

Sistem I/O

Sistem I/O

- ❖ Perangkat Keras I/O
- ❖ Aplikasi Antarmuka I/O
- ❖ Kernel I/O *Subsystem*
- ❖ Mengubah I/O *Request* Menjadi Operasi Perangkat Keras
- ❖ *Streams*
- ❖ *Performance*

Perangkat Keras I/O

- ❖ Banyaknya jenis perangkat keras I/O
- ❖ Konsep Umum :
 - *Port*
 - *Bus (Daisy chain atau shared direct access)*
 - *Controller (host adapter)*
- ❖ Perangkat kontrol instruksi I/O
- ❖ Perangkat-perangkat tersebut memiliki alamat, digunakan untuk:
 - Instruksi I/O langsung
 - *Memory-mapped I/O*

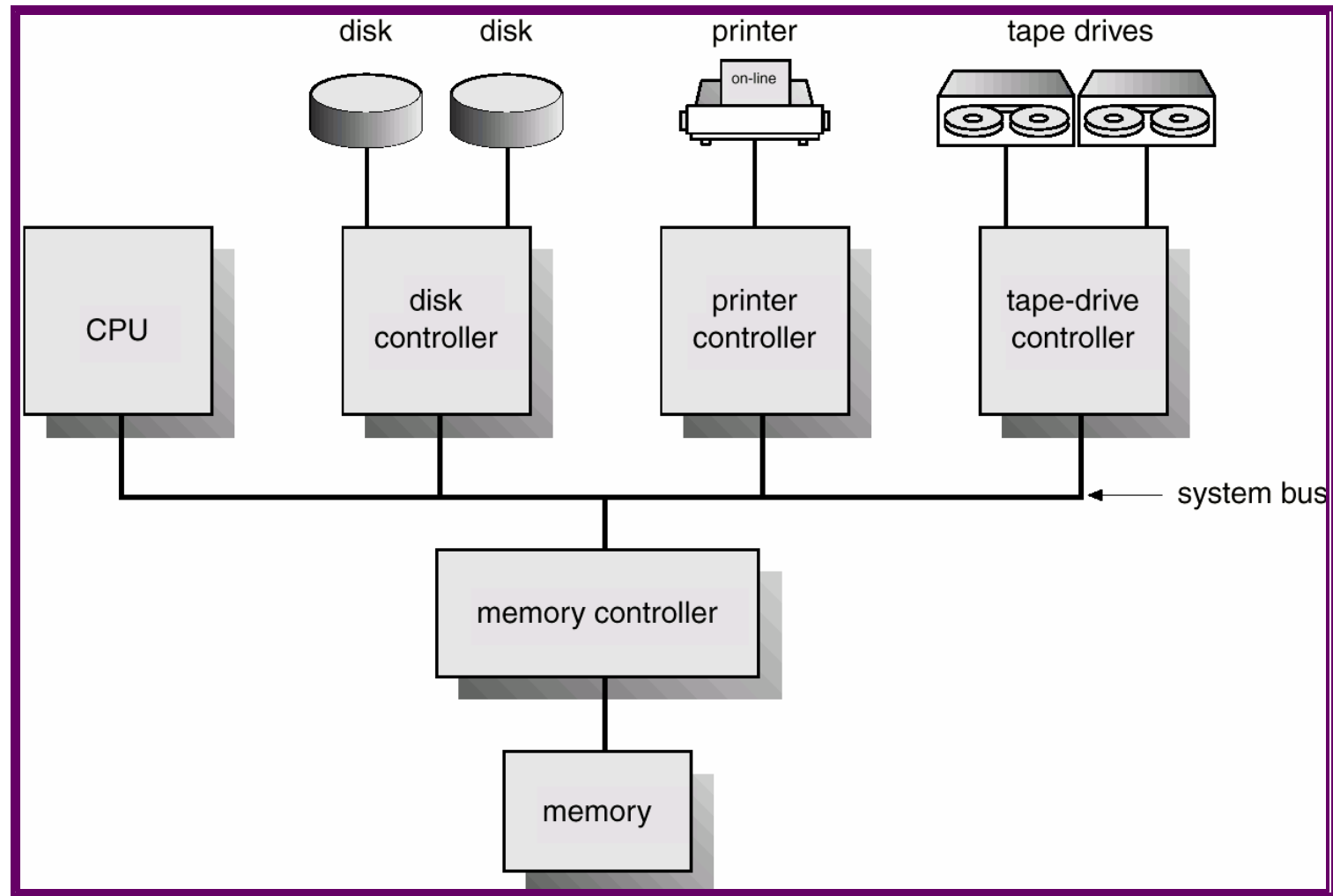
Jenis Perangkat Keras

- ❖ Perangkat penyimpan data
- ❖ Perangkat penghubung
- ❖ Perangkat antarmuka dengan *user*

Konsep Umum

- ❖ Suatu perangkat berhubungan dengan sistem komputer dengan cara mengirim sinyal melalui suatu kabel atau bahkan melalui udara
- ❖ Perangkat tersebut berkomunikasi dengan mesin melalui *port*
- ❖ Struktur komputer yang umum dipakai adalah *Daisy Chain*

Arsitektur Sistem Komputer



I/O Port Register

- ❖ *Register Status*
- ❖ *Register Control*
- ❖ *Register Data-in*
- ❖ *Register Data-out*

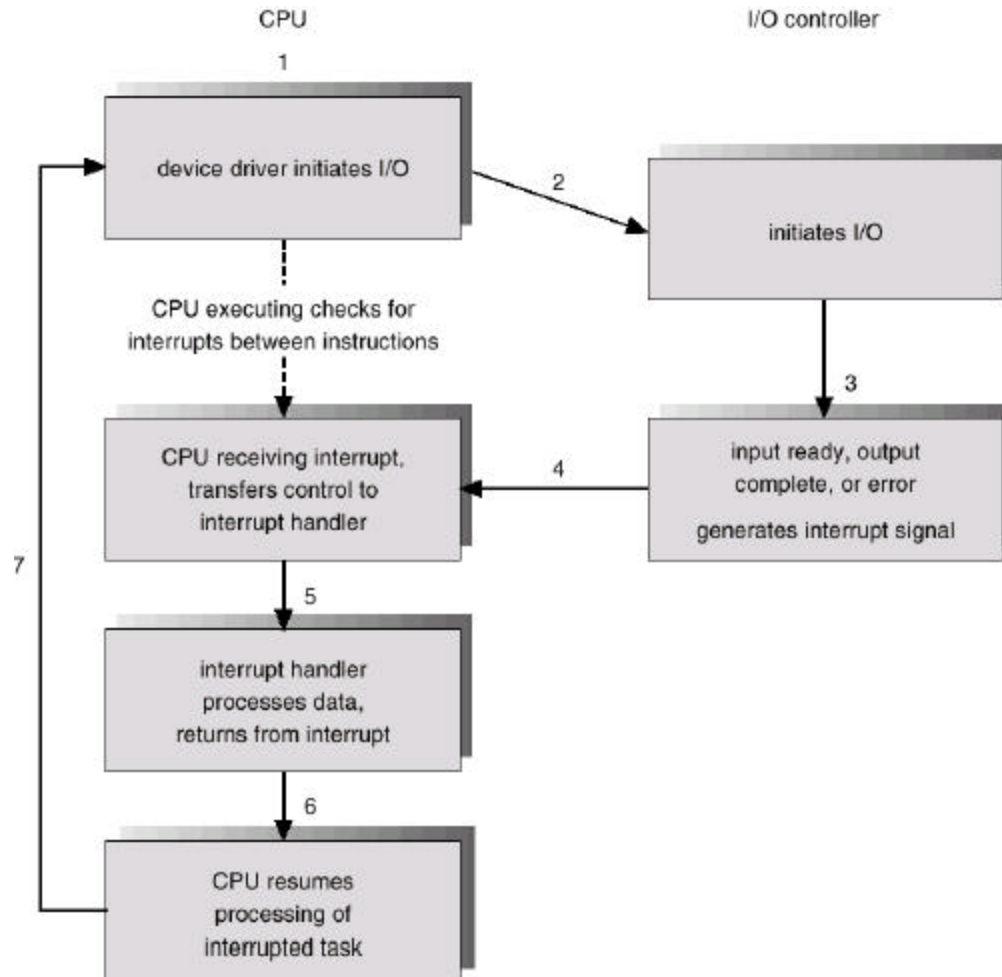
Polling

- ❖ Host terus membaca *busy-bit* secara berulang-ulang sampai bit tersebut *clear*
- ❖ *Host set write-bit* di *command-register* dan menulis satu byte di *data-out register*
- ❖ *Host set bit command-ready*
- ❖ Ketika controller mengetahui kalau *bit command-ready* di-set, dia men-set busy bit
- ❖ Controller membaca *command-register* dan melihat perintah tulis. Dia membaca *data-out register* untuk mendapatkan bytenya, dan melakukan operasi I/O
- ❖ Controller menghapus *bit command-ready*, membersihkan *bit error* di status register yang menandakan operasi I/O berhasil, dan menghapus *busy-bit* yang menandakan kalau operasi sudah selesai.

Interrupt

- ❖ Jalur *interrupt* dihasilkan oleh perangkat I/O
- ❖ *Interrupt Handler* menerima interrupt tersebut
- ❖ Mekanisme interrupt juga digunakan untuk penanganan *exception*

