Sistem Terdistribusi

Lukito E. Nugroho

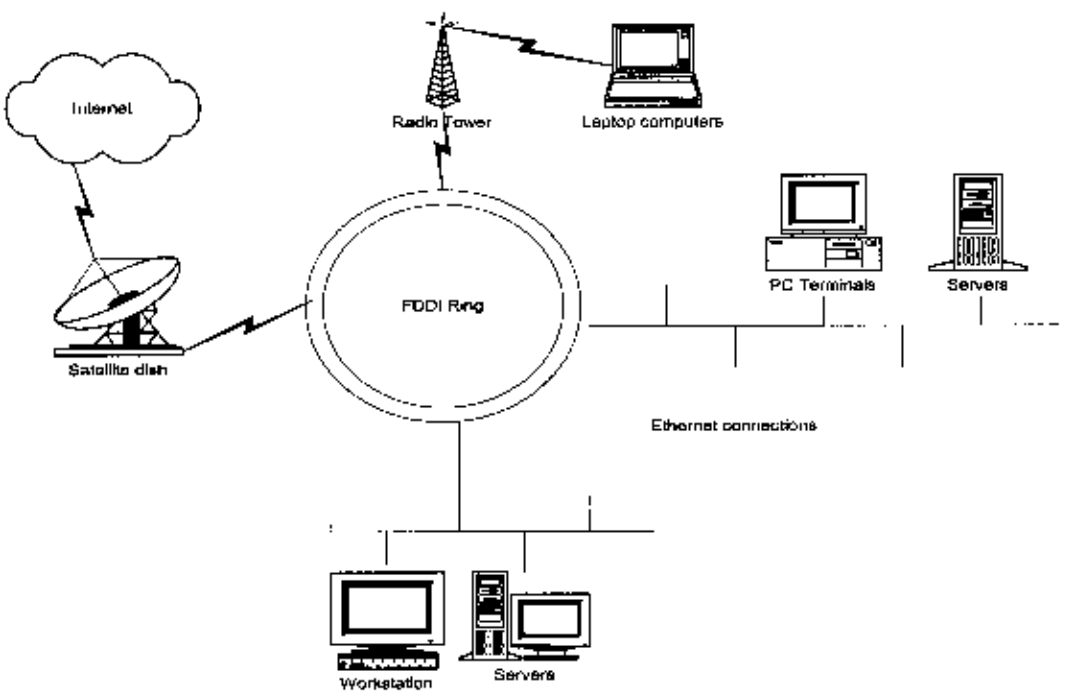
Jurusan Teknik Elektro
Fakultas Teknik UGM

Rencana Kuliah

- Topik
  - Karakteristik, isu perancangan
  - Jaringan komputer
  - Komunikasi: IPC, RPC
  - Layanan file
  - Sistem operasi terdistribusi
  - Transaksi dan pengaturannya
  - Waktu dan sinkronisasinya
- Evaluasi
  - Tugas #1 (mg 8, klp) – 25%
    - Pemrograman simulasi
    - Demo di lab Informatika: minggu ke-12
  - Tugas #2 (mg 12, klp) – 25%
    - Makalah
    - Presentasi: minggu terakhir
  - Ujian akhir – 50%
- Acuan
  - Distributed Systems: Concepts and Design, 2nd ed. (Coulouris, dkk – Addison-Wesley)

Pendahuluan

- Sistem terdistribusi
  - Kumpulan komputer
  - Mandiri *(autonomous)*
  - Jaringan komputer
  - Software untuk sistem terdistribusi
- Contoh
  - LAN, WAN, Internet
  - ATM bank, sistem reservasi hotel



Karakteristik Sistem Terdistribusi

- Pemakaian bersama atas sumber daya
  - Hardware dan software
  - Perlu resource manager (RM)
  - Hubungan antara resource dengan pihak yang menggunakannya
    - Client-server, remote evaluation, code on demand, dan mobile agent
- Keterbukaan
  - Syarat logis bagi sistem yang tumbuh dari komponen-komponen yang heterogen
  - Keterbukaan mensyaratkan interoperabilitas → membangun "jembatan" pengatur
- Konkurensi
  - Paralelisme dalam SD
  - Muncul dr sifat kemandirian dlm SD
  - Penting krn menyangkut integritas resource

Karakteristik Sistem Terdistribusi

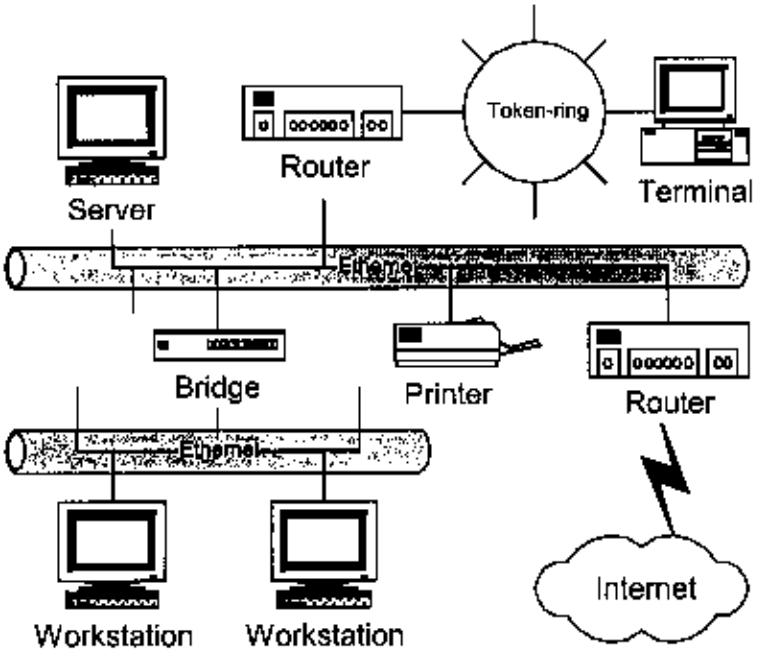
- Skalabilitas
  - Asumsi: ketersediaan resource tidak boleh dibatasi
  - Pertumbuhan SD tidak boleh berpengaruh pada sistem dan software
- Toleransi terhadap kesalahan
  - Menjaga kebenaran kinerja sistem dan ketersediaan layanan
  - Redundansi h/w dan pemulihan s/w
- Transparansi
  - Pemisahan antara user dan sistem
  - Transparansi akses dan lokasi paling dominan

Jaringan Komputer

- Infrastruktur bg sistem terdistribusi
- Diimplementasikan oleh:
  - komponen hardware
  - komponen software
  - protokol sbg "pengatur"
- Dibangun untuk sarana berkomunikasi → requirement berkisar pd aspek keterhubungan *(interconnectedness)*
  - Performance: kecepatan
  - Reliabilitas: kualitas keterhubungan
- Jenis-jenis jaringan komputer menurut luasan distribusi
  - LAN, dengan teknologi
    - Ethernet, Fast Ethernet
    - FDDI
    - Token ring
  - WAN, dengan teknologi
    - ISDN
    - ATM

