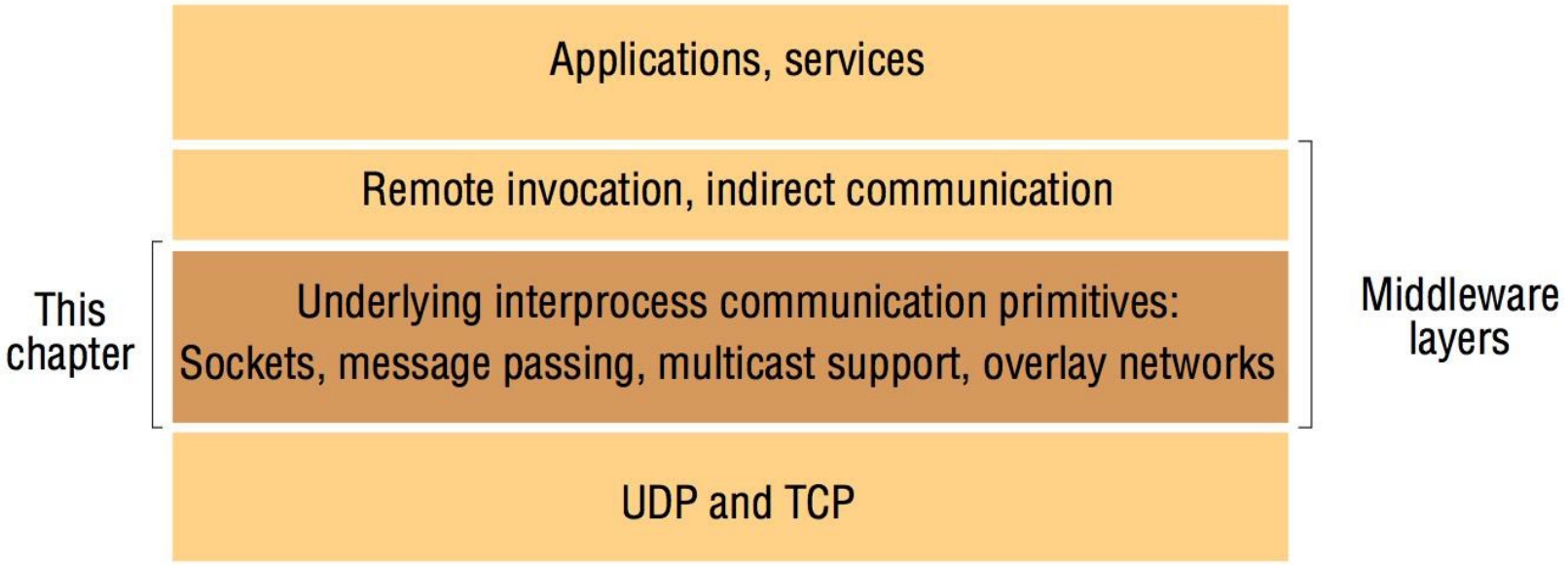
Fakultas Informatika
School of Computing
Telkom University

