Distributed Systems Transactions - II

Wouter Joosen

DistriNet, KULeuven

November 14, 2023






Overview

- Part 1: Transactions - recap

- Part 2: *Distributed* transactions
  - Flat and nested distributed transactions
  - Atomic commit protocols
  - Concurrency in distributed transactions
  - Distributed deadlocks
  - Transaction recovery

- Part 3: Replication

Distributed transactions

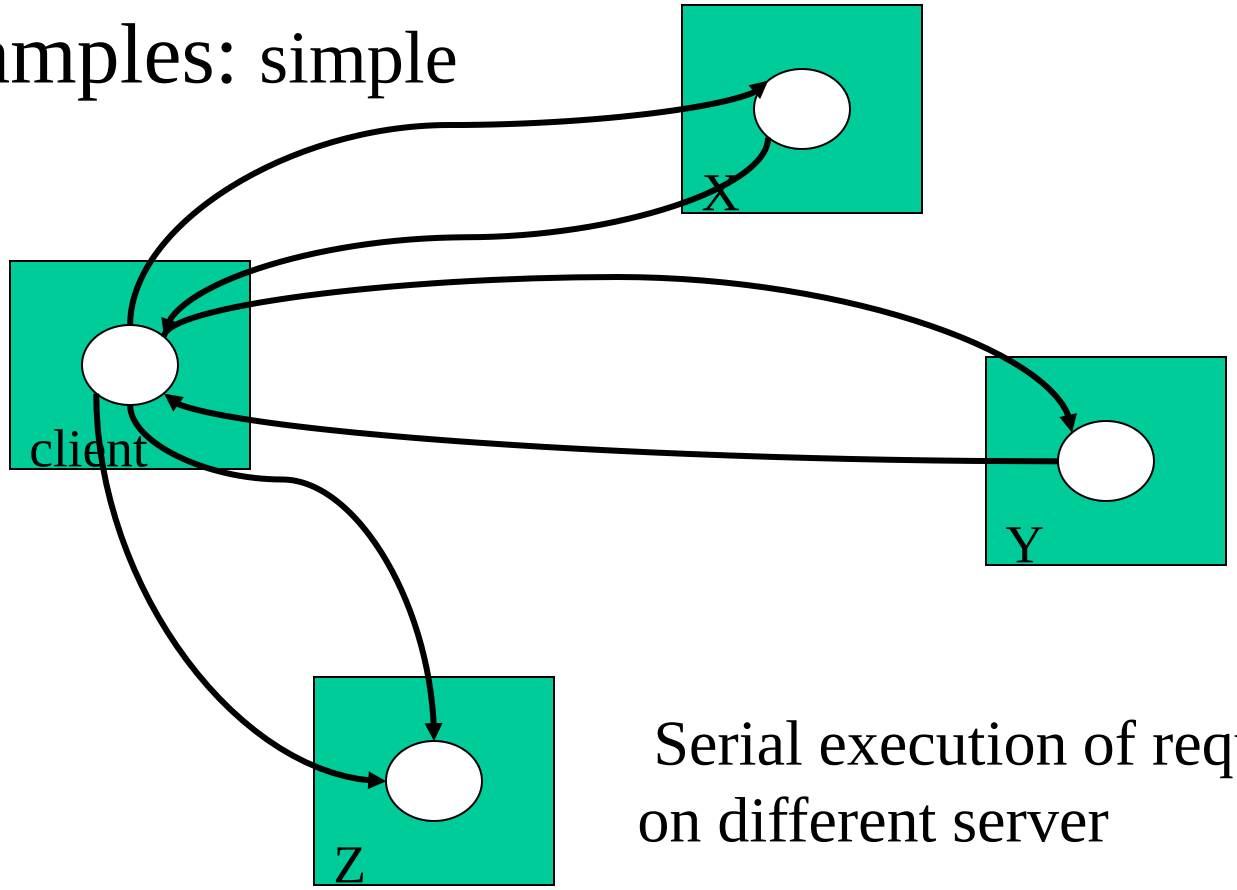
- Definition

  Any transaction whose activities involve multiple servers

- Examples
  - simple: client accesses several servers
  - nested: server accesses several other servers

Distributed transactions

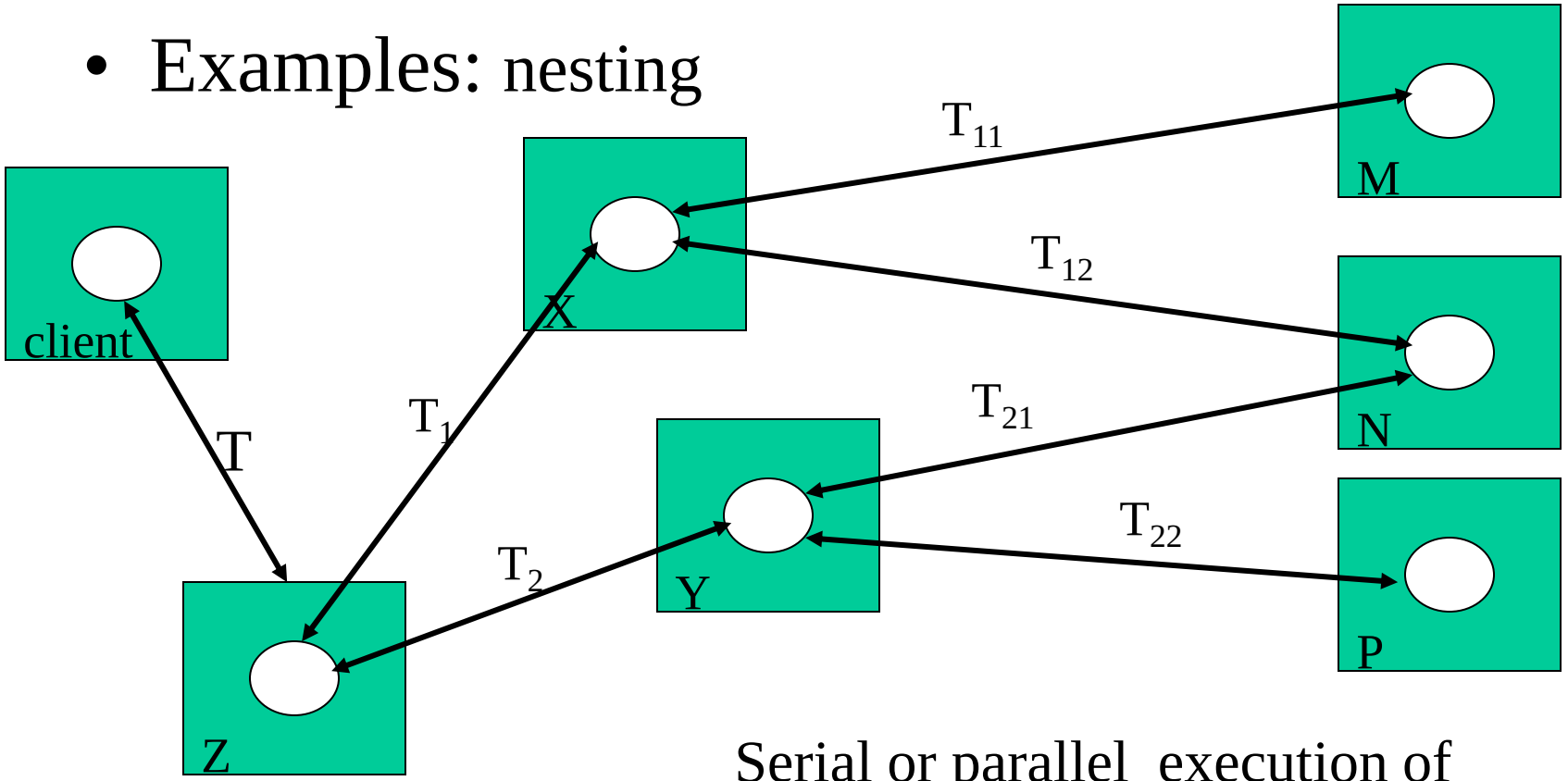
- Examples: simple



Serial execution of requests on different server

Distributed transactions

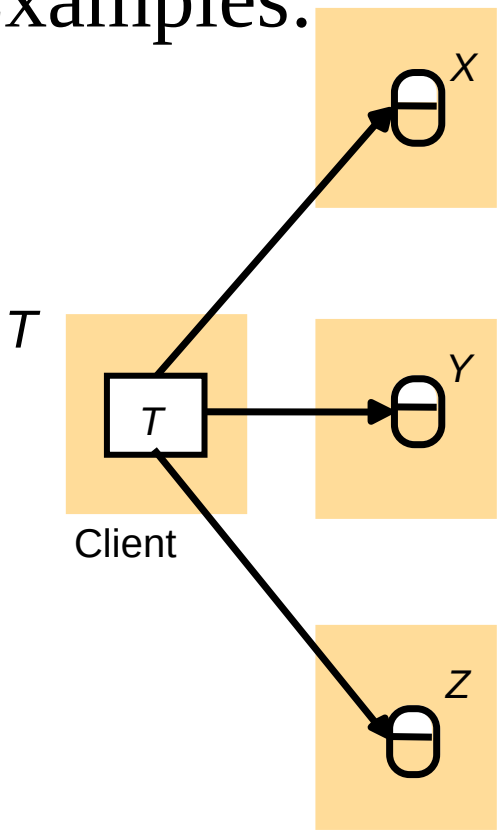
- Examples: nesting

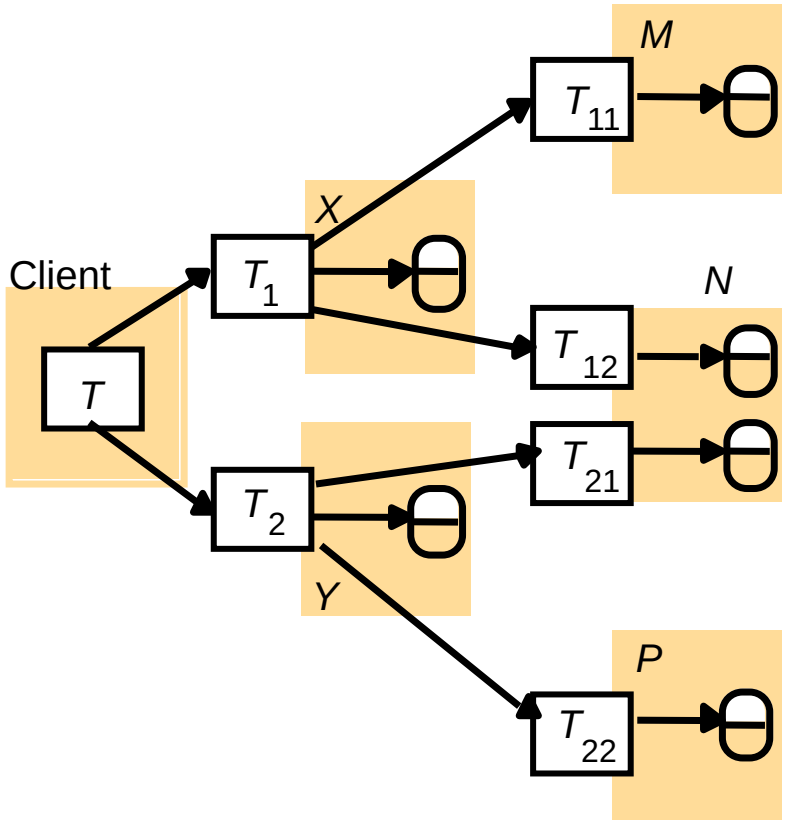


Serial or parallel execution of requests on different servers

Distributed transactions

- Examples:





Distributed transactions

- Commit: agreement between all servers involved
  - to commit
  - to abort
- take one server as coordinator

  ➔simple (?) protocol

  – single point of failure?
- tasks of the coordinator

  – keep track of other servers, called workers

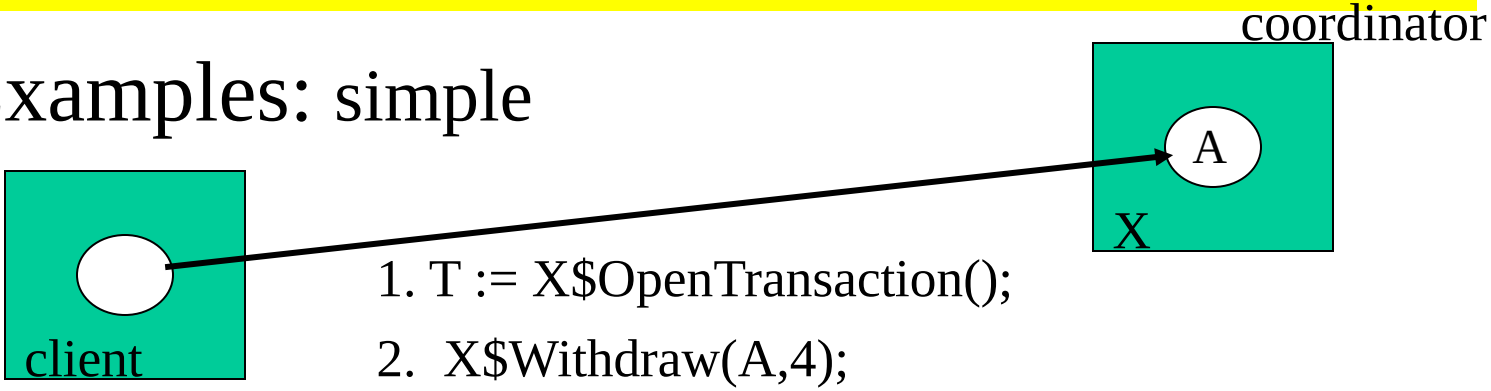
  – responsible for final decision

Distributed transactions

- New service operations:
  - *AddServer(TransID, CoordinatorID)*
    - called by clients
    - first operation on server that has not joined the transaction yet
  - *NewServer(TransID, WorkerID)*
    - called by new server on the coordinator
    - coordinator records ServerID of the worker in its workers list

