Sistem Terdistribusi

TIK-604

Sinkronisasi

Kuliah 06-07: 18 s.d 20 & 25 s.d 27 Maret 2019

Husni

Hari ini...

- Topik sebelumnya...
  - *Naming* (Selesai)
- Bahasan hari ini
  - Sinkronisasi
    - *Coordinated Universal Time* (UTC)
    - Tracking Time pada suatu Computer
    - Sinkronisasi jam: *Cristian's Algorithm*, *Berkeley Algorithm* dan *Network Time Protocol* (NTP)
- Pengumuman
  - ??

Sinkronisasi

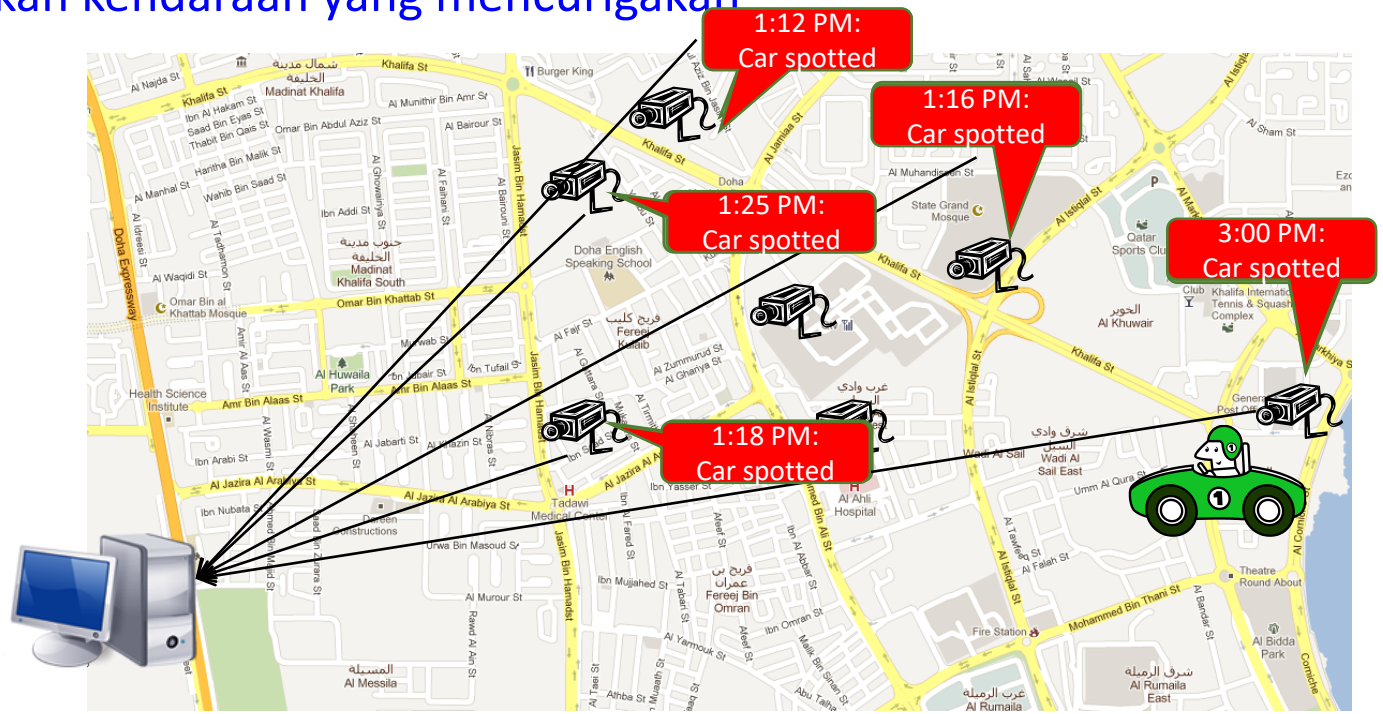
- Sampai sejauh ini, telah didiskusikan:
  - Bagaimana entitas berkomunikasi satu dengan lainnya
  - Bagaimana entitas dinamai dan diperkenalkan

- Sebagai tambahan terhadap requirements di atas, entitas-entitas dalam sister sering harus bekerjasama (*cooperate*) dan menyerempakkan (*synchronize*) untuk menyelesaikan masalahyang diberikan dengan benar
  - Misal: Dalam suatu sistem berkas terdistribusi, proses-proses harus berinskronisasi dan bekerjasama sehingga dua proses tidak dibolehkan untuk menulis ke bagian yang sama dari suatu file, pada waktu yang sama

Perlunya Sinkronisasi: Contoh 1

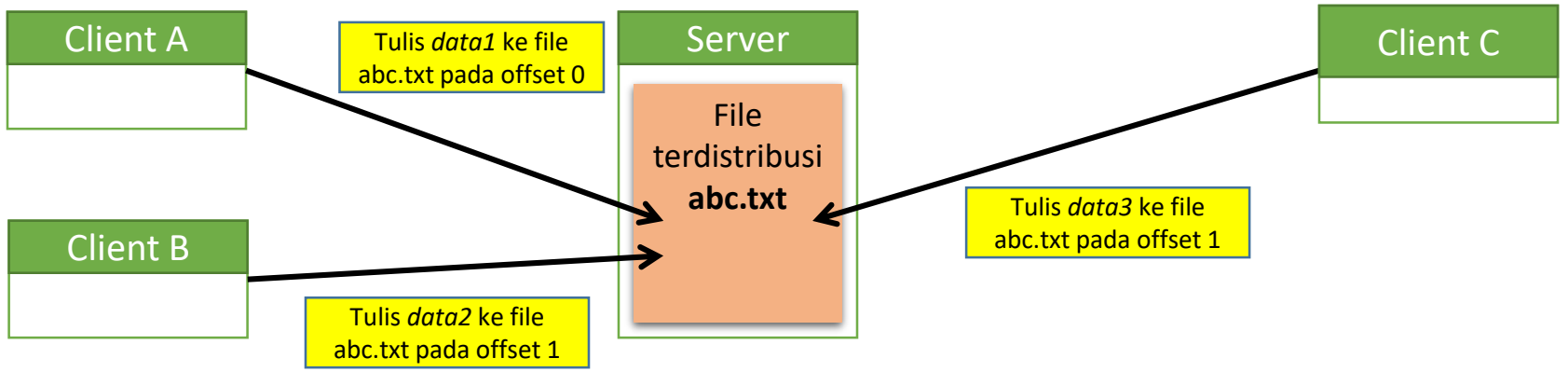
- *Pelacakan kendaraan* dalam suatu *Sistem Pengawasan Kota* menggunakan *Jaringan Sensor Kamera Terdistribusi*
  - **<u>Tujuan:</u> untuk mengetahui jejak (*track*) dari kendaraan yang mencurigakan**
  - Camera Sensor Nodes dipasang di seluruh kota
  - Setiap Camera Sensor yang mendeteksi suatu kendaraan melaporkan waktunya ke server pusat
  - Server merekam pergerakan kendaraan yang mencurigakan

Jika node-node sensor tidak mempunyai versi waktu yang konsisten, hasil lacakan kendaraan tersebut tidak dapat diandalkan.



Perlunya Sinkronisasi: Contoh 2

- Penulisan file dalam suatu Sistem Berkas Tersebar (*Distributed File System*)



Jika client-client yang terdistribusi tidak mensinkronkan operasi tulisnya ke file terdistribusi, maka data dalam file dapat mengalami kerusakan.

Taksonomi Sinkronisasi

Alasan bagi Sinkronisasi & Kerjasama	Entitas harus menyepakati pengurutan events	Entitas harus men-share sumber daya bersama
Contoh	Pelacakan kenddaraan dalam Camera Sensor Network; transasksi Keuangan dalam sistem e-Commerce Terdistribusi	Pembacaan dan Penulisan dalam suatu Sistem File Terdistribusi
Requirement bagi Entitas-entitas	Entitas harus mempunyai suatu pemahaman bersama mengenai waktu lintas komputer	Entitas harus berkoordinasi dan menyepakati kapan dan bagaimana mengakses sumber daya
Topik yang akan didiskusikan	**Sinkronisasi Waktu** (*Time Synchronization*)	**Pengecualian Bersama** (*Mutual Exclusion*)

Ikhtisar

Kuliah hari ini

- Sinkronisasi Waktu
  - Sinkronisasi Jam Fisik (atau Sinkronisasi Jam)
    - Di sini, waktu aktual pada komputer dalam jaringan disinkronkan
  - Sinkronisasi Jam Logis
    - Komputer dalam jaringan disinkronkan berdasarkanpada pengurutan relatif dari kejadian-kejadian
- Pengeluaran Bersama (*Mutual Exclusion*)
  - Bagaimana mengkoordinasikan antar proses yang mengakses sumber daya yang sama?
- Algoritma Pemilihan (*election*)
  - Di sini, grup entitas memilih satu entitas sebagai koordinator utuk menyelesaikan suatu persoalan.

Kuliah berikutnya

Ikhtisar

- Sinkronisasi Waktu
  - Sinkronisasi Jam (Fisik)
  - Sinkronisasi Jam Logis
- Pengecualian Bersama (MutEx)
- Algoritma Pemilihan

Sinkronisasi Jam

- Sinkronisasi jam merupakan mekanisme untuk menyerempakkan waktu dari semua komputer di dalam suatu Sistem Terdistribusi (SisTer)

- Akan didiskusikan:
  - *Coordinated Universal Time* (UTC)
  - Pelacakan waktu pada suatu Komputer
  - Algortima sinkronisasi Jam
    - Algoritma Cristian
    - Algoritma Berkeley
    - *Network Time Protocol* (NTP)

Sinkronisasi Jam

- *Coordinated Universal Time* (UTC)
- Pelacakan waktu pada suatu Komputer
- Algortima sinkronisasi Jam
  - Algoritma Cristian
  - Algoritma Berkeley
  - *Network Time Protocol* (NTP)

Coordinated Universal Time (UTC)

- Semua komputer biasanya disinkronkan ke suatu jam standard bernama Coordinated Universal Time (UTC)
  - UTC merupakan standard waktu primer yang digunakan oleh dunia untuk mengatur jam dan waktu
- UTC dibroadcast melalui satelit
  - Layanan broadcasting UTC menyediakan akurasi 0.5 msec
- Server-server komputer dan layanan online dengan **UTC receivers** dapat disinkronkan melalui broadcast satelit
  - Banyak protokol sinkronisasi terkenal dalam sistem terdistribusi menggunakan UTC sebagai waktu rujukan untuk mensinkronkan jam dari komputer-komputer mereka.

