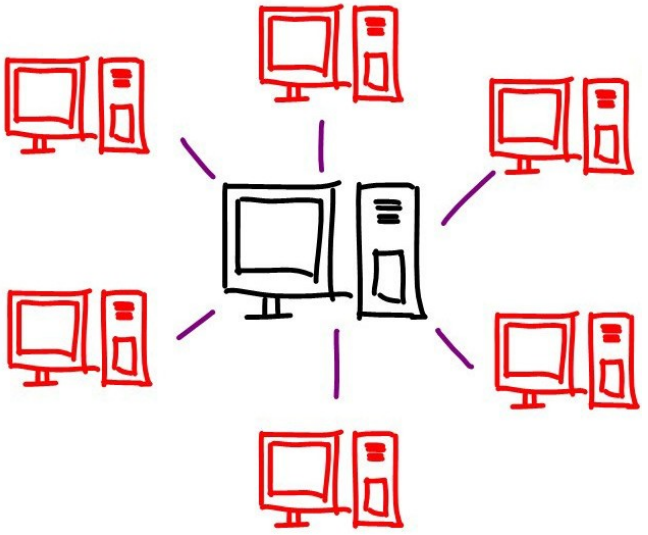
Sistem Terdistribusi (Distributed System)

Processes

Henry Saptono, M.Kom



Course/Slides Credits

- **Catatan**: *semua slide presentasi didasarkan pada buku yang dikembangkan oleh Andrew S. Tanenbaum dan Maarten van Steen. Judul buku mereka "Distributed Systems: Principles and Paradigms" (edisi 1 & 2). Dan juga berdasarkan slide presentasi yang dibuat oleh Maarten van Steen (VU Amsterdam, Dept. Computer Science)*

Pengantar

- Berbagai jenis **proses** memainkan perang penting dalam sebuah sistem terdistribusi
- Konsep suatu proses berasal dari bidang sistem operasi di mana ia umumnya didefinisikan sebagai **program dalam pelaksanaan** (eksekusi).
- Dalam sistem terdistribusi, kita mempertaruhkan kinerja. Disinilah peran penting dari **thread** untuk meningkatkan kinerja sistem terdistribusi

Pengantar threads

- Untuk memahami peran thread (utas) dalam sistem terdistribusi, penting untuk memahami apa proses itu, dan bagaimana keterkaitan proses dan thread (utas).
- Untuk menjalankan program, sistem operasi membuat sejumlah **prosesor virtual**, masing-masing untuk **menjalankan program** yang **berbeda**
- Untuk melacak prosesor virtual ini, sistem operasi memiliki tabel proses, yang berisi entri untuk menyimpan nilai register CPU, peta memori, file terbuka, informasi akun, ijin akses dll. Bersama-sama, entri-entri ini membentuk konteks proses (process context).

Pengantar threads

- Prosesor 2x virtual dibangun dalam perangkat lunak, di atas prosesor fisik:
  - **Prosesor**: Menyediakan seperangkat instruksi bersama dengan kemampuan secara otomatis menjalankan serangkaian instruksi tersebut.
  - **Thread**: Prosesor perangkat lunak minimal yang dalam konteksnya serangkaian instruksi dapat dieksekusi. Menyimpan konteks thread berarti menghentikan eksekusi saat ini dan menyimpan semua data yang diperlukan untuk melanjutkan eksekusi di tahap selanjutnya.
  - **Proses**: Prosesor perangkat lunak yang dalam konteksnya satu atau lebih utas dapat dijalankan. Mengeksekusi utas, berarti mengeksekusi serangkaian instruksi dalam konteks utas itu.

Pengalihan Konteks (Context Switching)

- **Contexts**
  - **Processor context:** Kumpulan minimal nilai yang disimpan dalam register prosesor yang digunakan untuk pelaksanaan (eksekusi) serangkaian instruksi (mis., stack pointer, addressing registers, program counter).
  - **Thread context**: Kumpulan minimal nilai yang disimpan dalam register dan memori, digunakan untuk pelaksanaan serangkaian instruksi (mis.,, Konteks prosesor, status).

