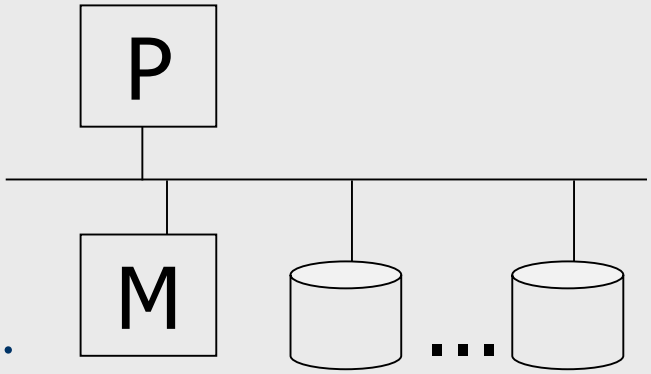
CURS 3

Baze de date distribuite

Introducere

- Sisteme de BD centralizate:
  - centralizarea blocărilor
  - dacă procesorul eşuează,

    întreg sistemul eşuează...



- Sisteme distribuite:
  - Procesoare (+ memorii) multiple
  - “Componente” autonome şi eterogene

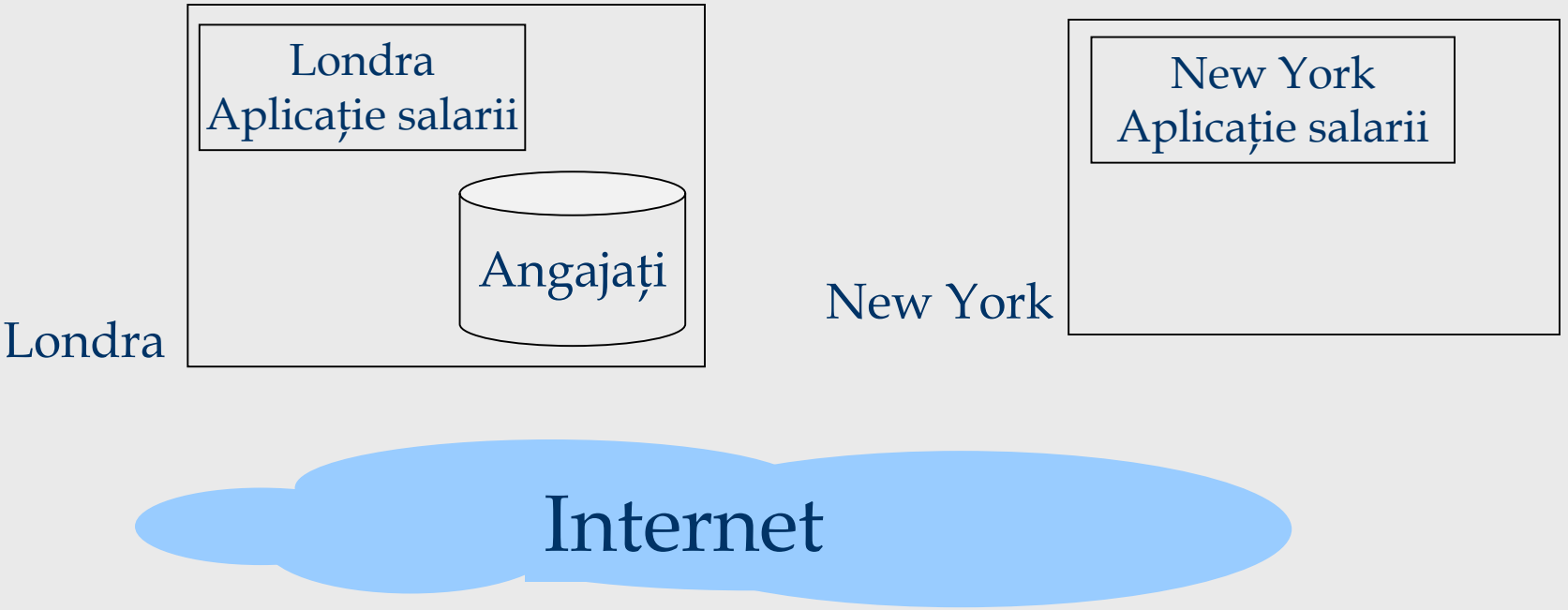
Baze de date distribuite

- Datele sunt stocate pe mai multe calculatoare, şi sunt gestionate de instanțe independente de SGBD-uri.
- Independența Datelor Distribuite: Locul unde sunt salvate datele este transparent utilizatorilor (extensie a principiilor de independență fizică şi logică a datelor).
- Atomicitatea Tranzacțiilor Distribuite: Utilizatorii pot implementa tranzacții ce accesează fragmente de date în mod similar cu programarea tranzacțiilor locale

De ce este nevoie de baze de date distribuite?

