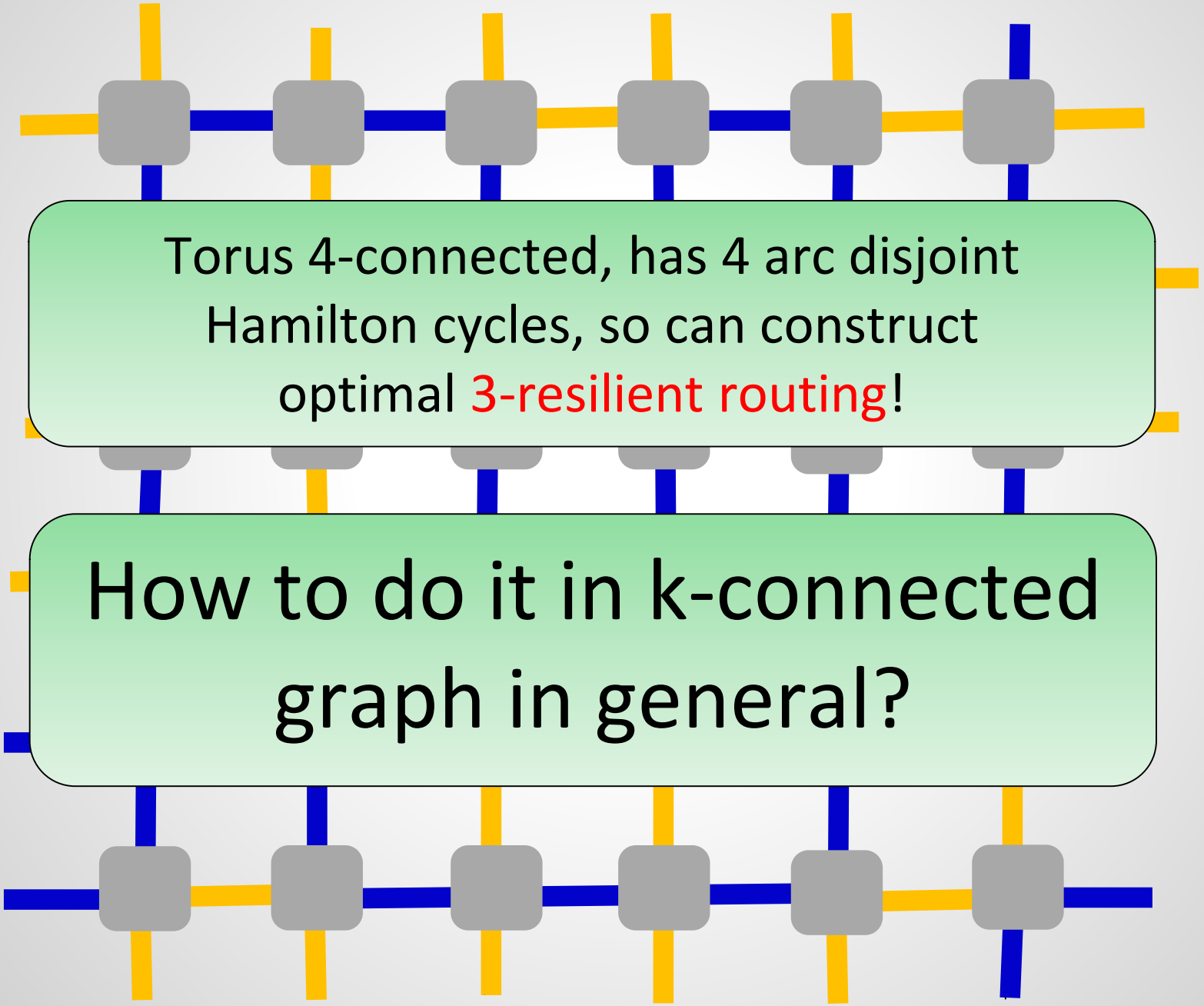
Example: 3-Resilient Routing Function for 2-dim Torus



Torus 4-connected, has 4 arc disjoint Hamilton cycles, so can construct optimal 3-resilient routing!

4 Arc-Disjoint Arborescences

Example: 3-Resilient Routing Function for 2-dim Torus



Torus 4-connected, has 4 arc disjoint Hamilton cycles, so can construct optimal 3-resilient routing!

How to do it in k-connected graph in general?

4 Arc-Disjoint Arborescences

Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Theory of demand-aware networks
- Security aspects (if time permits)

Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- **Algorithms IV: Theory of demand-aware networks**
- Security aspects (if time permits)

Extra slides!

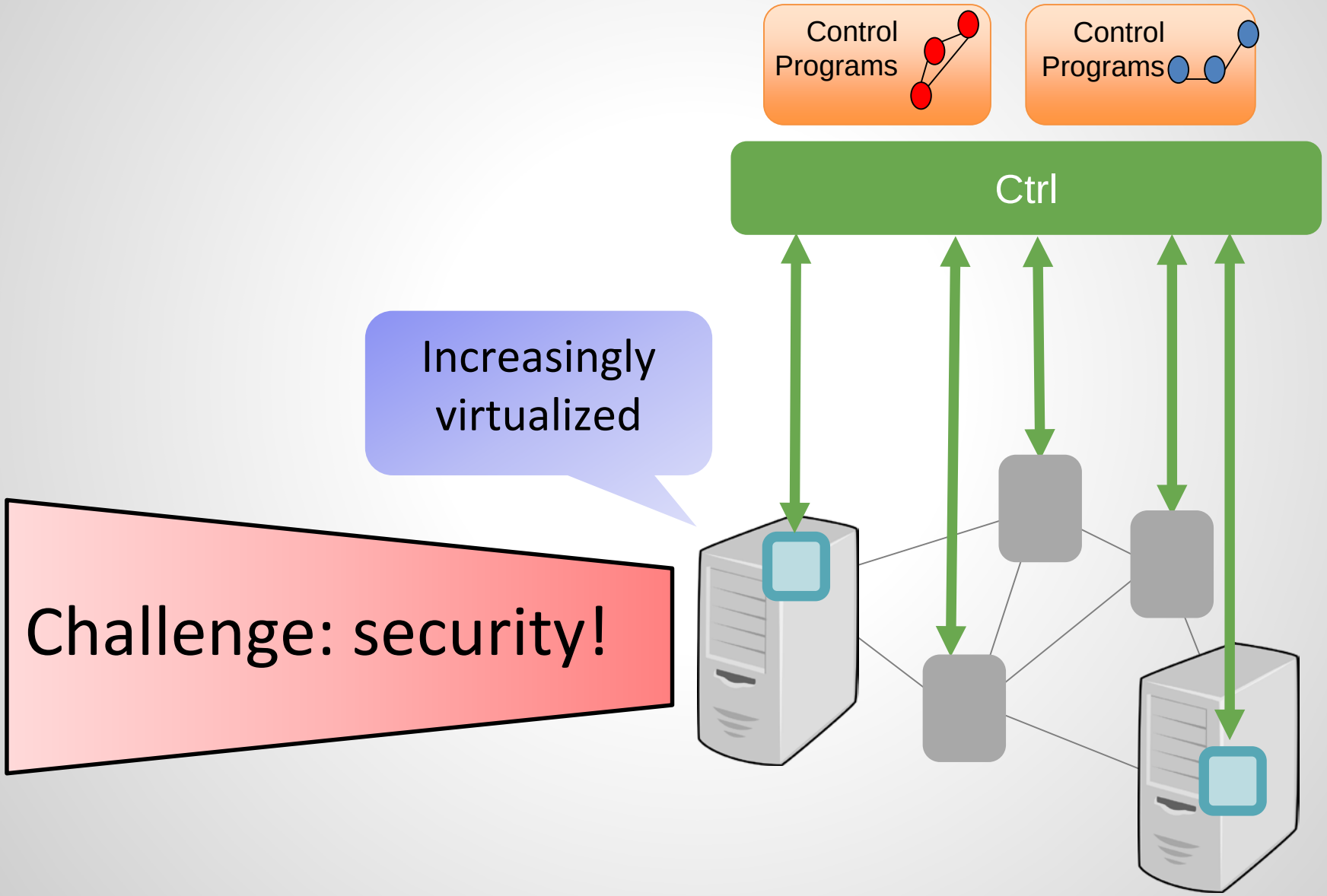
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Toward self-adjusting networks
- Security aspects (if time permits)

