NTNU

Innovation and Creativity

Preventing Orphan Requests by Integrating Replication and Transactions

Heine Kolltveit and Svein-Olaf Hvasshovd

Department of Computer and Information Science

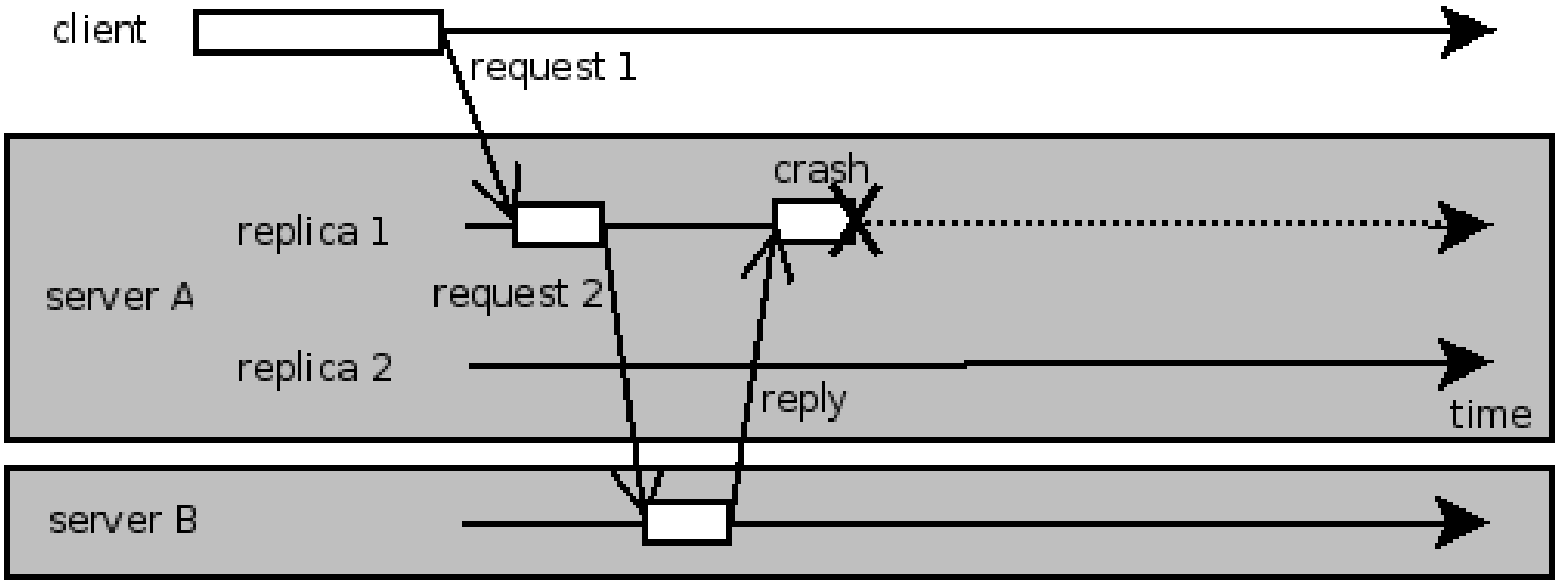
Norwegian University of Science and Technology

Outline

- Motivation
- Integration
- Performance Results
- Summary

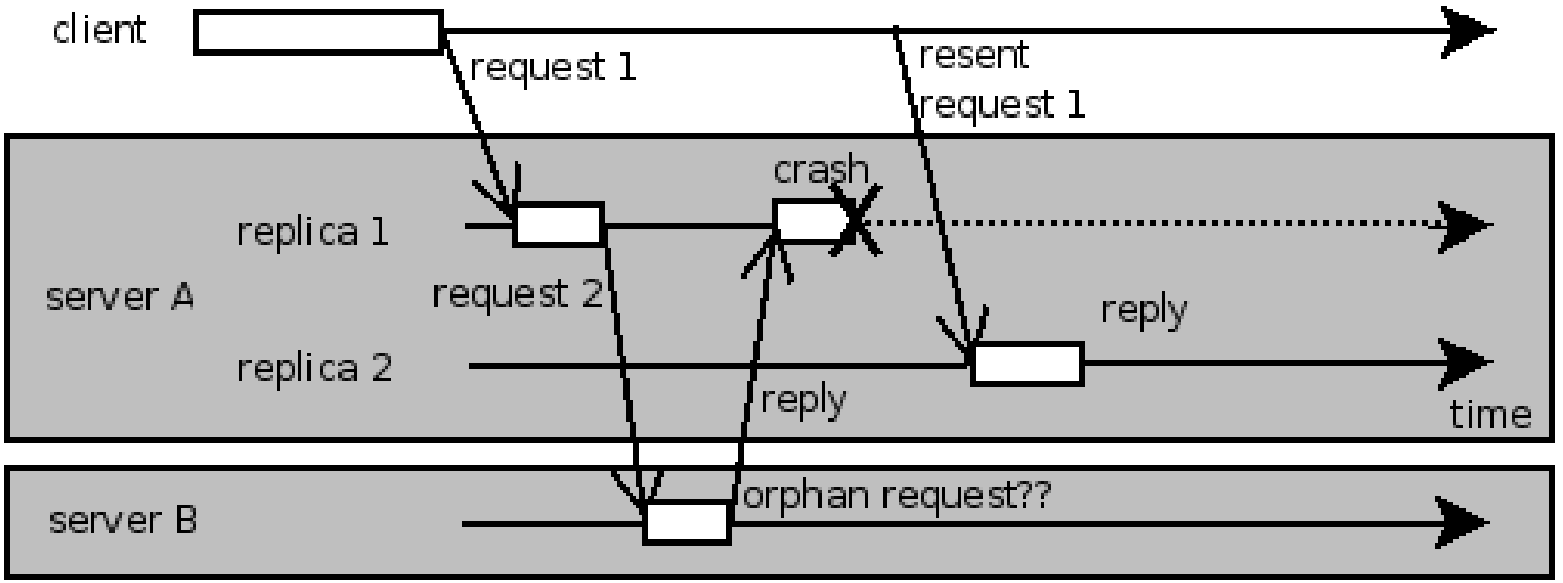
NTNU

Orphan Request

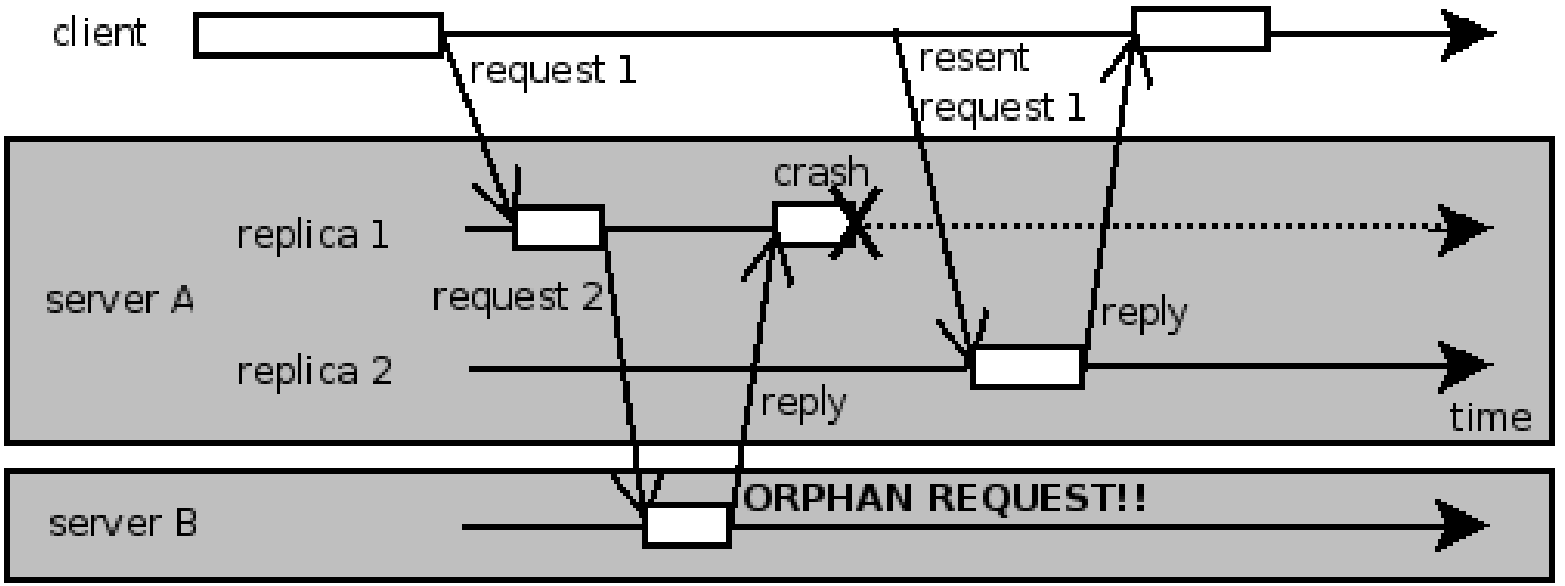


NTNU

Orphan Request



Orphan Request



NTNU

Determinism

- Identical input gives identical output and identical internal state changes.
- Sources of non-determinism [Poledna 1993]:
  - Multi-threading
  - Timeouts
  - Reading analogue sensors
  - Inconsistent order
  - ...

