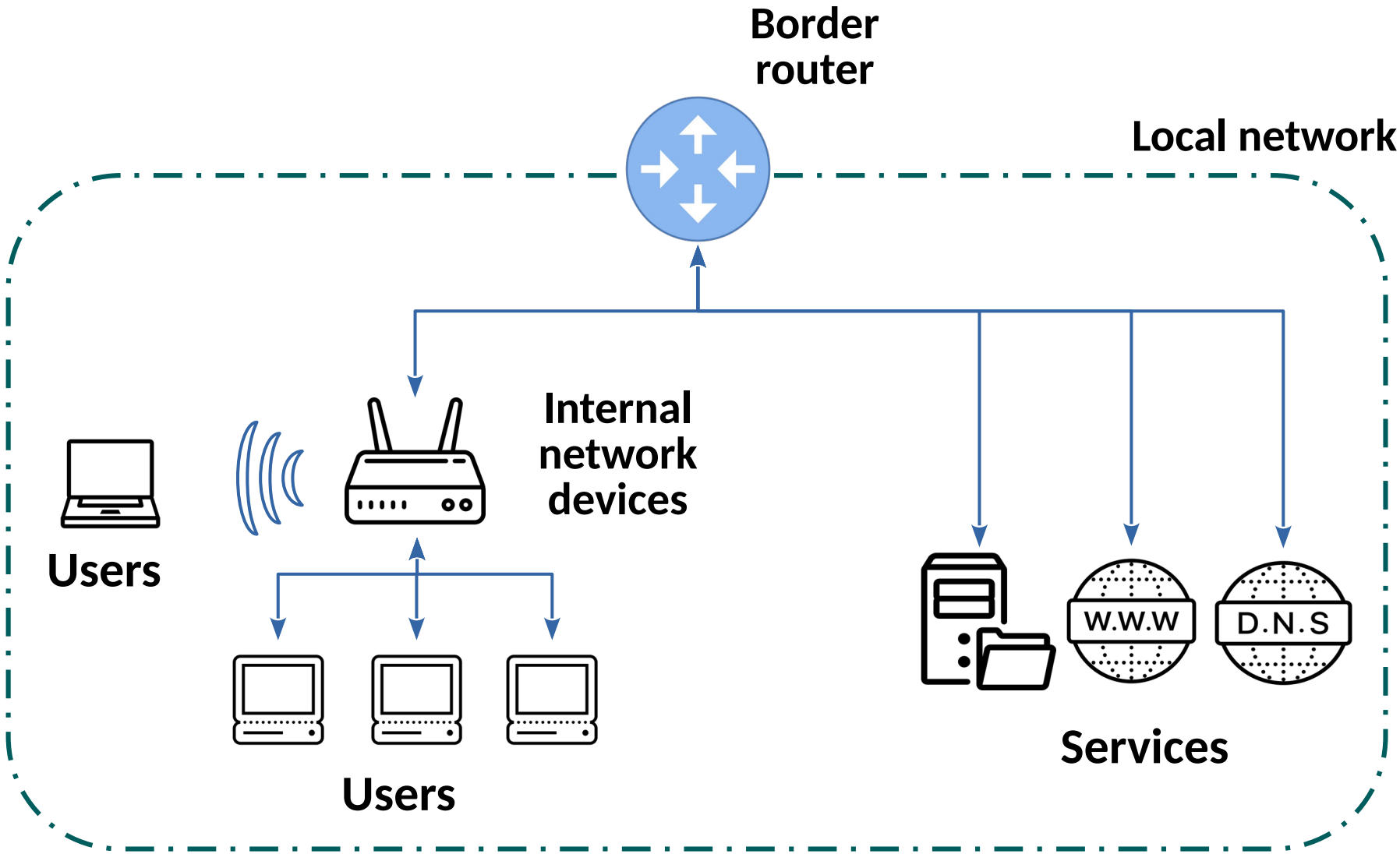
Security of local networks



Network security principles: Segmentation and Segregation [1]

- **Segmentation**: partitioning the system resources
  - From physical and/or logical perspectives
  - *Networks*, domains, roles, …

- **Segregation**: control accesses among segmented resources
  - Define policies to determine which accesses to allow
  - Deploy technologies to control and monitor

Network security principles: Segmentation and Segregation [3]

- We consider the following techniques and technologies:
  - Segmentation of Ethernet networks to avoid L2 attacks

    → **VLANs**
  - Segregation of network and transport layers

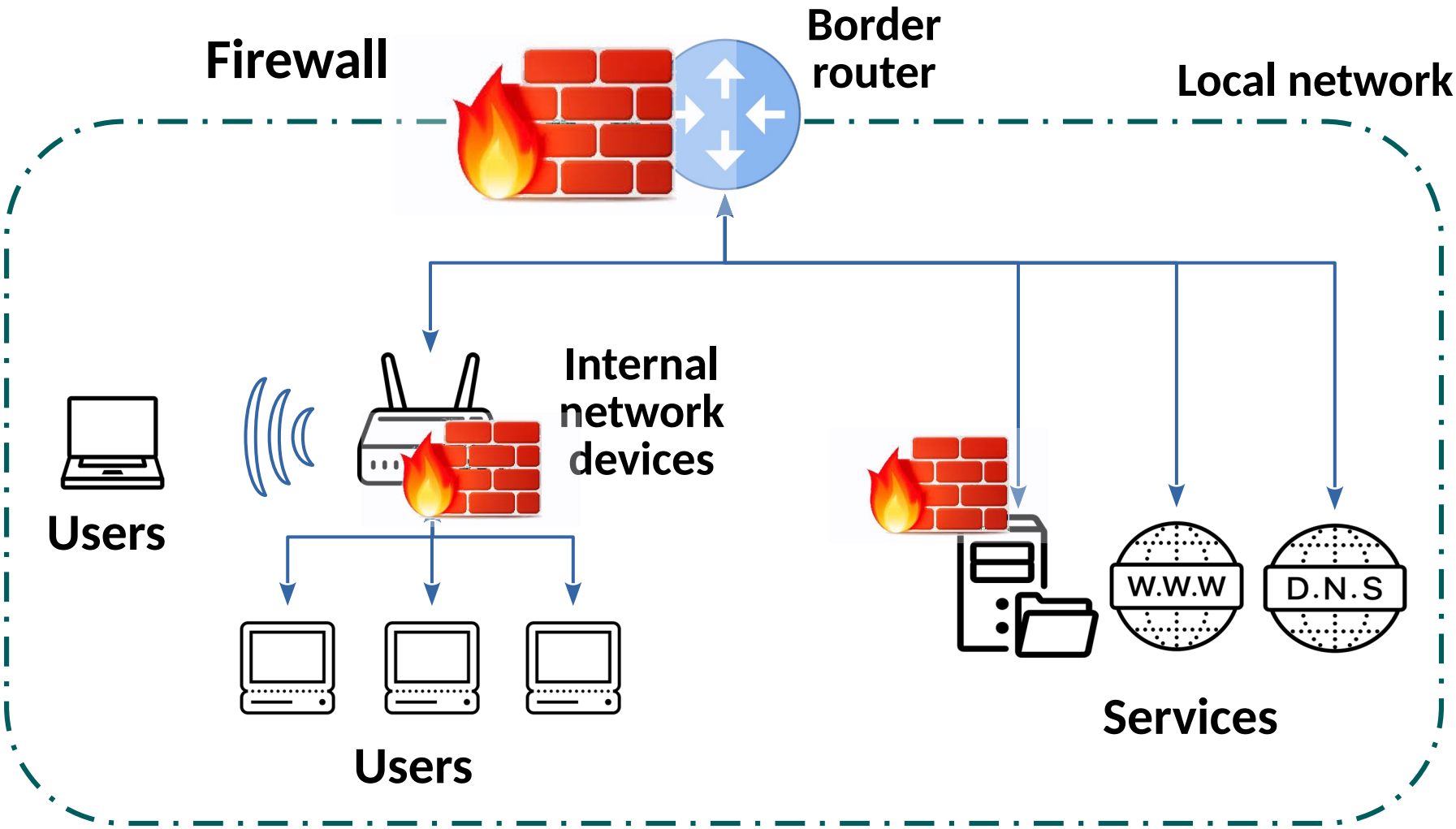
    → **L3 and L4 Firewalls**
  - Segregation of application communications

