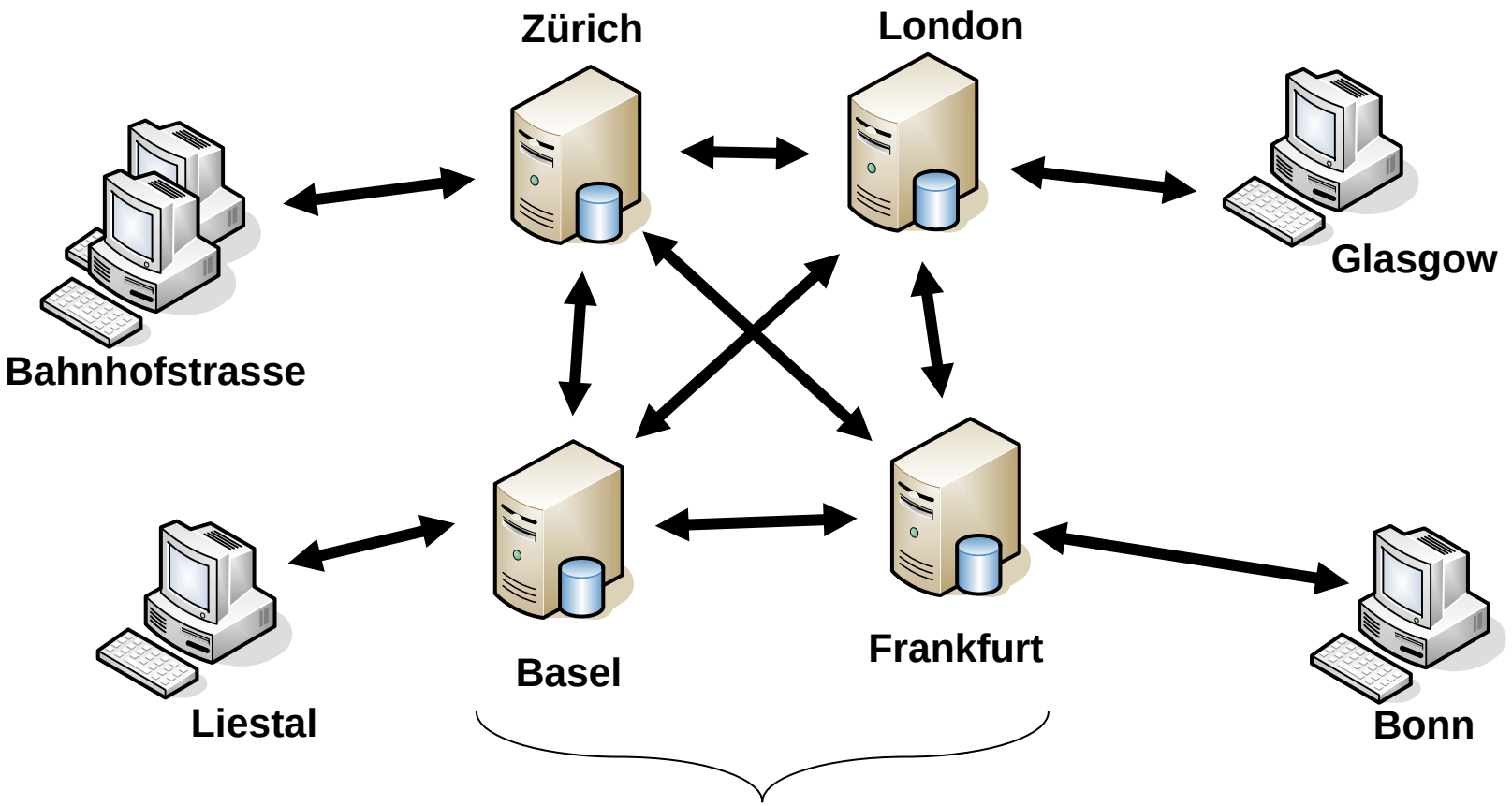
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Quorums

Christian Plattner, Gustavo Alonso
Exercises for Verteilte Systeme WS05/06
Swiss Federal Institute of Technology (ETH), Zürich
{plattner,alonso}@inf.ethz.ch

Setting: A Replicated Database (Wide-Area)

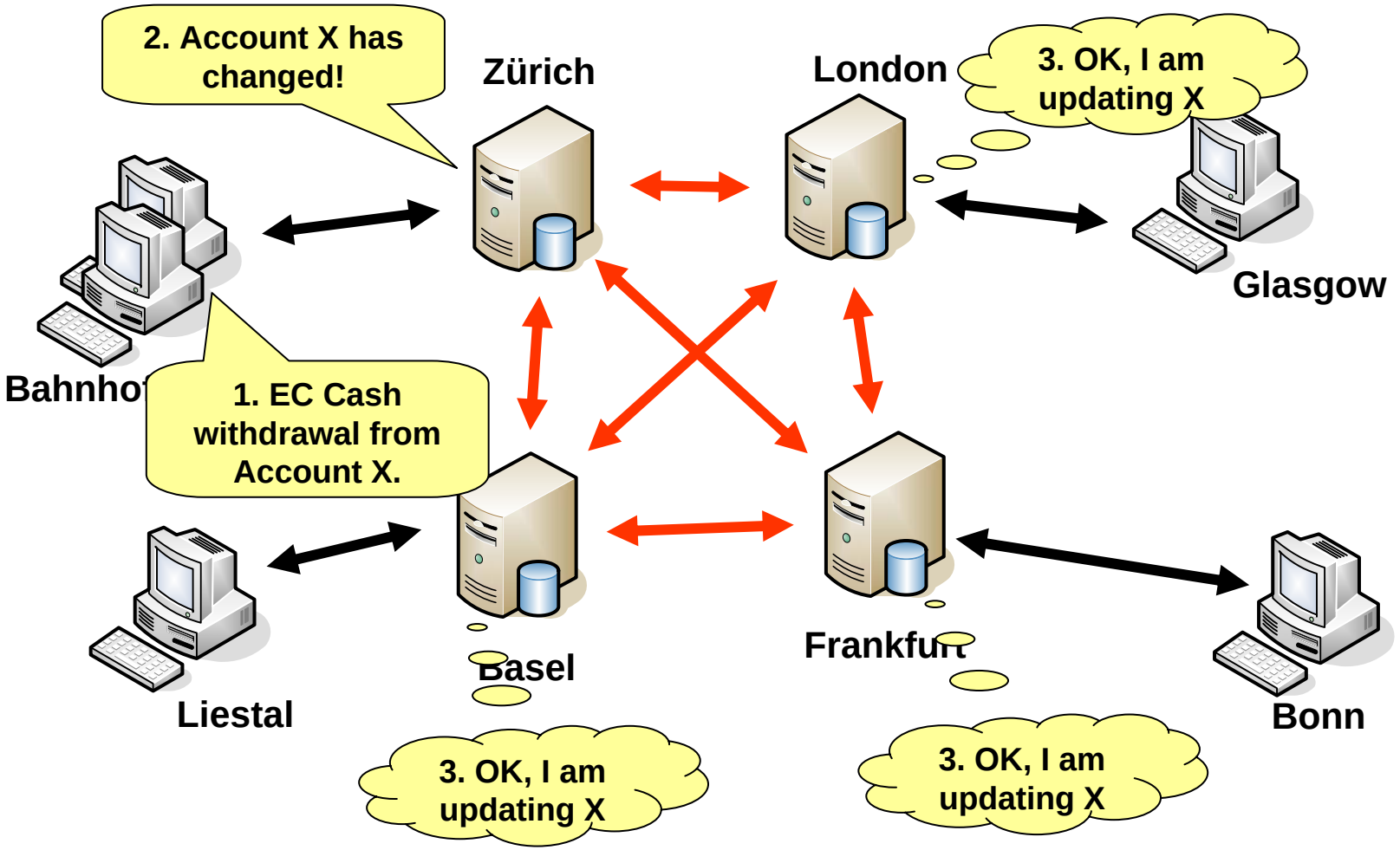


We assume full replication (all sites contain the same data)

