

IEMS 5703

Network Programming and System Design

Lecture 10 - Scalability of Network Applications

Albert Au Yeung
22nd March, 2018

Scalability of Network Applications

Scalability of Network Applications

- What does **scalable** mean? We need to understand what is **scalability** before we can build something that is scalable.
- Is **scalable** the same as:
 - High availability (always available to the users)?
 - High performance (handle every request very quickly)?
 - Cost effective (low cost to build a system that serves many users)?

Scalability

Why should online services be scalable?

- Consider what would happen when a lot of users are browsing your website, using your app, or accessing your APIs
- Growth in **traffic** and **data**
- Our objectives:
 - To keep the service **available**
 - To keep the **performance** of the system at a certain level
 - To keep it **cost-effective** to operate the service

Scalability

Consider the following systems:

- Common client-server system with a **database**
- Clients are distributed **geo-graphically**
- Different **types** of clients (e.g. Web, mobile, API)
- Different **kinds** of requests (e.g. retrieval, data submission, image uploading, file editing, video encoding...)
- Different kinds of **data** (e.g. relational data, text, images, videos...)
- ...

Scalability

Scalability does **NOT** only mean:

- Improving your codes
- Using a more powerful machine
- Adding more RAMs

Scalability is about

- Identifying and mitigating bottlenecks
- Considering the whole architecture of the system in order to engineer a solution

Scalability

Scalability is **NOT** equal to

- Performance
- Choosing a particular operating system
- Choosing a particular programming language
- Choosing a particular storage technology

Scalability - Related Concepts

High Availability (HA)

- A system is **highly available** if it is operational for most of the time in a specific period (e.g. a year)
- For example, if a service has 99.9% availability, it is expected that the service will be operational for $24 \times 365 \times 0.999 = 8751.24$ hours in a year (also known as **up time**)
- **Principles** of HA: eliminate single points of failure, avoid data loss in case of failure, and detection of failures as they occur
- A highly available system will be a distributed one, hence usually scalable

Scalability - Related Concepts

System Performance

- How efficient is the system in performing a task given certain amount of computing resources (CPU time, memory, etc.)
- Using a language that executes code faster at runtime
- Using a faster sorting algorithm
- Optimizing your SQL query to shorten the time to retrieve data from the database
- ...

Scalability - Definition

Two common definitions/views of "scalability":

1. Scalability is the ability to handle increased workload
(without adding resources to a system)
2. Scalability is the ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capacity

Reference: Charles et al., 2006. On System Scalability.

<http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=7887>

Scalability - Definition

The first definition is of less interest to us in this course

- It does NOT focus on whether the system can be improved / extended when the workload increases
- It focus on whether the system has good performance on a large problem, not on how the system can be scaled

Instead, we are more interested in the second definition, in particular:

- **"repeatedly applying a cost-effective strategy"** to extend the system's capacity

Scalability - Definition

"Repeatedly applying a cost-effective strategy"

- We are NOT interested in **one-time increase** in capacity, such as:
 - replacing a $O(n^2)$ algorithm with an $O(n \log n)$ one
 - coding in a language which produces programs that run faster
- We are NOT interested in solutions that requires **a relatively high cost**, such as:
 - replacing a 4-core CPU within a 16-core CPU
 - building a super-computer using advanced HPC technologies

Horizontal vs. Vertical

Vertical scaling – also known as "**scale UP**"

- Add resources to a single node in a system
- E.g. adding CPU or RAMs, increasing the disk size of a DB server, etc.

