Modulo: Finding Convergence Failure Bugs in Distributed Systems with Divergence Resync Models

Beom Heyn Kim[§†], Taesoo Kim[§‡], and David Lie[†]
[§]Samsung Research, [†]University of Toronto, [‡]Georgia Institute of Technology
{beomheyn.kim, tsgates.kim}@samsung.com, lie@eecg.toronto.edu

Samsung Research

UNIVERSITY OF TORONTO

Georgia Tech

Background: Various Services Rely on Replicated Distributed Storage Systems

1	OneDrive	6	Cisco WebEx	11	Concur	16	Lithium
2	Exchange Online	7	Skype for Business	12	Workday	17	Slack
3	Salesforce	8	Box	13	ServiceNow	18	LastPass
4	SharePoint Online	9	Zendesk	14	SuccessFactors	19	ADP
5	Yammer	10	Oracle Taleo	15	Cornerstone OnDemand	20	DocuSign

Replicated Distributed Storage Systems are Critical Components

APACHE ZooKeeper™

mongoDB®

redis

cassandra

Couchbase

Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

ZooKeeper / ZOOKEEPER-1319

Missing data after restarting+expanding a cluster

Details

Type:	Bug	Status:	CLOSED
Priority:	Blocker	Resolution:	Fixed
Affects Version/s:	3.4.0	Fix Version/s:	3.4.1, 3.5.0

Description

The scenario I see is this:

1) Start up a 1-server ZK cluster (the server has ZK ID 0).
2) A client connects to the server, and makes a bunch of znodes, in particular a znode called "/membership".
3) Shut down the cluster.
4) Bring up a 2-server ZK cluster, including the original server 0 with its existing data, and a new server with ZK ID 1.
5) Node 0 has the highest zxid and is elected leader.
6) A client connecting to server 1 tries to "get /membership" and gets back a -101 error code (no such znode).
7) The same client then tries to "create /membership" and gets back a -110 error code (znode already exists).
8) Clients connecting to server 0 can successfully "get /membership".

Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

ZooKeeper / ZOOKEEPER-1319

Missing data after restarting+expanding a cluster

Details

Type:	Bug	Status:	CLOSED
Priority:	Blocker	Resolution:	Fixed
Affects Version/s:	3.4.0	Fix Version/s:	3.4.1, 3.5.0

Description

The scenario I see is this:

1) Start up a 1-server ZK cluster (the server has ZK ID 0).
2) A client connects to the server, and makes a bunch of znodes, in particular a znode called "/membership".
3) Shut down the cluster.
4) Bring up a 2-server ZK cluster, including the original server 0 with its existing data, and a new server with ZK ID 1.
5) Node 0 has the highest zxid and is elected leader.
6) A client connecting to server 1 tries to "get /membership" and gets back a -101 error code (no such znode).
7) The same client then tries to "create /membership" and gets back a -110 error code (znode already exists).
8) Clients connecting to server 0 can successfully "get /membership".

Reproduction Step:

1. Start Server 0
2. Create data items on Server 0
3. Shutdown Server 0
4. Start Server 0 and Server 1
5. Clients cannot read data items on Server 1
6. Clients can read data items on Server 0

Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

ZooKeeper / ZOOKEEPER-1319

Missing data after restarting+expanding a cluster

Details

Type:	Bug	Status:	CLOSED
Priority:	Blocker	Resolution:	Fixed

Reproduction Step:

1. Start Server 0
2. Create data items on Server 0

complains those already exist

Servers are out-of-sync!
Clients may read inconsistent data and make incorrect decisions!

with ZK ID 1.
5) Node 0 has the highest zxid and is elected leader.
6) A client connecting to server 1 tries to "get /membership" and gets back a -101 error code (no such znode).
7) The same client then tries to "create /membership" and gets back a -110 error code (znode already exists).
8) Clients connecting to server 0 can successfully "get /membership".

Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

ZooKeeper / ZOOKEEPER-1319

Missing data after restarting+expanding a cluster

Activity

All | Comments | Work Log | History | Activity | Transitions

Patrick D. Hunt added a comment - 07/Dec/11 01:29

I see the problem, a change made in ~~ZOOKEEPER-1136~~ is causing this regression. setting of lastProposed in the lead() method of Leader was commented out for some reason. As a result the new follower is getting an empty diff rather than a snapshot.

This is a serious issue as it's causing the follower to get an inconsistent view of the database. We'll need to roll 3.4.1 asap.

Jeremy – Thanks for reporting this issue!

Problem: Bugs are Difficult to Find and Fix

ZooKeeper / ZOOKEEPER-1549

Data inconsistency when follower is receiving a DIFF with a dirty snapshot

Description

Initial Condition
1. Lets say there are three nodes in the
2. The current epoch is 7.
3. For simplicity of the example, lets say
4. The zxid is 73
5. All the nodes have seen the change
Step 1
Request with zxid 74 is issued. The lead
and B,C never write the change 74 to th
Step 2
A,B restart, A is elected as the new lead
it), then send diff to B, but B died befor
Step 3
B,C restart, A is still down
B,C form the quorum
B is the new leader. Lets say B minCom
epoch is now 8, zxid is 80
Request with zxid 81 is successful. On B, minCommitLog is now 71, maxCommitLog is 81
Step 4
A starts up. It applies the change in request with zxid 74 to its in-memory data tree
A contacts B to registerAsFollower and provides 74 as its ZxId
Since 71<=74<=81, B decides to send A the diff.
Problem:
The problem with the above sequence is that after truncate the log, A will load the snapshot again which is not correct.

Long Sequence of Events are Required to be Interleaved.
→ Difficult Bug to Find!

Problem: Bugs are Difficult to Find and Fix

ZooKeeper / ZOOKEEPER-1549

Data inconsistency when follower is receiving a DIFF with a dirty snapshot

Details

Type:	Bug	Status:	OPEN
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	3.4.3	Fix Version/s:	None

Dates

Created:	10/Sep/12 07:58
Updated:	03/Feb/22 08:50

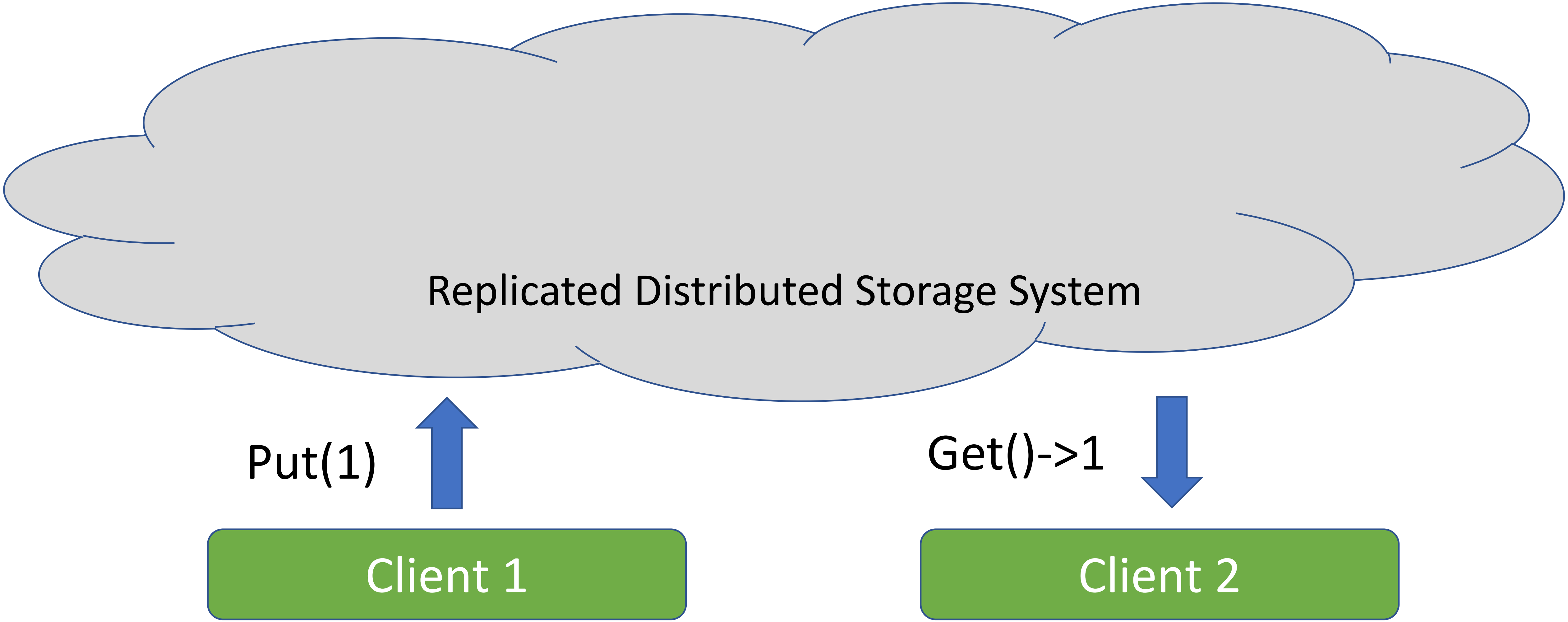
Sub-Tasks

1.	Leader should not snapshot uncommitted state	CLOSED	Flavio Paiva Junqueira
2.	Learner should not snapshot uncommitted state	OPEN	Hongchao Deng
3.	Change TRUNC to SNAP in sync phase for safety guarantee	OPEN	*Unassigned*

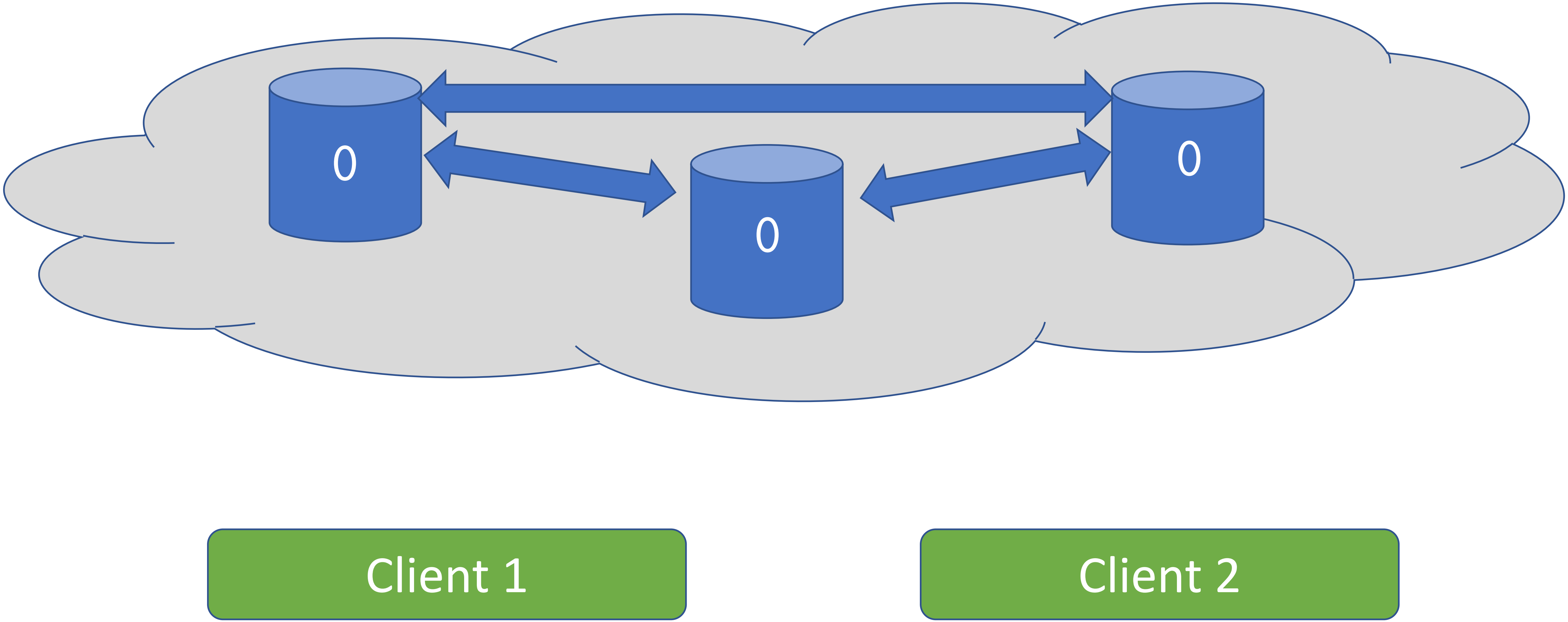
1. Created Several Sub-tasks
2. Not Resolved Yet (10 Yrs)
3. There are other similar bugs in Jira (recurring problem)

→ Need Automated Bug Finding Tool for These Bugs!

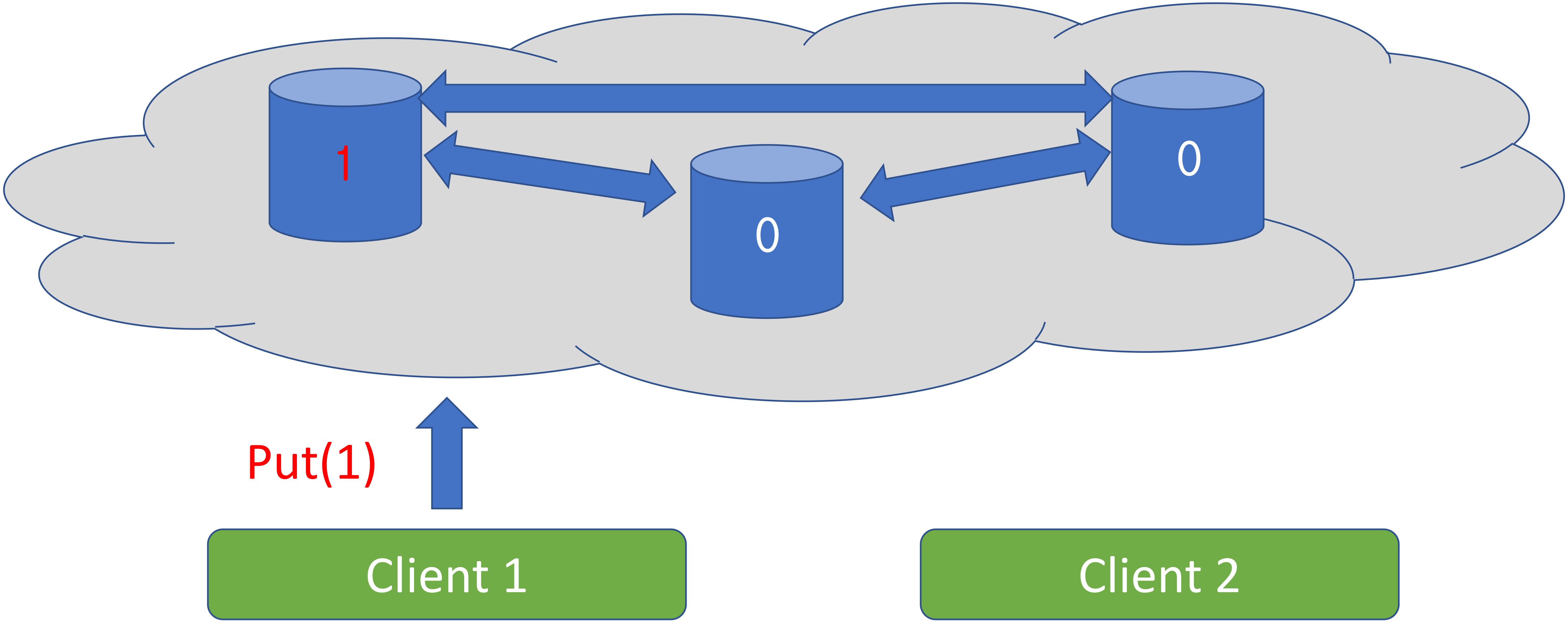
Replicated Distributed Storage Systems Provide Clients Consistent State/Data



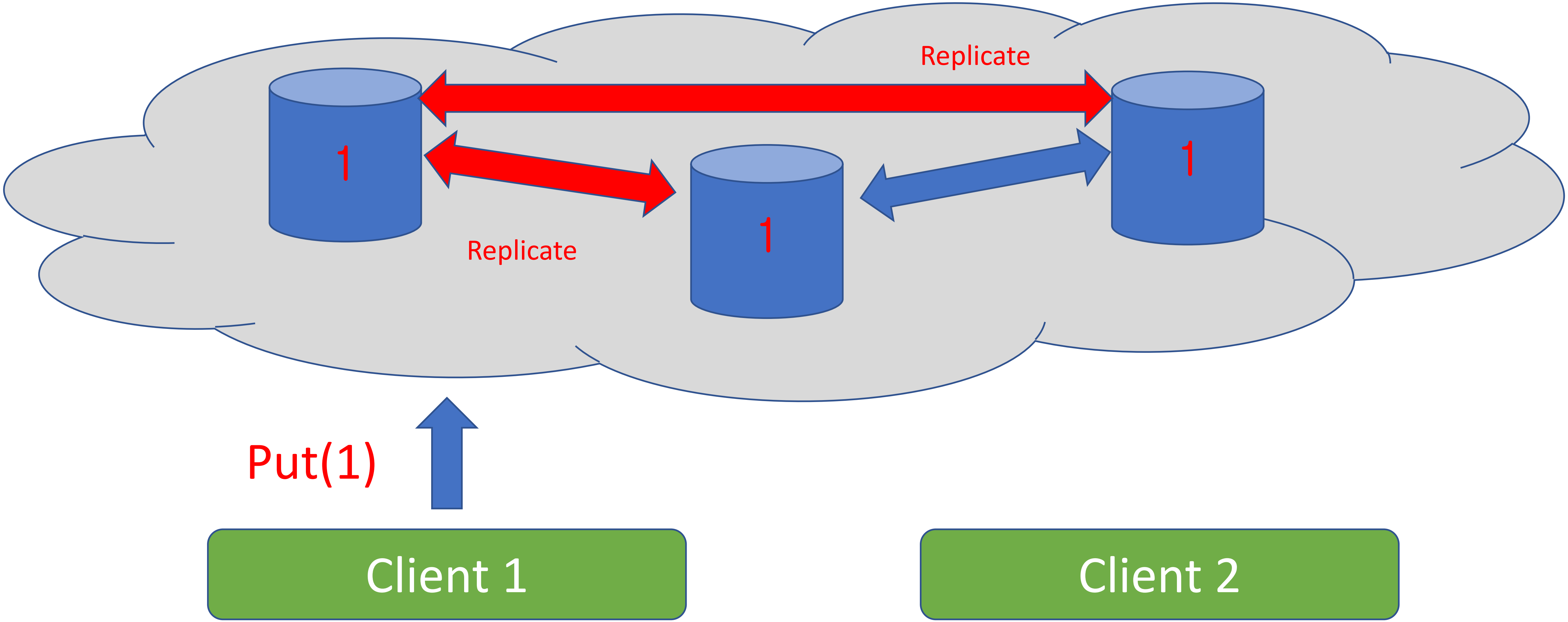
Key Enabler: Convergence Property Keeps Replicas Consistent



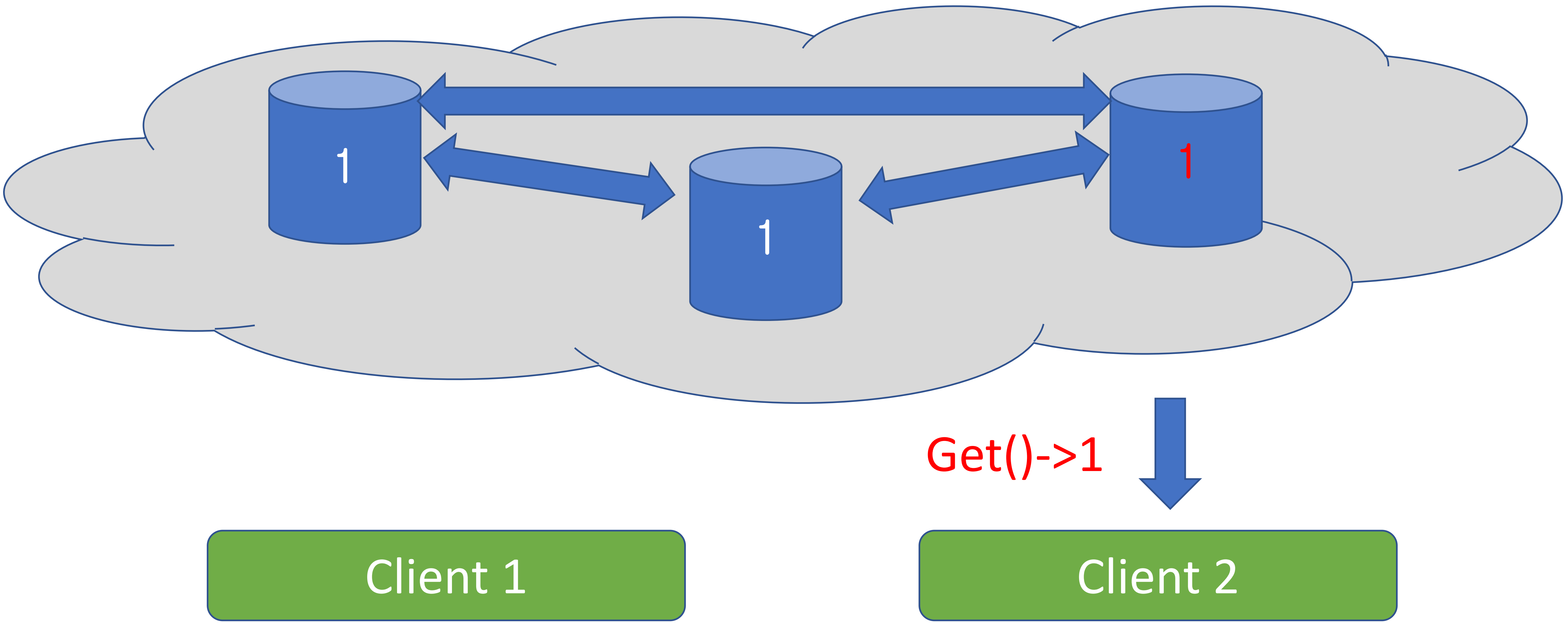
Step 1: A Replica Accepts Clients' Requests



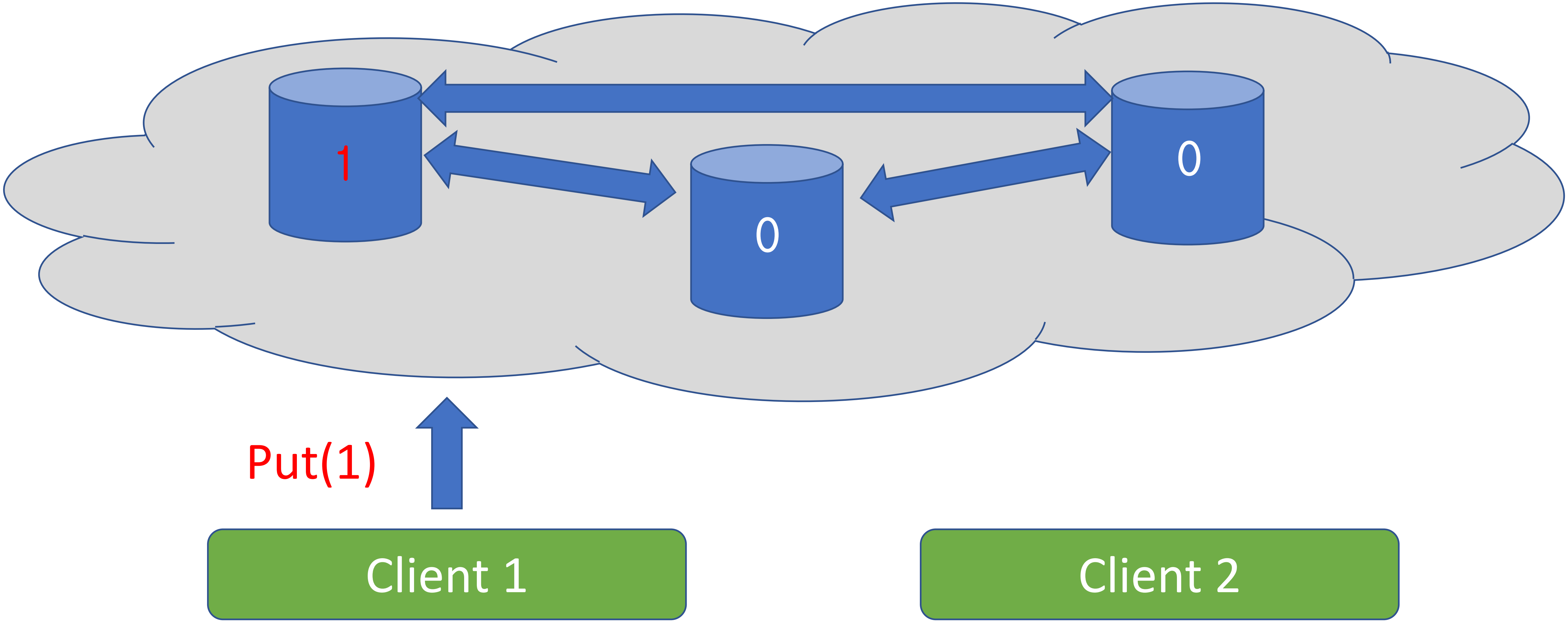
Step 2: Replicas Become Converged via Replication



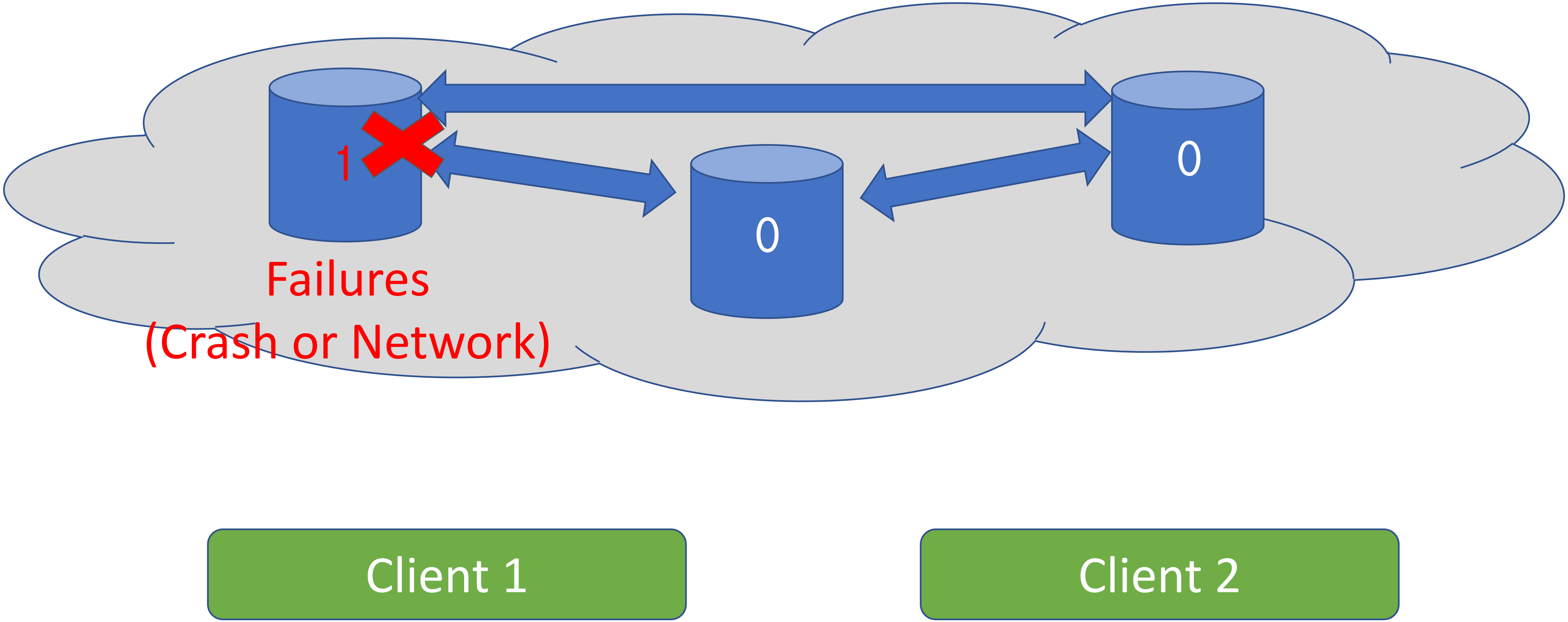
Step 3: Clients Read Consistent Data



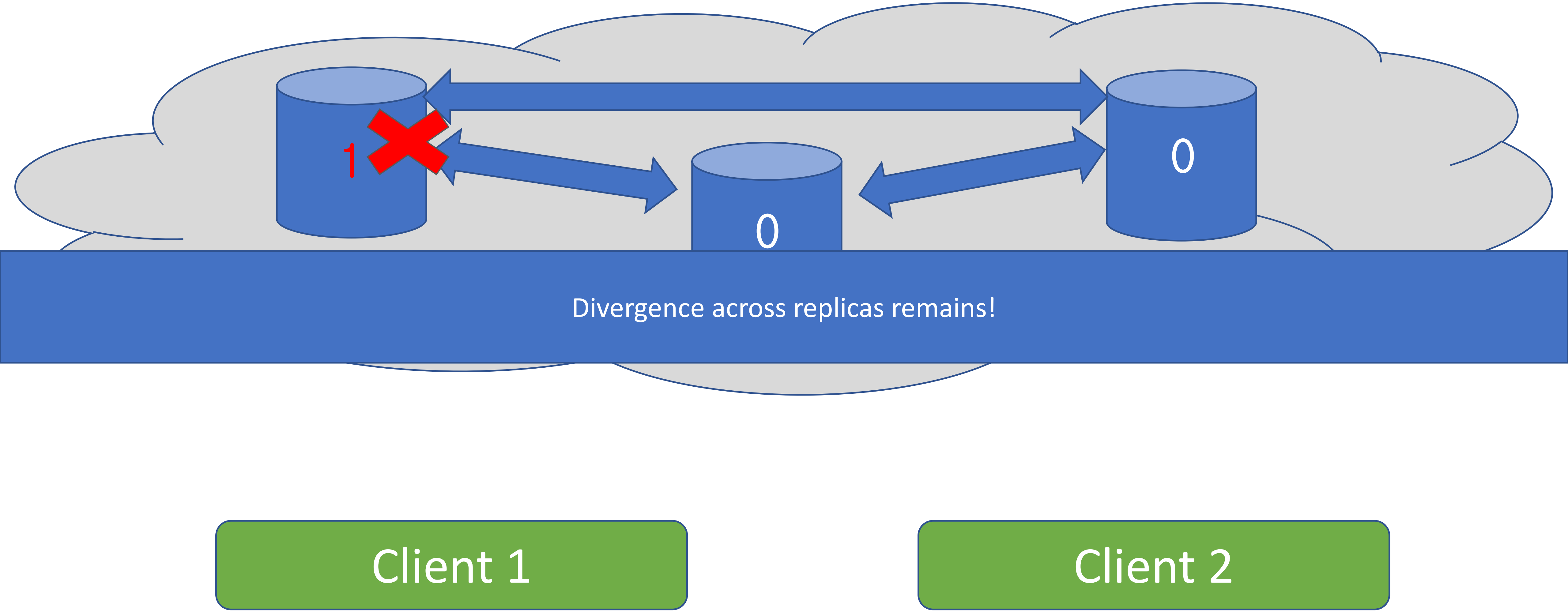
Problem 1: What if a Replica Fails?



