

Computer networks, pointers and strings

How messages travel between computers
How C programs refer to data in memory

Prof. Dr. Mehmet Göktürk · Gebze Technical University

Two questions for this week

How can many computers share a communication system and reach the right destination?

How can a function use an address to inspect or change data owned by its caller?

Networks and pointers solve different problems. Both require precise rules for interpreting addresses.

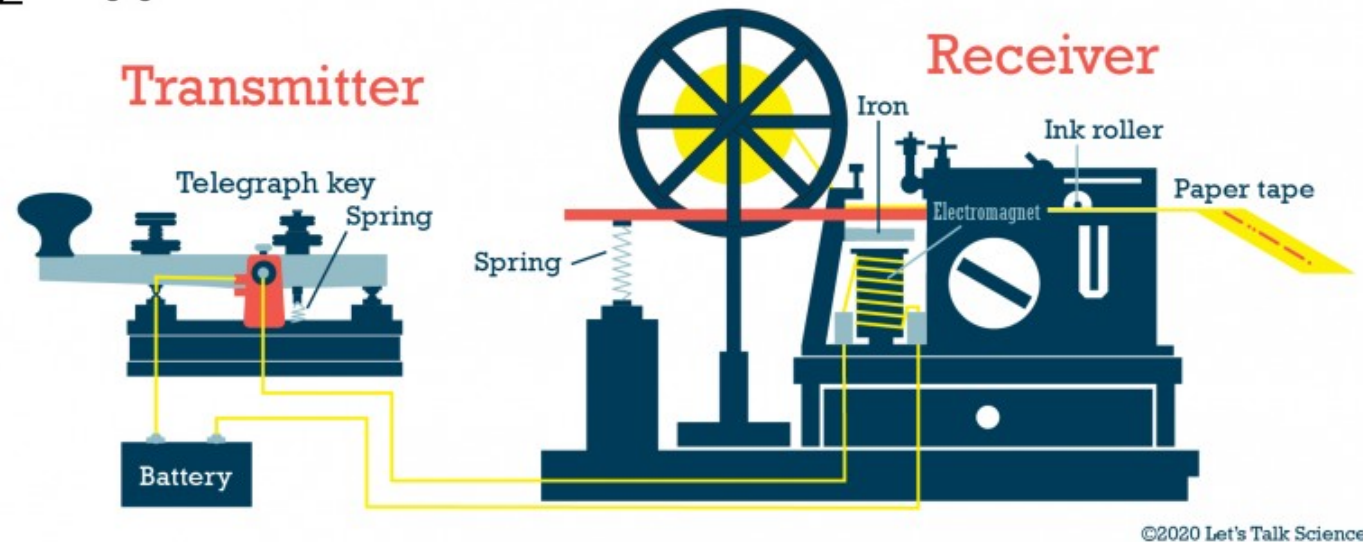
Messages before electronic networks



A message needs a representation, a carrier and a recipient.

Telegraph: agreed symbols over a wire

A ● -	J ● - - -	S ● ● ●
B - ● ● ●	K - ● -	T -
C - - ● ●	L ● - ● ●	U ● ● -
D - ● ●	M - -	V ● ● ● -
E ●	N - ●	W ● - -
F ● ● - ●	O - - -	X - ● ● -
G - - ●	P ● - - ●	Y - ● - -
H ● ● ● ●	Q - - ● -	Z - - ● ●
I ● ●	R ● - ●	



The sender and receiver must agree on how signals represent characters.

Keyboards and coded characters

LETTERS		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	CARRIAGE RETURN	LINE FEED	LETTERS	FIGURES	SPACE	ALL SPACE POST IN USE
FIGURES		-	?	:	3	%	@	£	8	9	0	1	4	'	5	7	=	2	/	6	+												
CODE ELEMENTS	1	●	●		●	●	●				●	●						●		●		●		●	●	●	●			●	●		
	2	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	3			●	●		●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	4		●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	5	●					●	●	●	●			●	●	●	●	●	●	●				●	●	●	●	●	●	●	●	●	●	●

● INDICATES A MARK ELEMENT (A HOLE PUNCHED IN THE TAPE)
○ INDICATES POSITION OF A SPROCKET HOLE IN THE TAPE

The International Telegraph Alphabet



Hughes telegraph, an early (1855) teleprinter built by Siemens and Halske. The centrifugal governor to achieve synchronicity with the other end can be seen.

Hughes printing telegraph and a separate teleprinter code chart: related developments, different systems.

Teleprinters: type here, print elsewhere

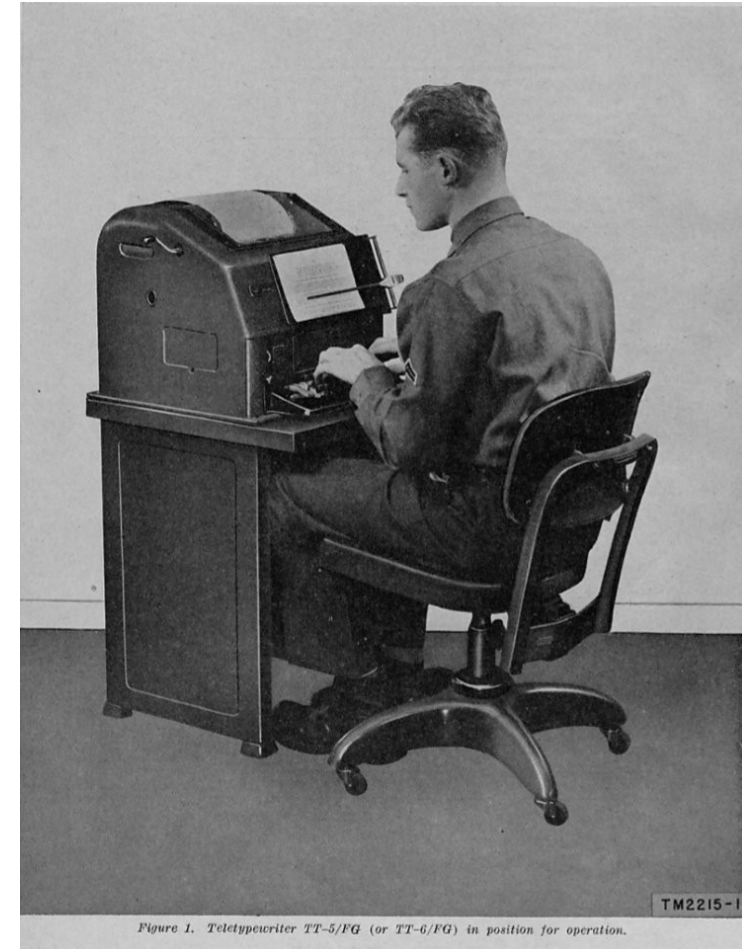


Figure 1. Teletypewriter TT-5/FG (or TT-6/FG) in position for operation.

Encoding and decoding move into machines. People work with readable text.

Telex: choosing a distant terminal

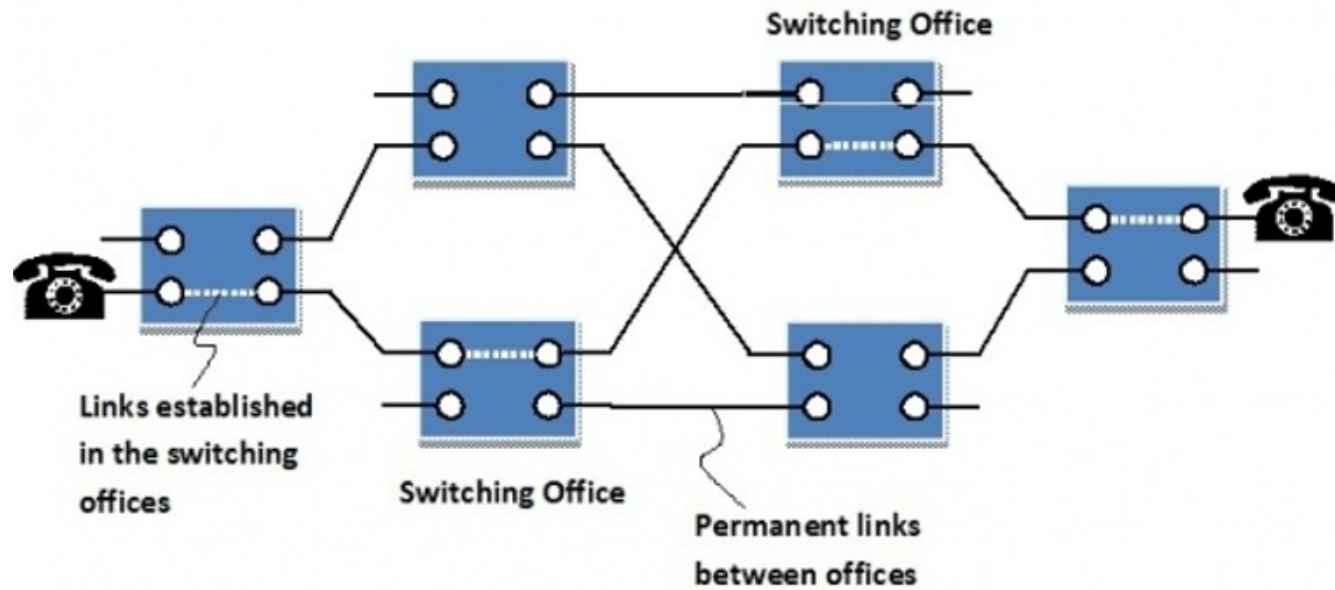


A direct wire for every pair does not scale.

Switching lets users share infrastructure.

A switched service connects a sender to the intended receiving terminal.

Circuit switching

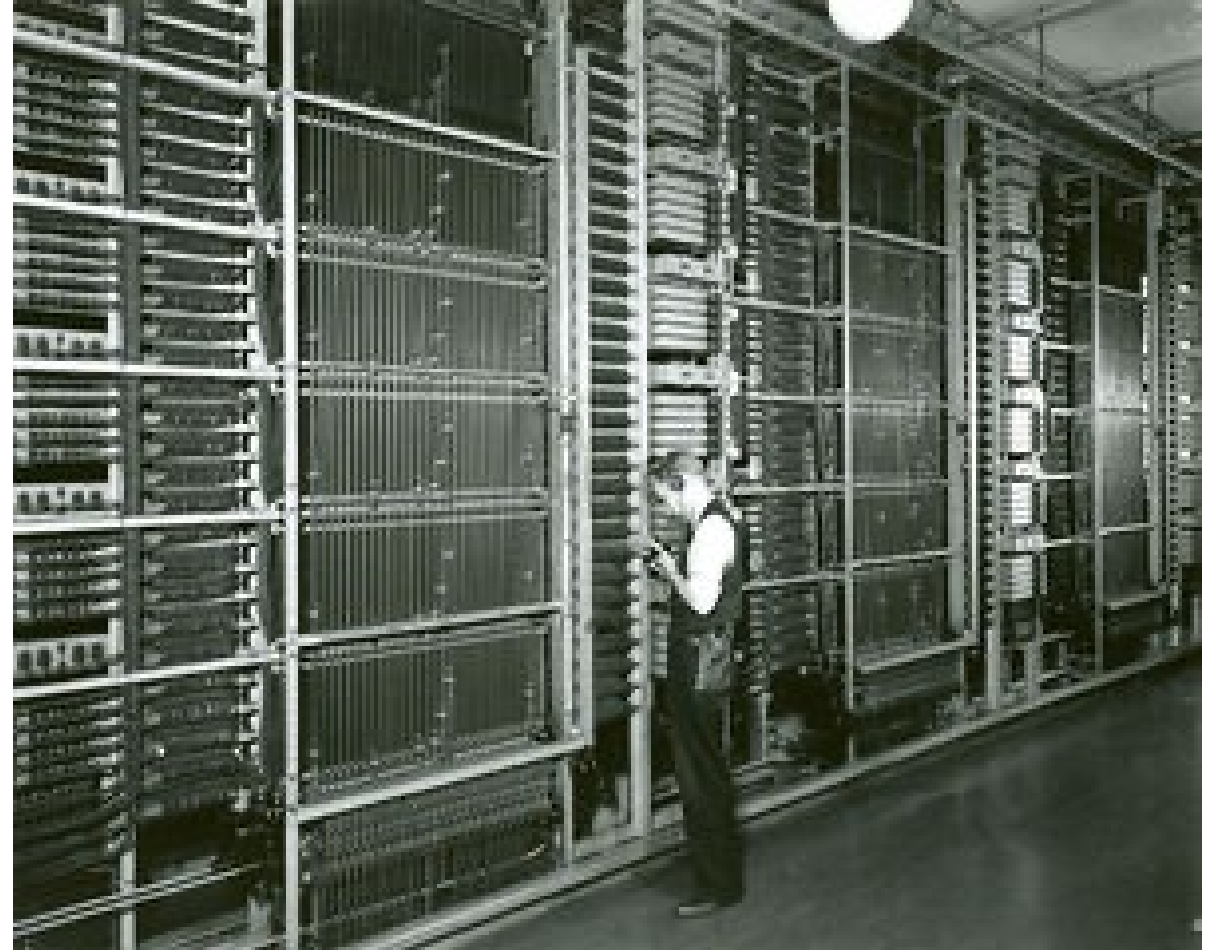


Setup establishes a circuit.

