Algoritma Co-ordination

Algoritma Co-ordination merupakan dasar dalam sistem terdistribusi:
• untuk resource sharing: concurrent updates terhadap
  – records dalam suatu database (record locking)
  – files (file locks dalam stateless file servers)
  – shared bulletin board
• untuk persetujuan pada actions: apakah terhadap proses:
  – commit/abort transaksi database
  – persetujuan pada suatu pembacaan dari group sensors
• untuk secara dinamis re-assign peran master
  – memilih primary time server setelah crash
  – memilih co-ordinator setelah network di-reconfigurasi

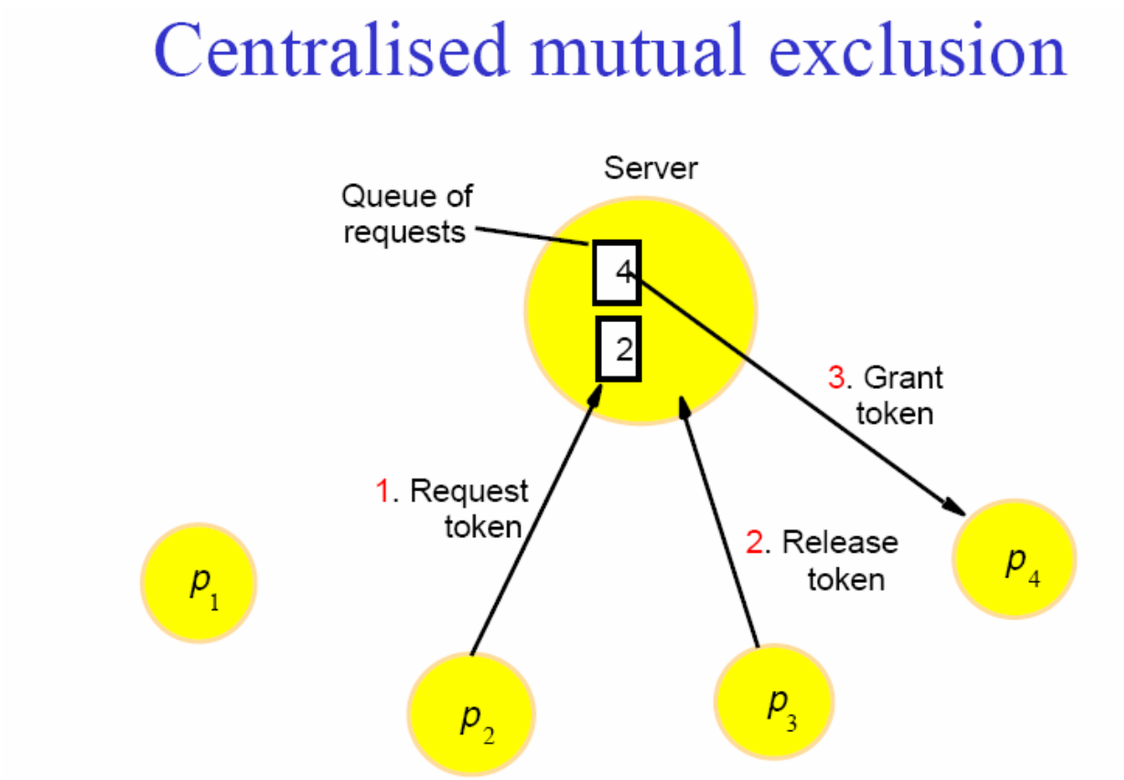
Kesulitan yang akan dijumpai
• Tidak sesuai untuk solusi terpusat
– communications bottleneck
• Tidak sesuai untuk Fixed master-slave arrangements
– process crashes
• Topologi jaringan yang bervariasi
– ring, tree, arbitrary; connectivity problems
• Tolerasi Failures jika dimungkinkan
– link failures
– process crashes
• Impossibility results

• Mutual exclusion
– bentuk terdisitribusi untuk masalah critical section
– harus menggunakan message passing
• Leader elections
– setelah terjadi crash failure muncul
– setelah network direconfigurasi
• Consensus (juga disebut Agreement):
– similar dengan sertangan yang terkoordinasi
– beberapa berbasis pada multicast communication
– variasi yang bergantung pada tipe kesalahan, network, dll