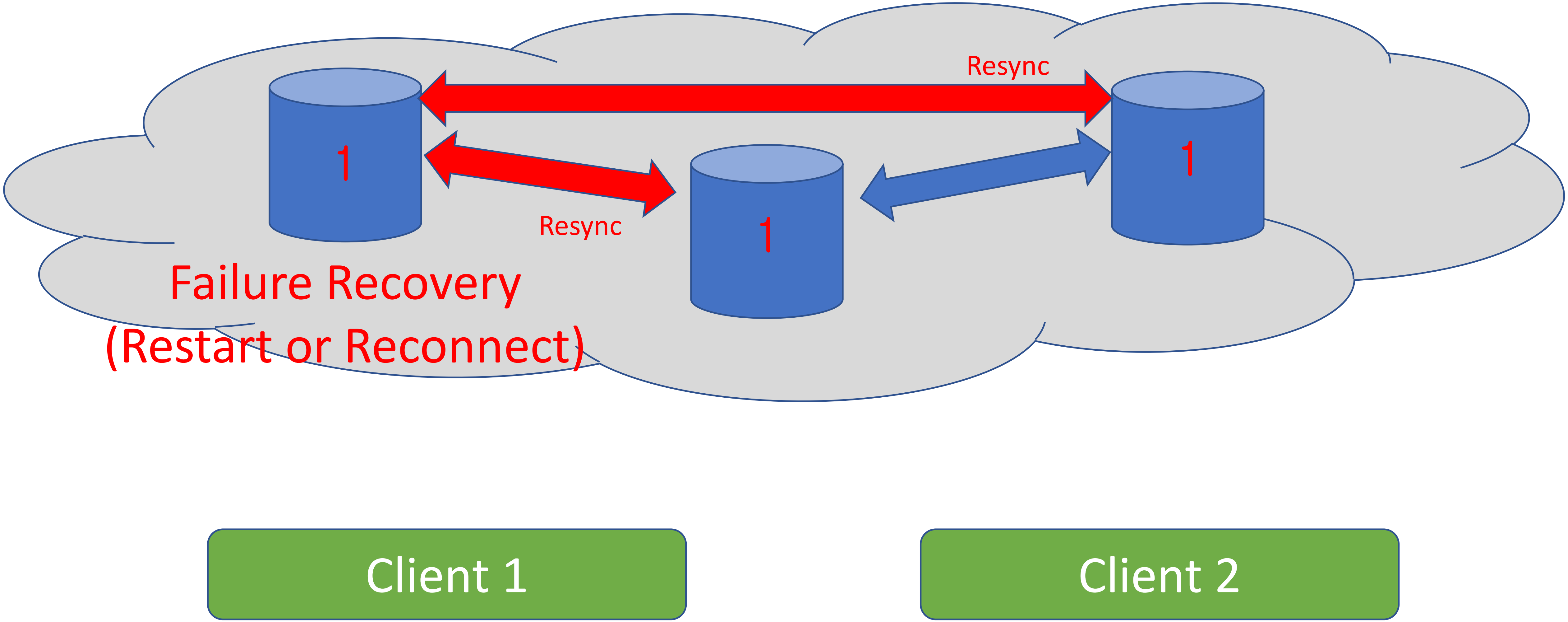
Step 1: A Replica Fails and Becomes Unavailable



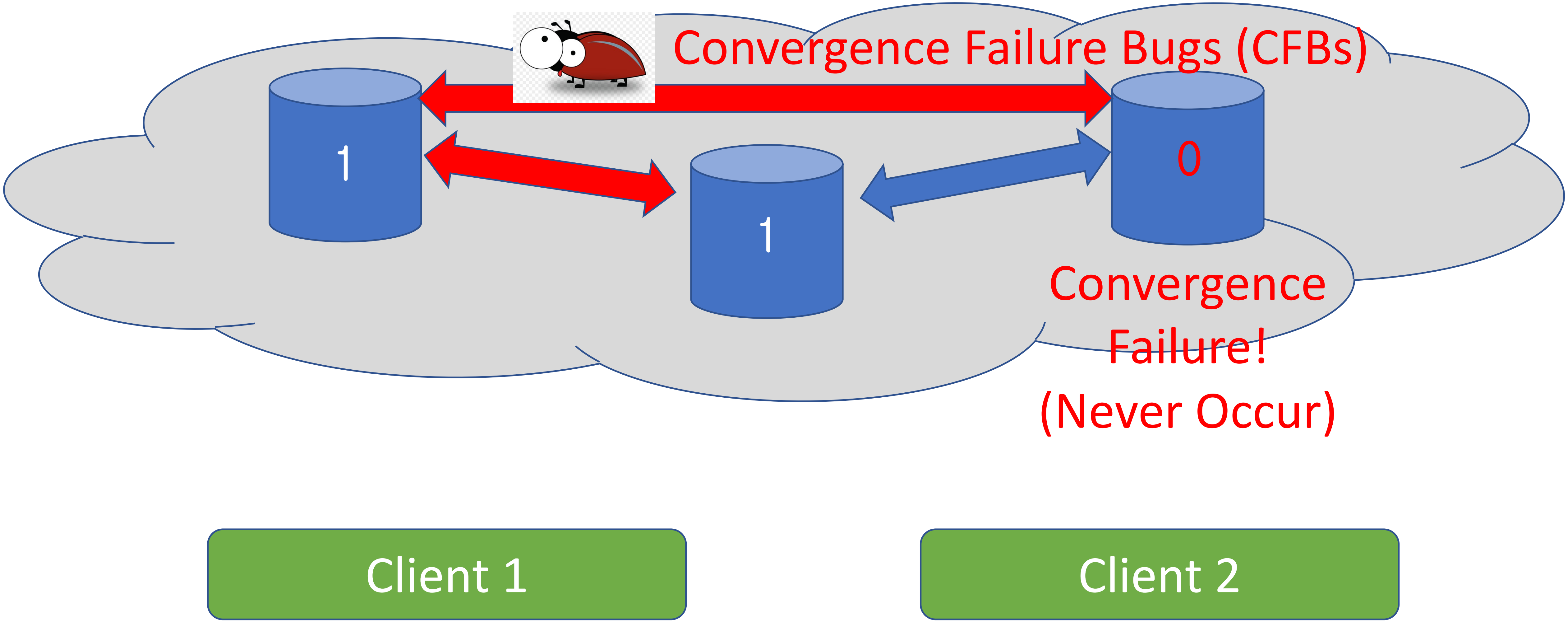
Step 2: Replication Does Not Occur and Replicas Remain Diverged



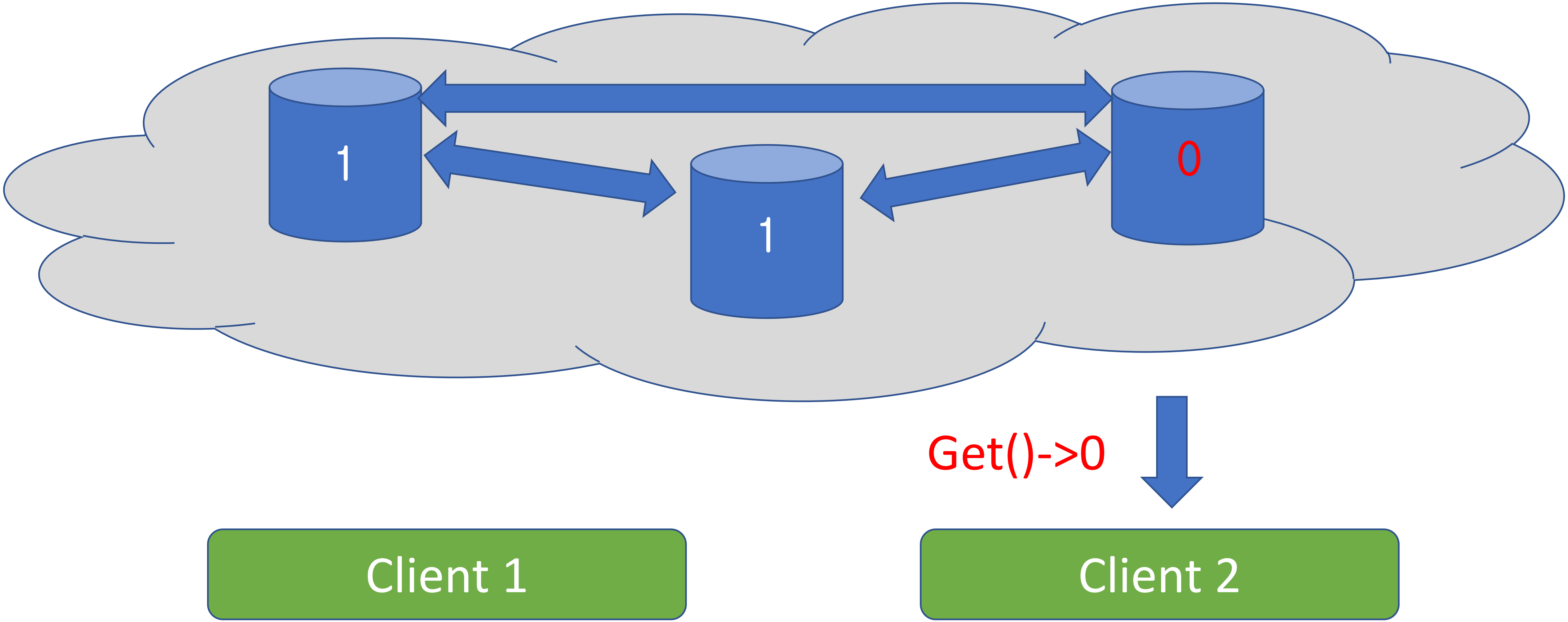
Step 3: Failure Recovery and Resync Makes Replicas Converged Again



Problem 2: Software Bugs in Resync Mechanisms May Cause Convergence Failures



Problems Cause Clients to Read Inconsistent Data



Existing Approaches: Model-based Approaches and Random Testing Approaches

- Model-Based Testing and Model-Checking
  - Problem: State space exploration is generic not targeted, therefore suffers from state explosion

- Manual Testing and Random Testing
  - Problem: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases

Our Approach: Targeted, Systematic and Exhaustive State Space Exploration to Overcome Limitation of Existing Approaches

- Model-Based Testing and Model-Checking
  - Problem: State space exploration is generic not targeted, therefore suffers from state explosion

    → Targeted State Space Exploration

- Manual Testing and Random Testing
  - Problem: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases

    → Systematic and Exhaustive State Space Exploration

Key Observation 1: Convergence Failure Bugs (CFB) can be Abstracted in Concise, Reproducible Steps

Crash Replica A and B	Crash Replica A and B
Cli: Put(1) Replica C: Processing Request Replica C: Proposing Replica C: Count for Txn ID Replica C: Record in Txn log	Put(1)
Crash Replica C	Crash Replica C
Start Replica A and B	Restart Replica A and B
Replica A: Leader election begins Replica B: Leader election begins … (Leader election events) … Replica A: Following Replica A: Getting diff from B Replica B: LEADING Replica B: Sync with A Replica B: Quorum is ready	Resync A and B

Key Observation 2: Interleave Abstracted Steps to Find New CFBs

Crash Replica A and B

Cli: Put(1)
Replica C: Processing Request
Replica C: Proposing
Replica C: Count for Txn ID
Replica C: Record in Txn log

Crash Replica C

Start Replica A and B

Replica A: Leader election begins
Replica B: Leader election begins
… (Leader election events) …
Replica A: Following
Replica A: Getting diff from B
Replica B: LEADING
Replica B: Sync with A
Replica B: Quorum is ready

Crash Replica A and B

Put(1)

Put(2)

Crash Replica C

Restart Replica A and B

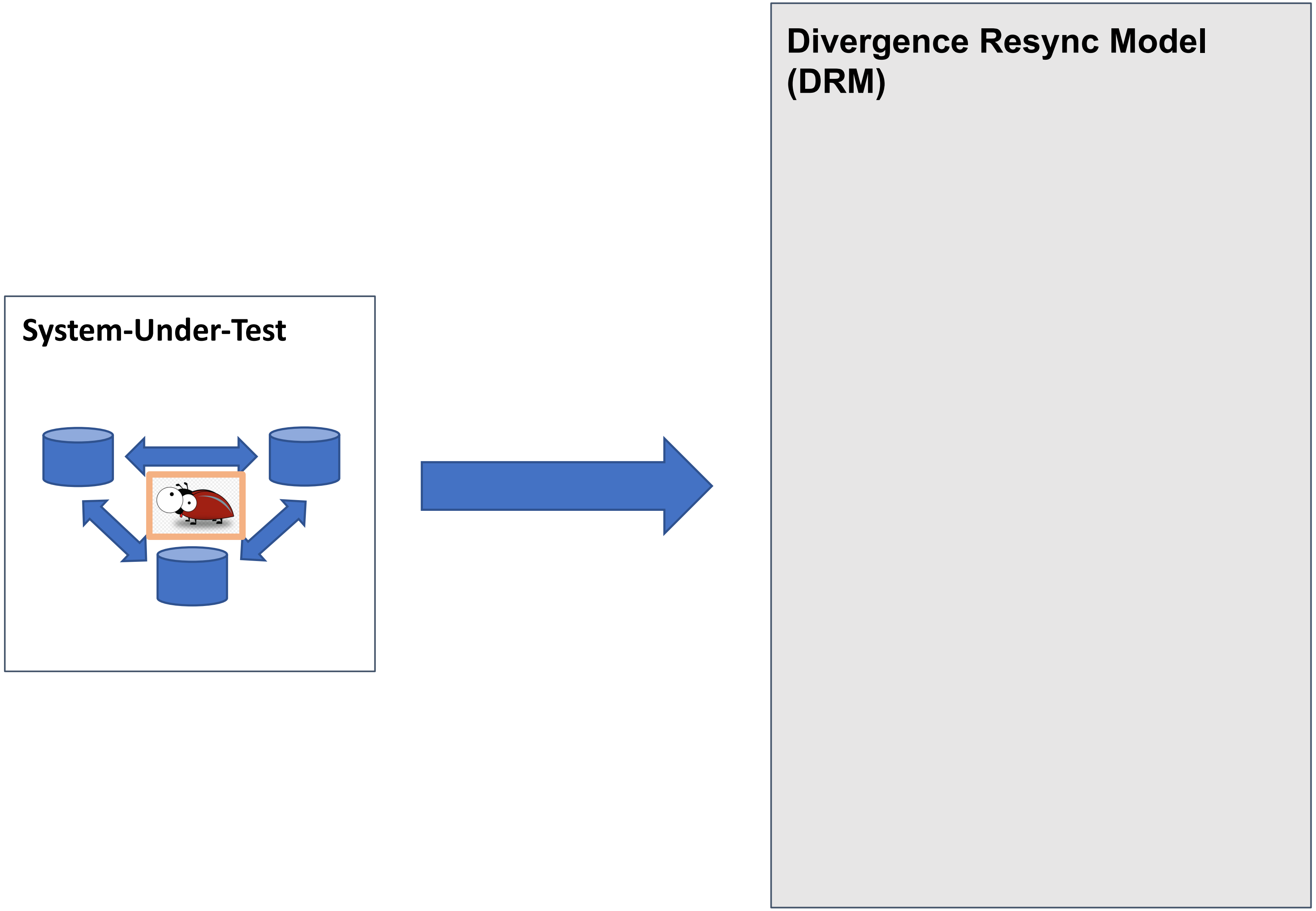
Resync A and B

Key Idea 1: Using Divergence and Convergence Events can Even Further Reduce State Exploration

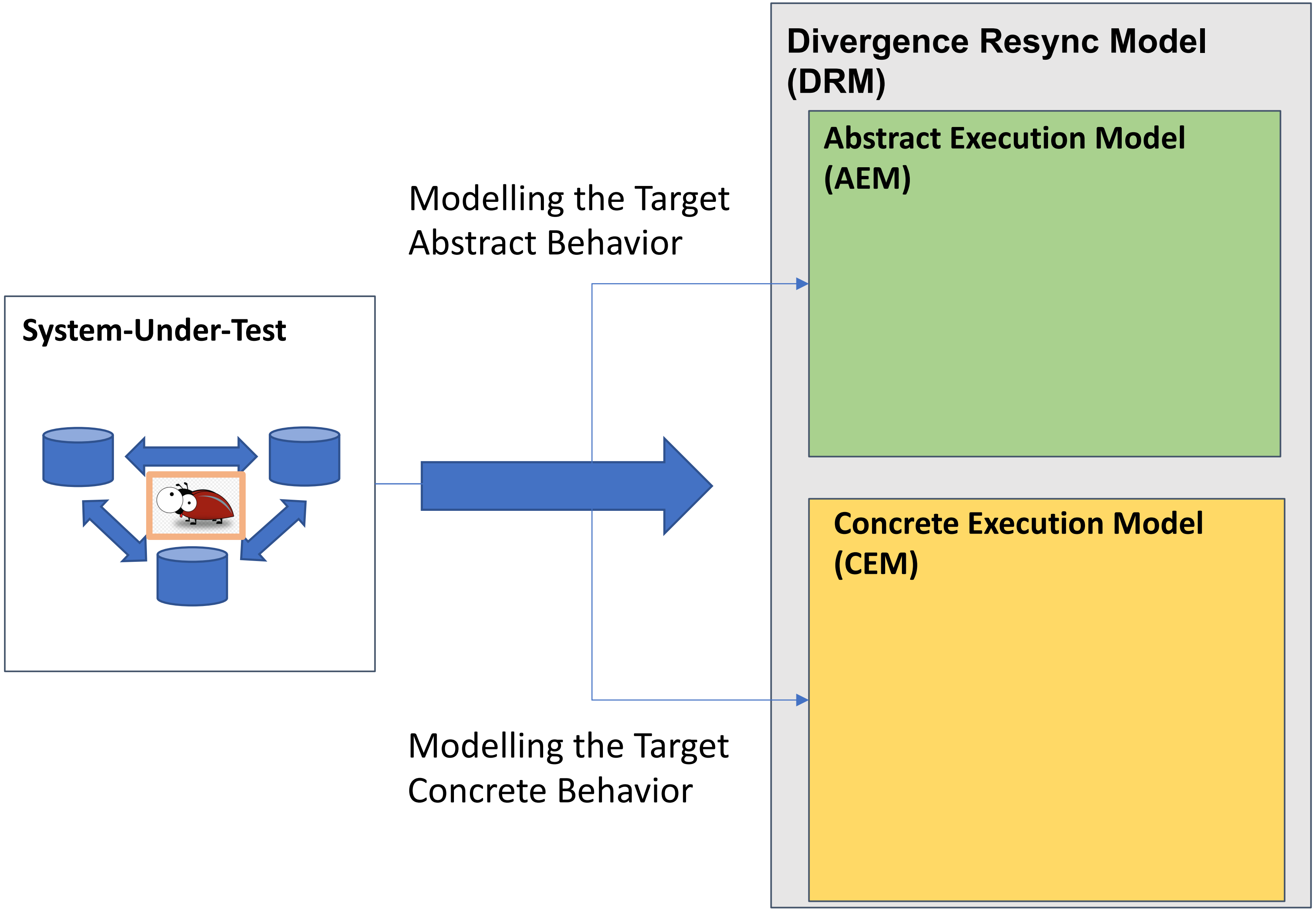
Events		Abstracted events		Type
Crash Replica A and B	→	Crash Replica A and B		
Cli: Put(1) Replica C: Processing Request Replica C: Proposing Replica C: Count for Txn ID Replica C: Record in Txn log	→	Put(1)	→	Divergence
Crash Replica C	→	Crash Replica C		
Start Replica A and B	→	Restart Replica A and B		
Replica A: Leader election begins Replica B: Leader election begins … (Leader election events) … Replica A: Following Replica A: Getting diff from B	→	Resync A and B	→	Convergence

By Focusing on Interleaving Divergence and Convergence, the state space to explore is further reduced.

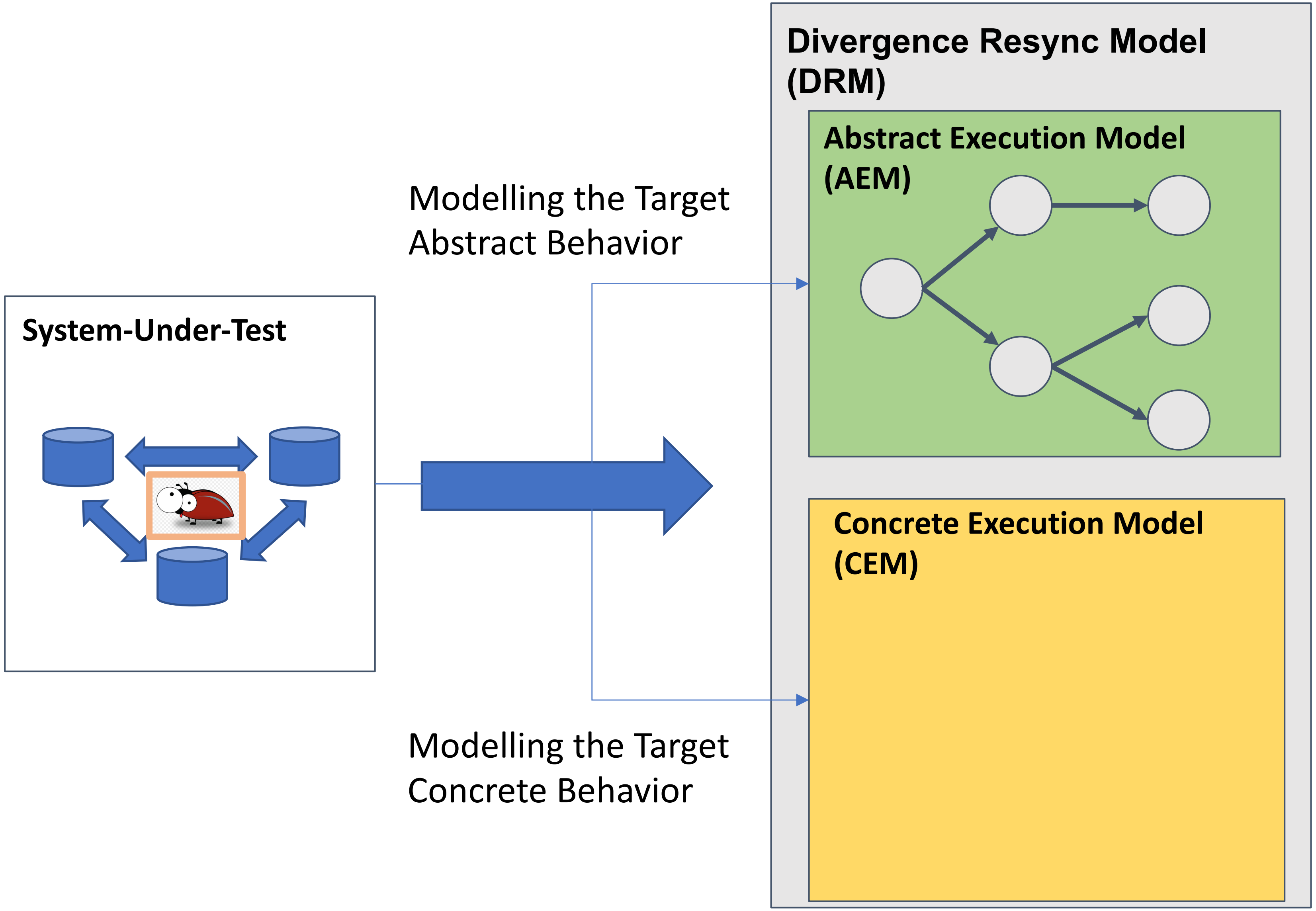
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)



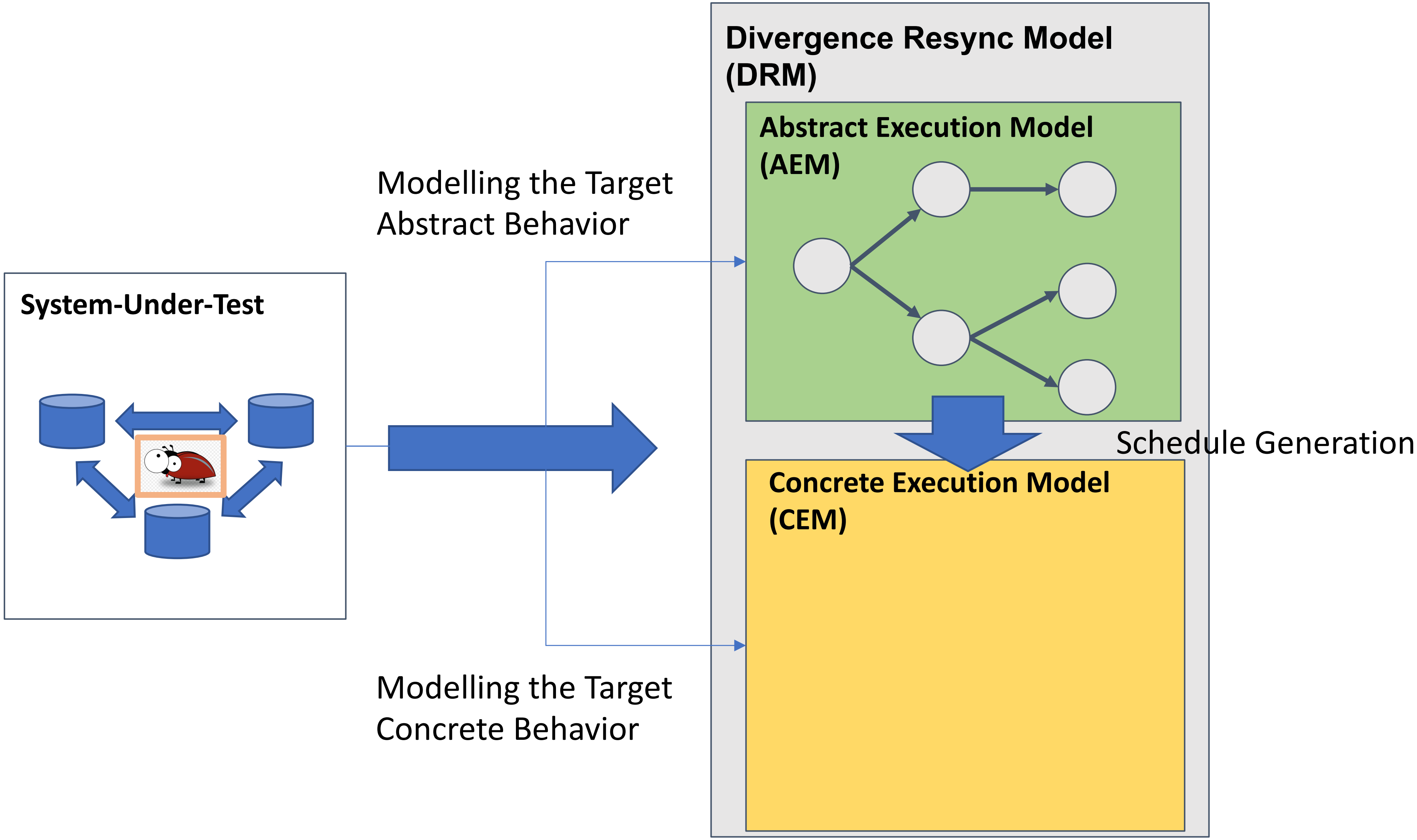
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)



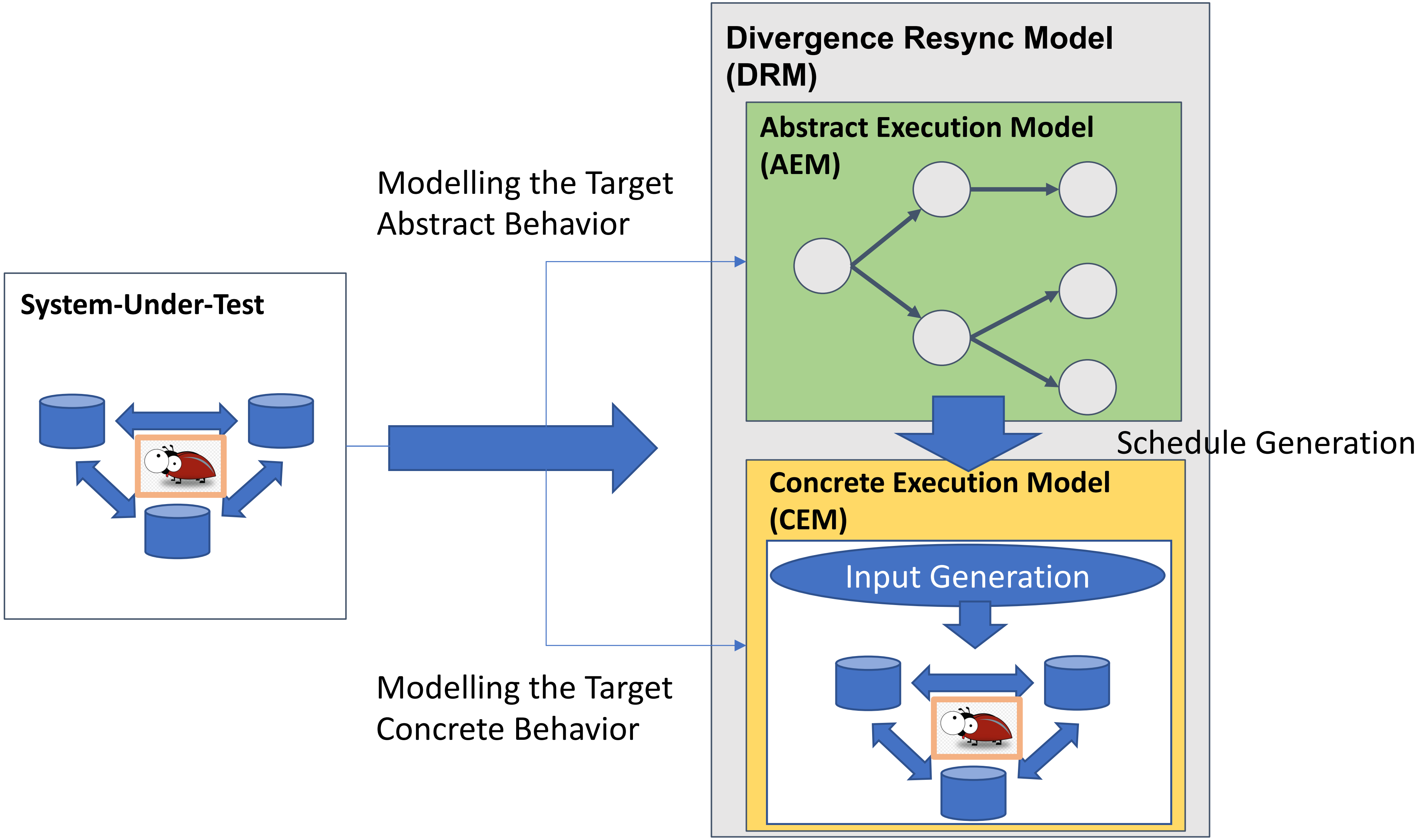
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)



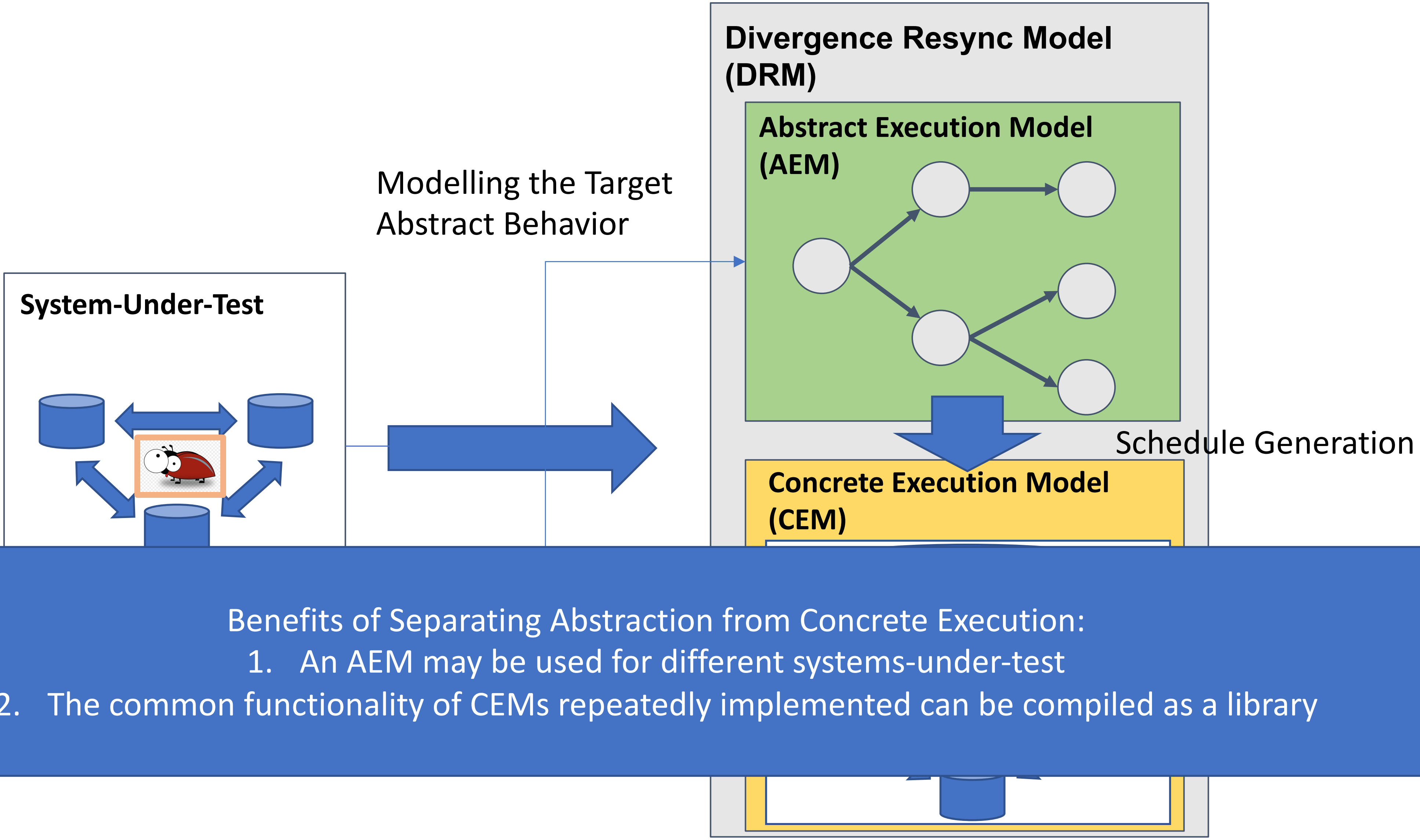
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)



Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)



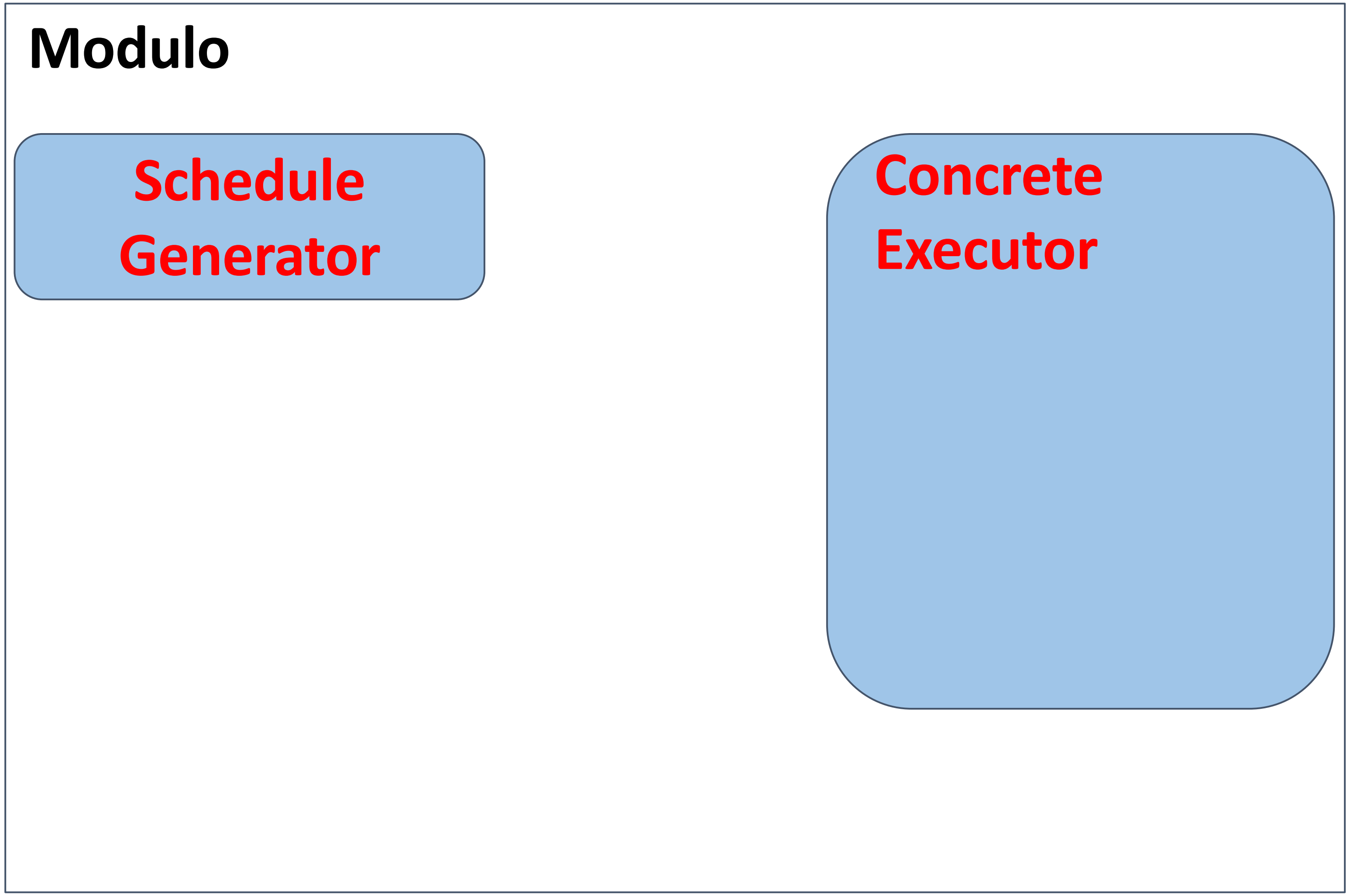
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)



