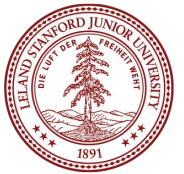
Programming Languages, Graphs, and more!

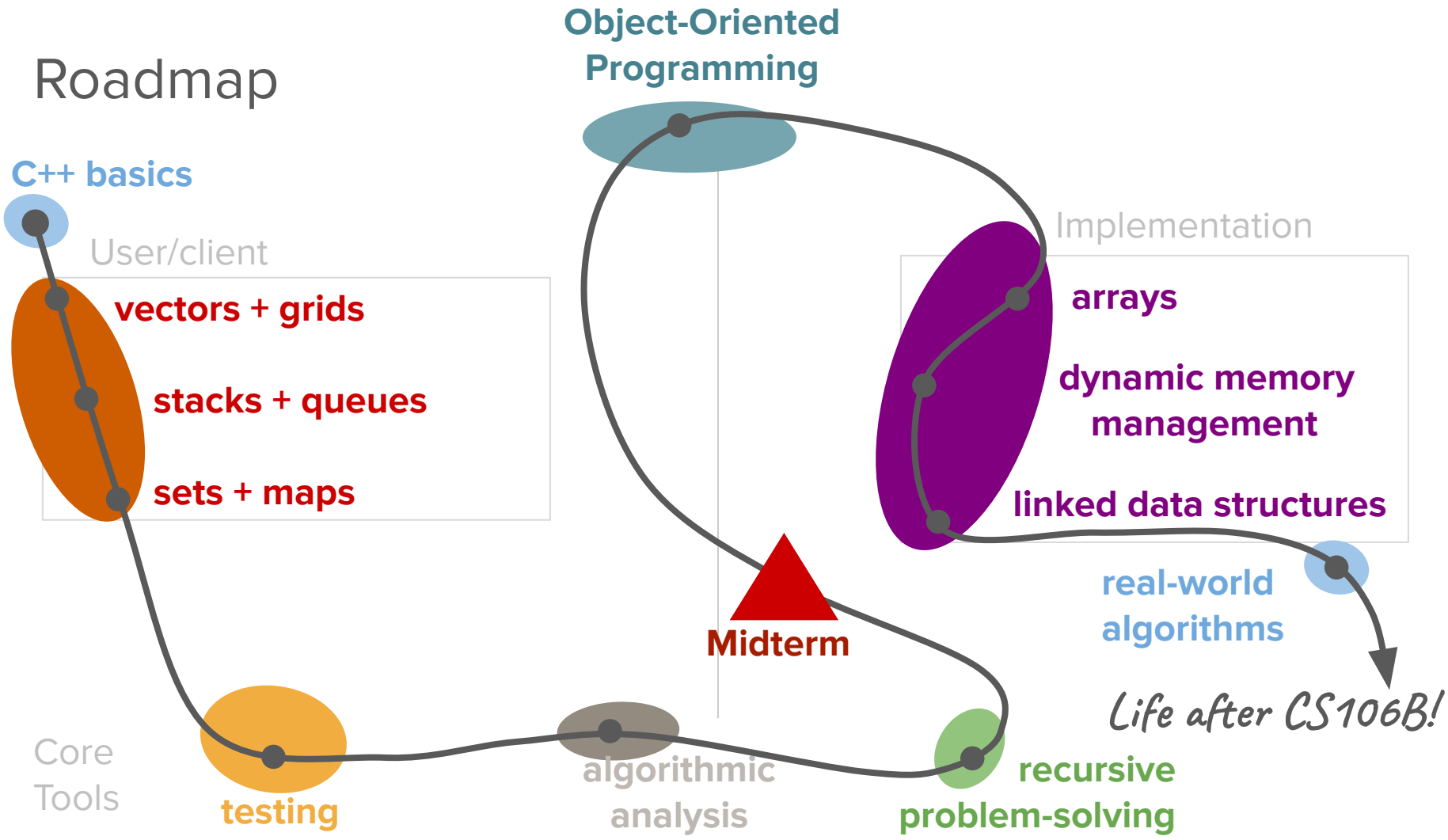
What was your favorite part about working on your final project?

PollEv.com/cs106bpolls

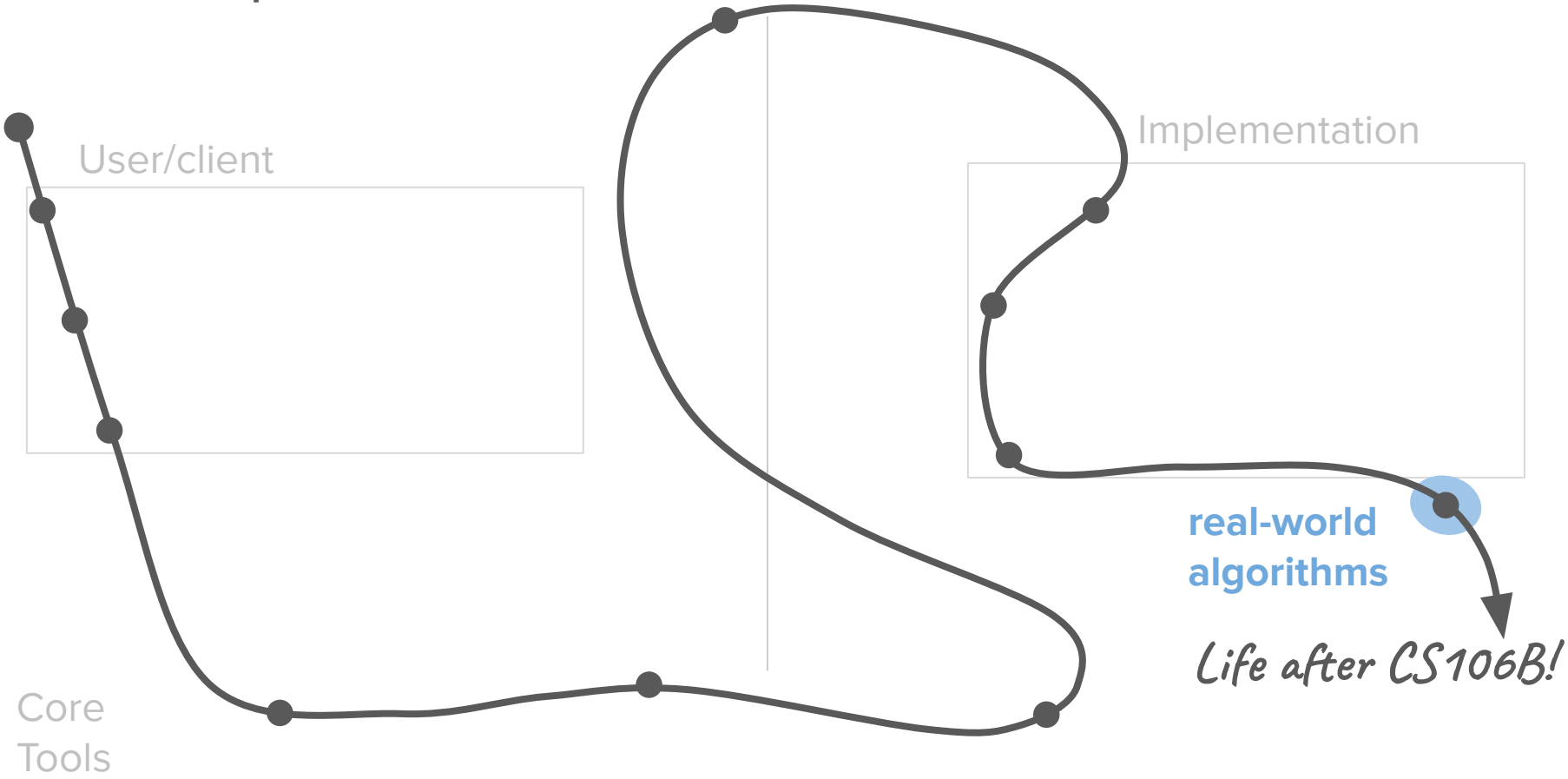


What was your favorite part about working on your final project?

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Midterm
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


User/client
Implementation
real-world
algorithms
Life after CS106B!
Core
Tools

Today's questions....by popular vote!

Why is programming language design like high fashion?

How can we represent real-world systems of connected components?

How do you start your own C++ projects?

Today's topics

1. Programming Languages
2. Graphs
3. Making your own C++ projects

Programming Language Design

C++ is a “a general-purpose programming language” with “object-oriented, generic, and functional features”

What does that mean?
Why is C++ this way?
What were the alternatives??

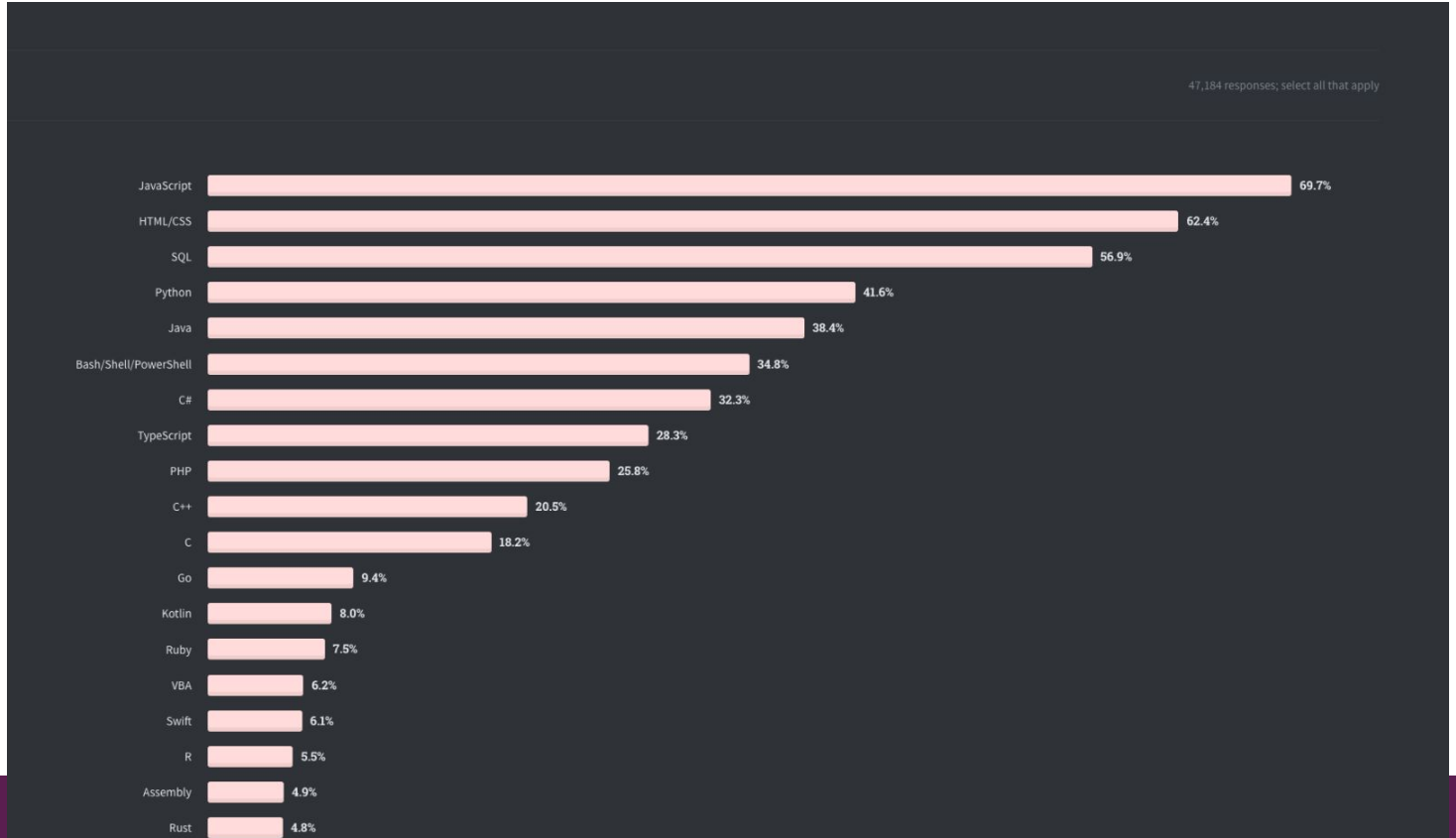
YES THERE WERE ALTERNATIVES

Shout out to Will Crichton who let us adapt the following slides from CS242!

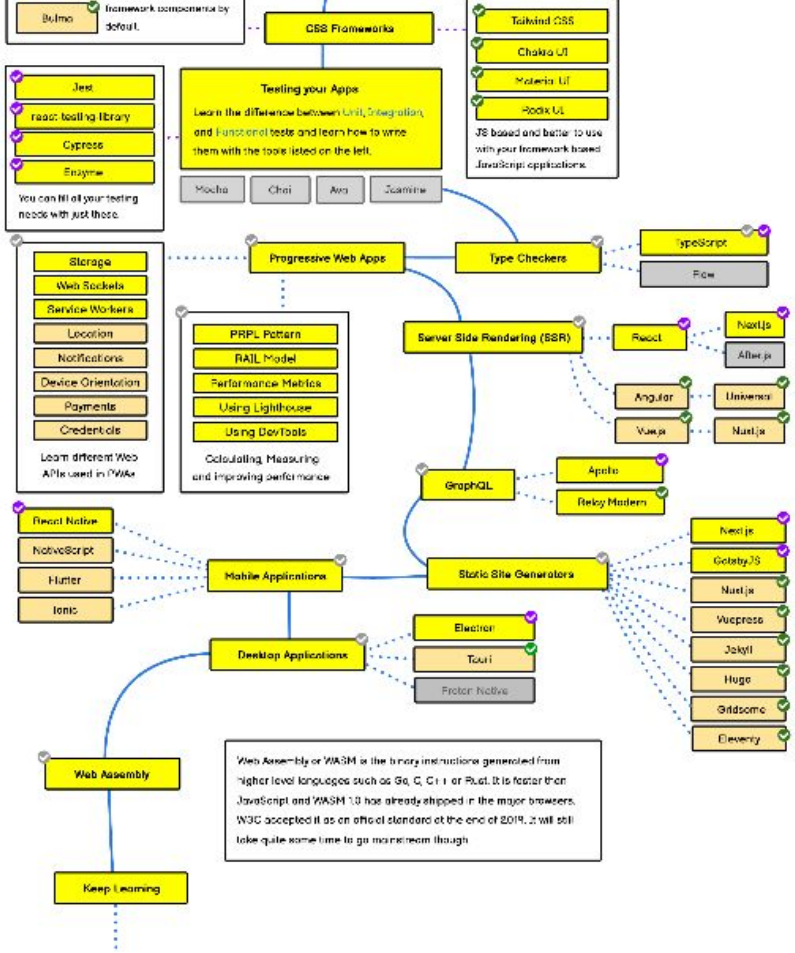
Guess the most popular programming language (for professional developers) in 2020

Guess the most popular programming language!

It’s Javascript! (Stack Overflow, 2020)



Software


Bulma
CSS Frameworks
Tailwind CSS
Chakra UI
Material UI
Radix UI
Jest
react-testing-library
Cypress
Enzyme
You can fill all your testing needs with just these.
Testing your Apps
Learn the difference between Unit, Integration, and Functional tests and learn how to write them with the tools listed on the left.
Mocha
Chai
Ava
Jasmine
Storage
Web Sockets
Service Workers
Location
Notifications
Device Orientation
Payments
Credentials
Learn different Web APIs used in PWAs
Progressive Web Apps
PRPL Pattern
RAIL Model
Performance Metrics
Using Lighthouse
Using DevTools
Calculating, Measuring and Improving performance
Type Checkers
TypeScript
Flow
Server Side Rendering (SSR)
React
NextJs
Afterjs
Angular
Universal
Vuejs
Nuxt.js
GraphQL
Apollo
Relay Modern
React Native
NativeScript
Flutter
Ionic
Mobile Applications
Static Site Generators
Next.js
GatsbyJS
Nuxt.js
Vuepress
Jekyll
Hugo
Gridsome
Eleventy
Desktop Applications
Electron
Tauri
Proton Native
Web Assembly
Web Assembly or WASM is the binary instructions generated from higher level languages such as Go, C, C++ or Rust. It is faster than JavaScript and WASM 1.0 has already shipped in the major browsers. W3C accepted it as an official standard at the end of 2019. It will still take quite some time to go mainstream though.
Keep Learning

std::move_if_noexcept

Defined in header <utility>

```
template< class T >
typename std::conditional<
    !std::is_nothrow_move_constructible<T>::value && std::is_copy_constructible<T>::value,
    const T&,
    T&&
>::type move_if_noexcept(T& x) noexcept;
```
(since C++11) (until C++14)

```
template< class T >
constexpr typename std::conditional<
    !std::is_nothrow_move_constructible<T>::value && std::is_copy_constructible<T>::value,
    const T&,
    T&&
>::type move_if_noexcept(T& x) noexcept;
```
(since C++14)

`move_if_noexcept` obtains an rvalue reference to its argument if its move constructor does not throw exceptions or if there is no copy constructor (move-only type), otherwise obtains an lvalue reference to its argument. It is typically used to combine move semantics with strong exception guarantee.

Parameters

x - the object to be moved or copied

Return value

`std::move(x)` or `x`, depending on exception guarantees.

Notes

This is used, for example, by `std::vector::resize`, which may have to allocate new storage and then move or copy elements from old storage to new storage. If an exception occurs during this operation, `std::vector::resize` undoes everything it did to this point, which is only possible if `std::move_if_noexcept` was used to decide whether to use move construction or copy construction. (unless copy constructor is not available, in which case move constructor is used either way and the strong exception guarantee may be waived)

Example

Run this code

```
#include <iostream>
#include <utility>

struct Bad
{
    Bad() {}
    Bad(Bad&&)  // may throw
    {
        std::cout << "Throwing move constructor called\n";
    }
```

Guess the most popular programming language in 1952

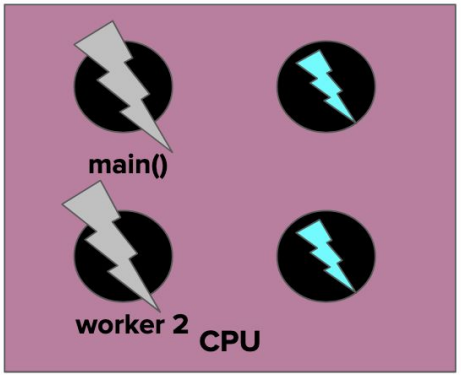
Guess again!

None! Only assembly languages

```
MOV AL, 1h        ; Load AL with immediate value 1
MOV CL, 2h        ; Load CL with immediate value 2
MOV DL, 3h        ; Load DL with immediate value 3
```

The syntax of MOV can also be more complex as the following examples show.[24]

```
MOV EAX, [EBX]    ; Move the 4 bytes in memory at the address contained in EBX into EAX
MOV [ESI+EAX], CL ; Move the contents of CL into the byte at address ESI+EAX
MOV DS, DX        ; Move the contents of DX into segment register DS
```



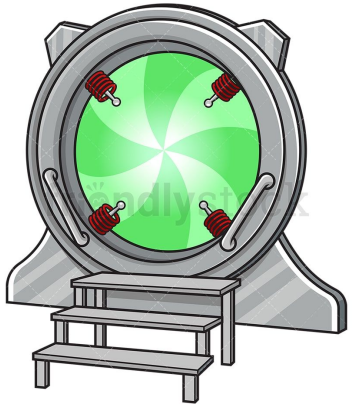
And before that… raw numeric machine codes!!!

Grace Hopper
HOPL Keynote, 1978

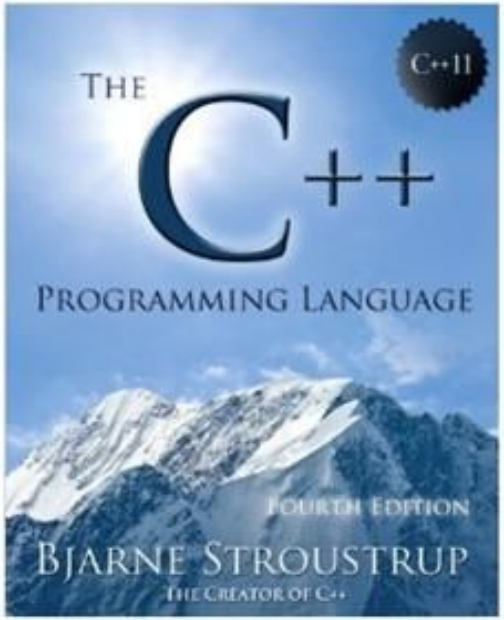
On assembly:

I think I spent 20 years fighting the "Establishment." In the early years of programming languages, the most frequent phrase we heard was that the only way to program a computer was in octal. Of course, a few years later a few people admitted that maybe you could use assembly language.

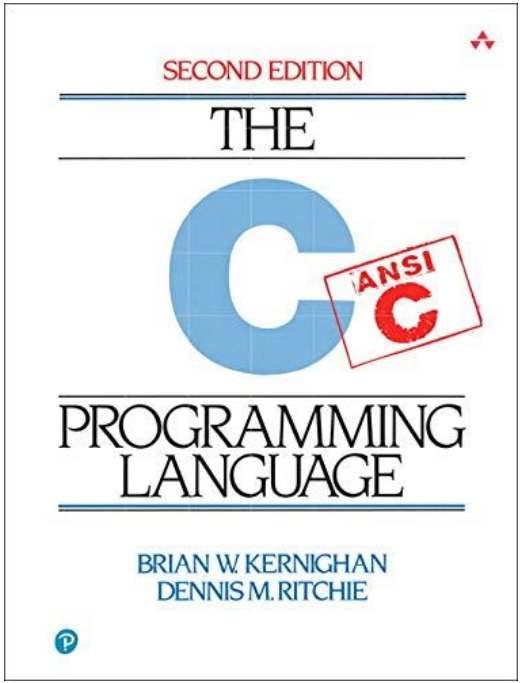
How did we get to 2022?



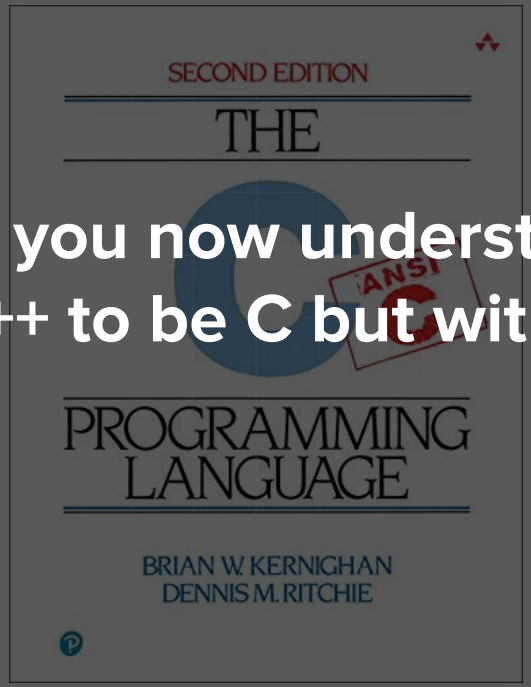
How did we get here? (C++, 1979)



How did we get here? (C, 1972)



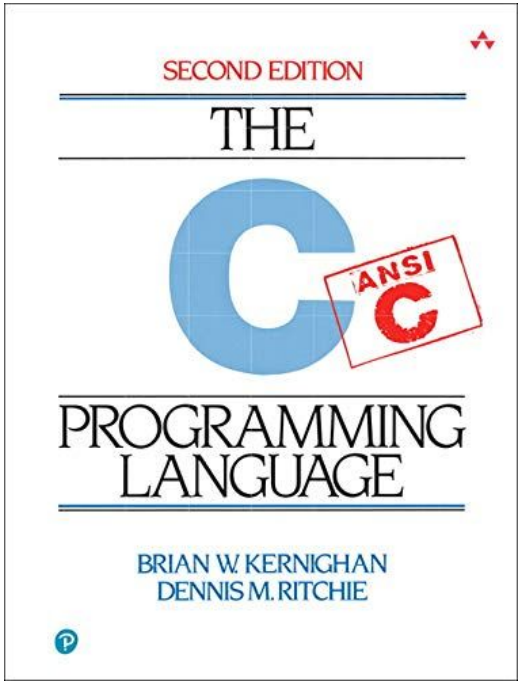
How did we get here? (1972)



Hey! Week 8 you now understands what it means for C++ to be C but with classes!

