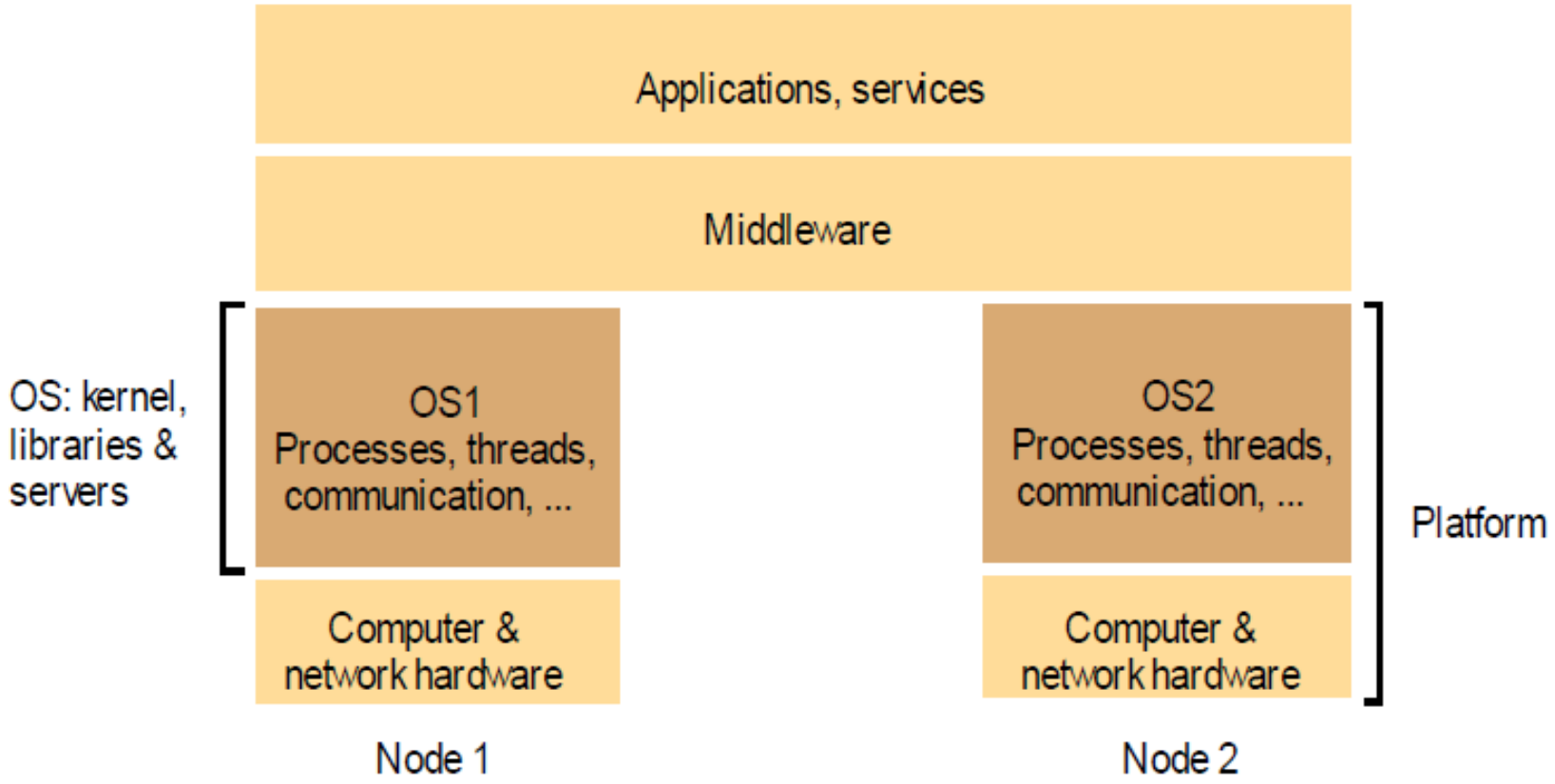
Fakultas Informatika
School of Computing
Telkom University

CSG3L3
SISTEM TERDISTRIBUSI

Topik 7 : Dukungan Sistem Operasi

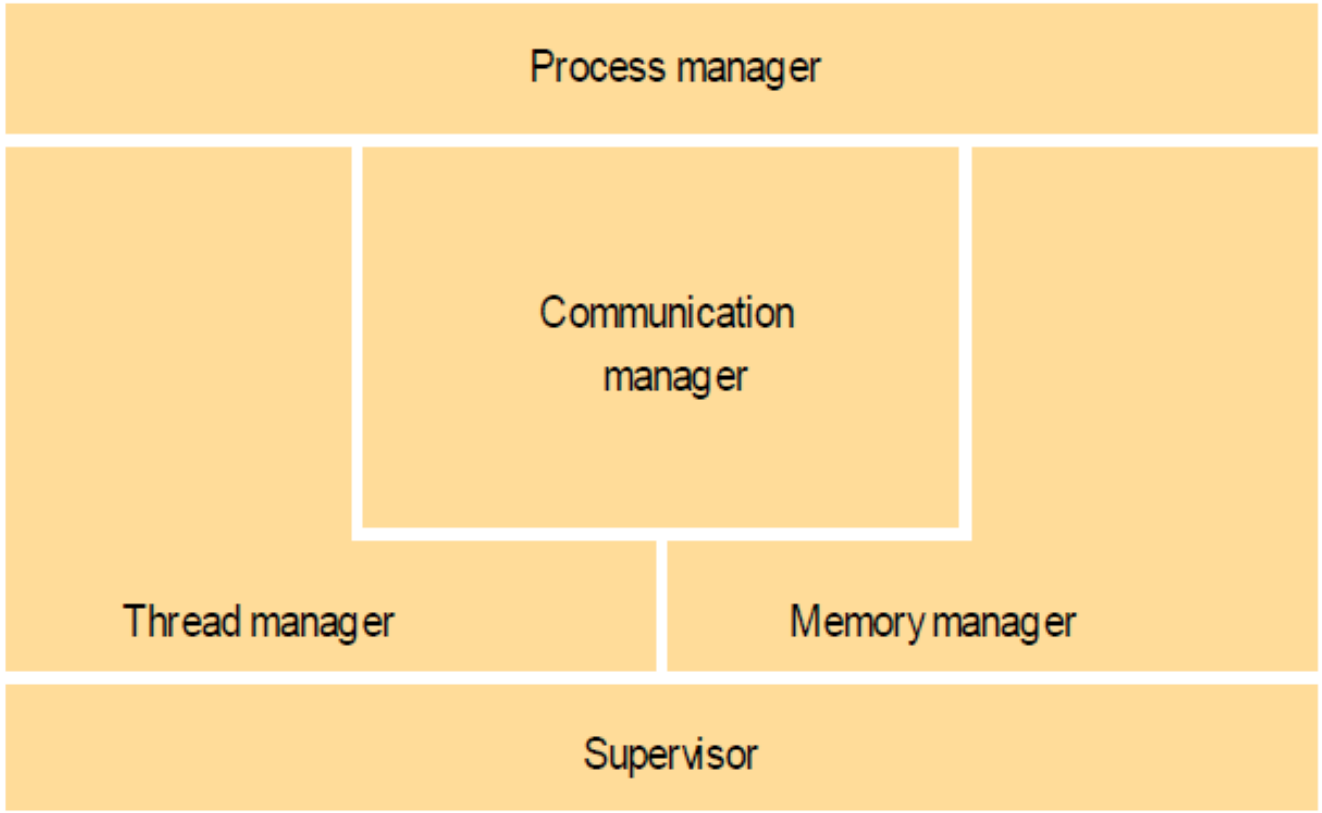
Layer Sistem Operasi



Tujuan Mempelajari Topik

- Mengamati dampak dari mekanisme sistem operasi tertentu dan kombinasi OS dan *middleware* terhadap kinerja *resource sharing* pada sistem terdistribusi
- Mengetahui secara detail peran sistem operasi pada sistem terdistribusi

