CS196

#13

Announcements

No Homework!

Midterm Extended

Today

Trees

Networking

CS196

Trees

Max Howell
@mxcl

Follow

Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so fuck off.

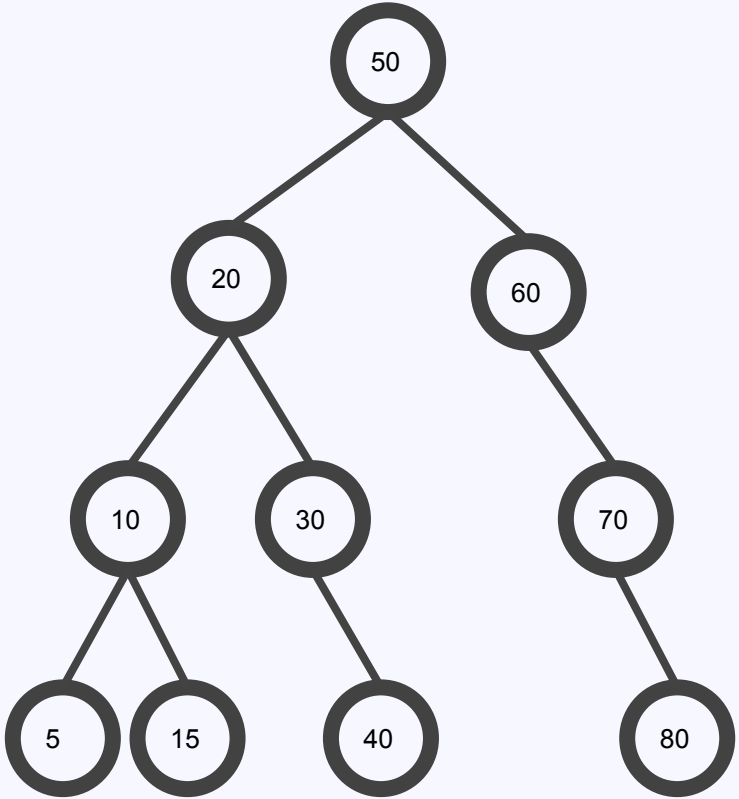
RETWEETS	LIKES
6,808	7,630