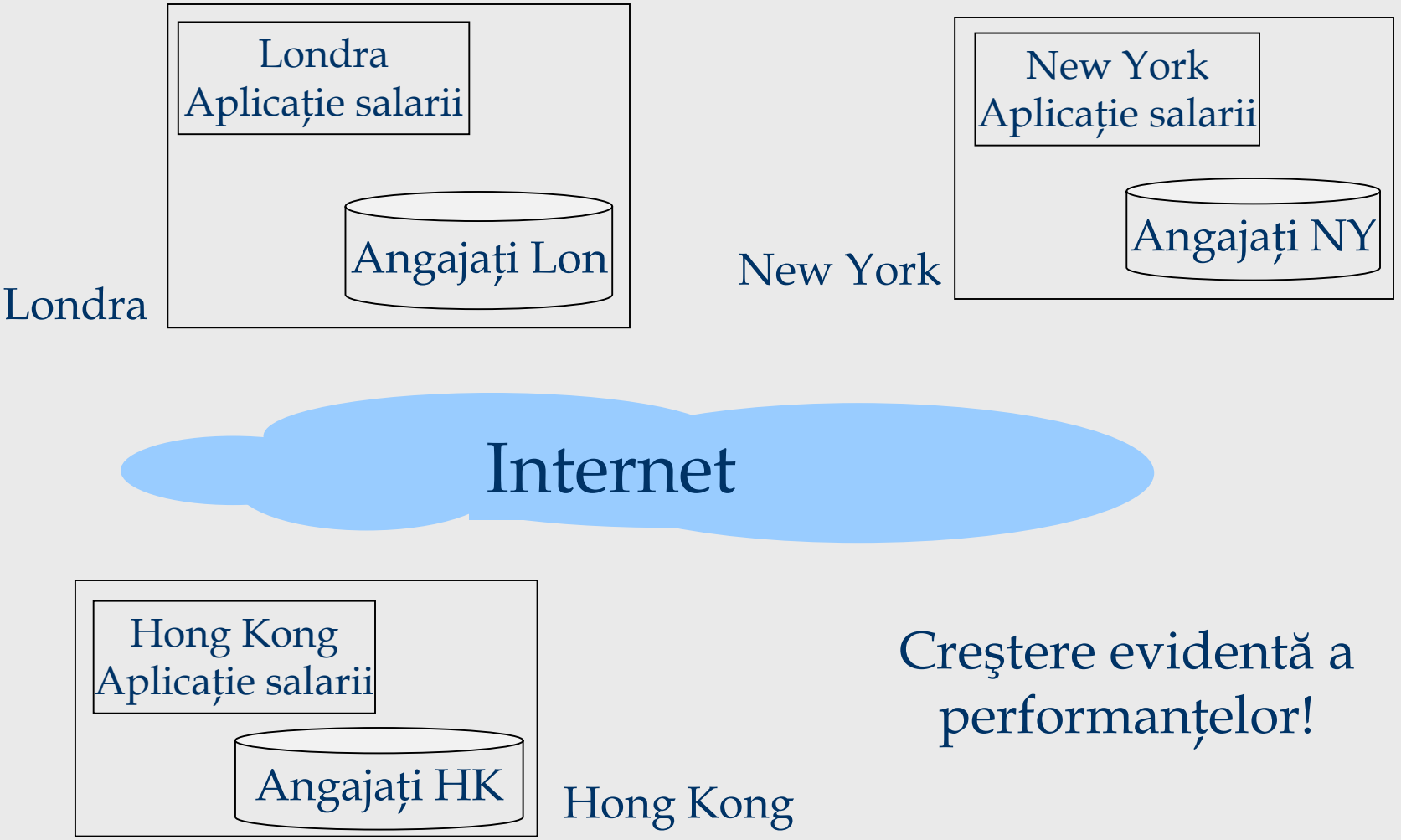
- Exemplu: Big Corp are birouri în London, New York şi Hong Kong.
- În general, datele unui angajat sunt gestionate de la biroul unde acest angajat lucrează
  - De ex. date legate de salarii, beneficii etc
- Periodic, Big Corp are nevoie de rapoarte ce conțin informații despre toți angajații săi
  - Ex. Calculul bonusului anual ce depinde de profitul global net.
- Unde ar trebui sa fie salvată baza de date de angajați?

De ce este nevoie de baze de date distribuite?