Horizontal scaling – also known as "**scale OUT**"

- Add more nodes to a system
- Systems are designed to be distributed
- E.g. adding more servers to a cluster of computers

Scaling Out

In order to achieve scalability, we usually aim to build a system that can easily scale-out (horizontally)

- Allow distributing workload easily
- If you scale-up, you will quickly reach a **limit** (either technically or financially)
- Trade-off: you need something to **manage** your large number of nodes

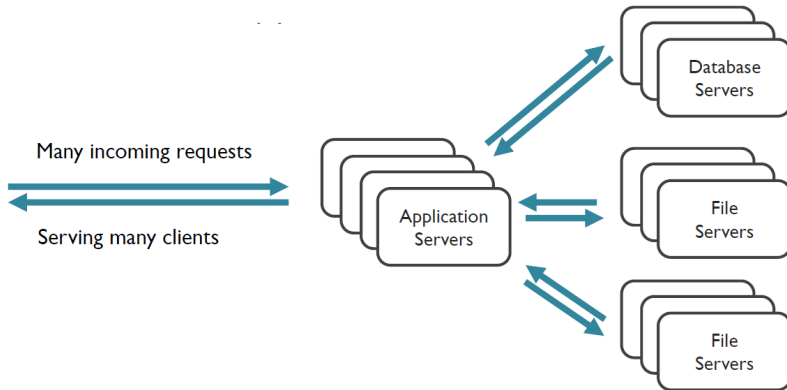
Scaling Out

- Have you ever imagined how many servers are supporting **Google** or **Facebook**?
 - Let's take a look at Facebook's report of [Fourth Quarter and Full Year 2017](#):
-

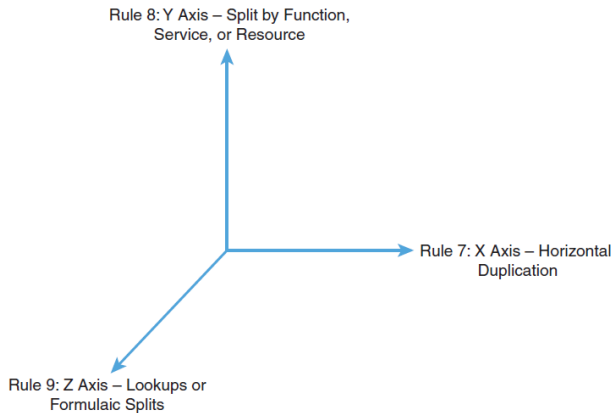
- **Daily active users (DAUs)** – DAUs were 1.40 billion on average for December 2017, an increase of 14% year-over-year.
- **Monthly active users (MAUs)** – MAUs were 2.13 billion as of December 31, 2017, an increase of 14% year-over-year.
- **Mobile advertising revenue** – Mobile advertising revenue represented approximately 89% of advertising revenue for the fourth quarter of 2017, up from approximately 84% of advertising revenue in the fourth quarter of 2016.
- **Capital expenditures** – Capital expenditures were \$2.26 billion and \$6.73 billion for the fourth quarter and full year 2017, respectively.
- **Cash and cash equivalents and marketable securities** – Cash and cash equivalents and marketable securities were \$41.71 billion at the end of the fourth quarter of 2017.
- **Headcount** – Headcount was 25,105 as of December 31, 2017, an increase of 47% year-over-year.

Scaling Out

- How more servers help you to scale?



Scalability

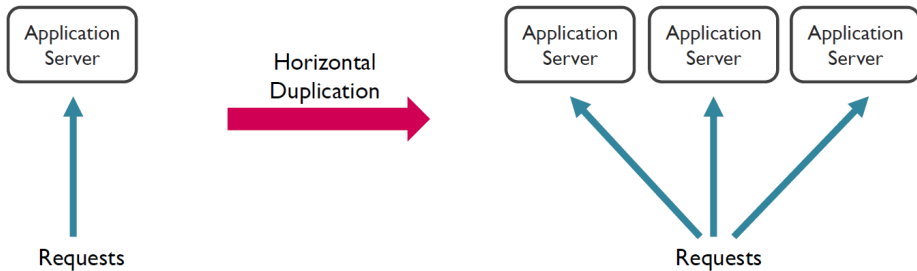


- Principles for distributing workloads in a system
 1. **Horizontal Duplication**
(Introduce redundancy)
 2. **Split by Functions**
(Split by verbs or nouns)
 3. **Split by Storage**
(Formulaic split: split database based on one or more columns)