Pengalihan Konteks (Context Switching)

- **Process context**: Kumpulan minimal nilai yang disimpan dalam register dan memori, digunakan untuk pelaksanaan thread (mis., thread context, nilai-nilai register MMU).

Pengalihan Konteks (Context Switching)

- Thread berbagi ruang alamat yang sama. Pergantian konteks thread dapat dilakukan sepenuhnya independen dari sistem operasi.
- Proses switching umumnya lebih mahal karena melibatkan mendapatkan OS di loop, yaitu, perangkap ke kernel.
- Membuat dan menghancurkan thread jauh lebih murah daripada melakukannya untuk proses.

Thread dan Sistem Operasi

- isu utama
  - Haruskah kernel OS menyediakan thread (utas), atau haruskah itu diimplementasikan sebagai paket tingkat pengguna?

Thread dan Sistem Operasi

- **Solusi ruang pengguna (User-space solution)**
  - Semua operasi dapat sepenuhnya **ditangani dalam satu proses** ⇒ implementasi bisa sangat efisien.
  - **Semua** layanan yang disediakan oleh kernel dilakukan **atas nama proses di mana sebuah thread berada** ⇒ jika kernel memutuskan untuk memblokir sebuah thread, seluruh proses akan diblokir.
  - Thread digunakan ketika ada banyak peristiwa eksternal: blok thread atas dasar per-peristiwa

Thread dan Sistem Operasi

- **Kernel solution**
  - Seluruh idenya adalah membuat kernel berisi implementasi dari paket thread. Ini berarti bahwa semua operasi kembali sebagai panggilan sistem (system calls)
    - Operasi yang memblokir thread bukan lagi sebuah masalah: **kernel menjadwalkan thread lain yang tersedia** dalam proses yang sama.
    - Menangani peristiwa eksternal mudah: **kernel** (yang menangkap semua peristiwa) **menjadwalkan thread yang terkait** dengan peristiwa tersebut
    - Masalahnya adalah hilangnya efisiensi karena fakta bahwa setiap operasi thread memerlukan perangkap ke kernel.

Thread dan Sistem Terdistribusi

- Multithreaded Web client
  - Menyembunyikan latensi jaringan:
    - Web browser memindai halaman HTML yang masuk, dan menemukan bahwa lebih banyak file harus diambil.
    - Setiap file diambil oleh thread terpisah, masing-masing melakukan permintaan HTTP (blocking).
    - Ketika file masuk, browser menampilkannya.

Thread dan Sistem Terdistribusi

- **Multiple requests-response call** ke mesin lain (RPC)
  - Sebuah klien melakukan beberapa panggilan sekaligus, masing-masing dengan thread berbeda.
  - Kemudian menunggu hingga semua hasil dikembalikan.
  - Catatan: jika panggilan ke server yang berbeda, kita mungkin memiliki kecepatan linier.

Thread dan Sistem Terdistribusi

- **Meningkatkan kinerja**
  - Memulai thread jauh lebih murah daripada memulai proses baru.
  - Memiliki server single-threaded menghalangi peningkatan skala sederhana ke sistem multiprosesor.
  - Seperti halnya klien: menyembunyikan latensi jaringan dengan bereaksi terhadap permintaan berikutnya sementara yang sebelumnya sedang dijawab.