😎

How did we get here? (C, 1972)



How did we get here? (B, 1969)

Article | Talk | Read | Edit | View history

B (programming

From Wikipedia, the free encyclopedi

This article is about a languag *berton, see ABC (programming language). For c*

B is a programming language dev *chie*.

B was derived from BCPL, and its *rker Dennis Ritchie speculated that the name might be based on Bon, an earlier* *for use on Multics.*[note 1]

B was designed for recursive, non *d language software.*[3] *It was a typeless language, with the only data type* *tever that might be. Depending on the context, the word was treated either as an integer or a memory address.*

As machines with ASCII processing became common, notably the DEC PDP-11 that arrived at Bell, support for character data stuffed in memory words became important. The typeless nature of the language was seen as a disadvantage, which led Thompson and Ritchie to develop an expanded version of the language supporting new internal and user-defined types, which became the C programming language.

Contents [hide]

1 History
2 Examples
3 See also

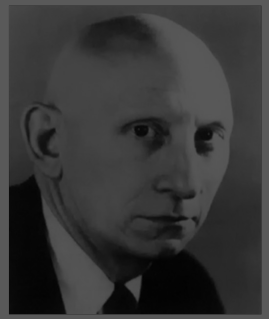
How did we get here? (Algol, 1959)

Alan Perlis,
"The American Side of the Development of Algol" 1978

Algol introduced into programming languages such terms as type, declaration, identifier, for statement, while, if then else, switch, the begin end delimiters, block, call by value and call by name, typed procedures, declaration scope, dynamic arrays, side effects, global and local variables.

Algol was strongly derived from FORTRAN and its contemporaries. The logic, arithmetic and data organizations were close to those then being designed into real computers. Certain simple generalizations of computer instructions such as switch, for statement, and if statements were included because their semantics and computer processing were straight-forward consequences of single statement processing.

How did we get here? (Algol, 1959)



Alan Perlis,
"The American Side of the Development of Algol" 1978

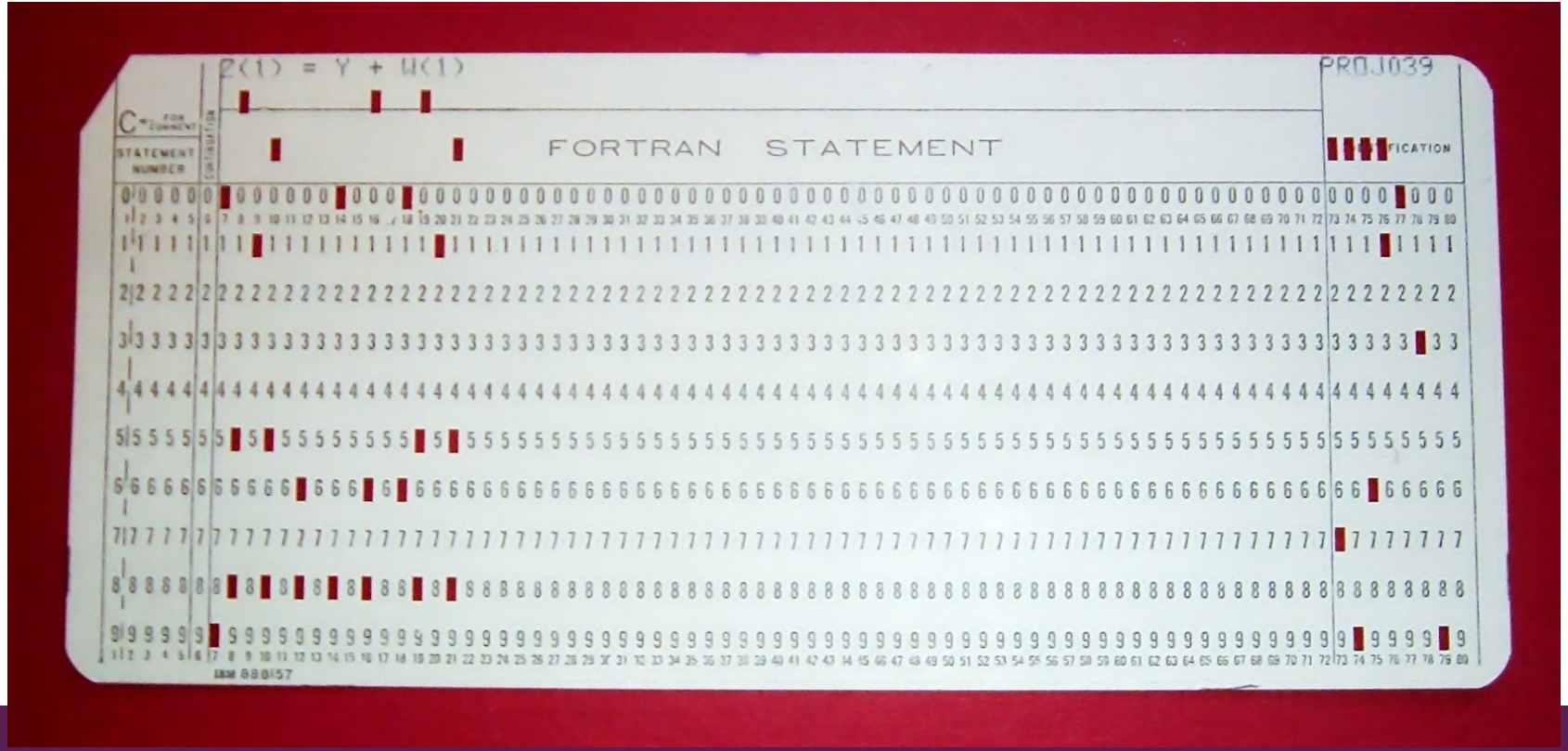
Algol introduced into programming languages such terms as type, declaration, identifier, for statement, while, if then else, switch, the begin end delimiters, block, call by value and call by name, typed procedures, declaration scope, dynamic arrays, side effects, global and local variables.

Algol was strongly derived from FORTRAN and its contemporaries. The logic, arithmetic and data organizations were close to those then being designed into real computers. Certain simple generalizations of computer instructions such as switch, for statement, and if statements were included because their semantics and computer processing were straight-forward consequences of single statement processing.

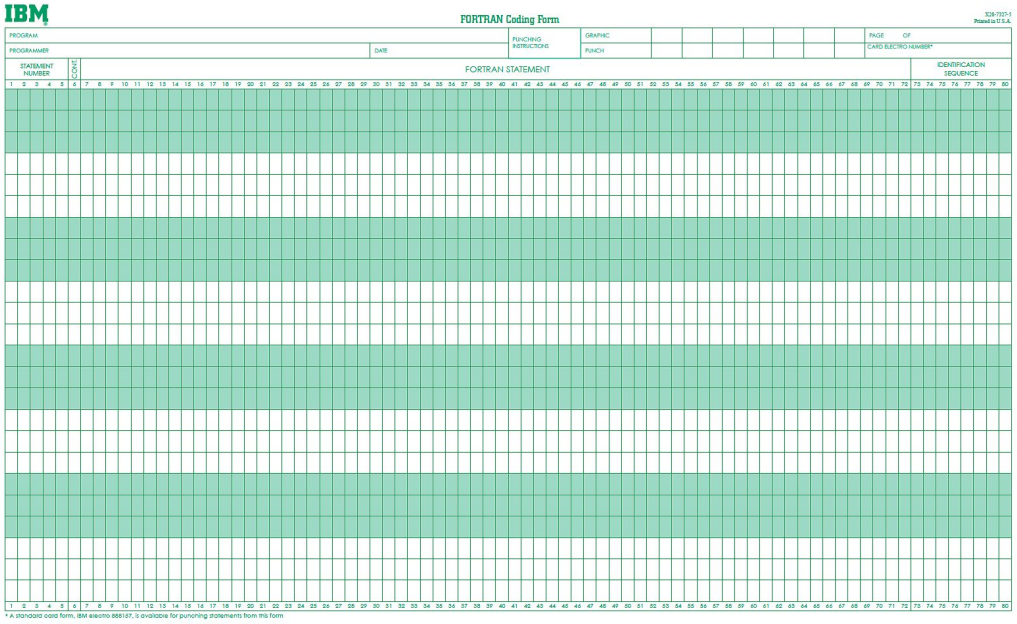
Type, variables, for-loops, dynamic arrays, while-loops, if-then-else, call by ref vs value...

All things we still use!

How did we get here? (Fortran, 1957)



The starter code:

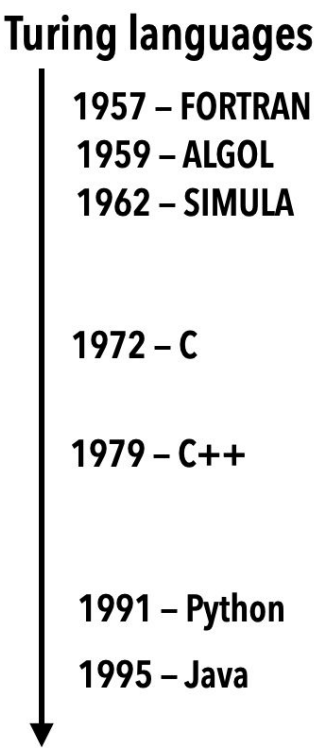


Fortran... kind of looks familiar!

```
1  PROGRAM main
2
3      IMPLICIT NONE
4
5      character(len = 15) :: hello = "Hello, World!"
6      print *, hello
7
8  END PROGRAM main
```

The big ideas in our programming languages haven't really changed since 1958.

statements, variable assignment, for loops, classes, and so on




It didn’t need to be this way!

And in fact, it hasn’t!

Turing languages

1957 – FORTRAN
1959 – COBOL, ALGOL
1962 – SIMULA

1972 – C, Smalltalk

1979 – C++

1991 – Python

1995 – Java

1959 – LISP

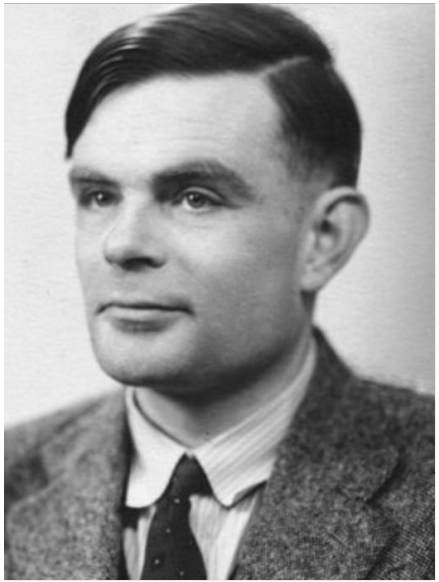
1966 – ISWIM

1972 – Prolog

1978 – ML

1990 – Haskell

Alan Turing



"On Computable Numbers" 1937

Alonzo Church

"A set of postulates for the foundation of logic", 1932

What do you think this does? What do you observe?

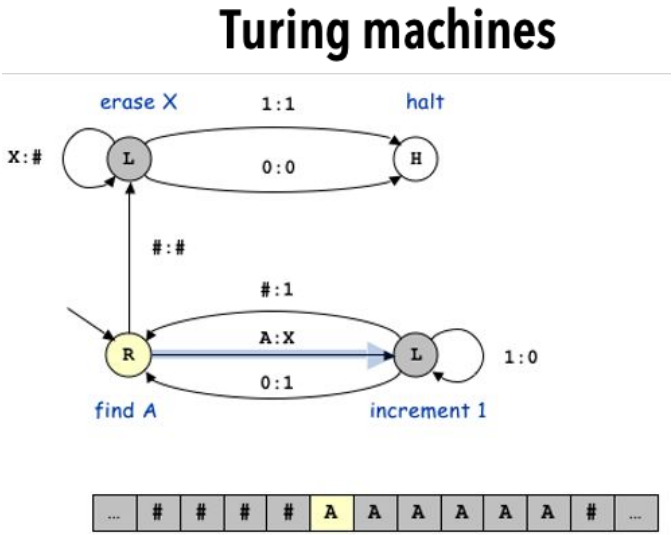
```
1  #lang racket
2
3  (define (extract str)
4    (substring str 4 7))
5
6  (extract "the cat out of the bag")
```

```
;; Push elem onto stack (default value null).
(define (push elem [stack null])
  (cons elem stack))

;; Pop item off stack, returning the stack after pop.
(define (pop stack)
  (unless (empty? stack)
    (rest stack)))

;; Peek at top element of stack, returning it.
(define (peek stack)
  (unless (null? stack)
    (first stack)))

;; Run a program defined as a stack.
(define (run prog)
  ;; Internal stack.
  (define stack null)
  ;; for each element in program...
  (for ([elem prog])
    ;; if current element is a procedure..
    (if (procedure? elem)
        ;; local bindings for the top two elements..
        (let ([num1 (peek stack)]
              [num2 (peek (pop stack))])
          ;; set! the internal stack to be the current stack with top two
          ;; elements popped off and the result of applying current elem to them
          ;; pushed to stack.
          (set! stack (push (elem num1 num2) (pop (pop stack)))))
        ;; Otherwise, set! the internal stack to the result of pushing the
        ;; current element to the stack.
        (set! stack (push elem stack))))
  stack)

```

Credit

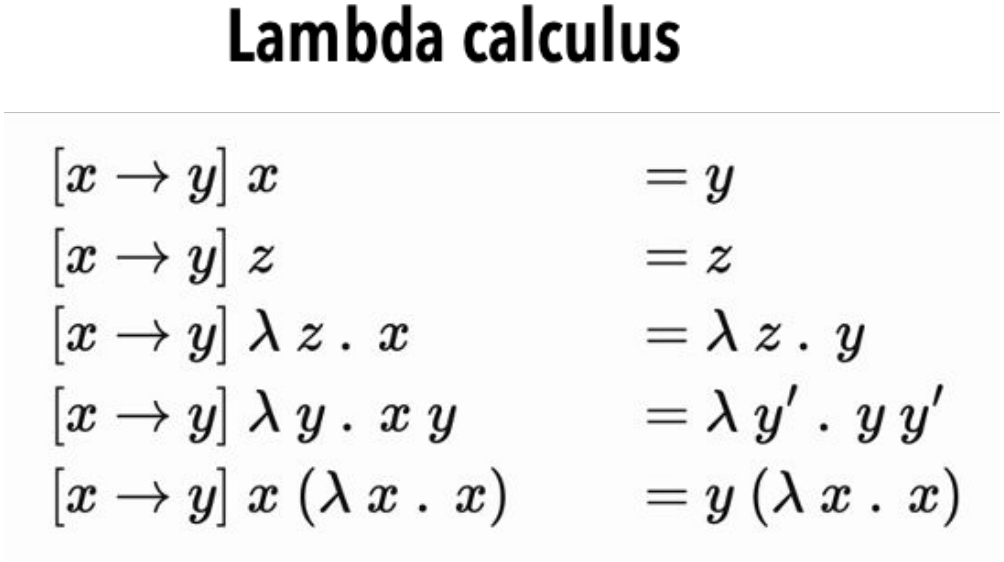
Turing languages

Inspired by compurers
Computation as operations on a machine



Church languages

Inspired by math!
Computation as mathematical functions



Turing languages

- **Focus on practical connection to computers**
  - Make hardware first, then design software abstractions around that
  - How does data get stored?
  - What is the order of execution (control of flow)?

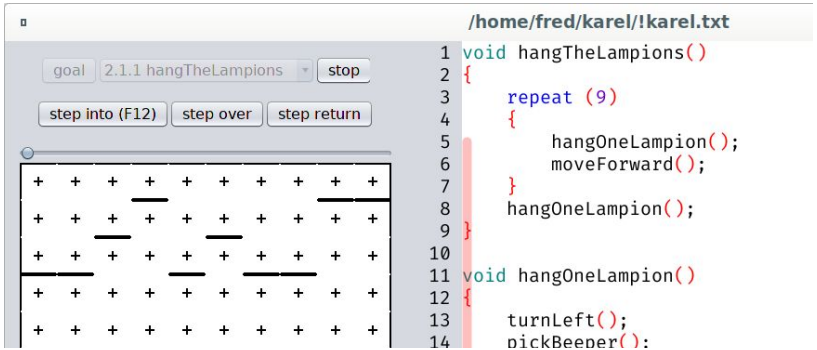
Church languages

- **Focus on mathematics**
  - Think through abstractions first. What is the “mathematical essence of a function” first?
  - What is a function?
  - How can we formally define functions?
  - How can we formally define variables?

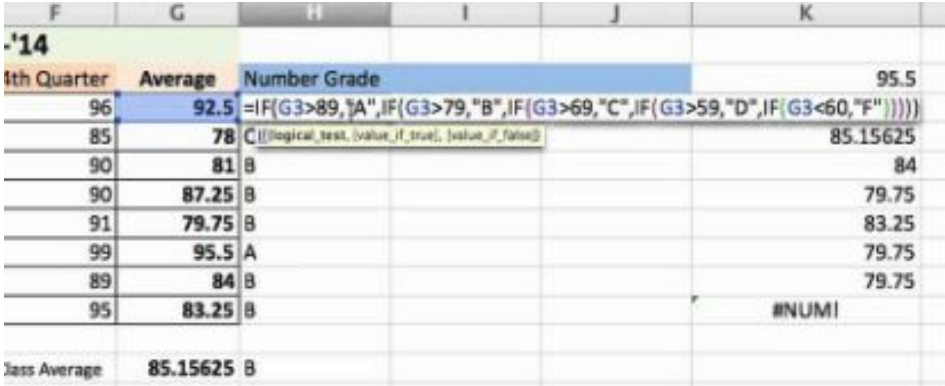
Key idea

Programming paradigm: a way we cluster languages by a combination of programming style and language behavior

Imperative paradigm



Functional paradigm



Object-oriented paradigm

```
#pragma once
#include "vector.h"
class RandomBag {
public:
  void add(int value);
  int removeRandom();


private:
  Vector<int> elems;
};
```

How do we compare across languages?

What should we compare across languages?

Are these two types of languages equally powerful?

Key idea

Church Turing Thesis: Anything that you could compute with a Turing Machine you could compute with lambda calculus.

Key idea

Turing complete: equivalent computational power to a Turing machine...aka any algorithm could be implemented

Used as a benchmark for what kind of stuff you can do with a programming language

Key idea

Turing complete: equivalent computational power to a Turing machine...aka any algorithm could be implemented

Used as a benchmark for what kind of stuff you can do with a programming language

C++

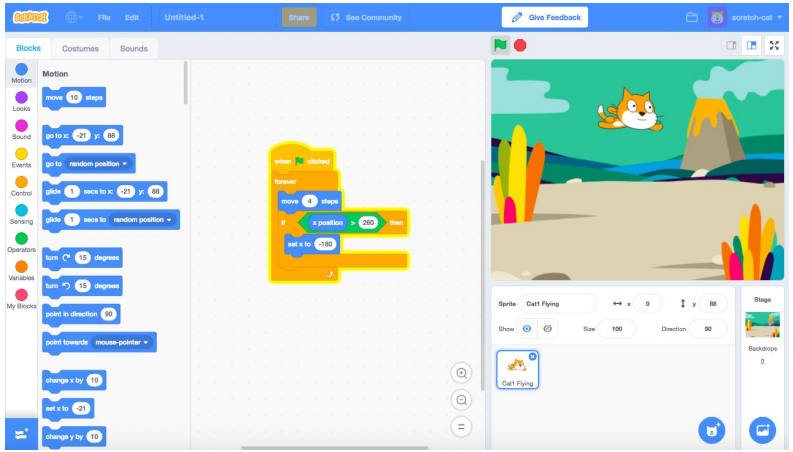
Python

C

Java

Which of these are NOT Turing complete?

Minecraft

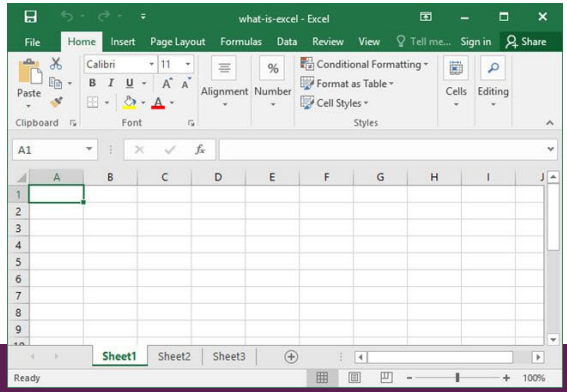


Scratch

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title goes here</title>
  </head>
  <body>

  </body>
</html>
```

Excel



Which of these are NOT Turing complete?

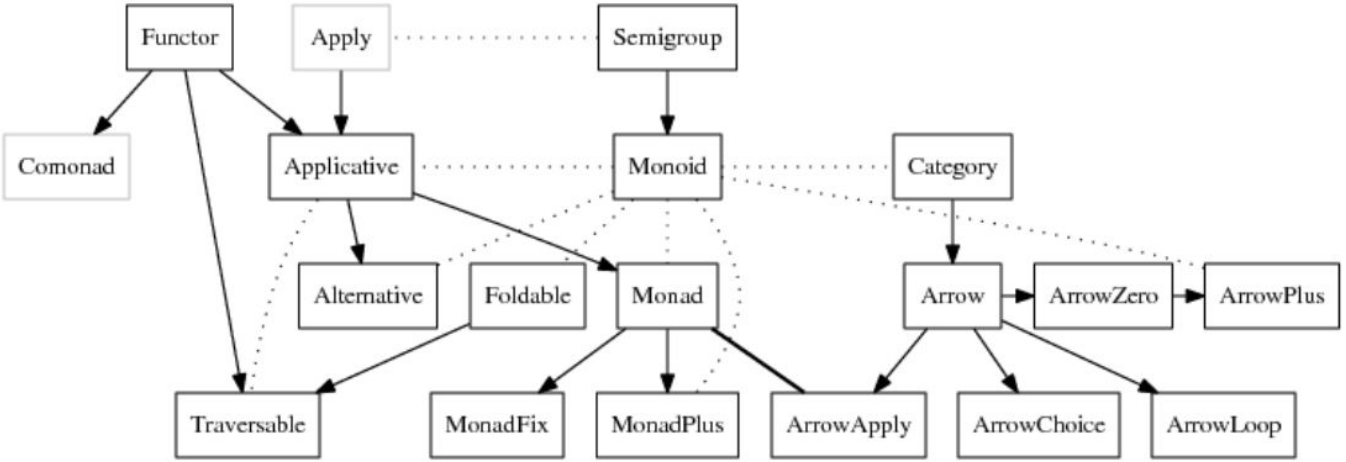
HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title goes here</title>
  </head>
  <body>

  </body>
</html>
```

- Describes data, not computation
- Doesn't allow you to execute for-loops, etc.
  - Not every algorithm can be implemented using HTML

Are these two types of languages equally easy to use?

