Sistem terdistribusi 3

Interprocess Communication

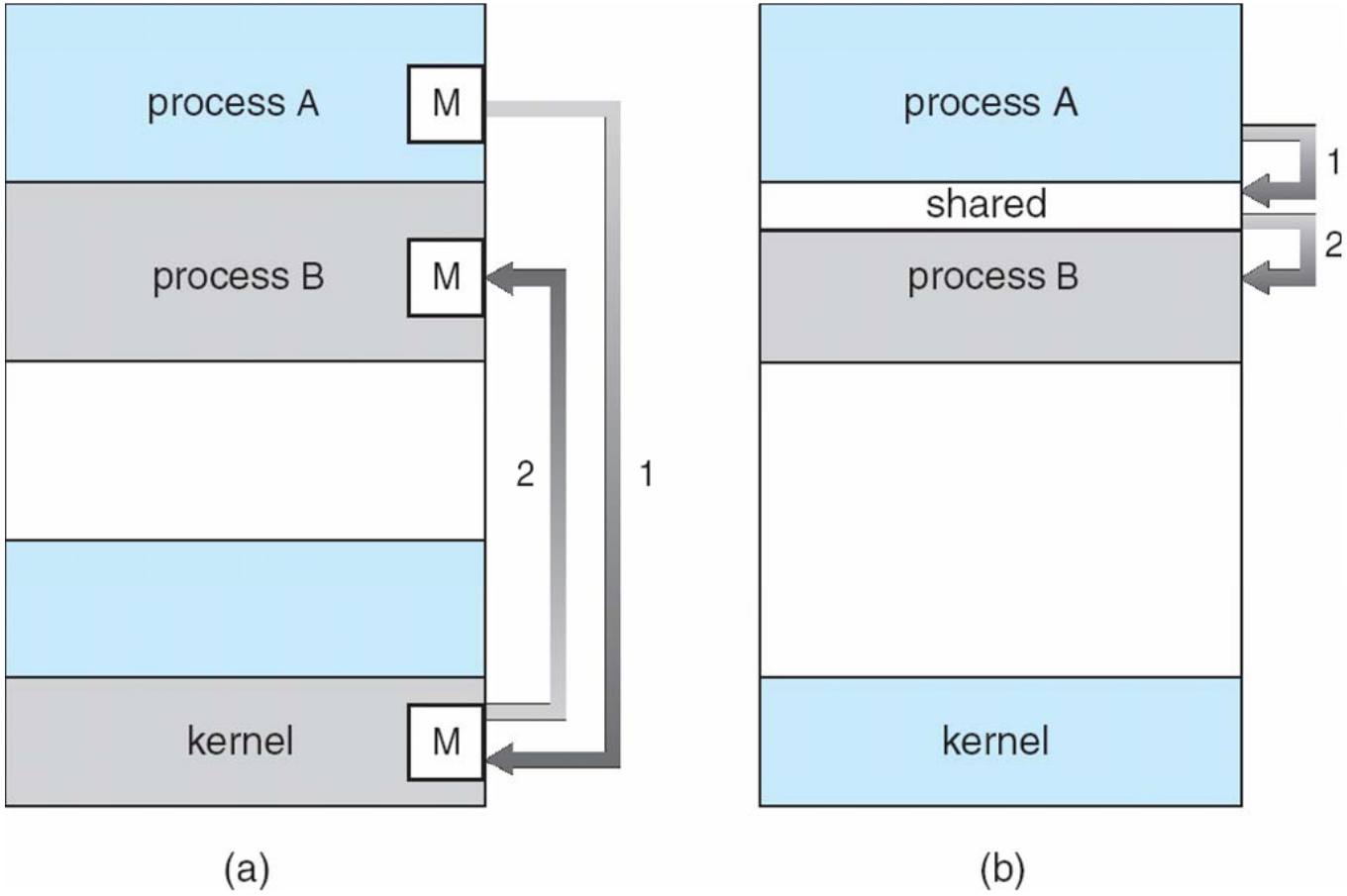
Prinsip berkomunikasi

- **Source**
  - generates data (text/binary) to be transmitted
- **Transmitter**
  - Converts data into transmittable signals
- **Transmission System**
  - Carries data
    - Ex: TCP/IP
- **Receiver**
  - Converts received signal into data
- **Destination**
  - Takes incoming data

Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
  - Process on different computer?
- Reasons for **cooperating** processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

Communications in local



Karakteristik IPC

- Synchronization dan Asynchronization of messages.
- Message destination : on Internet, using address dan local port.
  - http://192.168.1.2:8081
- Reliability data: validity dan integrity.
- Ordering data: urutan paket yang dikirim
- Different data processing: marshalling/unmarshalling

Kategori IPC

- **Pipes** : merupakan fasilitas yang menyediakan komunikasi **satu arah** antar proses dalam sebuah sistem atau disebut **half-duplex**, yaitu data mengalir hanya terjadi satu arah. (local computer)
- **FIFO** : fasilitas komunikasi secara FIFO (first in first out).
    - Mirip dengan Pipes
    - Message diantrikan
    - Terjadi di local computer
- **Shared memory** : suatu proses berbagi ruang dalam virtual address, sehingga proses manapun akan berbagi wilayah memory akan mampu menulis dan membacanya.
    - Dalam single processing (local computer)

Kategori IPC

- **Mapped memory** : berhubungan dengan mapping sebuah file dalam file system sesuai dengan memory yang ada.
  - Konsep virtual memory
  - Local computer
- **Message Queues** :mengirim pesan secara asynchronous.
  - Asynchronous berarti proses pengiriman data berlanjut disertai sebuah eksekusi tanpa harus menunggu penerima menerima atau mengenal informasi tersebut.

