Some details on Layer 5: Application layer (= layers 5,6,7 of OSI model)

DNS: Domain Name System

- By far, the most popular application on the Internet … because all the others rely on it.

- **A simple service that provides a translation between host names and IP addresses (and vice versa)**
  - Example: cnrsformation.cnrs.fr ⇔ 195.220.198.93
  - Supports both IPv4 and IPv6

- **Motivation:**
  - Humans prefer using textual names rather than numeric identifiers
  - … but the machines in the core of the Internet (routers …) only understand/manipulate IP addresses.
  - Thus, anytime a given host named H_A wants to send an IP packet to another host named H_B, H_A must first use the DNS to obtain the IP address of H_B.

DNS: Domain Name System (continued)

- **The DNS is designed as a decentralized system**, for several reasons:
  - Performance and fault tolerance … despite an exponential growth
  - Flexibility: domain owners can administer their namespaces without interacting with top-level authorities

- **Main components:**
  - **For storing/retrieving reference information:**
    - Root servers
    - TLD (Top Level Domain) servers (e.g., for “.com” and ”.fr” domains)
    - Authoritative servers (for a given domain, e.g., cnrs.fr)
  - **For managing/relaying queries:**
    - Local resolver (local operating system)
    - DNS servers/resolvers (either hosted by the local network or open/public servers)

- **Note: A compromised DNS server can become a major security weakness.**

DNS lookup – Example

```
$ dig cnrsformation.cnrs.fr

; <<>> DiG 9.10.6 <<>> cnrsformation.cnrs.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55092
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 7

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;cnrsformation.cnrs.fr.                 IN            A

;; ANSWER SECTION:
cnrsformation.cnrs.fr.     3562         IN            CNAME         ods.lamp.cnrs.fr.
ods.lamp.cnrs.fr.          1371         IN            A             195.220.198.93

;; AUTHORITY SECTION:
lamp.cnrs.fr.              1371         IN            NS            ns3.cnrs.fr.
lamp.cnrs.fr.              1371         IN            NS            ns0.cnrs.fr.
lamp.cnrs.fr.              1371         IN            NS            ns2.cnrs.fr.

;; ADDITIONAL SECTION:
ns0.cnrs.fr.               1104         IN            A             195.220.197.10
ns0.cnrs.fr.               3129         IN            AAAA          2001:660:500a:2210::10
ns2.cnrs.fr.               1567         IN            A             193.52.36.172
ns2.cnrs.fr.               1579         IN            AAAA          2001:660:500a:2103::10
ns3.cnrs.fr.               924          IN            A             193.55.90.12
ns3.cnrs.fr.               2903         IN            AAAA          2001:660:6608:5106::12
```

DNS full lookup – Example

```
$ dig cnrsformation.cnrs.fr +trace

. 37954 IN NS k.root-servers.net.
. 37954 IN NS m.root-servers.net.
. 37954 IN NS g.root-servers.net.
. 37954 IN NS a.root-servers.net.
. 37954 IN NS f.root-servers.net.
. 37954 IN NS l.root-servers.net.
. 37954 IN NS b.root-servers.net.
. 37954 IN NS c.root-servers.net.
. 37954 IN NS e.root-servers.net.
. 37954 IN NS h.root-servers.net.
. 37954 IN NS i.root-servers.net.
. 37954 IN NS d.root-servers.net.
. 37954 IN NS j.root-servers.net.
;; Received 228 bytes from 8.8.4.4#53(8.8.4.4) in 5 ms

fr. 172800 IN NS d.ext.nic.fr.
fr. 172800 IN NS d.nic.fr.
fr. 172800 IN NS e.ext.nic.fr.
fr. 172800 IN NS f.ext.nic.fr.
fr. 172800 IN NS g.ext.nic.fr.
;; Received 347 bytes from 199.7.91.13#53(199.7.
ms
```

```
cnrs.fr. 172800 IN NS camus.dr15.cnrs.fr.
cnrs.fr. 172800 IN NS panoramix.rap.prd.fr.
cnrs.fr. 172800 IN NS ns2.cnrs.fr.
cnrs.fr. 172800 IN NS ns3.cnrs.fr.
cnrs.fr. 172800 IN NS ns0.cnrs.fr.
;; Received 335 bytes from 194.146.106.46#53(194.146.106.46) in 5590 ms

cnrsformation.cnrs.fr. 3600 IN CNAME ods.lamp.cnrs.fr.
ods.lamp.cnrs.fr. 300 IN A 195.220.198.93
lamp.cnrs.fr. 300 IN NS ns3.cnrs.fr.
lamp.cnrs.fr. 300 IN NS ns0.cnrs.fr.
lamp.cnrs.fr. 300 IN NS ns2.cnrs.fr.
;; Received 264 bytes from 193.52.36.172#53(193.52.36.172) in 99 ms
```

HTTP (1/2)

- **“HyperText Transfer Protocol”**

- Initially created for the World Wide Web
- … but also used by many other applications due to its:
  - Simplicity
  - Pervasive support in networks and hosts
  - “Connectivity” (i.e., usually not filtered by firewalls)

- Can be used for two main purposes:
  - **“Static requests”**: To download or upload “static content” (i.e., a pre-existing file)
  - **“Dynamic requests”**: To trigger the execution of a program on a remote host and obtain the result (“dynamic content”)