Peran Utama Sistem Operasi

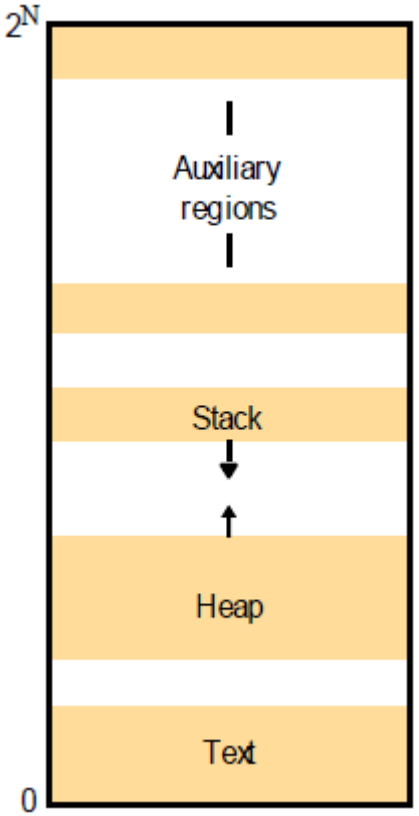


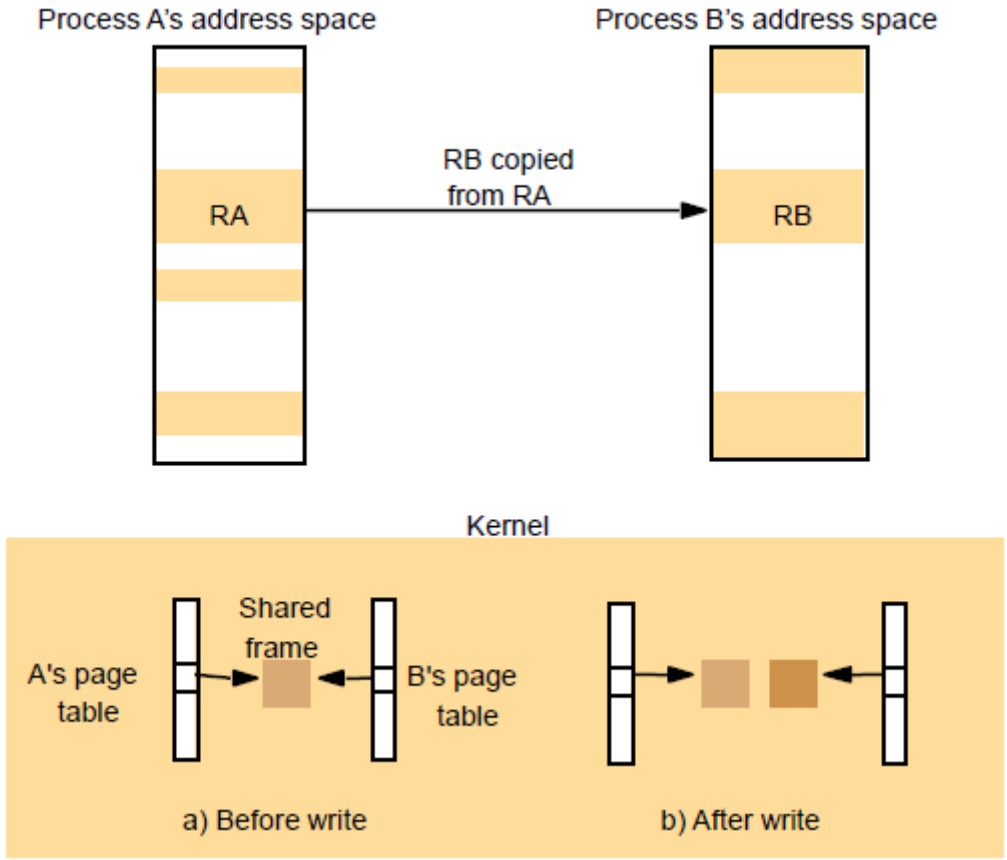
- *Process manager* – pembentukan dan pengoperasian tiap proses. Proses merupakan sebuah unit yang memerlukan manajemen *resource* yang terdiri dari alokasi memori untuk satu buah *thread* atau lebih
- *Thread manager* – Umumnya komunikasi antar *thread* terjadi di satu komputer, namun bisa juga dilakukan pada proses *remote* (antar proses pada komputer yang berbeda)
- *Memory manager* – Mangatur alokasi memori fisik maupun virtual
- *Supervisor* – mengendalikan adanya *interrupt*, *system call*, dan eksepsi lainnya.

Apa tujuan dari adanya *thread*?

- *Thread* dapat "diciptakan" dan "dimusnahkan" secara dinamis sesuai kebutuhan
- Menggunakan banyak *thread* dapat meningkatkan derajat konkurensi pemrosesan sehingga memungkinkan adanya proses input output yang saling *overlap*

Apa nama skema di bawah ini? Bagaimana cara kerjanya?

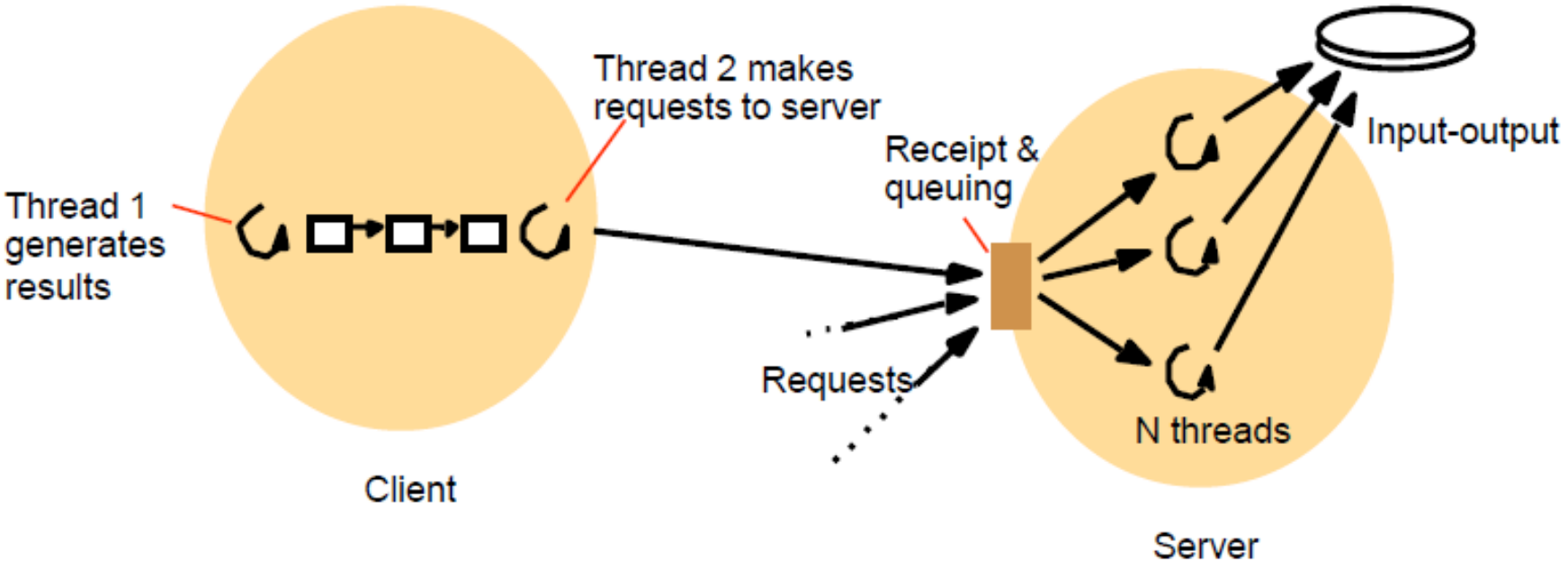




Beda *Stack* dan *Heap*?

