CS 5450

State Machine Replication

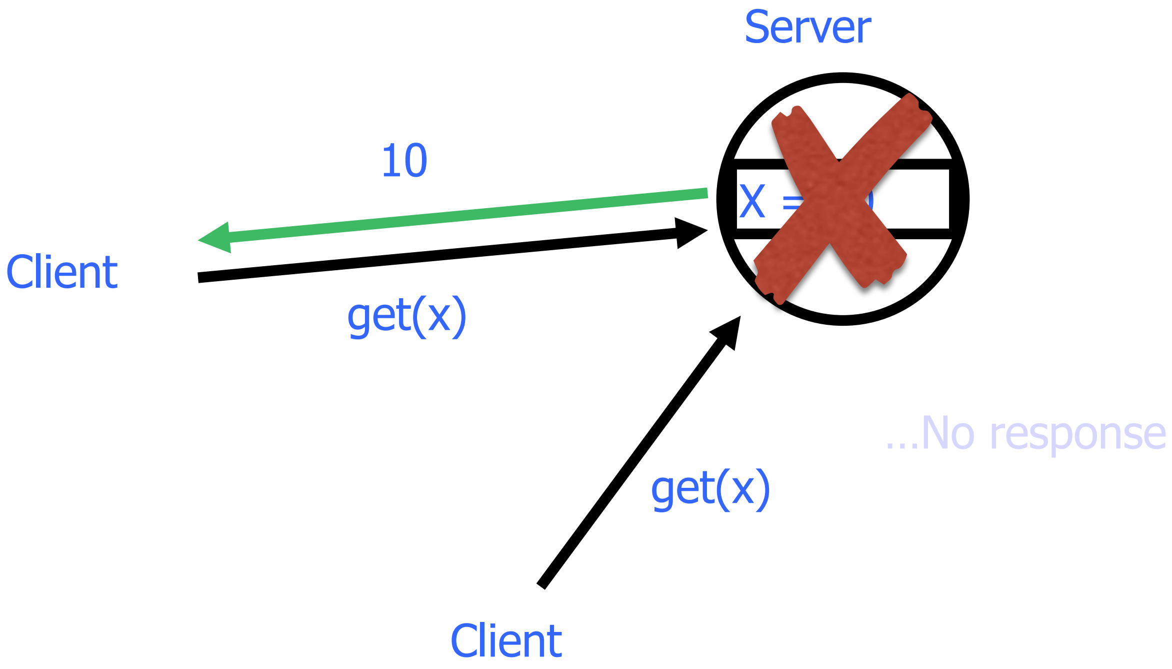
Key Ideas

To tolerate faults…

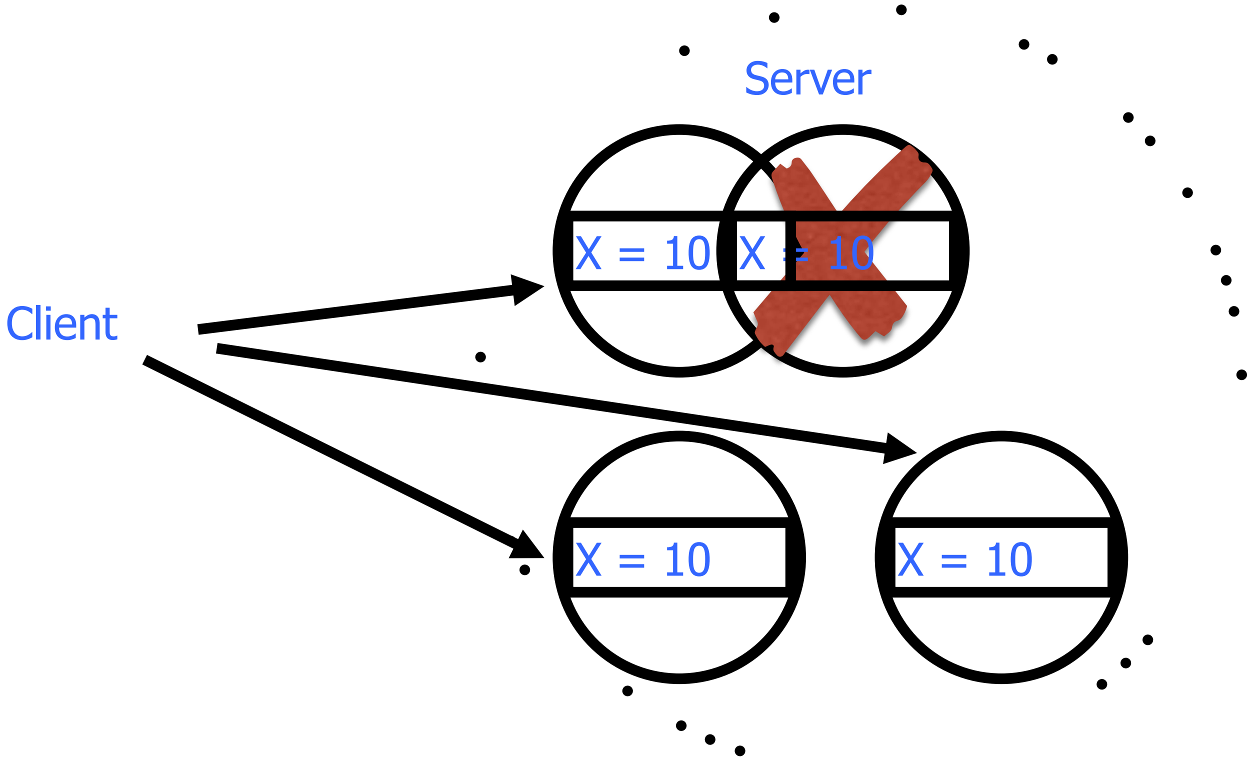
… replicate functionality!

- Can represent deterministic distributed system as replicated state machine (SMR)
- Each replica reaches the same conclusion about the system independently
- Examples of distributed algorithms that generically implement SMR
- Formal notion of fault-tolerance in SMR

Motivation



Motivation

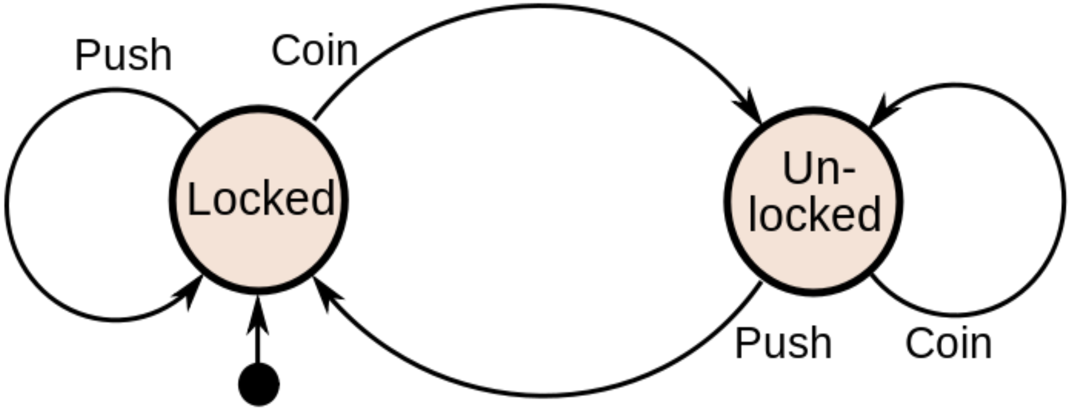


Motivation

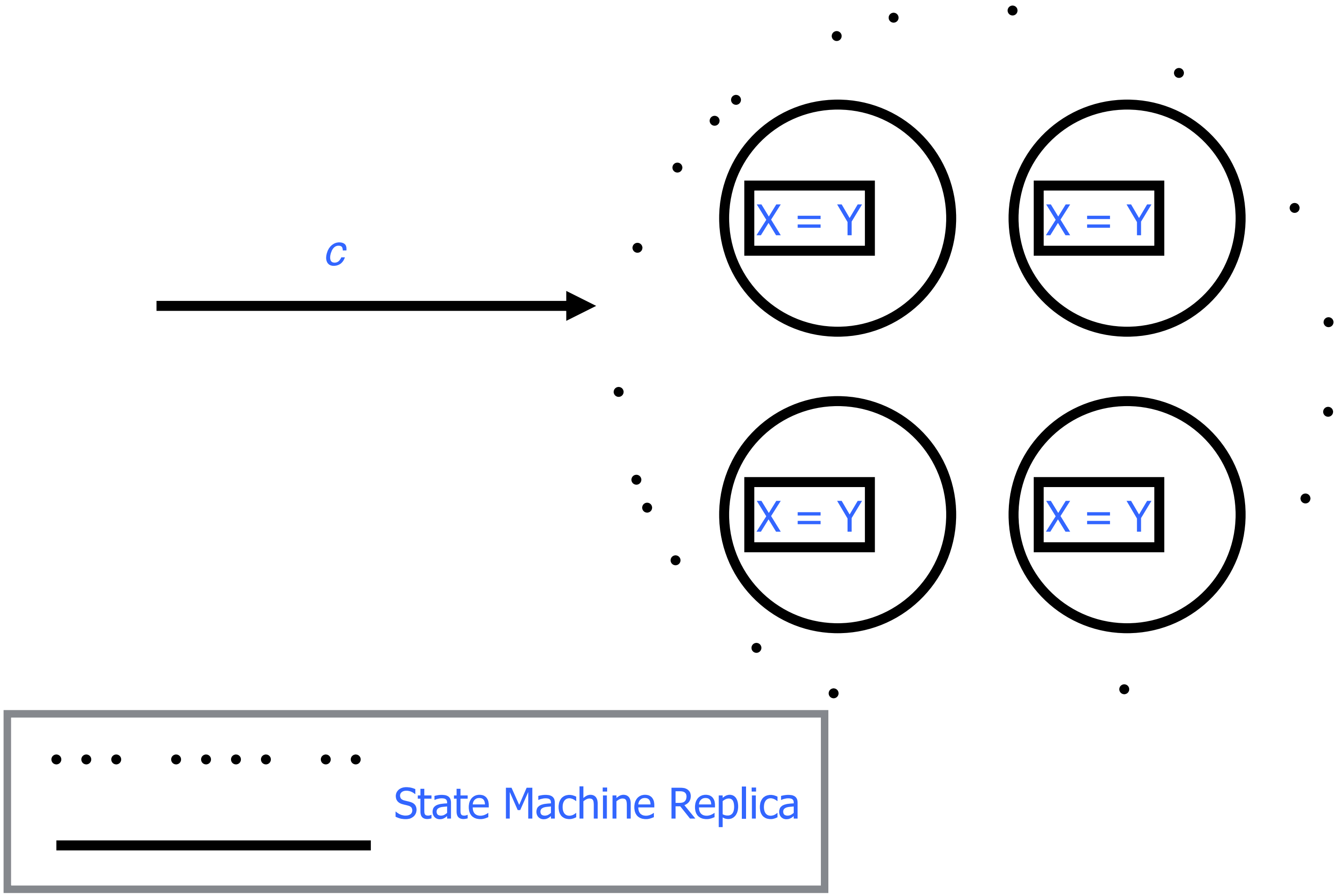
- Need replication for fault tolerance
  - Without replication, what happens to storage if disk fails? To a web service if network fals?
- Reason about failure tolerance
  - How badly can things go wrong and the system would continue to function?

State Machines

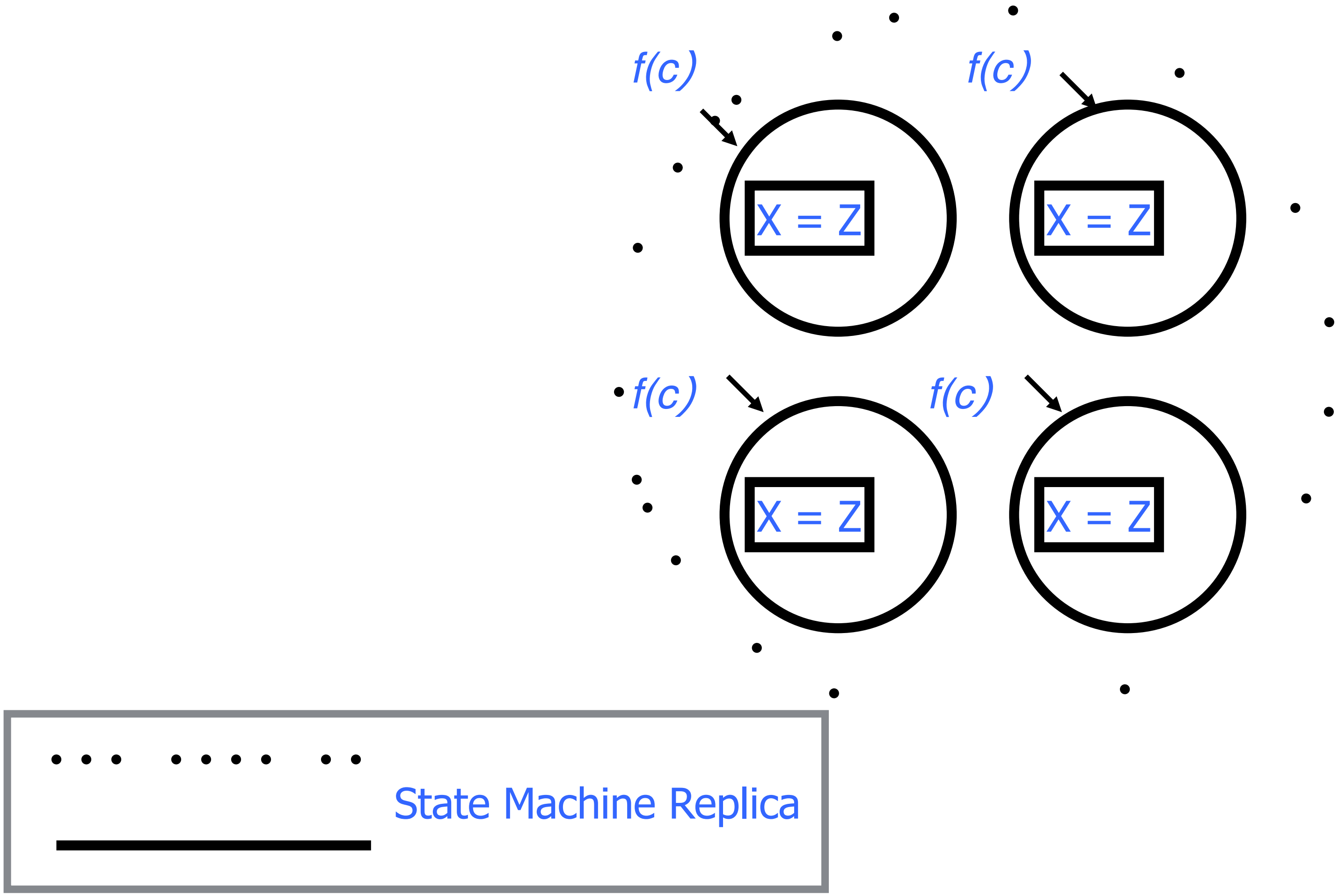
- State variables
- Deterministic commands



