UNIT VI

FAULT TOLERANCE

Q what is fault tolerance

① Fault tolerance is the ability of a system to continue operating and providing services in the presence of faults and failures.

② (A) Failure - It is a state of inability to perform normal function

(B) Error - It is an act involving an unintentional deviation from truth or accuracy

(C) Fault - It is the cause of an error that may lead to failure

③ Faults can arise from hardware malfunction, software errors, network errors or any unexpected disruptions

④ Fault tolerant systems are designed to handle such failures gracefully and ensure that the systems overall availability is maintained.

⑤ In distributed systems - Failures are partial, i.e some components might still be working.

In order to hide the effects of faults and recover from them, we need dependable systems

Following are requirements of dependable systems:

① Availability:

- It refers to ability of distributed systems to provide services to users and applications without any interruptions or downtime.

② - Reliability:

- Reliability is the measure of how well a DS performs its intended functions correctly and consistently over time

③ - Safety:

- Fault tolerance is essential in systems to prevent catastrophic consequences caused by faulty behaviour

④ - Maintainability

- Maintainability refers to the ease with which a distributed system can be repaired, modified and updated

Types of Faults

(1) Transient Faults :-

- occur only once and then disappear
- Eg: disturbance during wireless communication

(2) Intermittent Faults

- occurs randomly
- cannot easily be detected.

(3) Permanent Faults

- continue to exist until faulty components are repaired / replaced.

* Failure Models

- Failure models are formal or informal description of the types of faults that a system can experience.

- It helps to identify the potential source of failures, allowing system designers to reason about the systems behaviour and aids in designing appropriate fault-tolerance mechanisms.

(1) **Crash Failure** : A server or node in distributed systems halts abruptly and stops responding to any further communication.

- It can be due to a software crash hardware failure or sudden loss of power.

(2) **Omission Failure**

- A server fails to respond to a request from another server or client.

- this could happen due to nlw failures, message losses or processing delays.

(3) Timing failures - A servers response time exceeds the specified deadline causing timeouts or delays in the system's overall operation.

(4) - Response failure :- , when a server or node responds to request but the request is INCORRECT or VIOLATES expected behaviour.

(5) - Arbitrary failure also known as Byzantine failure -

- This refers to type of failure where server behaves arbitrarily or maliciously.
- It may send incorrect data or provide inconsistent responses, etc.
- They are especially hard to handle these failures as they violate the assumptions of traditional fault tolerant mechanism.

Resilience

* Process Resilience

- Resilience can be defined as the capability of the system to understand and manage failures occuring in the system.

- Resilience ∝ uptime and availibility

- In DS, critical processes may be REPLICATED and run on multiple nodes ensuring that even if a node or process fails, there is a backup instance ready to take over the ~~world~~ work load.

* Design Principles of Resilience in DS. How Failure masking can be achieved:

(1) Redundancy: Implement multiple instances of critical components that failure in one instance does not ~~disturb~~ disrupt the entire system

(2) Fault Isolation: Design the system so that faults are contained within specific components or modules, preventing them from cascading and affecting other parts of system.

(3) Graceful Degradation: Ensure that the system continues to operate, albeit at reduced functionality or performance when parts of it fails.

(4) Load balancing: Distribute work loads evenly across multiple servers or nodes to prevent any single point of overload.

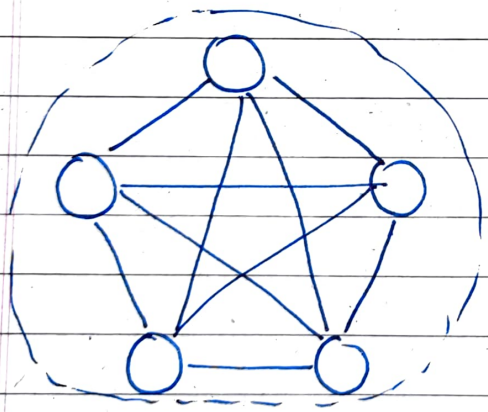
(5) Consistent state Management: Use distributed consensus algorithms and state replication to maintain a consistent view of the system state.

