Sistem Terdistribusi

TIK-604

Replikasi

Kuliah 11: 06 s.d 08 Mei 2019

Husni

Hari ini...

- Bahasan Terakhir:
  - Caching (atau replikasi sisi client)
- Diskusi hari ini:
  - Replikasi (lebih tepatnya replikasi sisi server)
    - Model konsistensi *Data-Centric*
    - Model Konsistensi *Client-Centric*
    - Protokol Konsisten
- Pengumuman:
  - Pengumuman-pengumuman.

Ikhtisar

Kuliah Hari ini

- Motivasi
- Model-model Konsistensi
  - Model Konsistensi Data-Centric
  - Model Konsistensi Client-Centric
- Protokol-protocol Konsistensi.

Kuliah Berikutnya

Ikhtisar

- Motivasi

- Model-model Konsistensi
  - Model Konsistensi Data-Centric
  - Model Konsistensi Client-Centric

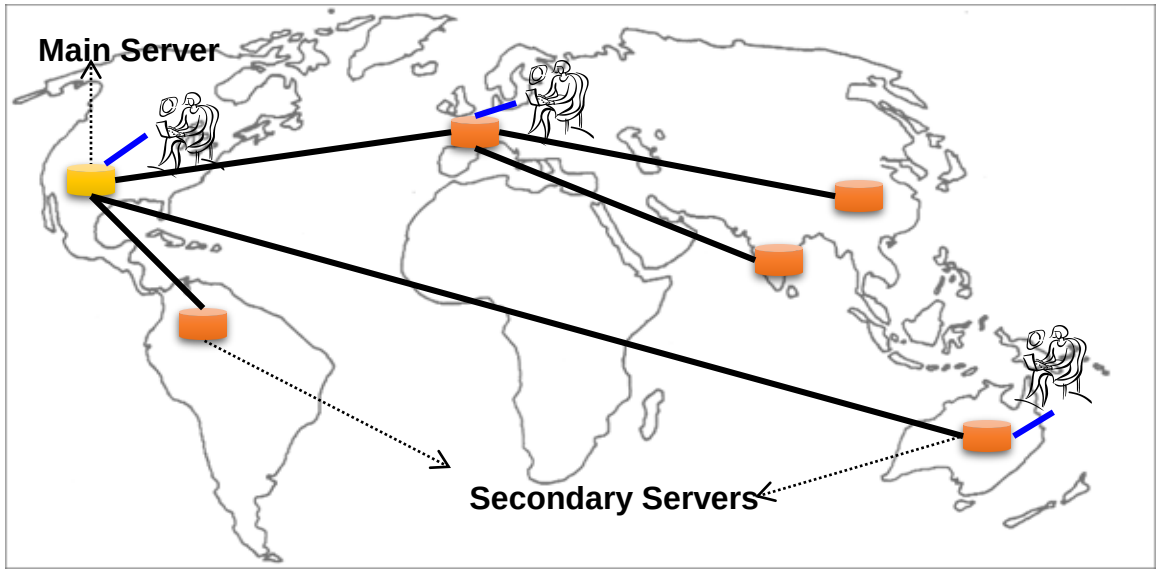
- Protokol-protocol Konsistensi.

Mengapa Replikasi?

- Replijasi penting untuk:
  1. Meningkatkan kinerja (*performance*)
     - Client dapat mengakses salinan replika yang dekat dan menghemat latensi (*latency*)
  2. Meningkatkan ketersediaan dari layanan (*availability*)
     - Replikasi dapat menutupi kegagalan seperti server crashes dan dikoneksi jaringan
  3. Meningkatkan skalabitas system (*scalability*)
     - Requests ke data dapat didistribusikan lintas banyak server, yang mengandung salinan replika dari data
  4. Mengamankan terhadap serangan jahat (*security*)
     - Bahkan jika beberapa replika merupakan *malicious*, keamanan dari data dapat dijamin dengan menyandarkan pada Salinan replikasi pada server yang *non-compromised* (tidak bersepakat jahat)

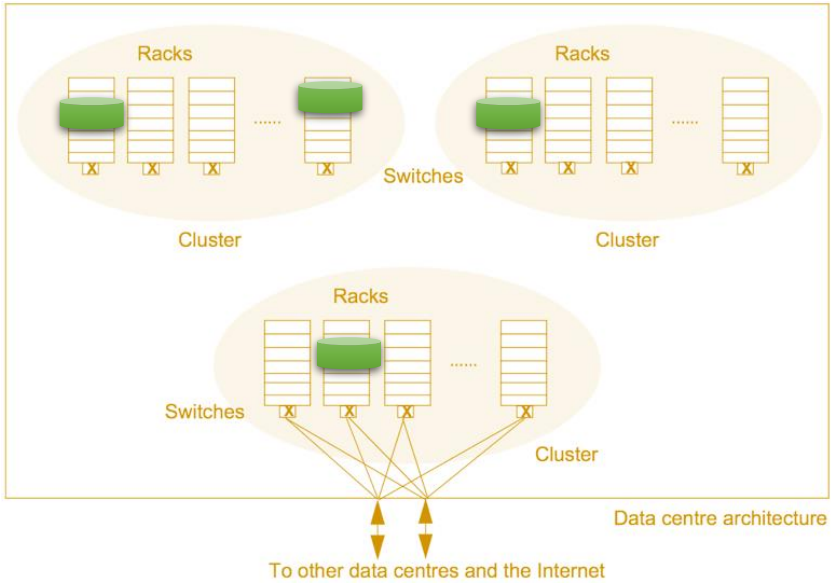
1. Replikasi Untuk Perbaikan Kinerja

- Contoh:
  - Replikasi pada server sekunder dalam *Content Delivery Network* (CDN)



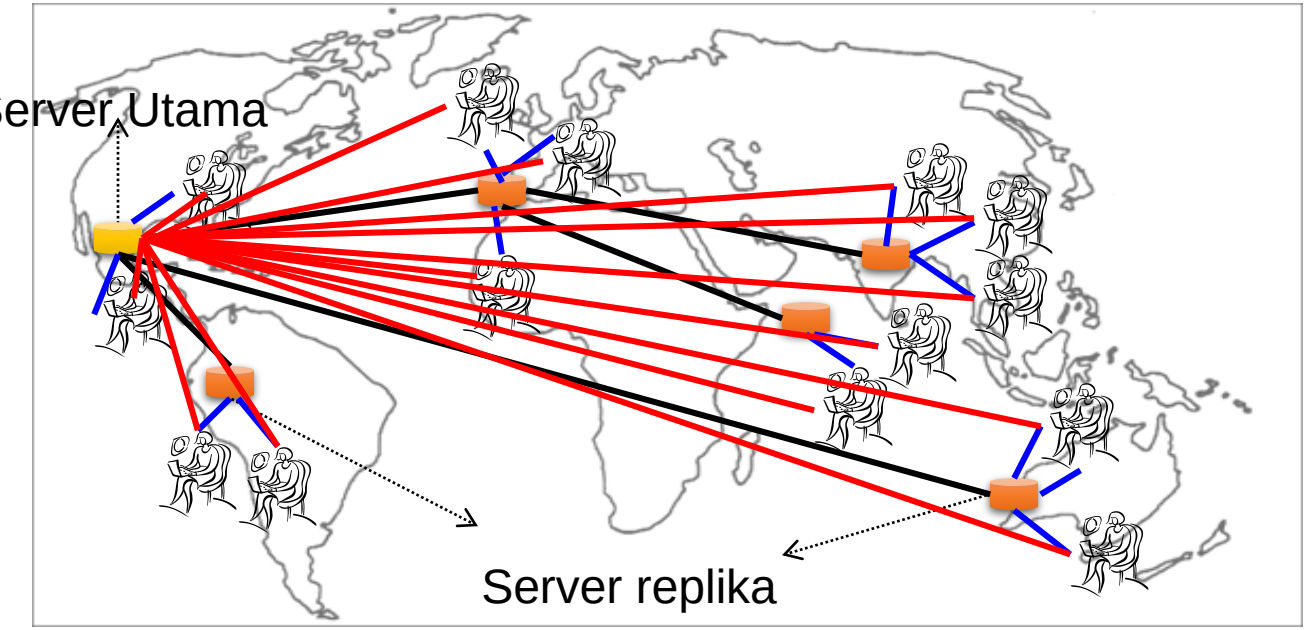
2. Replikasi Untuk High-Availability (HA)

- Contoh:
  - Google File-System mereplikasi blok-blok data pada komputer lintas rack, cluster dan data center
  - Jika satu komputer, rack atau cluster mengalami *crash*, block-block data masih dapat diakses dari sumber lainnya.



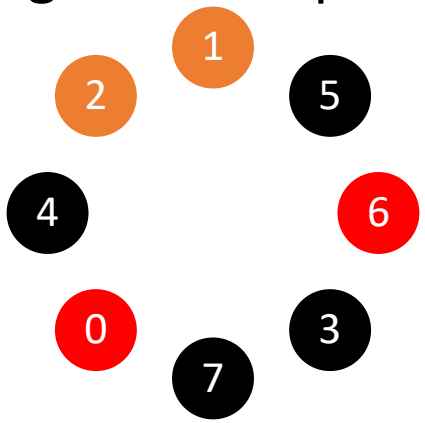
3. Replikasi untuk Meningkatkan Skalabilitas

- Mendistribusikan data lintas server replika membantu menjaga server utama dari menjadi suatu *performance bottleneck*
- Contoh: Content Delivery Networks dapat menurunkan beban pada server-server utama (primer)



4. Replikasi untuk Pengamanan Serangan Malicious

- Jika suatu minoritas dari server di dalam system bersifat *malicious*, server-server non-malicious dapat mengalahkan (dalam pemilihan) server yang sudah *malicious*
  - Teknik ini juga dapat digunakan untuk menyediakan *fault-tolerance* terhadap server non-malicious tetapi faulty (salah dalam bekerja)
- Contoh: Dalan sistem peer-to-peer (P2P), peers dapat berkoordinasi untuk mencegah penyampaian data yangh salah kepada requester



Jumlah server dengan data yang benar mengalahkan server yang salah

n = Server yang tidak mempunyai data yang direquest.

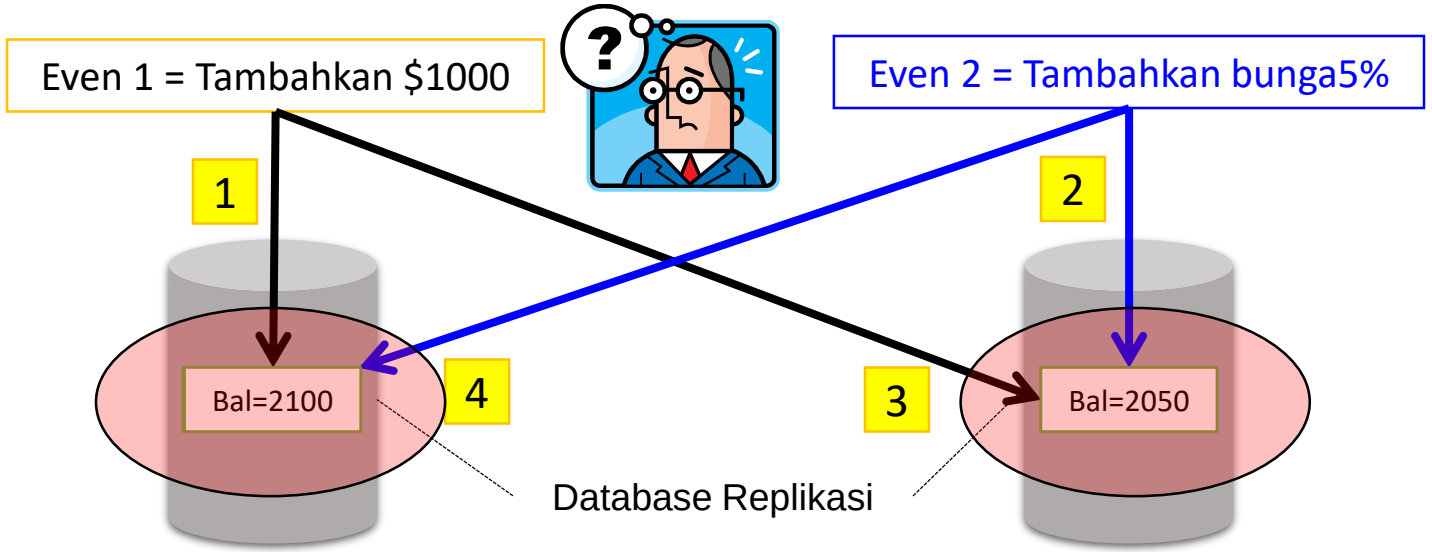
n = Server dgn correct data

n = Server dengan faulty data

Mengapa Konsistensi?

- Padahal replikasi (server-side) dating dengan biaya, yang memerlukan perawatan konsistensi (atau lebih tepatnya pengurutan konsisten dari update *consistent ordering of updates*)

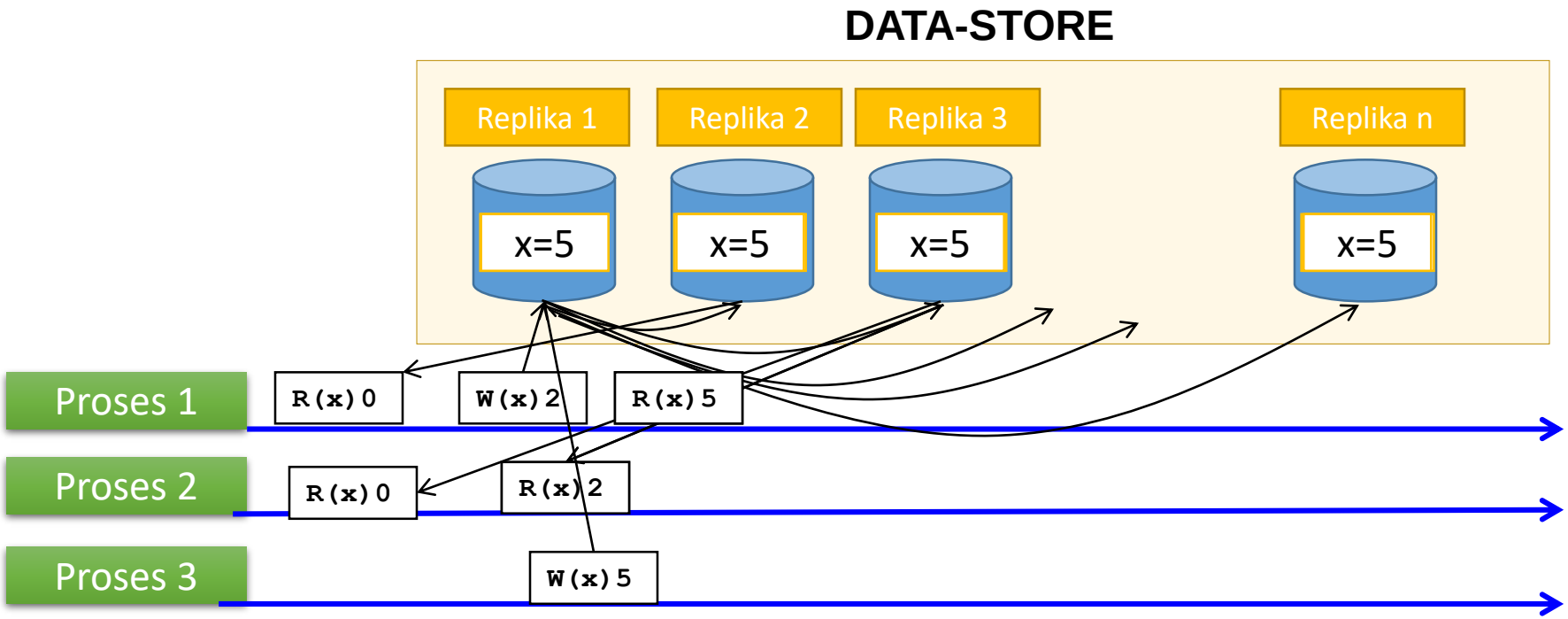
- Contoh:
  - Suatu Database Bank



Ikhtisar

- Motivasi
- Model-model Konsistensi
  - Model Konsistensi Data-Centric
  - Model Konsistensi Client-Centric
- Protokol-protocol Konsistensi.