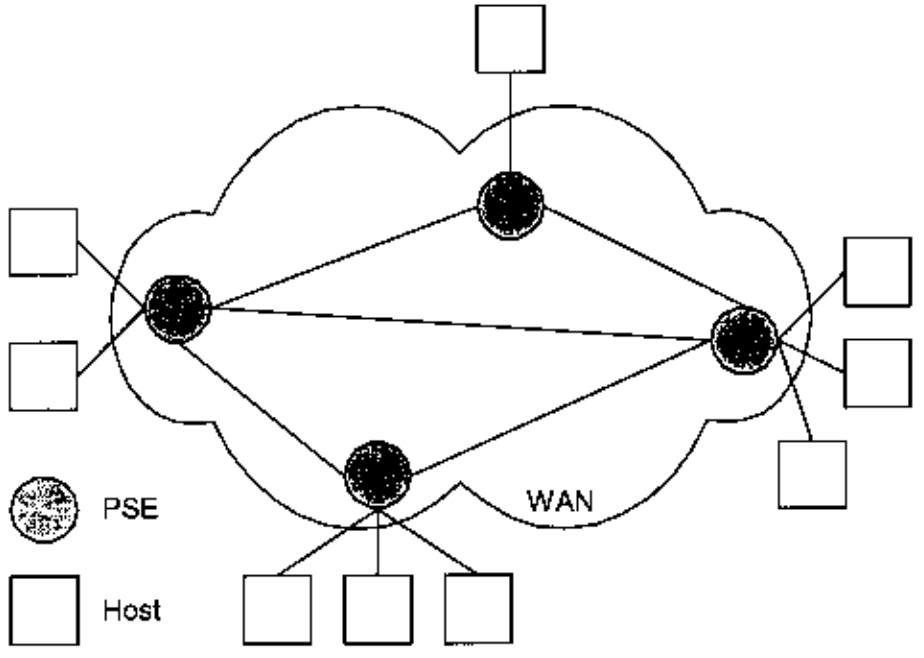
Internetworking

- Mewujudkan aspek keterbukaan dan ekstensibilitas
- Komponen
  - Jaringan subsistem
  - Teknik dan metode interkoneksi
- Diimplementasikan dng cara menghubungkan jaringan-jaringan subsistem
  - gateway, bridge, router
- Batas antar subsistem sering kabur → contoh: Internet



Wide Area Network

- Terdiri dari kumpulan saluran komunikasi yg menghubungkan PSE *(packet switching exchanges)*
- PSE menangani komunikasi data
  - Meneruskan paket data dari satu PSE ke PSE lain
  - Menentukan rute yg akan ditempuh
  - Bekerja dengan mode store-and-forward → paket data disimpan sementara di PSE sbl diteruskan ke PSE lain
  - Latency di PSE relatif besar



Local Area Network

- Dicirikan oleh topologinya
  - Bus
  - Star
  - Ring
- Operasinya menggunakan teknik *broadcast*
  - Tidak ada PSE
  - Paket data dikirim ke semua host yang ada
  - Jaringan adalah saluran yg digunakan secara bersama *(shared)*
  - Hanya ada satu pengirim pd satu saat
  - Interface berfungsi sbg pengirim dan penerima
  - Benturan antar paket *(collision)* hrs ditangani dng baik krn berpengaruh langsung thdp performance jaringan

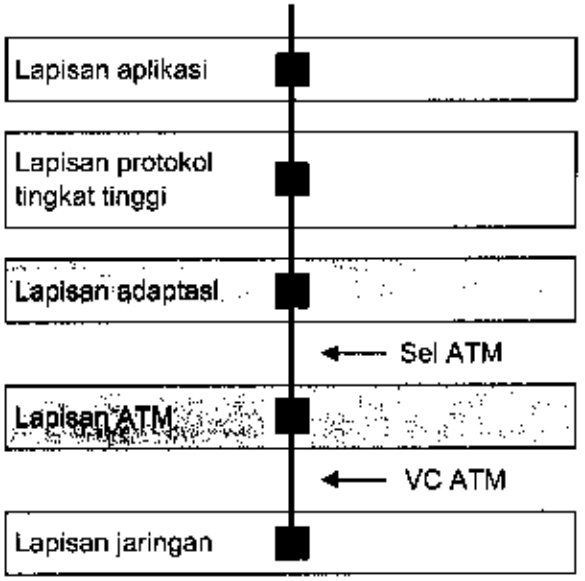
Teknologi Ethernet

- Dikembangkan di Xerox PARC, menuruti standar IEEE 802.3, beroperasi pd laju 10 Mbit/sec
- Menggunakan metode broadcasting
  - Semua host "mendengarkan" kiriman paket (frame) scr terus-menerus
  - Paket
    - Berisi data (4 - 1500 byte), alamat host asal (6 byte) dan tujuan (6 byte), dan sekuens checking
    - Panjang 64 - 1518 byte
  - Tiap host menerima paket kiriman, tapi hanya host tujuan yg akan memproses-nya lebih lanjut
- Penanganan benturan antar paket: metode CSMA/CD
  - Carrier sense → pengirim mendeteksi "sinyal" carrier sbl mengirim.
  - Collision detect → semua pengiriman dibatalkan bila terjadi benturan
  - Di host tujuan dilakukan pemeriksaan integritas paket. Jika gagal, pengiriman dianggap gagal dan proses diulangi lagi

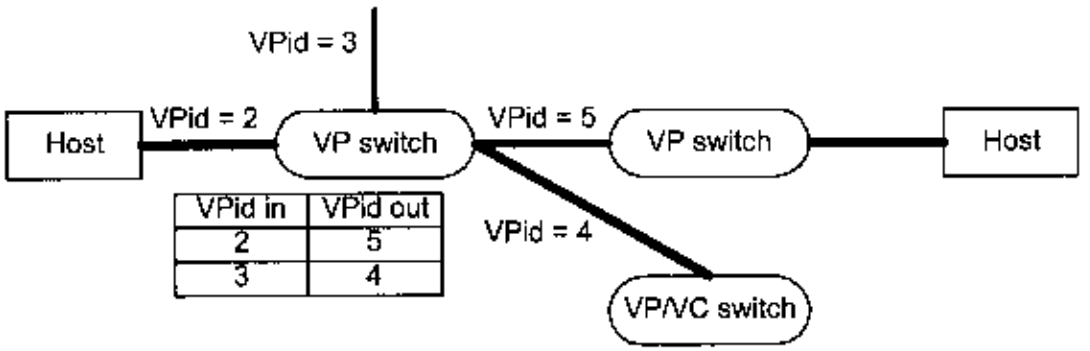
Teknologi ATM

- Teknologi packet switching melalui cell relay
- Performance tinggi, karena
  - Menghindari kontrol aliran dan pemeriksaan error di host perantara
  - Ukuran paket (cell) tetap (fixed)
- Bisa beroperasi bila ada koneksi ATM. Koneksi ATM bisa terbentuk bila resource tersedia. Sekali koneksi terbentuk, performance dapat dijamin
- Bisa digunakan untuk data multimedia
- Lapisan protokol ATM
  - Lapisan adaptasi: menyediakan protokol tingkat tinggi spt TCP/IP di atas lapisan ATM
  - Lapisan ATM: menyediakan layanan berbasis koneksi
    - Virtual channel (VC): asosiasi satu arah (logikal) antara dua titik dlm sebuah saluran
    - Virtual path (VP): kumpulan VC yg diasosia-sikan dengan saluran fisik antara dua host