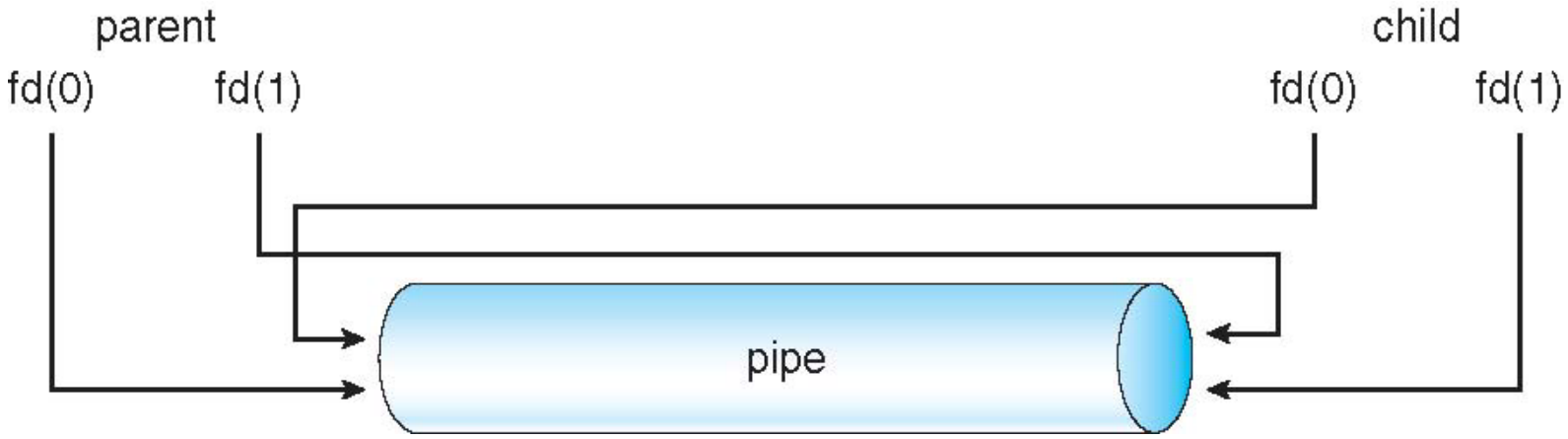
Kategori IPC

- **Semaphore** : struktur data yang di share ke beberapa proses untuk **sinkronisasi**
- **Socket** : sebagai endpoint dari komunikasi dua proses pada dua buah sistem komputer.
- **RPC (Remote Procedure Call)**: adalah sebuah protokol yang memungkinkan program komputer berjalan pada satu host dan mengakibatkan kode dapat dieksekusi pada host yang lain tanpa kebutuhan koneksi secara eksplisit.

Ordinary Pipes

- **Ordinary Pipes** allow communication in standard parent-child style
- Producer (parent) **writes** to one end (the *write-end* of the pipe)
- Consumer (child) **reads** from the other end (the *read-end* of the pipe)
- **Require** parent-child relationship between communicating processes
- A parent program opens **anonymous** pipes, and creates a new process and communicate using these pipes, or creates several new processes and arranges them

Pipe



Using FIFO as queue scheme

Example in UNIX **ls -l | less**

Named Pipes

- Named Pipes are more **powerful** than ordinary pipes
- A *named pipe* is **one-way** or **duplex** pipe for communication between the pipe server and **one** or **more** pipe clients.
- All instances of a named pipe **share** the same pipe name, but each instance has **its own buffers**
- Multiple pipe clients can use the same named pipe **simultaneously (system-persistent)**
- **No** parent-child relationship is necessary between the communicating processes
- Provided on both UNIX and Windows systems

Named Pipe In UNIX

For example, one can create a pipe and set up gzip to compress things piped to it:

```
mkfifo my_pipe
gzip -9 -c < my_pipe > out.gz
```

In a separate process shell, independently, one could send the data to be compressed:

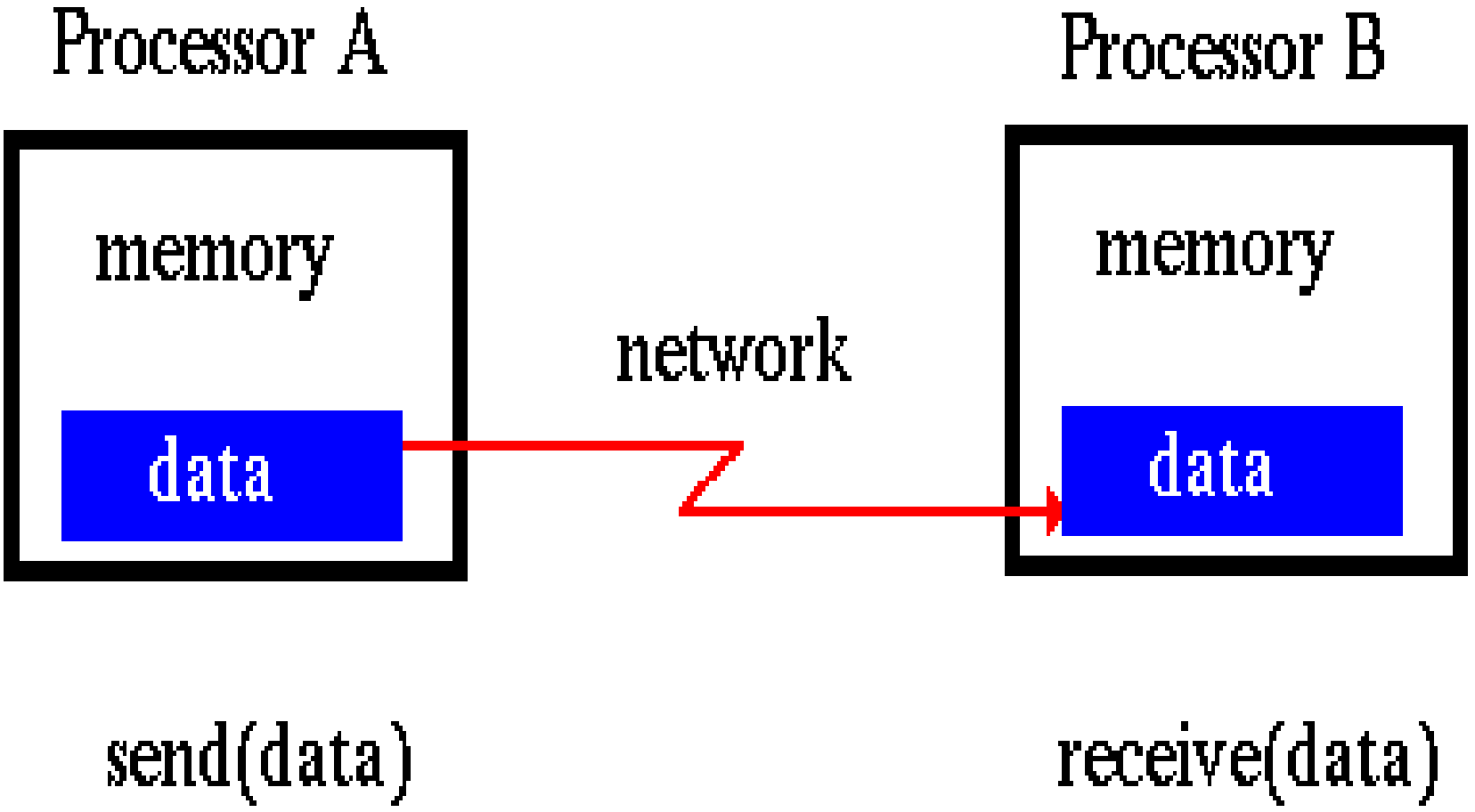
```
cat file > my_pipe
```

The named pipe can be deleted just like any file:

```
rm my_pipe
```

Message Passing

- Maybe provide latency, incompatibilities



Message Passing

- Basic Operations
  - Send > kirim message
  - Receive > terima message
- Variations
  - Connection-oriented vs Connectionless
  - Buffered vs Unbuffered
  - Reliable vs Unreliable
- Data representation
  - Marshalling & unmarshalling > pemformatan message

Communication Link

- Properties of communication link
  - **Links** are established automatically
  - A link usually is associated with exactly **one** pair of communicating processes
  - Between each pair there exists exactly **one** link
  - The link may be **unidirectional**, but is usually **bi-directional**

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of direct communication
  - **Links** are established automatically
  - A link is associated with exactly **one** pair of communicating processes
  - Between each pair there exists exactly **one** link
  - The link may be **unidirectional**, but is usually **bi-directional**

Indirect Communication

- Messages are directed and received from **ports**
  - Each port has a **unique id**
    - Well known ports: 0-1023
    - Recommended : > 1024 - 65535
  - Processes can communicate only if they **share a port**
- Properties of indirect communication link
  - Link established only if processes **share** a common port
  - A link may be associated with **many** processes
  - Each pair of processes may share **several** communication links
  - Link may be **unidirectional** or **bi-directional**

Indirect Communication

- Operations
  - create a new/open port
  - send and receive messages through port
  - destroy a port
- Primitives are defined as:
  - **send**(*A, message*) – send a message to **port** A
  - **receive**(*A, message*) – receive a message from **port** A

Indirect Communication

- Port sharing
  - P_1, P_2, and P_3 share **port** A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most **two** processes
  - Allow **only one** process at a time to execute a receive operation
  - Allow **all** processes receive the message
  - Allow the system to **select** arbitrarily the receiver.
    - Sender is **notified** who the receiver was.

Model comm: Syn & Asyn

- **Synchronous** communication
  - Acknowledge must be received
  - Blocking communication
  - Sender/Recipient must both active
  - Example: Registration system / Chatting
- **Asynchronous** communication
  - No acknowledge needed
  - Non-blocking communication
  - Message may be queued
  - Example: Email / Instant Messenger

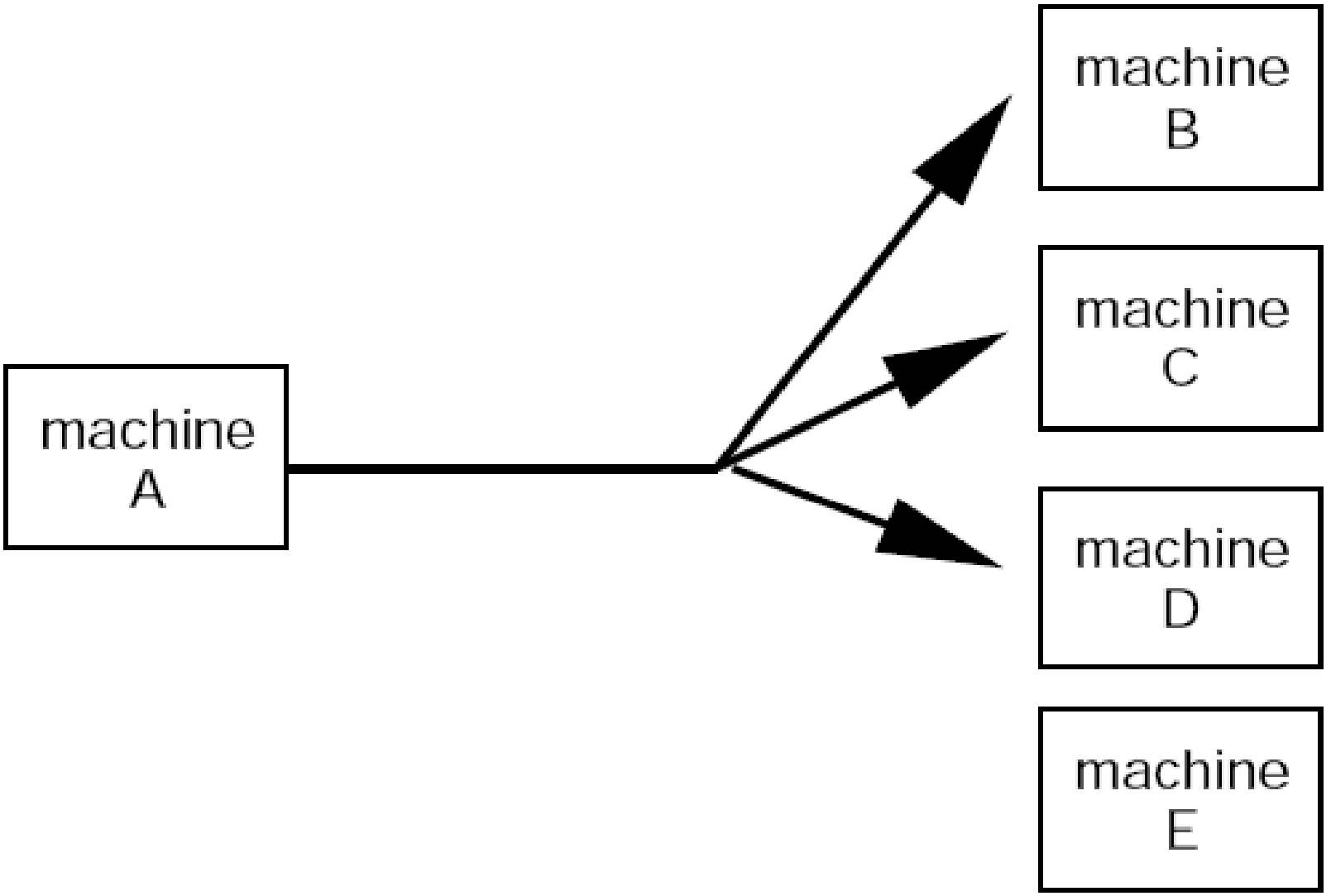
Transient & Persistent Comm.