HTTP (2/2)

- **Properties**: reliable delivery, congestion control

- **Two main versions currently deployed**
  - **“HTTP 1.1”** and **“HTTP/2”**
  - Conceptually very similar but the low-level details are different
    - In particular, the wire format for the message headers is different (ASCII text vs. specific binary protocol)
  - Both versions are based on TCP

- **“HTTPS” is a secure version of HTTP**
  - HTTP over SSL (instead of directly over TCP)
  - Other aspects are identical to HTTP

HTTP requests (1/2)

- **HTTP is a client-server protocol, based on a simple request-response model.** Typically:
  - 1-1 mapping: client sends 1 request and obtains 1 corresponding response
  - Requests are processed in order

- **Each request contains:**
  - **A request header**, which includes notably a target URL
    - If the target URL corresponds to a folder rather that a file (e.g., www.cnrs.fr/) then the server chooses a default file name in that folder (e.g., www.cnrs.fr/index.html)
  - **A request body** (optional or mandatory according to the request type)

- **Each response contains:**
  - **A response header**, which includes notably a status code (e.g., "200 OK")
  - **A response body** (optional or mandatory according to the request type)

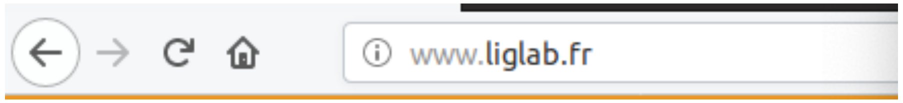
HTTP requests (2/2)

- **There are many types of requests defined by the HTTP specification but, in practice, only a few of them form the majority of the HTTP traffic.**
- **We will focus on the main types of requests (most used ones) :**
  - GET
  - PUT
  - DELETE
  - POST
- (In the following slides, we will illustrate them with the format of the HTTP 1.1 version: text headers)
- **cURL** (https://curl.haxx.se):
  - A useful tool to manipulate/issue HTTP and HTTPS requests (and also many other protocols)
  - From the command line and in scripts
  - Also, available as a library for C and many other languages (libcurl: https://curl.haxx.se/libcurl/)

HTTP GET request

- **The purpose of a GET request is to obtain a representation of the requested resource.**
- **The target URL is passed in the request header**
  - In two parts:
    - File path in the request line (first line)
    - Hostname in the “Host:” header field
- **In the case of a dynamic request, the request arguments are passed as a query string**
  - Appended behind the file path, with a “?” as separator
  - Format: ?arg1=value1&arg2=anothervalue&arg3=hello
- **In case of success, the representation of the requested resource is sent in the response body.**
  - The **type/format** of the resource (e.g., “image/jpeg”) is indicated in the **“Content-type:”** field of the response header
  - The **size** of the resource (in bytes) is indicated in the **“Content-size:”** field of the response header
- **Considered as a “safe” method:**
  - i.e., **should only be used for information retrieval (i.e., “read only” semantics)** and should not modify state on the Web server, nor have other external side effects.

HTTP GET request – Examples (1/3)



Request header:

```
GET / HTTP/1.1\r\n
Host: www.liglab.fr\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:69.0) Gecko/20100101 Firefox/69.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
Upgrade-Insecure-Requests: 1\r\n
\r\n
```

HTTP GET request – Examples (2/3)

Response header:

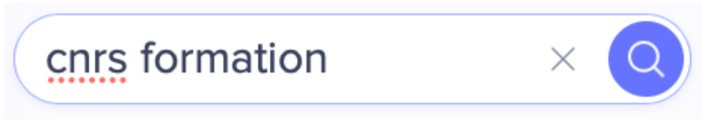
```
HTTP/1.1 200 OK\r\n
Date: Sun, 27 Oct 2019 10:32:45 GMT\r\n
Server: Apache/2.2.15 (CentOS)\r\n
X-Content-Type-Options: nosniff\r\n
X-Powered-By: PHP/5.3.3\r\n
X-Drupal-Cache: HIT\r\n
Etag: "1572092035-0"\r\n
Content-Language: fr\r\n
X-Frame-Options: SAMEORIGIN\r\n
X-Generator: Drupal 7 (http://drupal.org)\r\n
Cache-Control: public, max-age=0\r\n
Last-Modified: Sat, 26 Oct 2019 12:13:55
GMT\r\n
Expires: Sun, 19 Nov 1978 05:00:00 GMT\r\n
Vary: Cookie,Accept-Encoding\r\n
Content-Encoding: gzip\r\n
Content-Length: 7404\r\n
Keep-Alive: timeout=15, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=utf-8\r\n
\r\n
```

Response body (truncated):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr"
version="XHTML+RDFa 1.0" dir="ltr"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:og="http://ogp.me/ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:sioc="http://rdfs.org/sioc/ns#"
  xmlns:sioct="http://rdfs.org/sioc/types#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<head profile="http://www.w3.org/1999/xhtml/vocab">
  <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link rel="shortcut icon"
href="http://www.liglab.fr/sites/www.liglab.fr/files/pictures
/icon_lig.ico" type="image/vnd.microsoft.icon" />
<meta name="Generator" content="Drupal 7 (http://drupal.org)"
/>
  <title>LIG - Laboratoire d&#039;Informatique de Grenoble |
Laboratoire d'Informatique de Grenoble  <span>UMR
5217</span></title>
```

HTTP GET request – Examples (3/3)


cnrs formation

Request header (truncated):