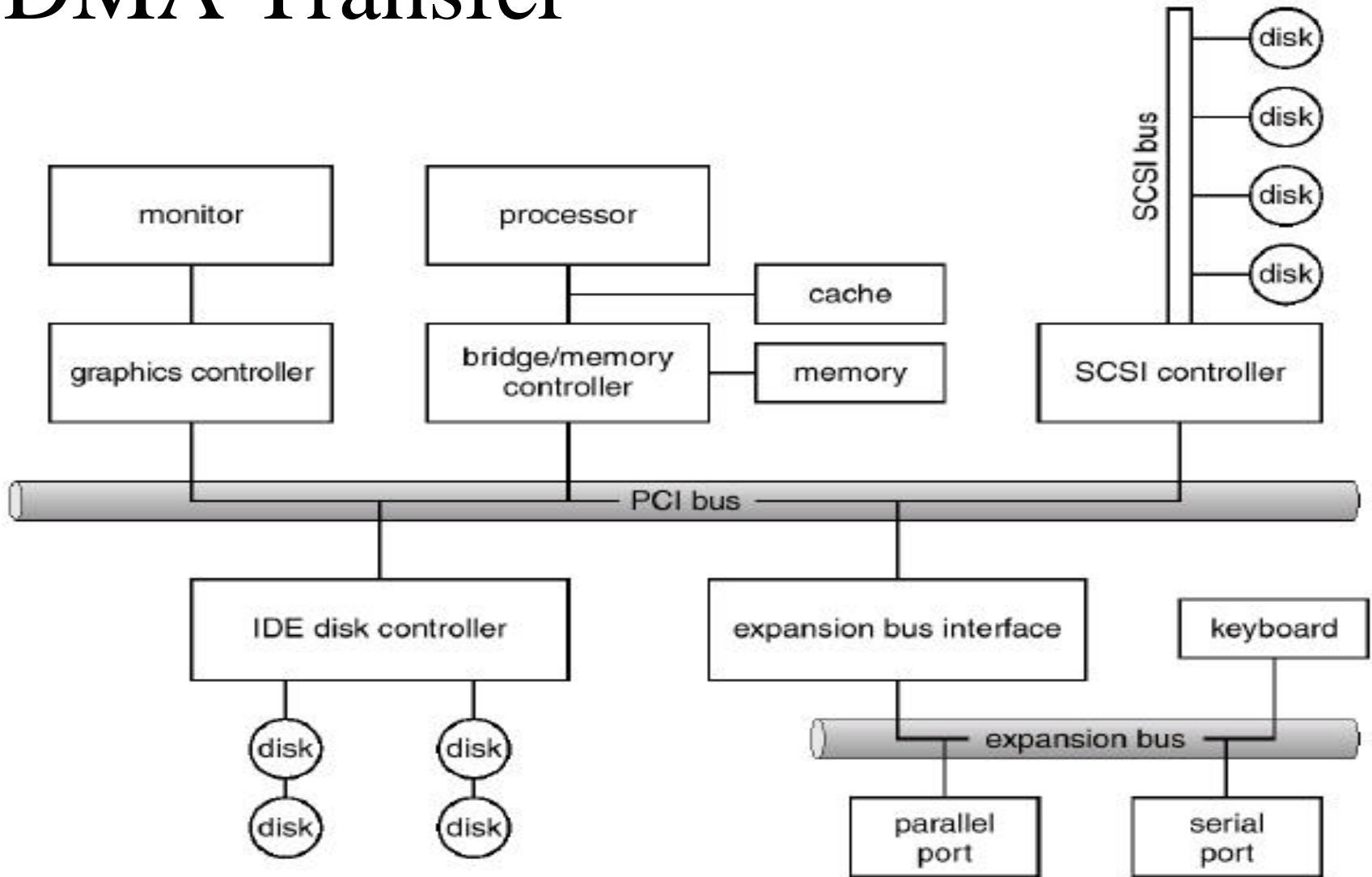
Interrupt-Driven I/O Cycle



Direct Memory Access (DMA)

- ❖ Generasi komputer yang sangat tua
 - *Controller* membaca dari perangkat
 - Sistem Operasi meminta *controller* membaca data
- ❖ Generasi komputer yang tua
 - *Controller* membaca dari perangkat
 - *Controller* meng-*interrupt* OS
 - Sistem Operasi menyalin data ke memori
- ❖ Generasi DMA
 - *Controller* membaca dari perangkat
 - *Controller* menyalin data ke memori
 - *Controller* meng-*interrupt* OS

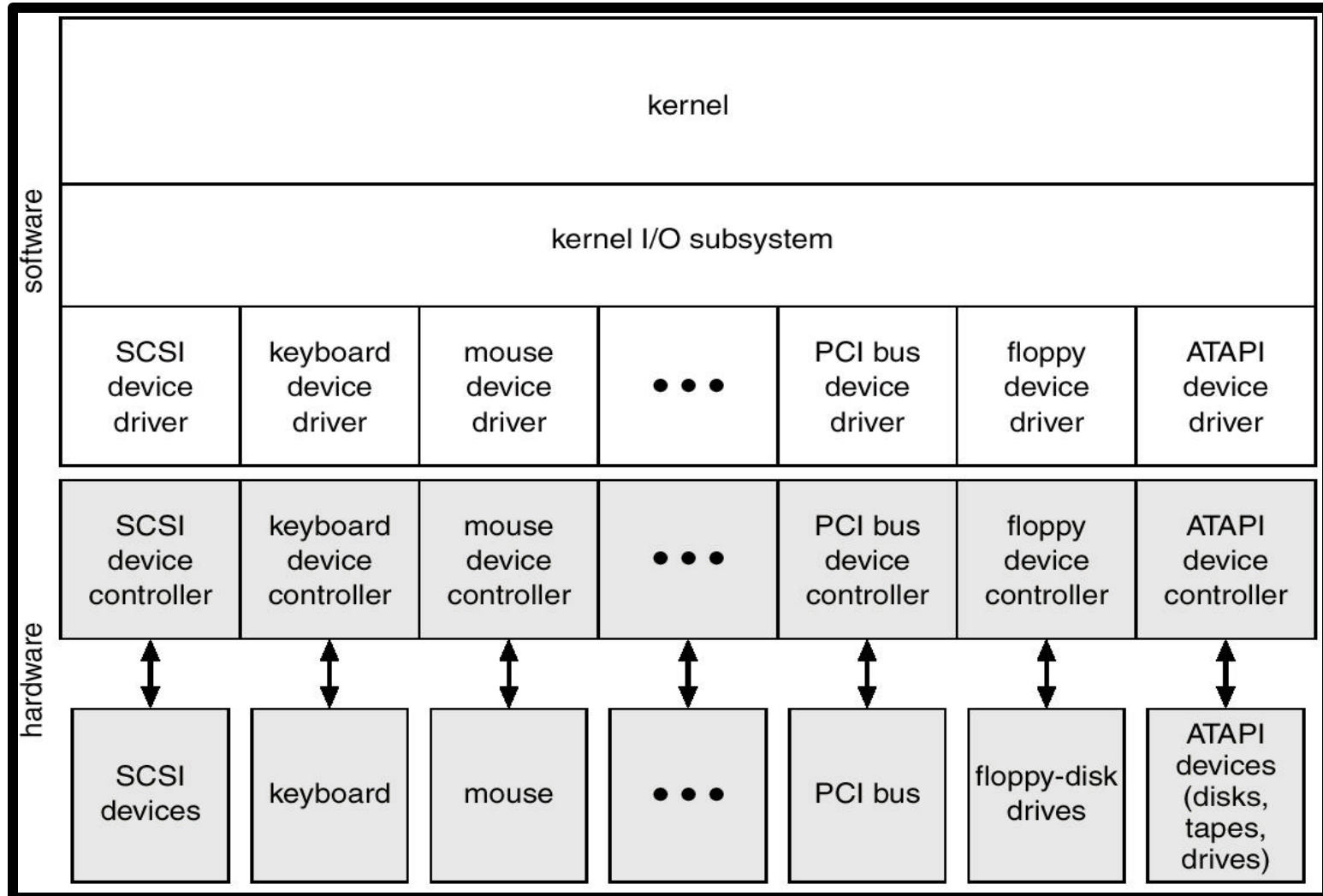
DMA Transfer



Aplikasi Antar-Muka I/O

- ❖ Sifat-sifat *perangkat* komputer diabstraksi oleh *I/O system call* berbentuk kelas-kelas umum.
- ❖ Lapisan *driver perangkat* menyembunyikan perbedaan-perbedaan *I/O controller* dari kernel.
- ❖ Ragam *device* dari beberapa sisi:
 - *Character-stream* atau *block*
 - *Sequential* atau *random-access*
 - *Synchronous* atau *asynchronous*
 - *Sharable* atau *dedicated*
 - *Speed* atau *operation*
 - *Read-write, read only, write only*

Struktur Kernel I/O



Karakteristik Perangkat I/O

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read&write	CD-ROM graphics controller disk

Perangkat *Block* dan *Character*

❖ Perangkat *block*:

- Meliputi berbagai *disk drive*
- Perintah baca, tulis, pencarian data
- Dimungkinkan untuk mengakses berkas secara *memory-mapped*

❖ Perangkat *character*:

- Contoh: *keyboard, mouse*
- Perintah menulis, mengambil
- Dapat dibuat *library* pengakses data per-baris

Perangkat Jaringan

- ❖ *Interface* berbeda dari baca, tulis *disk*, disebut *interface socket*.
- ❖ *Socket*: penghubung komputer dengan jaringan.
- ❖ *Local socket* dihubungkan dengan *remote socket*.
- ❖ Komunikasi antar komputer dilakukan melalui *socket*.

Clock dan Timer

- ❖ Fungsi *clock* dan *timer* pada *hardware*:
 - Waktu saat ini
 - Lama sebuah proses
 - *Trigger* proses pada suatu waktu
- ❖ *Programmable interval timer* : hardware pengukur waktu dan *trigger*.
- ❖ Sistem operasi mampu menangani *time request* lebih banyak dari jumlah *hardware timer*

Blocking dan Non-blocking I/O

- ❖ *Blocking* : proses dihentikan sementara
 - Lebih mudah dimengerti
 - Tidak cukup untuk beberapa hal
- ❖ *Non-blocking* : diimplementasikan lewat *multi-threading*
- ❖ *Asynchronous* : proses berjalan selama I/O dieksekusi

Kernel I/O Subsystem

❖ *Scheduling :*

- Permohonan I/O dilakukan berdasarkan antrian perangkat
- Beberapa sistem operasi berusaha untuk seadil mungkin

❖ *Buffering : menyimpan data di memori selama proses transfer antar perangkat*

- Solusi perbedaan *kecepatan* dari perangkat yang ada
- Solusi perbedaan ukuran transfer perangkat

