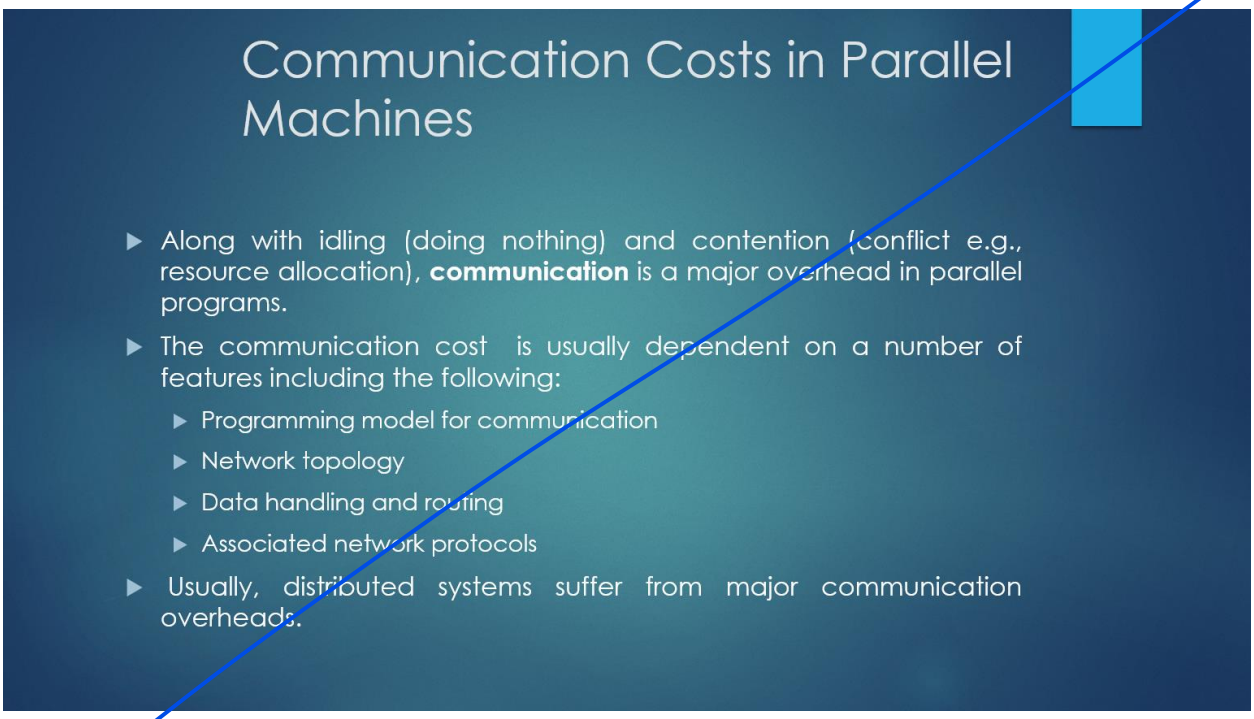


Programming model for communication—Required pattern of the communication in the program

Associated protocols: Security assessments UDP,TCP,

🧠 Concept: Communication Costs in Parallel Machines

When many computers (or processors) work together **in parallel**, they need to **share data and coordinate** with each other.

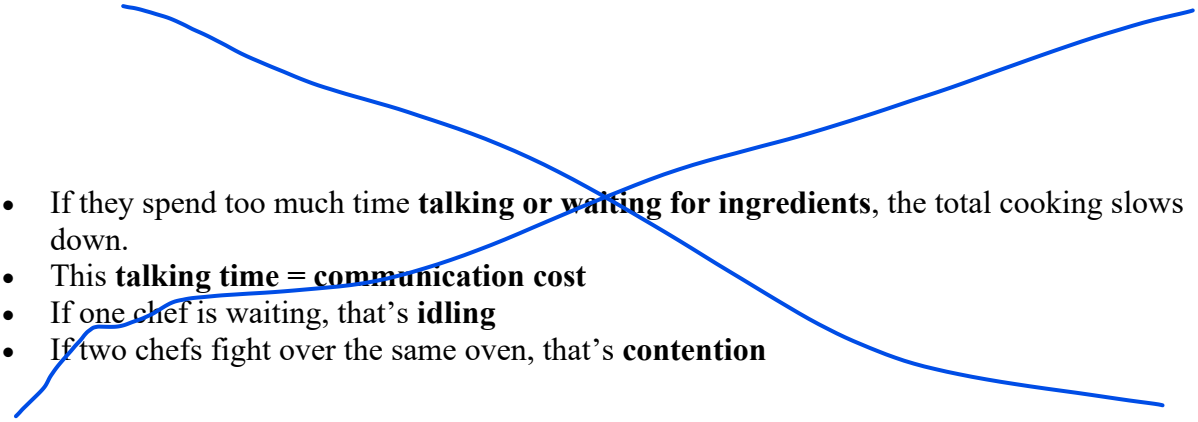
But — whenever they communicate, **time and resources are spent** ⏳
This extra time is called the **communication cost** (or communication overhead).