Sinkronisasi Jam

- *Coordinated Universal Time* (UTC)
- Pelacakan waktu pada suatu Komputer
- Algortima sinkronisasi Jam
  - Algoritma Cristian
  - Algoritma Berkeley
  - *Network Time Protocol* (NTP)

Pelacakan Waktu pada Komputer

- Bagaimana suatu komputer memelihara catatan waktunya?
  - Setiap komputer mempunyai hardware *timer*
    - Timer ini menyebabkan suatu interrupt ‘H’ kali dalam sedetik
  - Interrupt handler menambahkan 1 ke Jam Softwarenya (C)
- Persoalan dengan jam pada suatu komputer
  - Dalam praktek, hardware timer tidak tepat/teliti
    - Ia tidak benar-benar berinterupsi ‘H’ kali sedetik dikarenakan ketidak-sempurnaan material hardware dan variasi temperatur
    - Komputer menghitung waktu lebih lambat atau lebih cepat daripada waktu aktual
  - Sehingga muncul *Clock Skew* yaitu kemiringan antara:
    - jam komputer dan waktu sebenarnya (misalnya UTC)

Clock Skew

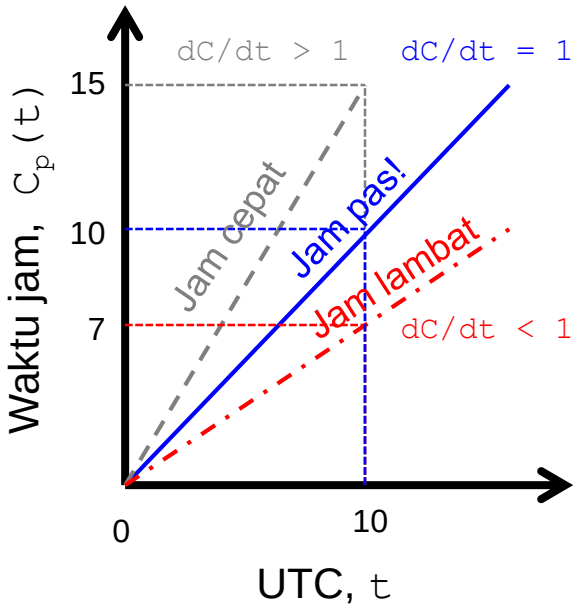
- Pada saat waktu UTC adalah `t`, misalkan jam pada komputer mempunyai waktu `C(t)`

- Tiga jenis jam yang mungkin
  - Jam sempurna:
    - Timer ber-detak ‘H’ interupsi sedetik

      `dC/dt = 1`
  - Jam cepat:
    - Timer berdetak lebih dari ‘H’ kali sedetik

      `dC/dt > 1`
  - Jam lambat:
    - Timer berdetak kurang dari ‘H’ interupsi per detik

      `dC/dt < 1`



Clock Skew (Lanj.)

- Frequency dari jam didefinisikan sebagai ratio dari jumlah detik yang dihitung oleh jam software untuk setiap detik UTC

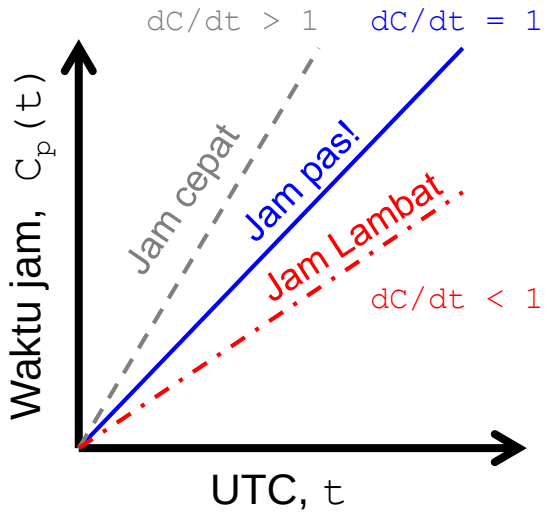
  Frequency = dC/dt

- Skew dari jam tersebut didefinisikan sebagai sejauh mana frekuensi berbeda dari jam yang sempurna

  Skew = dC/dt – 1

- Karenanya,

$$Skew \begin{cases} > 0 & \text{untuk jam yang cepat} \\ = 0 & \text{untuk jam yang sempurna} \\ < 0 & \text{untuk jam yang lambat} \end{cases}$$



Maximum Drift Rate dari Jam

- Pabrikan jam menetapkan batas atas dan bawah yang *clock skew* dapat berfluktuasi (turun-naik). Nilai ini dikenal sebagai *maximum drift rate (ρ)*

$$1 - \rho <= dC/dt <= 1 + \rho$$

- Seberapa jauh dua jam dapat berpisah jarak?
  - Jika dua jam menyimpang dari UTC dalam arah berlawanan, pada waktu Δt setelah keduanya disinkronkan, jaraknya sebanyak $2\rho\Delta t$ detik

- Menjamin simpangan maksimum antar komputer dalam suatu sistem terdistribusi
  - Jika maximum drift yang dibolehkan dalam suatu SisTer adalah δ detik, maka jam dari setiap komputer harus disinkron-ulangkan setidaknya setiap $\delta/2\rho$ detik.

Sinkronisasi Jam

- *Coordinated Universal Time* (UTC)
- Pelacakan waktu pada suatu Komputer
- Algortima sinkronisasi Jam
  - Algoritma Cristian
  - Algoritma Berkeley
  - *Network Time Protocol* (NTP)

Algoritma Cristian

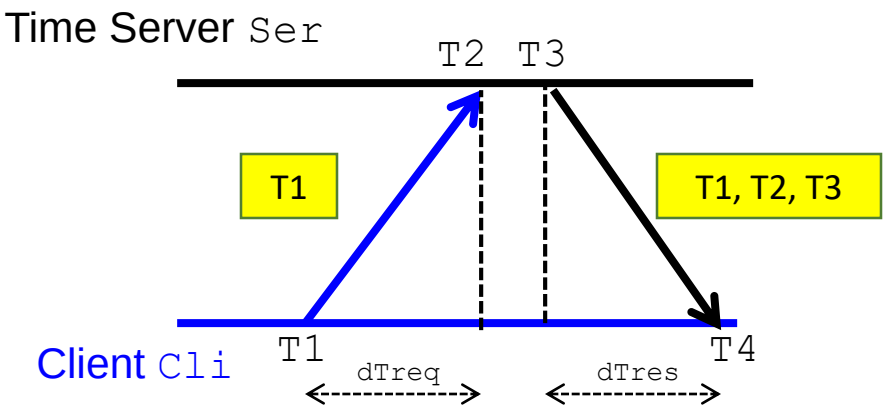
- Flaviu Cristian (1989) menawarkan suatu algoritma untuk menyerempakkan komputer-komputer dalam jaringan dengan suatu server waktu (*time server*)

- Gagasan dasarnya:
  - Identifikasi suatu *network time server* yang mempunyai sumber waktu akurat (misal server waktunya mempunyai suatu UTC *receiver*)
  - Semua client menghubungi *network time server* untuk sinkronisasi

- Namun, delay jaringan terjadi pada saat client menghubungi *time server* menghasilkan waktu yang ketinggalan
  - Algoritma tersebut memperkirakan delay jaringan dan menetapkan kompensasi untuk itu.

Algoritma Cristian: Pendekatan

- Client Cli mengirimkan request ke Time Server Ser, waktu jam lokalnya dicap sebagai T1
- Ser akan merekam waktu yang diterima T2 sesuai dengan jam lokalnya
  - dTreq delay jaringan selama transmisi request



- Ser membalas Cli pada waktu lokalnya T3, setelah T1 dan T2
- Cli menerima balasan pada waktu lokalnya T4
  - dTres adalah delay jaringan selama transmisi *response*
- Sekarang Cli mempunyai informasi T1, T2, T3 dan T4
- **Dengan menganggap delay transmisi dari Cli→Ser dan Ser→Cli adalah sama, maka:**

T2-T1 ≈ T4–T3

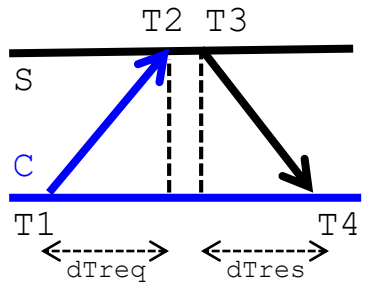
Algoritma Christian: Sinkronisasi Waktu Client

- Client C mengestimasi offset θ relatif terhadap Time Server S

  θ = T3 + dTres – T4

  = T3 + ((T2-T1)+(T4-T3))/2 – T4

  = ((T2-T1)+(T3-T4))/2



- Jika **θ > 0 atau θ < 0,** maka waktu client harus dinaikkan atau diturunkan sebesar **θ** detik

Sinkronisasi Waktu Bertahap pada Client

- Daripada mengubah waktu secara dramatis dengan **θ** detik, biasanya waktu secara gradual disinkronkan
  - Jam software di-update pada suatu kecepatan lebih kecil/lebih besar saat timer melakukan interupsi

Catatan: Mensetup jam ke belakang (katakanlah, jika `θ < 0`) tidak dibolehkan dalam SisTer karena penurunan nilai jam pada suatu komputer mempunyai efek merugikan pada beberapa aplikasi (misal: program *make*)

Algoritma Cristian: Diskusi

1. Asumsi mengenai delay transmisi paket

- Algoritma Cristian menganggap bahwa waktu round-trip untuk pertukaran *message* di jaringan adalah cukup pendek
- Algoritma menganggap bahwa *delay* untuk *request* dan respon adalah sama.
  - Akankah trend peningkatan lalulintas Internet menurunkan akurasi dari algoritma ini?
  - Dapatkah algoritma ini menangani delay asymmetry yang sudah lazim di Internet?
  - Dapatkah client berupa entitas mobile dengan koneksi yang intermittent?

Algoritma Cristian dimaksudkan untuk mensinkronkan komputer-komputer dalam intranet

2. Pendekatan probabilistik untuk menghitung delay

- Tidak ada batasan ketat mengenai maximum drift antar jam dari komputer-komputer

3. Kegagalan Time server atau Jam Server Salah

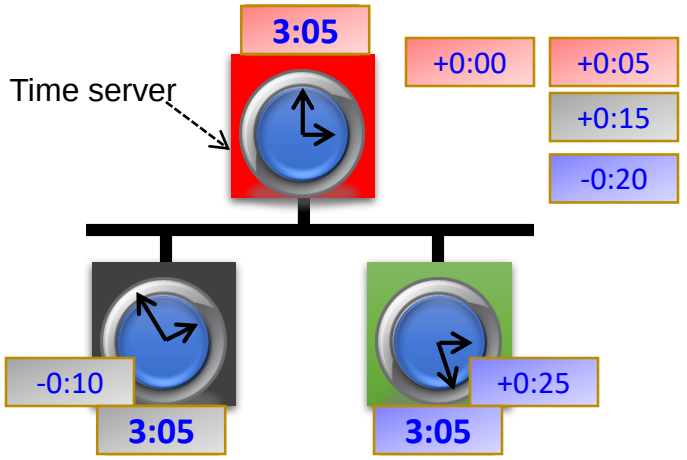
- Jam yang salah pada time server mengakibatkan ketidakakuratan jam di dalam seluruh SisTer
- Kegagalan dari time server akan mengakibatkan tidak mungkinnya sinkronisasi.