    → **Deep Packet Inspection (DPI)**

    → **Application Layer Firewalls**
- Other techniques that are not security-specific can also contribute security:
  - **NAT protocols** can also be used to **segment IP networks (separate address space)**

Segregation at L3 and L4 layers: Firewalls

- At the network and transport layers we can enforce separation and segregation by using **firewalls**



Firewall

- Firewalls are systems that aim at **segregating** network traffic between **separated** networks
- Some **principles**
  - Firewalls are not appliances that can be used as-is, they **must be configured** with regard to **requirements** and **policy**
  - Firewalls must be **highly secure**, because they themselves can be the target of attacks
  - Firewalls must be **correctly placed** within the network
    - Or, better, when designing the network we must consider how to separate the networks and where to place firewalls

Security policies

- Security policies are defined by **Access Control Lists (ACL)**:
  - In the context of firewalls, define which hosts and services are allowed and prohibited
- Two typical approaches:
  - **Implicit negation**: deny all, allow some
    - Prefer security over usability
    - This is the typically suggested approach for firewalls
  - **Implicit allow**: allow all, deny some
    - Prefer usability over security

Firewall access control policies

Examples of **Implicit negation** firewalls access control policies

- Deny all traffic in *both directions*, but
  - allow outgoing traffic to 80 and 443 TCP ports
    - But block source IPs that operate *too many requests*
  - allow outgoing traffic to 53 UDP port
  - allow incoming requests to a FTP server
- Deny all ingoing network, allow all outgoing traffic
  - allow …

Type of analysis

We distinguish two main types of analysis

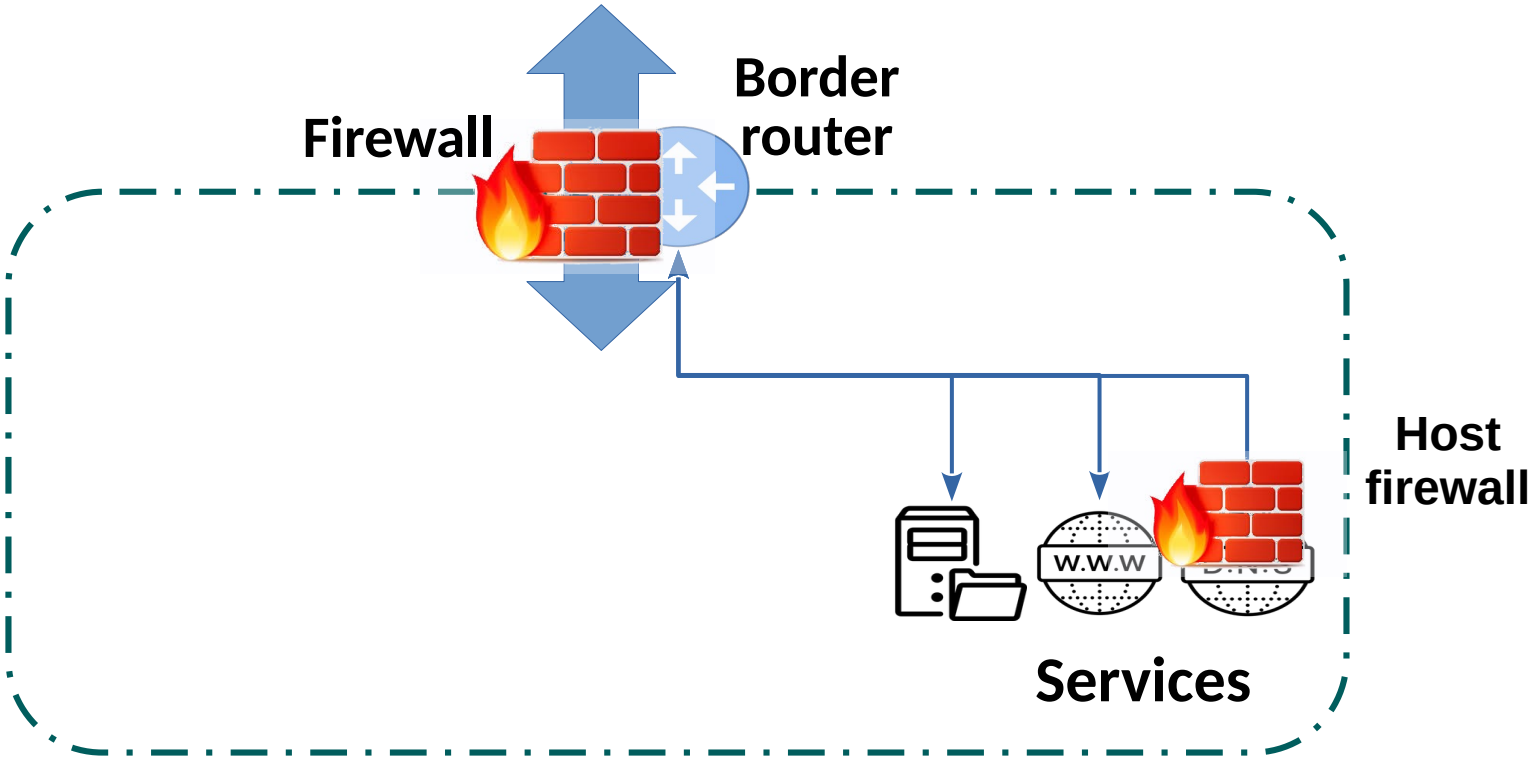
- **Static/stateless packet filtering**
  - Each packet is analyzed independently
  - Simpler and faster approach
  - Could not be able do enforce certain policies
- **Stateful packet filtering**
  - Packets are analyzed in groups
  - Able to analyze very important, such as ***connections***, but even complex **application-related states** (e.g., inspection of application payloads for dynamically allowing certain types of traffic)
  - Could be slower and/or have higher computational/memory requirements

Taxonomy of firewalls

- How it is distributed
  - **Hardware** → provided as/with appliance
  - **Software** → no appliance, software-only
- **Where it is deployed (or, which traffic can control)**
  - **Host** → installed on a personal computer or server
  - **Network** → installed on a router
- **Supported protocols**
  - L3 and L4 protocols
  - Application gateway

