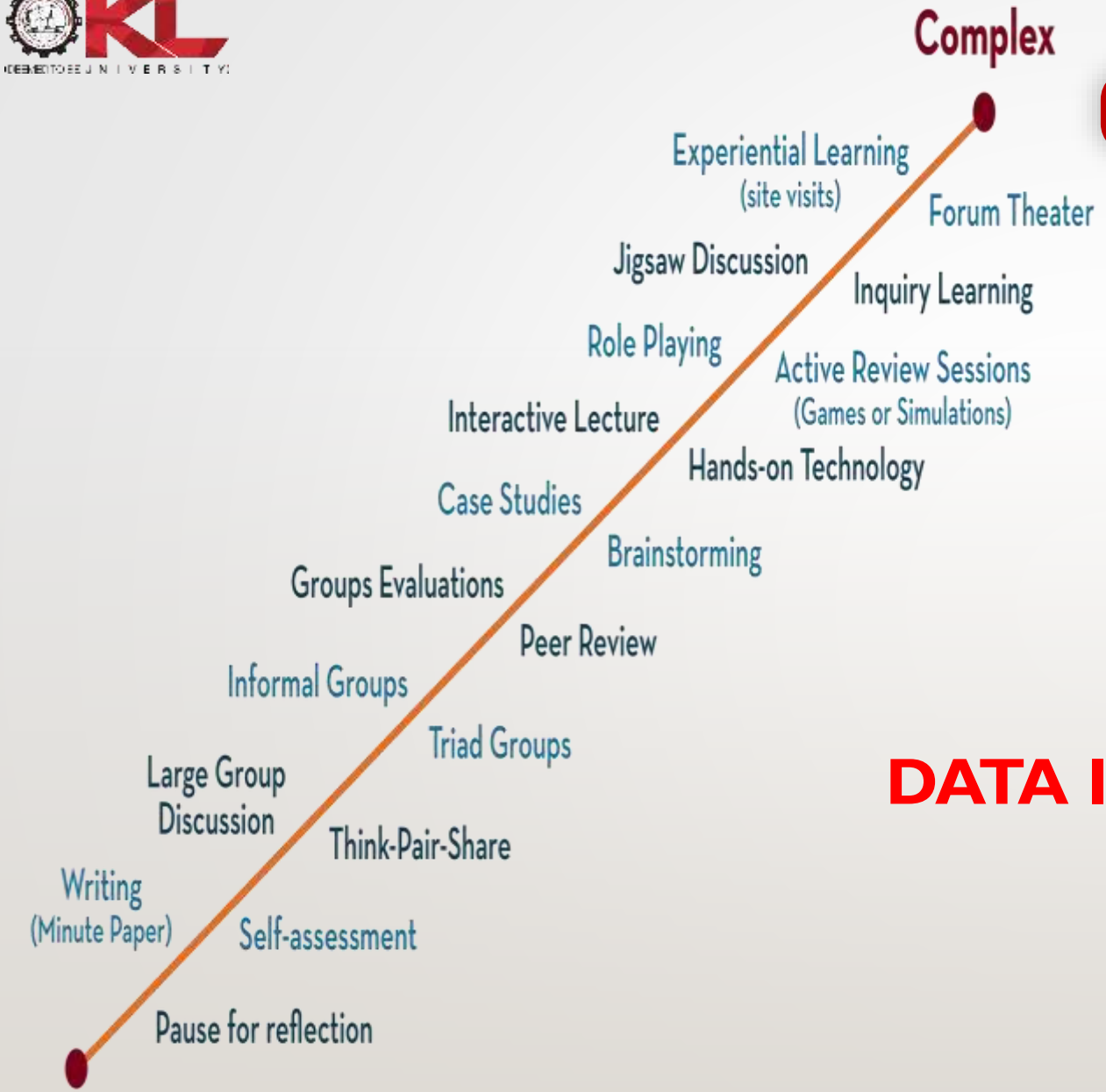


CO4

Session - 5

COURSE NAME: OPERATING SYSTEMS

COURSE CODE: 23CS2104R/A

DATA INTEGRITY AND PROTECTION, DISTRIBUTED SYSTEMS

AIM OF THE SESSION

To familiarize students with the basic concept of Data Integrity and Distributed Systems.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate what is meant by Data Integrity.
2. Demonstrate what is meant by Protection.
3. Describes the types of Protection Mechanisms.
4. Describe the Distributed Systems.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Defines what Data Integrity.
2. Describe Distributed Systems.
3. Summarize the Role of Data Integrity and Protection.