Sinkronisasi Jam

- *Coordinated Universal Time* (UTC)
- Pelacakan waktu pada suatu Komputer
- Algortima sinkronisasi Jam
  - Algoritma Cristian
  - Algoritma Berkeley
  - *Network Time Protocol* (NTP)

Algoritma Berkeley

- Algoritma Berkeley merupakan pendekatan terdistribusi untuk sinkronisasi waktu
- Pendekatan:
    1. Suatu time server secara berkala (sekitar sekali dalam 4 menit) mengirimkan waktunya ke semua komputer dan mengambil selisih waktunya
    2. Komputer menghitung selisih waktu dan kemudian membalas
    3. Server menghitung suatu selisih waktu rerata untuk setiap komputer
    4. Server memerintahkan semua komputer untuk mengupdate waktu mereka (dengan sinkronisasi waktu gradual)



Algoritma Berkeley: Diskusi

1. Asumsi mengenai delay transmisi paket

- Algoritma Berkeley memprediksi delay jaringan (mirip algoritma Cristian)
- Karena itu, ini efektif dalam intranet, dan tidak akurat dalam jaringan area luas

2. Tidak diperlukan UTC Receiver

- Jam-jam dalam sistem bersinkronisasi dengan pererataan semua waktu komputer

3. Menurunkan efek dari jam rusak

- Toleransi salah perhitungan rerata, dimana jam pencilan (*outlier*) diabaikan, dapat dengan mudah dikerjakan dalam algotima Berkeley

4. Kegagalan Time server dapat disembunyikan

- Jika suatu time server rusak, komputer lain dapat dipilih sebagai suatu time server

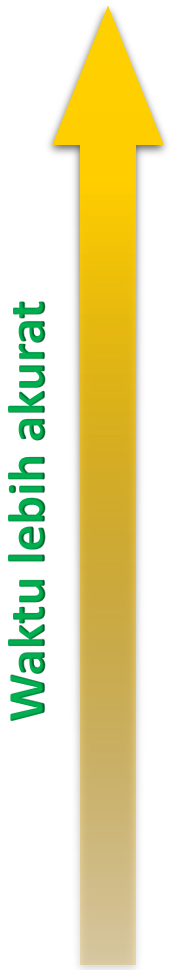
Sinkronisasi Jam

- *Coordinated Universal Time* (UTC)
- Pelacakan waktu pada suatu Komputer
- Algoritma sinkronisasi Jam
  - Algoritma Cristian
  - Algoritma Berkeley
  - *Network Time Protocol* (NTP)

Network Time Protocol (NTP)

- NTP mendefinisikan suatu arsitektur untuk layanan waktu dan protokol untuk mendistribusikan informasi waktu di Internet
- Dalam NTP, server-server dikoneksikan dalam suatu hirarki logis bernama *synchronization subnet*
- Level-level dari *synchronization subnet* disebut sebagai *strata*
  - Server-server Strata 1 mempunyai informasi waktu yang paling akurat (terhubung ke suatu UTC receiver)
  - Server-server dalam setiap strata bertindak sebagai time server untuk server-server di strata lebih rendah.

Organisasi Hirarkis dari Server NTP



Strata 1	• Strata ini berisikan server-sever primer yang secara langsung dikoneksikan ke UTC receivers
Strata 2	• Strata 2 adalah server-server sekunder yang disinkronkan secara langsung dengan server-server primer
Strata 3	• Strata 3 mensinkronkan waktunya dengan server-server Strata 2
Strata terakhir	• Komputer *end user* mensinkronkan waktunya dengan server-server pada strata lapisan lebih atas.

Operasi dari Protokol NTP

- Ketika time server **A** menghubungi time server **B** untuk sinkronisasi
  - Jika **strata(A) <= strata(B)**, maka **A** tidak mensinkronkan dengan B
  - Jika **strata(A) > strata(B)**, maka:
    - Time server **A** mensinkronkan dengan **B**
    - Suatu algortima mirip algoritma Cristian digunakan untuk mensinkronkan. Namun, sampel statisik lebih besar diambil sebelum dilakukan update jam tersebut.
    - Time server **A** mengupdate strata-nya

      strata(A) = strata(B) + 1

Rancangan NTP: Diskusi

Sinkronisasi Akurat ke Waktu UTC

- NTP memungkinkan client lintas Internet untuk disinkronkan secara akurat ke UTC
- Delay message yang besar dan variatif ditoleransi dengan penyaringan statistik dari timing data dari server-server berbeda

Skalabilitas

- Server NTP secara hirarkis terorganisir untuk mempercepat sinkronisasi, dan untuk menskalakan jumlah client dan server yang besar

Reliabilitas dan Toleransi Kegagalan

- Terdapat time server redundan, dan jalur redundan antar time servers
- Arsitekturnya menyediakan lanayan reliable yang dapat mentoleransi putusnya konektifitas yang lama
- Suatu synchronization subnet dapat mengkonfigurasi-ulang saat server-server menjadi *unreachable*. Sebagai contoh, jika server Strata 1 rusak, maka ia dapat menjadi suatu server sekunder Strata 2.

Keamanan

- Protokol NTP menggunakan otentikasi untuk memeriksa timing message yang berasal dari sumber yang dipercaya yang diklaim.

Sinkronisasi Jam: Rangkuman

- Jam fisik pada komputer tidaklah akurat
- Algoritma sinkronisasi jam menyediakan mekanisme untuk mensinkronkan jam-jam pada komputyer jaringan di dalam suatu SisTer
  - Komputer-komputer pada jaringan lokal menggunakan berbagai algortima untuk sinkronisasi
    - Beberapa algoritma (seperti algoritma Cristian) mensinkronkan waktu dengan menghubungi time server terpusat
    - Beberapa algoritma (seperti algoritma Berkeley) mensinkronkan dengan cara terdistribusi melalui pertukaran informasi waktu pada berbagai komputer
  - NTP menyediakan arsitektur dan protokol untuk sinkronisasi waktu pada jaringan area luas seperti Internet

Selanjutnya...

- Jam Logis:
  - Jam Lamport
  - Jam Vektor

- *Mutual Exclusion*
  - Bagaimana mengkoordinasikan antar proses yang mengakses sumber daya yang sama?

Sekarang...

- Diskusi terakhir:
  - Sinkronisasi: UTC, pelacakan waktu pada komputer, sinkronisasi jam fisik
- Bahasan saat ini:
  - Sinkronisasi Jam Logis
    - Jam Lamport dan Vektor
  - Pengantar *Distributed Mutual Exclusion*
- Pengumuman:
  - Progress Proyek?

Bahasan Tentang Sinkronisasi

Kuliah Sebelumnya

- Sinkronisasi Waktu
  - Sinkronisasi Jam Fisik (atau Sinkronisasi Jam)
    - Di sini, waktu aktual pada komputer disinkronkan
  - Sinkronisasi Jam Logis
    - Komputer disinkronkan berdasarkan pada urutan relatif dari kejadian-kejadian (events)

Kuliah Hari ini

- Pengeluaran Bersama (*Mutual Exclusion*)
  - Bagaimana koordinasi antar proses yang mengakses sumber daya sama?
- Algoritma Pemilihan
  - Di sini, sekelompok entitas memilih satu entitas sebagai koordinator untuk penyelesaian suatu masalah.

Tugas Membaca Mandiri

Ikhtisar

- Sinkronisasi Waktu
  - Sinkronisasi Jam
  - Sinkronisasi Jam Logis
- Mutual Exclusion
- Algoritma Pemilihan

Mengapa Jam Logis?

- Lamport (1978) memperlihatkan bahwa:
  - Sinkronisasi jam tidak diperlukan di dalam semua scenario
    - Jika dua proses tidak berinteraksi maka tidak perlu jam kedua proses tersebut disinkronkan
  - Sering sekali, adalah cukup jika proses menyepakati *urutan* terjadinya kejadian-kejadian di dalam SisTer
    - Sebagai contoh, untuk suatu utilitas make terdistribusi, adalah cukup untuk mengetahui jika suatu file input telah dimodifikasi sebelumatau setelah file obyeknya.

Jam Logis

- Jam logis digunakan untuk mendefinisikan urutan kejadian tanpa mengukur waktu fisik kapan kejadian itu terjadi

- Akan didiskusikan dua tipe jam logis:
  1. Jam logis Lamport (atau jam Lamport)
  2. Jam vektor

Jam Logis

- Akan didiskusikan dua jenis jam logis:

1. Jam Lamport
2. Jam Vektor

Jam Logis Lamport

- Lamport menganjurkan mempertahankan jam logis pada proses untuk melacak urutan peristiwa

- Untuk menyinkronkan jam logis, Lamport mendefinisikan hubungan yang disebut "terjadi sebelum" (***happened-before***)

- Ekspresi **a→b** (dibaca sebagai "**a** terjadi sebelum **b**") berarti bahwa semua entitas dalam DS setuju bahwa peristiwa a terjadi sebelum peristiwa b

Relasi Happened-before

- Hubungan yang terjadi sebelum ini dapat diamati secara langsung dalam dua situasi:
    1. Jika a dan b adalah peristiwa dalam proses yang sama, dan a terjadi sebelum b, maka **a→b** adalah benar
    2. Jika a adalah peristiwa pesan m sedang dikirim oleh suatu proses, dan b adalah peristiwa m (yaitu, pesan yang sama) diterima oleh proses lain, maka **a→b** adalah benar
- Hubungan yang terjadi sebelum ini bersifat transitif
    - Jika **a→b** dan **b→c**, maka **a→c**

Nilai Waktu dalam Jam Logis

- Untuk setiap peristiwa a, tetapkan nilai waktu logis C (a) di mana semua proses setuju (C masih sesuai dengan proses dan tidak dengan peristiwa tersebut, tetapi diperbarui ketika peristiwa itu terjadi)

- Nilai waktu untuk peristiwa memiliki properti yang:
  - Jika **a→b**, maka **C(a)< C(b)**

