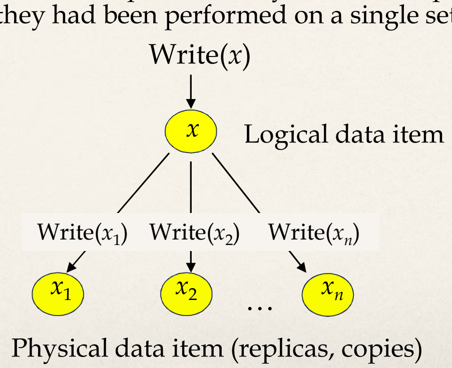
Data Replication

- Benefits of Replication
  - System Availability
    - May remove single points of failure by replicating data
  - Performance
    - Replication enables us to locate the data closer to their access point thereby reducing response point
  - Scalability
    - Replication allows for a way to support systems growth with acceptable response time
  - Application requirements
    - Some applications may wish to maintain multiple data copies as part of their operational specifications

Execution Model

- There are physical copies of logical objects in the system.
- Operations are specified on logical objects, but translated to operate on physical objects.
- One-copy equivalence
  - The effect of transactions performed by clients on replicated objects should be the same as if they had been performed on a single set of objects.



Issues in Design of Replication Protocols

- 1. Database design:
  - Distributed databases may be fully or partially replicated.
- 2. Database consistency
  - Mutual consistency:
    - A replicated database is said to be in mutually consistent (or strong consistent) state if all replicas of each of its data items have identical values.
  - Weak consistency
    - Eventual consistency: the copies are not identical when update transaction completes, but they eventually converge to the same value

Issues in Design of Replication Protocols

- 3. Where updates are performed
  - Centralized
  - Distributed
- 4. Update propagation techniques – how do we propagate updates to one copy to the other copies?
  - Eager
  - Lazy
- 5. Degree of replication transparency

Transactional Consistency

- How can we guarantee that the global execution history over replicated data is serializable?
- One-copy serializability (1SR)
  - The effect of transactions performed by clients on replicated objects should be the same as if they had been performed *one at-a-time* on a single set of objects.
- Weaker forms are possible
  - Snapshot isolation
  - RC-serializability

Example 1

Site A		Site B		Site C
x		x, y		x, y, z
T_1:	$x \leftarrow 20$ Write(x) Commit	T_2:	Read(x) $y \leftarrow x+y$ Write(y) Commit	T_3: Read(x) Read(y) $z \leftarrow (x*y)/100$ Write(z) Commit

Consider the three histories:

H_A={$W_1(x_A)$, C_1}
H_B={$W_1(x_B)$, C_1, $R_2(x_B)$, $W_2(y_B)$, C_2}
H_C={$W_2(y_C)$, C_2, $R_3(x_C)$, $R_3(y_C)$, $W_3(z_C)$, C_3, $W_1(x_C)$, C_1}

Global history non-serializable: H_B: $T_1 \rightarrow T_2$, H_C: $T_2 \rightarrow T_3 \rightarrow T_1$

Mutually consistent: Assume $x_A=x_B=x_C=10$, $y_B=y_C=15$, $z_C=7$ to begin; in the end $x_A=x_B=x_C=20$, $y_B=y_C=35$, $z_C=7$

Example 2

Site A	Site B
x	x

T_1: Read(x)
$x \leftarrow x$+5
Write(x)
Commit

T_2: Read(x)
$x \leftarrow x*$10
Write(x)
Commit

Consider the two histories:

H_A={$R_1(x_A)$,$W_1(x_A)$, C_1, $W_2(x_A)$, C_2}
H_B={$R_2(x_B)$, $W_2(x_B)$, C_2, $W_1(x_B)$, C_1}

Global history non-serializable: H_A: $T_1 \rightarrow T_2$, H_B: $T_2 \rightarrow T_1$
Mutually inconsistent: Assume x_A=x_B=1 to begin; in the end x_A=6, x_B=10

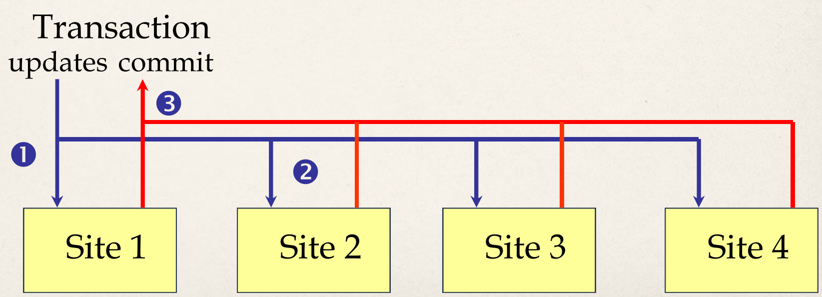
Update Management Strategies

- Depending on when the updates are propagated
  - Eager
  - Lazy
- Depending on where the updates can take place
  - Centralized
  - Distributed

