



## Transactions – Atomicity

**All or nothing:**

• A transaction either completes successfully, and the effects of all of its operations are recorded in the objects, or (if it fails or is deliberately aborted) it has no effect at all. (Note: [1] must handle crashes; [2] durable (permanent storage))

**Isolation:**

• Each transaction must be performed without interference from other transactions (i.e., the intermediate effects of a transaction must not be visible to other transactions).

**Serial equivalence (serializable):**

• Transactions are allowed to execute concurrently if they would have the same effect as a serial execution.

**“ACID” = atomicity, consistency, isolation, and durability**

```
Transaction{
  a.withdraw(100);
  b.deposit(100);
  c.withdraw(200);
  b.deposit(200);
}
```

CS451: Distributed Systems (Spring 2007)

## Transactions - the lost update problem

deposit, withdraw, getBalance, setBalance: atomic actions  
All accounts (a,b,c) start at \$200

Transaction T:	Transaction U:
<code>balance = b.getBalance();</code> <code>b.setBalance(balance*1.1);</code> <code>a.withdraw(balance/10)</code>	<code>balance = b.getBalance();</code> <code>b.setBalance(balance*1.1);</code> <code>c.withdraw(balance/10)</code>
<code>balance = b.getBalance();</code> \$200  <code>b.setBalance(balance*1.1);</code> \$220 <code>a.withdraw(balance/10)</code> \$80	<code>balance = b.getBalance();</code> \$200 <code>b.setBalance(balance*1.1);</code> \$220  <code>c.withdraw(balance/10)</code> \$280

CS451: Distributed Systems (Spring 2007)

## Transactions - the inconsistent retrievals problem

deposit, withdraw, getBalance, setBalance: atomic actions  
All accounts (a,b,c) start at \$200

Transaction V:	Transaction W:
<code>a.withdraw(100)</code> <code>b.deposit(100)</code>	<code>aBranch.branchTotal()</code>
<code>a.withdraw(100);</code> \$100  <code>b.deposit(100)</code> \$300	<code>total = a.getBalance()</code> \$100 <code>total = total+b.getBalance()</code> \$300 <code>total = total+c.getBalance()</code> ⋮

CS451: Distributed Systems (Spring 2007)

## Serial equivalence - basics

• If “correct” transactions are done one at a time in some order, the combined effect will also be correct.

**A serially equivalent interleaving:**

An interleaving of the operations of transactions in which the combined effect is the same as if the transactions had been performed one at a time in some order.

• Serial equivalence prevents lost updates and inconsistent retrievals.

CS451: Distributed Systems (Spring 2007)

## Serial equivalence – conflicting operations

• Two operations **conflict**: “their combined effect depends on the order in which they are executed.”

• Two transactions are **serially equivalent** iff all pairs of conflicting operations of the two transactions are executed in the same order at all of the objects they both access.

**Conflicts:**

- Read-read : no
- Read-write: yes
- Write-write: yes

CS451: Distributed Systems (Spring 2007)

## Concurrency Control Protocols

**Attempt to serialize transactions**

**3 approaches:**

- **Locking**
  - Server sets a lock (labeled with transaction ID) on each object just before access, removes locks when transaction completes (can cause deadlock)
- **Optimistic concurrency control**
  - Allowed to proceed; upon commit, server checks for conflicts with concurrent transactions; if so, transaction is aborted (“try again”)
- **Timestamp ordering**
  - Server records time of most recent reading/writing of each object; for each operation, “the timestamp of the transaction is compared with that of the object to determine if can be done immediately, delayed, or rejected”

CS451: Distributed Systems (Spring 2007)

## Chpt 15: Replication - basics

•Replication of data: Maintenance of copies of data at multiple computers.

•Goals:

- Enhanced performance.
- Increased availability.
- Fault tolerance.

•Some potential requirements:

- Replication transparency.
- Consistency: if a copy is modified, how and when the others are updated determines "the price of replication"

CS451: Distributed Systems (Spring 2007)

## More Replication Basics

•Simple math: if two independent servers, each with 5% chance of failing, then availability is:

$$1 - \text{prob (ALL failed)} = 1 - .05^2 = 99.75\%$$

•Diff between replication and caches?

- Caches might not necessarily include ALL objects of interest

CS451: Distributed Systems (Spring 2007)

## Replication – system model

•Asynchronous system, processes fail only by crashing, no network partitions.

•Each logical object implemented by a collection of physical objects / replicas.

•Replicas held by replica managers:

- Apply operations recoverably.
- Are state machines (deterministic).

