

# CSCI 210 Systems Programming

Week 15  
Advanced SSH functions

# Overview

- SSH overview
- Port forwarding via SSH, aka SSH tunneling
- Using SSH as dynamic proxy
- Tunneling graphics applications

# SSH basics

- SSH is a network protocol that allows secure remote access to systems over an unsecured network.
  - It has various implementations. OpenSSH is a commonly used one.
- It provides encryption and authentication mechanisms to protect data during transmission.
- SSH uses a client-server model, where the client connects to the server and establishes a secure channel for communication.
- By default, SSH operates on TCP port 22.

# Key concepts

- Public-key cryptography: SSH uses public-key cryptography to authenticate and establish secure connections.
- Public key: A key pair consists of a public key and a private key. The public key is stored on the server and the private key is kept securely by the client.
- Authentication: SSH supports different authentication methods, such as password-based authentication and public key authentication.
- SSH daemon: The SSH server process running on the remote system that listens for incoming SSH connections.
- SSH client: The program used to initiate an SSH connection to a remote server.

# Public key authentication

- Public key authentication is more secure and convenient than password-based authentication.
- Generate an SSH key pair using the `ssh-keygen` command.
- Add the public key (“id\_rsa.pub”) to the server’s “`~/.ssh/authorized_keys`” file and keep the private key (“`id_rsa`”) is kept on the client machine.
  - It is in the `.ssh` directory in user’s home
  - When connecting, the client signs a challenge from the server with its private key, and the server verifies it using the stored public key.

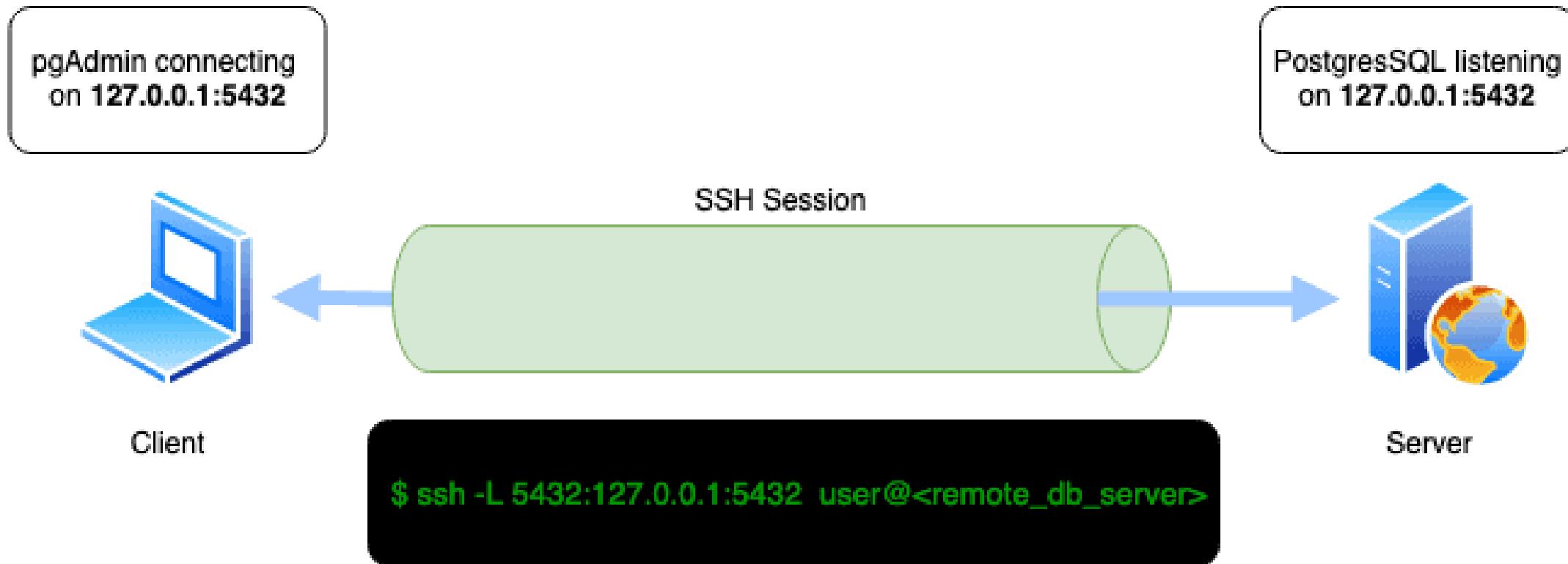
# SSH tunneling

- <https://goteleport.com/blog/ssh-tunneling-explained/>
- <https://iximiuz.com/en/posts/ssh-tunnels/>
- SSH tunneling is a way to transport arbitrary data with a dedicated data stream (tunnel) inside an existing SSH session.
- This can be achieved in various ways
  - local port forwarding,
  - remote port forwarding,
  - dynamic port forwarding

