


# Firewalls

**Instructor: Khaled Diab**

# So far...

---

- Network Analysis and Monitoring
  - Deployment
  - Practice 
- Network Attacks
  - L2: ARP
  - L3: IP
  - L4: TCP
  - L5: DNS
- Today: Architectural solutions to protect against (some) network attacks



# Outline

---

- What is a Firewall?
- Types of Firewalls
  - Packet filtering ]
  - Proxy server ]
- Evading Firewalls

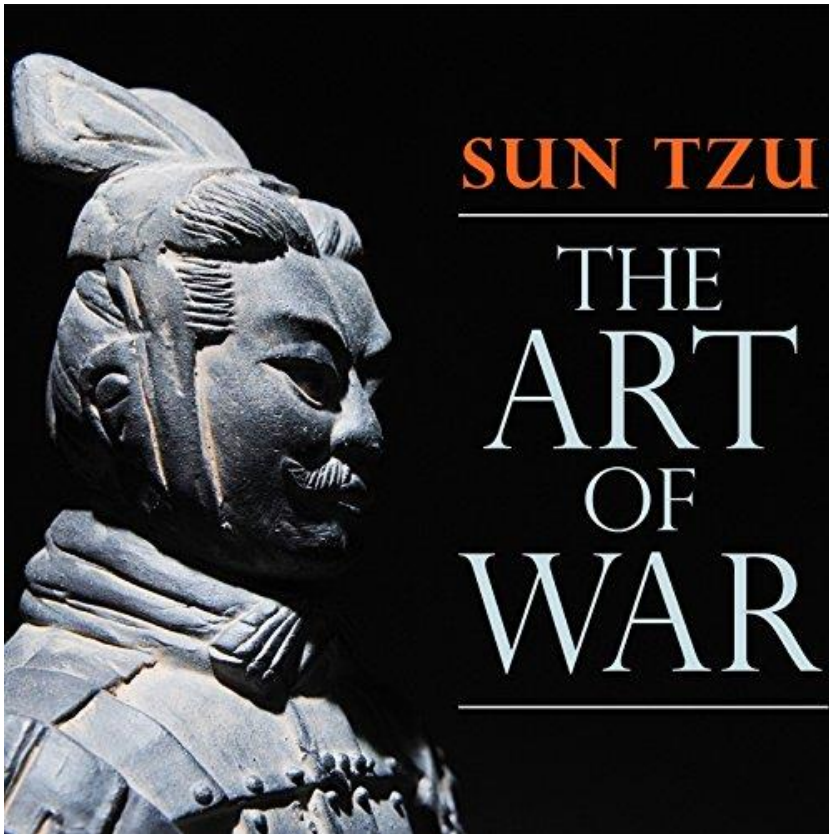
# Firewall Overview

# What is a Firewall?

---

*“On the day that you take up your command, block the frontier passes, destroy the official tallies, and stop the passage of all emissaries”*

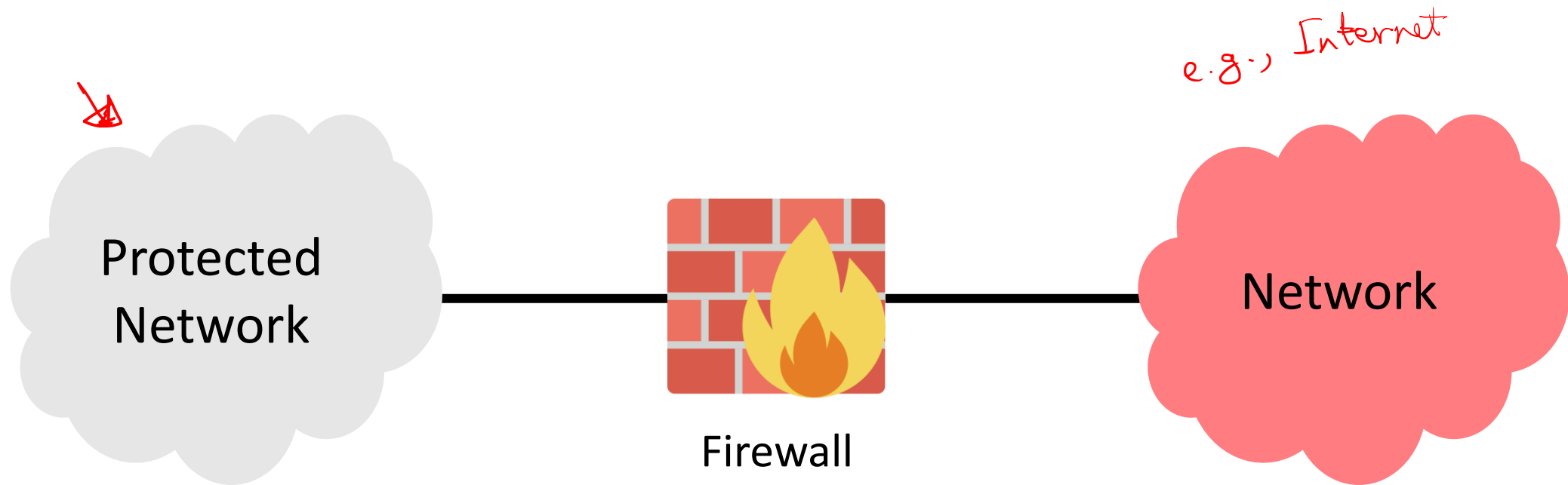
— The Art of War, Sun Tzu



# What is a Firewall?

---

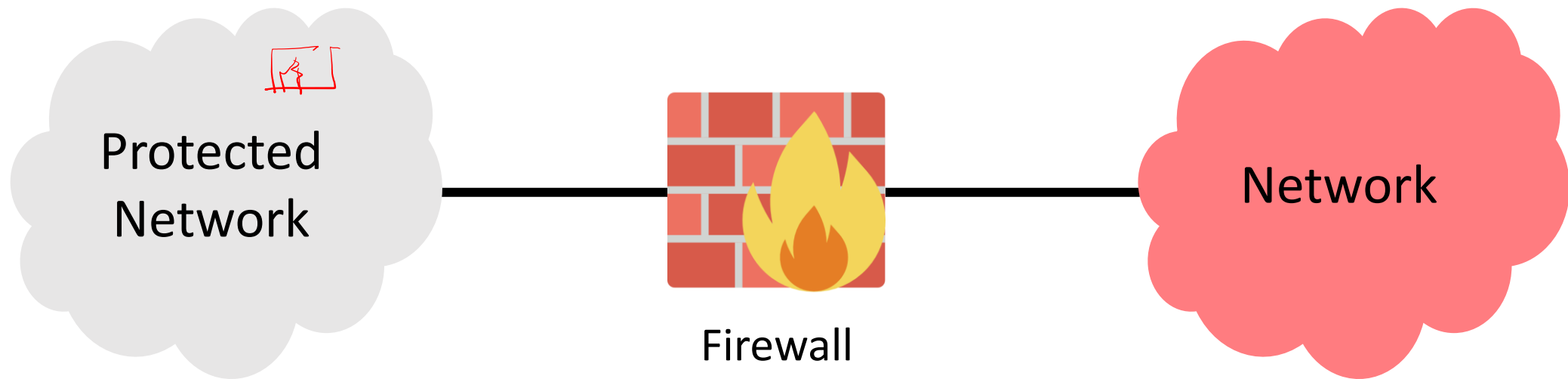
- A component that stops unauthorized traffic flowing from one network to another.
  - Often separates **trusted** and **untrusted** networks.



# What is a Firewall?

---

- Often separates **trusted** and **untrusted** networks.
- Differentiates networks within a trusted network.
- Can be implemented in software, hardware, or as a combination.



# Requirements of a Firewall [Bellovin and Cheswick'94]

---

- All traffic between two trust zones should pass through a firewall.
- Only authorized traffic, defined by the **security policy**, should be allowed to pass through.
- The firewall must be immune to penetration.



# Firewall Policy

---

- The rules that a firewall enforces.
- Rule types:
  - User control
  - Service control
  - Direction control

# Firewall Policy

---

- User Control
  - Controls access to data based on the user role (Who is accessing the data?)
  - Often used for users within a firewall zone
- Service Control
  - Access is controlled by the type of the service offered by the host protected by the firewall
  - Needs access to network address, port number, protocol etc.
- Direction Control
  - Allows traffic based on its direction: inbound or outbound.

telnet  
DNS...

# Firewall Actions

---

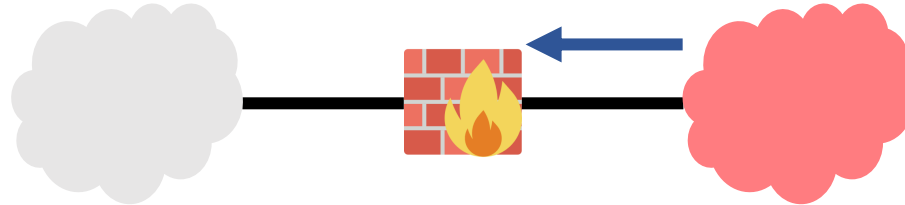
- Network packets going through a firewall result in one of three actions:
  - **ACCEPT**: Allowed to enter the protected host/network
  - **DENIED**: Not permitted to access the other side of the firewall
  - **REJECTED**: Similar to **DENIED**.
    - But the firewall attempts to tell the source of the packet about its decision.
    - How? **ICMP**

# Ingress and Egress Filtering

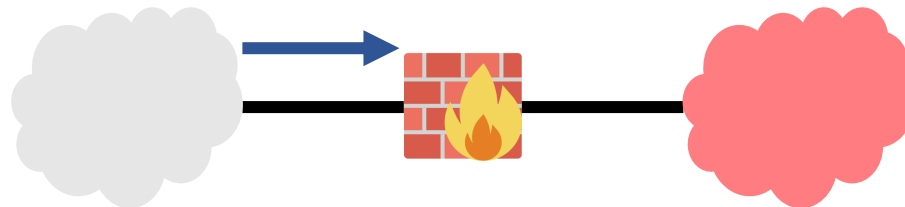
---

- Firewalls can inspect traffic from both directions.

- Ingress filtering



- Egress filtering



# Other Functions

---

- Besides **protecting** a network, a firewall may:
  - rewrite packet headers to route packets between networks
    - Example?
    - Machine works as a router
  - act as a NAT

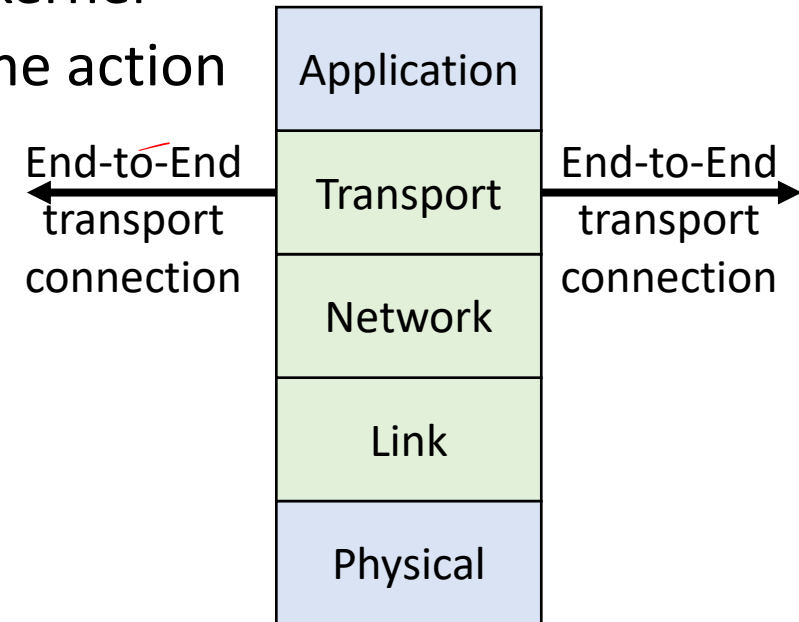
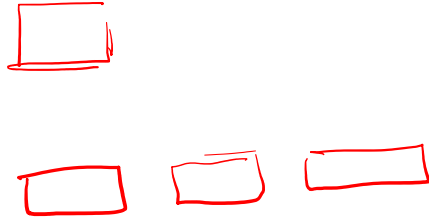


# Types of Firewalls

# Types of Firewalls

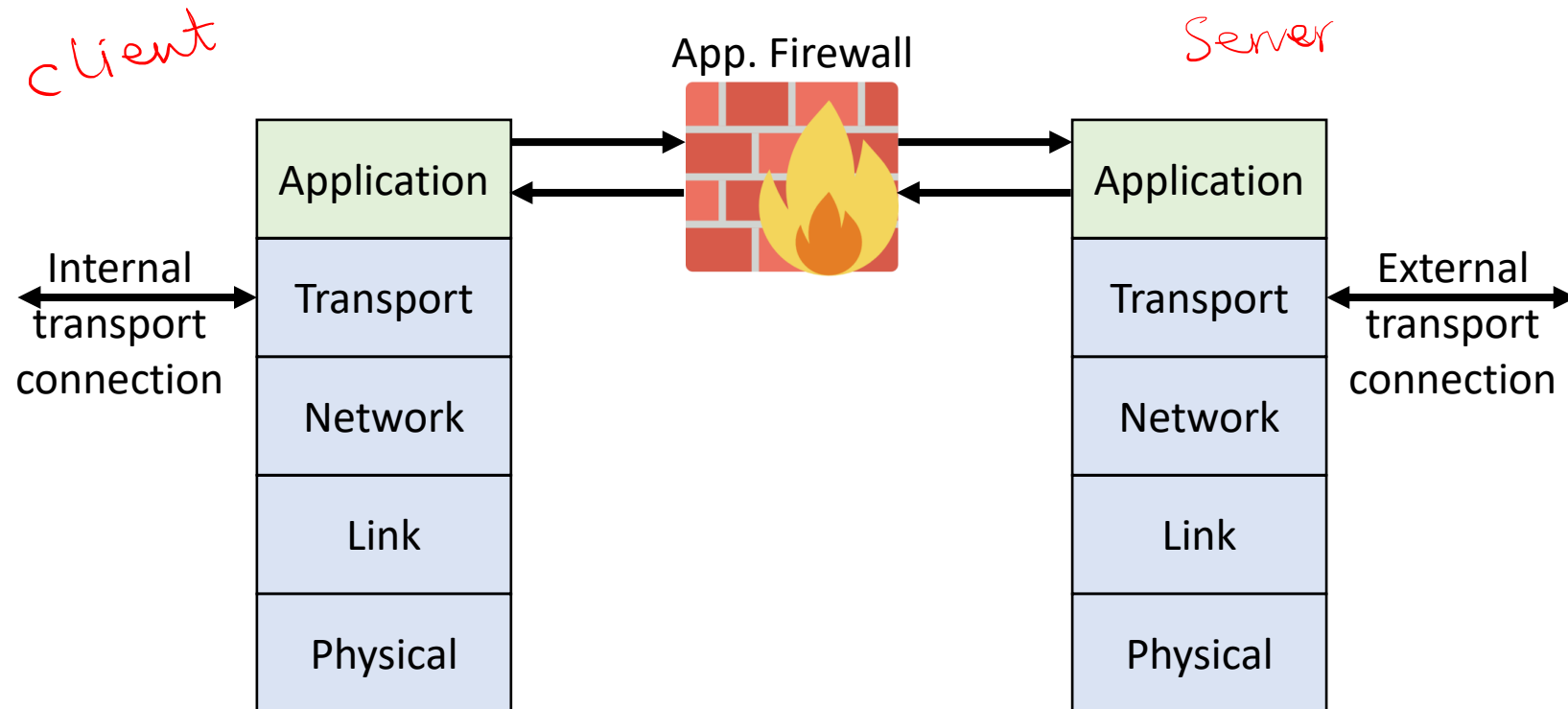
- Packet Filtering

- Most kernels implement ~~TCP~~/IP stack
- Filters are executed in the same address space of the kernel
- The kernel is in a position to immediately determine the action
- ~~Stateless~~ and Stateful<sup>2</sup> firewalls
  - Does a packet belong to a stream of traffic?