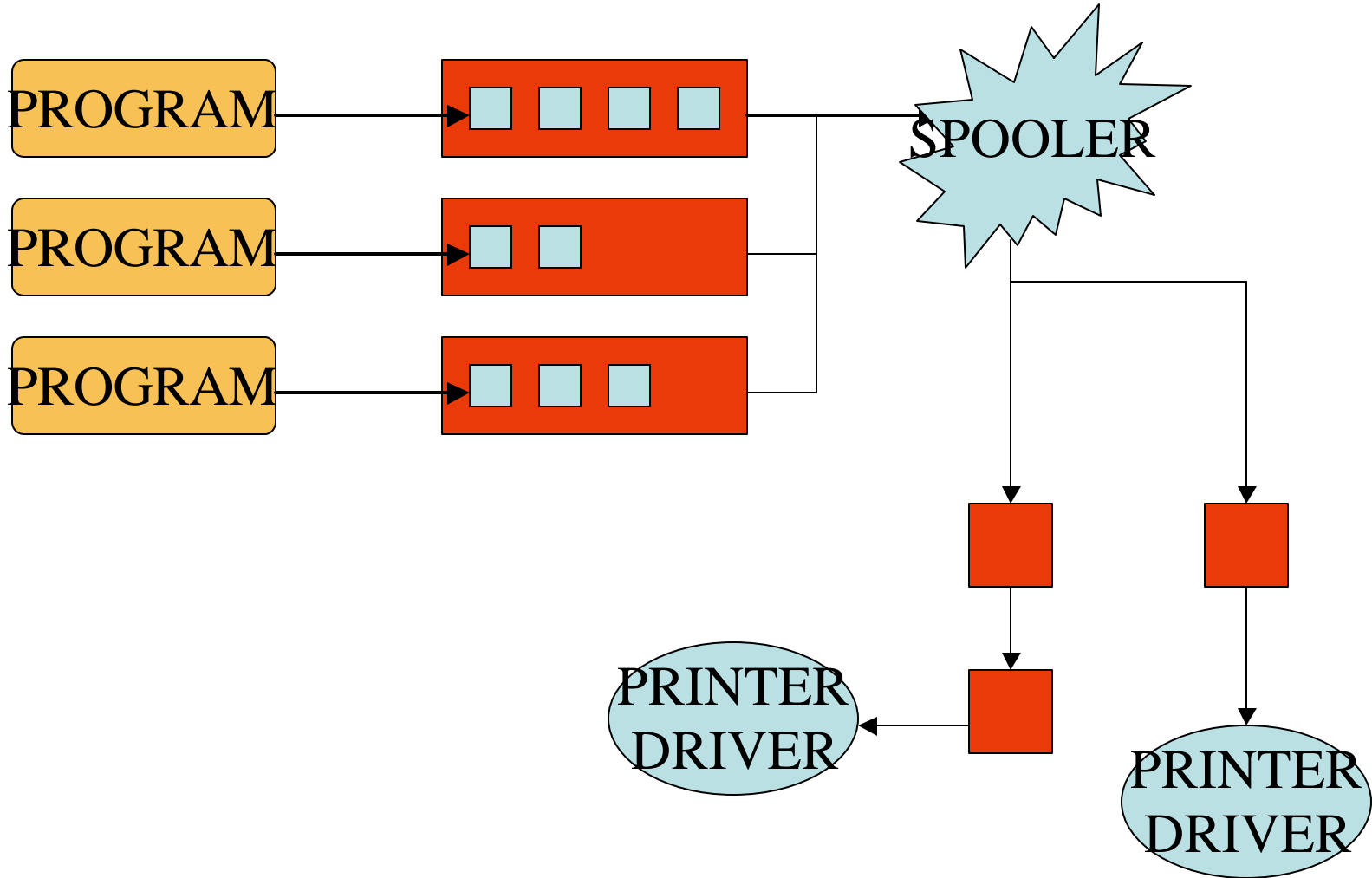
Caching

- ❖ *Cache* : area memori yang cepat, yang berisikan kopian-kopian data.
- ❖ Beda *BUFFER* dan *CACHE* :
 - *Buffer* dapat menyimpan satu-satunya copy dari sebuah item data yang ada.
 - *Cache* hanya menyimpan sebuah salinan dari data di tempat lain pada *storage* sehingga lebih cepat diakses.
- ❖ Peningkatan performa I/O, terutama untuk:
 - berkas yang digunakan secara bersama oleh beberapa aplikasi,
 - berkas yang sedang di baca/tulis secara berulang-ulang.

Spooling (1)

- ❖ *Spool* : *buffer* yang menyimpan output *device*
 - Tidak dapat menerima *interleaved data stream*.
- ❖ 1 device memenuhi 1 permintaan, tapi aplikasi bisa minta bersamaan.
- ❖ Sistem operasi meng-*intercept* semua output ke *device*.
Masing-masing output aplikasi di-*spooled* ke berkas *disk* yang berbeda.
- ❖ Setiap Sistem Operasi menyediakan *control interface* yang :
 - Membuat users dan administrator sistem menampilkan antrian,
 - Menyingkirkan pekerjaan yang tidak diinginkan.
 - dll.

Spooling (2)



Device Reservation

- ❖ Menyediakan akses eksklusif bagi sebuah *device*.
- ❖ *System Call* untuk alokasi dan dealokasi *device*.
- ❖ Punya parameter untuk membuka *system call* yang mendeklarasikan tipe akses yang diijinkan untuk *thread-thread* konkruen yang lain.
- ❖ Perlu waspada terhadap *Dead Lock*.

Error Handling

- ❖ Sistem Operasi dengan pelindung memori dapat bertahan dari berbagai jenis *error* dari perangkat keras dan aplikasi.
- ❖ Sistem Operasi sulit memperbaiki kesalahan permanen bila terjadi pada komponen penting,.
- ❖ Umumnya akan me-*return* sebuah *error number* atau kode ketika permintaan I/O gagal.
- ❖ Log *system error* menyimpan laporan masalah yang ada.

Struktur Data Kernel

- ❖ Kernel menyimpan informasi penggunaan komponen I/O, termasuk tabel *open-file*, koneksi *networking*, informasi karakter *device*.
- ❖ Struktur data yang rumit dapat digunakan untuk memeriksa *buffer*, alokasi memori, dan menentukan batasan sektor/blok.
- ❖ Beberapa sistem operasi menggunakan teknik *object oriented* untuk mengkapsulasikan perbedaan-perbedaan *semantik* yang ada.

Transformasi I/O Menjadi Operasi H/W

Proses:

- ❖ *Blocking read system call* diberikan pada pendeskripsi data dari data yang sudah terbuka sebelumnya.
- ❖ Kode di kernel memeriksa parameter. Dalam proses *input*, jika data sudah ada di *buffer*, data dikembalikan ke proses dan permintaan I/O selesai

Contoh: membaca data dari *disk* untuk di proses.

- ❖ Menentukan *device* yang mengandung data,
- ❖ Menerjemahkan nama ke perwakilan *device*
- ❖ Secara fisik memindahkan data dari disk ke *buffer*
- ❖ Mempersiapkan data untuk proses permintaan I/O
- ❖ Mengembalikan kontrol ke proses

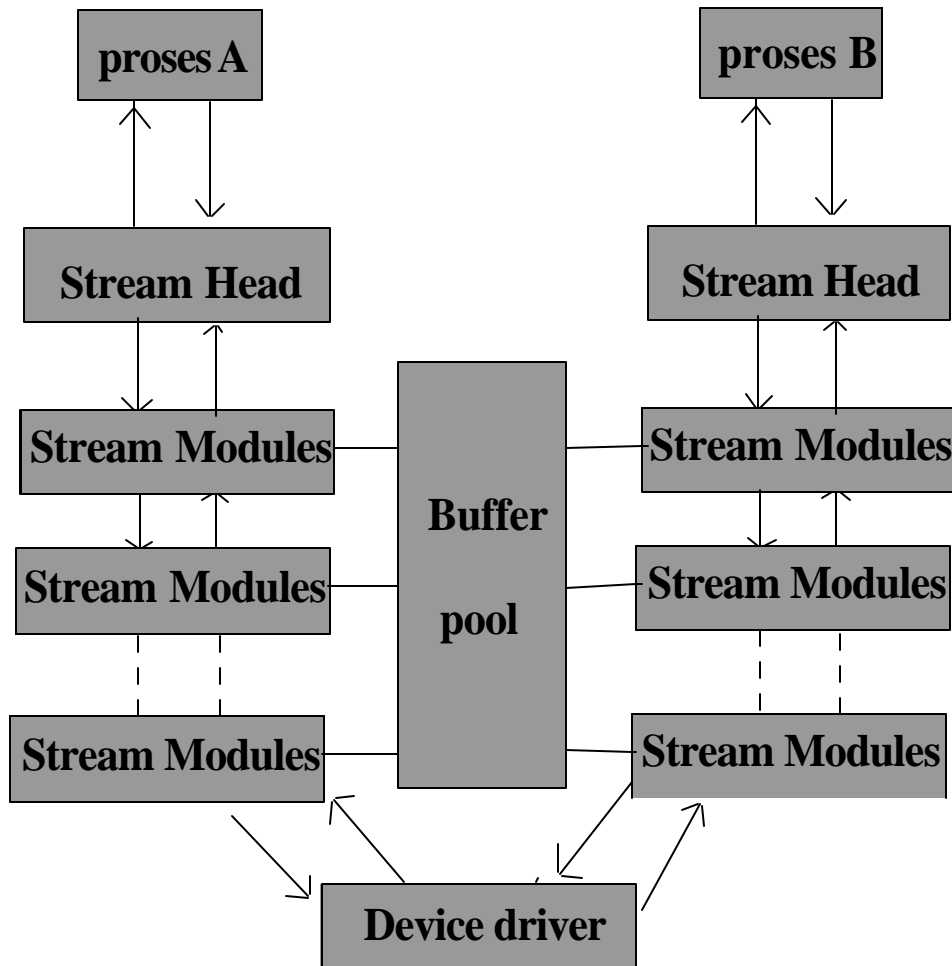
I/O Stream (1)

- ❖ *I/O stream* adalah suatu mekanisme pengiriman data secara bertahap dan terus menerus melalui suatu aliran data (dua arah)
- ❖ Biasa digunakan dalam *network protocol*
- ❖ *Asynchronous*
- ❖ Menggunakan *message passing* dalam *men-transfer* data

I/O Stream (2)

- ❖ Untuk memasukkan ke dalam stream digunakan *ioctl system call*
- ❖ Untuk menuliskan data ke *device* digunakan *write / putmsg system call*
- ❖ Untuk membaca data dari *device* digunakan *read / getmsg system call*

I/O Stream (3)



- ❖ *User process* berhubungan langsung dengan *stream head*
- ❖ Ada beberapa modul dengan *write* dan *read queue*
- ❖ *Device* berhubungan langsung dengan *driver end*

Kinerja I/O

- ❖ Pembuat CPU melaksanakan kode *device-driver*
- ❖ Memberitahukan ke-tidak efisien-an pada mekanisme penanganan *interrupt* dalam kernel
- ❖ Me-load *memory bus* sewaktu menyalin data yang dilakukan di *controller* dan *physical memory*

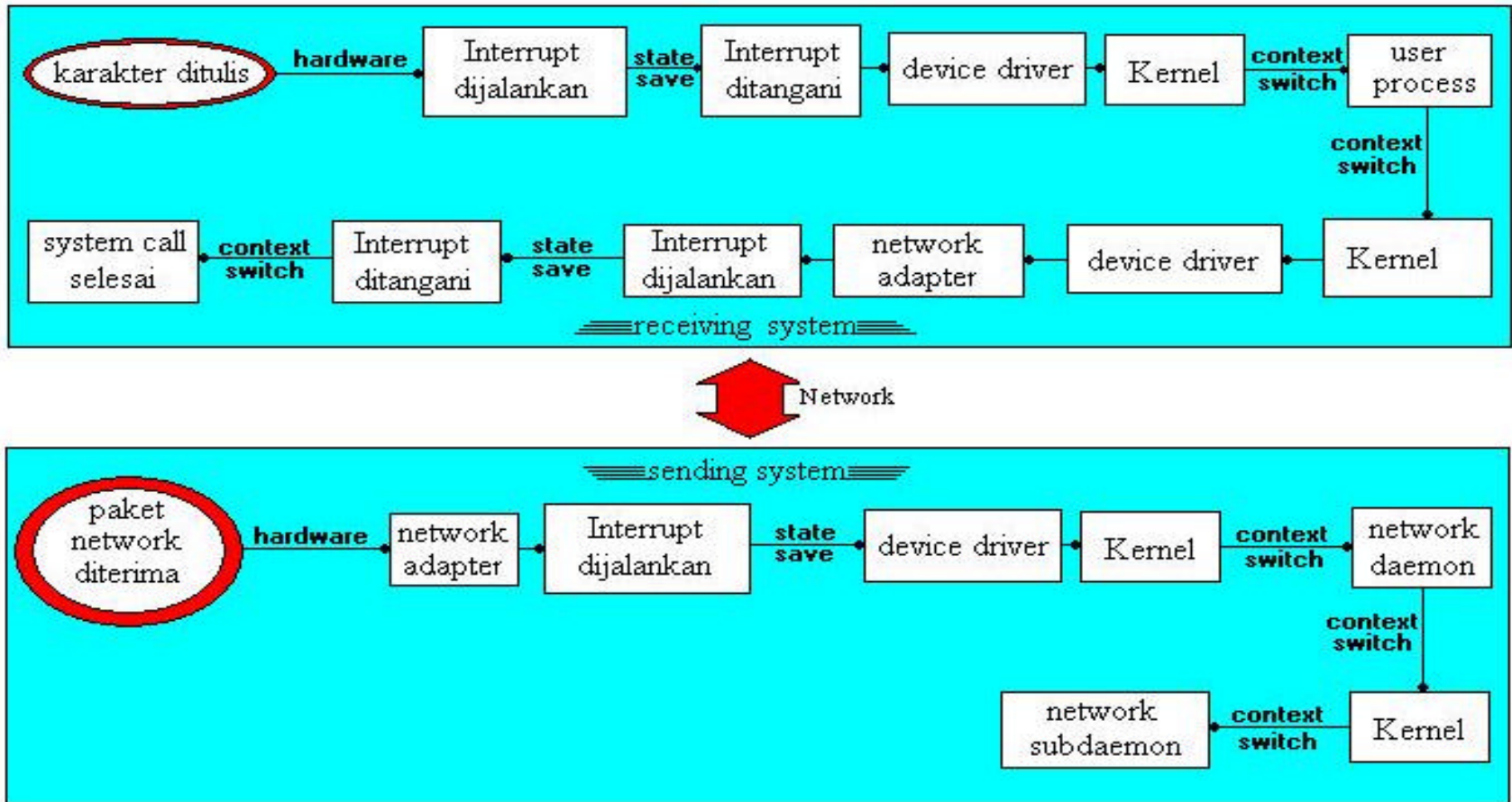
Meningkatkan Kinerja I/O (1)

