Κατανεμημένα Συστήματα

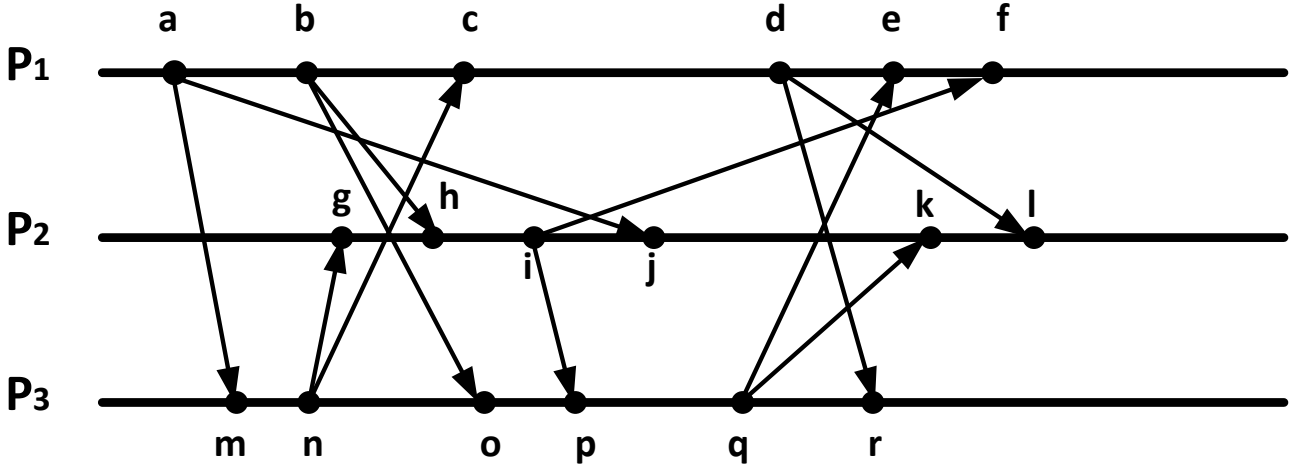
Ασκήσεις

2018-2019

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

Άσκηση 1

- 3 διεργασίες, η P1, η P2 και η P3 στέλνουν μεταξύ τους multicast μηνύματα. Θέλουμε να εξασφαλίσουμε:
- (α) ταξινόμηση FIFO
- (β) αιτιώδη (causal) ταξινόμηση



Flashback: Διάταξη FIFO

- Η διάταξη της παράδοσης των μηνυμάτων ακολουθεί τη σειρά αποστολής τους από κάθε διεργασία

- Αν μια σωστή διεργασία στείλει multicast(*g*,*m*) και μετά multicast(*g*,*m'*), τότε κάθε σωστή διεργασία που παραδίδει το *m'* θα έχει παραδώσει ήδη το m