(6) Monitoring and Alerting: continuously monitor system performance and health and setup alerts for anomolies or failures to enable quick responses

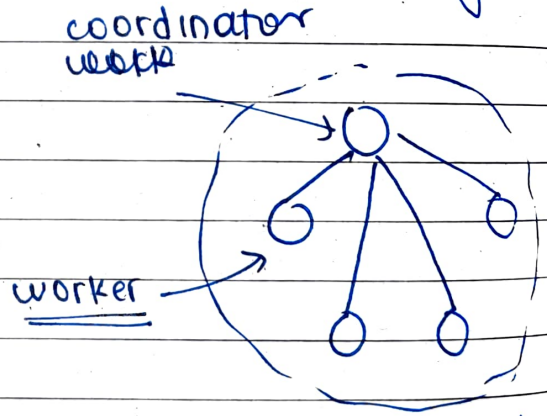
(7) Scalability: Design the system to scale horizontally by adding more nodes than scaling vertically.

1. Process Reliance by Process Group.

- Process reliance can be enhanced by organising ~~to~~ processes into process groups.
- Process groups are sets or related processes that work together to achieve a common ~~goal~~ goal.
- These groups ensure that if one of the processes fail, others can continue the operation without system interuptions.
- Group organisation can be done in two ways
  - Flat Group
  - Hierarchical group



Flat group



Hierarchical group

(A) - communication in flat group.

- All processes are equal, decisions are made collectively.

- Note: No single point of failure but decision making is complicated as consensus is required

- Good for fault tolerance as information exchange ~~occ~~ occurs immediately with group members

(B) communication in hierarchical group.

- one of the processes is elected as a coordinator which selects another process (a worker) to perform the operation.

- Note: single point of failure, however decisions are easily and quickly made by the coordinator without first having to get consensus.

Consistency and Replication

Replication and Reasons for Replication.

1. Replication of data is key technique used in distributed systems to enhance both system reliability and performance

2. By maintaining multiple copies of data on different nodes, the system becomes more fault tolerant. If one node fails, data can be ~~achieved~~ accessed from other replicas, ensuring high availibility and data durability.

3. Reasons for Replication are as follows

(1) Reliability / fault tolerance

1- Replication ensures reliability by providing fault tolerance.

2- If one node or replica fails, the system can continue to function

3- By keeping multiple copies of data, the system becomes resilient to node failures.

(2) High Availibility

1.- Replication enhances availibility as users can access replicated data or services even if some parts of system are down or unreachable

(3) Performance Improvement

- Load Balancing : By distributing the request across multiple replicas, the system reduces load on single server leading to better response times.

2- Reduced latency: Replicas can be placed closer to the user (i.e in different geographical locations) reducing time it takes to access data or services.

(4) Concurrency

1- In distributed systems, multiple clients may access the same data simultaneously.

2- Replication allows concurrent read operations across multiple nodes, improving system throughput.

* Replication as a scaling technique.

1. Replication is a powerful scaling technique that helps to handle increasing workloads by distributing operations across multiple replicated objects.

2. Scalability addresses how a system can grow to handle more significant workloads without significant degradation in performance. In other words, as more users, ~~of~~ data or traffic are added, the system should be able to still maintain acceptable performance

3. Distributed workloads

(A) Horizontal scaling: Replication enables horizontal scaling by adding more replicas to system.
As workload increases, additional ~~we~~ replicas can be introduced to share the load preventing any single node from becoming a bottleneck.