- ❖ Memperkecil jumlah *context switch*
- ❖ Memperkecil jumlah penyalinan data yang dilakukan sewaktu pengoperan data antara *device* dan aplikasi
- ❖ Memperkecil jumlah *interrupt* dengan menggunakan transfer secara besar-besaran, *smart controllers* dan polling (jika *busy-waiting* bisa diminimalisir)

Meningkatkan Kinerja I/O (2)

- ❖ Menambah konkurensi dengan menggunakan *DMA controllers* atau *channels* yang telah diketahui untuk meng-*offload* penyalin sederhana dari CPU
- ❖ Memindahkan proses-proses primitif ke perangkat keras, untuk membuat operasinya dalam *device controllers* konkuren dengan CPU dan operasi Bus
- ❖ Menyeimbangkan CPU, *memory subsystem*, *bus*, dan *I/O performance*, karena kelebihan di salah satu area akan membuat keterlambatan pada yang lain

Komunikasi Antarkomputer



Mengimplementasikan I/O

- ❖ I/O seharusnya diimplementasikan dalam pada waktu *application level*
- ❖ Ketika algoritma pada application-level sudah menunjukkan kegunaannya, implementasikan kembali dalam kernel
- ❖ Kinerja tertinggi bisa didapatkan dari implementasi spesial ke perangkat keras, baik dalam *device* atau dalam *controller*

Struktur *Disk*

- ❖ *Magnetic tape*

- Kapasitas besar

- Lambat

- ❖ *Disk Drive*

- Lebih cepat dibanding magnetic tape

- ❖ *Struktur Disk Drive :*

- *Constant Linear Velocity*

- *Constant Angular Velocity*

Penjadualan *Disk* (1)

- ❖ Efisien, cara??
- ❖ Komponen utama waktu *access time* :
 - ❖ *Seek time*
 - *Rotational latency*
 - *Disk Bandwidth*

Penjadualan *Disk* (2)

- ❖ Informasi yang dibawa proses yang melakukan *system call*
- ❖ Apakah operasi Input/Output
 - Alamat disk untuk proses tersebut
 - Alamat memori untuk proses tersebut
- ❖ Jumlah bytes yang akan ditransfer

FCFS Scheduling (1)

❖ Menggunakan algoritma *First-come first-served*

❖ Contoh :

Permintaan pada disk: 10, 45, 37, 56, 60, 25, 78, 48, 96,
70, 5, 20

Awal : silinder 50

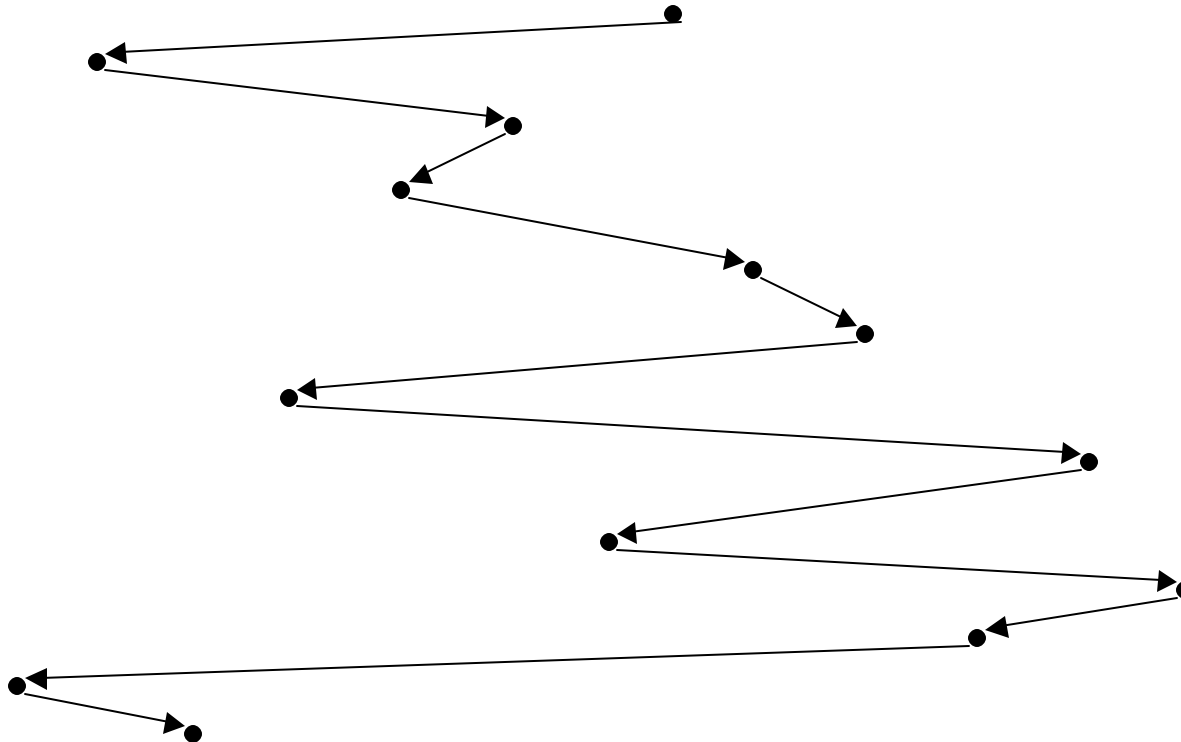
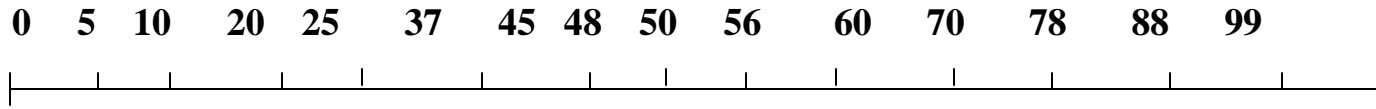
Cara Kerja : 50 ke 10, lalu ke 45, 37, 56, 60, 25, 78,
48, 88, 70, 5, 20

❖ Total pergerakan : 362

+ Sangat adil bagi tiap proses

– Lambat

FCFS Scheduling (2)



SSTF Scheduling (1)

❖ Menggunakan algoritma *shortest-seek-time-first*

❖ Contoh :

Permintaan pada *disk*: 10, 45, 37, 56, 60, 25, 78, 48,
96, 70, 5, 20

Awal : silinder 50

Cara Kerja: 50 ke 48, lalu ke 45, 37, 25, 20, 10, 5,
56, 60, 70, 78, 88

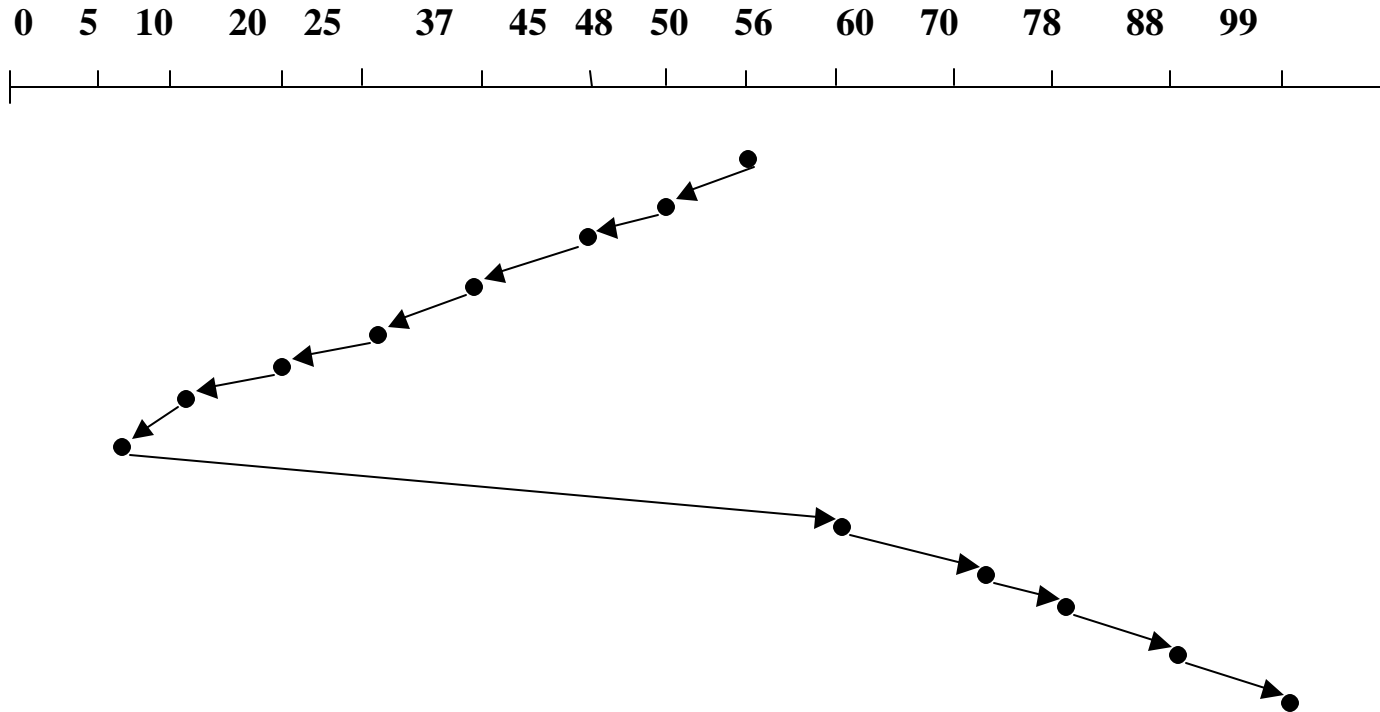
(memilih silinder yang paling dekat dengan posisi terakhir dari head)

