A Quorum-Based Replication Framework for Distributed Software Transactional Memory

Bo Zhang and Binoy Ravindran

ECE Department

Virginia Tech

Blacksburg, VA, USA

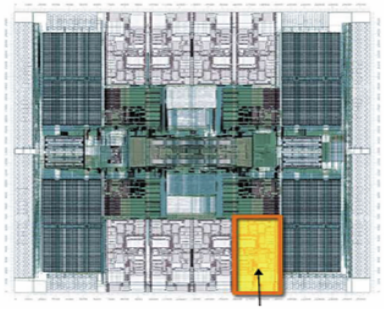
OPODIS

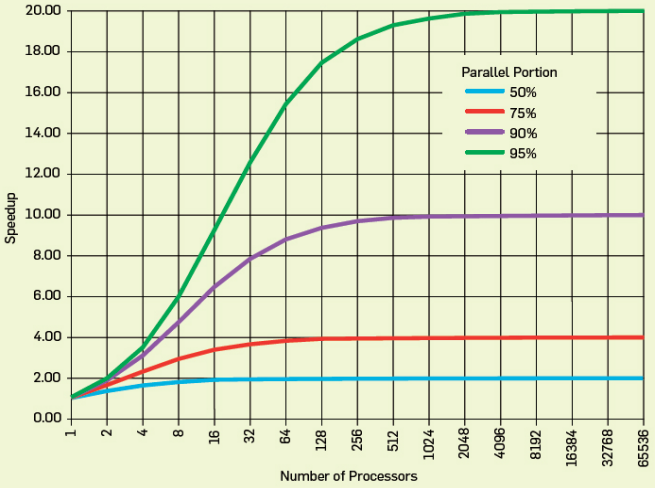
December 13, 2011

CONCURRENCY CONTROL ON CHIP MULTIPROCESSORS SIGNIFICANT AFFECTS PERFORMANCE (AND PROGRAMMABILITY)

- **Chip multiprocessors (CMPs/Multicore) are here**
  - Improve performance by exposing greater concurrency in software
- **Difficult: last *x*% involve significant coordination and synchronization**
  - *Amdahl's law: relationship between sequential execution time and speedup reduction is not linear*

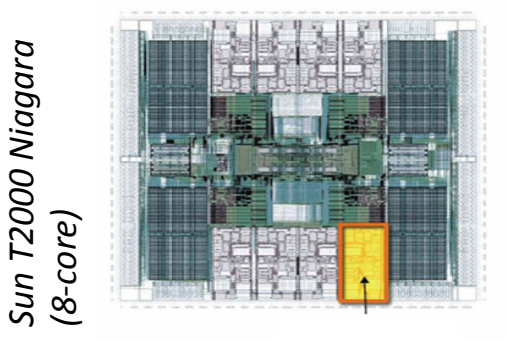
Sun T2000 Niagara (8-core)

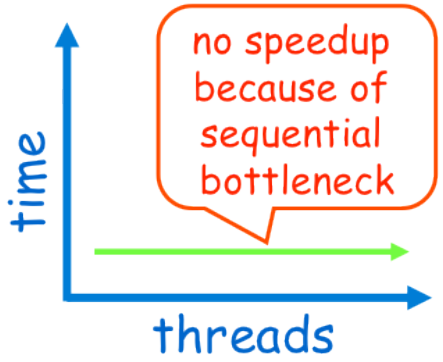




LOCK-BASED CONCURRENCY CONTROL IS CHALLENGING

- **Coarse-grained locking is simple, but permits little concurrency**





- **Fine-grained locking allows greater concurrency, but error-prone**
  - Must acquire only necessary and sufficient locks
  - Must avoid deadlocks
  - Livelocks, lock convoying, priority inversion,….

- **Challenges exacerbate in distributed systems**

TRANSACTIONAL MEMORY (TM): AN ALTERNATIVE TO LOCKS

- **Organize code that access shared memory as transactions that (appear to) execute *atomically* and in *isolation***

```
atomic{
   x = x + y;
}
```

- **Transactions optimistically execute, logging all changes**
- **Two transactions conflict if they access same object and one access is a write**
  - A conflict resolution policy is used: one is allowed to commit; other is aborted, changes rolled-back, retried
- **Gaining traction; implementations in software and hardware; but no silver bullet**

M. Herlihy and J. B. Moss (1993). Transactional memory: Architectural support for lock-free data structures. *ISCA*. pp. 289–300.
N. Shavit and D. Touitou (1995). Software Transactional Memory. *PODC*. pp. 204—213.

DISTRIBUTED SOFTWARE TRANSACTIONAL MEMORY (D-STM)

