Layers, Naming, and Sockets
CS 118
Computer Network Fundamentals
Peter Reiher

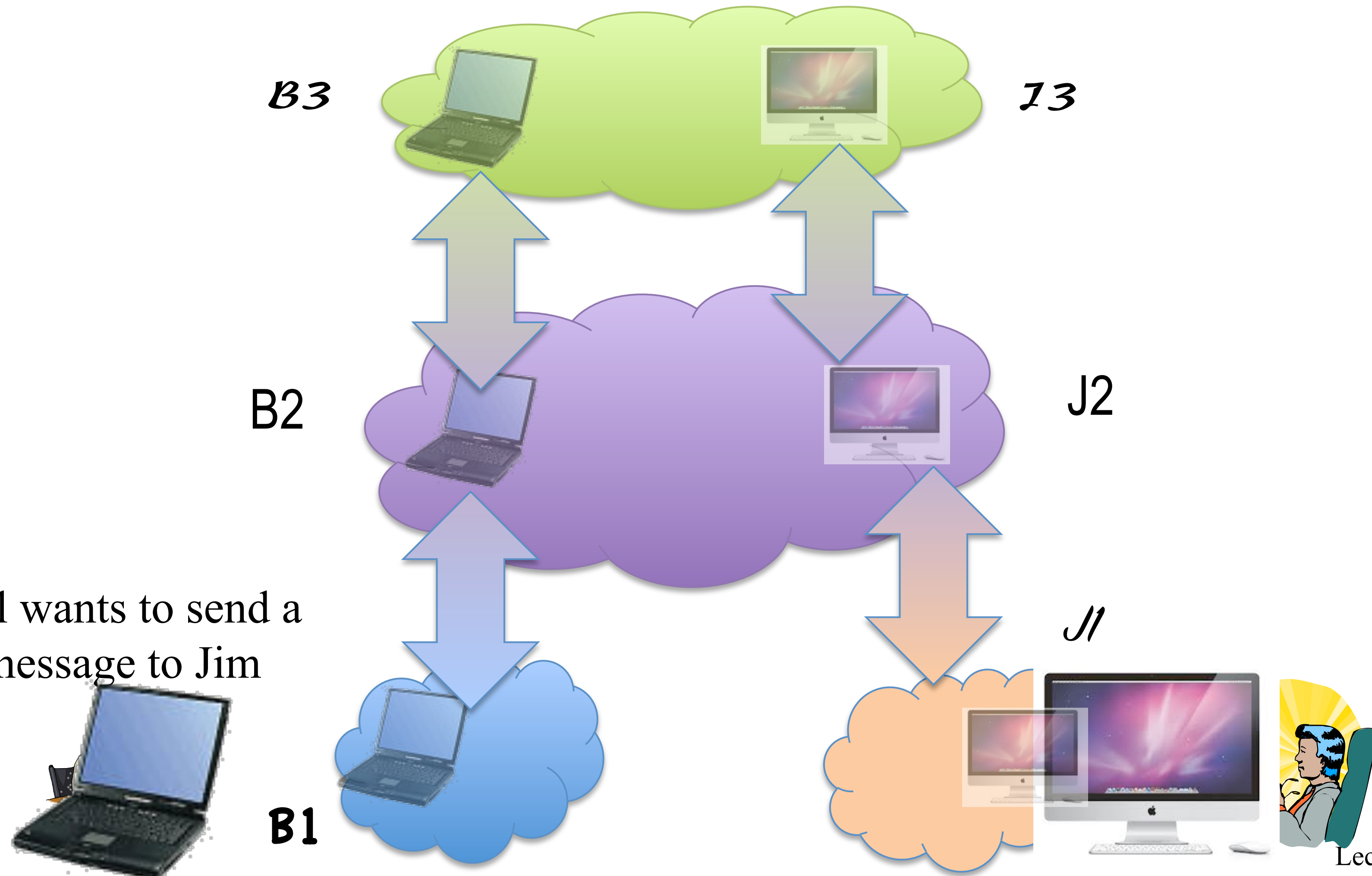
Outline

- What's a party?
- Inside names
- Outside names
- Linking the two
- Sockets as an example

Recall: definitions

- Communication
  - Methods for exchanging information between a fixed set of directly-connected ***parties*** using a single protocol
- Networking
  - Methods to enable communication between varying sets of indirectly connected ***parties*** that don't share a single protocol
- Protocol
  - A set of rules, agreed in advance [between the ***parties***], that enable communication

Names, layers, and translations



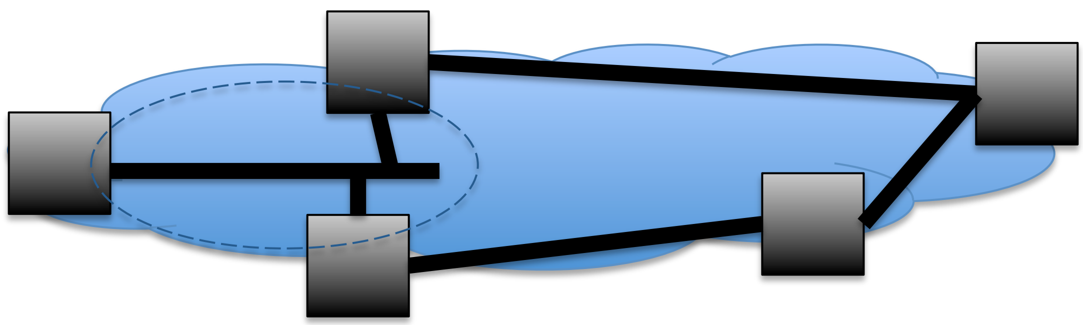
Bill wants to send a message to Jim

Today’s theme: #WTPA?

- *What* is a “party”?
- *Where* is that party?
  - Physically (in some physical place)
  - Logically (in some layer)

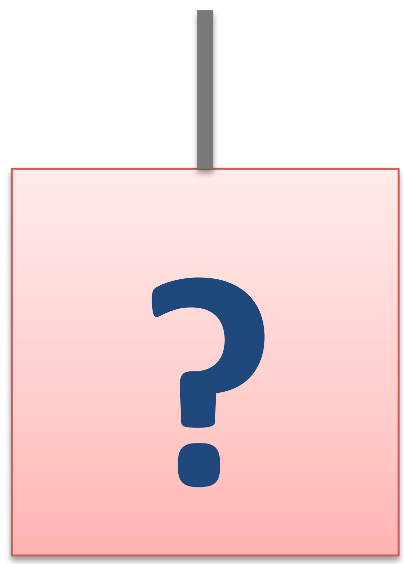
A network layer

- Nodes
  - Sources and sinks of information
- Links
  - Channels that connect two or more nodes