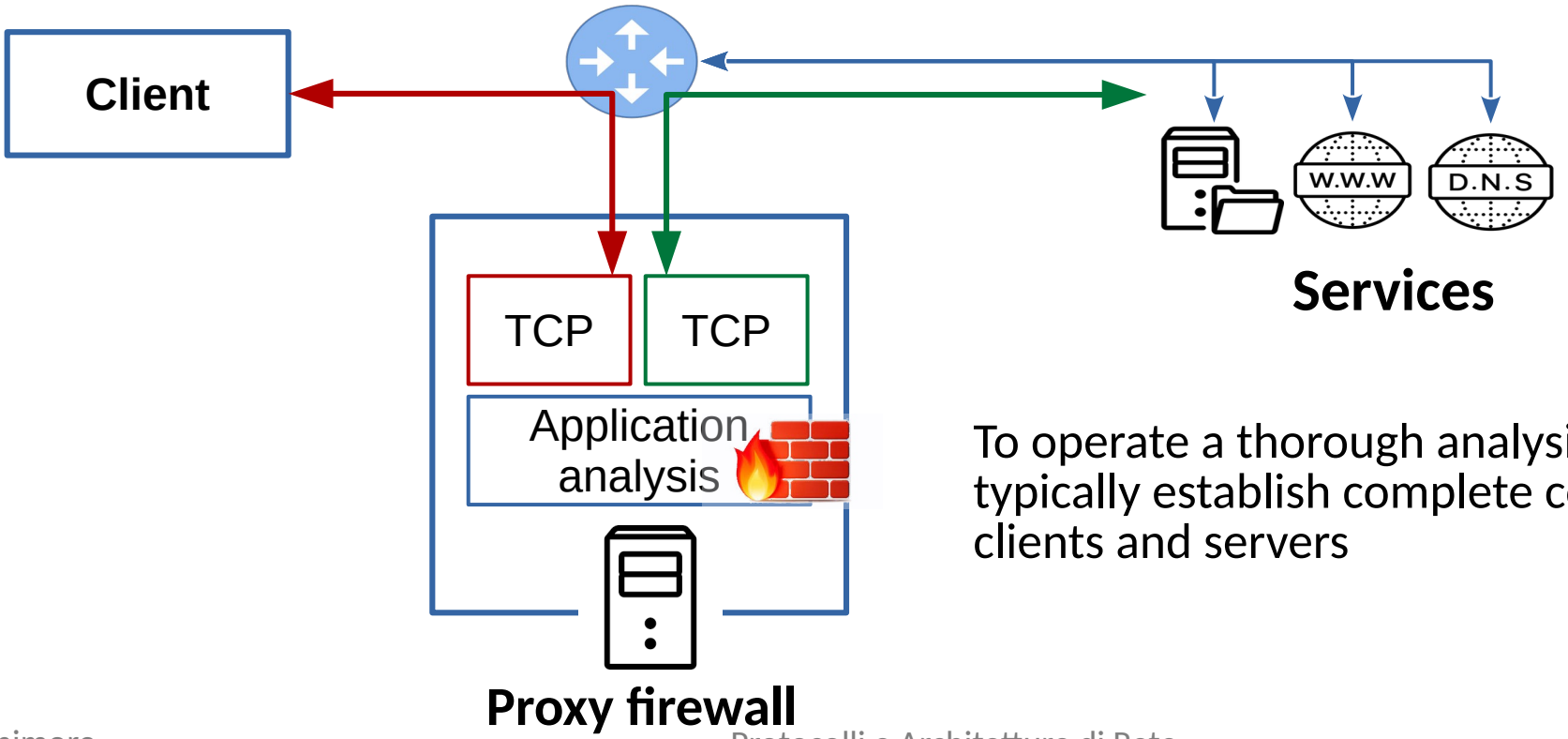
Deployment

- **Network:** placed on “strategic” **routers** within the network, analyze forwarded traffic
- **Host**: placed on a host, monitor input and output **traffic of local applications**
  - e.g., we can configure a firewall on our own computer



Proxy firewalls (application gateways)

- A typical firewall analyzes L3 and L4 layers
  - DPI allows *some* analysis of the application layer
- **Proxy firewalls** (also called **application gateways**) are software that fully implement specific applications logic to operate a *complete analysis*
  - Called **transparent proxy firewalls** if they do not require configuration of hosts
  - Beware of performance penalties



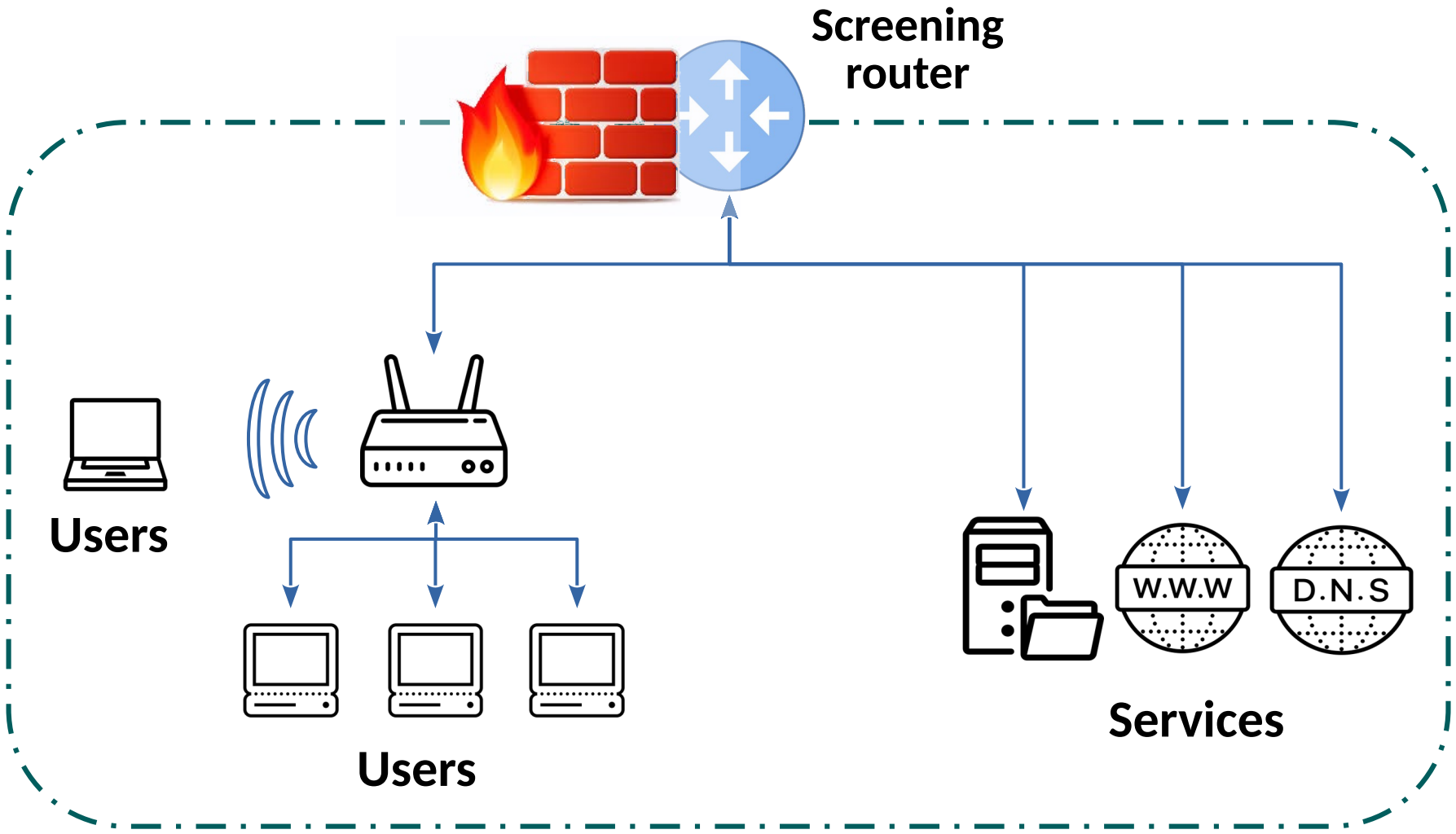
To operate a thorough analysis, the firewall typically establish complete connections with clients and servers