Reserved capacity may be idle between bursts.

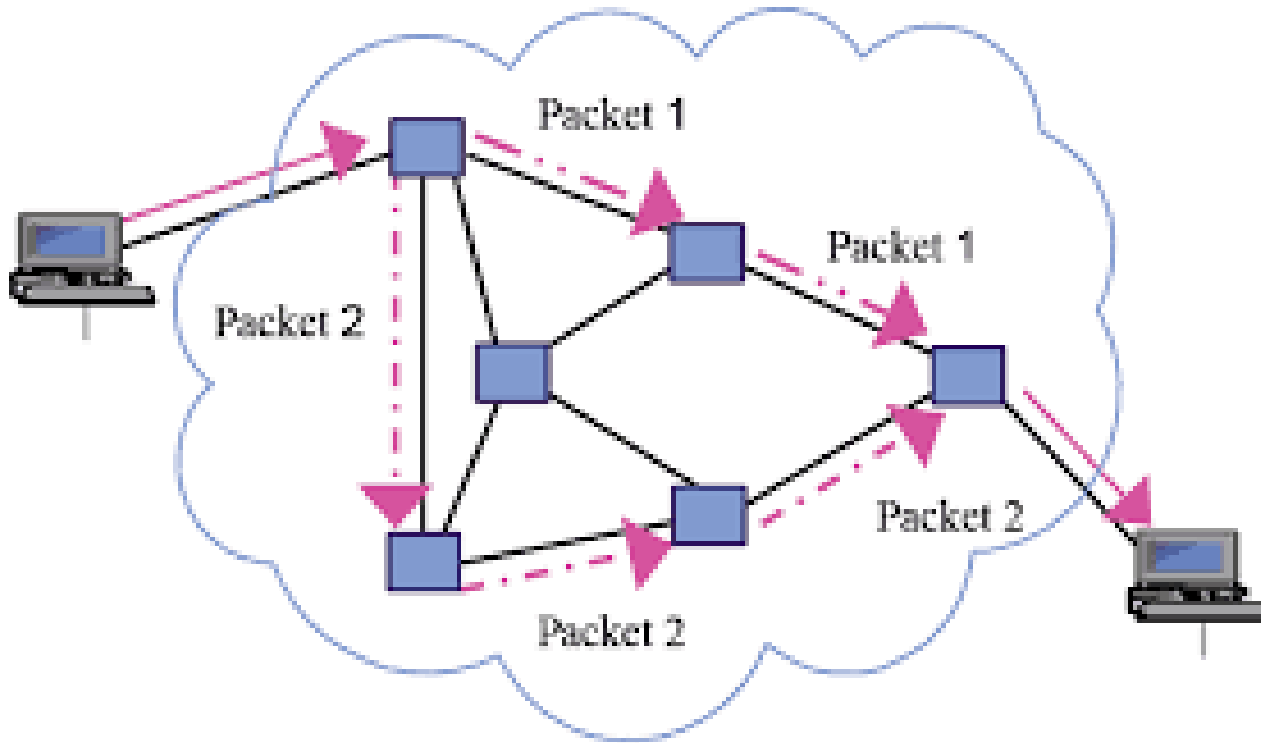
Resources along a path are reserved for the duration of a connection.

Physical switching equipment



Crossbar equipment connects selected input and output circuits.

Packet switching



A packet carries a header and payload.

Packets can wait in queues or be dropped.

Packets share links. Forwarding decisions move each packet toward its destination.

Store and forward: the postal analogy



Receive, examine the destination, then forward toward the next stop.

Final delivery and the destination



A router selects the next hop.

The receiving application interprets the message.

An intermediate sorting office and a final recipient have different responsibilities.

Circuit and packet switching

Question	Circuit switching	Packet switching
What is shared?	Infrastructure between circuits	Link time between packets
Before sending	Establish a circuit	Forward packets using headers
During a quiet period	Reserved capacity can sit idle	Other packets can use the link
Under heavy load	A new circuit may be refused	Queues grow and packets may drop

A packet network does not promise that every packet will arrive.

Bandwidth describes a transfer rate. Latency describes how long something takes to arrive.

Sending 1,000 bytes over a 1 Mbit/s link takes 8 ms to place those bits on the link.

Propagation, processing and queueing add delay. A faster link does not remove every delay.

Networks connect participating devices



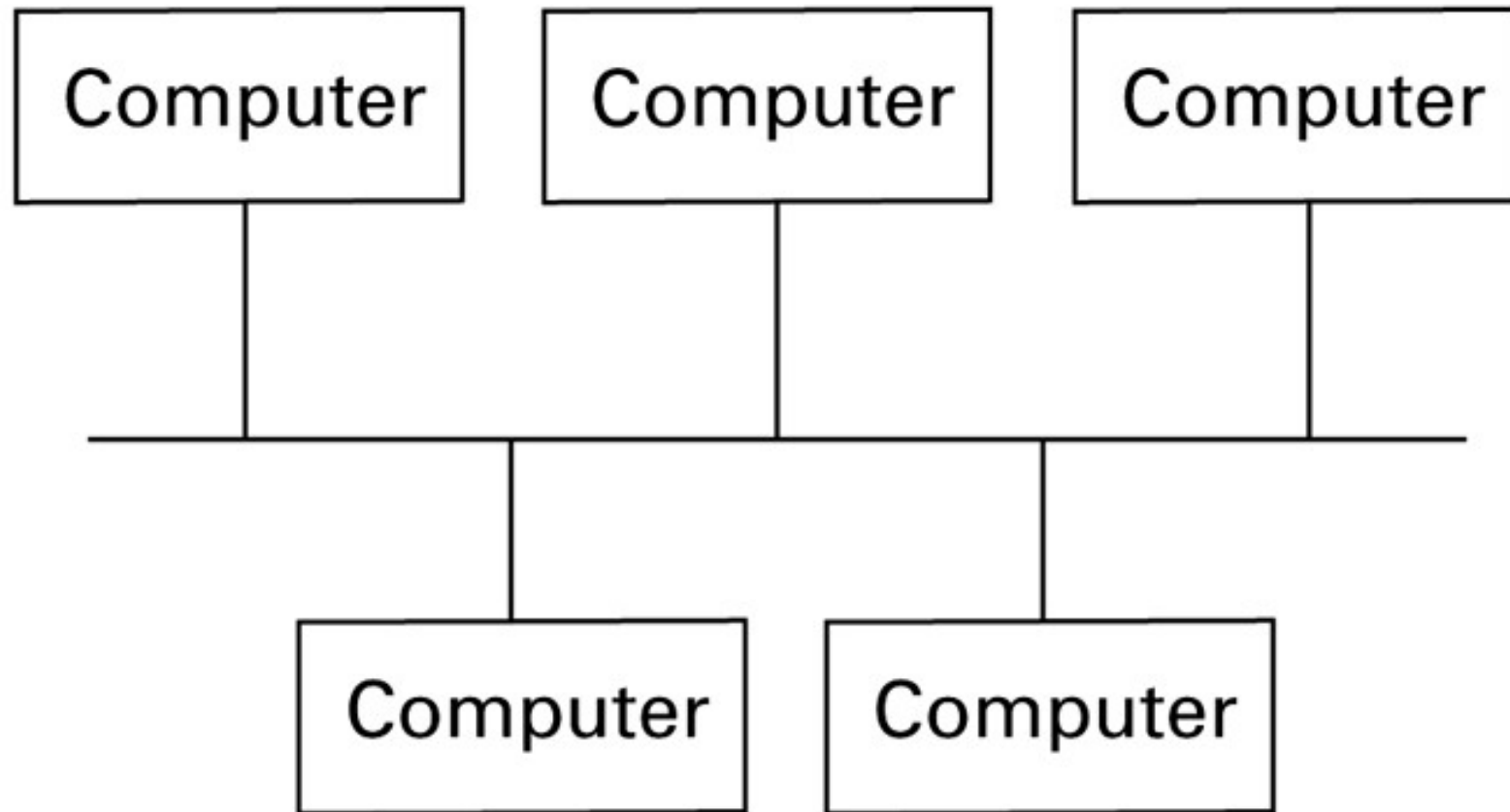
A network has endpoints, links and rules for communication. People use the services built on it.

Network scope

Scope	Example	What the label describes
PAN	Nearby personal devices	A small personal area
LAN	A room, building or campus segment	Local connectivity
WAN	Links between distant sites	A wide geographic area

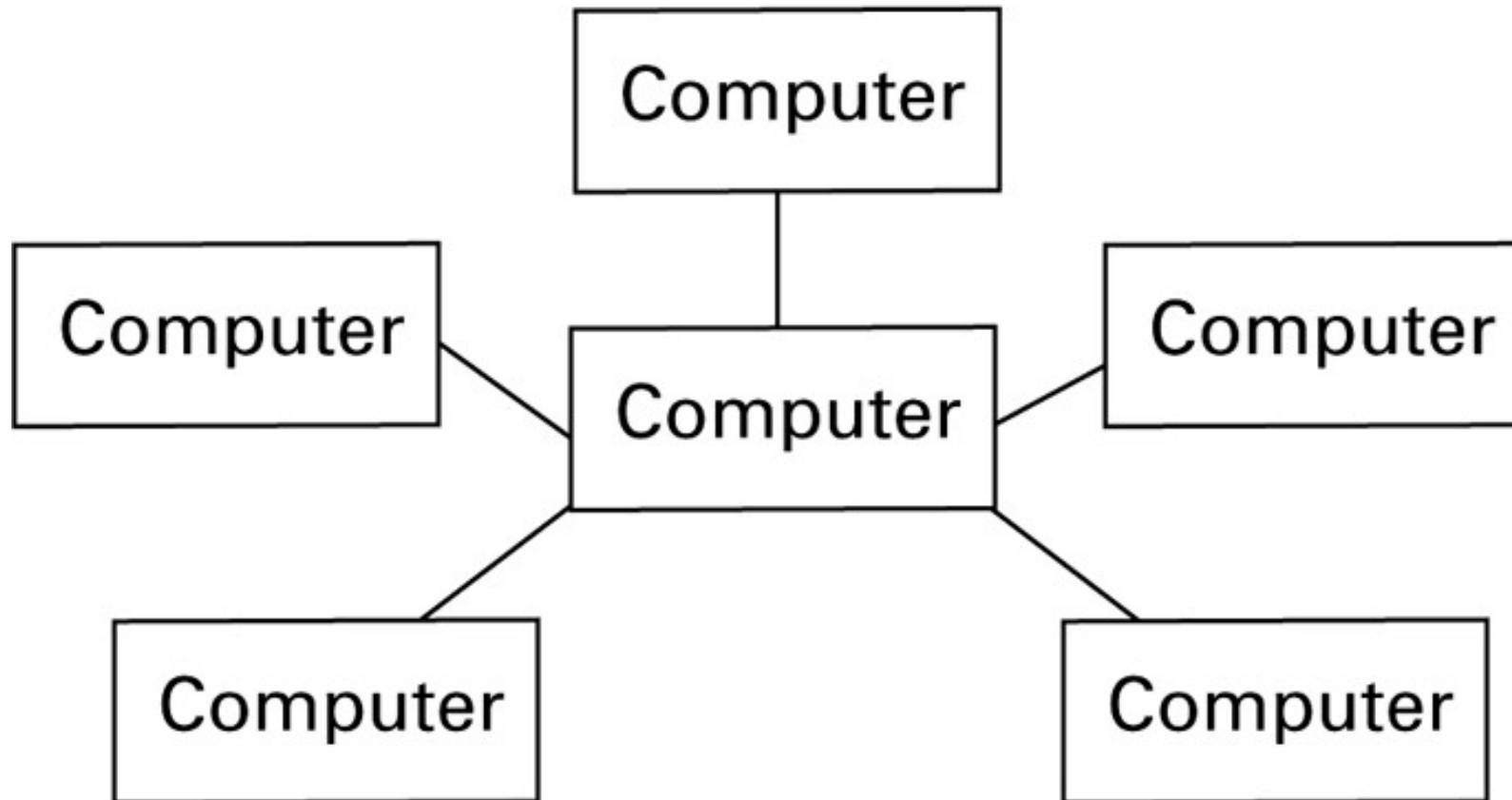
Scope alone does not tell us the speed, topology or ownership.

