Αντίγραφα

Κατανεμημένα Συστήματα

2019-2020

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

Replication

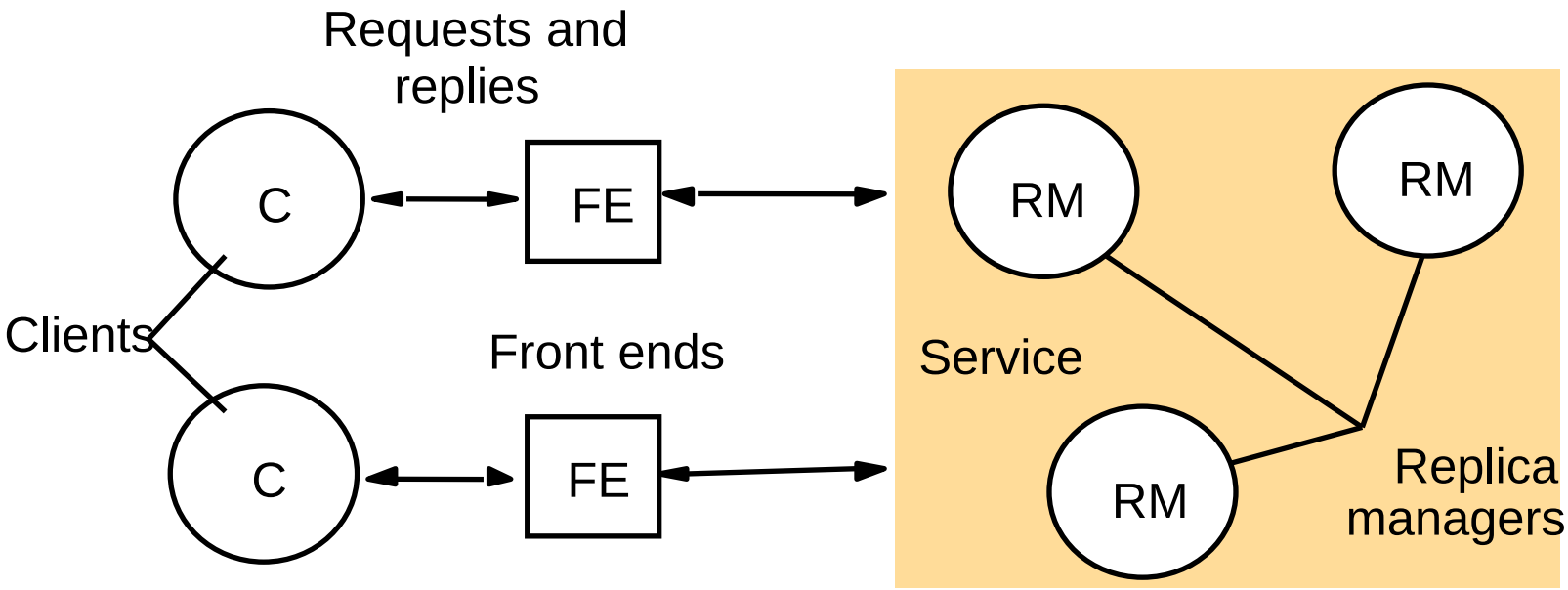
- Διατήρηση αντιγράφων δεδομένων σε πολλαπλούς υπολογιστές
- Γιατί;
  - Αυξημένη διαθεσιμότητα (availability) όταν οι servers αποτυγχάνουν
    - n servers με πιθανότητα αποτυχίας p

      Availability = 1 – prob (all servers fail) = $1 - p^n$

      π.χ. αν p = 5% τότε για έναν server: availability = 95%

      για 3 servers: availability = 99.875%
  - Ανοχή σε σφάλματα (fault tolerance)
    - Σωστή συμπεριφορά ακόμα κι αν f από τους f+1 servers αποτύχουν
  - Εξισορρόπηση φόρτου και επίδοση
    - Π.χ. πολλοί servers ανατίθενται στο ίδιο DNS name, οπότε κάθε φορά το DNS lookup επιστρέφει διαφορετικό IP με round-robin τρόπο

Στόχοι



- Διαφάνεια
  - Ο χρήστης δεν ξέρει ότι υπάρχουν πολλά αντίγραφα
- Συνέπεια
  - Τα δεδομένα είναι τα ίδια σε όλα τα αντίγραφα (ή τείνουν στο να γίνουν τα ίδια)

Replica Managers (RM)

- Διαχειρίζονται τα αντίγραφα και εκτελούν λειτουργίες πάνω τους (ατομικά)
- Είναι υπεύθυνοι για
  - **Συμφωνία**: Οι RMs προσπαθούν να έρθουν σε συμφωνία για το αποτέλεσμα ενός αιτήματος

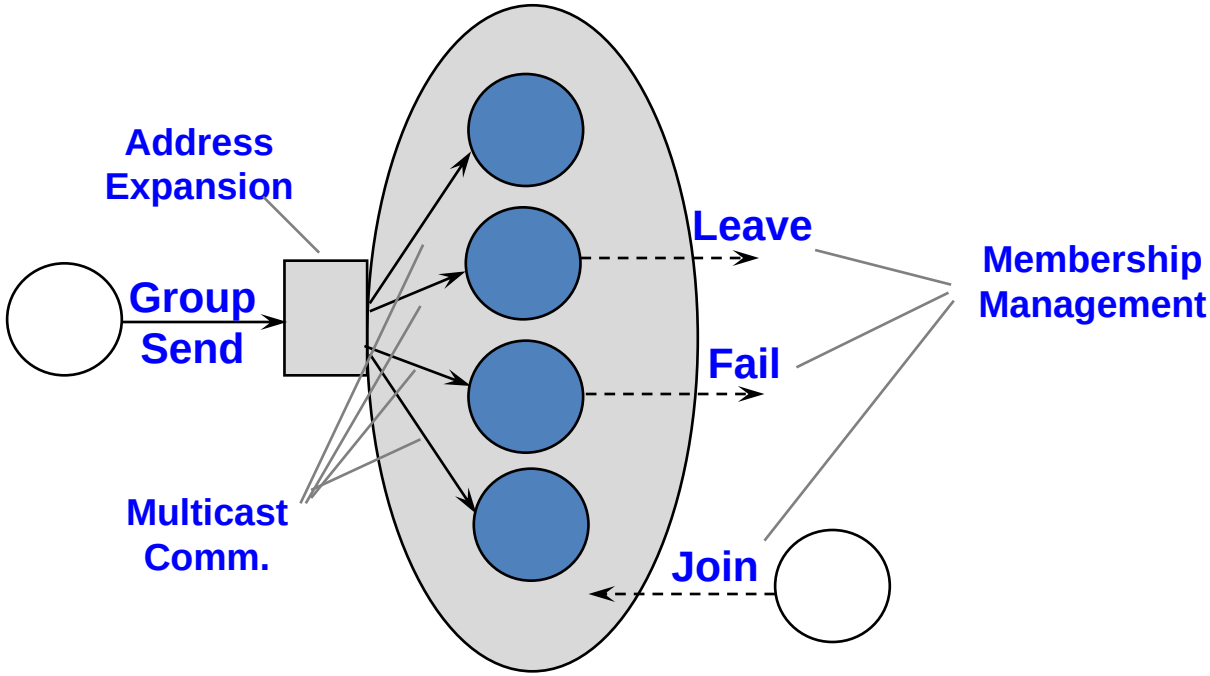
    Π.χ., με two phase commit: αν επιτύχει, το αποτέλεσμα γίνεται μόνιμο
  - **Απόκριση:** Τουλάχιστον ένας RM πρέπει να απαντήσει στον χρήστη

    Η πρώτη απάντηση που θα φτάσει αρκεί, αφού όλοι οι RMs θα επιστρέψουν την ίδια απάντηση

Replica Managers (RM)

- Ένας τρόπος να παρέχουν (αυστηρή) συνέπεια
  - Ξεκινούν με την ίδια αρχική κατάσταση
  - Συμφωνούν στην διάταξη των reads/writes και στο πότε ένα write γίνεται ορατό
  - Εκτελούν τις εντολές σε όλα τα replicas
- Κάθε RM είναι ένα *replicated state machine*
  - «Πολλά αντίγραφα της ίδιας μηχανής καταστάσεων που ξεκινούν από την κατάσταση Start και δέχονται τις ίδιες εισόδους με την ίδια σειρά θα καταλήξουν στην ίδια τελική κατάσταση έχοντας παράγει τα ίδια αποτελέσματα." [Wikipedia]
- Σας θυμίζει κάτι; Τι είδους επικοινωνία θα χρησιμοποιήσουμε;
  - reliable, ordered multicast

Ομαδική επικοινωνία



- Μέλος = διεργασία (ένας RM)
- Στατικές ομάδες: Οι ομάδες είναι προκαθορισμένες
- Δυναμικές ομάδες: Τα μέλη έρχονται και φεύγουν

Flashback: multicast

- *Ακεραιότητα* (integrity): Μια σωστή (χωρίς σφάλματα) διεργασία *p* παραδίδει ένα μήνυμα *m* το πολύ μια φορά
  - Σωστή: Τηρεί το πρωτόκολλο και είναι ζωντανή

- *Συμφωνία* (agreement): Αν μια σωστή διεργασία παραδώσει μήνυμα *m*, τότε όλες οι υπόλοιπες σωστές διεργασίας στην ομάδα group(*m*) θα παραδώσουν τελικά το *m*
  - «όλα ή τίποτα»

- *Ισχύς* (validity): Αν μια σωστή διεργασία στείλει μήνυμα *m*, τότε θα παραδώσει και η ίδια το m τελικά

Multicast σε δυναμικές ομάδες

- Πώς ορίζουμε κάτι αντίστοιχο του reliable multicast σε δυναμική ομάδα;
- Προσέγγιση
  - Διασφαλίζουμε ότι όλες οι διεργασίες γνωρίζουν την ίδια λίστα μελών
  - Διασφαλίζουμε ότι το reliable multicast γίνεται όσο η λίστα παραμένει ίδια

Όψεις (views)

- Group view = τρέχουσα λίστα των μελών της ομάδας
  - Κάθε μέλος έχει το δικό του τοπικό view
- Ένα view $V_p(g)$ διεργασίας p είναι η αντίληψή του για το group
  - Παράδειγμα:

    $V_{p.0}(g) = \{p\}$, $V_{p.1}(g) = \{p, q\}$, $V_{p.2}(g) = \{p, q, r\}$, $V_{p.3}(g) = \{p,r\}$

- Ένα νέο group view διαδίδεται σε όλο το group με κάθε είσοδο ή έξοδο μέλους
  - Ένα μέλος που ανιχνεύει αποτυχία άλλου μέλους στέλνει με reliable multicast ένα μήνυμα "view change" (απαιτεί ολική διάταξη για το multicast)
  - Ο στόχος: Η σειρά λήψης των views να είναι ίδια για όλα τα μέλη

Flashback Ολική διάταξη

- Κάθε διεργασία παραδίδει όλα τα μηνύματα με την ίδια σειρά
- Αν μια σωστή διεργασία παραδώσει το μήνυμα *m* πριν το *m'* (ανεξάρτητα από τους αποστολείς), τότε οποιαδήποτε άλλη σωστή διεργασία που παραδίδει το *m'* θα έχει παραδώσει ήδη το m
- Παράδειγμα
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8
- Ολική διάταξη
  - P1: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P2: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P3: m7, m1, m2, m4, m5, m3, m6, m0, m8

Όψεις

- Ένα γεγονός συμβαίνει στο view $v_{p,i}(g)$ αν η διεργασία p έχει παραδώσει το $v_{p,i}(g)$ αλλά όχι ακόμα το $v_{p,i+1}(g)$.
- Τα μηνύματα που αποστέλλονται κατά το view i *πρέπει να παραδίδονται στο ίδιο view* σε όλα τα μέλη της ομάδας
- Απαιτήσεις για την παράδοση του view
  - Σειρά: Αν η p παραδώσει $v_i(g)$ και μετά $v_{i+1}(g)$, καμία άλλη διεργασία q δεν παραδίδει $v_{i+1}(g)$ πριν το $v_i(g)$.
  - ακεραιότητα: Αν η p παραδώσει $v_i(g)$, τότε η p ανήκει στο view
  - Non-triviality: Αν μια διεργασία q γίνει μέλος ενός group και είναι προσβάσιμη από την p, τότε κάποια στιγμή (eventually) η q θα είναι πάντα παρούσα σε όλα τα views που παραδίδονται στο p.
    - Εξαίρεση: partitioning ενός group

View synchronous επικοινωνία

- View Synchronous Communication = Group Membership Service + Reliable multicast
- "What happens in the view, stays in the view"

Εγγυήσεις

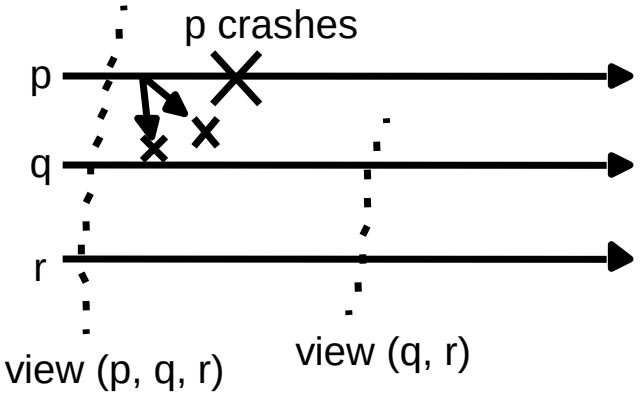
- Ακεραιότητα: Αν η p παρέδωσε το μήνυμα m δεν θα το παραδώσει ξανά. Επιπλέον η p και η διεργασία αποστολέας του m βρίσκονται στο ίδιο view στο οποίο η p παρέδωσε το m.