Defense-in-depth paradigm

- **Defense-in-depth** → Multiple firewalls can (or must) used at the same time
- Multiple approaches could also be **integrated within the same software/system**
  - e.g., a firewall that operate both packet filtering and has some application-level analysis capabilities

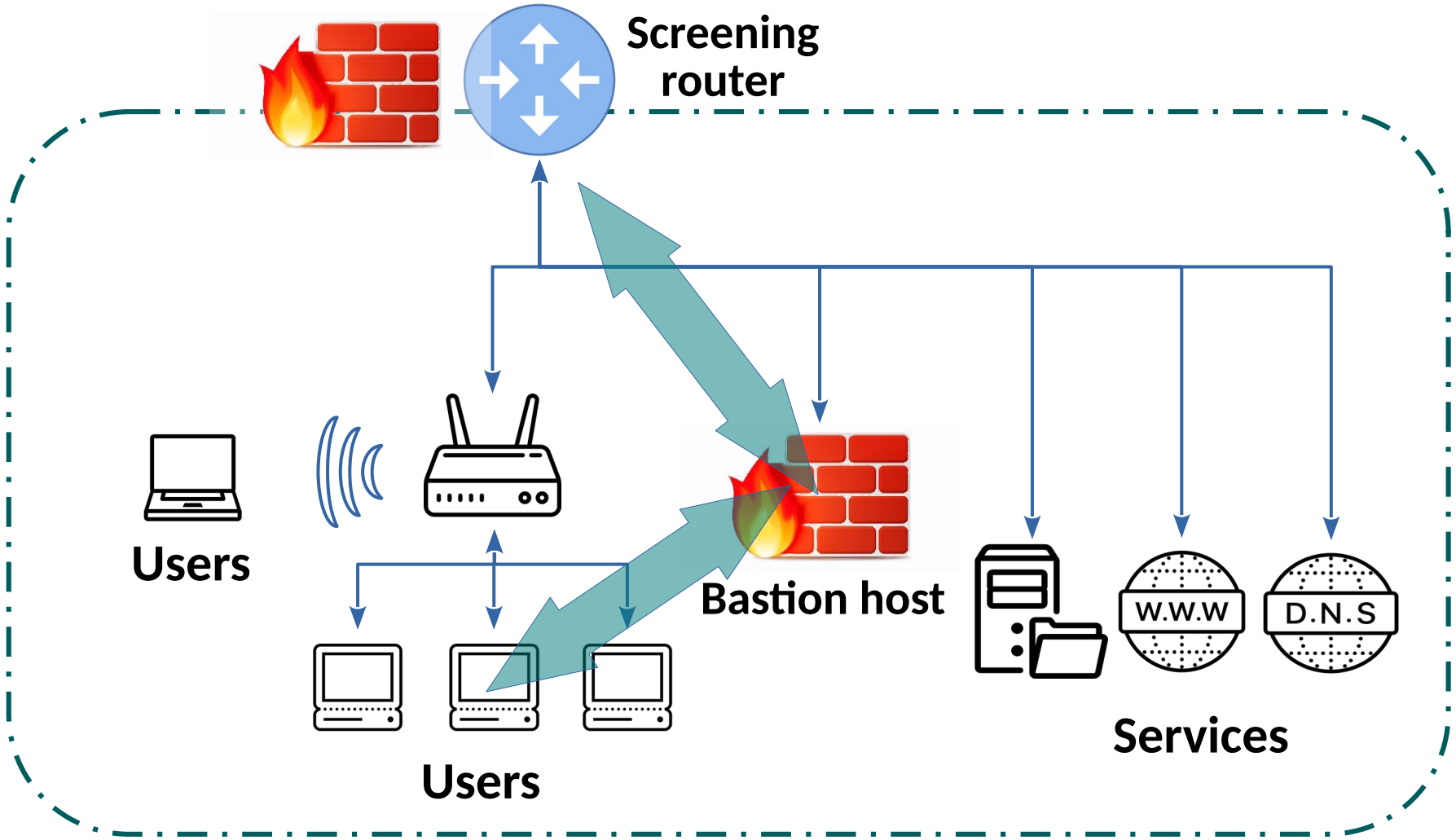
Single screening router [1]

- Single machine both as border router and as packet filter firewall



Screened-host gateways [1] – Bastion host

- Extends the screening router architecture with a special-purpose **proxy firewall called bastion host** dedicated to **inspect communications** between trusted and untrusted networks, typically at the application layer