Menjaga Konsistensi Data Replikasi

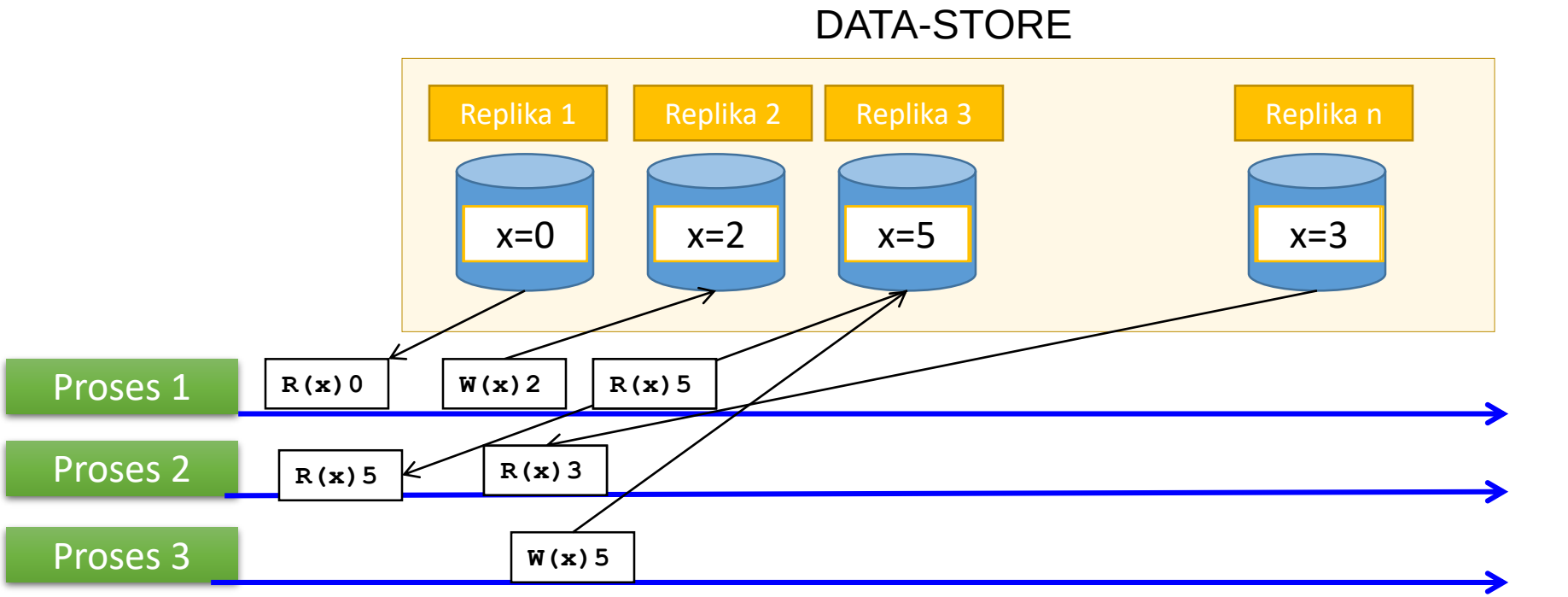


Konsistensi Keras (*Strict*)

- Data selalu *fresh* (update, terbaru)
  - Setelah suatu operasi write, update disebarkan (propagasi) ke semua replika
  - Suatu operasi read akan mengakibatkan pembacaan write paling baru
- Jika rasio read-to-write rendah, ini mengarah ke biaya yang besar

P1 =Proses P1 → =Timeline pada P1 R(x)b =Baca variabel x; Hasil b W(x)b = Write variabel x; Hasil b

Menjaga Konsistensi Data Replikasi



Konsistensi Loggar (Loose)

- Data mungkin saja basi (gak update)
  - Operasi baca dapat menghasilkan pembacaan nilai yang ditulis jauh sebelumnya
  - Replika umumnya tidak sinkron
- Replika dapat disinkronkan pada waktu *coarse grained*, sehingga mengurangi overhead

P1 =Proses P1 → =Timeline pada P1 R(x)b =Read variabel x; Hasil b W(x)b = Write variabel x; Hasil b

Trade-off Dalam Menjaga Konsistensi

- Perawatan konsistensi harus seimbang antara keketatan konsistensi vs. efisiensi (atau performa)
  - Konsistensi yang cukup baik tergantung pada aplikasi kita

Model Konsistensi

- *Consistency model* merupakan kontrak antara:
  - Proses yang ingin menggunakan data
  - Dan data-store yang menyediakan data

Suatu model konsistensi menyatakan level (atau derajat) konsistensi yang disediakan oleh data-store kepada proses selama membaca (*read*) dan menulis (*write*) data

Jenis Model Konsistensi

- Model konsistensi dapat dibagi ke dalam dua jenis:
  - Model konsistensi Data-Centric
    - Model jenis ini mendefinisikan bagaimana updates dipropagasi lintas replika untuk menjaga konsistensinya
  - Model Konsistensi Client-Centric
    - Model ini mengasumsikan bahwa clients terhubung ke replika-replikasi berbeda pada waktu berbeda
    - Memastikan bahwa kapanpun suatu client mengubungi suatu replika, replika tersebut membawakan yang up to date dengan replika yang client tersebut akses sebelumnya.

Ikhtisar

- Motivasi

- Model-model Konsistensi
  - Model Konsistensi Data-Centric
  - Model Konsistensi Client-Centric

- Protokol-protocol Konsistensi.

Pengurutan Konsisten dari Operasi

- Kita perlu mengekspresikan *semantics* dari akses-akses parallel saat *shared data* direplikasi

- Sebelum updates pada replika-replikasi dilaksanakan (commit), semua replika akan mencapai *suatu agreement on a global ordering* dari update tersebut
  - Yaitu, replika-replika dalam shared data-stores harus sepakat mengenai *consistent ordering of updates*

- Pengurutan update konsisten apa yang dapat disepakati oleh replika?

Jenis Pengurutan

- Berikut ini adalah tiga jenis pengurutan:
  1. Pengurutan Total
  2. Pengurutan Sequential
     - Model konsistensi Sequential
  3. Pengurutan Causal
     - Model konsistensi kausal

Jenis Pengurutan

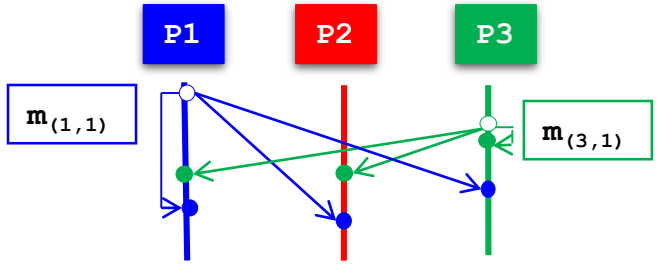
- Berikut ini adalah tiga jenis pengurutan:
  1. Pengurutan Total
  2. Pengurutan Sequential
     - Model konsistensi Sequential
  3. Pengurutan Causal
     - Model konsistensi kausal

Pengurutan Total

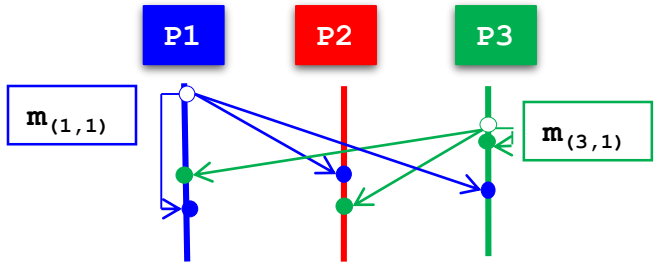
- Apa itu pengurutan total?
  - Jika proses P_i mengirimkan message m_i dan P_j mengirimkan m_j, dan jika satu proses yang benar menyampaikan m_i sebelum m_j maka setiap proses benar lainnya menyanmpaikan m_i *sebelum* m_j

- Message dapat menunjukkan update replika
  - Dalam Contoh 11, jika P_1 mengeluarkan operasi $m_{(1,1)}$: `x=x+1;` dan
  - Jika P_3 memberikan $m_{(3,1)}$: `print(x);` dan
  - P_1 atau P_2 atau P_3 mengantarkan $m_{(3,1)}$ sebelum $m_{(1,1)}$
  - Maka, pada semua replika P_1, P_2, P_3 urutan operasi berikut dieksekusi:

```
print(x);
x=x+1;
```



Contoh 1: Urut Total



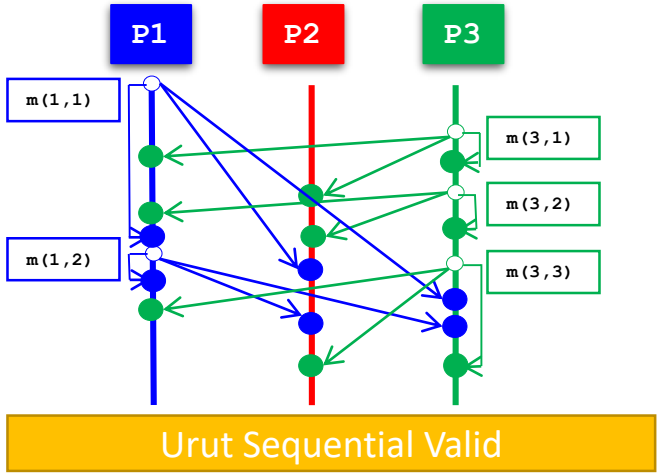
Contoh 2: Tidak Urut Total

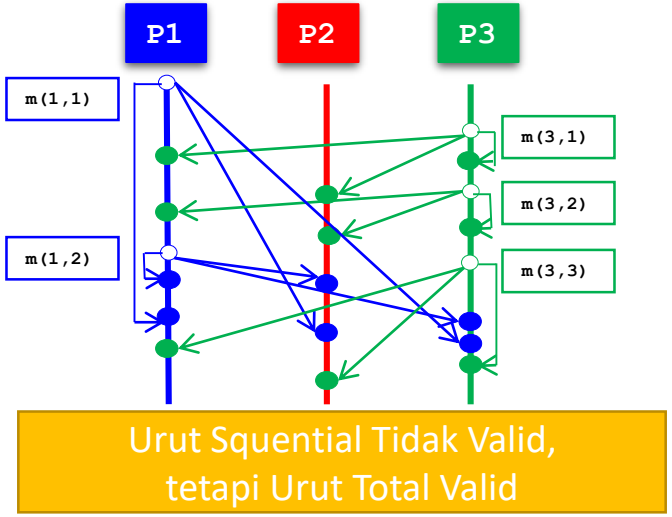
Jenis Pengurutan

- Berikut ini adalah tiga jenis pengurutan:
  1. Pengurutan Total
  2. Pengurutan Sequential
     - Model Konsistensi Sequential
  3. Pengurutan Kausal
     - Model Konsistensi Kausal

Pengurutan Sequential

- Apa itu pengurutan Sequensial?
  - Jika suatu proses **Pi** mengirimkan serangkaian messages $m_{(i,1)}, \ldots, m_{(i,ni)}$, dan
  - Proses **Pj** mengirimkan serangkaian messages $m_{(j,1)}, \ldots, m_{(j,nj)}$,
  - Maka:
    - Pada suatu proses, himpunan message tersebut diterima dalam urutan sequential (*some* sequential order)
    - Messages dari setiap proses individu harus muncul dalam urutan sama dengan yang dikirimkan oleh proses tersebut
      - Pada setiap proses, $m_{i,1}$ harus diantarkan sebelum $m_{i,2}$, yang harus diantarkan sebelum $m_{i,3}$ dan seterusnya...
      - Pada setiap proses, $m_{j,1}$ harus diantarkan sebelum $m_{j,2}$, yang harus diantarkan sebelum $m_{j,3}$ dan seterusnya...

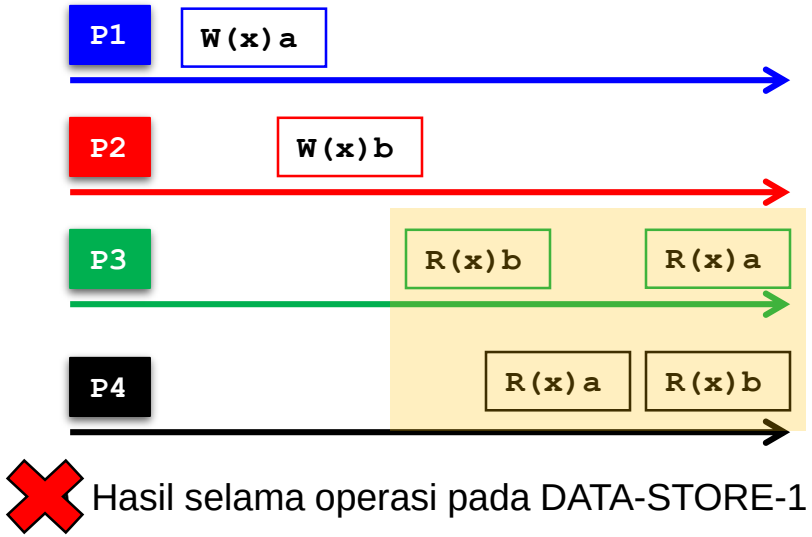




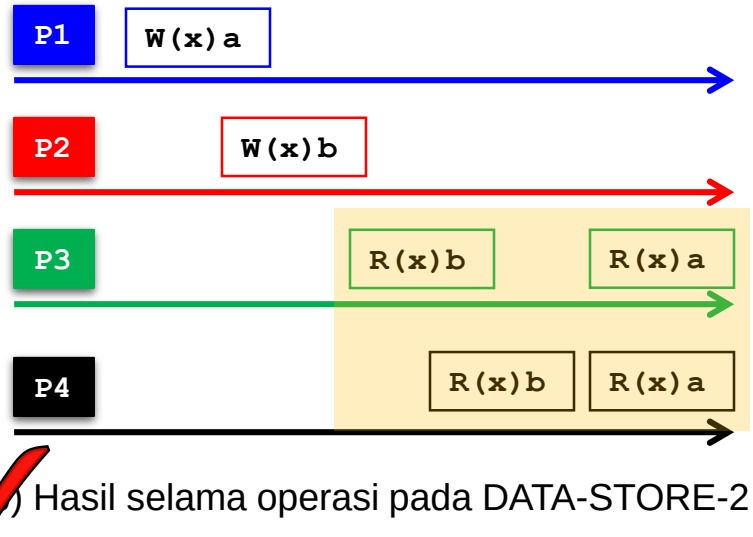
Model Konsistensi Sequential

- Model konsistensi Sequential mengharuskan bahwa semua operasi update diksekusi pada replika dalam suatu urutan sequential

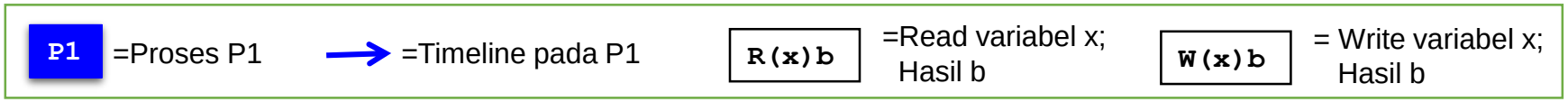
- Perhatikan suatu data-store dengan variable **x** (diberikan nilai awal **NULL**)
  - Dalam dua data-stores berikut, mana yang konsisten secara sekuensial?