- Ισχύς: Σωστές διεργασίες πάντα παραδίδουν όλα τα μηνύματα. Αν η p παραδώσει το μήνυμα m κατά το view v(g), και κάποια διεργασία $q \in v(g)$ δεν παραδώσει το m κατά το ίδιο v(g), τότε το επόμενο view v'(g) που θα παραδοθεί στο p δεν θα περιλαμβάνει το q.

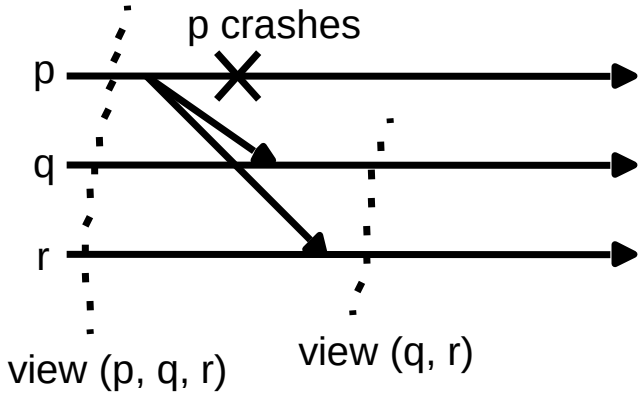
- Συμφωνία: Σωστές διεργασίες παραδίδουν την ίδια σειρά από views, και το ίδιο σύνολο μηνυμάτων σε κάθε view.
  - Αν η p παραδώσει το m κατά το V και μετά παραδώσει το V', τότε όλες οι διεργασίες στο $V \cap V'$ θα παραδώσουν το m κατά το view V

Παραδείγματα

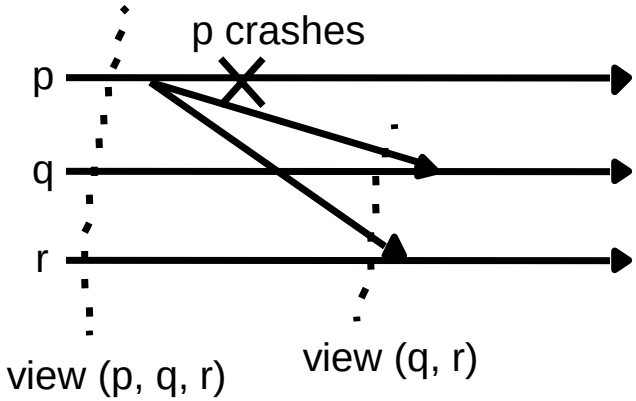
a (allowed).



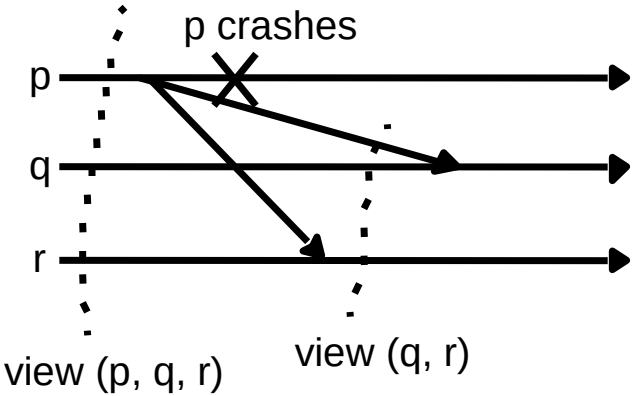
b (allowed).



c (disallowed).



d (disallowed).



Μεταφορά κατάστασης

- Όταν μια νέα διεργασία γίνεται μέλος ενός group, μπορεί να χρειαστεί μεταφορά κατάστασης για επικαιροποίηση
  - Η κατάσταση μπορεί να είναι η λίστα μηνυμάτων που έχουν παραδοθεί μέχρι τώρα ή η λίστα από τρέχουσες τιμές αντικειμένων (π.χ., μια βάση δεδομένων)

Συνέπεια (consistency)

- Μια υπηρεσία αποθήκευσης εξυπηρετεί αιτήματα read/write
- Πώς θα εξασφαλίσουμε συνέπεια σε όλα τα αντίγραφα ;
- Από την πλευρά του χρήστη, πότε ξέρουμε αν ένα αντικείμενο έχει νέα τιμή;
  - Εξαρτάται από το πότε τα writes γίνονται ορατά από τα reads
- Μπορούμε να παρέχουμε διάφορες εγγυήσεις:
  - Σειριοποιησιμότητα (Linearizability)
  - Ακολουθιακή συνέπεια (Sequential consistency)
  - Αιτιώδης συνέπεια (Causal consistency)

Παράδειγμα

Client 1	Client 2
$setBalance_B(x, 1)$	
$setBalance_A(y, 2)$	
	$getBalance_A(y) \rightarrow 2$
	$getBalance_A(x) \rightarrow 0$

Ο Replica Manager B πεθαίνει αμέσως μετά το $setBalance_B(x, 1)$

Linearizability

- Η πιο αυστηρή μορφή συνέπειας
- Linearizability
  - Ένα read επιστρέφει την πιο πρόσφατη τιμή του write
- Απλό για σύστημα με έναν εξυπηρετητή read/write.
  - Πώς;
- Τι συμβαίνει όταν έχουμε πολλούς servers;
  - Πολλοί χρήστες αλληλεπιδρούν με πολλούς servers, οι οποίοι διατηρούν replicas
  - Ο χρήστης C1 γράφει στον server S1 τη στιγμή t, ο χρήστης C2 διαβάζει από τον server S2 τη στιγμή t+1. Ο S2 θα πρέπει να επιστρέψει αυτό που έγραψε ο C1

Linearizability

- Ποια είναι η πρώτη απαίτηση για τη διατήρηση replicas;
  - Θα πρέπει το σύστημα να λειτουργεί έτσι ώστε να φαίνεται στο χρήστη ότι υπάρχει μόνο ένα αντίγραφο κάθε δεδομένου
- Πώς;
  - Αν έχουμε έναν χρήστη κι έναν server:
  - Δεδομένου ενός συνόλου από operations από τον χρήστη υπάρχει μια διάταξή τους που να εξηγεί ποιες τιμές γράφτηκαν και ποιες τιμές διαβάστηκαν σε ένα μοναδικό αντίγραφο
  - Πώς μεταφράζεται αυτό σε κατανεμημένο περιβάλλον;
- Single copy semantics
  - Υπάρχει μια μοναδική ένθεση (interleaving) από operations που εξηγεί τα αποτελέσματα των read/write λειτουργιών όλων των χρηστών σαν να έγιναν σε ένα αντίγραφο

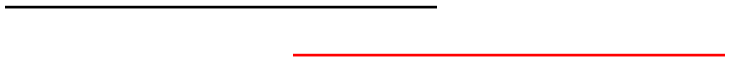
Linearizability

- Linearizability
  - Single-copy semantics
  - Ένα read επιστρέφει το πιο πρόσφατο write
- Real-time
  - Πάντα διαβάζεις αυτό που γράφτηκε αμέσως πριν
  - Ένα write πρέπει να είναι ορατό στο επόμενο read αμέσως
- Πρόβλημα: οι λειτουργίες read και write παίρνουν χρόνο

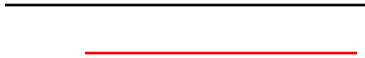
Θέματα linearizability

- Clear-cut (black---write & red---read)

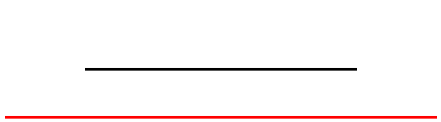
- Not-so-clear-cut (parallel)
  - Case 1:



  - Case 2:



  - Case 3:



Θέματα linearizability

- Μια λειτουργία παίρνει χρόνο για να ολοκληρωθεί
  - Π.χ, ένα read R ξεκινάει τη χρονική στιγμή X και τελειώνει την Y
- Μια τιμή ενός write γίνεται ορατό κάποια στιγμή κατά τη διάρκεια του operation.
  - Π.χ., ένα write W ξεκινά τη στιγμή X και τελειώνει την Y ms. Τη στιγμή Z (X < Z < Y), η τιμή γράφεται στον δίσκο και γίνεται ορατή
- Τι θα κάναμε λογικά;
  - Αν το W τελειώσει τη στιγμή X, το R ξεκινήσει τη στιγμή Y και X < Y, τότε το R θα πρέπει να διαβάσει αυτό που έγραψε το W
  - Αν το R συμπέσει με το W, τότε μπορεί να διαβάσει ή την προηγούμενη τιμή ή την τιμή που γράφτηκε από το W

Θέματα linearizability

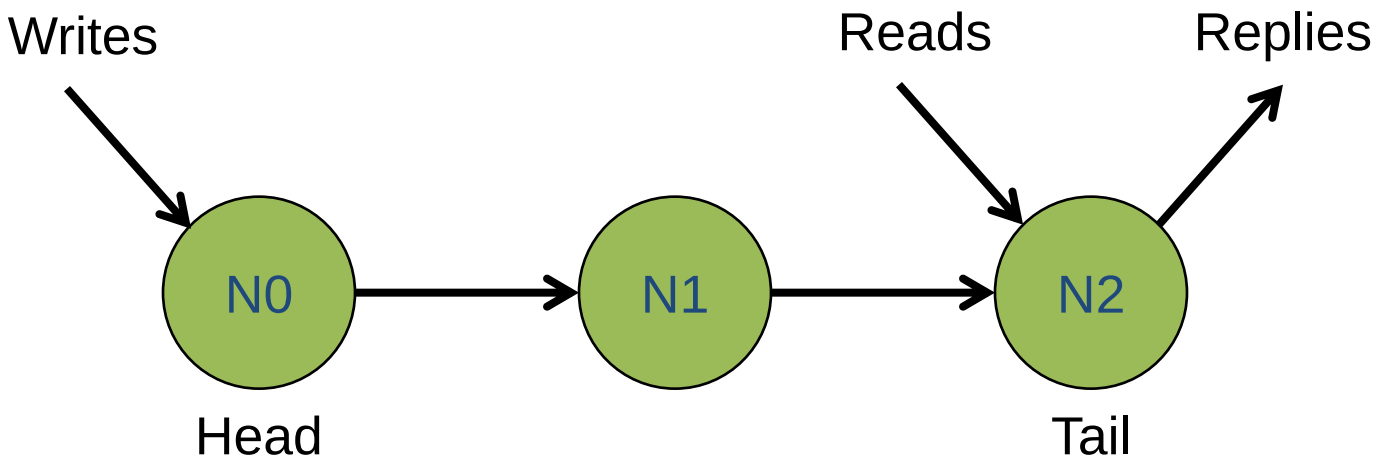
- Εγγύηση
- Χαλαρή εγγύηση σε αλληλοεπικάλυψη
- Case 1
- Case 2
- Case 3

Τελικά

- (Ορισμός από βιβλίο) Μια υπηρεσία διαμοιραζόμενων αντιγράφων είναι σειριοποιήσιμη αν για κάθε εκτέλεση υπάρχει κάποιο interleaving των operations από όλους τους clients έτσι ώστε
  - καλύπτει τον ορισμό ενός μοναδικού σωστού αντιγράφου
  - είναι συνεπές με τον πραγματικό χρόνο που έγινε κάθε operation κατά τη διάρκεια της εκτέλεσης
- Στόχος: Οποιοσδήποτε χρήστης οποιαδήποτε στιγμή βλέπει ένα αντίγραφο του αντικειμένου που είναι σωστό και συνεπές
- Η πιο ισχυρή μορφή συνέπειας