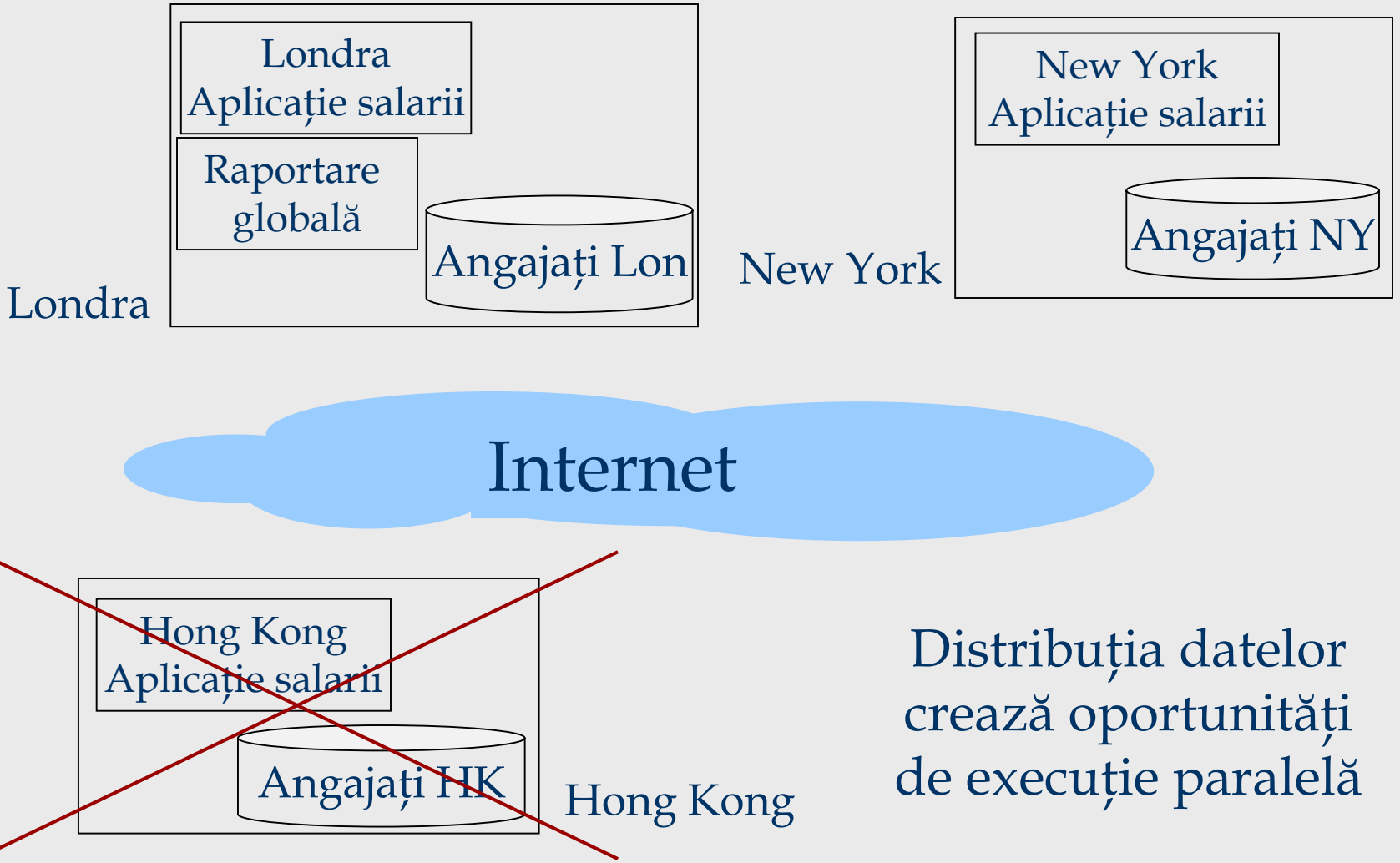
Aplicațiile de salarii din NY şi HK au performanțe reduse!

De ce este nevoie de baze de date distribuite?