Scalability Rules: 50 Principles for Scaling Web Sites Chapter 2 (Figure 2.2)

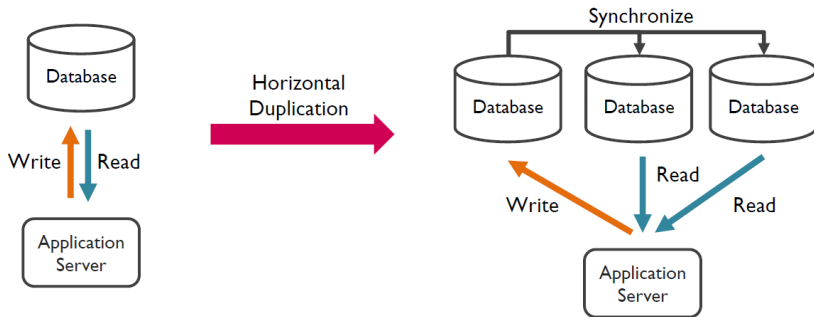
1. Horizontal Duplication

- When multiple instances of your service can operate **independently**:



1. Horizontal Duplication

- When the **read-to-write ratio** of your database is high (e.g. 5:1):
 - For many applications, the database is read much more frequently than is written to
 - Consider how often your post a new status on Facebook vs. how often you read your news feed
 - Simple way to scale: have multiple **read-only** database to handle the many read requests



2. Split by Functions

Scaling through distribution based on the **separation of distinct / different functions** (verbs) and **data** (nouns)

By functions (verbs)

- Registration
- Authentication / Login
- Search
- Recommend
- Resize images

By data (nouns)

- User profiles
- Items for sale
- Product catalogues
- Images

3. Split by Storage

- Also known as **sharding** - take a data set and partition it into multiple parts based on some rules
- Consider an online services that serves multiple enterprise users (each company has an account and data are not supposed to be seen by another company)
 - You are not going to combine data from different companies in any function
 - You can have dedicated servers storing data for each company

Summary

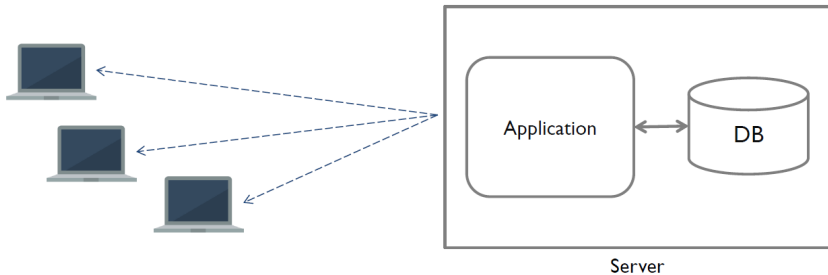
The keys to build scalable systems are

- Identifying **bottlenecks** of the system
- **Horizontal duplication**: allow scaling to be done quickly
- **De-coupling**: let loosely-coupled modules interact through simple and well-defined interfaces

Building a Scalable System - An Example

Building a Scalable System - An Example

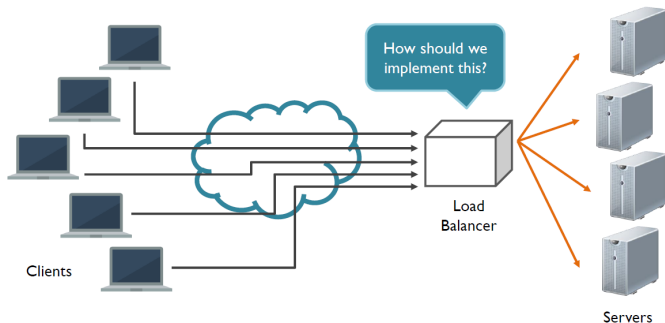
- Let's walk through the process of designing a scalable system
- Assuming that we are going to build a photo sharing application with some social functions
- We start with the following simple architecture