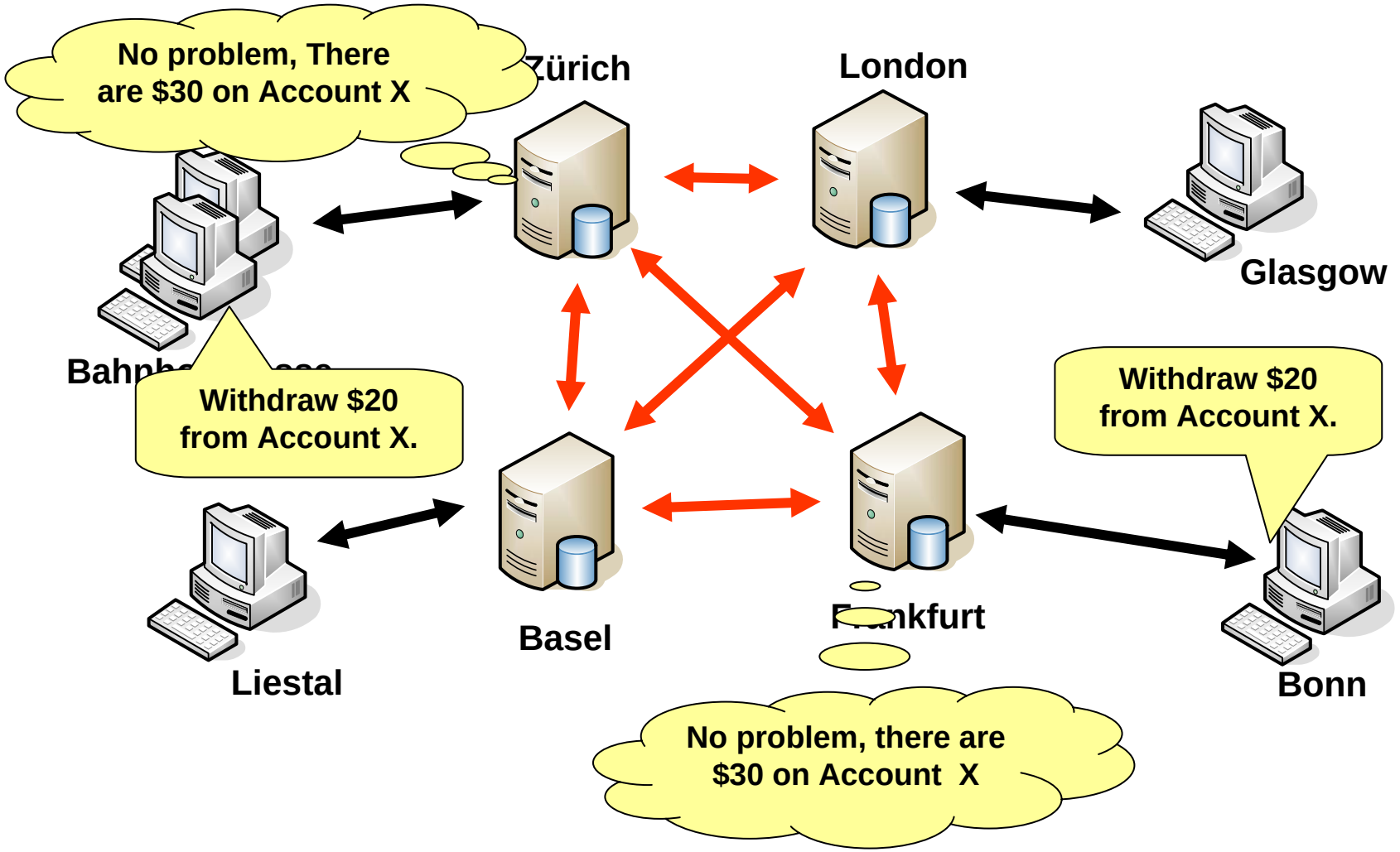
System Model

- A ***replicated database*** consists of a group of *n* sites which communicate by exchanging messages.
- Here we assume full replication, each site has a copy of the full database.
- Clients interact with the database by issuing transactions.
- Transactions are partially ordered sets of **read** and **write** operations
- Transactions are executed atomically, i.e., a transaction either commits or aborts.
- Typically, one distinguishes between two kinds of transactions:
- → Read-Only Transactions (also called Queries, consist only of read operations)
- → Update Transactions (read and write operations)
- A client submits a transaction to one of the sites in the system, the ***local site***, the rest of the sites are called the ***remote sites.***

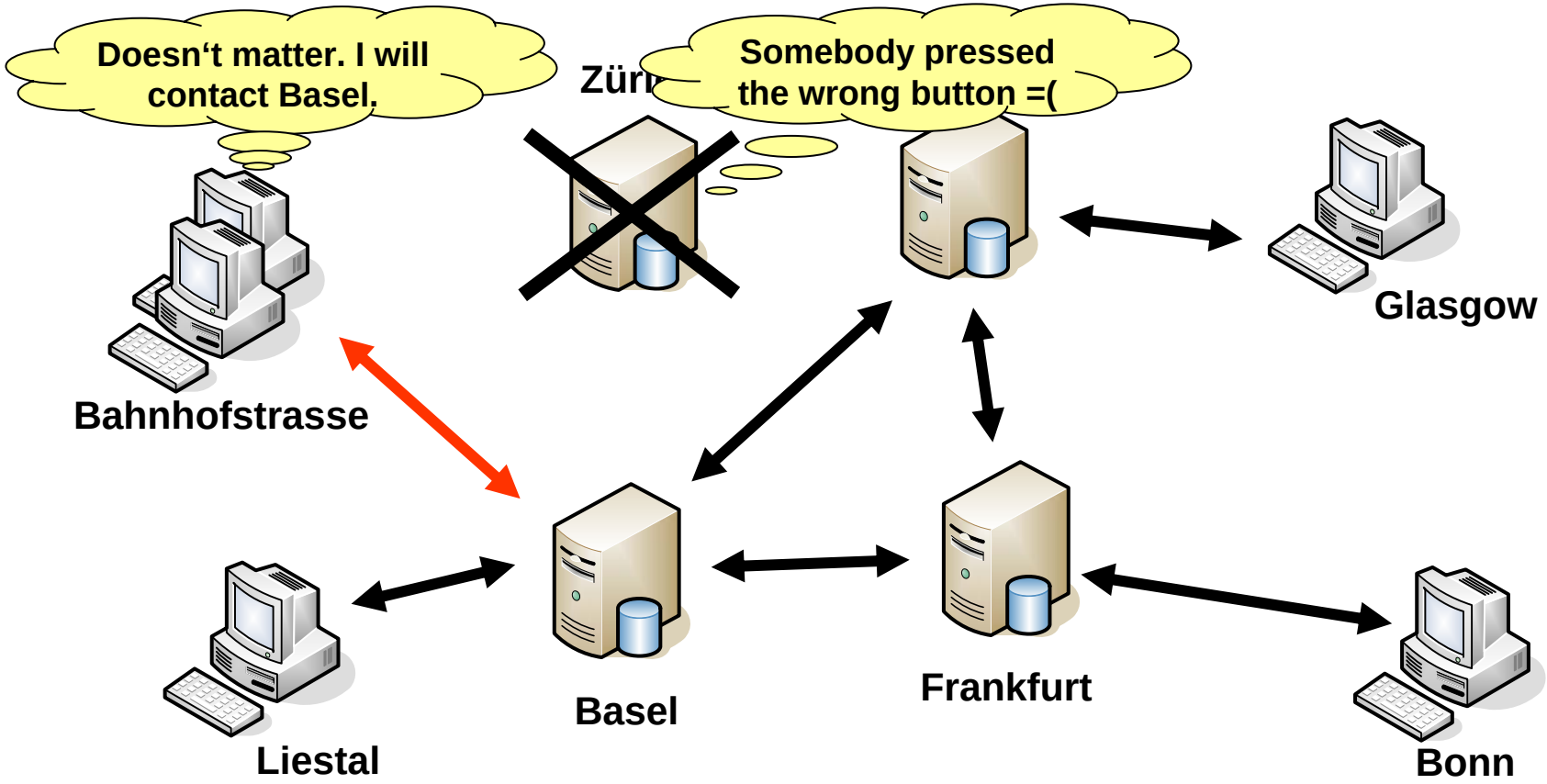
Problem: Synchronization and Consistency (a)