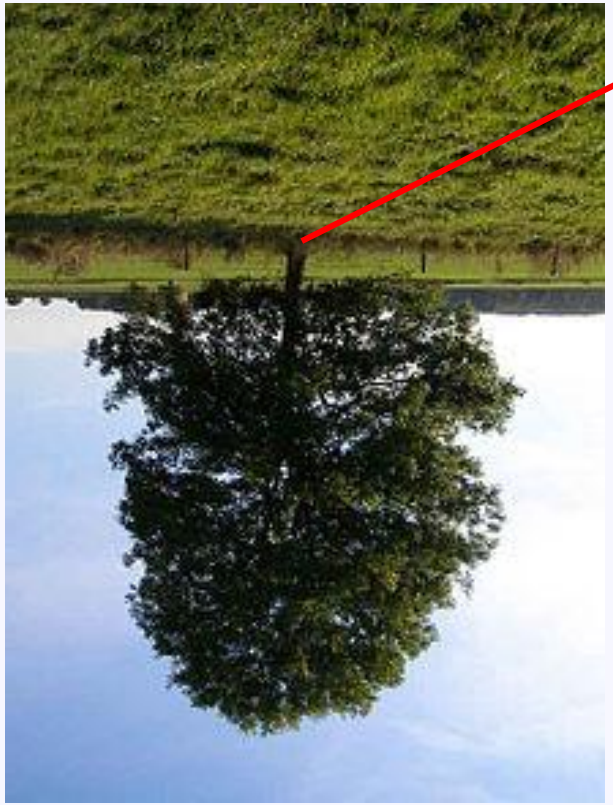
10:37 PM - 10 Jun 2015

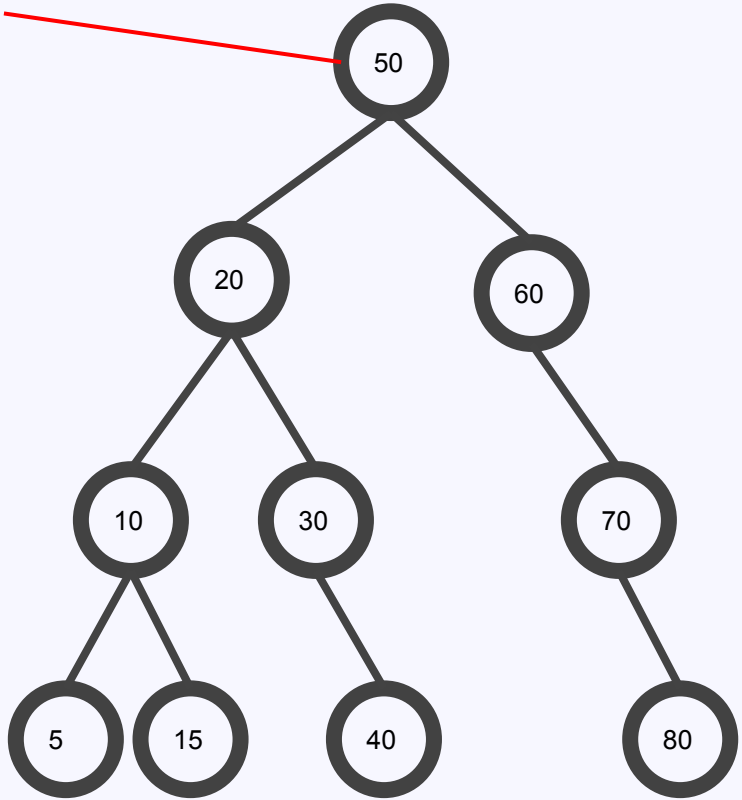
6.8K 7.6K

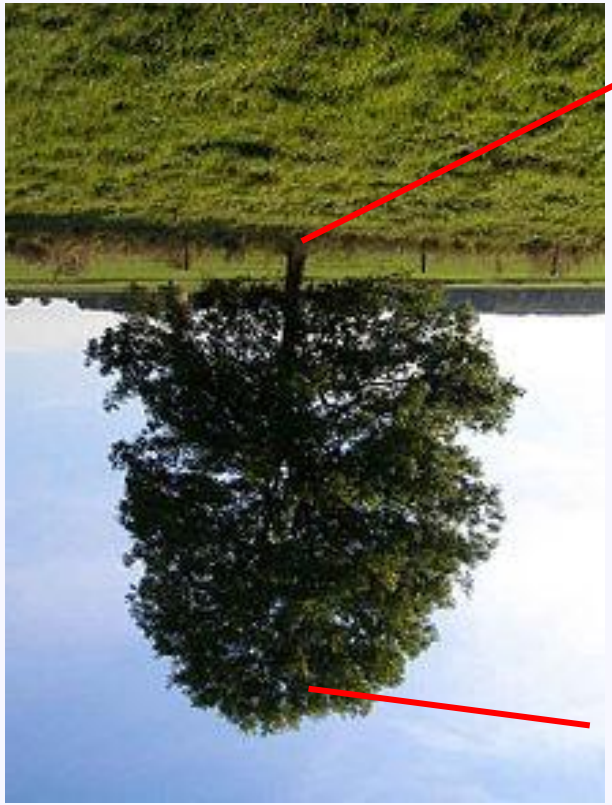
Trees IRL

Trees in CS


50
20
60
10
30
70
5
15
40
80

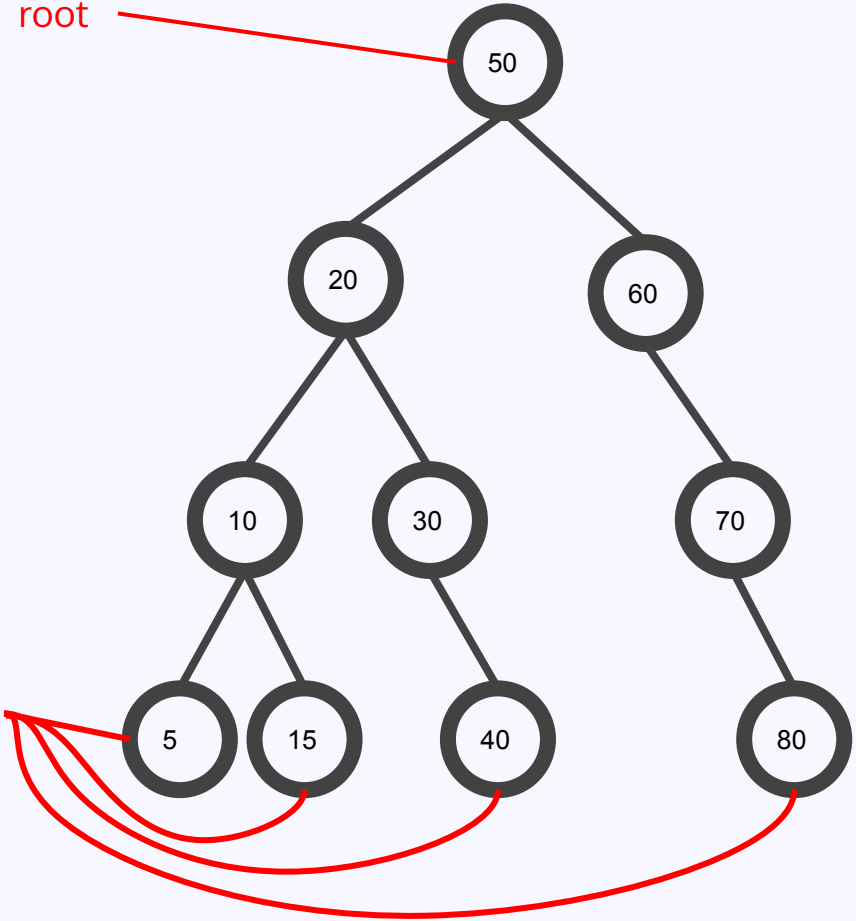