⚙️ Example (Real-Life Analogy)

Imagine 4 chefs 👨‍🍳👨‍🍳 working together in a big kitchen (parallel system):

- Each chef is cooking one part of a large meal.
- But sometimes, one chef needs ingredients or updates from another chef.

Now:



- If they spend too much time **talking or waiting for ingredients**, the total cooking slows down.
- This **talking time = communication cost**
- If one chef is waiting, that's **idling**
- If two chefs fight over the same oven, that's **contention**

💡 3 Major Overheads in Parallel Systems

1. **Idling** – a processor is doing nothing (waiting for others).
2. **Contention** – processors fight for shared resources (like memory or network).
3. **Communication Cost** – time spent sending/receiving data.

📦 Factors Affecting Communication Cost

Factor	Meaning	Example
Programming Model for Communication	How processors talk to each other — e.g., message passing, shared memory	MPI (Message Passing Interface) or OpenMP
Network Topology	How computers are connected — e.g., ring, mesh, star	In a ring, data must pass through several nodes → slower
Data Handling and Routing	How data travels between nodes	Direct vs indirect routing affects speed
Associated Network Protocols	The rules used for communication	TCP (reliable, slower), UDP (faster, less reliable)

🔐 Associated Protocols (Security & Performance)

- **TCP (Transmission Control Protocol)**
  Reliable connection → ensures data arrives safely but adds delay (more cost).
- **UDP (User Datagram Protocol)**
  Fast but unreliable → used when speed is more important than accuracy (like live video streaming).

In distributed systems, TCP is often used for accuracy, but it increases communication cost.

🚨 Why Distributed Systems Suffer More

In parallel machines (like shared-memory systems), communication may happen inside the same machine — **fast**.
But in distributed systems, processors are **on different computers connected by a network** — **slow and costly** due to:

- Network latency
- Packet loss
- Routing delays
- Security checks (protocol overhead)

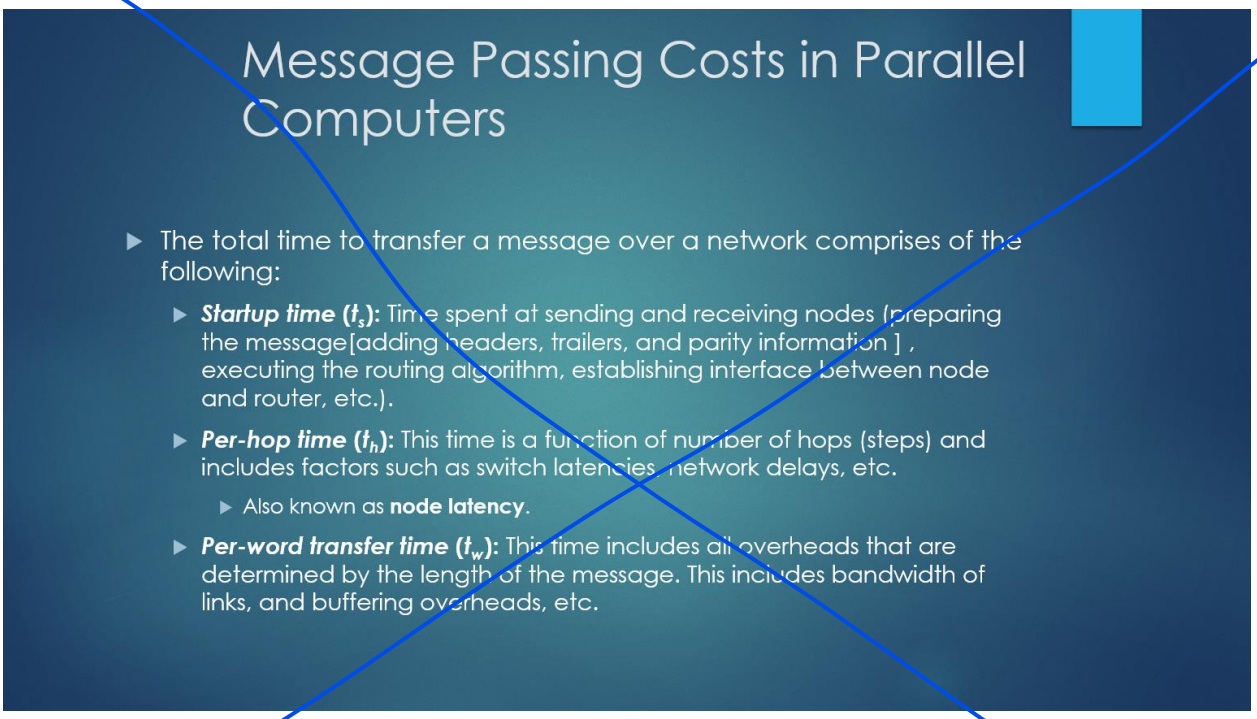
🧩 In Short

Communication cost = Time + Resources used to exchange data between processors.

