IF3230
Sistem Paralel dan Terdistribusi
Model Sistem

Achmad Imam Kistijantoro (imam@staff.stei.itb.ac.id)
Robithoh Annur (robithoh@stafff.stei.itb.ac.id)
Andreas Bara Timur (bara@staff.stei.itb.ac.id)
Maret 2023
Informatika – STEI – ITB

Model Sistem Terdistribusi

- Model system: asumsi yang kita gunakan terhadap bagaimana perilaku node dan jaringan

- Sistem terdistribusi dapat dilihat sebagai sistem komputer secara umum
  - Storage: penyimpanan data
  - Komputasi: proses

- Lokasi storage dan komputasi dapat terpisah

Model Sistem

- Program pada sistem terdistribusi
  - Berjalan konkuren pada node yang terdistribusi
  - Terhubung melalui jaringan => non determinism dan loss
  - Tidak memiliki shared memory dan shared clock

- Implikasi
  - Setiap node berjalan konkuren
  - Knowledge bersifat lokal, informasi tentang node lain/global mungkin tidak up to date
  - Node dapat gagal dan recover independen terhadap node lainnya
  - Message dapat tertunda atau hilang
  - Clock tidak tersinkron dengan node lain

Model sistem

- Model sistem menggambarkan sekumpulan asumsi terhadap lingkungan terdistribusi dan fasilitas yang tersedia
- Mencakup
  - Apa kapabilitas sebuah node, dan bagaimana model kegagalannya
  - Apa kapabilitas jalur komunikasi dan bagaimana model kegagalannya
  - Sifat sistem keseluruhan, seperti model waktu dan order

- Robust system: menggunakan asumsi yang paling lemah. Algoritma yang ditulis untuk sistem ini akan tolerant terhadap berbagai kesalahan yang mungkin terjadi
- Namun menggunakan asumsi yang lebih kuat (strong) akan memudahkan implementasi

Model sistem: Node

- Pemodelan node: komputer, dengan kapabilitas komputasi dan storage
  - Dapat menjalankan program
  - Mampu menulis ke volatile memory (data hilang saat crash) dan stable state (tidak hilang saat crash)
  - Memiliki local clock (mungkin tidak akurat)

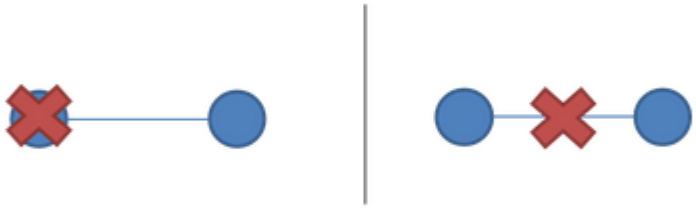
- Node menjalankan algoritma/program secara deterministik => komputasi lokal, state setelah komputasi dan message yang dikirim ditentukan berdasarkan pesan yang diterima dan state lokal saat pesan diterima

Model sistem: Node

- Model kegagalan:
  - **Crash failure:** node hanya dapat gagal dengan crashing
  - **Crash-recovery**: node hanya dapat gagal dengan crashing, yaitu node berhenti bekerja, dan dapat recover
  - **Byzantine:** node dapat gagal dengan memberikan perilaku yang beragam

