Sentinel

Detecting network attacks

DEFENDING OUR DIGITAL WAY OF LIFE

Lesson objectives

→ Learn about **network and port scans** in practice

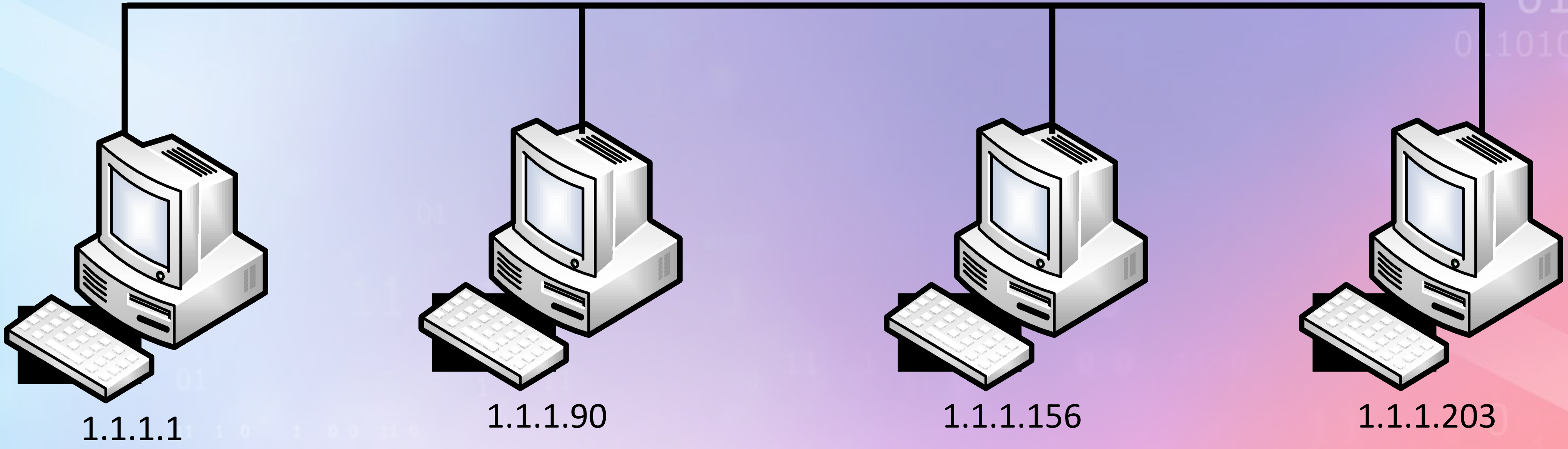
→ Think about **NIDS decision making**

→ Understand **Time-based sliding window**

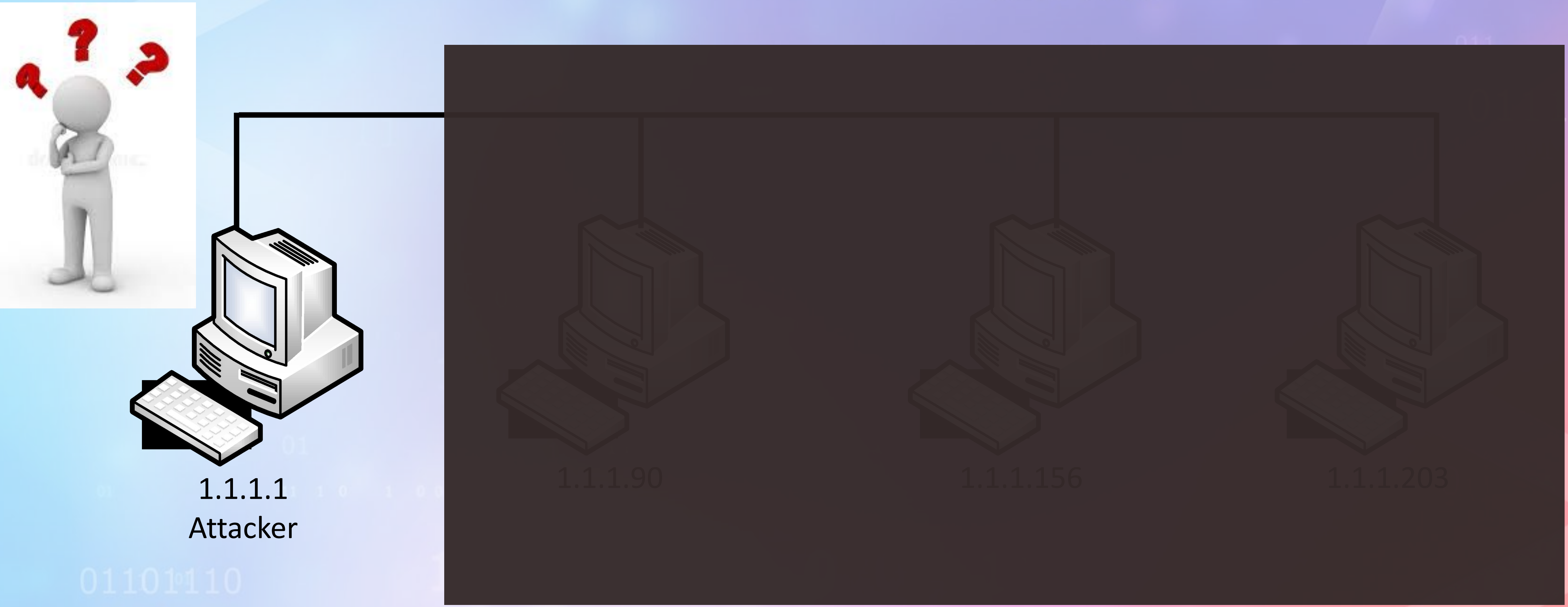
What is a network scan?

- What is the purpose?
- How would you implement it?

Subnet 1.1.1.1/24



Subnet 1.1.1.1/24 - Attacker POV

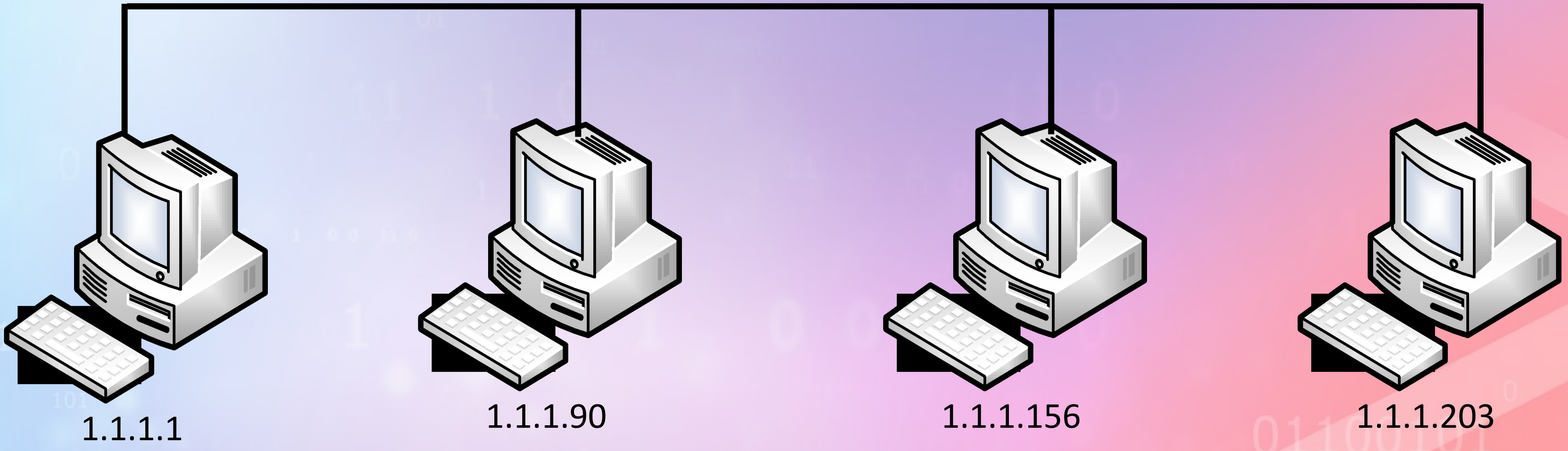


Network scan

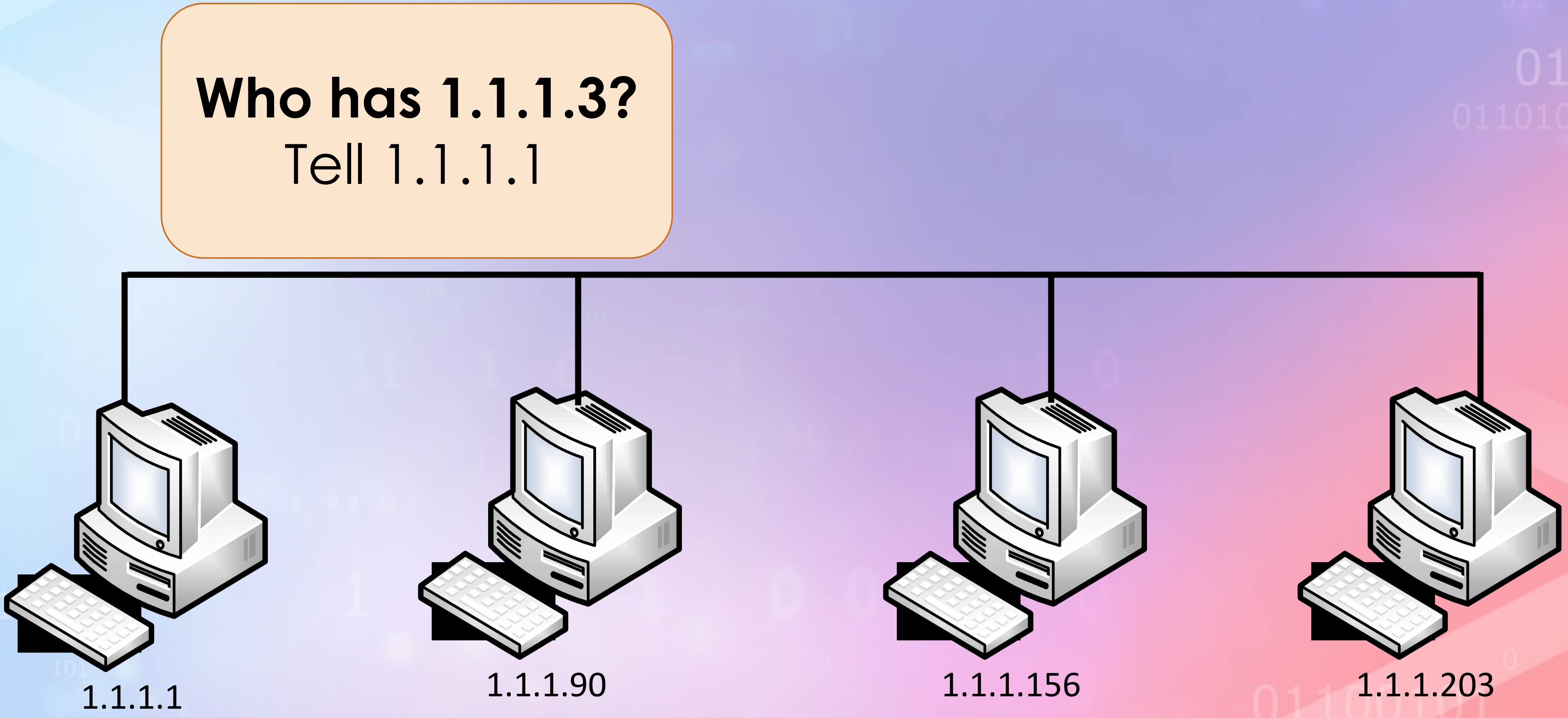
- We’re talking about internal network, or LAN segment
- Basic idea
  - Enumerate all possible IPs
    - ❓What are all possible IPs for my subnet if my IP is **192.168.213.1/24?**
  - For each, send a packet
    - Which that IP should respond to
    - Revealing itself