NTNU

Availability And Consistency

- Replication -> Availability
  - Active
  - Passive
- Transactions -> Consistency
  - ACID properties

NTNU

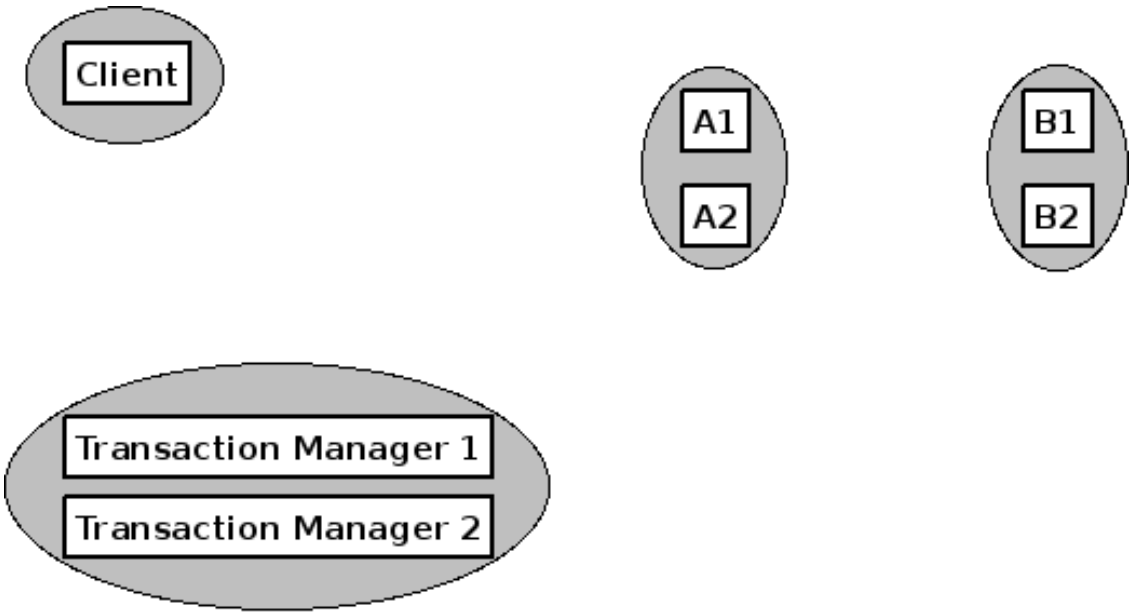
System Model

- Set S of fail-crash processes (nodes)
- Group G implements a service
- Node in G is called a replica
- Group view
- Passive replication
- Two-Phase Commit protocol
- Stateful, but no persistent state
- All replicas are non-deterministic

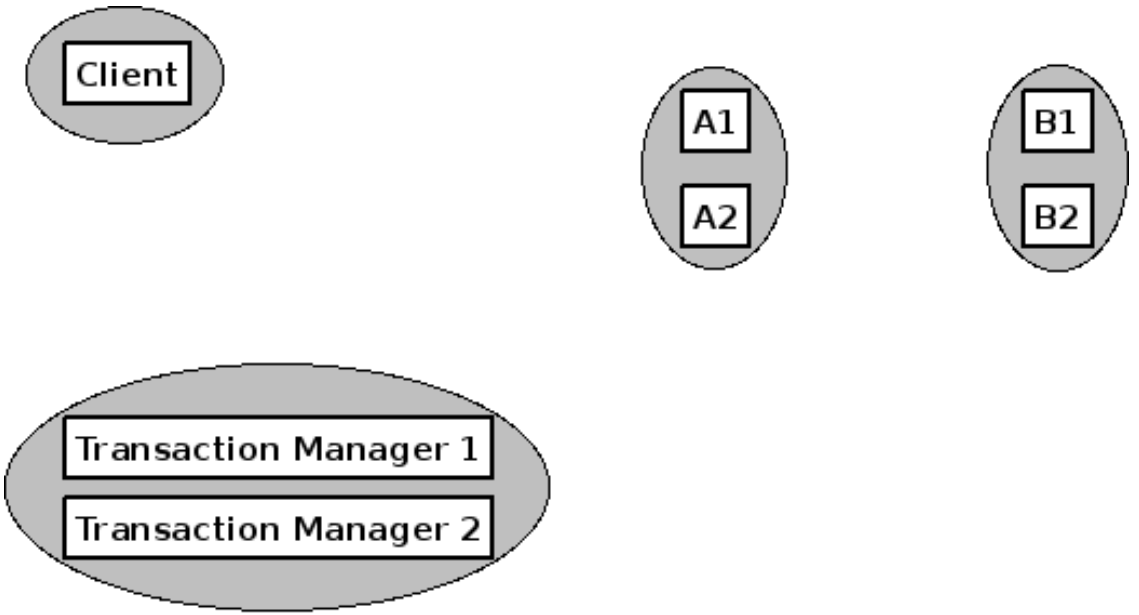
NTNU

Components

- Client - initiate transaction and commitment processing
- Transaction Manager - creates txn ID and manages commitment
- Transaction Participants - execute requests and vote