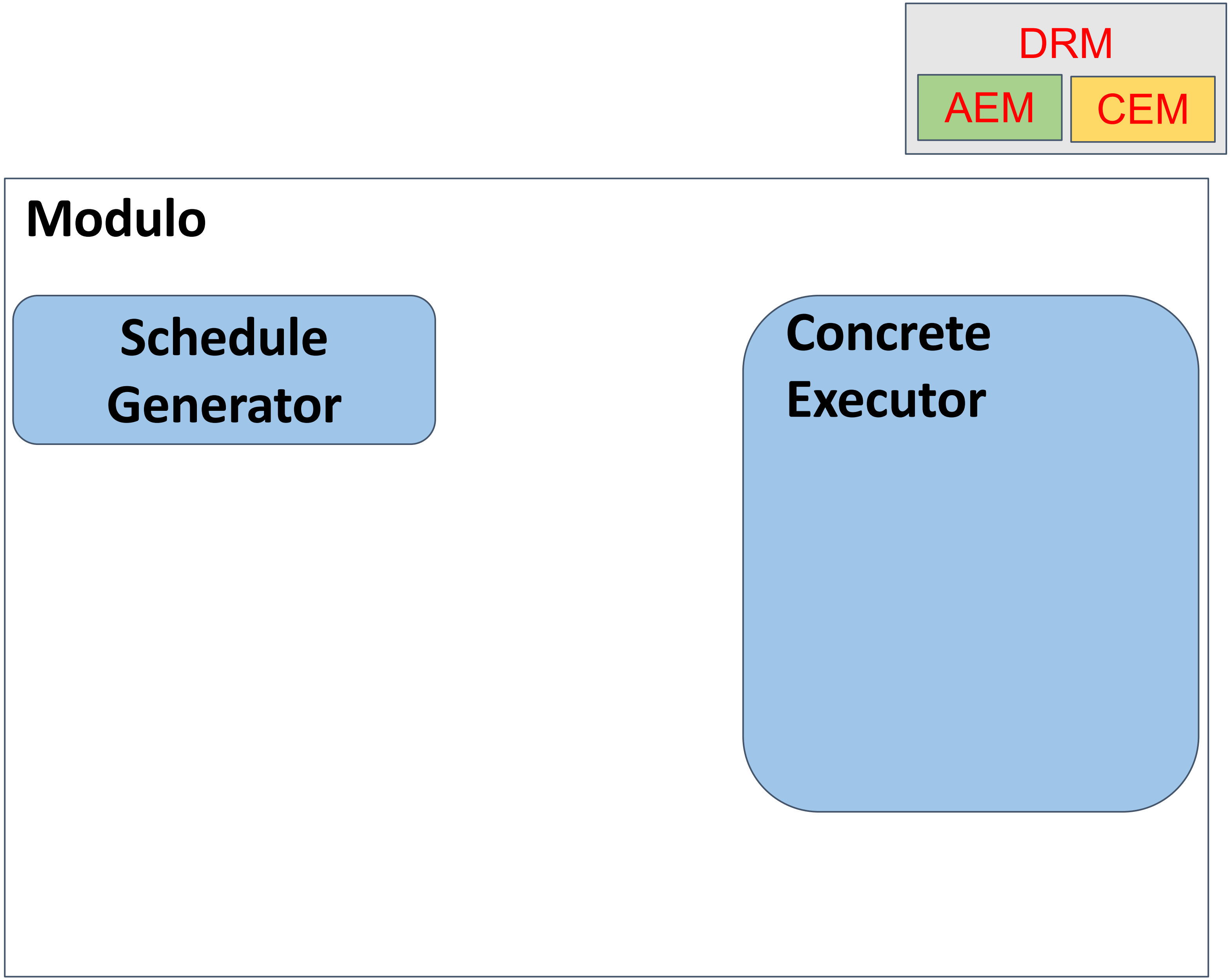
Benefits of Separating Abstraction from Concrete Execution:

1. An AEM may be used for different systems-under-test
2. The common functionality of CEMs repeatedly implemented can be compiled as a library

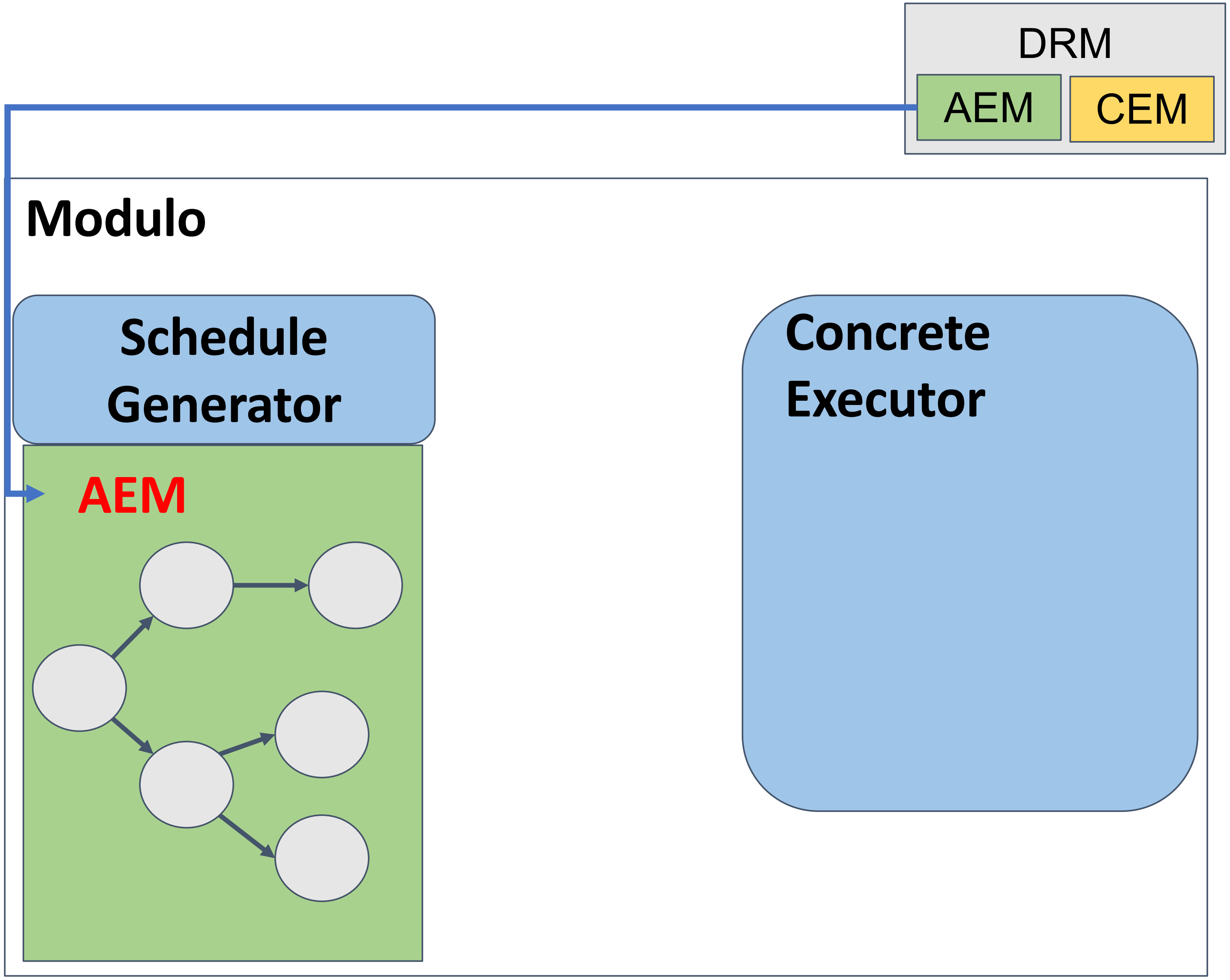
Modulo Architecture: Schedule Generator and Concrete Executor



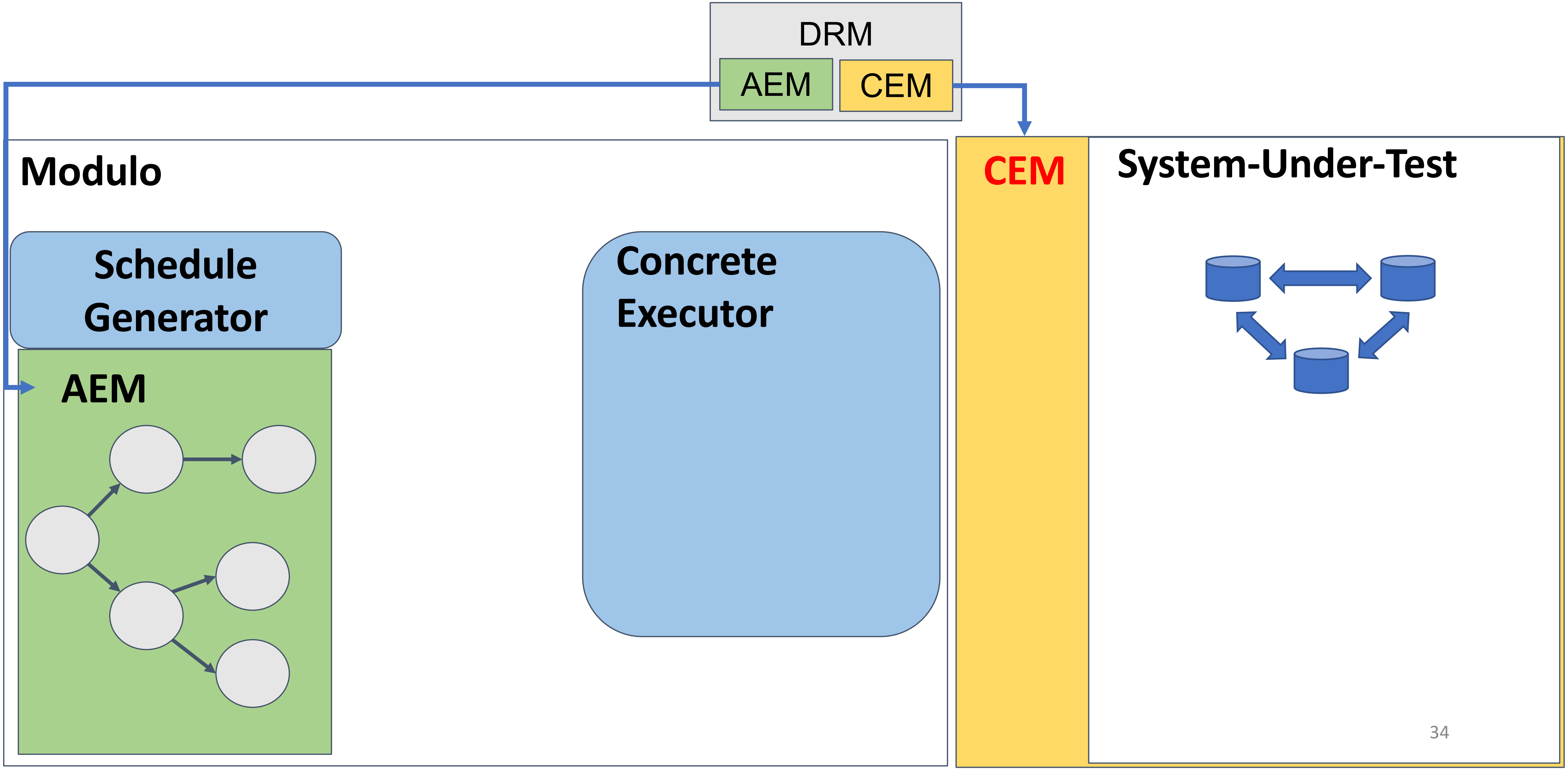
Modulo Architecture: Users Specify and Provide a DRM



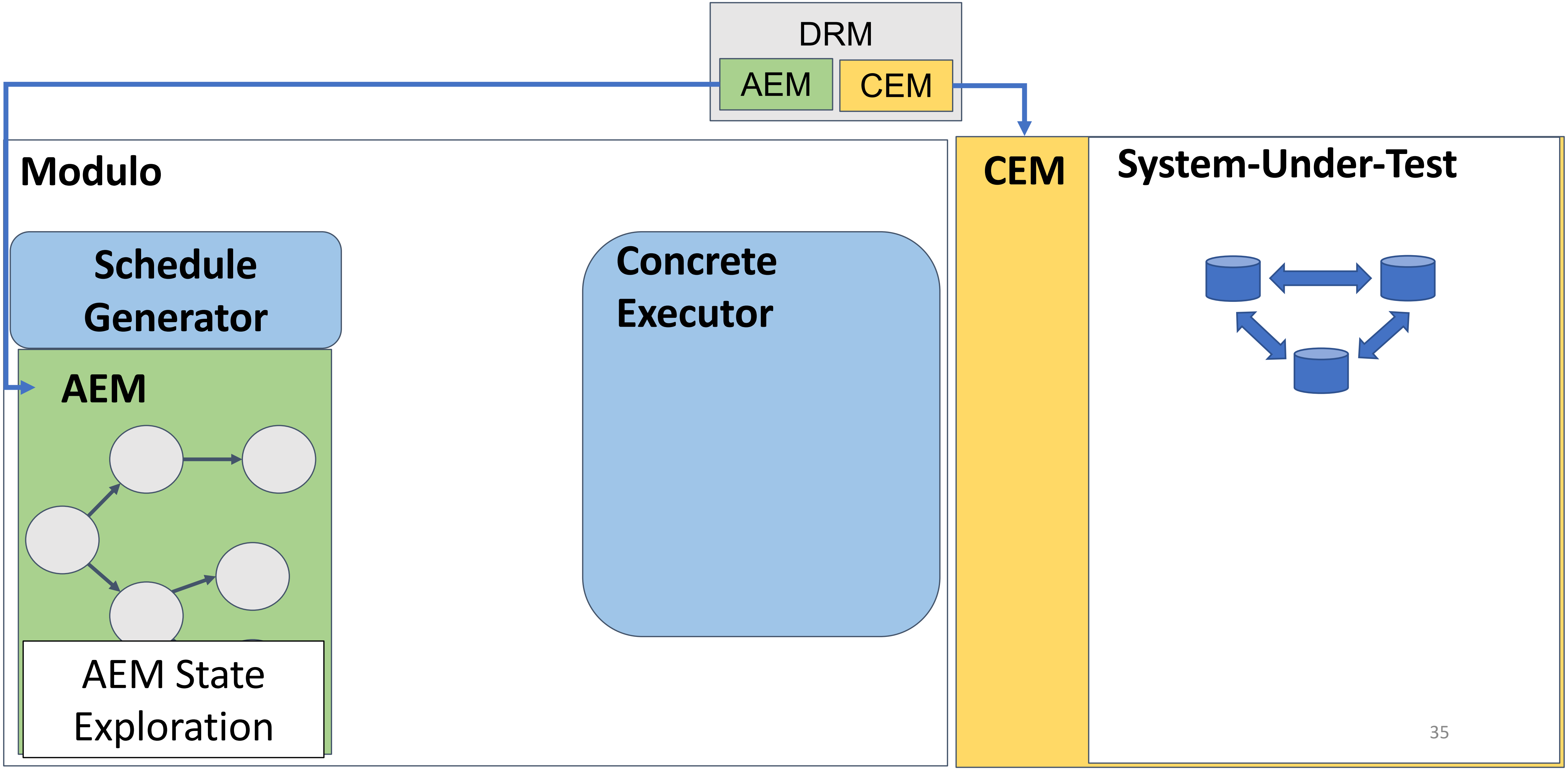
Modulo Architecture: AEM for Schedule Generator



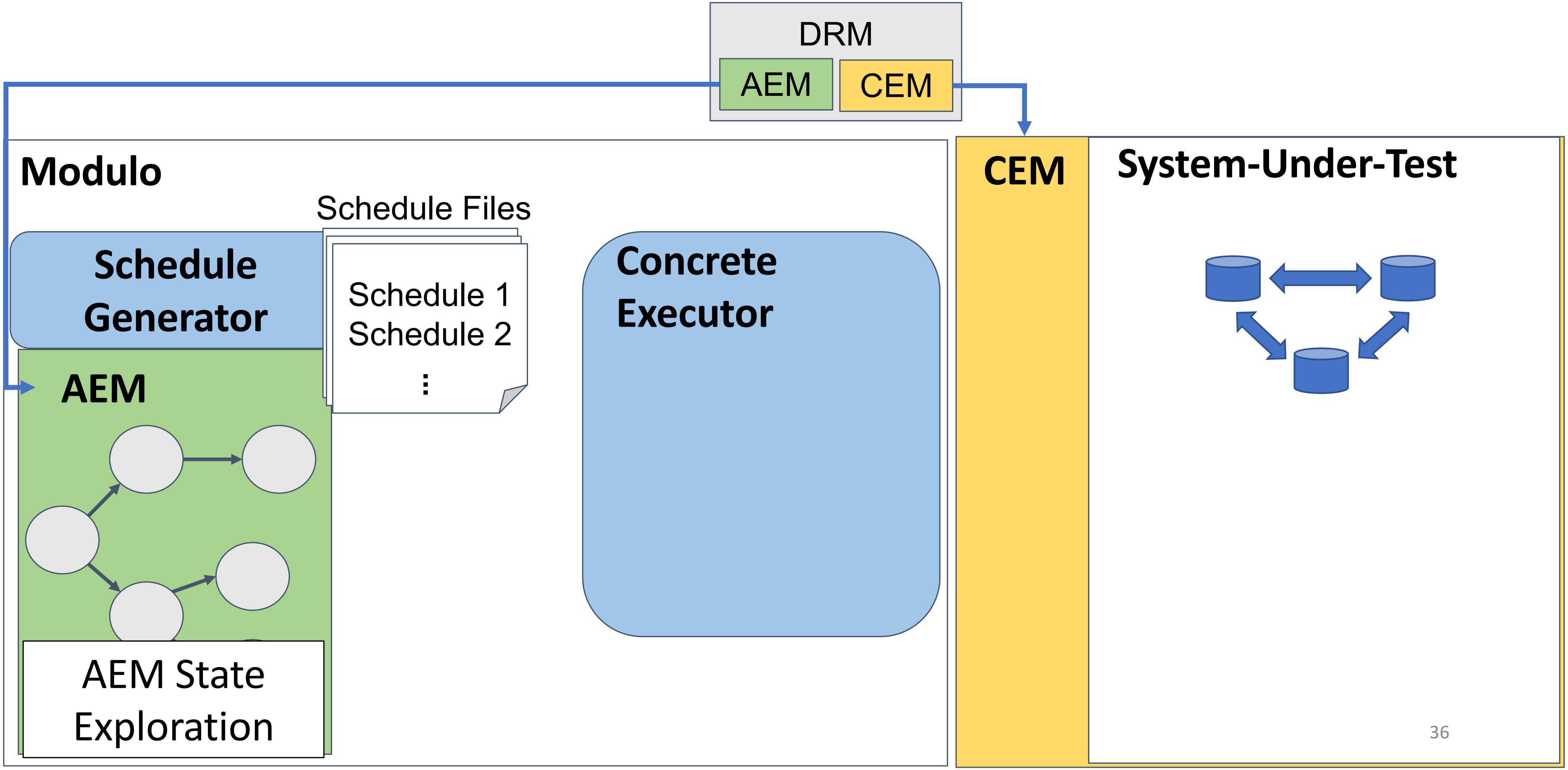
Modulo Architecture: CEM for Concrete Executor



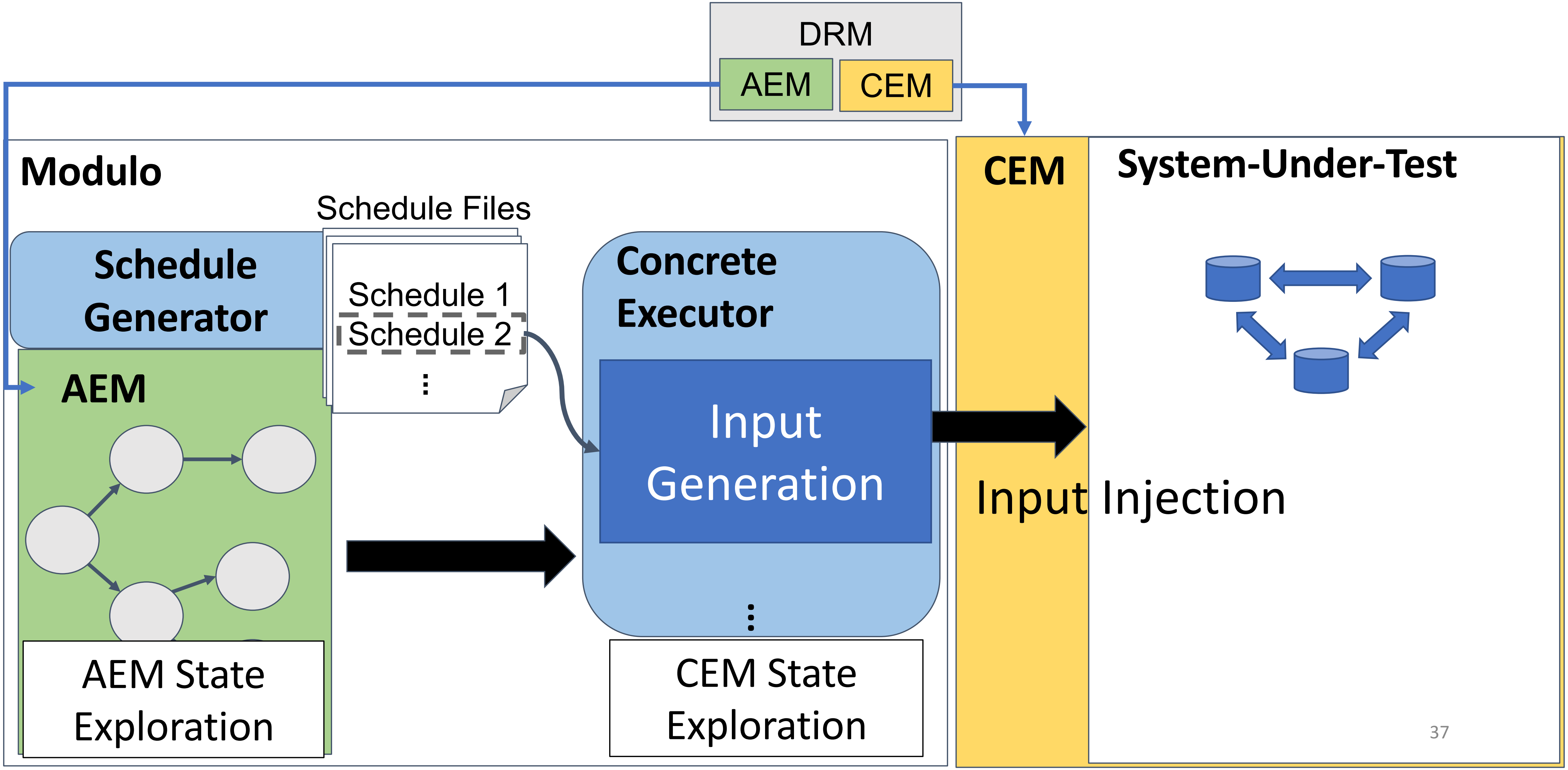
Modulo Architecture: AEM State Exploration



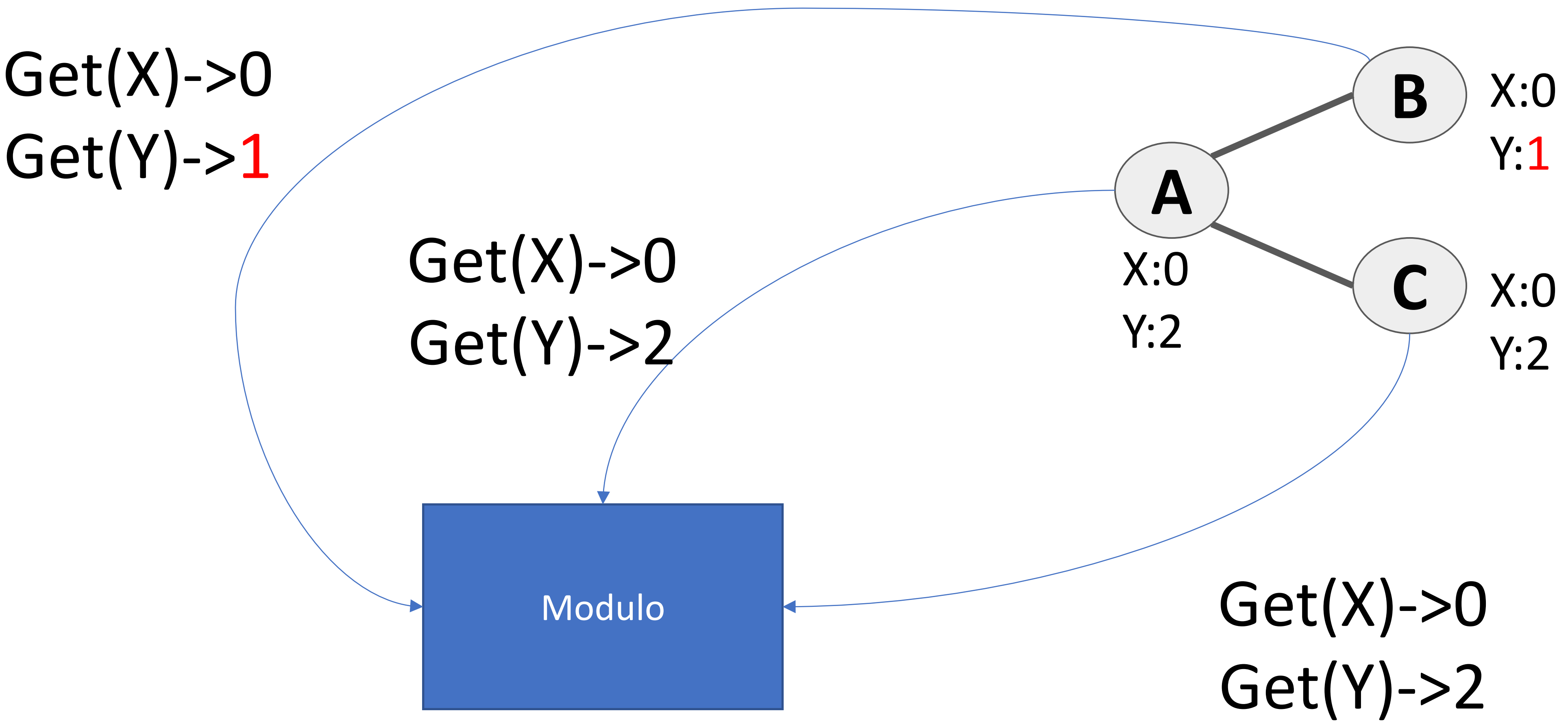
Modulo Architecture: AEM State Exploration



Modulo Architecture: CEM Input Injection



Modulo Architecture: Checking if Convergence Fails after Each Schedule Execution

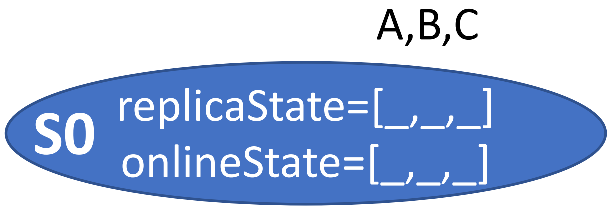


Convergence Failure Detected!

Abstract Execution Model: Each State Contains State Variables

Abstract Execution Model: User-Provided Parameters Make the State Space Concrete

numOps = 2
numReplicas = 3

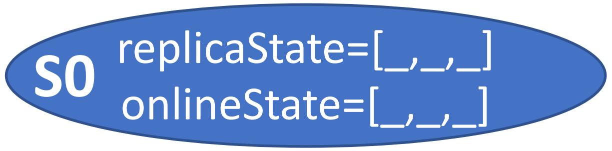


Abstract Execution Model: Predefined Write Sequence is Generated

setData(X,0) setData(X,1) setData(X,2)

numOps = 2
numReplicas = 3

A,B,C



Abstract Execution Model: Writing Monotonically Increasing Values

setData(X,**0**) setData(X,**1**) setData(X,**2**)

Values are monotonically increasing

numOps = 2
numReplicas = 3

A,B,C

S0 replicaState=[_,_,_]
onlineState=[_,_,_]

Abstract Execution Model: Indexing Each Write

	setData(X,0)	setData(X,1)	setData(X,2)
Index	0	1	2

numOps = 2
numReplicas = 3

A,B,C

S0 replicaState=[_,_,_]
onlineState=[_,_,_]

Abstract Execution Model: Meaning of Each State Variables

	setData(X,0)	setData(X,1)	setData(X,2)
Index	0	1	2

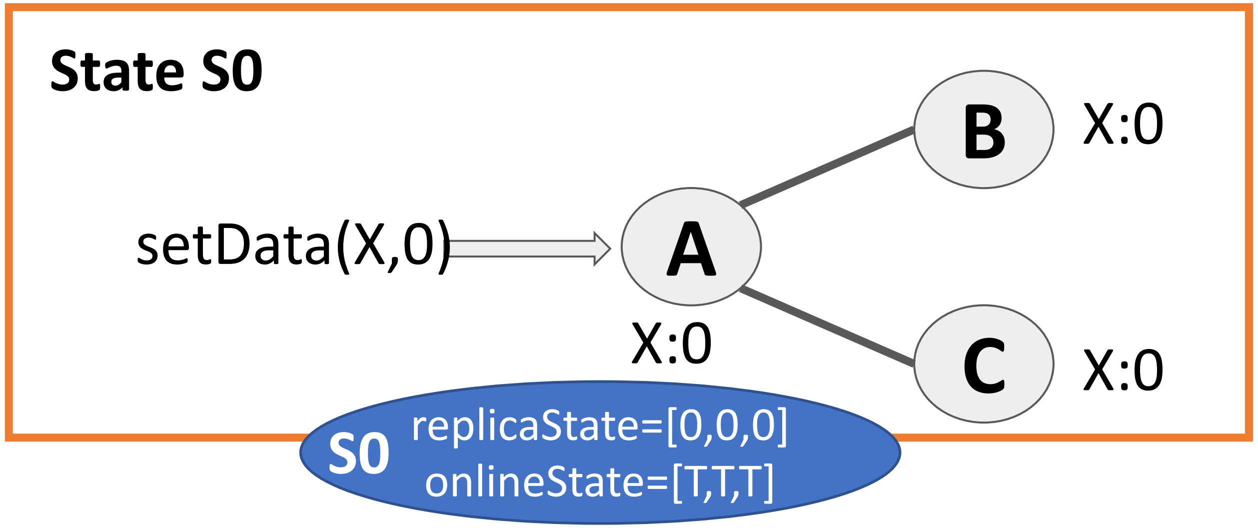
numOps = 2
numReplicas = 3

A,B,C

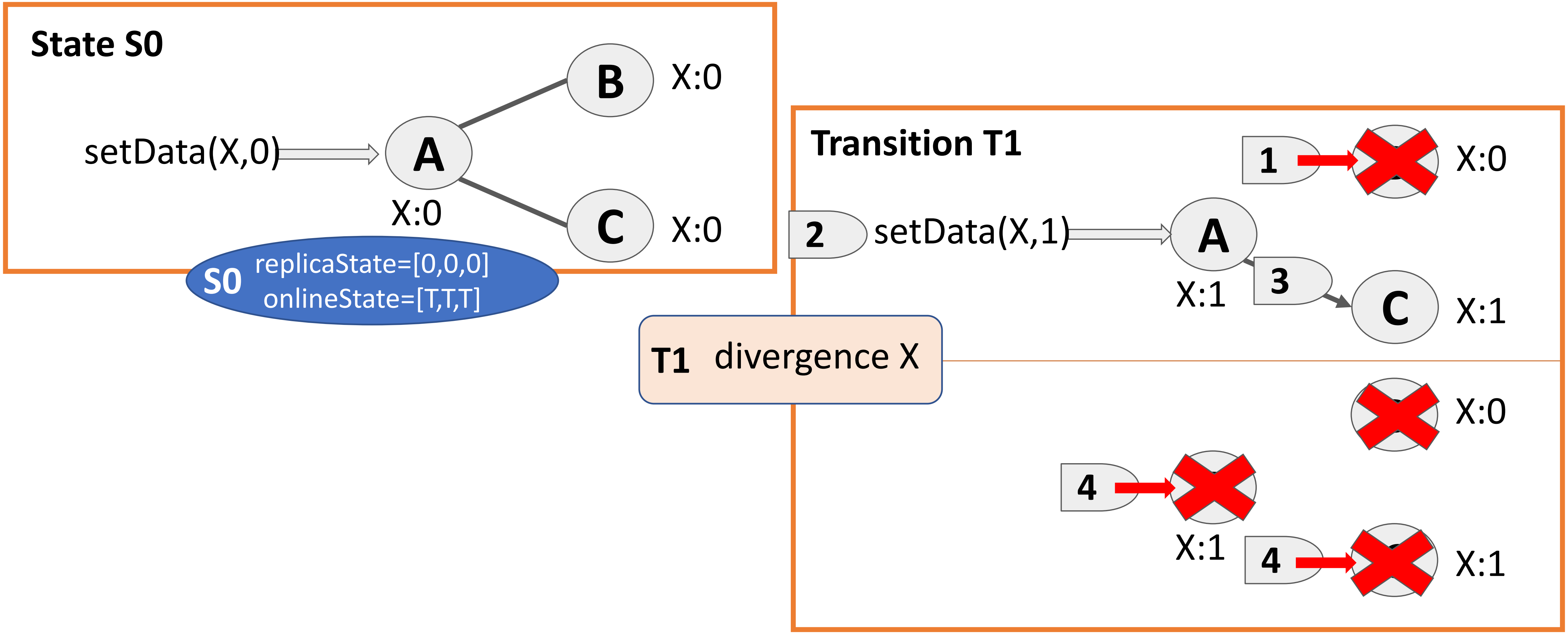
S0 replicaState=[_,_,_]
onlineState=[_,_,_]

Up to which write
Which replicas are available

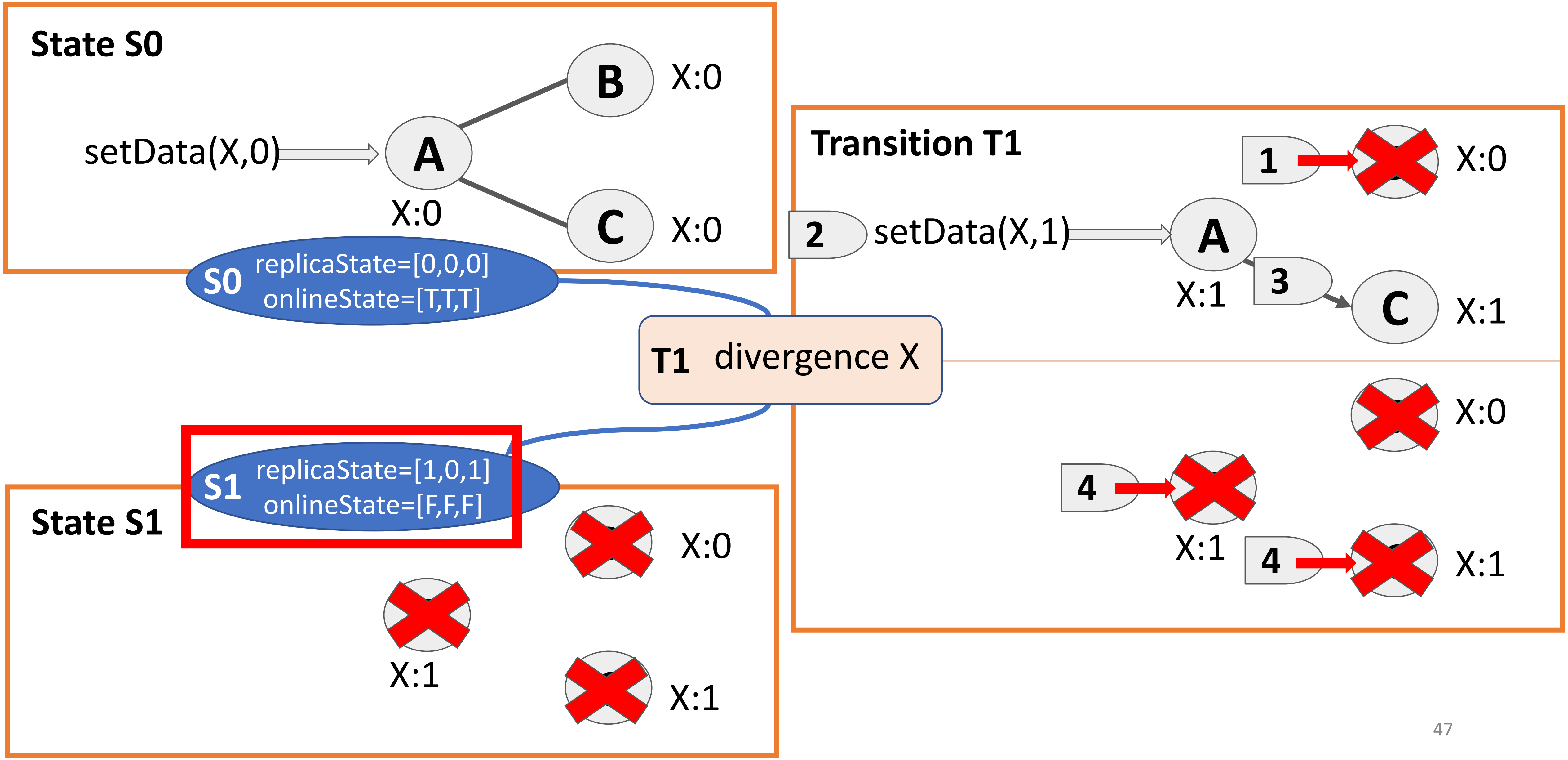
Abstract Execution Model: Initial State S0