ARP scan

Who has 1.1.1.2?
Tell 1.1.1.1

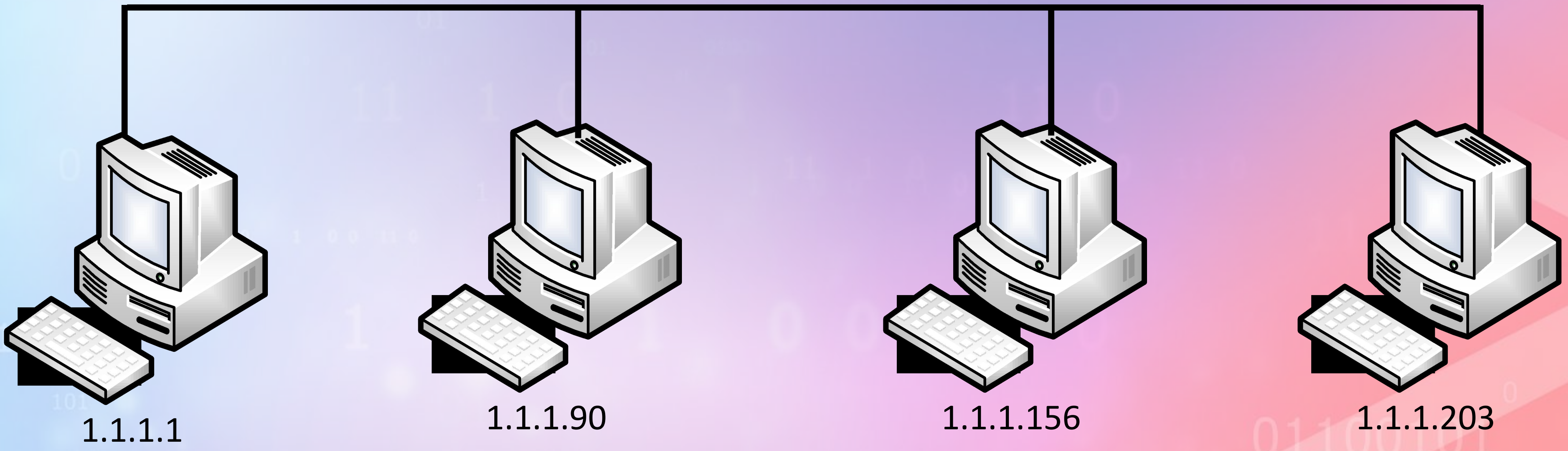


ARP scan

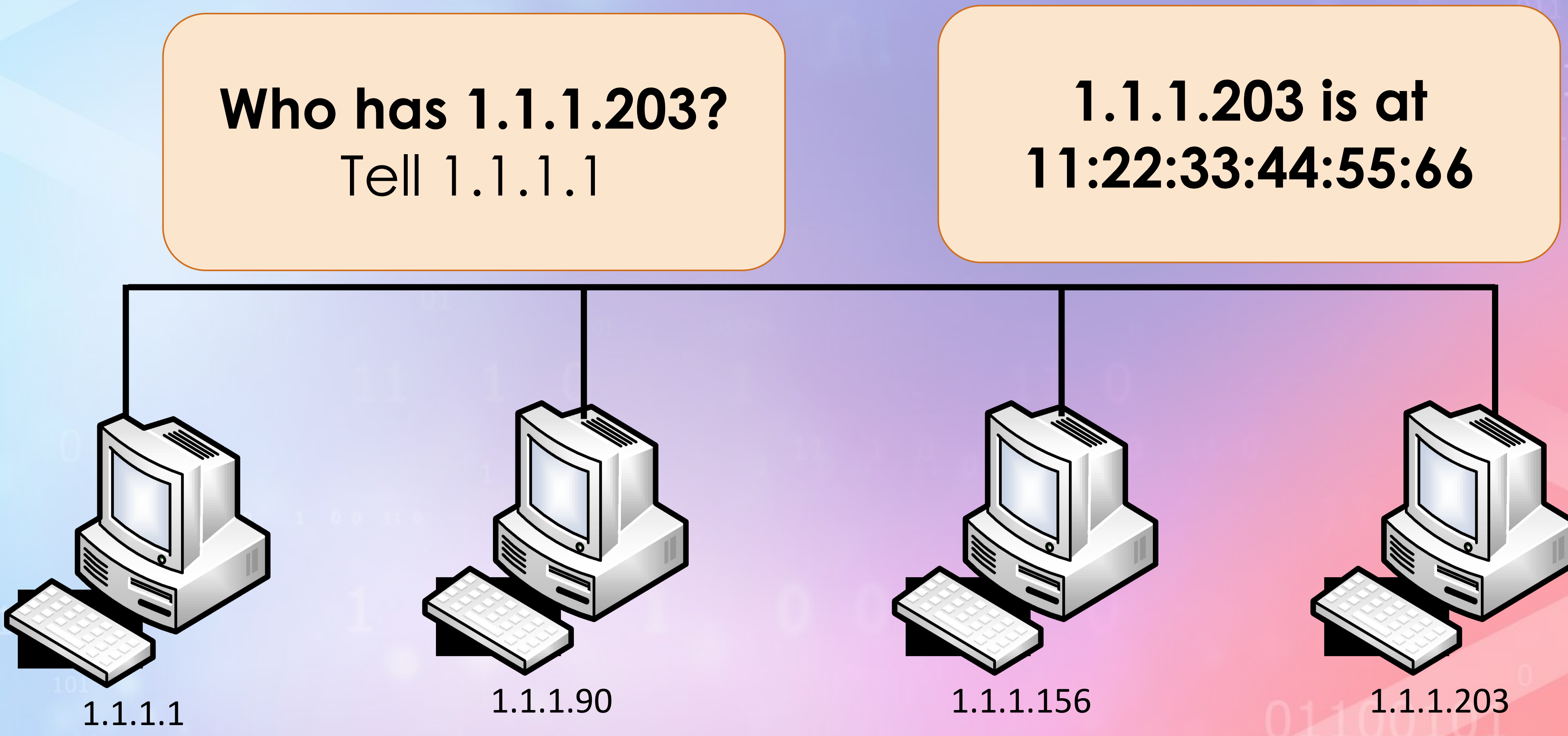

Who has 1.1.1.3?
Tell 1.1.1.1

ARP scan

Who has 1.1.1.202?
Tell 1.1.1.1



ARP scan



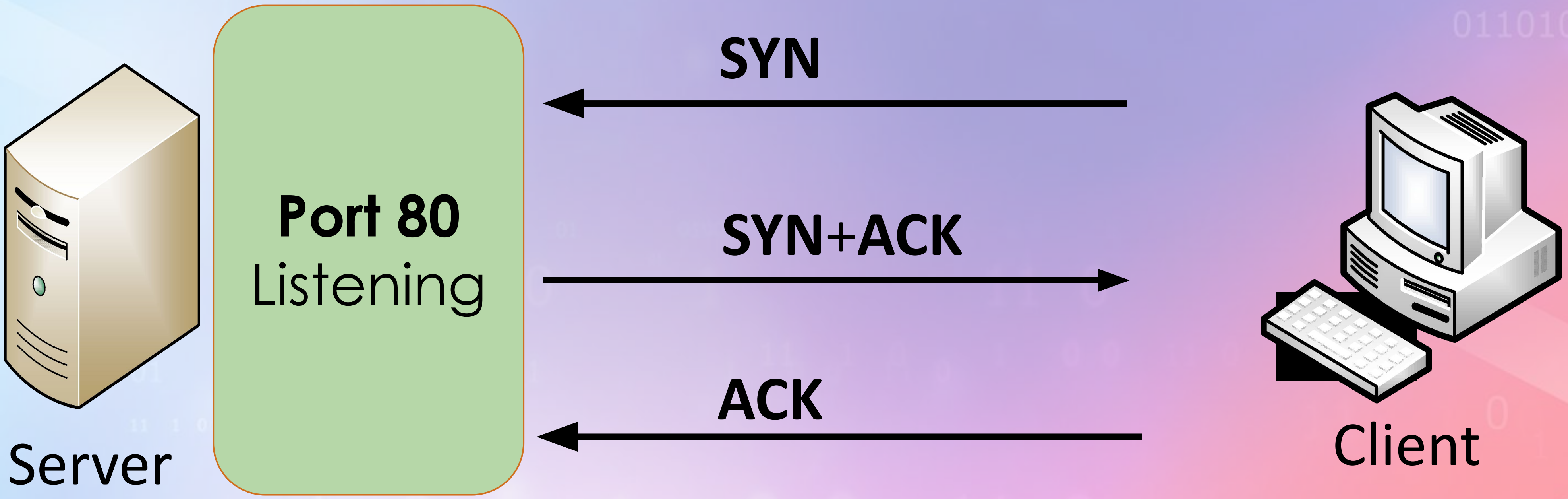
Who has 1.1.1.203?
Tell 1.1.1.1

1.1.1.203 is at 11:22:33:44:55:66

What is a port scan?

- What is the purpose?
- How would you implement it?

