Sistem Terdistribusi

TIK-604

Caching

Kuliah 09 & 10: 22 s.d 24 & 29, 30 April, 01 Mei 2019

Husni

Hari ini…

- Kuliah terakhir:
  - Message Passing Interface: MPI
- Pembahasan hari ini:
  - Latency dan Bandwidth
  - Pengantar Caching
  - Pendekatan Konsisten Cache
  - Kebijakan penggantian cache
- Pengumuman:
  - …

Latency dan Bandwidth

- Latency dan bandwidth secara parsial berkaitan erat
  - jika bandwidth penuh jenuh
    - Kemacetan terjadi dan latency meningkat
  - Jika bandwidth tidak di puncak
    - Kemacetan tidak akan terjadi, tetapi latensi TIDAK akan berkurang
    - Contoh: Mengirim suatu bit pada media 50Mbps yang tidak macet tidak akan lebih cepat daripada mengirim 32KB

- Bandwidth dapat dengan mudah ditingkatkan, tetapi pada dasarnya sulit untuk mengurangi latensi!

Latency dan Bandwidth

- Pada kenyataannya, latensi adalah pembunuhnya; bukan bandwidth

- Bandwidth dapat ditingkatkan melalui redundansi
  - Misal: Lebih banyak pipa, pipa lebih gemuk, lebih banyak jalur di jalan raya, lebih banyak pegawai di toko, dll.,
  - Membutuhkan uang, tetapi pada dasarnya tidak sulit

- Latensi jauh lebih sulit untuk diperbaiki
  - Biasanya, itu membutuhkan perubahan struktural yang dalam
  - Misal: Memperpendek jarak, mengurangi panjang jalur, dll.,
  - Bagaimana kita bisa mengurangi latensi dalam sistem terdistribusi?

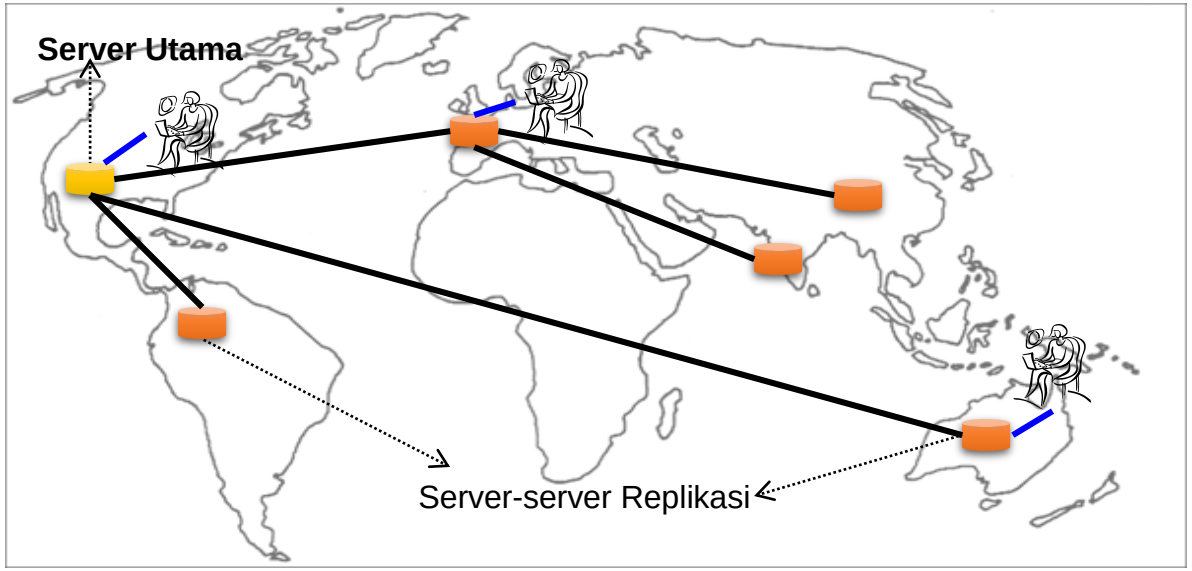
Replikasi dan Caching

- Salah satu cara untuk mengurangi latensi adalah dengan menggunakan replikasi dan *caching*

- Apa itu replikasi?
  - Replikasi adalah proses mempertahankan beberapa salinan data di berbagai lokasi
  - Setelah itu, klien dapat mengakses salinan yang direplikasi yang terdekat dengannya, berpotensi mengurangi latensi
- Apa itu *caching*?
  - Caching adalah jenis khusus replikasi yang dikendalikan klien
  - Secara khusus, replikasi sisi klien disebut sebagai caching

Replikasi dan Caching

- Aplikasi contoh
  - Caching halaman web pada client browser
  - Caching IP addresses pada clients dan DNS Name Servers
  - Replikasi dalam *Content Delivery Network* (CDNs)
    - Konten yang umum diakses, seperti perangkat lunak dan media *streaming*, di-cache di berbagai lokasi jaringan



Dilema

- CDN mengatasi dilema besar
  - Bisnis ingin mengetahui setiap klik dan penekanan tombol Anda
    - Ini untuk mempertahankan pengetahuan mendalam mengenai klien
  - Caching sisi klien menyembunyikan pengetahuan ini dari server
  - Jadi, server menandai halaman sebagai "tidak dapat disimpan"
    - Ini sering bohong, karena kontennya sebenarnya bisa disimpan dalam cache
  - Namun, kurangnya caching menyakitkan latensi dan pengalaman pengguna!!

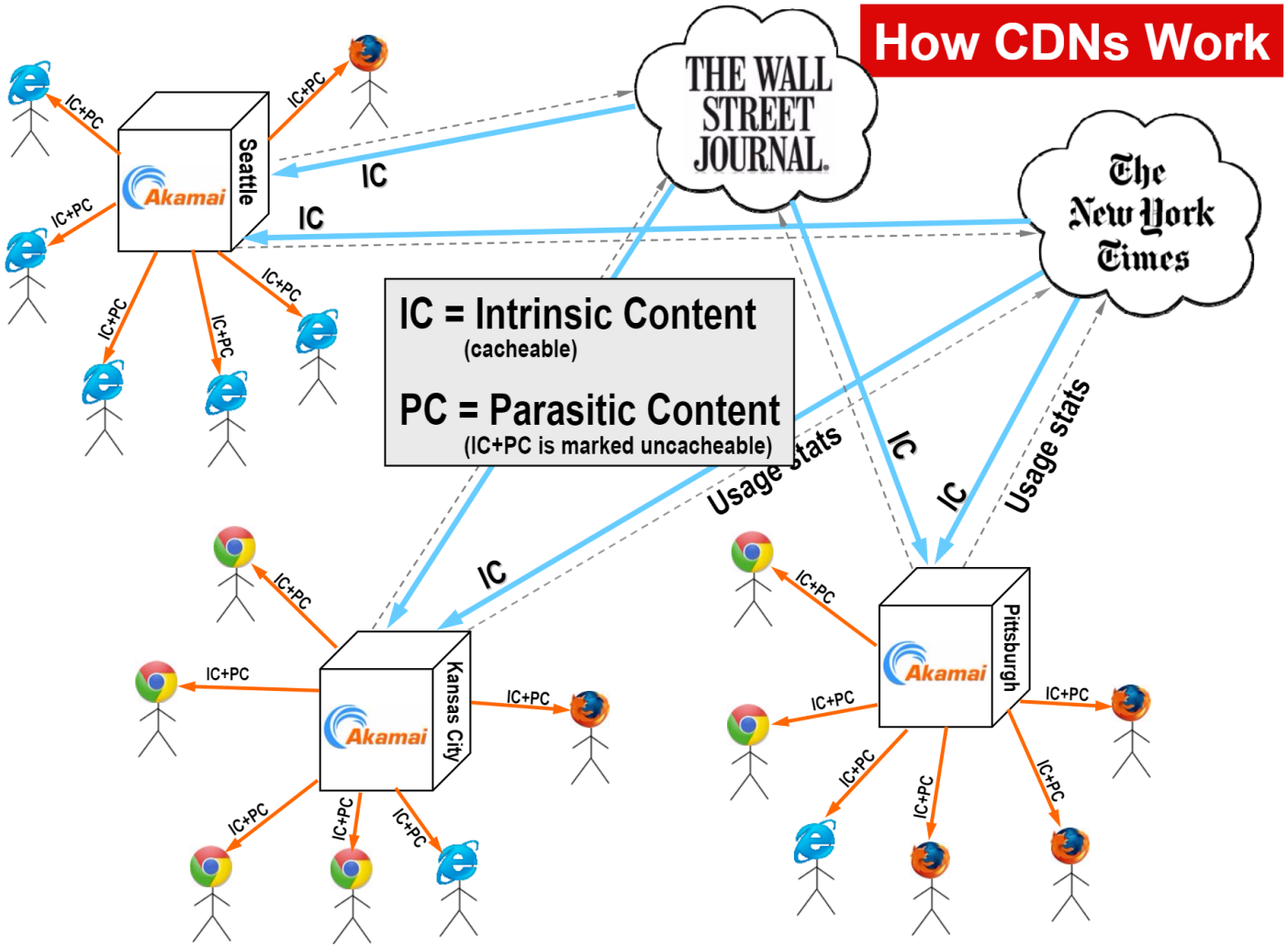
Mampukah bisnis mendapat manfaat dari caching tanpa melepaskan kendali?

CDN: Solusi Untuk Dilema Ini

- Situs caching pihak ketiga (atau *providers*) menyediakan *hosting services*, yang dipercayai oleh perusahaan
  - Provider memiliki koleksi server lintas Internet
  - Biasanya, hosting service-nya dapat secara dinamis mereplikasi file-file pada server-server berbeda
    - Misal: Berdasarkan pada popularitas dari suatu file di dalam suatu daerah

- Contoh:
  - Akamai (yang memelopori CDN pada akhir 90-an)
  - Amazon CloudFront CDN
  - Windows Azure CDN

CDN: Solusi Untuk Dilema Ini

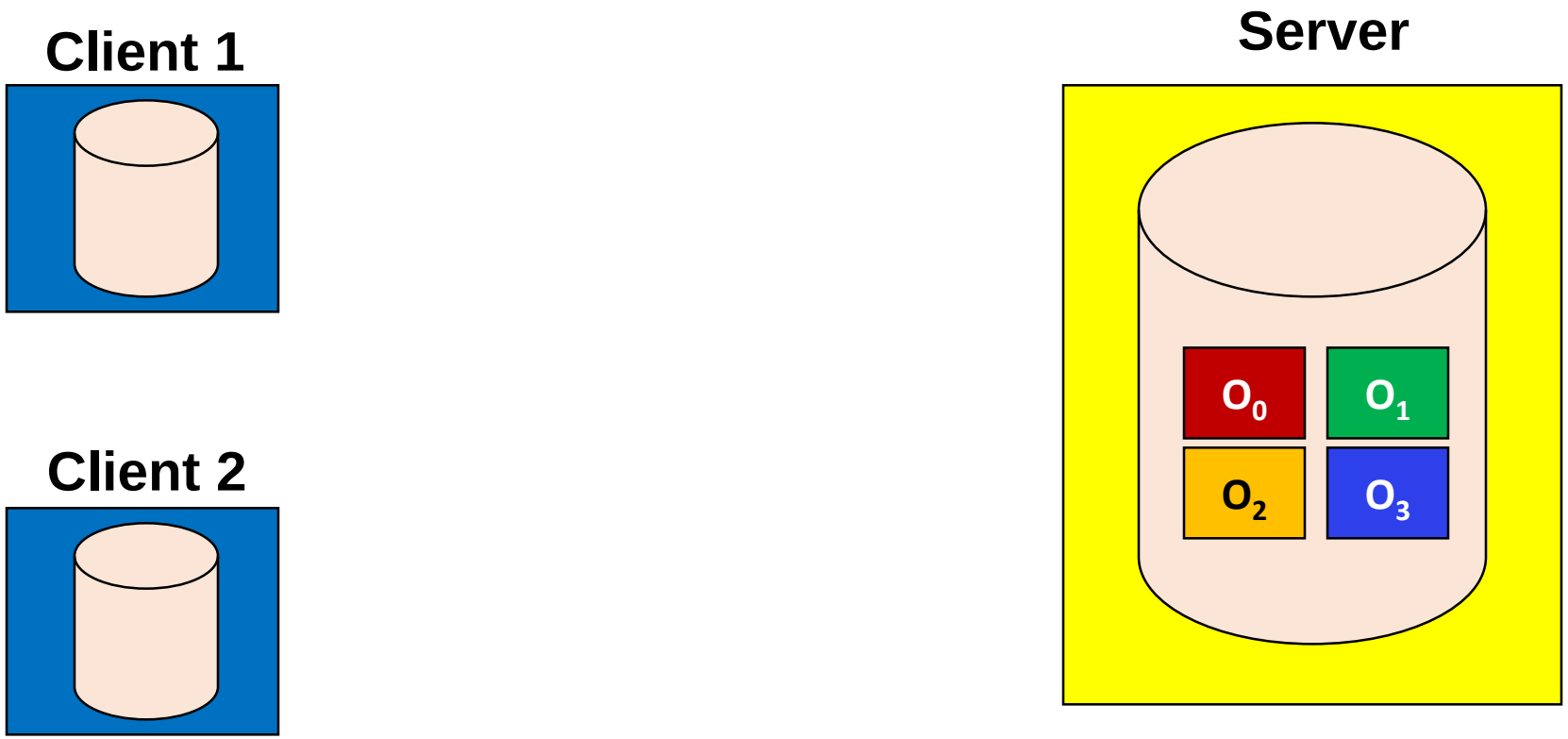


Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika client melaksanakan *non-overlapping* requests ke obyek data?
  - Ya, melalui client-side caching

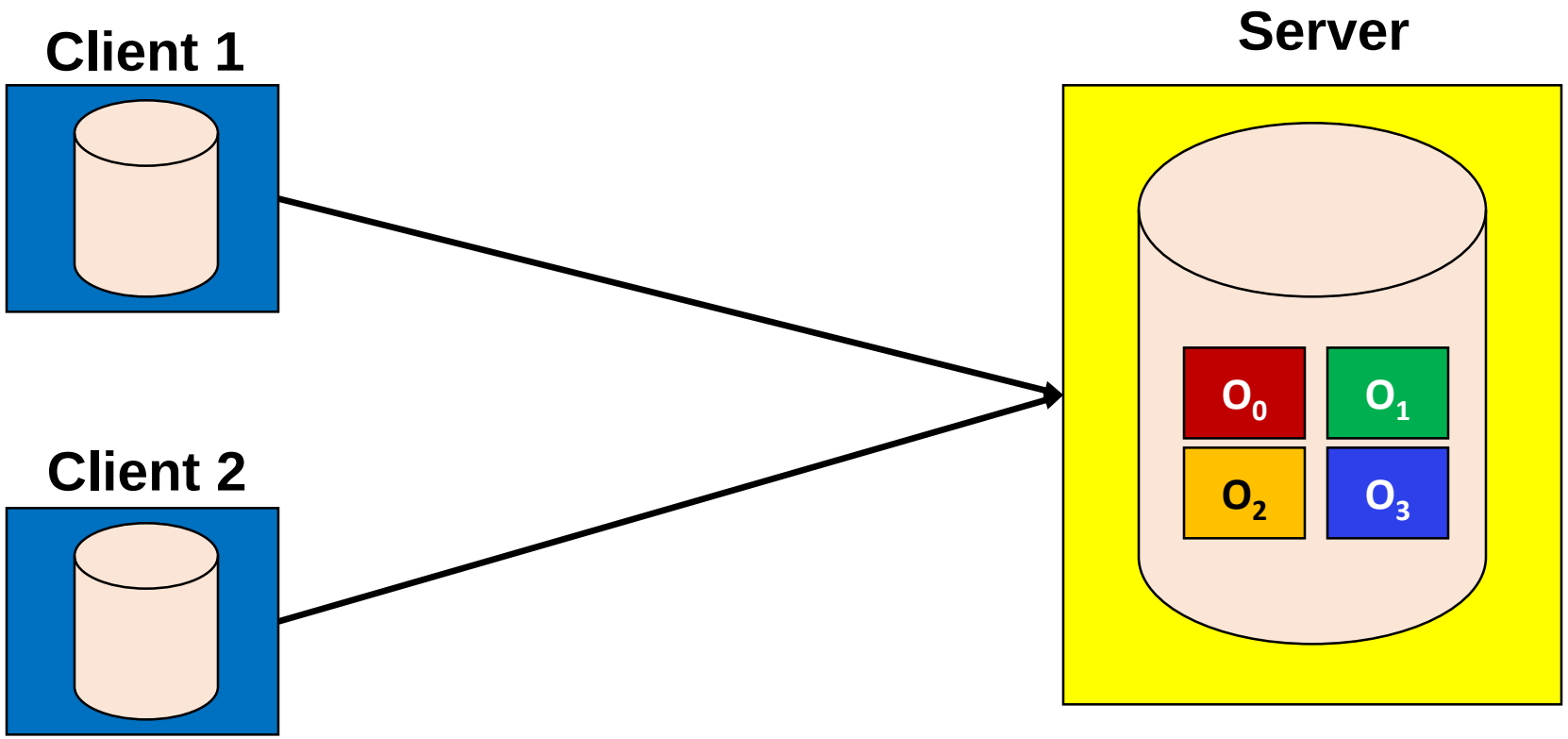
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika client melaksanakan *non-overlapping* requests ke obyek data?
  - Ya, melalui client-side caching



