

# Δίκτυα Ομότιμων Κόμβων

Κατανεμημένα Συστήματα  
2016-2017

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

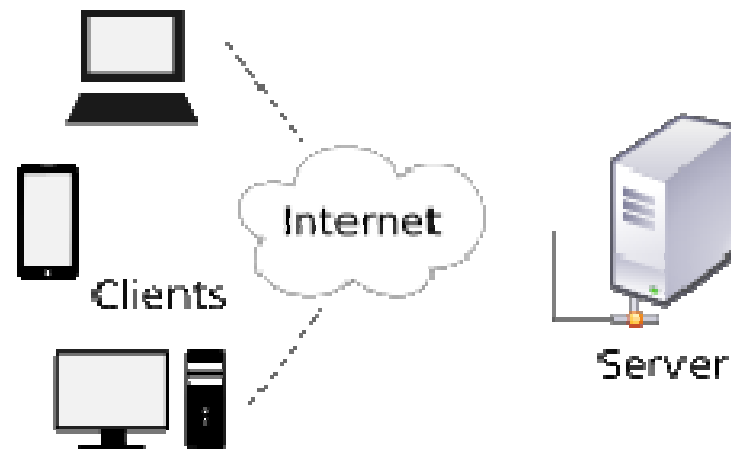
# Κατηγορίες κατακεμημένων συστημάτων

---

- Πώς οργανώνουμε τους κόμβους σε ένα κατακεμημένο σύστημα;
  - Μοντέλο πελάτη-εξυπηρετητή (client-server model)
  - Δίκτυα ομότιμων κόμβων (Peer-to-peer networks ή απλώς P2P)
    - Αδόμητα
    - Δομημένα

# Μοντέλο πελάτη-εξυπηρετητή

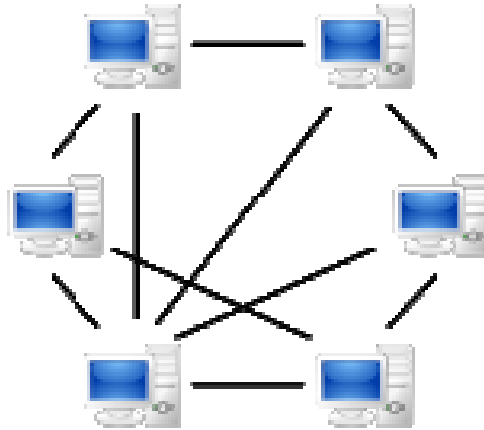
---



- Ο πελάτης στέλνει αιτήματα στον εξυπηρετητή
  - Ο εξυπηρετητής παρέχει πόρους ή υπηρεσίες στους πελάτες
  - Οι πελάτες δεν έχουν καμία μεταξύ τους επικοινωνία
- + Εύκολη υλοποίηση και διαχείριση**
- Single point of failure, δεν κλιμακώνεται εύκολα**
- E-mail, www, ftp, DNS, κλπ.