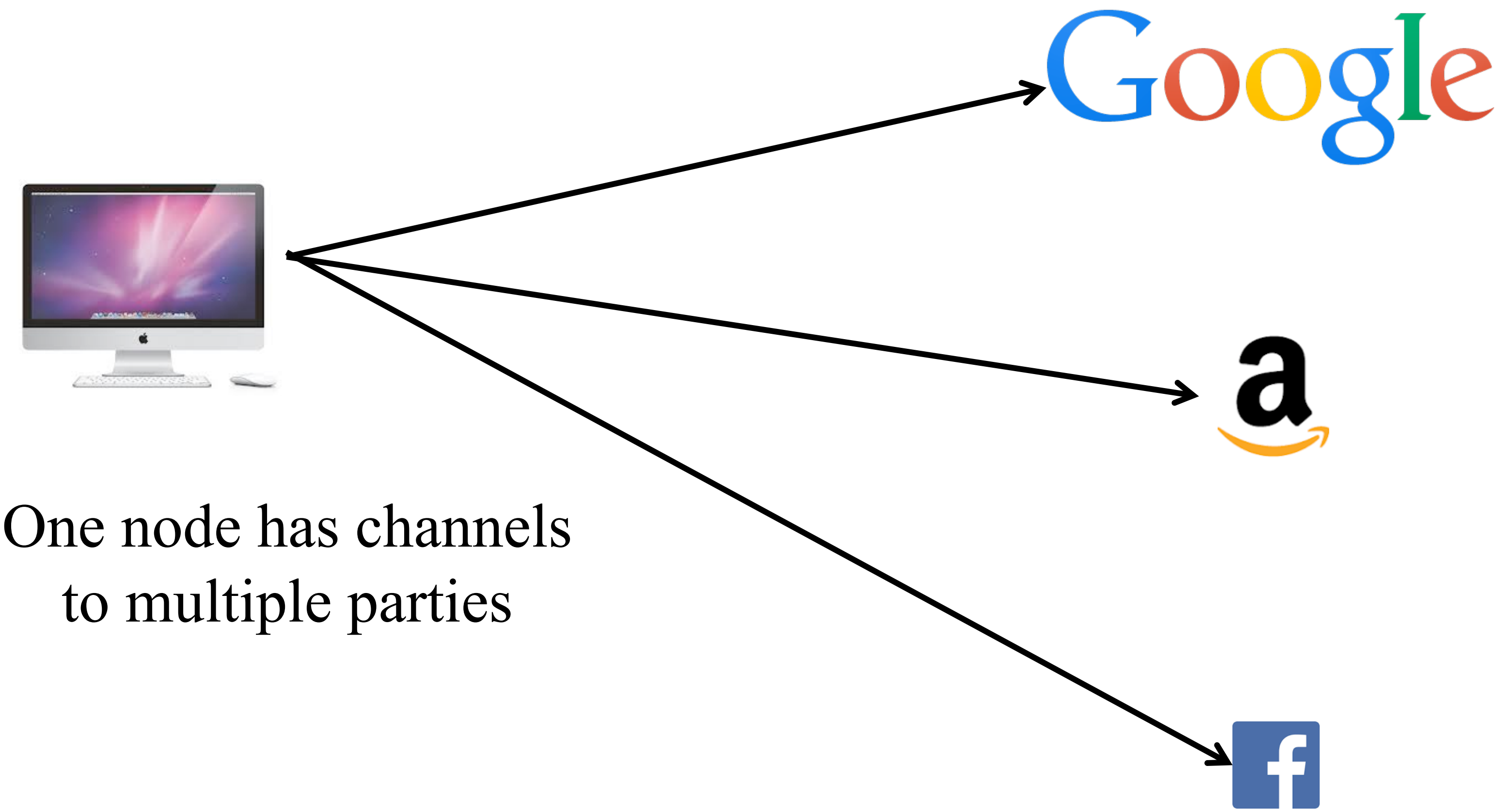


A closer look:

- What's inside a node?
  – What actually communicates to the outside?

