Sistem Terdistribusi

TIK-604

Remote Procedure Calls

Kuliah 03: 25 s.d 27 Februari 2019

Husni

Hari ini...

- Bahasan terakhir:
  - Prinsip-prinsip *networking*
    - Prinsip Networking: *Encapsulation*, *Routing*, dan kendali *Congestion*
- Kuliah hari ini:
  - *Remote Procedure Calls*
    - Sockets
    - Remote Invocations
- Pengumuman:
  - …

Entitas yang Berkomunikasi dalam Sistem Terdistribusi

- Entitas yang berkomunikasi dalam sistem terdistribusi dapat dikelompokkan ke dalam dua tipe:
  - Entitas berorientasi sistem
    - *Process* (program komputer yang sedang berjalan)
    - *Thread* (sub-program)
    - *Node* (mesin/komputer/*smartphone*)
  - Entitas berorientasi masalah (*problem*)
    - Obyek (dalam pendekatan berbasis *object-oriented programming*)

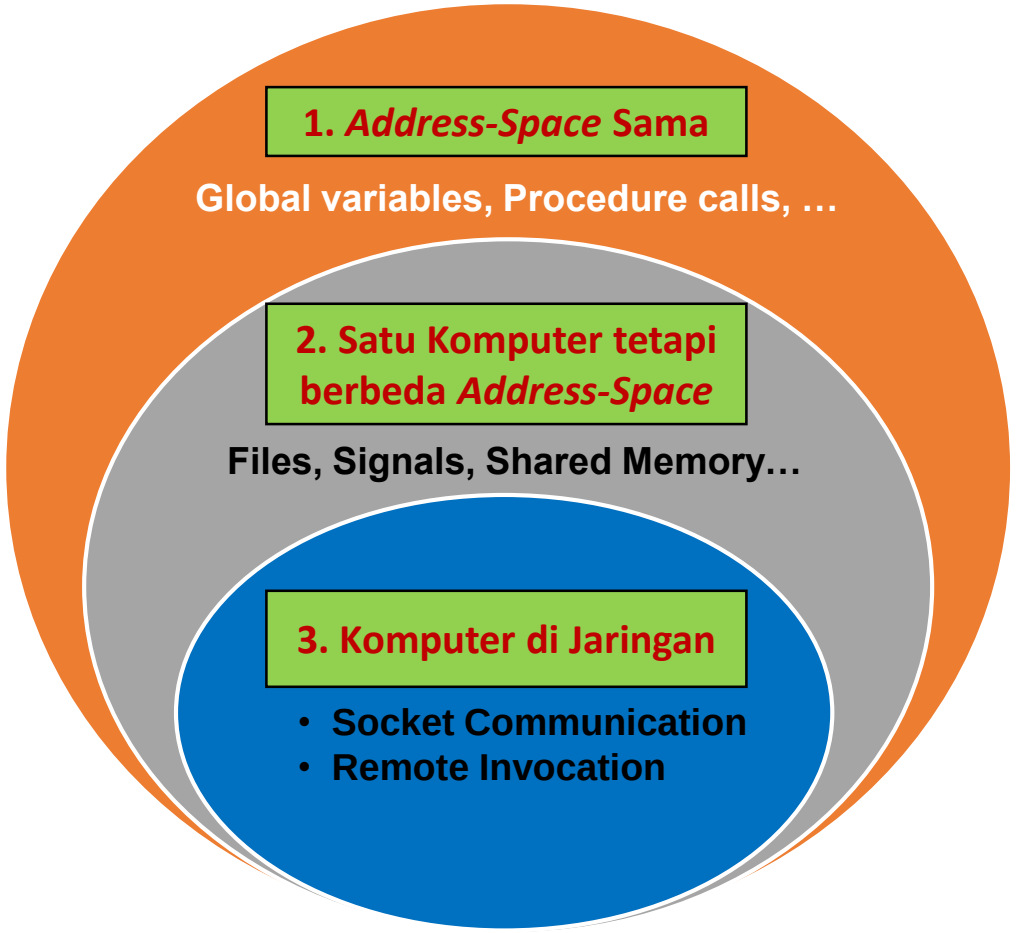
Bagaimana entitas dalam sistem terdistribusi berkomunikasi?

Paradigma Komunikasi

- Paradigma komunikasi menggambarkan dan mengelompokkan metode-metode yang dapat digunakan oleh entitas untuk berinteraksi dan bertukar data.

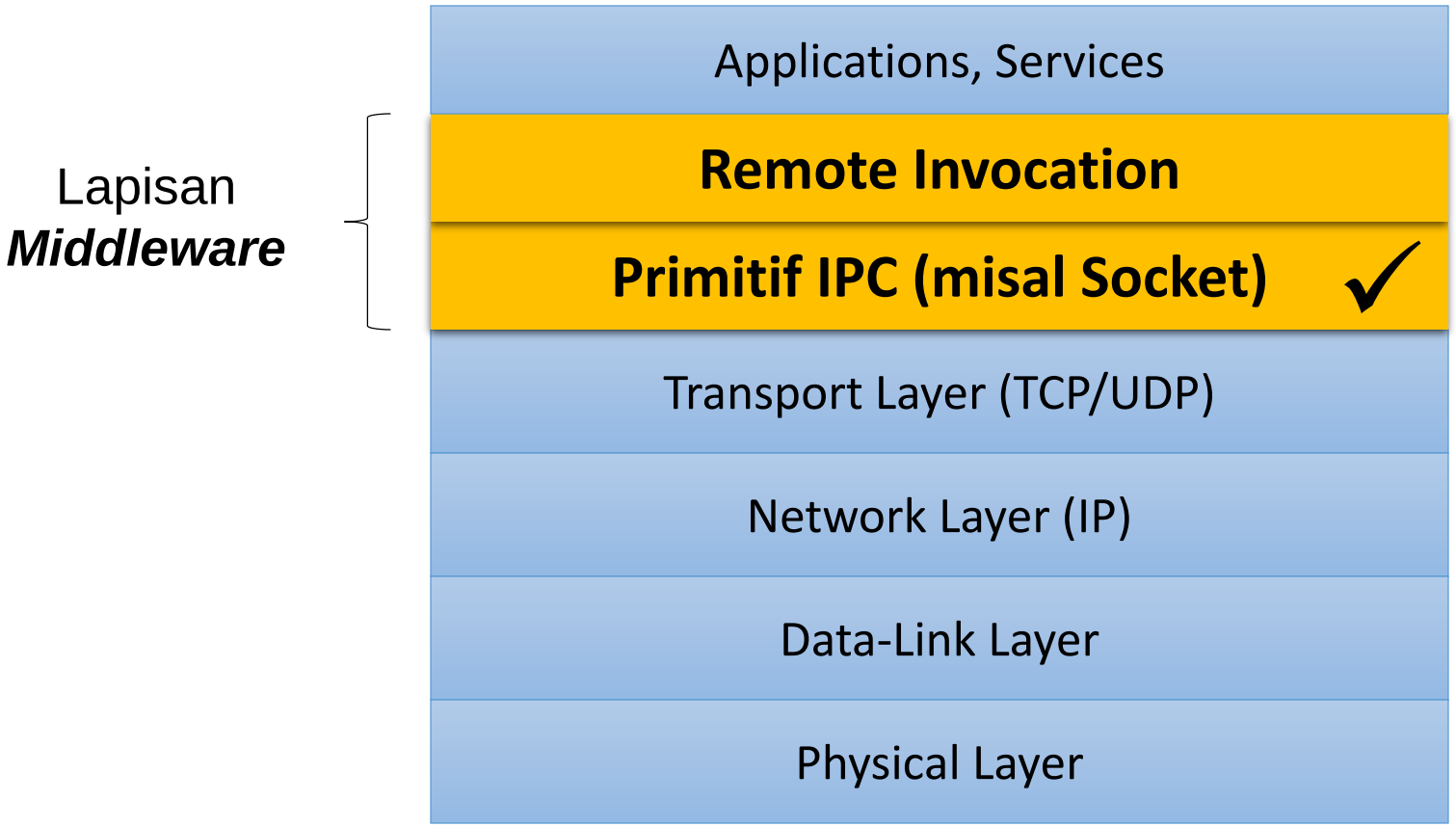
Klasifikasi Paradigma Komunikasi

- Paradigma komunikasi dapat dikategorikan ke dalam tiga tipe berdasarkan pada tempat entitas tersebut berada. Jika entitas berjalan pada:



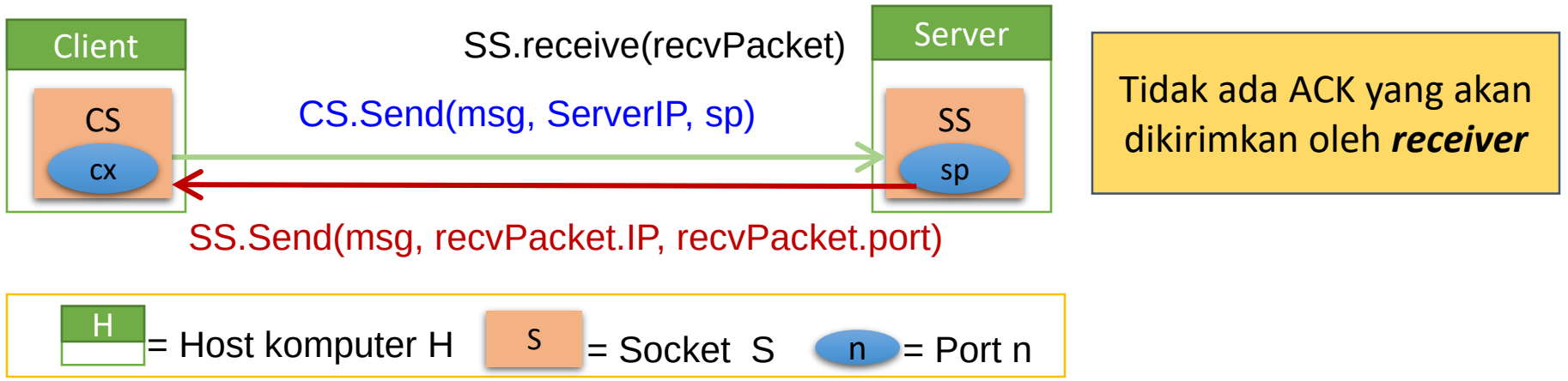
Hari ini, akan didiskusikan bagaimana entitas yang berada pada komputer jaringan berkomunikasi dalam sistem terdistribusi menggunakan komunikasi ***socket*** dan ***remote invocation***

Lapisan *Middleware*



Socket UDP

- UDP menyediakan komunikasi *connectionless*, tidak ada *acknowledgements* ataupun *message retransmissions*
- Mekanisme komunikasi:
  - Server membuka suatu socket UDP *SS* pada nomor port tertentu *sp*,
  - Socket *SS* menunggu datangnya request
  - Client membuka suatu socket UDP *CS* pada nomor port acak *cx*
  - Socket Client *CS* mengirimkan message ke ServerIP dan port *sp*
  - Socket Server SS dapat mengirimkan data balik ke *CS*



Tidak ada ACK yang akan dikirimkan oleh ***receiver***

UDP: Perhatian Rancangan

- *Sender* harus secara eksplisit memotong *message* yang panjang ke dalam potongan-potongan lebih kecil sebelum ditransmisikan
  - Ukuran maksimum yang disarankan untuk transmisi: 548 byte.

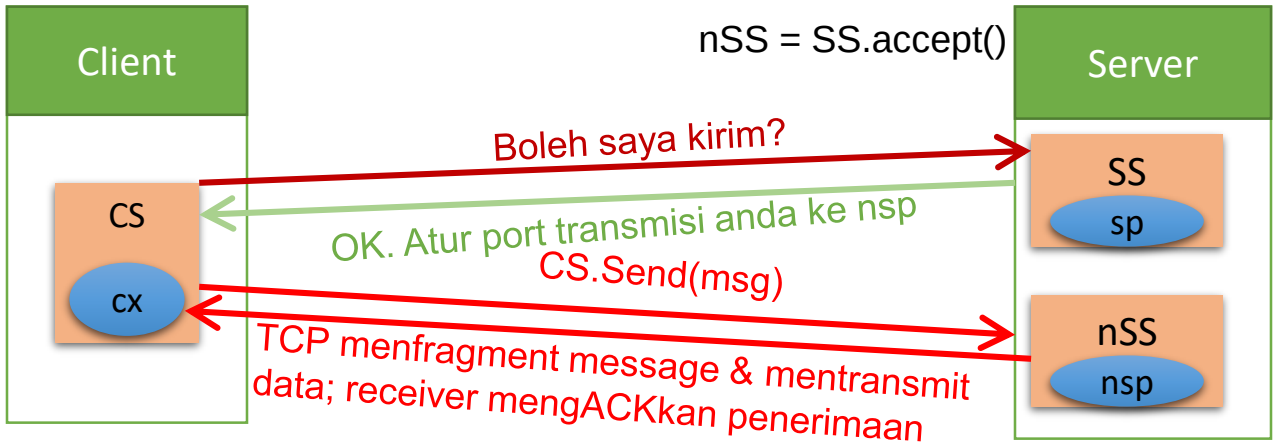
- *Messages* boleh dihantarkan secara *out-of-order*
  - Jika perlu, *programmer* harus mengurut-ulangkan paket-paket

- Komunikasi tidak *reliable* (tidak dapat diandalkan)
  - *Message* mungkin dijatuhkan karena *check-sum error* atau *buffer overflow* pada router

- *Receiver* harus mengalokasikan *buffer* yang cukup besar untuk menampung *message* dari *sender*
  - Jika tidak *message* akan terpotong.

Socket TCP

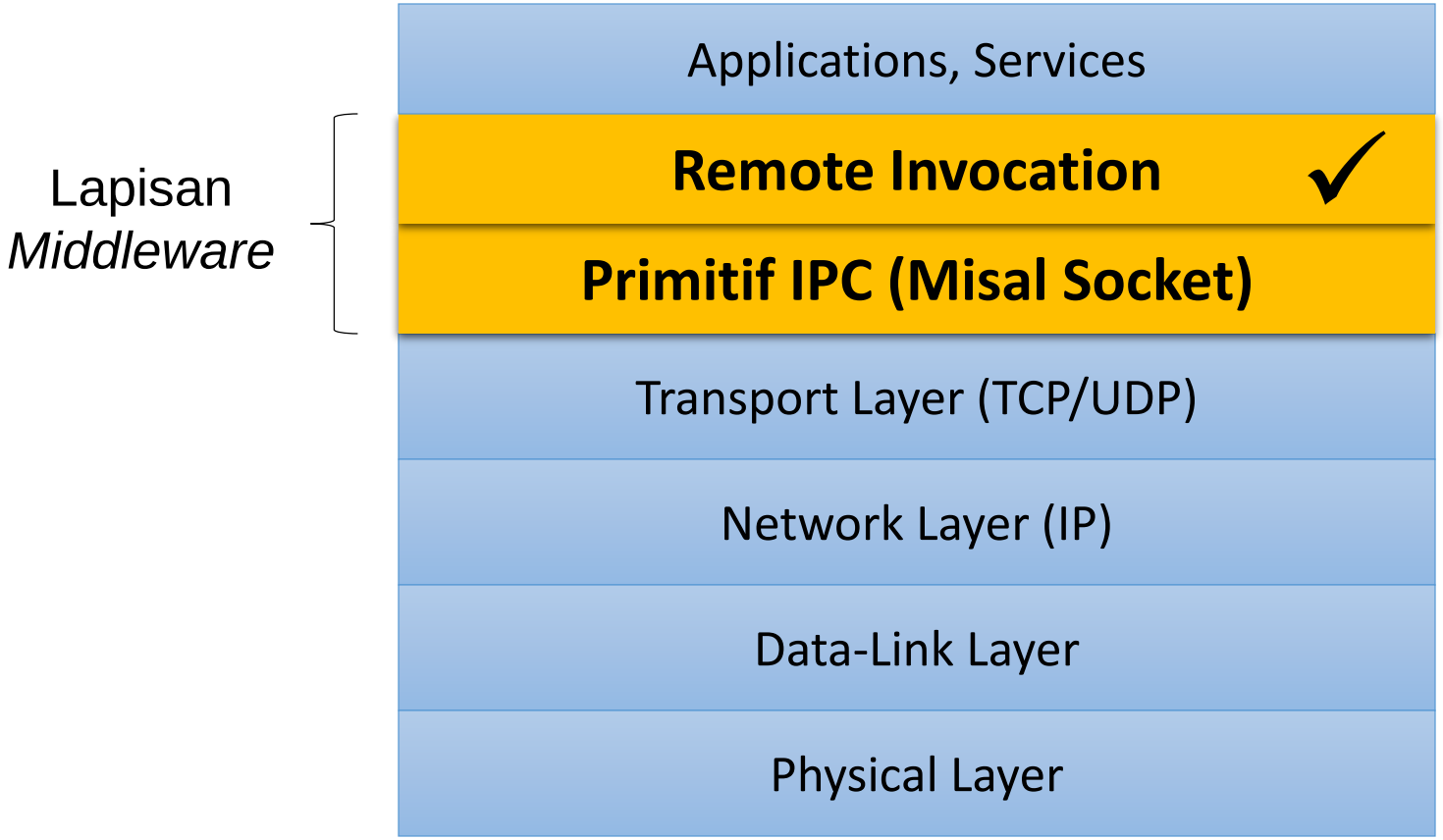
- TCP menyediakan penyampaian secara *in-order*, *reliability,* dan kendali *congestion*
- Mekanisme komunikasi:
  - Server membuka suatu TCP server socket *SS* pada port tertentu *sp*
  - Server menunggu datangnya request (menggunakan fungsi *accept*)
  - Client membuka suatu TCP socket *CS* pada nomor port acak *cx*
  - *CS* menginisiasi ***connection initiation message*** ke ServerIP dan port *sp*
  - Server socket SS mengalokasikan socket baru **NSS** pada nomor port acak ***nsp*** untuk client itu
  - *CS* dapat **mengirimkan data** ke *NSS*