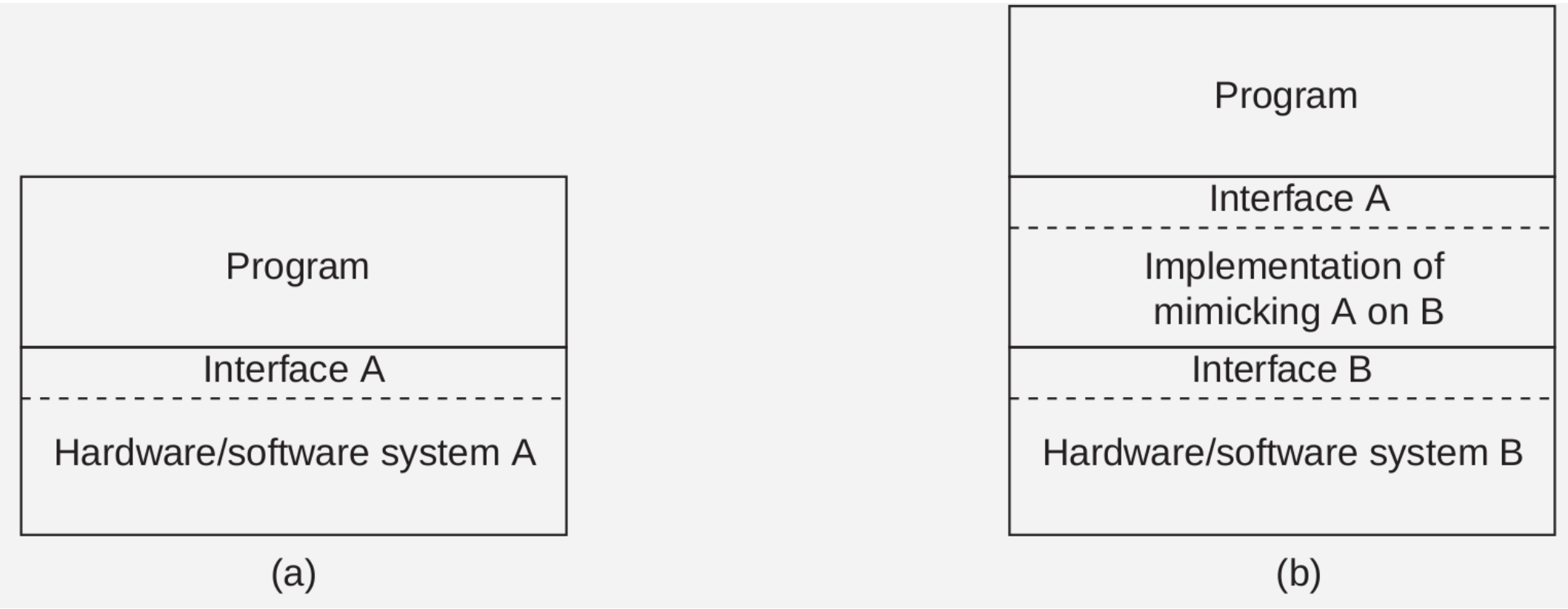
Thread dan Sistem Terdistribusi

- **Struktur yang lebih baik**
  - Sebagian besar server memiliki permintaan I / O yang tinggi. Dengan menggunakan panggilan pemblokiran yang sederhana dan mudah dipahami, akan menyederhanakan struktur keseluruhan.
  - Program multithread cenderung lebih kecil dan lebih mudah dipahami karena aliran kontrol yang disederhanakan.