# Types of Firewalls

- Application Firewall
  - It is a proxy server ✓
  - Impersonates the intended recipient
  - Connection terminates at the proxy, and another connection starts from the proxy

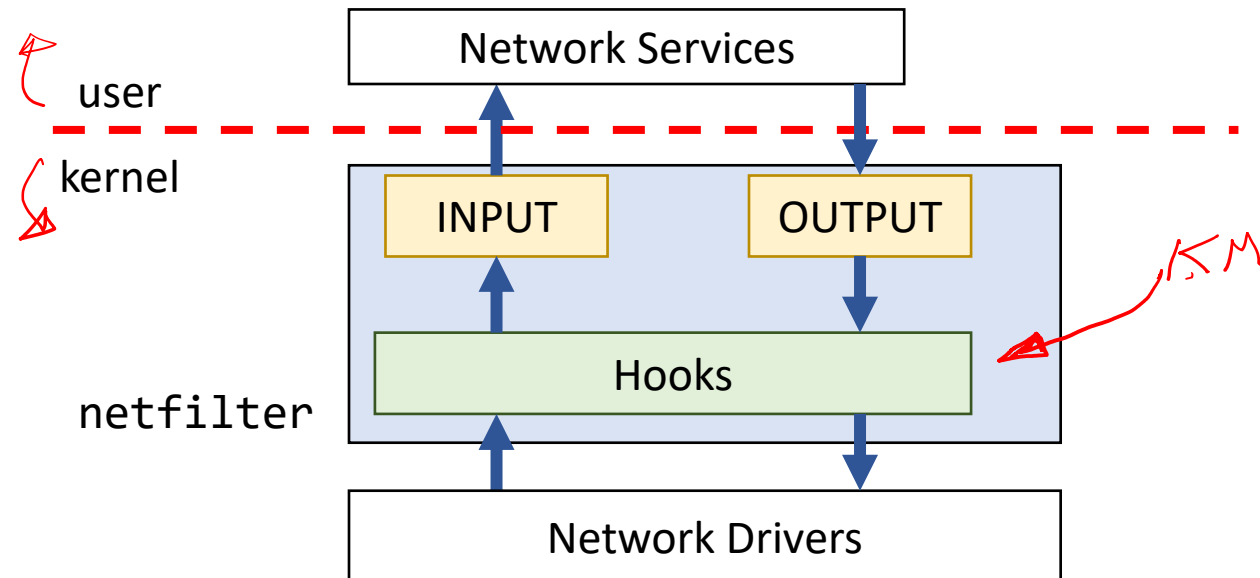




# Packet Filtering Firewall

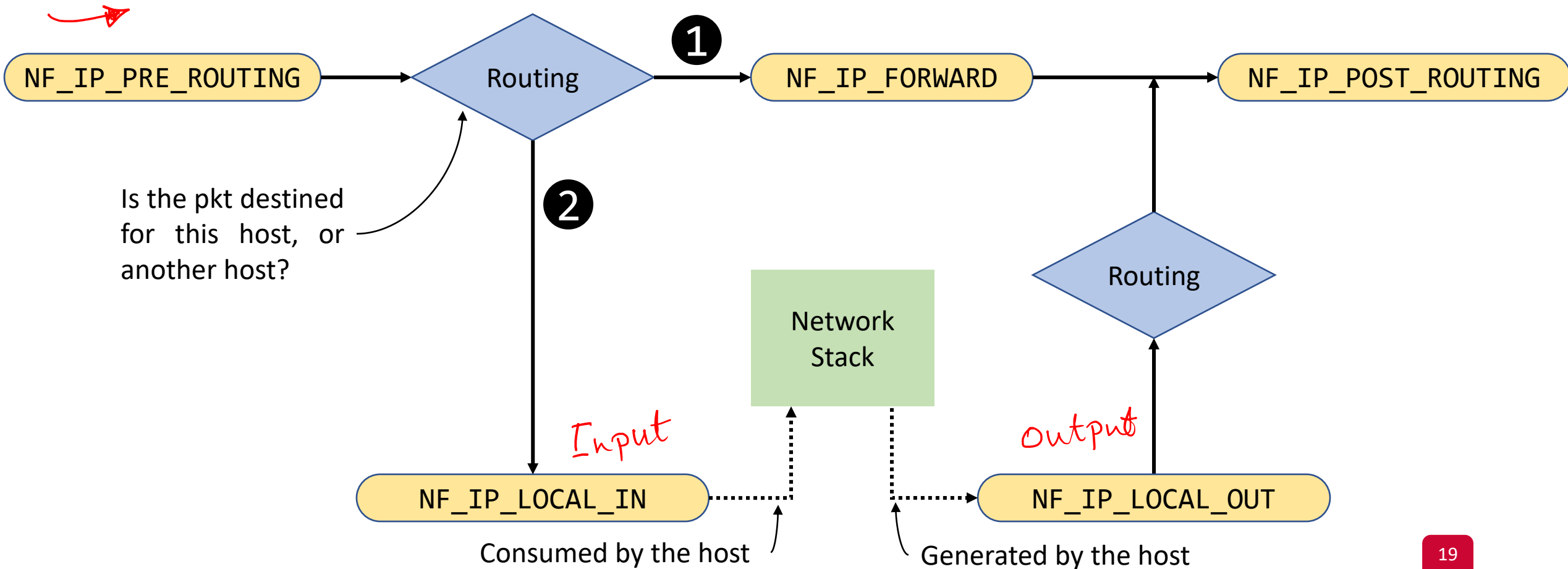
# netfilter

- A framework inside the Linux kernel
- Allows different networking-related functions to be implemented
  - Uses hooks that a program can register with
  - As packets traverse the the stack, they will trigger the kernel modules that have registered with these hooks



# netfilter Hooks

- A packet triggers the kernel modules that registered with netfilter hooks



# netfilter Calling Order

---

- Each registered kernel module provides a **priority** value
- netfilter calls a kernel module based on its priority
- What are possible decisions?

# netfilter Return Values

---

- Each registered kernel module returns one of these values:
  - **NF\_ACCEPT**: Let the packet go through the stack
  - **NF\_DROP**: Discard the packet
  - **NF\_QUEUE**: Pass the packet to the user space
  - **NF\_STOLEN**: Ask netfilter to forget this packet, and move responsibility to the calling module
  - **NF\_REPEAT**: Ask netfilter to call the calling module again

# Example: Block Outgoing Telnet Packets

- Logic

```
unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                        const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph + iph->ihl*4;

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}
```

Handwritten annotations:

- A red bracket above `struct sk_buff *skb` is labeled `Pkt`.
- A red arrow points to `iph->protocol`.
- A red arrow points to `tcph->dest` with the label `telnet`.

# Example: Block Outgoing Telnet Packets

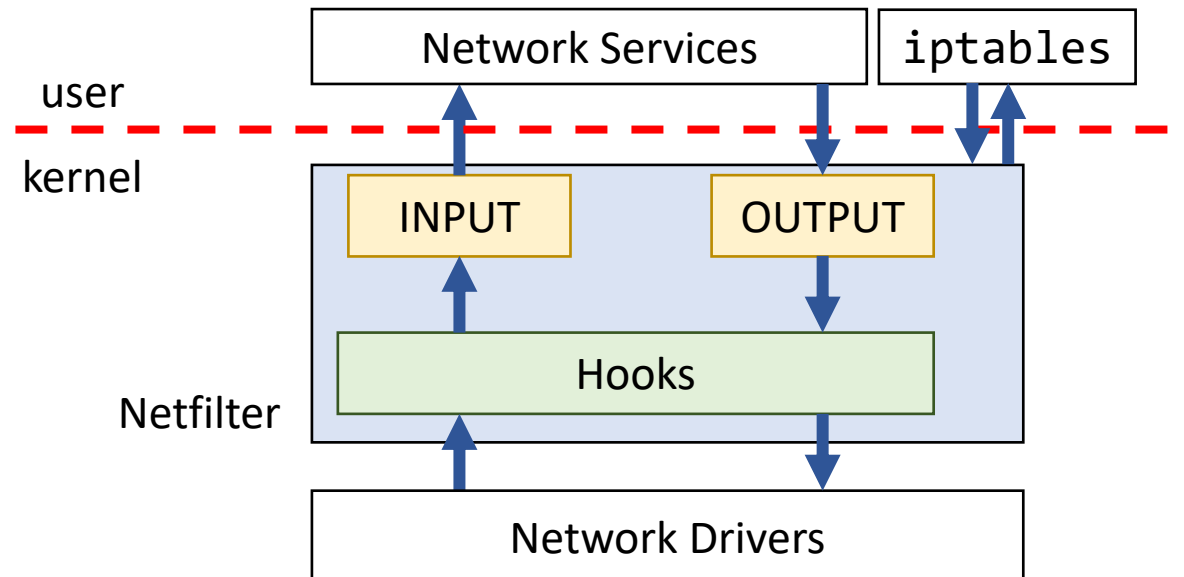
- Register our hook

*telnetFilter*

```
static struct nf_hook_ops telnetFilterHook;  
  
int setUpFilter(void) {  
    telnetFilterHook.hook = telnetFilter;  
    telnetFilterHook.hooknum = NF_INET_POST_ROUTING;  
    telnetFilterHook.pf = PF_INET;  
    telnetFilterHook.priority = NF_IP_PRI_FIRST;  
  
    // Register the hook  
    nf_register_hook(&telnetFilterHook);  
    return 0;  
}
```

# iptables

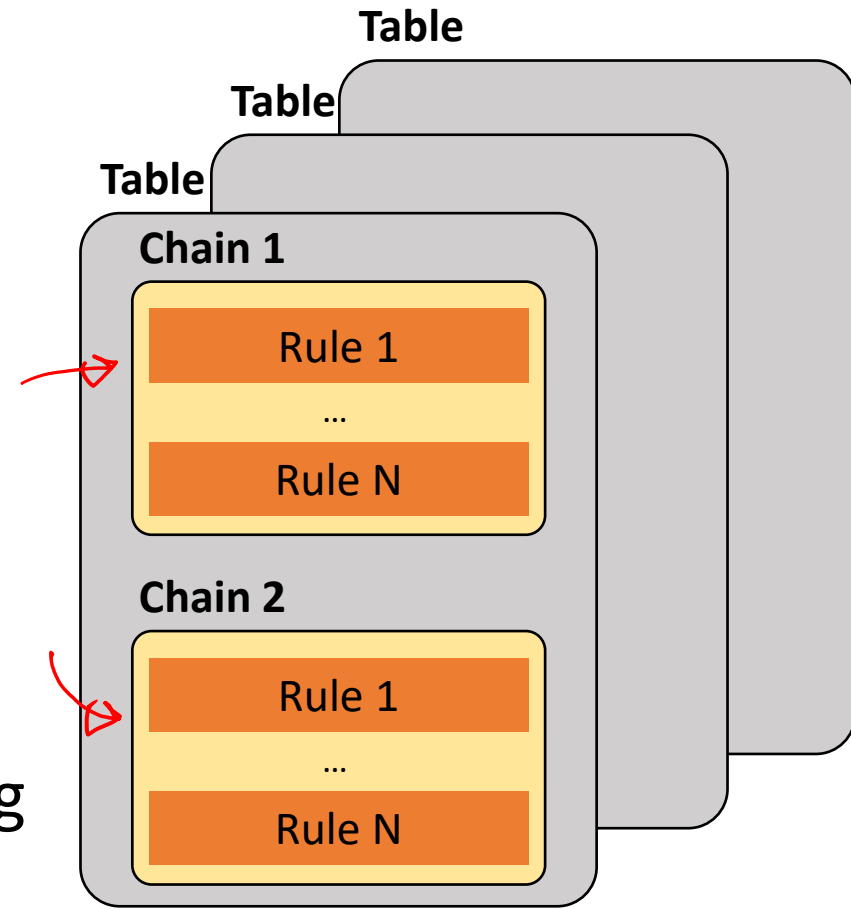
- A packet filter firewall is implemented using iptables
- Userspace program that interfaces with netfilter
- Installs and removes firewall rules
- Can implement *stateless* and *stateful* firewalls





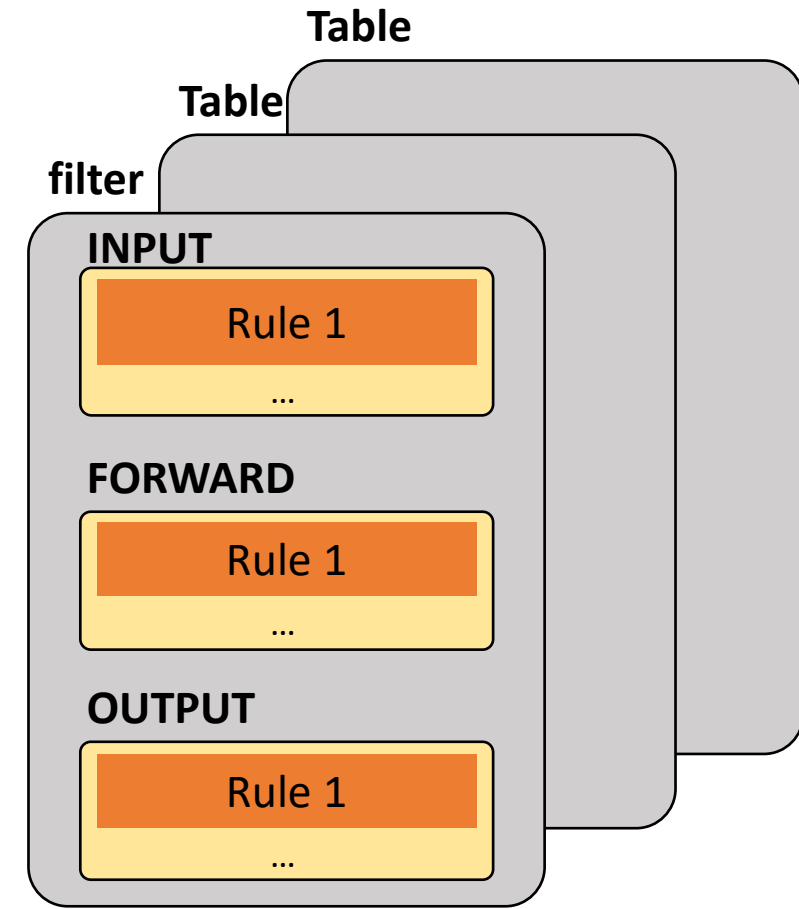
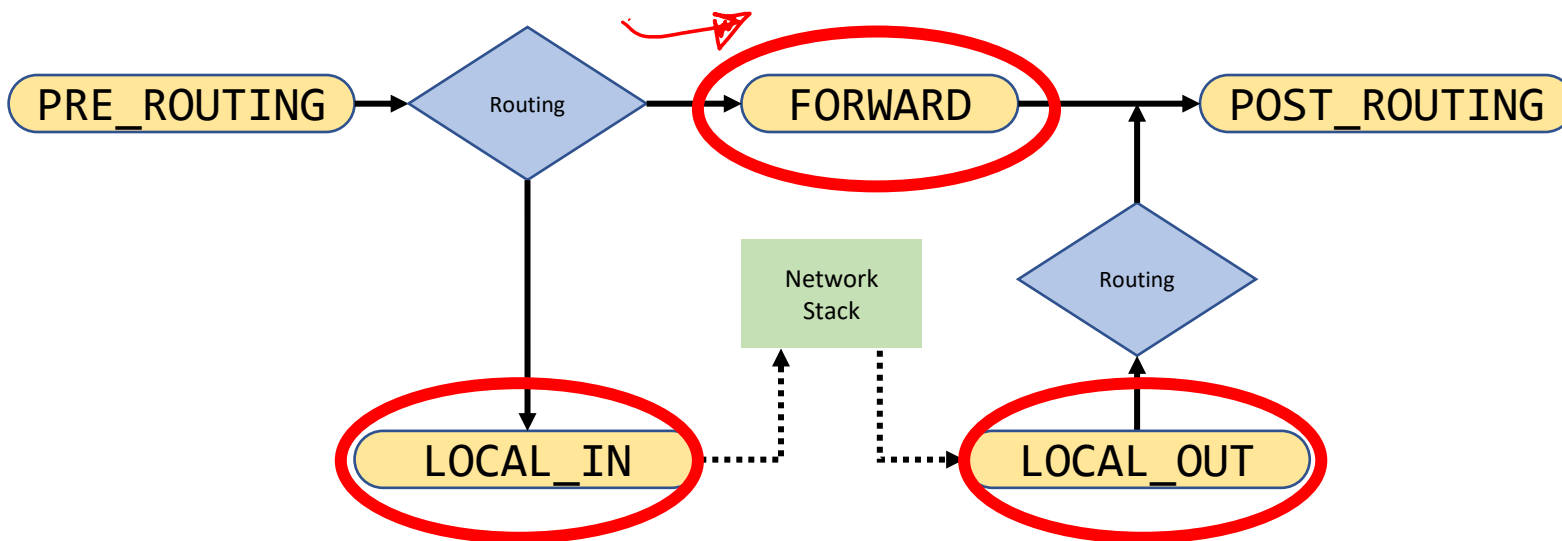
# Rule Organization

