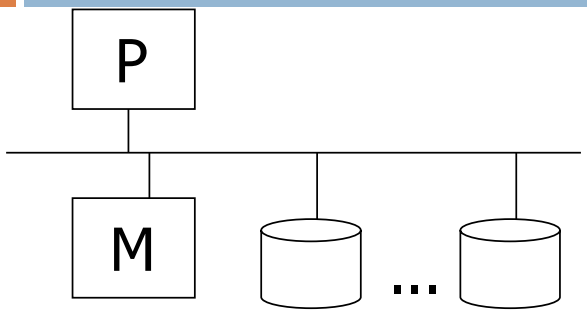
BANCOS DE DADOS DISTRIBUÍDOS

A ALTERNATIVA CORRETA AO BLOCKCHAIN
QUANDO NÃO HÁ CONSENSO DISTRIBUÍDO

Prof. Dr. Bruno de Carvalho Albertini
Curso Blockchain Developer

Bancos de Dados Centralizado



Software:

Application
SQL Front End
Query Processor
Transaction Proc.
File Access

- Simplificações:
  - Front end único
  - Só um local para manter dados, locks, etc.
  - Se o servidor falhar...

Introdução

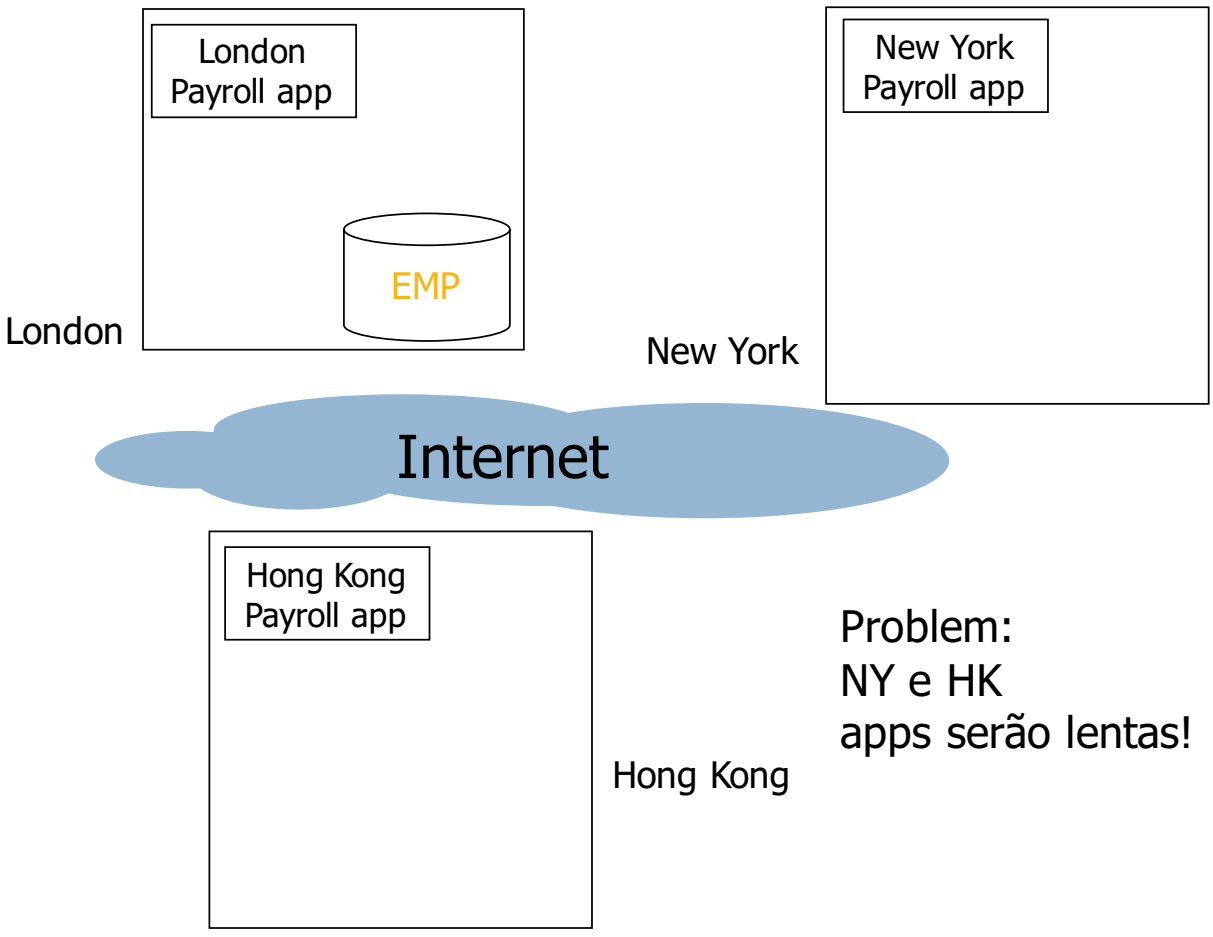
- Replicação: usar múltiplas cópias do banco de dados (réplicas) para aumentar a disponibilidade e o desempenho
- Problemas
  - Desempenho inferior: as atualizações devem ser realizadas em todas as réplicas sincronamente
  - Indisponibilidade: alguns algoritmos precisam que quase todas as réplicas estejam "vivas" para operarem

