

Κατανεμημένες δοσοληψίες

Κατανεμημένα Συστήματα
2018-2019

<http://www.cslab.ece.ntua.gr/courses/distrib>

Στο προηγούμενο μάθημα...

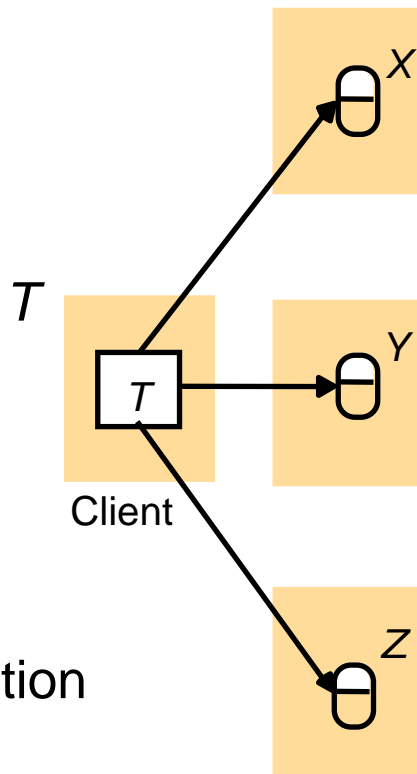
- Απλές δοσοληψίες (transactions)
- Ιδιότητες ACID
 - Και κυρίως atomicity και durability
- Σειριοποιησιμότητα (serializability)
- Έλεγχος ταυτοχρονισμού
 - Κλειδώματα (Locking)
 - Διάταξη χρονοσφραγίδων (Timestamp ordering)
 - Αισιόδοξος έλεγχος ταυτοχρονισμού (optimistic concurrency control)
- Αδιέξοδα (deadlocks)

Σε αυτό το μάθημα

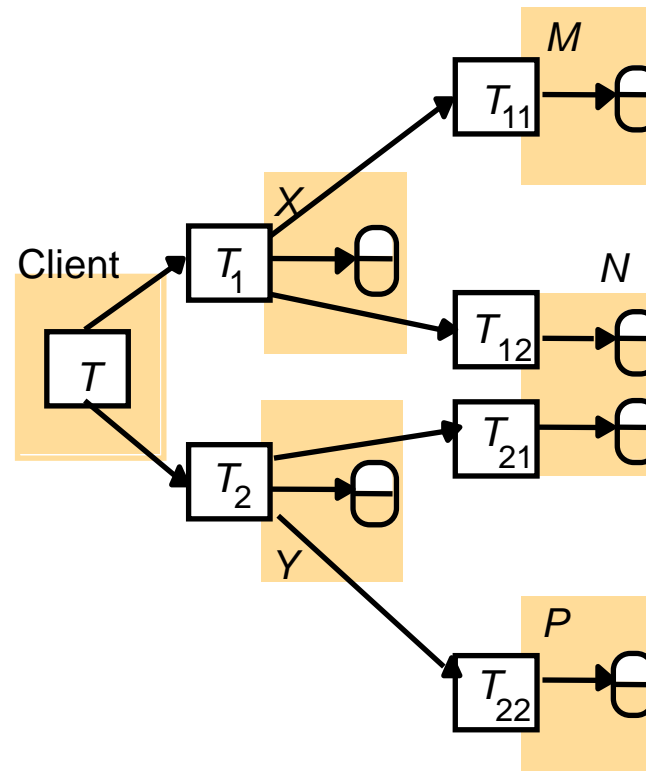
- Distributed transactions
 - Flat
- Atomicity σε distributed transactions
 - Ύπαρξη coordinator
 - Πρωτόκολλο two-phase commit
- Concurrency control σε distributed transactions
 - Τοπικά σε κάθε server
 - Καθολικά στο σύνολο των servers
- Distributed Deadlocks

Distributed Transactions

- Transactions που προσπελάζουν αντικείμενα διαχειριζόμενα από πολλούς servers



(a) Flat transaction



(b) Nested transactions

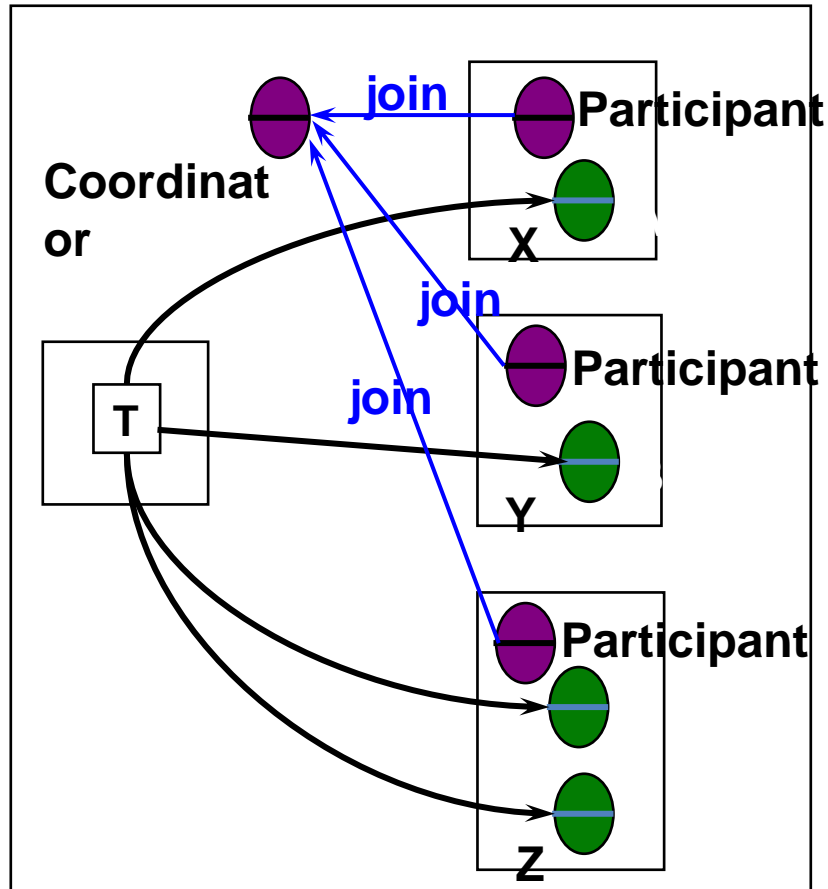
Κάθε distributed transaction...

- Έχει έναν server που αναλαμβάνει ρόλο coordinator, ενώ οι υπόλοιποι servers που συμμετέχουν στο transaction είναι οι participants
- Γιατί;
 - Χρειάζεται συνεννόηση μεταξύ των servers για να αποφασίσουν από κοινού commit ή abort -> atomicity σε distributed transactions

Coordinator και Participant

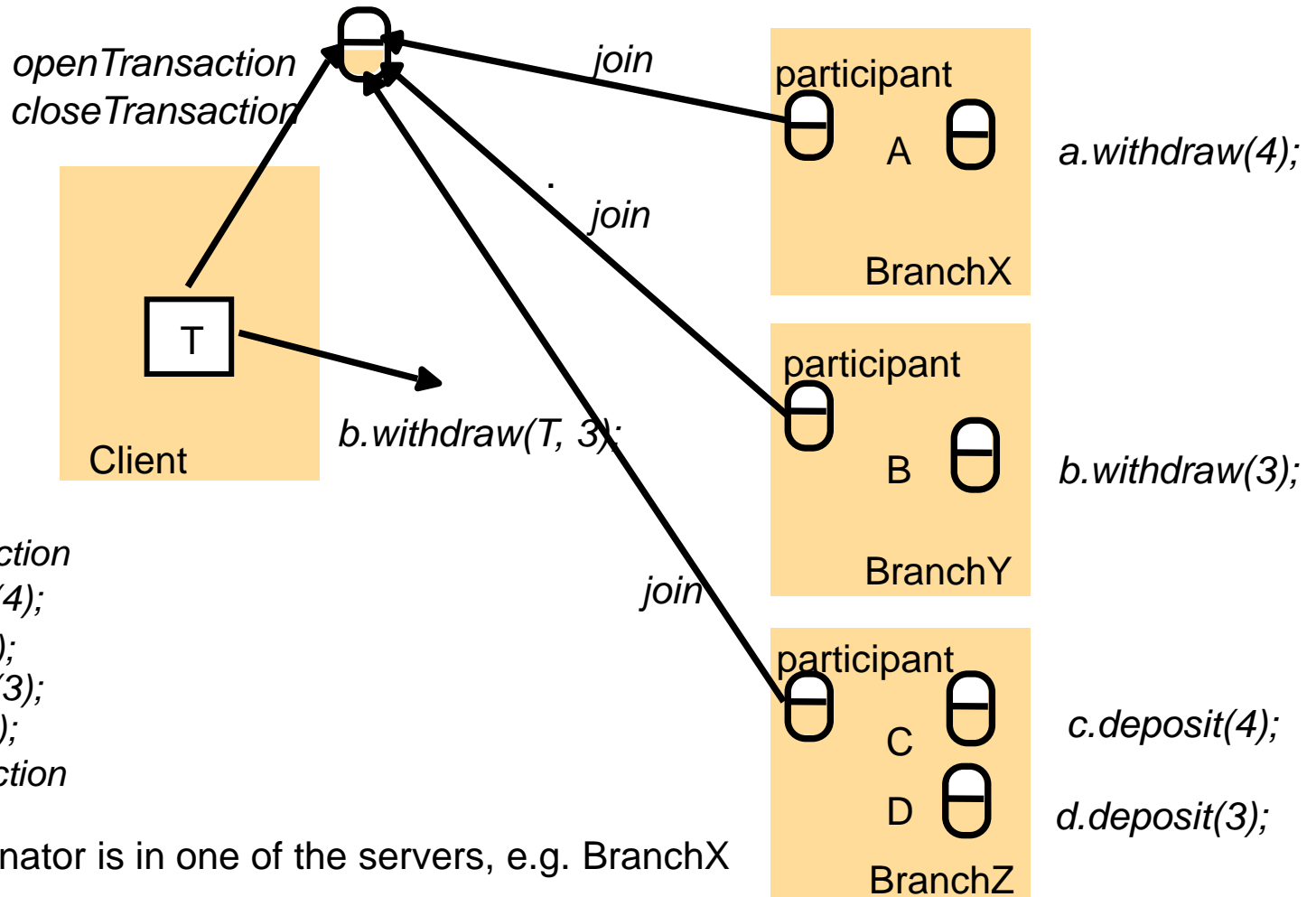
- Ο coordinator
 - Εκτελεί το openTransaction όταν καλείται από τον client επιστρέφοντας μοναδικό αναγνωριστικό (TID)
 - IP του server + αύξοντα αριθμό
 - Είναι υπεύθυνος για το commit ή το abort
 - Καταγράφει τους servers που συμμετέχουν στο distributed transaction (participants)
- Ο participants
 - Servers που εκτελούν τοπική επεξεργασία σε αντικείμενα που έχουν
 - Υπεύθυνοι για τα αντικείμενά τους
 - Συνεργάζονται με coordinator ακολουθώντας το πρωτόκολλο