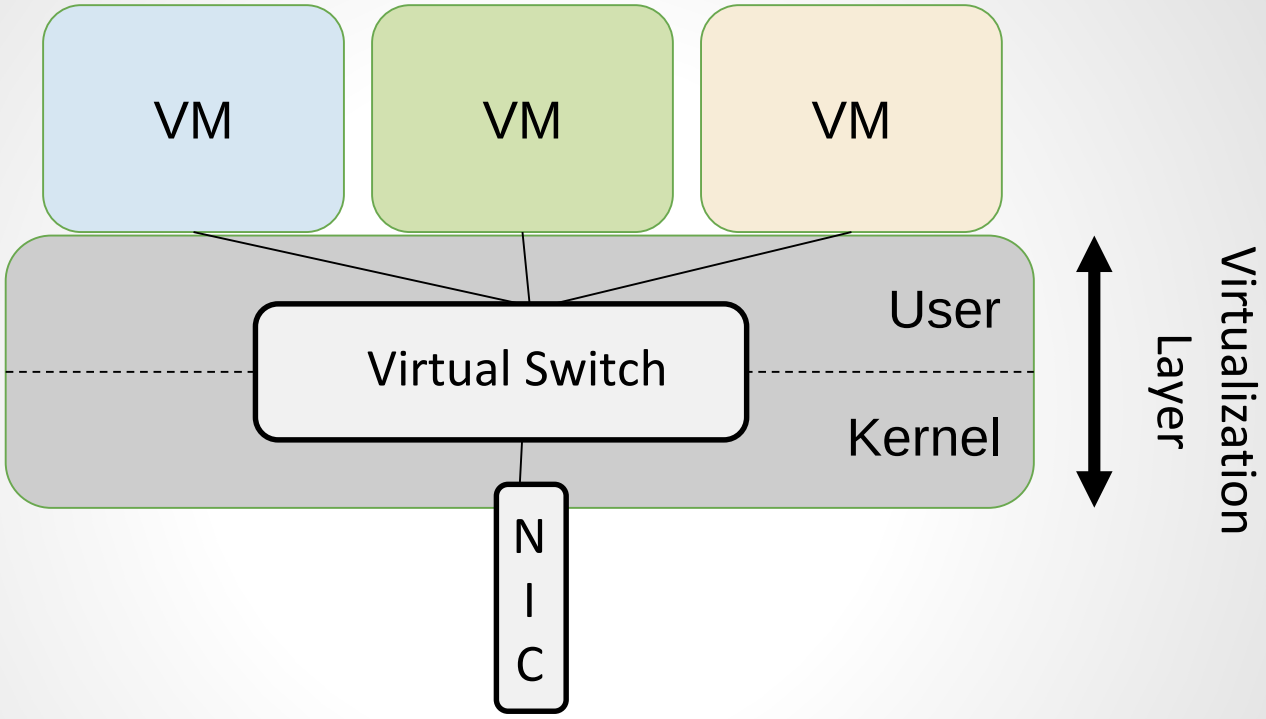
Roadmap

- A short recap:
  - Networking preliminaries
  - Opportunities and challenges of SDNs
  - On predictable performance
- Algorithms I: Flexible routing
- Algorithms II: Flexible embedding and slicing
- Algorithms III: Flexible fast rerouting
- Algorithms IV: Toward self-adjusting networks
- **Security aspects (if time permits)**