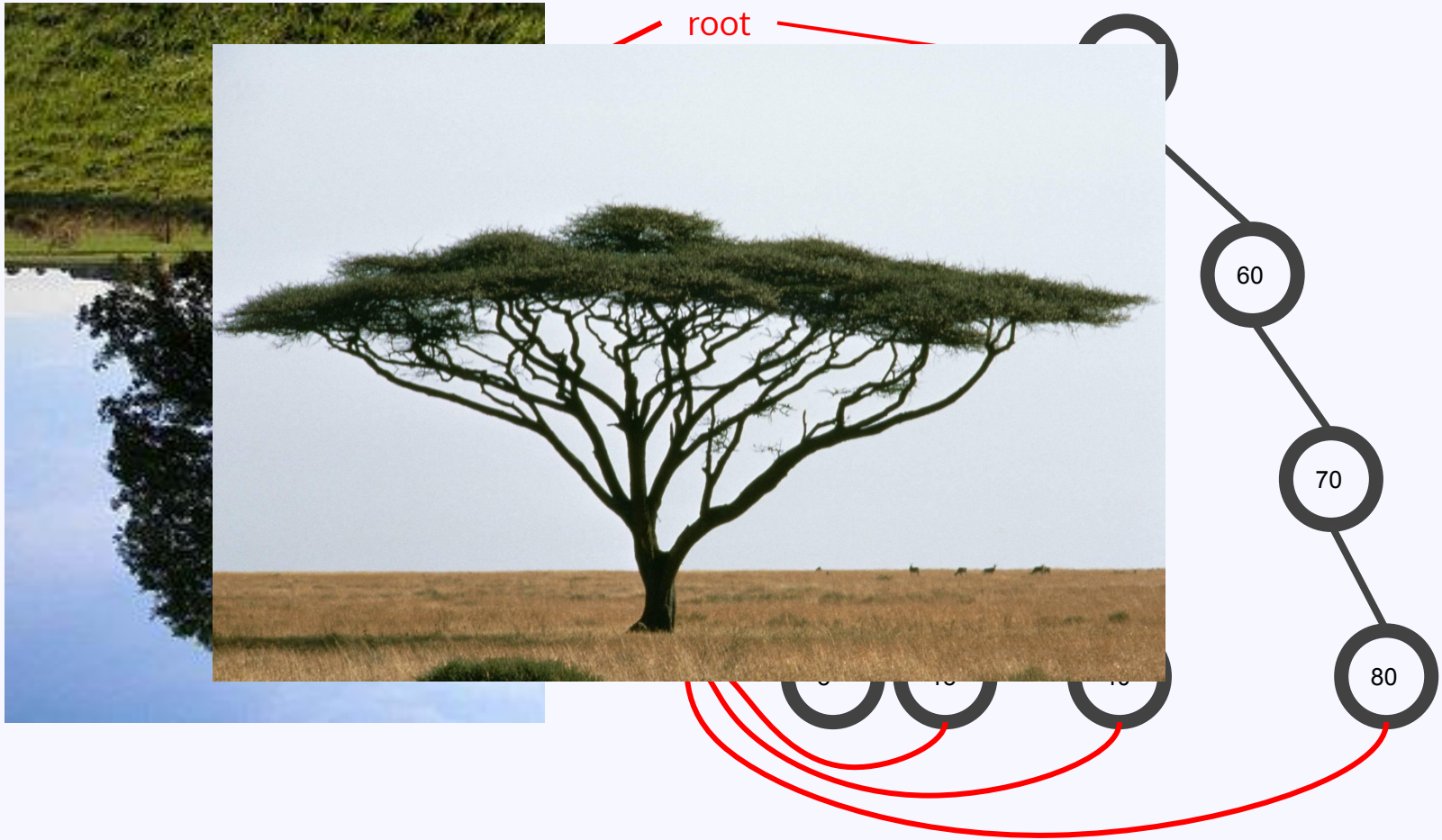

root
50
20
60
10
30
70
5
15
40
80

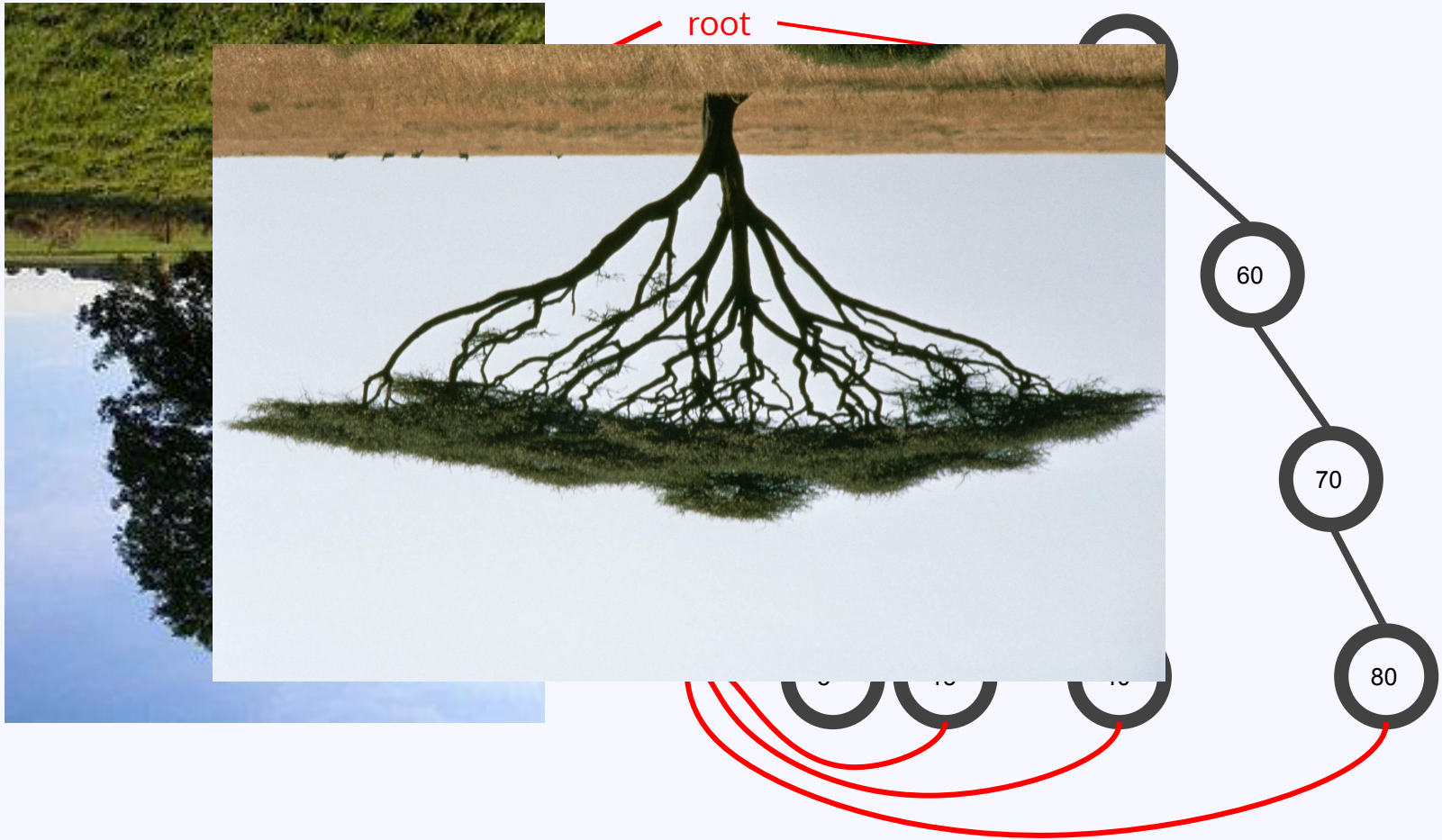


root

leaves

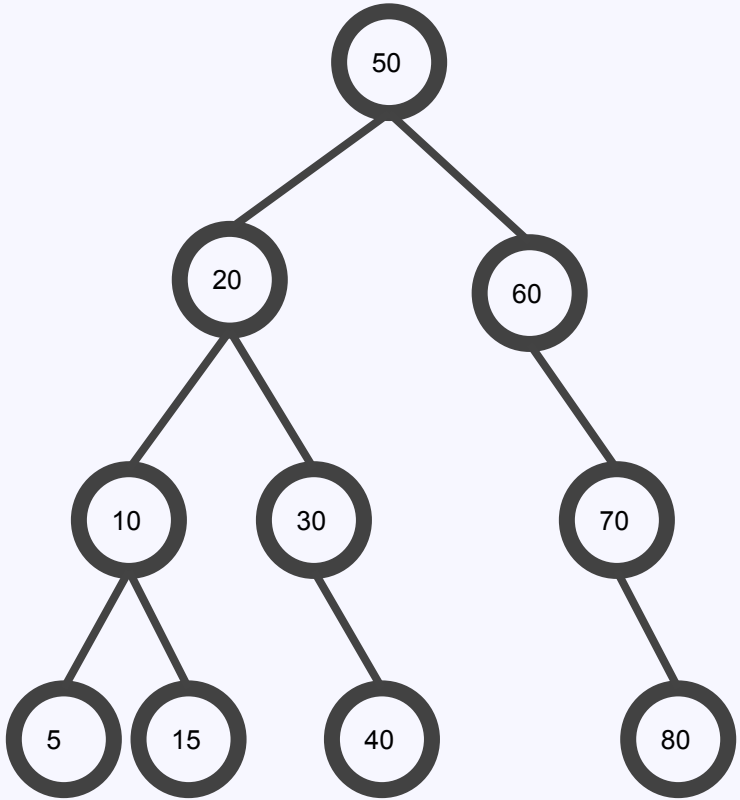



root
60
70
80


root
60
70
80

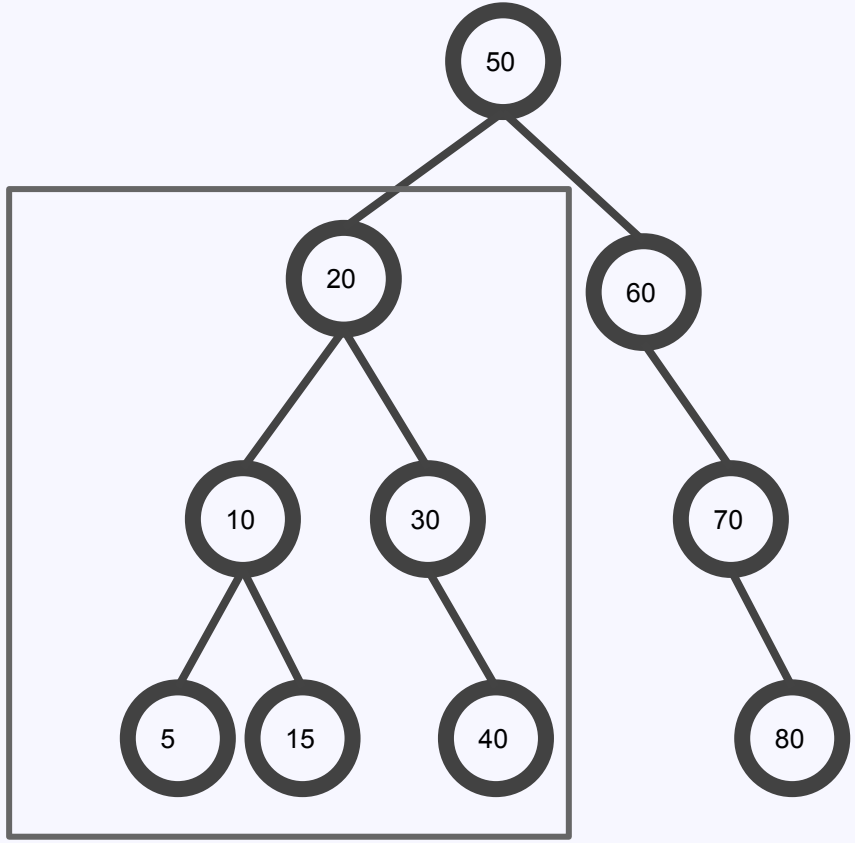
A Tree is either:

- Nothing
- A root with *Sub-trees*



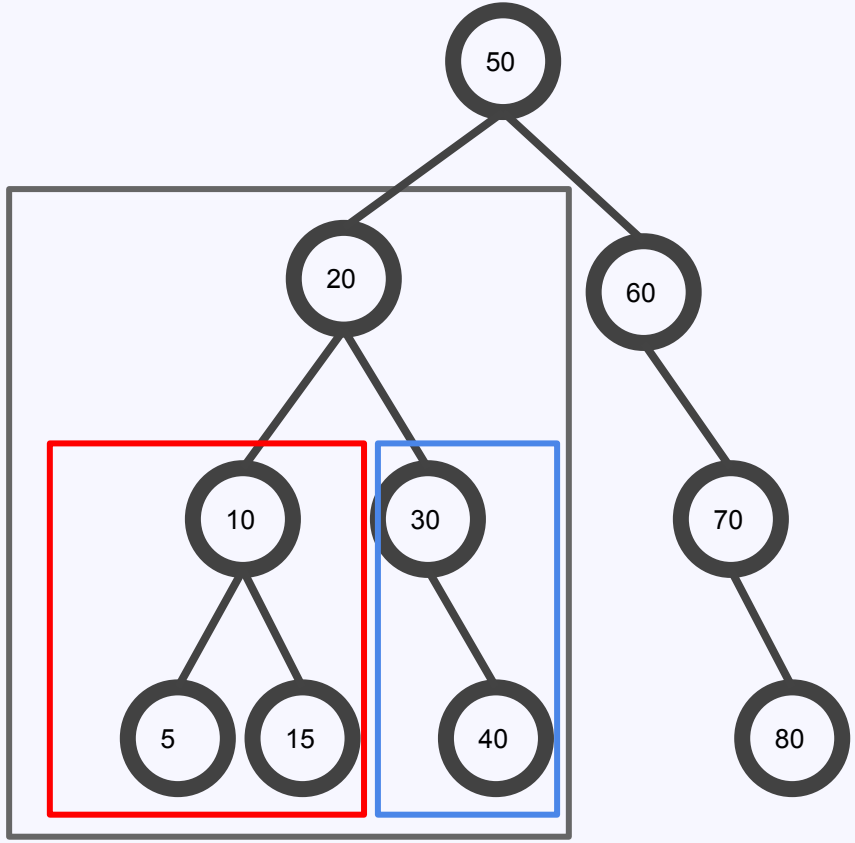
A Tree is either:

- Nothing
- A root with *Sub-trees*



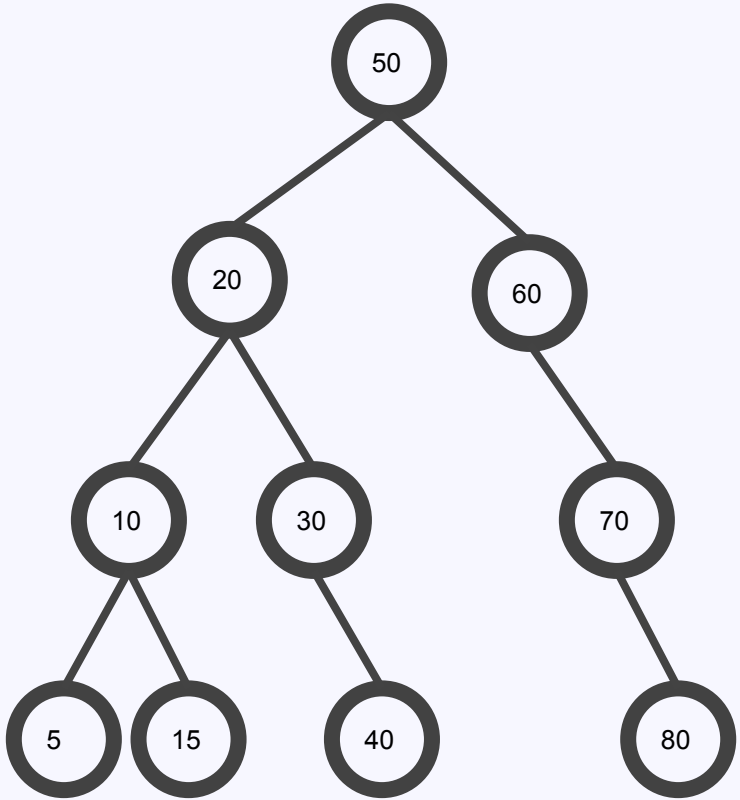
A Tree is either:

- Nothing
- A root with *Sub-trees*



An N-Ary Tree is either:

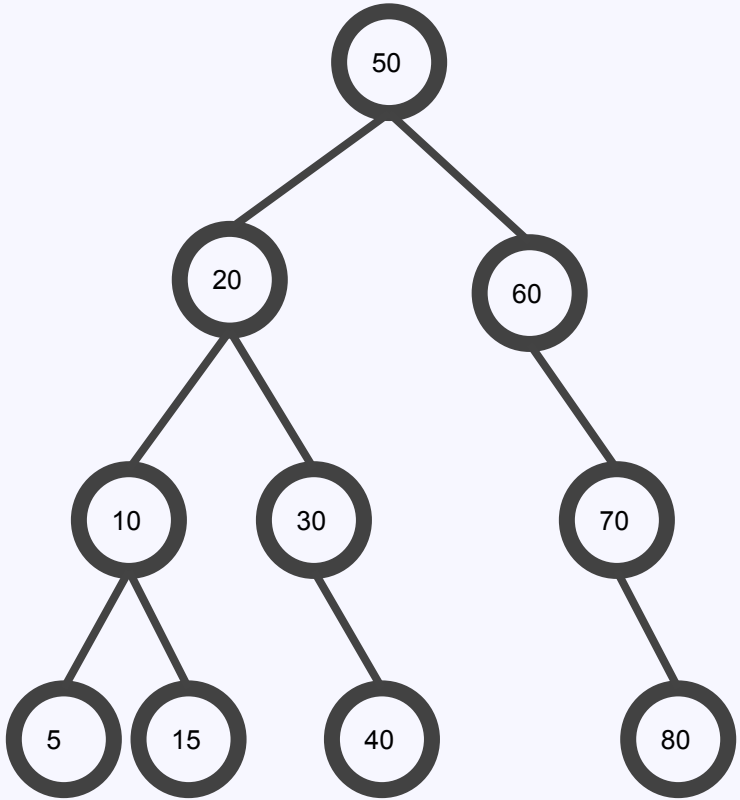
- Nothing
- A root with at most N Subtrees



An N-Ary Tree is either:

- Nothing
- A root with at most ~~N~~ 2 Subtrees

Binary Tree



Tree Code

```
class Tree {

     sometype value;

     Tree right;

     Tree left;

}
```

Tree Code

```
recursive_funct (root, …..):

     if root.value is something:
          # do something

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     return Something
```

Searching

```
binary_search (root, value):

     if root.value == value:
          return root

     if root.value > value:
          temp = binary_search(root.left, value)
          if temp: return temp

     if root.value < value:
          temp = binary_search(root.right, value)
          if temp: return temp

     return None
```

Pre Order Traversal

```
recursive_funct (root, …..):

     if root.value is something:
          # do something

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     return Something
```

In Order Traversal

```
recursive_funct (root, …..):

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.value is something:
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     return Something
```

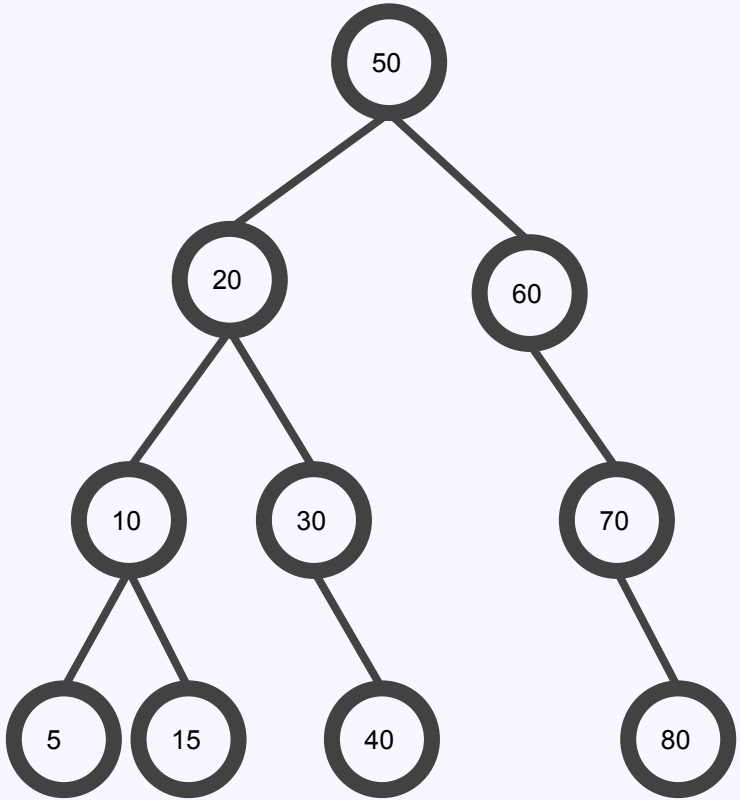
Post Order Traversal

```
recursive_funct (root, …..):

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     if root.value is something:
          # do something

     return Something
```

Pre Order Traversal

```
recursive_funct (root, …..):

     if root.value is something:
          # do something

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     return Something
```

Pre Order
50 20 10 5 15 30 40 60 70 80



In Order Traversal

```
recursive_funct (root, …..):

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.value is something:
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     return Something
```

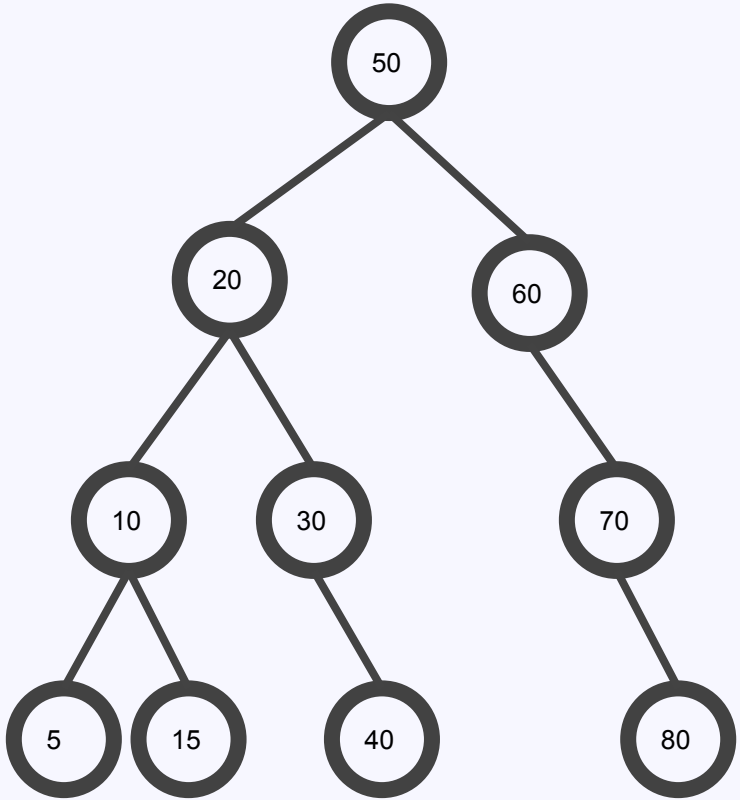
In Order Traversal

```
1) Create an empty stack S.
2) Initialize current node as root
3) Push the current node to S and set current =
current->left until current is NULL
4) If current is NULL and stack is not empty then
     a) Pop the top item from stack.
     b) Print the popped item, set current =
popped_item->right
     c) Go to step 3.
5) If current is NULL and stack is empty then we are
done.
```