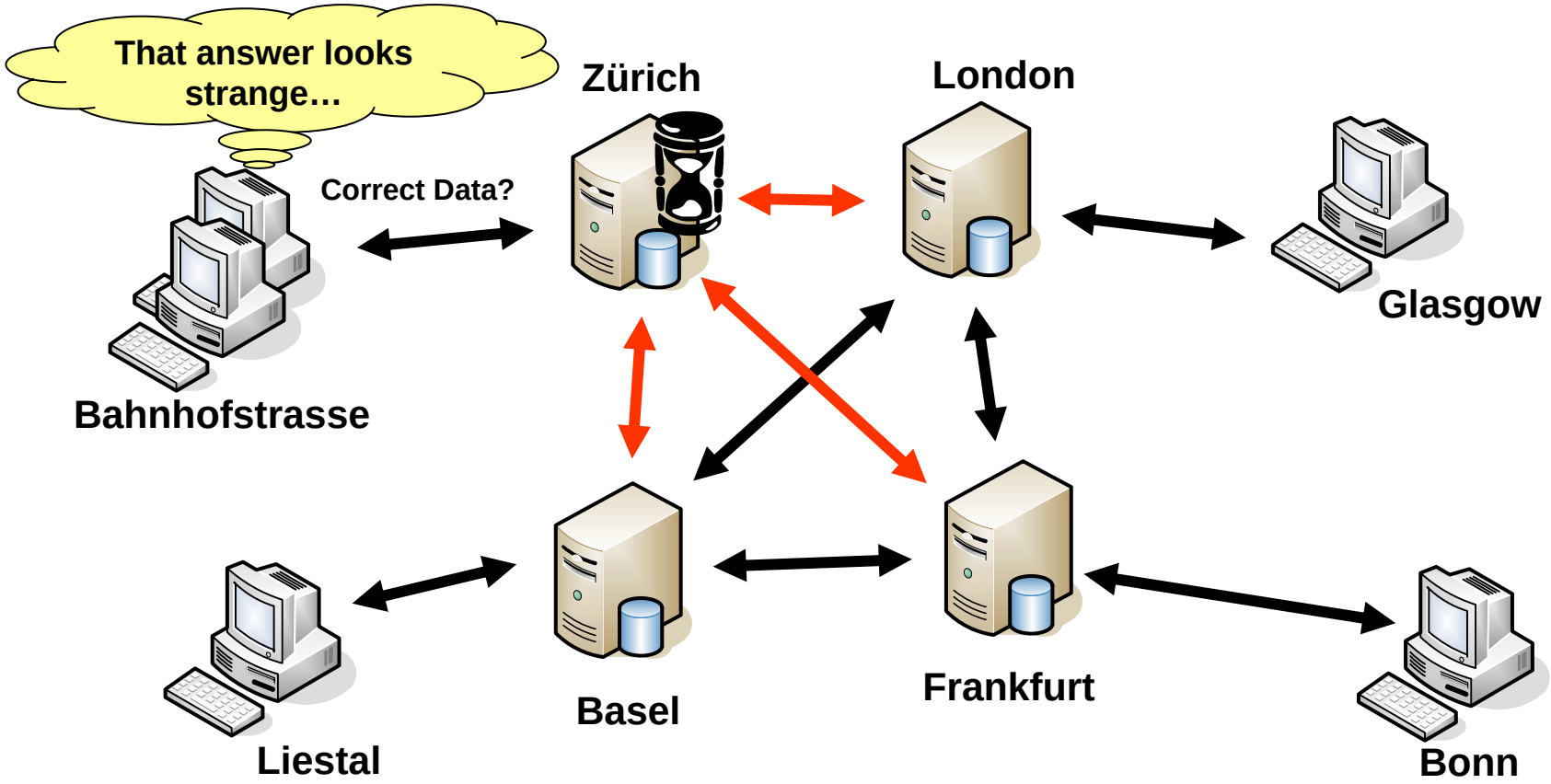
Problem: Synchronization and Consistency (b)