It depends on:

- Programming model
- Network structure
- Routing methods
- Protocols used

And it causes **overheads** (slowing down) in distributed systems.



t_h also accounts for the latency to take decision of choosing next channel to which this message shall be forwarded

If channel bandwidth is r words/s then each word take t_w= 1/r to traverse the link.

🧠 Concept Overview

In **parallel or distributed computers**, processors communicate by **sending messages** to each other.
This communication isn't free — it takes **time**.
So, the **total time to send a message** =

Startup time + Per-hop time + Per-word transfer time

⏱️ 1. Startup Time (ts)

📌 **Meaning:**
The time spent **before** the message actually starts moving through the network.

💬 **Includes:**

- Preparing the message (adding headers, trailers, parity bits, etc.)
- Executing routing algorithms
- Establishing connection between sender and receiver

💡 **Think of it like:**
You're writing a letter — you need to write it, put it in an envelope, write the address, and hand it to the post office.
That's all **startup time** ⏳

🧍 2. Per-hop Time (th)

📌 **Meaning:**
Time taken for the message to pass **through each intermediate node (hop)** in the network.

💬 **Includes:**

- Switch latency (delay in each router/switch)
- Decision-making delay (which path to forward message next)
- Network propagation delay

💡 **Analogy:**
Imagine your letter travels through 3 post offices before reaching the destination.
Each post office takes a few seconds to check and forward it — that's **per-hop time**.

➡️ **Also called:** *Node latency.*

📦 3. Per-word Transfer Time (tw)

📌 **Meaning:**
The time needed to actually **send the data bits (words)** over the link once it starts moving.
Depends on **message size** and **network bandwidth**.

💬 **Formula:**
If channel bandwidth = `r words/second`,
then

tw = 1 / r

💡 **Example:**
If the link can send 10,000 words per second,
then `tw = 1 / 10000 = 0.0001 seconds per word`.

🧩 Total Message Passing Cost Formula

Let:

- `m` = number of words in message
- `l` = number of hops

Then total time **T** is given by:

T = ts + l × th + m × tw

✅ **Where:**

- `ts` → setup/startup time
- `l × th` → total delay through all hops
- `m × tw` → time to send the actual data

🌍 Real-World Example

Imagine sending a 1 MB file through several routers on the internet:

- **Startup time (ts):** You open your laptop, compress the file, and click "Send" (setup phase).
- **Per-hop time (th):** The file travels through multiple routers — each router checks headers and forwards it (hop delay).
- **Per-word time (tw):** The actual bits of your file move through the network cables (depends on bandwidth).

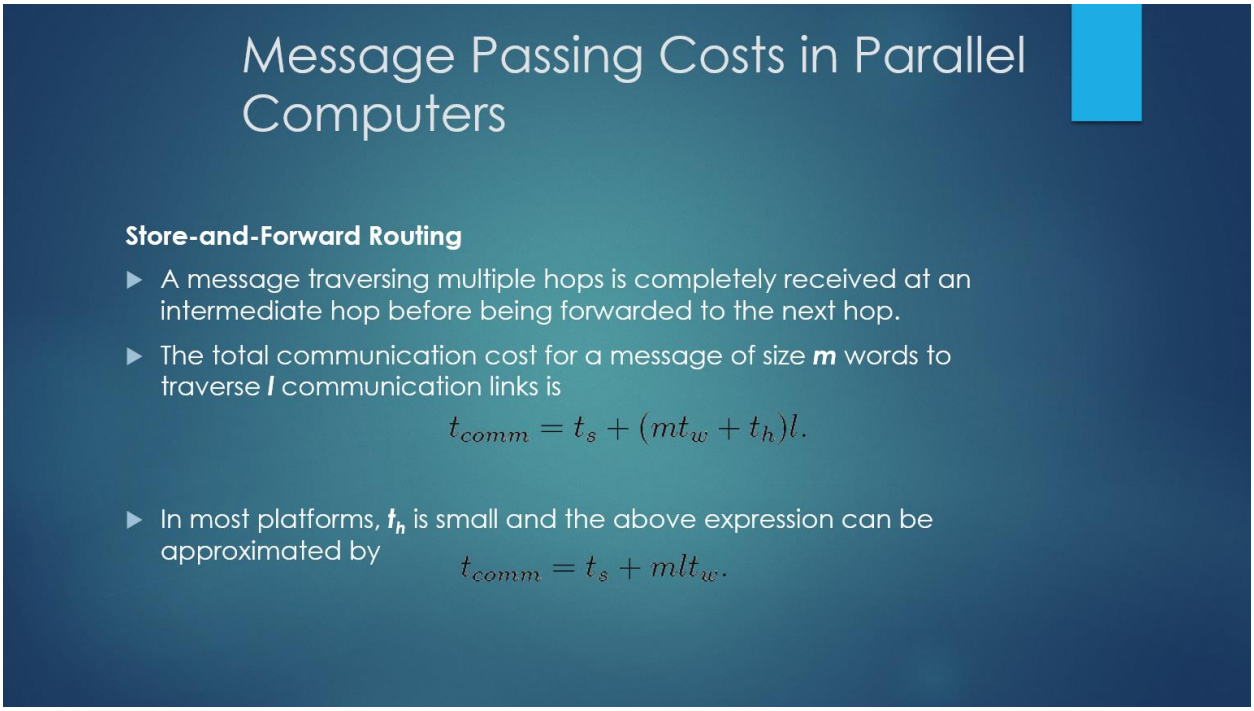
If one router is slow or the network is congested → total message passing time increases.