Church languages are intimidating




```
Couldn't match type `k0' with `b'
  because type variable `b' would escape its scope
This (rigid, skolem) type variable is bound by
  the type signature for
    groupBy :: Ord b => (a -> b) -> Set a -> Set (b, [a])
The following variables have types that mention k0
```

```
public class Person {
     private final String firstName;
     private final String lastName;
     private final Integer age;
     public Person(String firstName,
                   String lastName,
                   Integer age) {
          this.firstName = firstName;
          this.lastName = lastName;
          this.age = age;
     }
     public String getFirst() {
          return firstName;
     }
     public String getLast() {
          return lastName;
     }
     public Integer getAge() {
        return age;
     }
     public Boolean valid() {
        return age > 18;
     }
}
```

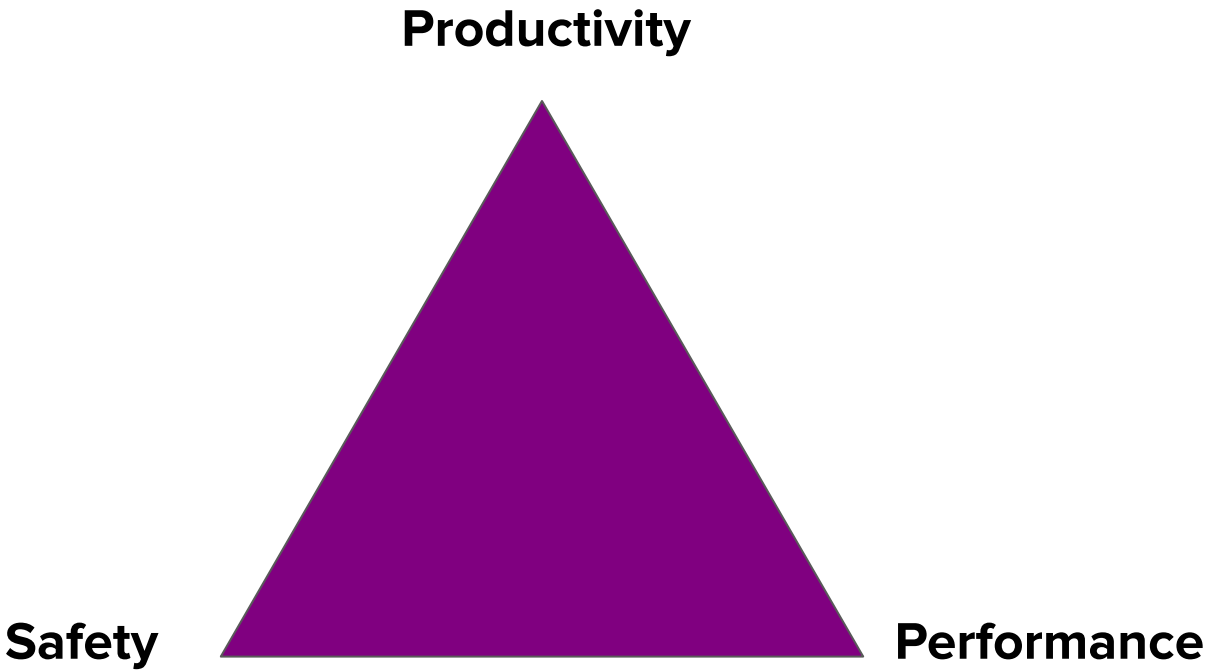
```
public static List<String> validByAge(List<Person> in) {
   List<Person> people = new ArrayList<Person>();
   for (Person p: in) {
     if (p.valid()) people.add(p);
   }

   Collections.sort(people, new Comparator<Person>() {
      public int compare(Person a, Person b) {
        return a.age() - b.age();
      }
   } );

   List<String> ret = new ArrayList<String>();
   for (Person p: people) {
     ret.add(p.first);
   }
   return ret;
}

List<Person> input = new ArrayList<Person>();
input.add(new Person("John", "Valid", 32));
input.add(new Person("John", "InValid", 17));
input.add(new Person("OtherJohn", "Valid", 19));

List<Person> output = validByAge(input)
```

```
case class Person(val first: String, val last: String, val age: Int) {
  def valid: Boolean = age > 18
}

def validByAge(in: List[Person]) =
  in.filter(_.valid).sortBy(_.age).map(_.first)

validByAge(List(
  Person("John", "Valid", 32),
  Person("John", "Invalid", 17),
  Person("OtherJohn", "Valid", 19)))
```

David Pollak, "Beginning Scala"

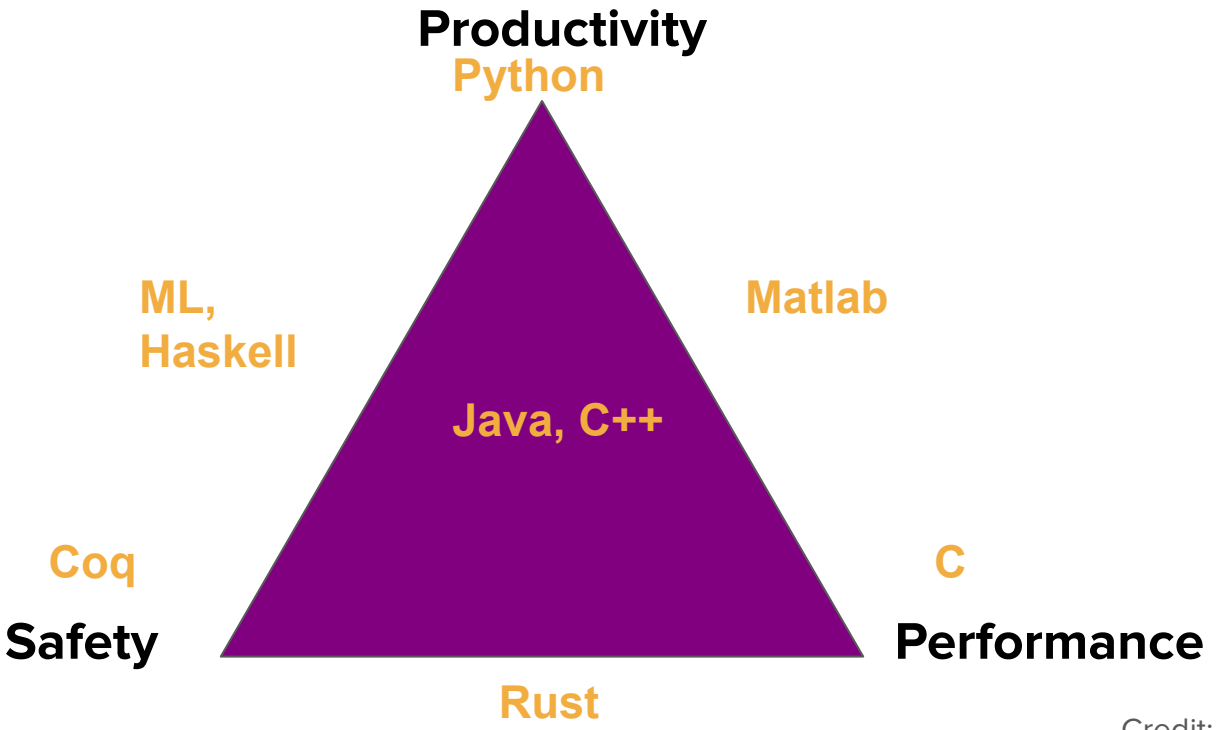
Other ways to compare programming languages

Tradeoffs



Credit: Alex Aiken

Tradeoffs



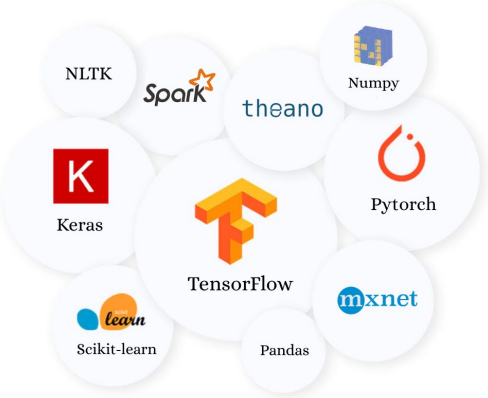
Credit: Alex Aiken

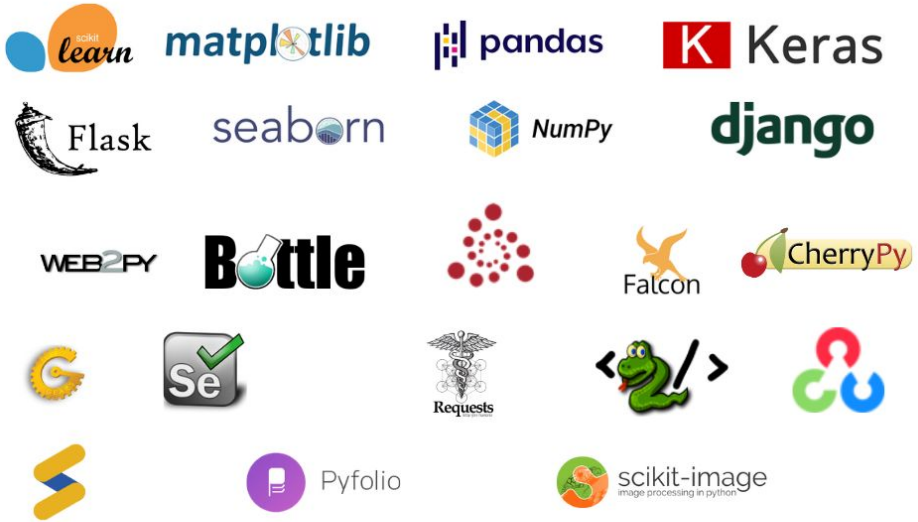
Design consideration: Productivity

Productivity:

- **Do we have big, easy-to-use building blocks (aka libraries) to get to powerful programs?**
- Is this programming language easy-to-use and read?

Productivity: libraries to build on top of?





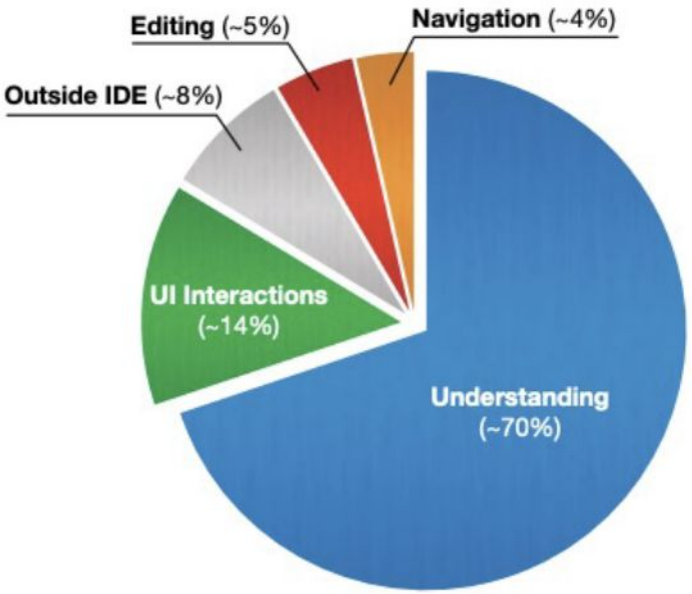
python takes the cake!

Productivity:

- Do we have big, easy-to-use building blocks (aka libraries) to get to powerful programs?
- **Is this programming language easy-to-use and read?**

What percentage of time do programmers spend actually writing code when they’re programming?

ONLY 5% 



Minelli et al. “I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time” ICPC ’15.

Project	Comprehension	Navigation	Editing	Others
Average	**57.62%**	**23.96%**	**5.02%**	**13.40%**

Xia et al. “Measuring Program Comprehension: A Large-Scale Field Study with Professionals.” IEEE Trans. Softw. Eng, 2018.

Productivity:

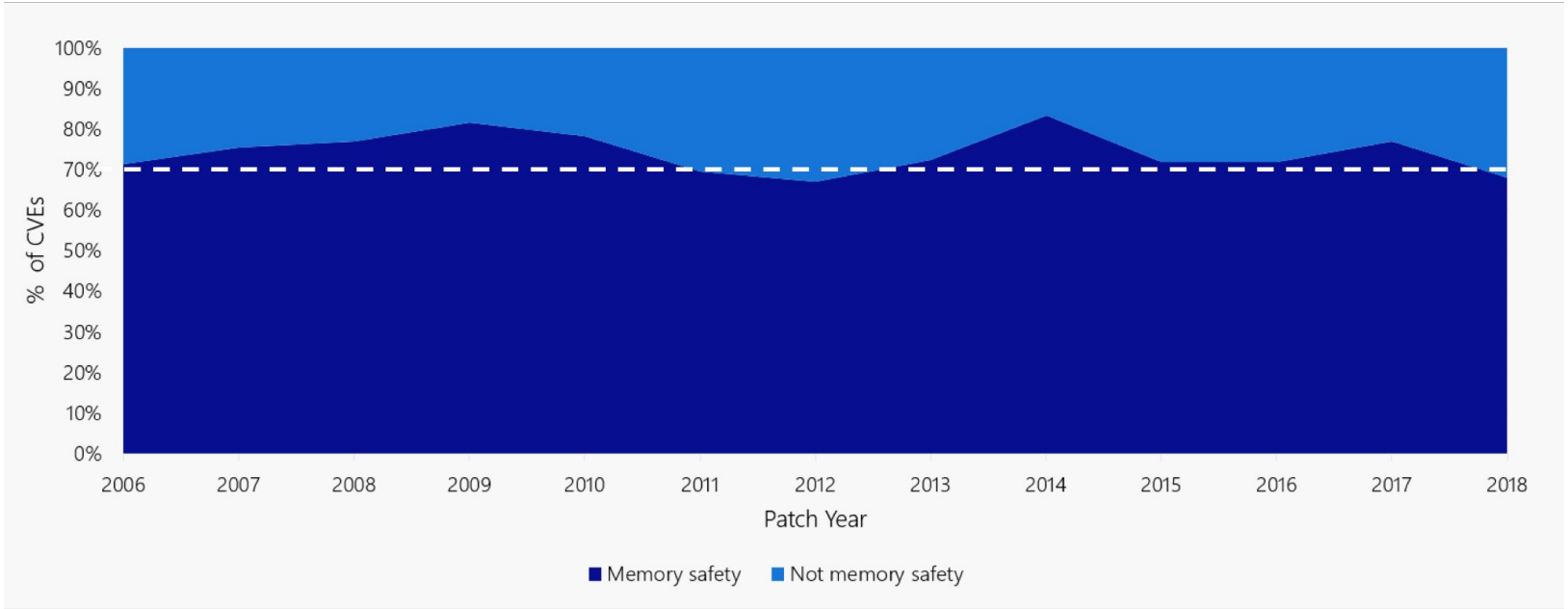
- Do we have big, easy-to-use building blocks (aka libraries) to get to powerful programs?
- Is this programming language easy-to-use and read?

Design consideration: Safety

Safety: reduce vulnerabilities, exposures, errors

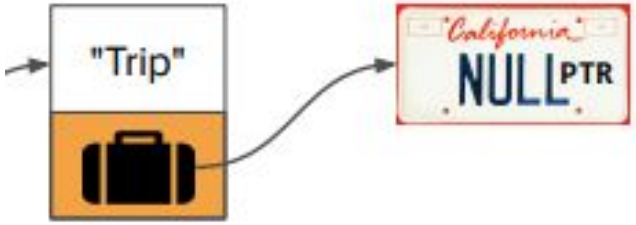
What percentage of Common Vulnerabilities and Exposures in C++ are caused by memory bugs?

"The majority of vulnerabilities fixed and with a CVE assigned are caused by developers inadvertently inserting memory corruption bugs into their C and C++ code."

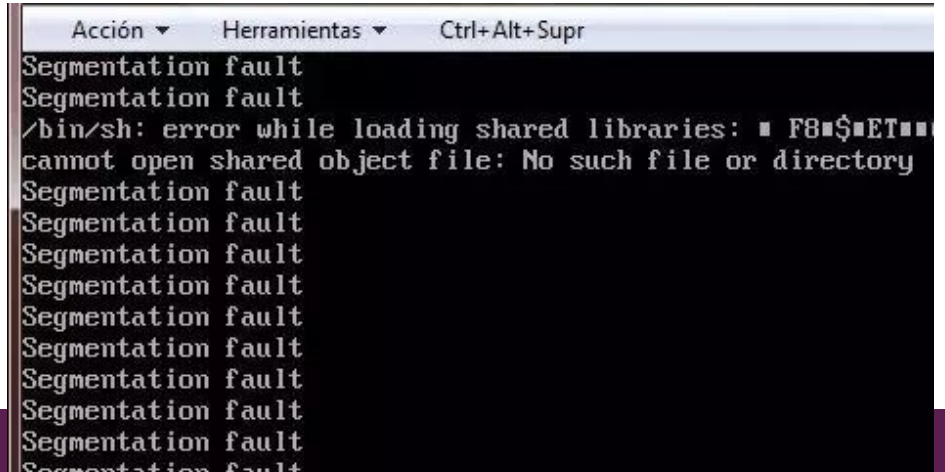


Microsoft Security Response Center. "A proactive approach to more secure code." 2019

Womp womp :(



“Don’t dereference a null pointer”

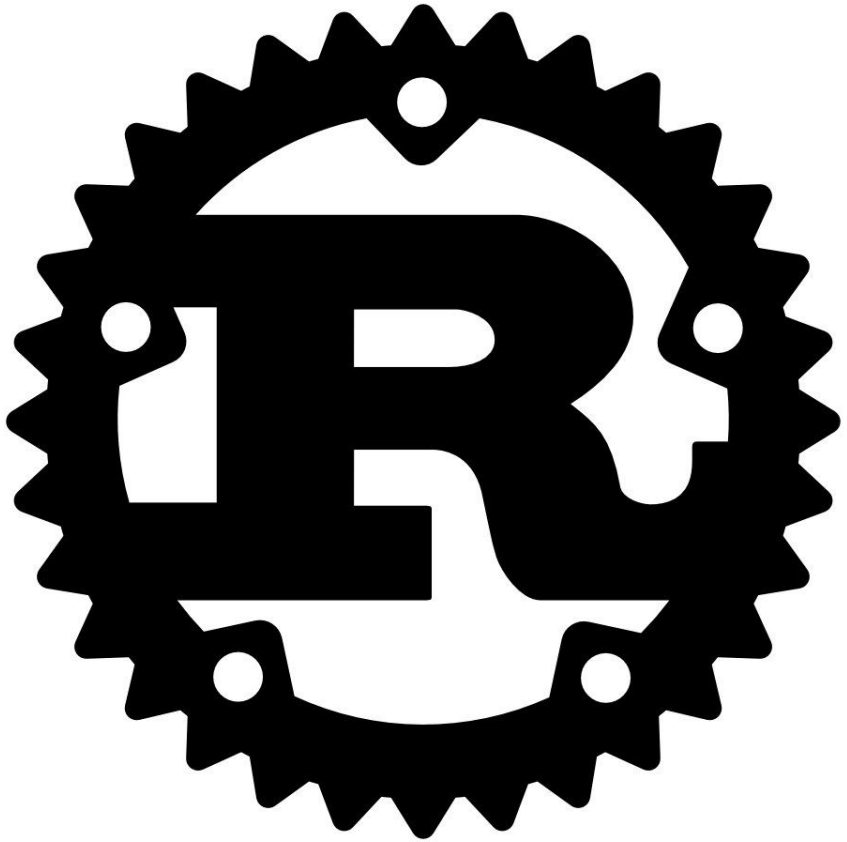


Womp womp

Buffer
(8 bytes

P	A	S	S	
0	1	2	3	

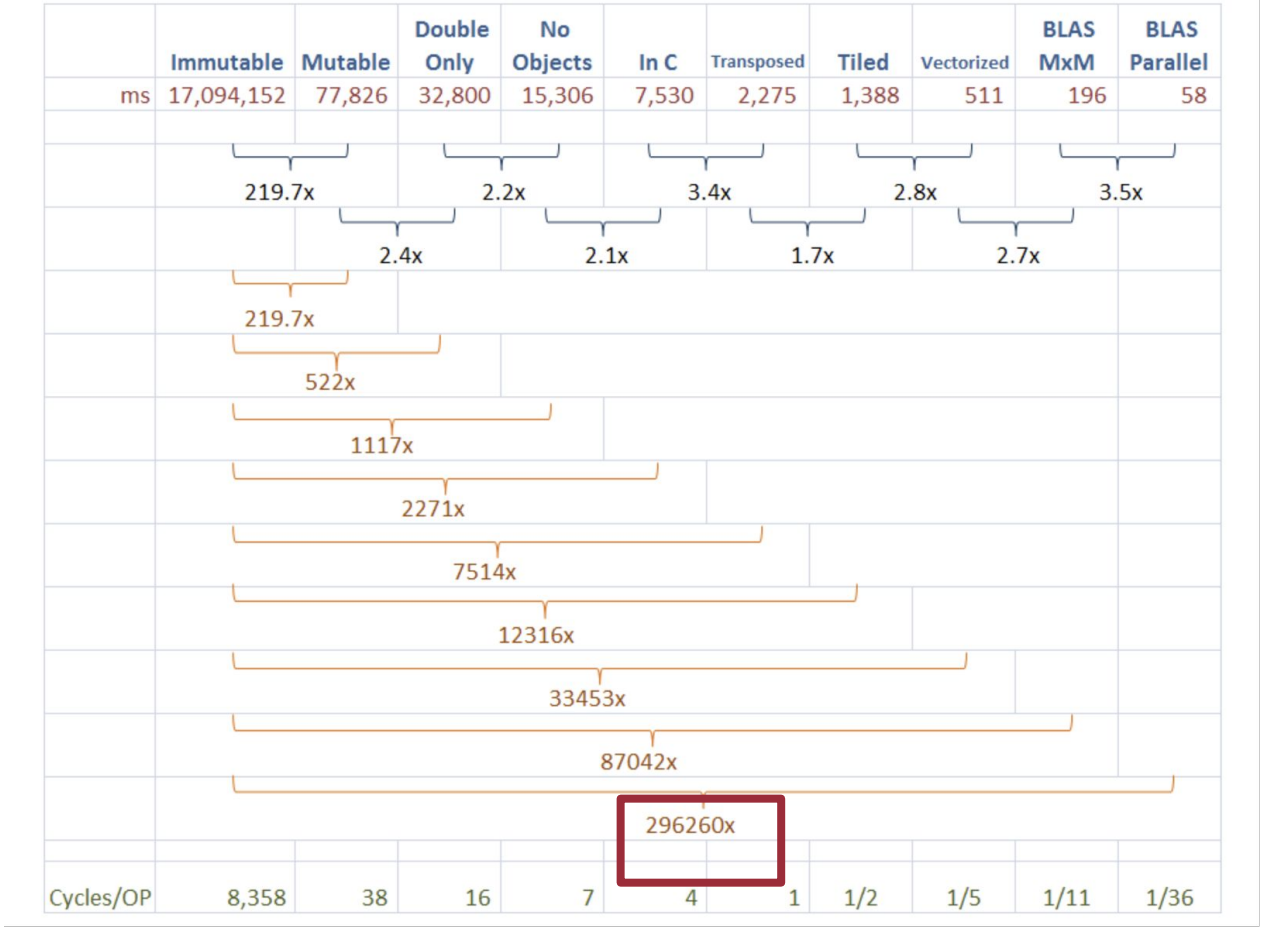
reference a null pointer”



The Rust Programming Language

Design consideration: Performance

How much faster is C compared to Java?