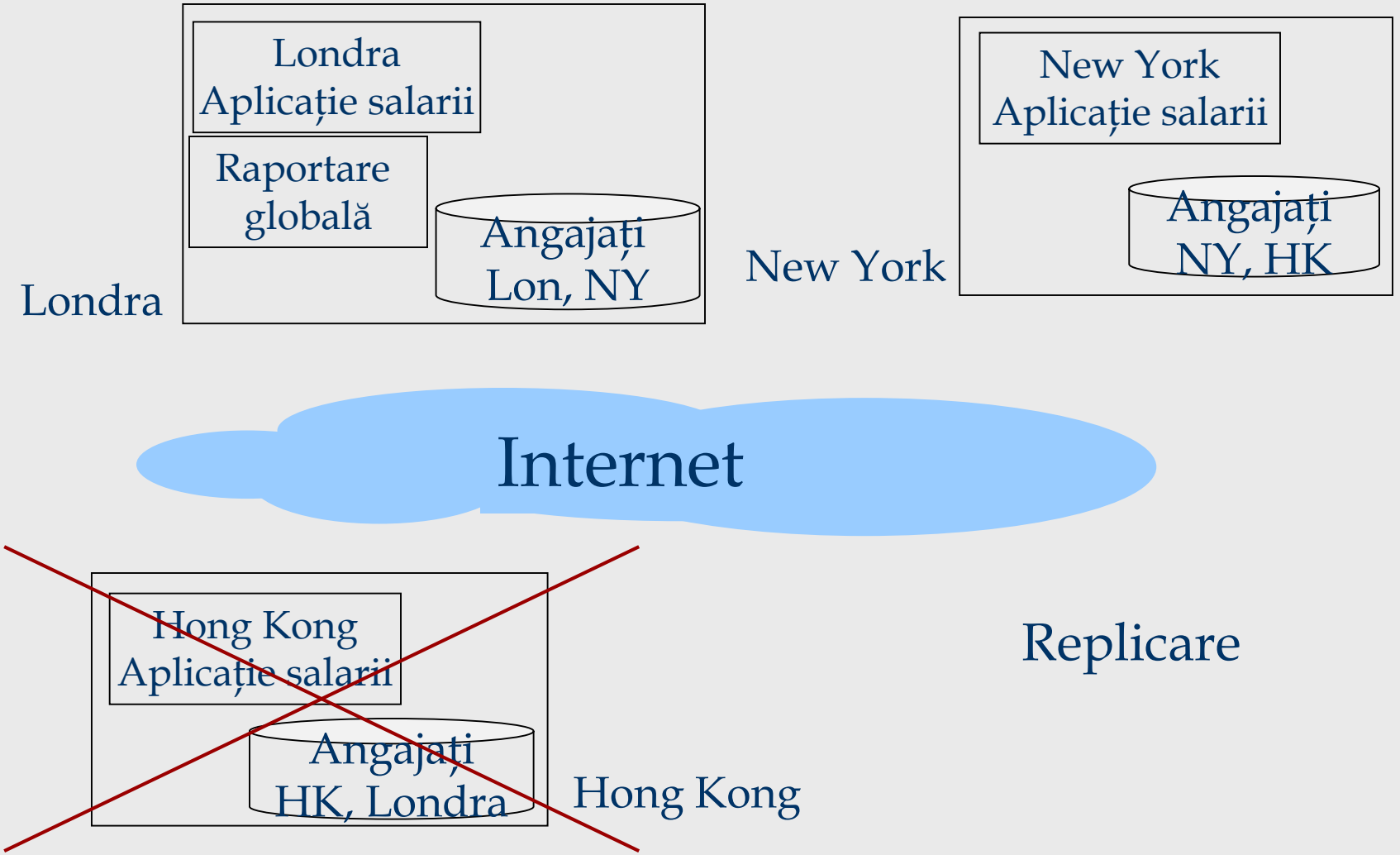
Creştere evidentă a performanțelor!

De ce este nevoie de baze de date distribuite?



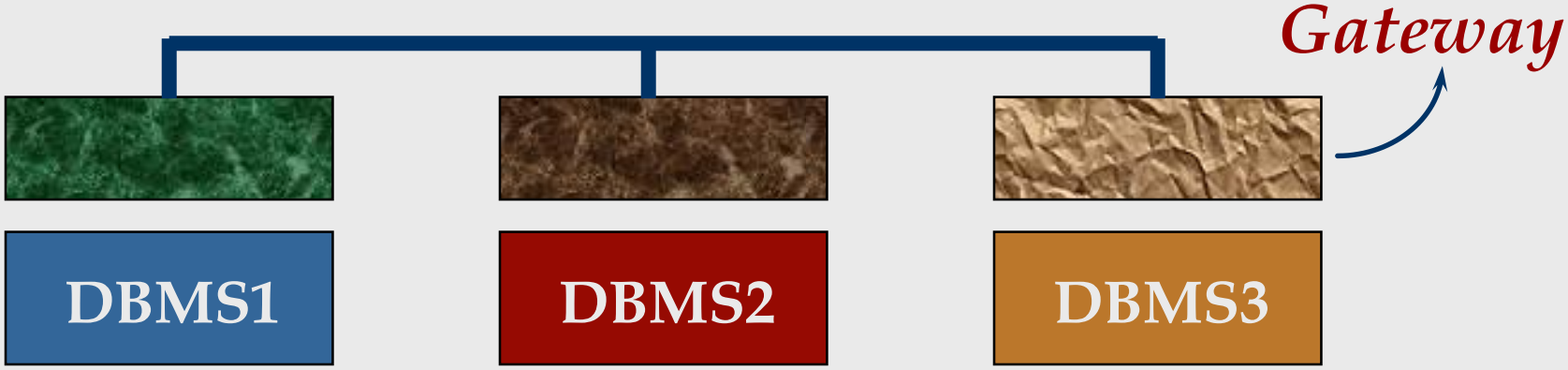
Distribuția datelor crează oportunități de execuție paralelă

De ce este nevoie de baze de date distribuite?