- Transient Communication
  - Message **discarded** if failed to delivered immediately
  - Example: HTTP Request
- Persistent Communication
  - Message **stored** until receiver can accept it
  - Example: Email, Messenger

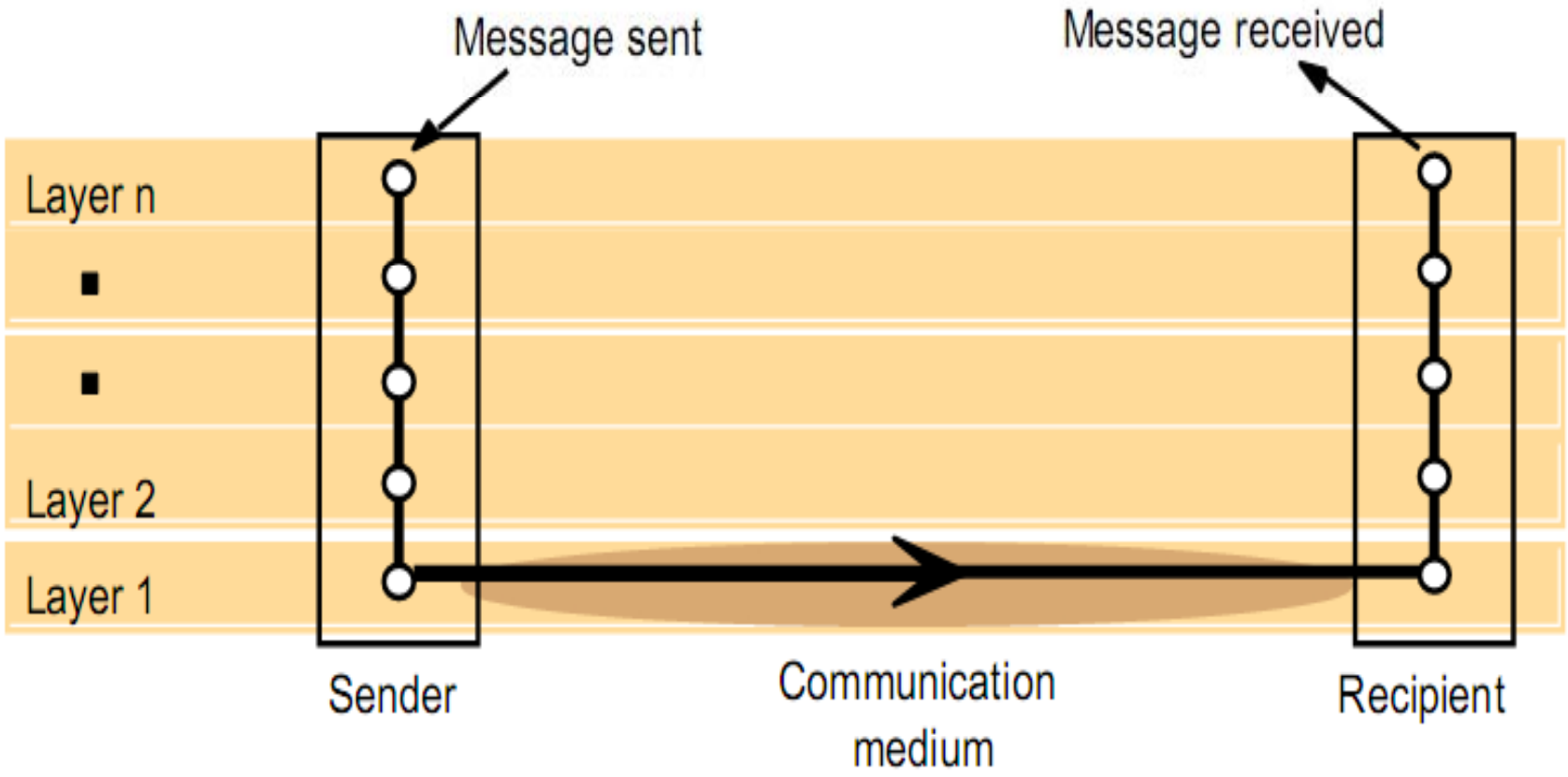
Group Communications

- **Multicast**: sent to specific group
  - Sender tidak tahu identitas penerima dan berapa yang menerima
- **Broadcast**: sent to everyone
- Used for
  - Replication of services/data
  - Service discovery
  - Event notification