TCP Handshake



Port 80
Listening

SYN

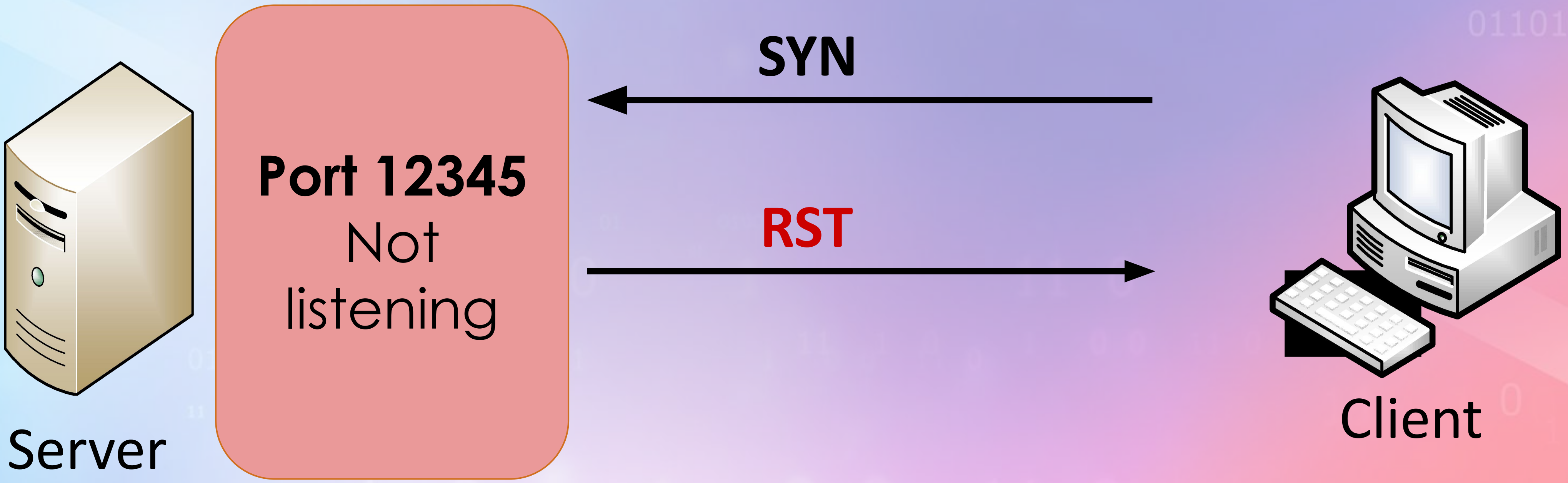
SYN+ACK

ACK

Server

Client

TCP Handshake



Server

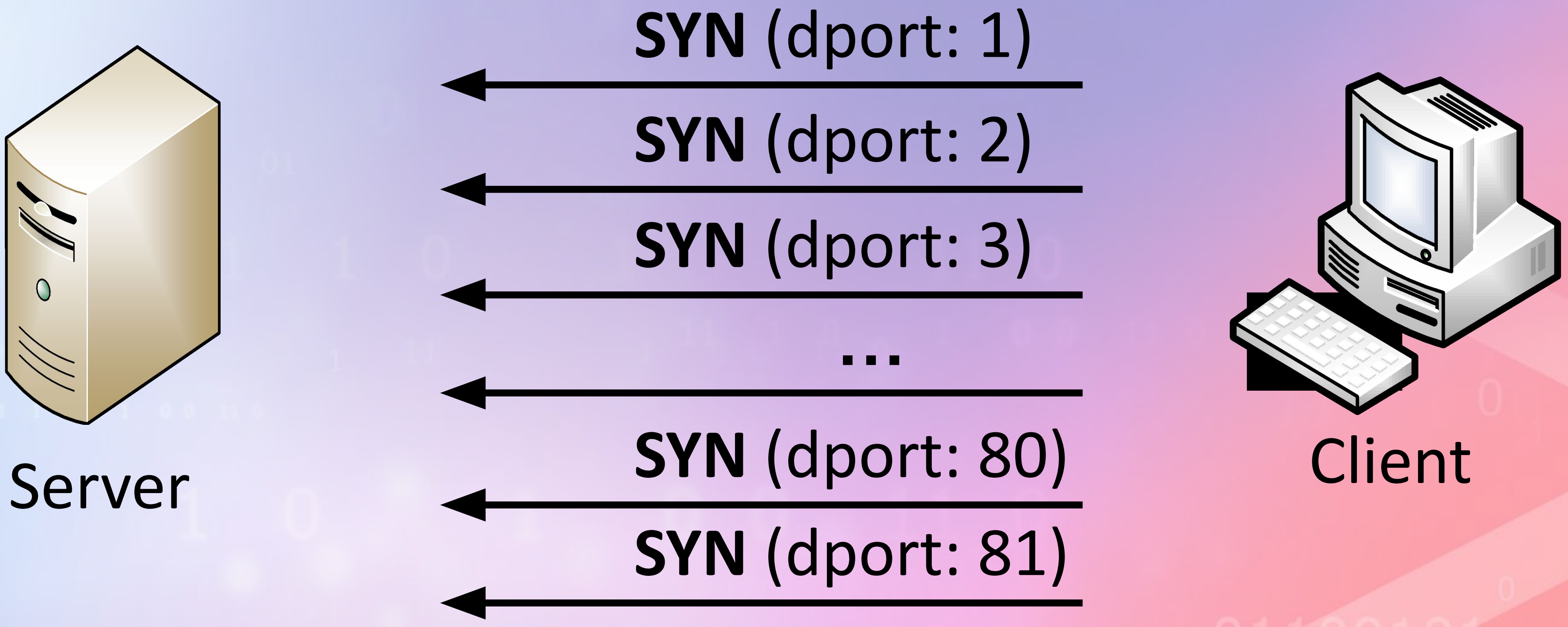
Port 12345
Not listening

SYN

RST

Client

TCP Port scan



Server

SYN (dport: 1)

SYN (dport: 2)

SYN (dport: 3)

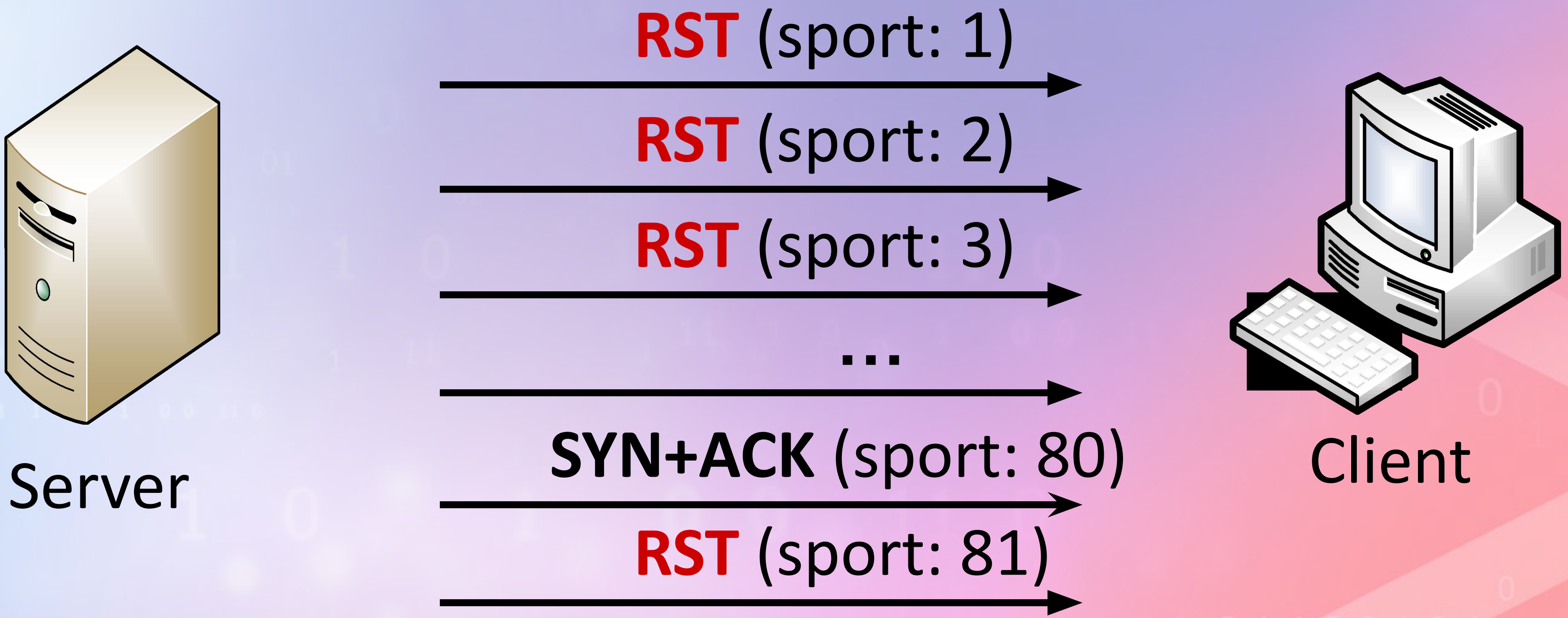
…

SYN (dport: 80)

SYN (dport: 81)

Client

TCP Port scan



How could we detect these scans?

Scan detection decision making

- So far, our detection was
  - If we see **a malicious packet**
- But here, it's different
  - The indication of malice is in the **repetition of specific packets**

How many packets are too many?

- Our requirement is to only alert malicious behaviour
- What if a client connects to different network shares really fast?
  - We would also see many ARP requests
  - But.. less than in an ARP scan
  - This shouldn’t trigger an alert
- If we set a low barrier
  - We could get many false positives (detecting normal use as malicious)
- If we set a high barrier
  - We could get many false negatives (missed detection of malicious activity)

How many packets are too many?