Distributed transactions

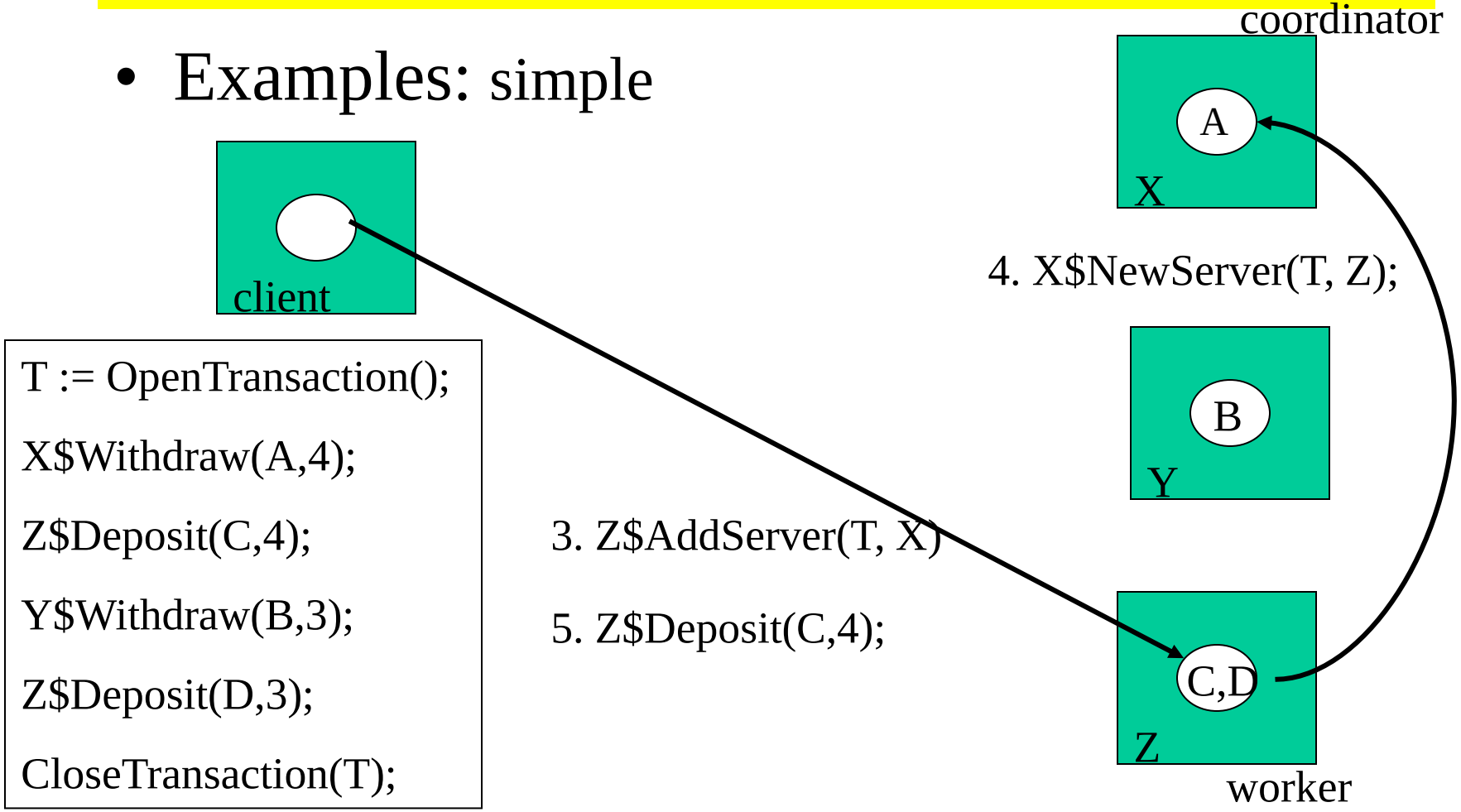
- Examples: simple




```
T := OpenTransaction();
X$Withdraw(A,4);
Z$Deposit(C,4);
Y$Withdraw(B,3);
Z$Deposit(D,3);
CloseTransaction(T);
```

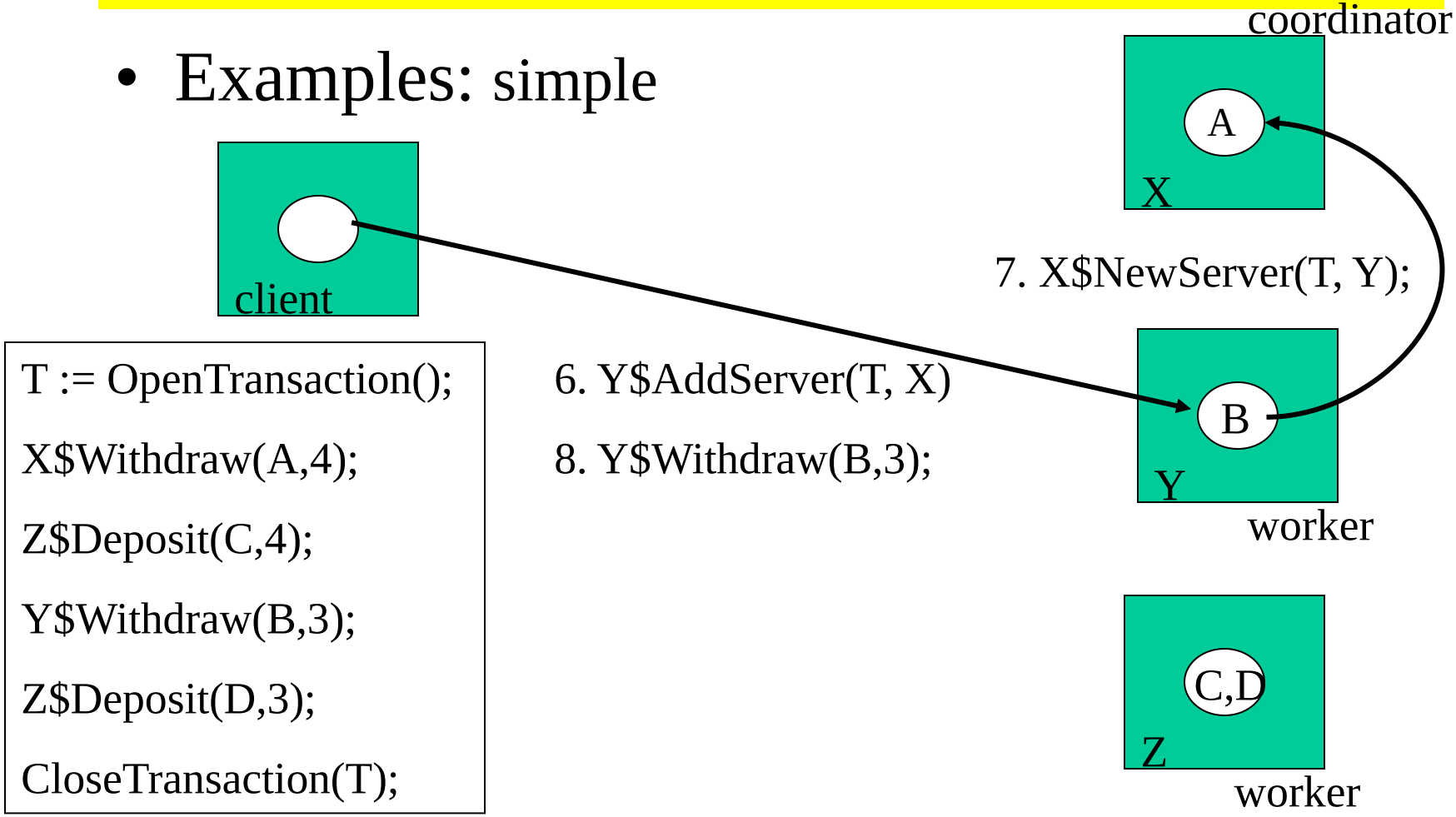
Distributed transactions

- Examples: simple



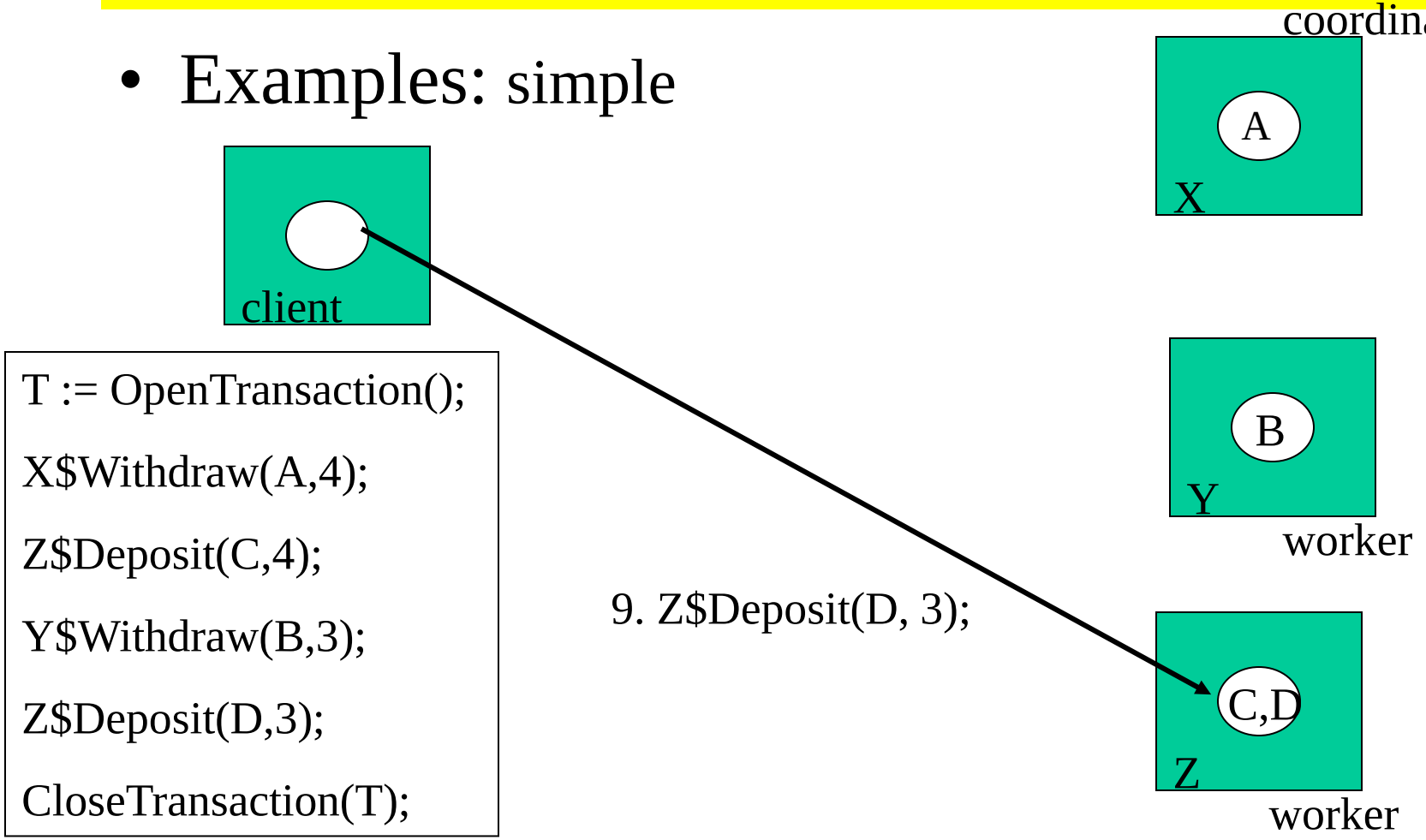
Distributed transactions

- Examples: simple



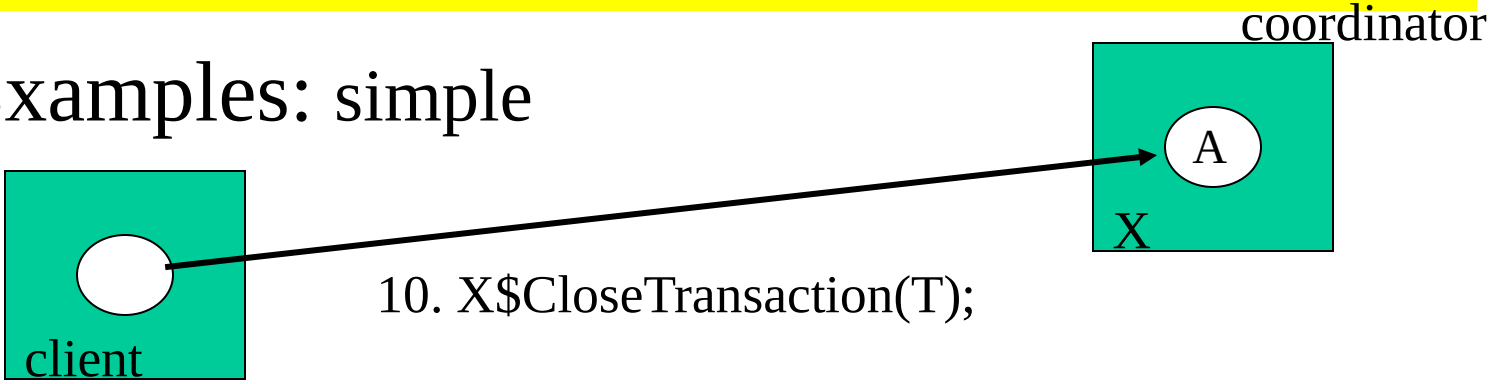
Distributed transactions

- Examples: simple



Distributed transactions

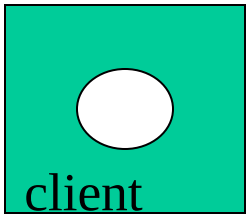
- Examples: simple

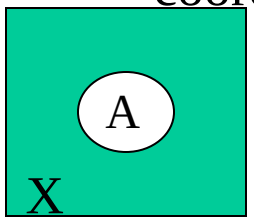


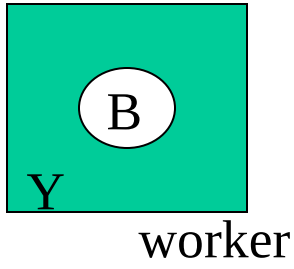

```
T := OpenTransaction();
X$Withdraw(A,4);
Z$Deposit(C,4);
Y$Withdraw(B,3);
Z$Deposit(D,3);
CloseTransaction(T);
```

