



PA152: Efficient Use of DB  
**12. Replication and  
High Availability**

Vlastislav Dohnal

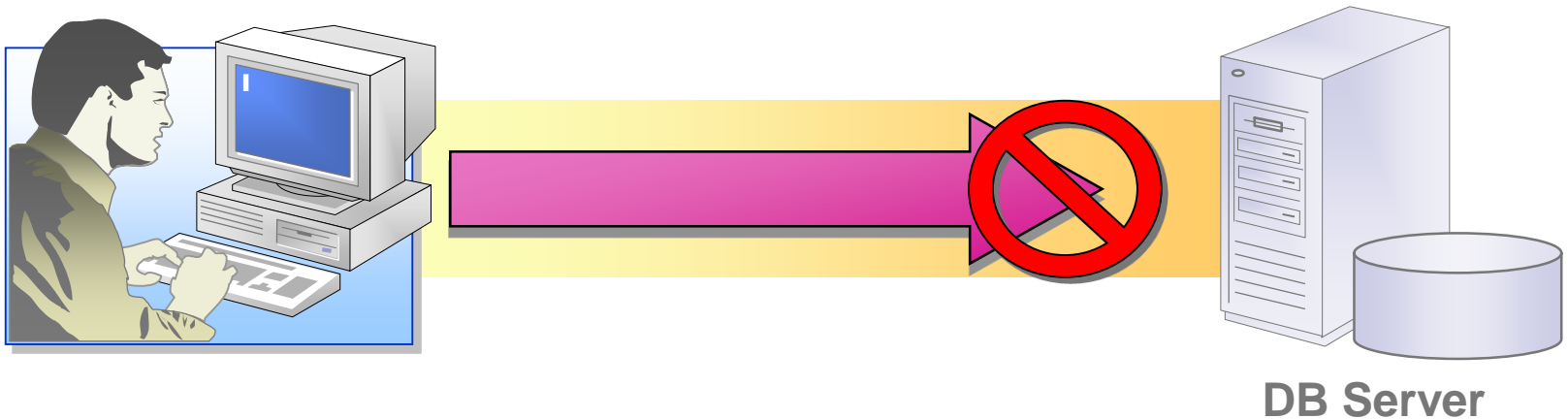
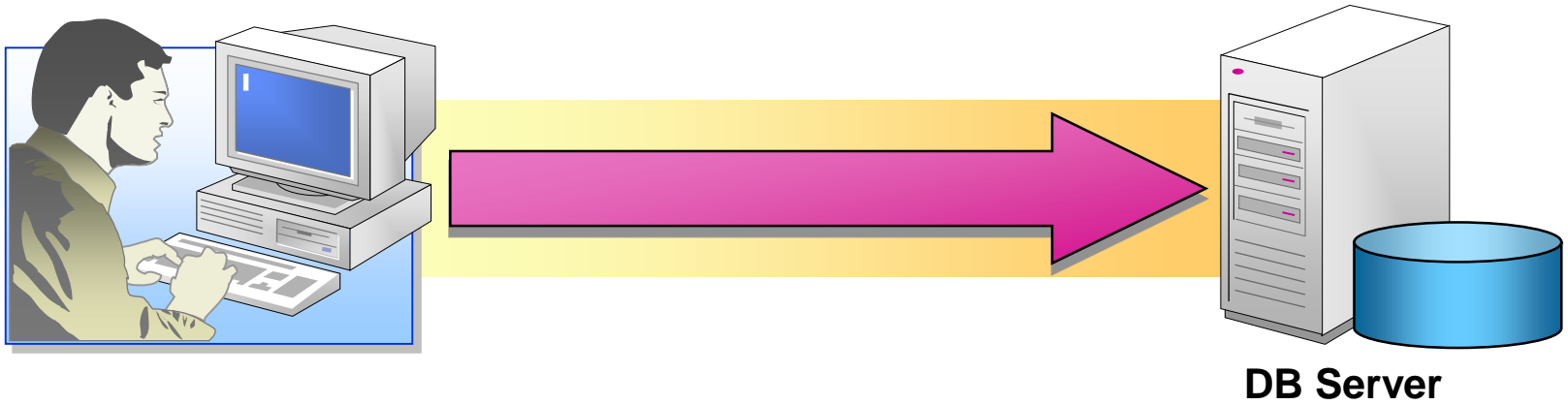
# Credits

- This presentation is based on:
  - Microsoft MSDN library
  - Course *NoSQL databases and Big Data management*
    - Irena Holubová
    - Charles University, Prague
    - <http://www.ksi.mff.cuni.cz/~holubova/NDBI040/>
  - PostgreSQL documentation
    - <http://www.postgresql.org/docs/9.3/static/high-availability.html>

# Contents

- Availability
- Data distribution & Replication
- High availability
- Failover
- Recommendations

# Availability



# Determining Availability Requirements

## ■ Hours of Operation

- Business hours vs. all of the time

  - intranet service vs. web services

  - shift workers vs. all-around the world customers

## ■ Connectivity Requirements

- Online vs. offline applications – so response time can be important!