a. Bus



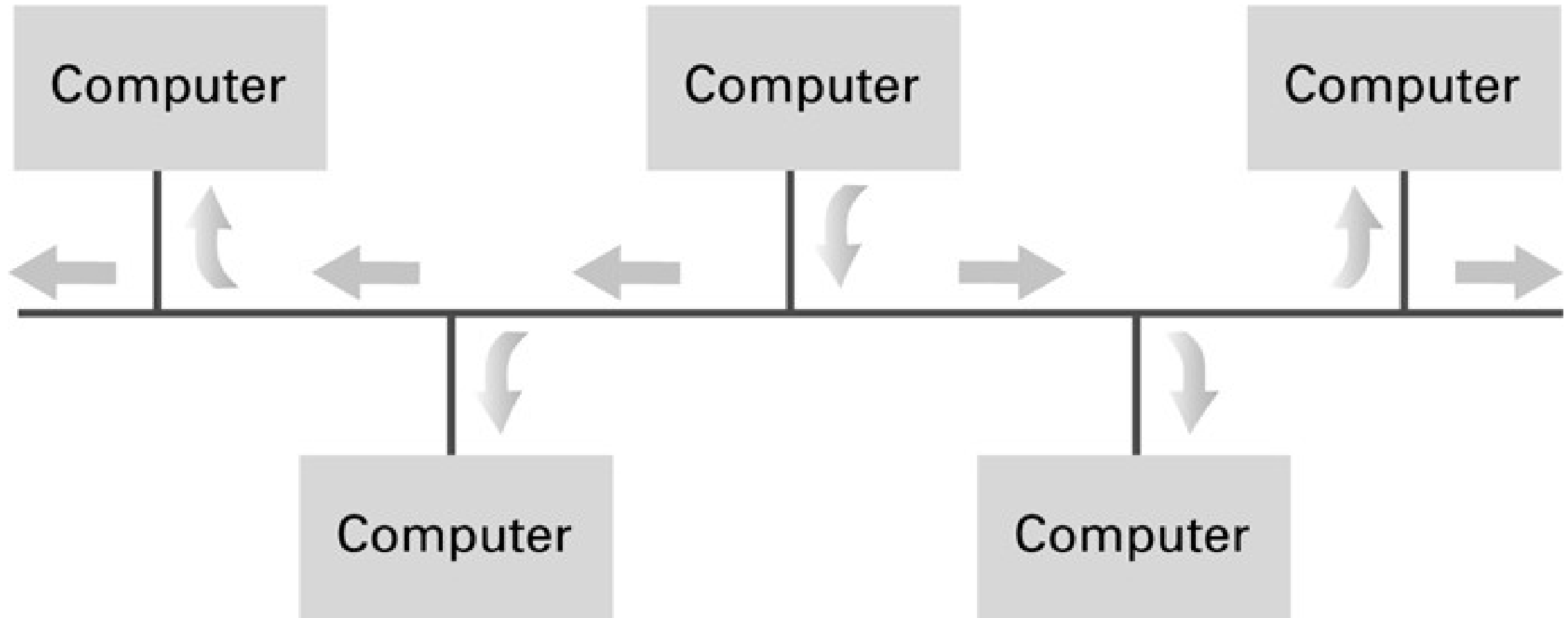
Devices share a communication medium. The arrangement affects access and failure behavior.

b. Star



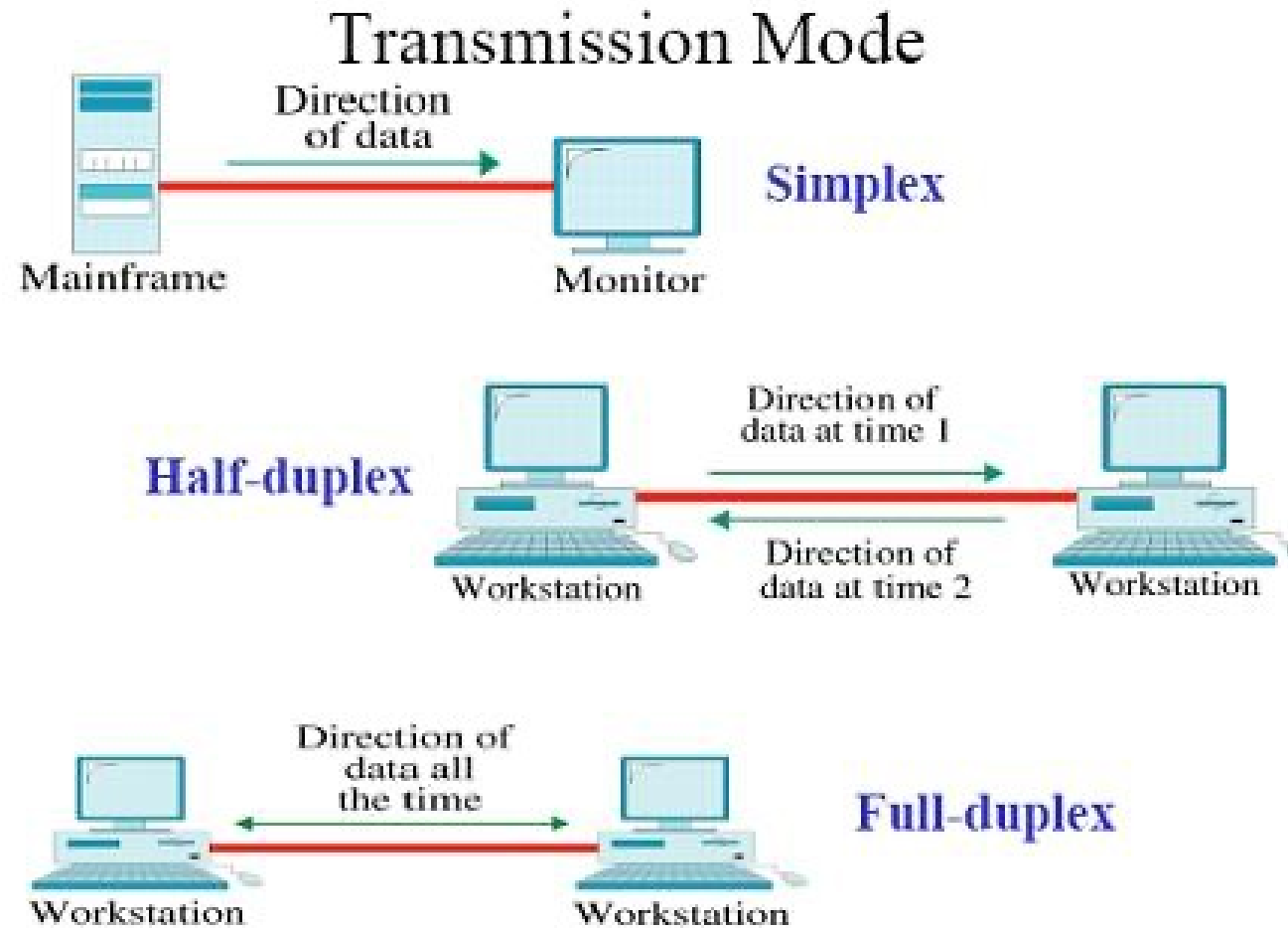
Links meet at a central device. In an Ethernet LAN this is typically a switch.

Sharing a medium



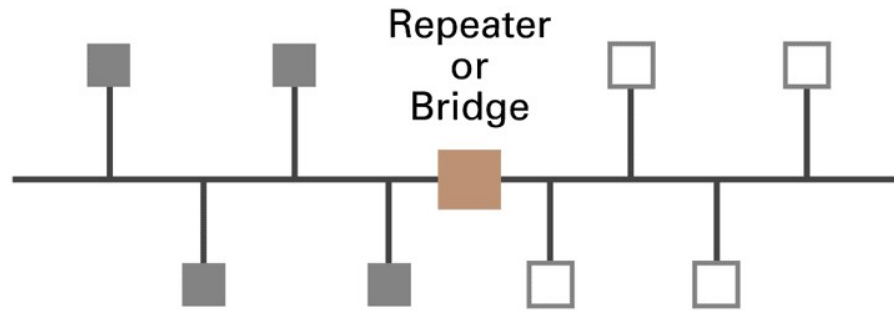
Shared half-duplex Ethernet used collision detection. Full-duplex Ethernet links do not use CSMA/CD.

Direction of communication

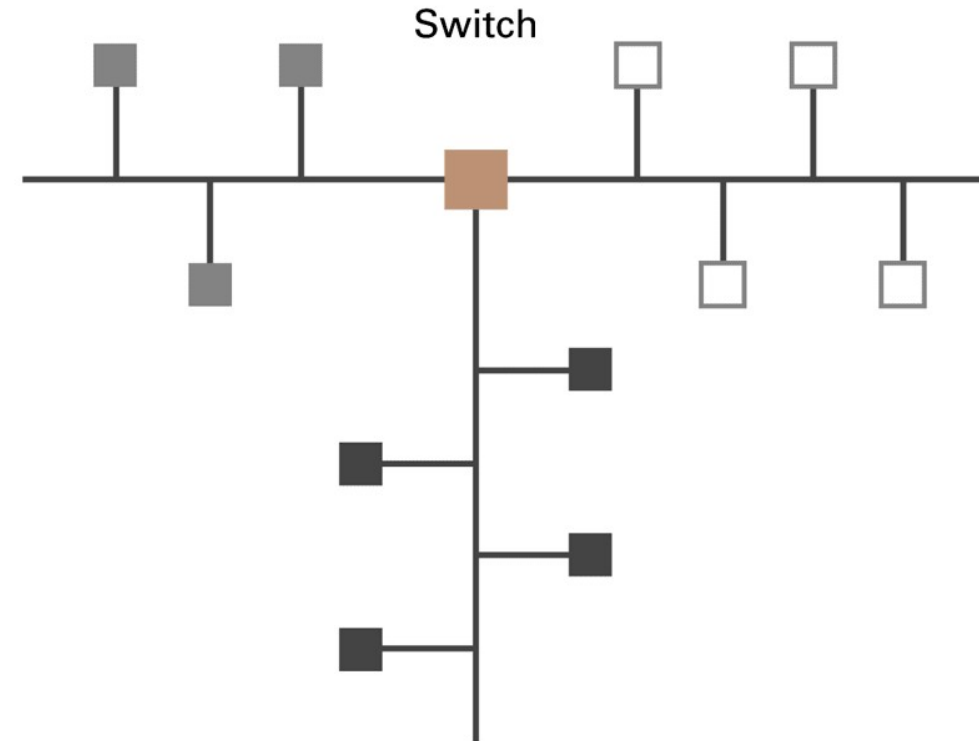


Simplex: one direction. Half duplex: take turns. Full duplex: both directions simultaneously.