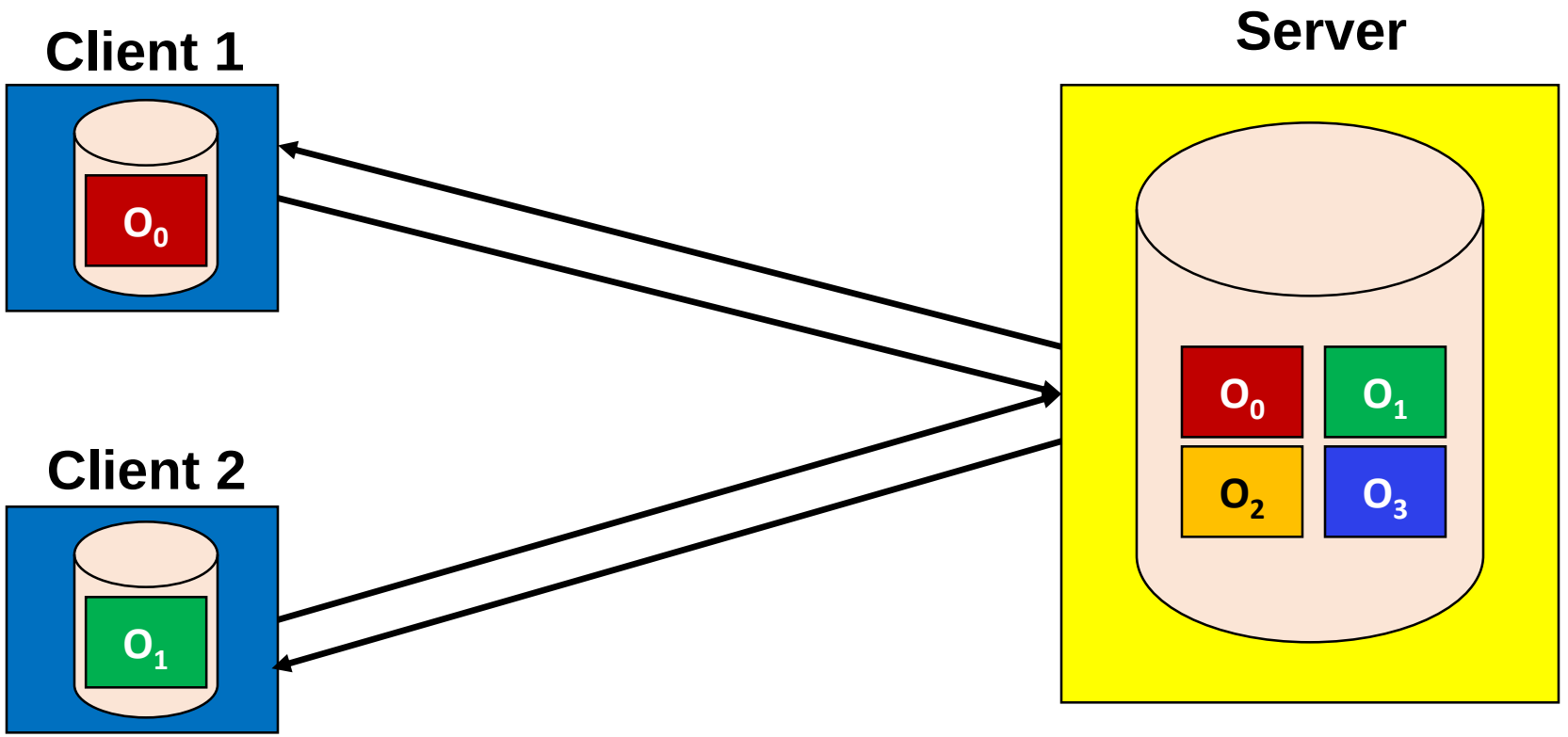
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika client melaksanakan *non-overlapping* requests ke obyek data?
  - Ya, melalui client-side caching



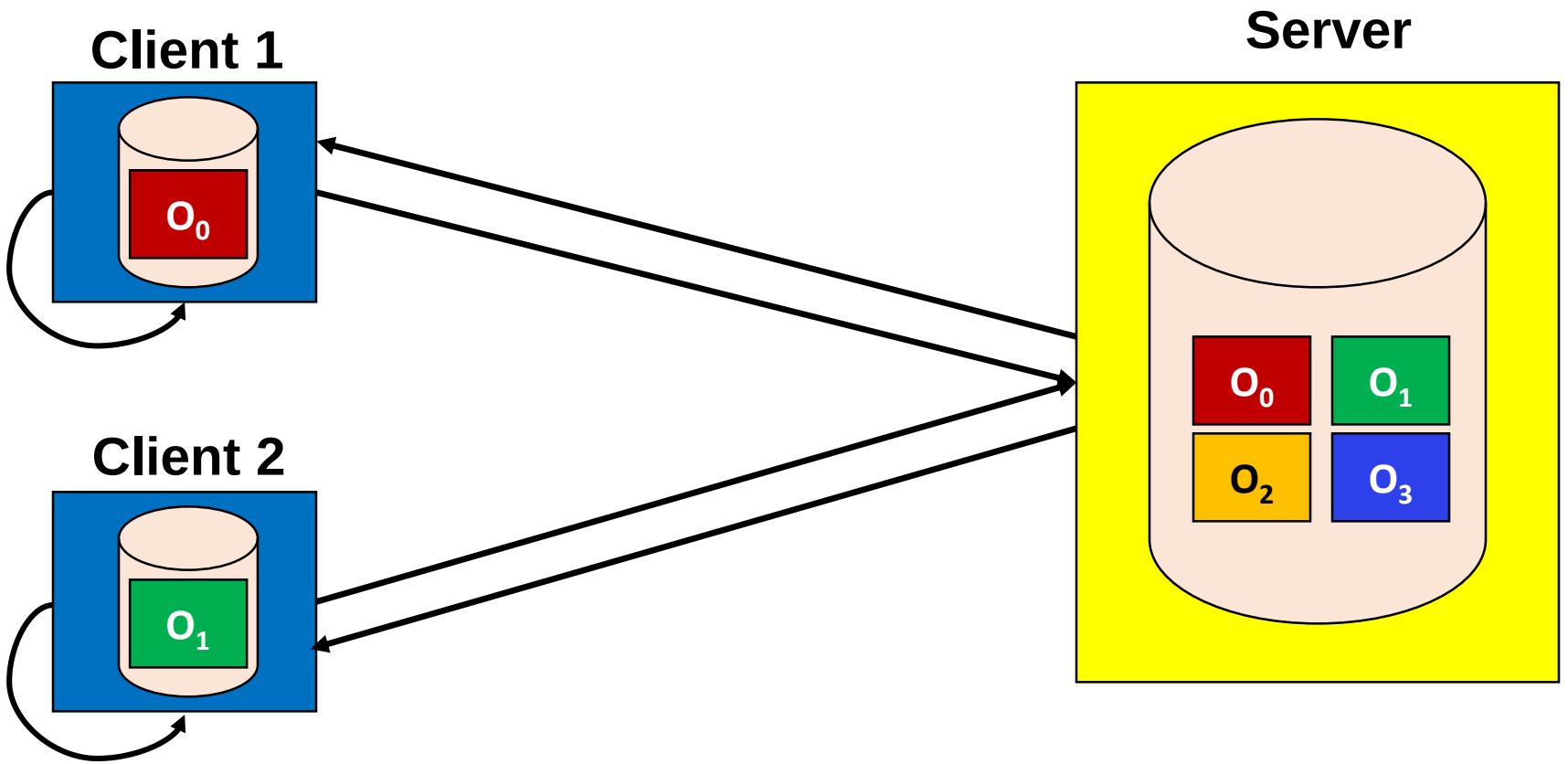
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika client melaksanakan *non-overlapping* requests ke obyek data?
  - Ya, melalui client-side caching



Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika client melaksanakan *non-overlapping* requests ke obyek data?
  - Ya, melalui client-side caching

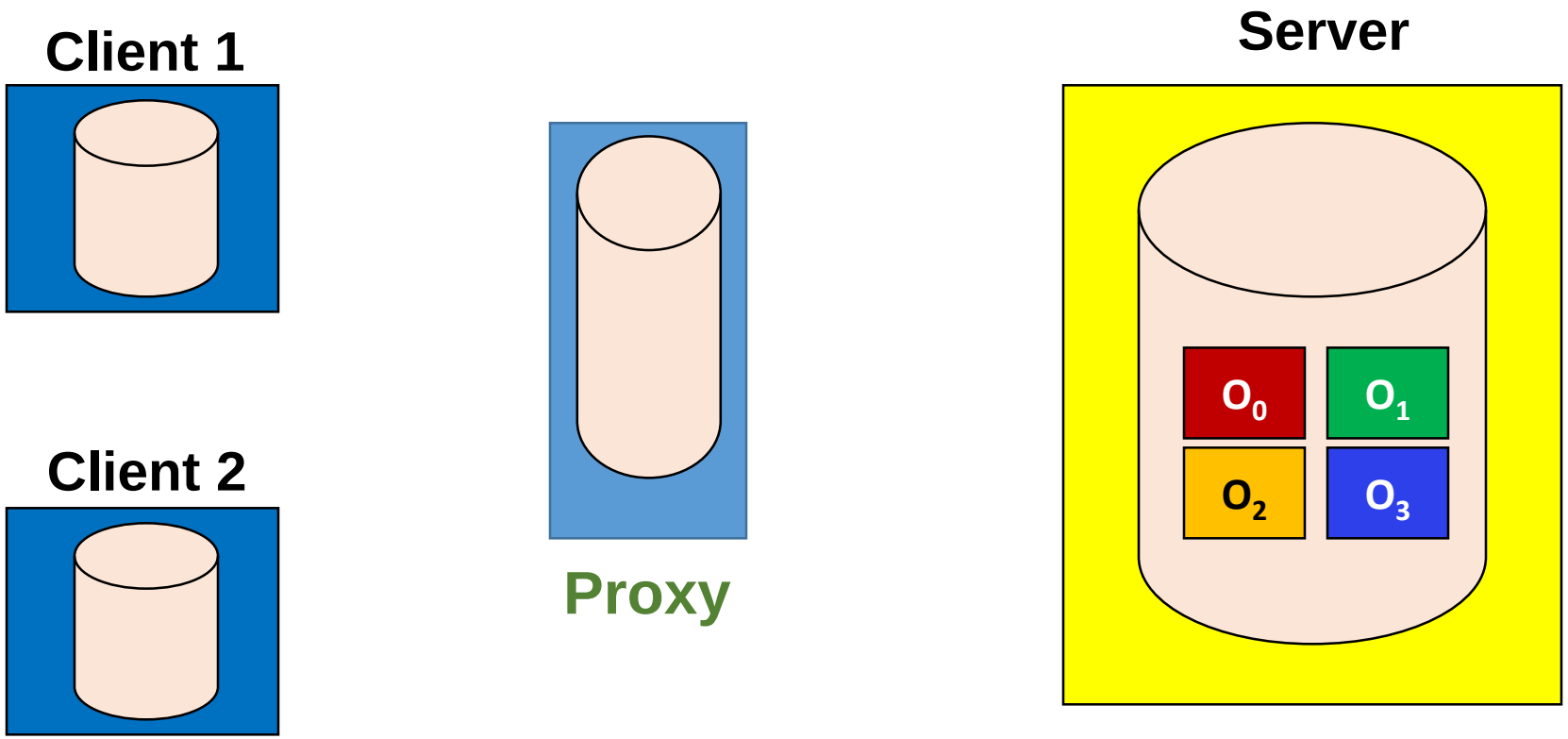


Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication

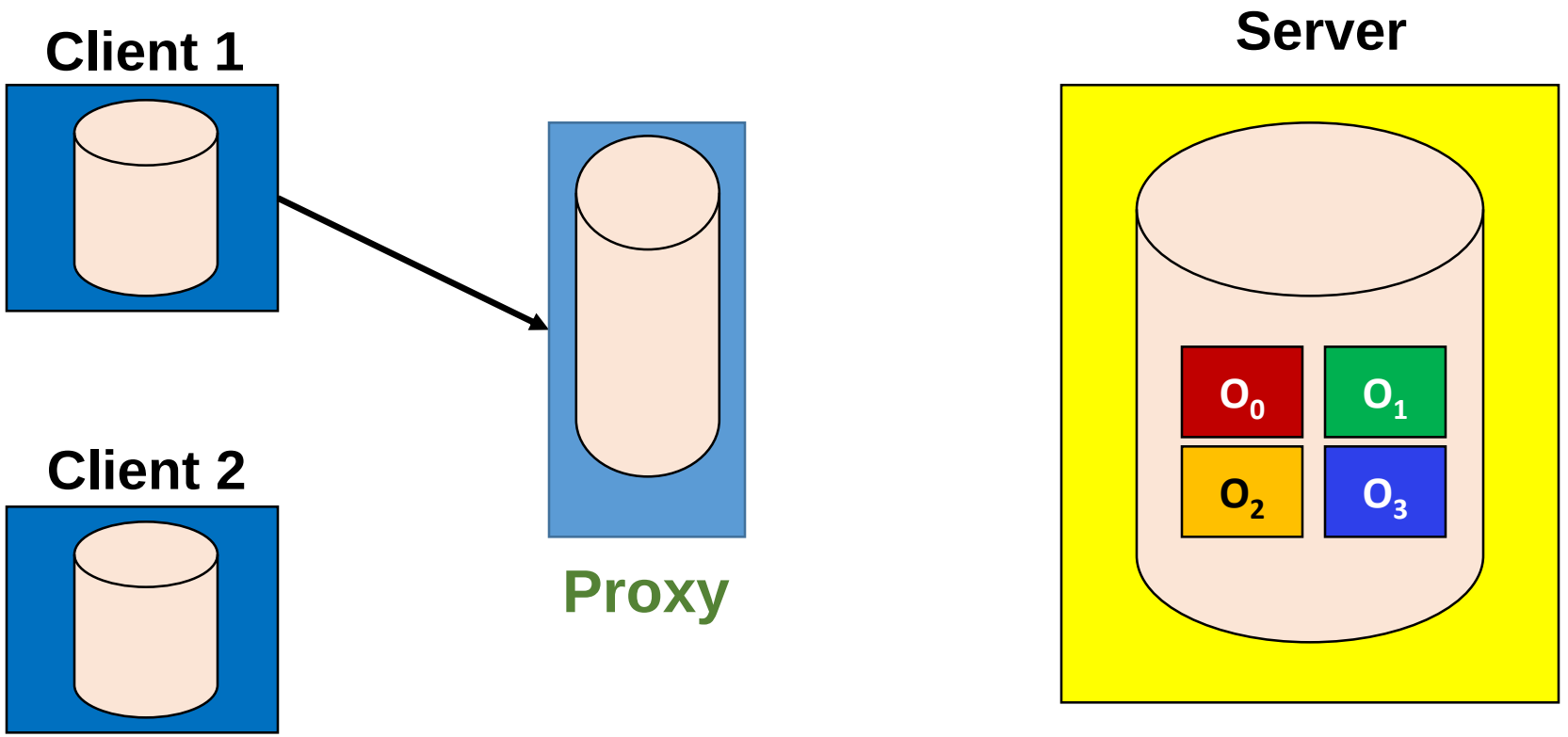
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



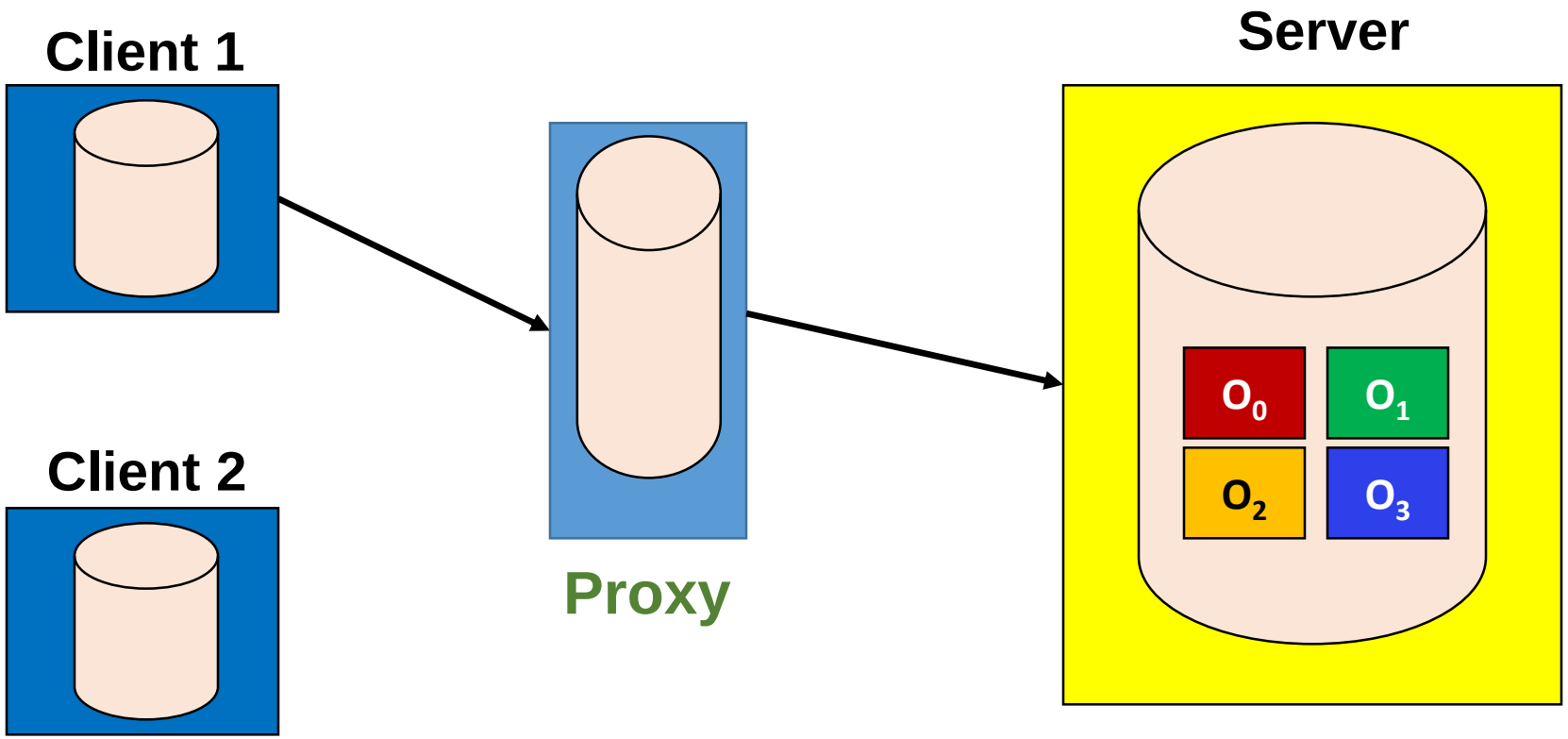
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



