Sentinel

Timing

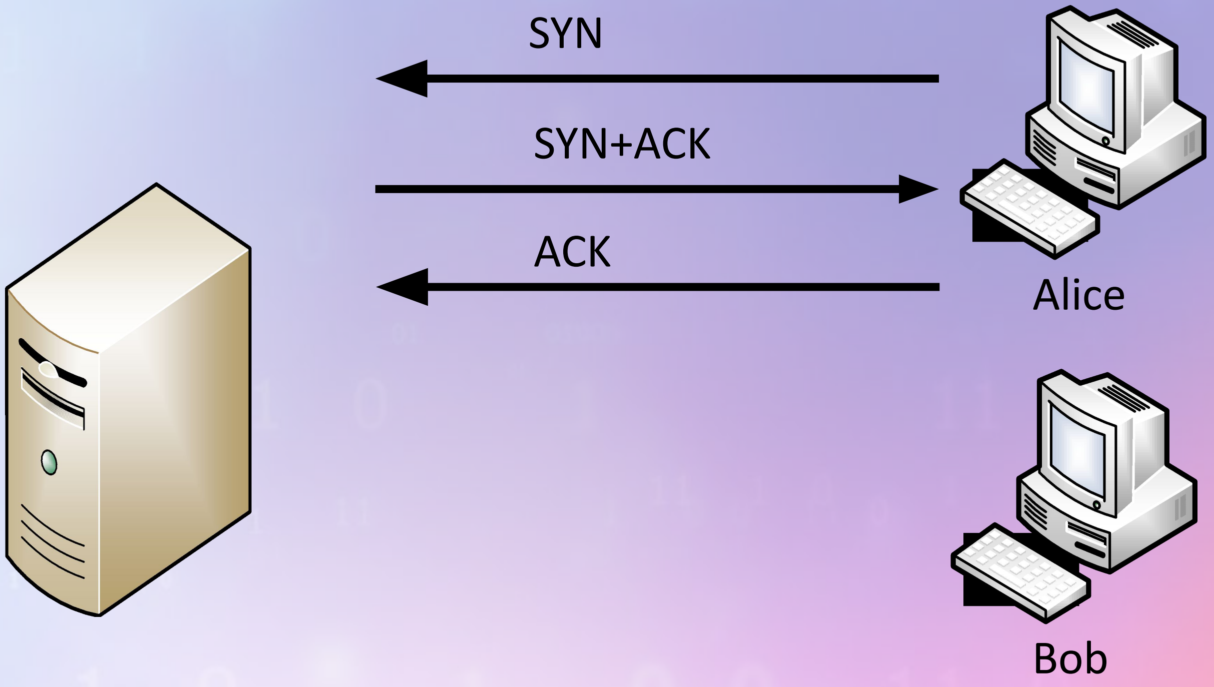
DEFENDING OUR DIGITAL WAY OF LIFE

Lesson objectives

→ Learn how to detect a **SYN Flood** attack

→ Understand **asynchronous programming**

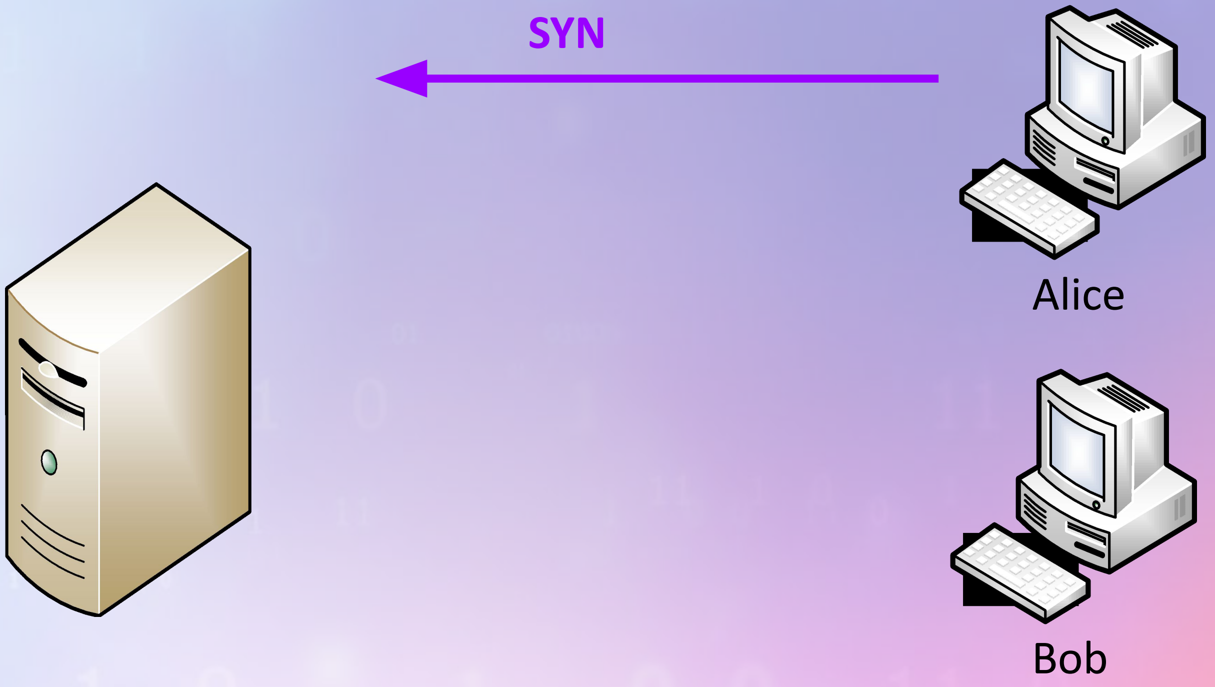
TCP 3-way Handshake



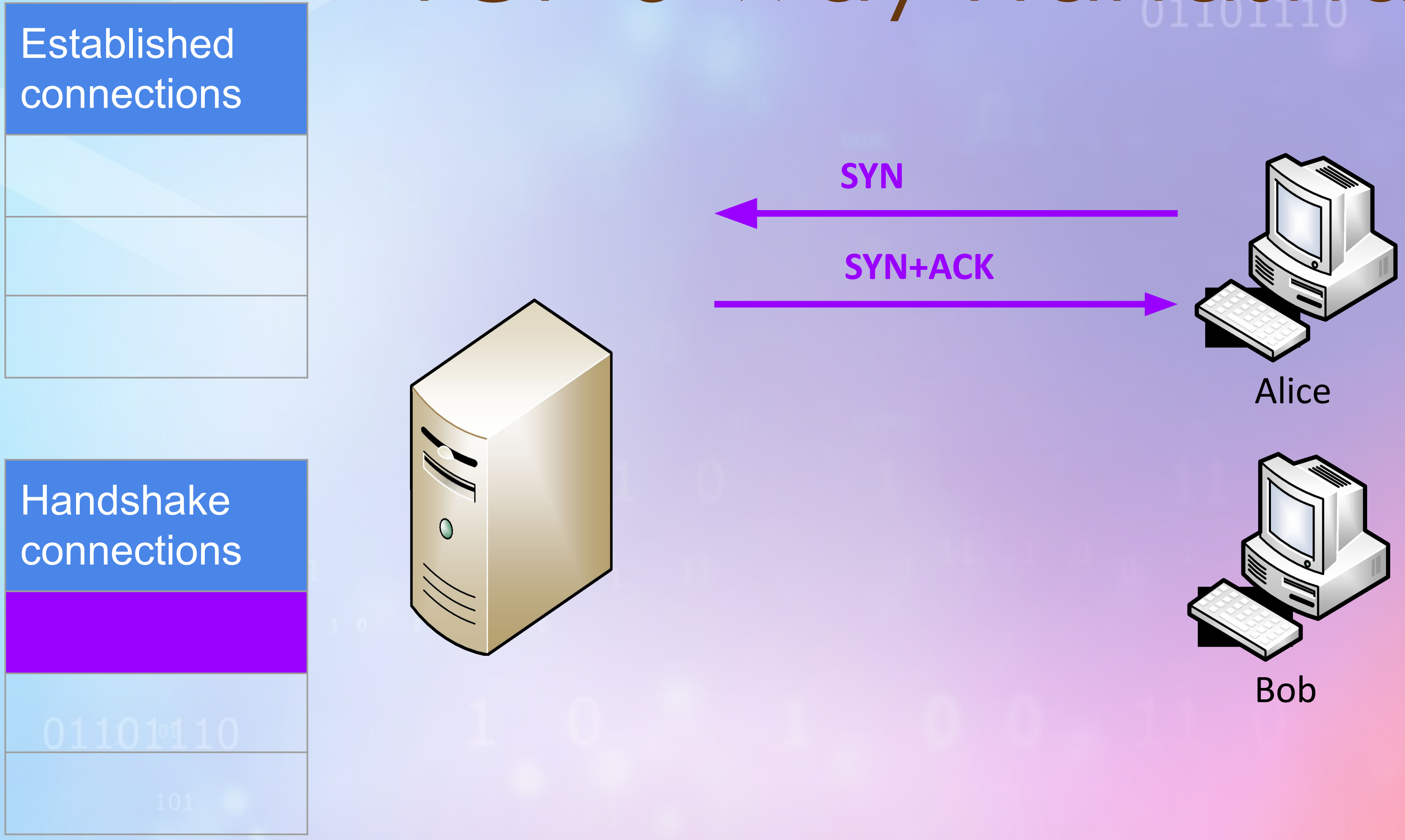
TCP 3-way Handshake

Established connections

Handshake connections



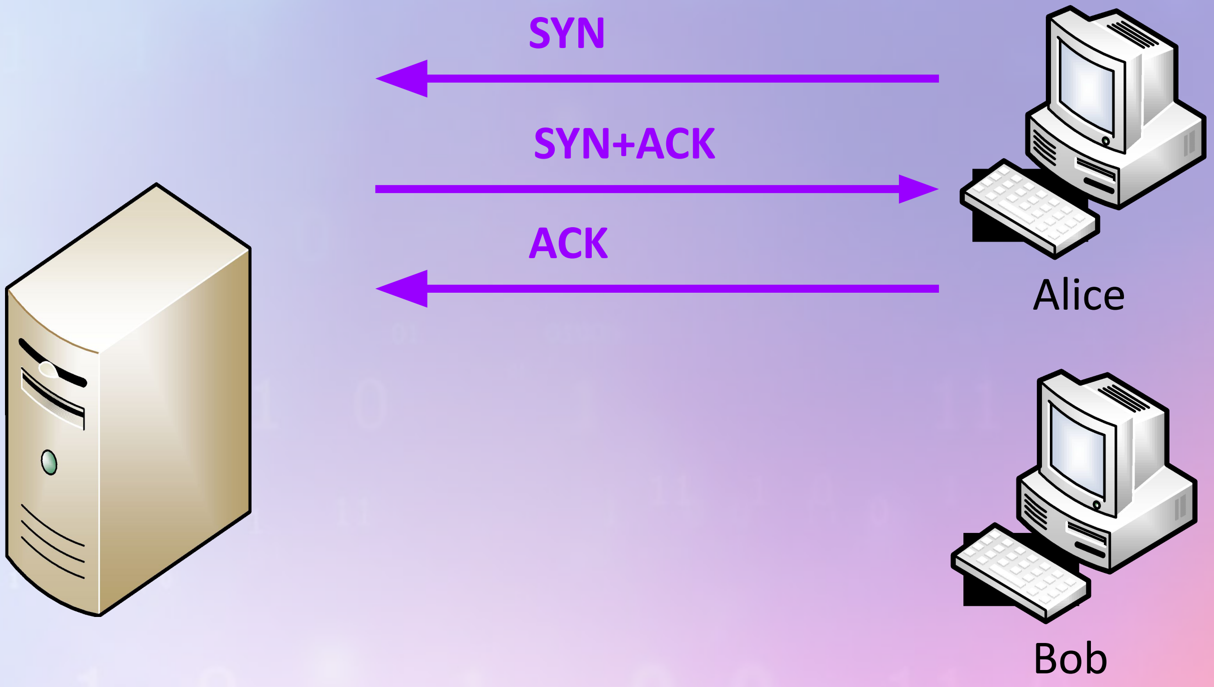
TCP 3-way Handshake



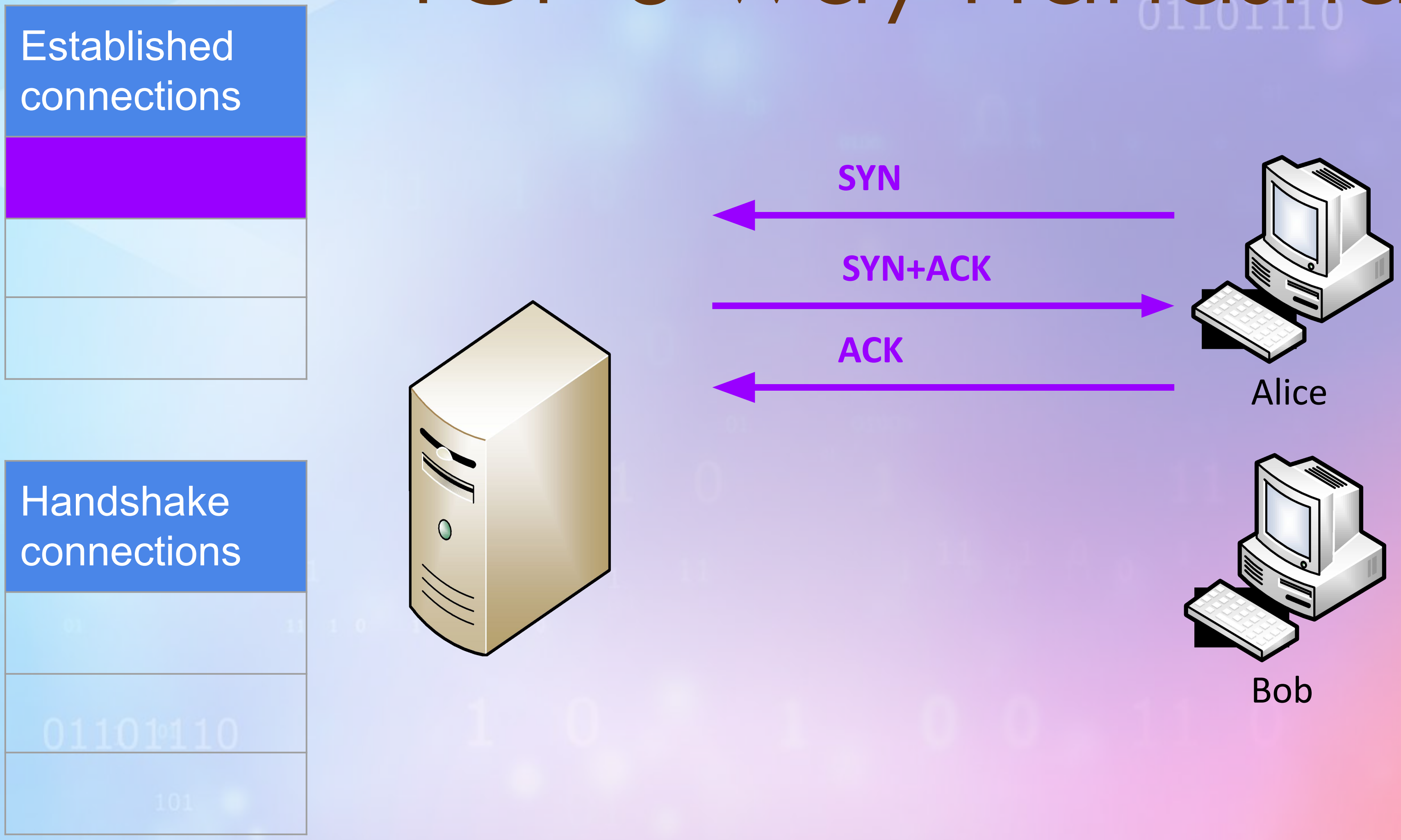
TCP 3-way Handshake

Established connections

Handshake connections