- *Stack* adalah sebuah bagian pada program yang terbentuk (diinisialisasi) oleh nilai yang tersimpan dalam file *binary* sebuah program, yang dapat diperluas menuju alamat memori virtual yang lebih tinggi
- *Heap* kebalikan dari *stack* yang bisa diperluas menuju alamat memori virtual yang lebih rendah

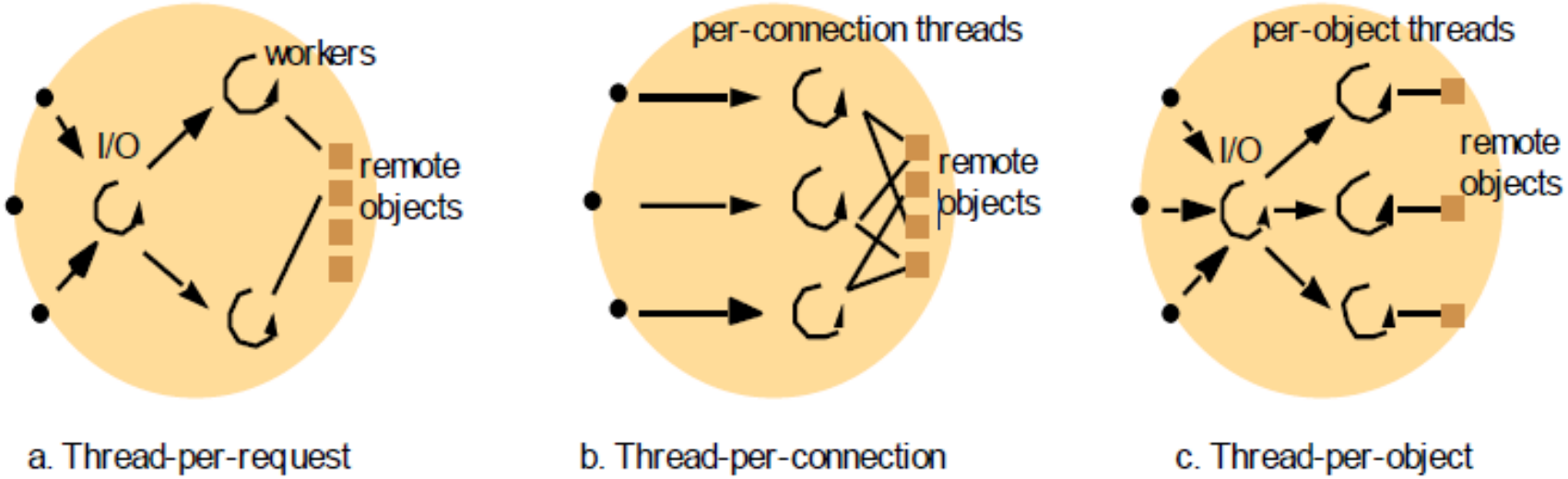
Client-Server Menggunakan *Thread*



Penjelasan *thread* pada client-server

- Jika waktu pemrosesan 2 ms dan waktu server memproses I/O 8 ms, maka *turnaround time* tiap *request* klien adalah 2+8 = 10 ms.
- *Throughput* server menjadi 100 *request* per detik.
- Jika menggunakan 2 *thread*, 1 untuk proses I/O dan 1 untuk memproses *request* berikutnya maka *throughput* akan naik menjadi 1000/8 = 125 *request* per detik.
- *Throughput* akan terus meningkat jika ditambah mekanisme *caching* dan *shared-memory multiprocessor*.

Alternatif Arsitektur *Threading* pada Server



a. Thread-per-request

b. Thread-per-connection

c. Thread-per-object

Detail Arsitektur *Threading* pada Server

- *Thread-per-request* – *thread* I/O membuat satu *thread* "*worker*" untuk tiap *request* yang datang dan "lenyap" setelah selesai meneksekusi *request* bersama dengan *remote object* yang sesuai. Arsitektur ini tidak didesain untuk *shared-queue.*
  - Kelebihan: *Throughput* maksimal
  - Kekurangan: bisa terjadi *overhead* karena terlalu banyak operasi penambahan dan penghancuran *thread*

Detail Arsitektur *Threading* pada Server (cont'd)

- *Thread-per-connection* – *Thread* "*worker*" akan terbentuk ketika klien membuat koneksi dengan server dan "lenyap" pada saat klien memutus koneksi. *Request* juga akan langsung ditangani oleh *thread* ini dan langsung berasosiasi dengan *remote object* yang dibutuhkan.
- *Thread-per-object* – *Thread* yang berasosiasi dengan *remote object* tertentu. *Thread* I/O menerima *request* dan membuat sebuah antrian objek yang siap dieksekusi oleh "*worker*".

Keuntungan/Kerugian 2 Arsitektur Terakhir

- Keuntungan: Meminimalkan kebutuhan manajemen *overhead*
- Kerugian: Klien mengalami jeda (delay) ketika sebuah *thread "worker"* melayani banyak *request* namun *thread "worker"* lain sedang tidak melayani *request* apapun (*idle*)

Kondisi (*State*) yang Berhubungan dengan *Thread* dan Lingkungan Eksekusi

Lingkungan Eksekusi	Thread
Address space tables	*Saved processor registers* – pada saat *thread* berstatus BLOCKED
Communication Interfaces (membuka beberapa file)	Prioritas dan status eksekusi (BLOCKED / RUNNABLE)
Semaphores dan objek sinkronisasi lain	Informasi mengenai *software interrupt handling*
Daftar *identifier* dari sebuah *thread*	*Identifier* dari sebuah lingkungan eksekusi
Pages dari *address space* di dalam memori dan HW *cache entries*	

Java Thread

- Thread(ThreadGroup g, Runnable t, String name)
  - Membuat sebuah *thread* baru dalam kondisi SUSPENDED dalam sebuah grup diidentifikasi oleh nama *thread*.
- setPriority(int newPriority), getPriority()
  - Memberikan dan mengambil nilai prioritas pada/dari sebuah *thread.*
- run()
  - Sebuah method yang dijalankan pada objek tertentu atau pada objek itu sendiri (implements Runnable).

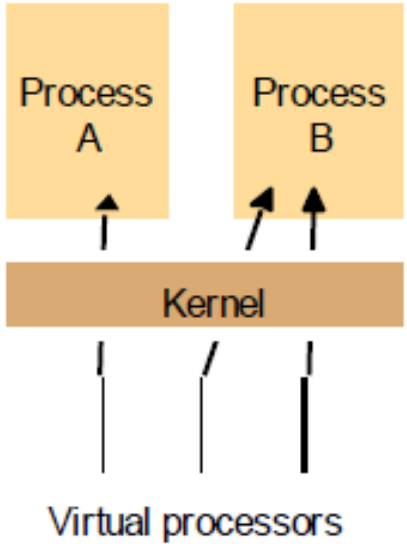
Java Thread (cont'd)

- start()
  - Mengubah *state* pada sebuah *thread* dari SUSPENDED ke RUNNABLE.
- sleep(int milisecond)
  - Menjadikan *thread* ke dalam status SUSPENDED selama waktu yang ditentukan.
- yield()
  - Mengubaj *thread* ke status READY dan meng-*invoke* scheduler.
- destroy()
  - Menghancurkan *thread.*