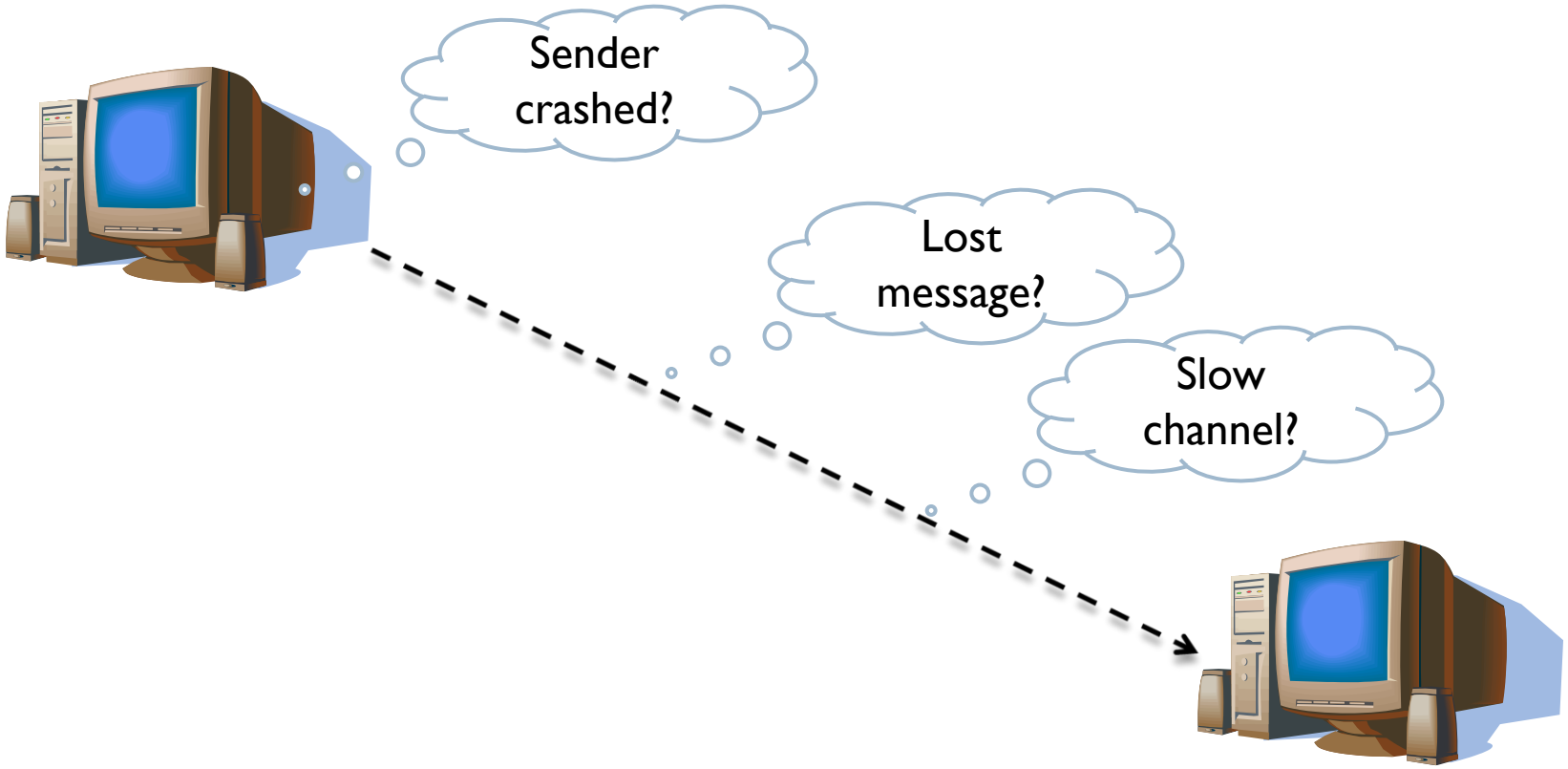
Model sistem: komunikasi

- Komunikasi menghubungkan antar node.
- Umumnya dimodelkan sebagai jalur yang menghubungkan antar 2 node duplex, FIFO order, dan message dapat hilang (unreliable network)
- Reliable network: message tidak hilang dan tidak pernah tertunda secara indefinite
- Network partition: terjadi jika ada kegagalan network, namun node tetap operasional



Model sistem: Timing & order

- Synchronous system:
  - proses2 pada node berjalan sinkron, ada batas waktu maks pengiriman pesan antar node, setiap proses memiliki clock yang akurat
- Asynchronous system:
  - Tidak ada asumsi timing. Proses berjalan pada rate yang independen, tidak ada batas waktu pengiriman pesan, clock local tidak akurat


Sender crashed?
Lost message?
Slow channel?