DISK FAILURE MODES

The overall precision, completeness, and continuity of data is known as data integrity.

LSEs arise when a disk sector (or group of sectors) has been damaged in some way

For example, if the disk head touches the surface for some reason (a **head crash**,

A disk block becomes **corrupt** in a way not detectable by the disk itself)

- Common and worthy of failures are the **frequency of latent-sector errors(LSEs)** and **block corruption**.

	Cheap	**Costly**
LSEs	9.40%	1.40%
Corruption	0.50%	0.05%

Frequency of LSEs and Block Corruption

DISK FAILURE MODES (CONT.)

- Frequency of latent-sector errors(LSEs)
  - Costly drives with more than one LSE are as likely to develop additional.
  - For most drives, annual error rate increases in year two
  - LSEs increase with disk size
  - Most disks with LSEs have less than 50
  - Disks with LSEs are more likely to develop additional LSEs
  - There exists a significant amount of spatial and temporal locality
  - Disk scrubbing is useful (most LSEs were found this way)

DISK FAILURE MODES (CONT.)

- Block corruption:
  - Chance of corruption varies greatly across different drive models
  - Within the same drive class
  - Age effects are different across models
  - Workload and disk size have little impact on corruption
  - Most disks with corruption only have a few corruptions
  - Corruption is not independent with a disk or across disks in RAID
  - There exists spatial locality, and some temporal locality
  - There is a weak correlation with LSEs

HANDLING LATENT SECTOR ERRORS

- Latent sector errors are easily detected and handled.
- Using **redundancy mechanisms**:
  - In a mirrored RAID or RAID-4 and RAID-5 system based on parity, the system should reconstruct the block from the other blocks in the parity group.

DETECTING CORRUPTION: THE CHECKSUM

- How can a client tell that a block has gone bad?
- Using **Checksum mechanisms:**
  - This is simply the result of a function that takes a chunk of data as input and computes a function over said data, producing a small summary of the contents of the data.

COMMON CHECKSUM FUNCTIONS (CONT.)

- Different functions are used to compute checksums and vary in strength.
  - One simple checksum function that some use is based on **exclusive or(XOR).**

```
365e c4cd ba14 8a92 ecef 2c3a 40be f666
```

If we view them in binary, we get the following:

```
0011 0110 0101 1110   1100 0100 1100 1101
1011 1010 0001 0100   1000 1010 1001 0010
1110 1100 1110 1111   0010 1100 0011 1010
0100 0000 1011 1110   1111 0110 0110 0110
```

It is easy to see what the resulting checksum will be:

```
0010 0000 0001 1011   1001 0100 0000 0011
```

The result, in hex, is 0x201b9403.

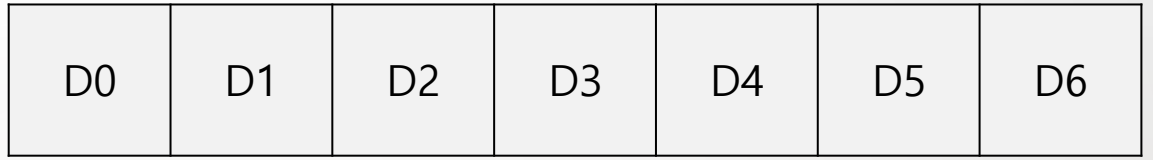
  - XOR is a reasonable checksum but has its limitations.
    - Two bits in the same position within each check summed unit changed the checksum will not detect the corruption.

COMMON CHECKSUM FUNCTIONS (CONT.)

