Unit 5 – Failure Management

Network Information System Technologies

Index

1. Introduction
2. Replication
3. Consistency
4. Learning Results

Goals

- Proper characterization of failure scenarios in a distributed system.
- Learning of replication models.
- Identification of consistency models.

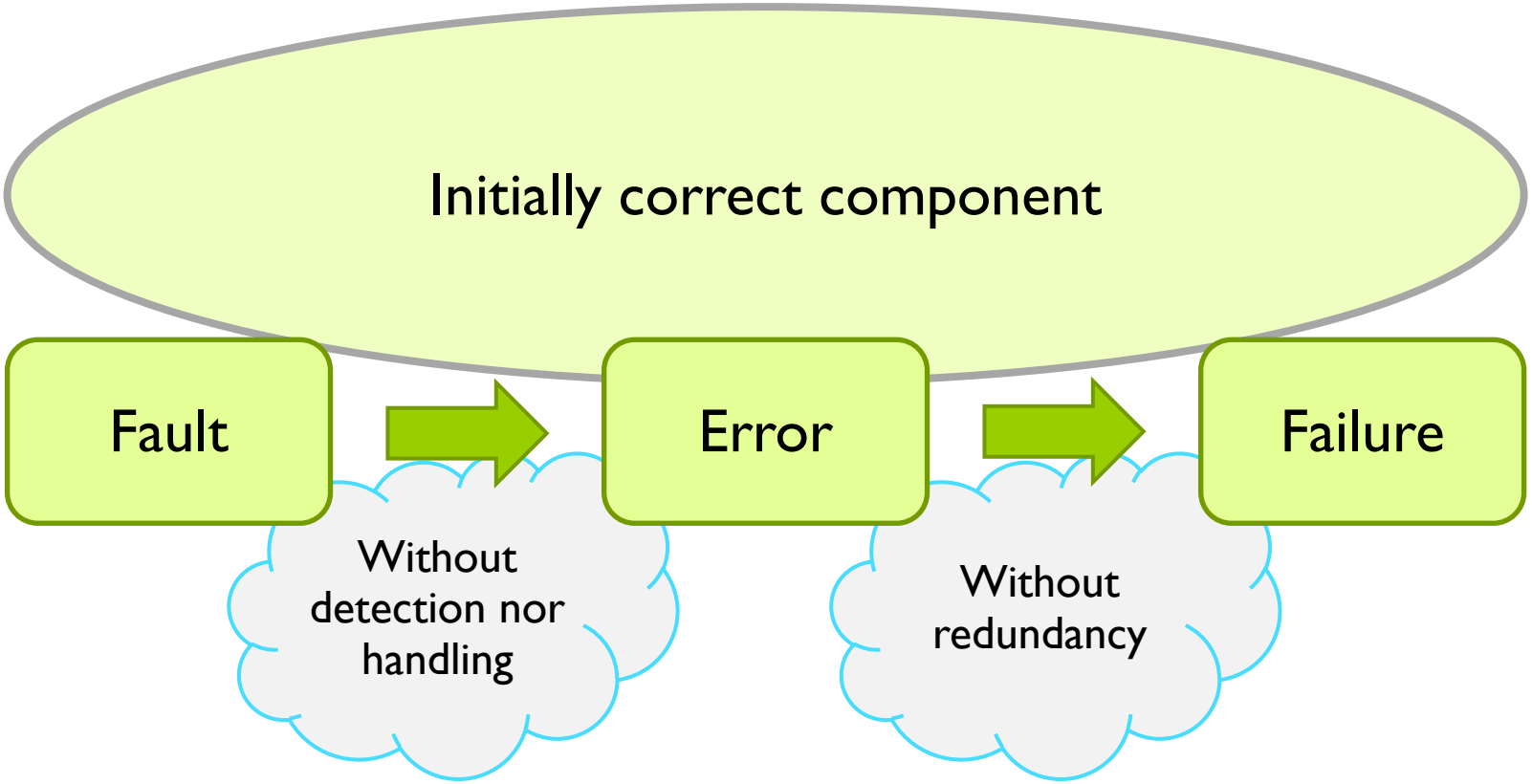
Index

1. Introduction
2. Replication
3. Consistency
4. Learning Results

1. Introduction

- Failure. Informal definition
  - There is a failure when any system component is unable to behave according to its specification
- Precise definition
  - Three different concepts should be identified:
    - **Fault**: Anomalous physical condition
      - Examples: Design errors, electromagnetic interferences, unexpected inputs, system misuse...
    - **Error**: Manifestation of a fault in a system
      - The logical state of an element differs from its intended value
    - **Failure**: A failure occurs when...
      - an element is unable to perform its designed function...
        - due to **errors** in the element or its environment,
        - which are caused by various **faults**

1. Introduction



1. Introduction

- A distributed system should provide failure transparency
  - Errors in components, when they happen, should not be perceived by users
    - Solution: Replication
      - A faulty replica should be diagnosed and separated
        - Once repaired, it will rejoin the system
      - Other replicas hide all these steps
  - This transparency is also known as "*fault tolerance*"
    - A **system** is fault-tolerant when it behaves correctly in case of faults
    - In order to be fault-tolerant, a **service** needs:
      - To be carefully designed (considering all possible use cases)
      - That all external services it depends on be also fault-tolerant

1. Introduction

- Failures may have multiple causes
  - They depend on faults and on which elements are affected by those faults
  - Multiple kinds of failures can be distinguished
- A given failure model should be assumed when a distributed algorithm is written
  - Models cannot include all failure scenarios
  - Higher level of abstraction
    - Concrete details are not considered
  - Middleware should translate the physical scenario into a logical one that matches what is considered in the assumed failure model

1. Introduction

- Network failures:
  - Connectivity should be considered
    - **Network partitioning** is a problem
  - *Partition*: When a group of nodes remains isolated from the other parts of the system
  - Options:
    1. Partitionable system
       - Each isolated subgroup is able to go on
       - Some kind of reconciliation protocol is executed when connectivity is recovered
    2. Primary partition model
       - Only the group with a majority of system nodes is able to go on
       - Sometimes, there is no such group

Index

1. Introduction
2. Replication
3. Consistency
4. Learning Results

2. Replication

- Replication is a basic mechanism in order to ensure component availability
  - Each component replica is placed in a different computer
  - Those computers do not depend on the same failure sources
    - Electricity grid, uninterruptable power supply, computer network,...
  - When a replica fails the others will not fail
    - On-going operations in the crashed replica may be resumed in any other replica
  - Replication makes easy the recovery of failed replicas
    - Correct replicas are the source of a state transfer once a crashed replica (is repaired and) rejoins the system

