Systems Advanced II
Linux

Firewall:
netfilter & iptables

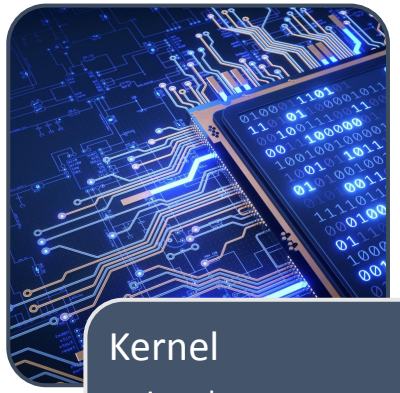
DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Doelstellingen

- De student:
  - De student kan Netwerk-services installeren, configureren en onderhouden.
  - De student kan microservices-infrastructuur opzetten en beheren.
  - De student kan een (eigen) cloud systeem opzetten a.d.h.v. opgelegde voorwaarden.
  - De student kan een systeem beveiligen.

Systems Advanced II - Linux



Kernel

- virtual memory
- user space and kernel space
- processes and threads
- sockets & pipes
- systemd
- Docker

Networking

- routing
- firewall
- eBPF
- certificates
- ssh
- latency
- bandwidth

Services

- nfs
- load balancers and reverse proxies
- databases
- logging

Clusters

- Storage
- Kubernetes

Licensing & open-source software

Systems Advanced II - Linux

kernel

- virtual memory
- user space and kernel space
- processes and threads
- sockets & pipes
- systemd
- Docker

Networking

- routing
- firewall
- eBPF
- certificates
- ssh
- latency
- bandwidth

Services

- nfs (rook)
- load balancers and reverse proxies
- databases
- logging

Clusters

- Storage
- Kubernetes

Systems Advanced II - Linux

kernel

- virtual memory
- user space and kernel space
- processes and threads
- sockets & pipes
- systemd
- Docker

Networking

- routing
- firewall
- eBPF
- certificates
- ssh
- latency
- bandwidth

Services

- nfs (rook)
- load balancers and reverse proxies
- databases
- logging

Clusters

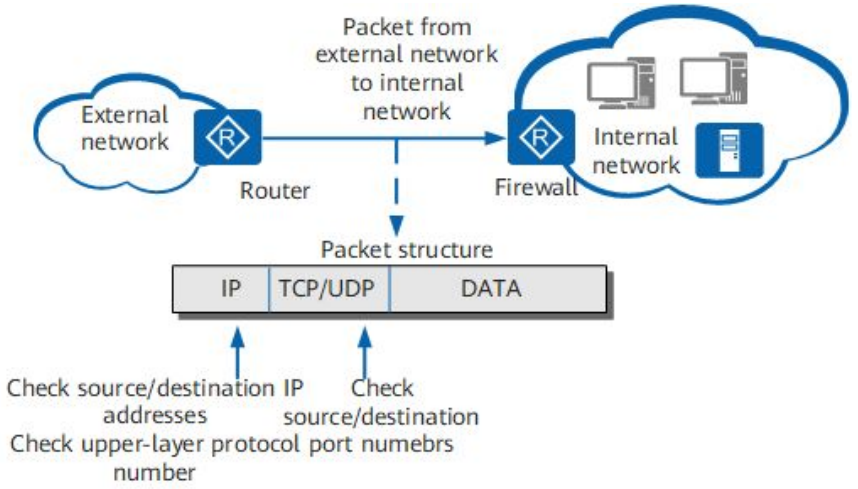
- Storage
- Kubernetes

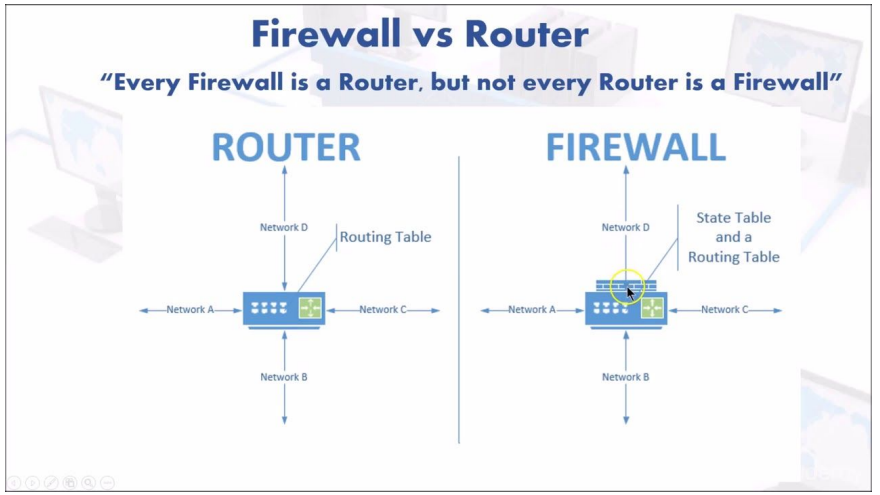
Agenda

- Firewall functions & features
- Linux firewall architecture
  - netfilter
  - iptables
- Tables & Chains
- Stateless vs. statefull
  - NAT & connection tracking
- LAB
- Rules & Targets

Functions of a Firewall

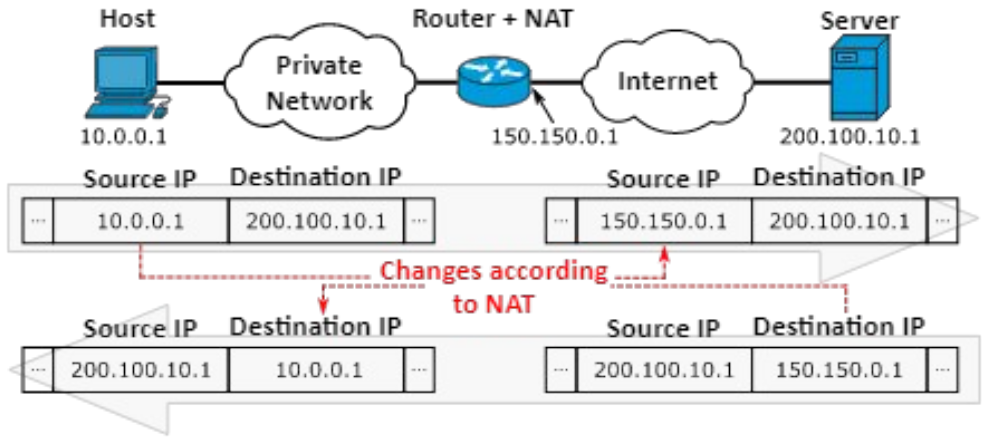
- Protects the host system from unauthorized connections.
  - Blocks connections to services that should not be public.
  - Enables restriction of connections to certain IP address ranges.
- Rewrites packet headers to route packets between networks.
  - Enables the machine to function as a network router.
  - Most consumer and business "router" devices are just small computers running a firewall with routing capabilities.





Advanced Firewall Functions

- Firewalls may implement additional features
  - Network Address Translation (NAT)
  - Quality of Service (QoS)



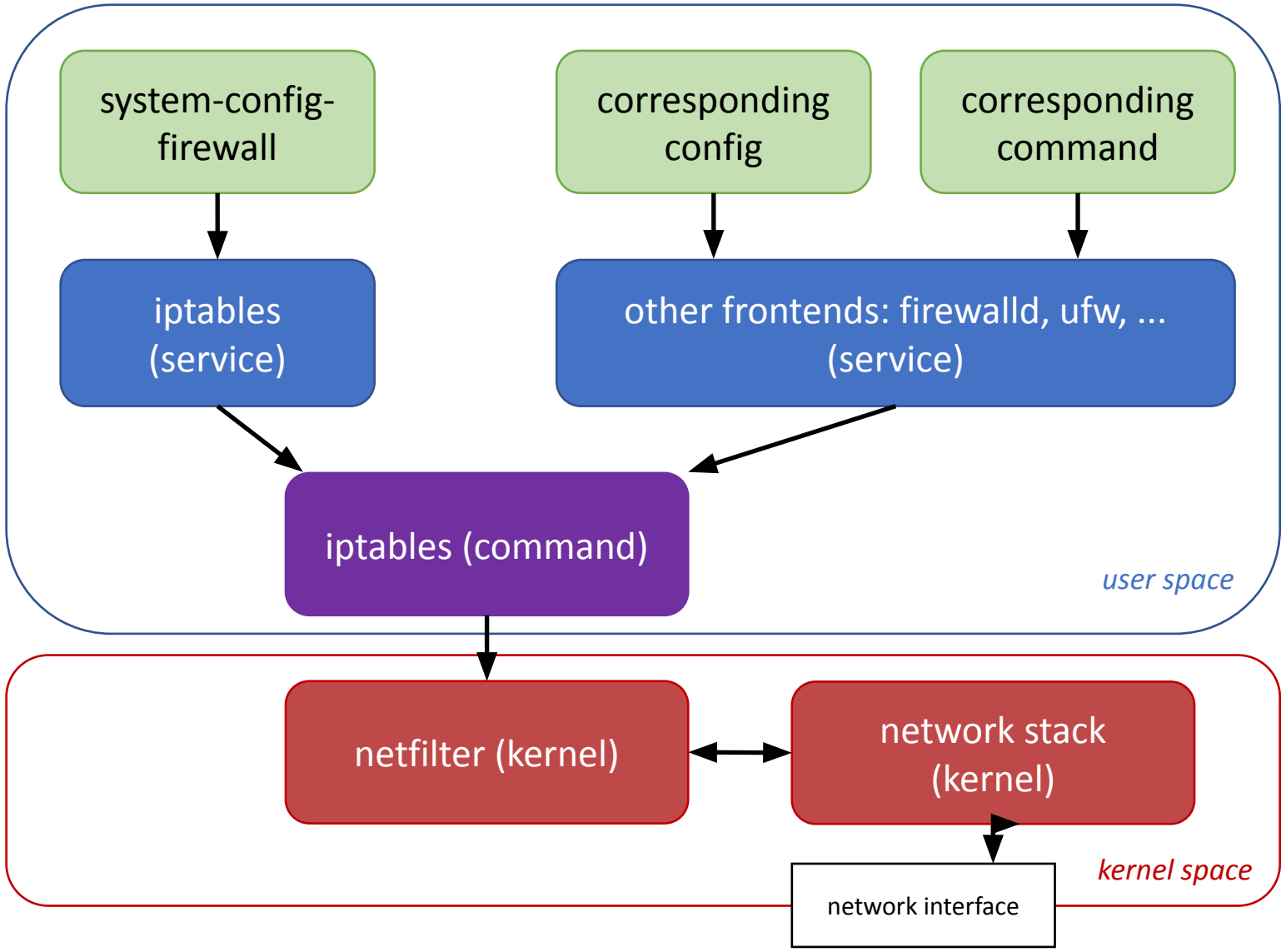
iptables and netfilter

- **iptables is a user space interface to the Linux kernel's *netfilter* system.**
- **netfilter** implements firewall and routing capabilities within the kernel, enabling any Linux machine or device to act as a firewall and/or a router.
- https://www.netfilter.org/
- Note: by convention, iptables is written in lower-case, since the command is lower-case.