- iptables firewall can:
  - filter packets, and
  - make changes to packets.
- Rules are organized in a hierarchical structure
  - Table
  - Chain
  - Rule
- A chain reflects **when** a rule is evaluated during the packet life cycle



# Rule Organization

- The table used for firewalls is the **filter** table
- **filter** table has three chains:
  - INPUT: incoming packets
  - FORWARD: packets routed through this machine
  - OUTPUT: outgoing packets



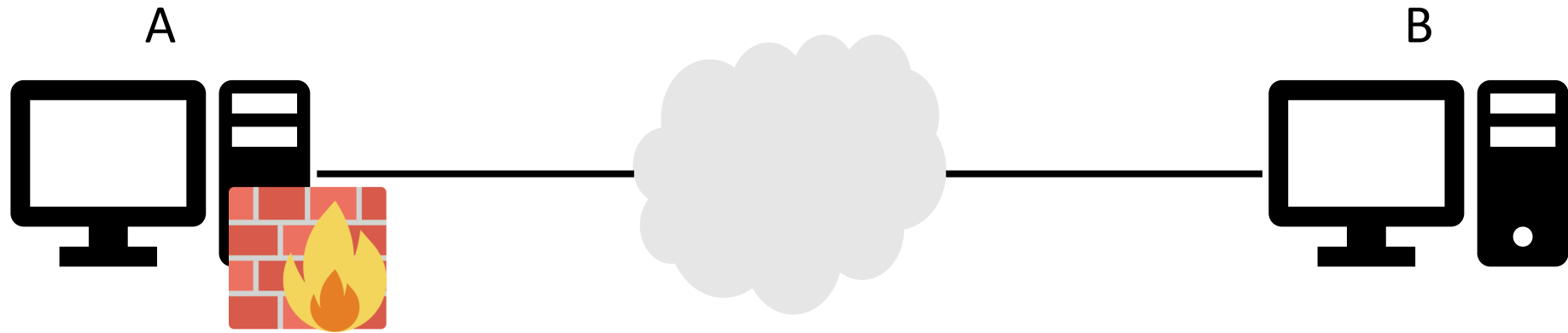
# Targets

---

- A target is the action that are triggered when a packet meets the matching criteria of a rule.
- Terminating targets: Stops the evaluation within a chain. E.g.,:
  - ➔ • ACCEPT
- Non-Terminating targets: Performs an action and continues the evaluation within a chain. E.g.,:
  - Jumping to user-defined chains

# Demo

---



We will run iptables  
at machine A

# Checking Rules

```
$ sudo iptables -L
```

*filter*

```
Chain INPUT (policy ACCEPT)
target prot opt source destination
```

```
Chain FORWARD (policy ACCEPT)
target prot opt source destination
```

```
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

No rules yet!

```
$ sudo iptables -t filter -F
```

To flush **filter** table

# Demo 1

---

```
$ sudo iptables ⏏ -A INPUT -p icmp --icmp-type echo-request -j DROP
```

Dropping all incoming ICMP echo requests

→ No one can ping machine A

# Demo 2

---

```
$ sudo iptables -A INPUT -p tcp --destination-port 22 -j ACCEPT  
$ sudo iptables -A INPUT -j REJECT
```

Allow others to ssh to machine A

AND

Machine A does not respond to other service request

- What if we switch the rule order?

# Demo 2

---

