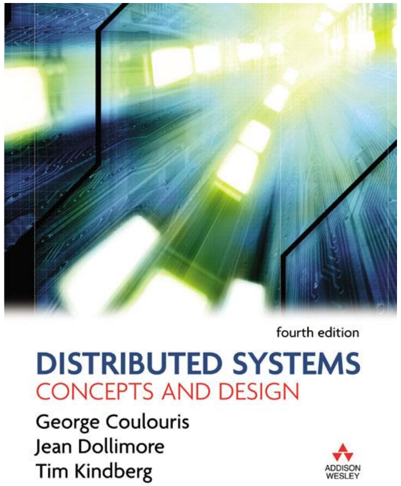


KATHOLIEKE UNIVERSITEIT LEUVEN

Distributed Systems:

Transactions – Part 3

Overview

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

Distributed transactions
Locking

- Locks are maintained locally (at each server)
  - it decides whether
    - to grant a lock
    - to make the requesting transaction wait
  - it cannot release the lock until it knows whether the transaction has been
    - committed
    - aborted

    at all servers
  - deadlocks can occur

Distributed transactions
Locking

- Locking rules for nested transactions
  - child transaction inherits locks from parents
  - when a nested transaction commits, its locks are inherited by its parents
  - when a nested transaction aborts, its locks are removed
  - a nested transaction can get a read lock when all the holders of write locks (on that data item) are ancestors
  - a nested transaction can get a write lock when all the holders of read and write locks (on that data item) are ancestors

Overview

- Transactions

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

- Replication

Distributed deadlocks

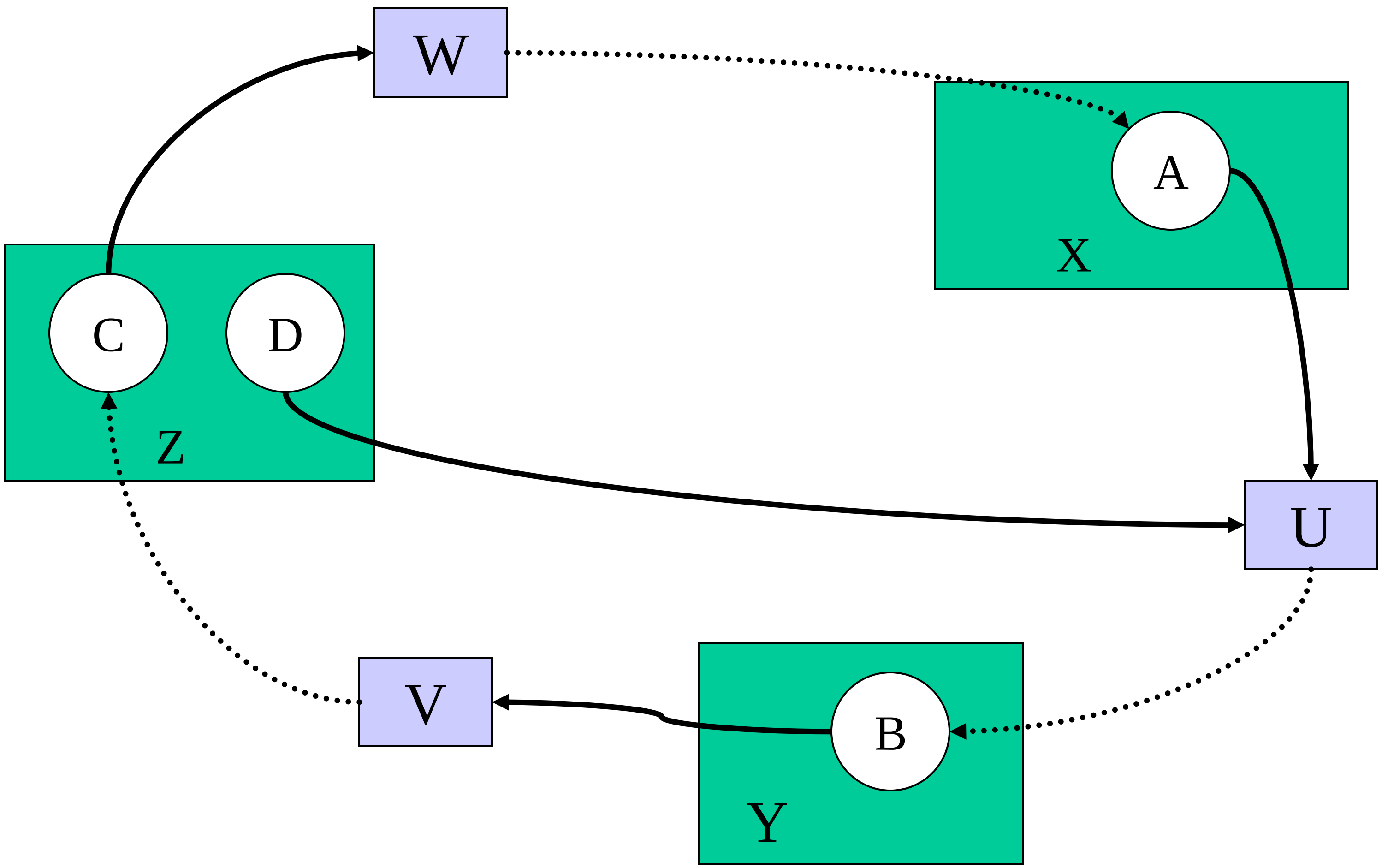
- Single server approaches
  - prevention: difficult to apply
  - timeouts: value with variable delays?

