CSE 333
Section 6

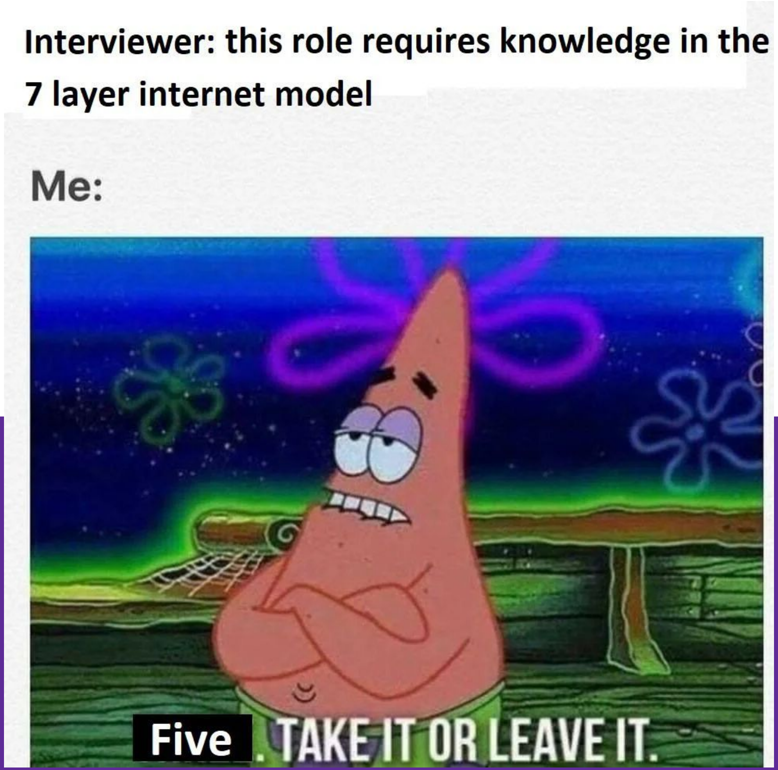
Network programming,
Inheritance, vtables (recap)

UNIVERSITY of WASHINGTON

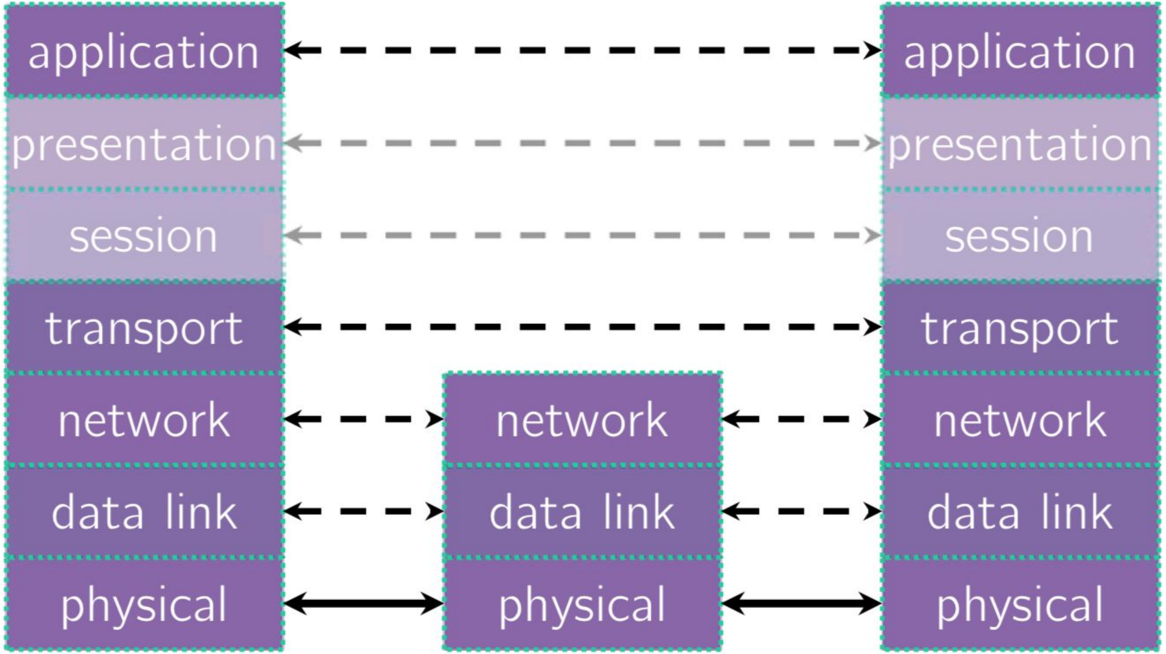
Logistics

- HW3:
  - Due **Today, 11:00 pm**
- Exercise 15:
  - Due (08/5) **Monday, 10 am**
- Exercise 16:
  - Due (08/07) Wednesday, 10 am