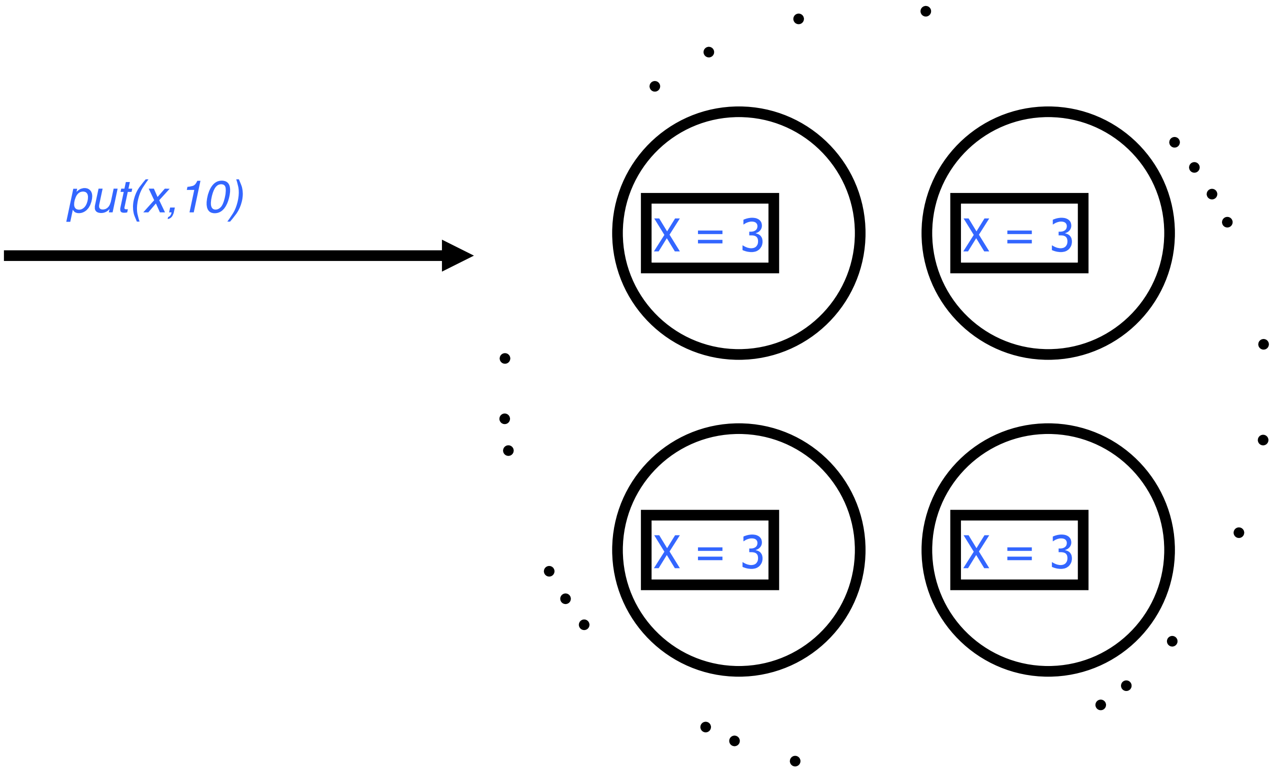
State Machine Replication



State Machine Replication



Write



put(x,10)

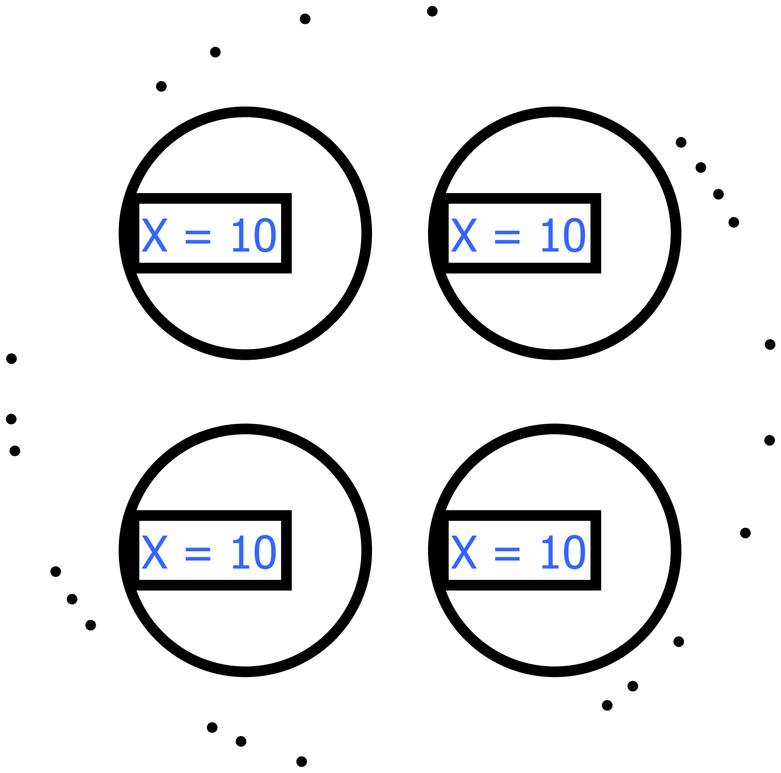
X = 3

X = 3

X = 3

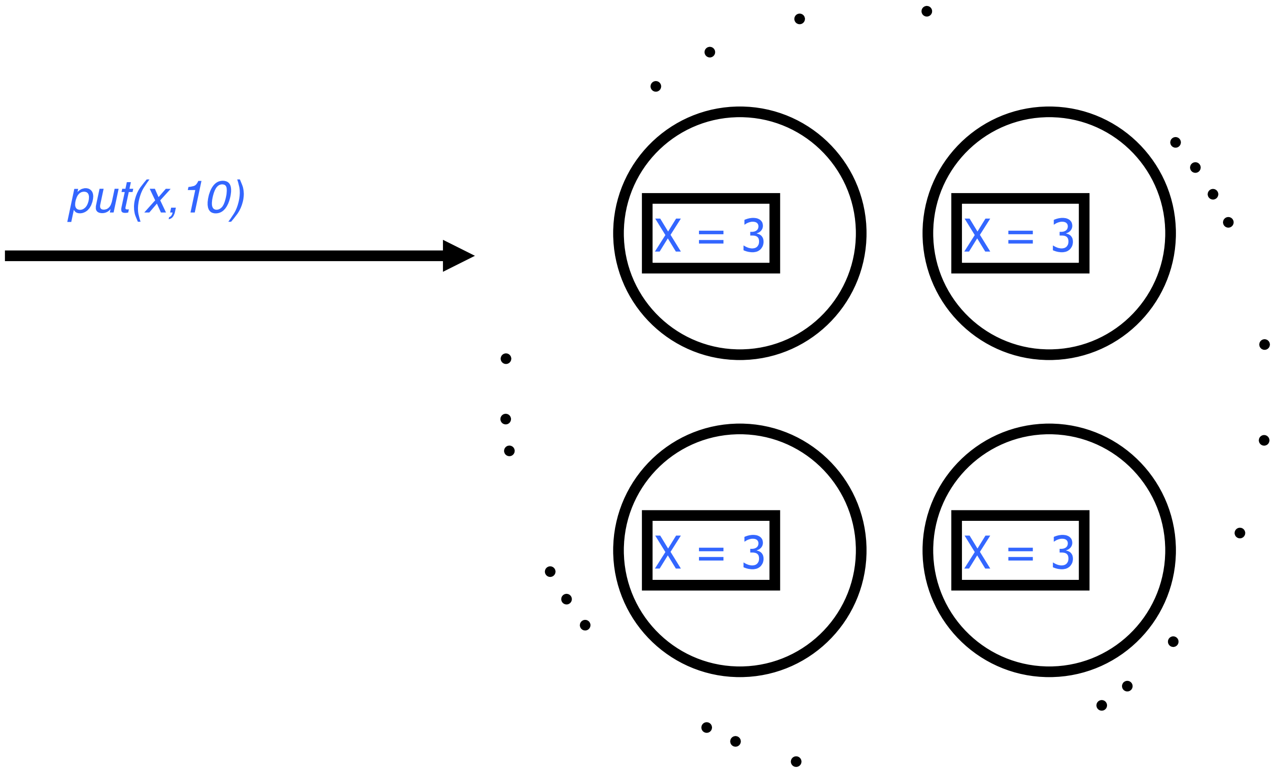
X = 3

After the Write

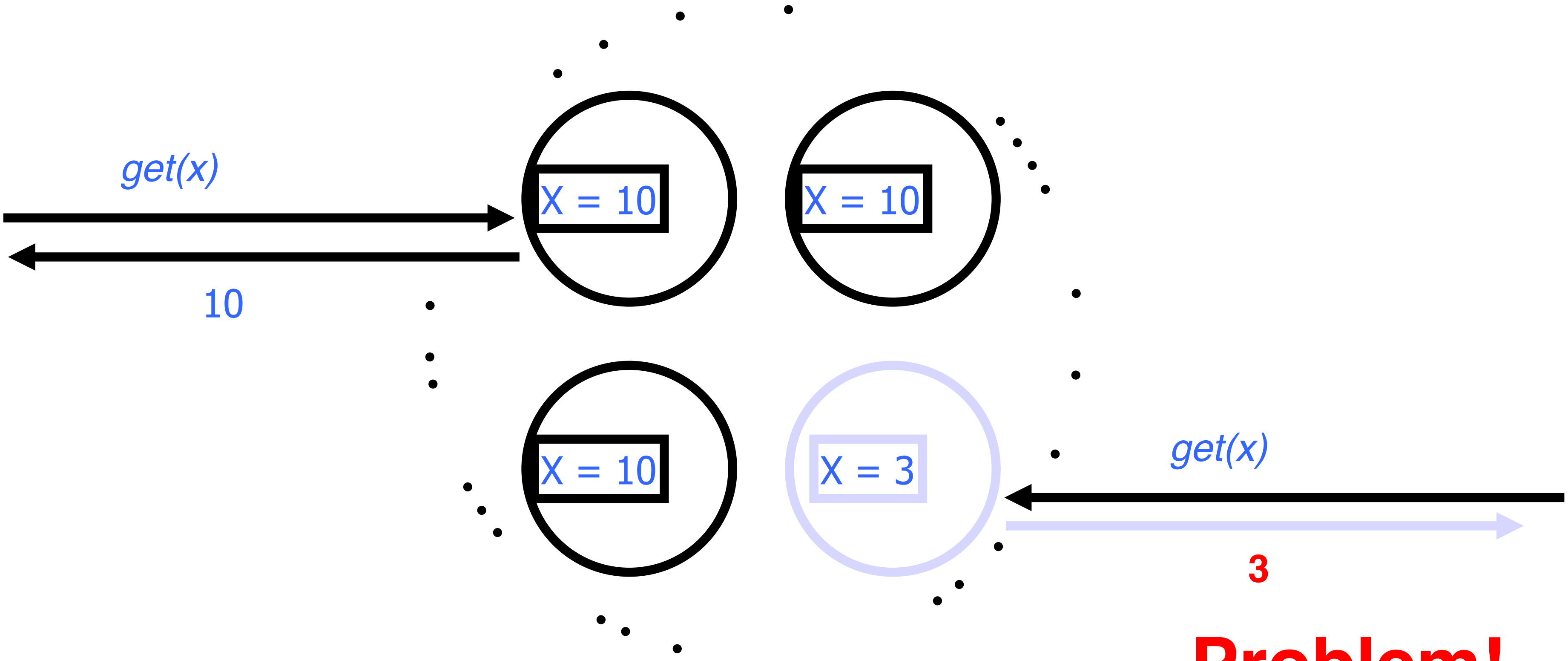


Great!