Teknologi ATM

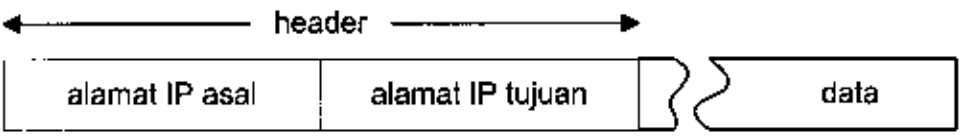


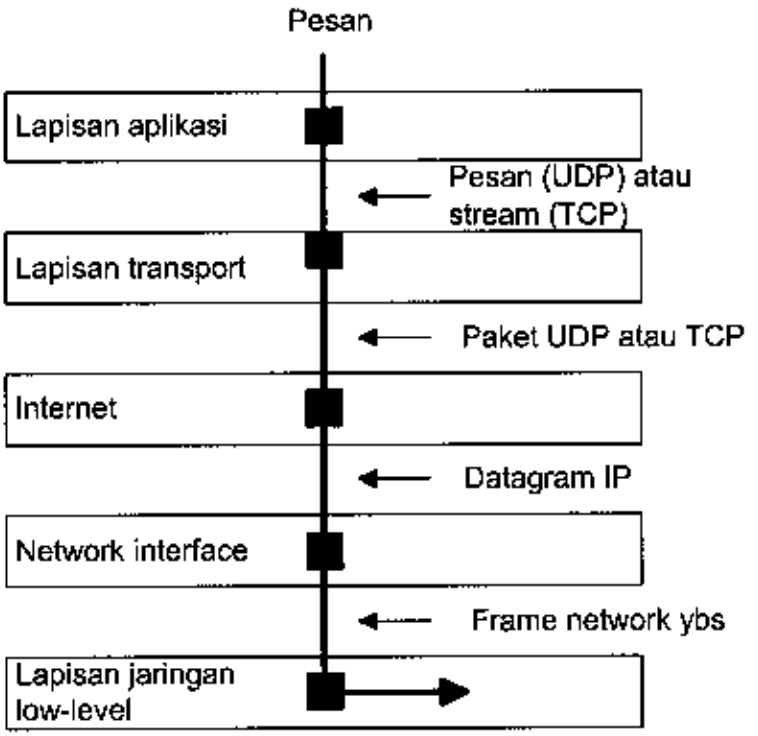
- Sebuah host ATM bisa berfungsi sebagai
  - host asal dan tujuan cell data
  - VP switch, menggunakan tabel asosiasi antara VP masuk dan VP keluar
  - VP/VC switch, menggunakan tabel sejenis untuk VP dan VC



Protokol TCP/IP

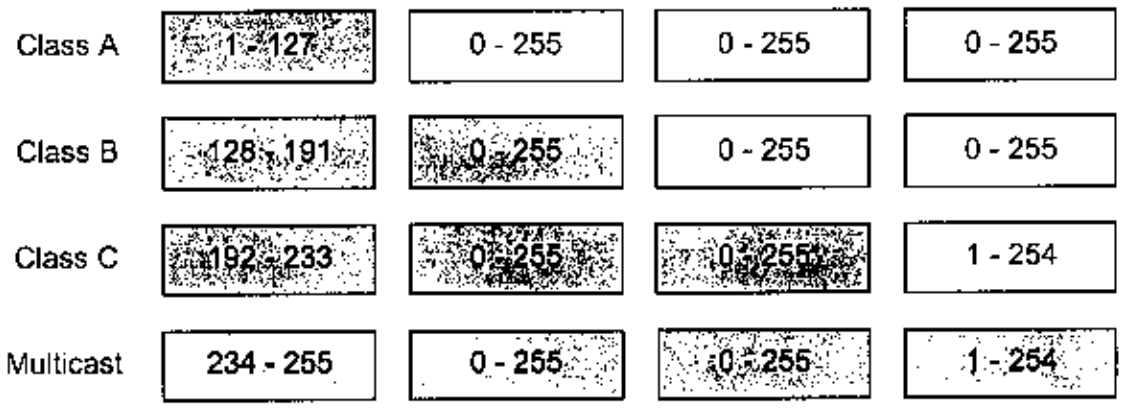
- Dirancang untuk interkoneksi jaringan
- Tidak menspesifikasikan lapisan di bawah lapisan datagram IP → paket IP dapat dikirim mengguna-kan berbagai metode → mendukung heterogenitas jaringan internetwork





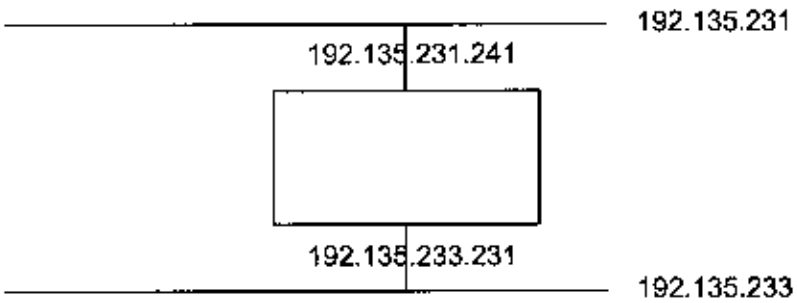
Protokol TCP/IP

- Mekanisme penamaan host
  - Nama simbolis untuk host dan network
  - Penamaan membentuk hirarki *domain* yg mencerminkan struktur organisasional
  - Hirarki penamaan tidak tergantung pd layout fisis dr network penyusun Internet
  - Nama simbolis harus ditranslasikan ke alamat IP
- Alamat IP
  - ID numeris 32 bit untuk host dan network



Protokol IP

- Resolusi alamat
  - Mengubah alamat IP ke alamat fisis (mis: alamat Ethernet)
  - Untuk Ethernet menggunakan protokol Address Resolution Protocol (ARP)
    - Database (alamat IP, alamat Ethernet) pada tiap host, dikelola oleh modul ARP
    - Pencocokan dilakukan dengan broadcasting paket ARP berisi alamat IP tujuan
    - Semua host membandingkan alamat IPnya dengan alamat IP di paket ARP
    - Jika cocok, reply diberikan ke host pengirim beserta alamat Ethernet host penerima, dan host pengirim menambahkan entry baru ini
- Routing
  - Router bertugas mengirimkan paket IP ke tujuan, atau mengirimnya ke network lain
  - Menggunakan ARP untuk menentukan alamat network tujuan

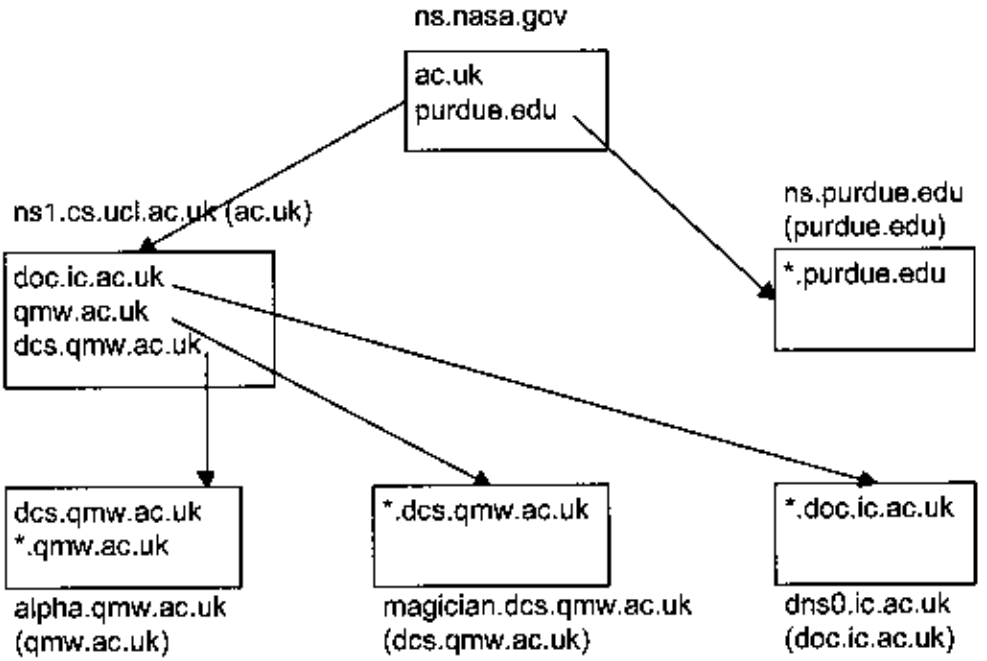


Layanan Penamaan

- Mekanisme layanan penamaan scr tersebar dan otonom
- Domain dan nama domain
  - server.te.ugm.ac.id → nama domain
  - te.ugm.ac.id → domain
- Resolusi penamaan
  - DNS digunakan untuk menentukan alamat IP dari sebuah nama domain
- Server-server DNS
  - Distribusi database nama
  - Tiap server melayani permintaan "lokal"
  - Server mengandung info ttg. server lain
  - Data DNS terbagi menjadi zona-zona. Tiap zona berisi informasi ttg:
    - Data atribut ttg nama-nama di sebuah domain
    - Nama dan alamat server yg berisi data autoritatif untuk zona ybs (versi-versi data zona yg bisa dipercaya krn up-to-date)
    - Nama dan alamat server autoritatif untuk sub domain yg didelegasikan pengelolaannya
    - Parameter-parameter manajemen zona (mis: untuk caching, replikasi)

