Consensus

Κατανεμημένα Συστήματα
2017-2018

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

Τι θα δούμε

- Προβλήματα που απαιτούν συμφωνία
  - Πότε και γιατί
  - Ορισμός
- Byzantine Generals
  - Σύγχρονα συστήματα
  - Ασύγχρονα συστήματα
- Αδυναμία consensus
  - Πρακτικές λύσεις

Έχουμε δει ήδη…

- Mutual exclusion
  - Συμφωνία για το ποιος έχει πρόσβαση σε κάποιον πόρο
- Αλγόριθμοι εκλογών
  - Συμφωνία για το ποίος αναλαμβάνει κάποιον ξεχωριστό ρόλο
- Εκτέλεση κατανεμημένων transactions
  - Commit ή abort?
- Ολική διάταξη μηνυμάτων multicast
  - Ποια είναι η σειρά παράδοσης
- Συγχρονισμός replicas
- Γενικό πρόβλημα consensus:
  - *Πώς ερχόμαστε σε ομόφωνη απόφαση για μια συγκεκριμένη τιμή;*

Στόχος του consensus

- Επιτρέπει σε ομάδα διεργασιών να συμφωνήσουν σε ένα αποτέλεσμα
  - Όλες οι διεργασίες πρέπει να συμφωνήσουν στην ίδια τιμή
  - Η τιμή πρέπει να έχει υποβληθεί από τουλάχιστον μια διεργασία (ένας αλγόριθμος consensus δεν μπορεί απλώς να επινοήσει μια τιμή)

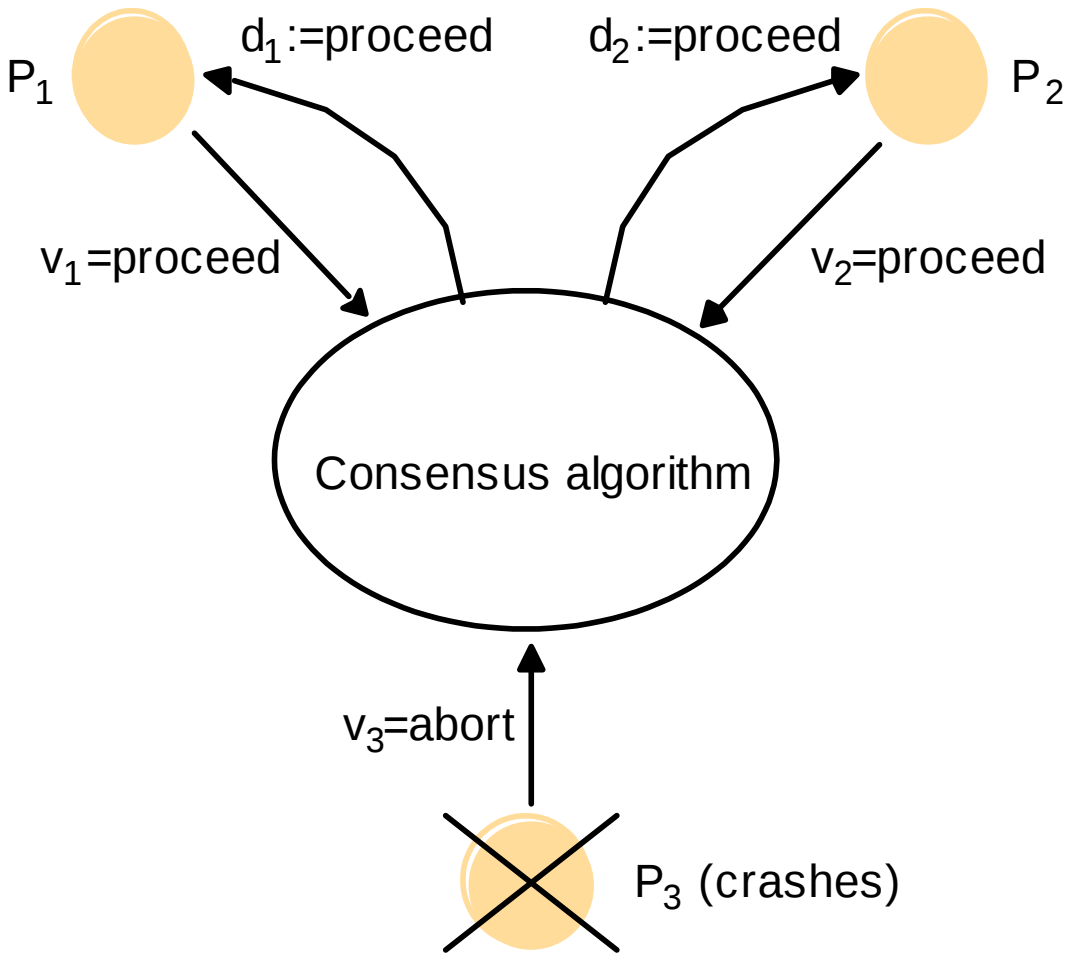
Μοντέλο συστήματος

- Οι διεργασίες αποτυγχάνουν
  - μόνο με *crash-stop*
  - Με οποιονδήποτε τρόπο – Βυζαντινά σφάλματα
- Σύγχρονο σύστημα: Όριο σε
  - Καθυστερήσεις μηνυμάτων
  - Μέγιστος χρόνος για κάθε βήμα επεξεργασίας
- Ασύγχρονο σύστημα: δεν υπάρχουν τέτοια όρια
  - Π.χ. το Internet

Ορισμός του προβλήματος

- Ν διεργασίες
- Αρχικά κάθε διεργασία p
  - Ξεκινά από απροσδιόριστη κατάσταση (undecided state)
  - Προτείνει μια αρχική τιμή από domain D
- Οι διεργασίες επικοινωνούν προσπαθώντας να συμφωνήσουν σε μια τιμή
  - Δεν μπορούν να αλλάξουν την απόφαση όταν μπουν σε decided state
- Το πρόβλημα consensus: Ορισμός πρωτοκόλλου έτσι ώστε
  - Όλες οι σωστές διεργασίες να θέσουν την ίδια τιμή
  - Ακόμα κι αν συμβεί σφάλμα

Consensus για 3 διεργασίες



Απαιτήσεις

- Ισχύς (validity)
  - Μπορούν να επιλεγούν μόνο τιμές που έχουν προταθεί
- Ομοιόμορφη συμφωνία
  - Δε γίνεται δύο κόμβοι να επιλέξουν διαφορετικές τιμές
- Ακεραιότητα (integrity)
  - Αν όλες οι σωστές διεργασίες προτείνουν την ίδια τιμή, τότε οποιαδήποτε διεργασία έχει έρθει σε decided state θα πρέπει να έχει επιλέξει την τιμή αυτή.
- Τερματισμός
  - Κάθε κόμβος θα επιλέξει τελικά (eventually) μια τιμή

Προς τη λύση

- Για απλότητα, θεωρούμε ότι δεν υπάρχουν σφάλματα
  - Οι διεργασίες στέλνουν με multicast προτεινόμενες τιμές στις άλλες
  - Περιμένουν μέχρι να συλλέξουν όλες τις N προτάσεις (μαζί με τη δική τους)
  - Αποφασίζουν με πλειοψηφία (⊥ ειδική τιμή για null)
    - Μπορεί να χρησιμοποιηθεί και minimum/maximum
- Δουλεύει γιατί…
  - Όλες οι διεργασίες καταλήγουν με το ίδιο σύνολο N τιμών
  - Η πλειοψηφία εξασφαλίζει ισχύ, συμφωνία και ακεραιότητα
- Τι γίνεται με σφάλματα;
  - process crash – σταματά να στέλνει τιμές
  - arbitrary failure – στέλνει διαφορετικές τιμές σε διαφορετικές διεργασίες

Βυζαντινοί στρατηγοί

- Το πρόβλημα [Lamport 1982]
  - 3 ή περισσότεροι στρατηγοί πρέπει να αποφασίσουν επίθεση ή οπισθοχώρηση
  - Ένας διοικητής (commander) δίνει την εντολή
  - Οι υπόλοιποι (lieutenants) αποφασίζουν
  - Ένας ή περισσότεροι είναι προδότες (=αναξιόπιστοι)
    - Προτείνουν επίθεση σε κάποιους και οπισθοχώρηση στους άλλους
    - Μπορεί να είναι προδότες είτε commander είτε lieutenants!
- Απαιτήσεις
  - Τερματισμός, συμφωνία όπως πριν
  - Ακεραιότητα: Αν ο commander είναι σωστός, τότε όλες οι σωστές διεργασίες πρέπει να αποφασίσουν την τιμή που πρότεινε ο commander

Σύγχρονο σύστημα+crash failures

- Χρησιμοποιεί βασικό multicast
  - Εγγυάται παράδοση από σωστές διεργασίες αν ο αποστολέας δεν πεθάνει
- Υποθέτει μόνο crash failures των διεργασιών
  - υποθέτουμε ότι μέχρι και f από τις N διεργασίες μπορεί να πεθάνουν
- Πώς δουλεύει…
  - σε f +1 γύρους
  - Βασίζεται σε timeouts (σύγχρονο σύστημα)

Σύγχρονο σύστημα+crash failures

- Αρχικά
  - Κάθε διεργασία προτείνει μια τιμή από ένα σύνολο D
- Κάθε διεργασία
  - Διατηρεί το σύνολο τιμών V_r που έλαβε στον γύρο r
- Σε κάθε γύρο r, όπου $1 \leq r \leq f+1$, κάθε διεργασία
  - Στέλνει με multicasts τις τιμές που γνωρίζει σε όλες τις άλλες (μόνο τις τιμές που δεν έχει ξαναστείλει στο παρελθόν, δλδ $V_r - V_{r-1}$)
  - Λαμβάνει μηνύματα multicast, ανανεώνοντας το V_r με τυχόν νέες τιμές
- Στον γύρο f+1
  - Κάθε διεργασία διαλέγει την ελάχιστη τιμή από το σύνολο V_{f+1} ως τελική

Γιατί δουλεύει;

- Θέτει το timeout ίσο με τον μέγιστο χρόνο που θέλει μια σωστή διεργασία να στείλει multicast
- Έτσι μπορεί να αποφανθεί αν μια διεργασία έχει πεθάνει (αν δε λάβει μήνυμα)
- Αν μια διεργασία πεθάνει, κάποια τιμή μπορεί να μην προωθήθηκε
- Στον γύρο f+1
  - Όλες οι σωστές διεργασίες έχουν το ίδιο σύνολο τιμών
  - οπότε αποφασίζουν για την ίδια τιμή (minimum)
  - τουλάχιστον f+1 γύροι απαιτούνται για ανοχή σε f crash failures

Απόδειξη

- Ας υποθέσουμε ότι 2 σωστές διεργασίες p_i και p_j διαφέρουν στο τελικό σύνολο τιμών ✉ proof by contradiction
- Ας υποθέσουμε ότι η p_i έχει μια τιμή v που δεν έχει η p_j
- Άρα η p_j δεν πρέπει να έχει λάβει την v σε κανένα γύρο

✉ Στον τελευταίο γύρο, κάποια τρίτη διεργασία p_k έστειλε v στην p_i και πέθανε πριν τη στείλει στην p_j.