# Το μοντέλο ομότιμων κόμβων (P2P)

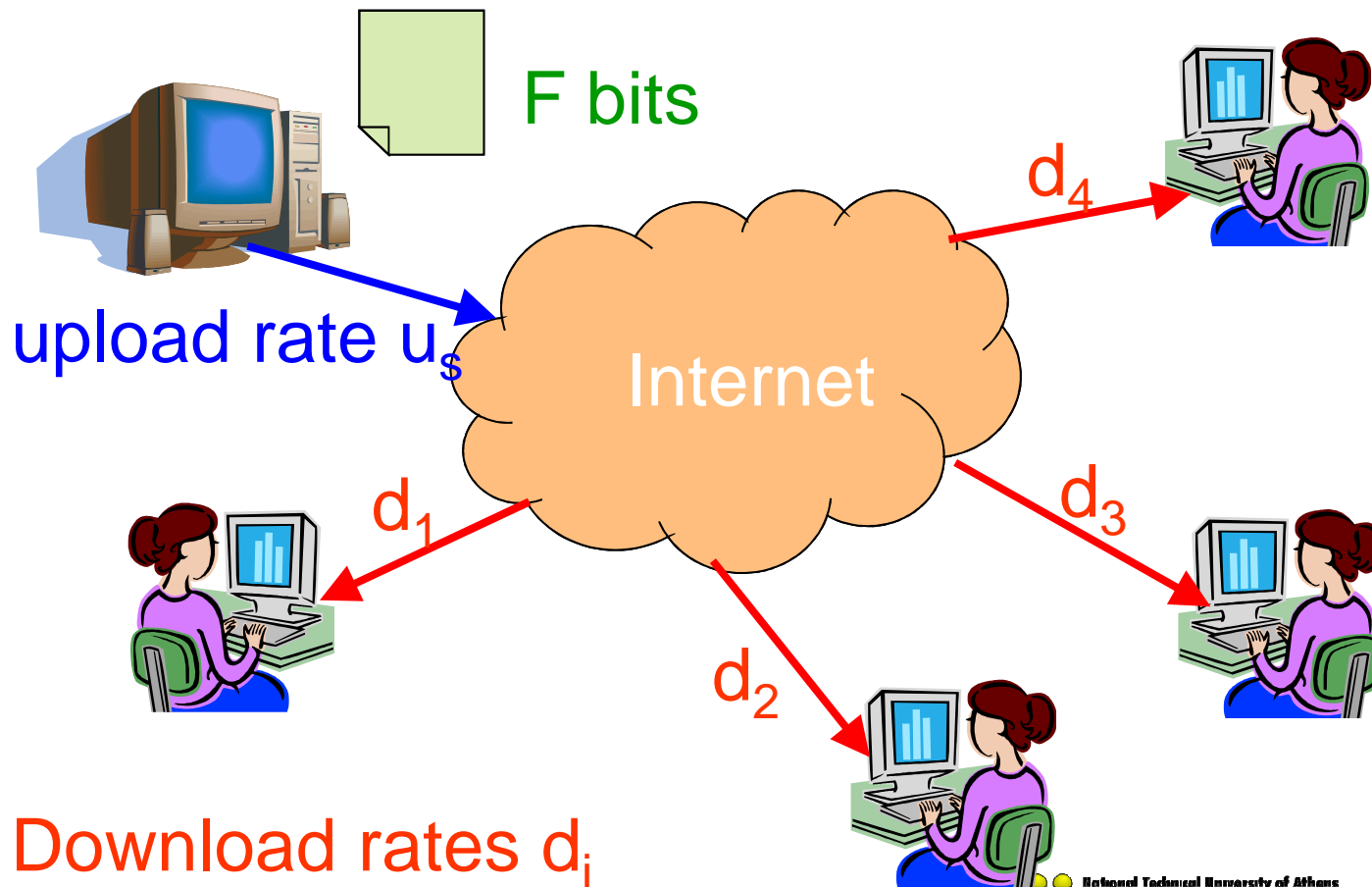
---



- Όλοι οι κόμβοι είναι ισότιμοι (και πελάτες και εξυπηρετητές)
  - Επικοινωνούν μεταξύ τους
- + Robustness, scalability, αυτό-οργάνωση**
- Δύσκολη διαχείριση, ασφάλεια**
- BitTorrent, skype, κλπ.

# Παράδειγμα: Διαμοιρασμός αρχείου

- Γίνεται και με client-server



# Προβλήματα

---

- ...όμως κάποιες φορές δεν είναι η καλύτερη λύση
  - Περιορισμένο bandwidth
  - Ένας server μπορεί να εξυπηρετήσει συγκεκριμένο αριθμό clients
- Αύξηση του upload rate από τη μεριά του server:
  - Σύνδεση με μεγαλύτερο bandwidth για έναν server
  - Πολλαπλοί servers, καθένας με δική του σύνδεση
  - Απαιτεί αλλαγή στην υποδομή
- Εναλλακτικά: Βάζουμε τους παραλήπτες να βοηθήσουν
  - Οι παραλήπτες λαμβάνουν αντίγραφο του αρχείου
  - Το αναδιανέμουν σε άλλους παραλήπτες
  - Μειώνουν τον φόρτο του server