Properti dari Jam Logis

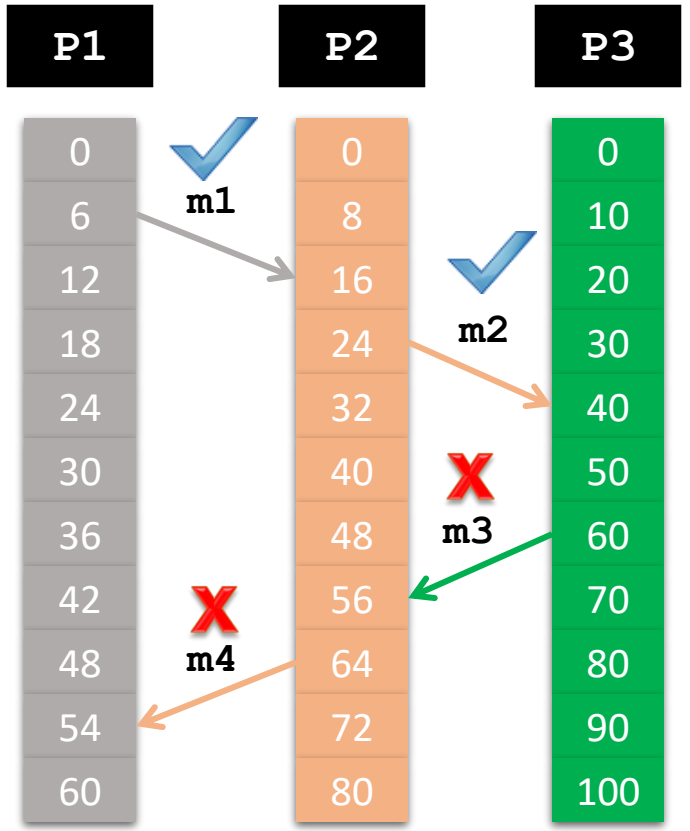
- Dari hubungan “terjadi sebelum”, kita dapat menyimpulkan bahwa:
  - Jika dua peristiwa a dan b terjadi dalam proses yang sama dan **a→b**, maka **C(a)** dan **C(b)** diberikan nilai waktu sedemikian rupa sehingga **C(a) < C(b)**
  - Jika a adalah peristiwa pengiriman pesan m dari satu proses (katakanlah P1), dan b adalah peristiwa menerima m (yaitu pesan yang sama) pada proses lain (katakanlah, P2), maka:
    - Nilai waktu C_1 **(a)** dan C_2 **(b)** diberikan sedemikian rupa sehingga kedua proses setuju bahwa C_1 **(a) < ** C_2 **(b)**
  - Waktu jam C harus selalu maju (meningkat), dan tidak pernah mundur (menurun)

Sinkronisasi Jam Logis

- Tiga proses P1, P2 dan P3 berjalan dengan laju yang berbeda

- Jika proses berkomunikasi antara satu sama lain, mungkin ada perbedaan dalam menyepakati pengurutan peristiwa
  - Pengurutan pengiriman dan penerimaan pesan m1 dan m2 sudah benar

  - Namun, m3 dan m4 melanggar hubungan “terjadi sebelum”

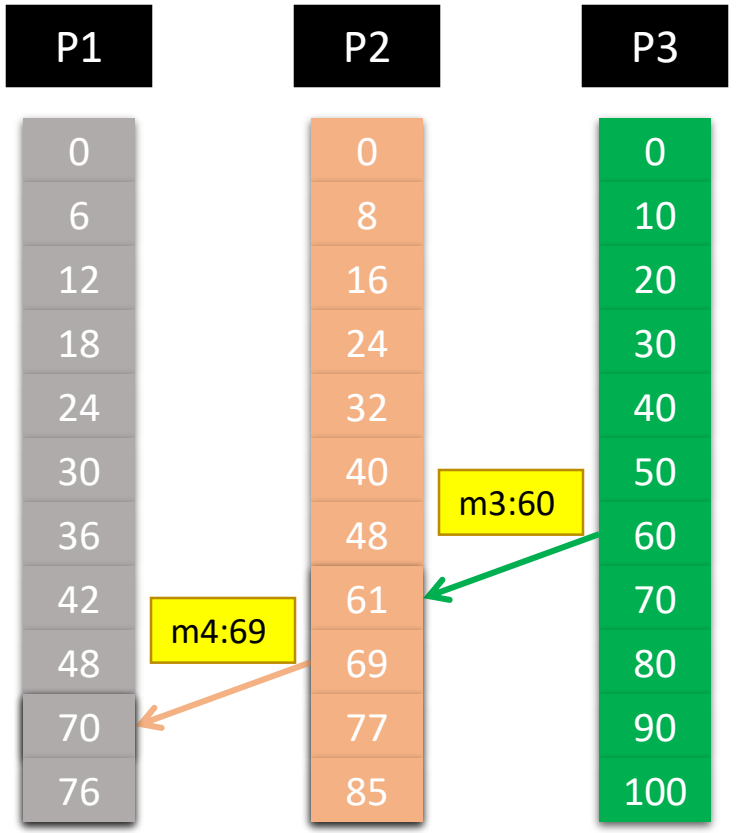


Algoritma Jam Lamport

- Ketika pesan sedang dikirim:
  - Setiap pesan membawa stempel waktu (*timestamp*) sesuai dengan jam logis pengirim

- Saat pesan diterima:
  - Jika jam logis penerima kurang dari waktu pengiriman pesan dalam paket, maka sesuaikan jam penerima sedemikian rupa sehingga:

```
currentTime = timestamp + 1
```



Jam Logis Tanpa Jam Fisik

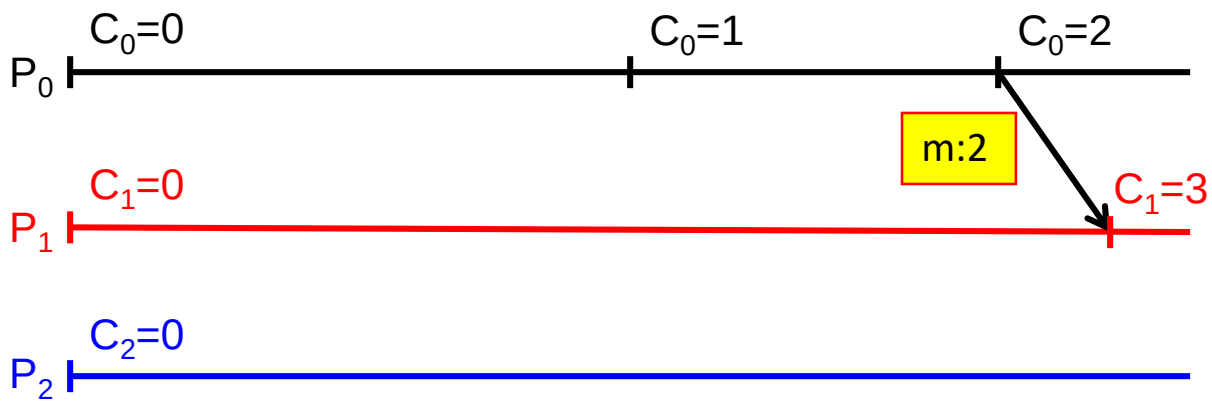
- Contoh sebelumnya mengasumsikan bahwa ada jam fisik di setiap komputer (mungkin berjalan pada laju yang berbeda)

- Bagaimana cara melampirkan nilai waktu pada suatu peristiwa ketika tidak ada jam global?

Implementasi Jam Lamport

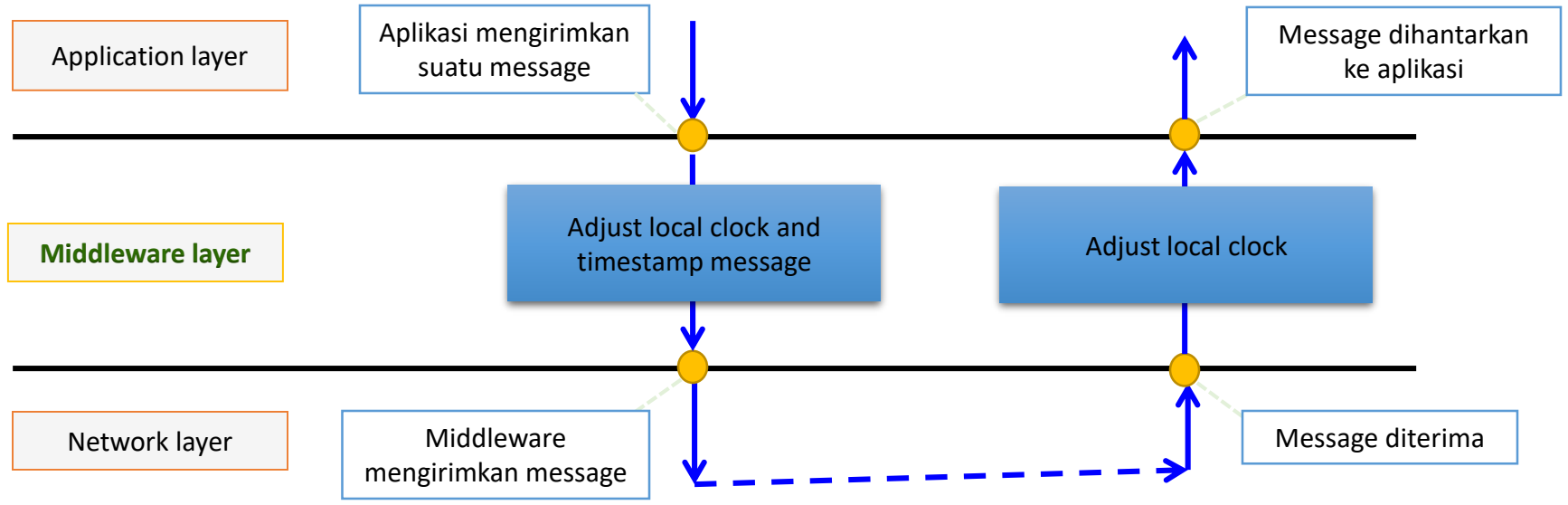
- Setiap proses P_i memelihara konter lokal C_i dan menyesuaikan penghitung ini sesuai dengan aturan berikut:

    1. Untuk dua peristiwa berturut-turut yang terjadi di dalam P_i, C_i dinaikkan sebesar 1
    2. Setiap kali pesan m dikirim oleh proses P_i , m diberikan suatu timestamp $ts(m) = C_i$
    3. Setiap kali pesan diterima oleh suatu proses P_j, P_j menyesuaikan konter lokalnya C_j menjadi **$max(C_j, ts(m)) + 1$**