→ Οποιαδήποτε διεργασία που έστειλε v στον προτελευταίο γύρο πρέπει να πέθανε, αλλιώς θα είχαν λάβει και οι δύο p_k και p_j την τιμή v

→ Προχωρώντας έτσι, σε καθέναν από τους προηγούμενους γύρους θα πρέπει να πέθαινε τουλάχιστον μια διεργασία

✉ Αλλά υποθέσαμε το πολύ f σφάλματα και f+1 γύρους ==> contradiction.

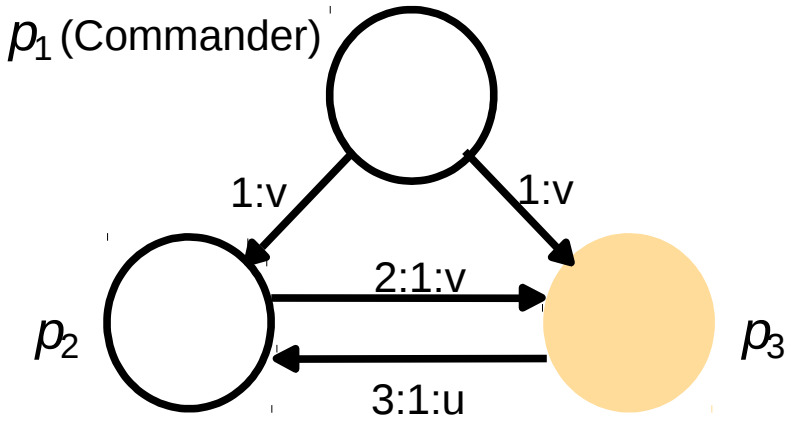
Βυζαντινοί στρατηγοί

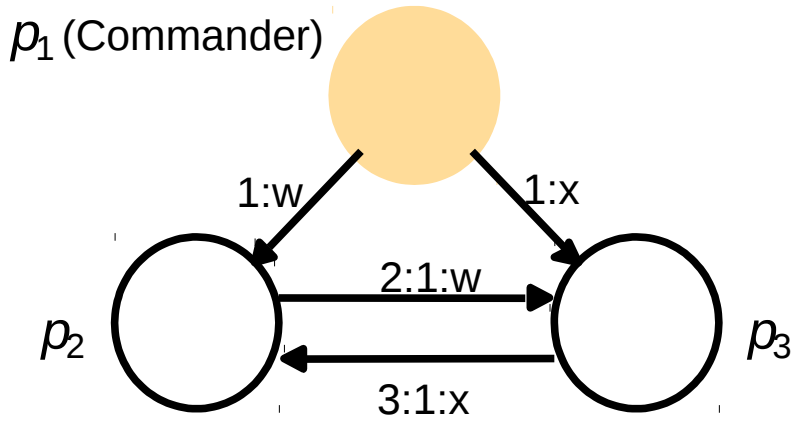
- Διεργασίες με «βυζαντινά σφάλματα» (arbitrary failures)
  - f από τις N διεργασίες αναξιόπιστες
- Σε σύγχρονο σύστημα
  - Χρησιμοποιούμε timeout για ανίχνευση απουσίας μηνύματος
  - Δεν μπορούμε να αποφανθούμε αν η διεργασία κράσαρε ή απλώς δεν απαντάει
  - Αδυναμία consensus με $N \leq 3f$

3 Στρατηγοί

- Assume synchronous system
  - 3 processes, one faulty
  - if no message received, assume ⊥
  - proceed in rounds
  - messages ‘3:1:u’ meaning ‘3 says 1 says u’
- Problem! ‘1 says v’ and ‘3 says 1 says u’
  - cannot tell which process is telling the truth!
  - goes away if digital signatures used…
- Show – no solution to agreement for N=3 and f=1
- Can generalise to impossibility for N ≤ 3f

3 Βυζαντινοί Στρατηγοί



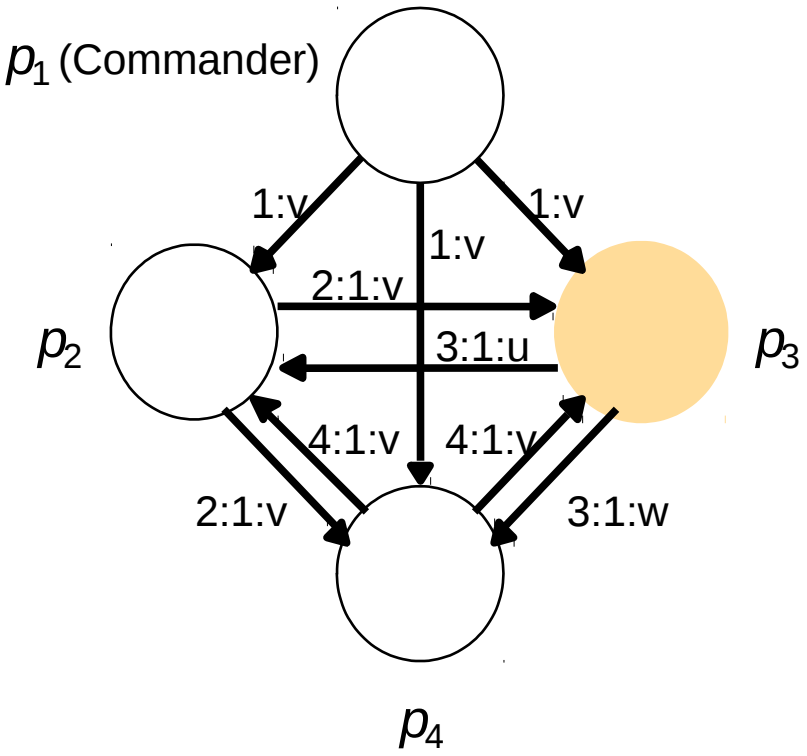


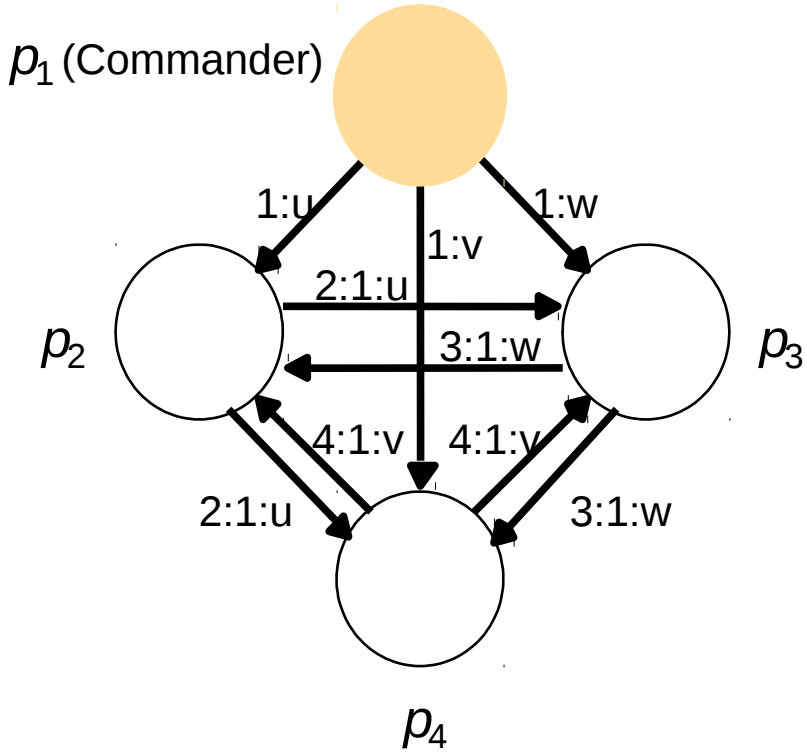
Faulty processes are shown coloured

Όμως…

- Solution exists for 4 processes with one faulty
  - commander sends value to each of the lieutenants
  - each lieutenant sends value it received to its peers
  - if commander faulty, then correct lieutenants have gathered all values sent by the commander
  - if one lieutenant faulty, the each correct lieutenant receives 2 copies of the value from the commander
- Thus – correct lieutenants can decide on majority of the values received
- Can generalize to $N \geq 3f + 1$

4 Βυζαντινοί Στρατηγοί





Faulty processes are shown coloured

p2 decides majority (v,u,v)
p4 decides majority (v,v,w)

No majority exists

Ασύγχρονο σύστημα

- Τα μηνύματα μπορούν να έχουν αυθαίρετη καθυστέρηση και οι επεξεργασίες να είναι αυθαίρετα αργές
- Είναι αδύνατον να εγγυηθούμε ότι θα πετύχουμε consensus
  - Ακόμα και μια αποτυχία είναι ικανή να αποτρέψει συμφωνία
  - Μια αργή διεργασία δε διακρίνεται από μια διεργασία που έχει πεθάνει
- Η αδυναμία ισχύει για οποιοδήποτε πρωτόκολλο consensus
- Αποδείχτηκε από Fischer, Lynch και Patterson το 1983 (FLP)
  - Λύθηκε διαφωνία που υπήρχε για μια δεκαετία

Είμαστε καταδικασμένοι;

- Τα ασύγχρονα συστήματα δεν μπορούν να εγγυηθούν ότι θα έρθουν σε συμφωνία ακόμα και με αποτυχία μιας μόνο διεργασίας

- Κλειδί: “εγγύηση”
  - Δε σημαίνει ότι οι διεργασίες δεν μπορούν να έρθουν σε συμφωνία αν μια από αυτές αποτύχει
  - Επιτρέπει συμφωνία με κάποια πιθανότητα μεγαλύτερη από 0
  - Στην πράξη πολλά συστήματα επιτυγχάνουν consensus

- Πώς το ξεπερνάμε;

Τεχνικές για να ξεπεράσουμε το αδύνατο

- Τεχνική 1: κρύβουμε τις αποτυχίες (crash-stop)
  - Χρήση persistent storage και τοπικών checkpoints
  - Μετά από αποτυχία, επανεκκίνηση της διεργασίας και ανάνηψη από το τελευταίο checkpoint
  - Μπορεί να εισάγει αυθαίρετες καθυστερήσεις

- Τεχνική 2: χρησιμοποιούμε ανιχνευτές λαθών
  - Π.χ., αν μια διεργασία είναι αργή, θεωρούμε ότι έχει αποτύχει
  - Τότε τη σκοτώνουμε πραγματικά ή αγνοούμε τα μηνύματά της από εκείνο το σημείο και μετά (fail-silent)
  - Αυτό μετατρέπει το ασύγχρονο σε σύγχρονο σύστημα
  - Οι ανιχνευτές λαθών δεν είναι 100% ακριβείς και απαιτούν μεγάλο timeout για να έχουν μια λογική συμπεριφορά

Σκοπός ενός αλγορίθμου consensus

- Ανοχή σε σφάλματα
  - Δεν μπλοκάρει όταν η πλειονότητα των διεργασιών δουλεύουν
- Συμφωνία σε ένα αποτέλεσμα ανάμεσα σε ομάδα διεργασιών ακόμα κι αν:
  - Κάποιες διεργασίες πεθάνουν
  - Κάποια μηνύματα χαθούν ή έρθουν εκτός σειράς
  - Αν παραδοθούν, τα μηνύματα δεν αλλοιώνονται

Paxos

- Ένας αλγόριθμος consensus
  - Ένας από τους πιο αποδοτικούς και κομψούς αλγορίθμους
  - Πασίγνωστος

- Τι γίνεται με το FLP (impossibility of consensus)?
  - Προφανώς δε λύνει το FLP
  - Βασίζεται σε ανιχνευτές σφαλμάτων για παράκαμψη

Η ιστορία

- Δημιουργήθηκε από τον Leslie Lamport
- Το abstract του paper
  - *"Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems."*

Η ιστορία

- Ο κόσμος αρχικά νόμιζε ότι πρόκειται για αστείο

- Ο Lamport δημοσίευσε τελικά το paper 8 χρόνια μετά τη συγγραφή του, το 1998 ("The Part-Time Parliament")
- Κανείς δεν το καταλάβαινε

- Ο Lamport έγραψε άλλο paper που εξηγεί το Paxos σε απλά Αγγλικά
  - Τίτλος: "Paxos Made Simple"
  - Abstract: "The Paxos algorithm, when presented in plain English, is very simple."