Σχηματικά



Coordinator & Participants

Παράδειγμα



T = openTransaction
a.withdraw(4);
c.deposit(4);
b.withdraw(3);
d.deposit(3);
closeTransaction

Note: the coordinator is in one of the servers, e.g. BranchX

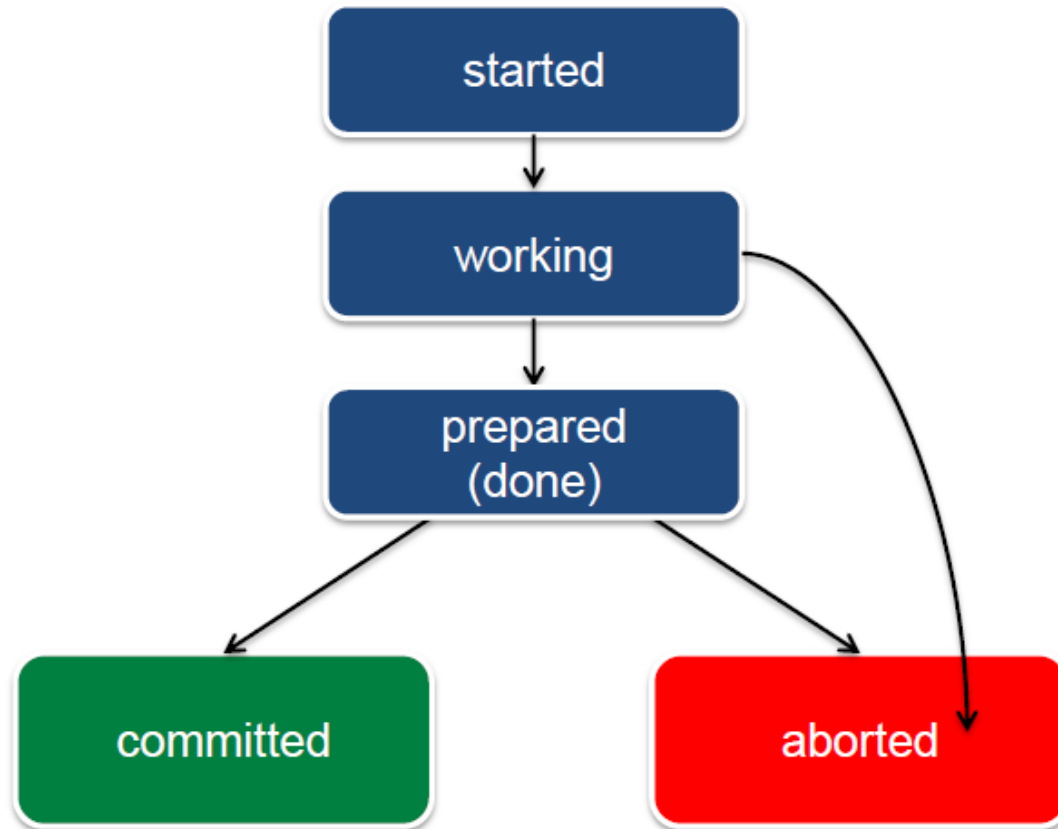
1-phase commit (1PC)

- Ο coordinator στέλνει σε όλους τους participants commit ή abort
- Περιμένει ack από όλους
- Ξαναστέλνει αν χρειαστεί
- Τι μπορεί να πάει στραβά;
 - Δε δουλεύει αν ένας participant πεθάνει πριν λάβει το μήνυμα
 - Δεν επιτρέπει σε κάποιον participant να αποφασίσει να κάνει abort το transaction (π.χ., σε περίπτωση deadlock, αποτυχία validation σε optimistic concurrency control, ανάνηψη από crash κλπ)

2-phase commit (2PC)

- Πρώτη φάση
 - Ο Coordinator συγκεντρώνει τις ψήφους των participants (commit or abort)
 - Για να ψηφίσει commit ένας participant πρέπει να εξασφαλίσει ότι θα μπορεί να το εκτελέσει (ακόμα και μετά από crash)
 - Αποθήκευση σε persistent storage πριν ψηφίσει
 - Μπαίνει σε “prepared state”
- Δεύτερη φάση
 - Αν όλοι οι participants έχουν ψηφίσει commit, ο coordinator στέλνει μήνυμα multicast για commit
 - Αν έστω κι ένας participant ψηφίσει abort ή πέθανε, ο coordinator στέλνει μήνυμα multicast για abort

Transaction states

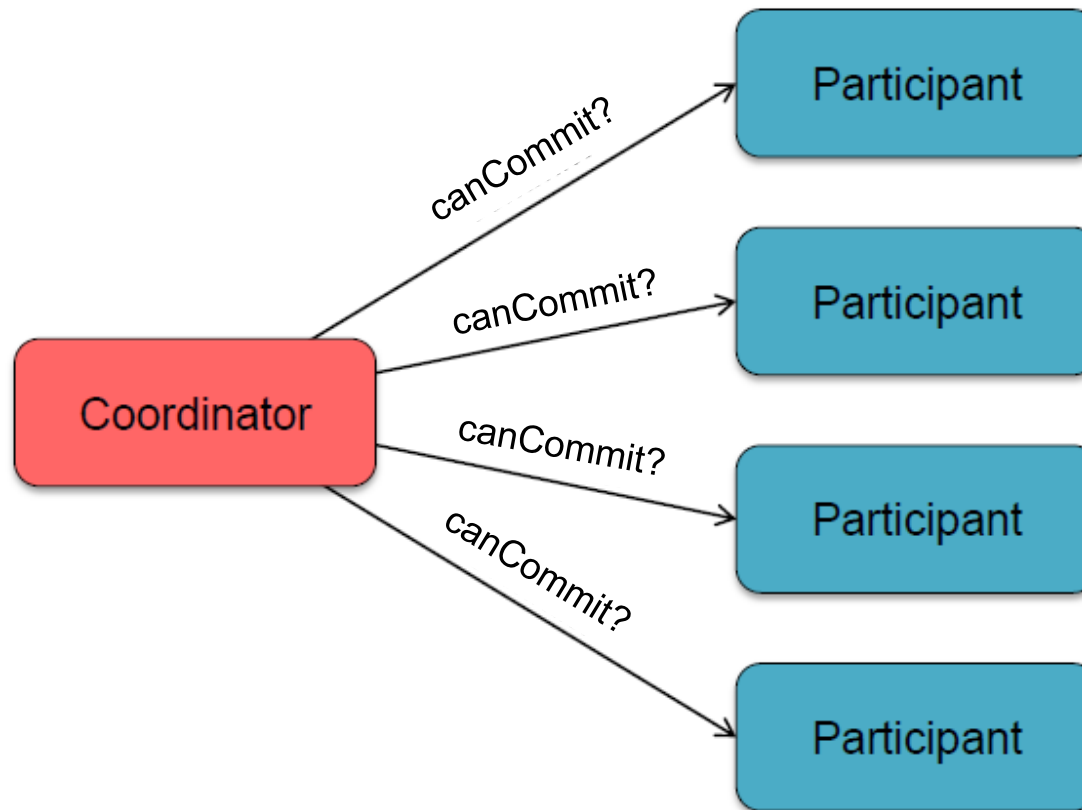


Λειτουργίες του 2PC

- `canCommit?(trans)` -> Yes / No
 - `doCommit(trans)`
 - `doAbort(trans)`
 - `haveCommitted(trans, participant)`
 - `getDecision(trans)` -> Yes / No
 - Call from participant to coordinator to ask for the decision on a transaction after it has voted *Yes* but has still had no reply after some delay. Used to recover from server crash or delayed messages.
- Participant interface
- Coordinator interface