(a) Hasil selama operasi pada DATA-STORE-1

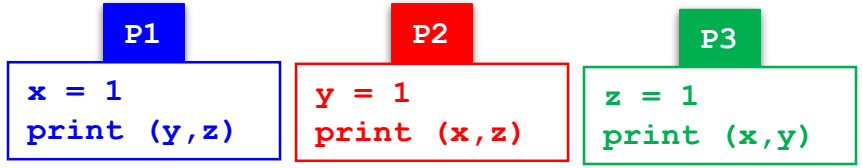


(b) Hasil selama operasi pada DATA-STORE-2



Konsistensi Sequential

- Perhatikan tiga proses P_1, P_2 dan P_3 yang mengeksekusi banyak instruksi pada tiga *shared* variables **x**, **y** dan **z**
  - Anggap bahwa **x**, **y** dan **z** dimulai dengan nilai nol



- Ada banyak *valid sequences* agar operasi dapat dieksekusi, dengan melihat konsistensi sequential (atau ***urutan program***)
  - Bagaimana suatu program dapat mengenali *wrong sequence* di antara rentetan berikut?

	x = 1 print (y,z) y = 1 print (x,z) z = 1 print (x,y)	x = 1 y = 1 print (x,z) print (y,z) z = 1 print (x,y)	print (y,z) y = 1 print (x,y) x = 1 print (x,z) z = 1	y = 1 z = 1 print (x,y) print (x,z) x = 1 print (y,z)
Output	001011	101011	000110	010111
Signature	001011	101011	001001	110101

Implikasi Adopsi Model Konsistensi Sequential bagi Aplikasi

- Ada beberapa kemungkinan kombinasi pengurutan yang konsisten secara sequensial
  - Jumlah kombinasi untuk total n instruksi = $O(n!)$

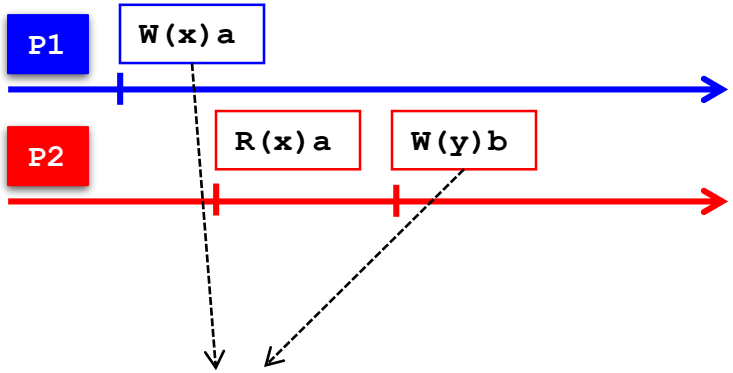
- Kontrak antara proses dan data-store terdistribusi adalah bahwa proses tersebut harus menerima semua sequential orderings sebagai hasil yang valid
  - Suatu proses yang bekerja untuk beberapa dari sequential orderings dan tidak untuk lainnya adalah INCORRECT

Jenis Pengurutan

- Berikut ini adalah tiga jenis pengurutan:
  1. Pengurutan Total
  2. Pengurutan Sequential Ordering
     - Model Konsistensi Sequential
  3. Pengurutan Kausal
     - Model konsistensi kausal

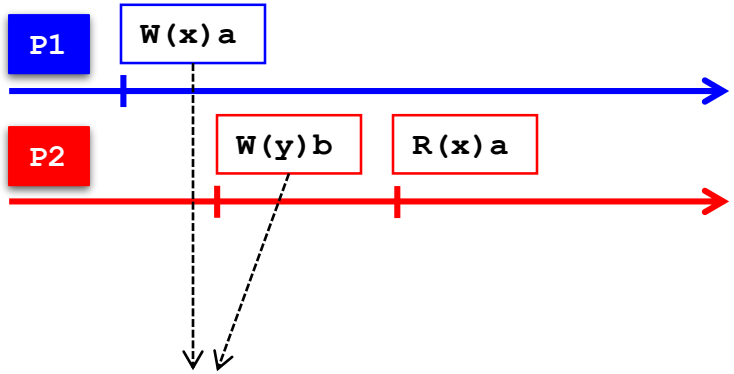
Kejadian Causal vs. Concurrent

- Perhatikan interaksi antara proses P_1 dan P_2 yang beroperasi pada data replikasi **x** dan **y**



Peristiwa terkait secara kausal
Events are not concurrent

- Computation of **y** at P_2 *may* have depended on the value of **x** written by P_1



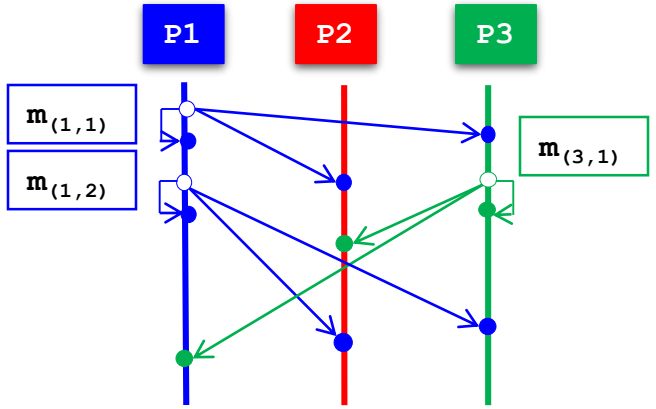
Peristiwa tidak saling terkait
Events are concurrent

- Computation of **y** at P_2 does not depend on the value of **x** written by P_1

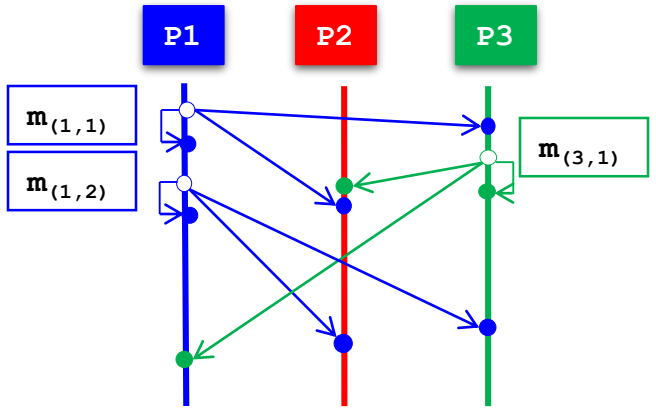
P1 =Proses P1 → =Timeline pada P1 R(x)b =Read variabel x; Hasil b W(x)b = Write variabel x; Hasil b

Pengurutan Causal

- Apa itu *causal ordering*?
  - Jika proses P_i mengirimkan message m_i dan P_j mengirimkan m_j, dan if $m_i \rightarrow m_j$ (operator '→' adalah relasi **happened-before/terjadi sebelum** Lamport) maka suatu proses benar yang mengantarkan m_j akan mengantarkan m_i sebelum m_j
- Contoh 1:
  - $m_{(1,1)}$ and $m_{(3,1)}$ are in Causal Order
  - $m_{(1,1)}$ and $m_{(1,2)}$ are in Causal Order
- Contoh 2:
  - $m_{(1,1)}$ and $m_{(3,1)}$ are NOT in Causal Order



Contoh 1: Urutan Kausal Valid

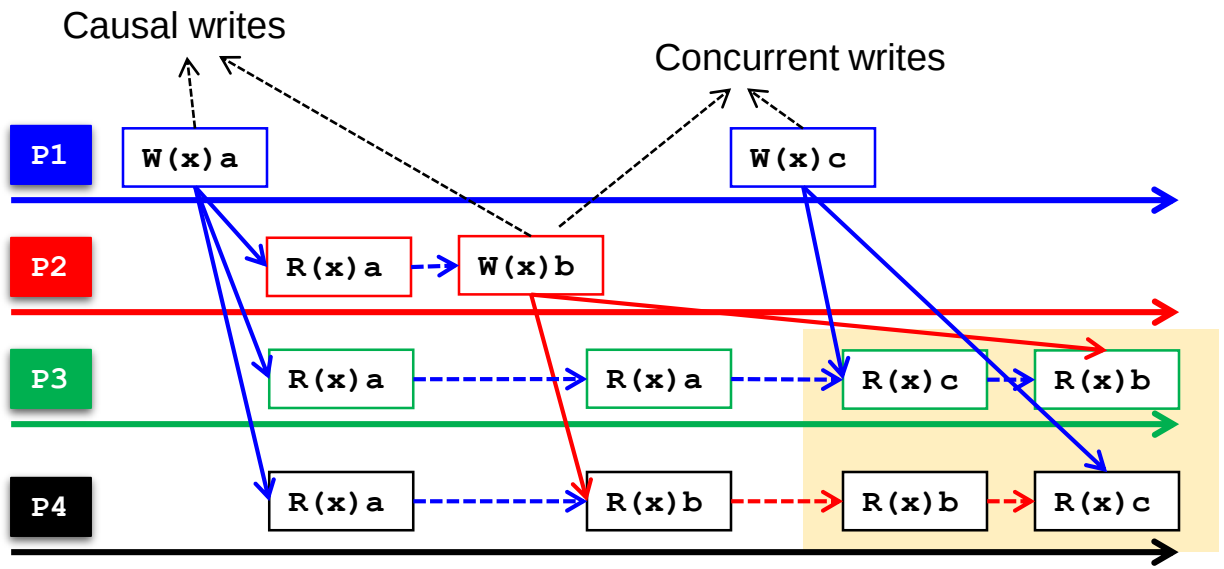


Contoh 2: Urutan Kausal Invalid

Model Konsistensi Causal

- Suatu data-store secara sebab-musabab konsisten jika:
  - Menulis yang berpotensi terkait secara kausal dilihat oleh semua proses dalam urutan yang sama
  - Tetapi penulisan bersamaan dapat dilihat dalam urutan berbeda pada mesin yang berbeda

Contoh *Causally Consistent Data-store*



A Causally Consistent Data-Store

But not a Sequentially Consistent Data-Store

P1 =Process P1 → =Timeline at P1 R(x)b =Read variable x; Result is b W(x)b = Write variable x; Result is b

Implikasi Adopsi *Causally Consistent Data-store* bagi Aplikasi

- Proses harus melacak proses mana yang telah menulis

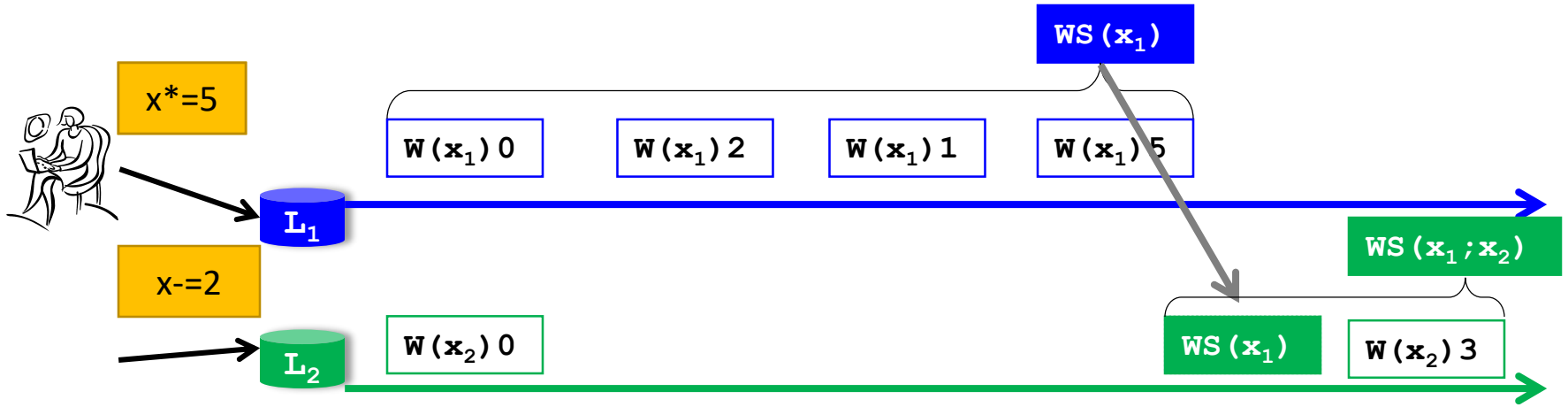
- Ini membutuhkan pemeliharaan grafik ketergantungan antara operasi tulis dan baca
  - Jam vektor menyediakan cara untuk mempertahankan penyimpanan data (data-store) yang konsisten secara kausal

Ikhtisar

- Motivasi
- Model-model konsistensi
  - Model konsisten Data-Centric
  - Model konsistensi Client-Centric
- Protokol Konsistensi

Garansi Konsistensi Client

- Konsistensi Client-centric menyediakan jaminan bagi client tunggal demi aksesnya ke data-store
  - Contoh: menyediakan jaminan konsistensi untuk suatu proses client bagi data **x** yang tereplikasi pada dua server. Katakanlah x_i merupakan salinan local dari data **x** pada server L_i



WS (x_1) = **Write Set for x_1** = Serangkaian operasi sedang dilakukan di beberapa replika that reflects how x_1 was updated at L_1 till this time

WS (x_1 ; x_2) = **Write Set for x_1 and x_2** = Serangkaian operasi sedang dilakukan di beberapa replika that reflects how x_1 was updated at L_1 and, later on, how x_2 was updated on L_2

L_i = Replica i

R(x_i)b = Read variable x at replica i; Result is b

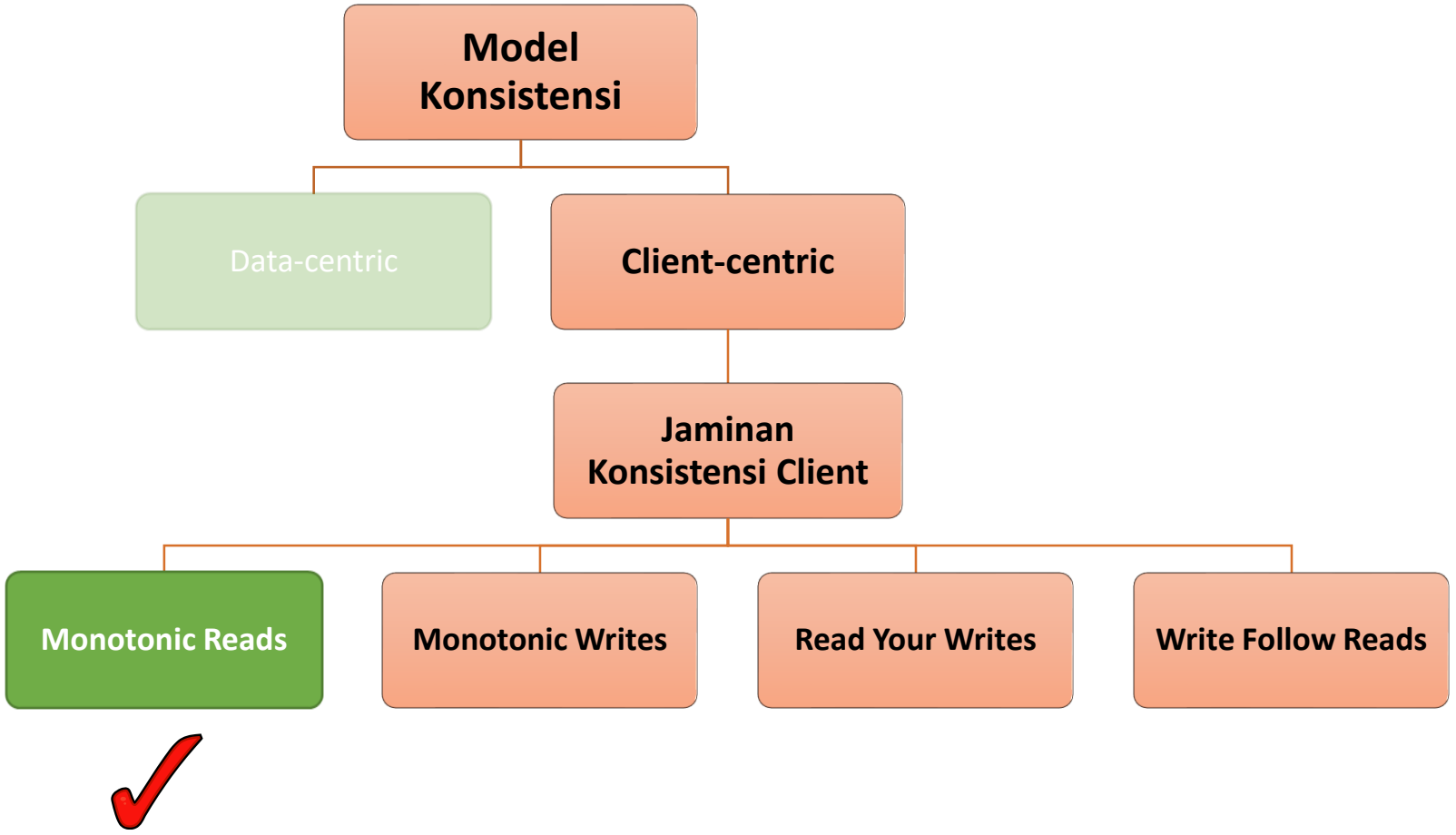
W(x)b = Write variable x at replica i; Result is b

WS (x_i) = Write Set

Client Consistency Guarantees

- Kita akan mempelajari empat jenis model konsisten client-centric
  1. Monotonic Reads (Membaca Monoton)
  2. Monotonic Writes
  3. Read Your Writes
  4. Write Follow Reads