Tipuri de baze de date distribuite

- Omogene: Pe fiecare *site* rulează acelaşi tip de SGBD.
- Eterogene: *Site*-uri diferite rulează tipuri diferite de SGBD (relaționale sau nu).



În acest context apare noțiunea de *gateway*, care reprezintă o componentă *software* ce acceptă cereri (într-un SQL standard), transmite cererile către SGBD-urile locale, şi returnează răspunsul (din nou, într-un format standard)

Provocări ale bazelor de date distribuite

Proiectarea bazelor de date distribuite

- Se decide locul unde sunt stocate părți ale bazei de date
- Depinde de modul de accesare a datelor al majorității aplicațiilor
- Două sub-probleme: fragmentarea & alocarea

Procesarea interogărilor distribuite

- Scopul alegerii unui plan de execuție contralizat: minimizarea numărului de accesări ale discului (cost)
- Factori suplimentari ce influențează estimarea costului unui plan:
  - Costuri de comunicare
  - Oportunitatea procesării paralele
- Mulțimea planurilor de execuție posibile e mai mare!

Provocări ale bazelor de date distribuite

Controlul concurenței

- Planificările tranzacțiilor trebuie sa fie serializabile la nivel global
- Gestiunea *deadlock*-urilor între tranzacții executate pe *site*-uri distincte
- Copii multiple ale datelor → propagarea modificărilor

Păstrarea consistenței bazelor de date

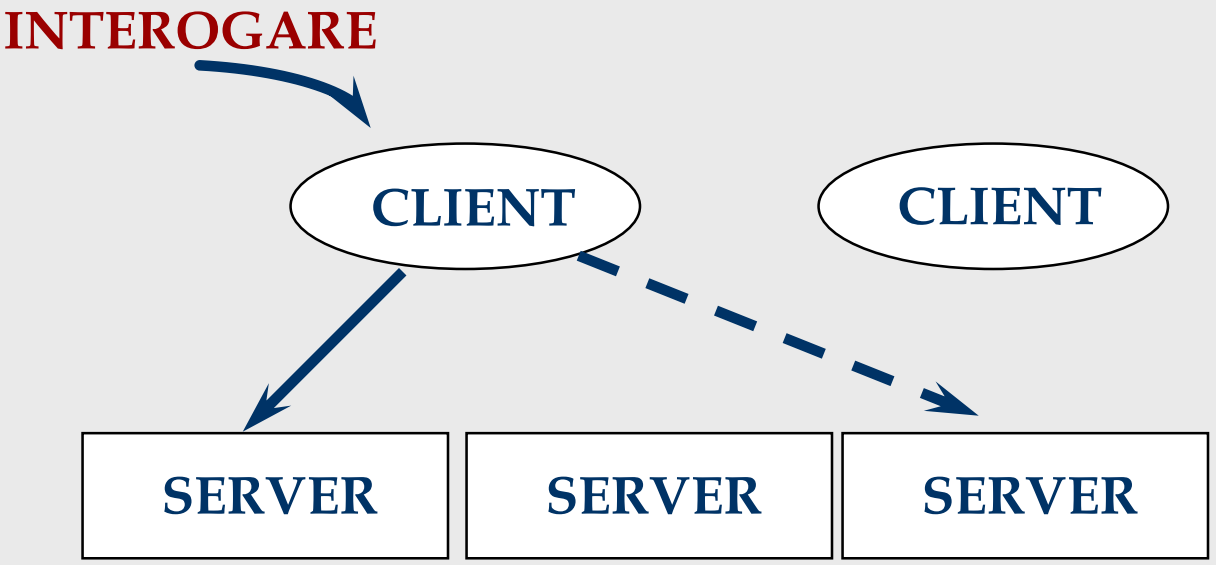
- Modalitățile prin care execuția anumitor tranzacții poate eşua sunt mult mai diverse:
  - Unul sau mai multe procesoare pot eşua
  - Rețeaua poate fi blocată sau parțial funcțională
- Datele trebuie sincronizate

Arhitecturi de SGBD distribuite

- *Client-Server*

Clientul transmite interogările către un singur *site*. Toate interogările sunt procesate pe *server*.