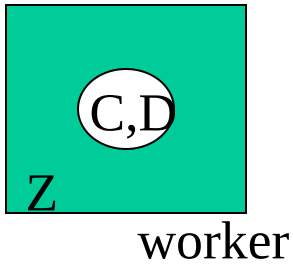
Distributed transactions

- Examples: data at servers









Server	Trans	Role	Coord.	Workers
X	T	Coord	(here)	Y, Z
Y	T	Worker	X	
Z	T	Worker	X	

Overview

- Transactions

- Distributed transactions
  - Flat and nested distributed transactions
  - Atomic commit protocols
  - Concurrency in distributed transactions
  - Distributed deadlocks
  - Transaction recovery

- Replication

Overview

- Transactions

- Distributed transactions
  - Flat and nested distributed transactions
  - Atomic commit protocols
  - Concurrency in distributed transactions
  - Distributed deadlocks
  - Transaction recovery

- Replication

Atomic Commit protocol

- Elements of the protocol
  - each server is allowed to abort its part of a transaction
  - if a server votes to commit it must ensure that it will eventually be able to carry out this commitment
    - the transaction must be in the prepared state
    - all altered data items must be on permanent storage
  - if any server votes to abort, then the decision must be to abort the transaction

Atomic Commit protocol

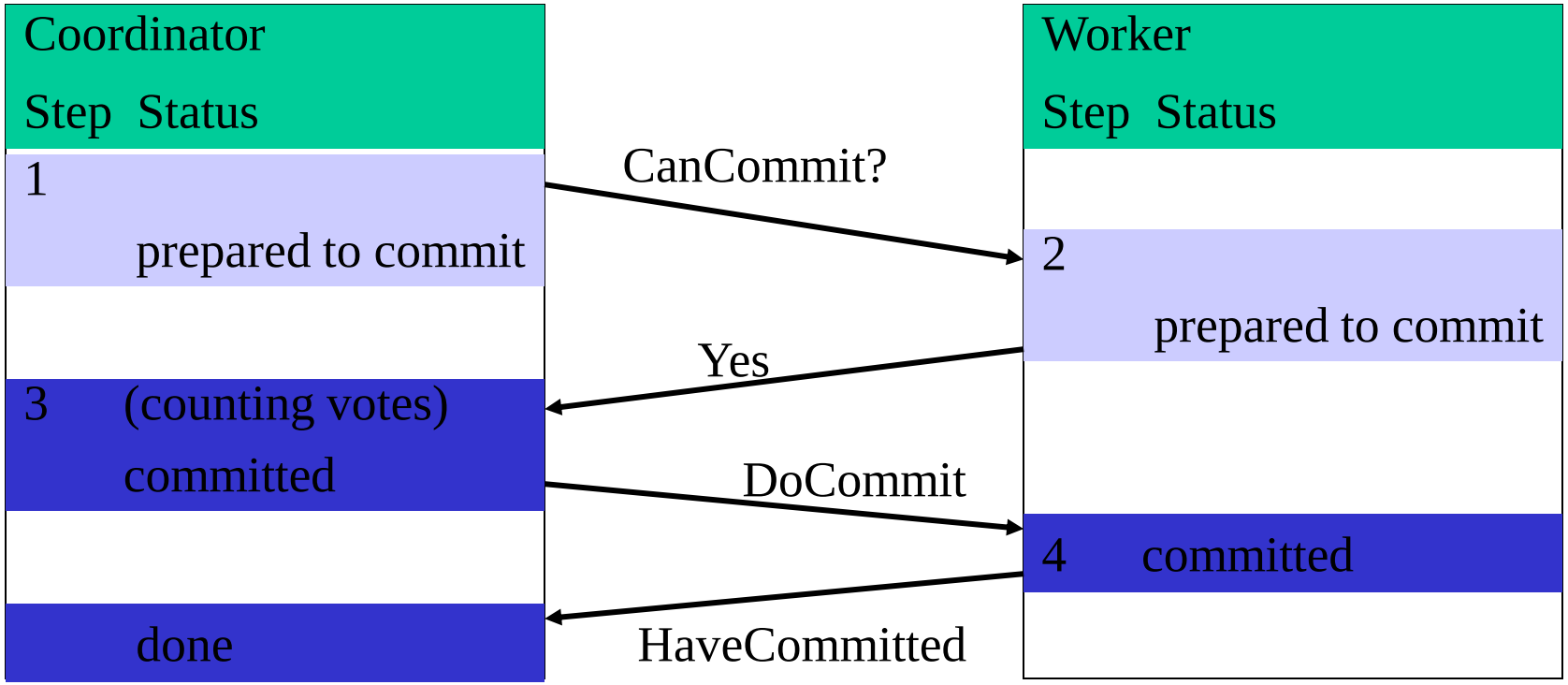
- Elements of the protocol *(cont.)*
  - the protocol must work correctly, even when
    - some servers fail
    - messages are lost
    - servers are temporarily unable to communicate

Atomic Commit protocol

- Protocol:
  - Phase 1: voting phase
  - Phase 2: completion according to outcome of vote

Atomic Commit protocol

- Protocol

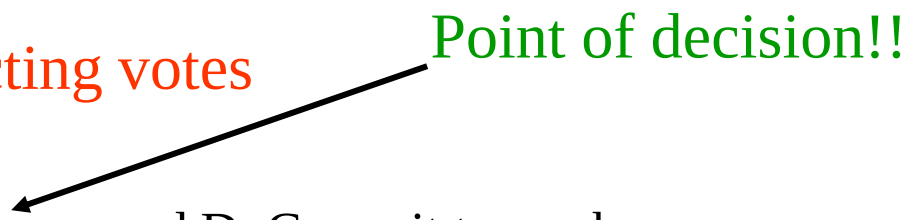


Atomic Commit protocol

- Protocol: Phase 1 voting phase
  - Coordinator: for operation CloseTransaction
    - sends CanCommit to each worker
    - behaves as worker in phase 1
    - waits for replies from workers
  - Worker: when receiving CanCommit
    - if for worker transaction can commit
      - saves data items
      - sends Yes to coordinator
    - if for worker transaction cannot commit
      - sends No to coordinator
      - clears data structures, removes locks

Atomic Commit protocol

- Protocol: Phase 2
  - Coordinator: collecting votes
    - all votes Yes:
      ➔commit transaction; send DoCommit to workers
    - one vote No:
      ➔abort transaction
  - Worker: voted yes, waits for decision of coordinator
    - receives DoCommit
      - makes committed data available; removes locks
    - receives AbortTransaction
      - clears data structures; removes locks



Atomic Commit protocol

- Timeouts:
  - worker did all/some operations and waits for CanCommit
    - unilateral abort possible
  - coordinator waits for votes of workers
    - unilateral abort possible
  - worker voted Yes and waits for final decision of coordinator
    - wait unavoidable
    - extensive delay possible
    - additional operation *GetDecision* can be used to get decision from coordinator or other workers

Atomic Commit protocol

- Performance:
  - C ➔ W: CanCommit N-1 messages
  - W ➔ C: Yes/No N-1 messages
  - C ➔ W: DoCommit N-1 messages
  - W ➔ C: HaveCommitted N-1 messages

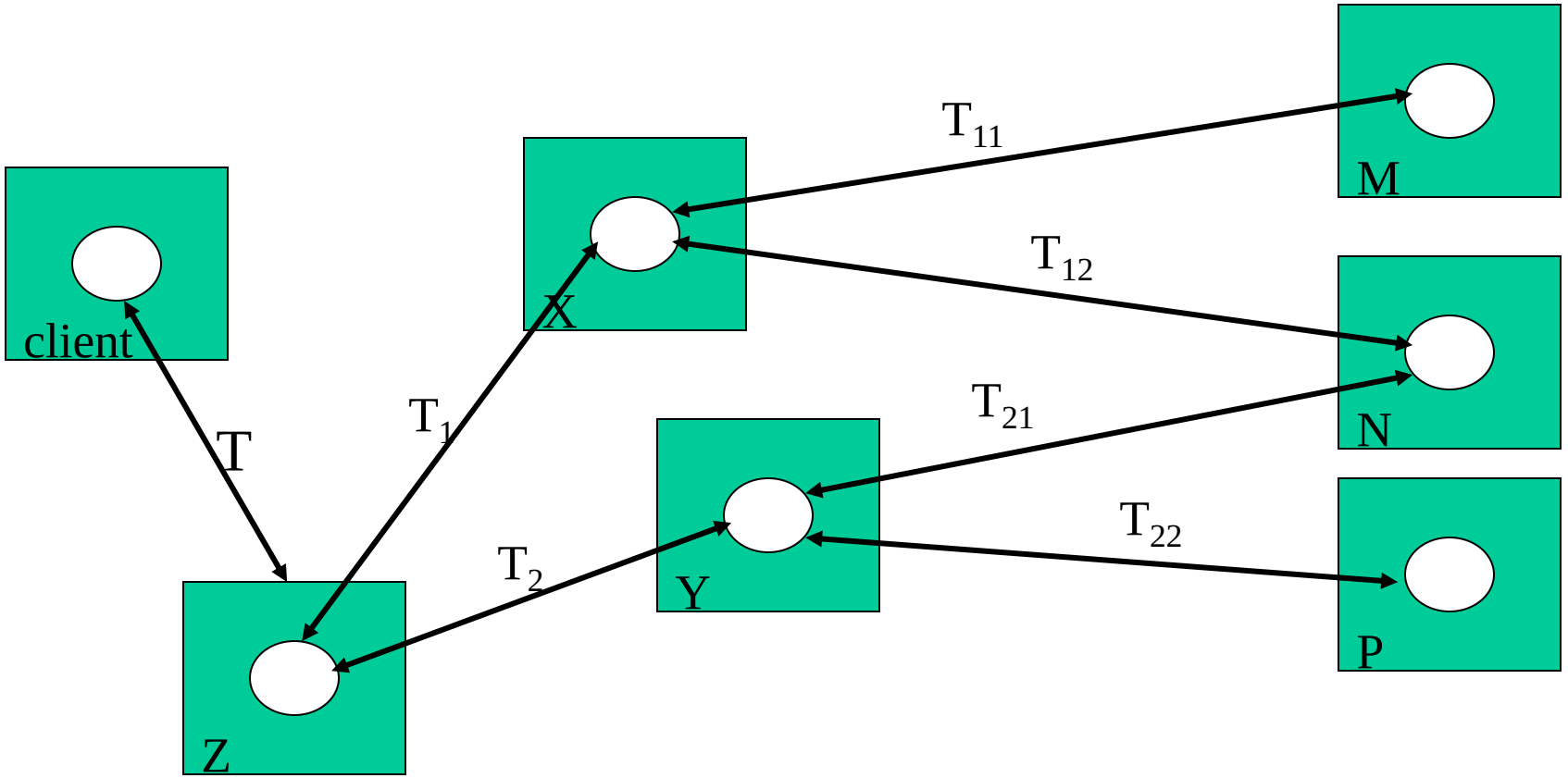
  \+ (unavoidable) delays possible

Atomic Commit protocol

- Nested Transactions
  - top level transaction & subtransactions

  ➔ transaction tree

Atomic Commit protocol



Atomic Commit protocol

- Nested Transactions
  - top level transaction & subtransactions

  ➔ transaction tree
  - coordinator = top level transaction
  - subtransaction identifiers
    - globally unique
    - allow derivation of ancestor transactions (why necessary?)