- Requires fine tuning in according to
  - Real, modern networks
  - Real, modern attack techniques
- During the workshop, you need to tune with the malicious/non-malicious trigger
  - As a general guideline to malicious thresholds

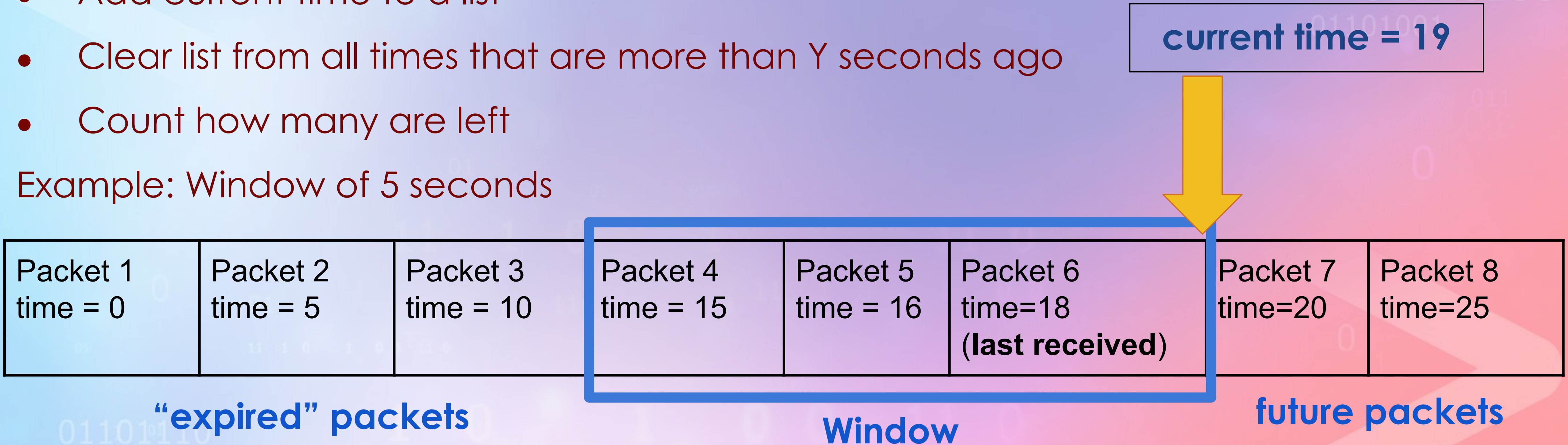
The precise requirement 📜

- Alert whenever
- There are more than X packets
- In the last Y seconds
- That match a specific condition

The algorithm - Sliding window

- If a packet arrives that matches the condition
- Add current time to a list
- Clear list from all times that are more than Y seconds ago
- Count how many are left

Example: Window of 5 seconds



current time = 19

Packet 1 time = 0	Packet 2 time = 5	Packet 3 time = 10	Packet 4 time = 15	Packet 5 time = 16	Packet 6 time=18 (**last received**)	Packet 7 time=20	Packet 8 time=25

"expired" packets

Window

future packets

Example - ARP scan

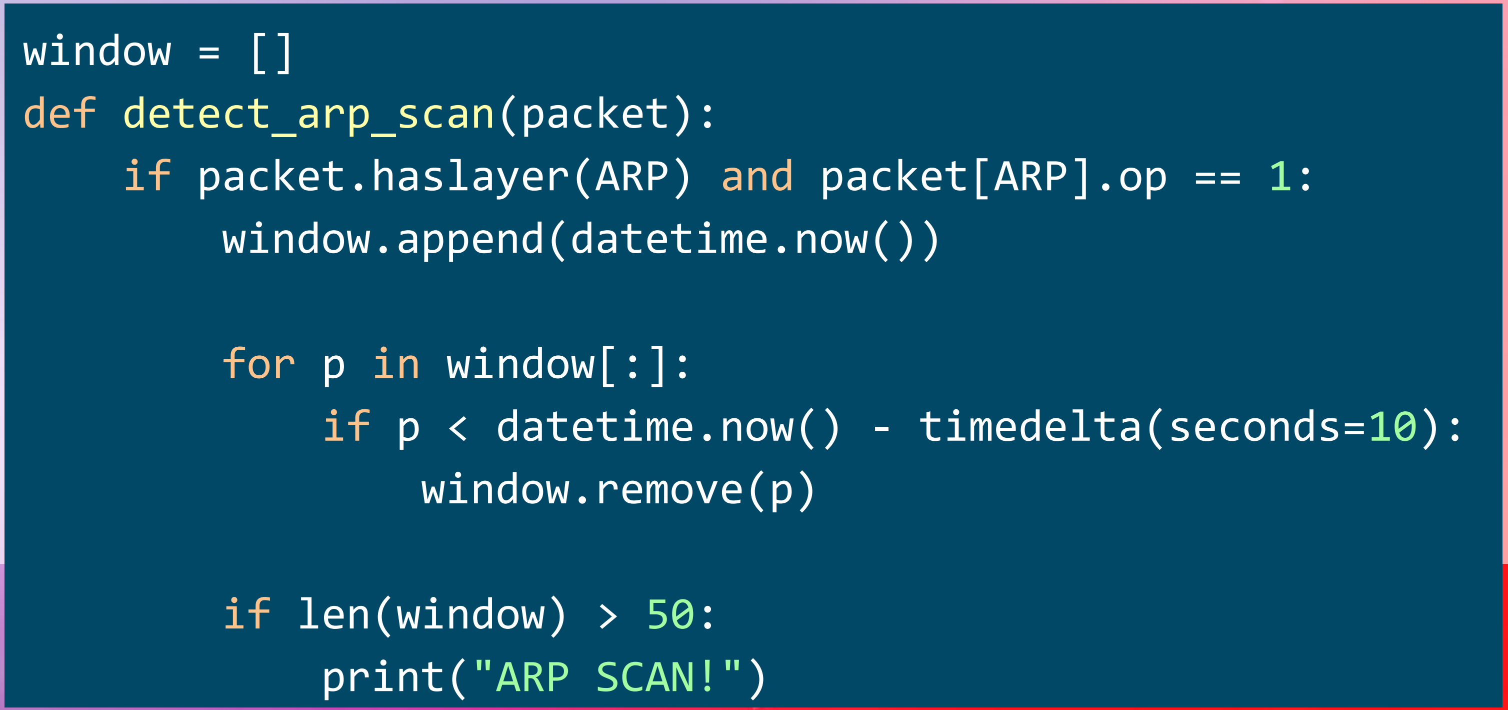
Window: 10 seconds

Threshold: 50 packets

```python
from datetime import datetime, timedelta
window = []

def detect_arp_scan(packet):
    if packet.haslayer(ARP) and packet[ARP].op == 1:
        window.append(datetime.now())

        for p in window[:]:
            if p < datetime.now() - timedelta(seconds=10):
                window.remove(p)

        if len(window) > 50:
            print("ARP SCAN!")

sniff(filter='arp', prn=detect_arp_scan)
```

What’s the problem?

- This code checks for threshold of total ARP requests
  - From any source!
- But we want to detect
- Whenever **one source** emits many packets



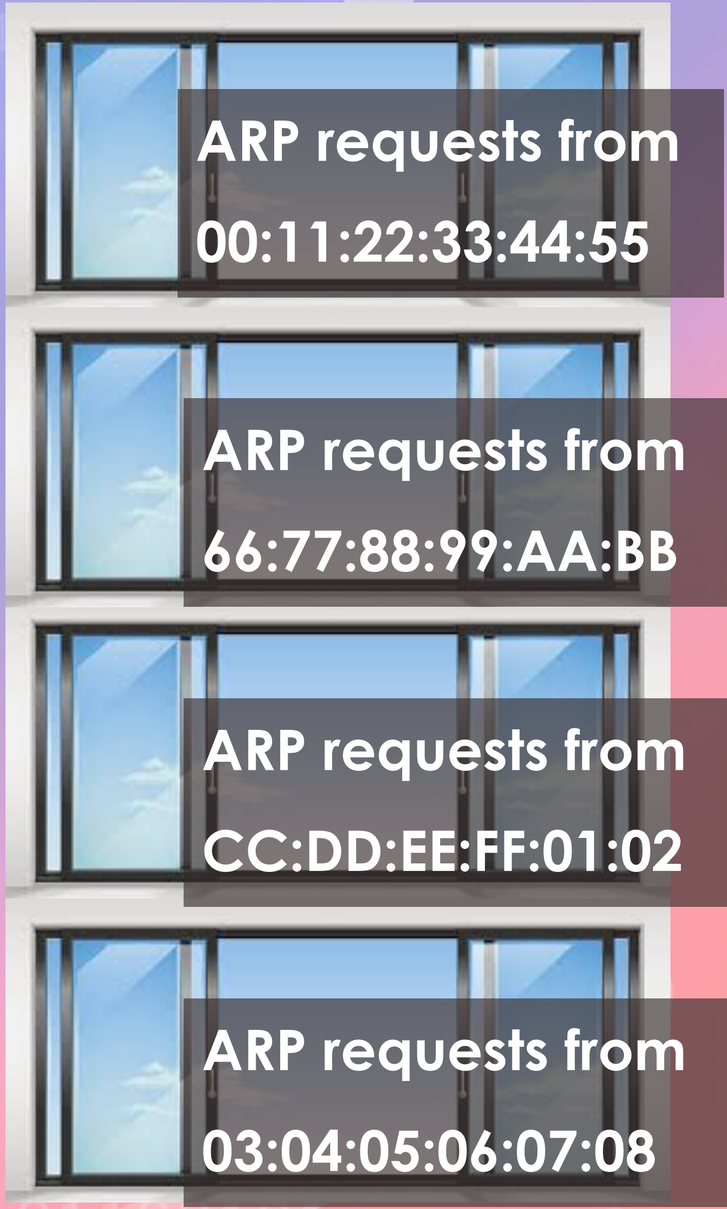
```
window = []
def detect_arp_scan(packet):
    if packet.haslayer(ARP) and packet[ARP].op == 1:
        window.append(datetime.now())

        for p in window[:]:
            if p < datetime.now() - timedelta(seconds=10):
                window.remove(p)

        if len(window) > 50:
            print("ARP SCAN!")
```

What’s a source?

- An identifier of an entity in the network
- Depending on the scope
  - LAN or Internet
- Could be an IP address or a MAC address

Solution?

- Maintain one window per source
- Required to define what is the source
  - In case of ARP scan - it’s the sender’s MAC
- In Python?
  - This sound like a job for a dict