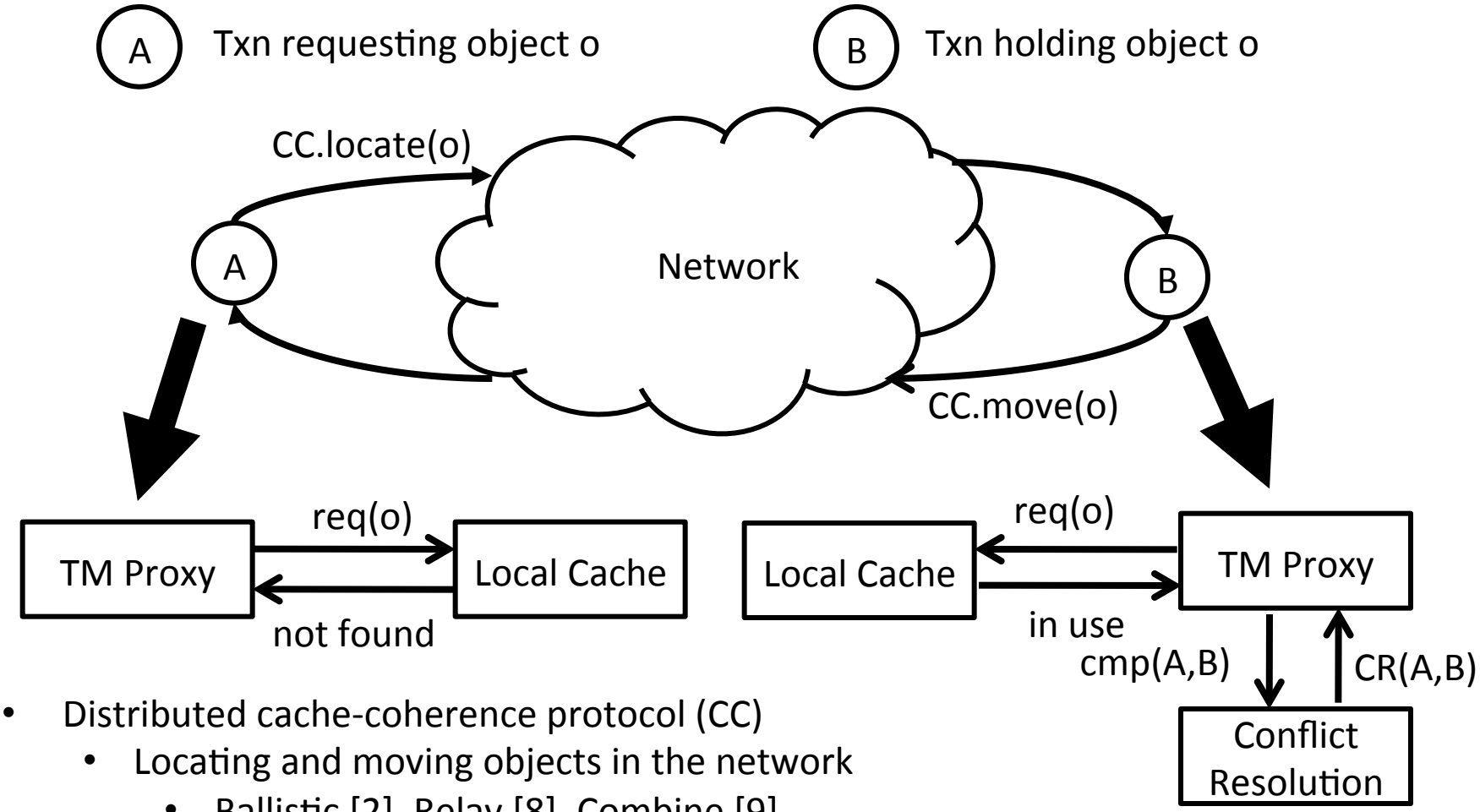
- **D-STM**
  - Support STM in distributed systems
  - Nodes communicate by message passing links

- **Control flow**
  - Transactions move, objects held locally
  - Inherited from database transactional synchronization
  - Consistency using distributed commit (e.g., two-phase commit)

- **Data flow**
  - Objects move, transactions run locally
  - Cache coherence, conflict resolution
  - No distributed commit
  - Easier to exploit locality

(paper's focus)

SINGLE COPY (SC) D-STM MODEL



- Distributed cache-coherence protocol (CC)
  - Locating and moving objects in the network
    - Ballistic [2], Relay [8], Combine [9]
  - Only one (writable) copy exists for each object
- Conflict resolution module (CR)
  - Resolve conflicts among transactions
  - Ensure progress and enhance concurrency

LIMITATIONS OF SC D-STM MODEL

- **No fault-tolerance property**
  - When node fails, held objects are lost
- **Limited support of concurrent reads**
  - CC protocol needs to maintain the consistency over read-only replicas while some transaction is writing the object
  - High communication cost to detect read/write conflict
  - Typical directory-based CC protocols often do not differentiate between read and write operations
- **Limited locality**
  - One major goal of directory-based CC protocols is to exploit locality
  - Directory-based CC protocols often keep track of the single writable copy
  - In practice, not all transactional requests are routed efficiently
  - Possible locality is often overlooked

QUORUM-BASED REPLICATION (QR) D-STM MODEL

- **Fault-tolerance guarantee**
  - Tolerate certain level of node failures

- **Inherent support of concurrent reads**
  - Multiple (writable) replicas of each object
  - Concurrent read transactions never serialized
  - (Conflicts resolved at commit)

- **Best-effort locality exploration under node failures**
  - Transactions need to find latest copy of requested object
  - Send request to a certain set of nodes
  - In case of node failures, do best effort to find the closest possible set of nodes which hold latest copy

SYSTEM MODEL

- **Failure-prone distributed system**
  - Nodes communicate by message passing links
  - Metric-space network of diameter *D*
  - Fail-stop node failures

- **Distributed transactions**
  - A set of transactions invoked by different nodes
  - Share a set of objects
  - R->W, W->R, W->W conflicts

- **A fixed contention manager**
  - Located at every node
  - Resolve conflicts based on a consistent policy