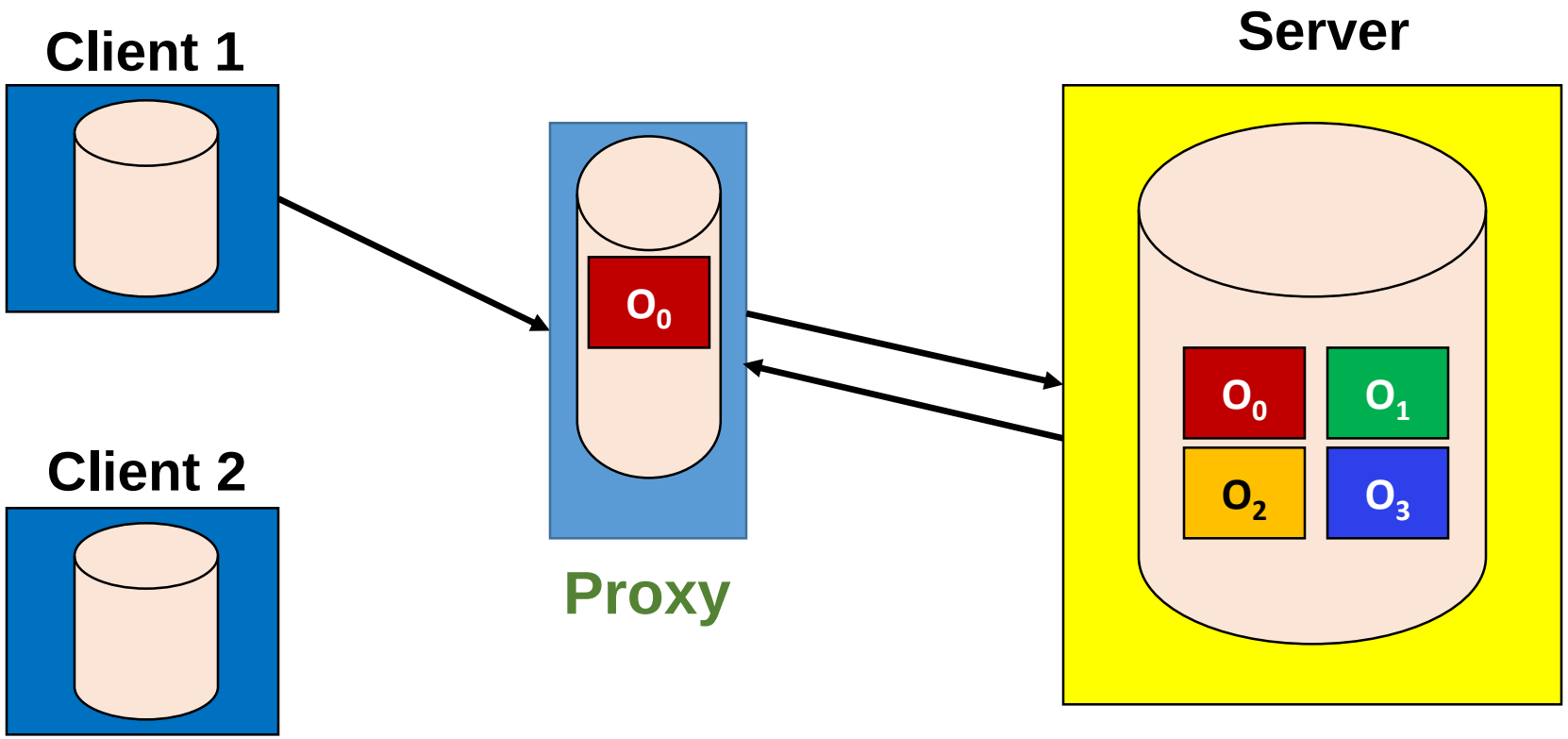
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



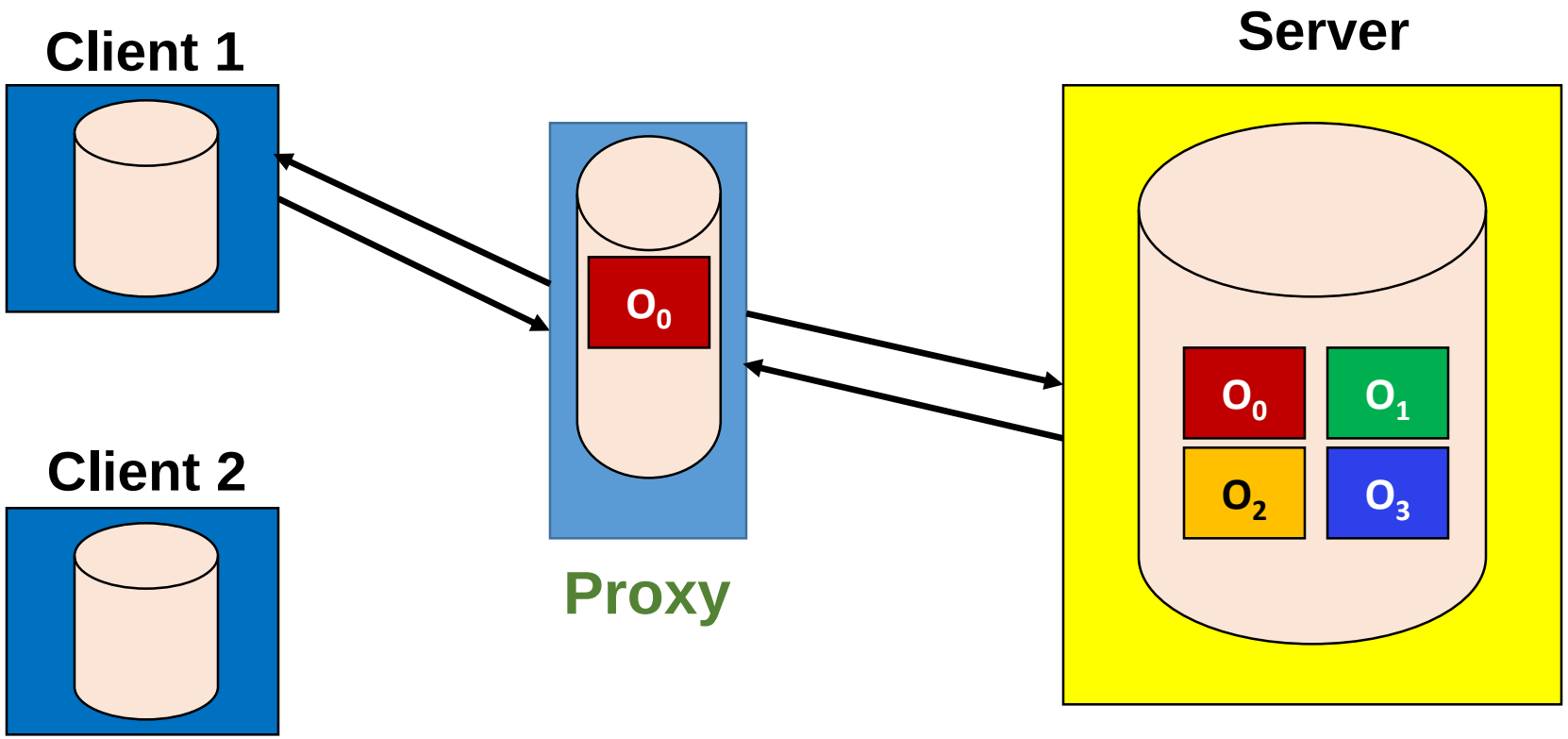
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



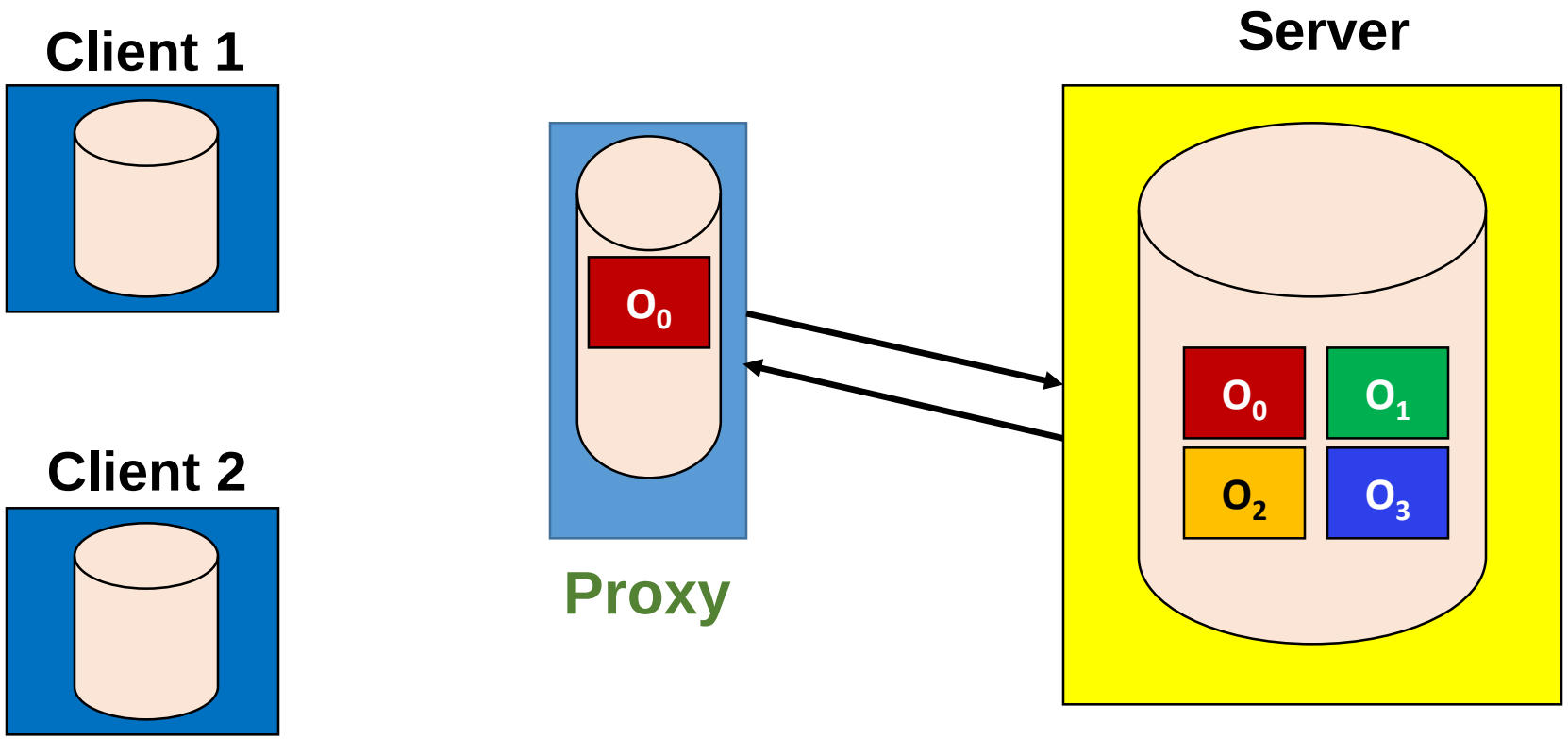
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



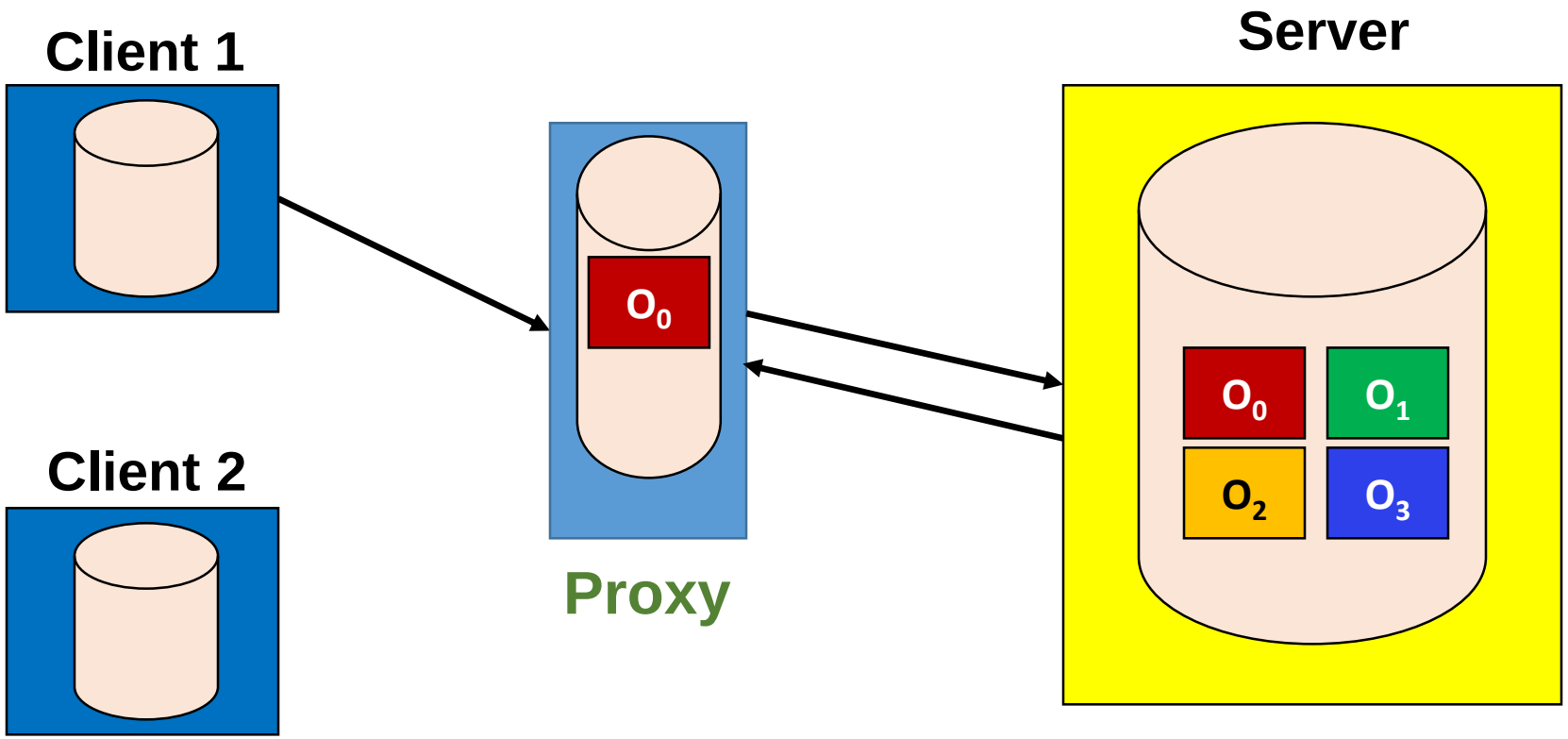
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



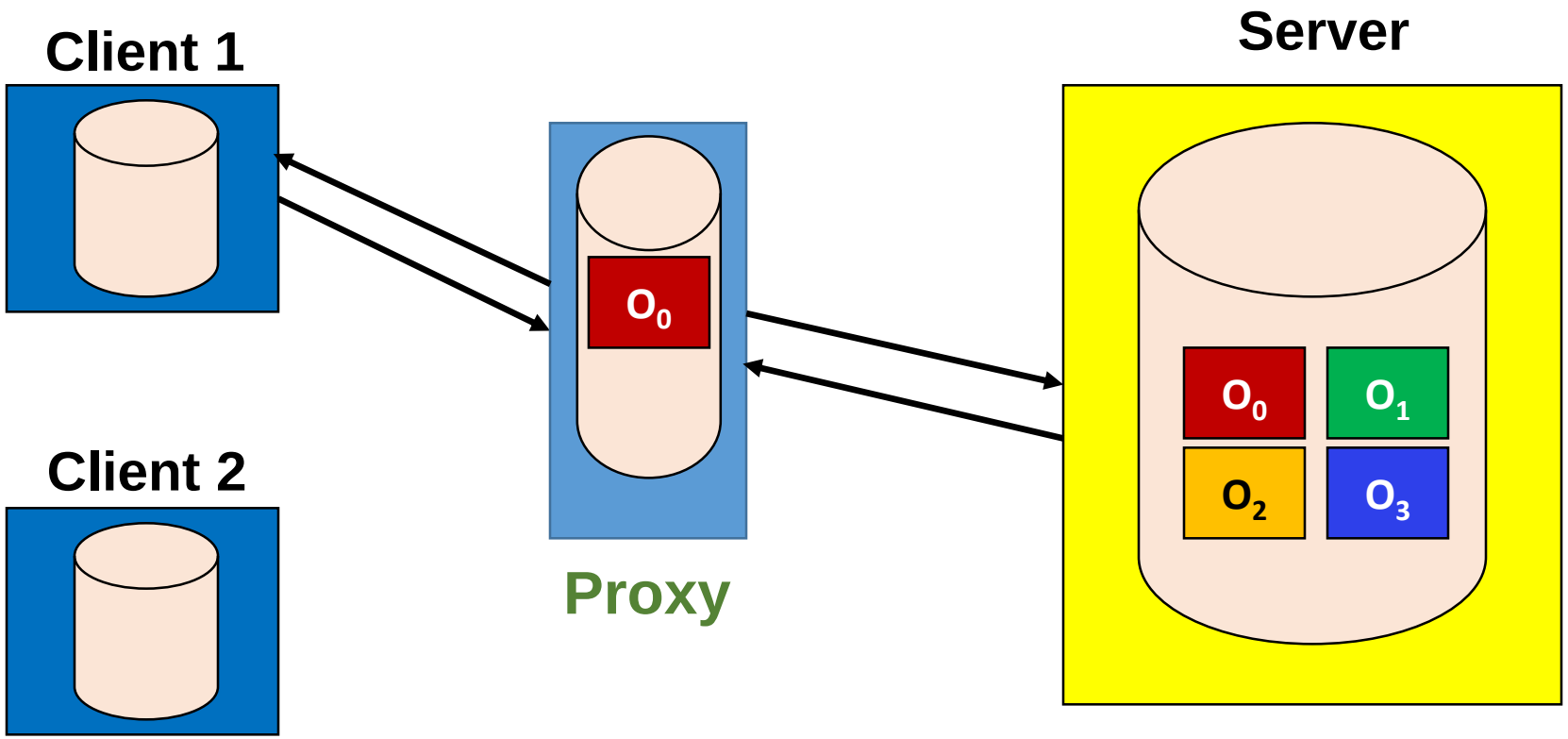
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