Chain Replication

- One technique to provide linearizability



Sequentially consistency

- Sequential consistency
  - Παρέχει τη συμπεριφορά μοναδικού αντιγράφου
  - Ένα read operation επιστρέφει το πιο πρόσφατο write
- Τι σημαίνει πιο πρόσφατο
  - για λειτουργίες στον ίδιο client: καθορίζεται από τον χρόνο (program order)
  - Για λειτουργίες σε πολλούς clients: Δεν καθορίζεται από τον χρόνο (μπορούν να αναδιαταχθούν)
  - Δηλαδή, πρέπει απλώς να διατηρήσουμε το program order κάθε client

Sequential consistency

- Για τον εξωτερικό παρατηρητή, το σύστημα παρέχει μια διάταξη των operations έτσι ώστε
  - Να δουλεύει σαν να είχε ένα μοναδικό αντίγραφο
  - Η διάταξη των operations από τον ίδιο client να διατηρείται
- Linearizability vs. sequential consistency
  - Με sequential consistency το σύστημα έχει την ελευθερία να κανονίσει πώς θα οργανώσει τα operations που προέρχονται από διαφορετικούς πελάτες αρκεί να διατηρείται η διάταξη των operations από τον ίδιο πελάτη
  - Με linearizability η ολική διάταξη για όλους τους πελάτες καθορίζεται από το χρόνο

Linearizability vs. Sequential Consistency

- Και τα 2 δίνουν την ψευδαίσθηση του μοναδικού αντιγράφου
  - Για έναν εξωτερικό παρατηρητή, το σύστημα συμπεριφέρεται (σχεδον) σαν να είχε ένα μόνο αντίγραφο
- Το linearizability ενδιαφέρεται για χρόνο
  - Η Κατερίνα γράφει στο facebook wall στις 11am.
  - Ο Γιάννης γράφει στο facebook wall στις 11:05am.
  - Όλοι θα δουν τις δημοσιεύσεις με αυτήν τη σειρά
- Η ακολουθιακή συνέπεια (sequential consistency) ενδιαφέρεται για τη διάταξη
  - Στο παραπάνω παράδειγμα δεν είναι απαραίτητο οι δημοσιεύσεις να εμφανιστούν με αυτήν τη σειρά (αλλά με την ίδια σειρά σε όλους τους clients)

Παραδείγματα

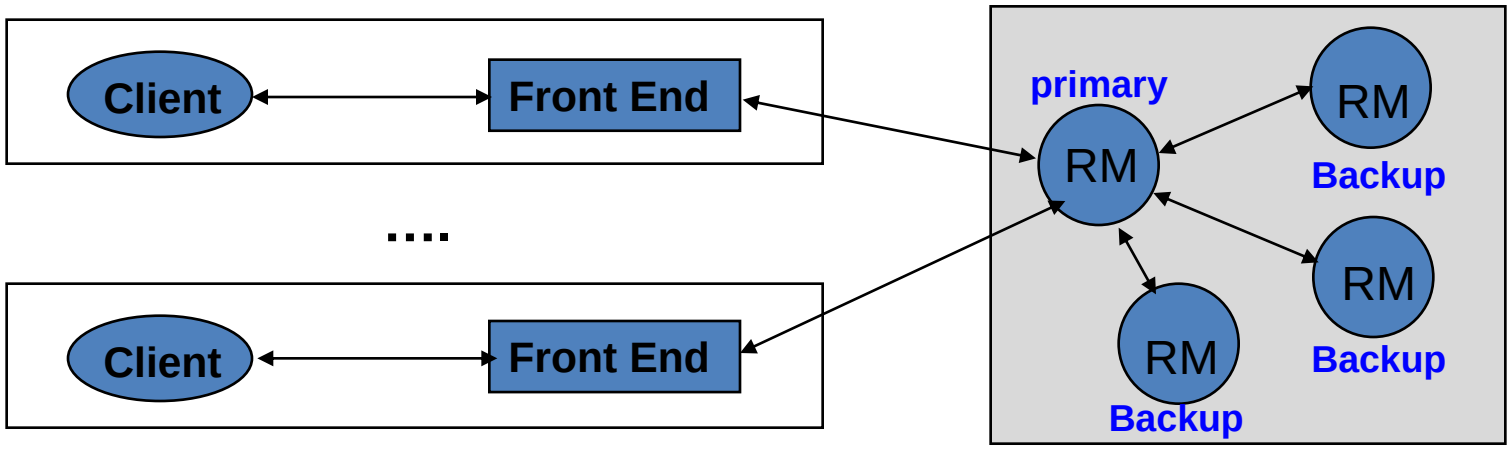
- Example 1 Legit
  - P1: a.write(A)
  - P2: a.write(B)
  - P3: a.read()->B a.read()->A
  - P4: a.read()->B a.read()->A

W2(x)b R3(x)b R4(x)b W1(x)a R3(x)a R4(x)a

- Example 2 not legit
  - P1: a.write(A)
  - P2: a.write(B)
  - P3: a.read()->B a.read()->A
  - P4: a.read()->A a.read()->B

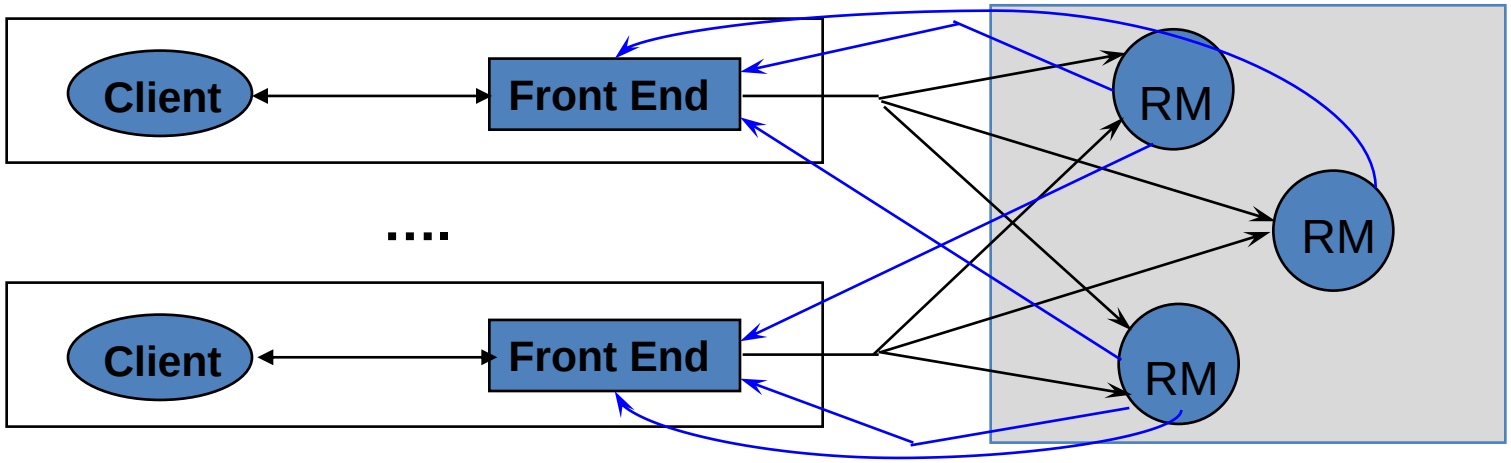
W1(x)a W2(x)b R3(x)b R4(x)a R3(x)a R4(x)b

Passive Replication



- Request : Το αίτημα γίνεται στον πρωτεύοντα RM και φέρει μοναδικό id
- Coordination: Ο πρωτεύων RM επεξεργάζεται τα αιτήματα ατομικά, με τη σειρά που τα λαμβάνει και ελέγχει το id για duplicates
- Execution: Ο πρωτεύων εκτελεί το αίτημα και αποθηκεύει την απάντηση
- Agreement: Αν πρόκειται για update, στέλνει την ενημερωμένη κατάσταση, την απάντηση και το id σε όλους τους backup RMs
- Response: Ο πρωτεύων στέλνει το αποτέλεσμα στον client

Active Replication



- Request : Το αίτημα φέρει μοναδικό id και στέλνεται με αξιόπιστο, ολικά διατεταγμένο multicast σε όλους.
- Coordination: Το πρωτόκολλο multicast διασφαλίζει ότι τα αιτήματα παραδίδονται σε όλους με την ίδια σειρά
- Execution: Κάθε αντίγραφο εκτελεί το αίτημα. (Όλοι οι RMs πρέπει να επιστρέψουν το ίδιο αποτέλεσμα)
- Agreement: Δε χρειάζεται κάποια συμφωνία χάρη στο multicast
- Response: Κάθε RM στέλνει απάντηση απευθείας στον πελάτη

3 ακόμα μορφές συνέπειας

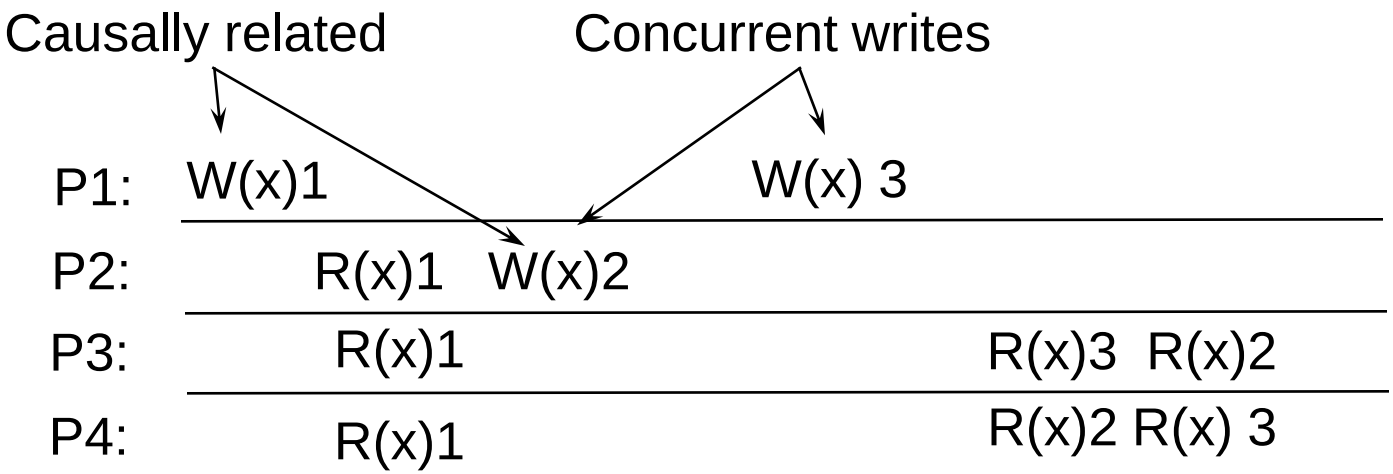
- Πιο χαλαρές από τις προηγούμενες
  - Δε μας νοιάζει καν να δίνεται η ψευδαίσθηση μοναδικού αντιγράφου
- Causal consistency
  - Μας ενδιαφέρει η διάταξη των writes που συνδέονται με αιτιώδη σχέση
- FIFO consistency
  - Μας ενδιαφέρει η διάταξη των writes που γίνονται στην ίδια διεργασία
- Eventual consistency
  - Αρκεί όλα τα αντίγραφα κάποια στιγμή να αποκτήσουν όλα την ίδια τιμή

Causal Consistency

- Όλες οι διεργασίες πρέπει να βλέπουν με την ίδια σειρά τα writes που συνδέονται με αιτιώδη σχέση. Διαφορετικές διεργασίες μπορούν να βλέπουν ταυτόχρονα writes με διαφορετική σειρά.
  - Πιο χαλαρό από sequential consistency

- Πώς εννοούμε την “αιτιώδη σχέση” ανάμεσα σε δύο writes?
  - Ένας client διαβάζει κάτι που έγραψε ένας άλλος client και μετά γράφει κάτι