ARP requests from 00:11:22:33:44:55

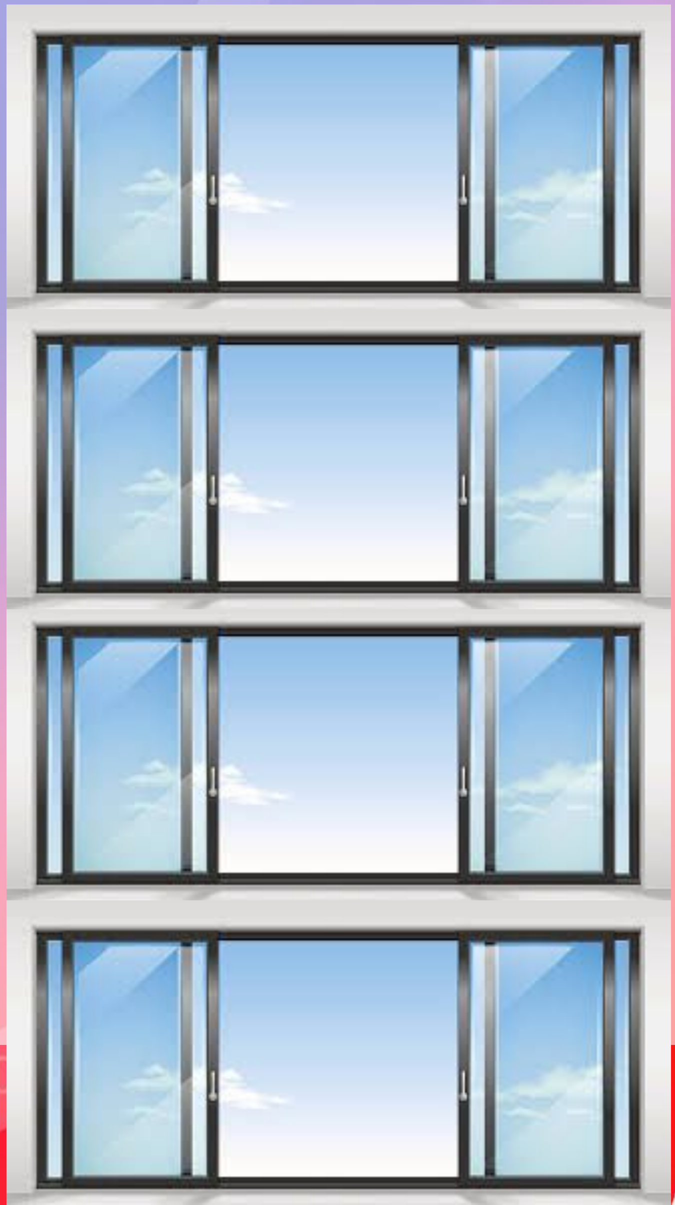
ARP requests from 66:77:88:99:AA:BB

ARP requests from CC:DD:EE:FF:01:02

ARP requests from 03:04:05:06:07:08

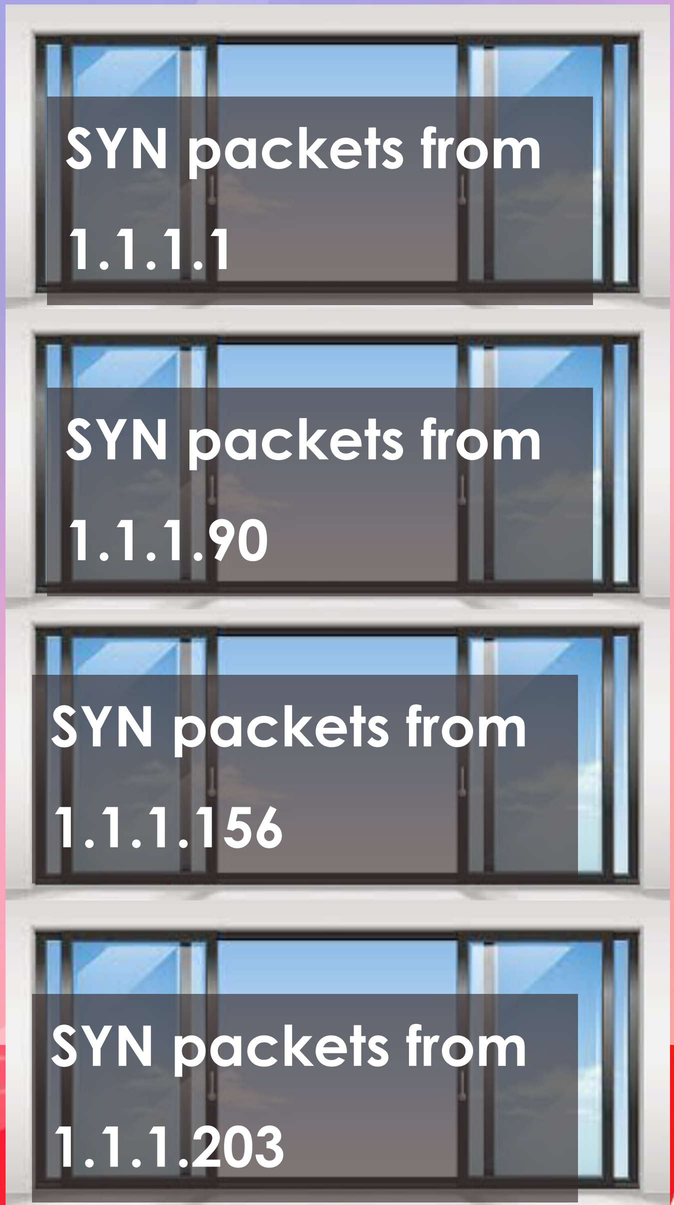
Example 2 - TCP port scan

- How do we detect a TCP port scan
- With our sliding window strategy?



Solution #1

- Problem?
- A port scan is defined as having a target
- These groups of SYN packets could have different destinations
- e.g. the same window could have at same time both:
  - A syn packet from 1.1.1.1 to 1.2.3.4
  - A syn packet from 1.1.1.1 to 5.6.7.8



SYN packets from 1.1.1.1

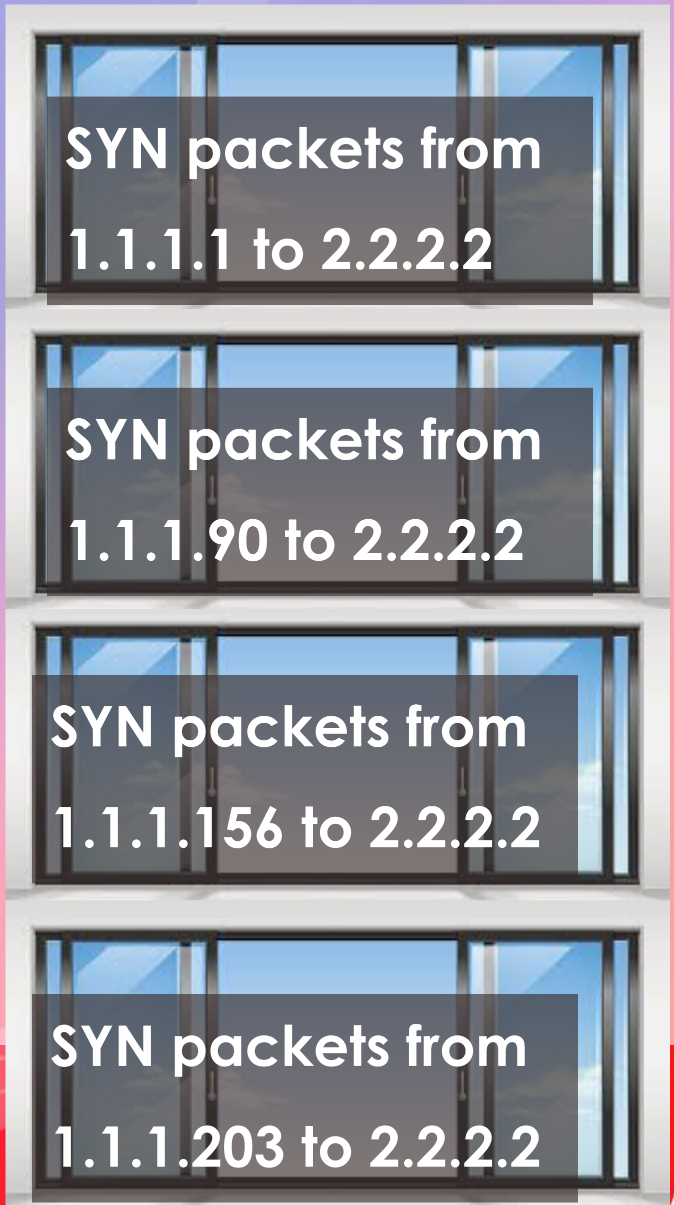
SYN packets from 1.1.1.90

SYN packets from 1.1.1.156

SYN packets from 1.1.1.203

Solution #2

- Problem?
- In a port scan, we expect each SYN to have different dst port
- What if a client is just crawling a website?
  - Initiating many HTTP connections
  - But it’s not a port scan
- e.g. the same window could have at same time both:
  - A syn packet from 1.1.1.1 to 2.2.2.2 (port 80)
  - A syn packet from 1.1.1.1 to 2.2.2.2 (port 80)



SYN packets from 1.1.1.1 to 2.2.2.2

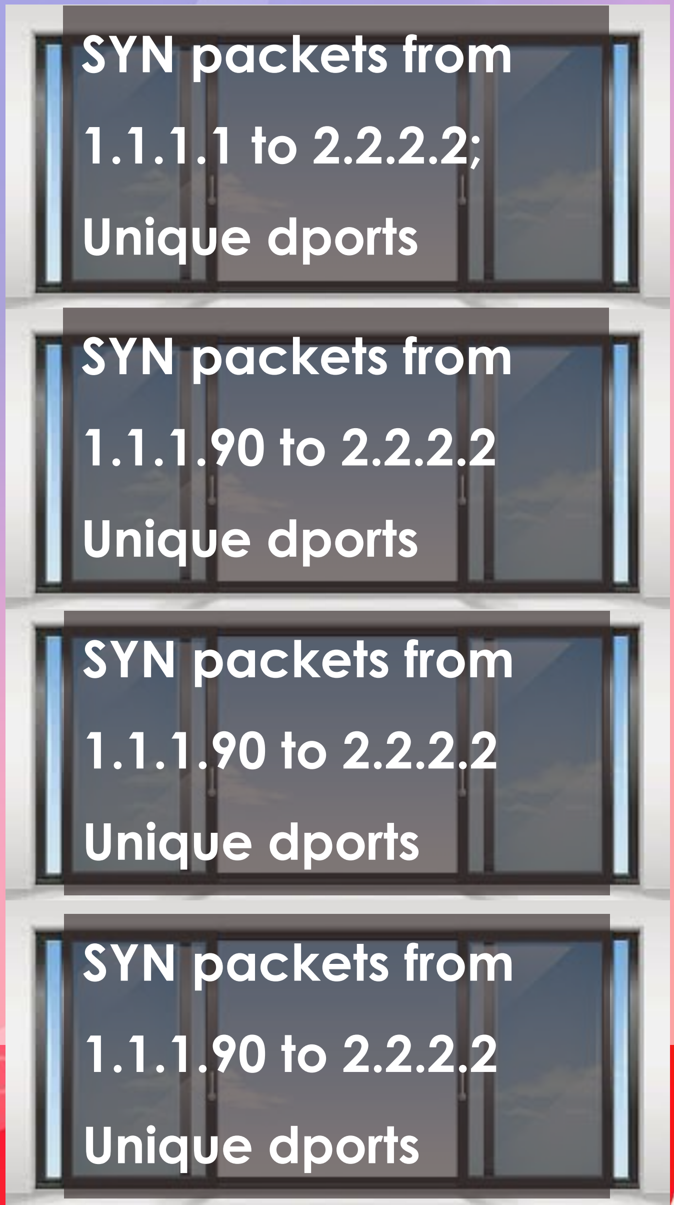
SYN packets from 1.1.1.90 to 2.2.2.2

SYN packets from 1.1.1.156 to 2.2.2.2

SYN packets from 1.1.1.203 to 2.2.2.2

Solution #3

- Generally, we make sure the window only has unique items
- Specifically for port scan
  - Uniqueness is determined by destination port
  - So we can’t have in the window 2 SYNs to same port
  - We would count any amount of SYNs to same port as one
- To summarize, we track
- “To how many ports of IP X have IP Y sent SYN packets to”