# Local port forwarding

- When local port forwarding is used, OpenSSH creates a separate tunnel inside the SSH connection that forwards network traffic from the local port to the remote server's port.
- In OpenSSH, this tunneling feature can be used by supplying -L flag.
- Internally, SSH allocates a socket listener on the client on the given port. When a connection is made to this port, the connection is forwarded over the existing SSH channel over to the remote server's port.

# Local port forwarding example

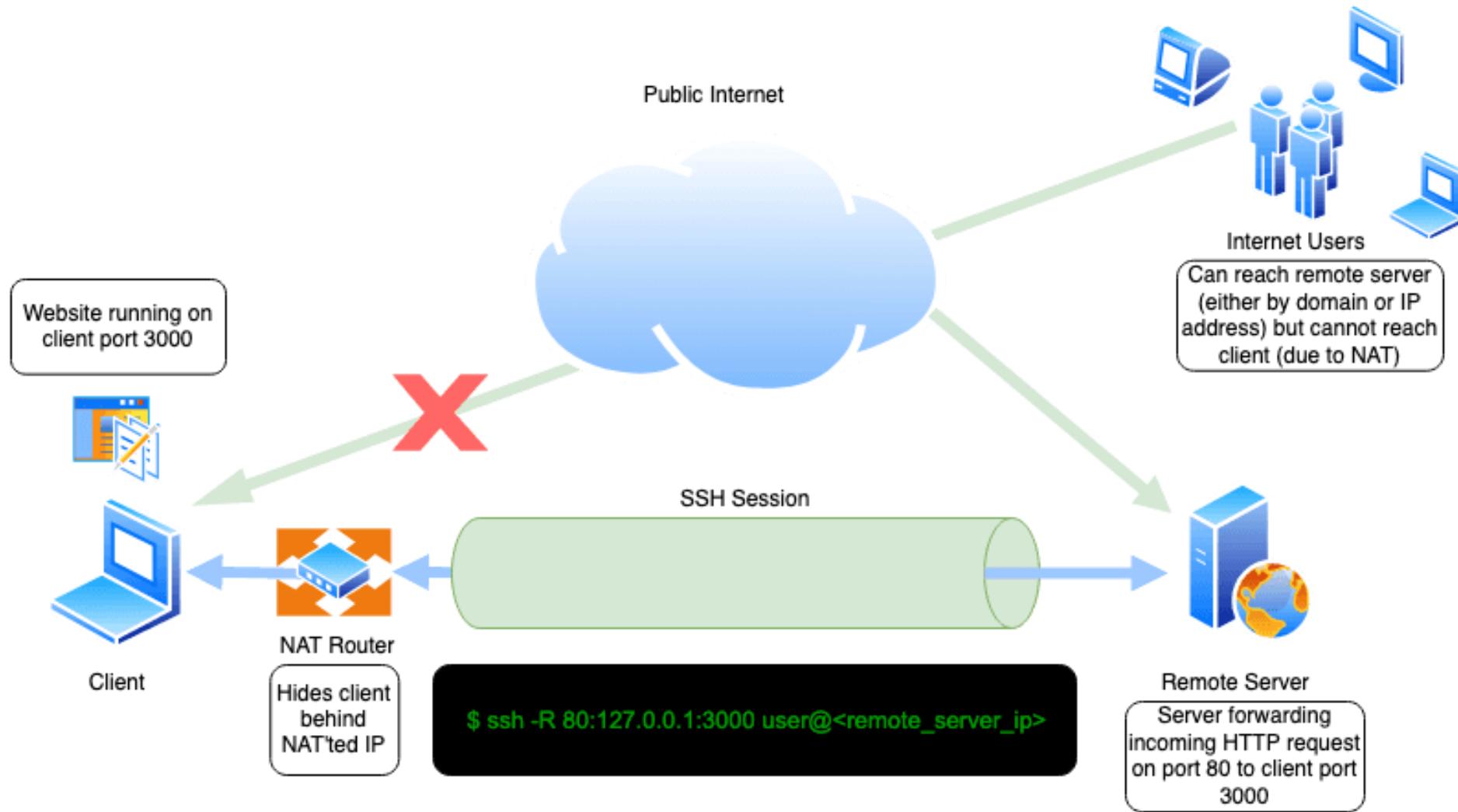


Note: By default, an interactive session is created for you when you command local port forwarding. To prevent interactive sessions, you can use the `-N` flag that tells SSH to not to execute remote commands.