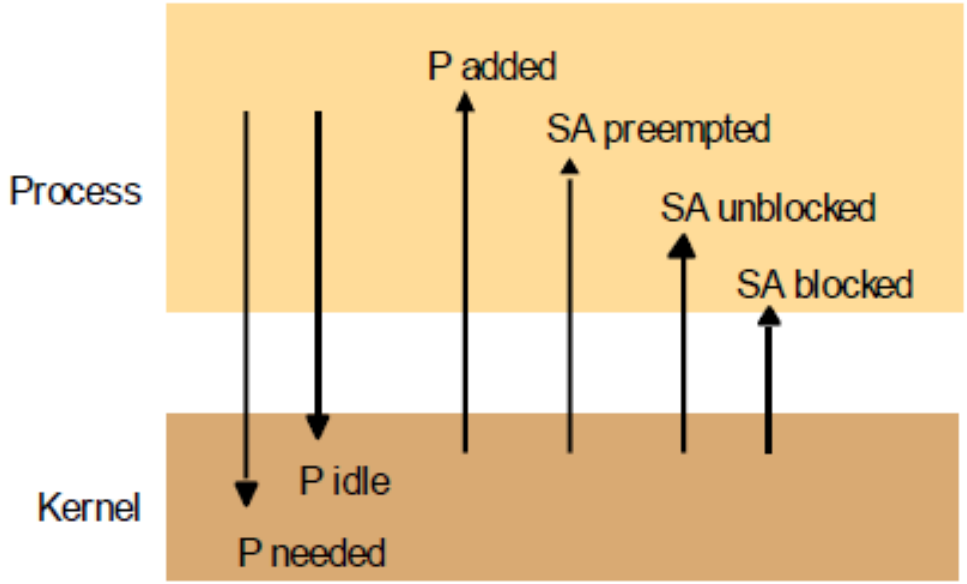
Java Thread Synchronization Calls

- thread.join(int milisecond)
  - Memblok *calling thread* selama waktu yang ditentukan hingga *thread* dimusnahkan.
- thread.interrupt()
  - Mengembalikan *thread* dari kondisi *blocking method call*, misal: sleep().
- object.wait(long milisecond, int nanosecond)
  - Memblok *calling thread* menunggu sampai ada perintah notify() atau notifyAll(), atau mengalami *interrupt*, atau waktu yang telah ditentukan habis.
- object.notify(), object.notifyAll()
  - Membangunkan *thread* yang mengeksekusi perintah wait().

Aktivasi *Scheduler*



Pengalokasian prosesor virtual ke dua buah proses A dan B

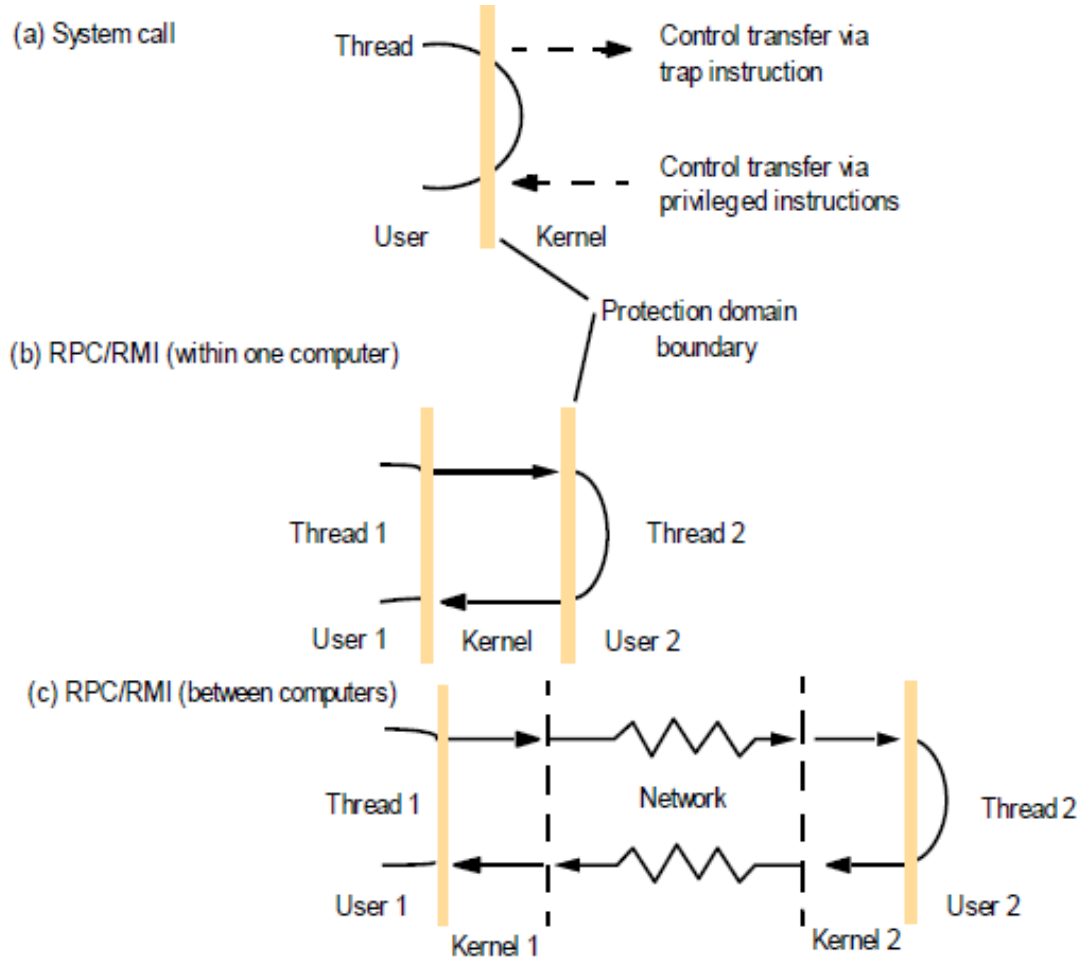


Event antara *user-level scheduler* dan kernel. **P** = prosesor; **SA** = Scheduler Activation

Penjelasan dari *Scheduler Activation*

- Dua even yang menyebabkan proses memberi notifikasi kepada kernel yaitu: saat prosesor virtual dalam kondisi *idle* dan sebuah proses membutuhkan prosesor virtual tambahan.
- Empat even yang menyebabkan kernel menotifikasi proses yaitu: kernel mengalokasikan prosesor virtual kepada proses, SA terblok di kernel, SA di-*unblock*, dan *SA preempted.*