Write



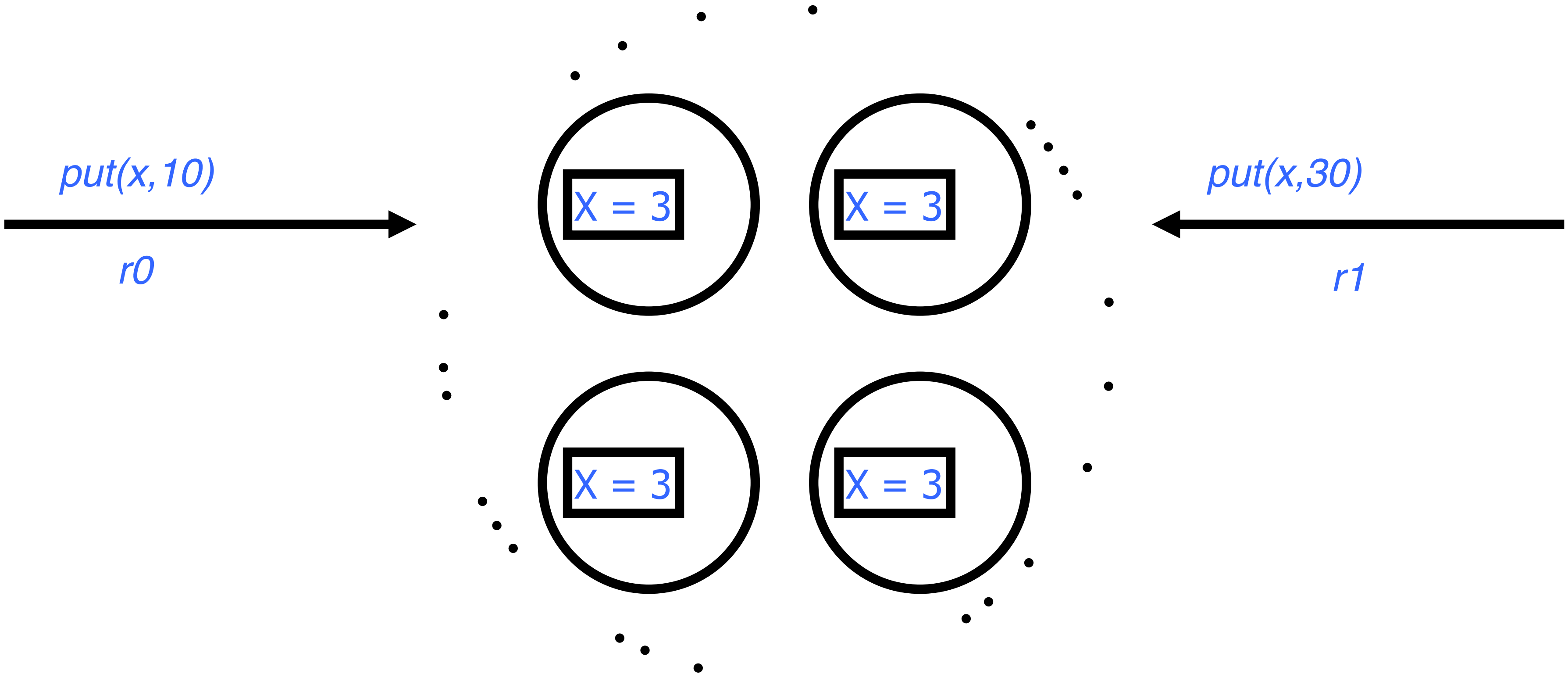
Need Agreement



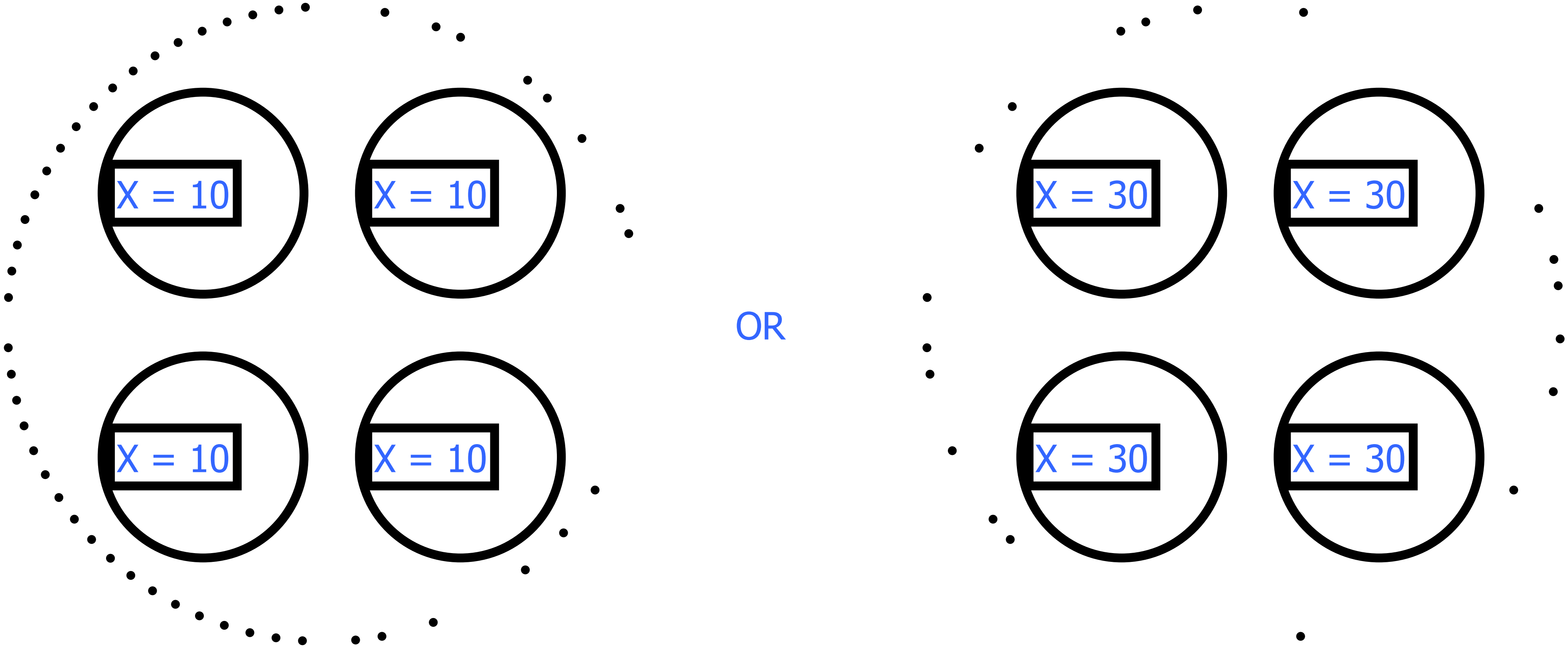
Problem!

Replicas need to **agree** which requests have been handled

Two Writes



Either Outcome is Fine



X = 10

X = 10

X = 10

X = 10

OR

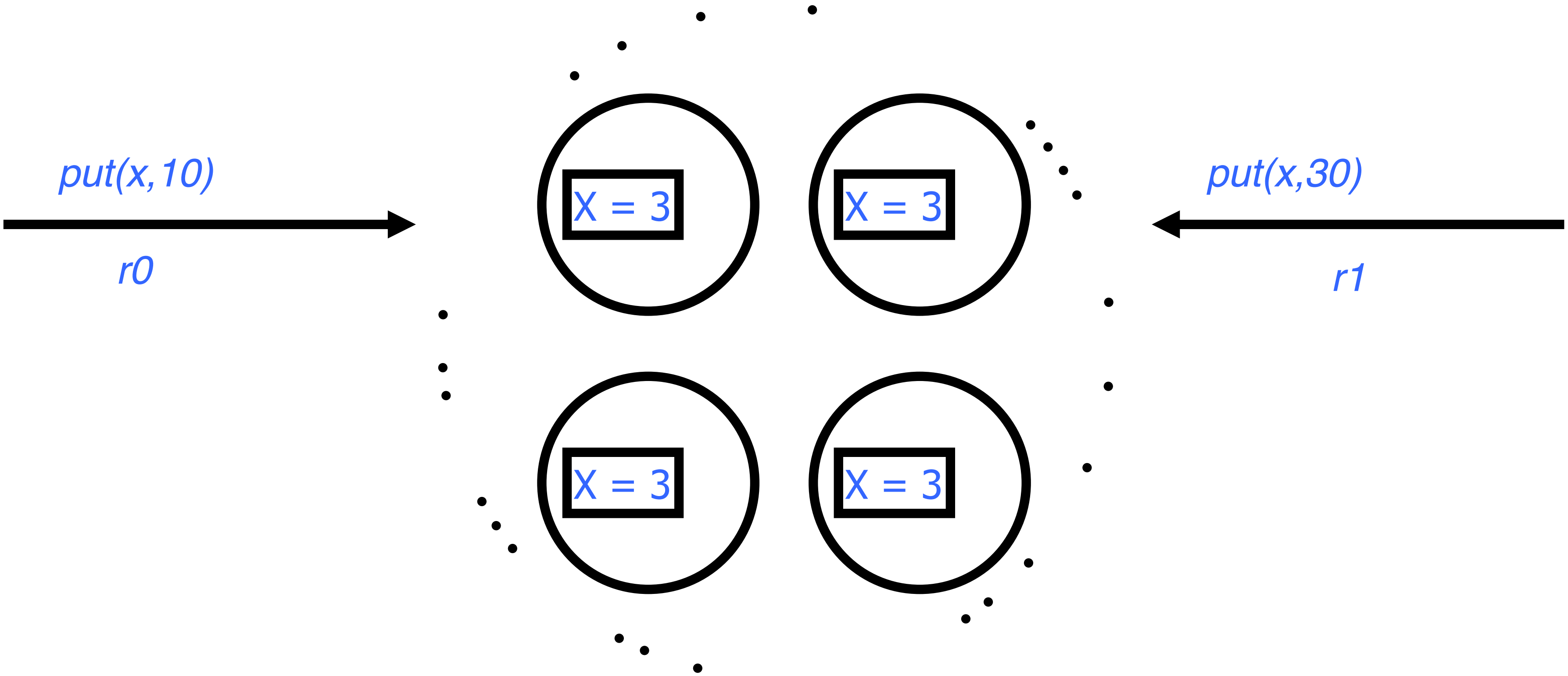
X = 30

X = 30

X = 30

X = 30

Order Matters



put(x,10)