## ■ Tight/Loose coupling of app and DBMS

- Synchronous vs. asynchronous data updates

# Availability

## ■ Definition in operation hours

□  $Av = \text{“up time”} / \text{“total time”}$

■ “up time” = the system is up and operating

□ More practical def.

■  $Av = (\text{total time} - \text{down time}) / \text{total time}$

## ■ Down time

□ Scheduled – reboot, SW/HW upgrade, ...

□ Unscheduled – HW/SW failure, security breaches, network unavailability, power outage, disasters, ...

□ Non-functional app requirements – response time

## ■ For “true” high-availability, down time is not distinguished

# Nines

- Availability as percentage of uptime

- Class of nines:  $c = \lfloor -\log_{10}(1 - Av) \rfloor$

- Assuming 24/7 operation:

Nine class	Availability	Downtime per year	Downtime per month	Downtime per week
1	90%	36.5 days	72 hours	16.8 hours
2	99%	3.65 days	7.20 hours	1.68 hours
3	99.9%	8.76 hours	43.8 minutes	10.1 minutes
4	99.99%	52.56 minutes	4.32 minutes	1.01 minutes
5	99.999%	5.26 minutes	25.9 seconds	6.05 seconds
6	99.9999%	31.5 seconds	2.59 seconds	0.605 seconds
7	99.99999%	3.15 seconds	0.259 seconds	0.0605 seconds

Source: Wikipedia.org

# Scalability

- Providing access to a number of concurrent users
- Handling growing amounts of data without losing performance
- With acceptable latency!



# Need for Distributing Data

- Brings data closer to its user
- Allows site independence
- Separates
  - Online transaction processing
  - Read-intensive applications
- Can reduce conflicts during user requests
- Process big data

# Scalability: Solutions

- **Scaling Up – vertical scaling** → vendor dependence
  - Increasing RAM
  - Multiprocessing
- **Scaling Out – horizontal scaling**
  - Replication
  - Read-only standby servers
  - Server federations / clusters / data distribution

# Horizontal Scaling

- Systems are distributed across multiple machines or nodes
  - Commodity machines → cost effective
  - Often surpasses scalability of vertical approach
- Fallacies of distributed computing by Peter Deutsch
  - Network
    - Is reliable, secure, homogeneous
    - Topology does not change
    - Latency and transport cost is zero
    - Bandwidth is infinite
  - One administrator

Source: <https://blogs.oracle.com/jag/resource/Fallacies.html>

# Brewer's CAP Theorem

## ■ Consistency

- After an update, all readers in a distributed system see the same data
- All nodes are supposed to contain the same data at all times
- E.g. in multiple instances, all writes must be duplicated before write operation is completed.

## ■ Availability

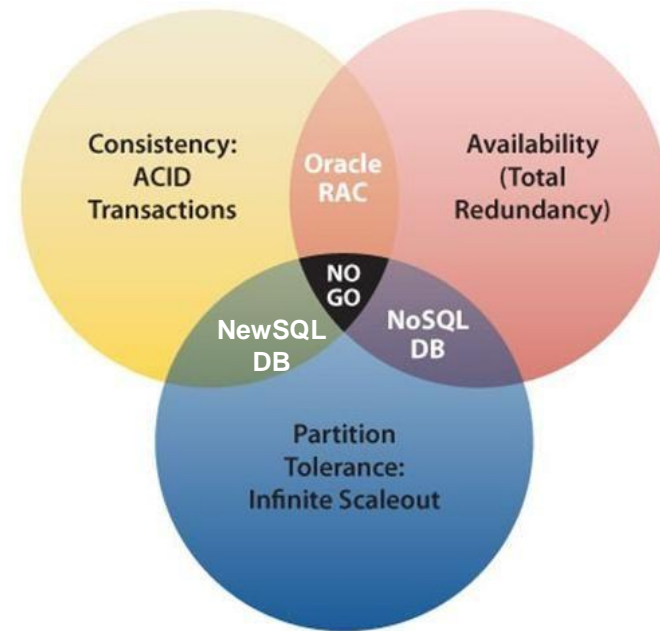
- Every request receives a response
  - about whether it was successful or failed

## ■ Partition Tolerance

- System continues to operate despite arbitrary message loss or failure of part of the system.