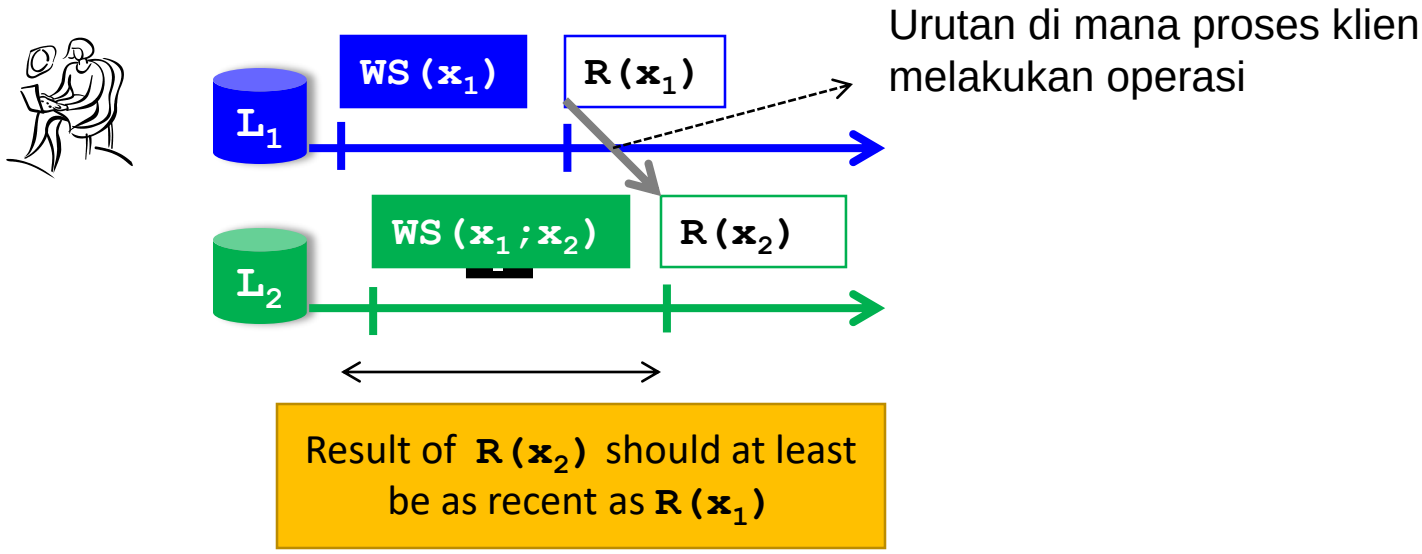
Ikhtisar


Model Konsistensi
Data-centric
Client-centric
Jaminan Konsistensi Client
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

Monotonic Reads

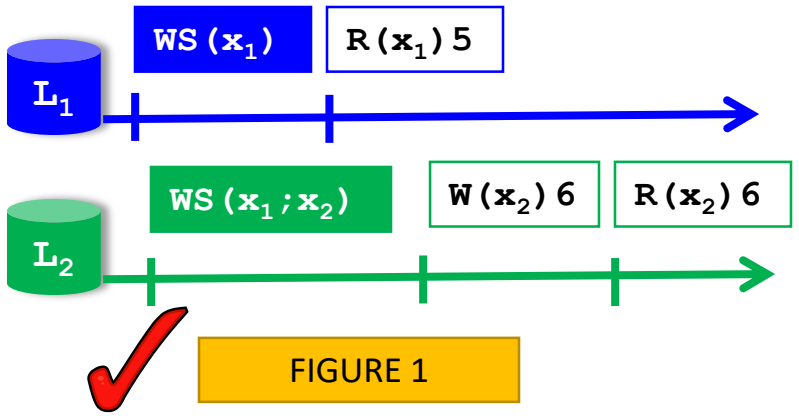
- Model ini menyediakan jaminan pada operasi read berturutan

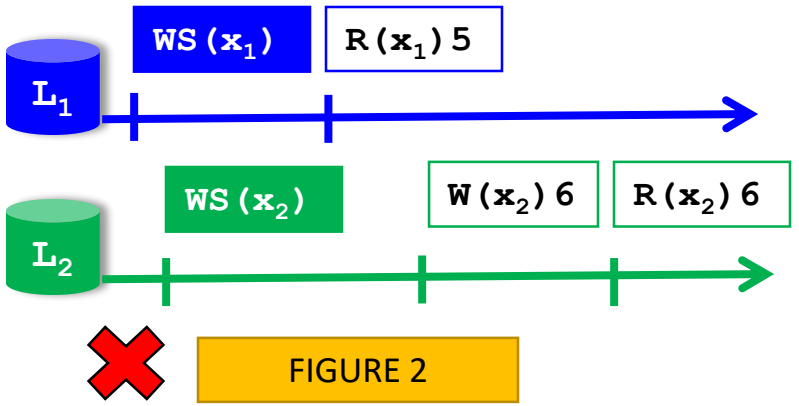
- Jika suatu proses client membaca nilai dari data item **x**, maka suatu operasi read berturutan oleh proses itu harus mengembalikan nilai **x** yang sama atau yang lebih baru

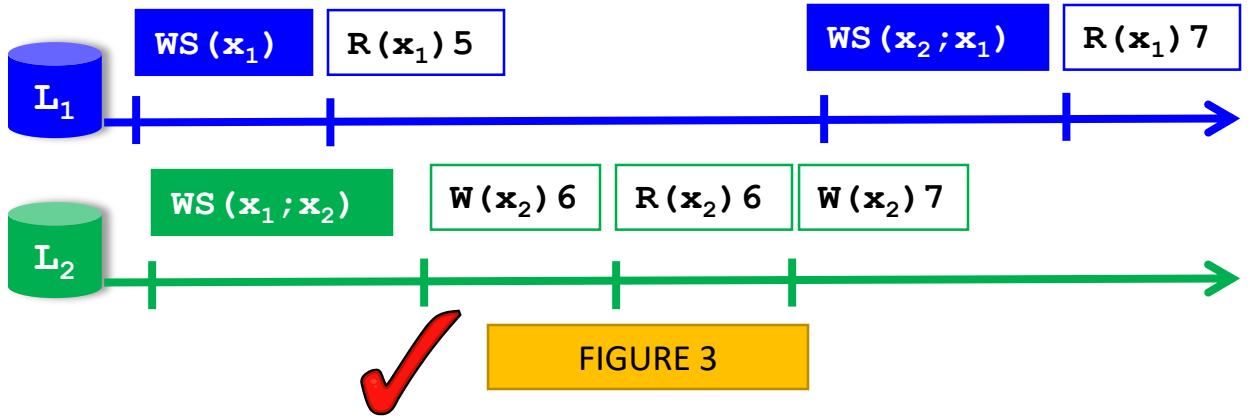


Monotonic Reads – Puzzle

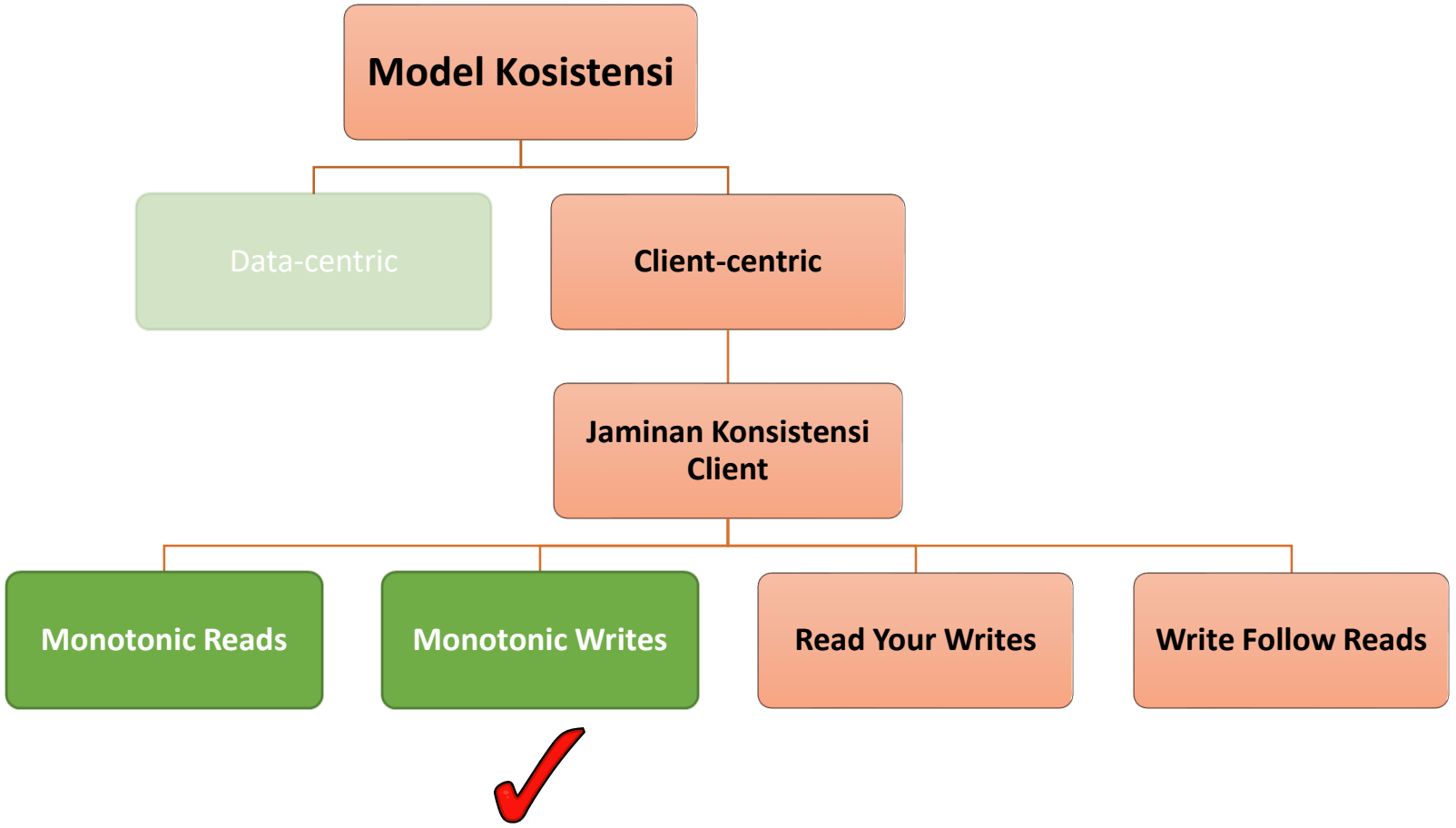
- Memahami data-stores yang menyediakan jaminan monotonic read.







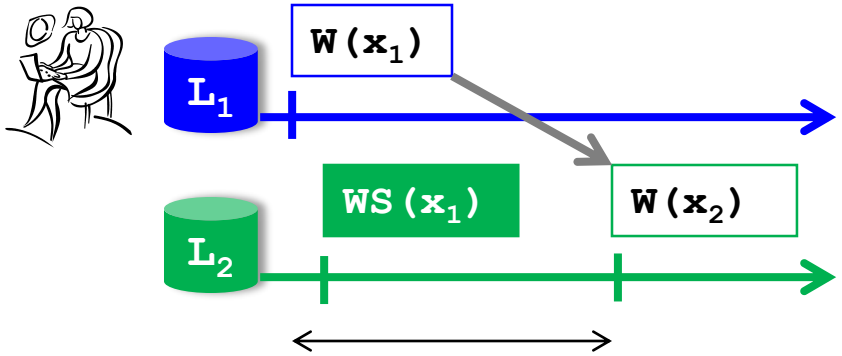
Ikhtisar


Model Kosistensi
Data-centric
Client-centric
Jaminan Konsistensi Client
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

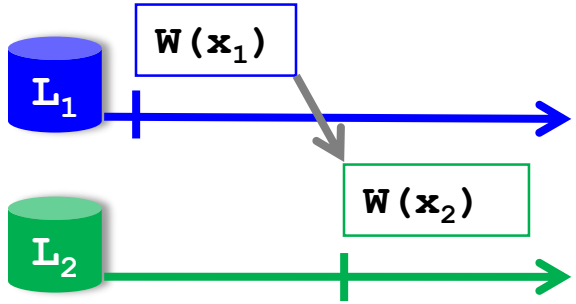
Monotonic Writes

- Model konsistensi ini memastikan bahwa operasi writes bersifat monotonic

- Operasi write oleh suatu proses client pada item data **x** diselesaikan *sebelum suatu operasi write berturutan* pada **x** oleh *proses yang sama*
  - Write baru pada suatu replika harus menunggu semua write lama pada replika apapun

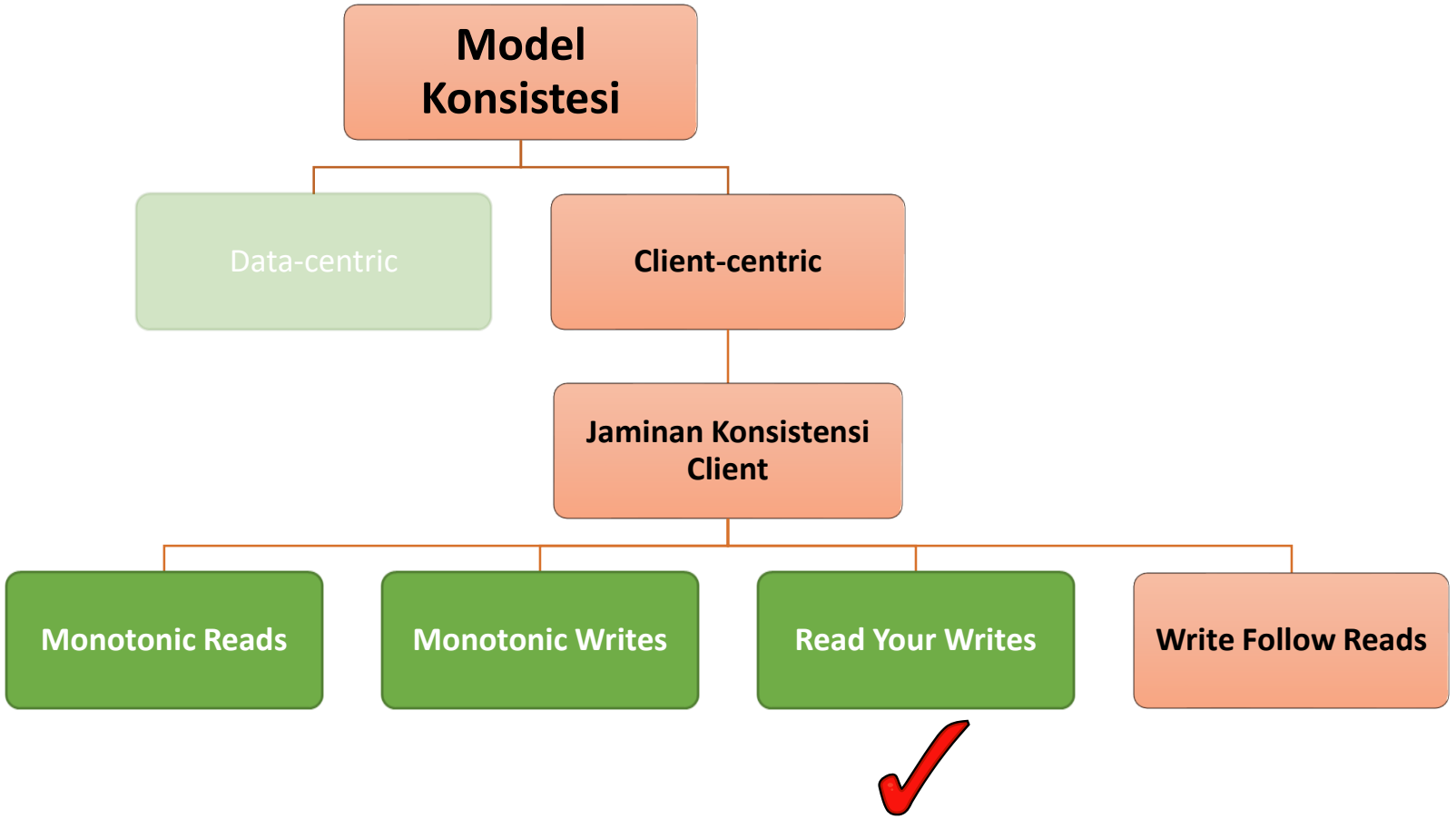


Operasi **W(x2)** harus dilaksanakan hanya setelah hasil dari **W(x1)** telah diupdate pada **L2**



Data-store ini tidak menyediakan konsistensi monotonic write

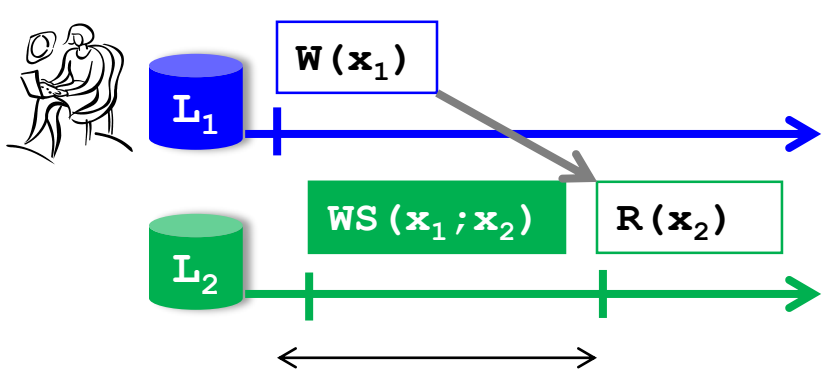
Ikhtisar


Model Konsistesi
Data-centric
Client-centric
Jaminan Konsistensi Client
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

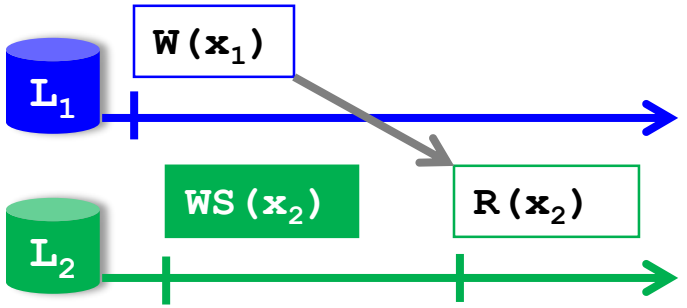
Read Your Writes

- Pengaruh dari suatu operasi *write* pada suatu item data **x** oleh suatu proses akan *selalu dilihat oleh suatu operasi read* berturutan pada **x** oleh proses yang sama

- Skenario contoh:
  - Dalam sistem dimana password disimpan di dalam suatu replicated data-base, perubahan password harus dipropagasi ke semua replika.