# Προκλήσεις

---

- Οι peers έρχονται και φεύγουν
  - Συνδέονται κατά διαστήματα
  - Οποιαδήποτε στιγμή, με γρήγορο ρυθμό
  - Μπορεί να επιστρέψουν με διαφορετικό IP
- Πώς εντοπίζουμε σχετικούς peers;
  - Peers που είναι αυτή τη στιγμή online
  - Peers που έχουν το αρχείο που ζητάμε
- Πώς παρακινώ τους peers να μείνουν συνδεδεμένοι;
  - Γιατί να μην αποχωρήσουν μόλις κατεβάσουν το αρχείο;
  - Γιατί να διαμοιράσουν περιεχόμενο σε άλλους peers;
- Πώς θα κατεβάσω ένα αρχείο αποδοτικά
  - Όσο πιο γρήγορα γίνεται

# Λύση με αδόμεητα P2P

---

- Δεν χρησιμοποιείται κανένας αλγόριθμος για την οργάνωση τους
- Ανάλογα με τον τρόπο εντοπισμού των κόμβων
  - Με κεντρικό κατάλογο (Napster)
  - Με πλημμύρα (Gnutella)
  - Ιεραρχικά (Kazaa, modern Gnutella)





# Napster

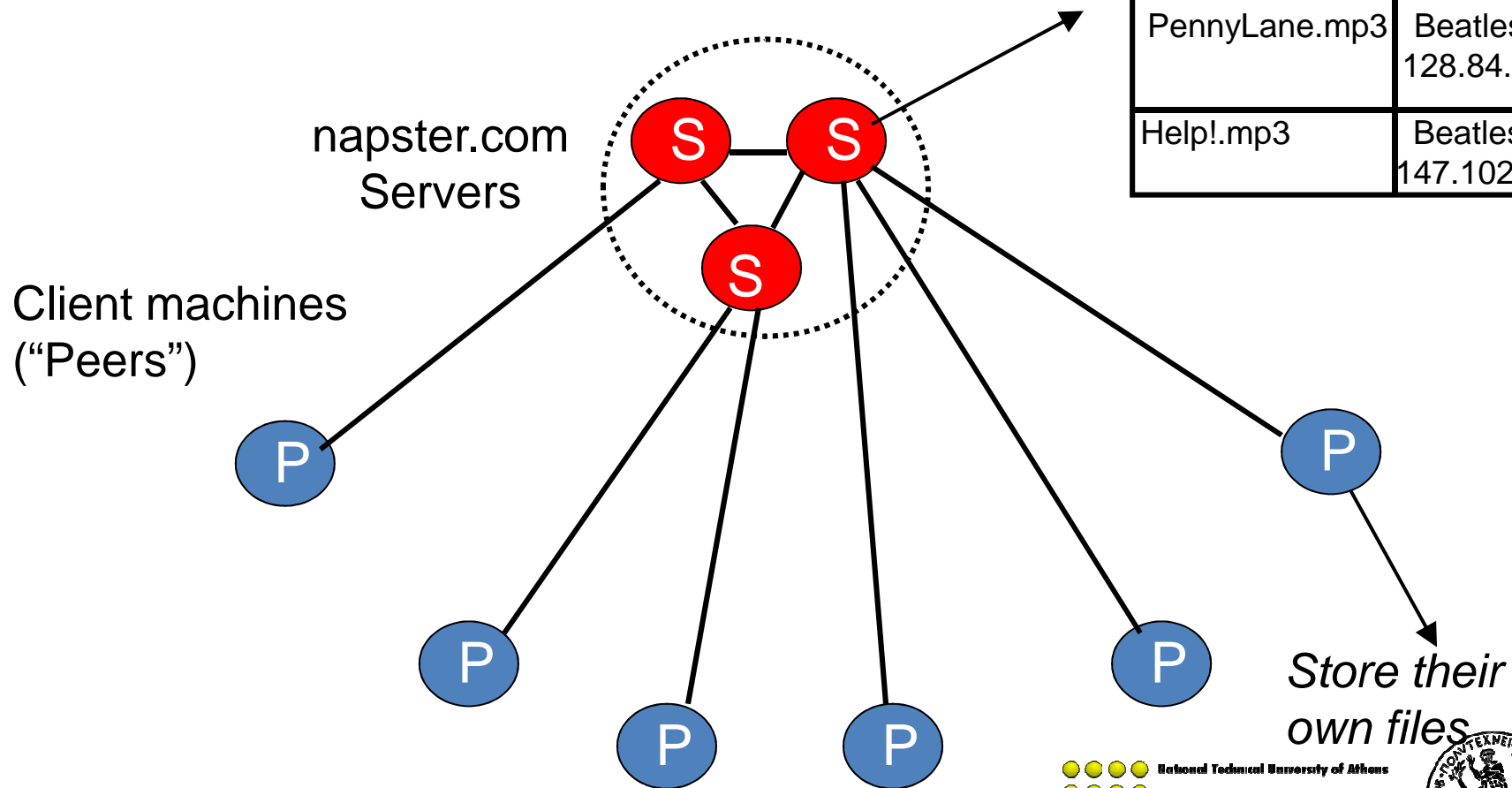
---

- Ένας κόμβος (ή κάποιοι κόμβοι) λειτουργεί ως κεντρικός κατάλογος
- Για να συνδεθούν στο δίκτυο οι clients πρέπει πρώτα να συνδεθούν στον κεντρικό κατάλογο
- Ο κεντρικός κατάλογος διατηρεί
  - πίνακα με στοιχεία για εγγεγραμμένους clients (IP, bandwidth κλπ.)
  - πίνακα με αρχεία ανά client και πληροφορίες για αυτά (π.χ. όνομα αρχείου, τύπος, ημερομηνία κλπ.)
- Βασικές λειτουργίες
  - join
  - search
  - download