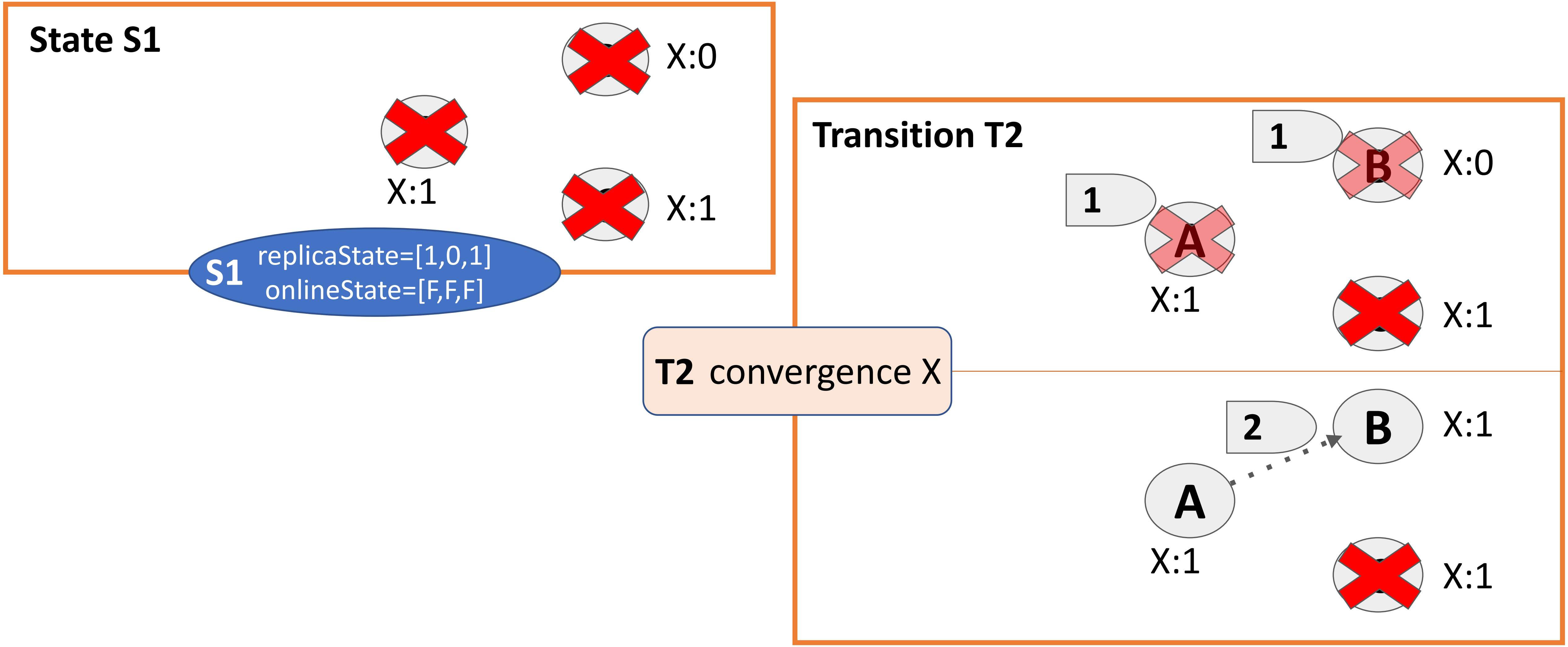
Abstract Execution Model: Applying a Divergence Transition



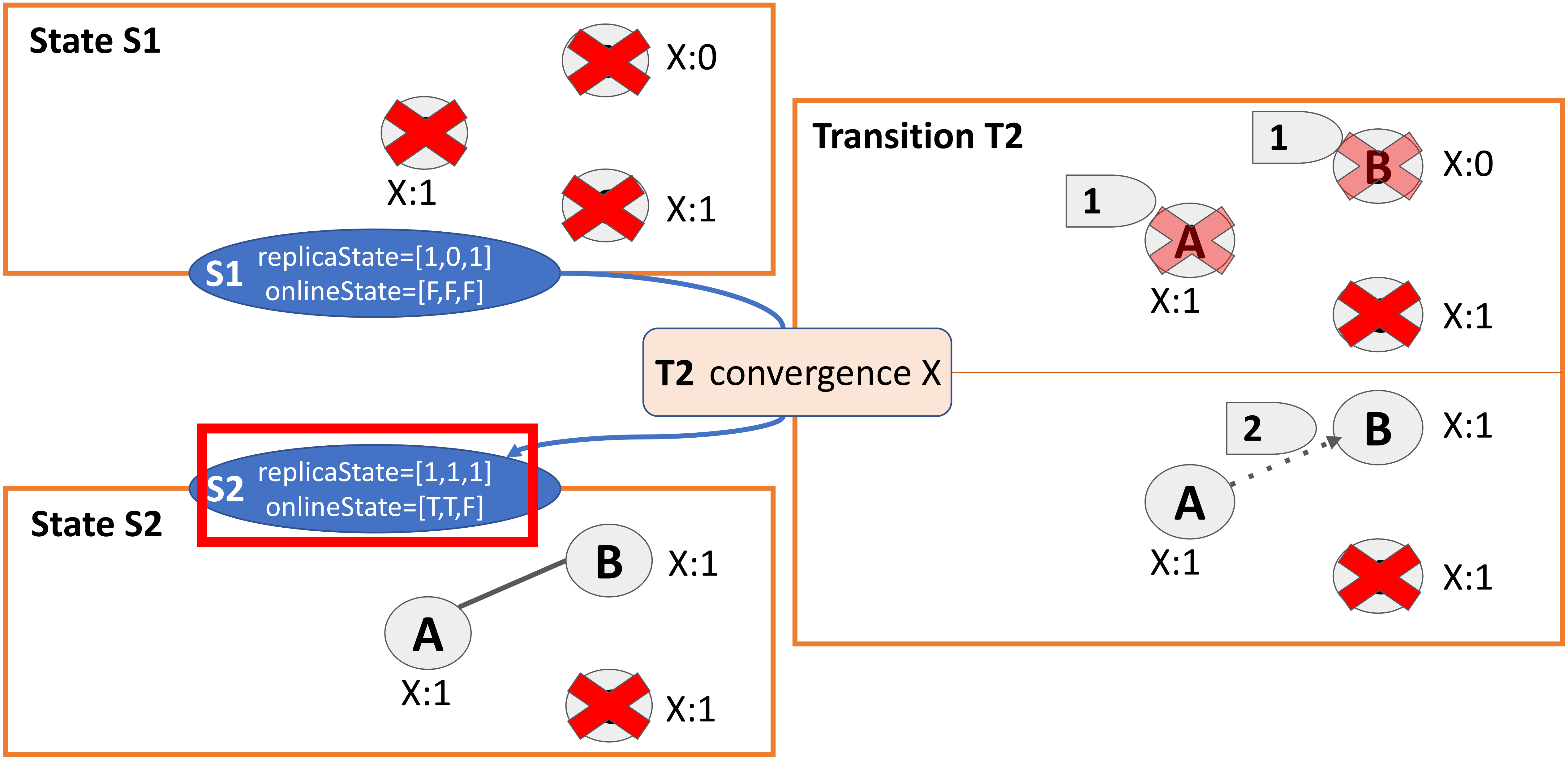
Abstract Execution Model: Updating State Variables



Abstract Execution Model: Applying a Convergence Transition



Abstract Execution Model: Updating State Variables



Concrete Execution Model: Generating Inputs by Translating AEM Transitions into Concrete Test Inputs

AEM Transitions

T1 divergence X

Intermediate Representation

Crash B
Write 1 to X
Crash A and C

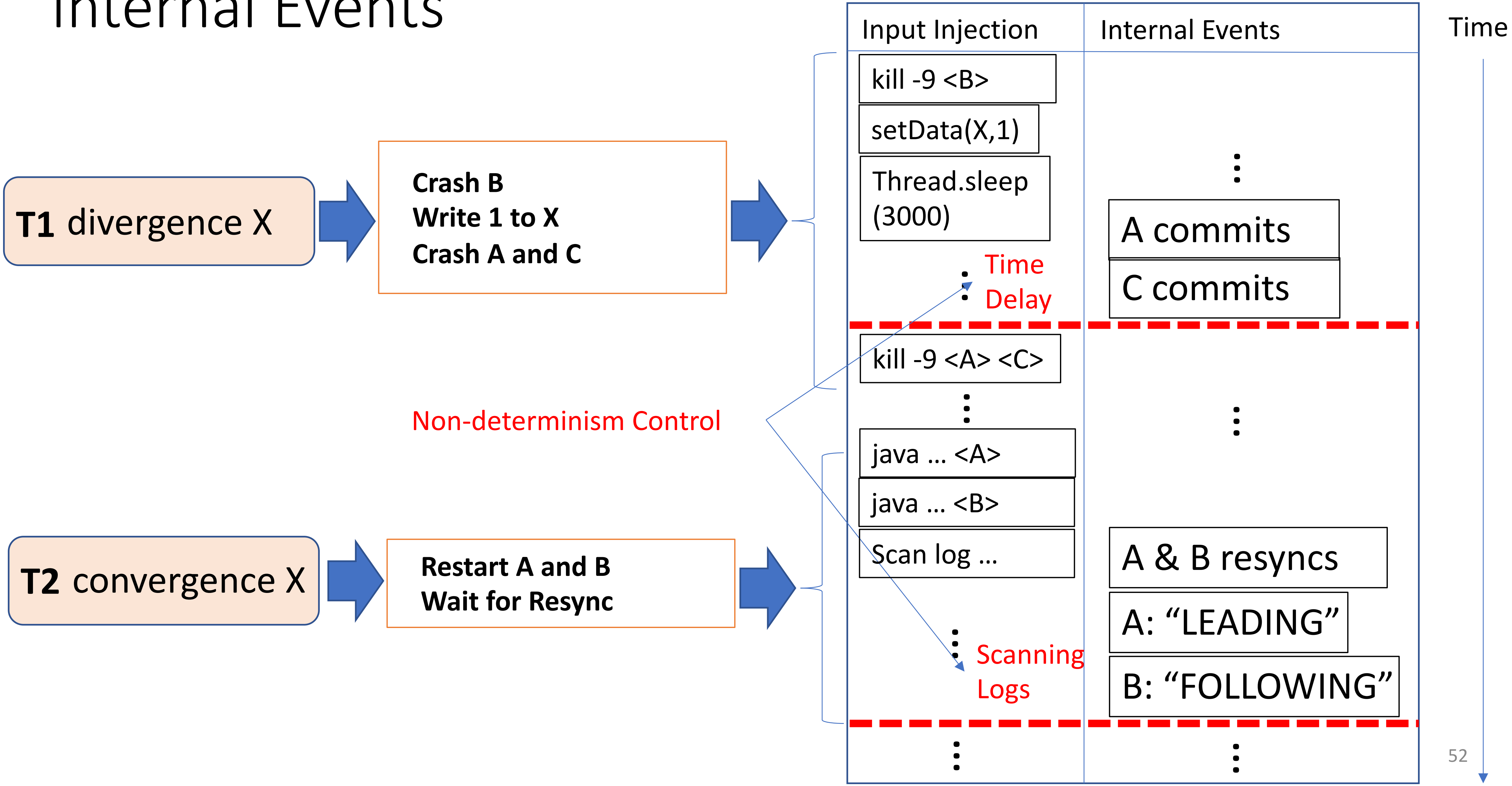
Concrete Test Inputs

```
$ kill -9 <B>
setData(X,1)
Thread.sleep(3000)
$ kill -9 <A> <C>
```

Concrete Execution Model: Generating Inputs by Translating AEM Transitions into Concrete Test Inputs

AEM Transitions	Intermediate Representation	Concrete Test Inputs
T1 divergence X	**Crash B** **Write 1 to X** **Crash A and C**	**$ kill -9 <B>** **setData(X,1)** **Thread.sleep(3000)** **$ kill -9 <A> <C>**
T2 convergence X	**Restart A and B** **Wait for Resync**	**$ java … QuorumPeerMain <A>/zoo.cfg** **$ java … QuorumPeerMain <B>/zoo.cfg** **Scan logs for “LEADING” or “FOLLOWING"**

Concrete Execution Model: Injecting Inputs Relative to Internal Events



Implementation

- 8.4K LoC in total
  - Schedule Generator: 0.3K LoC
  - Concrete Executor: 0.8K LoC
  - Divergence Resync Models: 7.3K LoC
    - AEMs: 2.8K LoC
    - CEMs: 4.6K LoC
- Applied to 3 Replicated Distributed Storage Systems
  - ZooKeeper
  - MongoDB
  - Redis

Modulo Found CFBs in Popular Distributed Systems

Bug ID	DRM	Root Cause
ZooKeeper Bug #1 (New Bug!)	Q/C/Z-DRM	Fail to remove invalid conflicting operations (missing TRUNC invocation)
ZooKeeper Bug #2 (New Bug!)	Q/C/Z-DRM	Fail to remove invalid conflicting operations (file handling logic error)
ZooKeeper Bug #3 (New Bug!)	Q/C/Z-DRM	Fail to replicate operations due to an incomplete log
ZooKeeper Bug #4(New Bug!)	Q/C/Z-DRM	Fail to truncate operations due to a pointer handling mistake
ZooKeeper Bug #5 (New Bug!)	Q/C/Z-DRM	Fail to truncate operations due to missing invocation
MongoDB Bug #1 (New Bug!)	Q/C/M-DRM	Fail to remove invalid conflicting operations (incomplete timestamp info)
MongoDB Bug #2	Q/C/M-DRM	Fail to replicate operations (incomplete protocol design)
Redis Bug #1		
Redis Bug #2		
Redis Bug #3		
Redis Bug #4		

We Found 11 CFBs:
Newly Discovered 5 CFBs in ZooKeeper and 1 CFB in MongoDB
Detected 1 known CFB in MongoDB and 4 known CFBs in Redis

The Size of State Space to Explore is Small Enough for Systematic and Exhaustive Search

DRM	*numOps*	*numReplicas*	# of Schedules
ZooKeeper's DRM	1	3	6
	2	3	80
	3	3	1035
	4	3	13381
	5	3	172993
	3	4	3428
	3	5	54655
Redis's DRM (Suspend)	2	4	13586
Redis's DRM (Link)	2	3	263
Redis's DRM (Crash+Link)	1	2	8
	2	2	96

We could systematically and exhaustively complete state space exploration!

Separating Abstraction from Concrete Execution Makes Modulo Portable and Extensible

DRM	USER/LIB	AEM	CEM	Total
ZooKeeper's DRM	USER	54	59	113
	LIB	339	620	959
MongoDB's DRM	USER	54	117	171
	LIB	339	907	1246
Redis's DRM (Suspend)	USER	33	39	72
	LIB	955	1240	2195
Redis's DRM (Link)	USER	0	110	[illegible]10
	LIB	955	1240	[illegible]195
Redis's DRM (Crash+Link)	USER	405	377	[illegible]2
	LIB	955	1240	2195

Portable (Reused)

Extensible (Library)

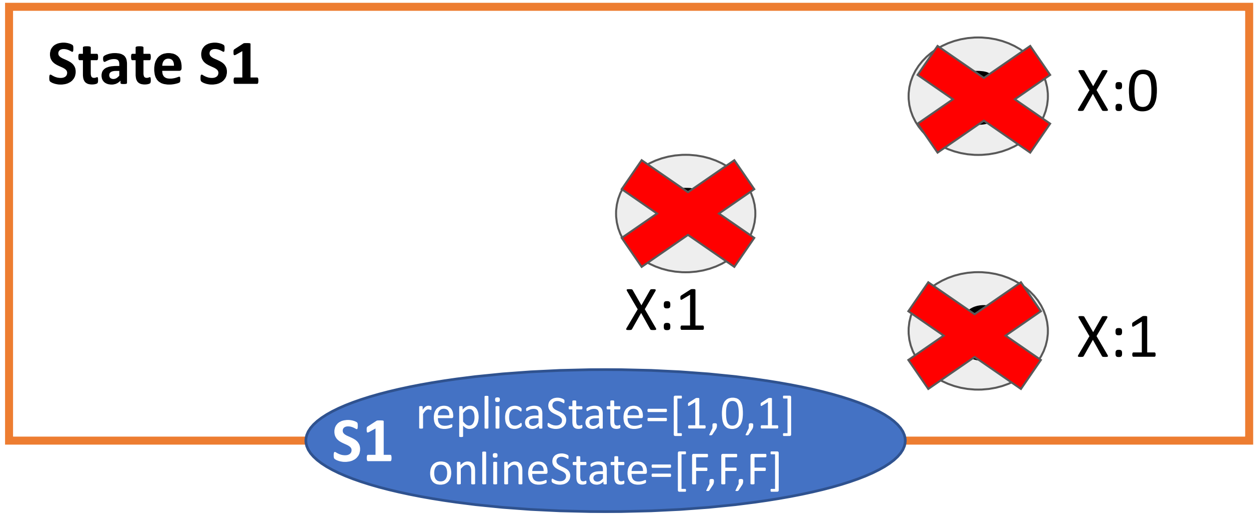
Conclusion

- Modulo is effective in finding bugs in real-world distributed systems
  - Key Approach: Targeted, Systematic and Exhaustive State Space Exploration
  - Key Ideas
    - Exploring only interleaving of divergence and convergence
      - State space to explore is significantly reduced
    - Separating abstraction from concrete execution by decoupling them into AEM and CEM
      - Modulo becomes portable and extensible
- Modulo can be extended to find bugs in your distributed systems!
  - Github: https://github.com/Kaelus/Modulo

Thank You!

Beom Heyn Kim[§†], Taesoo Kim[§‡], and David Lie[†]
[§]Samsung Research, [†]University of Toronto, [‡]Georgia Institute of Technology
{beomheyn.kim, tsgates.kim}@samsung.com, lie@eecg.toronto.edu

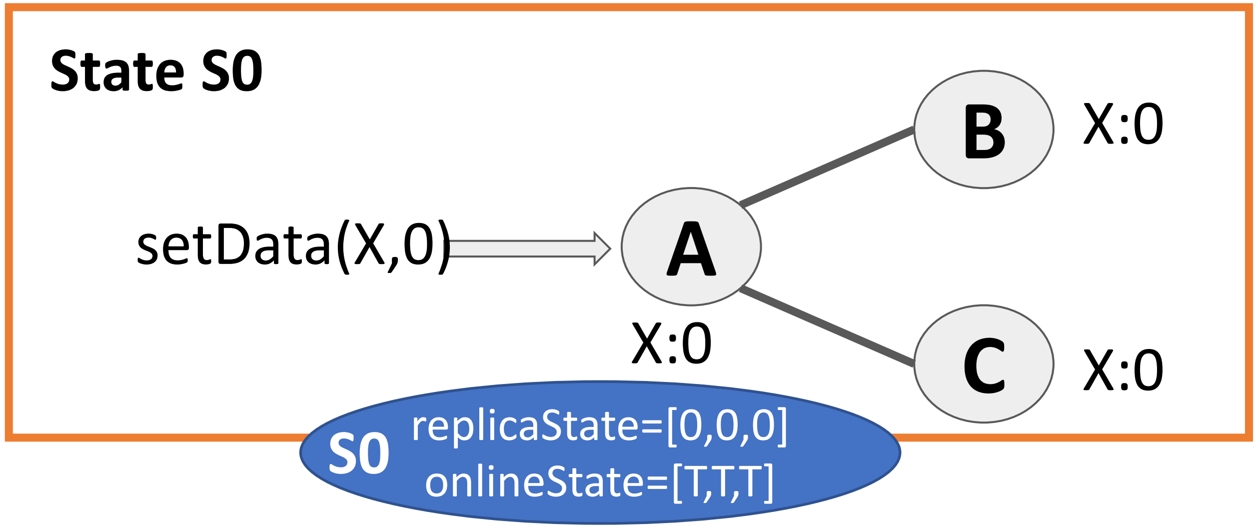
Abstract Execution Model: Enabled Transitions at S1



Enabled Transitions at S1

Divergence	Convergence
	convergence A ⋮ convergence Z

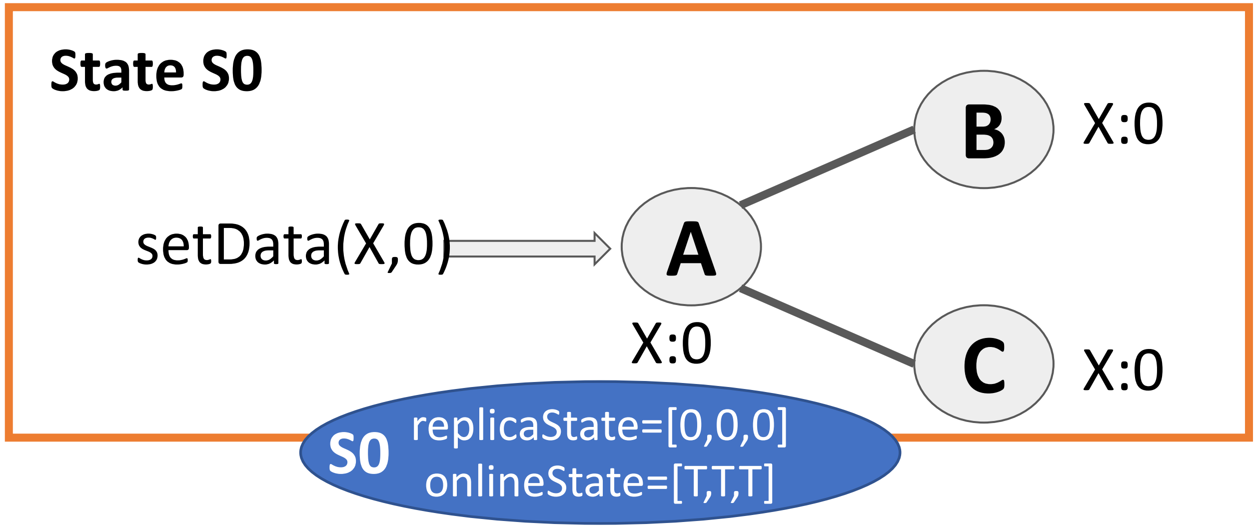
Abstract Execution Model: Enabled Transitions at S0



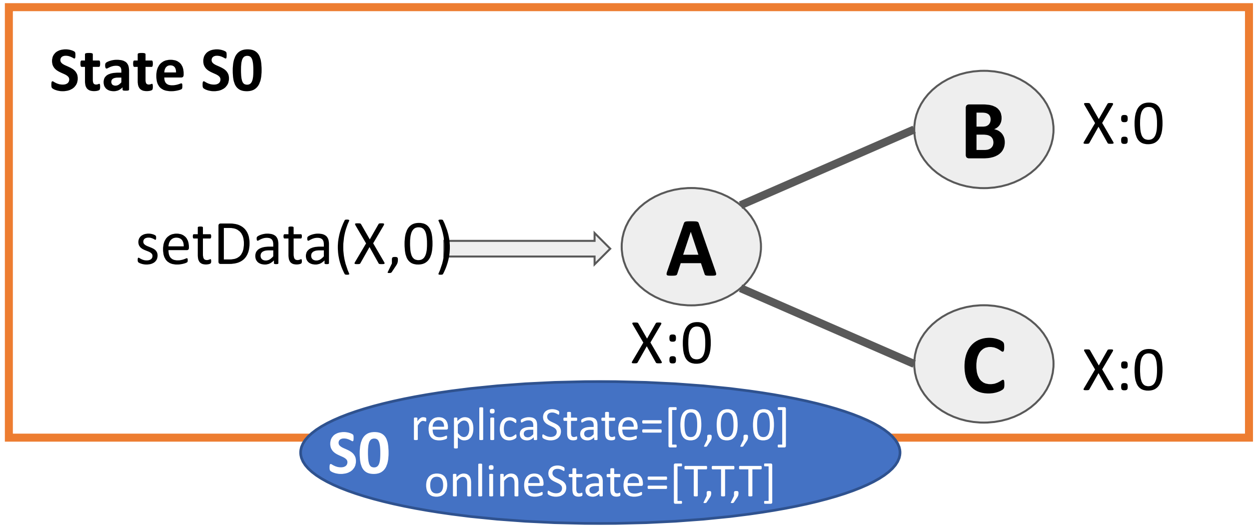
Enabled Transitions at S0

Divergence	Convergence
divergence A ⋮ divergence Z	

Abstract Execution Model: Initial State S0



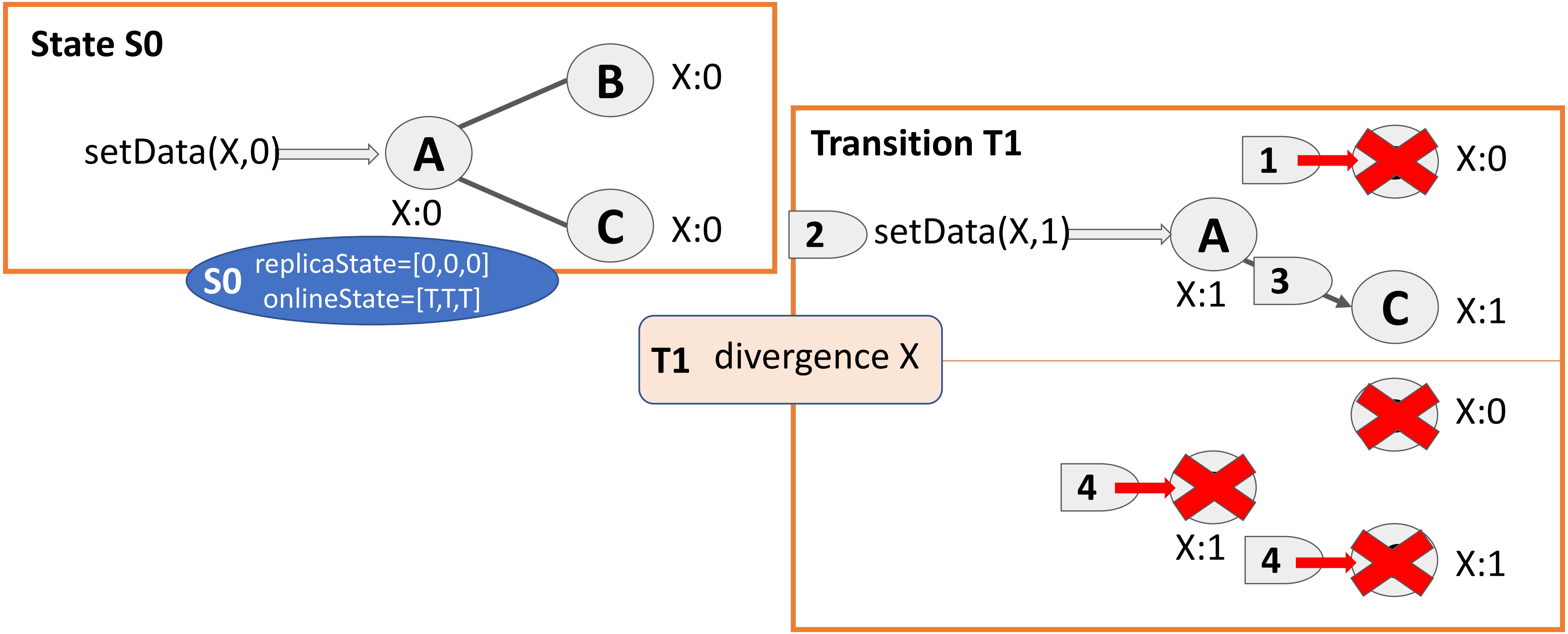
Abstract Execution Model: Enabled Transitions at S0



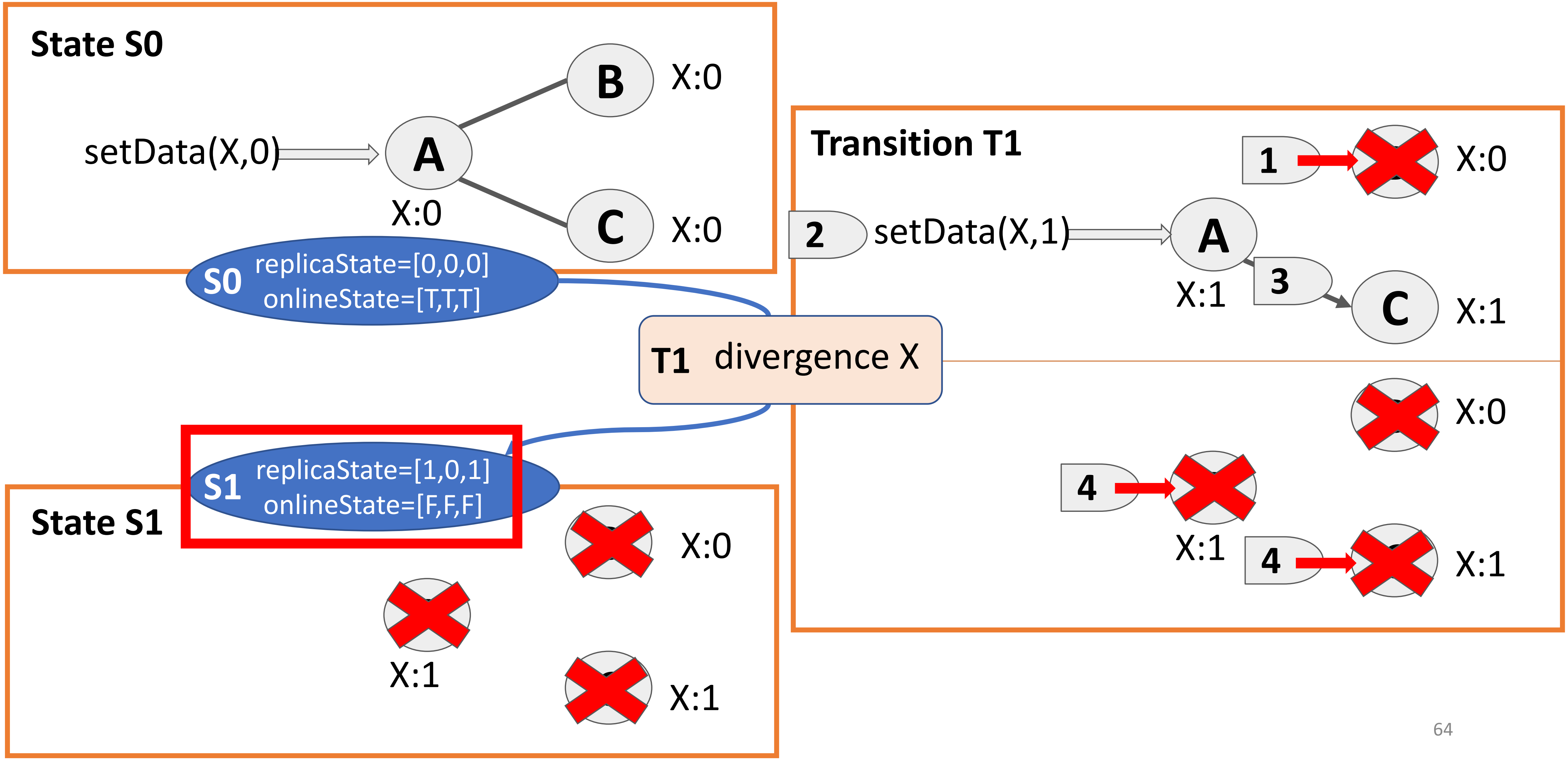
Enabled Transitions at S0

Divergence	Convergence
divergence A ⋮ divergence Z	

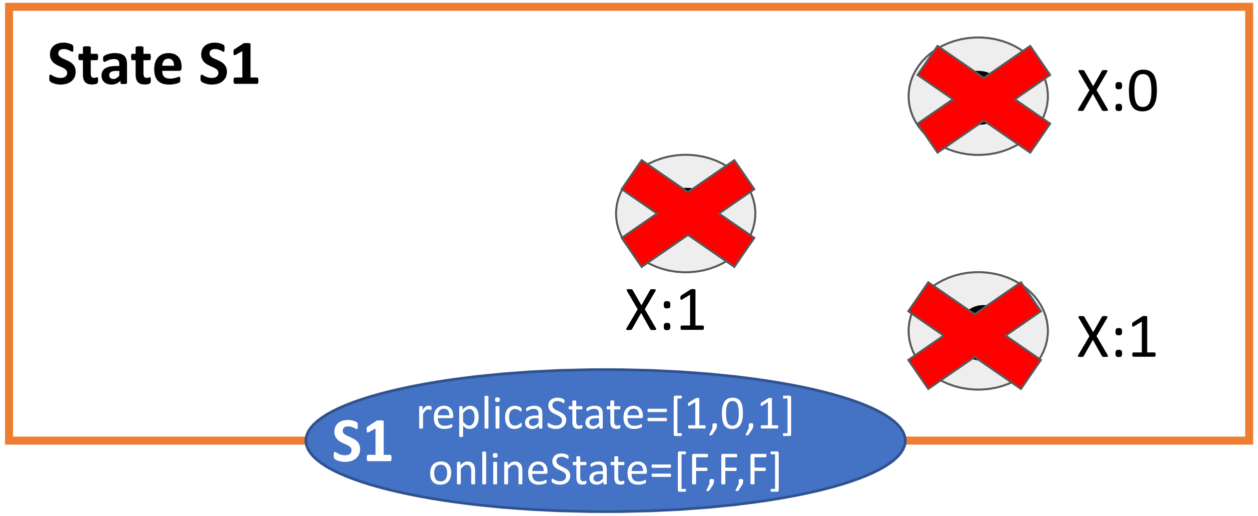
Abstract Execution Model: Applying a Divergence Transition