Saman Amarasinghe, MIT 6.172 "Performance Engineering". 2009

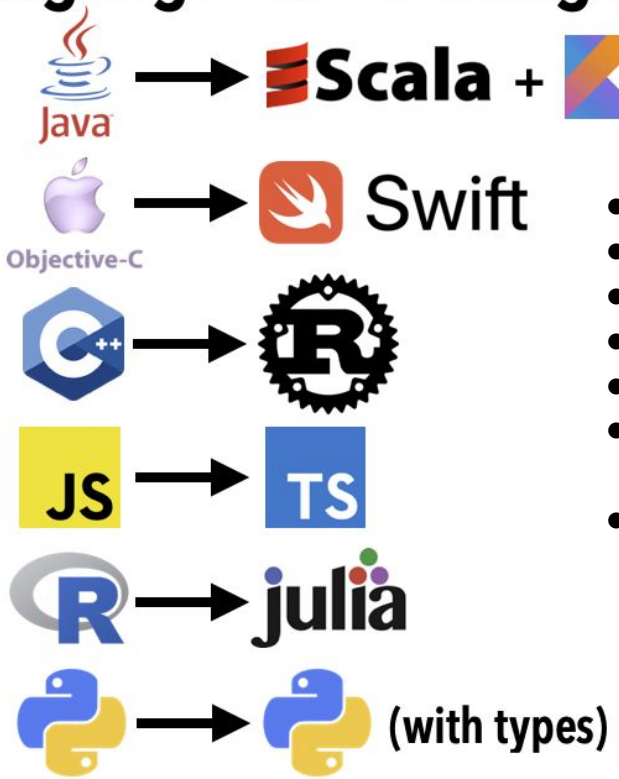
The Future of PLs

Designing cutting-edge programming languages is like designing high fashion

- Very few people will actually use the newest PLs out of PL research but the PL research is crucial to finding the boundaries / limits / new frontiers of programming as we know it
- There are trickle-down effects

Credit: Jean Yang

Church languages are coming!



- Types!
- Memory safety!
- Better error handling!
- Better compilers!
- Better abstract patterns!
- Generic/modular programming!
- Concurrency / parallelization!

Church languages are coming!

Messenger.com Now 50% Converted to Reason

September 8, 2017

Boom!

- Messenger used to receive bugs reports on a daily basis; since the introduction of Reason, there have been a total of **10 bugs** (that's during the whole year, not per week)! *
- Most of the messenger core team's new features are now developed in Reason.
- Dozens of massive refactors while iterating on ReasonReact. Refactoring speed went from days to hours to dozens of minutes. I don't think we've caused more than a few bugs during the process (counted toward the total number of bugs).

Things we didn’t cover in PL design

- Idea of programming paradigms
- Interpreted vs compiled language
- Static vs dynamic typing
- Usability / User Interactions
- Non-western traditions of computation

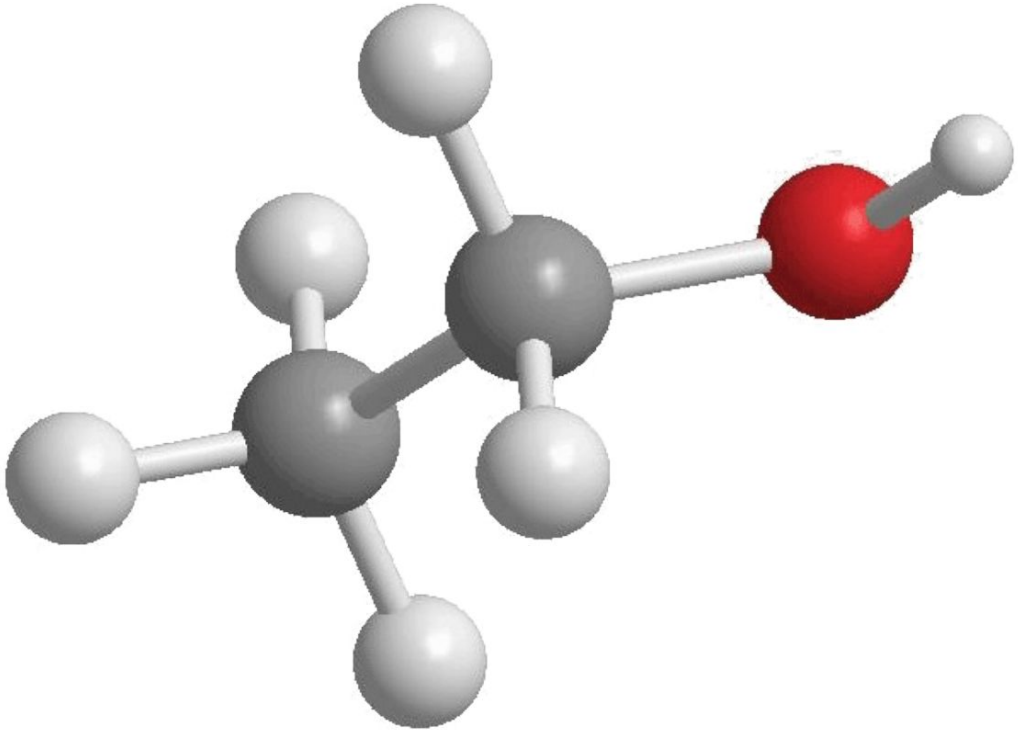
The future is out there!

How can we represent real-world systems of connected components?

Graphs

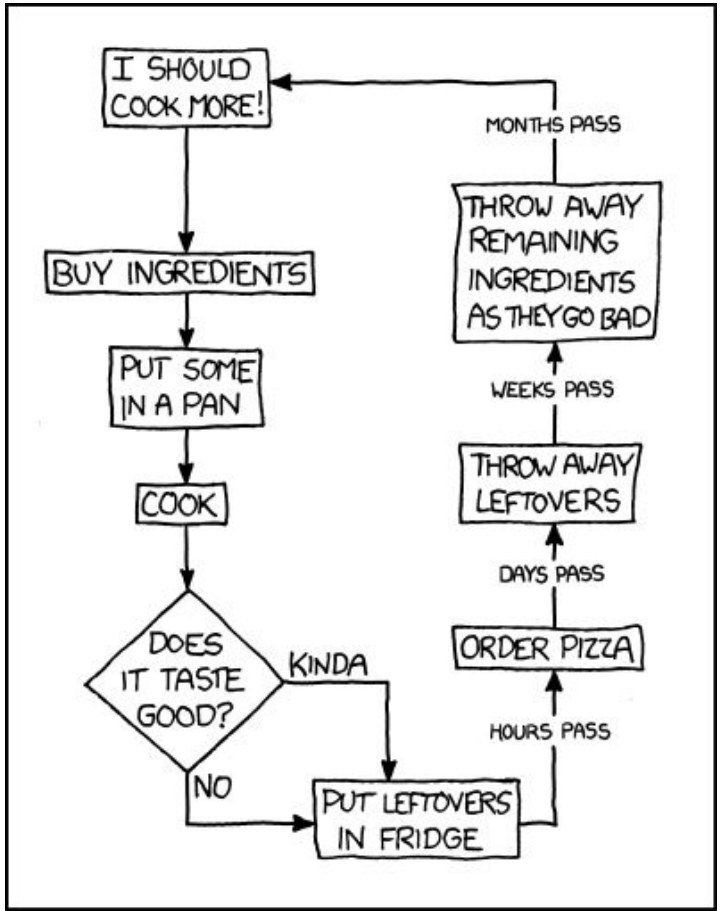
Social Networks

Chemical Bonds

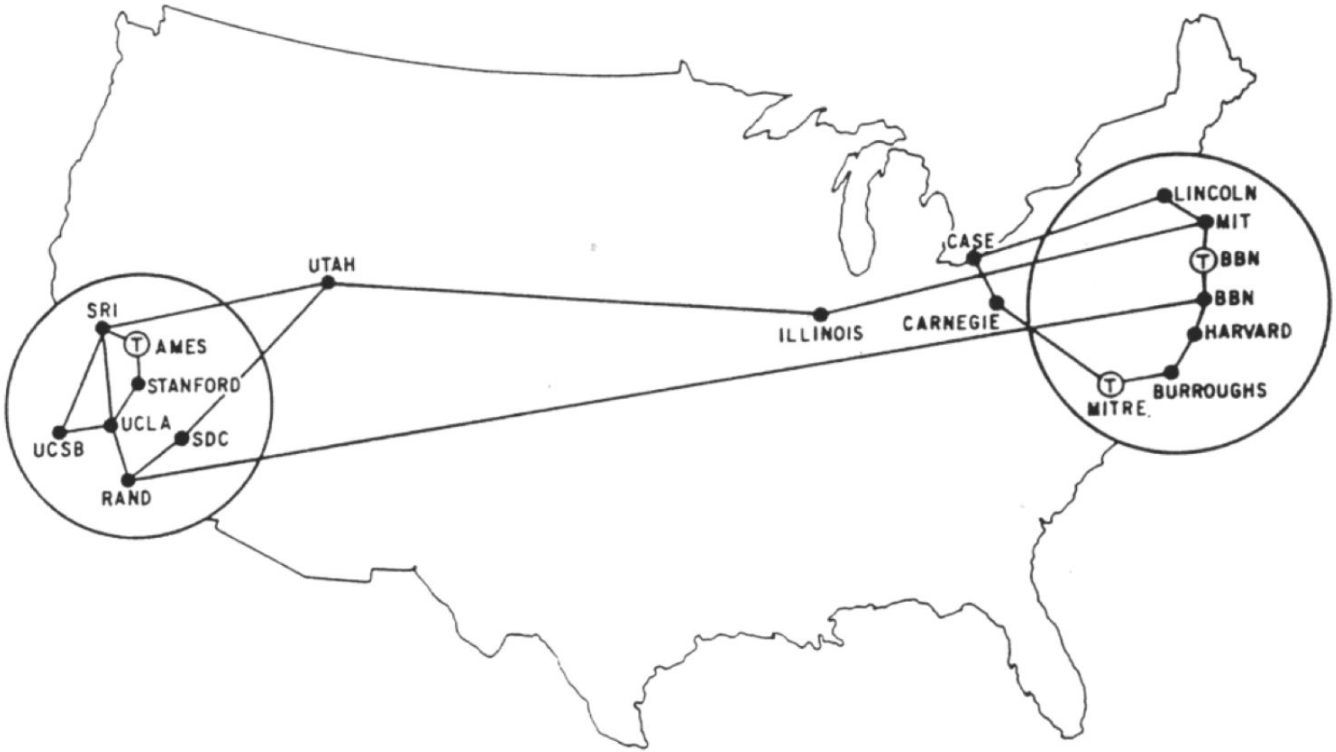


The Interstate Highway System

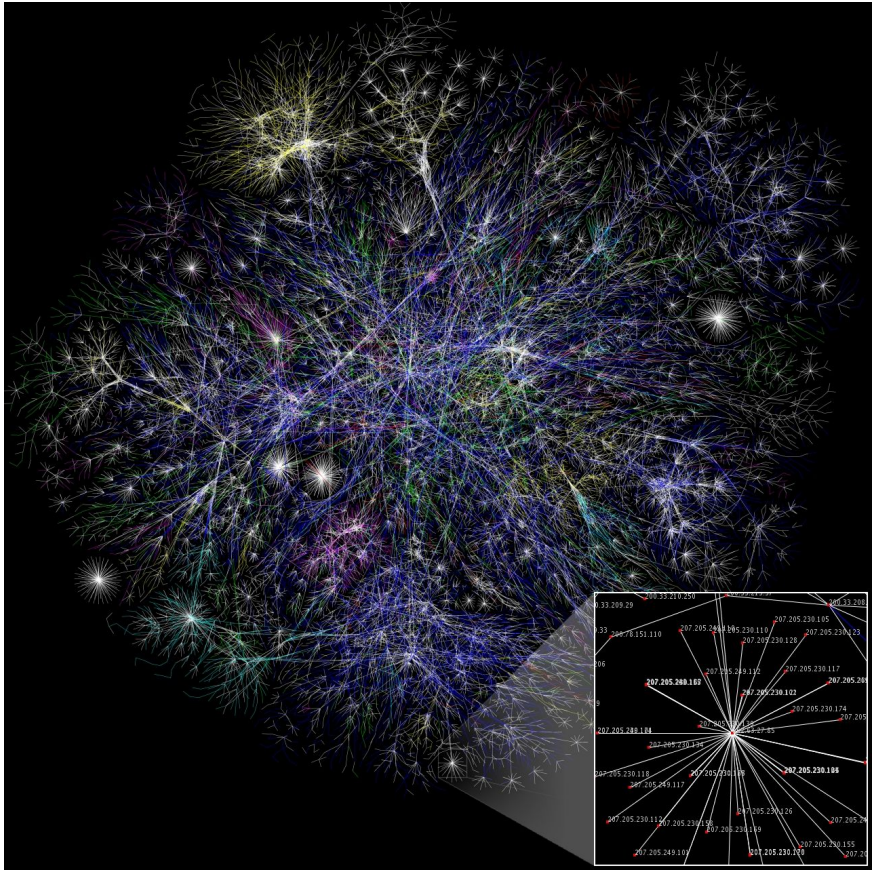
Flowcharts


I SHOULD COOK MORE!
BUY INGREDIENTS
PUT SOME IN A PAN
COOK
DOES IT TASTE GOOD?
KINDA
NO
PUT LEFTOVERS IN FRIDGE
HOURS PASS
ORDER PIZZA
DAYS PASS
THROW AWAY LEFTOVERS
WEEKS PASS
THROW AWAY REMAINING INGREDIENTS AS THEY GO BAD
MONTHS PASS

The Internet!



The Internet!



What is a graph?

Definition

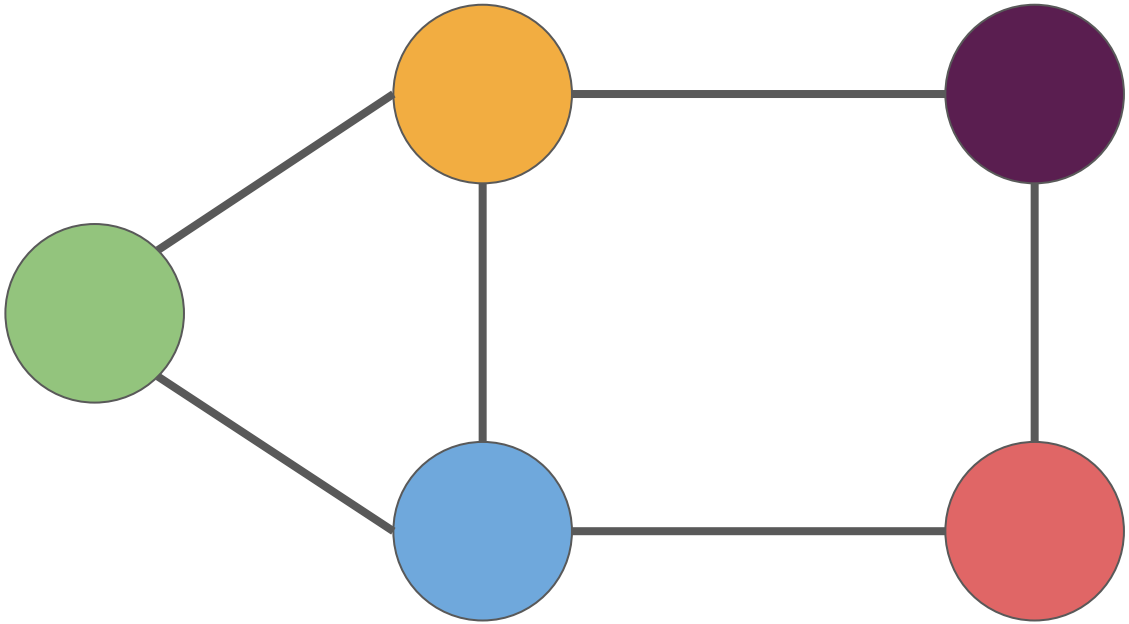
graph
A structured way to represent relationships between different entities.

Our first graph!

- A structured way to represent relationships between different entities.

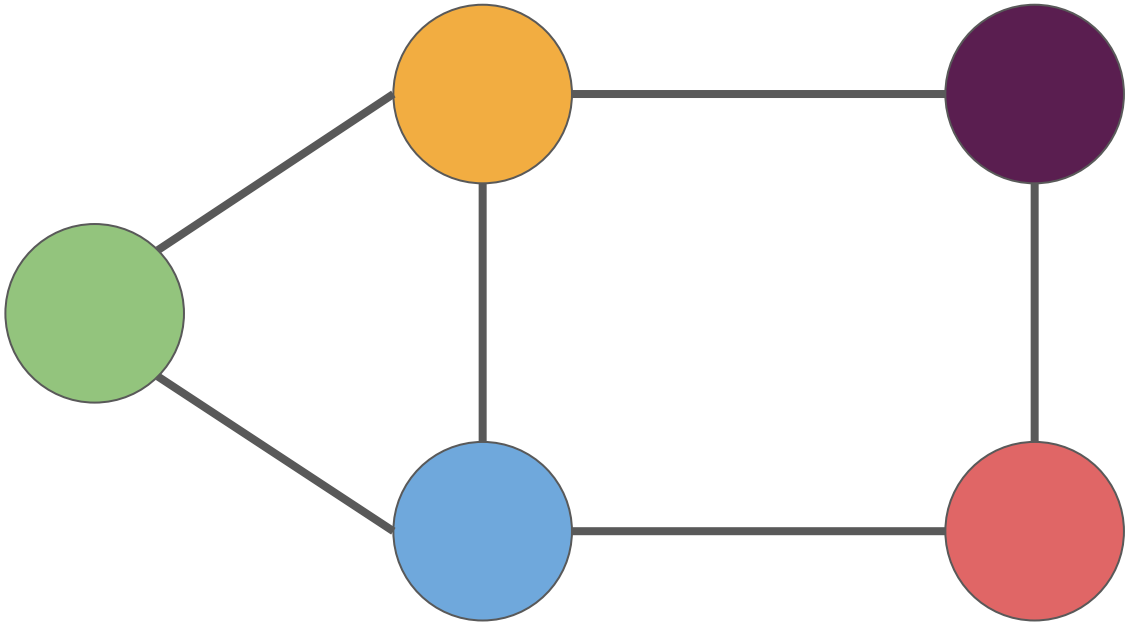
Our first graph!

- A structured way to represent relationships between different entities.



Our first graph!

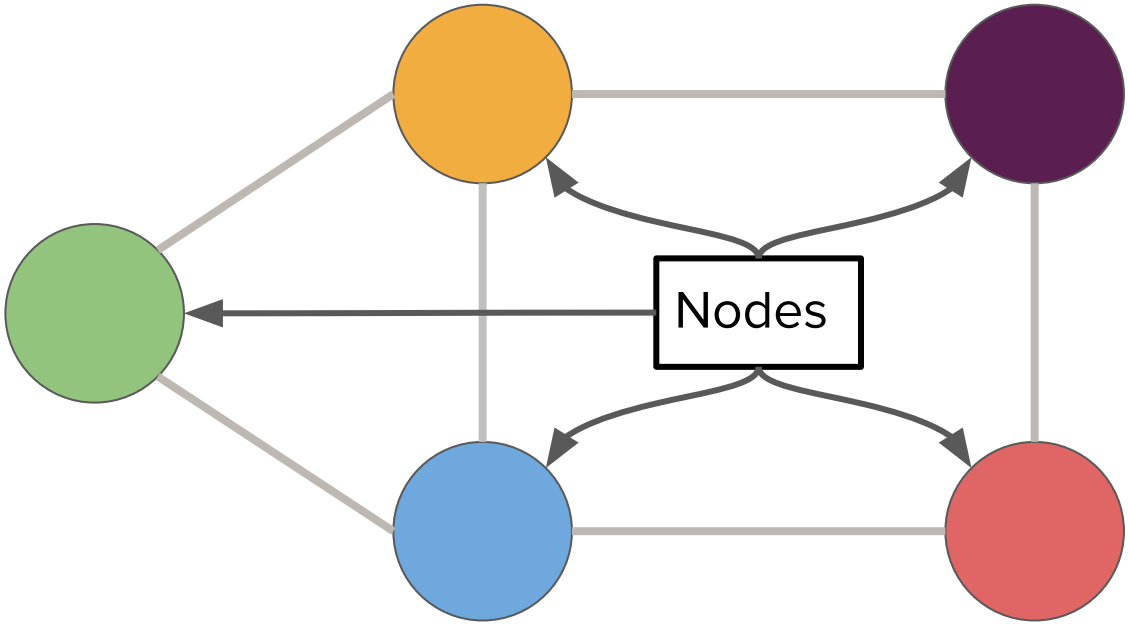
- A structured way to represent relationships between different entities.



A graph consists of a set of **nodes** connected by **edges**.

Our first graph!

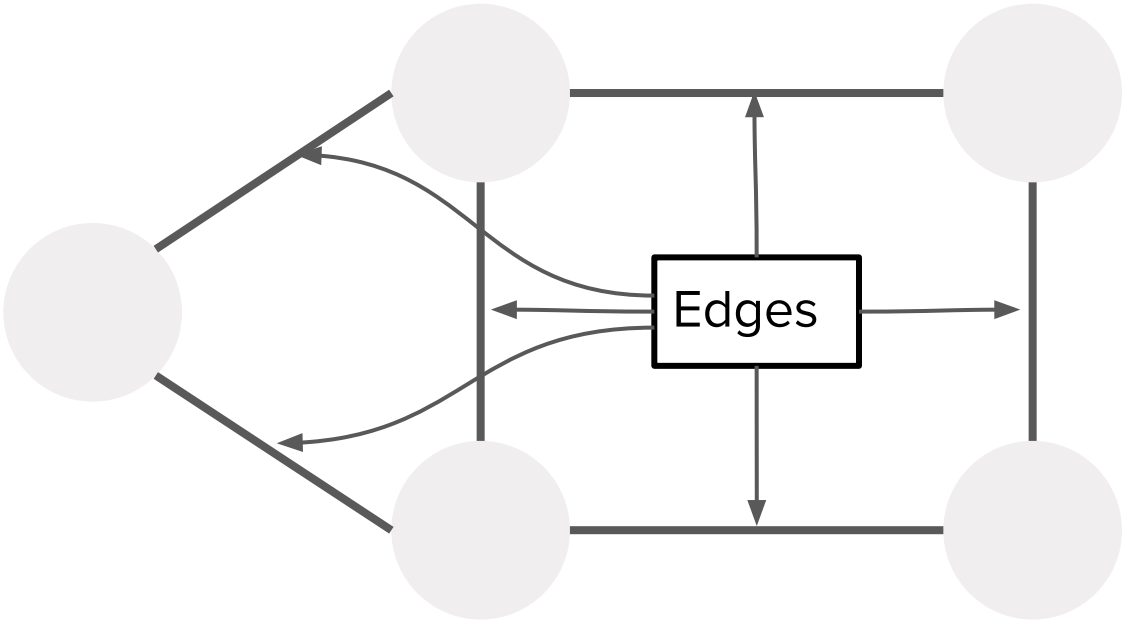
- A structured way to represent relationships between different **entities**.