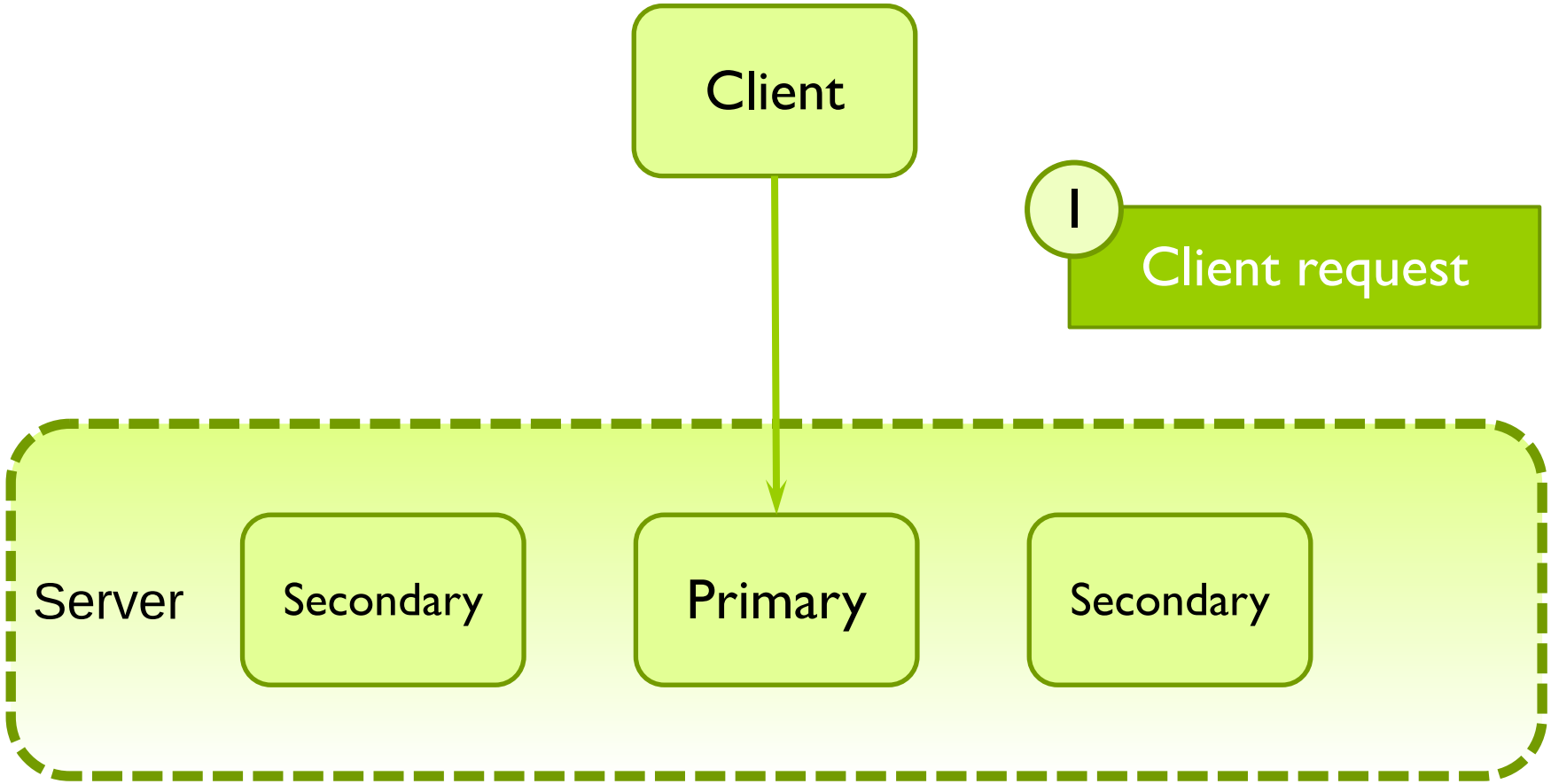
2. Replication

- Replication also improves service performance:
  - Read-only operations may be served by any single replica
    - Linear throughput that depends on the number of replicas
    - Excellent scalability
  - Update operations need to be applied in all replicas
    - This introduces non-negligible delays
    - Short operations may be executed in all replicas
      - Requests should be propagated to all replicas
    - Long operations that update a few data elements may be...
      - ...executed in a single replica
        - Propagating later the updated elements to all other replicas
    - Operations may be executed (or update-propagated) in different order in different replicas
      - Result: state divergence among replicas
      - Such degree of allowed divergence determines a consistency model

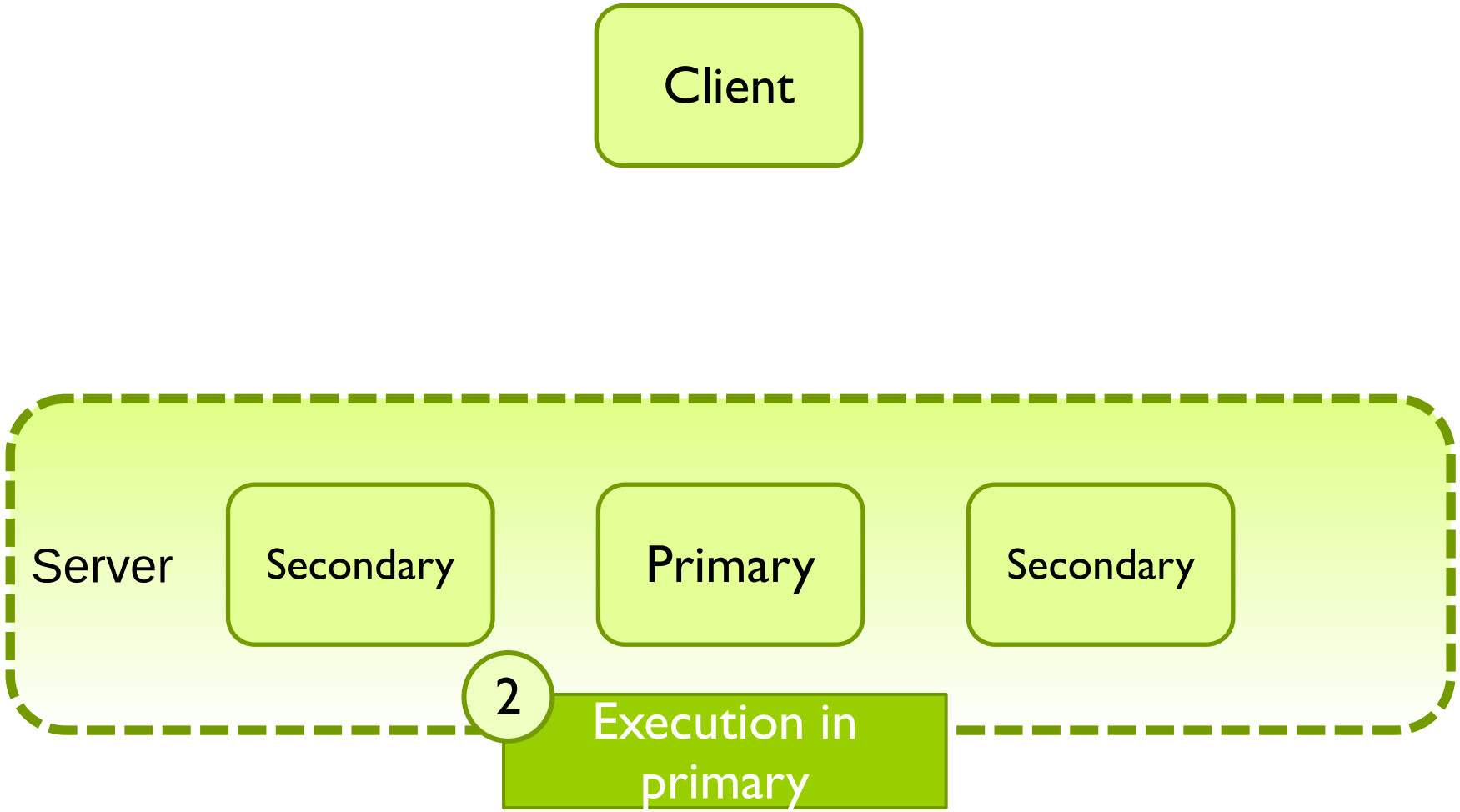
2. Replication

- Two classical replication models:
  1. Passive (or "primary/backup replication")
     - Clients send their requests to a single primary replica
     - Such replica executes all the operations
     - Once an operation is concluded:
       1. Its updates are propagated to the secondary (or back-up) replicas
       2. It answers the client
  2. Active (or "state-machine replication")
     - Each client request is propagated to all server replicas
     - Every server replica executes such operation
     - When a replica concludes the operation, it replies to the client.