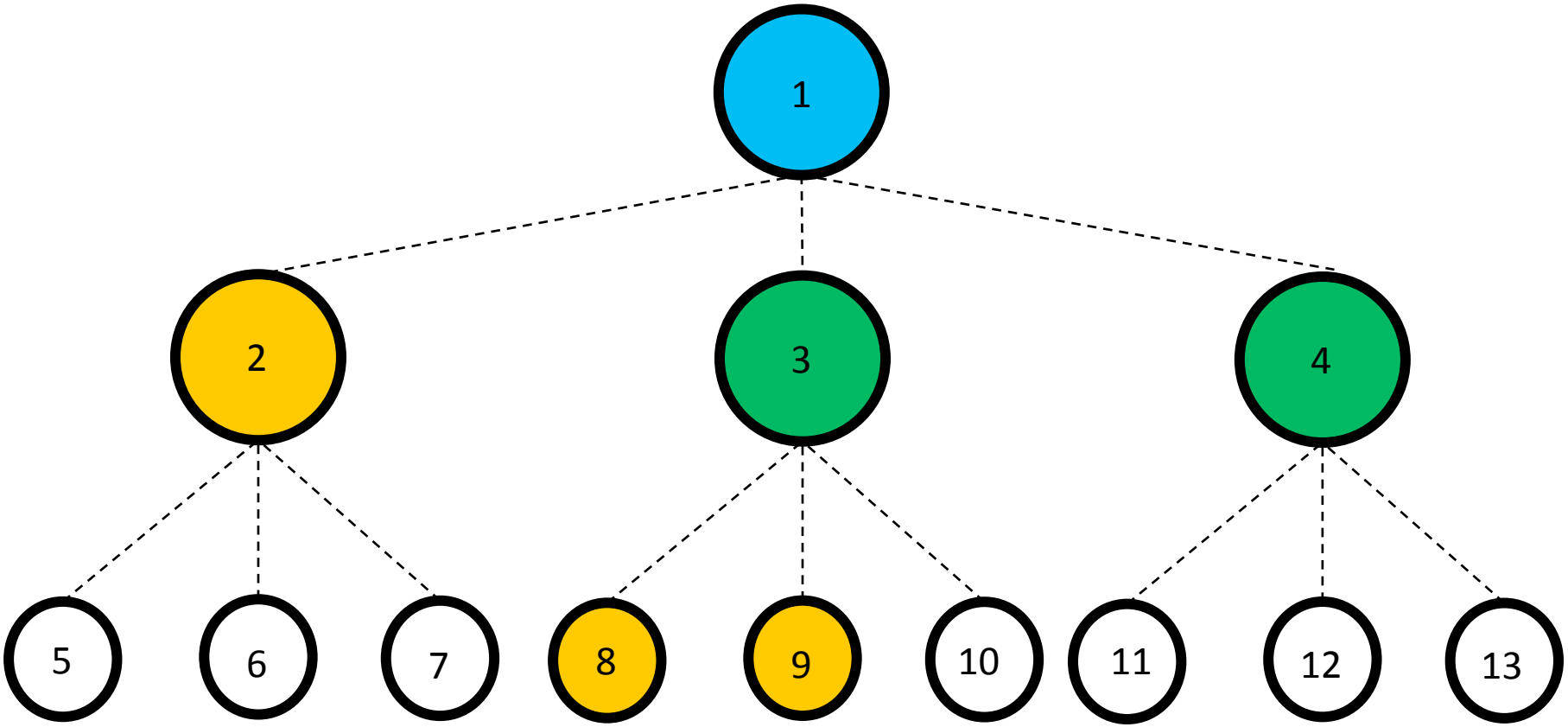
QR MODEL: PRELIMINARIES

- **Objects are replicated based on quorums**
  - A quorum system is constructed

- **Quorum: a collection of nodes**
  - Read quorum and write quorum
  - Two quorums *intersect* if one of them is a write quorum

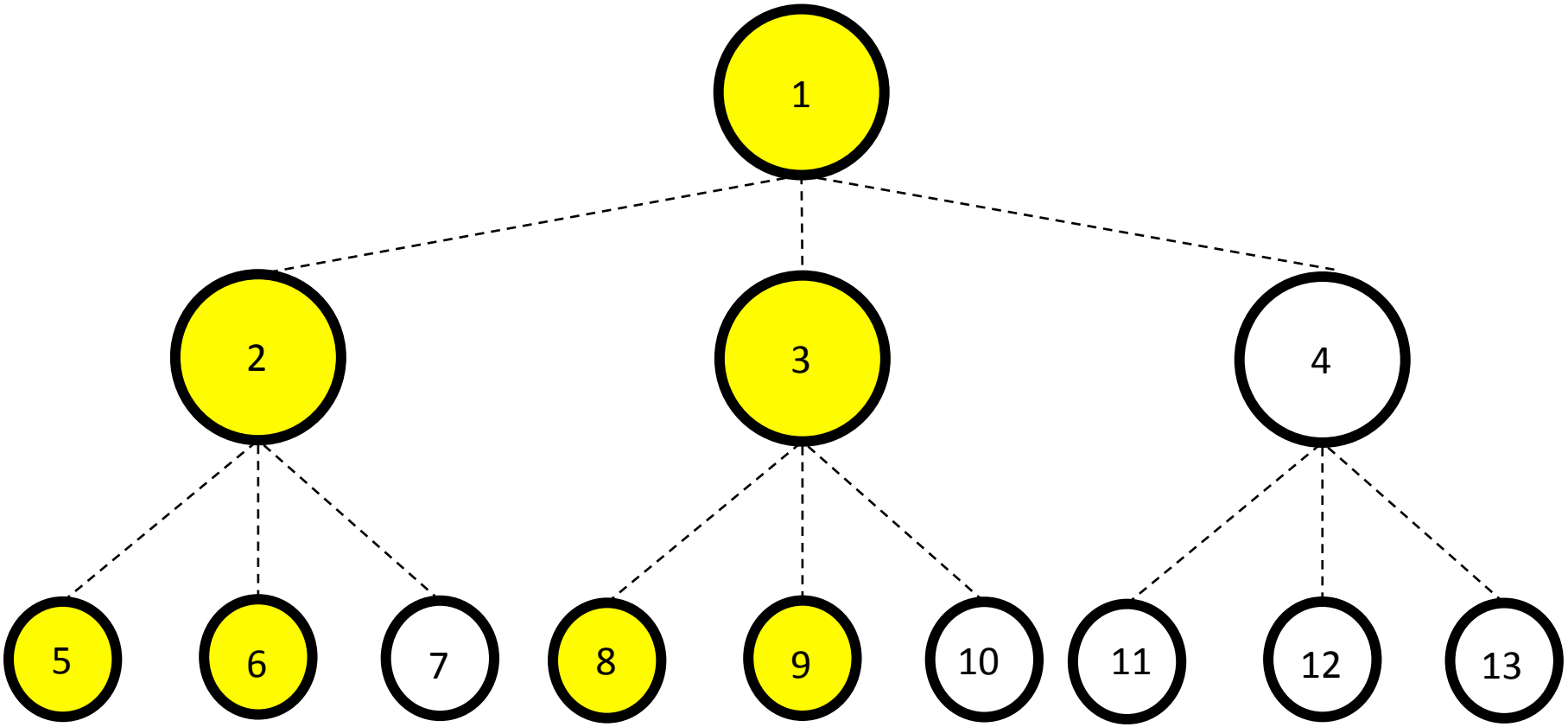
- **Five operations provided**
  - Read and write: regular operations
  - Request-commit: validate transaction after regular operations
  - Commit and abort: complete a transaction by the result of request-commit operation