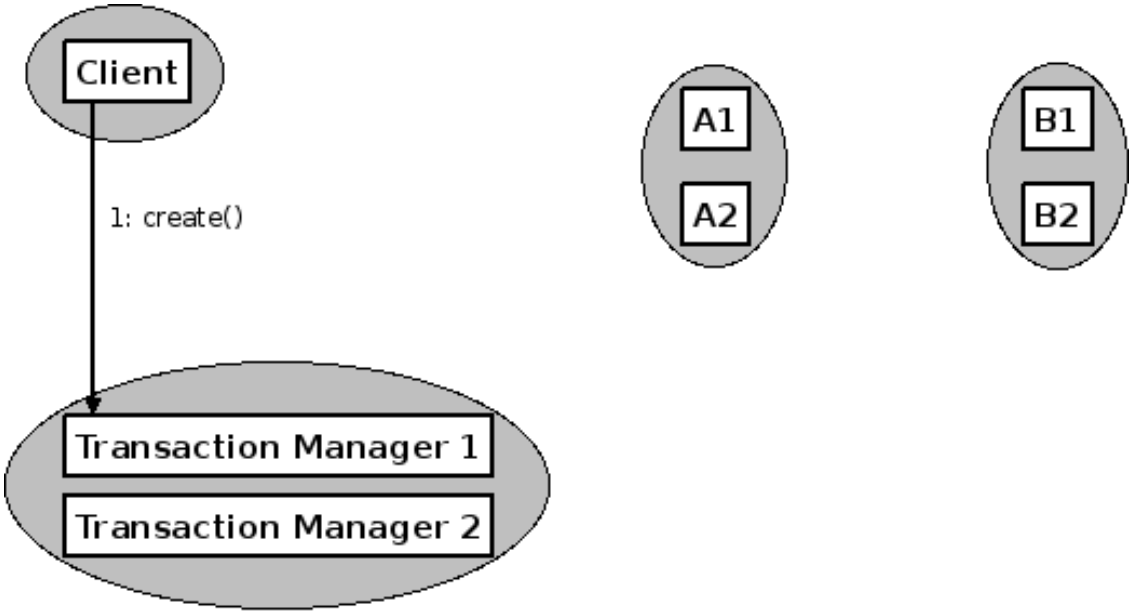


Transaction Initiation and Execution

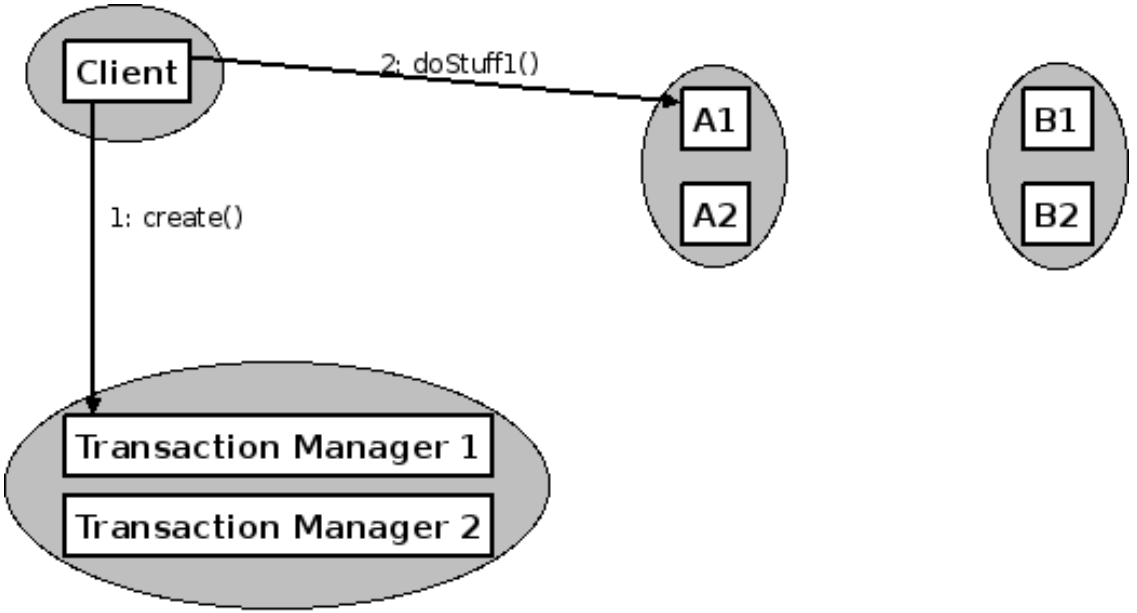


NTNU

Transaction Initiation and Execution

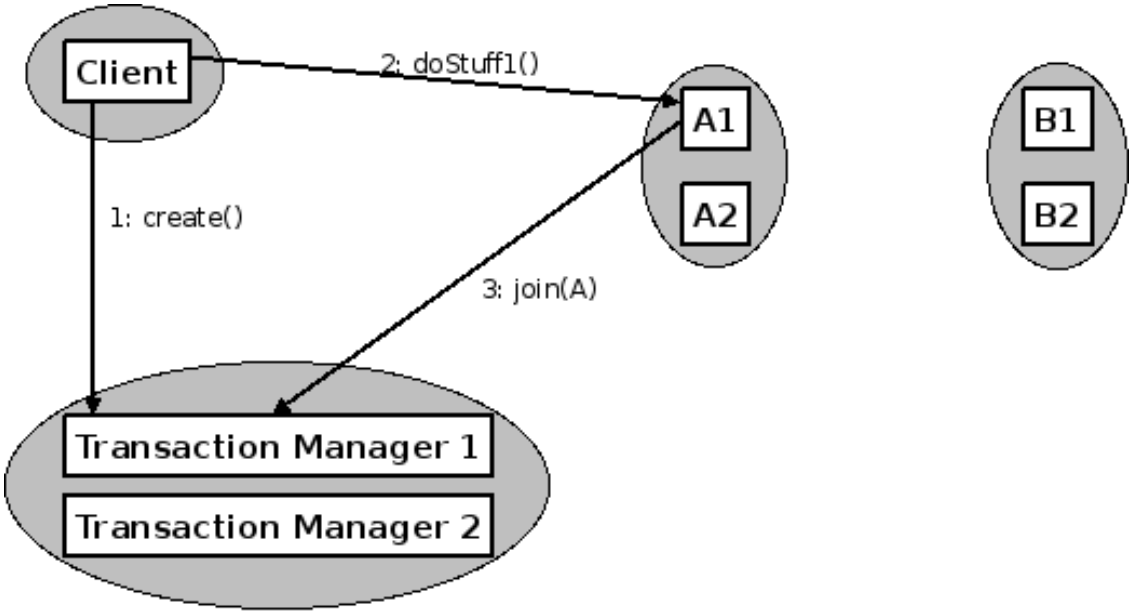


Transaction Initiation and Execution



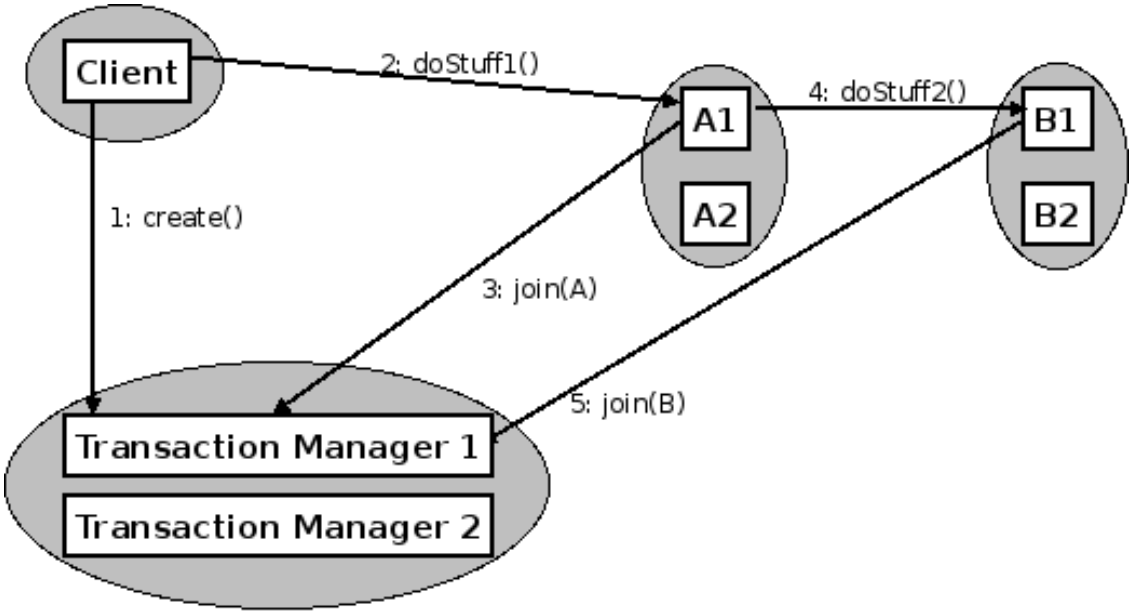
NTNU

Transaction Initiation and Execution



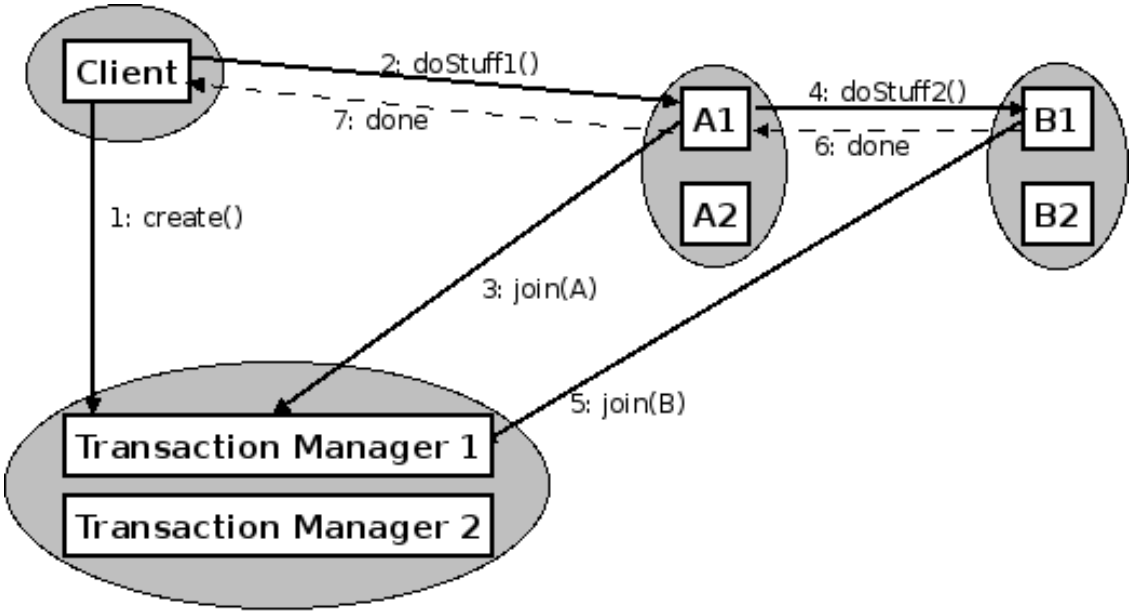
NTNU

Transaction Initiation and Execution

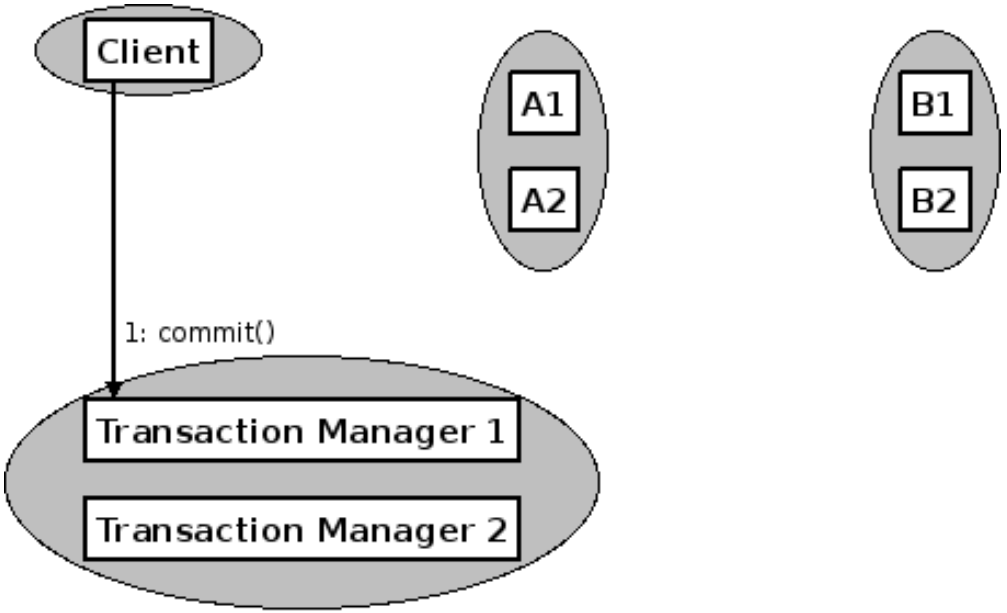