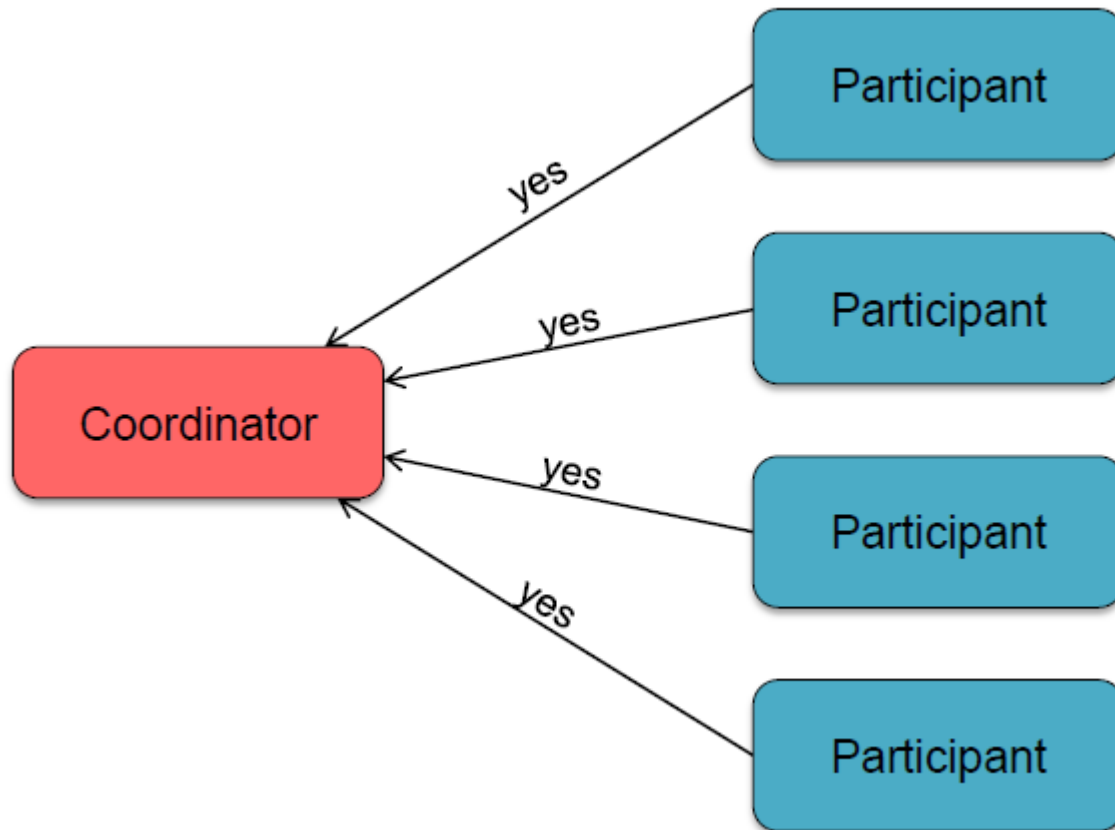
Το πρωτόκολλο

- 1^η φάση: Voting phase



Το πρωτόκολλο

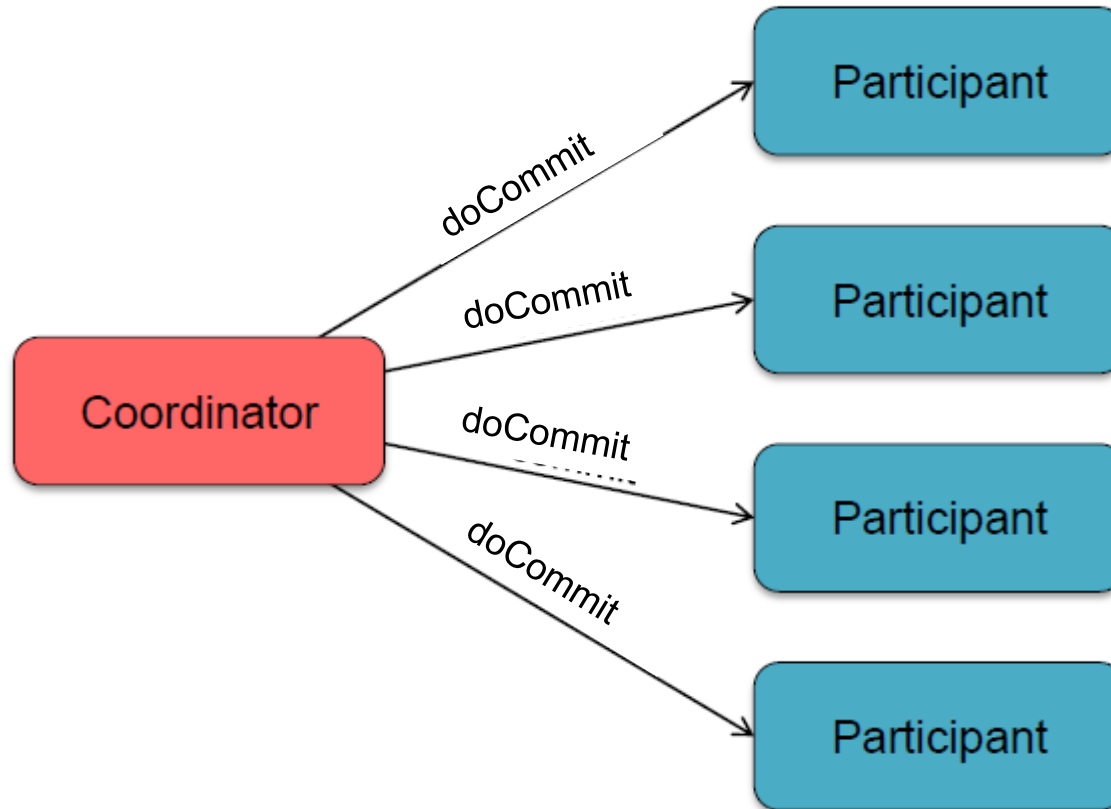
- 1^η φάση: Voting phase



Έστω κι ένα όχι σημαίνει abort

Το πρωτόκολλο

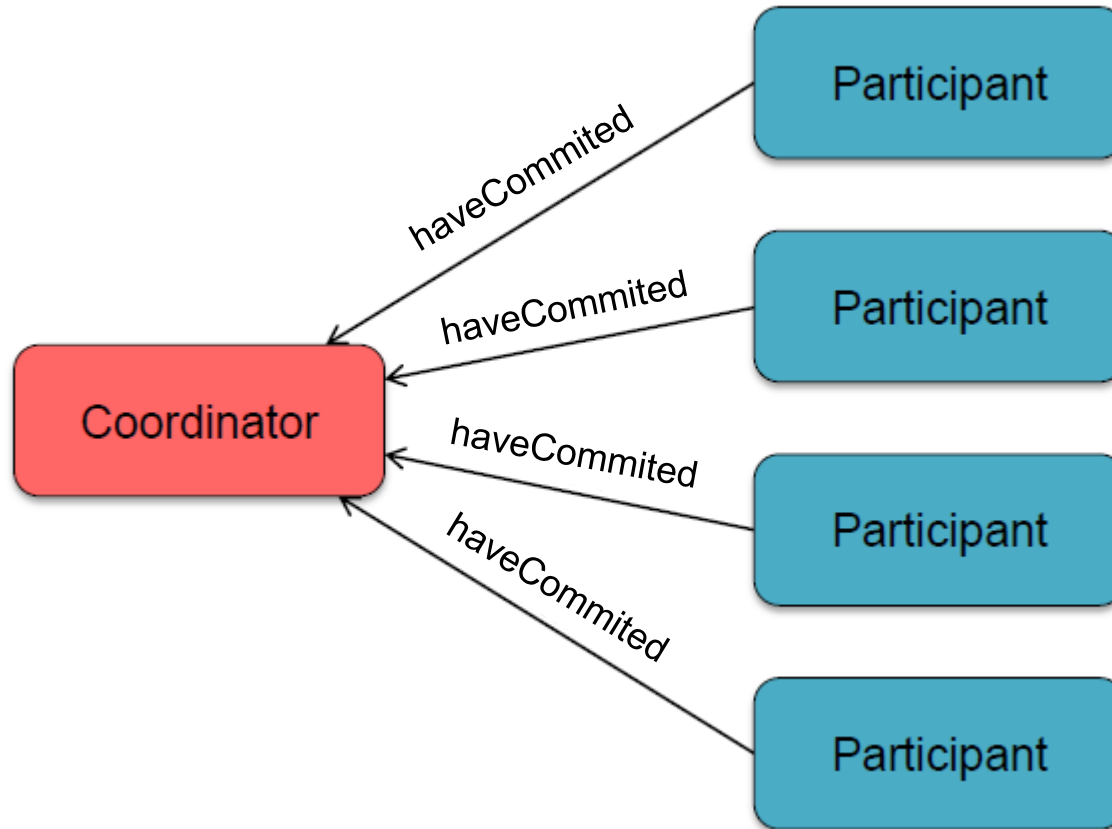
- 2^η φάση: Commit phase



Αποστολή abort αν έστω κι ένας ψήφισε όχι

Το πρωτόκολλο

- 2^η φάση: commit phase: ack από όλους

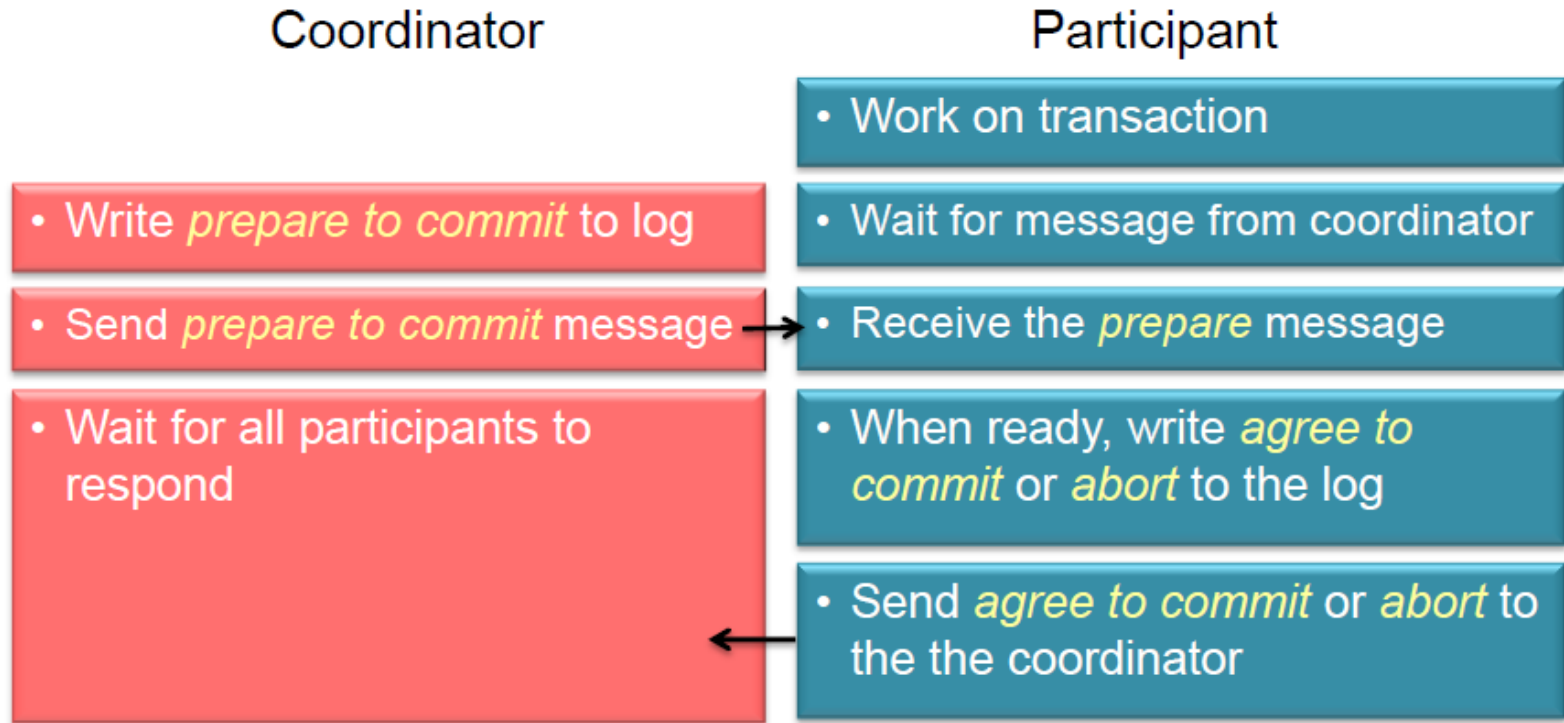


Χειρισμός σφαλμάτων

- Το πρωτόκολλο υποθέτει μοντέλο *fail-recover*
 - Όποιος κόμβος αποτύχει κάποια στιγμή θα ανανήψει
- Μετά από ανάνηψη ο κόμβος δε μπορεί να αλλάξει απόφαση
 - Αν ένας κόμβος συμφώνησε σε commit και μετά πέθανε, θα πρέπει μετά την ανάνηψη να εκτελέσει το commit
- Κάθε κόμβος χρησιμοποιεί ένα write-ahead (transaction) log
 - Κρατά το σημείο στο οποίο έχει φτάσει στο πρωτόκολλο (και σε τι έχει συμφωνήσει)
 - Κρατά τις τιμές των αντικειμένων ώστε να κάνει commit ή abort

Χρήση log: 1^η φάση

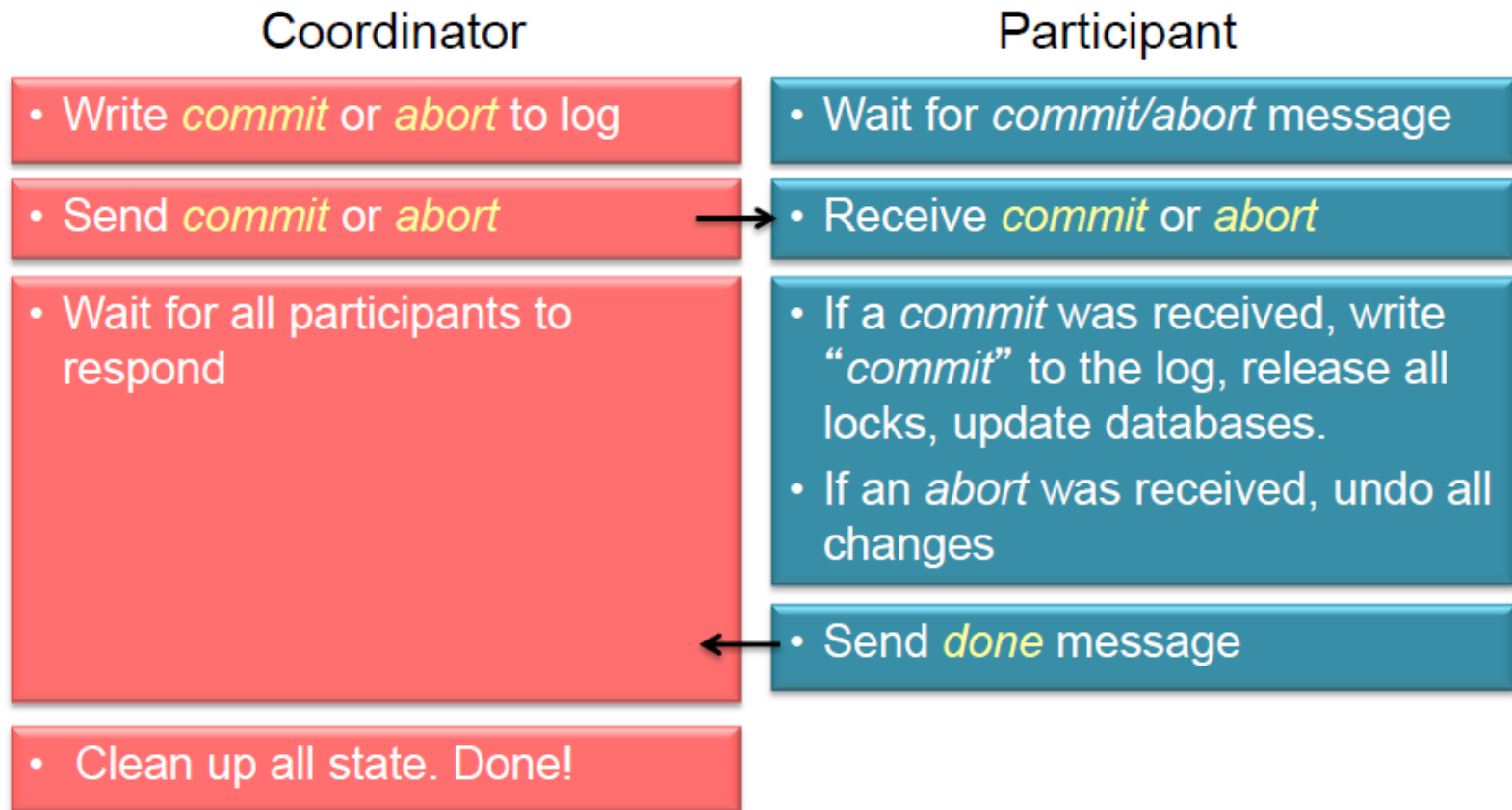
Voting Phase



Ερχόμαστε σε κατανεμημένη συμφωνία: ο coordinator ρωτά κάθε participant αν θα κάνει commit ή abort και λαμβάνει απαντήσεις

Χρήση log: 2^η φάση

Commit Phase