iptables and netfilter features

- **iptables is a user space interface to the Linux kernel's netfilter system.**
  - listing the contents of the packet filter ruleset
  - adding/removing/modifying rules in the packet filter ruleset
  - listing/zeroing per-rule counters of the packet filter ruleset
- **netfilter** implements firewall and routing capabilities within the kernel, enabling any Linux machine or device to act as a firewall and/or a router.
  - stateless packet filtering (IPv4 and IPv6)
  - stateful packet filtering (IPv4 and IPv6)
  - all kinds of network address and port translation, e.g. NAT (IPv4 and IPv6)
  - flexible and extensible infrastructure
  - multiple layers of API's for 3rd party extensions

iptables Architecture

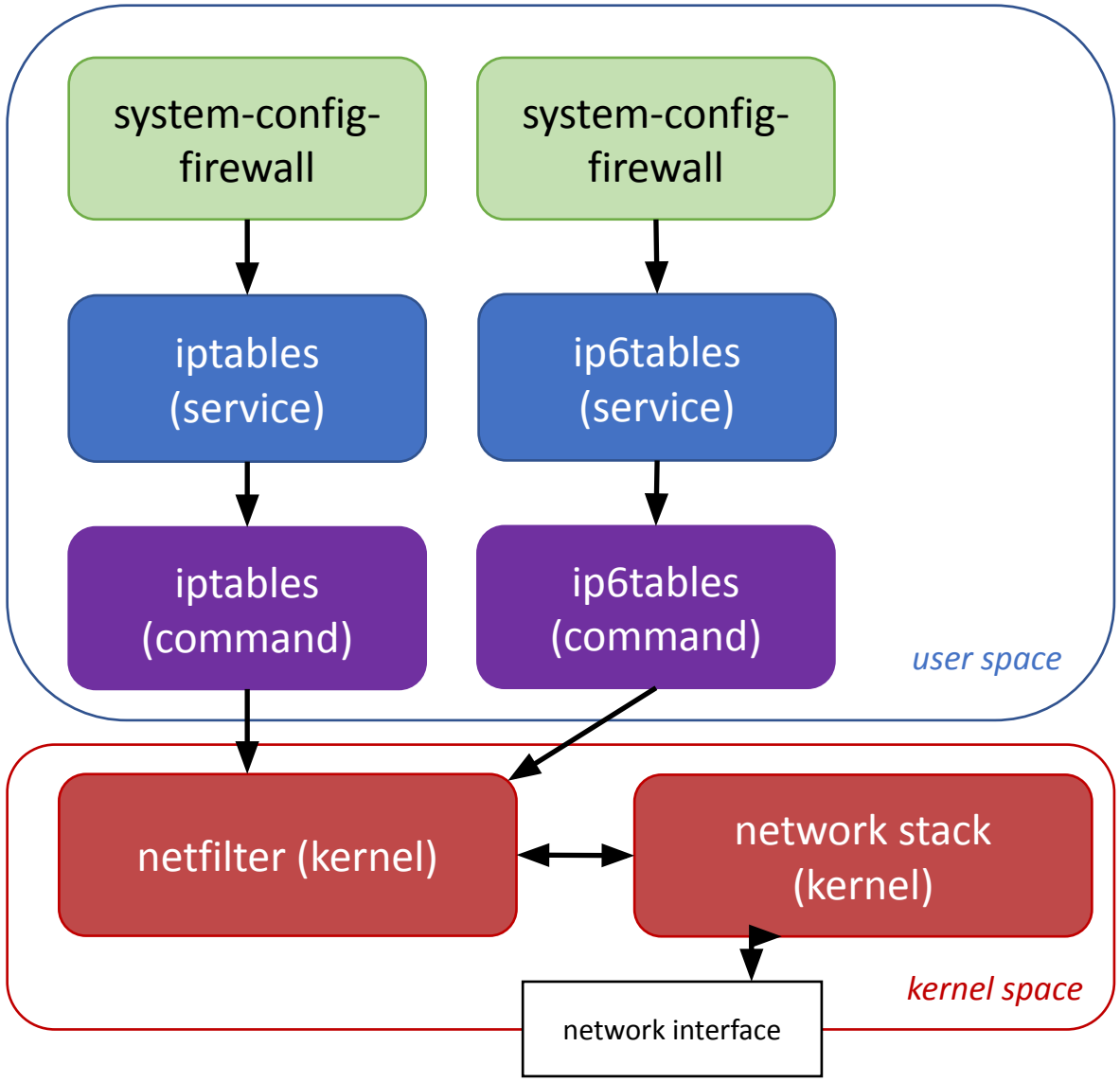


iptables is the user space interface to the linux kernel firewall.

iptables service is a systemd service that uses the iptables tool to interact with the kernel netfilter framework.

iptables is a Linux user space command line tool that manipulates the IPv4 network packet filtering tables and rules.

iptables Architecture: IPv6



- IPv6 has its own ip6tables command, service and config file
  - annoying to have separate IPv6 files
  - but
    - IPv4 still carries most Internet traffic
    - IPv6 still not used that much
      - status March 2023
        - 37.8% of Internet traffic
        - 20.8% of all websites
        - even less in company networks

nftables

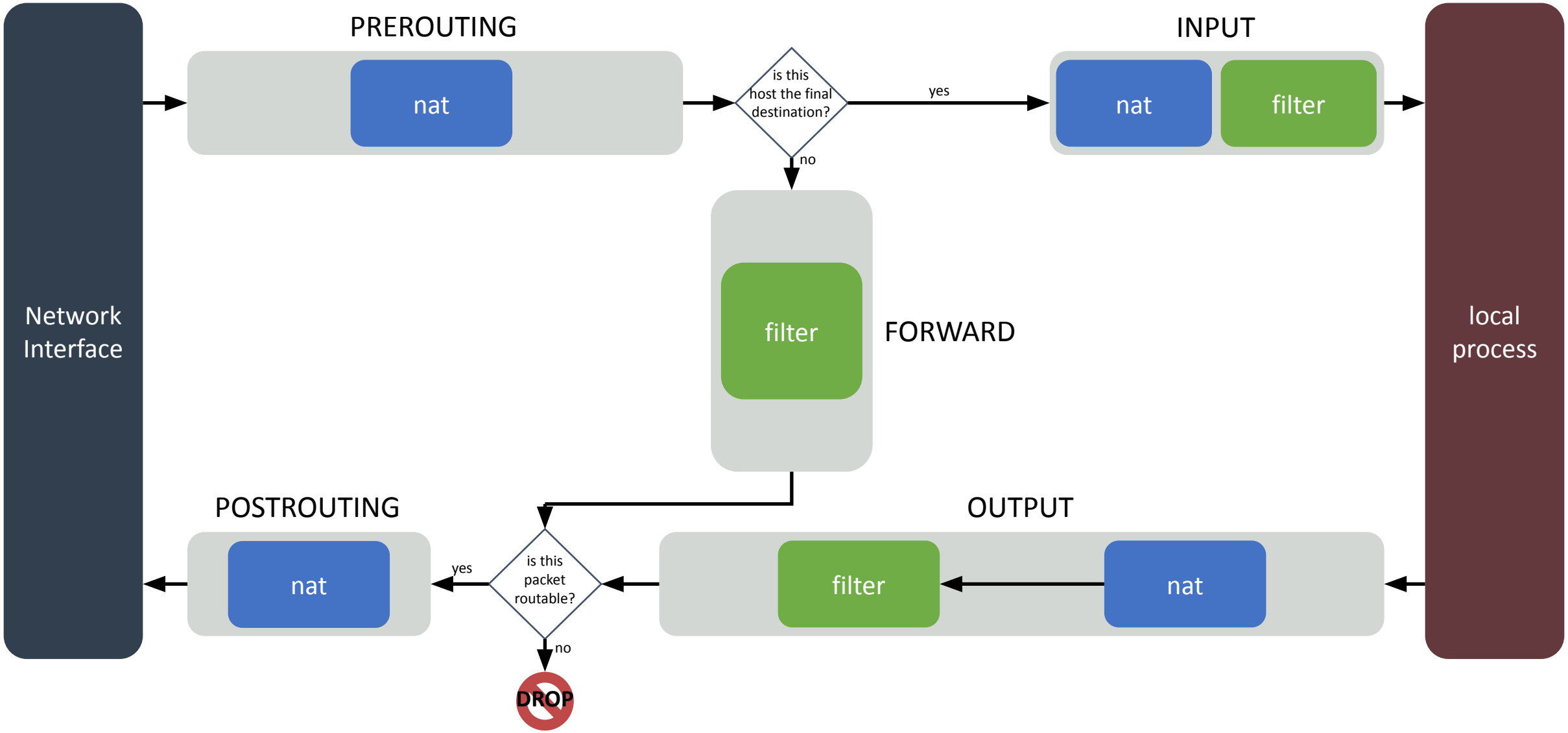
- The netfilter project has created nftables
  - Positioned as a replacement for iptables
  - Completely different configuration syntax :^(
  - As of October 2021, nftables is the default firewall backend for Ubuntu
    - iptables command is still present and gets translated to nftables
- iptables will be available for a long period of time in the future, even if nftables matures