CSG3L3
SISTEM TERDISTRIBUSI

Topik 4 : Komunikasi Inter-Proses

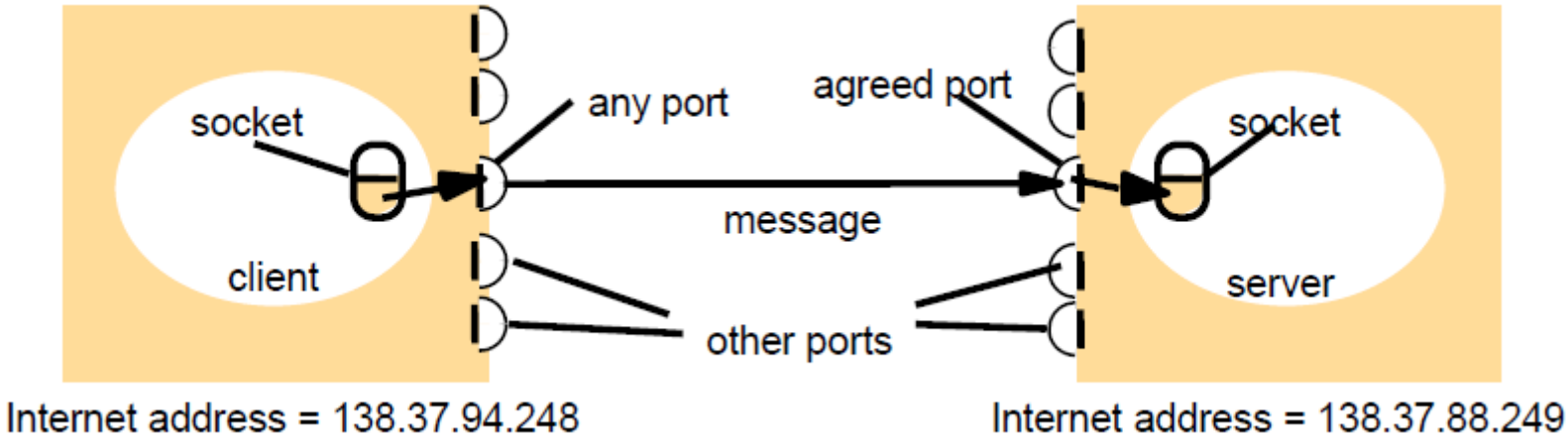
Struktur *Layer* pada *Middleware*



Letak lapisan *middleware (*agak berbeda dengan pemandangan jaringan komputer secara konsep *layer*). Komunikasi yang digunakan umumnya adalah komunikasi inter-proses

Bagaimana mekanisme komunikasi inter-proses antara *client* dan *server*?

Berikut ini adalah gambar ilustrasinya:



Contoh *source code* klien UDP mengirimkan sebuah *message* ke *server* dan mendapatkan balasan dari *server* menggunakan bahasa pemrograman Java

```
import java.net.*;

Import.java.io.*;

public class UDPClient {

    public static void main(String[] args) {

        DatagramSocket socket = null;

        try {

            socket = new DatagramSocket();
```

```
        byte[] msg = args[0].getBytes();

        InetAddress host = InetAddress.getByName(args[1]);

        int serverPort = 6789;

        DatagramPacket req = new DatagamPacket(msg,
msg.length(), host, serverPort);

        socket.send(req);

        byte[] buffer = new byte[100];

        DatagramPacket reply = new DatagramPacket(buffer,
buffer.length());

        socket.receive(reply);
```

```
        System.out.println("Reply: "+new
String(reply.getData()));
    } catch(SocketException e) {
        System.out.println("Socket: "+e.getMessage());
    } catch(IOException e) {
        System.out.println("IO: "+e.getMessage());
    } finally {
        if(socket != null) socket.close();
    }
```

Apa maksud *code* ini?

```
import java.io.*;
import java.net.*;
Public class UDPServer {
    public static void main(String[] args) {
        DatagramSocket socket = null;
        try {
            socket = new DatagramSocket(6789);
            byte[] buffer = new byte[1000];
```

```
            while(true) {

                DatagramPacket req = new
DatagramPacket(buffer, buffer.length());

                socket.receive(req);

                DatagramPacket reply = new
DatagramPacket(req.getData(), req.getLength(),
req.getAddress(), req.getPort());

                socket.send(reply);

            }

        } catch(SocketException e) {

            System.out.println("Socket: "+e.getMessage());
```

```
        } catch(IOException e) {
            System.out.println("IO: "+e.getMessage());
        } finally {
            if(socket != null) socket.close();
        }
    }
}
```

Klien TCP membuat koneksi ke *server*, mengirimkan *request*, dan menerima balasan dari *server*

```
import java.net.*;
import java.io.*;
public class TCPClient {
        public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
            try{
                    int serverPort = 7896;
                    s = new Socket(args[1], serverPort);
                    DataInputStream in = new DataInputStream( s.getInputStream());
                    DataOutputStream out =
                            new DataOutputStream( s.getOutputStream());
                    out.writeUTF(args[0]);          // UTF is a string encoding see Sn 4.3
                    String data = in.readUTF();
                    System.out.println("Received: "+ data) ;
                }catch (UnknownHostException e){
                            System.out.println("Sock:"+e.getMessage());
                }catch (EOFException e){System.out.println("EOF:"+e.getMessage());
                }catch (IOException e){System.out.println("IO:"+e.getMessage());}
        }finally {if(s!=null) try {s.close();}catch (IOException
e){System.out.println("close:"+e.getMessage());}}
        }
}
```