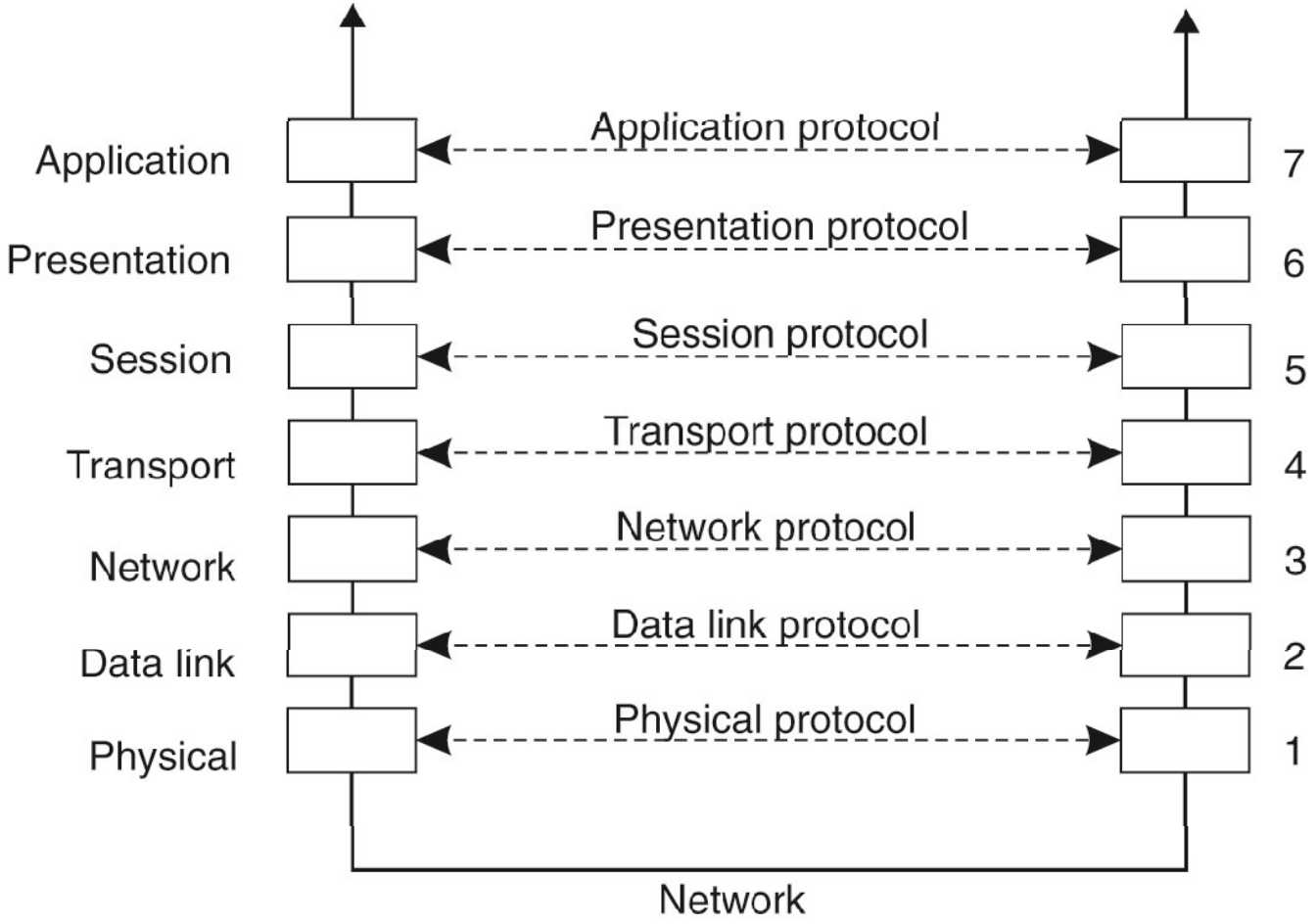
Group Communications



Komunikasi Jaringan



Remember: OSI



OSI Layers (1)

- Physical
  - Physical interface between devices
    - Mechanical
    - Electrical
    - Functional
    - Procedural
  - Contoh: Ethernet CARD
- Data Link
  - Means of activating, maintaining and deactivating a reliable link
  - Error detection
  - Contoh: PPP, Router

OSI Layers (2)

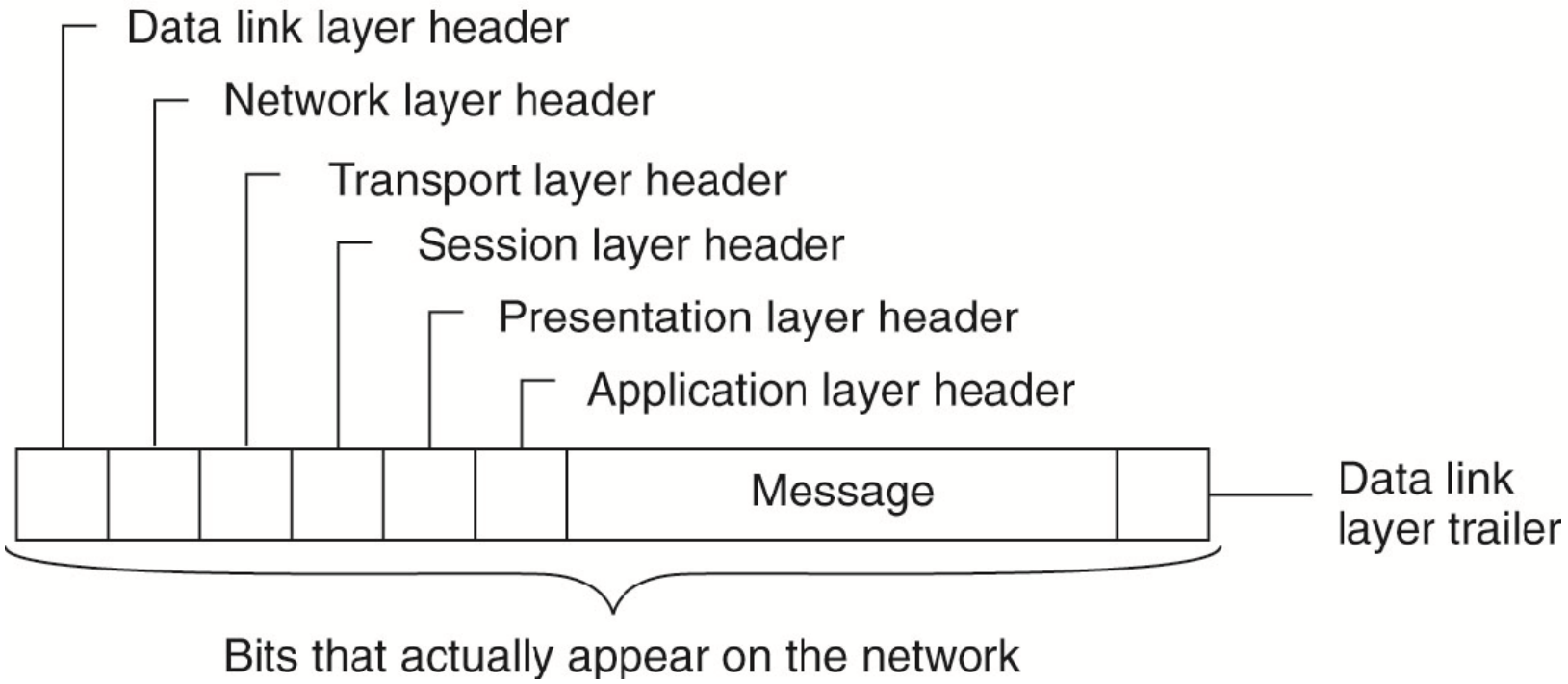
- Network
  - Transport of information
  - Contoh: Virtual Circuit & Internet Protocol