Layanan Penamaan

- Server-server DNS (lanjutan)
  - Data autoritatif disimpan dlm file master
  - DNS server primer membaca langsung file master, DNS sekunder download data dari DNS primer
  - Sebuah server bisa melakukan caching data → tidak harus berhubungan dng sumber data utk setiap resolusi nama, tapi data menjadi non-autoritatif
  - Tiap entri pd sebuah zona punya TTL (time-to-live) → menunjukkan umur data non-autoritatif (cached)



Paradigma Komunikasi

- Metafor pembuatan roti (layanan)
  - resep (pengetahuan ttg layanan)
  - bahan (resource yg dpt berpindah)
  - oven (resource yg tdk dpt berpindah)
  - koki (komponen komputasional utk mengeksekusi)
- Paradigma client-server

  A ingin membuat roti, tetapi ia tidak tahu resepnya, tidak punya bahannya dan oven yg dibutuhkan. A tahu B memiliki semua itu. A meminta B untuk membuatkan roti, dan B menyanggupinya. Maka B membuatnya dan mengirimkannya kembali ke A.
- Paradigme code on demand

  A ingin membuat roti. Ia punya bahan dan oven, tapi tidak tahu resepnya. A tahu B punya resepnya, jadi ia minta tolong B untuk memberitahu resepnya. Setelah diberitahu, A membuat roti di rumahnya.
- Paradigma remote evaluation

  A ingin membuat roti. Ia tahu resepnya, tapi ia tidak punya bahan dan oven. B punya keduanya, tapi ia tidak tahu cara membuat roti. B mau membuatkan roti untuk A, shg A memberitahu B resepnya. B lalu membuat roti untuk A di rumahnya.

Paradigma Komunikasi

- ## Paradigma mobile agent

  A ingin membuat roti. Ia punya bahan dan resepnya, tapi ia tidak punya oven di rumahnya. A tahu B punya oven dan bersedia meminjamkannya. Maka A pergi ke rumah B dan membuat roti di sana.

Paradigma	**Sebelum**		**Sesudah**	
	S_A	**S_B**	**S_A**	**S_B**
Client-server	A	know-how resource B	A	know-how resource **B**
Code on demand	resource A	know-how B	resource *know-how* **A**	B
Remote evaluation	know-how A	resource B	A	*know-how* resource **B**
Mobile agent	know-how A	resource	---	*know-how* resource ***A***

Komunikasi Grup

- Komunikasi dari satu proses ke sekelompok proses lainnya melalui *multicasting*
- Penggunaan:
  - implementasi layanan multi-proses
  - toleransi thdp kesalahan melalui replikasi layanan dan data
  - pencarian obyek (mis: mencari file dng memberikan query ke bbrp server sekaligus)
  - pembaruan (update)
- Atomisitas
  - Diperlukan untuk replikasi layanan shg semua server memiliki kemampuan yg sama & berada pd *state* yg sama pula
  - Multicast atomik: pesan diterima oleh semua server atau tidak sama sekali
  - Pendefinisian grup utk mengeliminasi server-server yg tidak menerima pesan

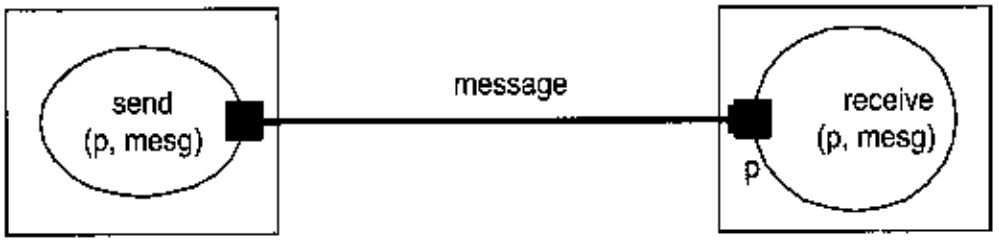
Komunikasi Grup

- Kehandalan
  - Multicast yg handal: berusaha mengirimkan pesan ke semua proses tapi tidak menjamin keberhasilannya
  - Digunakan untuk situasi yg tidak mensyaratkan semua server menjawab atau merespons pesan ybs.
- Implementasi multicast
  - Broadcast tunggal (sekali pengiriman tanpa ada verifikasi)
    - Ada pesan yg tidak sampai ke tujuan
    - Proses pengirim bisa gagal di tengah pengiriman (tidak semua tujuan menerima pesan)
  - Implementasi multicast atomik dan handal
    - Kirim pesan dan tunggu acknowledgement dari semua penerima
    - Jika ada acknowledgement yg tidak diterima dlm interval tertentu, ulangi s.d. n kali. Jika masih gagal, penerima dianggap gagal dan dihapus dari keanggotaan grup
    - Jika pengirim gagal di tengah jalan, posisinya hrs digantikan (agar atomisitas terjaga) oleh salah satu proses penerima
    - Utk itu sebuah multicast dianggap juga sbg isyarat bhw multicast sblnya sukses

Pemrograman Soket

Hall, Brian "Beej". Beej's Guide to Network Programming, 2001. www.ecst.csuchico.edu/~beej/guide/net

- Dalam UNIX, konsep IPC diwujud-kan dengan soket *(socket)*
- Soket:
  - abstraksi komunikasi
  - alamat soket = alamat IP + port
- Komunikasi dilakukan antara dua soket yang merepresentasikan dua proses (pengirim dan penerima)
- IPC diwujudkan dengan pertukaran pesan antar proses melalui sepasang soket



Pemrograman Soket

- Skenario umum IPC
  - Pembuatan soket melalui pemanggilan fungsi sistem socket()
    - Parameter: domain komunikasi (Internet), tipe komunikasi (datagram/stream), protokol
    - Kembalian: ID atau deskriptor soket
  - ID (deskriptor soket) dapat digunakan sbg referensi soket ybs
  - Penerima harus "mengikat" deskriptor soketnya ke sebuah alamat soket. Pengirim juga harus melakukan hal yg sama bila ia menginginkan jawaban
  - Pengiriman dan penerimaan pesan dapat dilaksanakan
- Jenis komunikasi
  - Komunikasi datagram (menggunakan protokol UDP + IP)
  - Komunikasi stream (menggunakan protokol TCP + IP)

Komunikasi Datagram

- Kedua proses membuat soket dan mencatat deskriptornya (fungsi socket())
- Kedua proses melakukan pengikatan soket ke alamat soket (fungsi bind()). Alamat soket penerima harus diketahui oleh pengirim
- Proses pengirim mengirimkan pesan melalui fungsi sendto(), dengan argumen:
  - soket yg digunakan untuk mengirim pesan
  - referensi ke alamat soket penerima
  - pesan yg dikirim

  Pesan dikirim tanpa acknowledgement
- Proses penerima memanggil fungsi recvfrom() untuk menerima pesan, dengan argumen
  - soket yg digunakan untuk menerima pesan
  - memori untuk buffer pesan
  - referensi ke alamat soket pengirim (diekstrak dari pesan yg diterima)

  Fungsi recvfrom() mengambil pesan dr queue di soket penerima, atau menunggu bila queue tsb. kosong

Komunikasi Stream

- Pesan diangap sbg. stream (aliran byte) & dibaca sesuai dng urutan penulisannya
- Sering digunakan utk mengimplementasikan paradigma client-server
  - Proses penerima membuat soket dan mengikatnya dng alamat soket. Kmd ia masuk ke mode "mendengarkan"
  - Proses penerima memanggil fungsi accept() utk menerima permintaan koneksi dr proses pengirim. Pengirim bisa membuat soket baru utk meneruskan komunikasi. Soket asli melanjutkan "mendengarkan"
  - Proses pengirim membuat soket dan meminta koneksi dng fungsi connect()
  - Setelah komunikasi terbentuk, pertukaran pesan dilakukan dng fungsi read() dan write()