EXAMPLE: TREE QUORUM SYSTEM



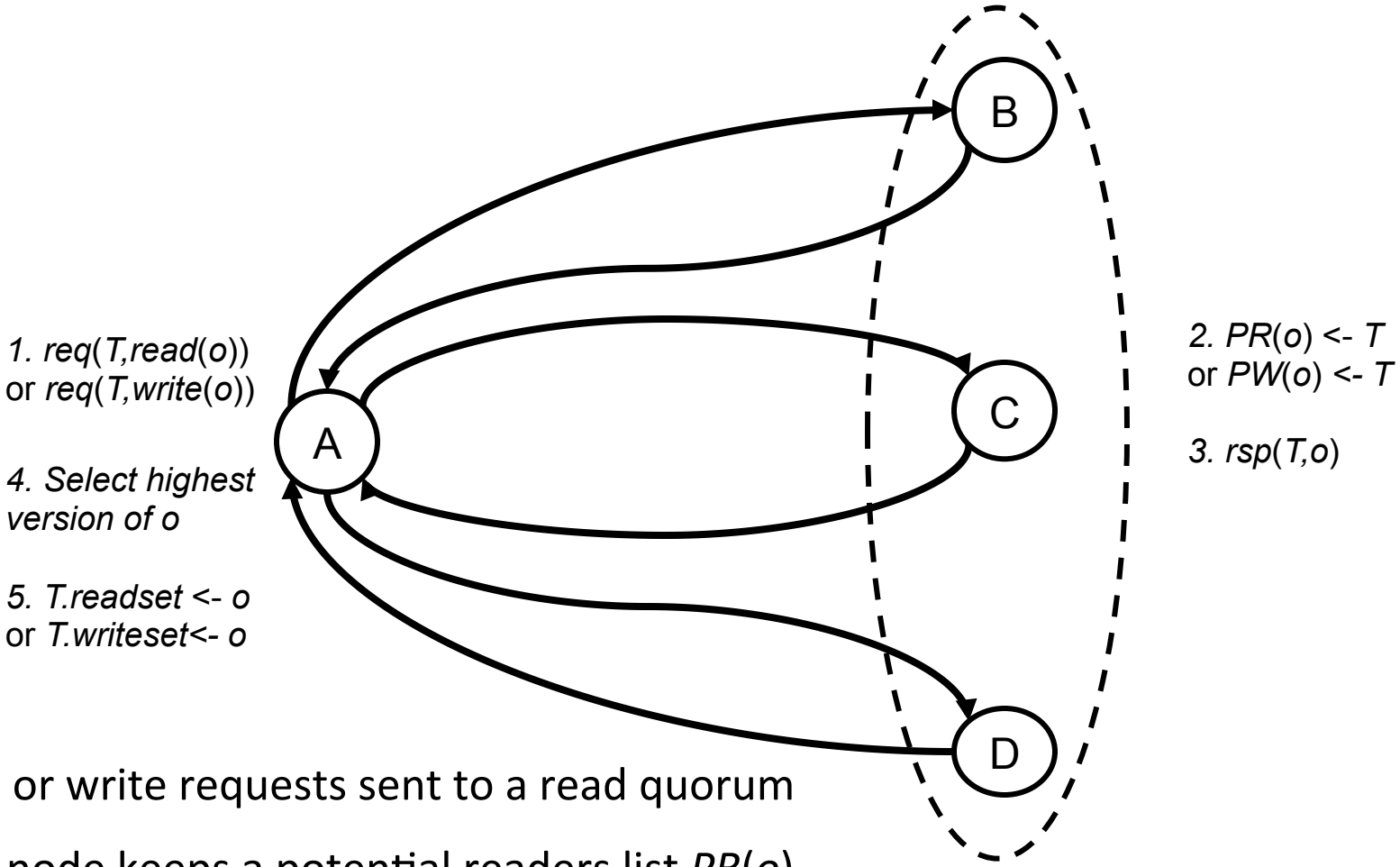
- Tree quorum system: Agrawal and Abbadi ‘90 [11]
  - Read quorum: the root, **or** replaced by its majority of children recursively
  - Three read quorums are shaded