Problem: Site Failures…

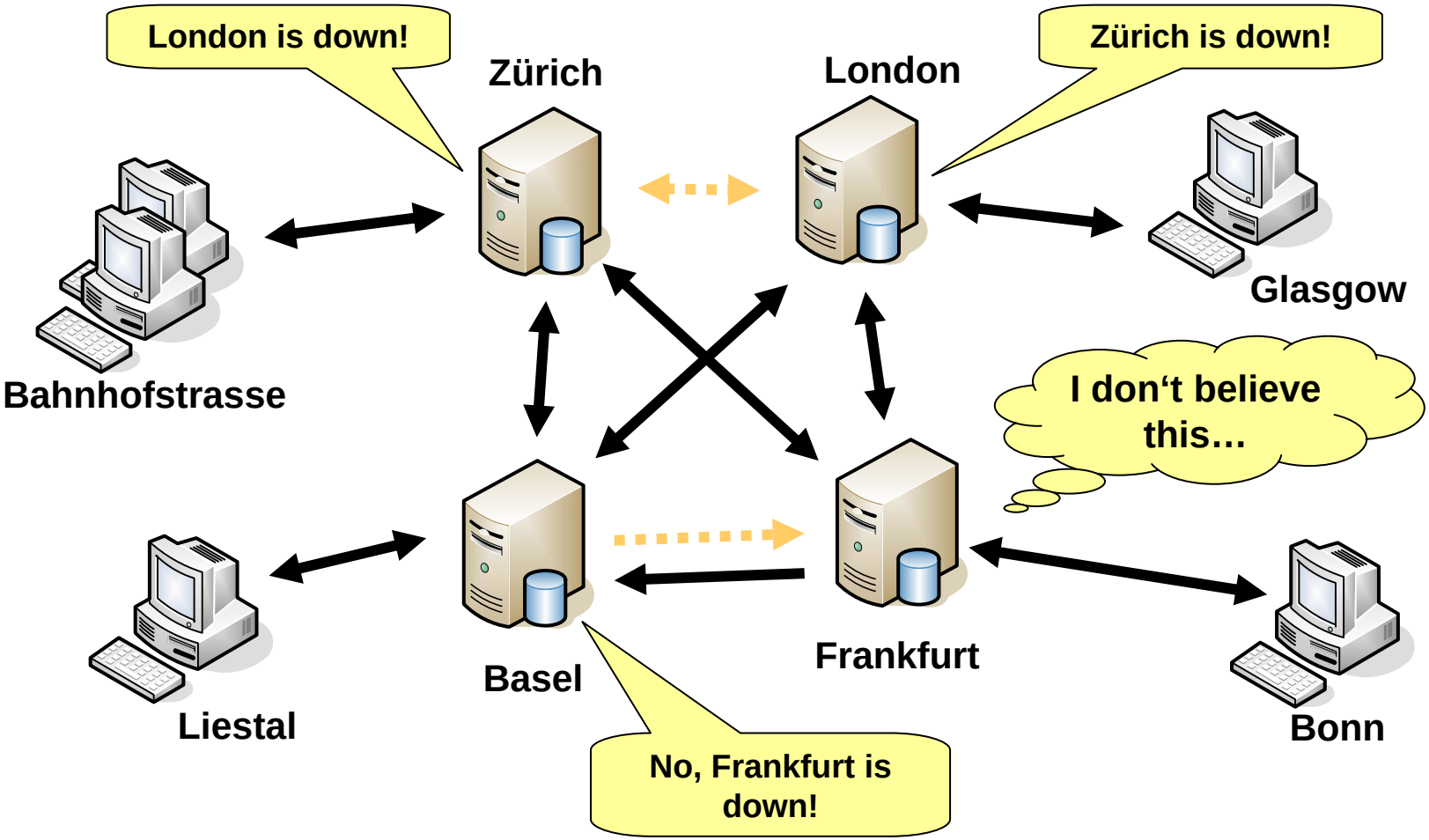


Problem: …and Recovery

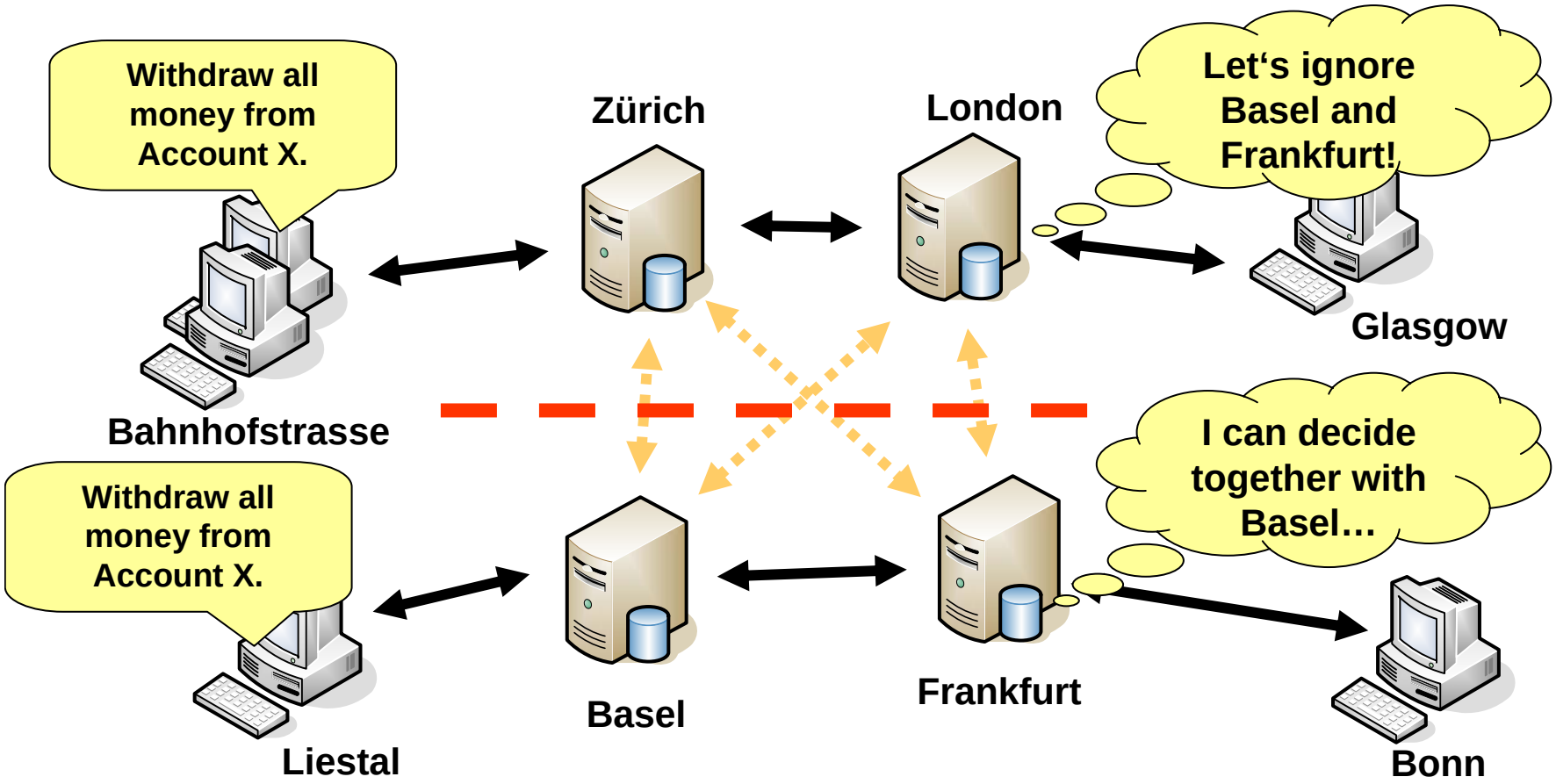


→ Site Zürich has to be updated to the freshest state otherwise clients should not be allowed to use it.

Problem: Communication Failures



Problem: Network Partitions



→ Partitions must not process transactions independently!

Replicated Databases

- **Advantages:**
  - More powerful (more CPUs, disks, etc. that can process requests in parallel)
  - Clients can access the „nearest“ site (latency).
  - Improved fault-tolerance.
- **Disadvantages:**
  - Sites have to be constantly synchronized. Consistency has to be guaranteed (e.g., conflicts between transactions).
  - Sites may fail and have to be properly recovered once they come up again.
  - We have to deal with network partitions.

ROWA - Read-One Write-All

The protocol belongs to the *synchronous/update everywhere* category.

- Assume all sites contain the same data (*Full Replication*).
  - Each site uses traditional 2-Phase Locking (2PL).
  - **Read operations** are performed locally.
  - **Write operations** are performed at all sites (using a distributed locking protocol).
  - Also, we assume that there are no communication failures.

This protocol guarantees that every site will behave as if there were only one database. The execution of transactions is *serializable* (correct) and all reads access the latest version of a data element.

ROWA - Read-One Write-All (II)

This simple protocol illustrates the main idea behind replication, but it needs to be extended in order to cope with realistic environments:

- Sites may fail, which reduces the availability (once a single site fails, no site can be updated anymore).
- Sites eventually have to properly recover (a recently restarted site may not have the latest updates).

Handling Site Failures: ROWAA

An approach which tolerates site failures:

Read-One Write-All-Available.