	Centralized	**Distributed**
Eager		
Lazy		

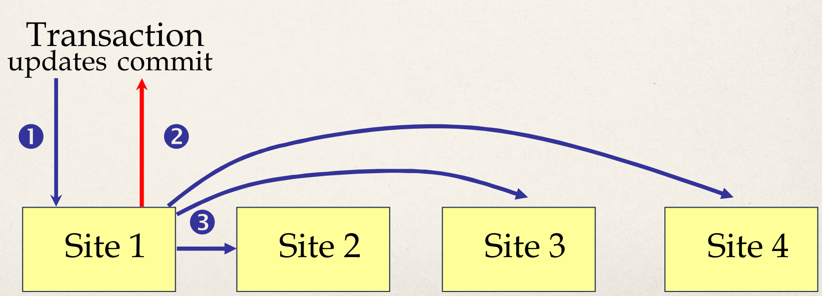
Eager Replication

- Changes are propagated within the scope of the transaction making the changes.



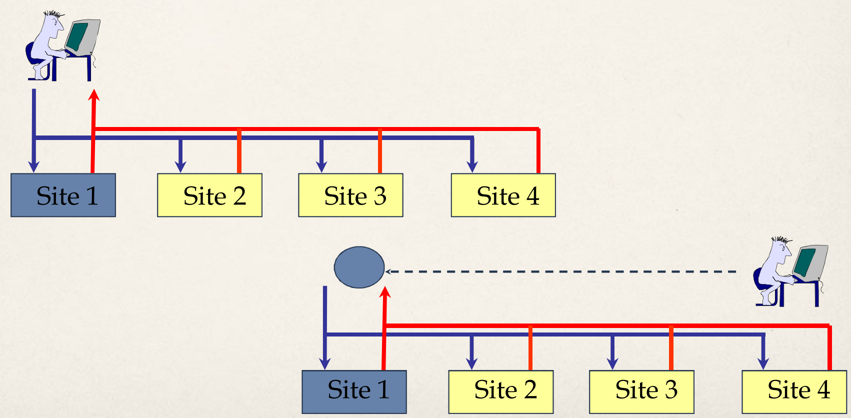
Lazy Replication

- Lazy replication first executes the updating transaction on one copy. After the transaction commits, the changes are propagated to all other copies (refresh transactions)



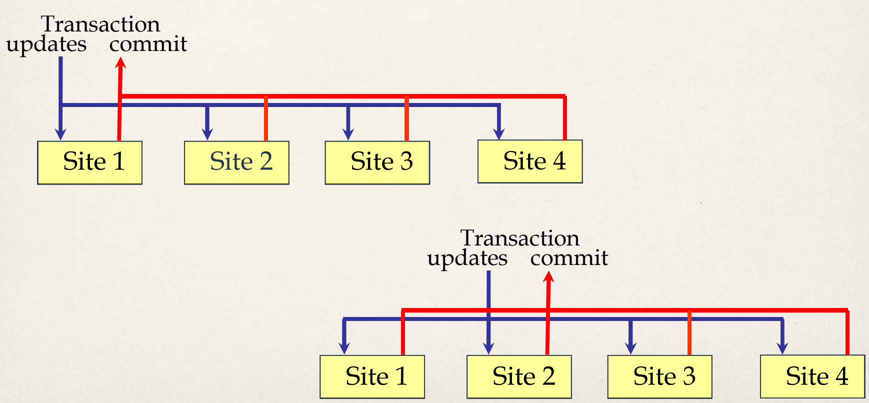
Centralized

- There is only one copy which can be updated (the master), all others (slave copies) are updated reflecting the changes to the master.



Distributed

- Changes can be initiated at any of the copies. That is, any of the sites which owns a copy can update the value of the data item.



Forms of Replication

Eagar

Eager

- \+ No inconsistencies (identical copies)
- \+ Reading the local copy yields the most up to date value
- \+ Changes are atomic
- − A transaction has to update all sites
  - − Longer execution time
  - − Lower availability

Lazy

- \+ A transaction is always local (good response time)
- − Data inconsistencies
- − A local read does not always return the most up-to-date value
- − Changes to all copies are not guaranteed
- − Replication is not transparent

Centralized

- \+ No inter-site synchronization is necessary (it takes place at the master)
- \+ There is always one site which has all the updates
- − The load at the master can be high
- − Reading the local copy may not yield the most up-to-date value

Distributed

- \+ Any site can run a transaction
- \+ Load is evenly distributed
- − Copies need to be synchronized