Virtualization

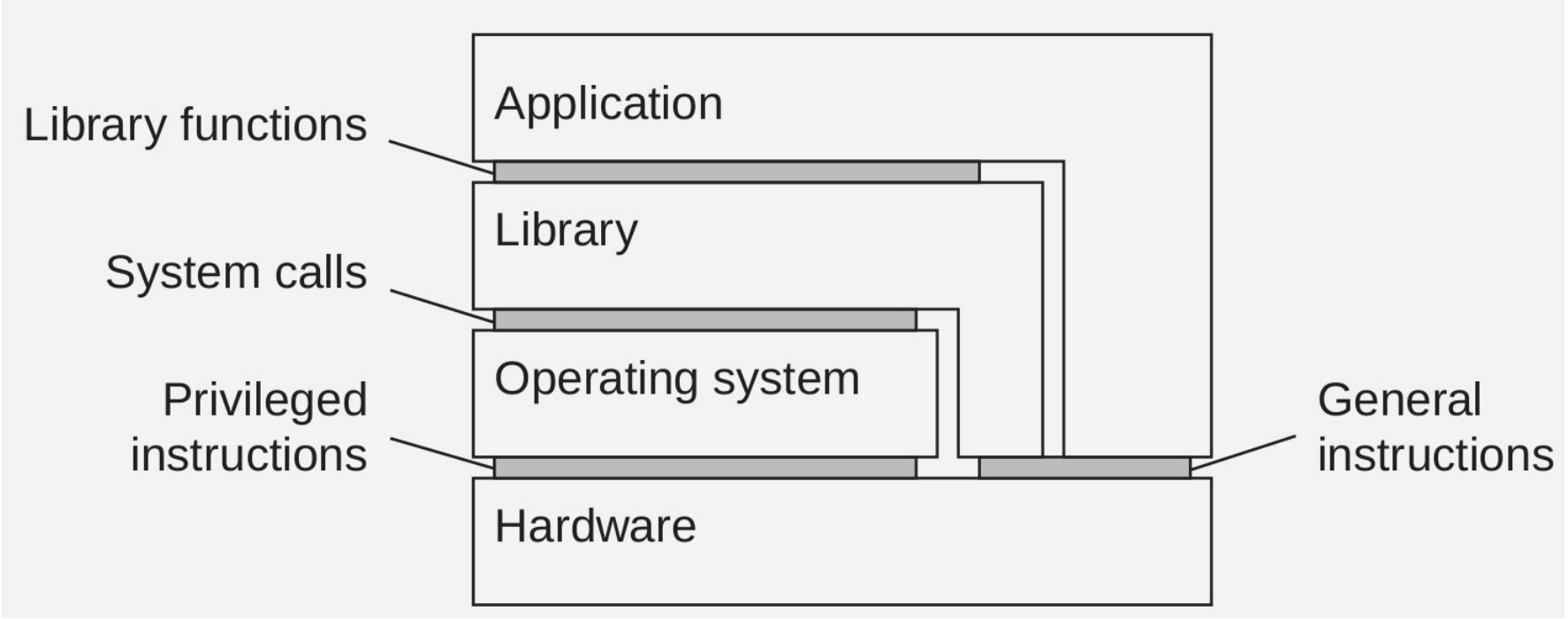
- Virtualisasi menjadi semakin penting:
  - Perubahan perangkat keras lebih cepat daripada perangkat lunak
  - Kemudahan portabilitas dan migrasi kode
  - Isolasi komponen yang gagal atau diserang