❖ Total pergerakan : 128

+ Lebih cepat dibanding FCFS scheduling

– Bisa menyebabkan “starvation” untuk permintaan tertentu

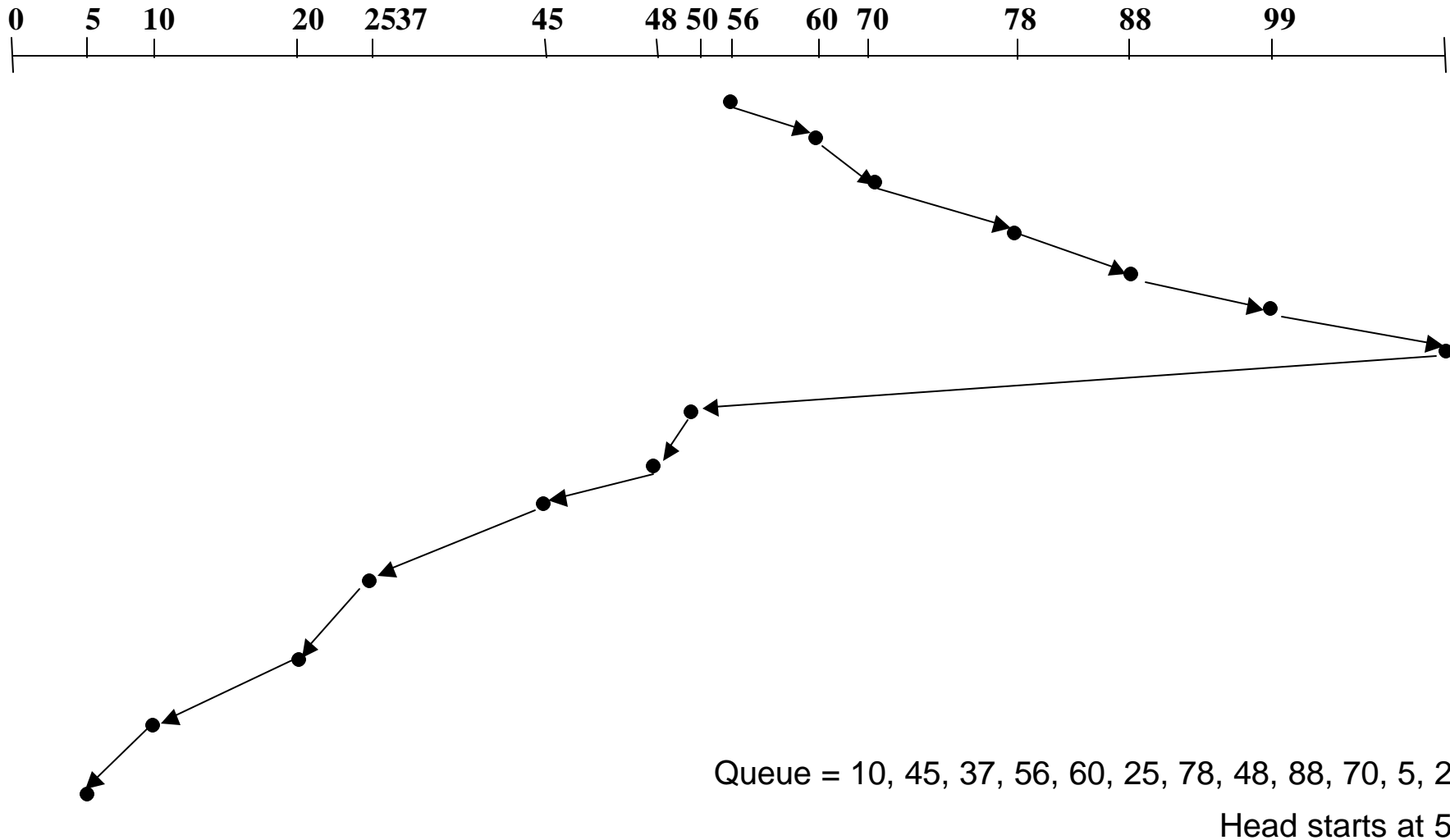
SSTF Scheduling (2)



SCAN

- ❖ Disebut Algoritma *Lift (Elevator Algorithm)* karena cara kerjanya seperti *lift*.
 - ❖ *Disk arm* bergerak sampai ke silinder paling ujung dari *disk*, kemudian berbalik arah gerak, menuju ke silinder paling ujung lainnya.
 - ❖ Aturan pelayanan : Permintaan yang berada di depan arah gerak *disk head* bisa dilayani terlebih dahulu. Permintaan yang berada di belakang arah gerak *disk head* harus menunggu sampai *disk head* mencapai salah satu ujung disk, kemudian berbalik arah geraknya.
- + Total pergerakan *disk arm* memiliki batas atas, yaitu 2 kali dari jumlah total silinder pada disk
- 1. Bisa terjadi *starvation*.
 - 2. *Disk arm* harus bergerak sampai ujung disk, padahal mungkin sudah tidak ada lagi permintaan di depan arah gerak *disk arm*.

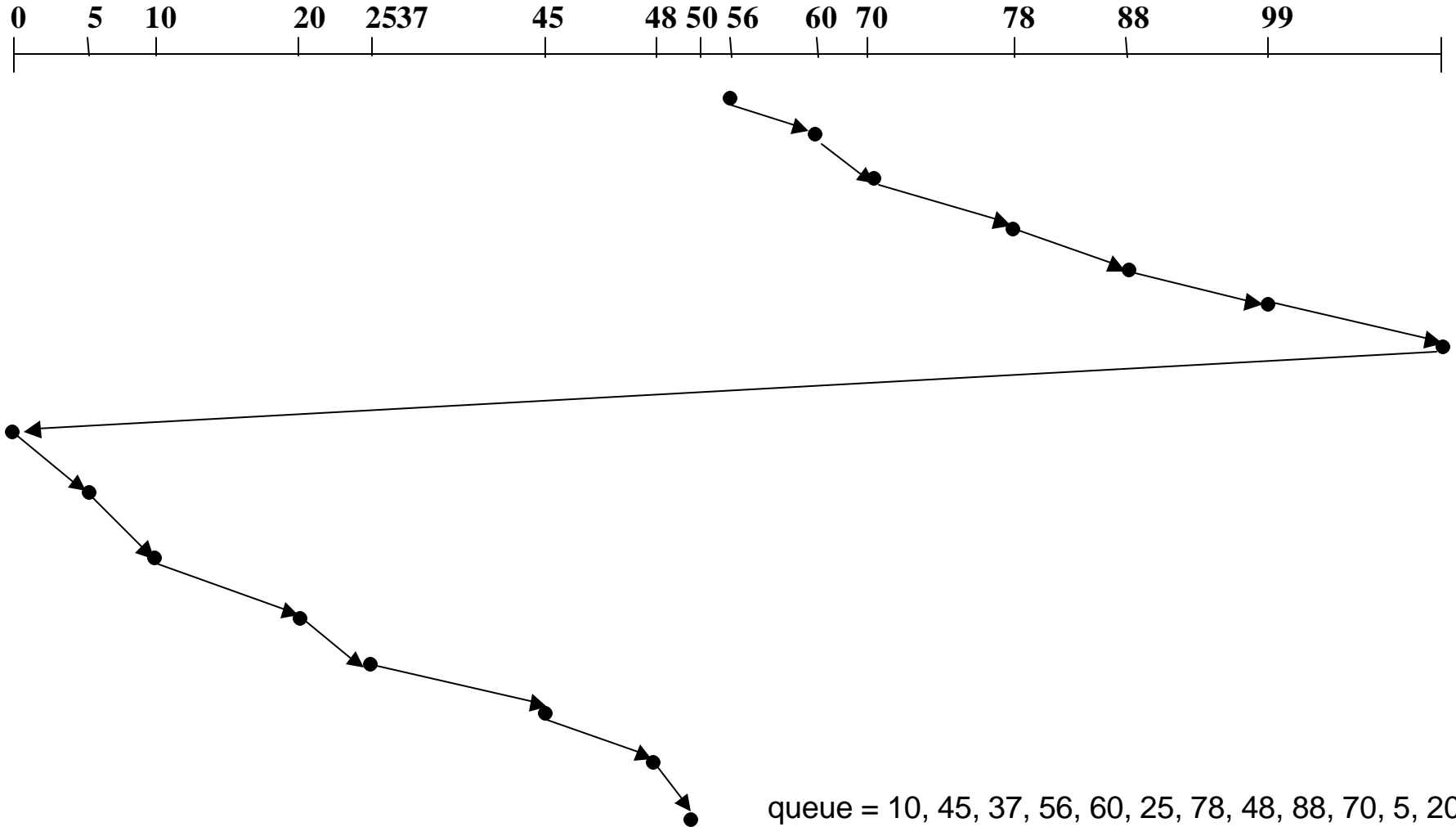
Contoh Cara Kerja SCAN



C-SCAN

- ❖ *Circular Scan* mempunyai aturan pelayanan yang sama dengan SCAN, tetapi memiliki perbedaan pada cara pergerakan *disk arm*.
 - ❖ *Disk arm* bergerak dari salah satu silinder paling ujung dari disk (misal: ujung 1) ke silinder paling ujung lainnya (misal : ujung 2), tetapi setelah sampai ke ujung 2, *disk arm* bergerak dengan sangat cepat untuk kembali ke ujung 1. Pada saat bergerak kembali ke ujung 1, permintaan tidak dilayani. Jadi, seolah-olah *disk head* hanya bergerak 1 arah dalam melayani permintaan.
- + Sangat mengurangi terjadinya *starvation*.
- Sama seperti SCAN, *disk arm* tetap harus bergerak sampai ujung disk, padahal mungkin sudah tidak ada lagi permintaan di depan arah gerak *disk arm*.