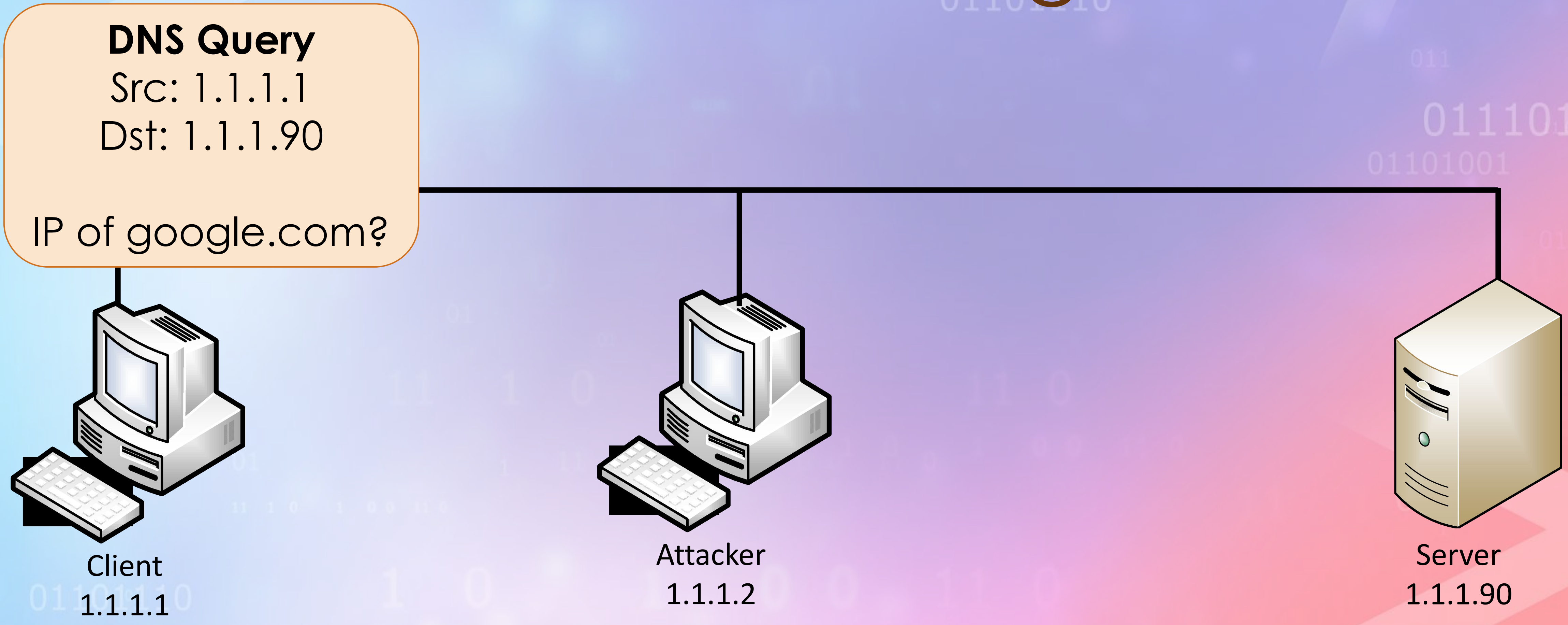
SYN packets from 1.1.1.1 to 2.2.2.2; Unique dports

SYN packets from 1.1.1.90 to 2.2.2.2 Unique dports

SYN packets from 1.1.1.90 to 2.2.2.2 Unique dports

SYN packets from 1.1.1.90 to 2.2.2.2 Unique dports

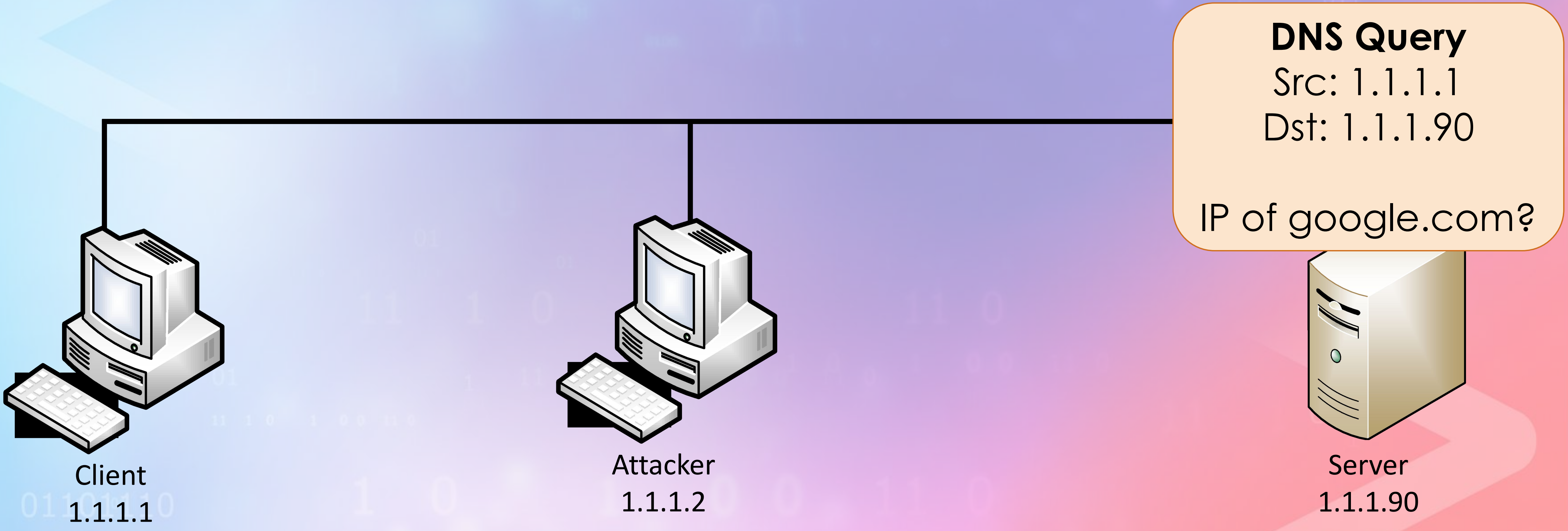
DNS resolving



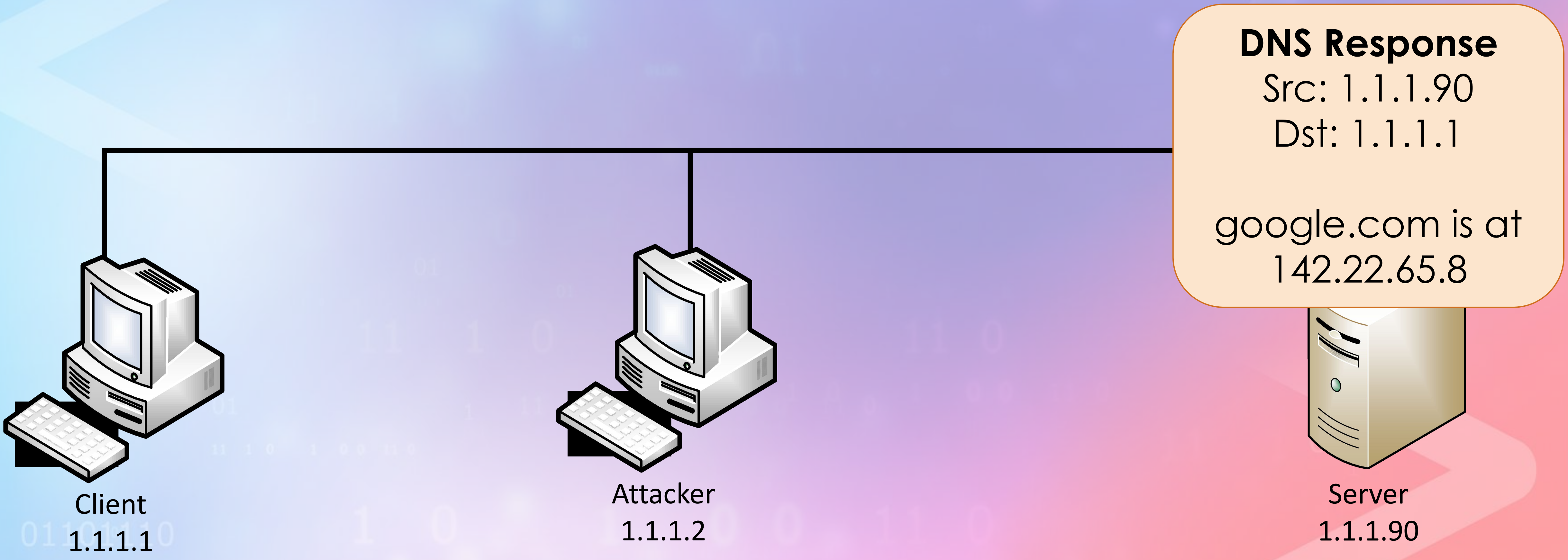
DNS Query
Src: 1.1.1.1
Dst: 1.1.1.90

IP of google.com?