```
GET
/sp/afs/ads?q=cnrs%20formation&adpage=1&gl=FR&r=m&client=startpage&channel=ch77&hl=en&adtest=off&adsafe=med
ium&type=0&psid=6873866198&oe=UTF-8&ie=UTF-
8&fexp=21404%2C17300003&format=p3&ad=p3&nocache=7061572180022100&num=0&output=uds_ads_only&v=3&adext=as1%2C
sr1&bsl=10&u_his=3&u_tz=60&dt=1572180022104&u_w=1449&u_h=935&biw=794&bih=423&psw=794&psh=1428&frm=0&uio=sl1
lo0sr1hcff2st18sd13sv14sa13sn13lt20ld20lv20va1-&cont=gcsa-
top&jsv=87645&rurl=https%3A%2F%2Fwww.startpage.com%2Fdo%2Fsearch&referer=https%3A%2F%2Fwww.startpage.com%2F
HTTP/1.1\r\n
Host: www.startpage.com\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:69.0) Gecko/20100101 Firefox/69.0\r\n
[…]
\r\n
```

HTTP PUT & DELETE requests

- **The purpose of a PUT request is to create a new resource on the Web server or to update an existing one.**
  - The target URL passed in the request header corresponds to the URL of the resource.
    - i.e., in the case of a creation, the proposed URL will be associated with the new resources.
  - **The data to be uploaded on the server is passed in the request body** (+ associated metadata in the request header)
    - Like for the POST request described on the next slide

- **The purpose of a DELETE request is to remove an existing resource on the server.**

- **Unlike a GET request, a PUT or DELETE request modifies the state on the server and may have external side effects.**
- **PUT and DELETE requests should be idempotent**
  - i.e., executing a request several times should produce the same result as a single execution

HTTP POST request (1/2)

- **Can be used for several purposes:**
  - **Main usages:**
    - Submission of a web form
    - Creation of a new resource on the server (upload)
  - Other usages:
    - Update of an existing resource on the server (upload)
    - Also, as a replacement for a GET request in some specific circumstances (e.g., dynamic request with a very long query string)
- **Unlike a GET request:**
  - May modify the state on the Web server or have external side effects
- **Unlike a PUT request:**
  - In the case of a creation of a new resource, the URL indicated in the request header is not the URL to be used for the new resource, but rather the URL of the resource that will handle the creation.
  - **A POST request is generally not idempotent**

HTTP POST request (2/2)

- **In any case, the data sent to the server is passed is the body of the POST request.**
  - In the case of the submission of a Web form, the data is formatted like a query string.

- **However, additional information is passed via the request header:**
  - The type/format of the resource (e.g., “image/jpeg”) is indicated in the “Content-type:” field of the request header
  - The size of the resource (in bytes) is indicated in the “Content-size:” field of the response header

REST APIs

- **Context: Network/Web services**
  - Nowadays **many services can be invoked**:
    - **Remotely** (i.e., from client machines over the network)