A graph consists of a set of **nodes** connected by edges.

Our first graph!

- A structured way to represent **relationships** between different entities.



A graph consists of a set of nodes connected by **edges**.

Types of graphs

Different types of graphs

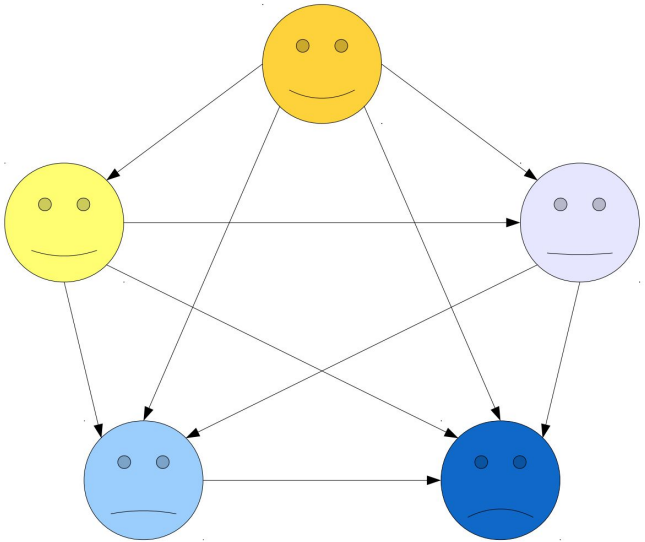
- Some graphs are **directed**. These represent situations where relationships are unidirectional (an action/verb that explicitly implies only one direction).

Different types of graphs

- Some graphs are **directed**. These represent situations where relationships are unidirectional (an action/verb that explicitly implies only one direction).
    - Ex: I follow Dwayne "The Rock" Johnson on Instagram, but he doesn't follow me back.

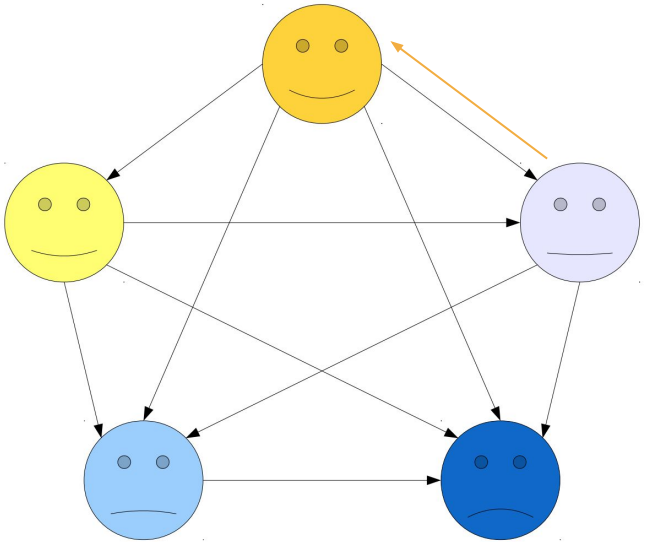
Different types of graphs

- Some graphs are **directed**. These represent situations where relationships are unidirectional (an action/verb that explicitly implies only one direction).
  - Ex: I follow Dwayne "The Rock" Johnson on Instagram, but he doesn't follow me back.



Different types of graphs

- Some graphs are **directed**. These represent situations where relationships are unidirectional (an action/verb that explicitly implies only one direction).
  - Ex: I follow Dwayne "The Rock" Johnson on Instagram, but he doesn't follow me back.



Note: It is possible for a relationship in a directed graph to go both ways between two nodes, but it would need to be explicitly stated.

Different types of graphs

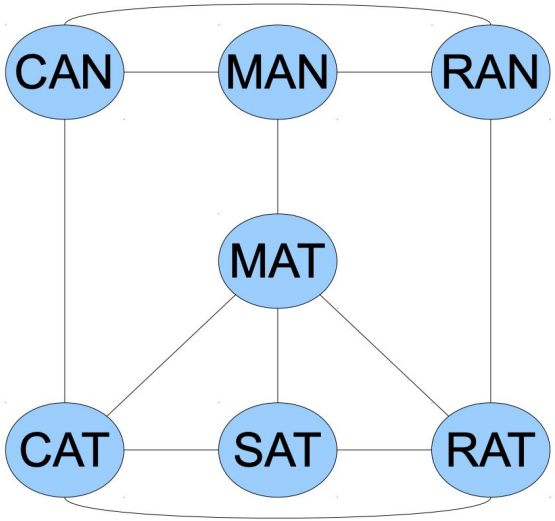
- Some graphs are **undirected**. These represent situations where relationships are bidirectional (the action/verb inherently applies to both entities).

Different types of graphs

- Some graphs are **undirected**. These represent situations where relationships are bidirectional (the action/verb inherently applies to both entities).
  - Ex: I am related to my brother, and he is related to me. The relationship applies to both of us.

Different types of graphs

- Some graphs are **undirected**. These represent situations where relationships are bidirectional (the action/verb inherently applies to both entities).
  - Ex: I am related to my brother, and he is related to me. The relationship applies to both of us.



Different types of graphs

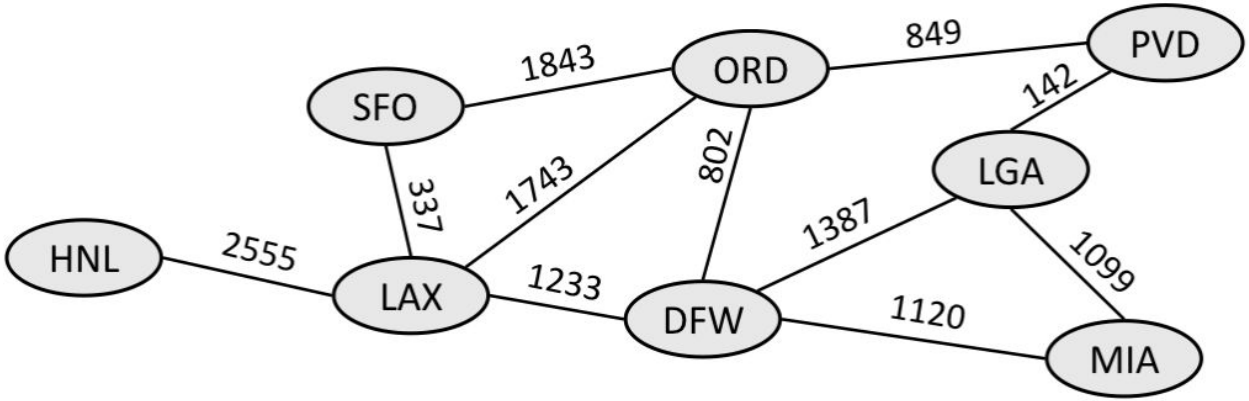
- Some graphs are **weighted**. These represent situations where not all relationships between entities are equal.

Different types of graphs

- Some graphs are **weighted**. These represent situations where not all relationships between entities are equal.
  - Ex: The different bonds between atoms in a single molecule all have different bond energies and strengths.

Different types of graphs

- Some graphs are **weighted**. These represent situations where not all relationships between entities are equal.
  - Ex: The different bonds between atoms in a single molecule all have different bond energies and strengths.

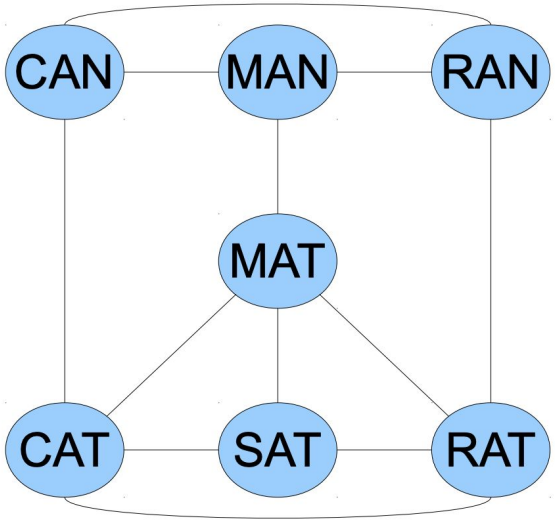


Different types of graphs

- Some graphs are **unweighted**. These represent situations where all relationships between entities have equal importance.

Different types of graphs

- Some graphs are **unweighted**. These represent situations where all relationships between entities have equal importance.
  - Ex: All connected words in a word ladder are one letter apart from one another.



Types of Graphs Summary

- **Directed**: Unidirectional relationships between nodes, represented with a pointed arrow.
- **Undirected**: Bidirectional relationships between nodes, represented with an arrow-less line.
- **Weighted**: Each edge is assigned a numerical "weight" representing its relative significance/strength.
- **Unweighted**: Each edge has equal significance, no labels assigned.

Revisiting Graph Examples

Revisiting Graph Examples: Social Network

Properties

- Nodes: ???
- Edges: ???
- Undirected or Directed?
- Unweighted or Weighted?

Revisiting Graph Examples: Social Network

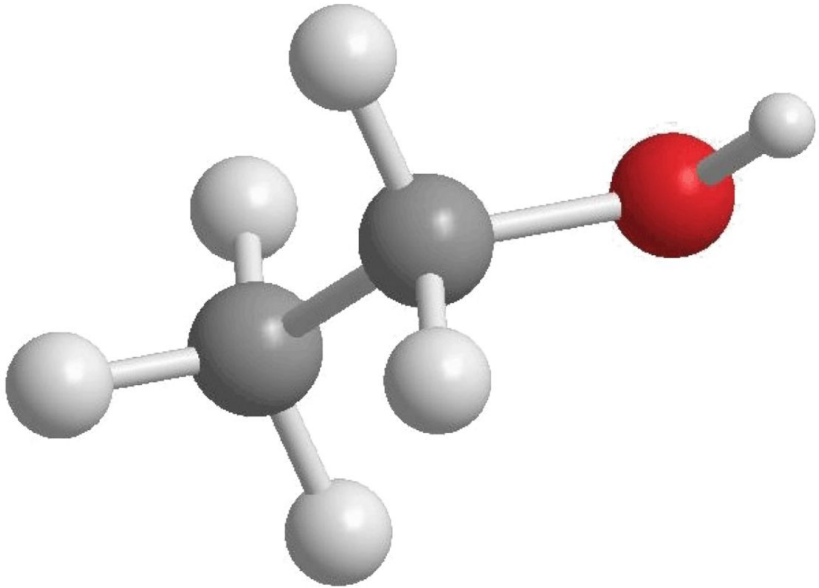
Properties

- Nodes: People
- Edges: "Friendship" or "Following"
- Undirected (Facebook) or Directed (Instagram)
- Unweighted

Revisiting Graph Examples: Chemical Bonds

Properties

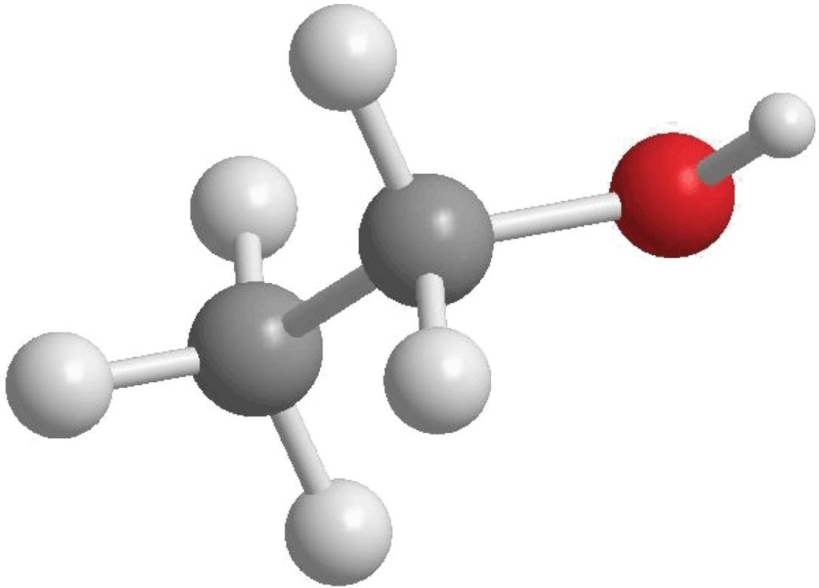
- Nodes: ???
- Edges: ???
- Undirected or Directed?
- Unweighted or Weighted?



Revisiting Graph Examples: Chemical Bonds

Properties

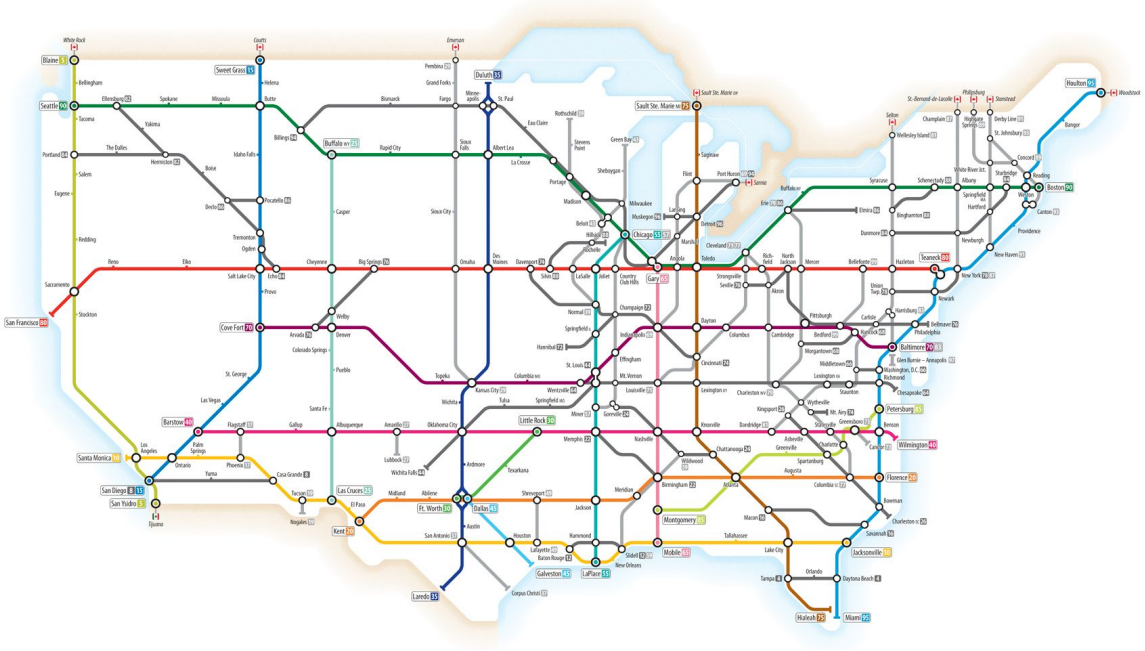
- Nodes: Atoms
- Edges: Bonds (covalent or ionic)
- Undirected
- Weighted



Revisiting Graph Examples: Interstate Highways

Properties

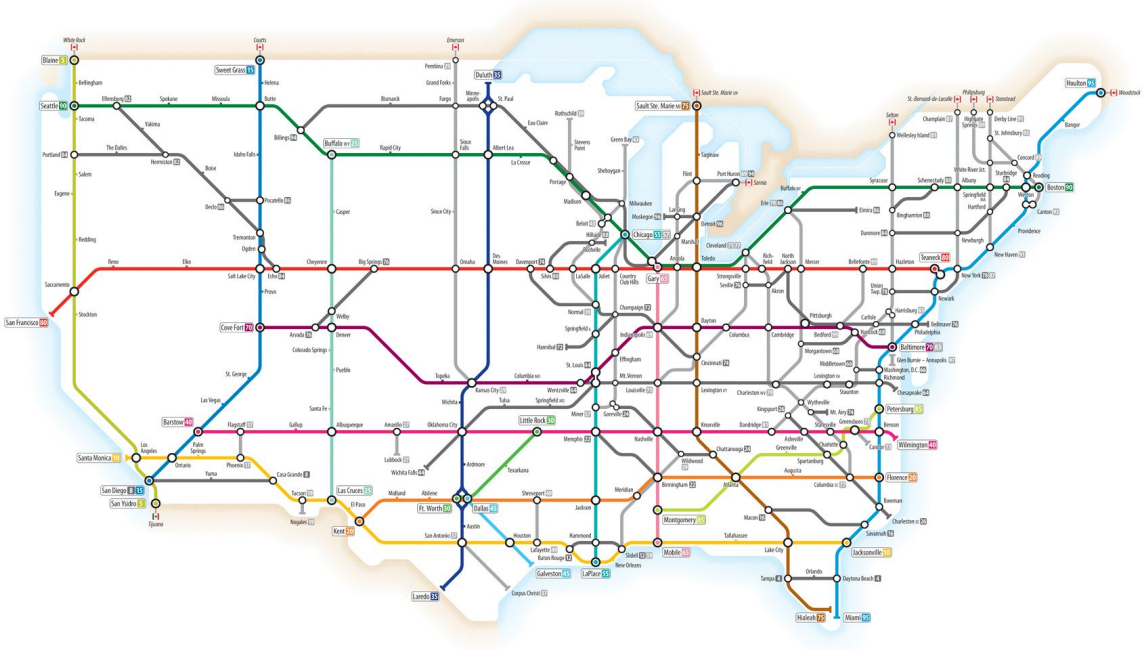
- Nodes: ???
- Edges: ???
- Undirected or Directed?
- Unweighted or Weighted?



Revisiting Graph Examples: Interstate Highways

Properties

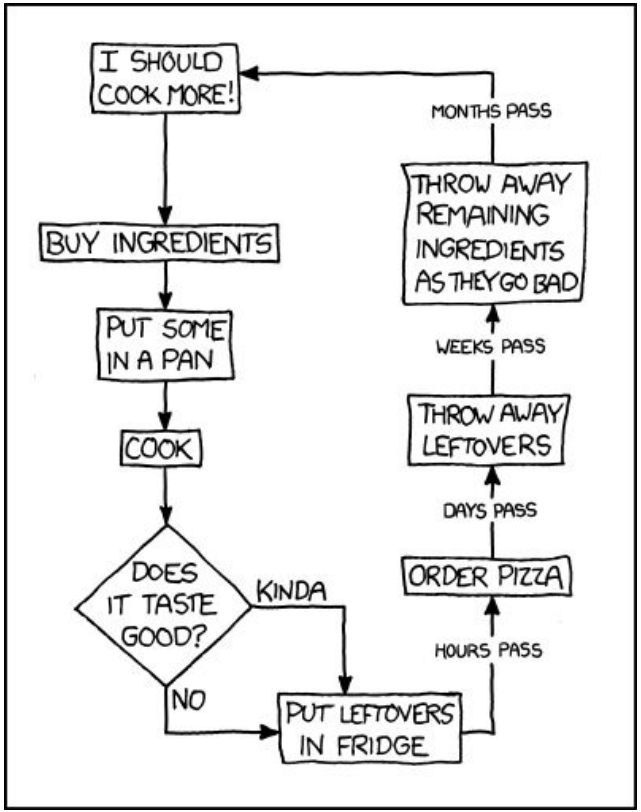
- Nodes: Cities
- Edges: Highways/roads
- Undirected
- Weighted



Revisiting Graph Examples: Flowcharts

Properties

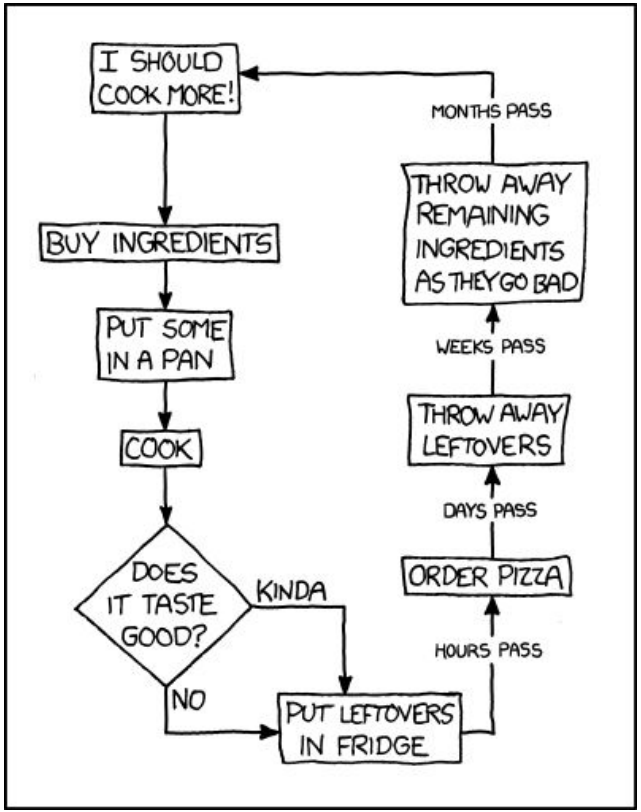
- Nodes: ???
- Edges: ???
- Undirected or Directed?
- Unweighted or Weighted?