Load Balancing

Load Balancing

The act of distributing workloads across multiple computing nodes



- Avoid overloading a single node
- Maximise utilization of different nodes
- Optimise usage of resources
- Increase reliability and availability

Load Balancing

Different ways to implement a load balancer

- DNS / Hardware / Software
- Implement it on different networking layers
- Algorithms: random / Round-robin / dynamic scheduling

Load Balancing Algorithms

- **Random**
- **Round-Robin**
 - Distribute load equally by using a rotating scheme
- **Weighted Round-Robin**
 - A performance weight is assigned to each server
- **Least Connections**
 - Sends requests to a server with the fewest number of connections
- **Fastest Response Time**
 - Select the server that responded in the shortest time

Server Health Checking

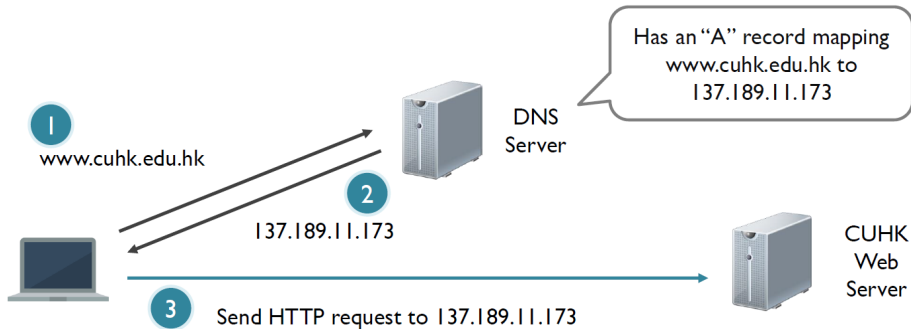
- A load balancer might need to check whether the servers are operating normally and are able to give responses.
- How can we check if one of the servers has died?
 1. **Passive: Observe** the **network traffic**
 2. **Active: Probe** the server for a **quick response**
- What are the pros and cons of these two?
- How can health checking be done?

Server Health Checking

- **Ping**
 - Send an ICMP message to the server and check for response
- **TCP Connection**
 - Attempt to establish a TCP connection to the server (on port 80)
- **HTTP GET (Header)**
 - Check the status code of a GET request
- **HTTP GET (Content)** Check the content returned by the server for a GET request
- **Question:** What are the limitations of the first three methods?

DNS

- DNS stands for **Domain Name System**
- A **directory service** of the Internet



DNS Load Balancing

A simple way of implementing load balancing

- Create two or more **“A” records** in the DNS zone
- The DNS server sends the client a list of records in **random order** or in a round-robin fashion
- The client attempts to connect to the application server using the **first IP address** in the list

DNS Load Balancing

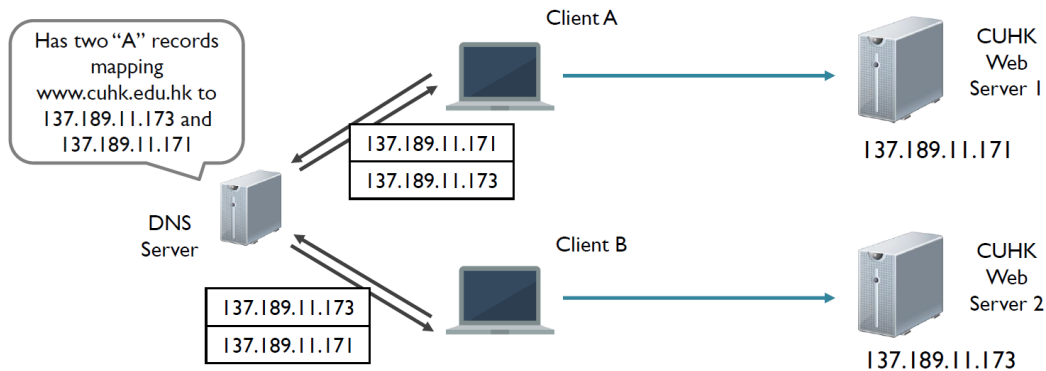
What are “A” records?