Kerangka Pemrograman Soket

■ Komunikasi datagram

Proses pengirim

```
s = socket(AF_INET,SOCK_DGRAM,0);
•
•
bind(s, ClientAddress);
•
•
sendto(s,message,ServerAddress);
```

Proses penerima

```
s = socket(AF_INET,SOCK_DGRAM,0);
•
•
bind(s,ServerAddress);
•
•
amount = recvfrom(s,buffer,from);
```

■ Komunikasi stream

Proses client

```
s = socket(AF_INET,SOCK_STREAM,0);
•
•
connect(s,ServerAddress);
•
•
write(s,message,length);
```

Proses server

```
s = socket(AF_INET,SOCK_STREAM,0);
•
bind(s,ServerAddress);
listen(s,5);
•
sNew = accept(s,from);
•
n = read(sNew,buffer,amount);
```

Contoh: Komunikasi Stream

- Program server

```
...
int main(void)
{
  ...
  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)  exit(1);

  my_addr.sin_family = AF_INET; // host byte order
  my_addr.sin_port = htons(MYPORT); // short, network byte order
  my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
  memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

  if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
    exit(1);

  if (listen(sockfd, BACKLOG) == -1) exit(1);

  while(1) {            // main accept() loop
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,&sin_size)) == -1)
      continue;
    printf("server: got connection from %s\n",inet_ntoa(their_addr.sin_addr));

    if (!fork()) { // this is the child process
      close(sockfd); // child doesn't need the listener
      if (send(new_fd, "Hello, world!\n", 14, 0) == -1) perror("send");
      close(new_fd);
      exit(0);
    }
    close(new_fd); // parent doesn't need this
  }
  return 0;
}
```

Contoh: Komunikasi Stream

- Program client

```
...
int main(int argc, char *argv[])
{
  ...

  if (argc != 2) {
    fprintf(stderr,"usage: client hostname\n");
    exit(1);
  }

  if ((he=gethostbyname(argv[1])) == NULL)  // get the host info
    exit(1);

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    exit(1);

  their_addr.sin_family = AF_INET; // host byte order
  their_addr.sin_port = htons(PORT); // short, network byte order
  their_addr.sin_addr = *((struct in_addr *)he->h_addr);
  memset(&(their_addr.sin_zero), 8); // zero the rest of the struct

  if (connect(sockfd, (struct sockaddr *)&their_addr,
      sizeof(struct sockaddr)) == -1)
    exit(1);

  if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1)
    exit(1);

  buf[numbytes] = '\0';
  printf("Received: %s",buf);
  close(sockfd);
  return 0;
}
```

Remote Procedure Calling (RPC)

- Mengintegrasikan konsep IPC pada level pemrograman
  - Komunikasi melalui pemanggilan prosedur, melibatkan dua proses (di dua mesin) yg berbeda
  - Menggunakan model client-server
- Transparansi sintaks: sama dengan sintaks utk pemanggilan lokal
- Semantik RPC
  - Parameter input dan output
    - Param input dikirim dng cara menyalinnya ke sebuah variabel yg kmd dikirim ke server dalam pesan request
    - Param output dikembalikan ke client melalui pesan reply, kmd menggantikan nilai variabel di sisi client
    - Untuk param inout, nilainya dikirimkan baik melalui pesan request dan reply
  - Param input → pass-by-value

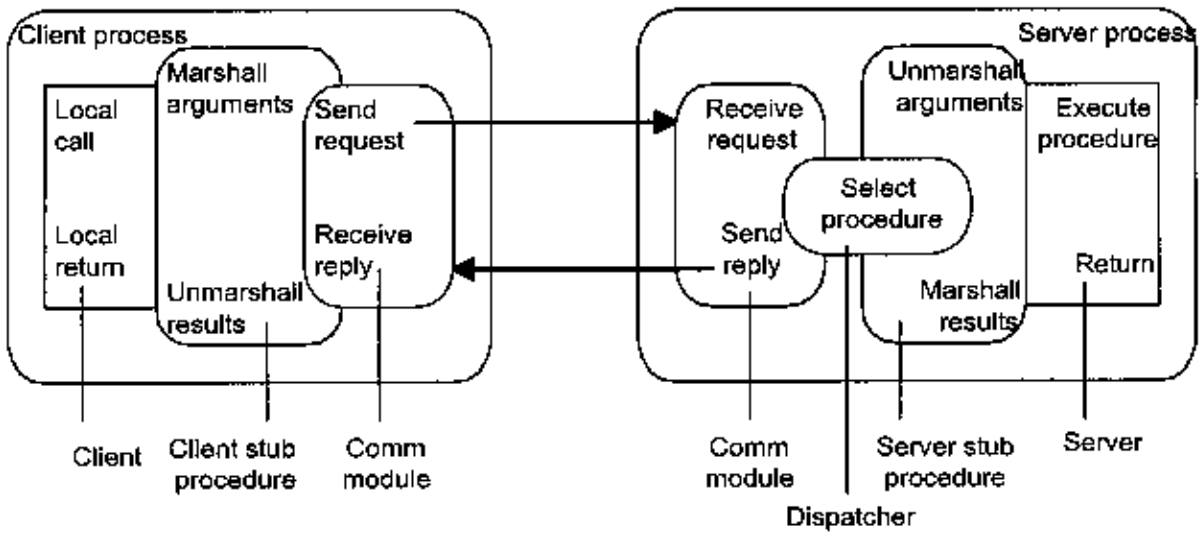
    Pass-by-reference → hrs didefinisikan sbg param input, output, atau inout → perlu IDL (interface definition language)
  - Prosedur jauh tdk bisa mengakses var di lingkungan pemanggil (mis: var global)
  - Pointer tidak punya arti → tidak boleh ada struktur data spt linked-list, tree, dsb.

Remote Procedure Calling (RPC)

- Jenis-jenis sistem RPC
  - Mekanisme RPC diintegrasikan dng bhs pemrograman yg memiliki notasi khusus utk mendefinisikan interface
  - Interface antara client dan server didefinisikan menggunakan bhs khusus
- Interface Definition Language (IDL)
  - Mendefinisikan karakteristik prosedur di sisi server yg dapat dipanggil oleh client
    - Nama prosedur
    - Parameter, tipe datanya, dan kategorinya (in, out, inout)
  - Spesifikasi interface dlm IDL dapat dikompilasi utk digunakan oleh bhs pemrograman yg berlainan → client & server dpt ditulis scr independen
- Exception handling
  - Harus lebih lengkap drpd local call krn melibatkan operasi-operasi lewat jaringan

Implementasi RPC

- Sebuah sistem RPC bertugas:
  - Mengolah spesifikasi interface dan mengintegrasikan mekanisme RPC dng bhs pemrograman
    - marshalling
    - unmarshalling
    - pemilihan prosedur jauh yg dipicu oleh pesan request dr client
  - Mengirimkan dan menerima pesan
  - Mencari server yg cocok utk sebuah layanan *(binding)*
- Komponen sistem RPC



Implementasi RPC

- IDL compiler -- IDL berlaku baik di sisi client maupun server
  - Membuat client stub
  - Membuat server stub
  - Memilih prosedur marshalling dan unmarshalling sesuai dng tipe argumen dan kembalian
  - Membuat kerangka (header) prosedur jauh -- pemrogram melengkapi badan prosedurnya
- Komunikasi antara client & server
  - Model request-reply
- Binding (pengikatan)
  - Pemetaan (mapping) antara sebuah nama (ID) dng sebuah obyek komunikasi (mis: host, port)
  - Agar host+port diketahui oleh client
  - Mekanisme binding
    - Server: registrasi
    - Client: lookup
  - Termasuk dlm layanan penamaan *(naming service)*

Contoh Kasus: Sun RPC

- Dirancang utk komunikasi client-server pd Sun NFS
- IDL: XDR, kompiler IDL: rpcgen
- Bahasa pemrograman: C