Again, we assume that there are *no communication failures*. The protocol could then be implemented as follows:

- **Read Operations**: Read from any site. If a site is down, try another site.
- **Write operations**: Write to all sites. If any site rejects the write (e.g. there is a conflict), abort the transaction. Sites that do not respond are ignored („missing sites“).
- **Committing a transaction**: Check that all missing sites are still down, if not, abort the transaction. Check that all other sites are still OK, if not, abort the transaction. Otherwise the transaction can commit.

Problem: Communication Failures

- The ROWA(A) approaches do not work if the system has to deal with communication failures → but this is a must for real systems.
- Communication failures can happen in different ways:
  - Sites seem to be down, but actually just the communication layer is temporarely down.
  - The network between the sites is partitioned, only partial communication is possible.

- Example: The network gets partitioned in to equal parts of sites: Then both subsets can operate at their own (they both think that the other machines are down) → overall database state can get inconsistent (overdrawing of bank accounts etc. possible).

Quorums

- Quorums are sets of sites which have certain properties regarding inter-sections of different quorums.
- Can be used to handle the so far discussed problems (site failures, recovery, communication failures and network partitions).
- Can reduce the number of copies involved in updating data.
- Costs of reads and writes can be balanced.

Definition: Quorum Systems

- Let $S = \{S_1, S_2, \ldots\}$ be a set of sites. A **quorum system** Q is **a set of subsets of S** with **pair-wise non-null intersection**. Each element of Q is called a **quorum**.
- Example: We have four sites, S_1, S_2, S_3 and S_4. A possible quorum system then consists of these three quorums: $\{S_1, S_2, S_3\}$, $\{S_2, S_3, S_4\}$ and $\{S_1, S_4\}$. There are many other possible quorum systems for these four sites!
- For replication purposes, two different kinds of quorums are defined, **read** and **write quorums:**
  - Any **read quorum** (rq) must overlap with any **write quorum**
  - Any **two write quorums** (wq) must overlap

Quorum System Types

- Voting quorums (weighted majority, hierarchical)
- Grid quorums
- Tree quorums

- …we just look at some of them.

Weighted Majority (Quorum Consensus)

- Uses voting to reach consensus
- Each site has an assigned weight (number of votes).
- Quorums are defined so that number of needed votes exceeds half of the total (→ majority).
- Let **n** be the sum of all assigned weights.
- Read and write quorums must then fulfill these constraints:

- **2 * |wq| > n** and **|rq| + |wq| > n**

- Minimum quorum sizes that work:

$$|wq| = \left\lfloor \frac{n}{2} \right\rfloor + 1 \qquad |rq| = \left\lceil \frac{n}{2} \right\rceil$$

Algorithm for Quorum Consenus

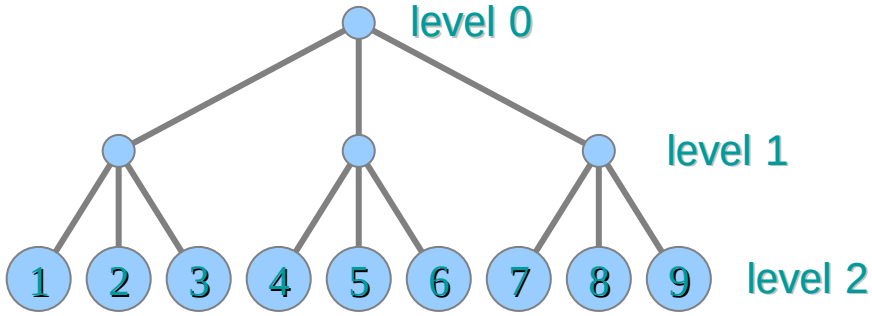
- Each site uses versions to tag data items.

- **Reads:** contact sites until a read quorum is reached. Then use the data item with the highest version number.
- **Writes:** contact sites until a write quorum is reached. Get the highest version number of the data item to be written. Increase the version number and write the data item to all members in the quorum.

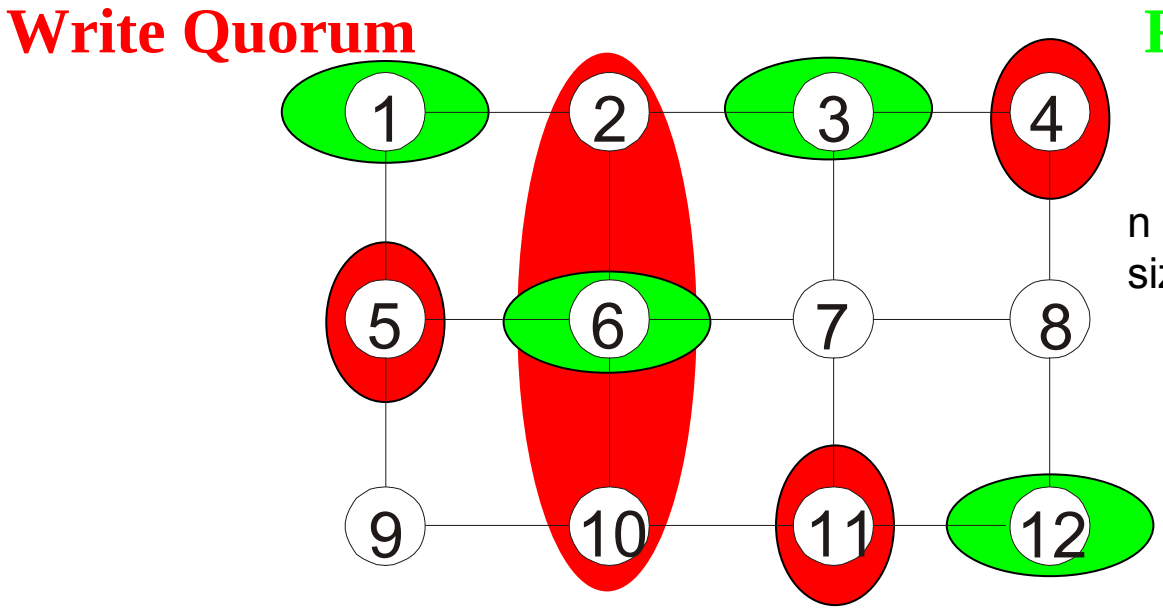
- **Recovery is already included!**
- But reads are now more expensive than in ROWA approaches…
- Dynamic reconfiguration (changing assigned votes, adding new machines) not easy, must be done in an atomic step (hard to solve when having to deal with communication failures).

Hierarchical Quorum Consensus

- A generalization of Majority Quorum. Idea: organizing the sites into a hierarchy.
- This hierarchy is represented as a complete tree where physical sites appear at the leaves of the tree.
- At each level (starting at the root level) of the tree, a majority of tree nodes must be chosen.
- For each node chosen at ***level i***, a majority of nodes at ***level i+1*** must be chosen.