Keunggulan Utama TCP

- TCP memastikan penyampaian pesan secara *in-order*
- Aplikasi dapat mengirimkan *message* berukuran berapa pun
- TCP memastikan *reliable communication* melalui penggunaan *acknowledgements* dan *retransmissions*
- Kendali kemacetan dari TCP mengatur ulang kecepatan *sender*, sehingga mencegah *network overload*.

Lapisan *Middleware*



Remote Invocation

- *Remote invocation* memungkinkan suatu entitas memanggil prosedur untuk dieksekusi pada komputer lain tanpa *programmer* harus secara eksplisit menuliskan kode rincian komunikasinya.
  - *Middleware* yang mendasari akan mengambil alih urusan komunikasi kasarnya
  - *Programmer* dapat *secara transparan* berkomunikasi dengan entitas *remote*

- Akan dibahas dua jenis *remote invocation*:

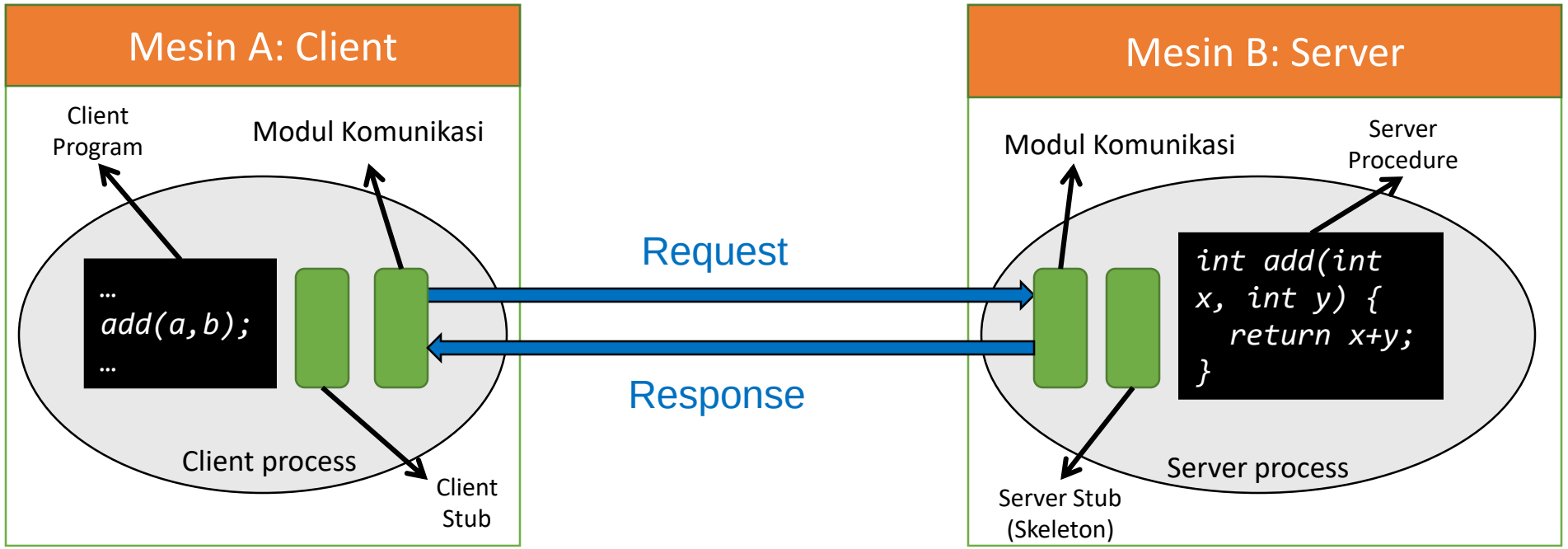
  a.Remote Procedure Calls (**RPC**)

  b.Remote Method Invocation (**RMI**)

Remote Procedure Calls (RPC)

- RPC memungkinkan *sender* berkomunikasi dengan *receiver* mengunakan suatu panggilan prosedur sederhana
  - Tidak ada komunikasi atau *message-passing* tampak untuk programmer

- Pendekatan RPC dasar:



Client Stub

- Komponen *client stub*:
  - Dipanggil oleh kode pengguna sebagai suatu prosedur lokal
  - Bungkus (atau *serializes* atau *marshals*) parameter-parameter ke dalam paket request (misalnya **request-pkt**)
  - Jalankan suatu rutin transport sisi client (Misalnya makerpc(request-pkt, &reply-pkt))
  - Buka bungkusan (atau *de-serializes* atau *unmarshals*) **reply-pkt** ke dalam parameter-parameter *output*
  - Kembali ke kode pengguna

Server Stub

- Komponen server stub:
  - Dipanggil setelah suatu rutin *transport* sisi server (misalnya getrequest()) dikembalikan
  - Unmarshals arguments, de-multiplexes opcode, dan memanggil kode server lokal
  - Marshals arguments, panggil rutin transport sisi server (misal sendresponse()), dan kembalikan ke loop server
    - Misal: Loop utama server:

```
while (1) {
        get-request (&p);       /* blocking call */
        execute-request (p);    /* demux based on opcode */
}
```

Tantangan dalam RPC

- *Parameter passing* melalui *marshaling*
  - Parameter dan hasil dari prosedur harus ditransfer pada jaringan sebagai bit-bit
- Representasi Data
  - Representasi data harus seragam
    - Arsitektur dari mesin *sender* dan *receiver* boleh berbeda
- Kerusakan/kegagalan independen
  - Client dan server dapat mengalami kegagalan secara independen

Tantangan dalam RPC

- *Parameter passing* melalui *marshaling*
  - Parameter dan hasil prosedur harus ditransfer pada jaringan sebagai bit-bit
- Representasi data
  - Representasi data harus seragam (uniform)
    - Arsitektur dari mesin sender dan receiver boleh berbeda
- Independensi kegagalan
  - Client dan server dapat mengalami kegagalan secara independen

Parameter Passing melalui *Marshaling*

- Mengemasi (*packing*) parameter-parameter ke dalam suatu *message* yang akan di-*transmit*-kan pada jaringan disebut *parameter marshalling*

- Parameter-parameter untuk prosedur dan hasilnya harus disusun sebelum mengirimkannya pada jaringan

- Dua cara yang dapat digunakan untuk melewatkan parameter:
  1. Parameter nilai (*value*)
  2. Parameter referensi

1. Melewatkan Parameter Nilai

- Parameter nilai mempunyai informasi yang lengkap mengenai variabel, dan dapat secara langsung di-encode-kan ke dalam *message*
  - Misal: integer, float, character

- Nilai-nilai dilewatkan melalui *call-by-value*
  - Perubahan yang dibuat oleh prosedur *callee* tidak tercerminkan dalam prosedur *caller*.