🟪



TCP 3-way Handshake



Established connections

Handshake connections

SYN

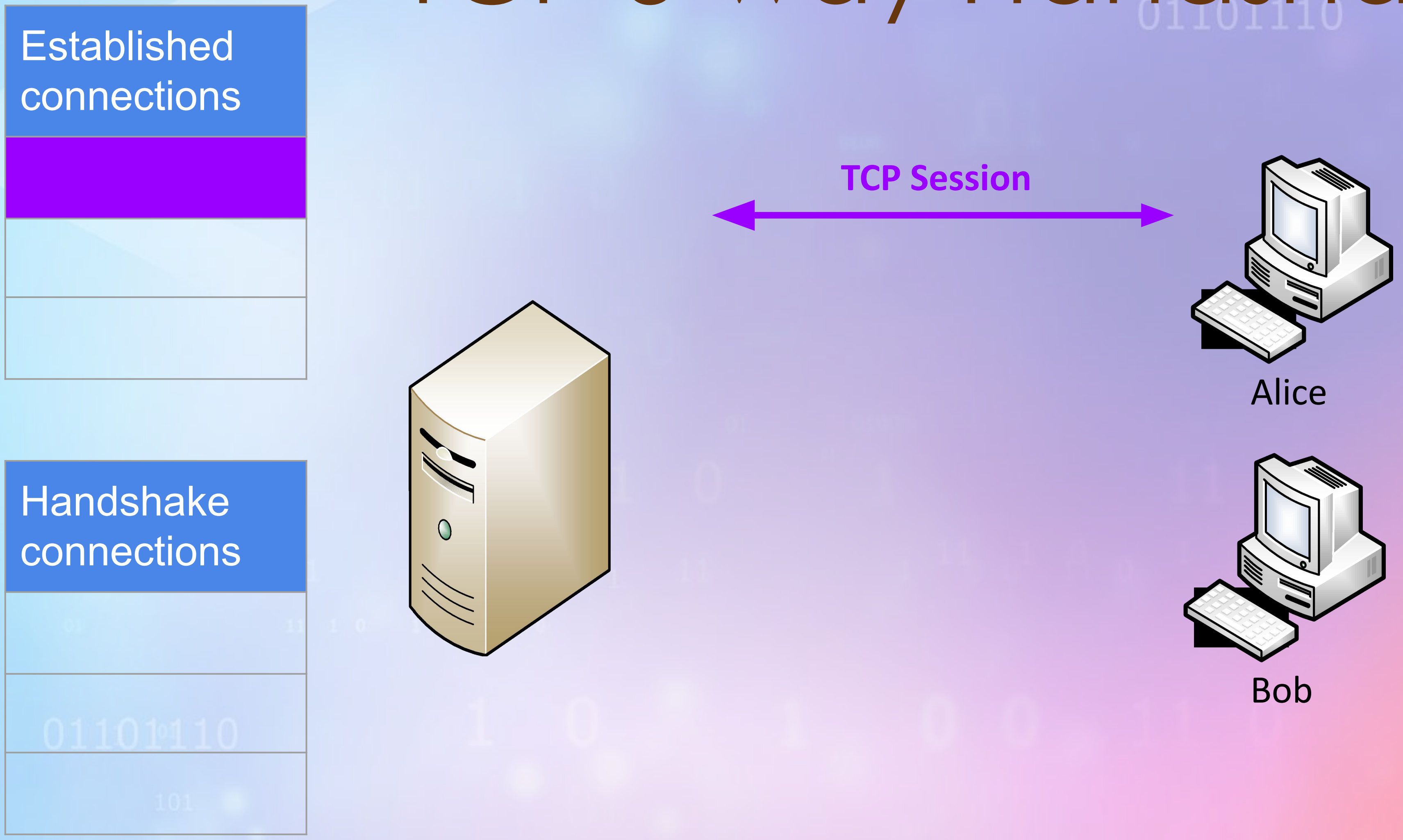
SYN+ACK

ACK

Alice

Bob

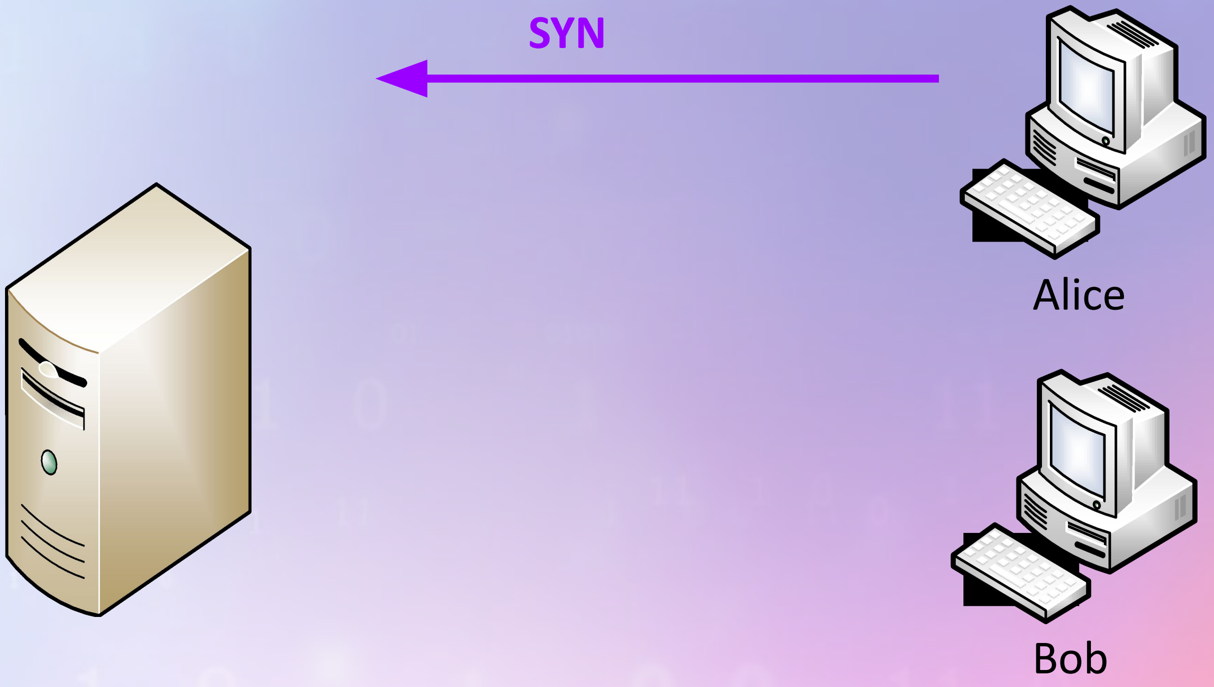
TCP 3-way Handshake



Broken TCP 3-way Handshake

Established connections

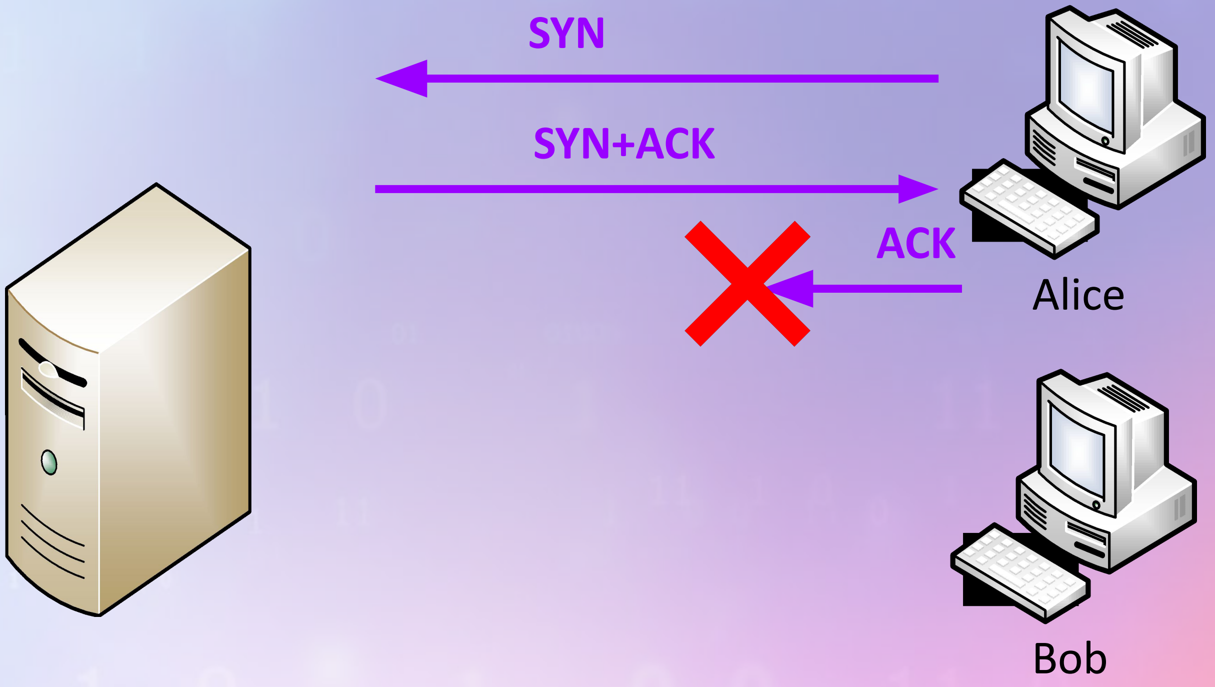
Handshake connections



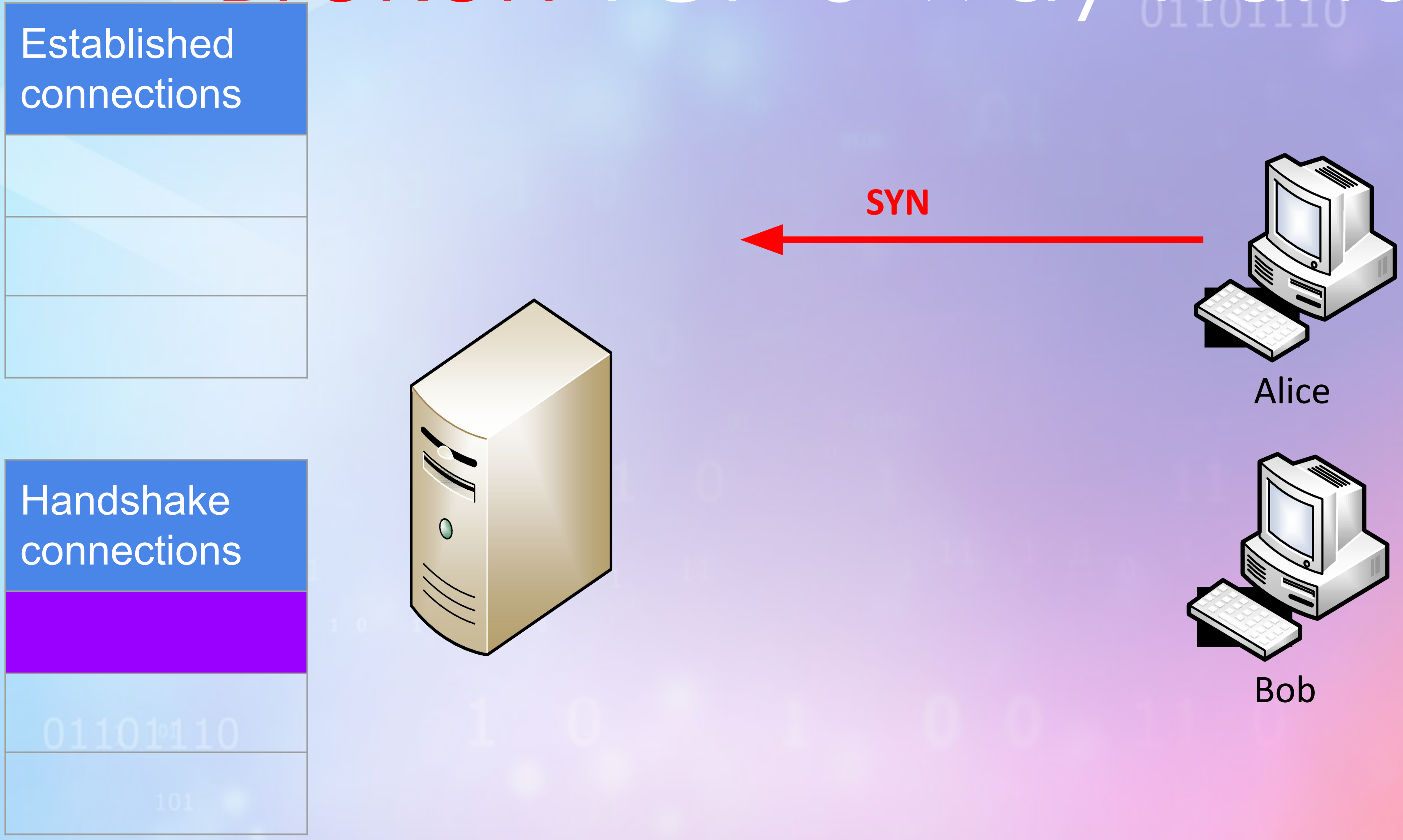
Broken TCP 3-way Handshake

Established connections

Handshake connections



Broken TCP 3-way Handshake



Established connections

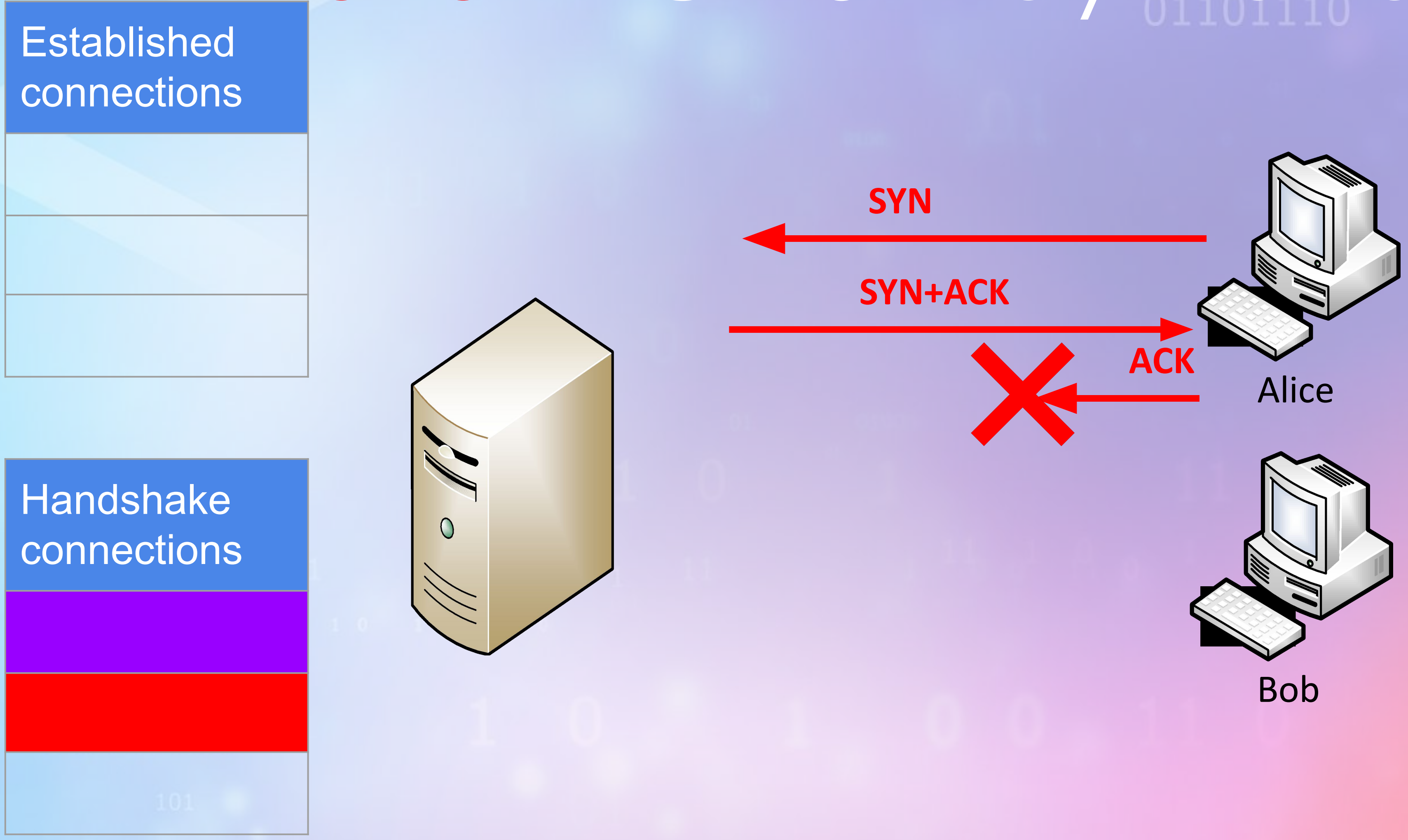