Programmable and Virtualized Networks



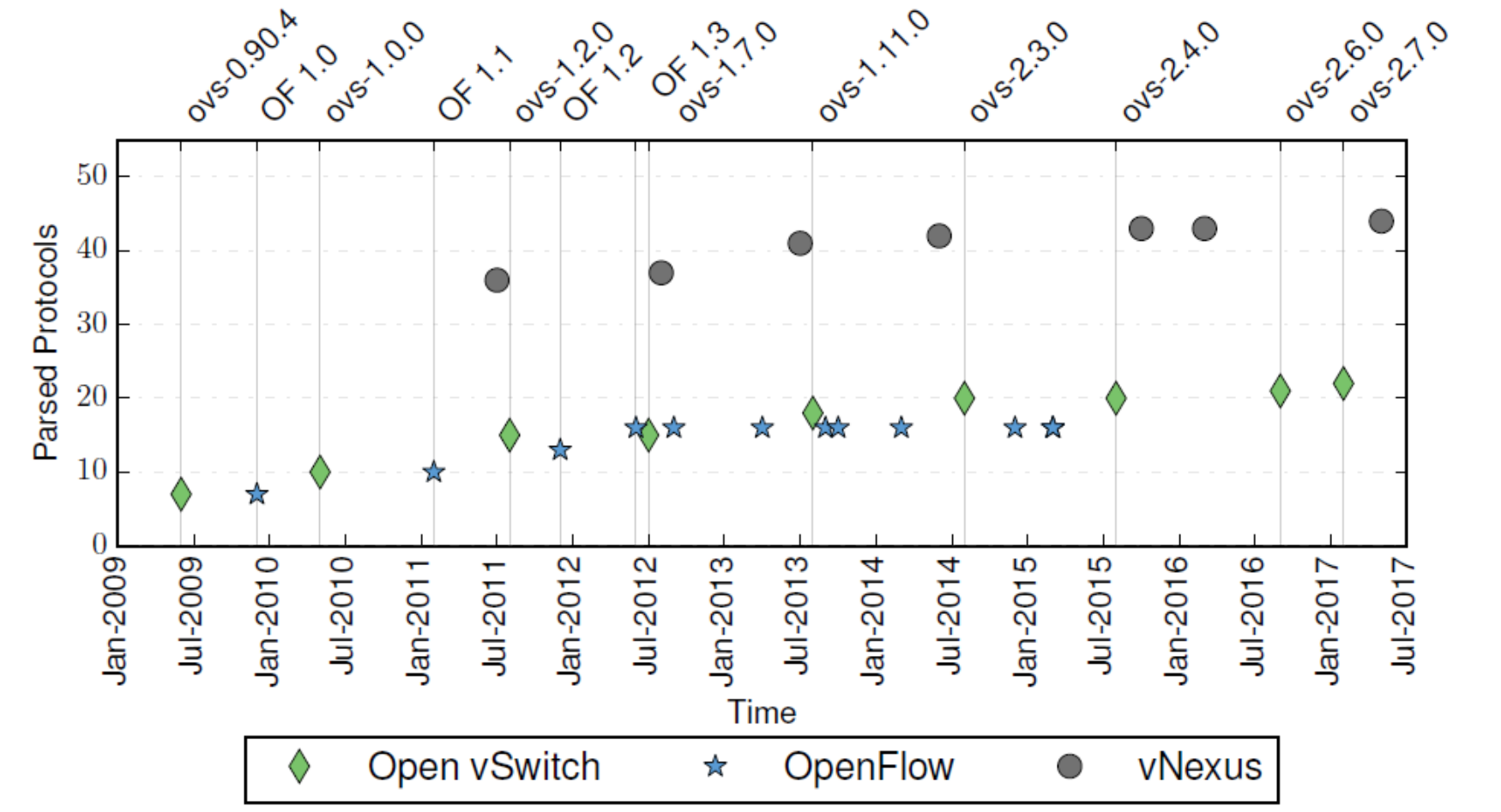
Virtual Switches



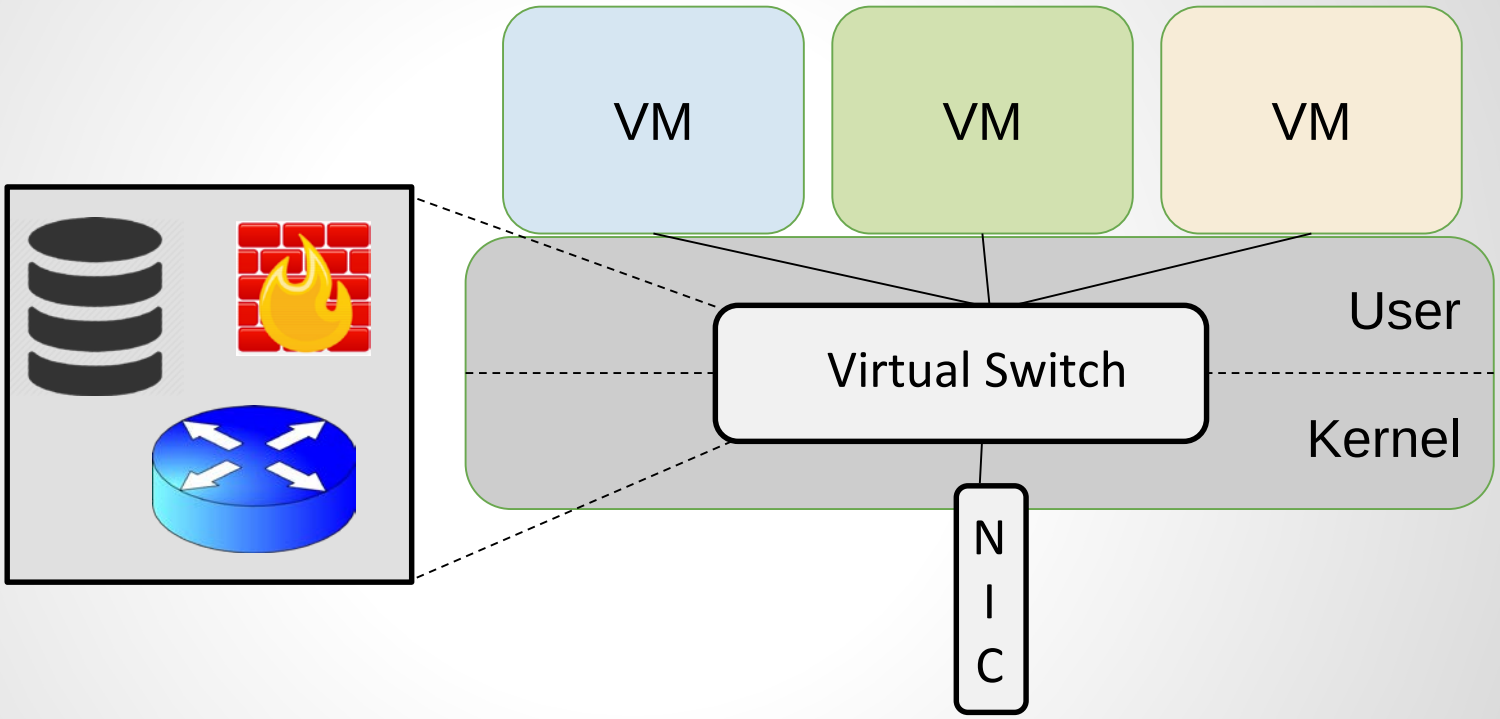
Virtual switches reside in the **server's virtualization layer** (e.g., Xen's Dom0). Goal: provide connectivity and isolation.

Increasing Complexity:
 # Parsed Protocols

Number of parsed high-level protocols constantly increases:



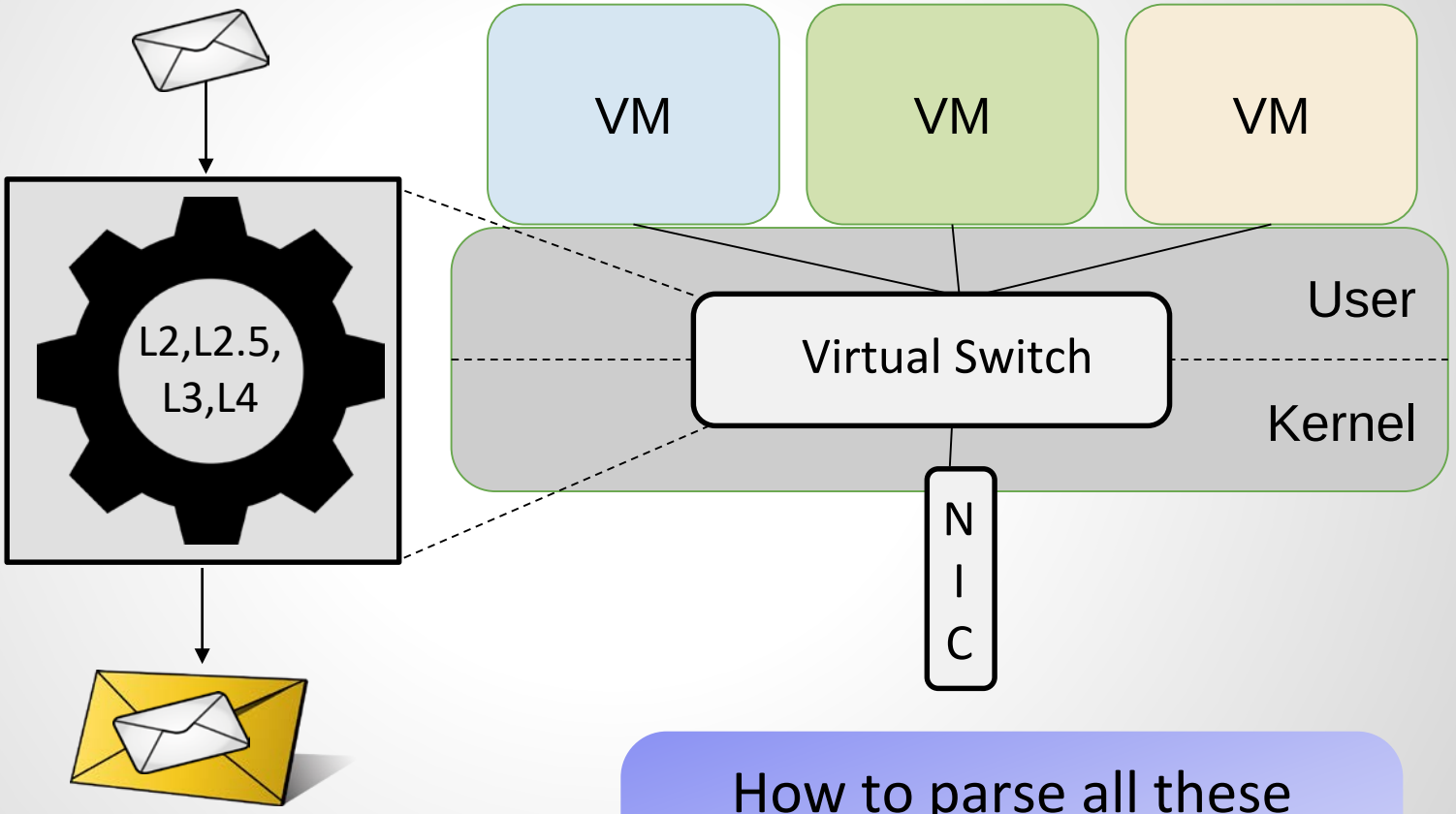
Increasing Complexity:
Introduction of middlebox functionality



Increasing workloads and advancements in network virtualization drive virtual switches to **implement middlebox functions** such as load-balancing, DPI, firewalls, etc.