Tables and Chains

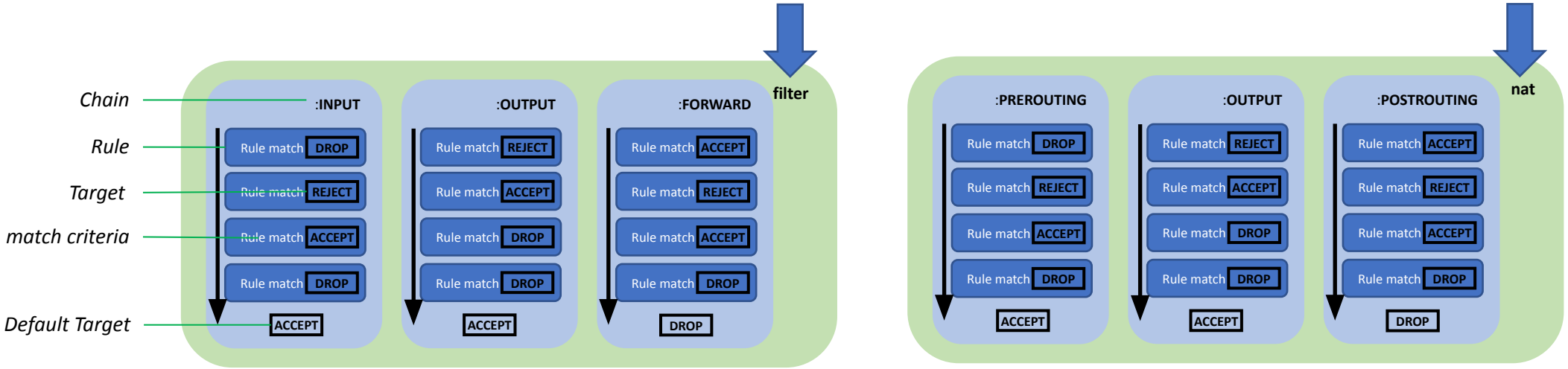
netfilter: how packets are processed

- When packets arrive from the network, the netfilter component of the Linux kernel recognizes and groups packets into streams or flows
  - netfilter performs Stateful Packet Inspection (SPI), which analyzes both packet headers and packet contents to track connections
- Each packet in a stream is processed by the firewall, with some performance optimizations
  - For example: NAT rules are only determined for the first packet in a stream, then all subsequent packets receive the same processing

Netfilter packet traversal - quick overview



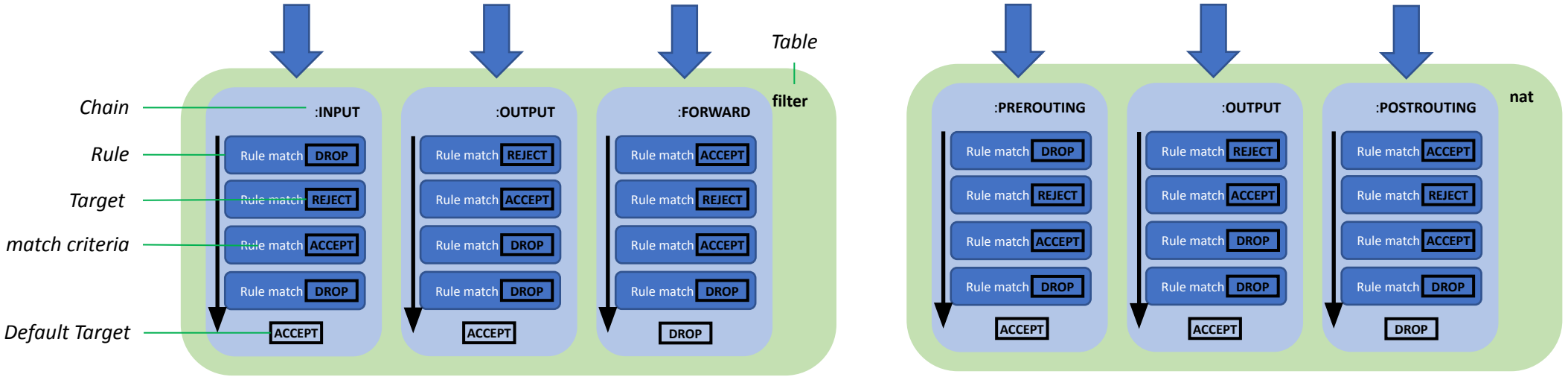
Tables



- An iptables table is a data structure that contains a number of chains. Several different tables are defined.
- The default table acted upon by the iptables command is "filter".

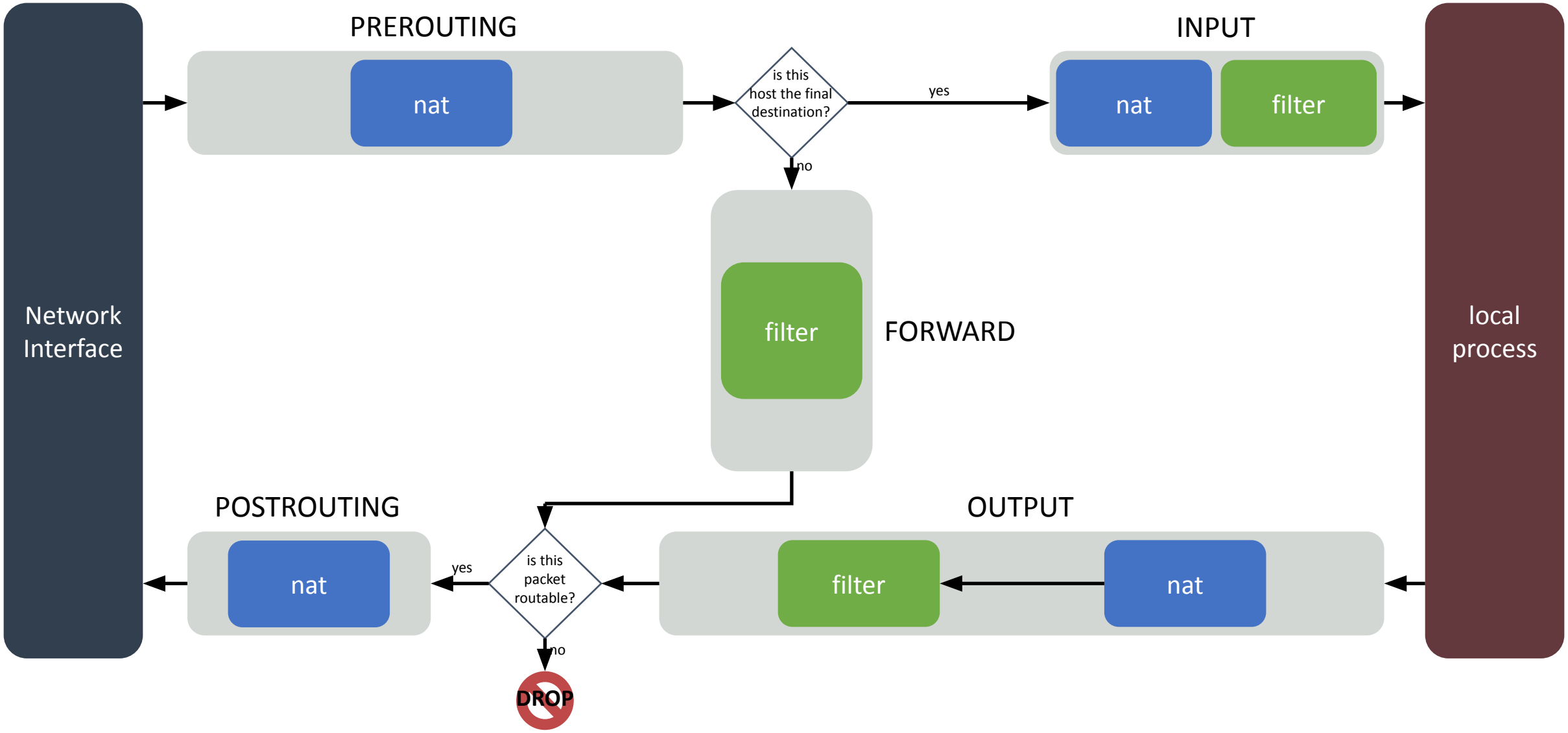
By convention, table names are given in lower-case, while chain names are given in upper-case.

Chains

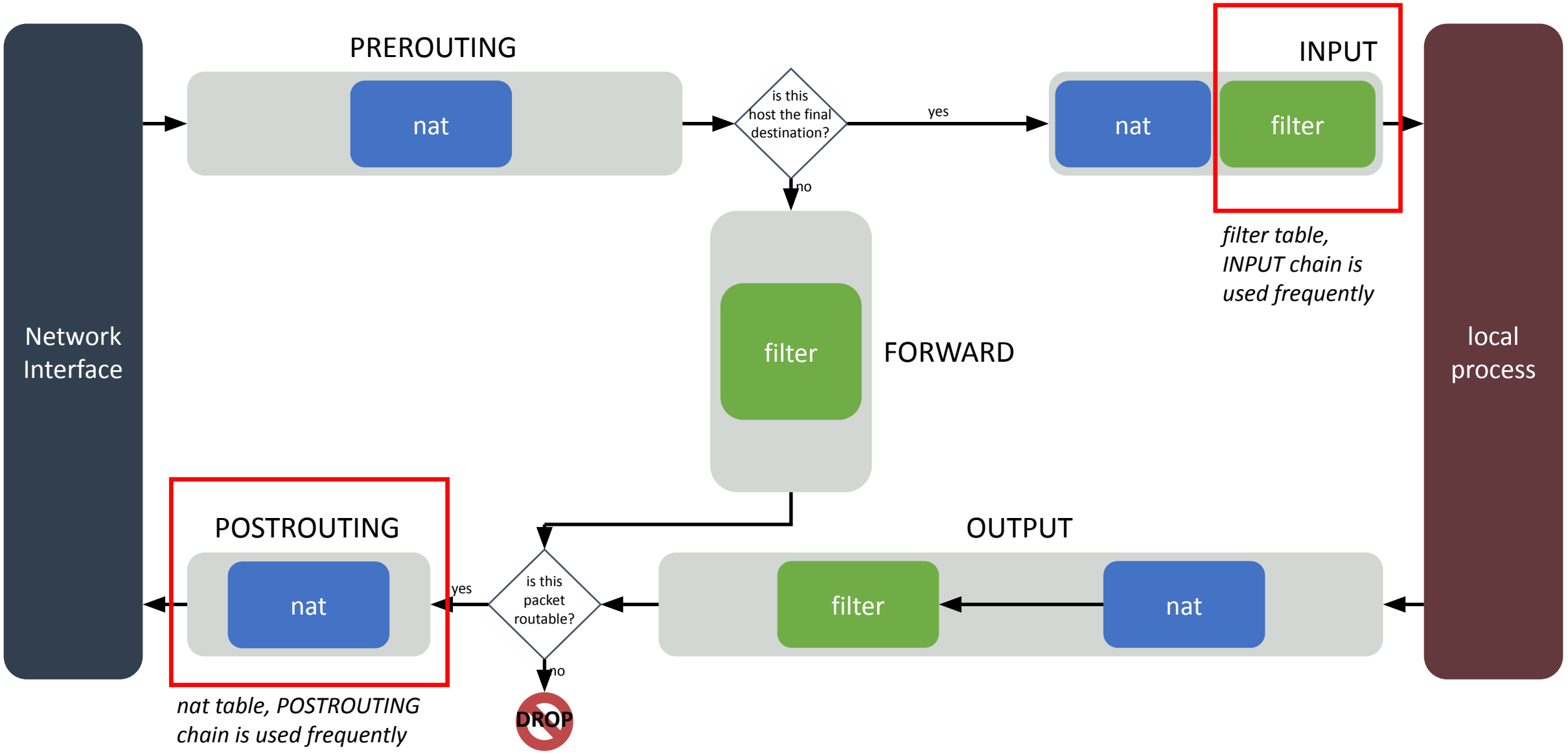


- Once it is decided a packet is matched against a certain table, the packet is matched against rules contained by that tables' chains.
  - Within a chain, a packet starts at the top (or head) of the chain and is matched rule-by-rule
  - When a match is found, processing normally jumps to another chain or target
- If a packet reaches the end of a chain, the default policy for that chain is applied