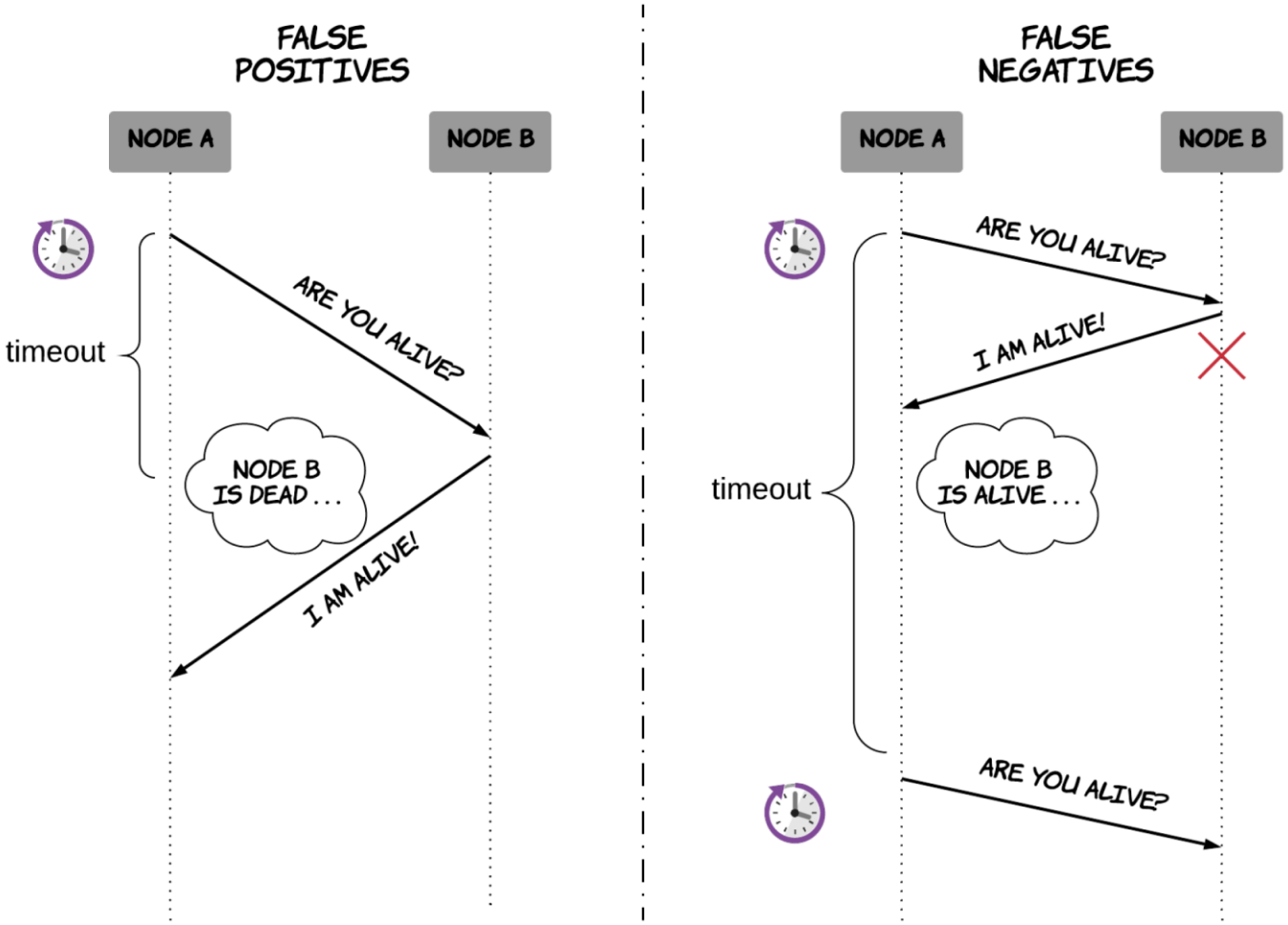


Figure 1.3: Trade-offs in failure detection

Dimos Raptis. Distributed Systems for Practitioners. 2020

Model asynchronous

- Pada model asynchronous, timing ditentukan dengan menggunakan logical clock
- Setiap proses menyimpan nilai integer yang disebut logical clock l_p dengan nilai awal 0
- Setiap ada event pada proses p, nilai l_p ditambah
- Ketika proses mengirim pesan ke proses lain, pesan diberi timestamp sesuai dengan nilai l_p saat pesan dikirim
- Ketika proses menerima pesan dengan timestamp l_m dari proses lain, l_p dinaikkan dengan cara $l_p = \max(l_p , l_m) + 1$

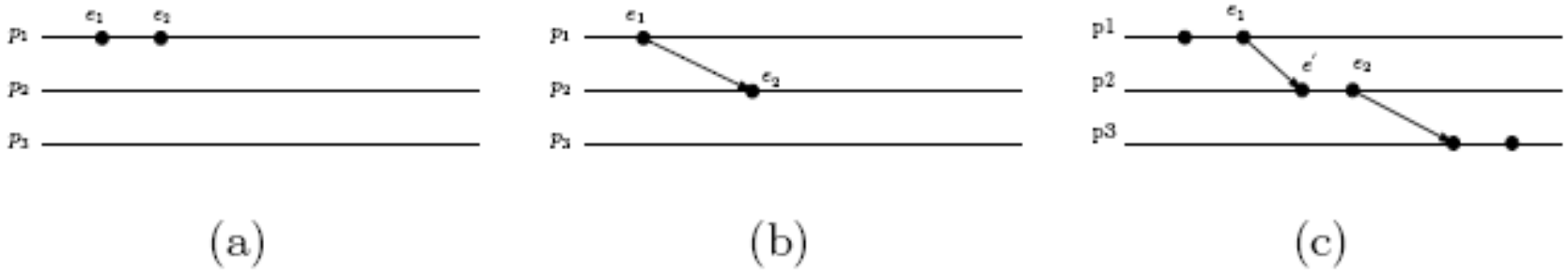


Fig. 2.4: The *happened-before* relation

- Logical clock menjamin relasi happened-before antar event yang berkaitan
- e_1 , e_2 terjadi pada proses yang sama dan e_1 terjadi sebelum e_2
- e_1 adalah event pengiriman pesan m, e_2 adalah event penerimaan pesan m: e_1 terjadi sebelum e_2
- Ada event e dimana $e_1 \rightarrow e$ and $e \rightarrow e_2$
- Jika $e_1 \rightarrow e_2$, maka $t(e_1) < t(e_2)$