    - … and possibly **automatically** (i.e., from a fully automated client program, without human/manual intervention)
    - … **through a network/Web "API"** (application programming interface)
  - **Example 1: Web browsing**
    - Integration of features/contents from multiple sources on the same web page (social media, ads, payment, dashboards …)
  - **Example 2: Cloud computing**
    - Web services can be used to easily and quickly request/release computing & storage resources from cloud providers

- **"REST" is:**
  - An acronym for "Representational State Transfer" (Also known as "RESTful interfaces/APIs")
  - **A set a design guidelines followed by most Web services**
  - **(Not a technology per se; only a set of best practices for designing web-based services)**

REST APIs – Some characteristics (1/2)

- **Using URLs as resource identifiers**
- **Using HTTP verbs as operation identifiers**
  - … rather than inlining the operation type in the URL.
  - **4 possible operations on a given resource ("CRUD"): Create, Read, Update, Delete**
  - Respectively implemented with the following HTTP methods:
    - Create → HTTP POST, Read → HTTP GET, Delete → HTTP DELETE
    - For "updating" a resource, 2 variants: Full replace → HTTP PUT, Partial modification → HTTP PATCH
  - These operations can target a single resource or a collection of resources
  - Side effects:
    - GET requests have "read-only" semantics (also known as "safe")
    - GET, PUT, and DELETE requests are idempotent (i.e., invoking a request N times produces the same final state as a single invocation)

REST APIs – Some characteristics (2/2)

- **Using HTTP message bodies as representations of the resources**
  - For example, in a reply to a GET request, a client may receive the state of the resource represented in JSON, XML, …
  - The chosen format can be specified in the HTTP headers

- **Using web links to express specific relations between resources**
  - For example: relationships “prev”, “next” and “last” may be used to explore a list of resources
  - The “rel” attribute must be used in the metadata of the hyperlinks

- **Using a HTTP parameter as an authentication token**
  - During the first step of a session, a client issues an authentication request (via a HTTP GET requests) and receives an authentication token (a text string) from the server
  - During the remainder of the session, the client can authenticate its requests by computing a specific hash (of the request + the token) and including this hash as a parameter of the request

REST APIs - Details

- **For further details, see the following references:**
  - https://en.wikipedia.org/wiki/Representational_state_transfer
  - **[recommended] https://www.restapitutorial.com**
  - [quick overview, in French] https://blog.nicolashachet.com/niveaux/confirme/larchitecture-rest-expliquee-en-5-regles/
  - Specification/list of possible values for link relations: https://www.iana.org/assignments/link-relations/link-relations.xml

- **Examples of “real life” REST APIs:**