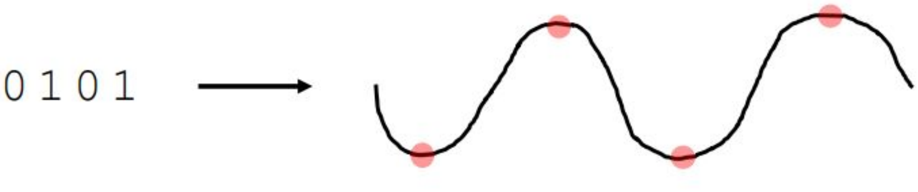
Computer Networking - At a High Level



Computer Networks: A 7-ish Layer Cake



Computer Networks: A 7-ish Layer Cake



Wires, radio signals, fiber optics

bit encoding at signal level

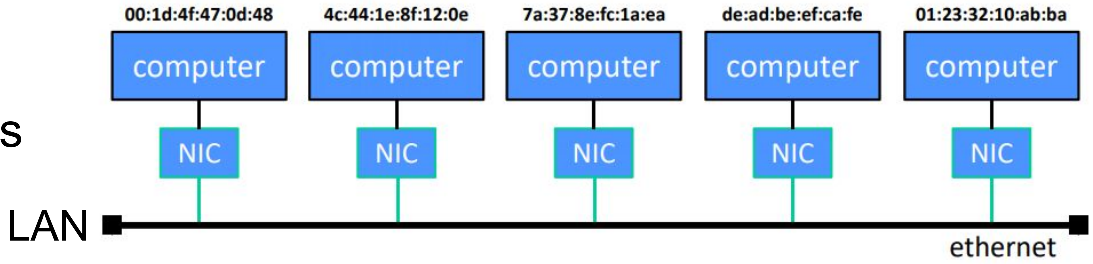
physical

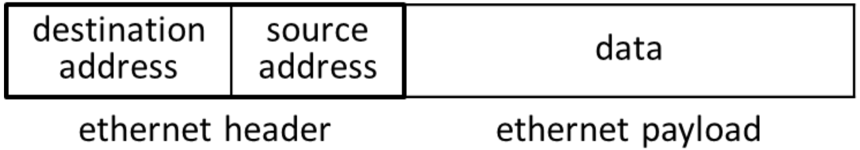
physical

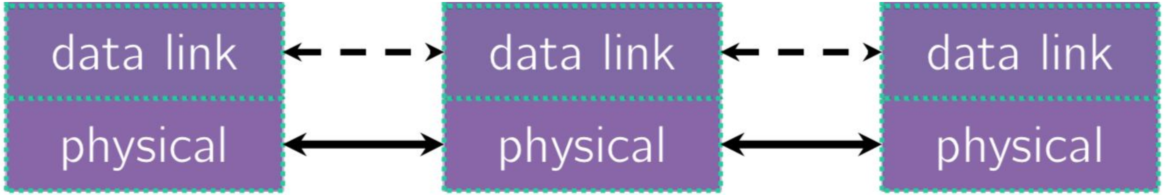
physical

Computer Networks: A 7-ish Layer Cake

WiFi, ethernet.
Connecting multiple computers



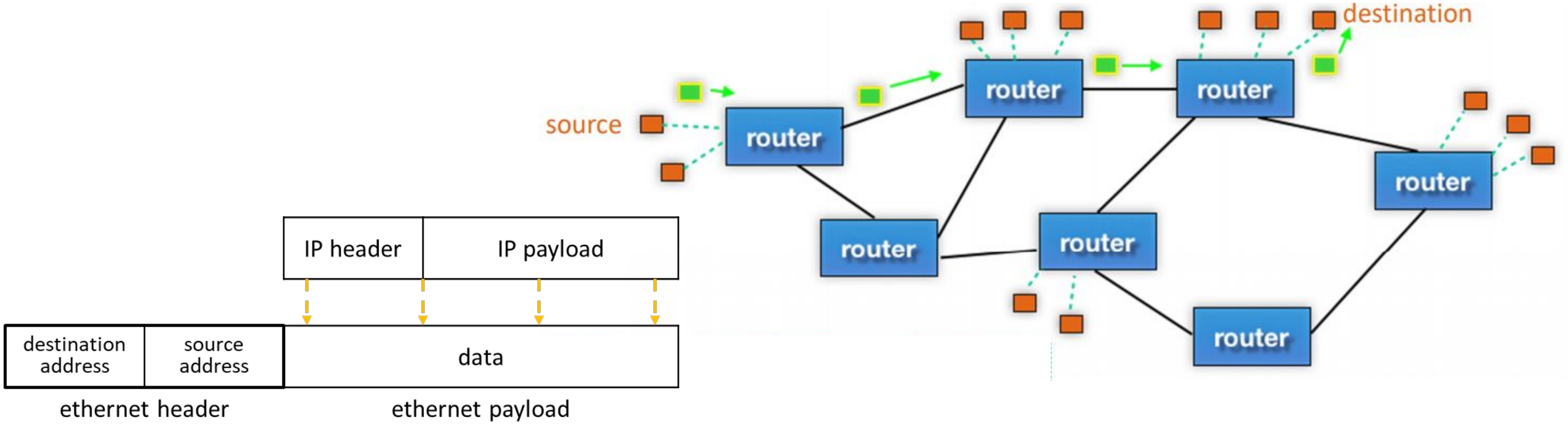




multiple computers on a local network

bit encoding at signal level

Computer Networks: A 7-ish Layer Cake

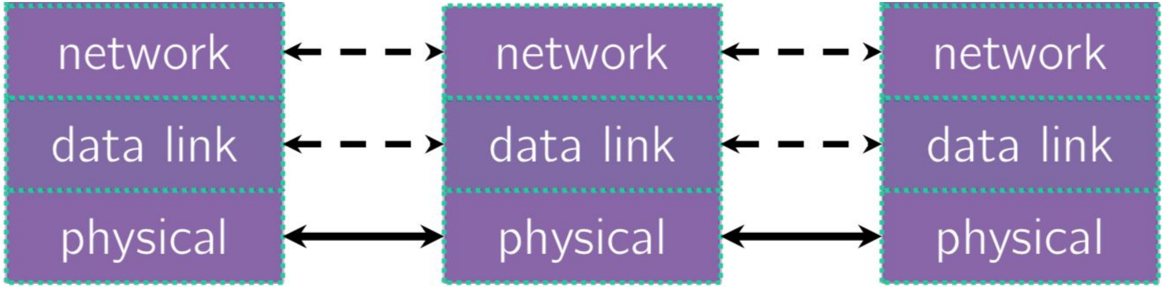