Connecting local network segments



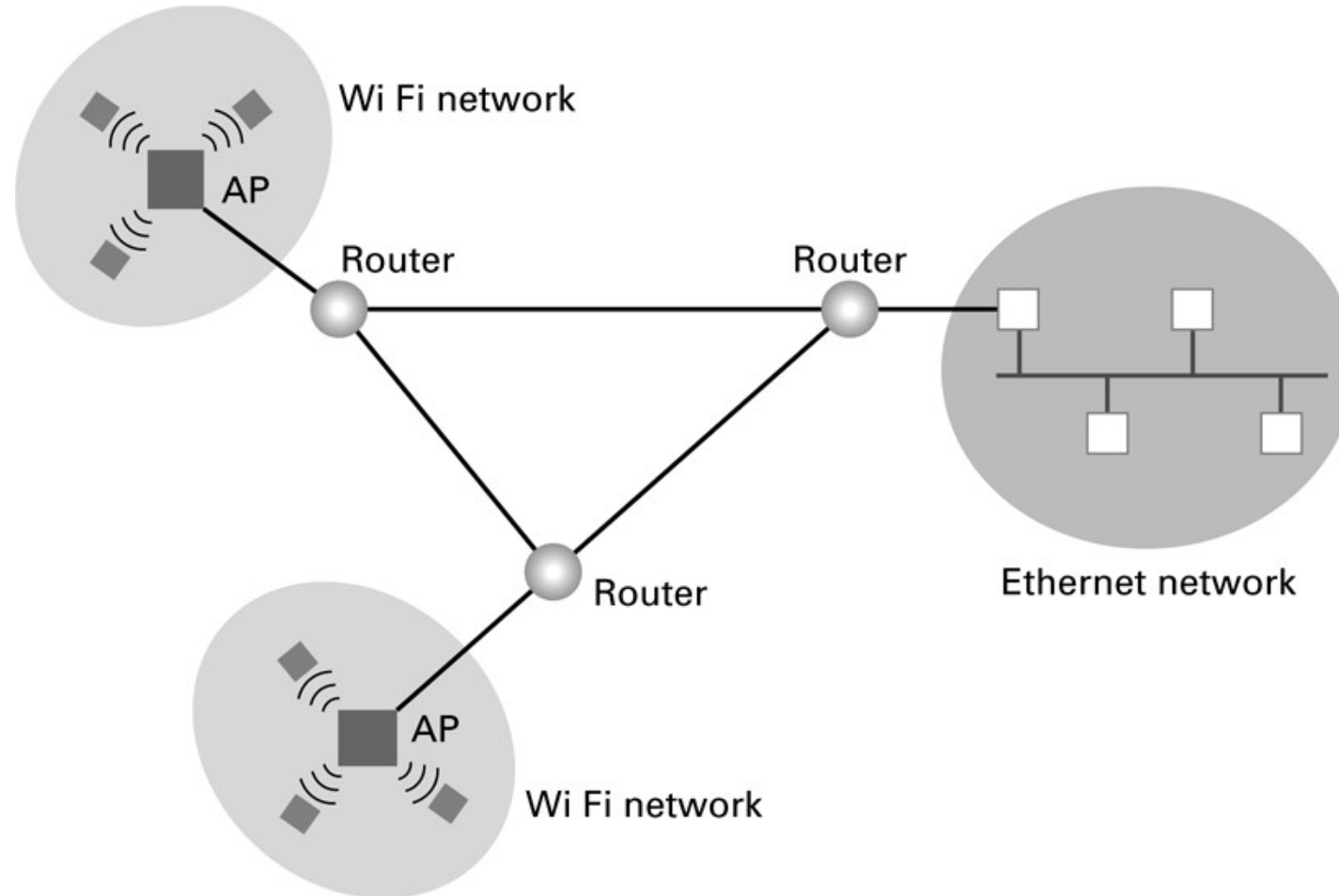
a. A repeater or bridge connecting two buses



b. A switch connecting multiple buses

Repeaters regenerate signals. Bridges and switches forward frames between link-layer segments.

Routers connect networks



An IP packet can cross different local link technologies on its journey.

Different jobs inside a network

Device	Main role	Decision basis
Access point	Connect wireless devices to a LAN	Link-layer communication
Ethernet switch	Forward frames within a LAN	MAC address and forwarding table
IP router	Forward packets between IP networks	Destination IP and forwarding table

A home “router” often combines several of these functions.

Addresses identify different things

An IP address identifies an interface or set of interfaces at the network layer. A device can have more than one.

A MAC address is used for delivery on an Ethernet or Wi-Fi link. A transport port helps identify a service endpoint.

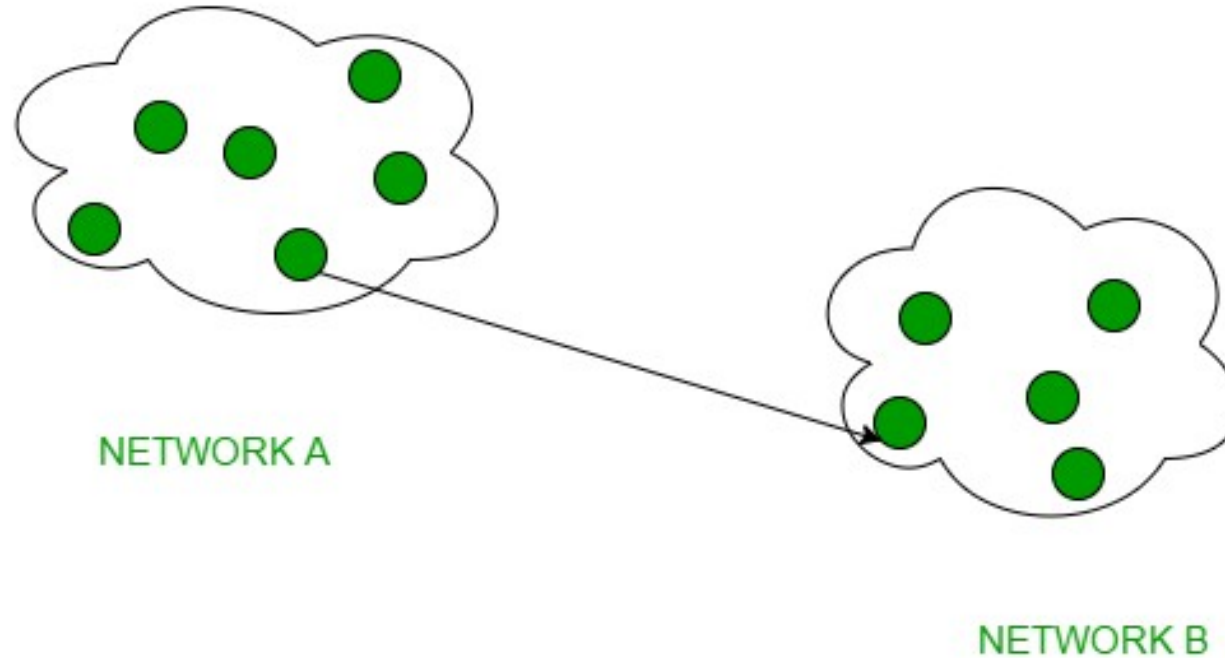
IPv4 uses 32-bit addresses. IPv6 uses 128-bit addresses. Human-readable names and DNS belong to next week.

A forwarding decision at router R

Destination network	Next hop from R
Network A	Deliver on the attached A link
Network B	Forward to router S
Network C	Forward to router T
Other destinations	Use a configured default, or report/drop if unavailable

A packet for C goes to T. R need not store the whole end-to-end journey.

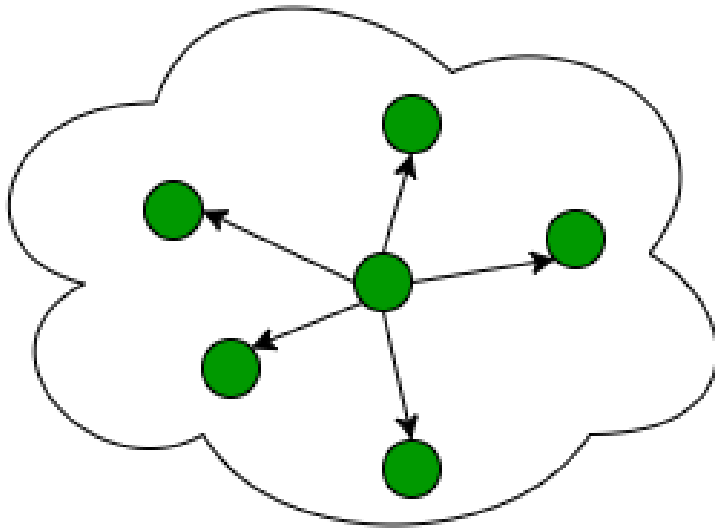
Unicast: one destination



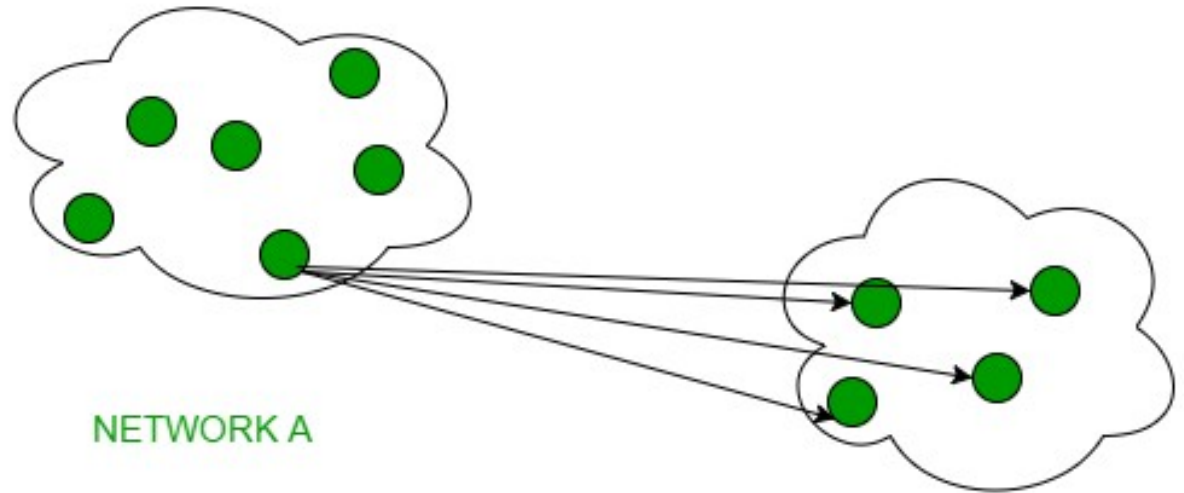
UNICAST EXAMPLE

A sender addresses one receiving endpoint.