Grid Quorums: Rectangular Grid

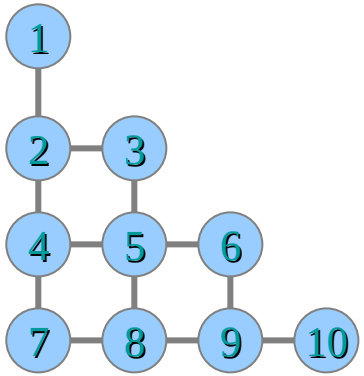


n sites are organized in a grid of size *r x c* (*r* rows and *c* columns)

n = r x c

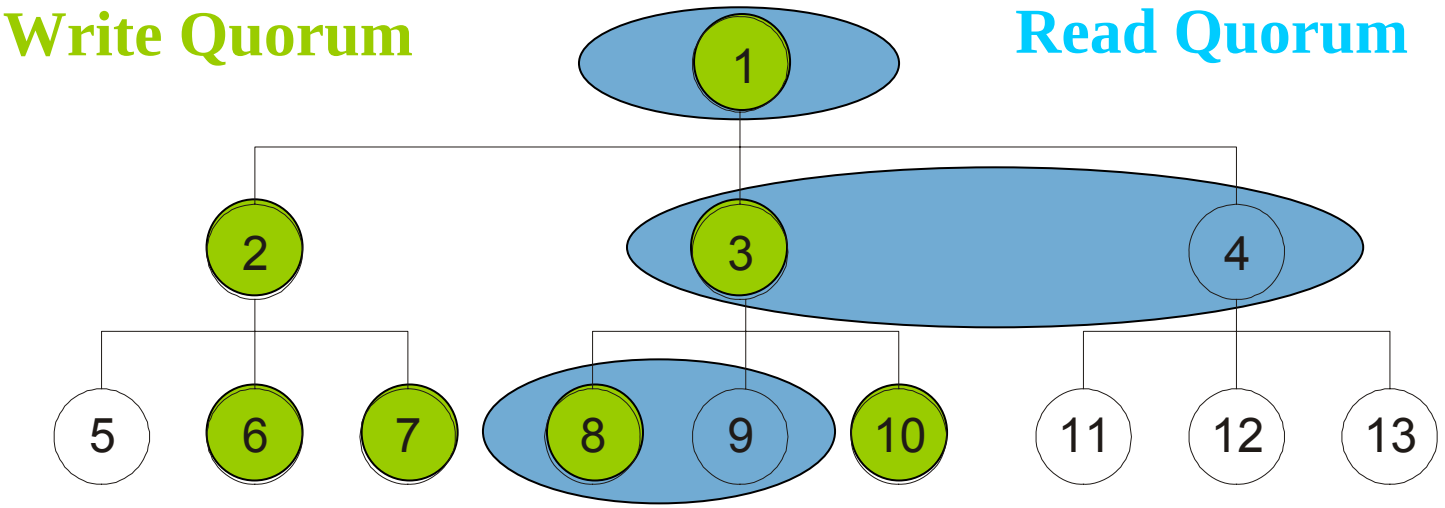
- A **read quorum** consists of an element of each column (|rq| = c)
- A **write quorum** requires an entire column and one element from each of the remaining columns (|wq| = r + c - 1)
- If the grid is a square → **SQUARE grid**: |rq| = √n |wq| = 2 * √n -1

Another Grid Quorum: Triangle



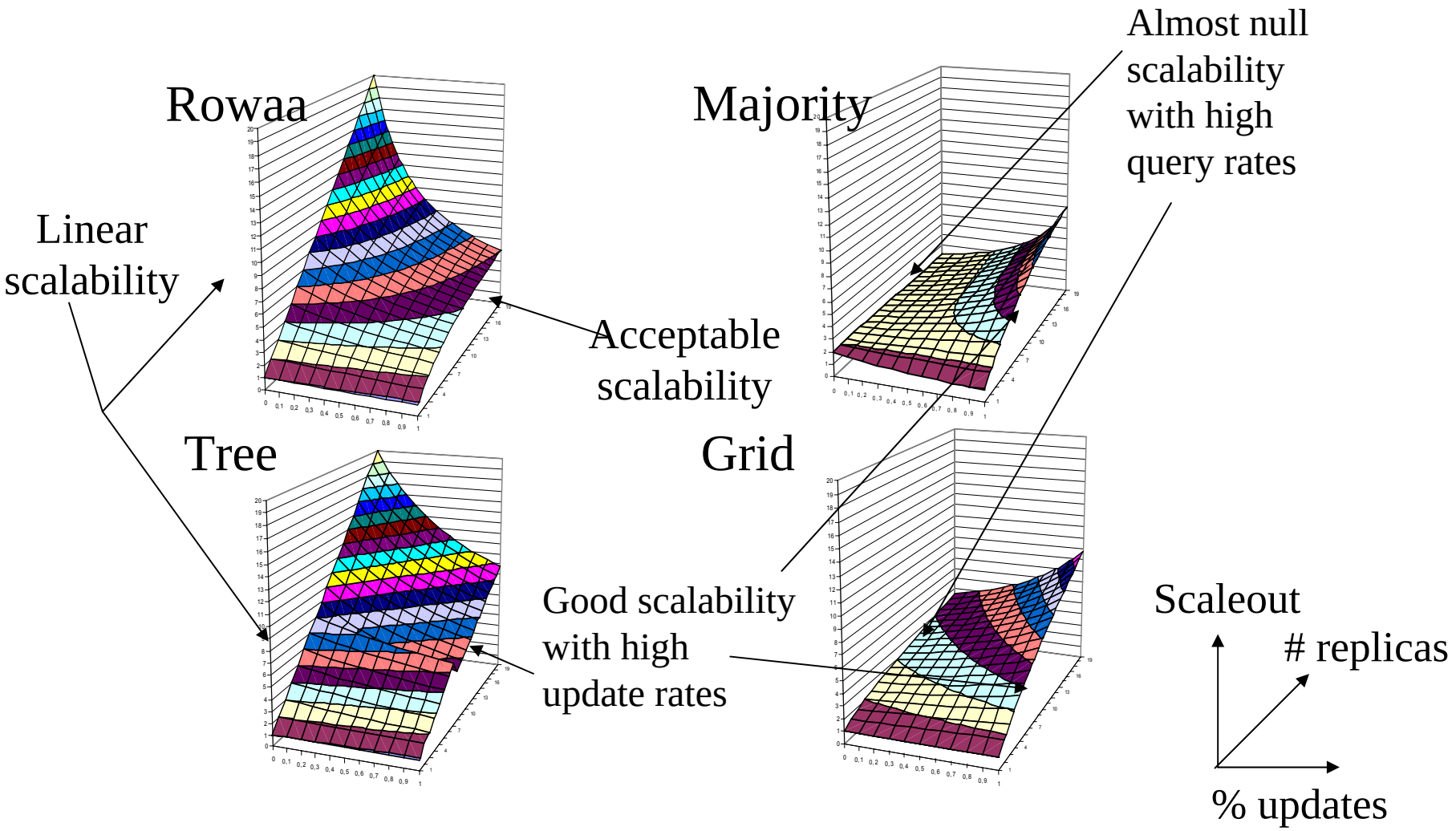
- Sites are arranged in ***d*** rows such that row *i* (*i* > 1) has *i* elements.
  → number of sites must be 1 or 3 or 6 or 10 or 15 …
- The quorum size is always ***d***.
- A **write quorum** is the union of one complete row and one element from every row below the full row.
- A **read quorum** can be either one element from each row or a write quorum.

Tree Quorums: A Basic One



- Sites are organized in a complete tree of an odd degree.
- Each node has ***d*** children → not any amount of sites possible.
- A **write quorum** consists of the root of the tree, a majority of its children, a majority of the children of each children, etc.
- A **read quorum** consists of the root of the tree. If the root is unavailable, the read quorum consists of a majority of its children, and so recursively.