IP header | IP payload

destination address | source address | data

ethernet header

ethernet payload

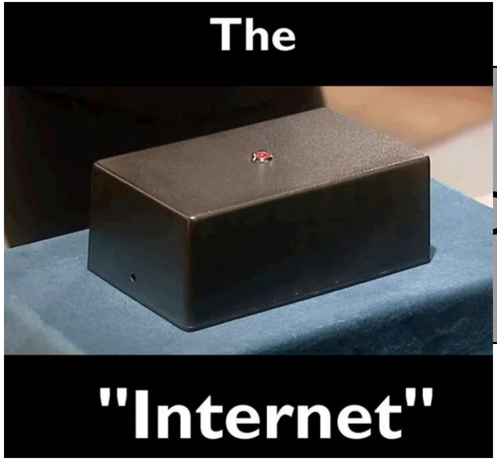


routing of packets across networks

multiple computers on a local network

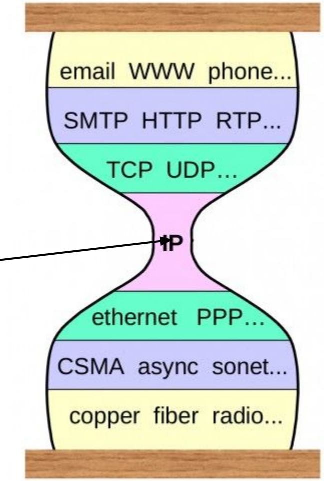
bit encoding at signal level

Computer Networks: A 7-ish Layer Cake



on/Interfa

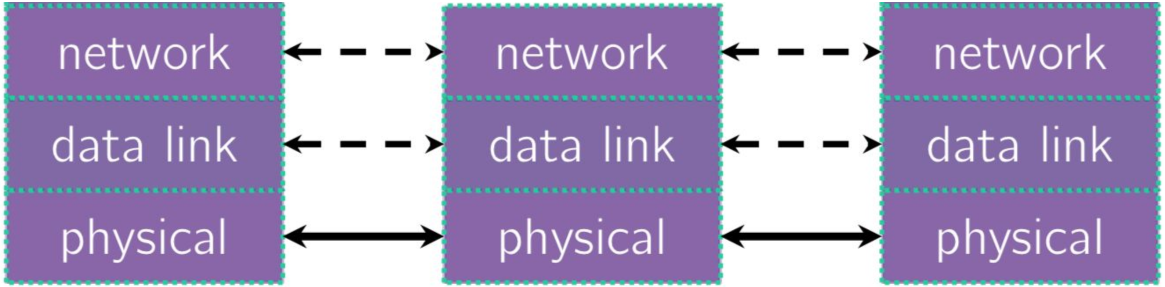
Backbone of the Internet!



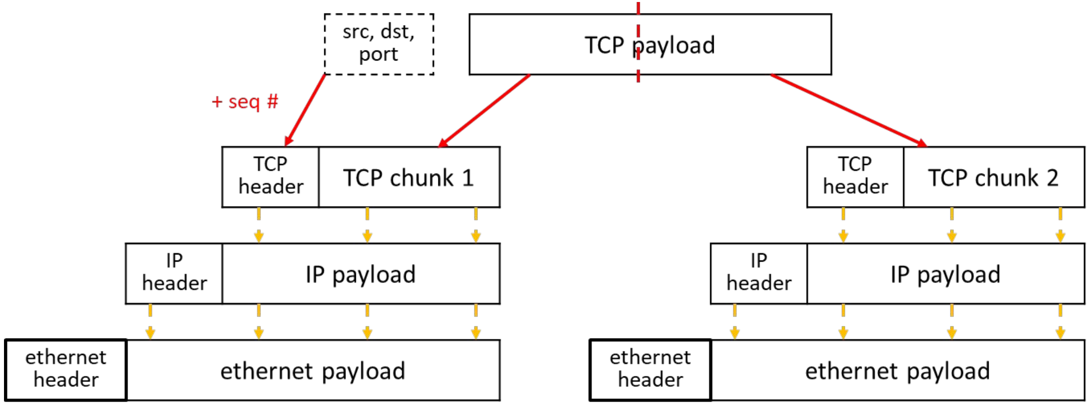
routing of packets across networks

multiple computers on a local network

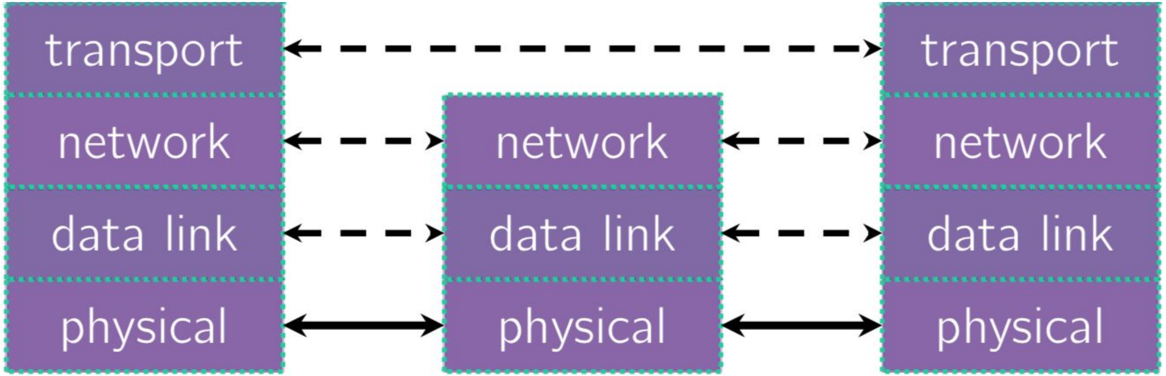
bit encoding at signal level



Computer Networks: A 7-ish Layer Cake



TCP, UDP, etc.