Abstract Execution Model: Updating State Variables



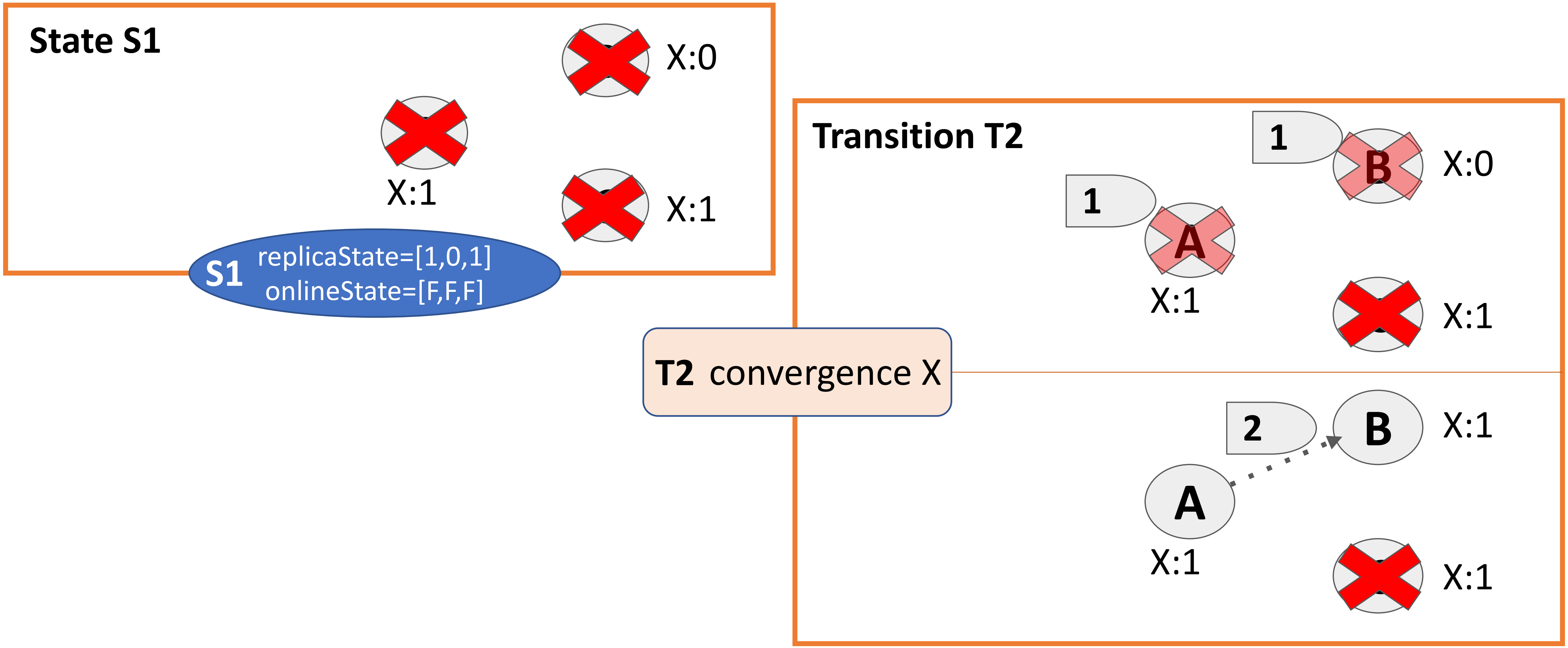
Abstract Execution Model: Enabled Transitions at S1



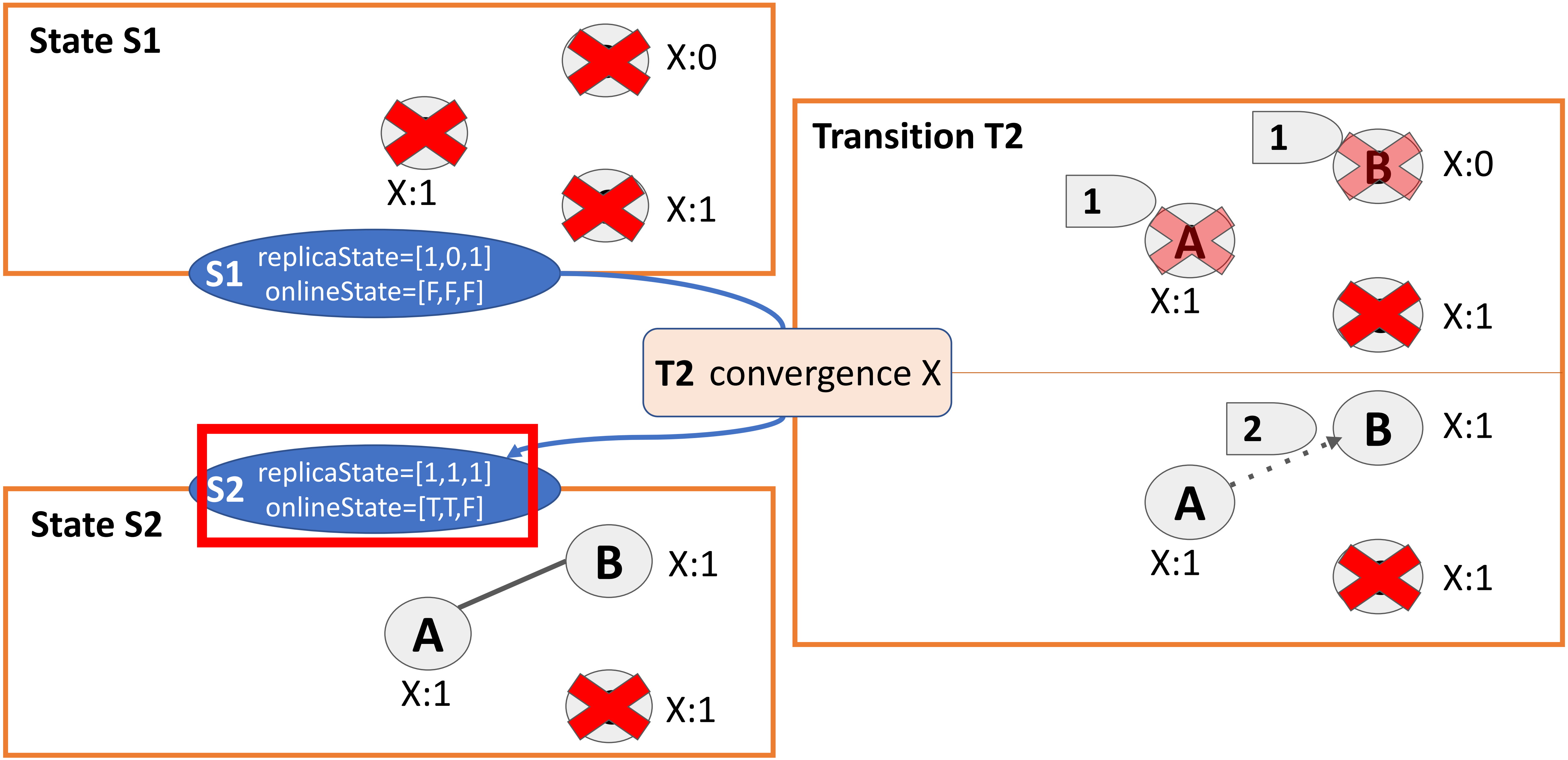
Enabled Transitions at S1

Divergence	Convergence
	convergence A ⋮ convergence Z

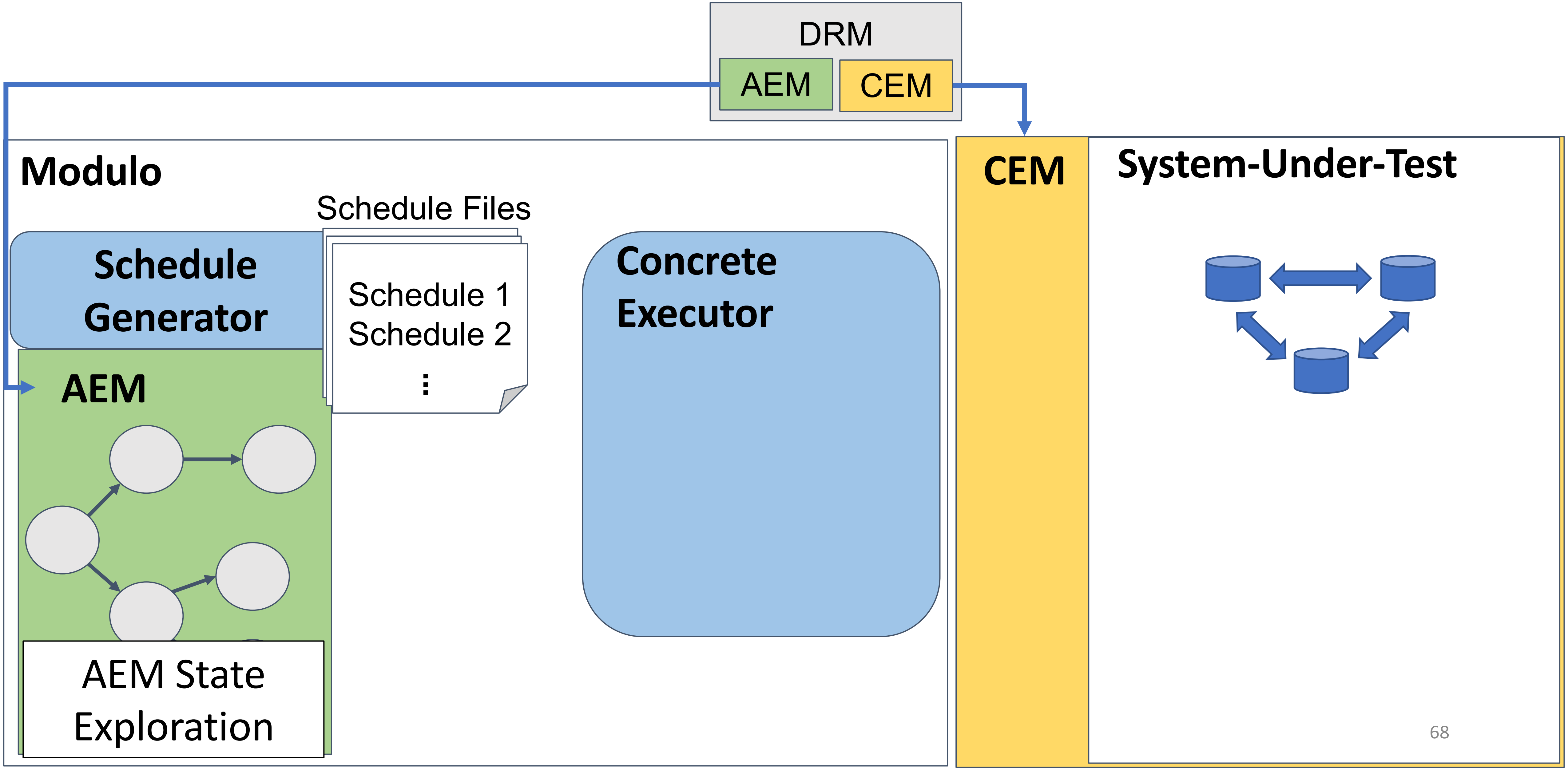
Abstract Execution Model: Applying a Convergence Transition



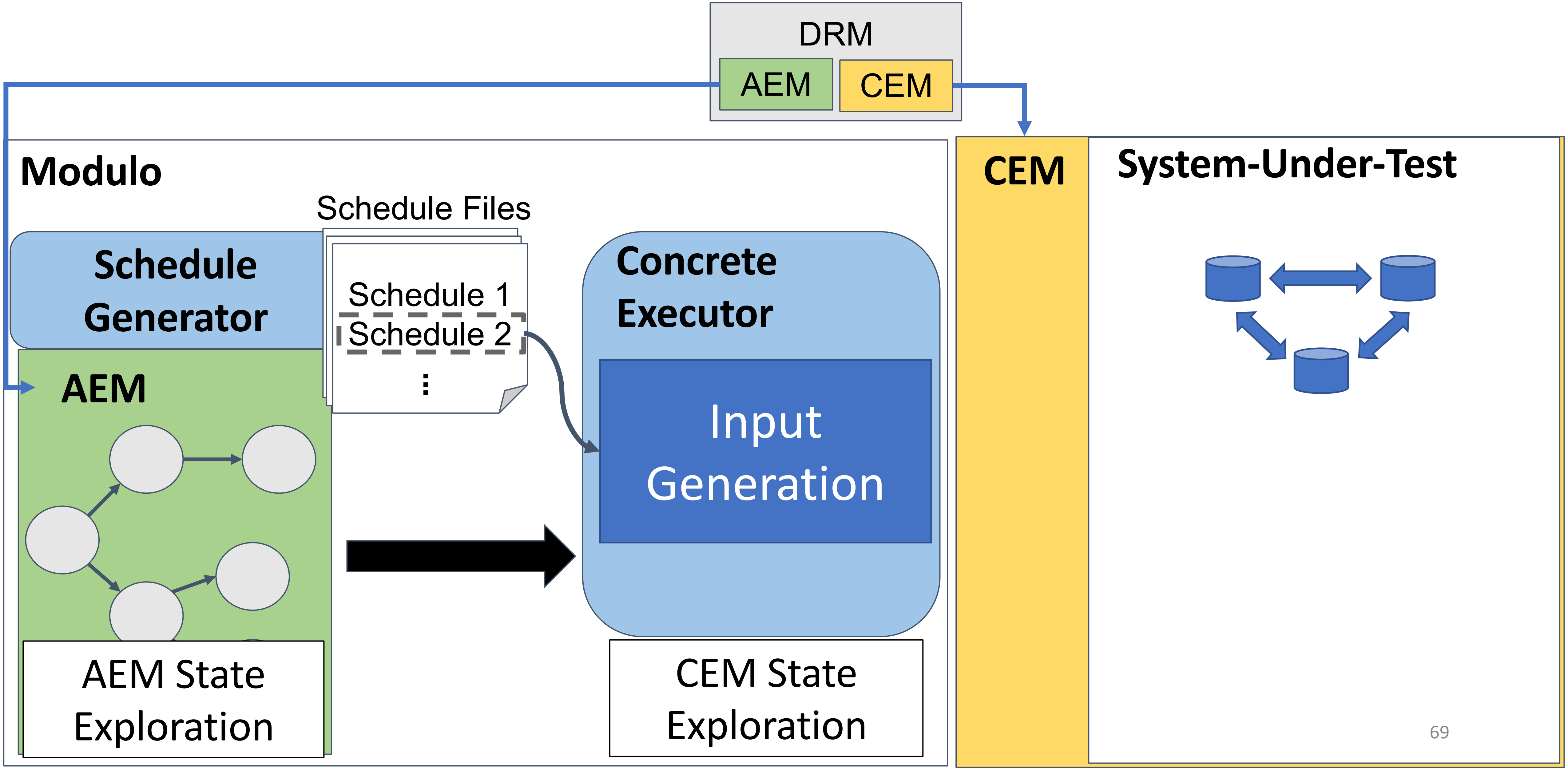
Abstract Execution Model: Updating State Variables



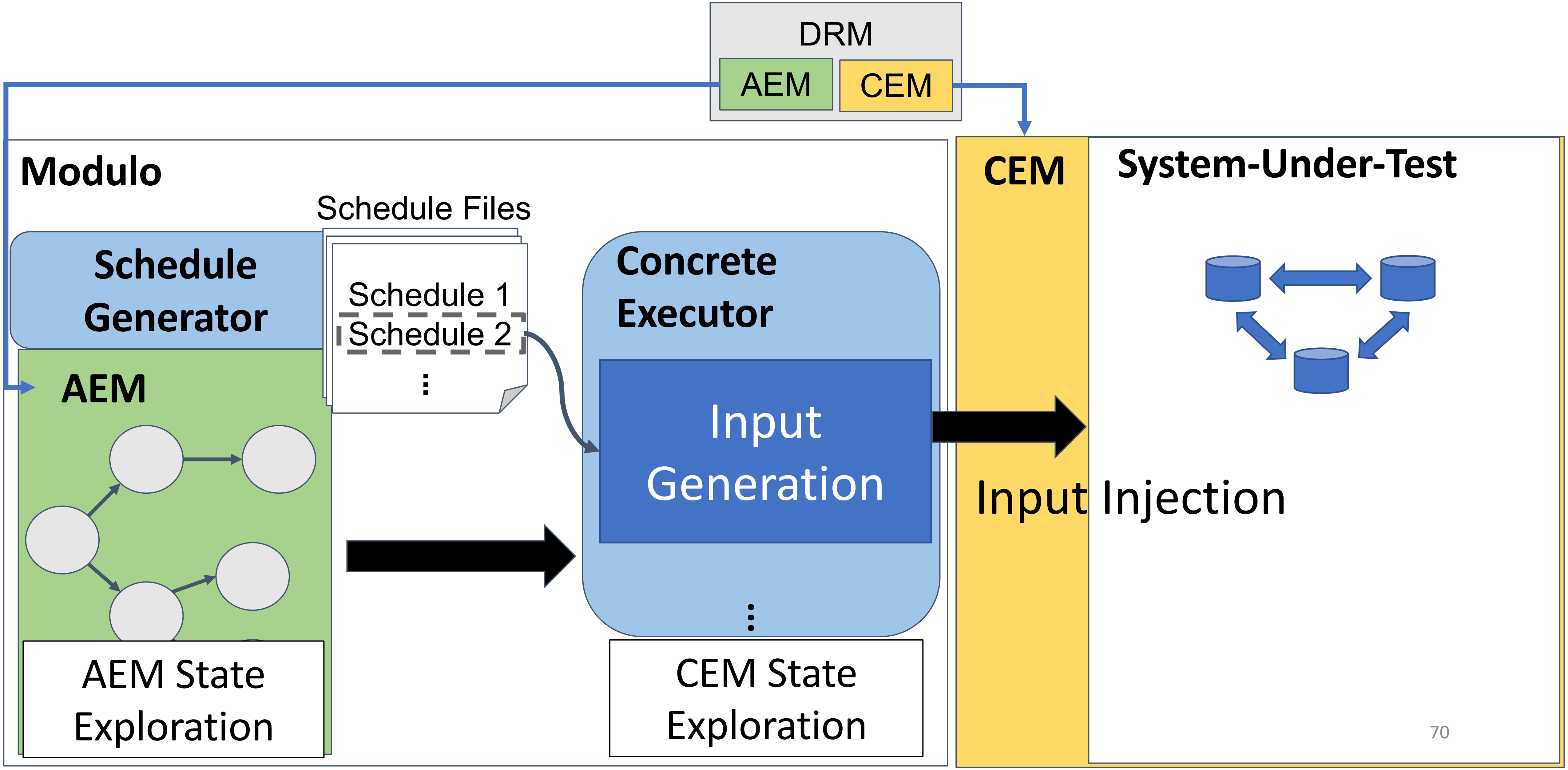
Modulo Architecture: AEM State Exploration



Modulo Architecture CEM Input Generation



Modulo Architecture: CEM Input Injection



Example: ZooKeeper

- Primary-backup replication
- Quorum for a leader election
- The leader serializes every write operation
- Followers replicate the write sequence directly from the leader
- After crash recovery, leader election and resync automatically begin

Example: ZooKeeper's Divergence Resync Model

- AEM
  - Crash failures only
  - Each divergence crashes remaining online replicas at the end
  - Each convergence restarts enough number of replicas to form a quorum

- CEM
  - To Kill: $ kill -9 <A>
  - To Write: setData API call (e.g. setData(x,1))
  - To Restart: java …QuorumPeerMain <A>/zoo.cfg

Implementation: DRM Example Comparison

Name	AEM	CEM	Lines of Code (AEM/CEM/Total)
Q/C/Z-DRM	Only consider crash failures Convergence ensures the quorum Crashes all replicas at the end of divergence	Using kill -9 for crash Confirm the quorum exists before writes Using log scanning before 3.5, but as of 3.5, relying on timeouts	USER 54/59/113 LIB 339/620/959
Q/C/M-DRM	Same as Q/C/Z-DRM	Using an API to compare timestamps of the last transaction on each replica	USER 54/117/171 LIB 339/907/1246
S/S/R-DRM	Only considers suspend failures Considers all replicas initially partitioned As recovering suspend failures, establish links between the replicas	Using kill –STOP and kill –CONT Using 'info' API and timeout to wait for resync completion Using 'slaveof' API to trigger resync	USER 33/39/72 LIB 955/1240/2195
S/L/R-DRM	Only considers link failures Replicas initially connected in a single chain	'slaveof' API for link failures and recoveries. Initially, forming links as a single slave chain	USER 0/110/110 LIB 955/1240/2195
S/CL/R-DRM	Considers both link and crash failures Consider two types of resync strategies: online resync and offline resync	For the offline resync strategy, a script copying over snapshots and starting up a replica with the snapshot is used	USER 405/377/782 LIB 955/1240/2195

Schedule generation is implemented in about 281 lines of code, and concrete execution takes about 766 lines

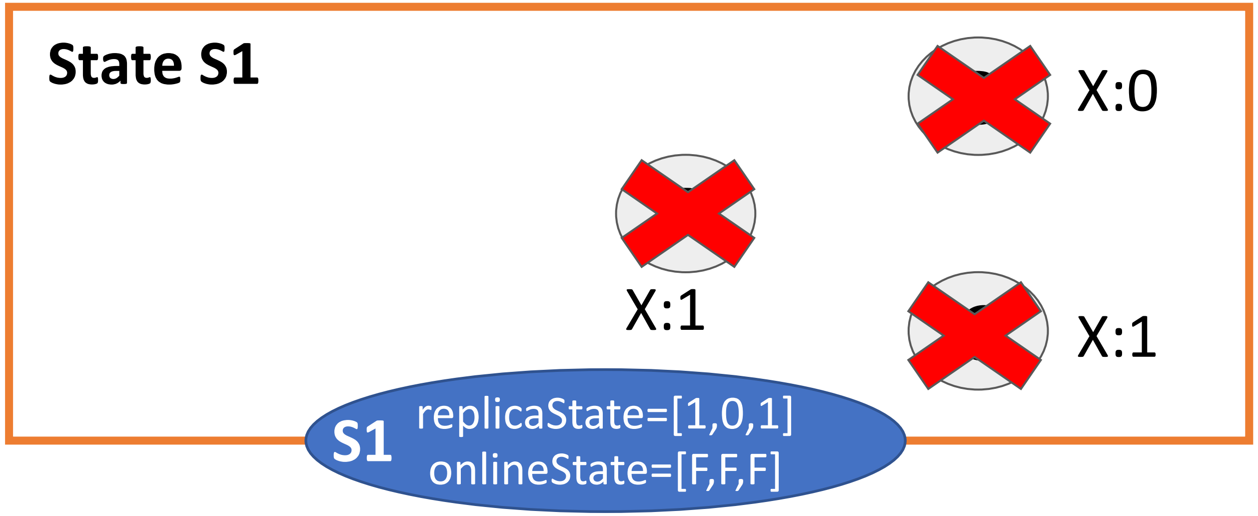
Evaluation: Testing Performance

Bug ID	DRM	Elapsed Time	Time/Schedule	# of Transitions
ZooKeeper Bug #1	Q/C/Z-DRM	11 hours	33 sec	11
ZooKeeper Bug #2	Q/C/Z-DRM	2 hours	39 sec	11
ZooKeeper Bug #3	Q/C/Z-DRM	23 min	33 sec	7
ZooKeeper Bug #4	Q/C/Z-DRM	47 min	30 sec	10
ZooKeeper Bug #5	Q/C/Z-DRM	20 hours	37 sec	10
MongoDB Bug #1	Q/C/M-DRM	18 min	6 min	3
MongoDB Bug #2	Q/C/M-DRM	4 hours	5 min	5
Redis Bug #1	S/S/R-DRM	6 hours	6 min	6
Redis Bug #2	S/CL/R-DRM	11 min	14 sec	4
Redis Bug #3	S/CL/R-DRM	2 min	6 sec	3
Redis Bug #4	S/L/R-DRM	2 min	33 sec	2

Conclusion

- Modulo employs targeted abstraction and concrete execution to mitigate the traditional state-explosion problems.
  - It does not explore states and state transitions that are not related to the concepts of convergence and divergence.

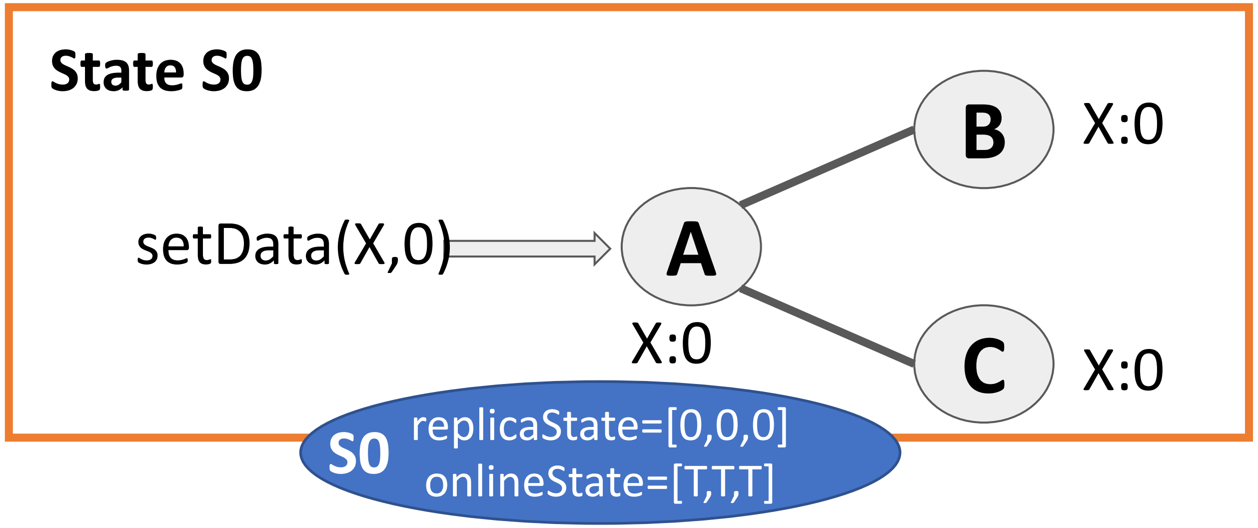
Abstract Execution Model: Picking a Convergence Transition



Enabled Transitions at S1

Divergence	Convergence
	T2 convergence **[A,B]**
	convergence **[B,C]**
	⋮
	convergence **[A,B,C]**

Abstract Execution Model: Picking a Divergence Transition



Enabled Transitions at S0

Divergence	Convergence
divergence A **[0,0,1]**	
divergence **[0,1,1]**	
⋮	
T1 divergence **[1,0,1]**	
⋮	
divergence **[1,1,2]**	
⋮	
divergence **[2,2,2]**	

Divergence Resync Model (DRM): Specifics about the ZooKeeper DRM Example

- ZooKeeper System
  - Primary-backup replication scheme (leader and follower in ZooKeeper’s parlance)
  - Quorum is required to elect a leader
  - The leader serializes every write operation
  - Followers replicate the write sequence directly from the leader
  - After crash recovery, leader election and resync automatically begin
- DRM for ZooKeeper Specifics
  - Crash failures only
  - Each divergence crashes remaining online replicas at the end
  - Each convergence restarts enough number of replicas to form a quorum

Key Observation 1: There Exist Externally Reproducible Convergence Failure Bugs

- **Reproducing Steps:** (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3,v3); (14) Crash C; (15) Restart B; (16) Restart C

Key Observation 1: There Exist Externally Reproducible Convergence Failure Bugs

- **Reproducing Steps:** (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3,v3); (14) Crash C; (15) Restart B; (16) Restart C

It will be more targeted approach to find these bugs if we explore interleaving of relevant events, e.g. Restart, Crash, Put.

Key Observation 1: There Exist Externally Reproducible Convergence Failure Bugs

- **Reproducing Steps:** (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3,v3); (14) Crash C; (15) Restart B; (16) Restart C

It will be more targeted approach to find these bugs if we explore interleaving of relevant events, e.g. Restart, Crash, Put.

Excluding irrelevant events from state exploration

Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

- **Reproduction Step:**
  (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B;
  (5) Restart A; (6) Restart C;
  (7) Put(k2, v2); (8) Crash A; (9) Crash C;
  (10) Restart B; (11) Restart C;
  (12) Crash B; (13) Put(k3,v3); (14) Crash C;
  (15) Restart B; (16) Restart C

Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

• **Reproduction Step:** (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3,v3); (14) Crash C; (15) Restart B; (16) Restart C	• **Divergence and Convergence:** (1) Divergence [0,1,0]; (2) Convergence [A,C]; (3) Divergence [1,0,1]; (4) Convergence [B,C]; (5) Divergence [0,0,1]; (6) Convergence [B,C];

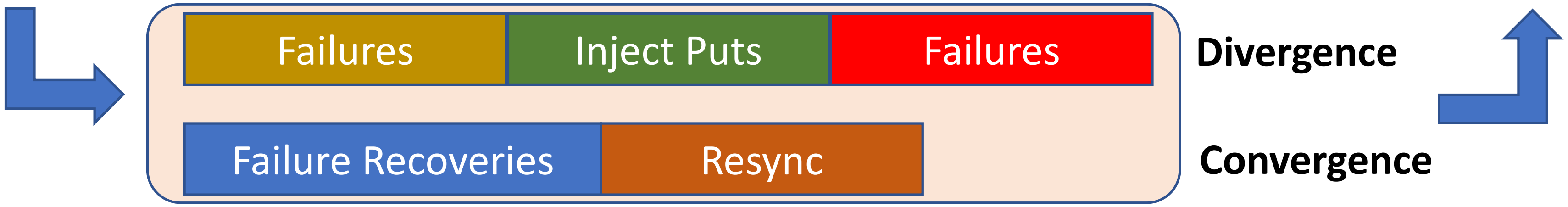
Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

- **Reproduction Step:**

  (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B;
  (5) Restart A; (6) Restart C;
  (7) Put(k2, v2); (8) Crash A; (9) Crash C;
  (10) Restart B; (11) Restart C;
  (12) Crash B; (13) Put(k3,v3); (14) Crash C;
  (15) Restart B; (16) Restart C

- **Divergence and Convergence:**

  (1) Divergence [0,1,0];
  (2) Convergence [A,C];
  (3) Divergence [1,0,1];
  (4) Convergence [B,C];
  (5) Divergence [0,0,1];
  (6) Convergence [B,C];

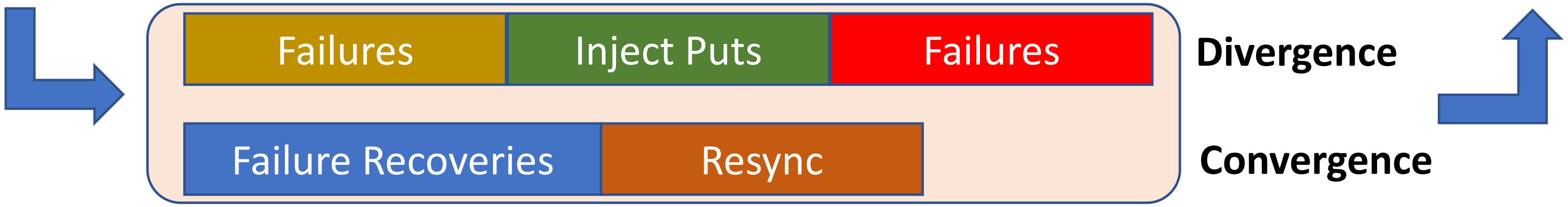


Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

- **Reproduction Step:**
  (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B;
  (5) Restart A; (6) Restart C;
  (7) Put(k2, v2); (8) Crash A; (9) Crash C;
  (10) Restart B; (11) Restart C;
  (12) Crash B; (13) Put(k3,v3); (14) Crash C;
  (15) Restart B; (16) Restart C

- **Divergence and Convergence:**
  (1) Divergence [0,1,0];
  (2) Convergence [A,C];
  (3) Divergence [1,0,1];
  (4) Convergence [B,C];
  (5) Divergence [0,0,1];
  (6) Convergence [B,C];

We can reduce a sequence of low level events into a sequence of higher level divergence and convergence events.