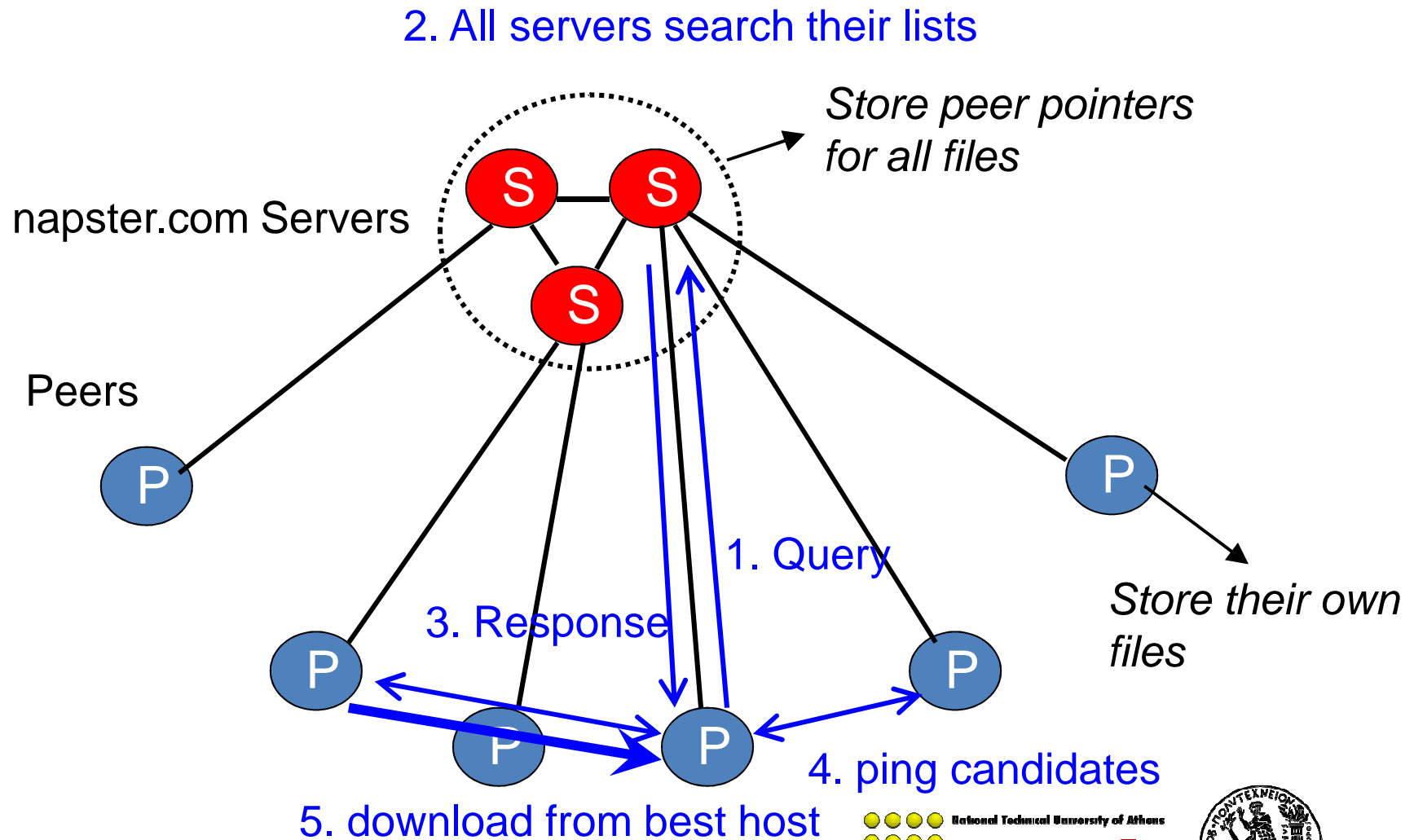
# Napster

*Store a directory, i.e.,  
filenames with peer pointers*

Filename	Metadata
Yesterday!.mp3	Beatles, @ 147.102.5.65:1006
PennyLane.mp3	Beatles, @ 128.84.92.23:1006
Help!.mp3	Beatles, @ 147.102.3.10:1006



# Napster



# Ιδιότητες

---

- Ο κατάλογος του server ανανεώνεται συνεχώς
  - Για να γνωρίζουμε πάντα τα διαθέσιμα αρχεία
  - Αδυναμία -> νομικές συνέπειες
- Peer-to-peer διακίνηση αρχείων
  - Ο server δεν υπερφορτώνεται
- Ειδικό πρωτόκολλο
  - Login, search, upload, download, status operations
- Θέματα bandwidth
  - Οι κόμβοι που έχουν τα αρχεία διατάσσονται με βάση το bandwidth & χρόνο απόκρισης
- Περιορισμός:
  - Κεντρικό lookup