Ανακοινώνει σε όλους τους participants αν θα κάνουν *commit* ή *abort* και λαμβάνει απαντήσεις ότι το έκαναν

Σφάλματα στην 1^η φάση

- Πεθαίνει ο coordinator
 - Κάποιοι participants ίσως έχουν απαντήσει, οι υπόλοιποι δεν έχουν ιδέα
 - ⇒ ο coordinator ξεκινά ξανά, ελέγχει το log, βλέπει ότι ήταν σε εξέλιξη η φάση voting
 - ⇒ ο coordinator επανεκκινεί τη φάση voting
- Πεθαίνει ένας participant
 - Πριν ή μετά την αποστολή της ψήφου στον coordinator
 - ⇒ Αν ο coordinator έλαβε ψήφο, περιμένει τους υπόλοιπους και μπαίνει στη 2^η φάση
 - ⇒ Αλλιώς: Περιμένει τον participant να ανανήψει και να απαντήσει (εξακολουθεί να τον ρωτάει)

Σφάλματα στη 2^η φάση

- Πεθαίνει ο coordinator
 - Μπορεί να έχει σταλεί commit/abort σε κάποιους participants
 - ⇒ Ο coordinator ξεκινά, ελέγχει το log, ενημερώνει όλους για commit ή abort
- Πεθαίνει ένας participant
 - Πριν ή μετά τη λήψη του commit/abort από τον coordinator
 - ⇒ ο coordinator εξακολουθεί να στέλνει την απόφαση στον participant
 - ⇒ ο participant ξεκινά, ελέγχει το log, παίρνει την commit/abort από τον coordinator
 - Αν έχει κάνει ήδη commit ή abort στέλνει μόνο ack
 - Αλλιώς επεξεργάζεται πρώτα το commit/abort και στέλνει ack