- Ακόμα κι έτσι, ο αλγόριθμος είναι δυσνόητος

Τελικά όμως…

- Πολλά πραγματικά συστήματα υλοποιούν το Paxos
  - Google Chubby
  - MS Bing cluster management
  - AWS
  - Cassandra
- Amazon CTO Werner Vogels (σε blog post “Job Openings in My Group”)
  - *“What kind of things am I looking for in you?”*
  - *“You know your distributed systems theory: You know about logical time, snapshots, stability, message ordering, but also acid and multi-level transactions. You have heard about the FLP impossibility argument. You know why failure detectors can solve it (but you do not have to remember which one diamond-w was). You have at least once tried to understand Paxos by reading the original paper.”*

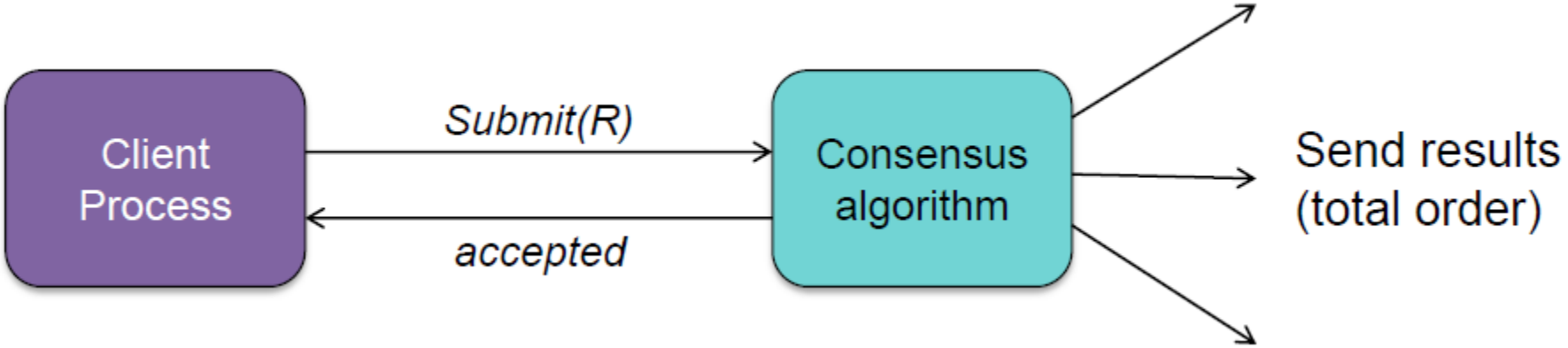
Υποθέσεις

- Το δίκτυο είναι ασύγχρονο με καθυστερήσεις στα μηνύματα
- Το δίκτυο μπορεί να χάσει μηνύματα ή να τα στείλει περισσότερες από μια φορές αλλά δεν μπορεί να τα αλλοιώσει
- Οι διεργασίες μπορεί να κρασάρουν (και να σταματήσουν)
- Οι διεργασίες έχουν *permanent storage* (δίσκο)
- Οι διεργασίες μπορούν να προτείνουν τιμές
- Ο στόχος είναι όλες οι διεργασίες να συμφωνήσουν σε μία από τις προτεινόμενες τιμές

Επιθυμητές ιδιότητες

- Safety
  - Μπορεί να επιλεγεί μόνο τιμή που έχει προταθεί
  - Επιλέγεται μια μόνο τιμή
  - Μια διεργασία μαθαίνει μια τιμή μόνο αν έχει επιλεγεί
- Liveness
  - Κάποια προτεινόμενη τιμή επιλέγεται τελικά (eventually)
  - Αν επιλεγεί μια τιμή, κάθε διεργασία τελικά θα τη μάθει

Η οπτική του χρήστη



- while (submit_request(R) != ACCEPTED) ;
- Το R θα μπορούσε να είναι ένα key:value ζεύγος μιας βάσης

Οι παίκτες του Paxos

- **Client:**
  - Στέλνει ένα αίτημα
- **Proposers:**
  - Λαμβάνουν αίτημα από τον client και τρέχουν το πρωτόκολλο
  - Leader: Εκλεγμένος coordinator ανάμεσα στους proposers (δεν είναι απαραίτητος, απλώς απλοποιεί τη διάταξη μηνυμάτων και διασφαλίζει ότι δεν υπάρχει διαφωνία)
- **Acceptors:**
  - Πολλαπλές διεργασίες που θυμούνται το state του πρωτοκόλλου
  - Quorum = πλειονότητα από acceptors
- **Learners:**
  - Όταν οι acceptors συμφωνήσουν, ο learner εκτελεί το αίτημα ή/και στέλνει απάντηση στον client

Διαφορετικοί ρόλοι μπορεί να συνυπάρχουν στον ίδιο κόμβο

Τι κάνει το Paxos

- Κάθε πρόταση (αίτημα) έχει ένα ID.
  - (proposal #, value) == (N, V)
  - Ο αύξων αριθμός του proposal είναι καθολικά μοναδικός

- Τρεις φάσεις
  - Φάση prepare: ένας proposer learns previously-accepted proposals from the acceptors.
  - Φάση Propose: ένας proposer στέλνει proposal.
  - Φάση Learn: οι learners μαθαίνουν το αποτέλεσμα

Τι κάνει το Paxos

- Proposers
  - Πριν να προτείνει κάποιος proposer θα ρωτήσει τους acceptors αν έχει ήδη προταθεί κάποια άλλη τιμή
  - Αν ναι, ο proposer θα προτείνει την ίδια
  - Η συμπεριφορά είναι αλτρουιστική: Ο στόχος του κάθε proposer είναι η ομοφωνία, όχι να επιλεγεί η τιμή που προτείνει
- Acceptors
  - Ο στόχος είναι να επιλέξει το proposal με τον μεγαλύτερο αύξοντα αριθμό από όλους τους proposers
- Learners
  - Οι learners είναι παθητικοί και περιμένουν το αποτέλεσμα

1η φάση

- Ένας proposer επιλέγει τον αύξοντα αριθμό N του proposal του και στέλνει *prepare request* στους acceptors με προτεινόμενη τιμή v.
  - “Hey, have you accepted any proposal yet?”

- Ένας acceptor πρέπει να απαντήσει:
  - Αν το proposal έχει μεγαλύτερο N από όλα τα proposals που έχει δεχτεί μέχρι τώρα τότε στέλνει υπόσχεση να μη δεχτεί πλέον κανένα proposal με αύξοντα αριθμό μικρότερο του N
  - Αν έχει ήδη δεχτεί κάποιο proposal, συμπεριλαμβάνει και την τιμή N’ και v’ του proposal με τον μεγαλύτερο αύξοντα αριθμό
  - Αν έχει ήδη δεχτεί proposal με N’ > N αγνοεί το request (ή στέλνει nack)

2η φάση

- Αν ένας proposer λάβει απάντηση από την πλειοψηφία, τότε στέλνει *accept request* για το proposal (N, V).
  - V: Η τιμή του proposal με τον μεγαλύτερο αύξοντα αριθμό N (από αυτές που επεστράφησαν στην 1η φάση).

- Μετά τη λήψη του (N, V), οι acceptors είτε:
  - Το δέχονται
  - Το απορρίπτουν αν υπάρχει άλλο prepare request με N' μεγαλύτερο του N και έχει απαντήσει σε αυτό

3η φάση

- Οι learners πρέπει να μάθουν ποια τιμή επελέγη
- Ένας τρόπος: Κάθε acceptor απαντάει σε όλους τους learners
  - ακριβό
- Άλλος τρόπος: εκλογή “distinguished learner”
  - Οι Acceptors απαντούν σε αυτόν
  - Αυτός ενημερώνει τους υπόλοιπους
  - Επιρρεπές σε σφάλματα
- Μίξη των δύο: σύνολο από distinguished learners