- Address record: mapping a domain name to an IPv4 address

Other records:

- AAAA: domain name to IPv6 address
- CNAME: alias of a domain name
- MX: mail exchange (identifies the mail server of the domain)
- NS: name server record

Round-robin DNS Load Balancing



DNS Load Balancing - An Example

```
$ dig www.youtube.com
; <<>> DiG 9.9.5-3ubuntu0.5-Ubuntu <<>> www.youtube.coma
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33788
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION: ; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION: ;www.youtube.com. IN A

;; ANSWER SECTION:
www.youtube.com. 21423 IN      CNAME  youtube-ui.l.google.com.
youtube-ui.l.google.com. 146 IN      A 64.233.187.190
youtube-ui.l.google.com. 146 IN      A 64.233.187.136
youtube-ui.l.google.com. 146 IN      A 64.233.187.91
youtube-ui.l.google.com. 146 IN      A 64.233.187.93
```

DNS Load Balancing

Simple, but a few limitations:

- **Stickiness:** It takes time to propagate changes in DNS records □ adding or removing servers can be slow
- Loading can NOT be balanced accurately due to **DNS caching** (e.g. when a lot of users are from the same ISP)
- It does NOT take into account transaction time, server load, network congestions, etc.
- NO **fault tolerance** (DNS server does not know if a server is operating or not)

DNS Load Balancing

- While having some limitations, DNS load balancing is an important method for achieving **availability**
- The **only way** to divert traffic to **different data centres**

Amazon Web Services Suffers Crash, Takes Down Netflix, Reddit, Tinder And Other Huge Parts Of The Internet

By [Romellaine Arsenio](#), Tech Times | September 23, 11:14 AM



The Amazon Web Services crashed Netflix, Reddit, Tinder and other online applications, causing users' problems reported throughout the day.
(Photo : David McNew | Getty Images)

A monstrous outage from Amazon Web Services crashed down Netflix, Reddit, Tinder and other major websites, sending netizens in fury for missing movies, hook-ups and other fun online activities.

AWS powers web and mobile applications, and provides data processing and warehousing, storage and archiving to websites all over the world.

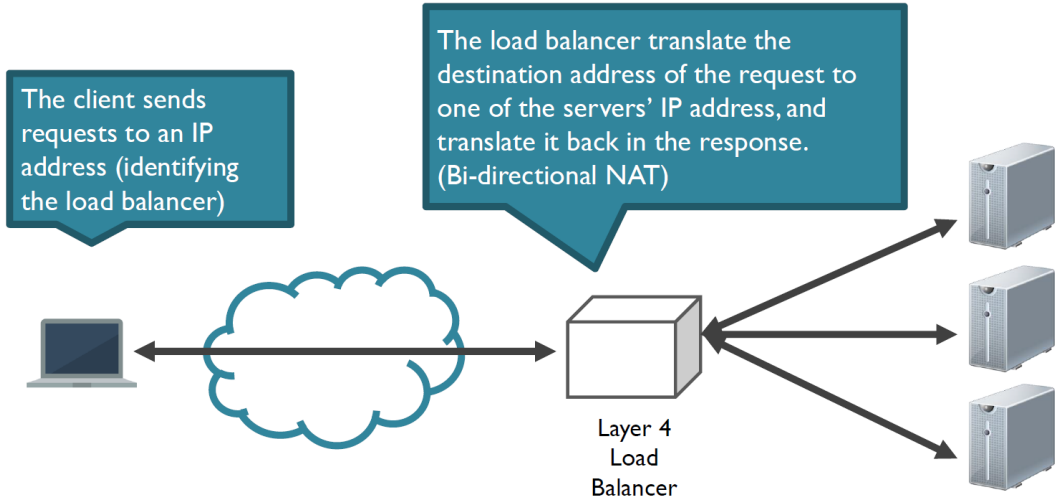
Hardware / Software Load Balancing

- Load balancing can be done at the different layers

OSI model Layer 4 (Transport Layer) Load Balancing

- Relatively simple
- Balancing server load without inspecting the content of messages
- Distribute traffic based on servers' response time
- Routing is based on inspecting the first packet of the data stream