Proses Invokasi Antar *Address Space*



Grafik *Delay* RPC v.s. Ukuran Data

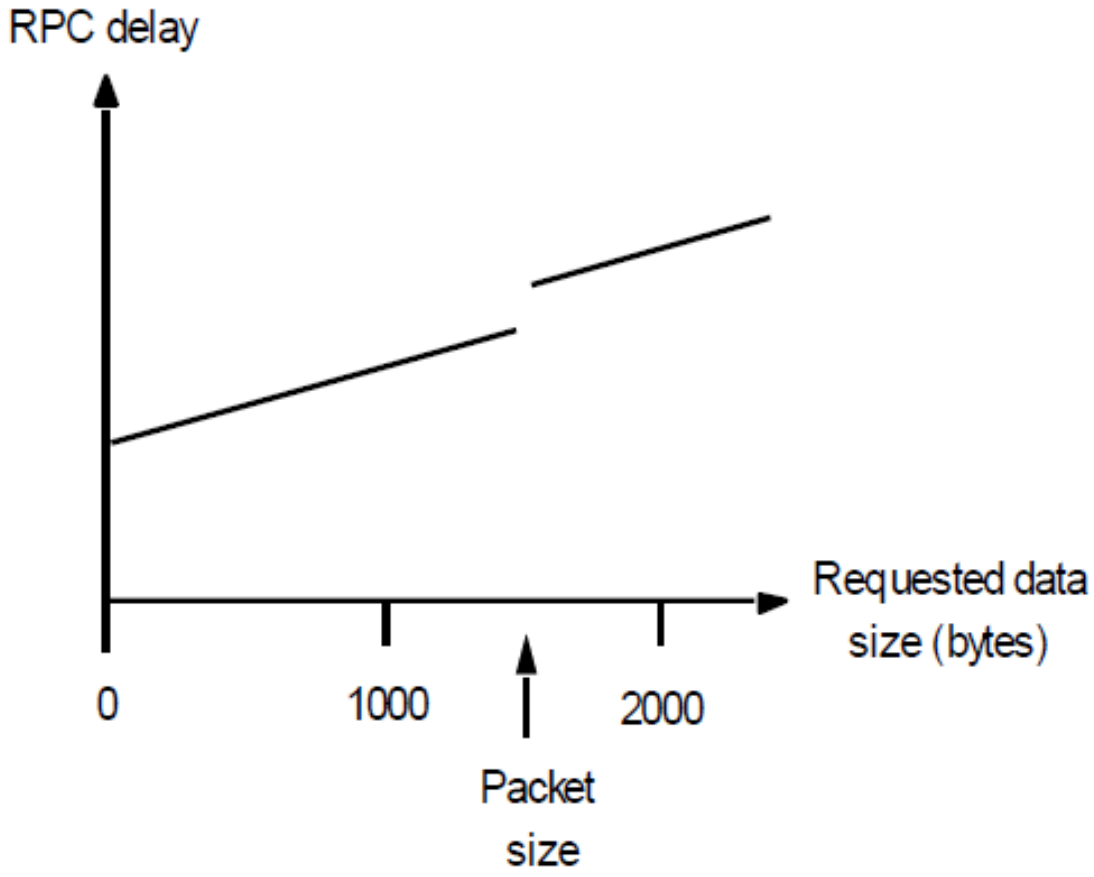
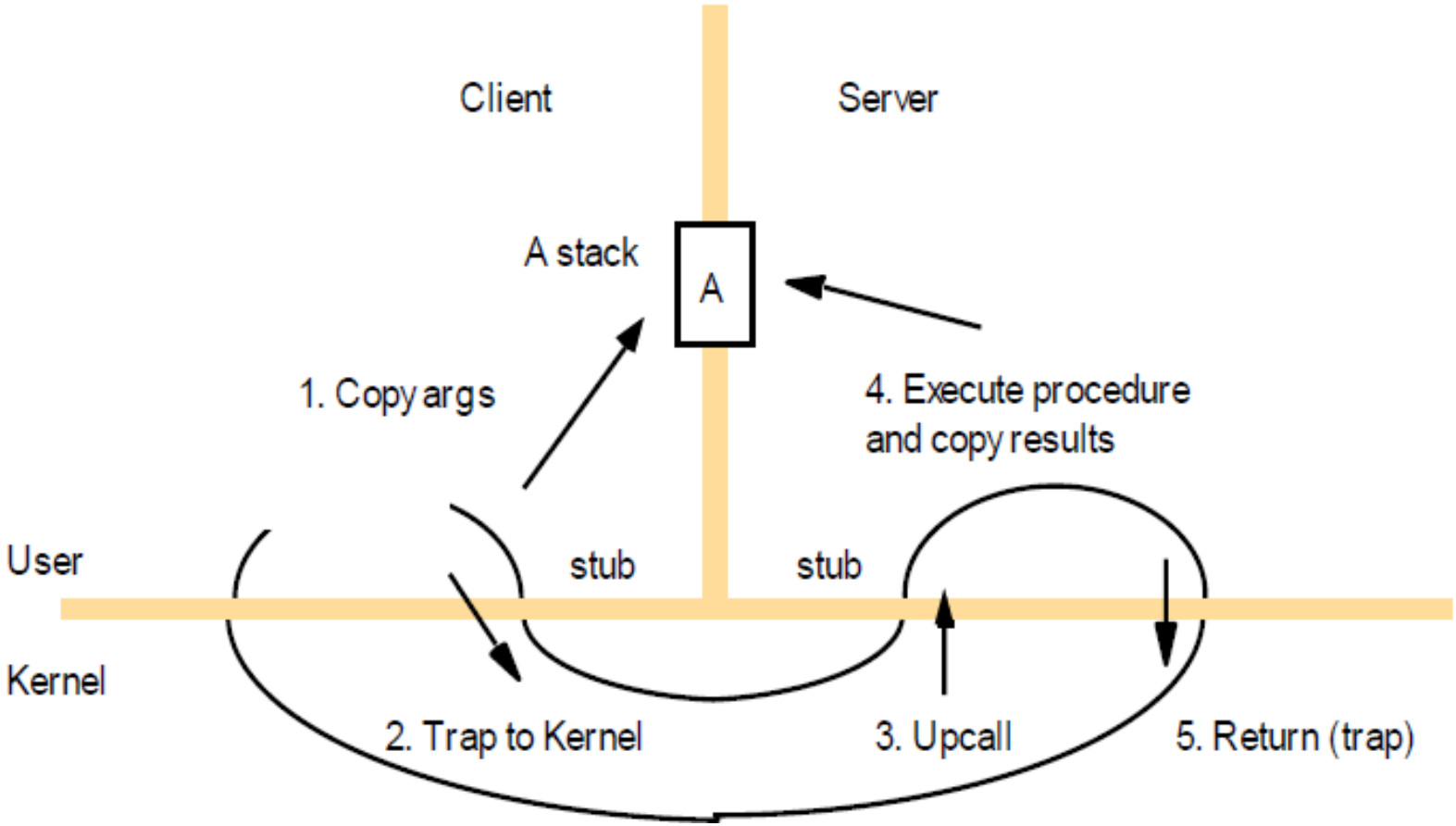
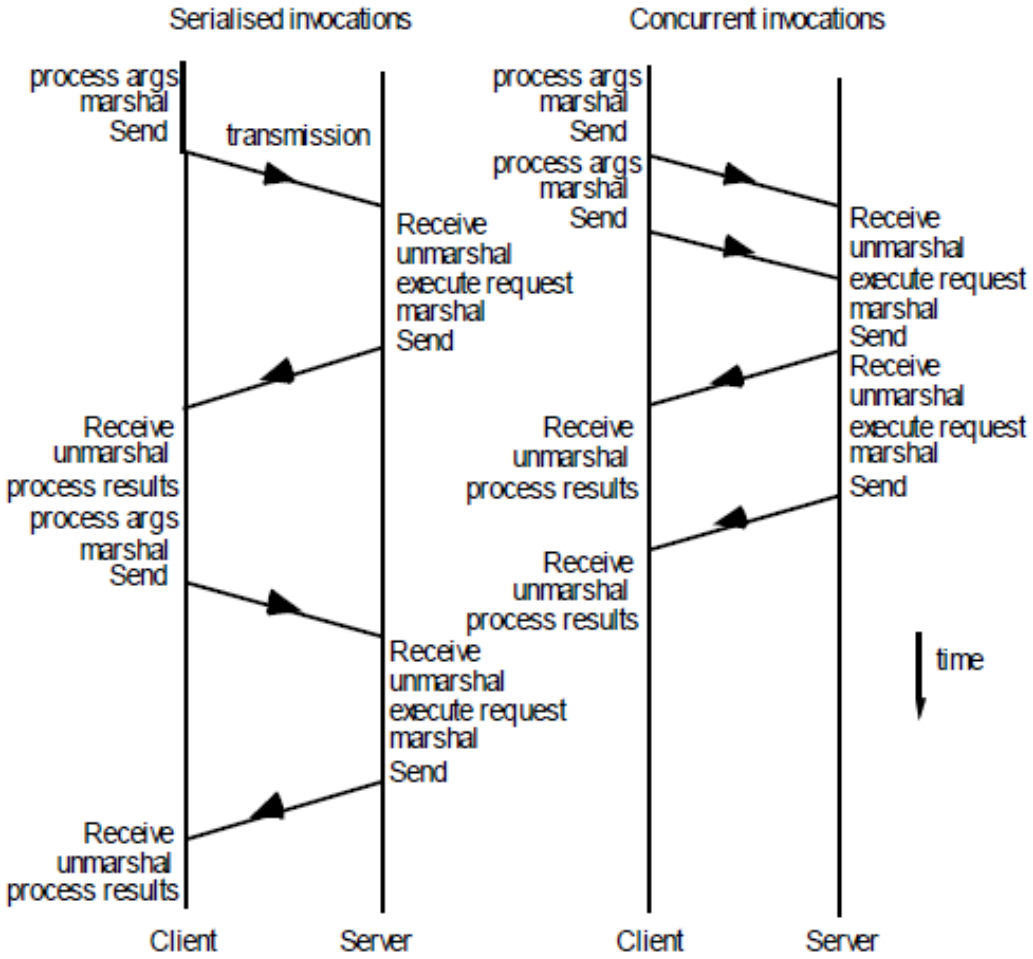