Netfilter packet traversal - overview



Netfilter packet traversal - overview



iptables: Tables and Chains

provides the 'statefullness' of the firewall

Table	**filter**	**nat**	*rarely used* **mangle**	**conntrack** *tracks incoming and outgoing packets **across the flow***	*rarely used* **raw** *provides opt-out of connection tracking with 'NOTRACK'*
PREROUTING chain		**PREROUTING**	**PREROUTING**	**PREROUTING**	**PREROUTING**
INPUT chain	**INPUT**	**INPUT**	**INPUT**		
FORWARD chain	**FORWARD**		**FORWARD**		
OUTPUT chain	**OUTPUT**	**OUTPUT**	**OUTPUT**	**OUTPUT**	**OUTPUT**
POSTROUTING chain		**POSTROUTING**	**POSTROUTING**		

processing order

iptables: filter and nat tables

decisions to let traffic through or not ('filtering')

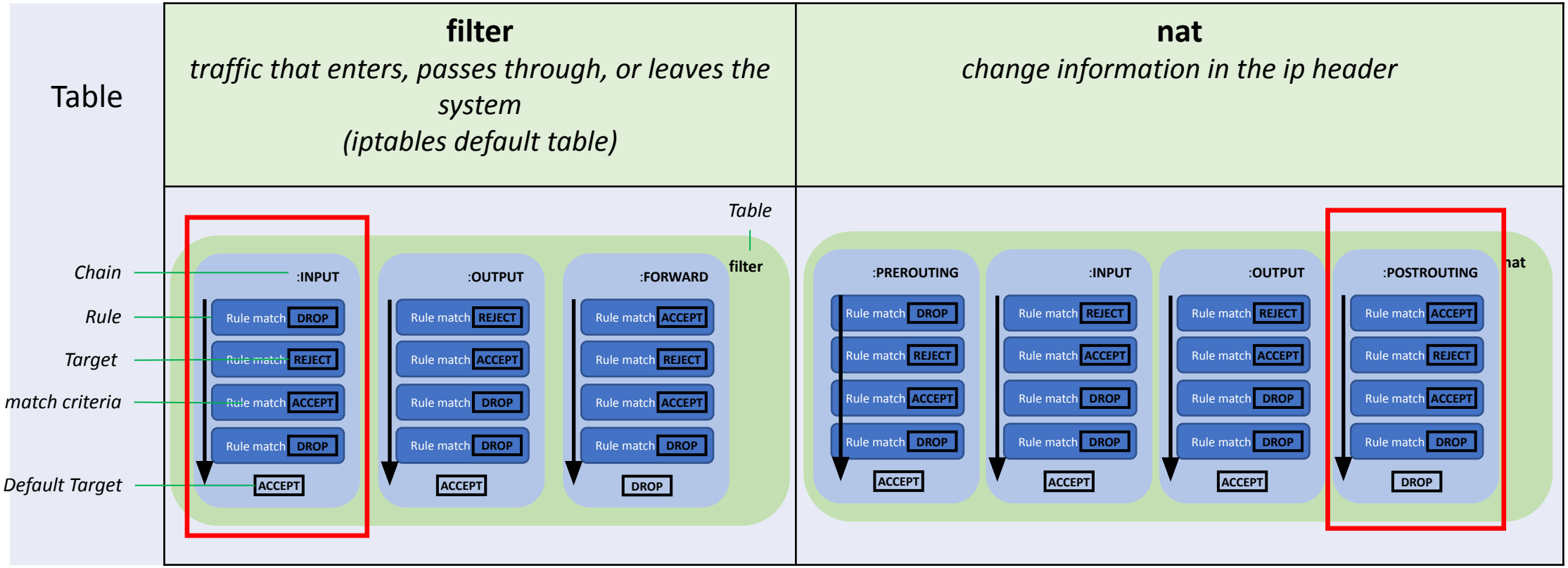
how and when are the source and destination of packages changed

Table	**filter** *traffic that enters, passes through, or leaves the system* *(iptables default table)*	**nat** *change information in the ip header*
Chains	**INPUT** *Handles packets destined for local sockets* **FORWARD** *Handles packets routed through this host to another destination* **OUTPUT** *Handles locally generated packets*	**PREROUTING** *Alters packets as they arrive from the network interface* **OUTPUT** *Alters locally-generated packets before routing them* **POSTROUTING** *Alters packets as they are about to be sent out from the host, **after** routing decision has been taken*

iptables: filter and nat tables

decisions to let traffic through or not ('filtering')

how and when are the source and destination of packages changed



Stateless vs. statefull firewall

- Stateless firewall
  - Packet filtering, usually only in the network layer (OSI). Sometimes there are protocols that are in a higher layer.
- Stateful firewall
  - Everything a stateless firewall does.
  - Also tracks whether certain packets have been seen before in a given session and applies access policies to packets based on what has already been seen for a given connection.
- netfilter/iptables are statefull thanks to the conntrack table

NAT masquerading with the conntrack Table

- **Connection tracking ("conntrack")** is an important kernel feature used by Linux machines to keep track of TCP connections going out and coming in.
- This connection tracking mechanism allows Linux machines to accurately send packets that are NAT-ed to the exact internal machines, which initiated the connection.
- The complete UDP/TCP connection status is stored in the connection tracking or **conntrack table**, containing
  - Ip addresses
  - Protocols
  - Port numbers
  - Status of the connection

```
cat /proc/net/ip_conntrack
```

Conntrack use cases

- **NAT** relies on the connection tracking information so that it can translate all packets in a flow in the same way.
  - E.g. when a pod accesses a Kubernetes service, NAT is used by kube proxy's load balancing to redirect the connection to a particular backend pod.
- **Stateful firewalls**, such as Calico, rely on the connection tracking information to exactly whitelist "response" traffic, once the outgoing connection has been established.

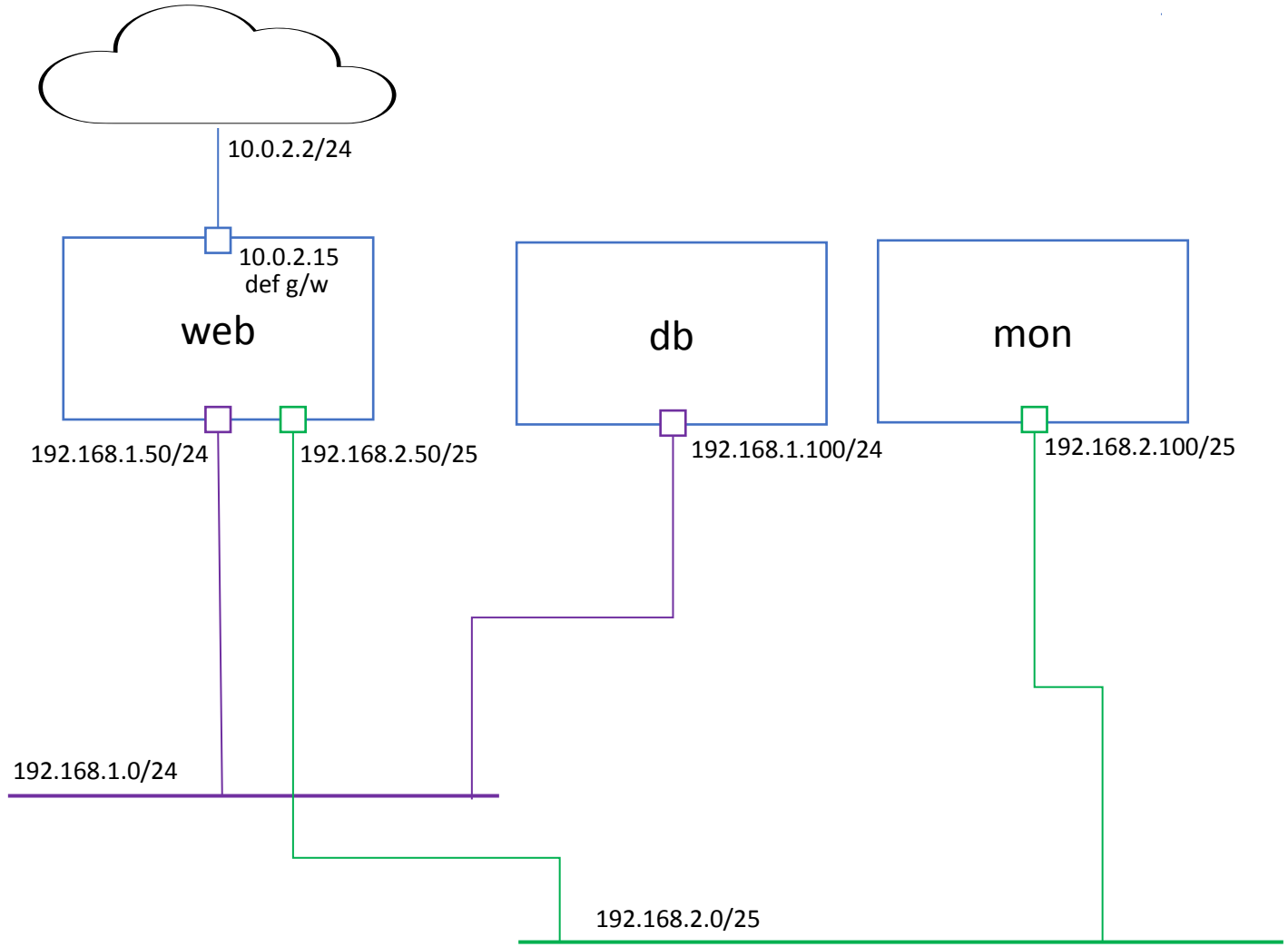
Conntrack gebruiken

- Op de router (in het lab, bv op 'web')
  - Installeer het package
    - `sudo apt install conntrack`
    - `sudo yum install conntrack`
  - `sudo conntrack -L`
  - `sudo conntrack -L --src-nat`
  - `man conntrack`