NTNU

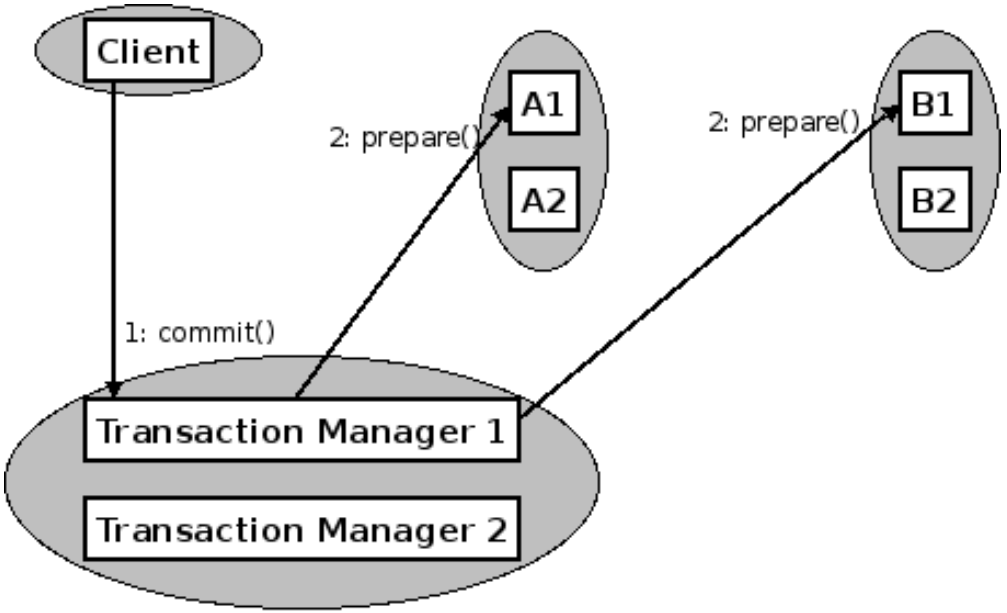
Transaction Initiation and Execution



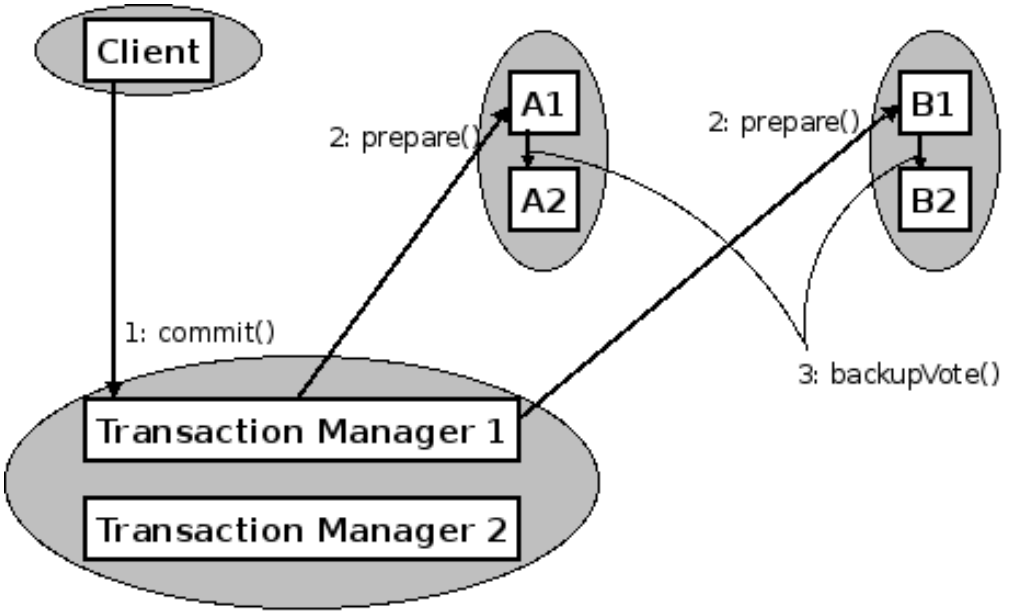
Transaction Termination



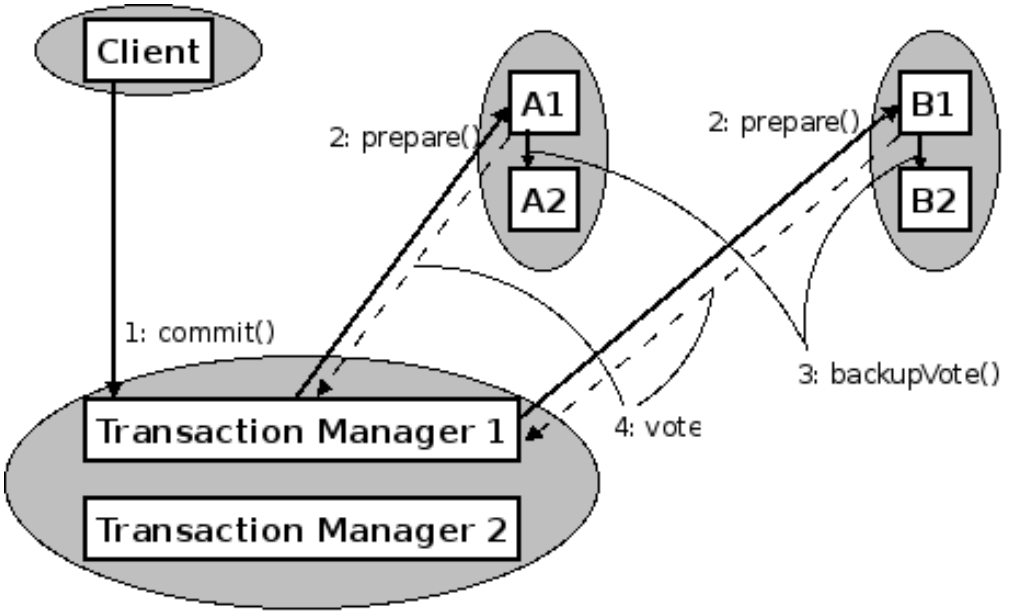
Transaction Termination



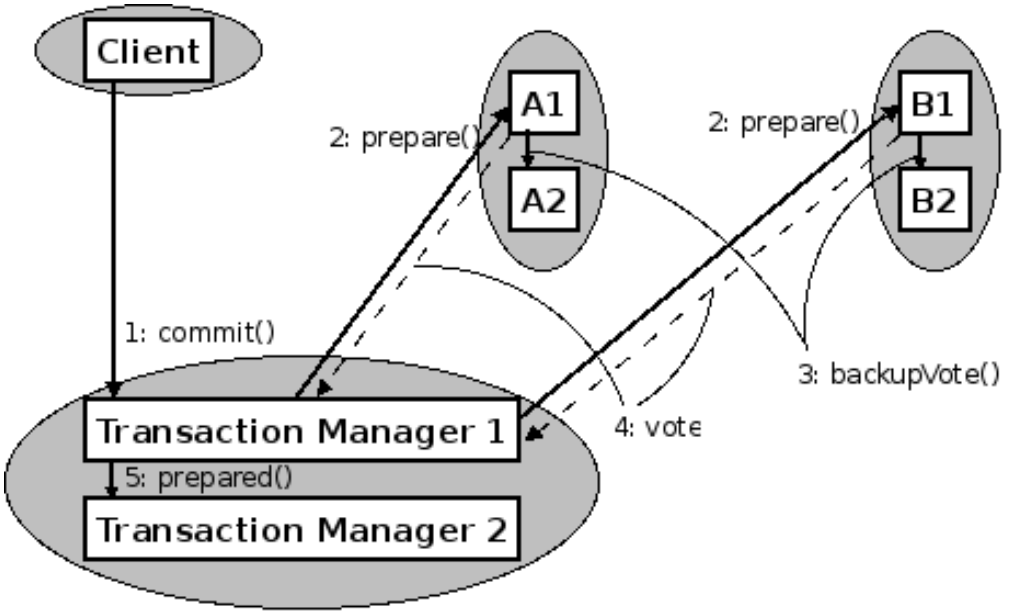
Transaction Termination



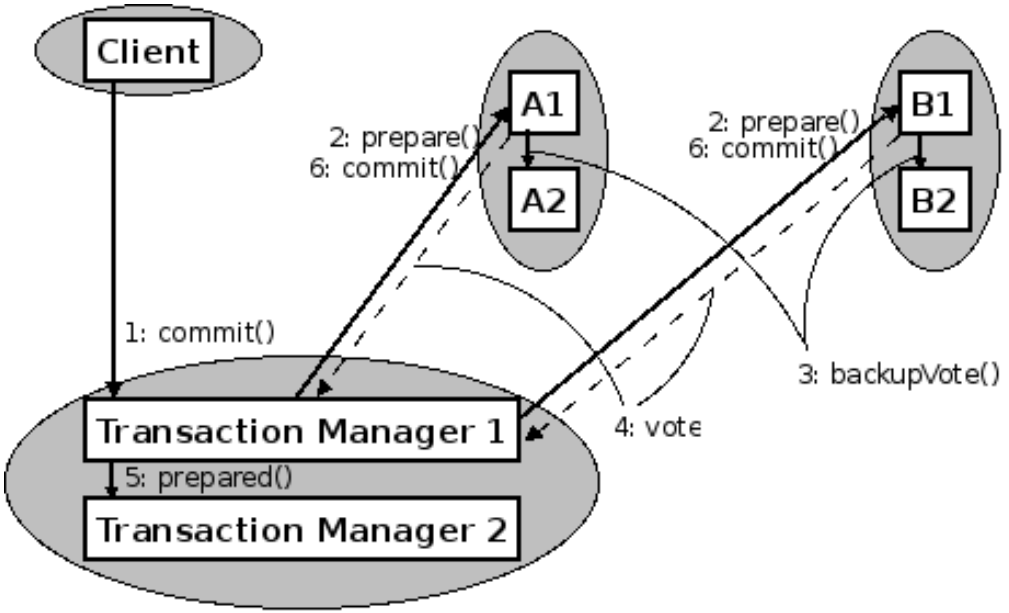
Transaction Termination



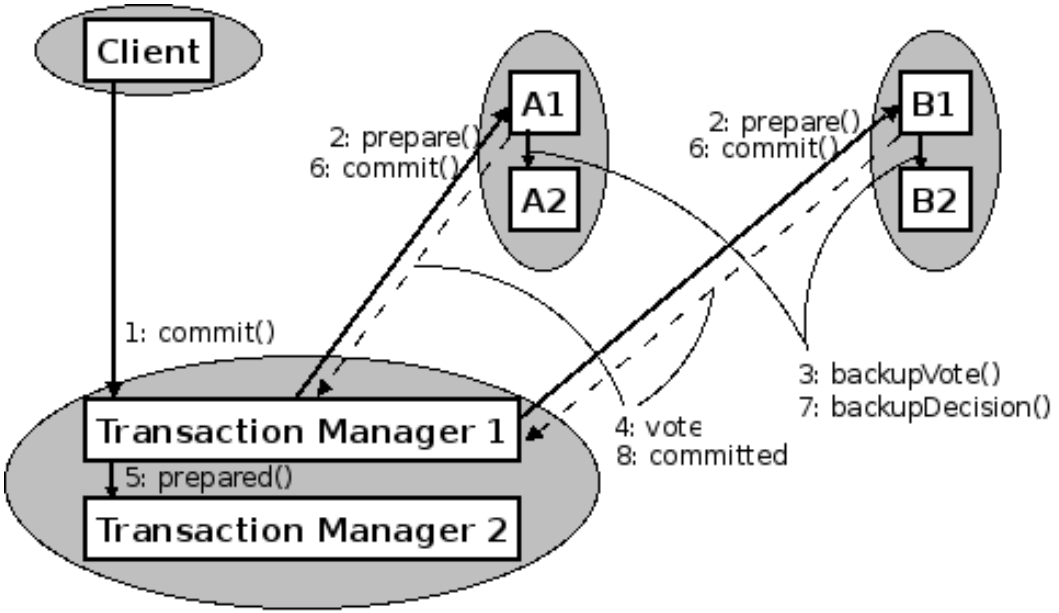
Transaction Termination