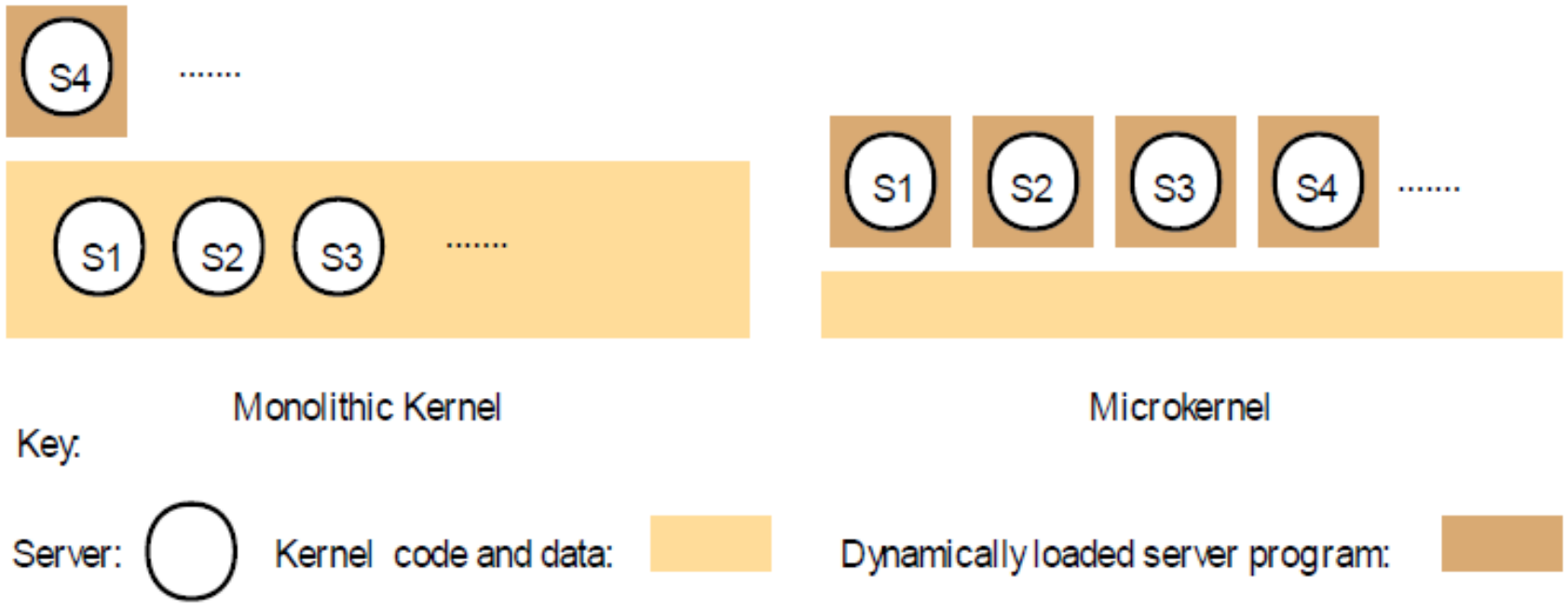
Diagram Sebuah Proses *Lightweight* RPC



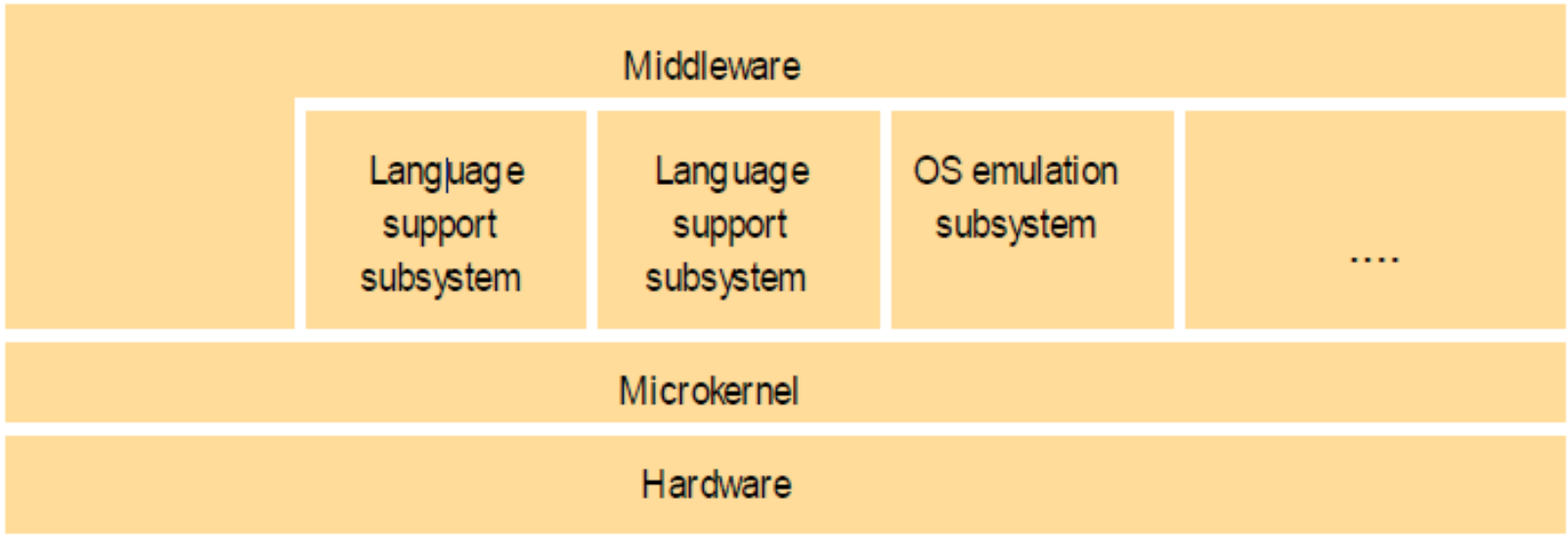
Serial v.s Konkuren *Invocation*



Monolithic Kernel v.s. *Microkernel*

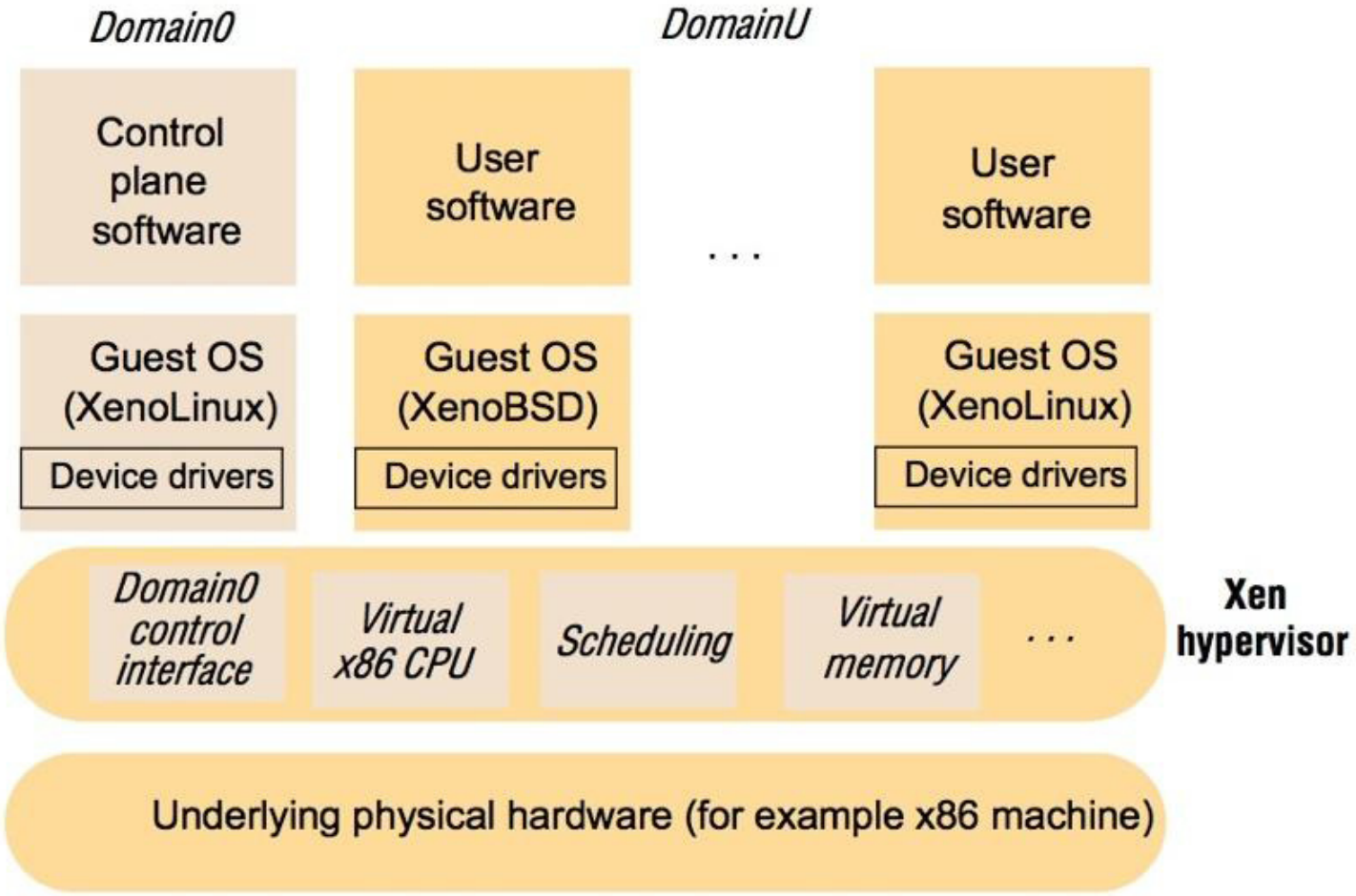