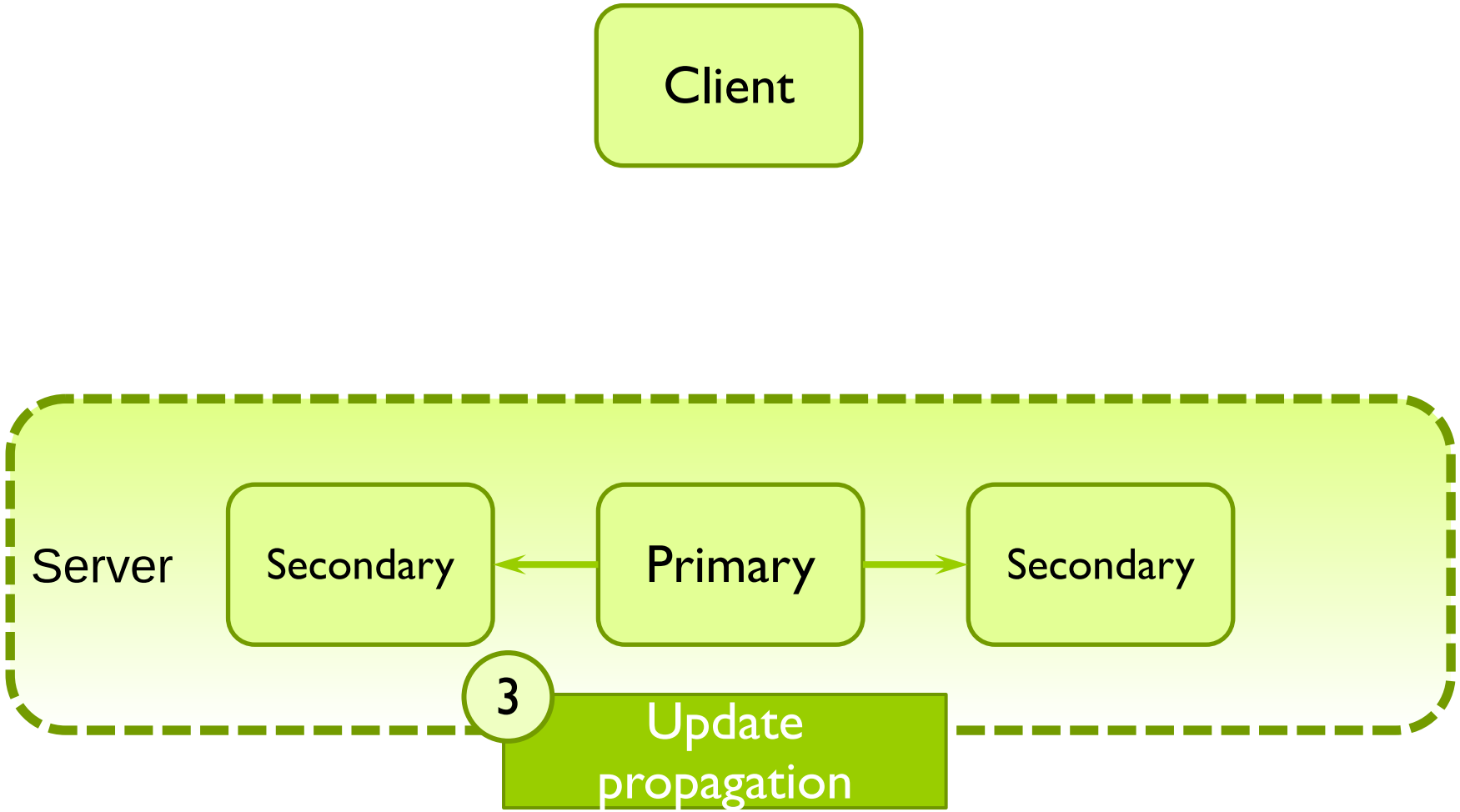
2.1. Passive Replication



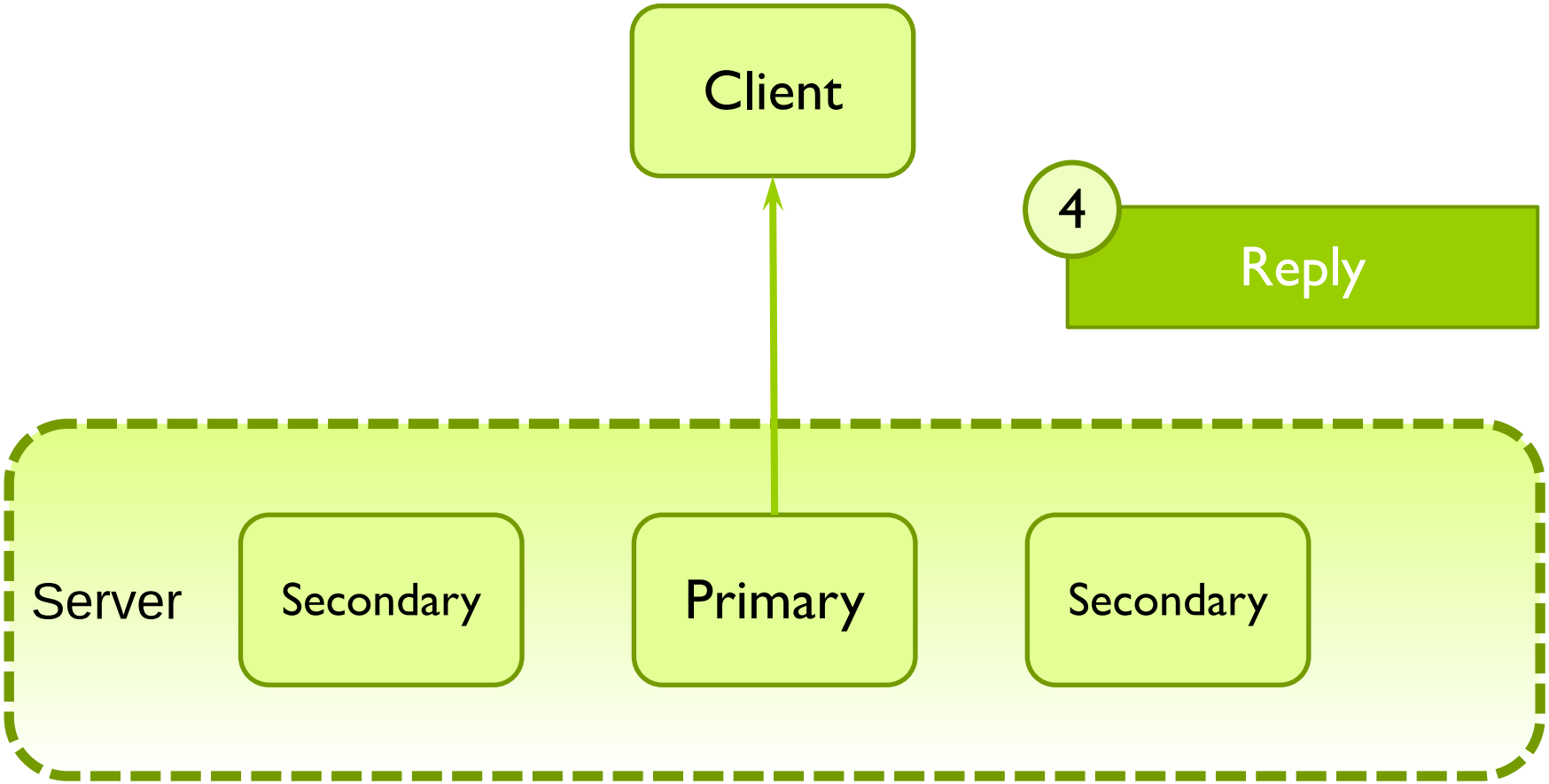
2.1. Passive Replication



2.1. Passive Replication



2.1. Passive Replication



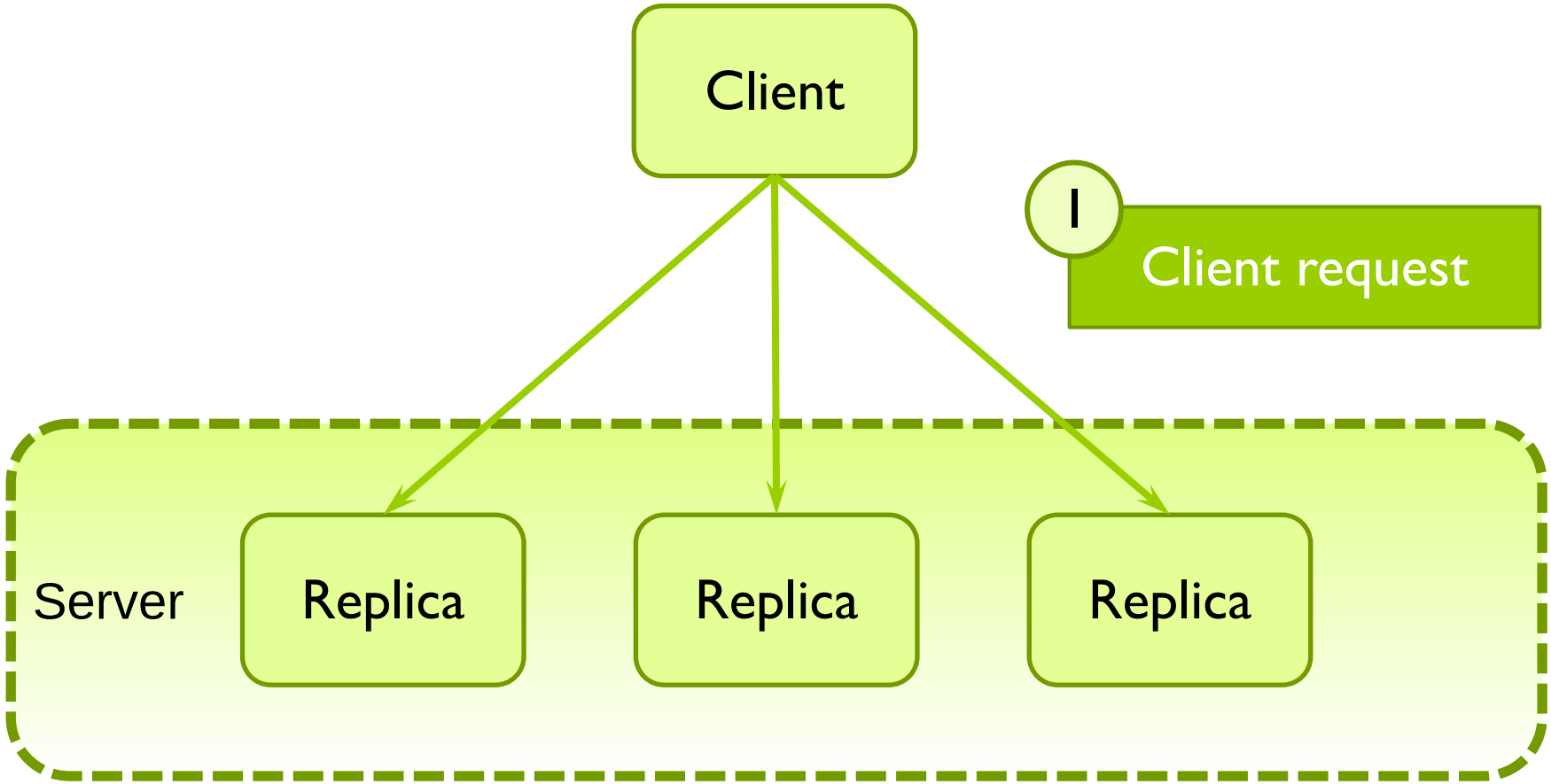
2.1. Passive Replication

- Advantages
  - **Minimum overhead**
    - Each request is served by a single replica
    - Read-only requests might be served by any secondary replica
      - Efficient load balancing
  - Updates can be easily **ordered**
    - The primary replica tags the updates with a sequence number before broadcasting them to secondary replicas.