Καθυστερήσεις (1)

- Ο participant έχει ψηφίσει και περιμένει doCommit ή doAbort από τον coordinator (uncertain state)
 - getDecision στον coordinator
 - Δεν μπορεί να πάρει μόνος του απόφαση (ούτε να αποδεσμεύσει τα locks του)
 - Περιμένει μέχρι να ανανήψει ή αντικατασταθεί ο coordinator
 - Εναλλακτικά: Μπορεί να ενημερωθεί από τους υπόλοιπους participants
 - Το πρόβλημα παραμένει αν όλοι οι participants βρίσκονται σε uncertain state

Καθυστερήσεις (2)

- Ο participant περιμένει canCommit (δεν έχει αίτημα για transaction για πολλή ώρα)
 - Timeouts για τα locks
 - Μπορεί να αποφασίσει abort μόνος του, αφού δεν έχει ψηφίσει ακόμα
- Ο coordinator περιμένει ψήφους
 - Μπορεί να αποφασίσει abort μετά από κάποια ώρα

Απόδοση του 2PC

- N canCommit μηνύματα
- N απαντήσεις
- N doCommit ή doAbort μηνύματα

Άρα $3N$

Δεν προσμετράμε τα haveCommitted

Recovery coordinator

- Κάποιος άλλος κόμβος μπορεί να αναλάβει το ρόλο του coordinator
 - Πιθανόν ένας participant που ανιχνεύει timeout από τον coordinator
- Ο αντικαταστάτης πρέπει να μάθει το state του πρωτοκόλλου
 - Επικοινωνεί με όλους τους participants για να μάθει τι ψήφισαν
 - Αν λάβει αποτελέσματα από όλους τους participants
 - Ξέρει ότι έχει ολοκληρωθεί η 1^η φάση
 - Αν όλοι ψήφισαν commit \Rightarrow στέλνει doCommit
 - Αλλιώς στέλνει abort
 - Αν έστω κι ένας participant δεν έχει ψηφίσει
 - Ξέρει ότι δεν έχει ολοκληρωθεί η 1^η φάση
 - \Rightarrow Ξεκινά το πρωτόκολλο από την αρχή
- Αλλά... αν ένας participant πεθάνει, τότε κολλήσαμε!
 - Αναμονή για ανάνηψη

Τι κακό έχει το 2-phase commit;

- Το μεγαλύτερο πρόβλημα: **είναι blocking πρωτόκολλο**
 - Αν ο coordinator πεθάνει, οι participants δεν ξέρουν αν πρέπει να κάνουν commit ή abort (uncertain state) και κρατάνε τα εμπλεκόμενα αντικείμενα κλειδωμένα
 - Ο recovery coordinator βοηθά σε κάποιες περιπτώσεις
 - Ένας participant που δεν αποκρίνεται επίσης μπλοκάρει τη ροή του πρωτοκόλλου
- Όταν ένας participant λάβει commit/abort, δε γνωρίζει αν όλοι οι άλλοι participants έχουν ενημερωθεί για το αποτέλεσμα

3-phase commit

- Ίδιο setup όπως στο two-phase commit:
 - Coordinator & Participants
- Διευκολύνει τη χρήση recovery coordinator
 - Στέλνει το αποτέλεσμα της ψηφοφορίας για commit/abort σε όλους τους participants πριν τους στείλει απόφαση
 - Επιτρέπει την ανάνηψη της κατάστασης αν οποιοσδήποτε participant πεθάνει
- Προσθέτει timeouts σε κάθε φάση που οδηγούν σε abort

Το πρωτόκολλο

- Χωρίζει τη 2^η φάση του 2PC σε δύο μέρη:
 - 2a. “Precommit” (ετοιμασία για commit)
 - Στέλνει μήνυμα Prepare σε όλους τους participants όταν λάβει ναι από όλους στην πρώτη φάση
 - Οι Participants μπορούν να προετοιμάσουν το commit αλλά όχι να κάνουν κάτι που δε μπορεί να ακυρωθεί
 - Οι Participants απαντούν με ack
 - Σκοπός: Κάθε participant ξέρει το αποτέλεσμα της ψηφοφορίας ώστε η κατάσταση να μπορεί να επανέλθει αν πεθάνει οποιοσδήποτε
 - 2b. “Commit” (όπως στο 2PC)
 - Αν ο coordinator λάβει ACKs από όλους
 - Στέλνει μήνυμα commit σε όλους τους participants
 - Αλλιώς θα στείλει abort

1^η φάση

Voting phase

- Ο coordinator στέλνει canCommit? στους participants & λαμβάνει απαντήσεις
- Σκοπός: Να μάθει αν όλοι συμφωνούν για commit
- [!] Αν ο coordinator δε λάβει απάντηση από κάποιον participant (timeout) ή λάβει αρνητική απάντηση
 - Στέλνει abort σε όλους
- [!] Αν κάποιος participant κάνει timeout περιμένοντας για αίτημα από τον coordinator
 - Κάνει abort τον εαυτό του (υποθέτει ότι ο coordinator πέθανε)
- Αλλιώς συνεχίζει στη 2^η φάση

2^η φάση

Precommit (or Prepare to commit) phase

- Στέλνει μήνυμα prepare σε όλους τους participants.
 - Λαμβάνει OK από αυτούς
- Σκοπός: γνωστοποιεί σε όλους τους participants την απόφαση για commit
- [!] Αν ο coordinator κάνει timeout: υποθέτει ότι ο participant πέθανε, στέλνει abort σε όλους

3^η φάση

Commit phase

- Στέλνει μήνυμα commit σε όλους τους participants και λαμβάνει απαντήσεις
- [!] Αν ένας participant κάνει timeout: επικοινωνεί με οποιονδήποτε άλλον participant και υιοθετεί το τρέχον state (commit ή abort)
- [!] Αν ο coordinator κάνει timeout περιμένοντας: δεν πειράζει

Ανάνηψη

- Αν πεθάνει ο coordinator
 - Ένας recovery coordinator μπορεί να μάθει την κατάσταση από οποιοδήποτε ζωντανό κόμβο
- Πιθανές καταστάσεις ενός participant:
 - Committed
 - Σημαίνει ότι όλοι οι άλλοι participants έχουν λάβει μήνυμα Prepare to Commit
 - Κάποιοι έχουν κάνει commit
 - ⇒ Στέλνει Commit σε όλους τους participants
 - Έχει ληφθεί Prepare to Commit
 - Σημαίνει ότι όλοι οι άλλοι participants έχουν συμφωνήσει σε commit; Κάποιοι μπορεί να έχουν κάνει commit
 - Στέλνει Prepare to Commit σε όλους
 - Περιμένει για acks και μετά προχωράει σε commit
 - Δεν έχει λάβει ακόμα Prepare
 - Σημαίνει ότι κανένας δεν έχει κάνει commit, κάποιοι μπορεί να έχουν συμφωνήσει
 - Το transaction μπορεί να γίνει abort ή να ξεκινήσει ξανά το πρωτόκολλο

Αδυναμίες του πρωτοκόλλου

- Μπορεί να δημιουργηθούν προβλήματα σε network partitioning
 - Partition A: οι κόμβοι έχουν λάβει μήνυμα Prepare
 - Ο recovery coordinator για το A αποφασίζει commit
 - Partition B: οι κόμβοι δεν έχουν λάβει μήνυμα Prepare
 - Ο recovery coordinator για το B αποφασίζει abort
 - Και οι δύο πράξεις είναι σωστές
 - Όμως όταν επανέλθει το δίκτυο, το σύστημα δε θα είναι συνεπές
- Πρόβλημα όταν ο coordinator επανέρχεται από crash
 - Πρέπει να ανακαλύψει ότι κάποιος άλλος έχει αναλάβει τον ρόλο του
 - Αλλιώς οι πράξεις του οδηγούν σε μη συνεπή κατάσταση

Πρόβλημα

Υποθέτουμε ότι:

- ο coordinator στέλνει μήνυμα Prepare σε όλους
 - Όλοι στέλνουν ack
 - Όμως ο coordinator πέθανε πριν λάβει όλα τα acks
- Ο recovery coordinator επικοινωνεί με κάποιον participant
 - Συνεχίζει με το commit: Στέλνει Prepare, παίρνει ACKs, στέλνει Commit
- Ταυτόχρονα επανέρχεται ο αρχικός coordinator
 - Αντιλαμβάνεται ότι του λείπουν κάποιες απαντήσεις στο Prepare
 - Κάνει timeout να στείλει abort σε όλους
- Μπορεί κάποιοι κόμβοι να κάνουν commit ενώ άλλοι abort
- Ο 3PC δουλεύει καλά όταν οι servers πεθαίνουν (fail-stop)
- Όχι όμως όταν επανέρχονται μετά από αποτυχία (fail-recover)

Έλεγχος ταυτοχρονισμού

- Κάθε server είναι υπεύθυνος για έλεγχο ταυτοχρονισμού στα αντικείμενα που διαχειρίζεται
- Οι servers που συνεργάζονται για distributed transactions είναι από κοινού υπεύθυνοι για την εκτέλεσή τους με serially equivalent τρόπο
- Αν ένα transaction T προηγείται του U στην προσπέλαση αντικειμένων ενός server τότε πρέπει να προηγείται και σε όλους τους servers που τα T και U θέλουν να προσπελάσουν κοινά αντικείμενα