```
$ sudo iptables -L
```

```
Chain INPUT (policy ACCEPT)
```

```
target prot opt source destination
```

```
ACCEPT tcp -- anywhere anywhere tcp dpt:ssh
```

```
REJECT 0 -- anywhere anywhere reject-with icmp-port-unreachable
```

```
Chain FORWARD (policy ACCEPT)
```

```
target prot opt source destination
```

```
Chain OUTPUT (policy ACCEPT)
```

```
target prot opt source destination
```



# Demo 2

---

```
$ sudo iptables -A INPUT -p tcp --destination-port 22 -j ACCEPT  
$ sudo iptables -A INPUT -j REJECT
```

Allow others to ssh to machine A

AND

Machine A does not respond to other service request

- What if we switch the rule order?

# Demo 2 Takeaways

---

- REJECT

- The port is closed



- DROP

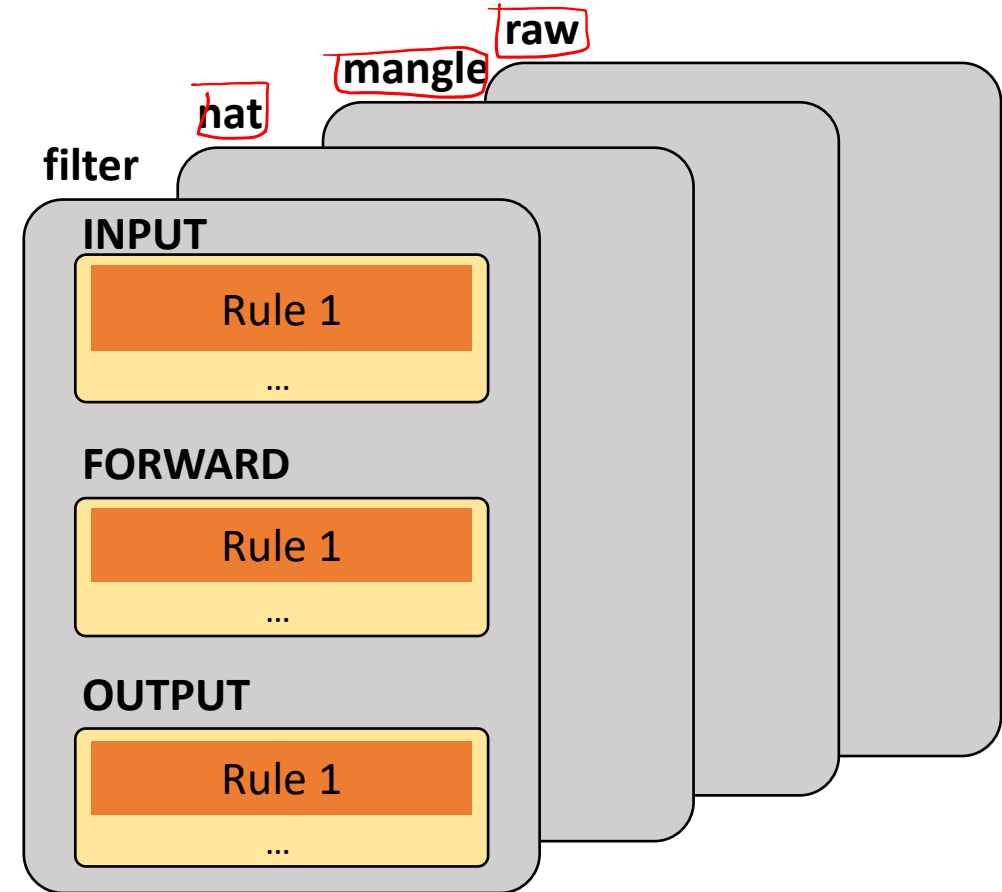
- The port is closed and invisible to the network

- Rule order is important (within a chain)

- Rules are evaluated top-down

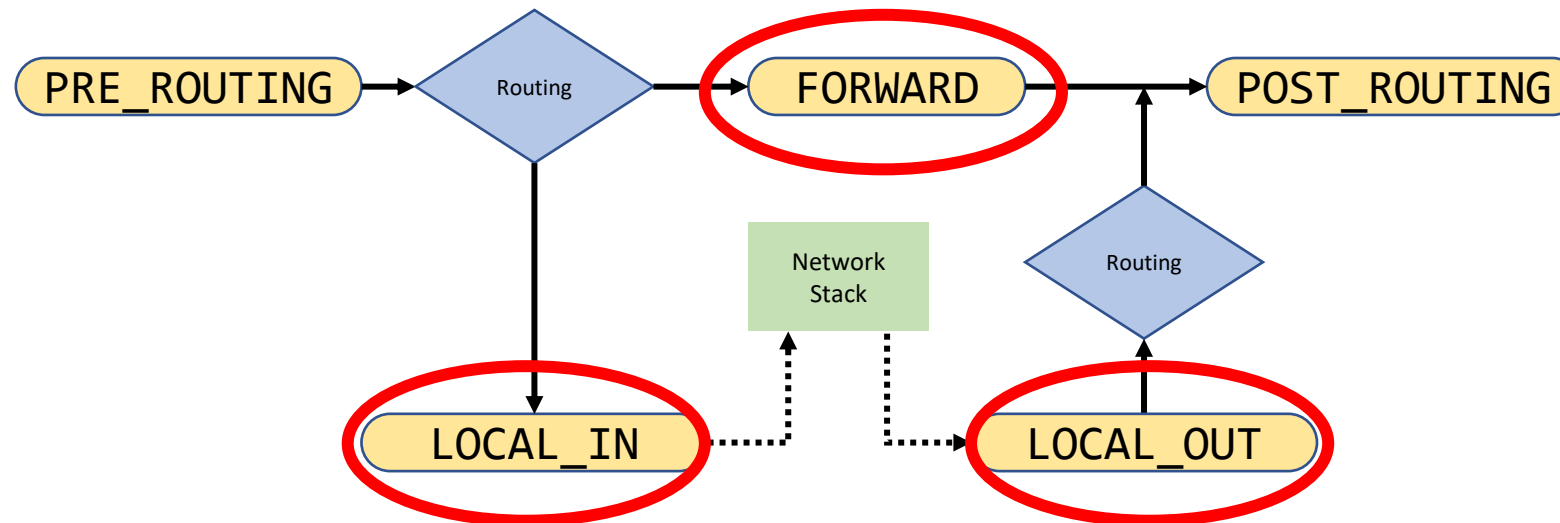
# Tables

- iptables uses **four** tables to organize its rules
  - filter, nat, mangle, raw
- These tables classify rules according to the type of decisions they are used to make
- It is important to know which chains are implemented in each table



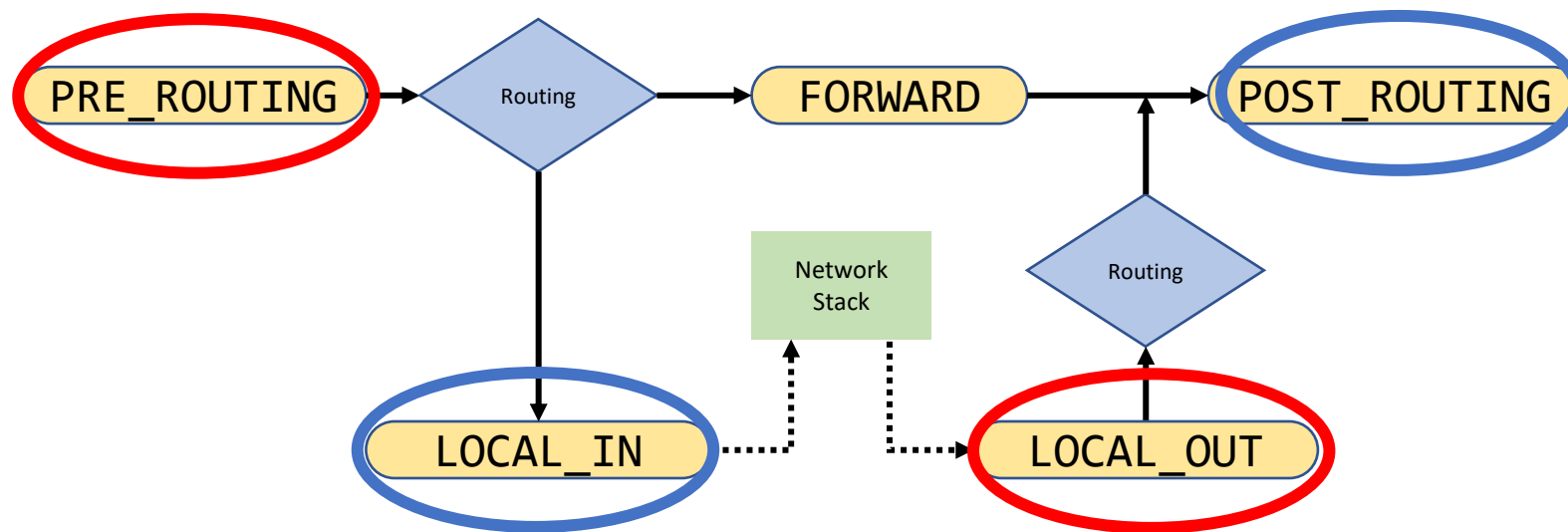
# The filter Table

- Most widely used to implement firewalls
- Decides whether to accept the packet or not
- Implements three chains



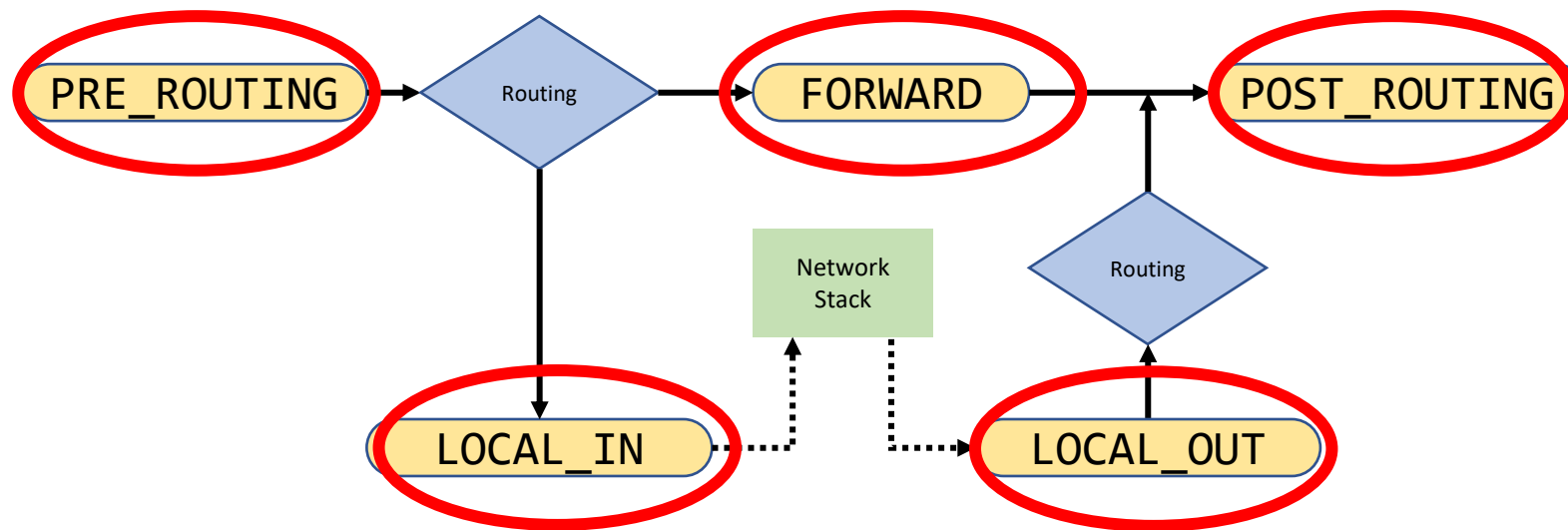
# The nat Table

- Determines whether and how to modify the source or destination addresses
  - to impact the way that the packet and any response traffic are routed
- **Destination NAT:**
  - modify the dst address/port (for incoming packets to the private network)
- **Source NAT:**
  - modify the src address/port (for outgoing packets from the private network)



# The mangle Table

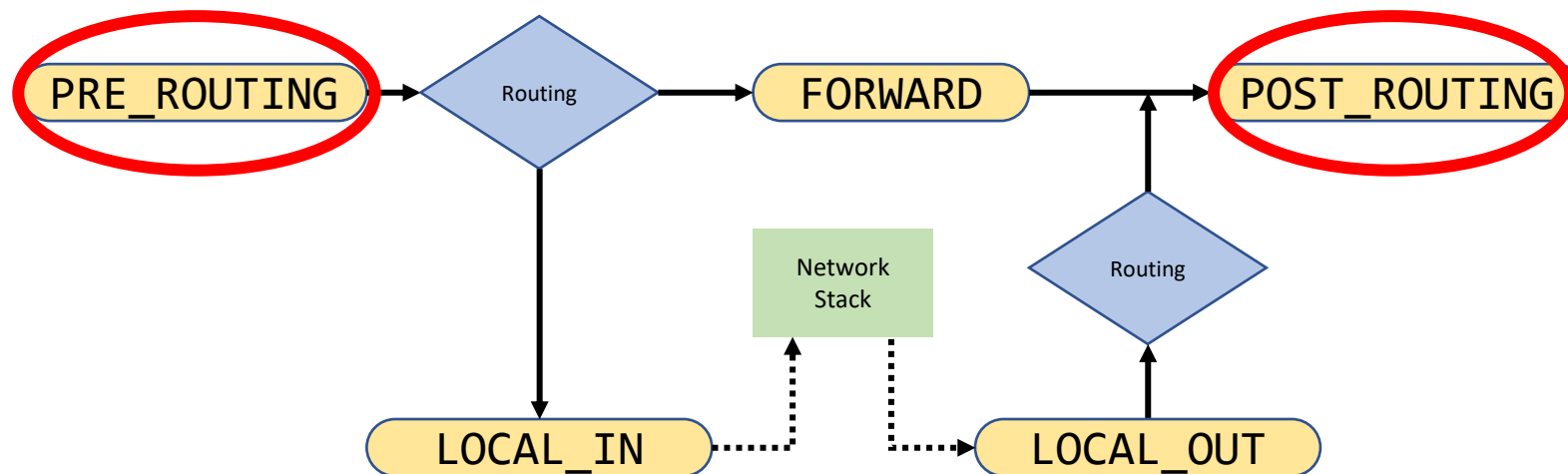
- Used to alter the IP header
  - E.g., TTL value
- Also, to enable marking the packets
  - Other network tools or tables may read this mark to process the packet differently
  - Internal to the kernel (i.e., marking doesn't modify the actual packet)



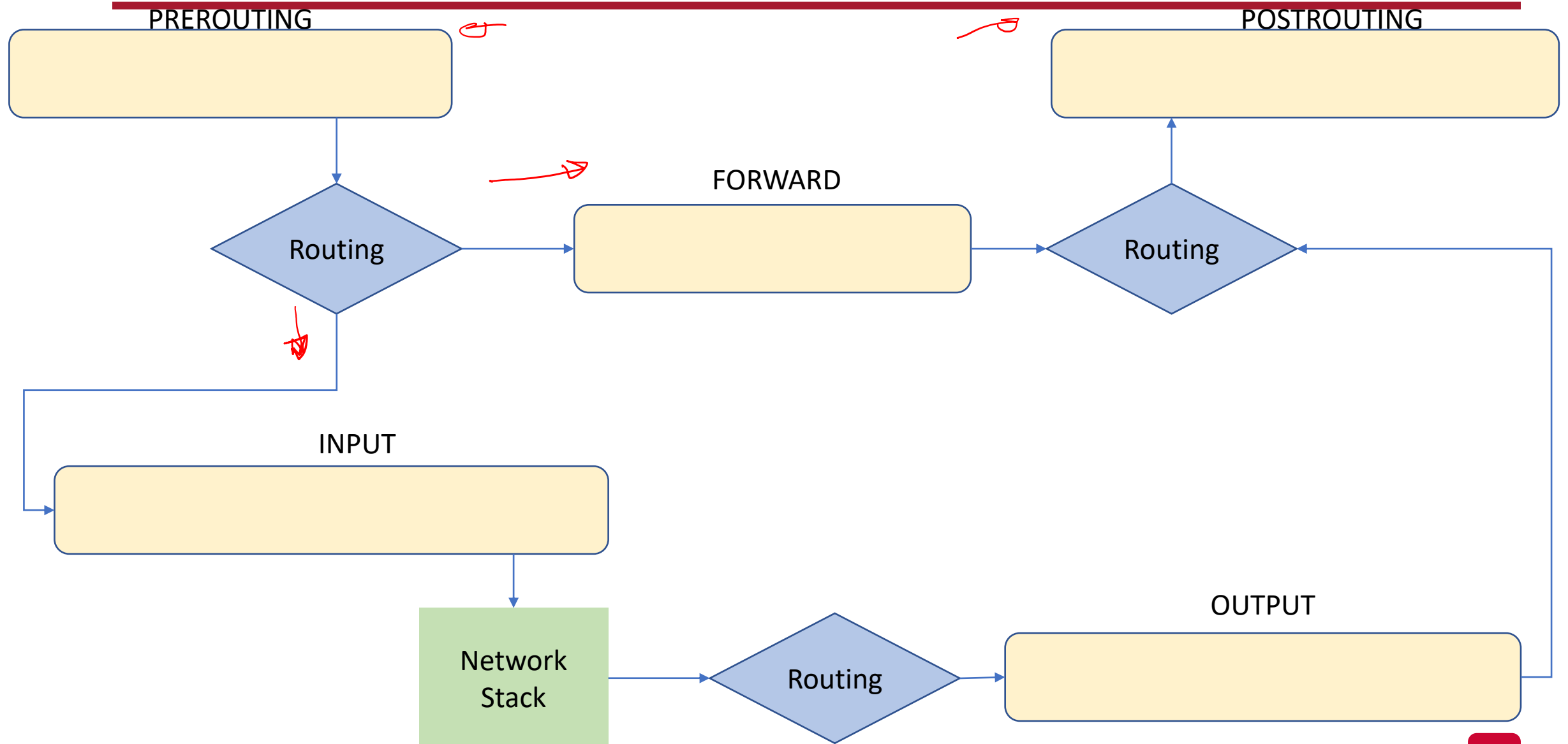
# The raw Table

---

- Used to disable stateful firewall for some packets
- Set the mark called NOTRACK

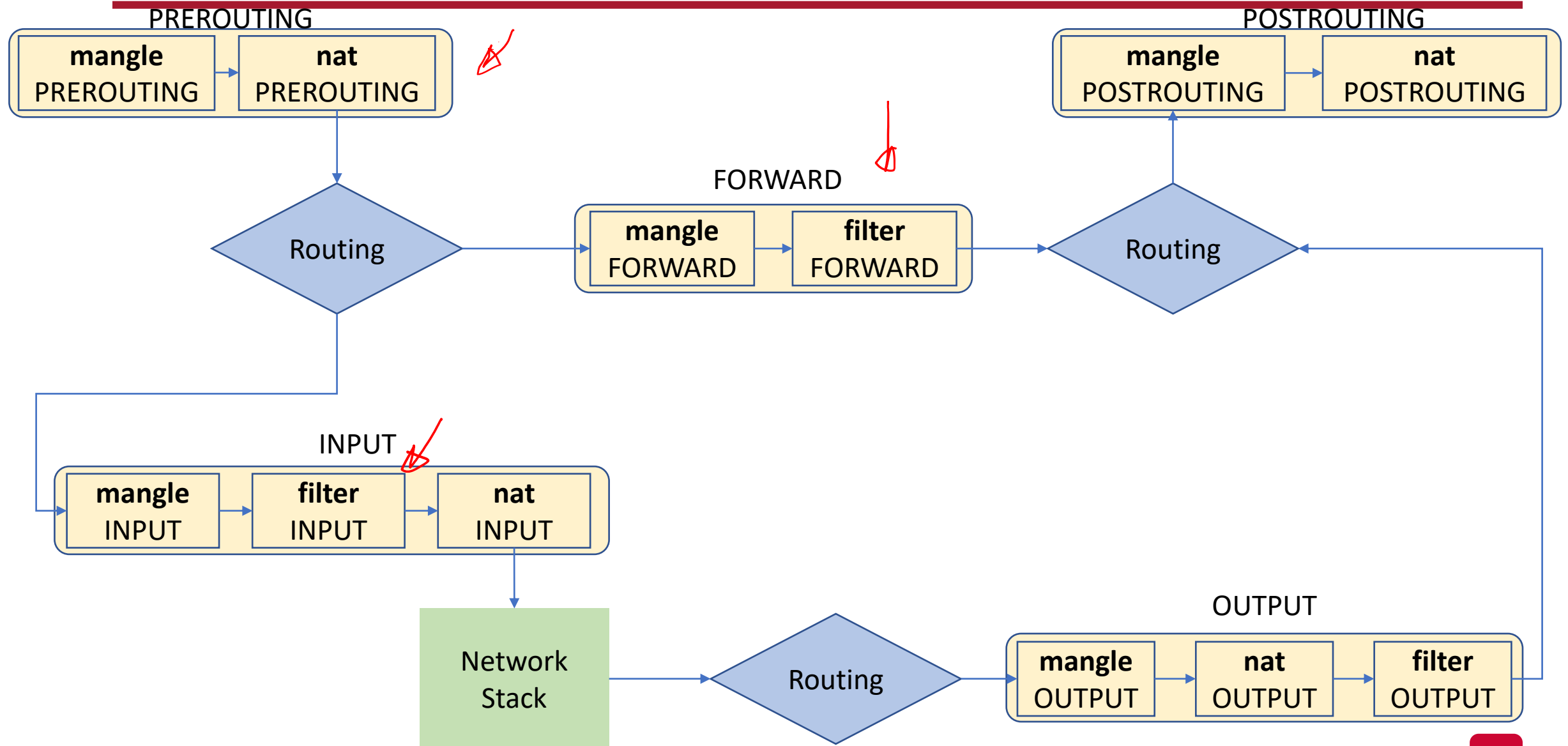


# Table/Chain Traversal Order





# Table/Chain Traversal Order



# Example: Determine Table and Chain

---

- To increase the TTL for all packets
  - Packet modification → mangle table
  - All packets → PREROUTING chain



```
$ sudo iptables -t mangle -A PREROUTING -j TTL --ttl-inc 5
```

# Extensions

---

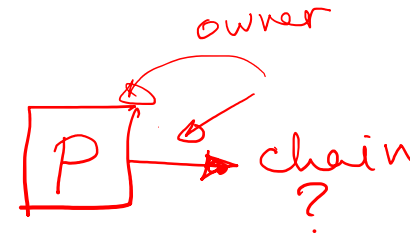
- Adding more functionalities to the core of iptables
  - Installing kernel modules
  - E.g., conntrack, owner, cgroup, cpu, etc.

```
$ ls /lib/modules/`uname -r`/kernel/net/netfilter/
```

```
nf_conntrack_snmp.ko  nfnetlink_cttimeout.ko  nft_fib.ko  
nft_reject_inet.ko  xt_comment.ko          xt_esp.ko  
xt_LOG.ko          xt_quota.ko  ...
```

# Extensions: Examples

- Disable telnet for a specific user
- Using the owner extension
  - Available at OUTPUT chain only



-p tcp -----

```
$ sudo iptables -A OUTPUT -m owner --uid-owner 1000 -j DROP
```

# Extensions: Examples

- Redirecting packets based on the handling CPU number
- Using the cpu extension





0 1 2 3 ...