📜 In Short (Exam-Ready Summary)

Term	Meaning	Depends On	Example
ts (Startup time)	Preparing the message for sending	CPU speed, routing, interface setup	Adding headers, establishing connection
th (Per-hop time)	Delay per intermediate node	No. of hops, switch latency	Router checking & forwarding
tw (Per-word time)	Time per data word sent	Bandwidth, buffering	Link speed (1/r words/sec)

Total message time:

T = ts + l × th + m × tw



Cost of header transfer at each hop (step) t_h.

Ts is startup time

Mtw id cost of transferring m words over this link

🧠 Concept: Store-and-Forward Routing

When a **message travels through multiple hops (routers/switches)** in a network, **each intermediate node (hop)** first **receives the entire message**, stores it, and **then forwards it** to the next hop.

Hence the name → **Store and Forward**.

⚙️ Step-by-Step Example (Simple Analogy)

Imagine 3 post offices 📦🏤🏤🏤 in a chain:

- You send a parcel from Office A → Office B → Office C → Office D.
- Office B must **receive the full parcel**, check it, and then **send it** forward to C.
- It cannot start forwarding before completely receiving it.

That is **store-and-forward routing.**

🧩 Formula for Total Communication Cost

If:

- `ts` = startup time
- `th` = per-hop time (latency per router/switch)
- `tw` = per-word transfer time
- `m` = number of words (message size)
- `l` = number of links (hops)

Then total time (T) to send the message is:

T = ts + l × (th + m × tw)

🔍 Explanation of Each Term

Symbol	Meaning	Description
ts	Startup time	Time to prepare message, establish link, add headers, etc.
th	Per-hop time	Delay at each node while processing header (decision-making, buffering, etc.)
m × tw	Message transfer time	Time to send the message (depends on message size and link bandwidth)
l	Number of hops	More hops → more total delay

💡 Simplified (Approximation)

In most platforms, **th is very small** compared to `m × tw`.
So we can approximate:

T ≈ ts + l × (m × tw)
or
T ≈ ts + m × tw × l

This means total delay mainly depends on **message size (m)** and **number of hops (l)**.

🧮 Alternative View

If channel bandwidth = `r words/second`, then

tw = 1 / r

So the total cost becomes:

T = ts + l × (th + m / r)