Server TCP membuka koneksi untuk tiap klien kemudian menampilkan *request* dari tiap klien

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
            try{
                    int serverPort = 7896;
                    ServerSocket listenSocket = new ServerSocket(serverPort);
                    while(true) {
                            Socket clientSocket = listenSocket.accept();
                            Connection c = new Connection(clientSocket);
                    }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
    }
}
```

```
class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
            try {
                    clientSocket = aClientSocket;
                    in = new DataInputStream( clientSocket.getInputStream());
                    out =new DataOutputStream( clientSocket.getOutputStream());
                    this.start();
             } catch(IOException e)  {System.out.println("Connection:"+e.getMessage());}
        }
        public void run(){
            try {                                       // an echo server
                    String data = in.readUTF();
                    out.writeUTF(data);
            } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
            } catch(IOException e) {System.out.println("IO:"+e.getMessage());}
            } finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
        }
}
```

Tipe bentukan pada CORBA CDR

Tipe	Representasi / Format
Sequence	Panjang (*unsigned long*) | Elemen data
String	Panjang (*unsigned long*) | Urutan ksarakter
Array	Elemen array berurut sesuai indeks (panjang tidak didefinisikan karena ukurannya tetap)
Struct	Urutan deklarasi komponen beserta tipe datanya
Enumerasi	*Unsigned long* | (nilai diletakkan sesuai urutan pendeklarasian)
Union	Tag tipe | elemen terpilih

Contoh *message* CORBA CDR

Indeks (dalam *byte*)	4 byte	Keterangan
0-3	5	Panjang dari string "Smith"
4-7	"Smit"	
8-11	"h___"	
12-15	6	Panjang dari string "London"
16-19	"Lond"	
20-23	"on___"	
24-27	1984	*Unsigned long*

Bentuk *"flattened"* dari *message* dengan tipe data bentukan *(Person)* yang nilainya: {'Smith', 'London', 1984}

Format data ter-serialisasi pada Java

Data yang di-serialisasi			
Person	Nomor versi (8-byte)		h0
3	Int year	java.lang.String name;	java.lang.String place;
1984	5 "Smith"	6 "London"	h1

Penjelasan			
Nama *class*	Nomor versi		*Handler* (h0)
Jumlah variabel	Tipe data & nama variabel 1	Tipe data & nama variabel 2	Tipe data & nama variabel 3
Nilai dari masing-masing variabel			*Handler* (h1)

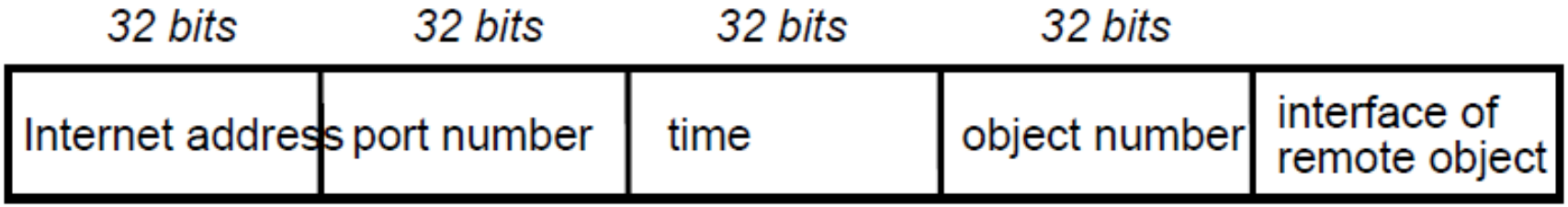
Coba teman-teman sekalian cari tahu apakah nilai dari *h0* dan *h1* serta apa fungsinya??

Bentuk Tipe Data *Person* dalam XML

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1984</year>
        <!-- a comment -->
</person >
```

Penting: Penggunaan *namespace* pada XML sangat penting. Mengapa? Bisakah teman-teman memberikan contohnya? Pelajari pula pendefinisian data (skema XML) atau DTD.