iptables LAB

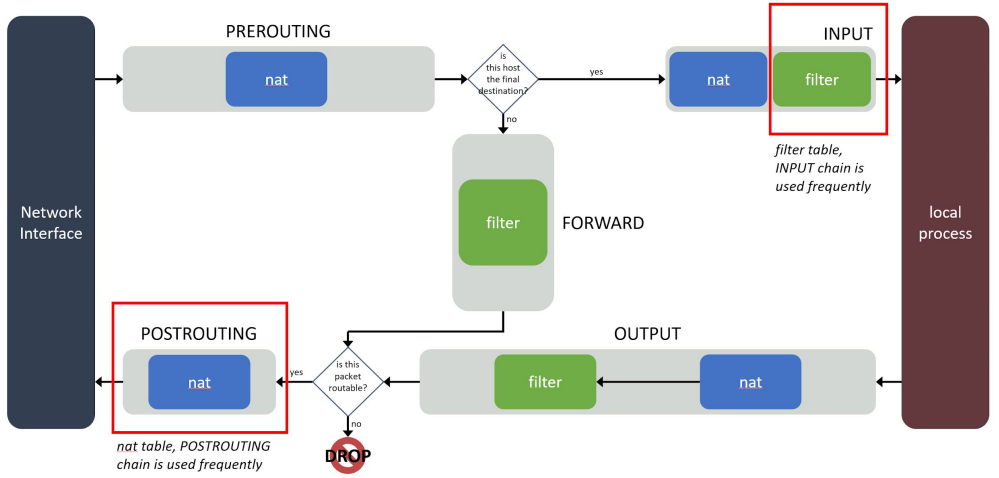
Lab set-up (PowerShell, vagrant)

- Fix vagrant network error
  - `sudo vim /etc/vbox/networks.conf`
    - `* 0.0.0.0/0 ::/0`
  - `sudo reboot now`
- Clone nieuwe environment
  - `cd ~`
  - `git clone https://github.com/PXLSystemsAdvancedII/sysadv2-2223`
  - `cd ~/sysadv2-2223/linux/iptables_lab`
- Check config
  - `code Vagrantfile`
- Start VMs
  - `vagrant up`
- Check ssh ports (optionally)
  - `vagrant ssh-config web`
  - `vagrant ssh-config db`
  - `vagrant ssh-config mon`
- Password ssh is opgezet tussen de machines
  - `user: vagrant`
  - `passwd: vagrant`
- Login vms en check ip, route table
  - `vagrant ssh web`
  - `ip a`
  - `ip r`
- Na de oefening afzetten en verwijderen
  - `vagrant destroy -f`



Hands-on: NAT

- enable routing/NAT op web:
  - zet ip forwarding aan
  - zet nat aan
  - check rules

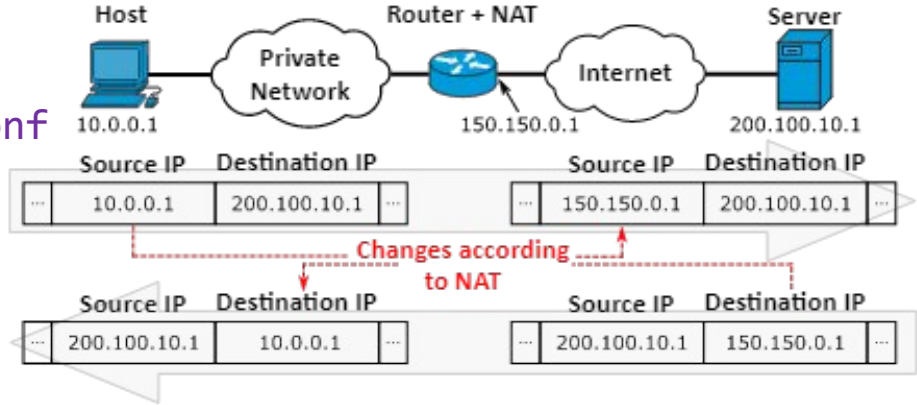


web

```
echo "net.ipv4.ip_forward=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

sudo iptables -L
sudo iptables -L -t nat -v
```



The iptables Service (EL7)

- On Enterprise Linux 7, firewalld is the default service for managing the system firewall
- We can install the iptables-services package to use iptables instead of firewalld
  - Either remove firewalld package or stop, disable, and mask the service with `systemctl`
  - Start and enable the iptables service

Hands-on: firewalld service door iptables service vervangen

- op db:
  - disable/mask firewalld
  - install & enable iptables-services
  - add default gateway
  - list iptables "filter" table
    - default view
    - verbose view
    - verbose numbered view

db

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld

ping google.com

ip r
sudo ip route add default via 192.168.1.50

ping google.com

sudo yum install iptables-services -y
sudo systemctl enable iptables
sudo systemctl start iptables

sudo iptables -L
sudo iptables -L -v
sudo iptables -L -v -n

sudo cat /etc/sysconfig/iptables
```

```
[vagrant@db ~]$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   31  2608 ACCEPT     all  --  any    any     anywhere             anywhere             state RELATED,ESTABLISHED
    0     0 ACCEPT     icmp --  any    any     anywhere             anywhere
    0     0 ACCEPT     all  --  lo     any     anywhere             anywhere
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere             state NEW tcp dpt:ssh
    0     0 REJECT     all  --  any    any     anywhere             anywhere             reject-with icmp-host-prohi
bited

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 REJECT     all  --  any    any     anywhere             anywhere             reject-with icmp-host-prohi
bited

Chain OUTPUT (policy ACCEPT 10 packets, 1520 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

Default Packet-Filtering Policy

- Een firewall is een device om access control policies te implementeren
- Je hebt 2 basis manieren om de default policy van je firewall te configureren
  - **Default DENY** en dan specifieke pakketten toelaten
    - Meest aanbevolen aanpak
    - Gemakkelijker om een **veilige** firewall op te zetten
    - Je moet wel het communicatie protocol voor elke service die je wil toelaten begrijpen
    - Meer werk om op te zetten
  - **Default ACCEPT** en dan specifieke pakketten weigeren
    - Gemakkelijker om initieel op te zetten
    - Je moet er aan denken om elke service die geïnstalleerd wordt te blokkeren tegen misbruik
    - Configuratie en onderhoud is meer werk
    - **minder veilig**

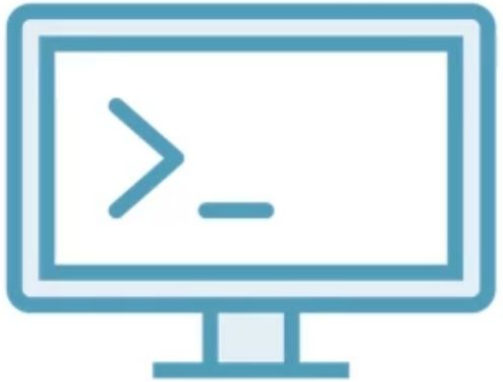
Basic Firewall Example

- This simple example firewall has default ACCEPT policies for all chains in the filter table, but will REJECT unmatched incoming packets at the end of the INPUT chain
- Only packets that are related to established connections, ICMP requests (such as ping), and incoming SSH connections will be permitted to pass through the firewall

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

iptables commando



iptables

- **Table** `-t` **defaults to** `FILTER`
- **Command (insert** `-I`**/append** `-A`**/delete** `-D`**)**
- **Chain (**`INPUT/FOWARD/OUTPUT`**)**
- **Match** `-m` **(IP, protocol, port)**
- **Target** `-j` **(**`ACCEPT/REJECT/DROP/LOG`**)**

```
iptables -t filter -I INPUT -m tcp -p tcp--dport 80 -j ACCEPT
```

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

End of table denoted with COMMIT

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

End of table denoted with COMMIT

Permitting Related Traffic

- Incoming packets related to connections initiated by our system need to be permitted, otherwise we couldn't receive responses to network requests
- **`-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT`**
  - **A**ppends a rule to the INPUT chain that **m**atches the state of the packet
  - If the **state** is RELATED or ESTABLISHED, **j**ump to the ACCEPT target

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

End of table denoted with COMMIT

Responding to ping

- The security parameters of a system determine whether or not ICMP requests will be permitted through the firewall
- To permit them, use:
- **`-A INPUT -p icmp -j ACCEPT`**
  - **A**ppend a rule to the INPUT chain for the icmp **p**rotocol. Any packet using the icmp protocol causes a **j**ump to the ACCEPT target.

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

End of table denoted with COMMIT

The Loopback Interface

- Linux systems have a special loopback interface (lo), which enables network connections to be made between different applications running on the local system
  - Traffic normally should be allowed on this interface, otherwise some programs will not work properly
- **`-A INPUT -i lo -j ACCEPT`**
  - **A**ppends a rule to the INPUT chain. All packets arriving on the loopback (lo) **i**nterface cause a **j**ump to the ACCEPT target.

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

End of table denoted with COMMIT

Allowing Incoming SSH Connections

- Secure Shell (SSH) is a standard remote access protocol widely used for remote system access and administration
  - By default, it runs on TCP port 22
- **`-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT`**
  - **A**ppend a rule for the tcp **p**rotocol to the INPUT chain, which **m**atches the state of the packet. If the **state** is a NEW packet, also **m**atch a tcp connection with a **d**estination **port** of 22, then **j**ump to the ACCEPT target if matched.

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

End of table denoted with COMMIT

Rejecting Packets

- Sometimes, we want to send an explicit rejection message back to the sender, instead of quietly dropping a packet
- **`-A INPUT -j REJECT --reject-with icmp-host-prohibited`**
  - **A**ppend a rule to the INPUT chain that unconditionally **j**umps to the REJECT target, sending an ICMP host-prohibited message back to the sender.
  - Instead of host-prohibited, we could send a wide variety of alternate messages back to the sender instead.