Revisiting Graph Examples: Flowcharts

Properties

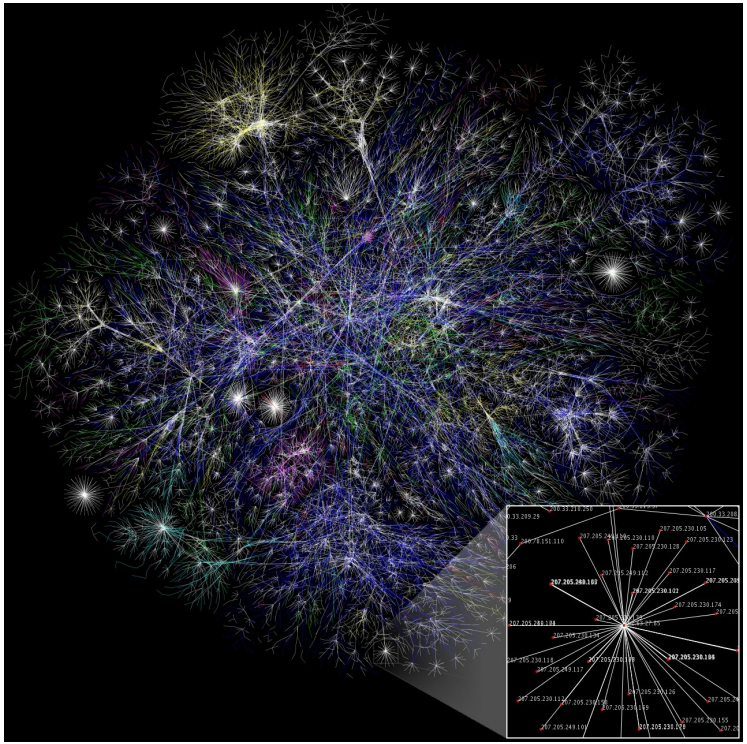
- Nodes: Events/Actions
- Edges: Transitions
- Directed
- Unweighted



Revisiting Graph Examples: The Internet

Properties

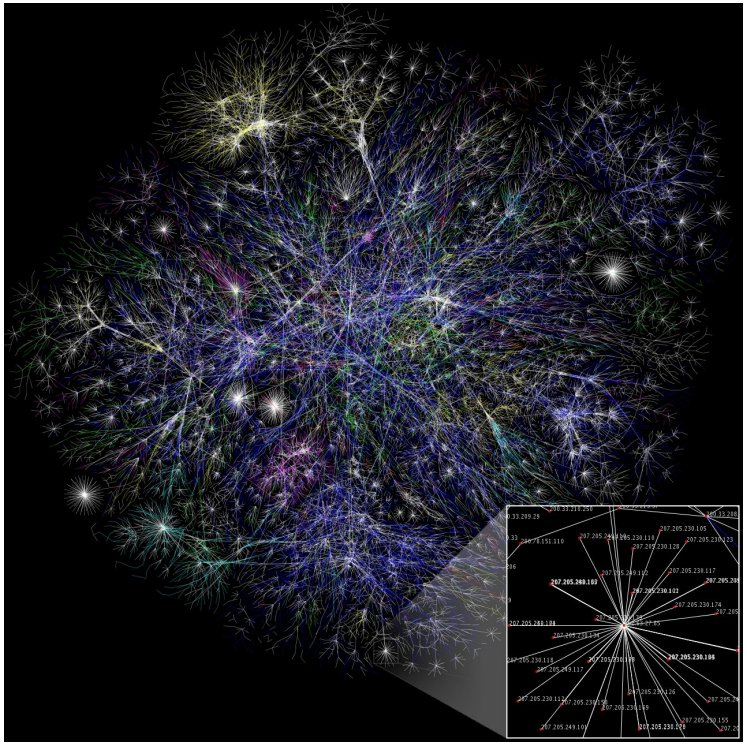
- Nodes: ???
- Edges: ???
- Undirected or Directed?
- Unweighted or Weighted?



Revisiting Graph Examples: The Internet

Properties

- Nodes: Devices (phones, computers, etc.)
- Edges: Connection pathways (Bluetooth, WiFi, Ethernet, cables)
- Undirected
- Can be weighted or unweighted



Graphs as Linked Data Structures

Putting it All Together

- We've seen nodes connected by edges (links) before when discussing linked lists and trees. These, along with graphs, are all **linked data structures**!

Putting it All Together

- We've seen nodes connected by edges (links) before when discussing linked lists and trees. These, along with graphs, are all **linked data structures**!

- What differentiates each of these linked data structures?

Putting it All Together

- We've seen nodes connected by edges (links) before when discussing linked lists and trees. These, along with graphs, are all **linked data structures**!

- What differentiates each of these linked data structures?
  - **Linked lists:** Linear structure, each node connected to at most one other node.

Putting it All Together

- We've seen nodes connected by edges (links) before when discussing linked lists and trees. These, along with graphs, are all **linked data structures**!

- What differentiates each of these linked data structures?
  - **Linked lists:** Linear structure, each node connected to at most one other node.
  - **Trees:** Nodes can connect to multiple other nodes, no cycles, parent/child relationship and a single, special root node.

Putting it All Together

- We've seen nodes connected by edges (links) before when discussing linked lists and trees. These, along with graphs, are all **linked data structures**!

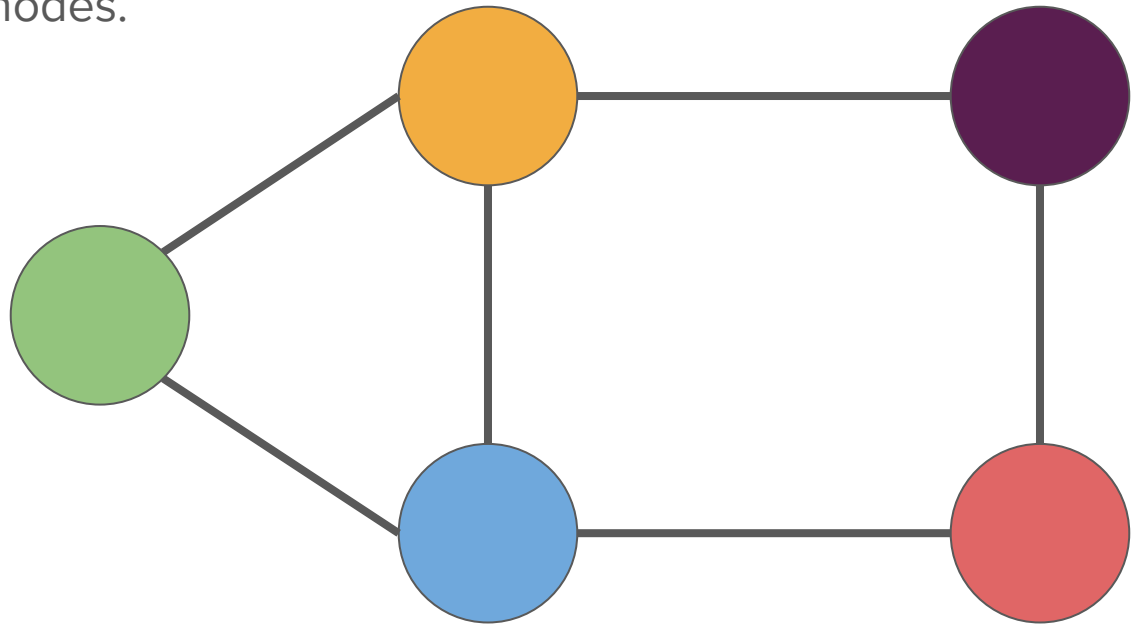
- What differentiates each of these linked data structures?
  - **Linked lists:** Linear structure, each node connected to at most one other node.
  - **Trees:** Nodes can connect to multiple other nodes, no cycles, parent/child relationship and a single, special root node.
  - **Graphs:** No restrictions. It's the wild, wild west of the node-based world!

The Wild World of Graphs

- Graphs can have cycles, and there is no notion of a parent-child relationship between nodes.

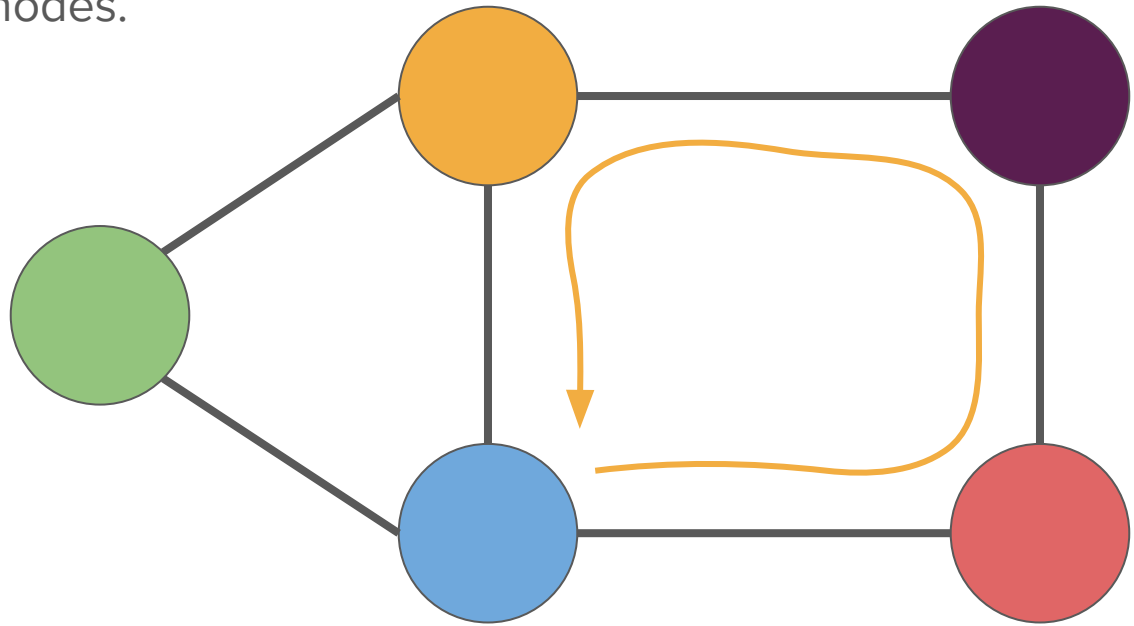
The Wild World of Graphs

- Graphs can have cycles, and there is no notion of a parent-child relationship between nodes.



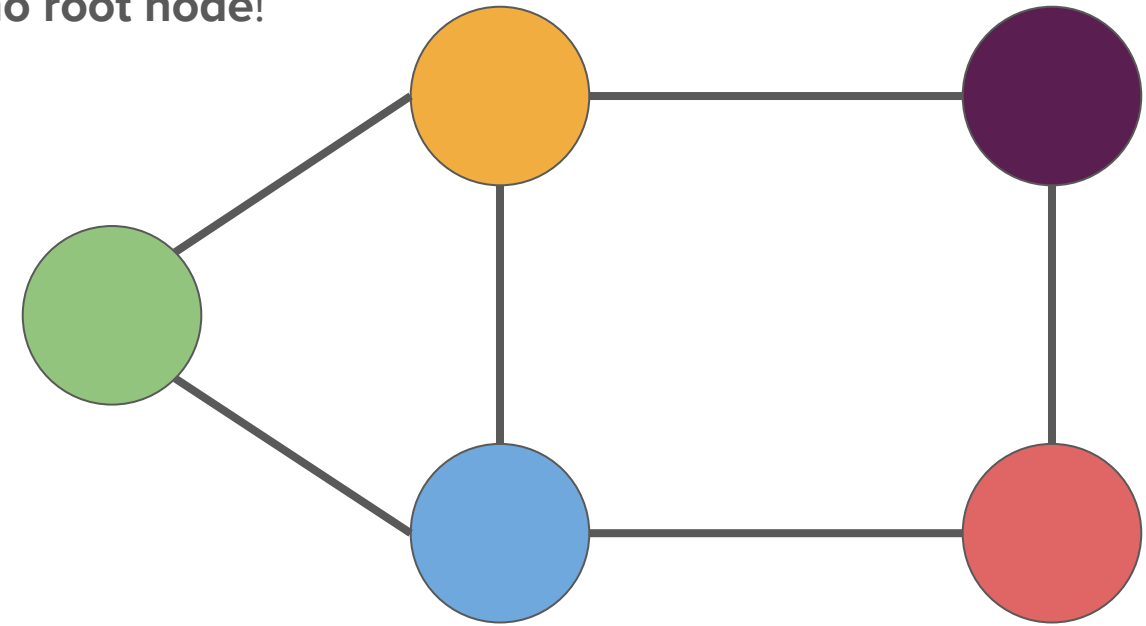
The Wild World of Graphs

- Graphs can have **cycles**, and there is no notion of a parent-child relationship between nodes.



The Wild World of Graphs

- Graphs have no nodes that are more important than other nodes. In particular, **there is no root node**!



Graphs are the most powerful, flexible, and expressive abstraction that we can use to model relationships between different distributed entities. You will find graphs everywhere you look!

Representing Graphs

How do we store and represent graphs in code?

Attendance ticket:

https://tinyurl.com/106bGraphs

Please don’t send this link to students who are not here. It’s on your honor!

The **Node** struct

```
struct Node {
    string data;
    Node* next;
}
```

The **Node** struct

```
struct Node {
    string data;
    Node* next;
}
```

How would our data be different for each application?

The **Node** struct

```
struct Node {
    string data;
    Node* next;
}
```

How can we better represent our edges in graphs?

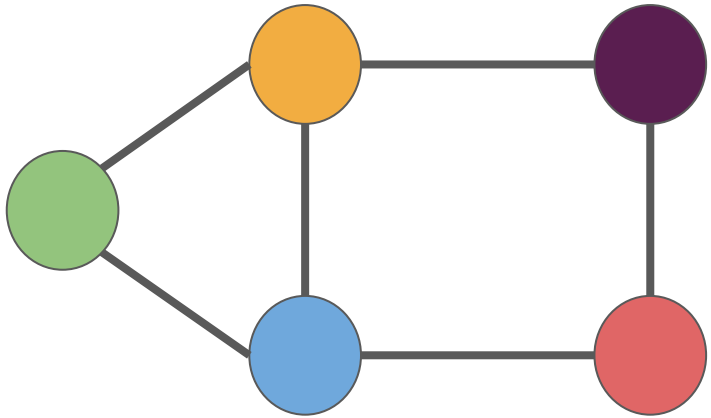
Approach 1: Adjacency List

Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.

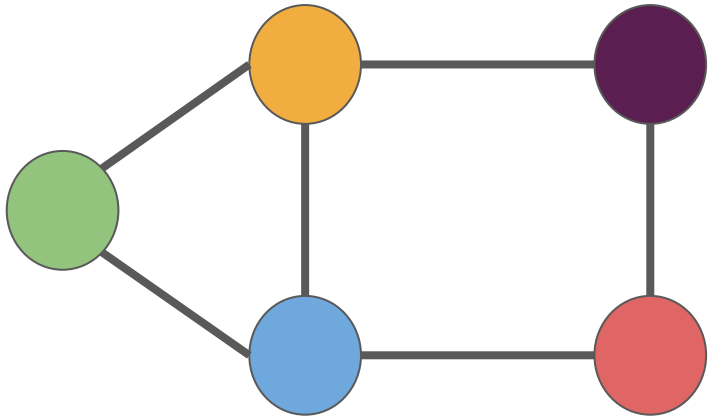
Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.



Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.

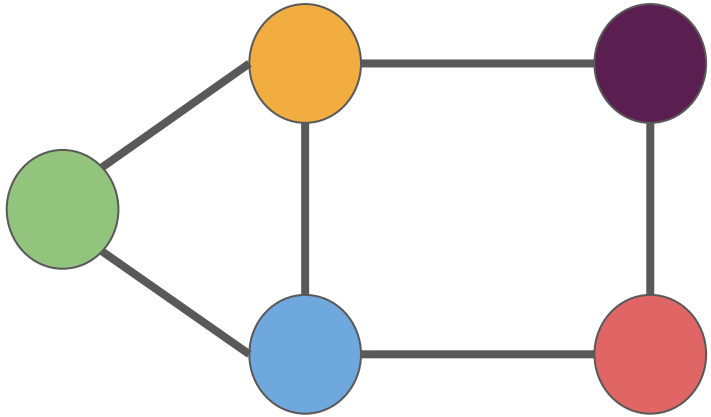


`Map<Node, Set<Node>>`

Node	Set<Node>>
Node	Adjacent to

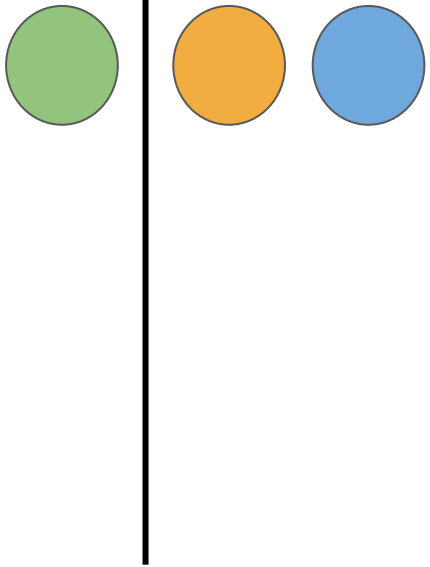
Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.



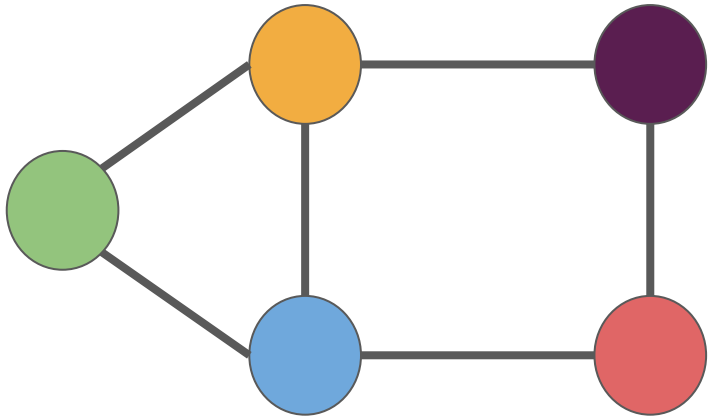
Map<Node, Set<Node>>

Node	Set<Node>>
Node	Adjacent to



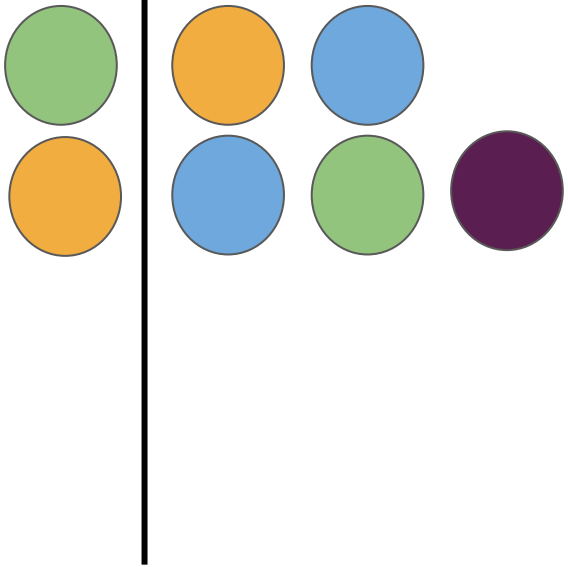
Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.



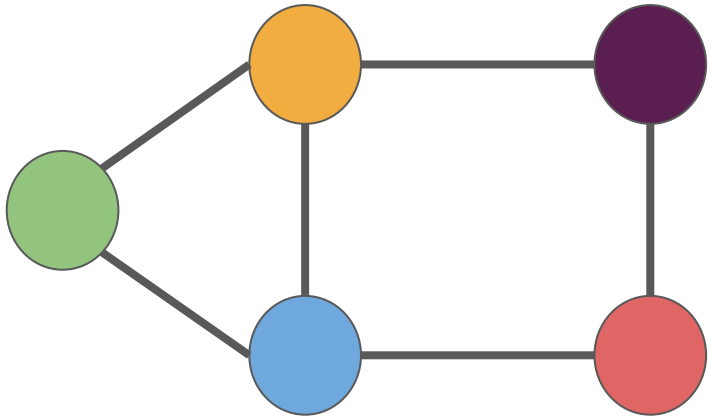
`Map<Node, Set<Node>>`

Node	Set<Node>
Node	Adjacent to

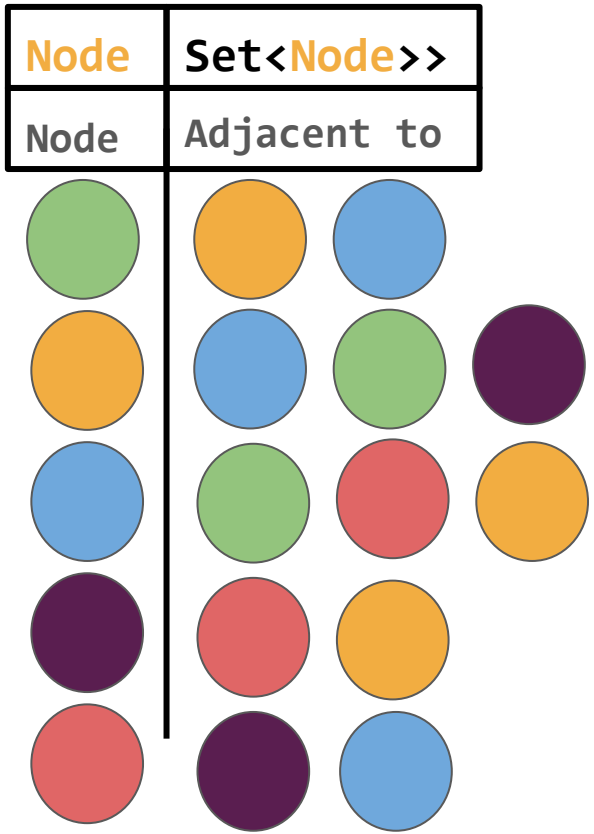


Approach 1: Adjacency List

- We can represent a graph as a map from nodes to the collection of nodes that each node is adjacent to.



Map<Node, Set<Node>>



Approach 1: Adjacency List

- The approach we just saw is called an adjacency list in comes in a number of different forms:
  - `Map<Node, Set<Node>>`
  - `Map<Node, Vector<Node>>`
  - `HashMap<Node, HashSet<Node>>`
  - `Vector<Node>` `<- in this case, the Node struct holds collection of its adjacent neighbors`

Approach 1: Adjacency List

- The approach we just saw is called an adjacency list in comes in a number of different forms:
  - `Map<Node, Set<Node>>`
  - `Map<Node, Vector<Node>>`
  - `HashMap<Node, HashSet<Node>>`
  - `Vector<Node>` `<- in this case, the Node struct holds collection of its adjacent neighbors`

- The core idea is that we have some kind of mapping associating each node with its outgoing edges (or neighboring nodes).

Approach 1: Adjacency List

- The approach we just saw is called an adjacency list in comes in a number of different forms:
  - `Map<Node, Set<Node>>`
  - `Map<Node, Vector<Node>>`
  - `HashMap<Node, HashSet<Node>>`
  - `Vector<Node>` `<- in this case, the Node struct holds collection of its adjacent neighbors`

- The core idea is that we have some kind of mapping associating each node with its outgoing edges (or neighboring nodes).

- How might you incorporate weights?

Approach 1: Adjacency List

- The approach we just saw is called an adjacency list in comes in a number of different forms:
  - `Map<Node, Set<Edge>>`
  - `Map<Node, Vector<Edge>>`
  - `HashMap<Node, HashSet<Edge>>`
  - `Vector<Node>` `<- in this case, the Node struct holds collection of its adjacent neighbors`

- The core idea is that we have some kind of mapping associating each node with its outgoing edges (or neighboring nodes).

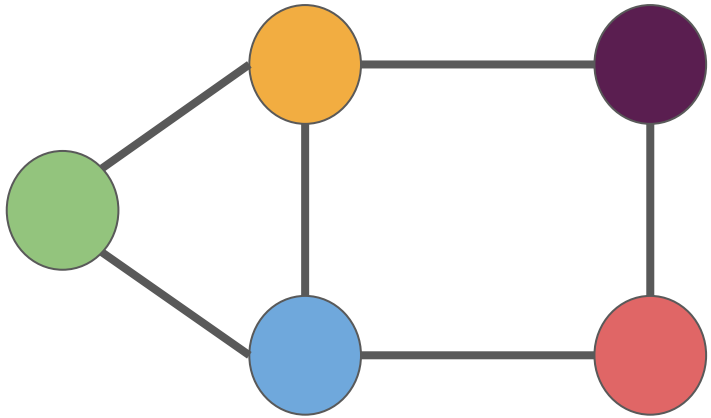
- **Create an Edge struct that holds both a Node and a weight**

Approach 2: Adjacency Matrix

- We can also use a two-dimensional matrix to represent the relationships in a graph.