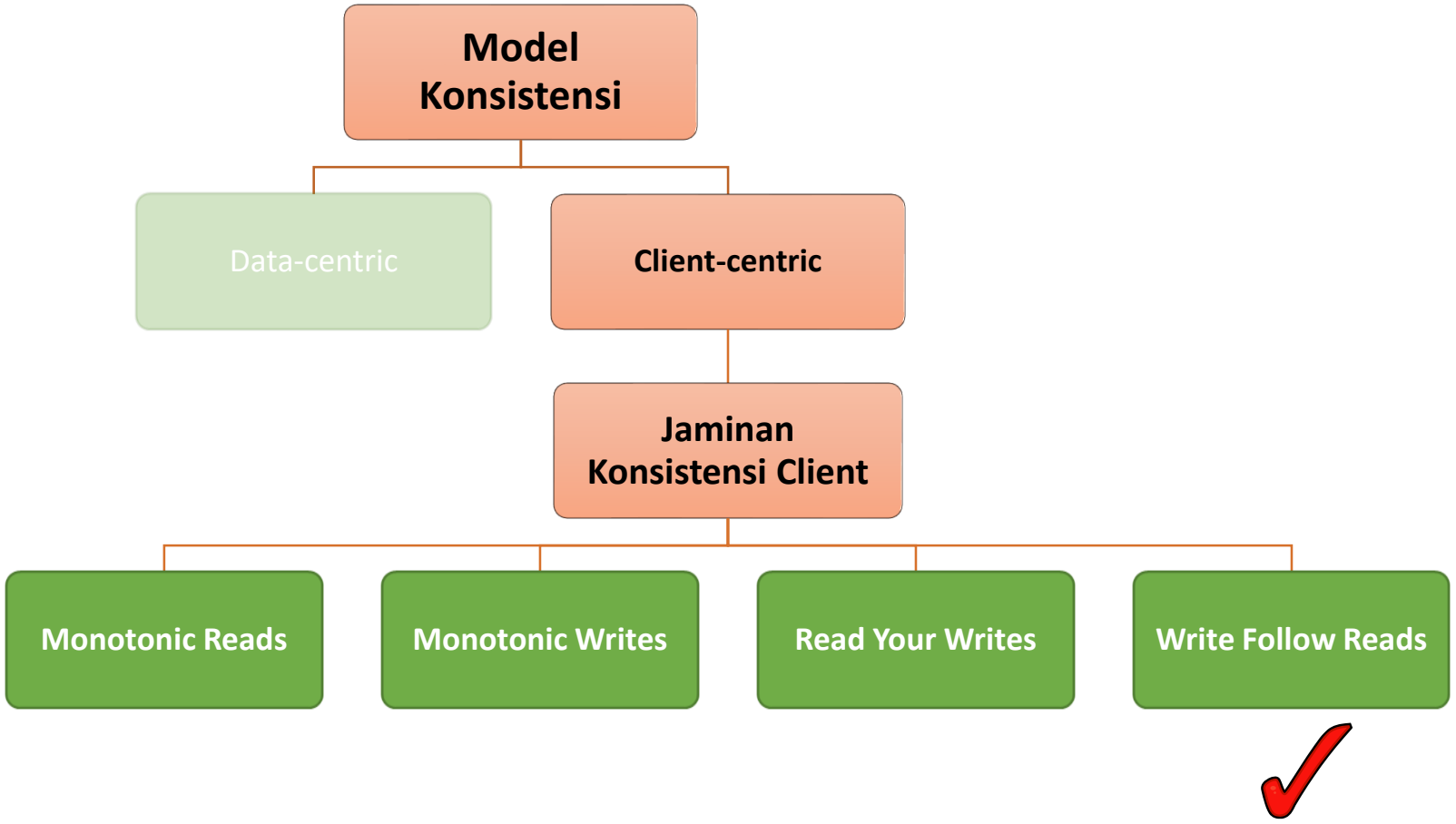


Operasi **R(x_2)** harus dikerjakan hanya setelah mempropagasi **WS(x_1)** ke **L_2**



Suatu data-store yang tidak menyediakan konsistensi *Read Your Write*

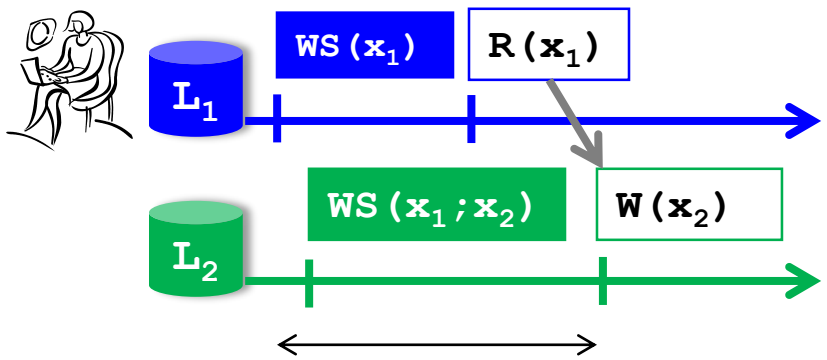
Ikhtisar


Model Konsistensi
Data-centric
Client-centric
Jaminan Konsistensi Client
Monotonic Reads
Monotonic Writes
Read Your Writes
Write Follow Reads

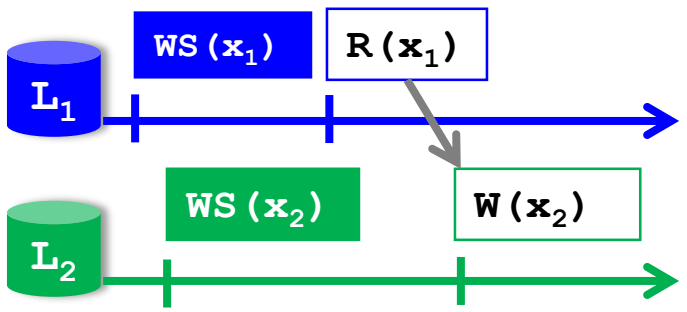
Write Mengikuti Reads

- Operasi tulis oleh suatu proses pada item data x setelah operasi baca sebelumnya pada x oleh proses yang sama dijamin untuk berlangsung pada nilai x yang sama atau yang lebih baru yang dibaca

- Skenario contoh:
  - Pengguna dari suatu newsgroup harus mempost komentar mereka hanya setelah mereka selesai membaca artikel dan (semua) komentar sebelumnbya



operasi **W (x_2)** harus dikerjakan hanya setelah semua write sebelumnya selesai dipropagasi



Suatu data-store yang tidak menjamin Model konsistensi Write Follow Read

Ikhtisar

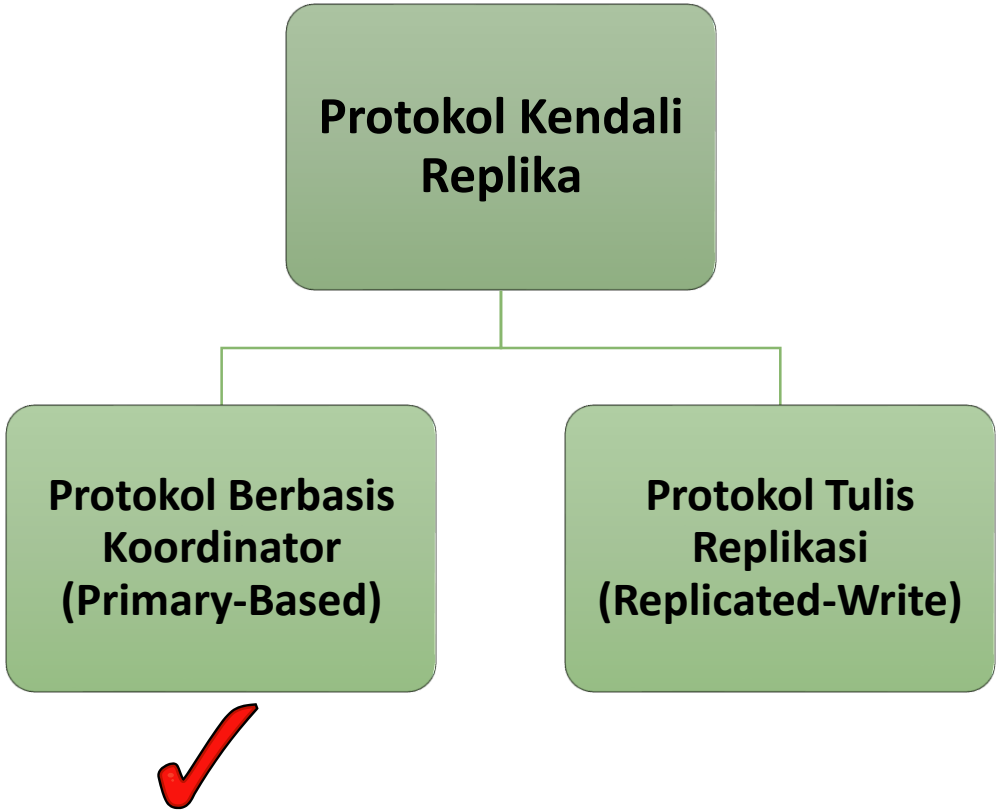
- Motivasi
- Model Konsistensi
  - Model konsistensi Data-Centric
  - Model konsistensi Client-Centric
- Protokol konsistensi

Protokol Konsistensi

- Suatu protocol konsistensi menggambarkan *implementasi* dari suatu model konsistensi spesifik (missal: konsistensi ketat)

- Kita akan kaji 2 jenis protocol konsistensi:
  - Protokol Primary-based
    - Satu koordinator primer *dipilih* untuk mengendalikan replikasi lintas banyak replika
  - Protokol Replicated-write
    - Banyak replika berkoordinasi untuk menyediakan jaminan konsistensi.

Protokol Konsistensi



Protokol Berbasis Koordinator

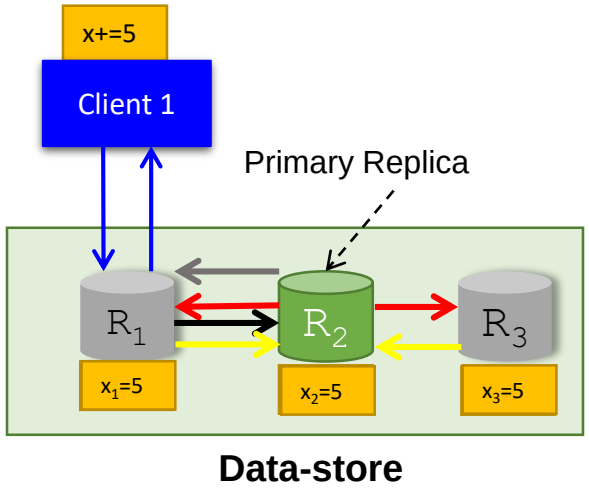
- Dalam protocol berbasis primer (koordinator), suatu rancangan terpusat sederhana digunakan untuk mengimplementasikan model konsistensi
  - Setiap data-item **x** mempunyai suatu "*replika primer*" berkaitan
  - Replika primer bertanggungjawab mengkoordinasi operasi write

- Kita akan mengkaji satu contoh protocol primary-based yang mengimplementasikan Model konsistensi *Strict*
  - Protokol Remote-Write

Protokol Remote-Write

- Dua aturan:
  - Semua operasi write diteruskan (diforward) ke replika primer
  - Operasi read diruntaskan secara lokal pada setiap replika

- Pendekatan untuk operasi write:
  - Client menghubungi beberapa replika $\mathbf{R_C}$
  - Jika client meminta operasi write terhadap $\mathbf{R_C}$
    - $\mathbf{R_C}$ meneruskan request tersebut ke replika primer $\mathbf{R_P}$, yang
      - Mengupdate nilai lokalnya
      - Kemudian menformward update itu ke replika lain $\mathbf{R_i}$
    - Replika lain $\mathbf{R_i}$ mengerjakan updates, dan mengirimkan ACKs balik ke $\mathbf{R_P}$
  - Setelah $\mathbf{R_P}$ menerima semua ACK, ia mengabarkan $\mathbf{R_C}$ bahwa operasi write tersebut telah sukses
  - $\mathbf{R_C}$ menjawab client, menyatakan bahwa operasi write telah berhasil.

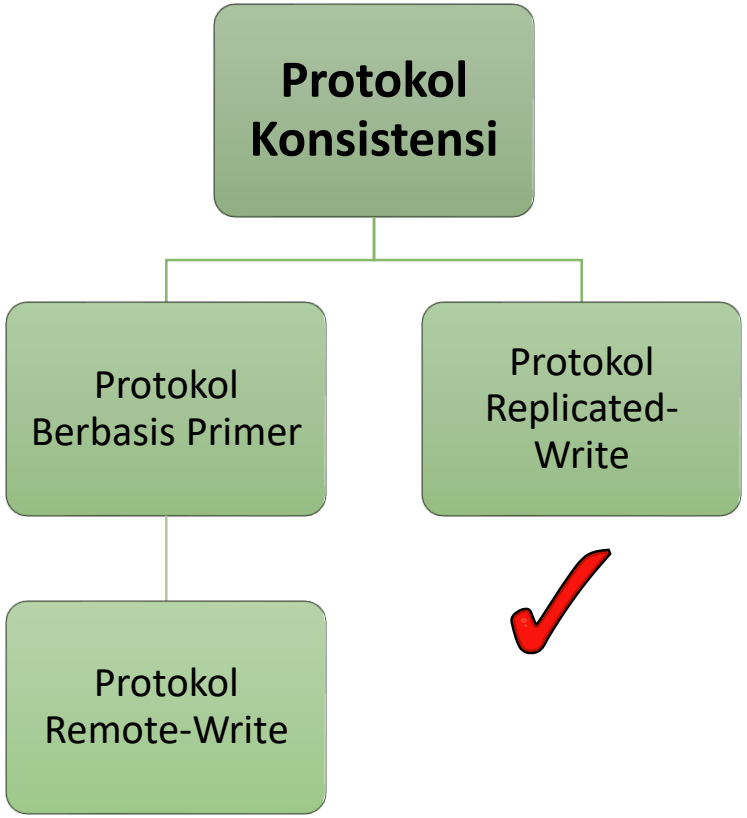


Protokol Remote-Write: Diskusi

- Protokol Remote-Write
  - Menyediakan suatu cara sederhana untuk mewujudkan konsistensi yang ketat
  - Menjamin bahwa clients melihat selalu nilai paling akhir (terbaru)

- Namun, latensi tinggi di dalam protocol Remote-Write ini
  - Client blocks sampai semua replika terupdate
  - Dalam scenario apa sebaiknya kita menggunakan protocol Remote-Write?
    - Biasanya, untuk database dan sistem file terdistribusi dalam data-centers (yaitu di lingkungan LAN)
      - Replika-replika ditempatkan pada LAN yang sama untuk mengurangi latensi.