# Brewer's CAP Theorem

- Only 2 of 3 guarantees can be given in a “shared-data” system.
  - Proved by Nancy Lynch in 2002
- ACID
  - provides Availability and Consistency
  - E.g. database on a single machine
- BASE
  - provides Availability and Partition tolerance
  - Reality: you can trade a little consistency for some availability
  - E.g. distributed database



Source: <http://bigdatanerd.wordpress.com>

# NewSQL

- Distributed database system that scales out
- CP system
  - trades availability for consistency when partition happens
- MySQL cluster, Google Spanner, VoltDB, ...
  - In fact, master-master replication with data sharding

# BASE Properties

## ■ Basically Available

- Partial failures can occur, but without total system failure

## ■ Soft state

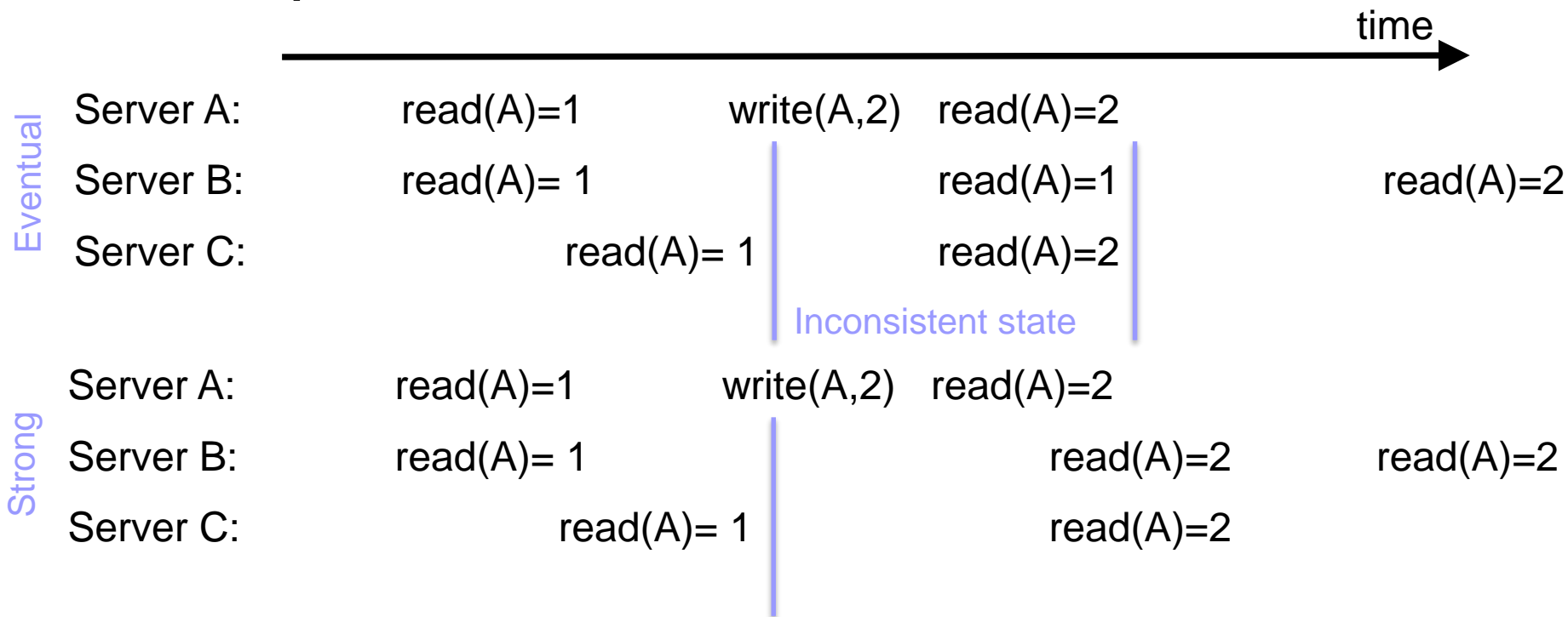
- System is in flux / non-deterministic
  - Changes occur all the time

## ■ Eventual consistency (replica convergence)

- is a liveness guarantee
  - reads eventually return the same value
- is not safety guarantee
  - can return any value before it converges