(B) Load Balancing: Replicas can be distributed across multiple servers and requests can be evenly distributed among these replicas. This reduces the pressure on individual servers and improves overall system performance

(C) Geographical distribution:
Placing replicas in different geographical locations brings data or services closer to user reducing latency and improve performance

⑤ Fault isolation :

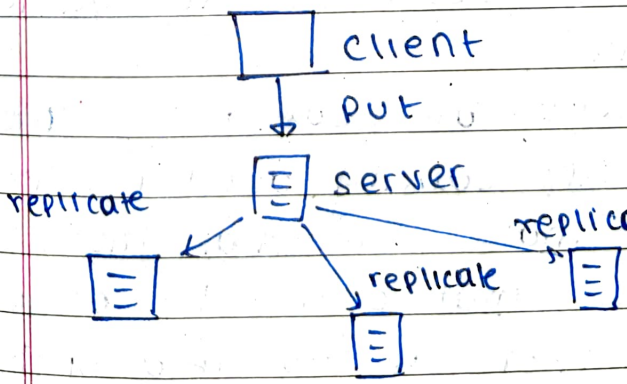
By having multiple replicas, failures in one part of the system do not necessarily affect the availibility of entire system, making it more robust.

* Data consistency Models

1. Data consistency models define a level of consistency that must be maintained for replicated data in distributed systems.
2. These models focus on how data is replicated, synchronised and accessed by multiple users and systems.

(I) Strict consistency

1. In this type of consistency, all nodes of system must agree on the order in which operations occurred.
2. Reads will always return the most recent version of data
3. When update occurs on one server, every other system reflects the changes immediately.
4. This model provides highest level of consistency but it can be slower.



(II) Sequential consistency

1. Sequential consistency is a consistency model that ensures that the order of operations executed in different processes appear to be consistent with global order of execution

2- All processes see the same sequence of reads and writes, but order of those operations may differ from the actual order in which they occurred

3- It is easier to implement than strict consistency and suitable where operation ordering matter.

(III) Casual consistency

1. In this type of consistency, all related events occur in the same logical order.
2. If two events are causally related (like one event causing another), the system will make sure that they are seen in that order by all users.
3. However if there is no relationship blw two operations, the system does not enforce any order, meaning different users might see operations in different sequence
4- It balances consistency and performance

(iv) Weak consistency

1. Weak consistency is a consistency model that provides no guarantee about the ordering of operation or state of data at any given time.

2. Clients may see different versions of data depending on which node they are connected to

3. This model provides highest availability and scalability but at cost of consistency

4. In weakly consistent system, there are no strict requirements on when or if replicas will converge to a consistent state.

5. It allows significant divergence blw replicas and permits temporary inconsistencies to exist

6. Guarantees only apply after a synchronisation event. Before that, no consistency is guaranteed.

(v) Entry consistency

1. Entry consistency is type of consistency model, specifically for shared memory systems where consistency is enforced at the scale of individual shared variables (or "entries")

2. In this model, a process must explicitly aquire a lock associated with the shared variable before accessing it.

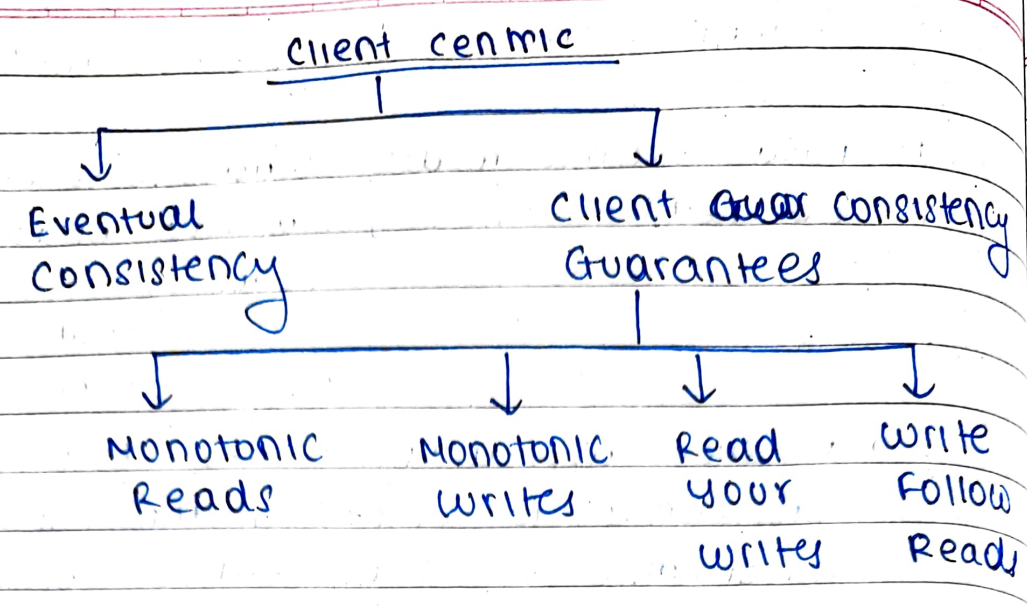
3. This ensures that processes have the most recent, consistent value of variable