- Transport
  - Exchange of data between end systems
  - Error free
  - In sequence / No sequence
  - No losses / losses
  - No duplicates / duplicates
  - Quality of service
  - Contoh: TCP dan UDP

OSI Layers (3)

- Session
  - Control of dialogues between applications
  - Recovery
- Presentation
  - Data formats and coding
  - Data compression
  - Encryption
  - Contoh: SSL
- Application
  - Means for applications to access OSI environment
  - Contoh: HTTP, FTP, SMTP

The Message


Data link layer header
Network layer header
Transport layer header
Session layer header
Presentation layer header
Application layer header
Message
Data link layer trailer
Bits that actually appear on the network

Konsep Pengiriman Data

- Data dikirim dalam bentuk **paket**
- Setiap paket memiliki **header** untuk keperluan administrasi routing
- Data disimpan dalam **body** sebuah paket
- Ukuran paket sangat bervariasi
  - Ethernet: 64 – 1518 byte
- Bisa dikirimkan dengan **TCP/UDP**

TCP dan UDP

- Dua protokol pada **transport layer**
- Menggunakan konsep port (16 bit) untuk membedakan aplikasi
  - HTTP: 80, HTTPS: 443, FTP: 21, ...
  - Port 1-1023 : well-known port
  - Port 1024-49151 : registered port
  - Port lain bisa digunakan secara bebas
  - Di Linux: /etc/services