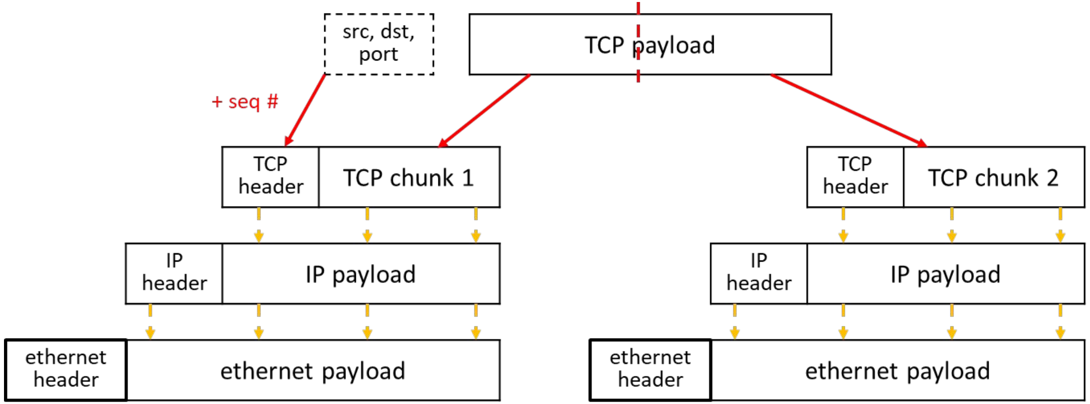
sending data end-to-end

routing of packets across networks

multiple computers on a local network

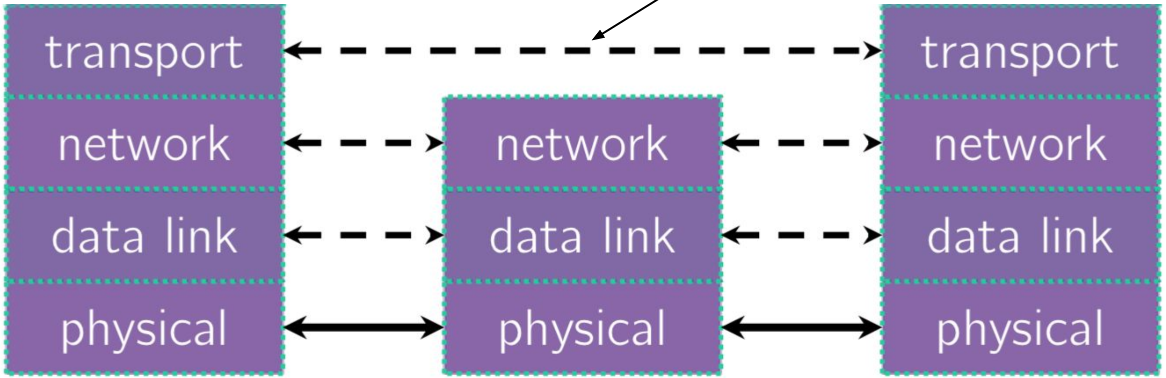
bit encoding at signal level

Computer Networks: A 7-ish Layer Cake



TCP, UDP, etc.

Stream abstraction!