EXAMPLE: TREE QUORUM SYSTEM



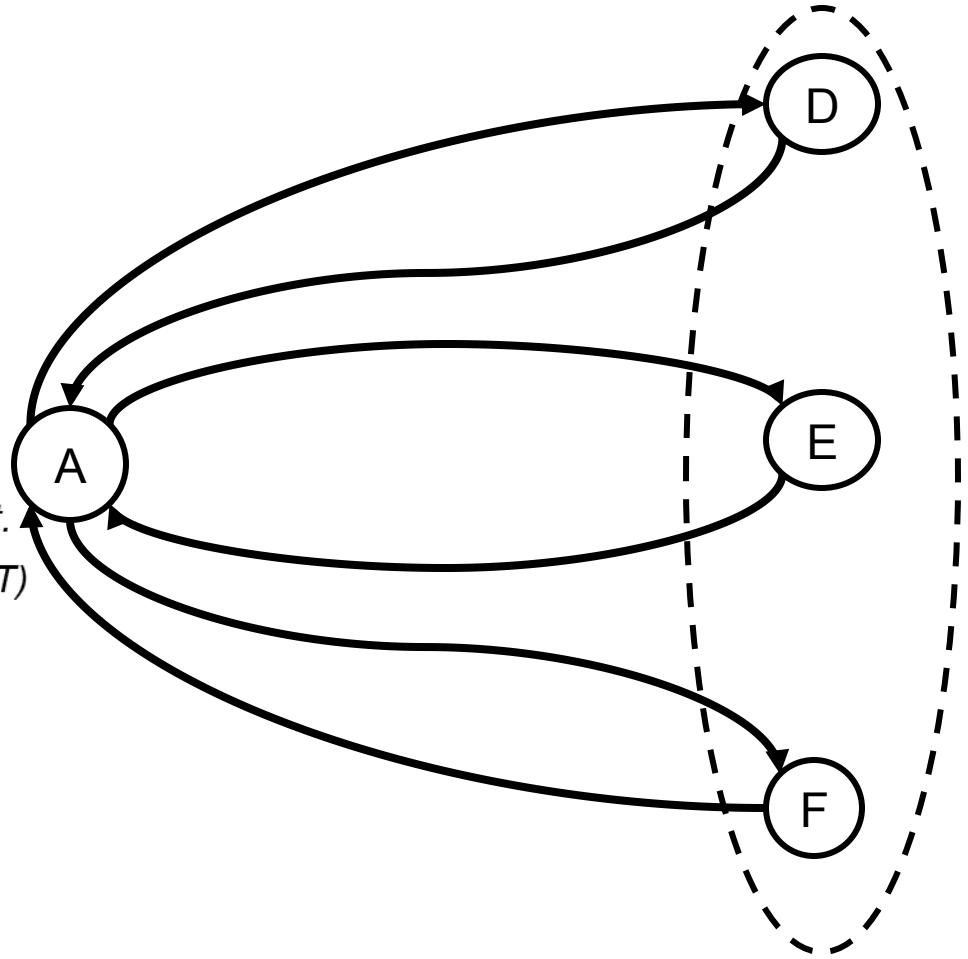
- Tree quorum system: Agrawal and Abbadi ‘90 [11]
  - Write quorum: the root **and** its majority of children recursively selected
  - One write quorum is shaded

QR MODEL: READ AND WRITE OPERATIONS



- Read or write requests sent to a read quorum
- Each node keeps a potential readers list *PR*(*o*) and a potential writers list *PW*(*o*) for each object *o*
- The highest version copy is selected when multiple copies received
- Each transaction keeps a readset and a writeset

QR MODEL: REQUEST-COMMIT OPERATION



1. req_cmt(T)

6. Abort if any rsp(T,abort) received

7. Else, commit.

AT(T) <- ∪CT(T)

2. Conflict-Detect(A,T) and update CT(T)

3. CM(T,CT(T))

4. If T aborts, rsp(T,abort); If not, rsp(T,cmt,CT(T)) and o.protected <- true for all o requested by T

5. Remove T from PR(o) and PW(o) for all o

QR MODEL: REQUEST-COMMIT OPERATION

- Request-commit after all regular operations
  - Request sent to a write quorum

- Remote: conflict detection and contention management
  - Conflicting transaction list: *CT*(*T*)
  - Conflicts are detected for each object requested by *T*, based on object's version number and potential reader and writer lists
  - *CM*(*T, CT*(*T*)): contention management for *T* and every transaction in *CT*(*T*)
  - *o.protected:* field to protect *o* from being overwritten after request-commit

- Local: determine commit or abort based on responses
  - If commit, update an abort transactions list *AT*(*T*) by including all received *CT*(*T*) from remote nodes

QR MODEL: COMMIT AND ABORT OPERATIONS

- Request sent to a write quorum

- Commit operation
  - Invoked immediately after request-commit operation
  - Local & remote: overwrite the object value, increase version number by 1, and set *o.protected* to *false* for every *o* requested by *T*
  - Local: send abort message to every transaction in *AT*(*T*)