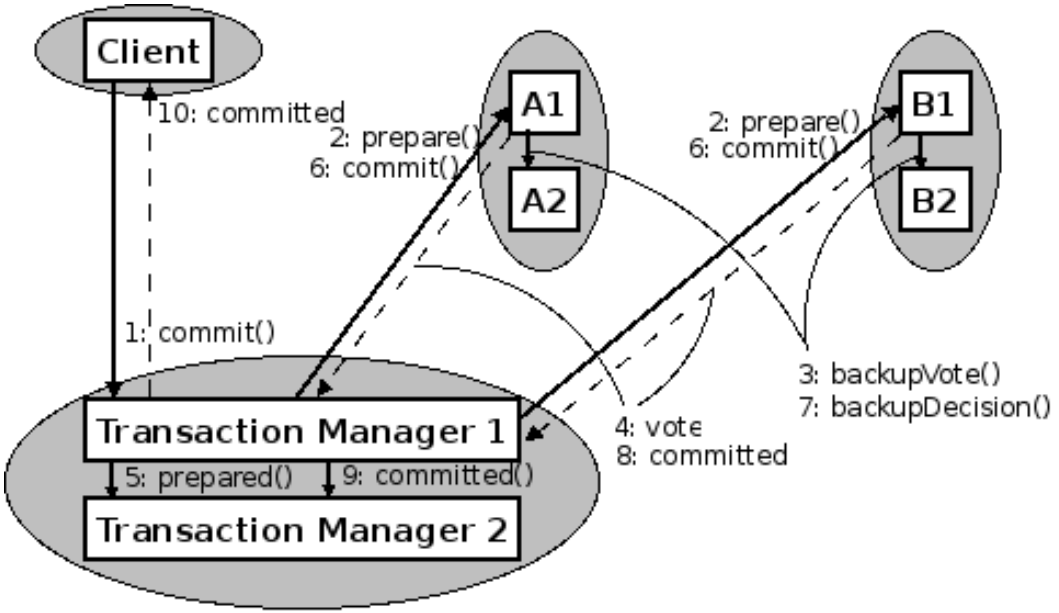
Transaction Termination



Transaction Termination



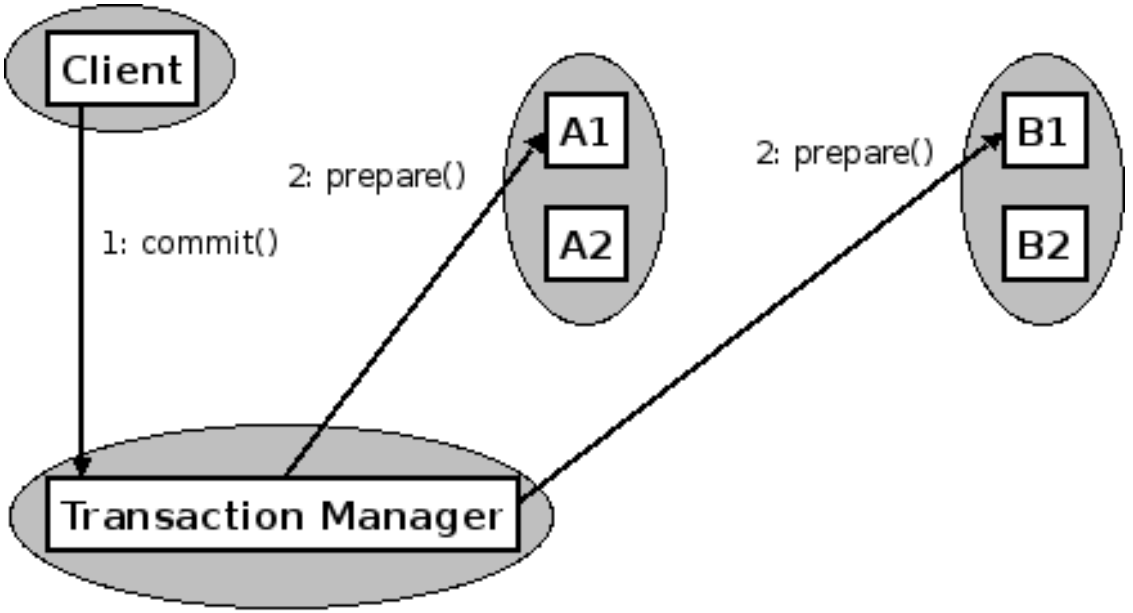
Transaction Termination



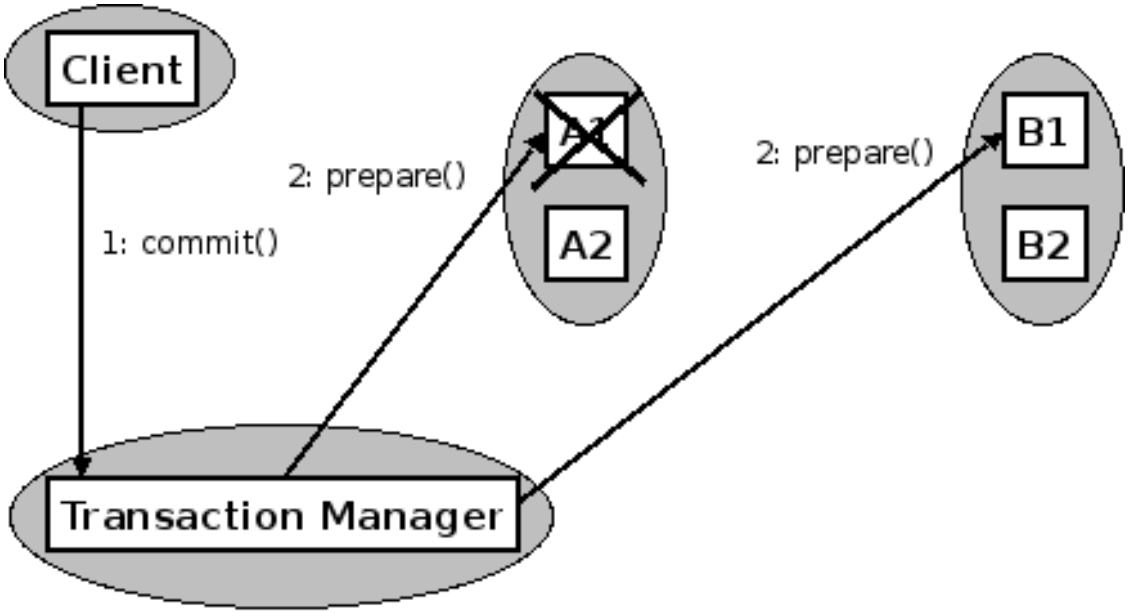
Replication Transparency

- The replication of the server is hidden from the client
- Server-side proxy sends message to primary replica
- Group communication mechanisms keeps the proxy up to date
- Failovers are handled by proxy, server does not know