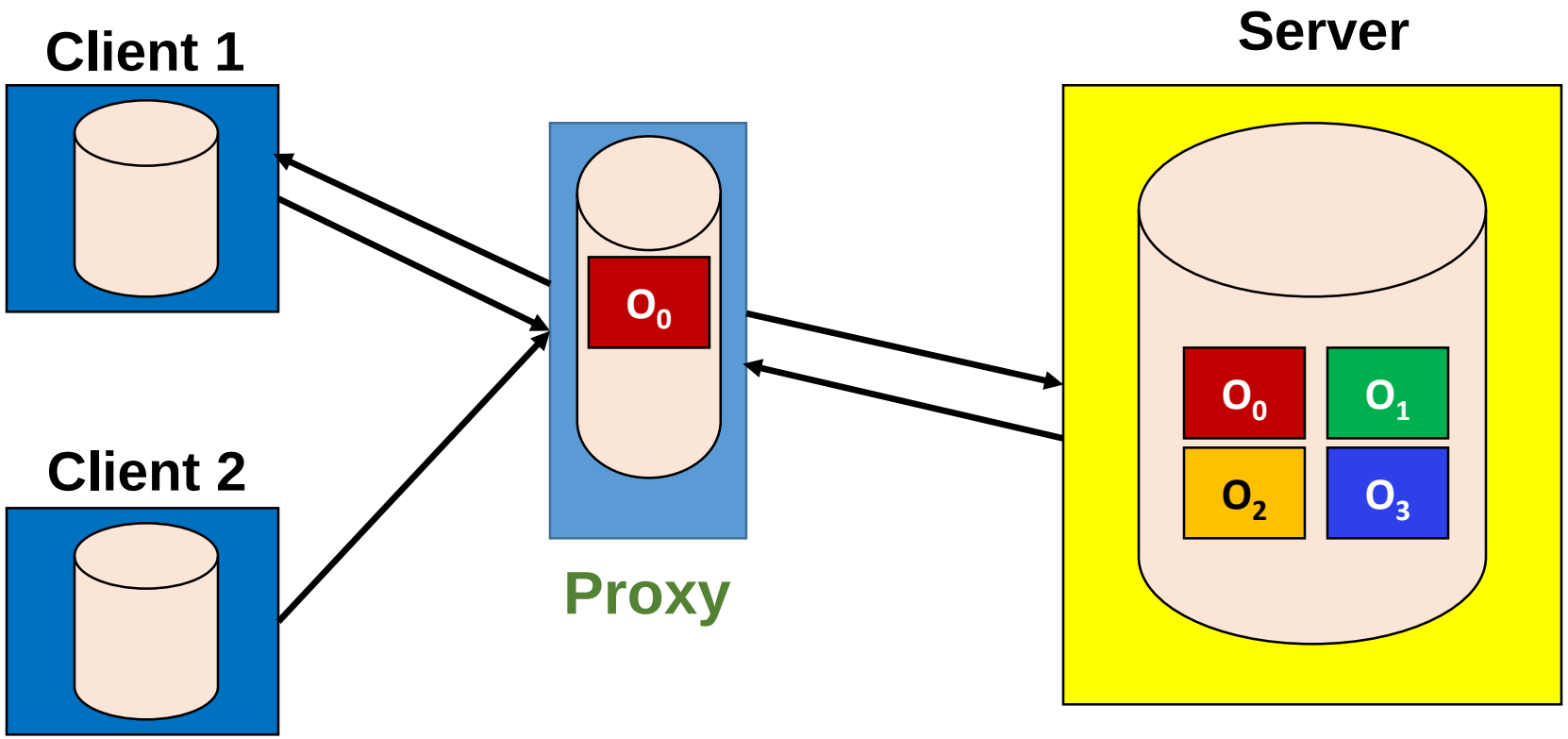
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



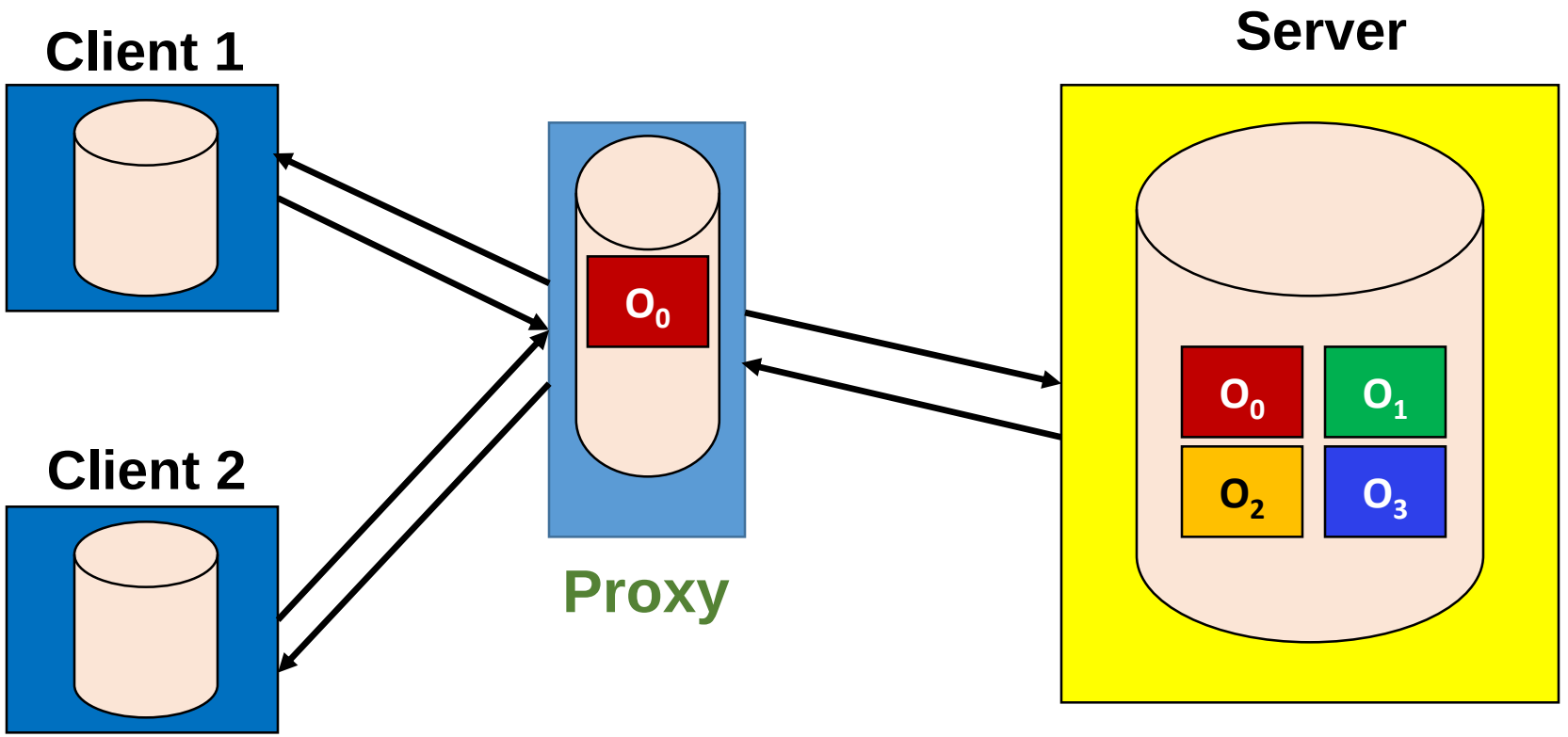
Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication



Replikasi Sisi Client vs. Server

- Akankah replikasi membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya, melalui server-side replication

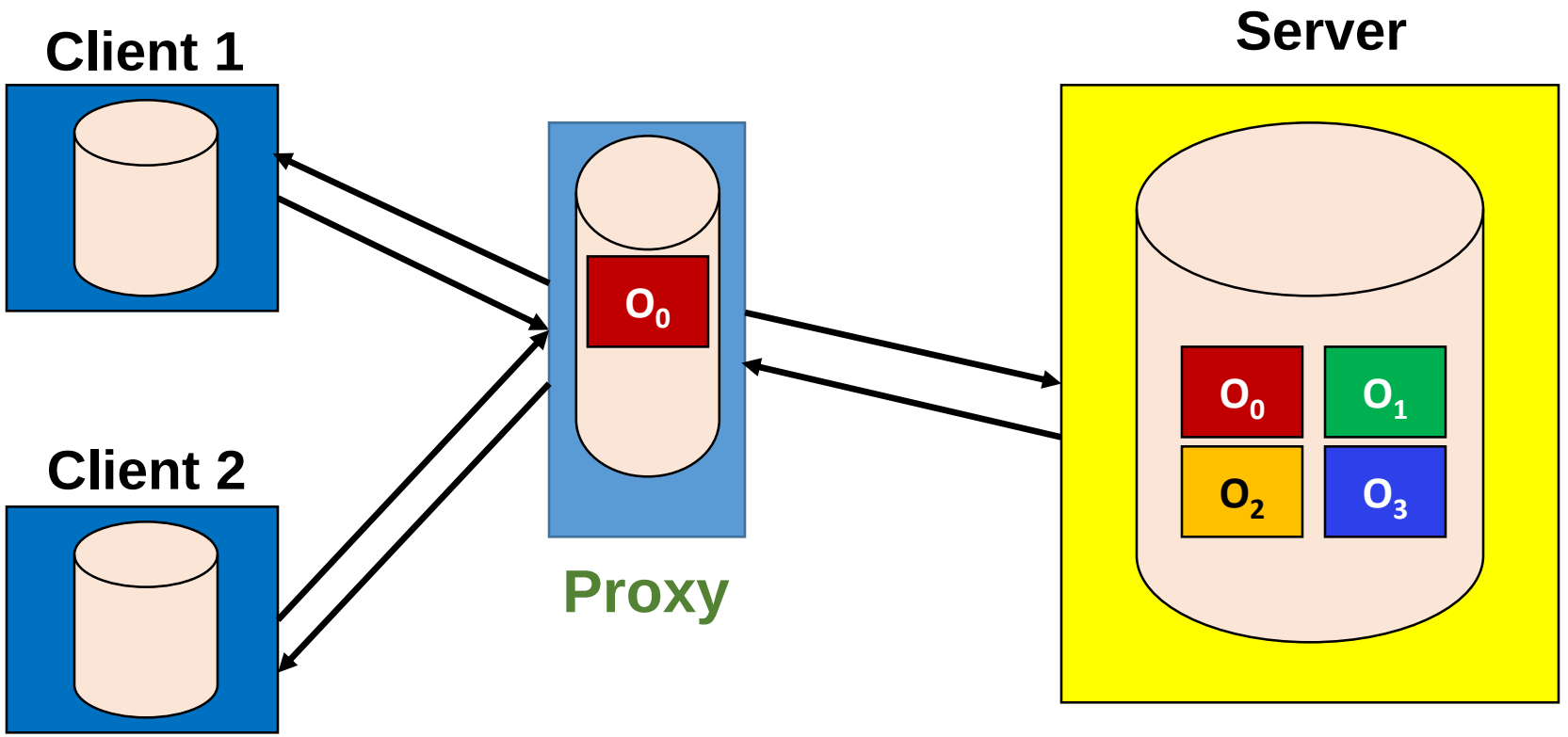


Replikasi Sisi Client vs. Server

- Akankah replikasi gabungan sisi client dan server membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya.

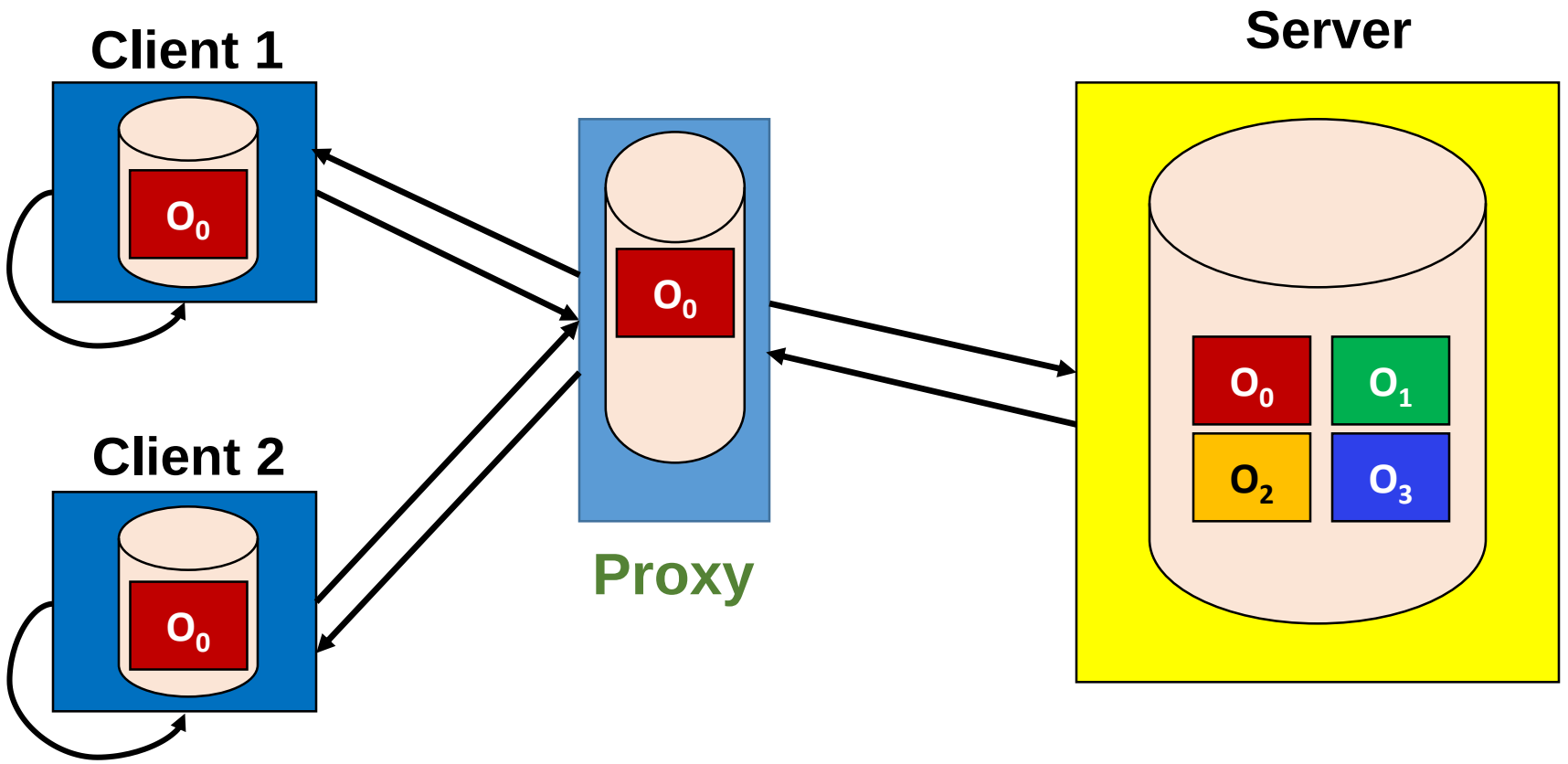
Replikasi Sisi Client vs. Server

- Akankah replikasi gabungan sisi client dan server membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya.



Replikasi Sisi Client vs. Server

- Akankah replikasi gabungan sisi client dan server membantu mengurangi latensi jika clients melaksanakan *overlapping* requests ke obyek data?
  - Ya.



Caching

- Kita akan fokus pada caching sebelum membahas replikasi

- Gagasan dasar dari caching (adalah sangat simpel):
  - Suatu obyek data disimpan di lokasi jauh
  - Client perlu membuat banyak referensi ke obyek tersebut
  - Suatu Salinan (copy atau *replika*) dari obyek tersebut dapat dibuat dan disimpan dekat
  - Client dapat *secara transparent* mengakses replikasi tersebut.

Storage lokal yang digunakan untuk replica sisi client dirujuk sebagai “cache”

Metriks Cache Sederhana

- References = Jumlah usaha untuk menemukan suatu obyek dalam cache
- Hits = Jumlah sukses
- Misses = Jumlah gagal
- Miss Ratio = Misses/References
- Hit Ratio = Hits/References = (1 − Miss Ratio)
- Biaya Referensi yang Diharapkan =
  (Miss Ratio × cost of miss) + (Hit Ratio × cost of hit)
- Keuntungan Cache = (Cost of Miss / Cost of Hit)
  - Dimana biaya diukur dalam waktu tunda untuk mengakses suatu objek

Mengapa Cache Efektif?

- Aplikasi cenderung menggunakan kembali data yang mereka akses terakhir
  - Disebut sebagai prinsip lokalitas (*principle of locality*)

- Dua jenis lokalitas berbeda :
  - Temporal locality
    - Objek yang baru diakses kemungkinan akan diakses lagi
  - Spatial locality
    - Objek yang berdekatan satu sama lain cenderung diakses secara berturut-turut

Mengapa Caching Efektif?