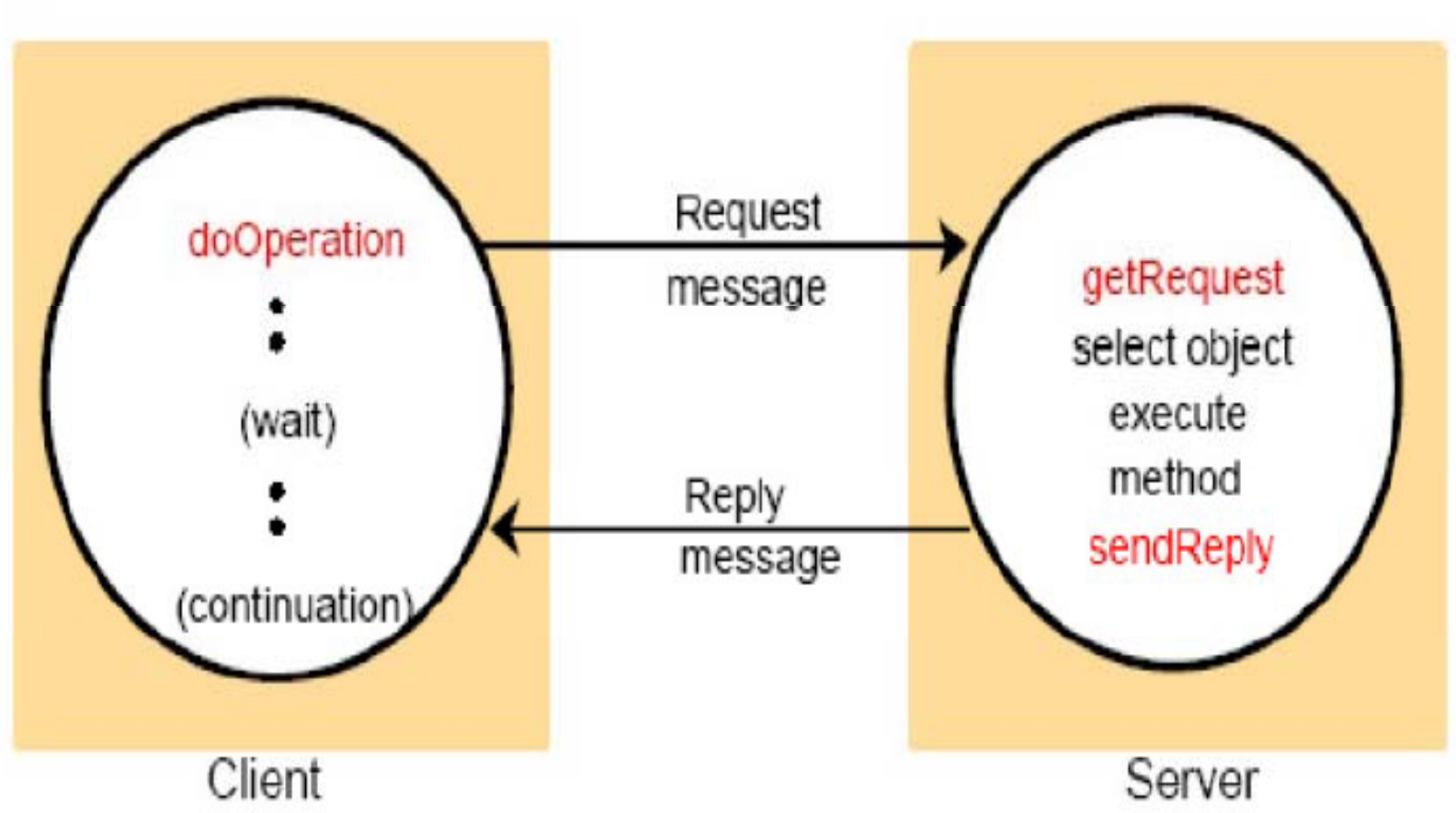
TCP

- Transmission Control Protocol, pada transport layer
  - Reliable connection
- Adanya pengecekan error
- Dijaga urutan message
- Komunikasi duplex – dua arah
- Segmentasi - TCP PDU
  - Called TCP segment
  - Includes source and destination port
    - Identify applications
    - Connection refers to pair of ports
- TCP tracks segments between entities on each connection

UDP

- User Datagram Protocol
- Not guaranteed delivery
- No preservation of sequence
- No protection against duplication
- Minimum overhead
- Adds port addressing to IP
- Contoh: DNS, streaming

Request / Reply



Review Failure model

- **Process failure**: crash
  - Deteksi dgn timeout
- **Communication failure**: message drop
  - Karena: transmission error, buffer overflow
- **Arbitrary failure**: proses melewatkan step yang harus dilakukan atau membawa data yg salah
  - Data korup, data double

Mekanisme handle failure Model of IPC

- **Timeout**, jika tidak dapat balasan, method **doOperation** akan mengirim terus request message sampai timeout.
- **Duplicate request message**, server menerima lebih dari sekali request message sehingga memprosesnya berulangkali.
  - solusi : request identifier & filter out duplicate.

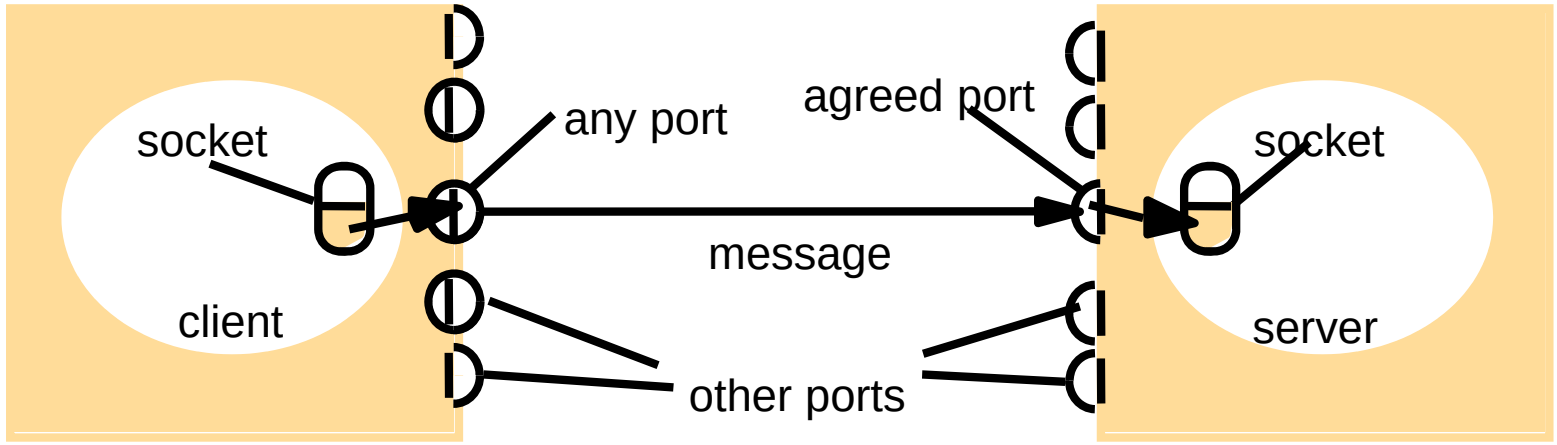
Failure Model of IPC

- **Lost reply message**,
  - server dapat **menyimpan** hasil proses request message, jika ada request message yang sama tidak perlu diproses ulang, server mengirim reply message berupa hasil proses dari request message yang telah disimpan.
  - **History**, server menyimpan struktur rekaman reply message yang telah dikirim.

Socket

- Menyediakan **jembatan** komunikasi antar proses
- Komunikasi antar proses: mengirimkan pesan antar socket pada satu proses menuju sebuah socket pada proses lain
- Bisa menggunakan TCP/UDP
  - Connection oriented dan connectionless oriented
- Melakukan **binding** ke sebuah port tertentu