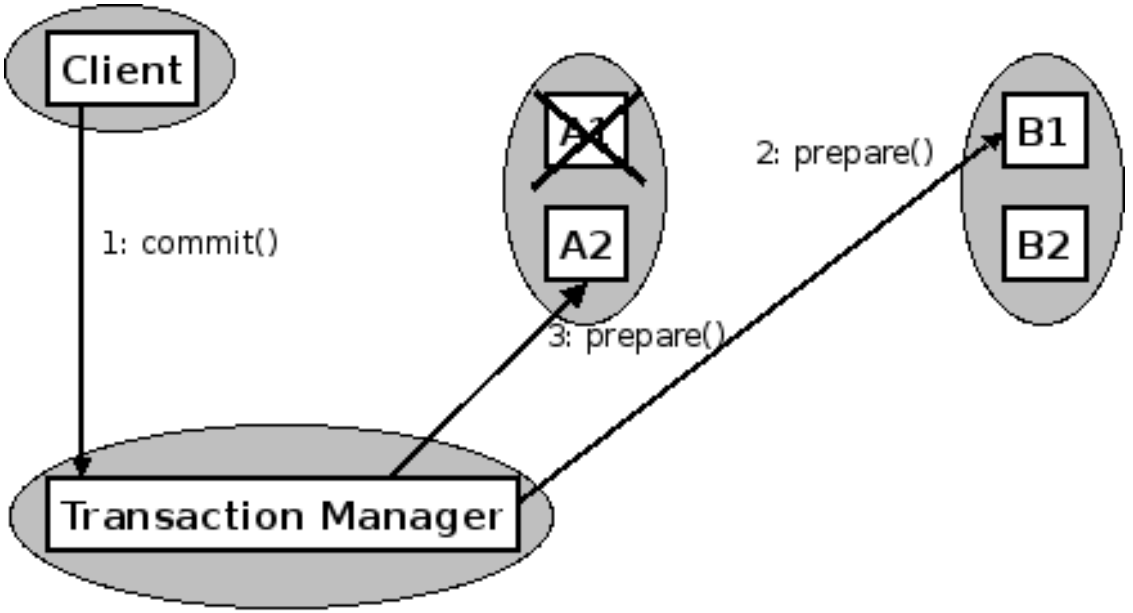
Failover


Client
1: commit()
2: prepare()
A1
A2
2: prepare()
B1
B2
Transaction Manager

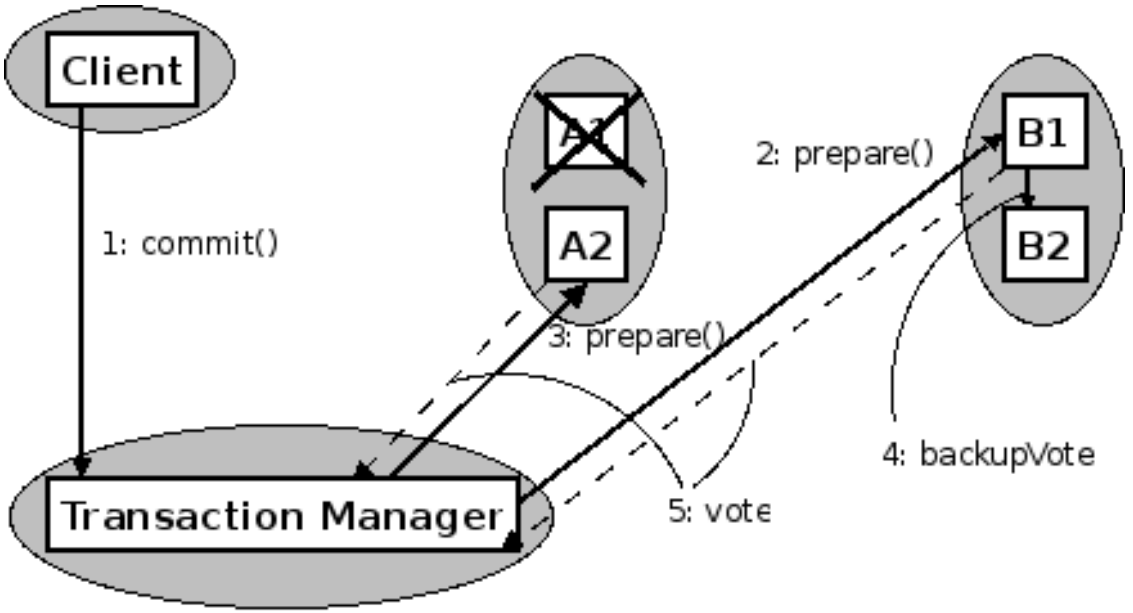
Failover


Client
2: prepare()
1: commit()
A2
2: prepare()
B1
B2
Transaction Manager

Failover

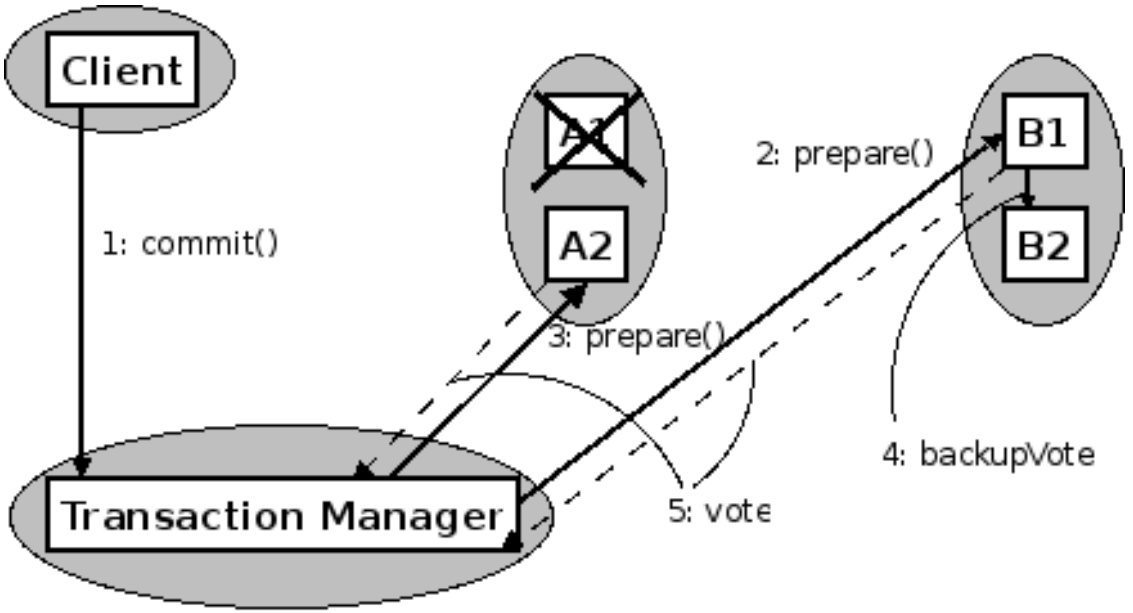

Client
A1
A2
B1
B2
1: commit()
2: prepare()
3: prepare()
Transaction Manager

Failover



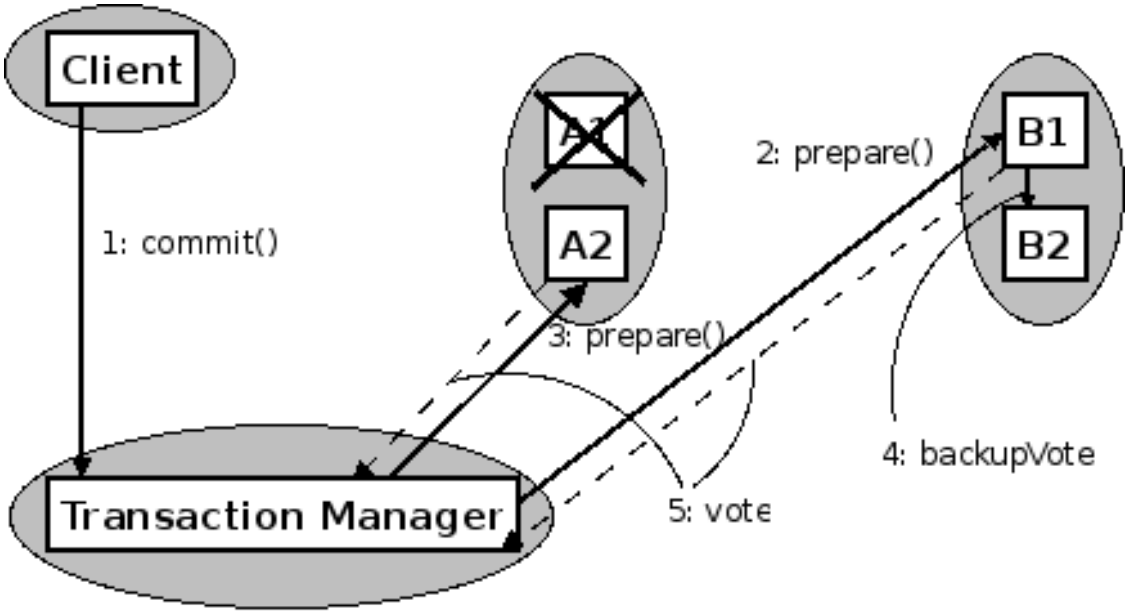
— Server B is in an inconsistent state

Failover



- Server B is in an inconsistent state
- Solution: Abort all transactions interacting with a replica that fails

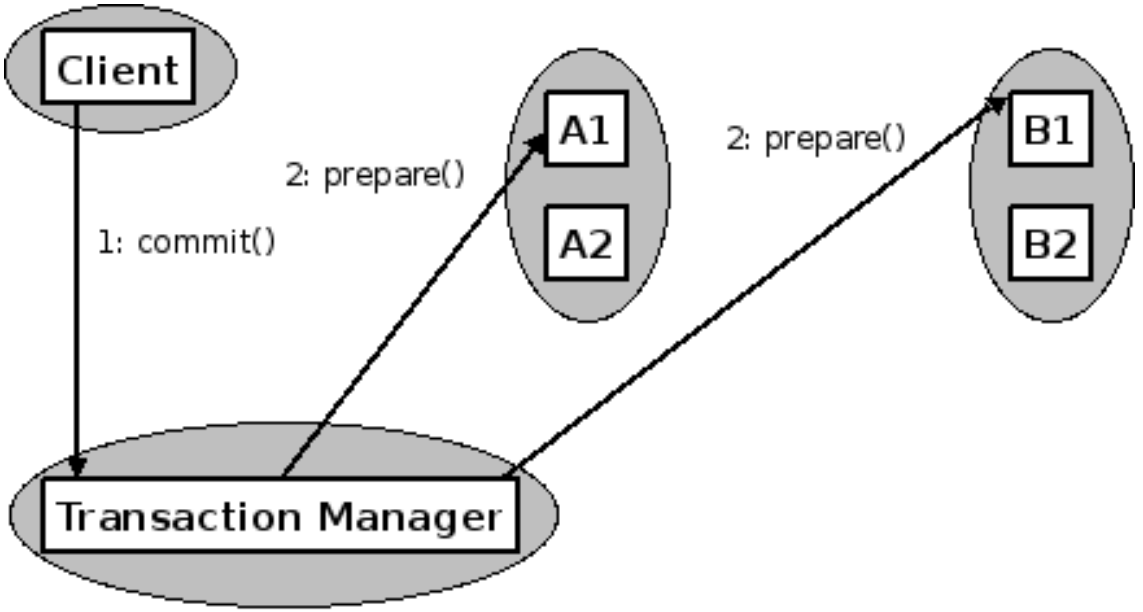
Failover



- Server B is in an inconsistent state
- Solution: Abort all transactions interacting with a replica that fails
- => Which transactions are caught in a failover

Solution

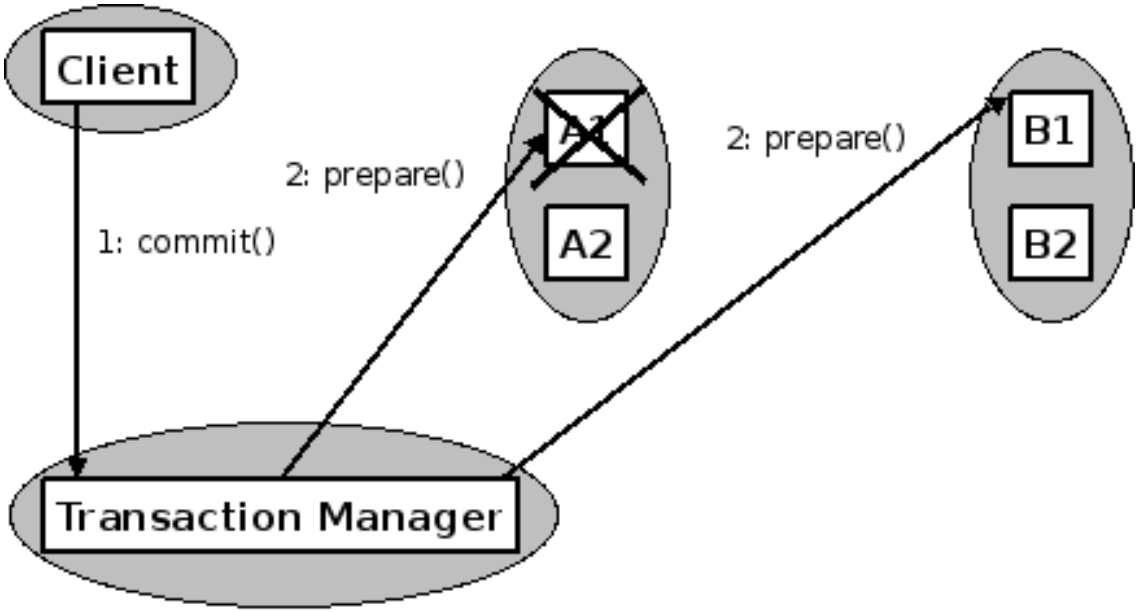
- Avoid failover of prepare messages
- Special proxies for the TM
- TM informed about failovers