2. Melewatkan Parameter Referensi

- Melewatkan parameter referensi seperti parameter nilai dalam RPC dapat mengakibatkan hasil tak sesuai dikarenakan dua alasan:

  a. Tidak validnya parameter referensi pada server
     - Parameter referensi valid hanya di dalam ruang alamat client
     - Solusi: Lewatkan parameter referensi dengan menyalin data yang direferensi

  b. Perubahan parameter referensi tidak tercerminkan balik pada client
     - Solusi: “Copy/Restore” data tersebut
       - Salin data yang direferensi bersama dengan parameternya
       - Salin-balik nilai pada server ke client.

Tantangan dalam RPC

- *Parameter passing* melalui *marshaling*
  - Parameter dan hasil prosedur harus ditransfer pada jaringan sebagai bit-bit

- Representasi data
  - Representasi data harus seragam (*uniform*)
    - Arsitektur dari mesin *sender* dan *receiver* boleh berbeda

- Independensi kegagalan
  - *Client* dan *server* dapat mengalami kegagalan secara independen

Representasi Data

- Komputer-komputer dalam Sistem Terdistribusi sering mempunyai arsitektur dan sistem operasi berbeda
  - Ukuran dari tipe data berbeda
    - Misal: Suatu tipe data *long* panjangnya 4-byte dalam Unix 32-bit, sedangkan di sistem Unix 64 bit panjangnya 8-byte.
  - Format dimana data disimpan berbeda
    - Misal: Intel menyimpan data dalam format *little-endian*, sedangkan SPARC menyimpan dalam format *big-endian*.
- Client dan server harus menyepakati bagaimana data sederhana direpresentasikan dalam *message*
  - Misal: Format dan ukuran dari tipe data seperti integer, char dan float

Tantangan dalam RPC

- *Parameter passing* melalui *marshaling*
  - Parameter dan hasil prosedur harus ditransfer pada jaringan sebagai bit-bit
- Representasi data
  - Representasi data harus seragam (*uniform*)
    - Arsitektur dari mesin *sender* dan *receiver* boleh berbeda
- Independensi kegagalan
  - *Client* dan *server* dapat mengalami kegagalan secara independen.

Independensi Kegagalan

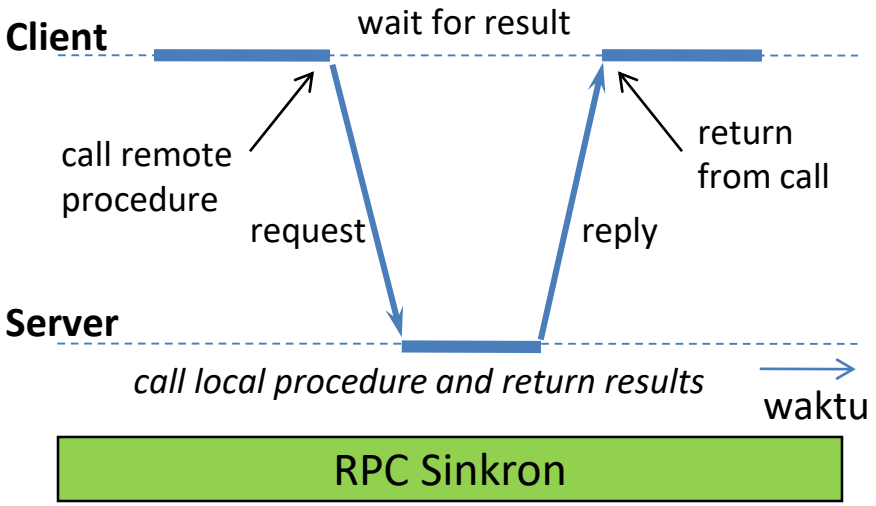
- Dalam kasus lokal, client dan server hidup atau mati bersama
- Dalam kasus *remote*, client menemui *jenis kegagalan* baru (lebih lanjut di pertemuan berikutnya)
  - Kegagalan jaringan (*network failure*)
  - Crash pada mesin Server
  - Crash pada proses Server
- Jadi, kode penanganan kegagalan harus lebih teliti (dan secara esensi pasti lebih kompleks)

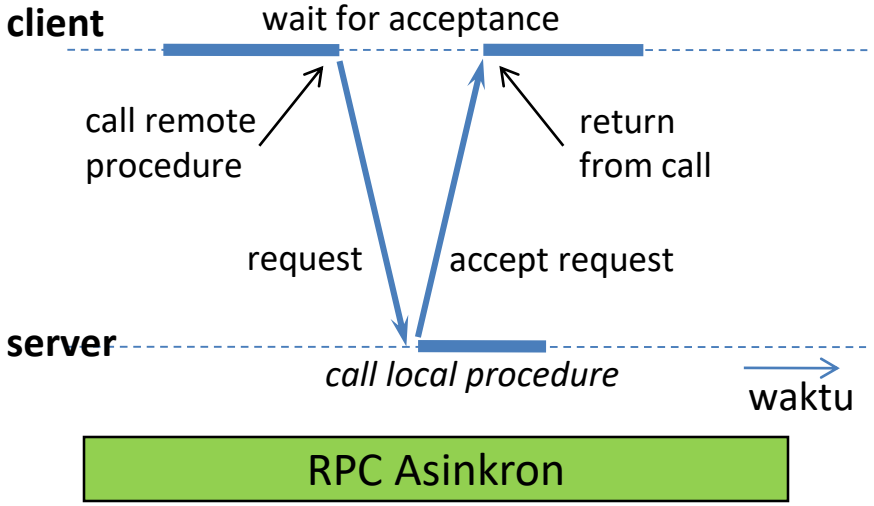
Tipe *Remote Procedure Call*

- *Remote procedure calls* dapat bersifat:
  - Sinkron (*Synchronous*)
  - Tak-Sinkron (*Asynchronous* atau Sinkron tertunda (*deferred)*)

RPC Sinkron vs. Tak-Sinkron

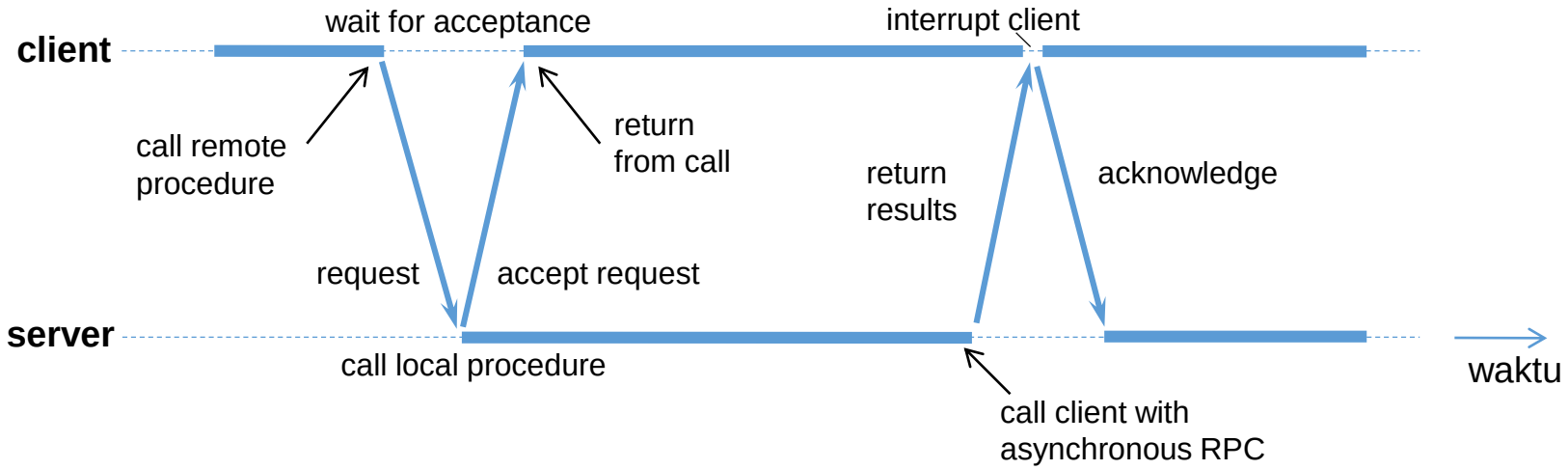
- Suatu RPC dengan *request-reply* sempurna memblokir *client* sampai *server* mengembalikan
  - *Blocking* membuang sumber daya di *client*
- RPC Asinkron digunakan jika client tidak memerlukan hasil dari server
  - *Server* dengan segera mengirimkan ACK balik ke *client*
  - *Client* melanjutkan eksekusi setelah ACK dari *server*