Example: Allow SSH

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Name of the table being configured (default is filter if none given)

Default packet counter values for each chain

Default policies for each chain

INPUT chain is built by appending rules in order

This firewall does not also function as a network router

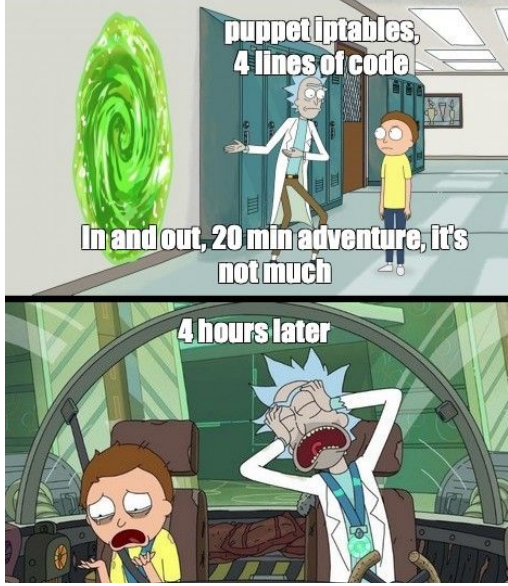
End of table denoted with COMMIT

iptables commando

Listing rules as tables	iptables	-L
Listing rules as configured	iptables	-S
Listing Rules with Counters	iptables	-L -n -v
Clear Packet Counters	iptables	-Z
List a particular table	iptables	-L
Flush Rules (all or specified)	iptables	-F
Append a new rule	iptables	-A
List rules with line numbers	iptables	-L --line-numbers
Insert a rule at a location	iptables	-I table [n]
Replace a rule at a location	iptables	-R table [n]
Delete a rule	iptables	-D
Delete a rule at a location	iptables	-D table [n]

```
man iptables
```

Hands-on - add / remove rules, persistency



- db
  - `sudo vi /etc/sysconfig/iptables`
  - `sudo iptables -I INPUT -s 192.168.2.0/24 -j REJECT`
  - `sudo iptables -L -v`
  - `sudo iptables -I INPUT -s 192.168.2.100 -p tcp -m tcp --dport 22 -j ACCEPT`
  - `sudo iptables -L -v`
  - `sudo iptables -L --line-numbers`
  - `sudo iptables -D INPUT 1`
  - `sudo iptables -D INPUT 1`
  - `sudo iptables -L --line-numbers`
  - `sudo iptables -F`
  - `sudo iptables -L`
  - `sudo service iptables reload`
  - `sudo iptables -L`
  - `sudo iptables -I INPUT -s 192.168.2.0/24 -j REJECT`
  - `sudo iptables -I INPUT -s 192.168.2.100 -p tcp -m tcp --dport 22 -j ACCEPT`
  - `sudo service iptables save`
  - `sudo vi /etc/sysconfig/iptables`
- ssh vanuit mon is nu toegelaten

Working with iptables

- Basic command: `iptables`
  - Possible to build the firewall one rule at a time by running this command repeatedly
  - Firewall is not saved upon reboot
- Easier solution: `iptables-restore`
  - Enables the firewall to be written as a plain text file, then loaded and made active with one command
  - Doesn't solve the firewall saving issue

Using `iptables-restore`

- Run the command:

  ```
  iptables-restore <filename>
  ```

  where `<filename>` is the path to the firewall configuration file
- By default, this command will flush the current firewall rules and replace them with the ones from the configuration file

Saving the Firewall Configuration

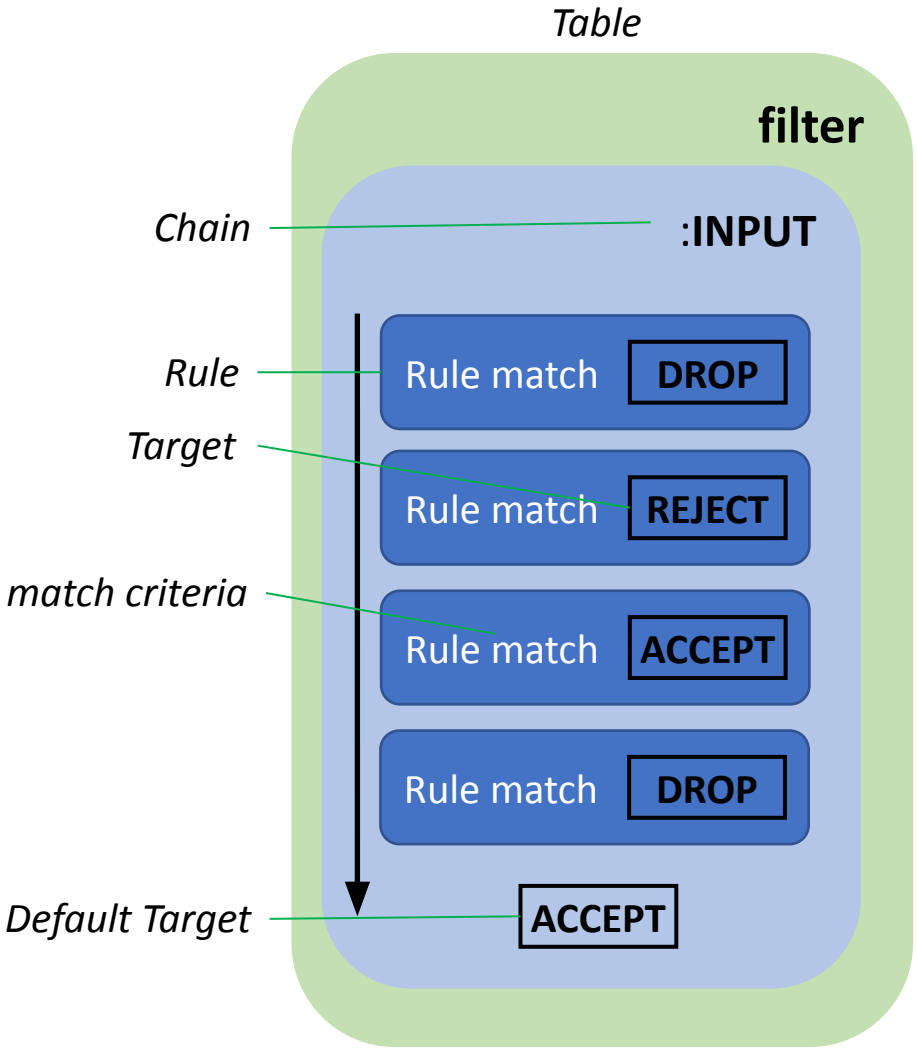
- Once we have loaded the firewall configuration we want, we can save it using the command:

```
service iptables save
```

- This command will save the firewall configuration to `/etc/sysconfig/iptables`, where it will be loaded automatically at boot time