Πρόβλημα

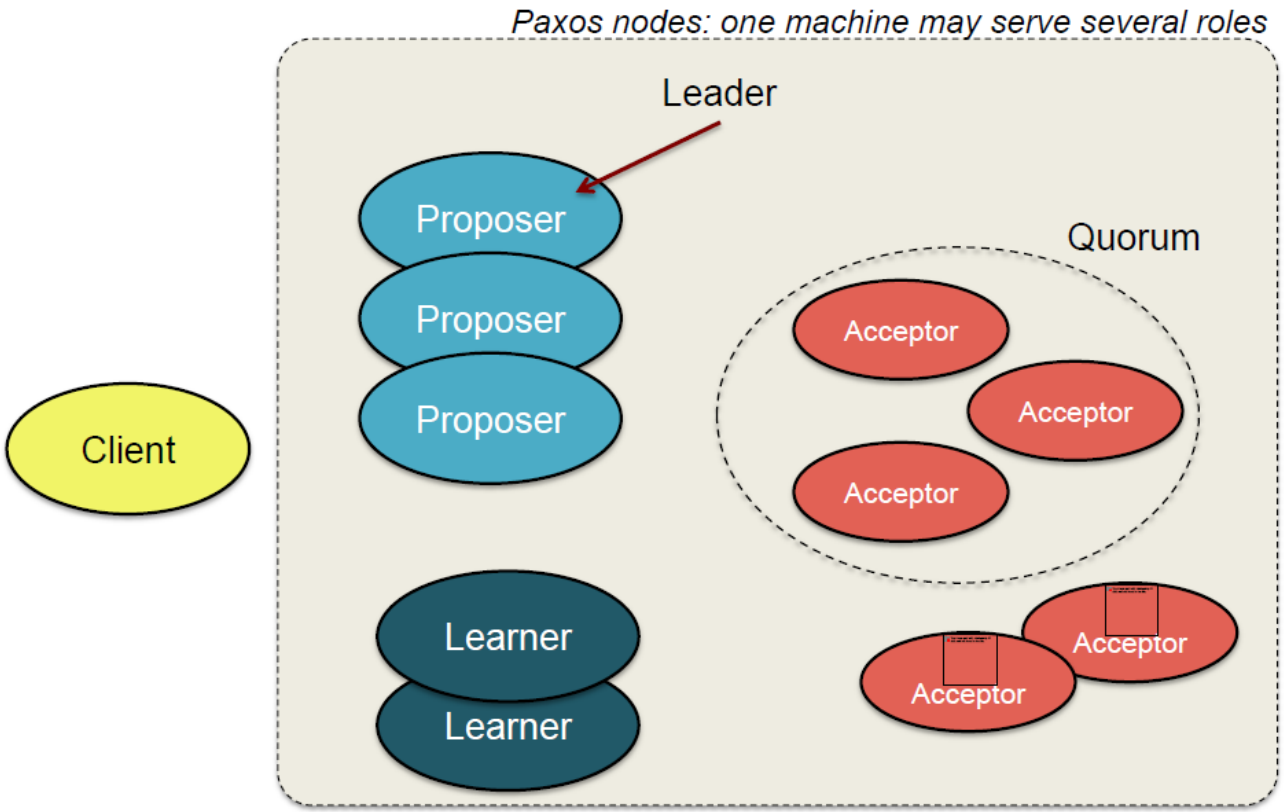
- *race condition για τα proposals.*
- Η P_0 ολοκληρώνει την φάση 1 με proposal number N_0
- Προτού η P_0 ξεκινήσει την φάση 2, η P_1 ξεκινά και ολοκληρώνει την φάση 1 με proposal number $N_1 > N_0$
- Η P_0 ολοκληρώνει την φάση 2, οι acceptors το απορρίπτουν
- Προτού η P_1 ξεκινήσει την φάση 2, η P_0 ξεκινά ξανά και ολοκληρώνει την φάση 1 με proposal number $N_2 > N_1$
- Η P_1 ολοκληρώνει την φάση 2, οι acceptors το απορρίπτουν
- …(συνεχίζεται επ' άπειρον)

Λύση

- Λύση: εκλογή ενός διακεκριμένου proposer
- Αν ο proposer μπορεί να επικοινωνήσει επιτυχώς με την πλειονότητα των κόμβων, τότε το πρωτόκολλο εξασφαλίζει liveness.
  - δηλαδή, το Paxos έχει ανοχή σε $f < 1/2 * N$ σφάλματα