Virtualization

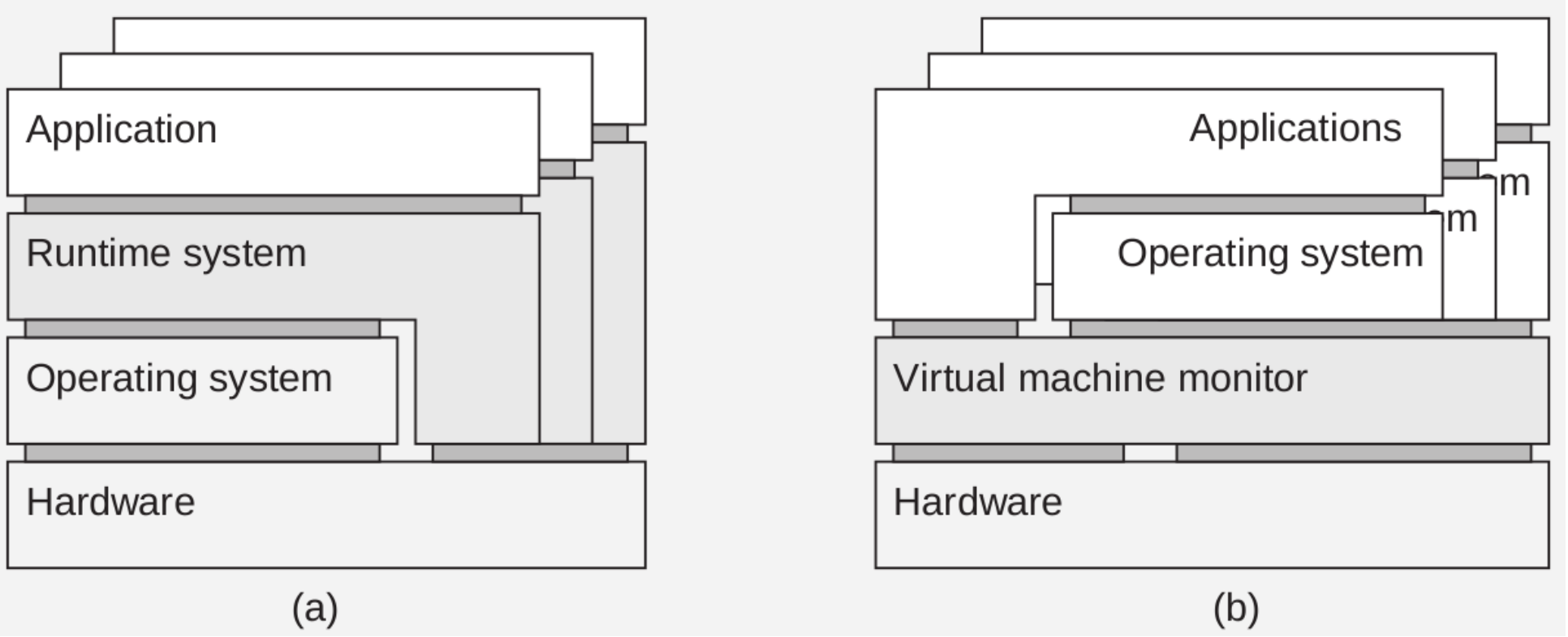


Arsitektur VM

- Virtualisasi dapat terjadi pada level yang sangat berbeda, sangat tergantung pada antarmuka yang ditawarkan oleh berbagai komponen sistem:



VM Proses versus VM Monitor



VM Proses versus VM Monitor

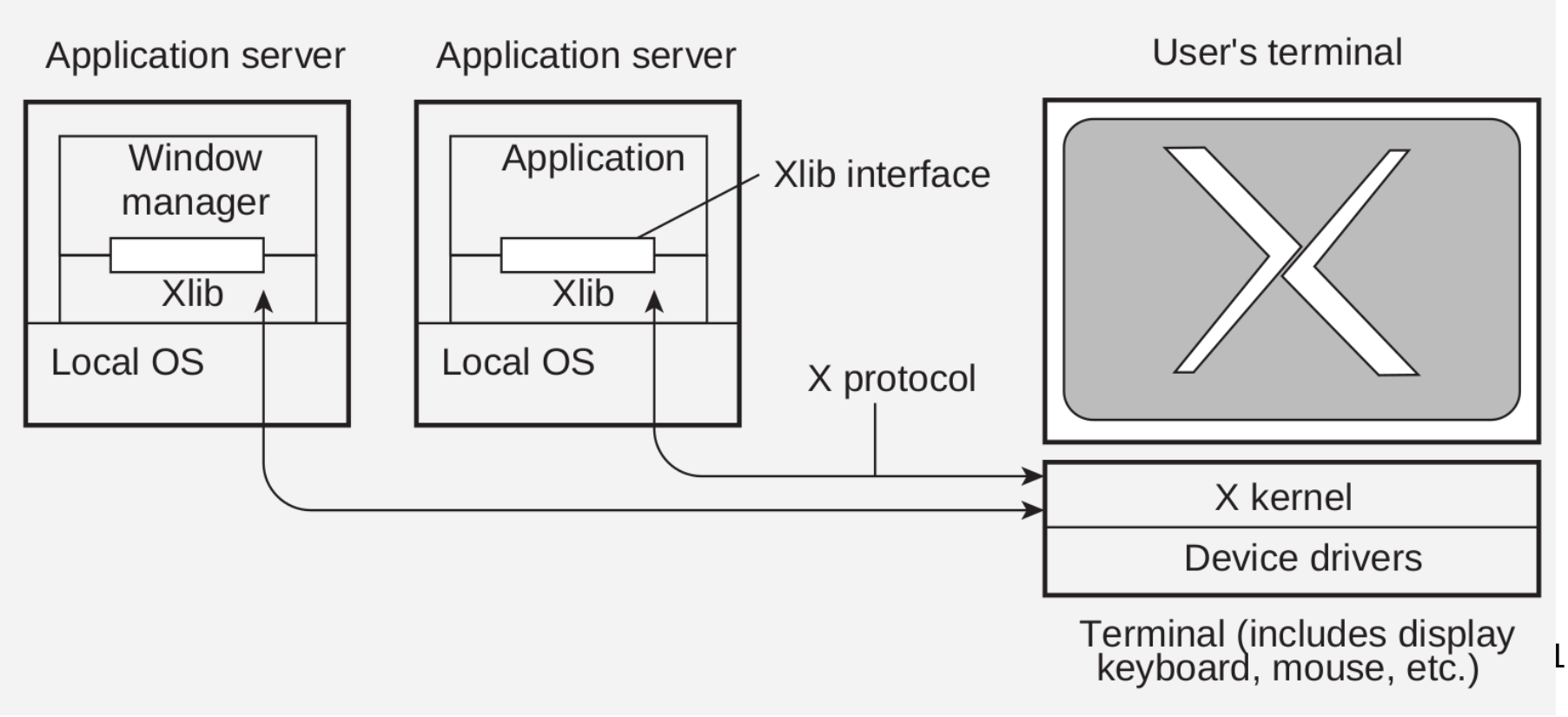
- VM Proses: Suatu program dikompilasi ke kode perantara (portabel), yang kemudian dieksekusi oleh sistem runtime (Contoh: Java VM).
- VM Monitor: Lapisan perangkat lunak terpisah meniru set instruksi perangkat keras ⇒ sistem operasi lengkap dan aplikasinya (Contoh: VMware, VirtualBox)