Model synchronous

- proses2 pada node berjalan sinkron, ada batas waktu maks pengiriman pesan antar node, setiap proses memiliki clock yang akurat

- Implikasi
  - Dapat dilakukan pendeteksian kegagalan dengan waktu
  - Pengukuran delay transit
  - Koordinasi berdasarkan waktu
  - Worst case performance
  - Synchronized clock

Partial synchrony

- Umumnya, sistem terdistribusi terlihat memiliki perilaku sinkron
- Ada batas waktu delay maksimum yang dapat digunakan, yang berlaku *most of the time*
- Pada saat tertentu (e.g. saat ada masalah jaringan), baru menjadi asinkron
- Model yang mendeskripsikan perilaku ini disebut sebagai partial synchrony
- Partial synchrony mendefinisikan asumsi sinkron akan terwujud *eventually*, namun tidak diketahui

Correctness in Distributed Systems

- Correctness pada sebuah sistem dinyatakan sebagai properties: pernyataan yang menyatakan perilaku sistem.

- Properties dapat digolongkan menjadi 2 jenis:
  - Safety properties
  - Liveness properties

Properties of distributed systems

- safety properties
  - properties that are never be violated by correct process
  - if violated at time t, the properties are never be satisfied again after t
  - something bad should never happen
  - easy thing: do nothing

properties of distributed systems

- liveness properties
  - properties that will be satisfied eventually
  - for any time t, there is some hope that the property can be satisfied at some time $t \geq t$.
  - eventually, something good happens

- challenge: to guarantee both liveness and safety

Contoh: Consensus Problem

- Problem mengambil keputusan/kesepakatan

  - **Agreement**: setiap proses yang benar/correct harus sepakat terhadap nilai/keputusan yang sama
  - **Integrity**: setiap proses yang benar/correct mengambil keputusan terhadap 1 nilai, dan nilai yang menjadi keputusan harus yang sebelumnya pernah diusulkan oleh sebuah proses
  - **Termination**: semua proses suatu saat akan mencapai keputusan/kesepakatan
  - **Validity**: jika semua proses yang benar/correct mengajukan sebuah nilai V yang sama, maka semua proses yang benar/correct akan mensepakati V