Broadcast: a scoped audience



NETWORK CLUSTER

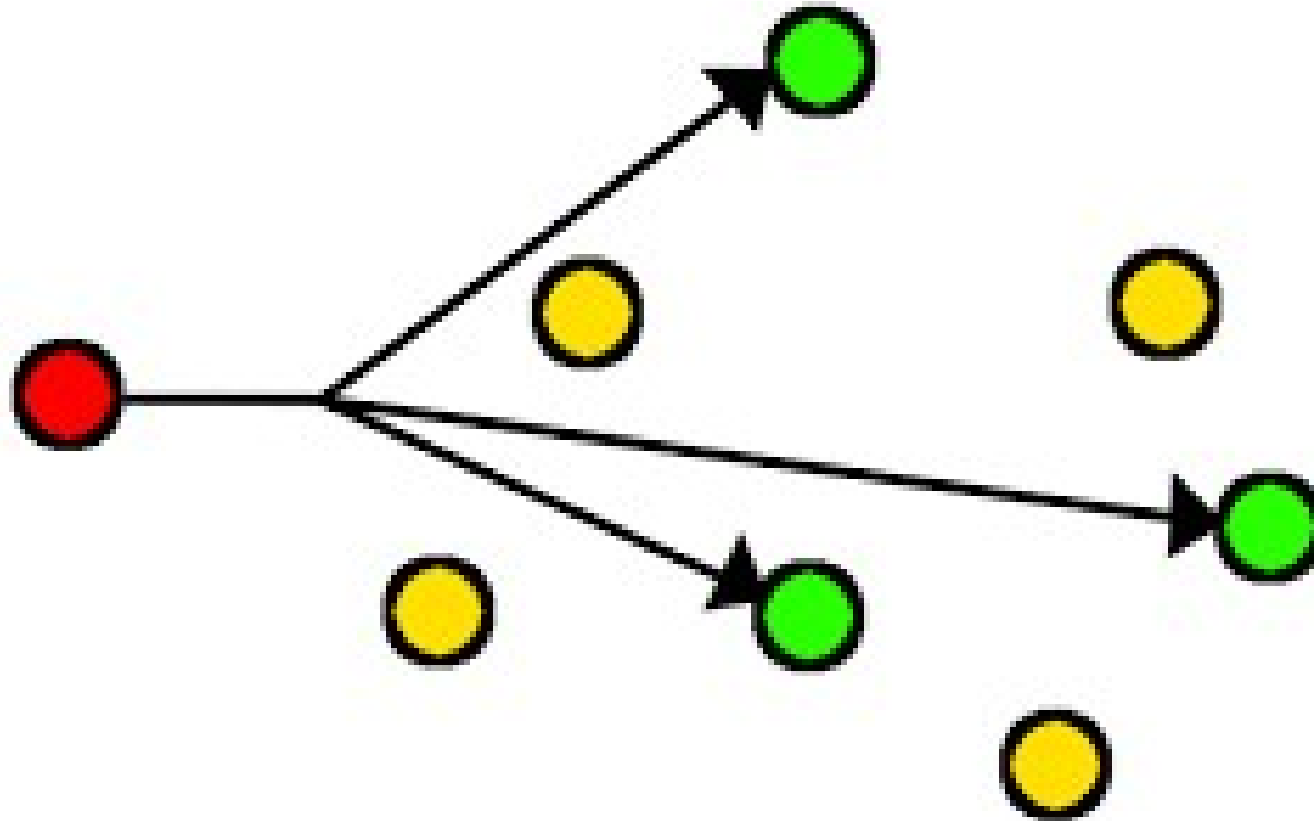


NETWORK A

NETWORK B

Broadcast targets a broadcast domain. IPv4 directed broadcast is normally disabled at routers.

Multicast: a selected group



A multicast group represents interested receivers. Support and delivery depend on the network.

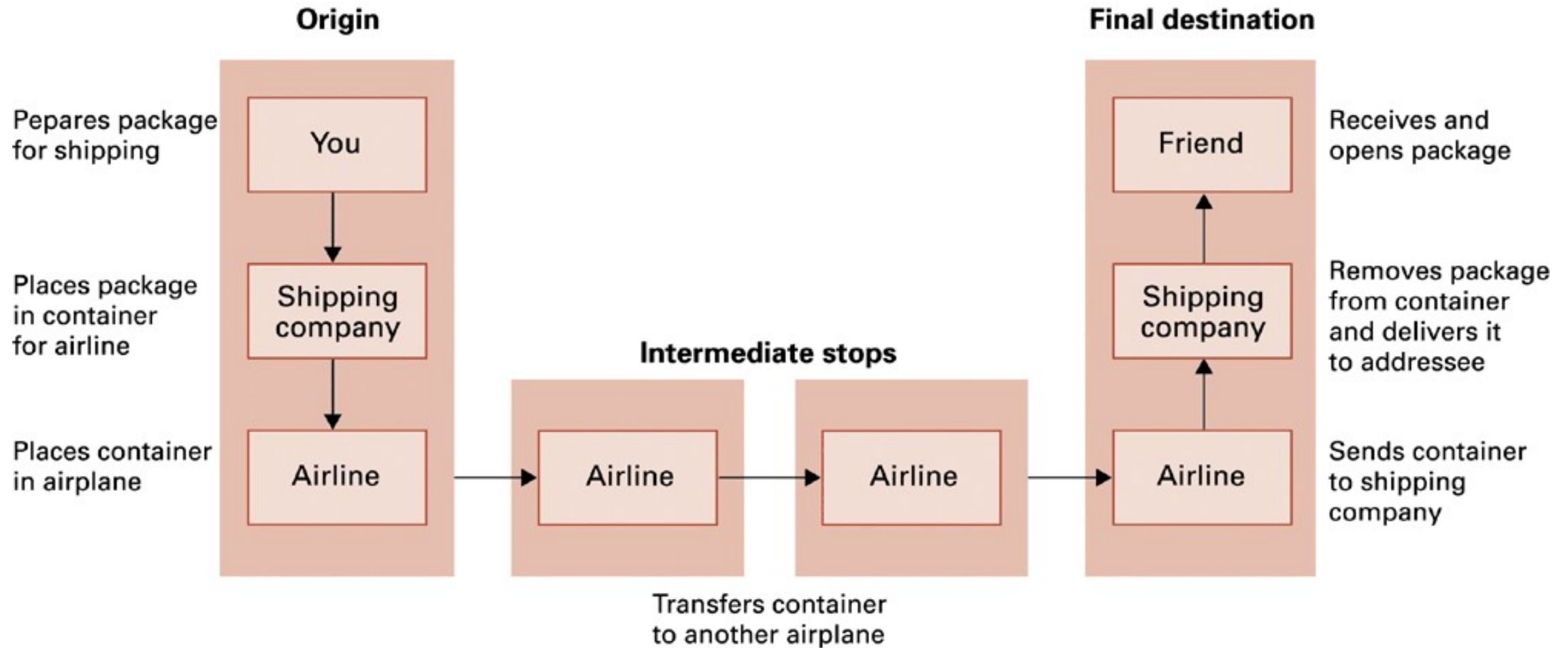
Protocols make independent systems cooperate

A protocol defines message formats and the actions taken when messages arrive, time out or fail.

The same physical link can carry many applications because the layers agree on interfaces.

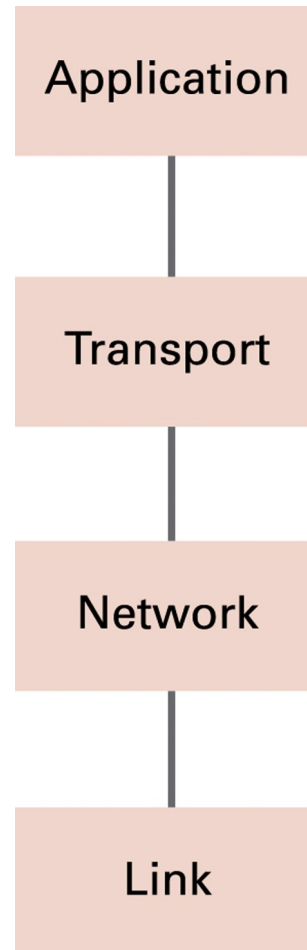
A packet can be lost, delayed, duplicated or reordered. Higher layers decide what the application needs.

Layering: the shipping analogy



Each layer performs a specific part of the delivery task.

Internet protocol layers



Application: meaning

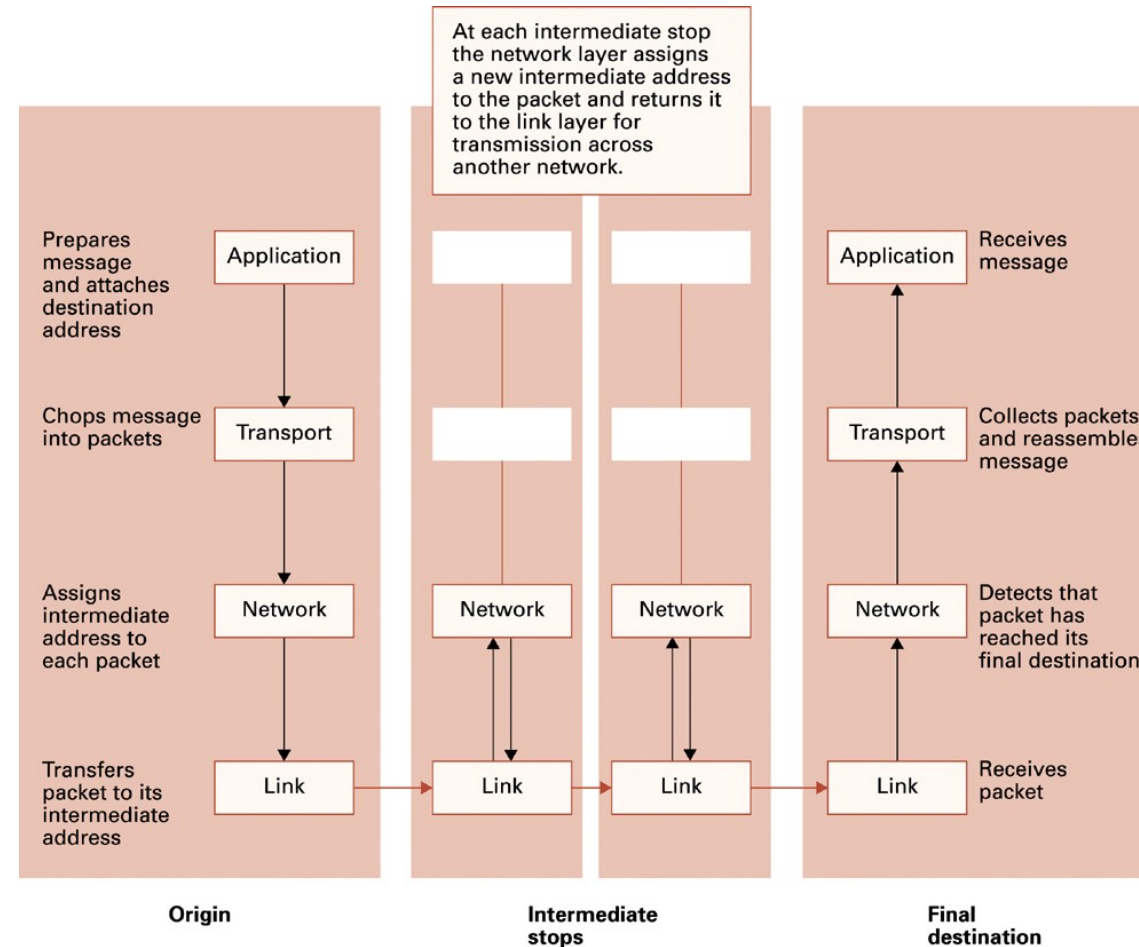
Transport: endpoint service

Network: IP forwarding

Link: one-hop delivery

The four-layer model groups responsibilities. The physical medium sits beneath link communication.

Following a message through the network



Routers choose next hops. Ordinary forwarding keeps the destination IP and changes link-layer delivery.