# Consistency

- Strong (ACID) vs. Eventual (BASE) consistency
- Example:

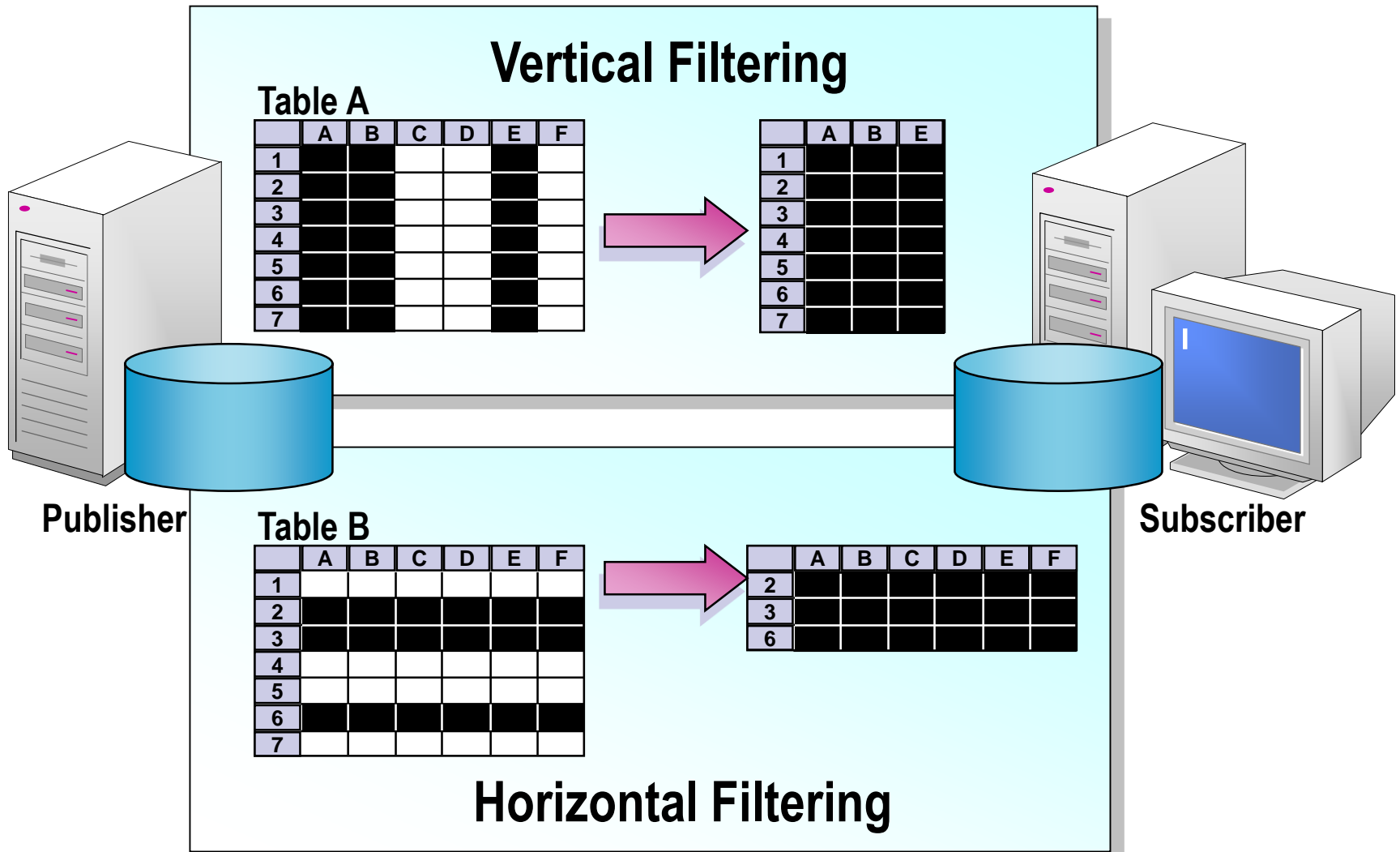




# Replication / Distribution Model

- Model of distributing data
  - Replication
    - The same data stored in more nodes.
  - Filtering data (sharding)
    - The data is partitioned and stored separately
    - Helps avoid replication conflicts when multiple sites are allowed to update data.