```
$ iptables -t nat -A PREROUTING -p tcp --dport 80 -m cpu --cpu 0  
-j REDIRECT --to-port 8080
```

Port forwarding

# Building a Simple Firewall

---

- Requirements
  - Allow SSH, HTTP, and ICMP
  - Allow loopback interface 
  - Allow DNS 
  - Allow VPN and HTTPs 
  - Allow all outgoing traffic 
- What is missing?
- Let's call it sFW

# Our sFW: R1

---

Allow SSH, HTTP, and ICMP

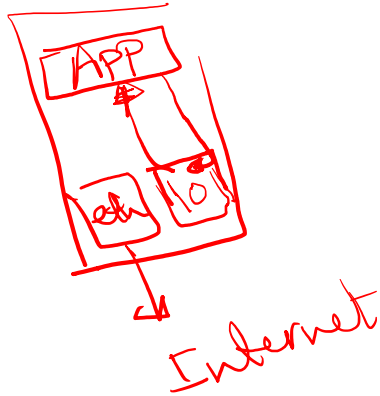
```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT}
iptables -A INPUT -p tcp --dport 80 -j ACCEPT}
iptables -A INPUT -p icmp --icmp-type any -j ACCEPT}
```

# Our sFW: R2

---

Allow loopback interface

```
iptables -A INPUT -p all -i lo -j ACCEPT
```





# Our sFW: R3

---

Allow DNS



```
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp --sport 53 -j ACCEPT
iptables -A INPUT -p udp --sport 53 -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT
```

# Our sFW: R4

---

Allow VPN and HTTPs

*ESP  
AH*

```
iptables -A INPUT -p 50 -j ACCEPT
iptables -A INPUT -p 51 -j ACCEPT
iptables -A INPUT -p udp --dport 500 -j ACCEPT
iptables -A INPUT -p udp --dport 10000 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

# Our sFW: R5

---

Allow outgoing traffic


```
iptables -P OUTPUT ACCEPT
```

Drop all other traffic

```
iptables -P INPUT DROP  
iptables -P FORWARD DROP
```

# Stateful Firewalls

---

- Packets are often not independent
  - Part of a TCP connection
  - ICMP packets triggered by other packets 
- Handling such packets independently may lead to inaccurate firewall
  - Our sFW drops incoming TCP packets on ports other than 22, 80 and 443
  - Even when they're part of an established TCP connection

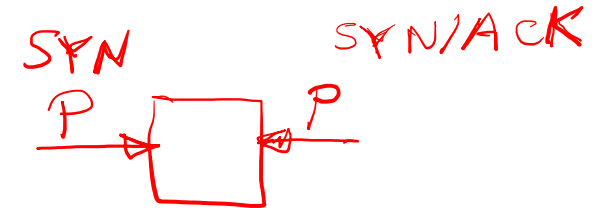
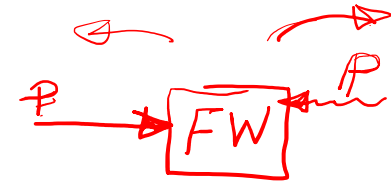
# Stateful Firewalls

---

- They monitor incoming and outgoing packets over a period of time
  - Record connection state
  - Connection state: attributes such as IP addresses, port numbers, sequence number etc.
- When the state is recorded, filtering decisions can be done
  - Note: TCP connection state is not the same as the firewall state
  - Firewall state determines if a packet is part of a flow or not
  - Thus, firewall state is available for both connection-oriented and connection-less protocols

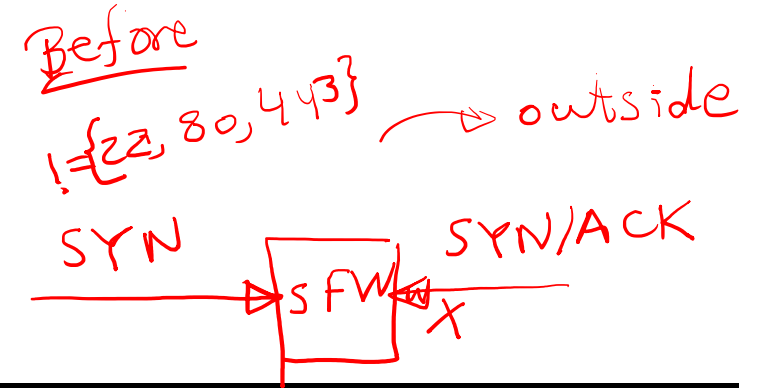
# The Connection Tracking Framework in Linux

- The Linux kernel provides connection tracking framework
  - Called nf\_conntrack
- Each packet is marked with a connection state:
  - NEW:
    - The connection is starting
    - This state exists for a connection if the firewall has only seen traffic in one direction
  - ESTABLISHED:
    - Two-way communication has been observed by the firewall
  - RELATED:
    - A packet that has a relationship with another ESTABLISHED connection
    - E.g., ICMP error messages



# sFW and Connection Tracking

- Let's enable packets that are part of a stream
  - That stream is initiated by our machine



```
iptables -A INPUT -p all -m conntrack --ctstate  
ESTABLISHED,RELATED -j ACCEPT
```

After

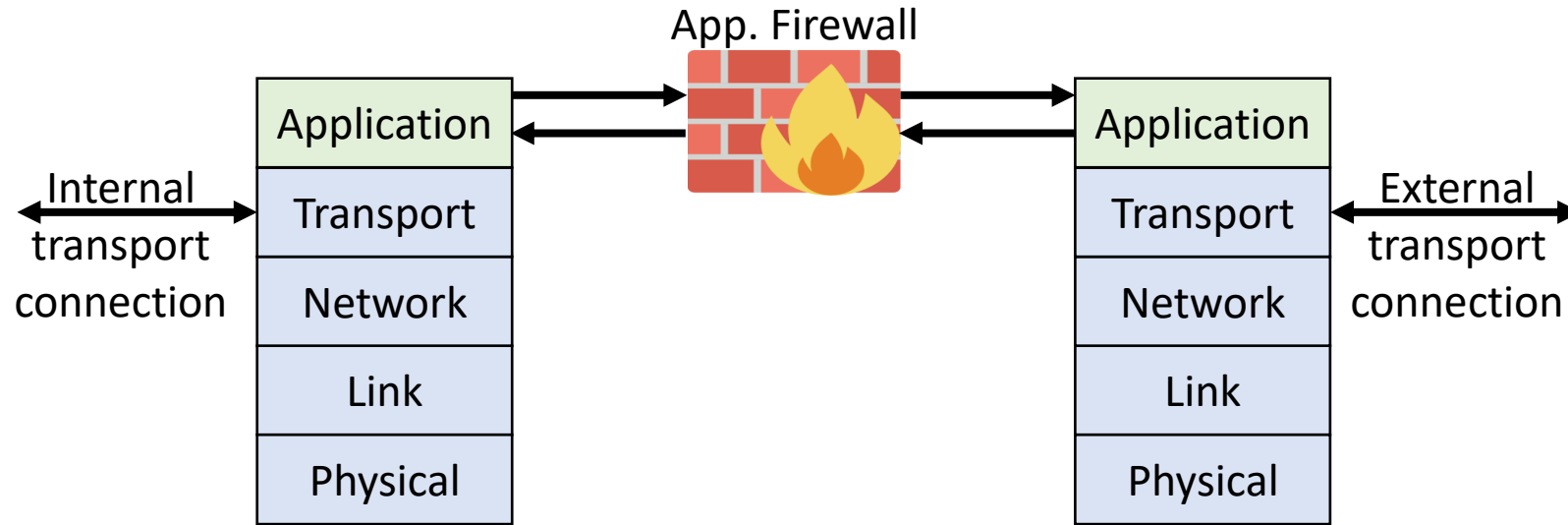


# sFW: Putting it All Together

---

- Requirements
  - Allow SSH, HTTP, and ICMP
  - Allow loopback interface ]
  - Allow DNS
  - Allow VPN and HTTPs
  - Allow all outgoing traffic ]
  - Allow established connections ↺
  - Drop other traffic ]





# Application Firewalls

# Application Firewalls

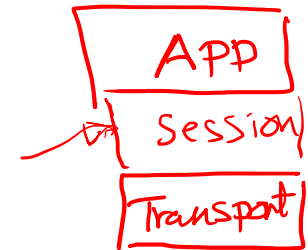
---

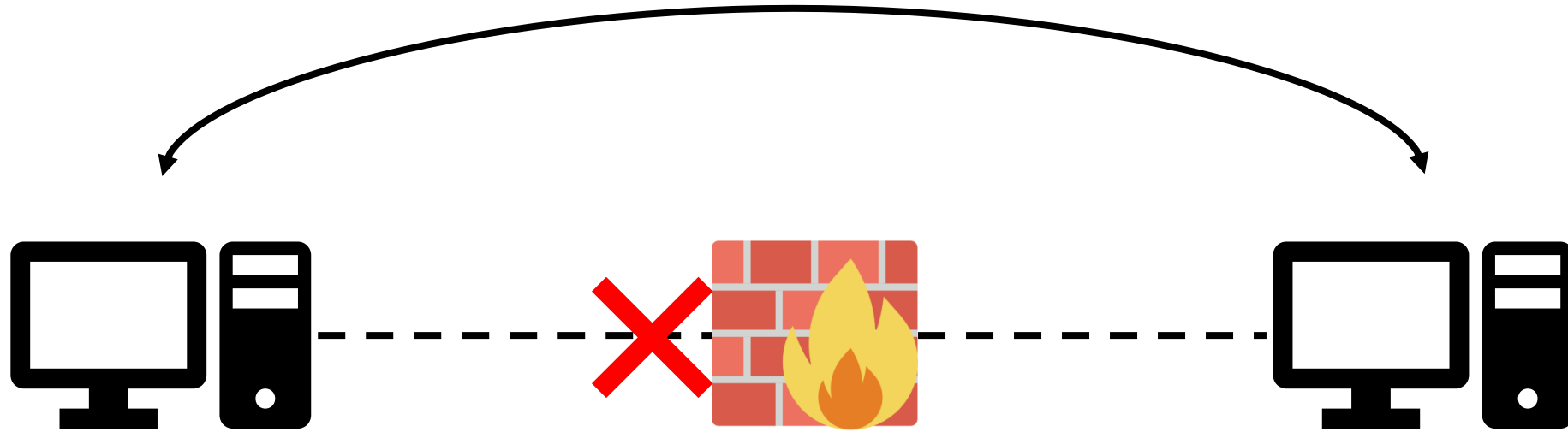
- Inspect traffic up to the application layer
- Examine the requested session for whether it should be allowed/disallowed based on:
  - where the session requests is coming from
  - the purpose of the requested sessions

# Application Firewalls

---

- We need a separate firewall for each different type of service
  - E.g., HTTP, FTP, SMTP
  - Application-dependent
- A more efficient alternative:
  - Using a protocol between the application and transport layers
  - Sometimes called shim layer
  - Application-independent
  - This protocol is called Socket Secure (SOCKS)





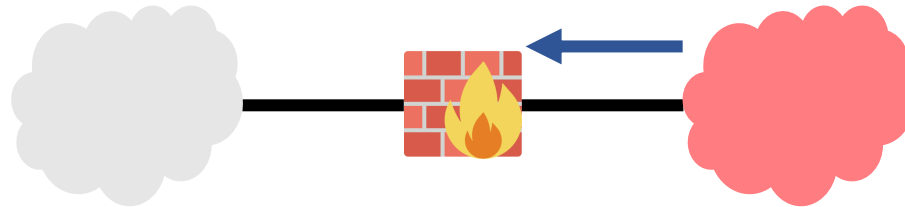
# Evading Firewalls

# Recall: Ingress and Egress Filtering

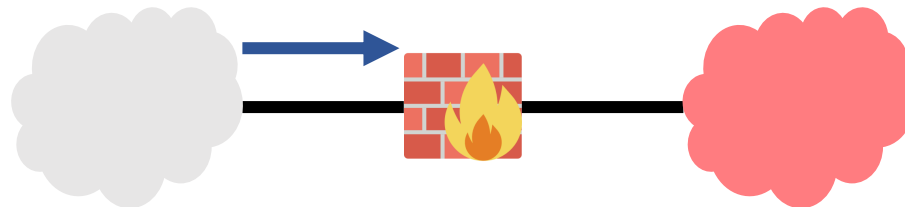
---

- Firewalls can inspect traffic from both directions.

- Ingress filtering

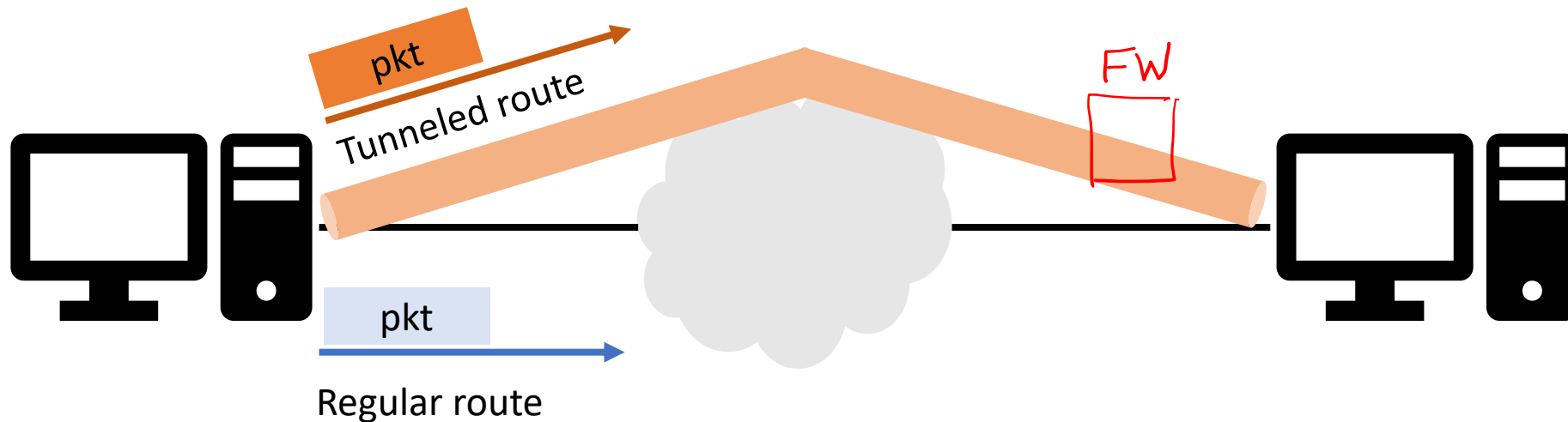


- Egress filtering



# Evading Firewalls: Rationale

- Some firewalls are restrictive
  - E.g., Egress filtering may block users from reaching out to certain websites or services
- *Tunneling* is the main technique to evade firewalls.