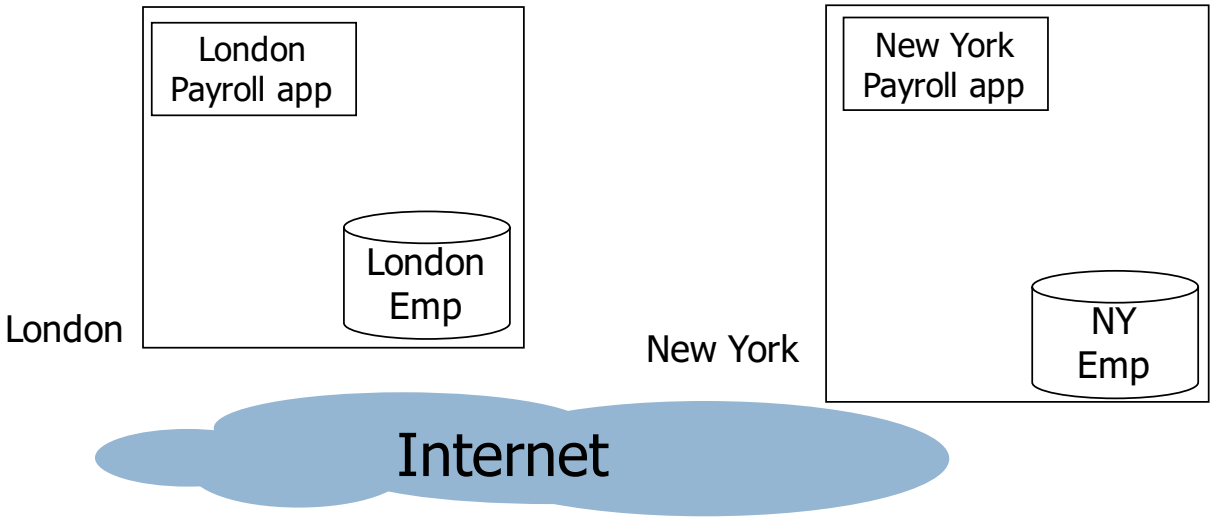
Por que precisamos de sistemas distribuídos?

- Imagine uma empresa global com escritórios em 7 países
- Dados dos empregados
  - EMP(ENO, NAME, TITLE, SALARY, …)
- Onde essa tabela deve ser mantida?