🌍 Real-World Example

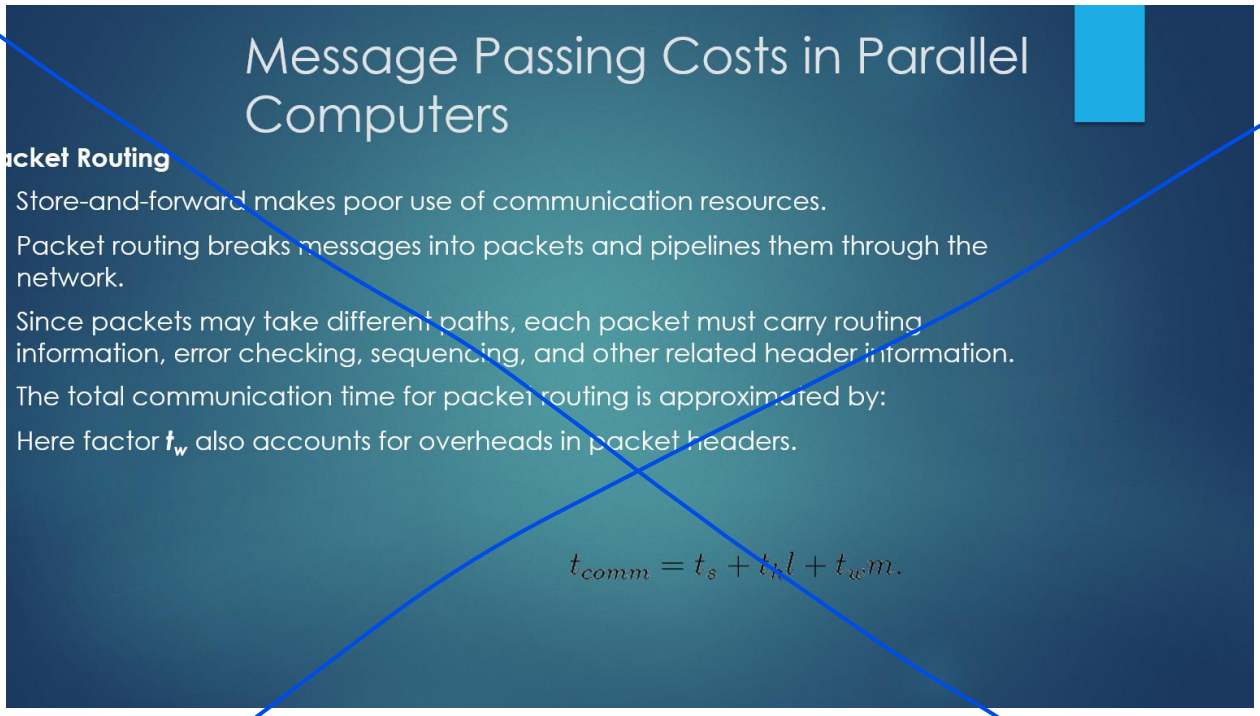
Imagine sending a 5 MB file through 3 routers:

- Each router must fully receive the file before sending it forward (store-and-forward).
- If one link is slow, the whole process slows down.
- That's why file transfers (like via email attachments) take time — every server in the chain processes and stores the full data before passing it on.

📜 Exam-Ready Summary Table

Term	Meaning	Depends On	Example
ts	Startup time	CPU setup, header, interface	Preparing message
th	Per-hop latency	Routing delay, switching	Each router's processing

Term	Meaning	Depends On	Example
m × tw	Message transfer cost	Bandwidth, message size	Sending all data bits
l	Number of hops	Network topology	No. of routers in path
T = ts + l × (th + m × tw)	Total cost	Combined effect	—



Error checking---parity information

Sequencing---packet order number

Related headers: layers headers, addressing headers

🧠 Concept: Packet Routing

In **Store-and-Forward Routing**, the *whole message* must reach one hop before moving to the next — which causes **delays** and **poor utilization** of the network.

To fix that, we use **Packet Routing** 🚀

In **Packet Routing**, the message is **divided into smaller packets**,
and these packets **travel through the network like a pipeline** — multiple packets can be in transit at the same time.

⚙️ Simple Analogy (Real-Life Example)

Imagine you have to send 100 books 📚 from Lahore to Karachi.

- **Store-and-Forward:** You wait till all 100 books are packed into one big box and sent — next truck starts only after receiving full box.
- **Packet Routing:** You send 10 smaller boxes (packets). As soon as the first box leaves, you start preparing and sending the next — pipeline effect!

✅ Result: Network is busy continuously → better utilization
❌ But: Each small box must have **address, tracking info, label** → overhead (header)

⚙️ How It Works

1. The message of **m words** is broken into **k packets**.
2. Each packet travels **independently** through the network.
3. Packets can even take **different routes**.
4. Each packet contains its **own header** (with routing, error checking, and sequencing info).
5. Receiver reassembles packets in correct order.

🧩 Formula for Total Communication Time

If:

- `ts` = startup time
- `th` = per-hop latency
- `tw` = per-word transfer time (includes header overhead)
- `m` = message size (in words)
- `l` = number of hops (links)

Then:

T ≈ ts + l × th + m × tw

💬 **Note:**
Here `tw` already includes the **packet overhead** (header + trailer + error check + sequencing).

🎞️ Pipeline Effect

Because packets are pipelined, the communication **overlaps** — meaning the next packet can start before the previous one finishes.
This reduces **total waiting time** compared to store-and-forward.

Hence packet routing is **faster** and **more efficient**.