➔Detection

- global wait-for-graph can be constructed from local ones
- cycle in global graph possible without cycle in local graph

Distributed transactions

Deadlocks



Distributed transactions
Deadlocks

- Algorithms:
  - centralised deadlock detection: not a good idea
    - depends on a single server
    - cost of transmission of local wait-for graphs
  - distributed algorithm:
    - Relatively Complex
    - In this course: edge chasing approach

Distributed transactions
Deadlocks

- Phantom deadlocks
  - deadlock detected that is not really a deadlock
  - during deadlock detection
    - while constructing global wait-for graph
    - waiting transaction is aborted

Distributed transactions
Deadlocks

- Edge Chasing
  - distributed approach to deadlock detection:
    - no global wait-for graph is constructed
  - servers attempt to find cycles
    - by forwarding probes (= messages) that follow edges of the wait-for graph throughout the distributed system

Distributed transactions
Deadlocks

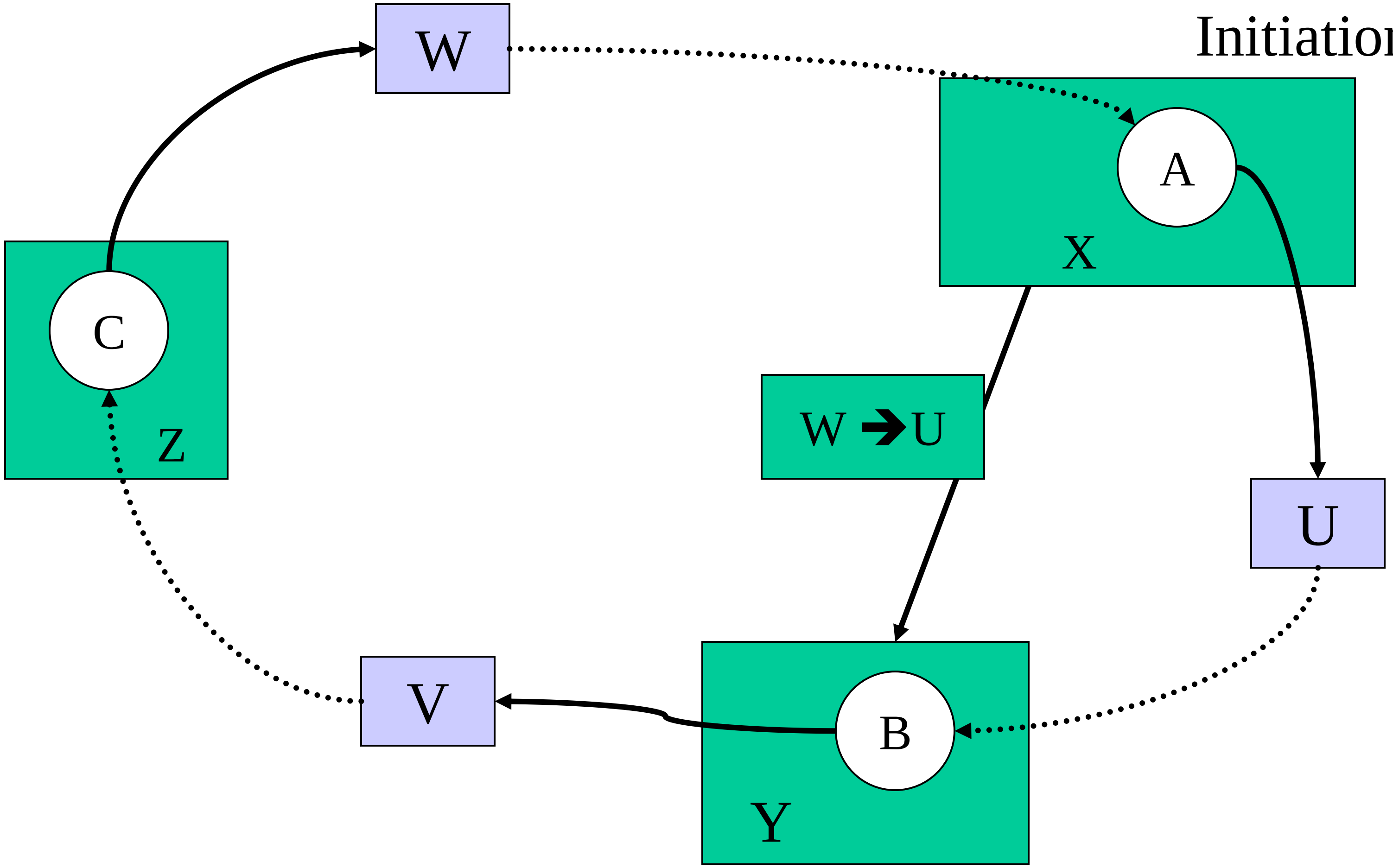
- Edge Chasing
  - three steps:
    - initiation: transaction starts waiting
      - new probe constructed
    - detection: probe received
      - extend probe
      - check for loop
      - forward new probe
    - resolution

Distributed transactions
Deadlocks

- Edge Chasing: initiation
  - send out probe T ➔ U

    when transaction T starts waiting for U (and U is already waiting for …)
  - in case of lock sharing, different probes are forwarded