  - **Local concurrency** control
    - No distributed algorithm is needed
  - Admits **non-deterministic operations**
    - They are only executed by the primary
    - Inconsistencies among replicas never arise

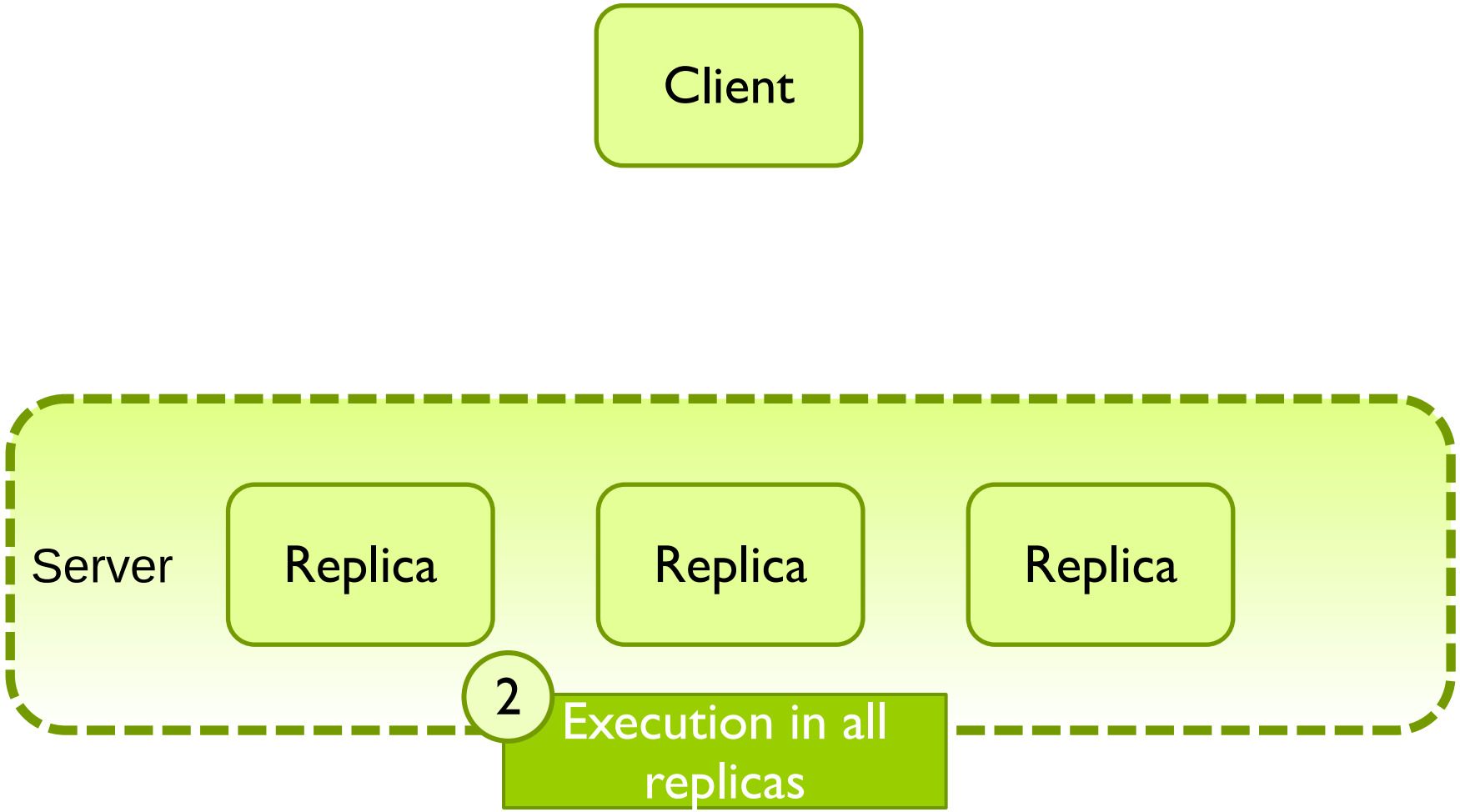
2.1. Passive Replication

- Inconveniences
  - **Difficult reconfiguration** when primary fails...
    - A secondary must be chosen and promoted to primary
    - Ongoing requests might be lost
  - **Arbitrary failures** can**not** be **managed**