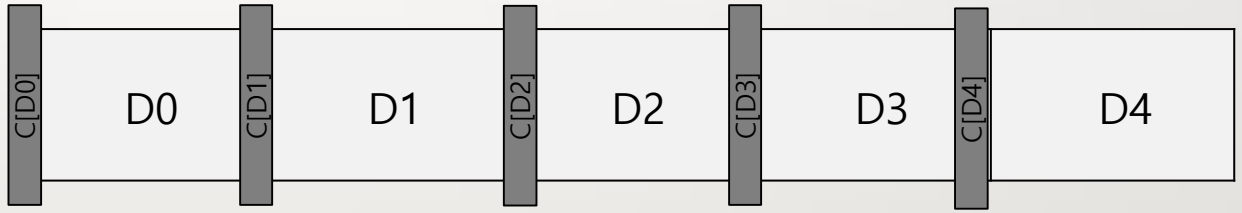
- Addition Checksum
  - This approach has the advantage of being fast.
  - Compute 2's complement addition over each chunk of the data
    - ignoring overflow
- Fletcher Checksum
  - Compute two check bytes, s1 and s2.
    - Assuming a block D consists of bytes d1…dn; s1 is simply in turn is
      - s1 = s1 + di mod 255(compute overall di);
      - s2 = s2 + s1 mod 255(again over all di);
- Cyclic redundancy check(CRC)
  - Treating D as a large binary number and dividing it by an agreed-upon value.
    - The remainder of this division is the value of the CRC.

CHECKSUM LAYOUT

- The disk layout without checksum:



- The disk layout **with checksum:**



- Store the checksums packed into 512-byte blocks.

USING CHECKSUMS

- When reading a block D, the client reads its checksum from disk

  Cs(D), **stored checksum.**

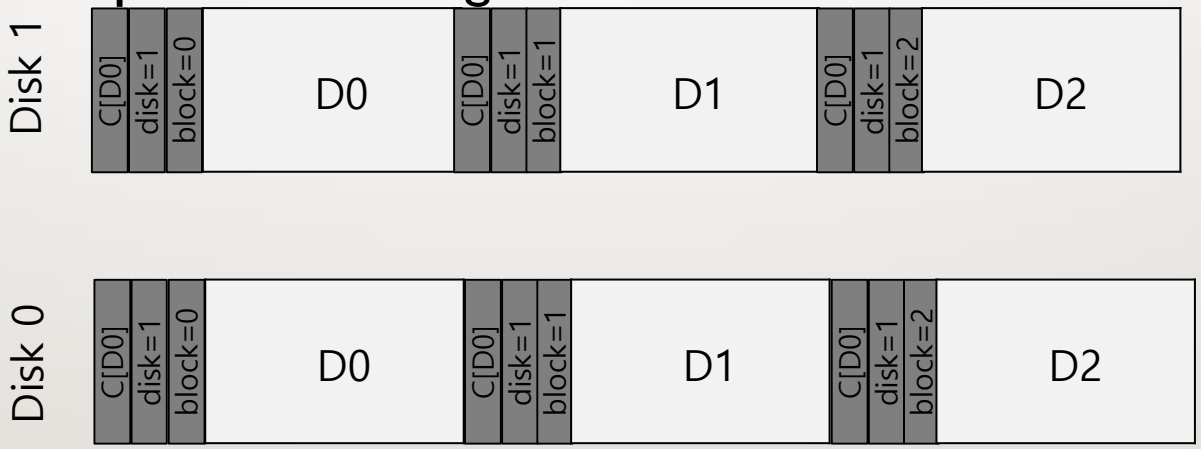
- Computes the checksum over the retrieved block D

  computed checksum Cc(D).

- Compares the stored and computed checksums;
  - If they are equal (Cs(D) == Cc(D)), the data is in safe.
  - If they do not match (Cs(D) != Cc(D)), the data has changed since the time it was stored (since the stored checksum reflects the value of the data at that time).

A NEW PROBLEM: MISDIRECTED WRITES

- Modern disks have a couple of unusual failure modes that require different solutions.
  - Misdirected write arises in disk and RAID controllers which the data to disk correctly, except in the *wrong* location



ONE LAST PROBLEM: LOST WRITES

- some modern storage devices also have an issue known as a lost write, which occurs when the device informs the upper layer that a write has completed but in fact, it never is persisted; thus, what remains is the old contents of the block rather than the updated new contents.
- There are several possible solutions:
  - One classic approach is to perform a write verify or read-after-write; by immediately reading back the data after a write
  - A system can ensure that the data indeed reached the disk surface.
  - This approach, however, is quite slow,.
  - Doubling the number of I/Os needed to complete a write.