Locking

- Ο τοπικός lock manager αποφασίζει αν θα δώσει ένα lock σε ένα transaction ή θα το αφήσει να περιμένει
- Δε μπορεί να απελευθερώσει lock αν δε γνωρίζει ότι το transaction έκανε commit ή abort σε όλους τους εμπλεκόμενου servers
 - Αντικείμενα παραμένουν κλειδωμένα και μη διαθέσιμα για άλλα transactions
- Ανεξάρτητα locks σε κάθε server -> διαφορετική διάταξη των transactions
 - Deadlock

Παράδειγμα

<i>T</i>			<i>U</i>		
<i>Write(A)</i>	at X	Lock A			
			<i>Write(B)</i>	at Y	lock B
<i>Read(B)</i>	at Y	Wait for U			
			<i>Read(A)</i>	at X	Wait for T

Timestamp ordering

- Σε έναν server:
 - Ο coordinator δίνει μοναδικό timestamp σε κάθε transaction όταν ξεκινά
 - serial equivalence -> commit τα versions των αντικειμένων με τη σειρά των timestamps των transactions που τα προσπέρασαν
- Σε πολλούς servers:
 - Ο πρώτος coordinator που προσπελάζεται από ένα transaction δίνει καθολικά μοναδικό timestamp
 - Οι servers συμφωνούν στην διάταξη των timestamps `<local_timestamp, server_id>`
 - Η διάταξη θα ισχύει ακόμα κι αν τα ρολόγια δεν είναι συγχρονισμένα, όμως καλύτερα να είναι για λόγους efficiency
 - Τα conflicts λύνονται όπως και σε έναν server

Optimistic concurrency control

- Σε έναν server:
 - Κάθε transaction γίνεται validate πριν γίνει commit
 - Δίνονται timestamps στα transactions στην αρχή της διαδικασίας validation τους
 - Τα transactions σειριοποιούνται με βάση τα timestamps
- Σε πολλούς servers
 - Κάθε server κάνει validate τα transactions που προσπελάζουν τα αντικείμενά του
 - Γίνεται στην 1^η φάση του 2PC πρωτοκόλλου

Commitment deadlock

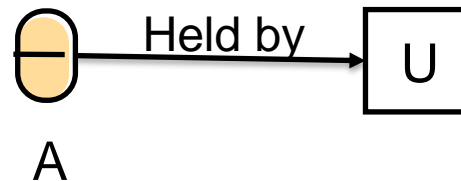
	<i>T</i>		<i>U</i>	
<i>read(A)</i>	at X		<i>read(B)</i>	at Y
<i>write(A)</i>			<i>write(B)</i>	
<i>read(B)</i>		at Y	<i>read(A)</i>	at X
<i>write(B)</i>			<i>write(A)</i>	

- Ο server X κάνει validate πρώτα το T ενώ ο server Y το U
- Μόνο ένα transaction μπορεί να εκτελεί validation και update κάθε φορά
- Κανένας server δεν μπορεί να κάνει validate το δεύτερο transaction αν δεν τελειώσει το πρώτο

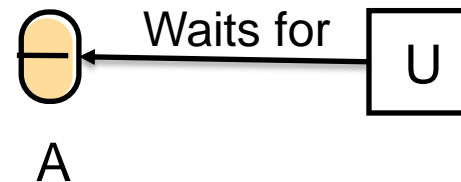
Deadlocks

- Resource allocation

- Ένας πόρος A χρησιμοποιείται από τη διεργασία U

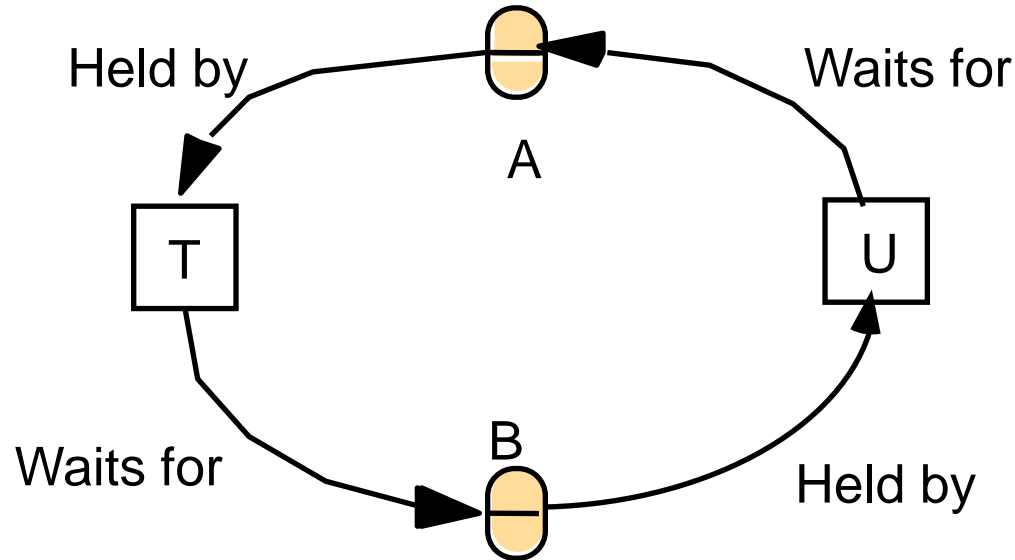


- Μια διεργασία U περιμένει τον πόρο A



- Ο γράφος που δημιουργείται λέγεται γράφος Wait-For

Παράδειγμα deadlock



Υπάρχει deadlock αν και μόνο αν υπάρχει κύκλος στον γράφο wait-for

Διαχείριση deadlocks

- Ισχύουν οι ίδιες συνθήκες στα κατανεμημένα όπως στα κεντρικά συστήματα
- Δυσκολότερο να ανιχνευθούν, να αποφευχθούν και να προληφθούν
- Στρατηγικές

1. Εντοπισμός

- Επιτρέπει στο deadlock να συμβεί, το εντοπίζει και το λύνει κάνοντας abort και επανεκκινώντας κάποιο από τα transactions που εμπλέκονται

2. Πρόληψη

- Κάνει αδύνατη την ύπαρξη deadlock απαντώντας σε αιτήματα έτσι ώστε να μην υπάρχει κυκλική εξάρτηση

3. Αποφυγή

- Σχεδιάζει την κατανομή πόρων έτσι ώστε να μη συμβαίνει deadlock (ο αλγόριθμος θα πρέπει να ξέρει ποιοι πόροι θα χρησιμοποιηθούν και πότε)