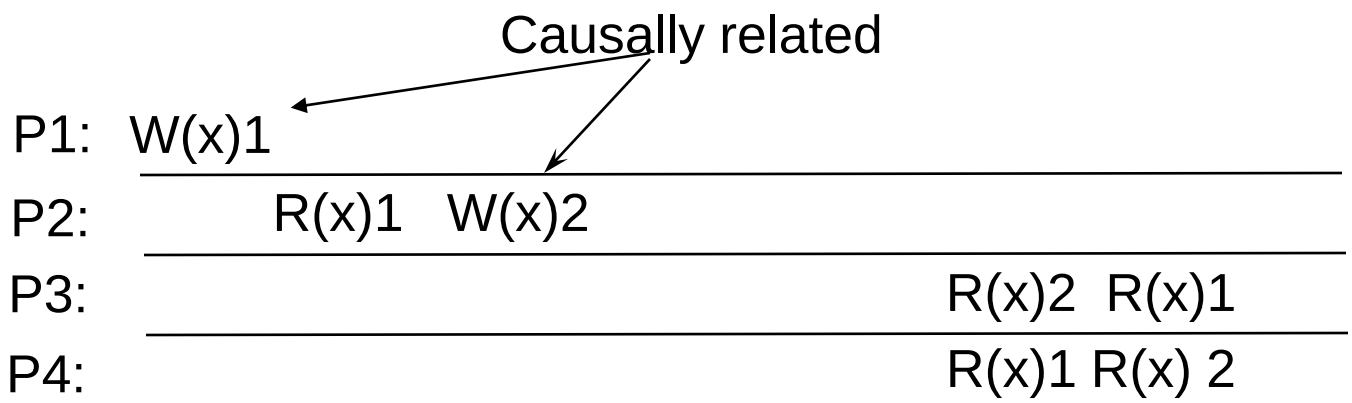
Παράδειγμα 1



This sequence obeys causal consistency

Παράδειγμα 2

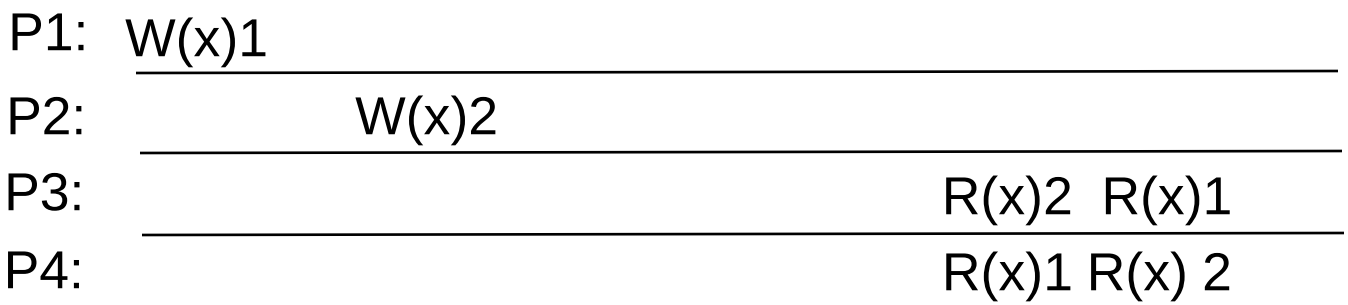
- Causally consistent?



- No!

Παράδειγμα 3

- Causally consistent?



- Yes!

FIFO consistency

- Writes of a single process are seen by all other processes in the order in which they were issued, but writes of different processes may be seen in a different order by different processes.

- In other words: There are no guarantees about the order in which different processes see writes, except that two or more writes of the same process must arrive in order (that is, all writes generated by different processes are concurrent).

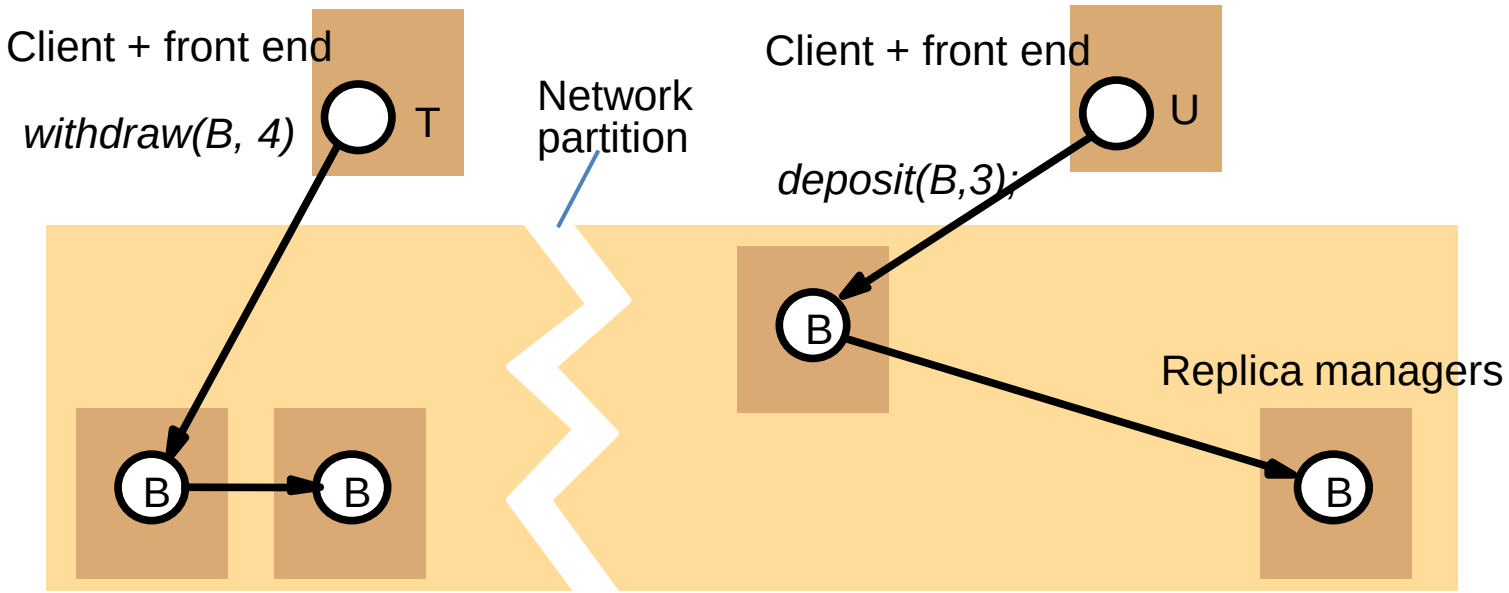
Παράδειγμα FIFO

P1: W(x)a						
P2:	R(x)a	W(x)b	W(x)c			
P3:				R(x)b	R(x)a	R(x)c
P4:				R(x)a	R(x)b	R(x)c

- A valid sequence of events of FIFO consistency
- Guarantee:
  - writes from a single source must arrive in order
  - no other guarantees. Easy to implement!

Consistency under network partition

- Το βασικό πρόβλημα εδώ είναι το network partition



Δίλημμα

- Όταν έχουμε ένα network partition:
- Για να κρατήσουμε τα αντίγραφα συνεπή, πρέπει να μπλοκάρουμε
  - Για έναν εξωτερικό παρατηρητή, το σύστημα είναι μη διαθέσιμο
- Αν συνεχίσουμε να εξυπηρετούμε αιτήματα από 2 partitions, τα αντίγραφα θα αρχίσουν να αποκλίνουν
  - Το σύστημα θα είναι διαθέσιμο, αλλά όχι συνεπές
- Το θεώρημα CAP εξηγεί το δίλημμα

Θεώρημα CAP

- Consistency
- Availability
- Partition tolerance

- Έχει αποδειχθεί ότι μόνο 2 από τα 3 μπορούν να συνυπάρχουν

- Άρα σε περίπτωση network partition πρέπει να διαλέξουμε ένα από τα 2

Διαχείριση του CAP

- Το πρόβλημα έγκειται στο Internet
  - Όταν το σύστημα περιλαμβάνει γεωγραφικά κατανεμημένες περιοχές, η περίπτωση του network partitioning είναι αναπόφευκτη
- Τότε πρέπει να θυσιάσει κανείς είτε availability είτε consistency
- Σχεδιαστική απόφαση ανάλογα με το σενάριο χρήσης
  - Αν διαλέξουμε consistency (π.χ. 2PC) το σύστημα θα μπλοκάρει μέχρι να επανέλθει η συνέπεια
  - Αν διαλέξουμε availability πάμε για eventual consistency

Διαχείριση των Network Partitions

- Κατά τη διάρκεια ενός partition, ζεύγη από αντικρουόμενα transactions μπορεί να εκτελέστηκαν σε διαφορετικά partitions. Η μόνη επιλογή είναι να διορθώσουμε την κατάσταση μετά από ανάνηψη του συστήματος
  - Ακυρώνουμε ένα από τα αντικρουόμενα transactions μετά την ανάνηψη
  - Βασική ιδέα: Επιτρέπουμε τη συνέχιση των λειτουργιών και λύνουμε τις διαφορές που προκύπτουν μετά την επανένωση του δικτύου

Απαρτία (quorum)

- Για να αποφασίσουμε αν επιτρέπονται τα reads και τα writes
- Υπάρχουν 2 τύποι: απαισιόδοξη απαρτία (pessimistic quorum) και αισιόδοξη απαρτία (optimistic quorum)
- Στην απαισιόδοξη οι ενημερώσεις επιτρέπονται μόνο όταν ένα partition έχει την πλειονότητα των RMs
  - Οι ενημερώσεις διαδίδονται στους υπόλοιπους RMs όταν επανέλθει το σύστημα

Στατική απαρτία

- Η απόφαση για το πόσοι RMs πρέπει να εμπλακούν σε ένα operation πάνω σε αντίγραφα λέγεται επιλογή απαρτίας (quorum selection)
- Κανόνες:
  - Τουλάχιστον r αντίγραφα πρέπει να προσπελαστούν για ένα read
  - Τουλάχιστον w αντίγραφα πρέπει να προσπελαστούν για ένα write
  - r + w > N, όπου N ο αριθμός των αντιγράφων
  - w > N/2
  - Κάθε αντικείμενο έχει ένα version number ή ένα συνεπές timestamp

Static Quorums

- Τι σημαίνει r + w > N;
  - Ο μόνος τρόπος να ικανοποιηθεί η συνθήκη είναι να υπάρχει αλληλοεπικάλυψη του read set και του write set.
  - Οπότε υπάρχει πάντα κάποιο αντίγραφο με την πιο πρόσφατη τιμή write.
- Τι σημαίνει w > N/2;
  - Όταν υπάρχει network partition, μόνο το partition που περιέχει πάνω από τους μισούς RMs μπορεί να εκτελέσει writes
  - Οι υπόλοιποι εξυπηρετούν τα reads με παλιά (stale) δεδομένα
- R και W είναι παραμετροποιήσιμα:
  - Π.χ., N=3, r=1, w=3: Μεγάλο read throughput σε κόστος του write throughput