Contoh Cara Kerja C-SCAN

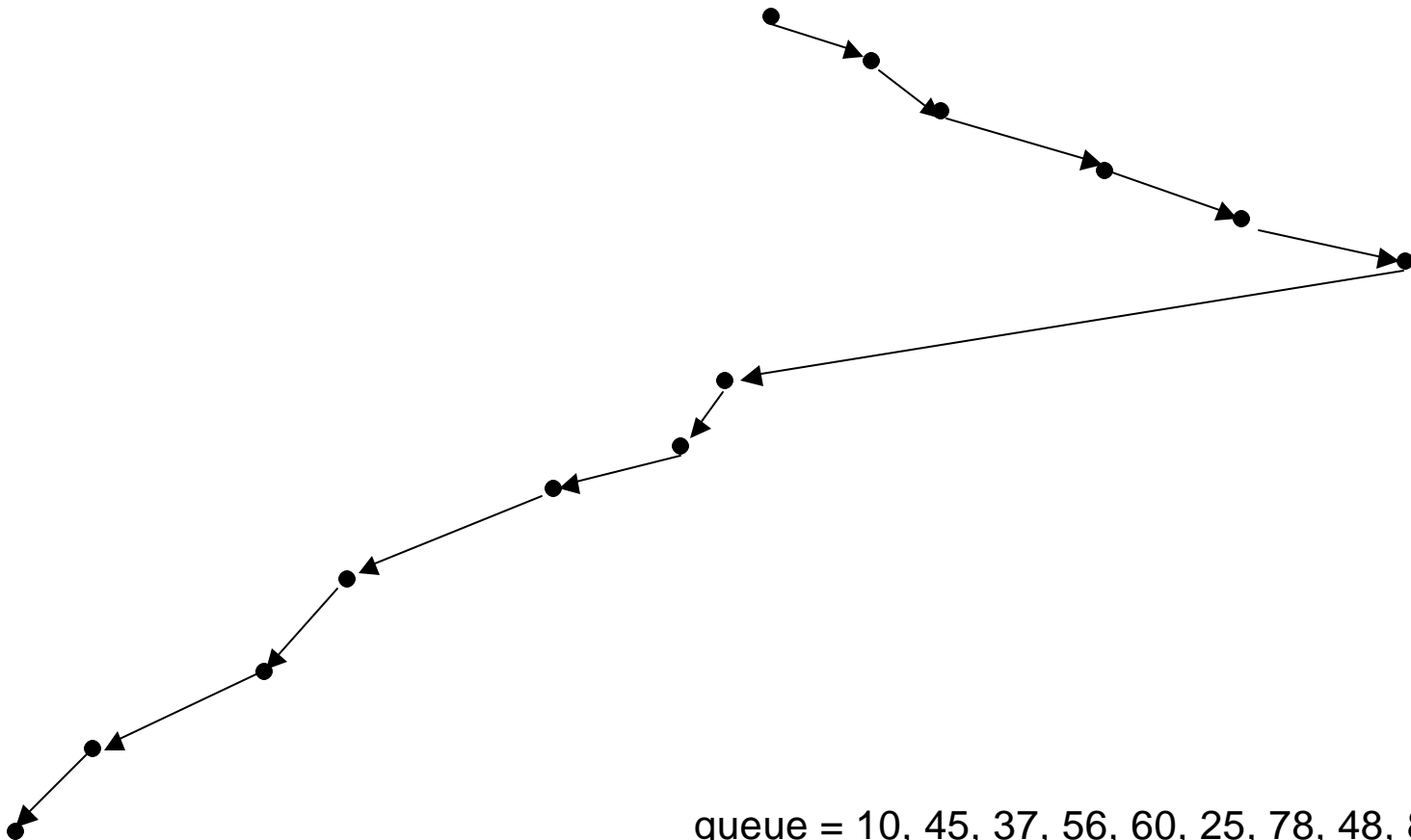
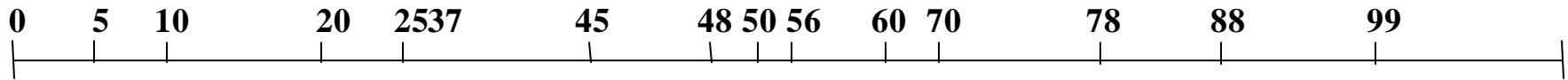


Head starts at 50

LOOK

- ❖ *Look* bisa dikatakan sebagai algoritma lift yang lebih pintar.
- ❖ Aturan pelayanan sama seperti Scan.
- ❖ Terdapat sedikit perbaikan pada cara pergerakan *disk arm*, yaitu: *Disk arm* tidak perlu benar-benar sampai ke silinder paling ujung dari disk, tetapi hanya menuju silinder paling ujung dari disk. Ketika di depan arah pergerakan *disk arm* sudah tidak ada lagi permintaan, *disk arm* langsung berbalik arah geraknya.
- + Lebih efisien, *disk arm* tidak harus bergerak sampai silinder paling ujung dulu untuk bisa berbalik arah.
- Untuk situasi yang menyebabkan terjadinya *starvation* pada Scan, juga menyebabkan *starvation* pada Look.

Contoh Cara Kerja LOOK



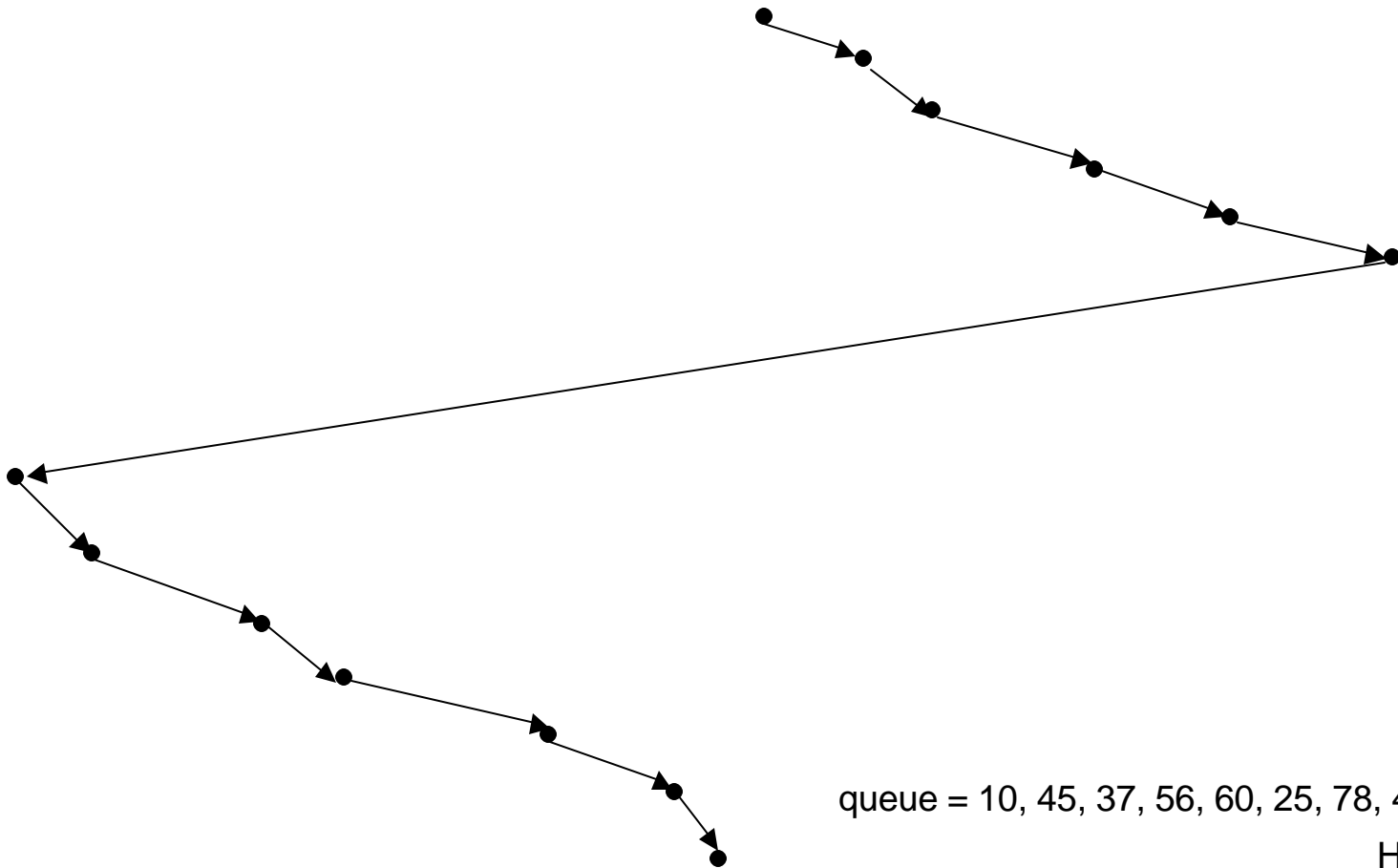
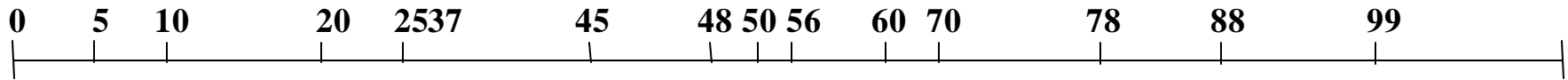
queue = 10, 45, 37, 56, 60, 25, 78, 48, 88, 70, 5, 20

Head starts at 50

C-LOOK

- ❖ Memiliki aturan pelayanan yang sama seperti Scan.
 - ❖ Pergerakan disk arm merupakan kombinasi Look dan C-Scan:
 - ❖ Disk arm bergerak menuju silinder paling ujung dari disk, tetapi ketika di depan arah pergerakannya sudah tidak ada lagi permintaan, disk arm langsung bergerak berbalik arah secara cepat ke permintaan yang paling dekat dengan ujung yang lainnya. Sama seperti C-Look, disk arm seolah-olah hanya bergerak 1 arah dalam melayani permintaan, karena ketika bergerak berbalik arah, permintaan tidak dilayani.
- + Mengurangi terjadinya starvation.
- ❖ Circular Look memperbaiki Look, seperti Circular Scan memperbaiki Scan.

Contoh Cara Kerja C-LOOK



queue = 10, 45, 37, 56, 60, 25, 78, 48, 88, 70, 5, 20

Head starts at 50

Memilih Algoritma Penjadualan *Disk*

- ❖ “*Tak Ada Gading yang Tak Retak*”: Tak ada algoritma yang sempurna untuk semua keadaan.
- ❖ Sangat bergantung pada jumlah dan jenis permintaan, sedangkan permintaan sangat dipengaruhi oleh metode penempatan berkas.
- ❖ *SSTF* dan *Look* sering dipakai sebagai algoritma *default*. *Scan* dan *C-Scan* sesuai untuk sistem dengan beban yang banyak.
- ❖ Oleh karena itu, pada Sistem Operasi terdapat modul terpisah untuk algoritma penjadualan *disk*, sehingga algoritma tsb bisa diganti dengan algoritma yang lain, sesuai keperluan.
- ❖ Algoritma-algoritma tersebut hanya mempertimbangkan *seek time*!
- ❖ *Disk* modern sangat dipengaruhi oleh *rotational latency*!
- ❖ Produsen *disk* mengimplementasikan algoritma penjadualan *disk* pada perangkat keras dengan mempertimbangkan pula *rotational latency*.

Disk Formatting

- ❖ Low level formatting/physical formatting

- Membagi disk jadi beberapa sektor.

- Diisi dengan struktur data *header* dan *trailer* yang menyimpan ECC dan nomor sektor, dan data area

- ❖ Perhitungan ECC saat *write* dan *read* dengan hasil berbeda menunjukkan adanya bit yang salah (*disk corrupted*).

- ❖ Formatting terdiri dari 2 tahap:

- mempartisi disk jadi silinder-silinder.

- logical formatting: pengisian struktur data dan peta dari sektor-sektor yang terisi dan yang kosong.

Boot Block

- ❖ Adalah program yang sudah diinisialisasikan (CPU Register, *system device*, *main memory*) dan dijalankan waktu proses *booting*.
- ❖ Berikutnya mencari kernel, load kernel ke *main memory*, lompat ke alamat awal untuk menjalankan Sistem Operasi.
- ❖ Boot strap disimpan di ROM:
 - Karena ROM tidak perlu inisialisasi dan letaknya tetap sehingga langsung dapat dijalankan prosesor.
 - Karena ROM tidak dapat diinfeksi virus.
 - Dalam bentuk tiny loader yang akan men-load full boot strap dari disk.

Bad Block

❖ Merupakan 1 atau lebih *bad sector*

❖ Solusi:

→ *Simple format* (MS DOS): mencari bad block memberi kode ke FAT entry untuk menggunakan block atau menguncinya.