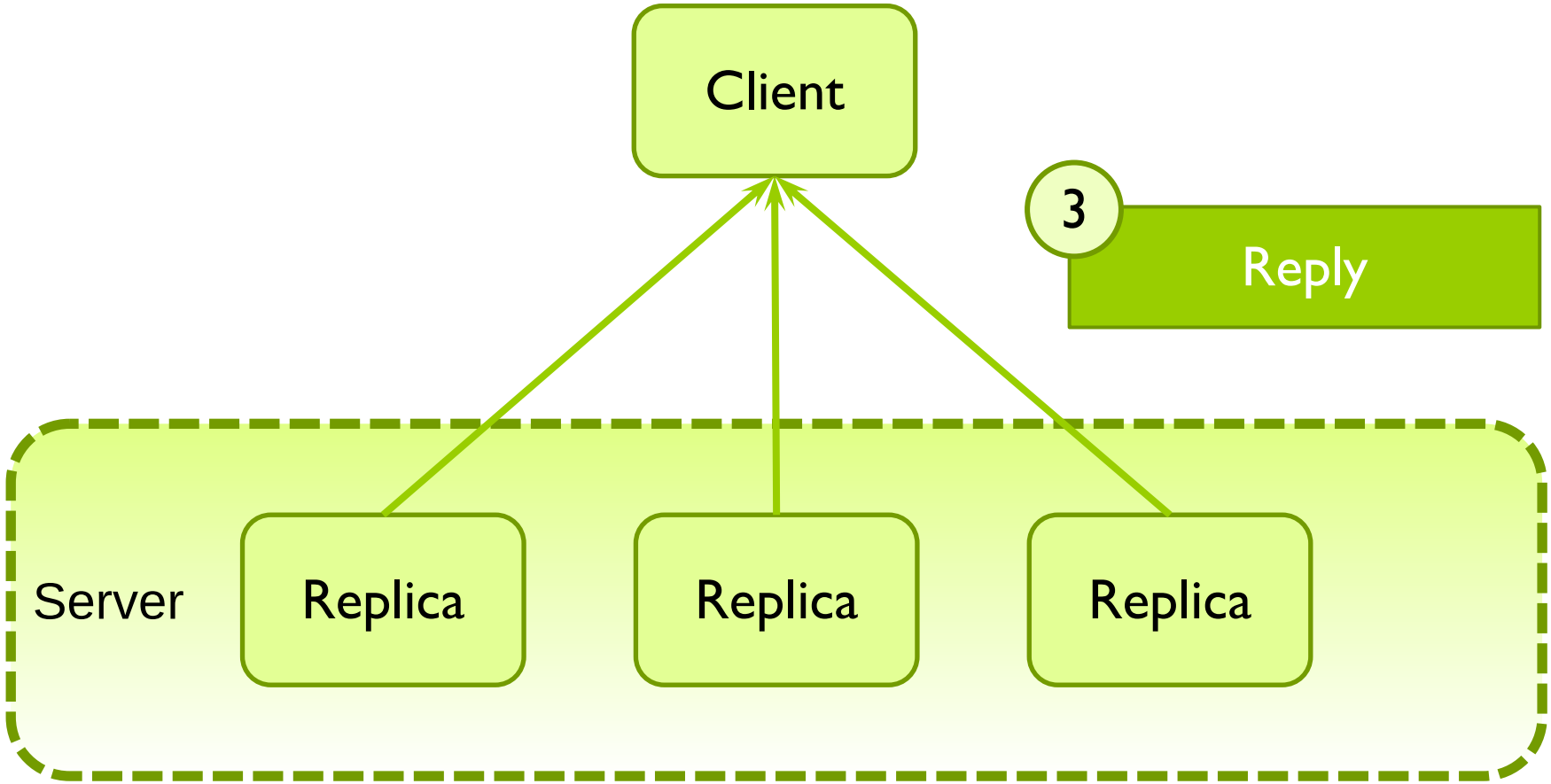
2.2. Active Replication



2.2. Active Replication



2.2. Active Replication



2.2. Active Replication

- Advantages
  - **Trivial recovery** in case of failure
    - Nothing needs to be done
      - Assuming that at least a replica survives ;-)
  - **Allows arbitrary failures**
    - Assuming a maximum of "f" simultaneous failures
    - At least 2f+1 replicas needed
    - Client waits for at least f+1 identical (i.e., correct) answers

2.2. Active Replication

- Inconveniences
  - When a **strong consistency** is needed, requests should be delivered in **total order**.
    - This needs consensus
    - Expensive (multi-round) broadcast protocols
  - Non-**deterministic operations** are difficult to manage
    - Each replica might obtain a different result
    - This leads to inconsistencies!!!
  - When several actively replicated servers interact, **requests** need to **be filtered**
    - Example: Service A calls service B
      - B should filter the many identical requests received
        - B should not execute N times that operation
      - Replies should be propagated to all A replicas

2.3.1. Comparison. General aspects

Aspect	Passive model	Active model
Request processing replicas	**1**	All
Avoids distributed request ordering among replicas	**Yes**	No
Avoids update propagation	No	**Yes**
Admits indeterminism	**Yes**	No
Tolerates arbitrary failures	No	**Yes** (However, to this end, even read only operations should run in all replicas)
Consistency	At least, sequential	At least, sequential
Recovery in case of failure	It needs election and reconfiguration stages in case of primary failure	**Immediate**

2.3.2. Comparison. Case 1. Scenario 1

- Let us assume the deployment of a service S1 with these characteristics:
  - 5 replicas
  - Computer network bandwidth: 1 Gbps
  - Inter-node propagation latency: 2 ms
  - Average processing time per operation: 200 ms
    - No concurrency is assumed
    - If N requests are processed concurrently by the same process, then the resulting processing time is 200*N
  - Average update size per operation: 10 KB
    - i.e., 80 Kb → 0.08 Mb → 0.00008 Gb
      - Short update transference delay among replicas
        - propagation + transmission (bandwidth delay)
        - 0.002 + 0.00008 sec → 2.08 ms
  - Update application time in backup replicas: 3 ms
  - Which replication model provides the best performance?
    - Let us compute the average request management time...

2.3.2. Comparison. Case 1. Scenario 1

Aspect	Passive Model		Active model
	Primary replica	Backup replica	
Request delivery	2 ms	--	4 - 6 ms (assuming a sequencer-based total order multicast algorithm)
Request processing	200 ms	--	200 ms
Update transfer	2.08 ms		0 ms
Update application	--	3 ms	0 ms
Update acknowledgement	2 ms		0 ms
Reply to client	2 ms	--	2 ms
TOTAL:	211.08 ms		206 – 208 ms

2.3.2. Comparison. Case 1. Scenario 2

- Discussion:
  - At a glance, the results seem to be similar in both models
  - However, let us consider that...
    - Each replica has been deployed in a different computer
    - We want to deploy 5 services like S1 in those 5 computers:
      - With passive replication, each service deploys its primary replica in a different computer
      - In this scenario, the obtained results are shown in the next slide
        - The passive model provides the best results!!