SCRUBBING

- When do these checksums get checked?
  - Most data is rarely accessed and thus remains unchecked.

- To remedy this problem, many systems utilize **disk scrubbing**.
  - By **periodically reading** through every block of the system
  - Checking whether checksums are still valid
  - Reduce the chances that all copies of certain data become corrupted

OVERHEAD OF CHECK SUMMING

- Two distinct kinds of overheads: **space** and **time**
- Space overheads
  - **Disk itself**: A typical ratio might be an 8-byte checksum per 4KB data block, for a 0.19% on-disk space overhead.
  - **Memory of the system**: This overhead is short-lived and not much of a concern.
- Time overheads
  - CPU must compute the checksum over each block
    - To reduce CPU overheads, combine data copying and checksumming into one streamlined activity.

DISTRIBUTED SYSTEMS

- A distributed system is a network that consists of autonomous computers that are connected using a distribution middleware.
  - Distribution Middleware help in sharing different resources and capabilities to provide users with a single and integrated coherent network.

Example

- **Client/server** distributed system (Web browser connects to a web server).
- Connecting to Google or Facebook.
  - Not just interacting with a single machine, however; behind the scenes, these complex services are built from a large collection (i.e., thousands) of machines, each of which cooperate to provide the particular service of the site.

DISTRIBUTED SYSTEMS

Challenges

- Complexity – Thousands of machines
- Failure (machines, disks, networks, and software all fail from time to time)
- Performance(reduce the number of messages to exchange)
- Security
- Communication

COMMUNICATION

Unreliable Communication Layers

-UDP

Reliable Communication

- TCP/IP

PACKET LOSS

- Packets are regularly lost, corrupted, or otherwise do not reach their destination.
  - During transmission, some bits get flipped due to electrical or other similar problems.
  - Network link or packet router or even the remote host, are damaged or otherwise not working correctly;
  - Network cables do accidentally get severed, at least sometimes.

USER DATAGRAM PROTOCOL(UDP)

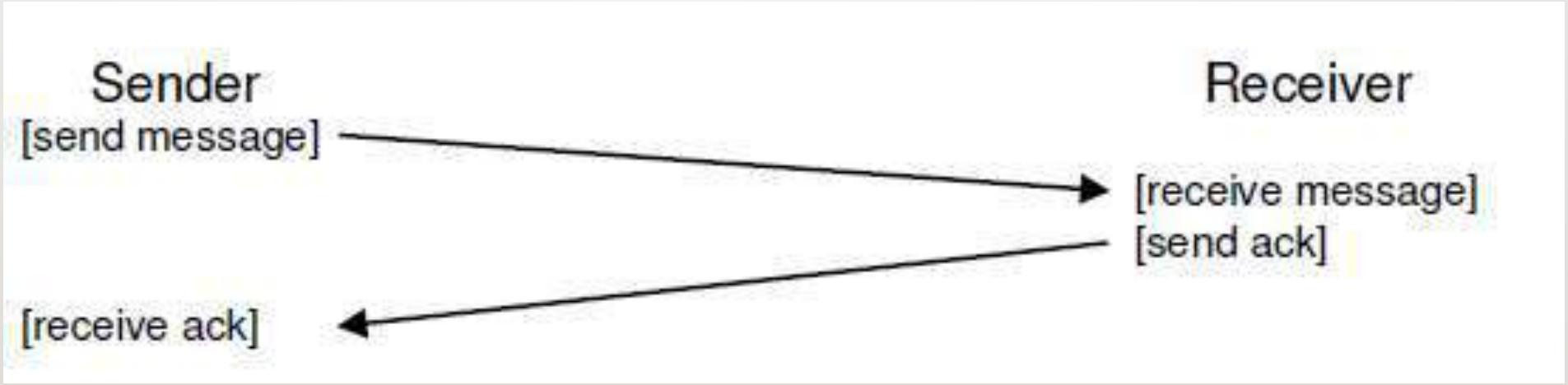
How to deal with packet loss in an unreliable communication layer?

- A process uses the **sockets** API to create a **communication endpoint**;
- Processes on other machines send UDP **datagrams** (a datagram is a fixed-sized message up to some max size)to the original process.
- UDP includes a **checksum** to detect some forms of packet corruption.