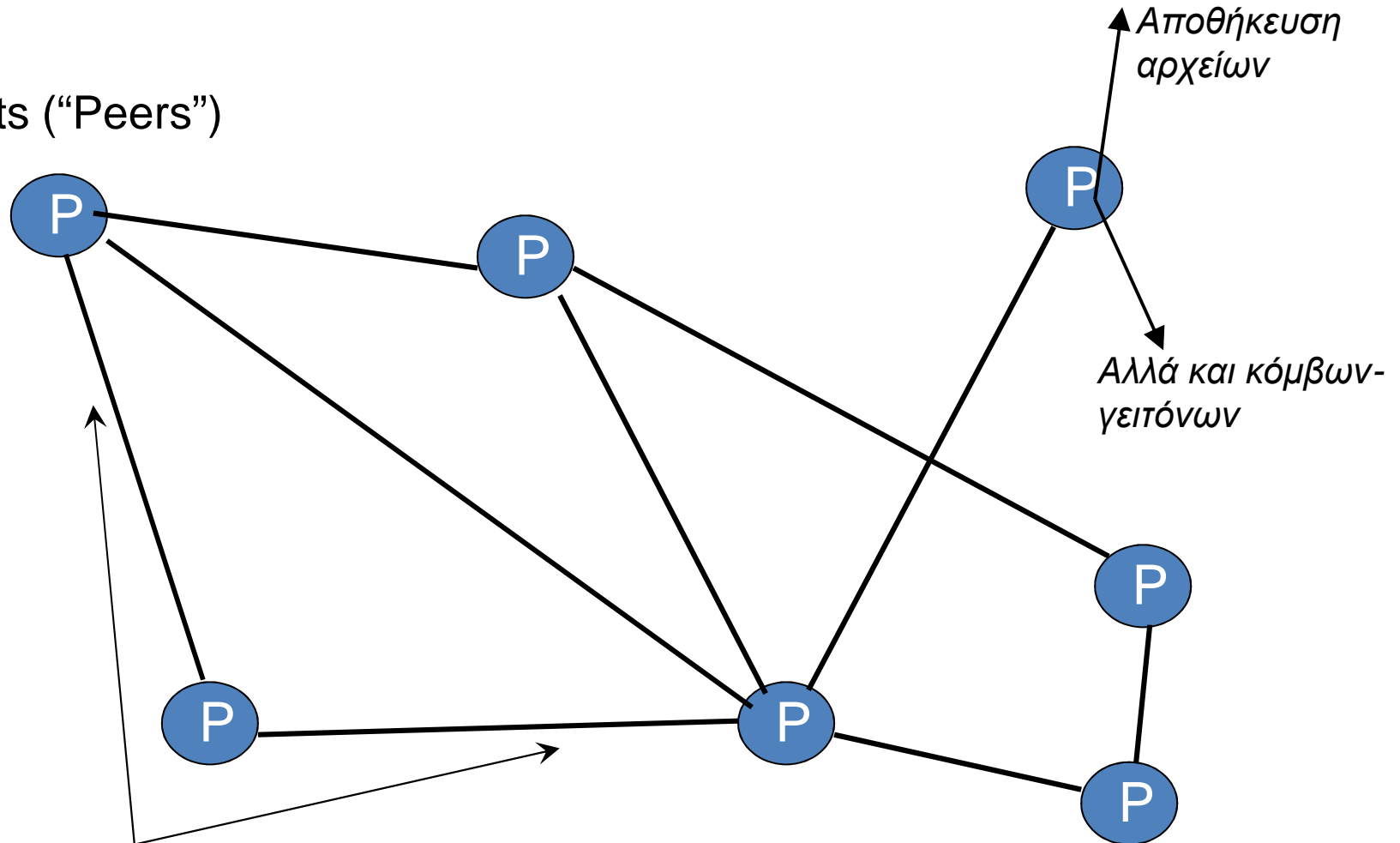
# gnutella Gnutella

---

- Όλοι οι κόμβοι πραγματικά ισότιμοι
  - Δεν υπάρχουν κόμβοι με ειδική λειτουργία
- Πλήρης αποκέντρωση