r0

X = 3

X = 3

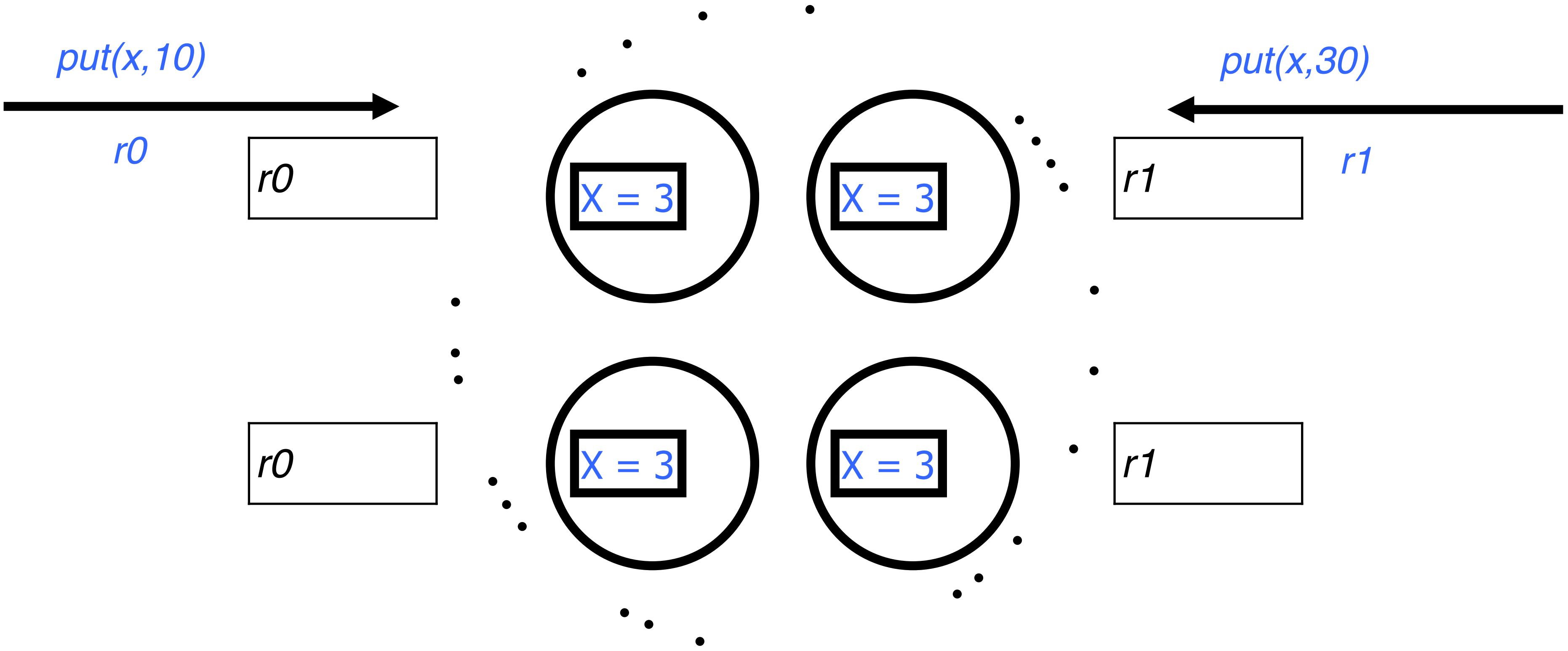
X = 3

X = 3

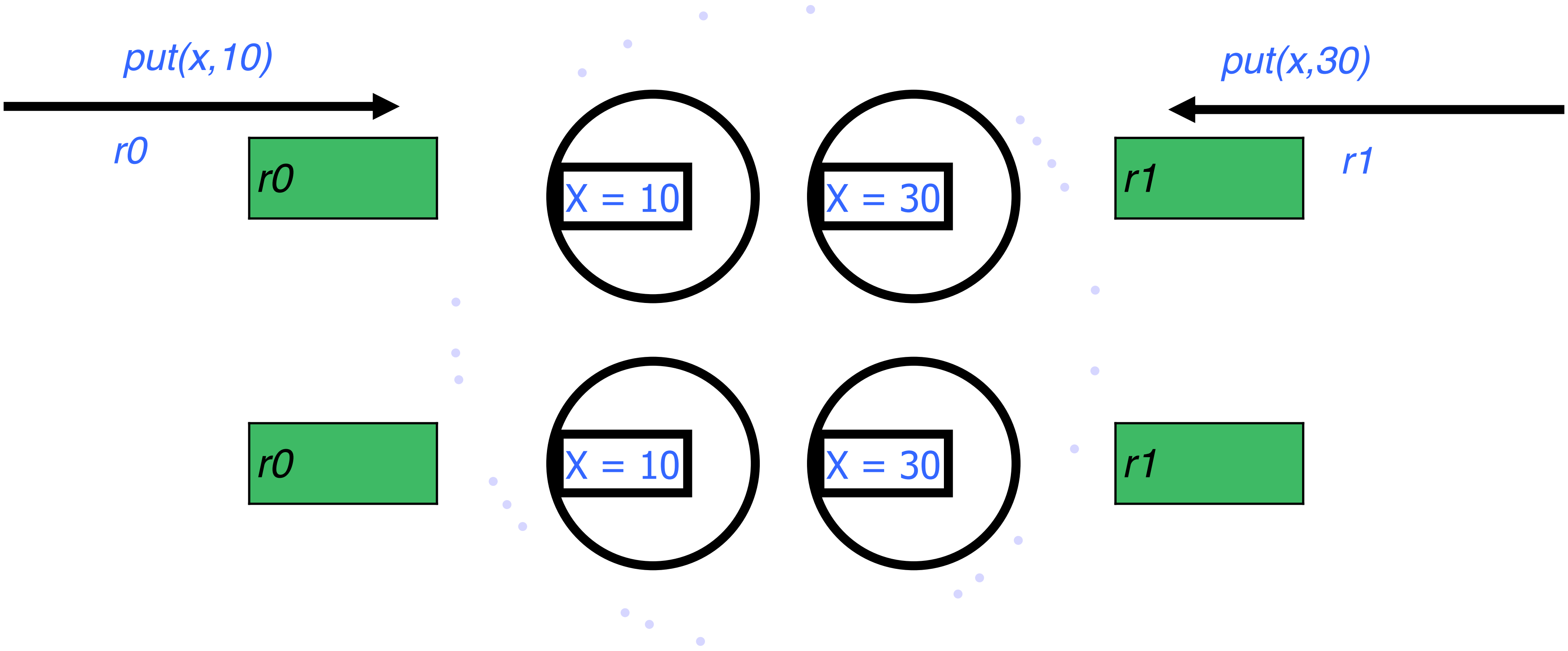
put(x,30)

r1

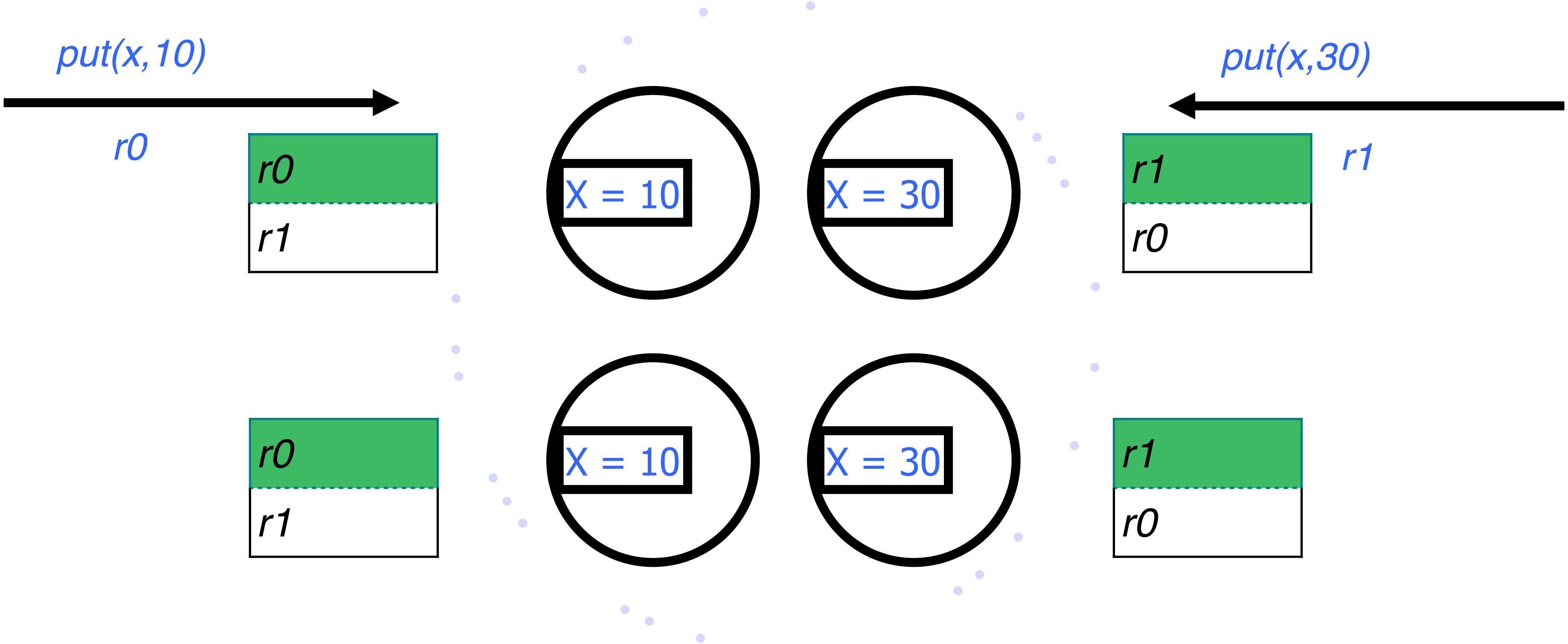
Order Matters



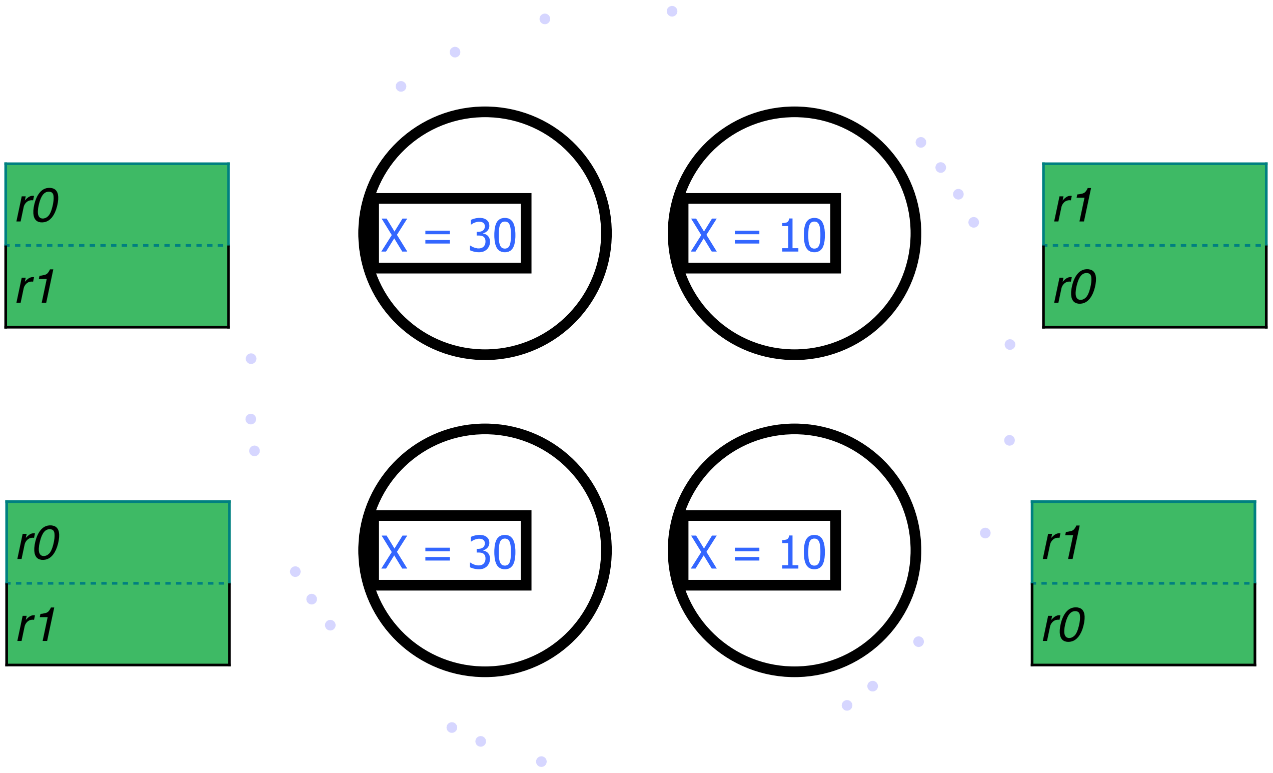
Order Matters



Order Matters



Order Matters



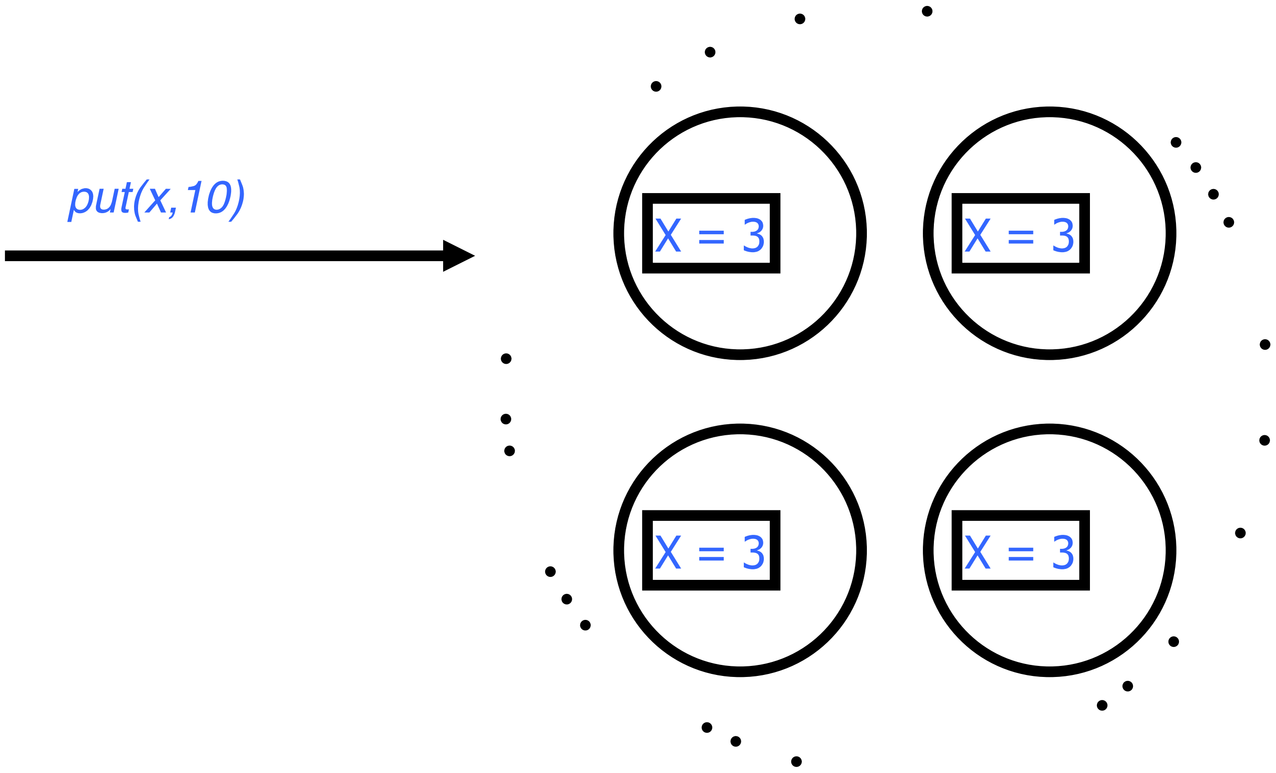
Replicas need to handle requests in the same **order**