# Remote port forwarding (reverse tunneling)

- Remote port forwarding redirects the remote server's port to the localhost's port.
- When remote port forwarding is used, at first, the client connects to the server with SSH. Then, SSH creates a separate tunnel inside the existing SSH session that redirects incoming traffic in the remote port to localhost (where SSH connection was created).
- Remote port forwarding can be achieved in OpenSSH by using -R flag.
  - Internally, SSH allocates a socket listener on the remote server on the given port. When a connection is made to this port, the connection is forwarded over the existing SSH channel over to the local client's port.

# Reverse tunneling use case scenario

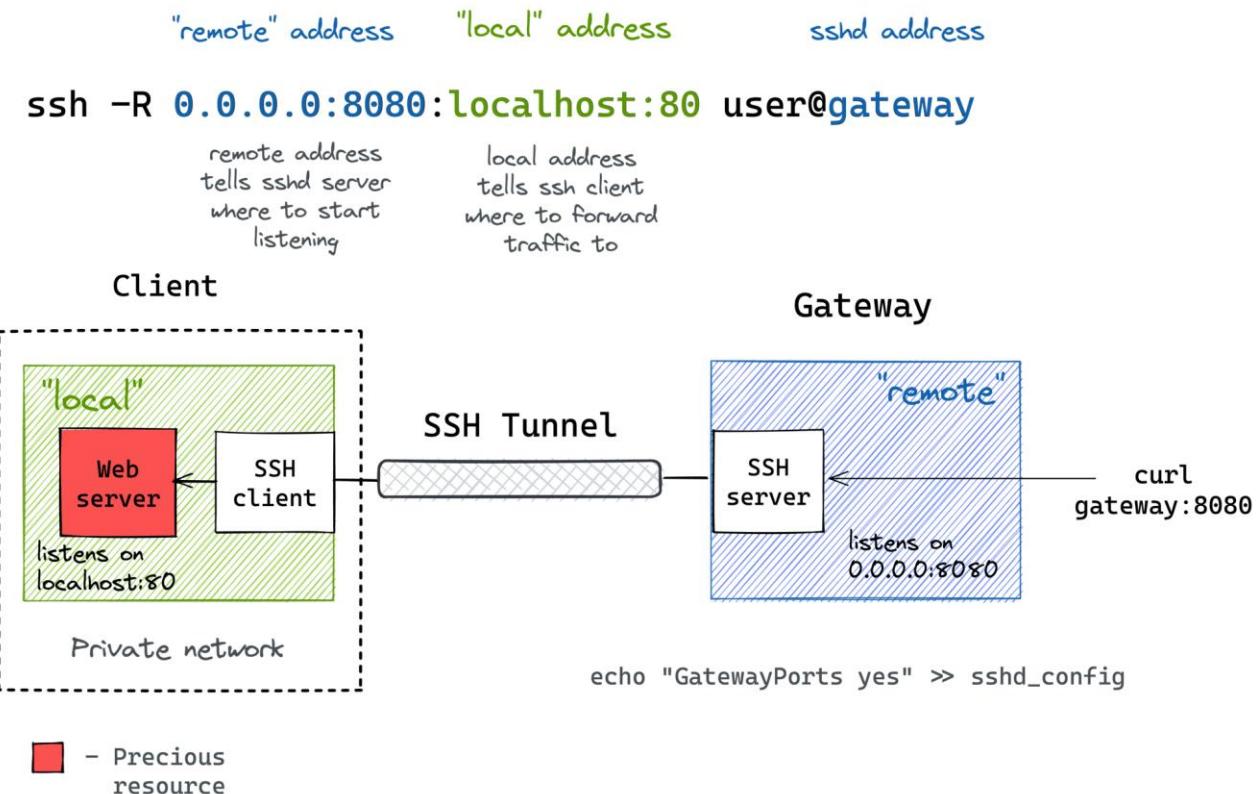
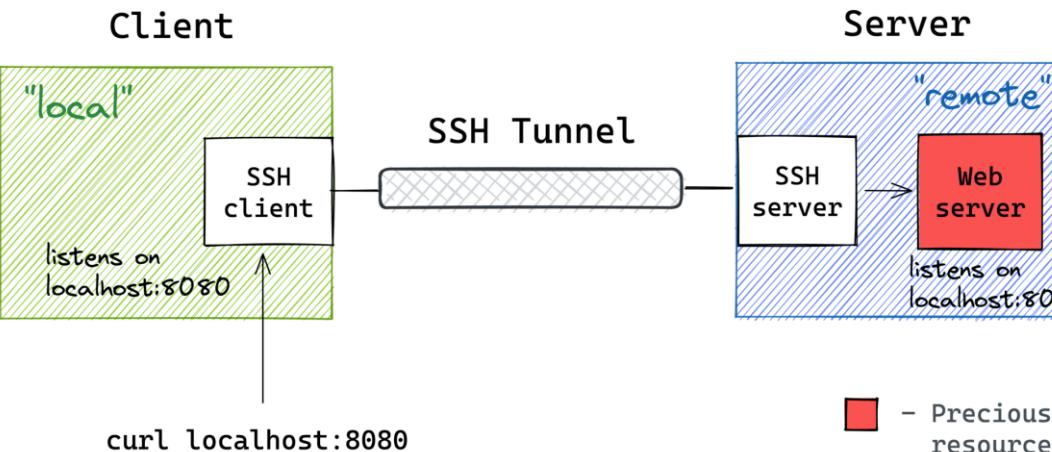