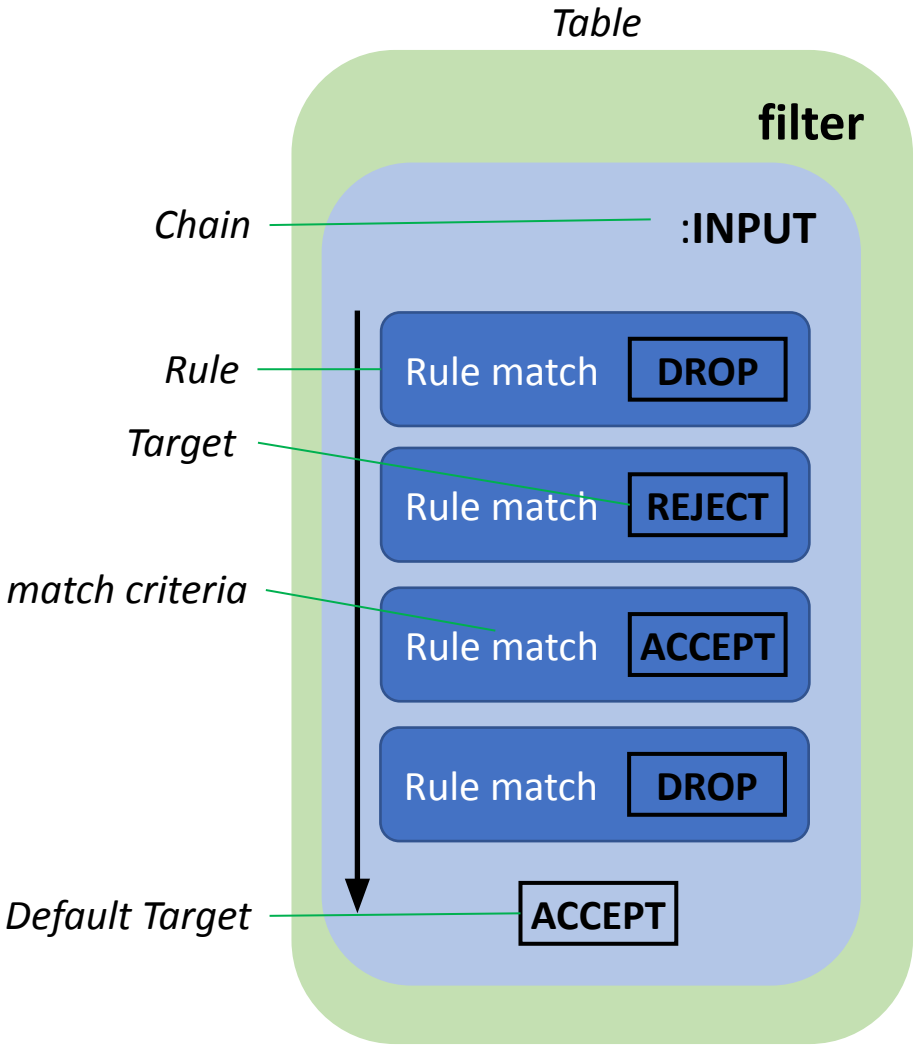
Rules and targets

Rules



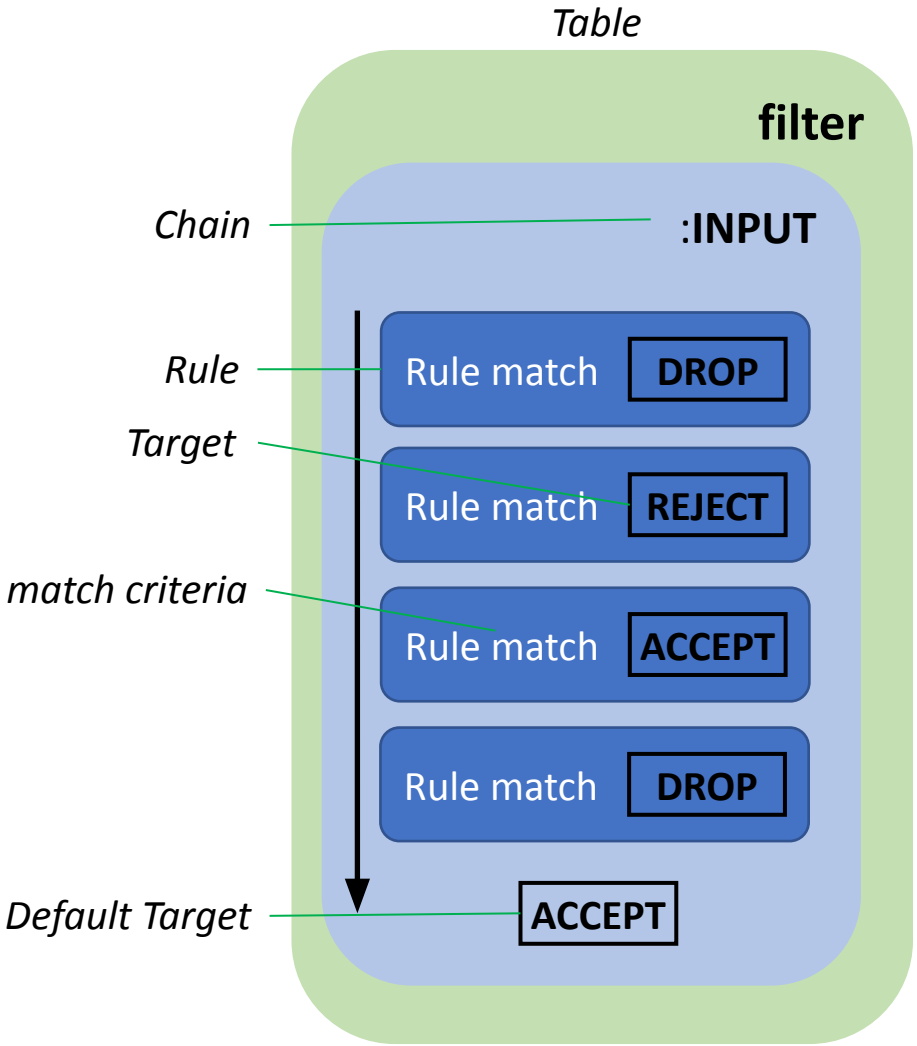
- Processing in iptables is done by trying to match packets using *rules*, which may specify properties
  - If a packet matches the properties, the rule is applied
  - If no matching properties are given, the rule is always applied
- Rules normally specify a *target* to which processing jumps if the rule matches

Targets



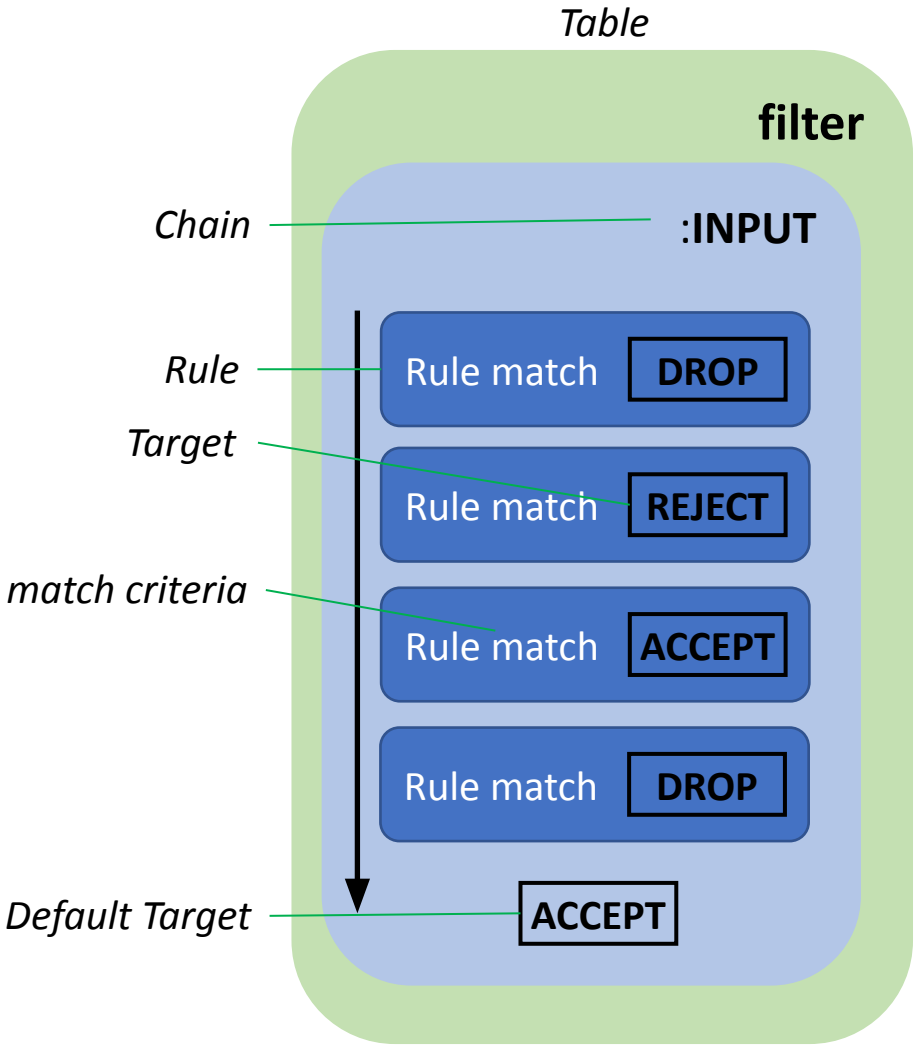
- As iptables processes a packet along a chain, the rules specified for the chain might cause the packet to jump to a target, which could be a special target or another chain
  - Eventually, the packet will wind up at one of the special targets built into iptables
- If the packet reaches the end of a chain, the chain's default policy specifies the target to which processing will jump

Chain Rules are in Order



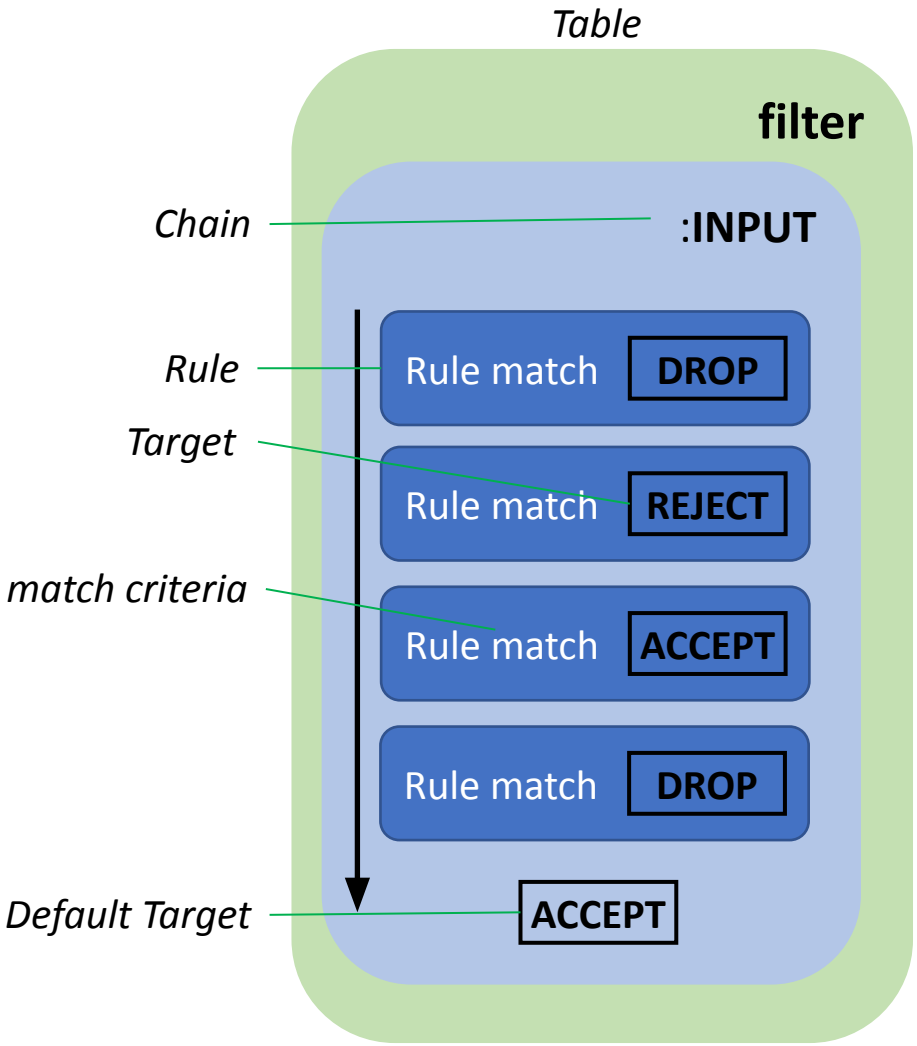
- Rules in a chain are processed in order from the top (or head) of the chain to the end
- The first matching rule that causes a jump either to another chain or to a special target terminates processing of that chain
  - The exception is that some targets are non-terminating, which means they RETURN to the original chain after processing