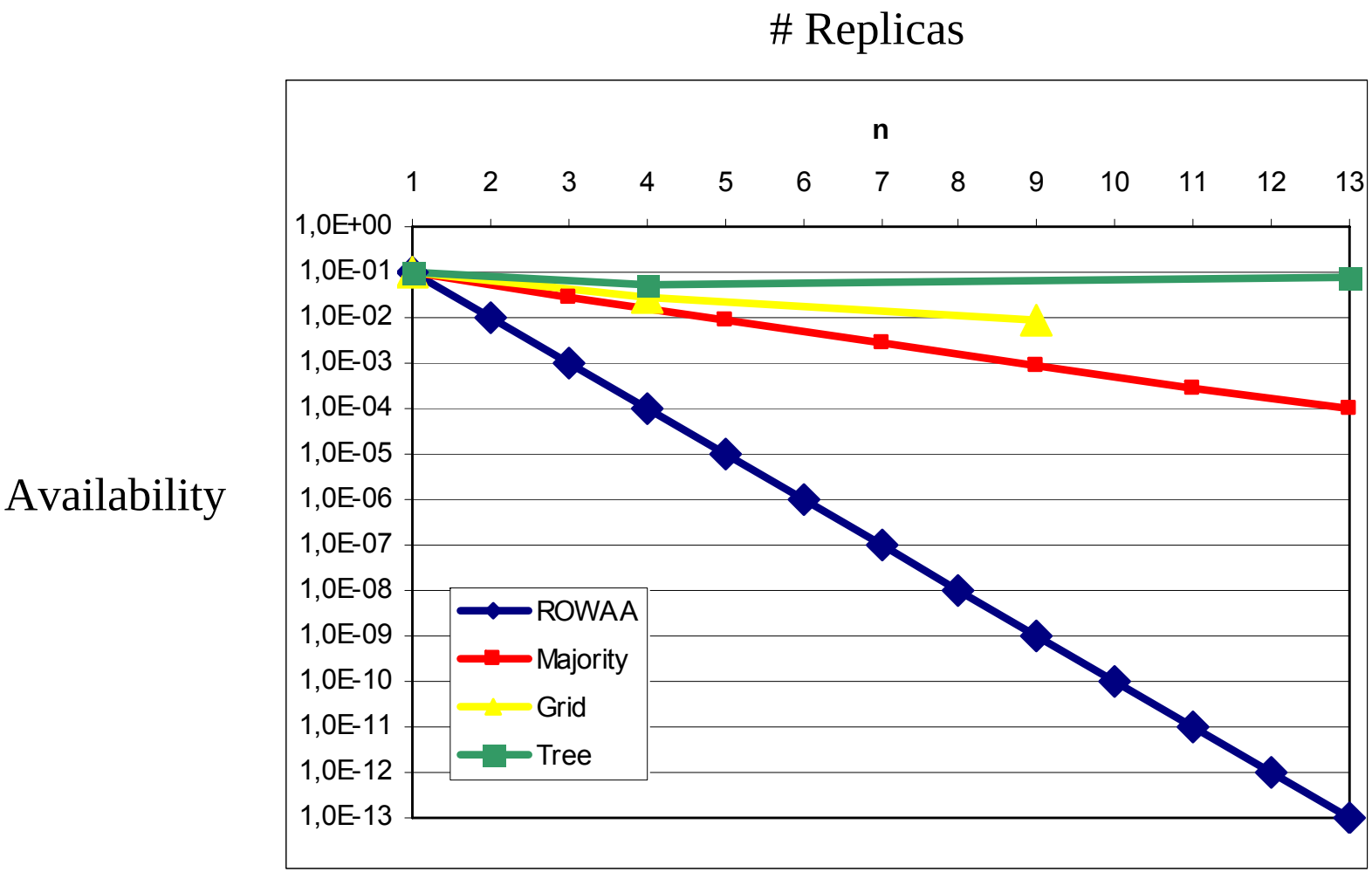
Performance: Scalability



Availability

- To be fair, some of the quorum protocols were designed for availability and not for scalability purposes
- However, there is a tight trade-off between availability and scalability [Naor98]:
  - For scalability: the smaller the quorum, the better
  - For availability: the larger the quorum, the better