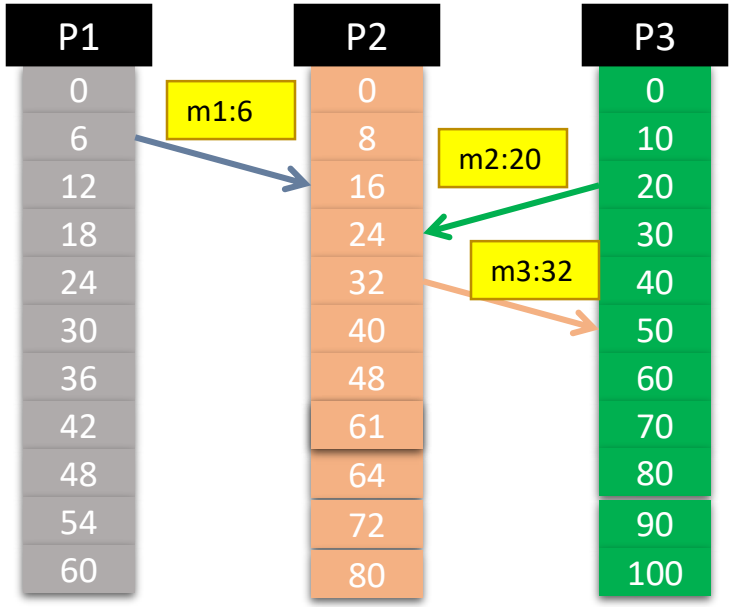
Penempatan Jam Logis

- Di komputer, beberapa proses dapat menggunakan jam logis yang berbeda
- Namun, alih-alih setiap proses mempertahankan jam logisnya sendiri, jam logis tunggal dapat diimplementasikan dalam *middleware* sebagai layanan waktu



Keterbatasan dari Jam Lamport

- Jam Lamport memastikan bahwa jika **a→b**, maka **C(a) < C(b)**
- Namun, ia tidak mengatakan apa-apa tentang dua kejadian sembarang (konkuren atau independen) a dan b dengan hanya membandingkan nilai waktu mereka
  - Untuk dua peristiwa sembarang **a** dan **b**, **C(a) < C(b)** tidak berarti bahwa **a→b**
- Contoh:



Bandingkan m1 dan m3

P2 dapat menyimpulkan bahwa m1 →**m3**

Bandingkan m1 dan m2

P2 tidak dapat menyimpulkan bahwa m1 →**m2** or **m2→m1**

Jam Lamport: Rangkuman

- Lamport menyarankan menggunakan jam logis
  - Proses menyinkronkan berdasarkan nilai waktu dari jam logis mereka daripada nilai waktu absolut dari jam fisik mereka
- Aplikasi mana dalam DS yang memerlukan jam logis?
  - Aplikasi dengan pengurutan peristiwa yang terbukti
    - Sinkronisasi jam fisik yang sempurna sulit dicapai dalam praktik
  - Aplikasi dengan acara langka
    - Peristiwa jarang dihasilkan, dan overhead sinkronisasi jam fisik tidak disuguhkan
- Namun, jam Lamport tidak dapat menjamin urutan peristiwa yang sempurna dengan hanya mengamati nilai waktu dari dua peristiwa sembarang.

Jam Logis

- Akan dikaji dua jenis jam logis:

1. Jam Lamport
2. Jam Vektor

Jam Vektor

- Jam vektor diusulkan untuk mengatasi keterbatasan pada jam Lamport
  - Properti yang menyimpulkan bahwa a terjadi sebelum b dikenal sebagai properti kausalitas

- Suatu jam vector bagi sistem dengan **N** proses adalah suatu aarray beranggotakan **N** integer

- Setiap proses $\mathbf{P_i}$ menyimpan jam vektornya sendiri $\mathbf{VC_i}$
  - Nilai waktu Lamport untuk peristiwa-peristiwa disimpan dalam $\mathbf{VC_i}$
  - $\mathbf{VC_i(a)}$ diberikan ke suatu peristiwa **a**

- Jika $\mathbf{VC_i(a) < VC_i(b)}$, maka kita dapat menyimpulkan bahwa **a→b** (atau lebih pasnya, bahwa kejadian **a** *causally* mendahului kejadian **b**)

Update pada Jam Vektor

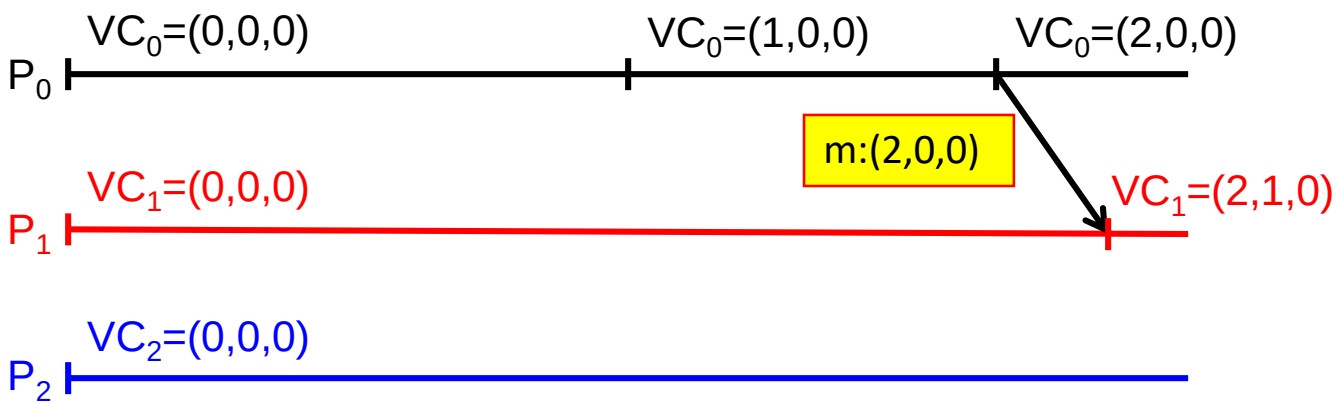
- Jam vektor dikonstruksi sebagai berikut:

1. $VC_i[i]$ adalah nomor peristiwa yang telah terjadi pada proses P_i sejauh ini
   - $VC_i[i]$ adalah jam logis lokal pada proses P_i

2. Jika $VC_i[j] = k$, maka P_i mengetahui bahwa k peristiwa telah terjadi pada P_j
   - $VC_i[j]$ adalah pengetahuan P_i mengenai wakti lokal pada P_j

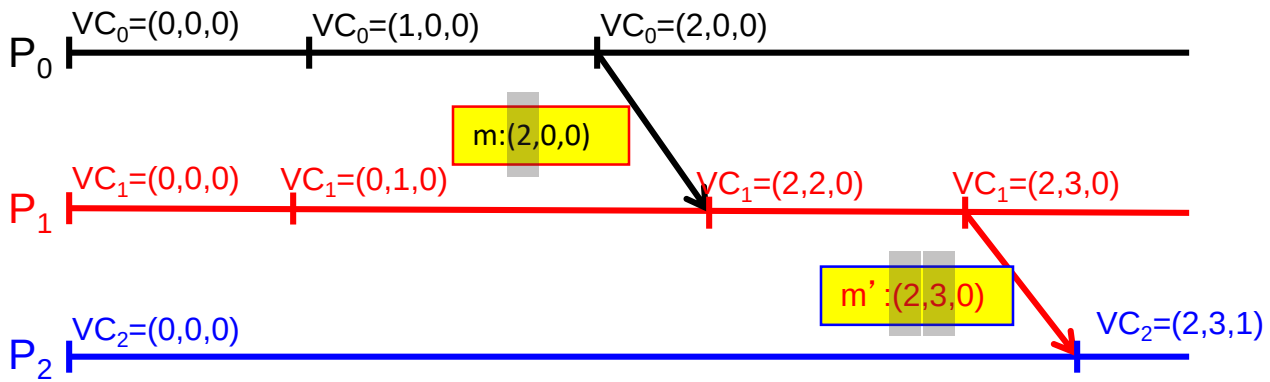
Algoritma Update Jam Vektor

- Kapanpun ada suatu peristiwa baru pada $\mathbf{P_i}$, naikkan $\mathbf{VC_i[i]}$
- Ketika suatu proses $\mathbf{P_i}$ mengirimkan message **m** ke $\mathbf{P_j}$:
  - Naikkan $\mathbf{VC_i[i]}$
  - Set timestamp dari **m**, **ts(m)** ke vector $\mathbf{VC_i}$
- Ketika message m diterima proses $\mathbf{P_j}$:
  - $\mathbf{VC_j[k] = max(VC_j[k], ts(m)[k])}$; (untuk semua **k**)
  - Naikkan $\mathbf{VC_j[j]}$



Menduga Kejadian dengan Jam Vektor

- Misalkan suatu proses P_i mengirimkan suatu message **m** ke P_j dengan timestamp **ts(m)**, maka:
  - P_j mengetahui nomor peristiwa pada sender P_i yang causally mendahului **m**
    - **(ts(m)[i] – 1)** menunjukkan nomor dari peristiwa pada P_i
  - P_j juga mengetahui nomor minimum dari peristiwa pada proses lain P_k yang causally mendahului **m**
    - **(ts(m)[k] – 1)** menunjukkan nomor minimum dari peristiwa pada P_k

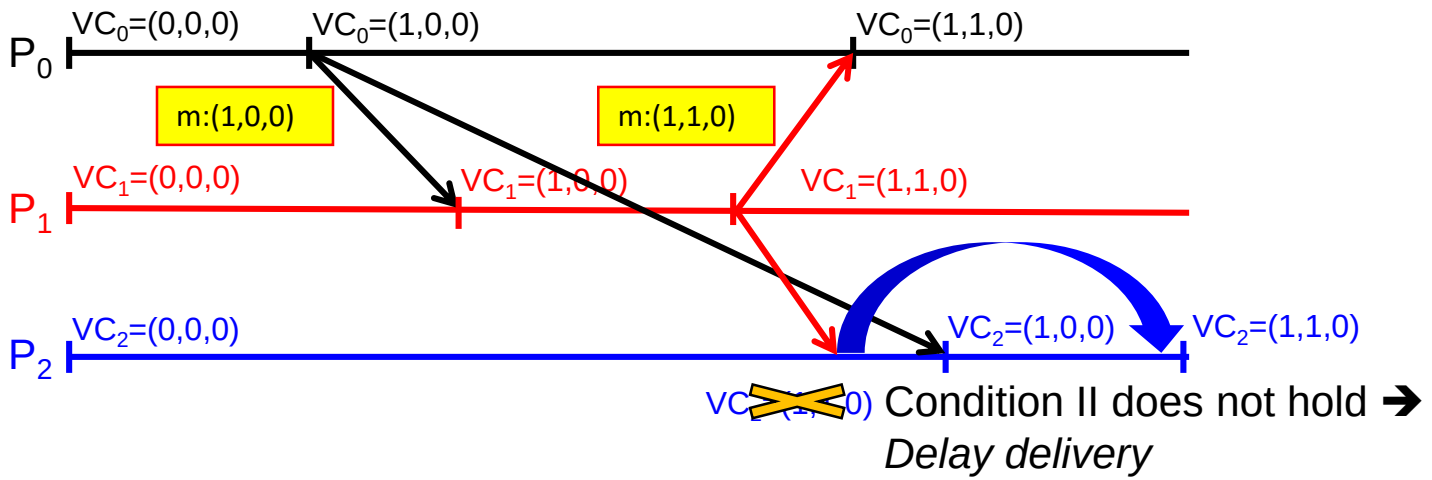


Pemaksaan Komunikasi Sebab-Akibat

- Anggaplah bahwa messages bersifat *multicast* di dalam se kelompok proses, P_0, P_1 dan P_2