NTNU

Solution

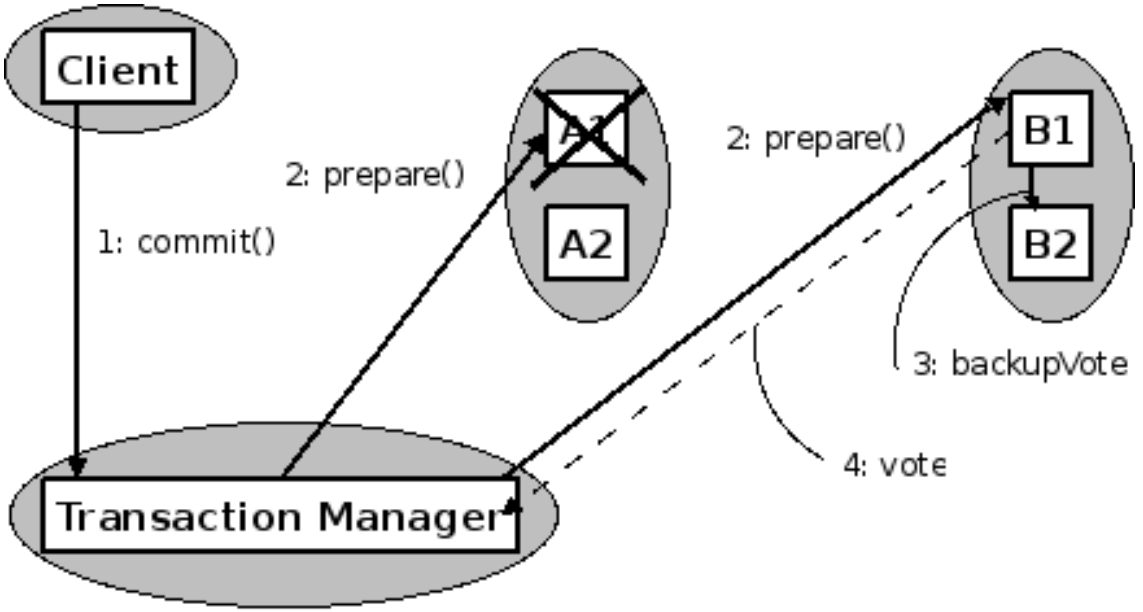
- Avoid failover of prepare messages
- Special proxies for the TM
- TM informed about failovers



NTNU

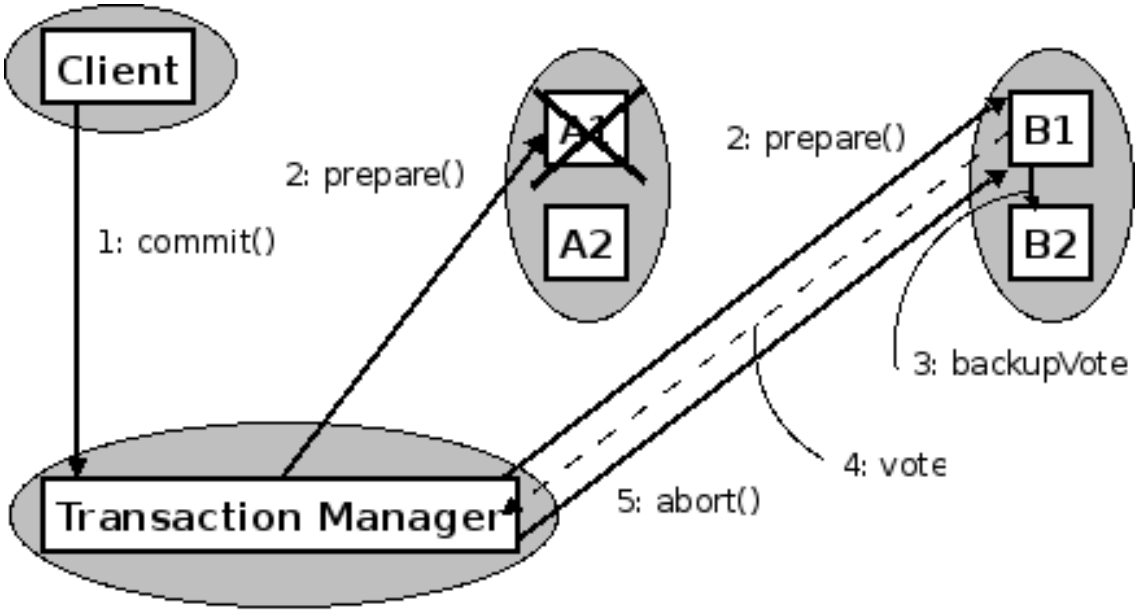
Solution

- Avoid failover of prepare messages
- Special proxies for the TM
- TM informed about failovers

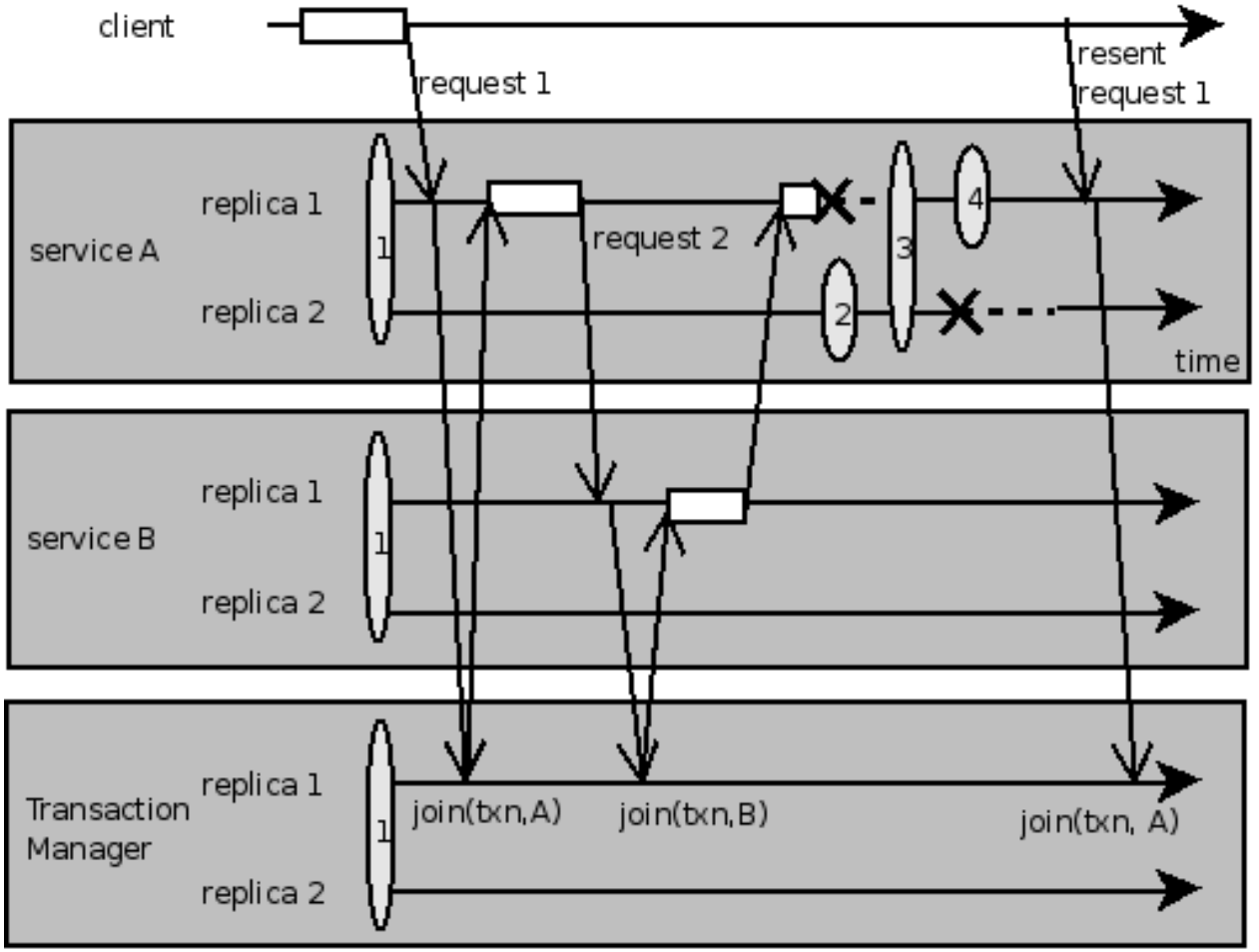


Solution

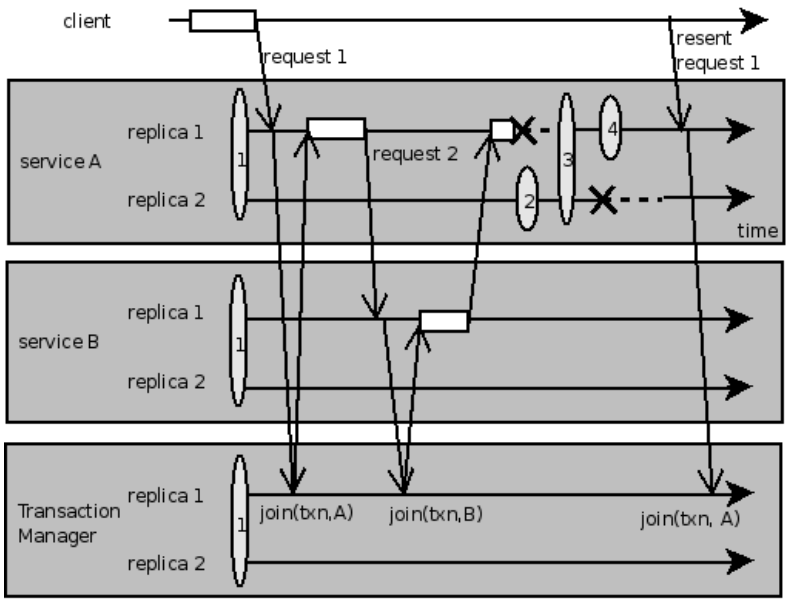
- Avoid failover of prepare messages
- Special proxies for the TM
- TM informed about failovers