Handshake connections

SYN

Alice

Bob

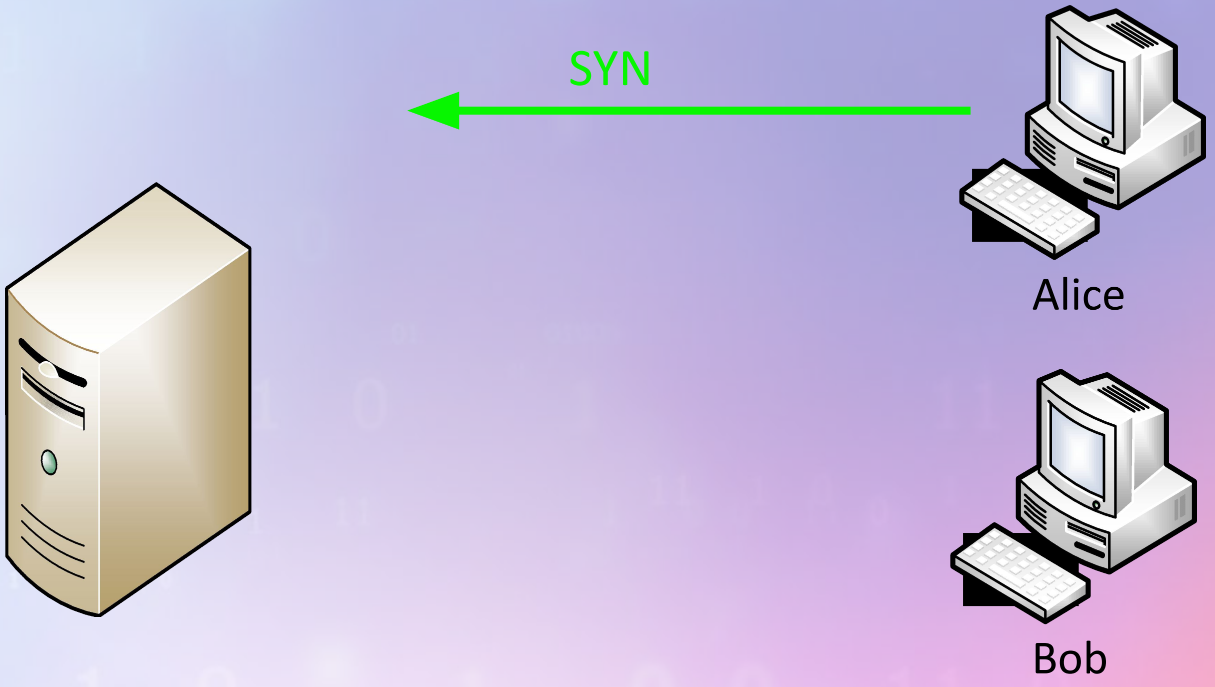
Broken TCP 3-way Handshake



Broken TCP 3-way Handshake

Established connections

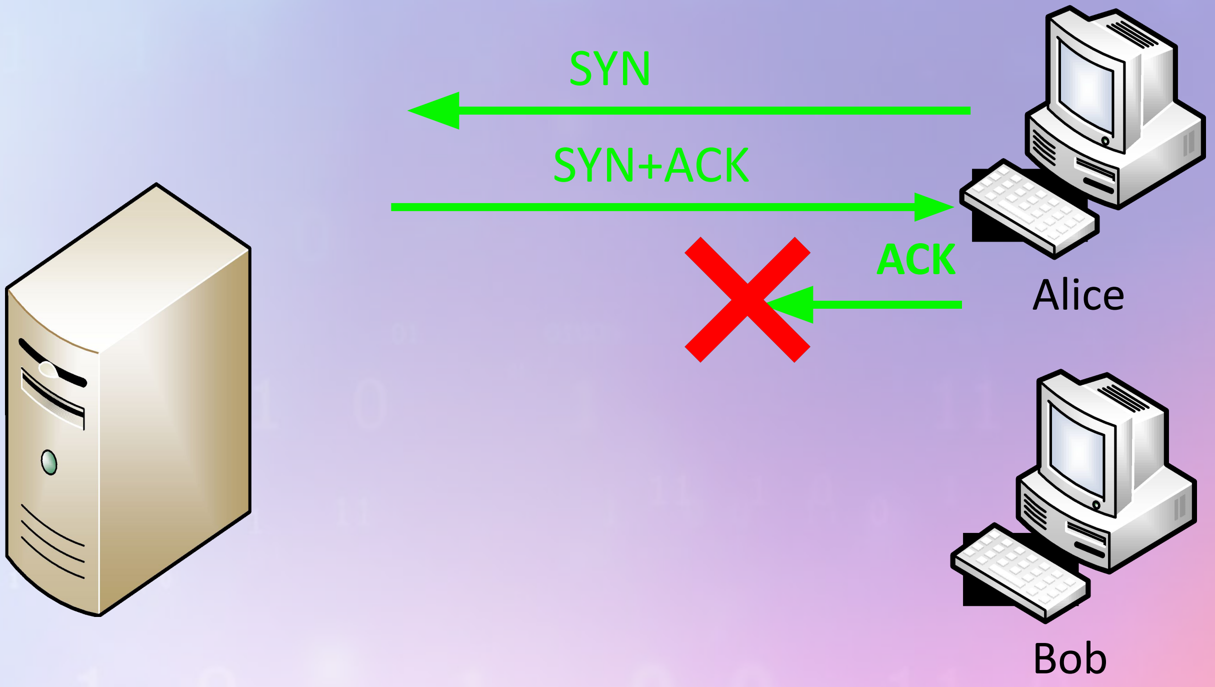
Handshake connections
🟪
🟥



Broken TCP 3-way Handshake

Established connections

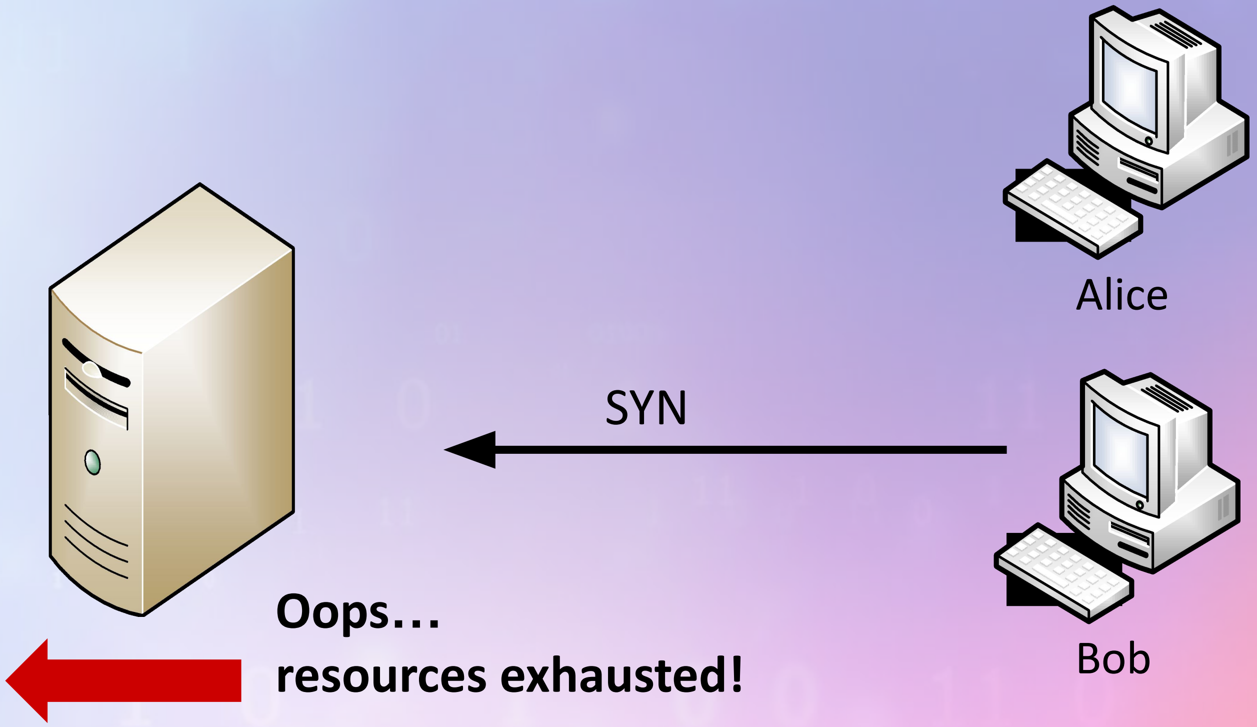
Handshake connections
🟪
🟥
🟩



Broken TCP 3-way Handshake

Established connections

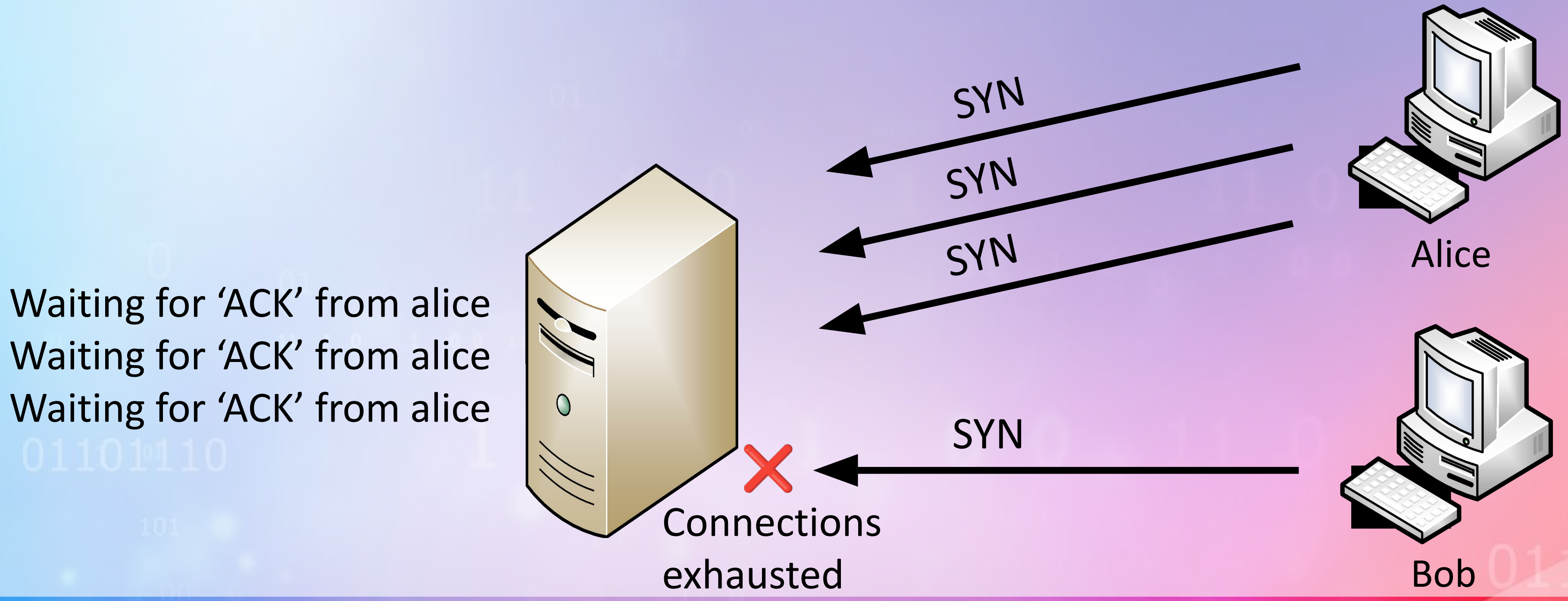
Handshake connections
🟪
🟥
🟩



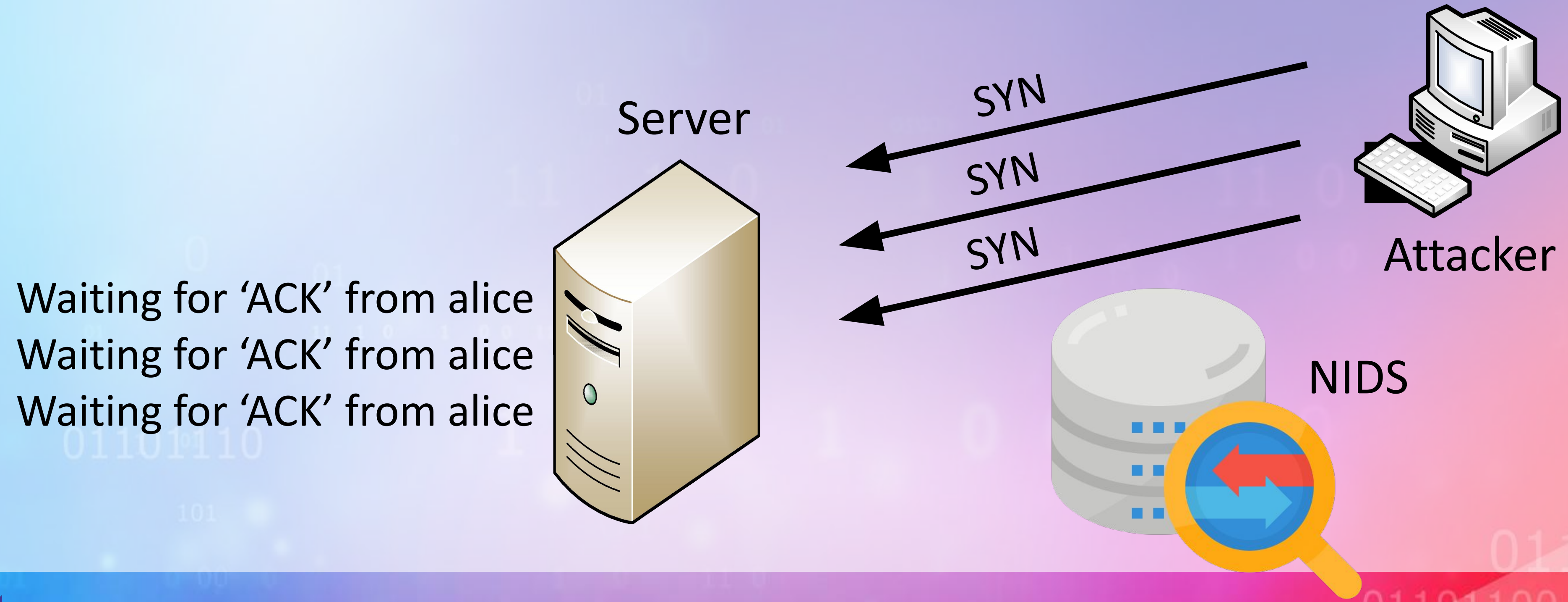
Oops…