Protokol Konsistensi



Protokol Replicated-Write

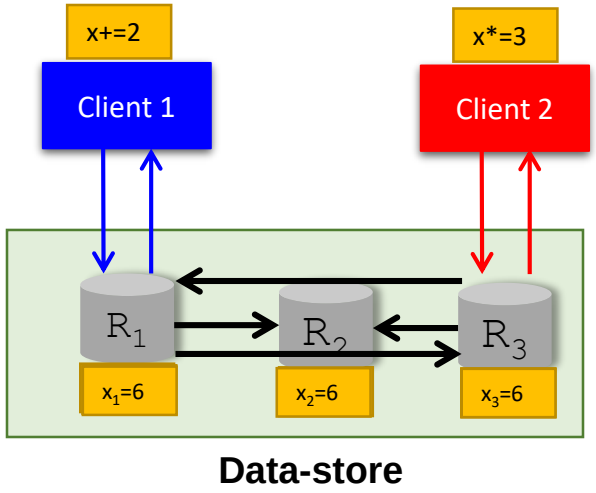
- Dalam protokol replicated-write, updates dapat langsung dilaksanakan pada banyak replika

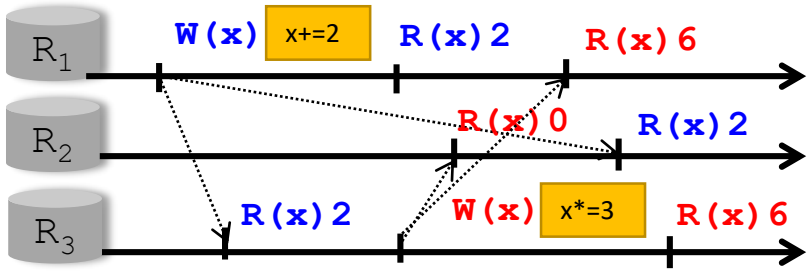
- Kita akan mengkaji dua contoh protokol replicated-write
  - Protokol Replikasi Aktif
    - Clients write (menulis) pada *suatu* replika (tidak ada replika primer)
    - Replika yang berubah akan mempropagasi update ke replika lain
  - Protokol Berbasis Quorum
    - Suatu **skema voting** digunakan

Protokol Replikasi Aktif

- Protokol: ketika suatu client menulis pada suatu replika, replika tersebut akan mengirimkan update itu ke semua replika lain

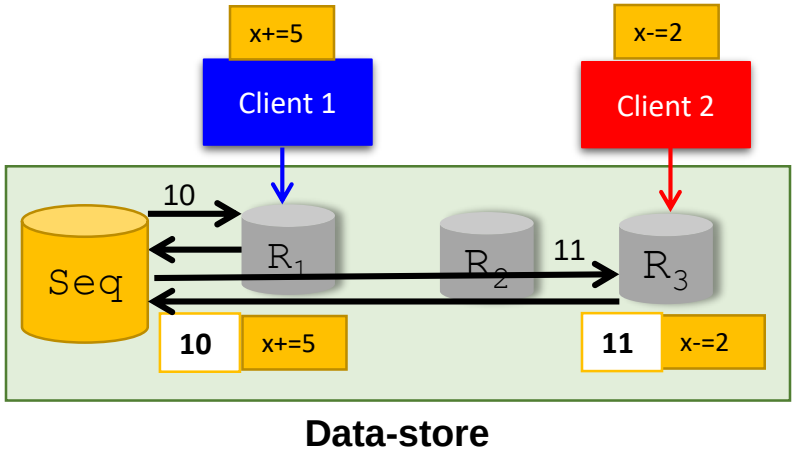
- Tantangan dengan Replikasi Aktif
  - Pengaturan operasi dapat berbeda yang mengarah pada konflik / ketidakkonsistenan
  - Jadi bagaimana mempertahankan pengurutan yang konsisten?





Protokol Replikasi Aktif Terpusat

- Suatu pendekatan yang mungkin:
  - Memilih suatu koordinator terpusat (dapat dinamakan *sequencer* (**Seq**))
  - Ketika suatu client menghubungi suatu replika $\mathbf{R_C}$ dan meminta operasi write
    - $\mathbf{R_C}$ meneruskan update itu ke **Seq**
    - **Seq** memberikan suatu *sequence number* untuk operasi update tersebut
    - $\mathbf{R_C}$ mempropagasi sequence number dan operasi tersebut ke replika lain
  - Operasi dituntaskan pada semua replika sesuai dengan urutan dari *sequence numbers*



Protokol Replicated-Write

- Dalam protokol *replicated-write*, updates dapat dilaksanakan langsung pada banyak replika

- Akan dikaji dua contoh protokol *replicated-write*
  - Protokol Replikasi Aktif
    - Clients menulis pada replika tertentu (bukan replika primer)
    - Replika tersebut akan mempropagasi update ke replika lain
  - Protokol Berbasis Quorum
    - Menggunakan suatu skema *voting*

Protokol Berbasis Quorum

- Replicated writes juga dapat disusun melalui penggunaan suatu skema *voting*, awalnya diajukan oleh Thomas (1979) kemudian digeneralkan oleh Gifford (1979)

- Gagasan dasarnya (*Rekap*):
  - Clients diharuskan untuk **meminta dan memperoleh** ijin dari banyak server sebelum *reading* atau *writing* dari atau ke item data tereplikasi
    - Rules pada reads dan writes harus dibangun lebih dulu
    - Setiap replika diberikan suatu *version number*, yang dinaikkan pada setiap operasi write

Protokol Berbasis Quorum

- Contoh Bekerja:
  - Perhatikan suatu sistem file terdistribusi dan anggap bahwa suatu file direplikasikan pada ***N*** server
  - Aturan write:
    - Suatu client pertama harus menghubungi ***N/2 + 1*** servers (suatu *majority*) sebelum mengupdate sebuah file
    - Sekali suara mayoritas diperoleh, file tersebut diupdate dan nomor versinya dinaikkan
      - Ini diteruskan pada situs-situs replika.

Protokol Berbasis Quorum

- Contoh Bekerja:
  - Perhatikan suatu sistem file terdistribusi dan anggap bahwa suatu file direplikasi pada ***N*** server
  - Aturan Read:
    - Suatu client harus menghubungi ***N***/2 + 1 servers, meminta mereka untuk mengirimkan nomor versi mereka dari file yang diminta (yang akan dibaca)
    - Jika semua nomor versinya tepat sama, ini haruslah versi paling baru dari file tersebut.

Protokol Berbasis Quorum

- Skema Gifford mengeneralkan usulan dari Thomas

- **Skema Gifford**:
  - Aturan Read:
    - Suatu client perlu untuk mengumpulkan suatu *read quorum*, yang merupakan suatu koleksi berubah-ubah dari sembarang N_R server, atau lebih
  - Aturan Write:
    - Untuk memodifikasi suatu file, suatu *write quorum* setidaknya sebanyak N_W server diwajibkan

Protokol Berbasis Quorum

- Nilai dari N_R dan N_W merupakan persoalan dua pembatasan berikut:
  - Konstrain 1 (atau C1): $N_R + N_W > N$
  - Konstrain 2 (atau C2): $N_W > N/2$

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

Protokol lain yang diajukan oleh Lamport (1998)
dan dikenal sebagai *Paxos*

Asumsi Dalam Paxos

- Paxos mengasumsikan model asinkron dan non-Byzantine (*le bih lanjut lihat bahasan mengenai fault-tolerance*), dimana:
  - Proses:
    - Beroperasi pada kecepatan berubah-ubah (*arbitrary*)
    - Dapat gagal dengan stopping, tetapi dapat di-restart
      - Dikarenakan suatu proses dapat gagal setelah suatu *value is chosen* dan kemudian restart, suatu solusi adalah tidak mungkin kecuali beberapa informasi dapat diingat (missal *melalui logging*) oleh suatu proses yang mengalami gagal dan direstart.
  - Messages:
    - Dapat hilang, diduplikasi, mengalami delay (sehingga diurut-ulangkan), tetapi **jangan** rusak (*corrupt*)

Peranan Dalam Paxos

- Proses di dalam Paxos dapat mengambil peran tertentu:
  - Client:
    - Menerbitkan suatu request (misal menulis pada file yang tereplikasi) ke sistem terdistribusi dan menunggu suatu response
  - Proposer (*atau suatu proses menawarkan menjadi koordinator/leader*):
    - Penyokong bagi suatu Client dan menganjurkan nilai untuk dipertimbangkan oleh *Acceptors*
  - Acceptor (*atau voter*):
    - Memperhatikan nilai yang diusulkan oleh Proposers dan membuat suatu keputusan accept/reject
  - Learner:
    - begitu suatu request Client telah ***disepakati*** oleh Acceptors, Learner dapat mengambil tindakan (missal mengeksekusi request tersebut dan mengirimkan suatu response kepada Client)

Quorum Dalam Paxos

- Sesuatu message yang dikirimkan ke suatu Acceptor harus dikirim ke suatu *quorum of Acceptors* yang terdiri dari *lebih setengah* dari semua Acceptors (yaitu mayoritas, bukan *kebulatan-suara*)

- Dua quorum harus mempunyai irisan tidak kosong (*nonempty intersection*)
  - Note biasa bertindak sebagai “tie-breaker”
  - Ini membantu menghindari masalah “split-brain” (atau situasi ketika keputusan Acceptors tidak dalam kesepakatan)

- Dalam suatu sistem dengan 2***m***+1 Acceptors, ***m*** Acceptors dapat gagal dan consensus masih dapat dicapai.

Algoritma Paxos: Fase I

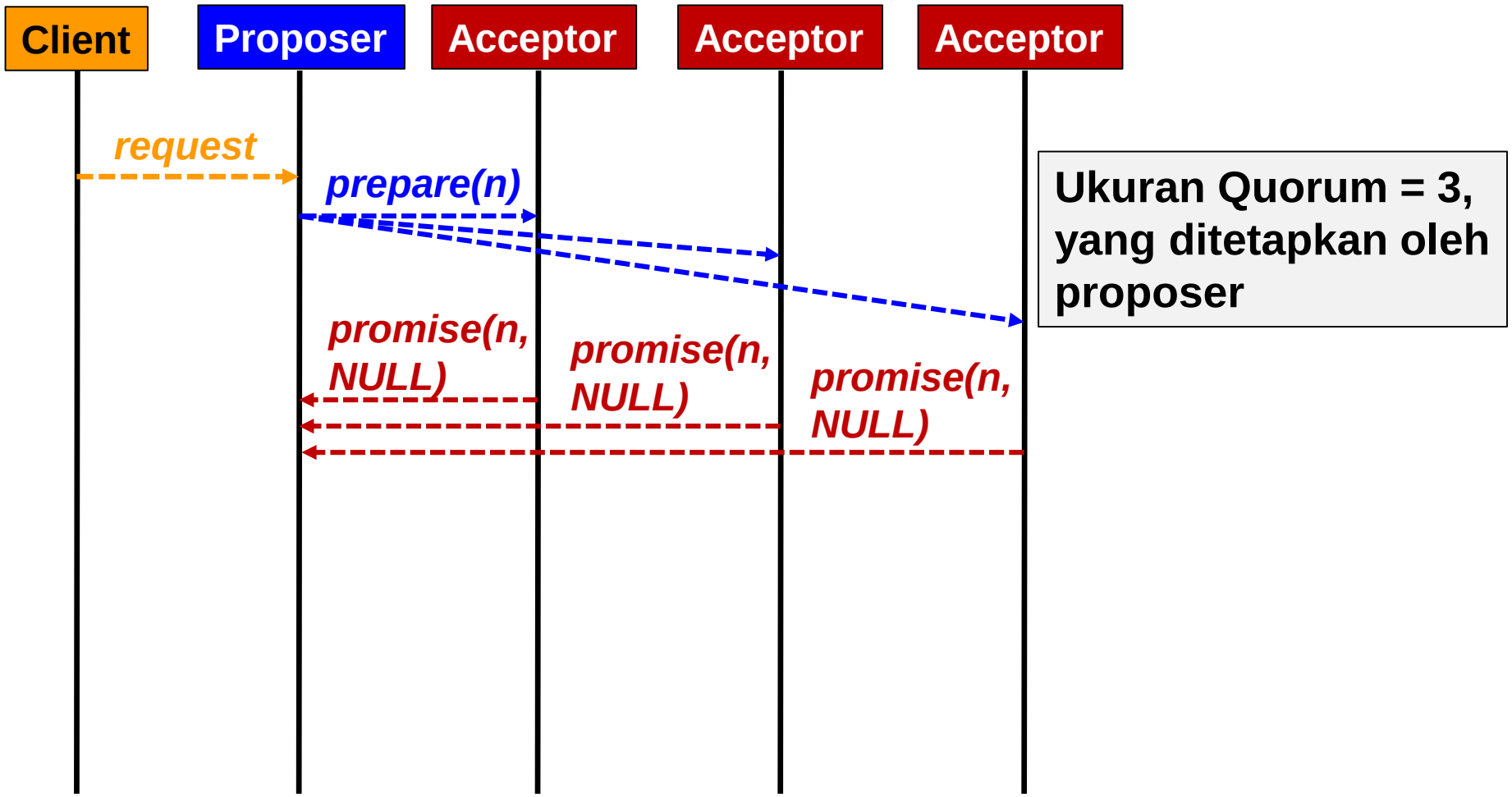
Fase I	
Langkah 1: Siap-siap	Proposer memilih suatu nomor urut *unik* (atau ronde) ***n*** dan mengirimkan suatu request ***prepare(n)*** ke suatu quorum Acceptors

- Catatan: banyak proses dapat meminta menjadi koordinator
- Karena itu, bagaimana dapat setiap koordinator memilih suatu nomor urut *unik* itu?
  - Setiap proses, ***P***, dapat diberikan suatu ID_P unik, antara 0 dan ***k*** – 1, dianggap total ada ***k*** proses
  - ***P*** dapat memilih nomor urut paling kecil, ***s***, yang lebih besar daripada semua nomor urut yang telah terlihat sejauh ini, sehingga ***s*** % ***k*** = ID_P
  - Misal, ***P*** akan mengambil nomor urut 23 untuk tawaran berikutnya jika ID_P = 3, ***k*** = 5, dan nomor terbesar sebelumnya = 20

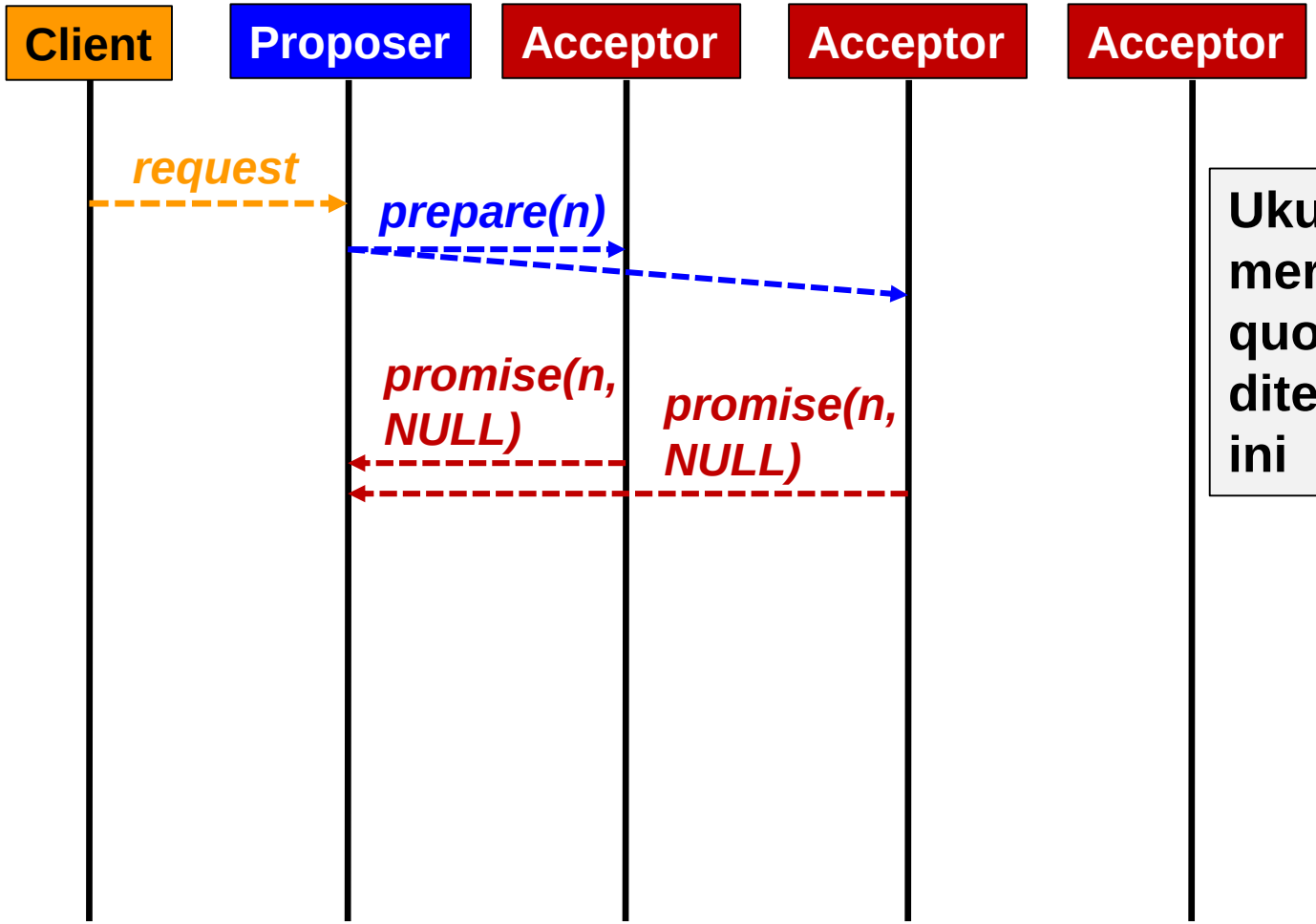
Algoritma Paxos: Fase I

Fase I	
Langkah 1: Siap-siap (prepare)	Proposer memilih suatu nomor urut *unik* (atau ronde) ***n*** dan mengirimkan suatu request ***prepare(n)*** ke suatu quorum Acceptors
Langkah 2: Berjanji (promise)	Setiap Acceptor melakukan berikut: ▪ Jika ***n*** > (nomor urut dari suatu promises atau acceptances sebelumnya) ▪ Menuliskan ***n*** ke suatu stable storage, berjanji bahwa ia tidak akan pernah menerima suatu nomor yang diajukan dimasa depan yang kurang dari ***n*** ▪ Mengirimkan suatu respon ***promise(n, (N, U))***, dimana ***N*** dan ***U*** adalah nomor urut terakhir dan nilai yang ia terima (*accepted*) sejauh ini (*jika ada*).