Availability: Comparison

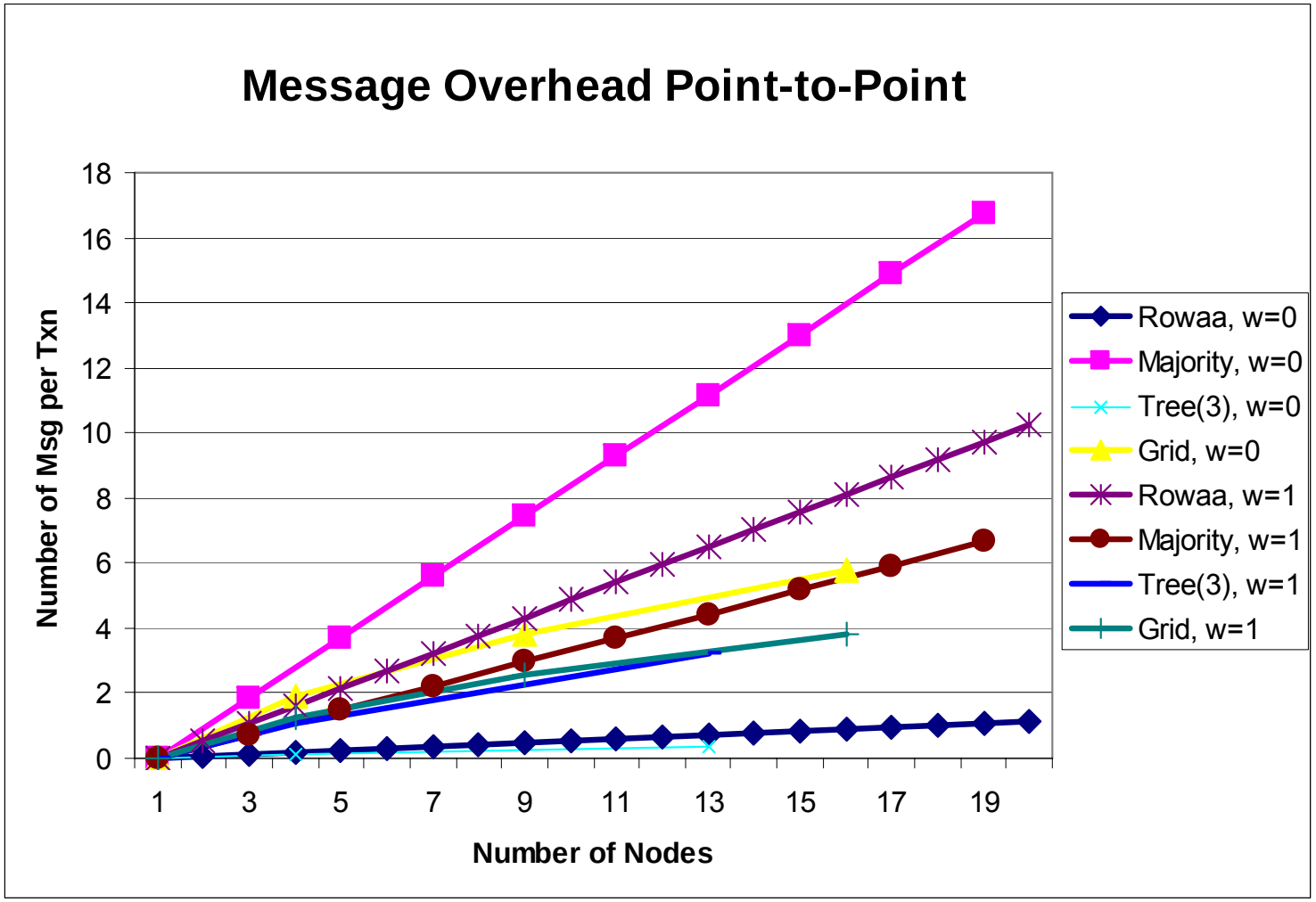


Performance: Communication Overhead

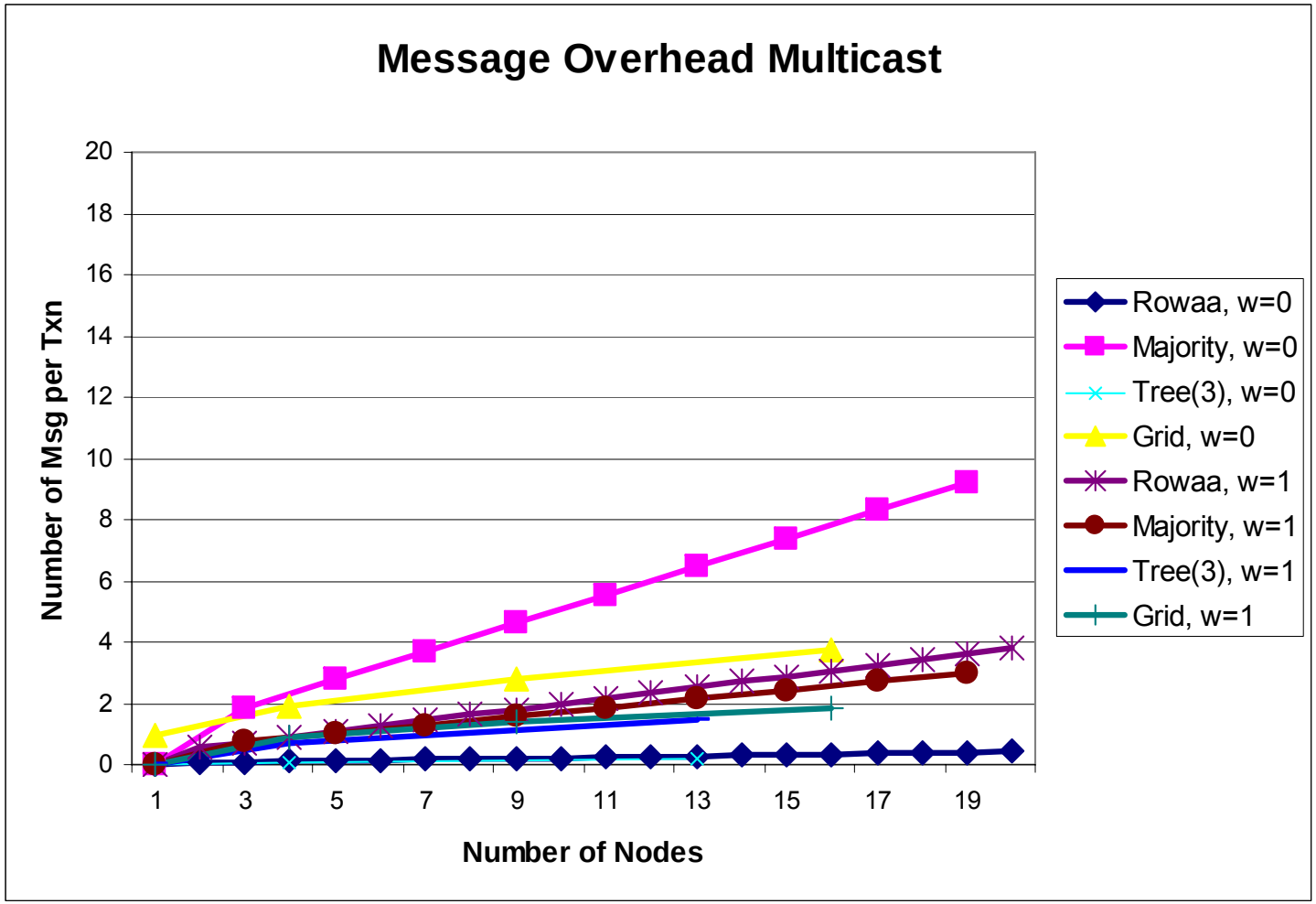
- Replication requires coordination among the replicas involving exchange of messages.
- This traffic has an impact on the overall scalability (i.e., if the system gets faster by adding more sites):
  - CPU cycles are lost handling messages
  - Network bandwidth may become a bottleneck
  - Transaction throughput is a key aspect:

    #of mssgs/s = TPS * #of mssgs/txn
- The message overhead is divided into overhead per read or write operation and overhead per transaction. The latter overhead is due to the use of distributed atomic commit protocol and update propagation.

Communication Overhead: Comparison



Communication Overhead: Comparison



Literature And References

- R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso and B. Kemme. **Are Quorums an Alternative for Data Replication?** *ACM Transactions on Database Systems*, 2003. http://portal.acm.org/citation.cfm?doid=937598.937601

- Naor, M. and Wool, A. **The load, capacity, and availability of quorum systems**. SIAM J. Comput. 27, 2 (Apr.), 423–447. 1998

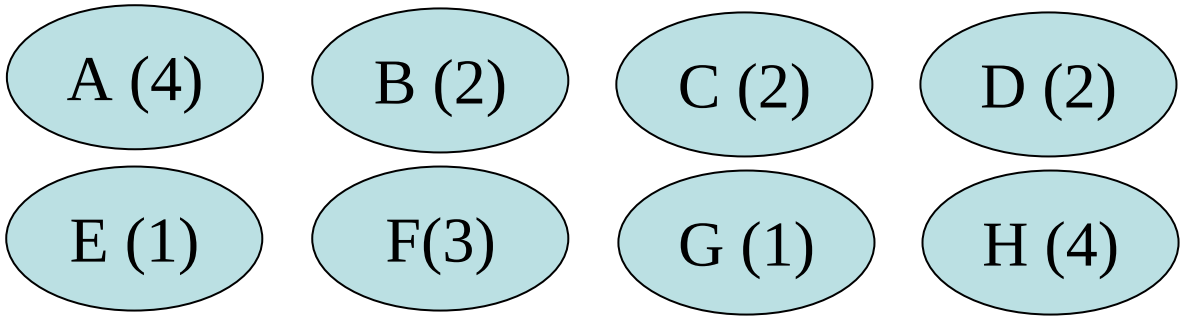
  http://epubs.siam.org/sam-bin/dbq/article/28123

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Paper Exercise

Question 1:

- In the majority quorum system, every replica (site) has one vote. This implies that every replica has the same priority. In real systems it is necessary to create replicas with higher priority, in case that these replicas are located in more powerful or robust machines (→ weighted majority quorum). Write and read quorums still need majority votes to proceed. Users can assign different priorities to each replica according to their needs. Please list all the possible write/read quorums for the largest weighted majority quorum system based on the sites (with weights) in the following figure:



Question 2:

- Compared to quorum systems, ROWAA approaches are simpler, more flexible and efficient. However, ROWAA has a fatal flaw if the network suddenly gets partitioned.
- How can one solve the problem by integrating some of the quorum concepts into the ROWAA approach?
- What are the advantages and disadvantages of the new system?

Question 3:

- Assume again that we have a replicated database setup with 4 fully replicated sites. Each of the sites holds the customers of a bank, the bank is present in 4 different countries (and each site is positionied in the headquarter of one of the countries).
- If the network gets partitioned in a way that each site is disconnected from any other site, then the so far proposed protcols cannot help, at least 3 sites have to stop processing updates.
- Please describe a way in how the problem can be solved on the application layer, so that every site can at least continue to process *some* updates.