Layer 4 Load Balancing



Layer 4 Load Balancing

What are the advantages of Layer 4 Load Balancing?

- Simple: easy to implement
- Efficient: load balancer only inspect the first packet However

However

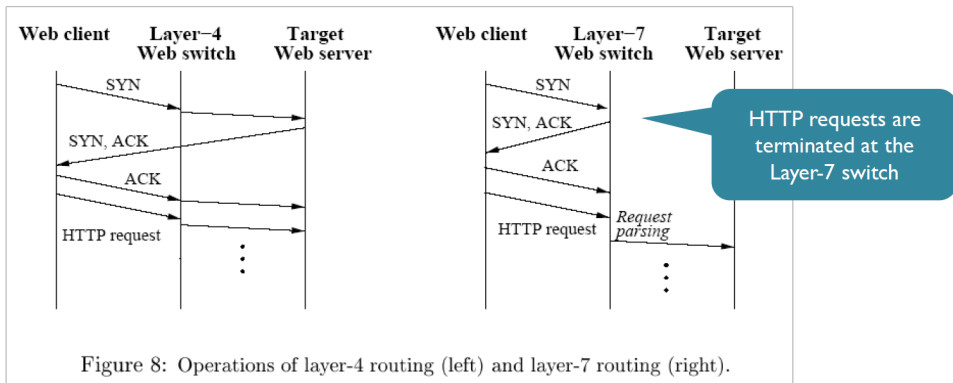
- It cannot maintain application session information
- It cannot route requests to different servers based on their content (e.g. requesting static content vs. dynamic content)

Layer 7 Load Balancing

Layer 7 (Application Layer) Load Balancing

- **Application-level** load balancing
- Parse requests in the **application layer** (e.g. HTTP), and distribute them to servers **based on the content** of the request (e.g. the URL or the cookie)
- Relatively **high overhead** in parsing the metadata
- Mostly **HTTP** (because of the popularity of Web apps)

Layer 7 Load Balancing



Ref: Cardellini, Casalicchio, Colajanni, and Yu. 2002. The state of the art in locally distributed Web-server systems. ACM Computer. Survey, 34, 2 (June 2002), 263-311.

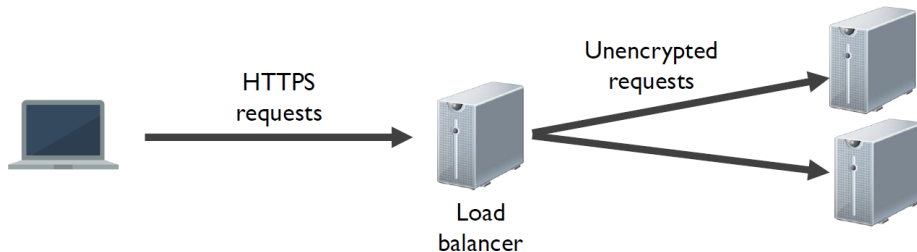
Layer 7 Load Balancing

Characteristics of Layer 7 Load Balancing

- More CPU-intensive (e.g. for parsing HTTP content)
- Can also apply compression, encryption or caching on the content to be delivered
- Does not require all servers in the backend to have serve the same content (compare with Layer 4 load balancing) Layer 7 load balancers are also called **Application Delivery Controllers**