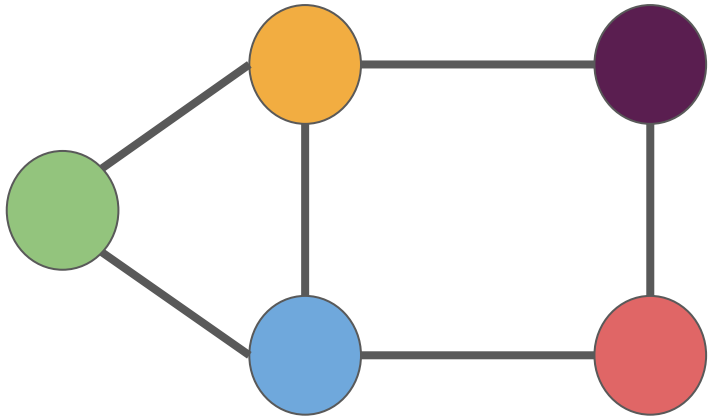
Approach 2: Adjacency Matrix

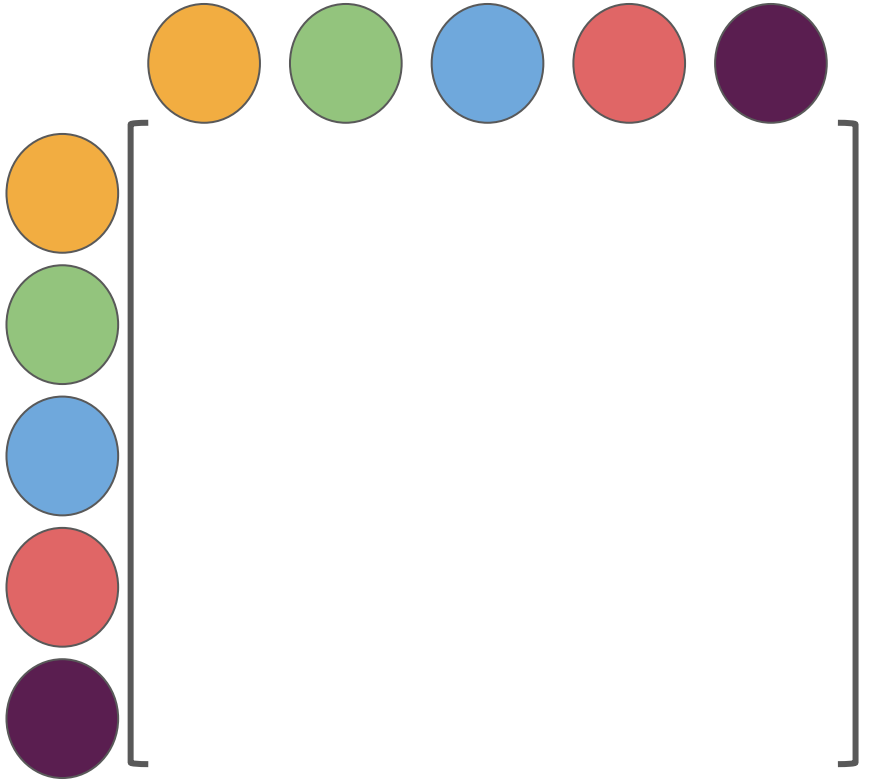
- We can also use a two-dimensional matrix to represent the relationships in a graph.



Approach 2: Adjacency Matrix

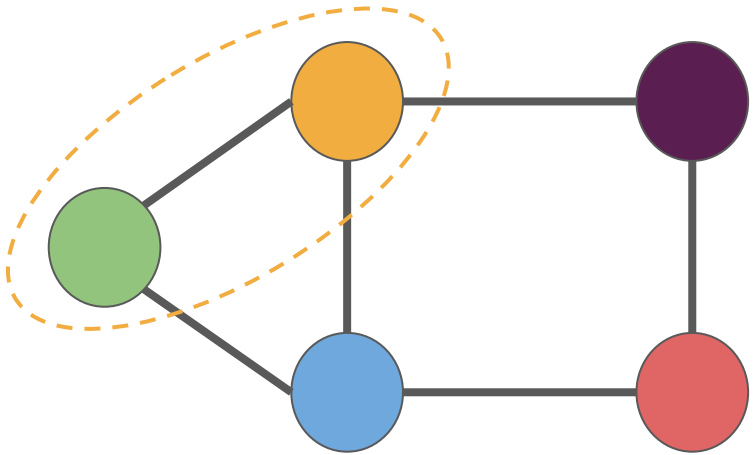
- We can also use a two-dimensional matrix to represent the relationships in a graph.

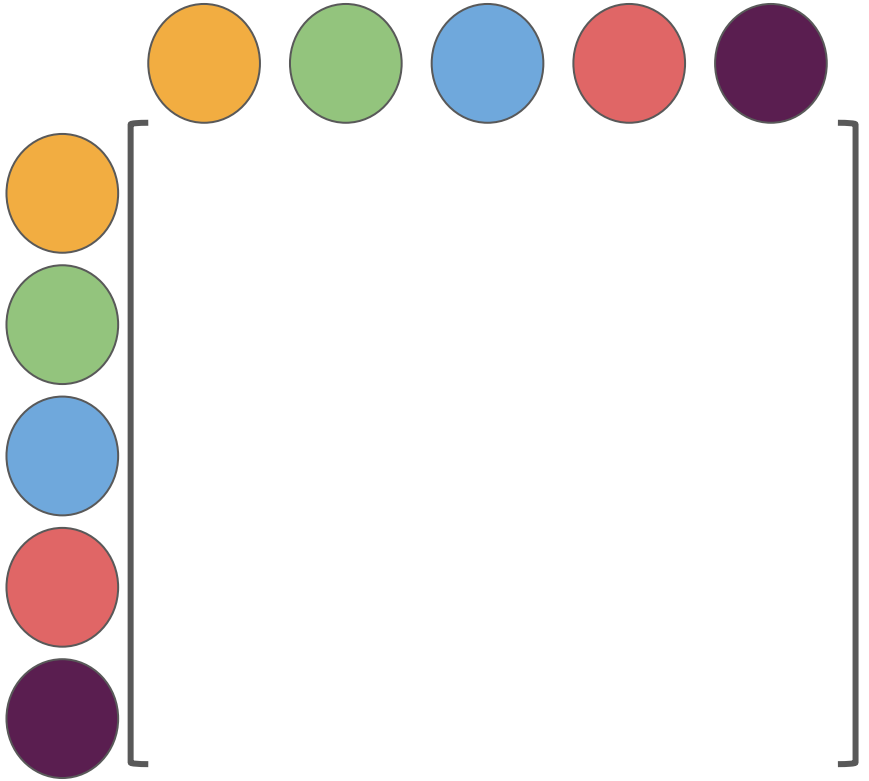




Approach 2: Adjacency Matrix

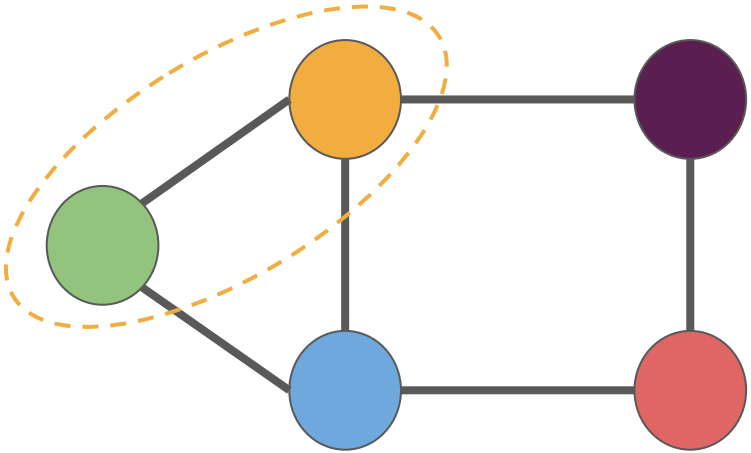
- We can also use a two-dimensional matrix to represent the relationships in a graph.

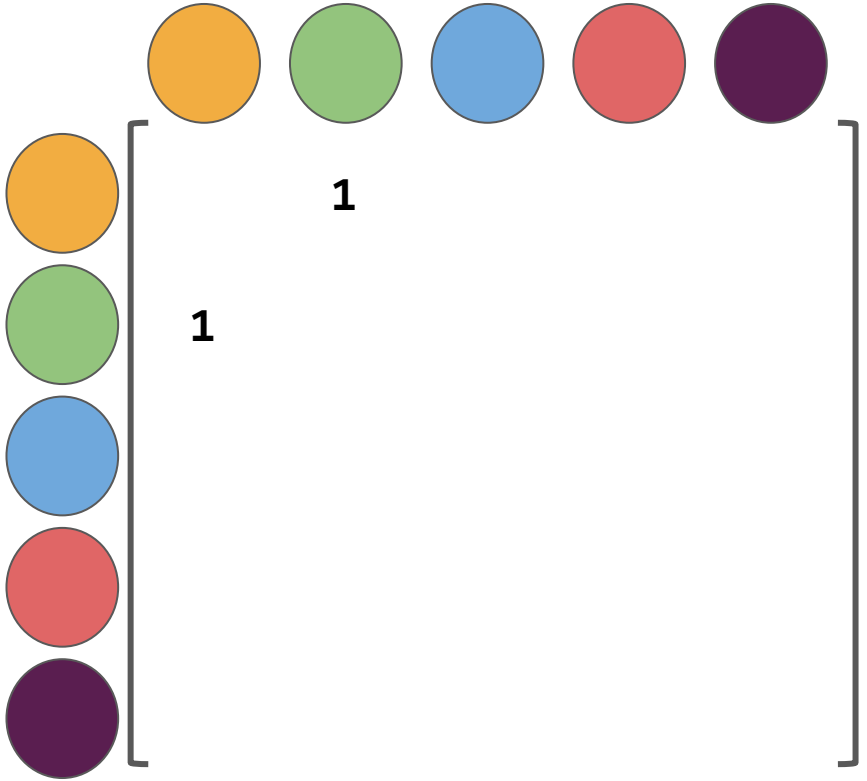




Approach 2: Adjacency Matrix

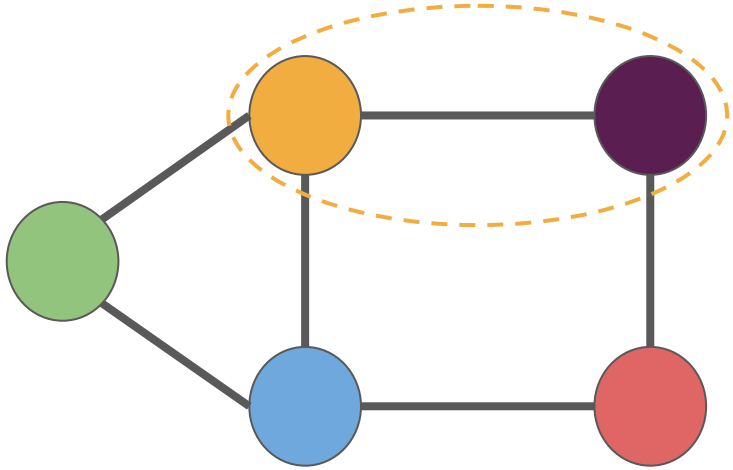
- We can also use a two-dimensional matrix to represent the relationships in a graph.

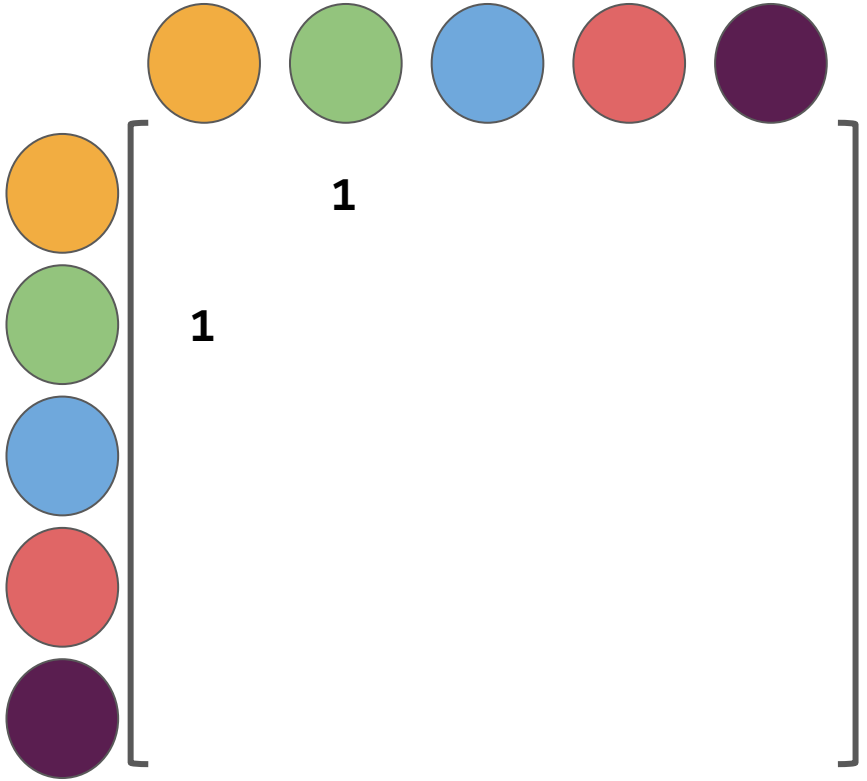




Approach 2: Adjacency Matrix

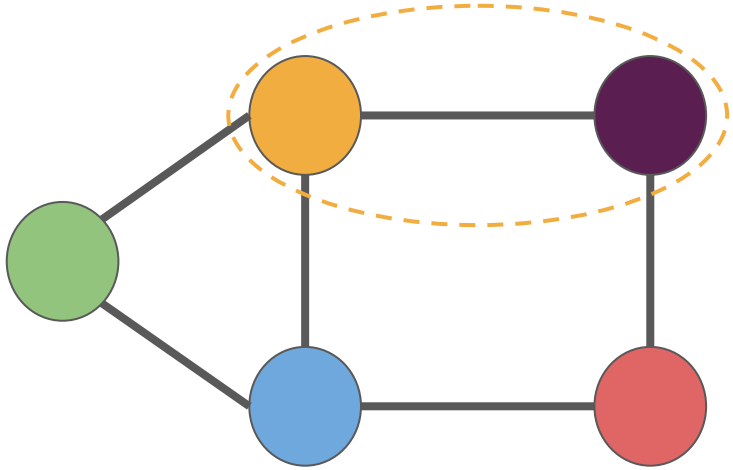
- We can also use a two-dimensional matrix to represent the relationships in a graph.

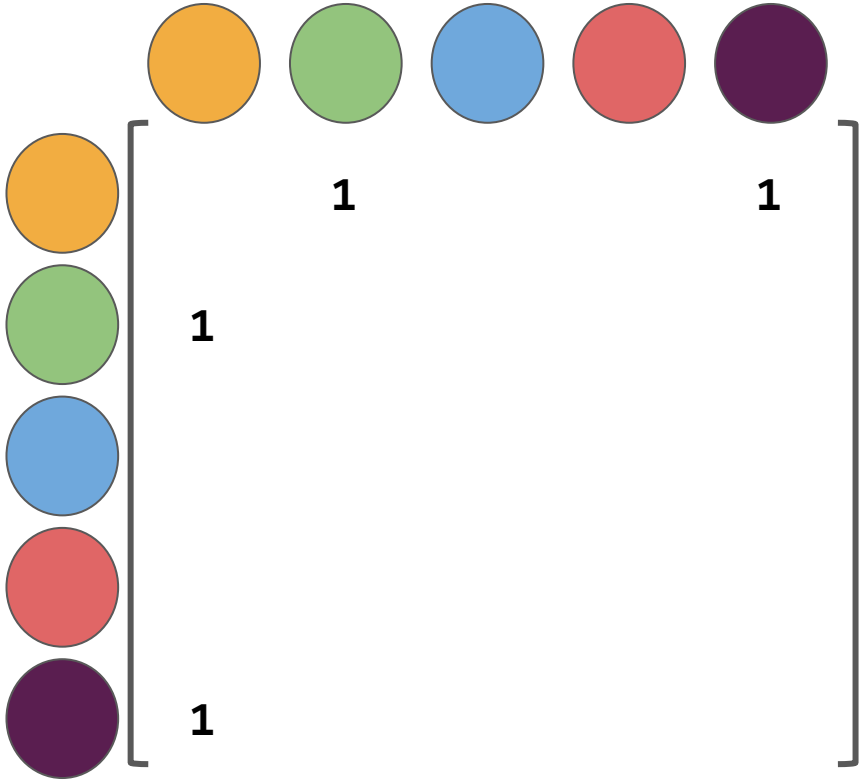




Approach 2: Adjacency Matrix

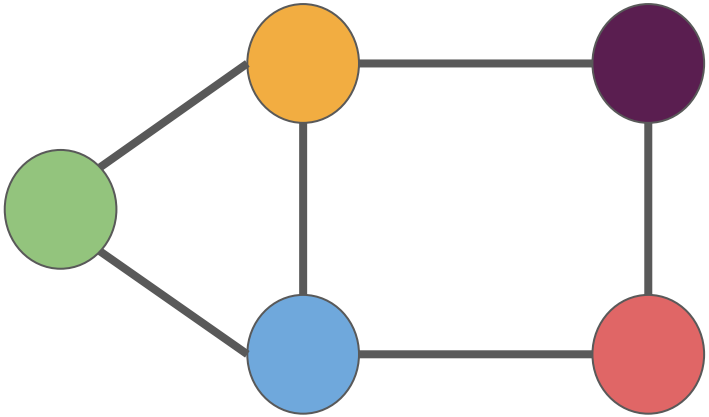
- We can also use a two-dimensional matrix to represent the relationships in a graph.

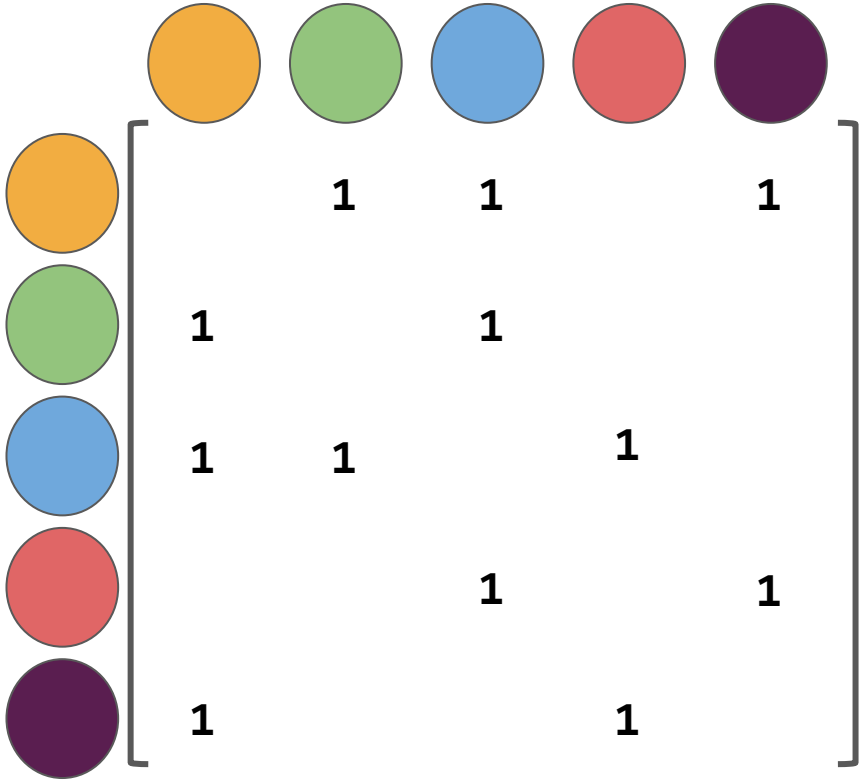




Approach 2: Adjacency Matrix

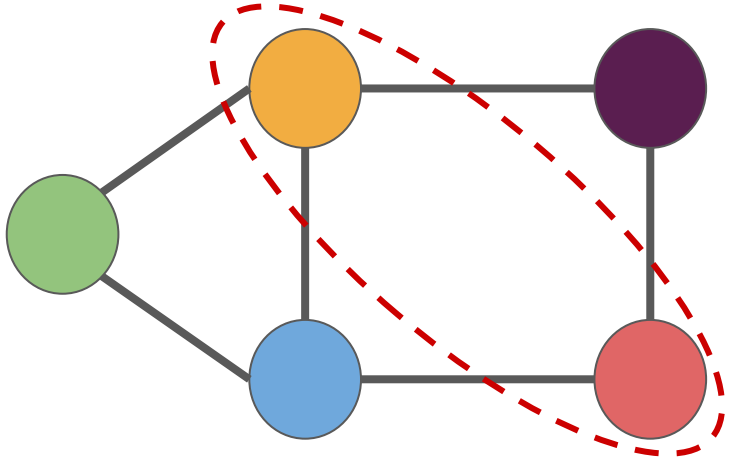
- We can also use a two-dimensional matrix to represent the relationships in a graph.

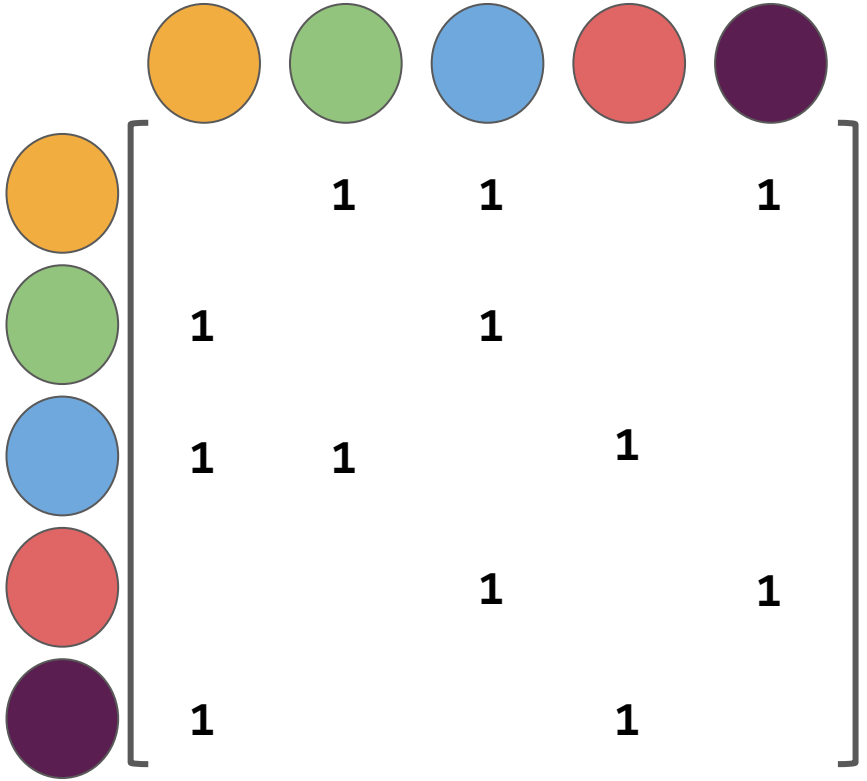




Approach 2: Adjacency Matrix

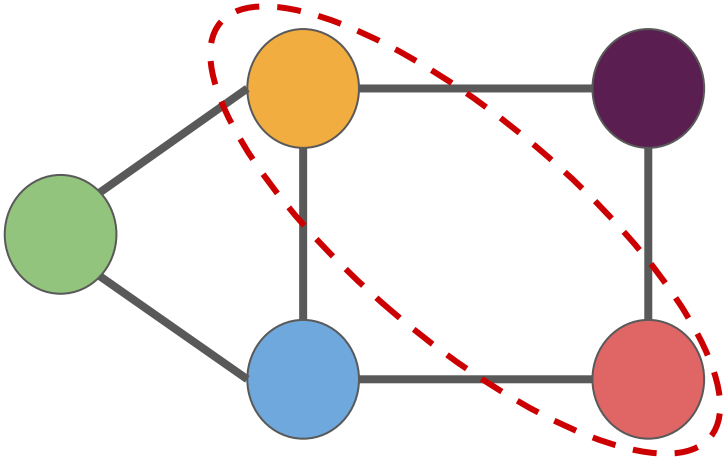
- We can also use a two-dimensional matrix to represent the relationships in a graph.