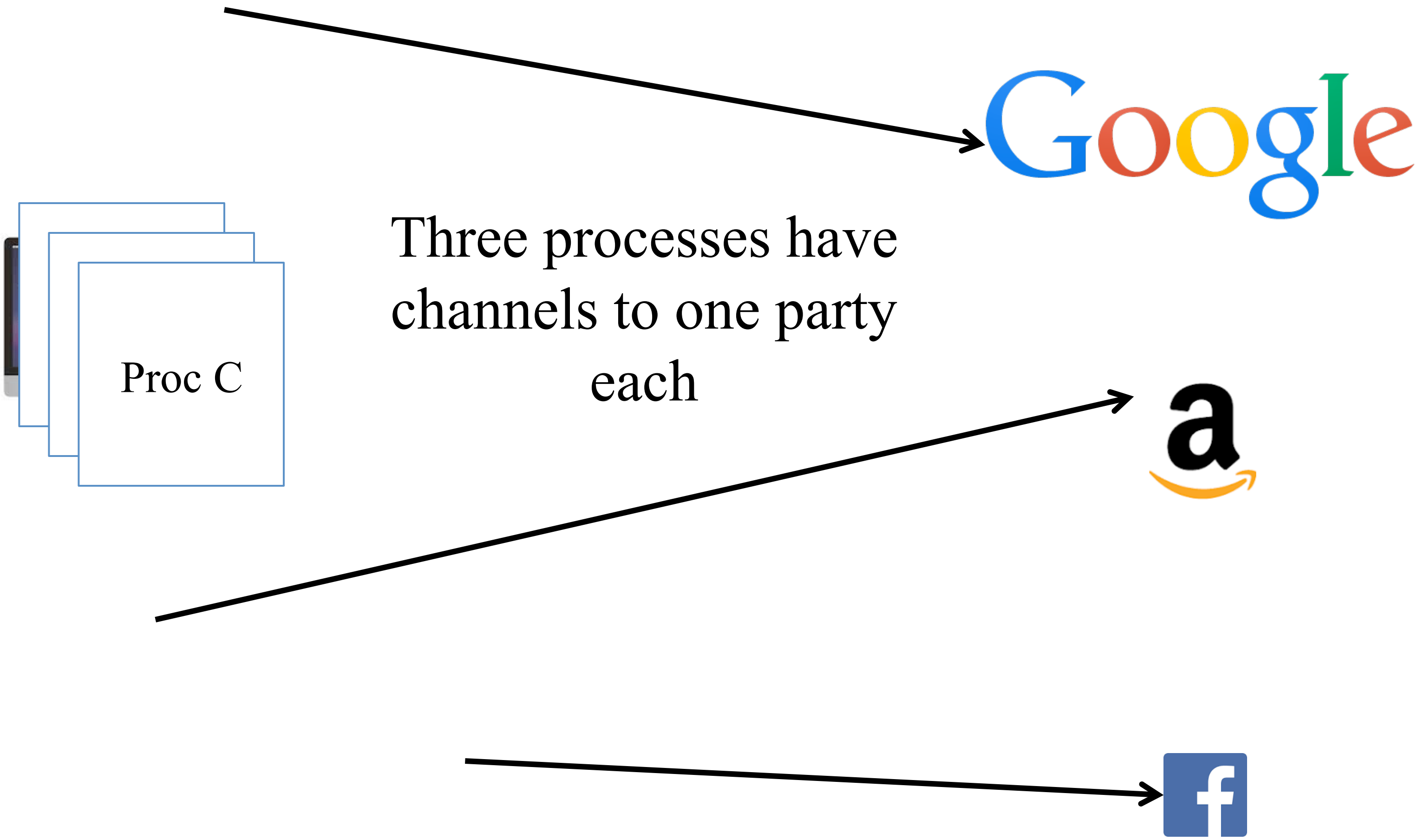


One View



One node has channels
to multiple parties

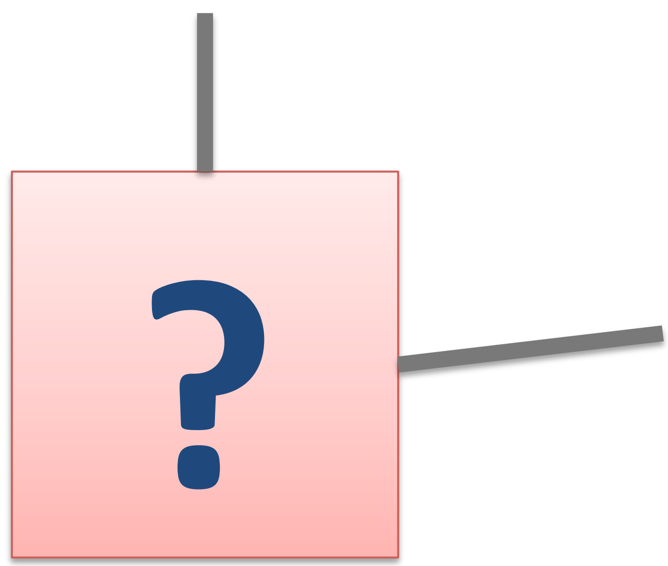
Another view



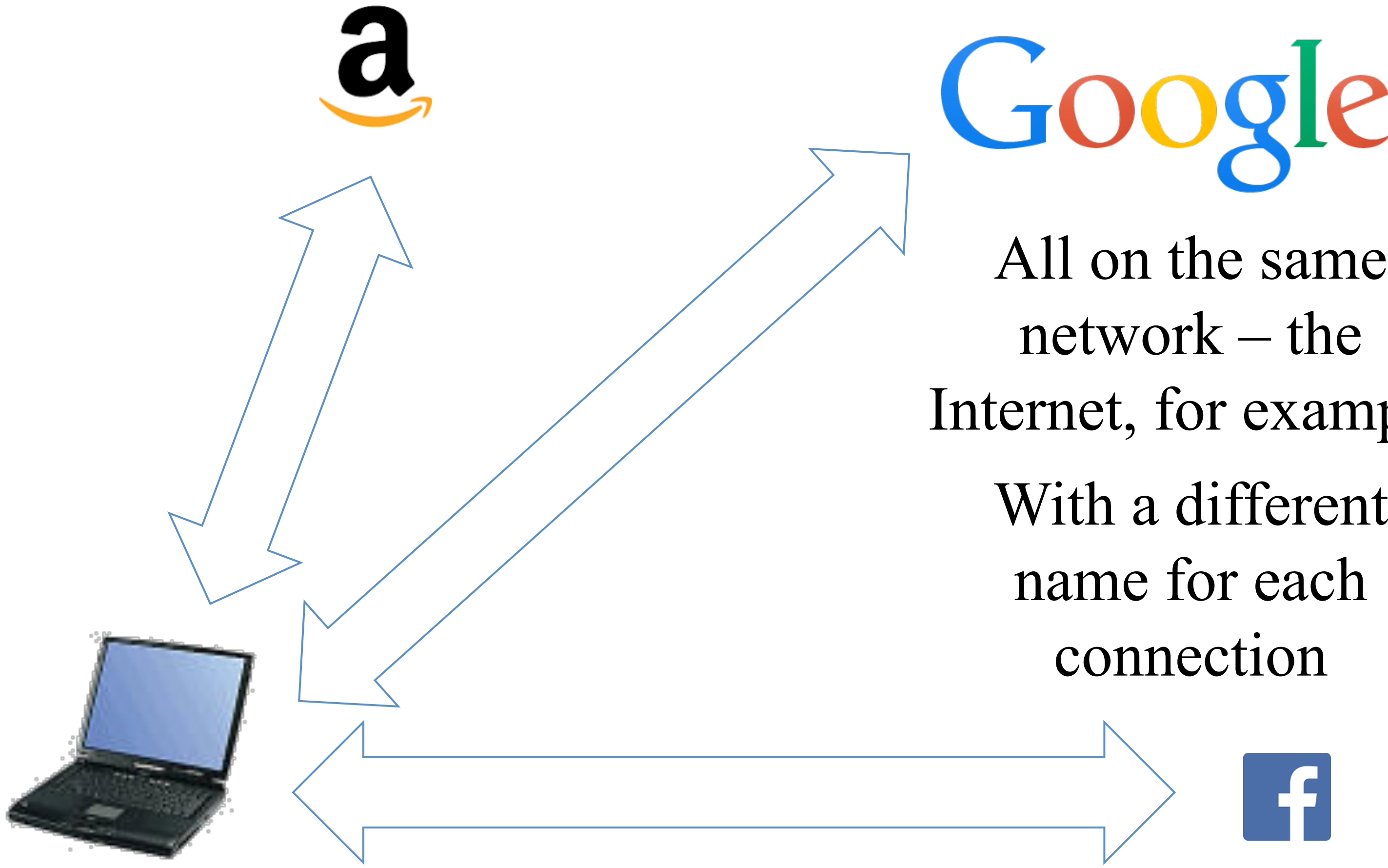
Three processes have channels to one party each

A closer look v2:

- What's inside a node when:
  - It has multiple channels on a single network (several names used external to the node)?



What Does That Mean?

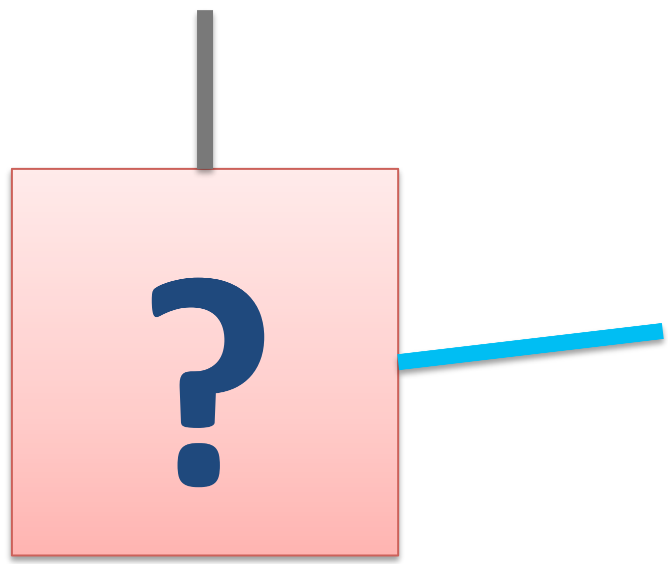


All on the same network – the Internet, for example

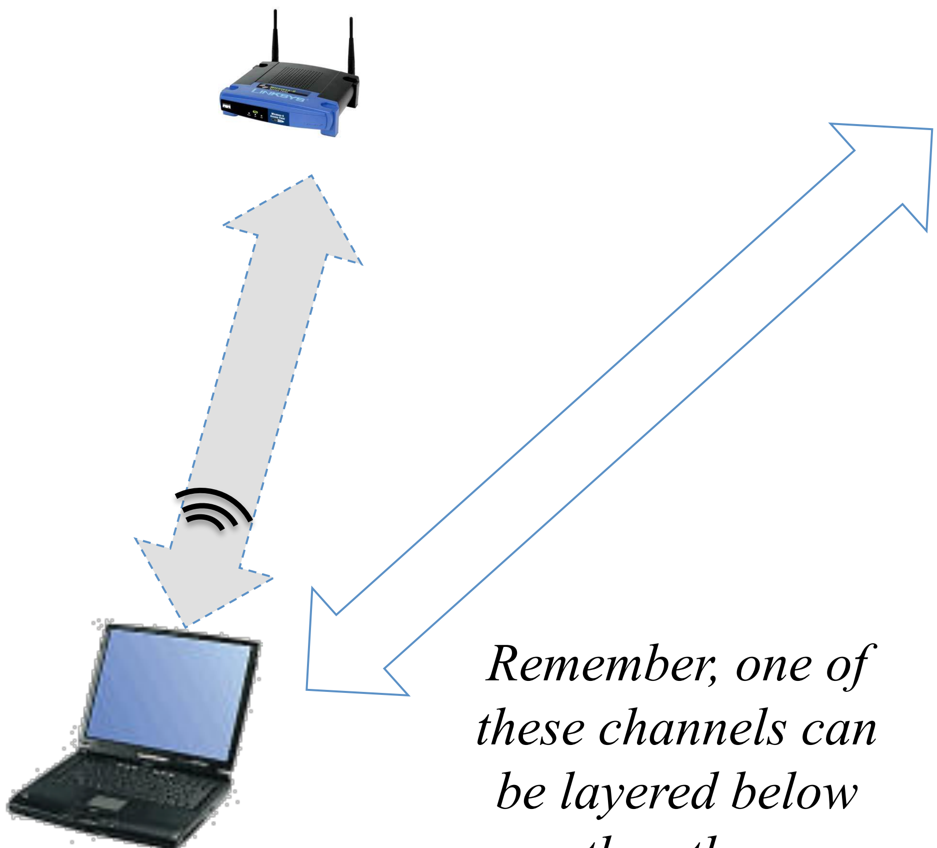
With a different name for each connection