RPC Sinkron Tertunda

- RPC Asinkron juga bermanfaat saat suatu client ingin hasilnya, tetapi tidak mau diblokir sampai pemanggilan selesai
- Client menggunakan RPC sinkron tertentuda (*deferred synchronous*)
  - Satu RPC *request-response* dipecah menjadi dua RPC
  - Pertama, client memicu suatu RPC Asinkron pada server
  - Kedua, ketika selesai, server *calls-back* client untuk menyampaikan hasilnya



Remote Method Invocation (RMI)

- RMI mirip dengan RPC, tetapi dalam dunia obyek terdistribusi
  - Programmer dapat menggunakan kekuatan ekspresif dari *object-oriented programming*
  - RMI tidak hanya membolehkan pengiriman parameter nilai, tetapi juga menyertakan referensi obyek

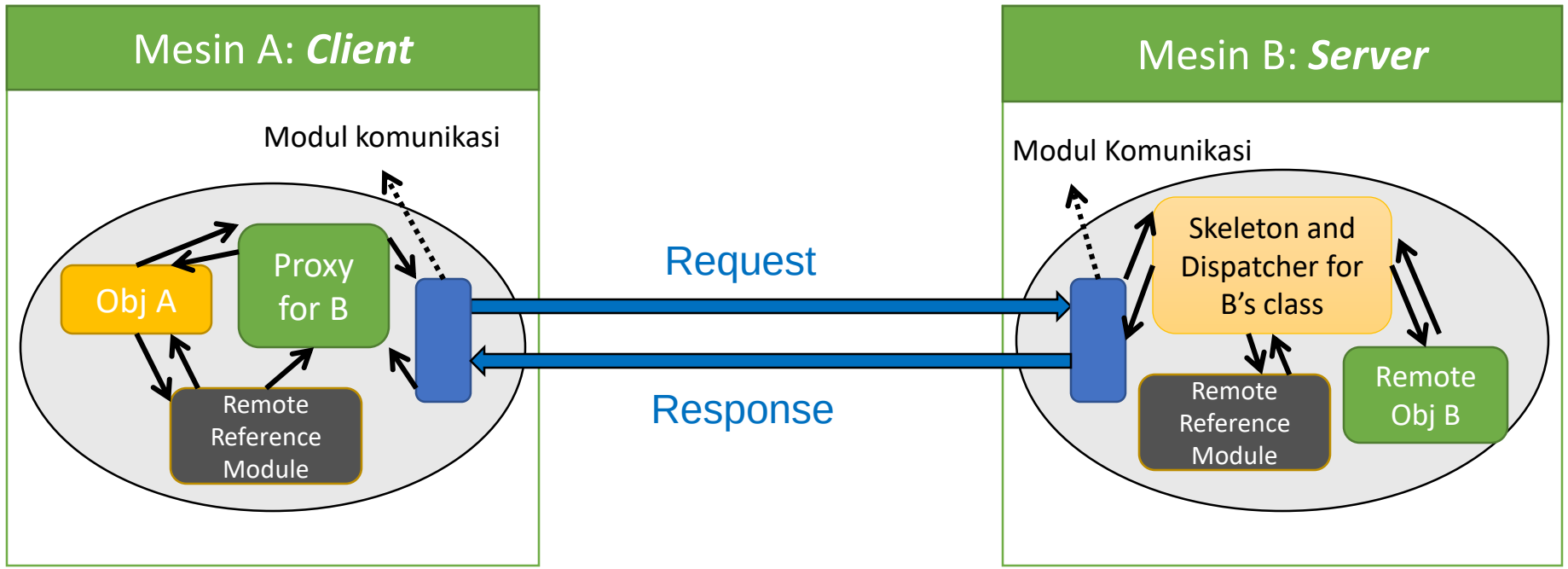
- Dalam RMI, pemanggilan obyek dapat meminta eksekusi suatu metode pada obyek jauh (*remote*).

Remote Object dan Modul Pendukung

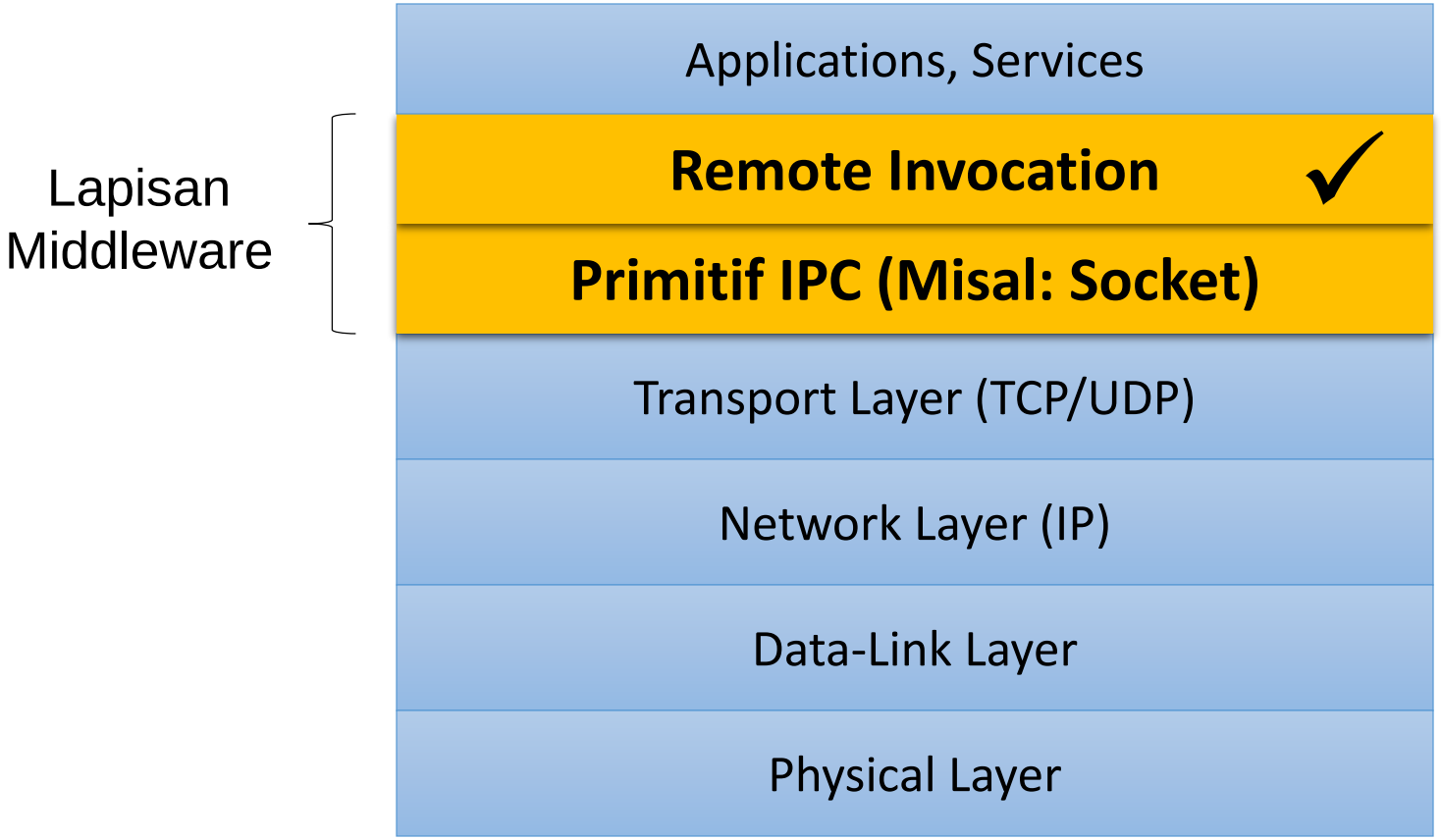
- Dalam RMI, obyek yang mempunyai metode-metode yang dapat dijalankan dari jauh dikenal sebagai "*remote objects*"
  - *Remote objects* mengimplementasikan *remote interfaces*