- Two tunneling mechanisms: SSH tunnels, and VPN

# SSH Tunneling

---

- SSH protocol:
  - Is used mainly to log in securely to a machine
  - Also supports tunneling and port forwarding
- An SSH tunnel consists of an encrypted link created through SSH protocol
  - Secure file transfers (e.g., FTP over an ssh tunnel)
  - Evading (or bypassing) firewalls

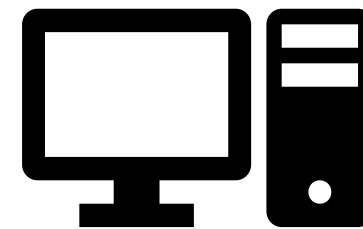
# SSH Tunneling

---

- Users need an access to an SSH server
- SSH tunnels are created using port forwarding
- Port forwarding includes techniques to:
  - translate the destination address and/or port to a new one
  - forward these packets according to the routing table



machine A



ssh server



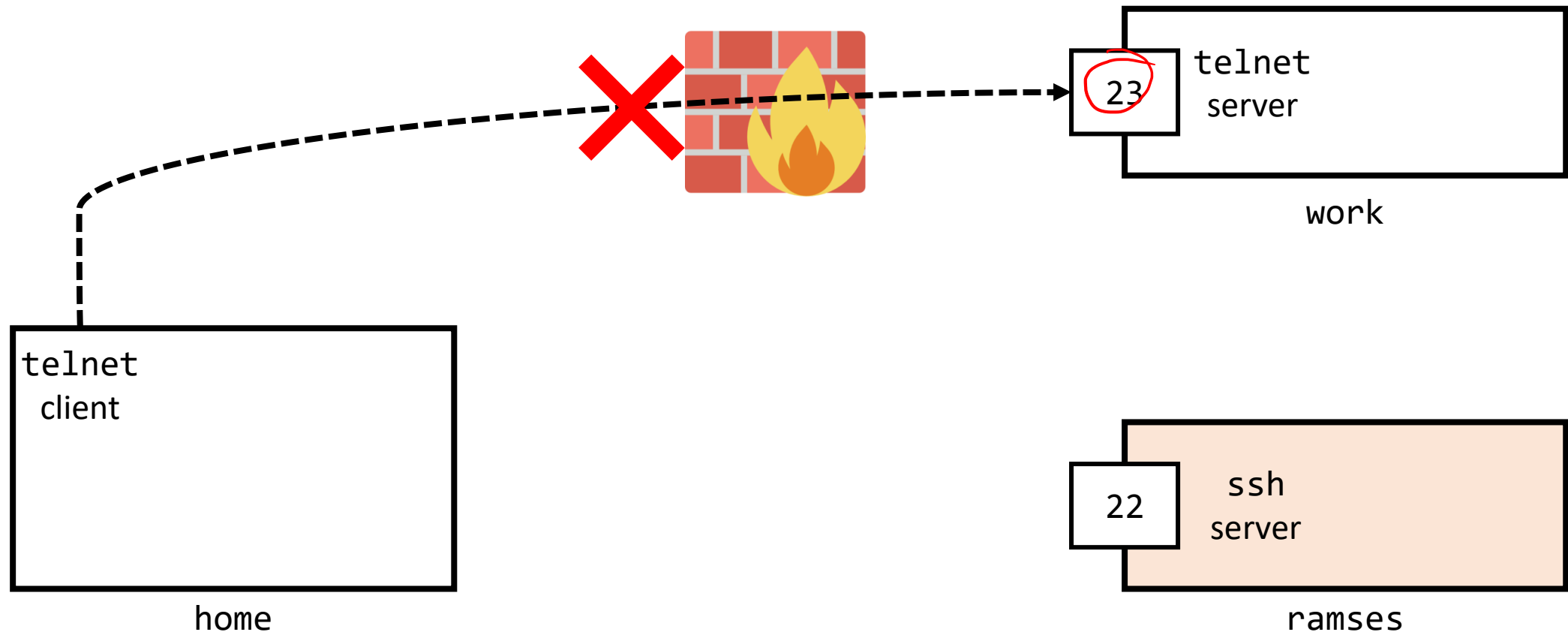
# SSH Tunneling

---

- Two techniques:
  - Tunneling using local port forwarding:
    - the local host performs forwarding
  - Reverse tunneling using remote port forwarding:
    - a remote host performs forwarding

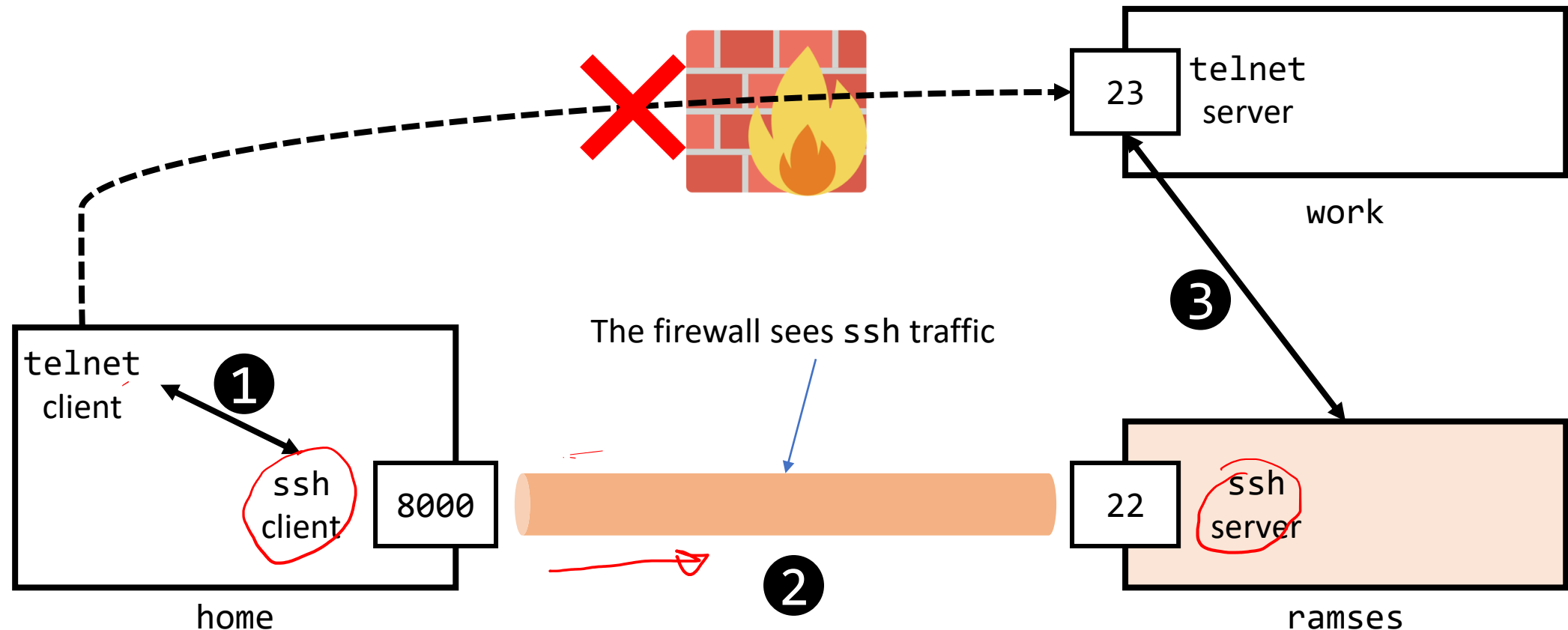
# Local Port Forwarding: Evading Ingress Filtering

- telnet traffic from home → work is blocked by the firewall



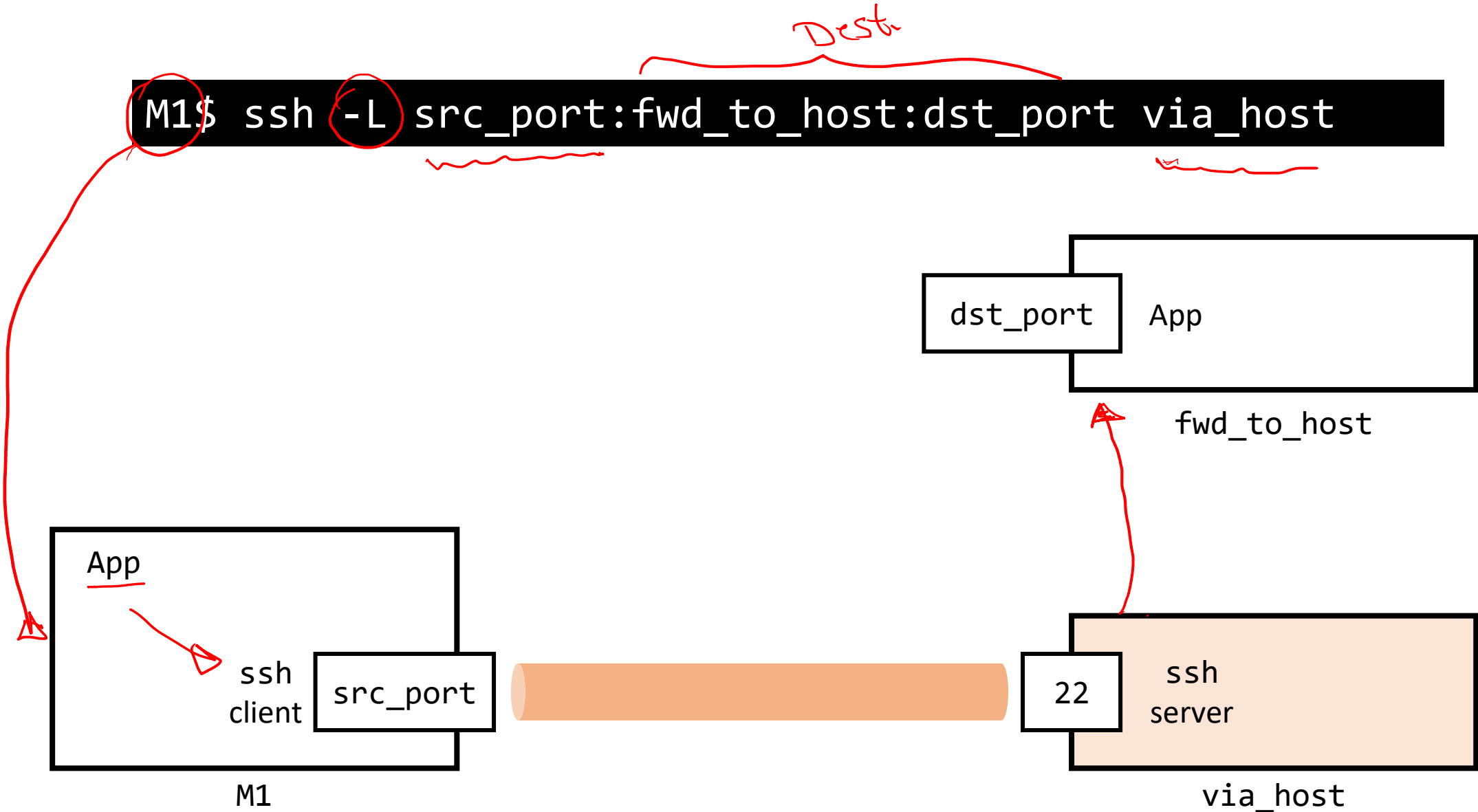
# Local Port Forwarding: Evading Ingress Filtering

- We establish an ssh tunnel: home  $\leftrightarrow$  ramses
  1. On home endpoint, the tunnel receives TCP packets from telnet client
  2. The tunnel forwards TCP packets to ramses endpoint
  3. At ramses, the data is put in another TCP packets and sent to work



# Creating an SSH Tunnel

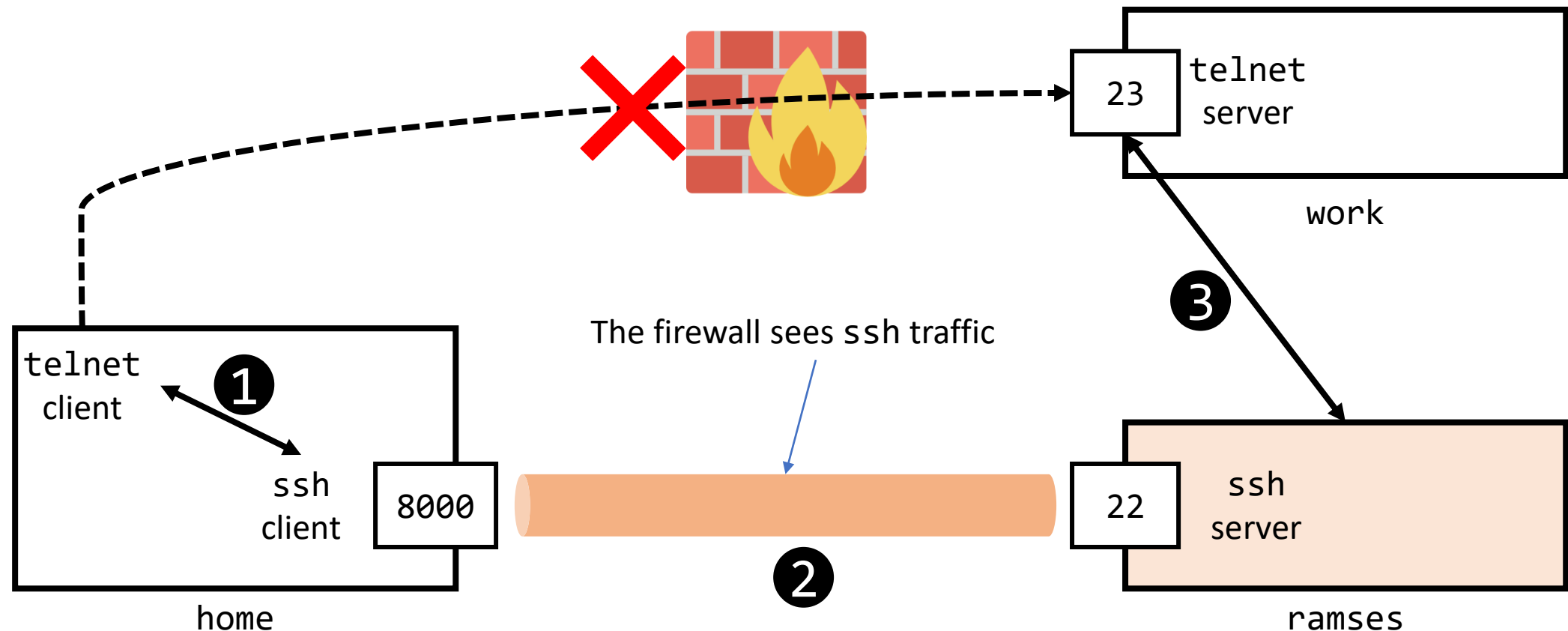
```
M1$ ssh -L src_port:fwd_to_host:dst_port via_host
```



# Local Port Forwarding: Evading Ingress Filtering

- Create an ssh tunnel:

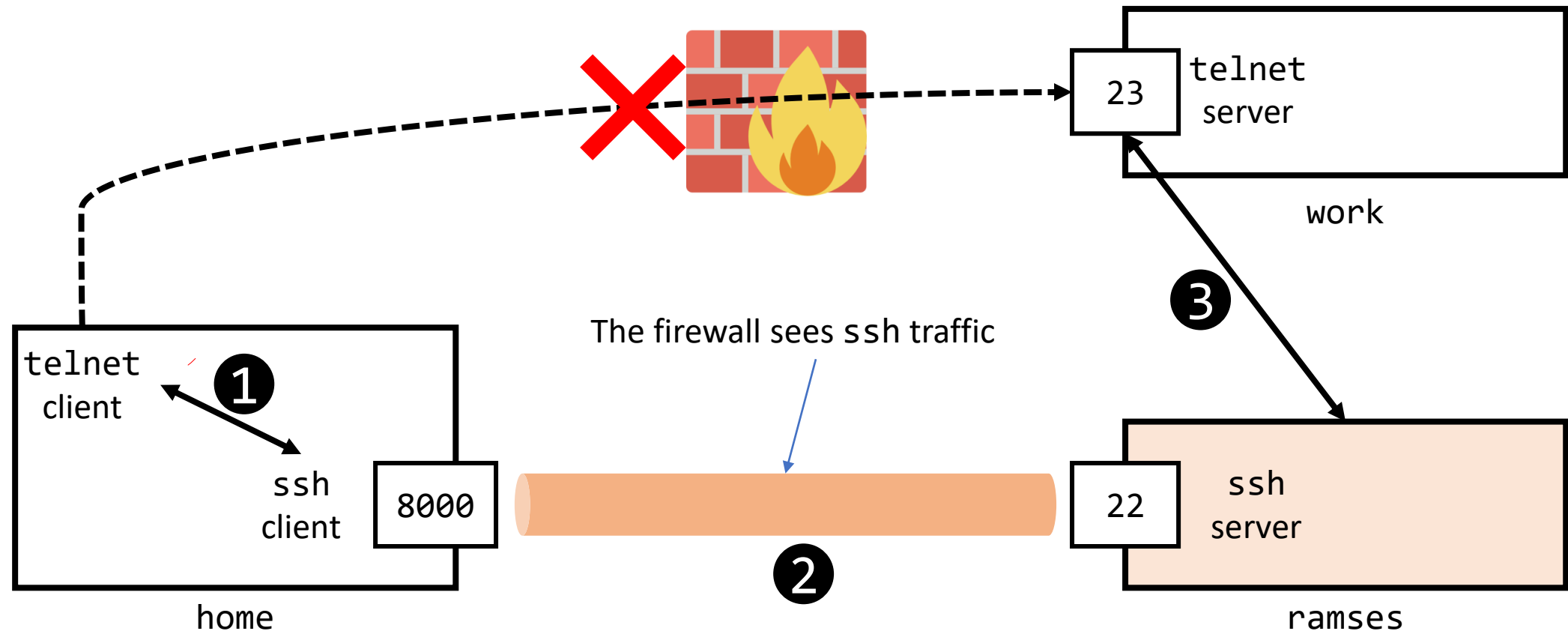
Who performs port forwarding      Final destination  
`home$ ssh -L 8000:work:23 user@ramses` *via host*