DNS resolving



DNS resolving



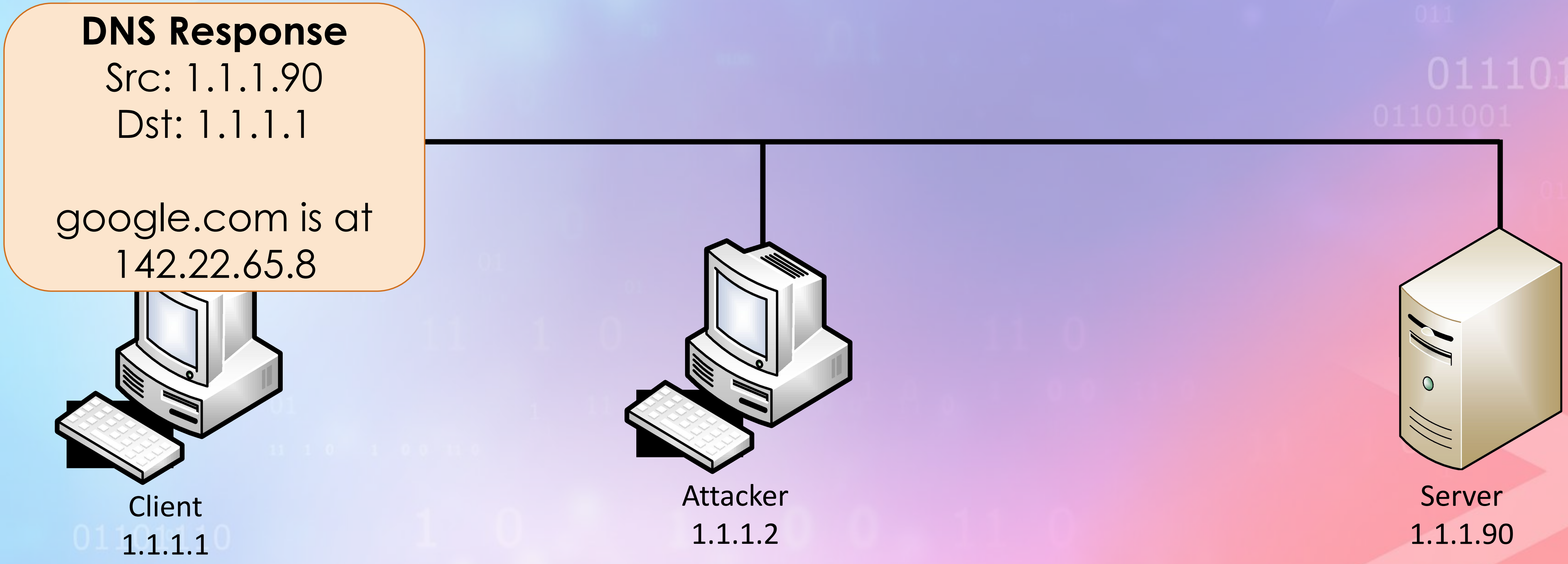
DNS Response
Src: 1.1.1.90
Dst: 1.1.1.1

google.com is at
142.22.65.8

DNS resolving

DNS Response
Src: 1.1.1.90
Dst: 1.1.1.1

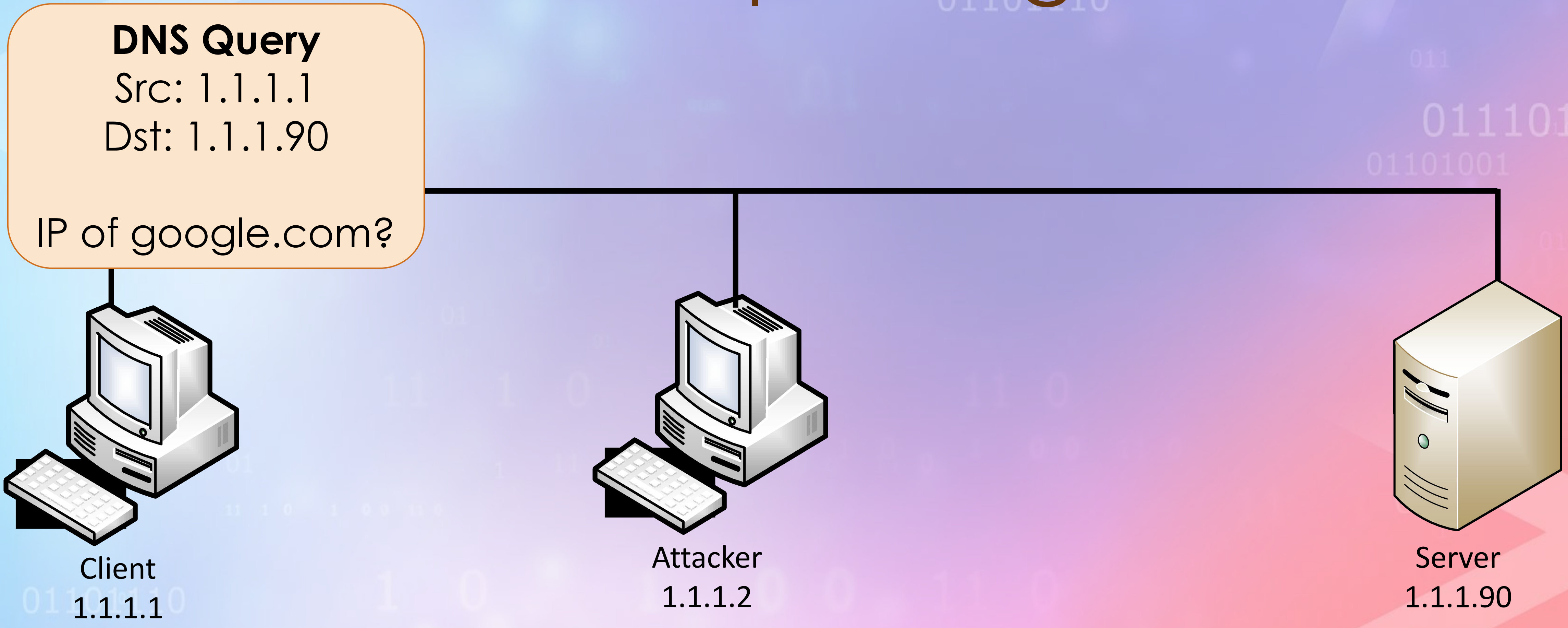
google.com is at
142.22.65.8



DNS spoofing

- Now assume the attacker can see all network traffic
- In real life this could be the result of a different attack, such as ARP poisoning

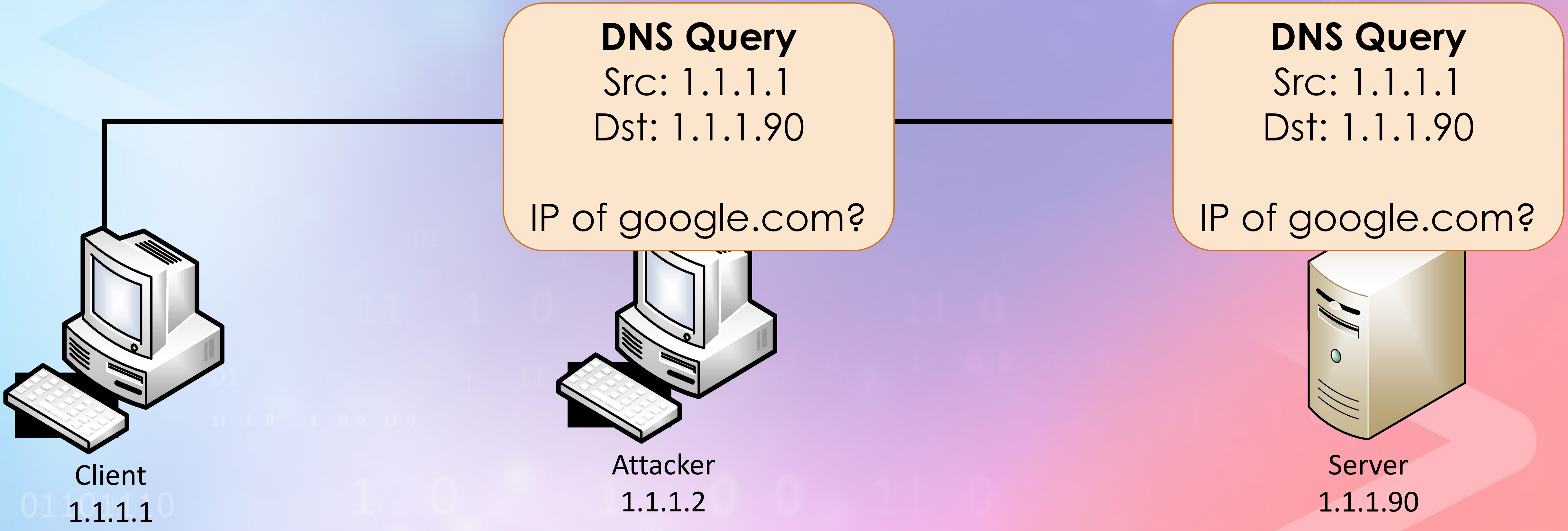
DNS spoofing



DNS Query
Src: 1.1.1.1
Dst: 1.1.1.90

IP of google.com?

DNS spoofing



DNS Query
Src: 1.1.1.1
Dst: 1.1.1.90

IP of google.com?

DNS Query
Src: 1.1.1.1
Dst: 1.1.1.90

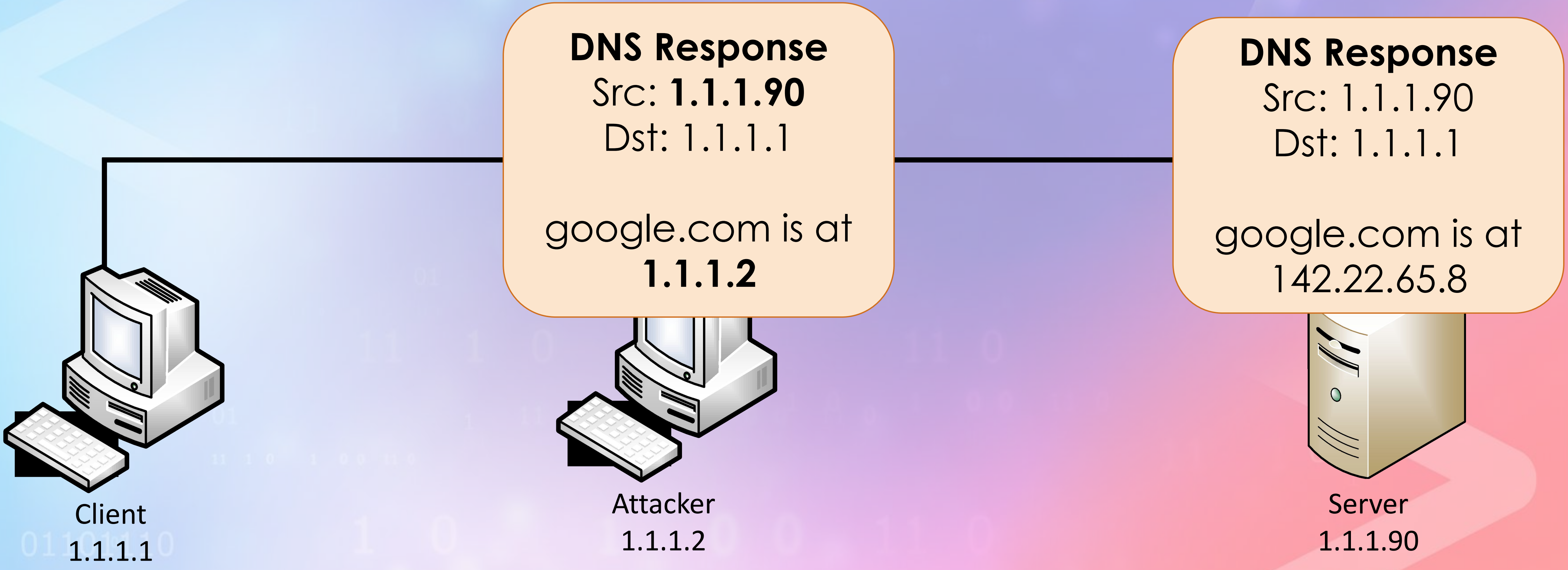
IP of google.com?