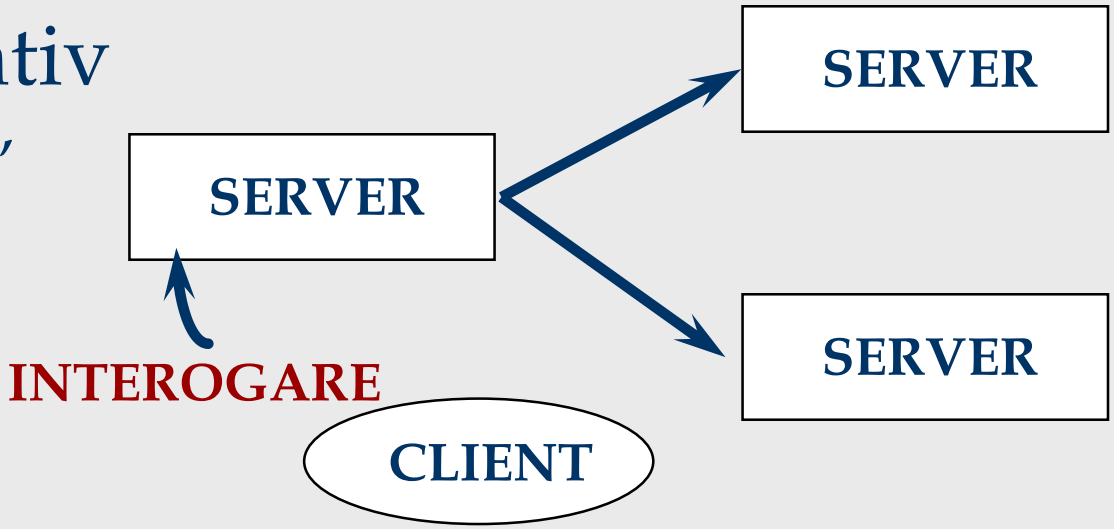
- Clienți *thin* şi *fat*.
- Comunicarea orientată pe mulțimi de date



Arhitecturi de SGBD distribuite

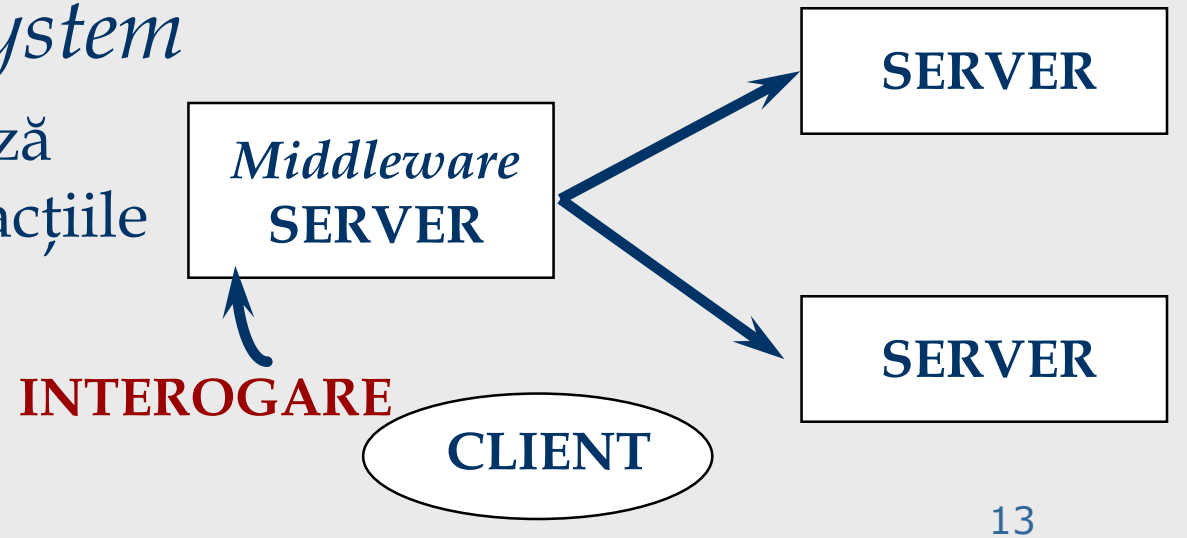
- Server colaborativ

Interogările "acoperă" mai multe *site*-uri



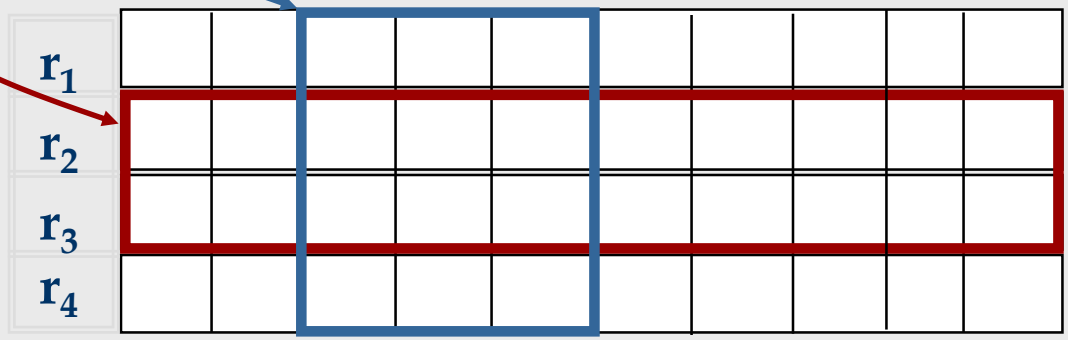
- *Middleware System*

Un server gestionează interogările şi tranzacțiile executate pe servere multiple



Stocarea datelor

- Fragmentare: "*spargerea*" unei tabele în tabele mai mici (fragmente), şi stocarea acestora (în locul tabelei inițiale)
  - *Orizontală*
    - **Primară**: depinde de câmpurile tabelei
    - **Derived**: depinde de legătura cu alte tabele
  - *Verticală*