Contoh


Client
Proposer
Acceptor
Acceptor
Acceptor
request
prepare(n)
promise(n, NULL)
promise(n, NULL)
promise(n, NULL)
Ukuran Quorum = 3, yang ditetapkan oleh proposer

Contoh



Ukuran Quorum = 2, merupakan ukuran quorum *min* yang dapat diterima dalam contoh ini

Algoritma Paxos: Fase II

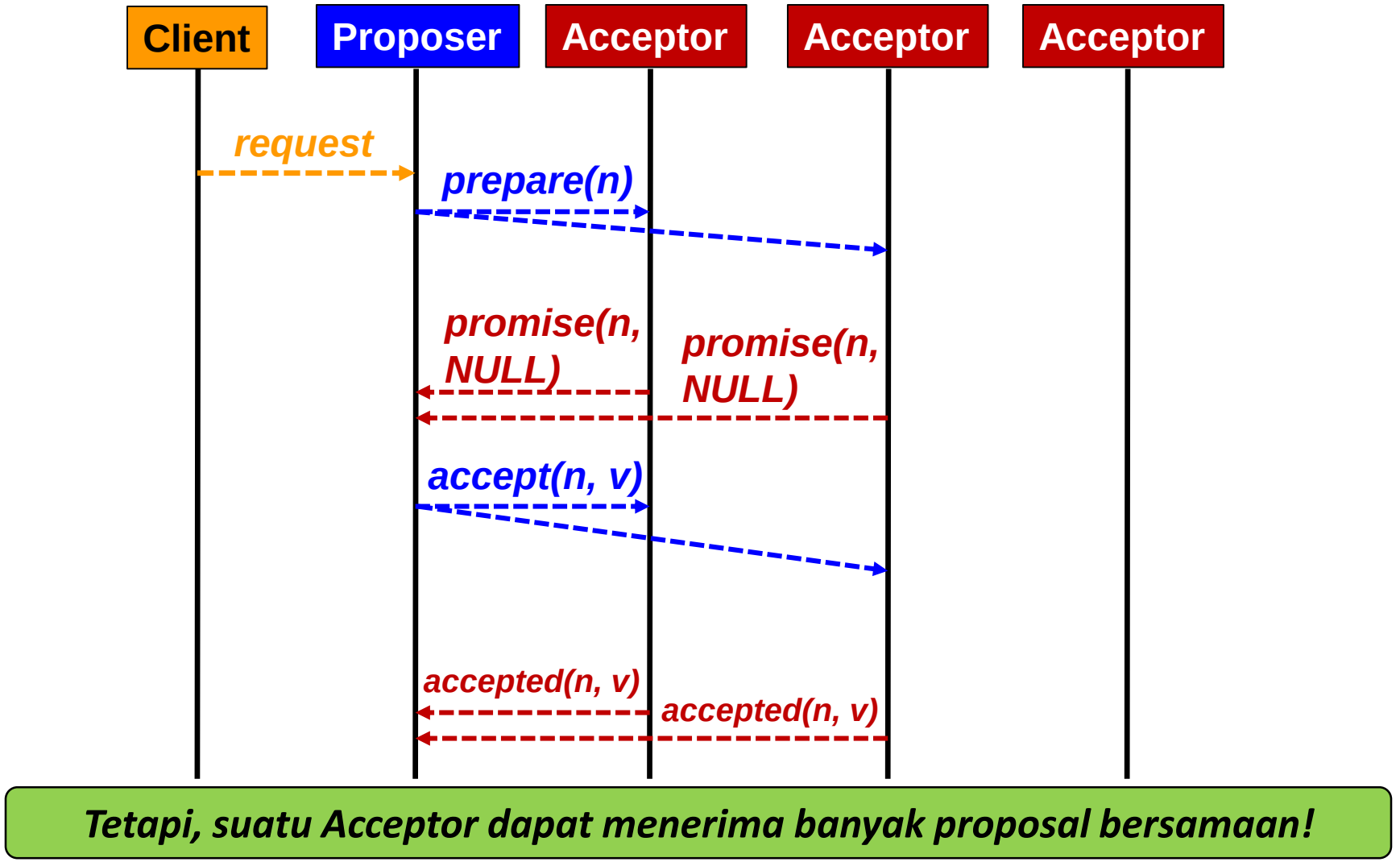
Fase II	
Langkah 1: Menerima (*accept*)	Jika Proposer menerima respon promise dari suatu quorum Acceptors, ia mengirim suatu request ***accept(n, v)*** ke para Acceptors itu (***v*** *adalah nilai dari proposal bernomor tertinggi di antara respon promise, atau suatu nilai jika promise tidak berisi suatu proposal*)

Algoritma Paxos: Fase II

Fase II	
Langkah 1: Menerima	Jika Proposer menerima respon promise dari suatu quorum Acceptors, ia mengirim suatu request ***accept(n, v)*** ke para Acceptors itu (***v*** *adalah nilai dari proposal bernomor tertinggi di antara respon promise, atau suatu nilai jika promise tidak berisi suatu proposal*).

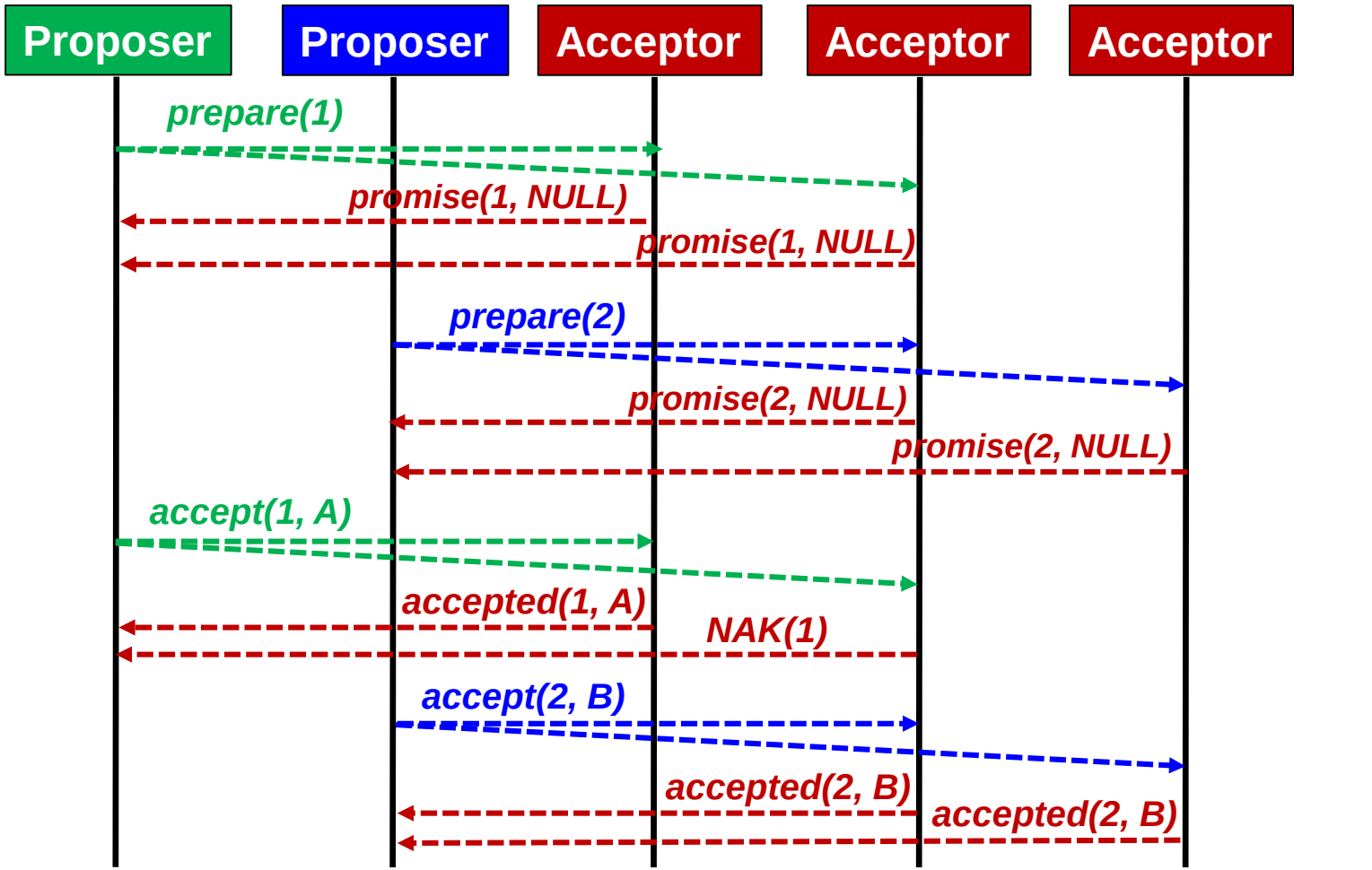
Langkah 2: Diterima (*accepted*)	Setiap Acceptor melakukan ini:
	
	▪ Jika ***n*** >= nomor dari suatu premise sebelumnya
	  ▪ Menuliskan (***n***, ***v***) ke suatu stable storage, mengindikasikan bahwa ia menerima (accept) proposal tersebut
	  ▪ Mengirimkan suatu respon ***accepted(n, v)***
	▪ Else (jika tidak)
	  ▪ Tidak menerima (mengirimkan suatu NACK)

Contoh



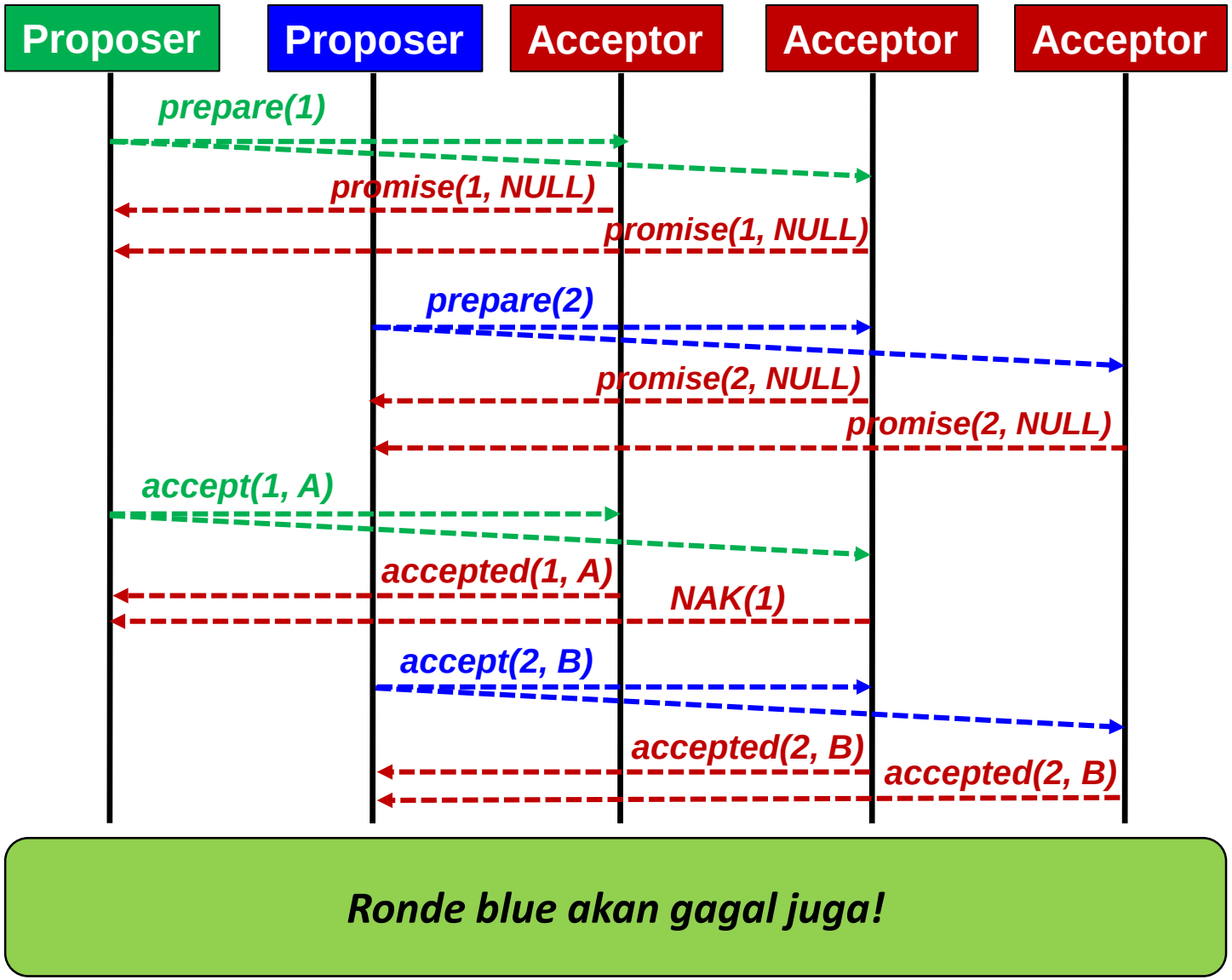
Tetapi, suatu Acceptor dapat menerima banyak proposal bersamaan!