Requirements

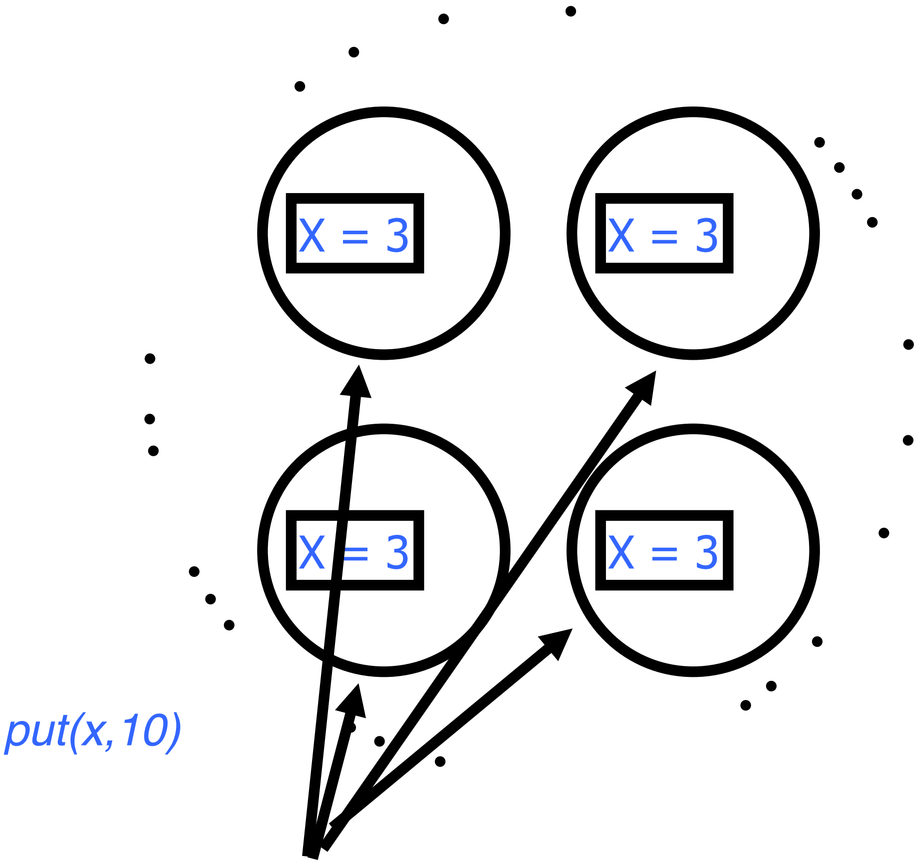
All non-faulty servers need…

- Agreement
  - Every replica needs to accept the same set of requests
- Order
  - All replicas process requests in the same relative order

Agreement



Agreement



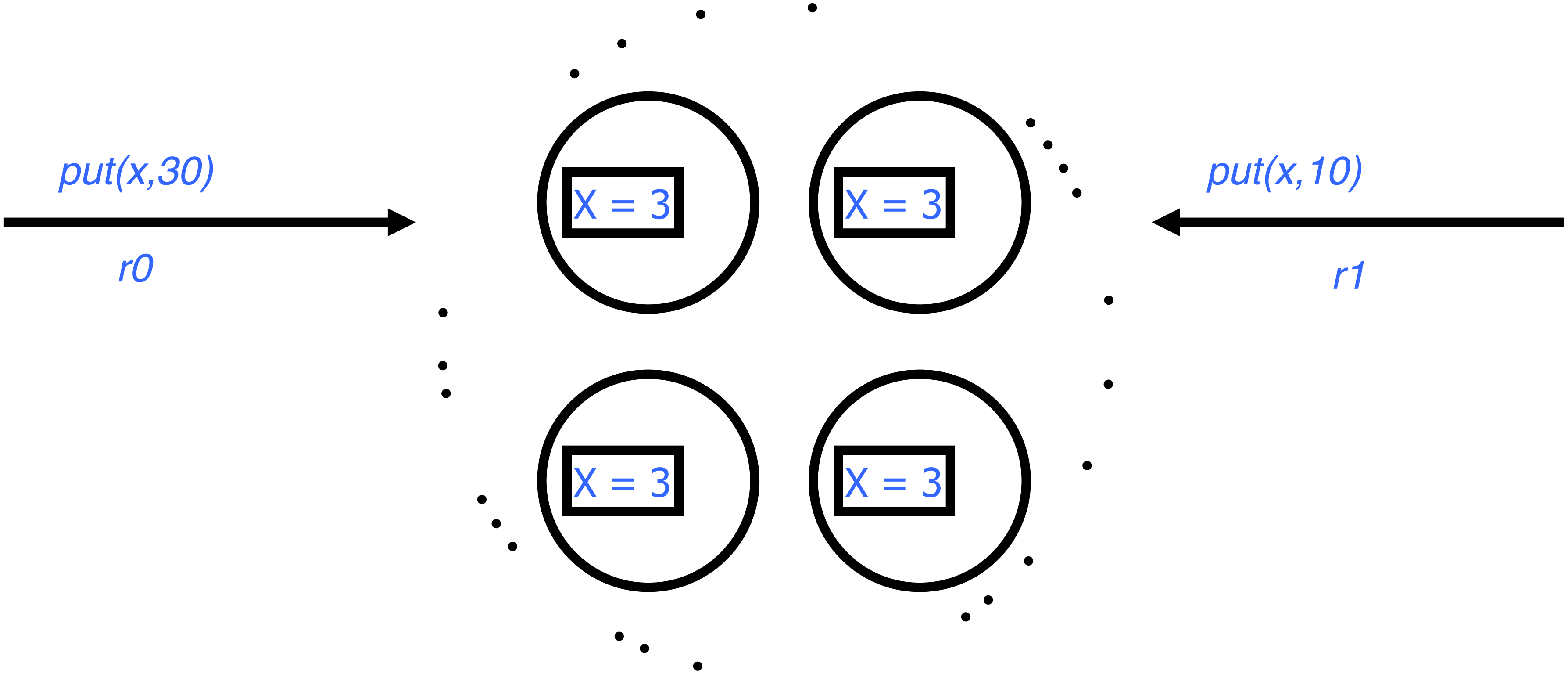
Non-faulty Transmitter

Idea for Order

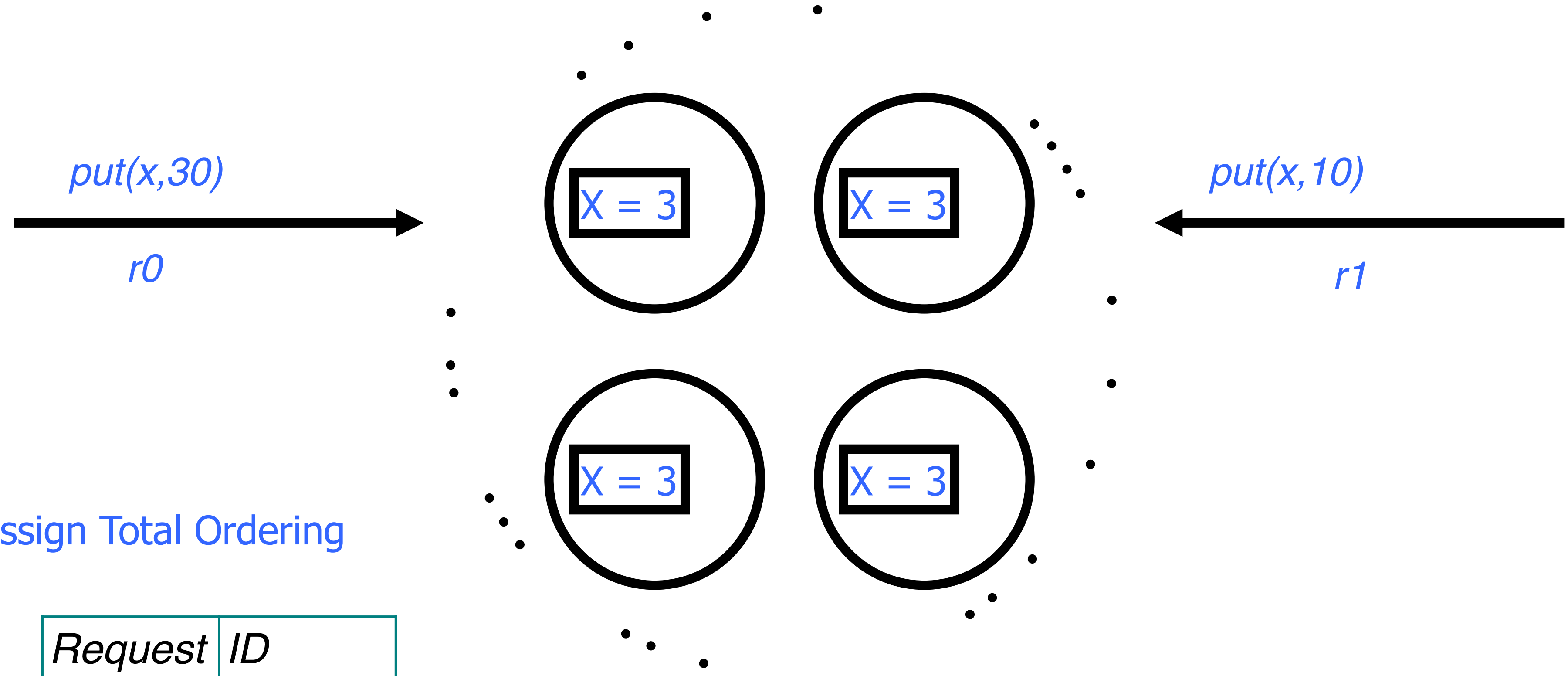
Assign unique ids to requests, process them in ascending order

- How do we assign unique ids in a distributed system?
- How do we know when every replica has processed a given request?