Asumsi Failure
Diasumsikan reliable links, tetapi dimungkinkan terjadinya process crashes.
• Layanan pendeteksian Failure
– query jika suatu proses telah failed
– caranya?
• processes mengirim pesan ‘P is here’ setiap T secs
• failure detector merekam replies
– unreliable, khsusunya pada sistem asynchronous
• Observasi failures:
– Suspected: tidak terdapat recent communication, tetapi terjadi kelambatan
– Unsuspected: tidak ada jaminan bahwa telah terjadi fail
– Failed: crash telah ditentukan

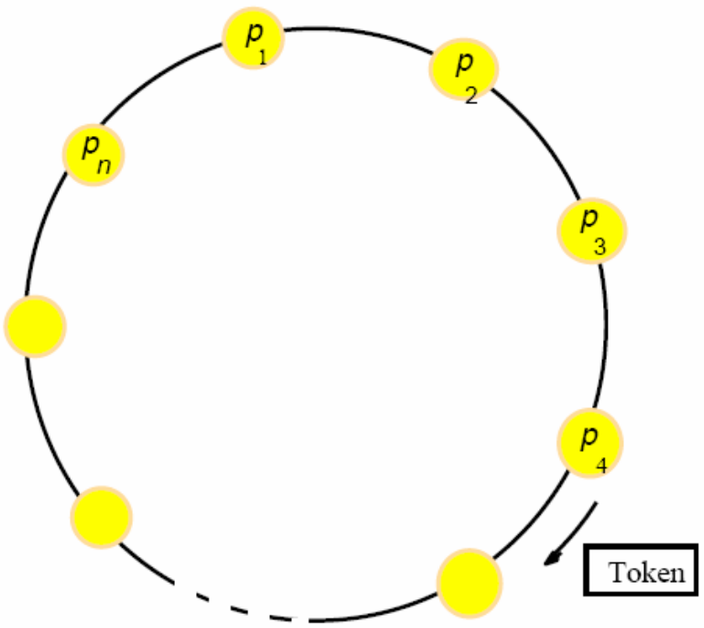
Distributed mutual exclusion
• Problem:
– N asynchronous processes, untuk sederhanyanya tidak terdapat failures
– Jaminan pengiriman message (reliable links)
– untuk mengeksekusi critical section (CS), setiap proses memanggil:
• **enter**()
• **resourceAccess**()
• **exit**()

• Kebutuhan
(**MC1**) Paling banyak terdapat satu proses dalam CS pada saat yang sama.
(**MC2**) Requests untuk **enter** dan **exit** diberikan.
(**MC3** - Optional, stronger) Requests untuk **enter** diberikan berdasarkan urutan casualitas.



Centralised service?
• Single server mengimplementasikan imaginary token:
– hanya process yang memegang token dapat berada di CS
– server menerima **request dari** token
– me-reply **grant**ing access jika CS free; selain itu, request diantri
– ketika suatu process melepas (**release**) token dan diberikan ke request yang paling lama di antrian

Ring-based algorithm



Arrange processes in a logical ring, let them pass token.

Ring-based algorithm
• No server bottleneck, no master
• Processes:
– secara kontinu mengirim token ke sekeliling ring, dalam satu arah
– jika tidak membutuhkan akses ke CS, maka mengirim tiken ke neighbour
– kalau membutuhkan akses ke CS maka menunggu token dan tetap memegang token apabila maish berada di CS
– ketika keluar maka mengirim token ke neighbour
• How it works
– secara terus menerus menggunakan bandwidth jaringan
– delay untuk masuk ke CS bergantung pada ukuran ring
– causality order requests tidak dipertimbangkan (MC3)