Atomic Commit protocol

- Nested Transactions: Transaction IDs

TID in example	actual TID
T	Z, n_z
T_1	Z, n_z ; X, n_x
T_{11}	Z, n_z ; X, n_x ; M, n_m
T_2	Z, n_z ; Y, n_y

Atomic Commit protocol

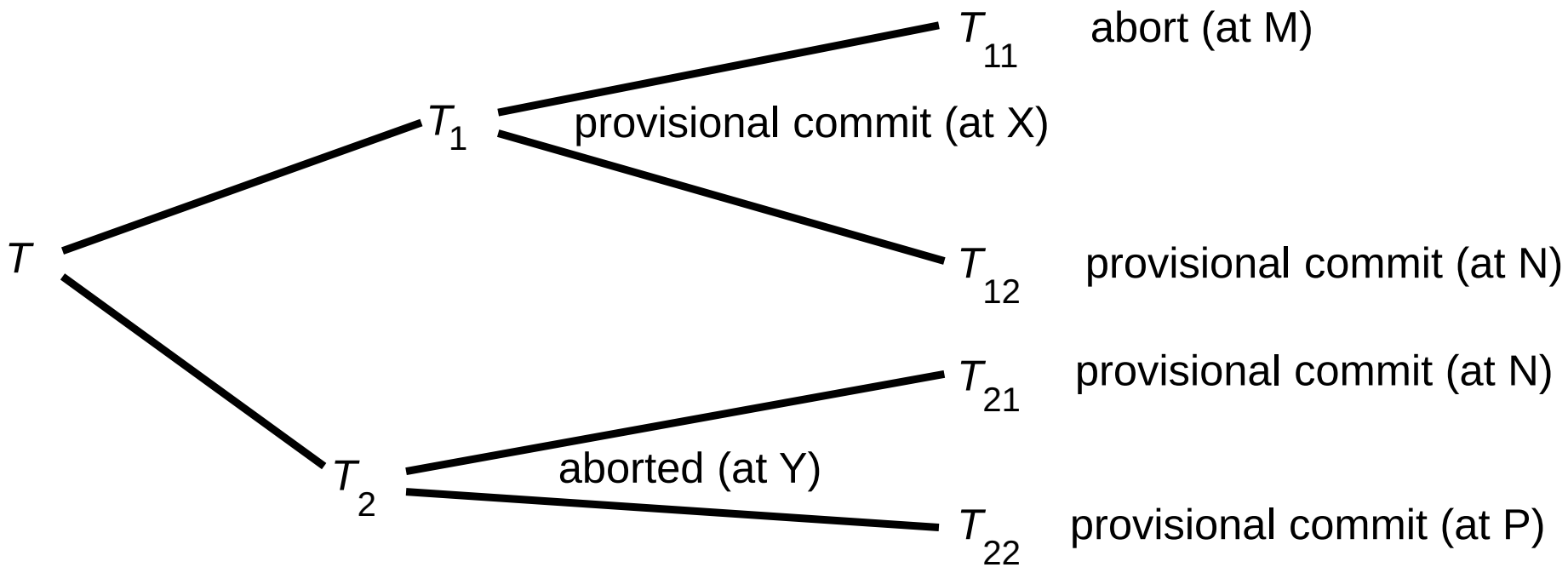
- Upon completion of a subtransaction
  - independent decision to commit or abort
  - commit of subtransaction
    - only provisionally
    - status (including status of descendants) reported to parent
    - final outcome dependant on its ancestors
  - abort of subtransaction
    - implies abort of all its descendants
    - abort reported to its parent (always possible?)

Atomic Commit protocol

- Data structures
  - commit list: list of all committed (sub)transactions
  - aborts list: list of all aborted (sub)transactions
  - example

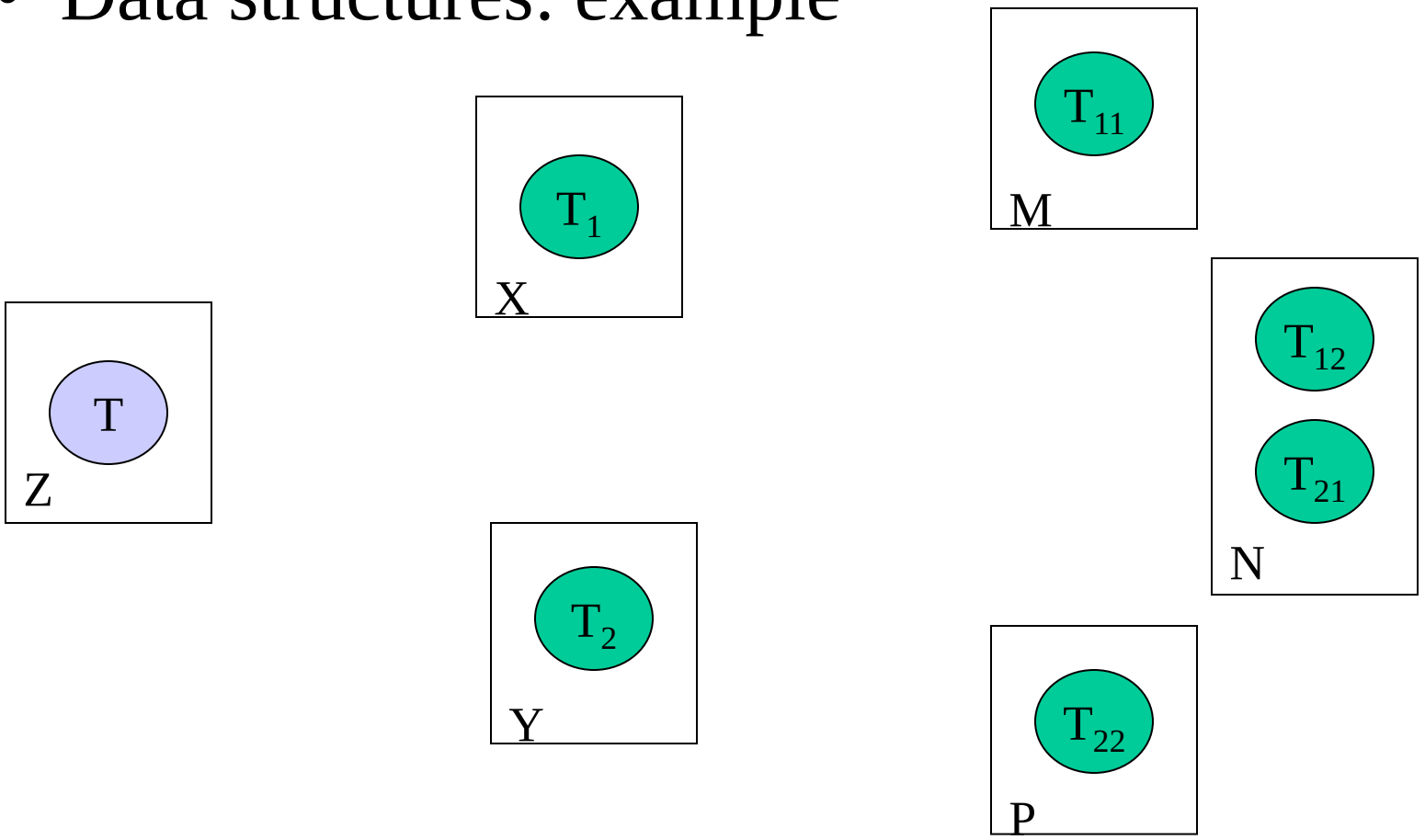
Atomic Commit protocol

- Data structures: example



Atomic Commit protocol

- Data structures: example



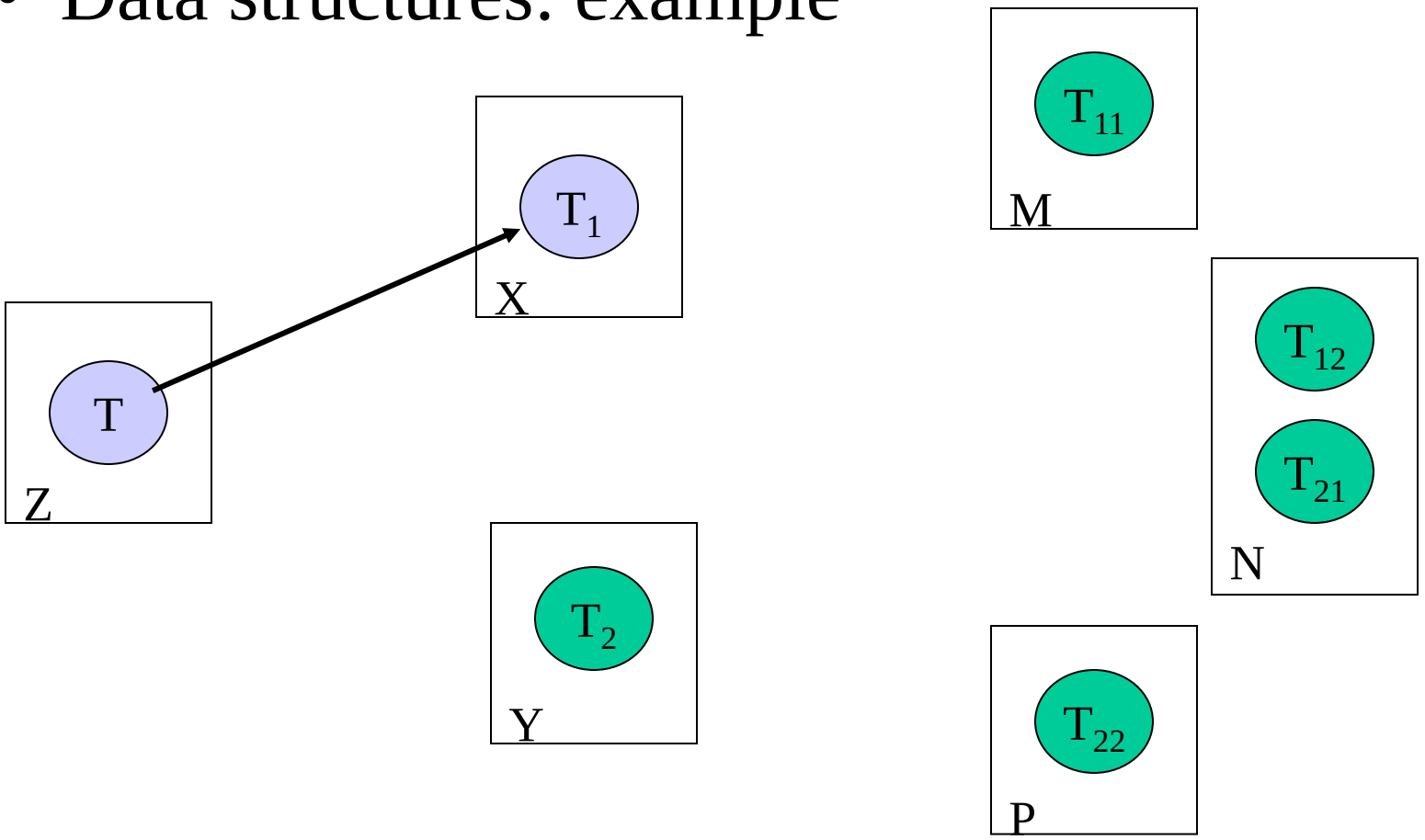
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T			
X				
Y				
M				
N				
P				

Atomic Commit protocol

- Data structures: example



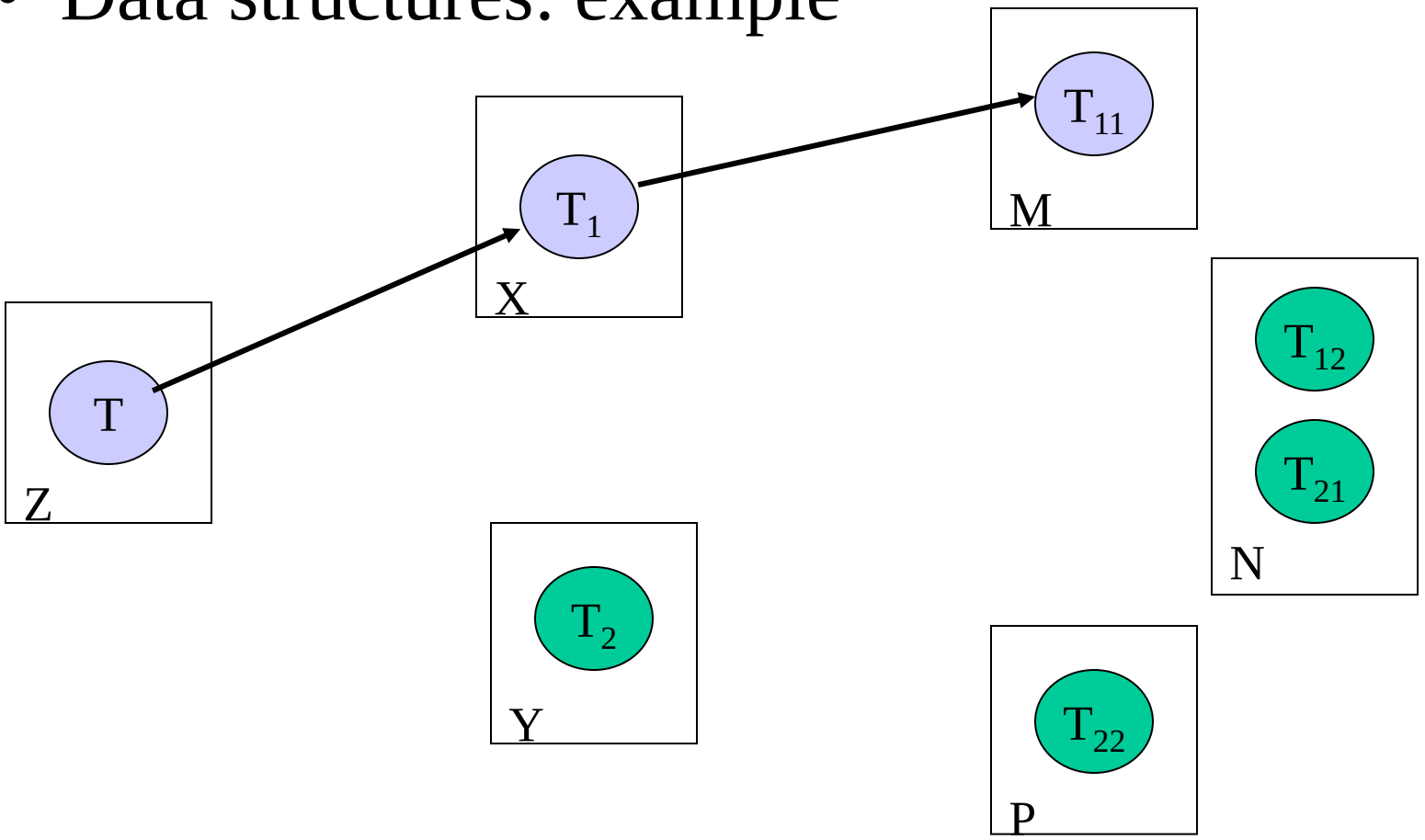
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1		
X	T_1			
Y				
M				
N				
P				