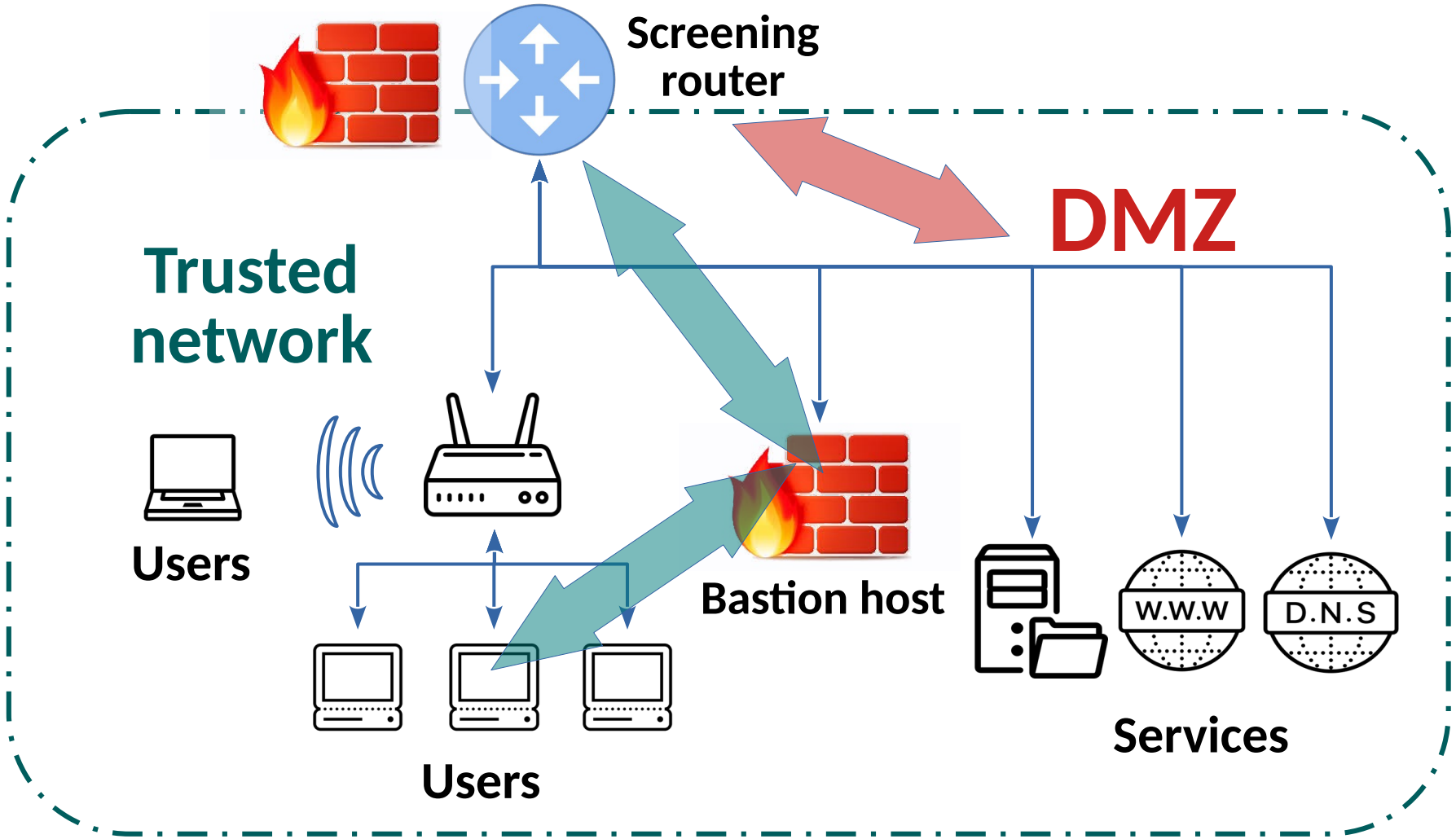


Screened-host gateways [2] – Bastion host

- The screening router operates acts as a **first defense layer**
  - Block all packets sent to the local network
  - Allows only packets sent to the **bastion host**
- The **bastion host** operates further inspections
  - Additional packet filtering operations
  - **Proxy firewall** for application-level inspection
- The bastion host typically has a single network interface and is physically connected to the same network of the other hosts
- The configurations of the screening router and the bastion host must be aware of each other
  - More complex to maintain, but also offer more flexibility
- *Note: some communications of the network could not require using the bastion host → DMZ*

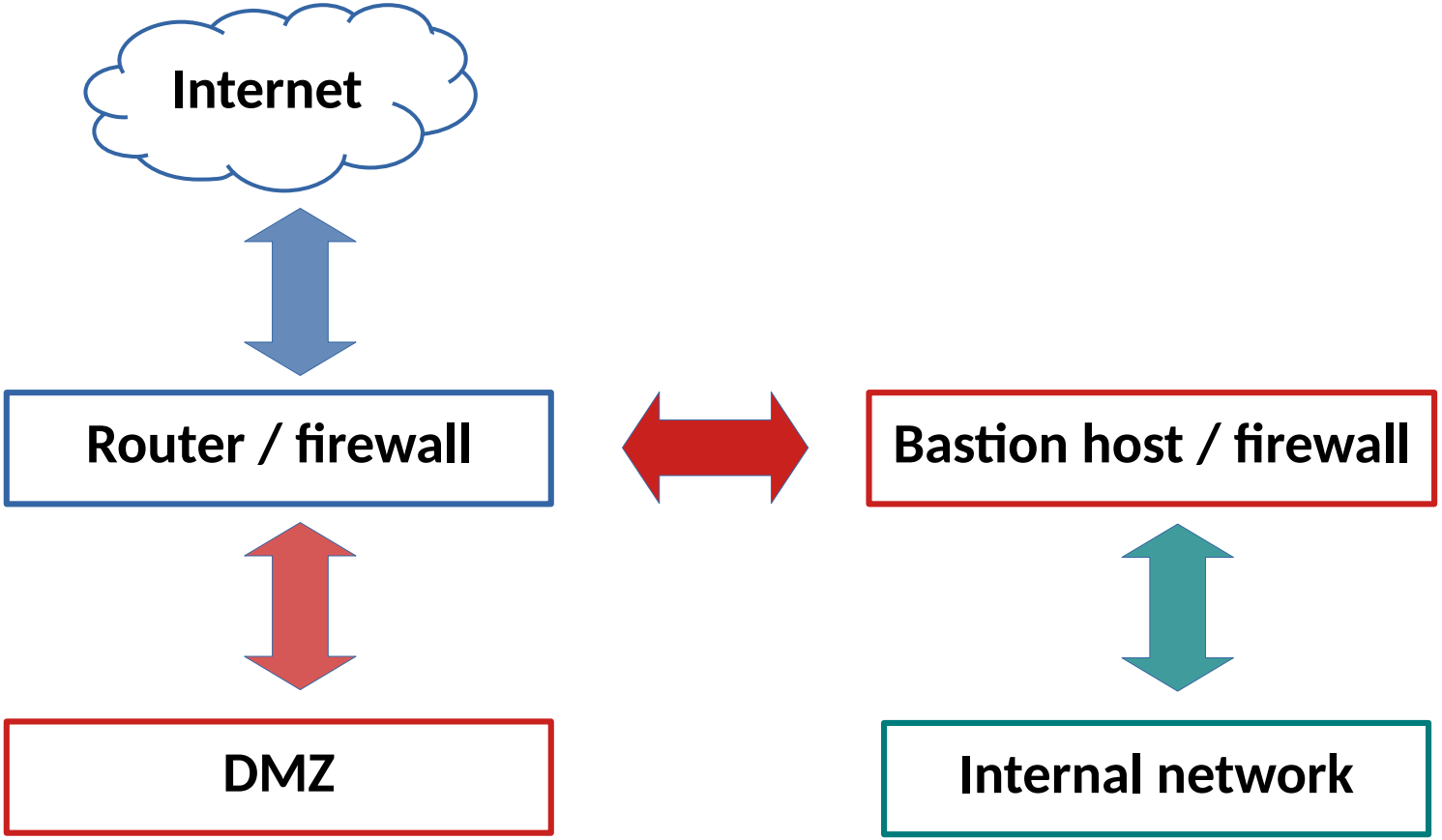
De-Militarized Zones (DMZ)

- A DMZ is an intermediate area between **external (untrusted)** and **internal (trusted)** networks
  - e.g., hosts within the DMZ can communicate directly with external networks, without using the bastion host