Increasing Complexity:
Unified Packet Parsing

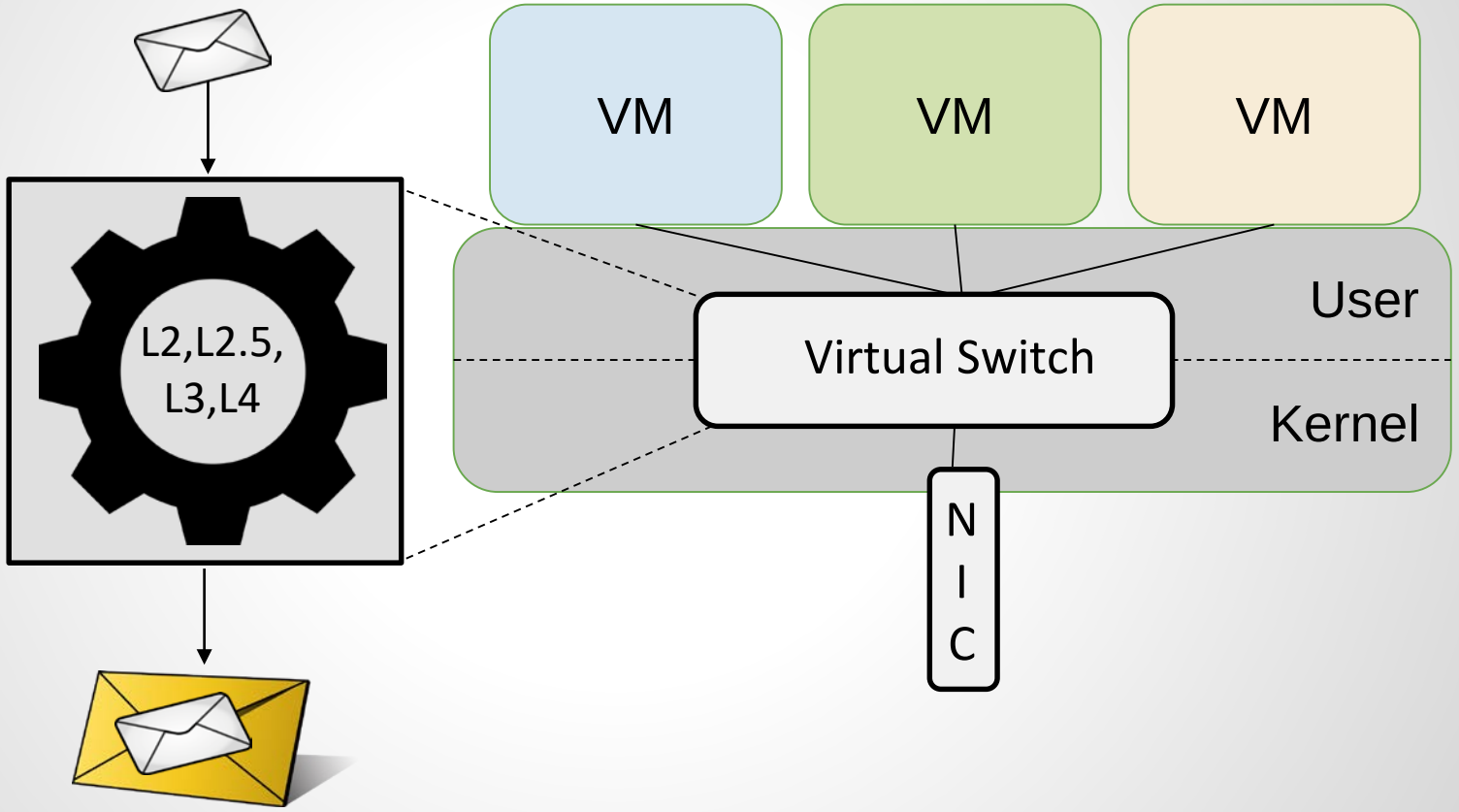
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



How to parse all these protocols without lowering forwarding performance?!

Increasing Complexity: *Unified Packet Parsing*

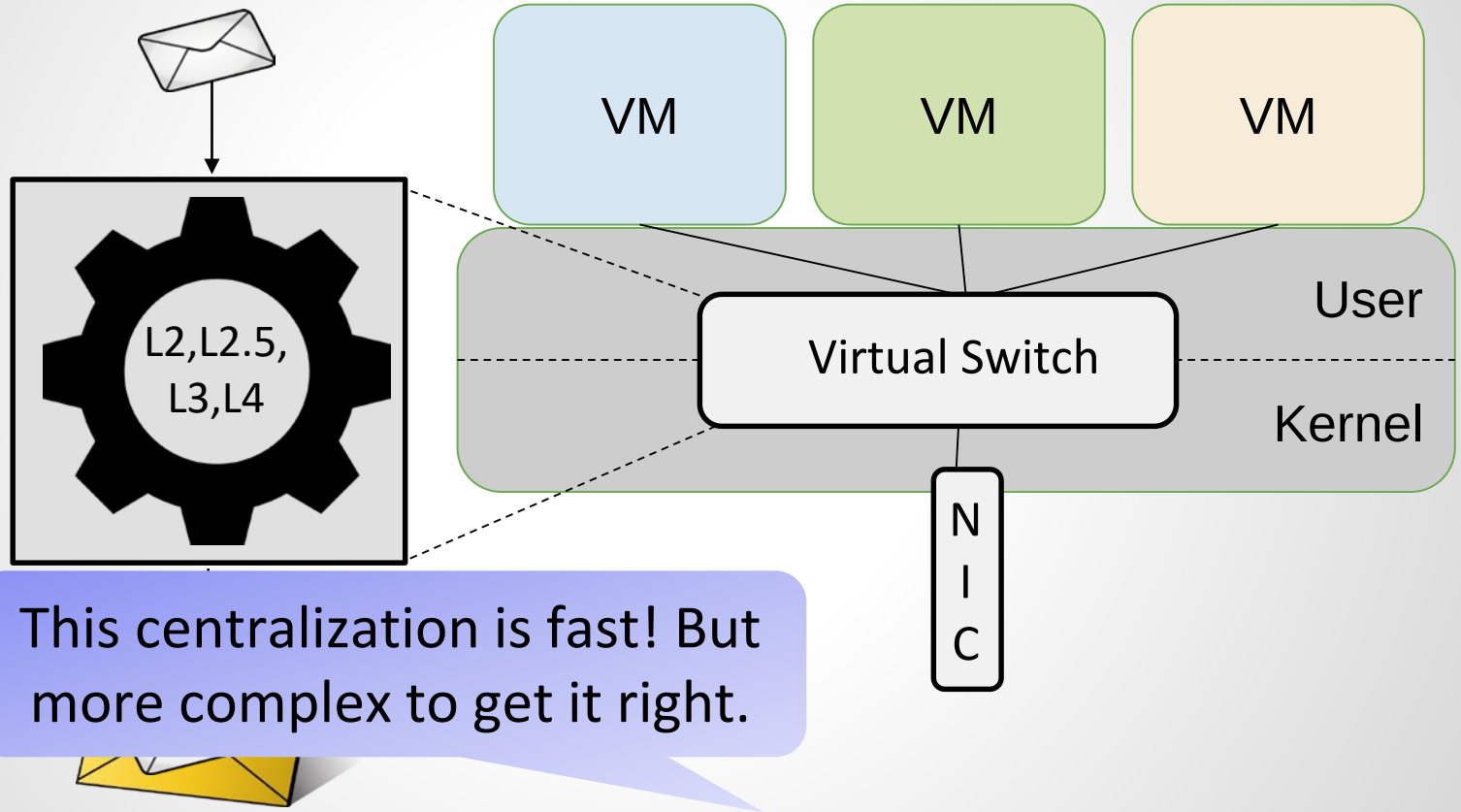
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



Unified packet parsing allows parse more and more protocols efficiently: in a single pass!

Increasing Complexity: *Unified Packet Parsing*

Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



Unified packet parsing allows parse more and more protocols efficiently: in a single pass!