A closer look v3:

- What's inside a node when:
  - It has channels on multiple networks (different kinds of external names)?



What does that mean?



Remember, one of these channels can be layered below the other

One is on the Internet, the other is on an 802.11 wireless network

One with an Internet name, the other with a wireless MAC address

Inside vs. outside names

- Another way of distinguishing names
  - That all "belong" to the same node
- Names depend on your viewpoint…

“Outside” names

- Giving a name to the source or destination on a network layer
  - Source address to enable N:1
  - Destination address to enable 1:N
  - Same address to enable bidirectional communications

“Outside” names

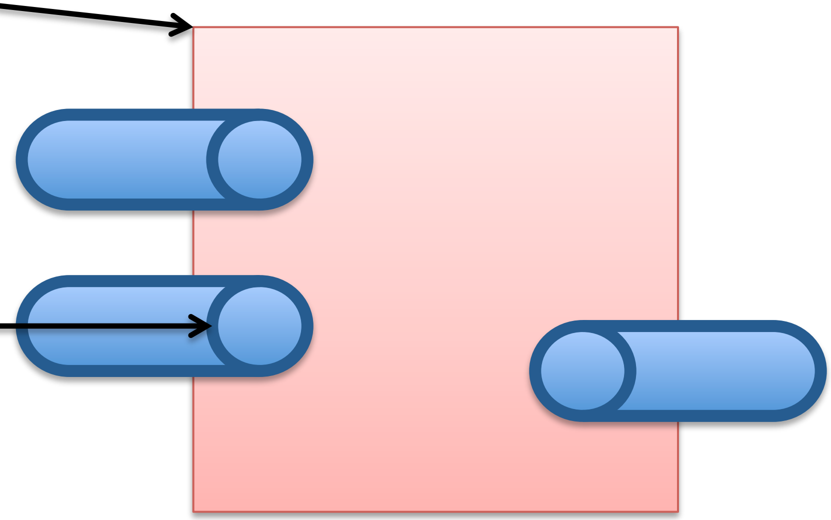
- Names of a party
  - Node names
  - Interface names

Node vs. interface

- Node
  - Where processes run
- Interface
  - Network attachment point

Node vs. interface

- Node
  - Source/sink of all network channels at a single place
- Interface
  - End of one network channel



Node names

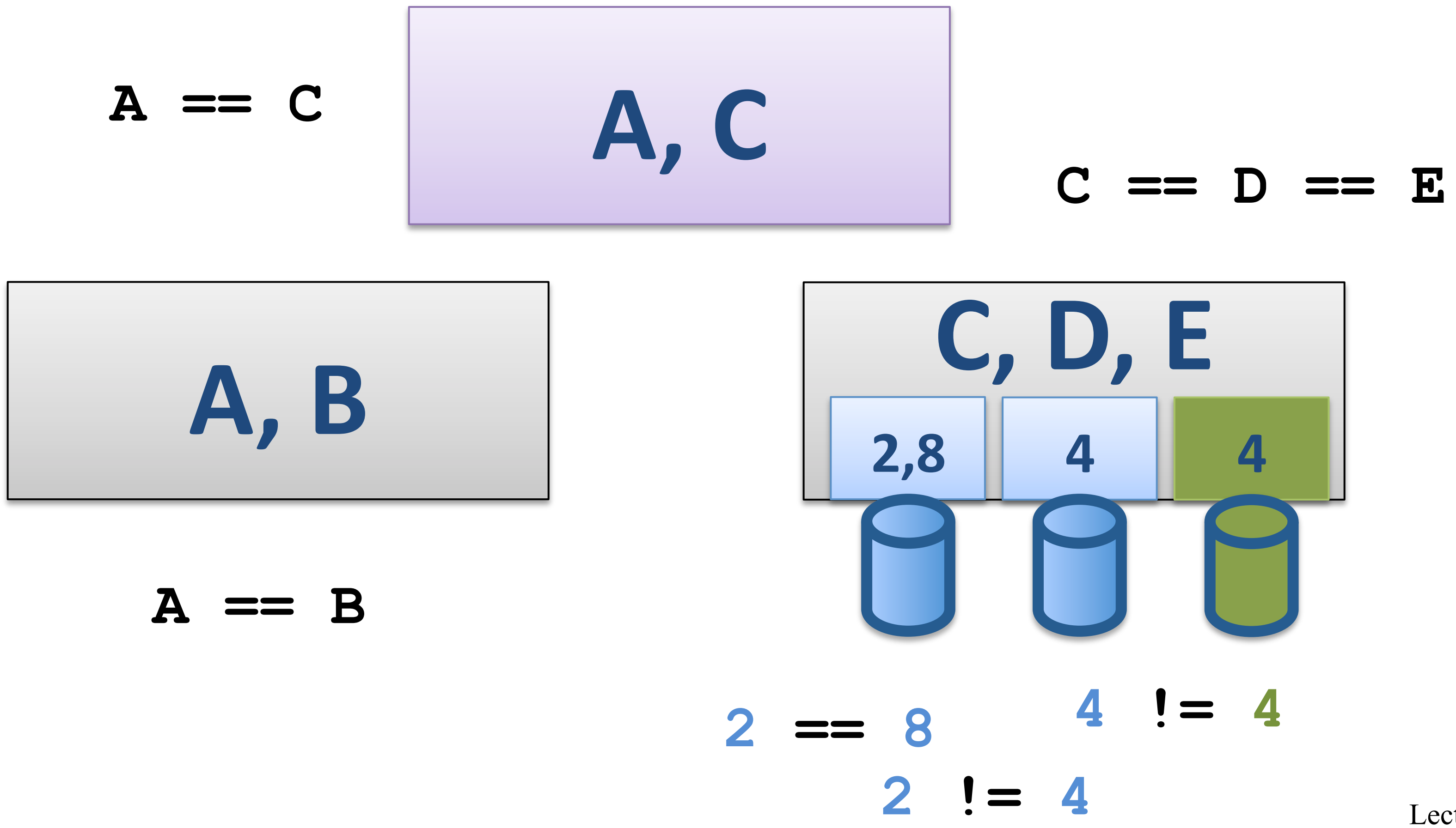
- Unique across
  - All nodes within a layer
- A node many have multiple
  - Node names
  - On the same or different layers
- Node names are equivalent
  - Within a node