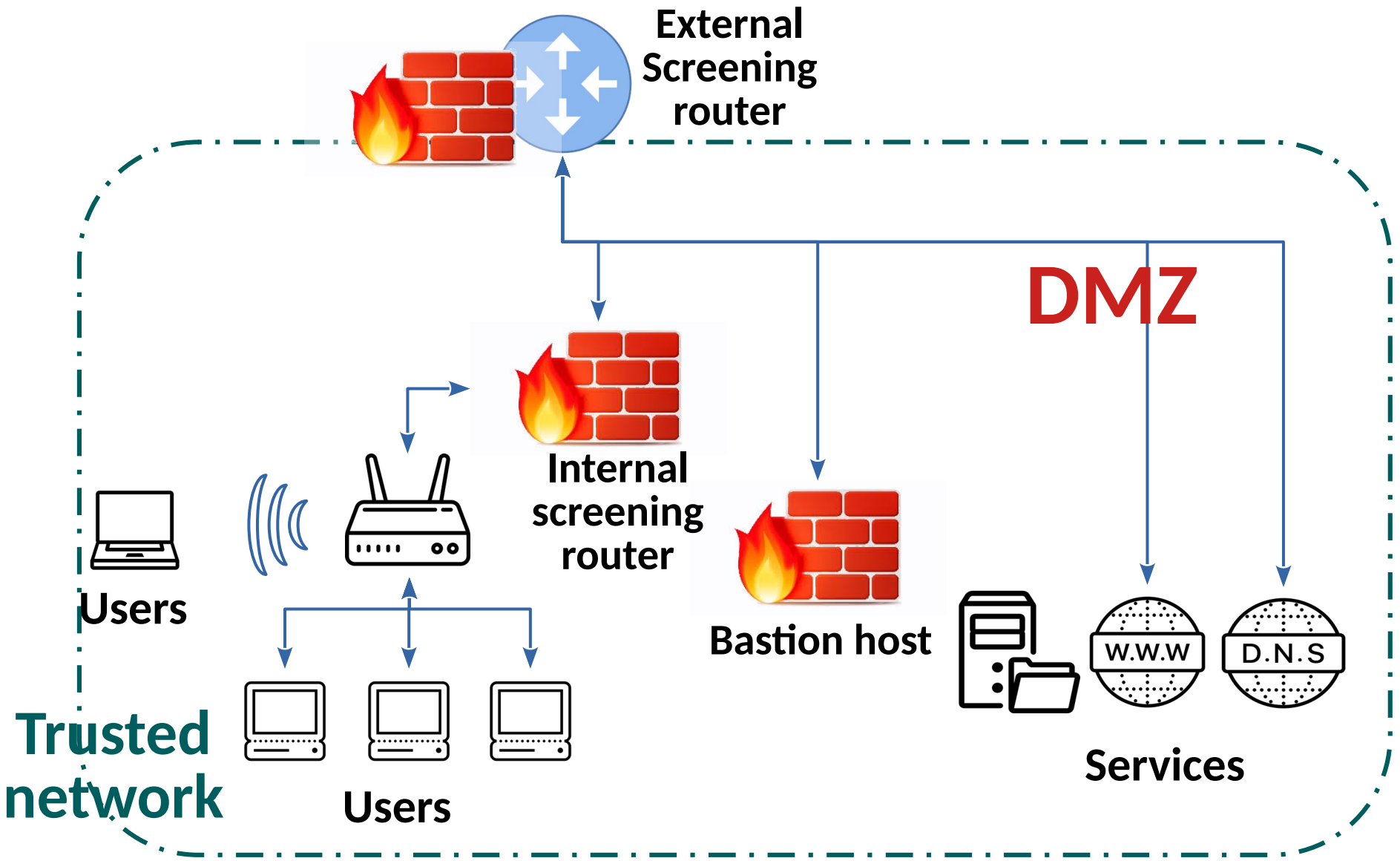


DMZ

- Example of flexible defense-in-depth and separation
  - The first layer of defense applies to the whole network
  - The second layer only applies to a subset of the network



Screened subnet [1]



Screened subnet [2]

- Example of multi-layered defense, separation and segregation in the context of networks
- The external firewall filters traffic between Internet and the DMZ
  - Policies in the external firewall allows traffic to the DMZ, including the bastion host
- The internal firewall protects both
  - The internal network from external attacks
  - The DMZ from attacks by the internal networks (malicious or compromised users)
- The approach could be further extended both vertically (e.g., additional security-critical networks within the internal network) and horizontally (e.g., multiple separated internal networks each with a dedicated firewall)

Firewall su Linux con IPtables

- IPtables è un software per implementare funzionalità di Packet Filtering, di Inspection, di NAT e di marking dei pacchetti.
  - Lo abbiamo già utilizzato per NAT
  - Ora lo utilizzeremo per realizzare dei firewall
  - **Ripassiamo il suo funzionamento nelle prossime 4 slide**

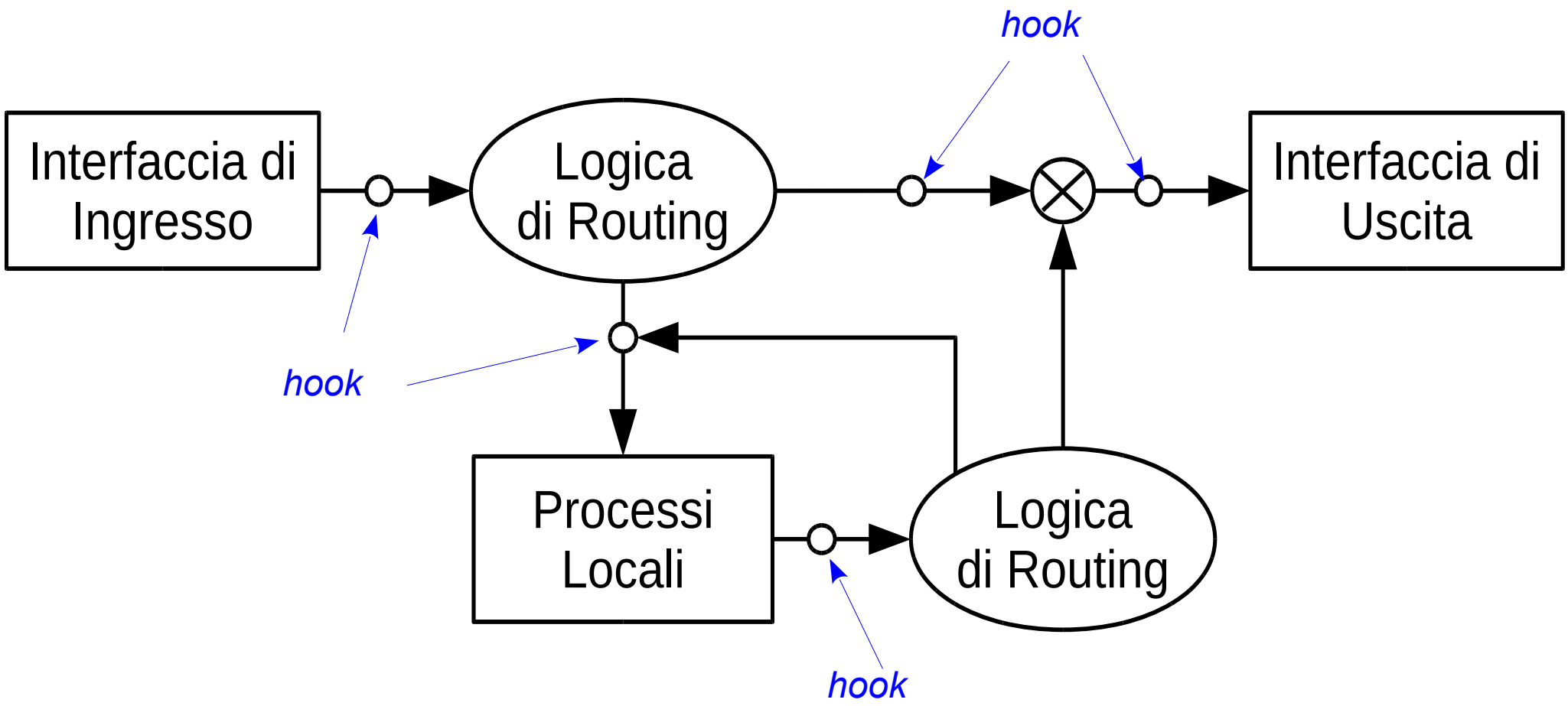
IPtables (1)

- È un front-end per il modulo **netfilter** del kernel Linux
- Sostituito dal front-end **nftables**, ma ancora utilizzato in contesti non avanzati per una maggiore semplicità
  - Nelle macchine virtuali impiegate in laboratorio è presente nftables, e ogni volta che inseriamo una regola con iptables questa viene convertita per nftables
- IPtables consente la realizzazione di ***regole*** per eseguire diversi tipi di operazioni sui pacchetti.
- Lo stack TCP/IP è gestito dal sistema operativo, quindi IPtables deve potersi interfacciare con il Kernel Linux.
- L'interfacciamento con il Kernel Linux, IPtables sfrutta il modulo Netfiler.
- Tale modulo opera fornendo agganci (***hooks***) al sistema operativo utilizzabili per intercettare i pacchetti in transito.