Basic Special Targets



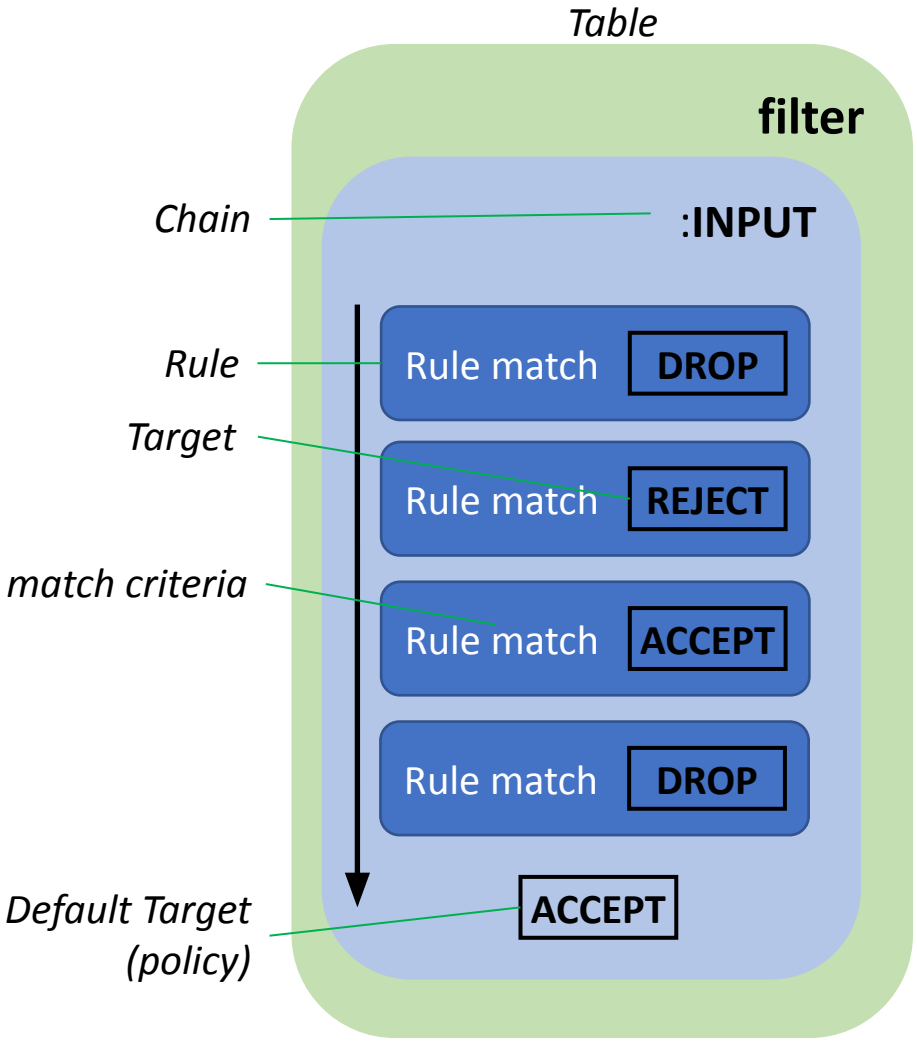
- **ACCEPT**
  - The packet is accepted at this point and goes on for further processing by the next table
  - At the security table on an inbound firewall, an ACCEPTed packet normally goes to the network socket of the application
- **DROP**
  - The packet is discarded
  - No response or acknowledgment is sent back to the transmitter

Packet Rejection



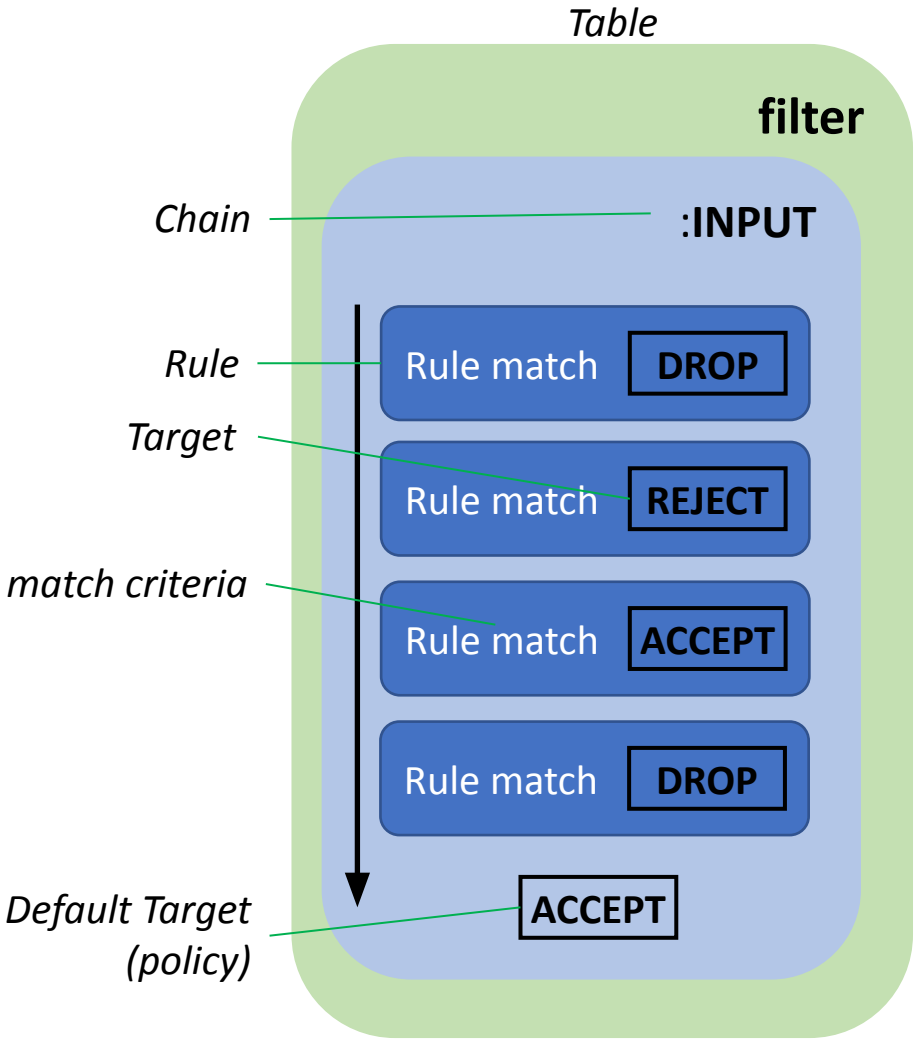
- If the firewall is set to reject a packet, it may do so by using either the DROP or REJECT target
- **REJECT** sends an error message packet (using ICMP) back to the sender
  - Desirable for correct operation of certain protocols, like TCP
  - Less desirable for security reasons
    - Acknowledges existence of the system, inviting adversaries to continue to attack it

Default Policy



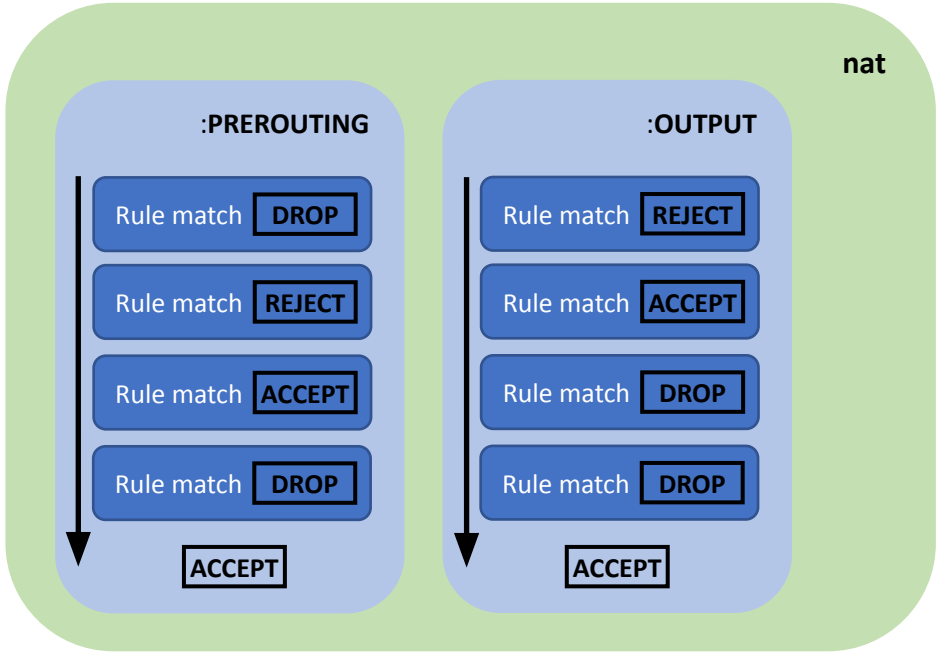
- Each chain in each table has a default policy, which is normally one of ACCEPT, DROP, or REJECT
- If a packet does not match a rule in the chain, the default policy will be applied to the packet

Source NAT Targets



- **MASQUERADE**
  - Enables Source NAT but uses the IP address of the outgoing interface as the address used for rewriting
  - Useful when the router itself obtains an address from DHCP
- SNAT
  - Rarely used
  - Enables Source NAT to rewrite source addresses to a static IP address, or load balance between a set of static IP addresses

Destination NAT Target



- **DNAT**
  - Enables the destination IP address of a packet to be rewritten
  - Normally used for port forwarding with inbound packets
  - Note that DNAT is only valid for use in the **nat** table, and only for the PREROUTING and OUTPUT chains

Logging

- LOG and ULOG targets
  - Allow detailed per-packet information to be stored in log files
  - Rules with non-terminating targets: chain processing continues after logging
- LOG target logs to system logging facility
  - journald or syslog
- ULOG target sends log data to a special interface
  - Various software daemons can listen to this interface and store logs in flexible locations (e.g. in a database)

Logging example

```
sudo iptables -A INPUT -m state --state NEW -p tcp --dport 80 -j LOG --log-prefix
"NEW_HTTP_CONN: "
```

- A request on port 80 from the local machine, then, would generate a log in dmesg that looks like this (single line split into 3 to fit this document)

```
[4304885.870000] NEW_HTTP_CONN: IN=lo OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00
SRC=127.0.0.1 DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=58288 DF PROTO=TCP
SPT=53981 DPT=80 WINDOW=32767 RES=0x00 SYN URGP=0
```

- The above log will also appear in `/var/log/messages`, `/var/log/syslog`, and `/var/log/kern.log`.
  - This behavior can be modified by editing `/etc/syslog.conf` appropriately or by installing and configuring ulogd and using the ULOG target instead of LOG.
- The ulogd daemon is a userspace server that listens for logging instructions from the kernel specifically for firewalls, and can log to any file you like, or even to a PostgreSQL or MySQL database.
- Making sense of your firewall logs can be simplified by using a log analyzing tool such as logwatch, fwanalog, fwlogwatch, or lire.

end

Agenda

- Firewall functions & features
- Linux firewall architectuur
  - netfilter
  - iptables
- Tables & Chains
- Stateless vs. statefull
  - NAT & connection tracking
- LAB
- Rules & Targets

Oefening: port forwarding

- zorg ervoor dat het ssh verkeer dat toekomt op de host 'mon' geforward wordt naar de host 'db'