- Prinsip lokalitas memungkinkan :
  - Effective caching
  - Prefetching (pengambilan awal)
    - Yaitu, Mengambil objek yang kemungkinan akan diminta sebelum benar-benar diminta, sehingga menghasilkan cache hit saat diminta
    - Diaktifkan terutama oleh lokalitas spasial

- Aplikasi dengan (minimal atau) tanpa penggunaan kembali data (mis., Aplikasi streaming seperti streaming video), tidak mendapat manfaat dari caching
  - Mereka mungkin mendapat manfaat dari prefetching

Lokalitas Temporal dan Spasial

- Lokalitas temporal dan spasial sangat berbeda
  - Implementasi caching sering kali menggabungkannya secara ketat
  - Yang satu bisa ada tanpa yang lain
    - Spasial tanpa temporal (mis., Pemindaian linear file besar)
    - Temporal tanpa spasial (mis., Loop ketat mengakses hanya satu objek)

- Contoh: “rm -f *”
  - The shell expands “*” into a list
  - Loop berulang melalui daftar (list)
    - stat object
    - unlink object
  - Direktori induk menunjukkan lokalitas temporal
  - Entri direktori menunjukkan lokalitas spasial

Tiga Pertanyaan Kunci

- Data apa yang harus di-cache dan kapan?
  - Kebijakan Pengambilan
- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan konsistensi atau propagasi update
- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan penggantian Cache

Tiga Pertanyaan Kunci

- Data apa yang harus di-cache dan kapan?
  - Kebijakan Pengambilan
- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan konsistensi atau propagasi update
- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan penggantian Cache

Kebijakan Pengambilan

- Secara garis besar, ada dua jenis:
  - Kebijakan pengambilan berbasis Push
  - Kebijakan pengambilan berbasis Pull

Push-Based Caching

- Push-based Caching (or *Full Replication*)
  - Setiap mesin yang berpartisipasi mendapat salinan data lengkap terlebih dahulu
  - Setiap file baru didorong (dipush) ke semua mesin yang berpartisipasi
  - Setiap pembaruan pada file didorong segera ke setiap replika yang sesuai
  - Contoh: Dropbox
    - Bekerja cukup baik dalam praktiknya (keunggulan teknis hanya berkorelasi lemah dengan keberhasilan bisnis)

Push-Based Caching: Scalability Issues

- Jelas, ini dapat membuat masalah skalabilitas besar
  - Dengan ukuran tim yang lebih besar dan / atau kumpulan data, model berbasis push mengkonsumsi bandwidth jaringan dan ruang disk yang lebih besar
    - Pada skala yang sangat besar, mungkin diperlukan waktu sehari untuk menyelesaikan operasi sinkronisasi!
- Ini mengalahkan tujuan replikasi penuh, yang biasanya untuk memungkinkan kolaborasi antar tim
- Pendekatan yang berbeda disebut sebagai upaya cache berbasis pull (pull-based) untuk menyelesaikan masalah ini

Pull-Based Caching

- Pull-based Caching (atau *On-demand Caching*)
  - File diambil hanya jika diperlukan
  - Pembaruan pada suatu file (tidak harus seluruh file) disebarkan ke file yang direplikasi hanya jika diperlukan
  - Hal ini mengarah pada pendekatan yang lebih halus dan selektif (sebagai lawan dari model berbasis push) untuk manajemen data
  - Contoh: AFS

Tiga Pertanyaan Kunci

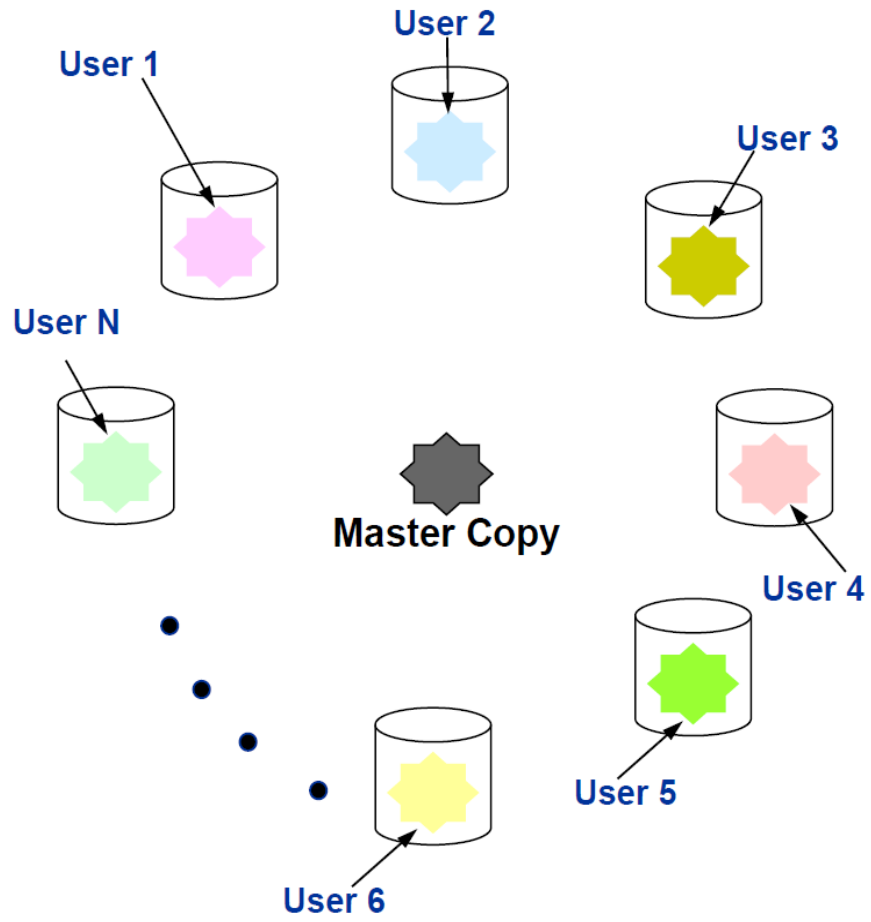
- Data apa yang harus di-cache dan kapan?
  - Kebijakan pengambilan

- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan Konsistensi atau Propagasi Update

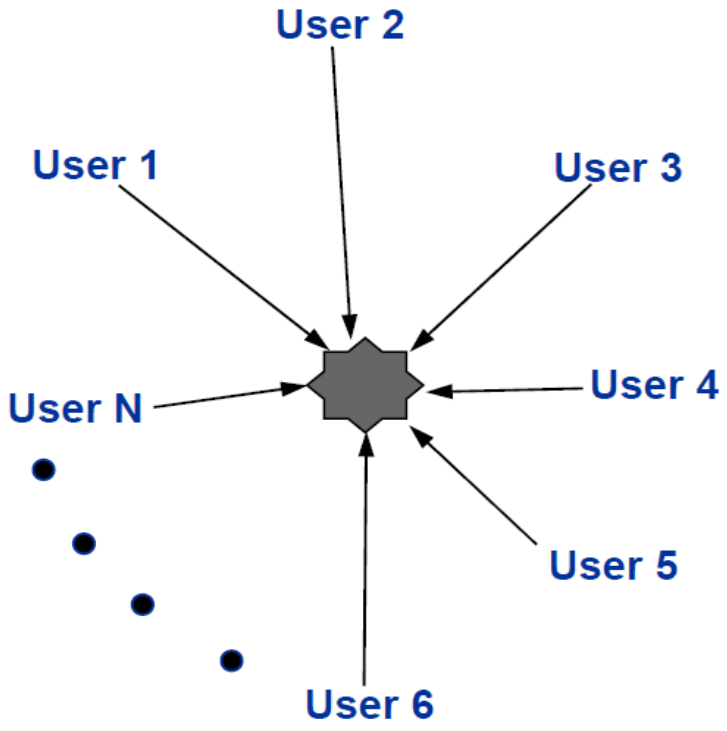
- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan penggantian cache

Semantik Satu Salinan

Caching Reality



Desired Illusion



One-Copy Semantic

Semantik Satu Salinan

- Sistem caching memiliki semantik satu-salinan (*one-copy*) jika dan hanya jika :
  - Tidak ada perbedaan fungsional yang dapat diamati secara eksternal sehubungan dengan sistem setara yang tidak melakukan caching
    - Namun, perbedaan kinerja / waktu dapat terlihat
- Ini sangat sulit dicapai dalam praktik
  - Kecuali dalam keadaan yang sangat sempit seperti sistem file yang berorientasi HPC dan DSM
  - Semua implementasi nyata adalah perkiraan

Apa yang Membuat Semantic Satu-Salin Sulit Diimplementasikan?

- Salinan master fisik mungkin tidak ada
  - Node biasanya melacak siapa yang memiliki salinan terbaru
  - Lebih mungkin terjadi di peer-to-peer daripada di sistem master-slave
- Jaringan mungkin terpecah antara beberapa klien dan salinan master
  - Situs yang terputus tidak melihat pembaruan lebih lanjut
  - Situs dapat dibagi menjadi beberapa wilayah, yang mungkin terisolasi dari satu sama lain
- Rasio baca-tulis yang rendah
  - Ini menghasilkan sejumlah besar lalu lintas propagasi cache
  - Interkoneksi mungkin menjadi penghambat untuk akses cache
  - Baik bias penulis (write-back) maupun bias pembaca (write-through) tidak membantu
  - Caching mungkin membuat secara efektif tidak berguna

Pendekatan Konsistensi Cache

- Kita akan mempelajari 7 pendekatan konsistensi cache:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

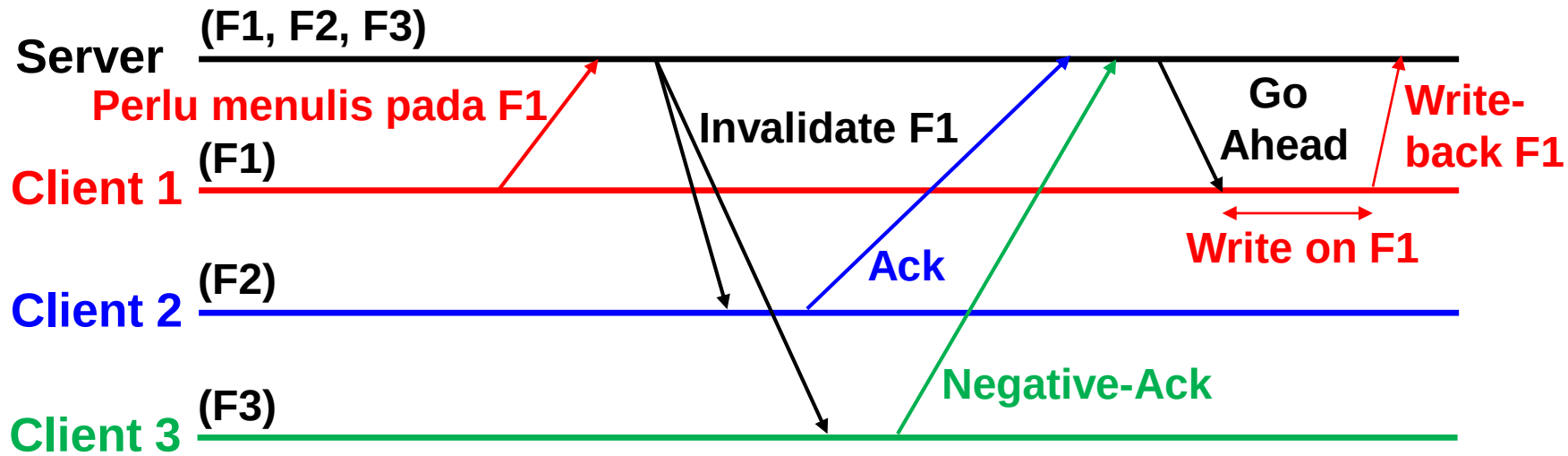
1. Validasi Siaran
2. Periksa Penggunaan
3. Telepon balik
   Sewa
   Lewati Bagian
   Menakutkan
   Caching Berbasis Iman
   Lewati uang itu

Pendekatan Konsistensi Cache

- Kita akan mempelajari 7 pendekatan konsistensi cache:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Broadcast Invalidations

- Write berjalan sebagai berikut:



- Membaca pada objek yang di-cache dapat diproses secara langsung

Broadcast Invalidations

- Server tidak memelihara direktori yang melacak siapa yang saat ini melakukan cache setiap objek
- Dengan demikian, pada setiap pembaruan ke objek apa pun, server menyiarkan pesan pembatalan ke setiap situs caching
- Jika sebuah situs melakukan caching objek, itu membatalkannya; jika tidak, ia mengirim pesan Ack negatif ke server
  - Jika tidak valid, referensi selanjutnya ke objek ini di situs ini akan menyebabkan kehilangan

Broadcast Invalidations

- Keuntungan:
  - Tidak ada status khusus (kecuali lokasi situs caching) yang dipertahankan di server
    - A stateless server
  - Tidak ada kondisi balapan yang dapat terjadi jika pembaru memblokir sampai semua salinan cache dari objek yang diminta (kecuali miliknya sendiri) tidak valid
  - Emulasi semantik satu salinan yang sangat ketat
  - Mudah diimplementasikan

Broadcast Invalidations

- Disadvantages:
  - Lalu lintas terbuang sia-sia, terutama jika tidak ada situs yang meng-cache objek yang diminta
  - Pembaru memblokir hingga proses invalidasi selesai
  - Tidak dapat diskalakan di jaringan besar
    - Dapat menyebabkan banjir jaringan jika jumlah menulis tinggi dan rasio baca / tulis rendah
  - Mengharuskan semua situs mendengarkan (atau mengintip) semua permintaan

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Check on Use

- Server tidak membatalkan salinan yang di-cache saat pembaruan

- Sebaliknya, seorang pemohon di situs mana pun memeriksa dengan server sebelum menggunakan objek apa pun
  - Versi dapat digunakan, di mana setiap salinan file diberi nomor versi
  - Apakah salinan saya masih valid?
    - Jika tidak, ambil salinan objek yang baru
    - Jika ya dan saya seorang pembaca, lanjutkan
    - Jika ya dan saya seorang penulis, lanjutkan dan balas menulis (write-back) setelah selesai

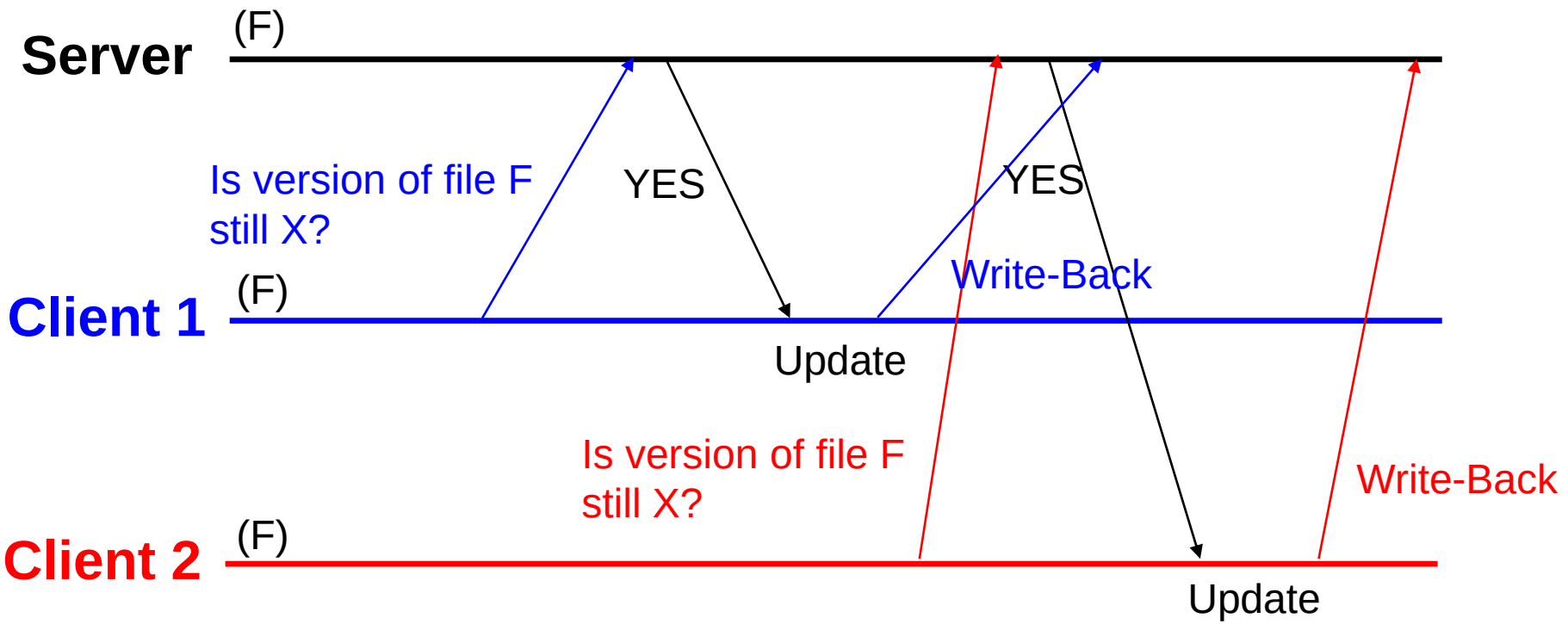
Check on Use

- Harus dilakukan pada granularity coarse (mis., Seluruh file atau blok besar)
  - Jika tidak, bacaan diperlambat secara berlebihan

- Ini menghasilkan semantik sesi jika dilakukan pada granularity seluruh file
  - Open {Read | Write}* Close ➔ “session”
  - Pembaruan pada file yang terbuka pada awalnya hanya terlihat oleh pembaru file
  - Hanya ketika file ditutup, perubahan yang dibuat terlihat oleh server

Check on Use

- Kekurangan:
  - "Mutakhir" relatif terhadap latensi jaringan



Pembaruan Bersamaan!

Check on Use

- Disadvantages:
  - Bagaimana menangani menulis bersamaan (*concurrent write*)?
    - The final result depends on whose write-back arrives last at the server
      - Ini dipengaruhi oleh latensi jaringan
        - Jika pembaruan A dan B persis sama
        - Dan mesin-mesin di mana mereka diperuntukkan adalah homogen
        - Dan A dimulai, selesai, dan dikirim sebelum B
        - Tidak perlu bahwa A akan mencapai server sebelum B
  - Lambar membaca
    - Terutama dengan server yang berbeban dan jaringan latensi tinggi

Check on Use

- Disadvantages:
  - Pendekatan pesimistis, terutama dengan beban kerja *read-most*
    - Bisakah kita menggunakan pendekatan yang optimis (Trust-and-Verify)?