# Filtering Data



Source: Microsoft

# Distribution Model: Replication

- Master-slave model

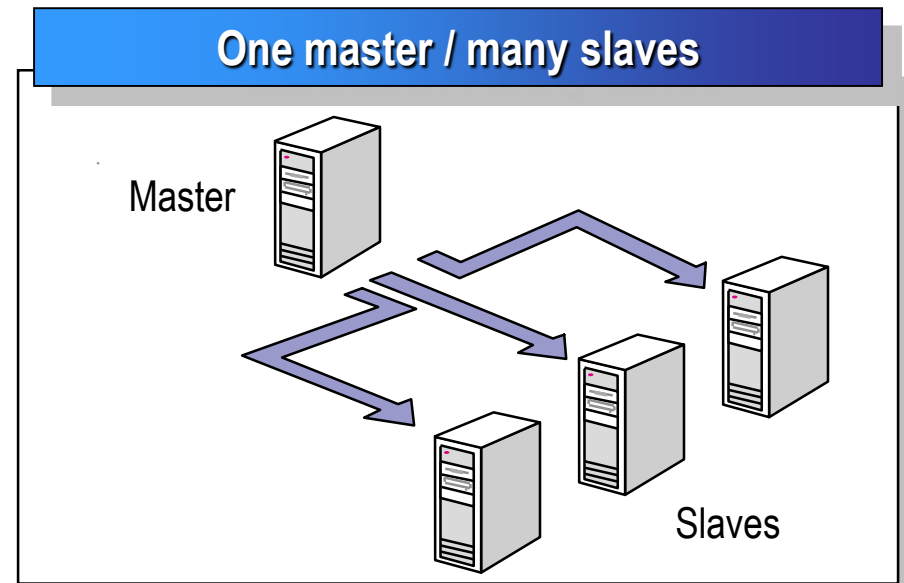
- Load-balancing of read-intensive queries

- Master node

- manages data
- distributes changes to slaves

- Slave node

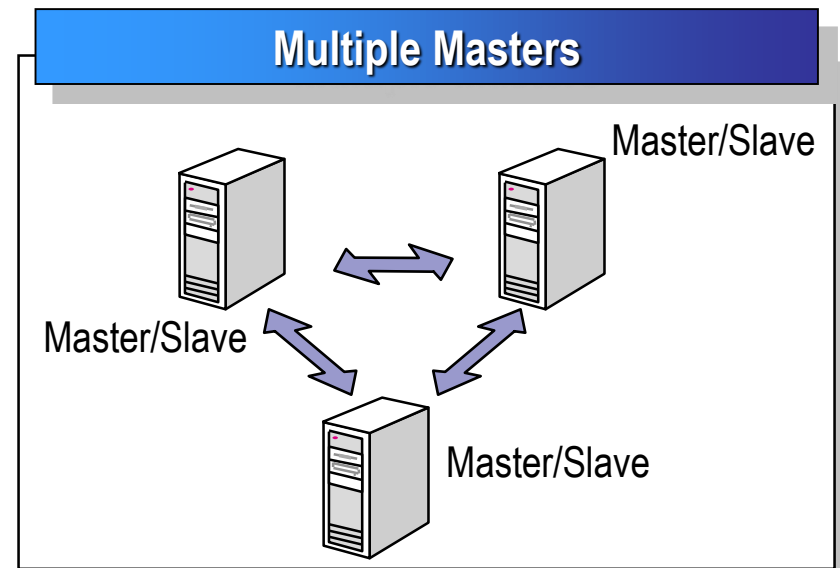
- stores data
- queries data
- no modifications to data



# Distribution Model: Replication

## ■ Master-master model

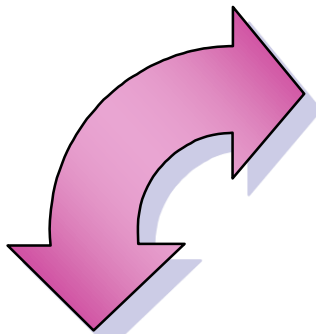
- Typically with filtering data
  - Master for a subset of data
  - Slave for the rest
- Consistency needs resolving of update conflicts



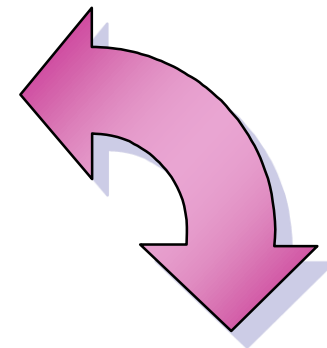
# Master-master Model

<i>Orders (Master A)</i>			
Primary Key			
Area	Id	Order_no	Qty
1	1000	~	15
1	3100	~	22
2	1000	~	32
2	2380	~	8
3	1000	~	7
3	1070	~	19

Master/Slave

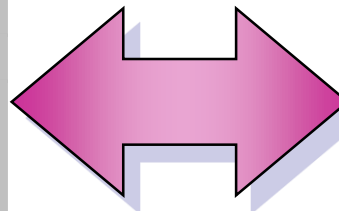


Master/Slave



Master/Slave

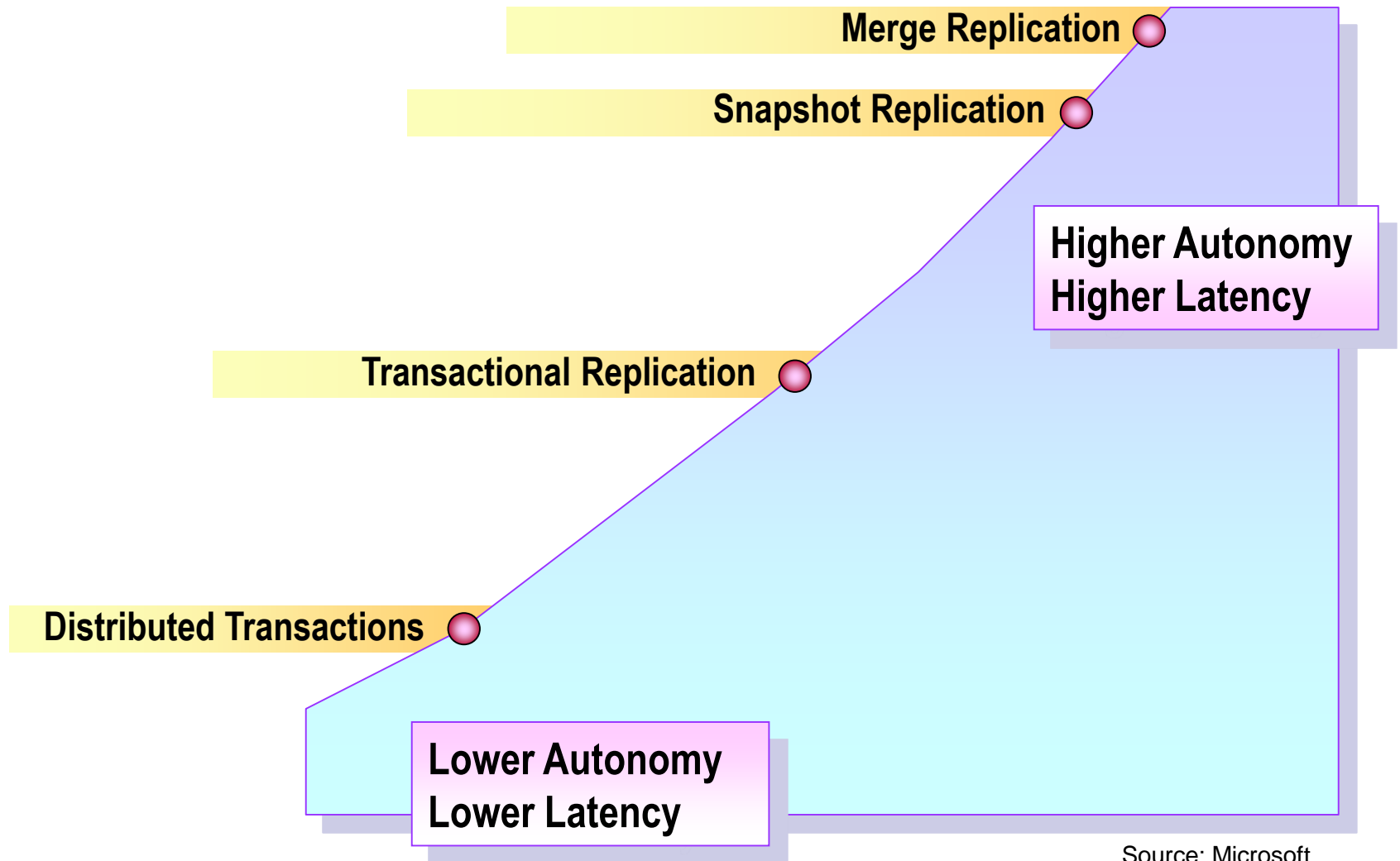
<i>Orders (Master B)</i>			
Primary Key			
Area	Id	Order_no	Qty
1	1000	~	15
1	3100	~	22
2	1000	~	32
2	2380	~	8
3	1000	~	7
3	1070	~	19



<i>Orders (Master C)</i>			
Primary Key			
Area	Id	Order_no	Qty
1	1000	~	15
1	3100	~	22
2	1000	~	32
2	2380	~	8
3	1000	~	7
3	1070	~	19

Source: Microsoft

# Replication Types for “real” multi-master model



Source: Microsoft

# Replication Types

## ■ Distributed Transactions

- For “real” master-master model, ensures consistency
- Low latency, high consistency

## ■ Transactional Replication

- Replication of incremental changes
- Minimal latency (typically online)
- Conflicts solves using shared locks

# Replication Types

## ■ Snapshot Replication

- Periodic bulk transfer of new snapshots of data
  - Intermediate updates to data might be unnoticed by “subscribers”
- Data changes – substantial but infrequent
- Slaves are read-only
- High latency is acceptable



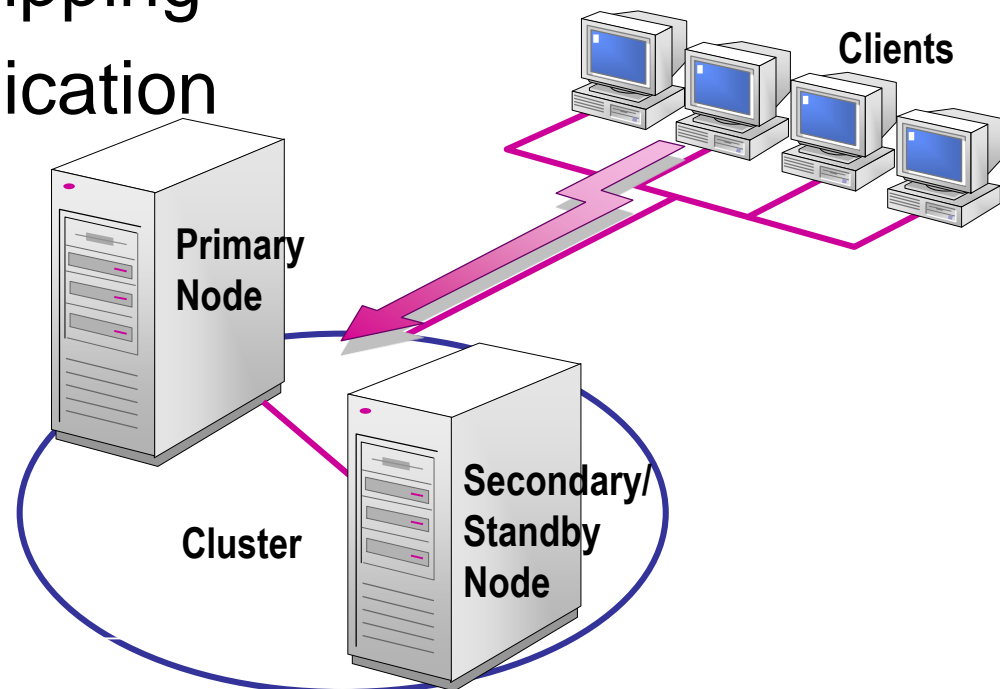
# Replication Types

## ■ Merge Replication

- Autonomous changes to replicated data are later merged
- Does not guarantee transactional consistency, but converges
- Default and custom conflict resolution rules
- Adv: Nodes can update data offline, sync later
- Disadv: Changes to schema needed.

# Maintaining High-Availability

- Standby server
  - Shared disk failover (NAS)
  - File system replication (DRBD)
  - Transaction log shipping
  - Trigger-based replication
  - Statement-Based Replication Middleware



# Log-shipping Standby Server

- Also called warm standby
- Primary node
  - serves all queries
  - in permanent archiving mode
    - Continuous sending of WAL records to standby servers
- Standby server
  - serves no queries
  - in permanent recovery mode
    - Continuous processing of WAL records arriving from primary node
- Log shipping can be synchronous/asynchronous
- Disadvantage: all tables are replicated typically
- Advantage: no schema changes, no trigger definitions

# Failover

- If primary fails, standby server begins failover.
  - Standby applies all WAL records pending,
  - marks itself as primary,
  - starts to serve all queries.
- If standby fails, no action taken.
  - After becoming online, catch-up procedure is started.
- Heartbeat mechanism
  - to continually verify the connectivity between the two and the viability of the primary server

# Failover

- Failover by standby succeeded
  - New standby should be configured
  
  - Original primary node becomes available
  - → inform it that it is no longer the primary
    - do so-called STONITH (Shoot The Other Node In The Head),
    - otherwise serious data corruption/loss may occur
  
  - Typically old primary becomes new standby

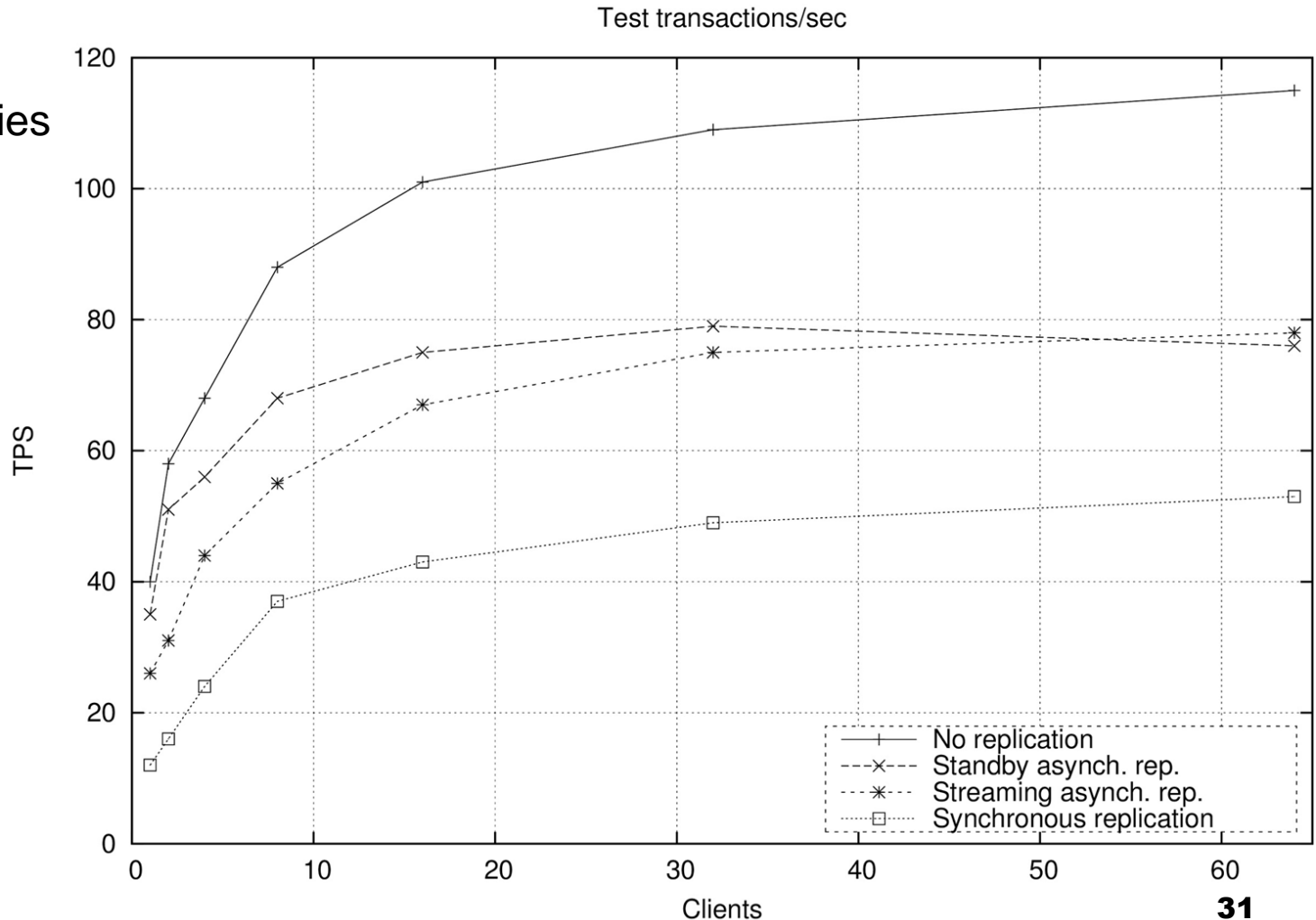
# Primary and Standby Servers

- Swap primary and standby regularly
  - To verify recovery steps
  - To do necessary maintenance on standby server
    - SW/HW upgrades, ...

# PostgreSQL: Replication

## ■ TPC Benchmark B

scale factor 1  
1 trans. = 5 queries



# Recommended Practices

- Maximize availability at each tier of the application
- Keep standby servers on a different subnet
- Independent power supply to the primary server
- Test whether your availability solution works