Representasi dari sebuah *remote object reference*



Contoh *code* sederhana *send* dan *receive* datagram pada sebuah "Grup Jaringan (Multicast)"

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
        public static void main(String args[]){
         // args give message contents & destination multicast group (e.g. "228.5.6.7")
        MulticastSocket s =null;
         try {
                InetAddress group = InetAddress.getByName(args[1]);
                s = new MulticastSocket(6789);
                s.joinGroup(group);
                byte [] m = args[0].getBytes();
                DatagramPacket messageOut =
                        new DatagramPacket(m, m.length, group, 6789);
                s.send(messageOut);
```

```
            // get messages from others in group
                    byte[] buffer = new byte[1000];
                    for(int i=0; i< 3; i++) {
                        DatagramPacket messageIn =
                                new DatagramPacket(buffer, buffer.length);
                        s.receive(messageIn);
                        System.out.println("Received:" + new String(messageIn.getData()));
                    }
                    s.leaveGroup(group);
                }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                }catch (IOException e){System.out.println("IO: " + e.getMessage());}
            }finally {if(s != null) s.close();}
    }
}
```

Tipe *Overlay* pada Sistem Terdistribusi

Tujuan	Jenis	Deskripsi
Dirancang untuk kebutuhan aplikasi	Tabel *hash* terdistribusi	Salah satu jenis *overlay* yang sering ditemukan pada jaringan, menawarkan layanan untuk mengelola sebuah pemetaan dari *keys* ke *values* (Map(*k*,*v*)) pada sebuah lingkungan jaringan desentralisasi dengan jumlah *node* yang sangat besar (sama seperti tabel *hash* standar)

Tipe *Overlay* pada Sistem Terdistribusi (cont'd)

Tujuan	Jenis	Deskripsi
Dirancang untuk kebutuhan aplikasi	P2P *file sharing*	Struktur *overlay* yang fokus untuk keperluan pengalamatan dan mekanisme *routing* untuk mendukung sistem *cooperative discovery & use of file* (contoh: *download)*