Order



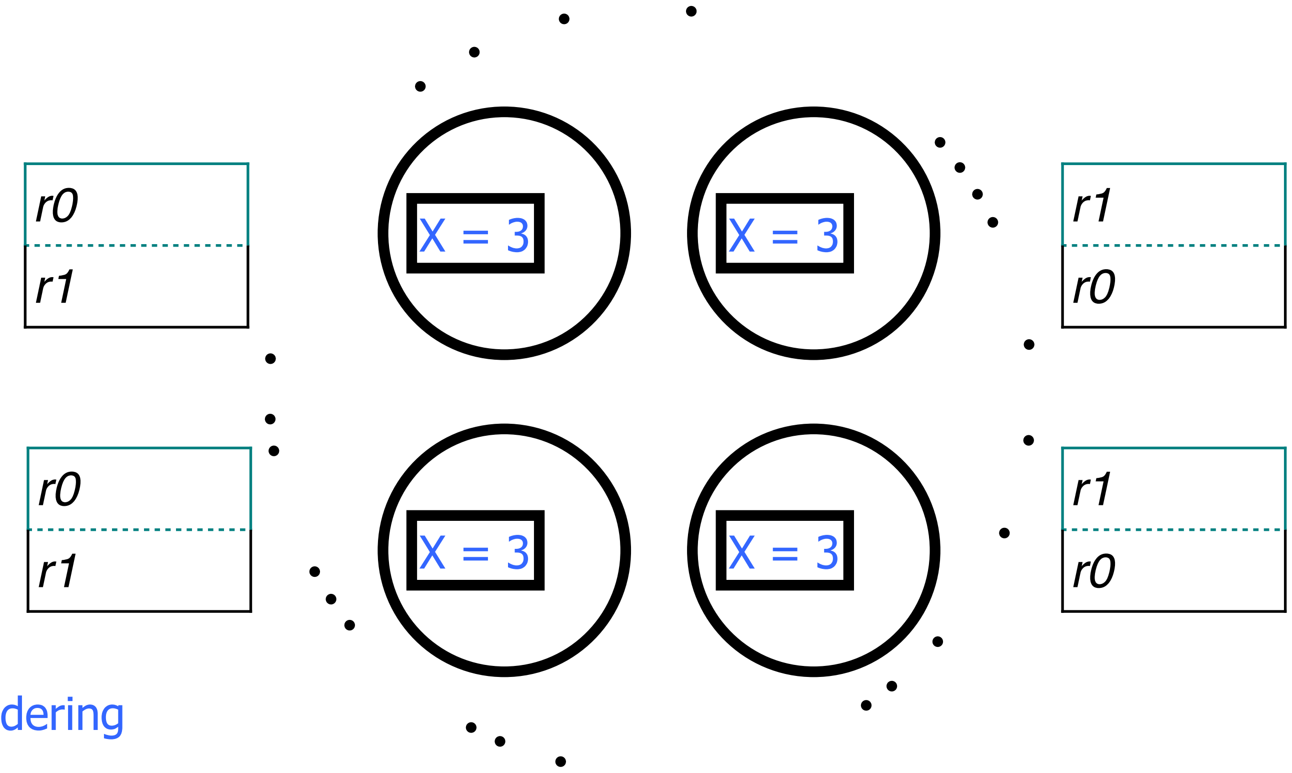
Order



Assign Total Ordering

Request	ID
r0	1
r1	2

Order



r0
r1

X = 3

X = 3

r1
r0

r0
r1

X = 3

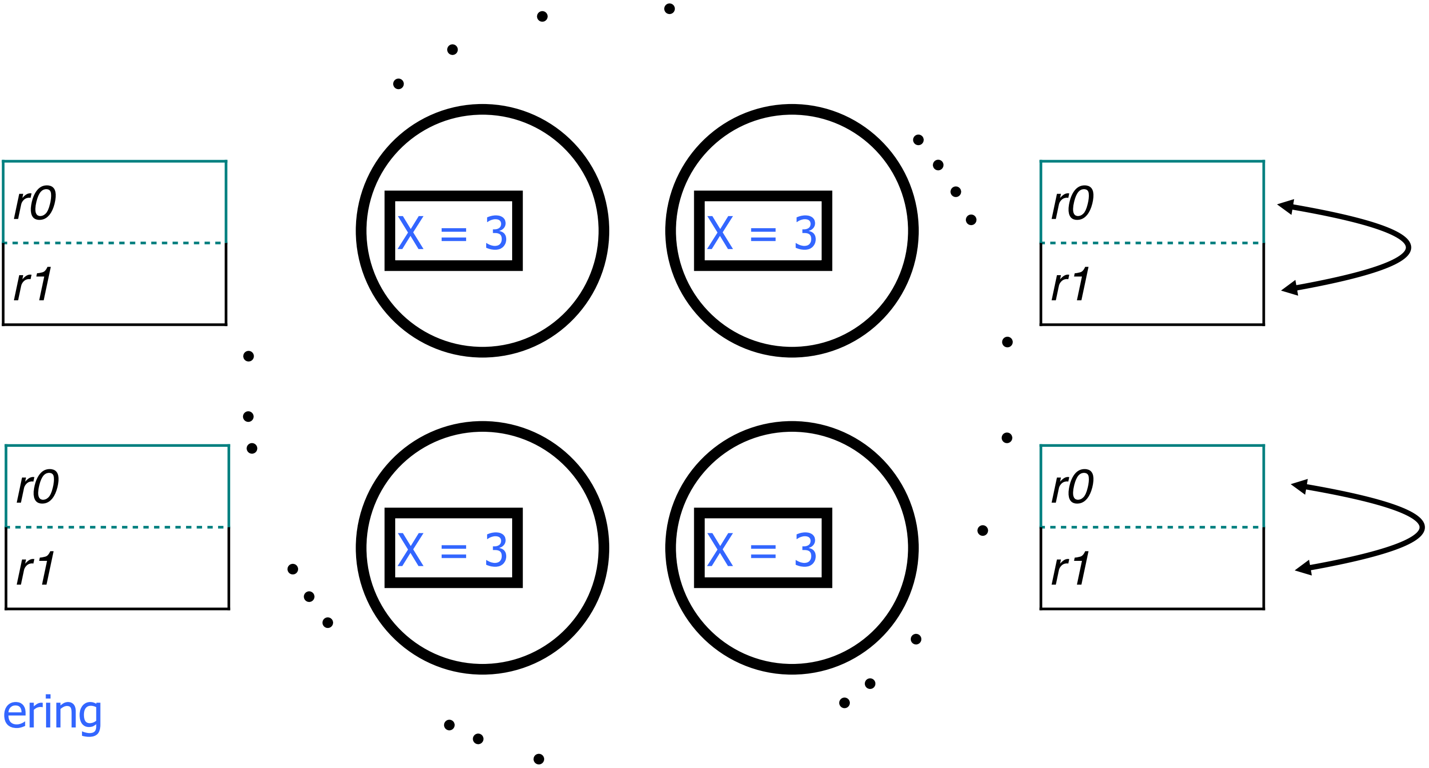
X = 3

r1
r0

Assign Total Ordering

Request	*ID*
r0	*1*
r1	*2*

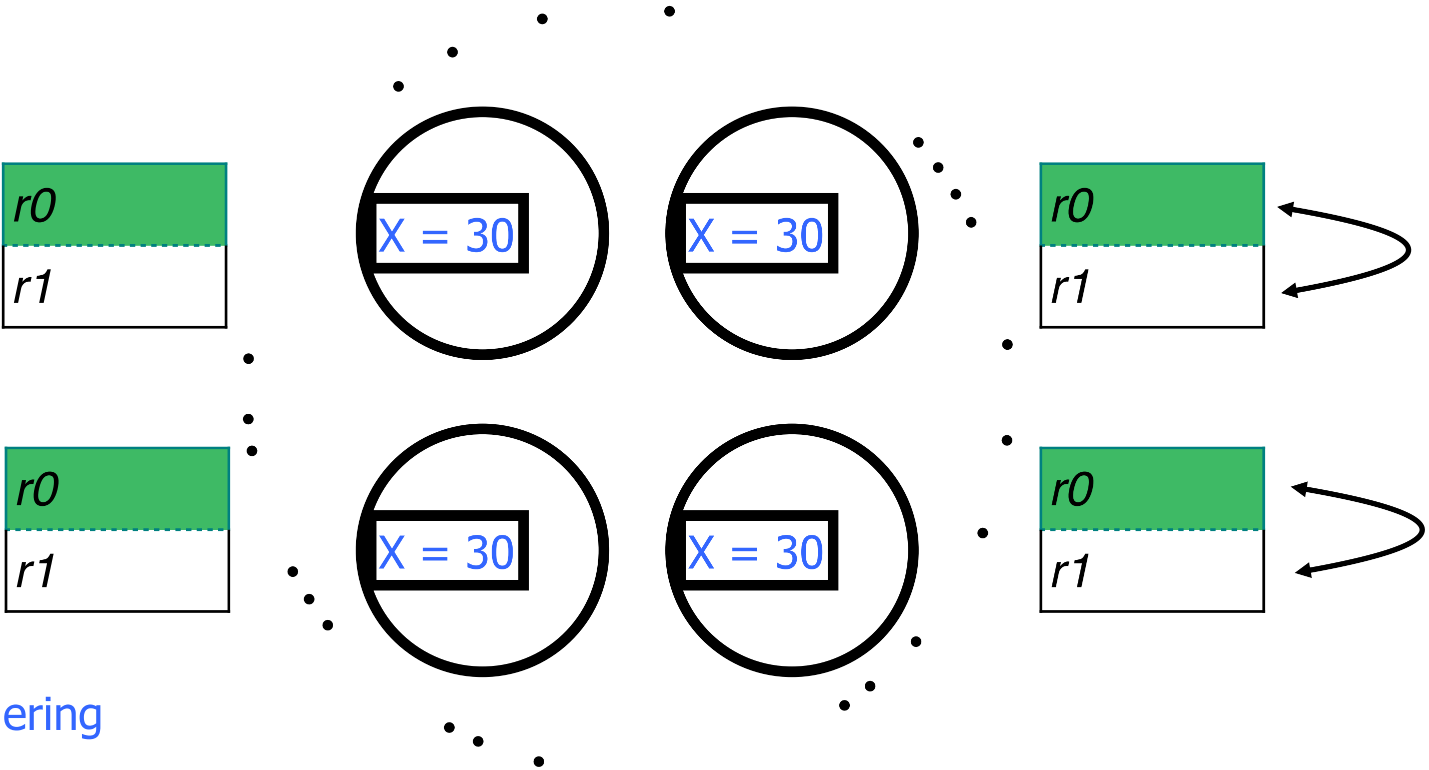
Order



Assign Total Ordering

Request	ID
r0	1
r1	2

Order



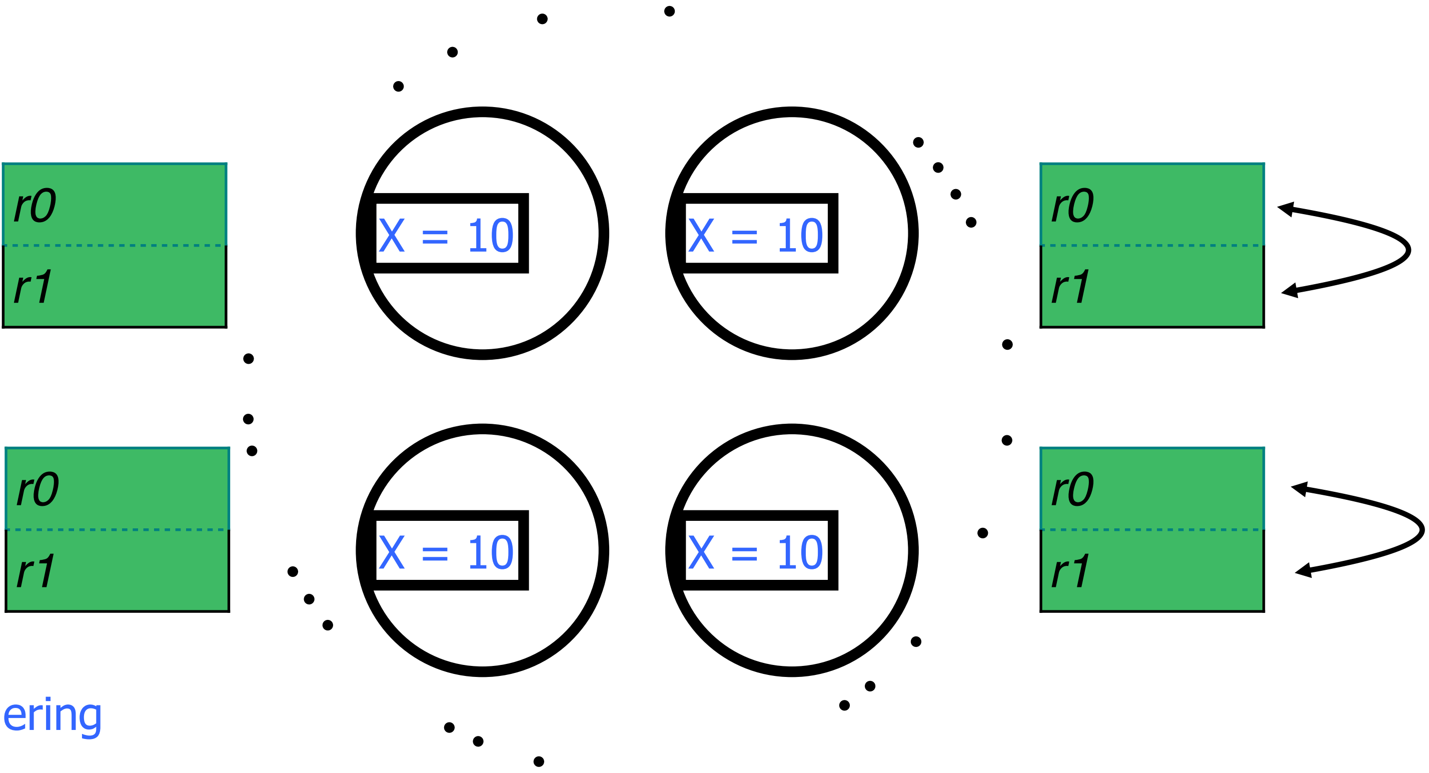
Assign Total Ordering

Request	ID
r0	1
r1	2

Cannot receive request with smaller ID

r0 is now stable!

Order



Assign Total Ordering

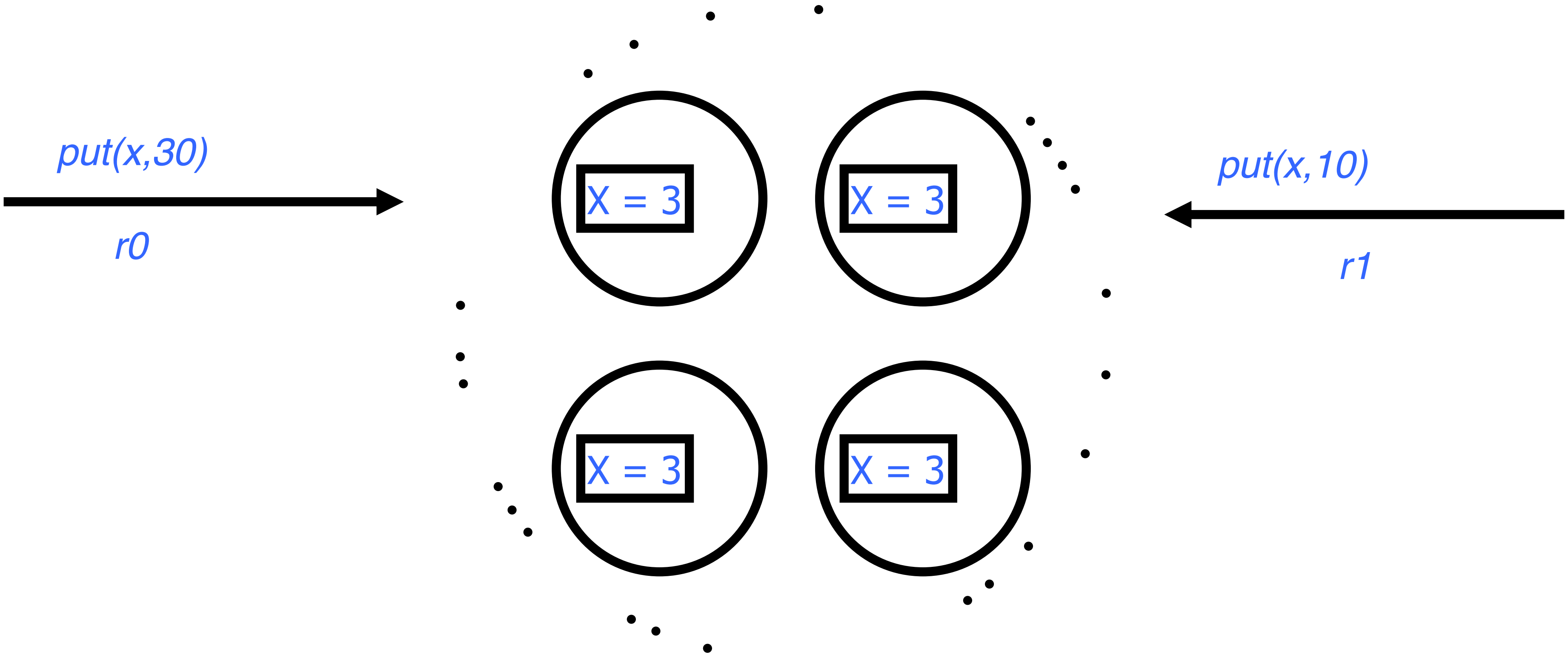
Request	ID
r0	1
r1	2

r0 is now stable!
r1 is now stable!