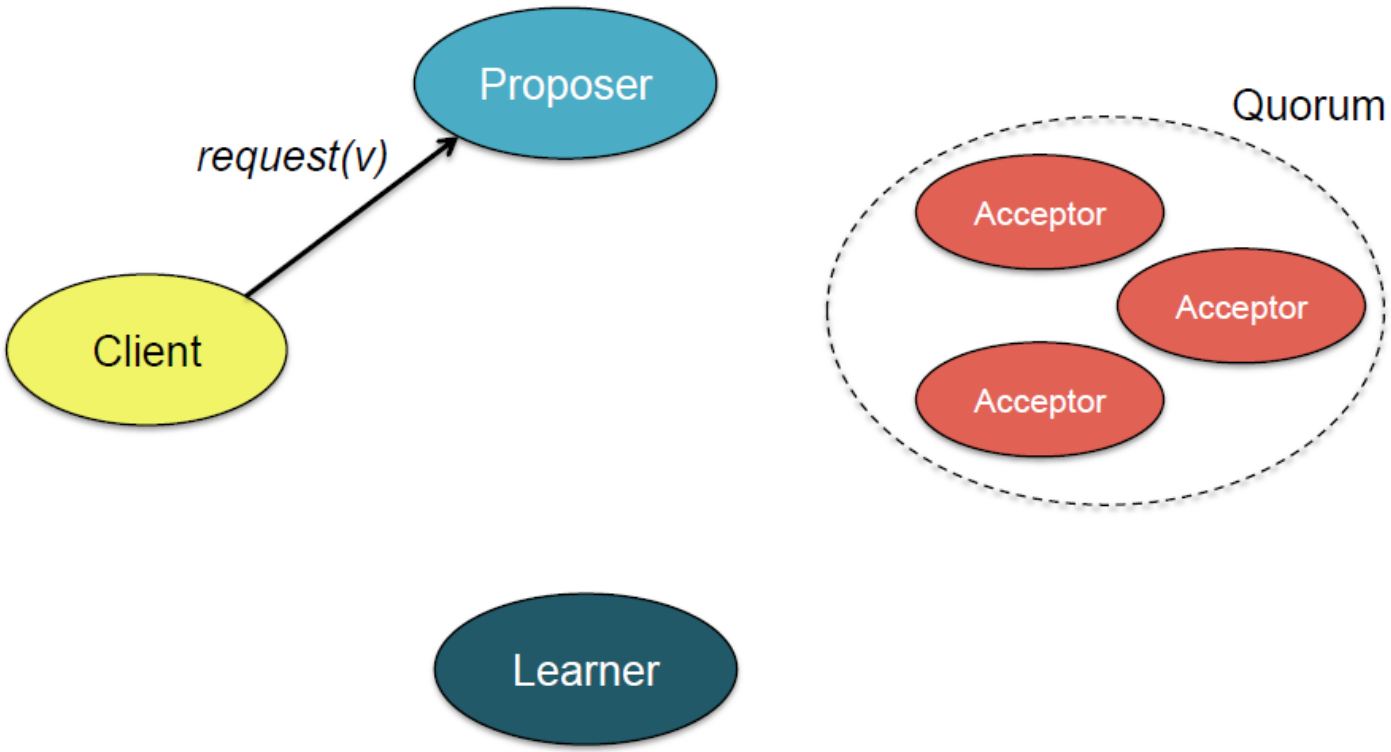
Paxos εν δράση

- Ένας proposer εκλέγεται ως leader και δέχεται όλα τα requests



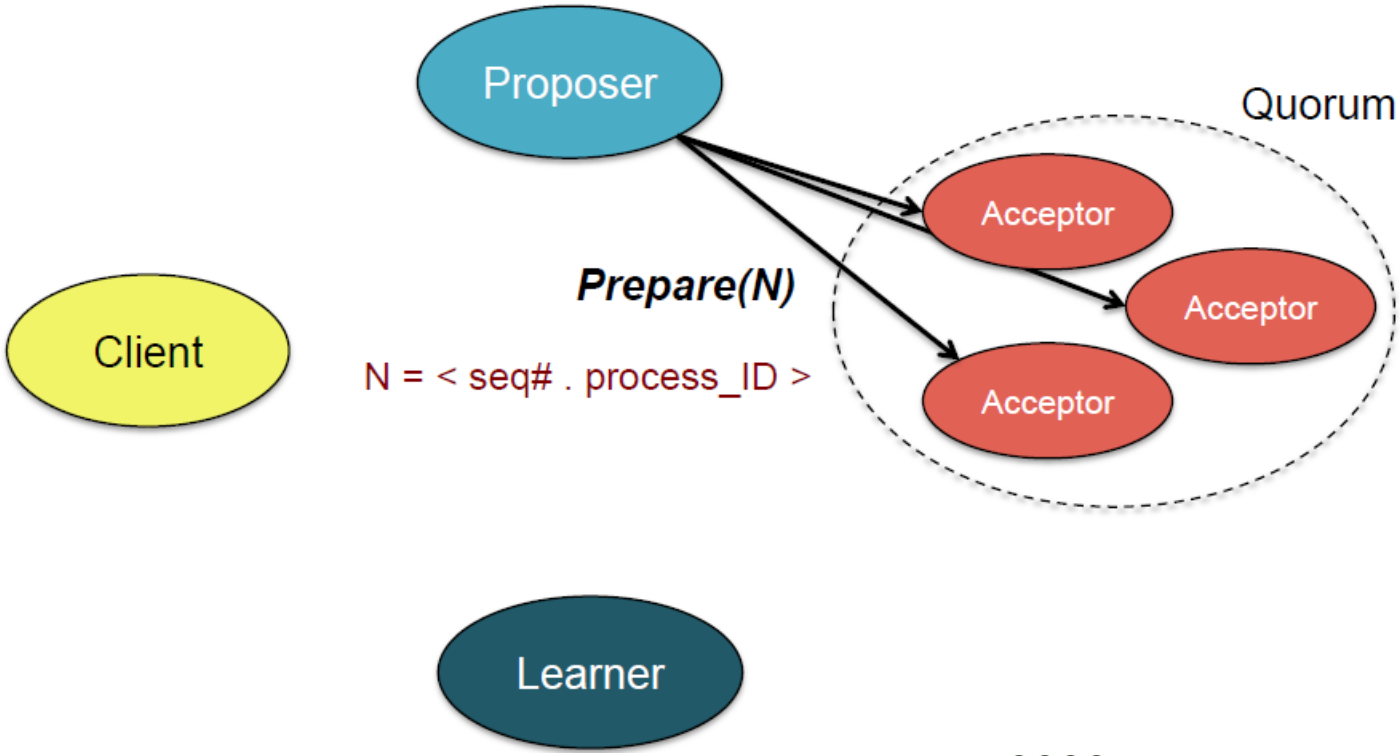
Phase 0

- Ο client στέλνει request στον proposer



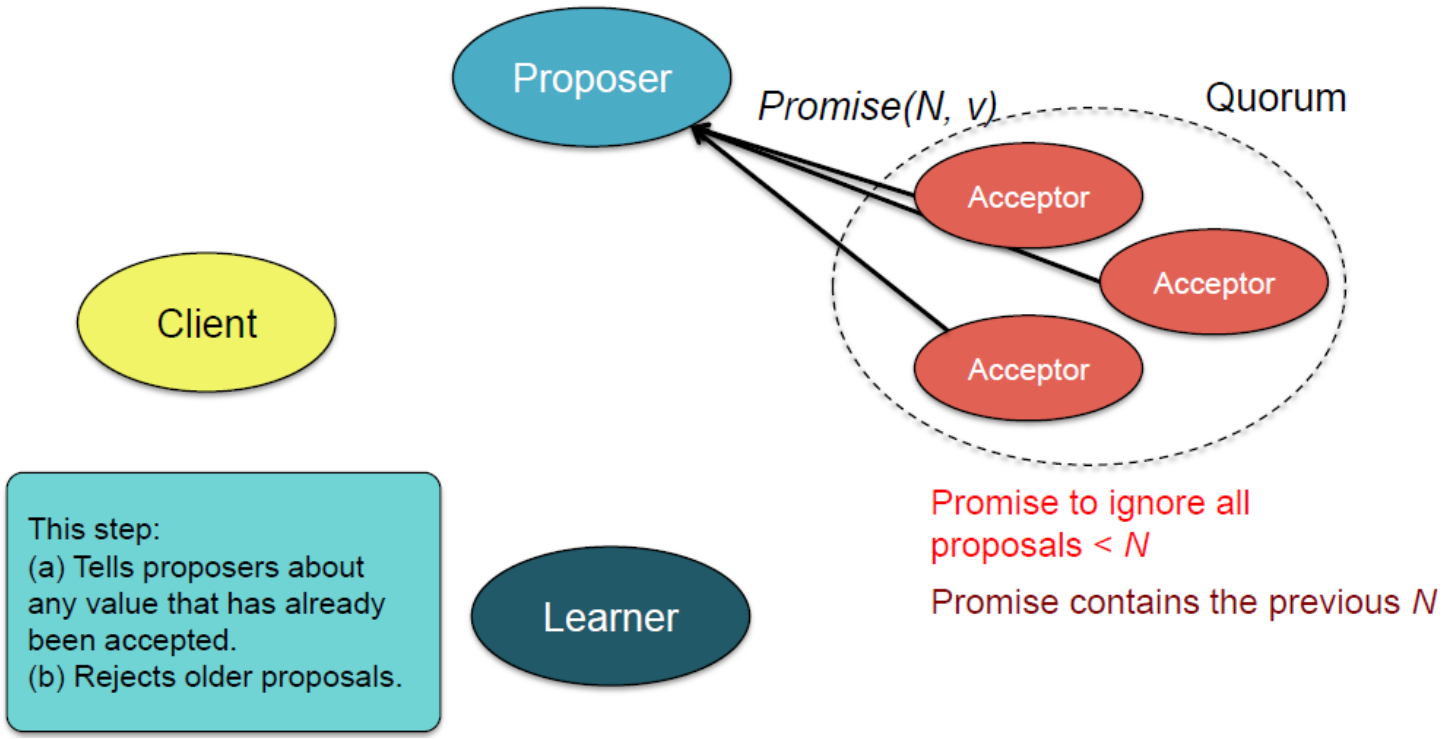
Phase 1a - PREPARE

- **Proposer: δημιουργεί ένα *proposal #N (N λειτουργεί σαν χρονοσφραγίδα Lamport)*,** όπου το N είναι μεγαλύτερο από οποιοδήποτε προηγούμενο proposal number που χρησιμοποιήθηκε από τον proposer
- **Αποστολή σε quorum από Acceptors (όσους μπορεί να προσπελάσει – σίγουρα πλειονότητα)**



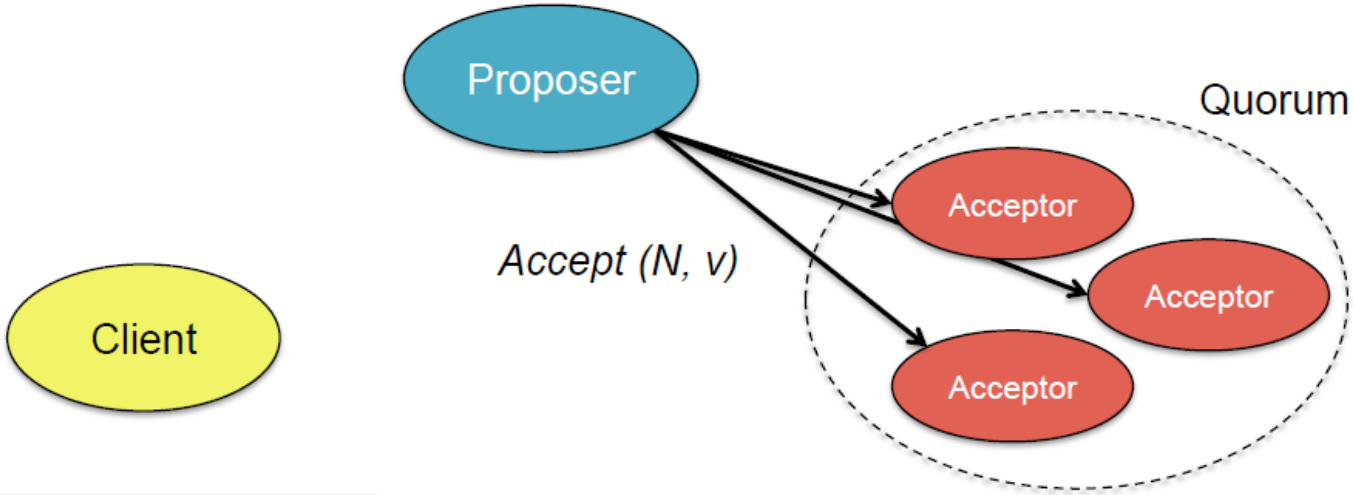
Phase 1a - PROMISE

- **Acceptor: αν το ID του proposal > οποιοδήποτε προηγούμενο proposal**
- Υπόσχεται να αγνοήσει όλα τα αιτήματα με ID < N και απαντάει με πληροφορίες για το πιο πρόσφατο proposal που έχει κάνει accept
- Αλλιώς το απορρίπτει



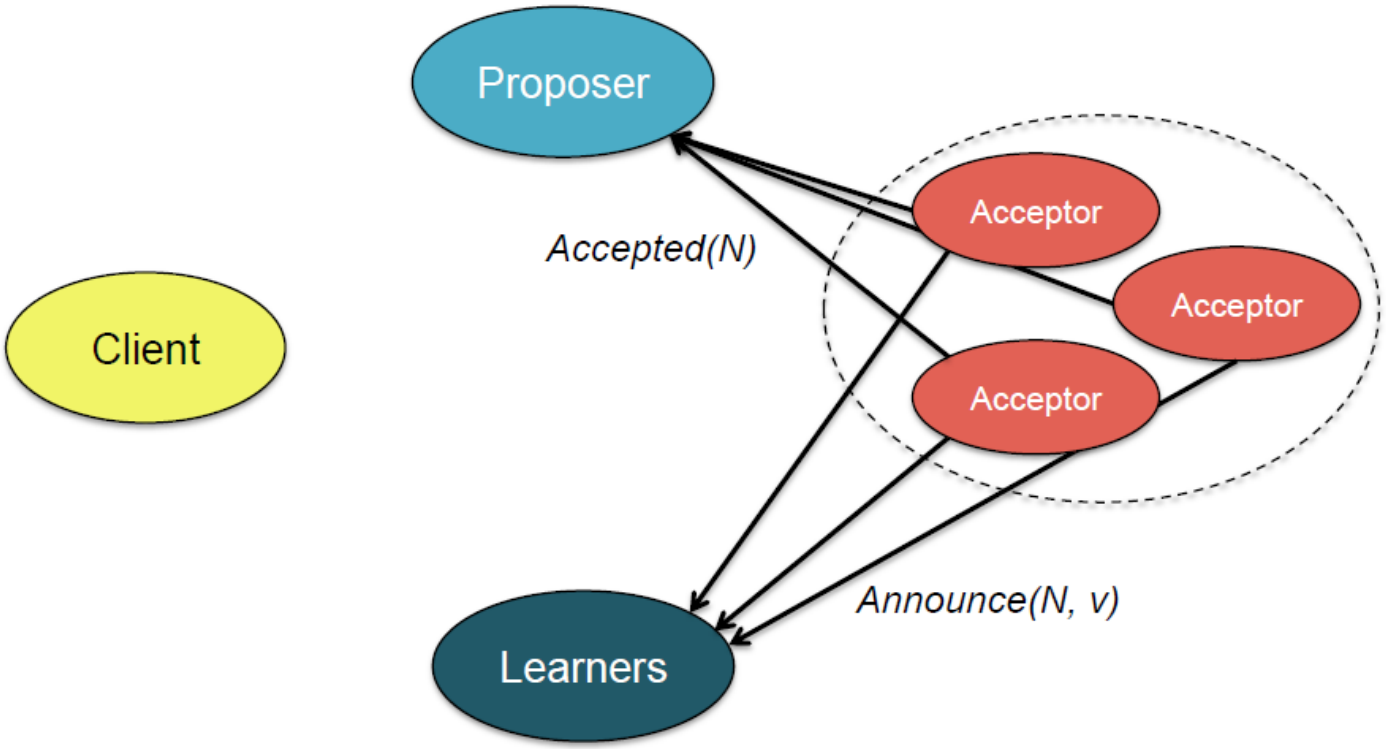
Phase 2a - ACCEPT

- Proposer: αν ο proposer λάβει υπόσχεση από την πλειονότητα
- Στέλνει *accept στο quorum με την επιλεγμένη τιμή v*
  - η τιμή που πρότεινε αρχικά, αν οι acceptors δεν είχαν κάνει accept σε καμία πρόταση στο παρελθόν
  - Η τιμή v για το μεγαλύτερο N που έλαβε από το promise των acceptors