TRANSMISSION CONTROL PROTOCOL/ INTERNET PROTOCOL (TCP/IP)

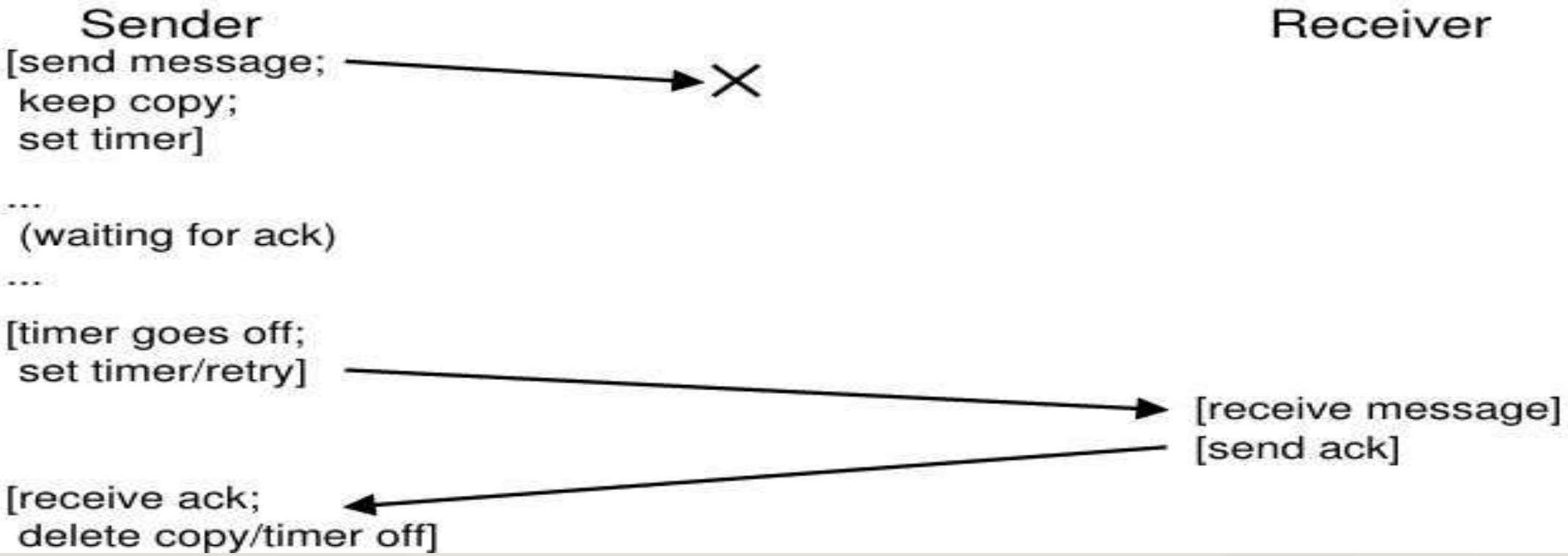
Techniques to handle packet loss

- **Acknowledgment**, or **ack** for short.
  - The idea is simple: the sender sends a message to the receiver;
  - The receiver then sends a short message back to acknowledge its receipt