2.3.2. Comparison. Case 1. Scenario 2

Aspect	Passive Model		Active model
	Primary replica	Backup replica	
Request delivery	2 ms	--	4 - 6 ms
Request processing	200 + 4*3 = 212 ms	--	200*5 = 1000 ms
Update transfer	2.08 ms		0 ms
Update application	--	3 ms	0 ms
Update acknowledgement	2 ms		0 ms
Reply to client	2 ms	--	2 ms
TOTAL:	223.08 ms		1006 – 1008 ms

2.3.3. Comparison. Case 2. Scenario 2

- Let us now assume the deployment of a set of 5 services S2 with these characteristics:
  - 5 replicas
  - Computer network bandwidth: 1 Gbps
  - Inter-node propagation latency: 2 ms
  - Average processing time per operation: 5 ms
    - No concurrency is assumed
    - If N requests are processed concurrently by the same process, then the resulting processing time is 5*N
  - Average update size per operation: 10 MB
    - i.e., 80 Mb → 0.08 Gb
      - Significant update transfer delay among replicas
        - propagation + transmission (bandwidth delay)
        - 0.002 + 0.08 sec → 82 ms
  - Update application time in backup replicas: 3 ms
  - Which replication model provides the best performance?
    - Let us compute the average request management time...

2.3.3. Comparison. Case 2. Scenario 2

Aspect	Passive Model		Active model
	Primary replica	Backup replica	
Request delivery	2 ms	--	4 - 6 ms
Request processing	5 + 4*3 = 17 ms	--	5*5 = 25 ms
Update transfer	82 ms		0 ms
Update application	--	3 ms	0 ms
Update acknowledgement	2 ms		0 ms
Reply to client	2 ms	--	2 ms
TOTAL:	108 ms		31 – 33 ms

2.3.4. Comparison. Analysis

- The replication model to be chosen depends heavily on:
  - The average request processing time
    - Passive model appropriate for long processing times
      - Since that processing is only done at a single replica
    - Active model appropriate for short processing times
  - The updates size
    - Passive model appropriate for small updates
    - Active model appropriate for big updates
      - Since it avoids update transfer

Index

1. Failures: Concept and Types
2. Replication
3. Consistency
4. Learning Results

3. Consistency

- Consistency
  - When data is replicated in multiple nodes, a consistency model specifies what divergences are allowed among the values observed of a given data element
  - Processes write in a single node
  - Such node propagates later the result to the other replicas
    - Consistency depends on the propagation delay
    - Also depends on the behavior of readers (would they block?) while that propagation is being done
  - When the consistency algorithm allows that both reads and writes return control without needing any message transmission…
    - We'll have a "fast" consistency model.
    - Otherwise, the consistency model is slow.
  - Each process reads in a single node: local reads.

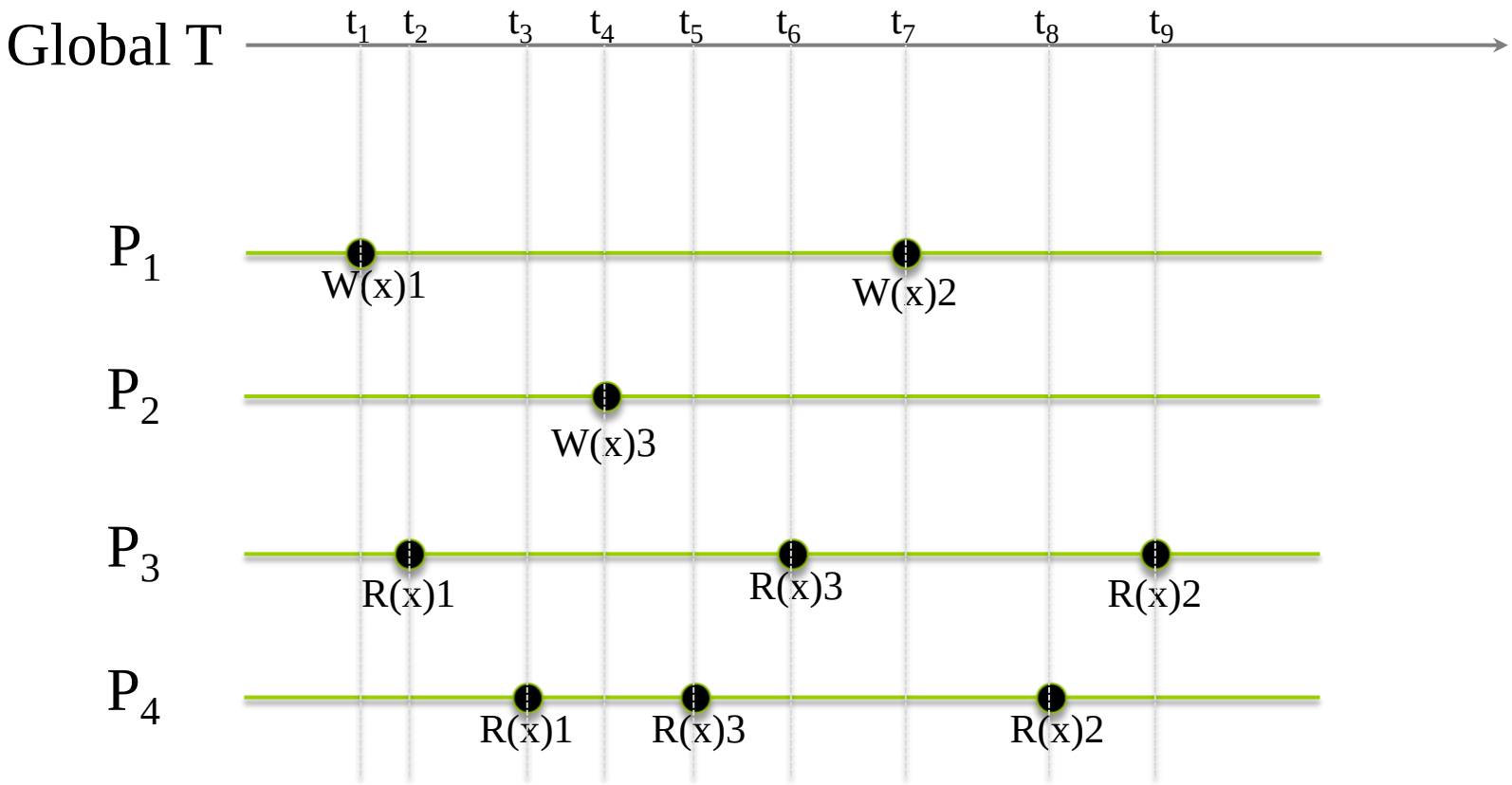
3. Consistency

- Consistency models
  - When no synchronization tools are assumed, the main models are:
    - Strict (slow)
    - Sequential (slow)
    - Processor (slow)
    - Cache (slow)
    - Causal (fast)
    - FIFO/PRAM (fast)
  - We'll only study some of these models
- Eventual consistency
  - In scalable systems, replicas converge when there are long time intervals without any update operation
  - When there are pending update operations, each replica serves them without worrying about what other replicas are doing
  - Later on, processes decide how to interleave those updates and which final value should be adopted
    - Trivial if operations are commutative

3.1. Strict consistency

- Intuitive notion on how strongly consistent memory should work
- Assumes a global clock that timestamps each event
  - Two writes cannot happen simultaneously
  - Writes are propagated immediately
  - Each read event on variable *x* returns the value of the closest earlier (by the timestamp order) write event on variable *x*
- Difficult implementation!!
  - It implies a strong coordination among all replicas
  - Not scalable