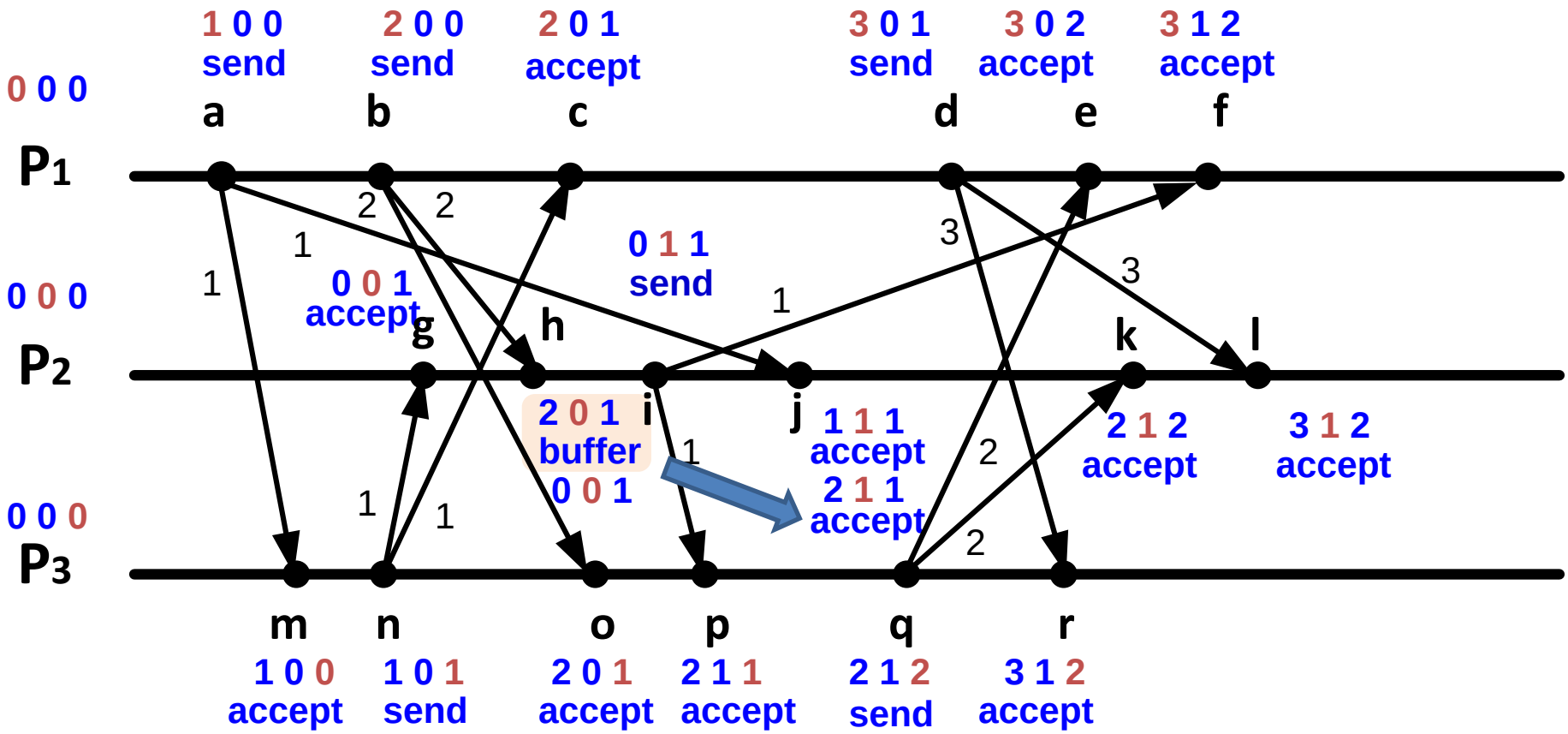
- Παράδειγμα:
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8

- FIFO -> (m0, m3, m6, m1, m4, m7, m2, m5, m8)

Διάταξη FIFO

- Εξέταση μηνυμάτων από κάθε διεργασία με τη σειρά με την οποία εστάλησαν
  - Κάθε διεργασία κρατά έναν αύξοντα αριθμό για κάθε μια από τις άλλες διεργασίες
  - Όταν παραλαμβάνεται ένα μήνυμα, αν ο αύξων αριθμός του είναι:
    - Ο αναμενόμενος (επόμενος στη σειρά), το αποδεχόμαστε
    - Μεγαλύτερος από τον αναμενόμενο, το τοποθετούμε σε ουρά
    - Μικρότερος από τον αναμενόμενο, το απορρίπτουμε

FIFO



Flashback: Αιτιώδης διάταξη

- Η διάταξη της παράδοσης των μηνυμάτων σε κάθε διεργασία διατηρεί τις σχέσεις happened-before ανάμεσα σε όλες τις διεργασίες

- Αν multicast(*g*,*m*) → multicast(*g*,*m'*) τότε κάθε 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
  - Σχέσεις happened-before: m0 → m4, m5 → m8

- Causal -> (m0, m3, m6, m1, m4, m7, m2, m5, m8)

Αιτιώδης διάταξη

- Κάθε διεργασία διατηρεί διανυσματικό ρολόι
  - Κάθε μετρητής αναπαριστά τον αριθμό των μηνυμάτων που έχουν ληφθεί από κάθε άλλη διεργασία