Layer 7 Load Balancing

- One important ability of a layer 7 load balancer is **SSL termination**
- The load balancer has to be able to decrypt the request in order to **inspect the content**
- It therefore must be configured with **a valid SSL certificate**



Building UI in Python

Curses

- **curses** is a module in the standard library for controlling **text mode** displays (for Linux and Mac only; try <http://effbot.org/zone/console-index.htm> if on Windows)
- It allows you to **display** texts on the screen and also handle **keyboard inputs**
- A tutorial of **curses**: <https://docs.python.org/3/howto/curses.html>
- **Example**: displaying the string **Hello** on line 3 starting at the 10th column

```
from curses import wrapper

def main(stdscr):
    stdscr.clear()           # Clear everything on the screen
    stdscr.addstr(3, 10, 'Hello') # Print 'Hello' on line 3, column 10
    stdscr.refresh()         # Refresh the screen
    stdscr.getkey()          # Wait for user input

wrapper(main)
```

Handle User Inputs

```
def main(stdscr):
    y, x = 5, 20 # initial position of the string
    while True:
        stdscr.clear()
        stdscr.addstr(y, x, "Hello")
        c = stdscr.getch()
        if c == curses.KEY_UP:
            y = max(0, y - 1)
        elif c == curses.KEY_DOWN:
            y = min(y + 1, curses.LINES - 1)
        elif c == curses.KEY_LEFT:
            x = max(0, x - 1)
        elif c == curses.KEY_RIGHT:
            x = min(x + 1, curses.COLS - 1)
```

- `getch()` returns an integer, which is **ASCII code** of the key pressed
- `getkey()` returns a string, such as `p` and `KEY_UP`
- **Example:** print the string in a new location based on key pressed

Multi-threading

- For network applications that want to use **curses** to implement the UI, multi-threading has to be used.
- One thread should handle input and output with **curses** (a **UI thread**), while other threads are used to handle network communications
- Use a **queue** or a **pipe** to enable communication between different threads
- Example: [curses_threading_01.py](#).

GUI Programming with Tkinter

- [`tkinter`](#) (stands for "Tk interface") is the standard Python interface to the [Tk GUI toolkit](#))
- Available on most Unix platforms and Windows systems
- Use the following command to check if `tkinter` is properly installed on your system:

```
$ python3 -m tkinter
```

- If it is properly installed, you will see a window opened with two buttons inside.
- Note: there are other libraries for GUI programming in Python, check the following link:
<https://wiki.python.org/moin/GuiProgramming>

GUI Example

- The following example will display a window with a **text box**, with two lines of text in it

```
import tkinter

# Create the main window of the application
root = tkinter.Tk()

# Create a text box which is added to the main window (its container)
text = tkinter.Text(root, height=20, width=100)

# Control where the text box should appear in the main window
text.pack()

# Insert some text into the text box
text.insert(tkinter.END, "Line 1\nLine 2\n")

# Starts the event loop of the GUI
tkinter.mainloop()
```

GUI Example 2

```
class Application(tkinter.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = tkinter.Button(self, text="Click Me", command=self.say_hi)
        self.hi_there.pack(side="top")
        self.quit = tkinter.Button(self, text="QUIT", command=root.destroy)
        self.quit.pack(side="bottom")

    def say_hi(self):
        print("hi there, everyone!")

app = Application(master=tkinter.Tk())
app.mainloop()
```

Multi-threading with GUI

- The `mainloop()` function starts an **event loop** for the GUI, it only takes care of things related to the user interface
- **Avoid** doing tasks that take a long time to run (e.g. anything longer than 500ms), otherwise the user cannot interact with the UI properly
- If you have any networking or file I/O operation, do it in **another thread**
- Use a **queue** to exchange data whenever possible
- You should always run the GUI event loop in the **main thread**
- Example: [gui_multithreading.py](#).
- References:
 - <https://docs.python.org/3.5/library/tkinter.html>
 - <https://wiki.python.org/moin/TkInter>

End of Lecture 10