# Local Port Forwarding: Evading Ingress Filtering

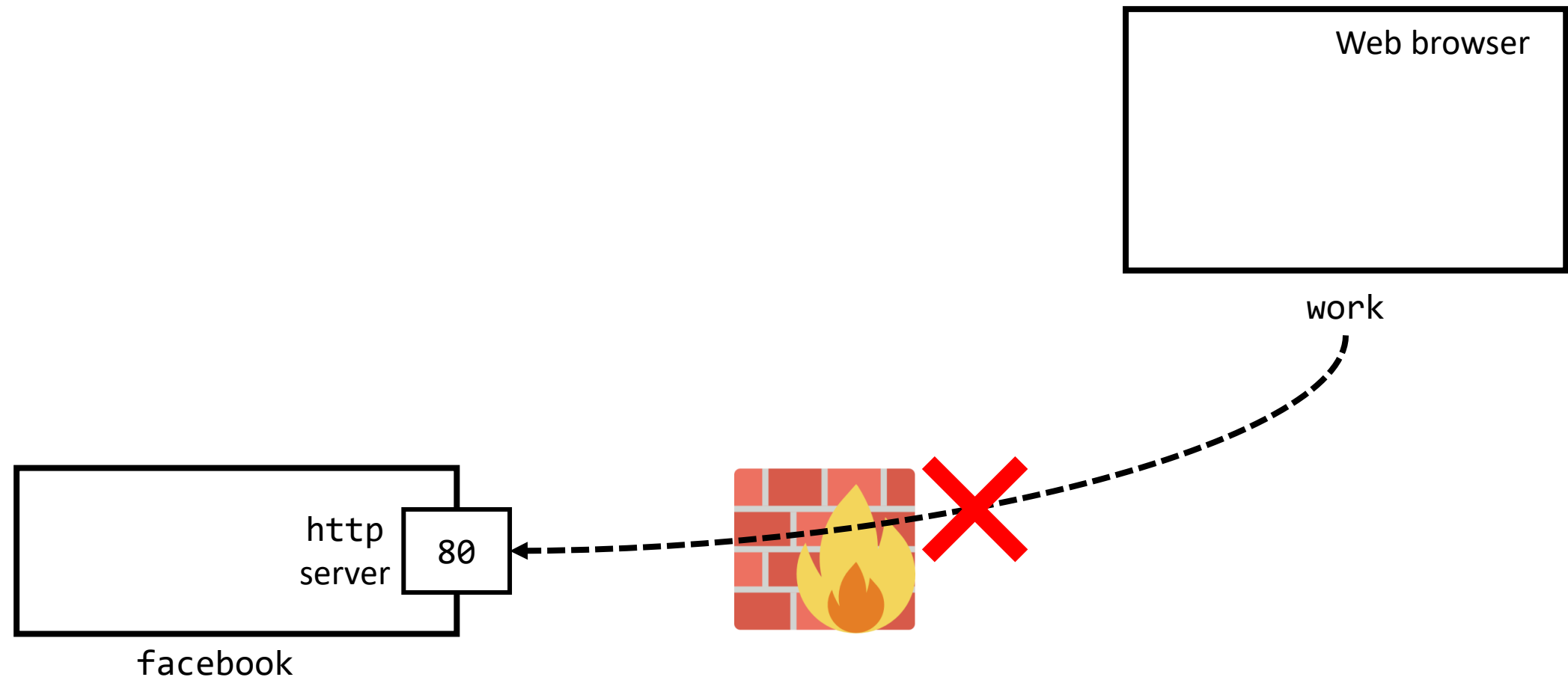
- Starting a telnet session at home:

```
home$ telnet localhost:8000
```



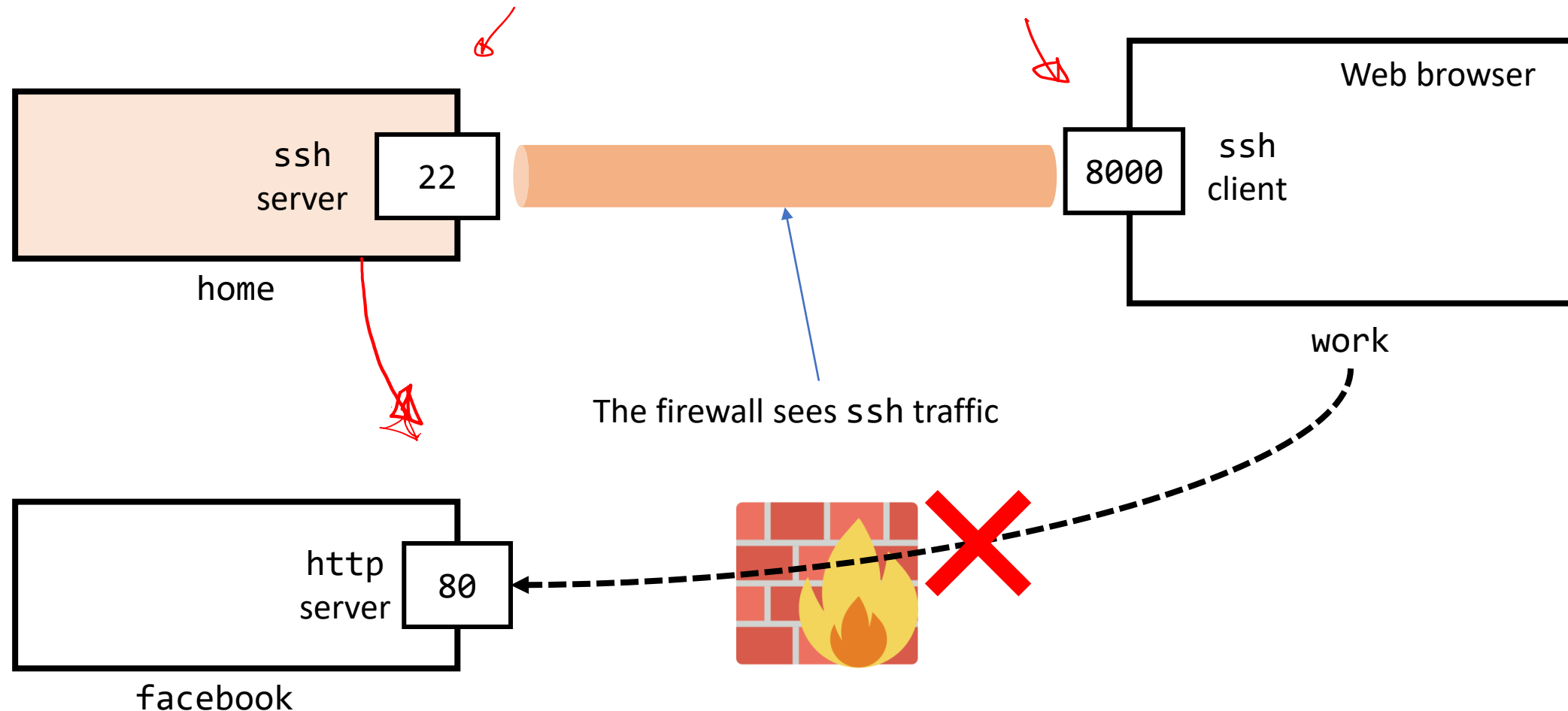
# Local Port Forwarding: Evading Egress Filtering

- Some Internet services may be blocked to users



# Local Port Forwarding: Evading Egress Filtering

- We establish an ssh tunnel: work  $\leftrightarrow$  home to access an Internet service

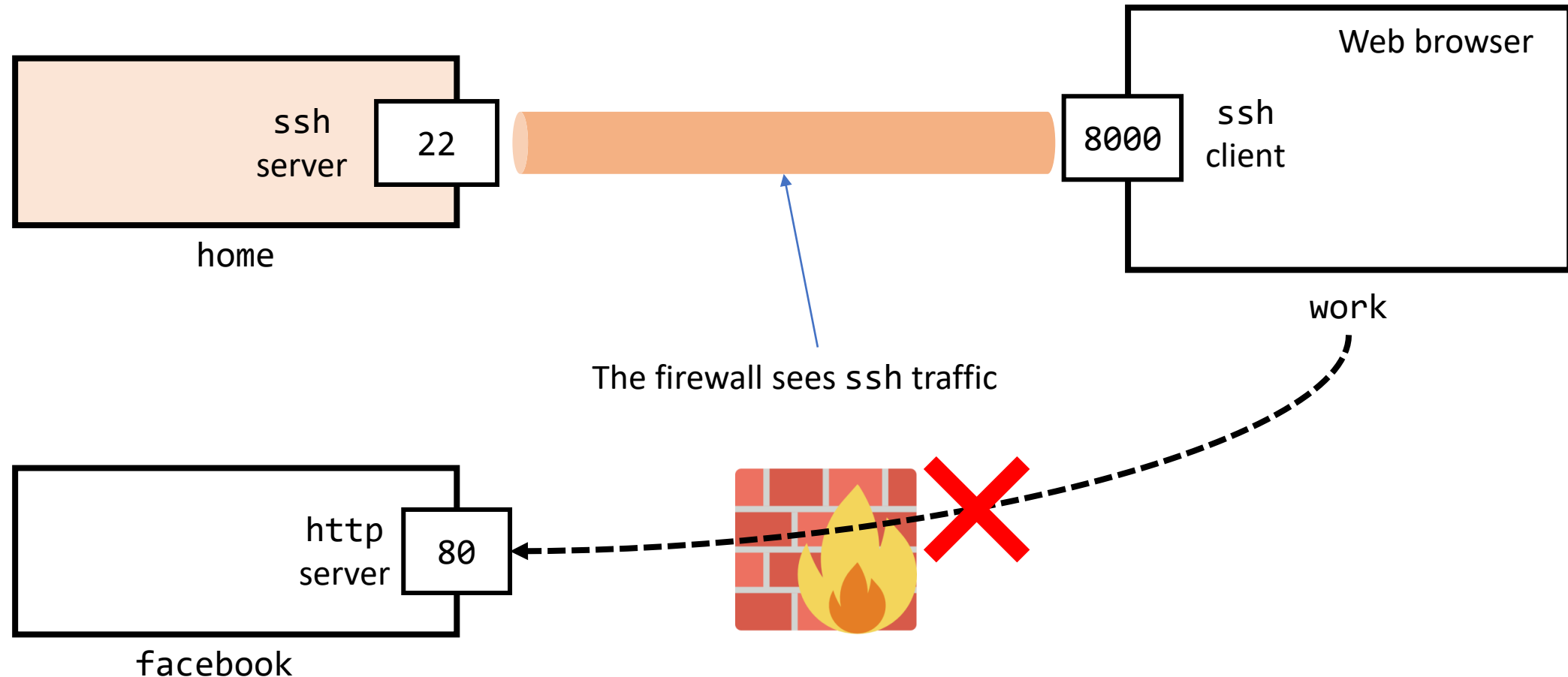




# Local Port Forwarding: Evading Egress Filtering

- Create an ssh tunnel:

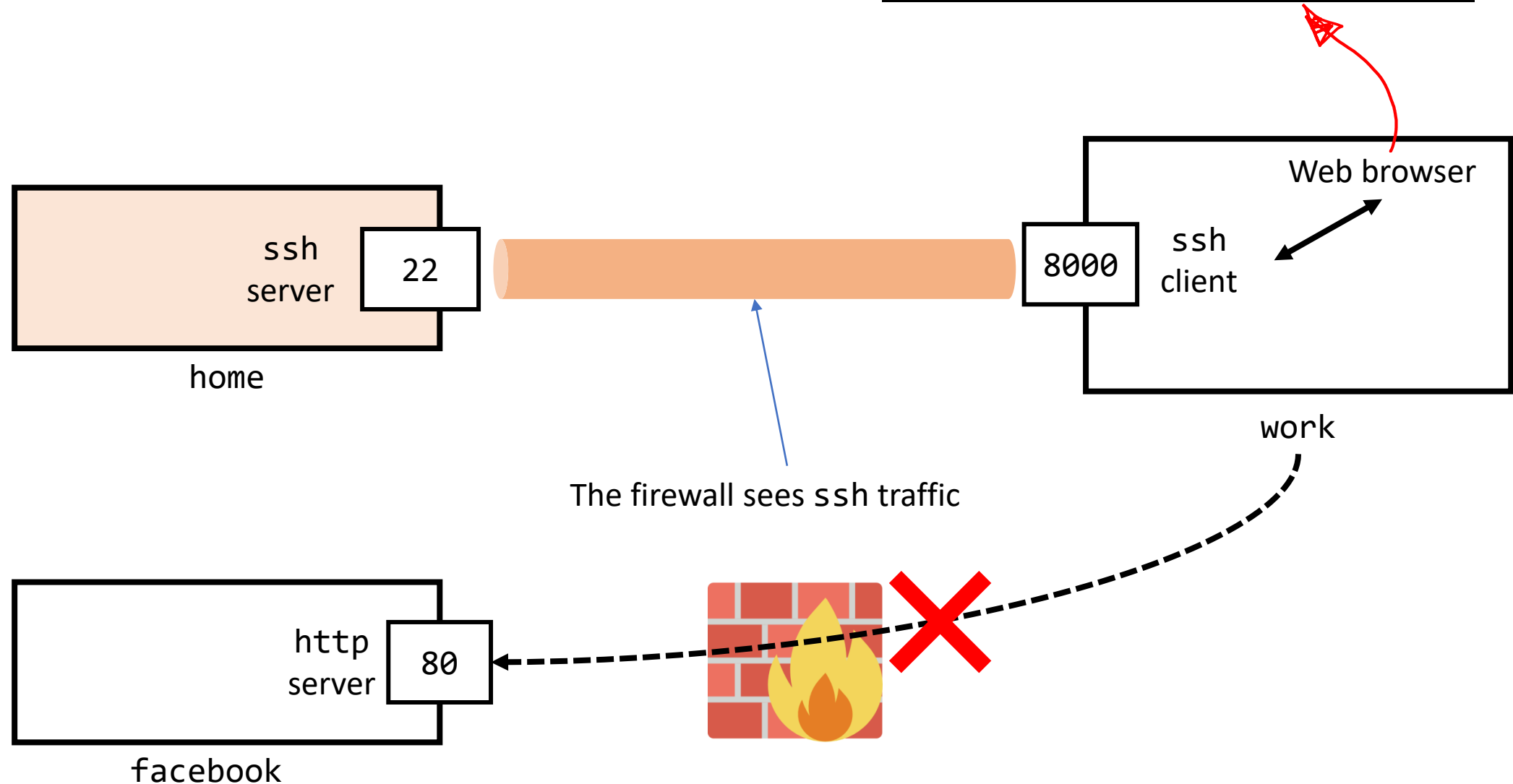
```
work$ ssh -L 8000:facebook.com:80 user@home
```