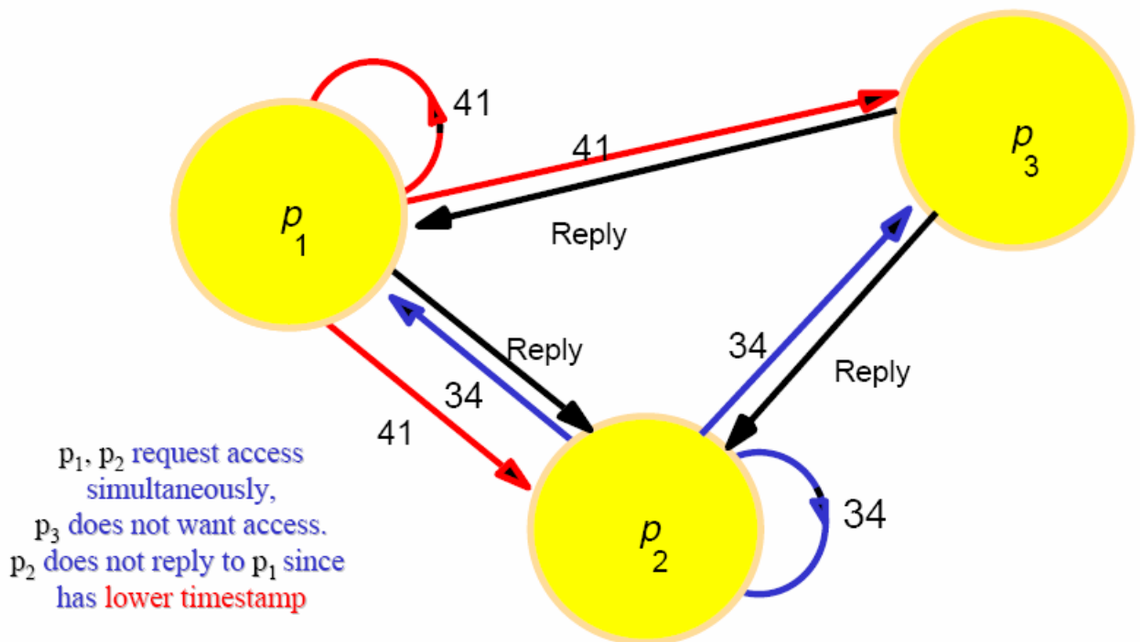
Ricart&Agrawala algorithm
• Berbasis pada multicast communication
– inter koneksi N proses yang asynchronous, dan masing-masing memiliki:
  • unique id
  • Lamport's logical clock
– processes multicast memohon untuk masuk (entry) ke CS:
  • menetapkan timestamped dengan Lamport's clock dan process id
– entry diberikan.
• ketika semua proses lainnya mereply
• simultaneous requests di-resolve dengan timestamp
• How it works
– memnuhi stronger property (MC3)
– jika pernagkat keras mendukung multicast, hanya satu message yang di-enter

Ricart&Agrawala algorithm

On initialization
state := RELEASED;
To ***enter the critical section***
state := WANTED;
Multicast *request* to all processes; } request processing deferred here
T := request's timestamp;
Wait until (number of replies received = (*N* – 1));
state := HELD;

On receipt of a request $<T_i, p_i>$ *at* p_j $(i \neq j)$
if (*state* = HELD) or ((*state* = WANTED) *and* $((T, p_j) < (T_i, p_i))$
then
queue *request* from p_i without replying;
else
reply immediately to p_i;
end if
To ***exit the critical section***
state := RELEASED;
reply to any queued requests;