- Selama panggilan metode tertentu, sistem harus memutuskan apakah metode tersebut akan dipanggil pada obyek lokal atau jauh
  - Panggilan lokal harus dilakukan pada obyek lokal
  - Panggilan jauh harus dipanggil melalui *remote method invocation*
  - *Remote Reference Module* bertanggungjawab untuk menerjemahkan antara referensi obyek lokal dan jauh

Aliran Kendali RMI



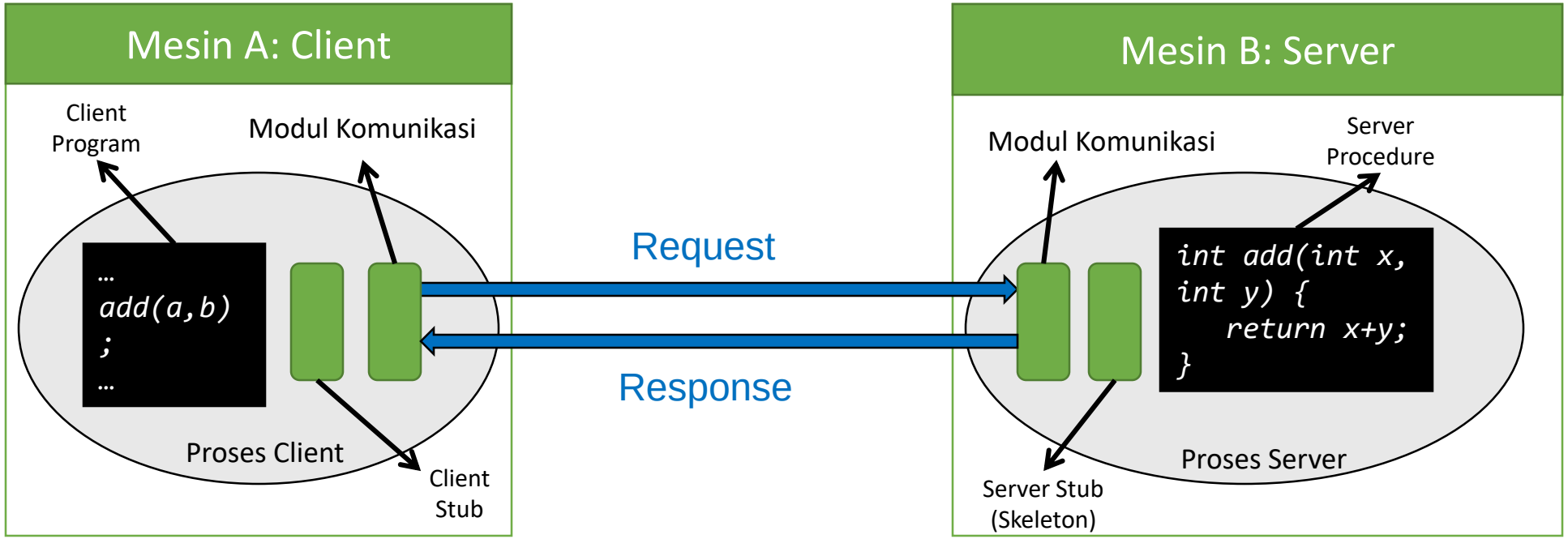
Lapisan Middleware



Remote Procedure Calls (RPC)

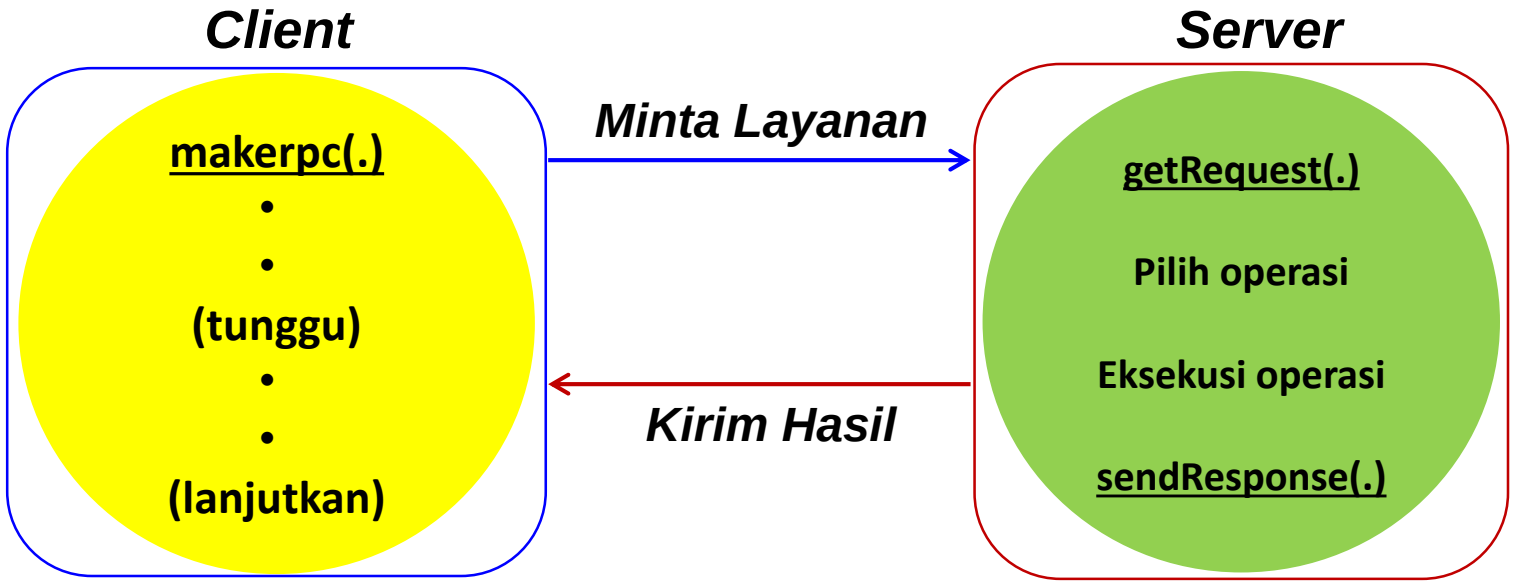
- RPC memungkinkan ***sender*** berkomunikasi dengan ***receiver*** mengunakan **panggilan prosedur** sederhana
  - Tidak ada komunikasi atau *message-passing* yang tampak bagi programmer

- Pendekatan RPC dasar:



Primitif Transport

- Modul komunikasi RPC (atau *transport*) sebagian besar didasarkan pada trio primitif komunikasi, *makerpc(.)*, *getRequest(.)* dan *sendResponse(.)*



Jenis Kegagalan

- Transport RPC dapat mengalami berbagai jenis kegagalan

Jenis Kegagalan	Penjelasan
• Crash Failure	• Server berhenti, tetapi telah bekerja dengan benar sampai ia dihentikan

• Omission Failure (kelalaian)
  • Receive Omission
  • Send Omission
	• Server gagal merespon request yang masuk
	  • Server gagal menerima *message* masuk
	  • Server gagal mengirimkan *message*

• Timing Failure (waktu)	• Respon server di luar interval waktu yang ditentukan

• Response Failure
  • Value Failure
  • State Transition Failure
	• Respon server tidak tepat
	  • Nilai dari respon salah
	  • Server menyimpang dari aliran kendali yang benar

• Byzantine Failure	• Server mungkin menghasilkan respon sembarang pada waktu sembarang

Mekanisme *Timeout*

- Untuk menangani kemungkinan suatu *message request* atau *reply* hilang, *makerpc(.)* dapat menggunakan suatu mekanisme *timeout*

- Ada berbagai opsi yang dapat dilakukan *makerpc(.)* setelah suatu *timeout*:
  - Segera kembali dengan suatu indikasi kepada client bahwa request telah gagal
  - Atau *retransmit* request tersebut berulang kali sampai suatu reply diterima atau server dianggap telah gagal.