sending data end-to-end

routing of packets across networks

multiple computers on a local network

bit encoding at signal level

Computer Networks: A 7-ish Layer Cake

HTTP, DNS, much more

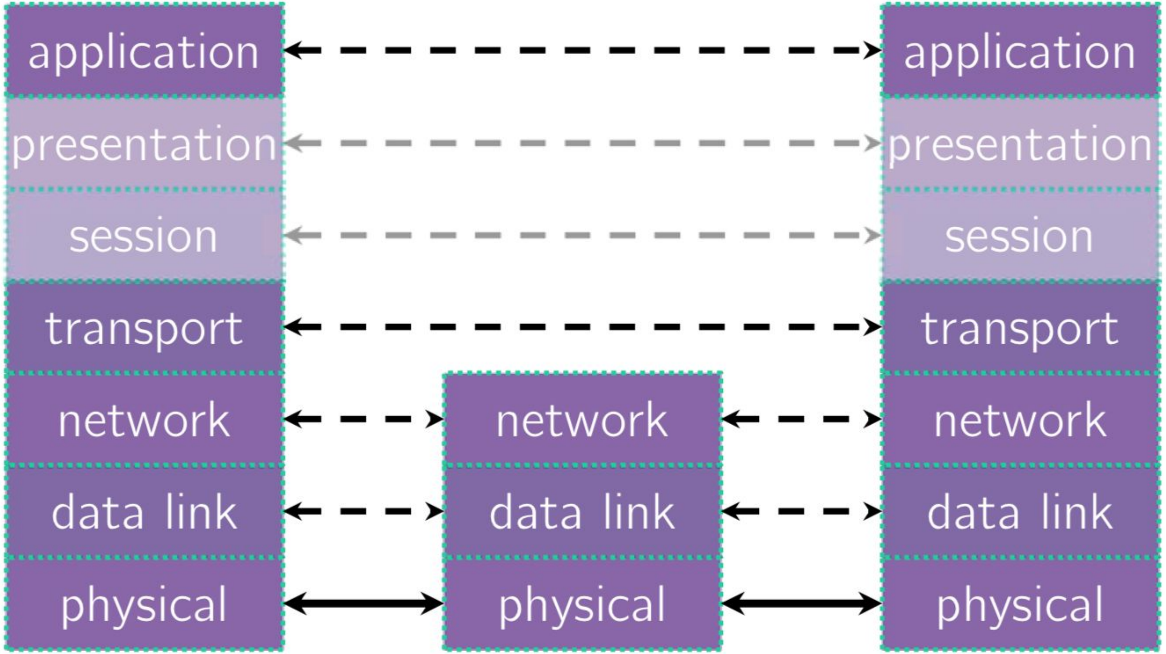
format/meaning of messages

sending data end-to-end

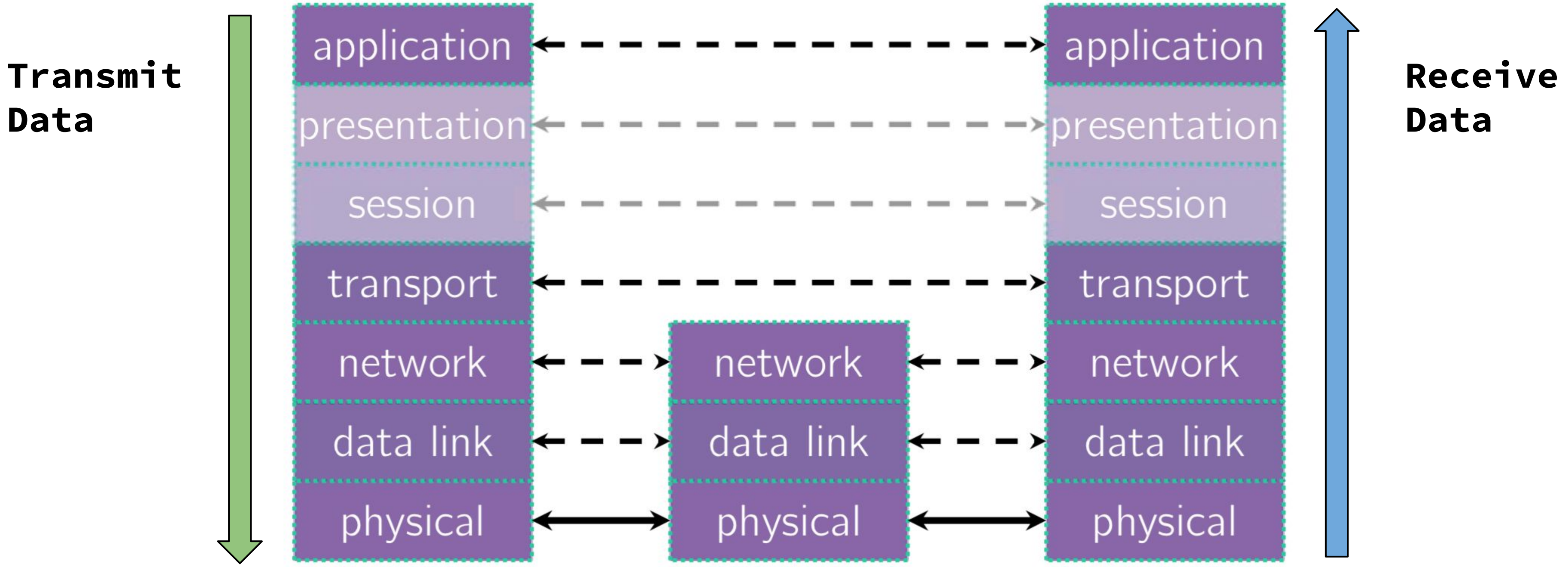
routing of packets across networks

multiple computers on a local network

bit encoding at signal level

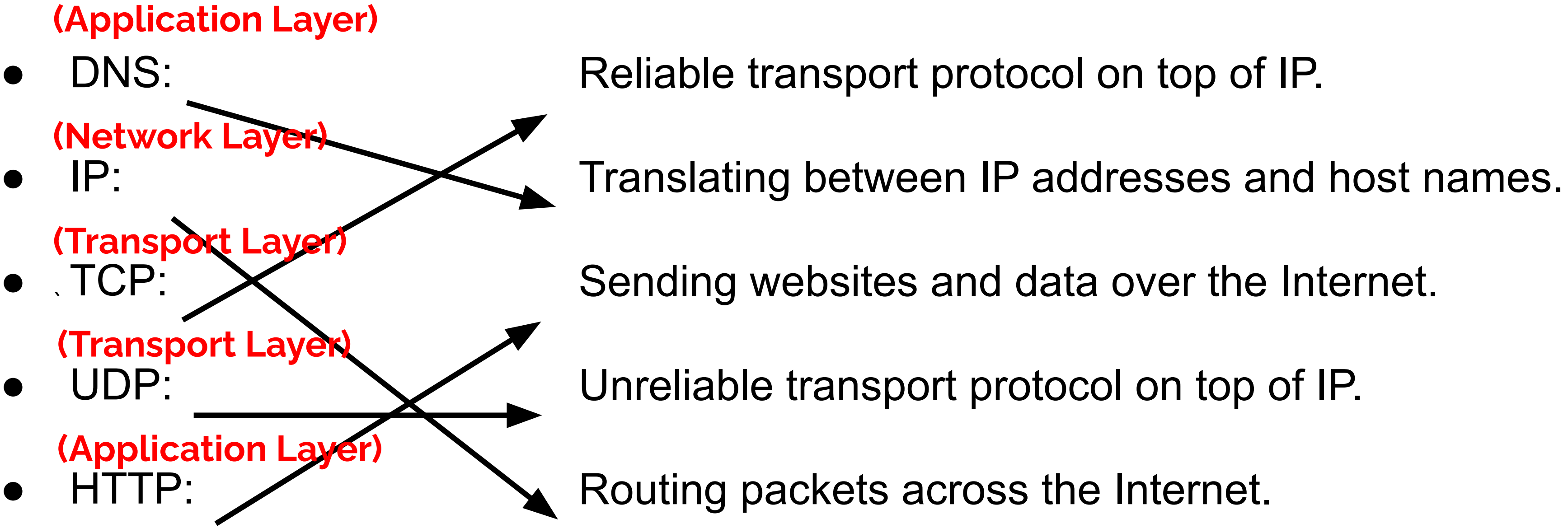


Data Flow


Transmit
Data
application
presentation
session
transport
network
data link
physical
network
data link
physical
application
presentation
session
transport
network
data link
physical
Receive
Data

Exercise 1

Exercise 1



(Application Layer)

- DNS:

(Network Layer)

- IP:

(Transport Layer)

- TCP:

(Transport Layer)

- UDP:

(Application Layer)

- HTTP:

Reliable transport protocol on top of IP.

Translating between IP addresses and host names.

Sending websites and data over the Internet.

Unreliable transport protocol on top of IP.

Routing packets across the Internet.

TCP versus UDP

Transmission Control Protocol (TCP):

- Connection-oriented service
- Reliable and Ordered
- Flow control

User Datagram Protocol (UDP):

- “Connectionless” service
- Unreliable packet delivery
- High speed, no feedback

TCP guarantees reliability for things like messaging or data transfers. UDP has less overhead since it doesn’t make those guarantees, but is often fine for streaming applications (e.g., YouTube or Netflix) or other applications that manage packets on their own or do not want occasional pauses for packet retransmission or recovery.

Netcat demo

netcat

- Command-line utility to setup a TCP/UDP connection to read/write data
    - Man page: https://www.commandlinux.com/man-page/man1/nc.1.html
- To start a server:
    - `nc -l <hostname> <port>`
- To connect to that server (as a client):
    - `nc <hostname> <port>`
- `<hostname>` can be:
    - `localhost`
    - `attu#.cs.washington.edu`

Inheritance (Recap)

Inheritance

- Motivation: Better modularize our code for similar classes!