🔍 Comparison: Store-and-Forward vs Packet Routing

Feature	Store-and-Forward	Packet Routing
Message Handling	Entire message sent as one unit	Message divided into packets
Utilization	Poor (waiting at each hop)	Better (pipelining)
Delay	Longer	Shorter
Header Overhead	Once per message	Repeated for each packet
Path	One fixed path	Packets can take different paths
Reliability	Simpler	Needs reordering & error checking
Efficiency	Lower	Higher overall

🌍 Real-World Example

When you send a video or file over the Internet, it's not sent as one giant block.
Instead, it's broken into thousands of **packets** using **TCP/IP**.
Each packet has:

- A **header** (source, destination, sequence number)
- **Payload** (actual data)
- **Error check**

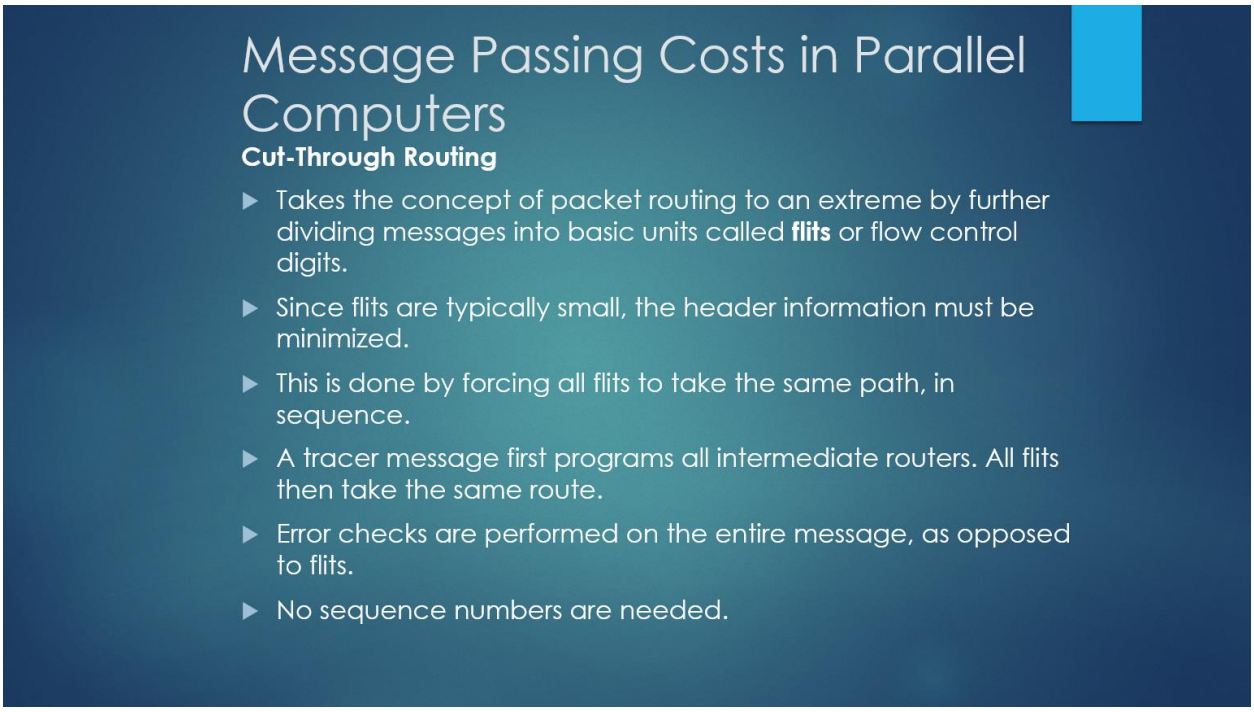
Then these packets travel possibly through **different routes**, and your device reassembles them when they arrive.

That's **packet routing** in action! 🌐

Exam-Ready Summary

Symbol	Meaning
ts	Startup time (initial setup)
th	Per-hop latency
tw	Per-word time (includes packet header overhead)
$T \approx ts + l \times th + m \times tw$	Total time for message transfer

✅ **Packet routing = faster + efficient + overhead in header.**



Sequencing information is not needed as all the packets are following same path which ensures in-order delivery

Message Passing Costs in Parallel Computers

Cut-Through Routing

- The total communication time for cut-through routing is approximated by:

$$t_{comm} = t_s + t_h l + t_w m.$$

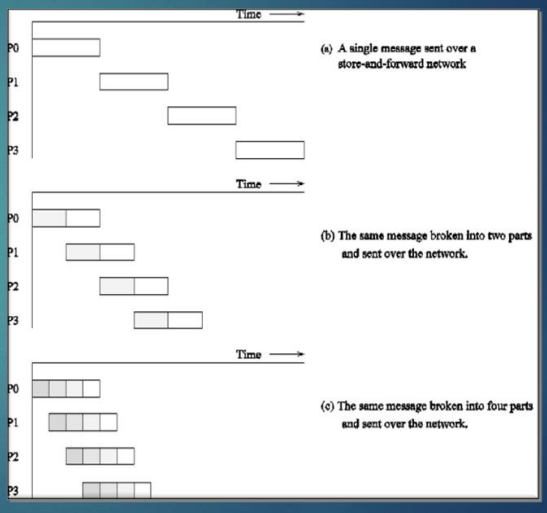
- This is identical to packet routing, however, t_w is typically much smaller.