Atomic Commit protocol

- Data structures: example



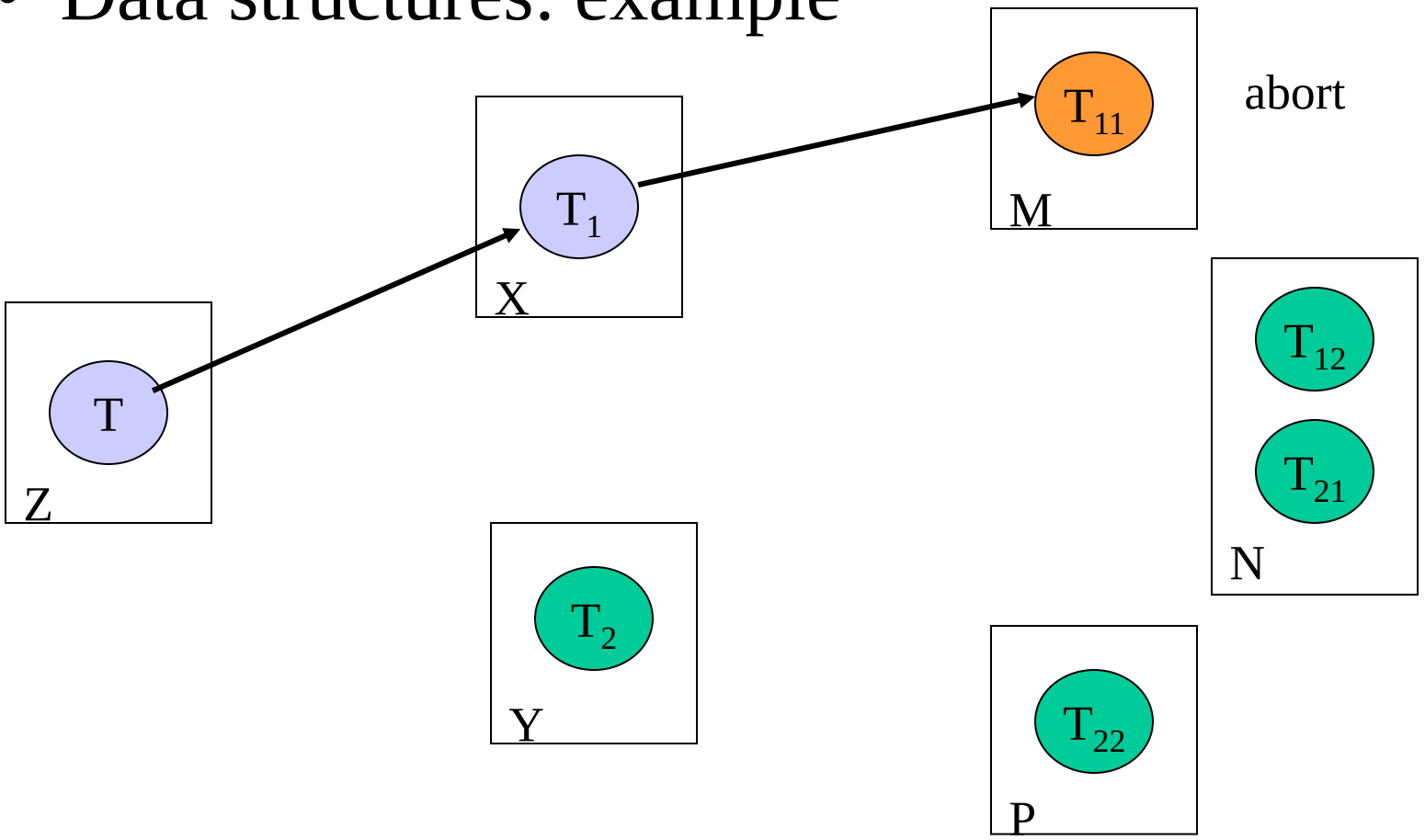
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1		
X	T_1	T_{11}		
Y				
M	T_{11}			
N				
P				

Atomic Commit protocol

- Data structures: example



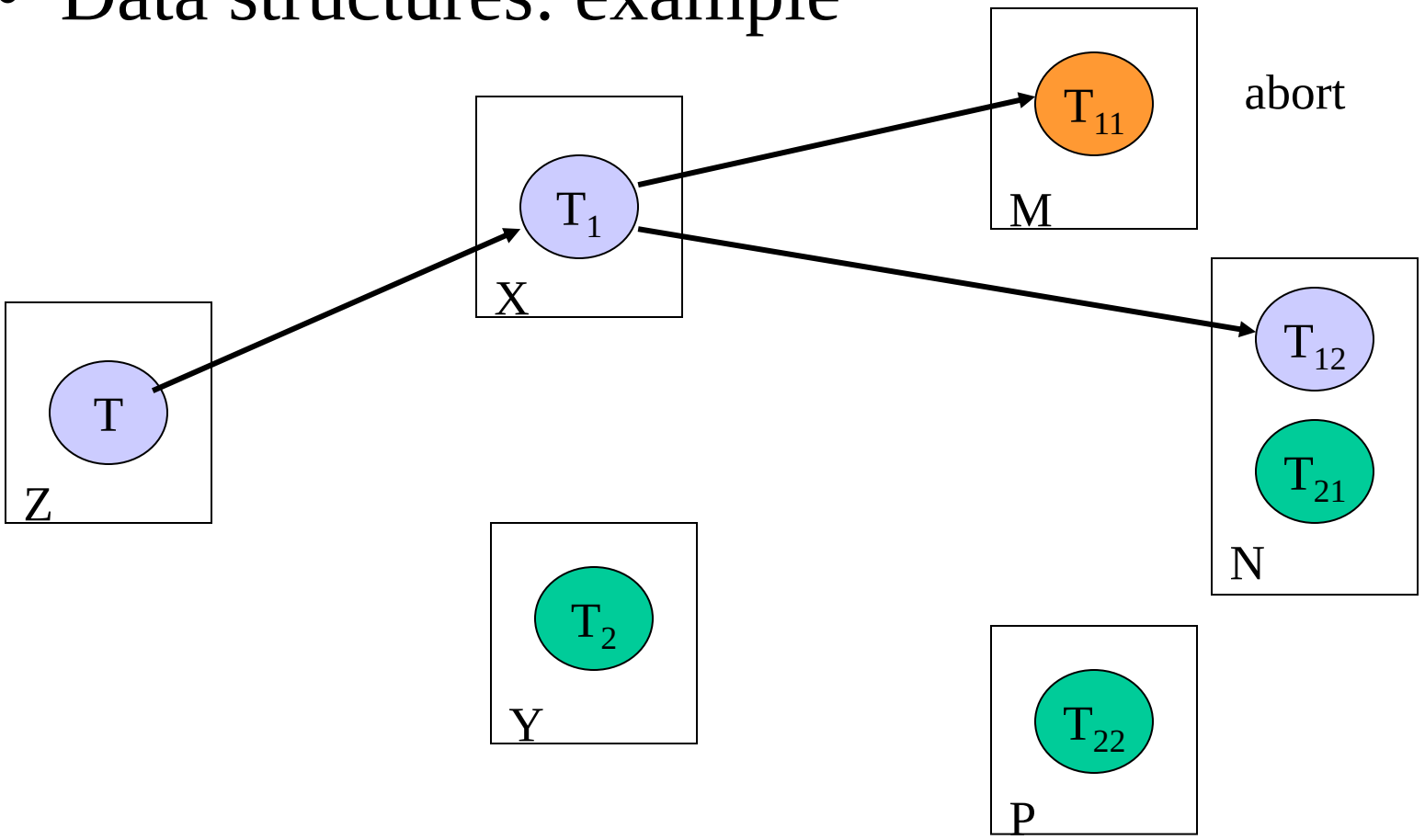
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1		
X	T_1	T_{11}		T_{11}
Y				
M	T_{11}			T_{11}
N				
P				

Atomic Commit protocol

- Data structures: example



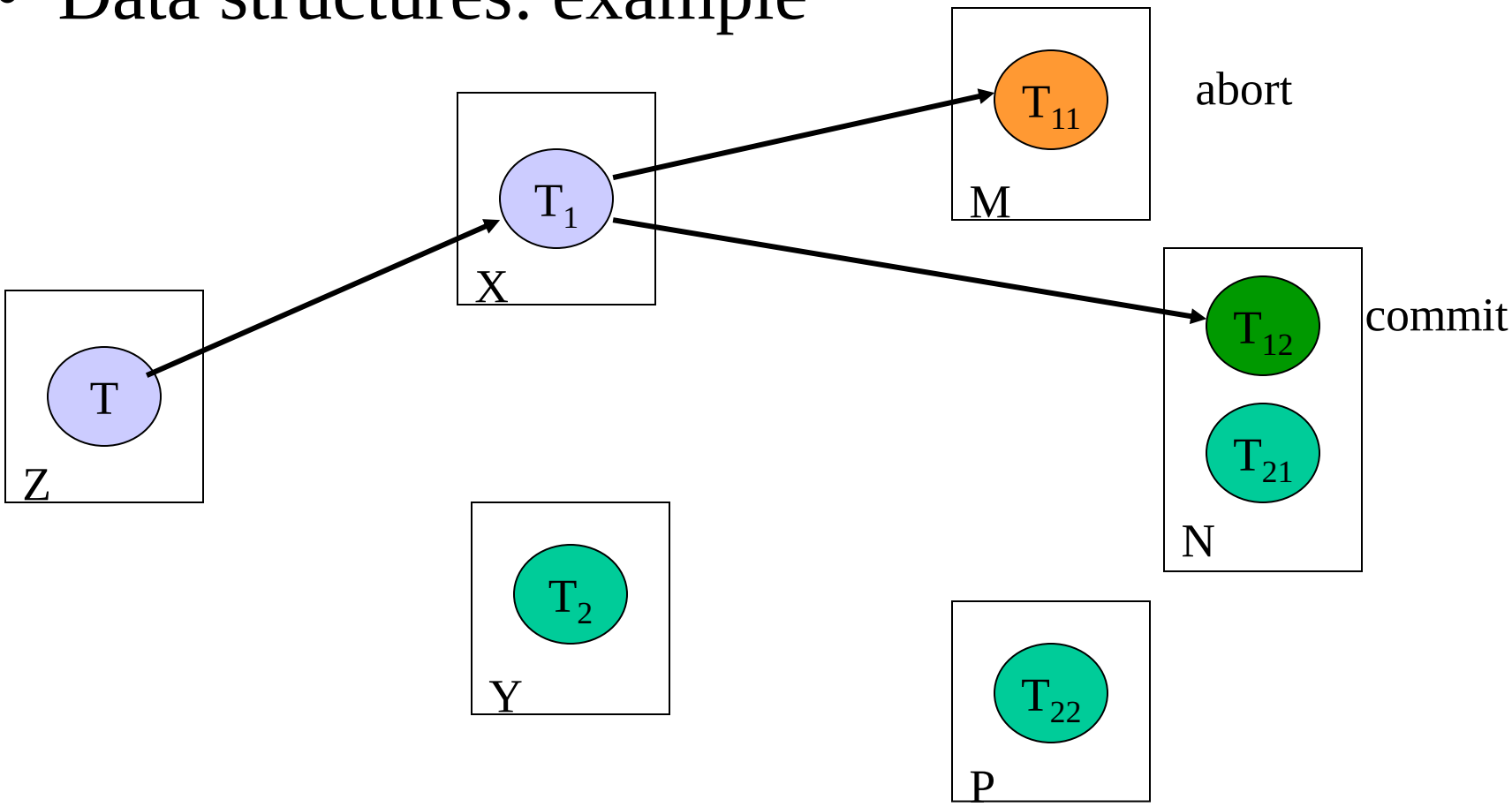
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1		
X	T_1	T_{11} , T_{12}		T_{11}
Y				
M	T_{11}			T_{11}
N	T_{12}			
P				