3.1. Strict consistency: example

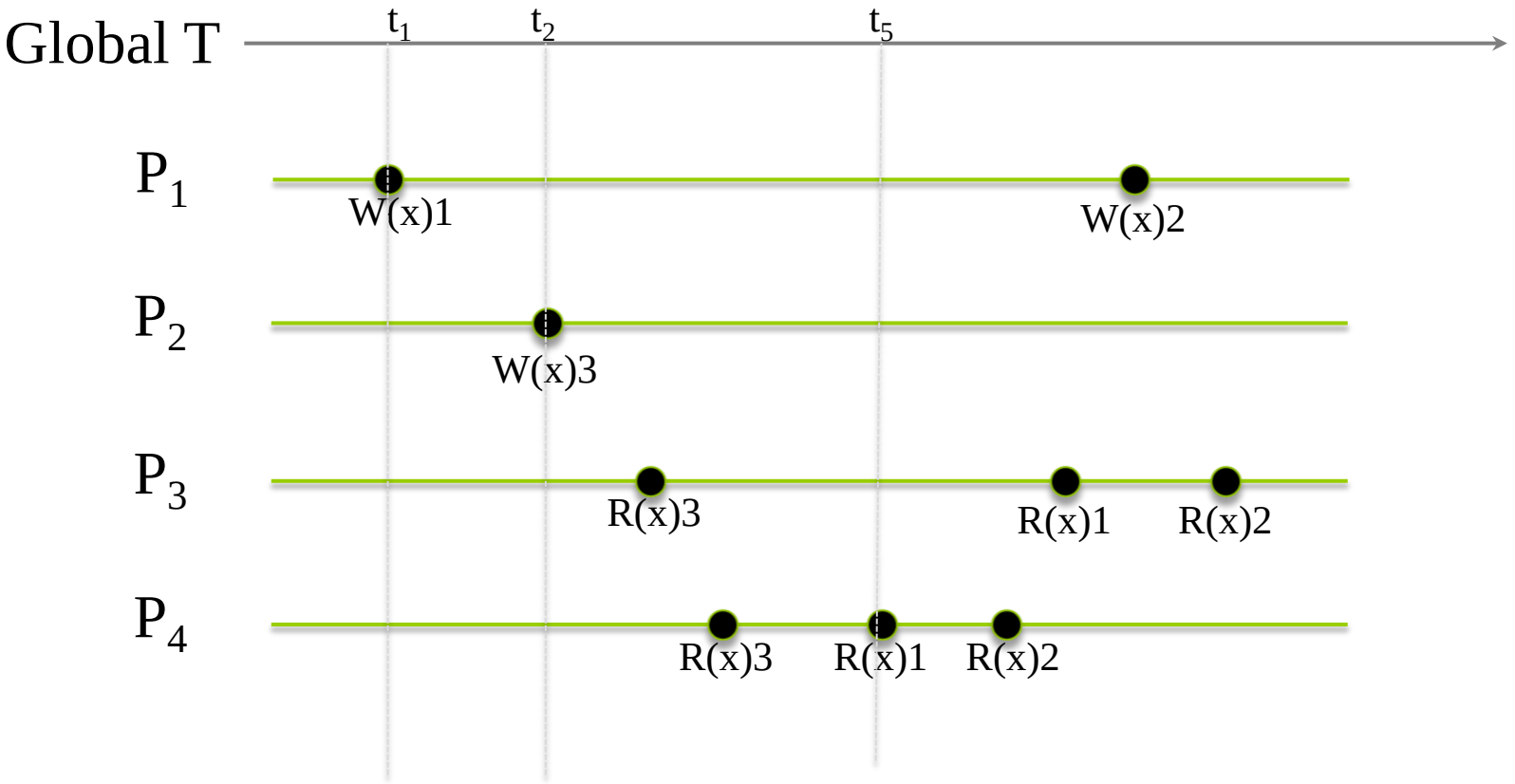


- Problem
  - Does not make sense to talk about global time in a DS

3.2. Sequential Consistency

- Informal definition:
  - "The result of an execution is the same as if the operations of all processes were executed in a sequential order, and the operations of each processor appear in this sequence in the order specified by its program"
    - All processes reach a consensus on the order of all writes
      - This order matches all process-writing orders
    - But each process advances at its own pace

3.2. Sequential Consistency: example

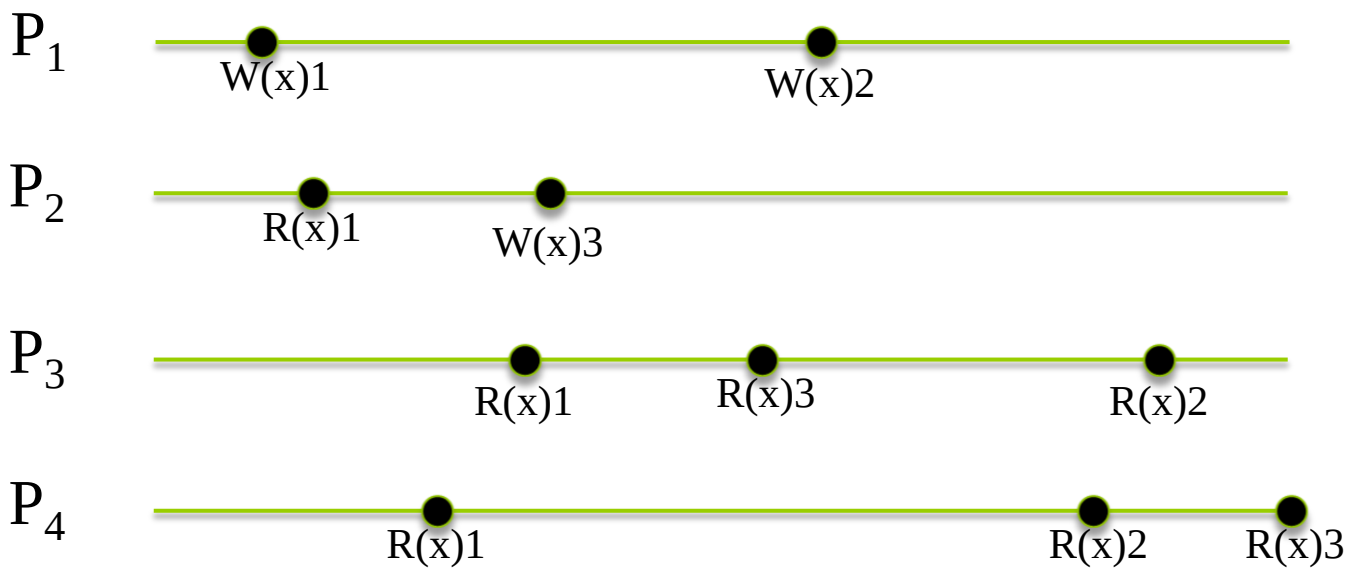


- Evidently, this execution does not satisfy the requirements of strict consistency assuming the global timeline
  - $t_1 < t_2$ but at t_5 we get R(x)1 in process P4, instead of R(x)3, as would be expected, contradicting the conditions of strict consistency.

3.3. Causal Consistency

- Complies with the "*happens before*" relation defined by Lamport
  - Relation used in logical clocks
  - Causal consistency assumes the mapping we introduced earlier
    - W(x)a → R(x)a