Interface names

- Unique across
  - All endpoints within a layer
- A node may have multiple
  - Interfaces
- An interface may have multiple
  - Interface names
- Endpoint names are equivalent
  - Within an interface

Node name uniqueness

- Node vs. interface uniqueness

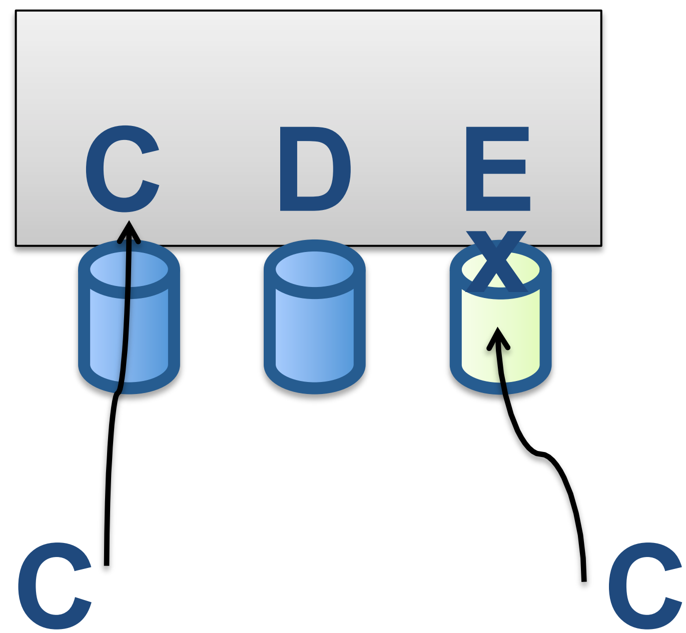


Strong vs. weak endpoint models

- We name interfaces AND nodes
  - What happens when we use those names?

Strong

- Names refer to the interface (channel end)
  - If a message arrives at a node from network A, it must be addressed to the endpoint address where that node attaches
  - All names belong (in effect) to the interface
  - Like the name of the doors of a house



Like at Downton Abbey

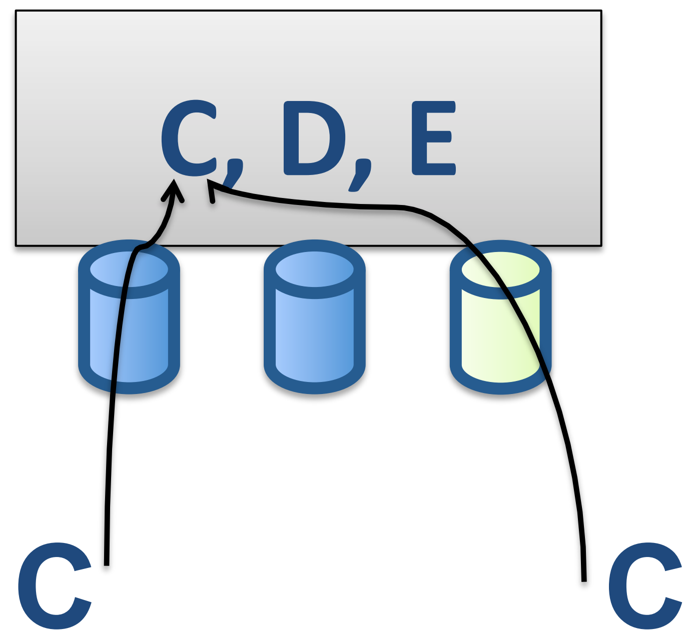
Guests to the front door

Tradesmen around the back

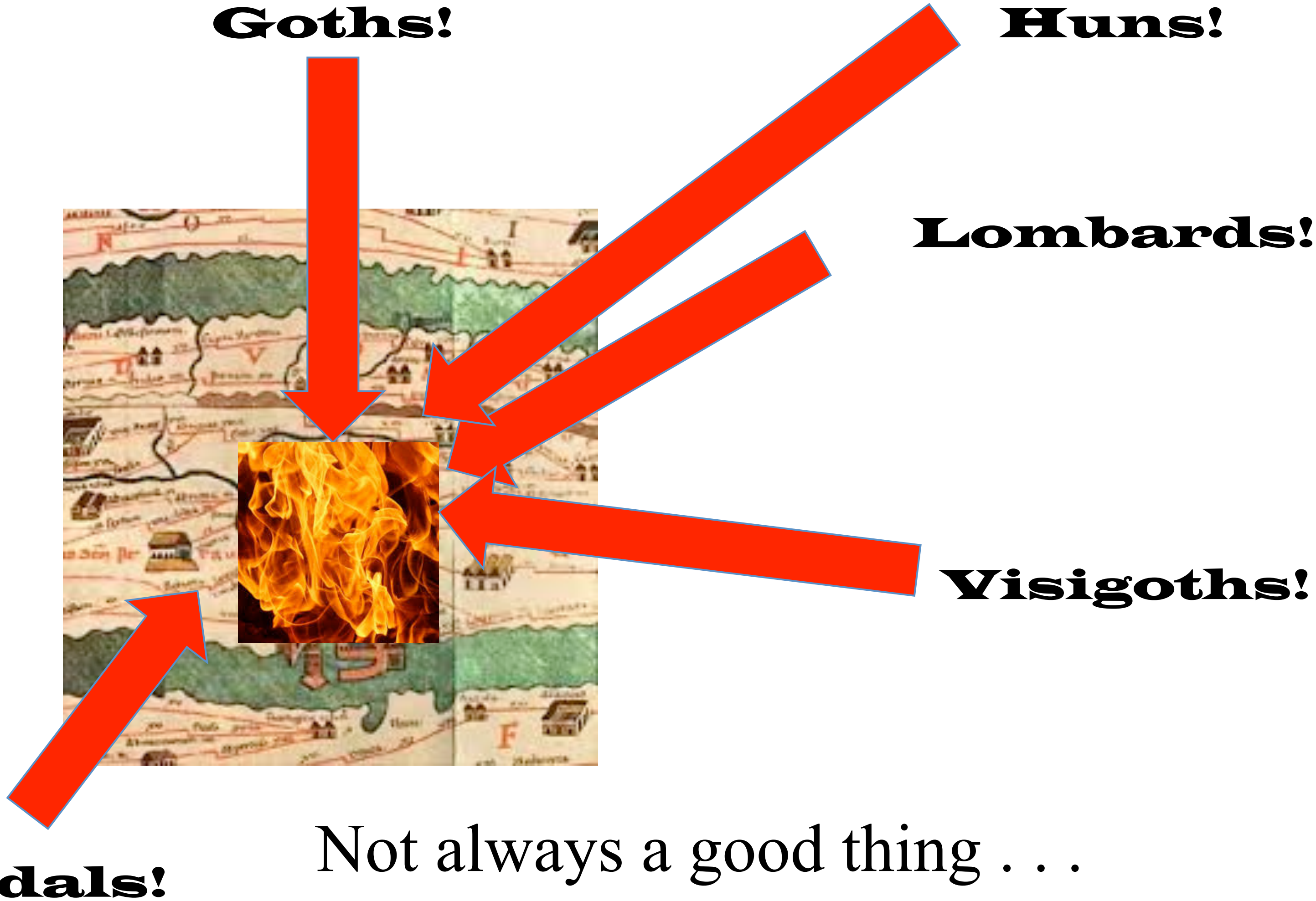
But they both end up in the house

Weak

- Names refer to the node
  - Even if assigned to the interface
  - If a message arrives at a node, it can be addressed to any endpoint address where that node attaches
  - All names belong (in effect) to the node
  - Like the names of a house