Peran *Microkernel*

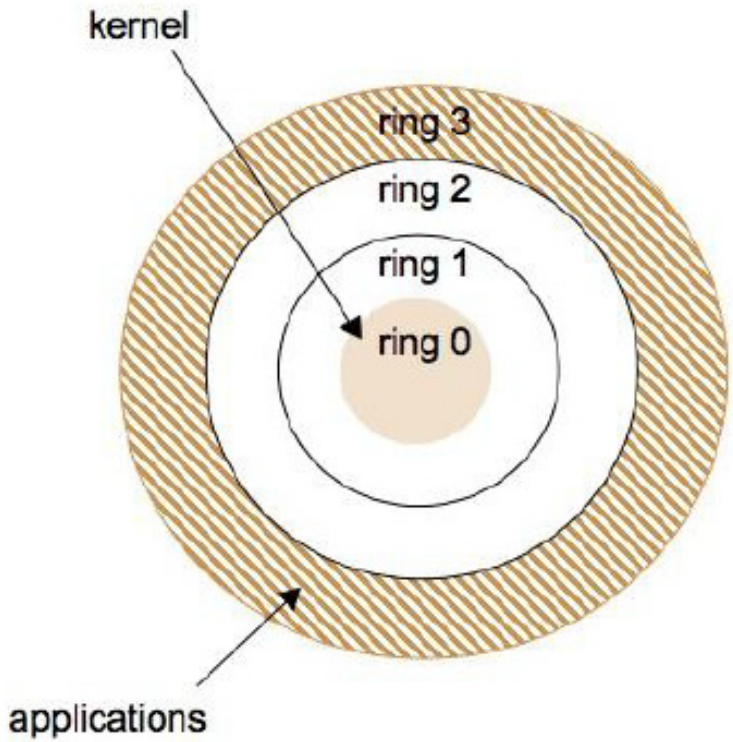


Microkernel mendukung *middleware* dari sisi subsistem

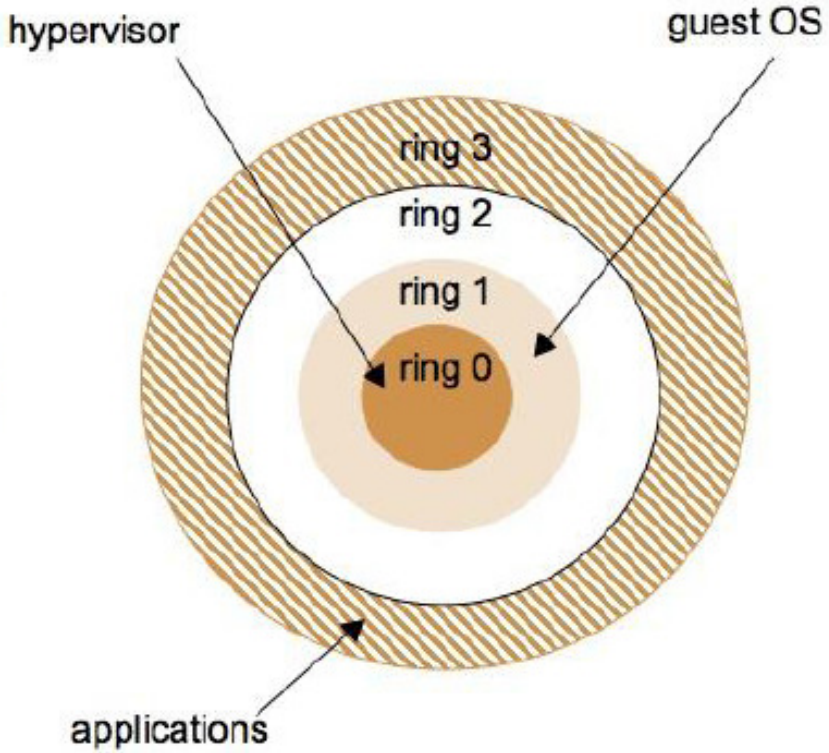
Arsitektur dari Xen



Penggunaan *Rings of Privilege*