Sockets and ports



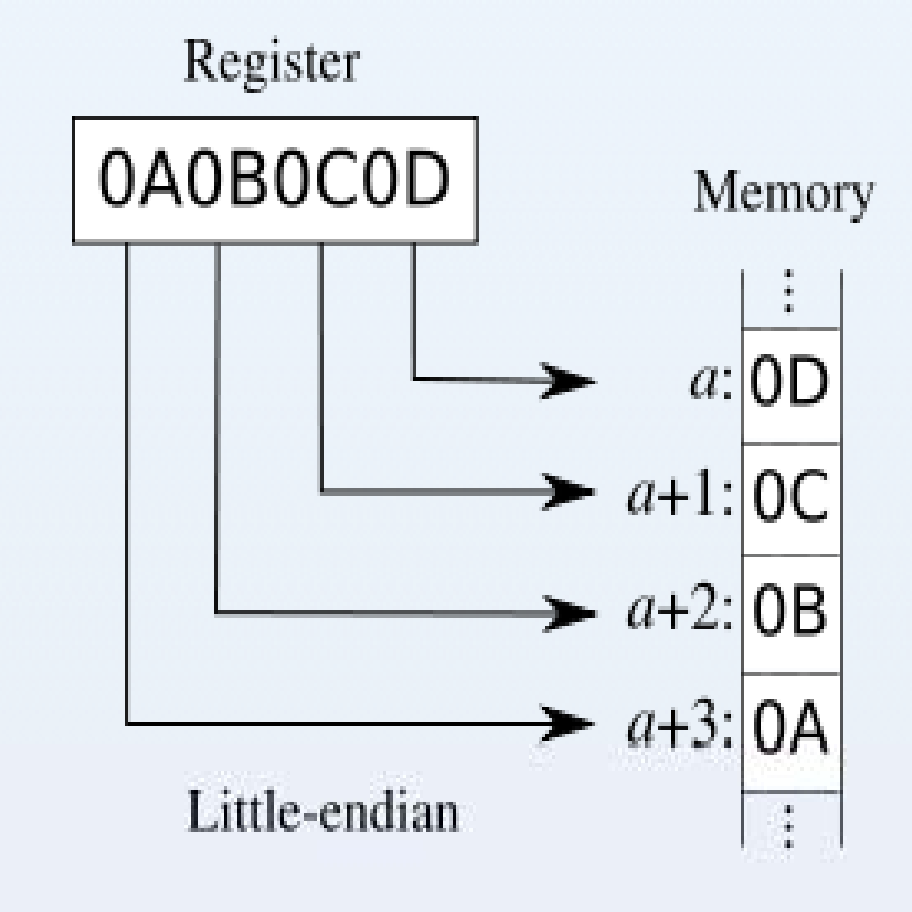
Operasi Socket

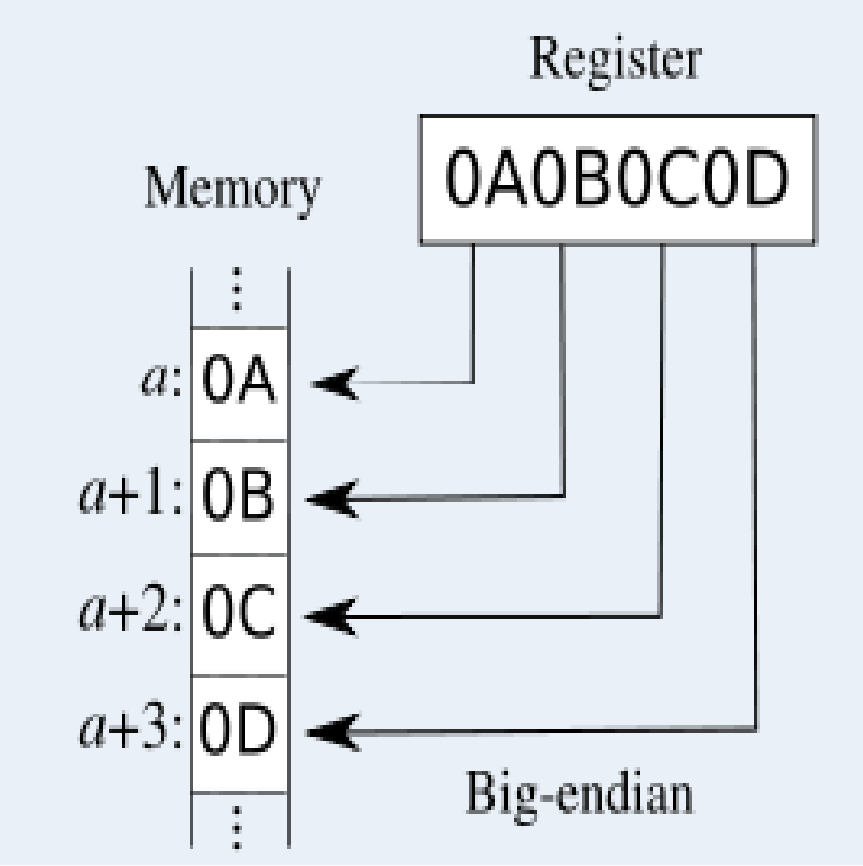
- Socket dapat melakukan operasi:
  - Koneksi ke mesin remote
  - Mengirim data
  - Menerima data
  - Mentutup koneksi
  - Bind to a port
- Di tiap mesin yang saling berinterkoneksi, **harus** terpasang socket.
  - Hard coded

Masalah Socket

- Informasi pada **program** : struktur data
- Informasi pada **message** : urutan byte
- Data harus **dikonversi** sebelum dan sesudah pengiriman agar bisa diproses dua pihak
- Masalah : representasi pada sistem bisa berbeda-beda
  - ASCII vs Unicode
  - Big-endian vs Little-endian

Big vs Little Endian system





Marshalling / Unmarshalling

- Marshalling : proses konversi data menjadi bentuk yang cocok/tepat untuk transmisi pesan antar host
  - Unmarshalling : proses kebalikannya
- Pendekatan yang umum digunakan:
  - CORBA dan RMI Marshalling
  - Java serialization
  - XML – XML RPC, Web Services

Contoh Pengiriman Data CORBA message

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	`5`	*length of string*
4–7	`"Smit"`	*'Smith'*
8–11	`"h  "`	
12–15	`6`	*length of string*
16–19	`"Lond"`	*'London'*
20-23	`"on  "`	
24–27	`1934`	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

Indication of Java **serialized** form

Serialized values				*Explanation*
Person	8-byte version number			*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1934	5 Smith	6 London		*values of instance variables*

XML definition of the Person structure

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1934</year>
        <!-- a comment -->
</person >
```

NEXT

- Pemrograman Socket
  - Sumber: Distributed Systems Chapter 4 khusus pemrograman socket, George
  - Sumber: An introduction to network programming with Java, Jan Graba
- Buatlah:
  - Pengertian, cara kerja
  - Contoh program socket connection oriented + connectionless oriented