- Κατά το multicast, η διεργασία-αποστολέας αυξάνει τον μετρητή της και επισυνάπτει το διάνυσμα στο μήνυμα

- Κατά τη λήψη ενός μηνύματος multicast ο παραλήπτης περιμένει μέχρι να ικανοποιηθεί η αιτιώδης διάταξη, δλδ:
  - Να έχει παραδώσει όσα μηνύματα έχει στείλει ο αποστολέας στο παρελθόν
  - Να έχει παραδώσει όλα τα μηνύματα που είχε παραδώσει ο αποστολέας πριν την αποστολή του multicast

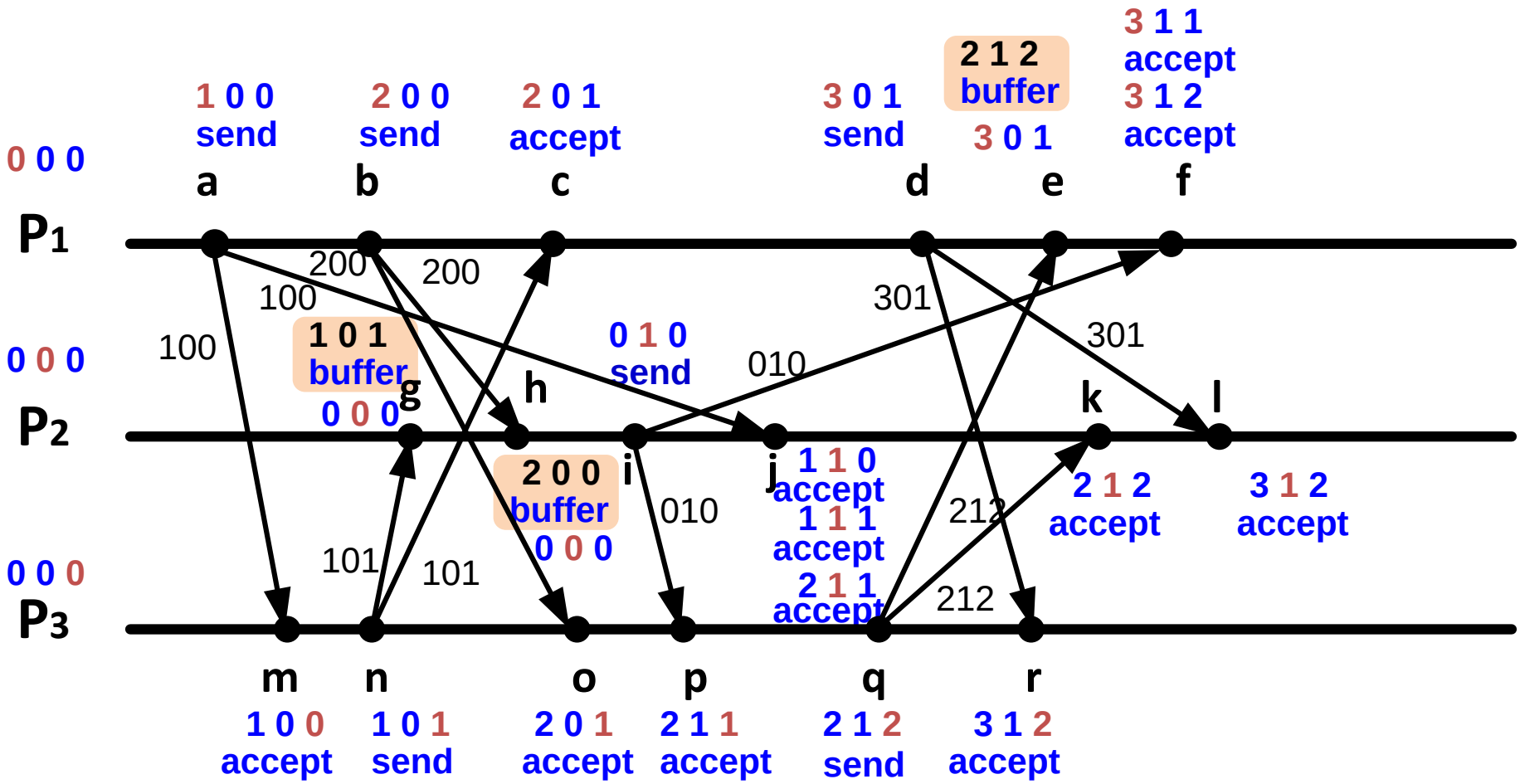
Αλγόριθμος

- Όταν η Pb στέλνει μήνυμα
  - **Αυξάνει τη δική του τιμή και στέλνει το διάνυσμα** Vb[b] = Vb[b] + 1 με το μήνυμα
- Όταν η Pa **λάβει μήνυμα από την Pb**
  - Τσεκάρει αν το μήνυμα ήρθε με σειρά FIFO από την Pb

    Vb[b] == Va[b] + 1 ?
  - Τσεκάρει ότι το μήνυμα δεν εξαρτάται από κάτι που δεν έχει δει ακόμα η Pa

    $\forall i, i \neq b: Vb[i] \leq Va[i]$?
  - Αν ικανοποιούνται και οι δύο συνθήκες, το Pa παραδίδει το μήνυμα
  - Αλλιώς το κρατά σε αναμονή μέχρι να ικανοποιηθούν οι συνθήκες

Causal



Άσκηση 2

Dataset με δημογραφικά στοιχεία:

{name, age, address, zipcode, salary}.