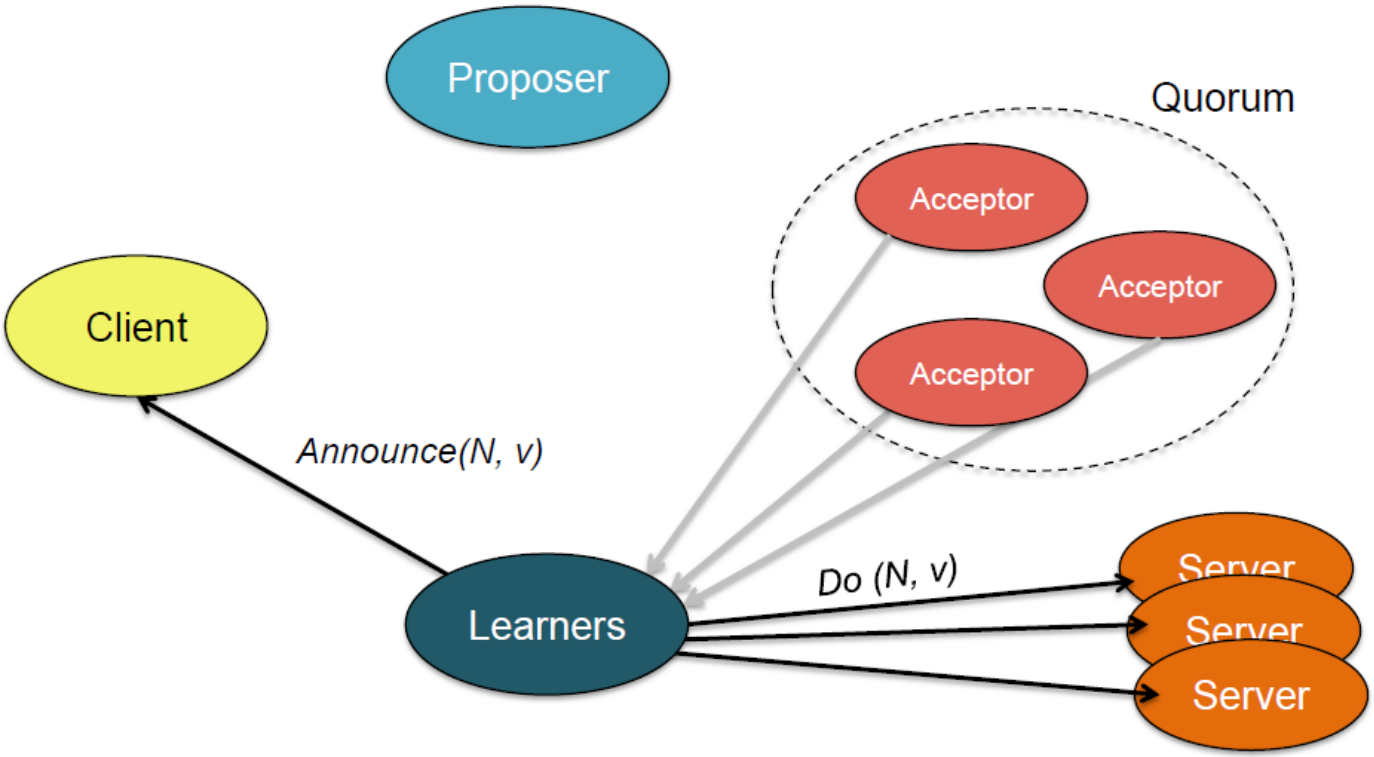
Phase 2b - ANNOUNCE

- **Acceptor: Αν η υπόσχεση ισχύει ακόμα, ανακοινώνει την τιμή *v***
- Στέλνει μήνυμα ***Accepted στον Proposer***
- Αλλιώς αγνοεί το μήνυμα (ή στέλνει *NACK)*