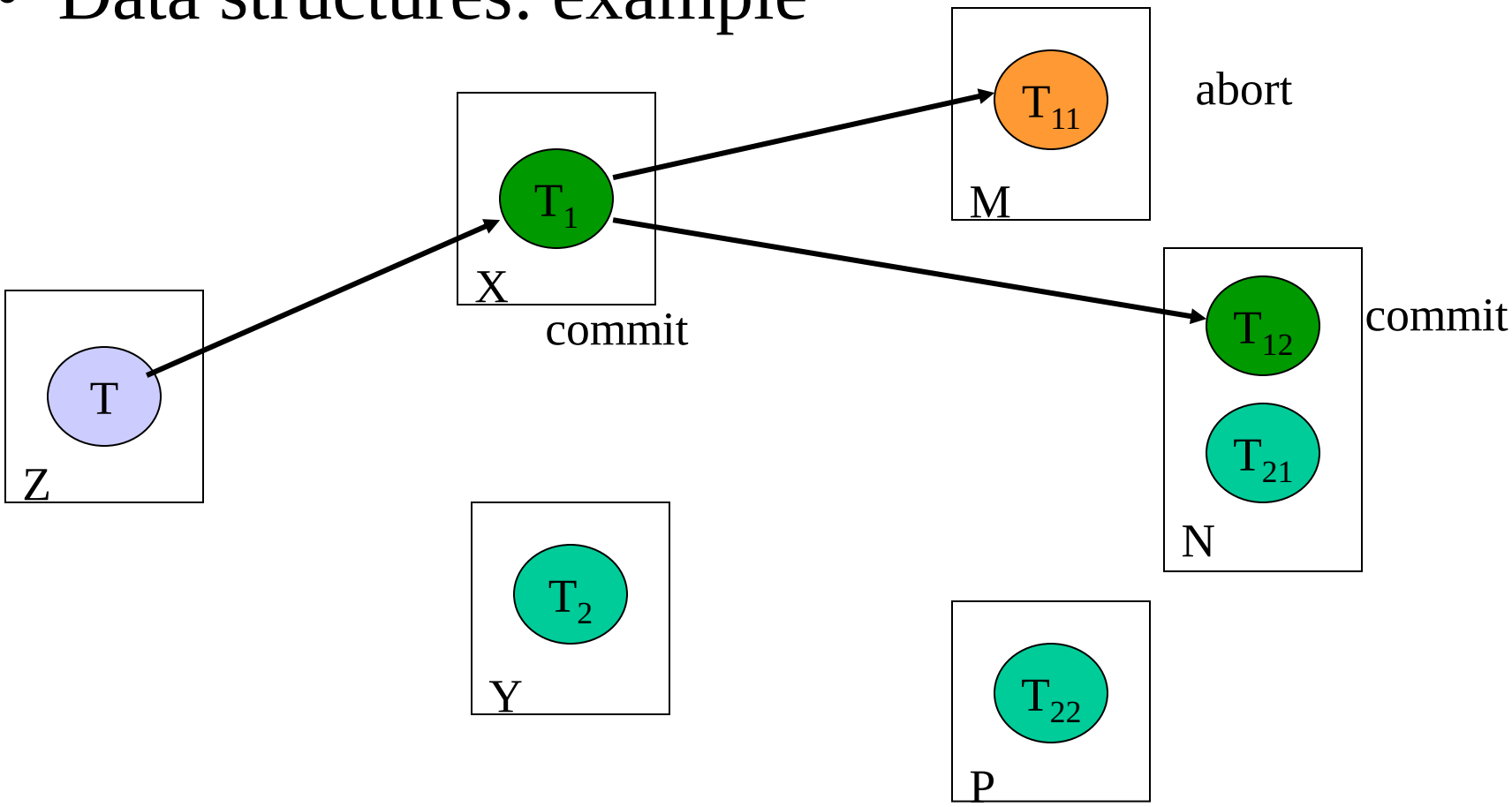
Atomic Commit protocol

- Data structures: example



Atomic Commit protocol

- Data structures: example



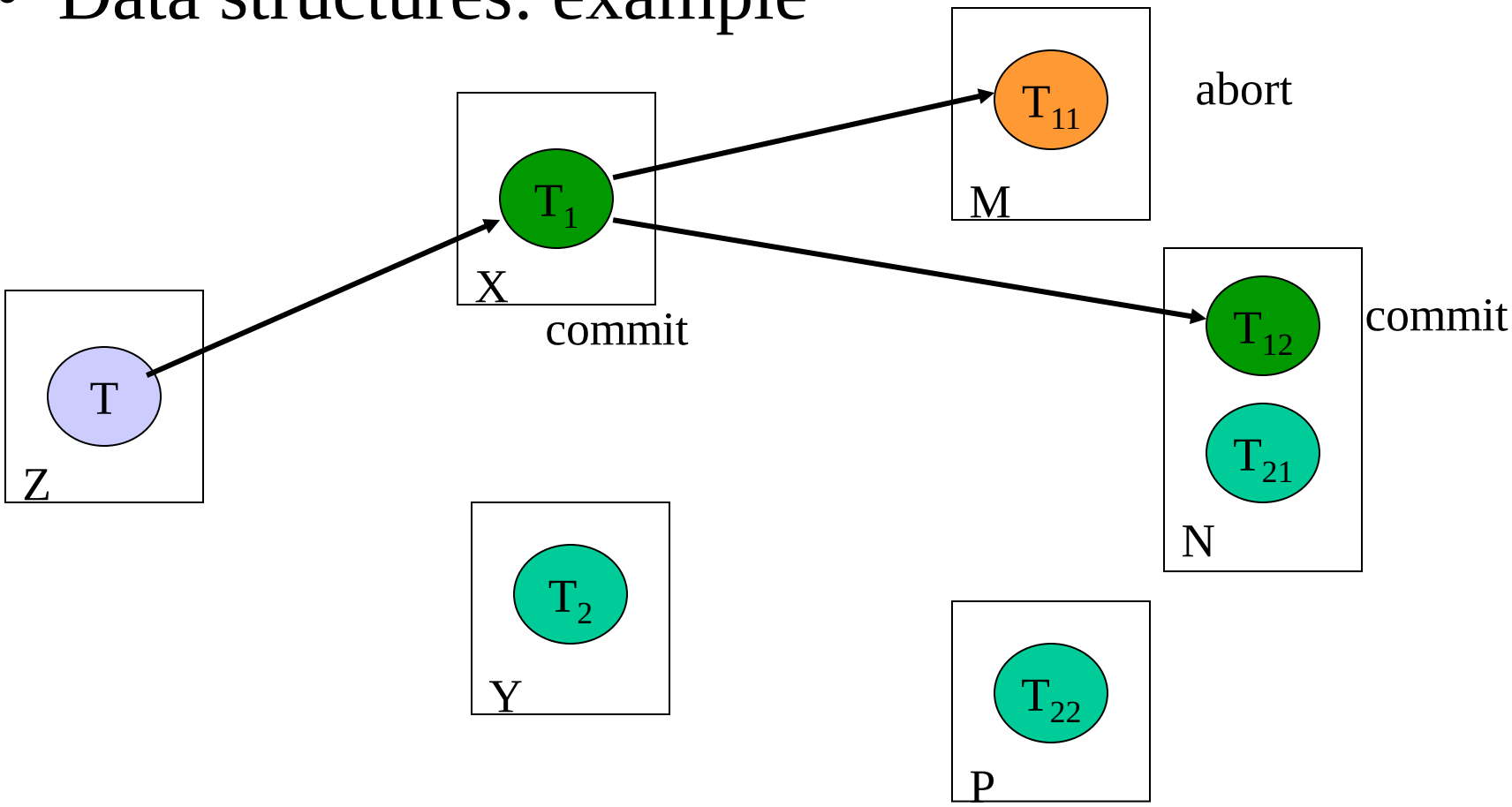
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1		
X	T_1	T_{11} ,T_{12}	T_{12}	T_{11}
Y				
M	T_{11}			T_{11}
N	T_{12}		T_{12}	
P				

Atomic Commit protocol

- Data structures: example



Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1		
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y				
M	T_{11}			T_{11}
N	T_{12}		T_{12}	
P				

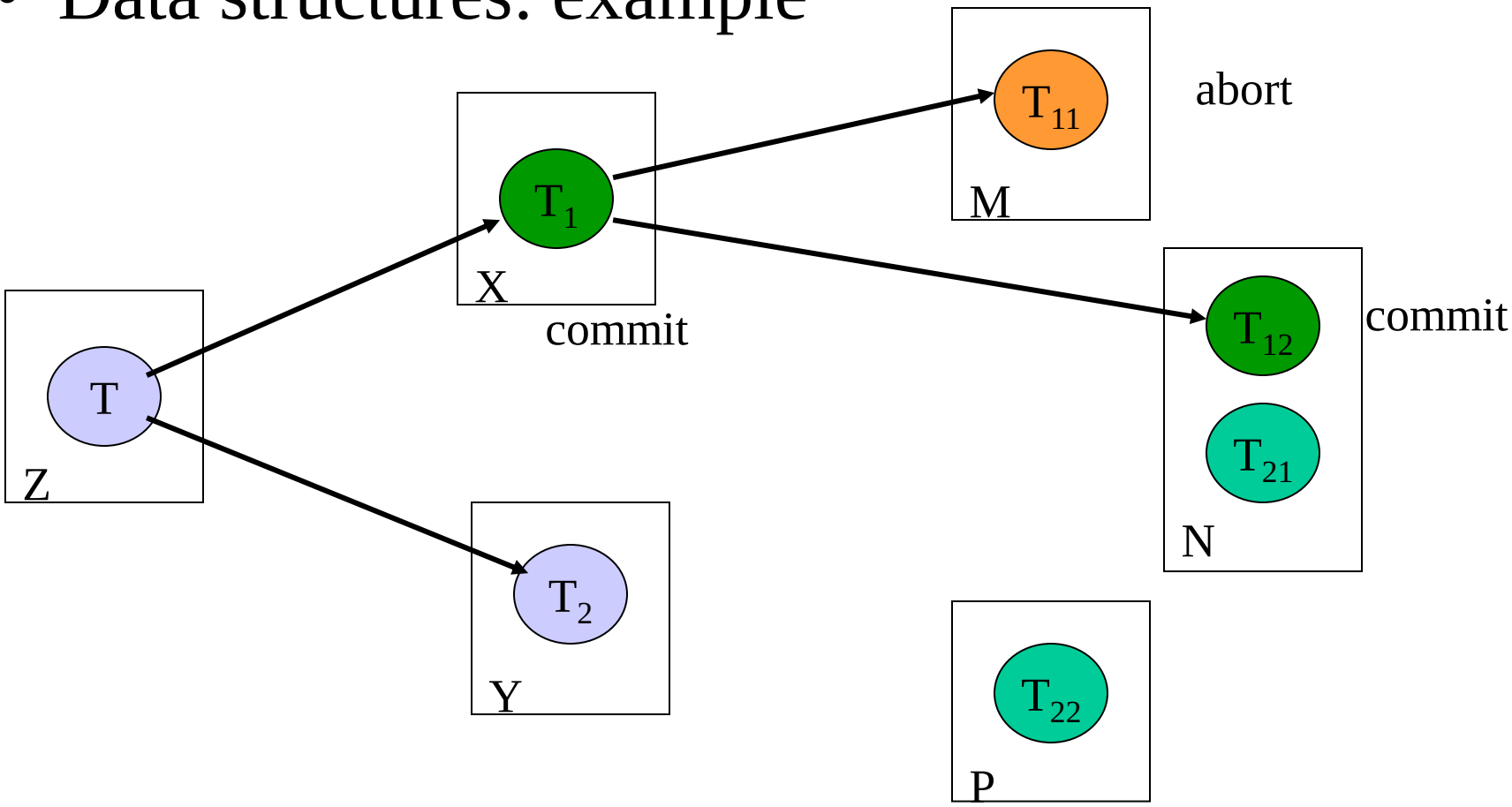
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1	T_{12} , T_1	T_{11}
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y				
M	T_{11}			T_{11}
N	T_{12}		T_{12}	
P				