Παράγετε τη λίστα από zipcodes στα οποία ο μέσος μισθός είναι:

(α) κάτω από $100K,

(β) από $100K μέχρι $500K, και

(γ) πάνω από $500K.

MAP(key1, value1):

//επεξεργασία για κάθε <key1,value1>

emit(key2,value2)

REDUCE(key2,list(value2)):

//επεξεργασία για κάθε <key2, list(value2)>

emit(key3,value3)

Άσκηση 2

1. Μέσος μισθός ανά zipcode

map(key, value):
// key: some tupleID;
//value: tuple {name, age, address, zipcode, salary}
emit(zipcode, salary)

reduce(zipcode, list(salary)):
// key: a zipcode; value: an iterator over salaries
result = 0
count = 0
for each salary s in list(salary){
result += s
count++ }
mean_salary_per_zip = result/count
emit(zipcode, mean_salary_per_zip)

Άσκηση 2

2. Ομαδοποίηση zipcodes

map(zipcode, mean_salary_per_zip):

Group = “ ”

if salary_per_zip <100k

    group = “less_than_100”

else if salary_per_zip>=500k

    group = “more_than_500”

else

    group = “between_100_500”

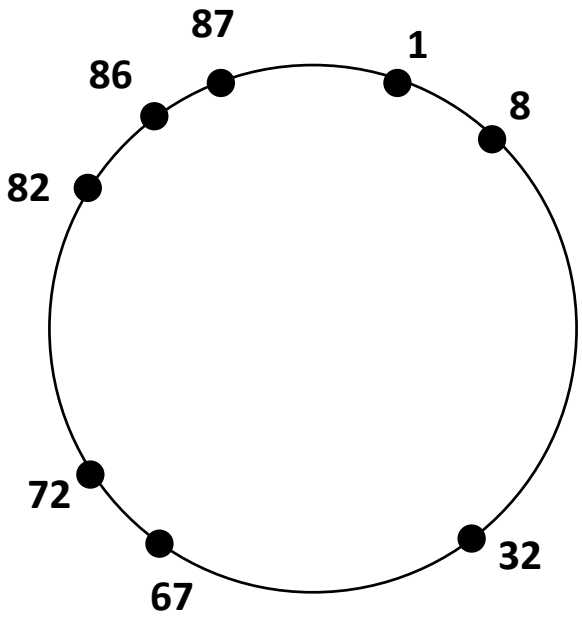
emit(group, zipcode)

reduce(group, list(zipcode)):

        emit(group, list(zipcode))