Double Failovers

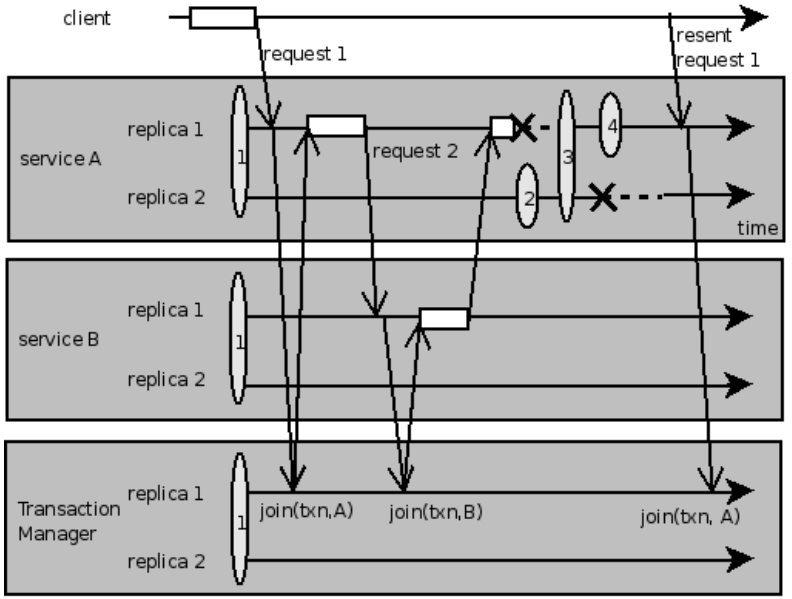


Double Failovers



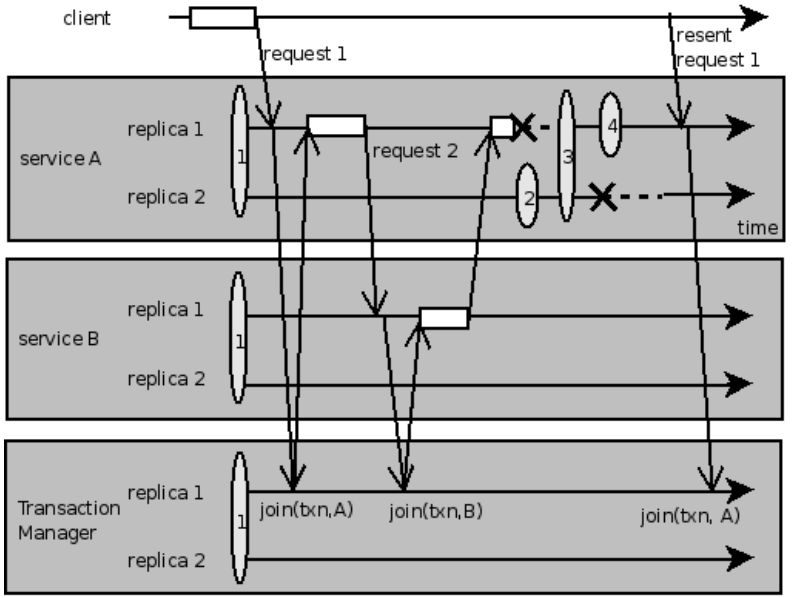
— Second join message from A looks like a resend of first

Double Failovers



- Second join message from A looks like a resend of first
- Solution: join(txnid, group) -> join(txnid, group, viewid)

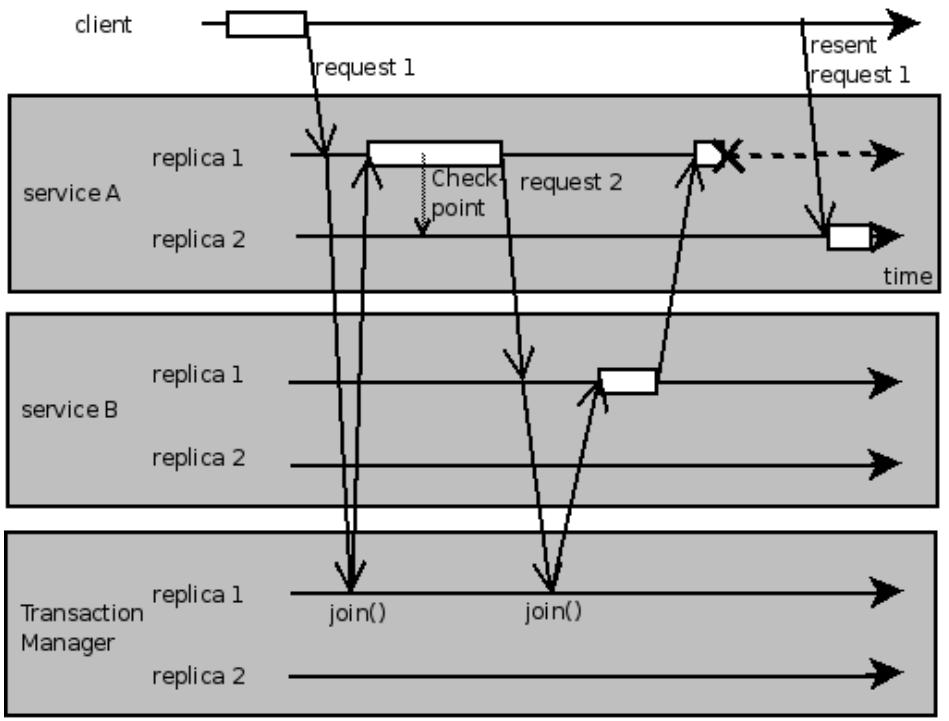
Double Failovers



- Second join message from A looks like a resend of first
- Still need to break replication transparency

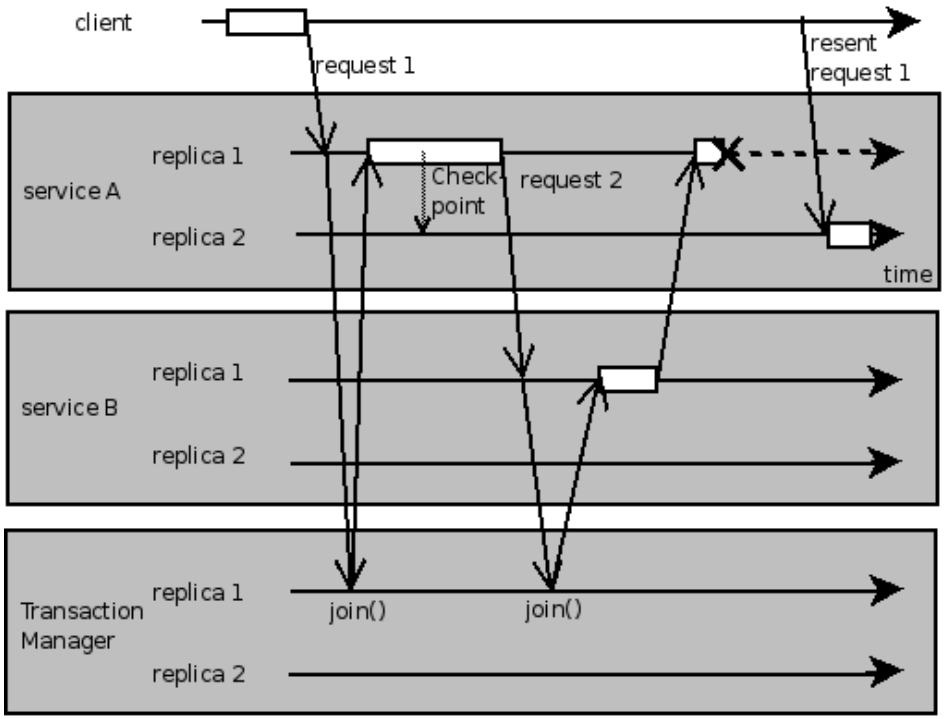
Checkpoints

— Can be taken anytime



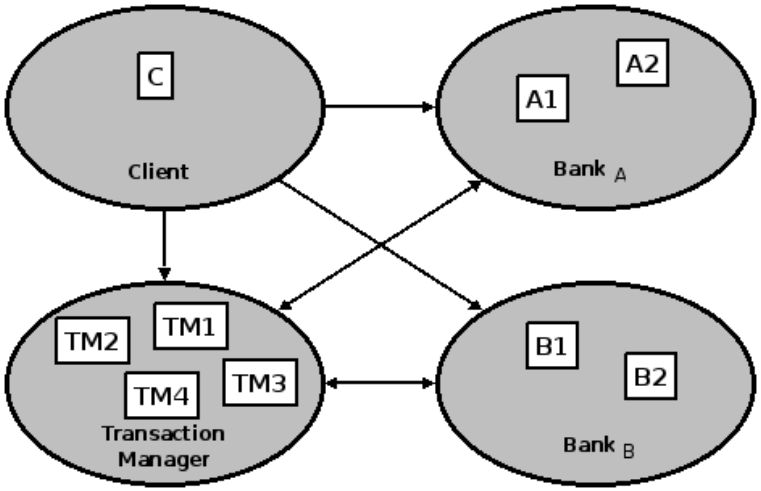
Checkpoints

- Can be taken anytime



- Prepare message sent to new primary A2 -> request 2 is not aborted
- Break replication transparency

Test Environment



- Hardware: 5 nodes, 100Mbit Ethernet, Dual AMD MP 1600+ 1,4GHz CPUs, 1024 MB RAM
- Software: Linux kernel 2.6, Java version 1.5.0, Jgroup/ARM GMS, Jini Transaction Service
- 500 transactions, measured client side response time

NTNU

Test Results

Test run	Description	Average (ms)
Nonreplicated system		
1	1 TM and 2 banks	47

Test Results

Test run	Description	Average (ms)
Nonreplicated system		
1	1 TM and 2 banks	47
Passive replication of the TM		
2	2 passive TMs and 2 banks	77
3	3 passive TMs and 2 banks	92
4	4 passive TMs and 2 banks	106

Test Results

Test run	Description	Average (ms)
Nonreplicated system		
1	1 TM and 2 banks	47
Passive replication of the TM		
2	2 passive TMs and 2 banks	77
3	3 passive TMs and 2 banks	92
4	4 passive TMs and 2 banks	106
Fully replicated system		
5	1 TM and 2x2 banks	75
6	2 passive TMs and 2x2 banks	148
7	3 passive TMs and 2x2 banks	164

NTNU

Test Results

Test run	Description	Average (ms)	Delay (%)
Nonreplicated system			
1	1 TM and 2 banks	47	0
Passive replication of the TM			
2	2 passive TMs and 2 banks	77	64
3	3 passive TMs and 2 banks	92	96
4	4 passive TMs and 2 banks	106	126
Fully replicated system			
5	1 TM and 2x2 banks	75	60
6	2 passive TMs and 2x2 banks	148	215
7	3 passive TMs and 2x2 banks	164	249

Conclusions

- Replication - High availability
- Transactions - Consistency
- Integrated without enforcing replica determinism
- Break replication transparency -> no orphan requests
- Open-source prototype:
  - Implementation too slow for RT systems
  - Jini and Jgroup are not tuned for performance

NTNU

Further Work

- Implement in a tuned environment
- Other transaction models
- Handle all failure scenarios of 2PC
- Replica management (create, restart, update)

NTNU