  - Twitter: https://developer.twitter.com/en/docs/api-reference-index
  - LinkedIn: https://www.linkedin.com/developers/
  - Google Cloud: https://cloud.google.com/apis/docs/overview

Network security

Firewalls

- **A firewall aims at filtering network communications in order to provide improved security guarantees.**
  - Can be placed at the level of individual machines or at the level of a network (router).
- **A firewall is configured with rules based on whitelisting and/or blacklisting.**
- **Filtering rules can be defined/applied at different layers:**
  - Layer 2: MAC addresses
  - Layer 3: IP addresses
  - Layer 4: Transport protocol and ports, connection state
  - Layer 5: Application-level protocols and their specific criteria (e.g., Web URLs)
- **Filtering is usually performed on message headers (for cost efficiency) but can also be performed on message contents.**

VPN: Virtual Private Network

- **A VPN creates a secure tunnel between a network N1 and an external machine M (located in another network N2).**
- All (or most of) the network traffic of M is redirected through the router of N1.
- From the point of view of the hosts within N1, M also belongs to N1 and has an IP address that fits in N1's IP range.
- Security guarantees (authentication and confidentiality) are achieved via cryptographic techniques (not detailed here).
- The IP packets exchanged between M and N1's router are encapsulated in secure messages, either at layer 3 (IPSEC) or at layer 5 (SSL).

- **Main motivations and use cases for using of a VPN:**
  - Providing a machine connected to an untrusted network with secure (and convenient/simple/full) access to the internal resources of a organization's network
  - Controlling/monitoring the network traffic of a machine regardless of its current physical location
  - Protecting against attacks of a hostile network (confidentiality, authentication, integrity of the communications)
  - Making a machine appear as being located in another country

SSL/TLS (1/2)

- **"Secure Socket Layer" and "Transport-Level Security"**
  - Roughly speaking, two very similar protocols

- **Purpose: Providing security and confidentiality guarantees for application-level communications**
  - **Authentication** (at least for the server, optional for the client)
  - **Confidentiality** of the exchanged data
  - **Integrity** of the **messages contents**, and of the **message sequences**

- Conceptually implemented as **an intermediate layer between the transport layer (e.g., TCP sockets) and high-level application protocols (e.g., HTTP)**.

SSL/TLS (2/2)

- Relies on various cryptographic techniques (asymmetric and symmetric cryptography) and chains of trust.

- **Many application-level protocols have a an SSL/TLS-enabled variant.** For example:
  - HTTP (default port: TCP 80) → HTTPS (default port: TCP 443)
  - SMTP (default port: TCP 25) → SMTPS (default port: TCP 465)
  - IMAP (default port: TCP 143) → IMAPS (default port: TCP 993)