Άσκηση 3

Chord με αναγνωριστικά 0-99 και 8 κόμβους:



Άσκηση 3

Πόσα fingers είναι απαραίτητα στον πίνακα δρομολόγησης;

Ids 0-99 -> 2^7 =128 > 100 οπότε 7 fingers

Καταγράψτε τον πίνακα finger του κόμβου 82

$f_n(i) = \text{Successor}((n+2^i) \bmod 100)$

$f_{82}(0) = \text{Successor}((82+1) \bmod 100) = \text{Successor}(83) = 86$

$f_{82}(1) = \text{Successor}((82+2) \bmod 100) = \text{Successor}(84) = 86$

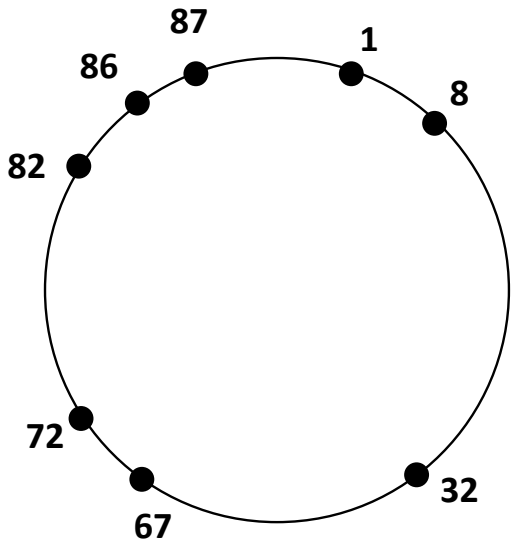
$f_{82}(2) = \text{Successor}((82+4) \bmod 100) = \text{Successor}(86) = 86$

$f_{82}(3) = \text{Successor}((82+8) \bmod 100) = \text{Successor}(90) = 1$

$f_{82}(4) = \text{Successor}((82+16) \bmod 100) = \text{Successor}(98) = 1$

$f_{82}(5) = \text{Successor}((82+32) \bmod 100) = \text{Successor}(14) = 32$

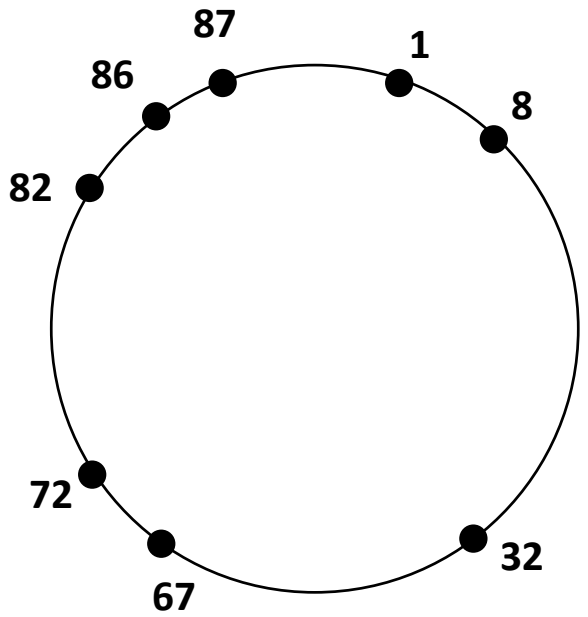
$f_{82}(6) = \text{Successor}((82+64) \bmod 100) = \text{Successor}(46) = 67$



Άσκηση 3

Σημειώστε τις περιοχές των αναγνωριστικών (ids) για τις οποίες είναι υπεύθυνος ο κάθε κόμβος στον δακτύλιο.