- Bagaimana memilih suatu nilai timeout?
  - At best, menggunakan statistika empiris/teoritis
  - At worst, Tidak ada nilai yang bagus

Operasi *Idempotent*

- Dalam kasus ketika request message dikirim-ulang, server dapat menerimanya *lebih dari sekali*
- Ini dapat mengakibatkan suatu operasi dieksekusi lebih dari sekali untuk request yang sama
- *Keberatan:* Tidak setiap operasi dapat dieksekusi lebih dari sekali dan mendapatkan hasil sama sekali kalinya!
- Operasi yang dapat dieksekusi berulang dengan efek sama disebut operasi *idempotent*

Penyaringan Duplikat

- Untuk menghindari masalah dengan operasi, server harus:
  - Mengenali *messages* berturut-turut dari client yang "sama"
    - Secara monoton menaikkan *sequence numbers* dapat berlakukan
  - Menyaring (filter) duplikat

- Saat menerima suatu request "duplicate", server dapat Salah satu:
  - Mengenksekusi-ulang (re-execute) operasi lagi dan reply
    - Mungkin hanya untuk operasi *idempotent*

  - Atau menghindari eksekusi ulang operasi melalui *mempertahankan* outputnya dalam suatu file (atau log) history non-volatile
    - Mungkin perlu *transactional semantics (lebih lanjut di kuliah selanjutnya)*

Pilihan Implementasi

- Transport RPC dapat diimplementasikan dengan beberapa cara berbeda untuk menyediakan *delivery guarantees* berbeda. Pilihan utamanya adalah:
  1. **Pengulangan *request service*** (sisi *client*): Mengontrol apakah melakukan retransmisi *request service* sampai suatu balasan diterima atau server dianggap telah gagal
  2. **Penyaringan Duplikat** (*server side*): Mengontrol kapan retransmisi digunakan dan apakah menyaring request duplikat pada server
  3. **Penyimpanan Hasil** (*server side*): Mengontrol apakah memelihara *history* dari pesan hasil sehingga memungkinkan balasan yang hilang diretransmisi tanpa harus melakukan eksekusi ulang operasi pada server.

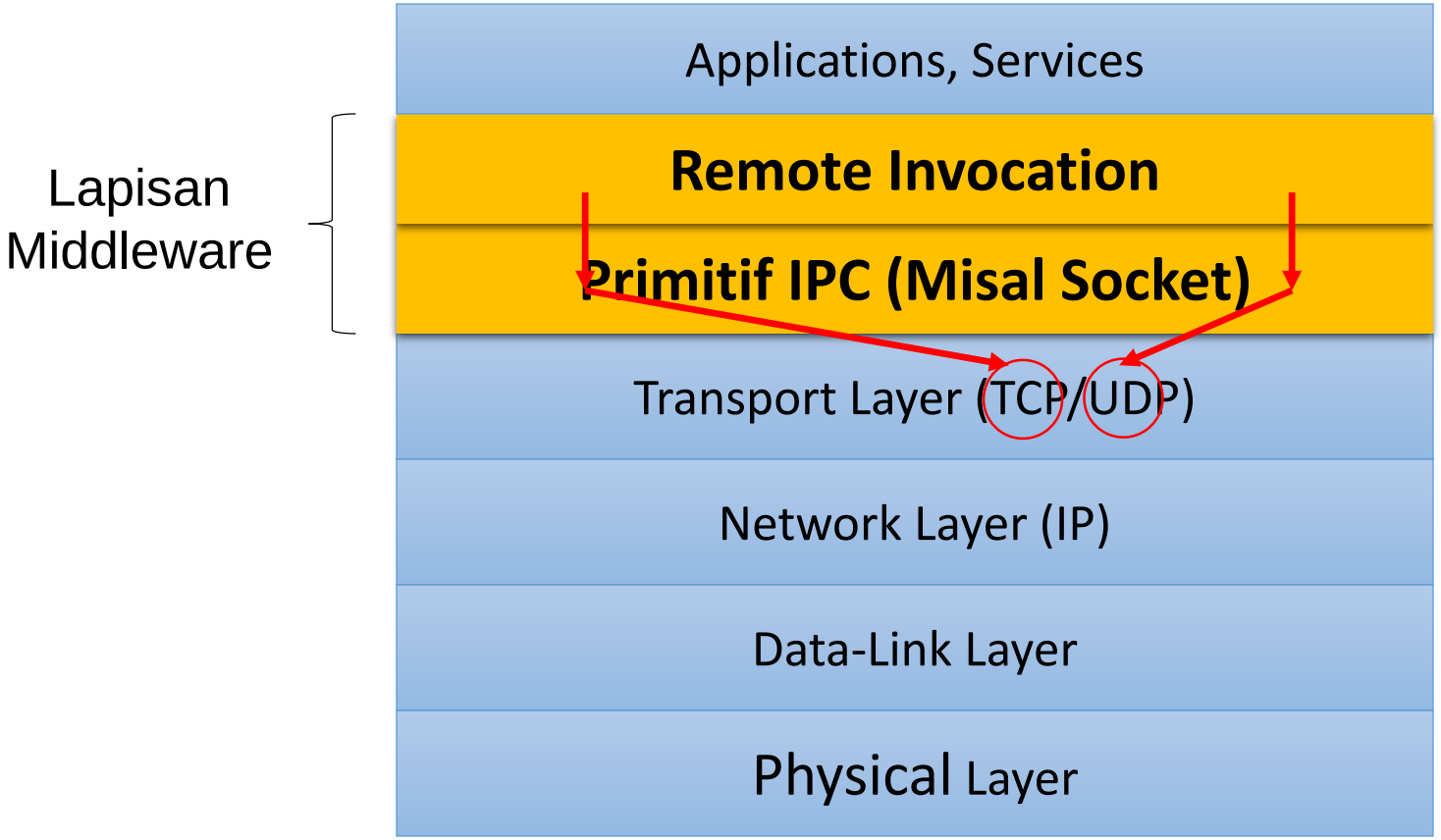
Semantik Panggilan RPC

- Kombinasi tindakan yang mengakibatkan beraneka semantika yang mungkin bagi reliabilitas RPC

Tindakan Fault Tolerance			Semantika Panggilan (Terkait dengan prosedur jauh)
Retransmisi Pesan Request	**Penyaringan Duplikat**	**Eksekusi-ulang Prosedur atau Retransmisi Balasan**	
No	N/A	N/A	***Maybe***
Yes	No	Eksekusi ulang Prosedur	***At-least-once***
Yes	Yes	Retransmisi Balasan	***At-most-once***

Idealnya, kita menginginkan semantik *exactly-once*!