Contoh

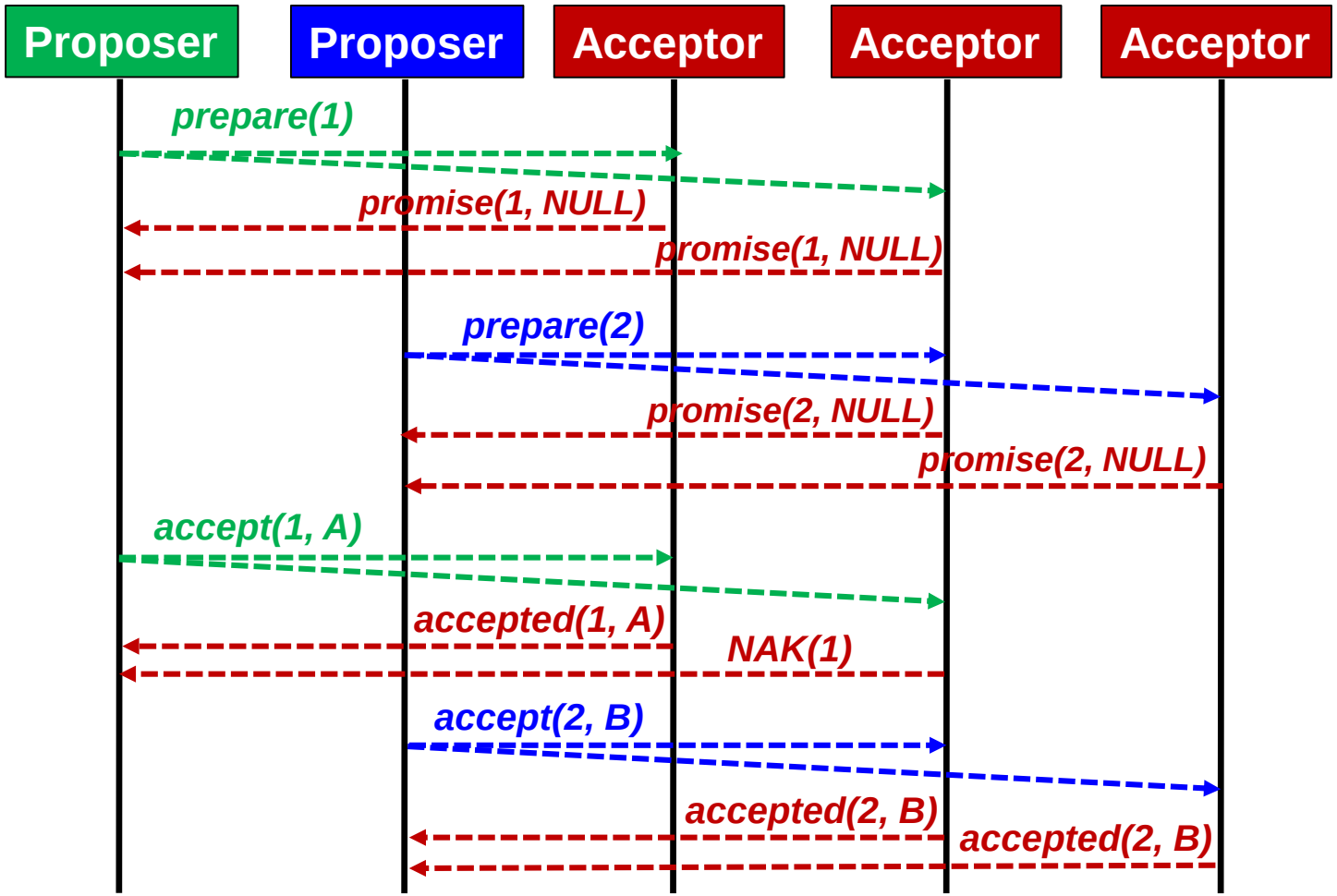


Tetapi, bagaimana jika sebelum blue Proposer mengirimkan pesan acceptnya, Proposer lain (dapat berupa yang green lagi) mensubmit suatu proposal baru dengan nomor urut lebih tinggi?

Contoh

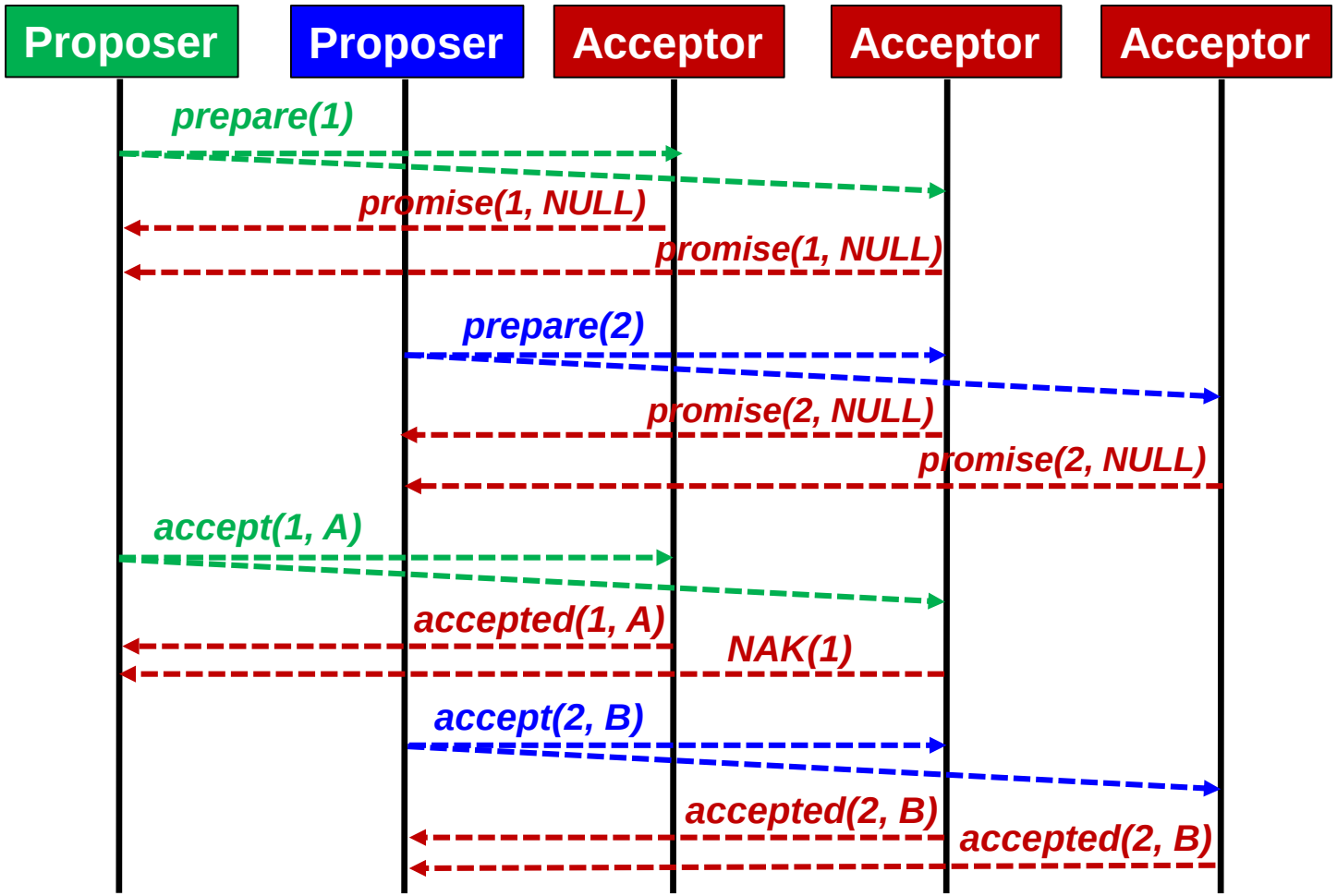

Proposer
Proposer
Acceptor
Acceptor
Acceptor
prepare(1)
promise(1, NULL)
promise(1, NULL)
prepare(2)
promise(2, NULL)
promise(2, NULL)
accept(1, A)
accepted(1, A)
NAK(1)
accept(2, B)
accepted(2, B)
accepted(2, B)
Ronde blue akan gagal juga!

Contoh



Bagaimana jika ini terus terjadi?

Contoh



Paxos tidak akan commit sampai skenarion ini berhenti!

Catatan Mengenai Liveness

- Jika dua Proposers tetap secara bersamaan membagikan proposals dengan nomor urut naik, tak satupun dari keduanya akan berhasil
  - Karenanya, Paxos tidak menjamin *liveness* (yaitu tidak dapat menggaransi bahwa suatu nilai yang diajukan akan dipilih *dalam suatu waktu terbatas*)

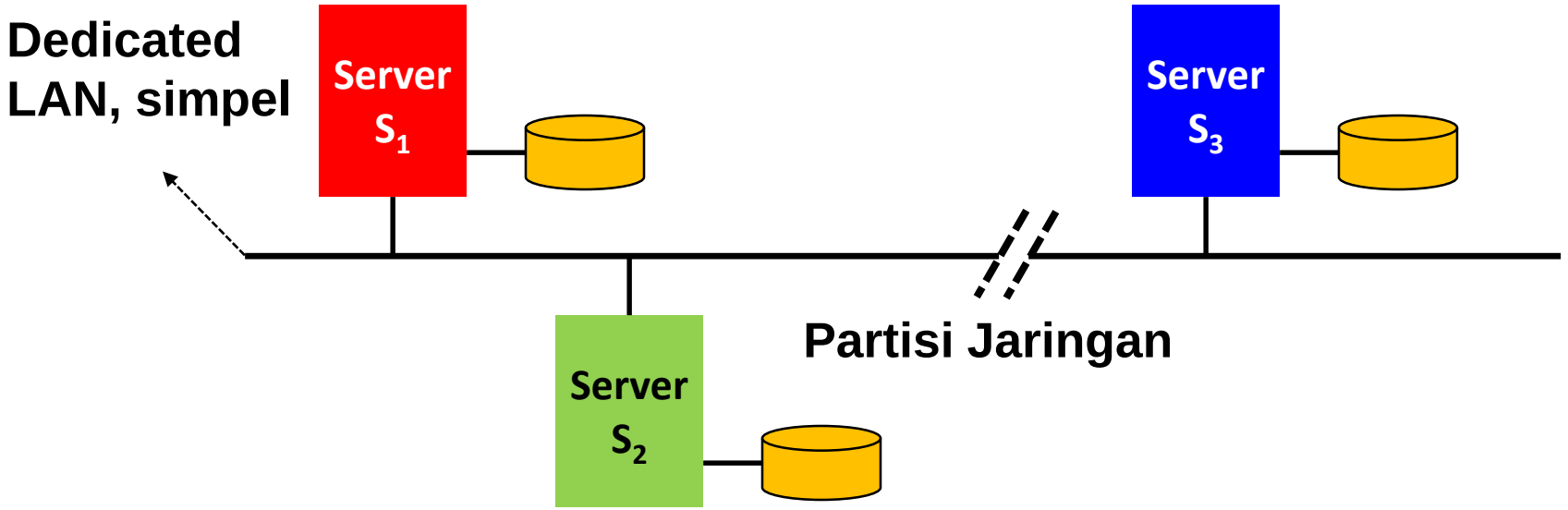
- Adakah cara agar liveness dapat digaransi di dalam Paxos dasar?
  - Jawaban pendek: No
  - Tetapi: Kita dapat menerapkan suatu optimisasi terhadap *potensi mempercepat (bukan menjamin) liveness* dalam kehadiran banyak Proposers bersamaan (*concurrent*)

Catatan Mengenai Liveness

- Untuk memperlancar *liveness*:
  - Suatu *Proposer* pembeda dapat dipilih sebagai the *only* entity untuk mencoba mensubmit proposals
    - Jika Proposer pemdeda ini:
      - Dapat berkomunikasi dengan sukses dengan mayoritas Acceptors
      - *Dan* menggunakan suatu nomor urut yang lebih besar daripada suatu nomor yang telah digunakan
      - Maka ia akan sukses menerbitkan proposal yang dapat diterima, *menganggap cukup sistem (Proposer, Acceptors dan network) bekerja dengan benar.*

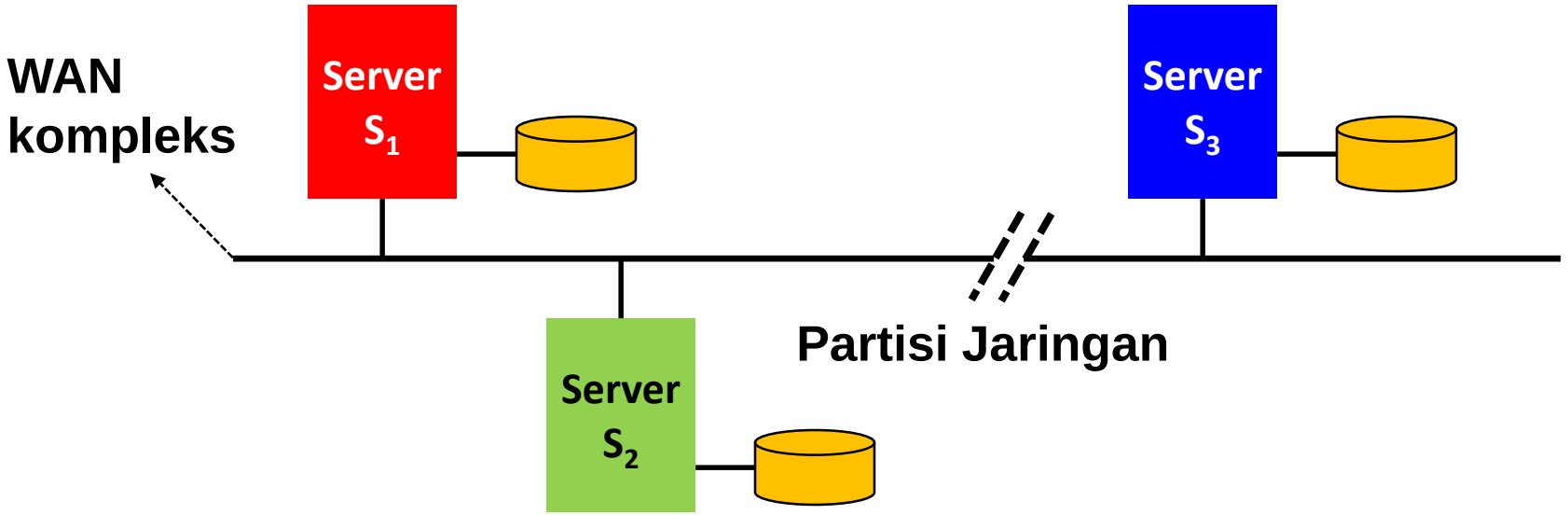
- Jelasnya, liveness menyisakan ketidakmungkinan untuk menjamin dalam waktu terbatas karena suatu komponen di dalam sistem dapat gagal (misal *network partition* dapat muncul)

Partisi Jaringan



- Kegagalan dari suatu media/perangkat komunikasi (misal router) antara dua jaringan dikenal sebagai partisi jaringan (*network partition*)
  - Di atas LAN simpel, proses-proses pada partisi berbeda dapat mengalami putus total
    - Misal, S3 dan S2 mungkin terdiskoneksi penuh, tetapi S1 dan S2 masih dapat berkomunikasi.

Partisi Jaringan



- Kegagalan dari suatu media/perangkat komunikasi (misal router) antara dua jaringan dikenal sebagai partisi jaringan (*network partition*)
  - Di atas jaringan dengan pilihan topologi kompleks dan *independent routing*, konektifitas dapat berupa:
    - *Asymmetric*: S1 dapat berkomunikasi dengan S3, tetapi tidak sebaliknya
    - *Intransitive*: S2 dapat berkomunikasi dengan S1, dan S1 dapat berkomunikasi dengan S3, tetapi S2 tidak dapat berkomunikasi dengan S3

Kemungkinan Gagal Dalam Paxos

- Akankah *network partition* mempengaruhi correctness dari Paxos *(Bukan liveness)*?
  - Tidak, karena adanya mekanisme quorum

- Bagaimana jika suatu Acceptor gagal?
  - Kasus 1: Acceptor bukan anggota dari quorum Proposer
    - Tidak diperlukan pemulihan
  - Kasus 2: Acceptor anggota dari quorum Proposer, tetapi ukuran quorum > mayoritas Acceptors
    - Tidak diperlukan pemulihan.

Kemungkinan Gagal Dalam Paxos

- Akankah *network partition* mempengaruhi correctness dari Paxos?
  - Tidak, karena mekanisme quorum, yang meminta that *at most one partition* akan mampu untuk membangun suatu mayoritas

- Bagaimana jika suatu Acceptor gagal?
  - Kasus 3: Acceptor anggota dari quorum Proposer dan ukuran quorum sama dengan mayoritas Acceptors
    - Sub-Kasus 3.1: Acceptor gagal *setelah* menerima proposal
      - Tidak diperlukan pemulihan, anggap Proposer akan menerima (atau telah menerima) pesan penerimaan (*acceptance*)-nya
    - Sub-Kasus 3.2: Acceptor gagal *sebelum* menerima proposal
      - Kasus terburuk: Quorum dan ronde baru dapat didirikan.

Kemungkinan Gagal Dalam Paxos

- Bagaimana jika suatu Proposer gagal?
  - Kasus 1: Proposer gagal *setelah* mengusulkan suatu value, tetapi *sebelum* suatu konsensus dicapai
    - Proposer baru dapat mengambil alih
  - Kasus 2: Proposer gagal *setelah* suatu konsensus dicapai, tetapi *sebelum* ia mengetahui tentangnya
    - Salah satu: kegagalan itu terdeteksi dan ronde baru diluncurkan
    - Atau, ia pulih dan memulai babak baru sendiri
  - Kasus 3: Proposer gagal *setelah* suatu konsensus dicapai dan *setelah* ia mengetahui tentangnya (*but before letting the Learner knowing*)
    - Salah satu: kegagalan itu terdeteksi dan ronde baru diluncurkan
    - Atau, itu pulih dan belajar lagi dari penyimpanan stabil bahwa ia telah berhasil dalam penawarannya

Pertemuan Selanjutnya...

- Topik konsep terakhir:
  - Toleransi kegagalan (*Fault-tolerance*)