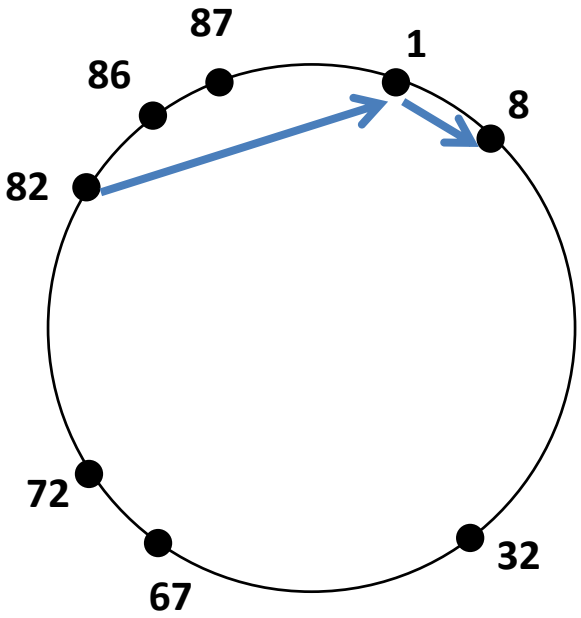
PeerID	From	To
1	88	1
8	2	8
32	9	32
67	33	67
72	68	72
82	73	82
86	83	86
87	87	87



Άσκηση 3

Ο κόμβος 82 εκτελεί αναζήτηση για το αντικείμενο με id 7. Καταγράψτε το μονοπάτι που θα ακολουθηθεί (με χρήση finger tables) μέχρι την εύρεσή του. Υποθέτουμε ότι το δίκτυο είναι σταθερό και δε συμβαίνουν σφάλματα.

82 -> 1 -> 8



Άσκηση 4

Τρεις servers, X, Y και Z, διατηρούν αντίγραφα των δεδομένων A και B. Για τη διατήρηση της συνέπειας των αντιγράφων εφαρμόζεται ο αλγόριθμος της ομοφωνίας με στατική απαρτία (static quorum consensus) με R = W = 2. Οι αρχικές τιμές των αντιγράφων για το A είναι 100 και για το B είναι 50 σε όλους τους servers. Ένας client διαβάζει την τιμή του A και μετά τη γράφει στο B.

Flashback

- 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

Άσκηση 4

Ένα network partition απομονώνει τον server Z από τους X και Y. Πώς θα εκτελεστούν οι λειτουργίες αν ο client μπορεί να προσπελάσει (i) μόνο τον X και Y και (ii) μόνο τον Z.

Έστω ότι αρχικά όλα στην v0

X	Y	Z
A= 100 (vo)	A= 100(vo)	A= 100(vo)
B= 50(vo)	B= 50(vo)	B= 50(vo)

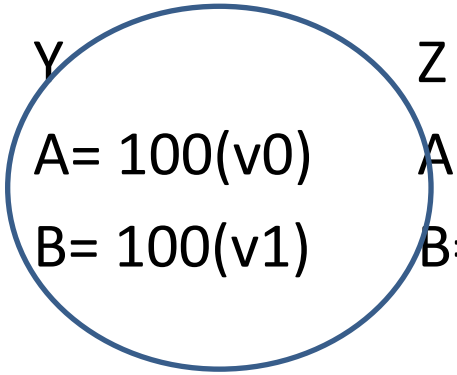
i) Αφού R=2, διαβάζει την τιμή A (100) από τον X ή τον Y

Αφού W=2, γράφει την τιμή B=100 στον X και στον Y

ii) Δεν μπορεί ούτε να γράψει ούτε να διαβάσει γιατί το quorum είναι 1 και στις 2 περιπτώσεις

Άσκηση 4

Ο διαμερισμός επιδιορθώνεται και στη συνέχεια συμβαίνει νέος διαμερισμός που χωρίζει τους X και Z από τον Y. Περιγράψτε τι θα συμβεί αν κάποιος client θέλει να διαβάσει την τιμή του B και να την αυξήσει κατά 20, αν μπορεί να προσπελάσει μόνο τον X και Z



X	Y	Z
A= 100 (v0)	A= 100(v0)	A= 100(vo)
B= 100(v1)	B= 100(v1)	B= 50(vo)

Το read request χρειάζεται quorum από X και Z. Διαφορετικές versions - > ο Z ανανεώνει την τιμή του B με την πιο πρόσφατη version από τον X

Υπάρχει read quorum 2 αλλά και write quorum 2