Stocarea datelor

- Proprietățile (dorite ale) fragmentării

$$R \Rightarrow \mathbf{F} = \{F_1, F_2, \ldots, F_n\}$$

Completitudine

Pentru fiecare $x \in R$, $\exists F_i \in \mathbf{F}$ astfel încât $x \in F_i$

Disjunctivitate

$\forall x \in F_i$, $\neg \exists F_j$ astfel încât $x \in F_j$, $i \neq j$

Reconstrucție

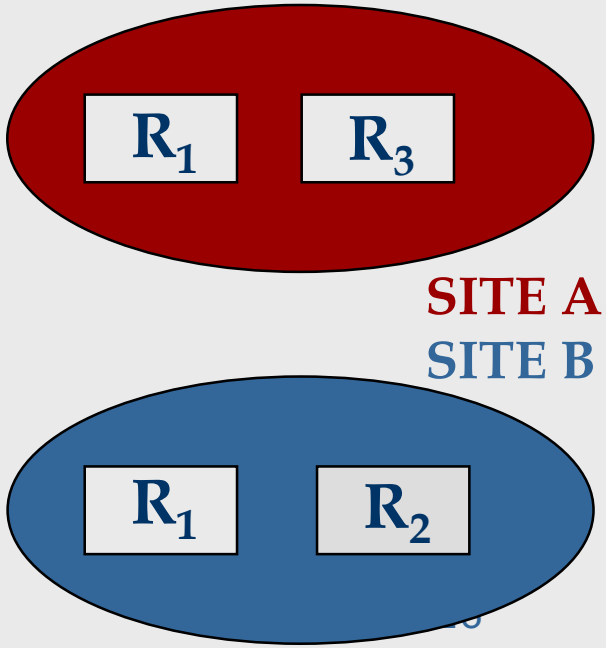
Există o funcție g astfel încât

$$R = g(F_1, F_2, \ldots, F_n)$$

Stocarea datelor

- Replicare: stocarea copiilor unei tabele sau a unui fragment de tabelă. O tabelă poate fi replicată pe unul sau mai multe site-uri.
  - Creşte disponibilitatea datelor.
  - Evaluarea interogărilor e mai rapidă (se pot utiliza copii locale)
  - Sincron vs. Asincron

R e fragmentat în R_1, R_2, R_3
R_1 e replicat pe ambele site-uri



Catalog distribuit

- Ține evidența modului în care datele sunt distribuite pe diverse *site*-uri
- Replica fiecărui fragment are un nume unic la nivel global, compus din:
  - <nume-local, site-origine, id_replica>
- Catalog: Descrie toate obiectele (fragmente, replici) aflate pe un *site* + ține evidența tuturor replicilor tabelelor create pe acel *site*
  - Pentru găsirea unei tabele se va consulta catalogul *site*-ului unde această tabelă a fost creată
  - Site-ul de origine al unei tabele nu se modifică, chiar dacă tabela se mută ulterior

Actualizarea datelor distribuite

- Replicare sincronă: Toate copiile unei tabele modificate de o tranzacție trebuie să fie actualizate înainte ca tranzacția să se comită.
  - Distribuirea datelor e transparentă utilizatorilor.
- Replicare asincronă: Copiile unei tabele modificate sunt actualizate doar periodic
  - Utilizatorii sunt conştienți de faptul că datele sunt distribuite.
  - Multe dintre produsele curente urmează această abordare

Replicare sincronă

- Există 2 tehnici de bază ce garantează că tranzacțiile "văd" aceeaşi valoare indiferent de copia pe care o accesează.
- Votare: tranzacția trebuie să modifice o majoritate de copii ale unui obiect; de asemenea trebuie citite suficiente copii pentru a se asigura accesul la una dintre copiile recente.
  - Ex. 10 copii; 7 actualizate la modificări; 4 copii citite.
  - Fiecare copie are un număr de versiune.
  - Citirile fiind activități comune, nu e o abordare des utilizată.
- Citeşte-orice Modifică-tot: Modificările sunt mai lente şi citirile sunt mai rapide în comparație cu tehnica votării.
  - Cea mai utilizată metodă de sincronizare a replicărilor.
- Alegerea tehnicii determină *ce* blocări sunt utilizate

Costul replicării sincrone

- Înainte ca o tranzacție ce face o modificare să fie comisă, aceasta va trebui să blocheze toate copiile tabelei/fragmentului modificat.
  - Se transmit cereri de blocare către diverse site-uri, iar până la primirea răspunsului se mențin alte blocări!
  - Dacă rețeaua/site-urile eşuează, tranzacția nu se poate comite până ce acestea nu-şi revin.
  - Chiar şi în absența unor eşuări, ***protocolul de comitere*** poate fi costisitor, cu multe mesaje
- Alternativa *replicării asincrone* este, de aceea, mai utilizată.