Multicast mutual exclusion

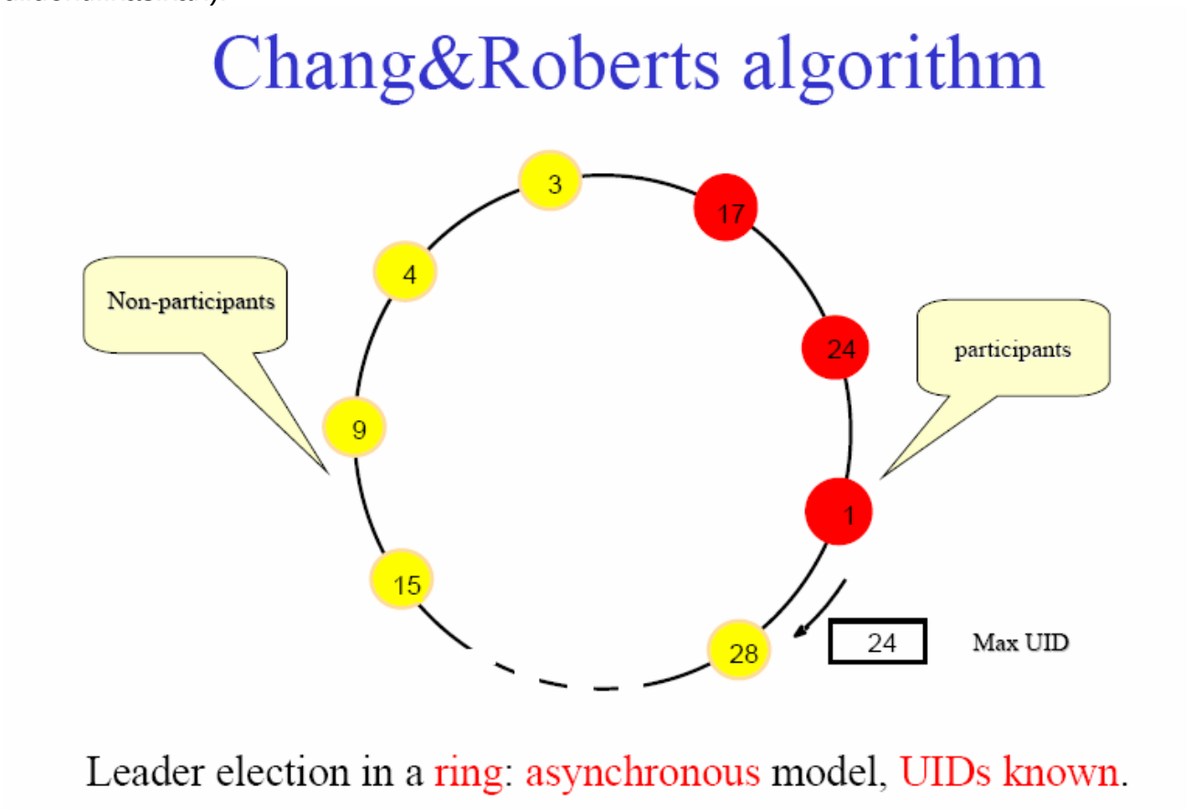


Kesimpulan Mutual exclusion
• Performance
– satu request-reply cukup untuk masuk
– relative banyak menggunakan bandwidth jaringan
– client delay bergantung pada frekuensi akses dan ukuran jaringan
• Fault tolerance
– biasanya mengasumsikan link yang handal
– beberapa dapat diadaptasikan untuk menerima crash
• Solusi Lainnya
– cukup untuk memperoleh persetujauan dari subset dari himpunan proses voting tertentu yang saling overlapping (Maekawa algorithm)

Leader election algorithms
• Masalah

– Terdapat N processes, yang dimungkinkan memiliki/tidak memiliki unique ids (UIDs)
– untuk sederhananya diasumsikan tidak terdapat crash
– harus memilih master coordinator yang unique diantara proses
– election dipanggil setalah tidak terdapat failure
– satu atau lebih proses dapat memanggil election secara simultan

• **Requirements**
(**LE1**) Setiap process mengetahui P, yang merupakan identitas leader, dimana P adalah proses id yang unique (biasanya nilai maksimum) atau belum terdefinisi.
(**LE2**) Semua proses berpartisipasi dan mencari identitas leader (yang tidak dapat tidak diidentifikasikan).



Algoritma Chang&Roberts
• Asumsi
– ring tak berarah, asynchronous, dan setiap proses memiliki UID

• Pemilihan (Election)
– setiap proses pada awalnya (initial) merupakan non-participant
– menentukan leader (melalui *election* message):
  • initiator menjadi participant dan mengirim UID yang dimilikinya ke neighbour
  • ketika non-participant menerima pesan *election*, kemudian mengirim maximum UID yang dimiliki dan yang diterima serta kemudian menjadi participant.
  • participant tidak mem-forward pesan *election*
– mengumumkan pemenang (*elected* message):
  • ketika participant menerima pesan *election* dengan UID-nya, maka kemudian menjadi leader dan non-participant, serta mem-forwards UID dalam *elected* message
  • selain itu, merekam UID leader, dan menjadi non-participant serta mem-forwards UID tersebut.

Algoritma Itai&Rodeh
• Asumsi
– terdapat N process, ring tak berarah, dan synchronous
– processes tidak memiliki UIDs
• Election
– setiap proses memilih ID secara random dari himpunan {1,..K}
• non-unique! Tetapi cepat
– process mengirim semua ID ke seluruh ring
– setelah one putaran, jika terdapat suatu ID yang uniqueif there exists a unique ID then elect maka dipilih maksismu unique ID
– jika tidak maka mengulangi proses

• Pertanyaan
– bagaimana loop berakhir ?