Middleware

- Muncul sekitar th 90an, berguna utk migrasi aplikasi mainframe ke aplikasi client/server
- Lapisan software yg memungkinkan bbrp proses bekerjasama di satu atau lebih komputer di jaringan
  - Berupa layanan-layanan yg beroperasi di antara lapisan aplikasi dan lapisan komunikasi jaringan, baik di sisi client maupun server
- Tanpa middleware, client & server langsung berhubungan dan harus menangani hal-hal detil:
  - Penyandian & translasi data/informasi
  - Perbedaan protokol
  - Pencarian resources
  - Pengendalian aliran informasi
  - Isu-isu portabilitas
  - Operasi asinkron
  - Penanganan kegagalan hw/sw

Middleware

- Keuntungan middleware
  - Interoperabilitas secara transparan
    - Melalui penyembunyian detil komunikasi
  - Independensi thdp layanan network
    - Akses melalui API yang homogen
  - Kehandalan
    - Implementasi mekanisme fault-tolerance
  - Skalabilitas
    - Penyembunyian koneksi langsung antara client dan server
  - Nilai tambah
    - Layanan pencarian
    - Pengamanan sistem
- Jenis-jenis middleware
  - Transaction processing (TP) monitor
  - Remote procedure call
  - Message-oriented middleware (MOM)
  - Object-request broker (ORB)

TP Monitor

- Memberikan kemampuan untuk mengelola transaksi-transaksi dlm aplikasi client/server
- Tujuan TP monitor adl untuk meningkatkan efisiensi dan kehandalan penanganan transaksi
- Area penggunaan:
  - Pemrosesan order pengiriman
  - Reservasi hotel dan pesawat terbang
  - Layanan pelanggan

Komputasi Terdistribusi

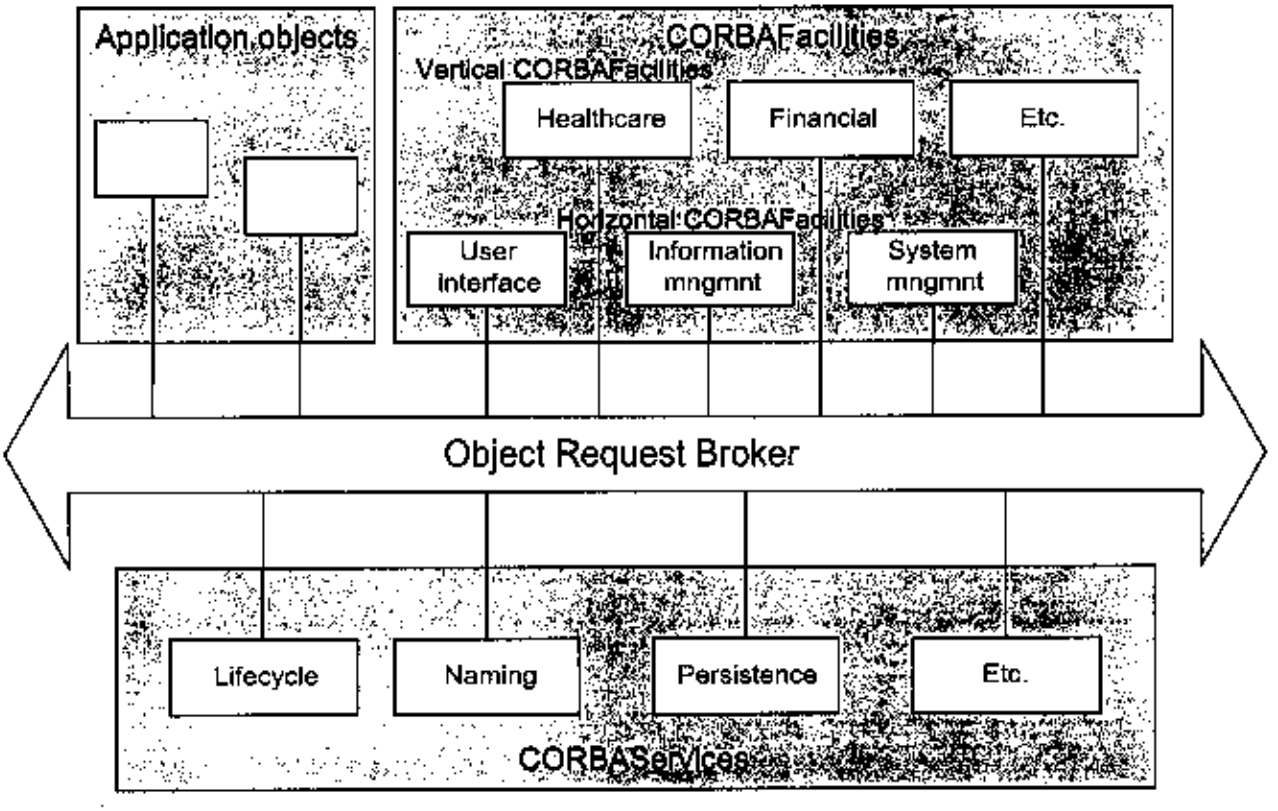
- Trend komputasi masa depan
  - Terdistribusi dan heterogen
  - *Pervasive*: merasuk ke semua aspek kehidupan manusia
  - *Very high-level, intelligent computing*
- Konsekuensi pada software dan persoalan-persoalan yg muncul
  - Software semakin kompleks → teknologi arsitektur software
  - Kebutuhan integrasi pd level tinggi → interoperabilitas pd level aplikasi
- Strategi yang ditempuh
  - Arsitektur berbasis komponen:
    - design, implementasi, & pemeliharaan
    - pemodelan
  - Standarisasi mekanisme interoperabilitas pd level aplikasi

Integrasi Berbasis OO

- Object-orientation
  - Object: komponen software yg bersifat diskret → berisi data dan cara memanipulasinya
  - Object bisa berkomunikasi satu sama lain melalui pengiriman pesan
  - Memodelkan entitas-entitas dunia nyata → problem domain sbg kumpulan obyek-obyek yg saling berkomunikasi
- Interoperabilitas object
  - Kebutuhan "plug-and-play" antar obyek
  - Problem
    - Perbedaan hardware
    - Perbedaan sistem operasi
    - Perbedaan bahasa pemrograman
  - Strategi: *interface* sebagai kontrak
    - Layanan yang disediakan sebuah obyek
    - Petunjuk kpd infrastruktur komunikasi utk bekerja dng pengiriman pesan

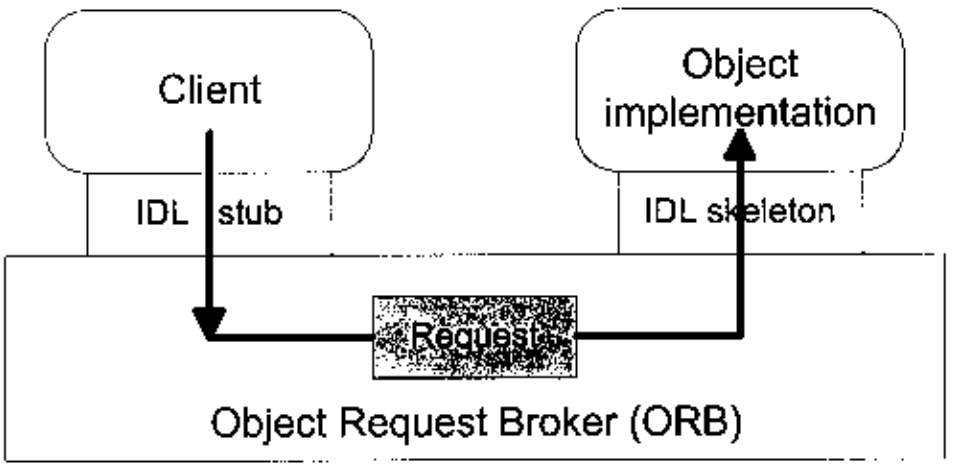
Integrasi Berbasis OO

- Integrasi pd level enterprise
  - Perlu arsitektur sbg framework
    - Standarisasi komponen
  - Konsep Object Management Architecture (OMA)



