Participatory Networking:
An API for Application Control of SDNs

Andrew Ferguson, Arjun Guha, Chen Liang,
Rodrigo Fonseca, and Shriram Krishnamurthi

BROWN Cornell

Participatory Networking

Participatory Networking integrates end-users and their applications directly into the management of the network.

1. SSHGuard
2. Ekiga
3. ZooKeeper
4. Hadoop

Motivation

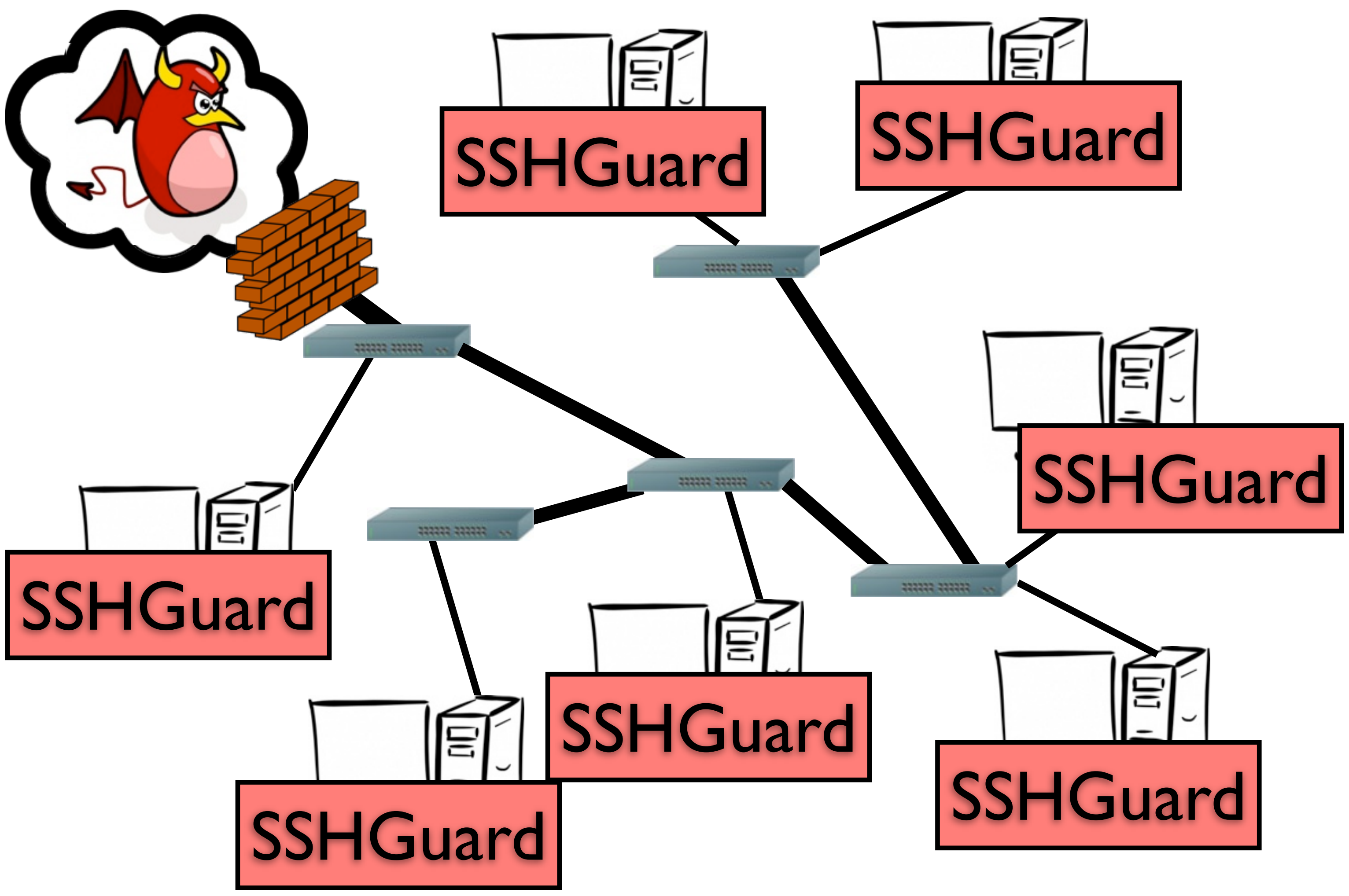
As a motivation, let's consider four applications which might like to manage the network

1. SSHGuard
2. Ekiga
3. ZooKeeper
4. Hadoop

blocks hosts in response to login attempts

uses knowledge from host OS

prefers to deny traffic close to source



today: block bad traffic at end host
“if it could…”

1. SSHGuard

2. Ekiga

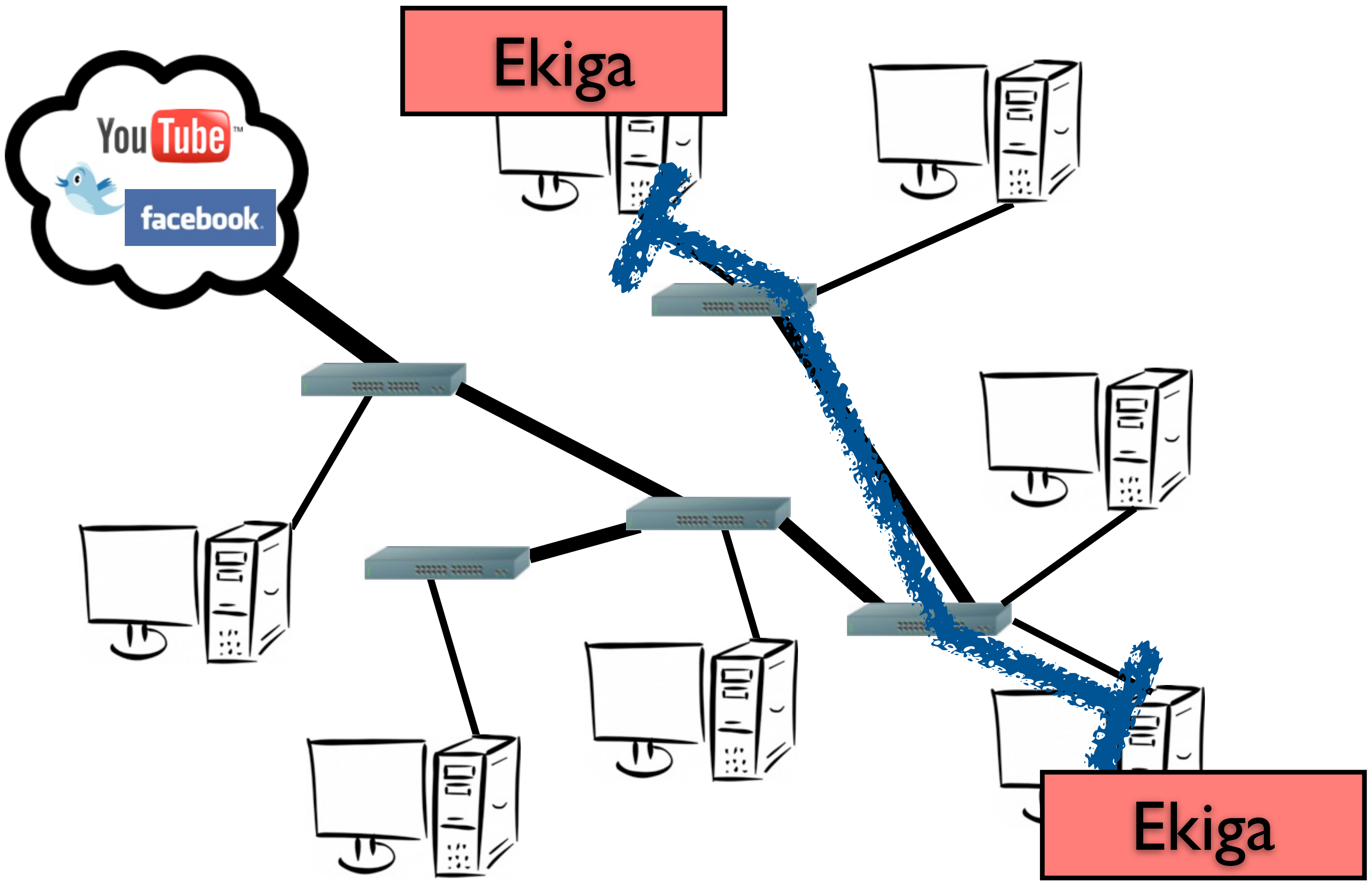
3. ZooKeeper

4. Hadoop

open source VOIP client

network needs dictated by end-user

prefers to reserve bandwidth



Explain Ekiga's traffic pattern

"if it could…"

1. SSHGuard

2. Ekiga

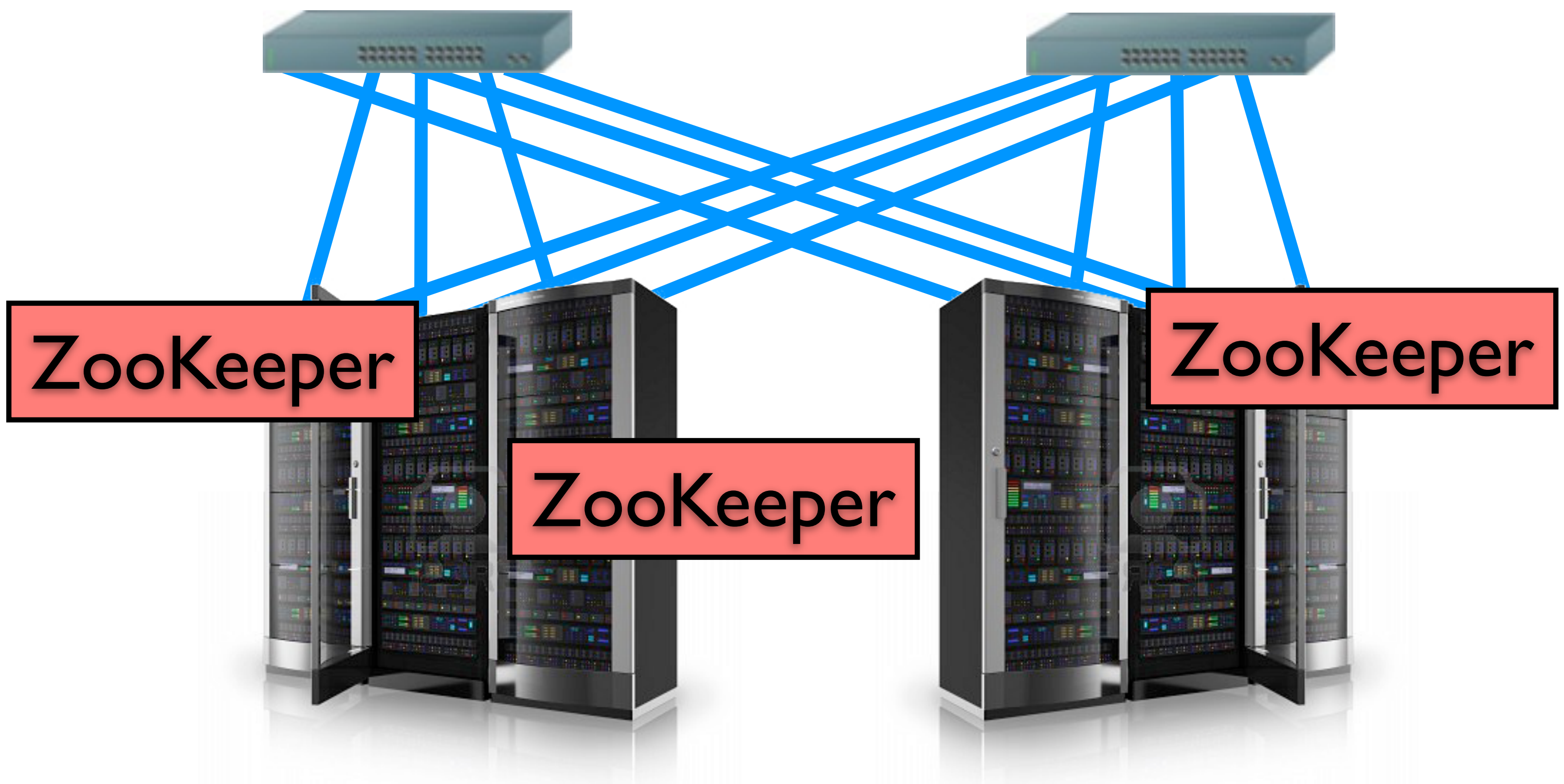
3. ZooKeeper

4. Hadoop

Paxos-like coordination service

network needs dictated by placement

prefers high-priority switch queues



Explain ZooKeeper's traffic pattern … "control-traffic"

"if it could…"

1. SSHGuard

2. Ekiga

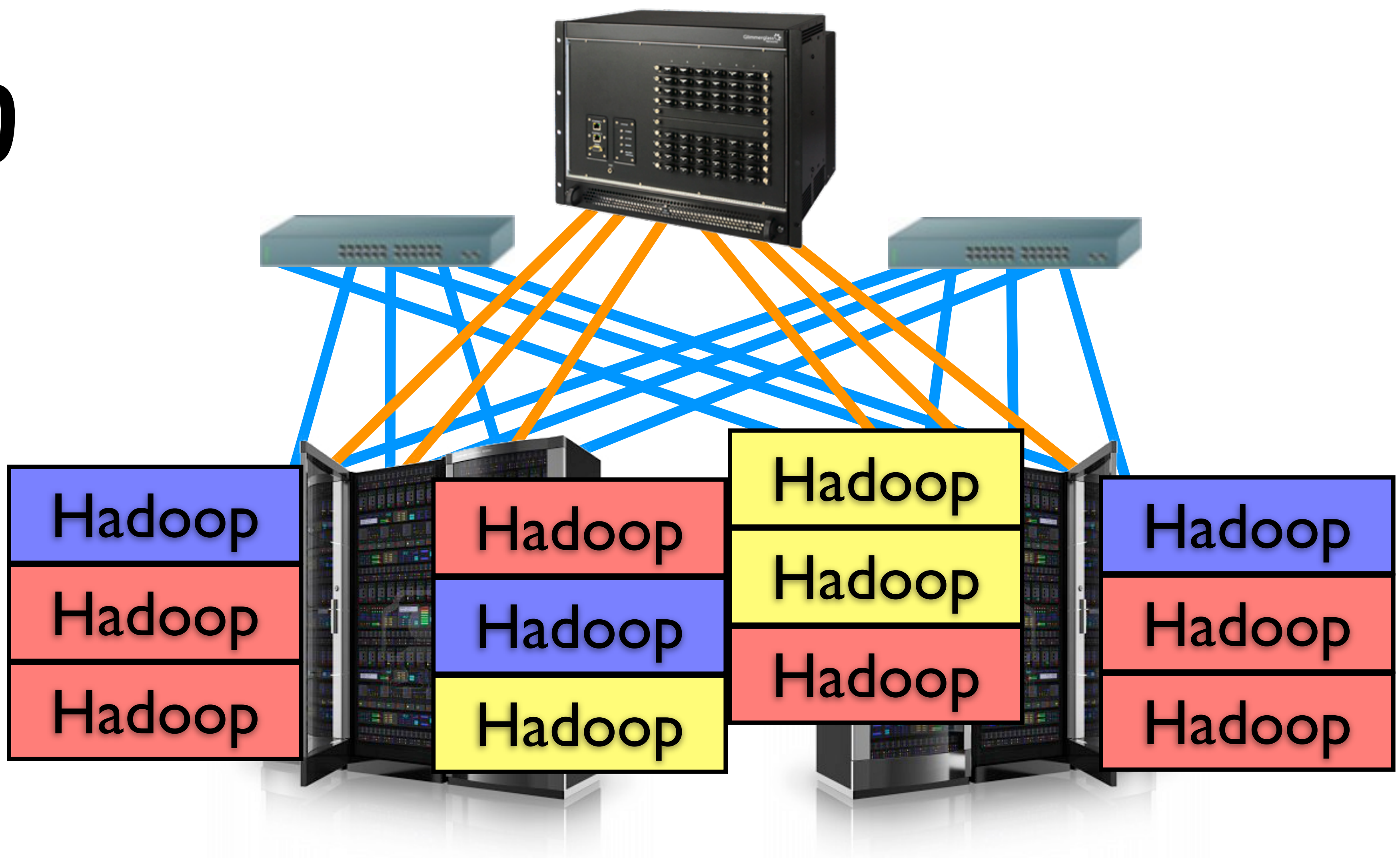
3. ZooKeeper

4. Hadoop

open source data processing platform

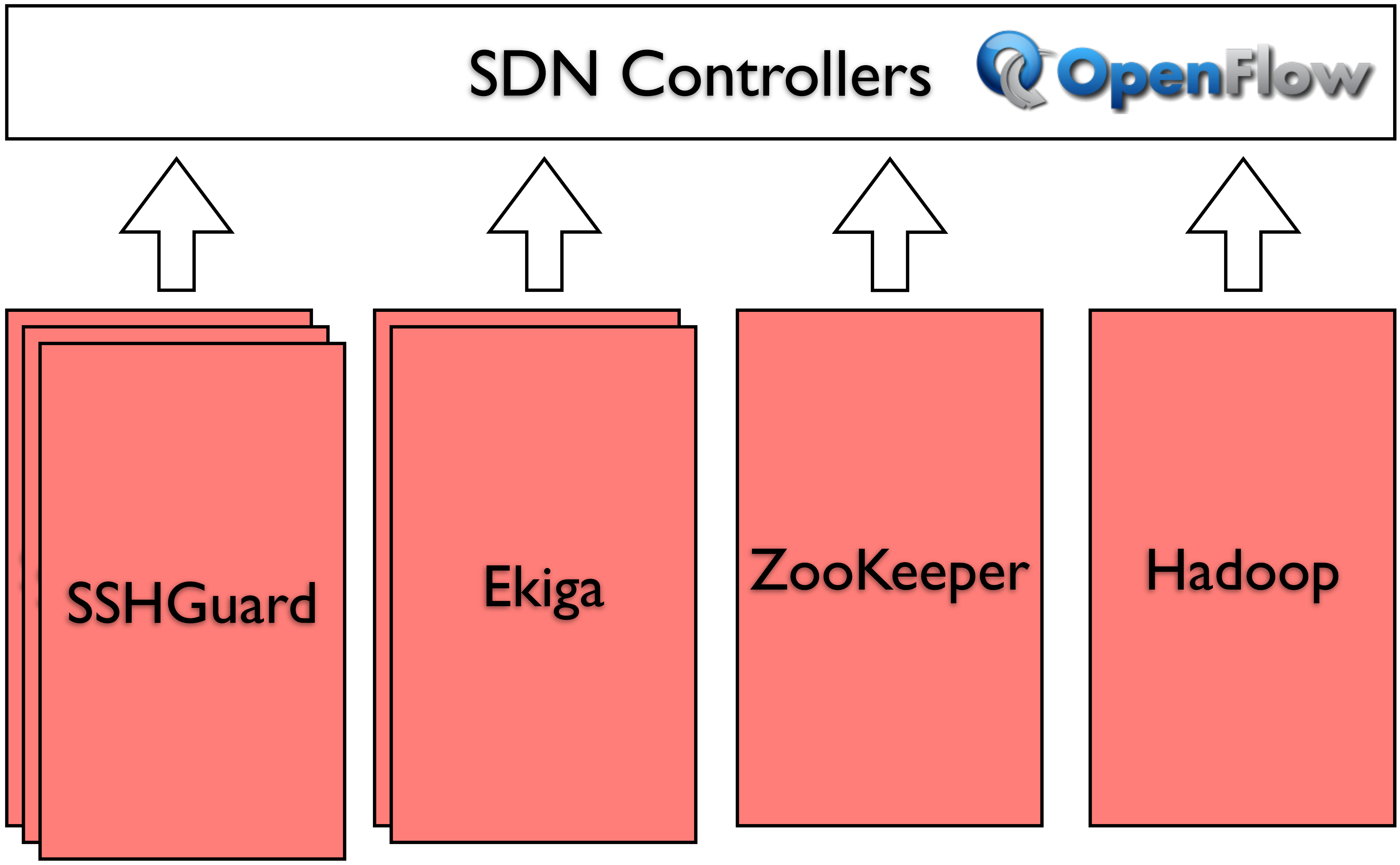
network weights known by scheduler

prefers to reserve bandwidth



weights are used to express the relative priority of jobs. today these weights affect the amount of CPU and memory for the job

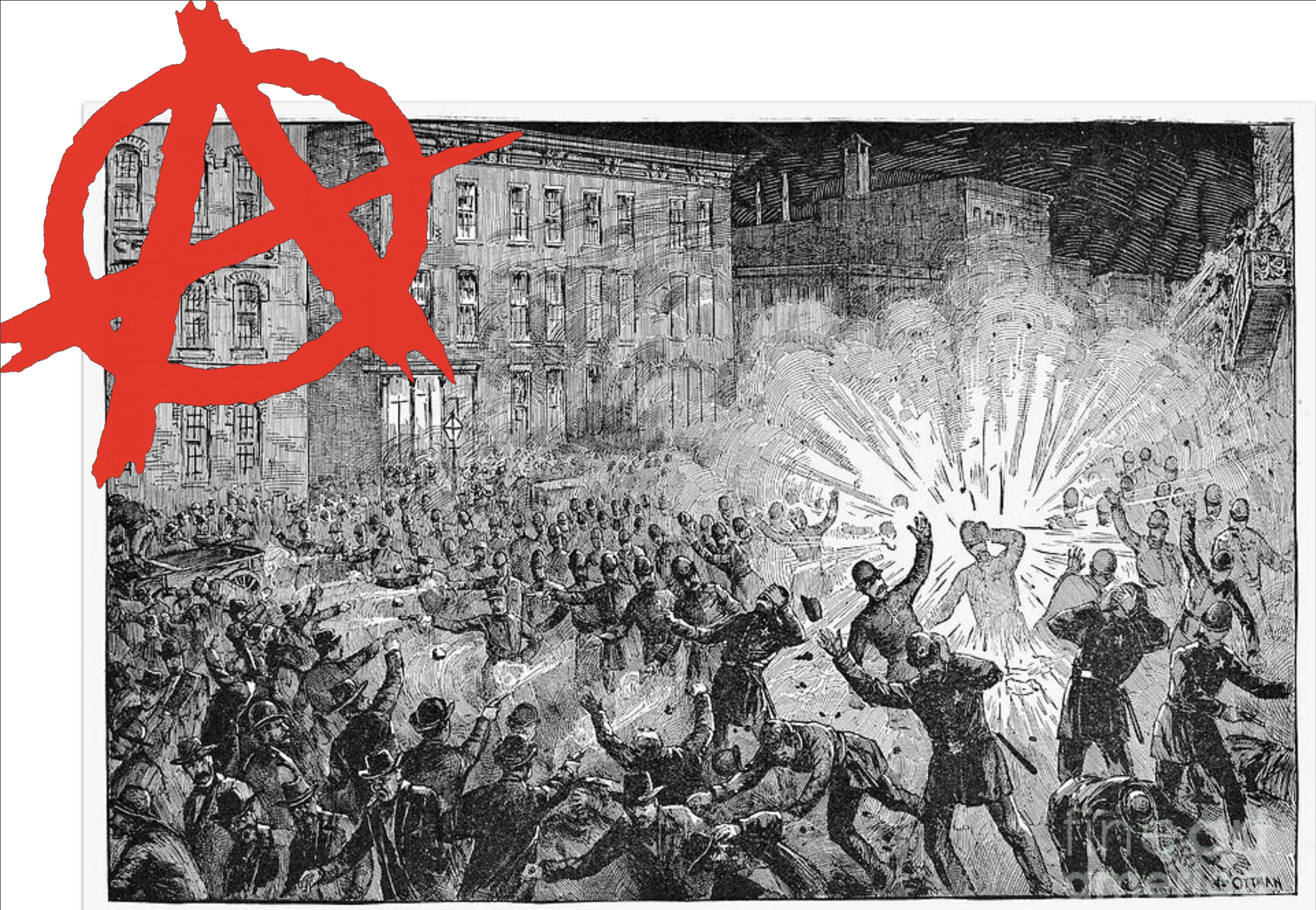
“if it could…”



how could we do this today? file a ticket with the network operators every few minutes as they have frequently changing dynamic needs, precluding a single, static policy

or today, we could program the network by writing an SDN controller for each application.

Combining these controllers would be difficult:
1) have to run as root, and 2) would be affected by the decisions of other controllers



THE HAYMARKET RIOT. The Explosion and the Conflict.

1. decompose control and visibility
2. resolve conflicts between requests

Challenges

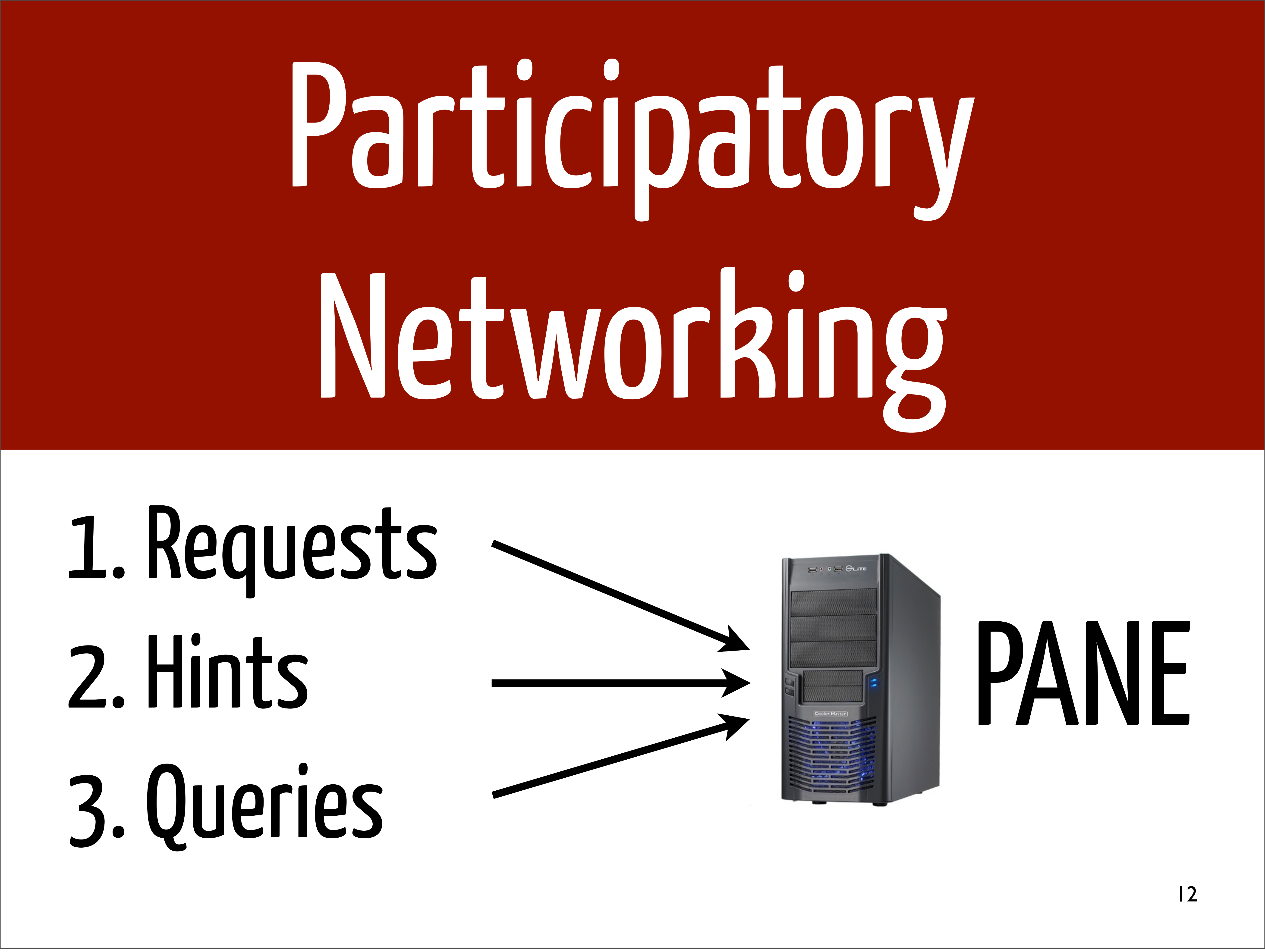
stepping back, we see there are two challenges we need to overcome to prevent this chaos.
(read slide)
or, in other words,
1. how do we keep programs from all running as root?
2. how do we keep programs from being affected by one another?

Participatory Networking

participatory networking is the approach we developed to solve these challenges.

to do so, we need to reason about changes being made to the network. to make such reasoning tractable, we don't allow general purpose programming.

instead, we provide applications with a restricted control-plane API.



In our API, users, their hosts, and their applications send three types of messages to a logically centralized network controller, which we call PANE.

The first are requests for resources, such as guaranteed minimum bandwidth, latency, path properties, or access control. The second are hints about future traffic. And the third are queries for current or future properties of the network.

The PANE controller serves as an arbiter for conflicting proposals, and ultimately performs the requested reconfigurations.

Participatory Networking

- End-user API for SDNs
- Exposes existing mechanisms
- No effect on unmodified applications

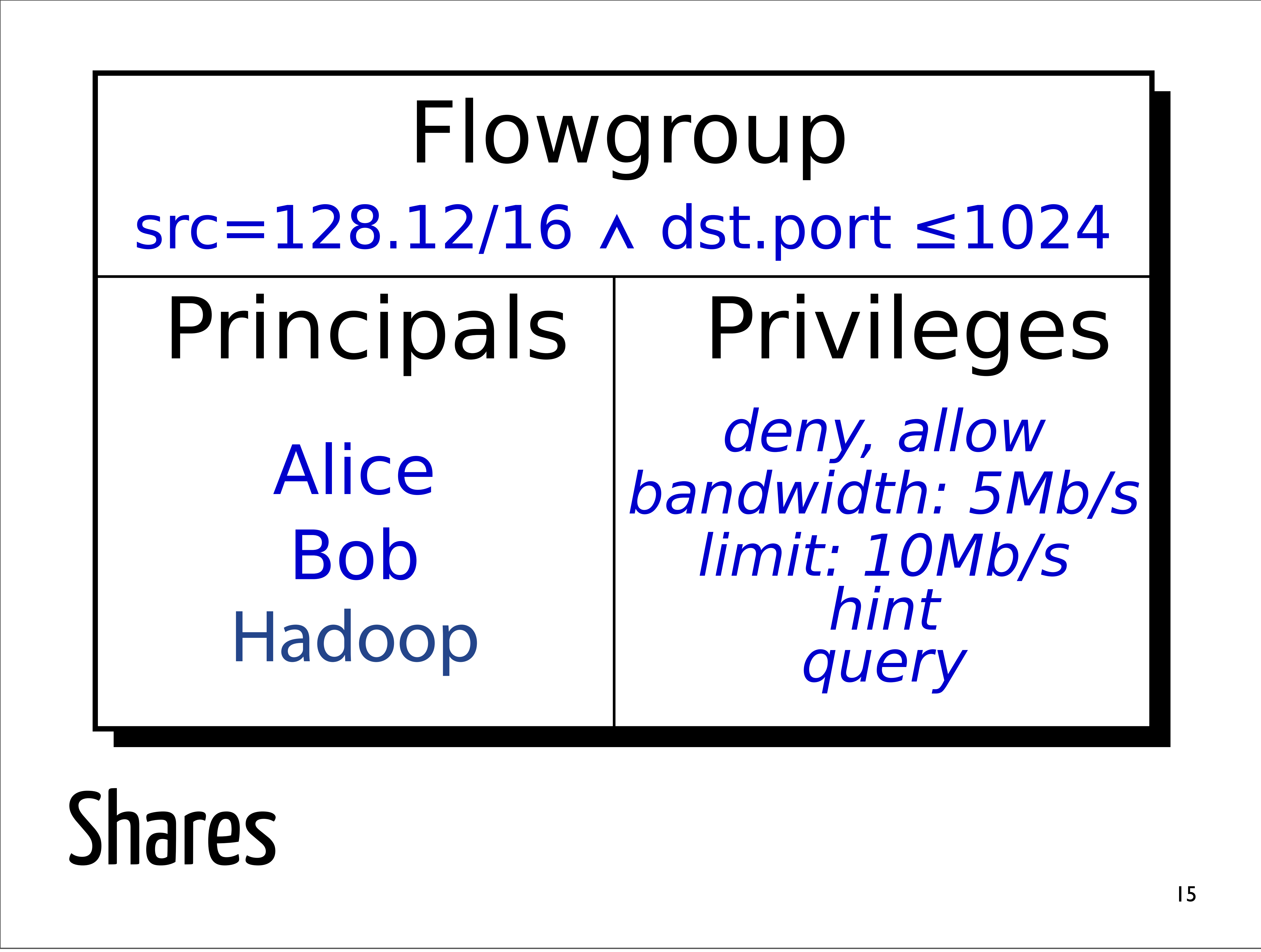
Participatory networking introduces an **end-user API** or **system calls** for software defined networks. It does **not** propose any new mechanisms or network resources such as QoS, routing, or access control -- it simply allows end-users and their applications to use them.

Unmodified applications, or those which choose not to participate, continue to receive the same best-effort performance of existing networks.

In our vision, network operators set baseline policies that enforce fairness and security, while end-users and their applications propose new configurations to meet their needs.

Decomposing Control

Let’s begin with the first challenge: how to decompose control and visibility of the network?



To divide authority, PANE uses a hierarchy of network “shares” which describe WHO can say WHAT about WHICH flows in the network.

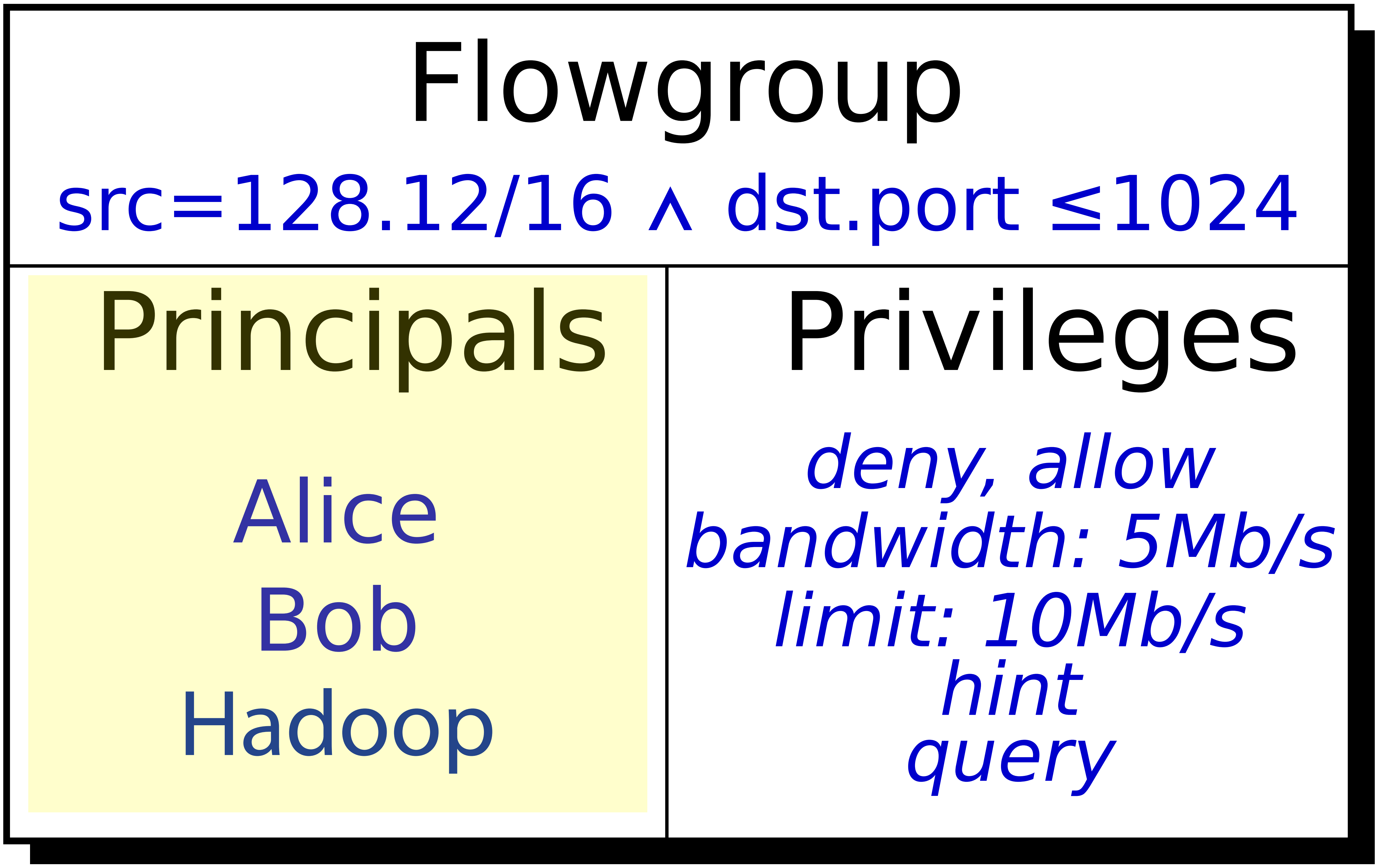
First, each share has a list of principals (click), who are the end users and applications authorized to use the share.

Second, each share refers to a particular flowgroup (click) -- a set of traffic flows identified by standard attributes such as source and destination IP and MAC addresses, protocols, and port numbers.

Finally, they have a list of **privileges** (click) indicating what can be performed using the share. For example, traffic can be allowed or denied, rate-limited, waypointed through a particular switch, or provided with guaranteed minimum bandwidth.

Shares can also authorize end-users to issue hints or make queries about particular traffic flows. These actions can also come with restrictions. For example, bandwidth reservations may be restricted using a token bucket.

(Pause)



Shares

To divide authority, PANE uses a hierarchy of network “shares” which describe WHO can say WHAT about WHICH flows in the network.

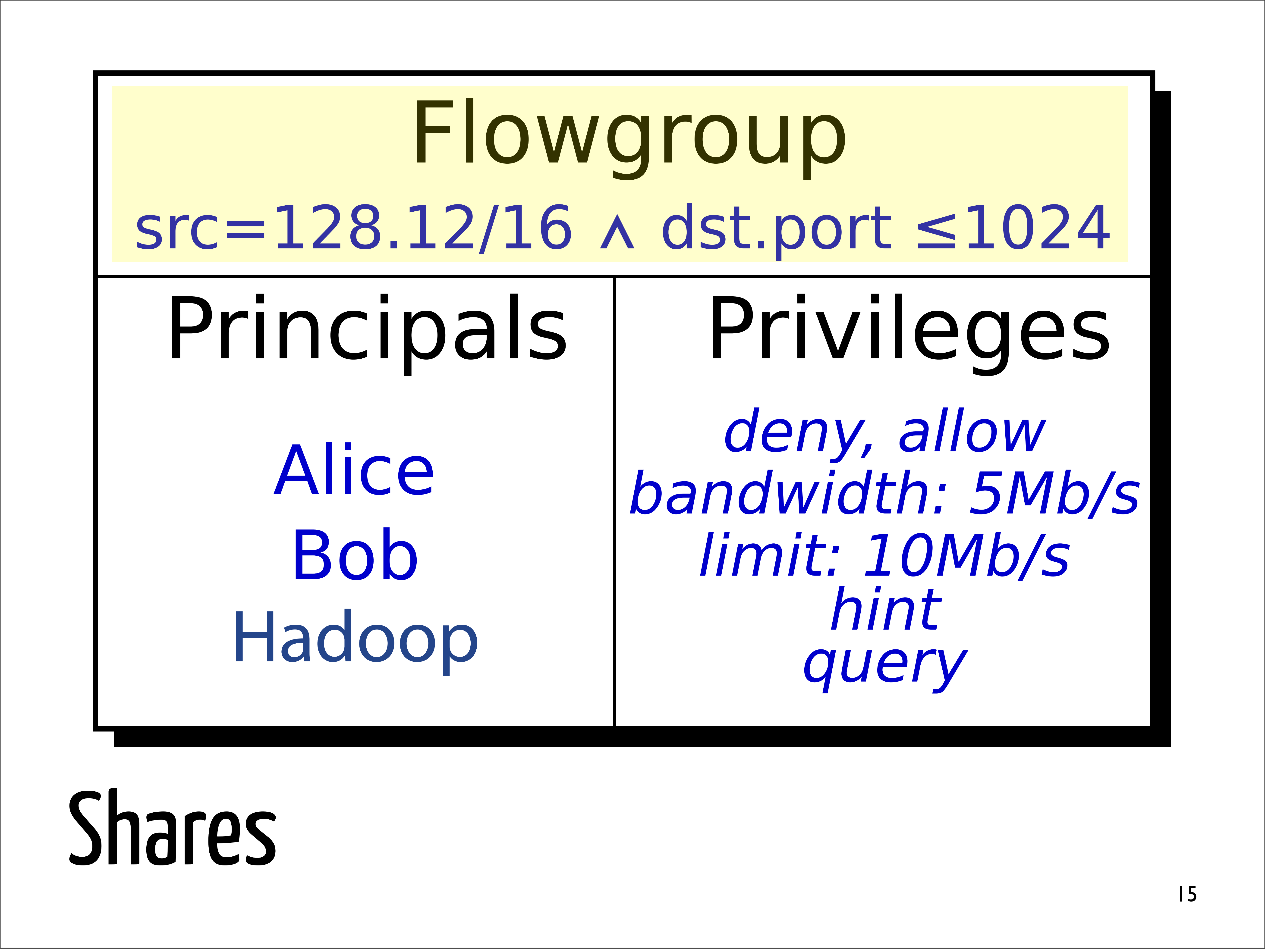
First, each share has a list of principals (click), who are the end users and applications authorized to use the share.

Second, each share refers to a particular flowgroup (click) -- a set of traffic flows identified by standard attributes such as source and destination IP and MAC addresses, protocols, and port numbers.

Finally, they have a list of **privileges** (click) indicating what can be performed using the share. For example, traffic can be allowed or denied, rate-limited, waypointed through a particular switch, or provided with guaranteed minimum bandwidth.

Shares can also authorize end-users to issue hints or make queries about particular traffic flows. These actions can also come with restrictions. For example, bandwidth reservations may be restricted using a token bucket.

(Pause)



To divide authority, PANE uses a hierarchy of network “shares” which describe WHO can say WHAT about WHICH flows in the network.

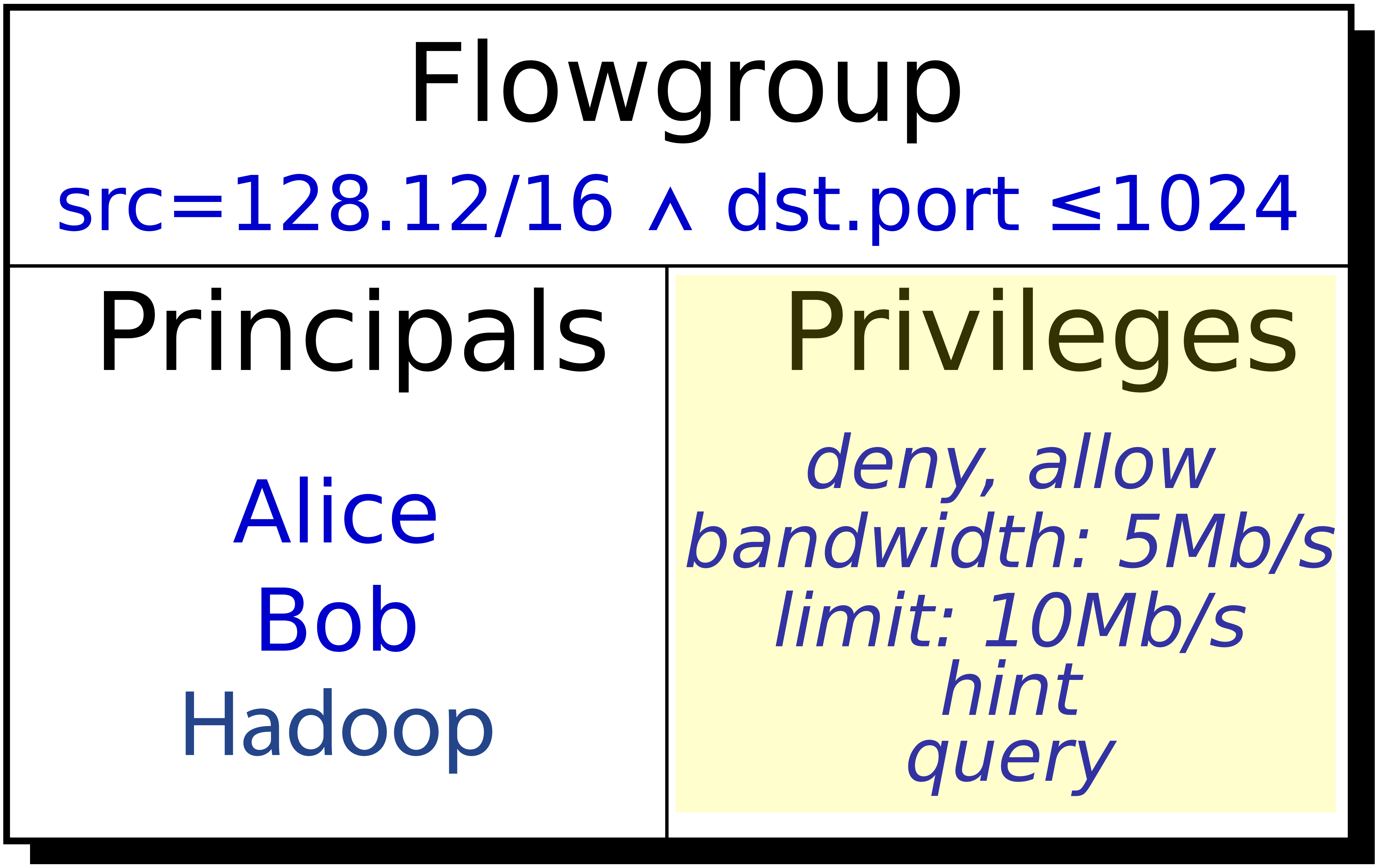
First, each share has a list of principals (click), who are the end users and applications authorized to use the share.

Second, each share refers to a particular flowgroup (click) -- a set of traffic flows identified by standard attributes such as source and destination IP and MAC addresses, protocols, and port numbers.

Finally, they have a list of **privileges** (click) indicating what can be performed using the share. For example, traffic can be allowed or denied, rate-limited, waypointed through a particular switch, or provided with guaranteed minimum bandwidth.

Shares can also authorize end-users to issue hints or make queries about particular traffic flows. These actions can also come with restrictions. For example, bandwidth reservations may be restricted using a token bucket.

(Pause)



Shares

To divide authority, PANE uses a hierarchy of network “shares” which describe WHO can say WHAT about WHICH flows in the network.

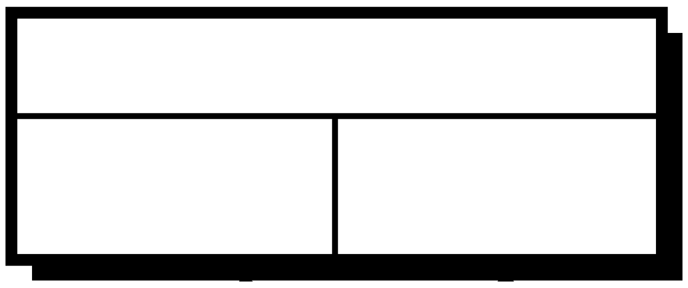
First, each share has a list of principals (click), who are the end users and applications authorized to use the share.

Second, each share refers to a particular flowgroup (click) -- a set of traffic flows identified by standard attributes such as source and destination IP and MAC addresses, protocols, and port numbers.

Finally, they have a list of **privileges** (click) indicating what can be performed using the share. For example, traffic can be allowed or denied, rate-limited, waypointed through a particular switch, or provided with guaranteed minimum bandwidth.

Shares can also authorize end-users to issue hints or make queries about particular traffic flows. These actions can also come with restrictions. For example, bandwidth reservations may be restricted using a token bucket.

(Pause)



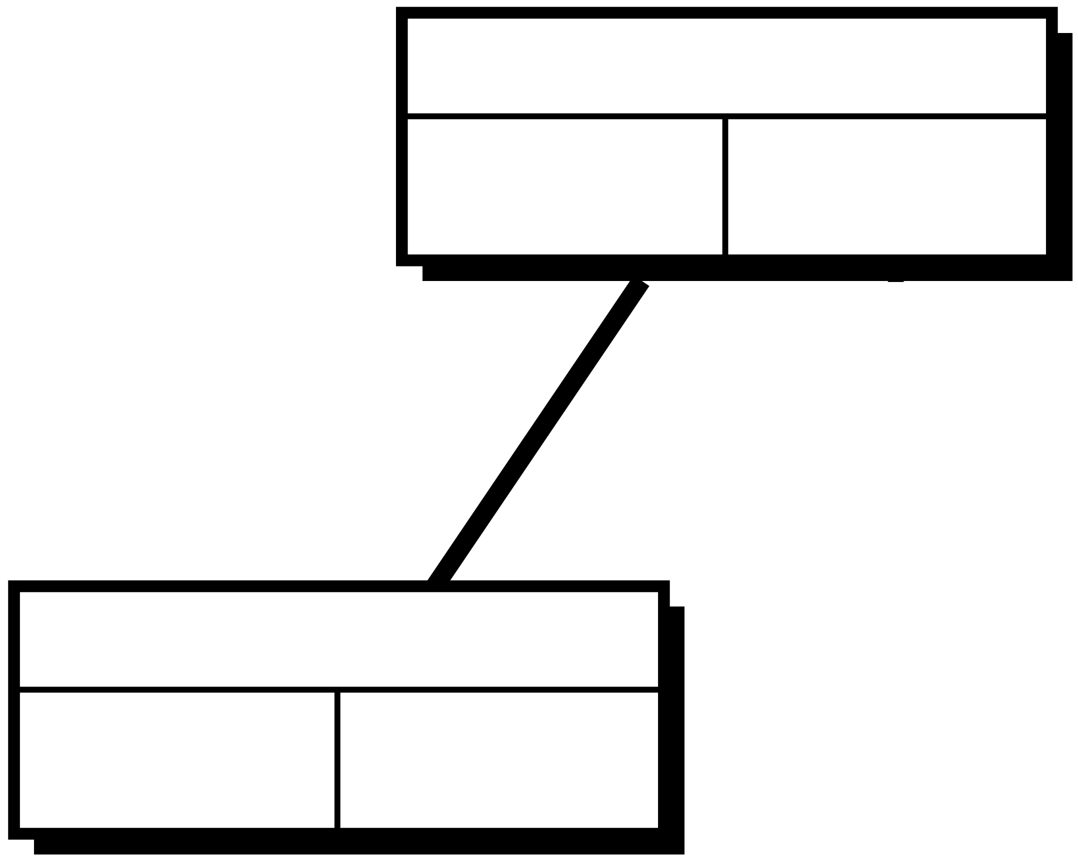
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



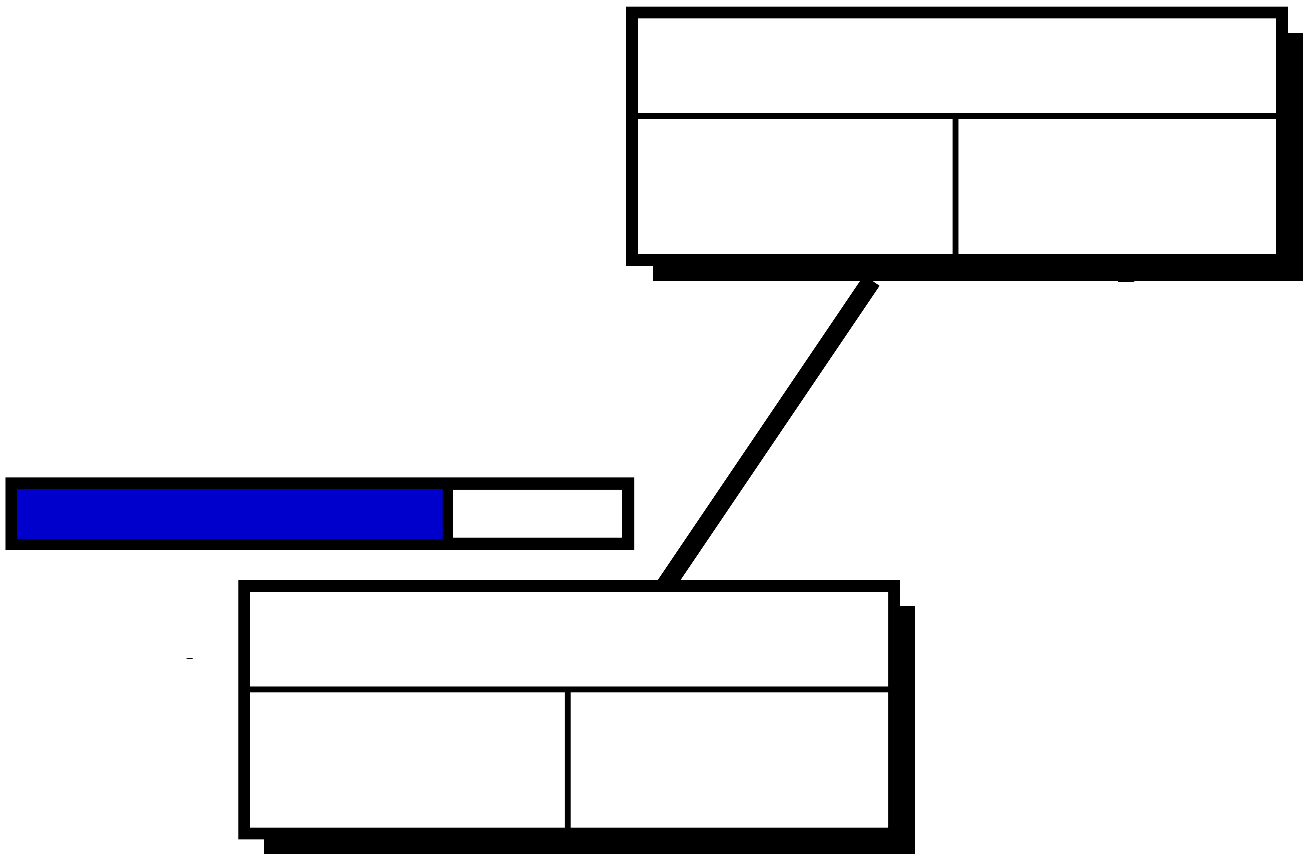
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



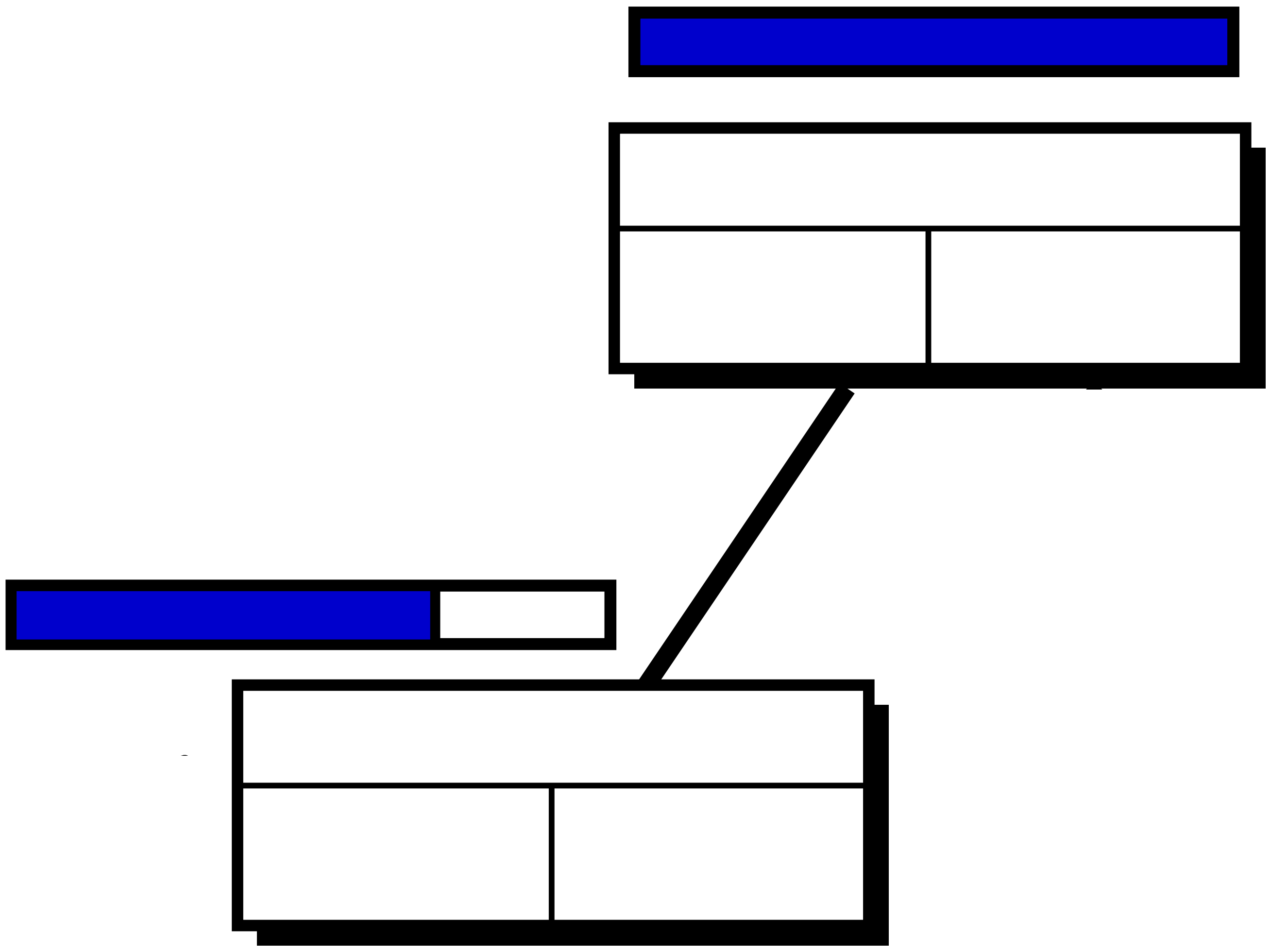
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



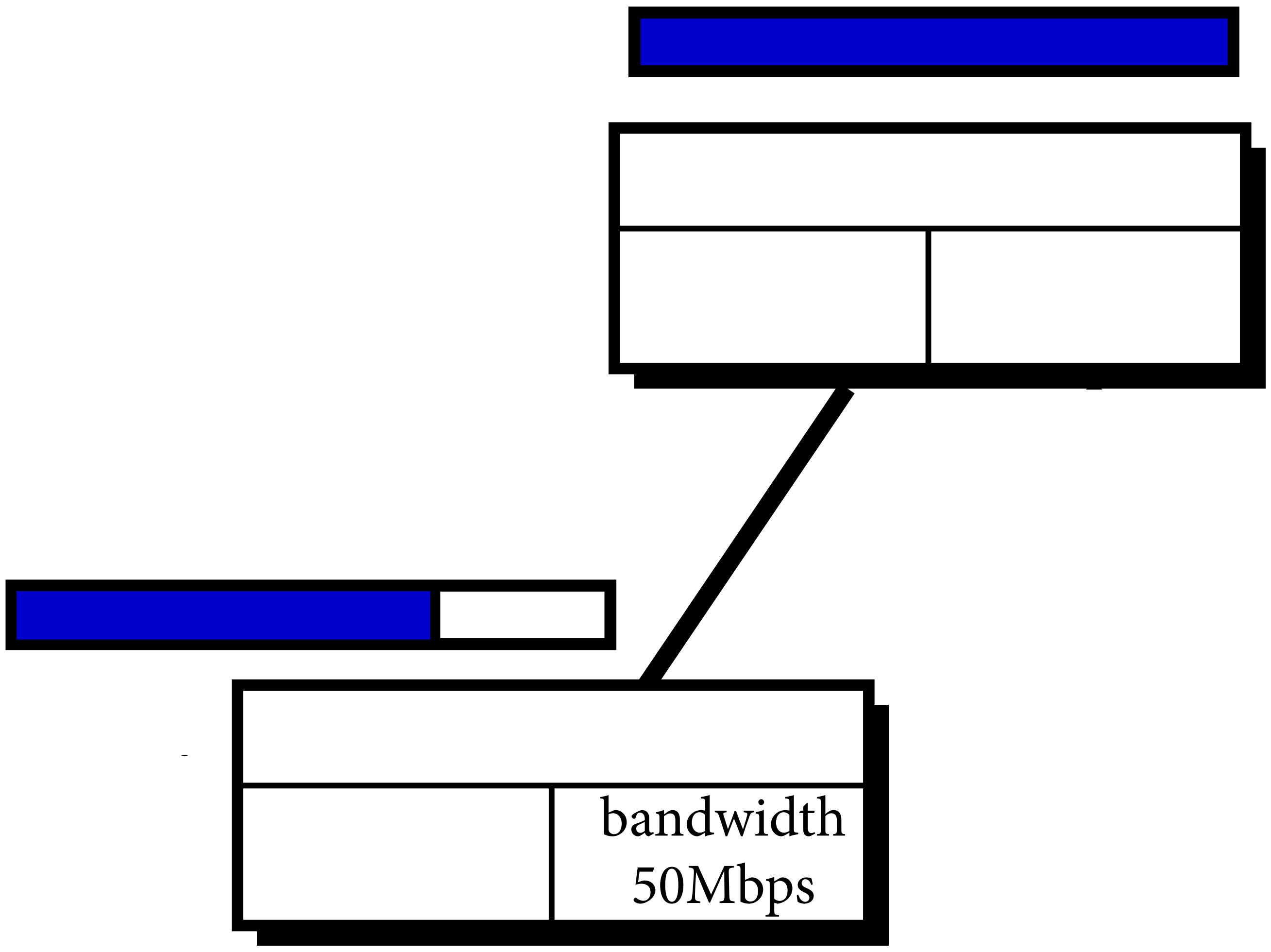
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



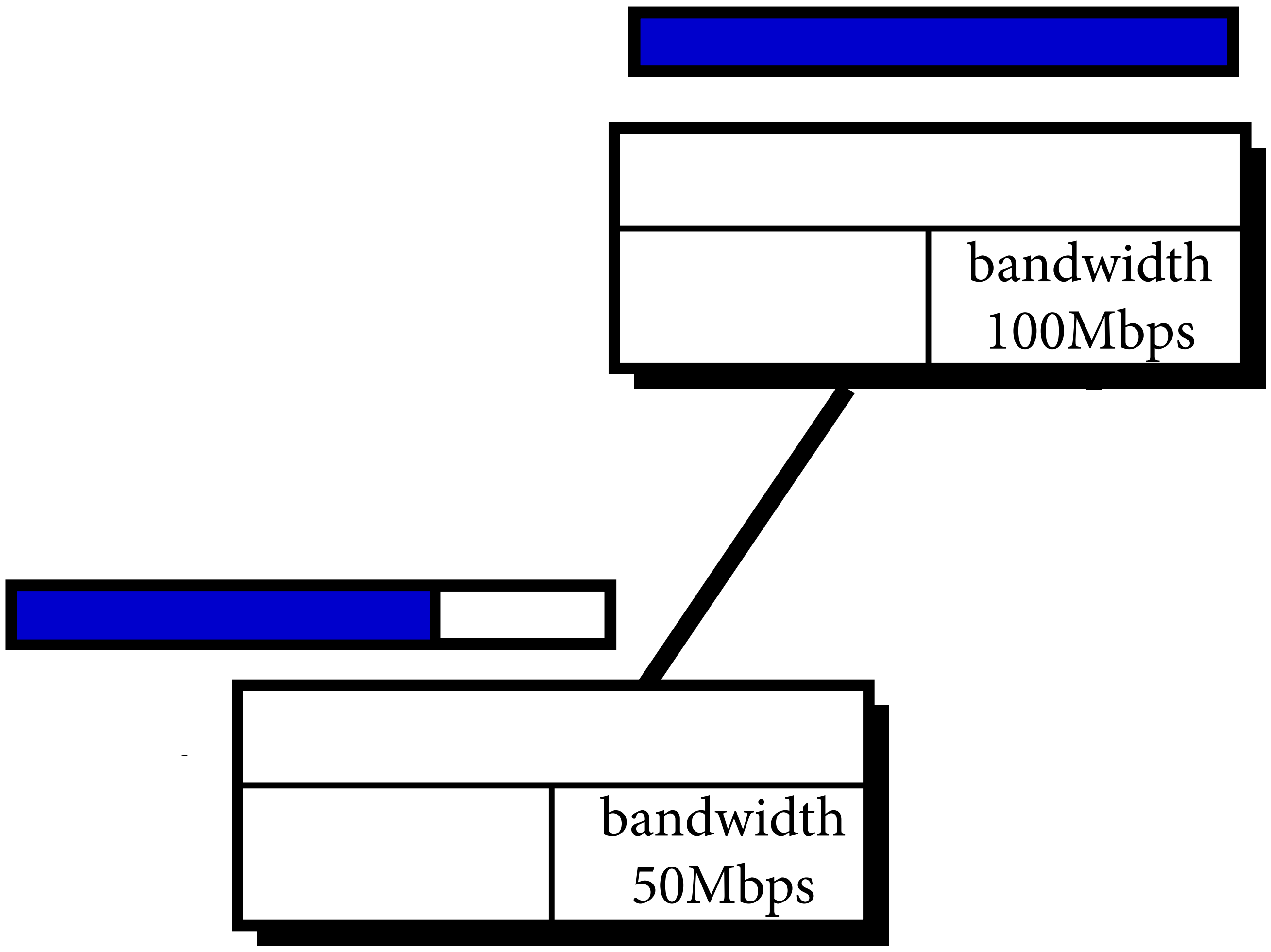
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



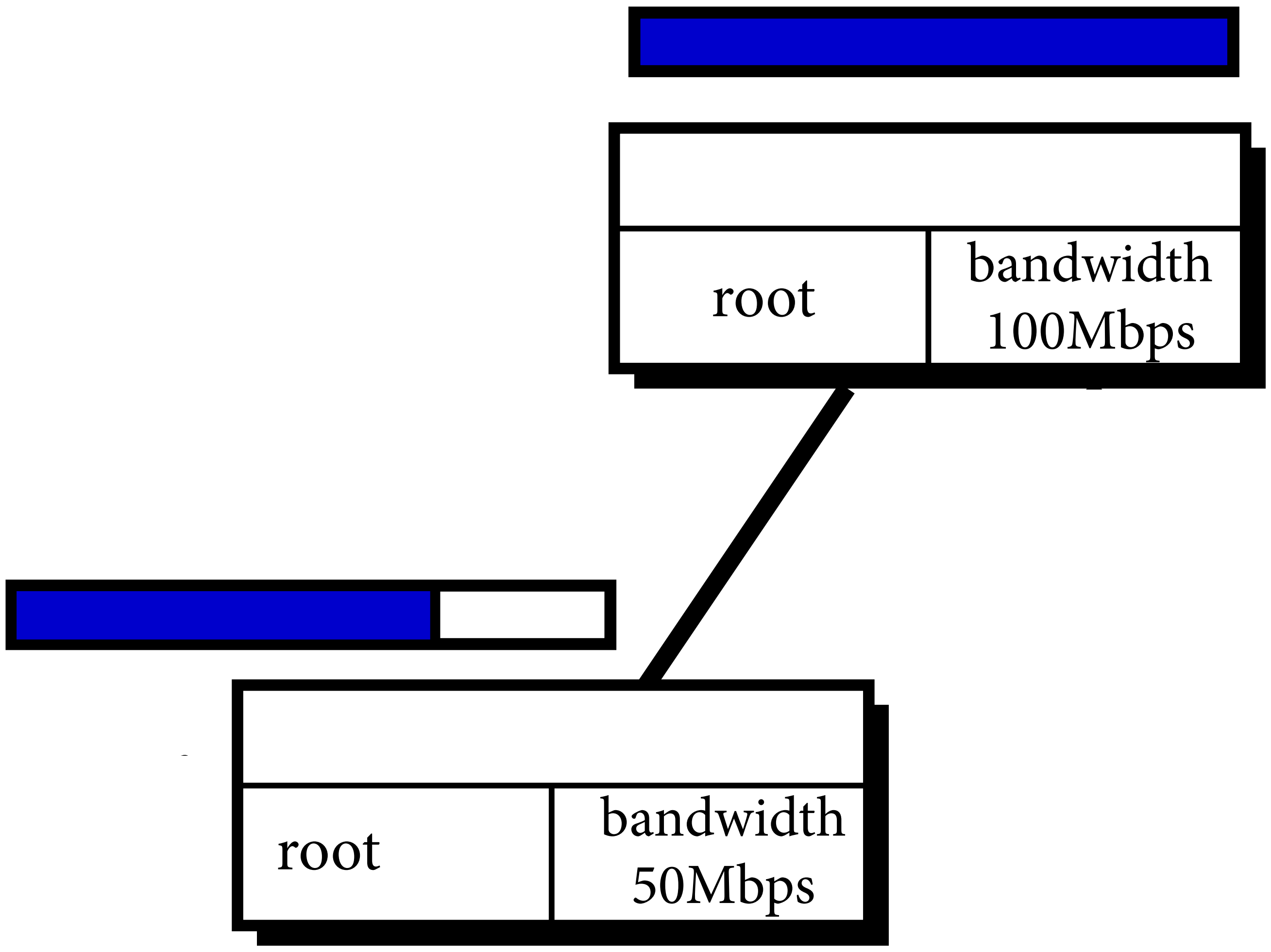
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