- The public interface of a derived class inherits all **non-private** member variables and functions (**except** for `ctor`, `cctor`, `dtor`, `op=`) from its base class
  - *Similar to*: A subclass inherits from a superclass

- Aside: We will be only using **public, single** inheritance in CSE 333

Polymorphism: Dynamic Dispatch

- **Polymorphism** allows for you to access objects of related types (base and derived classes) – Allows interface usage instead of class implementation

- **Dynamic dispatch**: Implementation is determined *at* **runtime** via lookup
    - Allows you to call the **most-derived** version of the actual type of an object
    - Generally want to use this when you have a derived class

- `virtual` replaces the class's default **static dispatch** with **dynamic dispatch**
    - Static dispatch determines implementation at compile time
    - Meaning it does **not** use dynamic dispatch (just calls its function)

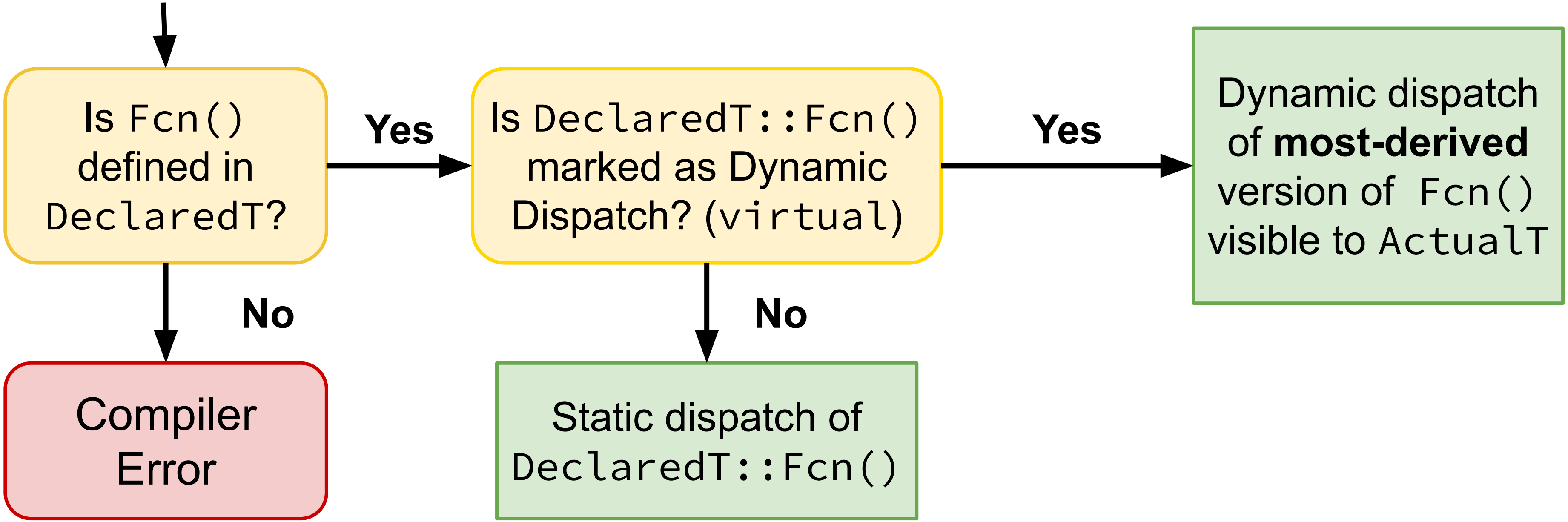
Dynamic Dispatch: Style Considerations

- Defining Dynamic Dispatch in your code base
  - Use `virtual` **only once** when first defined in the base class
    - (although in older code bases you may see it repeated on functions in subclasses)
  - All derived classes of a base class should use `override` to get the compiler to check that a function overrides a virtual function from a base class

- Use `virtual` for destructors of a base class – Guarantees all derived classes will use dynamic dispatch to ensure use of appropriate destructors

Dispatch Decision Tree

```
DeclaredT* ptr = new ActualT();
ptr->Fcn();  // which version is called?
```



Is `Fcn()` defined in `DeclaredT`?

Yes

Is `DeclaredT::Fcn()` marked as Dynamic Dispatch? (`virtual`)