Lapisan *Middleware*

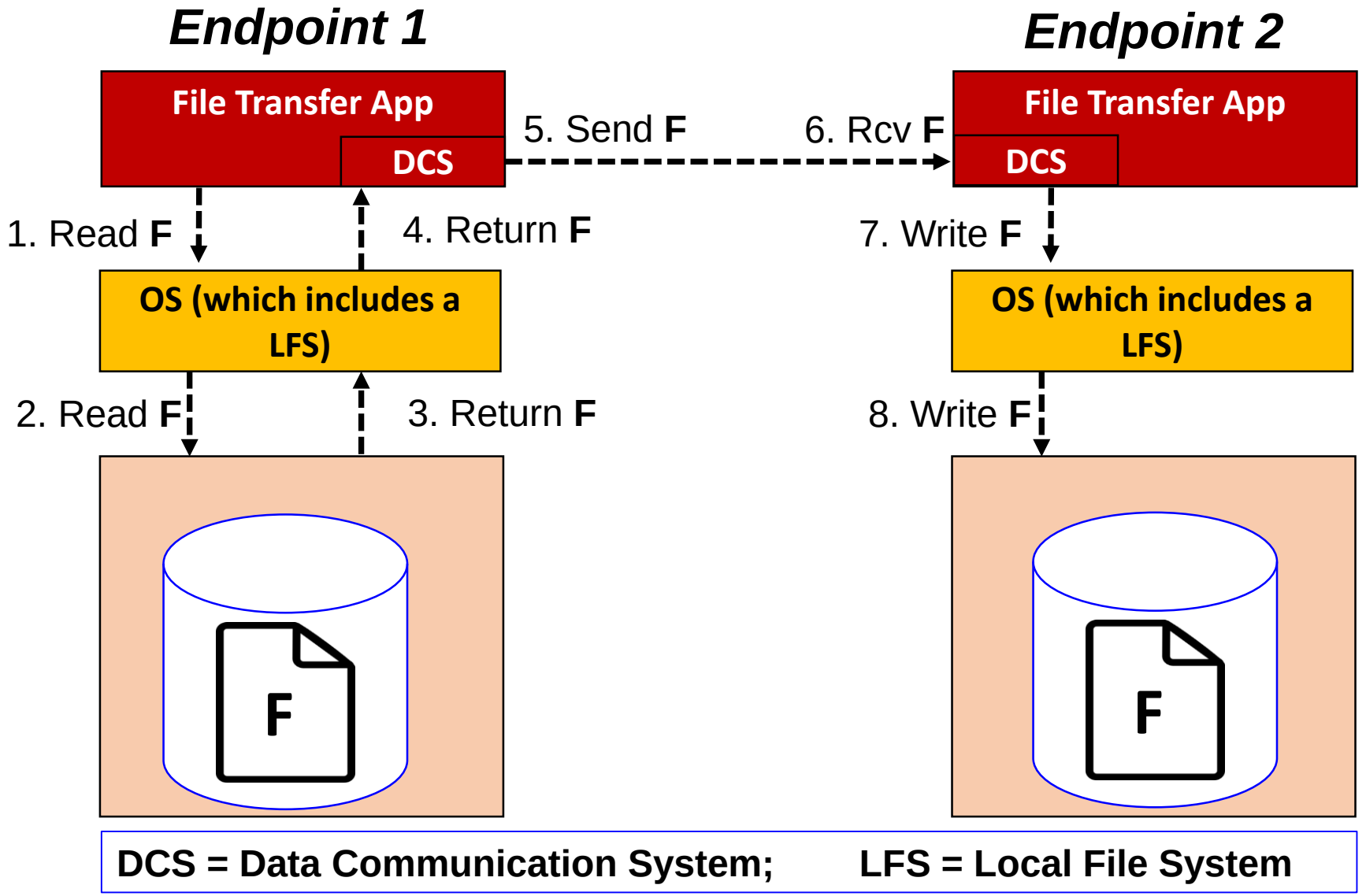


RPC di atas UDP atau TCP

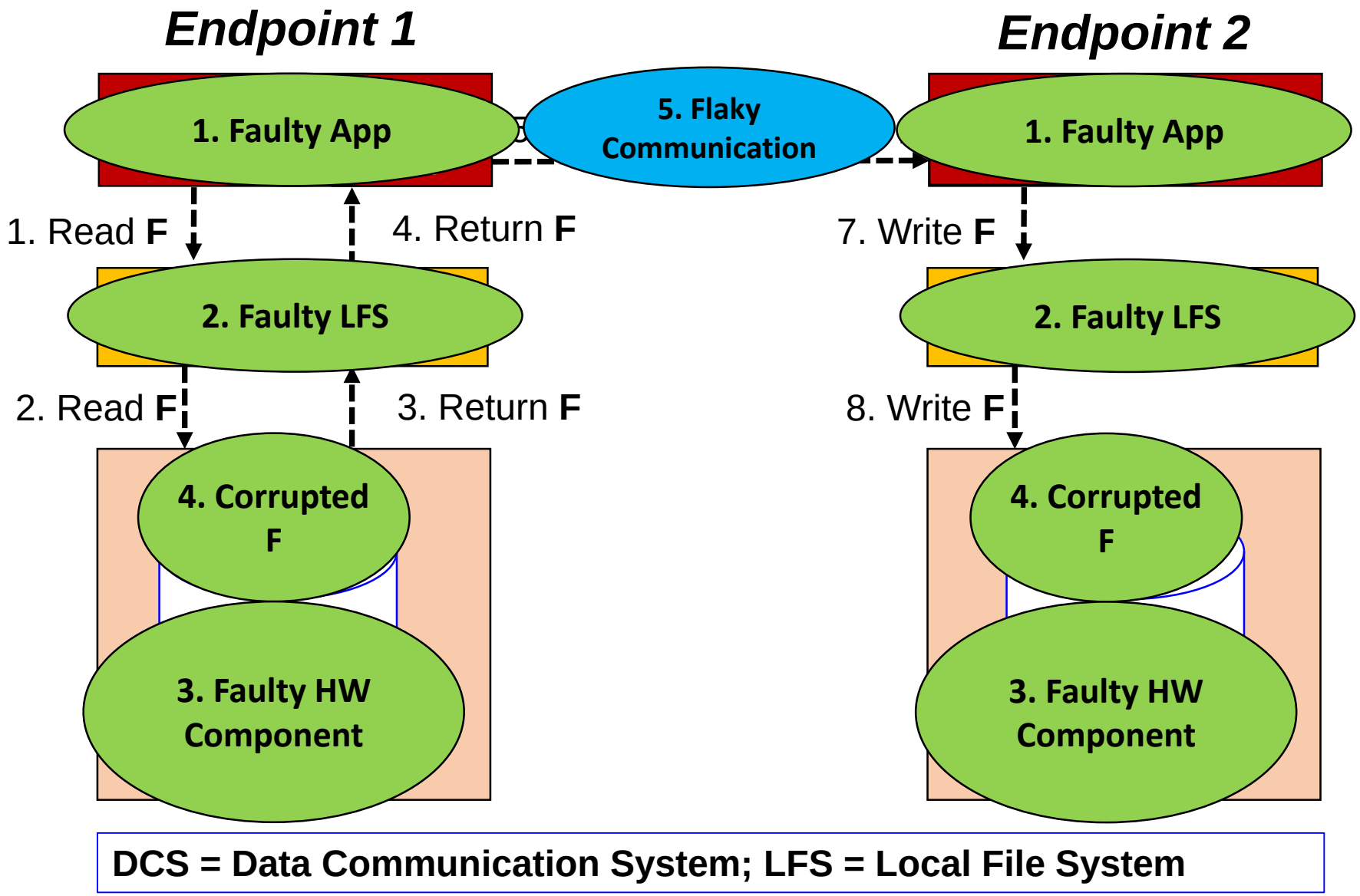
- Jika RPC di-layer-kan di atas UDP
  - Retransmisi akan/dapat ditangani dengan RPC

- Jika RPC di-layer-kan di atas TCP
  - Retransmisi akan ditangani oleh TCP
  - Apakah masih perlu melibatkan ukuran fault-tolerance di dalam RPC?
    - Ya, baca "*End-to-End Arguments in System Design*" oleh Saltzer *et. al.*

Careful File Transfer: Aliran



Careful File Transfer: Kemungkinan Ancaman



Careful File Transfer: End-To-End Check & Retry

- Endpoint 1 menyimpan bersama F suatu checksum C_A
- Setelah Endpoint 2 menulis F, ia membacanya lagi dari disk, menghitung checksum C_B, dan mengirimkannya balik ke Endpoint 1
- Endpoint 1 membandingkan C_A dan C_B
  - Jika $C_A = C_B$, laksanakan transfer file
  - Jika tidak, coba lagi transfer file

Careful File Transfer: End-To-End Check & Retry

- Berapa banyak “mencoba lagi”?
  - Biasanya 1 jika kegagalannya jarang
  - 3 kali percobaan ulang “dapat” mengindikasikan bahwa beberapa bagian sistem perlu perbaikan

- Bagaimana jika sistem komunikasi datanya menggunakan TCP?
  - Hanya threat 5 (Misal: packet loss dikarenakan *flaky communication*) dieliminasi
  - Frekuensi percobaan akan berkurang jika kesalahan disebabkan oleh sistem komunikasi
  - Lebih banyak *control* traffic, tetapi hanya bagian hilang dari F yang perlu dikirimkan ulang
  - Aplikasi transfer file masih perlu menerapkan tolok ukur keandalan end-to-end!

Careful File Transfer: *End-To-End Check & Retry*

- Bagaimana jika sistem komunikasi datanya menggunakan UDP?
  - Threat 5 (Misal: *packet loss* karena komunikasi serpihan) tidak dieliminasi - F perlu dihantarkan lagi oleh aplikasi jika tidak ada tindakan yang diambil untuk mengatasi ancaman ini
  - Frekuensi percobaan ulang dapat bertambah
  - Kinerja lebih buruk pada *flaky links*
  - *Aplikasi file transfer tersebut masih perlu menerapkan tolok-ukur reliabilitas end-to-end!*

Dalam dua kasus, aplikasi perlu menyediakan jaminan reliabilitas *end-to-end*!

Kuliah berikutnya…

- Arsitektur Sistem Terdistribusi