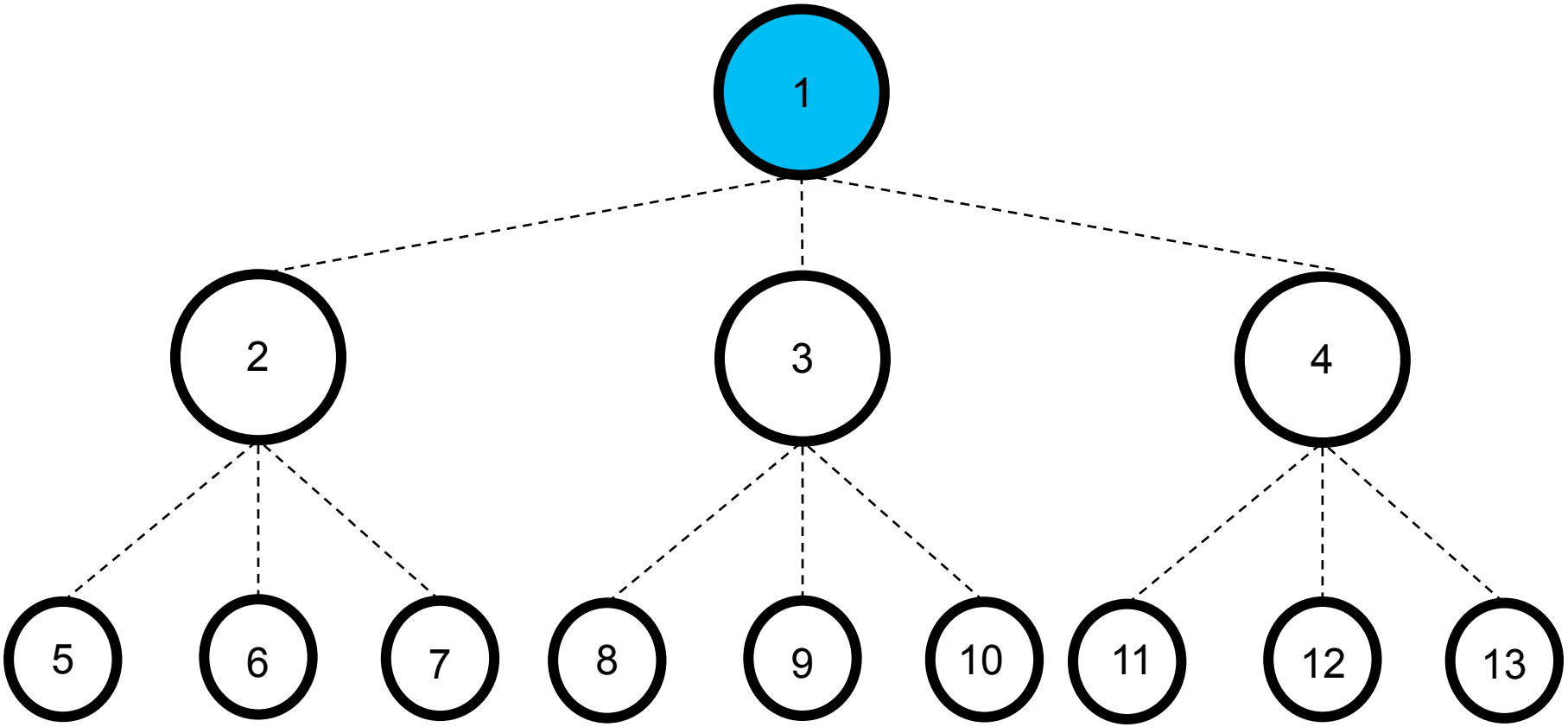
- Abort operation
  - Invoked immediately after any abort message received
  - Local: discard any changes made to objects
  - Remote: set *o.protected* to *false* for every *o* requested by *T* and remove *T* from all potential readers and writers list

QUORUM CONSTRUCTION

- A quorum construction is needed
  - Correctly define read and write quorums in the system
  - Determine which quorum should be selected for any operation invoked by any node
  - Find replacement of failed nodes