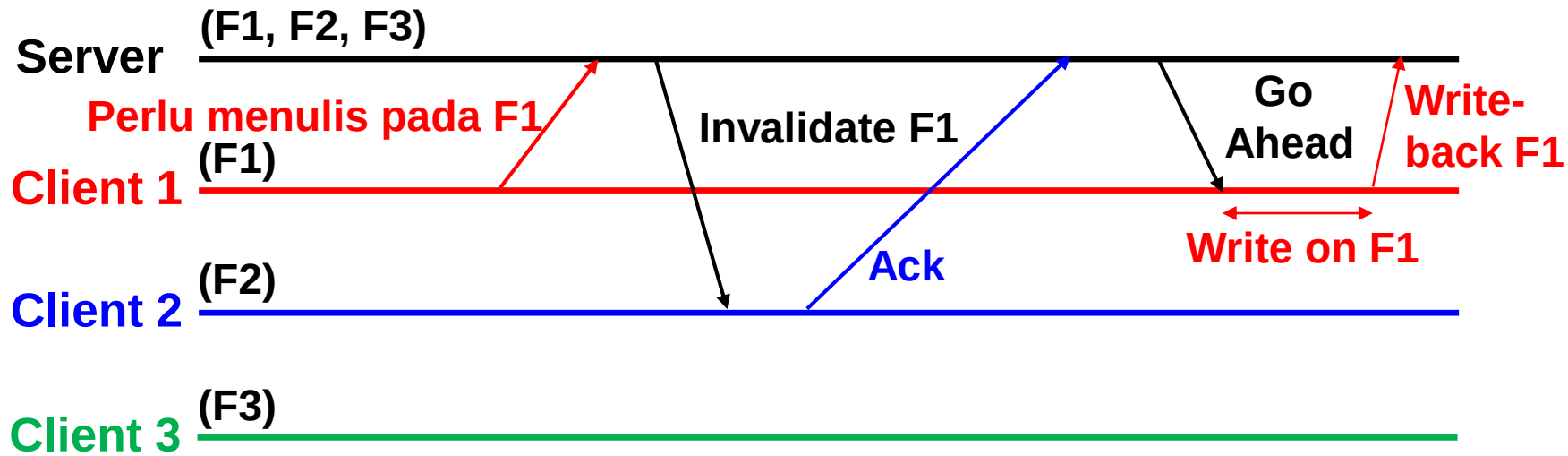
- Advantages:
  - Strict consistency (not across all copies) at coarse granularity
  - No special server state is needed
  - Server tidak perlu tahu apa-apa tentang situs caching
  - Easy to implement

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Callback

- Write berjalan sebagai berikut:



- Membaca pada objek yang di-cache dapat dilakukan secara langsung

Callback

- Server memelihara direktori yang melacak siapa yang saat ini melakukan cache setiap objek

- Dengan demikian, setelah pembaruan ke suatu objek, server mengirim pesan tidak valid (yaitu panggilan balik, callback) hanya ke situs yang saat ini melakukan caching objek
  - Biasanya dilaksanakan pada coarse granularity (mis, seluruh file)
    - Dapat dibuat bekerja dengan rentang byte

Callback

- Advantages:
  - Targeted notification of caching sites
  - Zero network traffic for reads of cached objects
  - Biases read performance in favor of write-performance
  - Skalabilitas luar biasa, terutama dengan beban kerja *read-most*

Callback

- Disadvantages:
  - Kompleksitas pelacakan objek yang di-cache pada klien
  - Sizable state on server
  - Diam di server tidak jelas untuk klien
    - Bagaimana jika klien telah membaca file untuk sementara waktu tanpa mendengar kembali dari server?
      - Perhaps the server is down
      - Mekanisme tetap hidup (atau keep-alive atau heartbeat/detak jantung) dapat dimasukkan, di mana server ping klien (atau sebaliknya) setiap sekarang dan kemudian menunjukkan bahwa ia masih hidup

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Leases

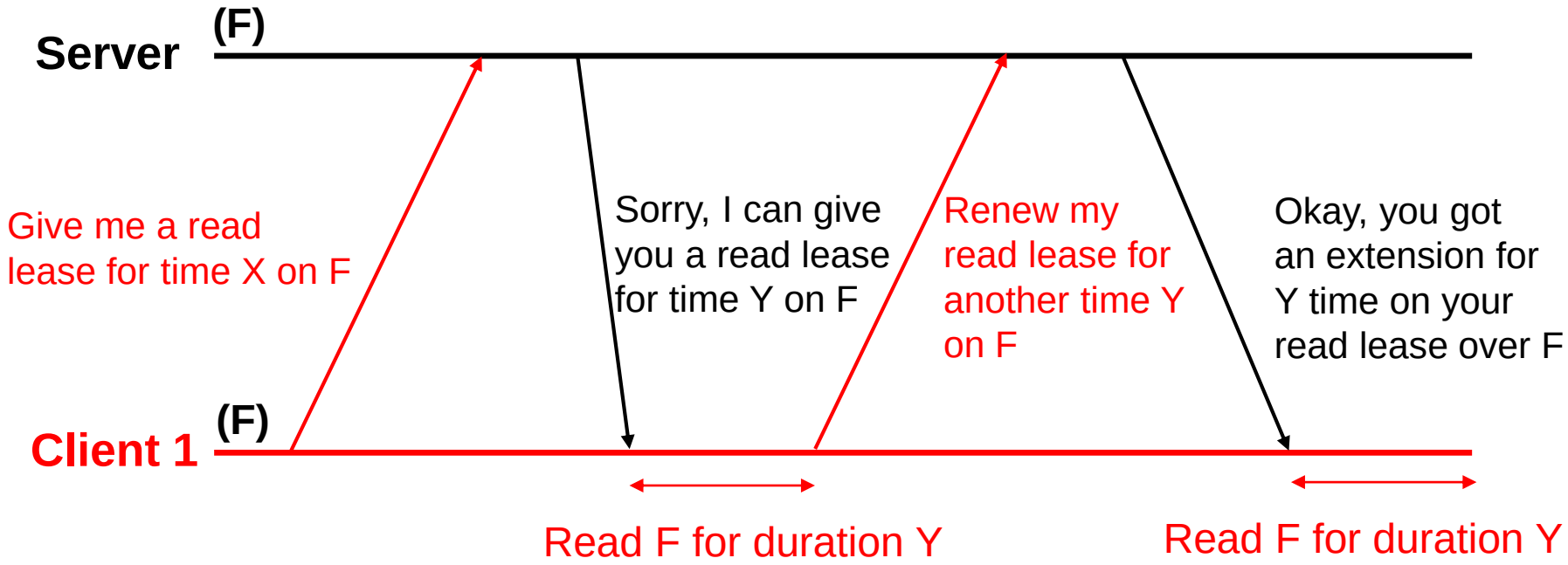
- Pemohon perlu mendapatkan kontrol durasi terbatas dari server
  - This duration is called a *lease period* (typically, for few seconds)

- There are three types of leases
  - *Read* and *write* leases, assuming an *invalidation-based protocol*
    - Banyak pemohon dapat memperoleh sewa baca pada objek yang sama, tetapi hanya satu yang bisa mendapatkan sewa tulis pada objek apa pun
  - *Open* leases, assuming *a check-on-use protocol*

- Pemohon kehilangan kendali saat masa sewa habis
  - Jika perlu, pemohon dapat memperbarui sewa

Pembaruan Sewa

- Contoh:



Jam Tersinkronisasi Diasumsikan di Semua Situs

Key Questions

- Data apa yang harus di-cache dan kapan?
  - Kebijakan pengambilan
- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan Konsistensi atau propagasi
- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan Penggantian Cache

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

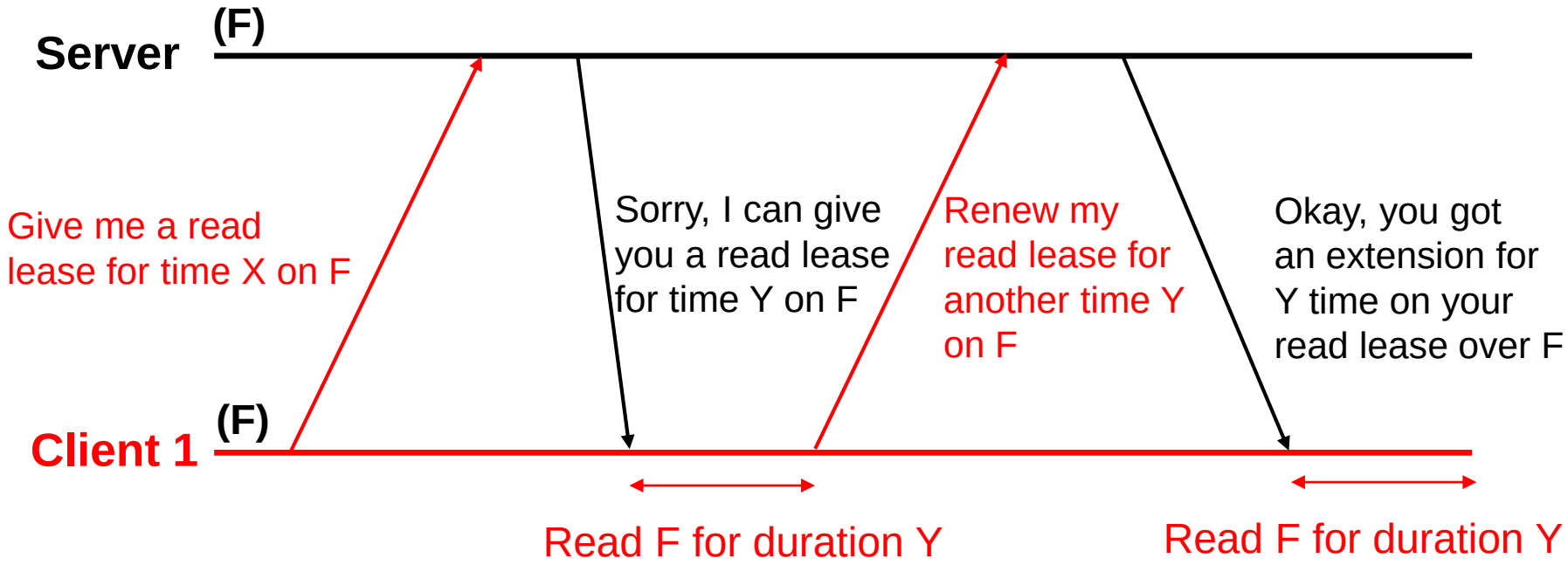
Leases

- Klien menempatkan permintaan untuk mendapatkan kontrol durasi terbatas dari server
  - Durasi ini dinamakan *lease period* (waktu sewa, biasanya beberapa detik)

- Ada tiga jenis sewa
  - Sewa *Read* dan *write*, dengan asumsi protokol berbasis tidak valid
    - Banyak pemohon dapat memperoleh sewa baca pada objek yang sama, tetapi hanya satu yang bisa mendapatkan sewa tulis pada objek apa pun
  - *Sewa terbuka, dengan asumsi protokol check-on-use*
- Pemohon kehilangan kendali saat masa sewa habis
  - Namun, ia dapat memperbarui sewa jika diperlukan.

Lease Renewal

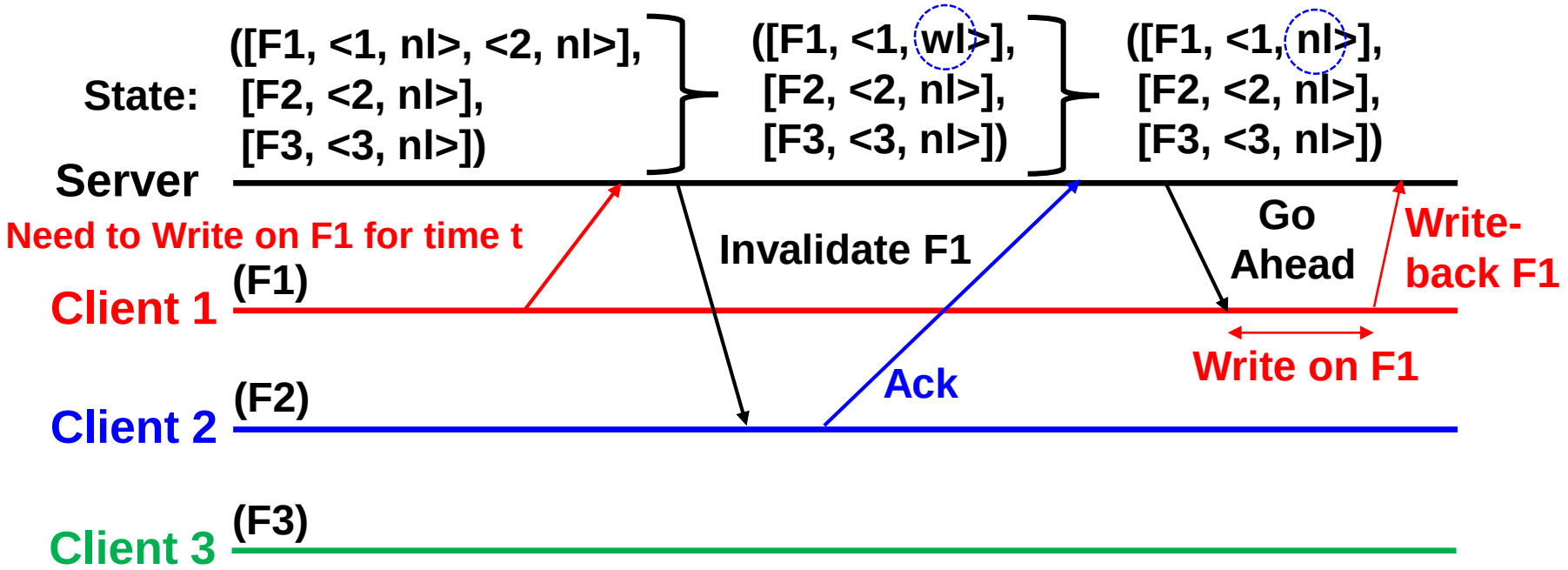
- Contoh:



Jam di Mesin Terlibat Diasumsikan Disinkronkan

Leases

- Write berjalan sebagai berikut, dengan asumsi protokol berbasis tidak valid:



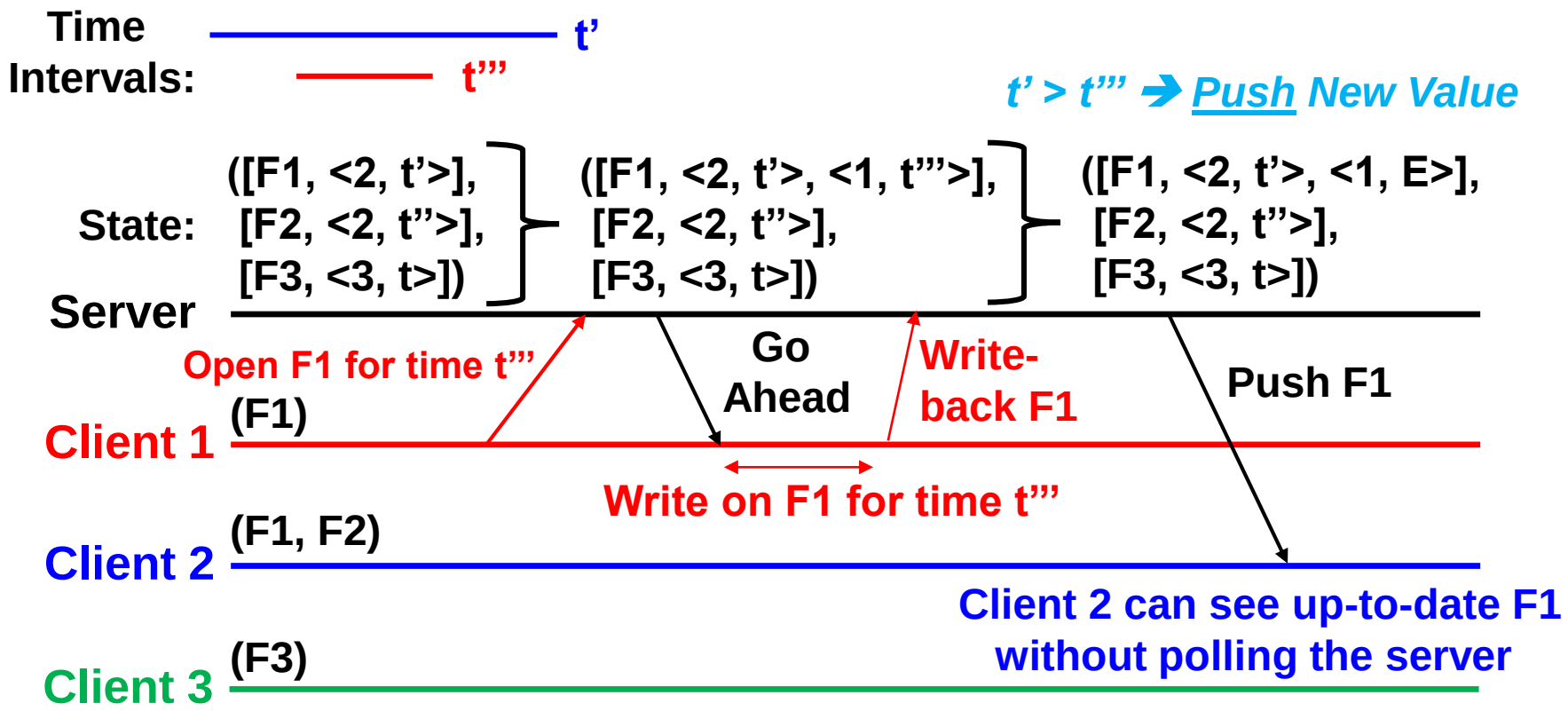
[Fi, <x, y>] = File **Fi** is cached at Client **x** and is either not leased (i.e., **y** = **nl**), or read-leased (**y** = **rl**), or write-lease (**y** = **wl**).

Leases

- Bagaimana jika permintaan write tiba di server di antaranya?
  - Itu bisa diantrikan, sampai permintaan sebelumnya terpenuhi
    - Hanya satu penulisan yang dapat dijalankan sekaligus dan beberapa permintaan dapat di-antri dan dilayani dalam urutan tertentu (mis., urutan FIFO)
  - Ketika diservis, salinan terbaru harus dikirim ke situsnya (karena salinannya telah dibatalkan sebelum mengizinkan penulisan sebelumnya untuk diproses)
- Bagaimana jika permintaan read tiba di server di antaranya?
  - Itu bisa diantrikan juga
  - After write is done, either another write is pursued singlehandedly, or one or more reads go in parallel
    - Dalam kasus apa pun, salinan terbaru harus dikirimkan juga

Leases

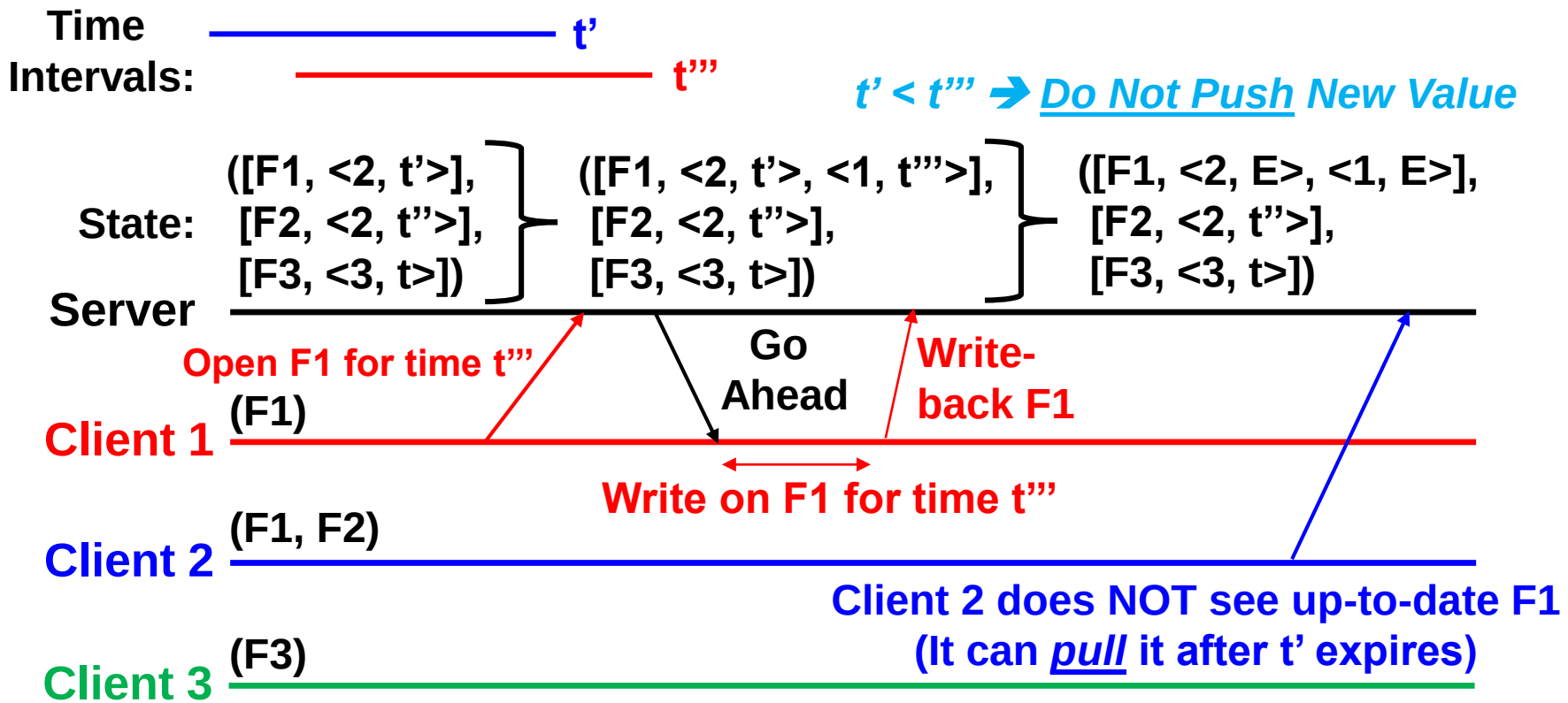
- Terbuka jalan sebagai berikut, dengan asumsi semantik-sesi:



[Fi, <x, y>] = File **Fi** is cached at Client **x** and either has its lease expired (i.e., **y** = **E**), or valid till end of **y**.