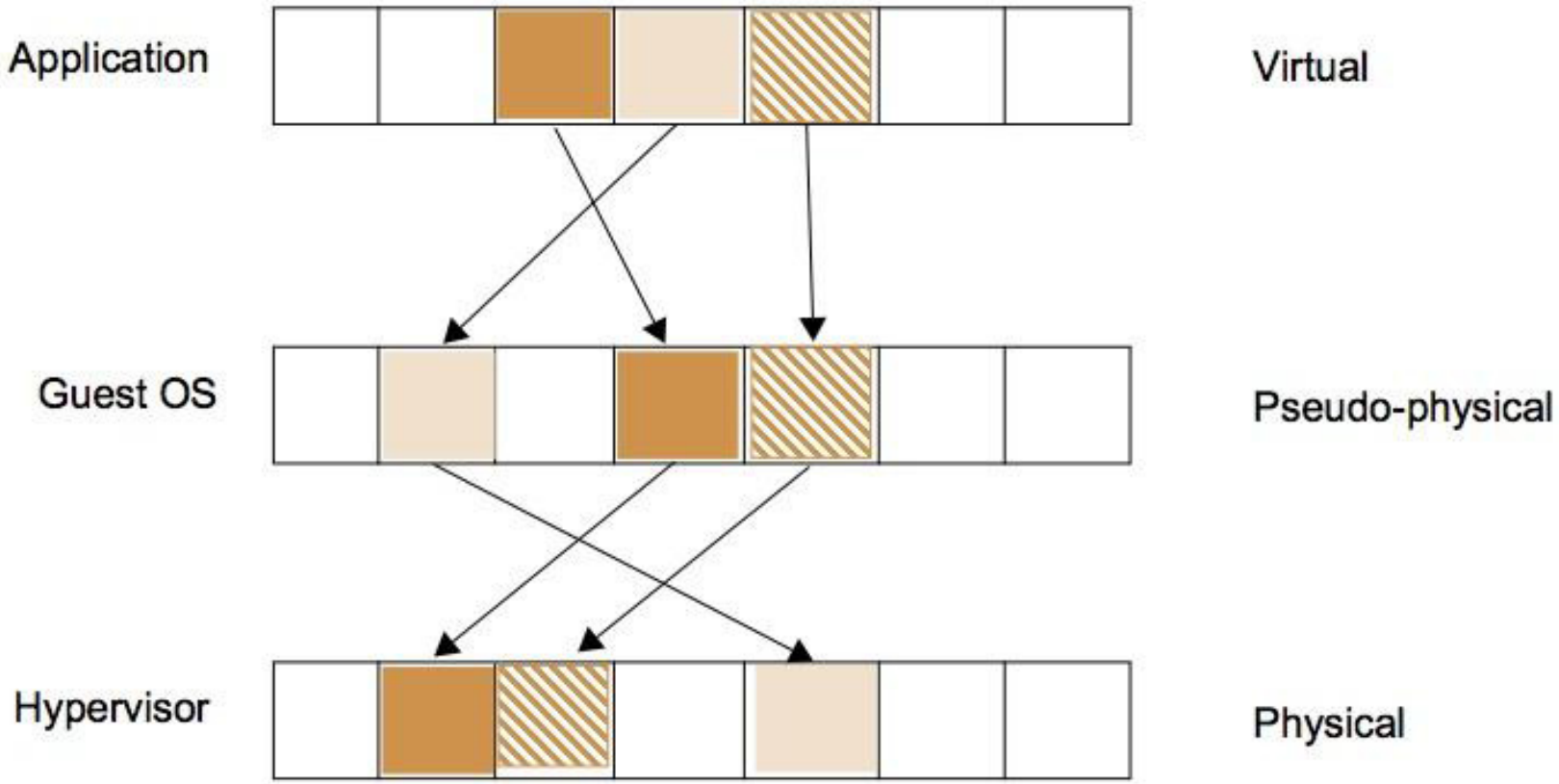


Sistem Operasi berbasis Kernel

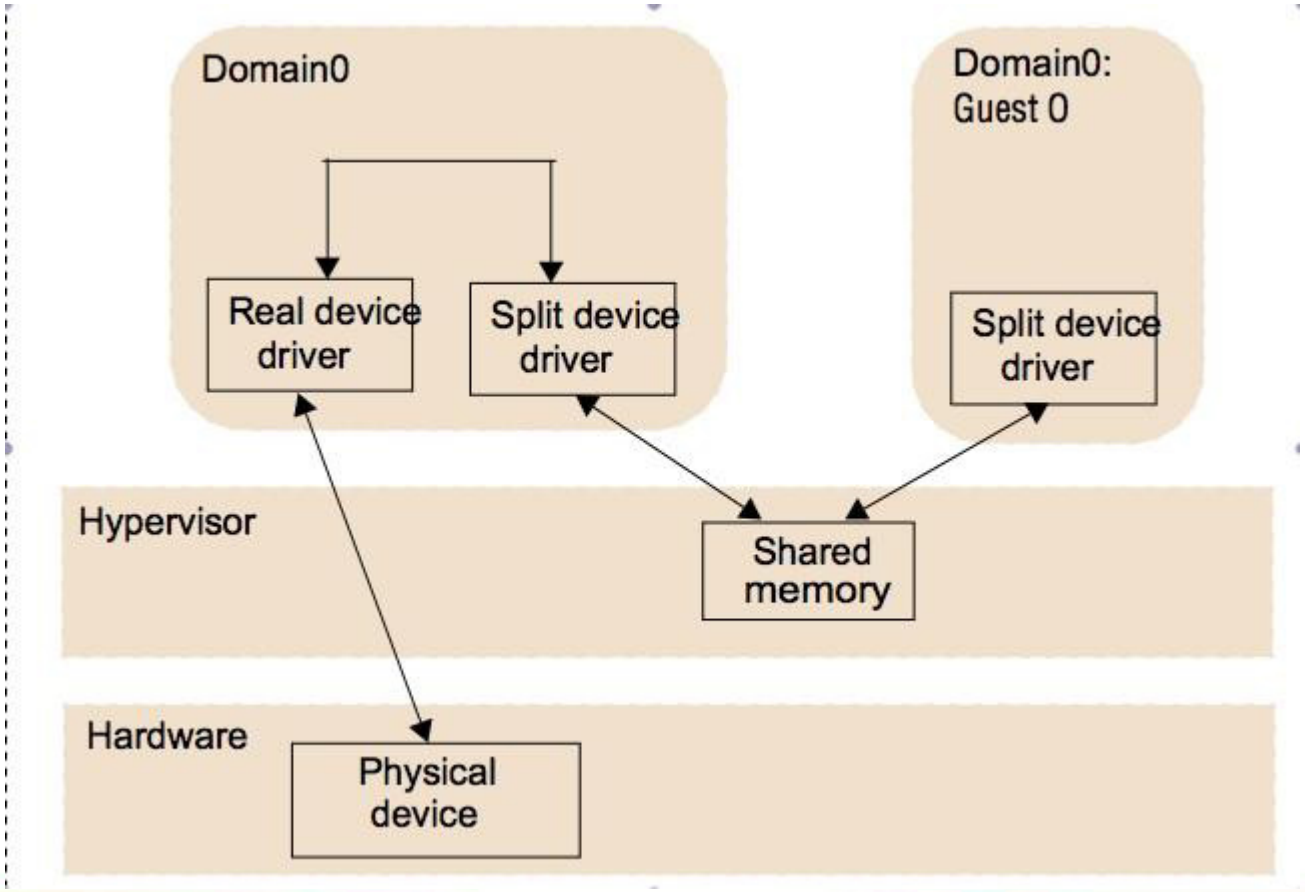


Paravirtualisasi di Xen

Virtualisasi Manajemen Memori



Split Device Driver



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