→ *Sector sparing* (SCSI): mendaftarkan *bad block* saat *level formatting*, menyediakan sektor kosong menggantikan yang rusak.

→ *Sector slipping*: jika menemukan sektor rusak, maka semua data mulai dari sektor tersebut akan digeser maju sampai sektor kosong pertama.

Swap Space Management (1)

- ❖ *swap space* : *virtual memory* menggunakan ruang pada *disk* sebagai perluasan dari *main memory*
- ❖ Jumlah dari *swap space* yang dibutuhkan pada sistem dapat bervariasi tergantung jumlah *physical memory*, jumlah *virtual memory* yang ditopang, dan cara penggunaannya
- ❖ Lokasi *swap space* :
 - *swap space* dapat diusahakan dari berkas sistem yang normal
 - *swap space* dapat berada di partisi *disk* yang terpisah

Swap Space Management (2)

❖ Management:

- 4.3 BSD mengalokasikan swap space saat mulai proses, menahan *text segment* dan *data segment*
- kernel menggunakan *swap maps* untuk melacak penggunaan swap space
- Solaris 2 mengalokasikan *swap space* hanya saat sebuah halaman dikeluarkan dari *physical memory*, bukan saat halaman virtual memory pertama diciptakan

Struktur RAID (1)

- ❖ RAID – *Redundancy Array Independent (atau Inexpensive) Disks* – sebuah set dari beberapa *physical drive* yang dipandang oleh sistem operasi sebagai sebuah *logical drive*.
- ❖ Penggunaan banyak *disk drive* ini meningkatkan kehandalan sistem penyimpanan data melalui *redundancy*.
- ❖ Data didistribusikan ke dalam array dari beberapa *physical drive*.
- ❖ Kapasitas *disk* yang berlebih digunakan untuk menyimpan informasi *parity*, yang menjamin data dapat diperbaiki jika terjadi kegagalan pada salah satu *disk*.

Struktur RAID (2)

- ❖ Peningkatan kinerja dapat dilakukan dengan mengakses banyak *disk* secara paralel.
- ❖ Penggunaan RAID meningkatkan kehandalan dan kinerja.
 - *Mirroring* atau *shadowing* menyimpan duplikat dari setiap *disk*.
 - *Block-interleaved parity* menyimpan blok-blok data pada beberapa *disk* dan blok *parity* pada sebuah *disk*.
 - *Data stripping* menggunakan sekelompok *disk* sebagai satu kesatuan unit penyimpanan, menyimpan bit data secara terpisah pada beberapa *disk* (paralel).

Level RAID (1)

- ❖ RAID dapat dibagi menjadi 6 level yang berbeda.
- ❖ RAID level 0:
 - Tidak ada *redundancy*.
 - Peningkatan kinerja transfer data dengan *data stripping* (*disk* paralel).
- ❖ RAID level 1:
 - *Mirroring*.
 - Penulisan data dilakukan pada kedua *disk*.
- ❖ RAID level 2:
 - *Pengorganisasian dengan error-correcting-code (ECC)*.
 - Jika terjadi kegagalan pada salah satu *disk*, data dapat dibentuk kembali dengan membaca *error-correction bit* pada *disk* lain.

Level RAID (2)

❖ RAID level 3:

→ Pengorganisasian dengan *bit-interleaved parity*.

→ Menggunakan sebuah bit *parity* untuk mengoreksi kesalahan.

❖ RAID level 4:

→ Pengorganisasian dengan *block-interleaved parity*, menggunakan *block-level stripping*.

→ Menyimpan sebuah blok *parity* pada sebuah *disk* terpisah.

❖ RAID level 5:

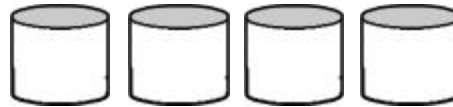
→ *Block-interleaved distributed parity*, mendistribusikan data dan *parity* ke semua *disk*.

❖ RAID level 6:

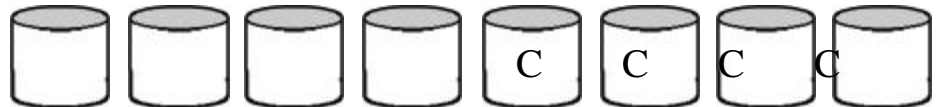
→ $P+Q$ *redundancy scheme*, seperti RAID level 5, menyimpan tambahan informasi jika terjadi kegagalan pada beberapa *disk*.

Level RAID (Gambar)

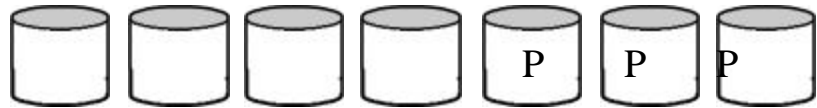
RAID 0: *stripping* tanpa *redundant*



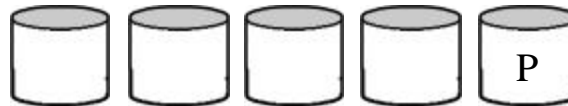
RAID 1: duplikasi *disk* (*mirroring*)



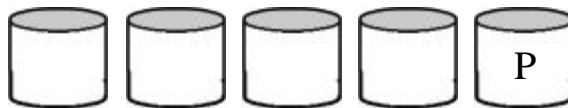
RAID 2: *Memory-style* ECC



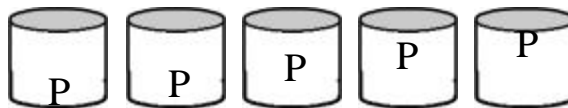
RAID 3: *Bit-interleaved* parity



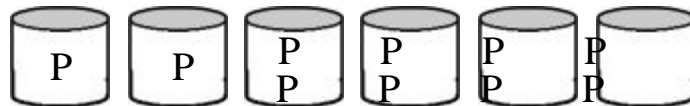
RAID 4: *Block-interleaved* parity



RAID 5: *Block-interleaved* parity
terdistribusi



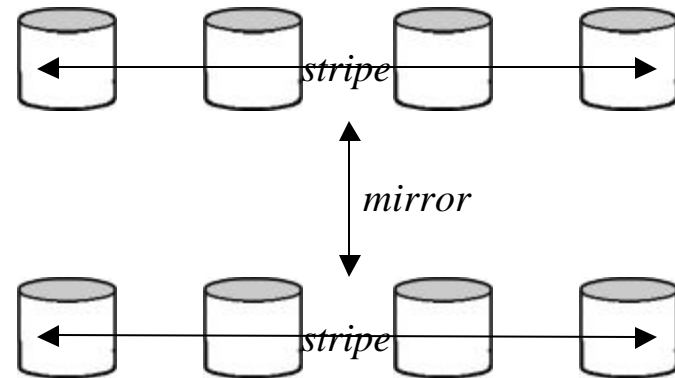
RAID 6: *P+Q* redundancy



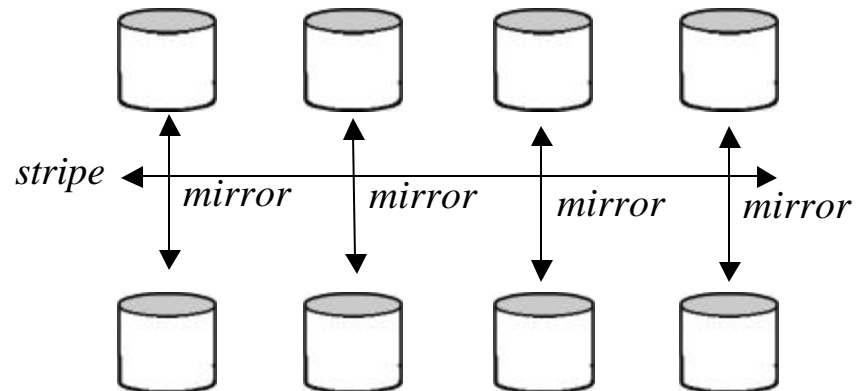
RAID (0 + 1) dan (1 + 0)

❖ Merupakan kombinasi dari RAID level 0 dan 1.