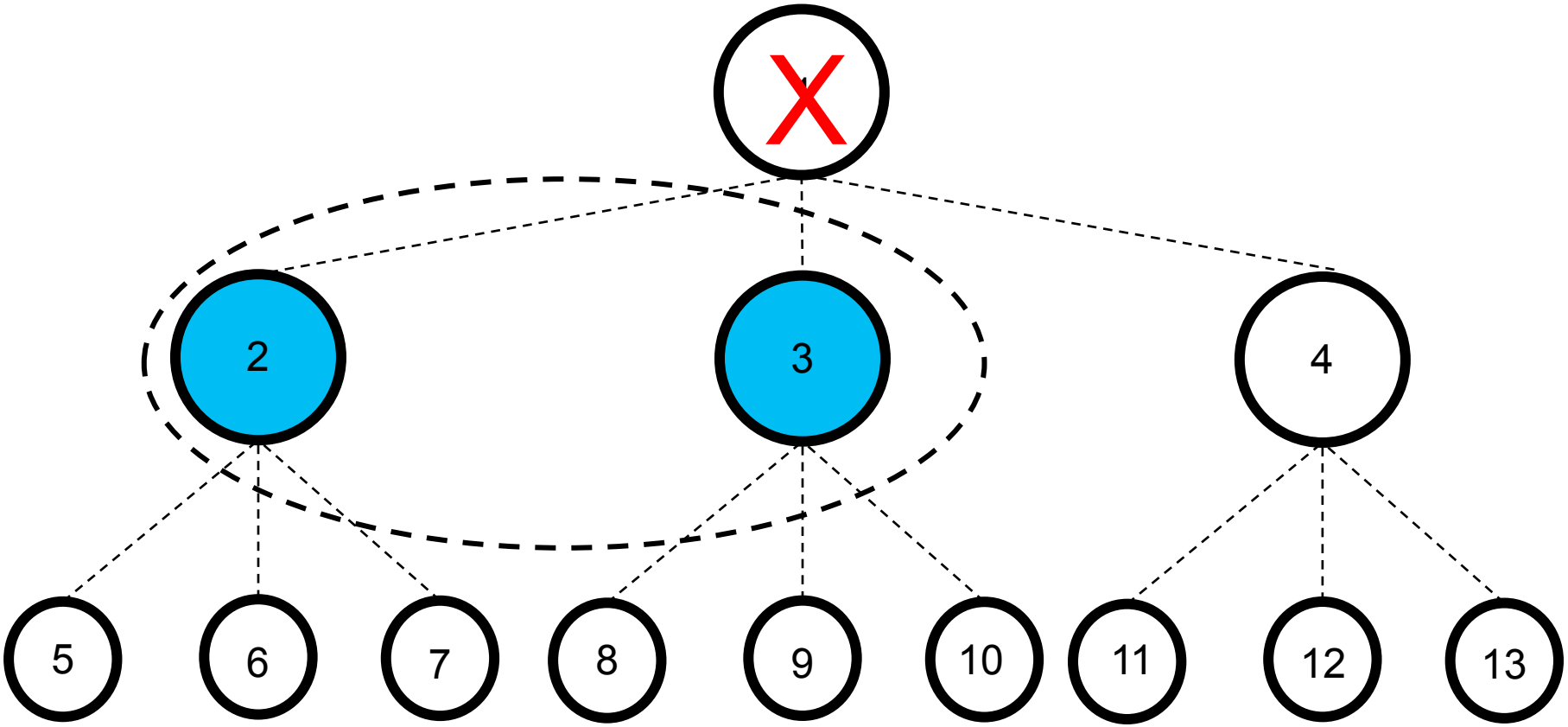
- FLOODING protocol
  - Motivated by tree quorum system by Agrawal and Abbadi [11]
  - An overlay tree constructed on Herlihy and Sun [2]'s hierarchical clustering structure
  - For each node, a basic read quorum and a basic write quorum is always selected when no node fails
  - When some node fails, a replacement is dynamically found