Atomic Commit protocol

- Data structures: example



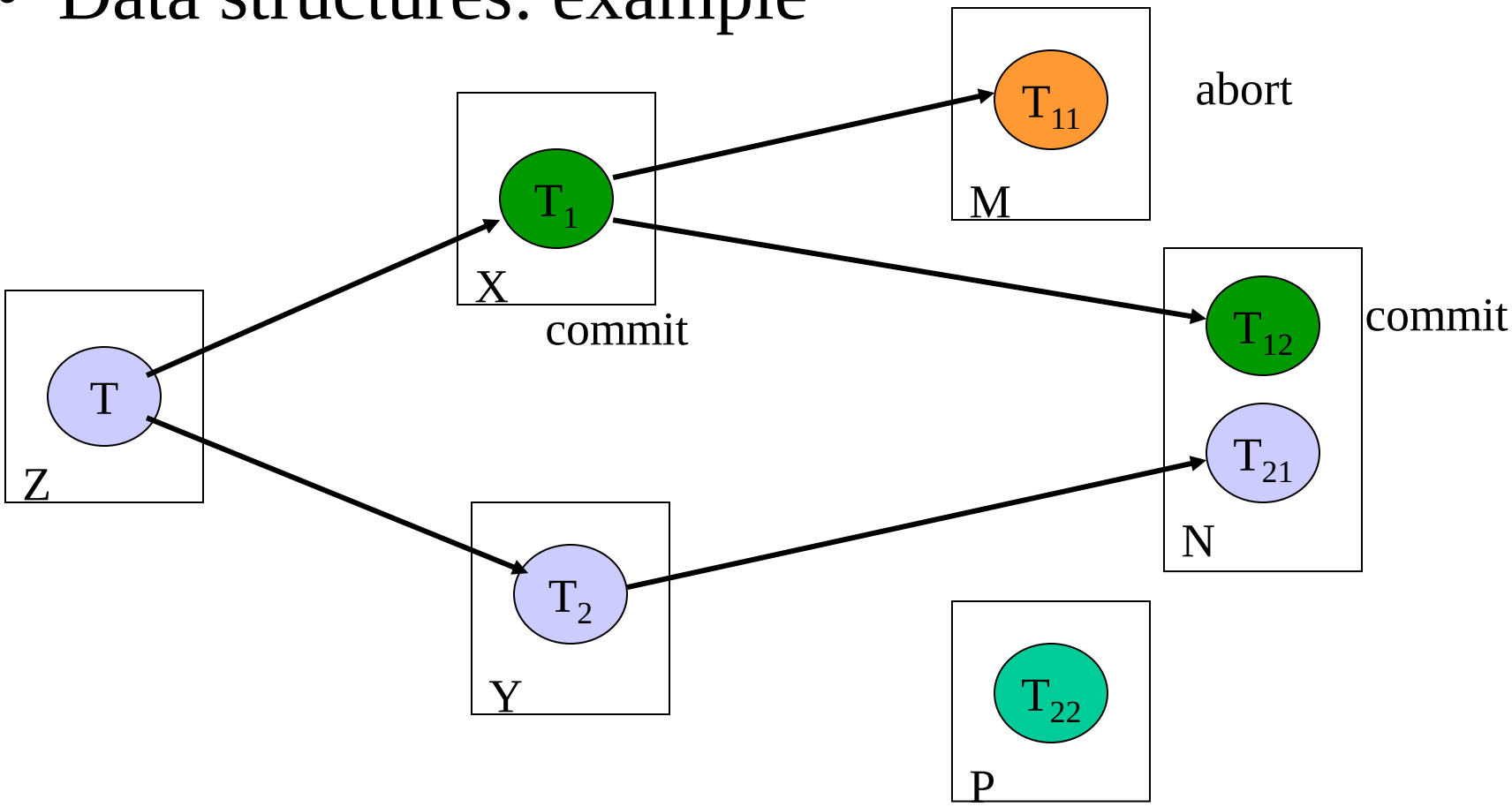
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 , T_2	T_{12} , T_1	T_{11}
X	T_1	T_{11} , T_{12}	T_{12} , T_1	T_{11}
Y	T_2			
M	T_{11}			T_{11}
N	T_{12}		T_{12}	
P				

Atomic Commit protocol

- Data structures: example



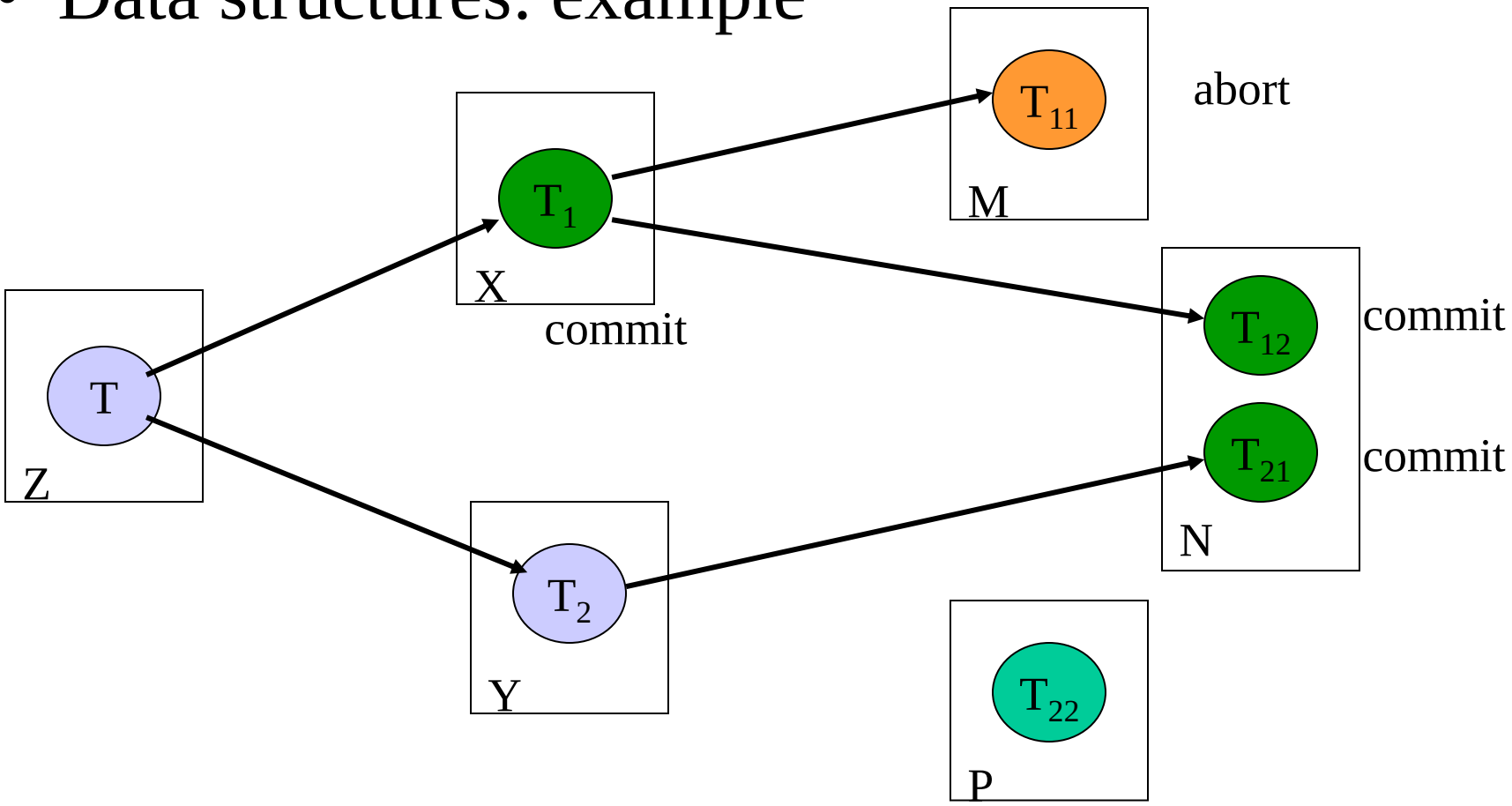
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 ,T_2	T_{12} , T_1	T_{11}
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y	T_2	T_{21}		
M	T_{11}			T_{11}
N	T_{12} ,T_{21}		T_{12}	
P				

Atomic Commit protocol

- Data structures: example



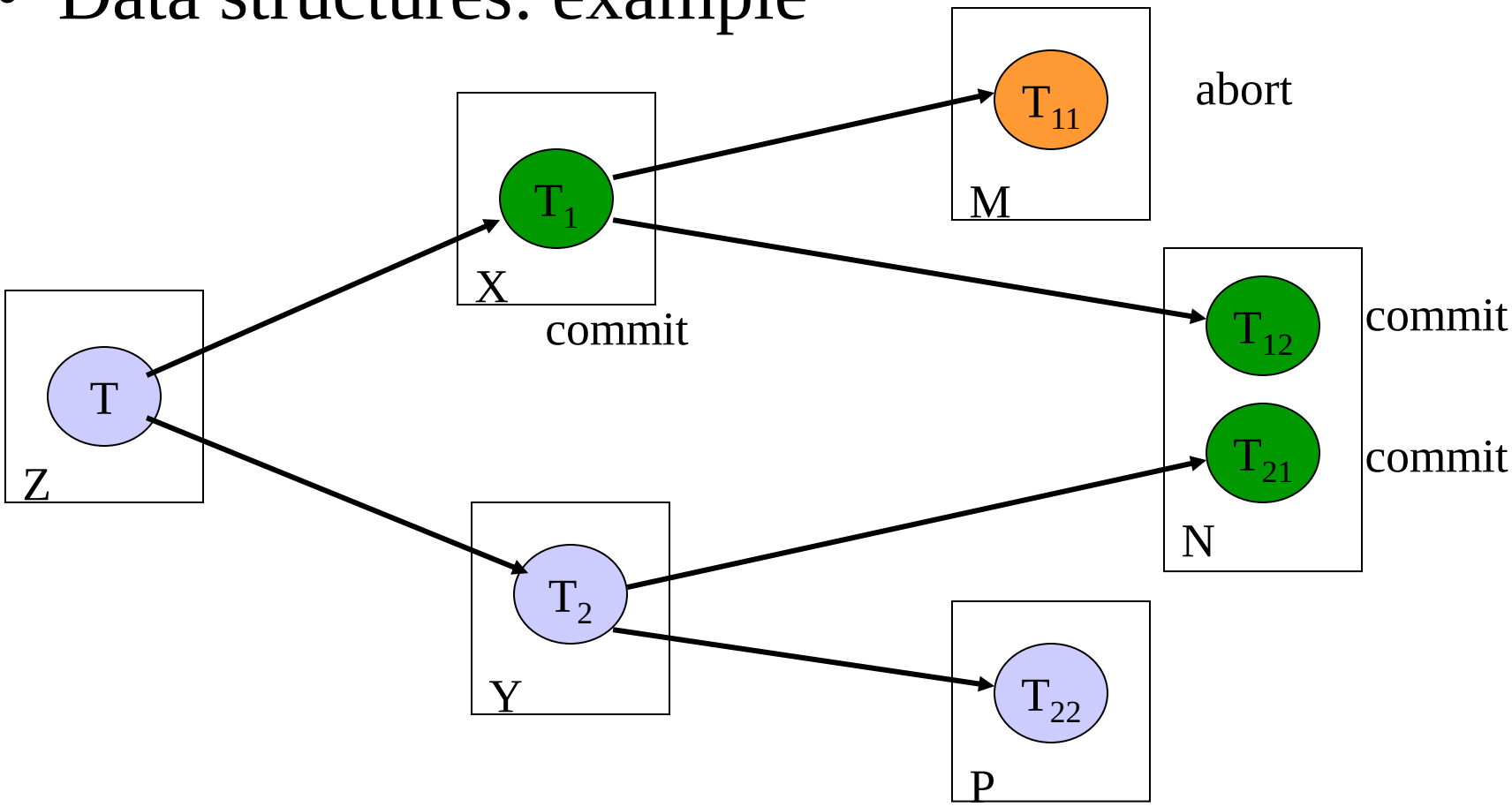
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 ,T_2	T_{12} , T_1	T_{11}
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y	T_2	T_{21}	T_{21}	
M	T_{11}			T_{11}
N	T_{12} ,T_{21}		T_{12} ,T_{21}	
P				