TCP and UDP provide different services

Property	TCP	UDP
Data unit for the application	Ordered byte stream	Individual datagrams
Loss and order	Retransmission and ordering	No built-in retransmission or ordering
Connection setup	Establishes a connection	No transport handshake
Application responsibility	Define message boundaries	Handle missing/late data as needed

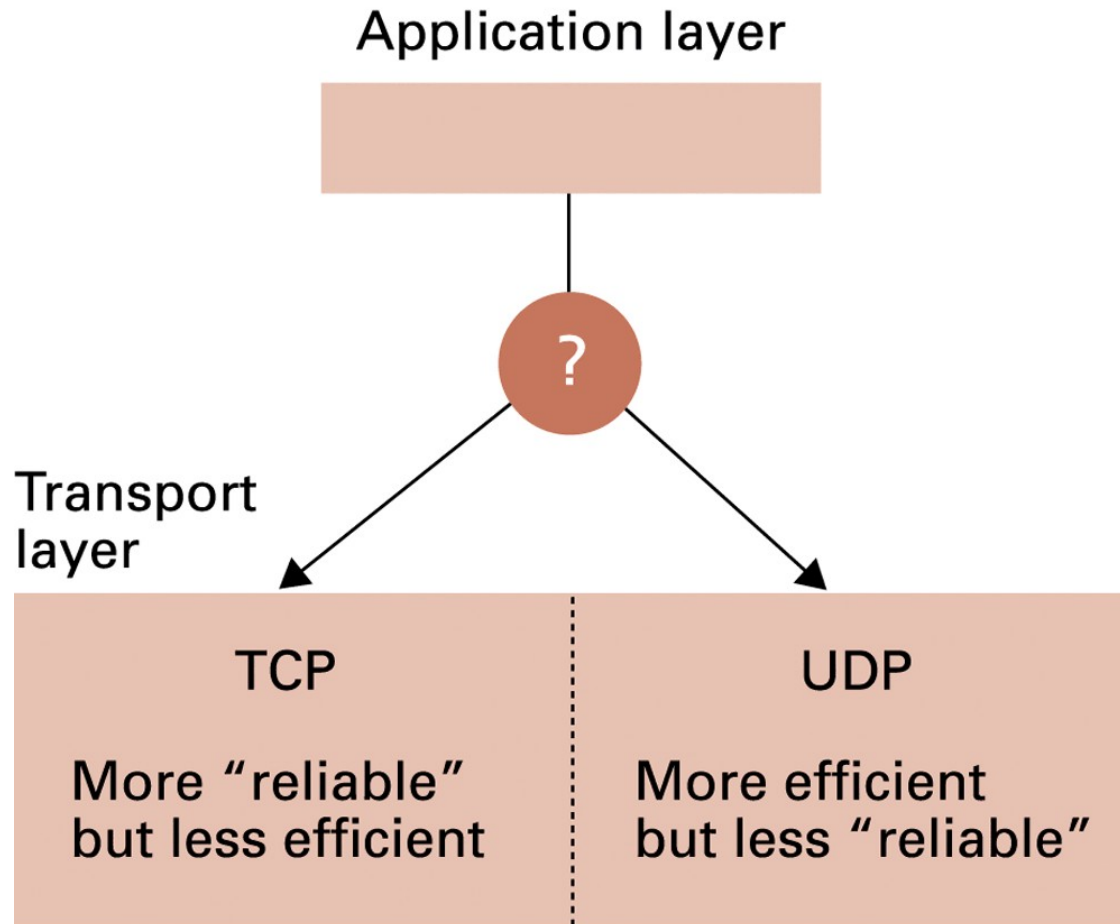
Reliability is a service property. Encryption and application correctness are separate concerns.

A small delivery trace

Event	What can happen next?
Sender sends data A, B, C	Packets travel across shared links
B is lost; C arrives	Receiver may have a gap in the stream
TCP recovers the missing bytes	Application reads ordered bytes A, B, C
UDP carries three separate datagrams	Application must decide how to handle absent B

This conceptual trace counts pieces of data, not actual TCP sequence-number fields.

TCP and UDP: transport guarantees



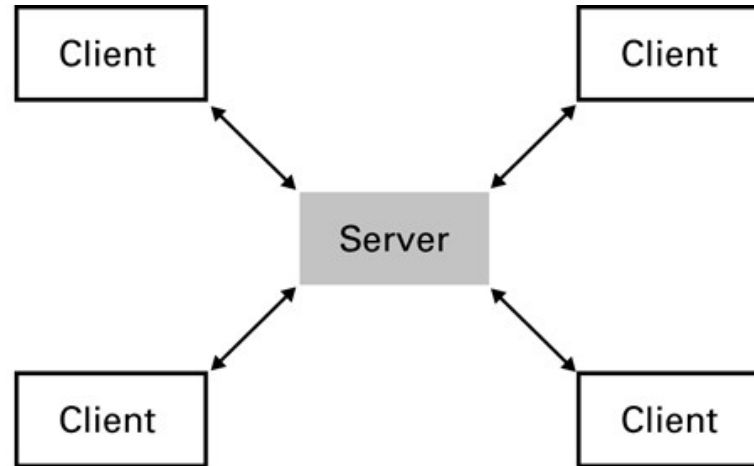
TCP recovers an ordered byte stream or reports failure.

UDP does not guarantee delivery.

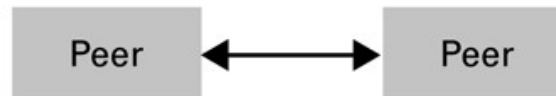
“More efficient” is not a universal speed ranking.

“Reliable” refers to transport service. UDP can support reliable protocols built above it.

Client/server and peer-to-peer



a. Server must be prepared to serve multiple clients at any time.



b. Peers communicate as equals on a one-to-one basis.

These describe application roles, not particular cables or physical topologies.

A command needs an application-level meaning

Suppose an application sends the text LED=1 to a controller. The application must define what that command means.

Transport success alone does not prove that an LED changed. A reply can report acceptance or failure.

A serial link can carry the same text without being an IP network. In the lab, communication support is supplied.

Networks: what to carry forward

Packet switching shares capacity, but queues and failures remain possible.

Addresses and protocol layers separate local delivery, routing and application meaning.

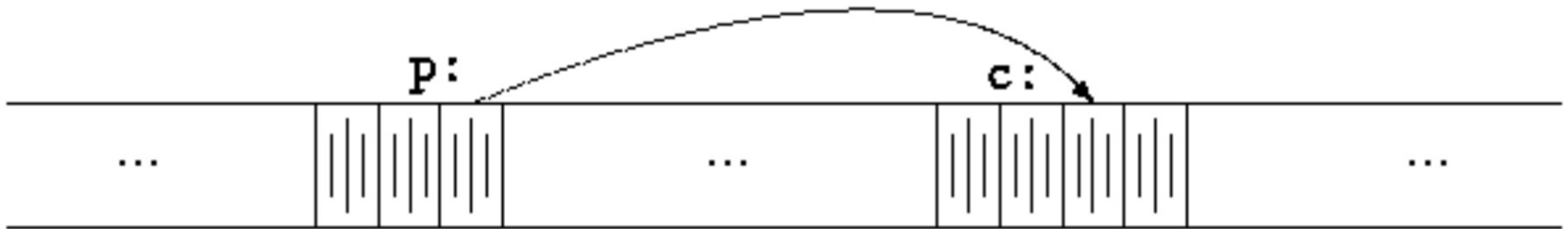
Next week: names, DNS, HTTP, browsers and Web services. Security mechanisms receive their own later treatment.

Addresses, objects and text

Memory diagrams first

Short C traces, then bounded buffers

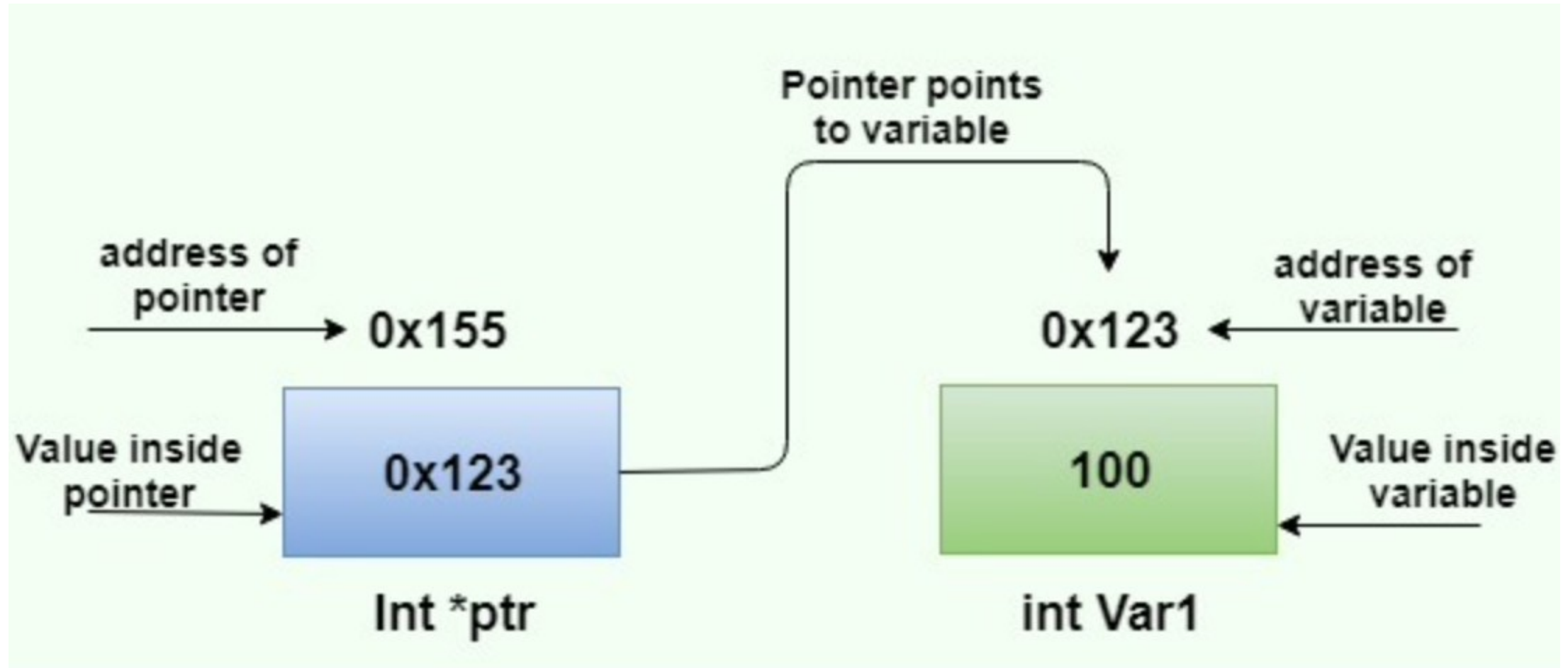
A pointer stores an address



`p = &c;`

The arrow represents the address stored in p. It is not an extra physical wire.

The pointer has its own storage



The pointer's address, the value it stores and the pointed-to value are different quantities.

Reading pointer notation

Expression	Meaning
<code>int *p;</code>	Declare p as a pointer to int
<code>&x</code>	Obtain the address of x
<code>p = &x;</code>	Store that address in p
<code>*p</code>	Access the int object to which p points

A declaration alone does not initialize a local pointer.

One object, two ways to access it

```
int x = 5;  
int *p = &x;
```

```
*p = 8;
```

```
printf("%d\n", x);  
/* prints 8 */
```

p stores the address of x.

Writing through *p changes x.

Changing x does not change p.

Pointer state trace

After statement	x	p	*p
int x = 5;	5	Not declared	Not applicable
int *p = &x;	5	Address of x	5
*p = 8;	8	Address of x	8

The pointer value stays the same while the pointed-to object changes.

Tracing addresses and dereferencing

```
int x = 1, y = 2, z[10];  
int *ip;          /* ip is a pointer to int */  
  
ip = &x;          /* ip now points to x */  
y = *ip;          /* y is now 1 */  
*ip = 0;          /* x is now 0 */  
ip = &z[0];        /* ip now points to z[0] */
```

Follow ip as it first refers to x, then to the first element of z.

Why swapping copies does not swap the caller

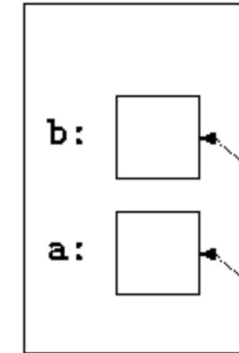
```
void swap(int x, int y)  /* WRONG */  
{  
    int temp;  
  
    temp = x;  
    x = y;  
    y = temp;  
}  
  
swap(a, b);
```

The local x and y change. The caller's a and b keep their original values.