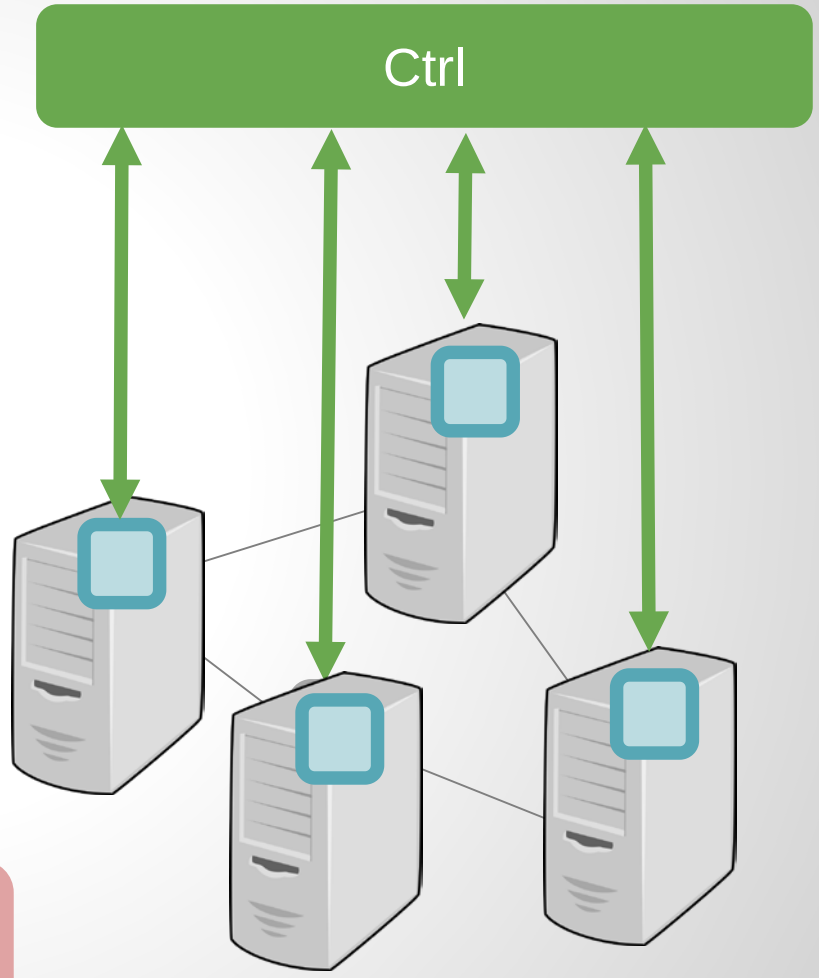
Complexity: The Enemy of Security!

- Data plane security not well-explored (in general, not only virtualized): most security research on control plane

- Two conjectures:

1. Virtual switches increase the attack surface.

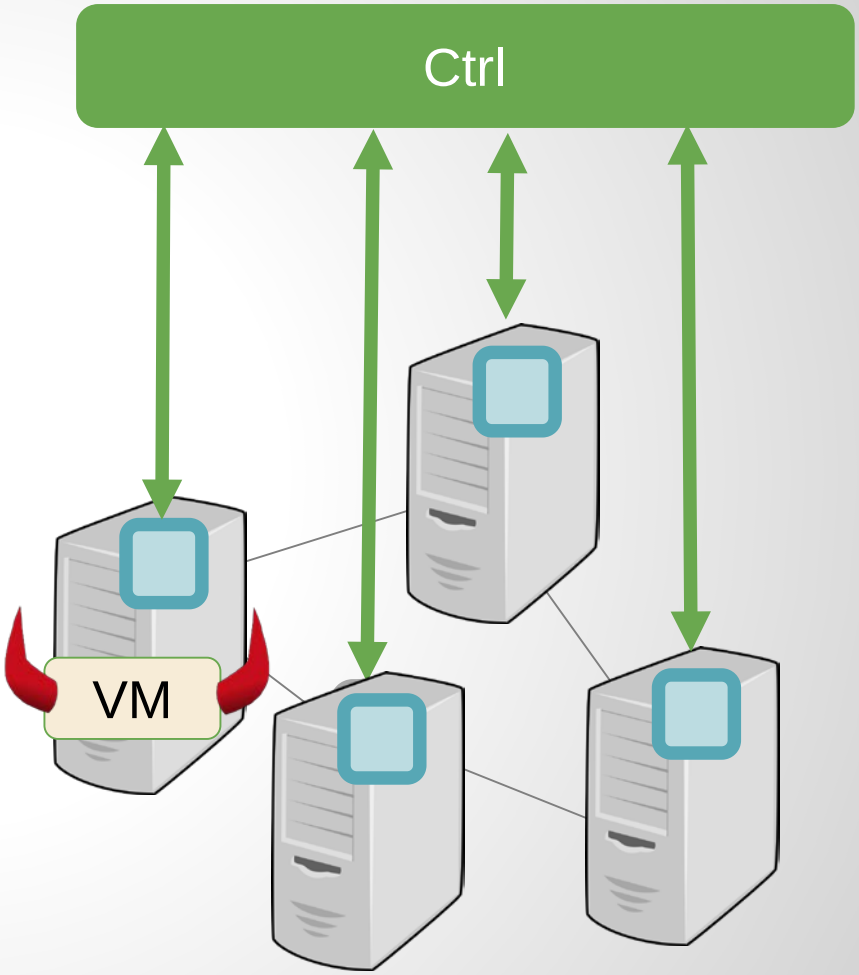
2. Impact of attack larger than with traditional data planes.



The Attack Surface: Closer…

Attack surface becomes closer:

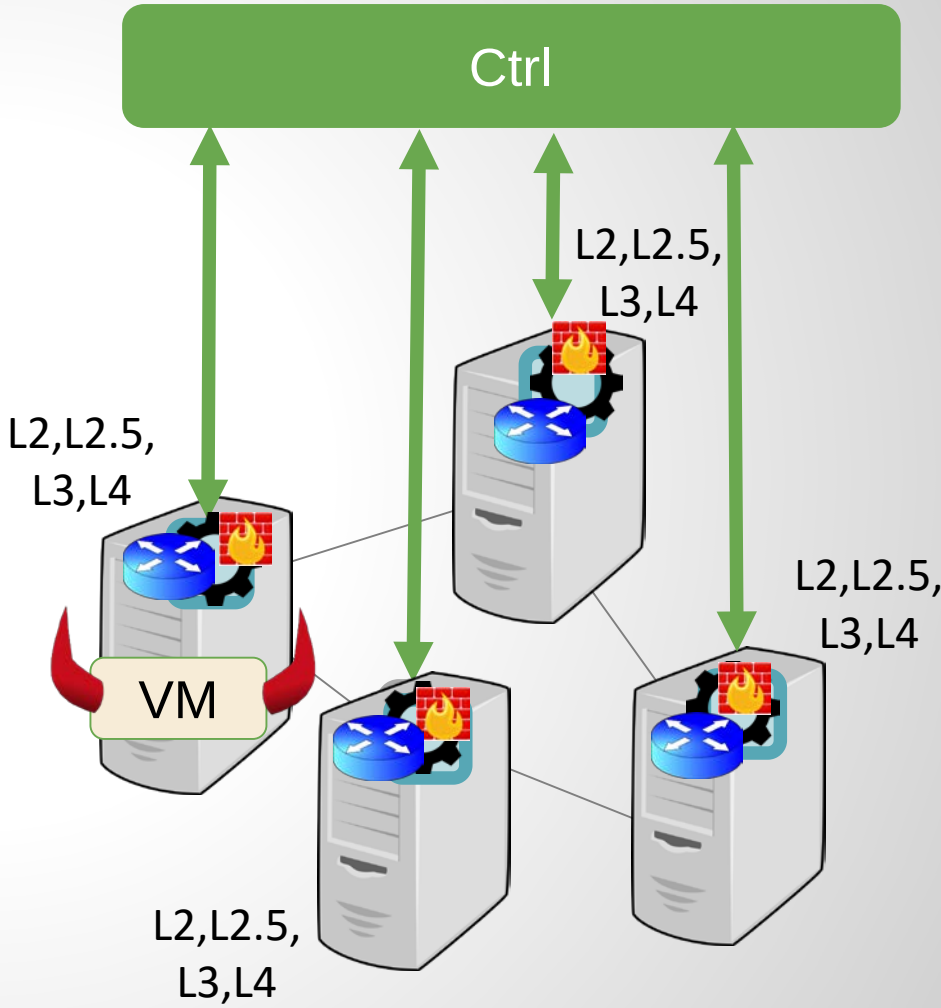
- Packet parser typically integrated into the code base of virtual switch
- First component of the virtual switch to process network packets it receives from the network interface
- May process attacker-controlled packets!



The Attack Surface: … More Complex …

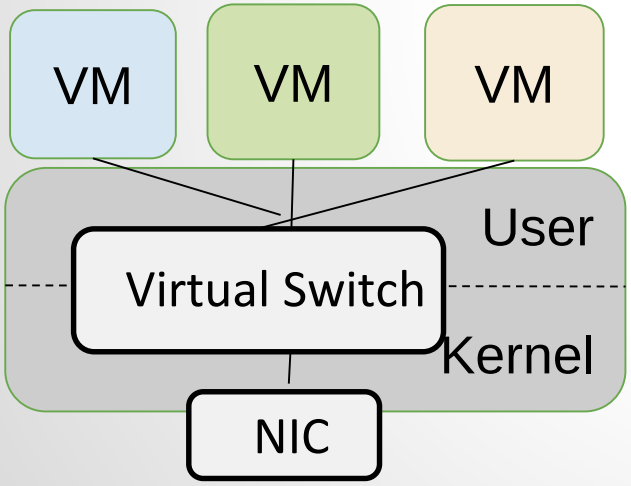
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN

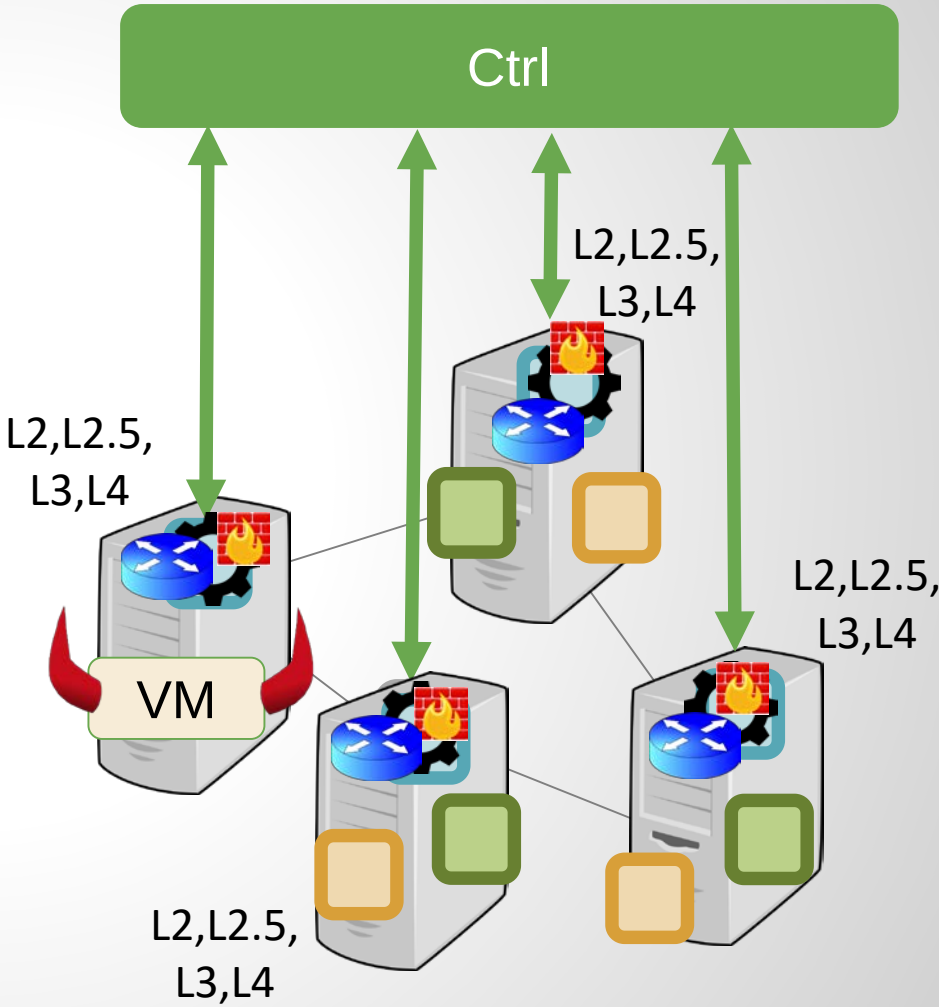
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP



… Elevated Priviledges and Collocation …

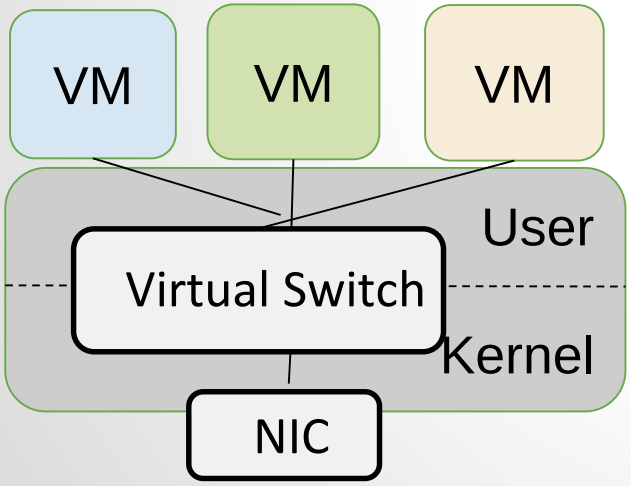
- Collocated (at least partially) with hypervisor's Dom0 kernel space, guest VMs, image management, block storage, identity management, …

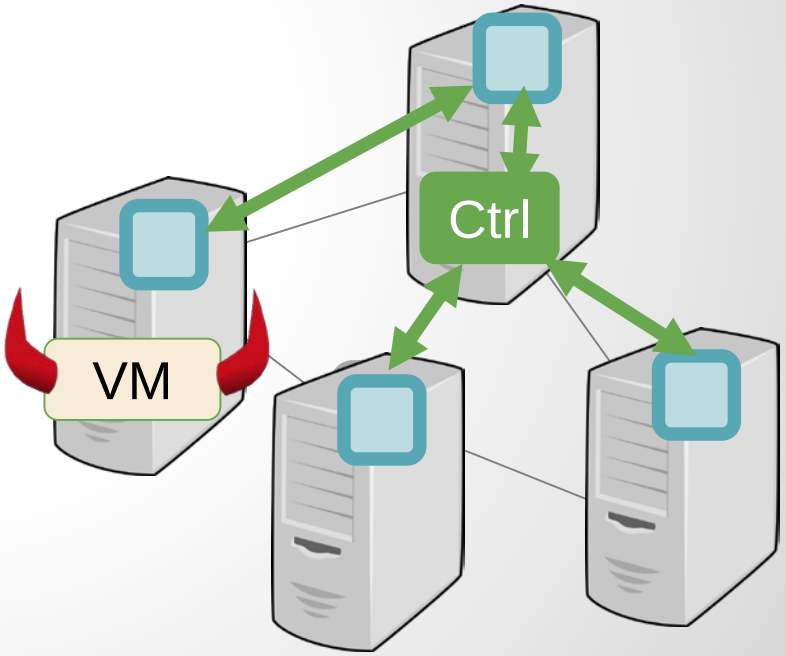




… Elevated Priviledges and Collocation …

- Collocated (at least partially) with hypervisor's Dom0 kernel space, guest VMs, image management, block storage, identity management, …



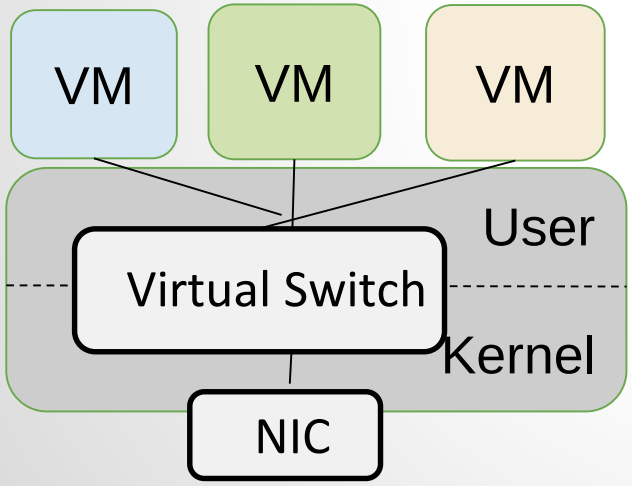


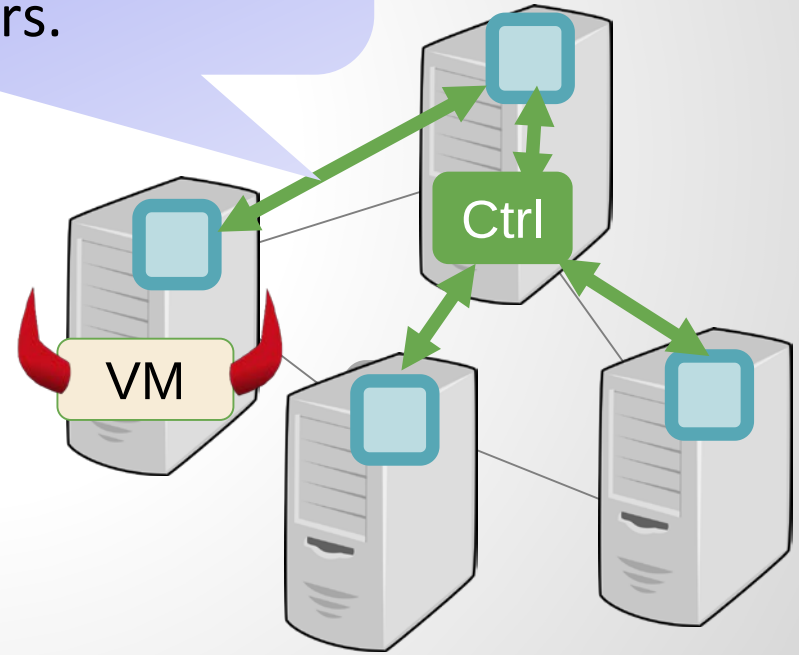
- … the controller itself.

… Centralization …

- Collocate
with hyp
space), g
managen
identity m

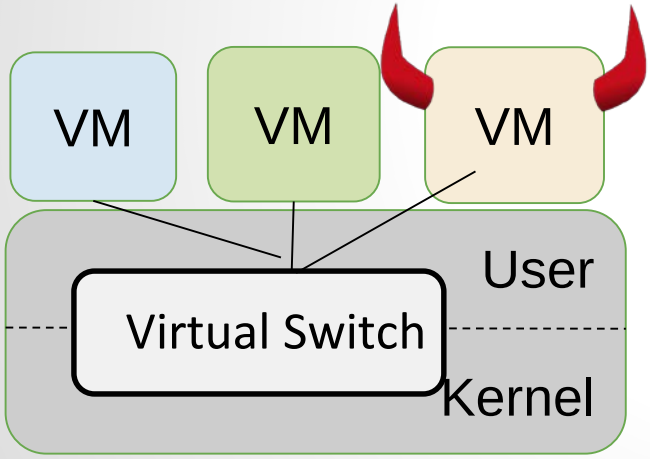
Available communication channels to (SDN/Openstack) controller! Controller needs to be reachable from all servers.





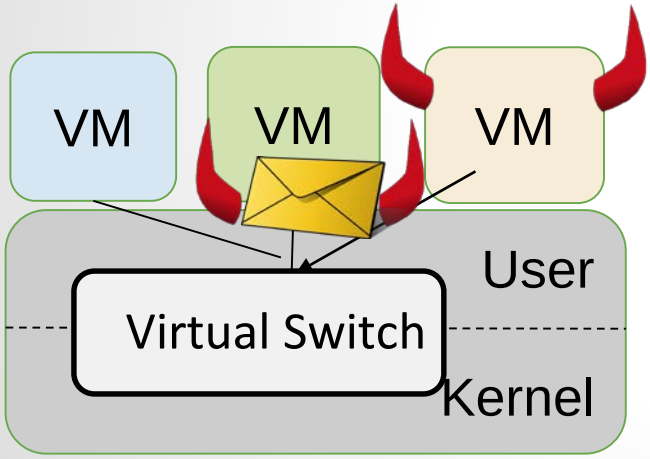
- … the controller itself.

Larger Impact: Case Study OVS



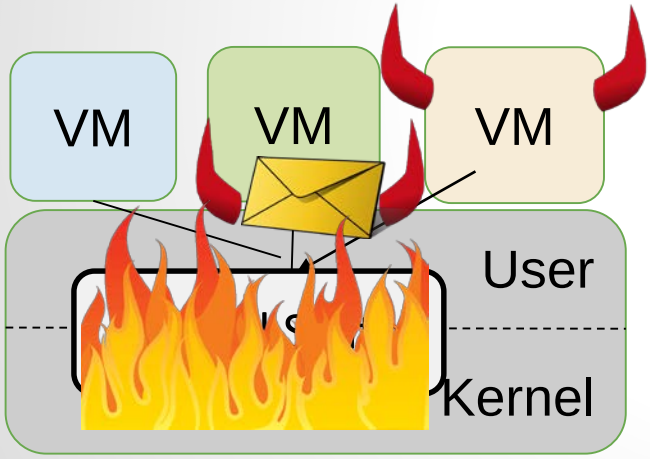
1. Rent a VM in the cloud (**cheap**)

Larger Impact: Case Study OVS



2. Send **malformed MPLS packet** to virtual switch (**unified parser** parses label stack packet beyond the threshold)

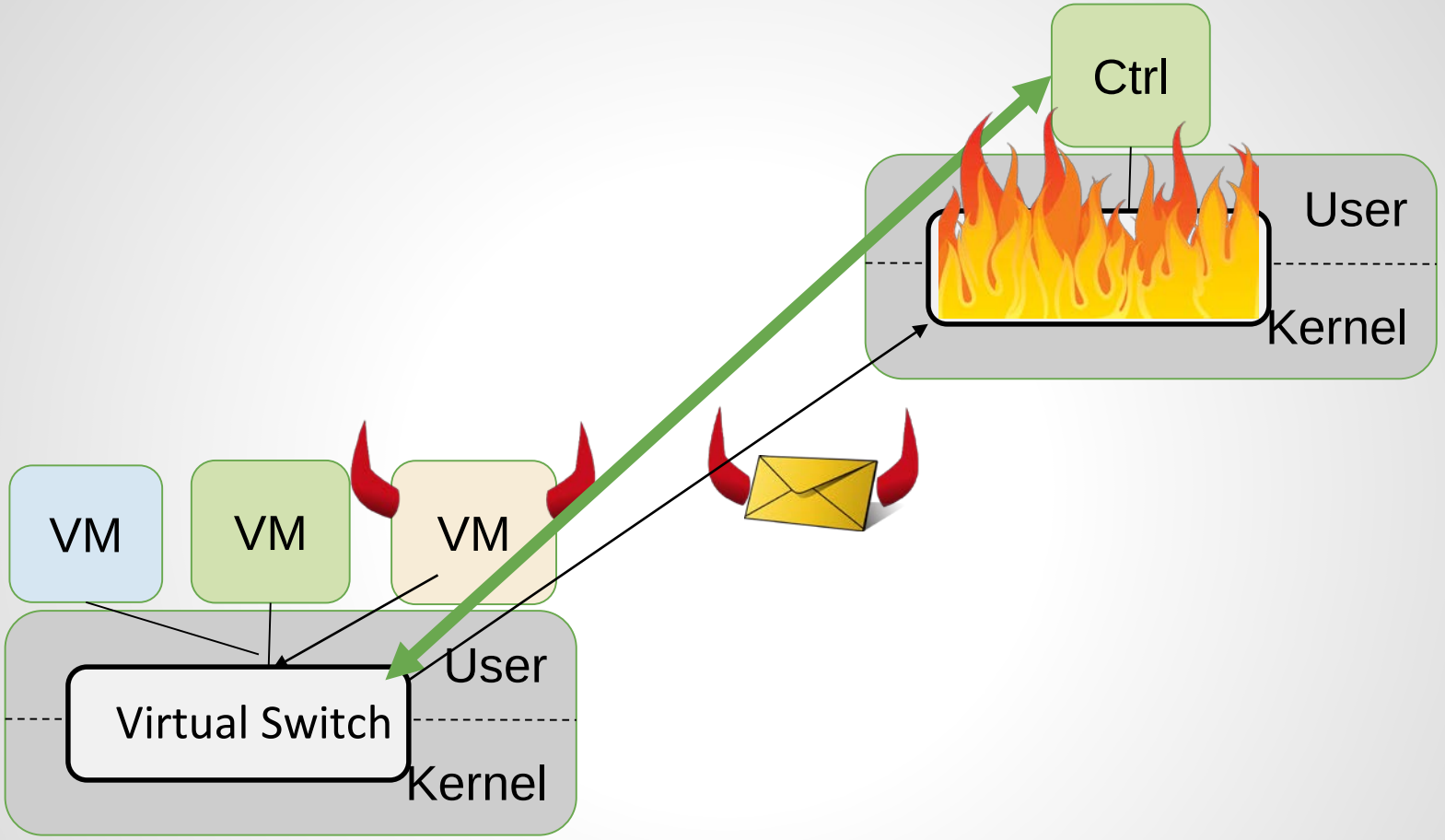
Larger Impact: Case Study OVS



3. **Stack buffer overflow** in (unified) MPLS parsing code:

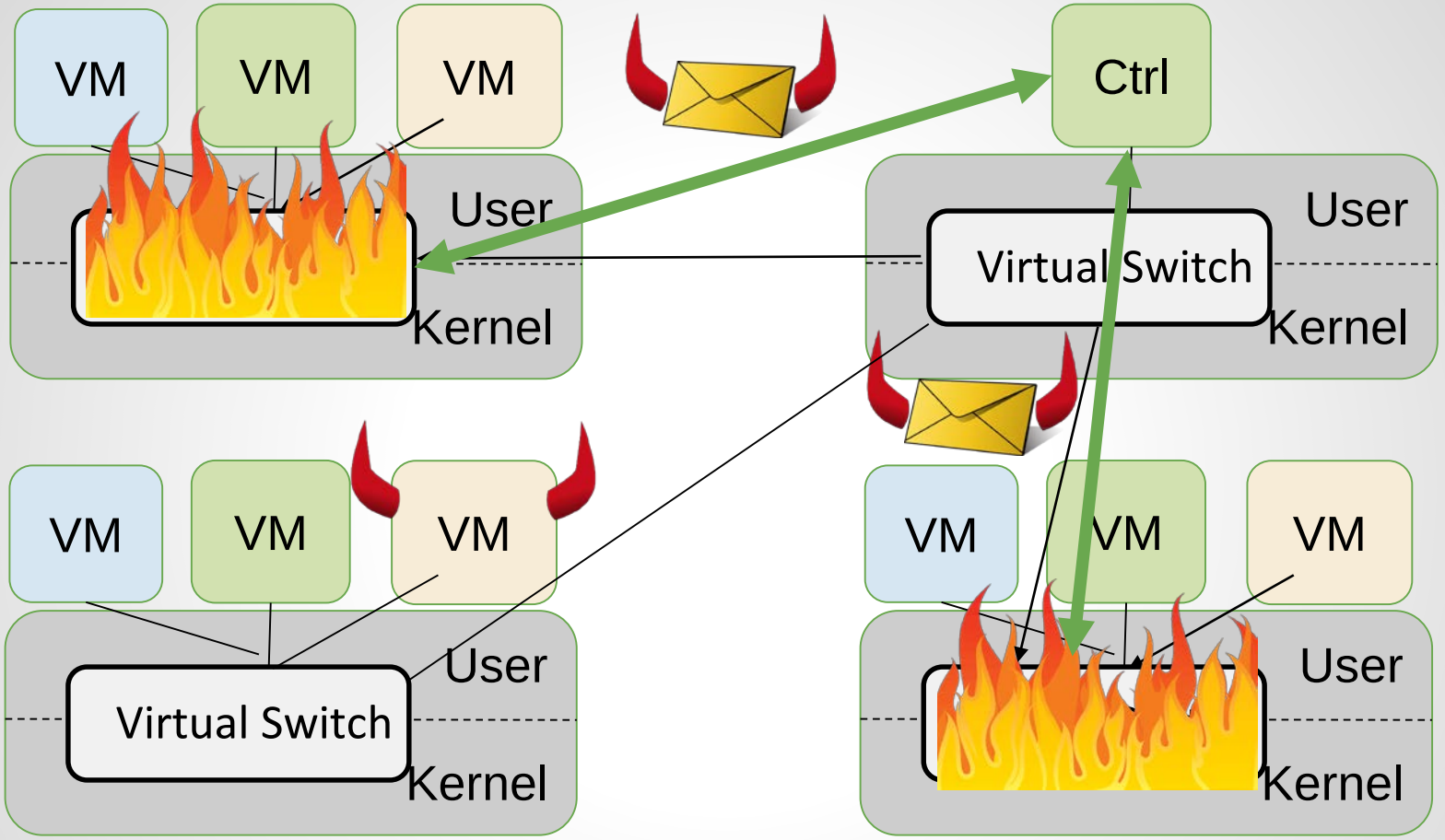
enables remote code execution

Larger Impact: Case Study OVS



4. Send malformed packet to server (virtual switch) where controller is located (use existing communication channel)

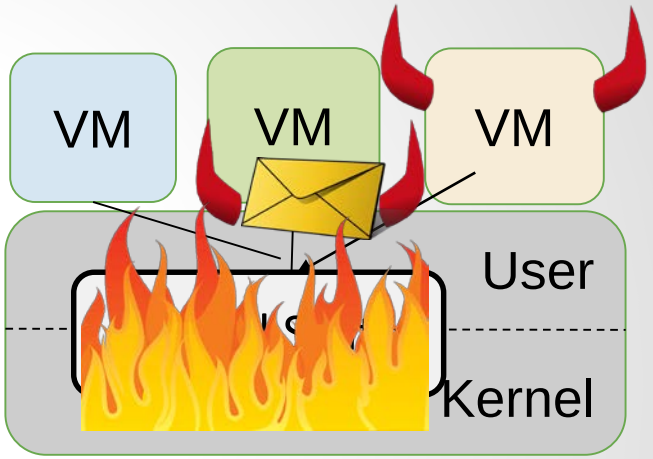
Larger Impact: Case Study OVS



5. Spread

A New Threat Model

- Limited skills required
  - Use standard fuzzer to find crashes
  - Construct malformed packet
  - Build ROP chain
- Limited resources
  - rent a VM in the cloud
- No physical access needed



No need to be a state-level attacker to compromise the dataplane (and beyond)!

Similar problems in NFV: need even more complex parsing/processing. And are often built on top of OvS.

Conclusion

- Programmability and virtualization: opportunities but also challenges
  - E.g.,: faster innovation, flexibilities in resource allocation, etc.
  - But, e.g.: performance isolation needs to be ensured across all involved resources, resulting resource allocation problems hard
- Algorithmic techniques for flexible resource allocation: from waypoint routing over virtual network embedding to online admission control
- Vision of demand-aware and self-adjusting networks: depends on the structure of demand!
- Security: even more opportunities and challenges ☺

References

Predictable performance

On The Impact of the Network Hypervisor on Virtual Network Performance
Andreas Blenk, Arsany Basta, Wolfgang Kellerer, and Stefan Schmid.
IFIP Networking, Warsaw, Poland, May 2019.

Waypoint Routing

Waypoint Routing in Special Networks
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, Mahmoud Parham, and Stefan Schmid.
IFIP Networking, Zurich, Switzerland, May 2018.

Walking Through Waypoints
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid.
13th Latin American Theoretical Informatics Symposium (**LATIN**), Buenos Aires, Argentina, April 2018.

Charting the Algorithmic Complexity of Waypoint Routing
Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), 2018.

Virtual Network Embedding

Virtual Network Embedding Approximations: Leveraging Randomized Rounding
Matthias Rost and Stefan Schmid.
IEEE/ACM Transactions on Networking (**TON**), 2019.

Parametrized Complexity of Virtual Network Embeddings: Dynamic & Linear Programming Approximations
Matthias Rost, Elias Döhne, and Stefan Schmid.
ACM SIGCOMM Computer Communication Review (**CCR**), January 2019.

Charting the Complexity Landscape of Virtual Network Embeddings (Best Paper Award)
Matthias Rost and Stefan Schmid.
IFIP Networking, Zurich, Switzerland, May 2018.

Fast Failover

Competitive and Deterministic Embeddings of Virtual Networks
Guy Even, Moti Medina, Gregor Schaffrath, and Stefan Schmid.
Journal Theoretical Computer Science (**TCS**), Elsevier, 2013.

Improved Fast Rerouting Using Postprocessing
Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
38th International Symposium on Reliable Distributed Systems (**SRDS**), Lyon, France, October 2019.

Security of Data Plane

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
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.