- **Typically implemented as a library** (with interface bindings available for most programming languages).
  - Some examples: OpenSSL, LibreSSL, BoringSSL
  - **A security-critical piece of code (many bugs/vulnerabilities discovered in the recent years).**

SSH

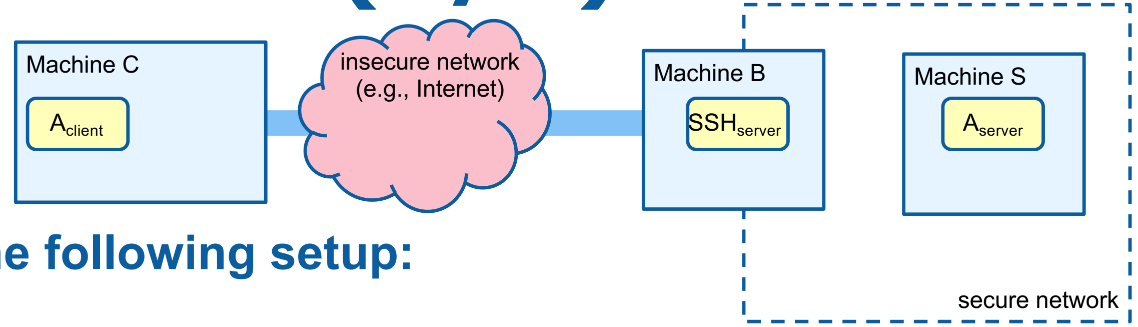
- **An application-level, client-server protocol for secure communications over a potentially insecure network.**
- **Provides the following guarantees:**
  - Confidentiality and integrity of communications
  - Client authentication
  - Server authentication
- Based on TCP (by default, server port 22). Not based on SSL/TLS
- **Main use cases:**
  - **Remote shell (interactive)**
  - Remote execution of commands/applications (possibly including graphical applications, but slow)
  - **SSH tunnel: encapsulation/forwarding of application-level protocols**
  - **SCP: copy of files to/from remote machine**
  - SFTP: FTP-like service
  - SSHFS: NFS-like service

SSH – Authentication

- **Client (user) authentication by the server**
  - By password
  - Or by asymmetric cryptography (preferred method)
    - Pair of cryptographic keys: public key + private key
    - The server stores copy of the client's public key
    - The private key always remains on the client side

- **Server authentication by the client**
  - Only by cryptographic techniques
  - At least, the client machine remembers the fingerprint of the server host key used during previous sessions and will check that the new fingerprint matches
  - The client can also be pre-configured with the public key of the server

SSH tunnel (1/7)



- **Motivating scenario**
  - **Let's assume that we have the following setup:**
    - Two machines: C and S
      - Machine C, hosting a client application A_{client}
      - Machine S, hosting a server application A_{server}
      - A_{client} and A_{server} use an arbitrary TCP-based protocol (e.g., HTTP, IMAP, FTP …)
    - C and S are interconnected via an insecure network (e.g., the Internet)
    - For security reasons, **the network that hosts machine S is only accessible from the outside through the SSH service running on machine B** (a.k.a. "SSH bastion")
    - The user (client) has a SSH account on machine B

- **How can we create a working and secure connection between A_{client} and A_{server}?**

SSH tunnel (2/7)

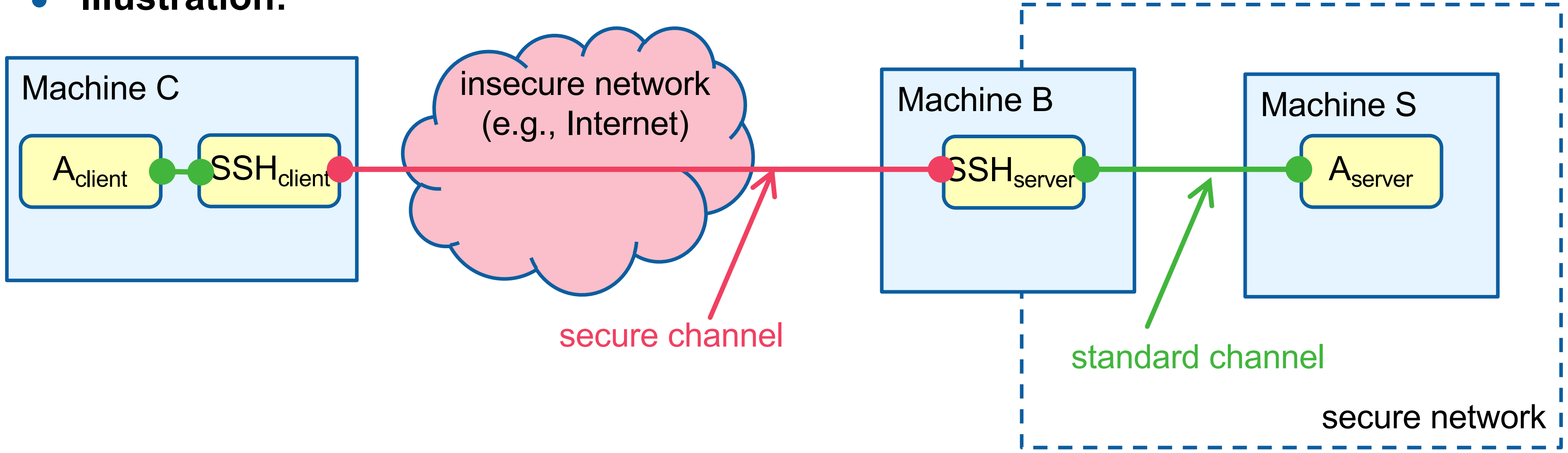
- **Solution: create a SSH tunnel, acting as a secure relay between A_{client} and A_{server}**
  - On machine C: SSH client will act as a proxy for A_{client}
  - On machine B: SSH server will connect to A_{server}

- **Configuration**
  - Step 1: Create the tunnel (from A_{client})
  - Step 2: Modify the configuration of A_{client} (server port and IP address)

- Note: a SSH server can support several tunnels simultaneously (from the same client and/or different clients)

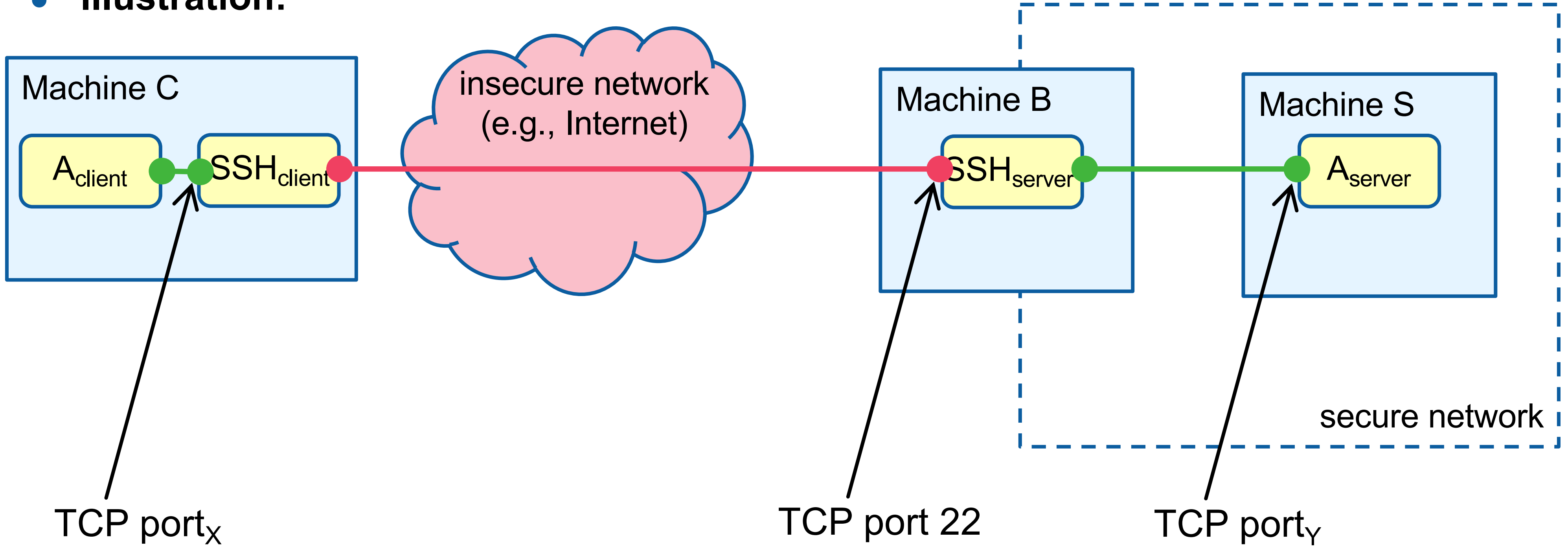
SSH tunnel (3/7)

- **Illustration:**



SSH tunnel (4/7)

- **Illustration:**

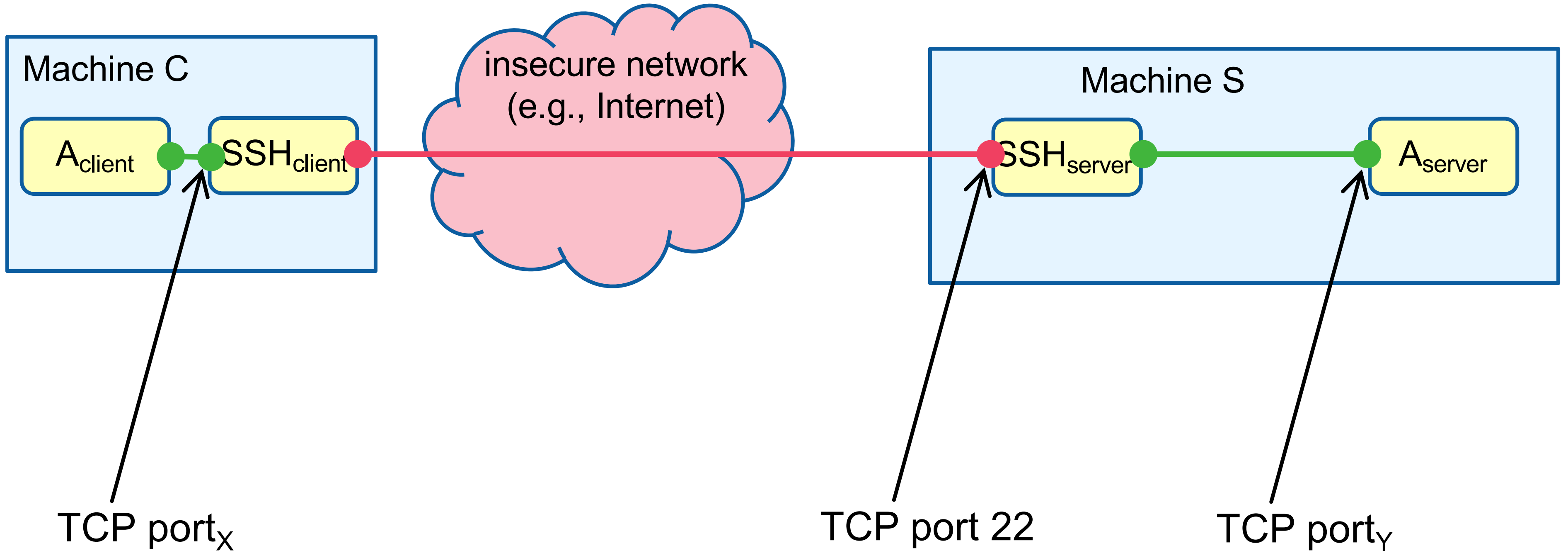


SSH command line for tunnel creation:

```
ssh -N –L localhost:portX:machineS:portY mylogin@machineB
```

SSH tunnel (5/7)

Variant : **if B and S are actually the same machine (with only the SSH service available from the outside):**



SSH command line for tunnel creation:

```
ssh –N –L localhost:portX:localhost:portY mylogin@machineS
```

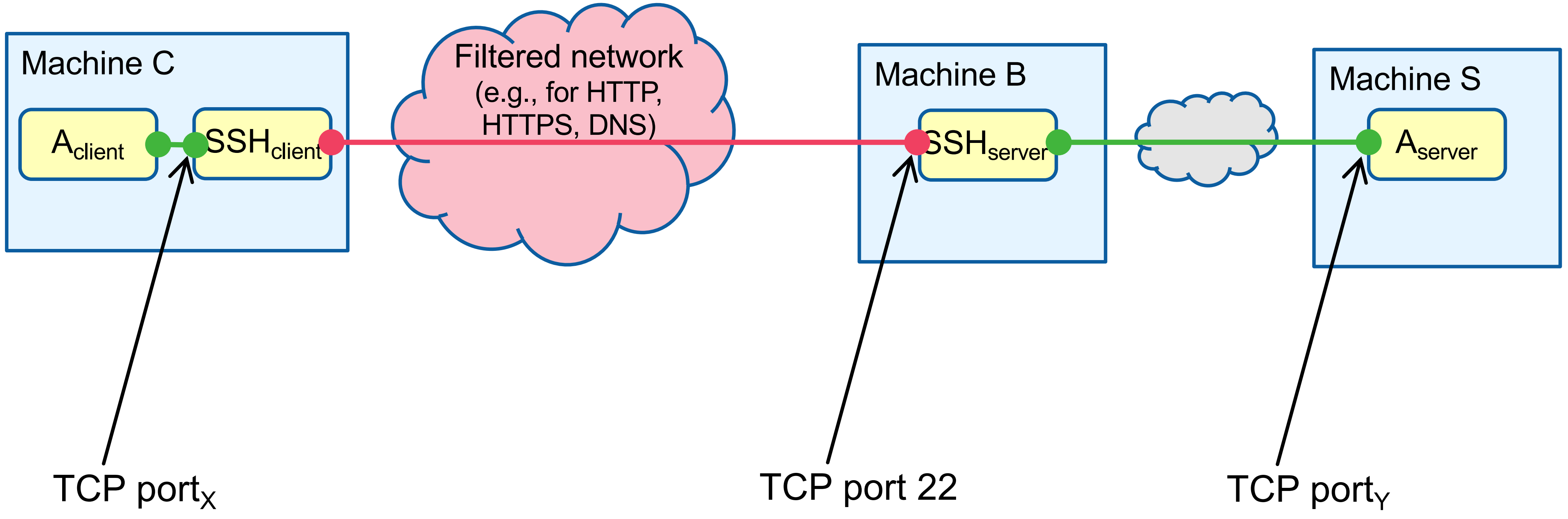
SSH tunnel (6/7)

- **Notes:**
  - **The configuration of client applications on machine C must be modified in order to contact the local SSH agent instead of the remote server.**
  - By default, the client application is not aware that it is interacting with a proxy.
    - This is flexible (no need to modify the client application).
    - However, **a distinct tunnel must be manually created for each TCP connection created from machine A (for the same application and/or different applications).**
  - **Another possibility is to make the client application aware that it is communicating through a proxy.** In that case, a new tunnel can be automatically created by the proxy upon each new outgoing connection request initiated by the client application(s).
    - The typical protocol used for the client – SSH proxy interactions is SOCKS.
    - Many client applications support SOCKS (e.g., Web browsers).

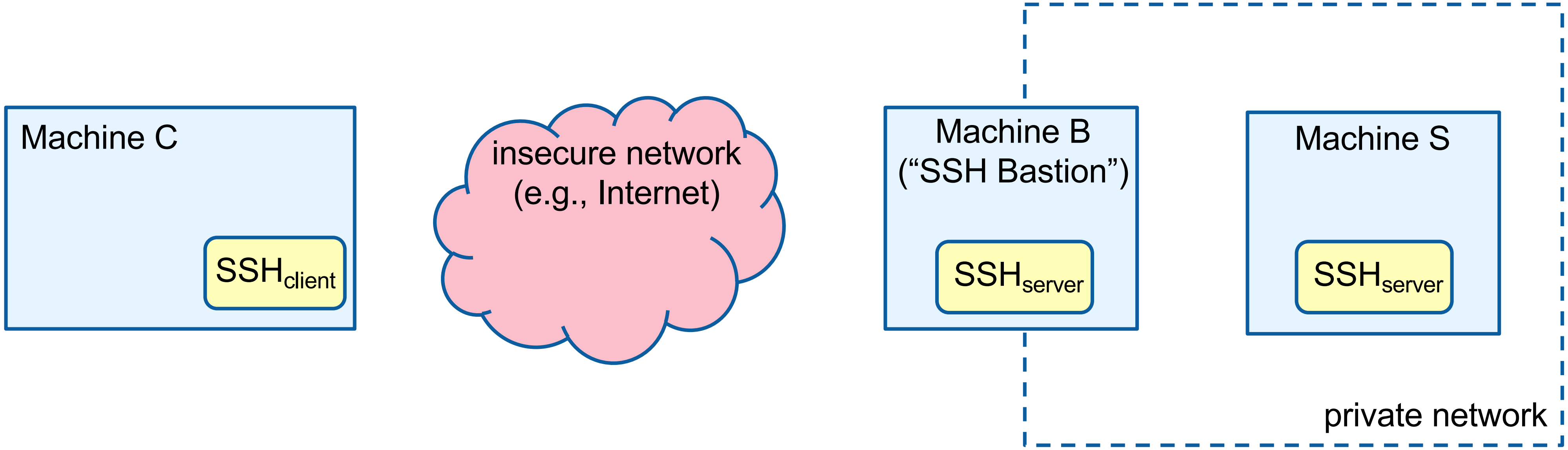