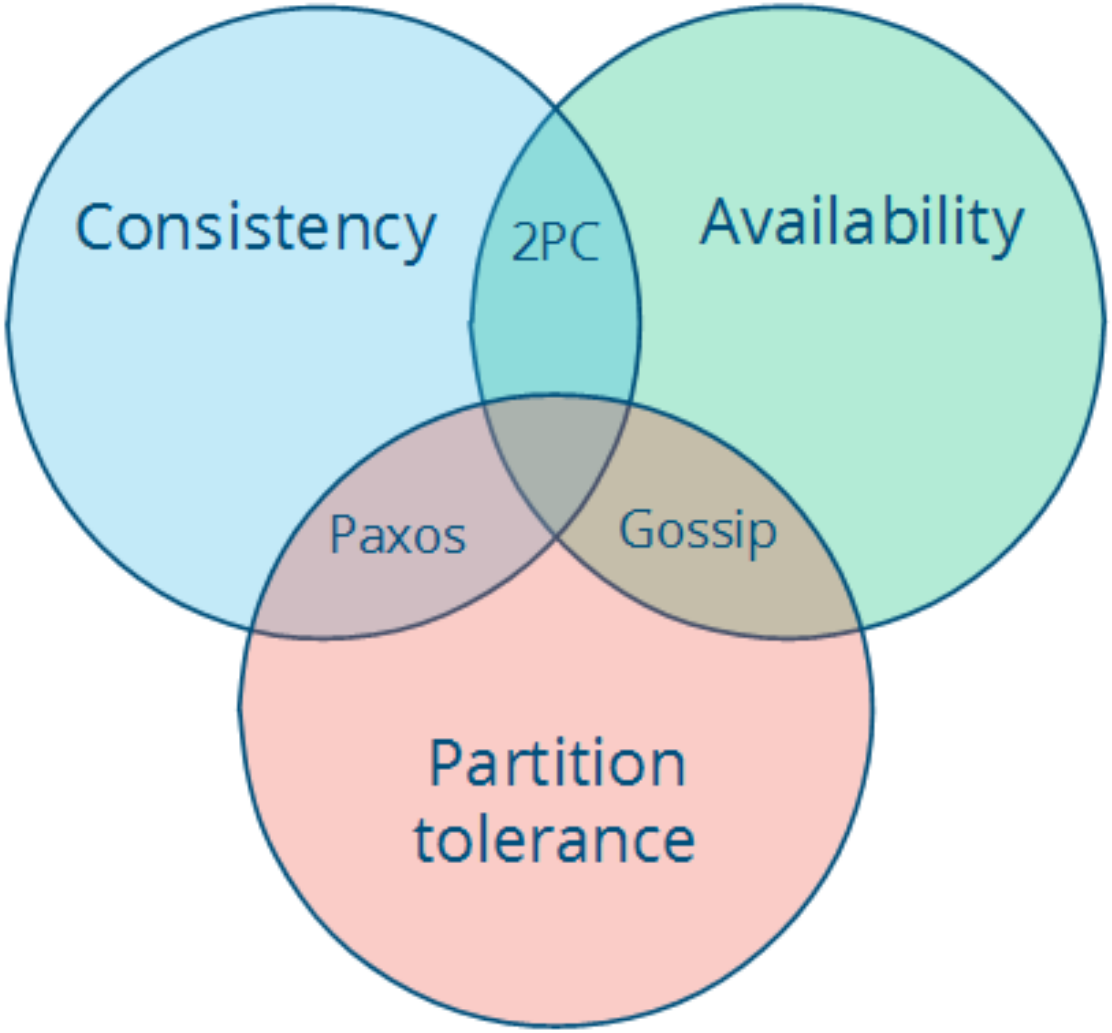
FLP Impossibility

- Fischer, Lynch & Paterson (1985) menyatakan bahwa tidak ada algoritma yang dapat selalu mencapai konsensus pada waktu yang tertentu pada sistem asynchronous
  - Asumsi: network reliable, nodes failed by crashing, asynchronous

- Algoritma untuk penyelesaian problem konsensus pada asynchronous system harus mengorbankan salah satu property: liveness atau safety

Teorema CAP

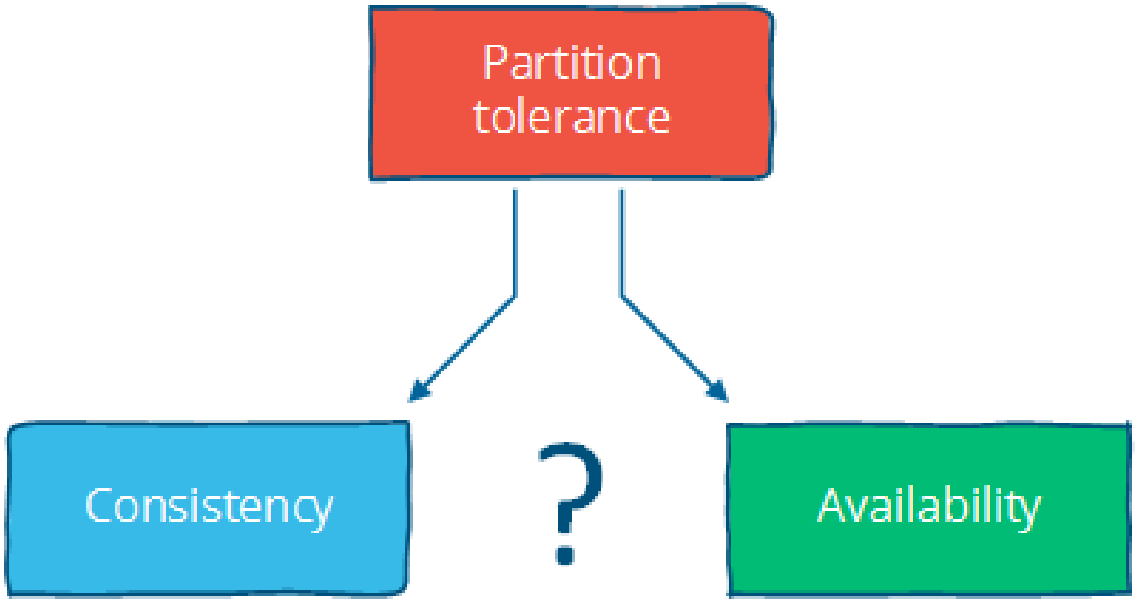
- Diajukan oleh Eric Brewer (Fox & Brewer, 1999).
- Desain sistem hanya dapat memiliki 2 dari 3 property:
  - Consistency: setiap node dapat melihat data yang sama pada saat yang sama. Setiap read request akan mengembalikan hasil dari write request yang terakhir dilakukan
  - Availability: kegagalan node tidak menghalangi node lain yang tidak gagal untuk tetap menyediakan layanan. Setiap request akan menghasilkan respons yang non error (*tidak harus hasil yang paling up to date*)
  - Partition tolerance: sistem tetap dapat beroperasi meskipun terdapat network partition