Αισιόδοξη απαρτία

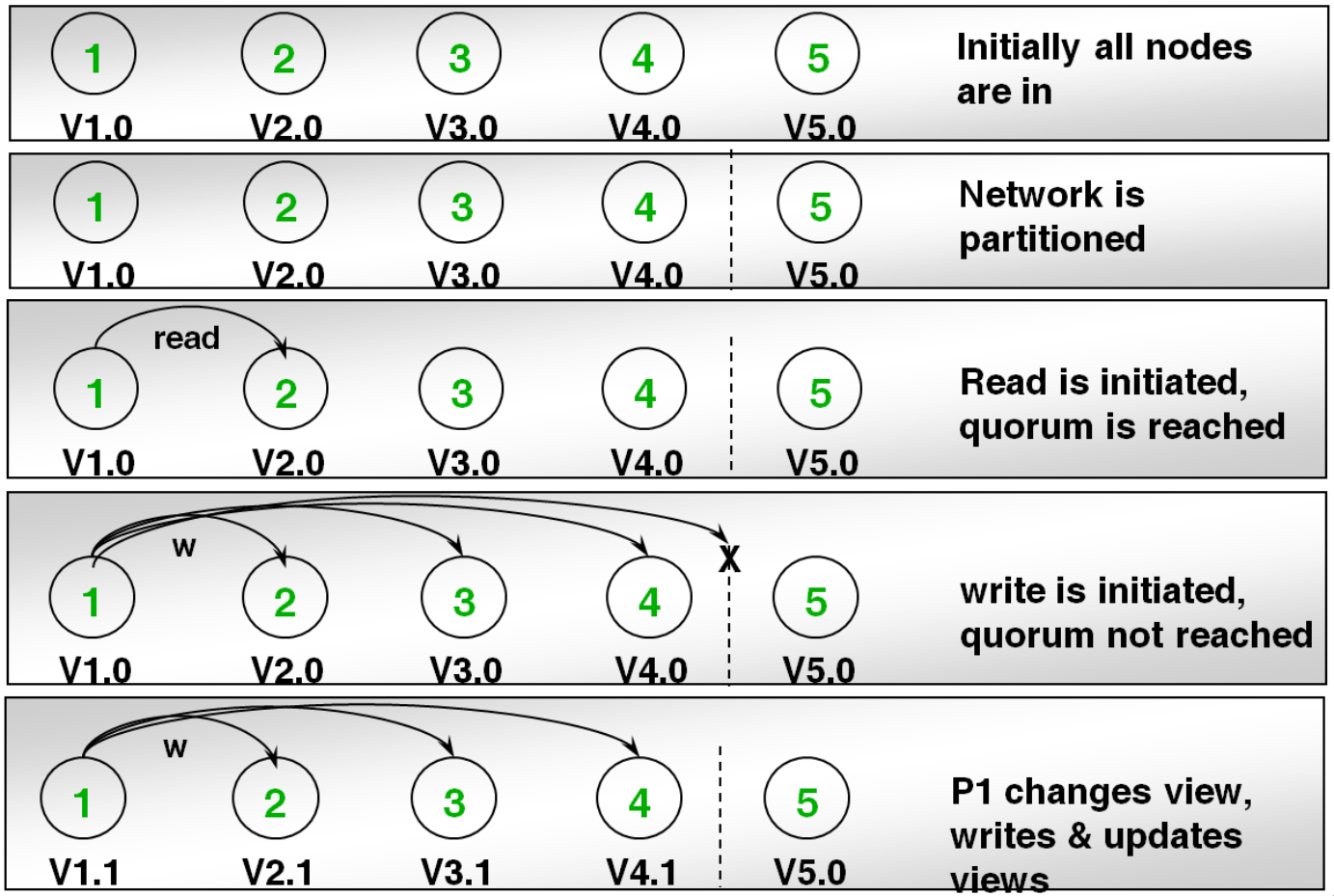
- Η επιλογή αισιόδοξης απαρτίας επιτρέπει τα writes να γίνονται σε οποιοδήποτε partition
- “Write, but don’t commit”
  - Παρά μόνο αφού το σύστημα ανανήψει κάποτε
- Επίλυση write-write conflicts μετά την ανάνηψη
- Η αισιόδοξη απαρτία είναι πρακτική όταν:
  - Οι αντικρουόμενες ενημερώσεις είναι σπάνιες
  - Τα conflicts είναι πάντα ανιχνεύσιμα
  - Οι επιπτώσεις των conflicts μπορούν να αντιμετωπιστούν
  - Τα partitions διαρκούν λίγο

View-based Quorum

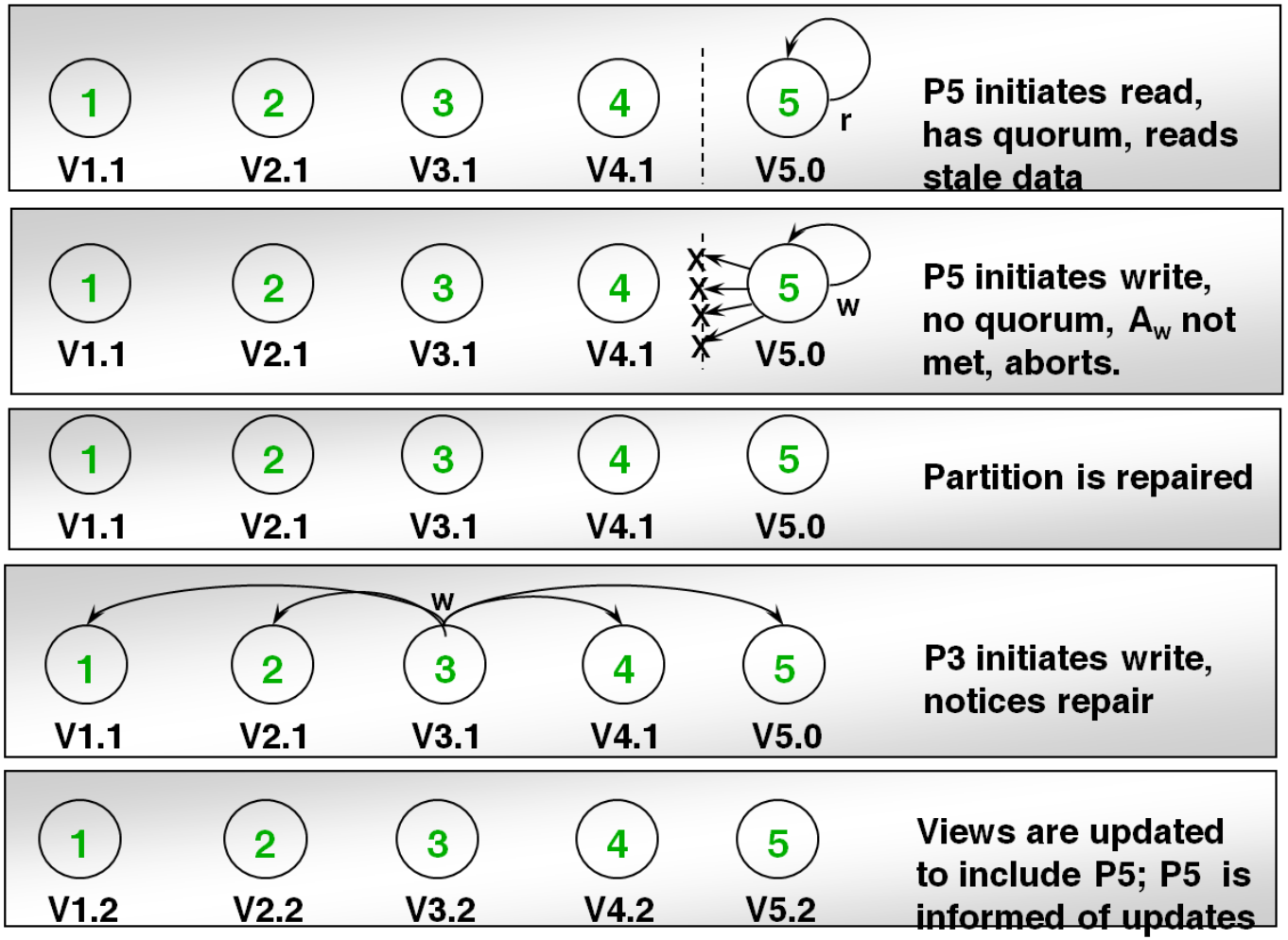
- Προσέγγιση αισιόδοξης απαρτίας
- Η απαρτία βασίζεται σε views οποιαδήποτε στιγμή
  - Χρησιμοποιεί τα primitives του multicast
- Ορίζουμε όρια (thresholds) για κάθε read και write :
  - W: η κανονική απαρτία για write
  - R: η κανονική απαρτία για read
  - A_w: Οι ελάχιστοι κόμβοι σε ένα view για write, π.χ., $A_w > N/4$
  - A_r: Οι ελάχιστοι κόμβοι σε ένα view για read
- Πρωτόκολλο
  - Δοκιμή κανονικής απαρτίας, αν δε δουλέψει, αλλαγή view. Αν ικανοποιηθεί το minimum προχωράμε
  - A_w και A_r καθορίζουν ποιο partition προχωράει

Παράδειγμα

- Consider: N = 5, w = 5, r = 1, A_w = 3, A_r = 1



Συνέχεια

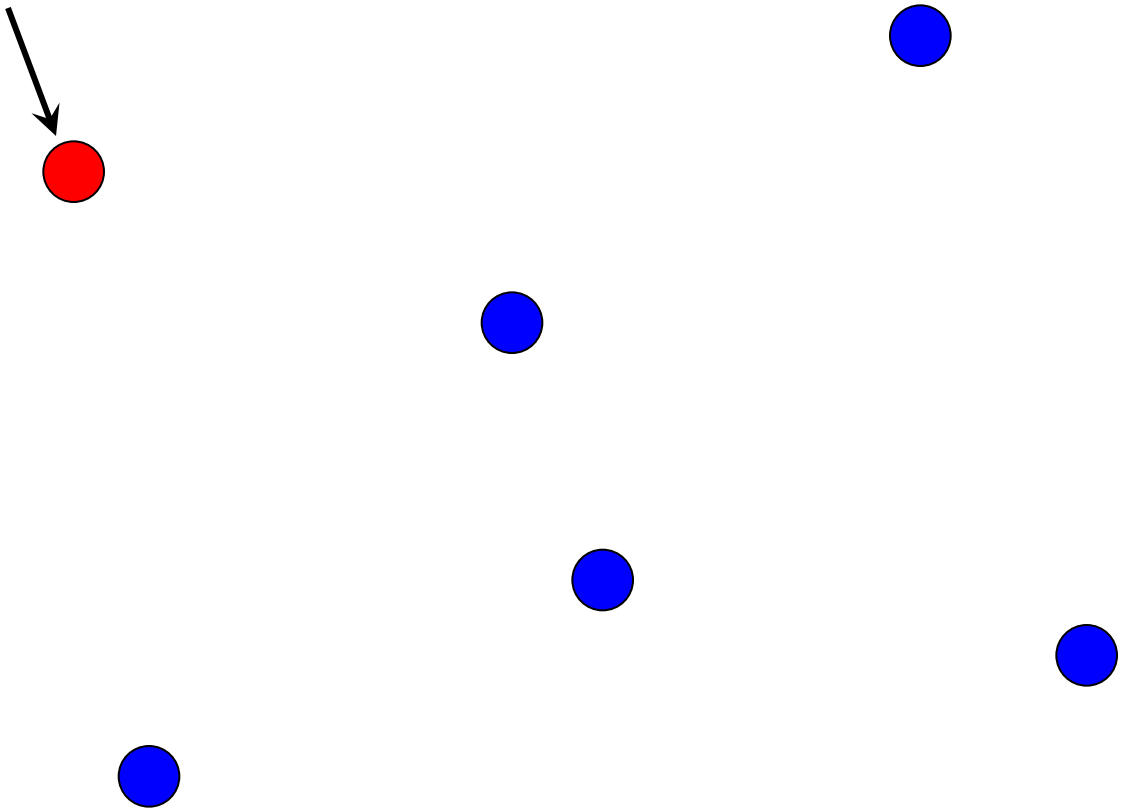


Eager vs. Lazy Replication

- Eager replication, π.χ., B-multicast, R-multicast
  - Αιτήματα με multicast σε όλους τους RMs αμέσως (active replication)
  - Αποτελέσματα με multicast σε όλους τους RMs αμέσως (passive replication)
- Εναλλακτικά: Lazy replication
  - Επιτρέπει στα αντίγραφα να συγκλίνουν eventually and lazily
  - Διάδοση των ενημερώσεων και των ερωτημάτων lazily (π.χ. όταν υπάρχει αρκετό network bandwidth)
  - Ο χρήστης περιμένει απάντηση μόνο από έναν RM
  - Επιτρέπει σε άλλους RMs να αποσυνδεθούν
  - Παρέχει ασθενέστερη συνέπεια αλλά βελτιώνει την επίδοση
- Lazy replication μέσω gossiping