Leases

- Terbuka jalan sebagai berikut, dengan asumsi semantik-sesi:



[Fi, <x, y>] = File **Fi** is cached at Client **x** and either has its lease expired (i.e., **y** = **E**), or valid till end of **y**.

Leases

- In this case:
  - A lease becomes a *promise* by the server that it will *push* updates to a client for a specified time (i.e., the lease duration)
  - When a lease expires, the client is forced to poll the server for updates and *pull* the modified data if necessary
    - Klien juga dapat memperbarui sewa dan mendapatkan kembali pembaruan yang didorong ke situsnya untuk durasi sewa baru
      - *Fleksibilitas dalam pilihan!*

Leases

- Advantages:
  - Menggeneralisasi skema check-on-use dan callback
  - Durasi sewa dapat disesuaikan untuk beradaptasi dengan tingkat mutasi
    - Ini adalah tombol tuning yang bersih untuk fleksibilitas desain
  - Secara konsep sederhana, namun fleksibel

Leases

- Disadvantages:
  - Pemegang sewa memiliki otonomi total selama masa sewa
    - Beban / prioritas dapat berubah di server
    - Pencabutan (di mana suatu leasing ditarik oleh server dari pemegang leasing) dapat dimasukkan
  - Dalam protokol berbasis sewa yang tidak valid:
    - Penulis akan ditunda pada suatu objek sampai semua sewa baca pada objek itu berakhir
    - Keep-alive callbacks dibutuhkan
    - Server stateful, yang biasanya menyiratkan toleransi kesalahan dan skalabilitas yang lebih rendah (dalam hal kapasitas dan komunikasi)

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Skip Scary Parts: Lewati Bagian Menakutkan

- Gagasan dasar:
  - Saat berbagi-tulisan (*write-sharing)* terdeteksi, caching dimatikan
  - Setelah itu, semua referensi langsung menuju salinan master
  - Caching dilanjutkan ketika *write-sharing* berakhir

- Keuntungan:
  - Semantik salinan tunggal yang tepat (bahkan pada konsistensi tingkat byte) Strategi fallback yang sangat baik
    - Mencontohkan Teknik yang baik: “menangani rerata kasus dengan baik; kasus terburuk tetap aman”
  - Adaptasi agresivitas caching yang baik terhadap karakteristik beban kerja (mis., Pola baca dan tulis)

Skip Scary Parts

- Disadvantages:
  - Server perlu mewaspadai setiap penggunaan data
    - Dengan asumsi itu digunakan bersamaan dengan check-on-use
      - Salah satu klien mengekspos keinginan mereka untuk membuat tulisan pada saat membuka file
      - Atau server bergantung pada balasan balik klien saat menutup file (yang mengindikasikan penulisan pada file)
  - Server memelihara beberapa *status pemantauan*

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

A Primer: Eventual Consistency

- Banyak aplikasi yang dapat mentolerir ketidakkonsistenan untuk waktu yang lama
  - Pembaruan halaman web, Pencarian Web - crawling, indexing dan ranking, Pembaruan ke Server DNS

- Dalam aplikasi seperti itu, dapat diterima dan efisien jika pembaruan jarang disebarkan

- Skema caching disebut sebagai akhirnya konsisten (eventually consistent) jika:
  - Semua replika akan secara bertahap menjadi konsisten tanpa adanya pembaruan

A Primer: Eventual Consistency

- Skema cache biasanya menerapkan konsistensi akhirnya jika:
  - *Konflik Write-write jarang terjadi*
    - Sangat jarang bagi dua proses untuk menulis ke objek yang sama
    - Secara umum, satu klien memperbarui objek data
      - E.g., Satu server DNS memperbarui pemetaan nama-ke-IP
    - Konflik langka dapat ditangani melalui mekanisme sederhana, seperti saling pengecualian (mutex)
  - *Konflik Read-write lebih sering terjadi*
    - Konflik di mana satu proses membaca objek, sedangkan proses lain sedang menulis (atau berusaha menulis) ke replika itu
    - Skema yang akhirnya konsisten harus fokus pada penyelesaian konflik ini secara efisien

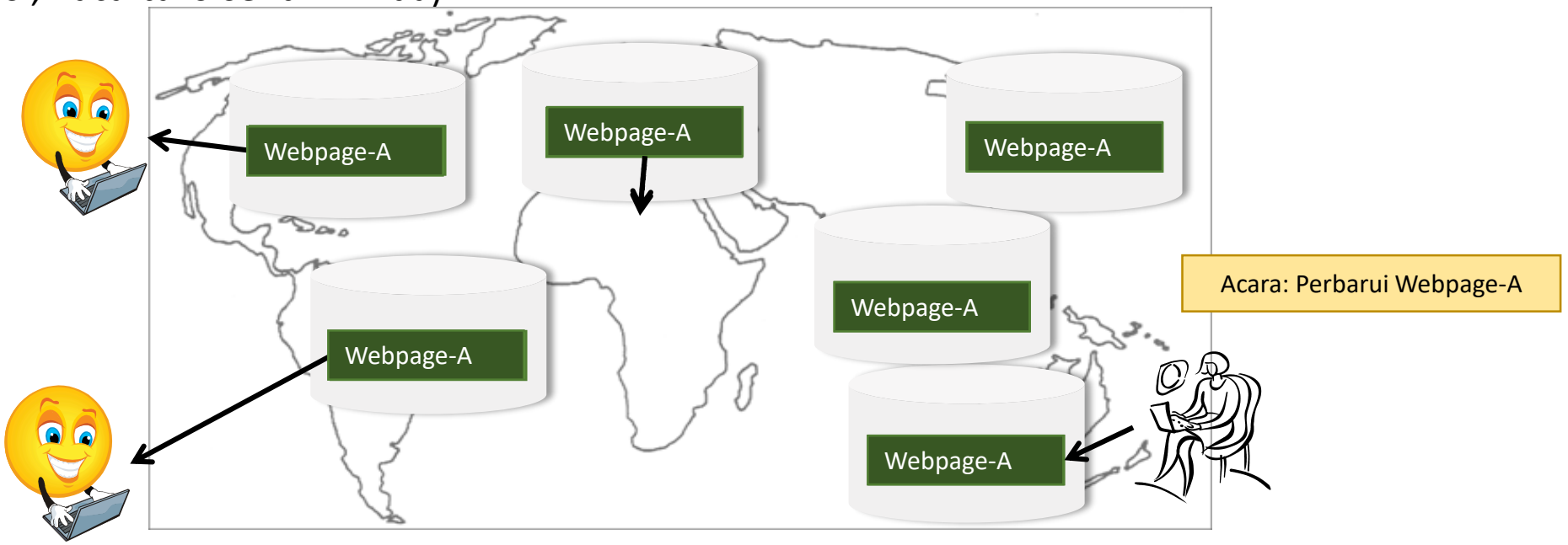
Faith-Based Caching

- Ide Dasar (implementasi konsistensi yang mungkin):
  - Klien secara membabi buta menganggap data yang di-cache valid untuk sementara waktu
    - Referred to as “trust period”
      - E.g., Dalam Sun NFSv3 file yang di-cache diasumsikan saat ini selama 3 detik, sedangkan direktori selama 30 detik
    - Varian kecil adalah menyetel field time-to-live (TTL) untuk setiap objek
  - Ini secara berkala memeriksa (berdasarkan waktu sejak pemeriksaan terakhir) validitas data yang di-cache
  - Tidak ada komunikasi yang terjadi selama periode kepercayaan

- Advantages:
  - Simple implementation
  - Server is stateless

Faith-Based Caching

- Disadvantages:
  - Potensi inkonsistensi yang terlihat oleh pengguna ketika klien mengakses data dari replika berbeda
    - Jaminan konsistensi biasanya diperlukan untuk satu klien saat mengakses salinan yang di-cache (mis., Baca-tulis-sendiri-Anda)



Ini menjadi lebih dari masalah konsistensi untuk replikasi sisi server (kami akan membahasnya nanti di bawah replikasi sisi server)

Pendekatan Konsistensi Cache

- Kita akan mempelajari 7 pendekatan konsistensi cache :
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Pass the Buck

- Ide Dasar (implementasi lain dari konsistensi yang mungkin terjadi)
  - Biarkan pengguna memicu validasi ulang cache (hit “reload”)
  - Jika tidak, semua salinan yang di-cache dianggap sah
  - Equivalent to infinite-TTL faith-based caching

- Advantages:
  - Implementasi sederhana
  - Menghindari lalu lintas pemeliharaan cache sembrono
  - Server is stateless

Pass the Buck

- Disadvantages:
  - Menempatkan beban pada pengguna (users)
  - Pengguna mungkin tidak mengerti tentang tingkat konsistensi yang dibutuhkan
  - Mengasumsikan keberadaan pengguna
    - Penderitaanuntuk menulis skrip / program

Cache Consistency Approaches

- We will study *7* cache consistency approaches:
  1. Broadcast Invalidations
  2. Check on Use
  3. Callback
  4. Leases
  5. Skip Scary Parts
  6. Faith-Based Caching
  7. Pass the Buck

Banyak varian kecil selama bertahun-tahun, tetapi ini telah bertahan dalam ujian waktu!

Key Questions

- Data apa yang harus di-cache dan kapan?
  - Kebijakan pengambilan
- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan Konsistensi atau Propagasi Update

- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan Penggantian Cache

Key Questions

- Data apa yang harus di-cache dan kapan?
  - Kebijakan pengambilan
- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan Konsistensi atau Propagasi Update
- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan Penggantian Cache

Key Questions

- Data apa yang harus di-cache dan kapan?
  - Kebijakan pengambilan
- Bagaimana pembaruan dapat dibuat terlihat di mana-mana?
  - Kebijakan Konsistensi atau Propagasi Update
- Data apa yang harus digusur untuk membebaskan ruang?
  - Kebijakan Penggantian Cache

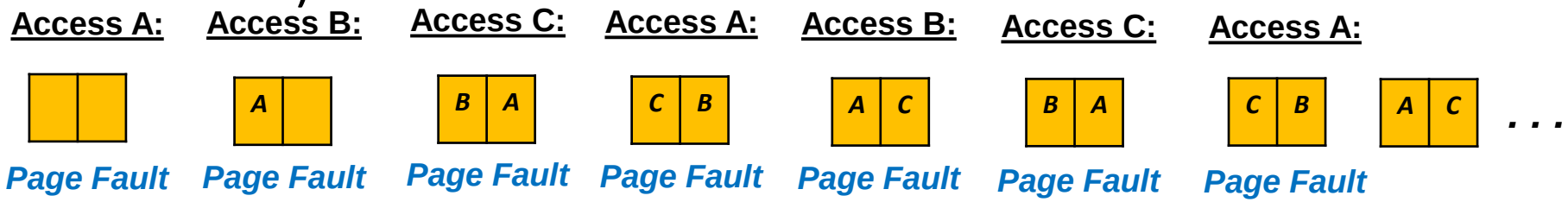
Working Sets

- Diberikan interval waktu T, WorkingSet (T) didefinisikan sebagai set objek data berbeda yang diakses selama T
  - Ini adalah fungsi dari lebar T
  - Ukurannya (atau apa yang disebut sebagai ukuran set kerja) adalah yang terpenting
  - Itu menangkap kecukupan ukuran cache sehubungan dengan perilaku program

- Apa yang terjadi jika proses klien melakukan akses berulang ke beberapa data, dengan ukuran set kerja yang lebih besar dari cache yang mendasarinya?

The LRU Policy: Sequential Flooding

- Untuk menjawab pertanyaan ini, anggaplah :
  - Tiga halaman, A, B, dan C sebagai unit cache ukuran tetap
  - Pola akses: A, B, C, A, B, C, dll.
  - Cache pool yang hanya terdiri dari dua frame (yaitu, Kontainer halaman berukuran sama)



- Meskipun pola akses menunjukkan lokalitas temporal, tidak ada lokalitas yang dieksploitasi!
- Fenomena ini dikenal sebagai "banjir sekuensial"
- Untuk pola akses ini, MRU bekerja lebih baik!

Types of Accesses

- Mengapa LRU tidak berkinerja baik dengan pola akses ini, meskipun “dapat berulang"?
  - Ukuran cache dikerdilkan oleh ukuran set yang berfungsi

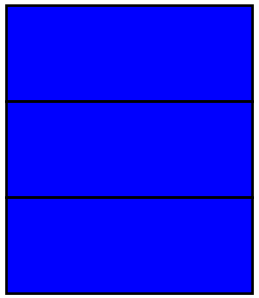
- Ketika interval waktu T meningkat, bagaimana ukuran kerja yang ditetapkan akan berubah, dengan asumsi :
  - Sequential accesses (mis., pemindaian penuh yang tidak dapat diulang)
    - Ini akan meningkat secara monoton
    - Working set akan membuat cache sangat tidak ramah
  - Regular accesses, which demonstrate typical good locality
    - Ini akan meningkat secara non-monoton (mis., Naik dan turun kemudian naik dan turun, tetapi tidak harus dengan lebar yang sama di seluruh fase program)
    - Set kerja akan menjadi ramah cache hanya jika ukuran cache tidak dikerdilkan oleh ukurannya
  - Akses acak, yang menunjukkan tidak ada atau sangat sedikit lokalitas (mis., Akses ke tabel hash)
    - Set yang berfungsi akan menunjukkan cache tidak ramah jika ukurannya jauh lebih besar dari ukuran cache

Contoh

LRU Chain:

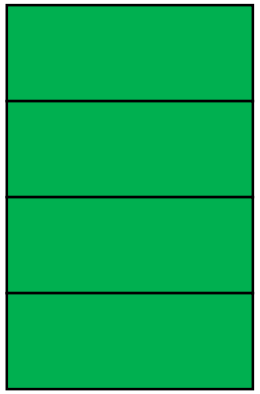
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4

Cache X (size = 3)



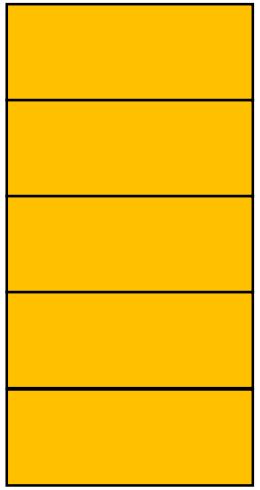
# of Hits:
# of Misses:

Cache Y (size = 4)



# of Hits:
# of Misses:

Cache Z (size = 5)



# of Hits:
# of Misses:

Contoh

LRU Chain: 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	7	7	7
Frame 1			
Frame 2			
Frame 3			
Frame 4			
	# of Hits: # of Misses: 1	# of Hits: # of Misses: 1	# of Hits: # of Misses: 1

Contoh

LRU Chain: 0 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	7	7	7
Frame 1	0	0	0
Frame 2			
Frame 3			
Frame 4			
	# of Hits: # of Misses: 2	# of Hits: # of Misses: 2	# of Hits: # of Misses: 2

Contoh

LRU Chain: 1 0 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	7	7	7
Frame 1	0	0	0
Frame 2	1	1	1
Frame 3			
Frame 4			
	# of Hits: # of Misses: 3	# of Hits: # of Misses: 3	# of Hits: # of Misses: 3

Contoh

LRU Chain: 2 1 0 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	2	7	7
Frame 1	0	0	0
Frame 2	1	1	1
Frame 3		2	2
Frame 4			
	# of Hits: # of Misses: 4	# of Hits: # of Misses: 4	# of Hits: # of Misses: 4

Contoh

LRU Chain: 0 2 1 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	2	7	7
Frame 1	0	0	0
Frame 2	1	1	1
Frame 3		2	2
Frame 4			
	# of Hits: 1 # of Misses: 4	# of Hits: 1 # of Misses: 4	# of Hits: 1 # of Misses: 4

Contoh

LRU Chain: 3 0 2 1 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	2	3	7
Frame 1	0	0	0
Frame 2	3	1	1
Frame 3		2	2
Frame 4			3
	# of Hits: 1 # of Misses: 5	# of Hits: 1 # of Misses: 5	# of Hits: 1 # of Misses: 5

Contoh

LRU Chain: 0 3 2 1 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	2	3	7
Frame 1	0	0	0
Frame 2	3	1	1
Frame 3		2	2
Frame 4			3
	# of Hits: 2 # of Misses: 5	# of Hits: 2 # of Misses: 5	# of Hits: 2 # of Misses: 5