As in, All Roads Lead To Rome

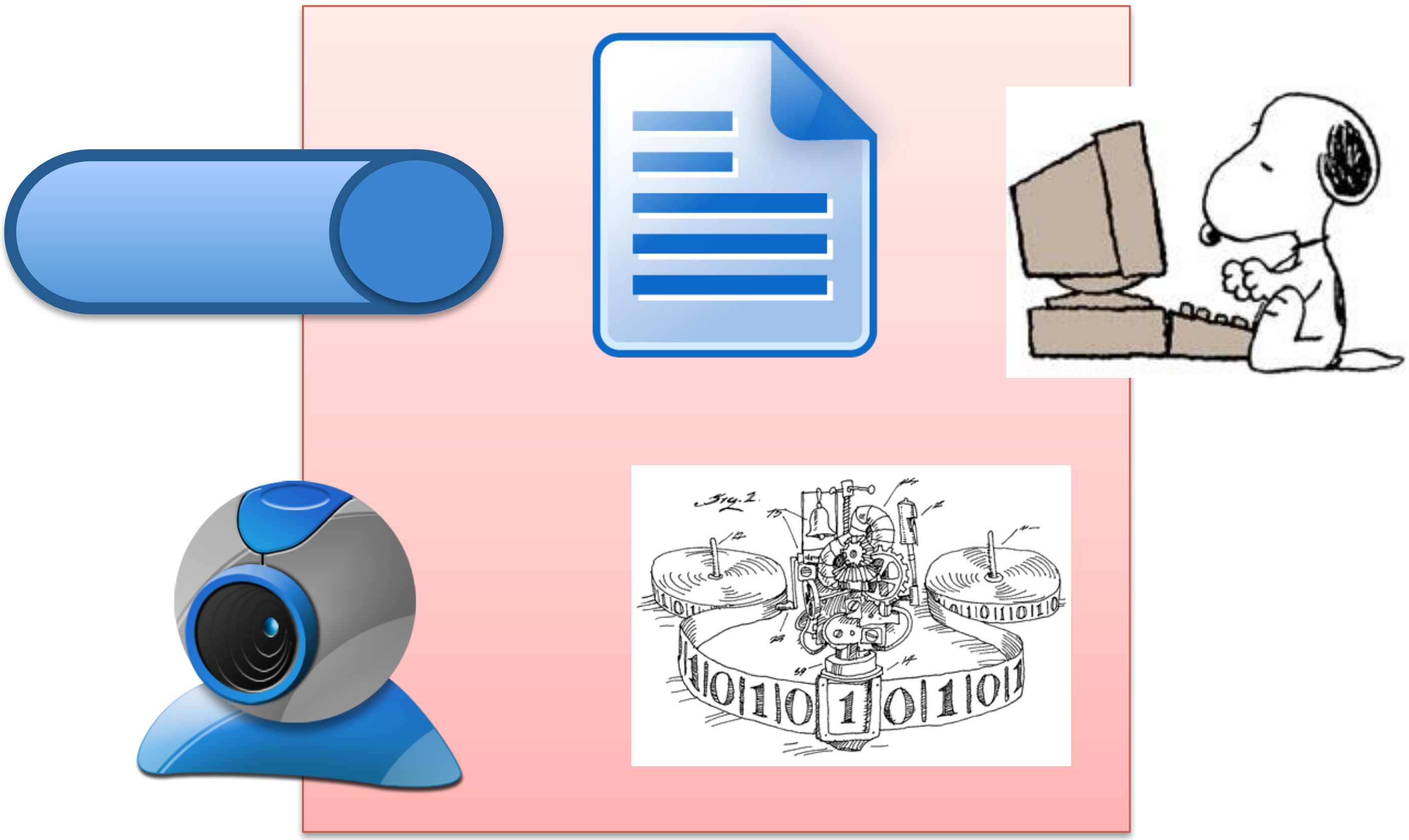


Not always a good thing . . .

Kinds of outside names

- Ethernet
  - A name for a channel endpoint for Ethernet messages (Ethernet layer)
- IP
  - A name for a channel endpoint for IP messages (IP layer)
- TCP, UDP
  - A name *within* an IP endpoint called a *port* (we'll get back to that shortly…)

A look inside the endpoint…



"Inside" names

- Names within a party
  - A communication source or sink from the view within the endpoint

Inside names…

- What do we need to refer to?
  - The data itself (objects)
  - The process that uses or creates it

Object related names

- File names (static data)
  - C:\Users\guest\Desktop\file.doc
  - /usr/include/stdio.h
- I/O names (infinite source/sink of data)
  - LPT1:, COM0:
  - /dev/pty0, /dev/ttya, /dev/eth3
  - Socket descriptor (complex data structure)

Process related names

- Process
  - 8842
- Thread
  - 223
- Other related names
  - User – 521, “reiher”
  - Group – 9111, “lasr”

OS Review

- Process
  - Smallest independent running program with its own memory space
  - Resources include program code, memory, and thread(s)
- Thread
  - Smallest independently-schedulable running program

Why we prefer process names to…

- Thread names
  - Single address space of a process ensures each process name is unique
  - Thread names might be unique only within their parent process space
- File, I/O, etc. names
  - In this class, comm. endpoints are TMs
  - A TM more closely maps to a process

Properties of inside names

- Syntax
  - Defined only for that node
- Value
  - Unique within the node

Meaningless as network identifiers

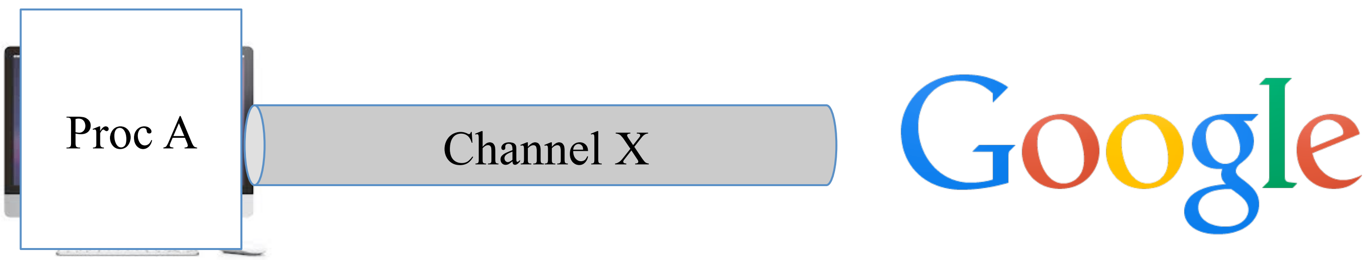
Job of an OS

- Coordinate resource sharing
  - Share memory, CPU capacity, devices, *channels*, etc.
- Provide abstractions
  - Of machines
    - To allow multiprocessing
  - Of other resources
    - Like the network layers

How do OSes abstract layer endpoints?

- Socket
  - Created by ARPAnet research (RFC33, 1970)
  - A communication endpoint from the view of the "user" (program)
  - Usually two-way
  - Basically: a socket is an inside name for outside communication…

What's that mean?



We need to tell the computer's operating system to connect

Process A

To channel X

A socket is A's inside name for the outside name (channel X)

Room for confusion . . .