Client centric consistency models

1. Data-centric models lead to excessive overheads where
   - majority operations are reads
   - Frequent updates from one client process
2. For such application, weaker form of consistency is employed for improving performance - client centric consistency
3. They relax the ~~go~~ global consistency requirements and prioritize high availability and low latency for clients
4. Client centric consistency models specify two requirements.

(A) Client consistency Guarantees:
- The model should ensure ~~that~~ a specific level of consistency for individual clients while accessing data
- Ensure that client sees a consistent view of the data
- Eg Read your writes, monotonic reads, monotonic writes.

(B) Eventual consistency
- Eventual consistency refers to a system wide guarantee stating that replicas will converge to the same consistent state after the updates propogate through the system assuming no further updates occur.



(I) Monotonic Reads

1. It is a client centric consistency model that ensures that once a client reads a value, subsequent reads by the client will never return an older value.

2. In other words, a client will always observe data that has either stayed the same or been updated to a newer value.

3. The reads are non-decreasing and consistent for a single client. It doesn't require all clients or replicas to be consistent.

(II) Monotonic writes

1. It is a client centric consistency model that guarentees a clients writes are applied in the same order across all replicas.
2. It ensures that writes made by client are not seen out of order by that client across all replicas.
3. It ensures that if client writes a value, subsequent write will not be applied in reverse order.

Eg:
If client writes $x = 5$ and then $x = 10$ then no replica will apply $x = 5$ after $x = 10$.
writes will always be applied in the order client made them.

(III) Read your writes.

1. ~~In this model, distributed~~
2. This model ensures that once a client writes a value, it will immediately be able to read same value or a newer one on subsequent reads.
2. It ensures that client sees his own updates immediately ~~after~~ after they are ~~returned~~ written. Thus providing consistent view for client