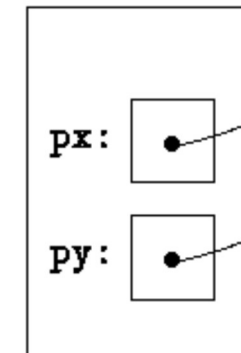
Passing addresses to the function

```
swap(&a, &b);
```

in caller:



in swap:



C copies the pointer values. Dereferencing those copies accesses the caller's objects.

Swapping through pointers

```
void swap(int *px, int *py) /* interchange *px and *py */
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

The caller uses `swap(&a, &b)`. Both pointers must designate writable int objects.

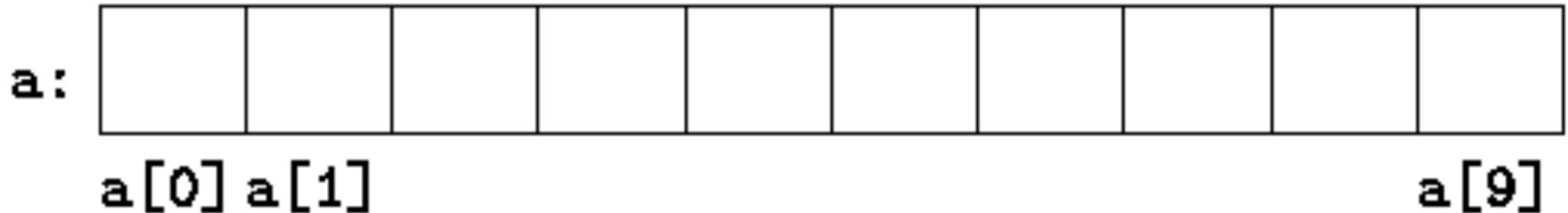
The caller's values during swap

Statement	a	b	temp
Before the call	5	8	Not declared
temp = *px;	5	8	5
*px = *py;	8	8	5
*py = temp;	8	5	5

The pointer parameters refer to the original objects throughout this call.

An array occupies consecutive elements

```
int a[10];
```



Each element has an address. The array also has a fixed extent.

A pointer to the first array element

```
int a[10];
```



```
int *pa;
```

```
pa = &a[0];
```

For valid indices, $pa[i]$ means $*(pa + i)$. The array itself is not a pointer variable.

Pointer movement and bounds

Expression	Meaning for <code>int a[10]</code> and <code>int *p = a</code>
<code>p + 1</code>	Address of the next int element
<code>*(p + 3)</code>	The same element as <code>a[3]</code>
<code>p + 10</code>	One-past pointer: may be formed, not dereferenced
<code>p + 11</code>	Outside the permitted array arithmetic range

A pointer does not carry an automatic array-length check.

Valid pointer use has conditions

Initialize a pointer before use. A null pointer deliberately points to no object and must not be dereferenced.

The target object must still exist and permit the requested read or write. An address of a finished local variable is invalid.

Keep the pointer inside its array's permitted range. A non-null pointer alone is not proof of validity.

A string is a terminated character sequence

Index	0	1	2	3	4	5	6	7
buf[index]	L	E	D	=	1	\0	\0	\0

`char buf[8] = "LED=1";` Length: 5 characters. Capacity: 8 bytes. First terminator: index 5.

Characters, strings and zero

Notation	Meaning
'1'	One character representing the digit 1
"1"	A two-element character array: digit 1, then \0
'\0'	The null character, with value zero
NULL	A null pointer constant, used in pointer contexts

The digit character, string terminator and null pointer play different roles.

Writable array and string literal

```
char buf[8] = "LED=1";  
buf[4] = '0';  
/* buf now contains "LED=0" */
```

buf owns a writable array.

```
const char *label = "LED=1";  
/* Read the literal through label.  
   Do not modify its characters. */
```

label stores an address.

Changing a string literal has
undefined behavior.

Capacity and string length

Declaration	Capacity	Initial length
<code>char a[] = "LED=1";</code>	6 bytes	5
<code>char b[8] = "LED=1";</code>	8 bytes	5
<code>char c[1] = "";</code>	1 byte	0
<code>char d[5] = "LED=1";</code>	5 bytes	No terminator: not a C string

`strlen` requires an existing terminator. It cannot establish that an arbitrary buffer is safe.

Searching for the terminator within a bound