- Χρησιμοποιεί εικονικό δίκτυο επικάλυψης (overlay network)
  - Δικό του routing layer

# Gnutella

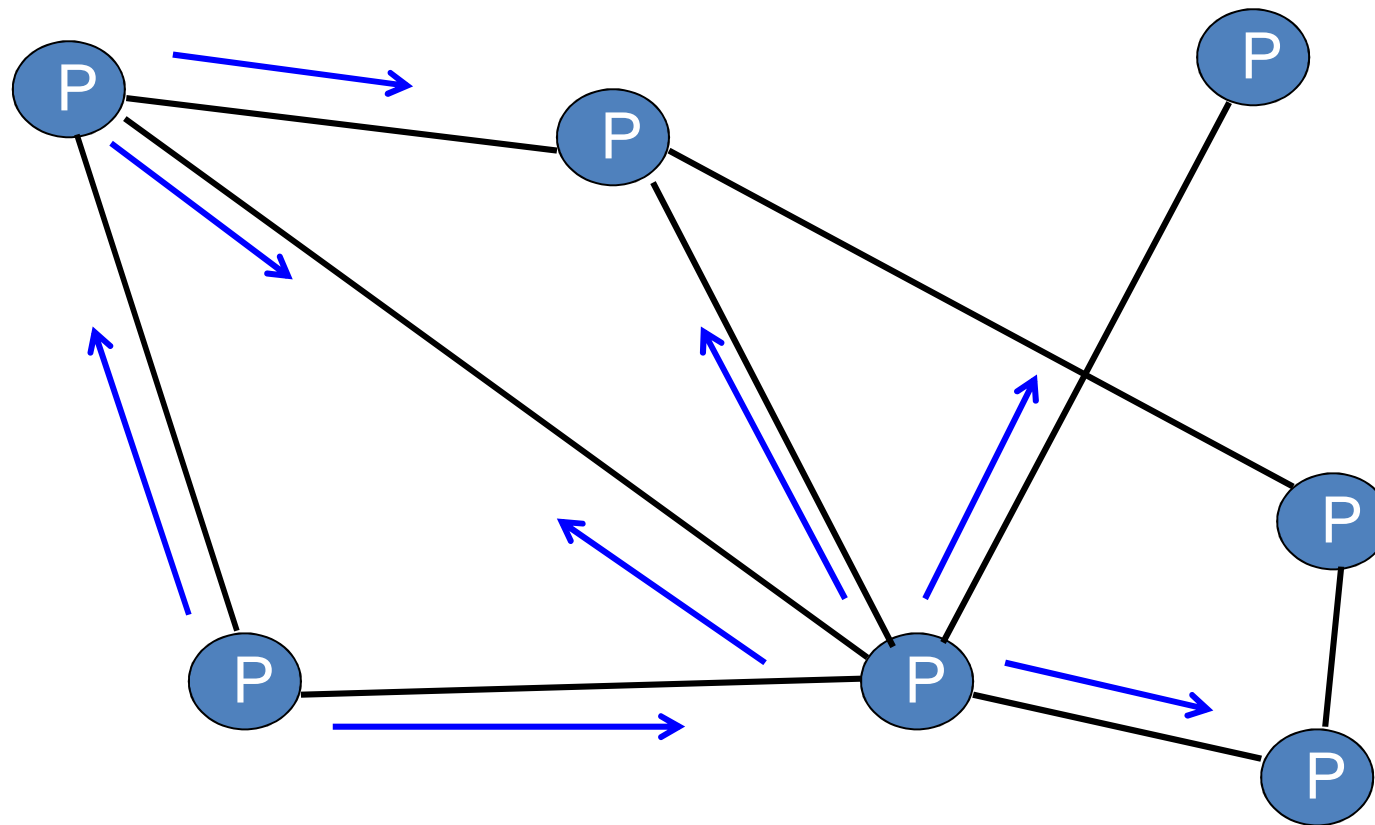
Servants ("Peers")