(iv) Write follows Read

1. This model ensures that a write operation is only allowed to proceed if the preceeding read has occurred

2. It guarantees that a client's write will always follow a previous read of same data.

3. It means client cannot write the value if unless it has previously read the value

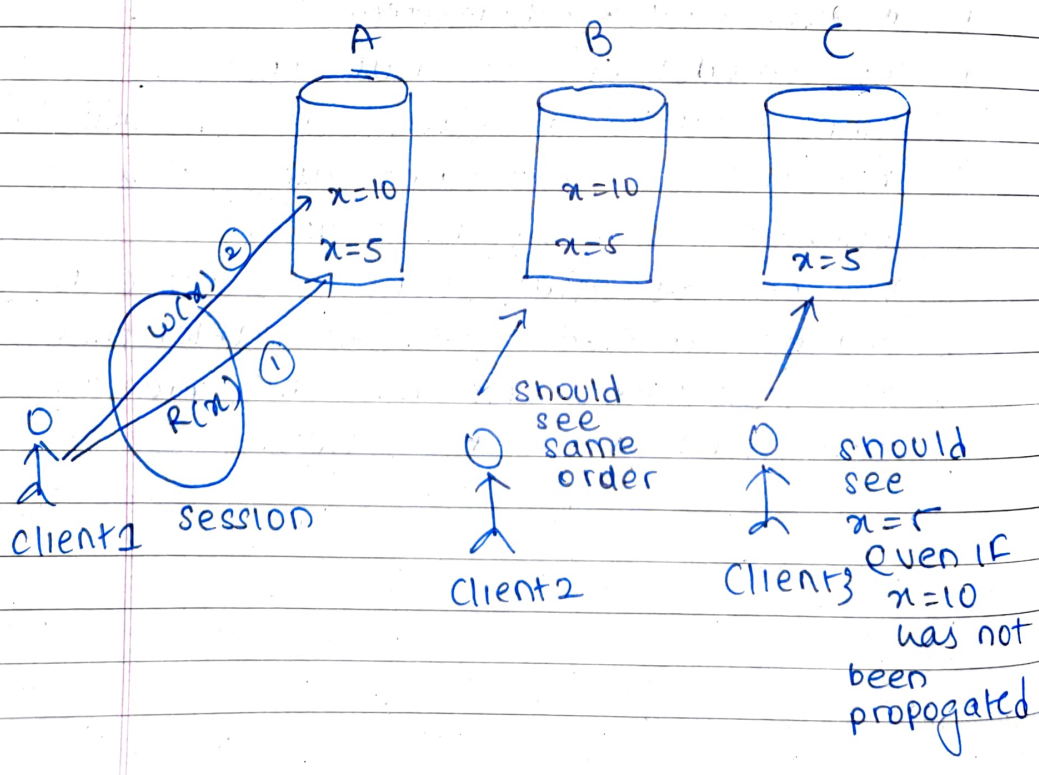
4 Eg:

(i) Suppose client reads $x=5$ from 'A'

(ii) After reading $x=5$, client updates $x=10$

This sequence is maintained

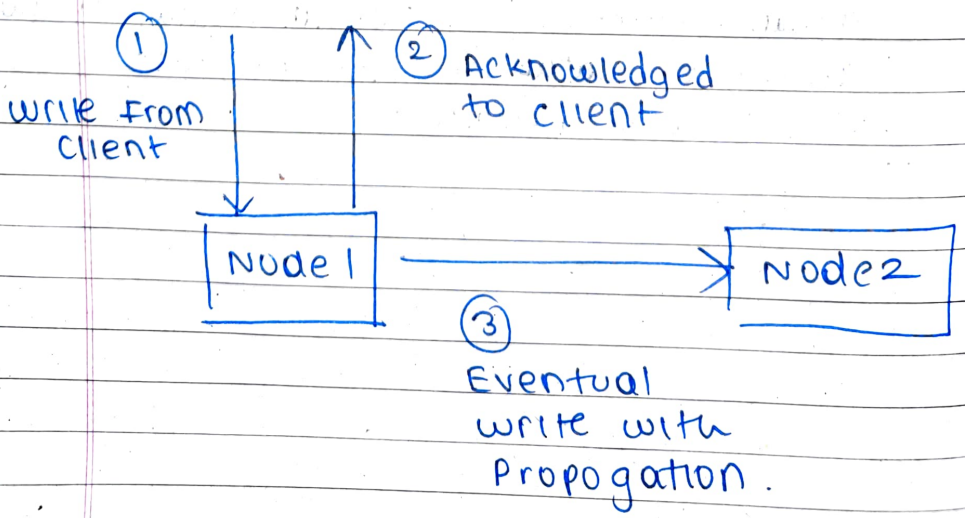
If client tries to write $x=10$ before reading, the system would prevent it