Related Works: Exhaustive State Search Suffers from State Explosion

- Model-based testing (OAuthTester, MBTC) and model-checking (PACE, CMC, Verisoft, MaceMC, MODIST, CrystalBall, dBug, SAMC, FlyMC, etc.): employing state-space exploration to systematically check for the absence of bugs
  - Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion

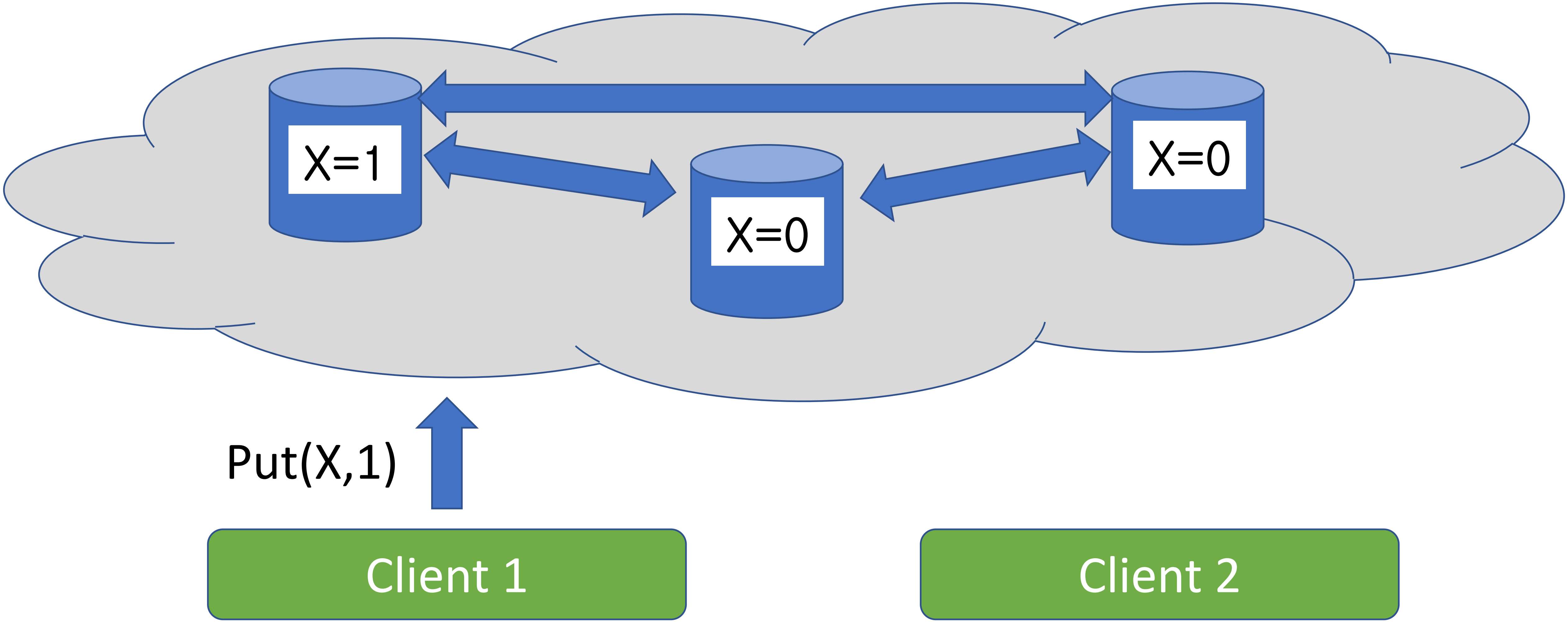
Related Works: Non-Systematic State Search May Miss Bugs

- Model-based testing (OAuthTester, MBTC) and model-checking (PACE, CMC, Verisoft, MaceMC, MODIST, CrystalBall, dBug, SAMC, FlyMC, etc.): employing state-space exploration to systematically check for the absence of bugs
  - Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion
- Manual testing and random testing (Jepsen): Scope of testing is usually targeted to find specific types of bugs

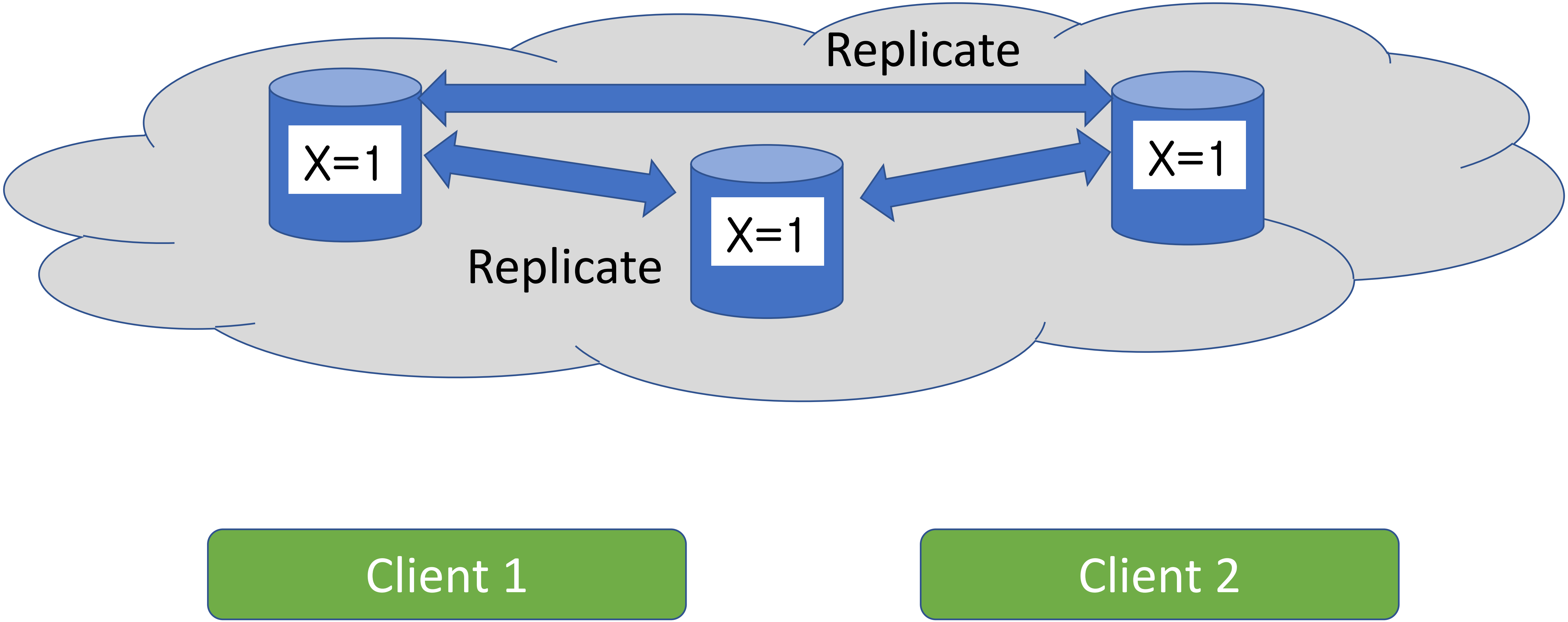
Related Works: Non-Systematic State Search May Miss Bugs

- Model-based testing (OAuthTester, MBTC) and model-checking (PACE, CMC, Verisoft, MaceMC, MODIST, CrystalBall, dBug, SAMC, FlyMC, etc.): employing state-space exploration to systematically check for the absence of bugs
    - Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion
- Manual testing and random testing (Jepsen): Scope of testing is usually targeted to find specific types of bugs
    - Limitation: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases

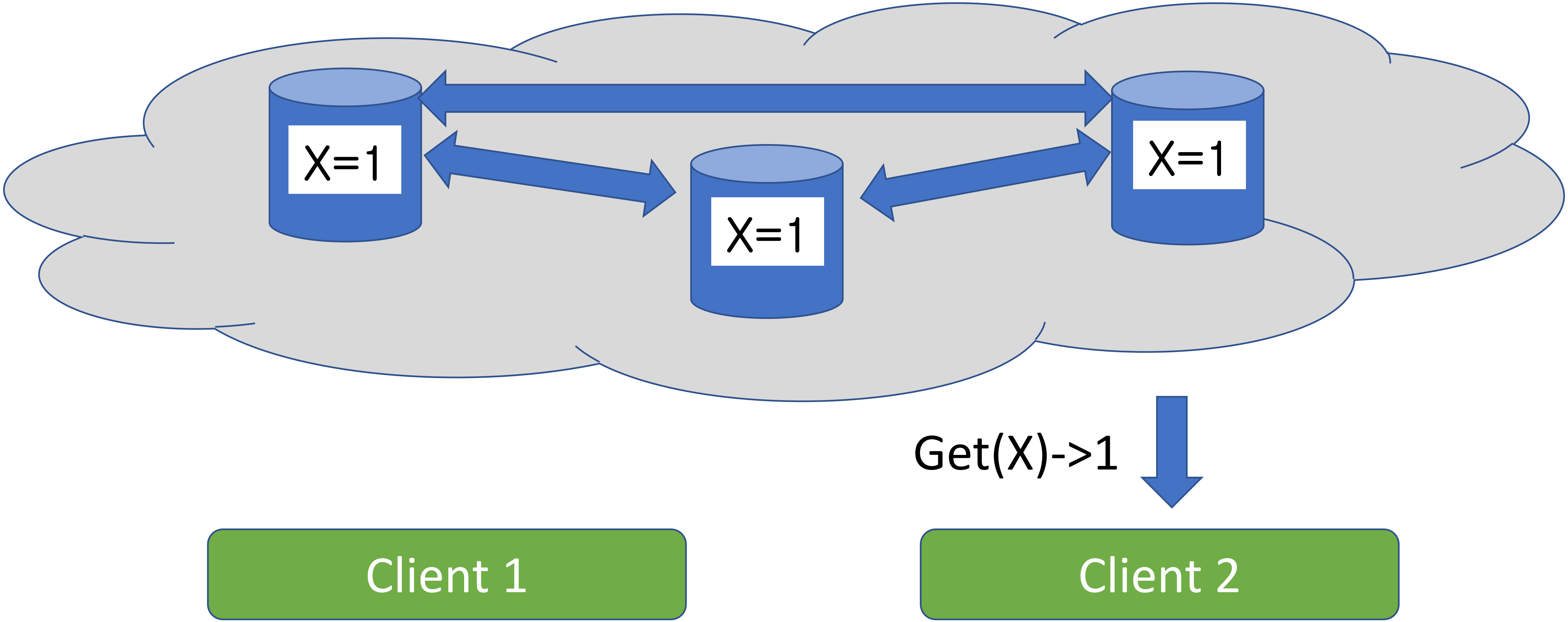
Background: Convergence Property Keeps Replicas Consistent



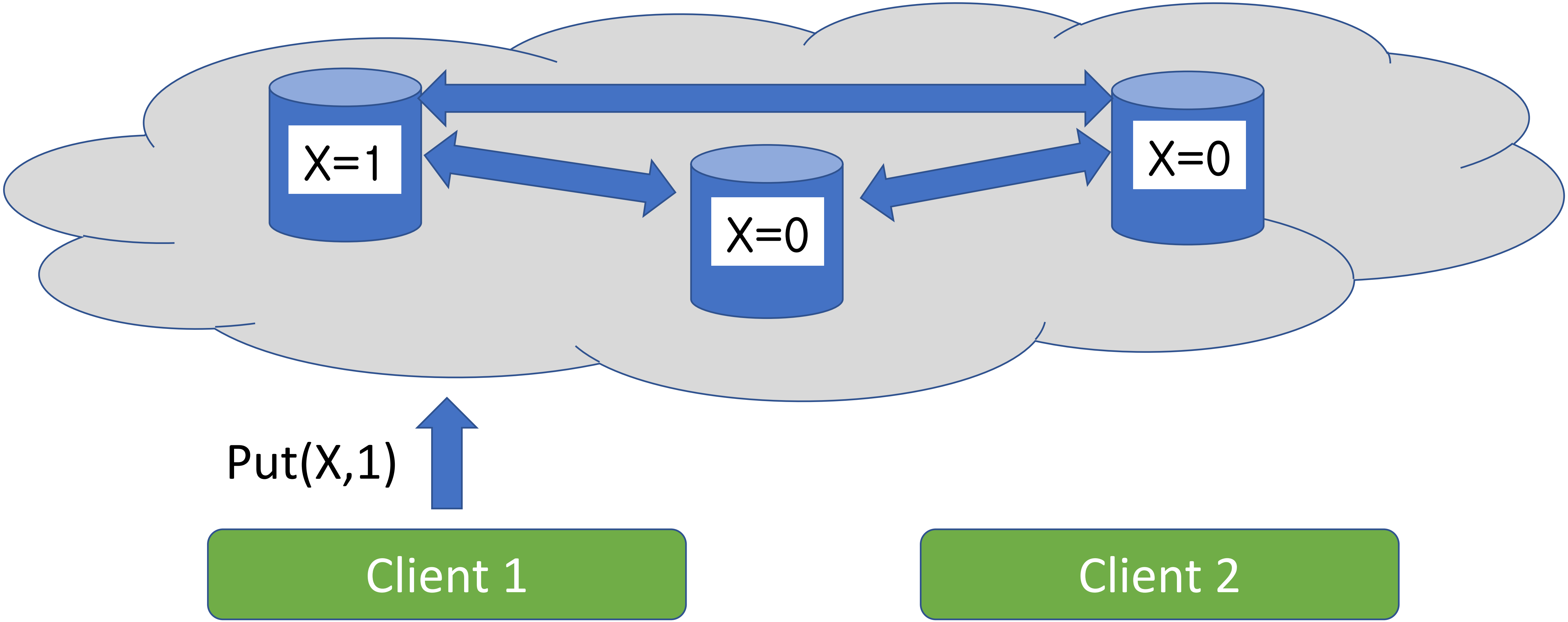
Background: Convergence Property Keeps Replicas Consistent



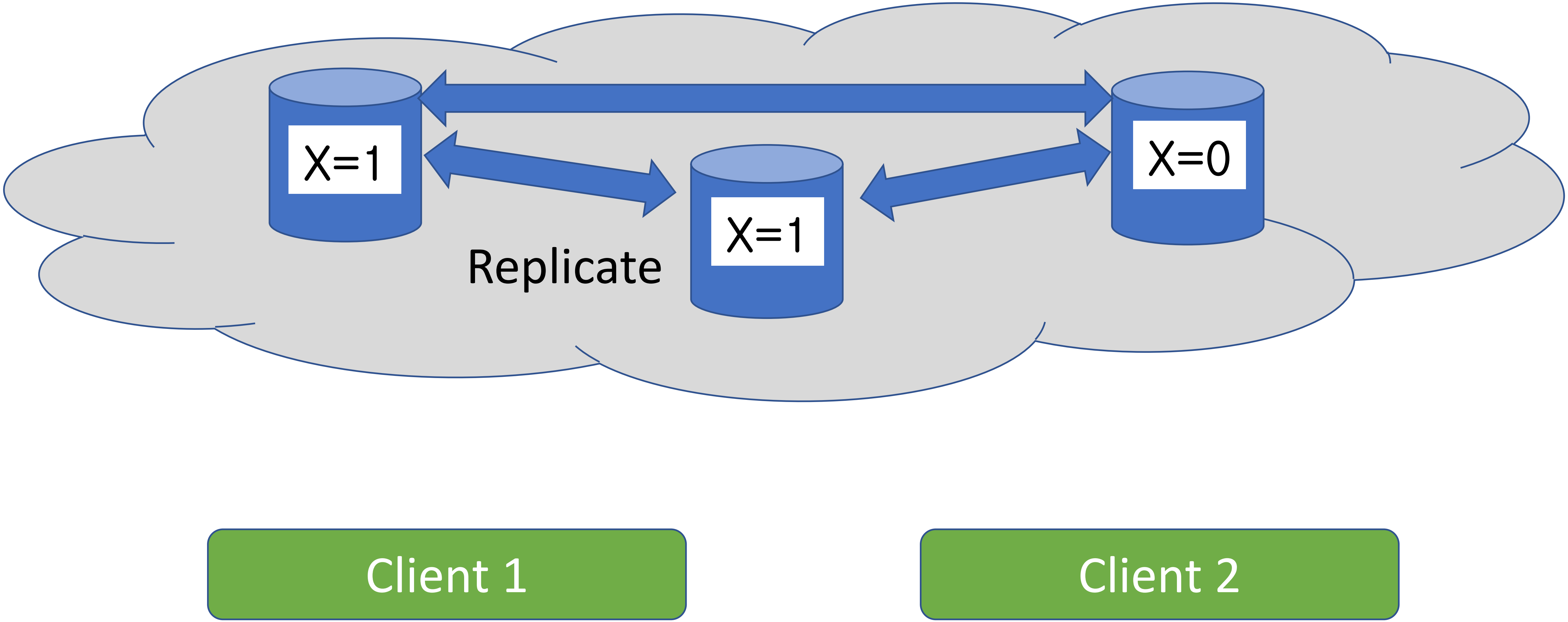
Background: Convergence Property Keeps Replicas Consistent



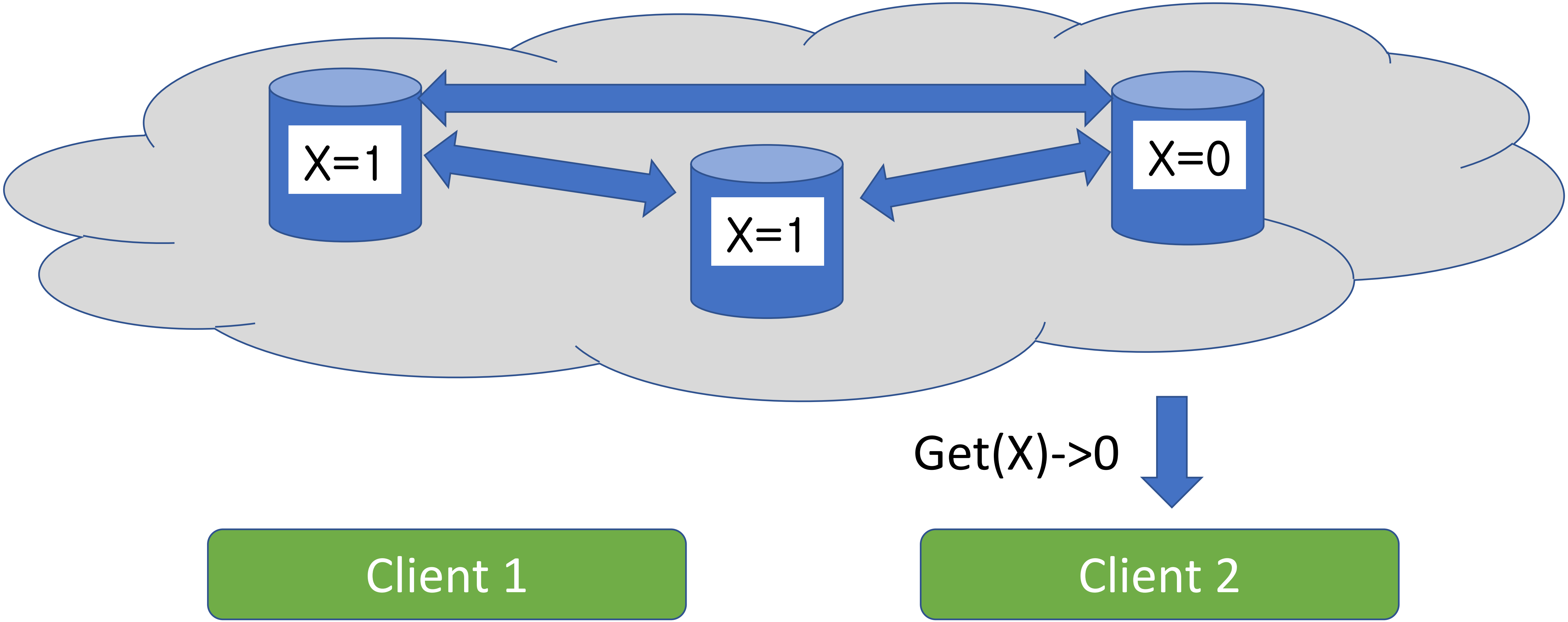
Background: Divergence Can Be Observed by Clients



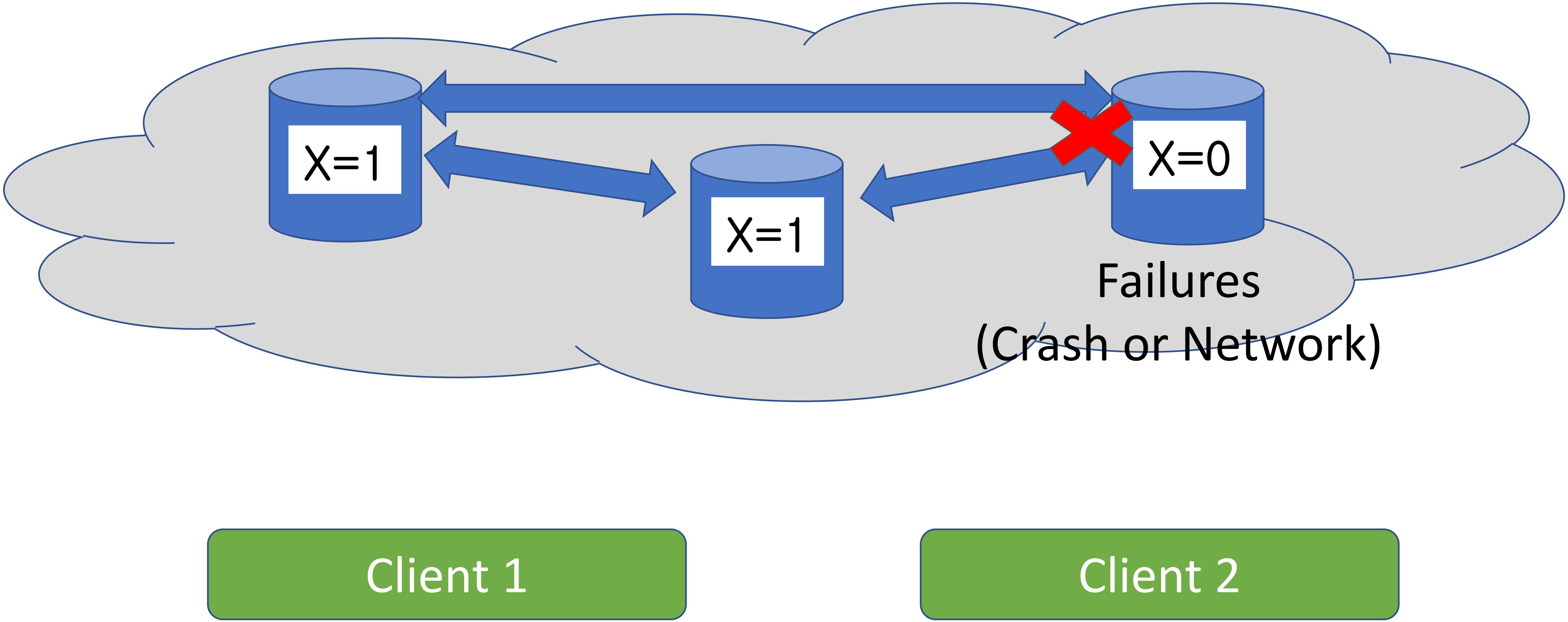
Background: Divergence Can Be Observed by Clients



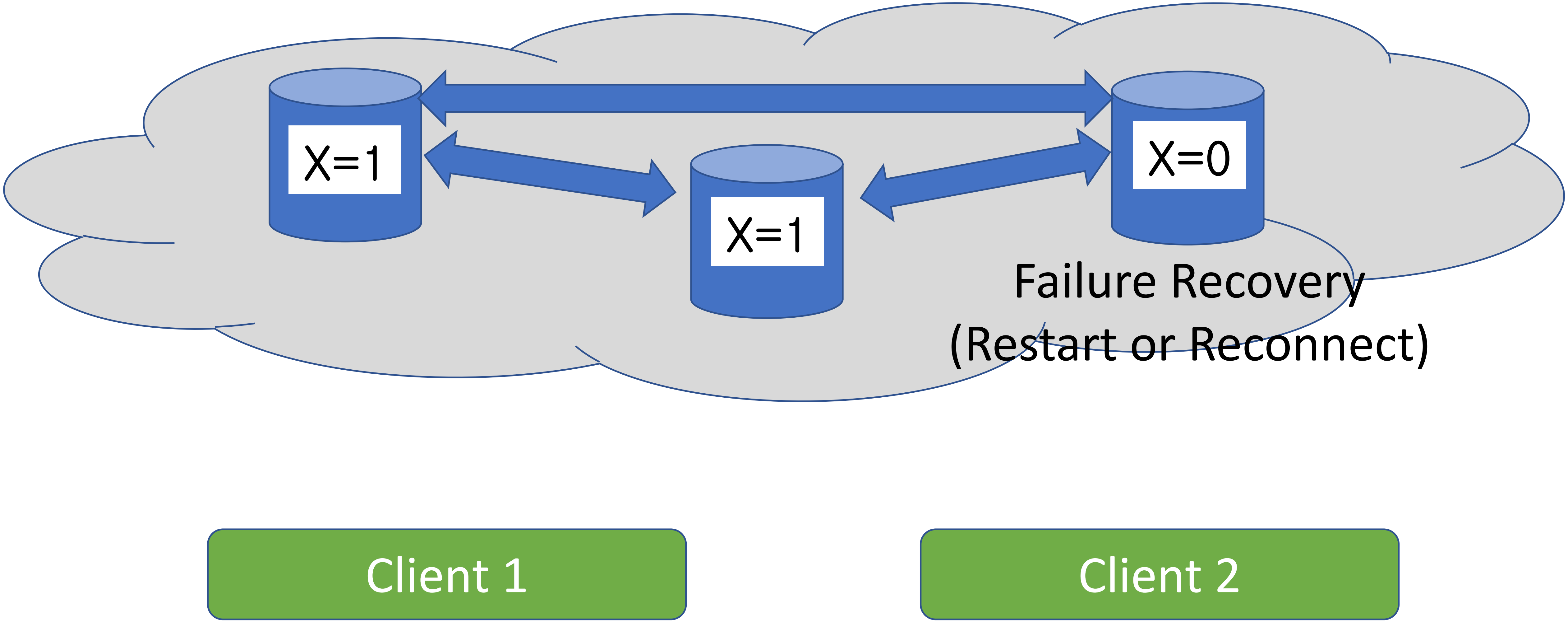
Background: Divergence Can Be Observed by Clients



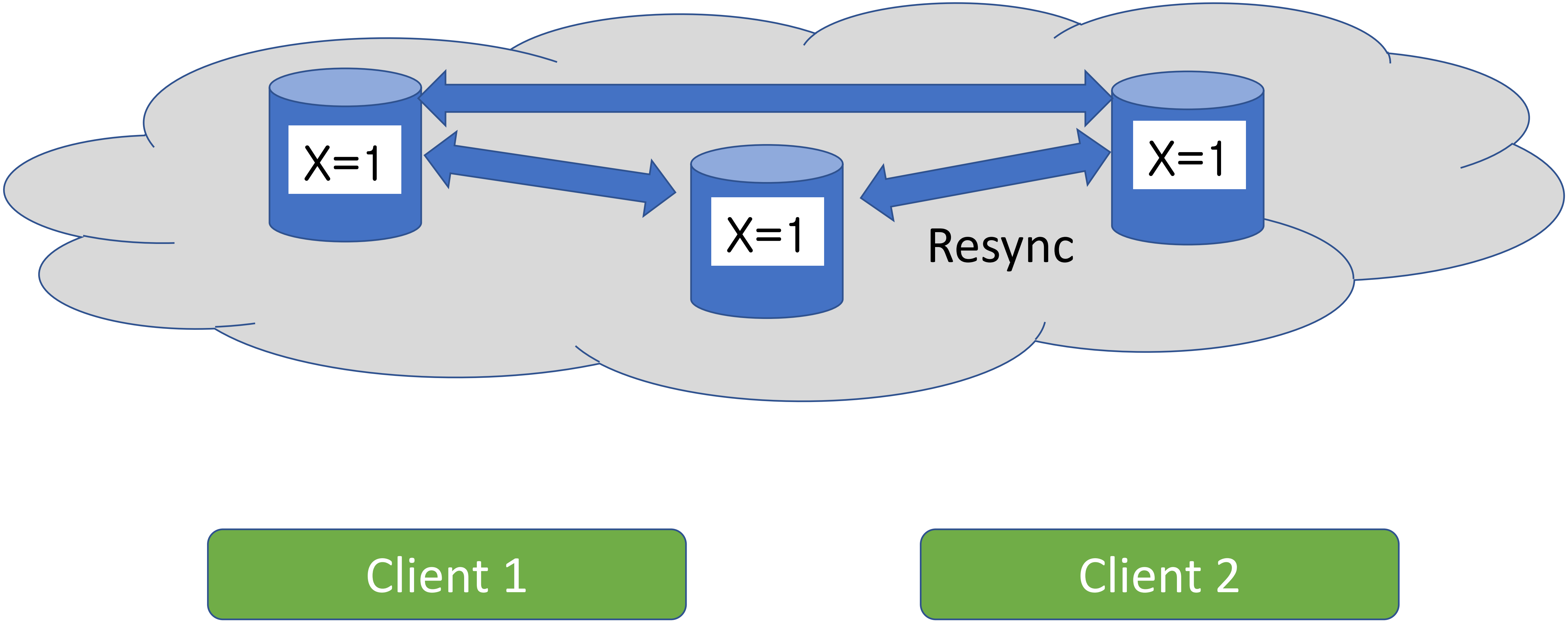
Background: Failures Extends Divergence's Lifetime Until Recovery and Resync



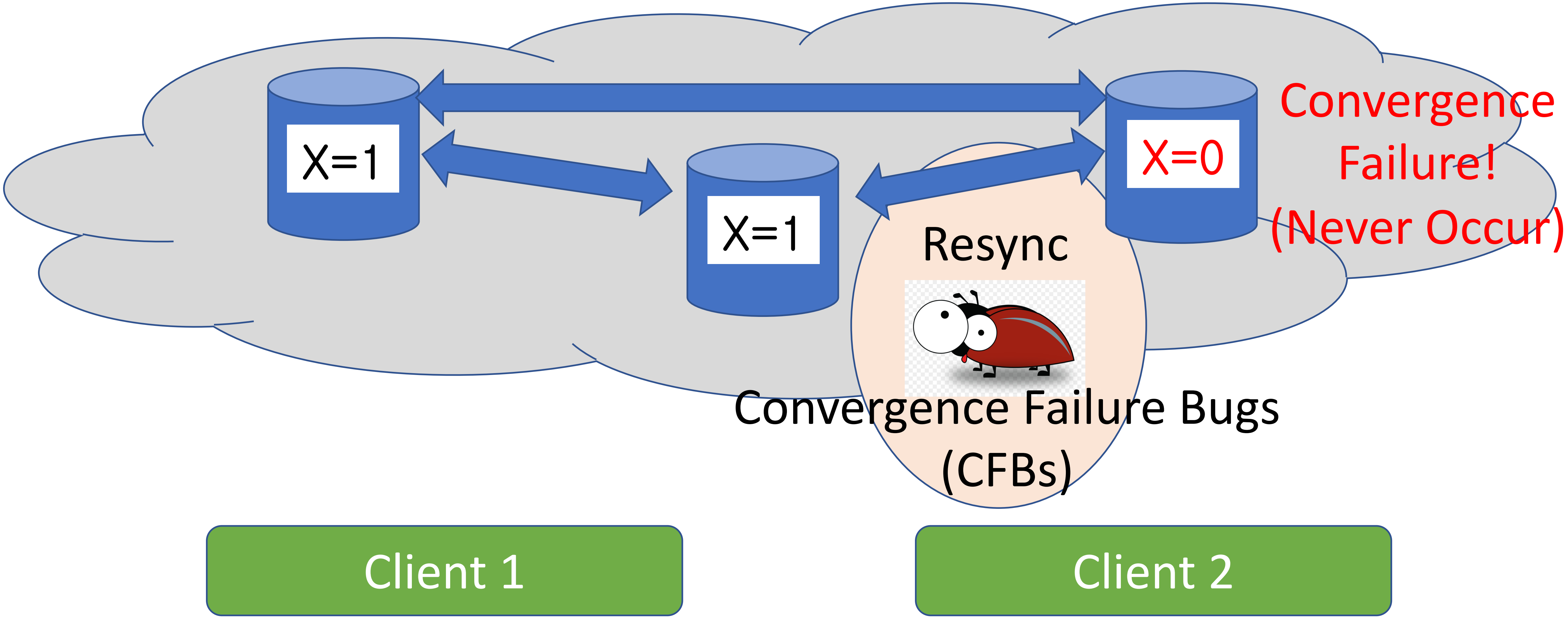
Background: Recovery and Resync Reduces Divergence and Restores Convergence



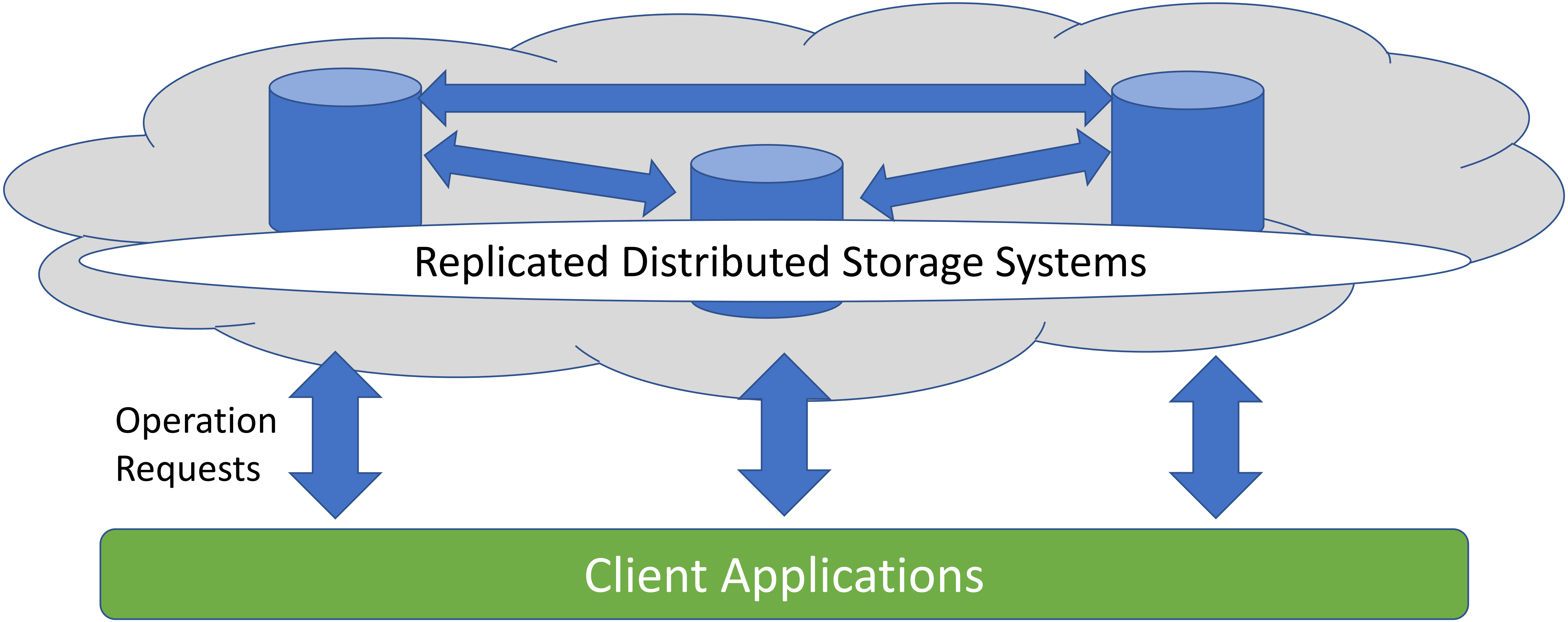
Background: Recovery and Resync Removes Divergence and Restores Convergence



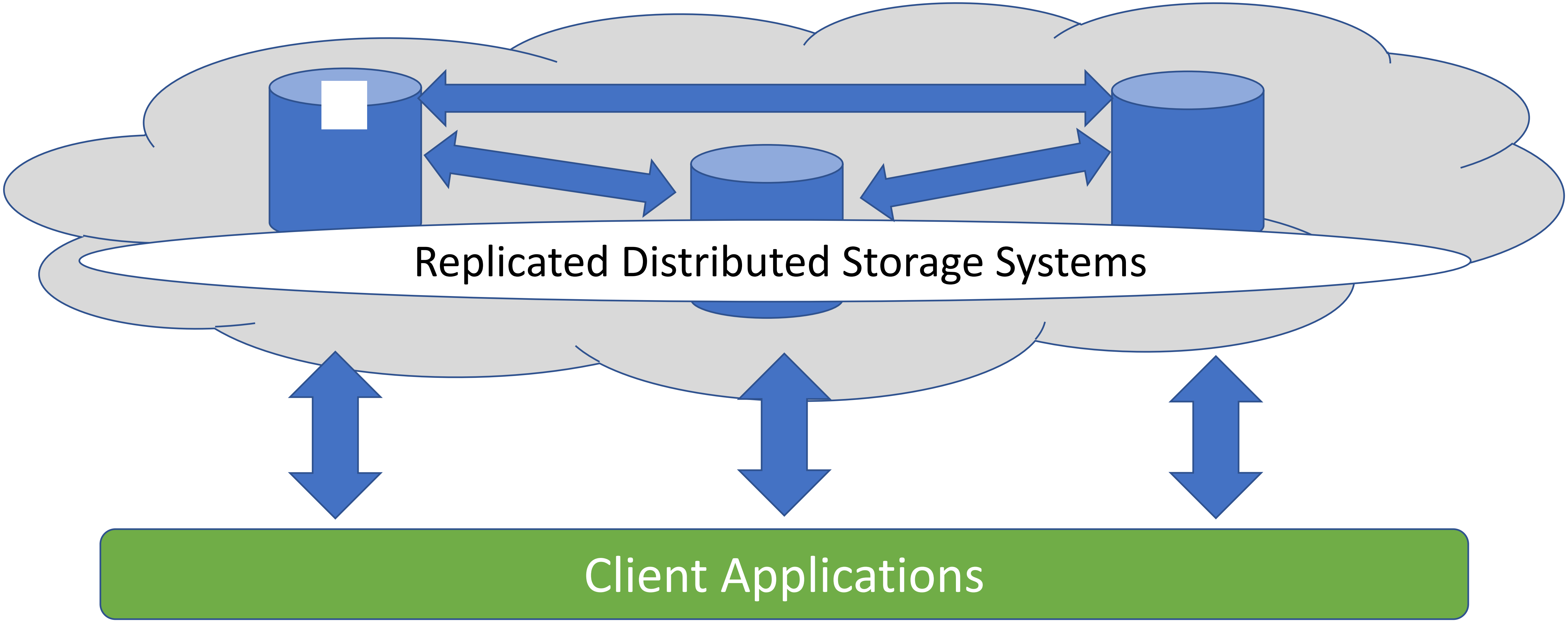
Background: Software Bugs in Resync Mechanisms May Cause Convergence Failures