Generating IDs

- Order via clocks (client timestamp = id)
  - Logical clocks
  - Synchronized clocks
- Two-phase ID generation
  - Every replica proposes a candidate
  - One candidate is chosen and agreed upon by all replicas

Replica ID Generation



put(x,30)

r0

X = 3

X = 3

X = 3

X = 3

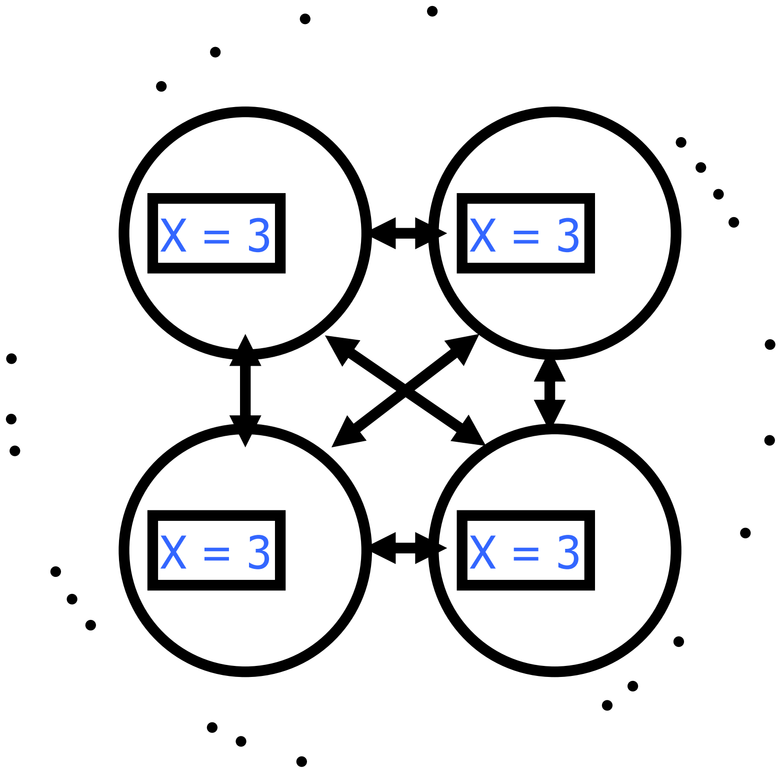
put(x,10)

r1

Replica ID Generation

Req.	*CUID*	*UID*
r0	*1.1*	
r1	*2.1*	

Req.	*CUID*	*UID*
r0	*1.2*	
r1	*2.2*	



Req.	*CUID*	*UID*
r1	*1.3*	
r0	*2.3*	

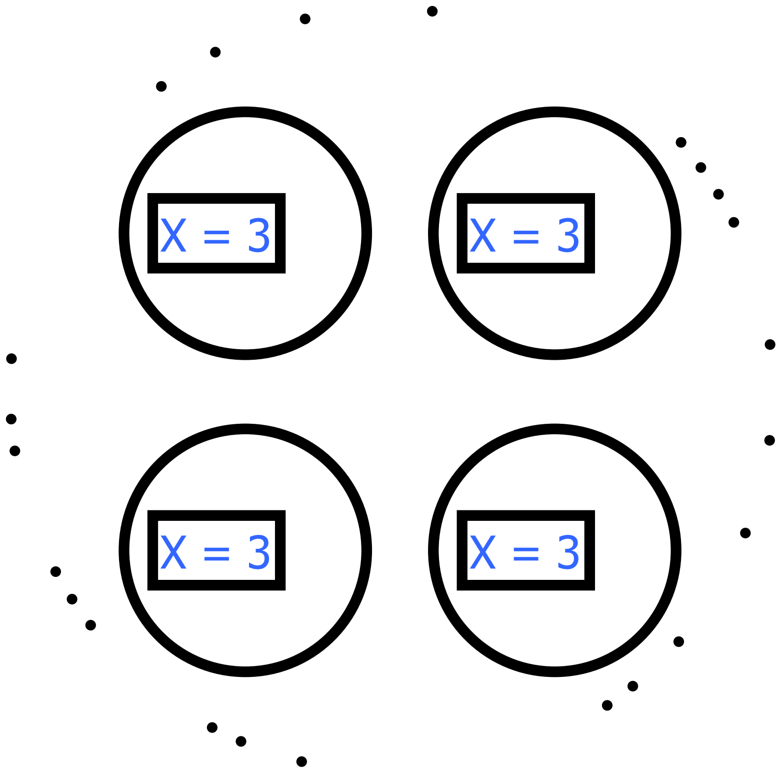
Req.	*CUID*	*UID*
r1	*1.4*	
r0	*2.4*	

1) Propose candidates

Replica ID Generation

Req.	CUID	UID
r0	*1.1*	*2.4*
r1	*2.1*	

Req.	CUID	UID
r1	*1.3*	
r0	*2.3*	*2.4*



Req.	CUID	UID
r0	*1.2*	*2.4*
r1	*2.2*	

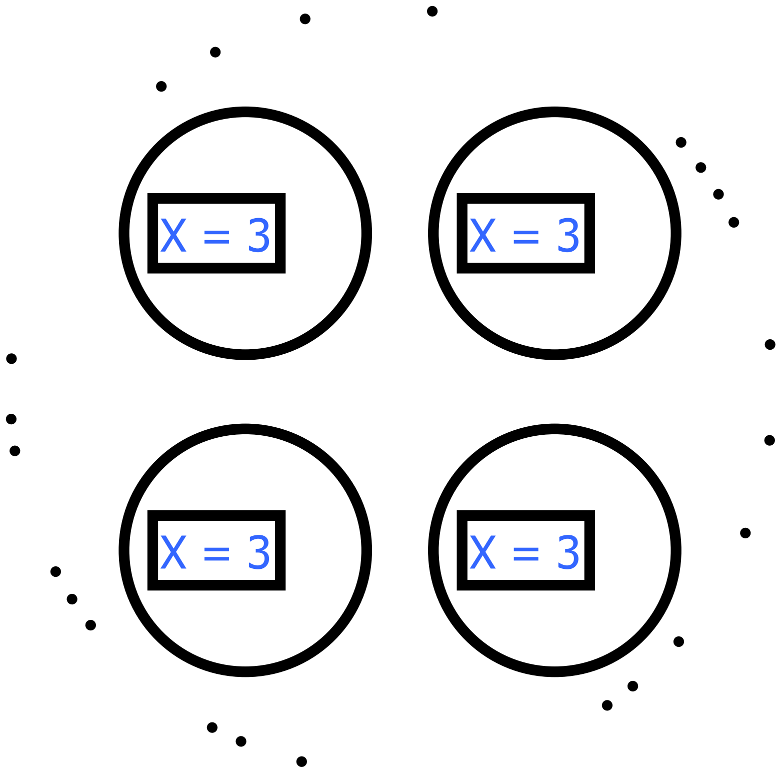
Req.	CUID	UID
r1	*1.4*	
r0	*2.4*	*2.4*

2) Accept *r0*

Replica ID Generation

Req.	*CUID*	*UID*
r0	*1.1*	*2.4*
r1	*2.1*	*2.2*

Req.	*CUID*	*UID*
r1	*1.3*	*2.2*
r0	*2.3*	*2.4*



Req.	*CUID*	*UID*
r0	*1.2*	*2.4*
r1	*2.2*	*2.2*

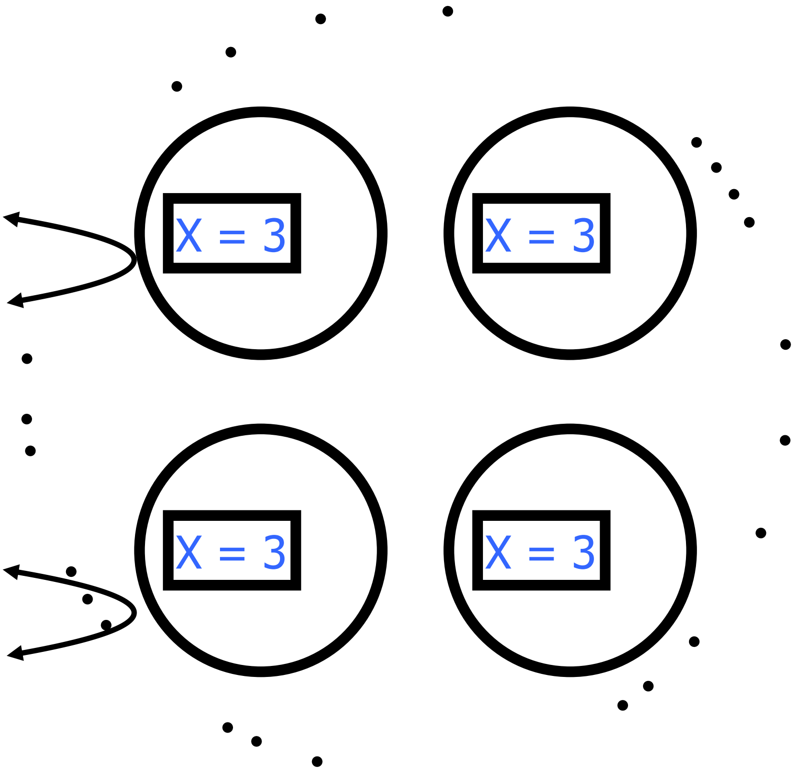
Req.	*CUID*	*UID*
r1	*1.4*	*2.2*
r0	*2.4*	*2.4*

3) Accept *r1*

Replica ID Generation

Req.	CUID	UID
r1	2.1	2.2
r0	1.1	2.4

Req.	CUID	UID
r1	1.3	2.2
r0	2.3	2.4



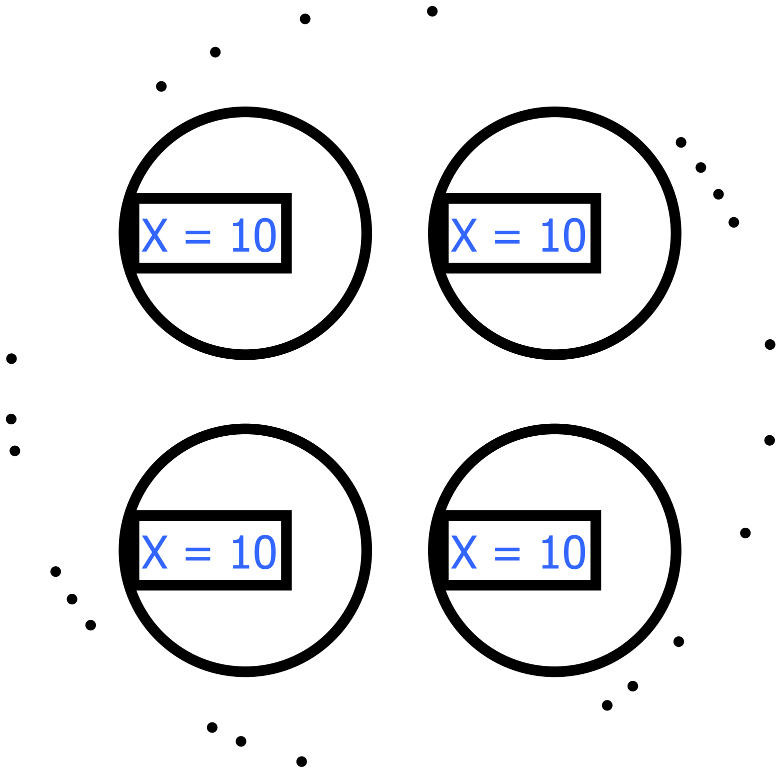
Req.	CUID	UID
r1	2.2	2.2
r0	1.2	2.4

Req.	CUID	UID
r1	1.4	2.2
r0	2.4	2.4

r1 is now stable

Replica ID Generation

Req.	*CUID*	*UID*
r1	*2.1*	*2.2*
r0	*1.1*	*2.4*



Req.	*CUID*	*UID*
r1	*1.3*	*2.2*
r0	*2.3*	*2.4*

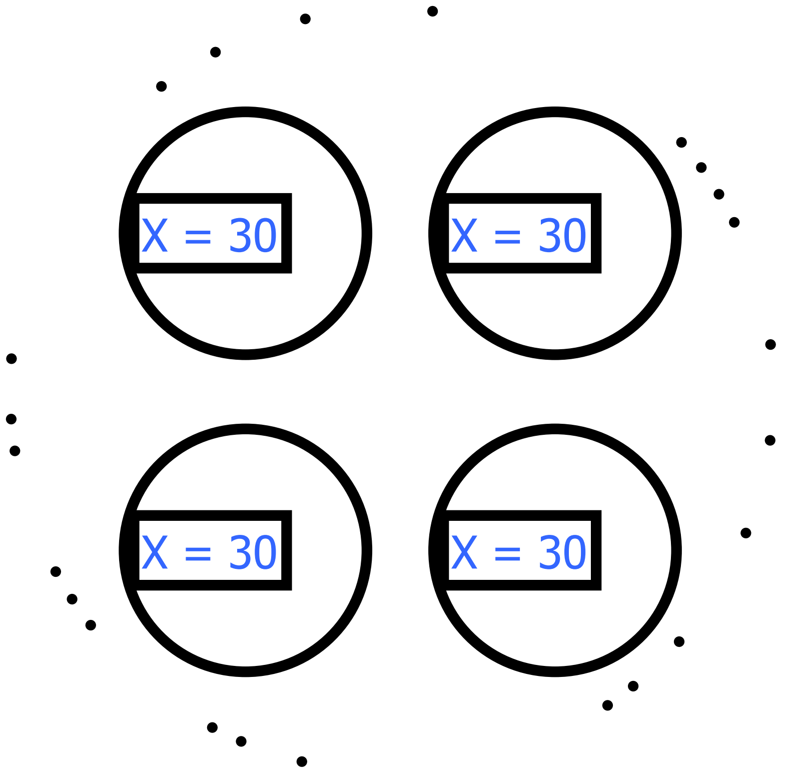
Req.	*CUID*	*UID*
r1	*2.2*	*2.2*
r0	*1.2*	*2.4*