# Local versus Reverse port address syntax

- The mnemonics are "ssh -L local:remote" and "ssh -R remote:local" and it's always the left-hand side that opens a new port.

Short form  
ssh -L 8080      "local" address      "remote" address      sshd address  
ssh -L localhost:8080:localhost:80 user@server      local address tells ssh client where to start listening      remote address tells sshd server where to forward traffic to

Long form  
ssh -L localhost:8080:localhost:80 user@server



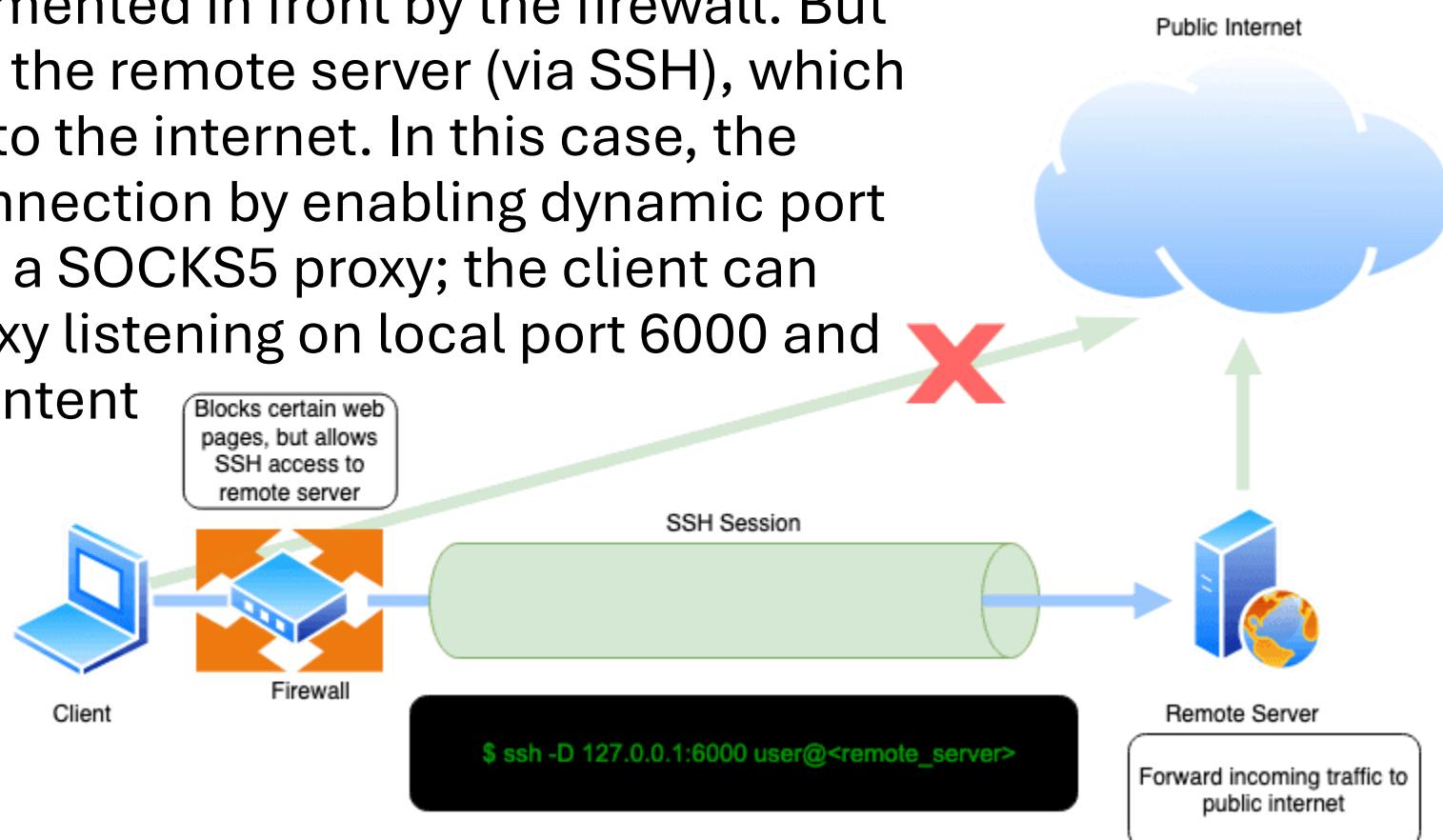
# Dynamic port forwarding

- Both local and remote port forwarding require defining a local and remote port.
- What if the ports are unknown beforehand or if you want to relay traffic to an arbitrary destination?
  - Also known as dynamic tunneling, or SSH SOCKS5 proxy, dynamic port forwarding allows you to specify a connect port that will forward every incoming traffic to the remote server dynamically.
- Dynamic port forwarding turns your SSH client into a SOCKS5 proxy server. SOCKS is an old but widely used protocol for programs to request outbound connections through a proxy server.

# Dynamic port forwarding use case

- Bypassing a firewall:

- Imagine that the client cannot access certain web pages due to content filtering implemented in front by the firewall. But the client can connect to the remote server (via SSH), which has unrestricted access to the internet. In this case, the client can initiate ssh connection by enabling dynamic port forwarding, thus creating a SOCKS5 proxy; the client can point web browser to proxy listening on local port 6000 and access restricted web content