Header of the message takes l* t_h to reach the destination and entire message arrives in time mt_w after the message header

Message Passing Costs in Parallel Computers

(a) through a store-and-forward communication network;

b) and (c) extending the concept to cut-through routing.



Shaded regions here represent the time where message is in transit (travel)

The startup time associated with this message transfer is assumed to be zero

---

🧠 Concept: Cut-Through Routing

Cut-through routing is an **improvement over packet routing** — it pushes the idea of *pipelining* even further ⚡

In **packet routing**, an entire packet must be received at each hop before forwarding starts.
But in **cut-through routing**, as soon as the **first few bits (called flits)** of the packet arrive, the router **immediately starts forwarding** them to the next hop — without waiting for the full packet.

💬 Definition

In cut-through routing, a message is divided into **flits (flow control digits)**, which are **very small pieces of data**.
As soon as the **header flit** arrives and decides the path, all following flits **stream through the same path** — like water flowing through a pipe 💧.

🧩 Key Points

Term	Meaning
Flit (Flow Control Digit)	Smallest unit of data in cut-through routing
Tracer Message	The first small message that programs all routers along the path
Fixed Path	All flits follow the same route (no sequence numbers needed)
Error Check	Done once for the entire message, not for each flit
Header Overhead	Very small (only one header for all flits)
Routing Decision	Made once when first flit arrives

⚙️ Step-by-Step Analogy

Imagine a water pipeline 🚰💧

- **Store-and-Forward:** Fill one tank → send full tank → next tank → wait.
- **Packet Routing:** Send small buckets continuously, but each has its own address label.

- **Cut-Through Routing:** Open a tap — as soon as water starts flowing, it flows continuously through all pipes.
  No need to stop or relabel every bucket.

That's **cut-through routing** — fast, continuous, minimal delay ⚡

🧮 Formula: Total Communication Time

Let:

- `ts` = startup time
- `th` = per-hop latency
- `tw` = per-word transfer time
- `m` = number of words
- `l` = number of hops

Then, just like packet routing:

T ≈ ts + l × th + m × tw

But note:

tw (cut-through) is much smaller than in packet routing
because flits are tiny, and there's **less overhead** (no separate headers, no per-packet error checks).

⚡ Why Cut-Through is Fast

✅ Routers **start forwarding immediately**
✅ All flits **use same route → no reordering needed**
✅ **Less waiting** → reduced total delay
✅ **Single error check** → less overhead

🧩 Comparison Between Routing Methods

Feature	Store-and-Forward	Packet Routing	Cut-Through Routing
Data Unit	Entire message	Packets	Flits (very small units)

Feature	Store-and-Forward	Packet Routing	Cut-Through Routing
Forwarding	After full message received	After full packet received	As soon as header flit arrives
Header Info	Once per message	Each packet has header	Single header for all flits
Path Selection	Fixed	May differ per packet	Same path for all flits
Error Check	Per message	Per packet	Per message
Latency	Highest	Medium	Lowest
Efficiency	Low	Better	Best

🌍 Real-World Analogy

Think of **video streaming** 🎥
When you watch a video on YouTube:

- Data arrives continuously — not the whole video at once.
- As soon as initial bits arrive, playback starts while more data keeps coming.
  This behavior is **similar** to cut-through routing (continuous flow, no waiting for full data chunks).

📝 Exam-Ready Summary

Symbol	Meaning
ts	Startup time (prepare + route setup)
th	Per-hop latency
tw	Per-word transfer time (very small in cut-through)
$T \approx ts + l \times th + m \times tw$	Total communication cost
Note:	tw is smaller than in packet routing

✅ **Cut-through routing = fastest, lowest latency, continuous streaming of flits.**

📝 Routing Techniques in Parallel Computers — Summary Sheet

Feature	Store-and-Forward Routing	Packet Routing	Cut-Through Routing
Basic Idea	Entire message received at each node before forwarding.	Message is divided into small packets pipelined through the network.	Message divided into very small units called *flits* (flow control digits) — forwarded immediately.
Working	Wait → Receive full message → Forward to next hop.	Start sending next packet while previous packets are still moving (pipelined).	As soon as header flit arrives, forwarding begins without waiting for the full packet.
Unit of Transmission	Full message	Packets	Flits
Header Overhead	Once per message	Each packet has its own header	One header for all flits (minimal)
Path Selection	Fixed	Each packet can take different paths	All flits take same pre-programmed path
Error Checking	Once per message	Done per packet	Done once per message
Sequence Numbers	Not needed	Required (since packets may arrive out of order)	Not needed (same path)
Utilization of Network	Poor (waiting at each hop)	Good (pipelined)	Excellent (continuous streaming)
Latency / Delay	Highest	Moderate	Lowest
Efficiency	Lowest	Better	Best
When Used	Simple, reliable systems	General-purpose networks (e.g. Internet, TCP/IP)	High-performance interconnects (supercomputers, GPUs, etc.)