- Unix-style systems also use sockets for machine-internal IPC
  - Where one process communicates to another
  - With no actual (or even virtual) networking involved
- Our concern is with network sockets

Inside and outside

- How do we link:
  inside names and outside names?

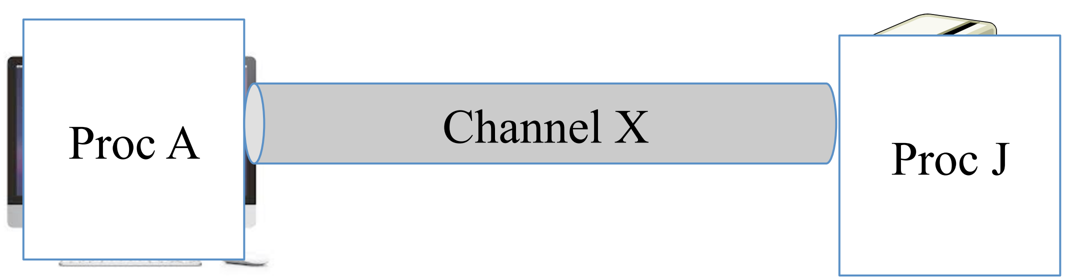
Linking the two

- Bind
  - Currently common OS convention
  - OS operation linking an internal I/O name to an external communication layer name

Two sides to a socket

- Server side

- Client side



A socket (either side)

- Ask the OS to create a placeholder
  - Attached to the process that creates it
  - A data structure that will link to the outside

```
if ((sockfd = socket(AF_INET, SOCK_STREAM,0)) == -1) {
  perror("Server: socket");
}
```

- Now I've got a socket
  - But I need to attach it to an inside name

Common kinds of sockets

- Datagram (e.g., Ethernet, IP, UDP, ATM AAL0)
  - Direct to the channel
  - Separate messages
  - Individually addressed
- Stream (e.g., TCP, ATM AAL2-5)
  - Two-party association ("connection")
  - Two steps:
    - Establish shared context with an address
    - Exchange data using that shared context
- Others are possible, but not common

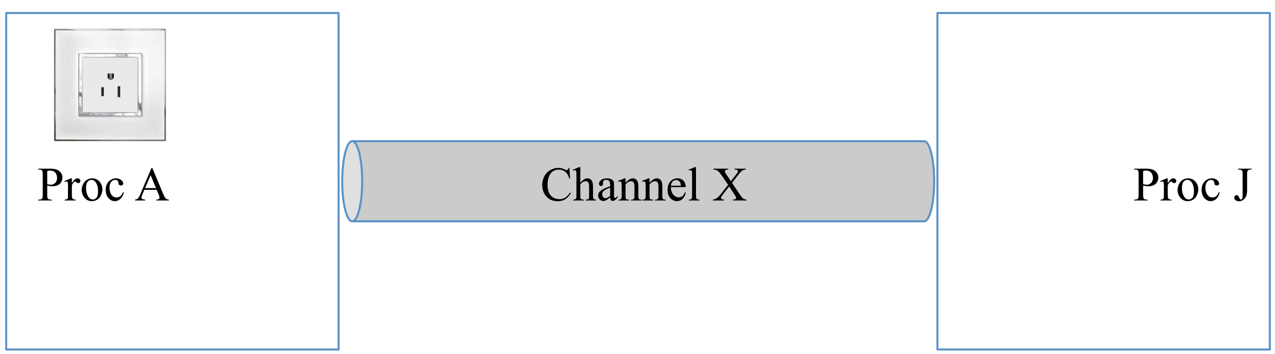
Bind

- Link a socket to an address on the server end
  - For TCP
    - Describes server end of the connection
  - For UDP
    - To limit messages you receive
    - To avoid source-addressing each message sent

```
if (bind(sockfd, (struct sockaddr *)&server, sockaddr_len) == -1) {
   close(sockfd);
   perror("Server: bind");
}
```

Server first steps

- Socket
  - Create a channel placeholder local to the process
- Bind
  - Link the channel placeholder to an external name



Stateless: receiving messages

- Recvfrom
  - Accept a message
  - Indicate <u>*who*</u> it is <u>*from*</u> (other end)

```
recvlen = recvfrom(sockfd, inbuf, MAXDATASIZE, 0,
          (struct sockaddr *)&client, &clientlen);
```

Stateless: sending messages

- Sendto
  - Send a message
  - Indicate *who* it is *to* (other end)

```
if (sendto(sockfd, outbuf, outbuflen, 0,
        (struct sockaddr *)&client, clientlen) < 0) {
    perror("Server: sendto failed");
}
```

Server side - connections

- Listen
- Accept

Listen

- Wait for incoming connection
  - Mark socket available for incoming requests
  - Prepare for someone to connect to the other end
  - Limit max waiting to be handled

```
if (listen(sockfd, MAX_CLIENTS) == -1) {
   perror("Server: listen");
   exit(1);
}
```

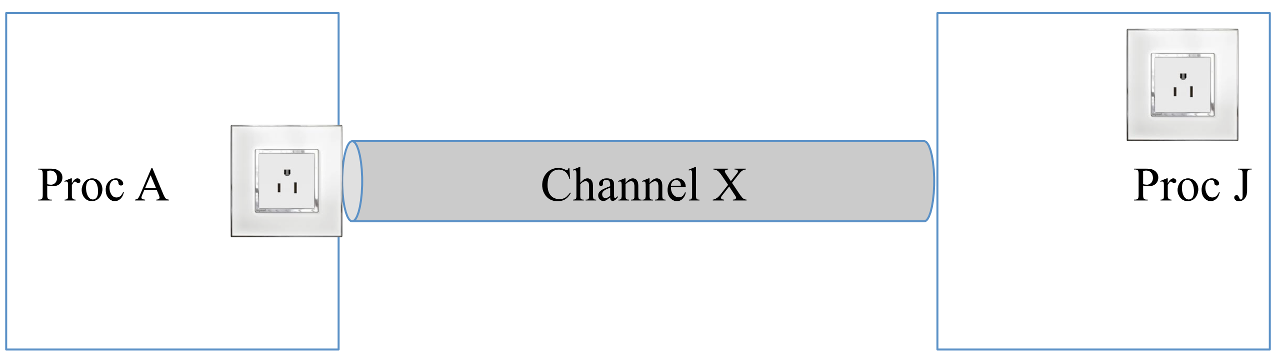
Accept

- Turns a socket into a socket pair
  - Socket pair defines a connection (both ends)
  - Now someone is connected to the other end
  - NB: in Unix, a socket and a socket pair are both described by the same data structure (a Unix socket)

```
new_fd = accept(sockfd,
  (struct sockaddr *)&client,
  (socklen_t *) &sockaddr_len);
```

Client side – connections

- Socket
  - Need something to connect to
- Connect
  - Connect socket to the channel



Connect

- Initiate a connection to a remote end
  - Indicate the remote end
  - Wait for the connection to be accepted

```
if ((connect(sockfd,
     (struct sockaddr *)&server,sizeof(server))) == -1) {
  perror("Client: connection error");
  exit(-1);
}
```

`sockaddr` and names

- What is the sockaddr?
- A data structure containing an external name
  - A name the client can use to specify which server socket to connect to
- In practice, an IP address and a port
  - Which is, remember, the type of name used by TCP and UDP

Client and server data exchange

- Send
- Recv

Send

- Write data on the connected socket
  - Same as sendto with NULL remote endpoint
  - Can be wrapped with a `write` call for simpler use

```
if (send(sendsock,sendbuf,strlen(sendbuf), 0) == -1){
   perror("Client: send");
   exit(1);
 }
```