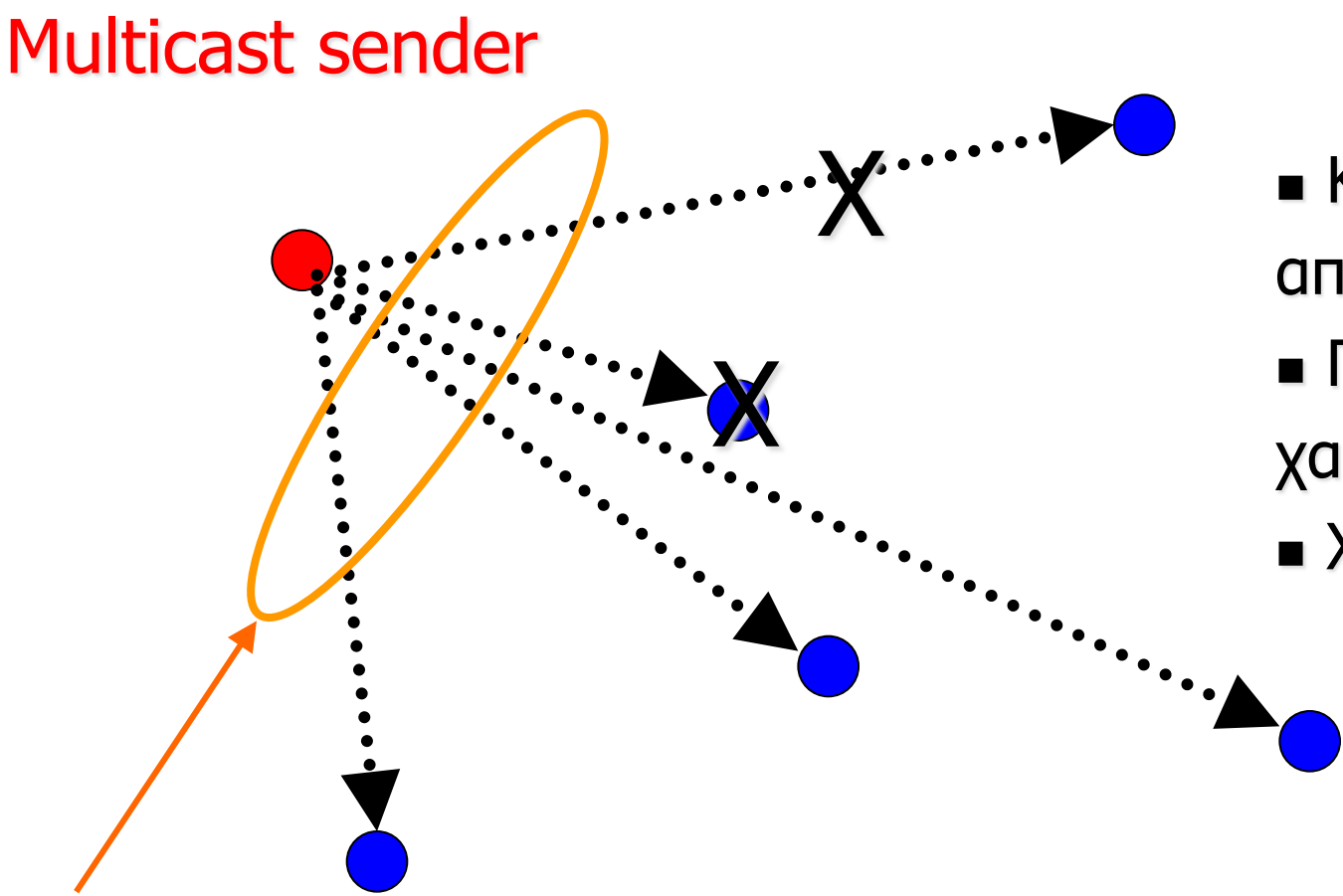
Multicast

Κόμβος με πληροφορία που πρέπει να μοιραστεί με όλους



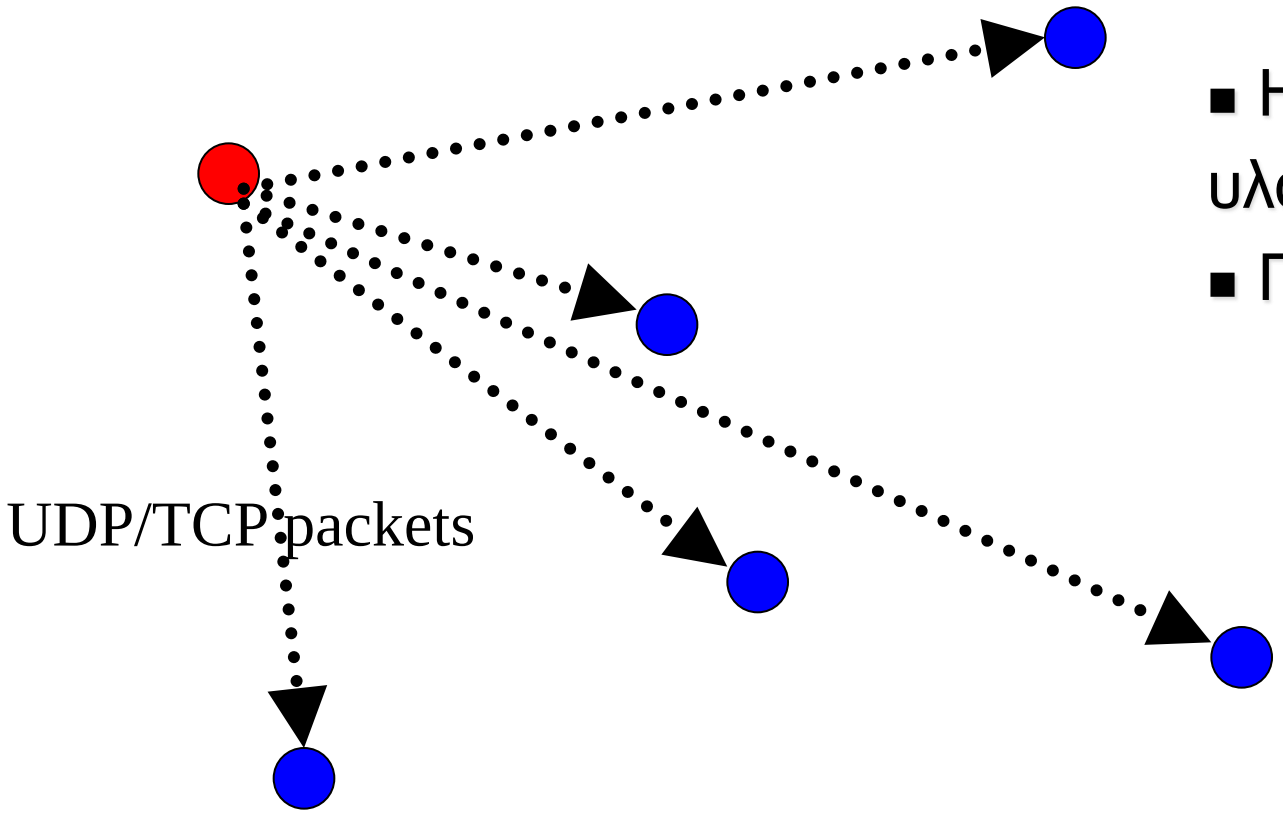
Distributed Group of "Nodes"= Processes at Internet-based hosts

Fault-Tolerance και Scalability



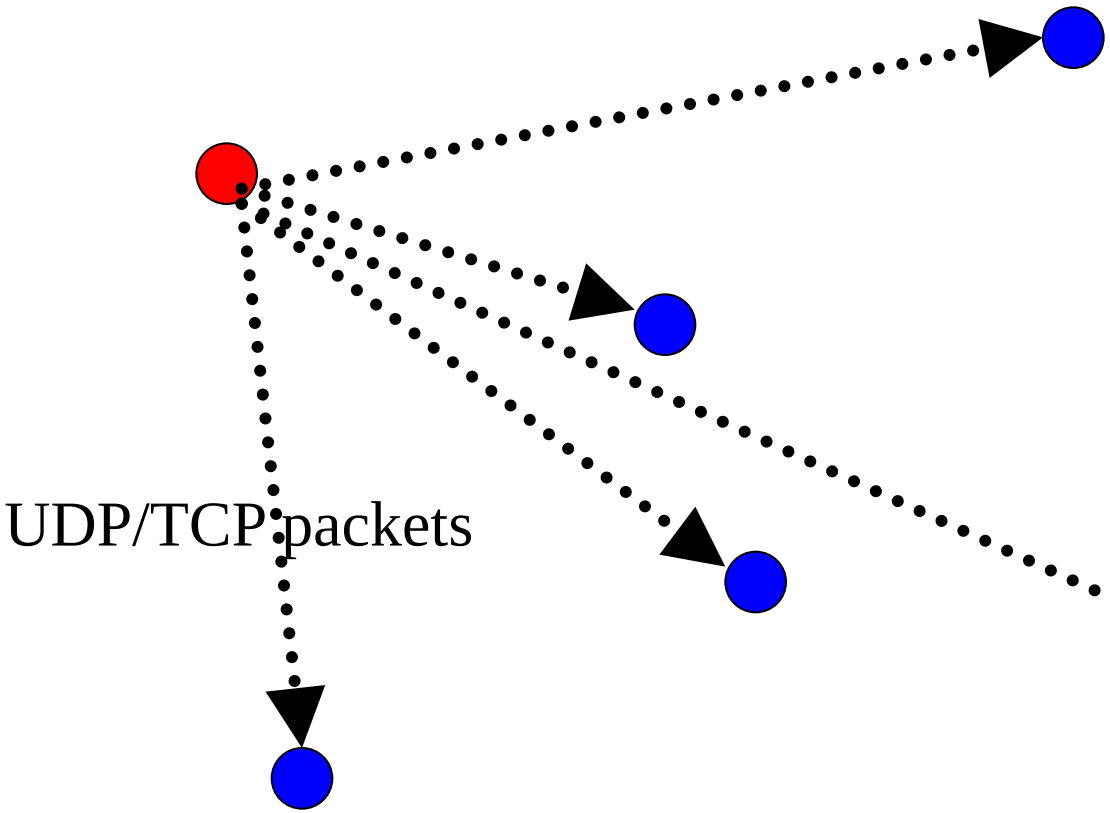
- Κόμβοι μπορεί να αποτύχουν
- Πακέτα μπορεί να χαθούν
- Χιλιάδες κόμβοι

B-Multicast



- Η απλούστερη υλοποίηση
- Προβλήματα;