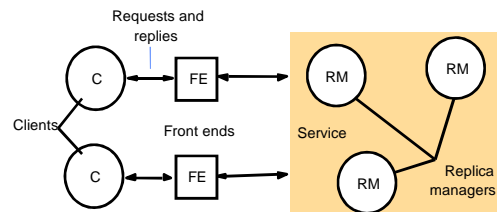
•A collection of replica managers provides a service to clients.

•Each client's requests are handled by a front end.

•Note: we're not going to go over the "Group Communication" material

CS451: Distributed Systems (Spring 2007)

## Replication - basic architectural model



CS451: Distributed Systems (Spring 2007)

## Replication – phases in handling a single request

1. **Request:** Front end issues request.
2. **Coordination:** Replica managers coordinate in preparation for executing the request consistently, including ordering decisions (FIFO, causal, total).
3. **Execution:** Replica managers execute the request.
4. **Agreement:** Replica managers reach consensus on the effect of the request.
5. **Response:** One or more replica managers responds to the front end.

CS451: Distributed Systems (Spring 2007)

## Replication for Performance

- "the cure may be worse than the disease!"
- Solution: loosen the consistency constraints

CS451: Distributed Systems (Spring 2007)

### Fault Tolerance Basics

- Availability:** probability that the system is operating correctly at any given moment
- Reliability:** property that a system can run continuously without failure (“it will continue to work without interruption over a long period of time”)
- System **fails** when it cannot meet its promises.
- An **error** is part of the system state that may lead to failure.
- The cause of an error is a **fault**.
- TMR: Triple Modular Redundancy: Failure masking by redundancy**

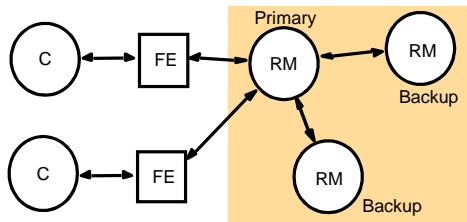
CS451: Distributed Systems (Spring 2007)

### Fault tolerance - basics

- How to provide a service that is correct despite up to  $f$  process failures, by replicating data and functionality at replica managers.
- Reliable communication, no partitions, replica managers fail only by crashing.**
- Intuitively, a service based on replication is correct if:**
  - it keeps responding despite failures, and
  - if clients cannot tell the difference between the service they obtain from an implementation with replicated data and one provided by a single correct replica manager.

CS451: Distributed Systems (Spring 2007)

### Fault tolerance - the passive (primary-backup) model



CS451: Distributed Systems (Spring 2007)

### Fault tolerance –sequence of events in passive (primary-backup) replication

- Request:** The front end issues the request, containing a unique identifier, to the primary replica manager.
- Coordination:** The primary takes each request atomically, in the order in which it receives it. It checks the unique identifier, in case it has already executed the request and if so, it simply re-sends the response.
- Execution:** The primary executes the request and stores the response.
- Agreement:** If the request is an update, then the primary sends the updated state, the response and the unique identifier to all the backups. The backups send acks.
- Response:** The primary responds to the front end, which hands the response back to the client.

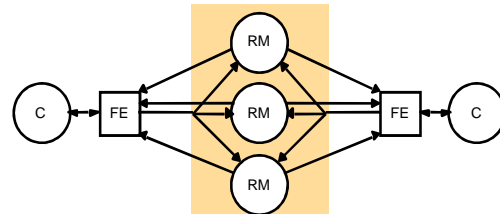
CS451: Distributed Systems (Spring 2007)

### Fault tolerance – primary fails in passive (primary-backup) replication

- Actions:**
  - The primary is replaced by a unique backup.
  - Ensure that the replica managers that survive agree on which operations had been performed at the point when the replacement primary takes over.
- Techniques:**
  - Organize replica managers as a group.
- Linearizability holds.**

CS451: Distributed Systems (Spring 2007)

### Fault tolerance - active replication



CS451: Distributed Systems (Spring 2007)

### Fault tolerance – sequence of events in active replication (1/2)

- Request:** The front end attaches a unique identifier to the request and multicasts it to the group of replica managers, using a totally ordered, reliable multicast primitive. The front end waits for response.
- Coordination:** The group communication system delivers the request to every correct replica manager in the same (total) order.
- Execution:** Every replica manager executes the request. Since they are state machines and since requests are delivered in the same total order, correct replica managers all process the requests identically. The response contains the client's unique request identifier.

---

CS451: Distributed Systems (Spring 2007)

### Fault tolerance – sequence of events in active replication (2/2)

- Agreement:** No agreement phase is needed, because of the multicast delivery semantics.
- Response:** Each replica manager sends its response to the front end. The number of replies that the front end collects depends upon the failure assumptions and on the multicast algorithm.
- Sequential consistency, but not linearizability, holds.**

---

CS451: Distributed Systems (Spring 2007)