Recv

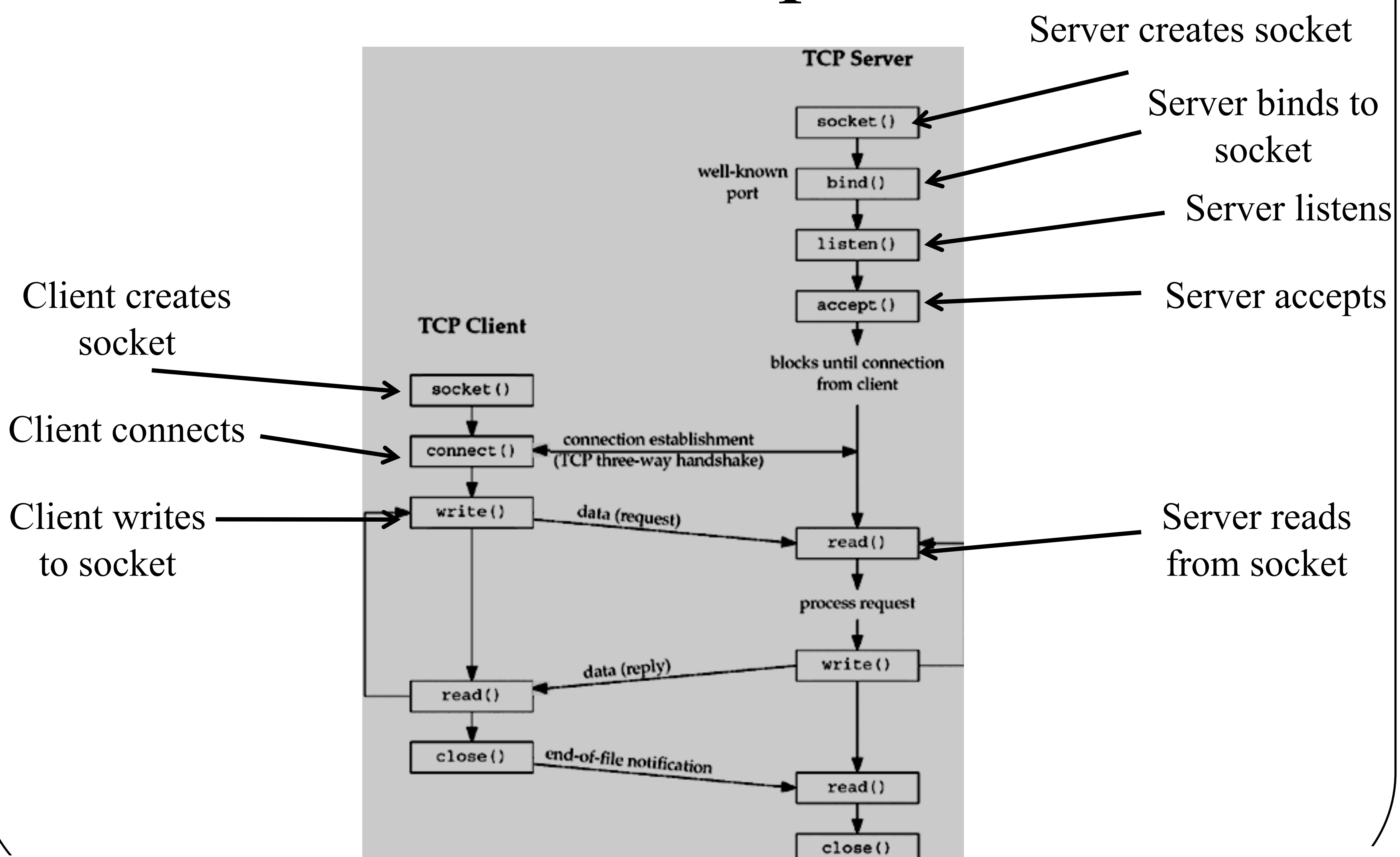
- Read data from a connected socket
  – Same as recvfrom with NULL remote endpoint
  – Can be wrapped with a `read` call for simpler use

```
if ((num = recv(recvsock, &buf, MAXLEN, 0)) == -1) {
     perror("Server: recv failed");
     exit(1);
}
```

Putting it all together

- How do you arrange a client/server connection with sockets?
- Server creates a local `socket` and `binds` to it
- Client creates a local `socket` and `connects` it to the server's external name
- Server `listens` on the socket and `accepts` incoming messages
- Client `writes`, server `reads`

For example,



Server creates socket
Server binds to socket
Server listens
Server accepts
Server reads from socket
Client creates socket
Client connects
Client writes to socket
TCP Server
socket()
bind()
well-known port
listen()
accept()
blocks until connection from client
TCP Client
socket()
connect()
connection establishment (TCP three-way handshake)
write()
data (request)
read()
process request
write()
data (reply)
read()
close()
end-of-file notification
read()
close()

Issues

- Messages without bind
- A horse with no name
- Socket type and boundaries

No bind, no problem

- Stateless (connectionless) messages
  - Bind indicates local end
  - What if you omit bind?
  - The OS figures out where the message should go and adds the source address itself

Automatic source addressing

- What do you already indicate?
  - Destination address (required in sendto)
  - Includes “address family”
    - Unix-speak for layer name
    - Only one layer of each name!
  - Includes destination address
    - Use that with an internal (route) table to pick an outgoing interface
    - Set the source address to the outgoing interface

What about ports?

- Recall:
  - Port distinguishes different TCP or UDP layer endpoints within a IP layer
- How do you know the one to send to?
  - Someone tells you!
  - From a published list
  - Because you're replying to a message (or within a connection) already know

What's your port number?

- Messages
  - The one you know to send to
  - The one you got a message from (to reply)
- Connections
  - The one a server LISTENs on
  - The one a client CONNECTs to

Port numbers

- Ports are local to the pair communicating
  - Identifies the socket (thus the process) on each end
- *Sometimes* ports have common meaning
  - At “first contact”, they help you pick who you’re talking with (i.e., client-side)
  - That’s why they’re registered by IANA

Two meanings of ports

- During first contact – expected process
  - E.g., web server (80), secure server (443), email server (110), etc.
- After that, *just* an endpoint identifier
  - At the TCP/UDP layer

Port meaning

- By common convention (***assumption***)
  - Groups:
    - System ports (80, 110, 53)
    - User ports (8080, etc.)
    - Dynamic ports (unassigned!)
  - Assigned to "services" (TM expecting messages)

- By other coordination
  - Because you and the other endpoint agree
  - Port can mean anything you (and they) want

Having no name

- Bind with no name?
  - Technically, you cannot
  - Bind to “0” = ANY (i.e., “don’t care”)
  - Works for IP address, TCP/UDP port
- What happens when you need a name?
  - If you picked ANY, the OS assigns you one
  - Address = based on path, from ones you “own”
  - Port = pick one not in use

Socket types and boundaries

- Sometimes send/recv boundaries match
  - E.g., the channel preserves the boundaries
  - Sending messages
  - Sending data over a connection with markers
- Sometimes, not so much
  - E.g., TCP!
  - If you send 100 bytes, that might go in one TCP message, two, three, etc.
  - When the other side recvs, you don't know what data is ready

Summary

- Naming is more than just for networking
  - Names inside the machine
  - Binding between inside and outside names
- Names are linked in a set of steps
  - We used Unix as an example
- Names also set expectations
  - E.g., port number implies TM type (“service”)