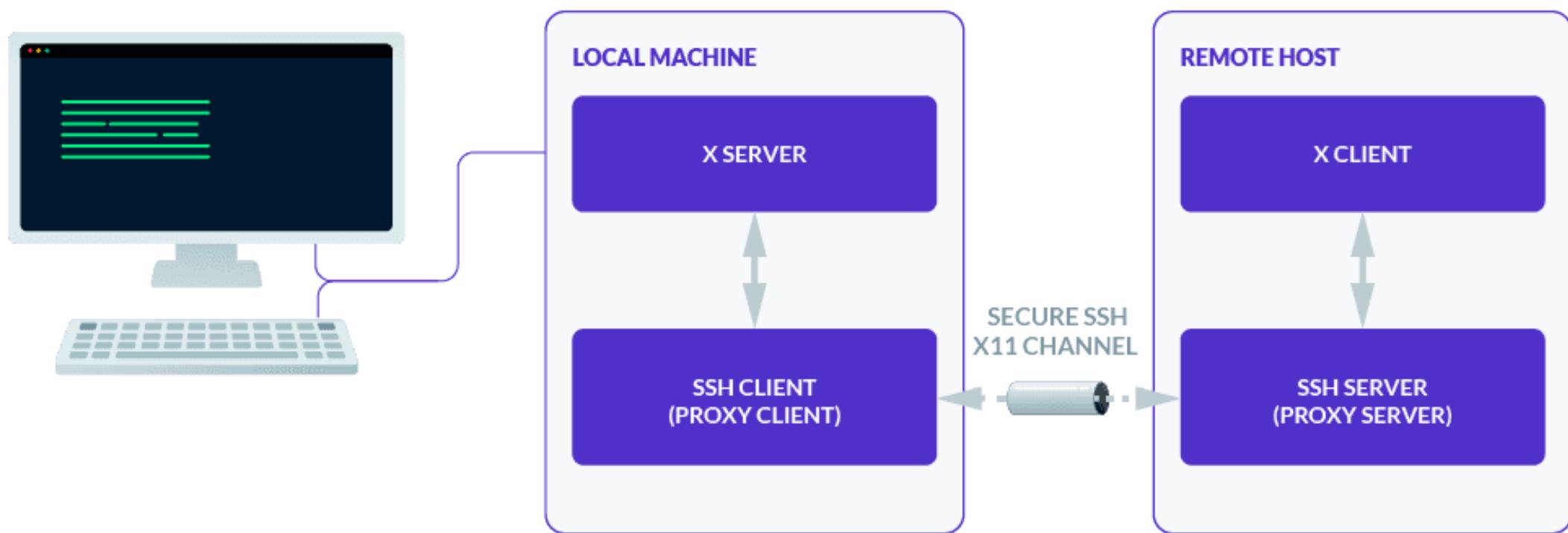
# X11 forwarding via ssh -X

- <https://goteleport.com/blog/x11-forwarding/>

# X11

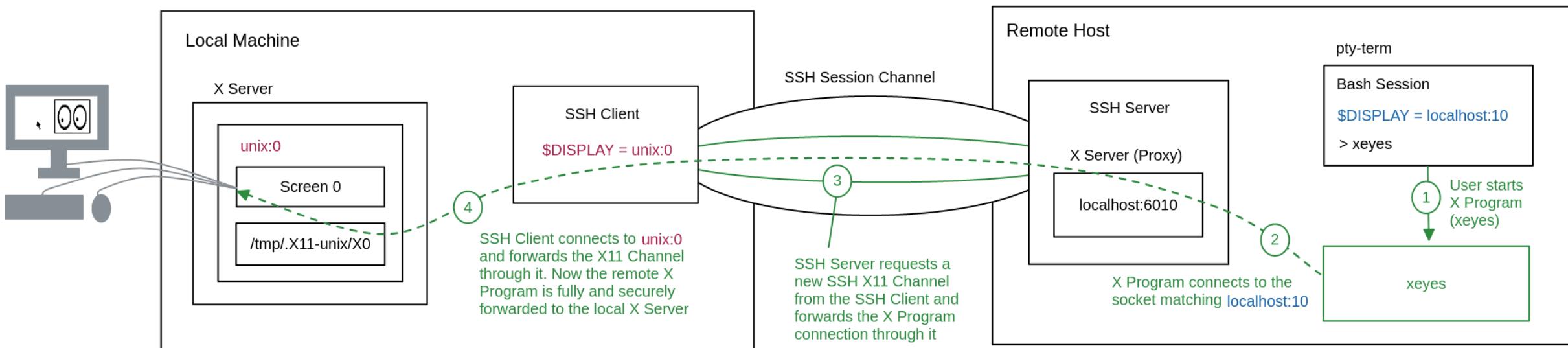
- X11 uses a client-server model,
  - where an X Server is a program on a machine which manages access to graphical displays and input devices (monitors, mice, keyboards, etc.),
  - and an X Client is a program which handles graphical data.
- With this model, an X Client application can form a connection to an X Server to communicate with the X Server's devices through graphical primitives.
- This client-server terminology can be confusing in remote scenarios, so remember that in most situations an X Server runs on the user's machine and the X Client runs on the remote machine.

# X11 forwarding via ssh



# X11 forwarding example

## X11 Forwarding an X Program



# Security concerns on ssh tunneling

- The power of SSH tunneling is also often misused by malicious users.
- Hiding malicious traffic within an SSH tunnel is a common classic way to go undetected inside a network.
- Example: Android malware used SSH tunnel to access corporate network
  - <https://www.bleepingcomputer.com/news/security/milkydoor-android-malware-uses-ssh-tunnels-to-access-secure-corporate-networks/>
- Example: Misuse of SSH tunnels to send spam
  - <https://www.rackaid.com/blog/spam-ssh-tunnel/>