SSH tunnel (7/7)

Another variant: machine S can be outside of the network of machine B

- In that case, the main motivation is to overcome a filtered network (e.g., for Web browsing)
- In a typical setup for this use case, SSH_{client} appears as a SOCK proxy for A_{client}



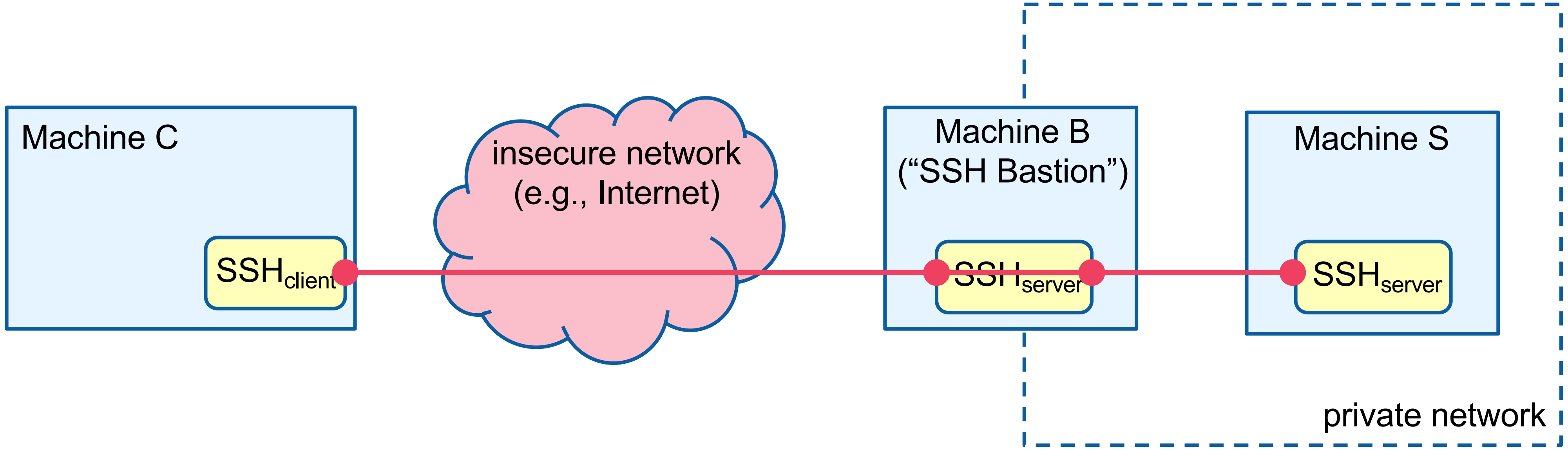
SSH jump host (1/3)



Another common scenario:

- A user on machine C has an account on Machine B and on Machine S
  - Note: Credentials (login, password/keys) may or may not be the same on both machines
- **The user wants to establish an SSH session with machine S**
- However, in this setup, only Machine B is reachable from the outside

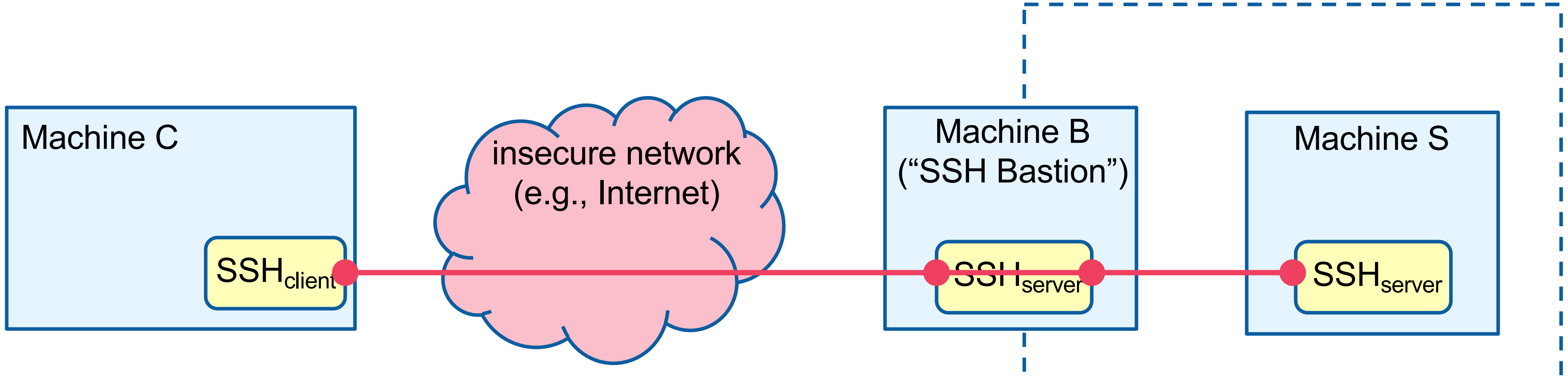
SSH jump host (2/3)



Solution:

- Use Machine B as a "jump host" (also known as "jump server"): Machine B acts as relay of SSH channel between C and S
- The private SSH keys never leave Machine C (Machine B cannot "steal" secrets)

SSH jump host (3/3)



SSH command line :

```
ssh –J mylogin1@machineB mylogin2@machineS
```

Can also be used for file transfers:

```
scp -o 'ProxyJump mylogin1@machineB' myfilename mylogin2@machineS:/my/directory
```

Can also be used with a sequence of multiple jump hosts:

```
ssh –J mylogin1@bastion1,mylogin2@bastion2 mylogin3@machineS
```