Consistency
2PC
Availability
Paxos
Gossip
Partition
tolerance

- Jenis sistem
- CA (consistency + availability) contoh: two-phase commit
- CP (consistency + partition tolerance), contoh: majority quorum protocols, paxos
- AP (availability + partition tolerance), contoh: mekanisme yang menyediakan conflict resolution, e.g. Dynamo

Teorema CAP

- Pada WAN/Internet, partition tidak bisa dihindari



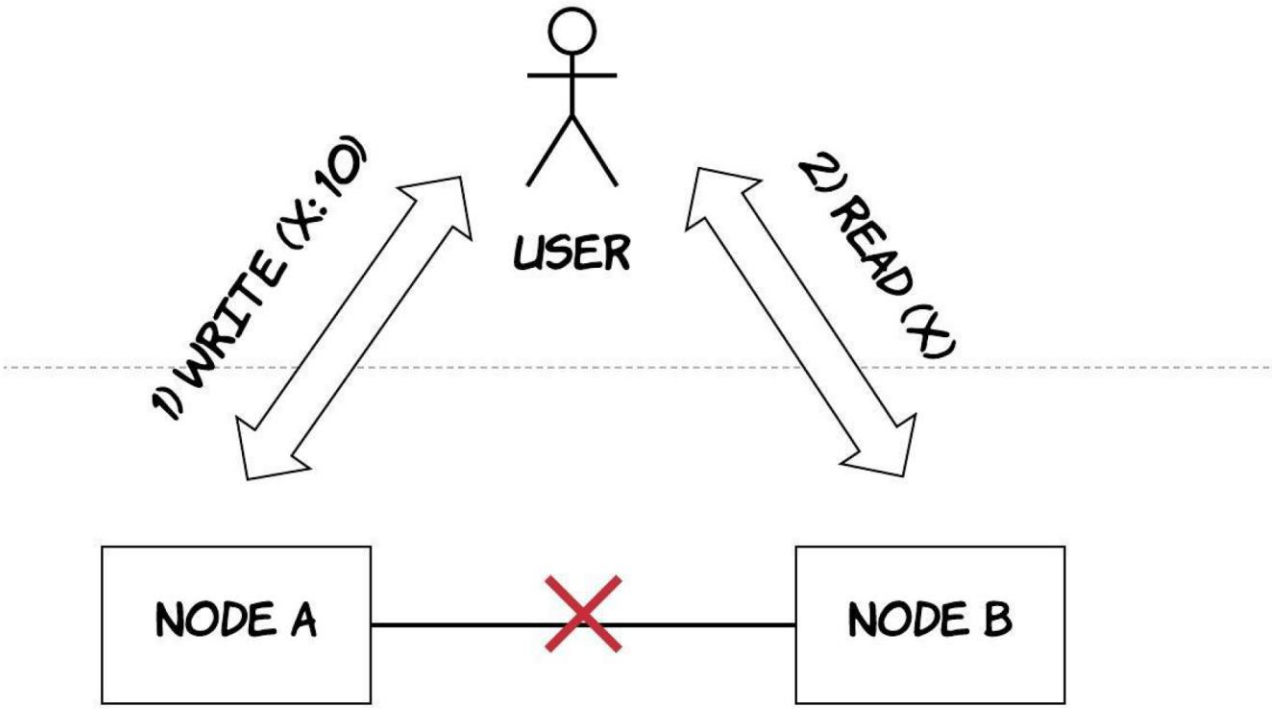


Figure 2.6: Handling a network partition in a distributed system

Teorema CAP

- Banyak desain sistem yang tidak mempertimbangkan network partition, misal pada distributed database systems
- Jika ada network partition, perlu memilih antara consistency dan availability
- Sistem dengan strong consistency, harus siap dengan non availability saat partition
- Sistem yang ingin available, harus siap dengan weaker consistency

Teorema CAP

- Strong consistency tradeoff dengan performance
  - Strong consistency memerlukan koordinasi antar node pada setiap operasi
- Consistency model:
  - kontrak antar programmer dengan sistem, jaminan apa yang diberikan oleh sistem, jika programmer mengikuti aturan tertentu, sehingga sistem menjadi predictable
- jika kita bisa jalan dengan weaker consistency model, latency bisa dikurangi pada kondisi normal, dan sistem bisa tetap available saat terjadi network partition