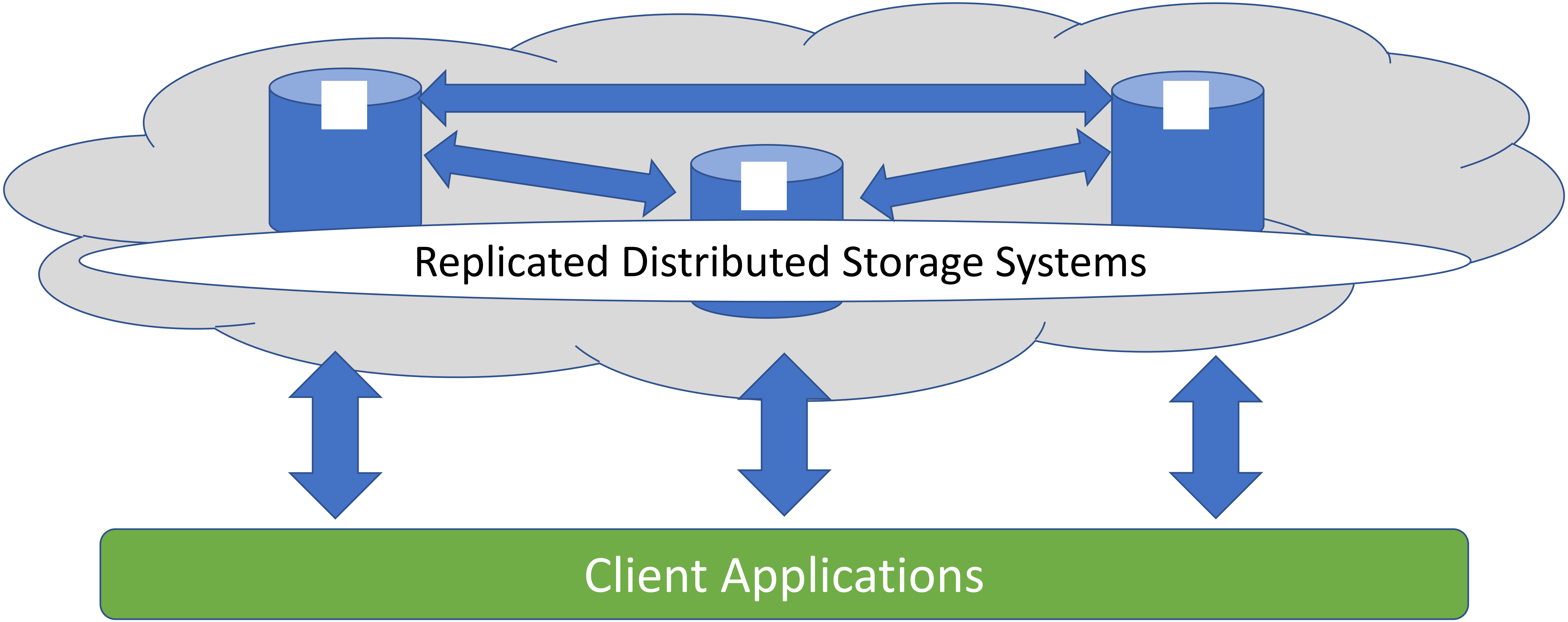
Divergence and Convergence



Divergence and Convergence

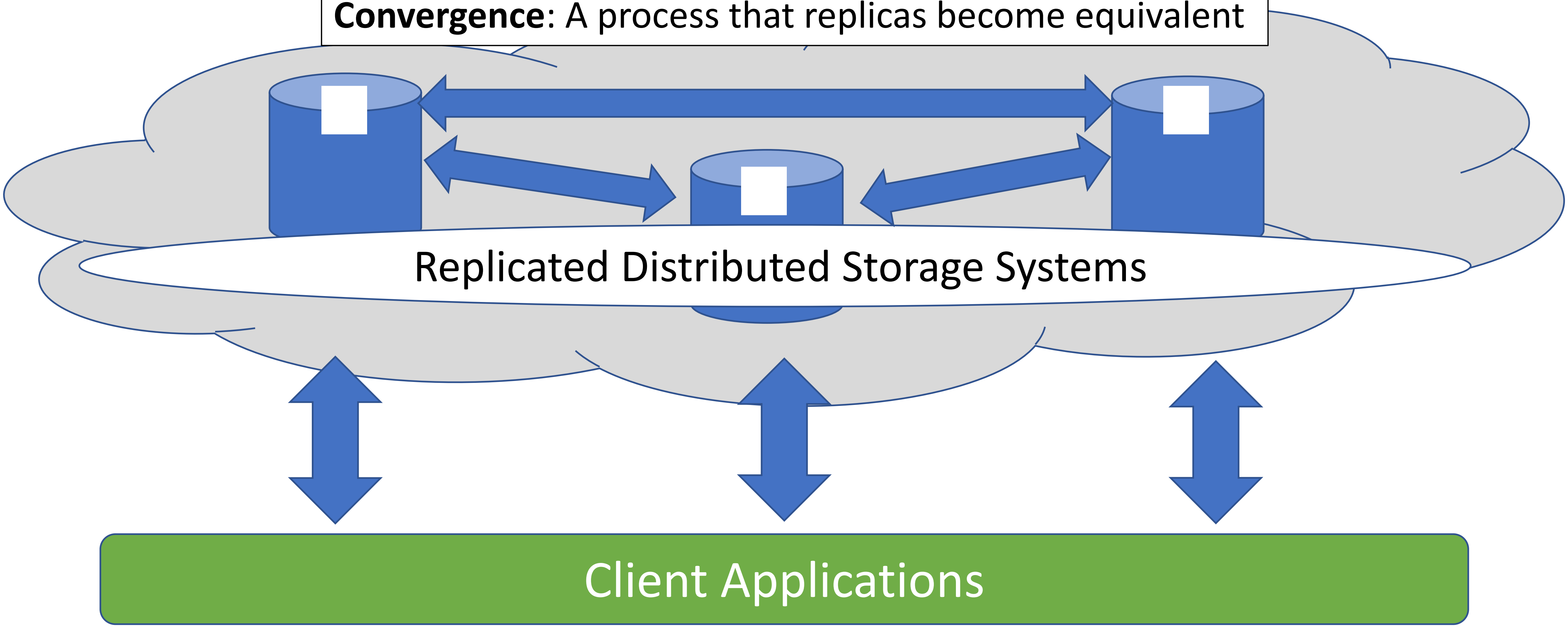


Divergence and Convergence

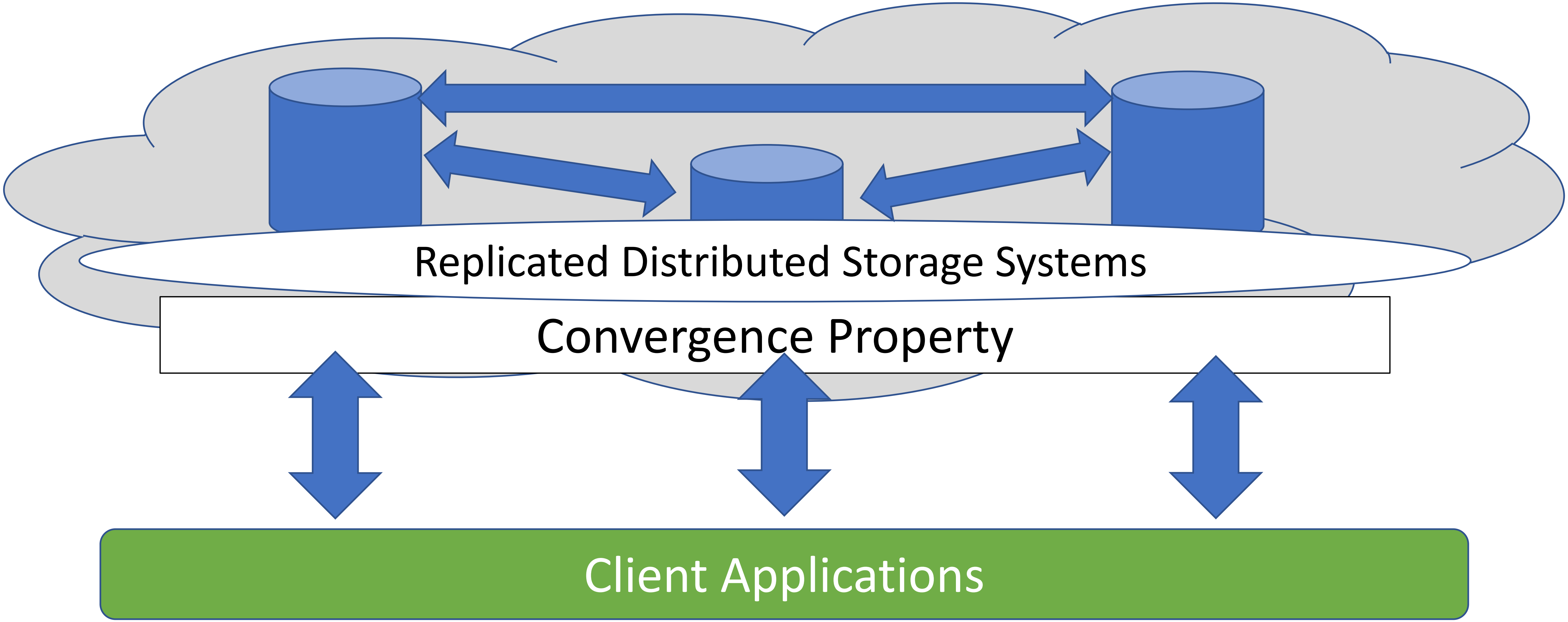


Divergence and Convergence

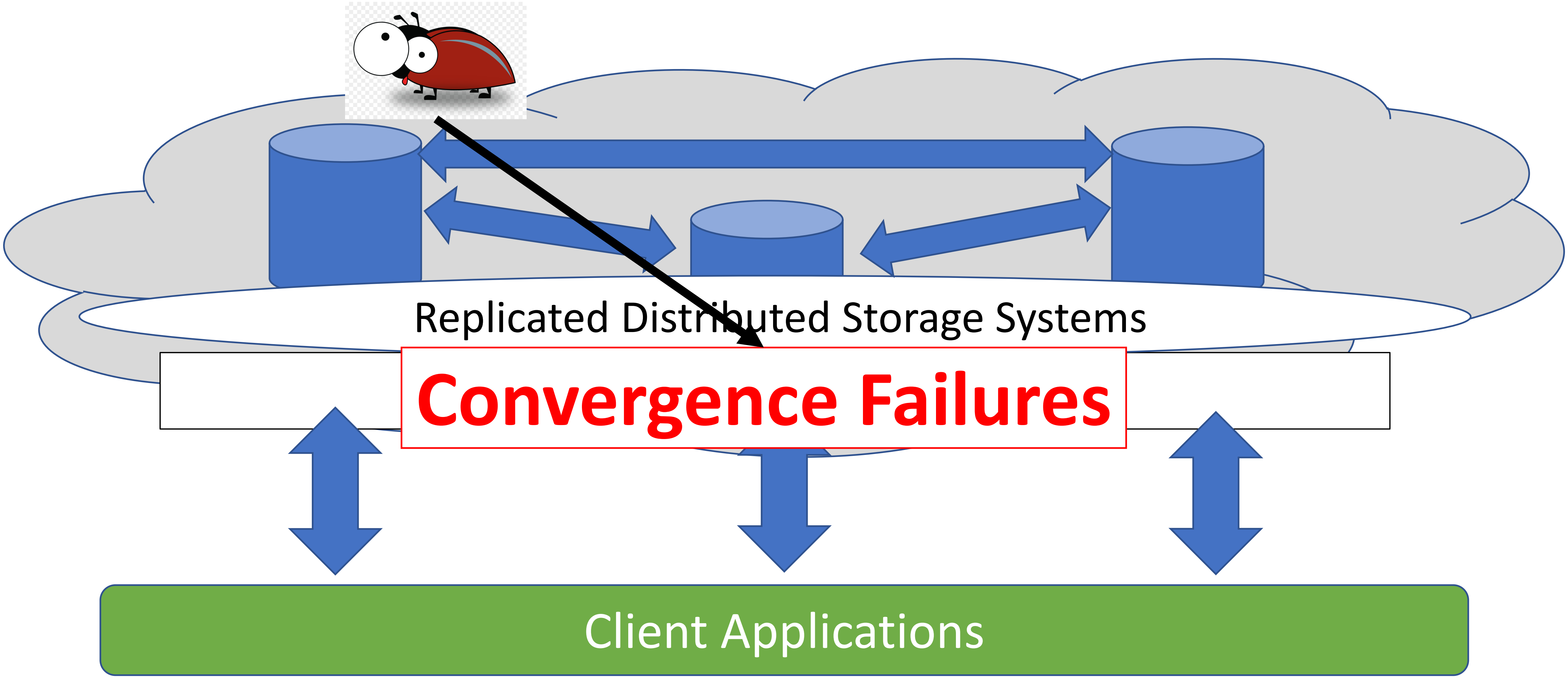
Divergence: A process that replicas become different
Convergence: A process that replicas become equivalent



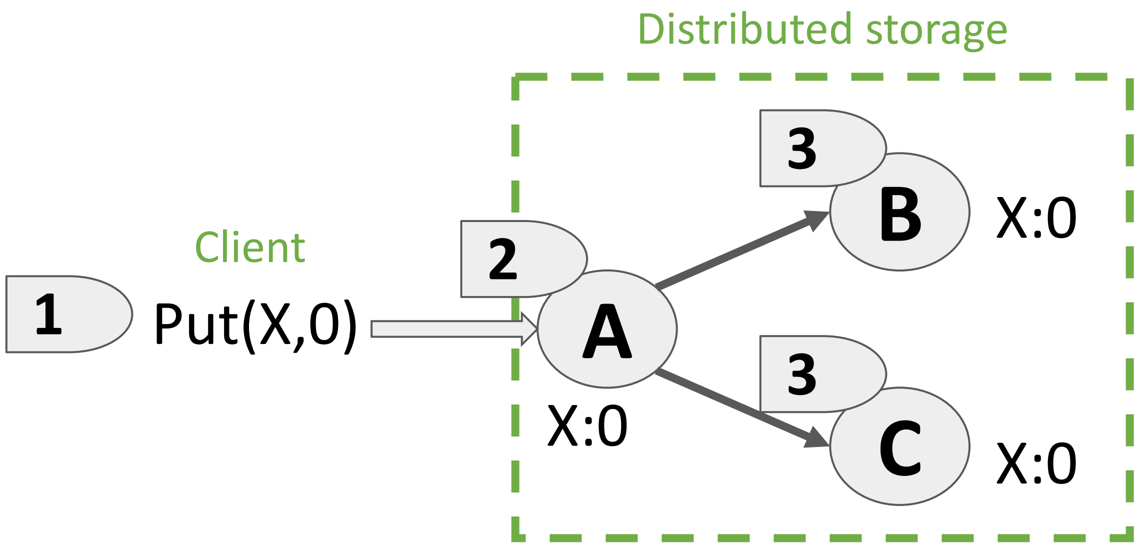
Convergence Property



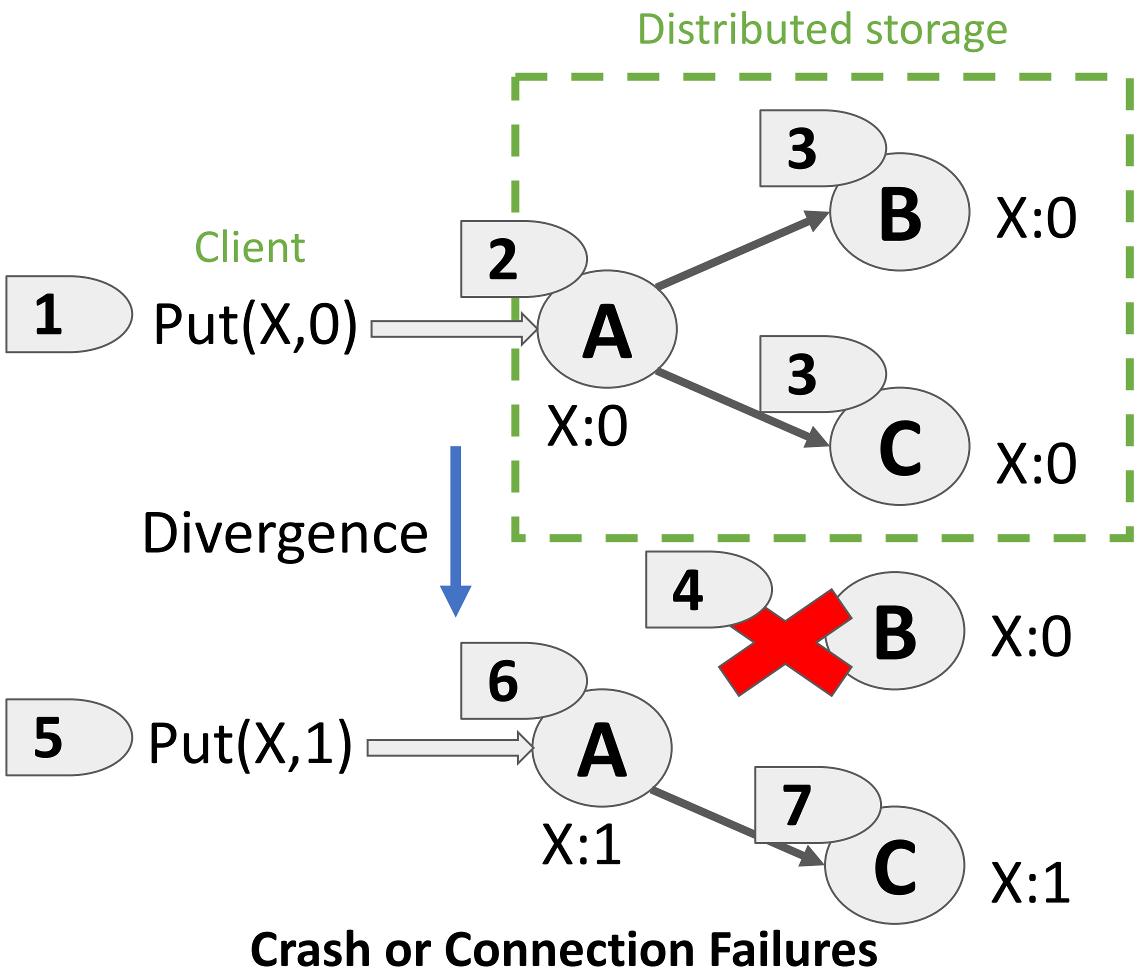
Consistency Models



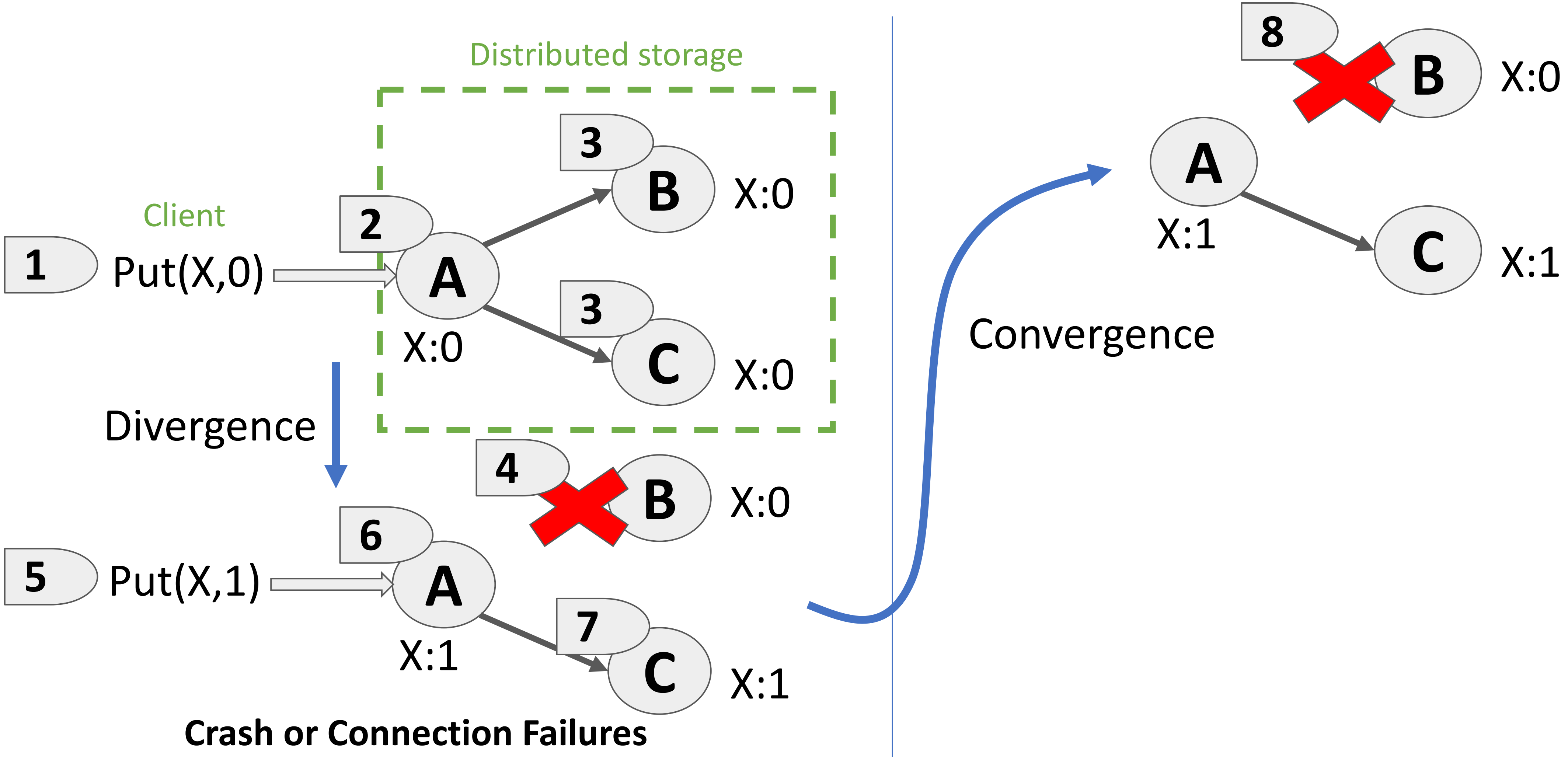
Convergence Failure Bugs (CFBs) Can Occur



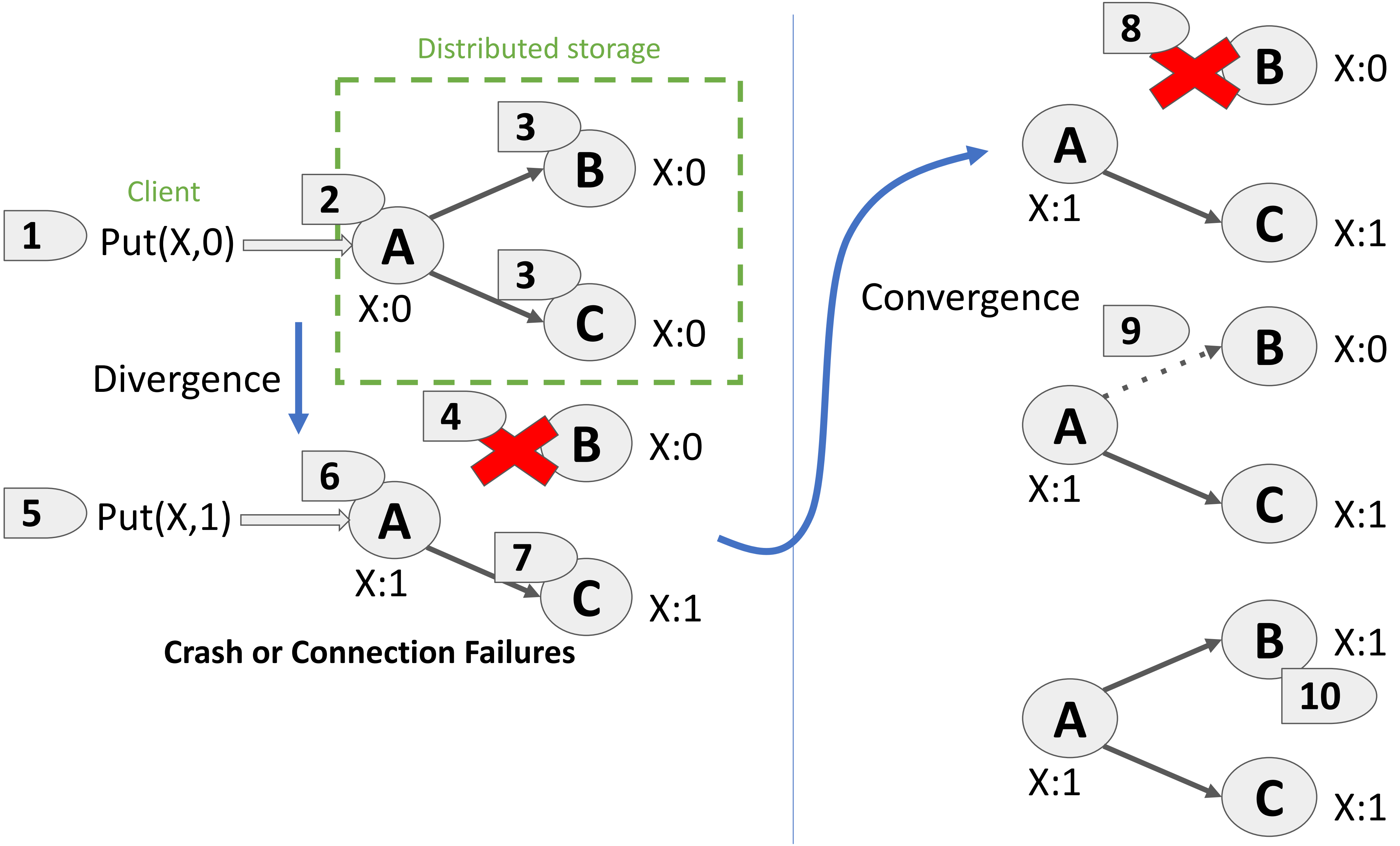
Convergence Failure Bugs (CFBs) Can Occur



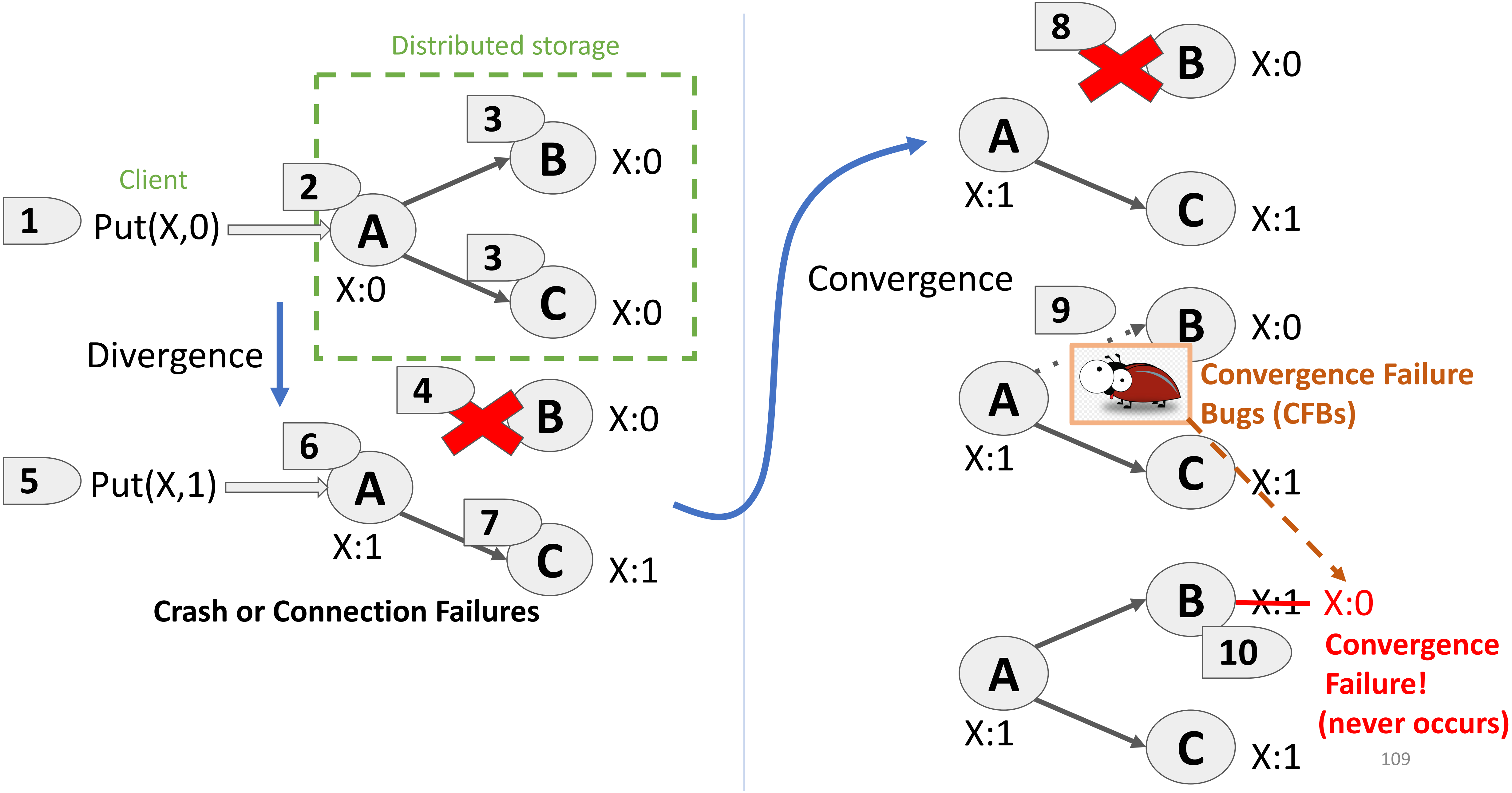
Convergence Failure Bugs (CFBs) Can Occur



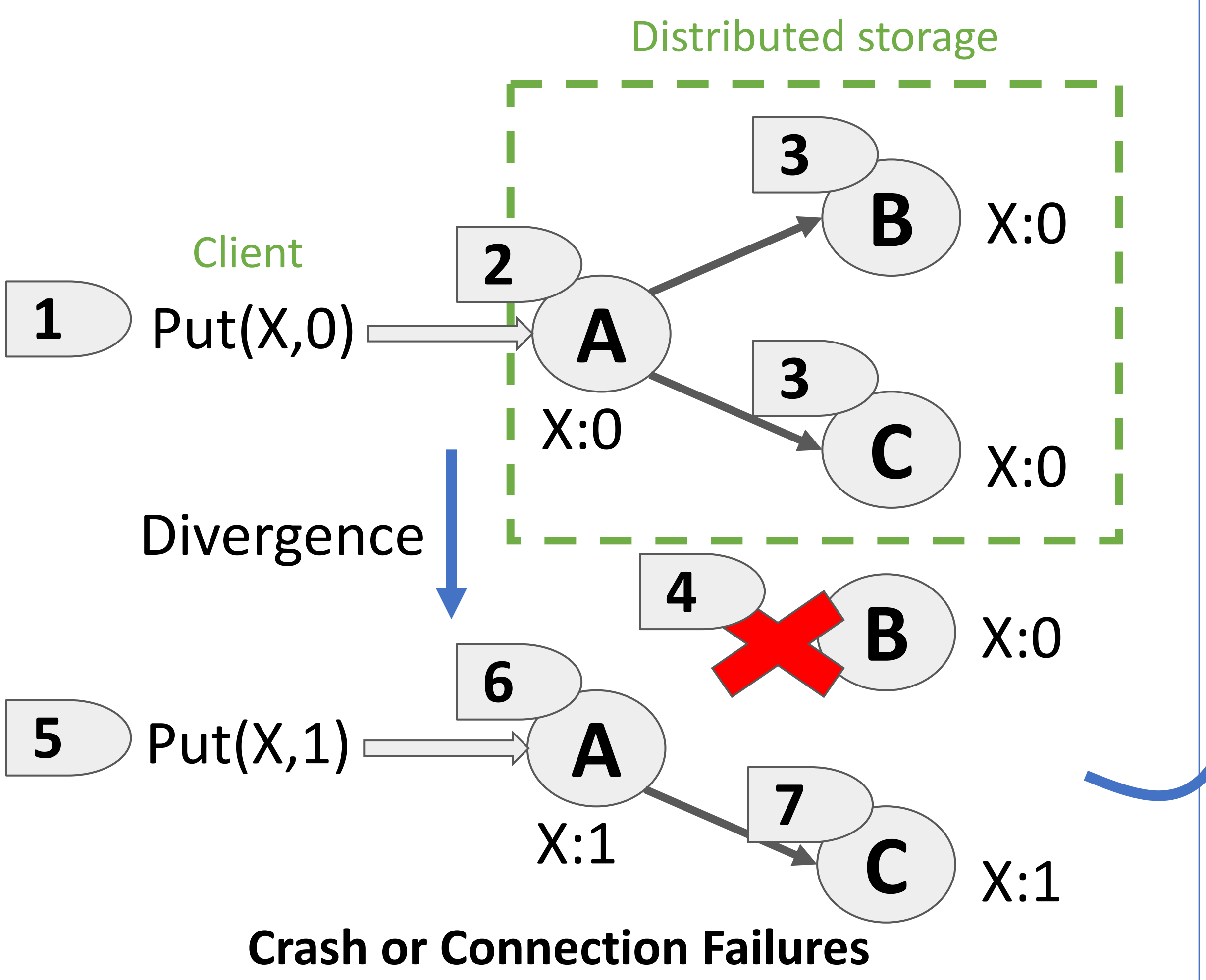
Convergence Failure Bugs (CFBs) Can Occur

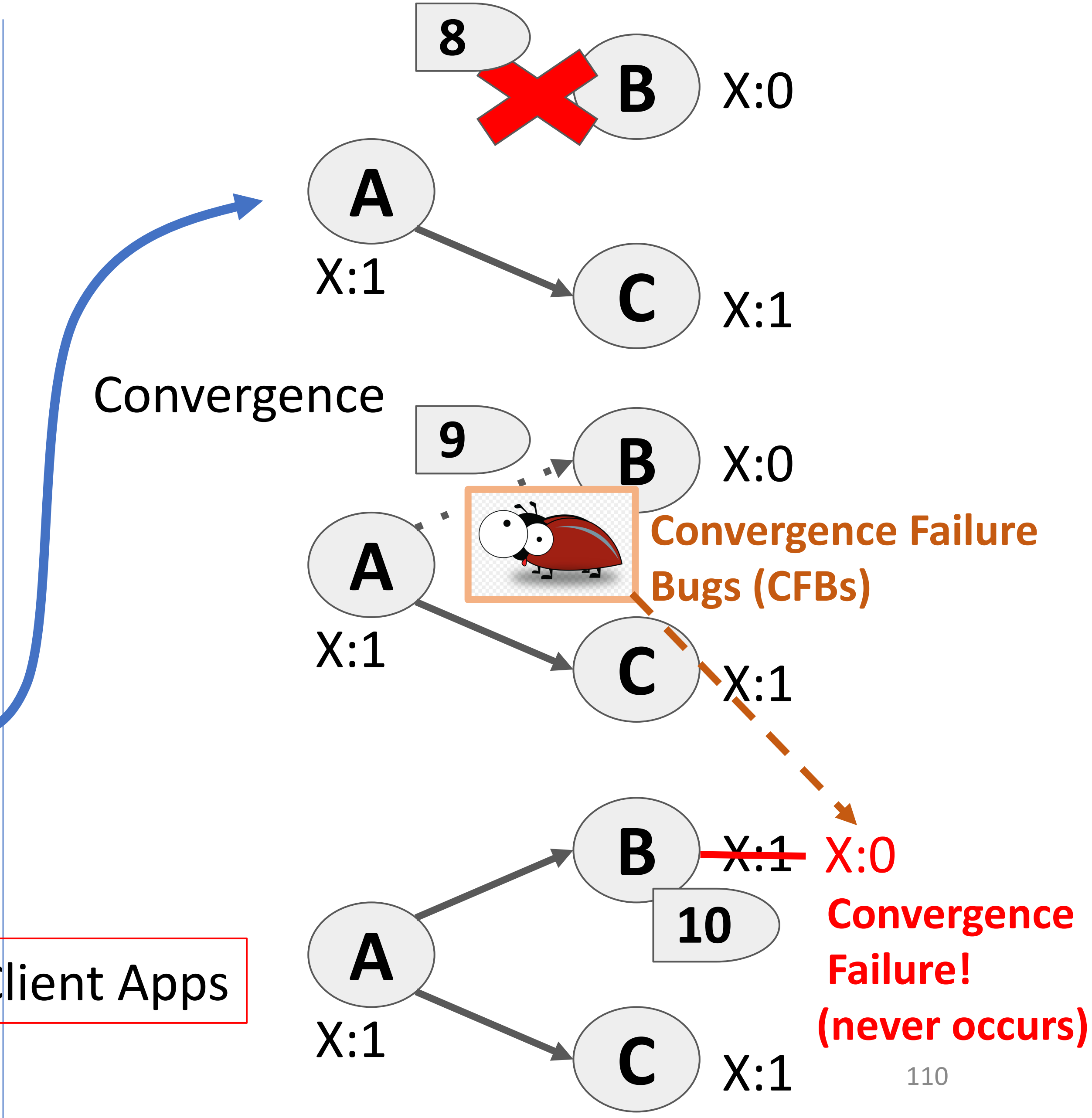


Convergence Failure Bugs (CFBs) Can Occur



Convergence Failure Bugs (CFBs) Can Occur





Convergence Failures → Incorrect Decisions of Client Apps

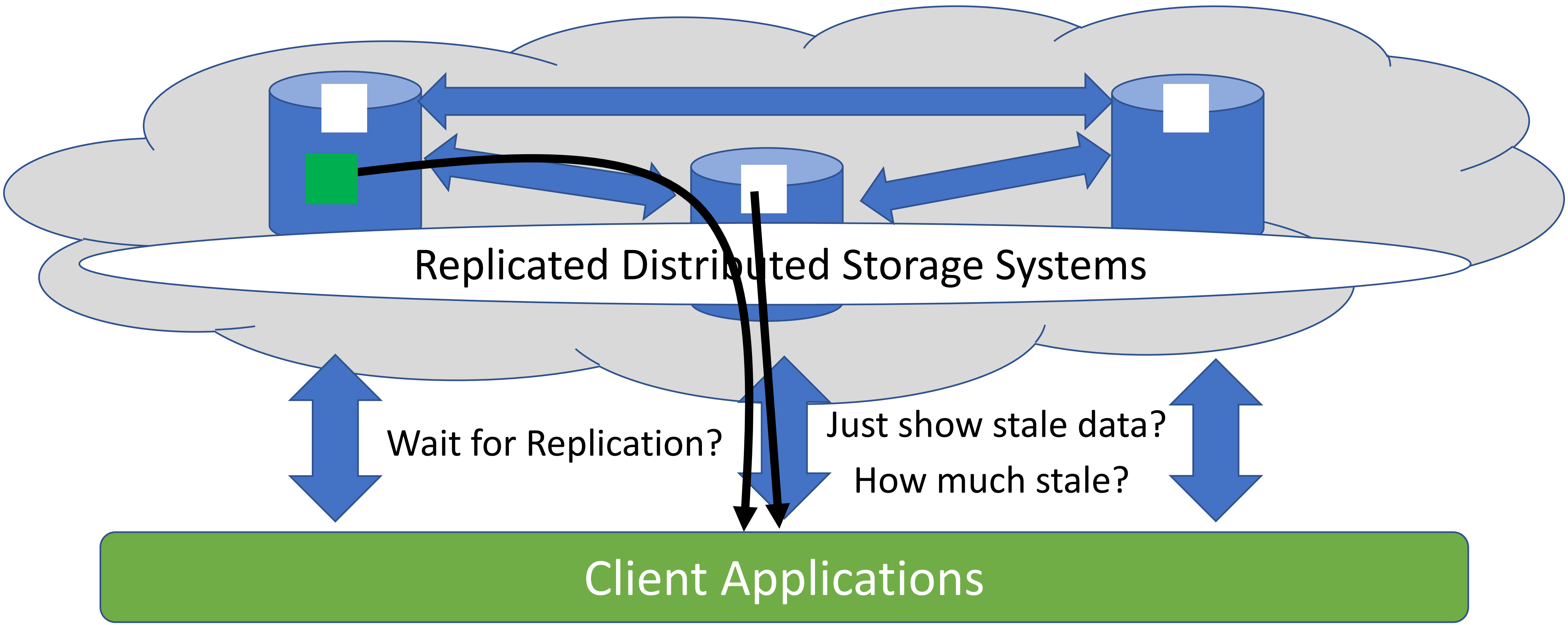
Goal: Finding Convergence Failure Bugs!