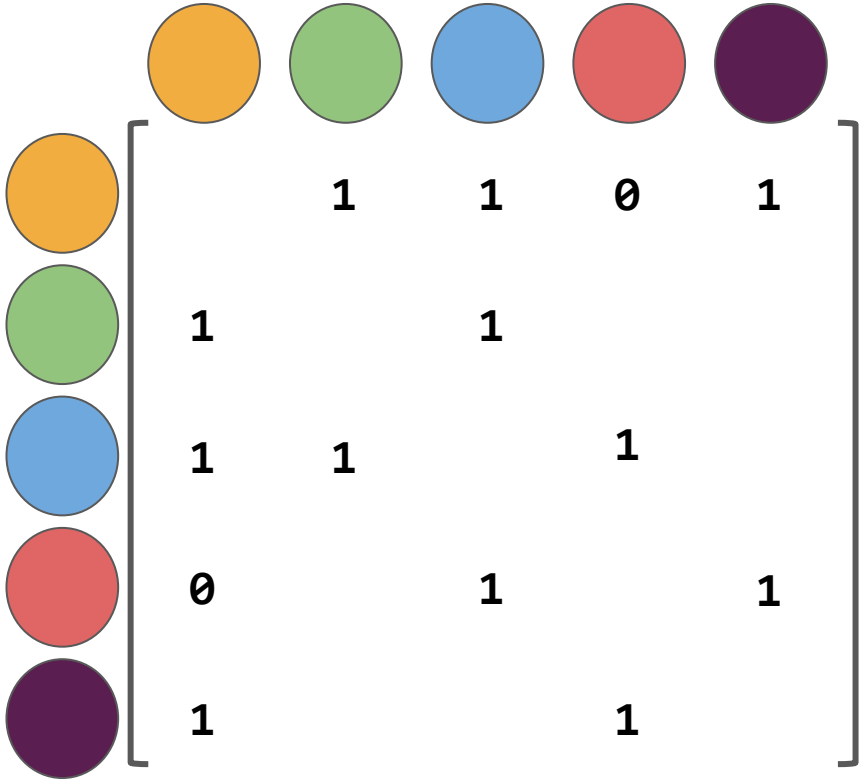




Approach 2: Adjacency Matrix

- We can also use a two-dimensional matrix to represent the relationships in a graph.





Approach 2: Adjacency Matrix

- To add weights, store other numbers besides 1 in the matrix.

- Adjacency matrices are beneficial when our graph isn't **sparse**, i.e. there aren't a lot of 0s. Otherwise, storing a mostly-0s matrix is not space efficient.

- Other benefits:
  - Grid lookup is super fast!
  - Storing weights is more straightforward than in the adjacency list.
  - Computer hardware has been optimized for matrix math - so using a grid can help us perform complex matrix operations for data analysis.

Graph Algorithms

Graph Algorithms

- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

Graph Algorithms

- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

- Some famous examples include:

Graph Algorithms

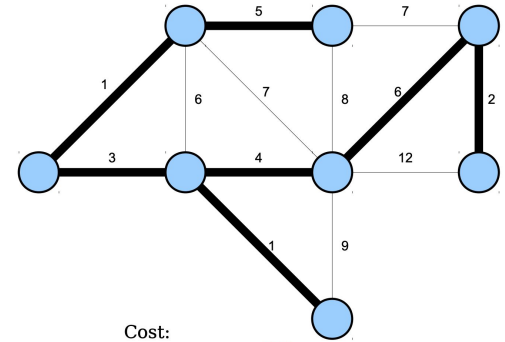
- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

- Some famous examples include:
  - **BFS, Dijkstra's algorithm, and A* Search**: Find the shortest path between two nodes in a graph.

Graph Algorithms

- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

- Some famous examples include:
  - **BFS, Dijkstra's algorithm, and A* Search**: Find the shortest path between two nodes in a graph.
  - **Kruskal's Algorithm**: Find a minimum spanning tree from a given graph.

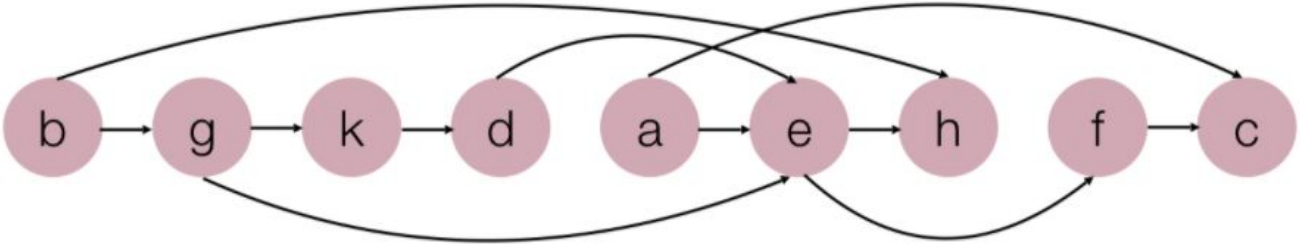


Cost:
1 + 3 + 5 + 4 + 1 + 6 + 2 = **22**

Graph Algorithms

- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

- Some famous examples include:
  - **BFS, Dijkstra's algorithm, and A* Search**: Find the shortest path between two nodes in a graph.
  - **Kruskal's Algorithm**: Find a minimum spanning tree from a given graph.
  - **Topological Sort**: "Sort" the nodes in a dependency graph in such a way that traversing the nodes in order results in all dependencies being fulfilled at each point in time.



Graph Algorithms

- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

- Some famous examples include:
  - **BFS, Dijkstra's algorithm, and A* Search**: Find the shortest path between two nodes in a graph.
  - **Kruskal's Algorithm**: Find a minimum spanning tree from a given graph.
  - **Topological Sort**: "Sort" the nodes in a dependency graph in such a way that traversing the nodes in order results in all dependencies being fulfilled at each point in time.
  - **Traveling salesman**: Given a map of cities and the distances between them, find the shortest path that traverses all cities in the map.

Graph Algorithms

- There are many, many different graph algorithms out there.
  - Check out this graph search algorithms demo.

- Some famous examples include:
  - **BFS, Dijkstra's algorithm, and A* Search**: Find the shortest path between two nodes in a graph.
  - **Kruskal's Algorithm**: Find a minimum spanning tree from a given graph.
  - **Topological Sort**: "Sort" the nodes in a dependency graph in such a way that traversing the nodes in order results in all dependencies being fulfilled at each point in time.
  - **Traveling salesman**: Given a map of cities and the distances between them, find the shortest path that traverses all cities in the map.

- Graphs can also be used in conjunction with machine learning algorithms to accomplish cool things. Take CS224W to learn more!

Announcements

Announcements

- Assignment 5 revisions and Assignment 6 are due today at 11:59pm PDT. Remember that this is a hard deadline, and there is no grace period!

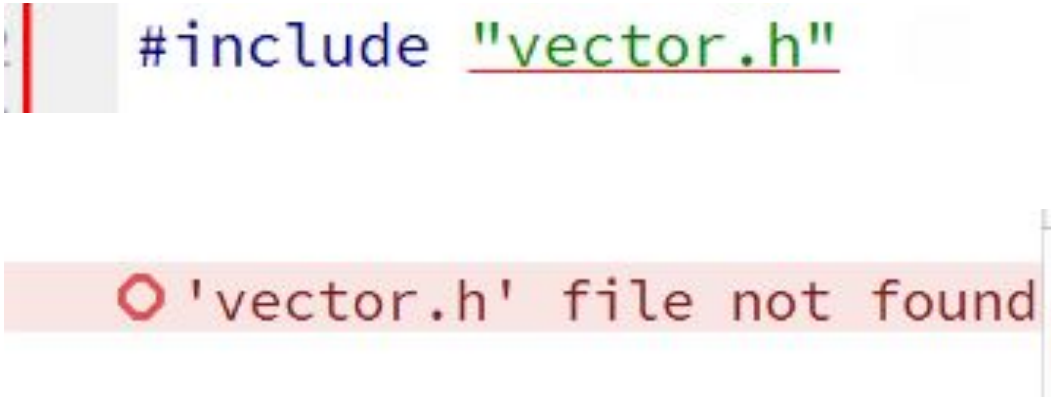
- There is no official section this week, but keep an eye out for an email from your SL's in case they are hosting an optional section, or if the section time is being used for Final Project presentations.

- Submit questions for tomorrow's Ask Us Anything here (also in this week's announcements). No lecture on Thursday so tomorrow is our last day of class.

- Tomorrow will be our last group OH (Kylie and Jenny will be there!).

Making our own projects!

```
#include "vector.h"
```


#include "vector.h"
'vector.h' file not found

```
#include "vector.h"
```

'vector.h' file not found

?????????

```
#include "vector.h"
```

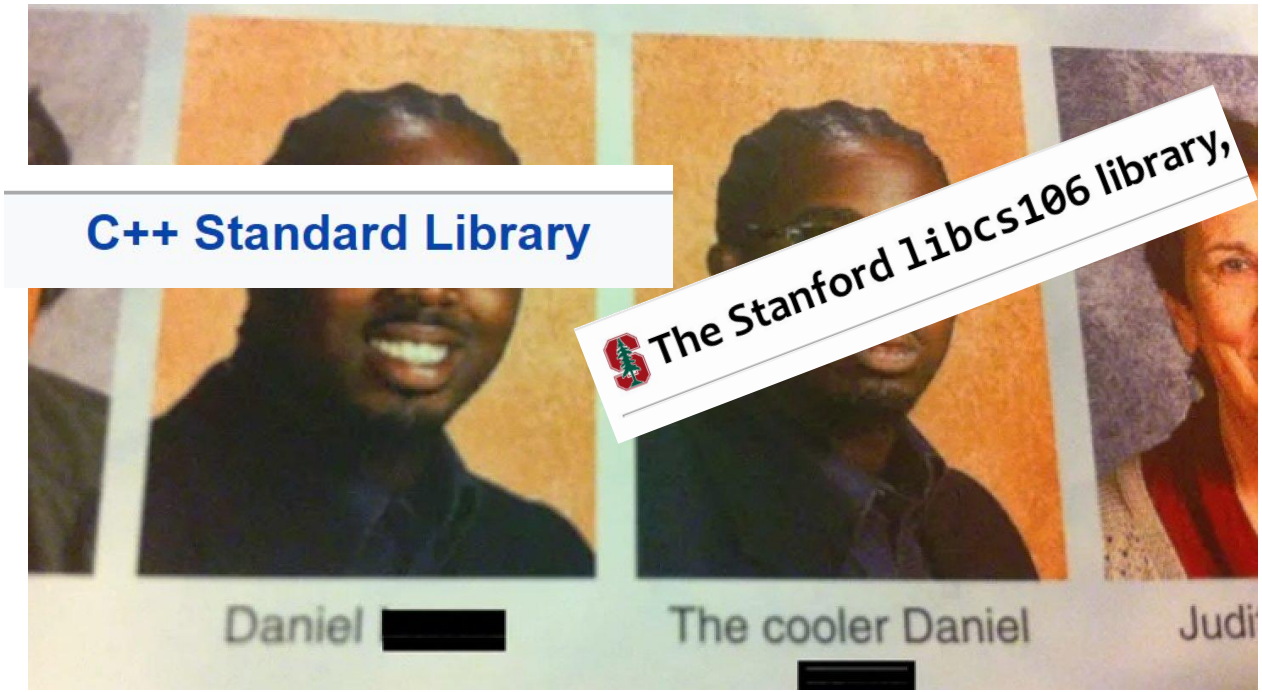
```
'vector.h' file not found
```

?????????

The Standard Template Library

“using namespace std??”

What we know so far



How do I use STL ADTs?

A 5 step guide:

1. Include your ADT

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
    a. **#include <vector>**

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
    a. **#include <vector>**
2. Write the ADT in lowercase

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
    a. **#include <vector>**
2. Write the ADT in lowercase
    a. **vector<int> myVector;**

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
   a. **#include <vector>**
2. Write the ADT in lowercase
   a. **vector<int> myVector;**
3. Put std:: in front of it

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
    a. **#include <vector>**
2. Write the ADT in lowercase
    a. **vector<int> myVector;**
3. Put std:: in front of it
    a. **std::vector<int> myVector;**

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
    a. **#include <vector>**
2. Write the ADT in lowercase
    a. **vector<int> myVector;**
3. Put std:: in front of it
    a. **std::vector<int> myVector;**
4. ???

How do I use STL ADTs?

A 5 step guide:

1. Include your ADT
   a. **#include <vector>**
2. Write the ADT in lowercase
   a. **vector<int> myVector;**
3. Put std:: in front of it
   a. **std::vector<int> myVector;**
4. ???
5. Profit!

Spot the difference!

What you want to do	Stanford `Vector<int>`	`std::vector<int>`
Create a new, empty vector	`Vector<int> vec;`	`std::vector<int> vec;`
Create a vector with `n` copies of 0	`Vector<int> vec(n);`	`std::vector<int> vec(n);`
Create a vector with `n` copies of a value `k`	`Vector<int> vec(n, k);`	`std::vector<int> vec(n, k);`
Add a value `k` to the end of a vector	`vec.add(k);`	`vec.push_back(k);`
Remove all elements of a vector	`vec.clear();`	`vec.clear();`
Get the element at index `i`	`int k = vec[i];`	`int k = vec[i];` (does **not** bounds check)
Check size of vector	`vec.size();`	`vec.size();`
Loop through vector by index `i`	`for (int i = 0; i < vec.size(); ++i) ...`	`for (std::size_t i = 0; i < vec.size(); ++i) ...`
Replace the element at index `i`	`vec[i] = k;`	`vec[i] = k;` (does **not** bounds check)

Table courtesy of Frankie Cerkvenik and Sathya Edamadaka!

Makefiles

How does our code actually compile?

Makefiles and cmake

A Makefile is the recipe for your build!

It tells the compiler:

- What files to include
- What dependencies they have
- What code to run to put it all together

```
targets: prerequisites
  command
  command
  command
```

Makefiles and cmake

A Makefile is the recipe for your build!

It tells the compiler:

- What files to include
- What dependencies they have
- What code to run to put it all together

These can be a little manual – **cmake** abstracts a lot of it for you!

Example cmake file (from CS106L)

```
cmake_minimum_required(VERSION 3.0)
project(wikiracer)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED True)

find_package(cpr CONFIG REQUIRED)

# adding all files
add_executable(main main.cpp wikiscraper.cpp.o error.cpp)

target_link_libraries(main PRIVATE cpr)
```

Example cmake file (from CS106L)

```
cmake_minimum_required(VERSION 3.0)
project(wikiracer)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED True)

find_package(cpr CONFIG REQUIRED)

# adding all files
add_executable(main main.cpp wikiscraper.cpp.o error.cpp)

target_link_libraries(main PRIVATE cpr)
```

Take CS 107/107E next to learn more about compiling!

Leveling up classes

Speaking of vector<int>...

What’s up with <>?

When we make classes, we can initialize them in the constructor with some parameters!

What’s up with <>?

When we make classes, we can initialize them in the constructor with some parameters!

These have to have a certain type…

What’s up with <>?

When we make classes, we can initialize them in the constructor with some parameters!

These have to have a certain type…

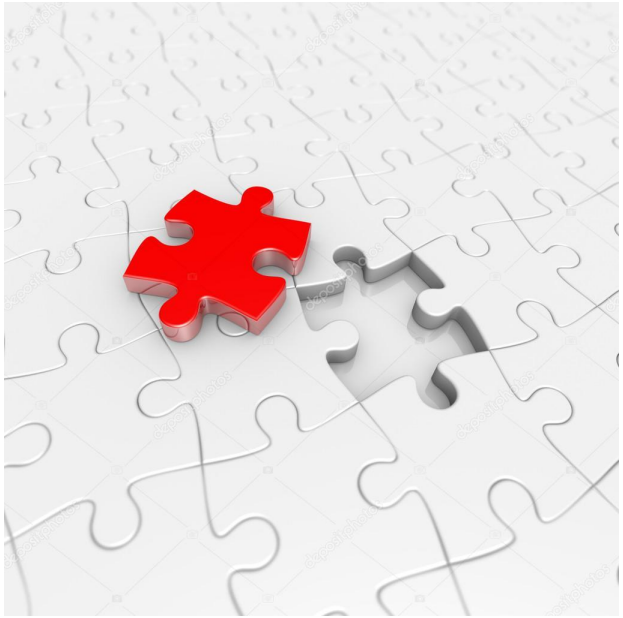
How can we make a class that can take in any type of parameters?

Template classes!

All of our favorite ADTs are template classes!

Pros:

- Can take in any type
- Generalized
- Easy for the client and the programmer



Template classes!

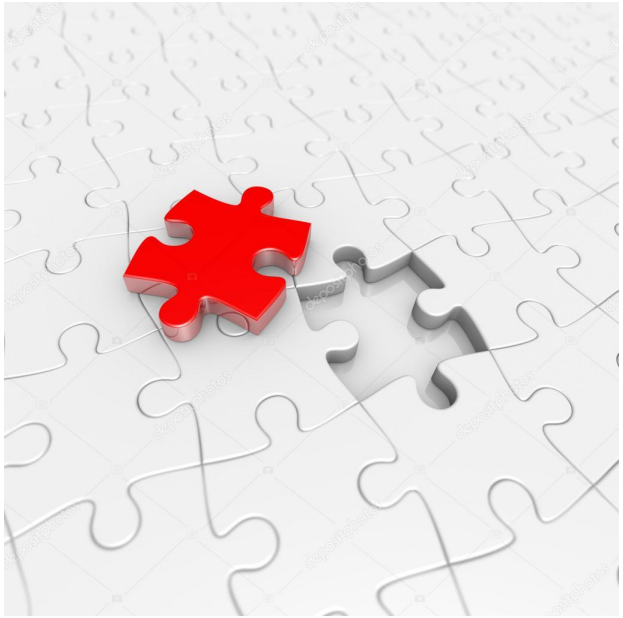
All of our favorite ADTs are template classes!

Pros:

- Can take in any type
- Generalized
- Easy for the client and the programmer

Cons:

- We don’t know how to make them yet??



Syntax

Instead of:

- class ClassName { ... }
- FIXED_TYPE var;

Use:

- template<class typeName> class ClassName { ... }
- typeName var;

Syntax

Instead of:

- class ClassName { … }
- FIXED_TYPE var;

Use:

- template<class typeName> class ClassName { … }
- typeName var;

Anywhere you use a fixed type, use typeName!

What about operators?

Want to learn more?

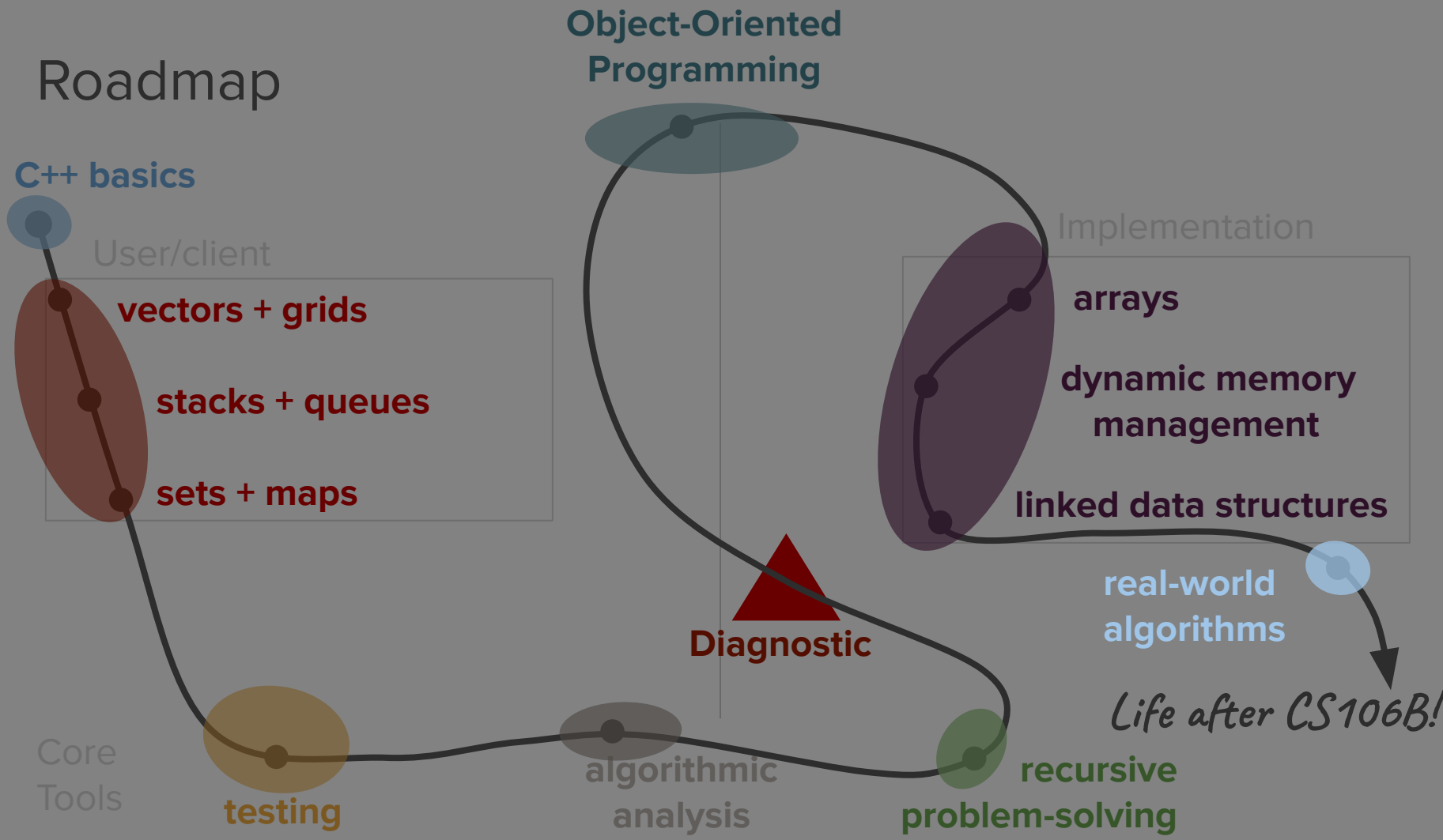
Take CS106L!

havenw@stanford.edu

Go forth and code!

What’s next?

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving