PA152: Efficient Use of DB

11. Replication and High Availability

Vlastislav Dohnal

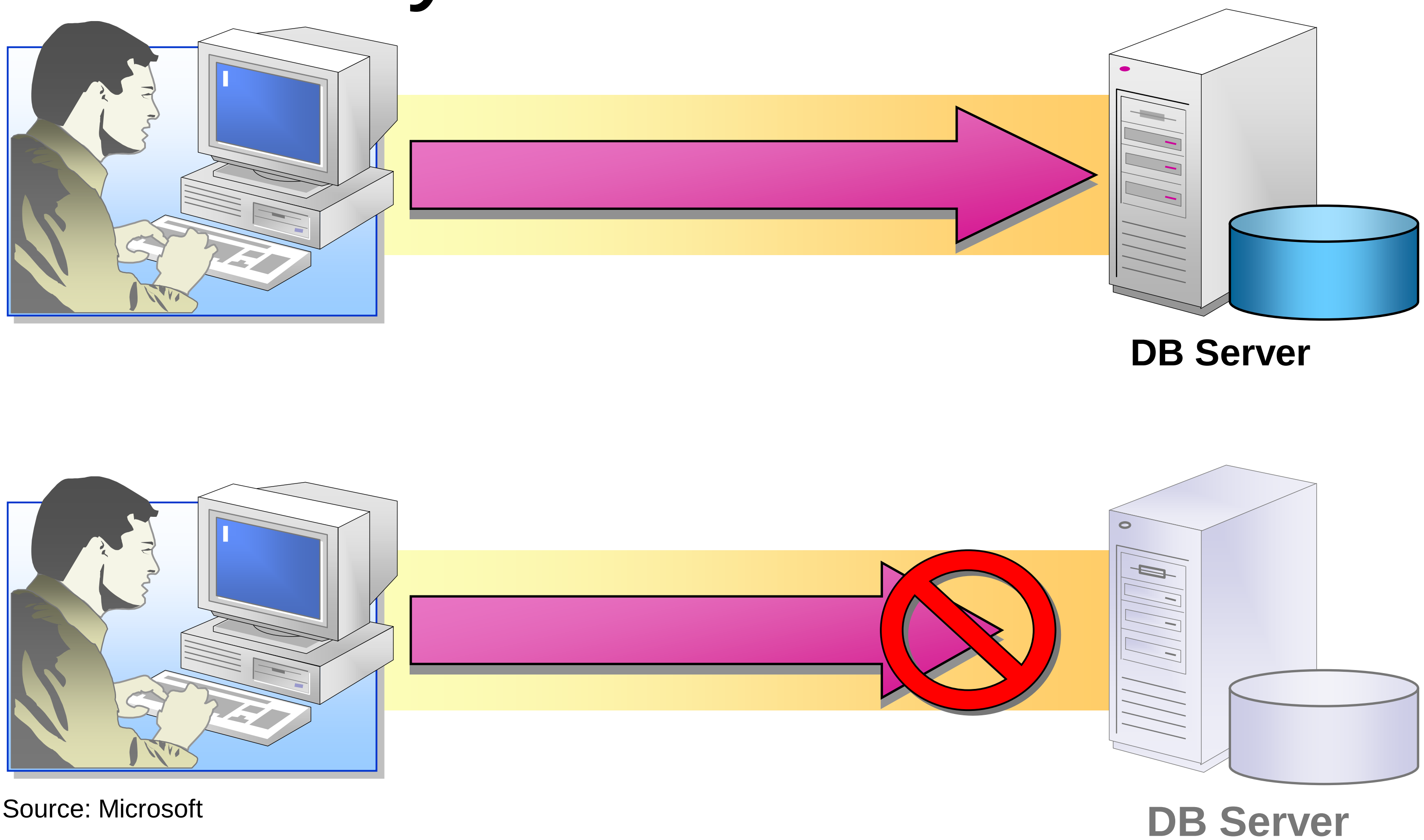
Credits

- 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
  - 451 Research
    - M. Aslett: CAP Theorem – two out of three ain't right

Contents

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

Availability



Source: Microsoft

High availability (HA) is the ability of a system to operate continuously without failing for a specified period of time.

Determining Availability

- Hours of Operation
  - Business hours vs. all of the time
    - intranet service vs. web service
    - shift workers vs. all-around-the-world customers
- Connectivity Requirements
  - Tight/Loose coupling of app and DBMS
    - Synchronous vs. asynchronous data updates
  - Online vs. offline applications – so response time can be important!

Availability

- Definition using operation hours
  - Av = “up time” / “total time”
    - “up time” = the system is up and operating
  - More practical def.
    - Av = (total time - down time) / 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

Quantifying Availability: 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!

Scalability: Solutions

- Scaling Up – **vertical** scaling
  - Increasing RAM
  - Multiprocessing
  - $\rightarrow$ vendor dependence

- Scaling Out – **horizontal** scaling
  - Server federations/clusters
  - $\rightarrow$ data distribution

Horizontal Scaling

- Systems are distributed across multiple machines or nodes
  - Commodity machines $\rightarrow$ 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

Need for Distributing Data

- Bring data closer to its user
  - geographic scalability
- Allow site independence
- Separate
  - Online transaction processing
  - Read-intensive applications
- Reduce conflicts during user requests
- Process large volumes of data

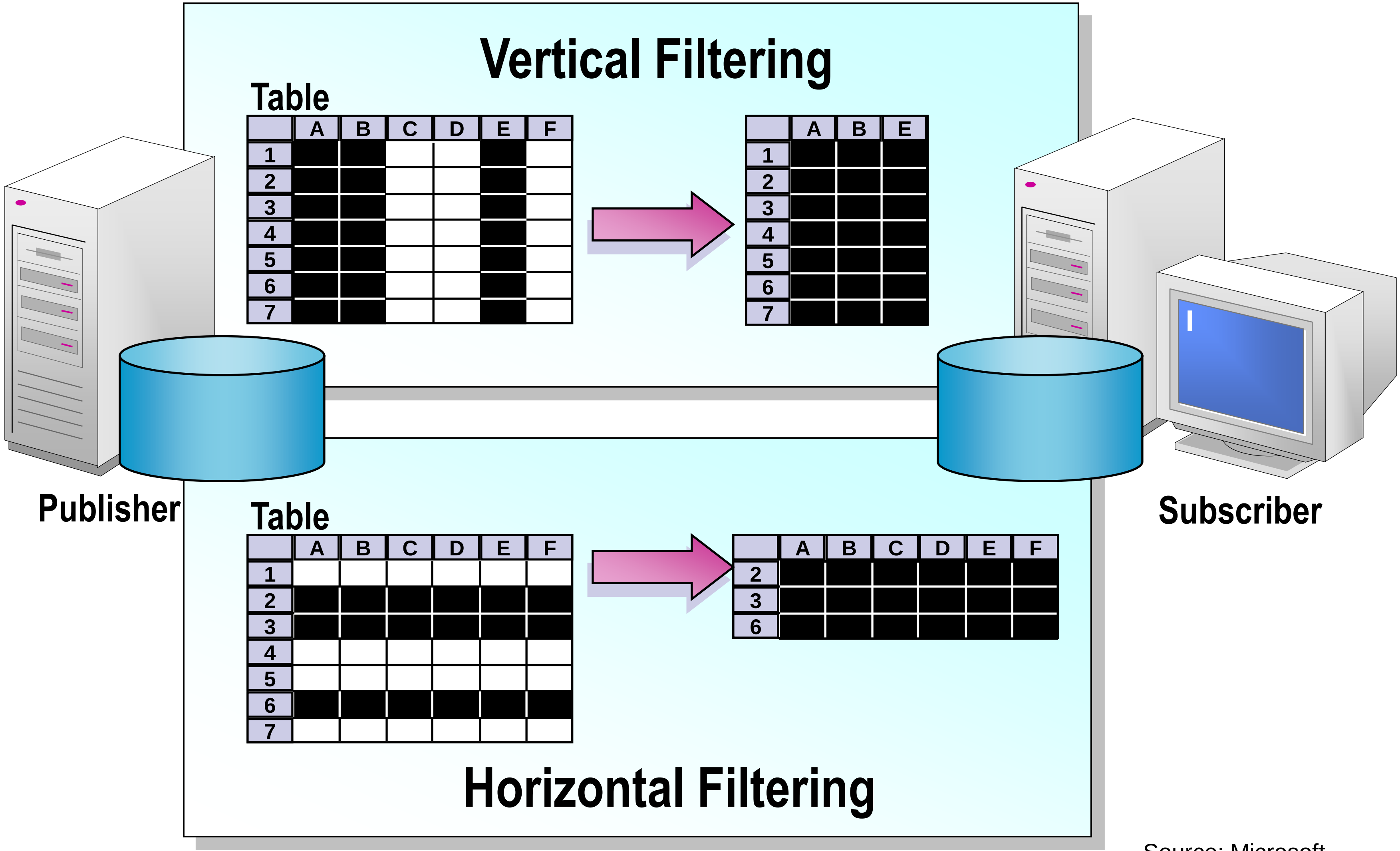
Distributing Data

- Approaches
  - Data partitioning
    - Sharding
    - E.g.,
      - Peer-to-peer networks
  - Replication
    - Copies of data
    - E.g.,
      - One writeable and many read-only (standby) servers

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.

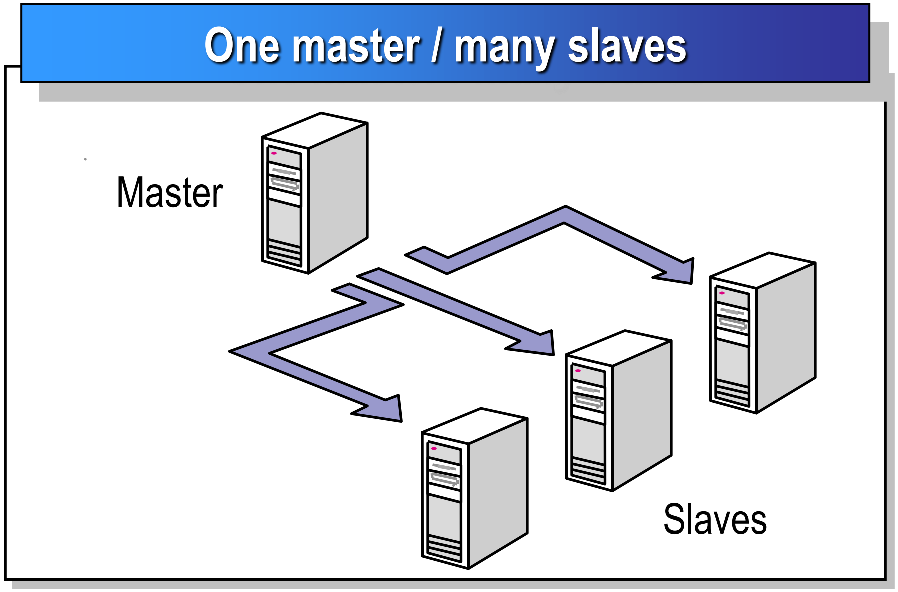
Filtering Data (in general)



Source: Microsoft

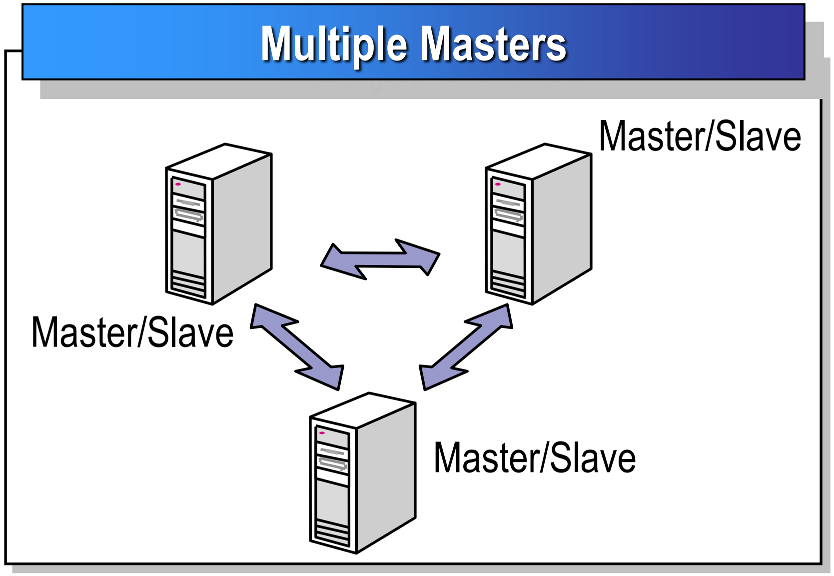
Distribution Model: Replication

- 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
  - Consistency needs resolving update conflicts
    - In “real” master-master model (or peer-to-peer)
  - Typically, with sharding (filtering) data
    - Master for a subset of data
    - Slave for the rest



Master-Master Model with Sharding

Orders (Master A)

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

Orders (Master B)

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

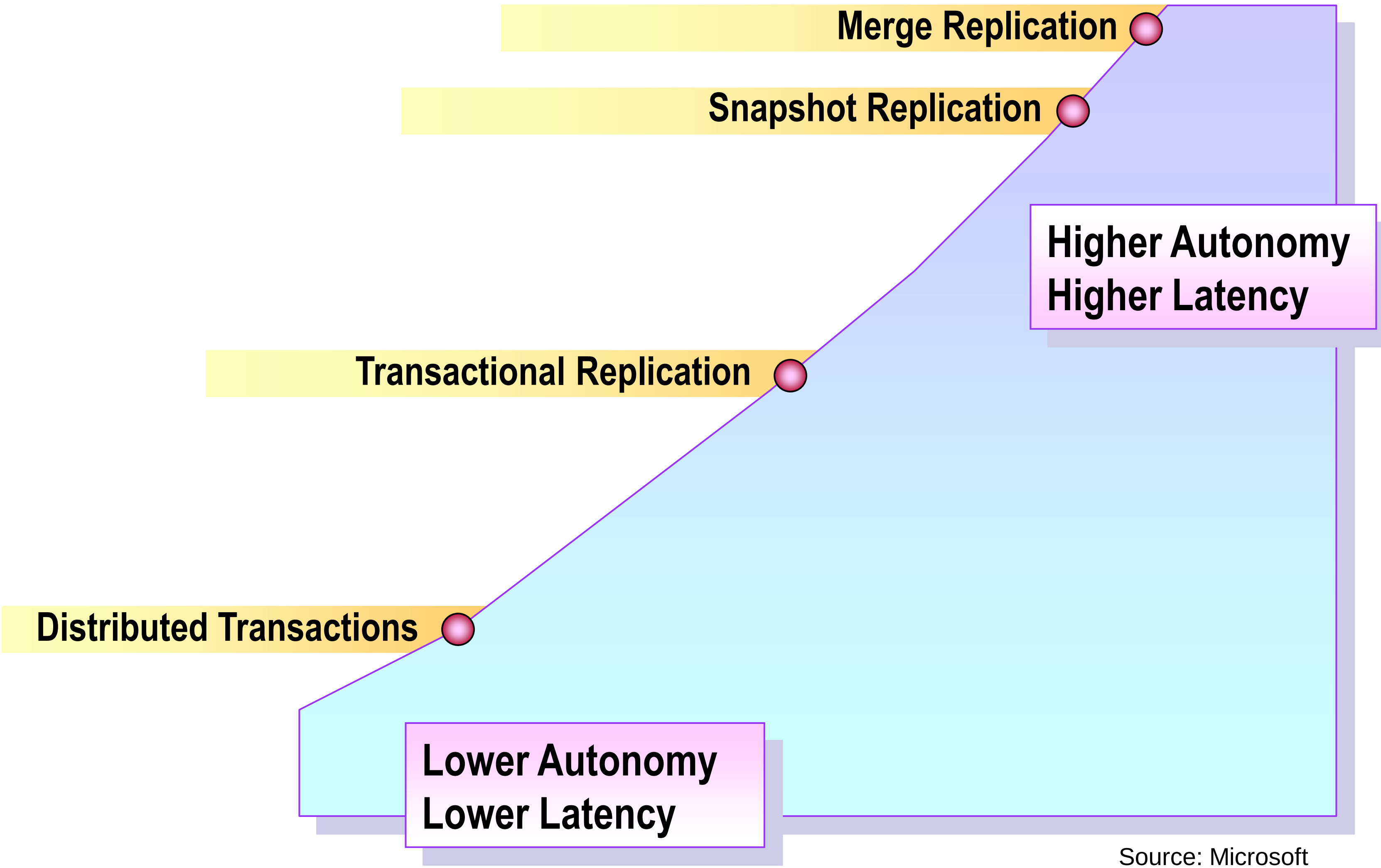
Master/Slave

Orders (Master C)

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

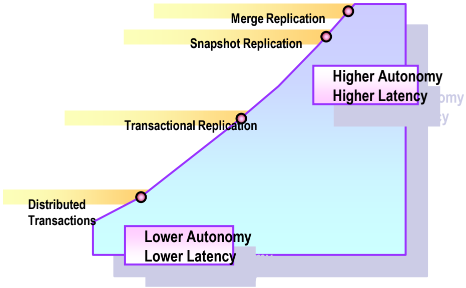
Replication Types (for "real" multi-master model)



Source: Microsoft

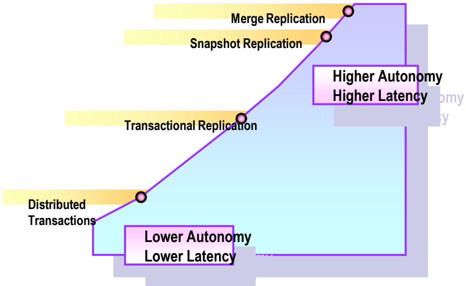
Replication Types

- Distributed Transactions
  - For “real” master-master model, ensures consistency
  - Low latency, high consistency
- Transactional Replication
  - Replication of incremental changes
  - Minimal latency (typically online)
  - Conflicts are solved using shared locks



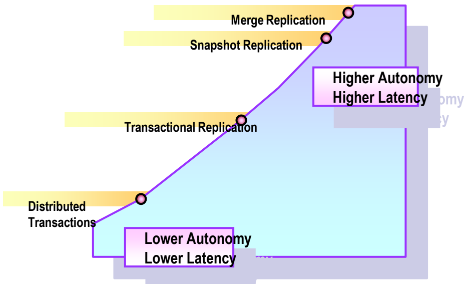
Replication Types

- Snapshot Replication
  - Periodic bulk data transfer as new snapshots
    - Intermediate updates to data might be unnoticed by “subscribers”
    - So, copies can be out-of-date
  - Data changes – substantial but infrequent
  - Replica servers are read-only
  - High latency is acceptable



Replication Types

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

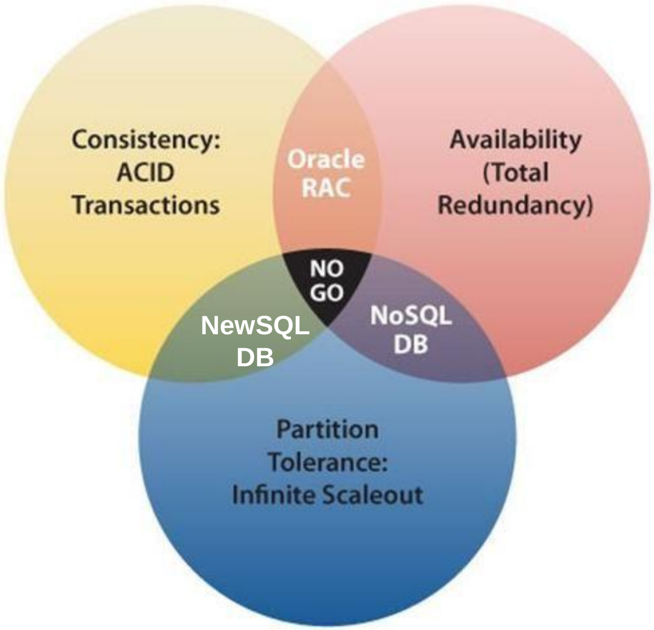


Issues of Distributed Systems

- Consistency
  - After an update, all readers in a distributed system see the same data
  - All nodes are supposed to always contain the same data
  - 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., replication with distributed transactions
- BASE
  - provides Availability and Partition tolerance
  - Reality: you can trade a little consistency for some availability
  - E.g., sharding with merge replication



Source: http://bigdatanerd.wordpress.com

NewSQL

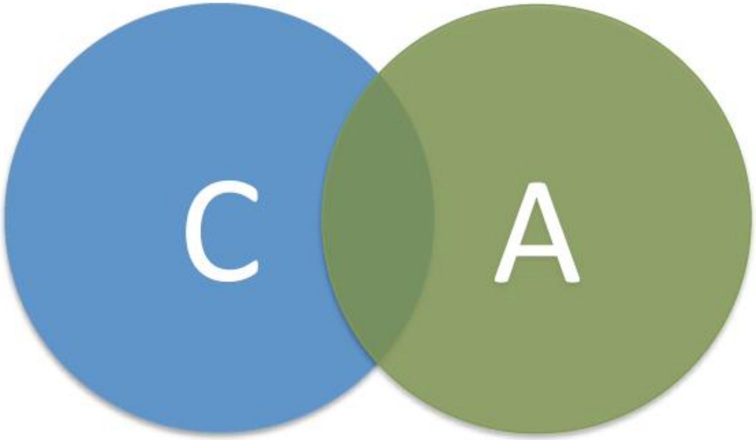
- Distributed database systems that scale out
- CP systems
  - trade availability for consistency when partition occurs
- MySQL cluster, Google Spanner, VoltDB, …
  - In fact, master-master replication with data sharding

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
    - Read requests eventually return the same value
  - is not safety guarantee
    - can return any value before it converges

Two out of three ain't right

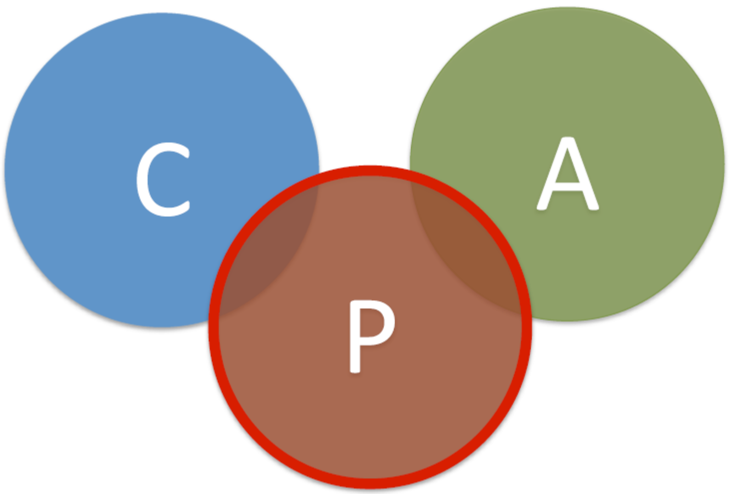
- Can a distributed system be not tolerant of partitions?



- Partition Tolerance is mandatory in distributed systems!
  - Network partitions are a given; therefore, consistency and availability cannot be achieved.

Two out of three ain't right

- In reality, there are two types of systems when there is a partition:
  - give up availability
  - give up consistency.



- Incorrect conclusion:
  - Consistent, partition tolerant, databases cannot be available, and are therefore not feasible

Consistency or Availability?

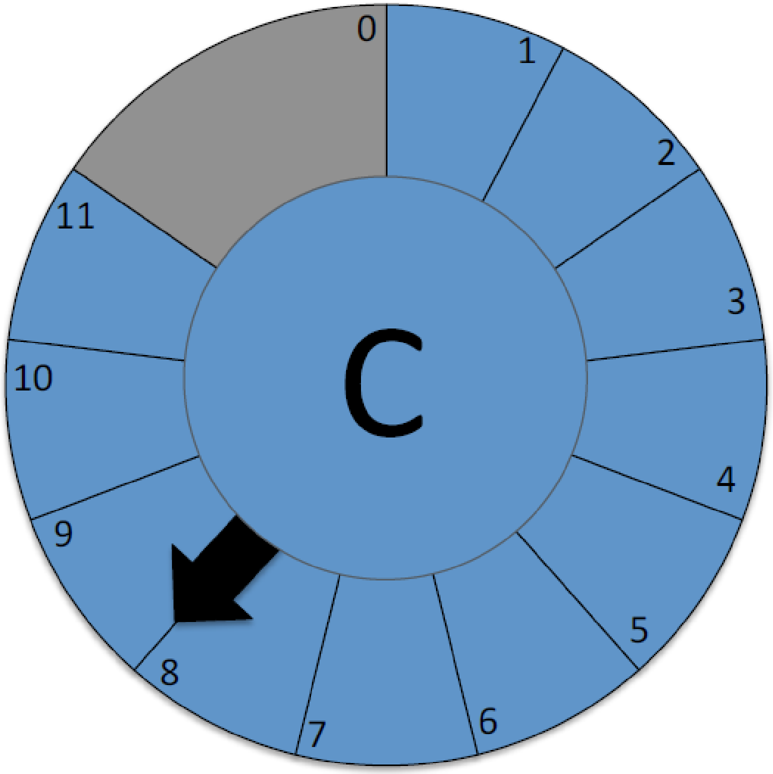
- NoSQL databases relax consistency in favor of availability
  - but are not inconsistent.
- NewSQL databases sacrifice availability for consistency
  - but are not unavailable.

- When partition occurs, make the decision.
  - Rather make the choice during the development process

Tunable Consistency

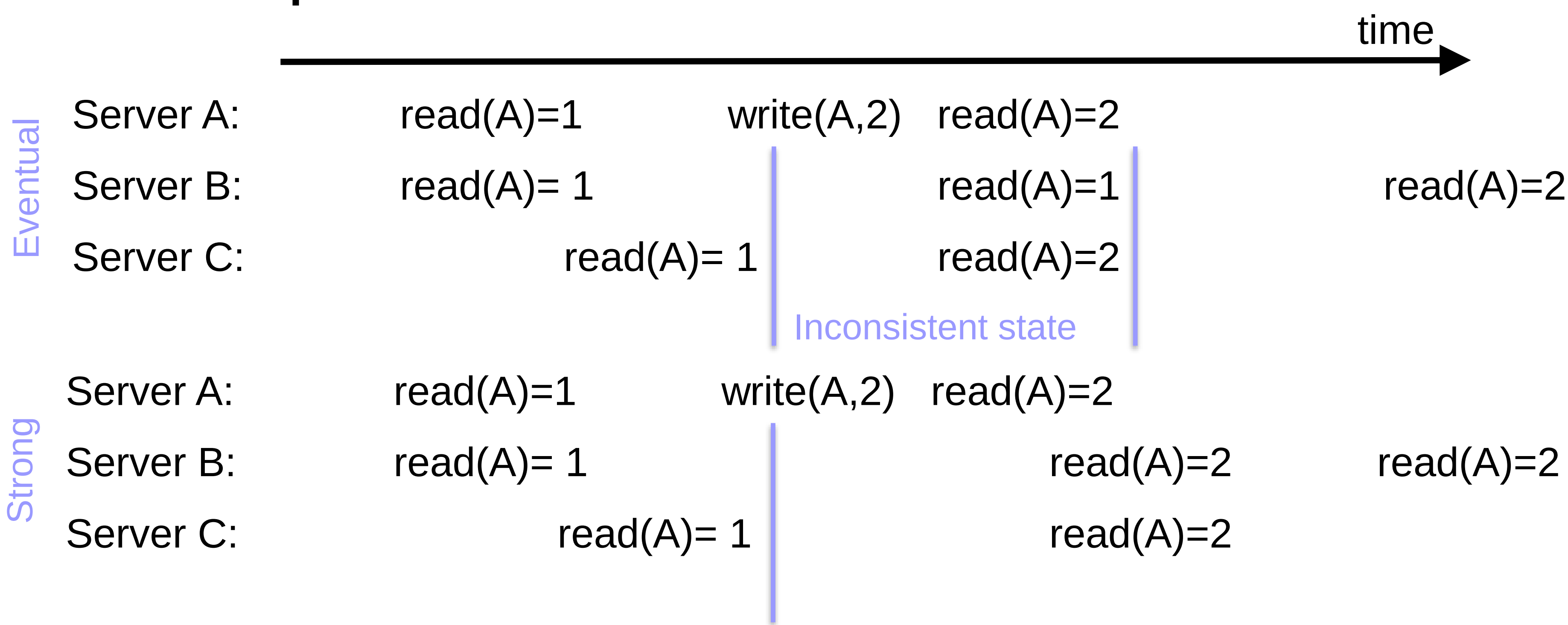
- Allows developers to adopt a consistency model required for a specific application, workload or query.

- Strong vs Eventual consistency



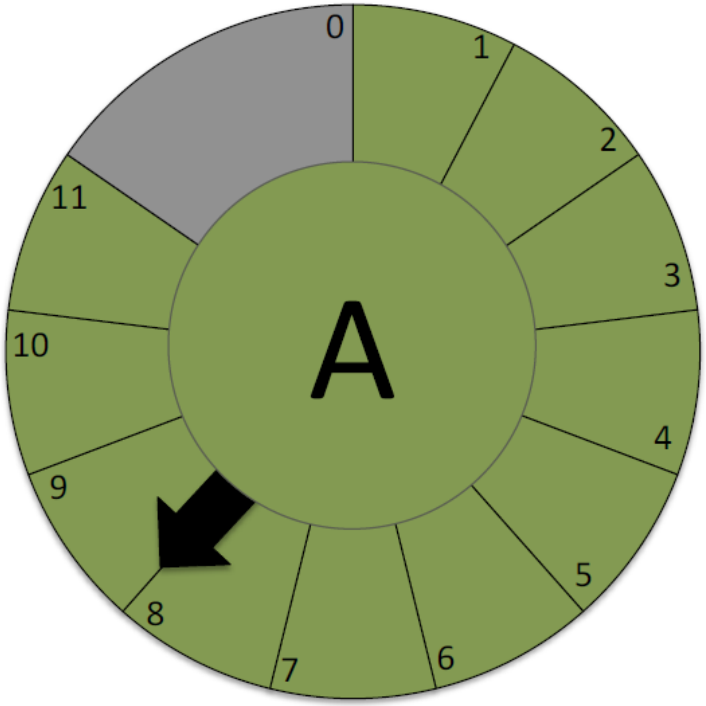
Consistency

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



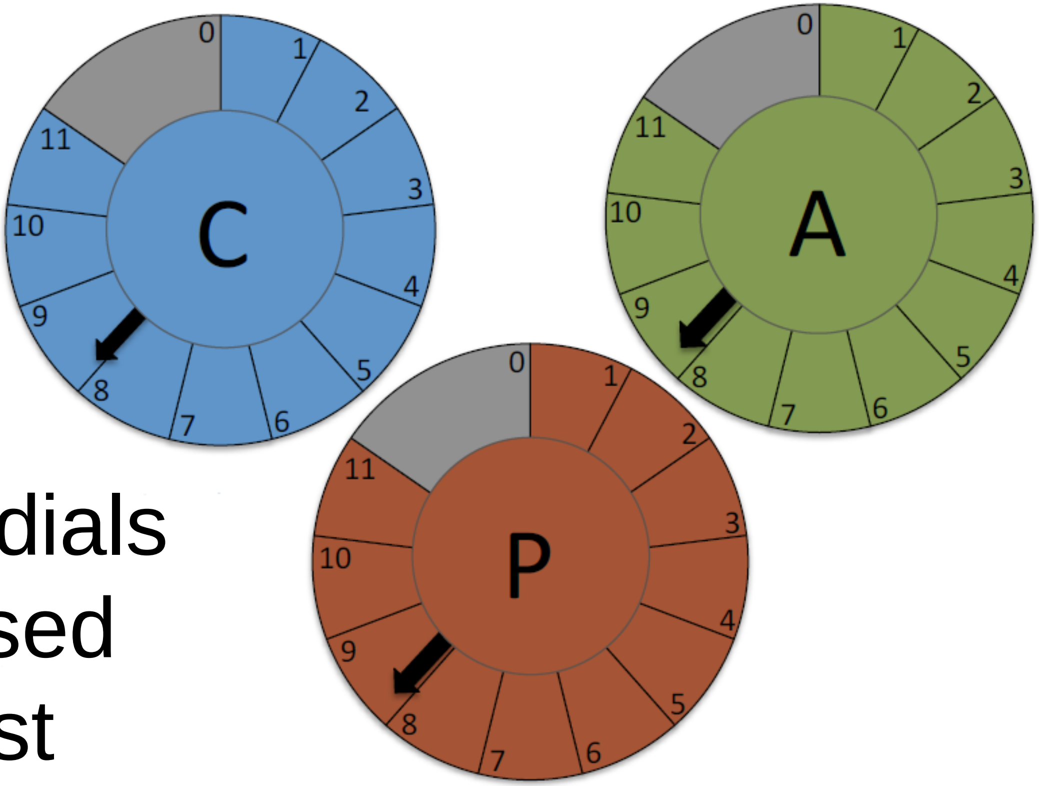
Tunable Consistency

- Would allow developers to adopt an availability model required for a specific application, workload or query.
- But tunable availability is a trade-off of tunable consistency
- It is called **latency.**



Beyond CAP Theorem

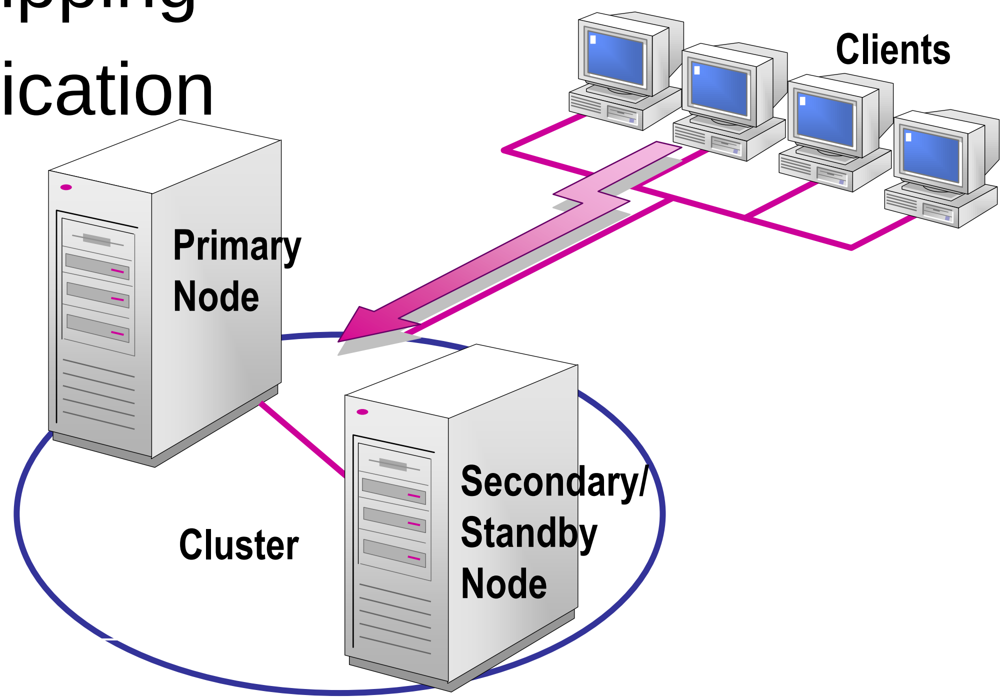
- It's not "pick two"
- Or even pick one



- Think off them as dials on a dashboard used to achieve the most appropriate balance of consistency, availability and partition tolerance
  - for specific workloads

Maintaining High Availability of DBMS

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



Log-shipping Standby Server

- 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
  - Also called warm standby
- Log shipping can be synchronous/asynchronous
- Disadvantage: all tables are replicated typically
- Advantage: no schema changes, no trigger definitions

Log-shipping: Failure of a node

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

Log-shipping: After failover

- When failover by standby succeeded,
  - a new standby should be configured.

- When the original primary node becomes available
  - → detect 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

- Old primary usually becomes new standby.

Primary and Standby Servers

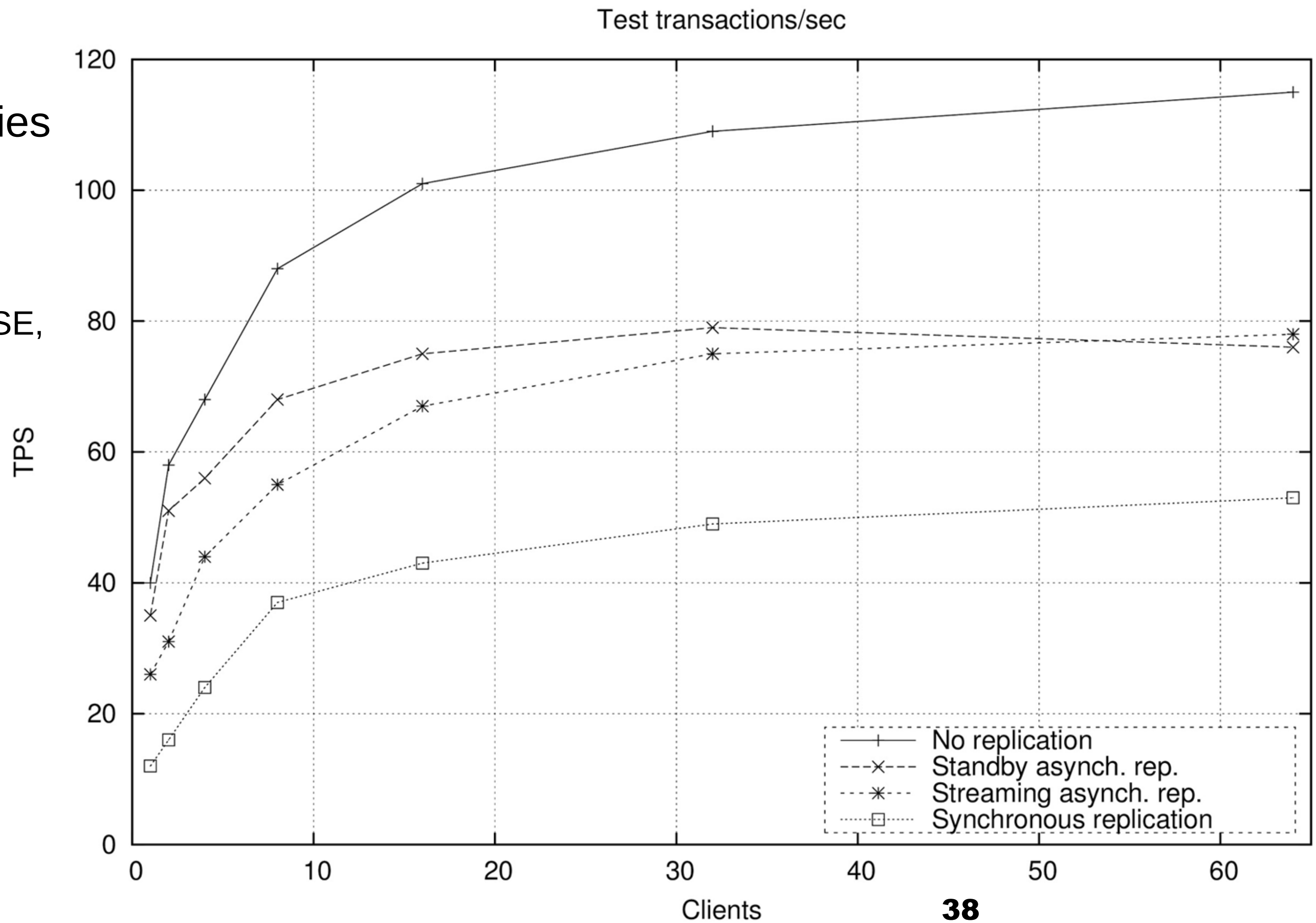
- Swap primary and standby regularly
  - To verify recovery steps
  - To do necessary maintenance on standby server
    - SW/HW upgrades, …
    - Mind limits on SW upgrades – see DBMS docs

PostgreSQL: Replication

- TPC Benchmark B

scale factor 1
1 trans. = 5 queries

2x server
(AMD Opteron 8439 SE,
1024 MB RAM,
20 GB HDD)



Recommended HA Practices

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

Lecture Takeaways

- Term of Availability and its classification
- Possible techniques (sharding / replication)
- CAP Theorem
  - ACID & BASE systems
- Know possible implementation in relational DBMS

- The future of databases is distributed.