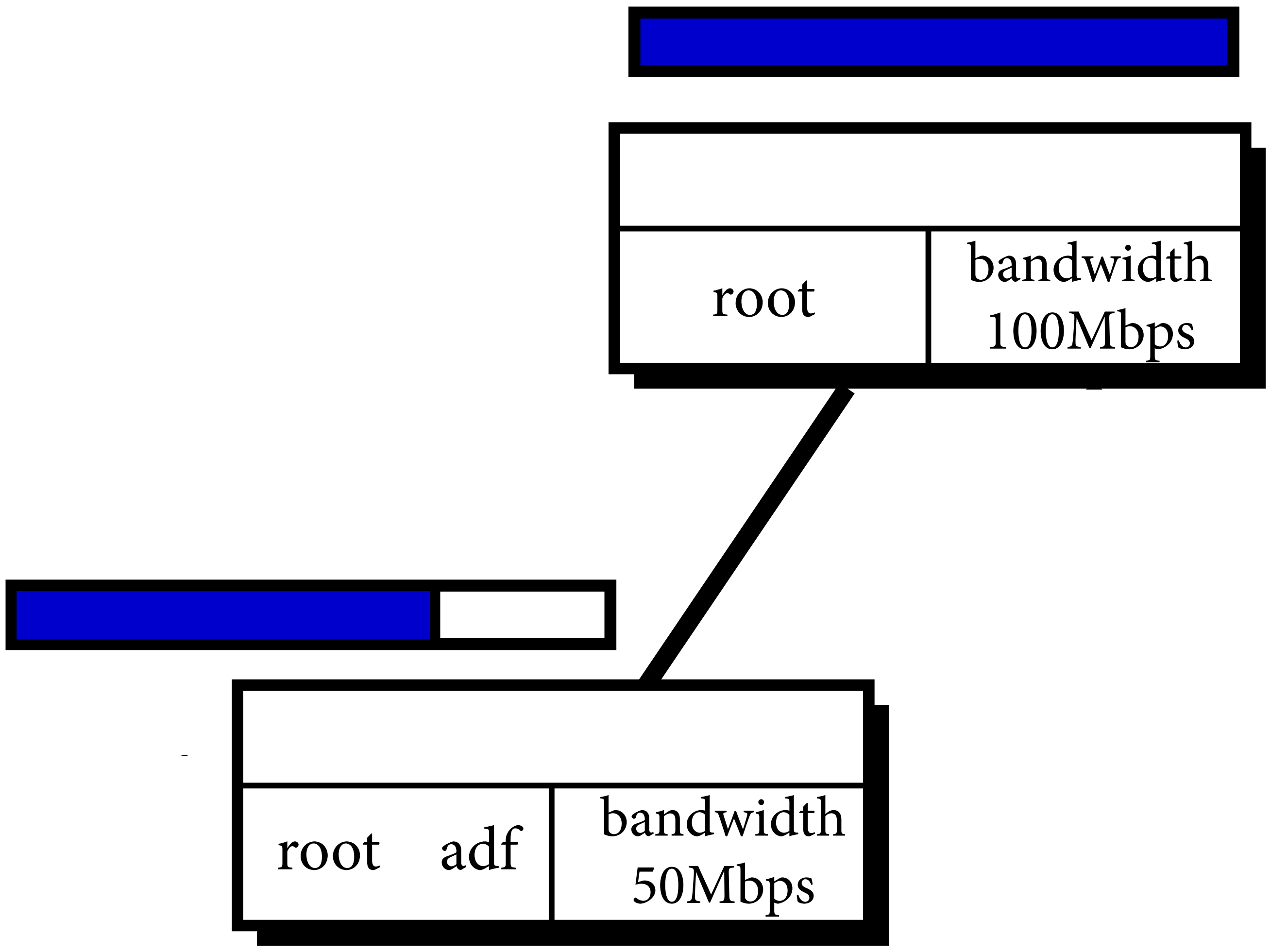
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



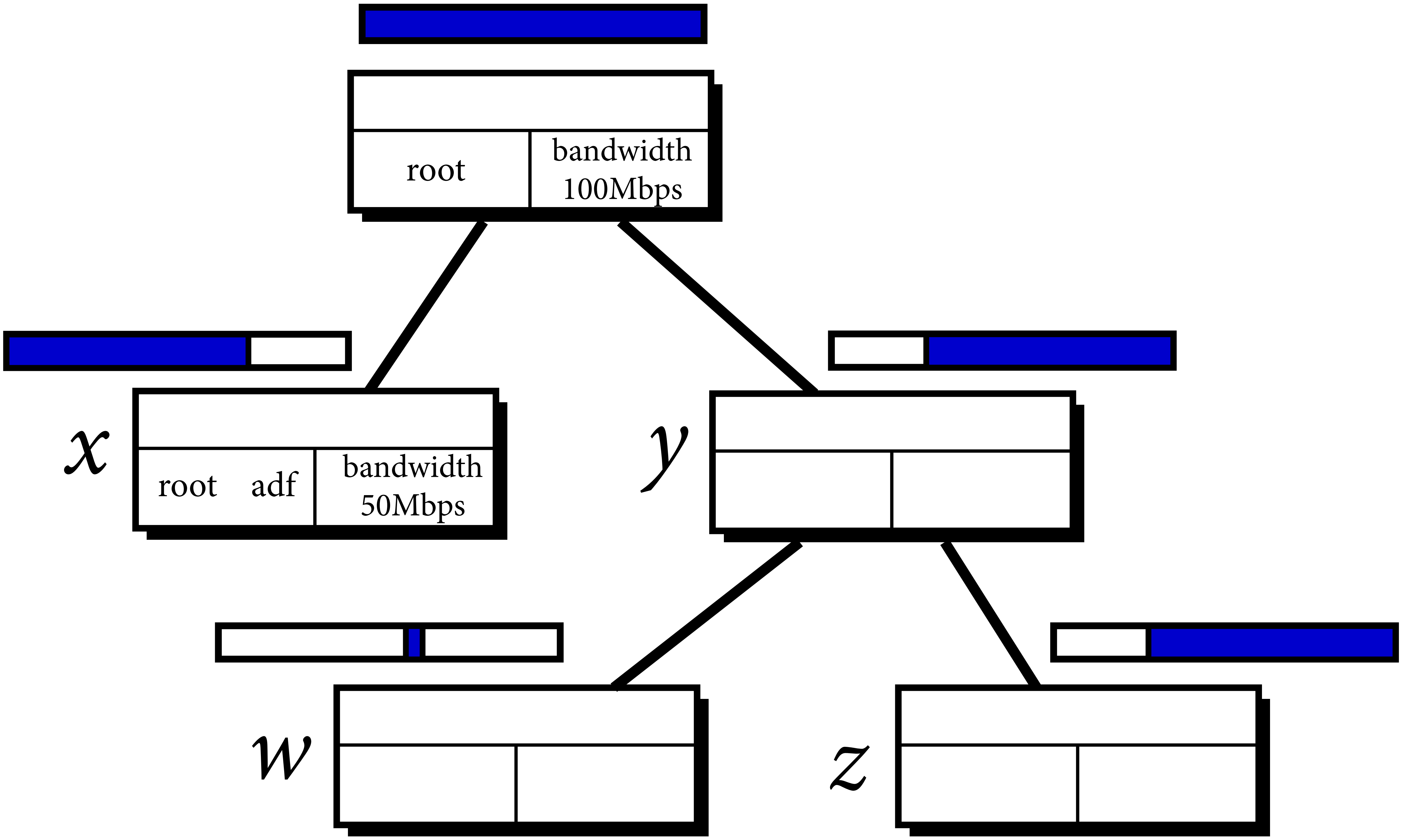
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



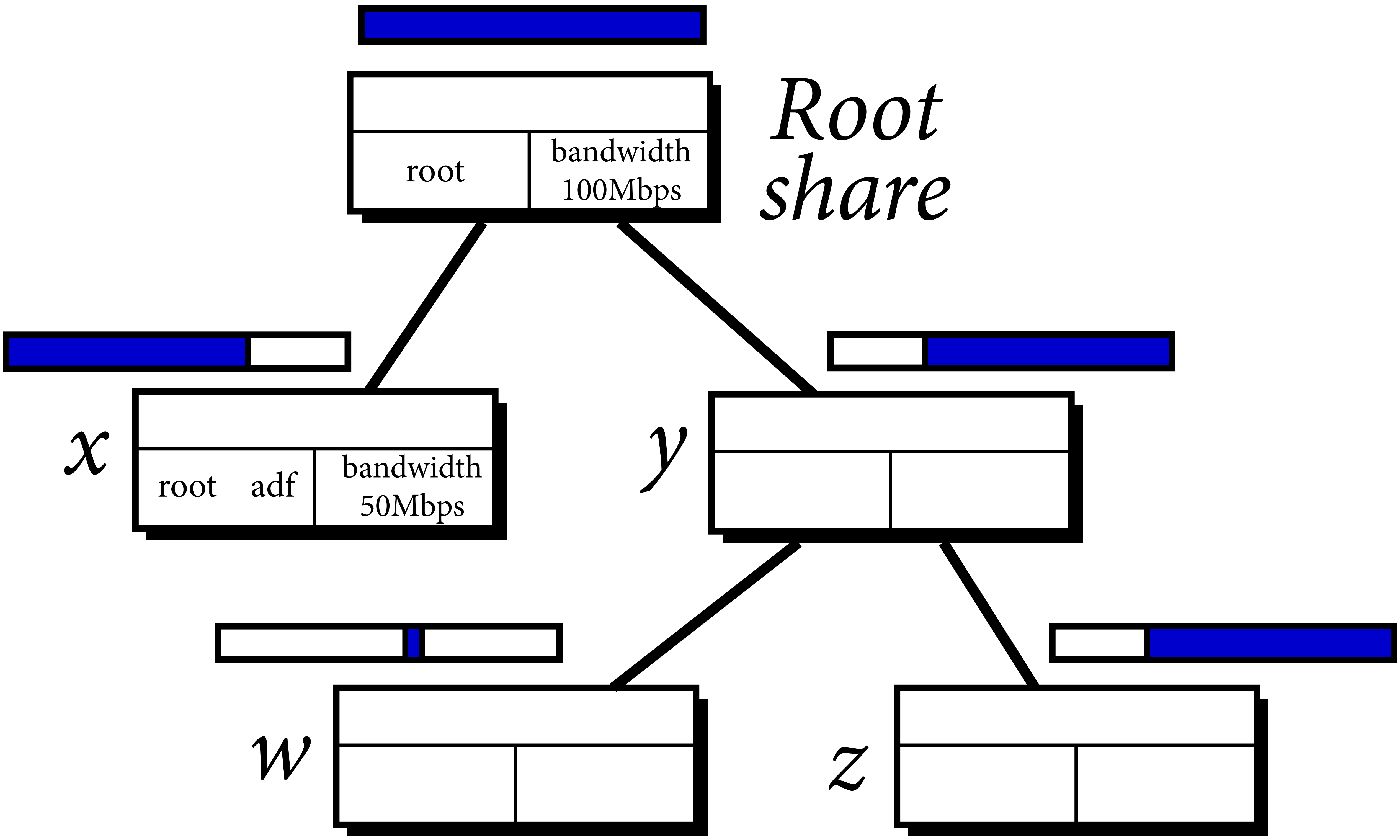
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



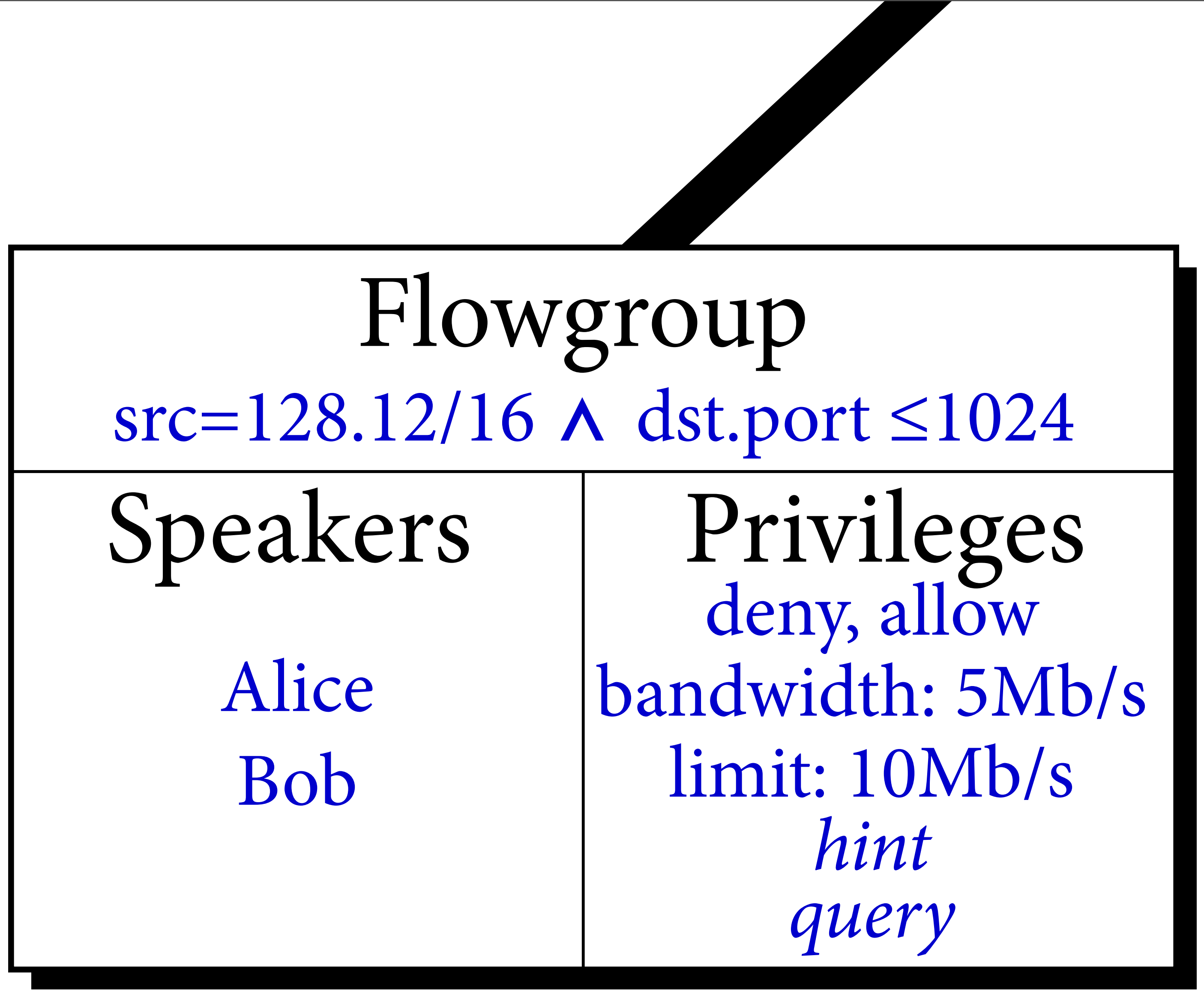
Share Tree

A share's principals also have the capability to delegate privileges by creating subshares (click). The creation of subshares is guided by the principle that you can't give away more authority than you have.

For example, a subshare's flowgroup (click) must be contained within the parent share's flowgroup (click). Here, the blue bar represents each flowgroup's range of permitted source IP addresses.

Furthermore, a subshare may not have a more permissive action set (click) than the parent (click), and initially, the subshare's only principal is its creator (click). Other users can later be added as additional principals (click).

This process of creating subshares develops a privilege hierarchy we call the "Share Tree" (click). The root of the share tree is "the rootShare" (click) -- a share which contains all traffic in the network, comes with all privileges, and has a single root user as the principal.



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)

Flowgroup	
src=128.12/16 ∧ dst.port ≤1024	
Speakers	Privileges
Alice Bob	deny, allow bandwidth: 5Mb/s limit: 10Mb/s *hint* *query*

PANE

The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

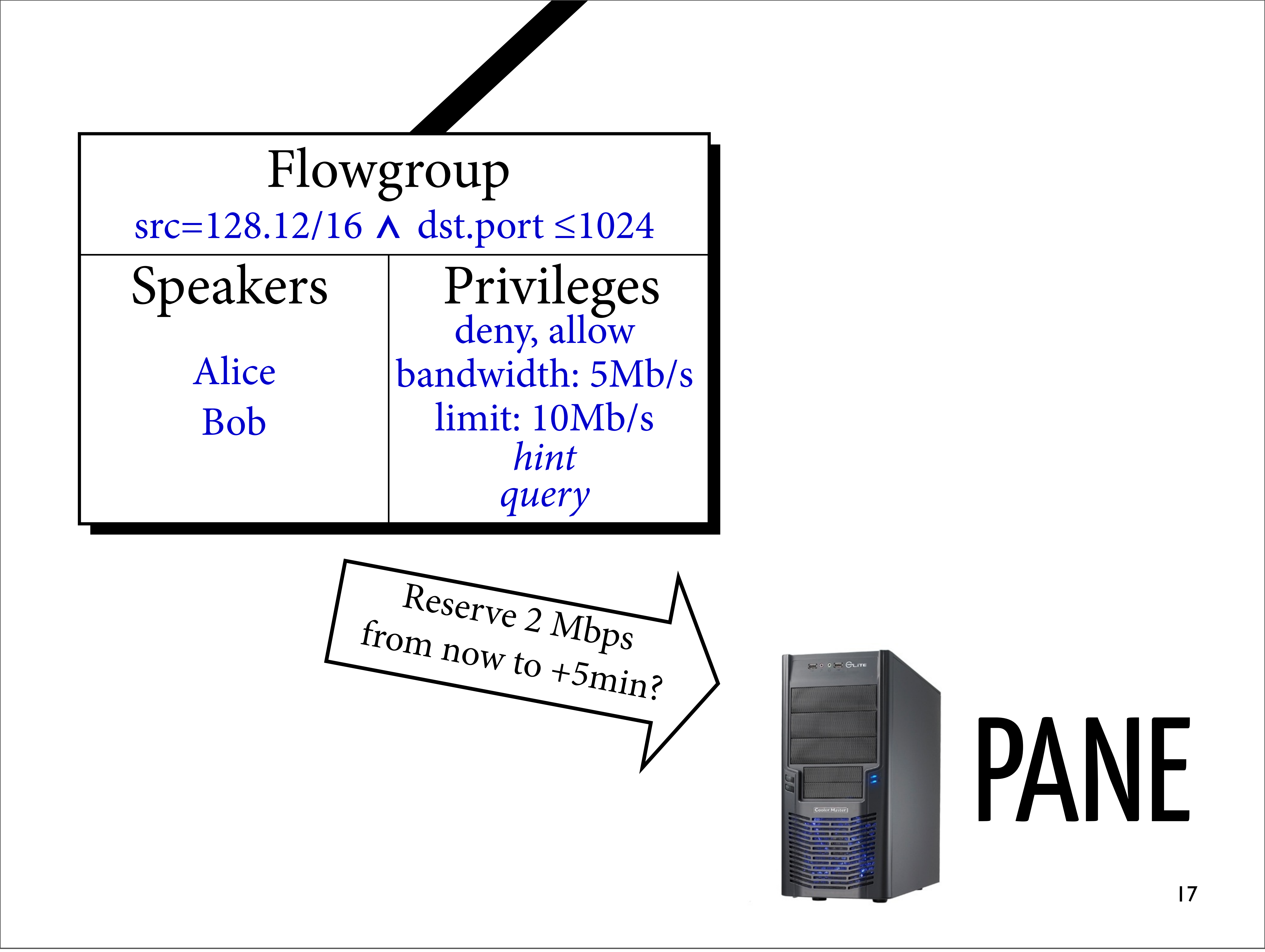
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

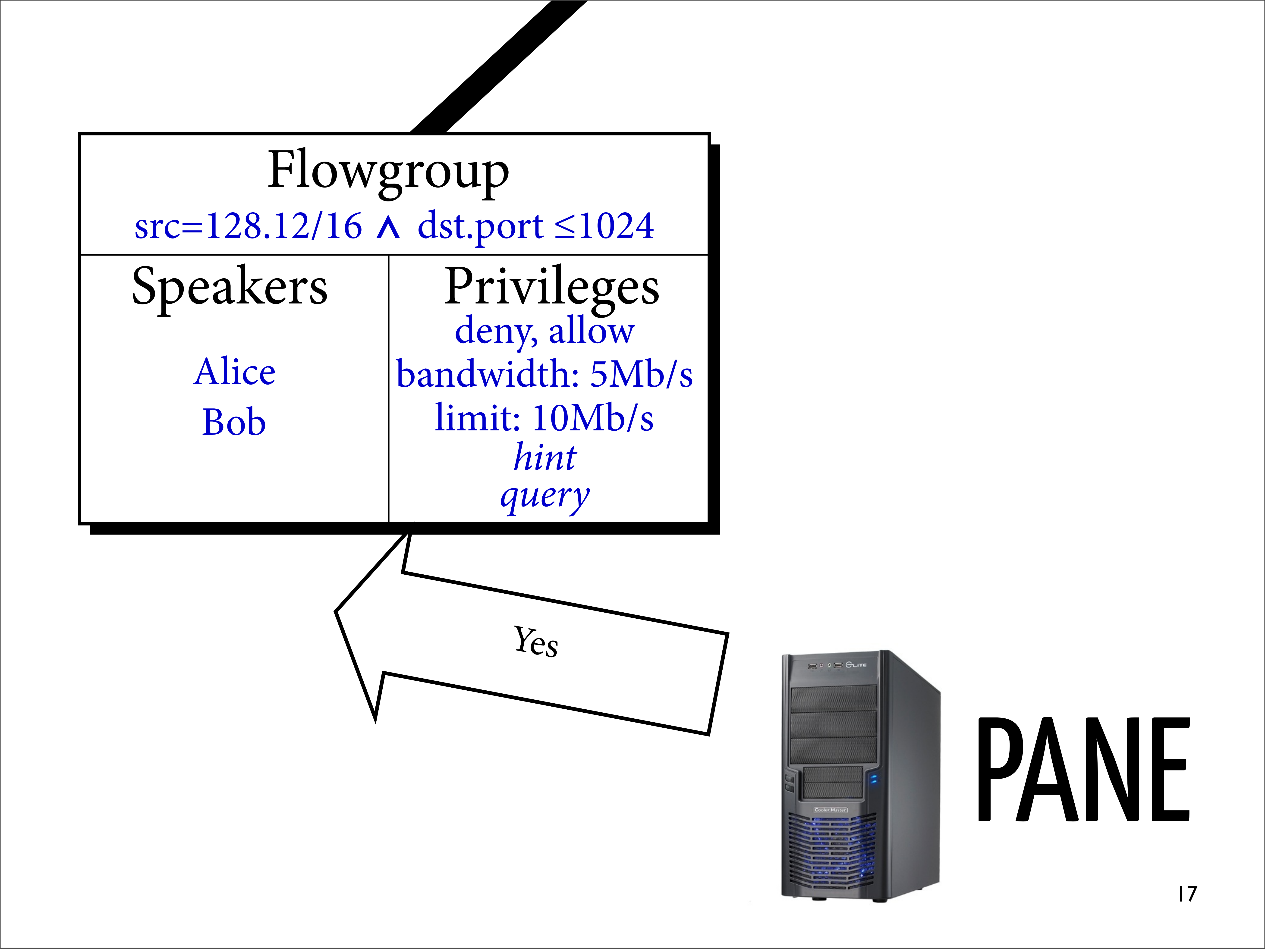
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

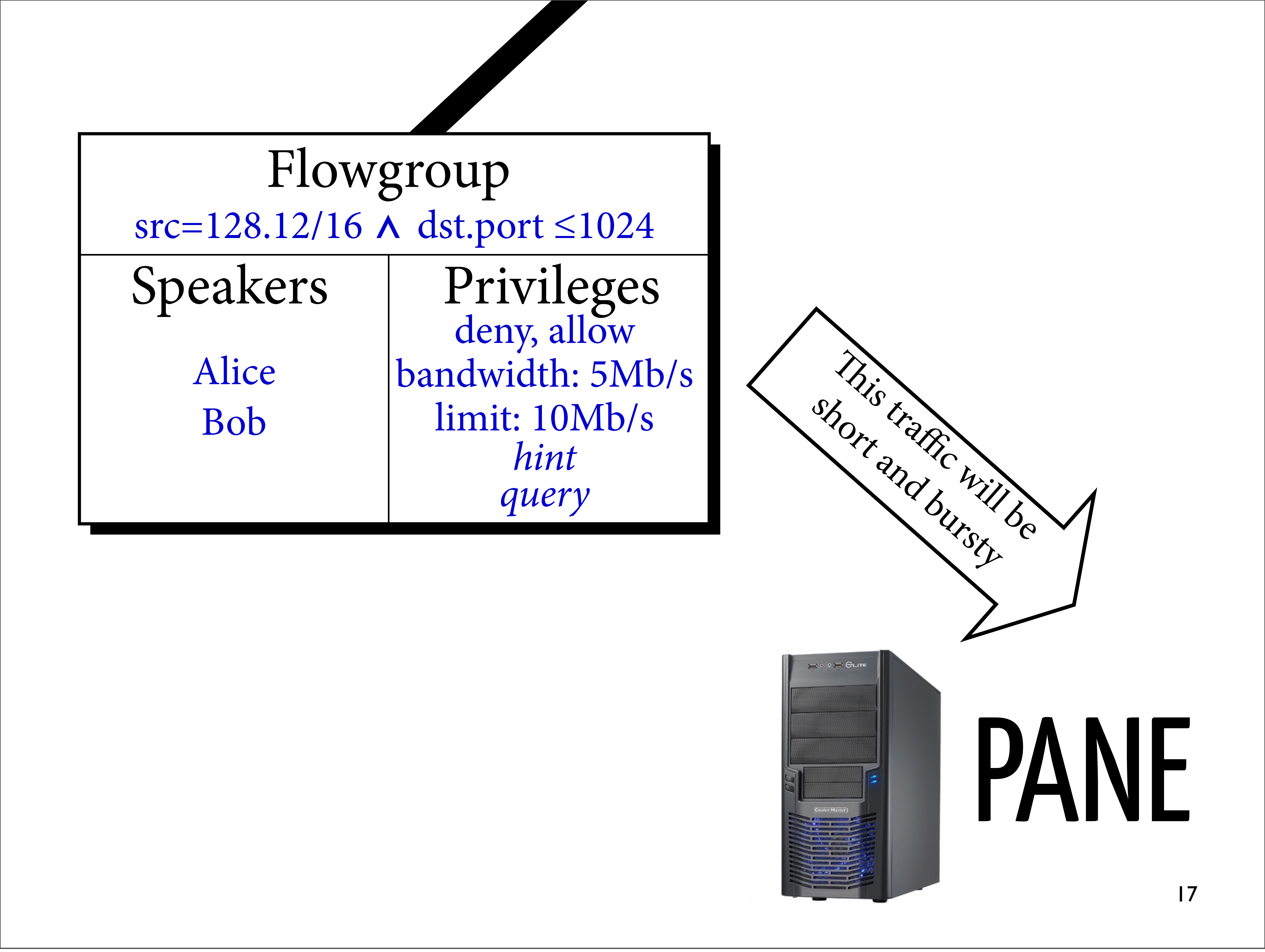
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

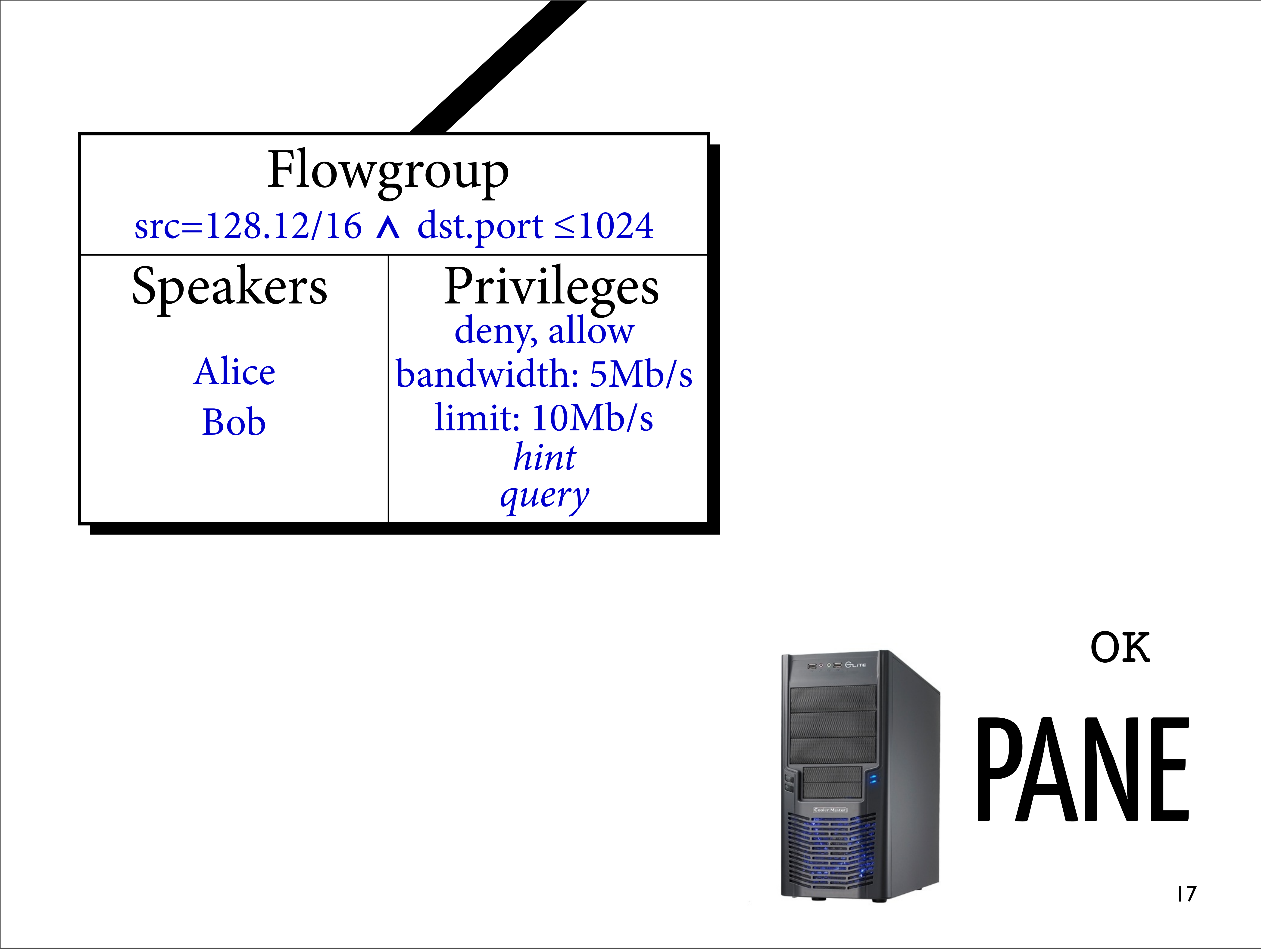
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

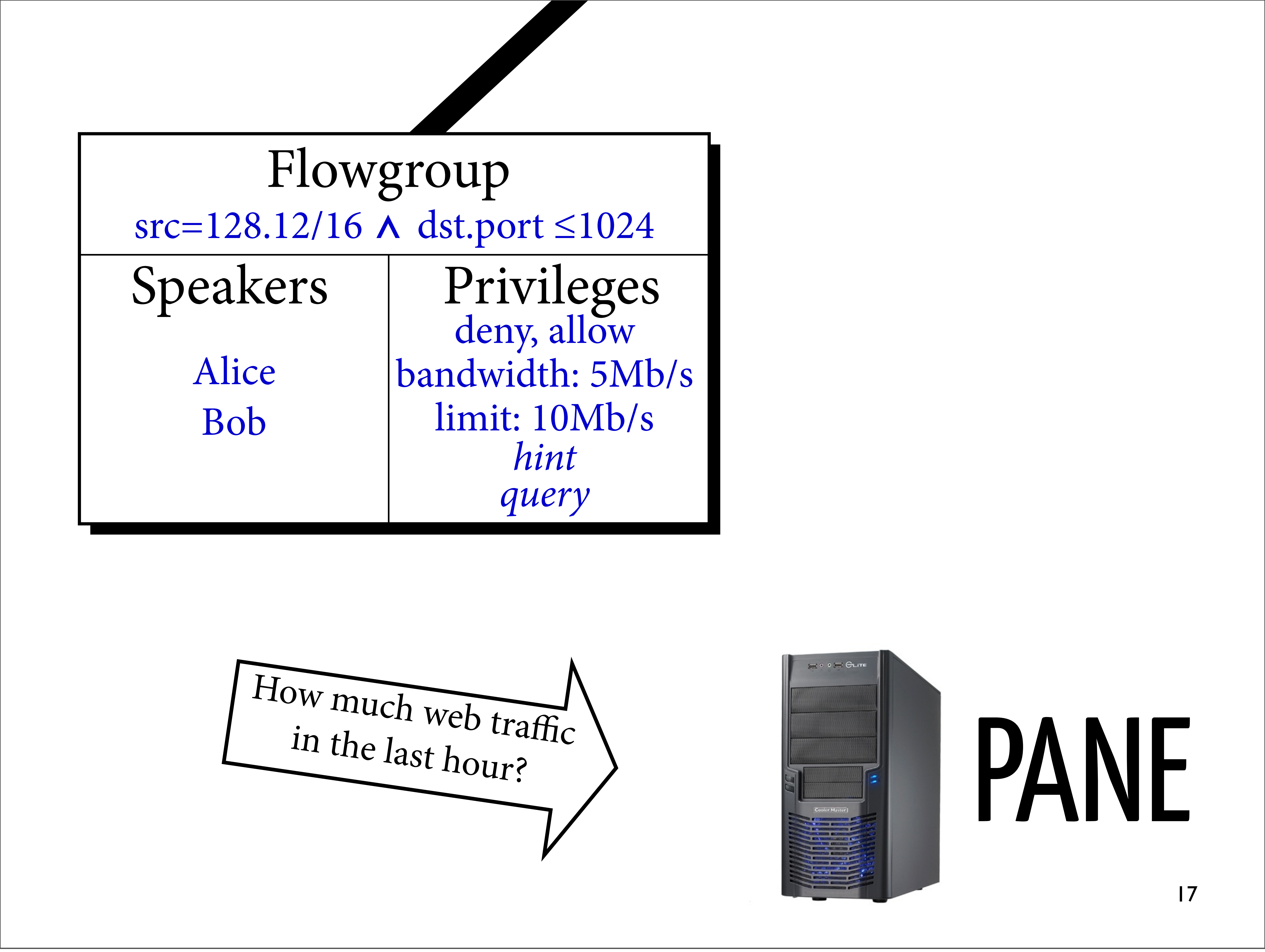
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

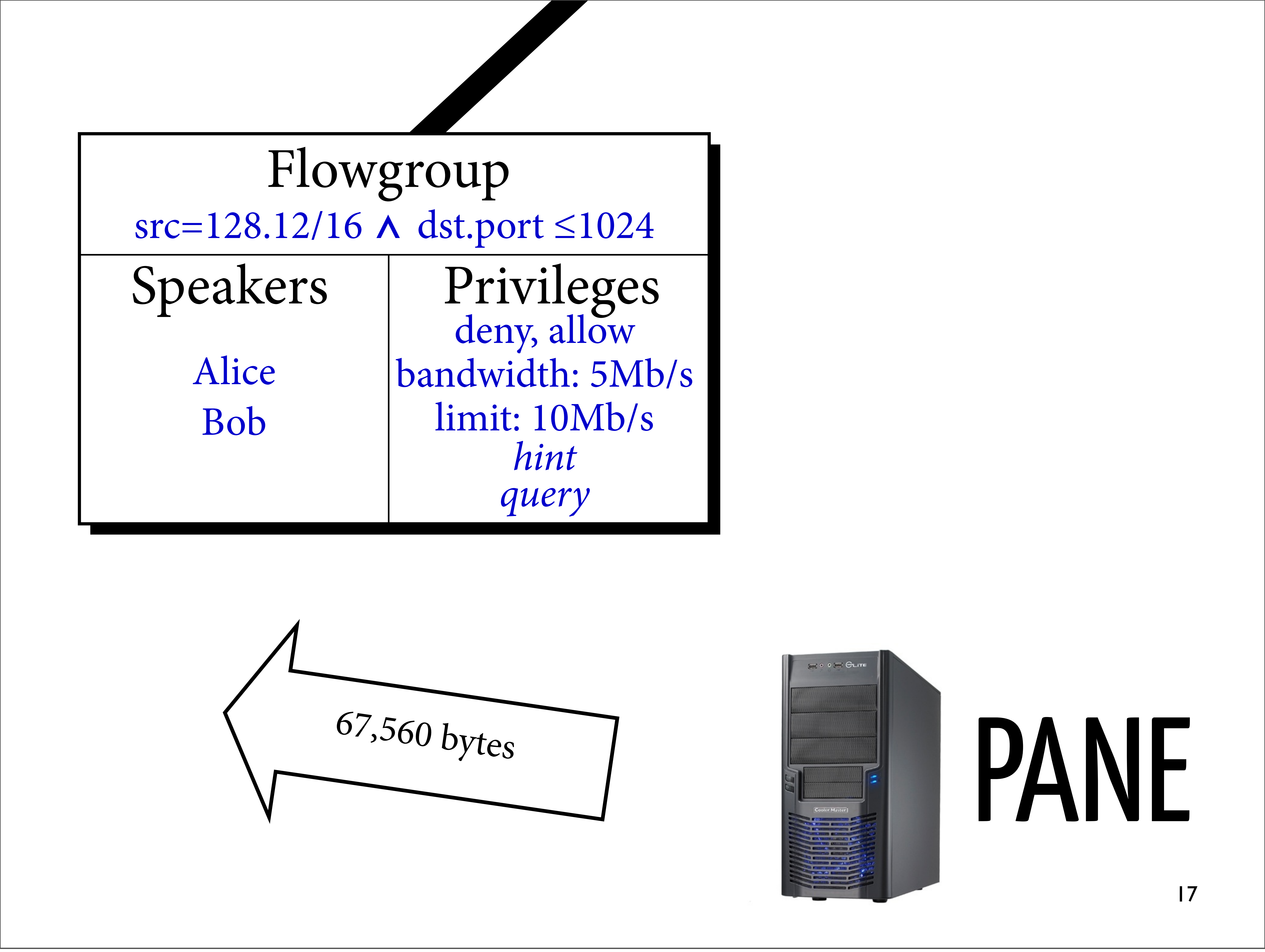
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



The share tree only sets the static context for configuring the network. The actual configuration is performed by requests and hints to the PANE controller (click).

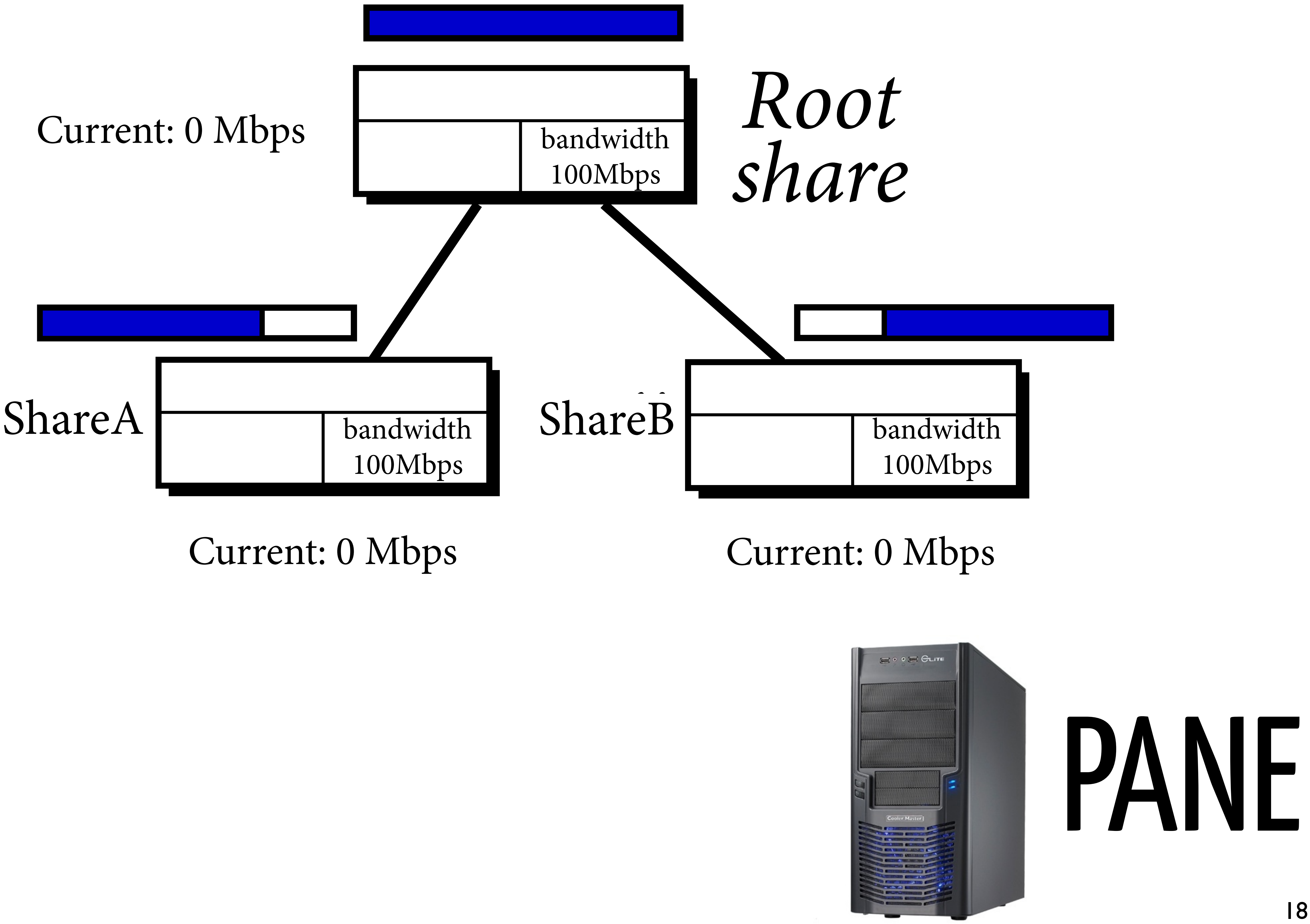
Requests describe an action the principal would like to perform on a flowgroup during a given time interval (click). After evaluating the request, the PANE controller returns an immediate response indicating an accept or reject (click).

Hints provide information about current or future traffic patterns (click). The PANE controller is not required to respond to hints and may optionally choose an action to perform on the traffic (click).

Shares may also provide principals with the right to issue queries about given flowgroups (click), such as for traffic statistics (click).

To keep things simple, I'm going to focus on requests for the remainder of this talk. More details about hints and queries can be found in our paper.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

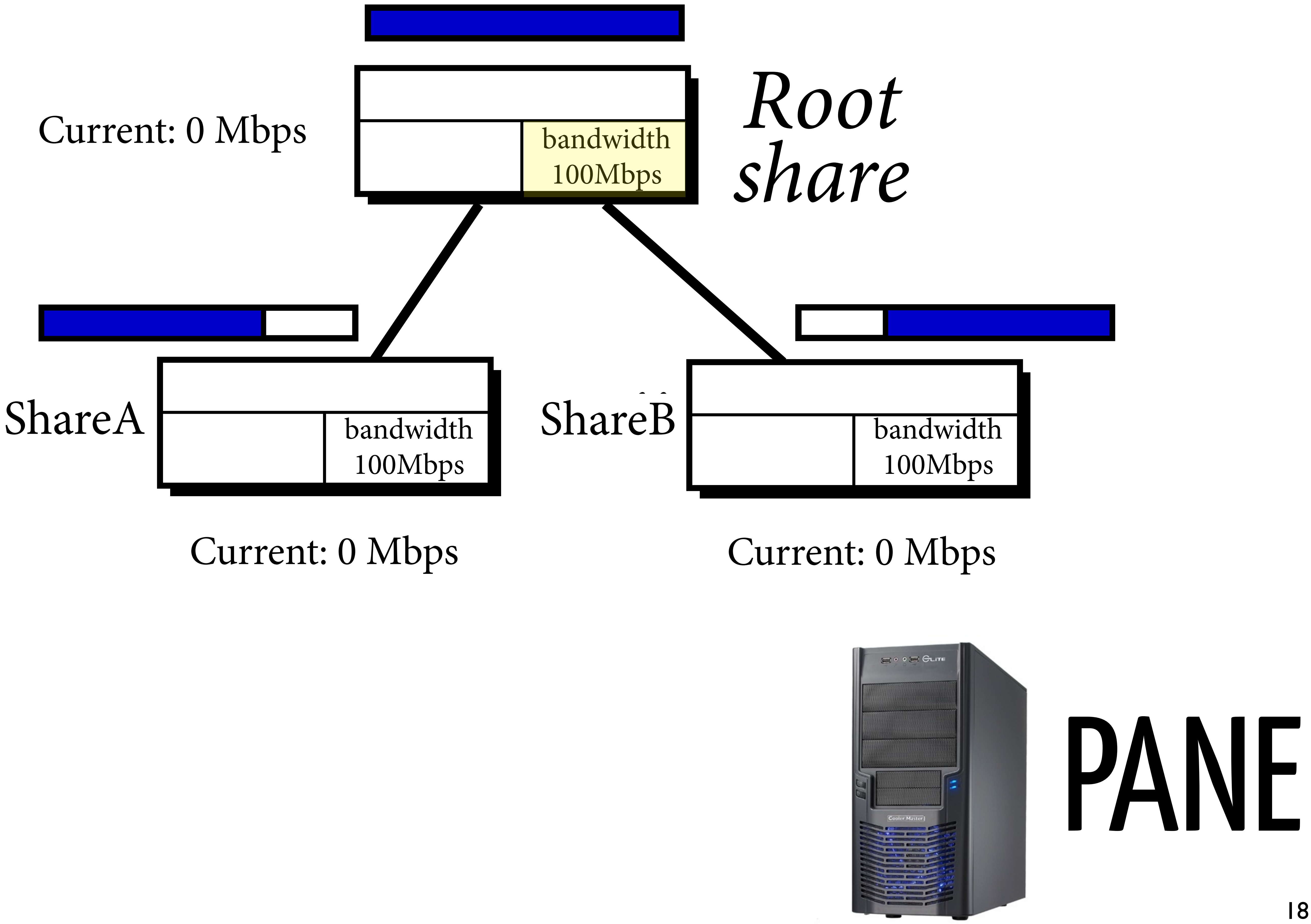
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

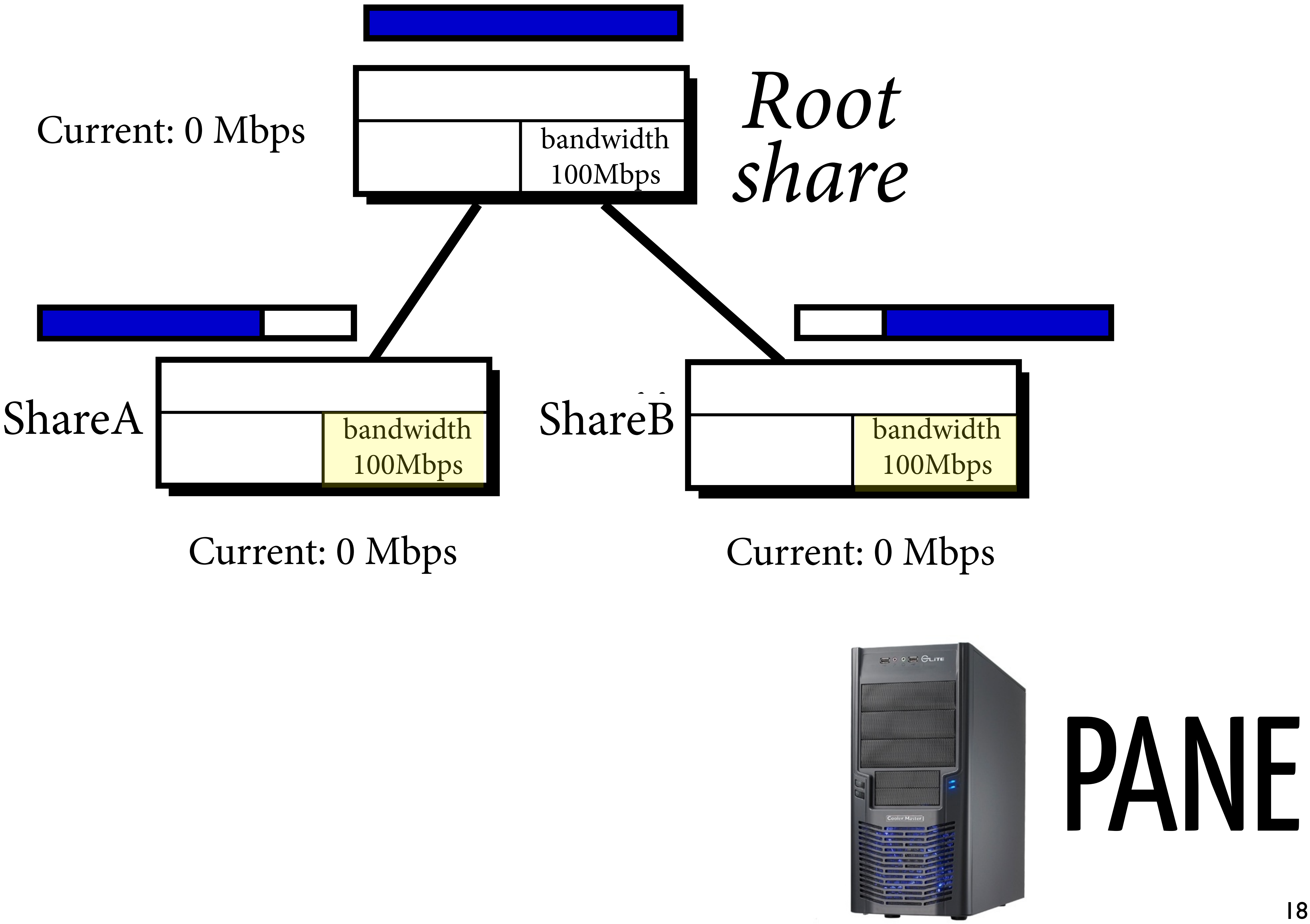
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

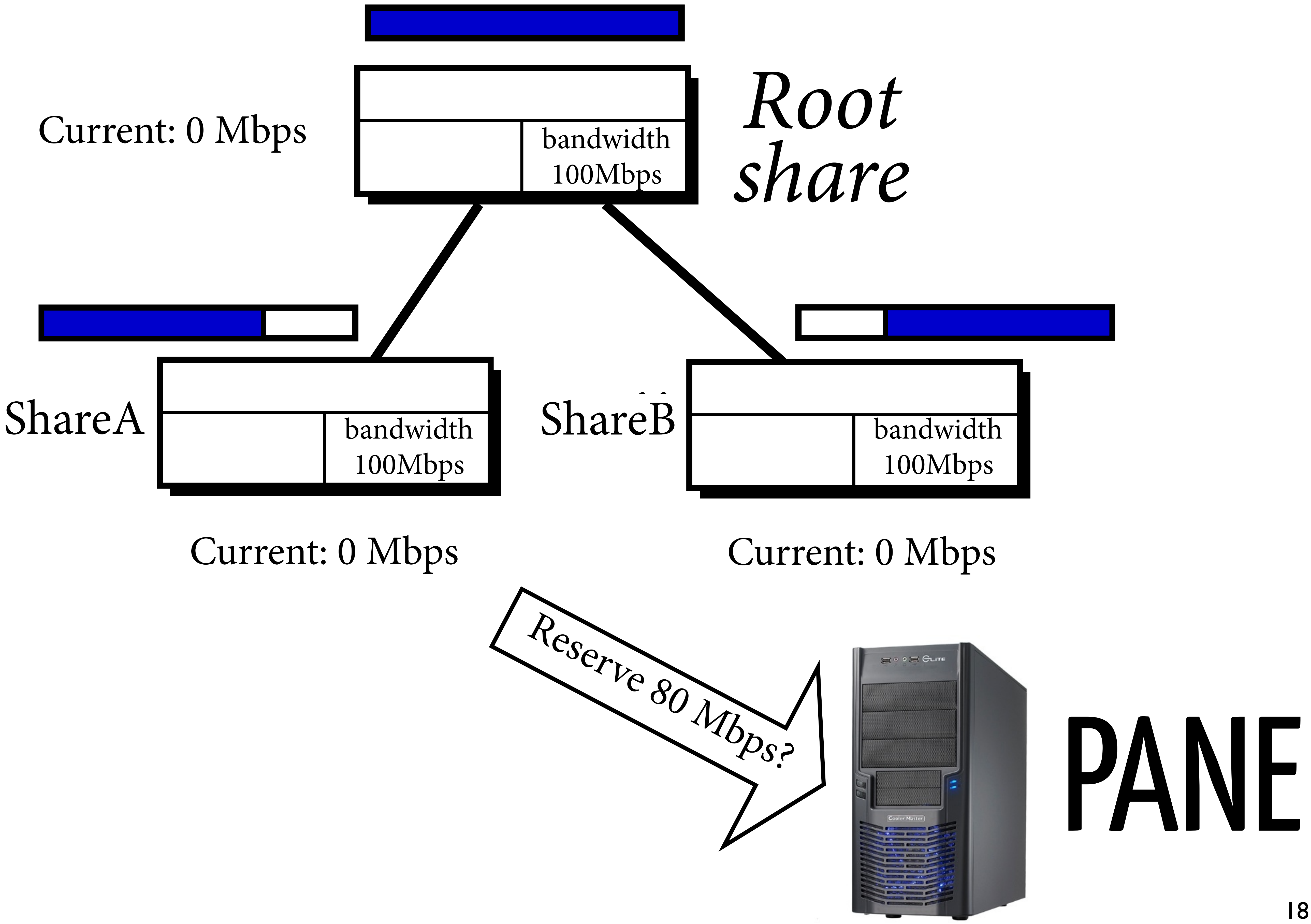
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

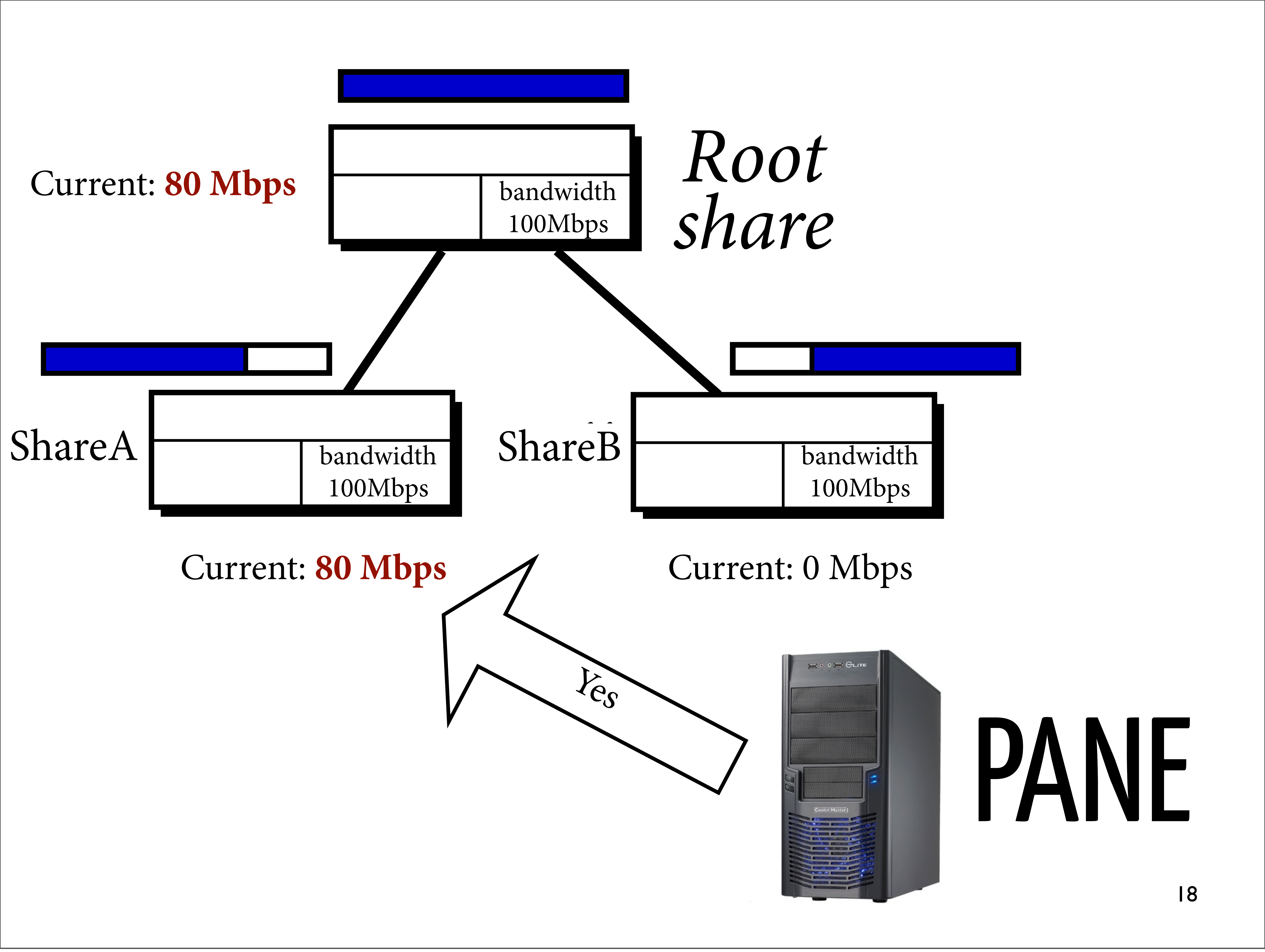
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

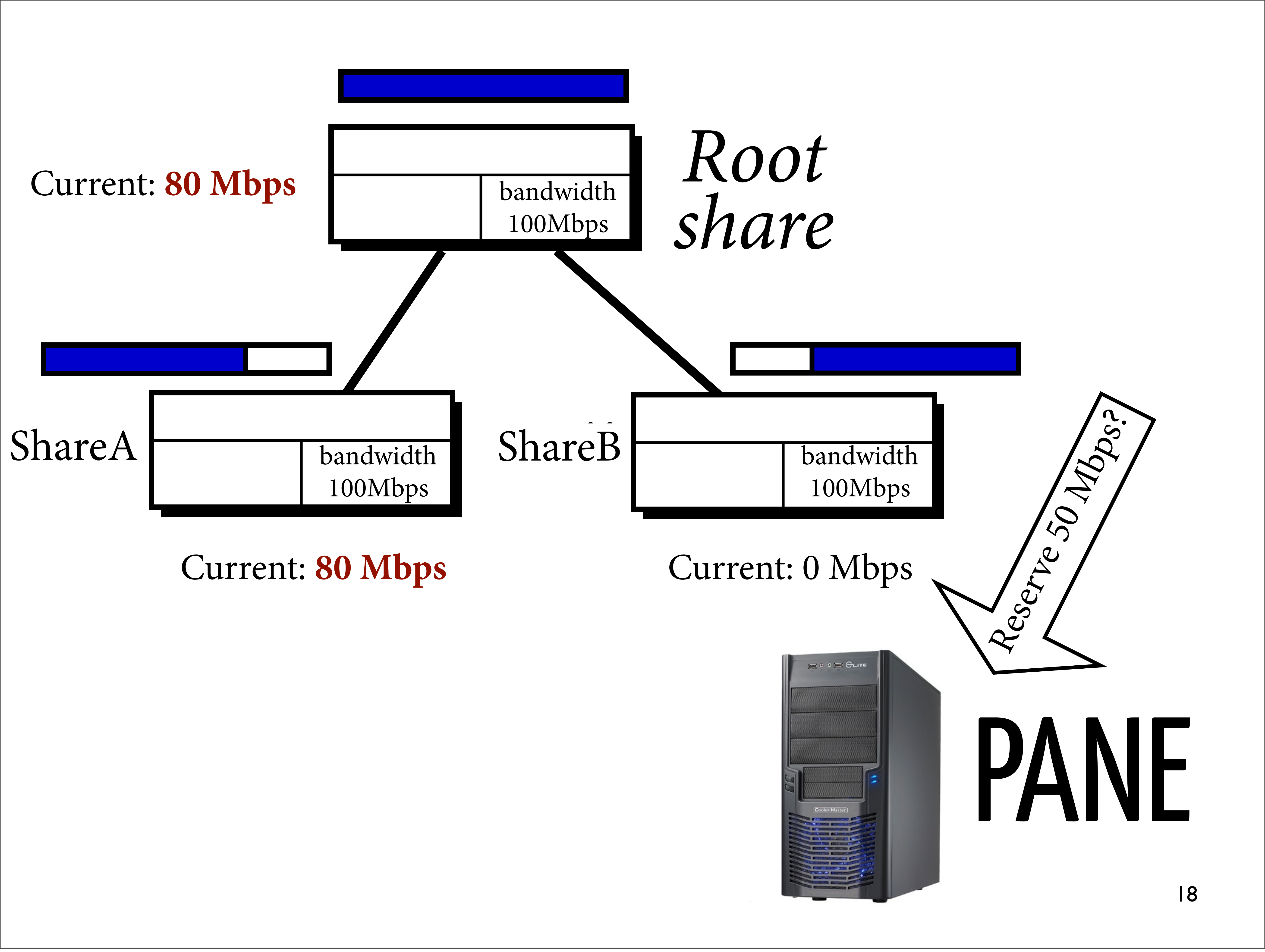
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

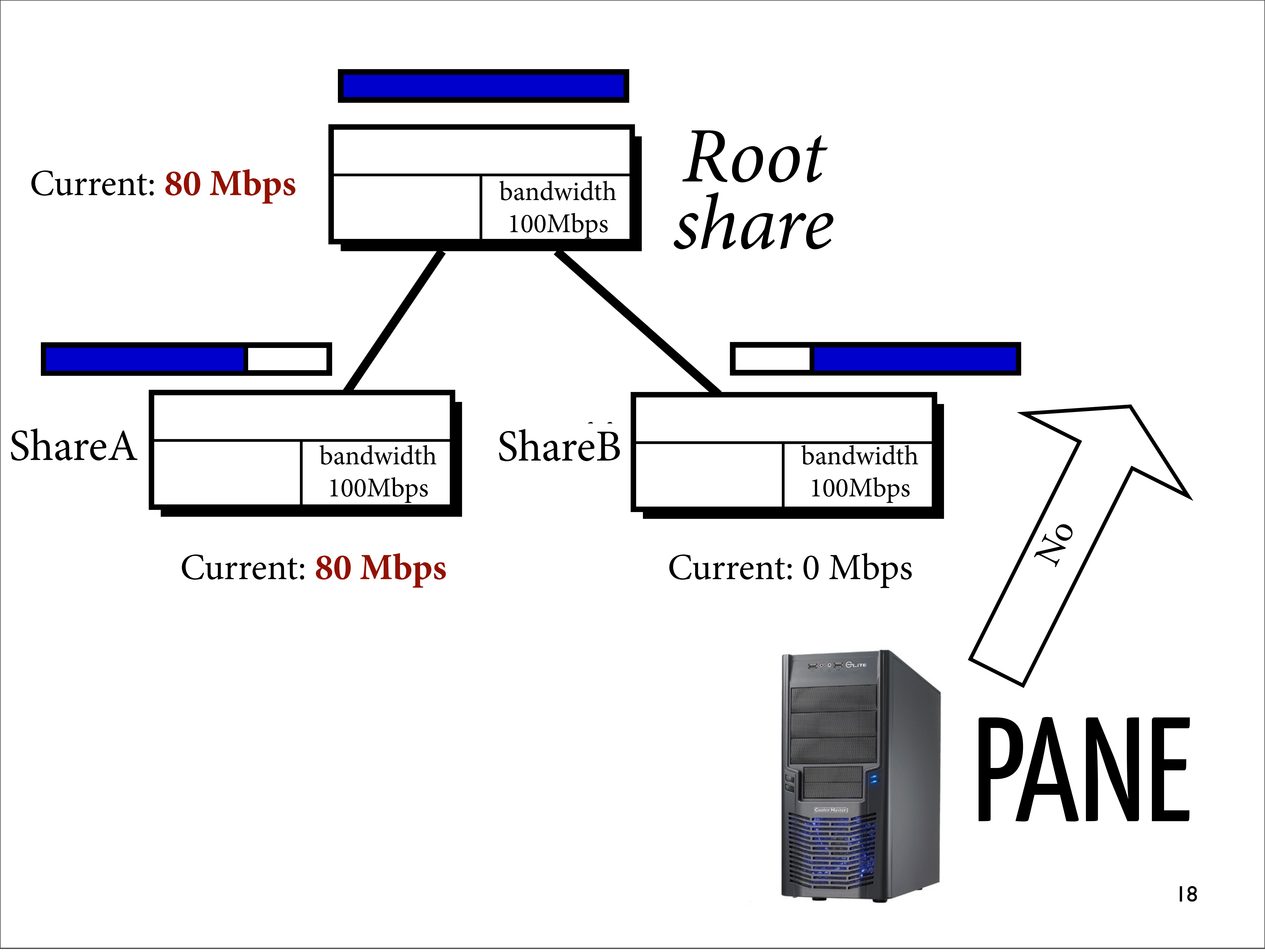
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

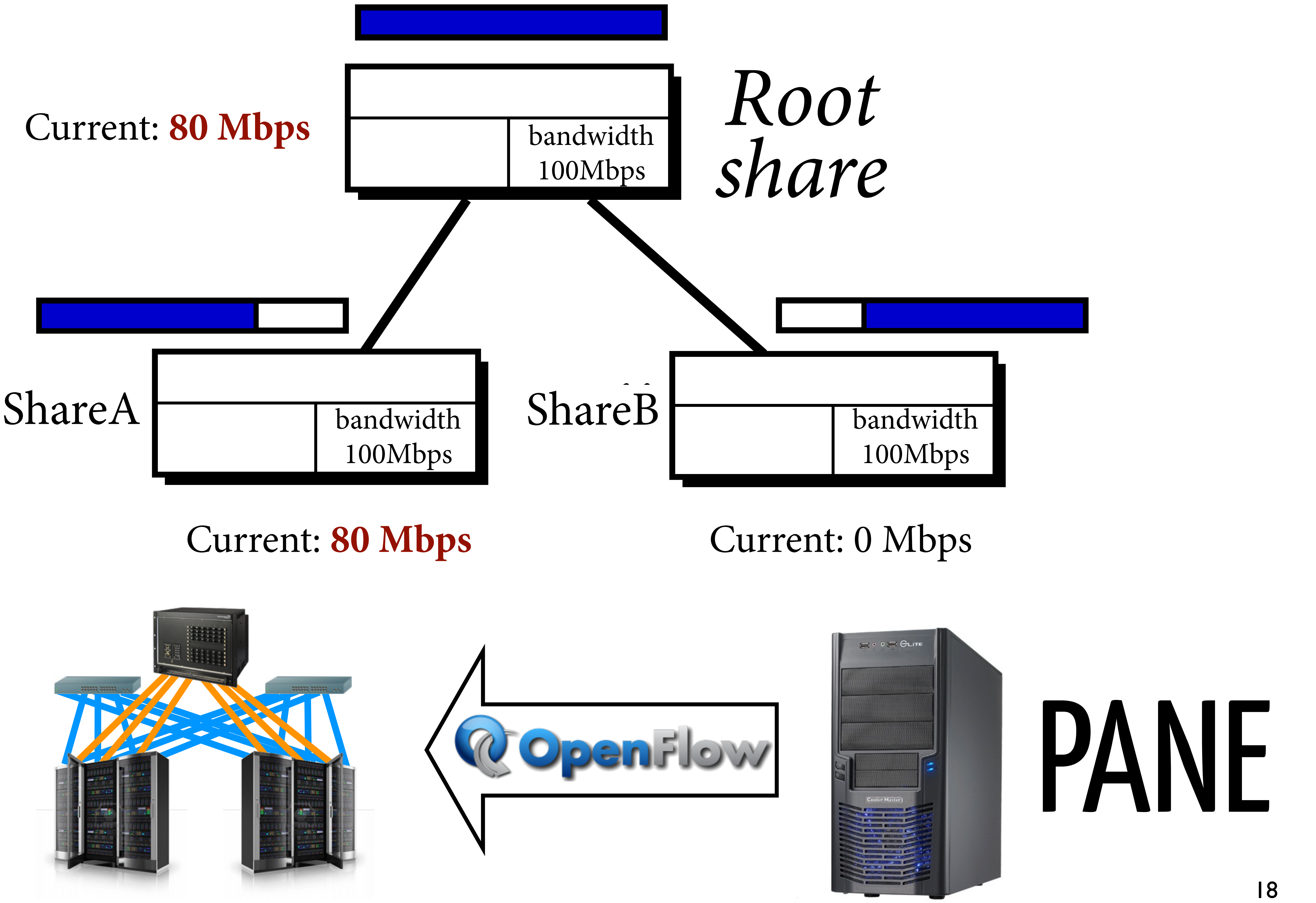
For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)



By design, a share's resources may be over-subscribed by its subshares.

For example, a share which is permitted up to (click) 100 Mbps of guaranteed minimum bandwidth may permit each of its subshares (click) to make reservations up to the same limit. In order to ensure that these restrictions are never violated, new requests are recursively evaluated up the tree.

For example, if a user of ShareA requests (click) 80 Mbps of guaranteed bandwidth, the PANE controller accepts the request (click) and accounts for the reservation in ShareA and the rootShare.

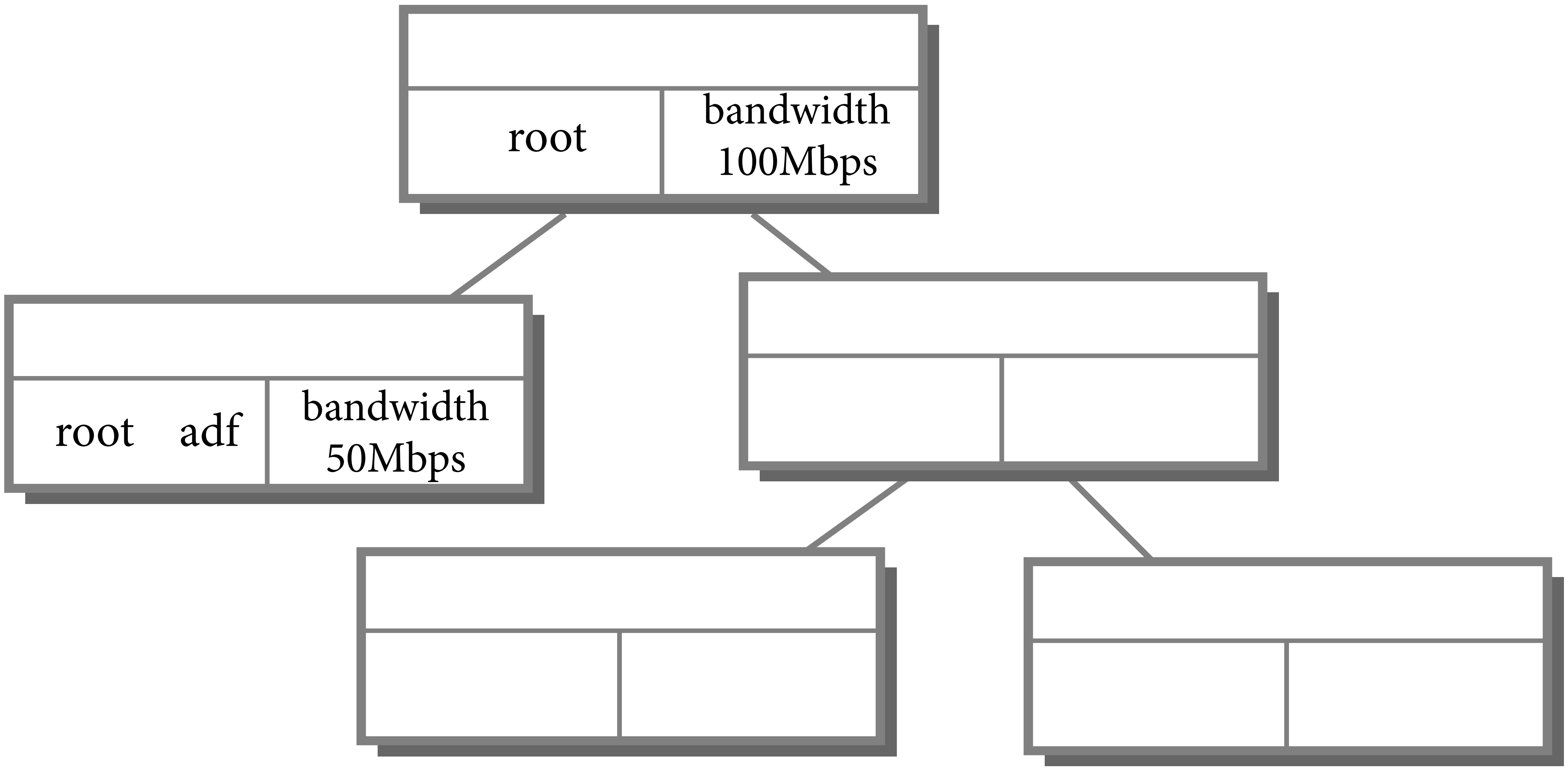
If a user of ShareB then requests (click) 50 Mbps of guaranteed bandwidth, the PANE controller rejects the request (click) to prevent a violation on the rootShare.