FLOODING PROTOCOL: EXAMPLE OF READ QUORUM



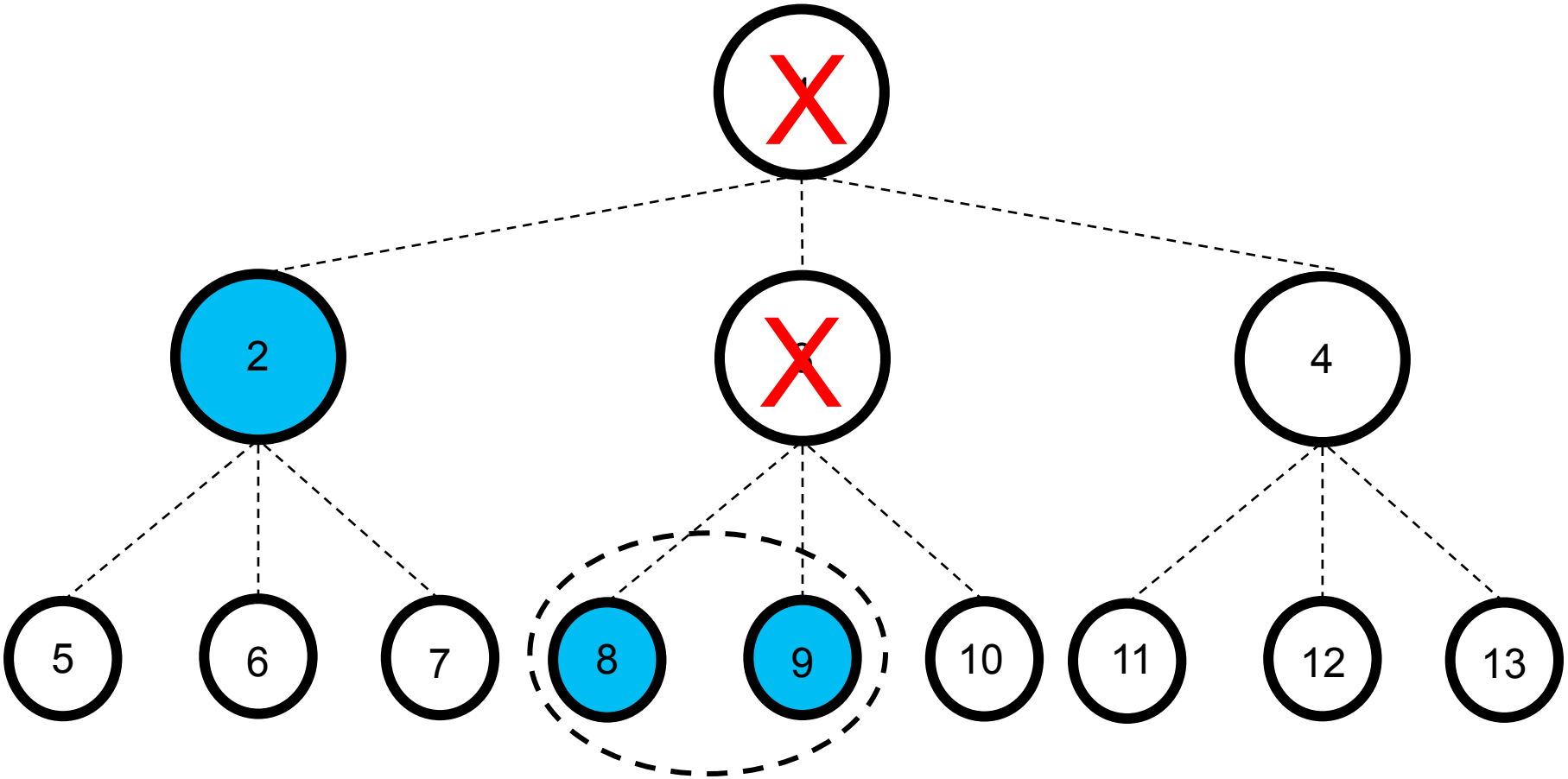
- Basic read quorum: root node

FLOODING PROTOCOL: EXAMPLE OF READ QUORUM



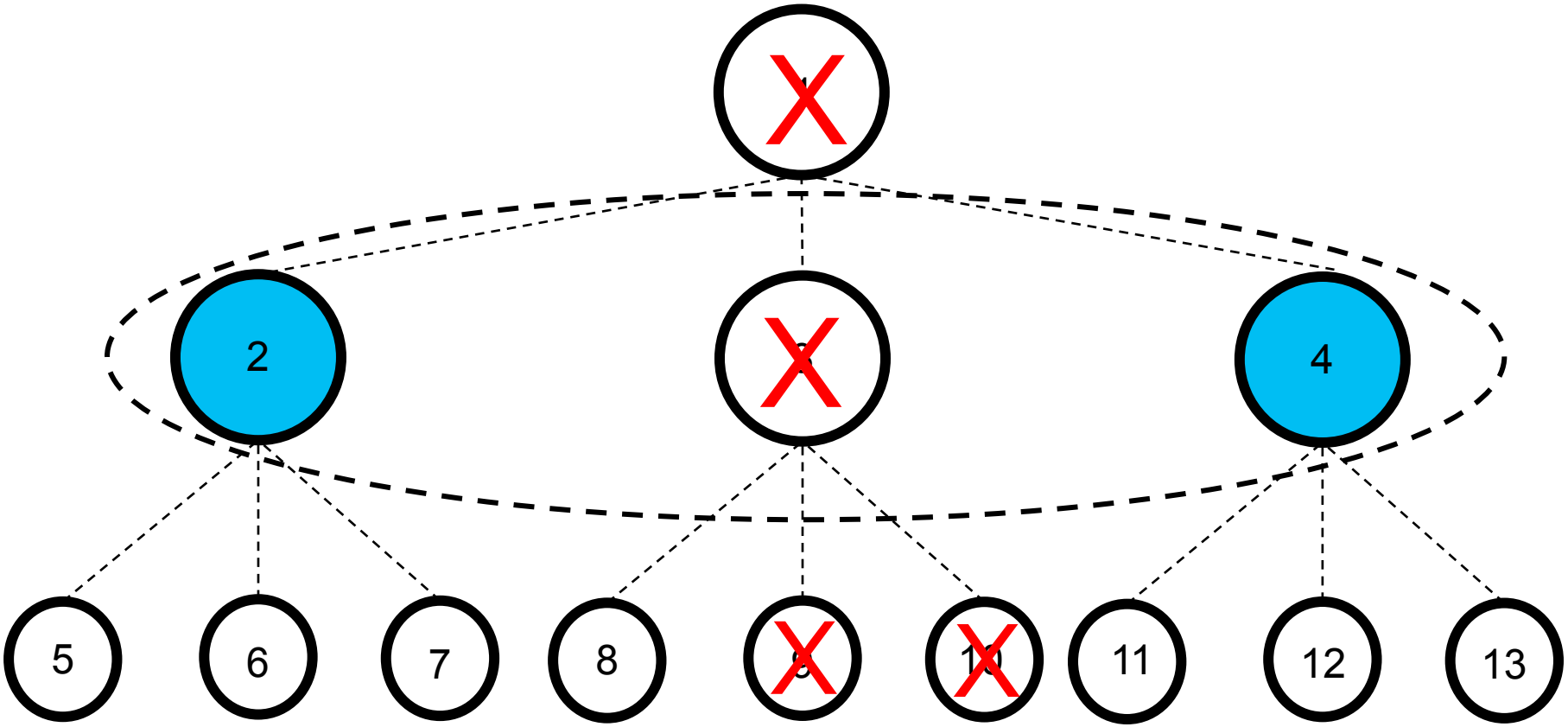
- If root node fails, the closest majority set is selected for replacement
- A new read quorum selected

FLOODING PROTOCOL: EXAMPLE OF READ QUORUM



- Closest majority set of nodes is selected recursively

FLOODING PROTOCOL: EXAMPLE OF READ QUORUM



- No available read quorum when reach the bottom
- Go up one level, find closest majority set
- A read quorum will finally be probed if at least one read quorum lives

QR MODEL: ANALYSIS

- Correctness: 1-copy serializability

- Communication cost when k nodes fails
  - Read and write: $O(k \cdot d(v, q{\downarrow}r(v)))$, where $q{\downarrow}r(v)$ is a live read quorum
  - Request-commit, commit and abort: $O(k \cdot d(v, q{\downarrow}w(v)))$, where $q{\downarrow}w(v)$ is a live write quorum

- Availability: similar to the classic tree quorum system
  - Can afford a certain level of node failures when at least one read and one write quorums live in the system

CONCLUSIONS

- Fast read/write operations, resolve conflicts at commit phase
- When no node fails, provide competitive communication cost as SC model
- When nodes fail, communication cost increases linearly as the number of failed nodes increases
- Provide similar availability as classic tree quorum system

❑ Future…

- Implementation
- Nested transactions?
- Other quorum systems to balance load?
- Reduce communication/improve concurrency?