In Order Traversal

```
recursive_funct (root, …..):

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.value is something:
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     return Something
```

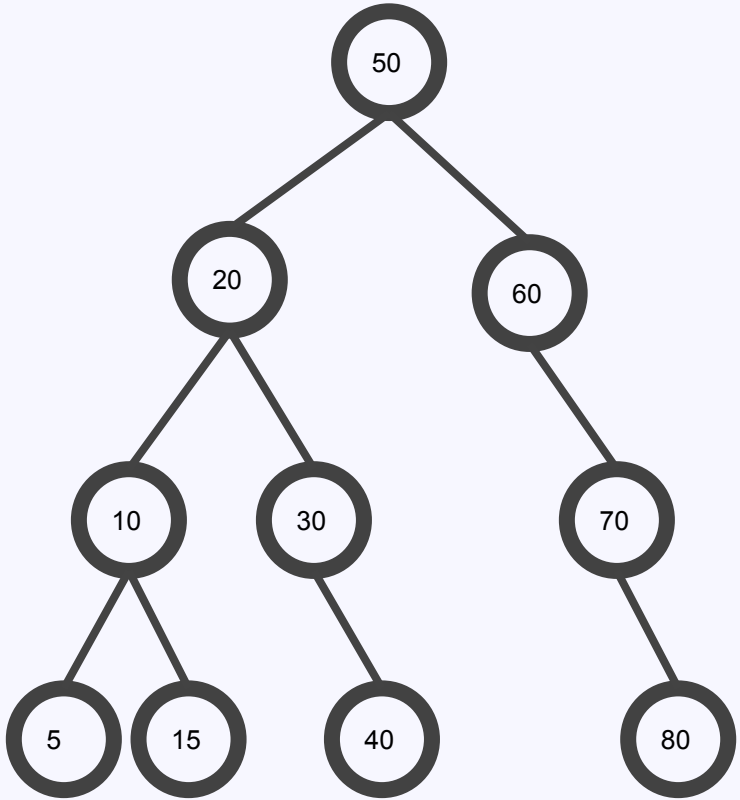
In Order
5 10 15 20 30 40 50 60 70 80



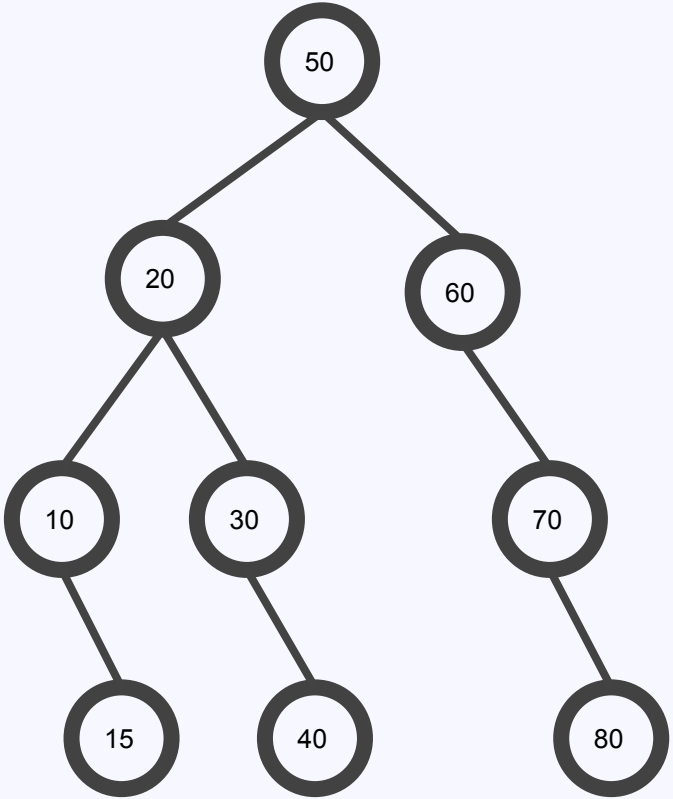
Post Order Traversal

```
recursive_funct (root, …..):

     if root.left:
          temp = recursive_funct(root.left, value)
          # do something

     if root.right:
          temp = recursive_funct(root.right, value)
          # do something

     if root.value is something:
          # do something

     return Something
```