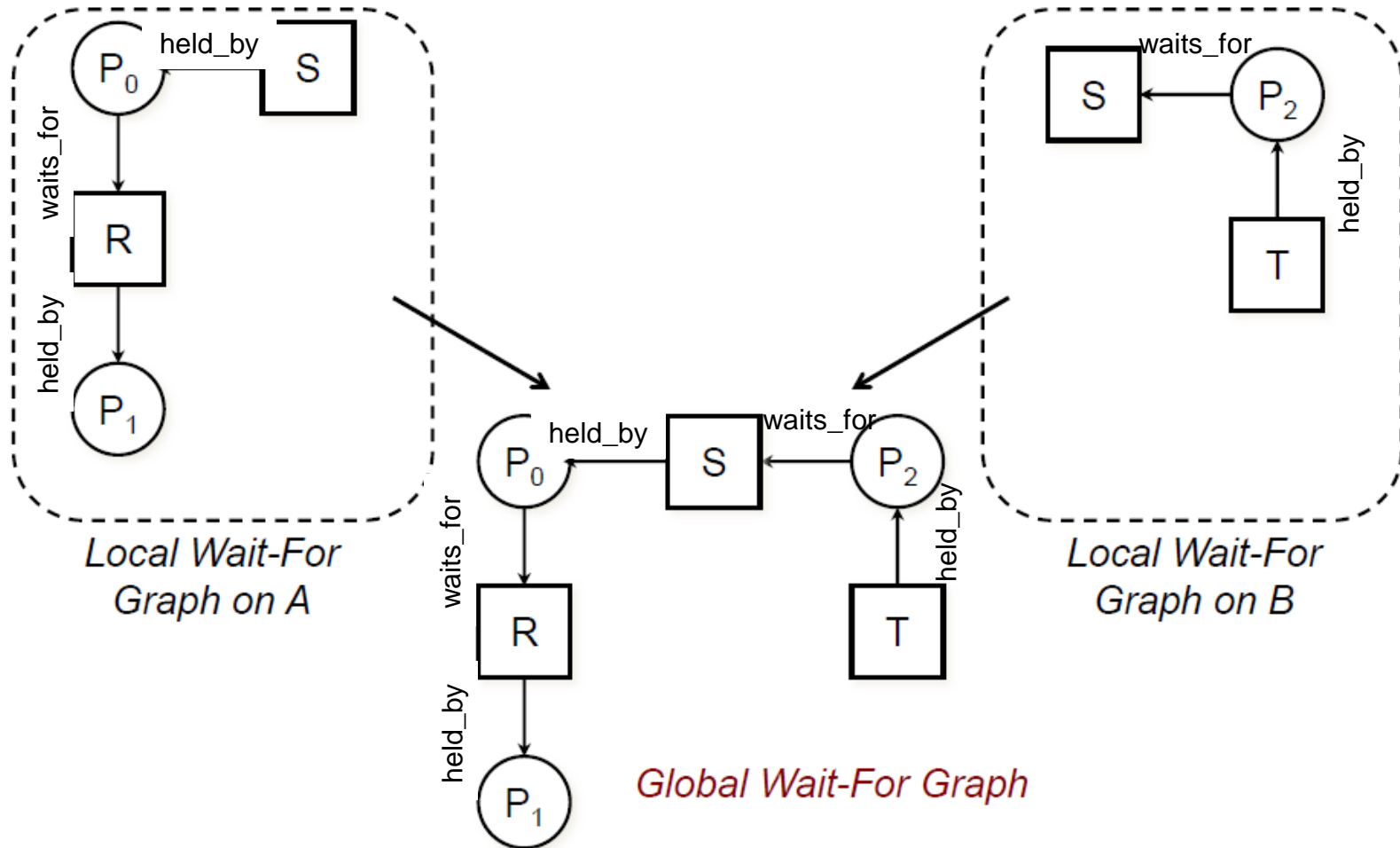
Εντοπισμός deadlock

- Τι θα μπορούσαμε να κάνουμε όταν εντοπίσουμε deadlock;
 - Σκοτώνουμε μια ή περισσότερες εμπλεκόμενες διεργασίες
 - Αυτό σπάει την κυκλική εξάρτηση
 - Δε φαίνεται και πολύ λογικό!
- Τα transactions όμως είναι σχεδιασμένα να είναι abortable
- Απλώς κάνουμε abort σε ένα ή περισσότερα transactions
 - Το σύστημα επανέρχεται στην κατάσταση που βρισκόταν πριν την έναρξη του transaction
 - Το transaction μπορεί να ξεκινήσει ξανά αργότερα
 - Τότε η κατανομή των resources μπορεί να είναι διαφορετική στο σύστημα, οπότε το transaction να πετύχει χωρίς να δημιουργήσει deadlock

Κεντρικός εντοπισμός

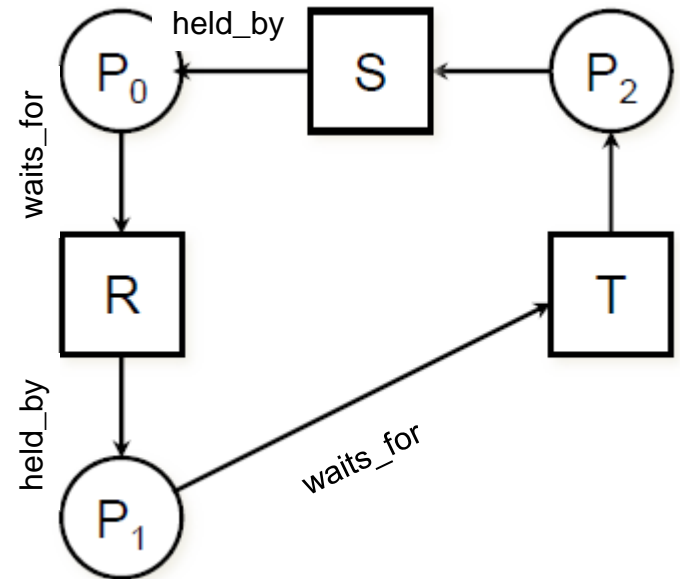
- Μιμείται τον μη κατανεμημένο αλγόριθμο με χρήση coordinator
- Κάθε κόμβος διατηρεί τοπικό Wait-For γράφο για τις διεργασίες και τους πόρους του
- Ένας κεντρικός coordinator διατηρεί τον καθολικό Wait-For γράφο του συστήματος (συνδυασμός τοπικών Wait-For γράφων)
 - Ένα μήνυμα με το τελευταίο αντίγραφο του τοπικού wait-for γράφου αποστέλλεται από έναν κόμβο στον coordinator κάθε φορά που προστίθεται ή αφαιρείται μια ακμή (held_by ή wait_for)
 - Η αποστολή μπορεί να γίνεται και περιοδικά (με κίνδυνο τον πιο αργό εντοπισμό ενδεχόμενου deadlock)

Κεντρικός εντοπισμός

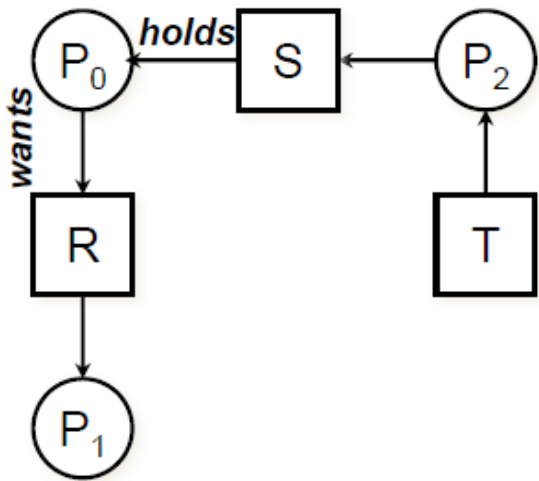


Phantom deadlocks

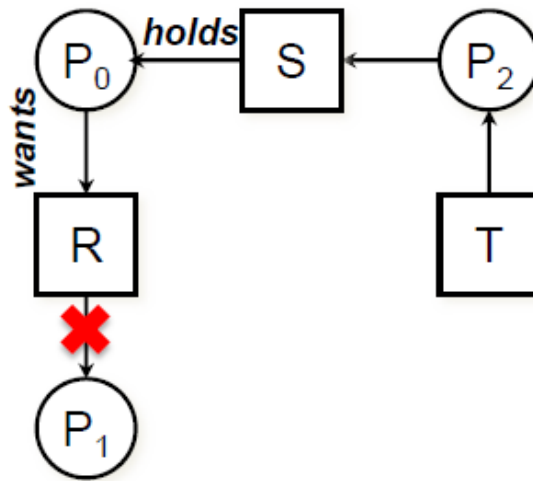
- Συμβαίνουν 2 γεγονότα:
 1. Η διεργασία P_1 απελευθερώνει τον πόρο R στον κόμβο A
 2. Η διεργασία P_1 ζητά τον πόρο T από τον κόμβο B
- Δυο μηνύματα αποστέλλονται στον coordinator:
 - 1 (από τον A): απελευθέρωσε τον R
 - 2 (από τον B): περιμένω τον T
- Αν το μήνυμα 2 φτάσει πρώτο, ο coordinator κατασκευάζει γραφο με κύκλο \rightarrow εντοπίζει deadlock (που στην πραγματικότητα δεν υπάρχει). Αυτό είναι ένα **phantom deadlock**



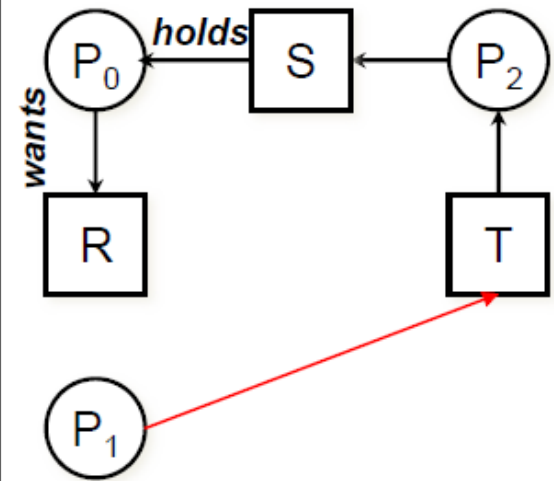
Παράδειγμα phantom deadlock



No deadlock



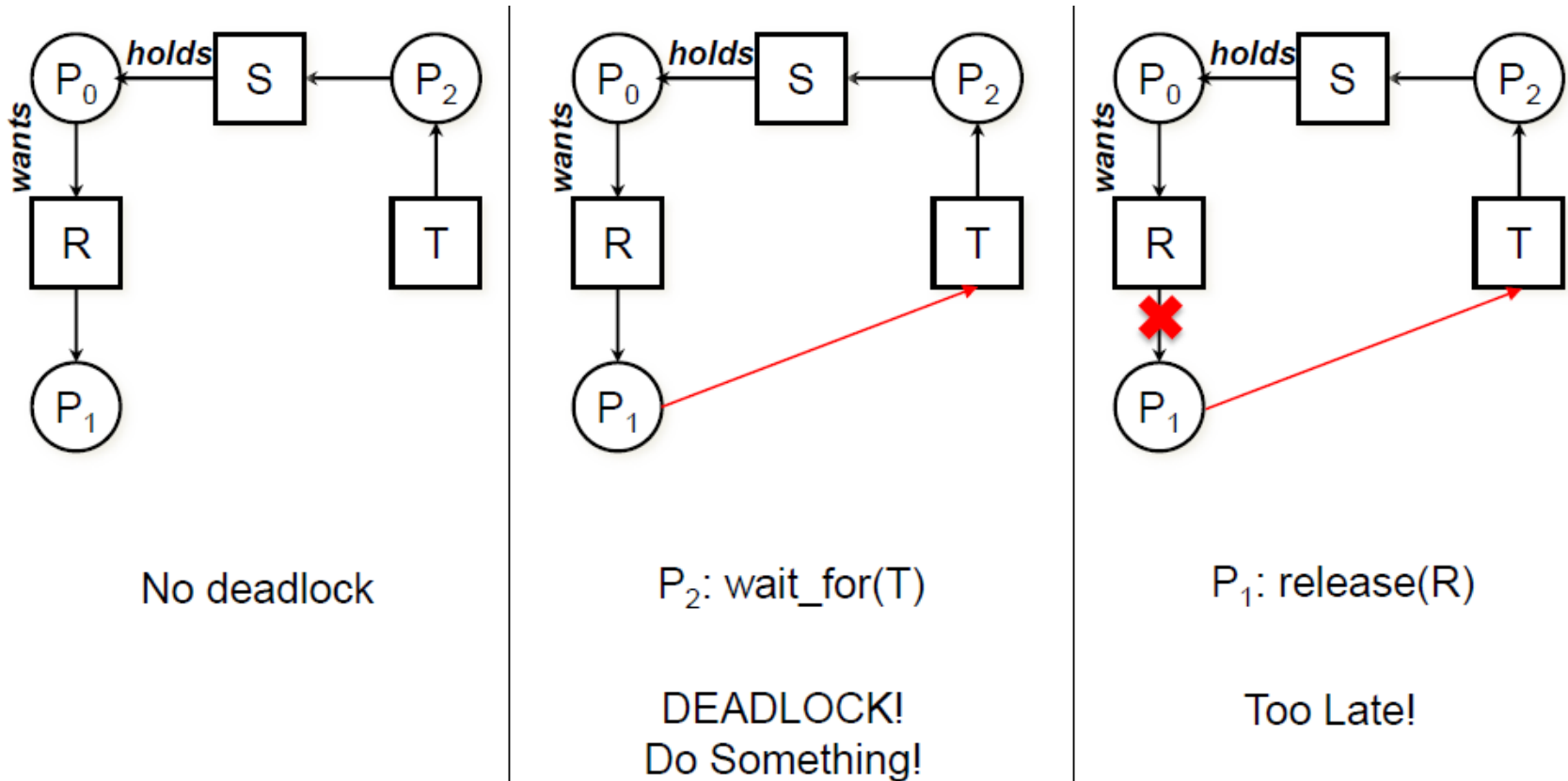
P_1 : release(R)