- Untuk menegakkan *causally-ordered multicasting*, penghantaran suatu message ***m*** dikirim dari P_i ke P_j dapat ditunda sampai dua kondisi berikut dipenuhi:
  - $ts(m)[i] = VC_j[i] + 1$ (**Kondisi I**)
  - $ts(m)[k] <= VC_j[k]$ untuk semua k != i (**Kondisi II**)

Asumsikan bahwa P_i hanya menaikkan $VC_i[i]$ saat pengiriman m dan menyesuaikan $VC_i[k]$ ke $\max\{VC_i[k], ts(m)[k]\}$ untuk setiap k saat penerimaan suatu message m'



Jam Logis: Rangkuman

- Jam logis digunakan ketika proses harus menyetujui urutan peristiwa relatif, tetapi tidak harus waktu kejadian aktual

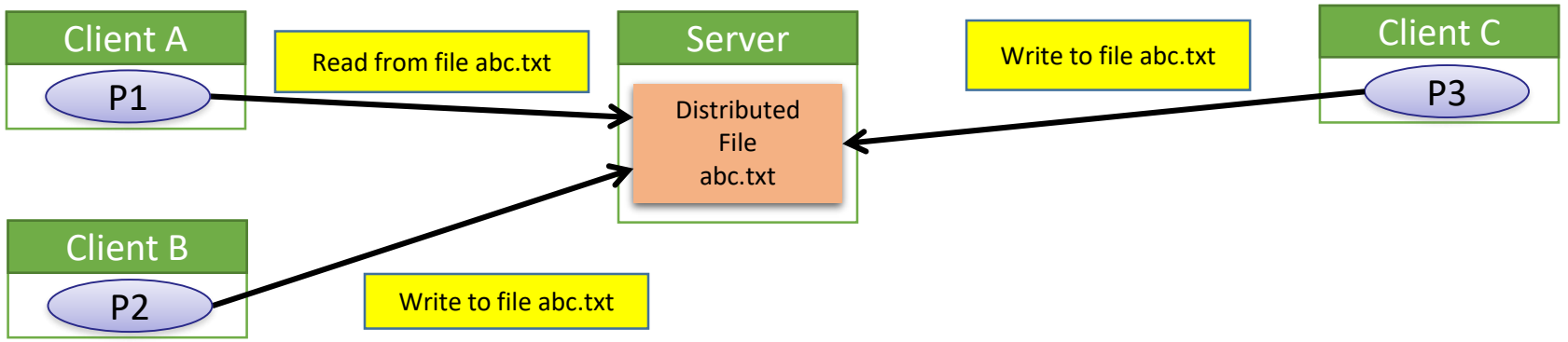
- Dua jenis jam logis:
  - Jam logis Lamport
    - Mendukung urutan relatif peristiwa di berbagai proses dengan menggunakan hubungan “terjadi sebelum”
  - Jam Vektor
    - Mendukung pengurutan kejadian secara kausal

Ikhtisar

- Sinkronisasi Waktu
  - Sinkronisasi Jam
  - Sinkronisasi Jam Logis
- Mutual Exclusion
- Algoritma Pemilihan

Perlunya Mutual Exclusion

- Proses terdistribusi perlu berkoordinasi untuk mengakses sumber daya bersama
- Contoh: Menulis file dalam Sistem File Terdistribusi



Dalam sistem uniprocessor, mutual exclusion untuk sumber daya bersama disediakan melalui variabel bersama atau dukungan sistem operasi

Namun, dukungan seperti itu tidak cukup untuk memungkinkan mutual exclusion dari entitas terdistribusi

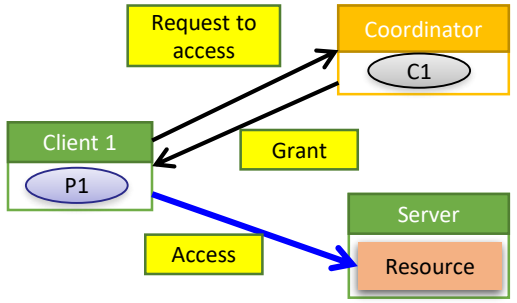
Dalam sistem terdistribusi, proses mengoordinasikan akses ke sumber daya bersama dengan mengirimkan pesan untuk menegakkan saling pengecualian (*mutual* exclusion) terdistrbusi

Jenis *Distributed Mutual Exclusion*

- Algoritma *mutual exclusion* diketegorikan ke dalam kelas:

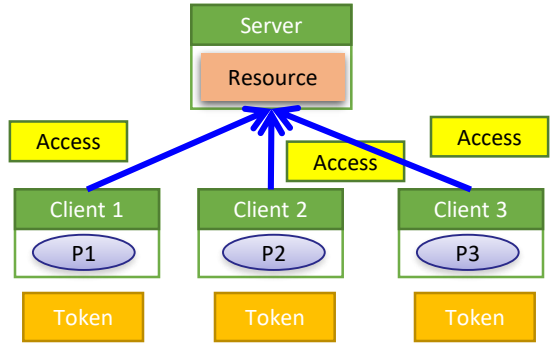
1. Pendekatan berbasis Permisi

   - Suatu proses, yang ingin mengakses sumber daya bersama, meminta ijin dari satu atau lebih koordinator



2. Pendekatan berbasis Token

   - Setiap sumber daya bersama mempunyai suatu token
   - Token digilirkan antara semua proses
   - Suatu proses dapat mengakses sumber daya tersebut jika ia mempunyai tokennya.



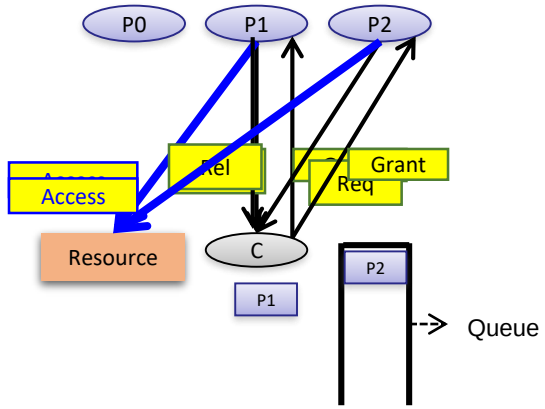
Pendekatan Berbasis Permisi

- Ada dua tipe algoritma mutual exclusion berbasis permisi
  1. Algortima terpusat (*centralized*)
  2. Algoritma tersebar (*decentralized*)

- Akan dipelajari contoh dari setiap jenis dari algoritma tersebut

I. Algoritma Tersentral

- Satu proses **terpilih** sebagai koordinator (**C**) atas sumber daya yang dishare
- Koordinator memelihara suatu **Antrian** permintaan akses
- Kapanpun suatu proses ingin mengakses sumber daya tersebut, ia mengirimkan suatu request message ke Koordinator untuk mengakses sumber daya tersebut
- Saat koordinator menerima request tersebut:
  - Jika tidak ada proses lain yangs edang mengakses sumber daya tersebut, ia memberikan ijin kepada proses itu dengan mengirimkan message “grant”
  - Jika proses lain sedang mengakses sumber daya itu, Koordinator mengantrikan request tersebut, dan tidak membalas request tersebut.
- Proses yang sedang *in action* melepaskan akses eksklusifnya setelah mengakses sumber daya
- Kemudian, Koordinator mengirimkan message “grant” kepada proses berikutnya dalam antrian tersebut.



I. Algoritma Tersentral

(+) Fleksibilitas: Blocking versus non-blocking requests

- Koordinator dapat mem-*block* proses yang meminta sampai sumber dayanya bebas
- Atau, Koordinator dapat mengirimkan suatu message "permission-denied" balik ke proses
  - Proses dapat poll Koordinator pada waktu kemudian
  - Atau, Koordinator mengantrikan request (tanpa memblokir requestor). Begitu sumber daya dilepas, Koordinator akan mengirimkan message "grant" ekslisit kepada proses

(+) Simplisitas: algoritma menjamin mutual exclusion dan mudah diimplementasikan

(-) Kekurangan Fault-Tolerance

- ***Centralized algorithm*** rentan terhadap ***single-point of failure*** (pada Koordinator)
  - Proses-proses tidak dapat membedakan antara koordinator mati dan *request blocking*

(-) Kemacetan Kinerja

- Dalam sistem skala besar, koordinator tunggal dapat kewalahan dengan *requests*

II. Algoritma Terdesentral

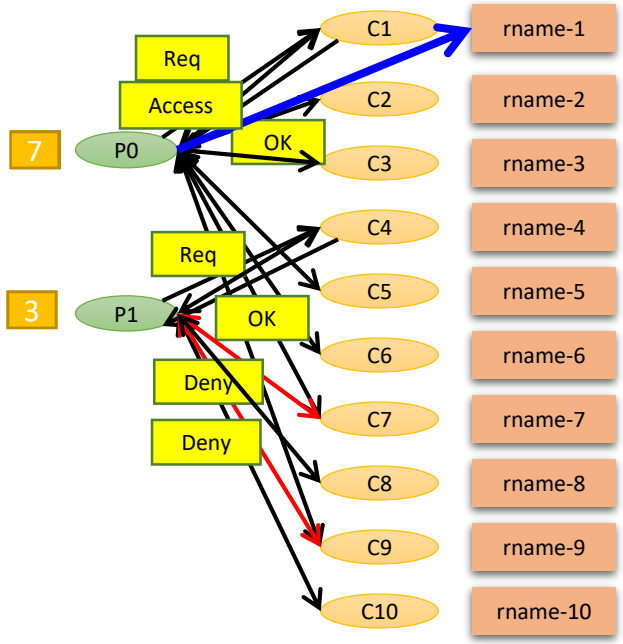
- Untuk menghindari kelemahan dari algoritma terpusat, Lin et al. (2005) menganjurkan algoritma eksklusi mutual yang terdesentralisasi

- Asumsi:
  - Proses terdistribusi berada dalam sistem berbasis Tabel Hash Terdistribusi (DHT)
  - Setiap sumber daya direplikasi n kali
    - Replika ke-i dari suatu sumber daya **`rname`** dinamakan sebagai **`rname-i`**
  - Setiap replika memiliki koordinator sendiri untuk mengendalikan akses
    - Koordinator untuk rname-i ditentukan dengan menggunakan fungsi hash

- Pendekatan:
  - Setiap kali suatu proses ingin mengakses sumber daya, ia harus mendapatkan suara mayoritas dari m **`> n/2`** koordinator
  - Jika koordinator tidak ingin memilih proses (karena sudah memilih proses lain), koordinator akan mengirim pesan "izin ditolak" ke proses tersebut

Algoritma Terdesentral: Contoh

- Jika **n=10** dan **m=7**, maka suatu proses memerlukan setidaknya **7** suara (vote) untuk mengakses sumber daya tersebut.



rname-x = replika ke-x odari sumber daya rname

Cj = Koordinator j

Pi = Proses i

n = Jumlah suara yang diperoleh

Fault-Tolerance Dalam Algoritma Desentralisasi

- Algoritma desentralisasi ini mengasumsikan bahwa koordinator pulih dengan cepat dari kegagalan

- Namun, koordinator akan mengatur ulang statusnya setelah pemulihan
  - Koordinator bisa saja melupakan suara yang telah diberikan sebelumnya

- Karenanya, koordinator dapat secara tidak benar memberikan izin untuk suatu proses
  - Mutual exclusion tidak dapat dijamin secara deterministik
  - Namun, algoritmanya masih secara probabilistik menjamin pengecualian bersama.