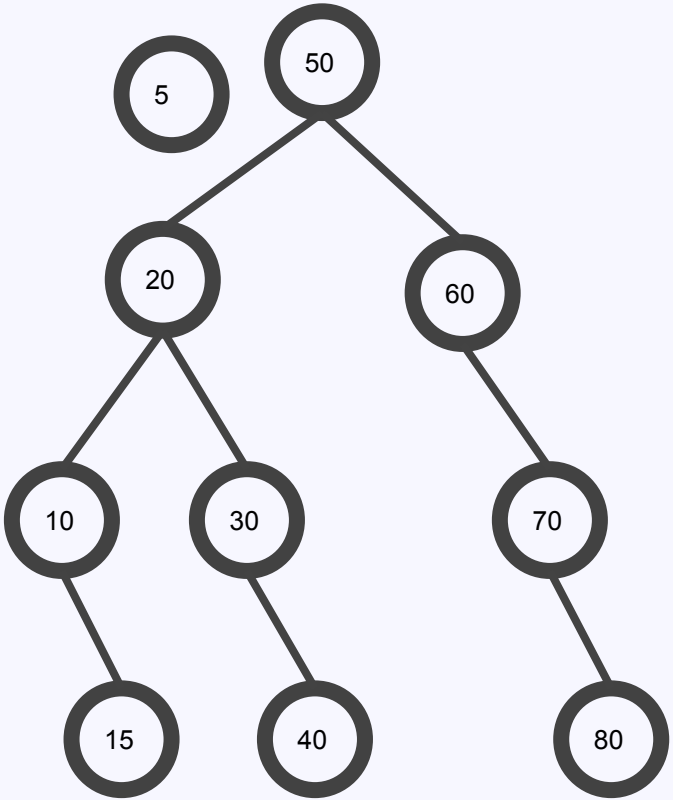
Post Order
5 15 10 40 30 20 80 70 60 50



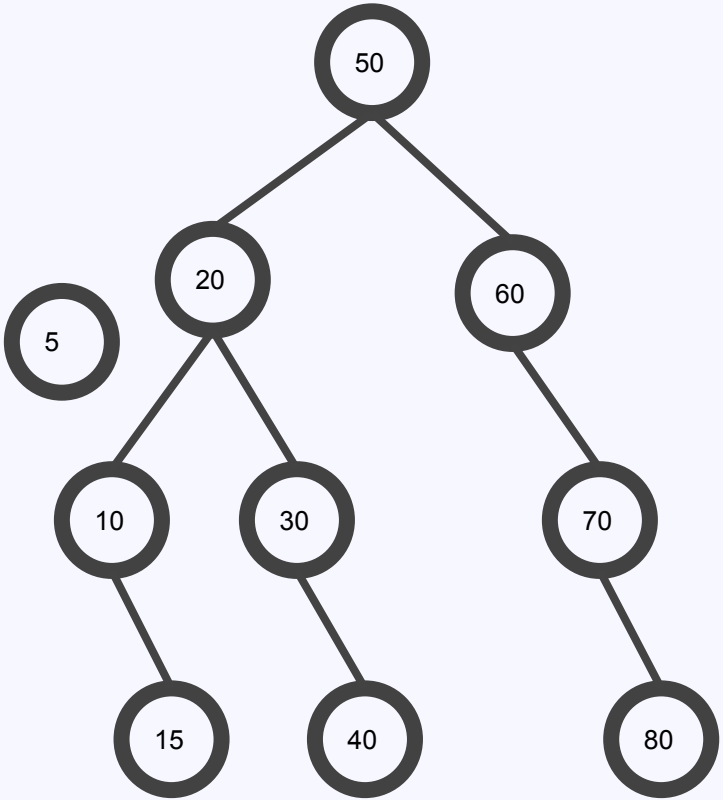
Insertion



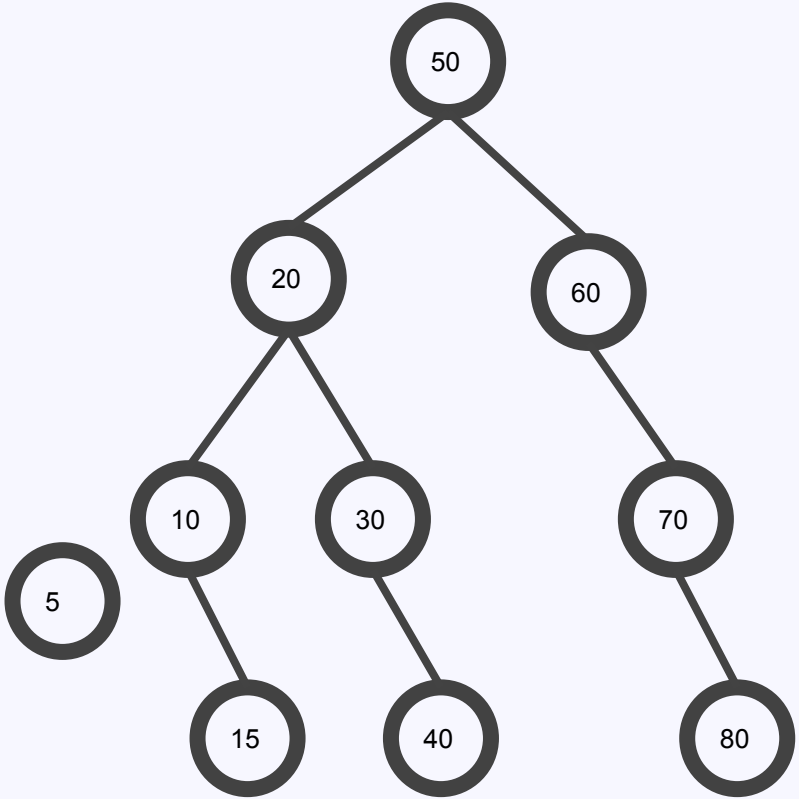
Insertion



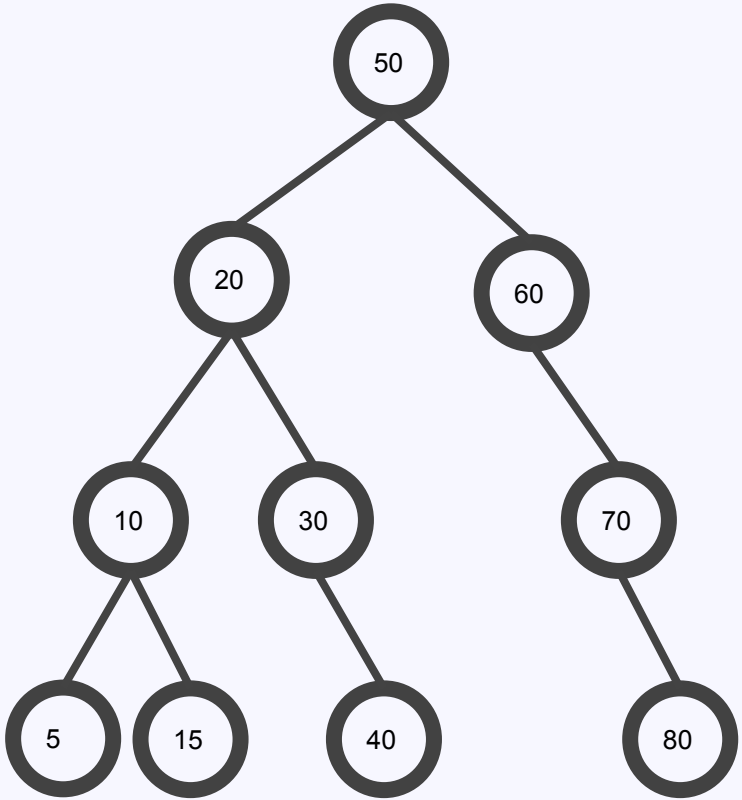
Insertion



Insertion



Insertion



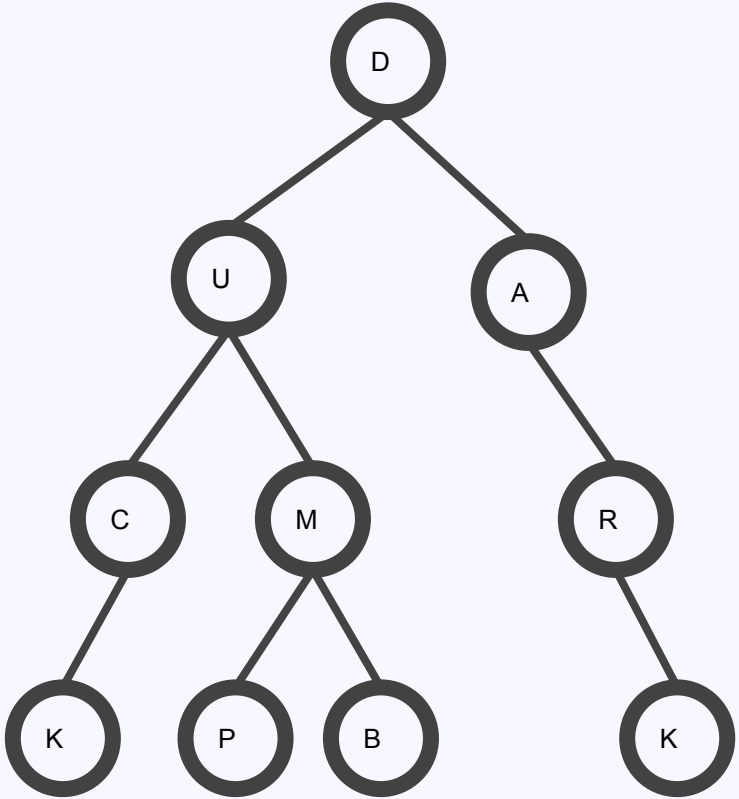
Recursion?