Yes

Dynamic dispatch of **most-derived** version of `Fcn()` visible to `ActualT`

No

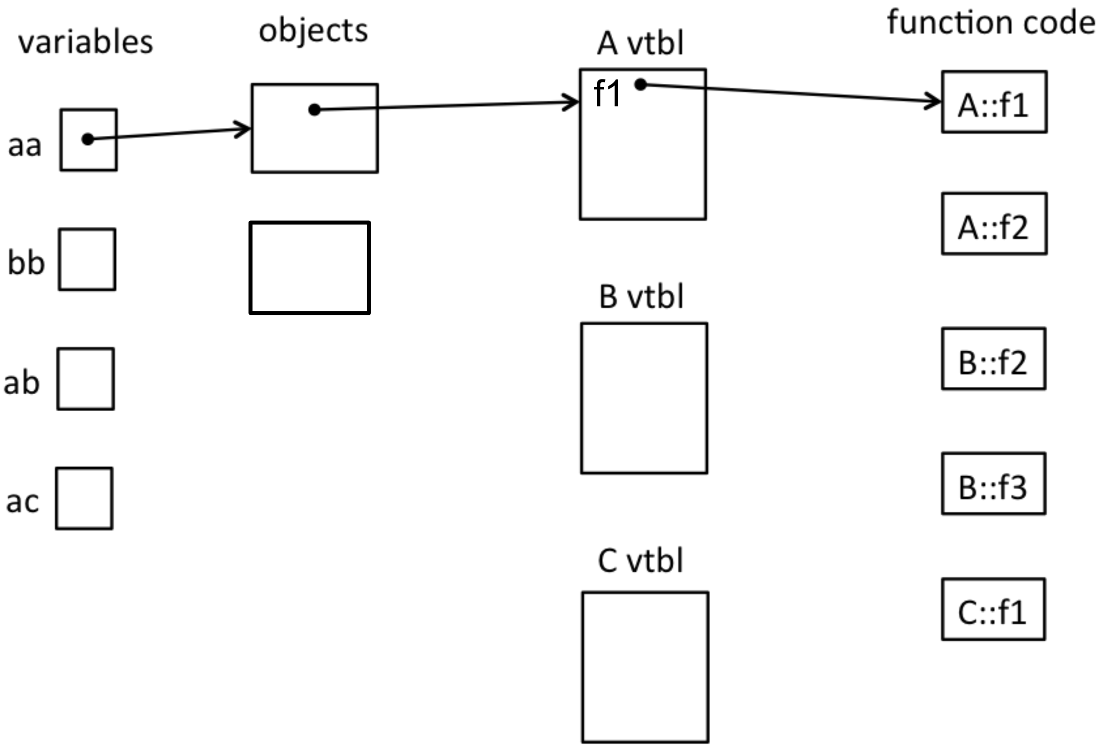
Compiler Error

No

Static dispatch of `DeclaredT::Fcn()`

Exercise 1

Exercise 1 (Drawing vtable diagram)



Exercise 1 Solution (pointers)

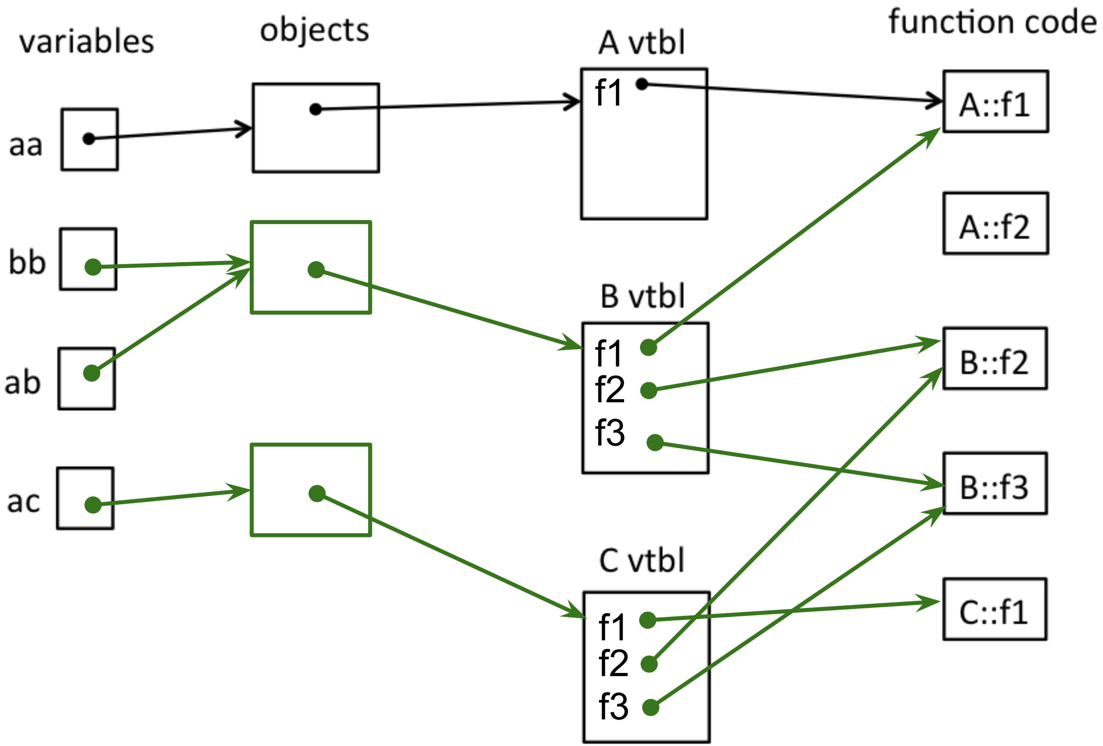
```
#include <iostream>
using namespace std;

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```

```
int main() {
  A* aa = new A();
  B* bb = new B();
  A* ab = bb;
  A* ac = new C();
```



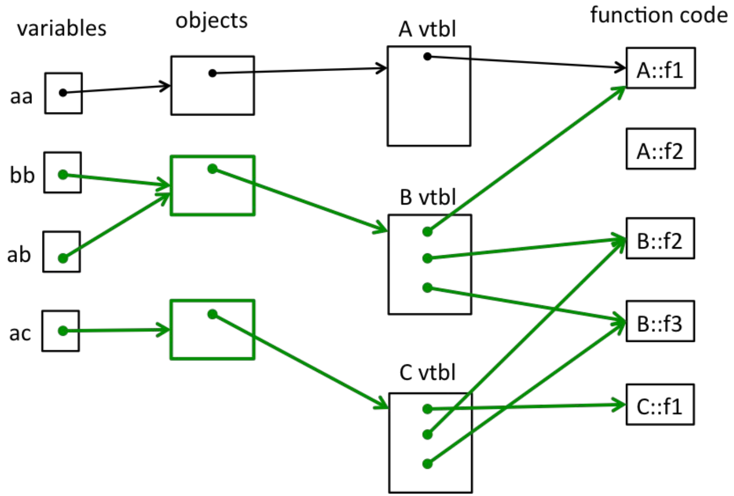
Exercise 1 Solution (output)

```
#include <iostream>
using namespace std;

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```



```
A* aa = new A();

aa->f1();
```