P_1 : wait_for(T)

Still no deadlock!

Παράδειγμα phantom deadlock



We detected deadlock because the coordinator received the messages out of order

Κατανεμημένος εντοπισμός

- Μια διεργασία μπορεί να περιμένει (waits-for) για πολλούς πόρους
 - Είτε τοπικούς πόρους
 - Είτε πόρους που βρίσκονται σε διαφορετικούς κόμβους
- Κάθε φορά που μια διεργασία πρέπει να περιμένει για έναν πόρο που βρίσκεται σε άλλον server καλείται ο αλγόριθμος κατανεμημένου εντοπισμού

Αλγόριθμος κατανεμημένου εντοπισμού

Edge Chasing

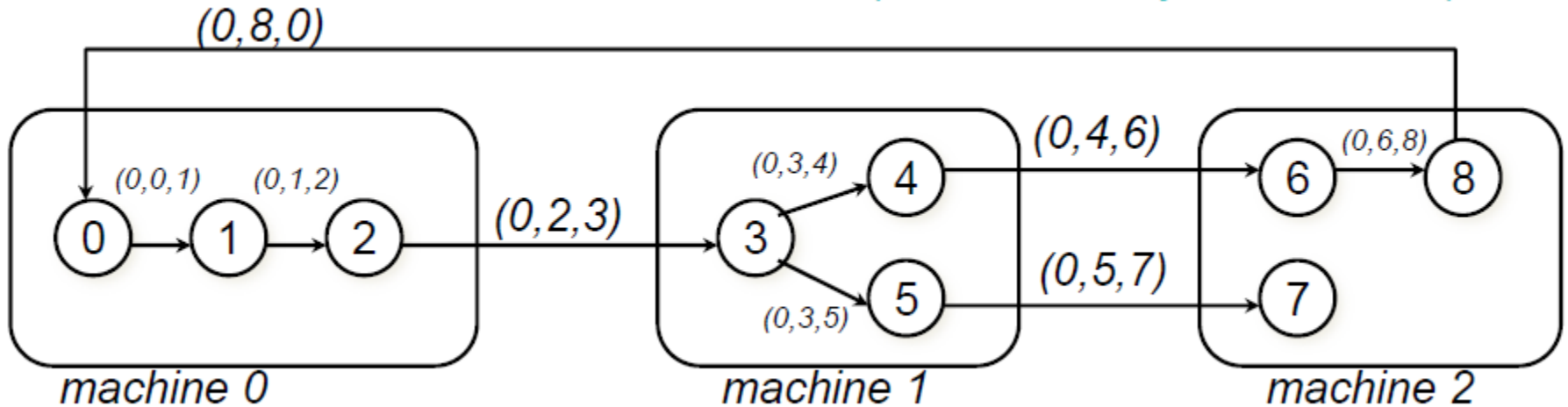
- Κάθε server έχει τοπικό κομμάτι του wait-for γράφου
- **Δημιουργείται ένα μήνυμα Probe**
 - Αποστέλλεται σε όλες τις διεργασίες που χρησιμοποιούν τον πόρο που περιμένουμε
 - Το μήνυμα περιέχει 3 process IDs: $\{blocked\ ID, my\ ID, holder\ ID\}$
 1. Η διεργασία που περιμένει
 2. Η διεργασία που στέλνει το μήνυμα
 3. Η διεργασία παραλήπτης (που χρησιμοποιεί τον πόρο)

Αλγόριθμος κατανεμημένου εντοπισμού

- Κατά την παραλαβή μηνύματος *probe* ο παραλήπτης ελέγχει αν ο ίδιος περιμένει κάποιον πόρο
 - Αν ναι, ενημερώνει και προωθεί το μήνυμα: $\{blocked\ ID, my\ ID, holder\ ID\}$
 - Αντικαθιστά το *myId* με το δικό του *process ID*
 - Αντικαθιστά το *holder ID* με τη διεργασία την οποία περιμένει
 - Στέλνει το μήνυμα σε κάθε διεργασία που τη μπλοκάρει
- Αν ένα μήνυμα επιστρέψει στον αρχικό αποστολέα, σημαίνει ότι υπάρχει κύκλος
 - Έχουμε *deadlock*

Παράδειγμα

(blocked ID, my ID, holder ID)



- Η διεργασία 0 μπλοκάρεται από την 1
 - Αρχικό μήνυμα από τη διεργασία 0 στην 1: $(0,0,1)$
 - Η P1 στέλνει $(0, 1, 2)$ στην P2, η P2 στέλνει $(0, 2, 3)$ στην P3, κ.ο.κ.
- Το μήνυμα $(0,8,0)$ επιστρέφει στην P0
 - Υπάρχει κύκλος: *deadlock*

Πρόληψη κατανεμημένου deadlock

- Σχεδιάζουμε το σύστημα έτσι ώστε τα deadlock να είναι δομικά αδύνατον να δημιουργηθούν
- Αποφυγή κυκλικών `wait_for`
- Αναθέτουμε μοναδικό `timestamp` σε κάθε `transaction`
- Διασφαλίζουμε ότι ο καθολικός *Wait-For* γράφος κατευθύνεται από τις νεότερες στις παλαιότερες διεργασίες ή το αντίστροφο

Πρόληψη καταναεμημένου deadlock

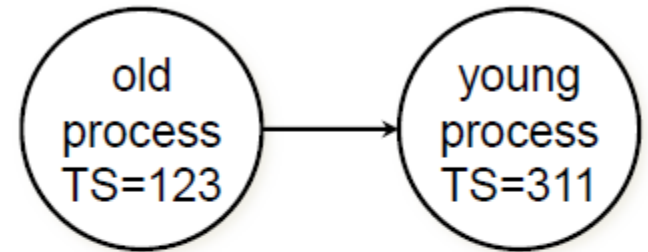
- Όταν μια διεργασία είναι έτοιμη να μπλοκάρει περιμένοντας έναν πόρο που χρησιμοποιείται από μια άλλη διεργασία τότε
 - Ελέγχει ποια διεργασία είναι η παλαιότερη (αυτή με το μικρότερο timestamp)
- Επιτρέπεται αναμονή (wait_for) μόνο αν η διεργασία που θα περιμένει είναι παλαιότερη
- Ακολουθώντας τον γράφο τα timestamps πάντα αυξάνουν, οπότε οι κύκλοι είναι αδύνατοι
- Εναλλακτικά: επιτρέπεται η αναμονή όταν η διεργασία που θα περιμένει είναι η νεώτερη (έχει το μεγαλύτερο timestamp)

Αλγόριθμος wait-die

- Μια παλιά διεργασία χρειάζεται έναν πόρο που χρησιμοποιεί μια νεότερη διεργασία
 - Η παλιότερη διεργασία περιμένει
- Μια νεότερη διεργασία χρειάζεται έναν πόρο που χρησιμοποιεί μια παλιότερη διεργασία
 - Η νεότερη διεργασία αυτοκτονεί

*wants
resource*

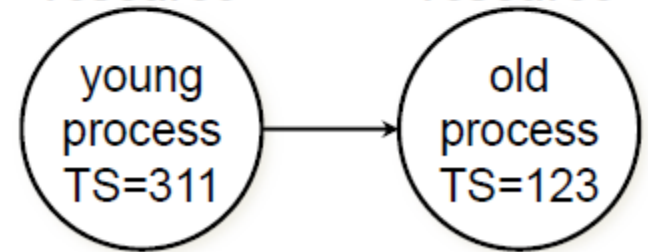
*holds
resource*



waits

*wants
resource*

*holds
resource*



dies

Ανακεφαλαίωση

- Distributed transactions
 - Flat
- Atomicity σε distributed transactions
 - 1-phase commit
 - 2-phase commit
 - 3-phase commit
- Concurrency control σε distributed transactions
 - Τοπικά σε κάθε server
 - Καθολικά στο σύνολο των servers
- Distributed Deadlocks
 - Detection
 - Prevention