Atomic Commit protocol

- Data structures: example



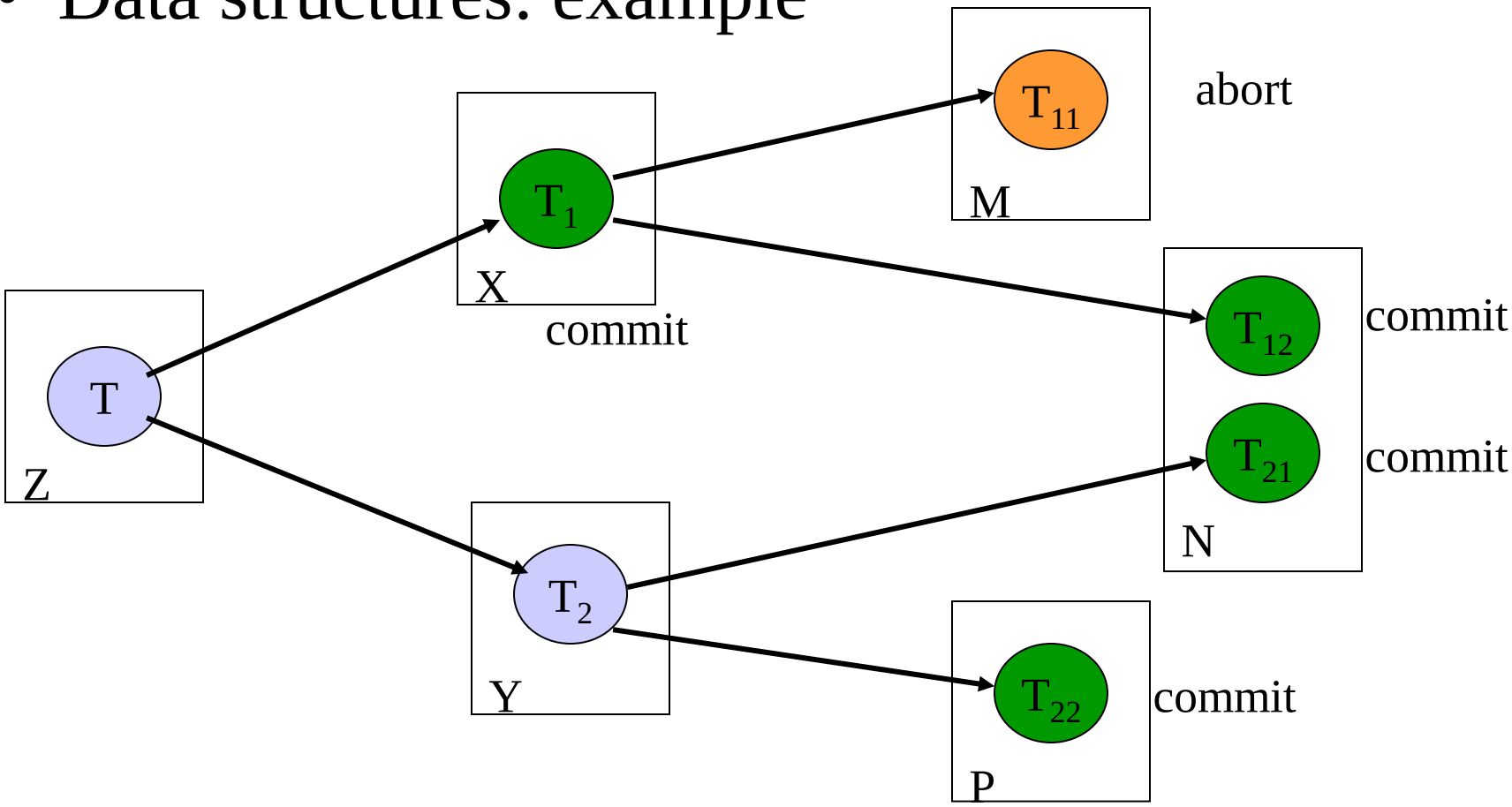
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 ,T_2	T_{12} , T_1	T_{11}
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y	T_2	T_{21} ,T_{22}	T_{21}	
M	T_{11}			T_{11}
N	T_{12} ,T_{21}		T_{12} ,T_{21}	
P	T_{22}			

Atomic Commit protocol

- Data structures: example



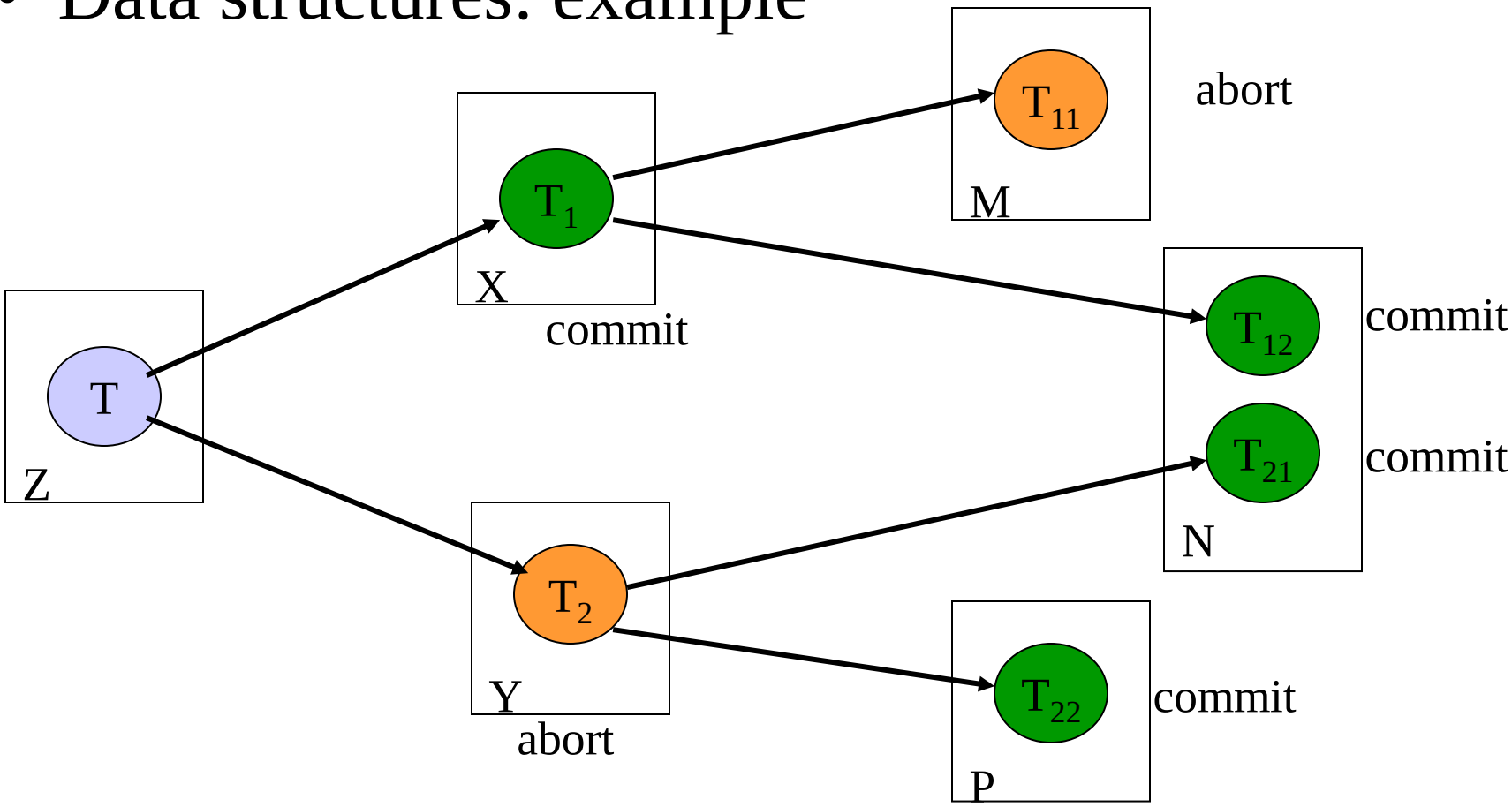
Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 ,T_2	T_{12} , T_1	T_{11}
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y	T_2	T_{21} ,T_{22}	T_{21} ,T_{22}	
M	T_{11}			T_{11}
N	T_{12} ,T_{21}		T_{12} ,T_{21}	
P	T_{22}		T_{22}	

Atomic Commit protocol

- Data structures: example



Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 ,T_2	T_{12} , T_1	T_{11}
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y	T_2	T_{21} ,T_{22}	~~T_{21} ,T_{22}~~	T_2
M	T_{11}			T_{11}
N	T_{12} ,T_{21}		T_{12} ,T_{21}	
P	T_{22}		T_{22}	

Atomic Commit protocol

- Data structures: example

Server	Trans	Child Trans	Commit List	Abort List
Z	T	T_1 ,T_2	T_{12} , T_1	T_{11} ,T_2
X	T_1	T_{11} ,T_{12}	T_{12} , T_1	T_{11}
Y	T_2	T_{21} ,T_{22}	~~T_{21} ,T_{22}~~	T_2
M	T_{11}			T_{11}
N	T_{12} ,T_{21}		T_{12} ,T_{21}	
P	T_{22}		T_{22}	

Atomic Commit protocol

- Data structures: final data

Server	Trans	Child Trans	Commit List	Abort List
Z	T		~~T_{12}~~ , ~~T_1~~ N, X	T_{11} , T_2
X	T_1		T_{12} , T_1	T_{11}
Y M				
N	T_{12} , T_{21}		T_{12} , T_{21}	
P	T_{22}		T_{22}	

Atomic Commit protocol

- Algorithm of coordinator (flat protocol)
  - Phase 1
    - send CanCommit to each worker in commit list
      - TransactionId: T
      - abort list
    - coordinator behaves as worker
  - Phase 2 (as for non-nested transactions)
    - all votes Yes:

      ➔commit transaction; send DoCommit to workers
    - one vote No:

      ➔abort transaction

Atomic Commit protocol

- Algorithm of worker (flat protocol)
  - Phase 1 (after receipt of CanCommit)
    - at least one (provisionally) committed descendant of top level transaction:
      - transactions with ancestors in abort list are aborted
      - prepare for commit of other transactions
      - send Yes to coordinator
    - no (provisionally) committed descendant
      - send No to coordinator
  - Phase 2 (as for non-nested transactions)

Atomic Commit protocol

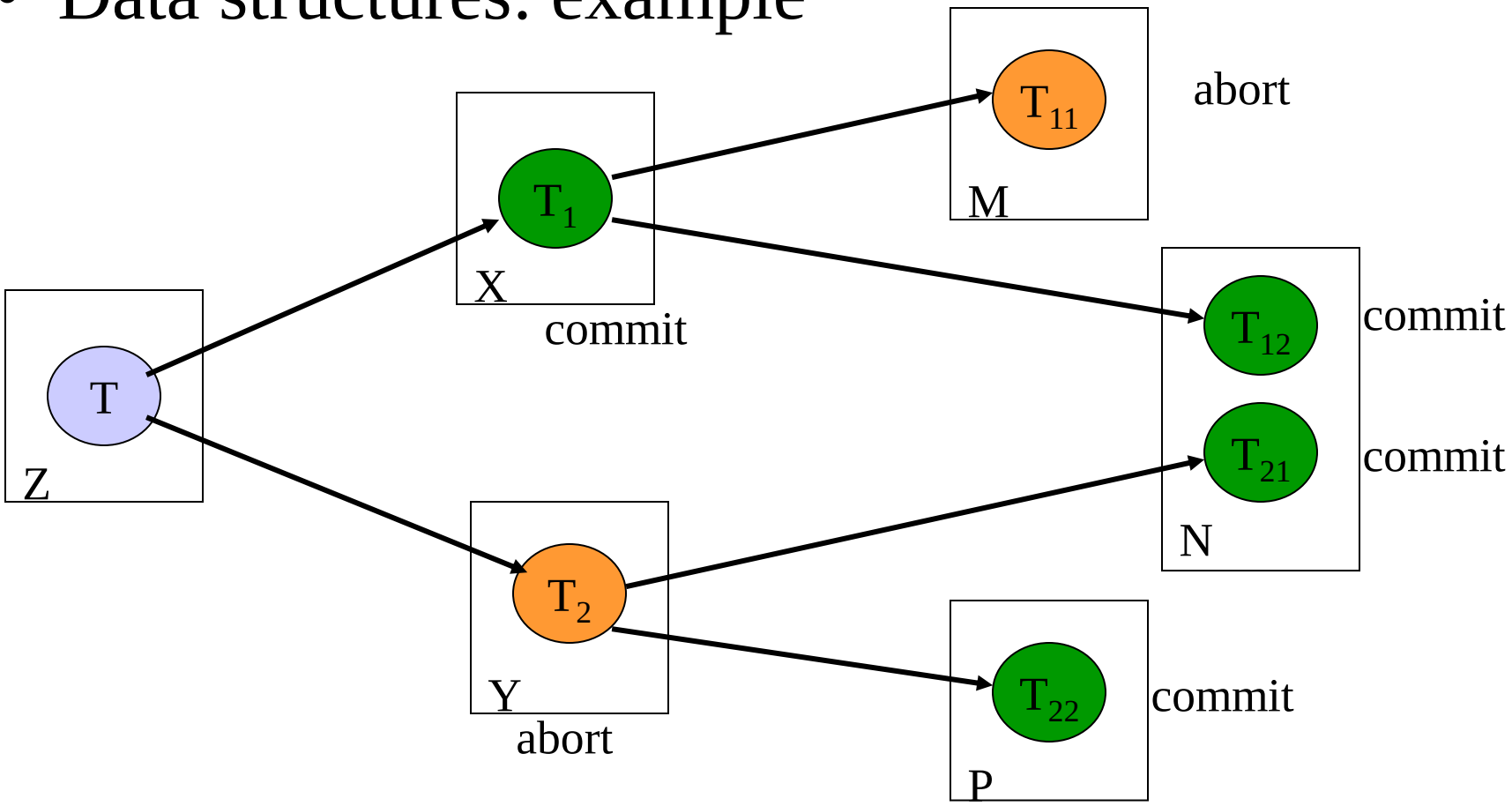
- Algorithm of worker (flat protocol)
  - Phase 1 (after receipt of CanCommit)
  - Phase 2 voted yes, waits for decision of coordinator
    - receives DoCommit
      - makes committed data available; removes locks
    - receives AbortTransaction
      - clears data structures; removes locks

Atomic Commit protocol

- Timeouts:
  - same 3 as above:
    - worker did all/some operations and waits for CanCommit
    - coordinator waits for votes of workers
    - worker voted Yes and waits for final decision of coordinator
  - provisionally committed child with an aborted ancestor:
    - does not participate in algorithm
    - has to make an enquiry itself
    - when?

Atomic Commit protocol

- Data structures: example



Overview

- Transactions

- Distributed transactions
  - Flat and nested distributed transactions
  - Atomic commit protocols
  - Concurrency in distributed transactions … (Part 3)
  - recoveryDistributed deadlocks
  - Transaction

KATHOLIEKE UNIVERSITEIT LEUVEN

Distributed Systems: Transactions Part 2 – Questions?

14 November 2023