Consistency Model

- Mendefinisikan set of execution histories yang valid untuk sebuah consistency model tertentu
- History: koleksi operasi, beserta struktur konkurensinya (urutan eksekusi operasi, dan kemungkinan berjalan konkuren)
- Jika sebuah history yang memenuhi consistency model A berarti juga memenuhi consistency model B, maka A subset dari B, dan ***A stronger than B***
- Contoh: *linearizable consistency is stronger than sequential consistency model*

Consistency model

- Strong consistency model
  - Linearizable consistency: setiap operasi terlihat dieksekusi secara atomic dan *instantaneous*, dan sesuai dengan real-time ordering
  - Sequential consistency: setiap operasi terlihat dieksekusi secara atomik, dengan order yang konsisten di semua client
- Weak consistency model
  - Client-centric consistency model: konsistensi terkait dengan sesi client. Misal: client tidak pernah melihat data yang dia tulis hilang atau kembali ke versi lama
  - Causal consistency: konsistensi hanya dijamin pada operasi yang berhubungan
  - Eventual consistency: konsistensi akan terjadi eventually

Linearizable consistency vs sequential consistency

- Eksekusi yang di bawah tidak linearizable, tapi sequential
- Linearity mengharuskan eksekusi sinkron
- Pada sequential, order dapat ditentukan (tidak harus mengikuti urutan real event)