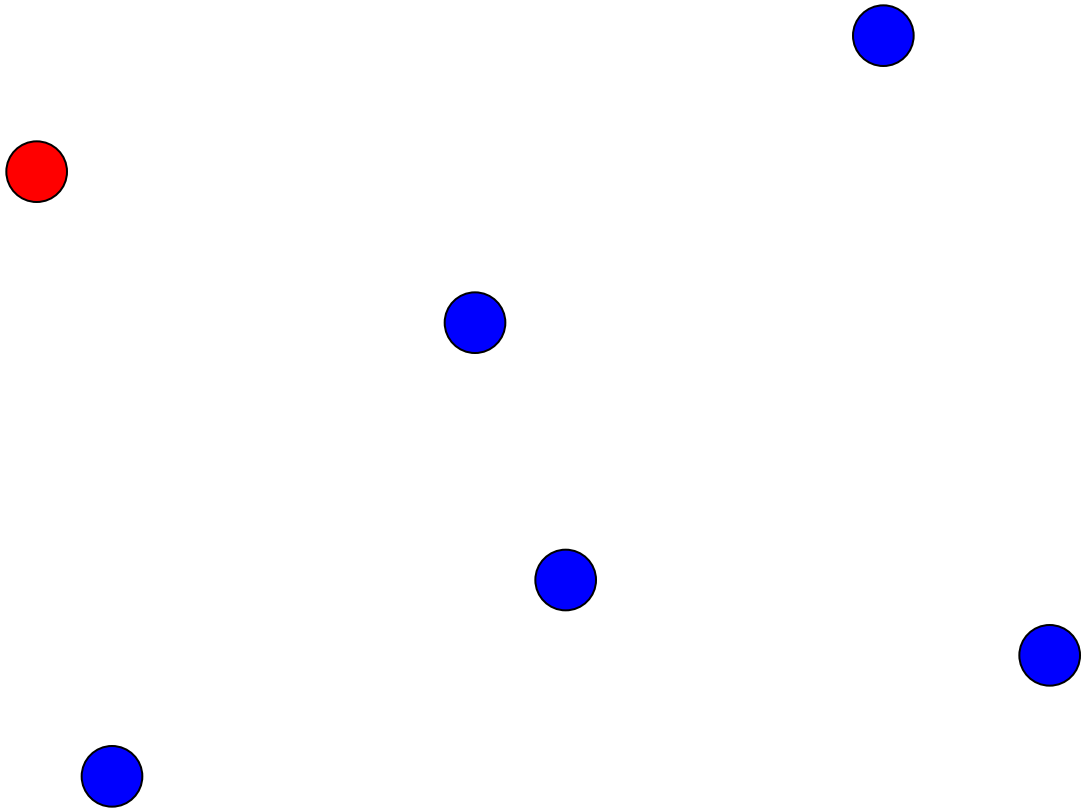
R-Multicast



- Αυστηρότερες εγγυήσεις
- Το overhead είναι ανάλογο του N^2

Μια άλλη προσέγγιση

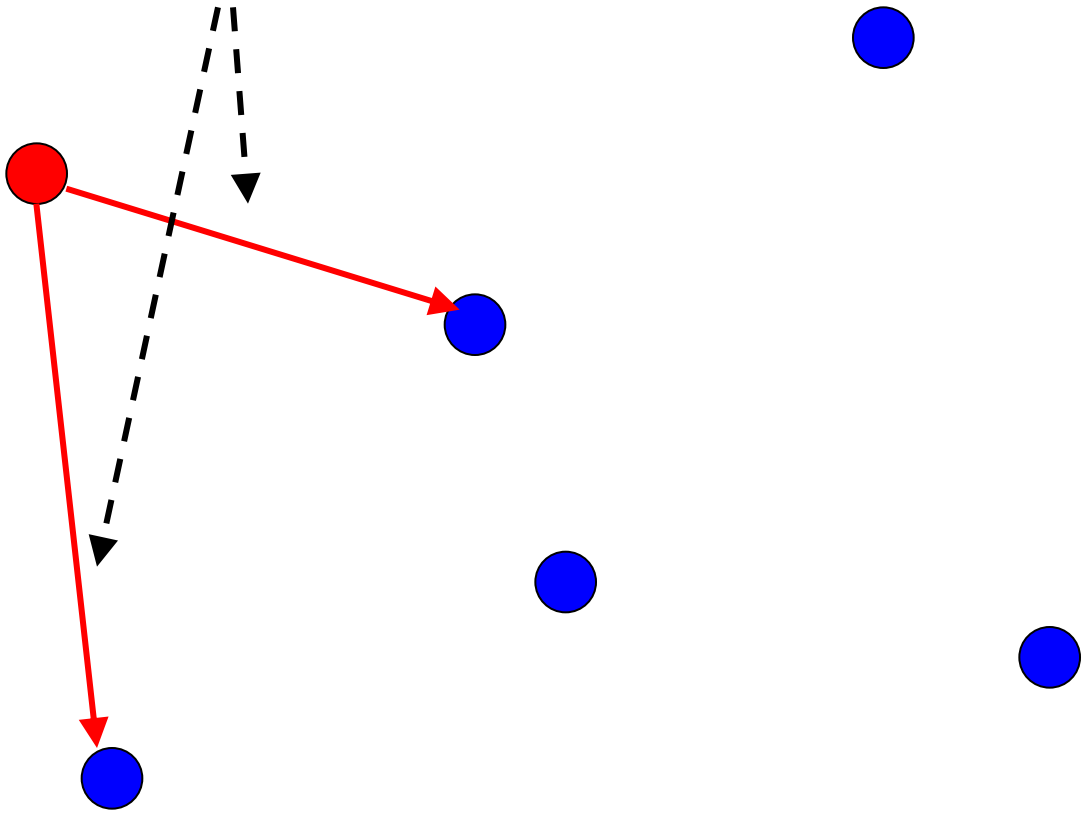
Multicast sender



Μια άλλη προσέγγιση

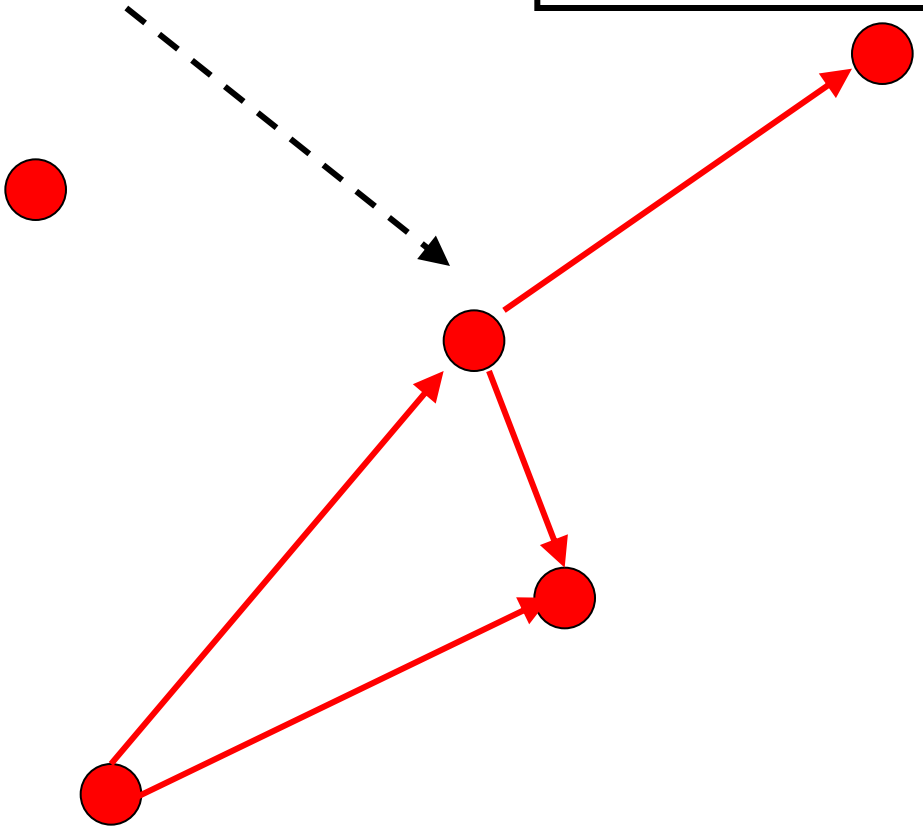
Periodically, transmit to *b* random targets

Gossip messages (UDP)



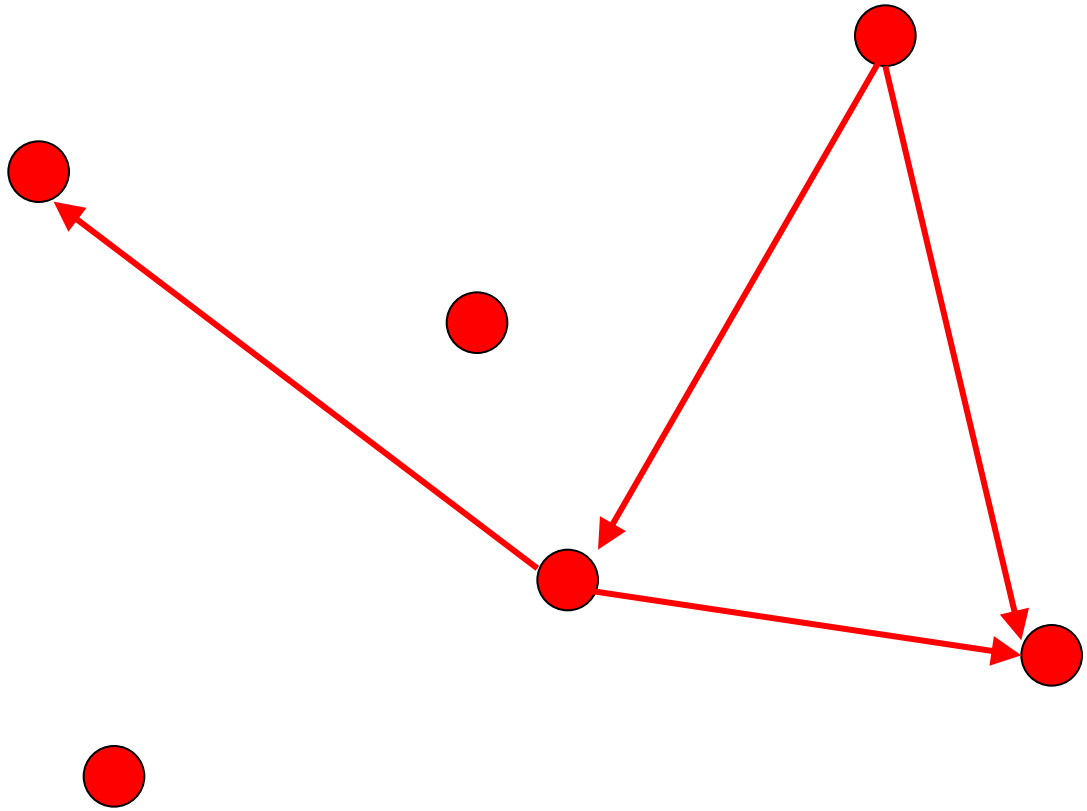
Μια άλλη προσέγγιση

Other nodes do same after receiving multicast

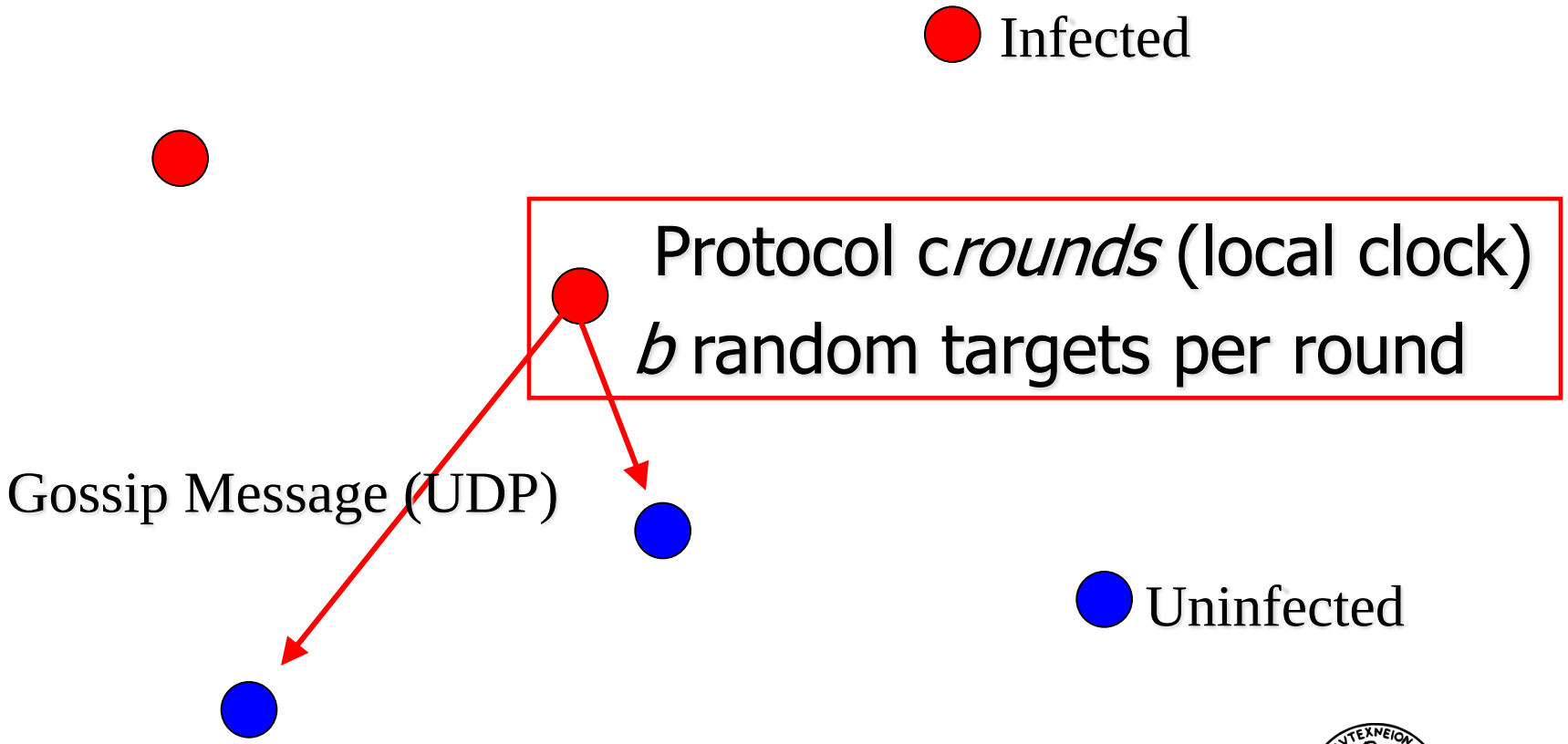


→ Gossip messages (UDP)

Μια άλλη προσέγγιση



“Gossip” (or “Epidemic”) Multicast



Ιδιότητες

- Γρήγορη διάδοση
- Μεγάλη ανοχή σε σφάλματα
- Παράμετροι *c,b*:
  - *c* για τους γύρους: (*c*log(n)), b*: # των κόμβων για επικοινωνία
  - Μπορούν να είναι μικροί αριθμοί, ανεξάρτητοι του n, π.χ., *c=2; b=2;*
- Μέσα σε *c* γύρους (χαμηλό latency)
  - Όλοι εκτός από $\frac{1}{n^{cb-2}}$ κόμβους έχουν λάβει multicast (αξιοπιστία)
  - ~ *c*b*log(n)* μηνύματα gossip (lightweight)