Mekanisme CORBA

- Interaksi antar object
  - Model client-server
  - Location-transparent, object memiliki *handle* sbg ID
  - ORB sbg perantara
    - Penanganan distribusi object
    - Implementasi pengiriman pesan
  - Transparansi melalui interface
    - Pemisahan antara level konsep dan implementasi
    - Program aplikasi berkonsentrasi sepenuhnya utk problem solving
    - Fleksibilitas utk substitusi implementasi di balik interface

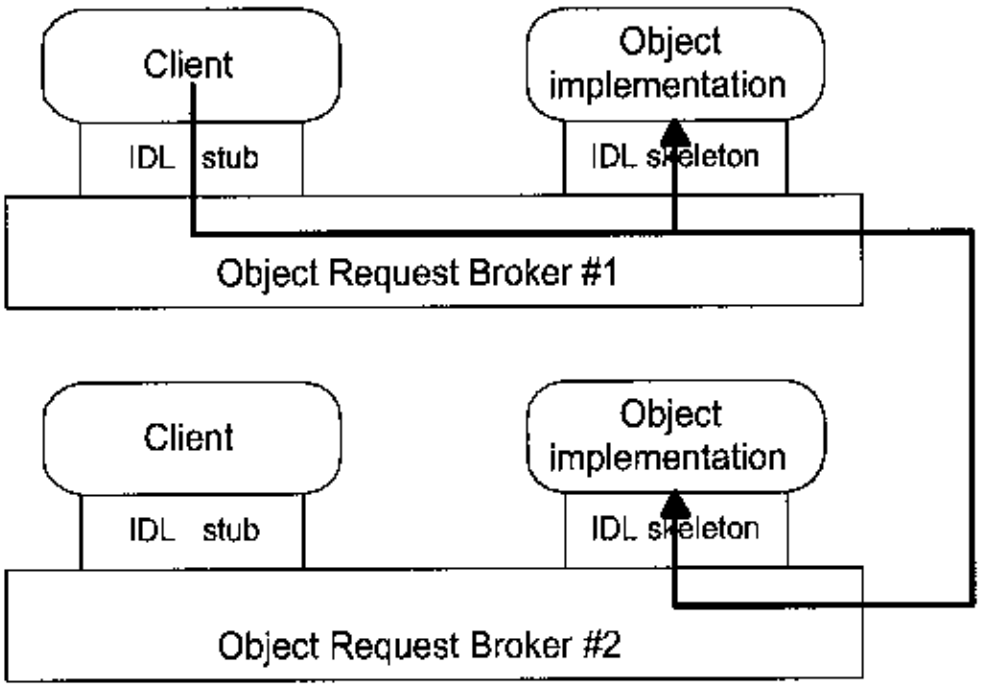


Object Request Broker (ORB)

- Konsep interoperabilitas CORBA
  - Paradigma RPC
  - Tujuan akhir: interoperabilitas antar ORB
    - Lokasi: ke mana request disampaikan?
    - Translasi: komunikasi antar ORB
- Referensi object
  - ID object → sbg *handle* object
  - Tidak berubah selama object exists
  - Transparan thdp programmer
- Interface Definition Language
  - Bahasa utk mendeskripsikan interface
  - Def. interface berlaku sbg kontrak antara client dan object impl.
  - Def. interface dpt disimpan di interface repository (IR)

Object Request Broker (ORB)

- Dynamic Invocation Interface
  - Mengakomodasi dinamika object
  - Memungkinkan client untuk:
    - Menemukan object-object baru dan mengambil interface mereka
    - Memanggil layanan yg mereka miliki
  - Cara kerja
    - Bekerjasama dengan IR (browsing IR)
    - Sering melibatkan layanan penamaan (naming dan trader services)
- Interoperabilitas antar ORB



Interface Definition Language.

IDL

- Bahasa utk spesifikasi interface
  - *What*, bukan *how*
  - Sintaks mirip bhs pemrograman
    - Konsep modul, typing, scoping
    - Operasi, exception
    - Inheritance

```
// POS Object IDL example

module POS {
   typedef string Barcode;

   interface InputMedia{
      typedef string OperatorCmd;
      void barcode_input(in Barcode item);
      void keypad_input(in OperatorCmd cmd);
   };

   interface OutputMedia{
      boolean output_text(in string outtext);
   };

   interface POSTerminal{
      void end_of_sale();
      void print_POS_sales_summary();
   };
};
```

IDL

- Language mapping
  - Dari IDL ke bhs pemrograman
  - Pemetaan 1-1 antara konstruksi IDL ke konstruksi bhs pemrograman
    - Tipe data, konstanta, object
    - Pemanggilan operasi
    - Pemberian dan pengambilan nilai atribut
    - Exception handling
  - Pemetaan yang sudah distandarisasi
    - C
    - C++
    - Smalltalk
    - Ada

Pemrograman CORBA

- MICO (www.mico.org)
  - Open source, free
  - Support C++ saja
- Contoh: bank account
  - Object account
  - Operasi deposit, withdraw, getBalance
- Langkah #1:
  - Definisikan interface dng IDL

```
interface Account {
   void deposit(in unsigned long amount);
   void withdraw(in unsigned long amount);
   long balance();
};
```

- Compile file IDL:

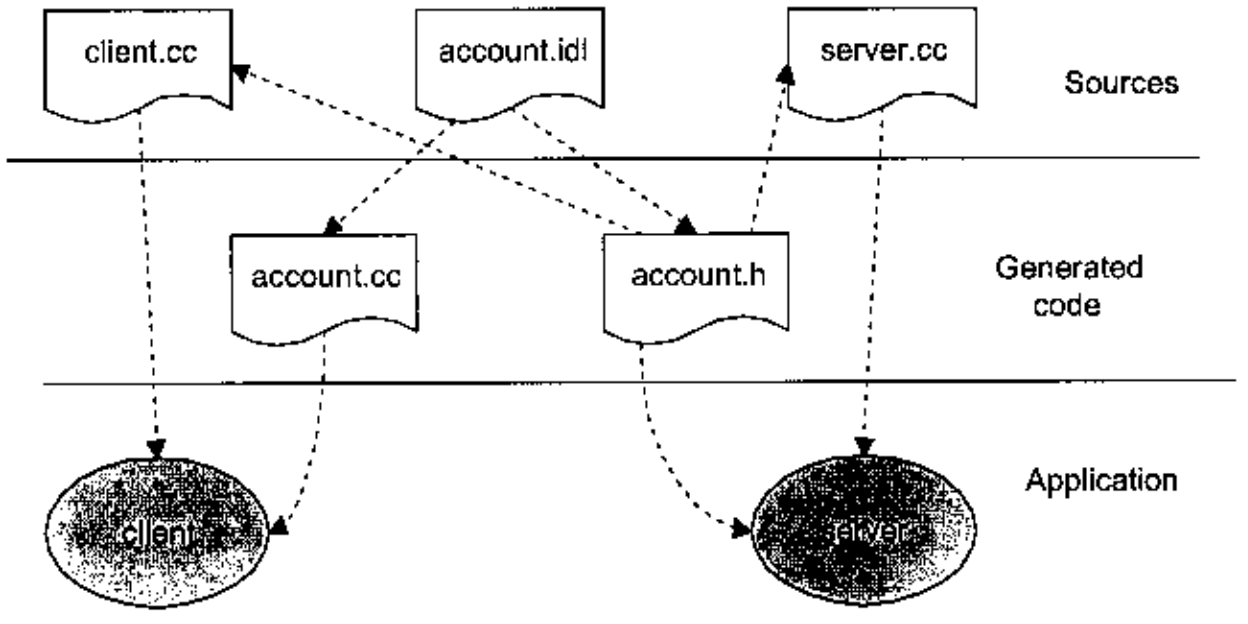
```
idl --boa --no-poa account.idl
```

- ... akan menghasilkan 3 kelas
  - Account
  - Account_skel
  - Account_stub

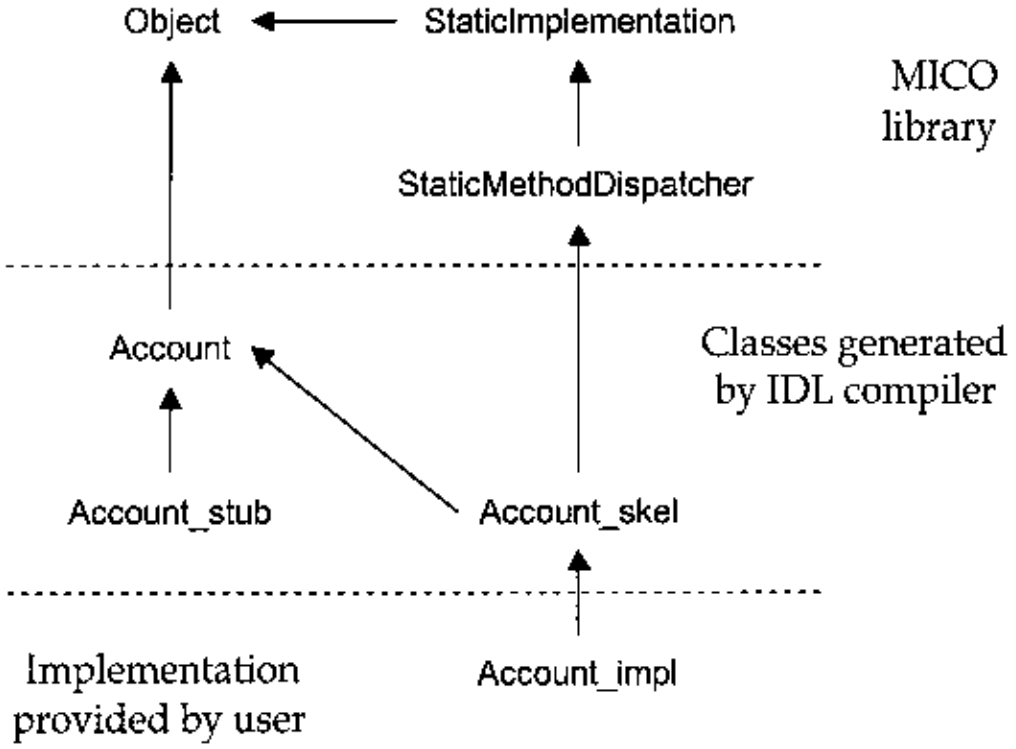
```
// from account.h
class Account : virtual public CORBA::Object {
   ...
public:
   ...
   virtual void deposit (CORBA::ULong amount) = 0;
   virtual void withdraw (CORBA::ULong amount) = 0;
   virtual CORBA::Long balance() = 0;
}


// from account.cc
class Account;
typedef Account *Account_ptr;

class Account_stub: virtual public Account {
   ...
public:
   ...
   void deposit(CORBA::ULong amount) {
      //marshalling code for deposit
   }
   void withdraw(CORBA::ULong amount) {
      //marshalling code for withdraw
   }
   CORBA::Long balance() {
      //marshalling code for balance
   }
}
```



Pemrograman CORBA



- Langkah #2
  - Tuliskan implementasi account dng cara menurunkan (inherit) dari Account_skel
- Langkah #3
  - Tuliskan kode client yg memanggil method dari object implementation

Pemrograman CORBA

```
// account_impl.cc
#include "account.h"

class Account_impl: virtual public Account_skel {
private:
   CORBA::Long _current_bal;

public:
   Account_impl() { currentbal = 0; }
   void deposit(CORBA::ULong amount) {
      _current_bal += amount;
   }
   void deposit(CORBA::ULong amount) {
      _current_bal += amount;
   }
   CORBA::Long balance() { return _current_bal; }
}

int main(int argc, char *argv[]) {
   // ORB initialization
   CORBA::ORB_var orb = CORBA::ORB_init(argc,argv,"local_orb");
   CORBA::ORB_var boa = orb->BOA_init(argc,argv,"local_boa");

   // server side
   Account_impl* server = new Account_impl;
   CORBA::String_var ref = orb->object_to_string(server);
   count << "Server reference: " << ref << endl;

   // ------------------------------------------------------

   // client side
   CORBA::Object_var obj = orb->string_to_object(ref);
   Account_var client = Account::_narrow(obj);

   client->deposit(700);
   client->withdraw(250);
   count << "Balance is " << client->balance() << endl;

   // server code
   CORBA::release(server);
   return 0;
}
```

Memisahkan Client dan Server

- Mekanisme file

```
// Account object implementation ver #2
#include "account.h"

class Account_impl: virtual public Account_skel {
   // unchanged
}

int main(int argc, char *argv[]) {
   // ORB initialization
   CORBA::ORB_var orb = CORBA_ORB_init(argc,argv,"local_orb");
   CORBA::BOA_var boa = orb->BOA_init(argc,argv,"local_boa");

   Account_impl* server = new Account_impl;

   // activate the account object
   boa->impl_is_ready(CORBA::ImplementationDef::_nil());
   orb->run();
   CORBA::release(server);
   return 0;
}


// Account client code ver #2
#include "account.h"

int main(int argc, char *argv[]) {
   // ORB initialization
   CORBA::ORB_var orb = CORBA_ORB_init(argc,argv,"local_orb");
   CORBA::BOA_var boa = orb->BOA_init(argc,argv,"local_boa");

   ifstream in("/tmp/account.objid");
   char ref[1000];
   in >> ref;
   in.close();

   CORBA::Object_var obj =
      orb->bind("IDL::Account:1.0","inet:localhost:8888");
   if (CORBA::is_nil(obj)) {
      // object not found
   }
   Account_var client = Account::_narrow(obj);

   client->deposit(700);
   client->withdraw(250);
   cout << "Balance is " << client->balance() << endl;
   return 0;
}
```

Memisahkan Client dan Server

- Layanan penamaan
  - Menjalankan daemon nsd
  - Beritahu client dan server ttg alamat nsd
  - Daftarkan layanan milik server
  - Client melakukan query ke nsd utk mencari layanan server
- Implementasi layanan penamaan
  - Pemetaan (alamat,RepositoryID) ke object reference
  - RepositoryID: string ID utk object CORBA
  - Alamat
    - Internet: inet:<host>:<port>
    - UNIX: unix:<socket file name>
    - Lokal: local:

Memisahkan Client dan Server

```
// Account object implementation ver #2
#include "account.h"

class Account_impl: virtual public Account_skel {
   // unchanged
}

int main(int argc, char *argv[]) {
   // ORB initialization
   CORBA::ORB_var orb = CORBA_ORB_init(argc,argv,"local_orb");
   CORBA::BOA_var boa = orb->BOA_init(argc,argv,"local_boa");

   Account_impl* server = new Account_impl;

   // activate the account object
   boa->impl_is_ready(CORBA::ImplementationDef::_nil());
   orb->run();
   CORBA::release(server);
   return 0;
}


// Account client code ver #2
#include "account.h"

int main(int argc, char *argv[]) {
   // ORB initialization
   CORBA::ORB_var orb = CORBA_ORB_init(argc,argv,"local_orb");
   CORBA::BOA_var boa = orb->BOA_init(argc,argv,"local_boa");

   ifstream in("/tmp/account.objid");
   char ref[1000];
   in >> ref;
   in.close();

   CORBA::Object_var obj =
      orb->bind("IDL::Account:1.0","inet:localhost:8888");
   if (CORBA::is_nil(obj)) {
      // object not found
   }
   Account_var client = Account::_narrow(obj);

   client->deposit(700);
   client->withdraw(250);
   cout << "Balance is " << client->balance() << endl;
   return 0;
}
```