	DATA CENTRIC CONSISTENCY	CLIENT CENTRIC CONSISTENCY
1.	Makes a globally accessible and globally consistent data store	1. Instead of a globally consistent view, maintains consistent views for individual clients.
2.	Difficult to deal with inconsistencies	2. Easier to deal with inconsistencies.
3.	Eg: Linearizability, sequential consistency	3. Eg: Eventual consistency montonic reads, monotonic writes, etc
4.	Ensures all clients see the same data	4. Ensures that clients see their own actions in consistent order
5.	More complex due to coordination b/w different replicas	5. Simpler as it focuses on individual client interactions.
6.	May incur higher latency and reduced availibility for global consistency	6. Offer lower latency and higher availibility for ~~global consistency~~ client experience.
7.	Eg Finance systems, Distributed databases, collaborative application	7. Eg Social media, personalised content, etc.

Eventual consistency model

1. Eventual consistency model is a relaxed form of consistency.
2. It is used in distributed systems where high availability and partition tolerance are prioritized over strict consistency
3. It guarentees that if no new updates are made to a given data item, eventually all replicas will converge to the same final state.
4. The system ensures eventual convergence even in the presence of nlw partitions or delays.
5. Updates are propogated asynchronously
6. If two replicas store conflicting versions of data temporarily, the reconciliation process ensures that all replicas eventually agree on same value using techniques such as versioning, time stamps, conflict resolution strategies.



* Replication Management in DS.

1. Key issue of DS that support replication is to decide where, when and by whom replica should be placed and subsequently which mechanisms to use for keeping replicas consistent.

2. Replication management refers to the process of creating, maintaing and updating replicas across DS

3. RM involves making decision about replicas as well as maintaing consistency of these replicas. Following are important aspects OF RM
   (1) Replica server placement
   (2) content replication and placement
   (3) content distribution.

(I) Replica Server placement.

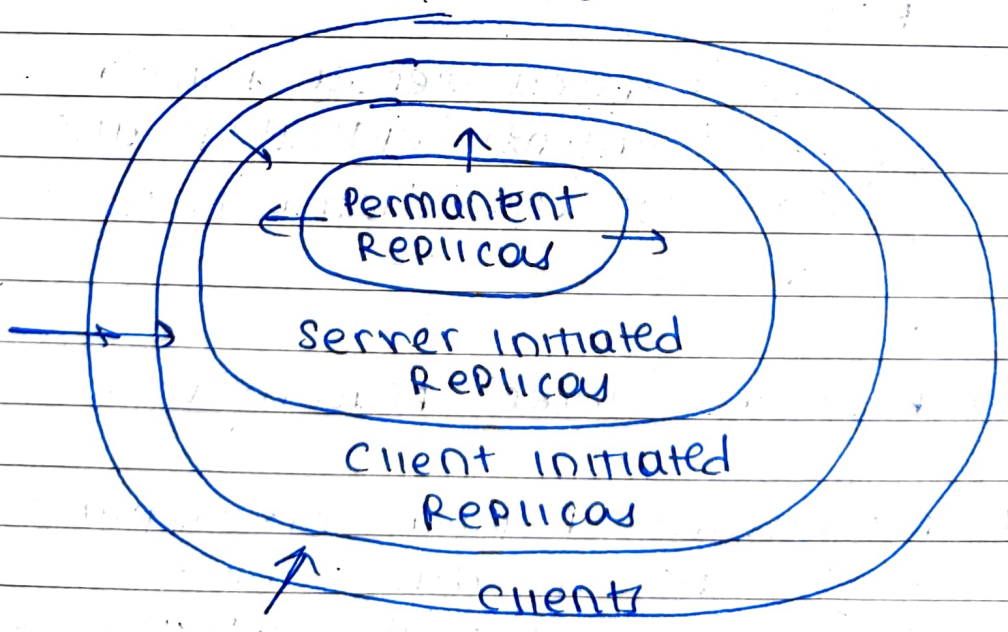
1. Definition: Replica server placement refers to deciding where to place the physical and virtual servers that will host the replicas.
2. The goal is to optimise performance, availibility
3. Performance is optimized by reducing latency — placing replicas closer to clients.
4. Placing replicas in failure independent

zones to ensure system remains operational even if one fails.