Jaminan Probabilistik dalam Algoritma Terdesentral

- Berapa jumlah minimum koordinator yang harus gagal untuk melanggar pengecualian bersama?
  - Setidaknya n-m + 1 koordinator harus gagal
- Misalnya peluang melanggar mutual exclusion berupa P_v
  - Derivasi dari P_v
    - Katakan T waktu hidup dari koordinator
    - Katakan $p=\Delta t/T$ peluang yang suatu koordinator crash selama interval waktu Δt
    - Katakan $P[k]$ peluang bahwa k keluar dari m koordinator crash selama interval yang sama

$$P[k] = \binom{m}{k} p^k (1-p)^{m-k}$$

    - Peluang pelanggaran mutual exclusion P_v dapat dihitung sebagai: $P_v = \sum_{k=2m-n}^{n} P[k]$

- Prakteknya, peluang ini biasanya sangat kecil
  - Untuk T=3 jam, Δt=10 s, n=32, dan m=0.75n : $P_v = 10^{-40}$

Protokol Berbasis Quorum

- Algoritma ini merupakan implementasi dari protokol yang lebih umum yang dikenal sebagai protokol berbasis kuorum

- Protokol berbasis kuorum dapat diimplementasikan menggunakan skema pemungutan suara, awalnya diusulkan oleh Thomas (1979) kemudian digeneralisasi oleh Gifford (1979)

- Gagasan dasar:
  - Klien diharuskan untuk meminta dan mendapatkan izin dari beberapa server sebelum membaca atau menulis dari atau ke item data yang direplikasi
    - Aturan membaca dan menulis harus ditetapkan
    - Setiap replika diberi nomor versi, yang dinaikkan pada setiap penulisan

Protokol Berbasis Kuorum

- Contoh Kerja:
  - Pertimbangkan sistem file terdistribusi dan misalkan file direplikasi di N server

  - Aturan Tulis (Write):
    - Klien harus terlebih dahulu menghubungi N / 2 + 1 server (mayoritas) sebelum memperbarui file
    - Setelah suara mayoritas diperoleh, file diperbarui dan nomor versinya bertambah
      - Ini dilakukan di semua situs replika

Protokol Berbasis Quorum

- Contoh Kerja:
  - Pertimbangkan sistem file terdistribusi dan misalkan file direplikasi di N server

  - Aturan baca (Read):
    - Klien harus menghubungi N / 2 + 1 server, meminta mereka untuk mengirim nomor versi file yang diminta
    - Jika semua nomor versi sama, ini harus merupakan versi terbaru dari file
      - Ini karena upaya untuk memperbarui server yang tersisa akan gagal karena jumlahnya tidak cukup
      - E.g., jika N = 5 dan klien menerima 3 nomor versi yang semuanya sama dengan 8, mustahil 2 server yang tersisa akan memiliki versi 9
        - Setiap pembaruan yang berhasil dari versi 8 ke versi 9 mengharuskan mendapatkan 3 server untuk menyetujuinya, bukan hanya 2

Protokol Berbasis Quorum

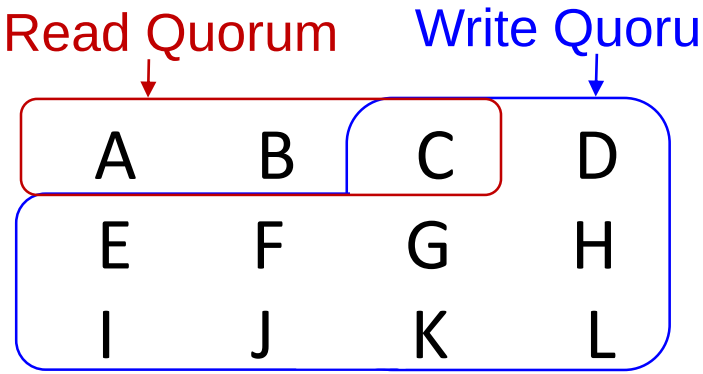
- Skema Gifford menggeneralisasi skema Thomas
- **Skema Gifford**:
  - Aturan Read:
    - Suatu client perlu mengumpulkan *read quorum*, yang merupakan koleksi sembarang dari N_R server, atau lebih
  - Aturan Write:
    - Untuk memodifikasi suatu file, suatu *write quorum* dari setidaknya N_W servers diwajibkan.

Protokol Berbasis Quorum

- Nilai dari N_R dan N_W merupakan subyek bagi dua Batasan berikut:
  - Batasan 1 (atau C1): $N_R + N_W > N$
  - Batasan 2 (atauC2): $N_W > N/2$

- Tututan:
  - C1 mencegah konflik read-write (RW)
  - C2 mencegah konflik write-write (WW)

Contoh 1



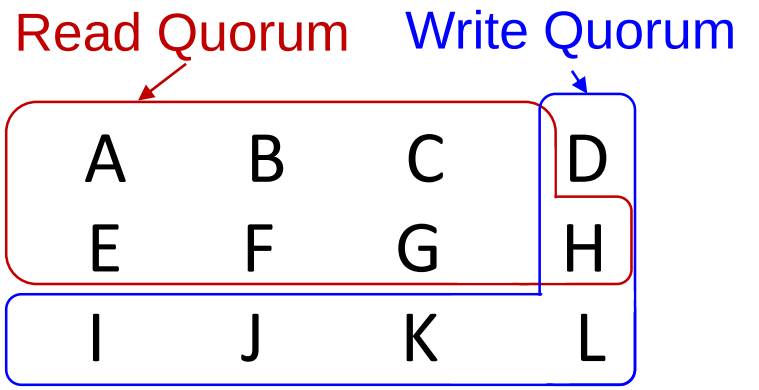
$N_R = 3$ and $N_W = 10$

C1: $N_R + N_W = 13 > N = 12$
➔ No RW conflicts

C2: $N_W > 12/2 = 6$
➔ No WW conflicts

- Kuorum tulis terbaru terdiri dari server {C, D,…, L}
  - Server-server ini mendapatkan nilai dan nomor versi baru
- Setiap kuorum baca berikutnya harus mengandung setidaknya 1 anggota dalam kuorum tulis {C, D, …, L}
  - Ketika klien melihat versi anggota ini, ia akan melihat bahwa ia memiliki nomor versi tertinggi, karenanya, ia akan membawanya

Contoh 2



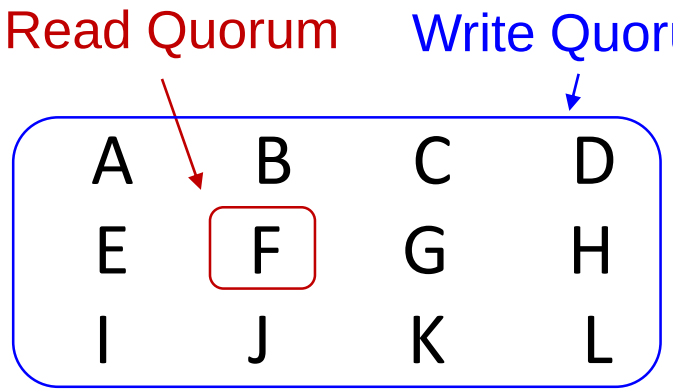
$N_R = 7$ and $N_W = 6$

C1: $N_R + N_W = 13 > N = 12$
➔ Tidak ada konflik RW

C2: $N_W \nmid 12/2 = 6$
➔ Konflik WW dapat muncul

- Mengapa melanggar C2 menyebabkan konflik WW?
  - Jika satu klien memilih {A, B, C, E, F, G} sebagai write set
  - Dan klien lain memilih {D, H, I, J, K, L} sebagai write set
  - Kedua pembaruan akan diterima tanpa mendeteksi bahwa mereka benar-benar bertentangan, sehingga mengarah pada pandangan yang tidak konsisten!

Contoh 3



$N_R = 1$ dan $N_W = 12$

C1: $N_R + N_W = 13 > N = 12$
➔ Tidak ada konflik RW

C2: $N_W > 12/2 = 6$
➔ Tidak ada konflik WW

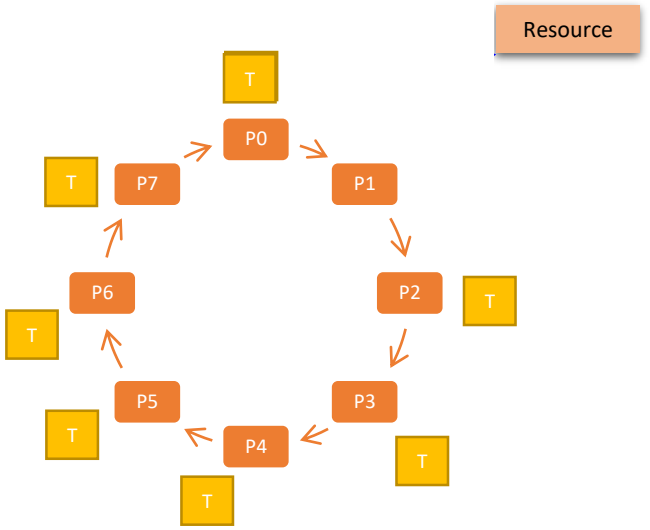
- Suatu klien dapat membaca file yang direplikasi dengan menemukan salinan apa pun
  - Performa baca yang bagus!
- Suatu klien perlu mendapatkan kuorum tulis pada semua salinan
  - Performa menulis lambat!
- Contoh ini menunjukkan skema yang umumnya disebut sebagai ROWA (*Read-Once, Write-All*)

Ikhtisar

- Sinkronisasi Waktu
  - Sinkronisasi Jam
  - Sinkronisasi Jam Logis
- Mutual Exclusion (Pengecualian Bersama)
  - Pendekatan berbasis Permisi (ijin)
  - Pendekatan berbasis Token
- Algoritma Pemilihan.