VM Monitor pada sistem operasi

- VMM berjalan di atas sistem operasi yang ada.
  - melakukan terjemahan biner: saat menjalankan aplikasi atau sistem operasi, menterjemahkan instruksi ke instruksi mesin yang mendasarinya.
  - Membedakan instruksi instruksi sensitif: perangkap ke kernel asli (pikirkan system calls, atau instruksi istimewa).
  - Instruksi sensitif diganti dengan panggilan ke VMM.

Clients: User Interfaces

- Bagian utama dari perangkat lunak sisi klien difokuskan pada antarmuka pengguna (grafis).



Client-Side Software

- Umumnya dirancang untuk transparansi distribusi
  - access transparency: client-side stubs for RPCs
  - location/migration transparency: let client-side software keep track of actual location
  - replication transparency: multiple invocations handled by client stub:
  - failure transparency: can often be placed only at client (we’re trying to mask server and communication failures).

Server: General organization

- Server adalah proses yang menunggu permintaan layanan masuk di alamat transportasi tertentu. Dalam praktiknya, ada pemetaan satu-ke-satu antara sebuah port dan layanan.

ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
	24	any private mail system
smtp	25	Simple Mail Transfer
login	49	Login Host Protocol
sunrpc	111	SUN RPC (portmapper)
courier	530	Xerox RPC

Server: General organization

- Jenis-jenis server
  - **Superservers**: Server yang mendengarkan beberapa port, mis., Menyediakan beberapa layanan independen. Dalam praktiknya, ketika permintaan layanan masuk, mereka memulai subproses untuk menangani permintaan tersebut (UNIX inetd)
  - **Iterative vs. concurrent servers**: Server iterative hanya dapat menangani satu klien pada satu waktu, berbeda dengan server concurrent

Servers and state

- **Stateless servers**
  - Jangan pernah menyimpan informasi yang akurat tentang status klien setelah menangani permintaan:
    - Jangan merekam apakah file telah dibuka (cukup tutup kembali setelah akses)
    - Jangan menjanjikan untuk membatalkan cache klien
    - Jangan melacak klien Anda