	Distribusi konten jaringan	Fokus kepada replikasi, *caching*, dan strategi penempatan infrastruktur untuk meningkatkan performansi *content delivery* kepada user. Menawarkan fitur *real-time* untuk kebutuhan *streaming*.

Tipe Overlay pada Sistem Terdistribusi (cont'd)

Tujuan	Jenis	Deskripsi
Dirancang untuk *support* berbagai tipe jaringan	Jaringan *wireless ad-hoc*	Menyediakan protokol *routing* kustom untuk jaringan *wireless ad-hoc*, termasuk skema pembentukan topologi *routing* secara efektif dan fleksibel di atas *node* yang digunakan dan skema reaktif pembentukan *routes on demand* yang biasanya dilakukan dengan cara *flooding*.

Tipe Overlay pada Sistem Terdistribusi (cont'd)

Tujuan	Jenis	Deskripsi
Dirancang untuk *support* berbagai tipe jaringan	Jaringan *Disruption-Tolerant*	Dirancang untuk beroperasi pada lingkungan yang istilahnya "kacau" atau memiliki potensi kegagalan (*failure*), kerusakan, dan *delay* pada *node* dan *link* yang sangat tinggi

Tipe Overlay pada Sistem Terdistribusi (cont'd)

Tujuan	Jenis	Deskripsi
Dirancang untuk fitur tambahan	*Multicast*	Awal mula penggunaan *overlay* pada jaringan *internet* adalah menyediakan akses layanan "multicast" dimana belum adanya *router* yang digunakan khusus untuk kebutuhan *multicast*. ("Mbone": *Multicast Backbone*)

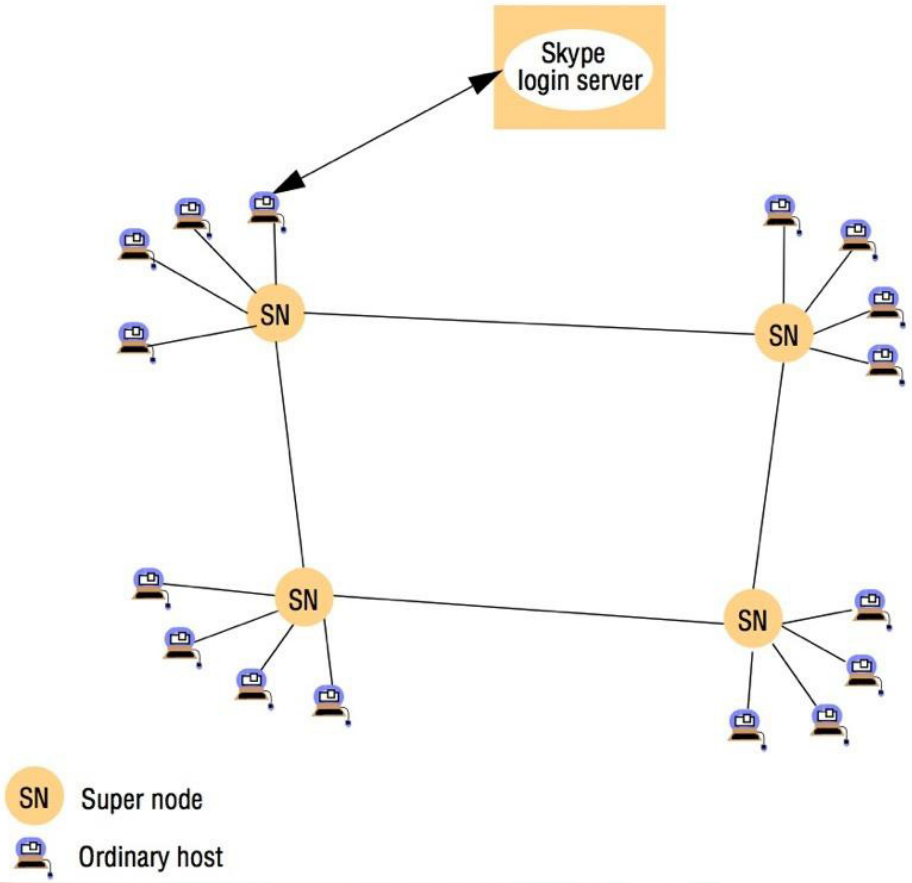
Tipe Overlay pada Sistem Terdistribusi (cont'd)

Tujuan	Jenis	Deskripsi
Dirancang untuk fitur tambahan	*Resilience*	Teknik *overlay* yang digunakan untuk mencari/mengukur orde perbesaran (*magnitude*) dari peningkatan kekuatan (*robustness*) dan ketersediaan dari sebuah jalur pada jaringan *internet*. (nms.csail.mit.edu)

Tipe Overlay pada Sistem Terdistribusi (cont'd)

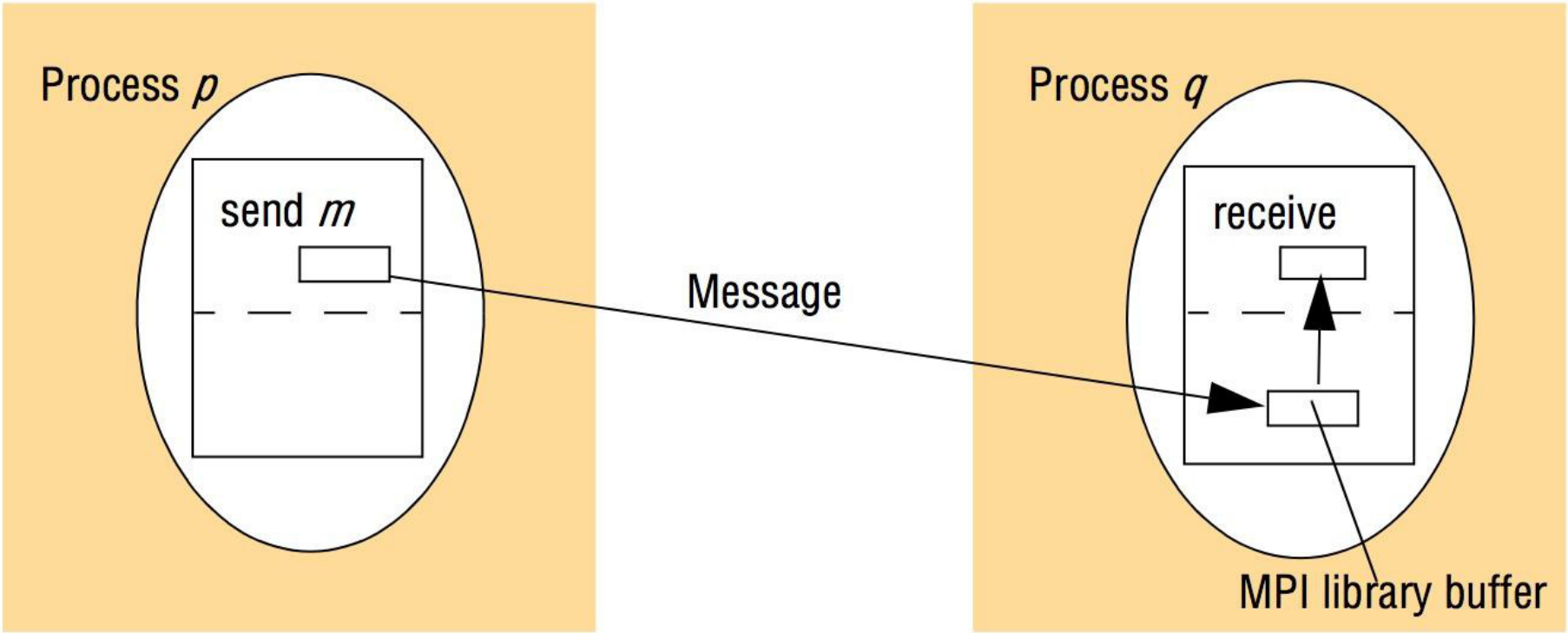
Tujuan	Jenis	Deskripsi
Dirancang untuk fitur tambahan	Keamanan (*security*)	Menawarkan jaringan yang lebih aman (*secure*) di bawah *underlying* IP termasuk *Virtual Private Network* (VPN)

Contoh arsitektur *overlay* yang digunakan oleh "Skype"



Termasuk jenis *overlay* apakah ini?

