# Cloud DBMS: An Overview

DB/AI Bootcamp

2018 Summer

Datalab, CS, NTHU

# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

# Definition

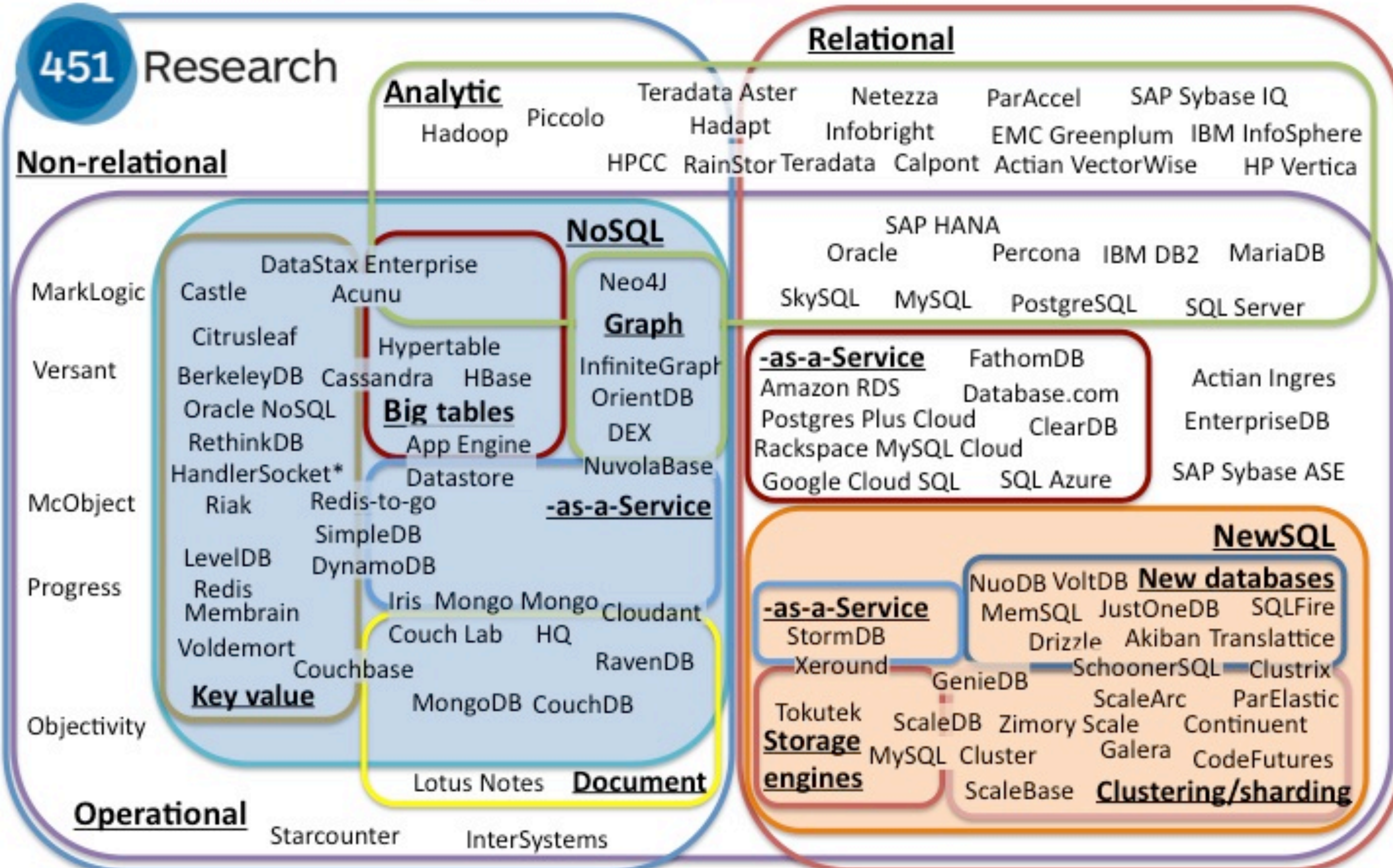
- A ***cloud DBMS*** is a DBMS designed to run in the cloud
  - Machines could be either physical or virtual
- In particular, some manages data of tremendous applications (called ***tenants***)
  - A.k.a. ***multi-tenant DBMS***
- Is MySQL a cloud database?
  - I can run MySQL in a Amazon EC2 VM instance
  - No

# What's the Difference

- Ideally, in addition to all features provided by a traditional database, a cloud database should ensure ***SAE***:
- ***high Scalability***
  - High max. throughput (measured by Tx/Query per second/minute)
  - Horizontal, using commodity machines
- ***high Availability***
  - Stay on all the time, despite of machines/network/datacenter failure
- ***Elasticity***
  - Add/shutdown machines and re-distribute data on-the-fly based on the current workload

# What do we have now?

# The evolving database landscape



# Why So Complicated?

- Full DB functions + SAE = a goal no one can achieve (currently)

- Even with Oracle 11g + SPARC SuperCluster
  - 30,249,688 TPC-C transactions per minute
  - $30+ million USD
- ***You lose elasticity!***

Wait, what do you mean by
full DB functions + SAE?

# DB Functions

- Expressive relational data model
  - Almost all kinds of data can be structured as a collection of tables
- Complex but fast queries
  - Across multiple tables, grouping, nesting, etc.
  - Query plan optimization, indexing, etc.
- Transactions with ACID
  - Your data are always correct and durable

# List of Desired Features

| Feature | Why? |
|---|---|
| Relational model | Models (almost) all data, flexible queries |
| Short latency | 100ms more response time = significant loss of users (Google) |
| Tx and ACID | Keeps data correct and durable |
| Scalability | You are (or will be) big |
| Availability | No availability, no service! |
| Elasticity | Save $ |

# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

# Scalability (OLTP Workloads)

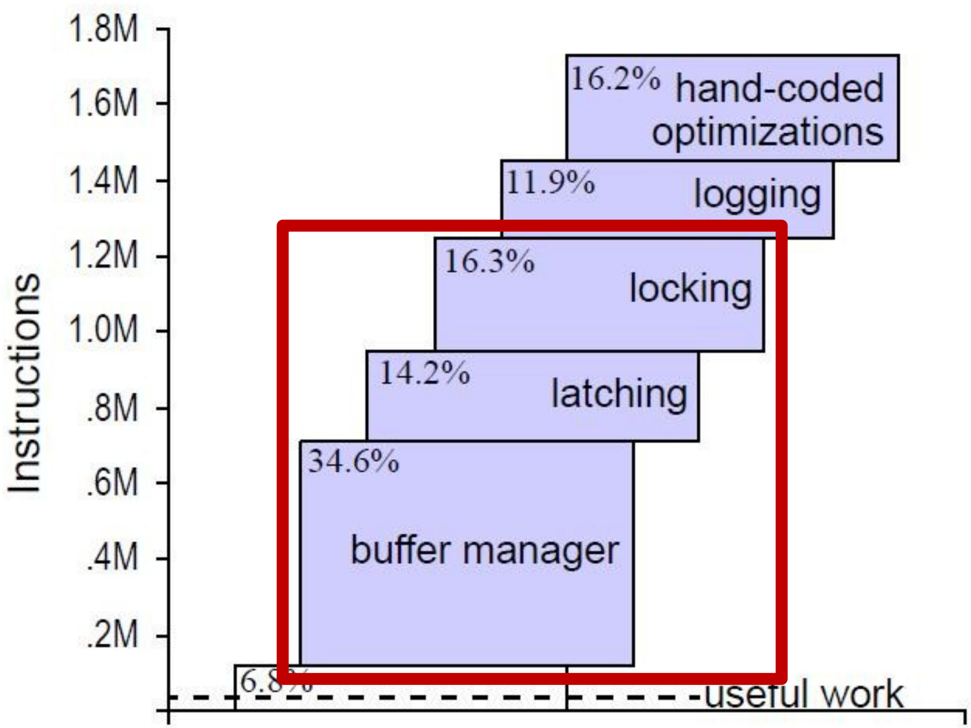
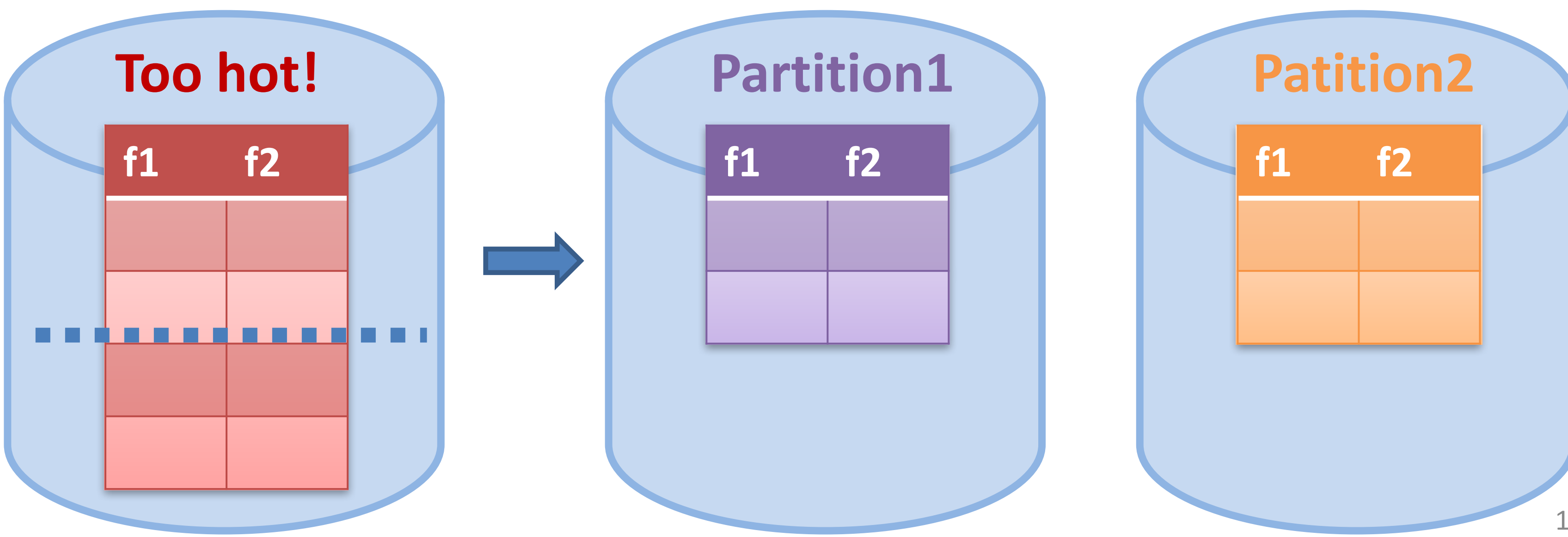



**Figure 1. Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C. The top of the bar-graph is the original Shore performance with a main memory resident database and no thread contention. The bottom dashed line is the useful work, measured by executing the transaction on a no-overhead kernel.**

- ***Max tx throughput ∝ (contention footprint * tx conflict rate)$^{-1}$***
  - A tx holding locks for a long time blocks all conflicting txs and reduces throughput
- Contention footprint increases when accessing hot tables (due to I/O bottleneck)
- How to improve the throughput?

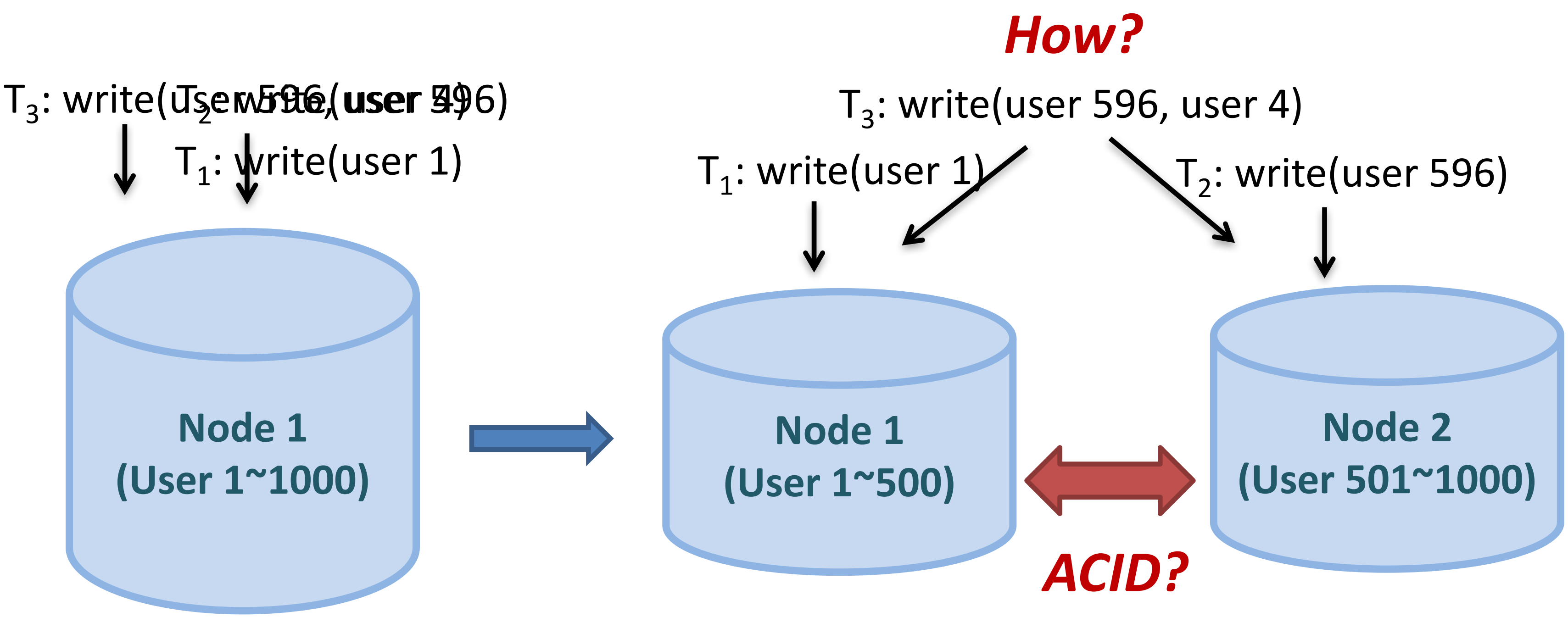
# Scalability and Partitioning

- ***Partition*** your hot tables
  - Either horizontally  or vertically
  - Distribute read/write load to different servers



# Complications in Distributed DBs

- Records spread among partitions on different servers



# Complications in Distributed DBs

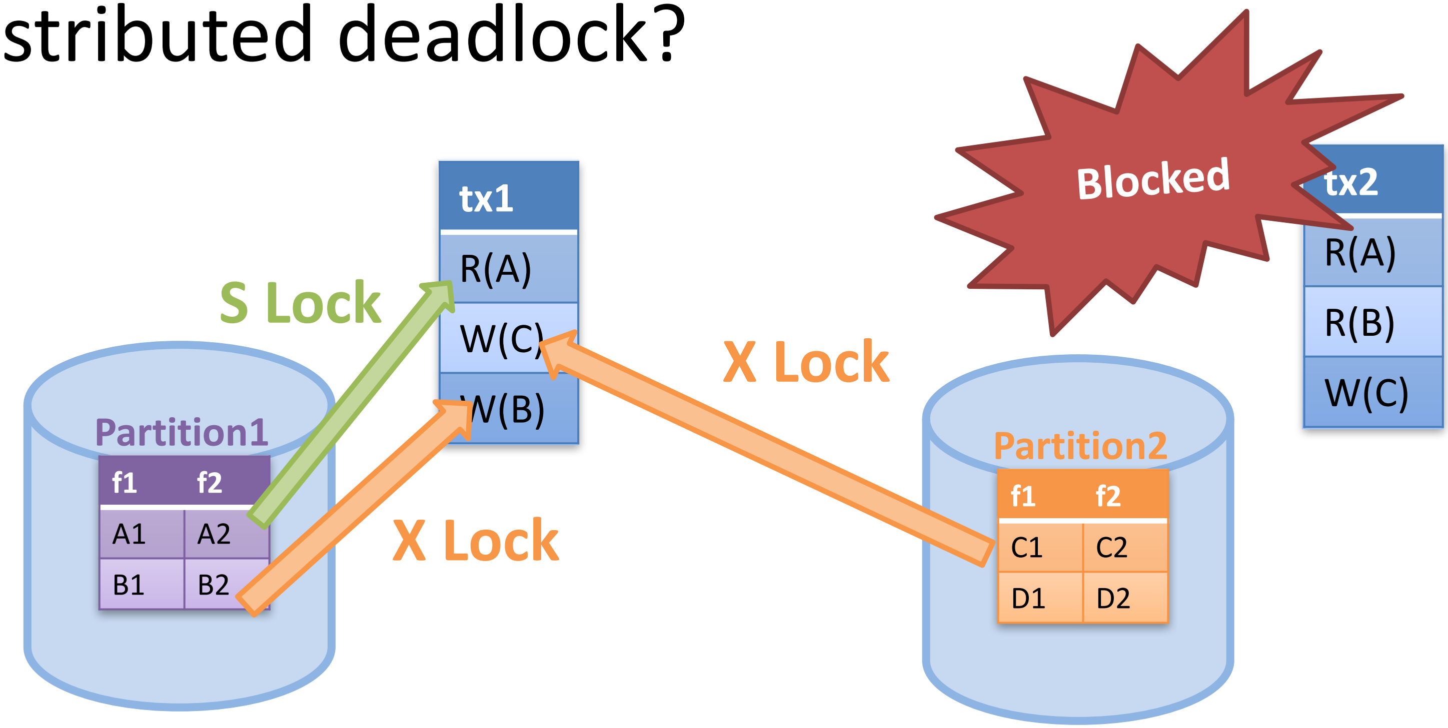
- Records spread among partitions on different servers
- Distributed metadata manager
- Distributed query processor
  - Best global-plan and its local-plans?
- ***Distributed transactions***
  - ACID of a global-transaction T and its local-transactions {Ti}?

# Isolation Revisited

- Requires a distributed CC manager
- For 2PL
  - Dedicated lock server, or
  - Primary server for each lock object (***Distributed S2PL***)
- For timestamp and optimistic CC
  - The problem is how to generate the global unique timestamps
  - E.g., "local_counter@server_ID"
  - To prevent one server counts faster, each server increments its own counter upon receiving a timestamp from others

# Distributed S2PL

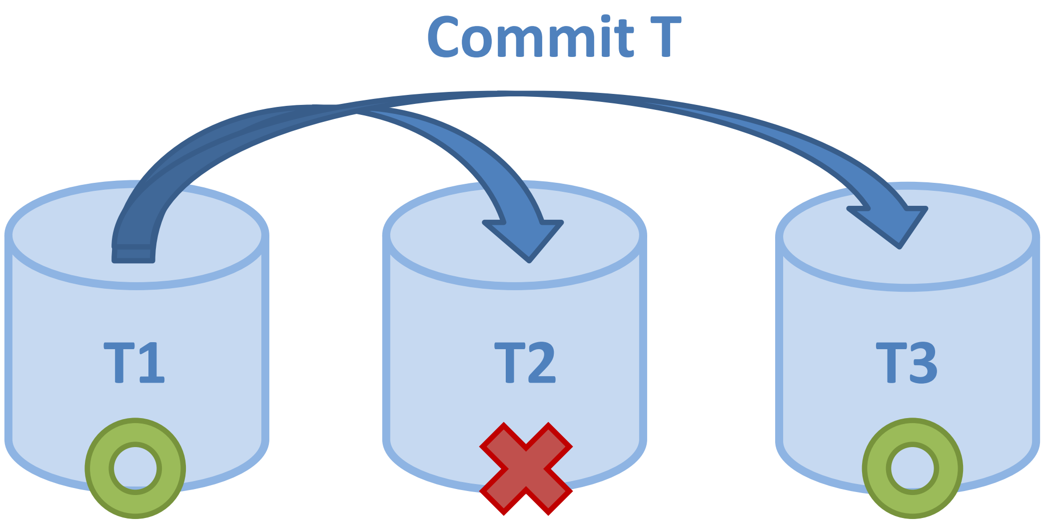
- Primary server of an object: machine owning the corresponding partition
- Distributed deadlock?



# Atomicity Revisited

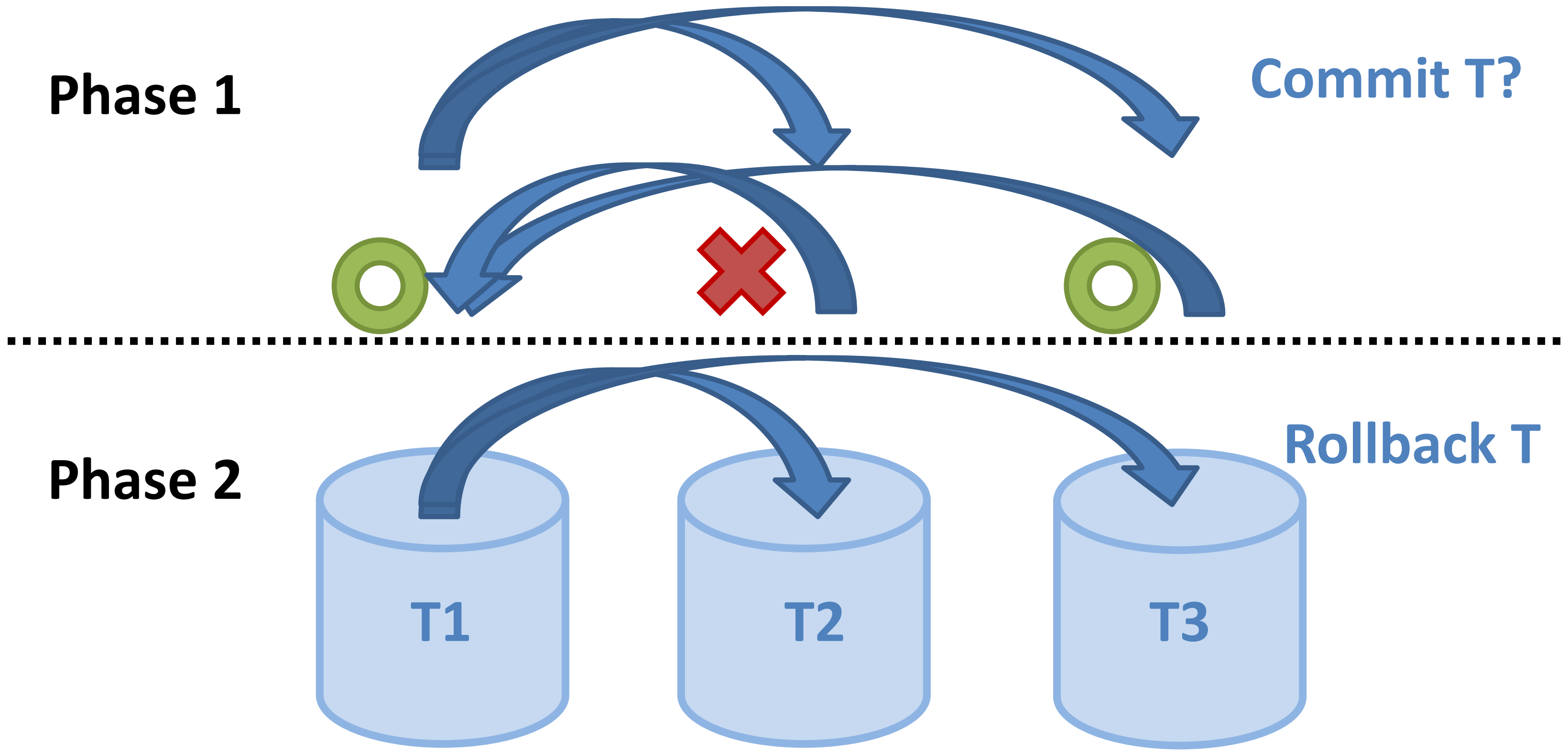
- Committing T means committing all local-transactions
- If any local-transaction rolls back, then T should roll back
  - When will this happen?
  - ACID violation (e.g., in OCC)
  - Deadlock
  - Node failure (detected by some other nodes such as replica)

# One-Phase Commit




- If T2 rolls back (due to ACID violation or failure), then T is partially executed, violating atomicity
  - The effect of T1 and T3 cannot be erased due to durability

# Two-Phase Commit



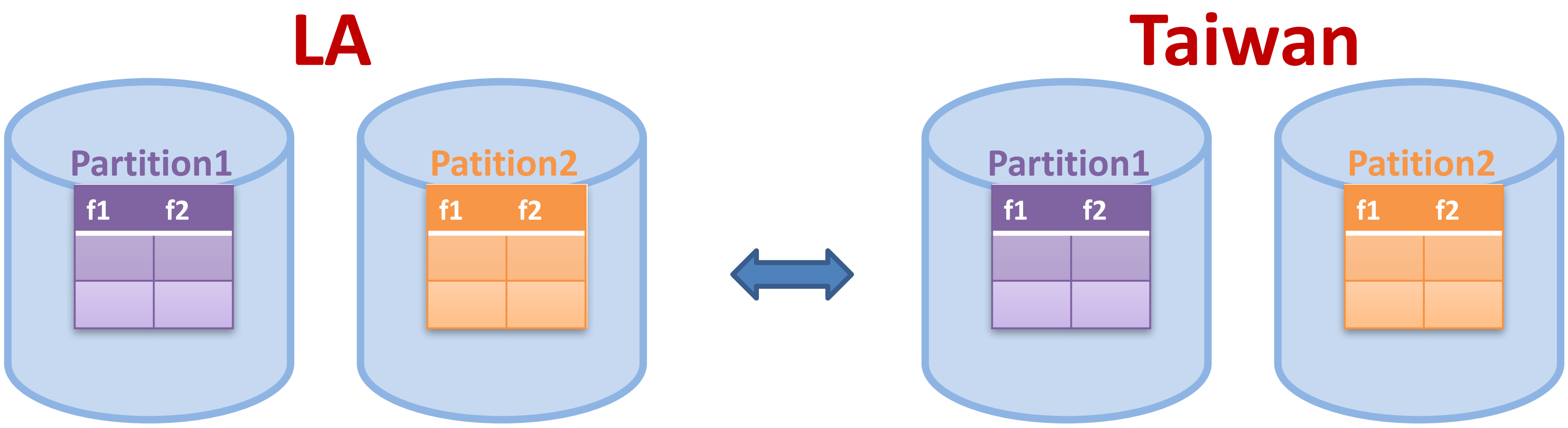

- Drawback: long delay
  - T blocks all conflicting txs and reduces throughput
- Partition helps only when the overhead of communication (2PC) < overhead of slow I/Os

# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

# Availability

- ***Replicate*** all tables across servers
  - If servers in one region fails, we have spare replicas
- Ideally, across geographically-separated regions
  - To deal with disaster

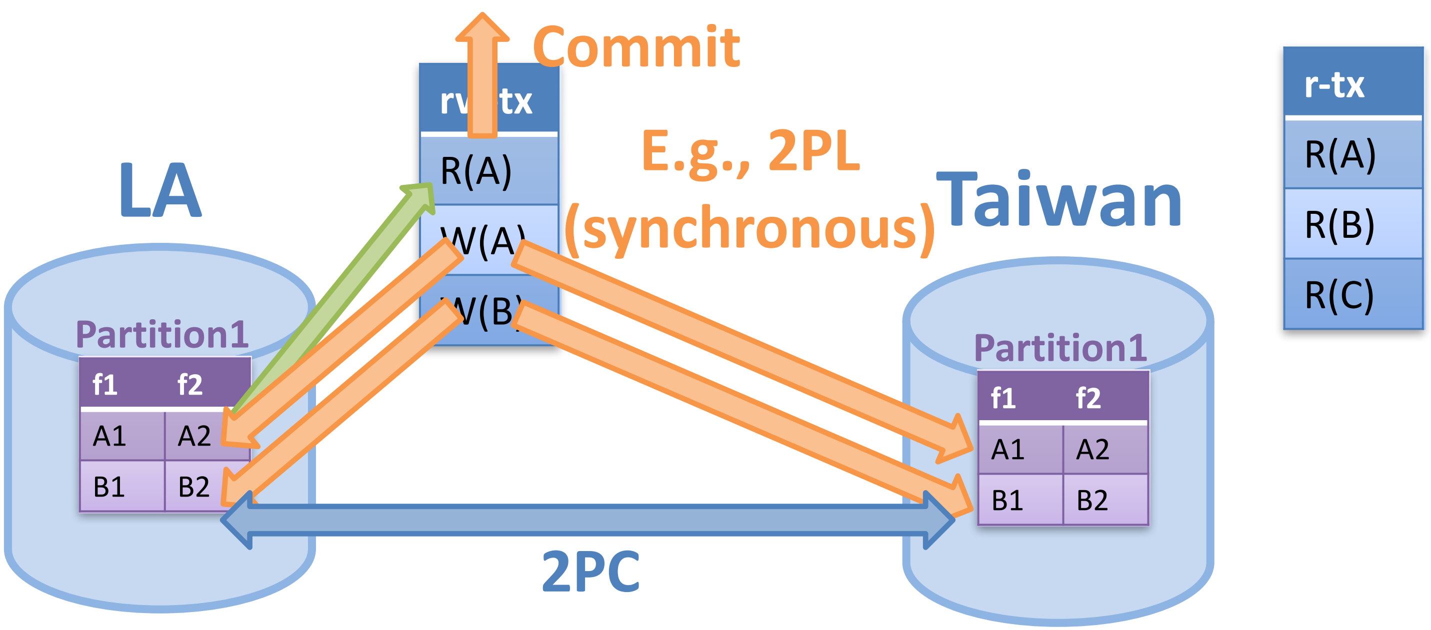


# Consistency Revisited

- Consistency
  - Txs do not result in violation of rules you set
- In distributed environments, consistency also means “all replicas ***remain the same*** after executing a tx”
  - Tx reads local, writes all (R1WA)
  - Side-benefit: a read-only tx can be on any replica
- Changes made by a tx on a replica need to be propagated to other replicas
- When? ***Eager vs. Lazy***
- By whom? ***Master/Slave vs. Multi-Master***

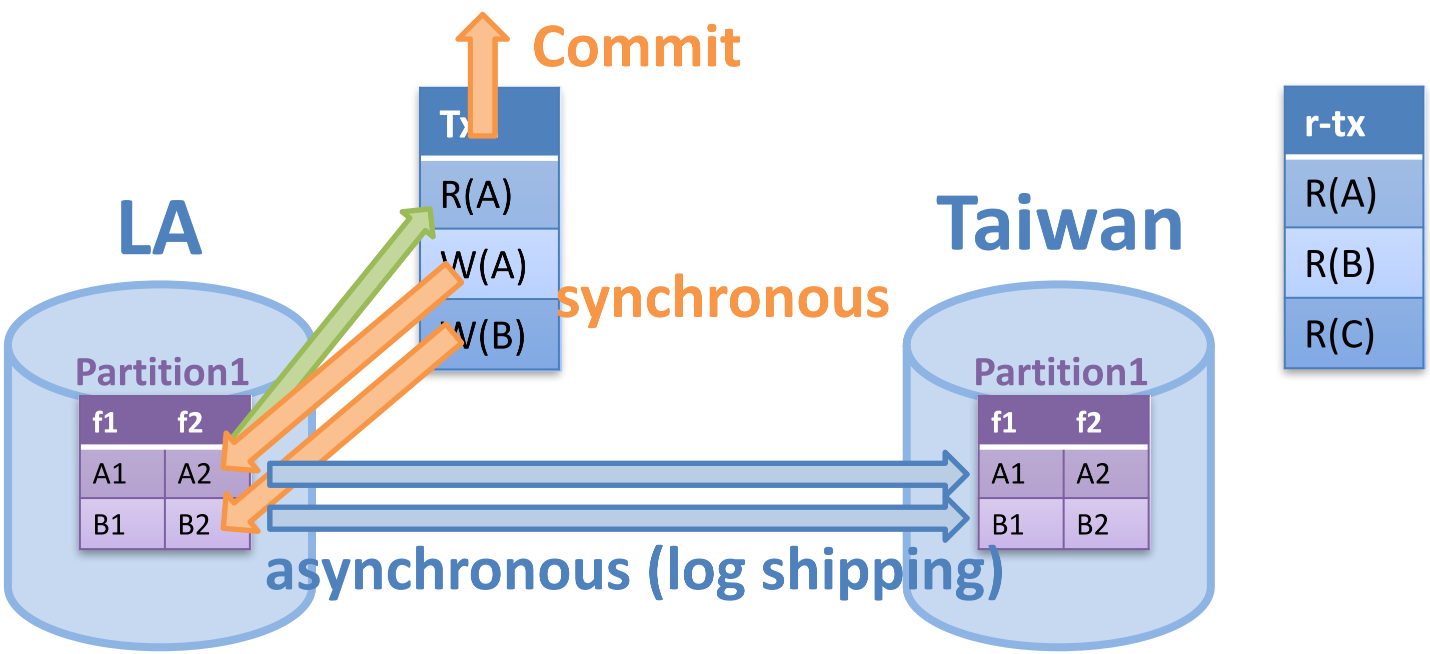
# Eager Replication

- Each write operation must completes on all replicas ***before a tx commits***
  - 2PC required
  - Failure on any replica causes tx's rollback
- Slow tx, but strong consistency



# Lazy Replication

- Writes complete locally, but are propagated to remote replicas ***in the background*** (with a lag)
  - Usually by shipping a batch of logs
  - Fast tx, but eventual consistency



# Who Writes?

- ***Master/Slave*** replication
  - Writes of a record are routed to a specific (called master) replica
  - Reads to others (slave replicas)
- ***Multi-Master*** replication
  - Writes of a record can be routed to any replica
  - I.e., two writes of the same records may be handled by ***different*** replicas

# The Score Sheet

| | Eager MM | Lazy M/S | Lazy MM |
|---|---|---|---|
| Consistency | Strong | ***Eventual*** | ***Weak*** |
| Latency | ***High*** | Low | Low |
| Throughput | ***Low*** | High | High |
| Availability upon failure | Read/write | ***Read-only*** | Read/write |
| Data loss upon failure | None | ***Some*** | ***Some*** |
| Reconciliation | No need | No need | ***User or rules*** |
| Bottleneck, SPF | None | ***Master*** | None |

- Eager M/S and MM make no much difference with 2PL + 2PC
- Lazy MM needs reconciliation of conflicting writes
  - Either by user or rules, e.g., last-write-wins

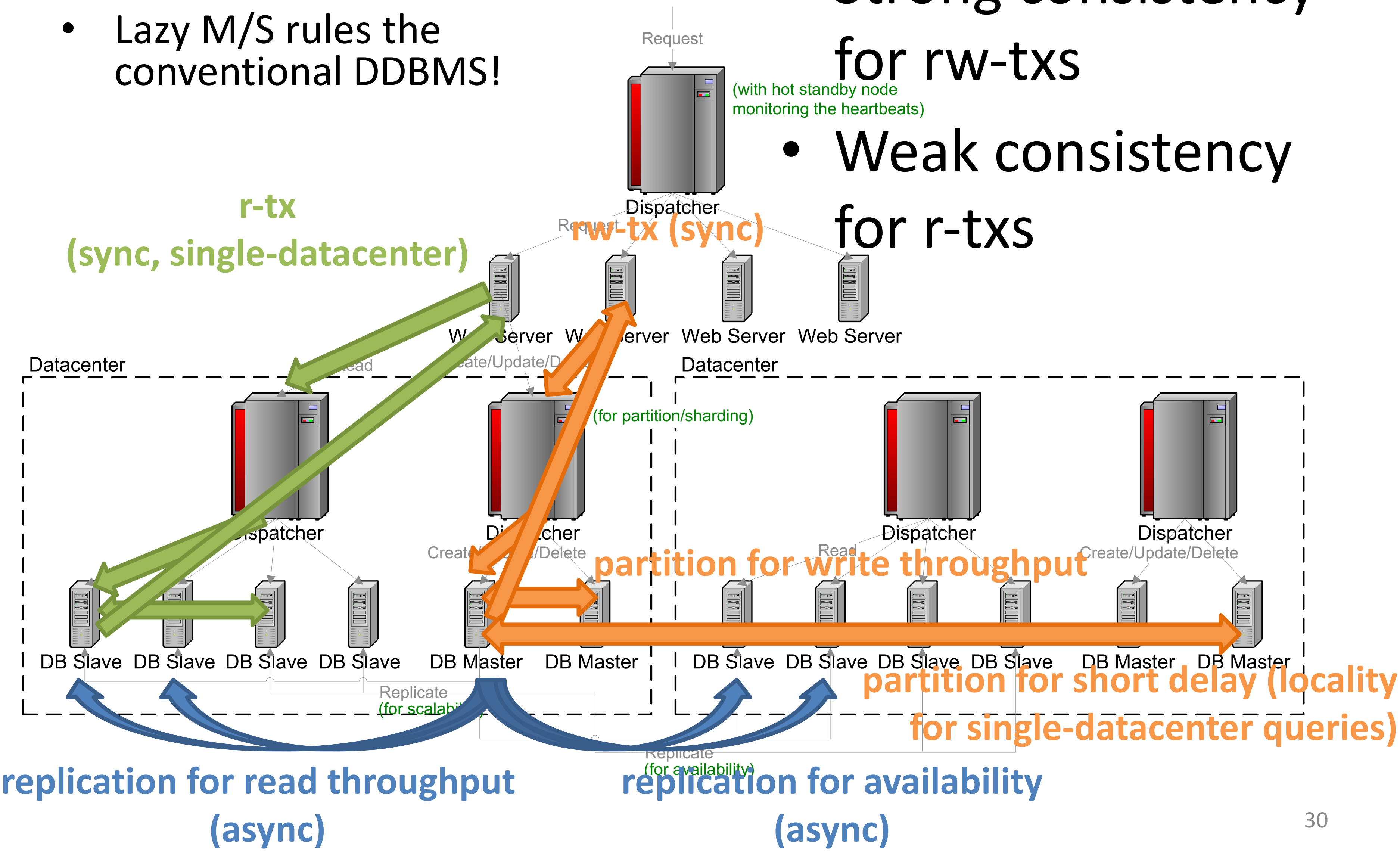
# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

# A Real-World Distributed DBMS (DDBMS)

- Lazy M/S rules the conventional DDBMS!

- Strong consistency for rw-txs
- Weak consistency for r-txs



# What's the problem?

# "S" only with High-End Machines

- Dist. txs are costly
  - Even within a datacenter (High latency due to 2PC)
- Every rw-tx routes to the masters, a potential performance bottleneck
- Solution: careful data partitions + super-powerful masters
  - Reduces distributed txs (especially those across regions) as many as possible
  - Reduces the number of masters
- Scale ***up*** (rather than ***out***) = exponentially increase in $

# Without Failure: Trade-Off between C and Latency

- Lazy M/S replication: trade C for short latency
  - A read-only tx is routed to slaves
  - Reads an inconsistent snapshot of data if spanning across partitions
- To support full C, clients are allowed to require a read-only tx to go through the masters
  - But you need more powerful masters (and more $)

# With Failure: Trade-Off between C and Av, Plus No D

- Machine failure:
  - Each mater has few hot-stand-by servers
  - Placed nearby to reduce overhead
- Datacenter failure:
  - Due to power shortage, disasters, etc.
  - Trade-off: read-only during failover (no Av) or allow inconsistency for read-write txs (no C)
  - Either way, no D: lazy M/S causes data loss

| DDBMS | |
|---|---|
| Data model | Relational |
| Tx boundary | Unlimited |
| Consistency | W strong, R eventual |
| Latency | Low (local partition only) |
| Throughput | High (scale-up) |
| Bottleneck/SPF | Masters |
| Consistency (F) | W strong, R eventual |
| Availability (F) | Read-only |
| Data loss (F) | Some |
| Reconciliation | Not needed |
| Elasticity | No |

DDBMS venders:

If you are willing to pay a lot and sacrifice C and D, I can offer S and limited Av.

# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

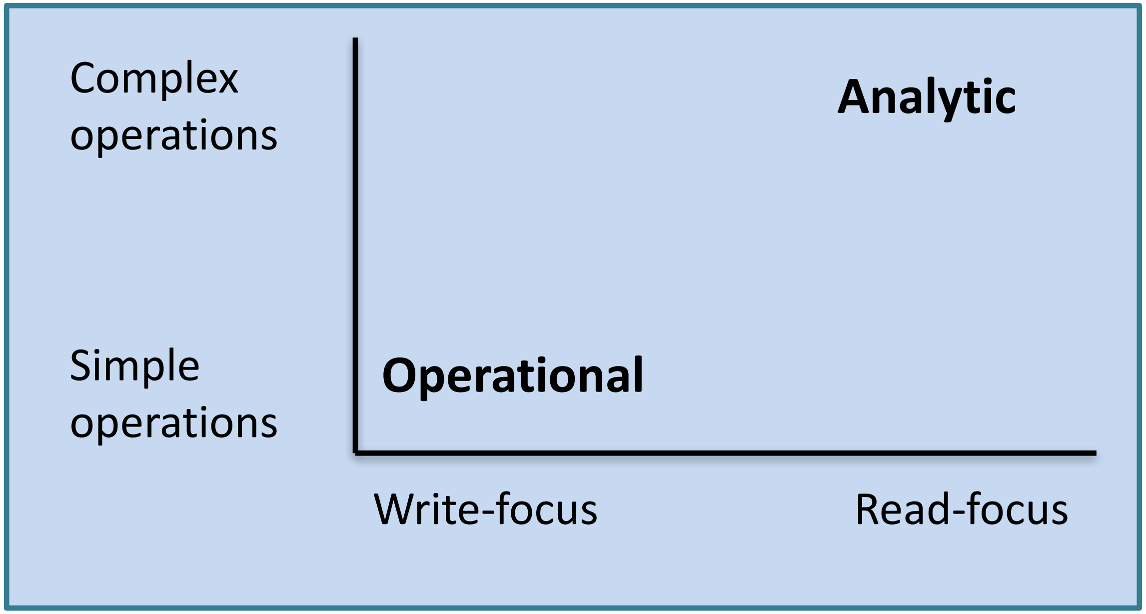
# Do you really need full DB functions + SAE?

| Feature | Why? |
|---|---|
| Relational model | Models (almost) all data, flexible queries |
| Short latency | 100ms more response time = significant loss of users (Google) |
| Tx and ACID | Keeps data correct and durable |
| Scalability | You are (or will be) big |
| Availability | No availability, no service! |
| Elasticity | Save $ |

# DB Workloads (1/3)

- ***Operational*** workloads
  - For daily business operations
  - A.k.a. ***OLTP*** (On-line Transaction Processing)
- ***Analytic*** workloads
  - For data analysis and decision support
  - Online (***OLAP***) or offline

# DB Workloads (2/3)



# DB Workloads (3/3)

| | Operational | Analytic |
|---|---|---|
| Data purpose | Latest snapshot | Historical or consolidated from multiple op sources |
| Data volume | < few tens of TBs | hundreds TBs to hundreds PBs, or more |
| R/W ratio | 60/40 to 98/2 | Mostly reads |
| Query complexity | Short reads, or short reads + writes | Complex reads, batch writes |
| Delay tolerance | < few seconds | Minutes, hours, days, or more |
| Tx + ACID | Required | Not really matters |

# Outline

- Definition and requirements
- S through partitioning
- A through replication
- Problems of traditional DDBMS
- Usage analysis: operational vs. analytic workloads
- The end of one-size-fits all era

# The Era of One-Size-Fits-All

- OLTP players
  - Oracle (incl. MySQL), IBM DB2, PostgreSQL, MS SQL Server, etc.
- OLAP players
  - IBM Infosphere, Teredata, Vertica, Oracle, etc.
  - Based on ***star-schema***

# Today's Challenges

- SAE is a must!
  - E also means automation
- Unhappy OLTP users:
  - I don't want to pay a lot to scale up
  - Data may not be well-partitioned, even with few masters
  - Consistency is desirable in many cases
- Unhappy analytic users:
  - My data is at web scale
  - Data are ill-formatted and heterogeneous
  - Schema changes over time
- We focus on the OLTP systems in this course

# The evolving database landscape