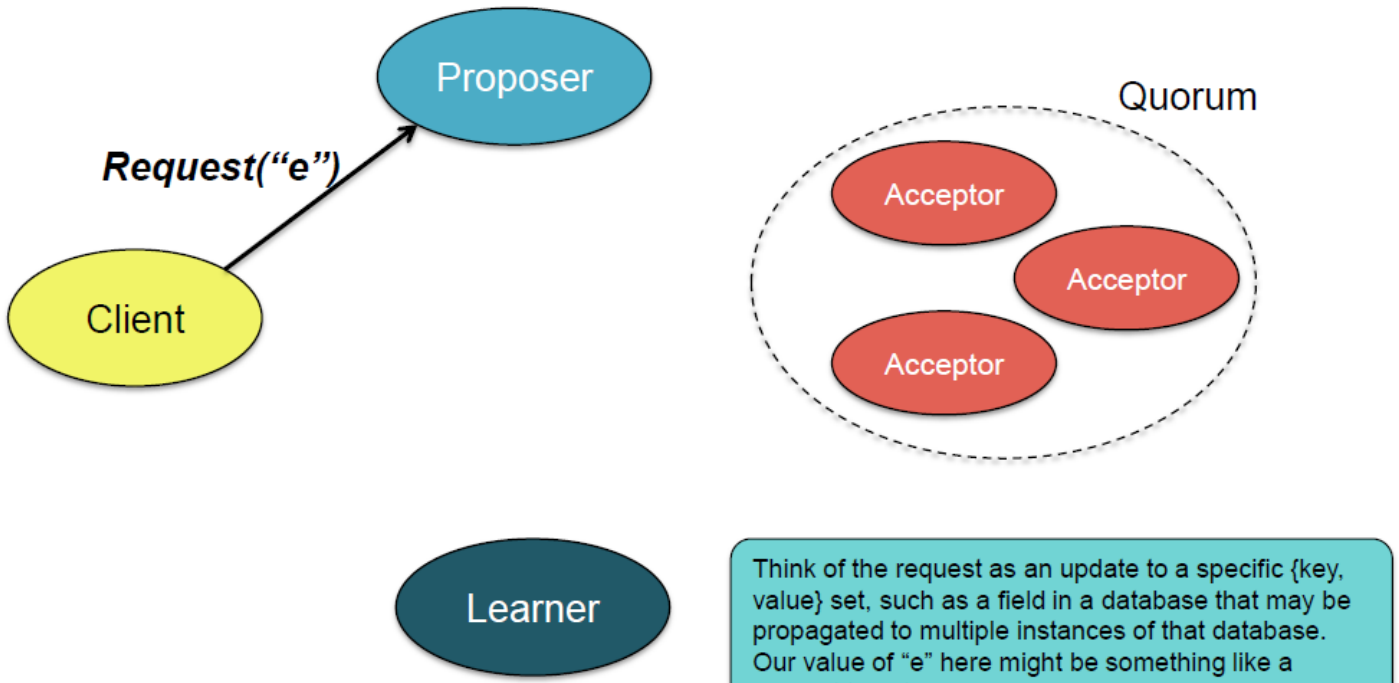


Phase 3

- **Learner: Απαντά στον client και κάνει τη δουλειά που πρέπει**

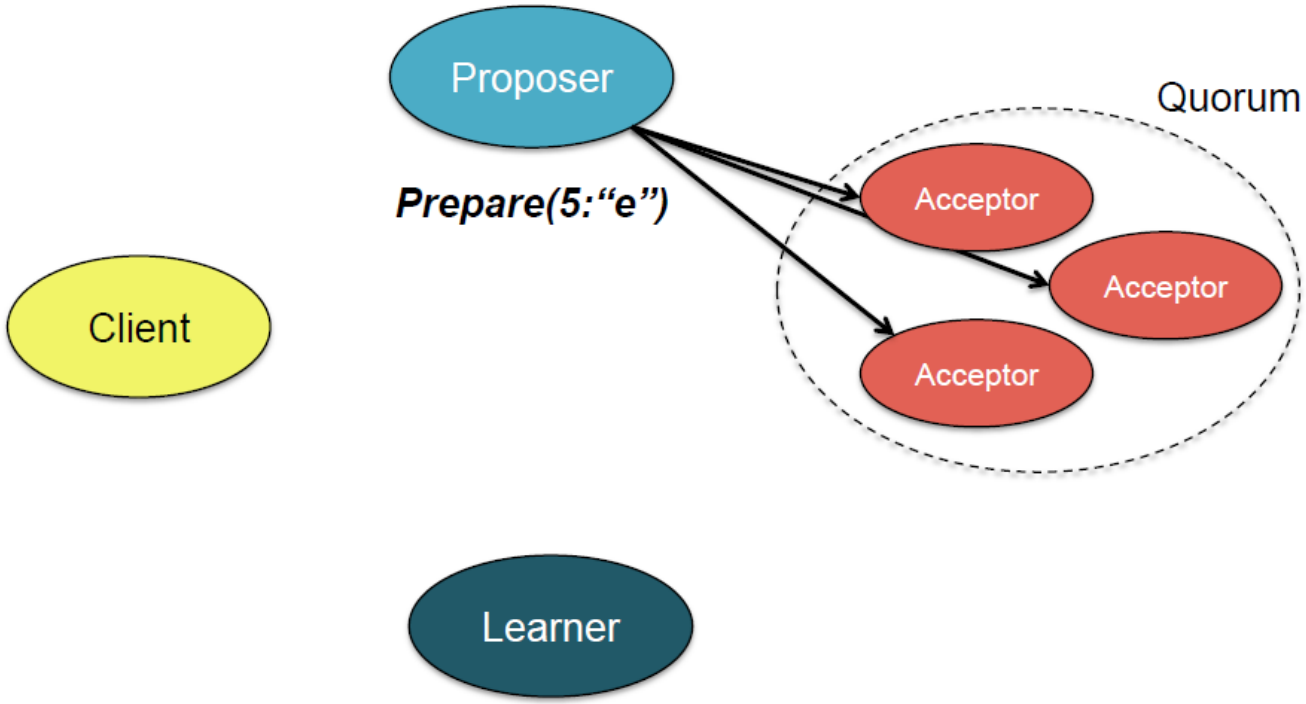


Παράδειγμα 1: phase 0



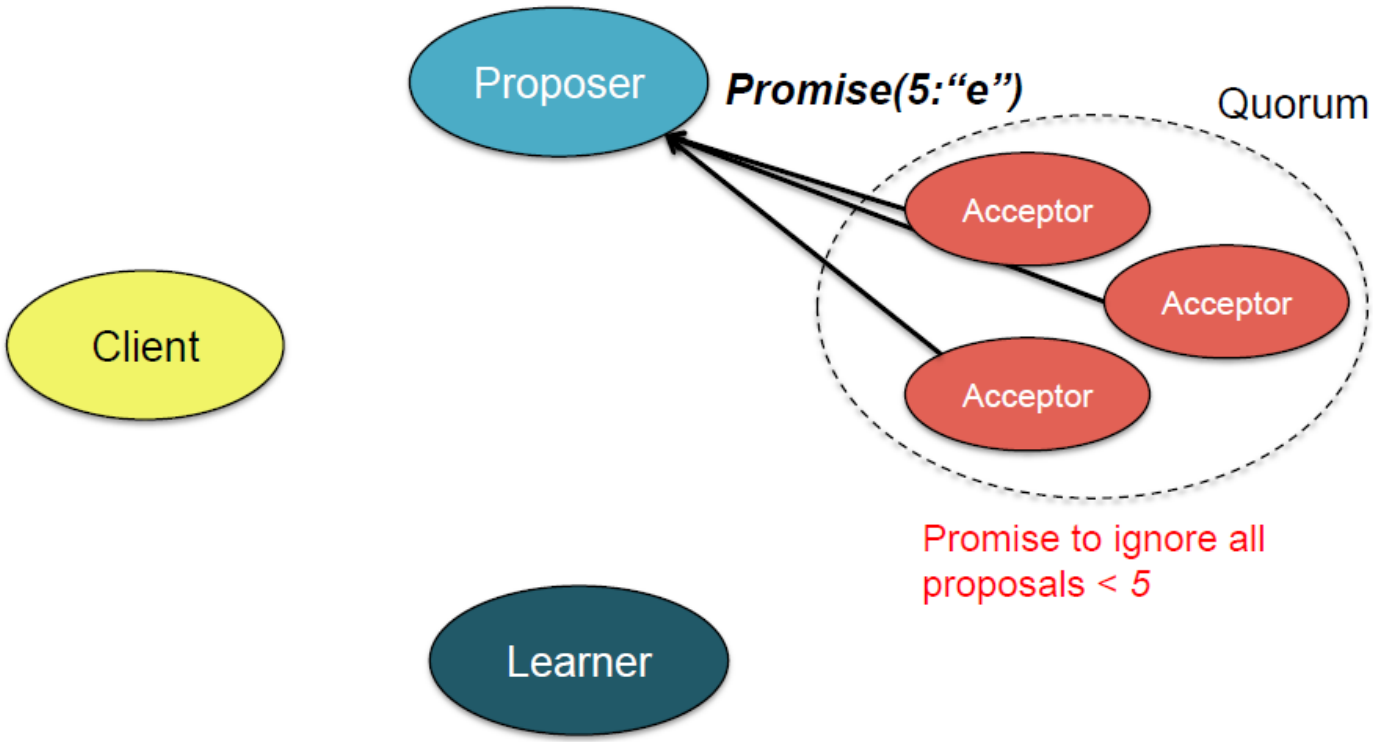
Phase 1a

- **Proposer:** διαλέγει id 5



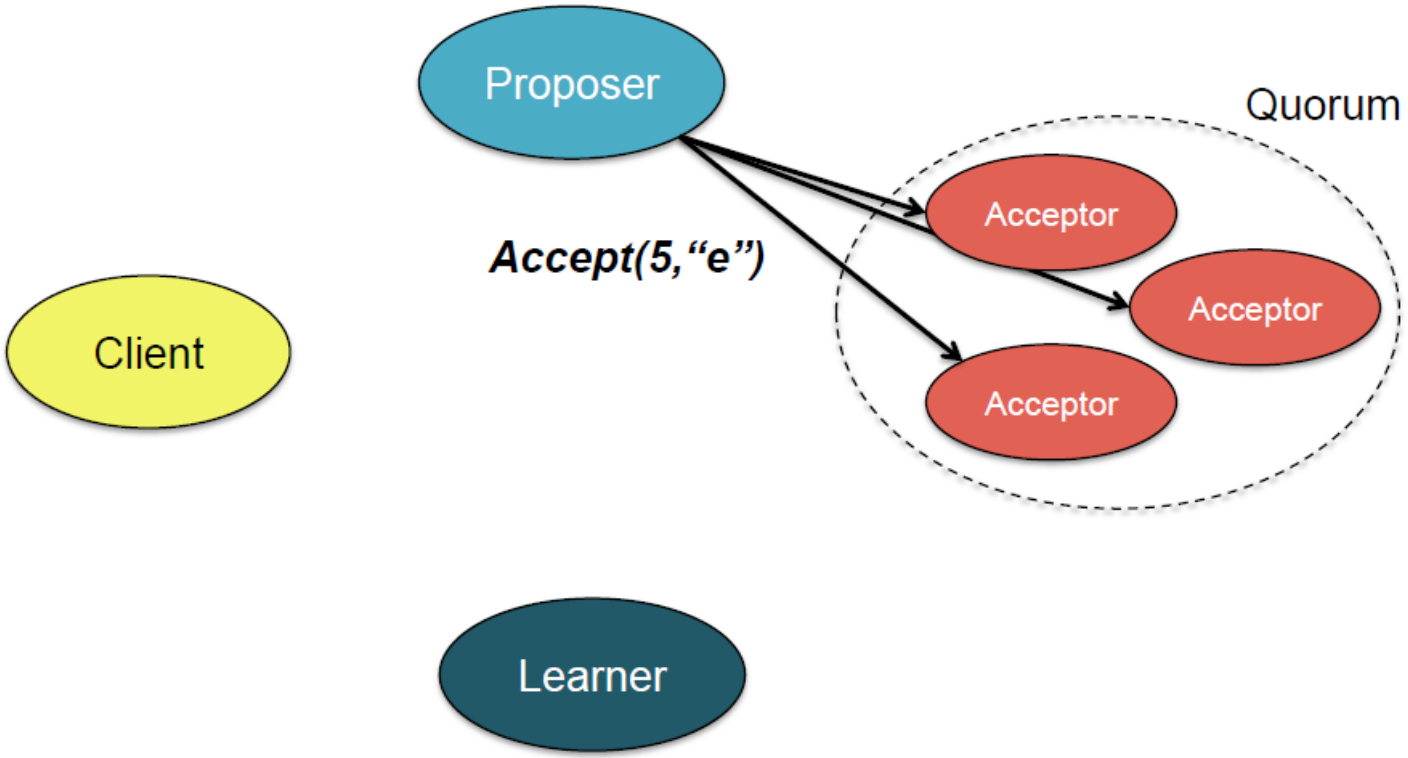
Phase 1b

- **Acceptor:** Αν το 5 είναι το μεγαλύτερο id που έχει δει, στέλνει υπόσχεση να μη δεχτεί μικρότερα ids



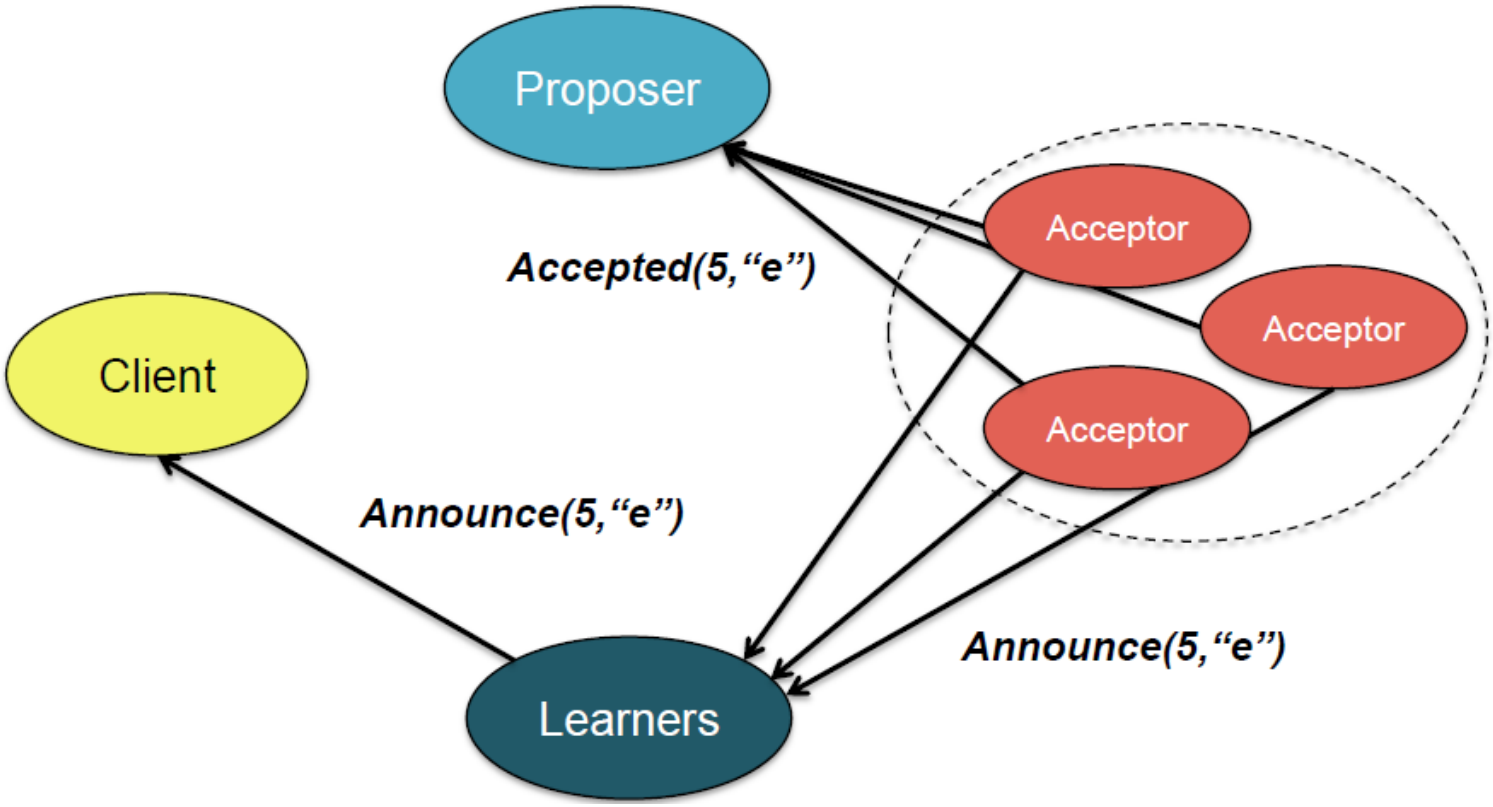
Phase 2a

- **Proposer:** λαμβάνει υπόσχεση από την πλειονότητα των acceptors
- Πρέπει να δεχτεί αυτό το <id, value>