Finally, when accepted requests become active, the PANE controller uses OpenFlow (click) to reconfigure the network and implement the request.

(Pause)

Resolving Conflicts

To solve participatory networking's second challenge -- how to resolve conflicts between requests -- we developed Hierarchical Flow Tables, or HFTs.

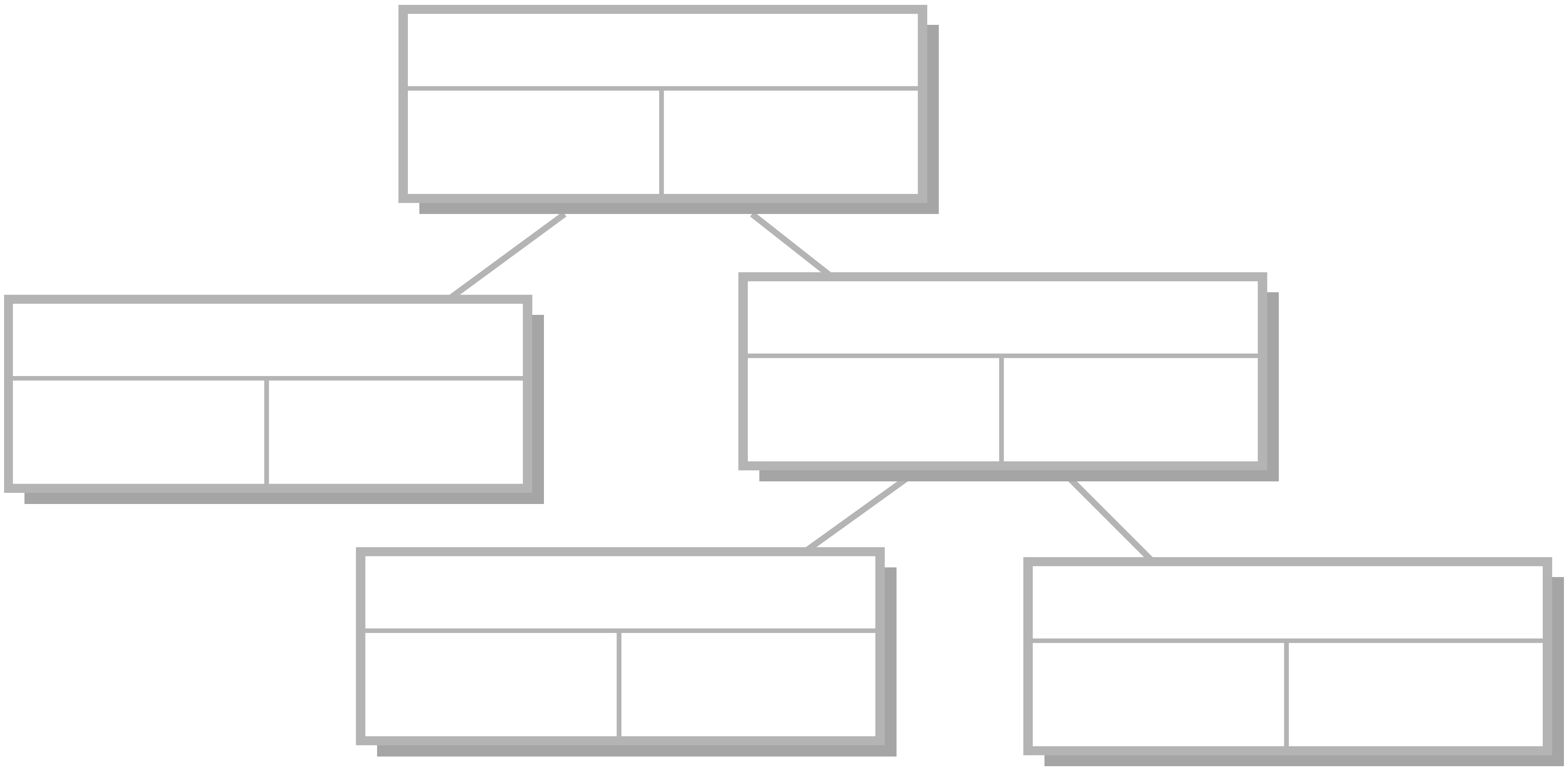


Share Tree

In PANE, we have two hierarchies.

(pause)

The first is a static hierarchy of the privileges granted to users and applications. This hierarchy sets the stage ...

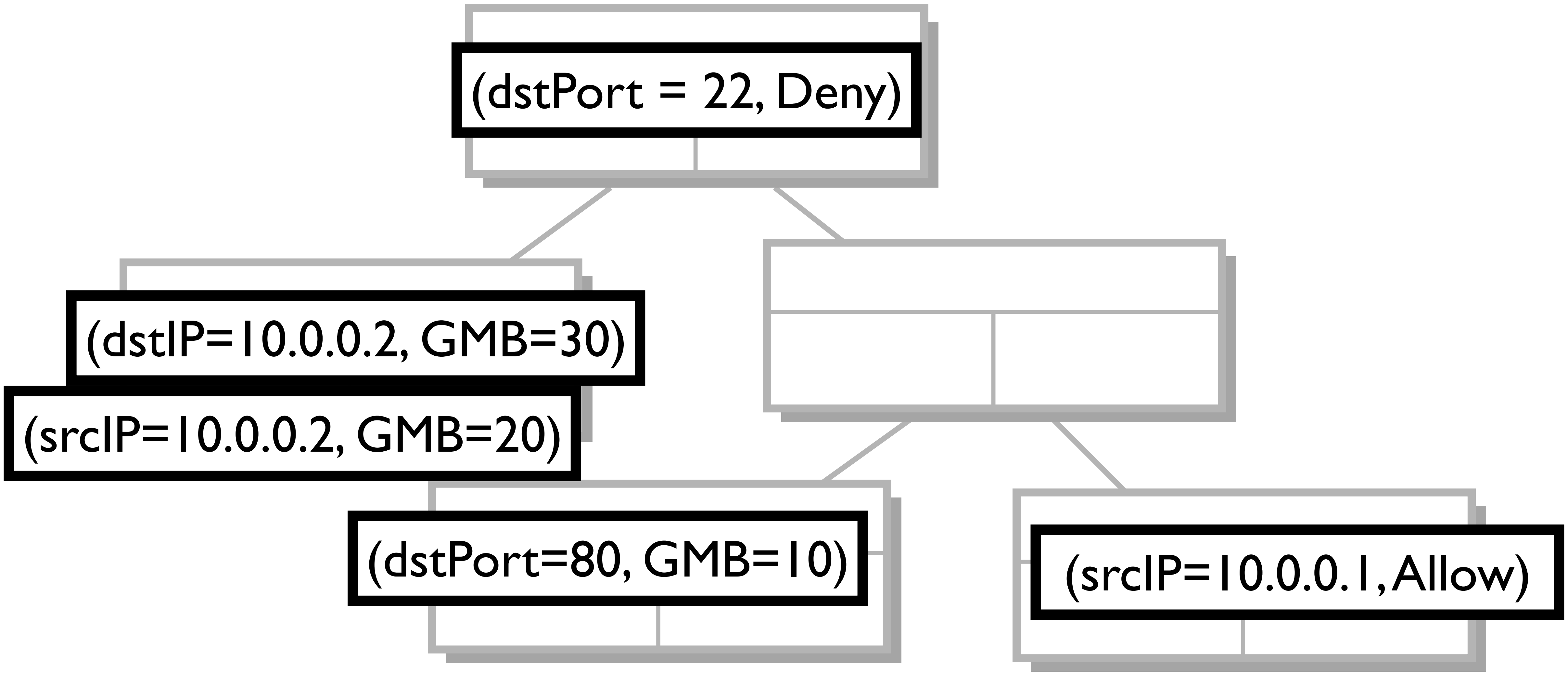


Policy Trees

… for a dynamic hierarchy of policy requests. As users and applications make requests,

(click) the policy trees evolve, always within the bounds set by the Share Tree.

(pause)

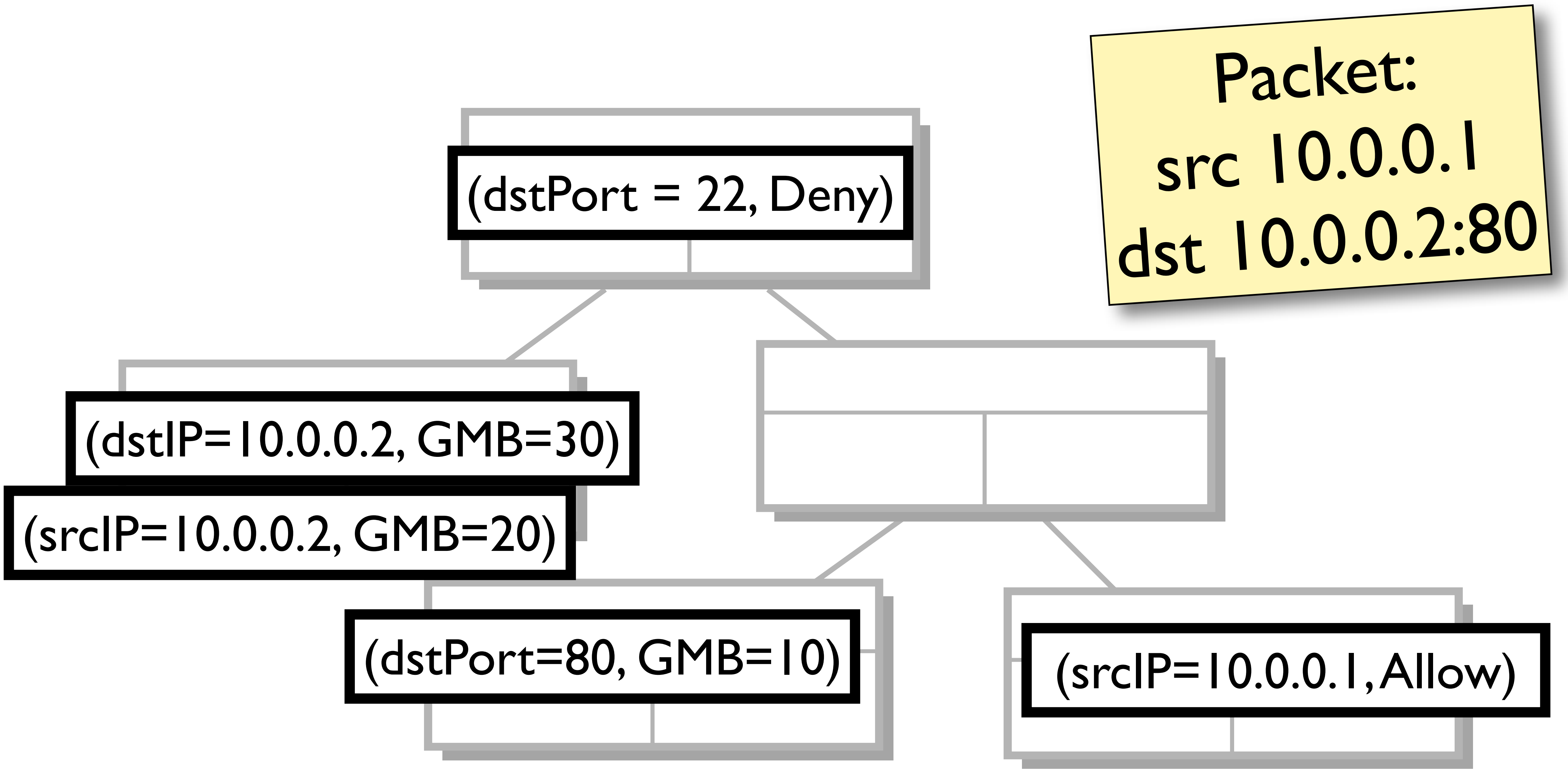


Policy Trees

… for a dynamic hierarchy of policy requests. As users and applications make requests,

(click) the policy trees evolve, always within the bounds set by the Share Tree.

(pause)

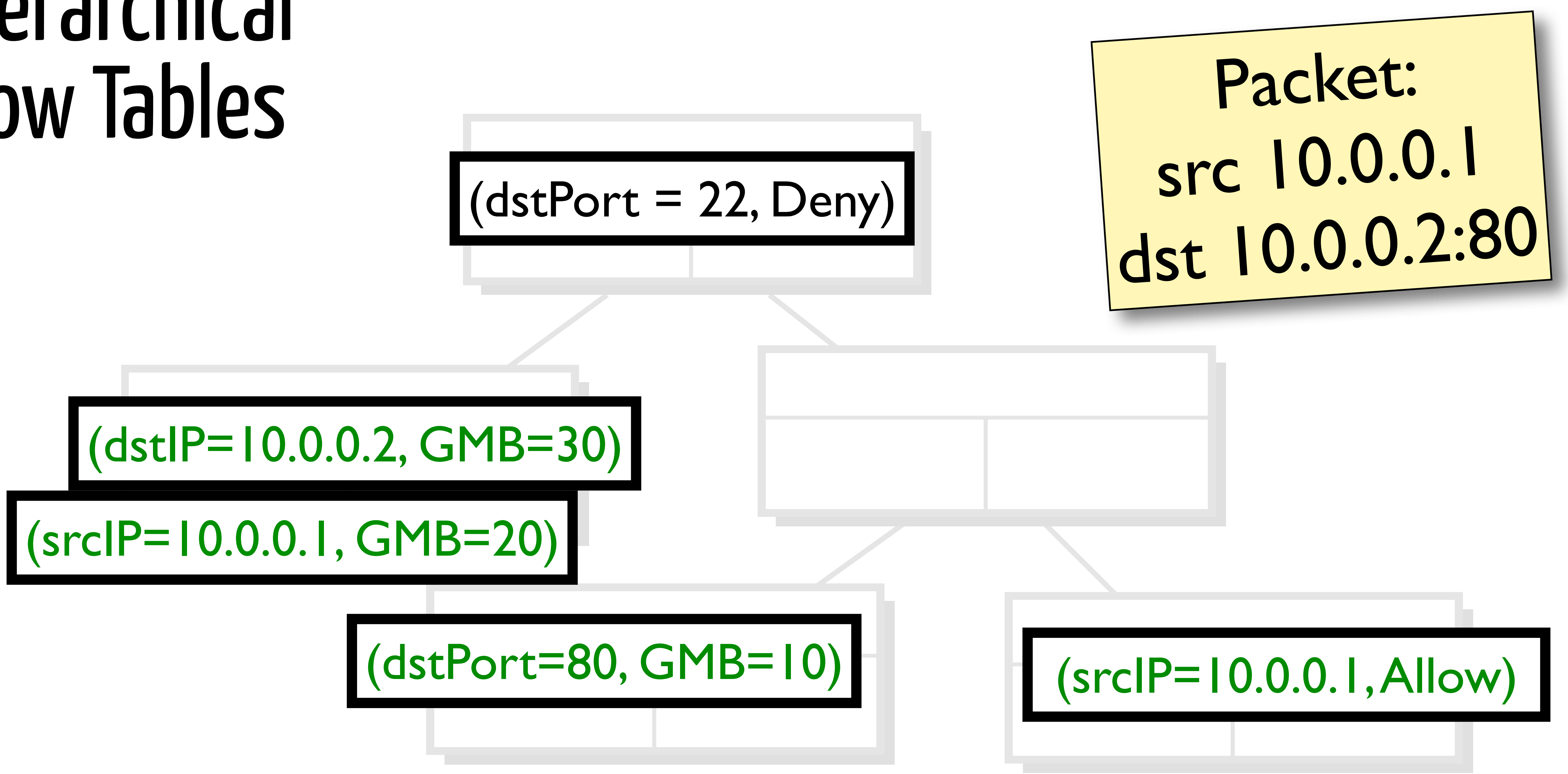


Policy Trees

Following the Ethane model, we imagine every packet is processed against a global policy by the central controller.

Here, packet processing is the result of evaluating each packet using the current policy tree.

Hierarchical Flow Tables



Packet Evaluation

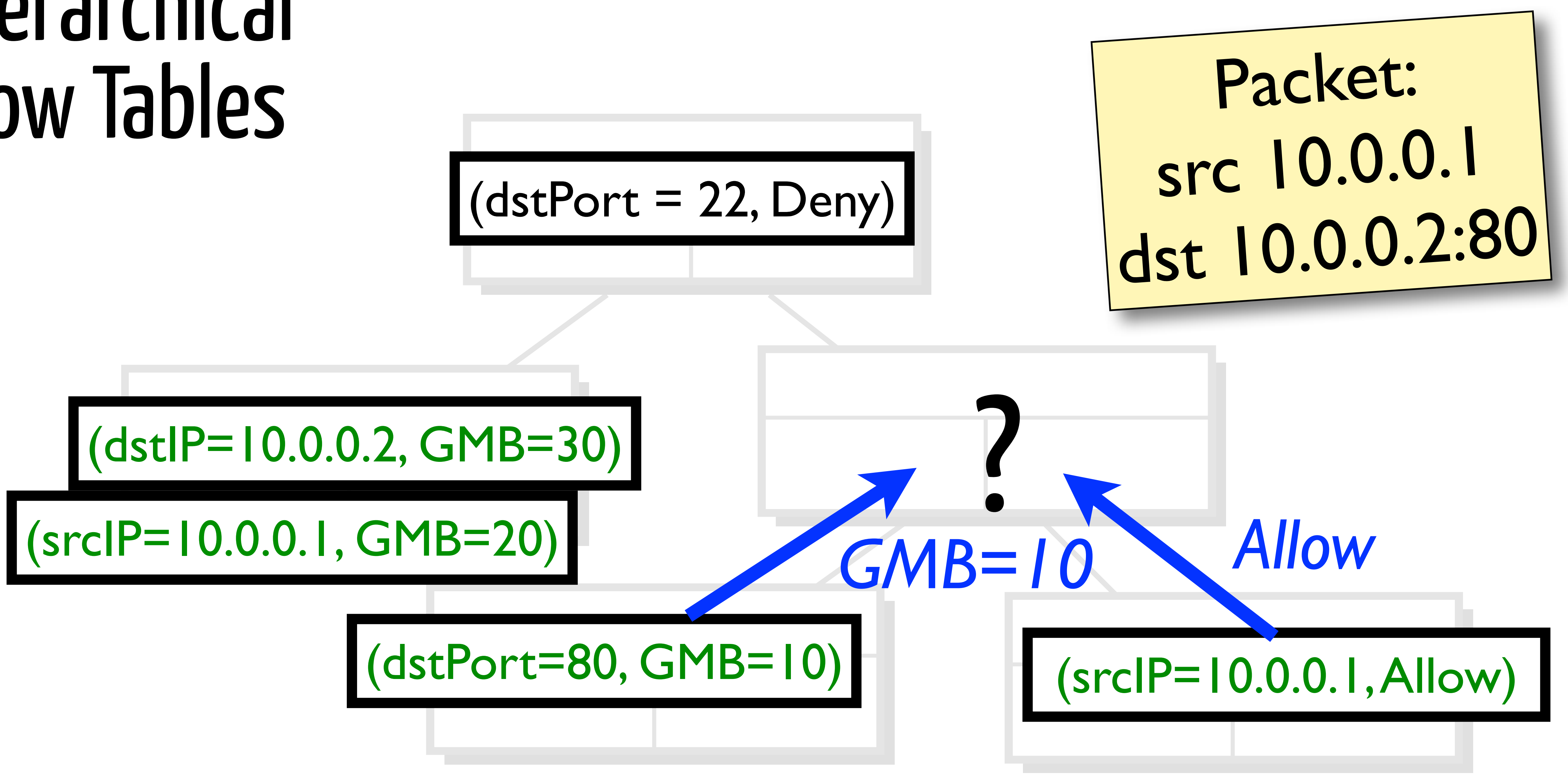
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children's action with the parent's using a parent operator. Note that in this case, the parent did produce any action, which we denote by "0", a special "don't care" action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Packet Evaluation

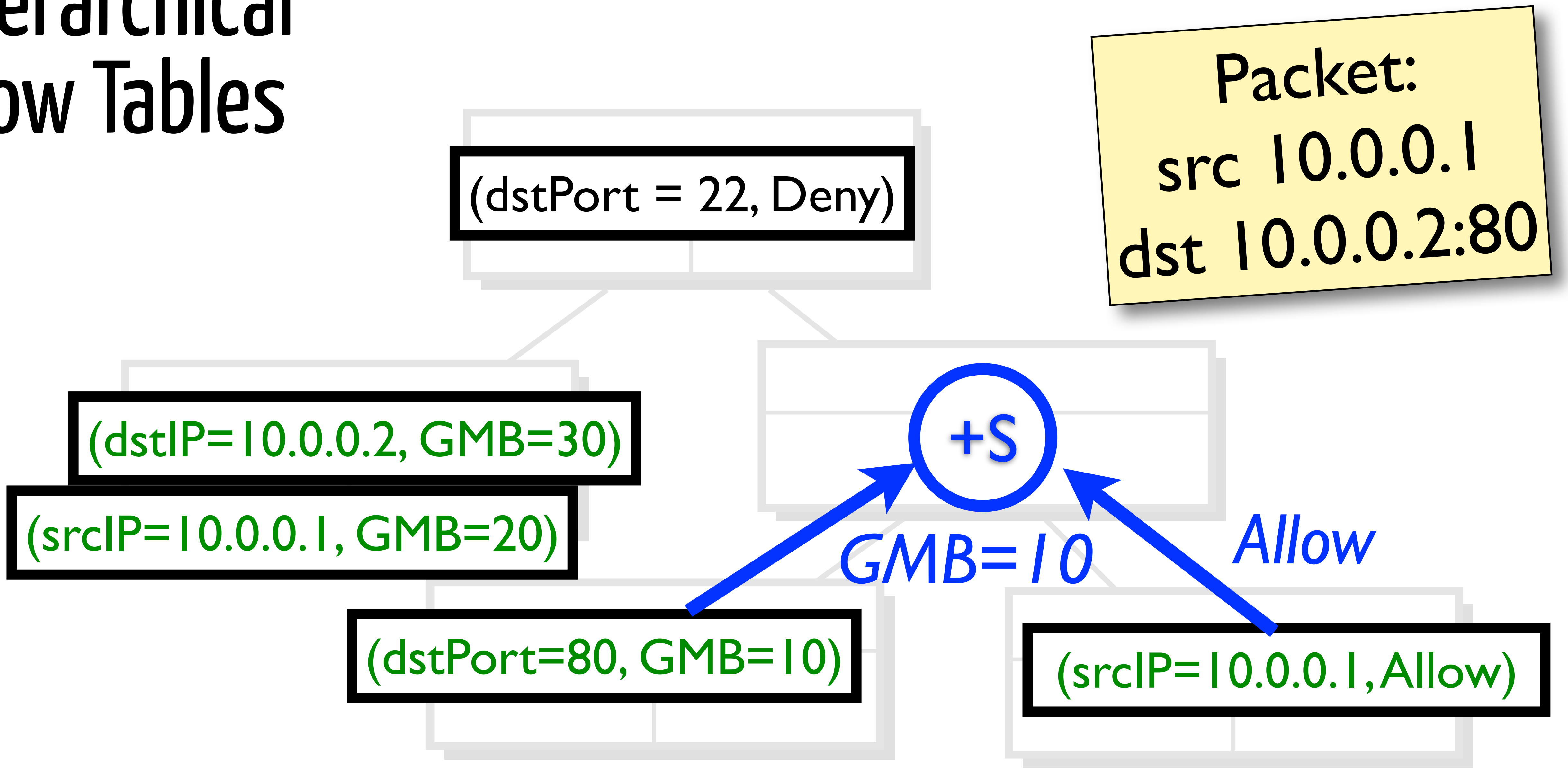
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children's action with the parent's using a parent operator. Note that in this case, the parent did produce any action, which we denote by "0", a special "don't care" action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Packet Evaluation

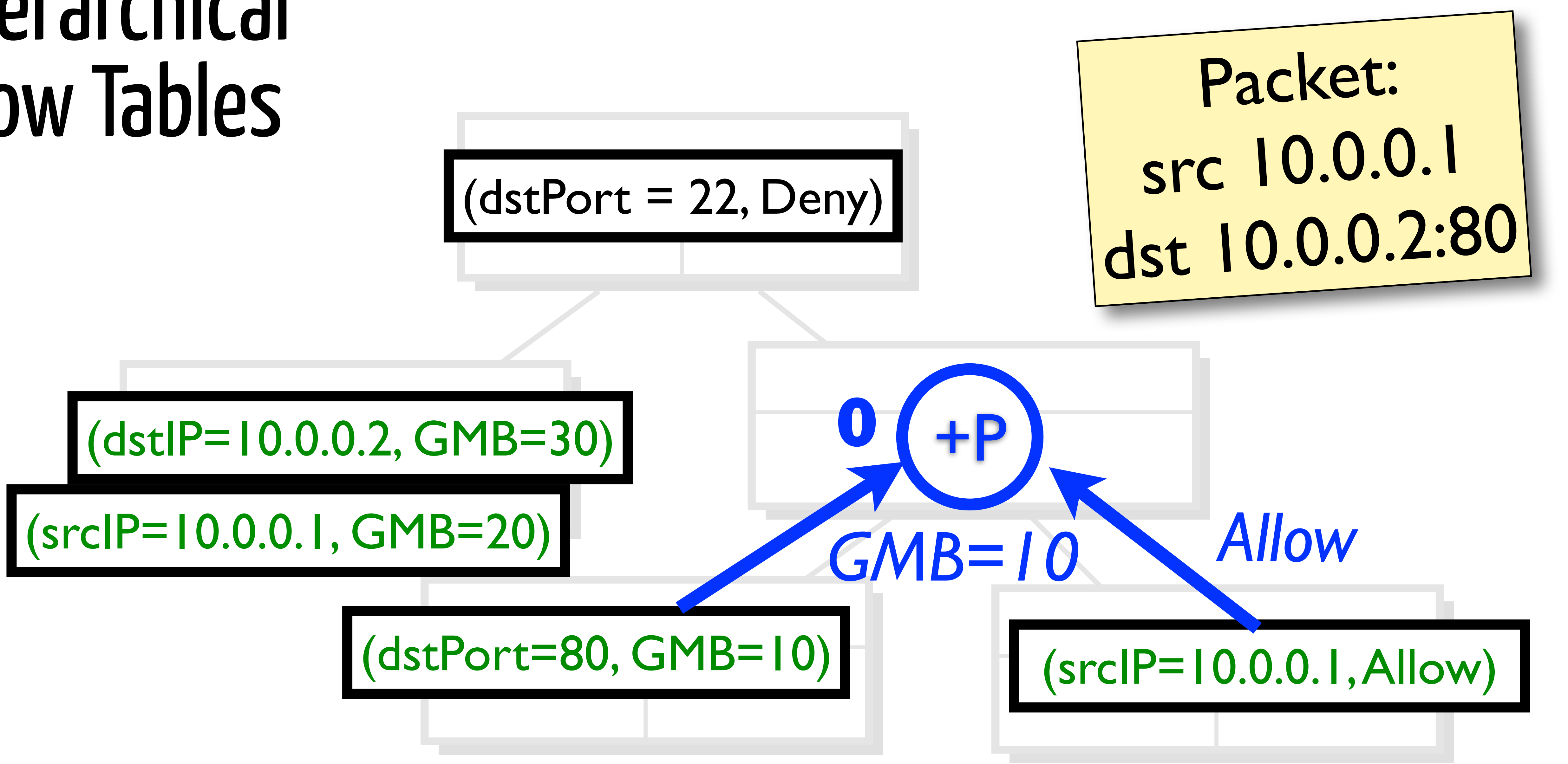
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children's action with the parent's using a parent operator. Note that in this case, the parent did produce any action, which we denote by "0", a special "don't care" action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Packet Evaluation

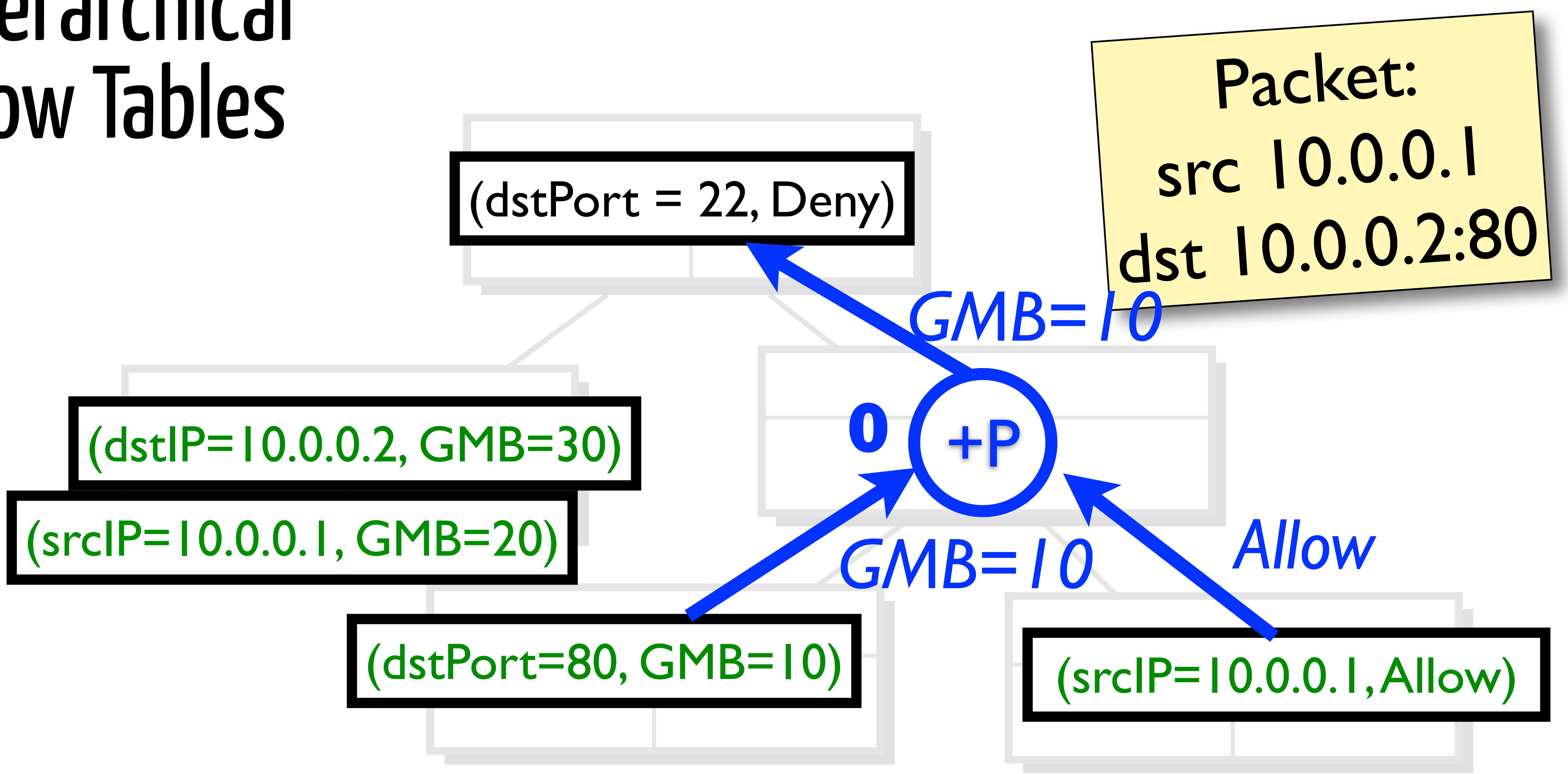
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children's action with the parent's using a parent operator. Note that in this case, the parent did produce any action, which we denote by "0", a special "don't care" action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Packet Evaluation

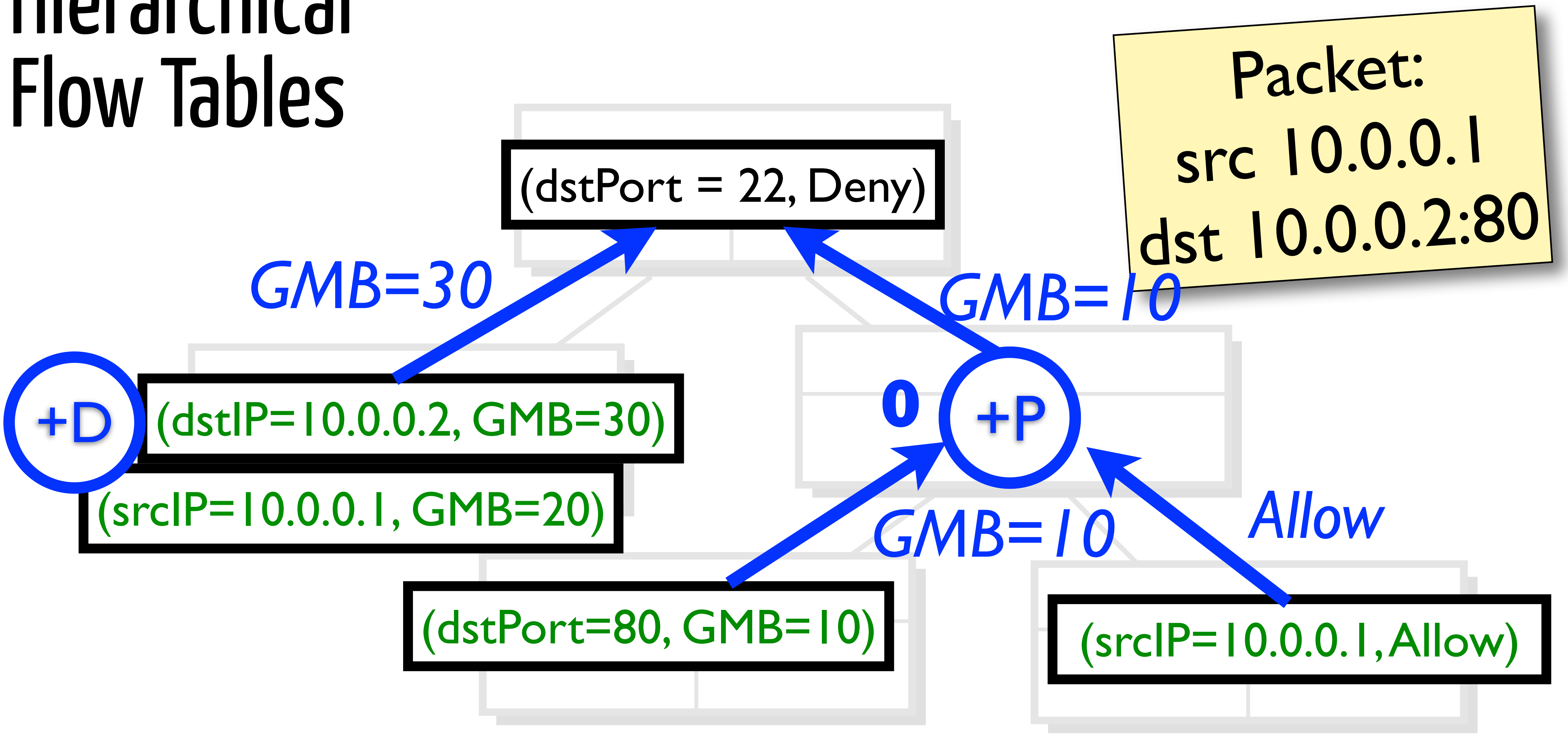
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children's action with the parent's using a parent operator. Note that in this case, the parent did produce any action, which we denote by "0", a special "don't care" action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Packet Evaluation

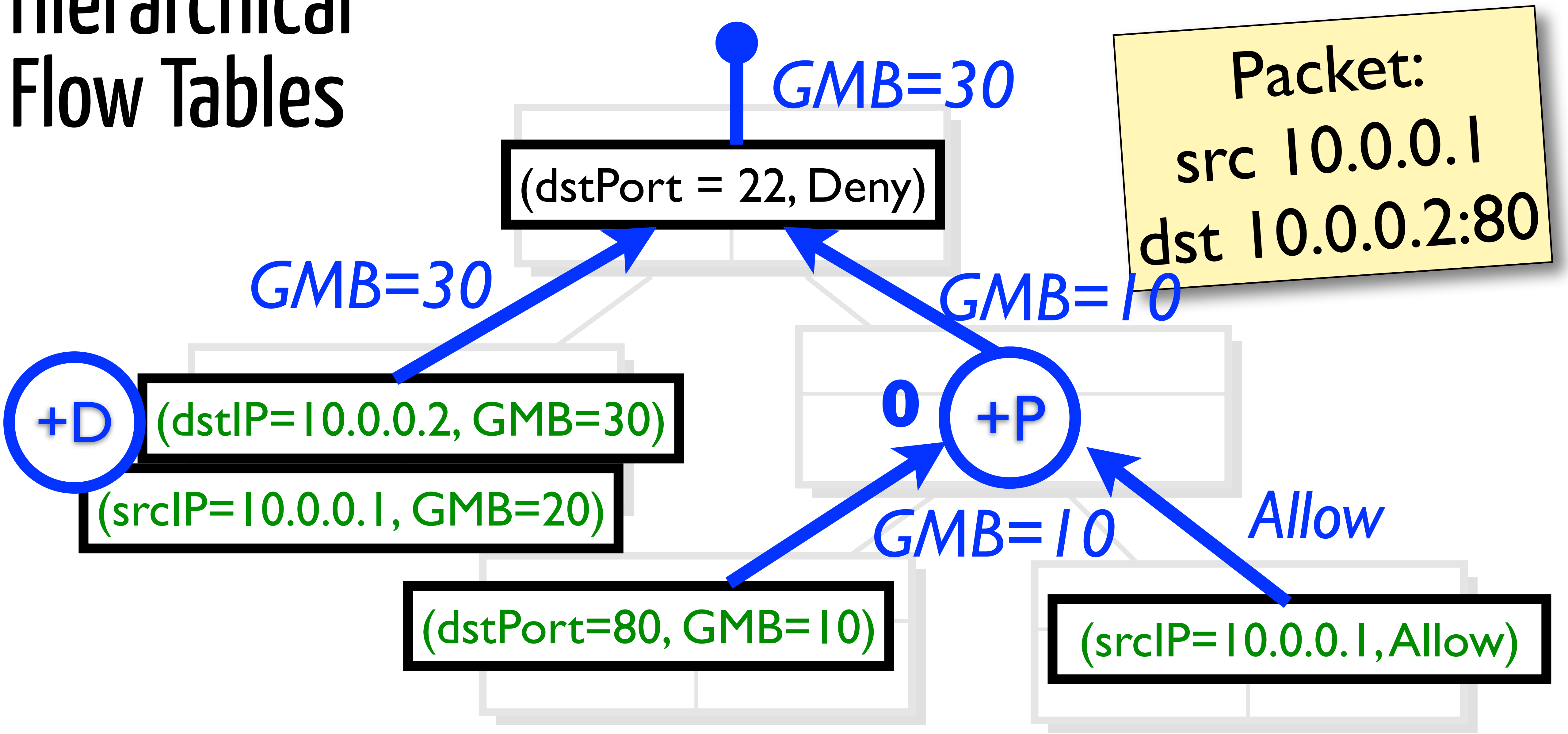
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children’s action with the parent’s using a parent operator. Note that in this case, the parent did produce any action, which we denote by “0”, a special “don’t care” action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Packet Evaluation

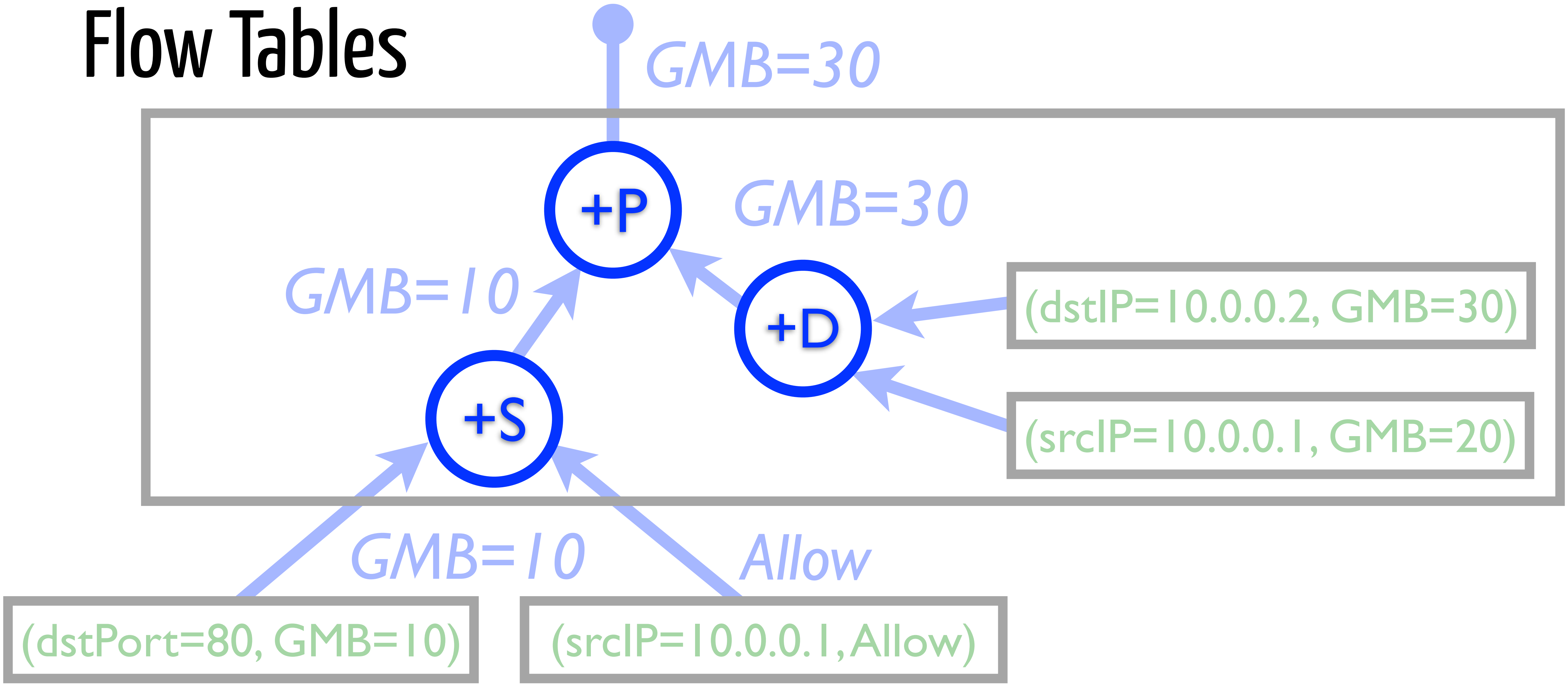
First, we identify the matching policy atoms, shown here in green. Next, policy atoms emit their actions.

(click) When multiple subtrees have produced actions, we apply user-defined operators (click) at each node in the tree to combine the actions. Here, the sibling operator was applied.

(click) Next, we combine the children's action with the parent's using a parent operator. Note that in this case, the parent did produce any action, which we denote by "0", a special "don't care" action.

(continue until GMB=30 is emitted)

Hierarchical Flow Tables



Conflict Resolution

Participatory networking uses three combination operators within each node to resolve conflicts.

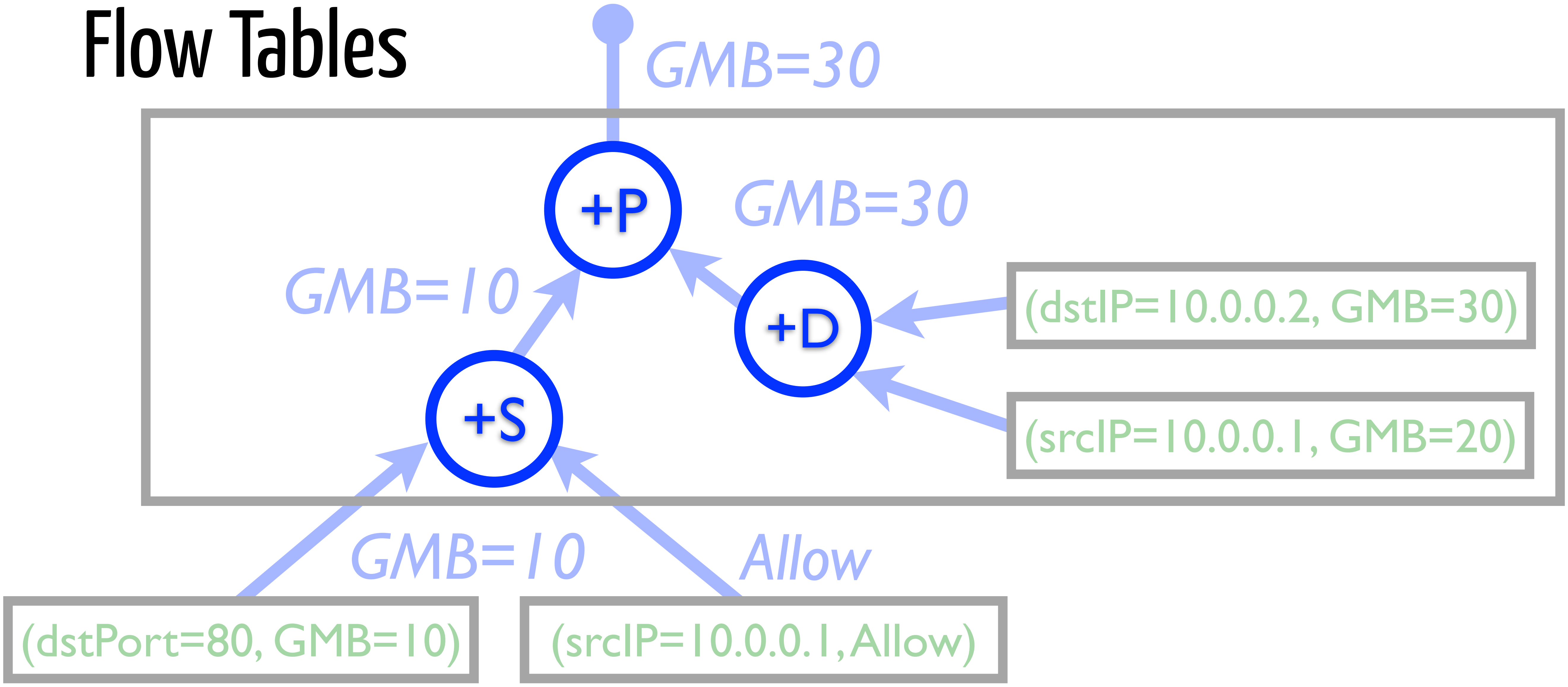
The first is the +S operator, which combines sibling actions.
The second is the +D operator, which combines multiple actions inside a single node.
Finally, the +P operator combines the previously resolved actions of a parent and child.

(Pause)

The requirements on these operators are very basic:
(click) first, they must be associative -- this allows us to resolve conflicts in a pairwise fashion. And second, they must support the 0 or “don’t care” action as their identity value.

With these minimal requirements, we can convert the HFT into an efficient implementation.

Hierarchical Flow Tables



Conflict Resolution

Only Requirements:
Associative, **0**-identity

Participatory networking uses three combination operators within each node to resolve conflicts.

The first is the +S operator, which combines sibling actions.
The second is the +D operator, which combines multiple actions inside a single node.
Finally, the +P operator combines the previously resolved actions of a parent and child.

(Pause)

The requirements on these operators are very basic:
(click) first, they must be associative -- this allows us to resolve conflicts in a pairwise fashion. And second, they must support the 0 or “don’t care” action as their identity value.

With these minimal requirements, we can convert the HFT into an efficient implementation.

PANE’s Conflict Resolution Operators

+D *In node*

+S *Sibling*

D and S identical.
Deny overrides Allow.
GMB combines as **max**
Rate-limit combines as **min**

+P *Parent-Sibling*

Child overrides Parent
for Access Control
GMB combines as **max**
Rate-limit combines as **min**

The conflict resolution operators’ flexibility creates a design space for our system.

This slide is a summary of the choices we made for PANE. When sensible, we strive to combine both requests. For example a request to guarantee a minimum bandwidth, can combine with one that limits below a maximum rate.

In other cases, we need a single outcome. PANE’s +D and +S operators implement a basic policy in which Deny requests override Allow requests, take the maximum of two bandwidth guarantees, and the minimum of two rate-limits. With the +P operator, PANE allows access control requests in child shares to override those in parent shares.

The HFT itself is agnostic to the specific policies of the operators, as long as they satisfy the identity and associativity requirements. For example, we could develop operators that resolve conflicts according to priority.

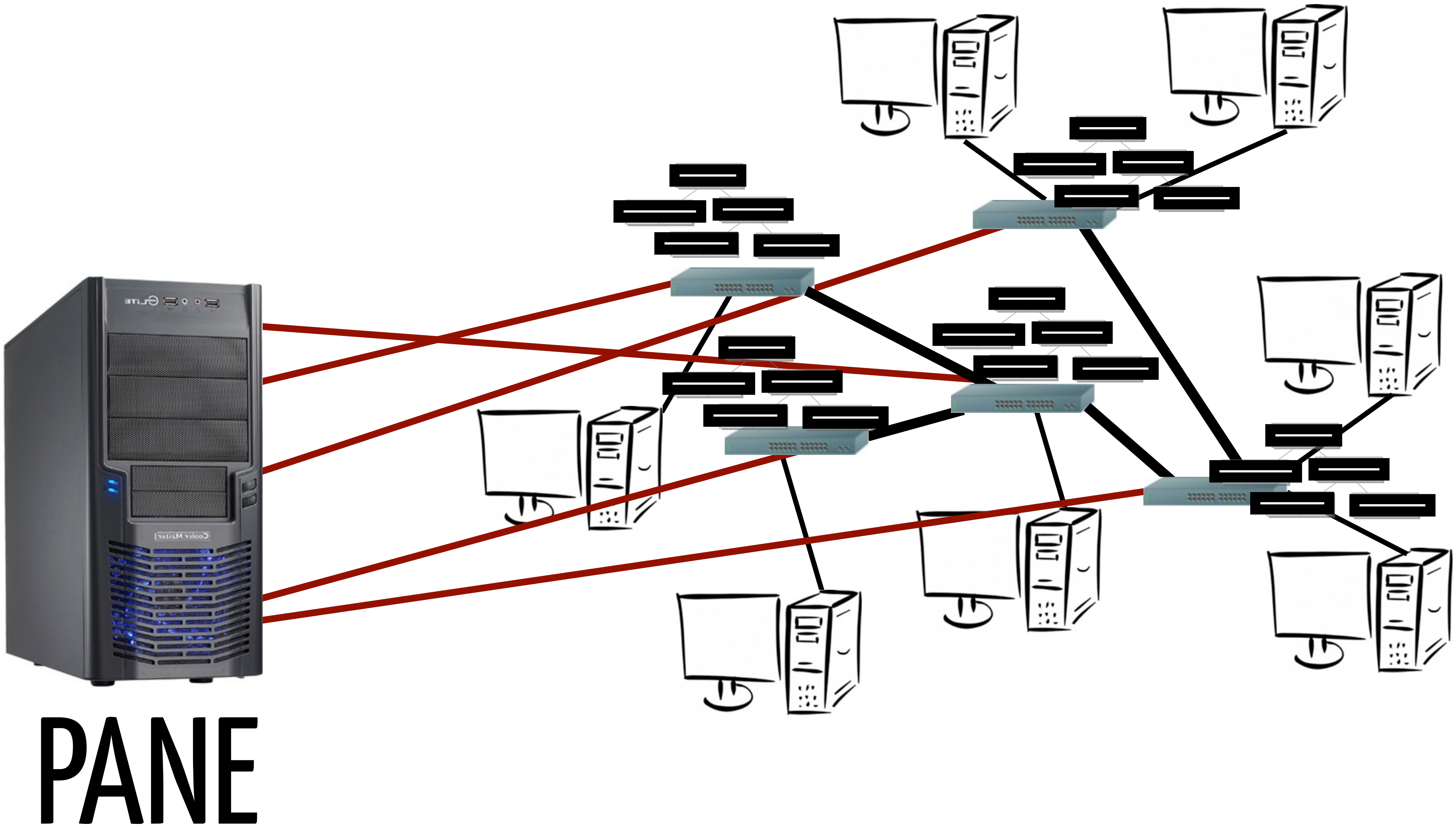
(Pause)

Implementation

So, how do we implement this system?

(pause)

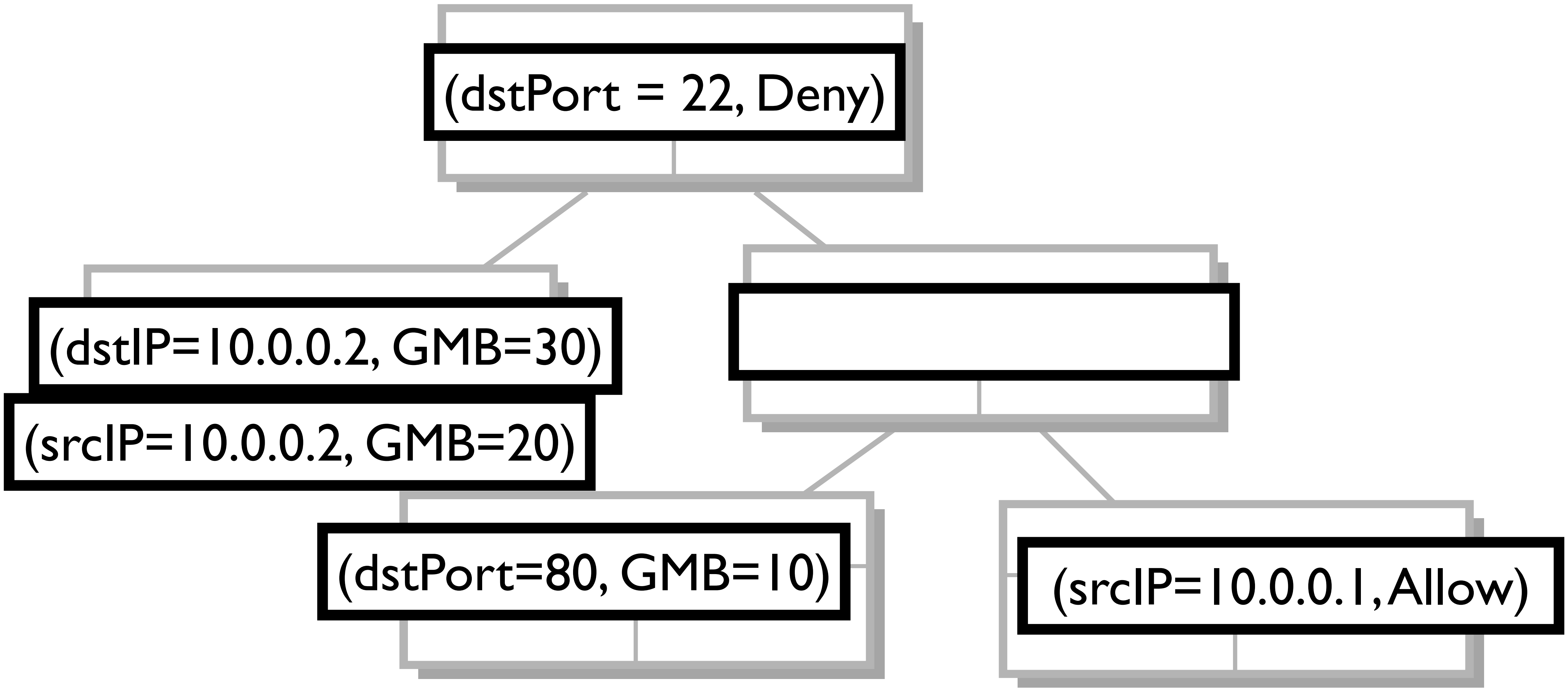
In an ideal world, we could simply pass each new HFT to the switches...



… and when packets arrive, the switches would evaluate the tree just as we did on the previous slides.

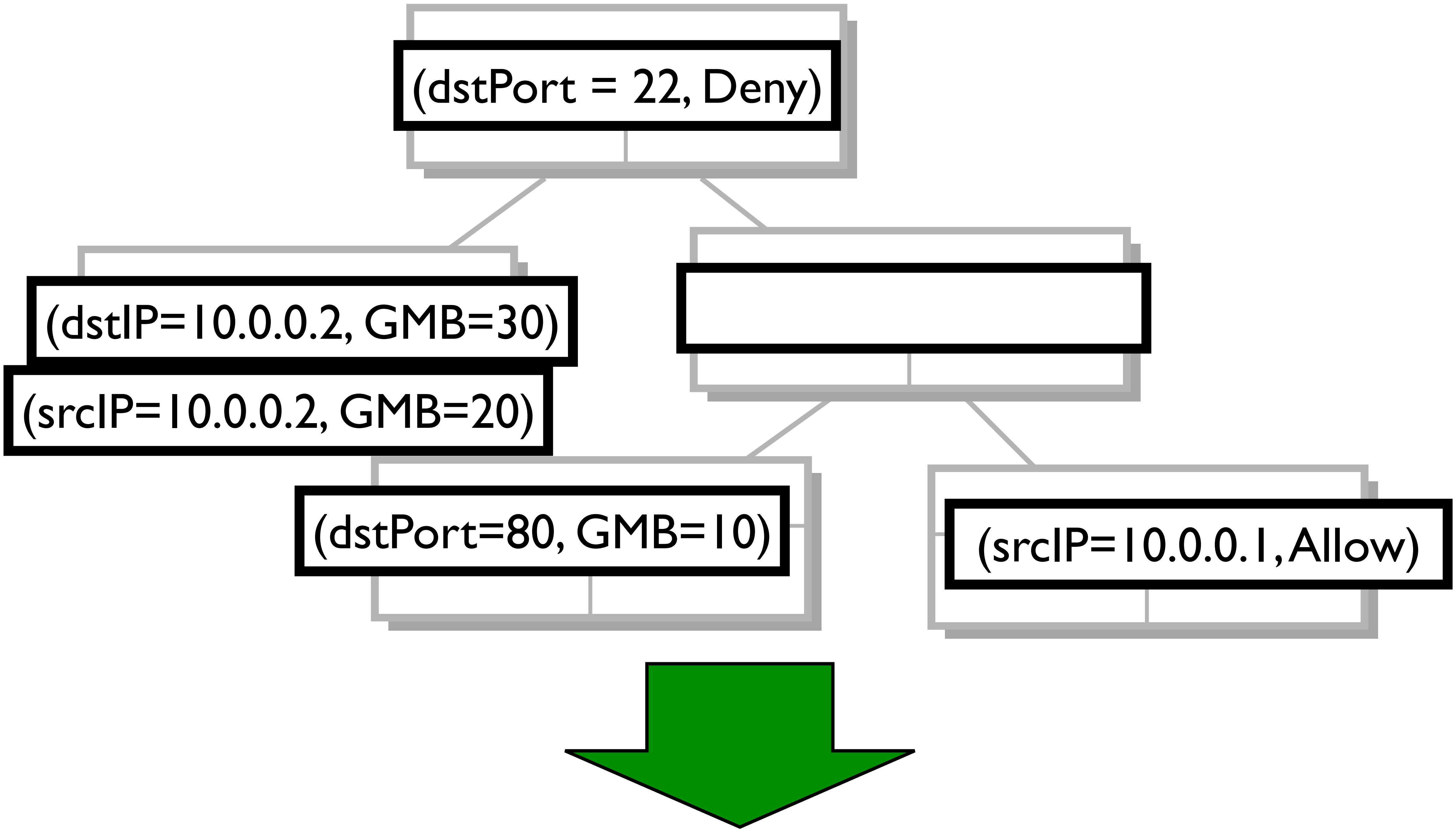
However, today’s switches aren’t capable performing this evaluation.

Therefore, rather than send every packet to the controller …



… we developed a compiler which linearizes

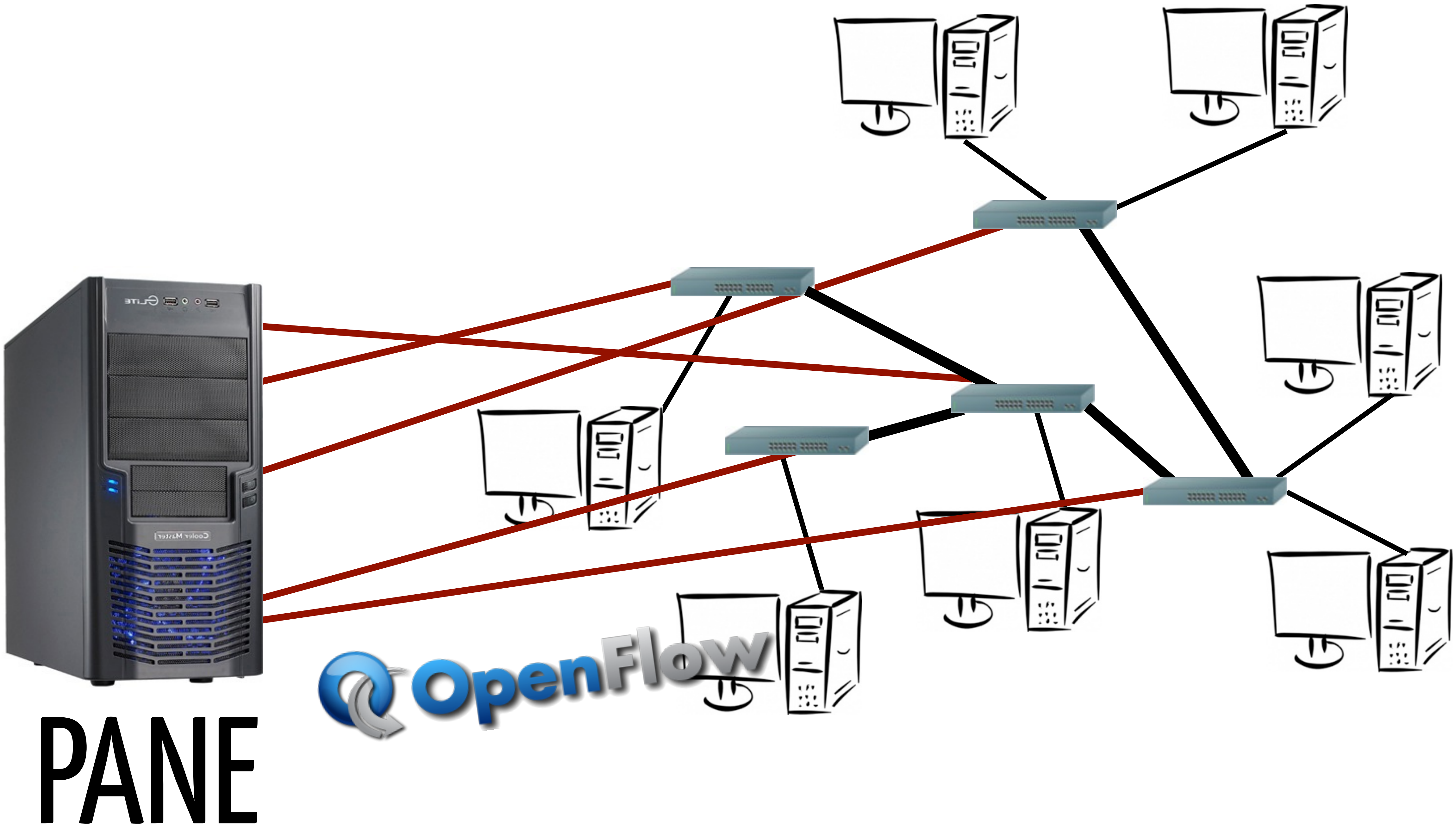
(click) an HFT instance into traditional, flat OpenFlow tables that are collectively equivalent to the logical policy tree. This compilation process is quadratic in the size of the tree, as we explain in our paper.



Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6
*	*	*	*	*	*	*	*	*	22	drop

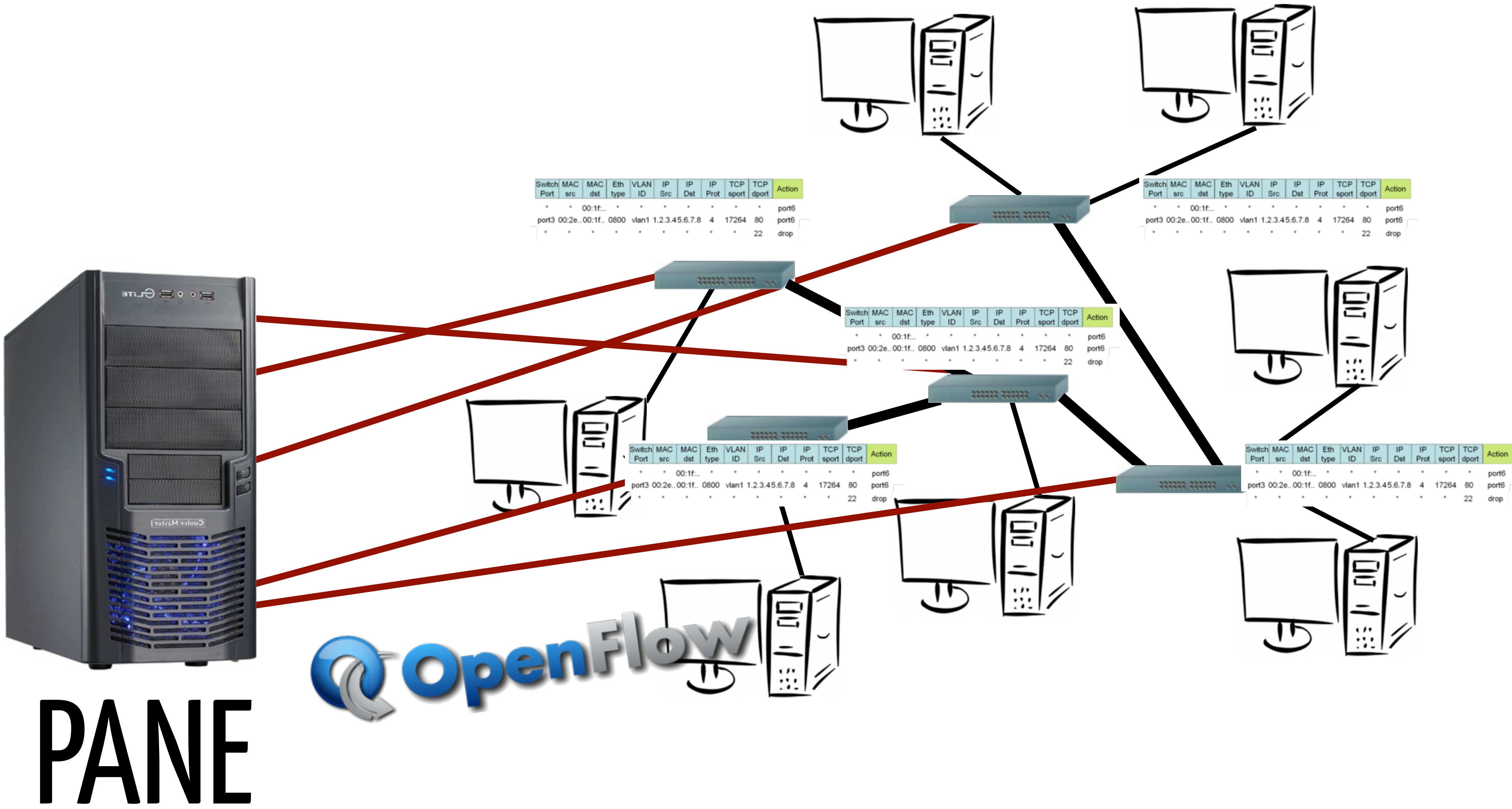
… we developed a compiler which linearizes

(click) an HFT instance into traditional, flat OpenFlow tables that are collectively equivalent to the logical policy tree. This compilation process is quadratic in the size of the tree, as we explain in our paper.