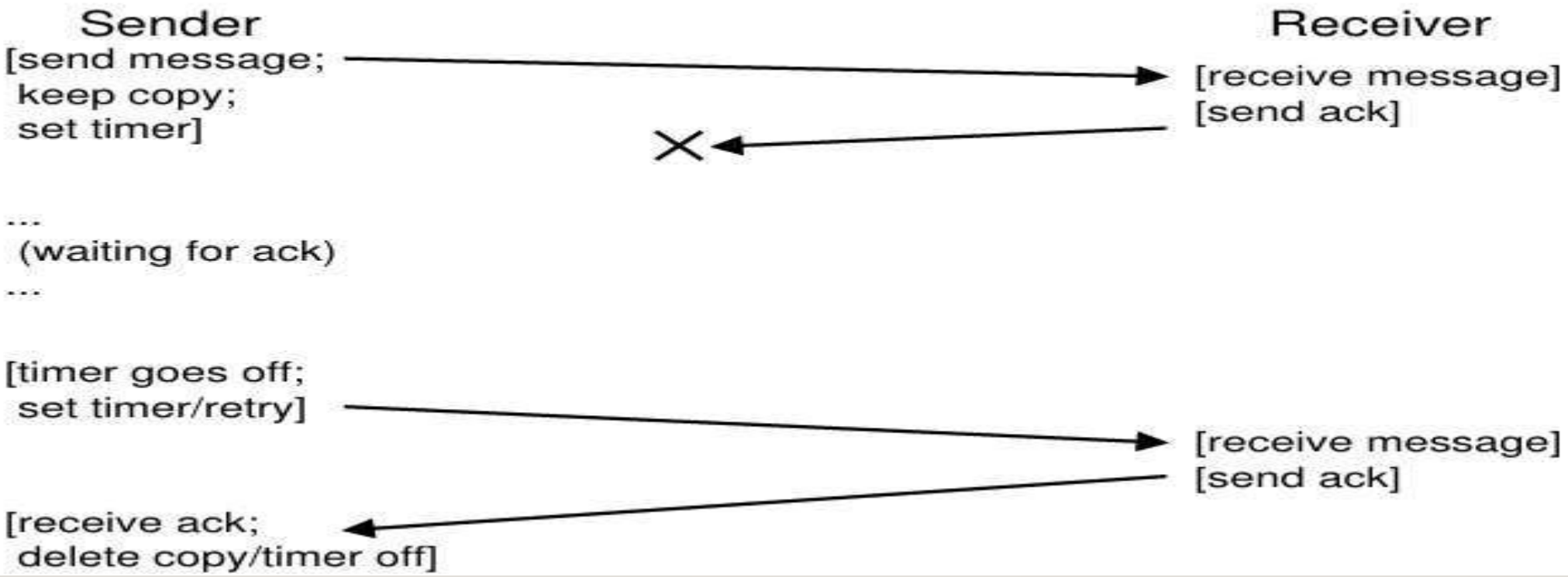
Message Plus Acknowledgment

DROPPED REQUEST



Message Plus Acknowledgment: Dropped Request

DROPPED REPLY



Message Plus Acknowledgment: Dropped Reply

COMMUNICATION ABSTRACTIONS

- **Distributed Shared Memory -** distributed shared memory (DSM) systems enable processes on different machines to share. This abstraction turns a distributed computation into something that looks like a multi-threaded application; the only difference is that these threads run on different machines instead of different processors within the same machine a large, virtual address space
- **Remote Procedure Call (RPC) -** Remote procedure call packages all have a simple goal: to make the process of executing code on a remote machine as simple and straightforward as calling a local function.

DISTRIBUTED SHARED MEMORY(DSM)

- Systems enable processes on different machines to share a large, virtual address space.
- When a page is accessed on one machine, two things can happen.
  - In the first (best) case, the page is already local on the machine, and thus the data is fetched quickly.
  - In the second case, the page is currently on some other machine. A page fault occurs, and the page fault handler sends a message to some other machine to fetch the page, install it in the page table of the requesting process, and continue execution.
- This approach is not widely in use today for a number of reasons.

DISTRIBUTED SHARED MEMORY(DSM)