RAID 0 + 1 dengan kegagalan satu *disk*



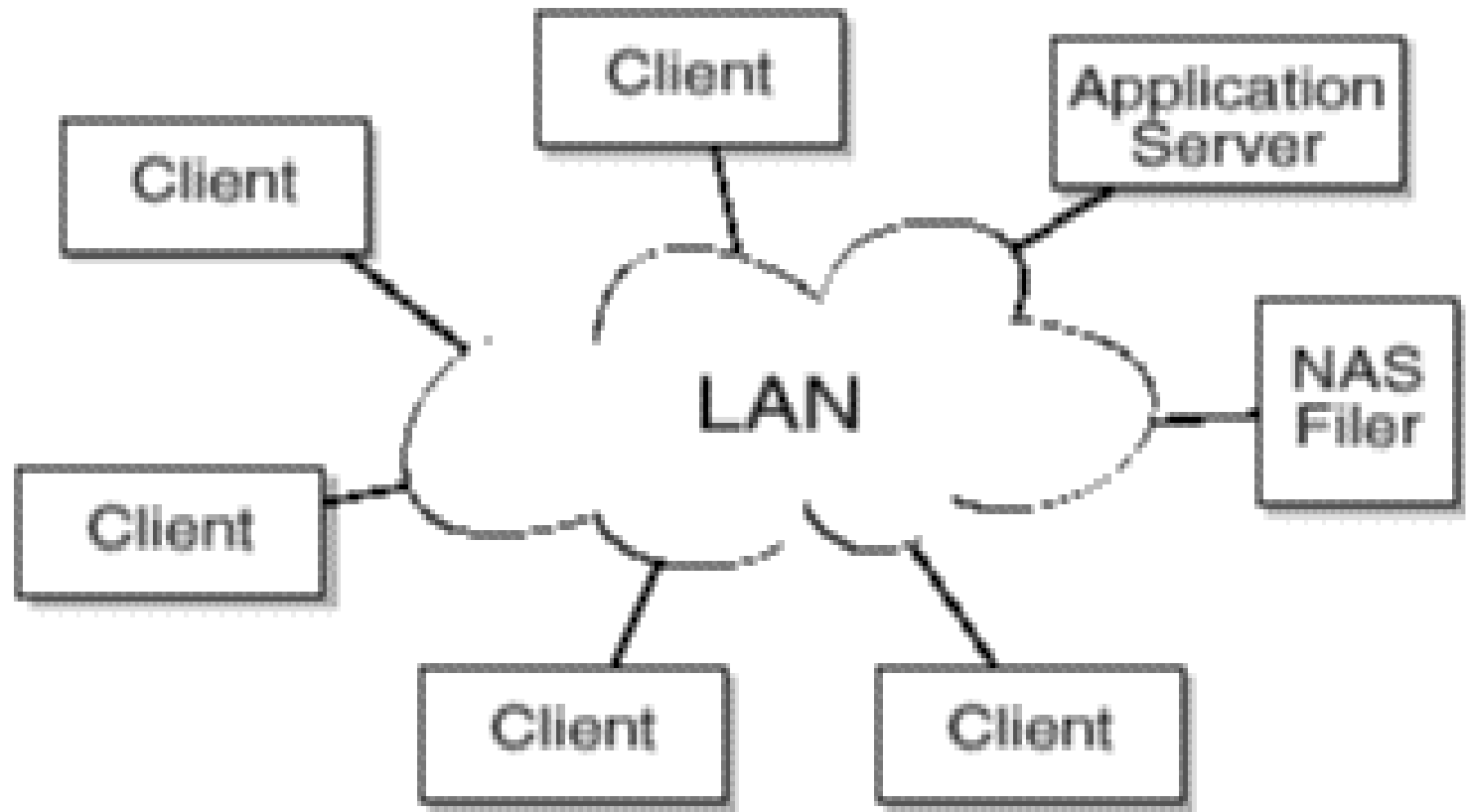
RAID 1 + 0 dengan kegagalan satu *disk*



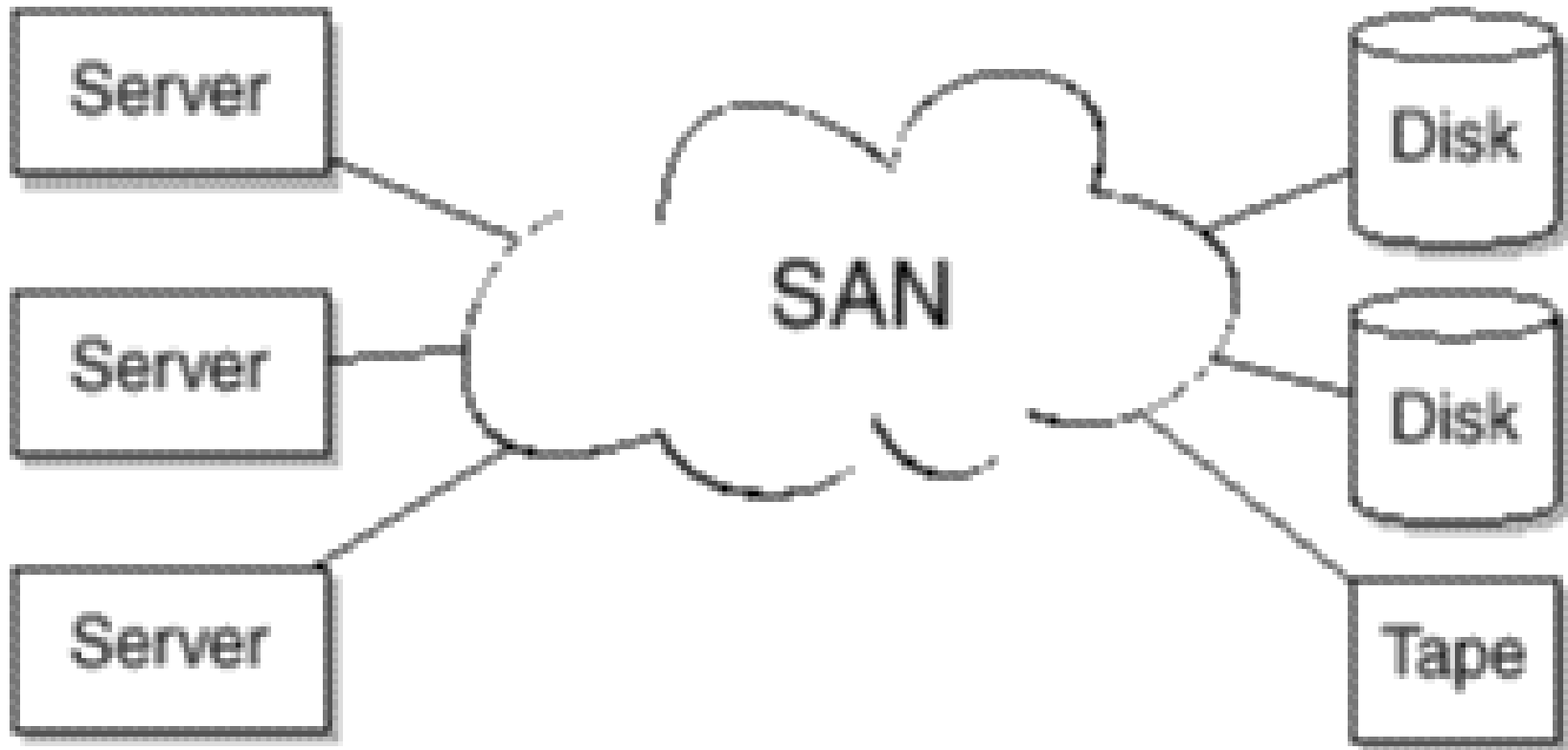
Host-Attached Storage

- ❖ *Storage system* yang terdapat pada komputer
- ❖ Dihubungkan dengan *bus adaptor*
- ❖ Tersedia hanya pada komputer itu sendiri

Network-Attached Storage



Storage-Area Network



NAS vs SAN

NAS

- ❖ *Network-centric*
- ❖ Biasanya digunakan untuk menyatukan data dalam LAN

SAN

- ❖ *Data-centric - a network dedicated to storage of data*
- ❖ Media penyimpanan terpisah dari jalur komunikasi network biasa

Implementasi *stable-storage*

Disk-write menyebabkan 1 dari 3 kemungkinan ini :

1. Successful completion
2. Partial failure
3. Total failure

Tertiary Storage Structure

❖ Karakteristik dari *tertiary storage device*

→ Menggunakan *removable media*

→ Biaya produksi lebih murah

→ Contoh: 1 VCR dengan banyak kaset akan lebih murah daripada 1 VCR yang hanya bisa memainkan satu kaset saja

Removable Disk (1)

Floppy disk

- ❖ Media penyimpanan yang terbuat dari cakram fleksibel tipis yang dilapisi oleh bahan magnetik dan ditutupi oleh plastik.
- ❖ Ciri-ciri *floppy disk* :
 - Memiliki kapasitas kecil > 1 – 2 MB
 - Kemampuan aksesnya hampir secepat hard disk
 - Lebih rentan terhadap gesekan di permukaan magnetiknya

Removable Disk (2)

Magneto-optic disk

- ❖ Data ditulis di atas sebuah piringan keras yang dilapisi oleh suatu bahan magnetik lalu dilapisi pelindung untuk melindungi *head* dari disk tsb
- ❖ Dalam suhu ruangan, medan magnet yang ada tidak dapat digunakan untuk menyimpan bit data sehingga harus ditembakkan laser dari disk head. Tempat yang terkena sinar laser ini dapat digunakan untuk menyimpan bit
- ❖ Head membaca data yang telah disimpan dengan bantuan *Kerr Effect*.

Optical disk

- ❖ *Disk tipe* ini tidak menggunakan magnetik melainkan suatu bahan yang dapat dibelokkan oleh sinar laser

Contoh *Removable Disk*



WORM *Disk*

- ❖ WORM singkatan dari *Write Once Read Many-times*
- ❖ *Aluminium film* yang dilapisi oleh plastik di bagian atas dan bagian bawahnya
- ❖ Untuk menulis data, pada media ini digunakan sinar laser untuk membuat lubang pada aluminiumnya sehingga disk ini hanya dapat ditulis sekali.
- ❖ Ciri-ciri WORM *Disk*:
 - Hanya dapat ditulis sekali
 - Data lebih tahan lama dan dapat dipercaya

Tapes

- ❖ Dapat menyimpan data lebih banyak dari *optical* maupun *magnetic disk cartridge*, harga *cartridge* dari *tape drive* lebih murah namun memiliki *random access* yang lebih lambat
- ❖ *Tape* ini biasa digunakan oleh *supercomputer center* untuk menyimpan data yang besar dan tidak membutuhkan *random access* yang cepat.
- ❖ *Robotic tape changers*: sebuah alat yang dipakai untuk mengganti *tape* dalam skala yang lebih besar
- ❖ *Tape* biasa disimpan di dalam sebuah *library*. *Stacker* menyimpan beberapa *tape*, sedangkan *silo* untuk menyimpan ribuan *tape*.

Operating System Issues

- ❖ Suatu *Operating System* bertugas untuk mengatur *physical devices* serta menampilkan suatu abstraksi dari *virtual machine* ke suatu aplikasi.
- ❖ OS menyediakan dua abstraksi untuk *hard disk*, yaitu:
 - *Raw device* = array dari beberapa blok
 - *File System* = sistem operasi menyusun dan menjadwalkan permintaan interleaved dari beberapa aplikasi

Application Interface

- ❖ Kebanyakan sistem operasi menangani *removable media* hampir sama dengan *fixed disk*, yaitu *cartridge* diformat dan dibuat berkas sistem yang kosong pada *disk*.
- ❖ Sebuah *tape* ditampilkan sebagai *raw media storage* dan ketika sebuah aplikasi membuka sebuah *tape*, dia akan otomatis membuka seluruh *tape*.
- ❖ Beberapa contoh operasi dasar *tape drive*:
 - *Locate* > menetapkan posisi *tape head*
 - *Position* > memberitahu posisi *tape head*
 - *Space* > memindahkan posisi *tape head*

Penamaan Berkas

- ❖ Untuk *fixed disk*, penamaan *berkas* tidak sulit namun tidak demikian dengan *removable disk*.
- ❖ Sekarang OS biasanya mendiadakan masalah penamaan berkas ini dan membiarkan aplikasi dan user untuk mengatur penamaan berkas tersebut.
- ❖ Untuk mengatasi masalah penamaan, beberapa media sekarang sudah distandarisasi sehingga semua komputer akan menggunakannya dengan cara yang sama. Contoh: CD musik

Hierarchical Storage Management

- ❖ HSM memperluas storage hierarchy di atas primary memory dan secondary storage untuk membentuk tertiary storage, yang biasa diimplementasikan dalam bentuk juke box dari kumpulan tapes atau removable disk.
- ❖ Berkas-berkas yang ukurannya kecil dan sering digunakan dibiarkan berada di dalam disk.
- ❖ Berkas-berkas yang ukurannya besar dan jarang digunakan disimpan dalam jukebox.
- ❖ HSM biasanya digunakan pada super komputer dan large installations yang menggunakan data-data dalam volume sangat besar.

Performance Issues (1)

Speed

- ❖ Kecepatan dari *tertiary storage* dipengaruhi oleh 2 aspek: *bandwidth* dan *latency*.
- ❖ *Sustained bandwidth* adalah waktu rata-rata ketika melakukan transfer dalam ukuran yang besar, yaitu jumlah byte dibagi waktu transfer. Istilah *bandwidth* dari suatu drive dapat dimengerti sebagai *sustained bandwidth*.
- ❖ *Effective bandwidth* menghitung rata-rata waktu IO, termasuk waktu untuk *seek* atau *locate* dan waktu penggantian *cartridge* di dalam *jukebox*.
- ❖ *Access latency* adalah waktu yang dibutuhkan untuk menemukan lokasi dari suatu data.

Performance Issues (2)

Reliability

- ❖ *Fixed hard disk* lebih dapat dipercaya dibandingkan *removable magnetic disk*.
- ❖ *Optical disk* dianggap yang paling dapat dipercaya karena lapisan yang menyimpan bits dilindungi oleh plastik atau lapisan kaca transparan.
- ❖ Tingkat *reliability* dari *magnetic tape* berbeda-beda tergantung dari jenis *drive*.
- ❖ Satu kelemahan *fixed magnetic disk* adalah rusaknya data jika ada *crash* pada *hard disk*. Sementara kegagalan pada *tape drive* atau *optical drive* tidak merusak data.