Distributed transactions
Deadlocks



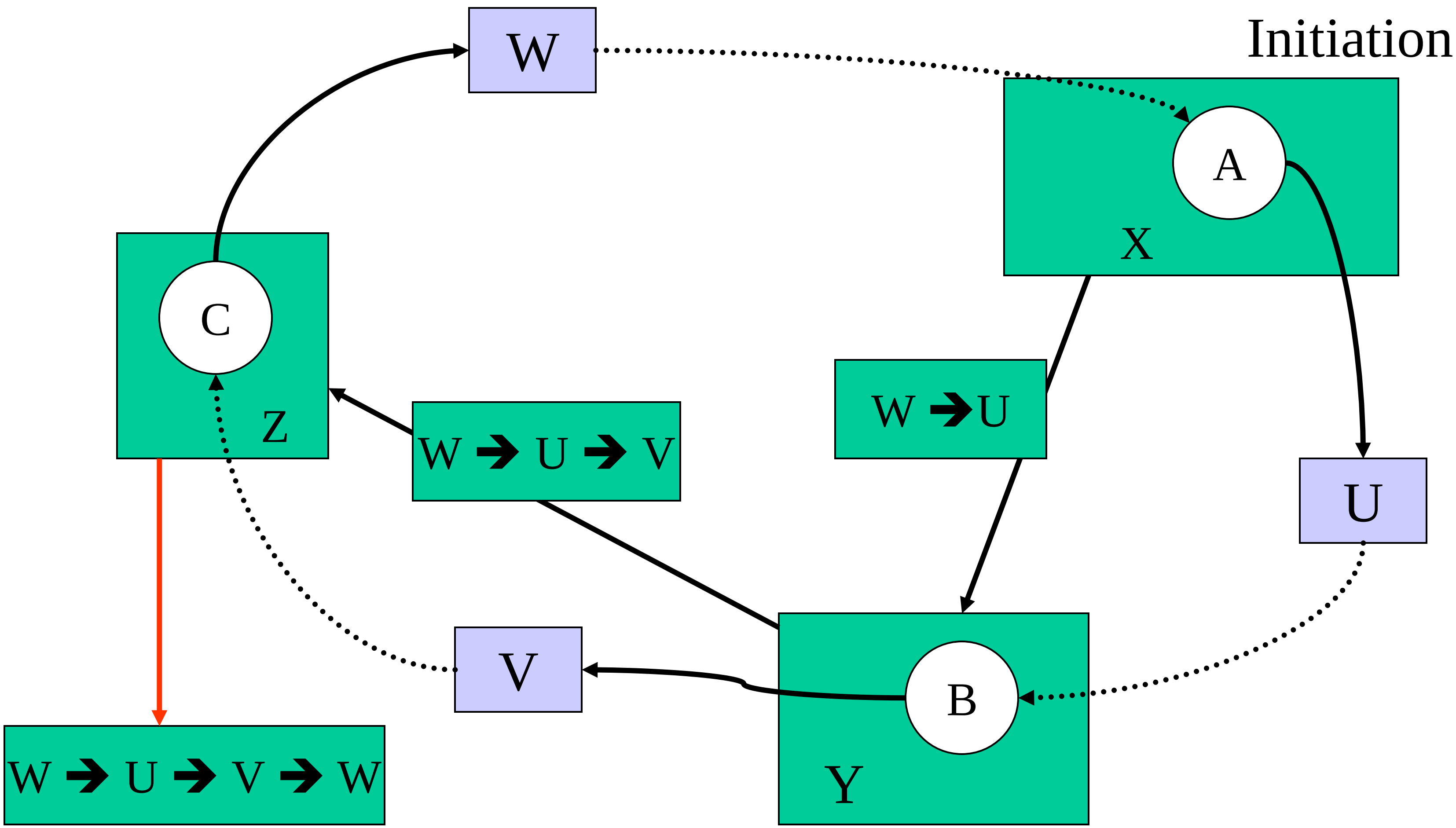
Distributed transactions

Deadlocks

- Edge Chasing: detection
  - when receiving probe T ➔ U
    - Check if U is waiting
    - if U is waiting for V (and V is waiting) add V to probe T ➔ U ➔ V
    - check for loop in probe?
      - yes ➔ deadlock
      - no ➔ forward new probe

Distributed transactions

Deadlocks



Distributed transactions

Deadlocks

- Edge Chasing: resolution
  - abort one transaction
  - problem?
    - Every waiting transaction can initiate deadlock detection
    - detection may happen at different servers
    - several transactions may be aborted
  - solution: transactions priorities

Distributed transactions
Deadlocks

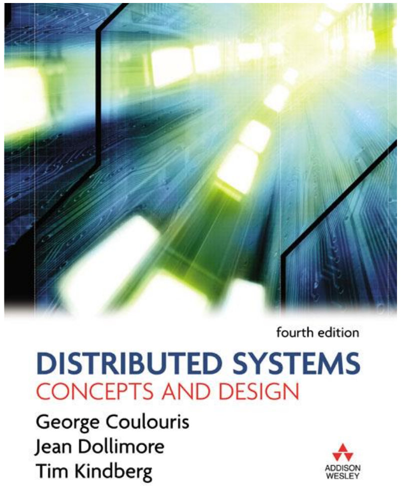
- Edge Chasing: transaction priorities
  - assign priority to each transaction, e.g. using timestamps
  - solution of problem above:
    - abort transaction with lowest priority
    - if different servers detect same cycle, the same transaction will be aborted

Distributed transactions
Deadlocks

- Edge Chasing: transaction priorities
  - other improvements
    - number of initiated probe messages ➘
      - detection only initiated when higher priority transaction waits for a lower priority one
    - number of forwarded probe messages ➘
      - probes travel downhill -from transaction with high priority to transactions with lower priorities
      - probe queues required; more complex algorithm

Overview

- Transactions
- Distributed transactions
  - Flat and nested distributed transactions
  - Atomic commit protocols
  - Concurrency in distributed transactions
  - Distributed deadlocks
  - Transaction recovery (skipped 2023-2024)



KATHOLIEKE UNIVERSITEIT LEUVEN

Distributed Systems:

Transactions – Part 3

19/12/2023