Req.	*CUID*	*UID*
r1	*1.4*	*2.2*
r0	*2.4*	*2.4*

4) Apply *r1*

Replica ID Generation

Req.	*CUID*	*UID*
r1	*2.1*	*2.2*
r0	*1.1*	*2.4*



Req.	*CUID*	*UID*
r1	*1.3*	*2.2*
r0	*2.3*	*2.4*

Req.	*CUID*	*UID*
r1	*2.2*	*2.2*
r0	*1.2*	*2.4*

Req.	*CUID*	*UID*
r1	*1.4*	*2.2*
r0	*2.4*	*2.4*

5) Apply *r0*

Rules for Replica-Generated IDs

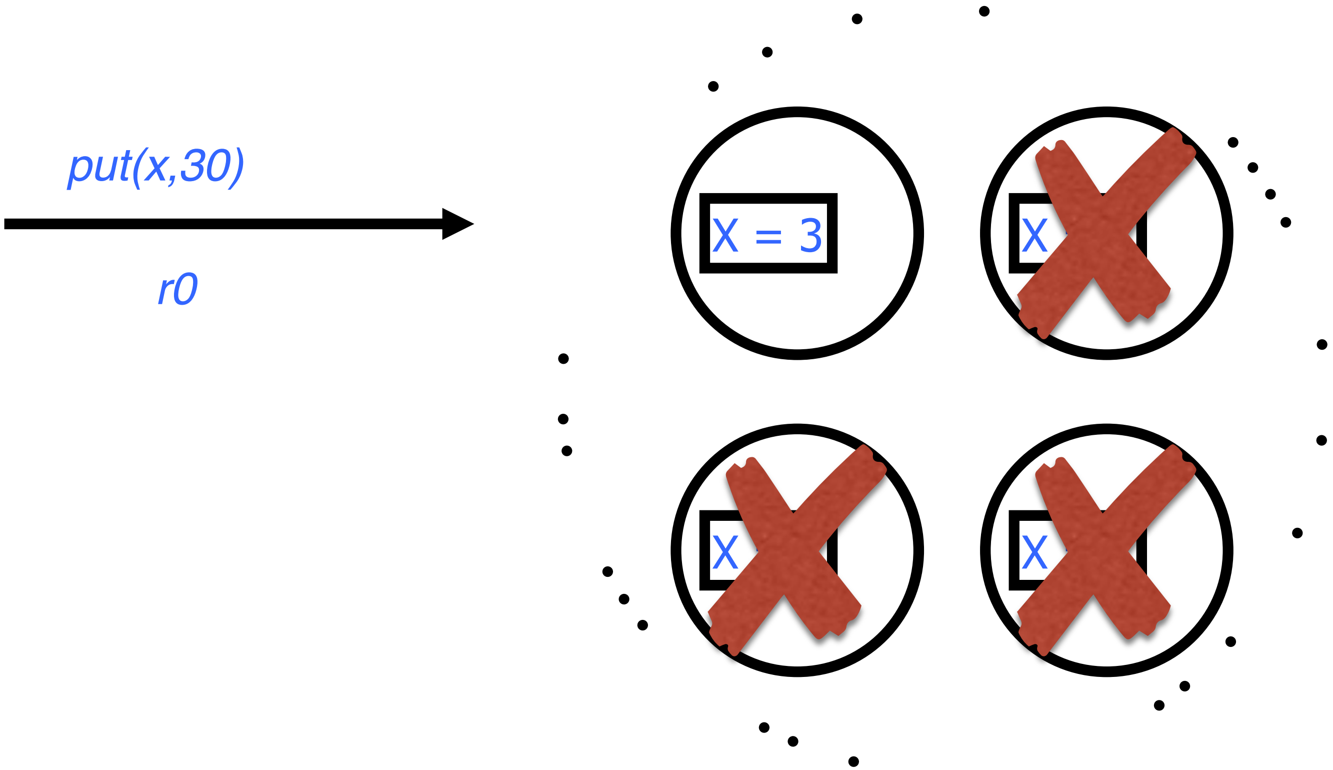
- Any new candidate ID must be > ID of any accepted request
- The ID selected from the candidate list must be >= each candidate

- When is a candidate stable?
  - It has been accepted
  - No other pending request with a smaller candidate ID

Fail-Stop Faults

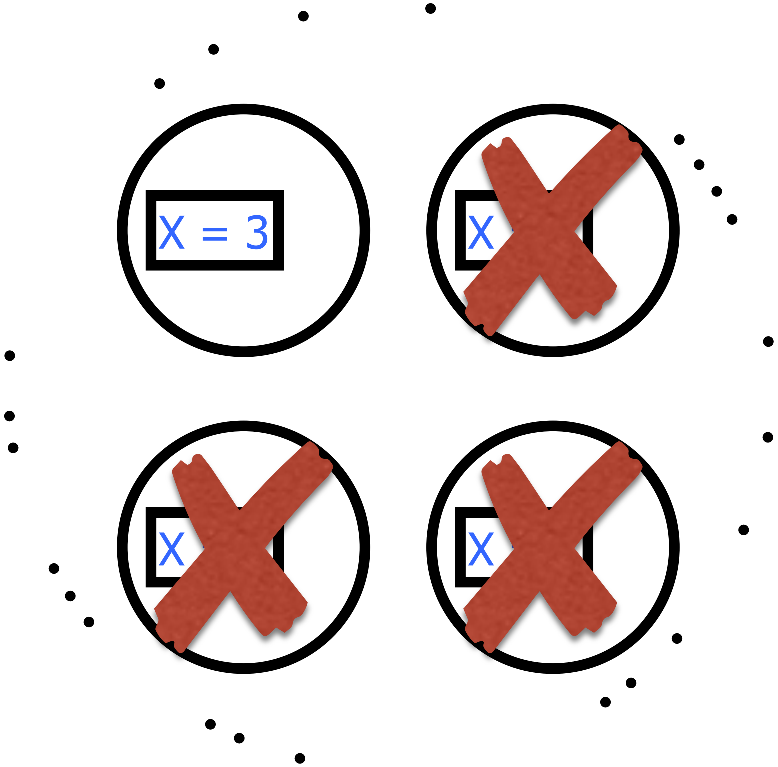
- A faulty server can be detected as faulty
  - Weakest? failure model

Fail-Stop Tolerance



Fail-Stop Tolerance

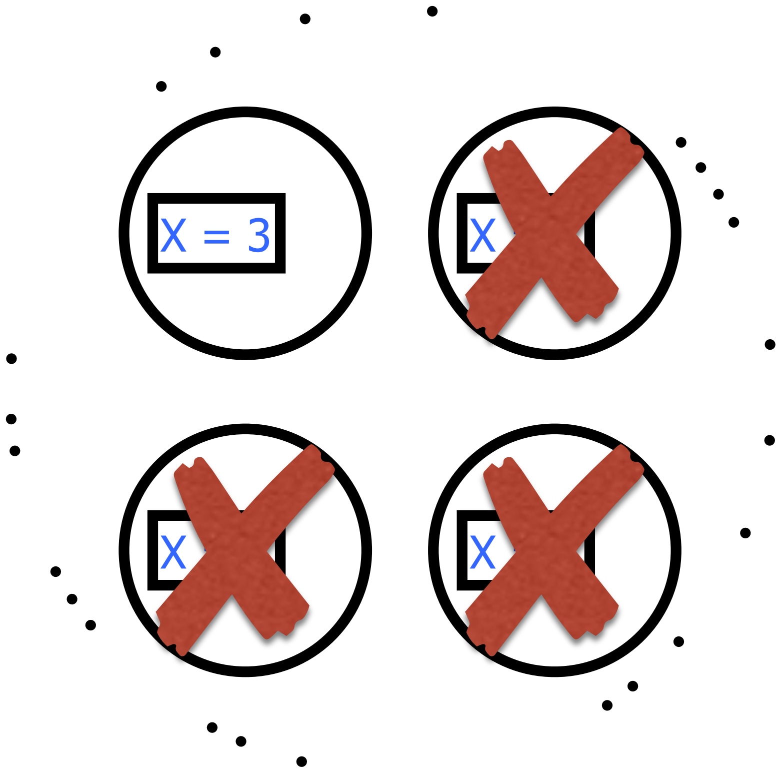
Req.	*CUID*	*UID*
r0	*1.1*	



1) Propose Candidates….

Fail-Stop Tolerance

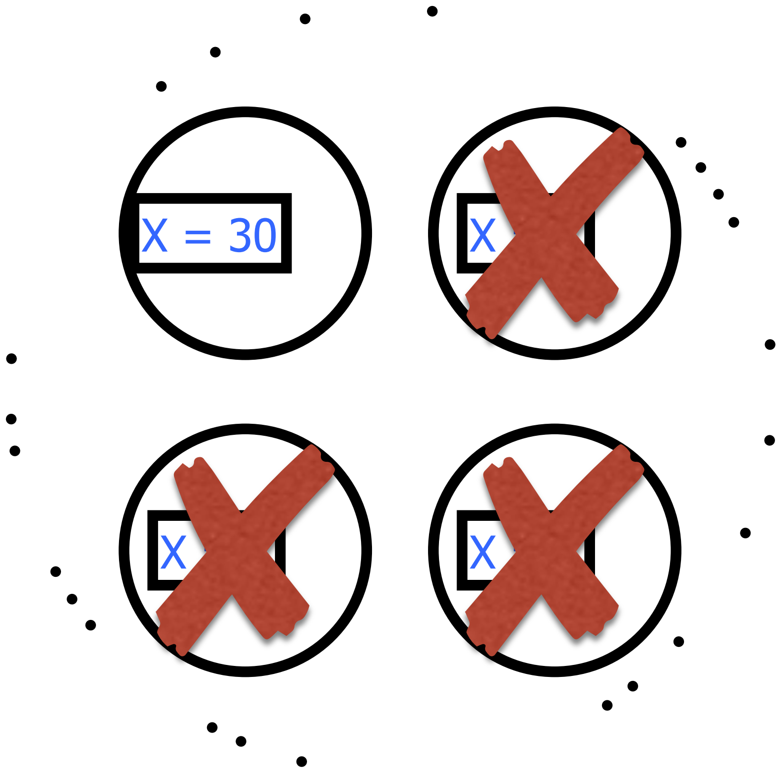
Req.	*CUID*	*UID*
r0	*1.1*	*1.1*



2) Accept *r0*

Fail-Stop Tolerance

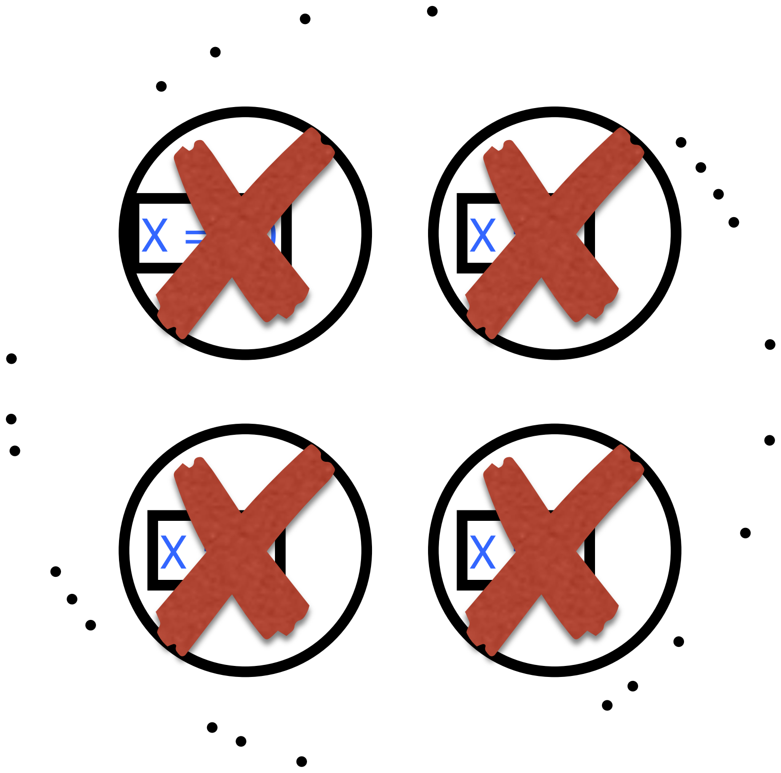
Req.	*CUID*	*UID*
r0	*1.1*	*1.1*



2) Apply *r0*

Fail-Stop Tolerance

GAME OVER!!!



2) Apply *r0*

Fail-Stop Fault Tolerance

- To tolerate t failures, need t+1 servers
  - As long as 1 non-faulty server remains, we’re OK
- Only need to participate in protocols with other live servers