So what are trees good for?

CS196

Tries / Prefix Trees

Simple Auto Complete



Questions?

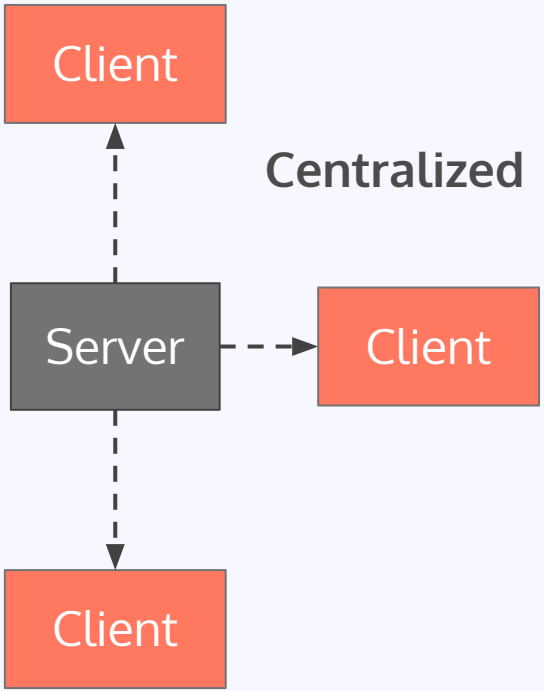
CS196

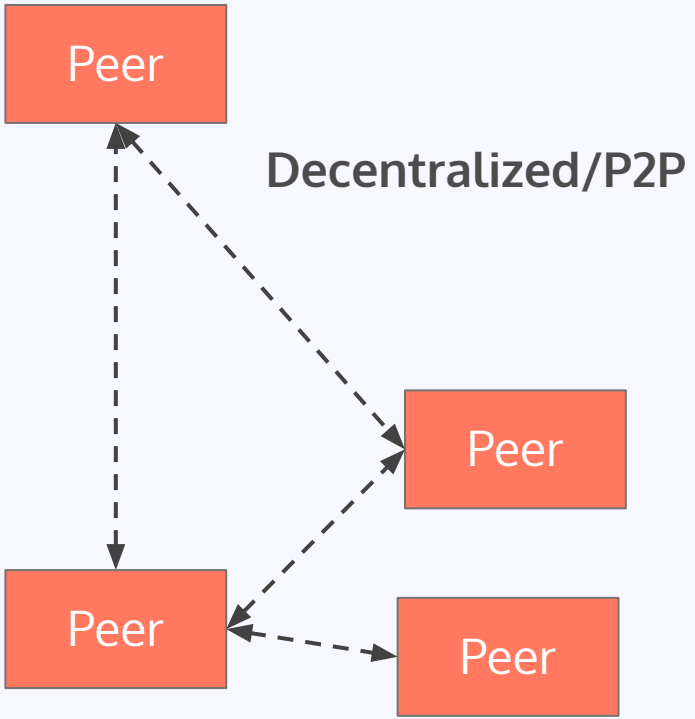
Networking

Have you heard the story of TCP/UDP?

I thought not. It's not a story the web devs would tell you.

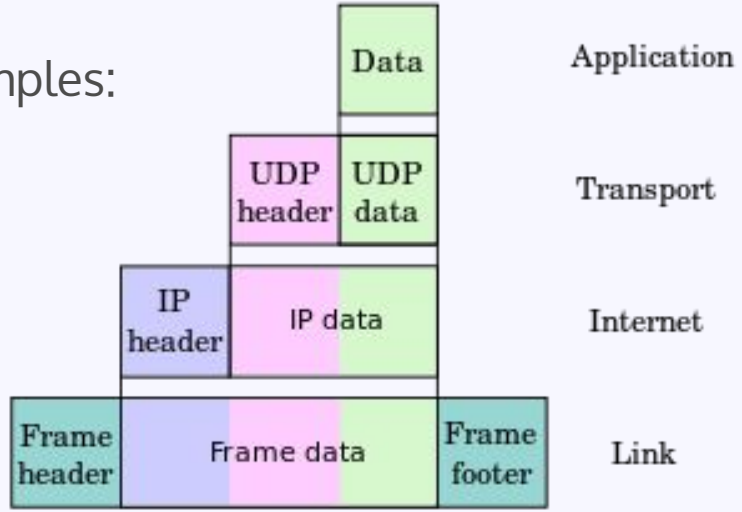
It's a systems programming legend.


Client
Centralized
Server
Client
Client


Peer
Decentralized/P2P
Peer
Peer
Peer

Internet Protocol Suite

- Hierarchy of abstractions in the Internet
- TCP/UDP are in the Transport Layer
- Applications are in highest layer. Examples:
  - FTP
  - SSH
  - TELNET
  - SMTP
  - HTTP
- Internet layer is IP protocols
- Link layer uses physical connections



TCP vs UDP

- Requires handshake to start connection
- Bi-directional communication
- Multiple attempts to deliver data
- Strict ordering with buffering
- Data read as continuous byte stream instead of messages

- Sends first message immediately
- One-directional communication
- No concept of acknowledgment
- No enforced message arrival order
- Packets sent and received individually
- Can multicast packets

Why use lower-level protocols?

- High performance
- Low bandwidth
- Less overhead
- Less server resource consumption

Network sockets

- User-friendly abstraction of networking
- Objects identified by
  - Transmission protocol (UDP, TCP, or raw IP)
  - IP address and a port number
- Generally found in standard libraries across many languages
- `socket` library in Python2 and Python3
- Some languages bake in a client-server relationship
  - `socket` - generally functions as the client
  - `SocketServer` - generally functions as the server

Applications

- Distributed Systems
- Embedded Systems
- Internet of Things
- Peer-to-Peer Networking
- Federated Learning
- Sensor Networks

Interested?

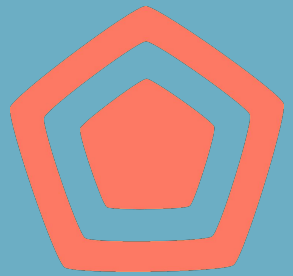
Let's talk after lecture

Attendance

(Reprise)

https://github.com/cs196illinois/sockets-lecture.git

192.17.96.7:10000



Keyword

Turkey

https://goo.gl/iYigpc