Limitations of Existing Techniques

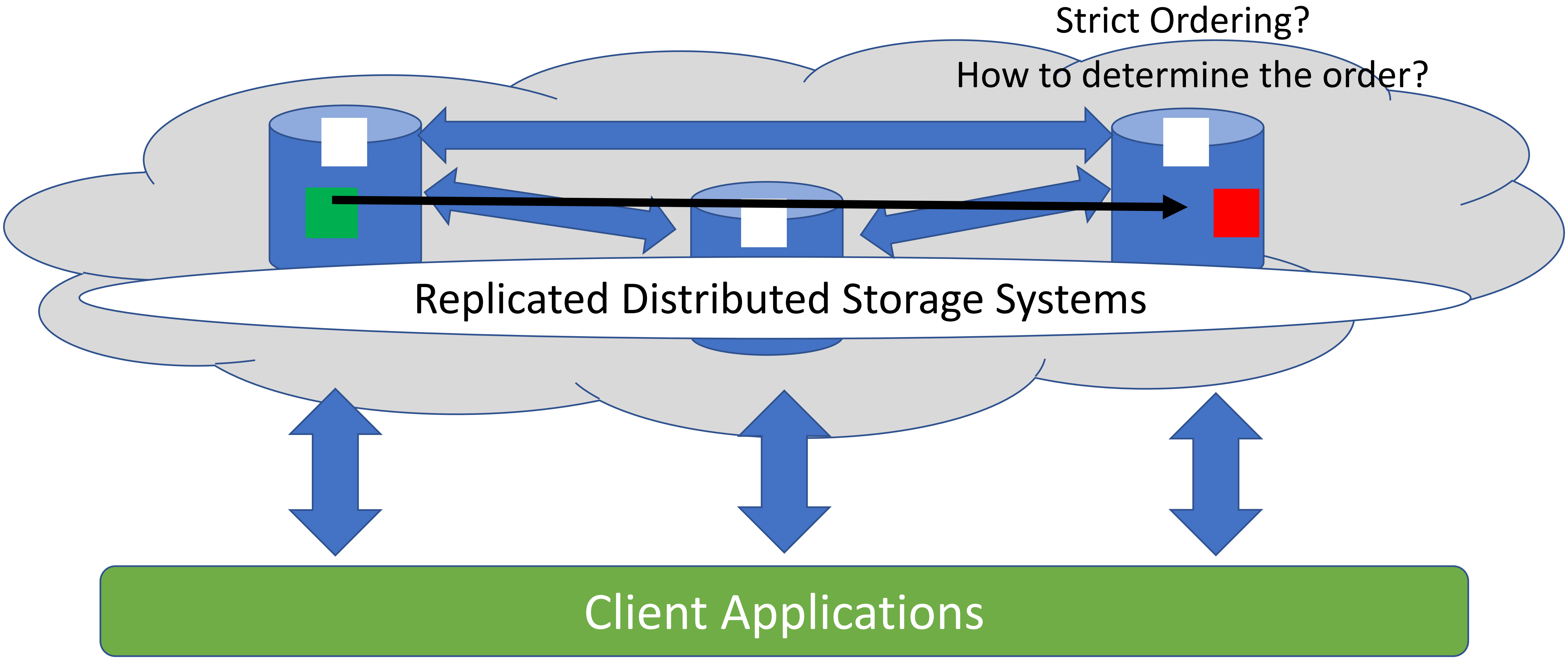
- Model-based testing and model-checking: employing state-space exploration to systematically check for the absence of bugs
  - Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion
- Manual testing and random testing: Scope of testing is usually targeted to find specific types of bugs
  - Limitation: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases

Modulo: Using a targeted approach to abstraction and concrete execution based on that abstraction to overcome those limitations

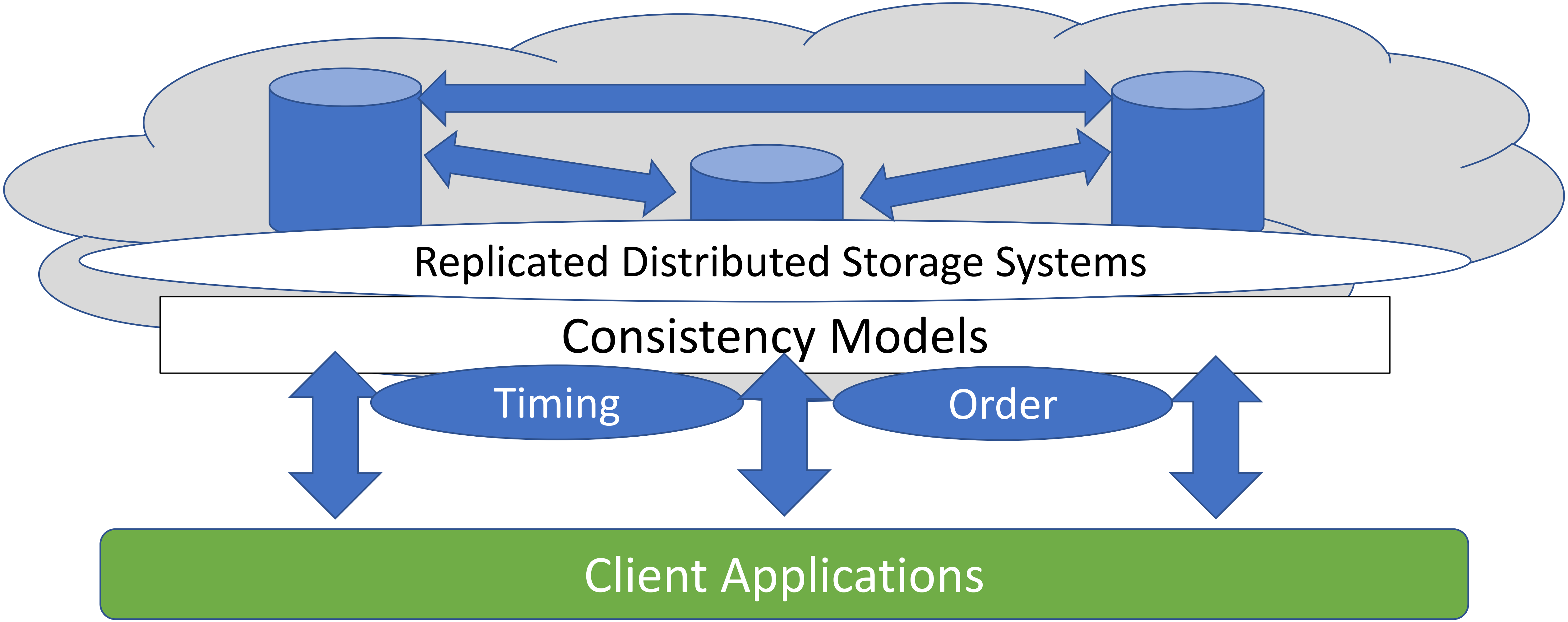
Data Consistency?



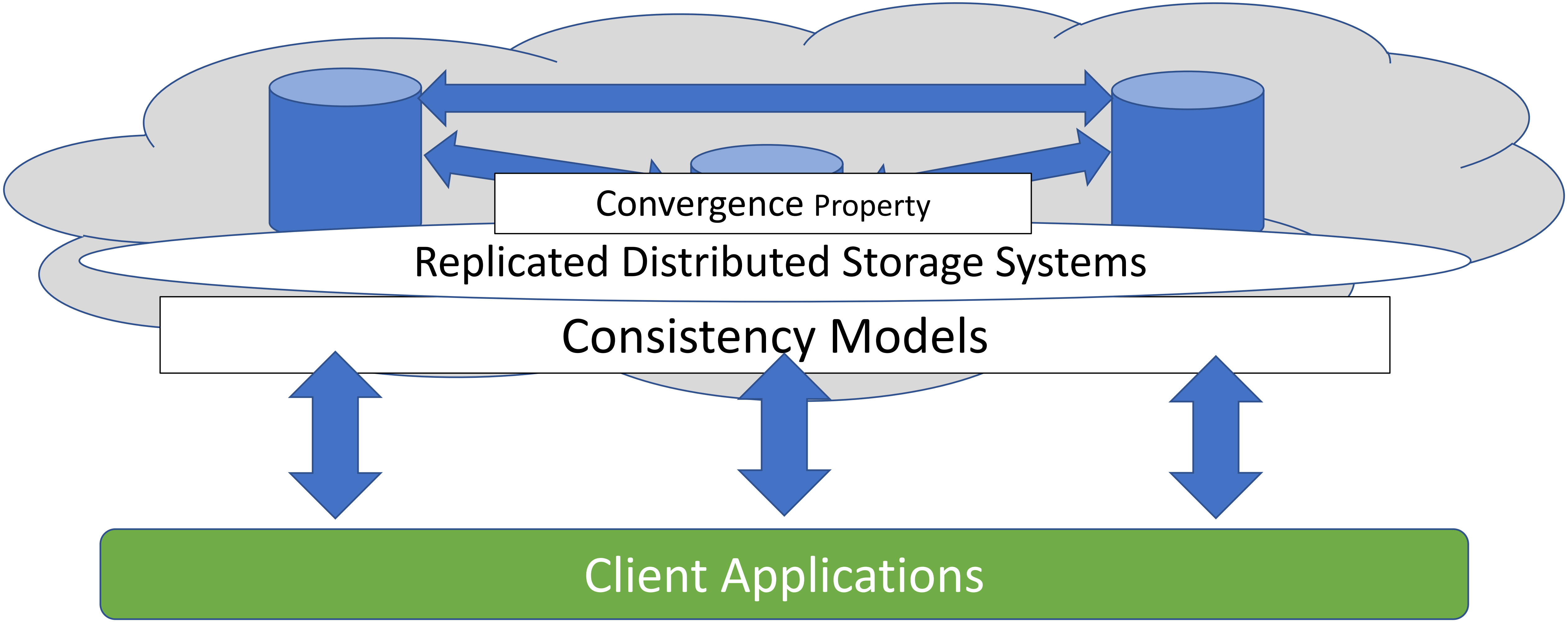
Data Consistency?



Consistency Models

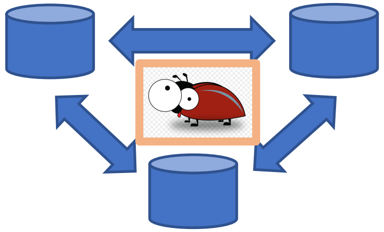


Consistency Models



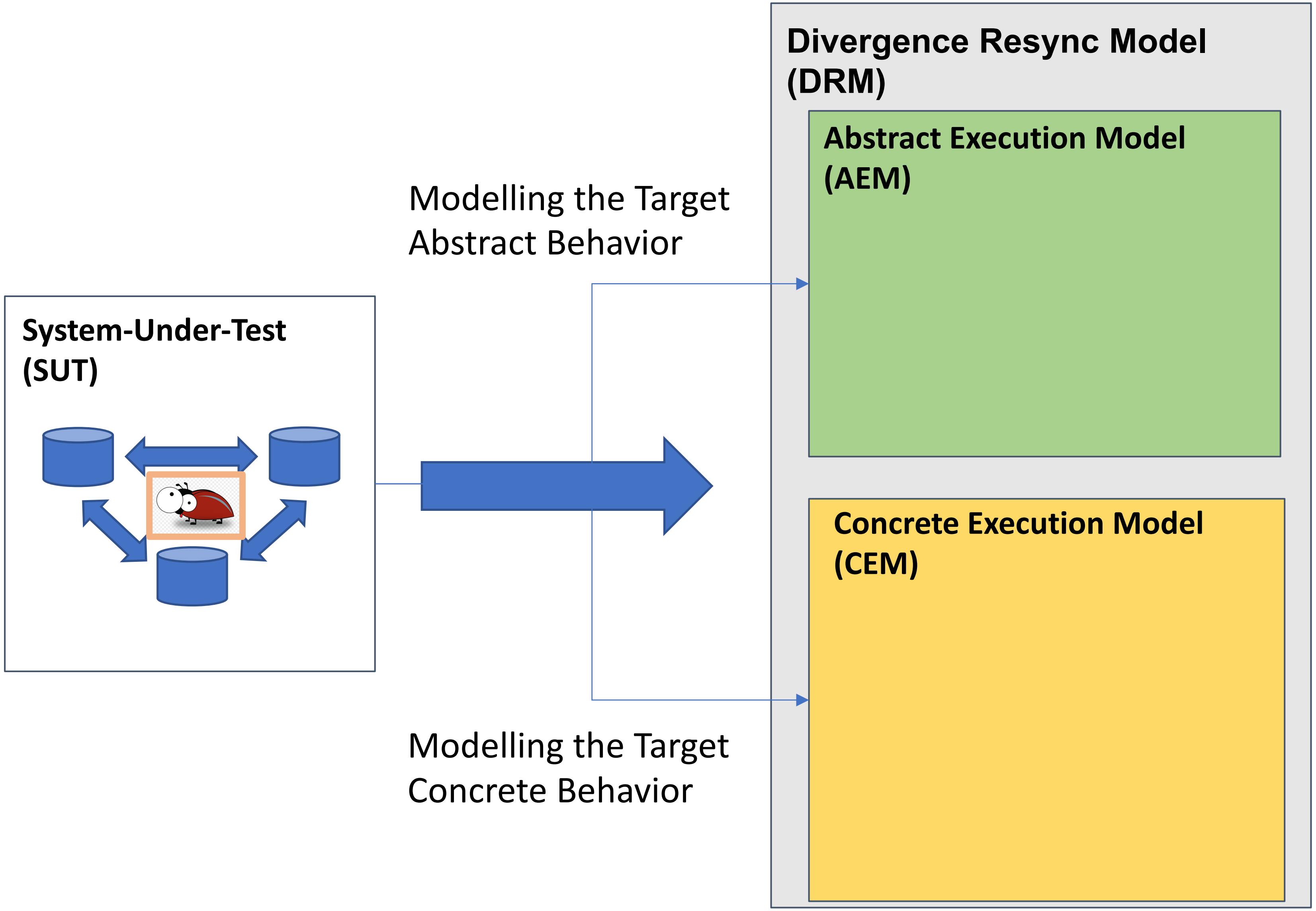
Model-based Testing with Divergence Resync Models

System-Under-Test (SUT)

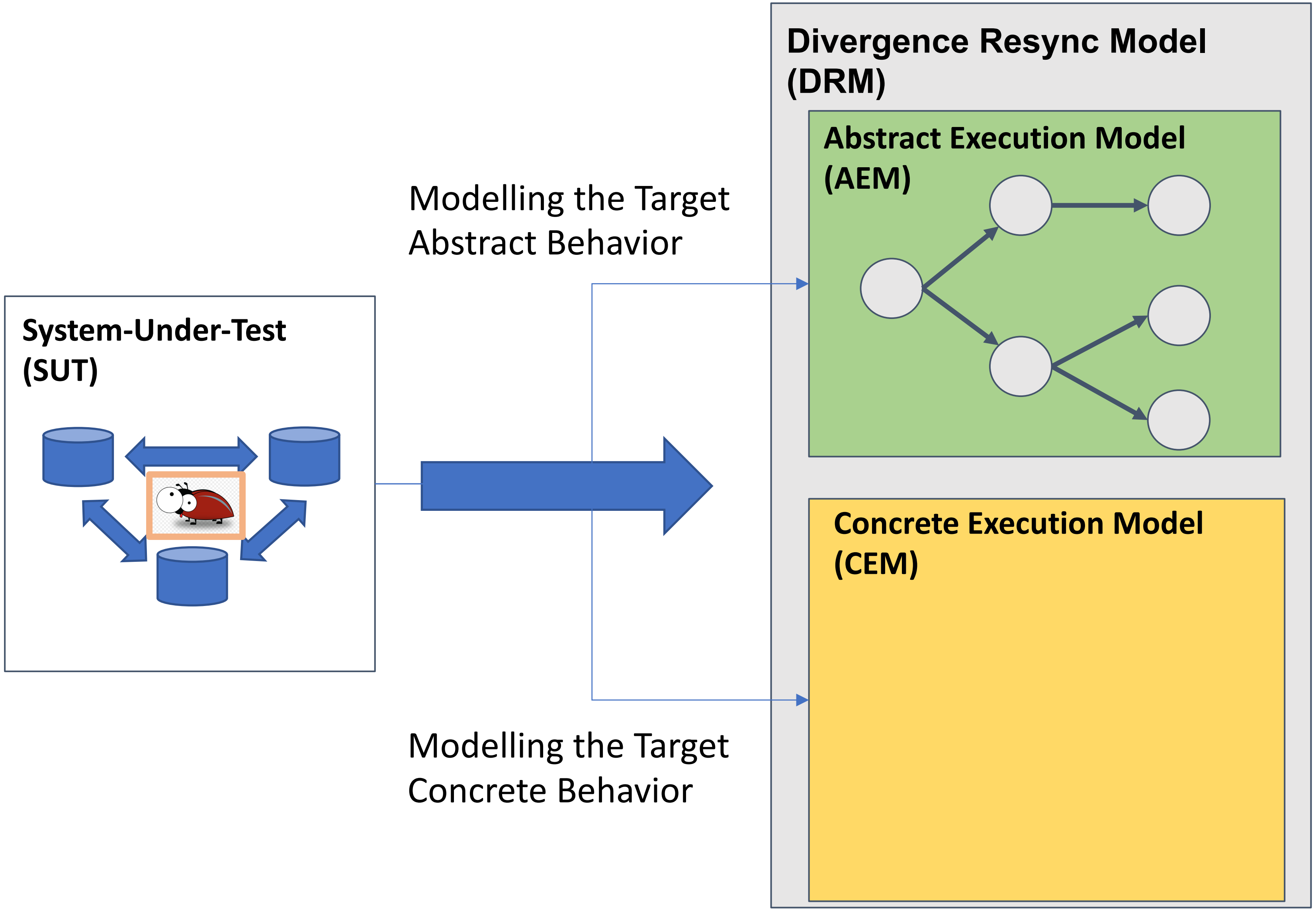


Divergence Resync Model (DRM)

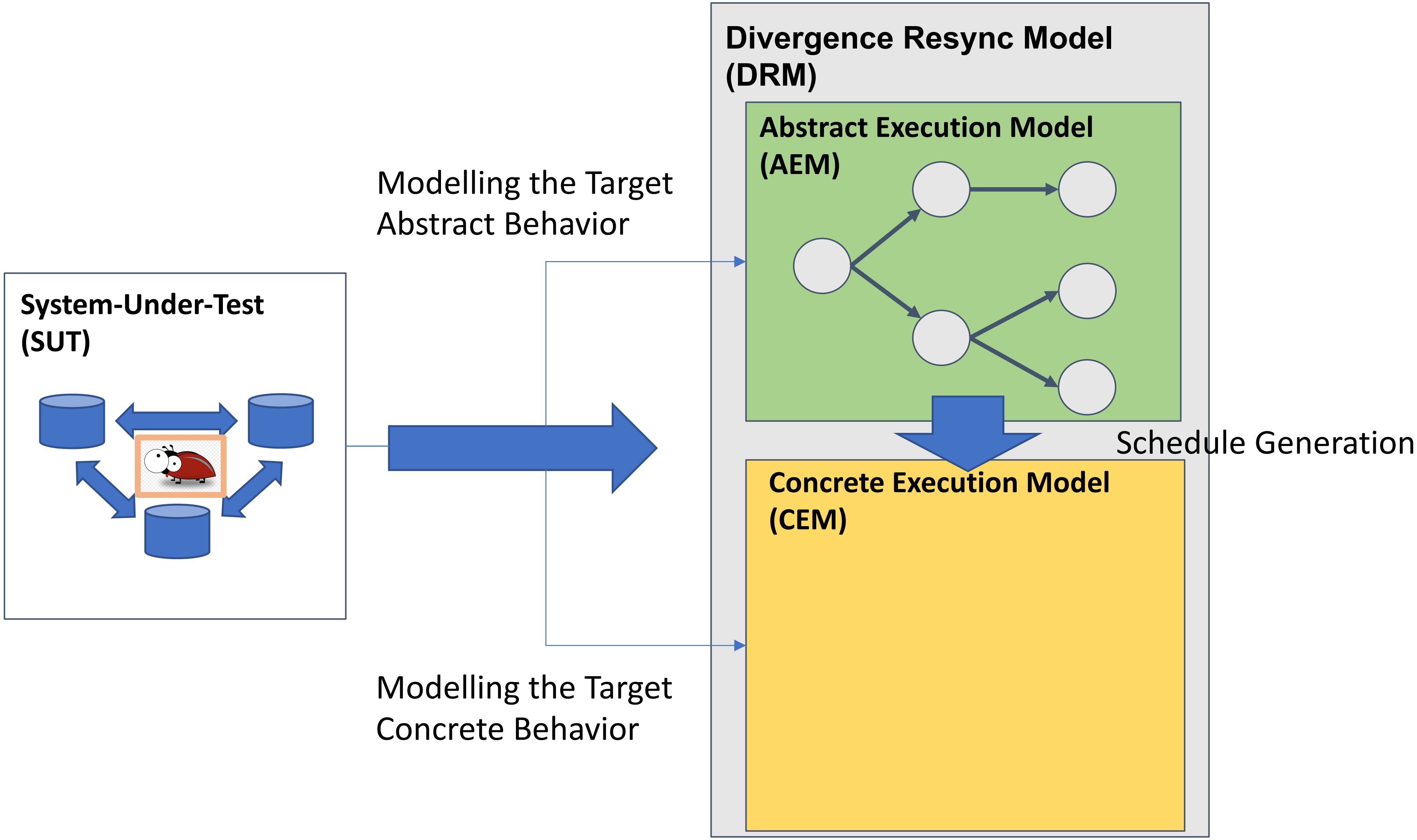
Model-based Testing with Divergence Resync Models



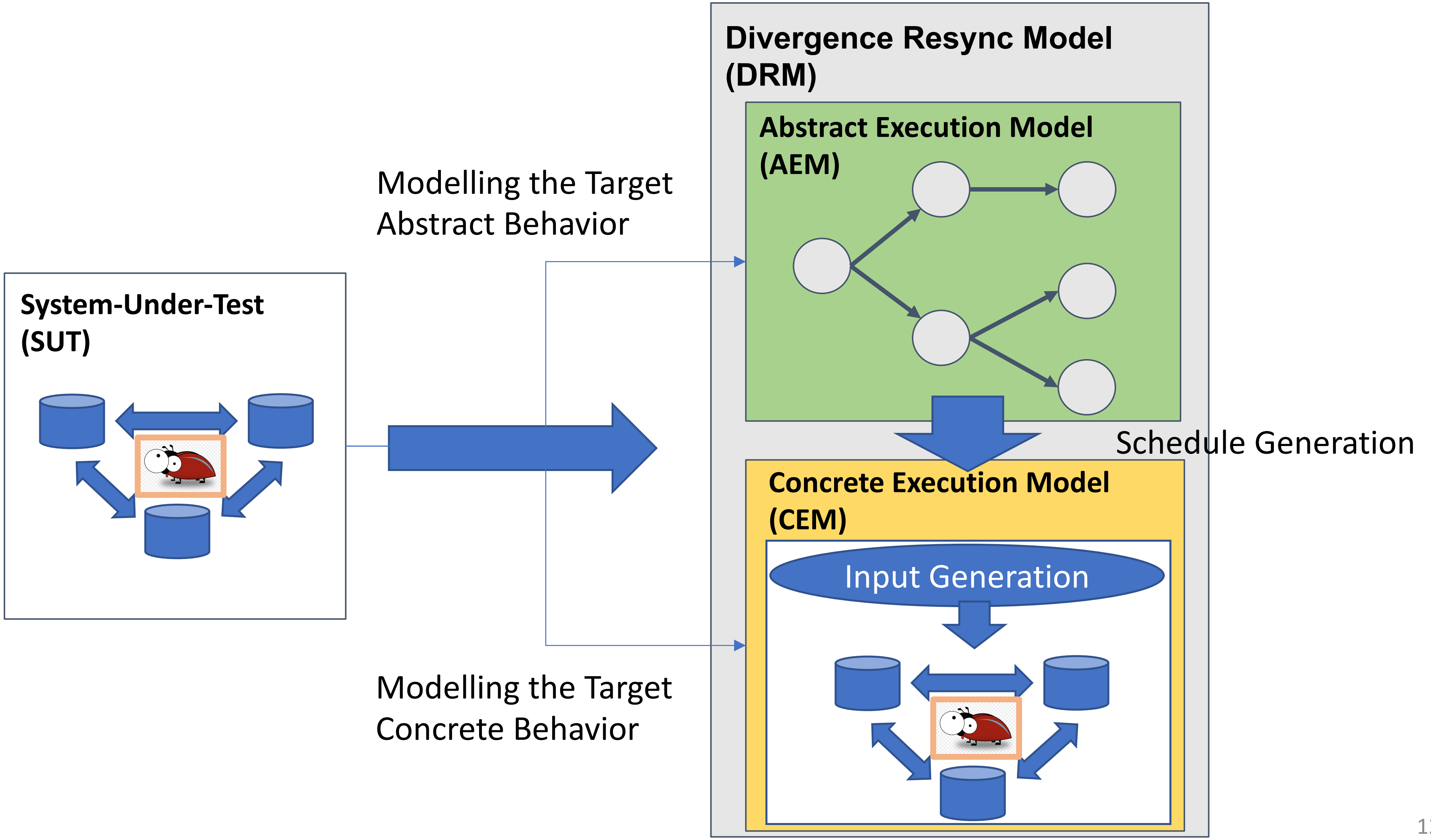
Model-based Testing with Divergence Resync Models



Model-based Testing with Divergence Resync Models



Model-based Testing with Divergence Resync Models



Differences in DRMs

- Q/C/Z-DRM CEM
  - Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  - Since version 3.5, log scanning is no longer reliable, thus fall back to time delay

Differences in DRMs

- Q/C/Z-DRM CEM
  - Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  - Since version 3.5, log scanning is no longer reliable, thus fall back to time delay
- Q/C/M-DRM
  - For MongoDB, but AEM is same as Q/C/Z-DRM
  - For CEM, it uses an API to get timestamps of the last transaction on each replica to confirm that resync completes

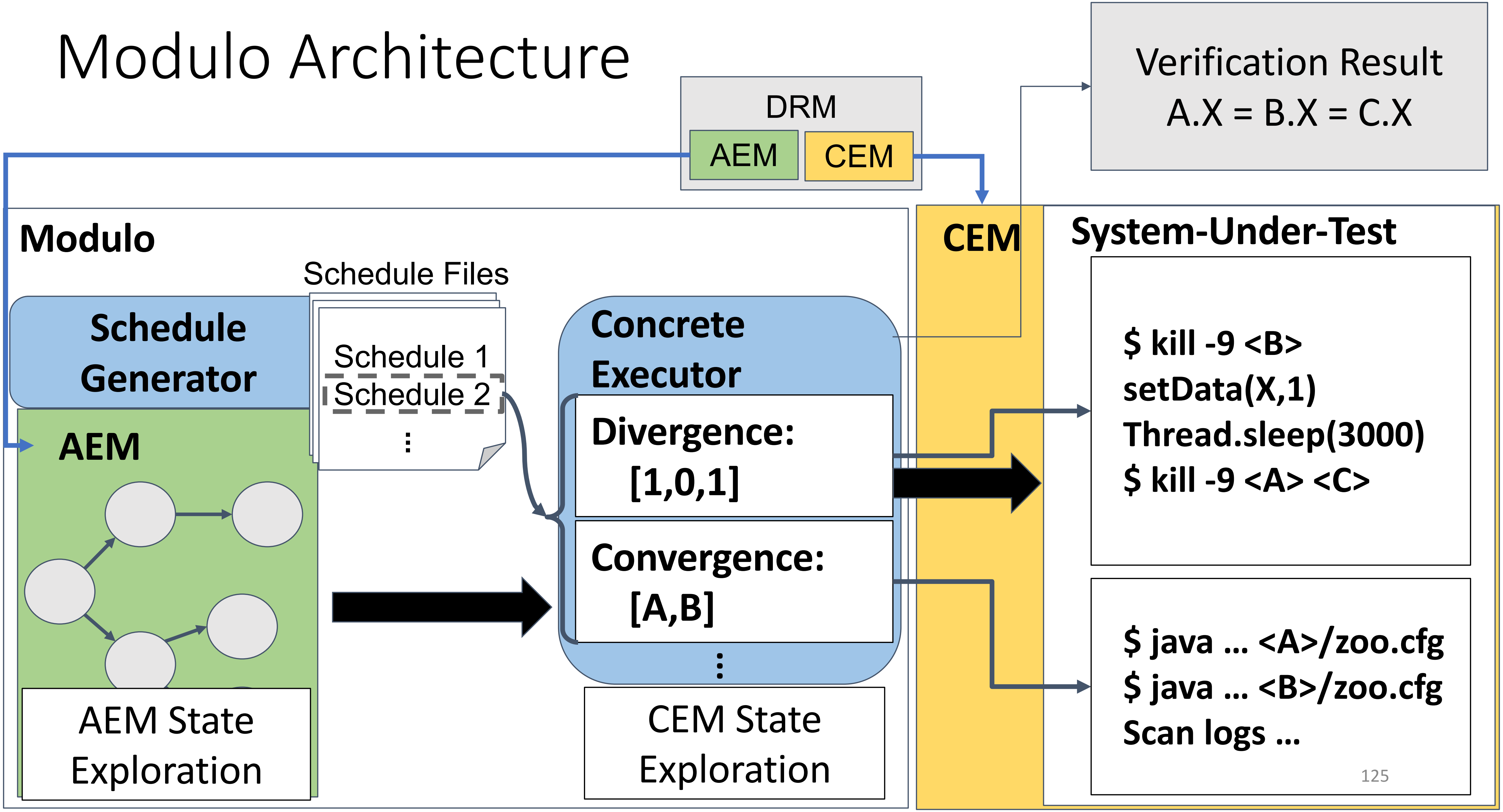
Differences in DRMs

- Q/C/Z-DRM CEM
  - Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  - Since version 3.5, log scanning is no longer reliable, thus fall back to time delay
- Q/C/M-DRM
  - For MongoDB, but AEM is same as Q/C/Z-DRM
  - For CEM, it uses an API to get timestamps of the last transaction on each replica to confirm that resync completes
- S/S/R-DRM, S/L/R-DRM, S/CL/R-DRM
  - Models for Redis uses more failure models, including link failures which requires extended AEM to keep track the status of network links between replicas

Divergence Resync Model (DRM): Differences in DRMs

- DRM for ZooKeeper
  - Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  - Since version 3.5, log scanning is no longer reliable, thus fall back to time delay
- DRM for MongoDB
  - For MongoDB, but AEM is same as the DRM for ZooKeeper
  - For CEM, it uses an API to get timestamps of the last transaction on each replica to confirm that resync completes
- DRMs for Redis
  - Models for Redis uses more failure models, including link failures which requires extended AEM to keep track the status of network links between replicas

Modulo Architecture



State Space Size

DRM	numOps	numReplicas	# of Schedules
Q/C/Z	1	3	6
	2	3	80
	3	3	1035
	4	3	13381
	5	3	172993
	3	4	3428
	3	5	54655
S/S/R	2	4	13586
S/L/R	2	3	263
S/CL/R	1	2	8
	2	2	96

Discussion

- Methodology
  - First, write DRMs in a top-down approach
  - Second, focus on the specific behavior that is important to manifest target bugs
  - Third, pay attention to configuration parameters of the system-under-test
- Modulo requires users manual effort to provide DRMs
  - Target users are developers with expertise who are interested in stress the specific behavior of the system-under-test.
  - For novice users, we expect that it requires about 2 weeks to learn about the system-under-test and about 2 weeks to write DRMs
  - Effective DRMs do require a good intuition and insight about target bugs

Conclusion

- Modulo employs targeted abstraction and concrete execution to mitigate the traditional state-explosion problems.
  - It does not explore states and state transitions that are not related to the concepts of convergence and divergence.
- Our work identified several factors that lead to CFBs:
  - (1) employing several resync or failure-handling mechanisms whose interactions are difficult to foresee
  - (2) hard limits or inadequate designs for handling large amounts of divergence
  - (3) assumptions about length of time that replicas may have failed and failures that span events like leader transitions.
- Modulo's performance is heavily affected to delays from executing and controlling the real distributed system