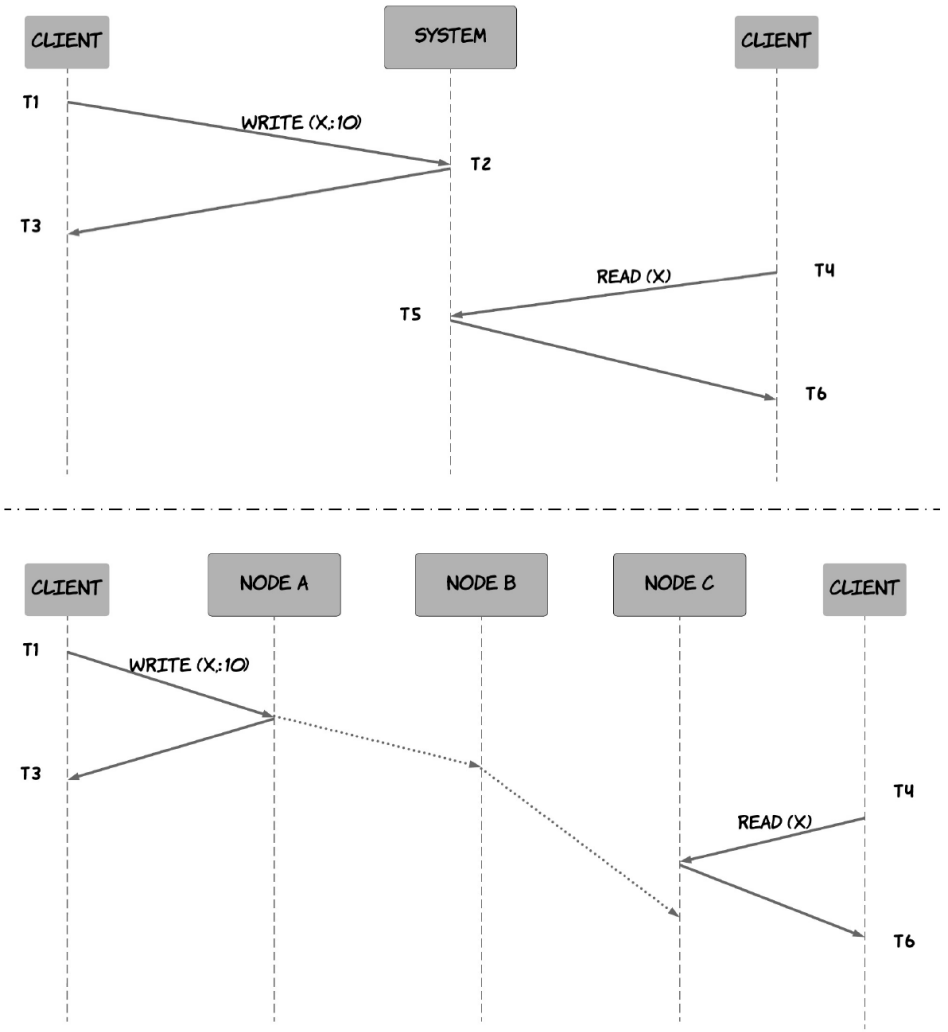
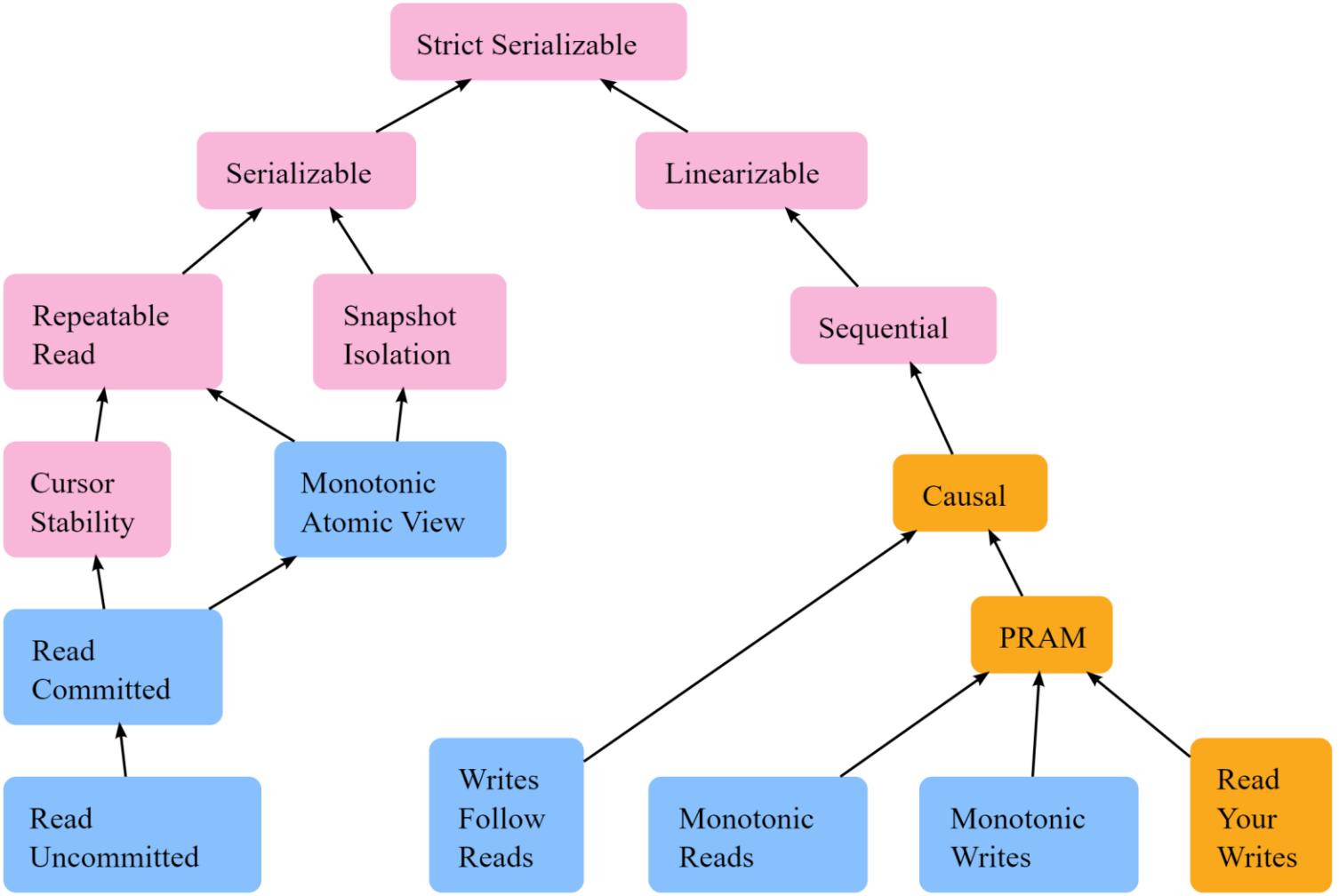


Figure 2.8: Why linearizability is not obvious in a distributed system

Consistency Model



https://jepsen.io/consistency

Rekap CAP Model

- CAP: pilih 2 dari 3, bukan strict
- Consistency: spektrum dari strongest hingga weakest
- Availability: short latency vs long latency

Sumber

- Mikito Takada. Distributed Systems for fun and profits. http://book.mixu.net/distsys/index.html
- R. Guerraoui and L. Rodrigues. Introduction to Reliable Distributed Programming. Springer-Verlag 2006
- Eric Brewer. CAP Twelve Years Later: How the "Rules" Have Changed. http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- Dimos Raptis. Distributed Systems for Practitioners. 2020.