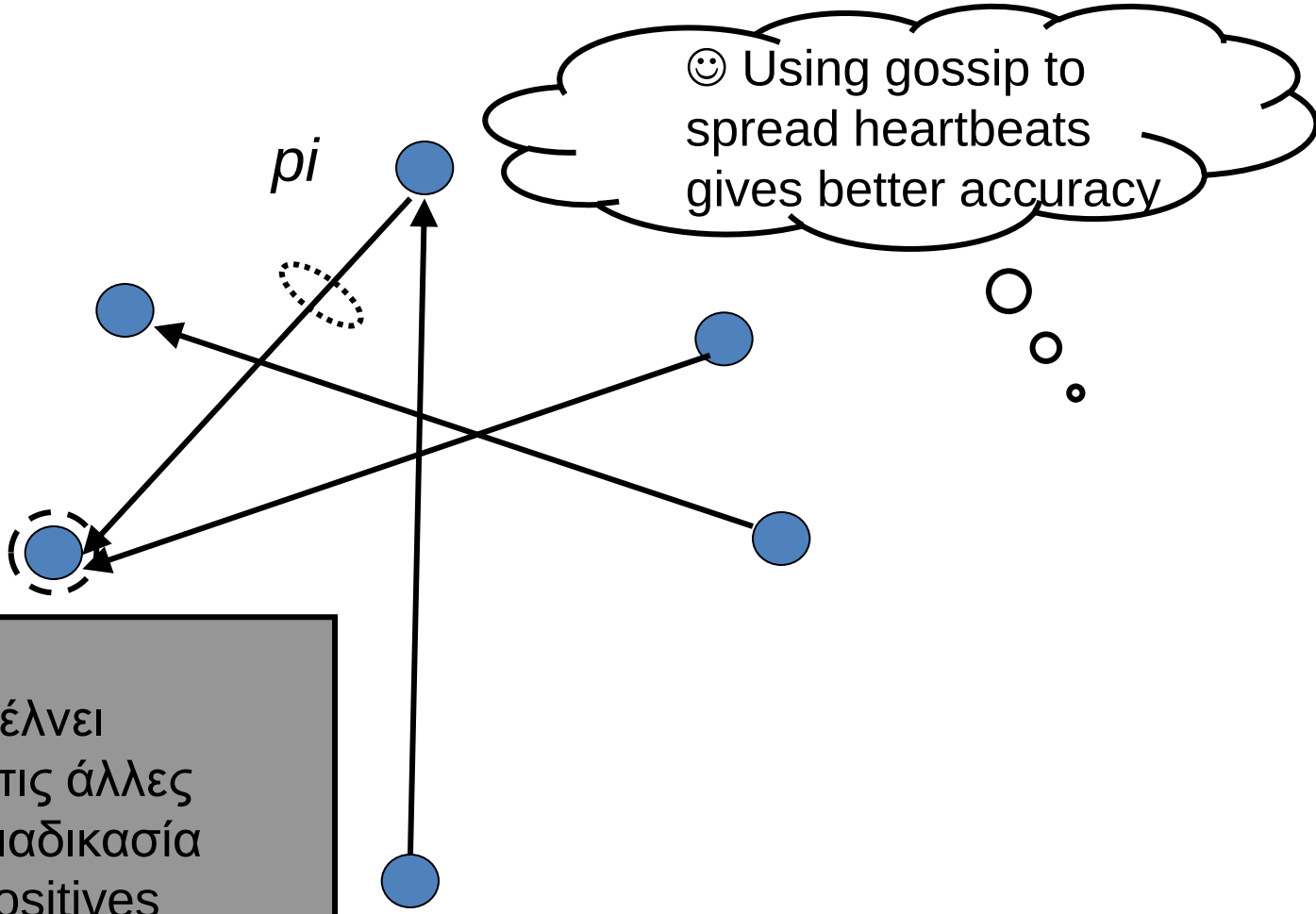
Ανοχή σε σφάλματα

- Απώλεια πακέτων
  - 50% απώλεια πακέτων: ανάλυση με *b/2* αντί για b
  - Για να πετύχουμε ίδια αξιοπιστία όσο με 0% απώλεια πακέτων χρειάζονται διπλάσιοι γύροι
- Αποτυχίες κόμβων
  - 50% των κόμβων αποτυγχάνουν: ανάλυση με n/2 αντί για n και b/2 αντί για b
  - Όπως παραπάνω

Αρχιτεκτονική

- Οι RMs ανταλλάσουν “gossip” μηνύματα
  - Περιοδικά και μεταξύ τους
  - Τα gossip μηνύματα μεταφέρουν ενημερώσεις και βοηθούν στη σύγκλιση όλων των RMs
- Σκοπός: αυξημένη διαθεσιμότητα
- Εγγύηση:
  - Κάθε πελάτης λαμβάνει συνεπές service over time: Ως απάντηση σε ένα αίτημα ένας RM ίσως πρέπει να περιμένει για να λάβει τις ενημερώσεις που πρέπει από άλλους RMs. Ο RM τότε παρέχει στον πελάτη τα δεδομένα που αντικατοπτρίζουν τουλάχιστον τις ενημερώσεις που έχει δει ο client μέχρι τώρα.
  - Χαλαρή συνέπεια ανάμεσα σε replicas: Οι RMs μπορεί να είναι ασυνεπείς οποιαδήποτε στιγμή. Ωστόσο όλοι οι RMs τελικά θα λάβουν όλες τις ενημερώσεις και θα τις εκτελέσουν με τη σωστή διάταξη

Gossip για ανίχνευση σφαλμάτων



All-to-all heartbeating
- Κάθε διεργασία στέλνει heartbeat σε όλες τις άλλες
- Μείον: Μια αργή διαδικασία δημιουργεί false positives

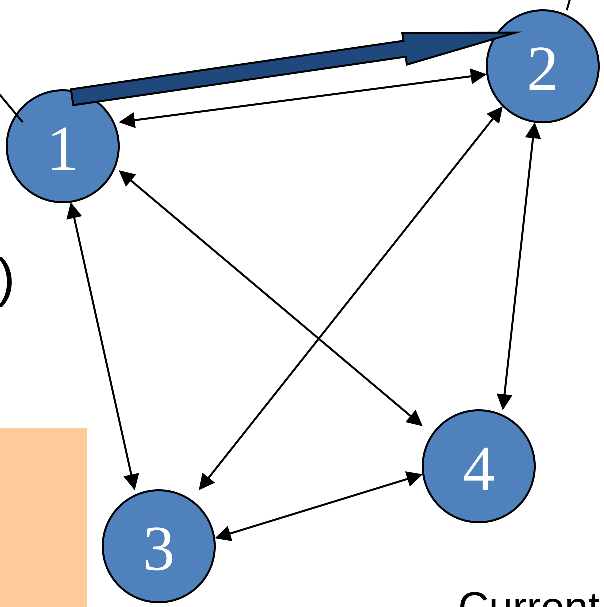
Gossip για ανίχνευση σφαλμάτων

1	10120	66
2	10103	62
3	10098	63
4	10111	65

Address

Heartbeat Counter

Time (local)



1	10118	64
2	10110	64
3	10090	58
4	10111	65

1	10120	70
2	10110	64
3	10098	63
4	10111	65

Current time : 70 at process 2

(asynchronous clocks)

Πρωτόκολλο:

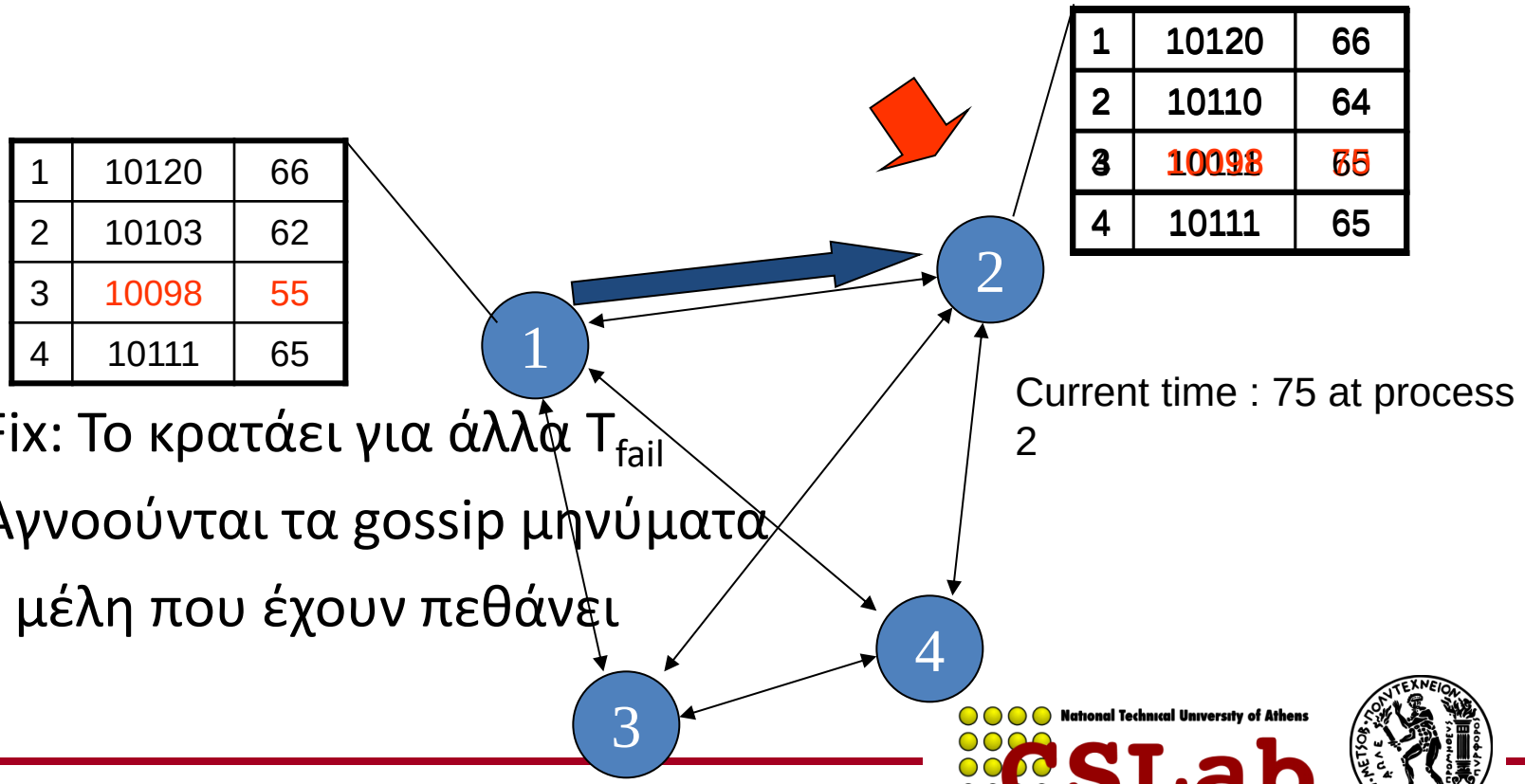
- Οι διεργασίες περιοδικά επικοινωνούν με gossip τη λίστα μελών
- Κατά την παραλαβή ενημερώνονται οι τοπικές λίστες

Gossip για ανίχνευση σφαλμάτων

- Αν το heartbeat δεν έχει αυξηθεί για πάνω από T_{fail} δευτερόλεπτα, το μέλος θεωρείται νεκρό
- Αλλά δε σβήνεται αμέσως
- Περιμένουμε άλλα $T_{cleanup}$ δευτερόλεπτα και μετά σβήνουμε το μέλος από τη λίστα

Gossip για ανίχνευση σφαλμάτων

- Αν κάποια διεργασία σβηστεί αμέσως μετά από T_{fail} seconds?



- Fix: Το κρατάει για άλλα T_{fail}
- Αγνοούνται τα gossip μηνύματα για μέλη που έχουν πεθάνει

Σύνοψη

- Σειριοποιησιμότητα
  - Η διάταξη των λειτουργιών καθορίζεται από τον χρόνο
- Ακολουθιακή συνέπεια
  - Η διάταξη των λειτουργιών καθορίζεται από την διάταξη του προγράμματος σε κάθε πελάτη
- Causal consistency & eventual consistency
- Απαρτία
  - Στατική, αισιόδοξη, view-based
- Eager replication vs. lazy replication
  - Lazy replication -> ενημερώσεις στο background
- Gossiping
  - Στρατηγική για lazy replication
  - High-level of fault-tolerance & quick spread