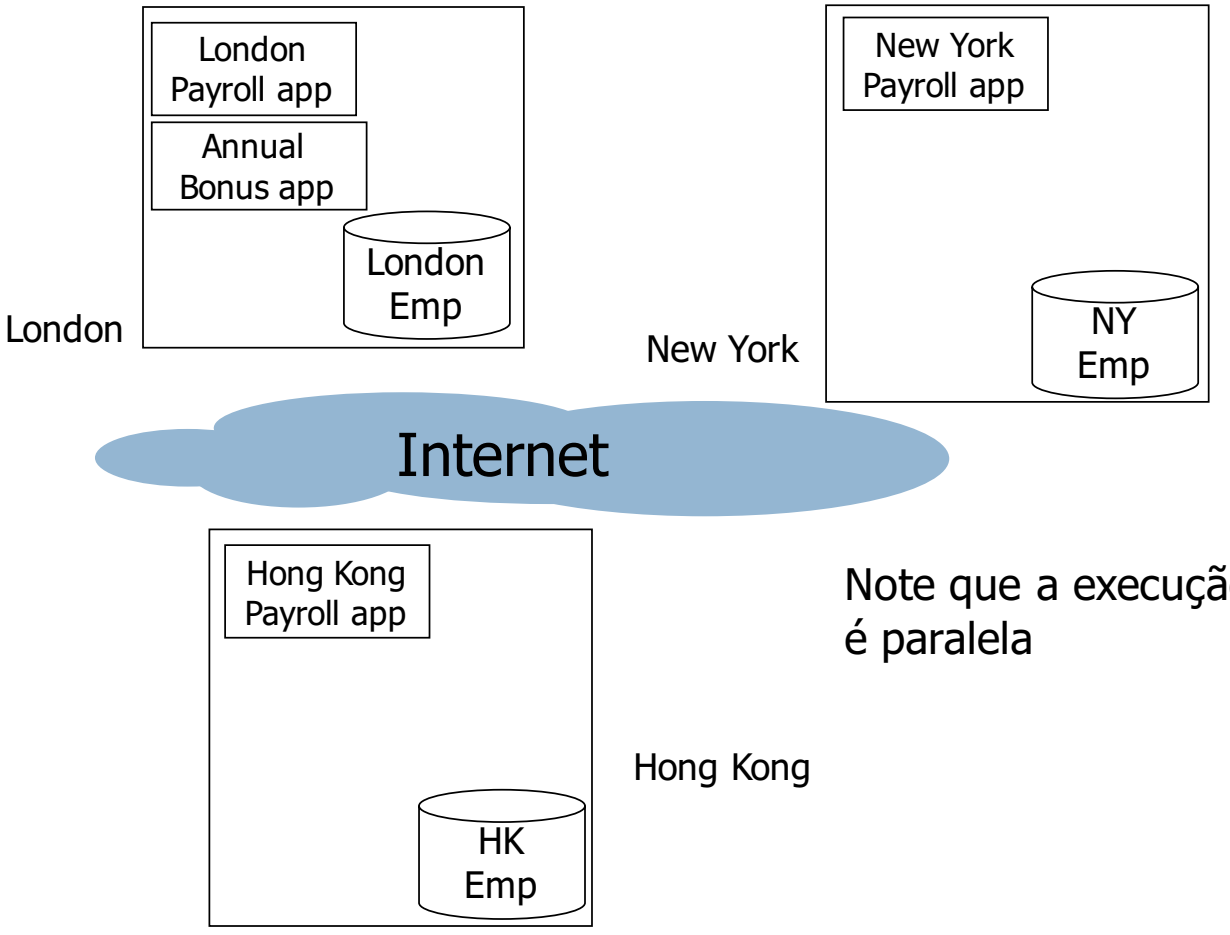
Padrão de acesso aos dados

- Na maioria das vezes, os dados dos empregados são acessados no país onde o empregado trabalha
  - E.g., folha de pagamento, benefícios, etc.
  - Periodicamente, a matriz quer um relatório consolidado
    - E.g., pagamento de PL global