```
int text_length(const char *s, int cap)
{
    int i = 0;
    while (i < cap && s[i] != '\0')
        ++i;
    return i;
}
```

Contract: $\text{cap} \geq 0$ and s points to at least cap readable characters.

A result below cap locates the terminator.

A result equal to cap means none was found.

Tracing text_length(buf, 8)

i	Character read	Next action
0, 1, 2	L, E, D	Advance once for each character
3, 4	=, 1	Advance
5	\0	Stop and return 5

The terminator occupies a byte but is not counted in the string length.

A buffer without a terminator

Call	What is inspected?	Result
<code>text_length(raw, 3)</code>	All three bytes of {L, E, D}	3
<code>text_length(raw, 2)</code>	Only L and E	2
<code>text_length(raw, 0)</code>	Nothing	0

Each result equals its bound: no terminator was found in the permitted region.

Text comparison and safe output

For valid terminated strings, `strcmp(a, b) == 0` tests equal content. Pointer equality tests addresses.

Printing with `%s` also requires a terminator. Establish termination inside the buffer before using string functions.

Receive at most capacity minus one characters when you need to append a terminator. A supplied transport helper must state its buffer contract.

Small tests expose different defects

Test	Expected reasoning
Empty string	Length is 0, terminator is at index 0
LED=1 in an 8-byte buffer	Length 5, spare capacity remains
Terminator in the last available byte	Valid string, no room for another character
No terminator within the bound	Reject as an unterminated buffer

Test the boundary as well as a typical example.

Connection to recitation and Lab 8

Recitation: draw the pointer state before and after a call. Trace a short string and its terminator.

Lab: process a bounded text buffer using fixed test data first. Arduino can supply the text through provided transport.

The programming target is memory and text reasoning. Dynamic allocation and file APIs come later. Network programming is not a prerequisite here.

Check your explanation

Why does `*p = 8` change `x` when `p = &x`? Because both expressions designate the same object.

Why does the pointer version of swap work? The function receives addresses of the caller's objects.

Why is a five-byte array holding `LED=1` not a C string? It has no space for the terminating null character.

Brookshear & Brylow: Chapter 4, Network Fundamentals and Internet Protocols. Original Set 5 supplies the historical and network figures.

Original Set 12 supplies pointer diagrams and the swap examples. GNU C manual: Pointers and String Constants.

IETF RFC 9293, RFC 768 and RFC 8200 explain TCP, UDP and IPv6. Internet Society: A Brief History of the Internet.

Communication history and physical signaling

Additional original figures for explanation and restudy

Signals need a physical carrier



Speaking and listening illustrate a communication channel. The photographs are an analogy.

Why modulation is used

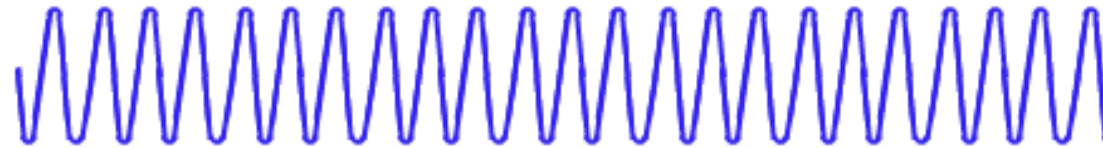


Modulation varies a carrier to convey information. The listed benefits depend on the system and modulation method.

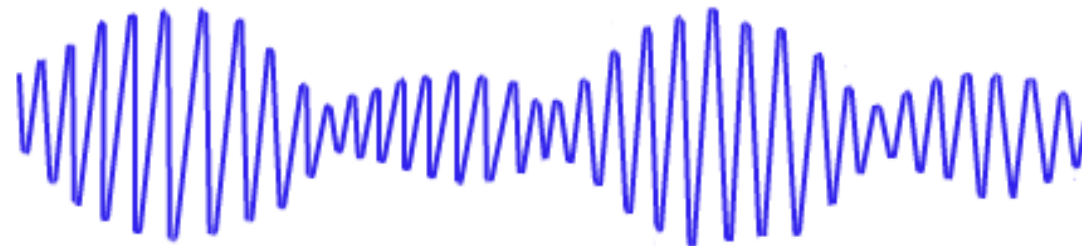
Amplitude modulation



Modulation Signal



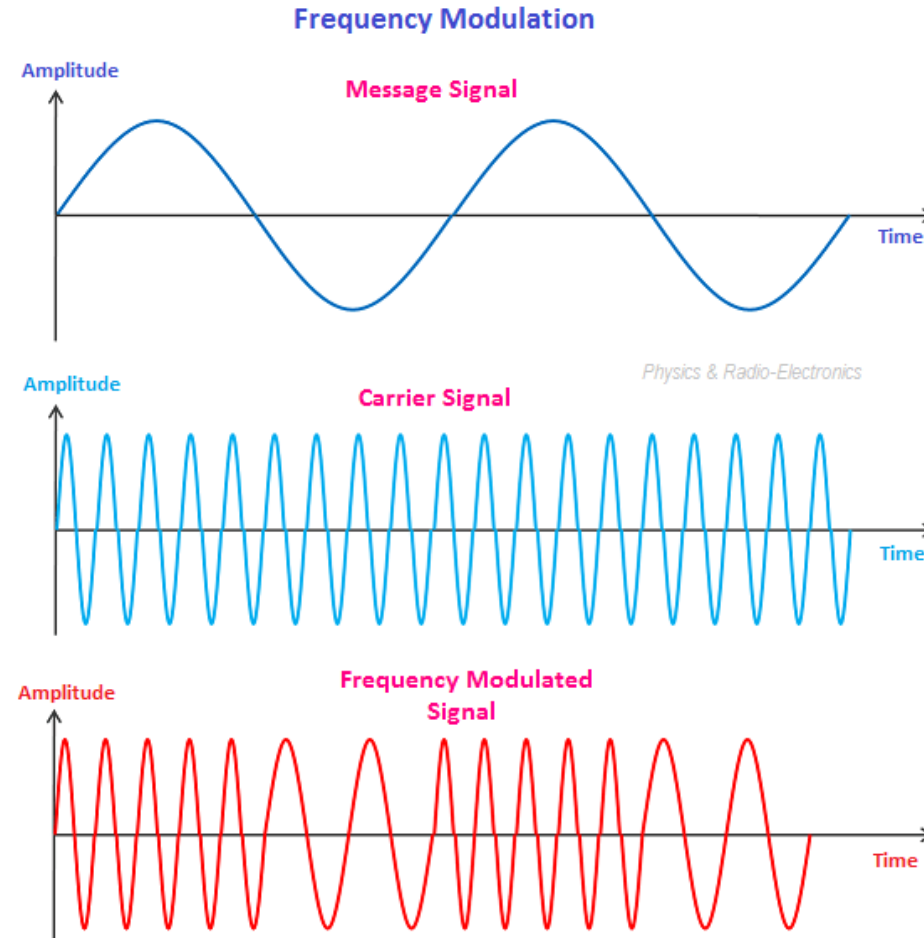
Carrier Signal



Modulated Carrier Signal

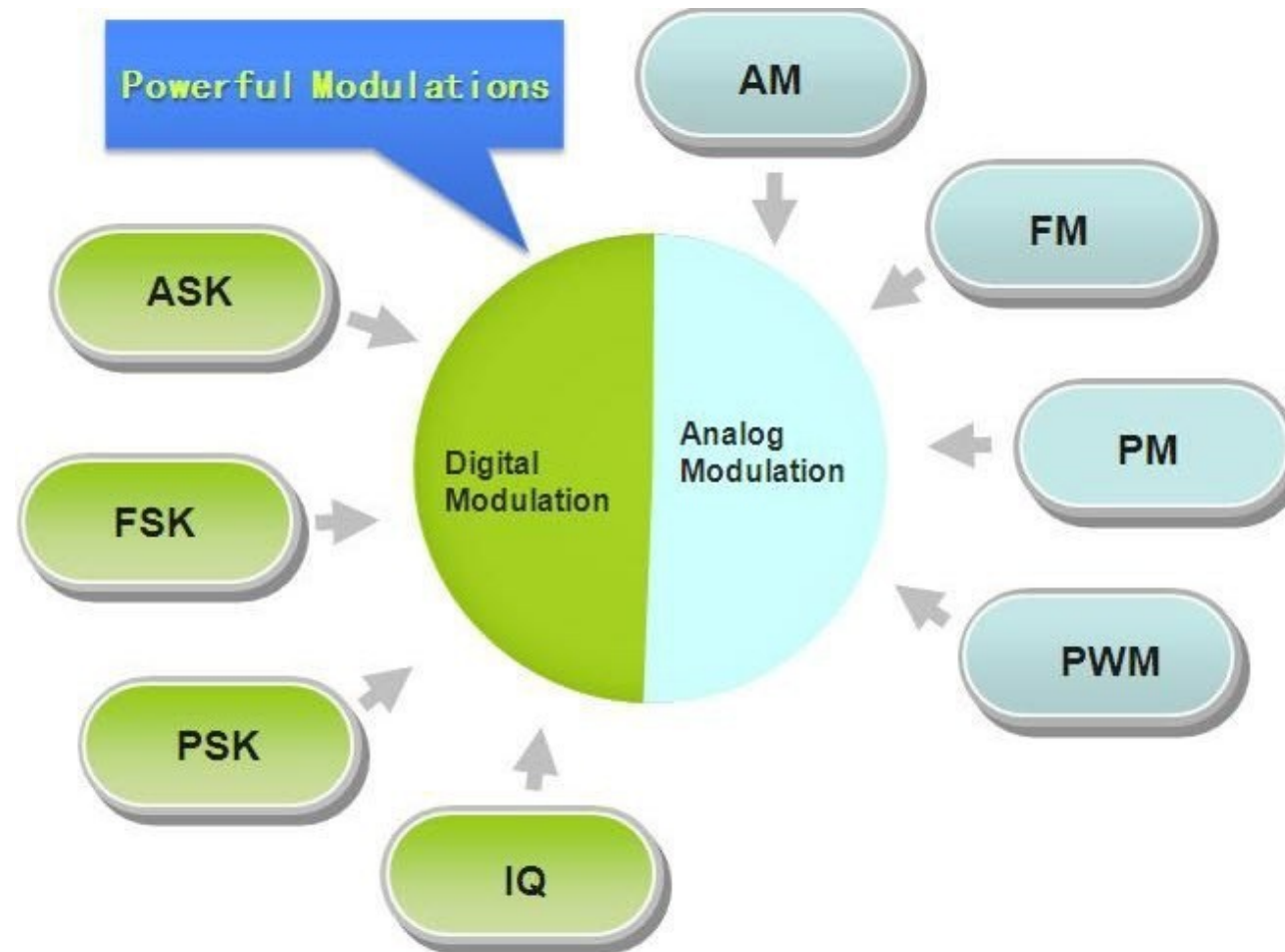
The carrier amplitude varies with the message signal.

Frequency modulation



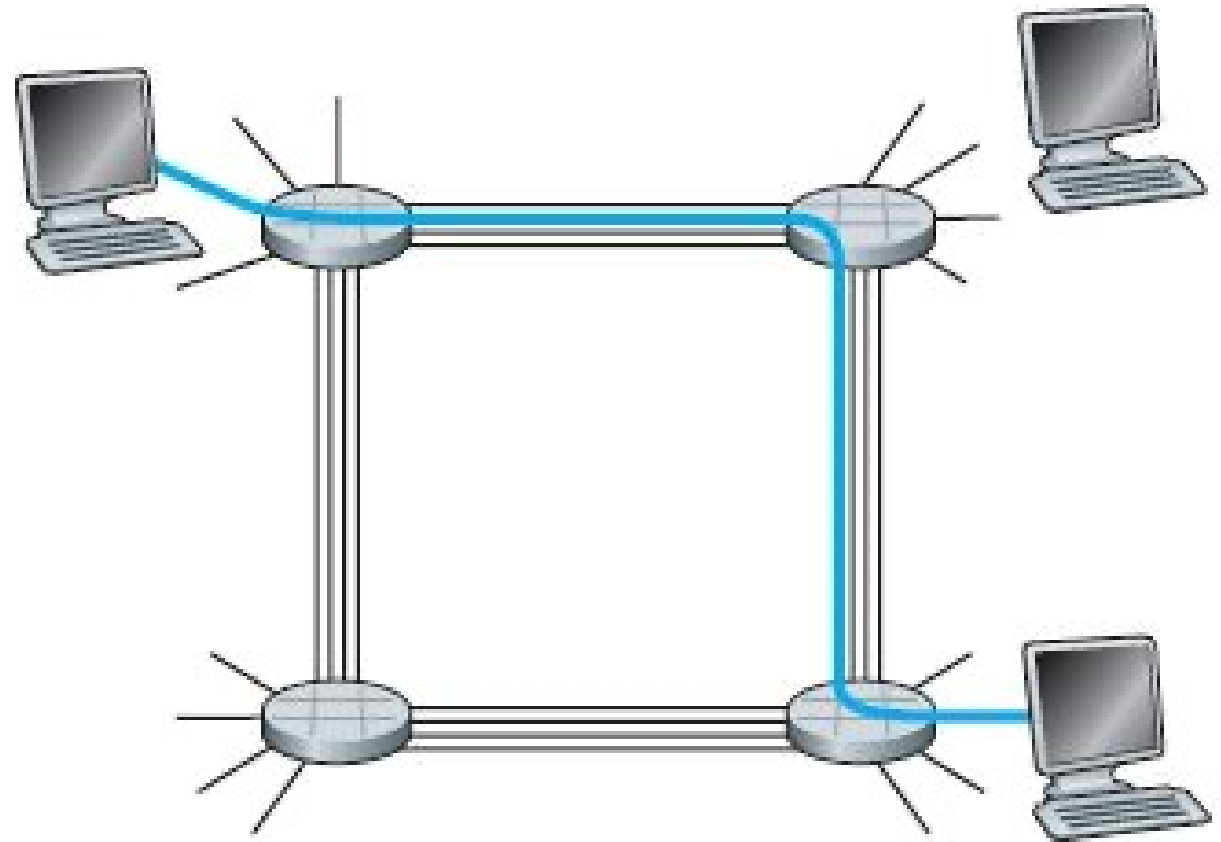
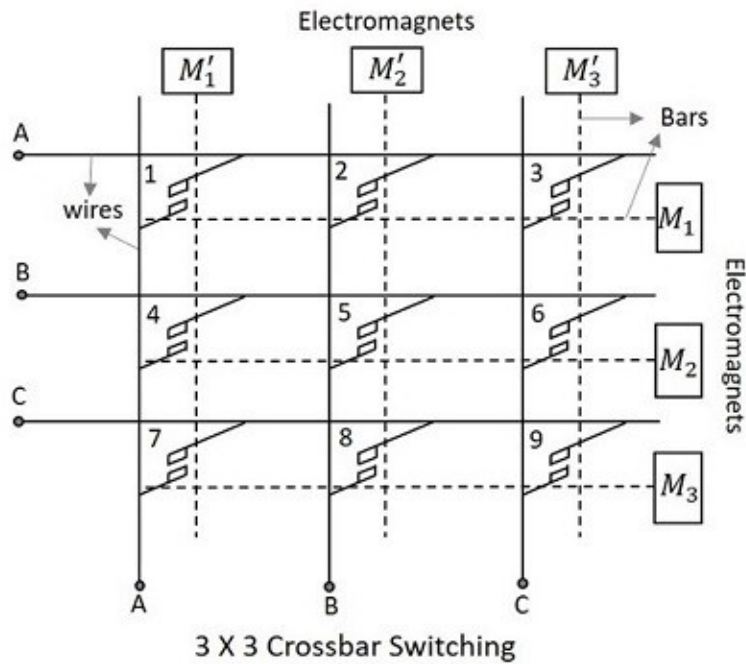
The carrier frequency varies with the message signal.

A family of modulation methods

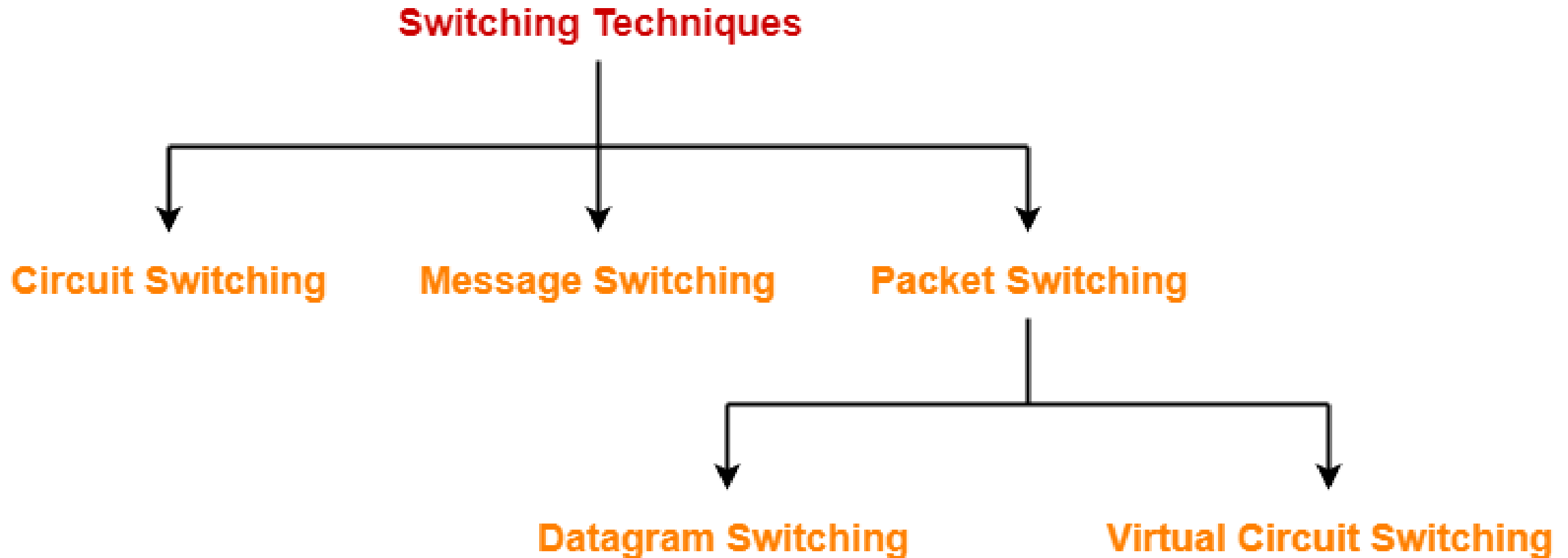


Methods vary amplitude, frequency, phase or pulse properties. I/Q is a representation used by many methods.

Circuit paths and crossbar connections



Two original views of circuit switching: a selected path and a crossbar mechanism.



Message and packet switching use store-and-forward ideas at different granularities.

Reference: a complete pointer example

```
#include <stdio.h>

void swap(int *px, int *py)
{
    int temp = *px;
    *px = *py;
    *py = temp;
}

int main(void)
{
    int a = 5, b = 8;
    swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
```

Expected output: 8 5

Both arguments point to writable local int objects.

Reference: using the bounded result

```
char buf[8] = "LED=1";  
int n = text_length(buf, 8);  
  
if (n < 8)  
    printf("%s (%d)\n", buf, n);  
else  
    puts("No terminator in buffer");
```

Expected: LED=1 (5)

Use the helper shown earlier.

The example uses a fixed buffer, so the bound is known.