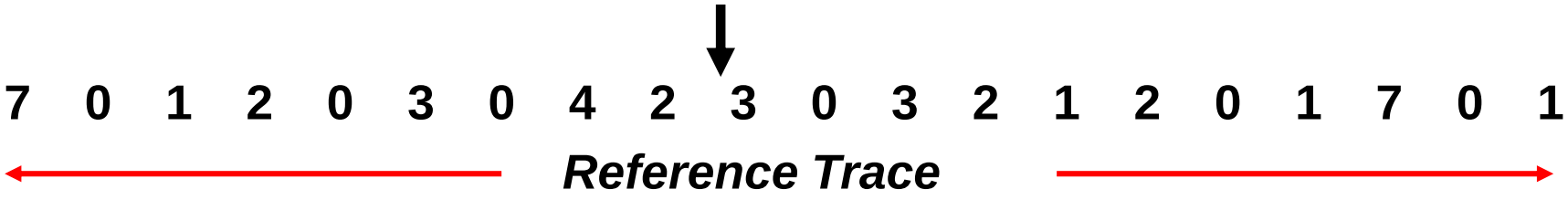
Contoh

LRU Chain: 4 0 3 2 1 7

	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	4	3	4
Frame 1	0	0	0
Frame 2	3	4	1
Frame 3		2	2
Frame 4			3
	# of Hits: 2 # of Misses: 6	# of Hits: 2 # of Misses: 6	# of Hits: 2 # of Misses: 6

Contoh

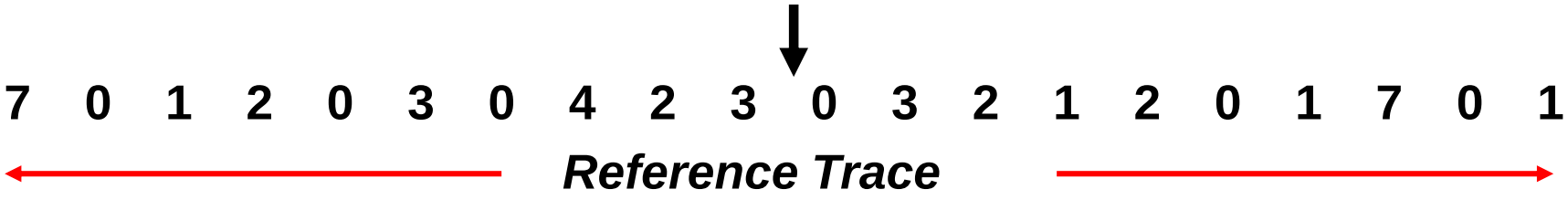
LRU Chain: 2 4 0 3 1 7



	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	4	3	4
Frame 1	0	0	0
Frame 2	2	4	1
Frame 3		2	2
Frame 4			3
	# of Hits: 2 # of Misses: 7	# of Hits: 3 # of Misses: 6	# of Hits: 3 # of Misses: 6

Contoh

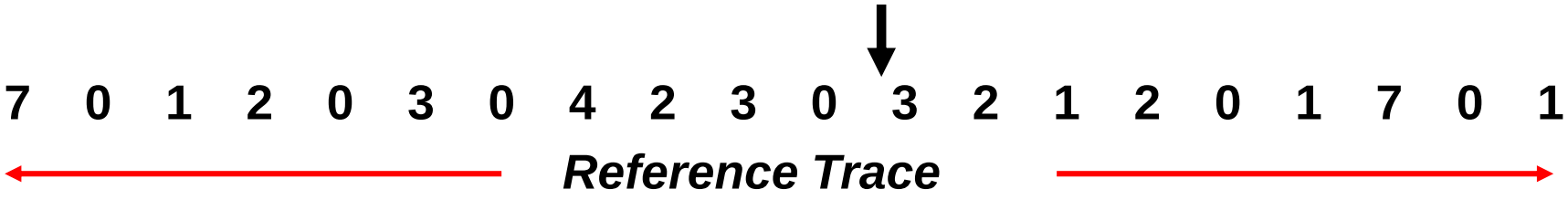
LRU Chain: 3 2 4 0 1 7



	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	4	3	4
Frame 1	3	0	0
Frame 2	2	4	1
Frame 3		2	2
Frame 4			3
	# of Hits: 2 # of Misses: 8	# of Hits: 4 # of Misses: 6	# of Hits: 4 # of Misses: 6

Contoh

LRU Chain: 0 3 2 4 1 7



	Cache X (size = 3)	Cache Y (size = 4)	Cache Z (size = 5)
Frame 0	0	3	4
Frame 1	3	0	0
Frame 2	2	4	1
Frame 3		2	2
Frame 4			3
	# of Hits: 2 # of Misses: 9	# of Hits: 5 # of Misses: 6	# of Hits: 5 # of Misses: 6

Pengamatan: Properti Stack

- Menambahkan ruang cache tidak pernah sakit, tetapi mungkin atau mungkin tidak membantu
  - Ini disebut sebagai *"Stack Property"*

- LRU memiliki properti stack, tetapi tidak semua kebijakan penggantian memilikinya
  - E.g., FIFO tidak memilikinya

Competing Workloads

- Apa yang terjadi jika beberapa beban kerja berjalan secara paralel, berbagi cache yang sama?
  - Thrashing (atau gangguan) akan muncul, berpotensi mencemari cache, terutama jika satu beban kerja adalah suatu *only-one-time scan*

- Bagaimana kita bisa mengisolasi efek dari gangguan?
  - Terapkan partisi statis (atau tetap), di mana cache diiris menjadi beberapa partisi tetap
    - Ini membutuhkan pengetahuan a priori tentang beban kerja
    - Dengan pengetahuan penuh sebelumnya, OPT dapat diterapkan!

  - Terapkan partisi dinamis, di mana cache diubah ukurannya secara adaptif berdasarkan pola akses yang berkembang dari beban kerja
    - Ini membutuhkan pemantauan dan pelacakan karakteristik beban kerja

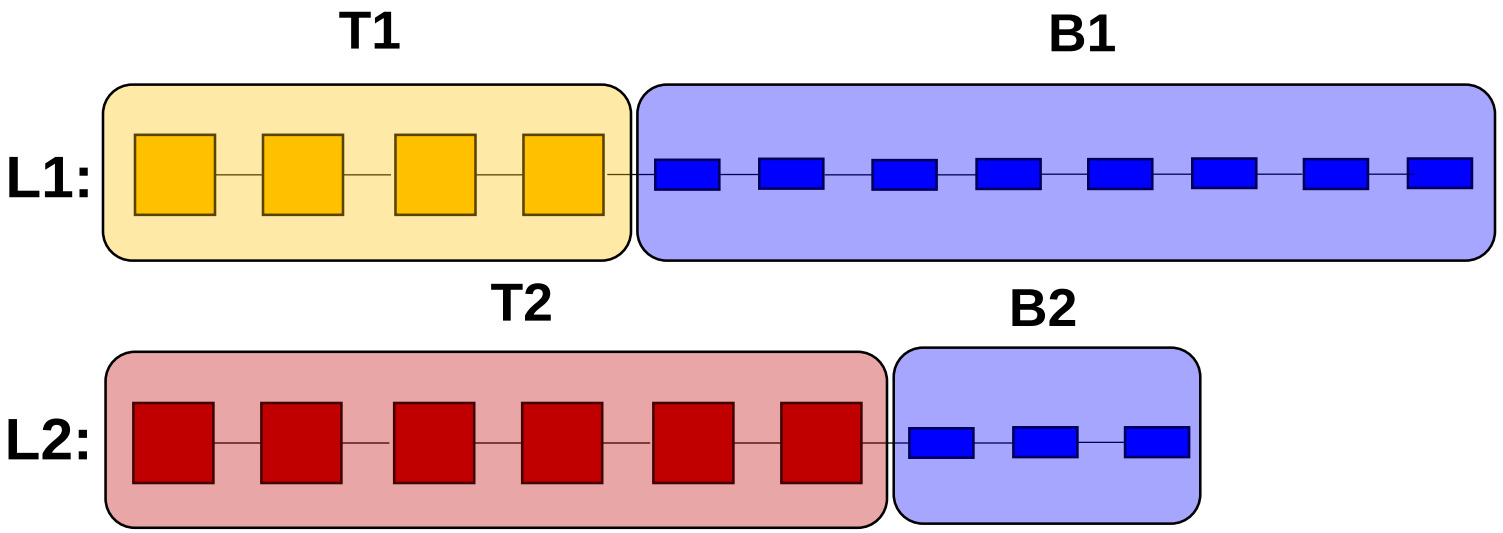
Adaptive Replacement Cache

Sebagai contoh cache yang menerapkan partisi dinamis, kita akan mempelajari :

***Adaptive Replacement Cache* (ARC)**

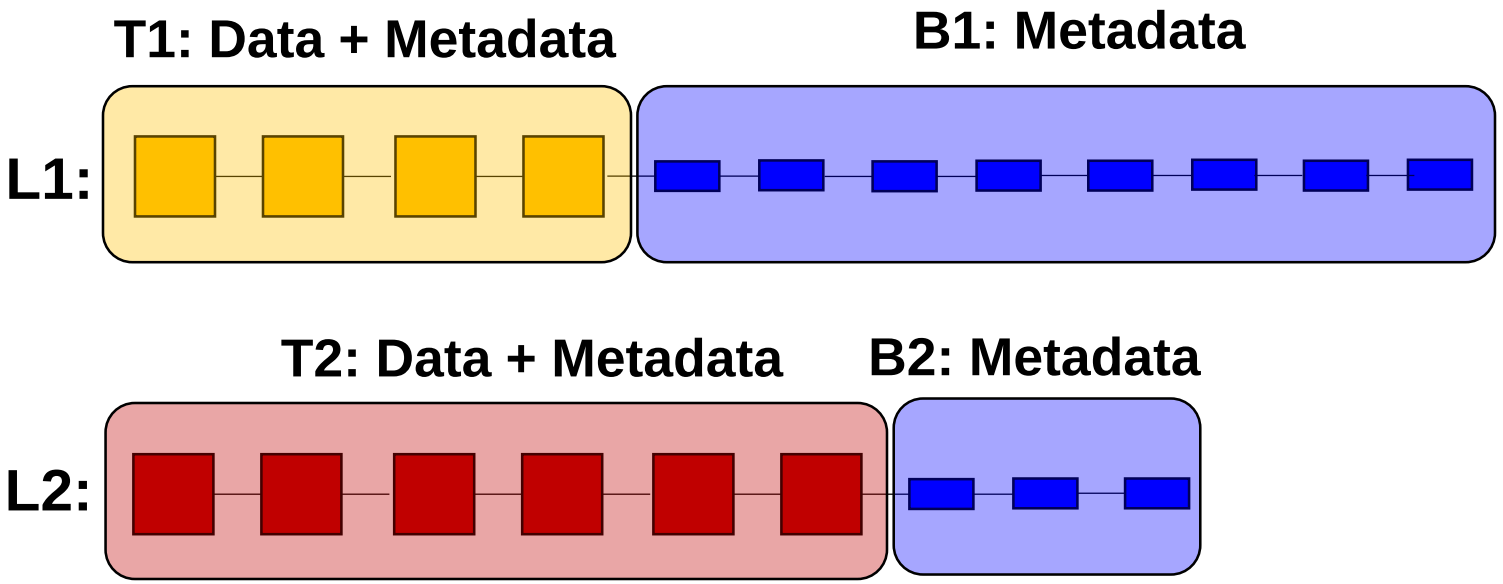
ARC Structure

- ARC membagi cache menjadi dua daftar LRU:
  - L1 = Bagian Atas (T1) + Bagian Bawah (B1)
  - L2 = Bagian Atas (T2) + Bagian Bawah (B2)



ARC Structure

- Content:
  - T1 dan T2 berisi objek dan sejarah yang di-cache
  - B1 dan B2 hanya berisi riwayat (mis., Kunci untuk objek yang di-cache)



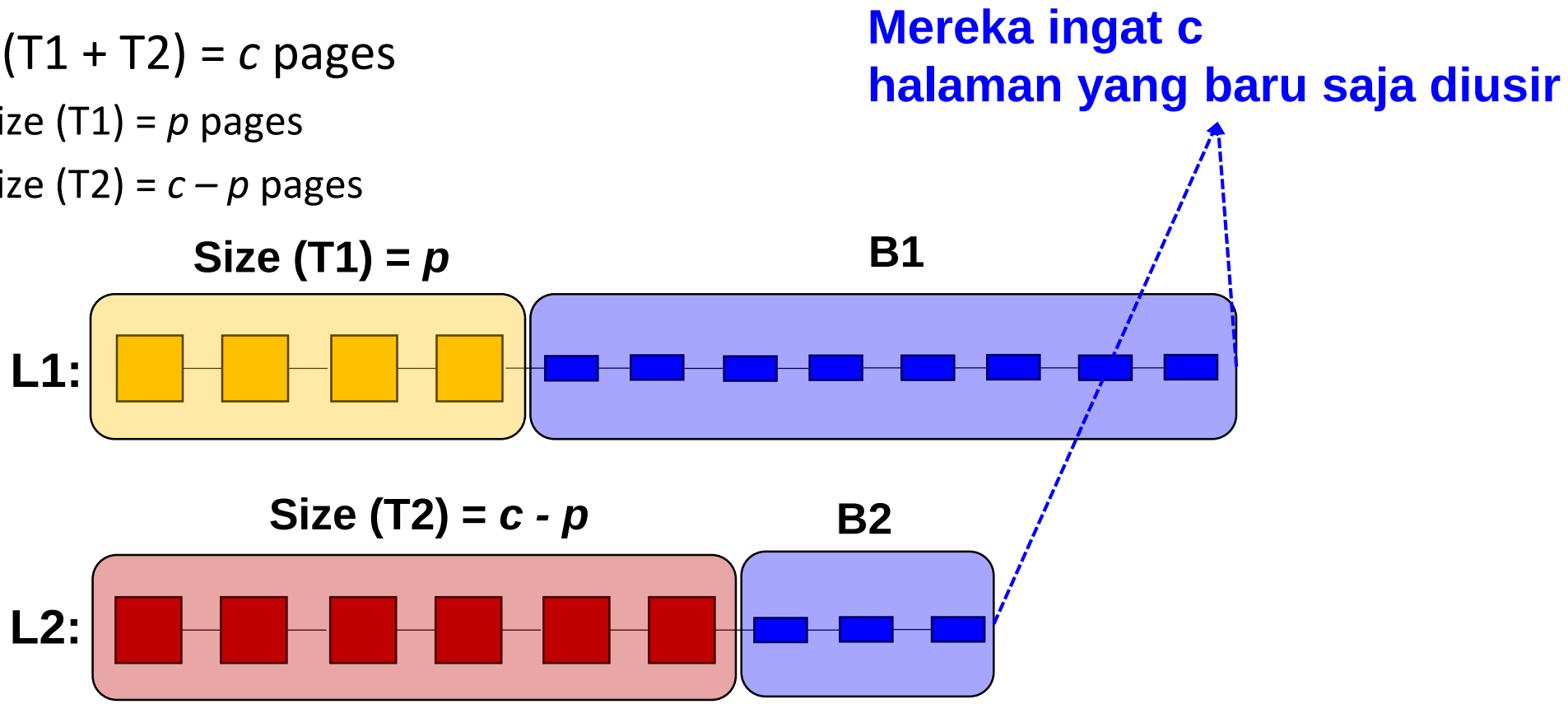
ARC Structure

- Content:
  - T1 dan T2 berisi objek dan sejarah yang di-cache
  - B1 dan B2 hanya berisi riwayat (mis., Kunci/key untuk objek yang di-cache)

- Bersama-sama, mereka ingat persis dua kali jumlah halaman yang sesuai dengan cache!
- Ini sangat membantu dalam menemukan pola halaman yang terusir (evicted)!

ARC Structure

- Sizes:
  - Size (T1 + T2) = *c* pages
    - Size (T1) = *p* pages
    - Size (T2) = *c* – *p* pages



ARC Policy

- Rules:
  - L1 meng-host halaman yang telah dilihat hanya sekali (*only once*)
    - L1 menangkap kebaruan (*recency*)
  - L2 meng-host halaman yang telah dilihat setidaknya dua kali (*at least twice*)
    - L2 menangkap frekuensi
- Key Idea:
  - Secara adaptif (sebagai respons terhadap karakteristik beban kerja yang diamati) tentukan berapa banyak halaman yang harus disimpan pada L1 versus L2
    - Ketika kebaruan mengganggu dengan frekuensi, ARC mendeteksi itu dan bertindak dengan cara yang menjaga lokalitas temporal di L2

ARC Policy: Details

- Untuk halaman yang diminta Q, satu dari empat kasus akan terjadi:
  - Case I: a hit in T1 or T2
    - Jika klik ada di T1, usir halaman LRU di T2 dan catat di B2
    - Pindahkan Q ke posisi MRU di T2
  - Case II: a miss in T1 U T2, but a hit in B1
    - Remove *Q*'s record at B1 and increase T1's size via increasing *p*
      - *Ini akan secara otomatis mengurangi T2 karena Ukuran (T2) = (c - p)*
    - Keluarkan halaman LRU di T2 dan catat di B2
    - Ambil Q dan letakkan di posisi MRU di T2

ARC Policy: Details

- Untuk halaman yang diminta Q, satu dari empat kasus akan terjadi:
  - Case III: a miss in T1 U T2, but a hit in B2
    - Remove *Q*'s record at B2 and increase T2's size via decreasing *p*
      - Ini akan secara otomatis mengurangi T1 karena Ukuran (T1) = p
    - Keluarkan halaman LRU di T2 dan catat di B2
    - Ambil Q dan letakkan di posisi MRU di T2
  - Case IV: a miss in T1 U B1 U T2 U B2
    - Keluarkan halaman LRU di T1 dan catat di B1
    - Ambil Q dan letakkan di posisi MRU di T1

Scan-Resistance of ARC

- Perhatikan bahwa halaman baru selalu ditempatkan di posisi MRU di T1

- Dari sana, secara bertahap menuju ke posisi LRU di T1
  - Kecuali jika digunakan sekali lagi sebelum penggusuran
  - Tetapi ini tidak akan terjadi dengan pemindaian satu kali saja
  - Karenanya, T2 tidak akan terpengaruh oleh pemindaian!

Scan-Resistance of ARC

- Ini membuat ARC scan-resistance, di mana T2 akan
  - Terisolasi secara efektif
  - Tumbuh dengan mengorbankan T1 karena lebih banyak hit akan terjadi pada B2 (yang menyebabkan peningkatan ukuran T2)
  - Menangani lokalitas temporal secara efektif, bahkan dengan beban kerja campuran (mis., Beban kerja dengan dan tanpa lokalitas berjalan bersamaan)

Pertemuan Selanjutnya…

- Replikasi Sisi Server