Techniques { **static**: predefined locations and fixed. Eg CDNs
dynamic: Replicas are added or migrated on real time access patterns. Eg: AWS, Azure

(II) Content replication and placement.

1. Content replication is the process of replicating content across servers and determining which content should be stored on each replica server.
2. There are three types of replicas that can be distinguished



3. These replicas are example of policies or triggers used to decide when and where replicas are created.

(1) permanent replicas: There are long term copies of data or content stored at designated locations in the system

(2) server initiated replicas: server initiated replicas created by the server based on changing conditions like high demand for specific content, load balancing requirements.

(3) client initiated replicas: These replicas are created in response to specific requests, demands from clients. These replicas often represent the cached copies of content at or near client side to improve speed. Replication is triggered by clients activities rater than servers internal algorithm.

(III) Content Distribution

1- Content distribution refers to mechanisms and protocols used for propogation of updated content to replica servers.

2- Content distribution is implemented with different techniques; categorized by
   (A) State vs Operations
   (B) Push vs Pull protocols
   (C) Unicasting vs Multicasting

(A) State vs Operations

1. The classification focusses on type of information shared b/w replicas during updates.
2. In state transfer, entire state or snapshot is propogated to the replicas. They are simple to implement but is bandwidth intensive.
3. Operation based distribution, only the operations (or changes) ~~are~~ performed are propogated to ~~servers~~ replicas.

(B) Pull vs Push protocol

1. This classification focuses on how updates are distributed.
2. In pull protocol, the replicas request the updates from source periodically or needed
3. In push protocol, the source proactively sends updates to the replicas as soon as changes occur.

Unicasting vs Multicasting

1. This classification focuses on how updates are sent to replicas.
2. In unicasting, updates are sent individually to each replica.
3. In multicasting, updates are sent simultaneously to group of replicas using group communication mechanisms.

* Continuous consistency Model

1. Continuous consistency model is used to measure inconsistencies and express what inconsistencies can be ~~expre~~ expected by the system.

2. It is a framework for measuring and expressing consistency in replicated data stores.

3. Unlike strict consistency which enforces a single, global view upon data, continuous consistency allows a system to balance b/w consistency, availibility and performance.

4. In this model, the level of consistency is defined by
   - (1) Bounding Numerical Deviation
   - (2) Bounding Ordering Deviation
   - (3) Bounding Staleness Deviation.

(1) **Numerical Deviation**
- This defines the allowable difference in values b/w replicas

(2) **Ordering Deviation**
- This defines how much order of applying updates differ across replica

③ Staleness Deviation :

- This refers to the time lag b/w an update being made at one replica and the same update being visible at another replica.

Cache coherence protocols

~~1. Cache is a small sized and~~

1. Cache coherence protocols are mechanisms used in distributed systems or multiprocessor systems to ensure that multiple cached copies of shared data remain consistent.
2. When multiple nodes access the same memory location and cache its value locally, a lack of coherence can lead to inconsistencies
3. Cache coherence protocols ensure that caches are consistent regardless of node.
4. Cache coherence protocols rely on consistency models like sequential consistency or weak consistency models to enforce shared memory behaviour.
5. Basic Approaches:
   - (A) Write Invalidate: Invalidate all caches holding a data when write occurs
   - (B) Write update: Broadcast updates to all caches so they can update their copies.
6. Types of cache coherence protocols
   - (I) Directory-based Protocol:
     - Use a centralised or distributed directory to maintain state of each memory block.
     - The directory tracks which ~~type of~~ caches have copies of data

and whether the data is modified.
- If the data is modified, node fetches the updated copy

(II) Snooping Protocols

- All nodes have/share a common bus and caches monitors (or snoops) bus transaction to maintain coherence
- Used in small scale systems ~~where~~ with shared buses.