Algoritma Token Ring

- Dengan suatu algoritma token ring:
  - Setiap sumber daya dikaitkan dengan token
  - Token diedarkan di antara proses
  - Proses dengan token dapat mengakses sumber daya
- Bagaimana token diedarkan di antara proses?
  - Semua proses membentuk cincin logis di mana setiap proses mengetahui proses selanjutnya
  - Satu proses diberikan token untuk mengakses sumber daya
  - Proses dengan token memiliki hak untuk mengakses sumber daya
  - Jika proses telah selesai mengakses sumber daya ATAU tidak ingin mengakses sumber daya:
    - Ia melewati token ke proses selanjutnya di dalam ring



Diskusi Mengenai Token Ring

- Pendekatan token ring menyediakan saling pengecualian (mutex) deterministik
  - Ada satu token, dan sumber daya tidak dapat diakses tanpa token
- Pendekatan token ring menghindari kelaparan
  - Setiap proses akan menerima token
- Token ring memiliki overhead pesan yang tinggi
  - Ketika tidak ada proses yang membutuhkan sumber daya, token bersirkulasi dengan kecepatan tinggi
- Jika token hilang, itu harus dibuat kembali
  - Mendeteksi kehilangan token itu sulit (terutama jika jumlah waktu antara penampilan token tidak dibatasi)
- Proses yang mati harus dibersihkan dari cincin
  - Pengiriman token berbasis ACK dapat membantu dalam membersihkan proses mati.

Perbandingan Algoritma Mutual Exclusion

Algoritma	Keterlambatan sebelum suatu proses dapat mengakses sumber daya (dalam waktu pesan)	Jumlah pesan yang diperlukan bagi suatu proses untuk mengakses dan melepaskan sumber daya bersama	Masalah
Centralized	2	3	• Koordinator crash
Decentralized	2mk	2mk + m; k=1,2,…	• Jumlah message besar
Token Ring	0 to (n-1)	1 to n	• Token dapat hilang • Ring dapat berhenti ke yg eksisi karena proses crash

- Anggaplah bahwa:

  n = Jumlah proses dalam sistem terdistribusi

  Untuk algoritma Desentralisasi :

  m = jumlah minimum koordinator yang harus menyetujui proses untuk mengakses sumber daya

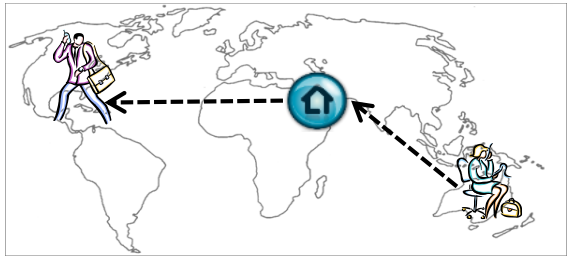
  k = jumlah rata-rata permintaan yang dibuat oleh proses ke koordinator untuk meminta pemungutan suara.

Ikhtisar

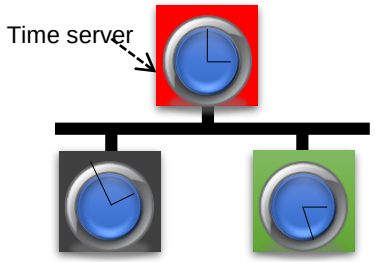
- Sinkronisasi Waktu
  - Sinkronisasi Jam
  - Sinkronisasi Jam Logis
- Mutual Exclusion (Pengecualian Bersama)
  - Pendekatan berbasis Permisi (ijin)
  - Pendekatan berbasis Token
- Algoritma Pemilihan.

Pemilihan dalam Sistem Terdistribusi

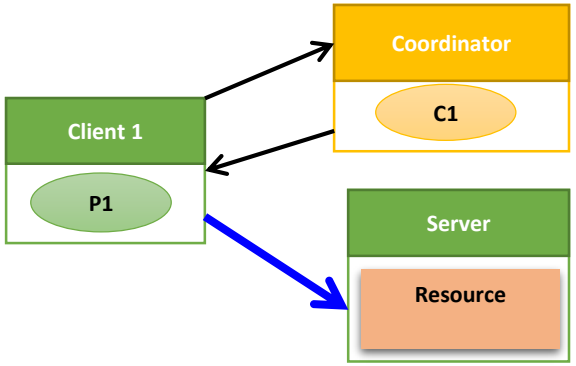
- Banyak algoritma terdistribusi memerlukan satu proses untuk bertindak sebagai koordinator
  - Biasanya, tidak masalah proses mana yang dipilih sebagai koordinator



Pemilihan Home Node dalam Naming



Algoritma Sinkronisasi Jam Berkeley



Suatu Algoritma Mutual Exclusion terpusat

Proses Pemilihan Singkatnya

- Kita anggap bahwa ada proses P_i yang dapat memulai algoritma pemilihan untuk memilih koordinator baru
- Pada akhir algoritma pemilihan, koordinator terpilih harus unik
- Setiap proses dapat mengetahui ID proses dari setiap proses lainnya, tetapi tidak tahu proses mana yang macet
- Secara umum, kita mensyaratkan bahwa koordinator adalah proses dengan ID proses terbesar
  - Idenya dapat diperluas untuk memilih koordinator terbaik
    - Contoh: Pemilihan koordinator dengan beban komputasi paling sedikit
      - Jika beban komputasi dari proses P_i ditandai dengan $load_i$, maka koordinator akan berupa proses dengan $1/load_i$ paling tinggi. Ikatan rusak dengan mengurutkan proses ID.

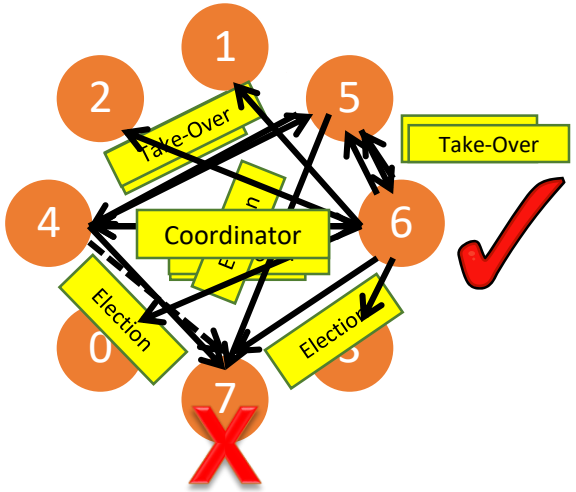
Algoritma Pemilihan

- Mari kita pelajari dua algoritma pemilihan:
  1. Bully Algorithm
  2. Ring Algorithm

1. Algoritma Bully

- Suatu proses (katakanlah $\mathbf{P_i}$) memulai algoritma pemilihan ketika pemberitahuan bahwa koordinator yang ada tidak merespons

- Proses $\mathbf{P_i}$ menyerukan pemilihan sebagai berikut:
  1. $\mathbf{P_i}$ mengirimkan suatu pesan “Election” kepada semua proses dengan ID proses yang lebih tinggi
  2. Ketika proses $\mathbf{P_j}$ dengan **j>i** menerima pesan tersebut, ia merespon dengan suatu pesan “Take-over”. $\mathbf{P_i}$ Tidak ada lagi kontes dalam pemilihan
     - Proses $\mathbf{P_j}$ memulai kembali panggilan lain untuk pemilihan. Langkah 1 dan 2 berlanjut
  3. Jika tidak ada respons, $\mathbf{P_i}$ memenankan pemilihan. $\mathbf{P_i}$ mengirimkan pesan “Koordinator” ke setiap proses



2. Algoritma Ring

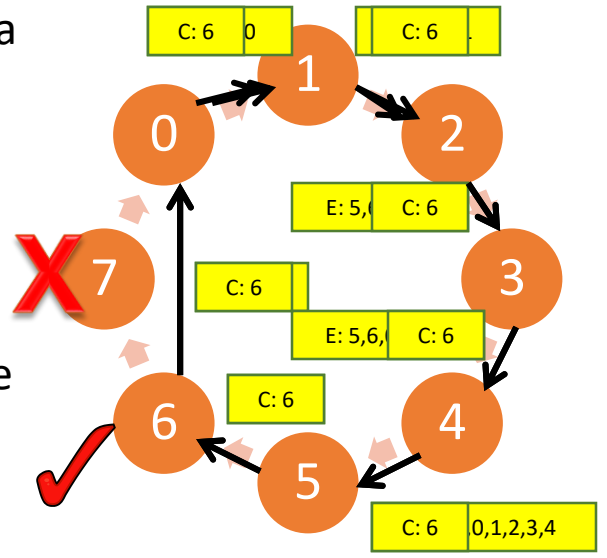
- Algoritma ini umumnya digunakan dalam topologi ring

- Ketika proses P_i mendeteksi bahwa koordinator telah mogok, ia memulai algoritma pemilihan

1. P_i membangun suatu pesan “Election” **(E)** , dan mengirimkannya ke node berikutnya. Ia memasukkan ID-nya ke dalam pesan Pemilihan

2. Saat proses P_j menerima pesan tersebut, ia menambahkan ID-nya dan meneruskan pesan itu
   i. Jika node berikutkan mengalami crash, P_j mencarikan node hidup berikutnya.

3. Ketika pesan kembali ke P_i:
   i. P_i memilih proses dengan ID tertinggi sebagai koordinator
   ii. P_i mengubah jenis pesan menjadi pesan "Koordinasi" (C) dan memicu peredarannya di dalam ring



Perbandingan Algoritma Pemilihan

Algorit,a	Jumlah Pesan untuk Memilih Koordinator	Masalah
Bully Algorithm	$O(n^2)$	• Biaya pesan besar
Ring Algorithm	2n	• Diperlukan topologi cincin hamparan (*overlay*)

- Anggaplah bahwa:
  n = Jumlah proses dalam sistem terdistribusi

Ringkasan Algoritma Pemilihan

- Algoritma pemilihan digunakan untuk memilih proses unik yang akan mengoordinasikan kegiatan tertentu

- Pada akhir algoritma pemilihan, semua node harus secara unik mengidentifikasi koordinator

- Kita mempelajari dua algoritma untuk melakukan pemilihan:
  - Bully algorithm
    - Proses berkomunikasi secara terdistribusi untuk memilih koordinator
  - Ring algorithm
    - Proses dalam topologi ring mengedarkan pesan pemilihan untuk memilih koordinator

Selanjutnya...

- *Message Passing Interface* (MPI)