Client
1.1.1.1

Attacker
1.1.1.2

Server
1.1.1.90

DNS spoofing



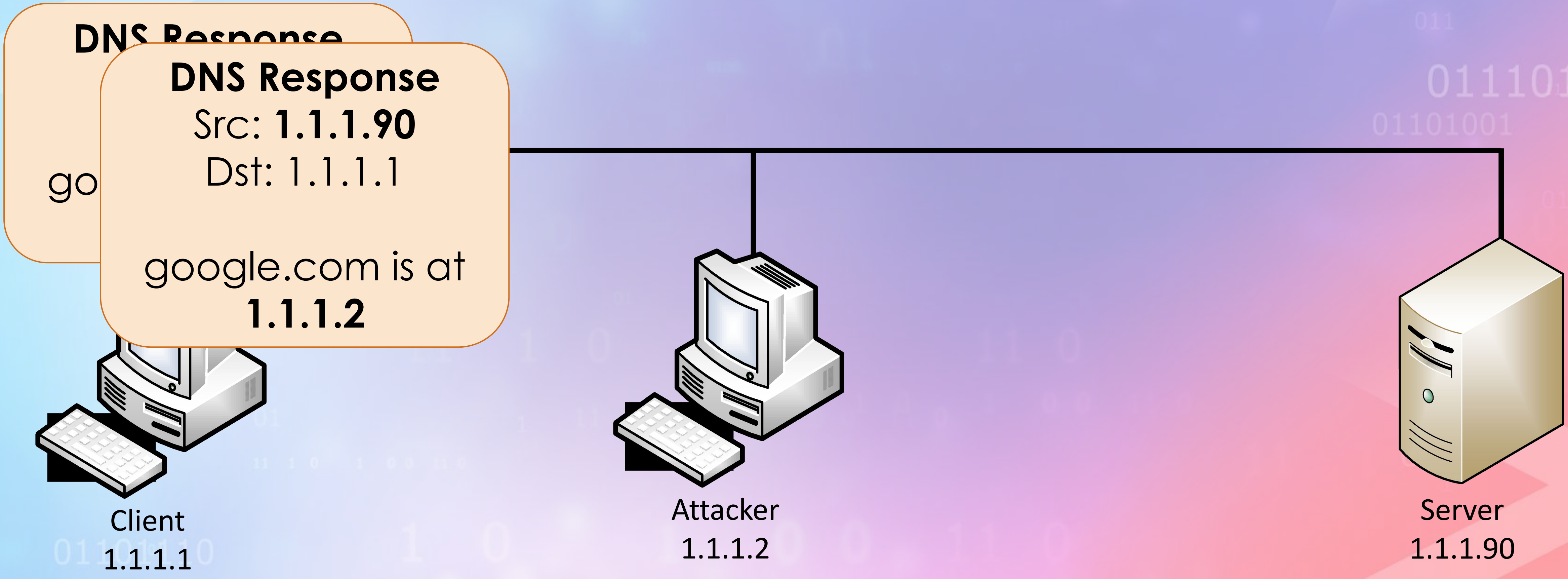
DNS Response
Src: **1.1.1.90**
Dst: 1.1.1.1

google.com is at
1.1.1.2

DNS Response
Src: 1.1.1.90
Dst: 1.1.1.1

google.com is at
142.22.65.8

DNS spoofing



DNS Response

go

DNS Response
Src: **1.1.1.90**
Dst: 1.1.1.1

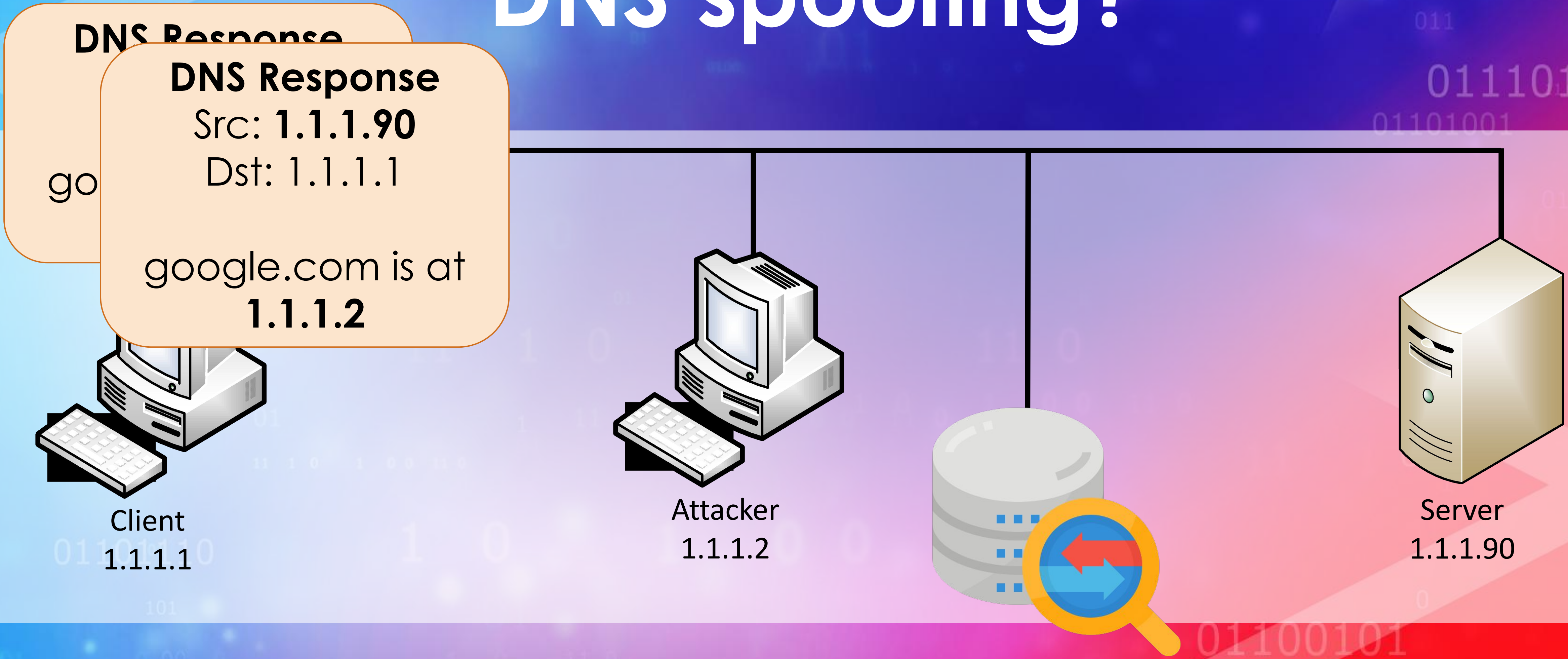
google.com is at
1.1.1.2

Client
1.1.1.1

Attacker
1.1.1.2

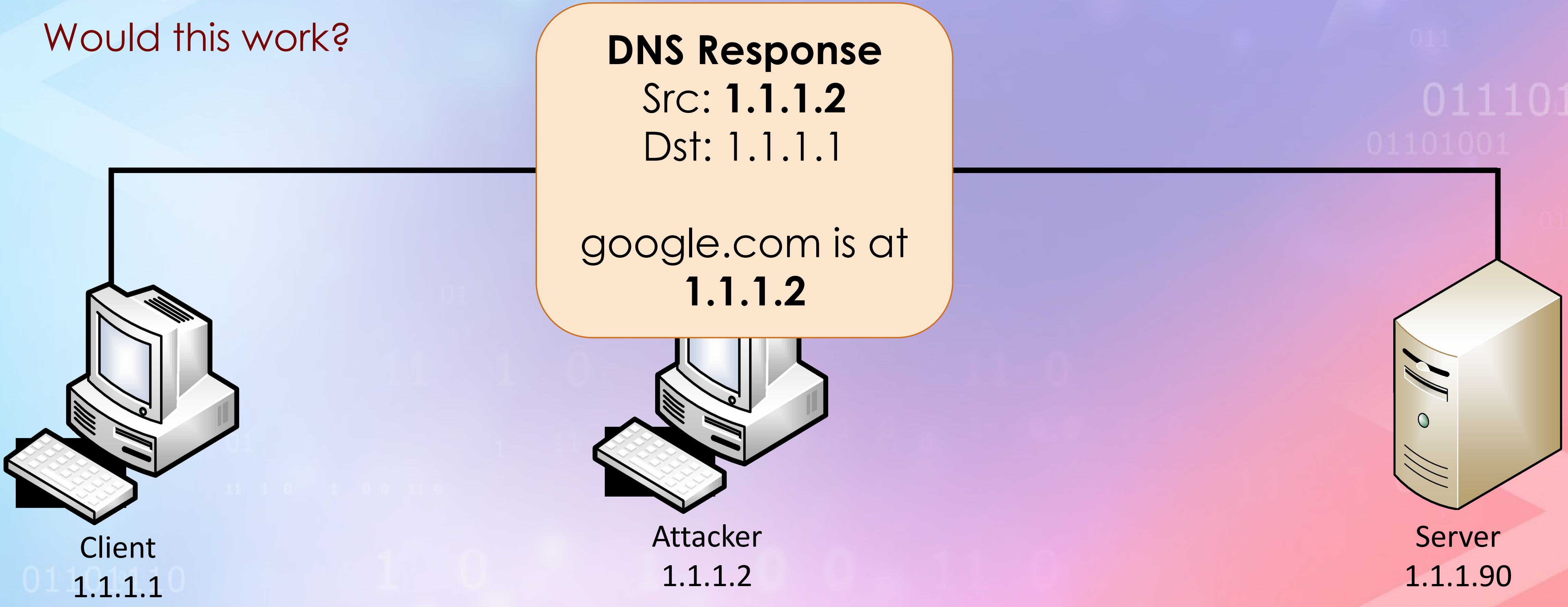
Server
1.1.1.90

How could we detect a DNS spoofing?



IP spoofing

- Would this work?



DNS Response
Src: **1.1.1.2**
Dst: 1.1.1.1

google.com is at
1.1.1.2

IP spoofing

- Client sent request to 1.1.1.90
- Response will be accepted only if it comes from 1.1.1.90
  - At least that's what the headers should say 😉
- What's stopping the attacker from setting the header's source IP e
  - to an IP that's "not his"?
- The MAC could optionally be spoofed as well to match the real DNS server's

Solution

- A server would never send two responses
- If we see a duplicate response
  - But with different answers
- Then it’s indication of DNS spoofing

Summary

→ Developing detections requires understanding of attack techniques

→ ARP and port scans can be detected by noticing repetition threshold

→ Maintaining a dict mapping a “source” to a list of its packets

→ And strictly defining how to group packets

Q&A