Our OpenFlow controller then installs these tables on the switches

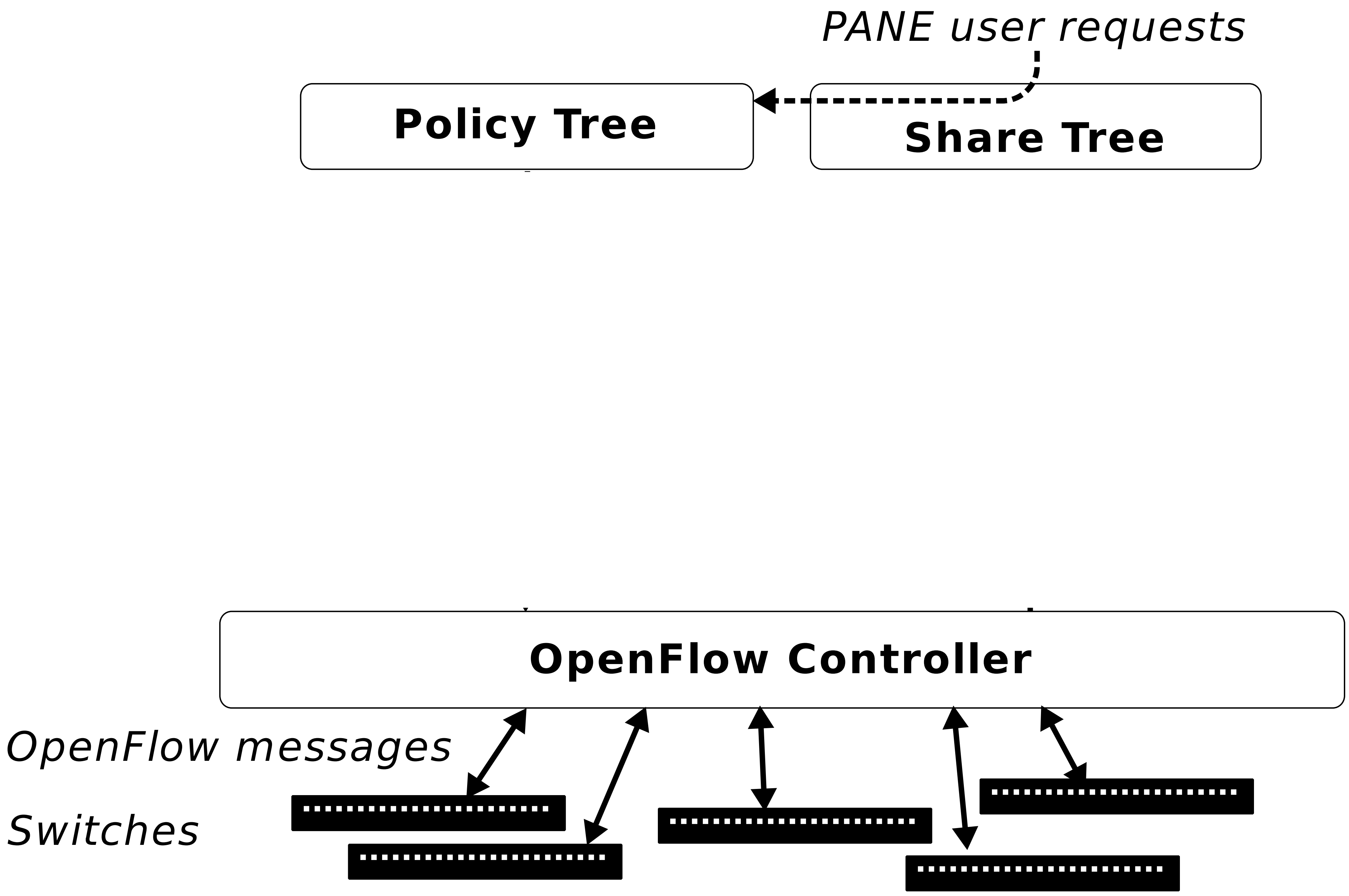
(click), allowing the network to implement the HFT with hardware support.



Our OpenFlow controller then installs these tables on the switches

(click), allowing the network to implement the HFT with hardware support.

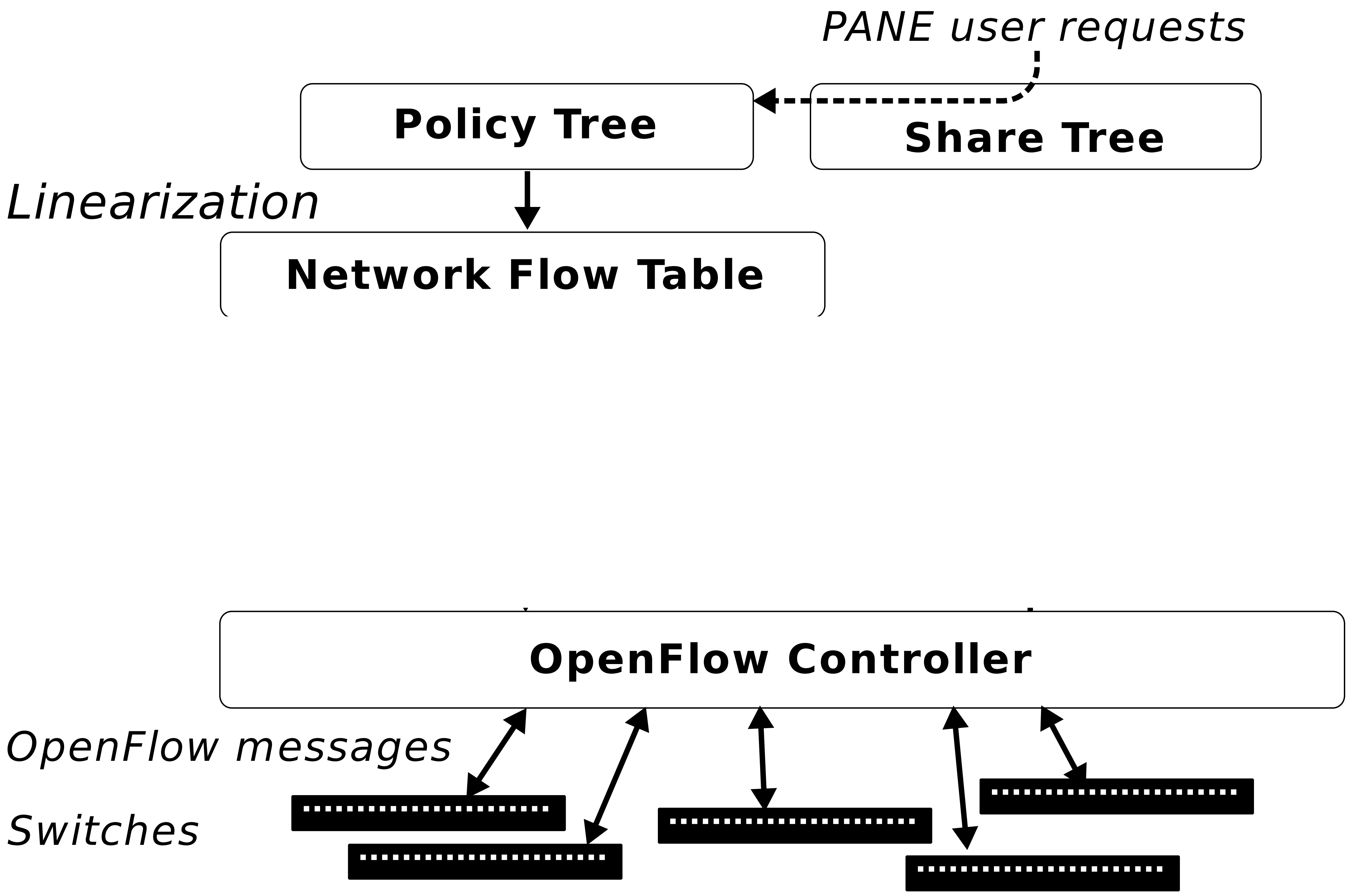
PANE



Our compiler works in two stages.

(click) First, the compiler linearizes the HFT into a single table we call the "network flow table." If the network were connected by a single, big switch, we might install this network flow table directly onto that switch

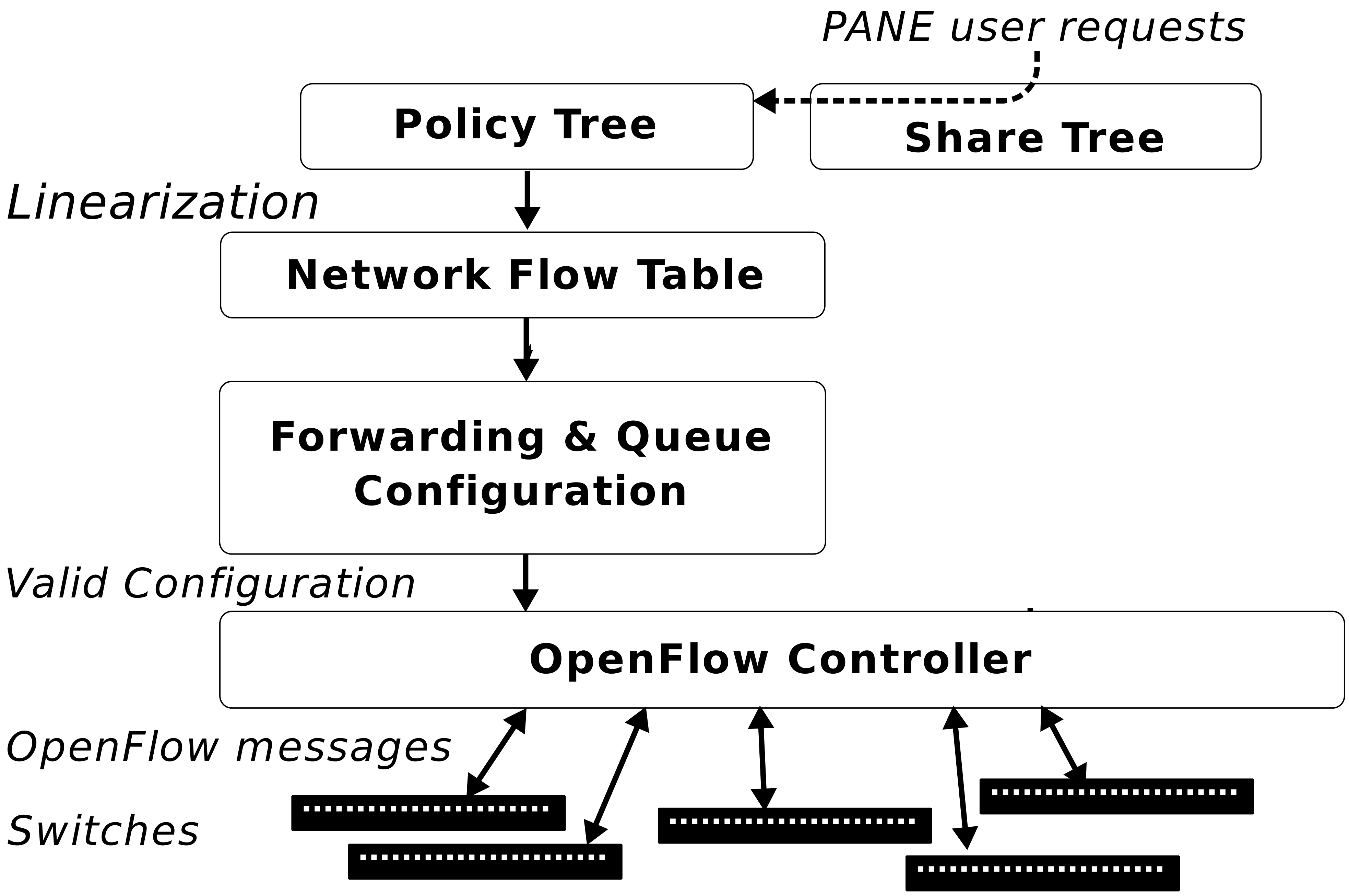
PANE



Our compiler works in two stages.

(click) First, the compiler linearizes the HFT into a single table we call the "network flow table." If the network were connected by a single, big switch, we might install this network flow table directly onto that switch

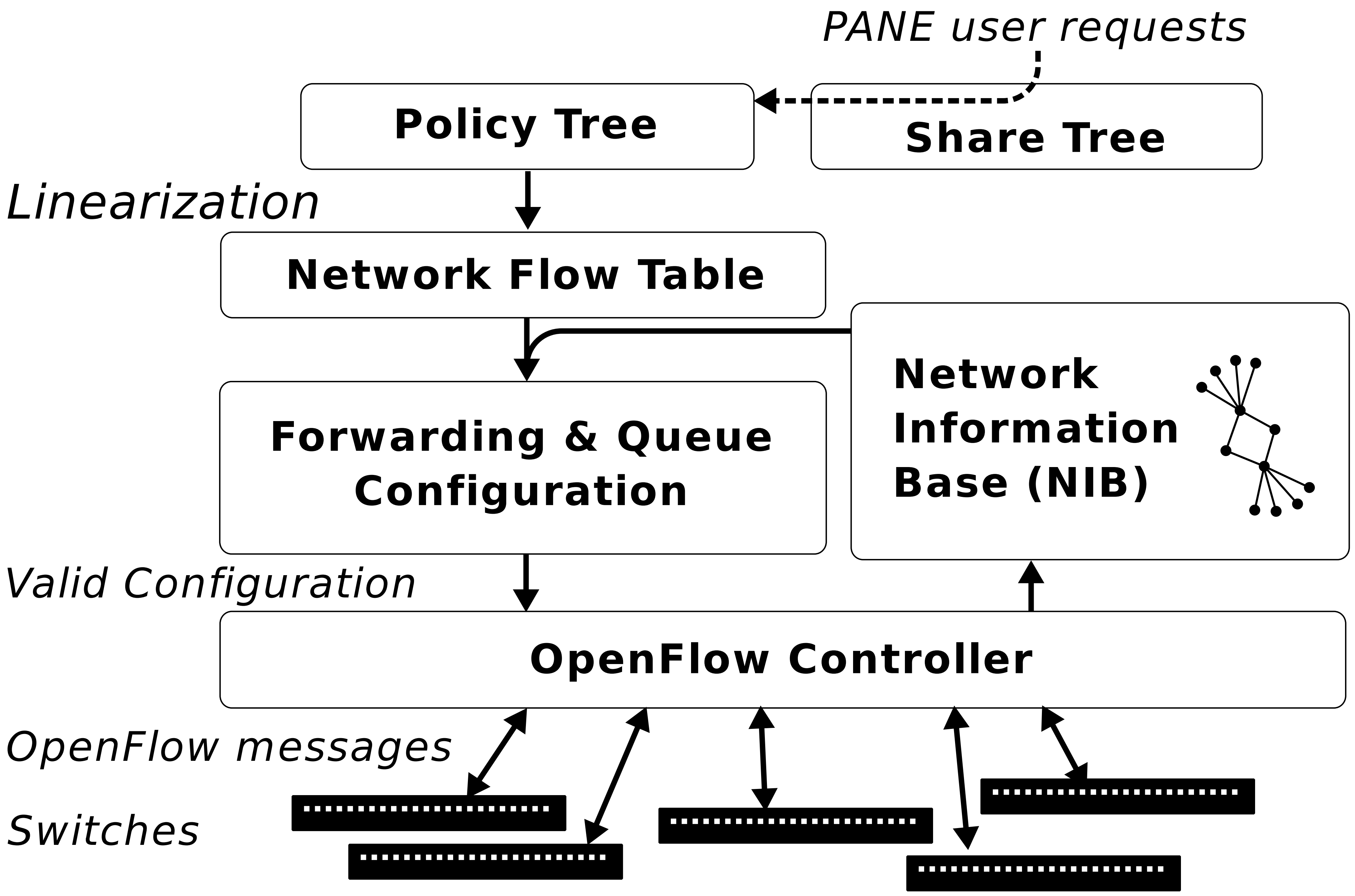
PANE



In the compiler's second stage, it translates the network flow table into individual flow tables for the distributed OpenFlow switches. During this stage,

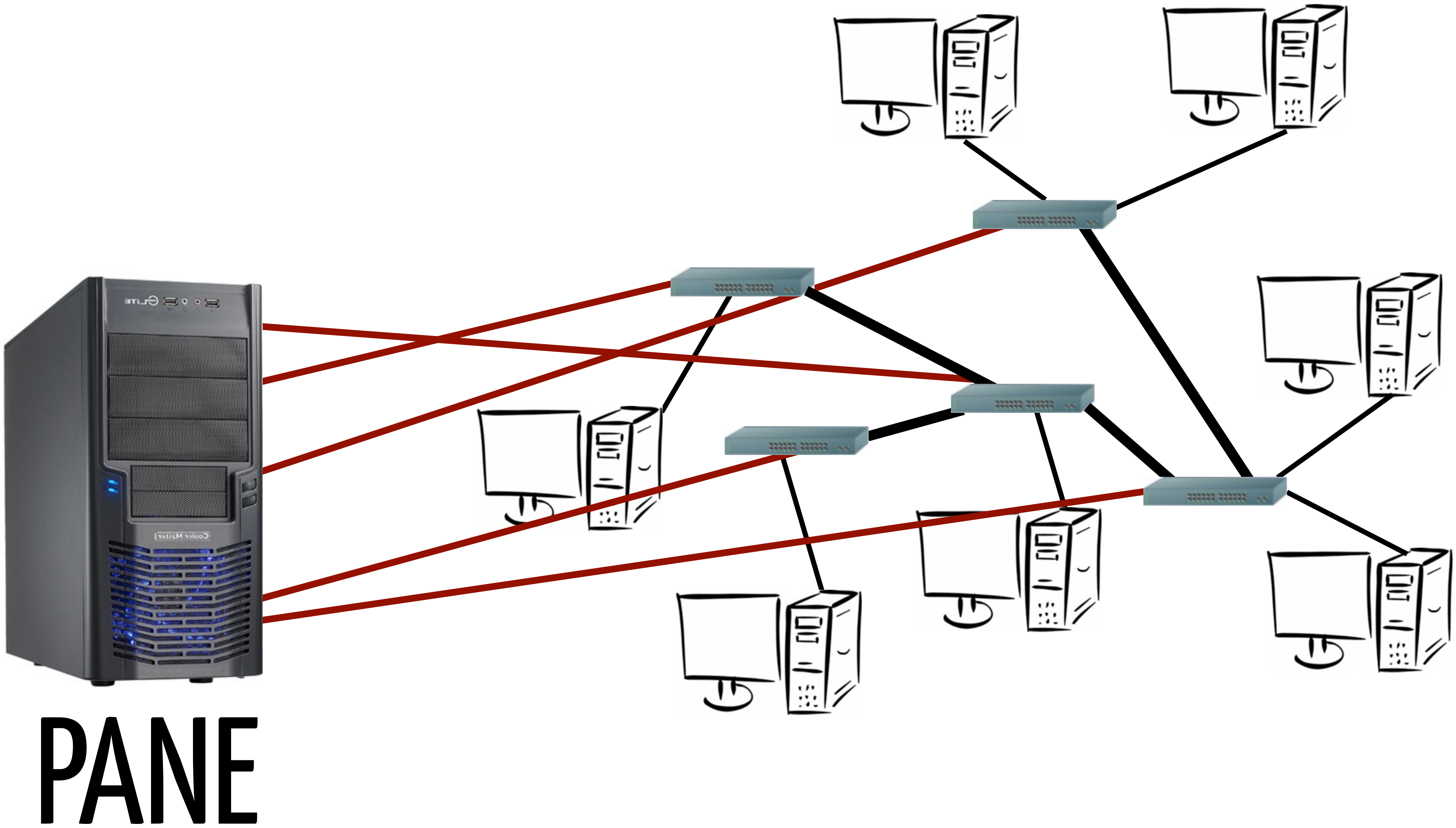
(click) the compiler relies on a Network Information Base or NIB. The design of our NIB is inspired by Onix, and it describes the state of the network, including host locations, link statuses, queue availability, switch configurations, and more.

PANE

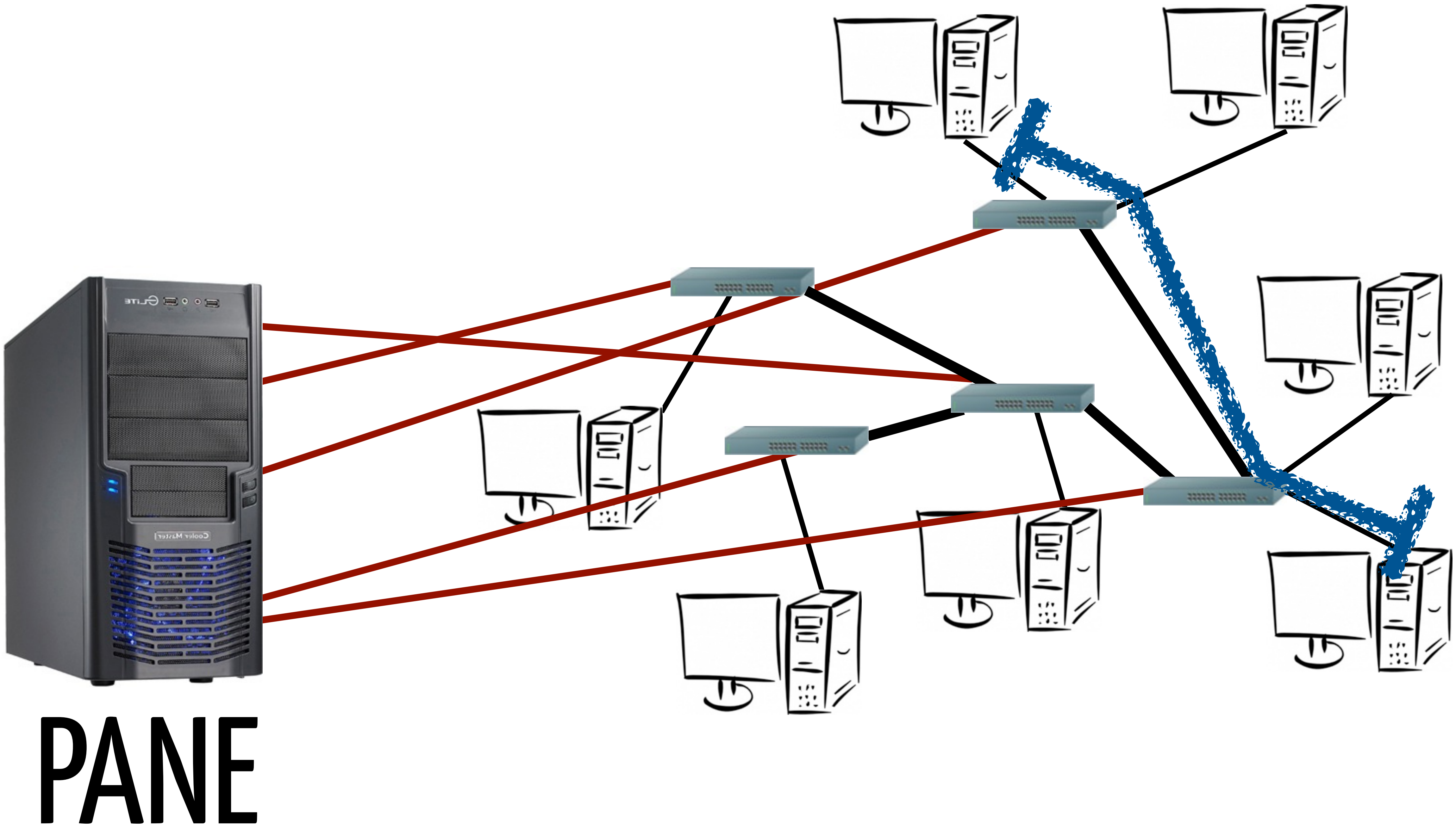


In the compiler's second stage, it translates the network flow table into individual flow tables for the distributed OpenFlow switches. During this stage,

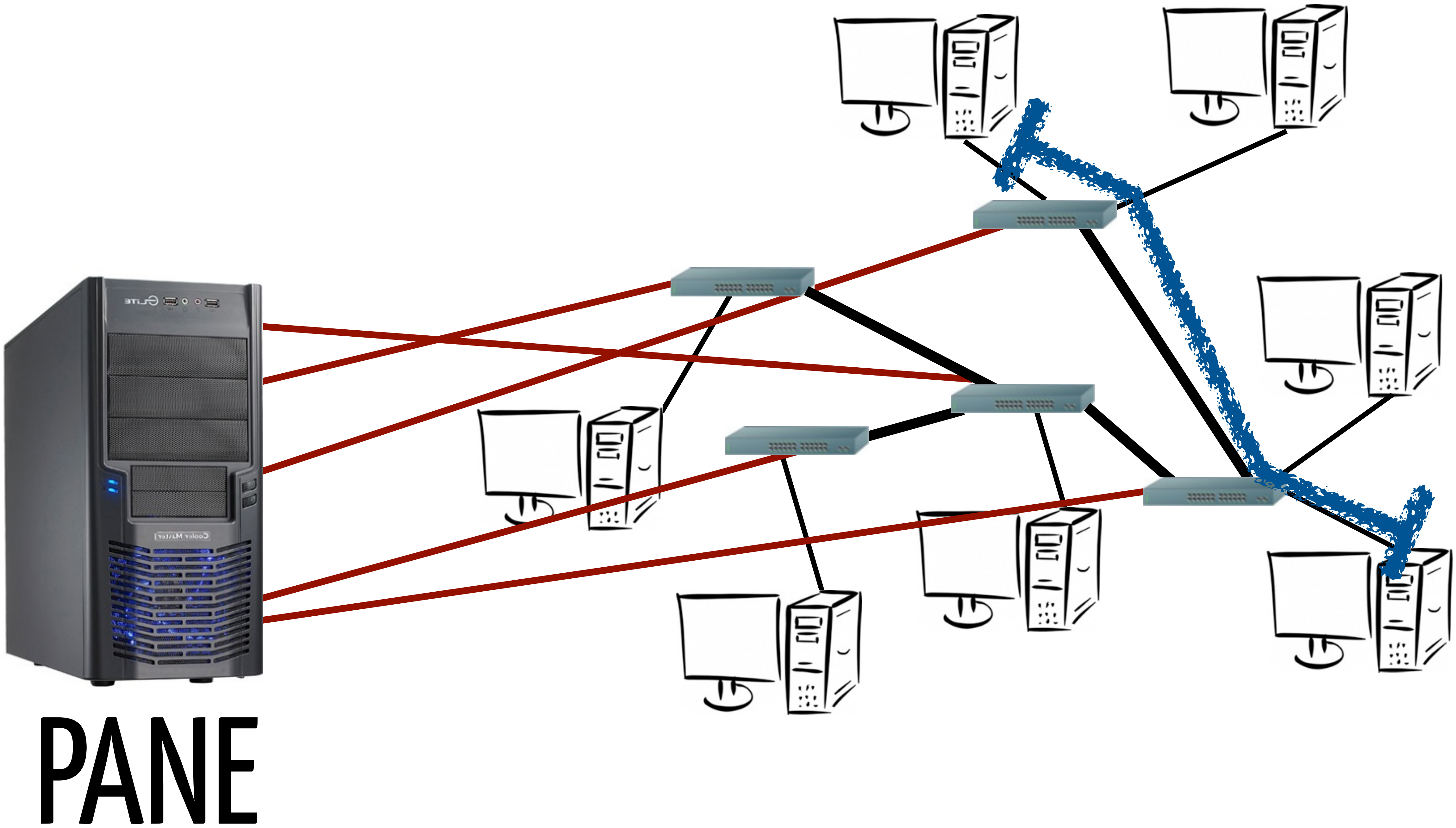
(click) the compiler relies on a Network Information Base or NIB. The design of our NIB is inspired by Onix, and it describes the state of the network, including host locations, link statuses, queue availability, switch configurations, and more.



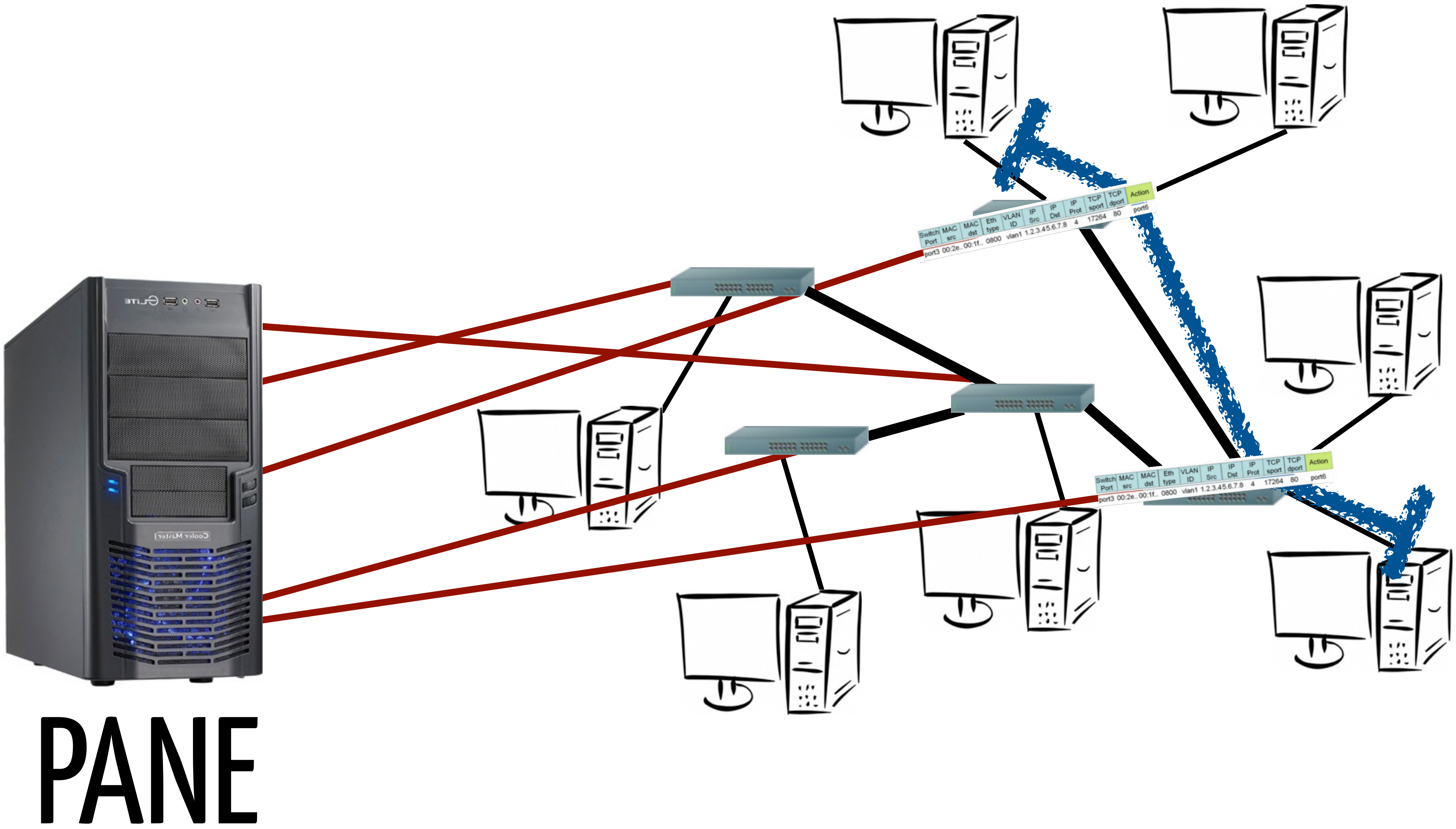
For example, the compiler uses the NIB to implement a bandwidth reservation by finding a circuit with the requested bandwidth,
(click) sending commands to create the necessary queues,
(click) and finally, updating the OpenFlow tables
(click) with the required forwarding decisions.



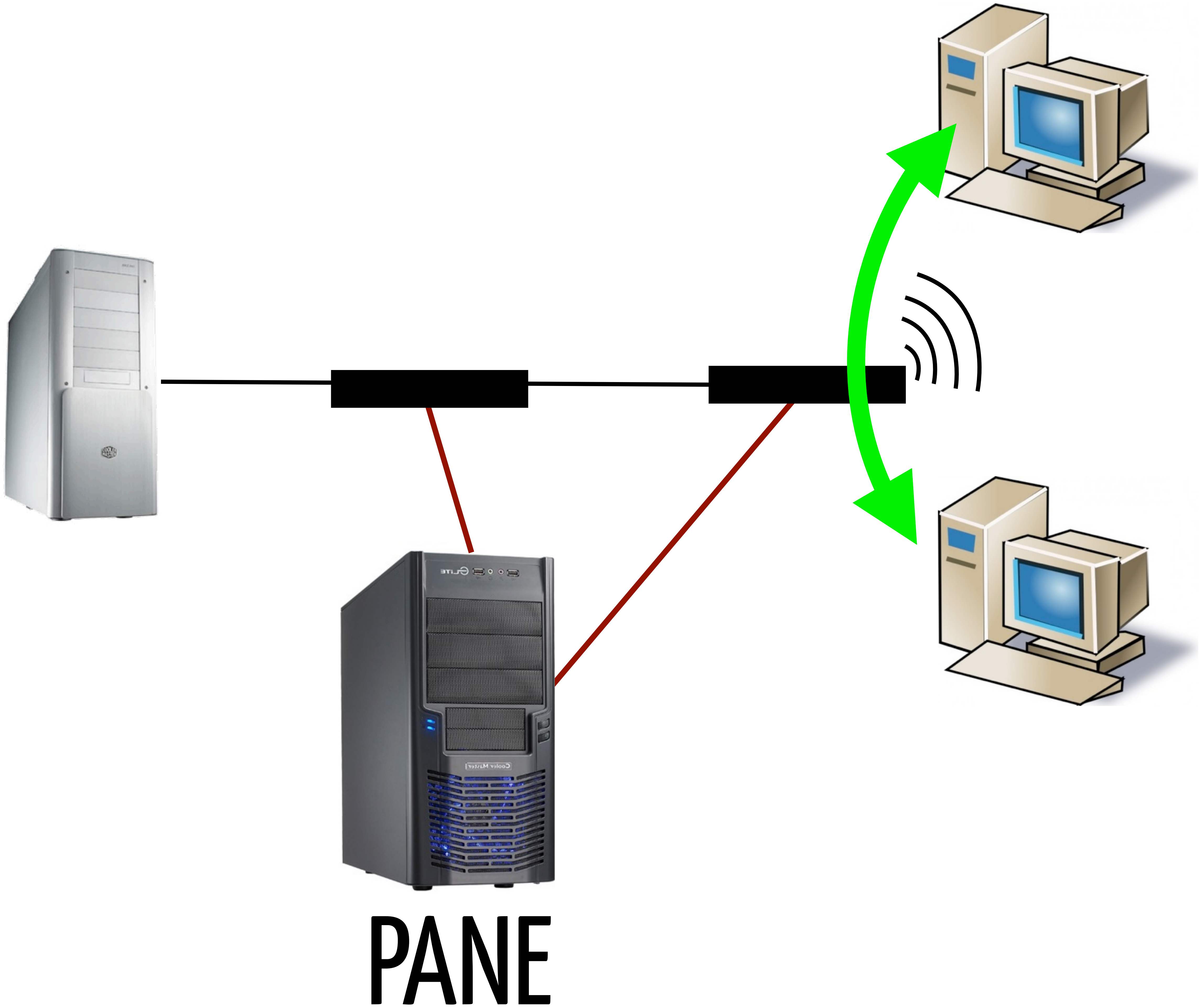
For example, the compiler uses the NIB to implement a bandwidth reservation by finding a circuit with the requested bandwidth,
(click) sending commands to create the necessary queues,
(click) and finally, updating the OpenFlow tables
(click) with the required forwarding decisions.



For example, the compiler uses the NIB to implement a bandwidth reservation by finding a circuit with the requested bandwidth,
(click) sending commands to create the necessary queues,
(click) and finally, updating the OpenFlow tables
(click) with the required forwarding decisions.



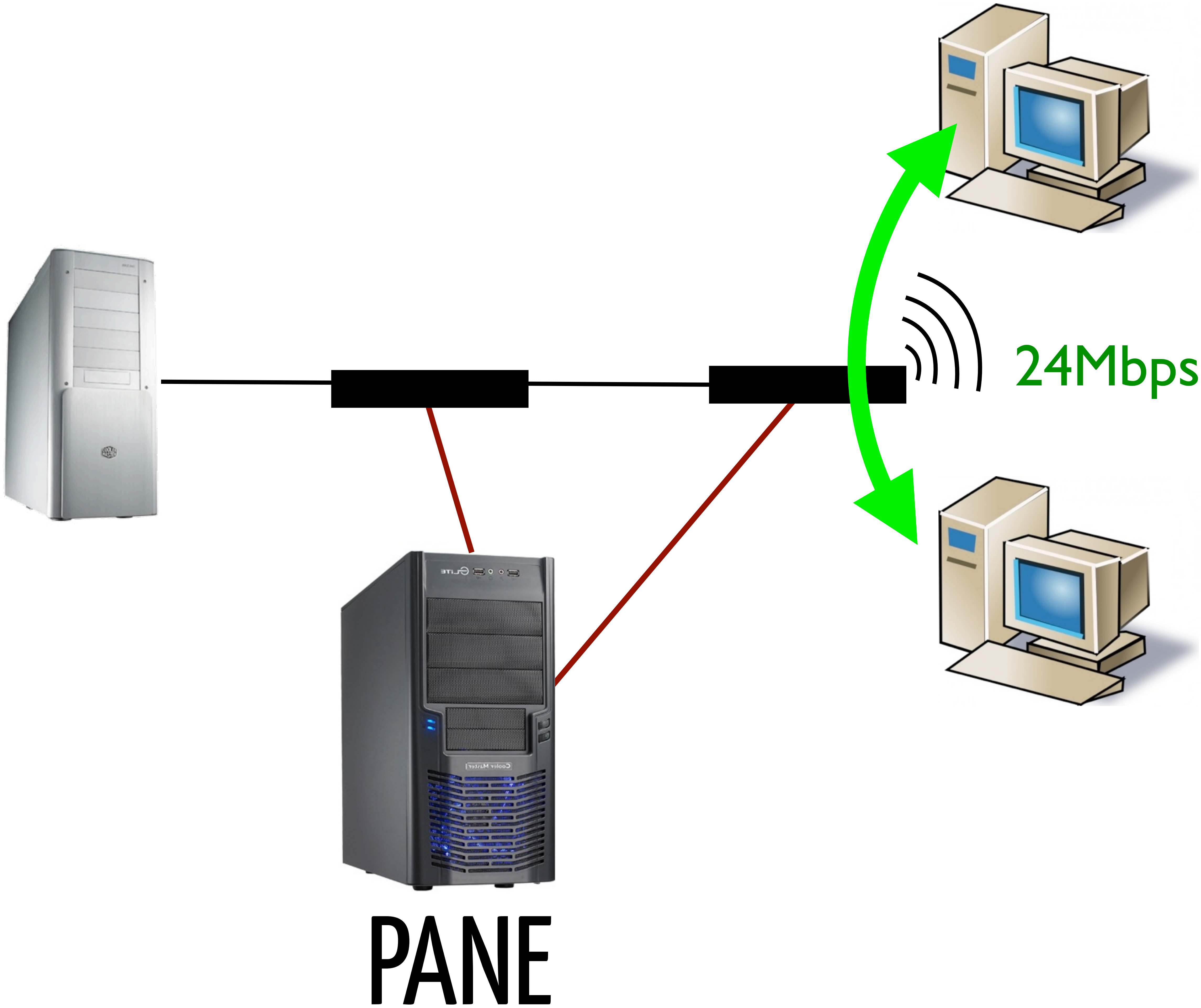
For example, the compiler uses the NIB to implement a bandwidth reservation by finding a circuit with the requested bandwidth,
(click) sending commands to create the necessary queues,
(click) and finally, updating the OpenFlow tables
(click) with the required forwarding decisions.



The NIB also allows PANE’s compiler to choose where in the network to implement desired policies.

As a simple example, it places rules which drop traffic as close as possible to the traffic's ingress port.
In this experiment, we have two wireless clients communicating. One suffers from an attack,
(click) and the transfer rate drops. With a local firewall rule,
(click) the transfer only slightly recovers. Using PANE to install the rule,
(click) the transfer fully recovers.

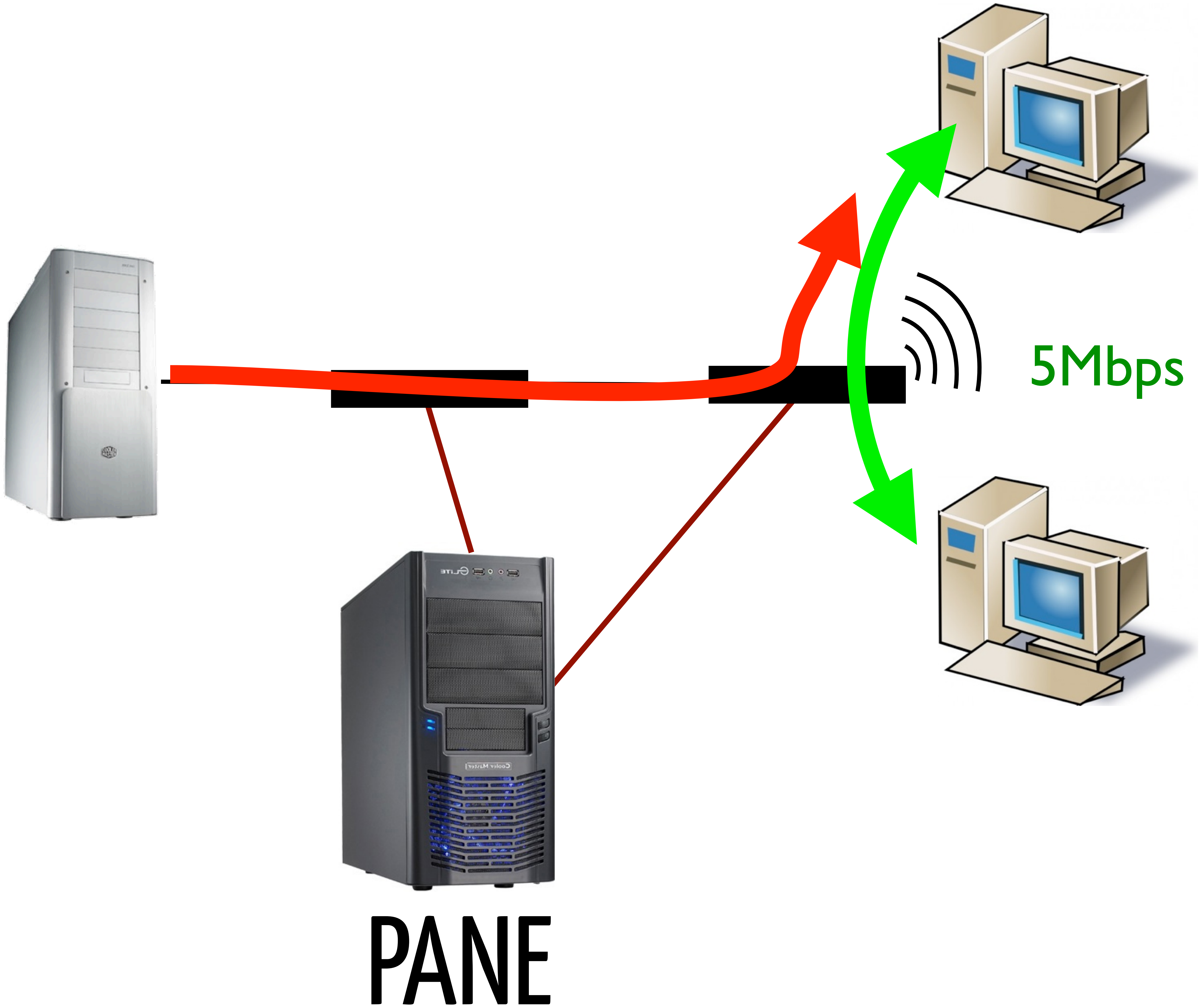
And as the source of the traffic moves ...



The NIB also allows PANE’s compiler to choose where in the network to implement desired policies.

As a simple example, it places rules which drop traffic as close as possible to the traffic's ingress port.
In this experiment, we have two wireless clients communicating. One suffers from an attack,
(click) and the transfer rate drops. With a local firewall rule,
(click) the transfer only slightly recovers. Using PANE to install the rule,
(click) the transfer fully recovers.

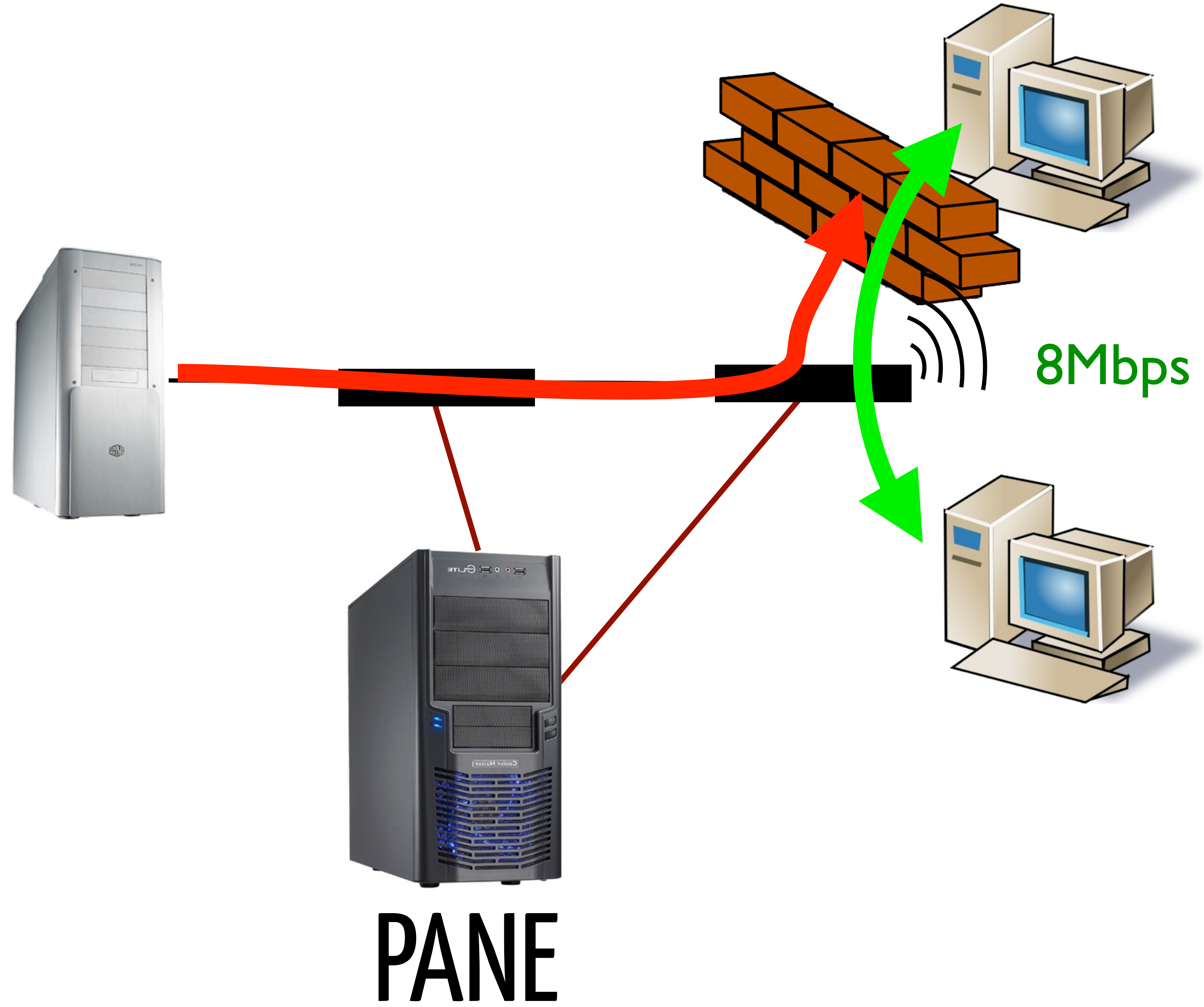
And as the source of the traffic moves ...



The NIB also allows PANE's compiler to choose where in the network to implement desired policies.

As a simple example, it places rules which drop traffic as close as possible to the traffic's ingress port. In this experiment, we have two wireless clients communicating. One suffers from an attack,
(click) and the transfer rate drops. With a local firewall rule,
(click) the transfer only slightly recovers. Using PANE to install the rule,
(click) the transfer fully recovers.

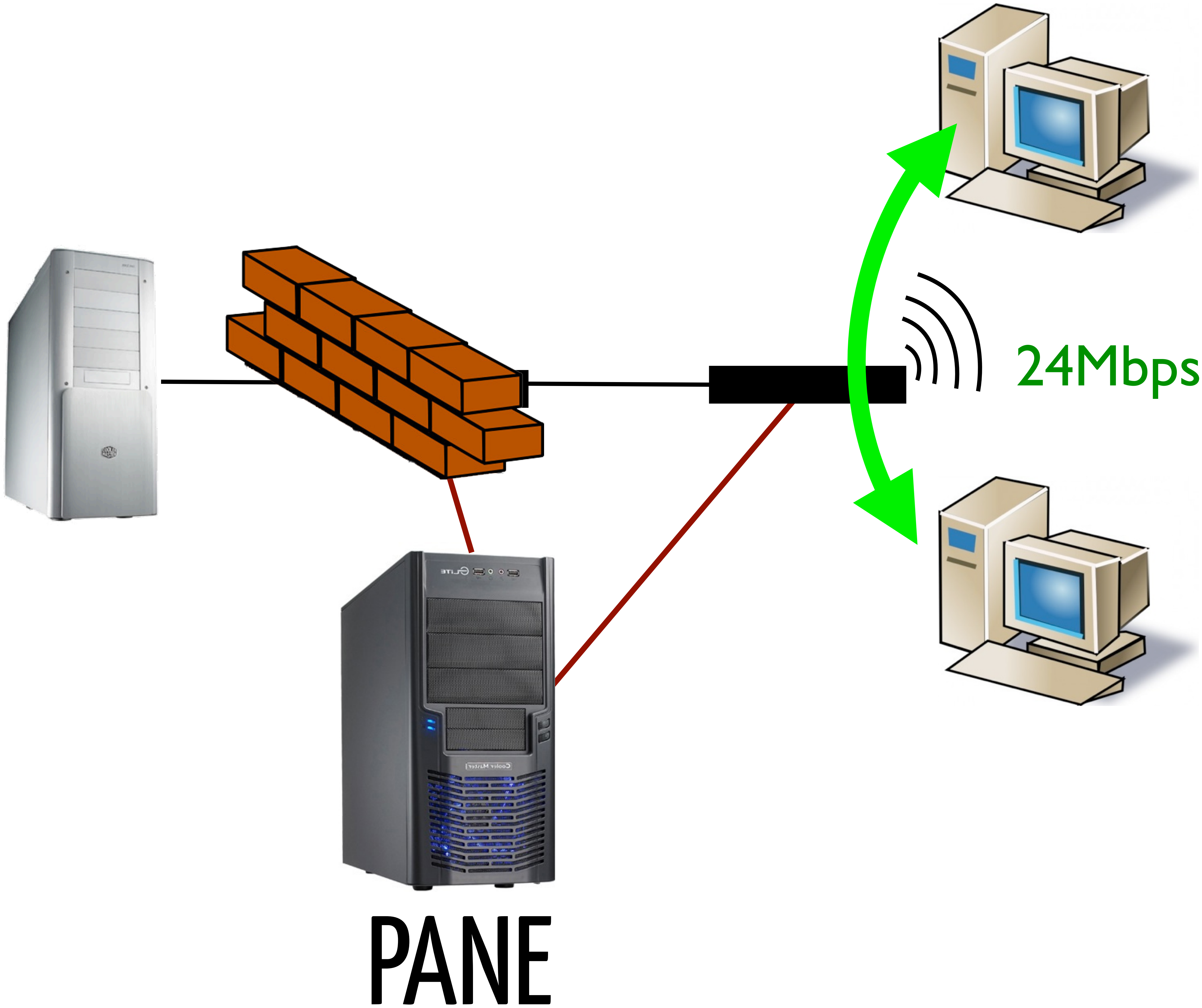
And as the source of the traffic moves ...



The NIB also allows PANE's compiler to choose where in the network to implement desired policies.

As a simple example, it places rules which drop traffic as close as possible to the traffic's ingress port. In this experiment, we have two wireless clients communicating. One suffers from an attack,
(click) and the transfer rate drops. With a local firewall rule,
(click) the transfer only slightly recovers. Using PANE to install the rule,
(click) the transfer fully recovers.

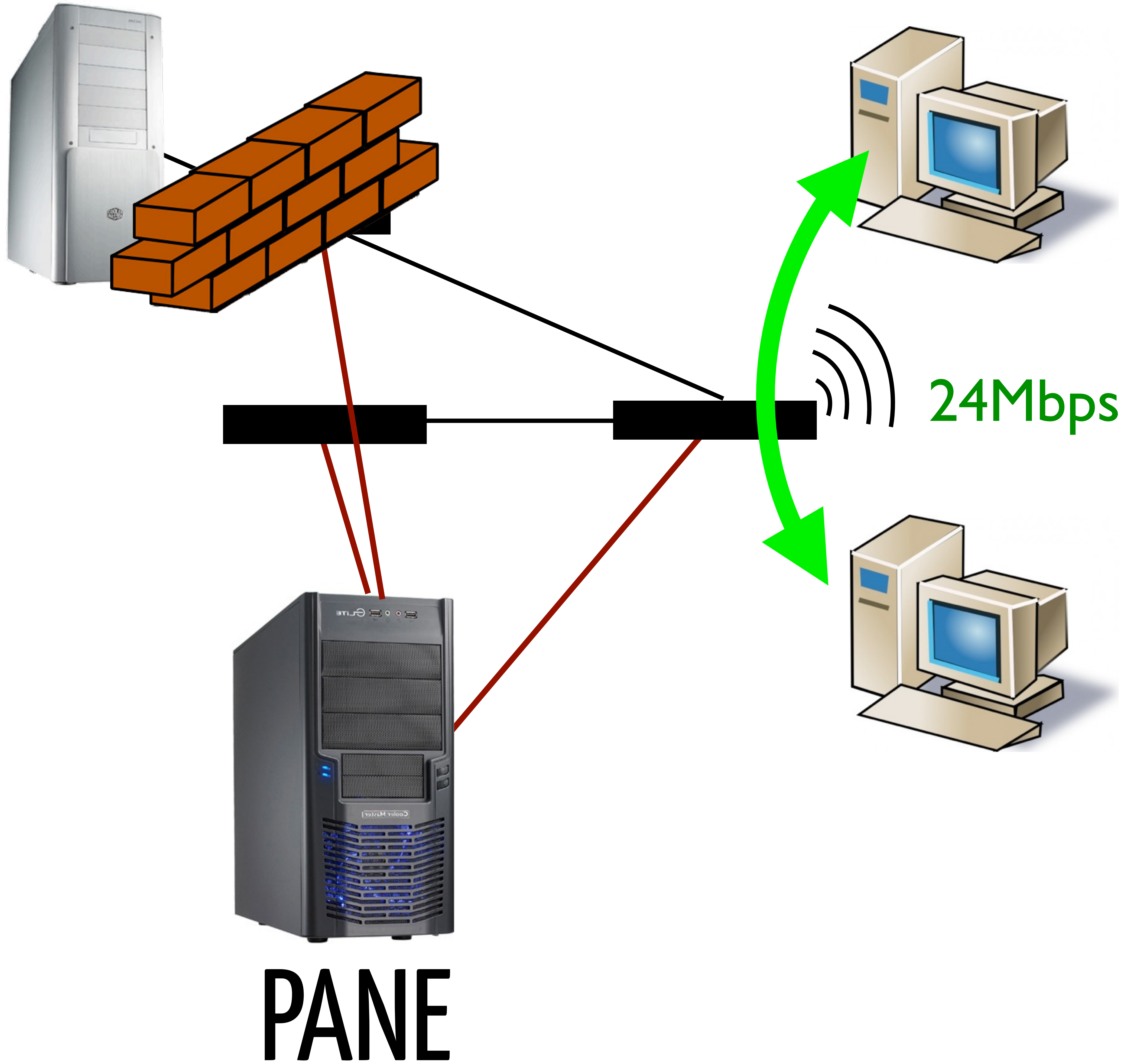
And as the source of the traffic moves ...



The NIB also allows PANE's compiler to choose where in the network to implement desired policies.

As a simple example, it places rules which drop traffic as close as possible to the traffic's ingress port. In this experiment, we have two wireless clients communicating. One suffers from an attack,
(click) and the transfer rate drops. With a local firewall rule,
(click) the transfer only slightly recovers. Using PANE to install the rule,
(click) the transfer fully recovers.

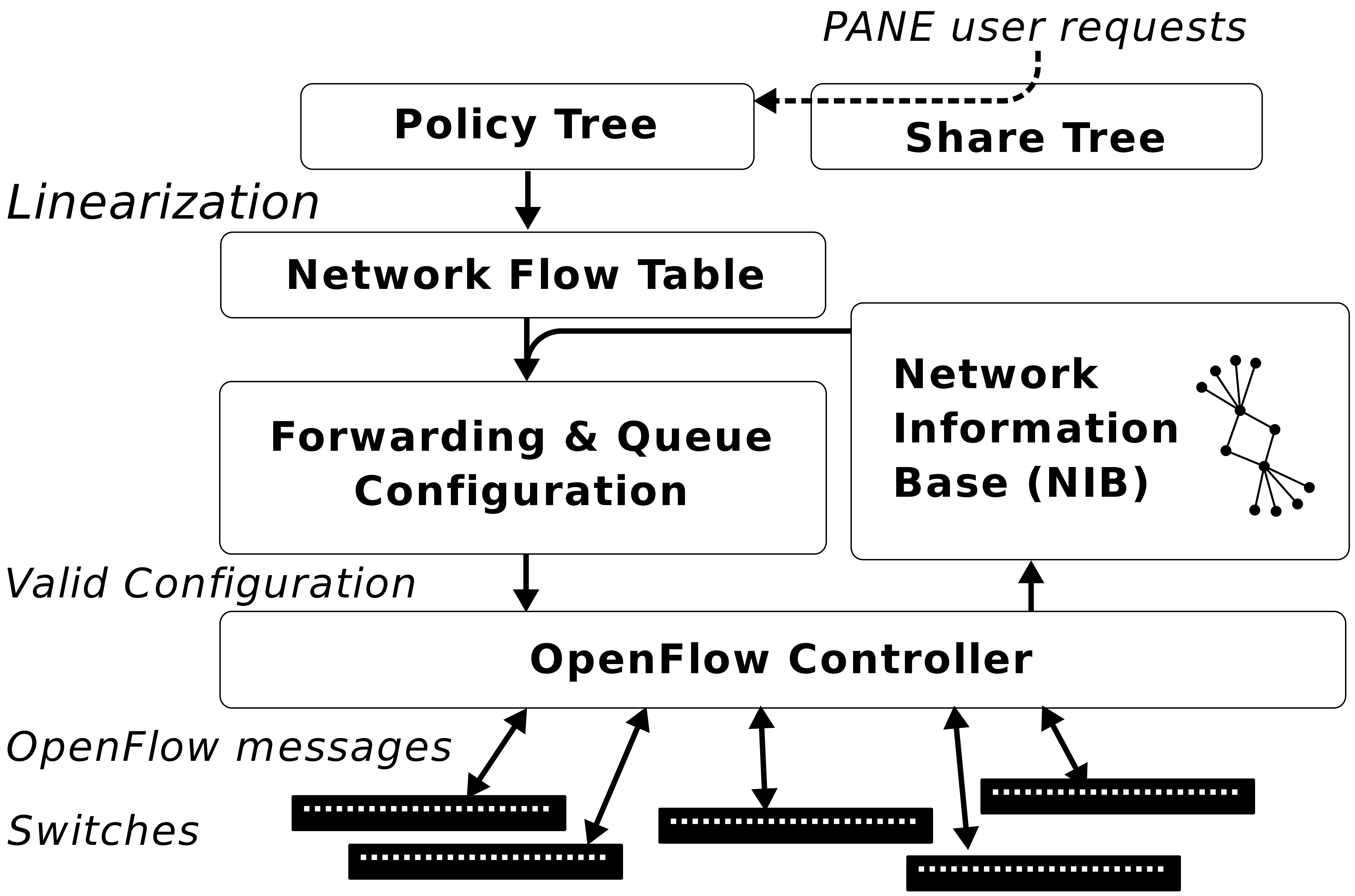
And as the source of the traffic moves ...



… the rule can shift with it.

(Pause)

PANE

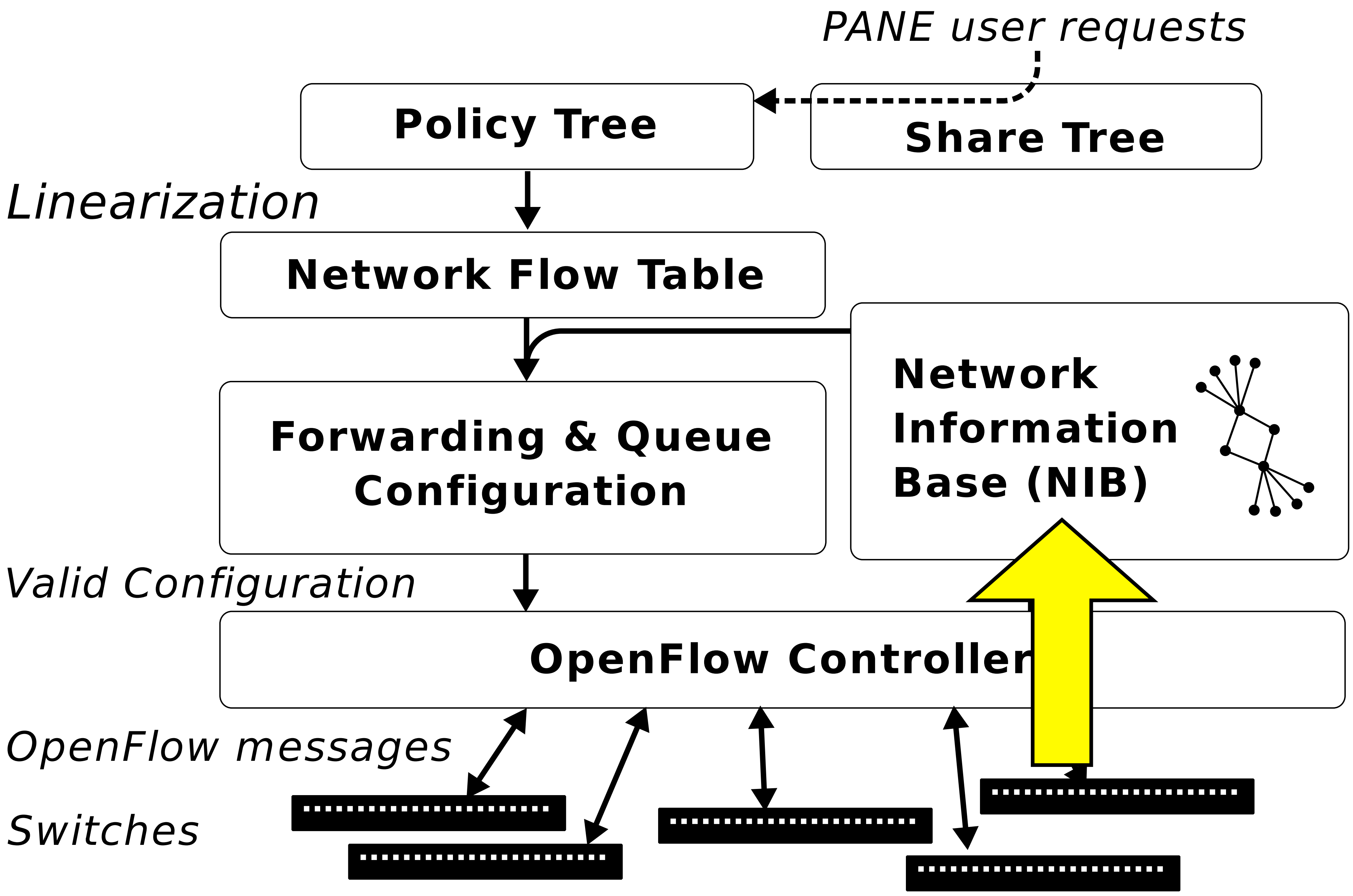


Updating the NIB is the responsibility of our OpenFlow controller,
(click) and any updates are propagated back into our compiler service.
(click) The compiler may then construct a new set of OpenFlow tables
(click) which continue to implement the decisions of the *network* flow table in the new environment.

Seen this way, you can think of the network flow table as a set of invariants we would like to maintain, and our compiler's second stage as a service which maintains those invariants.

(Pause)

PANE

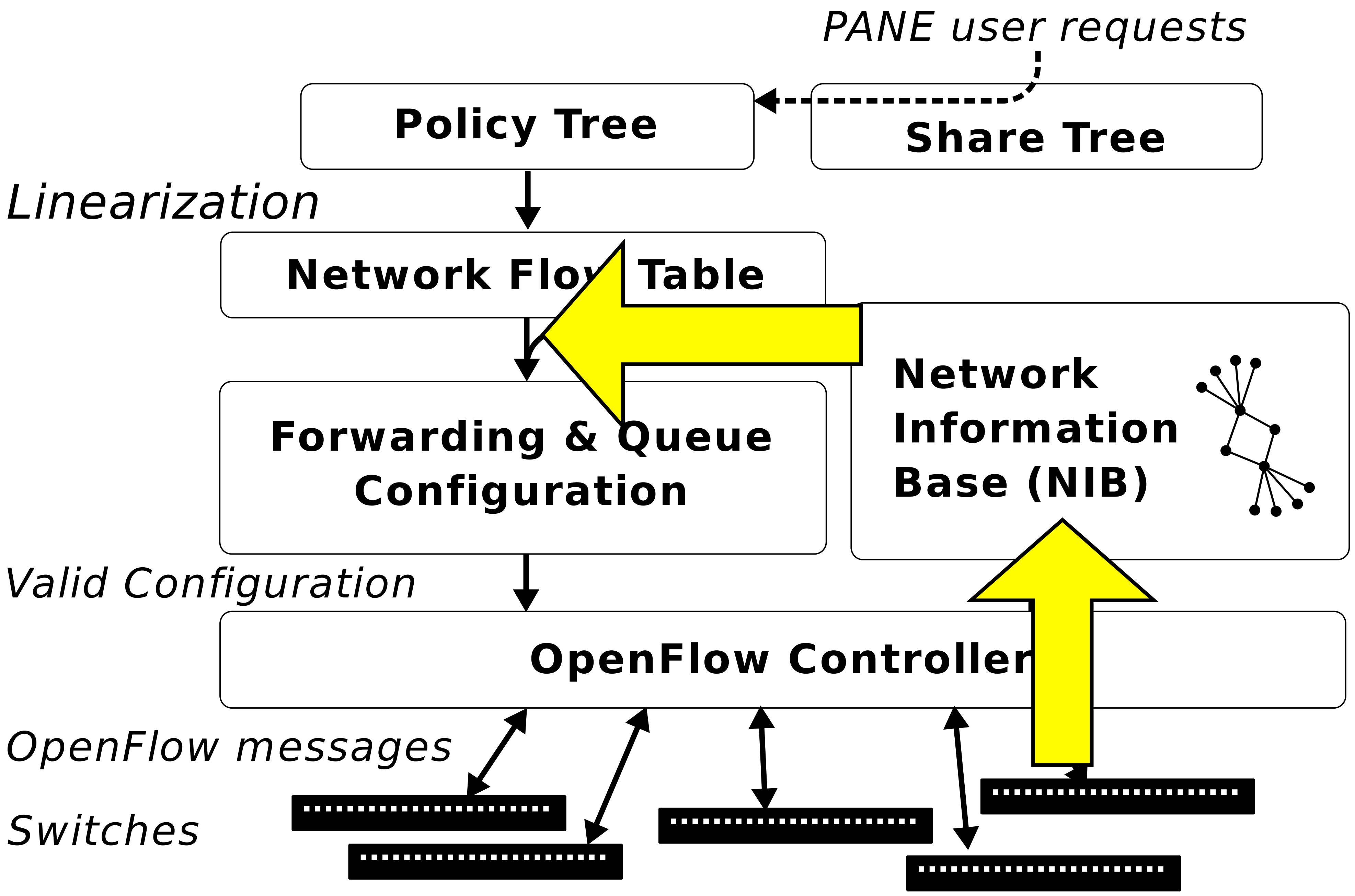


Updating the NIB is the responsibility of our OpenFlow controller,
(click) and any updates are propagated back into our compiler service.
(click) The compiler may then construct a new set of OpenFlow tables
(click) which continue to implement the decisions of the *network* flow table in the new environment.

Seen this way, you can think of the network flow table as a set of invariants we would like to maintain, and our compiler's second stage as a service which maintains those invariants.

(Pause)

PANE

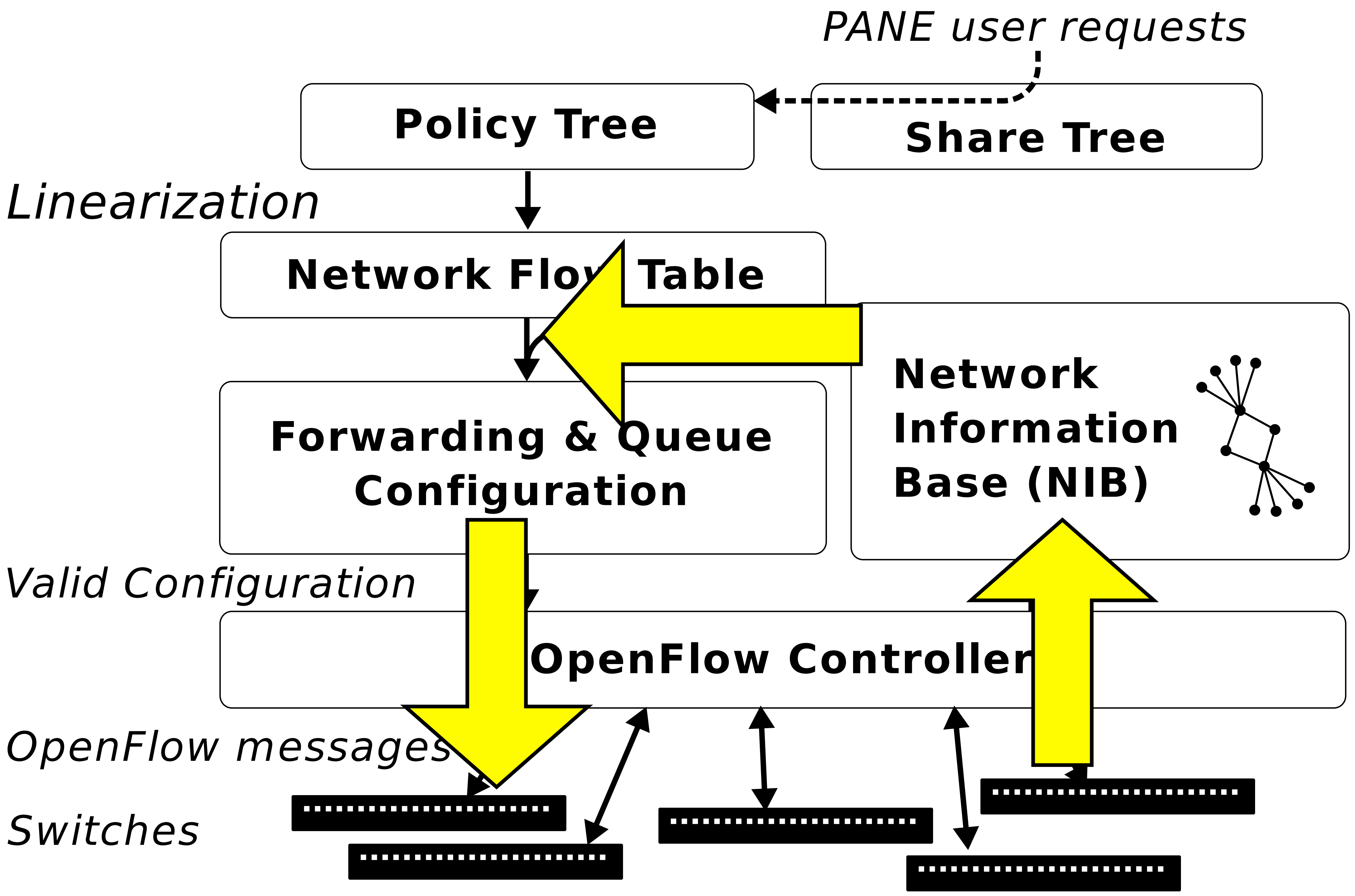


Updating the NIB is the responsibility of our OpenFlow controller,
(click) and any updates are propagated back into our compiler service.
(click) The compiler may then construct a new set of OpenFlow tables
(click) which continue to implement the decisions of the *network* flow table in the new environment.