IPtables (2)

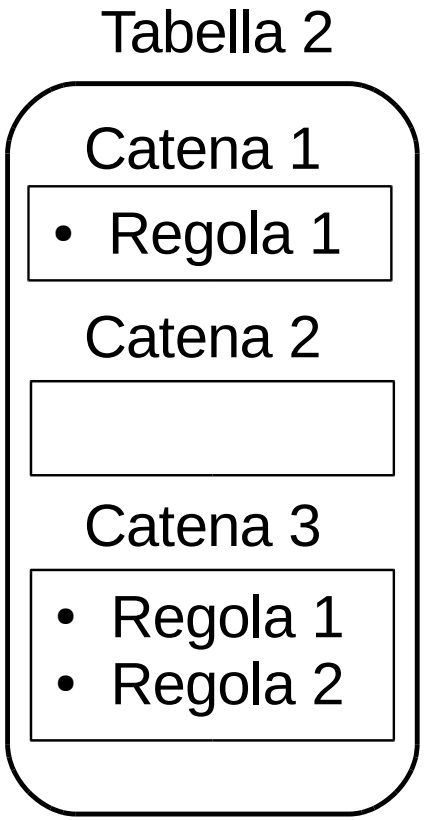
- Ogni volta che un pacchetto attraversa un ***hook***, Netfilter controlla se a quel determinato punto è stata assegnata una *funzione di gestione*:
  - se sì, il pacchetto viene passato alla funzione;
  - se no, il pacchetto passa all'hook successivo

IPtables (3)


hook
Interfaccia di Ingresso
Logica di Routing
Interfaccia di Uscita
hook
Processi Locali
Logica di Routing
hook

IPtables (4)

- In sintesi, ogni ***regola***:
  - viene applicata in una certa fase di gestione dei pacchetti in base all'***hook*** su cui agisce
  - definisce su **quali** pacchetti deve essere applicata (e.g. IP, porta, iface)
  - definisce **come** modificare i pacchetti
- Le regole sono raggruppate in ***tabelle*** in base alle funzionalità alle quali si riferiscono.



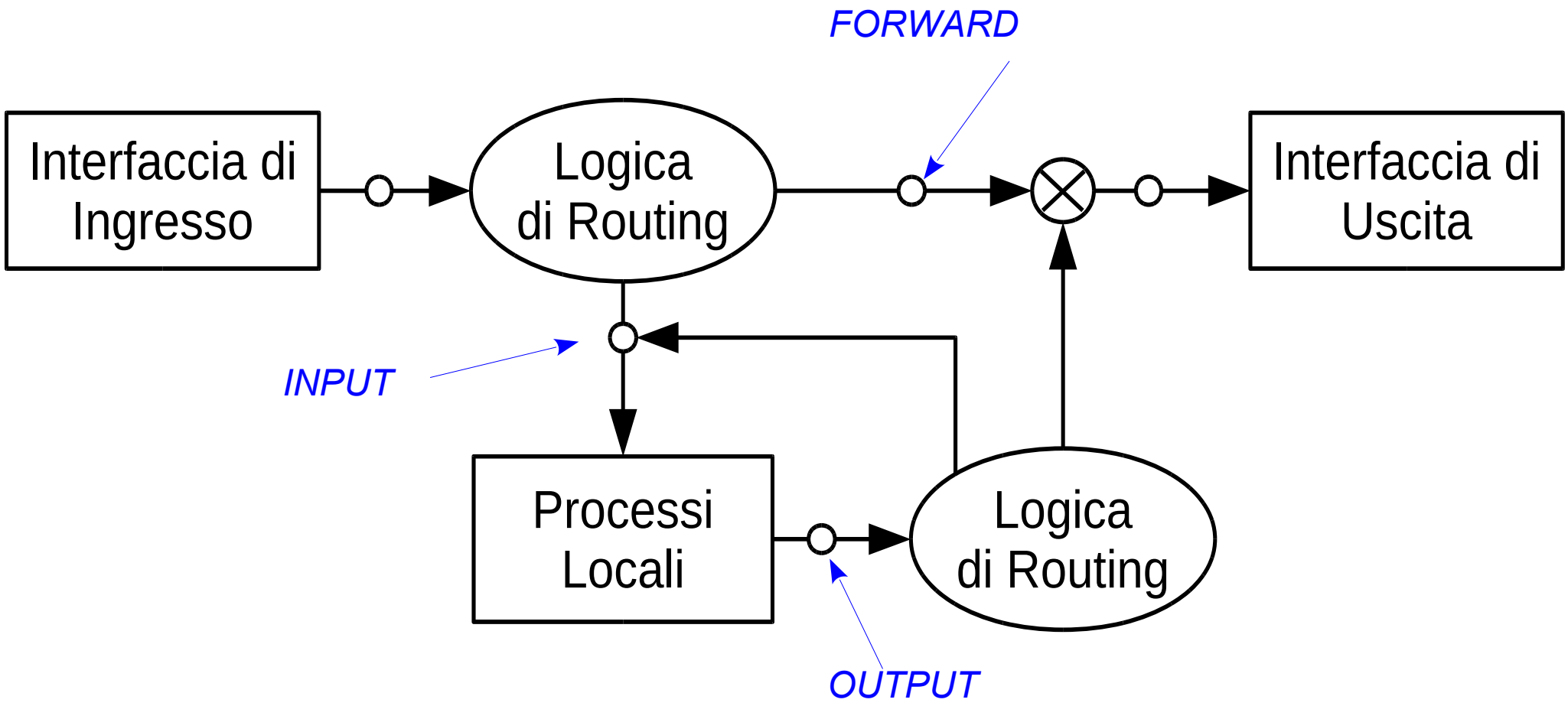
- Regole appartenenti a una stessa tabella che “insistono” sullo stesso hook appartengono a una stessa ***catena***, e vengono eseguite in una seguenza ordinata.

Packet Filtering con IPtables (1)

Di default sono presenti tre ***tabelle:***

- **Filter**: per operazioni di filtraggio.
- **Mangle**: per le funzionalità di *marking* dei pacchetti e per effettuare modifiche ai campi TOS e TTL.
- **Nat**: per le funzioni di *Masquerading*, *Port Forwarding* e *Transparent Proxy*.