resources exhausted!

SYN flood

- Known as "half-open attacks"
- Denial of service
- Modern OS have protections by default


SYN
SYN
SYN
Alice
Waiting for 'ACK' from alice
Waiting for 'ACK' from alice
Waiting for 'ACK' from alice
SYN
Connections exhausted
Bob

How could we detect a SYN flood?



Waiting for 'ACK' from alice
Waiting for 'ACK' from alice
Waiting for 'ACK' from alice

SYN flood detection decision making

- So far, our detection was
  - If we see **too many packets** that match a condition
- But here, it's different
  - Malicious activity is indicated by the **lack of many specific packets**

The precise requirement 📜

- ✅ Alert whenever
- ✅ There are more than X packets
- ✅ In the last Y seconds
- ❓That were SYN+ACK packets
- ❓**For which there was no response**

The precise requirement 📜

- ❓SYN+ACK packets
- ❓**For which there was no response**
- This implies a time delay
  - Between a packet’s arrival
  - And the addition to the window
- How can we implement this?

Timeouts!

- Start a timer per SYN+ACK packet
  - ***threading.Timer*** in Python
- If a corresponding ACK packet arrives
  - Cancel the timer
- If the timer times out
  - It means the ACK never arrived
  - Now add it to the sliding window

Asynchronous programming

- Our programming is usually
  - Do task #1, then
  - Do task #2, then
  - ...
- Also called synchronous programming
- But with asynchronous, we don’t block the code
- We handle events when their time comes

Asynchronous programming

- We’ve done this before!
- Multi-client server with select()
  - We read from a socket in the event that data arrived to it
- sniff(prn=handle_packet)
  - We get called back when a packet arrives

Remember our stream reassembly?

- We start tracking a stream when we see a **SYN** packet
- We finish tracking when we see a **FIN/RST** packet
- What would happen if the (**malicious**) client never sends FIN?
  - Could also be a normal client that has network issues
  - i.e. **FIN** packet lost in transit
- Memory leak - we will hold the stream in memory forever!
  - Too many streams and we will crash
- Solution?

Solution in words

- If we're tracking a stream
- And no packet arrives that corresponds to it in the last X seconds
- Drop the stream, cleaning resources
- How can we do it?

Timeouts!

- Maintain a timer per tracked stream
  - ***threading.Timer*** in Python
- Once a packet arrives that’s part of the stream, extend the timer
- If the timer ever gets to time out
  - The callback will delete the stream
  - Stopping the tracking of it

Summary

→ An **NIDS** is a real-time monitoring

→ It requires **asynchronous code** to make **time-related decisions**

→ We must think about **real-time resource management**

Q&A