3.3. Causal Consistency: example

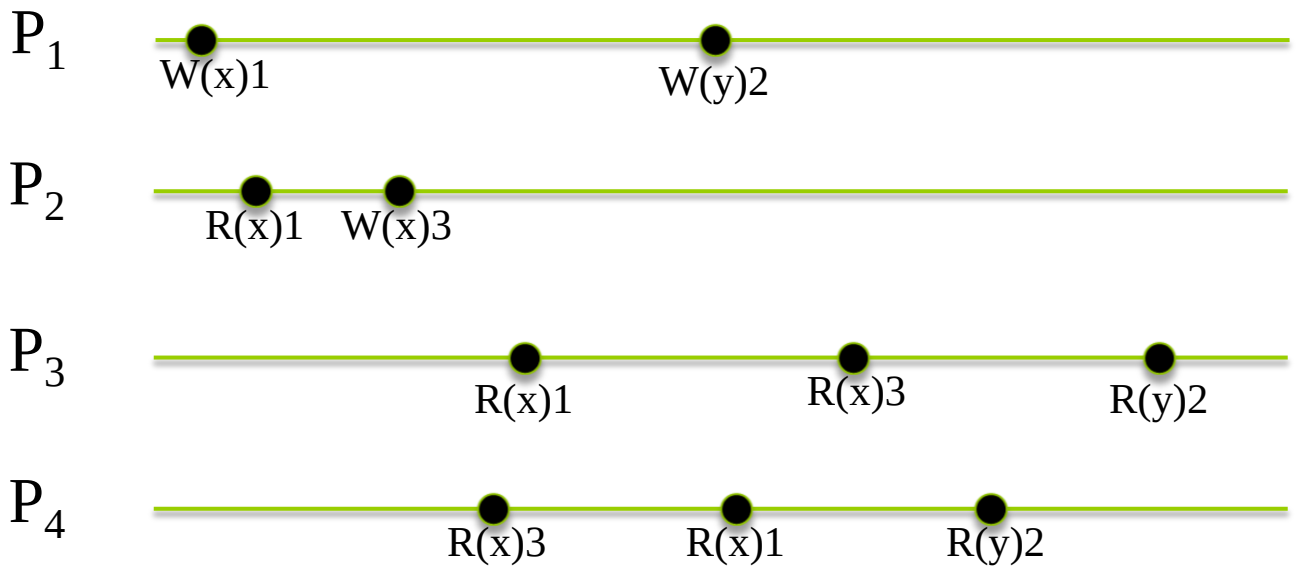


- W(x)1 causes / happens before W(x)2
- W(x)1 causes / happens before W(x)3
- W(x)2 and W(x)3 are concurrent.
  - Processes P3 and P4 may order them arbitrarily.

3.4. FIFO / PRAM Consistency

- Requires that all writes made by each process are read in writing order by all other processes
- But there is no constraint for interleaving what has been written by different processes

3.4. FIFO / PRAM Consistency

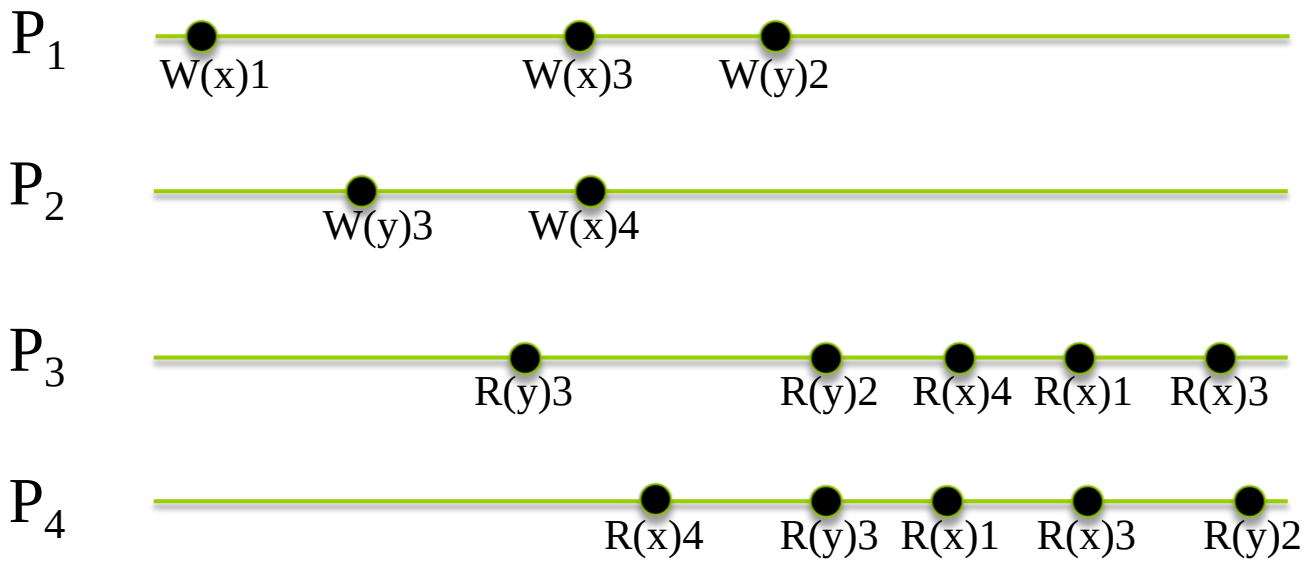


- In this example, a single constraint exists: all readers should get value 1 before value 2
  - Since both were written by P1
  - But value 3 may be read at any point in each local read sequence

3.5. Cache Consistency

- This model requires that writes onto each variable are seen in the same order by all processes.
  - Additionally, when a given process writes multiple times onto a variable, these writes are seen in such write order by all processes.
- But there is no constraint in order to interleave reads made on different variables.

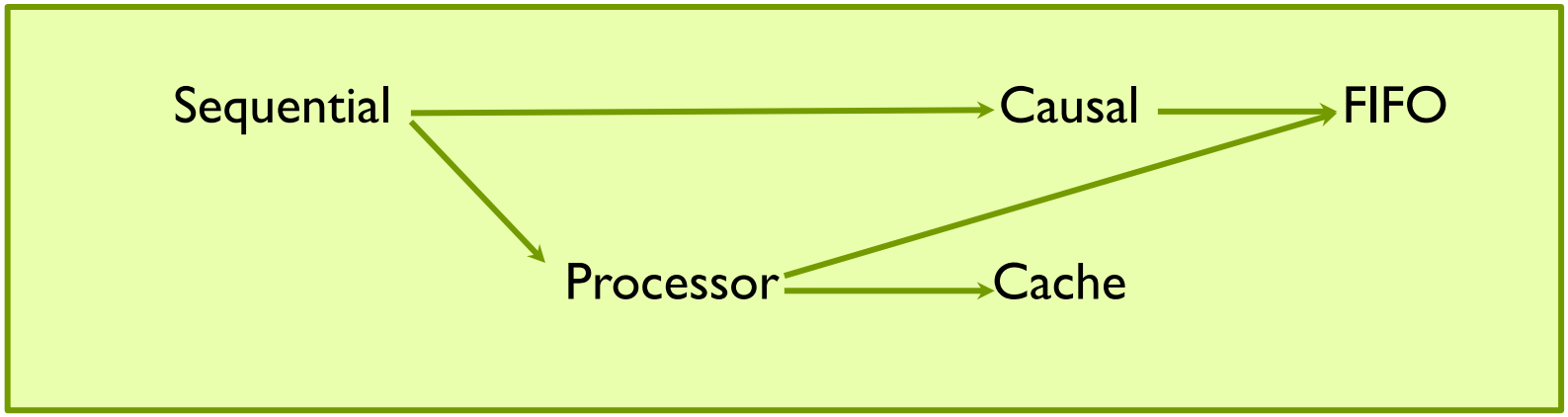
3.5. Cache Consistency



- Processes agree on:
  - Read order for "x" values should be 4, 1, 3.
    - Value 1 is obtained before 3 since both were written by P1.
  - Read order for "y" values should be 3, 2.
- How reads on "x" and "y" are interleaved is irrelevant.
- We obtain **"processor" consistency** when both cache and FIFO consistencies are respected.

3.6. Consistency

- Model hierarchy



- Arrows mean that the source model is stricter than the target model
  - If the source model is respected, the target model is also respected
- Some models cannot be compared
  - Causal and processor
  - FIFO and cache
  - Causal and cache

Index

1. Failures: Concept and Types
2. Replication
3. Consistency
4. Learning Results

4. Learning Results

- At the end of this unit, the student would be able to:
  - Distinguish between faults, errors and failures
  - Identify replication as the approach needed for overcoming failures and improving system scalability
  - Know classical replication models and choose the most adequate one for each kind of component in scalable applications
  - Know different consistency models and forecast their effect on system scalability