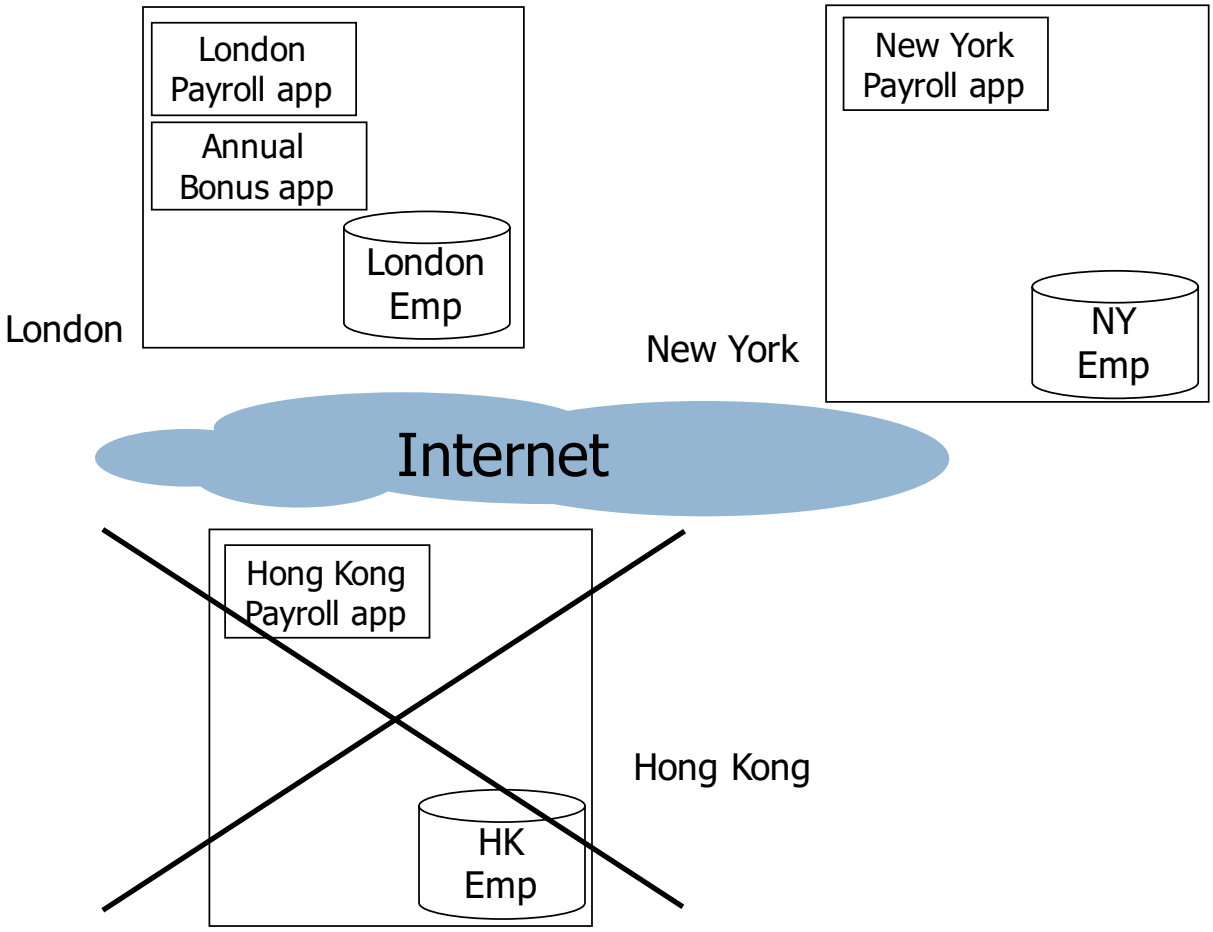


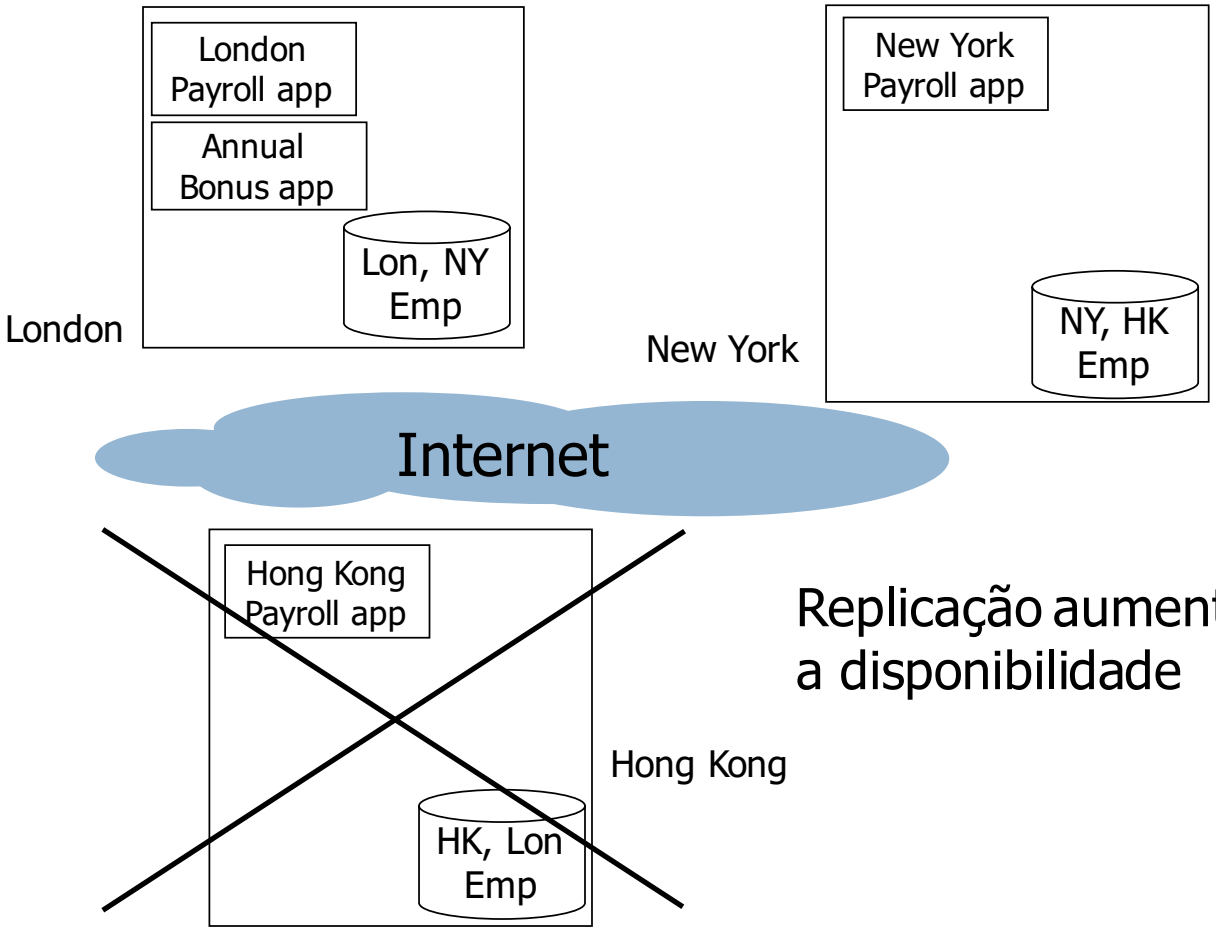


Melhor...



Note que a execução é paralela





Replicação aumenta a disponibilidade

Definições

- **BD Distribuído:** Um único DB lógico espalhado fisicamente em diversos nós conectados de alguma forma
- **DB Descentralizado:** Uma coleção de DBs independentes (não necessariamente conectados)

Não é a mesma coisa!!!

Bancos Distribuídos

- **Homogêneos**
  - Todos na rede tem o mesmo software
  - Conhecem uns aos outros e concordam com o que fazem
  - Aparenta ser um sistema monousuário
- **Heterogêneos**
  - Cada um na rede pode ter um SW
    - Armazenam de maneira diferente
    - Processam pedidos de processmento diferentemente
  - Pode ser que um usuário não saiba da existência do outro

BC: heterogênea por natureza, na concepção!

Figure 13-1 – Distributed database environments (adapted from Bell and Grimson, 1992)

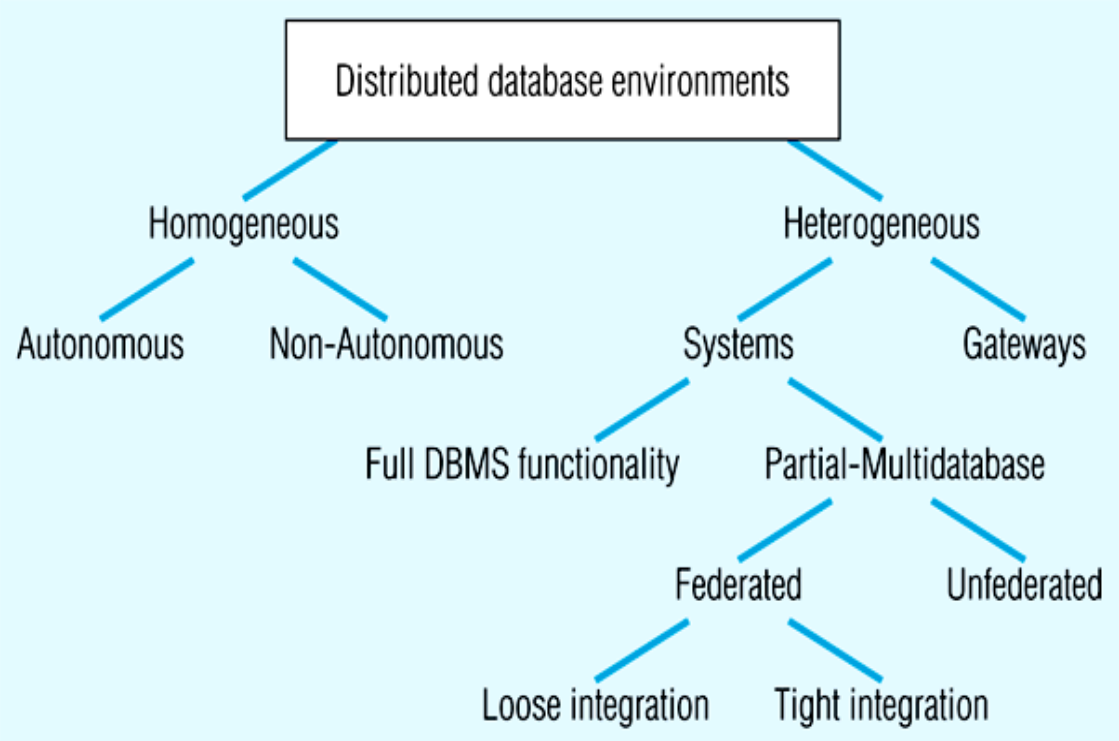


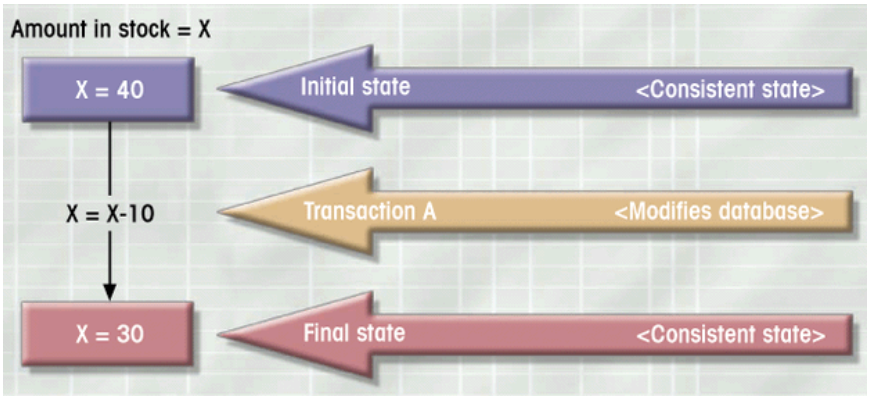
Table 13-1: Distributed Design Strategies

Table 13-1 Comparison of Distributed Database Design Strategies

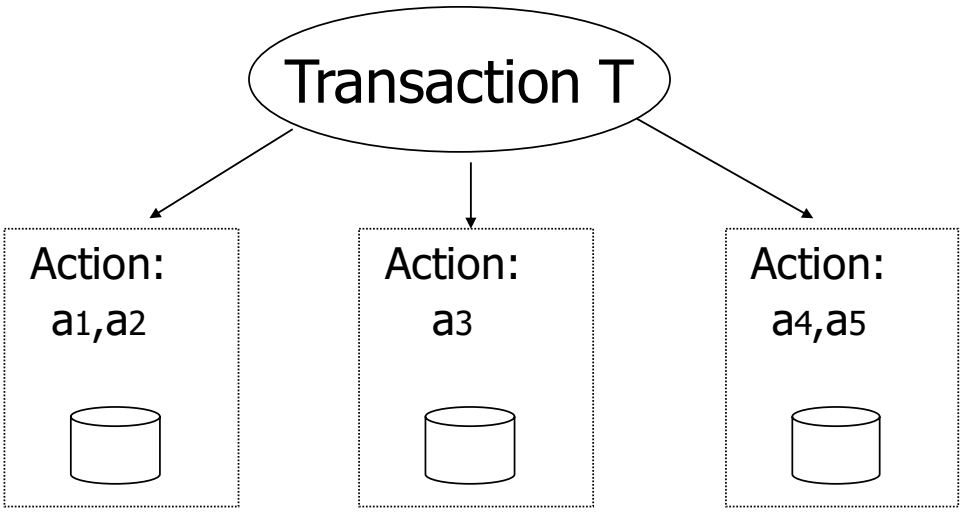
Strategy	*Reliability*	*Expandability*	*Communications Overhead*	*Manageability*	*Data Consistency*
Centralized	**POOR:** Highly dependent on central server	**POOR:** Limitations are barriers to performance	**VERY HIGH:** High traffic to one site	**VERY GOOD:** One, monolithic site requires little coordination	**EXCELLENT:** All users always have same data
Replicated with snapshots	**GOOD:** Redundancy and tolerated delays	**VERY GOOD:** Cost of additional copies may be less than linear	**LOW to MEDIUM:** Not constant, but periodic snapshots can cause bursts of network traffic	**VERY GOOD:** Each copy is like every other one	**MEDIUM:** Fine as long as delays are tolerated by business needs
Synchronized replication	**EXCELLENT:** Redundancy and minimal delays	**VERY GOOD:** Cost of additional copies may be low and synchronization work only linear	**MEDIUM:** Messages are constant, but some delays are tolerated	**MEDIUM:** Collisions add some complexity to manageability	**MEDIUM to VERY GOOD:** Close to precise consistency
Integrated partitions	**VERY GOOD:** Effective use of partitioning and redundancy	**VERY GOOD:** New nodes get only data they need without changes in overall database design	**LOW to MEDIUM:** Most queries are local but queries which require data from multiple sites can cause a temporary load	**DIFFICULT:** Especially difficult for queries that need data from distributed tables, and updates must be tightly coordinated	**VERY POOR:** Considerable effort, and inconsistencies not tolerated
Decentralized with independent partitions	**GOOD:** Depends on only local database availability	**GOOD:** New sites independent of existing ones	**LOW:** Little if any need to pass data or queries across the network (if one exists)	**VERY GOOD:** Easy for each site, until there is a need to share data across sites	**LOW:** No guarantees of consistency, in fact pretty sure of inconsistency

Transações Distribuídas

- Série de passos a serem completados por um DBMS para realizar uma tarefa
- Atômica (ou todos são completados ou nenhum)
- O DB não deve assumir estados intermediários

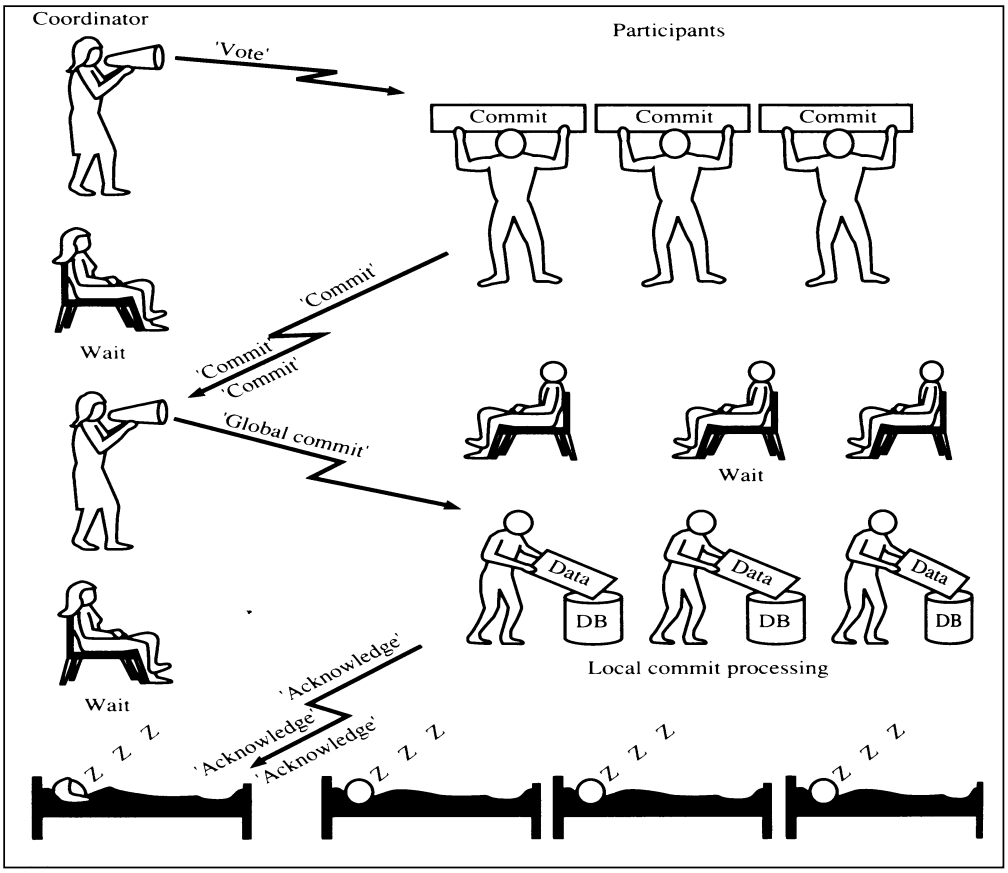


Problema do *commit* distribuído

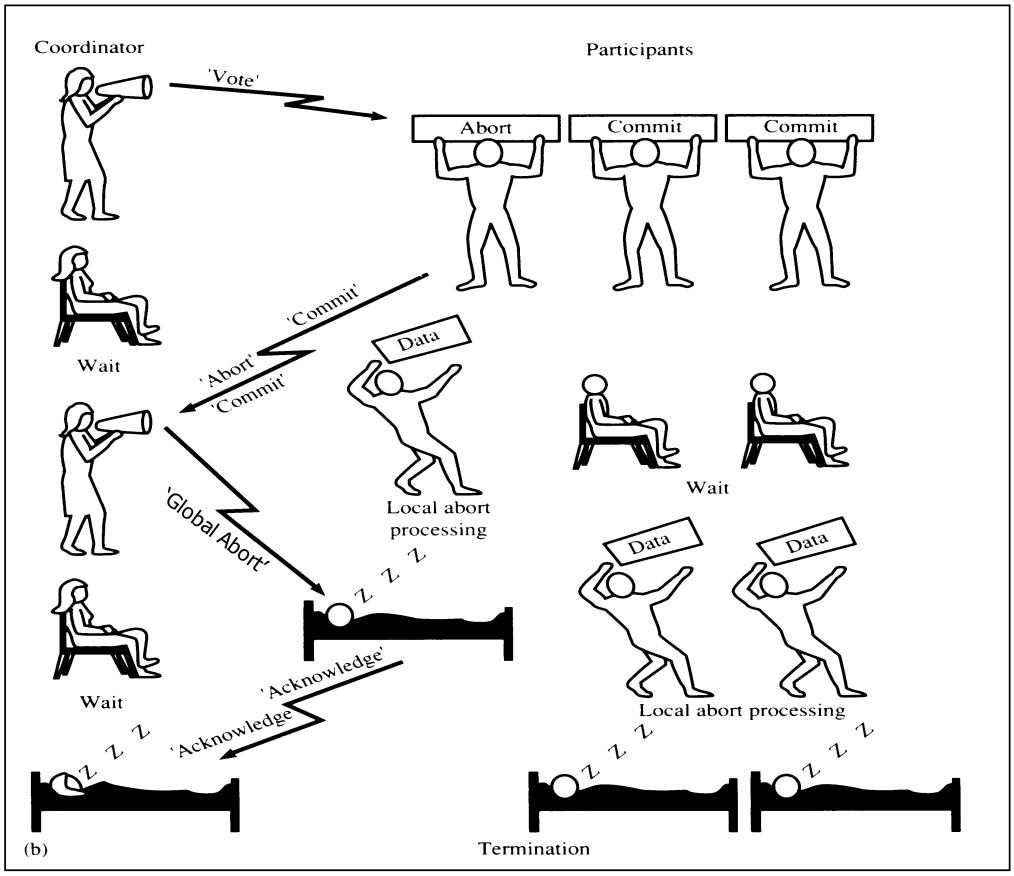


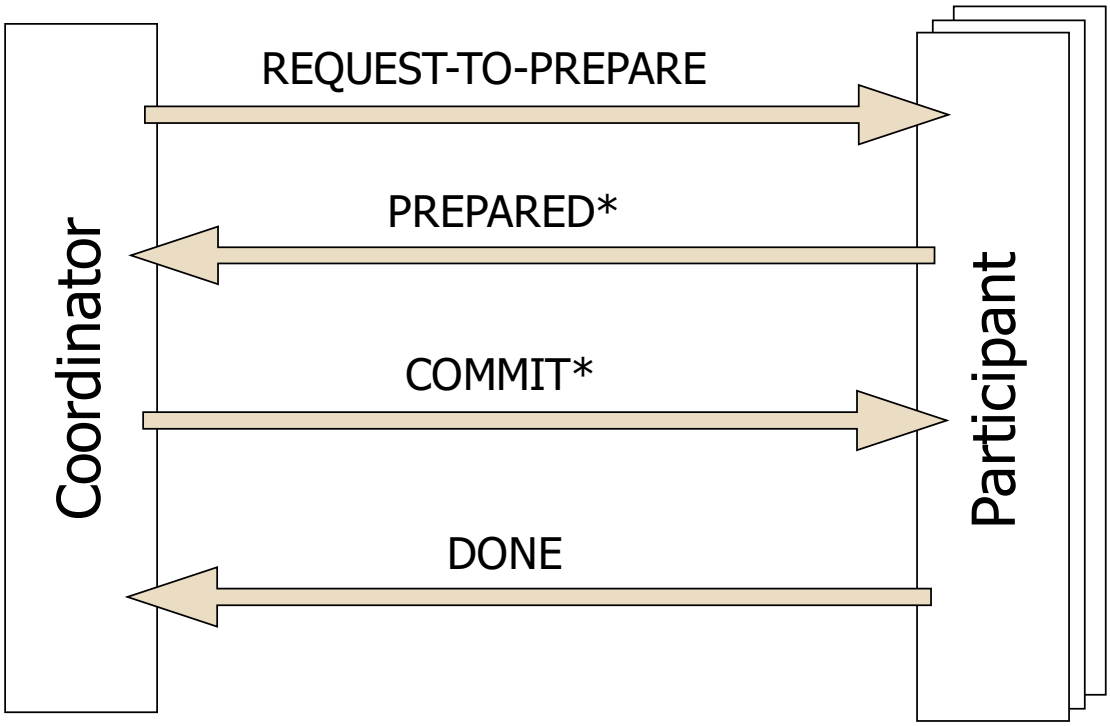
O *commit* deve ser atômico!

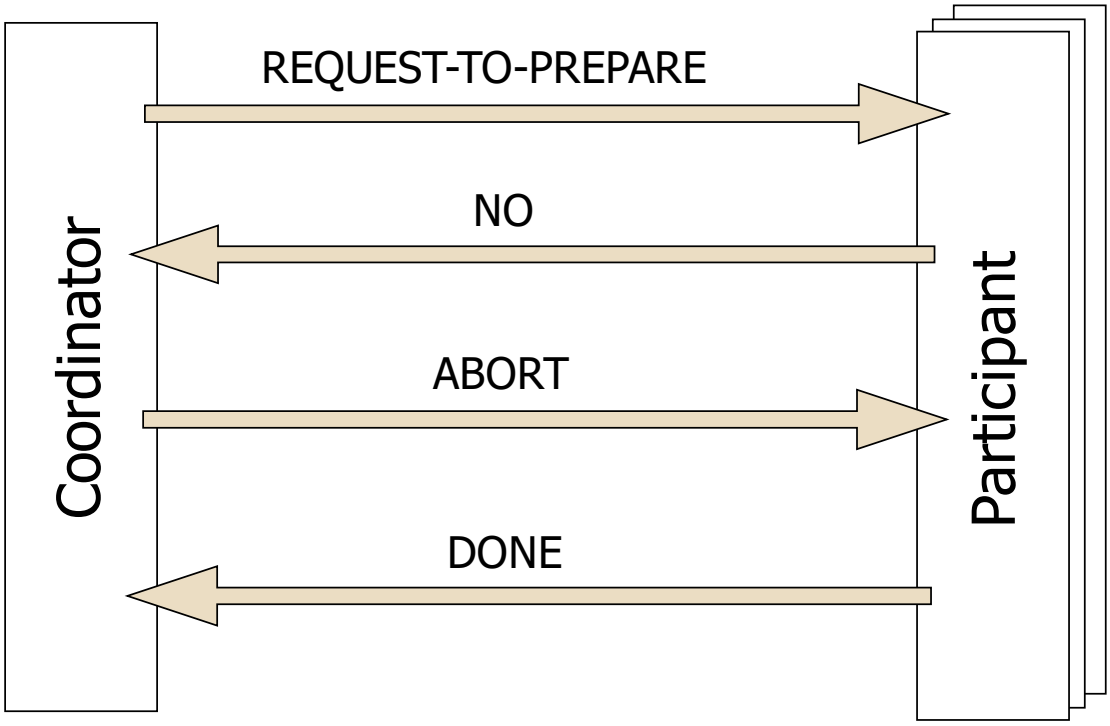
TWO-PHASE COMMIT (2PC) - OK



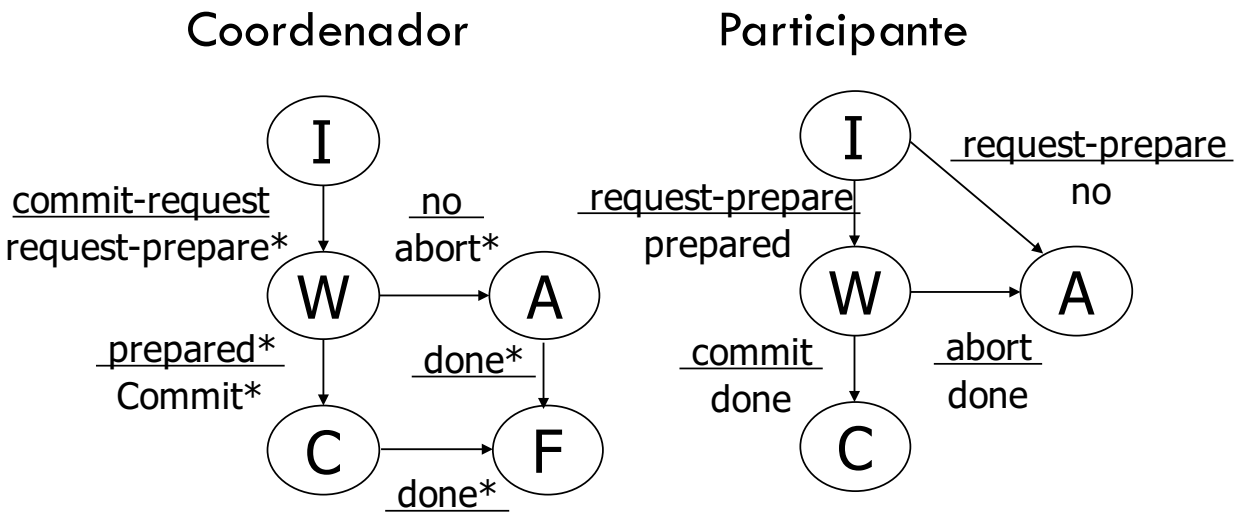
TWO-PHASE COMMIT (2PC) - ABORT




REQUEST-TO-PREPARE
PREPARED*
COMMIT*
DONE
Coordinator
Participant


REQUEST-TO-PREPARE
NO
ABORT
DONE
Coordinator
Participant

Versão centralizada



Bancos de Dados Comerciais

Todos suportam replicação e atomicidade!

- Oracle (>=9)
- DB2 (todos)
- MSQL (>=2000)
- HSQLDB
- PostgreSQL
- DHT (Kademlia, Tapestry, Cassandra)
- NoSQL (Cassandra, ZooKeeper, Ignite, etc)

Comparação vs. BC

- Não tem *proof-of-work*
- Máquinas devem se conhecer (e *link* deve ser bom)
- Controle de acesso também é replicado
- Permite otimizar espaço e processamento
  - Esquemas de replicação híbridos
  - Máquinas *cold* não fazem processamento

- Não resolve o consenso...