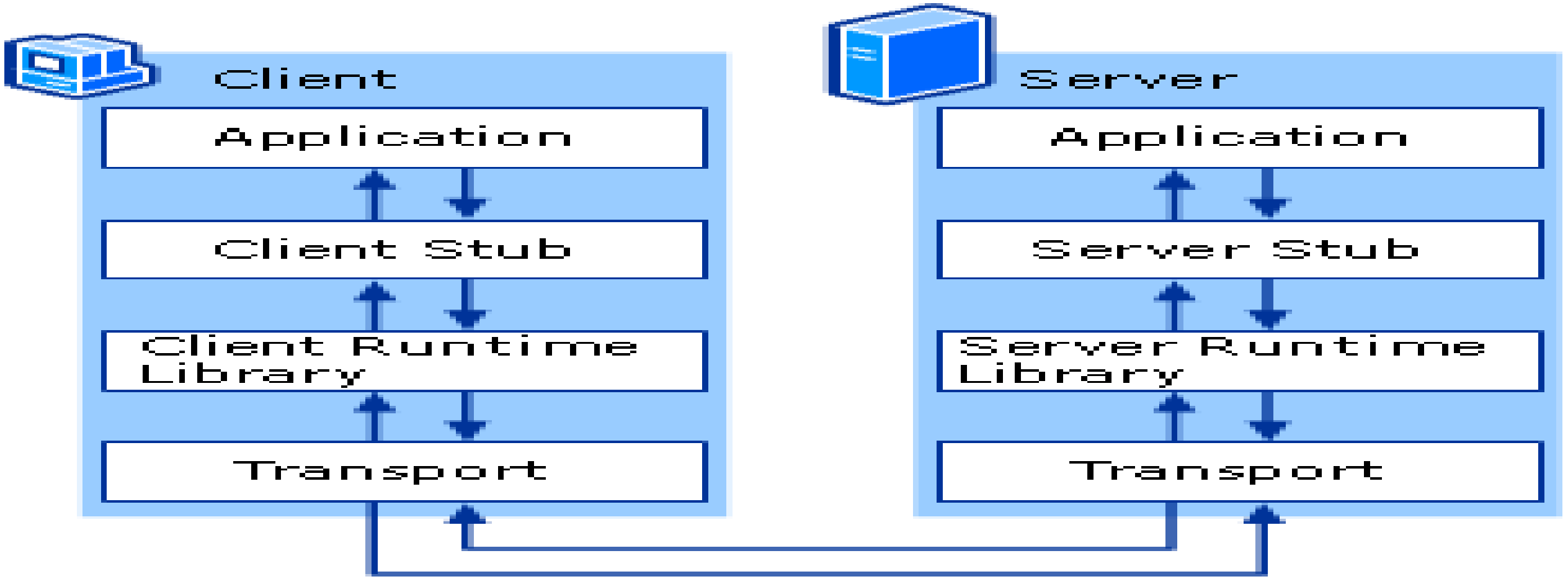
Problems

- Failure
  - If a machine fails
  - what happens to the pages on that machine?
  - what if the data structures of the distributed computation are spread across the entire address space?
- Performance.
  - Page faults
  - Fetches from remote machines.

REMOTE PROCEDURE CALL(RPC)

- Simple goal: To make the process of executing code on a remote machine as simple as calling a local function.
  - To a client, a procedure call is made, and sometime later, the results are returned.
  - The server simply defines some routines that it wishes to export.
  - The rest of the magic is handled by the RPC system
    - **Stub generator** (sometimes called a **protocol compiler**), and
    - **Run-time library**.

RPC


Client
Application
Client Stub
Client Runtime Library
Transport
Server
Application
Server Stub
Server Runtime Library
Transport

FUNCTIONS IN THE STUB

1. Create a message buffer
2. Pack the needed information into the message buffer(**marshaling** of arguments or the **serialization** of the message)
3. Send the message to the destination RPC server
4. Wait for reply
5. Unpack return code and other arguments(**unmarshaling** or **deserialization**.)
6. Return to caller

SERVER ACTIONS

1. Unpack the message (un marshaling)
2. Call into the actual function
3. Package the results
4. Send the reply

RUNTIME LIBRARY

- The run-time library handles Performance and Reliability issues
- How to locate a remote service. (Problem of **naming) -** hostnames and port numbers
  - The client must know the hostname or IP address of the machine running the desired RPC service, as well as the port number it is using (a port number is just a way of identifying a particular communication activity taking place on a machine, allowing multiple communication channels at once).
- Which transport-level protocol should RPC be built upon.
  - TCP/IP or UDP?

TCP/IP OR UDP IN RPC

- In TCP/IP -
  - When the client sends an RPC request to the server, the server responds with an acknowledgment so that the caller knows the request was received.
  - Similarly, when the server sends the reply to the client, the client acks it so that the server knows it was received.
  - By building a request/response protocol (such as RPC) on top of a reliable communication layer, two "extra" messages are sent.
  - For this reason, many RPC packages are built on top of unreliable communication layers, such as UDP.
- Doing so enables a more efficient RPC layer, but does add the responsibility of providing reliability to the RPC system.
  - The RPC layer achieves the desired level of responsibility by using timeout/retry and acknowledgments.

THANK YOU

Team - Operating System