A	B	C	D
B::f2 A::f1	A::f2 C::f1	A::f2 A::f1	B::f2 C::f1

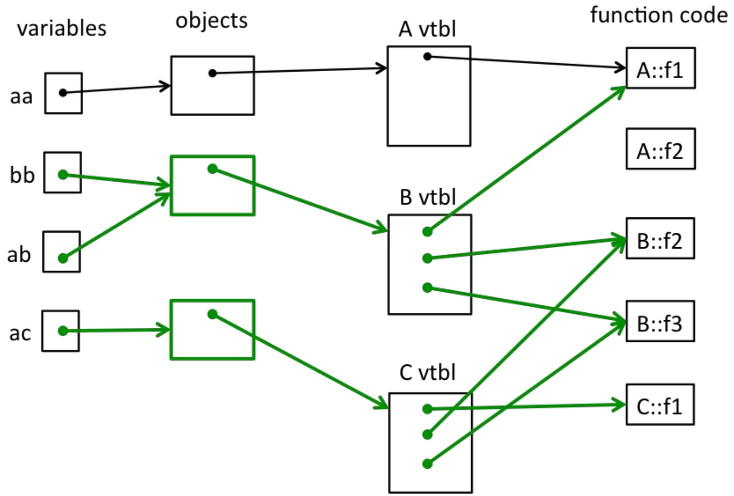
Exercise 1 Solution (output)

```
#include <iostream>
using namespace std;

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```



```
B* bb = new B();

bb->f1();
```

A	B	C	D
B::f2 A::f1	A::f2 C::f1	A::f2 A::f1	B::f2 C::f1

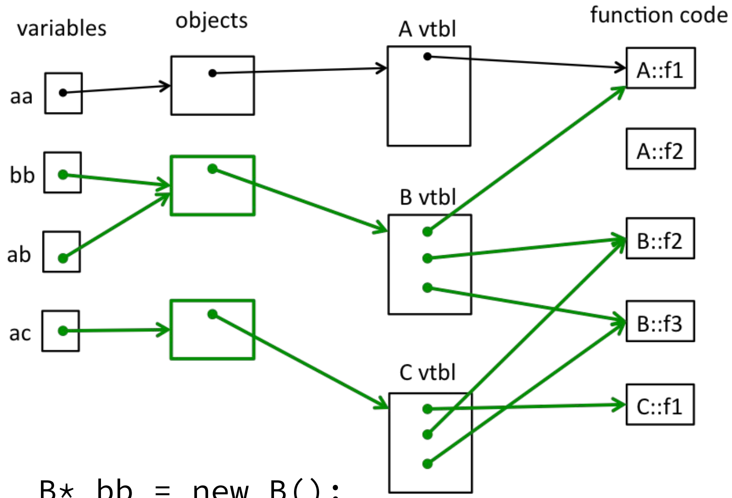
Exercise 1 Solution (output)

```
#include <iostream>
using namespace std;

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```



```
B* bb = new B();
A* ab = bb;

bb->f2();
cout << "----" << endl;
ab->f2();
```

A	B	C	D
B::f2 ---- B::f2	A::f2 ---- B::f2	B::f2 ---- A::f2	A::f2 ---- A::f2

Exercise 1 Extension

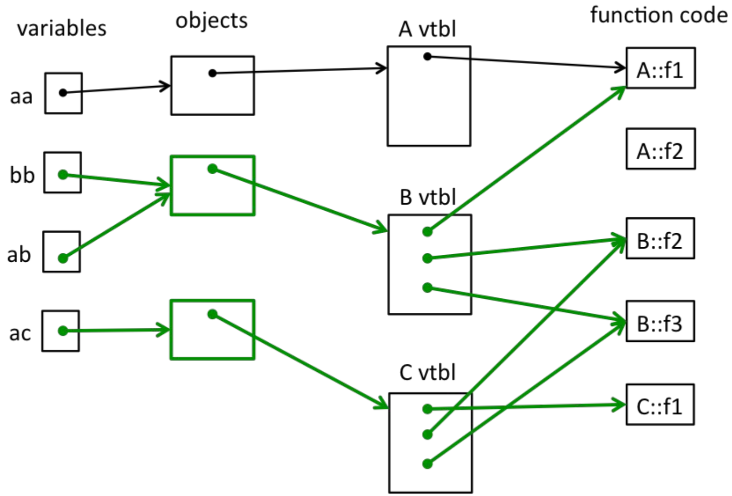
Exercise 1 Solution (output)

```
#include <iostream>
using namespace std;

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```



```
B* bb = new B();

bb->f3();
```

A	[✓ B]	C	D
B::f2 A::f1 B::f3	A::f2 A::f1 B::f3	A::f2 C::f1 B::f3	B::f2 C::f1 B::f3

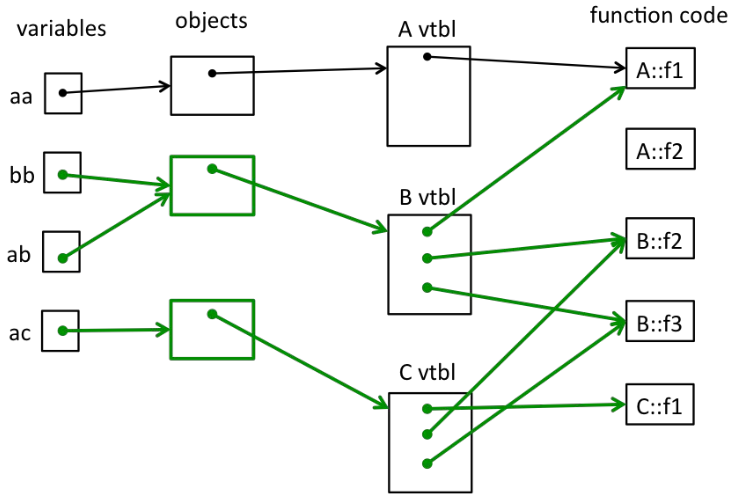
Exercise 1 Solution (output)

```
#include <iostream>
using namespace std;

class A {
 public:
  virtual void f1() { f2(); cout << "A::f1" << endl; }
  void f2() { cout << "A::f2" << endl; }
};

class B: public A {
 public:
  virtual void f3() { f1(); cout << "B::f3" << endl; }
  virtual void f2() { cout << "B::f2" << endl; }
};

class C: public B {
 public:
  void f1() { f2(); cout << "C::f1" << endl; }
};
```



```
A* ac = new C();

ac->f1();
```

A	B	C	D
B::f2 A::f1	A::f2 C::f1	A::f2 A::f1	B::f2 C::f1