Seen this way, you can think of the network flow table as a set of invariants we would like to maintain, and our compiler's second stage as a service which maintains those invariants.

(Pause)

PANE



Updating the NIB is the responsibility of our OpenFlow controller,
(click) and any updates are propagated back into our compiler service.
(click) The compiler may then construct a new set of OpenFlow tables
(click) which continue to implement the decisions of the *network* flow table in the new environment.

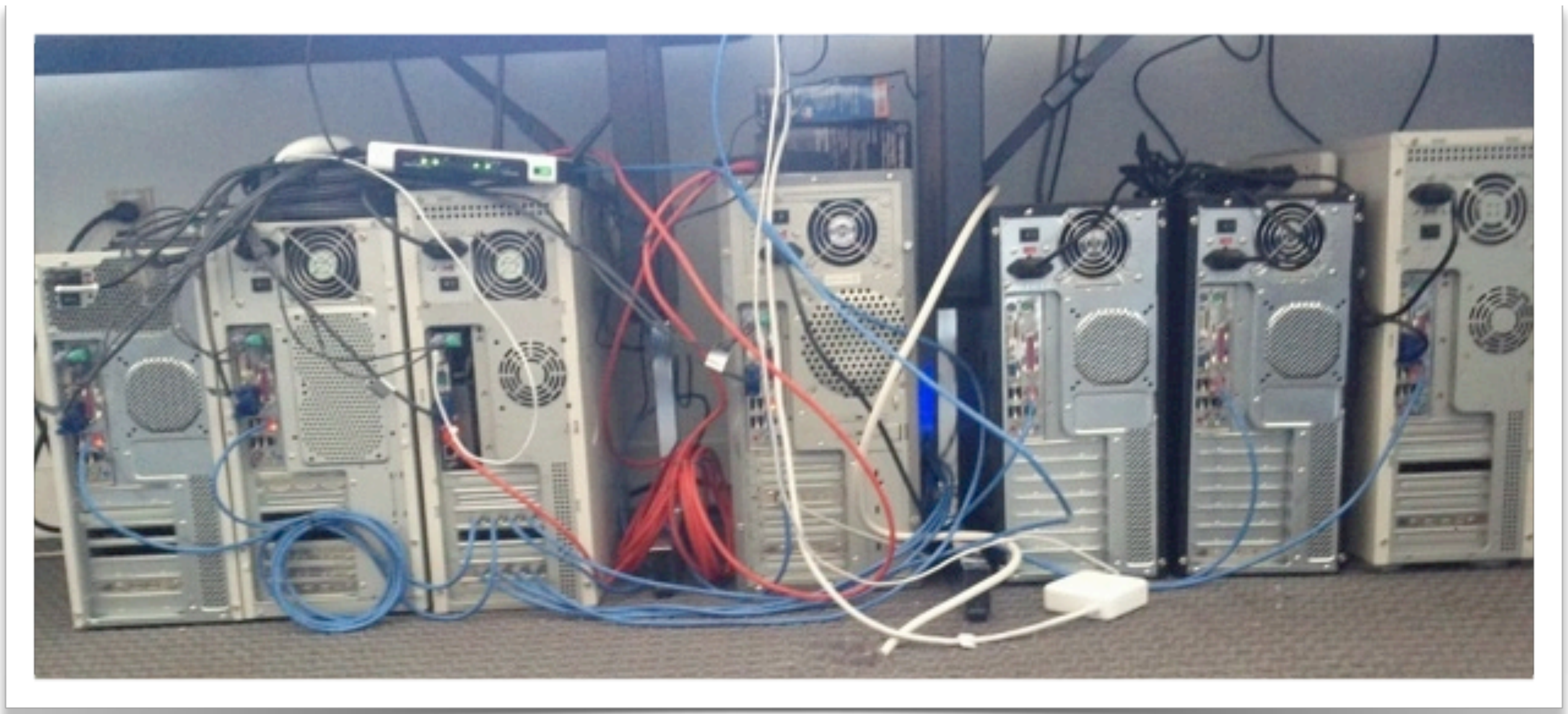
Seen this way, you can think of the network flow table as a set of invariants we would like to maintain, and our compiler's second stage as a service which maintains those invariants.

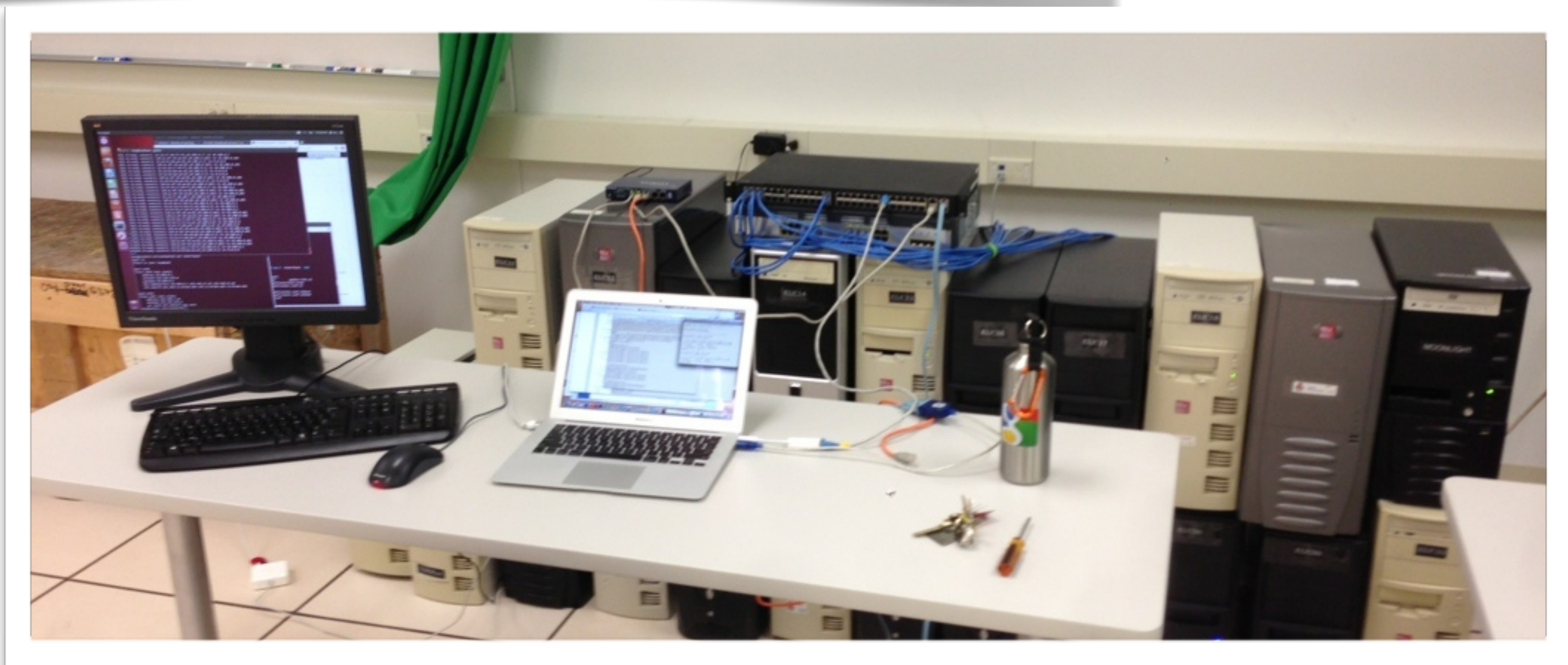
(Pause)

Evaluation

(pause)

We have been running several prototype PANE-controlled networks ...





… which carry traffic in our labs on several hardware and software switches. It provides our day-to-day development and internet connectivity.

1. SSHGuard — access control
2. Ekiga — bandwidth reservations
3. ZooKeeper — queues for low latency
4. Hadoop — centralized traffic weights

Evaluation

We also adapted each of the four applications I discussed earlier to use PANE.
SSHGuard and Ekiga directly use our simple ASCII protocol, while ZooKeeper and Hadoop use an object-oriented Java library we developed.

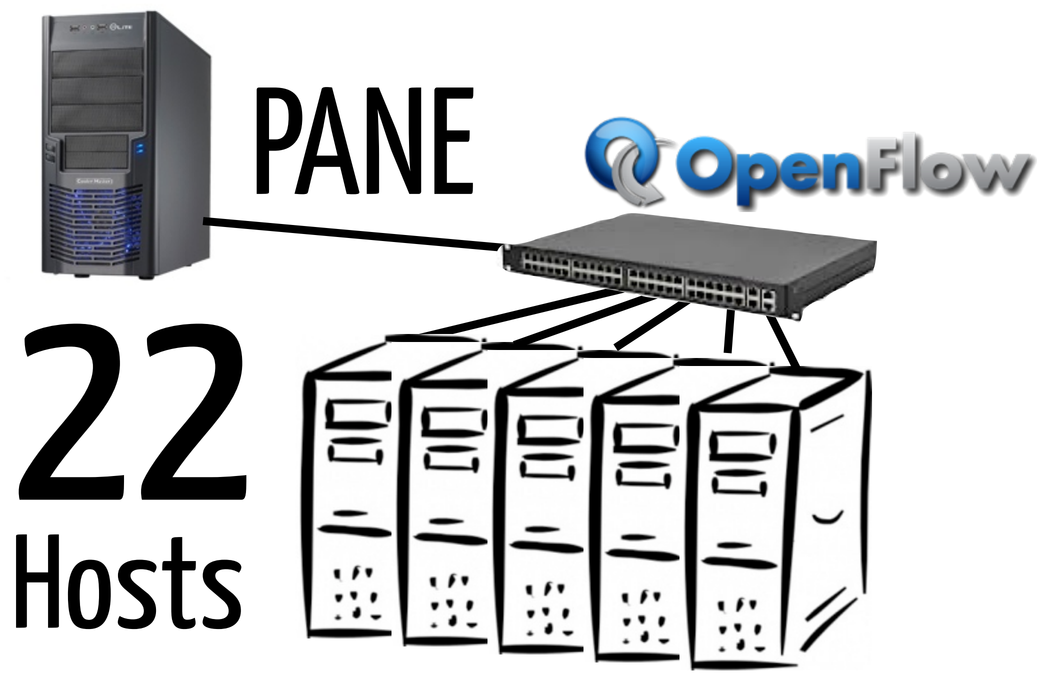
Three equal-sized sort jobs:

- Two Low Priority with 25% weight
- One High Priority with 50% weight

I want to briefly take a look at the Hadoop case:
1) (job mix)
2) (network topology: 20 slaves plus 2 masters)
3) (PANE rules)
4) (outcome: high pri 23% faster, lowpri 10% because of work-conservation)

Three equal-sized sort jobs:

- Two Low Priority with 25% weight
- One High Priority with 50% weight

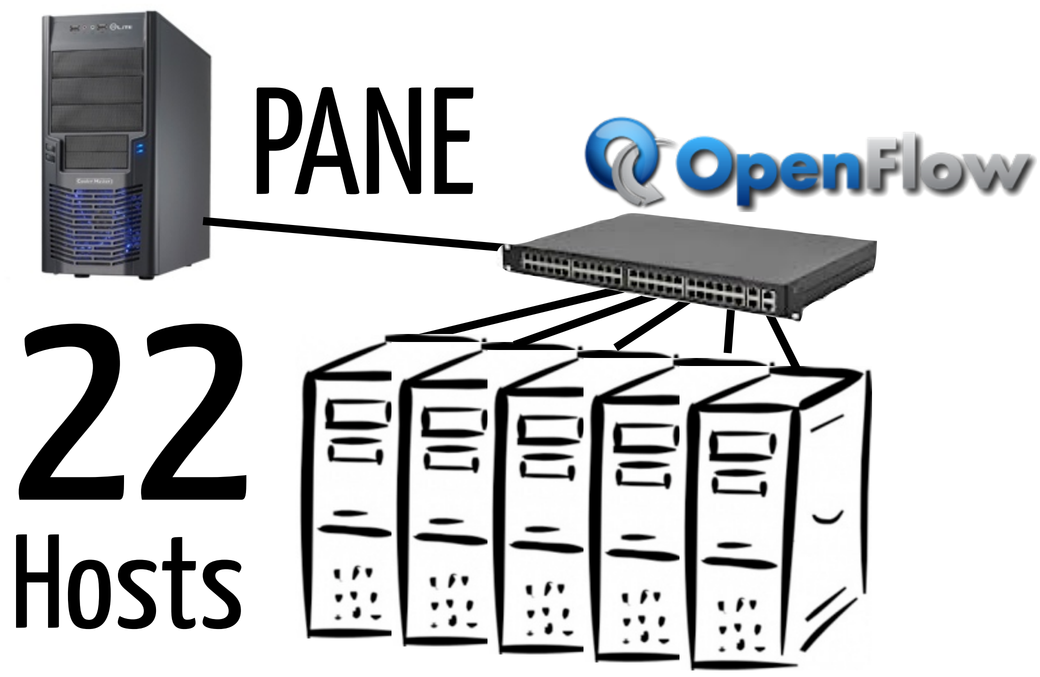


I want to briefly take a look at the Hadoop case:
1) (job mix)
2) (network topology: 20 slaves plus 2 masters)
3) (PANE rules)
4) (outcome: high pri 23% faster, lowpri 10% because of work-conservation)

hadoop

Three equal-sized sort jobs:

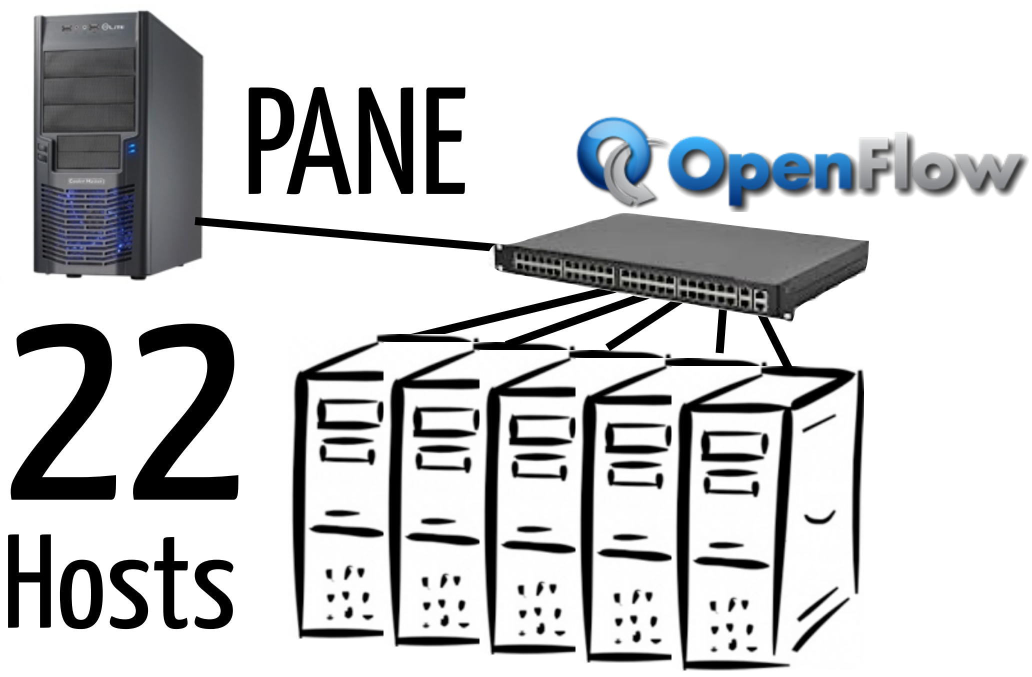
- Two Low Priority with 25% weight
- One High Priority with 50% weight

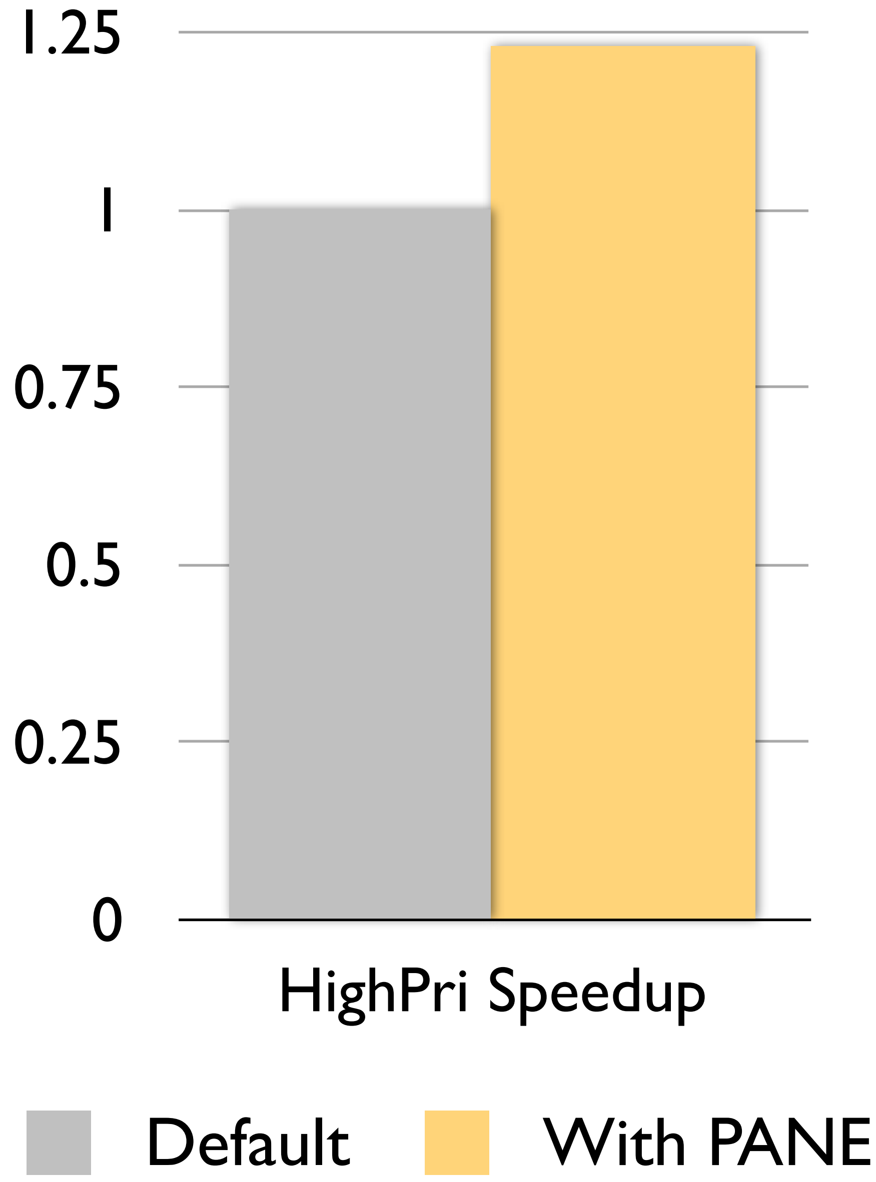


Dynamically apply QoS to High Priority flows using PANE.

I want to briefly take a look at the Hadoop case:
1) (job mix)
2) (network topology: 20 slaves plus 2 masters)
3) (PANE rules)
4) (outcome: high pri 23% faster, lowpri 10% because of work-conservation)

Three equal-sized sort jobs:
- Two Low Priority with 25% weight
- One High Priority with 50% weight



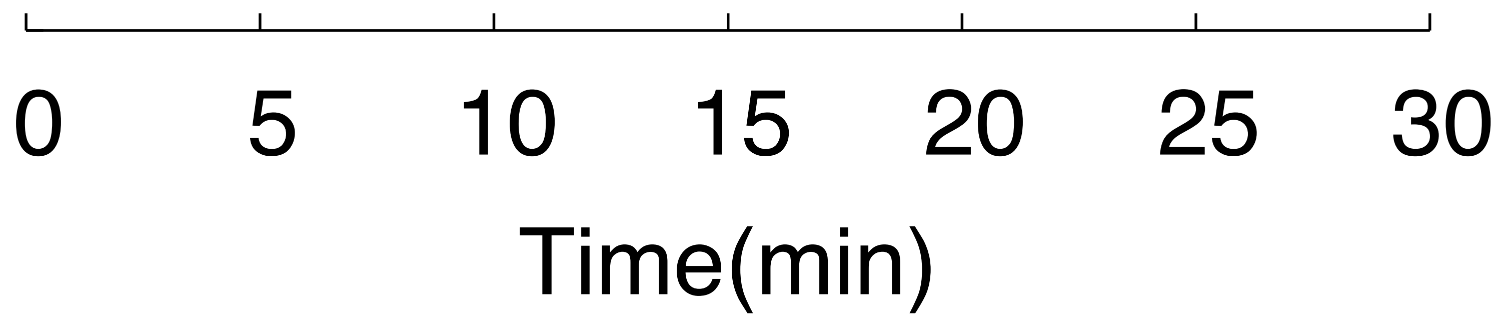


Dynamically apply QoS to High Priority flows using PANE.

I want to briefly take a look at the Hadoop case:
1) (job mix)
2) (network topology: 20 slaves plus 2 masters)
3) (PANE rules)
4) (outcome: high pri 23% faster, lowpri 10% because of work-conservation)

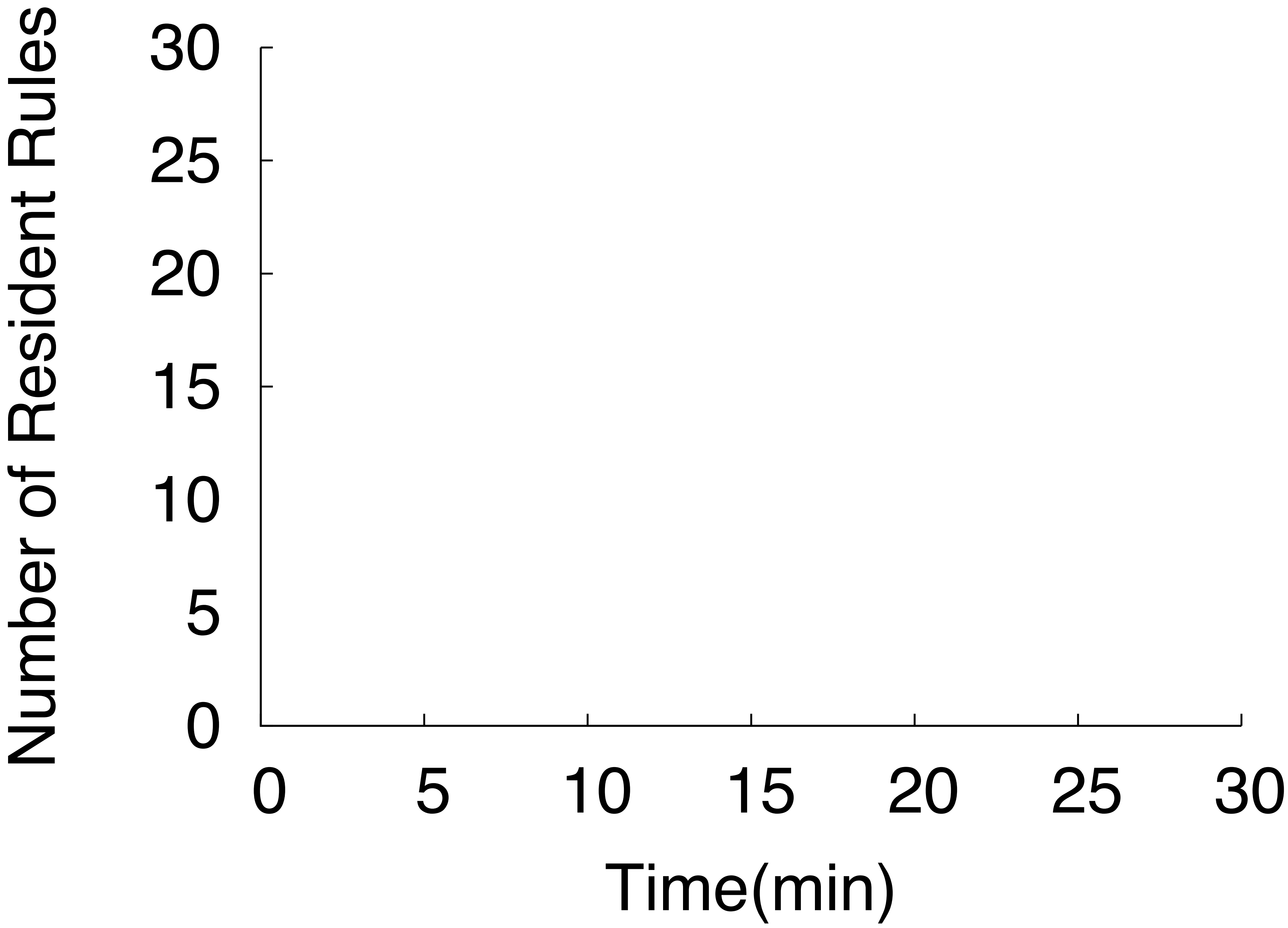
Hadoop's OpenFlow rules

x-axis: time
y-axis: number of rules created by one job running across 22 hosts



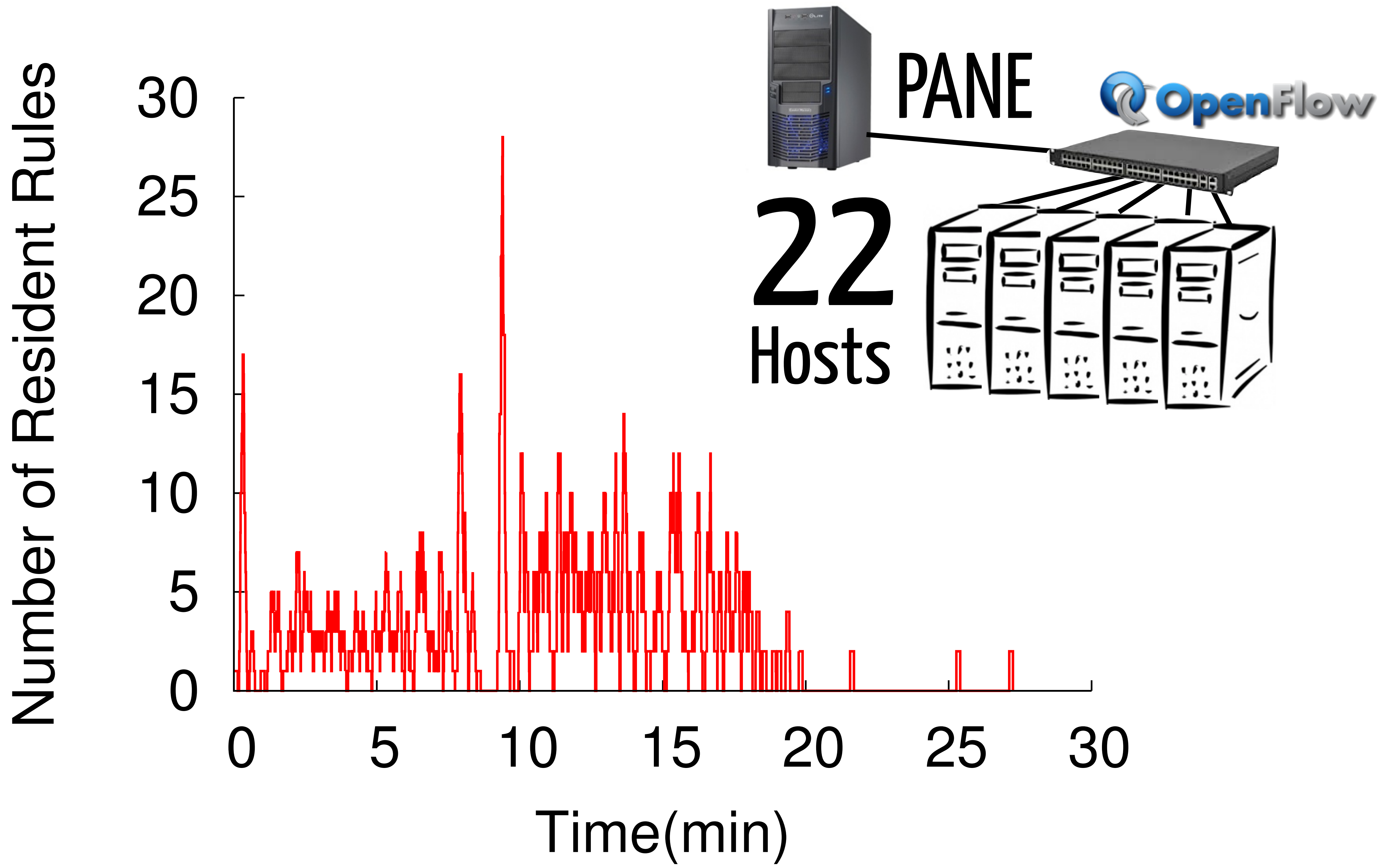
Hadoop's OpenFlow rules

x-axis: time
y-axis: number of rules created by one job running across 22 hosts



Hadoop's OpenFlow rules

x-axis: time
y-axis: number of rules created by one job running across 22 hosts



Hadoop's OpenFlow rules

x-axis: time
y-axis: number of rules created by one job running across 22 hosts

1. For applications that know what they want from the network

2. Allows these applications to co-exist

Conclusion

In conclusion, PANE is designed for applications and users that know what they want from the network. PANE provides a way for applications to talk back to the control-plane and use any mechanisms exposed by network. So far we've explored bandwidth, access control, routing, and rate-limiting, and hope to support new mechanisms in the future.

And second, PANE allows all of these application requests to co-exist with a single network by deterministically resolving conflicting requests into a single policy.

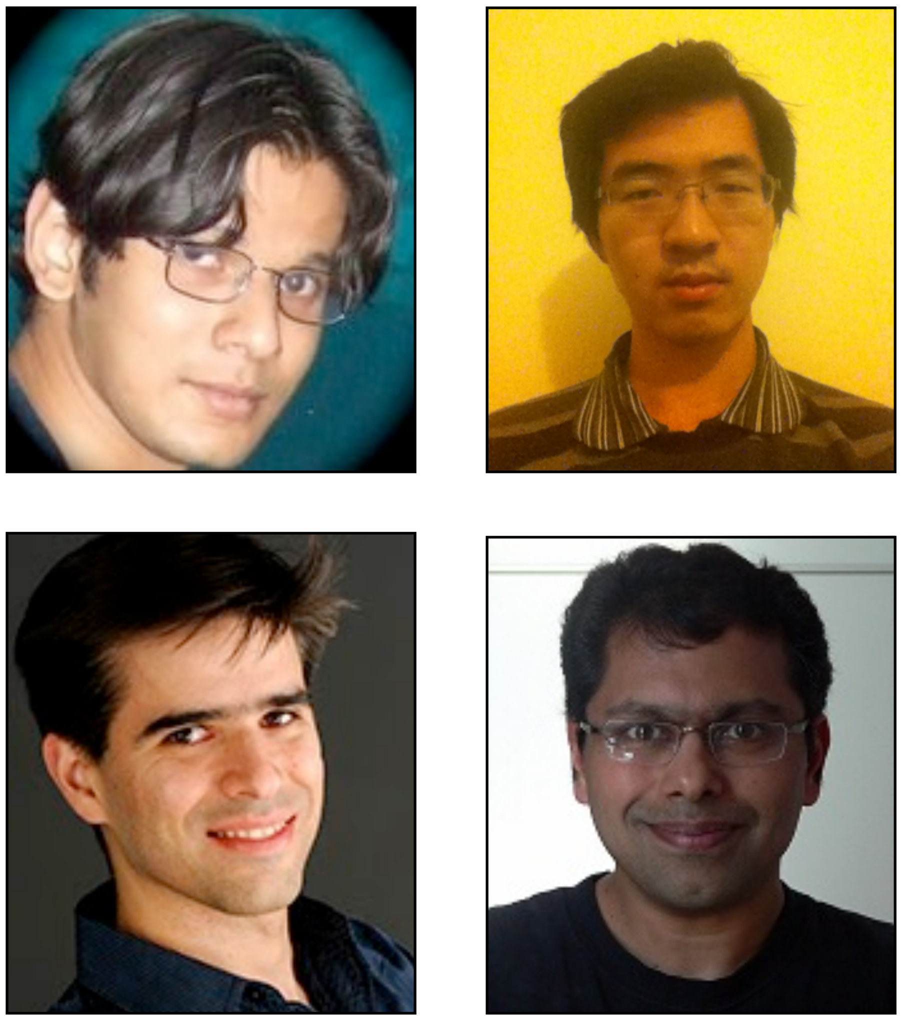
pane.cs.brown.edu

Andrew Ferguson
adf@cs.brown.edu

I'm happy to take your questions at this time…

Co-authors

- **Arjun Guha**
  Brown ↦ Cornell ↦ UMass Amherst
- **Chen Liang**
  Brown ↦ Duke
- **Rodrigo Fonseca**
  Brown
- **Shriram Krishnamurthi**
  Brown



pane.cs.brown.edu

Andrew Ferguson
adf@cs.brown.edu

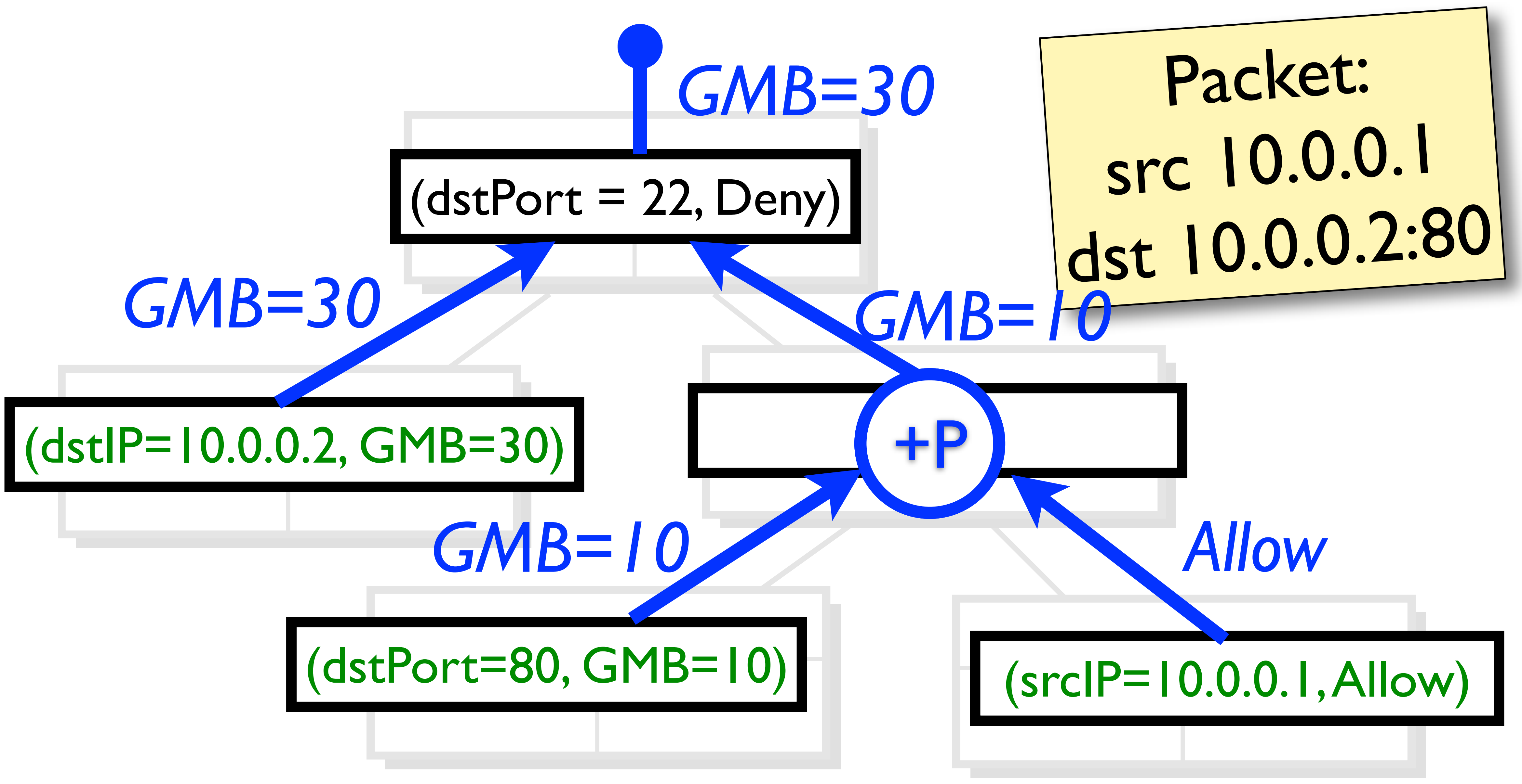
... or you can contact any of my collaborators as well.

Thank you very much!

Backup Slides

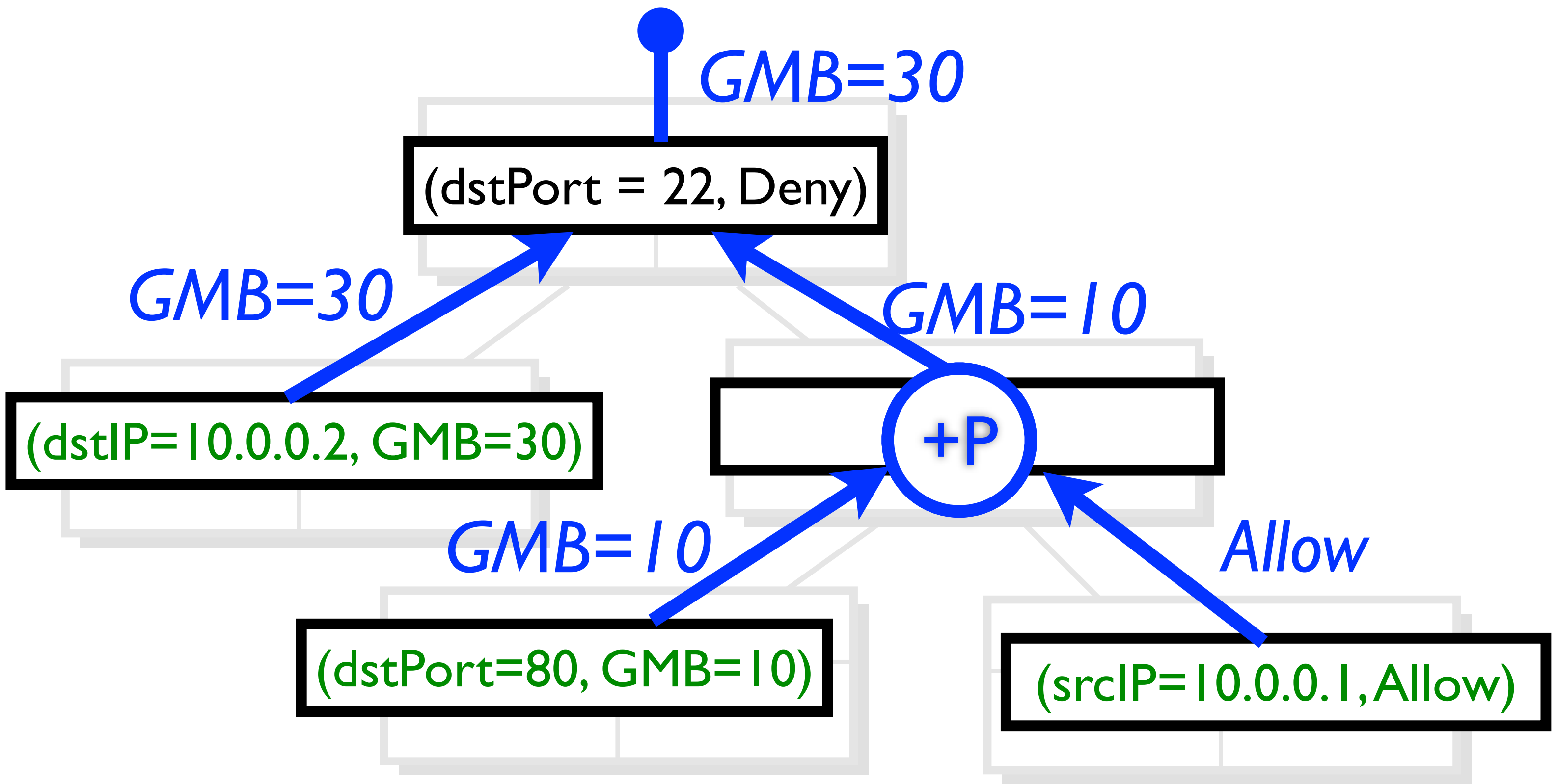
Proof of Correctness

- As we saw on the last slide, this is a complex, concurrent system.
- And complex systems have bugs, even if you write them in Haskell, as we did.
- I'd like to briefly tell you how we proved a key portion of the system correct.



Hierarchical Flow Tables

- As a starting point, we know what it means for a hierarchical flow table to process a packet: the packet enters the switch, the policy tree nodes produce their actions, and a result action is produced after applying the combination operators.

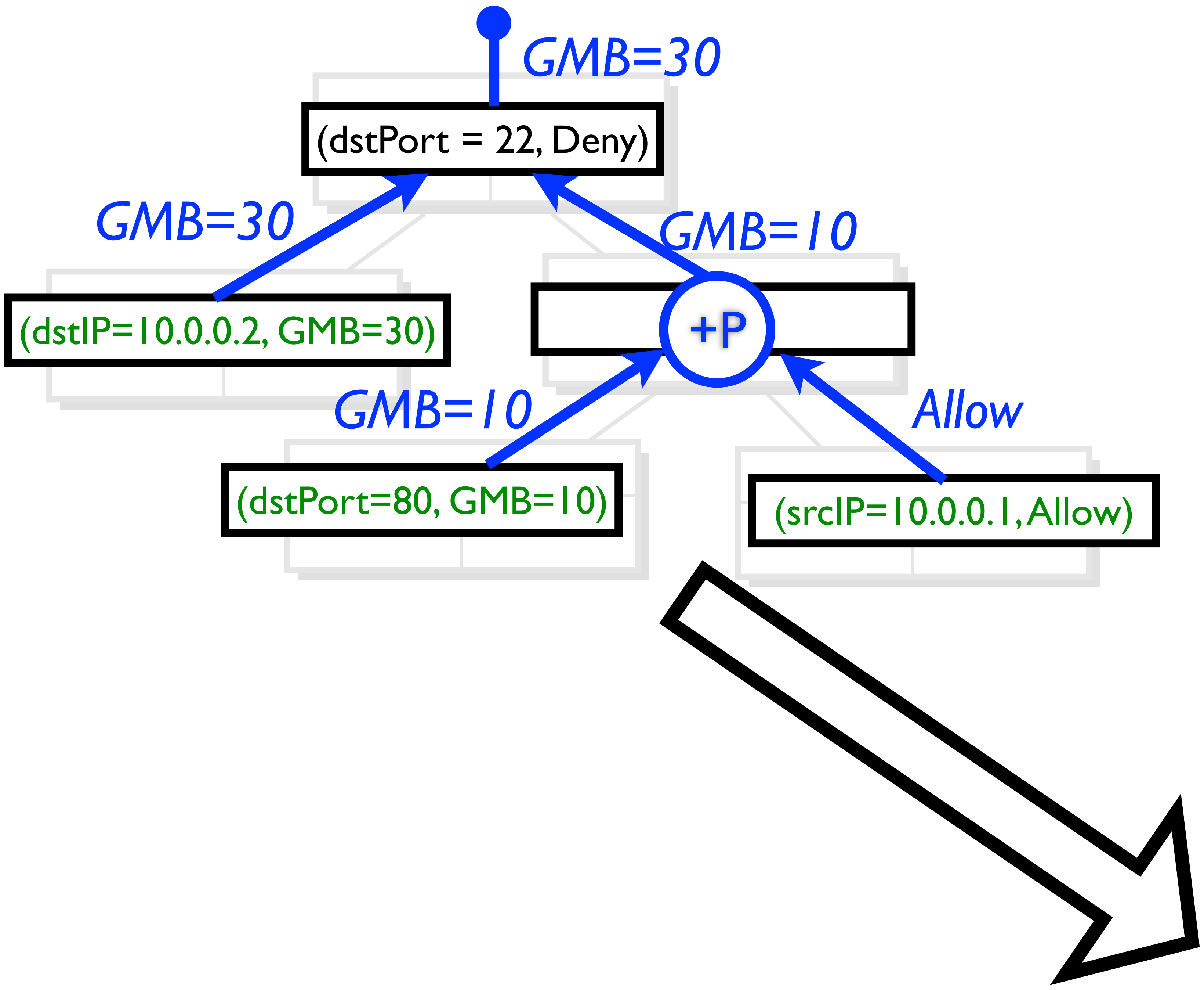


Compiler Correctness

- To make this efficient, we've built a compiler
  (click) from declarative, hierarchical policies to linear, flow tables.
- How do we know that this compiler is actually correct?
- Compilers are also notorious difficult to get right.
- If this compiler has a bug,

(click) it's not that a program may crash, but the entire network may go down.

- Or, a more subtle error may occur, such as traffic that should be blocked, may instead be permitted.



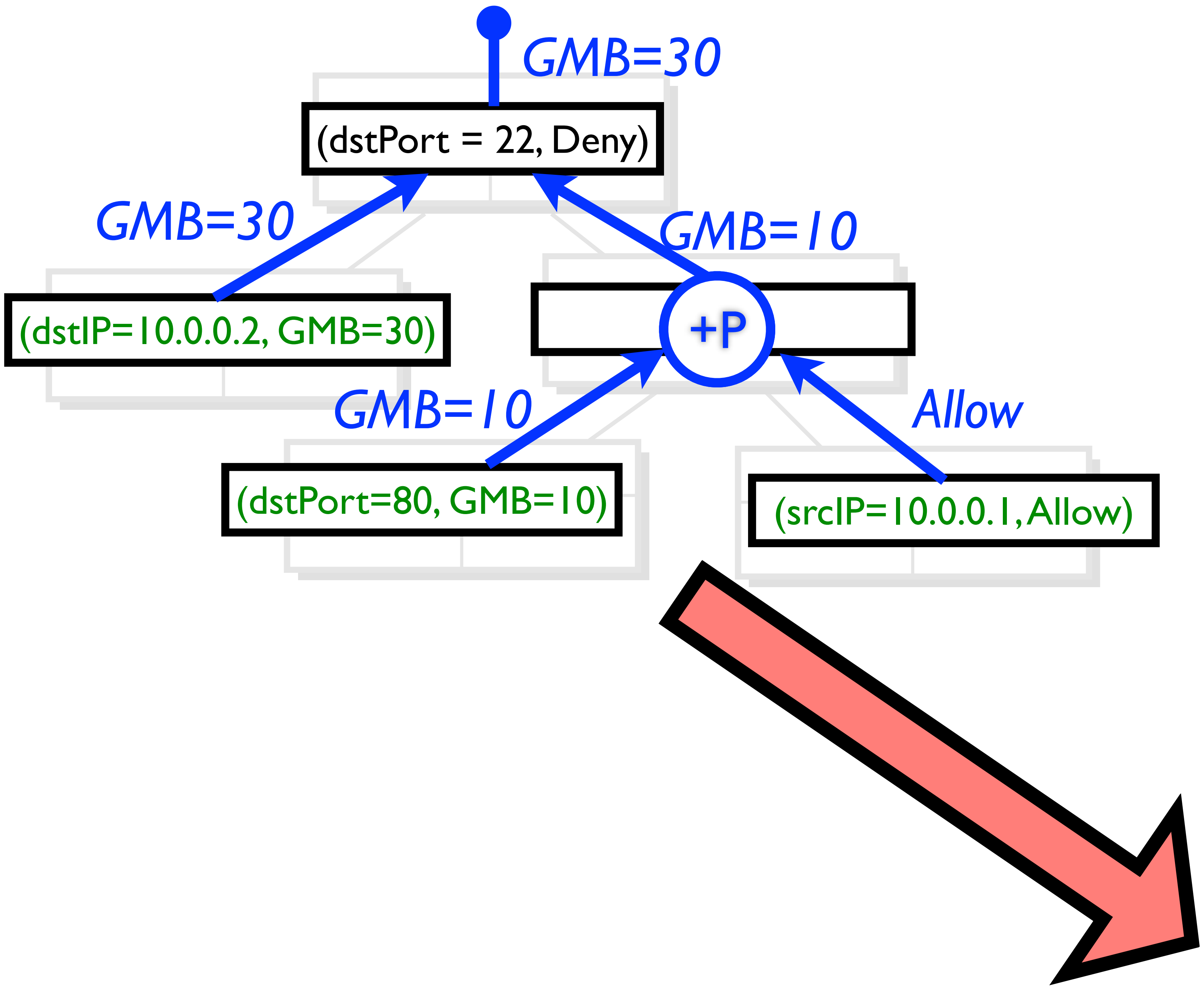
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6
*	*	*	*	*	*	*	*	*	22	drop

Compiler Correctness

- To make this efficient, we've built a compiler
  (click) from declarative, hierarchical policies to linear, flow tables.
- How do we know that this compiler is actually correct?
- Compilers are also notorious difficult to get right.
- If this compiler has a bug,

(click) it's not that a program may crash, but the entire network may go down.

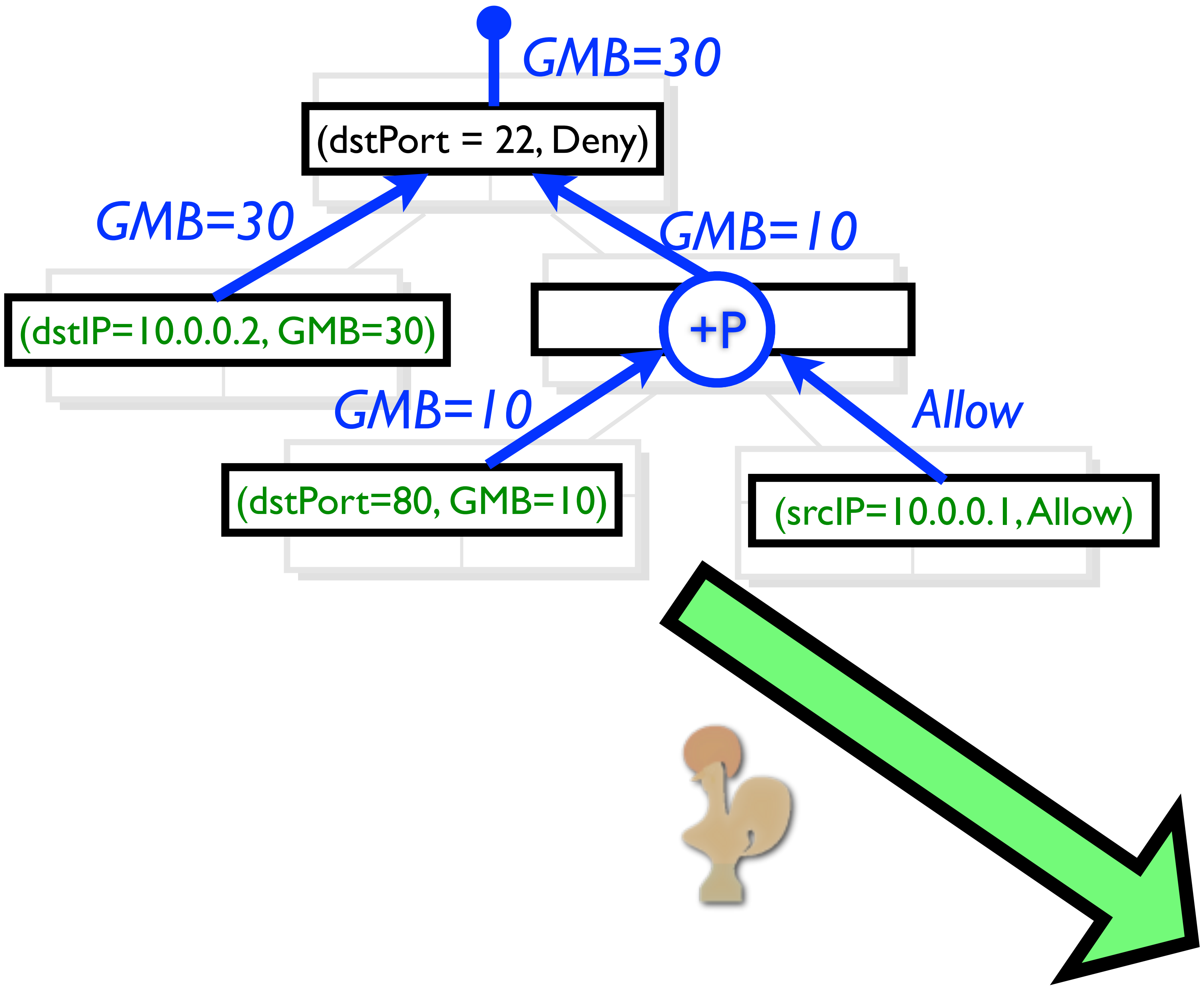
- Or, a more subtle error may occur, such as traffic that should be blocked, may instead be permitted.



Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6
*	*	*	*	*	*	*	*	*	22	drop

Compiler Correctness

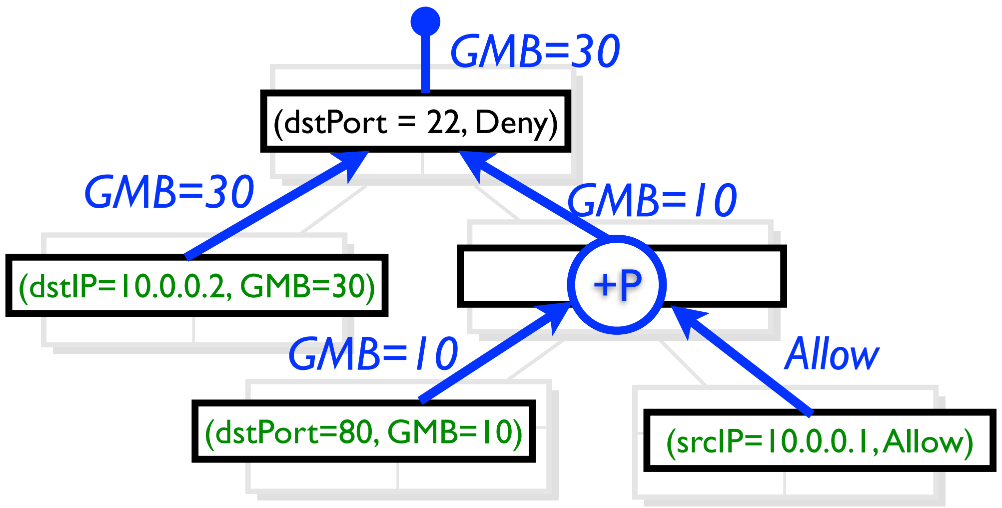
- To make this efficient, we've built a compiler
  (click) from declarative, hierarchical policies to linear, flow tables.
- How do we know that this compiler is actually correct?
- Compilers are also notorious difficult to get right.
- If this compiler has a bug,
(click) it's not that a program may crash, but the entire network may go down.
- Or, a more subtle error may occur, such as traffic that should be blocked, may instead be permitted.

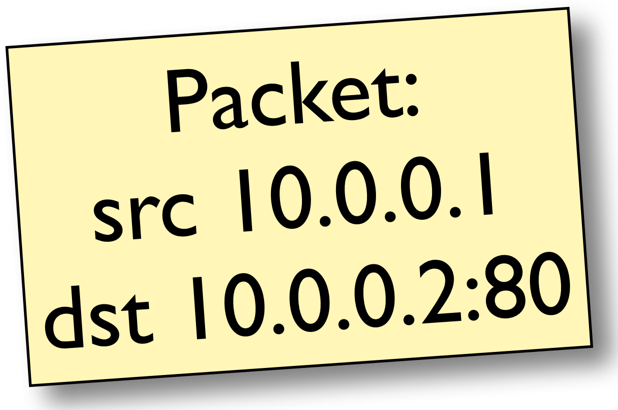


Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6
*	*	*	*	*	*	*	*	*	22	drop

Coq Proof Assistant

- Using the Coq proof assistant, we modeled and wrote a proof of the translation from HFT to OpenFlow tables
- Coq lets us write programs in a functional language, similar to ML or Haskell, and gives us the ability to prove properties of these programs.



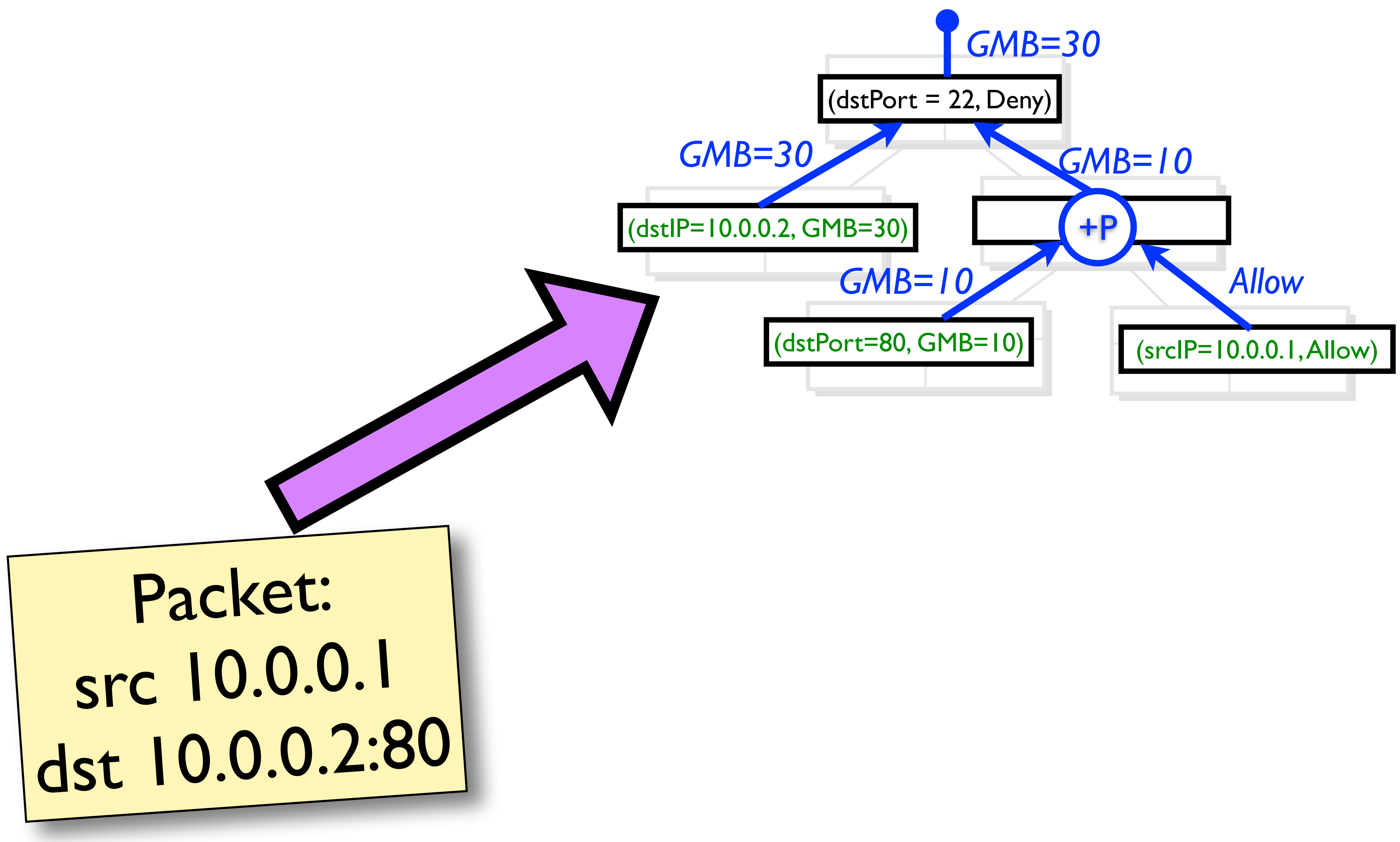


Theorem

– So, let’s look at what we actually proved. Logically, the HFT processes a packet
(click) and produces an action

(click) When we compile the HFT to a network flow table,
(click) the flow table produces exactly the same action (click) on the same packet.

Proving this theorem requires a formal semantics for Hierarchical Flow Tables, which you can find in detail in our paper. The paper also contains the precise statement of this theorem, and the mechanized Coq proofs are available on our website.

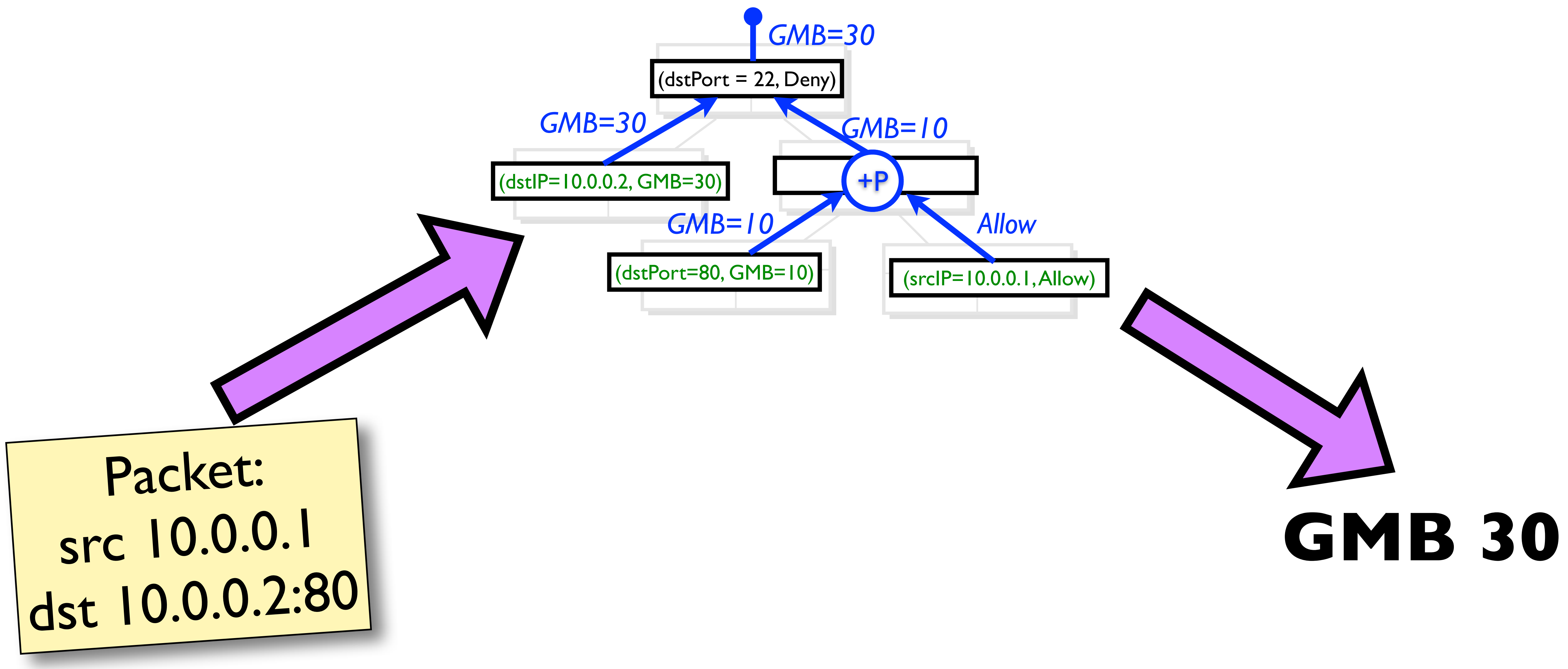


Theorem

– So, let's look at what we actually proved. Logically, the HFT processes a packet (click) and produces an action

(click) When we compile the HFT to a network flow table,
(click) the flow table produces exactly the same action (click) on the same packet.

Proving this theorem requires a formal semantics for Hierarchical Flow Tables, which you can find in detail in our paper. The paper also contains the precise statement of this theorem, and the mechanized Coq proofs are available on our website.

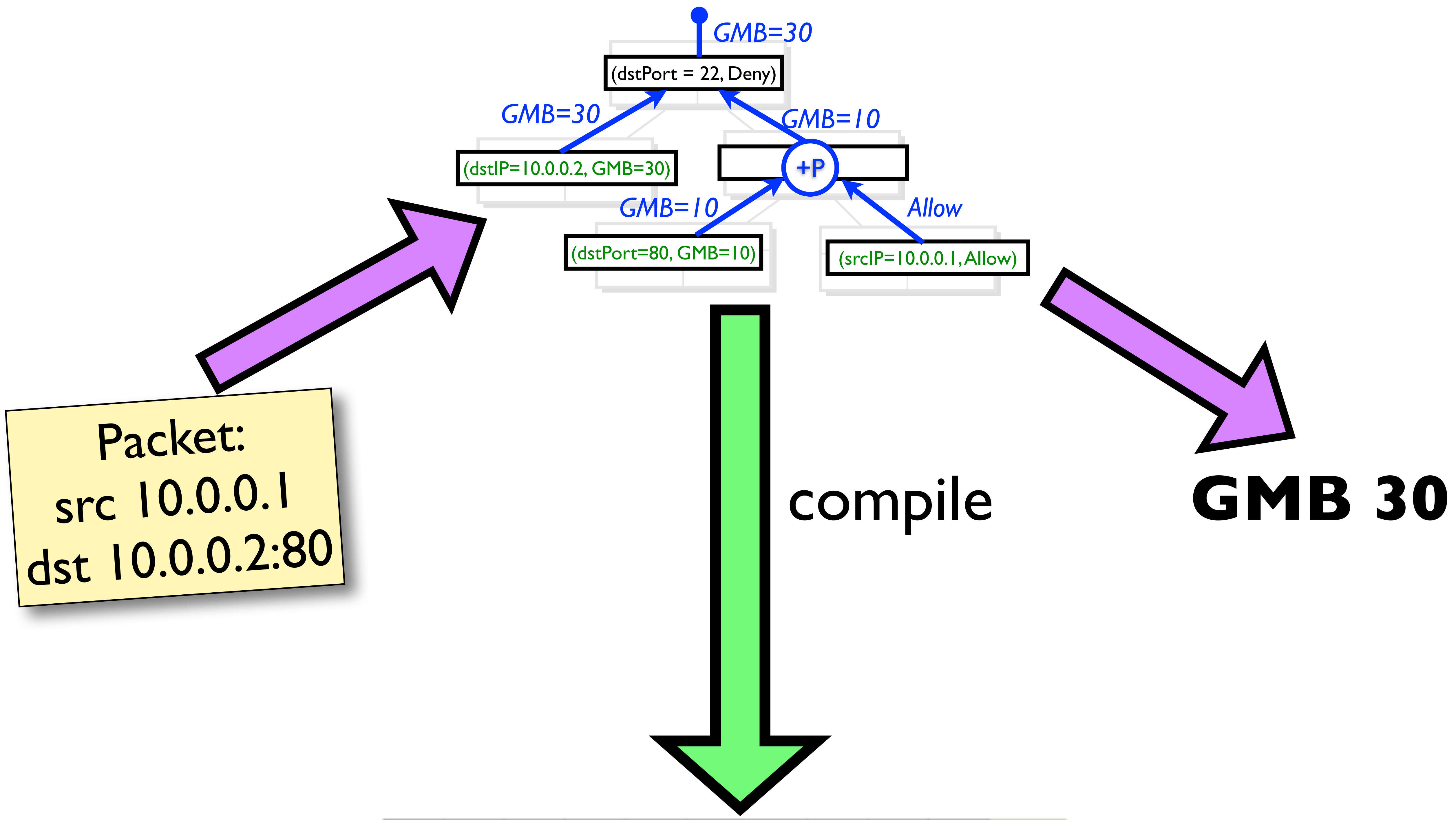


Theorem

- So, let's look at what we actually proved. Logically, the HFT processes a packet (click) and produces an action

(click) When we compile the HFT to a network flow table,
(click) the flow table produces exactly the same action (click) on the same packet.

Proving this theorem requires a formal semantics for Hierarchical Flow Tables, which you can find in detail in our paper. The paper also contains the precise statement of this theorem, and the mechanized Coq proofs are available on our website.



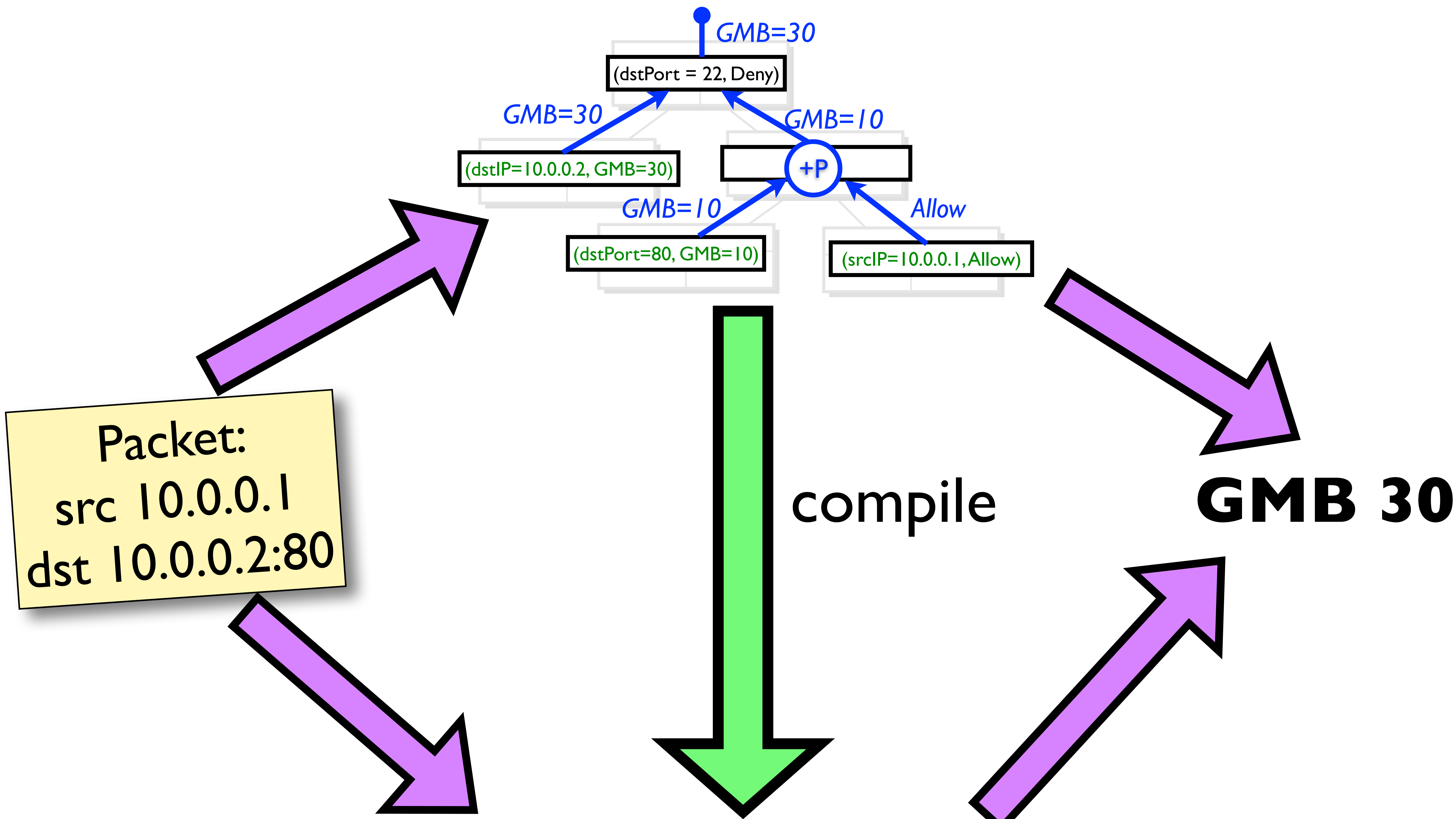
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6
*	*	*	*	*	*	*	*	*	22	drop

Theorem

– So, let's look at what we actually proved. Logically, the HFT processes a packet
(click) and produces an action

(click) When we compile the HFT to a network flow table,
(click) the flow table produces exactly the same action (click) on the same packet.

Proving this theorem requires a formal semantics for Hierarchical Flow Tables, which you can find in detail in our paper. The paper also contains the precise statement of this theorem, and the mechanized Coq proofs are available on our website.



Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6
port3	00:2e..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6
*	*	*	*	*	*	*	*	*	22	drop

Theorem

– So, let's look at what we actually proved. Logically, the HFT processes a packet (click) and produces an action

(click) When we compile the HFT to a network flow table,
(click) the flow table produces exactly the same action (click) on the same packet.

Proving this theorem requires a formal semantics for Hierarchical Flow Tables, which you can find in detail in our paper. The paper also contains the precise statement of this theorem, and the mechanized Coq proofs are available on our website.

Protocol

Now that we've explored PANE's semantics, we'll take a brief look at its protocol for interactively using and delegating network resources.

PANE

As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

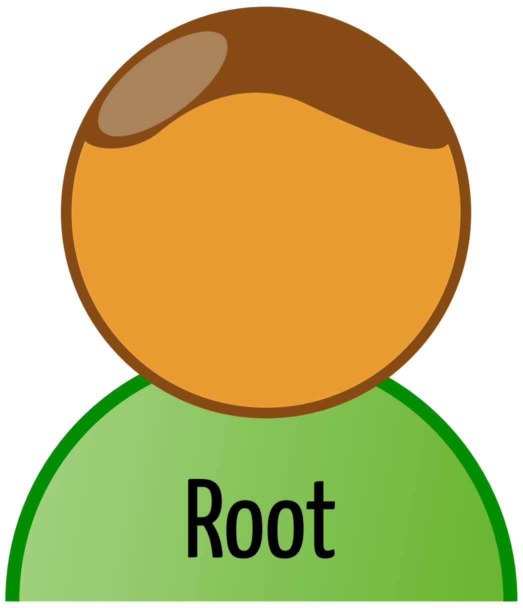
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



PANE

As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

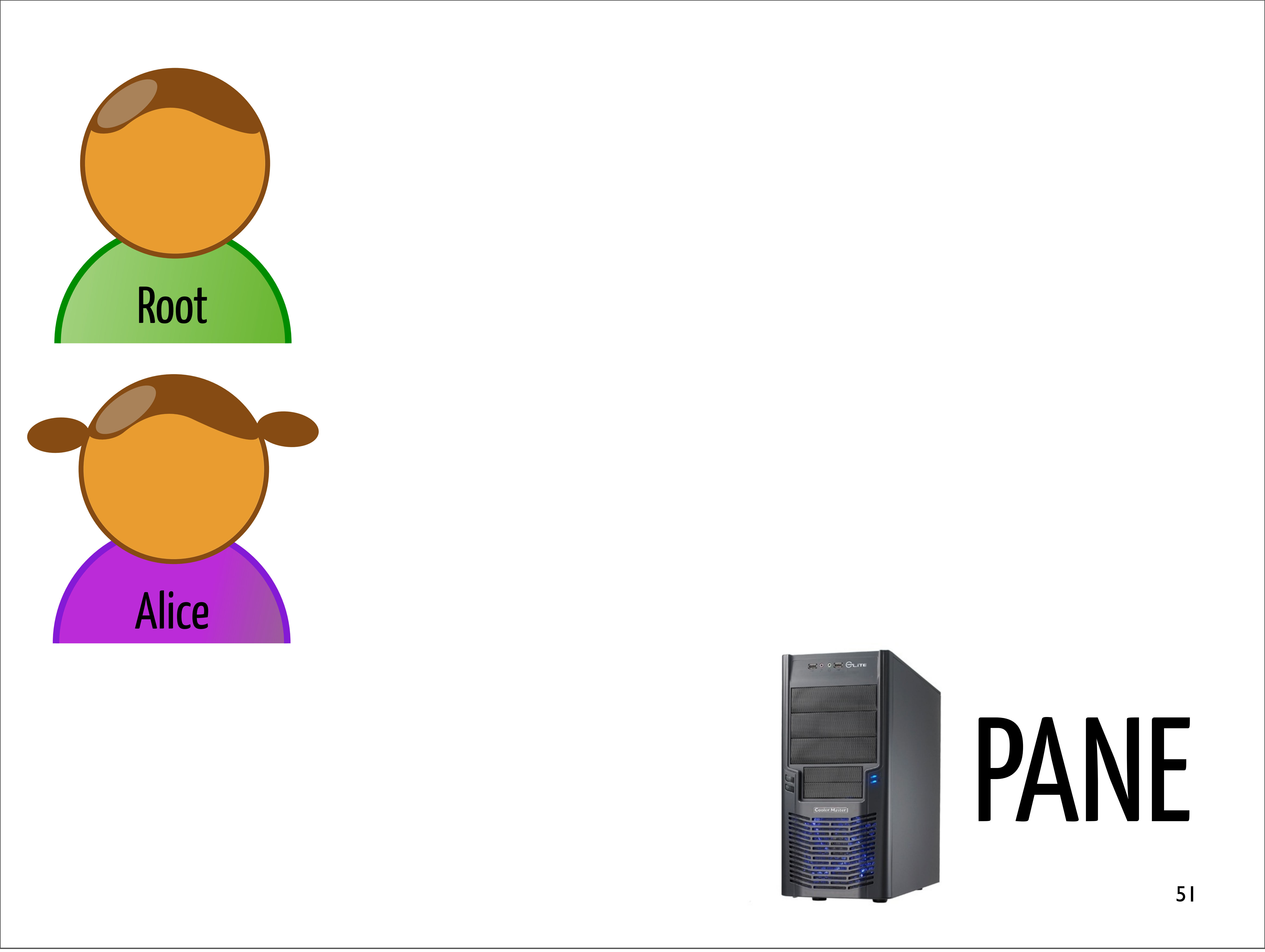
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

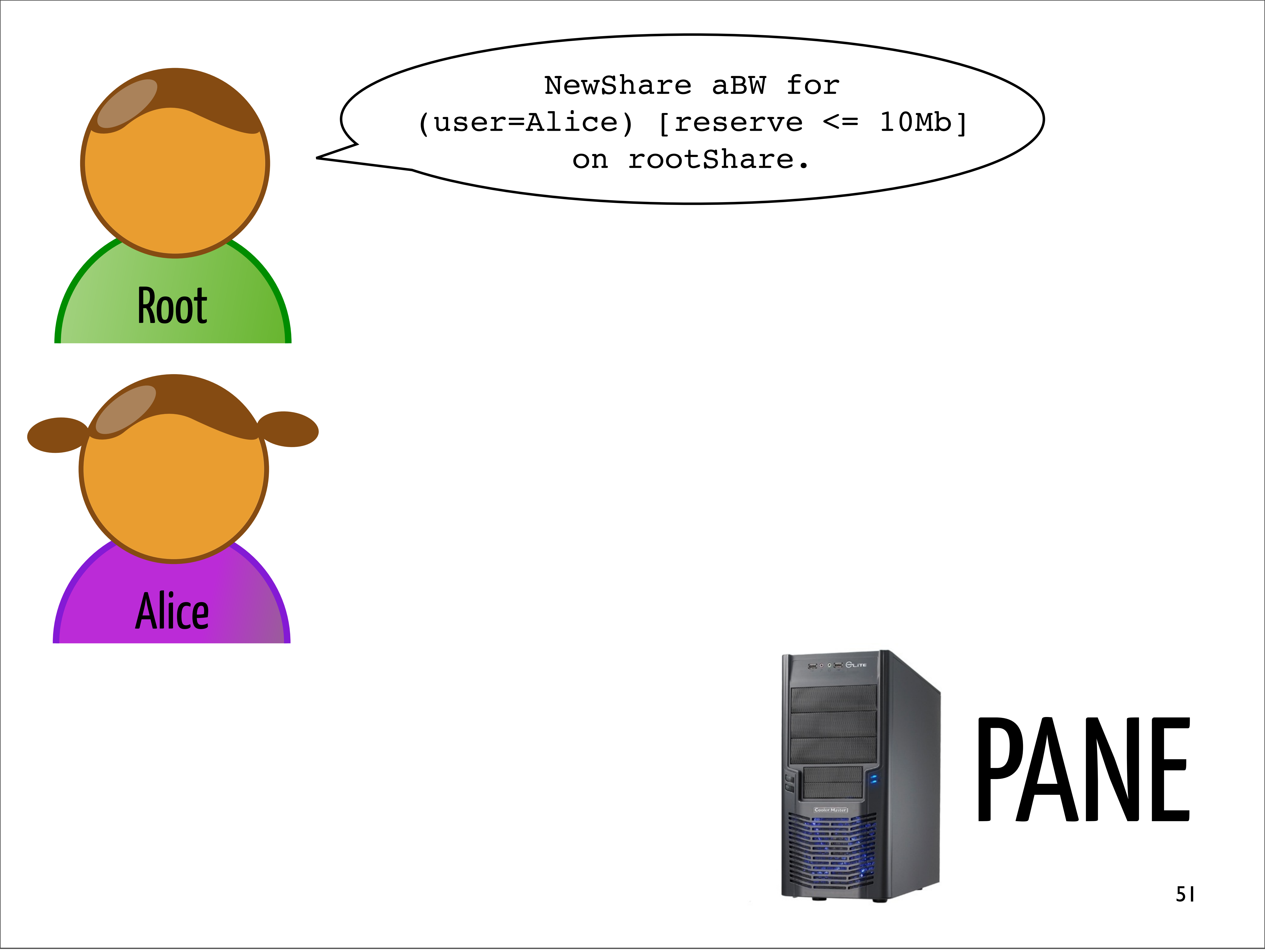
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

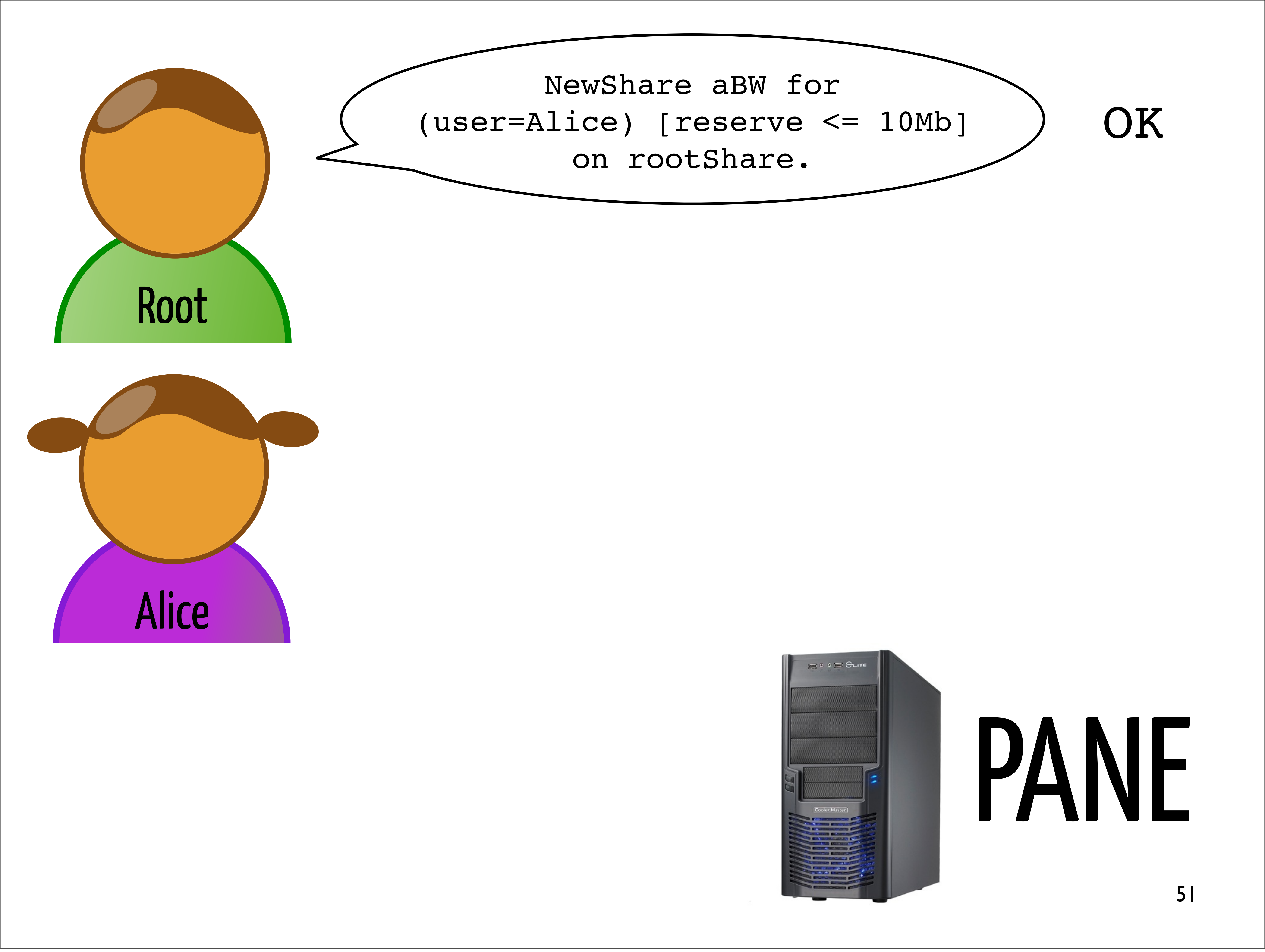
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

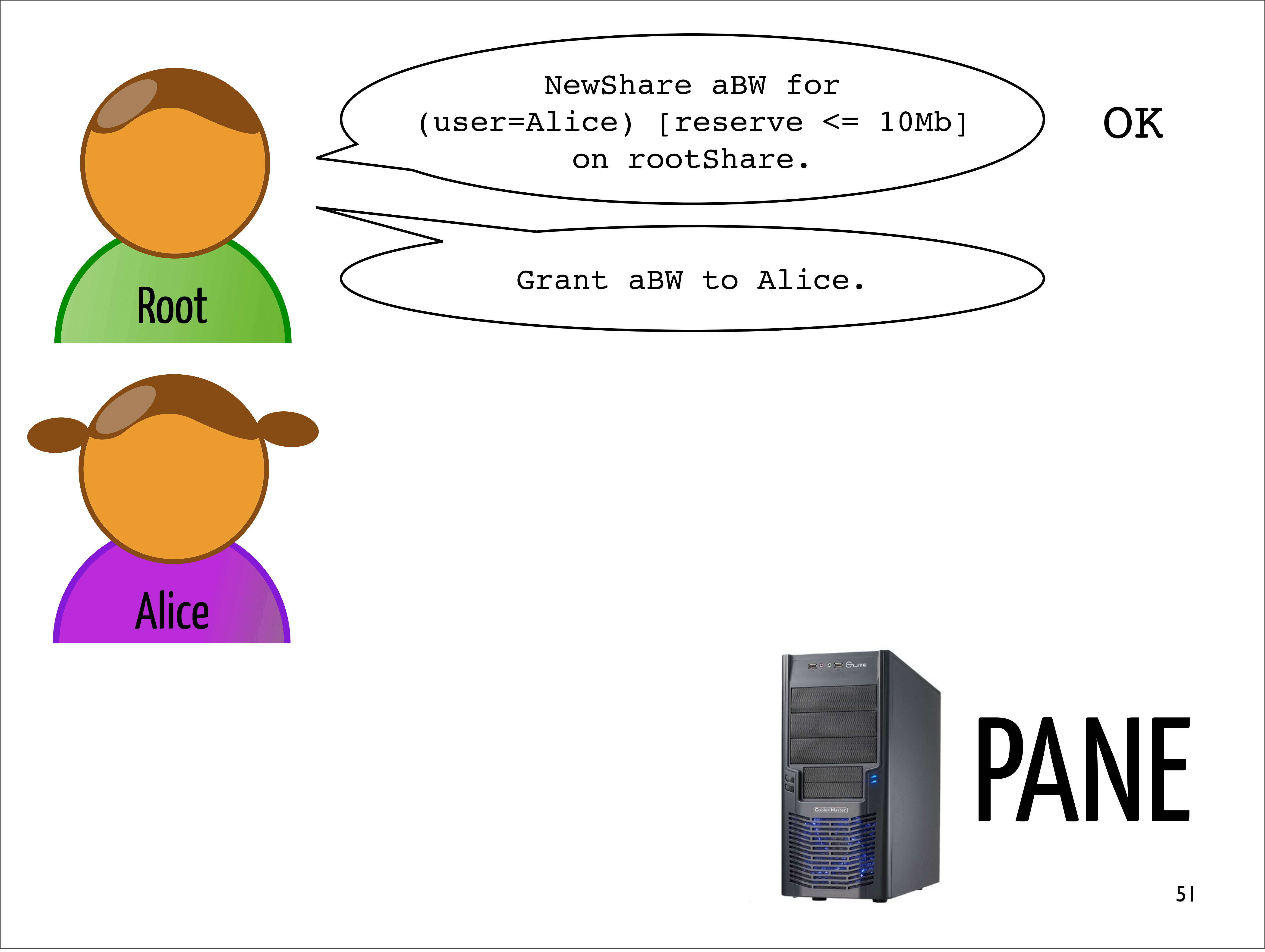
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

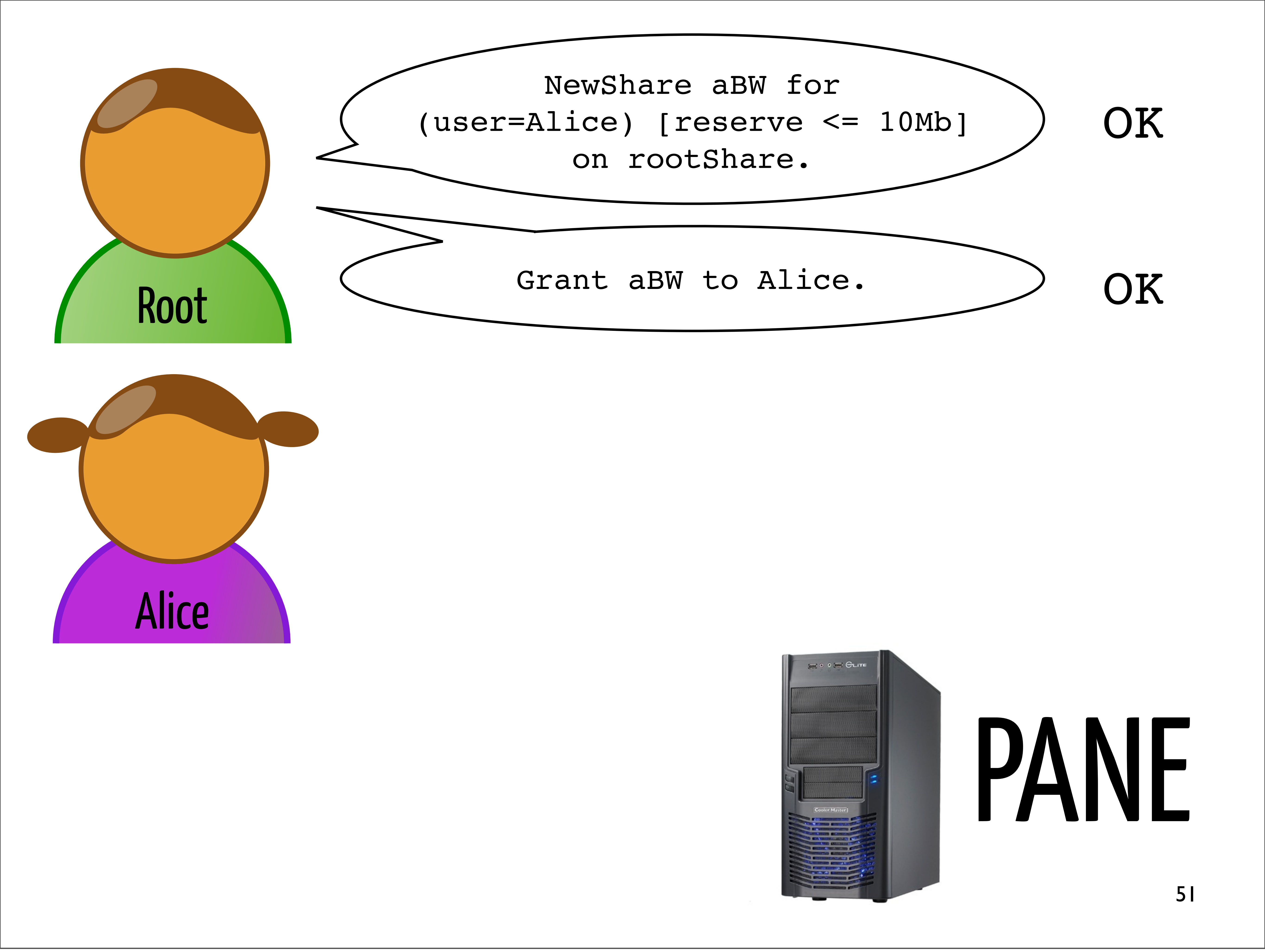
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

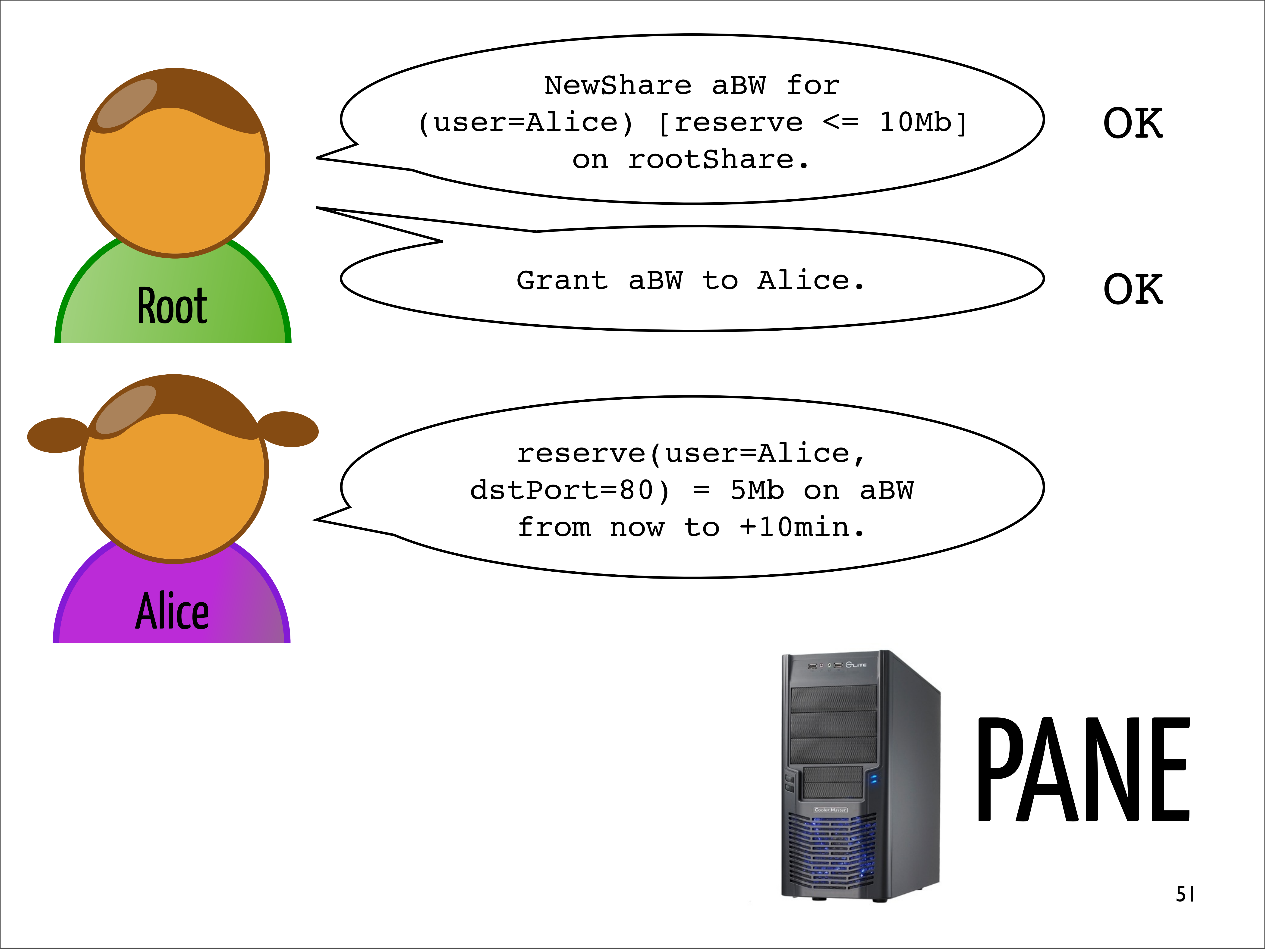
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

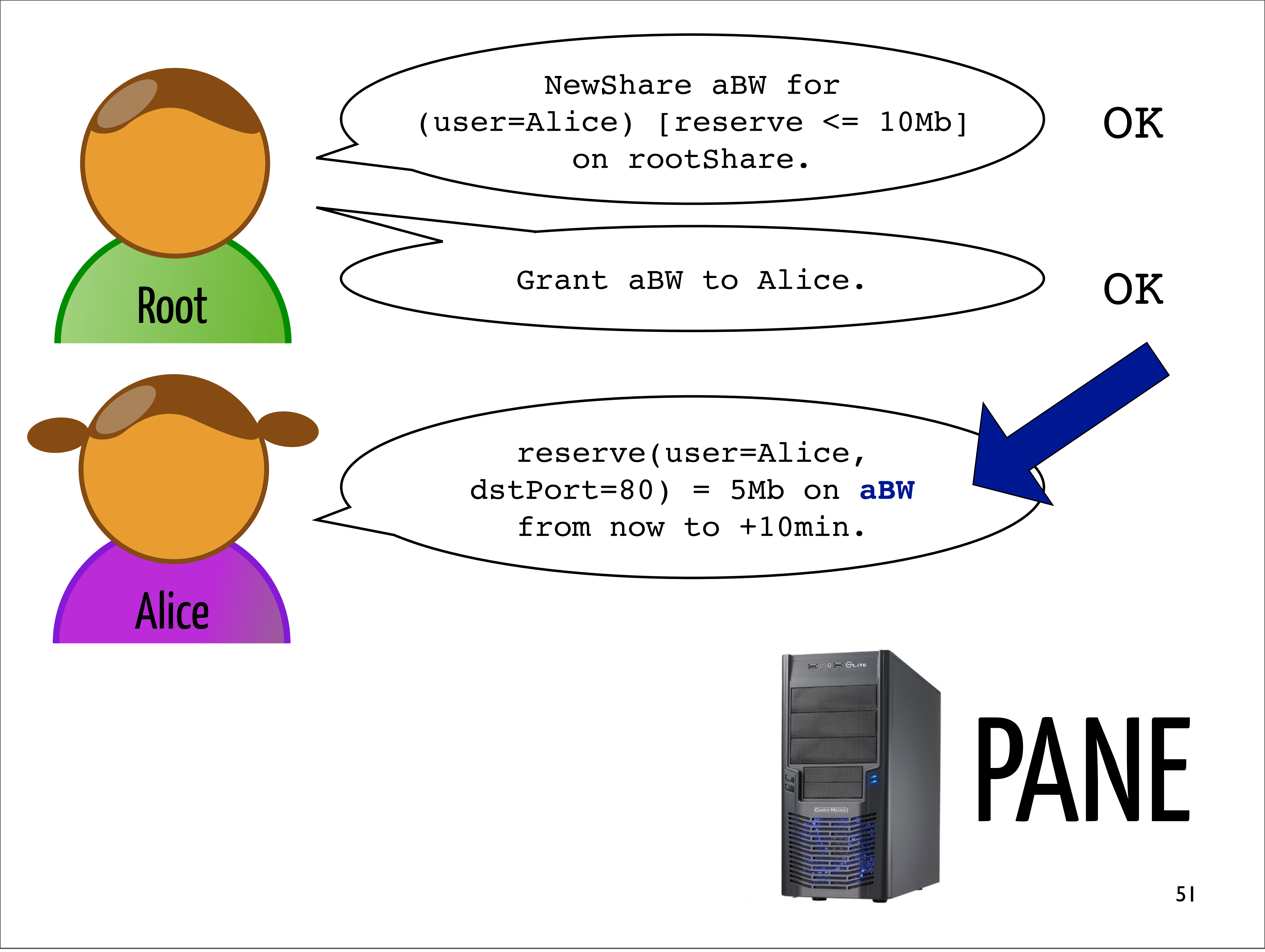
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

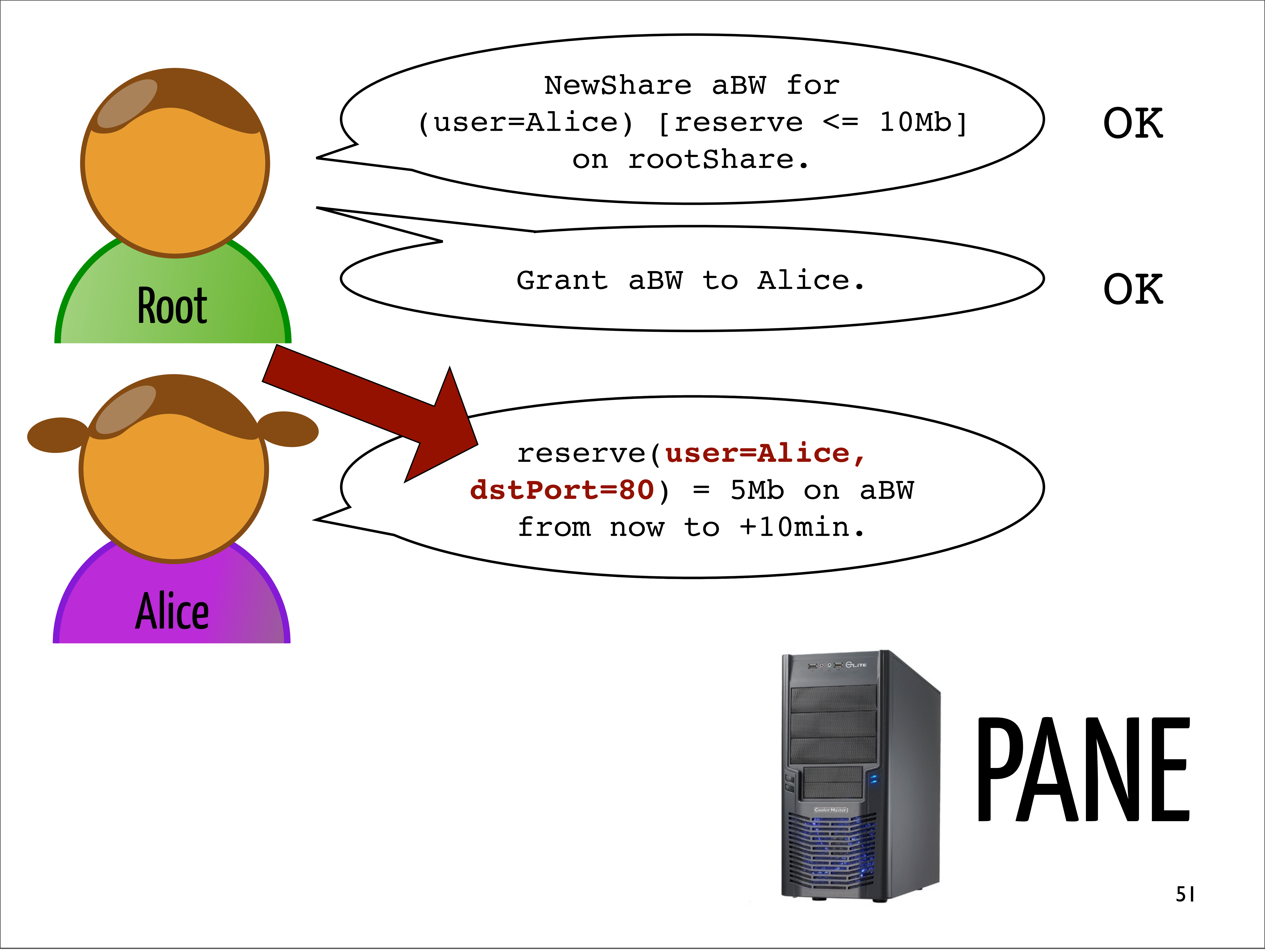
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

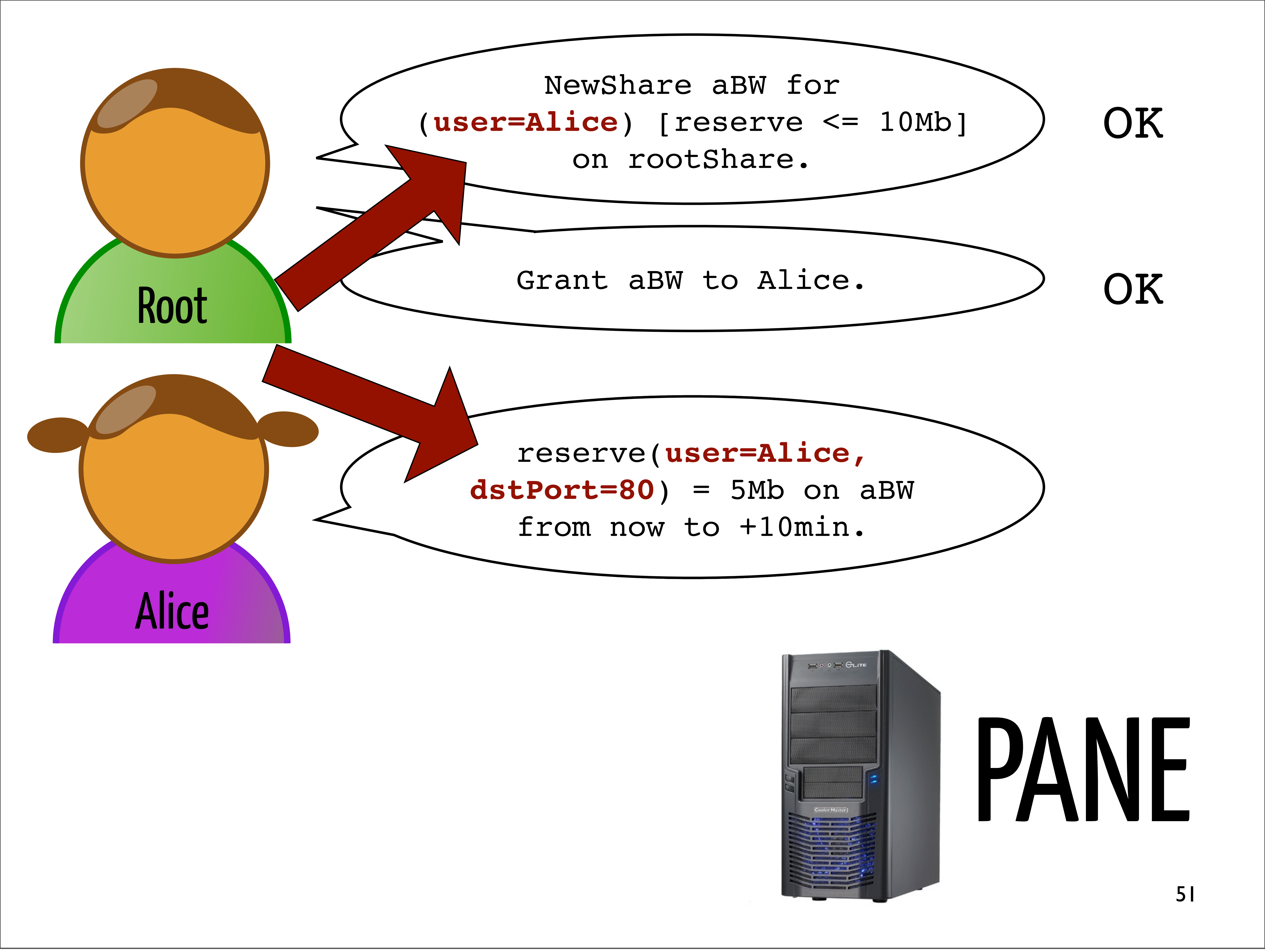
To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the



As I described earlier, the privileges in PANE derive from the root user's (click) access to the share tree.

To allow a regular user, Alice, (click) to reserve bandwidth, Root first creates a subshare with an appropriate flowgroup and privilege (click).

In this example, the subshare is for all traffic sent or received by Alice, with the authority to reserve up to 10 Mbps of guaranteed minimum bandwidth. After checking that Root has the necessary authority to create this share, the PANE controller accepts the request (click).

But Alice is not yet a principal in this share. Root must explicitly grant Alice the privilege to use the share (click). As the root user is a principal on this new share, the PANE controller accepts the command to add Alice as well (click).

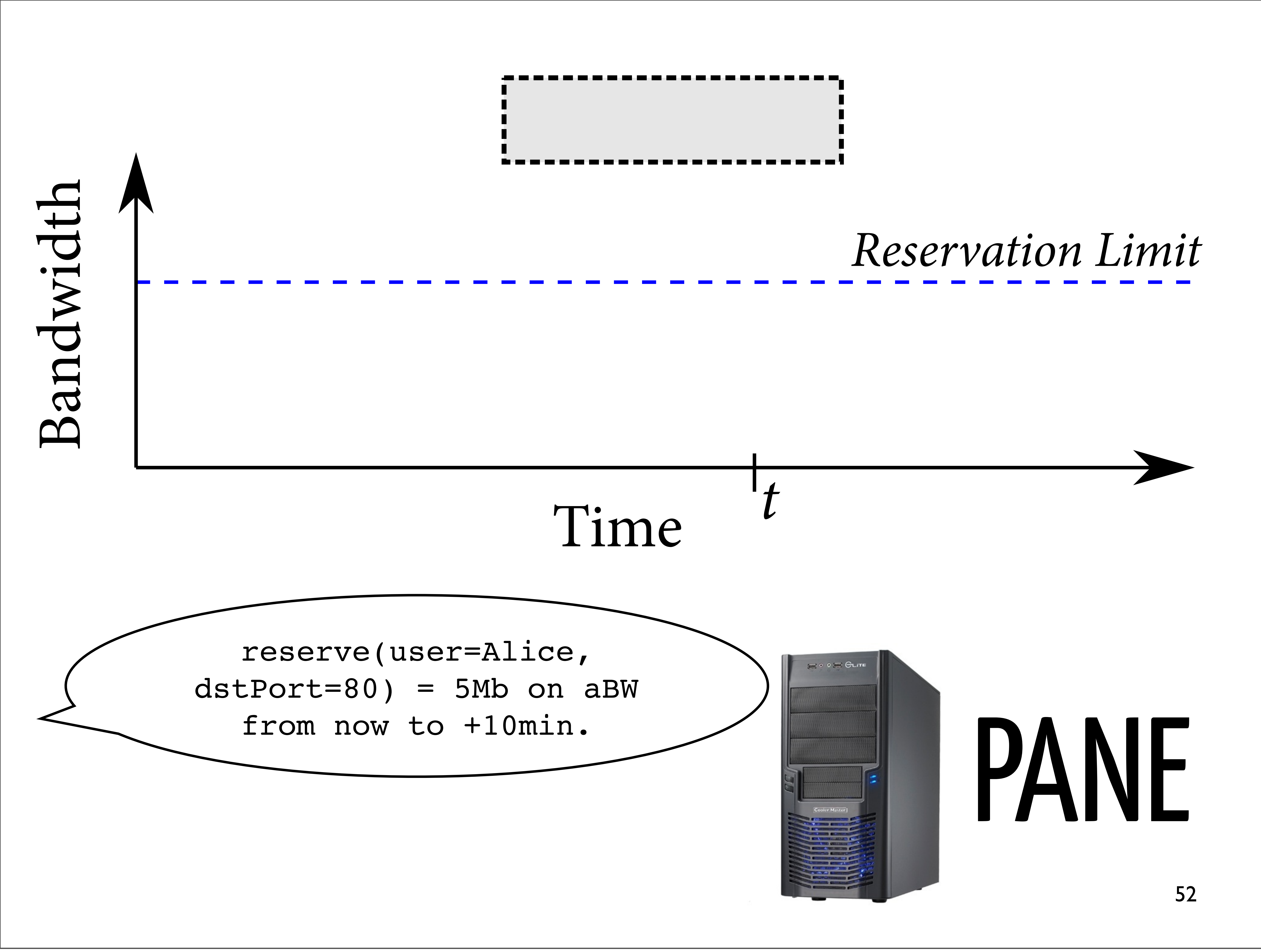
Alice now tries to make a reservation using this share (click). She requests 5 Mbps of guaranteed minimum bandwidth for the next 10 minutes. Her message explicitly indicates which share she is using to make the request (click).

The PANE controller first checks that the FlowGroup on the request (click) is a subset of the

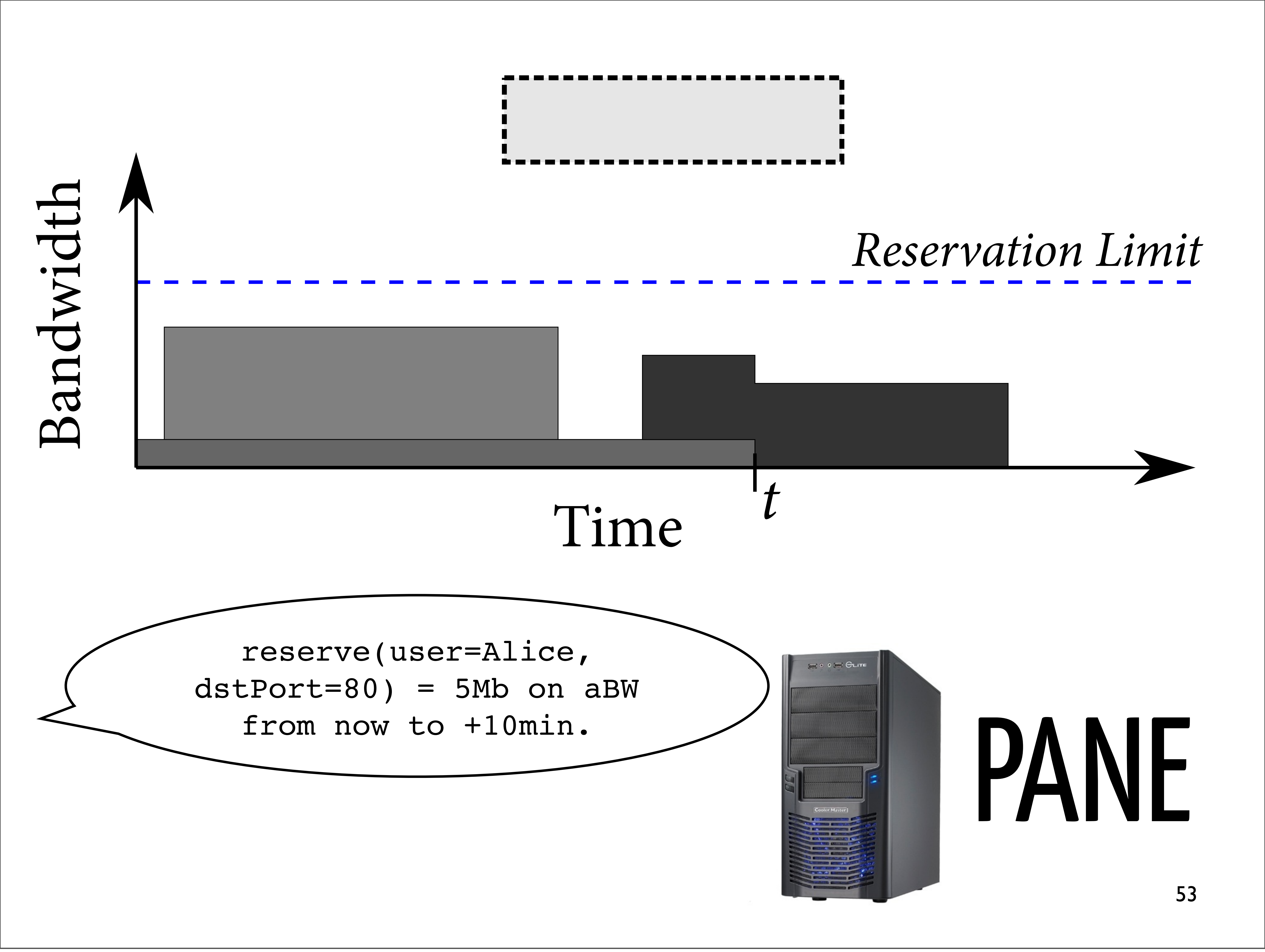
reserve(user=Alice,
dstPort=80) = 5Mb on aBW
from now to +10min.

PANE

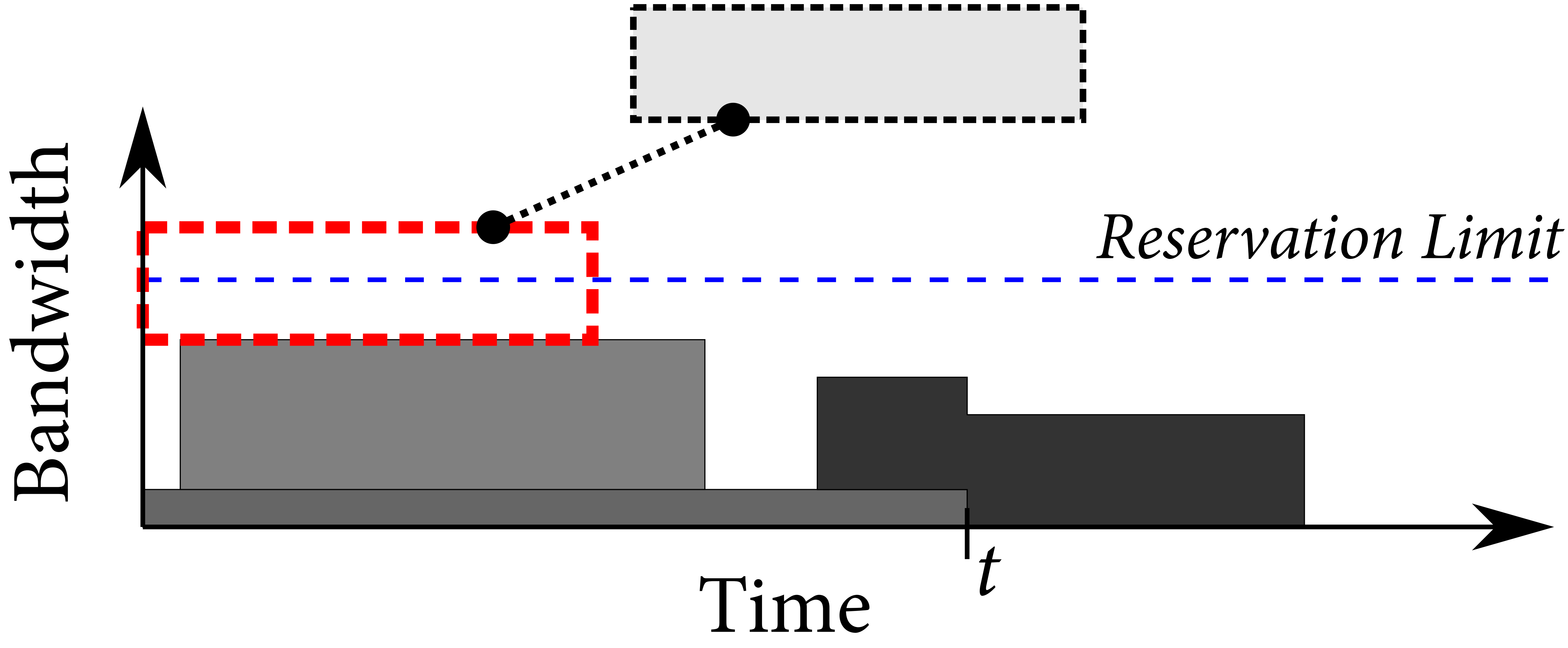
… next examines the schedule of accepted reservations in the aBW share (click). As there are currently no reservations ...

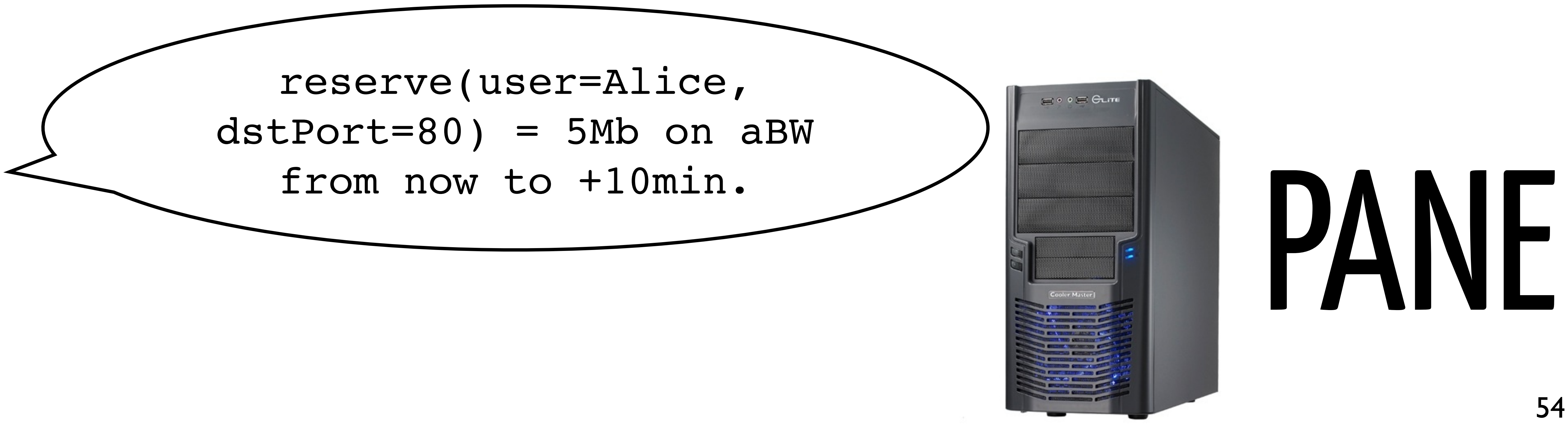


… next examines the schedule of accepted reservations in the aBW share (click). As there are currently no reservations ...

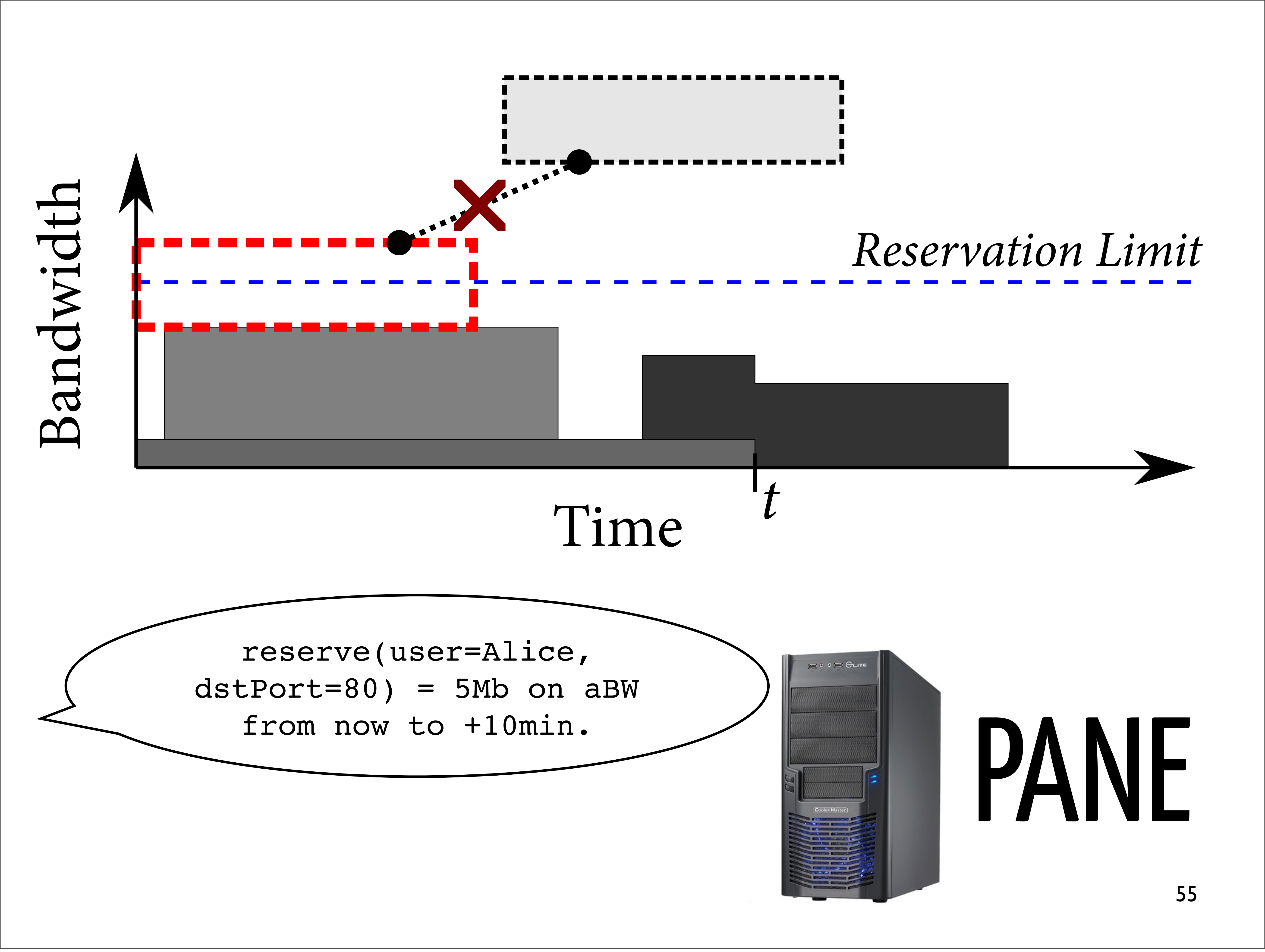


… the controller then recursively checks for other reservations up the share tree.



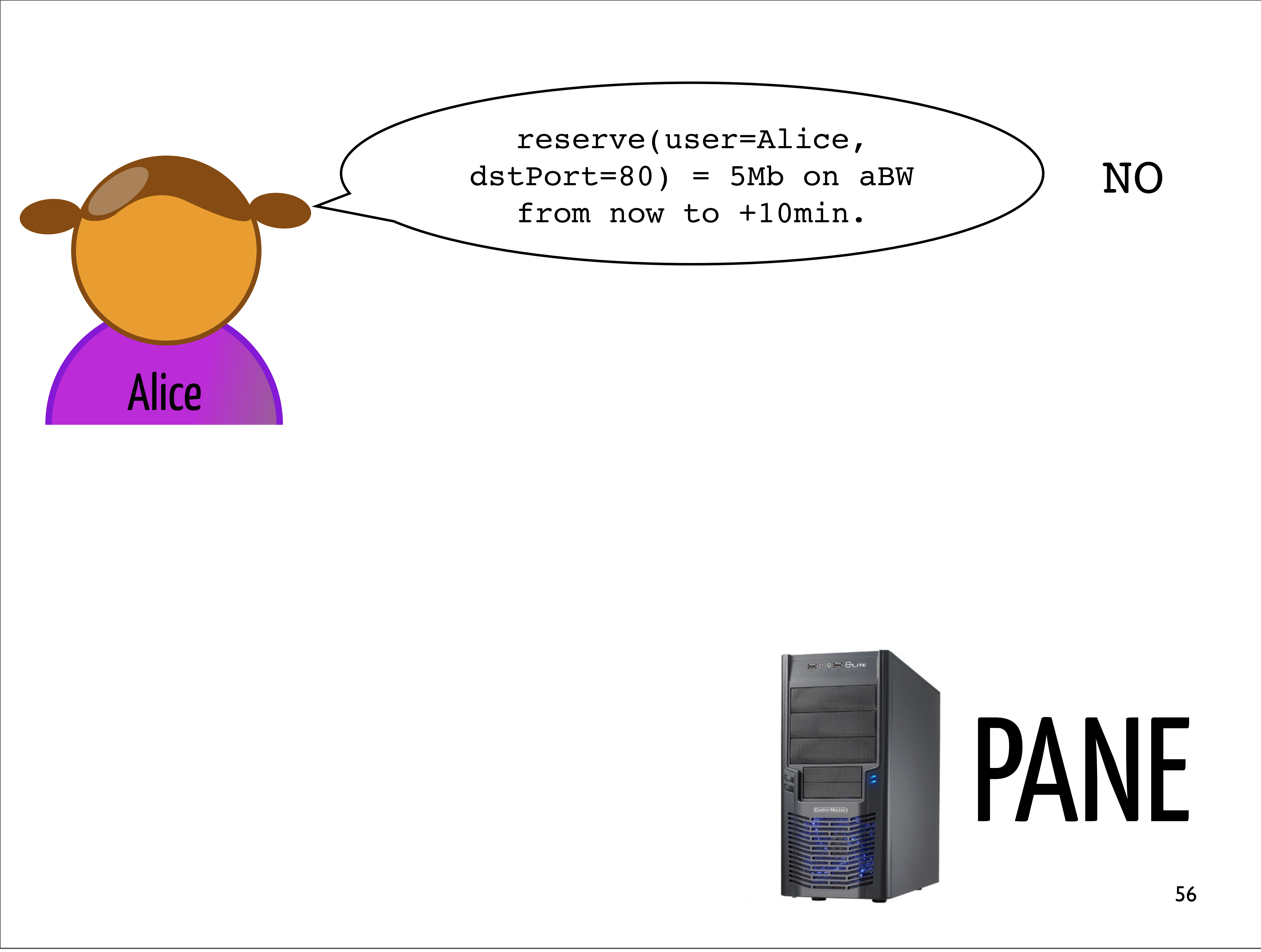


When the controller tries to install the reservation…

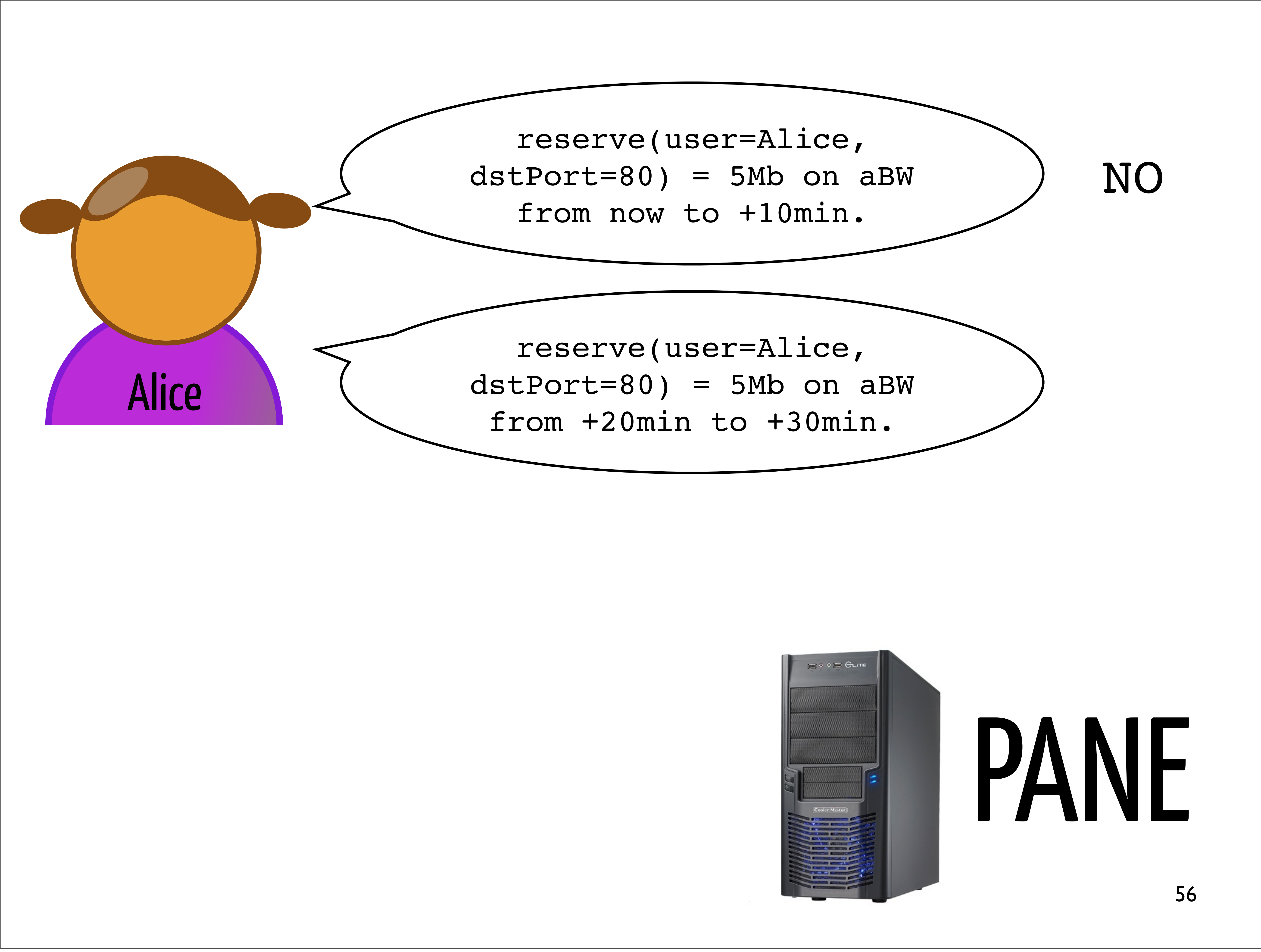


… it detects a conflict with the existing reservations.

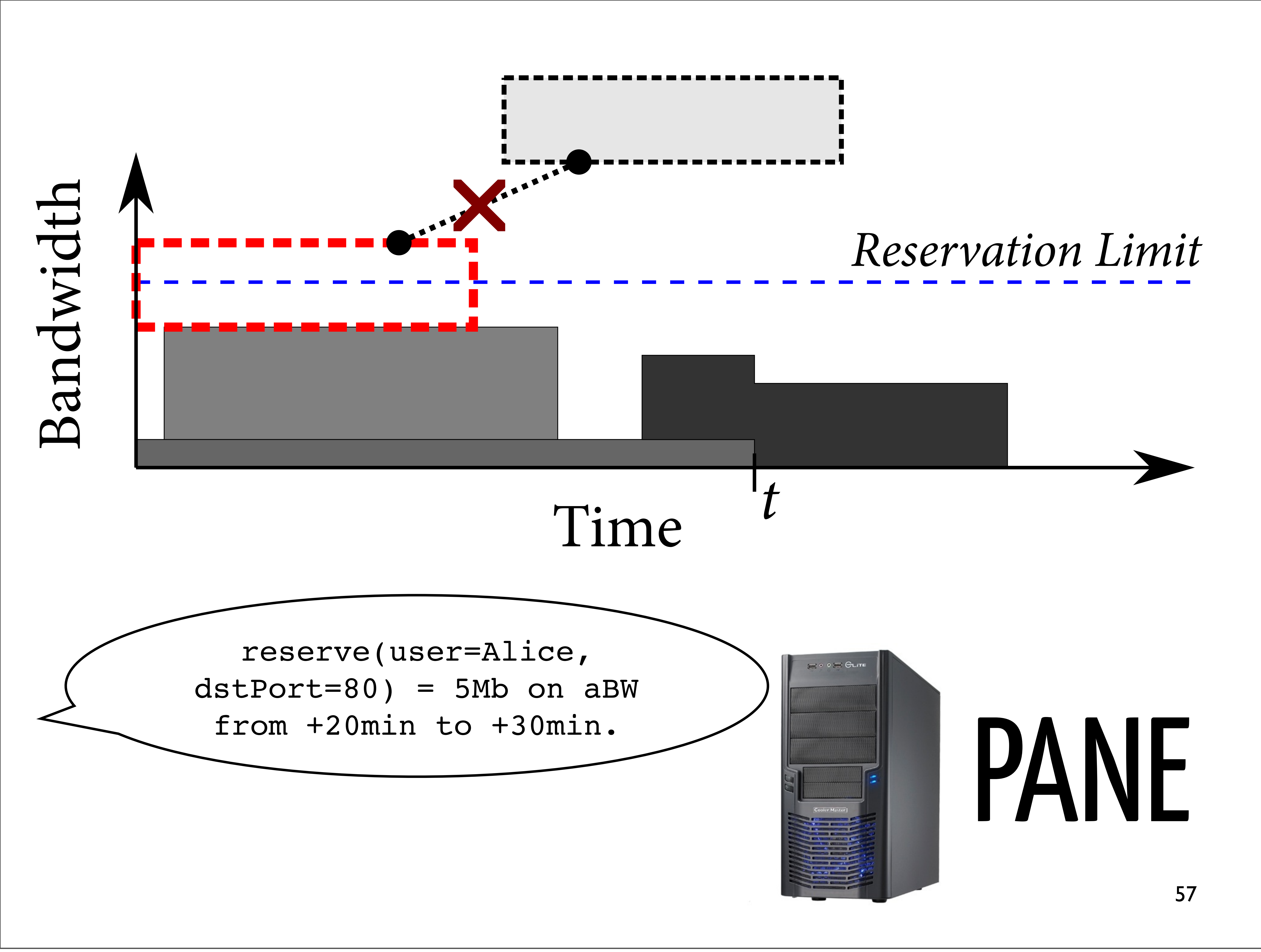
(Pause)



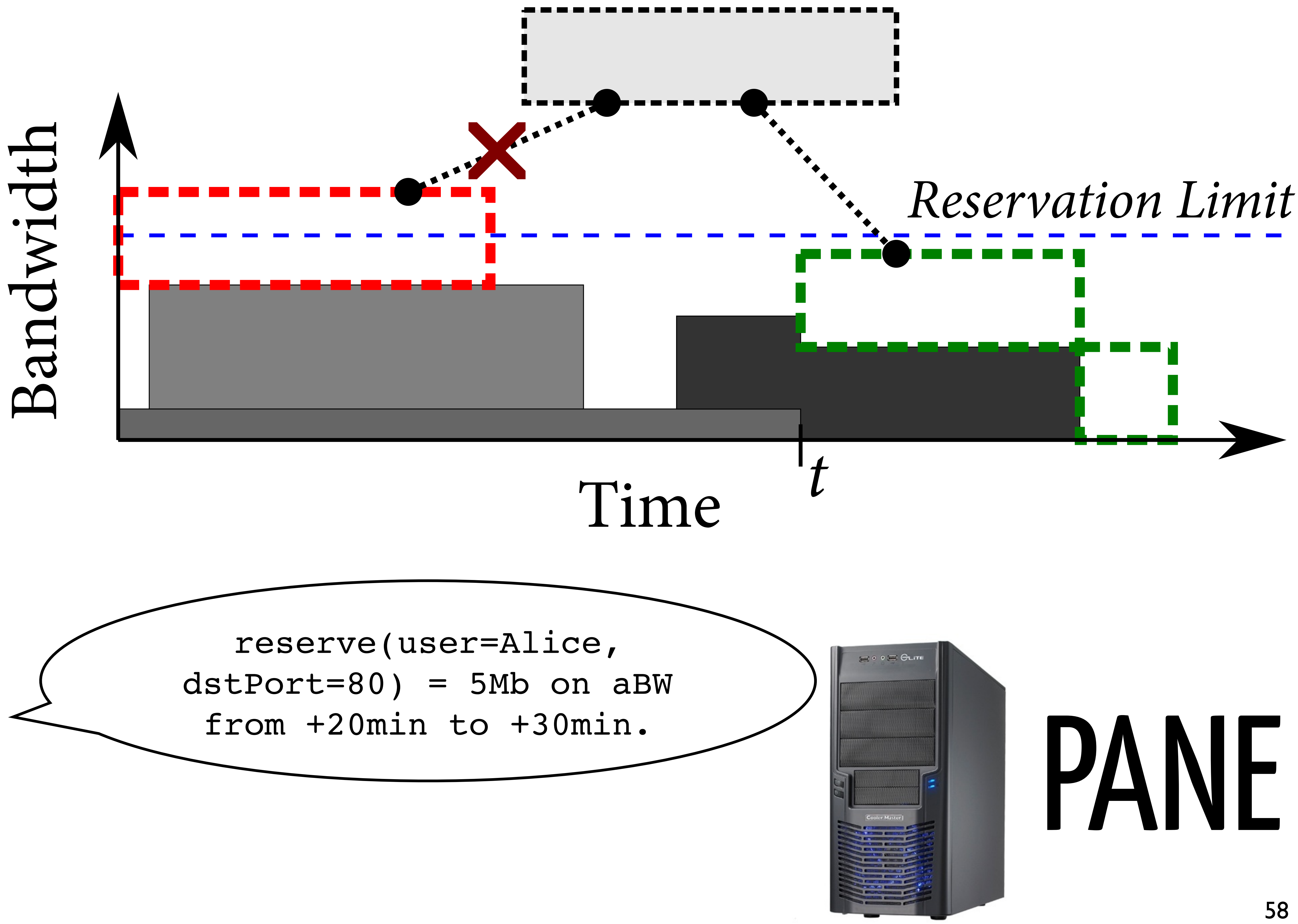
Therefore, the controller denies Alice’s initial request. Next, Alice retrieves the schedule of accepted requests from the controller, and creates a new request (click) for the same bandwidth, now starting 20 minutes in the future.



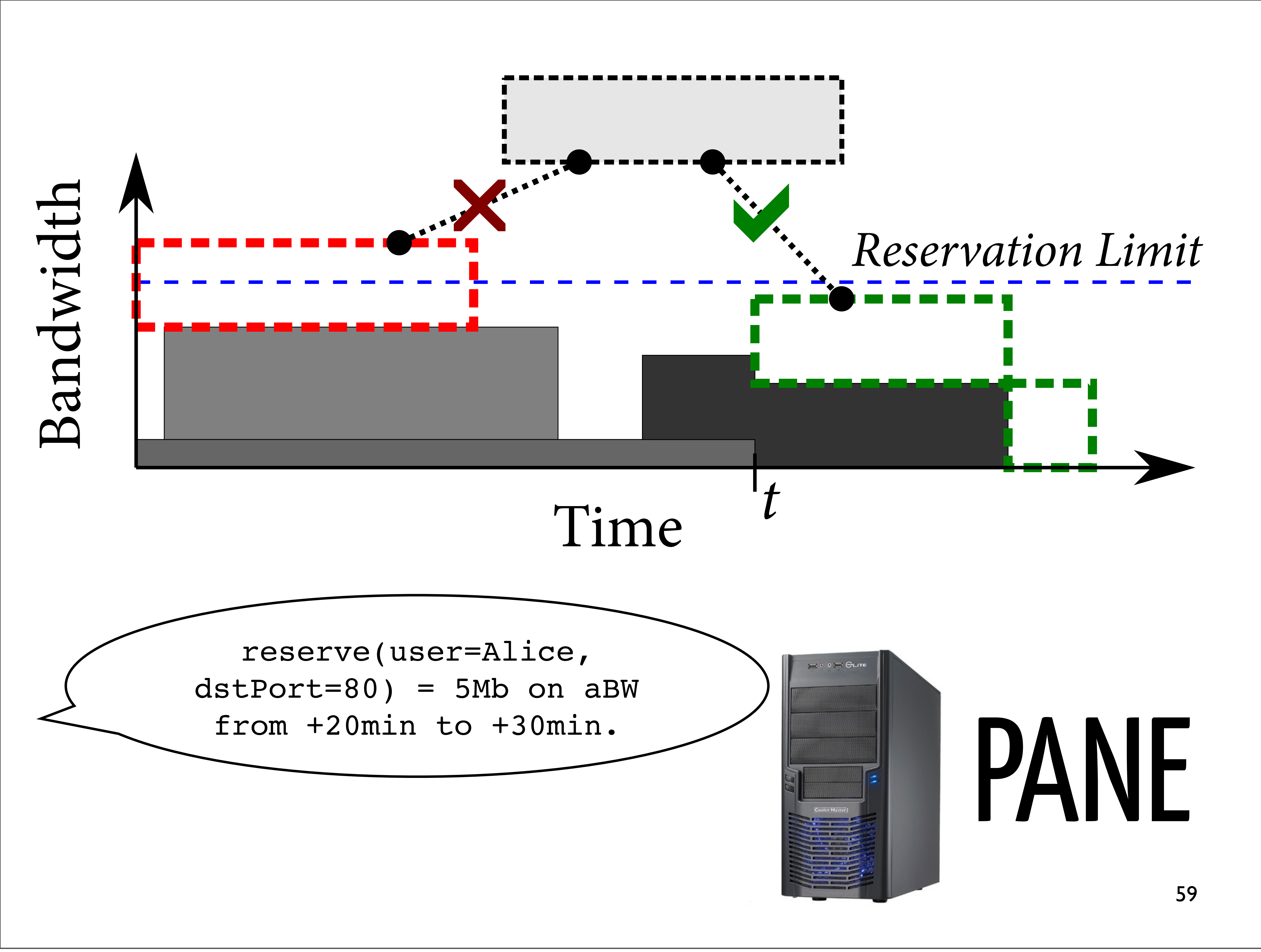
Therefore, the controller denies Alice’s initial request. Next, Alice retrieves the schedule of accepted requests from the controller, and creates a new request (click) for the same bandwidth, now starting 20 minutes in the future.



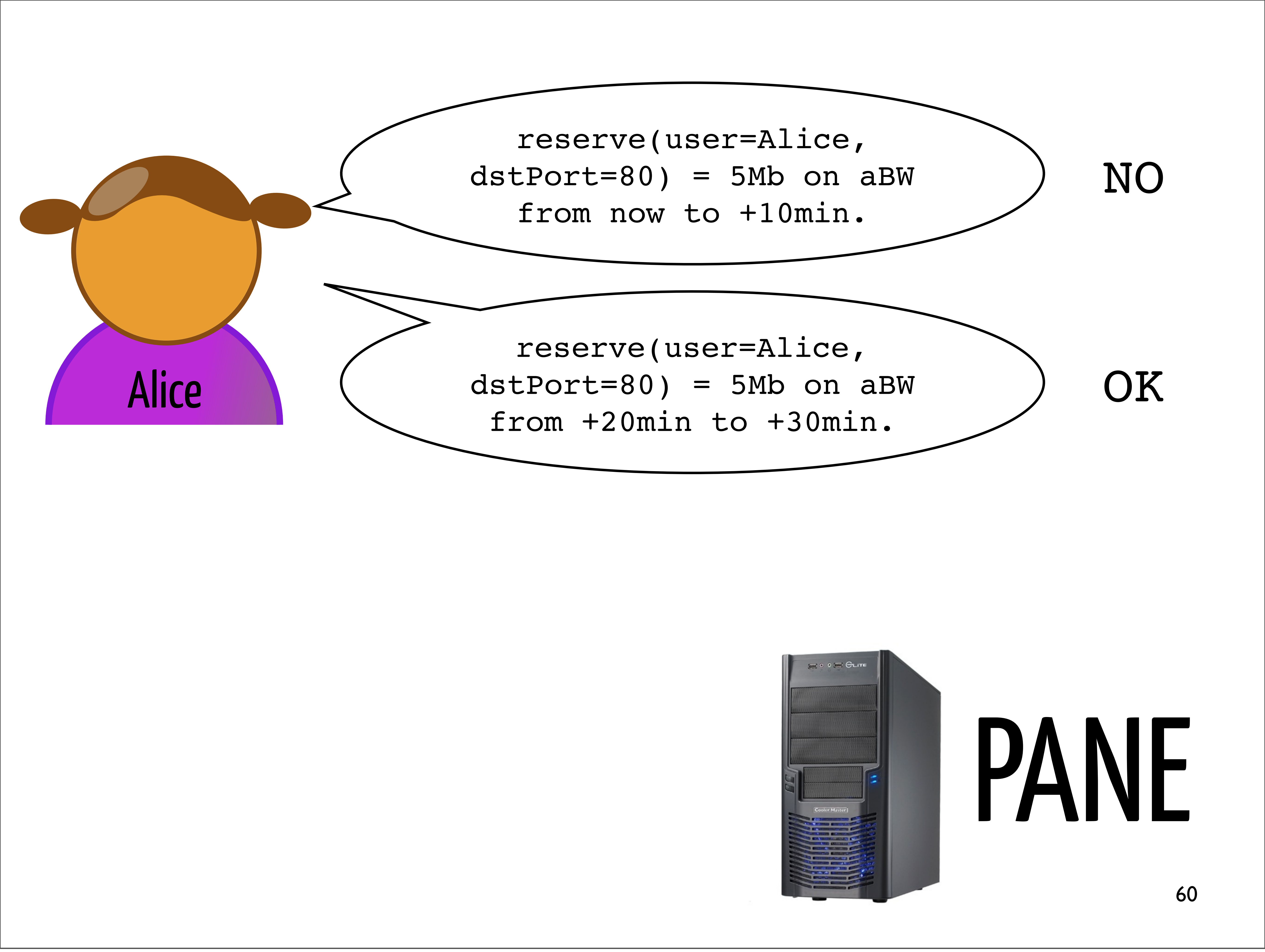
The controller takes the new request ...



… and checks if it can be installed at the new time.



Because accepting this reservation would no longer exceed the limit ...



the controller returns a successful confirmation to Alice. When the reservation begins in 20 minutes, the PANE controller will establish the appropriate queues on the switches and provide Alice’s traffic with 5 Mbps of guaranteed minimum bandwidth.

(Pause)

Let’s now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice’s computer …

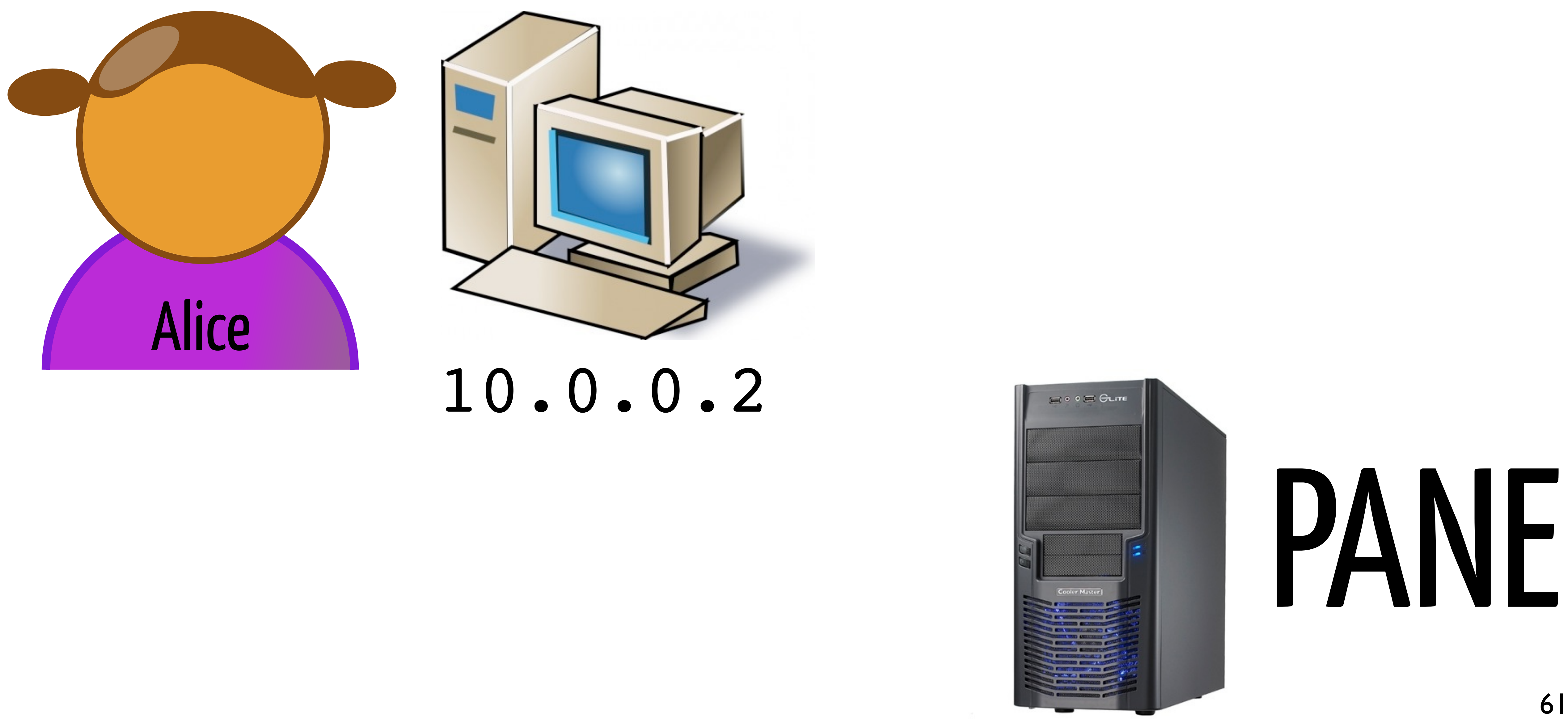
PANE

Let’s now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice’s computer …

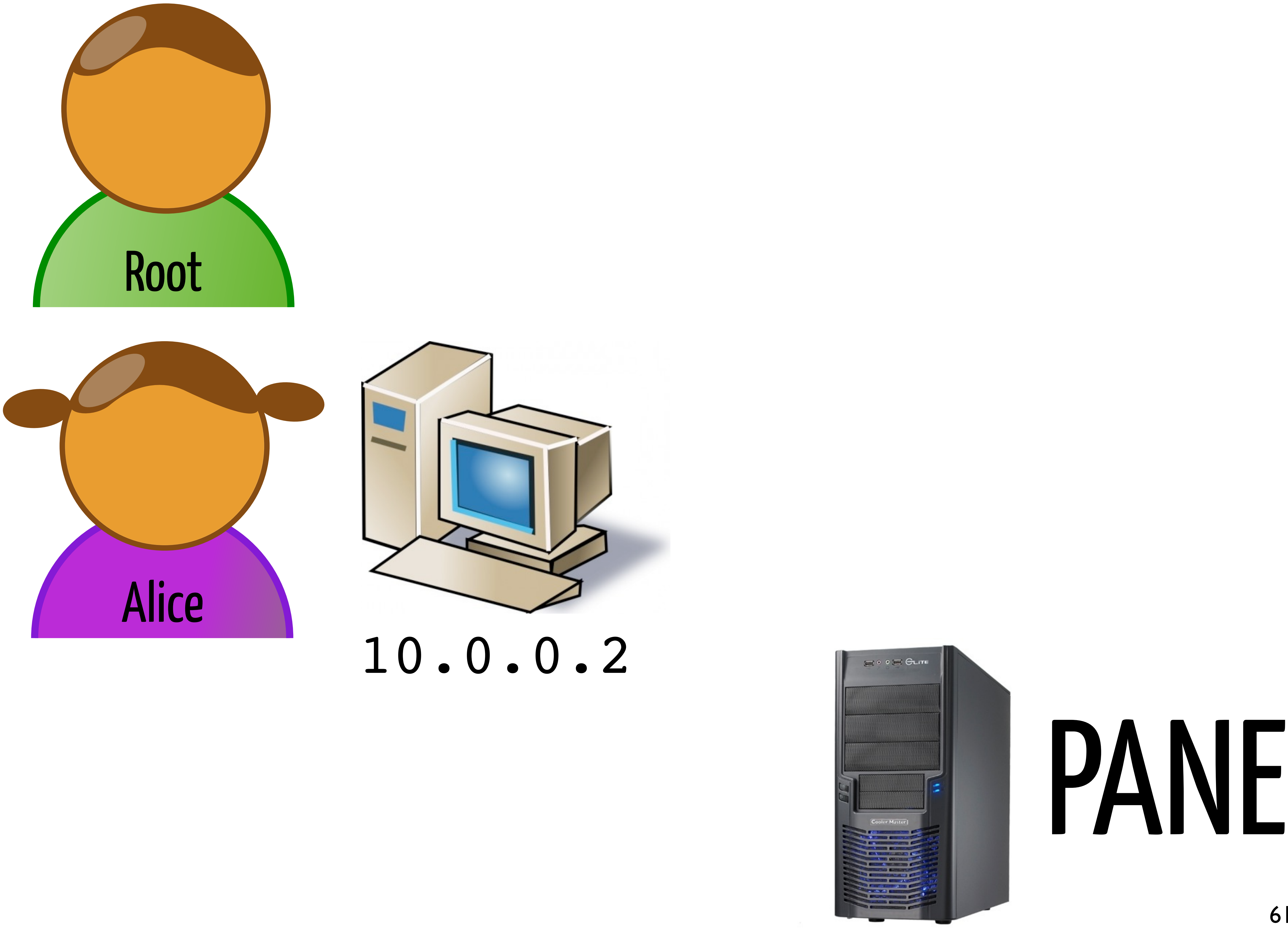


Let’s now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice’s computer …

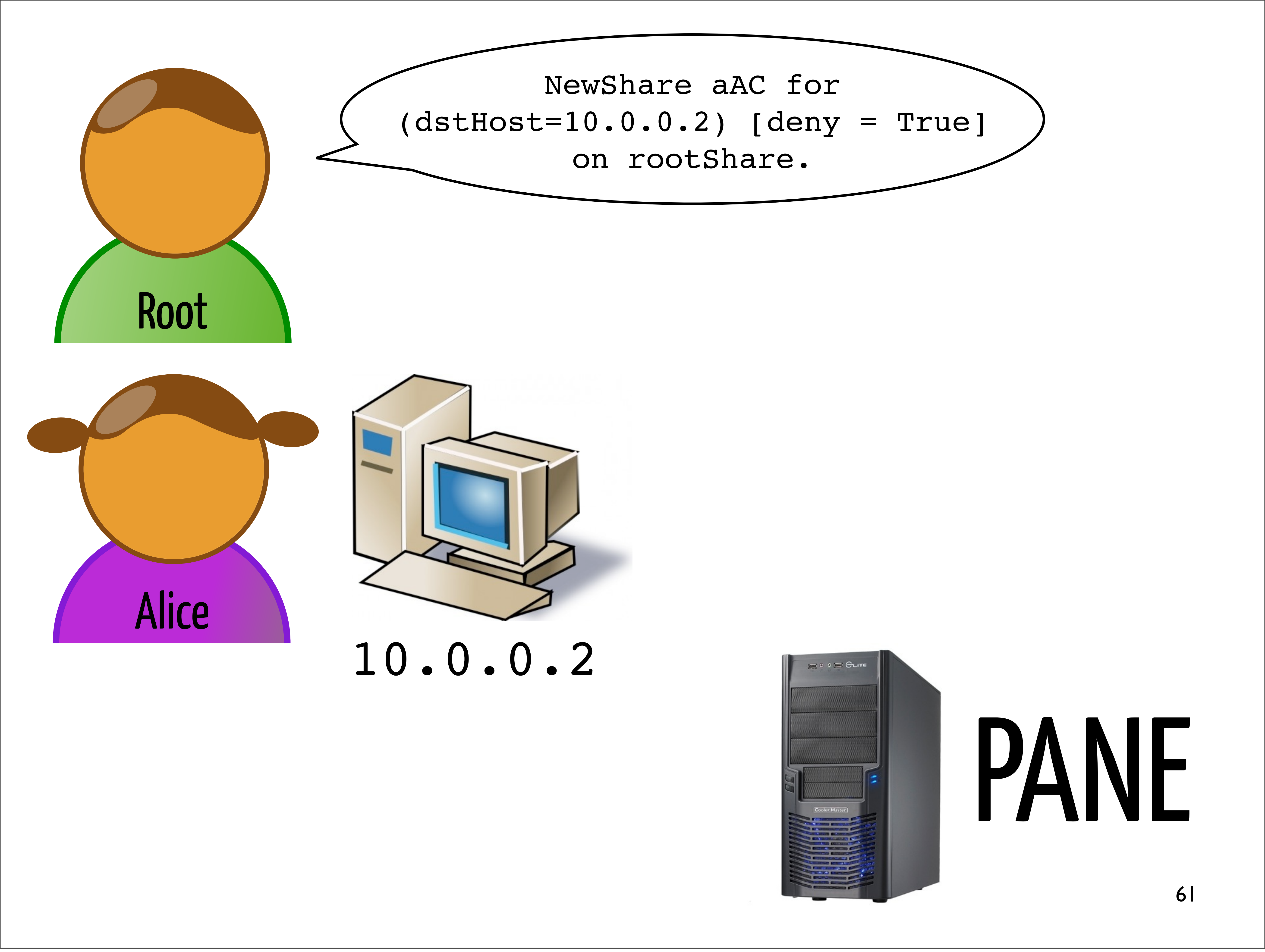


Let’s now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice’s computer …

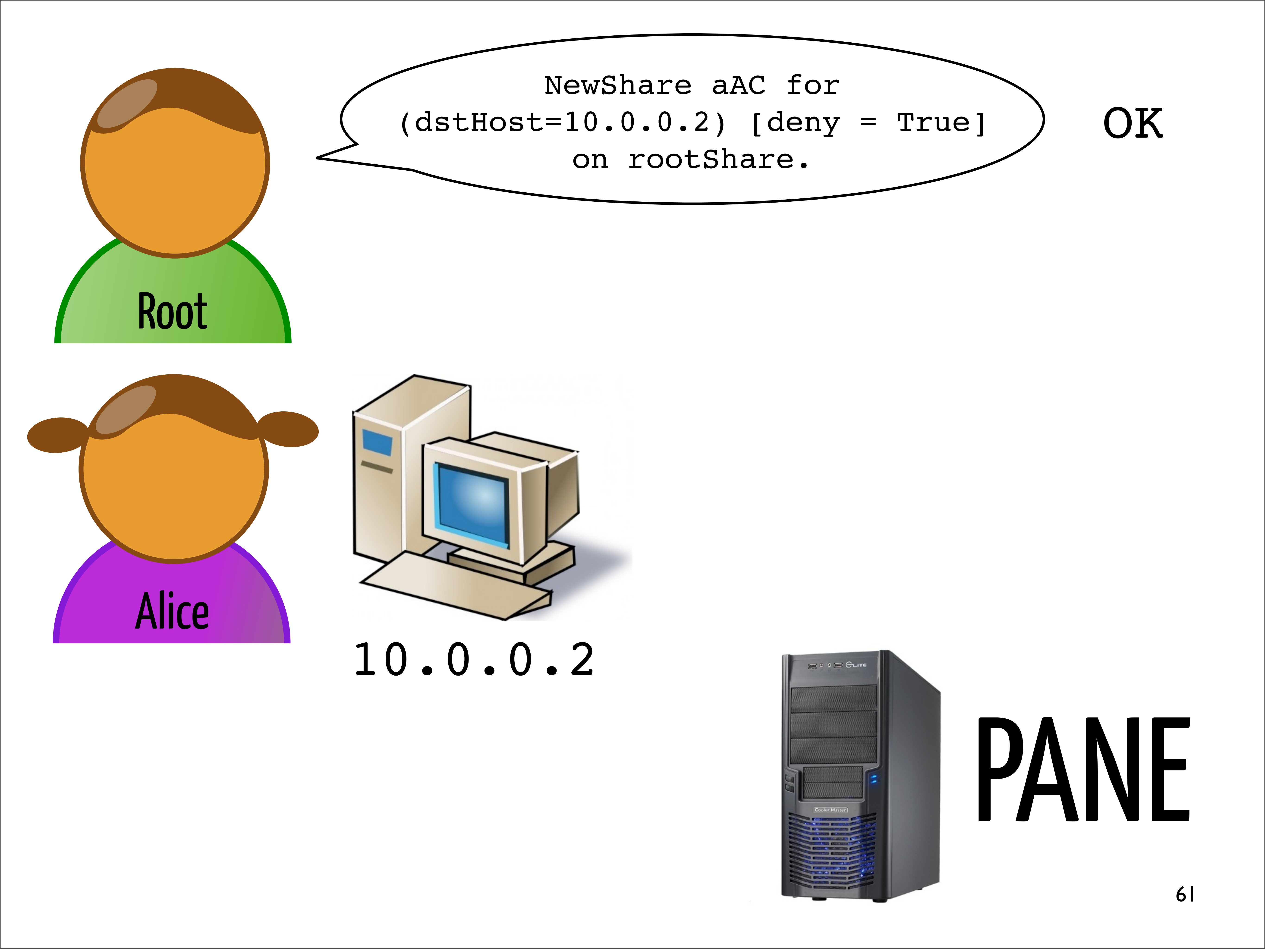


Let’s now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice’s computer …

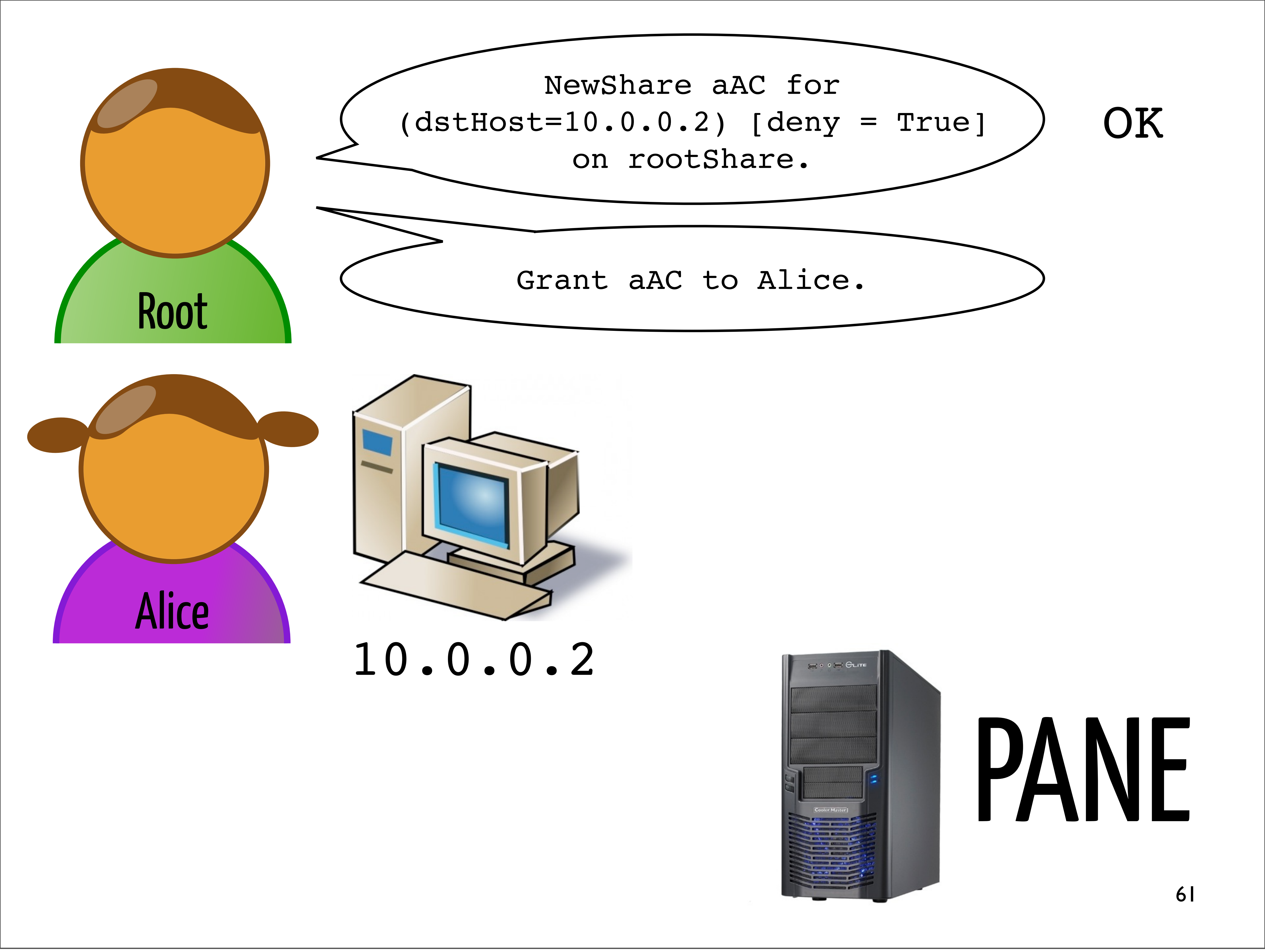


Let's now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice's computer …

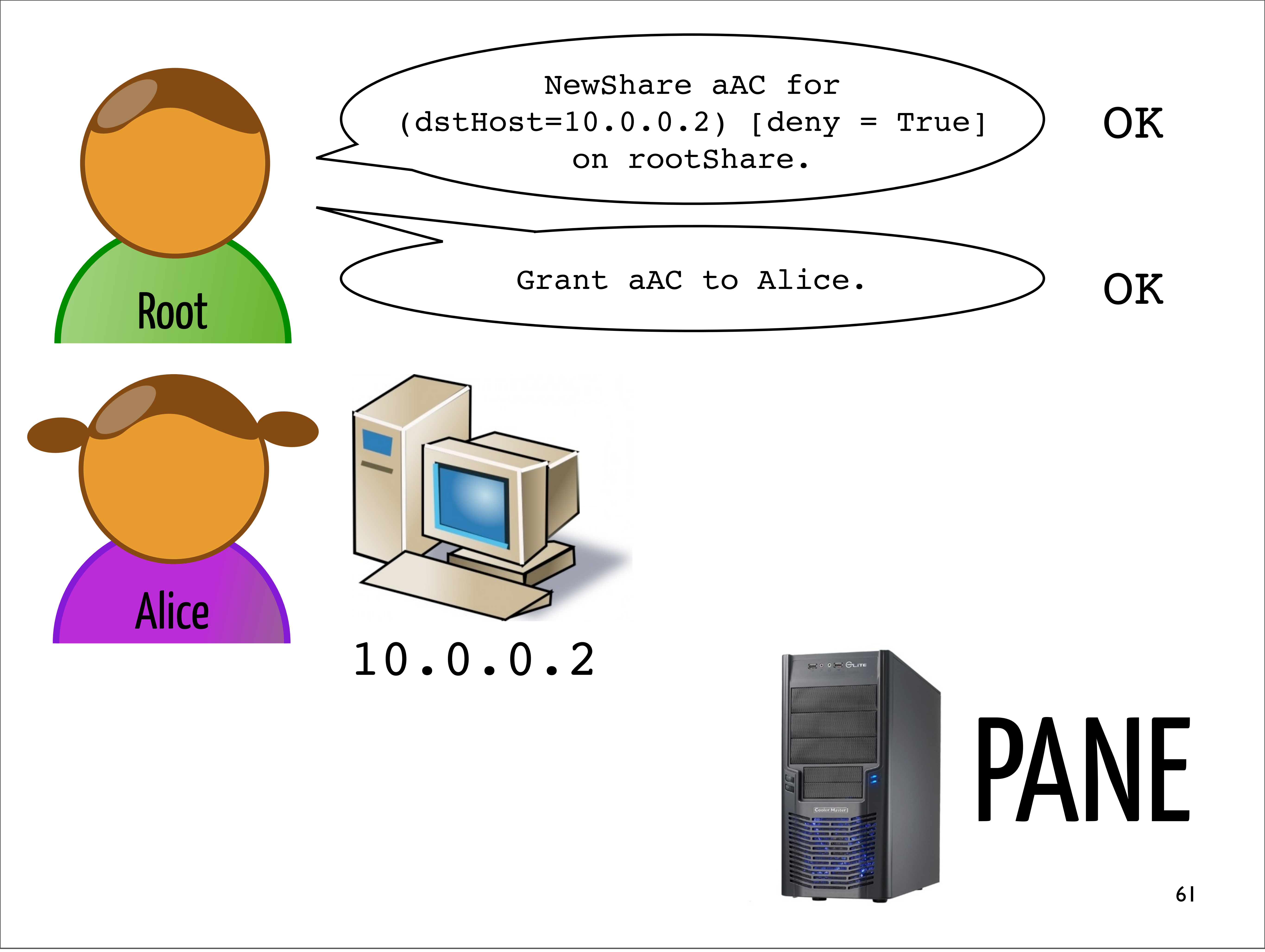


Let's now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice's computer …

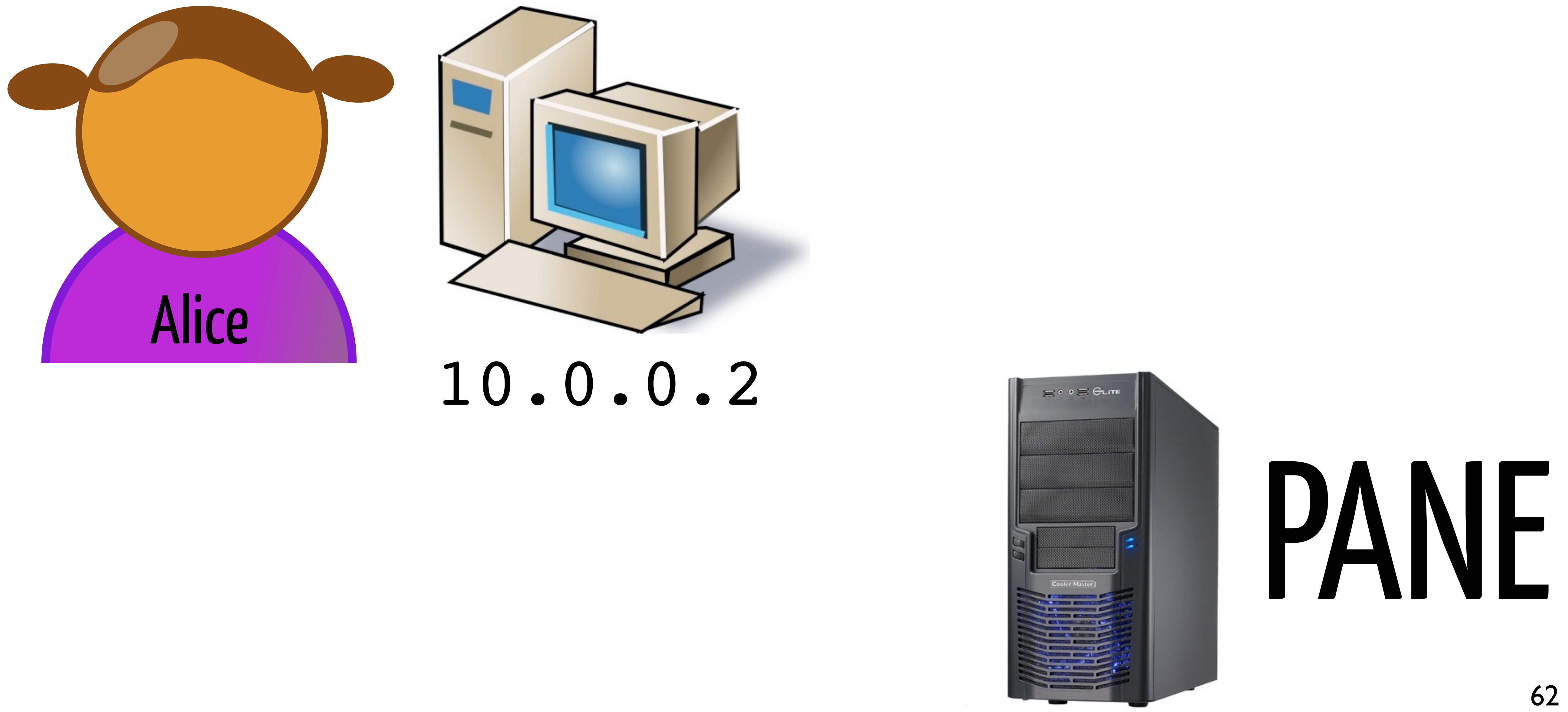


Let’s now consider a second example. If Alice (click) wants to block some traffic to her computer (click), she can ask the root user (click) to create a subshare (click) for her with the deny privilege (click).

After creating this share, the root user grants use of the share (click) to Alice, as we saw previously (click).

(Pause)

If Alice’s computer …



… is being attacked by Eve (click), she can send a deny request (click) to the PANE controller to have Eve's traffic blocked for the next five minutes.

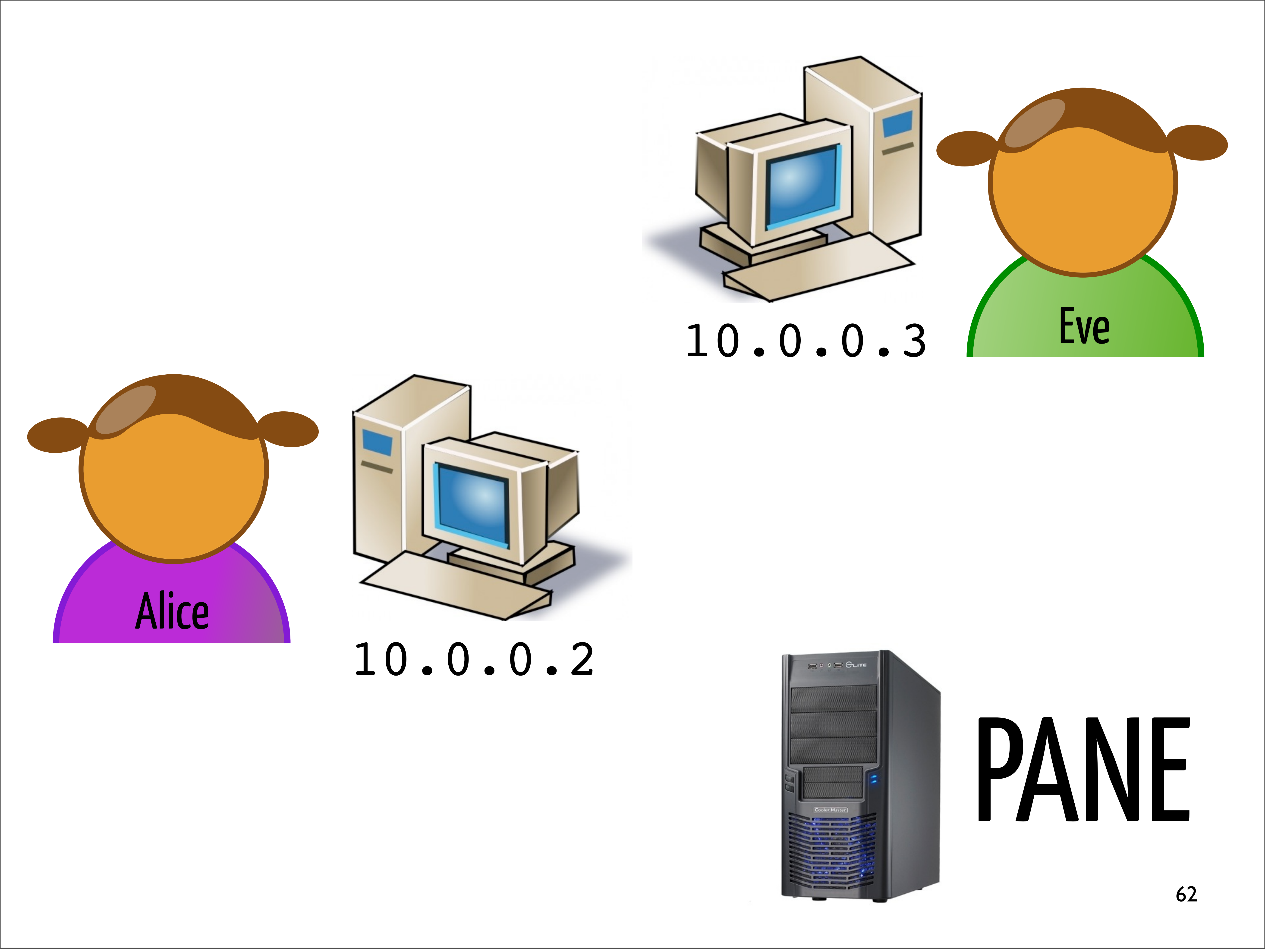
Because Alice was previously granted this authority, the PANE controller accepts her request (click), and uses OpenFlow to reconfigure the switches and block traffic from Eve's computer destined to Alice's (click).

If Alice tried to block Eve's traffic to another computer by changing the dstHost parameter on her request, the request would be denied as the flow would no longer be contained within the FlowGroup of the aAC subshare. (possibly make this its own slide?)

(Pause)

This has been a short sample of the PANE protocol. Our prototype supports several additional commands, for example, to establish rate-limits, manage users, and query the state of the ShareTree.

(Pause)



… is being attacked by Eve (click), she can send a deny request (click) to the PANE controller to have Eve's traffic blocked for the next five minutes.

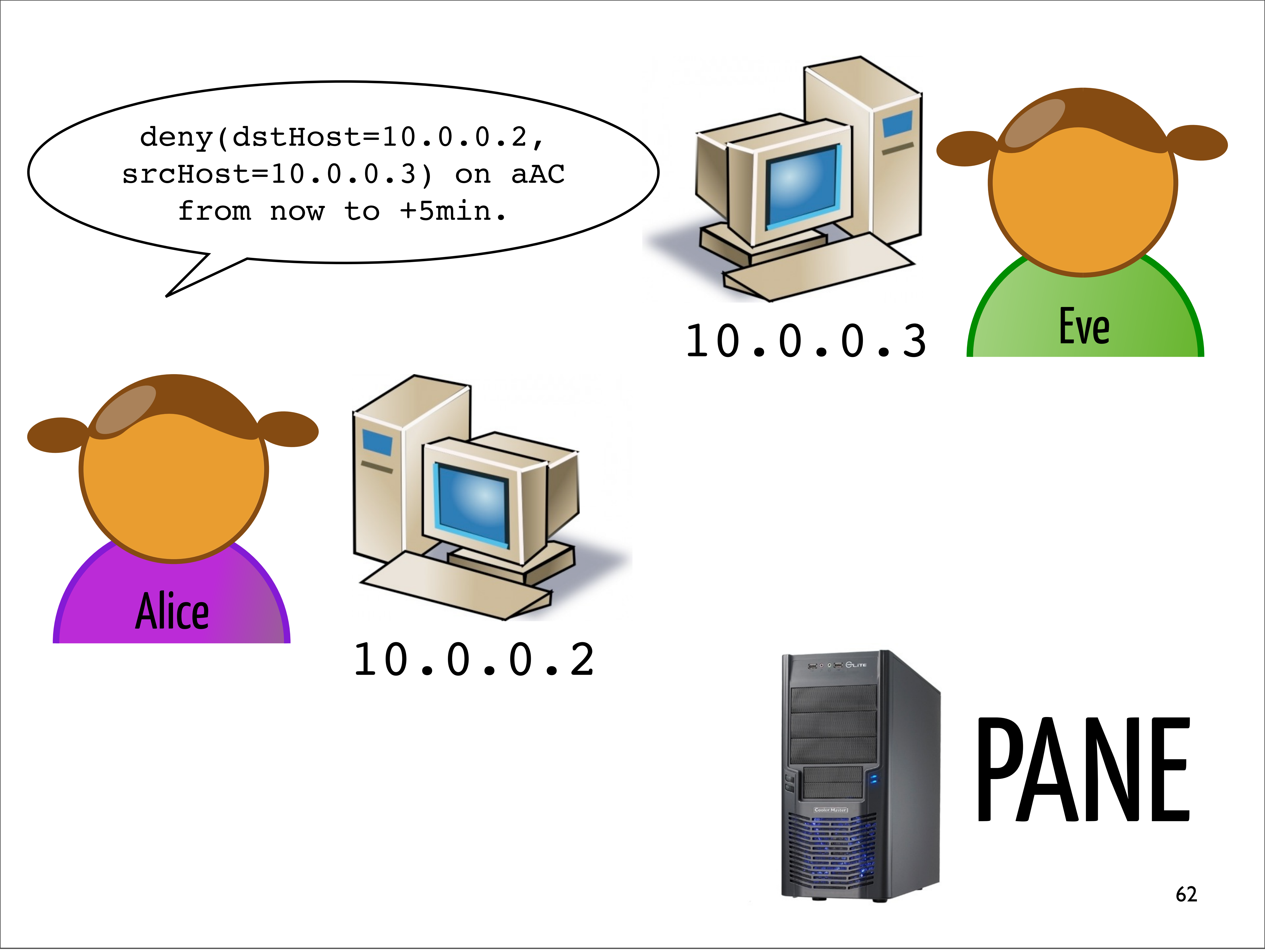
Because Alice was previously granted this authority, the PANE controller accepts her request (click), and uses OpenFlow to reconfigure the switches and block traffic from Eve's computer destined to Alice's (click).

If Alice tried to block Eve's traffic to another computer by changing the dstHost parameter on her request, the request would be denied as the flow would no longer be contained within the FlowGroup of the aAC subshare. (possibly make this its own slide?)

(Pause)

This has been a short sample of the PANE protocol. Our prototype supports several additional commands, for example, to establish rate-limits, manage users, and query the state of the ShareTree.

(Pause)



… is being attacked by Eve (click), she can send a deny request (click) to the PANE controller to have Eve's traffic blocked for the next five minutes.

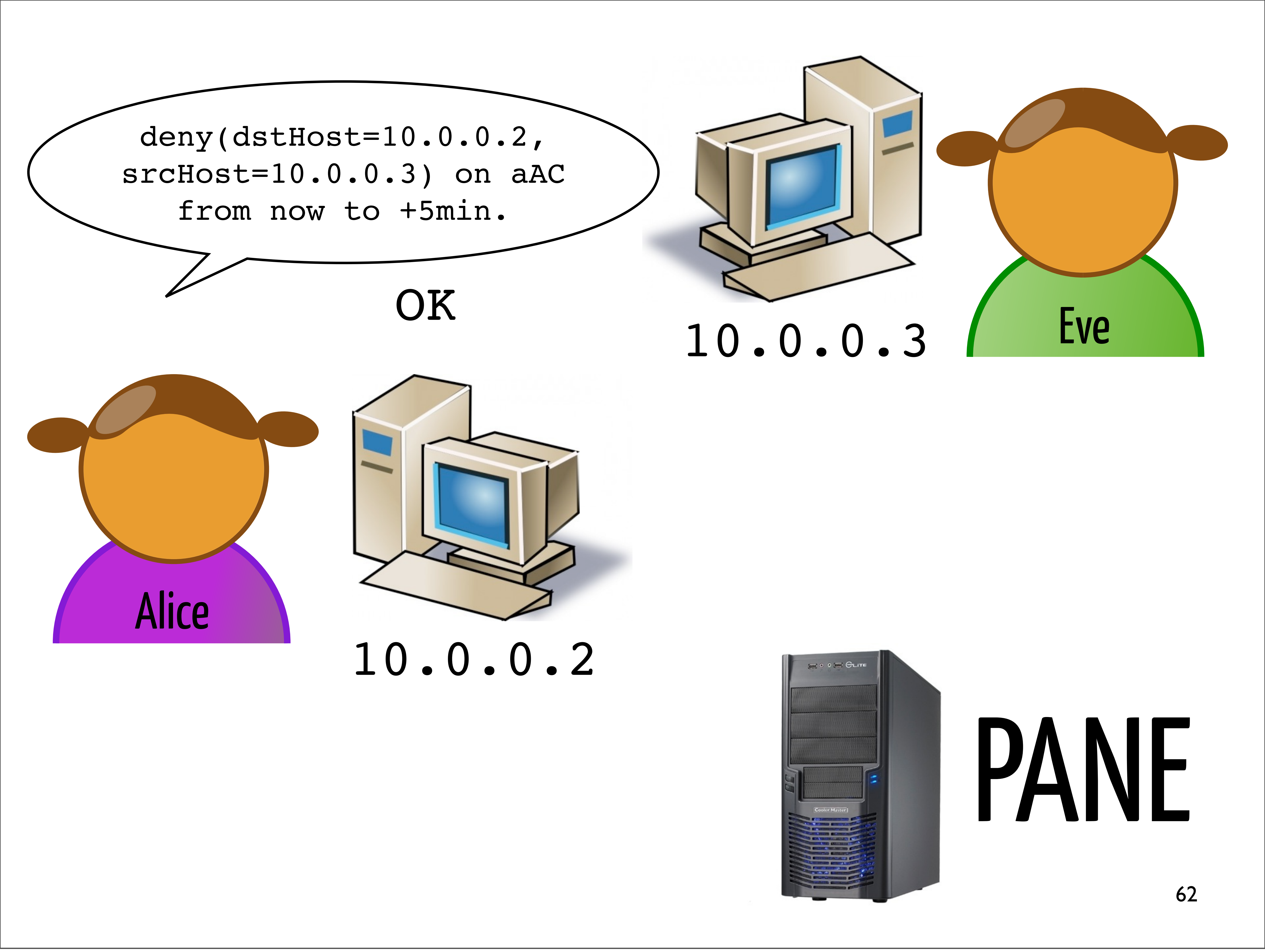
Because Alice was previously granted this authority, the PANE controller accepts her request (click), and uses OpenFlow to reconfigure the switches and block traffic from Eve's computer destined to Alice's (click).

If Alice tried to block Eve's traffic to another computer by changing the dstHost parameter on her request, the request would be denied as the flow would no longer be contained within the FlowGroup of the aAC subshare. (possibly make this its own slide?)

(Pause)

This has been a short sample of the PANE protocol. Our prototype supports several additional commands, for example, to establish rate-limits, manage users, and query the state of the ShareTree.

(Pause)



… is being attacked by Eve (click), she can send a deny request (click) to the PANE controller to have Eve's traffic blocked for the next five minutes.

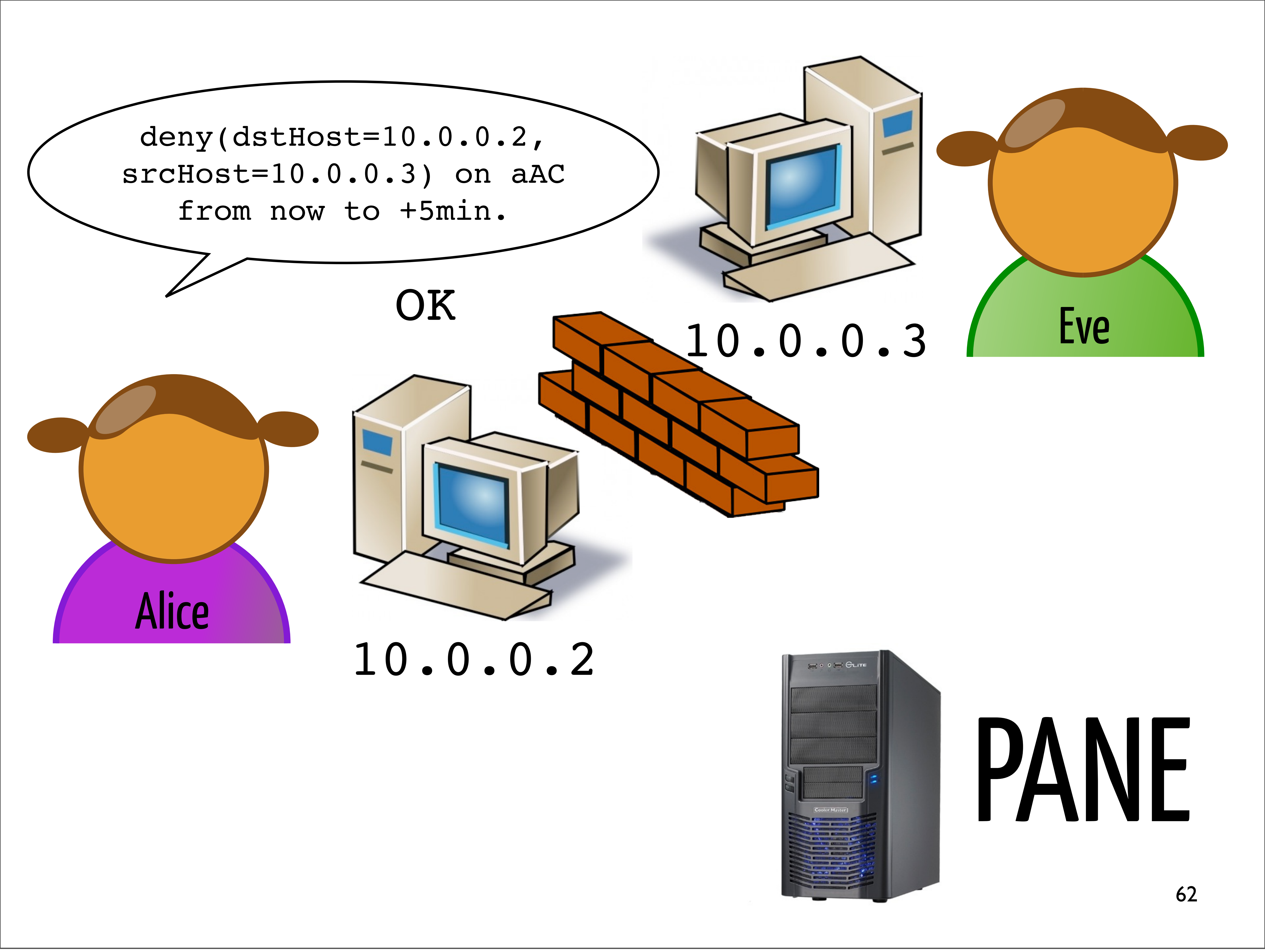
Because Alice was previously granted this authority, the PANE controller accepts her request (click), and uses OpenFlow to reconfigure the switches and block traffic from Eve's computer destined to Alice's (click).

If Alice tried to block Eve's traffic to another computer by changing the dstHost parameter on her request, the request would be denied as the flow would no longer be contained within the FlowGroup of the aAC subshare. (possibly make this its own slide?)

(Pause)

This has been a short sample of the PANE protocol. Our prototype supports several additional commands, for example, to establish rate-limits, manage users, and query the state of the ShareTree.

(Pause)



… is being attacked by Eve (click), she can send a deny request (click) to the PANE controller to have Eve's traffic blocked for the next five minutes.

Because Alice was previously granted this authority, the PANE controller accepts her request (click), and uses OpenFlow to reconfigure the switches and block traffic from Eve's computer destined to Alice's (click).

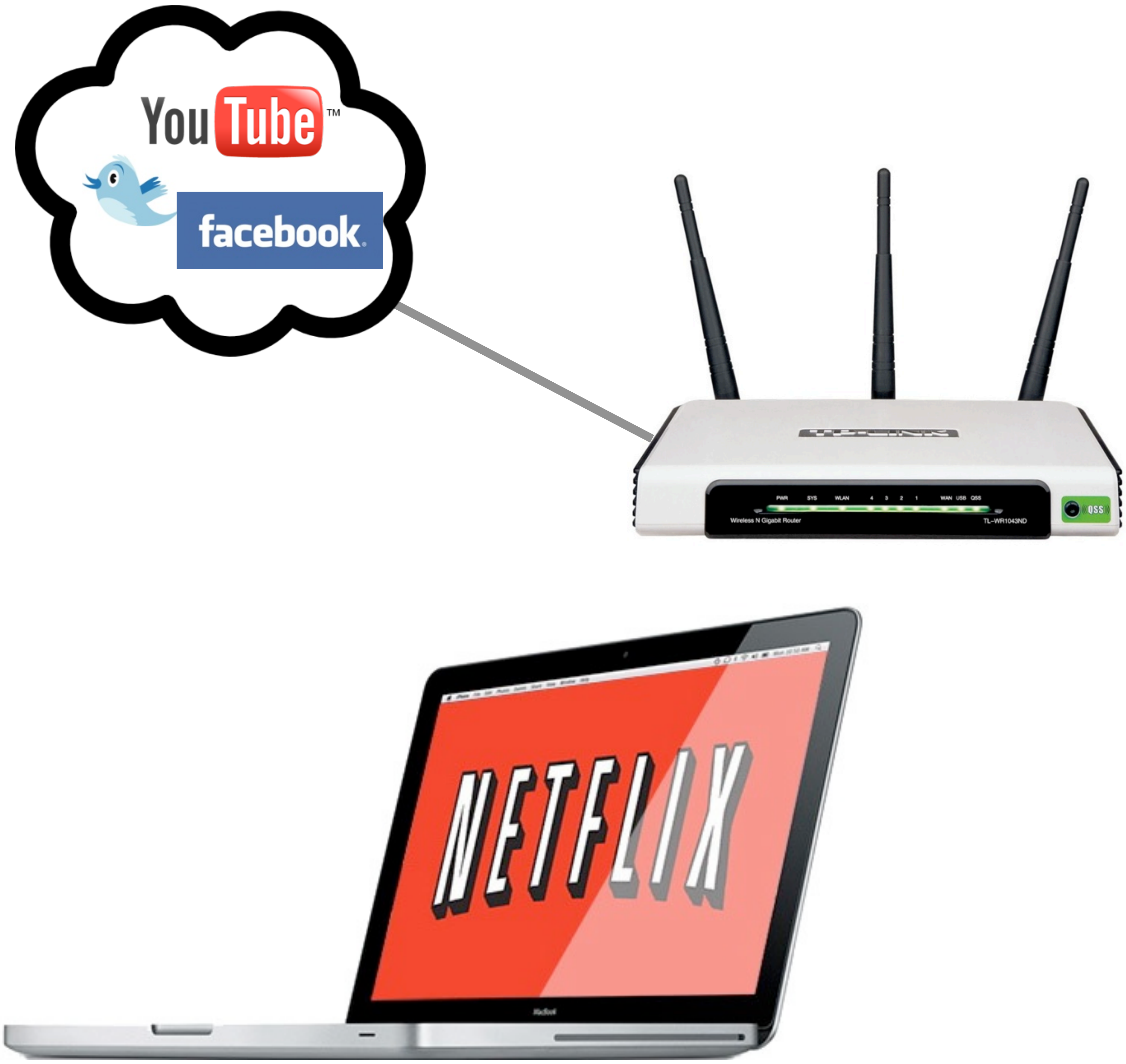
If Alice tried to block Eve's traffic to another computer by changing the dstHost parameter on her request, the request would be denied as the flow would no longer be contained within the FlowGroup of the aAC subshare. (possibly make this its own slide?)

(Pause)

This has been a short sample of the PANE protocol. Our prototype supports several additional commands, for example, to establish rate-limits, manage users, and query the state of the ShareTree.

(Pause)

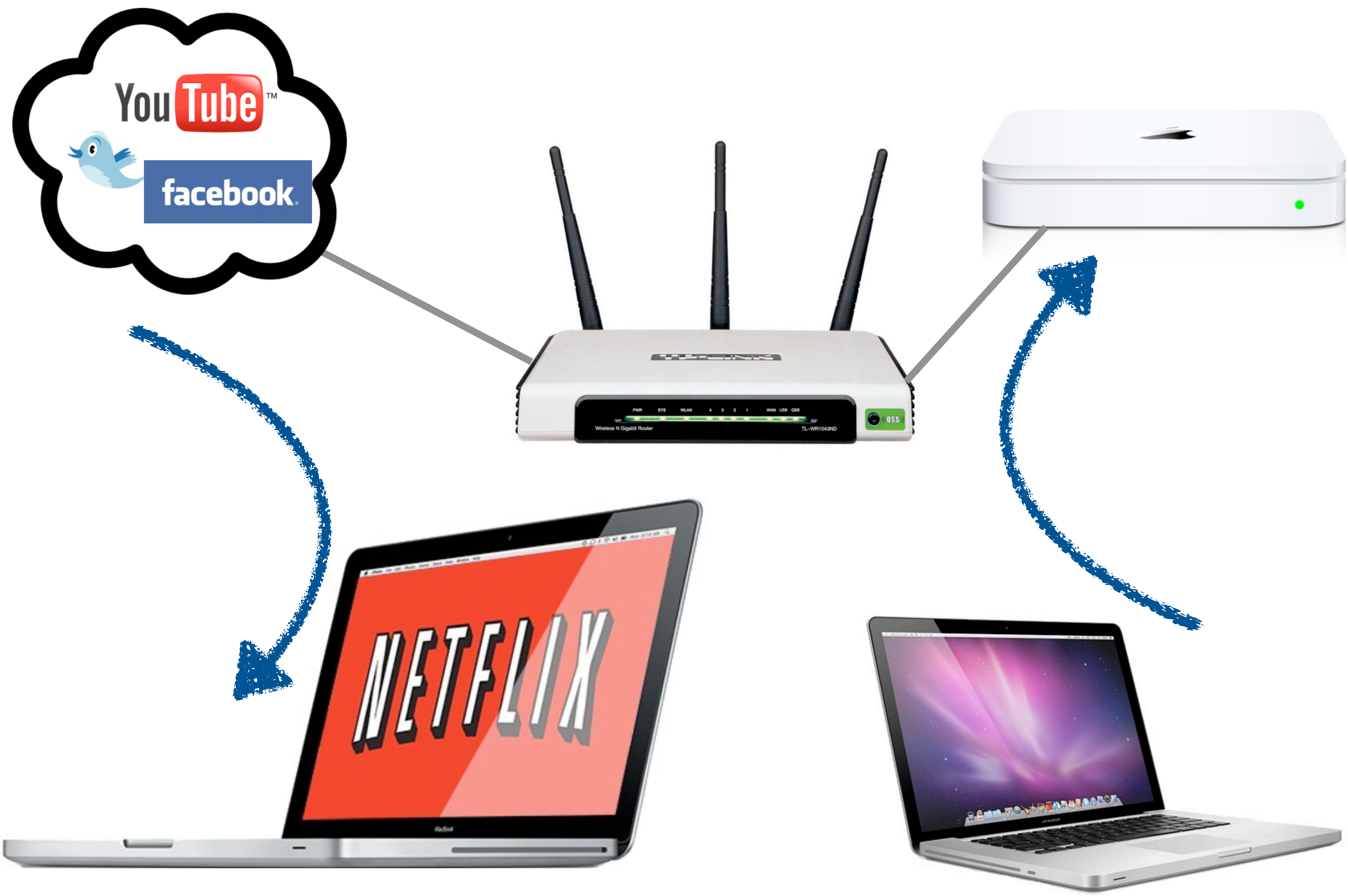
Netflix



For example, I like to watch movies at home with Netflix.

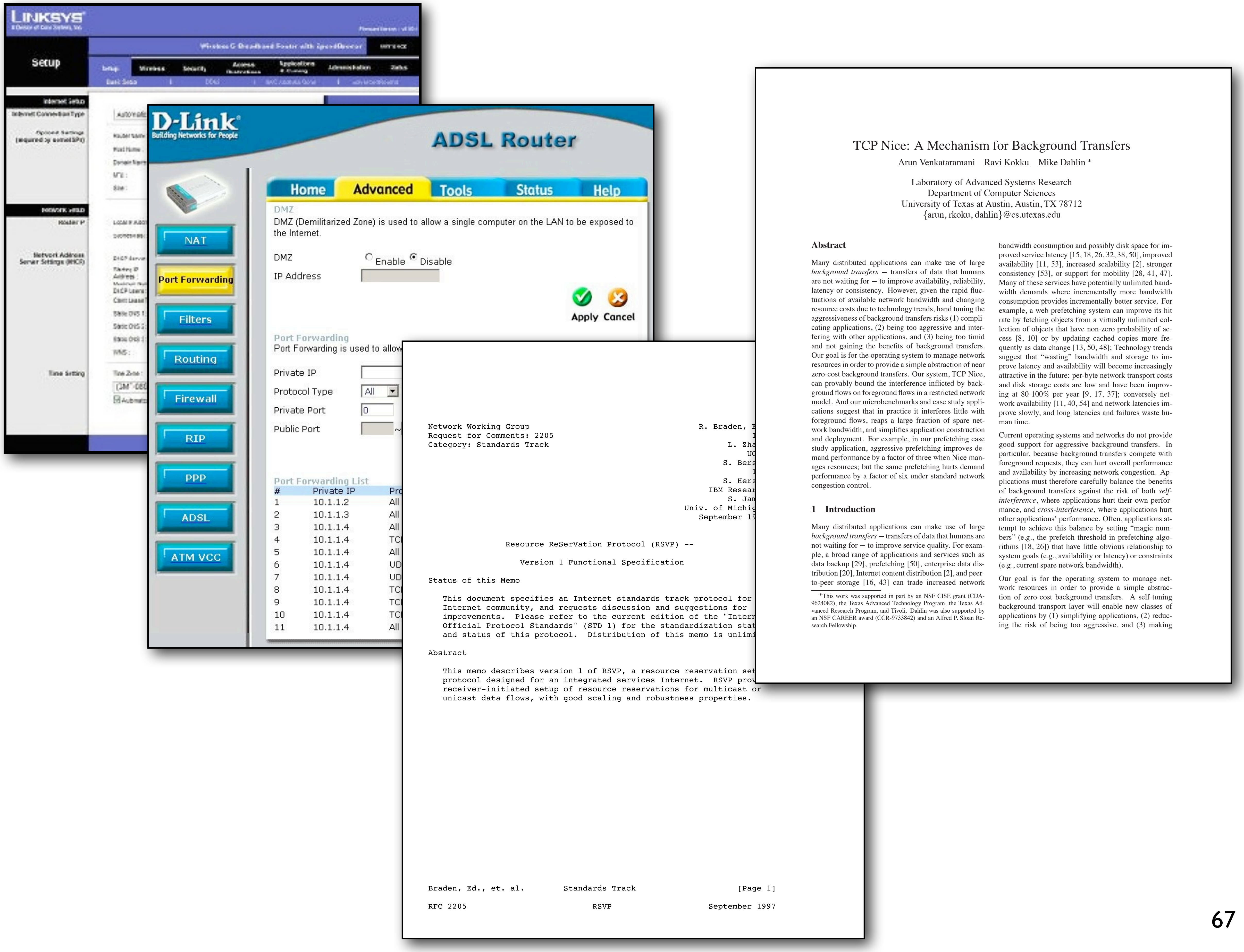
And while there are many reasons …

… why Netflix may begin to buffer, one reason is because ….



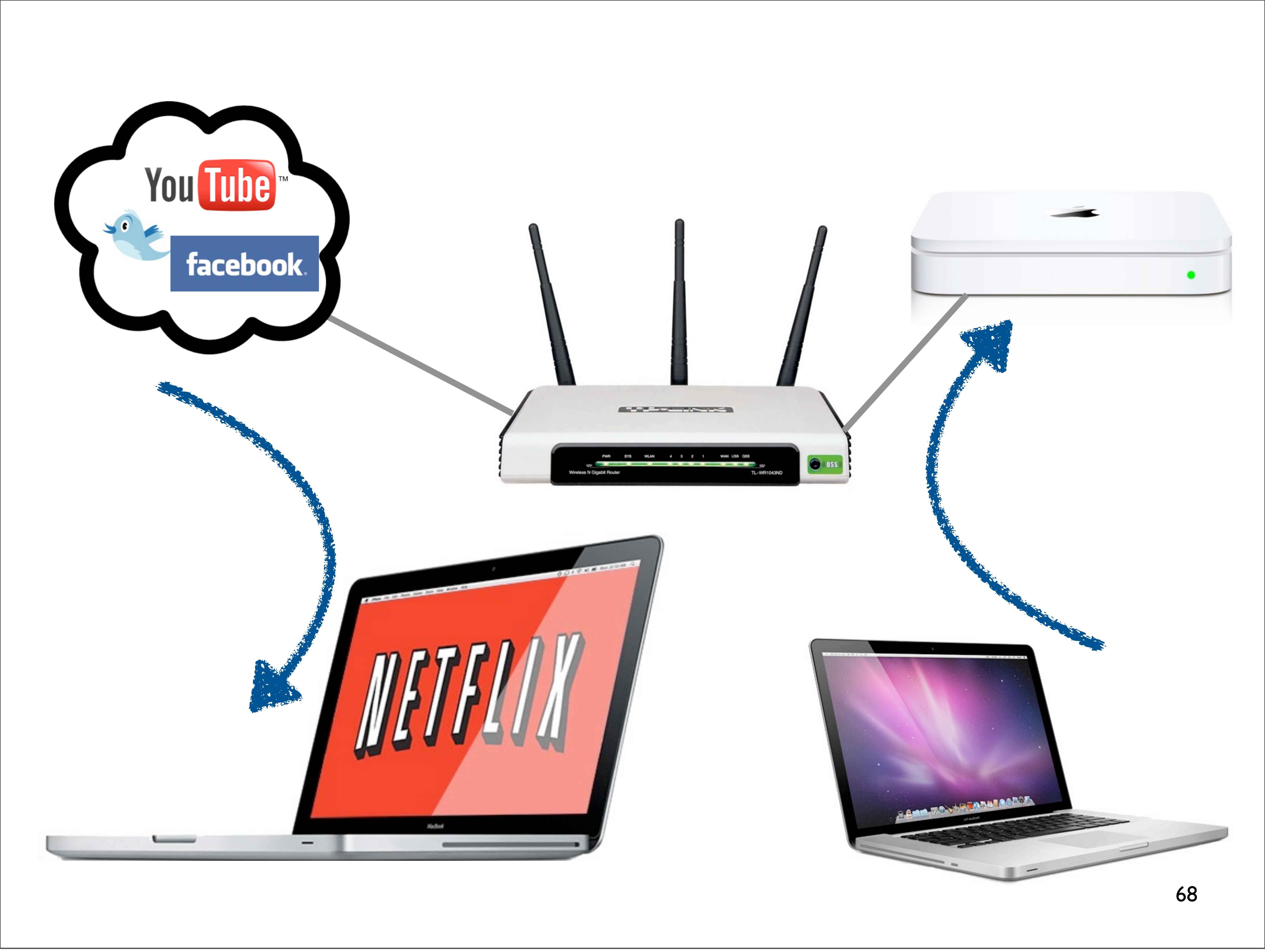
… a second laptop has begun a network backup.

And while there are …



… many proposals for how to solve this problem, it still exists.

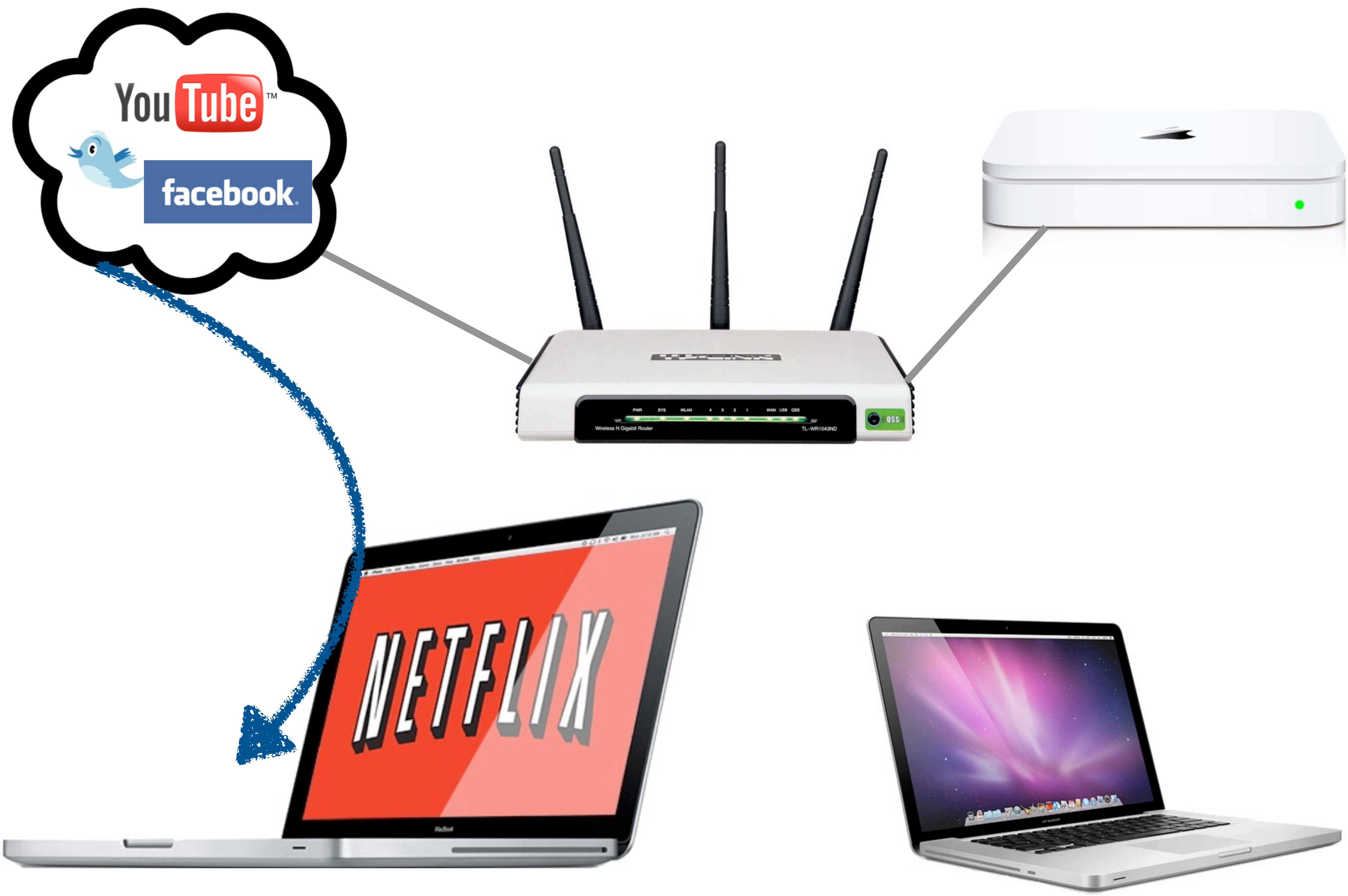
With participatory networking ...



the Netflix application can inform my home network of its bandwidth and latency requirements (click), and be guaranteed a level of service.

(pause)

Turning now to an enterprise network ...



the Netflix application can inform my home network of its bandwidth and latency requirements (click), and be guaranteed a level of service.

(pause)

Turning now to an enterprise network ...

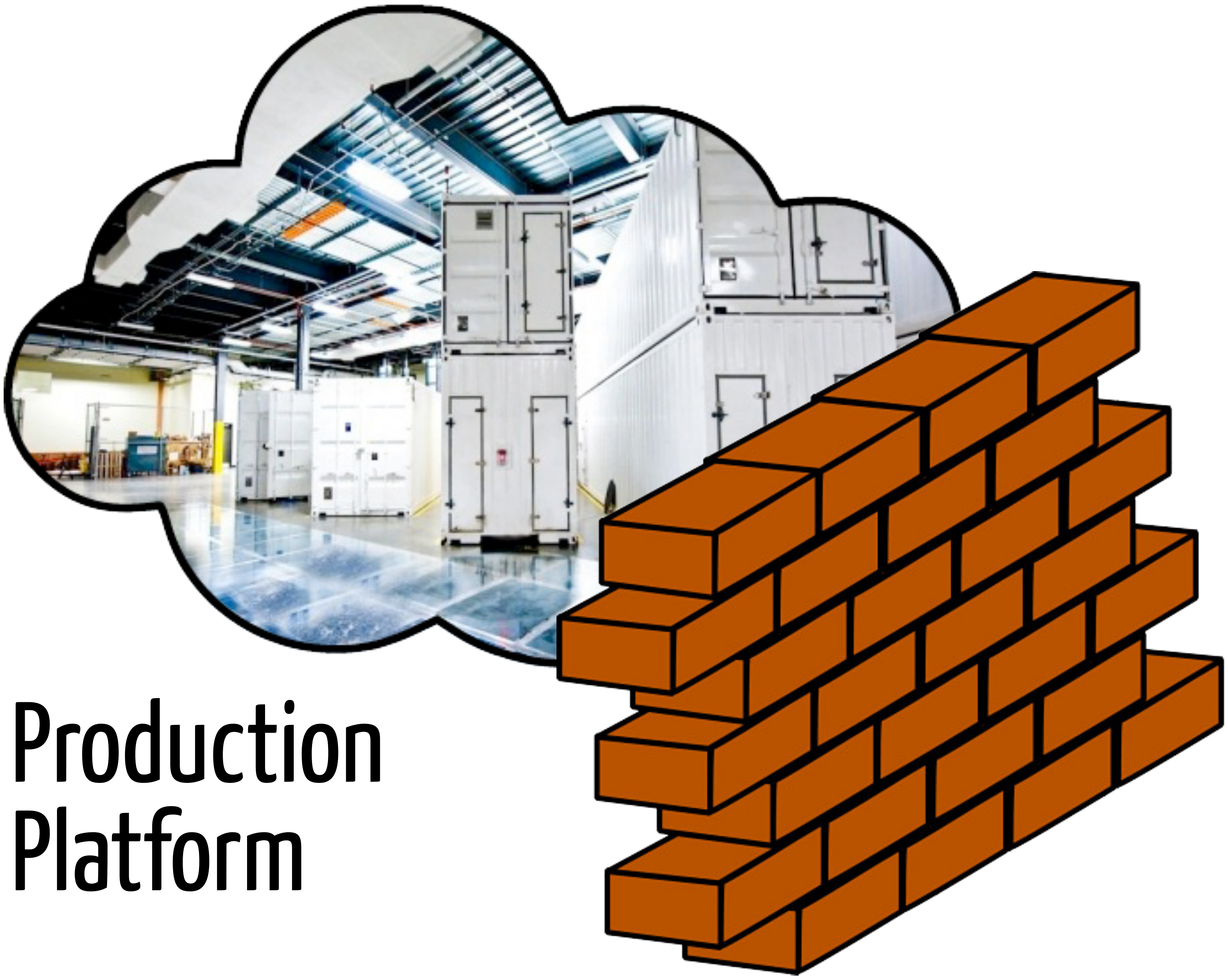
Datacenter

Production Platform

Based on "Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering" (EuroSys 2012)

on the Azure cloud environment (click) a firewall is used to isolate untrusted (click) customer virtual machines while booting.

After boot-up (click), the VM configuration can be made more secure ...

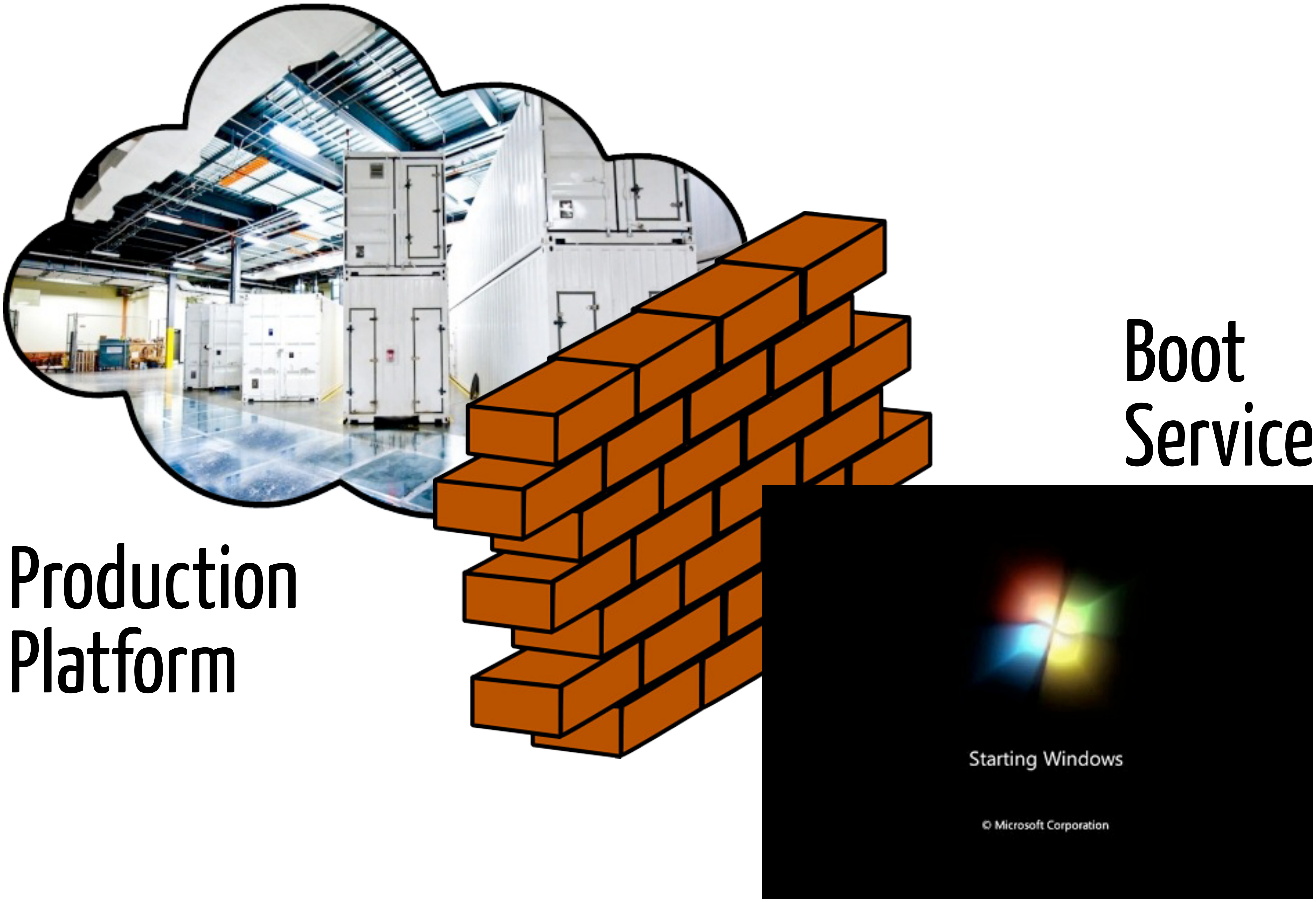


Production Platform

Based on “Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering” (EuroSys 2012)

on the Azure cloud environment (click) a firewall is used to isolate untrusted (click) customer virtual machines while booting.

After boot-up (click), the VM configuration can be made more secure ...



Based on “Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering” (EuroSys 2012)

on the Azure cloud environment (click) a firewall is used to isolate untrusted (click) customer virtual machines while booting.

After boot-up (click), the VM configuration can be made more secure ...

Based on “Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering” (EuroSys 2012)

on the Azure cloud environment (click) a firewall is used to isolate untrusted (click) customer virtual machines while booting.

After boot-up (click), the VM configuration can be made more secure ...

Boot
Service

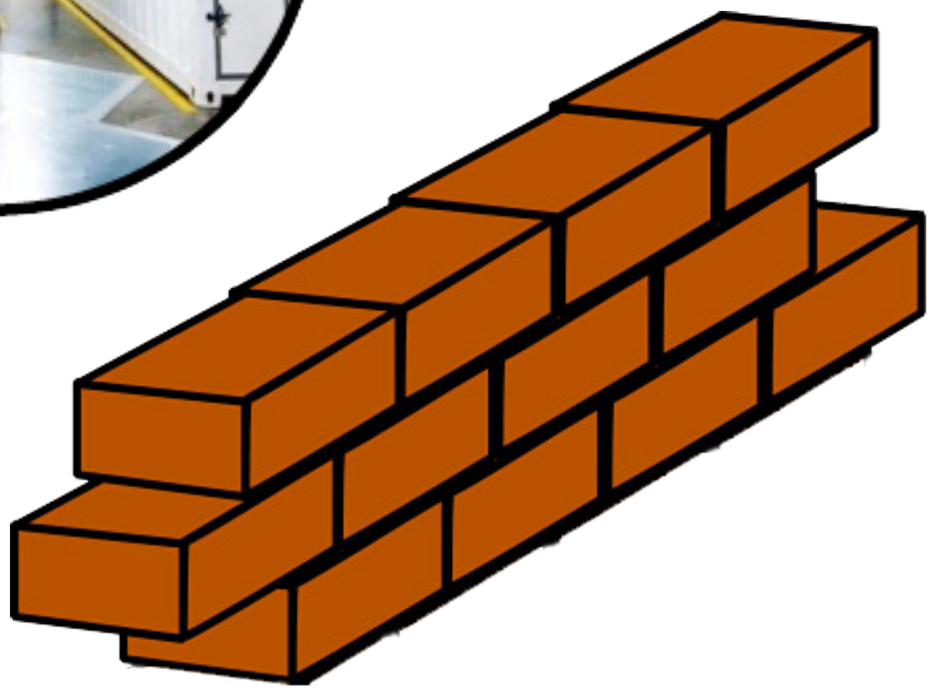
Production
Platform

Based on "Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering" (EuroSys 2012)

... the firewall lowered ...

Based on "Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering" (EuroSys 2012)

… and the VM image transferred to the production-side of the cloud.

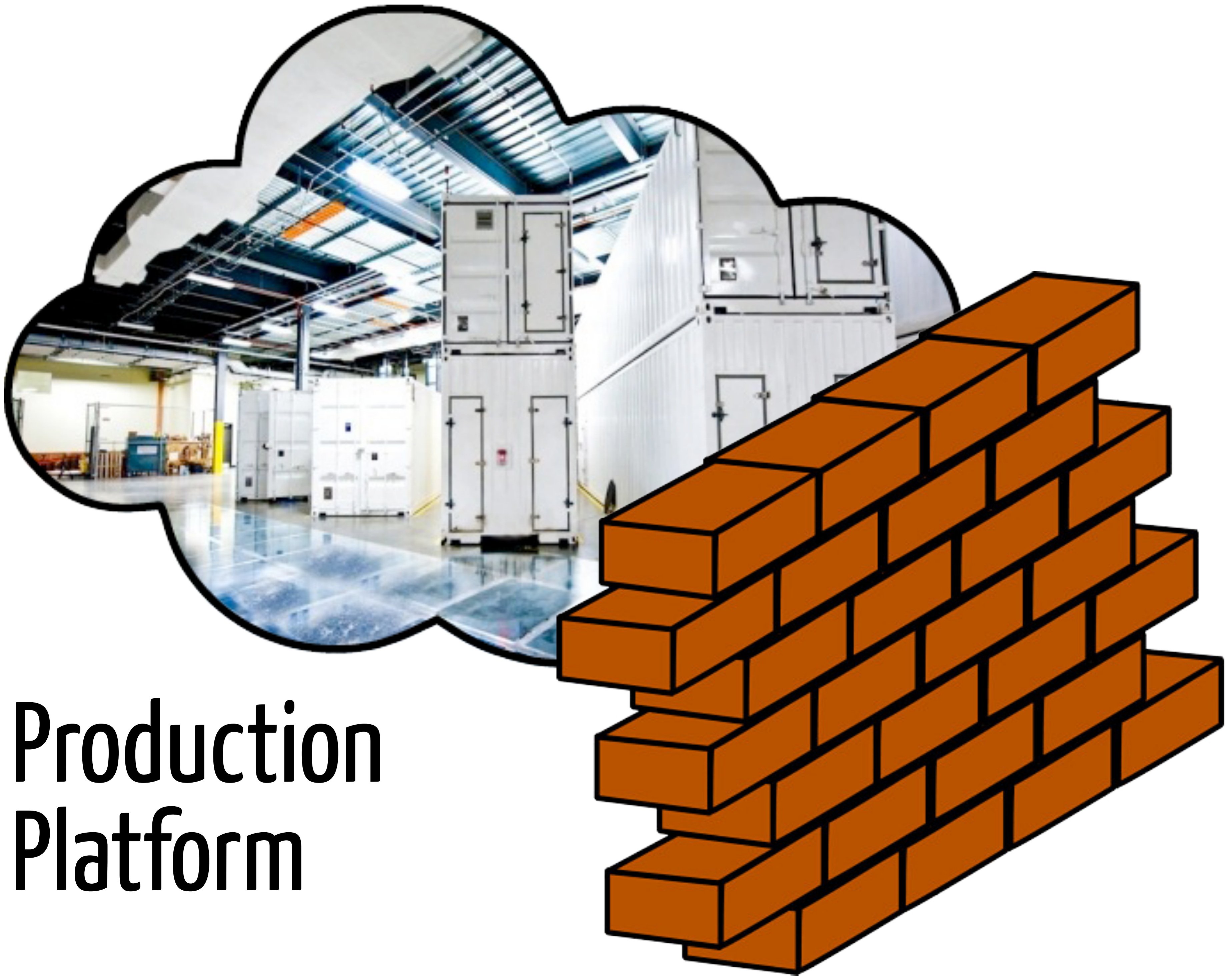


Boot
Service

Production
Platform

Based on "Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering" (EuroSys 2012)

… and the VM image transferred to the production-side of the cloud.

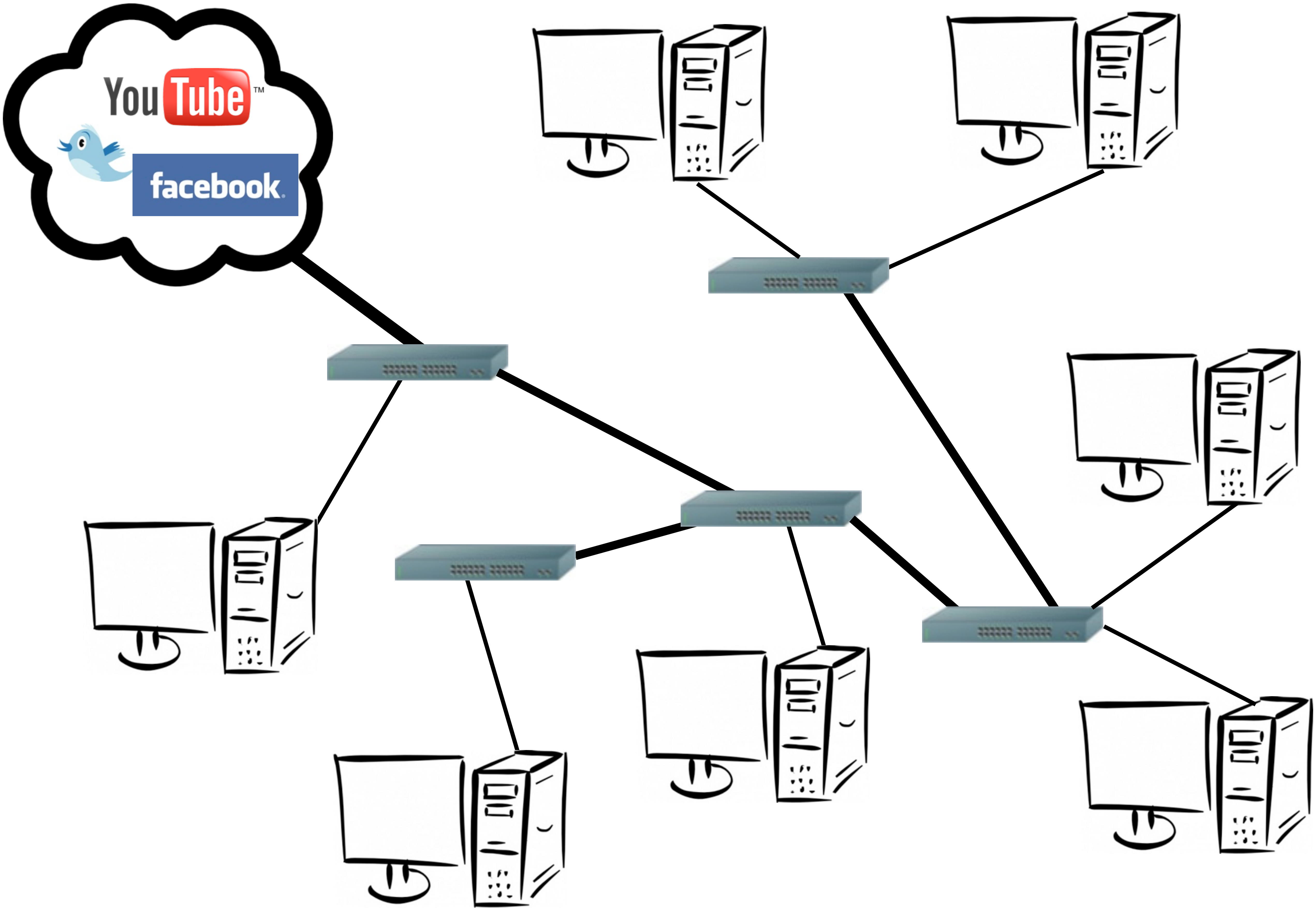


Based on "Delusional Boot: Securing Cloud Hypervisors without Massive Re-Engineering" (EuroSys 2012)

Lacking a practical API for managing the firewall via the virtual machine boot service, the implementation uses programmable MAC addresses on the servers, a static configuration on the firewall, and the usual duck tape we find in networks to achieve the result.

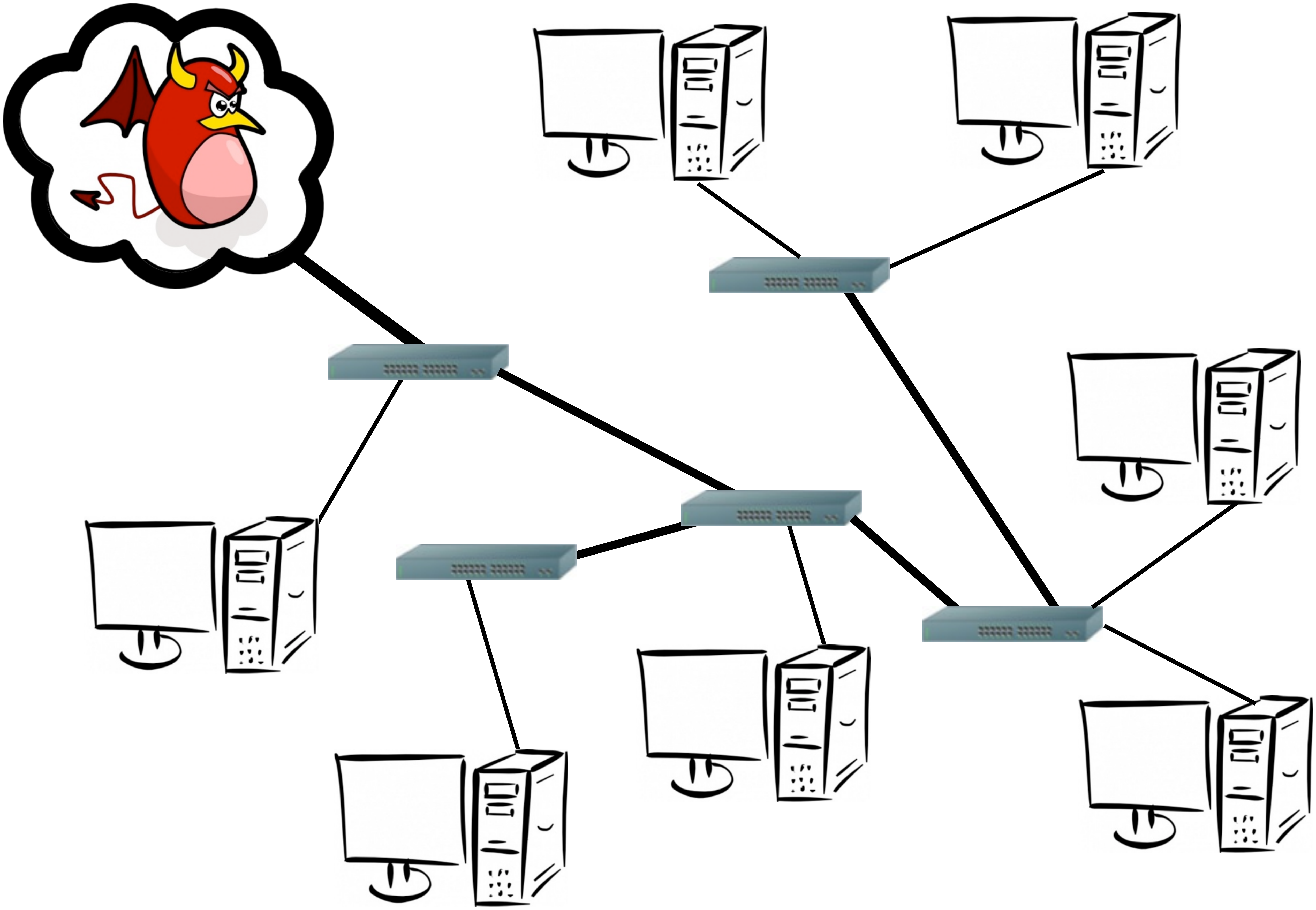
So again we can ask, why is this knowledge about managing the network trapped inside the end-hosts?

Enterprise



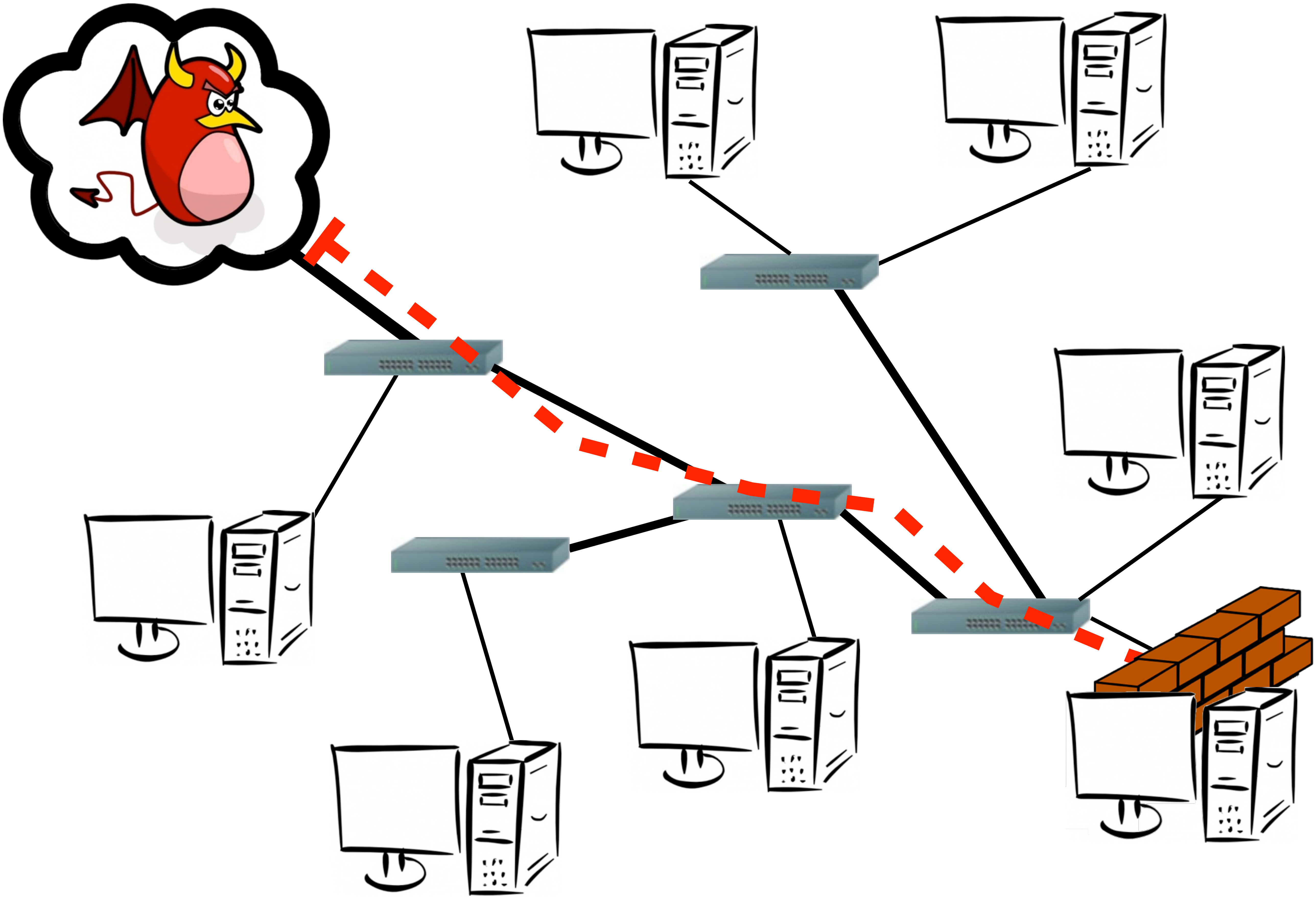
... we see shared links supporting many hosts.

And if one host suffers from a denial of service attack...



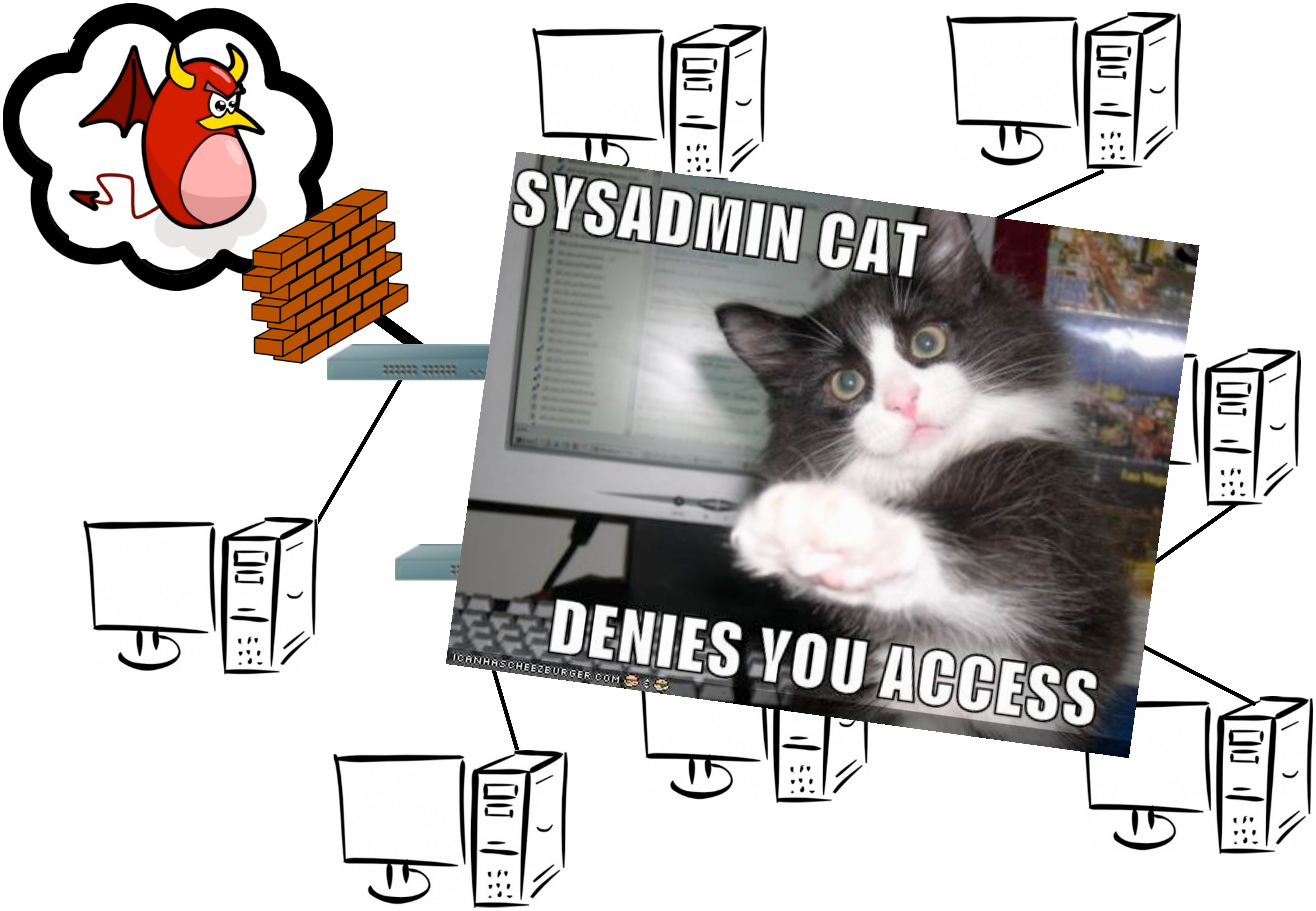
… we may need more than a local firewall rule to protect the network.

Today, we can call …

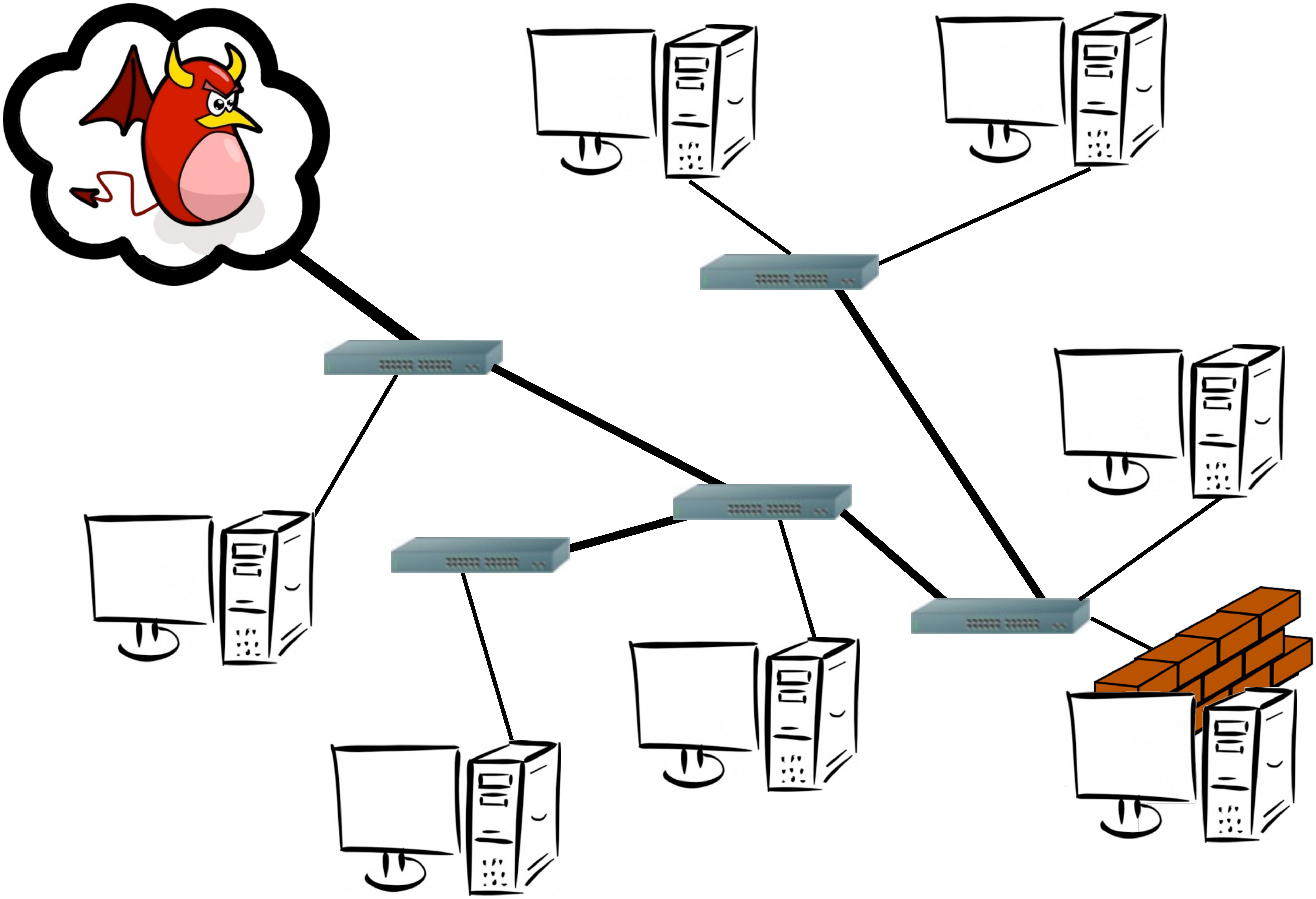


… we may need more than a local firewall rule to protect the network.

Today, we can call …



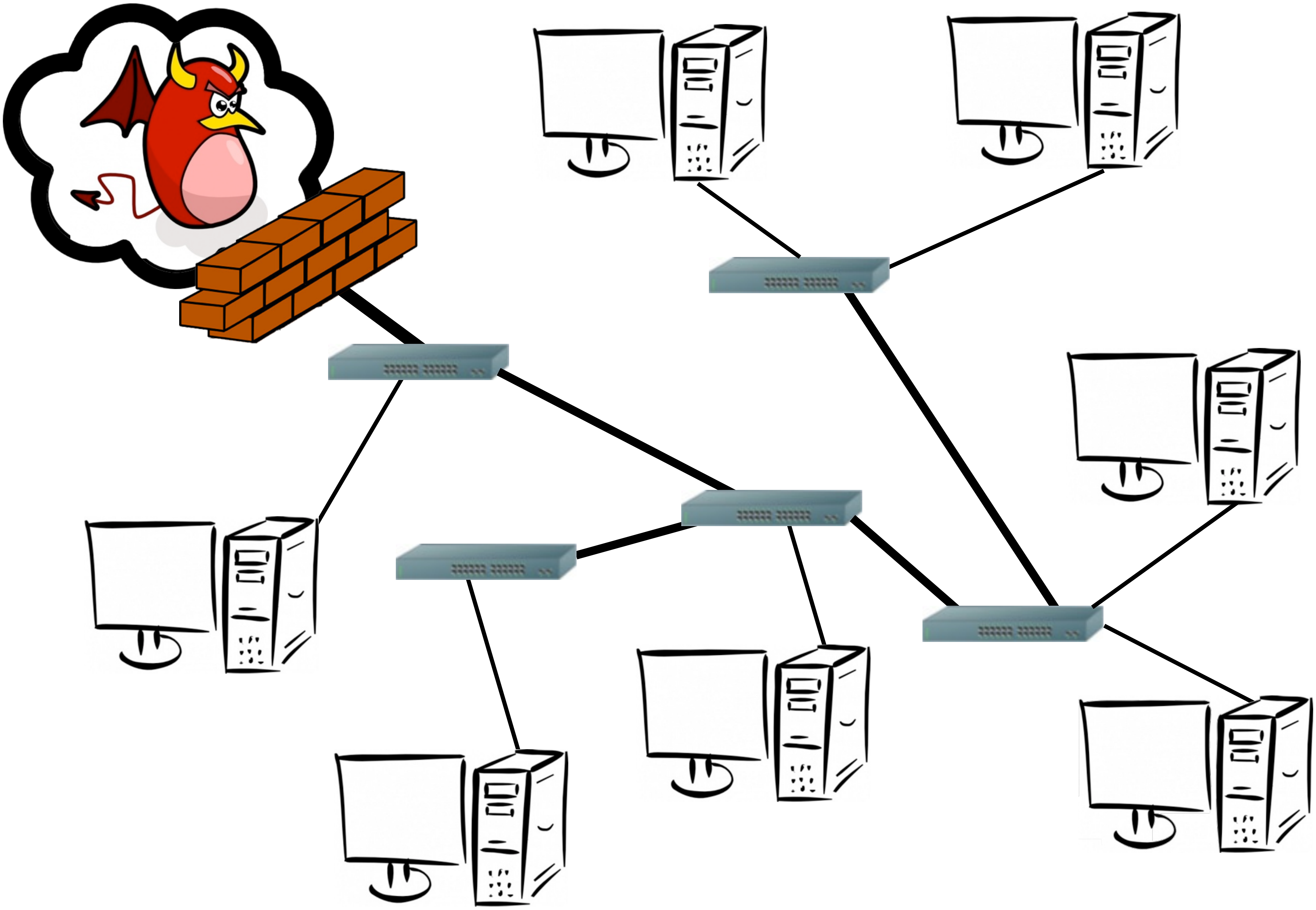
… the network administrator, or with participatory networking, the victim host ...



… can install a network firewall rule on its own.

(pause)

Furthermore, in Microsoft datacenters ...



78

… can install a network firewall rule on its own.

(pause)

Furthermore, in Microsoft datacenters ...

A problem in the datacenter

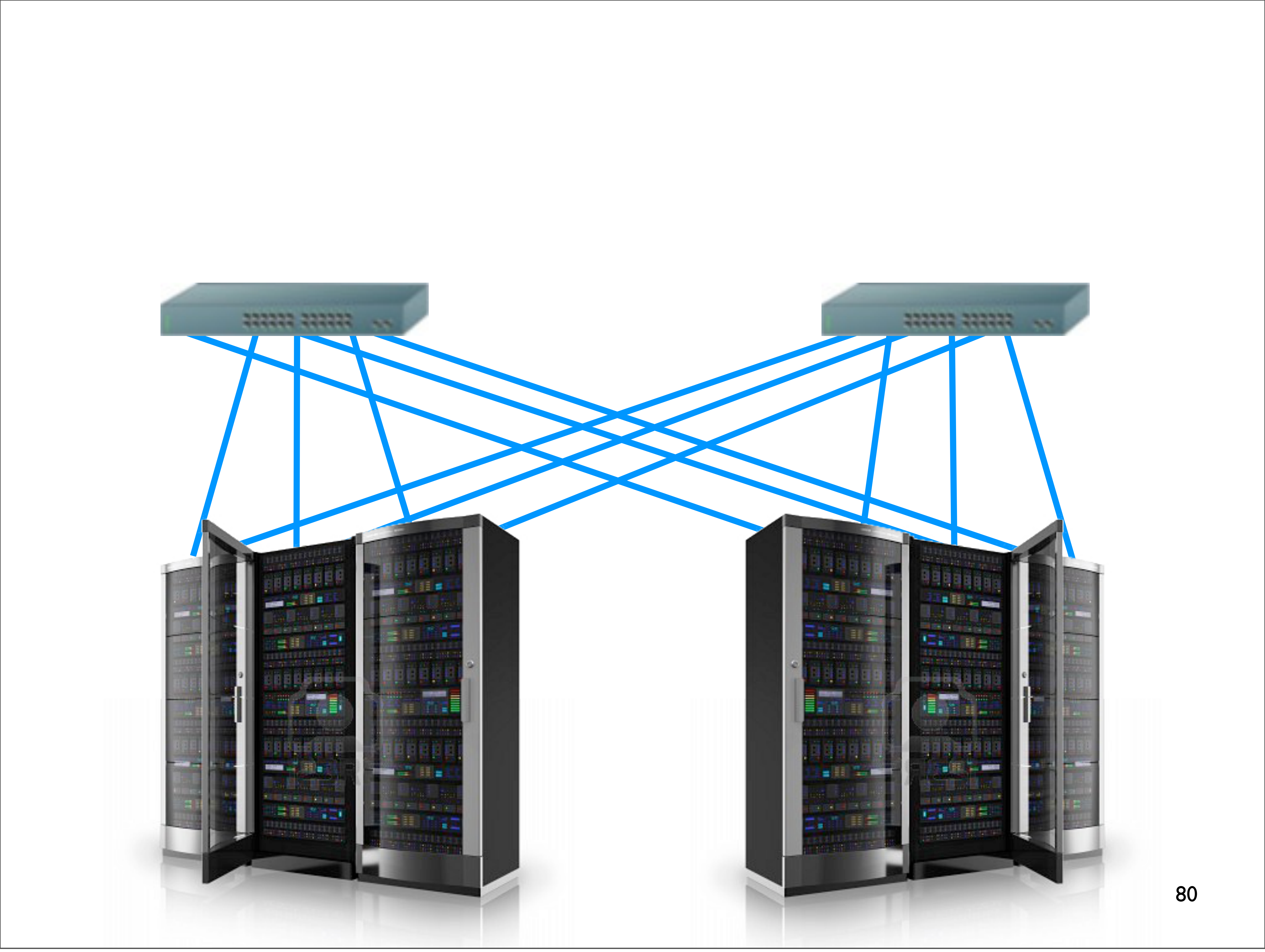
The final problem I want to look at exists in current proposals for hybrid optical-electrical networks.

80

In these hybrid networks, connectivity is primarily provided by Ethernet running over the usual copper cables (click). In addition, the top-of-rack switches are also connected by a fully optical network (click).

The optical switch can create circuits between rack pairs (click), but cannot be reconfigured quickly because of physical delays when aligning the internal mirrors.

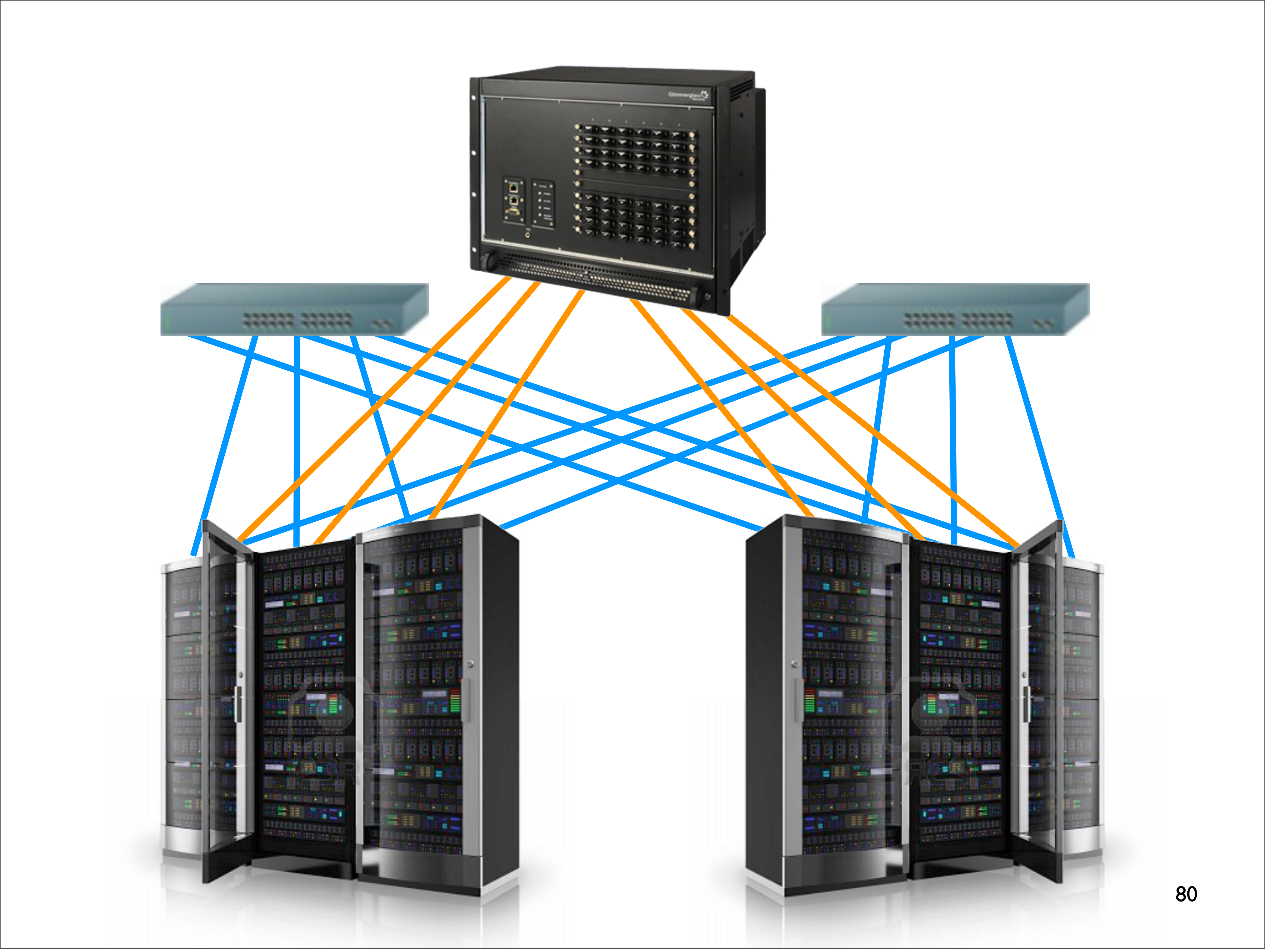
In the current proposals...



In these hybrid networks, connectivity is primarily provided by Ethernet running over the usual copper cables (click). In addition, the top-of-rack switches are also connected by a fully optical network (click).

The optical switch can create circuits between rack pairs (click), but cannot be reconfigured quickly because of physical delays when aligning the internal mirrors.

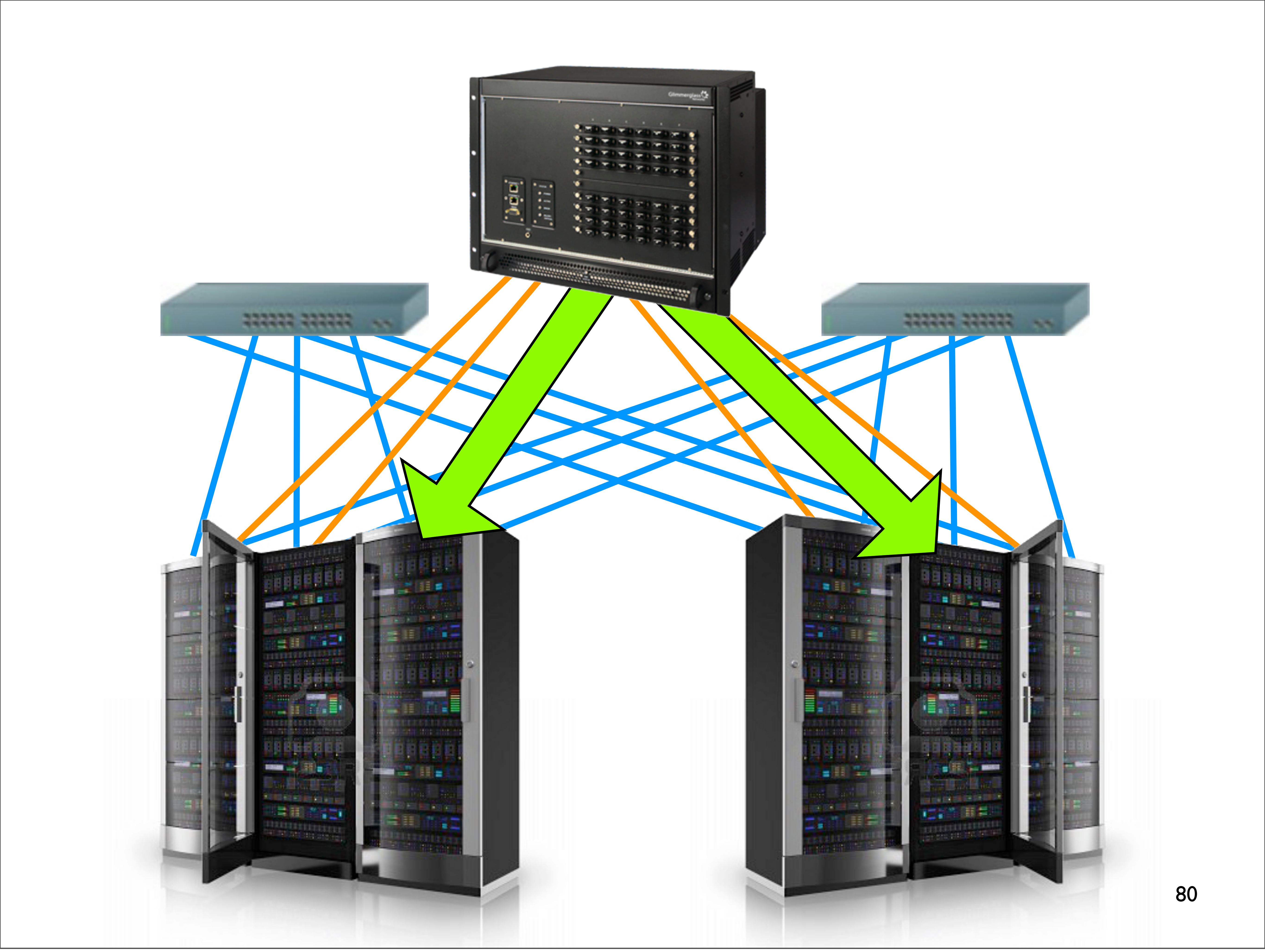
In the current proposals...



In these hybrid networks, connectivity is primarily provided by Ethernet running over the usual copper cables (click). In addition, the top-of-rack switches are also connected by a fully optical network (click).

The optical switch can create circuits between rack pairs (click), but cannot be reconfigured quickly because of physical delays when aligning the internal mirrors.

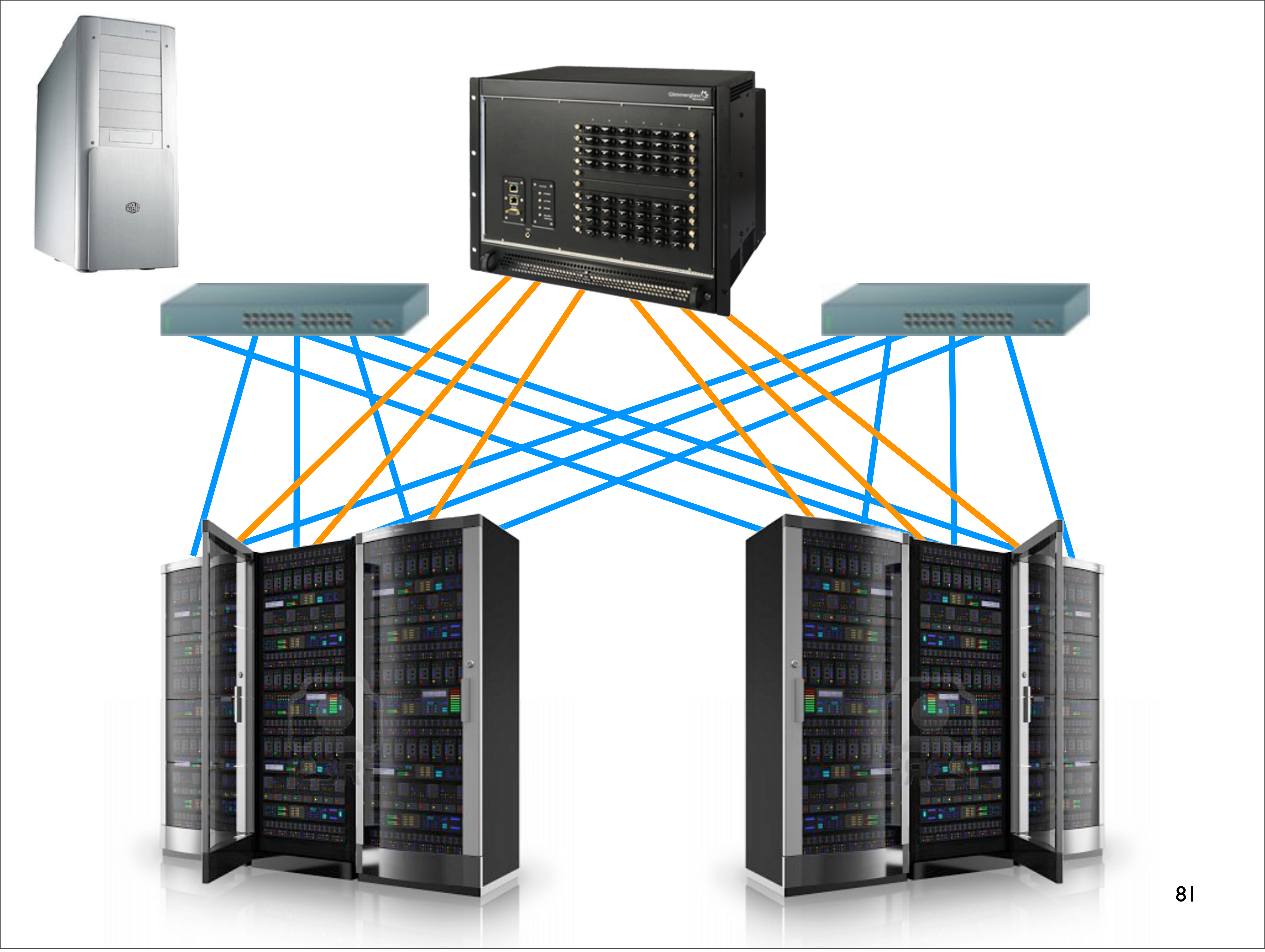
In the current proposals...



In these hybrid networks, connectivity is primarily provided by Ethernet running over the usual copper cables (click). In addition, the top-of-rack switches are also connected by a fully optical network (click).

The optical switch can create circuits between rack pairs (click), but cannot be reconfigured quickly because of physical delays when aligning the internal mirrors.

In the current proposals...



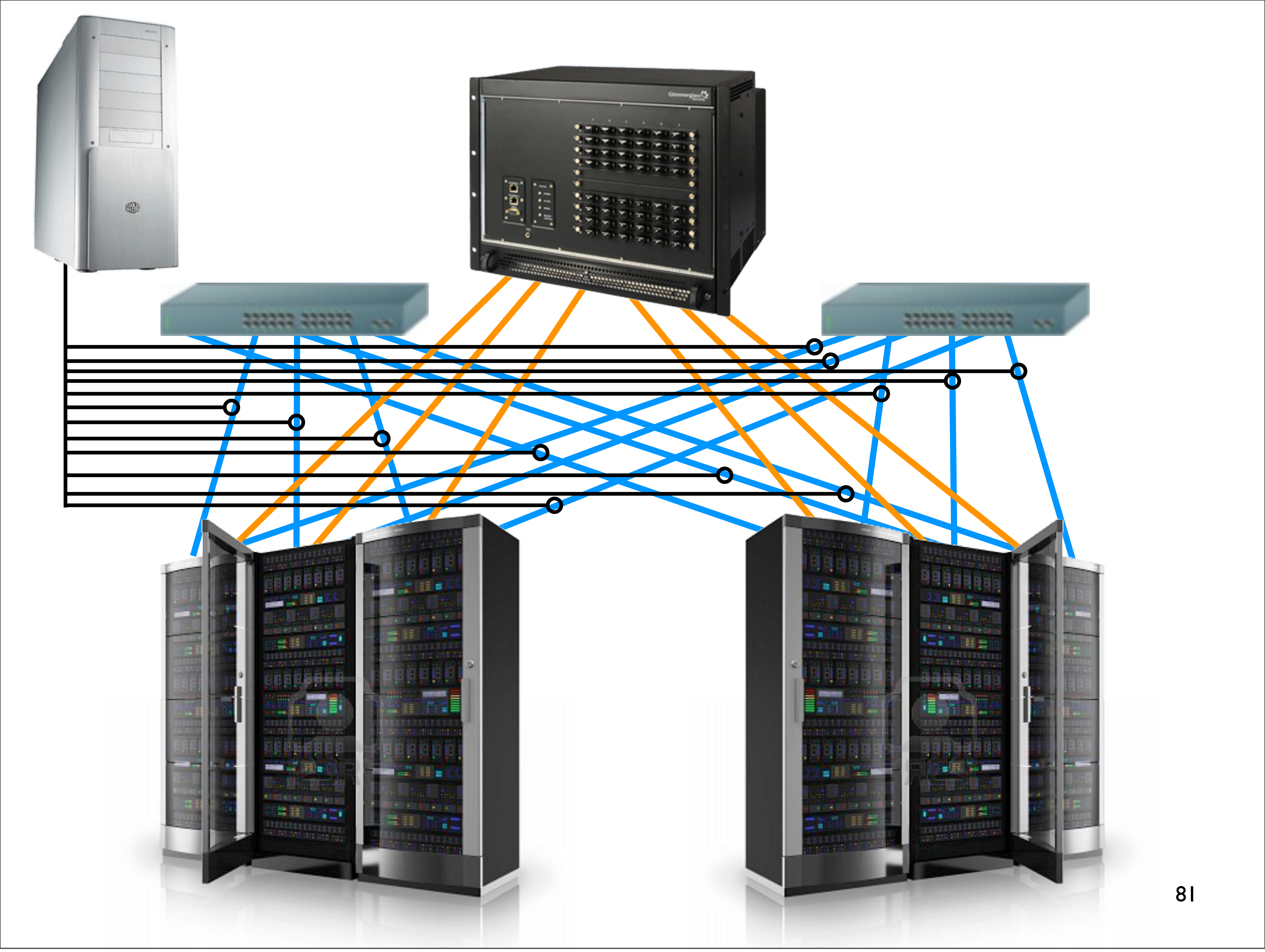
a management server monitors the traffic matrix (click) on the copper Ethernet and uses a heuristic to detect large, long-lasting flows that would benefit from the higher bandwidth and lower latency of an all-optical path.

When such flows are detected, the optical switch is reconfigured (click), and the heavy traffic eventually moved to the new path.

But such a detect-and-react strategy is not always necessary! There are many applications inside the datacenter that know in advance how much traffic they will generate. For example, virtual machine migrations and shuffle stages in MapReduce-like frameworks.

By now, I think you know the question to ask: why is this knowledge about managing the network trapped inside the end-hosts?

(5 minutes)



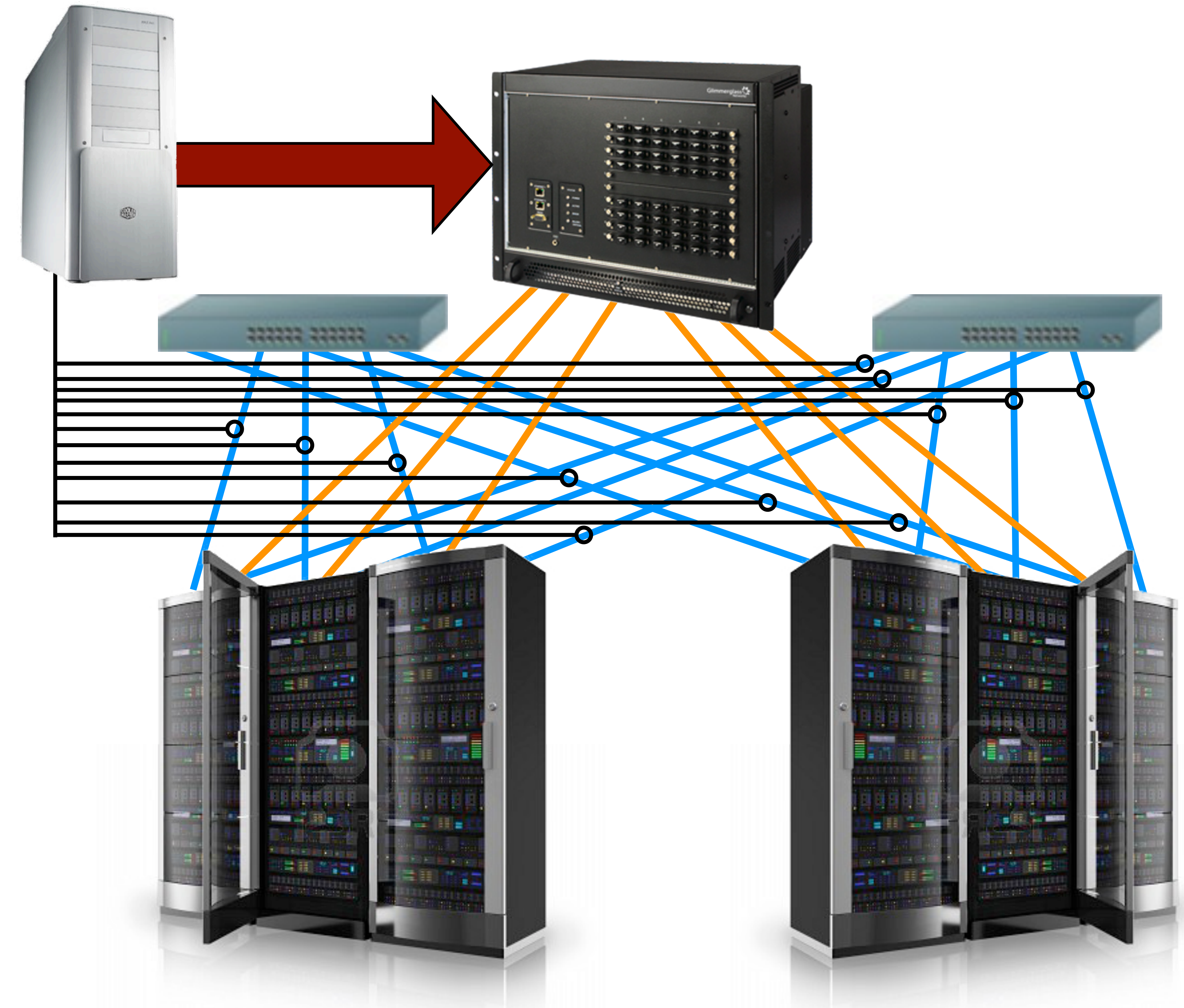
a management server monitors the traffic matrix (click) on the copper Ethernet and uses a heuristic to detect large, long-lasting flows that would benefit from the higher bandwidth and lower latency of an all-optical path.

When such flows are detected, the optical switch is reconfigured (click), and the heavy traffic eventually moved to the new path.

But such a detect-and-react strategy is not always necessary! There are many applications inside the datacenter that know in advance how much traffic they will generate. For example, virtual machine migrations and shuffle stages in MapReduce-like frameworks.

By now, I think you know the question to ask: why is this knowledge about managing the network trapped inside the end-hosts?

(5 minutes)



a management server monitors the traffic matrix (click) on the copper Ethernet and uses a heuristic to detect large, long-lasting flows that would benefit from the higher bandwidth and lower latency of an all-optical path.

When such flows are detected, the optical switch is reconfigured (click), and the heavy traffic eventually moved to the new path.

But such a detect-and-react strategy is not always necessary! There are many applications inside the datacenter that know in advance how much traffic they will generate. For example, virtual machine migrations and shuffle stages in MapReduce-like frameworks.

By now, I think you know the question to ask: why is this knowledge about managing the network trapped inside the end-hosts?

(5 minutes)

Participatory Networking

If we follow the analogy that software defined networks are developing an operating system for the network, Participatory Networking is building the end-user system calls -- an API for SDNs.

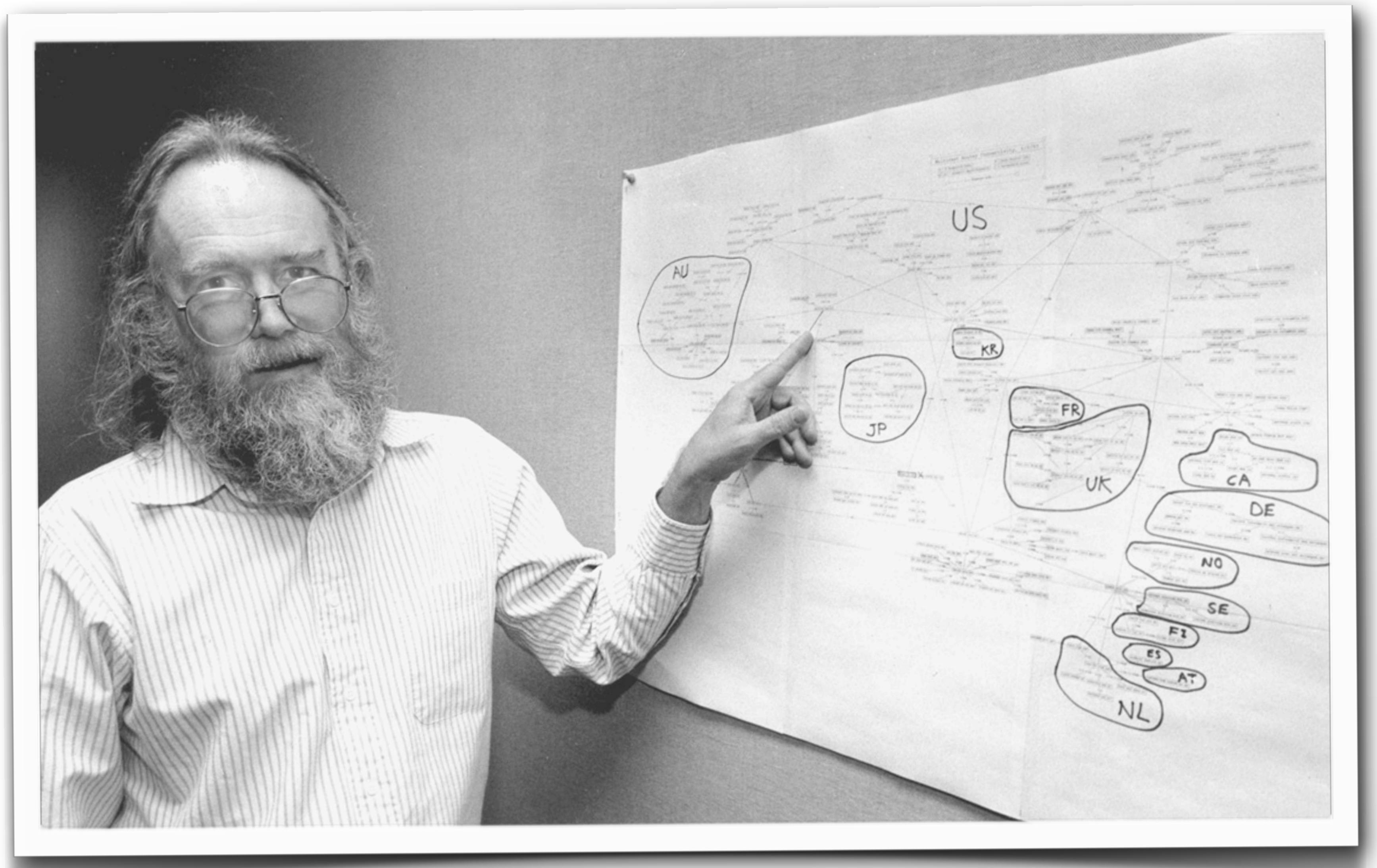
(pause)

Like previous work on operating systems ...

... SDNs began by providing abstractions over the hardware; we believe it's time for SDNs to similarly evolve into arbiters that support multiple principals sharing and controlling those resources.

(pause)

One challenge, of course, is the development and implementation of a semantics which delegates authority ...



Jon Postel

… from the network administrators ...

… to the people, without sacrificing high-level requirements such as ...

Participatory Networking

Safe?

Secure?

Fair?

Loop freedom?

Black holes?

safety, security, and fairness, and low-level properties such as freedom from routing loops and traffic black holes.