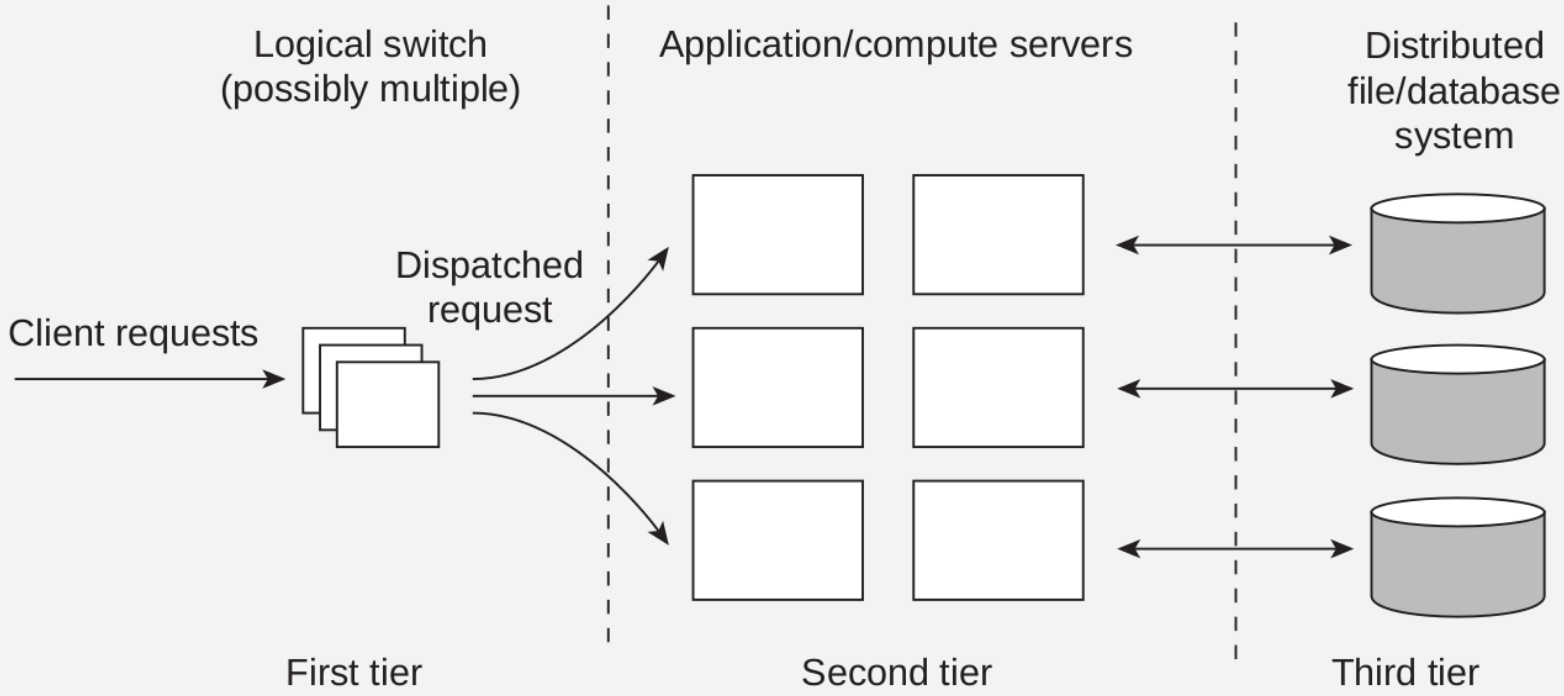
Servers and state

- **Konsekwensi**
  - Klien dan server sepenuhnya independen
  - Inkonsistensi state karena klien atau server crash berkurang
  - Kemungkinan hilangnya kinerja karena, mis., Server tidak dapat mengantisipasi perilaku klien
- Pertanyaan:
  - Does connection-oriented communication fit into a stateless design?

Servers and state

- **Stateful servers**
  - Melacak status dari klien-klien:
    - Rekam bahwa file telah dibuka, sehingga pengambilan awal dapat dilakukan
    - Mengetahui data mana yang telah di-cache oleh klien, dan memungkinkan klien untuk menyimpan salinan lokal dari data bersama
- Kinerja server stateful bisa sangat tinggi, asalkan klien diizinkan untuk menyimpan salinan lokal. Ternyata, keandalan bukanlah masalah besar.

Server clusters: three different tiers



Crucial element

The first tier is generally responsible for passing requests to an appropriate server.

Request Handling

- Pengamatan:
  - Memiliki tier pertama menangani semua komunikasi dari / ke cluster dapat menyebabkan kemacetan.
- Solusi:
  - Beragam, tetapi yang populer adalah TCP-handoff

Request Handling