Komunikasi *Point-to-Point* pada MPI



Operasi pengiriman (*send*) pada MPI

Operasi	Blocking	Non-blocking
Generic	MPI_Send: Pengirim melakukan *blocking* hingga proses eksekusi selesai dilakukan – pesan mengalami *transit* atau telah terkirim dan *buffer* aplikasi milik si pengirim dapat digunakan kembali	MPI_Isend: Aksi pengiriman pesan dilakukan segera dan *programmer* diberikan sebuah *request* penanganan (*handle*) yang dapat digunakan untuk mengecek progres pengiriman pesan melalui pemanggilan *method* MPI_Wait atau MPI_Test

Operasi pengiriman (*send*) pada MPI (cont'd)

Operasi	Blocking	Non-blocking
Synchronous	MPI_Ssend: pengirim dan penerima melakukan sinkronisasi dan eksekusi terhadap sebuah *method* "send" selesai jika pesan telah diterima oleh penerima (*receiver*)	MPI_Issend: Menjalankan fungsi MPI_Isend disertai dengan pemanggilan fungsi MPI_Wait atau MPI_Test untuk memastikan pesan telah diterima oleh *receiver*
Buffered	MPI_Bsend: *sender* secara eksplisit mengalokasikan sebuah *buffer* MPI dan proses eksekusi pengiriman selesai jika data telah di-*copy* ke dalam *buffer* tersebut	MPI_Ibsend: Mengeksekusi MPI_Isend dilanjutkan dengan MPI_Wait atau MPI_Test untuk memastikan pesan telah di-*copy* ke *buffer* MPI milik si pengirim

Operasi pengiriman (*send*) pada MPI (cont'd)

Operasi	Blocking	Non-blocking
Ready	MPI_Rsend: Pemanggilan fungsi selesai jika *buffer* aplikasi si pengirim dapat digunakan kembali (identik dengan MPI_Send), namun *programmer* harus memastikan apakah *receiver* sudah siap menerima pesan untuk menghasilkan implementasi yang lebih optimal (optimasi)	MPI_Irsend: Efeknya sama dengan pemanggilan fungsi MPI_Isend dan MPI_Rsend. Kedua *node* baik *sender* maupun *receiver* harus siap menampung pesan berikutnya barulah pengiriman selanjutnya dimulai (ideal).

Ada Pertanyaan?

Referensi

- Coulouris, G. F., Dollimore, J., & Kindberg, T. (2012). *Distributed Systems: Concepts and Design 5th Edition*. London: Pearson Education.

Fakultas Informatika
School of Computing
Telkom University

THANK YOU