Packet filtering con IPtables (1)



Packet filtering con IPtables (2)

Due catene per regolare il traffico degli **host**:

- **INPUT**: controlla le regole di accesso per il traffico in ingresso verso i processi locali
- **OUTPUT**: controlla le regole di accesso per il traffico in uscita dai processi locali

Una catena per regolare il traffico in transito su un **router:**

- **FORWARD**: controlla le regole di accesso per traffico in transito fra due interfacce di rete

Visualizzazione delle regole della tabella filter

```
# iptables [-t filter] [-v] [-n] -L [<chain>]  [--line-numbers]
```

Esempio:

```
# iptables -t filter -nvL
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target prot opt in out  source    destination
    0     0 ACCEPT tcp  --  *  *    0.0.0.0/0 0.0.0.0/0   tcp dpt:80 state NEW,ESTABLISHED

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target prot opt in out  source    destination
    0     0 ACCEPT tcp  --  *  *    0.0.0.0/0 0.0.0.0/0   tcp spt:80 state ESTABLISHED
```

Eliminare una regola

```
# iptables [-t filter] -D <chain> <rule_number>
```

Esempio:

```
# iptables [-t nat] -D INPUT 1
```

Elimina la prima regola di iptables della chain *INPUT* nella tabella nat

Impostazione Policy

- Dobbiamo scegliere la **policy** che definisce il comportamento di default del firewall:
  - Policy di **accettazione implicita**
  - Policy di **negazione implicita**

```
iptables -P <chain> <TARGET>
```

Esempi:

```
# iptables -P INPUT ACCEPT
```
← **accettazione implicita**

```
# iptables -P INPUT DROP
```
← **negazione (drop) implicita**

Aggiungere eccezioni

- Possiamo aggiungere regole alle catene per creare delle eccezioni alle regole di default:
  - Specificando delle **condizioni** di applicazione
  - Specificando il **target** (e.g., ACCEPT, REJECT, DROP)

```
iptables -A <chain>
   -i <input-interface>
   -o <output-interface>
   -s <source-ip/network>
   -d <destination-ip/network>
   -p {udp,tcp} --dport <destination-port> --sport <source-port>
   -m state --state <states>
```

- Inseriamo le regole opportune in base alla situazione e alle necessità

Rendere permanenti le modifiche di IPtables

***Opzione 1 (sconsigliata)*:**

impostare le regole di IPtables nel file */etc/network/interfaces* con il comando ***post-up***, alla stregua delle regole di routing

***Opzione 2 (consigliata)*:**

usare **netfilter-persistent**

`netfilter-persistent save` → memorizza le regole correnti

`netfilter-persistent reload` → ricarica a runtime le regole persistenti (utile per ripristinare una configurazione "stabile" in seguito ad alcuni tentativi di modifica a runtime)

Note:

netfilter-persistent è un servizio di sistema (gestibile con systemctl)

le regole sono memorizzate nel file `/etc/iptables/rules.v4` e `/etc/iptables/rules.v6` rispettivamente per IPv4 e IPv6

Filtri ICMP

- Possiamo creare filtri per diversi tipi di protocolli già visti
- Ad esempio, per protocolli ICMP (e.g., typename = {**echo-request**,**echo-reply**}):

  ```
  -p icmp --icmp-type <typetype>
  ```

  o più genericamente indicando il codice del protocollo icmp su cui vogliamo agire

  ```
  -p icmp --icmp-type <type>[/code]
  ```

- Nota: come gestire politiche di Path MTU discovery?

Gestione stato flussi: NEW, ESTABLISHED

- Con l'opzione **`--state`** possiamo indicare dei filtri sullo *stato* di **flussi di pacchetti**
  - Nota: vale sia quando lo stato della connessione è esplicito nel protocollo (protocolli connection-oriented come TCP), sia "estrapolando" una logica di connessione in altri contesti (e.g., identificativo pacchetti ICMP, o pacchetti UDP di "risposta" entro certi timeout)
- Consentire solo pacchetti relativi a una nuova "connessione"

```
iptables … -m state --state NEW
```

- Consentire pacchetti relativi sia a una nuova connessione, sia a una connessione già aperta

```
iptables … -m state --state NEW,ESTABLISHED
```

Esempi di regole di configurazione: Host

Accettare il *nuovo* traffico HTTP in ingresso (interfaccia di rete imprecisata)

```
iptables -A INPUT -p tcp --dport 80 \
         -m state --state NEW,ESTABLISHED -j ACCEPT
```

Accettare il traffico HTTP in uscita solo se è relativo a richieste già accettate in ingresso (come prima, interfaccia di rete imprecisata)

```
iptables -A OUTPUT -p tcp --sport 80 \
         -m state --state ESTABLISHED -j ACCEPT
```

Esempi di regole di configurazione: Router

Accettare il traffico HTTP destinato a un server interno (assumendo eth0 come interfaccia pubblica, eth1 come interfaccia interna)

```
iptables -A FORWARD -p tcp --dport 80 \
         -i eth0 -o eth1 \
         -d <ip-address> \
         -m state --state NEW,ESTABLISHED -j ACCEPT
```

Accettare il traffico HTTP in uscita solo se è relativo a richieste già accettate in ingresso (interfacce come sopra)

```
iptables -A FORWARD -p tcp --sport 80 \
         -i eth1 -o eth0 \
         -s <ip-address> \
         -m state --state ESTABLISHED -j ACCEPT
```

Esempi di regole di configurazione: Ping

- Esempio relativo a un host che vogliamo riuscire a “pingare”

Accettiamo i pacchetti icmp echo request in ingresso (interfaccia di rete imprecisata)

```
iptables -A INPUT -p icmp --dport 80 \
         -i eth0 -o eth1 \
         -d <ip-address> \
         -m state --state NEW,ESTABLISHED -j ACCEPT
```

Accettiamo i pacchetti icmp echo response in uscita associati a un flusso di traffico già aperto

```
iptables -A FORWARD -p tcp --dport 80 \
         -i eth0 -o eth1 \
         -d <ip-address> \
         -m state --state NEW,ESTABLISHED -j ACCEPT
```

L'opzione RELATED

- L'opzione ESTABLISHED serve ad abilitare una logica di analisi stateful strettamente legata al concetto di "flusso di pacchetti"
  - e.g., quintupla IP SRC + IP dst + Port Src + Port Dst + Proto L4; echo id
- A volte, ci sono logiche di funzionamento più complesse, in cui pacchetti apparentemente non appartenenti
  - e.g., vari pacchetti di risposta ICMP (e.g., redirect, too big, unreachable), e alcuni protocolli applicativi basati su molteplici connessioni TCP (e.g., FTP)
- L'opzione RELATED può consentire al firewall iptables di cercare di abilitare una logica di rilevazione più flessibile
  - NOTA: maggiore flessibilità in alcuni casi può essere sfruttata in maniera malevola, quindi RELATED va usato se ritenuto necessario

Esempio RELATED

- Consentire a un host di inviare pacchetti ICMP port unreachable nel caso di servizi non disponibili

```
iptables -A OUTPUT … -p icmp \
      --icmp-type port-unreachable \
      -m state --state RELATED -j ACCEPT
```

- Filtri analoghi possono essere impiegati anche:
  - Su Router, inviare diversi tipi di segnalazioni ICMP (e.g., destination unreachable, too big…)
  - Su Host accettare i relativi pacchetti ICMP

- Vedere un elenco più esaustivo con

```
iptables -p icmp -h
```