# Local Port Forwarding: Evading Egress Filtering

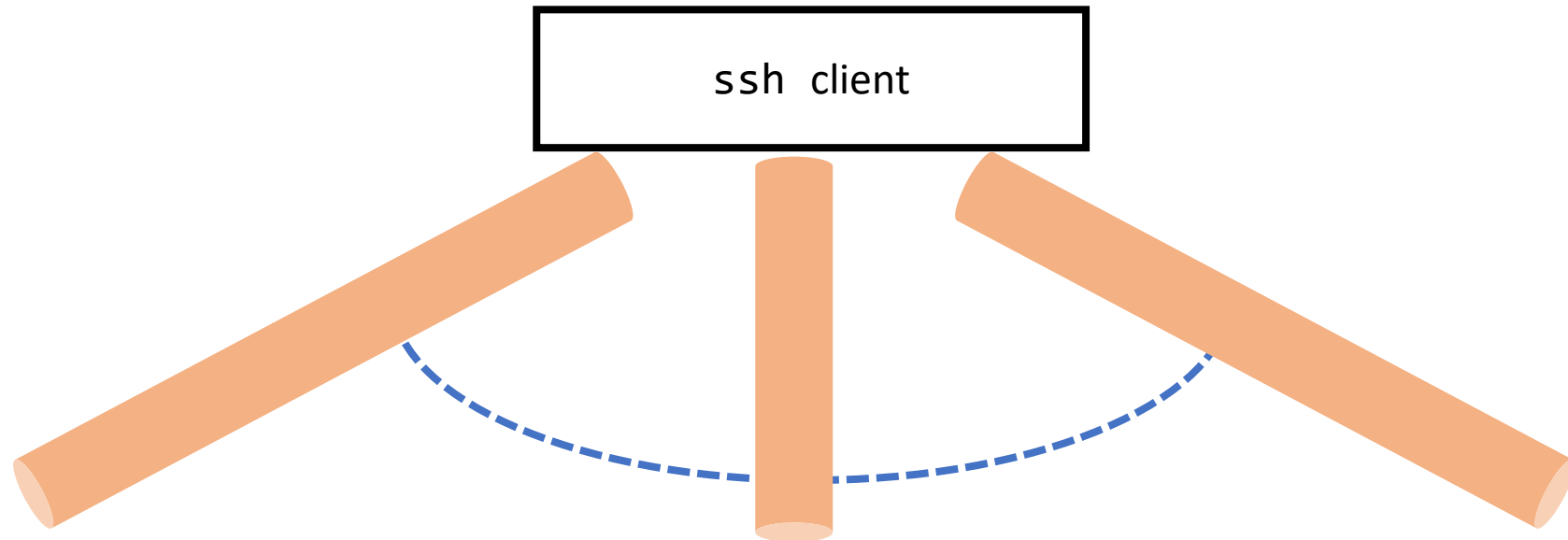
- Visit the website (from the browser)

localhost:8000



# Local Port Forwarding: Dynamic Port Forwarding

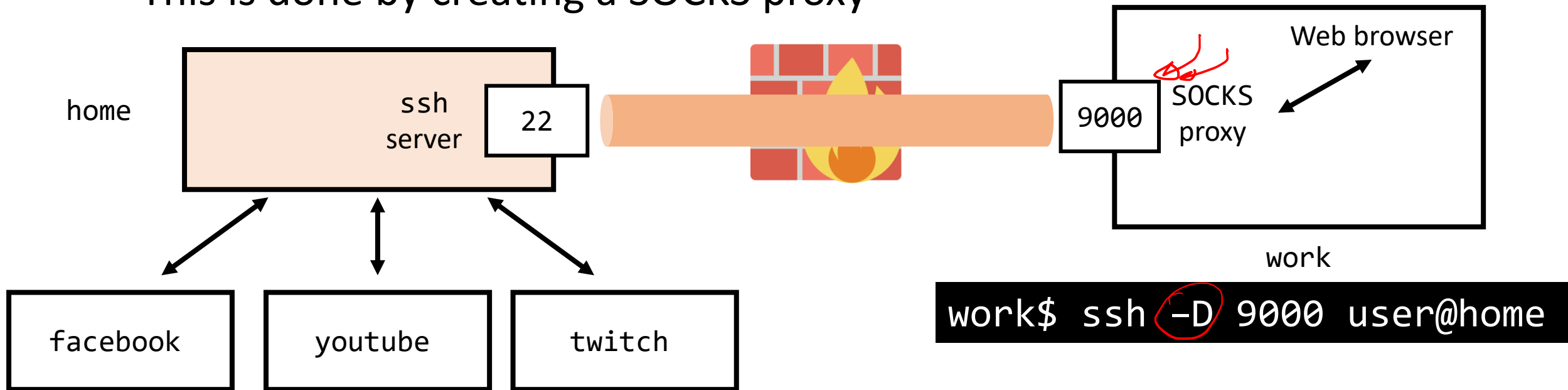
- Previous techniques use static port forwarding
- What happens if the firewall blocks many services?



Creating/maintaining individual tunnels is complex

# Local Port Forwarding: Dynamic Port Forwarding

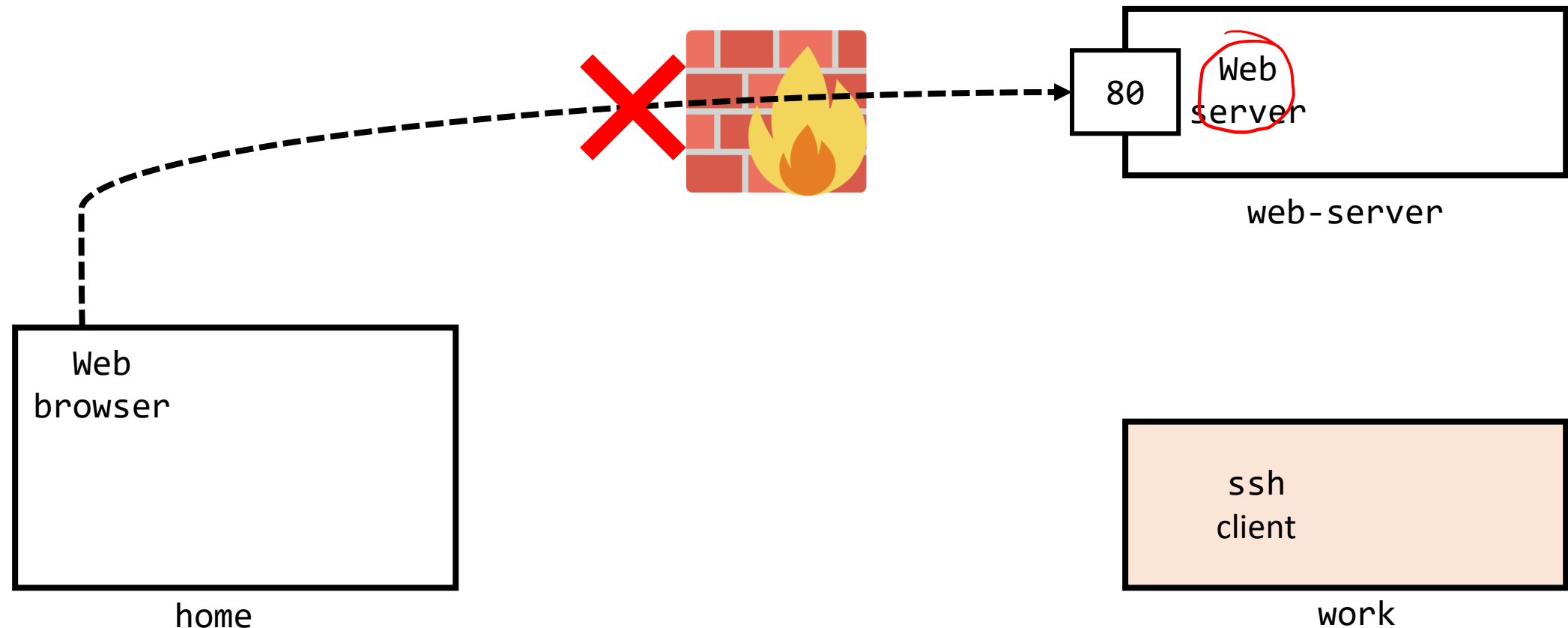
- Dynamic port forwarding allows configuring one local port for tunnelling data to all remote destinations
- This is done by creating a SOCKS proxy



- The application (e.g., Web browser) needs to support SOCKS

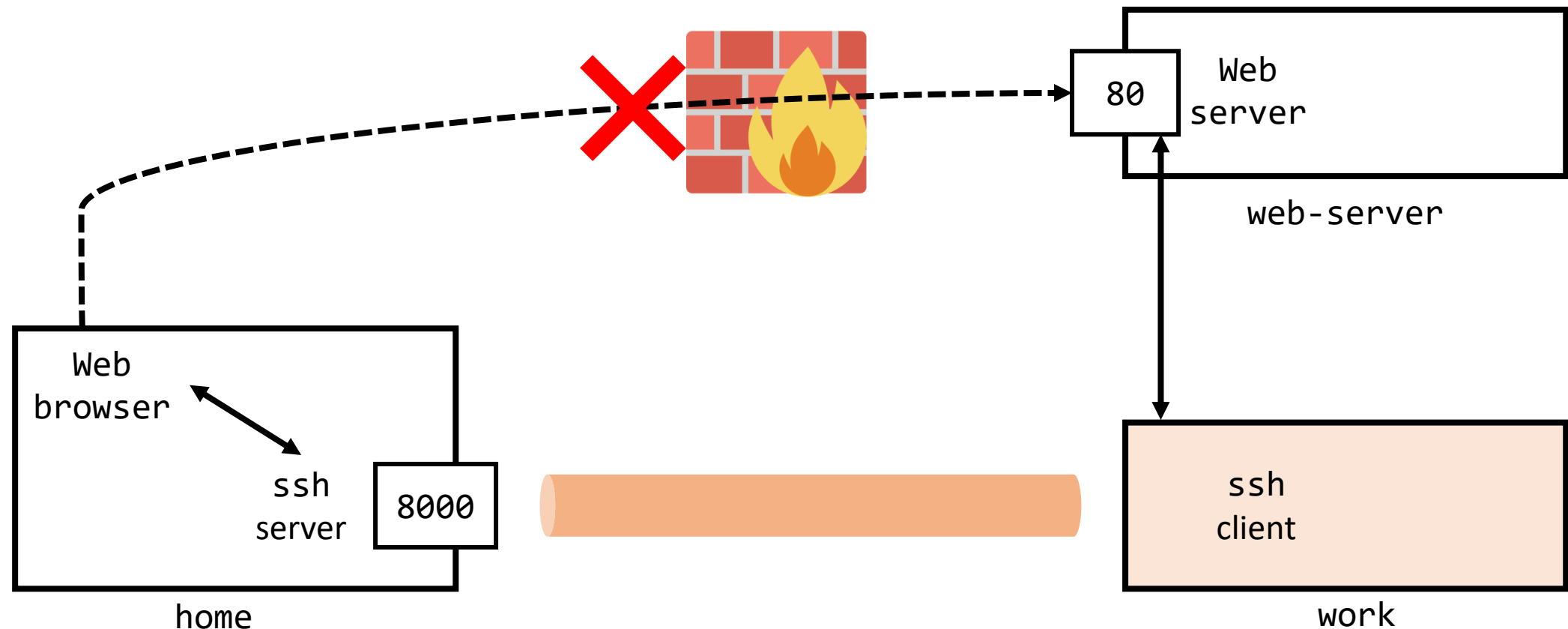
# Remote Port Forwarding

- Used to access a service inside a private network
  - Especially, when inbound ssh is not allowed, but outbound ssh is allowed



# Remote Port Forwarding

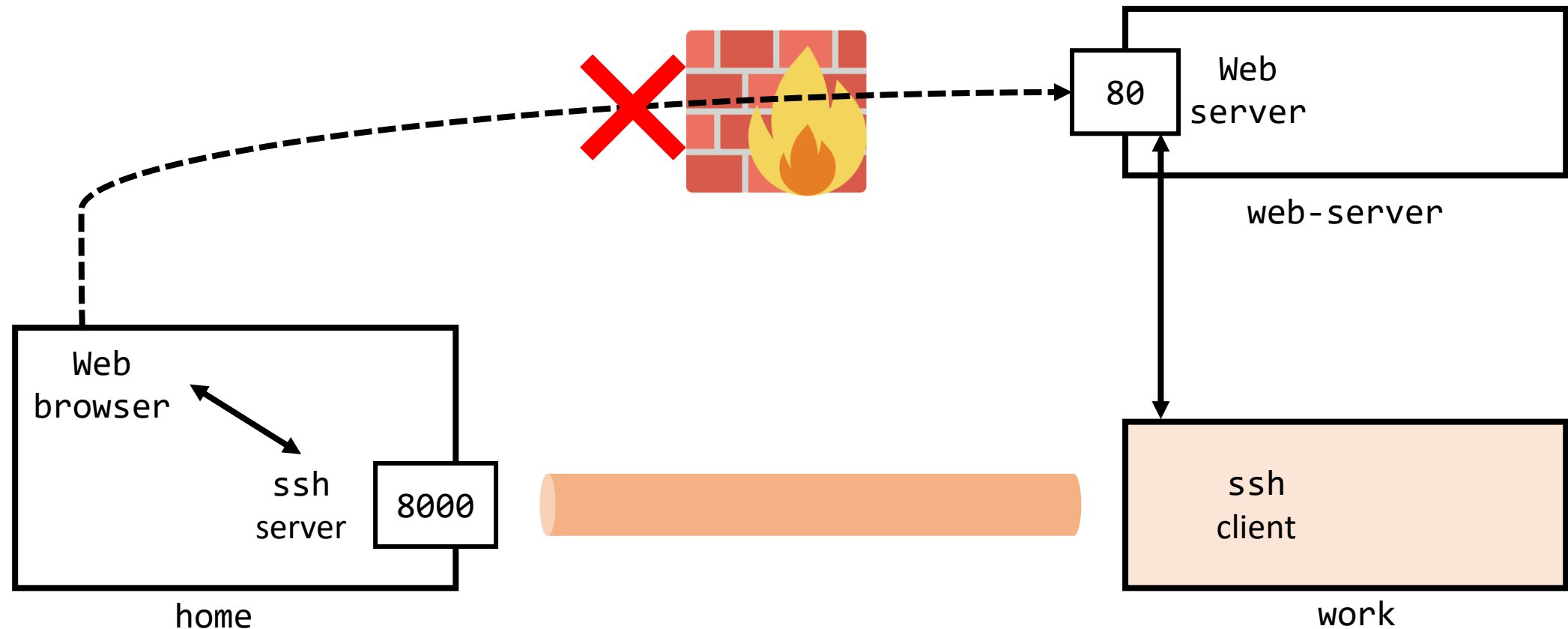
- We create a reverse SSH tunnel from work
  - On home, the user sends HTTP requests to port 8000
  - SSH tunnel forwards the requests to the SSH client on work
  - work forwards traffic to web-server



# Remote Port Forwarding

- We create a reverse SSH tunnel

```
work$ ssh -R 8000:web-server:80 user@home
```



# Summary

---

- Firewalls allow/block traffic based on the security policies
- Packet filtering firewalls:
  - Look at transport, network and link layers
  - Are implemented using iptables
- Application proxy firewalls support session-based policies
- Tunneling is the major technique to bypass firewalls