🧮 Formulas for Total Communication Time

Routing Type	Formula	Notes
Store-and-Forward	**$T = ts + l \times (th + m \times tw)$**	Each hop waits for full message → slowest
Packet Routing	**$T \approx ts + l \times th + m \times tw$**	tw includes packet header overhead; pipelined
Cut-Through Routing	**$T \approx ts + l \times th + m \times tw$**	tw is much smaller (flits, minimal overhead)

📘 Symbol Meaning

Symbol	Meaning
ts	Startup time — preparing message, adding headers, establishing link
th	Per-hop (node) latency — delay per intermediate node
tw	Per-word transfer time — time to send one word (1/r, where r = bandwidth)
m	Message size (in words)
l	Number of hops (links)

🌍 Real-World Analogies

Type	Analogy
Store-and-Forward	Send a full parcel → next office sends it after receiving completely.
Packet Routing	Break parcel into smaller boxes → next truck starts as soon as first box leaves (pipelining).
Cut-Through Routing	Open a water tap — flow starts and continues through all pipes (continuous streaming).

✅ Quick Revision Tips

- **Store-and-Forward:** Waits → Slowest ⏳
- **Packet Routing:** Pipelined → Faster 🚀
- **Cut-Through Routing:** Continuous flow → Fastest ⚡
- **Formula same**, only **tw (per-word time)** changes — smallest in cut-through routing.

Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- The cost of communicating a message between two nodes l hops away using cut-through routing is given by

$$t_{comm} = t_s + lt_h + t_w m.$$

- In this expression, t_h is typically smaller than t_s and t_w. For this reason, the second term in the RHS does not show, particularly, when m is large.

- For these reasons, we can approximate the cost of message transfer by

$$t_{comm} = t_s + t_w m.$$

For communication using flits, start-up time dominates the node latencies.

Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- It is important to note that the original expression for communication time is valid for only uncongested networks.

- Different communication patterns congest different networks to varying extents.

- It is important to understand and account for this in the communication time accordingly.

◆ **Simplified Cost Model for Communicating Messages**

When a message is sent between two nodes that are `l` hops away (using **cut-through routing**), the **total communication cost** is given by:

$$
T_{comm} = t_s + l \times t_h + m \times t_w
$$

where:

- **ts** → Startup time
  (time for preparing message, headers, establishing connection, etc.)
- **th** → Per-hop latency
  (delay for each intermediate switch/router)
- **tw** → Per-word transfer time
  (depends on message size and link bandwidth)
- **l** → Number of hops (links between sender and receiver)
- **m** → Message size (in words)

🔶 Simplification (Approximation)

Usually,
`th` (per-hop latency) is **very small** compared to `ts` and `tw`.
So, for **large messages (m is large)**,
the effect of `th` becomes negligible.

Hence, we can simplify the equation as:

$$
T_{comm} \approx t_s + m \times t_w
$$

➡️ Meaning:
In large data transfers, **startup time and message size** dominate the total communication time.

🔶 Key Observations

1. **Flit-based (cut-through) communication**:
   - The network uses **flits (flow control digits)**.
   - **Startup time (ts)** dominates because once the message starts flowing, flits follow each other quickly.
2. **Assumption**:

- This model assumes the network is **uncongested** (no traffic or contention).

3. **In real systems**:
   - Network congestion can **increase communication time** significantly.
   - Different communication **patterns** (e.g., broadcast, scatter, all-to-all) cause **different levels of congestion**.

💡 Example (for better understanding)

Suppose:

- `ts = 10 µs`
- `th = 1 µs`
- `tw = 0.01 µs`
- `m = 1000 words`
- `l = 3 hops`

Then:
[
T_{comm} = 10 + (3)(1) + (1000)(0.01) = 10 + 3 + 10 = 23 \ µs
]

If `m` increases to 10,000 words:
[
T_{comm} = 10 + 3 + (10000)(0.01) = 10 + 3 + 100 = 113 \ µs
]

Here you can see — when message size increases, the `th` part (3 µs) becomes **insignificant** compared to the total.