Replicare asincronă

- Permite ca tranzacțiile să fie comise înainte ca toate copiile sa fie actualizate (şi citirile se fac folosind o singură copie).
  - Utilizatorii trebuie sa fie conştienți ce copie citesc şi de faptul că, pentru o scurtă perioadă de timp, copiile pot să fie desincronizate.
- Două abordări: Site Principal şi Peer-to-Peer
  - Diferența constă în numărul de copii ``actualizabile'' sau ``master''.

Replicare *Peer-to-Peer*

- Mai multe copii ale unui obiect pot fi *master* în această abordare.
- Modificările unei copii *master* trebuie să fie propagate către celelalte copii.
- Trebuie rezolvat conflicte ce apar atunci când două copii *master* sunt modificate (conflict: Site 1: vârsta lui Joe se modifică la 35; Site 2: la 36)
- E cea mai bună abordare în cazurile când nu pot apărea conflicte:
  - Ex: fiecare site *master* deține un fragment disjunct.
  - Ex: Drepturile de actualizare sunt deținute de un singur *master* la un moment dat

Replicare cu *site* principal

- Doar o copie a unei tabele este considerată copie primară sau *master*. Replicile facute pe alte site-ri nu pot să fie modificate direct.
  - Copia primară este publicată.
  - Celelalte *site*-uri subscriu la această copie; ele se numesc copii secundare.
- Cum se propagă modificările dinpre copia primară către copiile secundare?
  - În doi paşi: mai întâi se capturează modificările făcute de tranzacţiile comise apoi se aplică aceste modificări

Implementarea etapei Capture

Pe bază de log

- logul (menținut pentru recuperare) se utilizează la generarea structurii *Change Data Table* (CDT)
- modificările tranzacțiilor care se anulează trebuie înlăturate din CDT
- în final, CDT conține doar înregistrările log de tip update ale tranzacțiilor comise

Procedural

- captarea este realizată de o procedură invocată automat (e.g., un *trigger*); aceasta realizează un *snapshot* al copiei primare

Implementarea etapei Capture

Captarea *pe bază de log* este mai bună

(mai puțin costisitoare, mai rapidă),

dar se bazează pe unele particularități ale logului specifice sistemului

Implementarea etapei Apply

Etapa *Apply* aplică schimbările captate în faza anterioară (în CDT sau *snapshot*) cópiilor secundare

- *site-ul* primar poate trimite continuu CDT

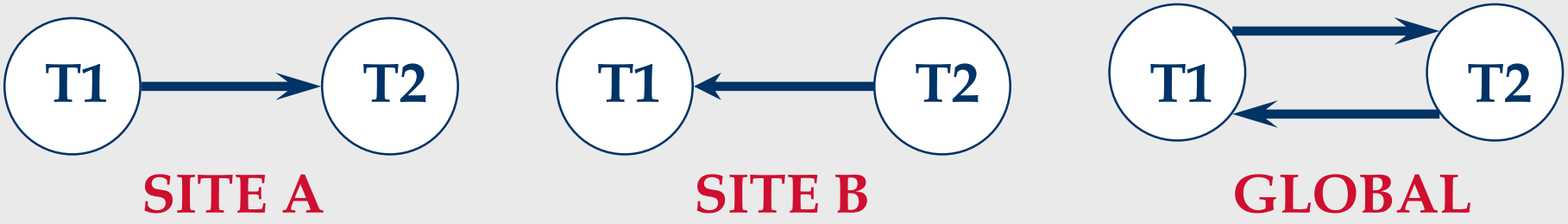
sau

- site-ul secundar poate solicita periodic (ultima porțiune din) CDT sau un *snapshot* de la site-ul primar; intervalul dintre solicitări poate fi controlat de un *timer* sau din aplicație
- pe fiecare *site* secundar rulează o copie a procesului Apply

Blocare distribuită

- Centralizat
  - vulnerabilitate ridicată – depinde de un *site*.
- Copie principală
  - Citirea unui obiect necesită acces atât la *site*-ul principal cât şi la *site*-ul unde e salvat obiectul
- Complet distribuit
  - Modificarea unui obiect presupune blocarea tuturor site-urilor unde se găseşte obiectul respectiv

Detectarea *deadlock*-urilor distribuite



Centralizat (grafurile locale sunt trimise către un singur *site*);

Ierarhic (organizare ierarhică a *site*-urilor, grafurile locale sunt transmise părintelui);

Timeout (tranzacțiile sunt întrerupte dacă durează prea mult).