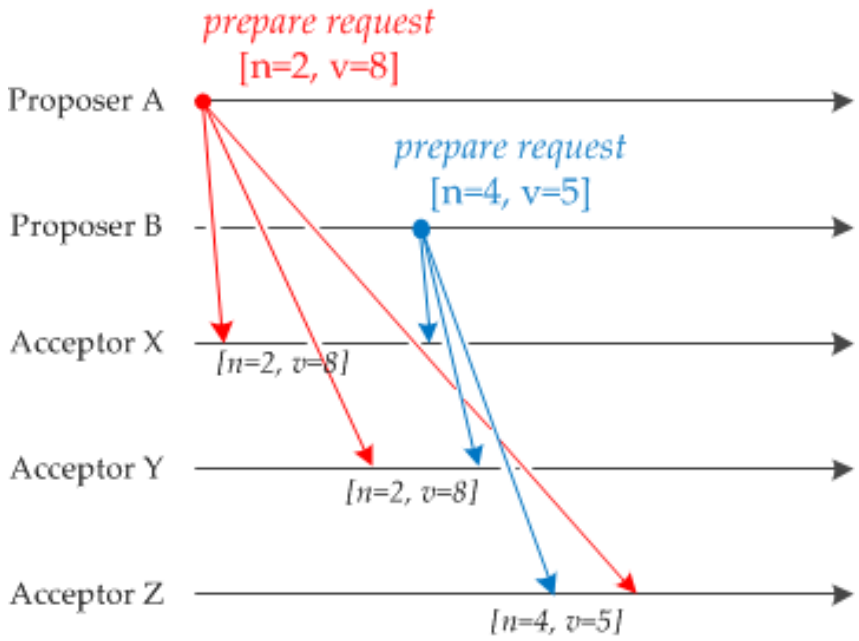


Phase 2b

- **Acceptor:** ανακοινώνουν απόφαση

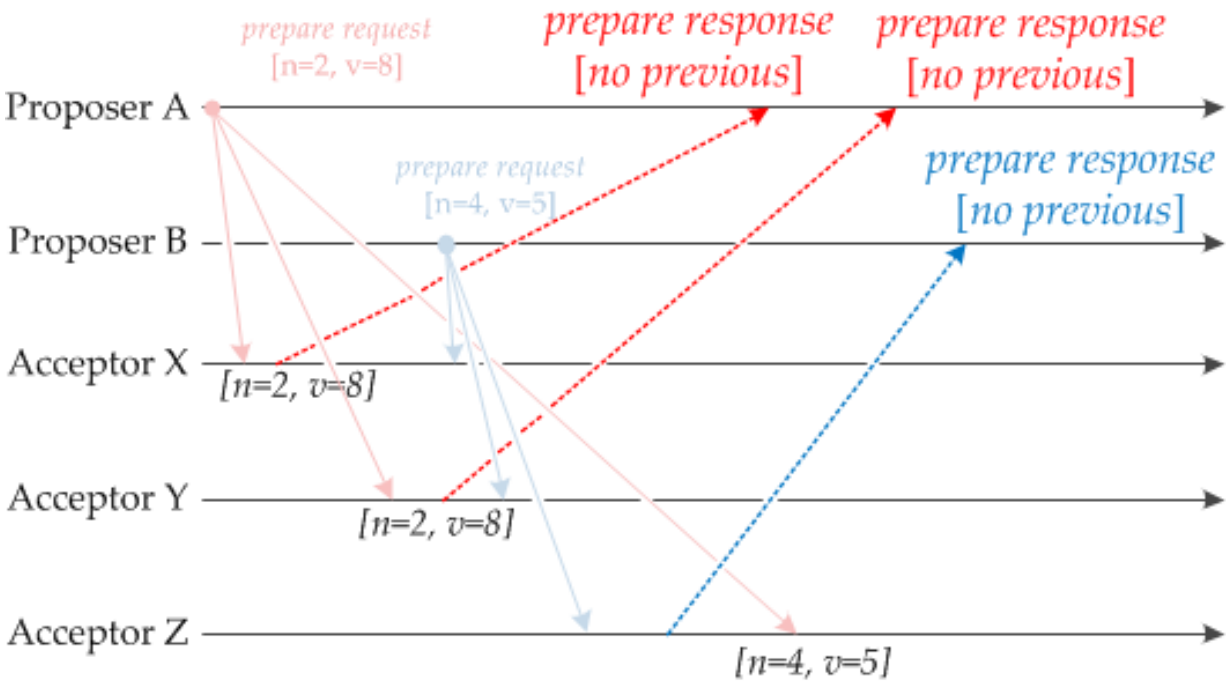


Παράδειγμα 2 – Phase 1a



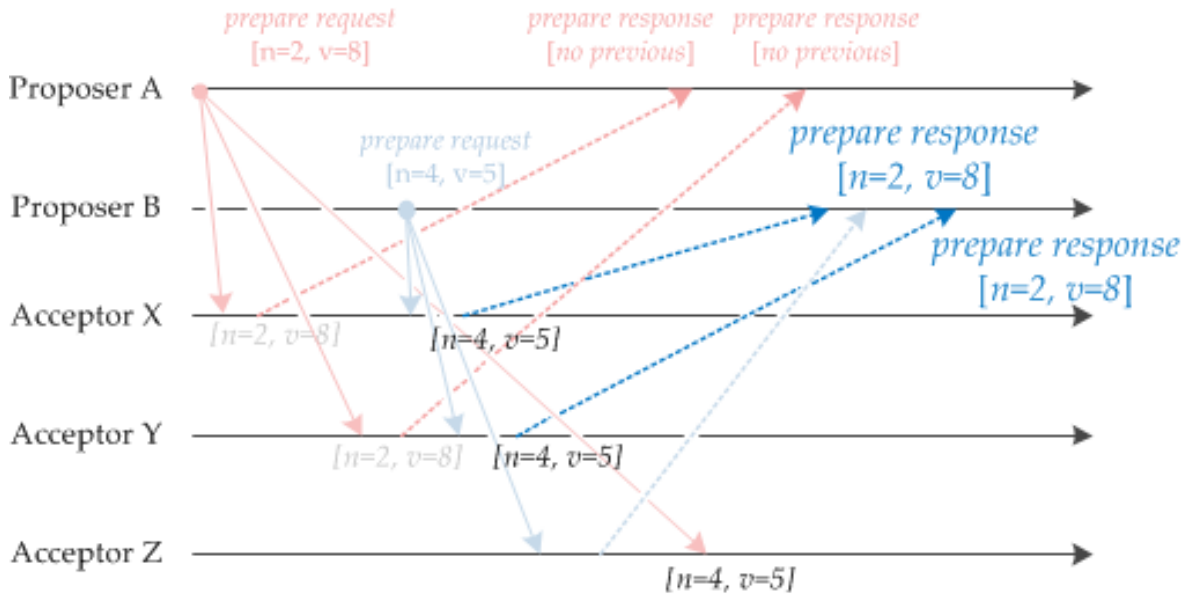
Proposers A and B each send prepare requests to every acceptor. In this example proposer A's request reaches acceptors X and Y first, and proposer B's request reaches acceptor Z first.

Phase 1b



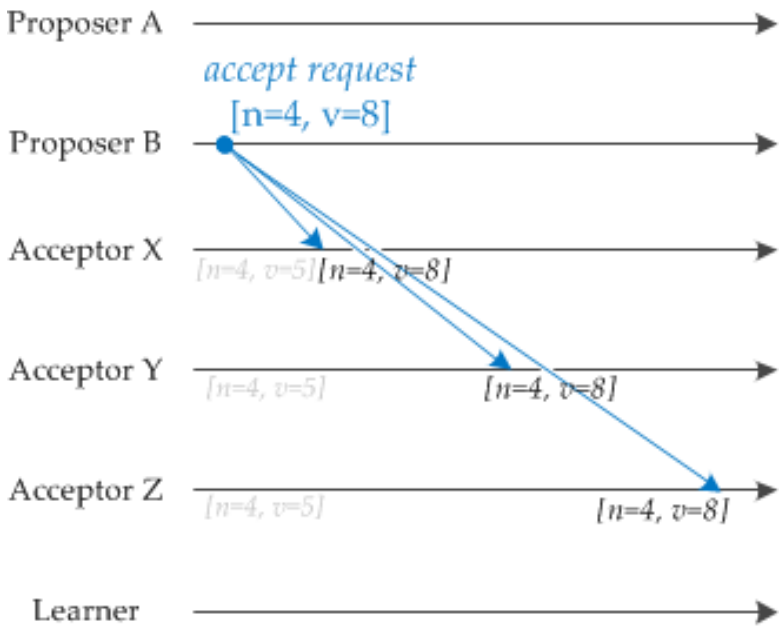
Each acceptor responds to the first prepare request message that it receives.

Phase 1b



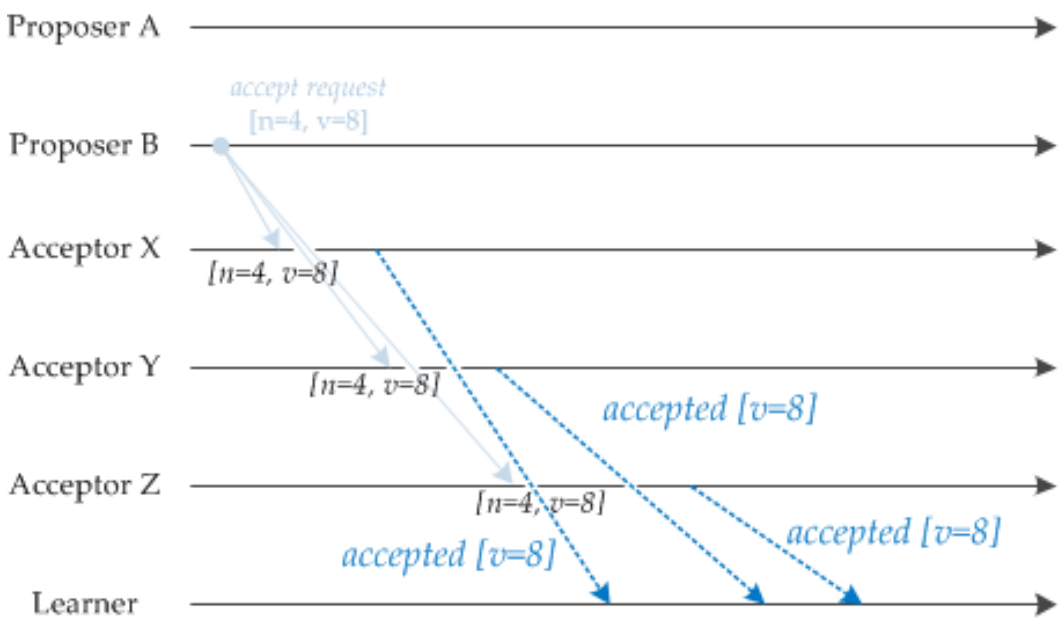
Acceptor Z ignores proposer A's request because it has already seen a higher numbered proposal (4 > 2). Acceptors X and Y respond to proposer B's request with the previous highest request that they acknowledged, and a promise to ignore any lower numbered proposals.

Phase 2a



Proposer B sends an accept request to each acceptor, with its previous proposal number (4), and the value of the highest numbered proposal it has seen (8, from [n=2, v=8])

Phase 2b



Συνέχισε να προσπαθείς

- Ένα proposal N μπορεί να αποτύχει γιατί
  - Ένας acceptor μπορεί να έχει δώσει υπόσχεση να αγνοήσει όλα τα proposals με id μικρότερο κάποιου M >N
  - Ένας proposer δε λαμβάνει quorum από απαντήσεις είτε στη φάση 1b είτε στη φάση 2b
- Ο αλγόριθμος τότε πρέπει να ξανα-ξεκινήσει με μεγαλύτερο proposal id

Επιμύθιο

- Το Paxos μας επιτρέπει να διασφαλίσουμε συνεπή ολική διάταξη σε σύνολο από events
  - Events = εντολές, ενέργειες, ενημερώσεις κατάστασης
- Κάθε κόμβος θα έχει την τελευταία κατάσταση ή κάποια προηγούμενη εκδοχή της
- Χρησιμοποιείται σε:
  - Cassandra lightweight transactions
  - Google Chubby lock manager / name server
  - Google Spanner, Megastore
  - Microsoft Autopilot cluster management service from Bing
  - VMware NSX Controller
  - Amazon Web Services

Σύνοψη του Paxos

- Για να κάνεις αλλαγή στο σύστημα:
  - Λες στον *proposer το event που θες να προσθέσεις*
    - *Αυτά τα αιτήματα μπορεί να συμβούν ταυτόχρονα*
    - *Leader = εκλεγμένος proposer. Δεν είναι απαραίτητος για τη λειτουργία του Paxos, αλλά είναι βελτιστοποίηση για διασφάλιση μοναδικής πηγής από αύξοντες αριθμούς στα proposals*
  - Ο proposer επιλέγει το επόμενο event ID και ζητά από τους acceptors να το φυλάξουν
    - Αν οποιοσδήποτε acceptor έχει δει μεγαλύτερο event ID, απορρίπτει την πρόταση και επιστρέφει το event ID αυτό
    - Ο proposer προσπαθεί ξανά με μεγαλύτερο event ID
  - Όταν η πλειονότητα των **acceptors αποδεχτούν το proposal, τα event στέλνονται στους** learners για να αρχίσουν να ενεργούν (π.χ., να ενημερώσουν την κατάσταση του συστήματος)
    - Fault tolerant: χρειαζόμαστε *2k+1 servers για k fault tolerance*