Connected in an overlay graph (== each link is an implicit Internet path)

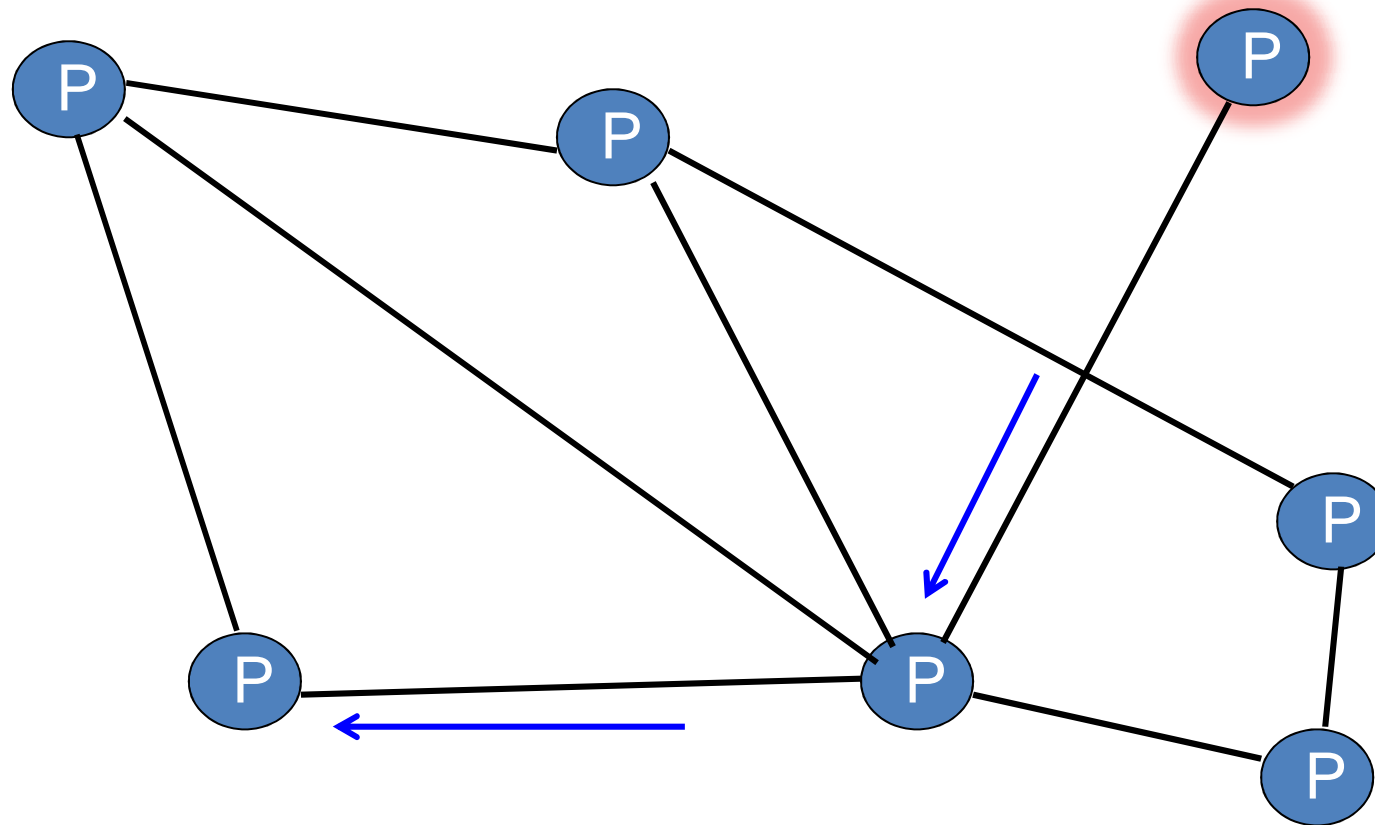
# Gnutella

Query's flooded out, ttl-restricted, forwarded only once



# Gnutella

Successful results QueryHit's routed on reverse path





# Ιδιότητες

---

- Πλεονεκτήματα
  - Πλήρως αποκεντρωμένο
  - Το κόστος της αναζήτησης διαμοιράζεται
  - Η επεξεργασία ανά κόμβο επιτρέπει ισχυρή σημασιολογία στην αναζήτηση
- Μειονεκτήματα
  - Η διάρκεια αναζήτησης μπορεί να είναι μεγάλη
  - Κόστος διατήρησης overlay, κόμβοι μπαινοβγαίνουν συχνά



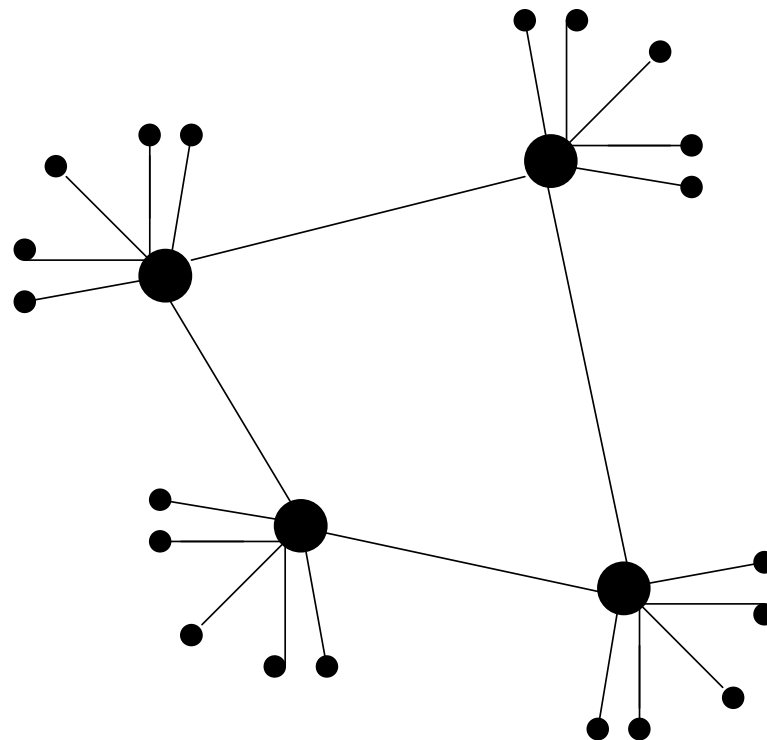
# Kazaa

---

- Κάτι ανάμεσα σε Napster & Gnutella
- Εισάγει την έννοια των supernodes
  - Εξυπηρετούν μικρό μέρος του δικτύου
    - Indexing
    - Caching
  - TCP ανάμεσα σε supernode και απλούς peers του υποδικτύου που εξυπηρετούν
  - TCP ανάμεσα σε supernodes
  - Επιλέγονται αυτόματα με βάση bandwidth και επεξεργαστική ισχύ
  - Όλες οι αναζητήσεις περνούν από αυτούς

# Kazaa

---



- ordinary peer
- supernode
- neighboring relationships in overlay network

# Ιδιότητες

---

- Ένας supernode αποθηκεύει καταλόγους
  - `<filename,peer pointer>` όπως οι servers του Napster
- Οι supernodes αλλάζουν
- Οποιοσδήποτε κόμβος μπορεί να γίνει (και να παραμείνει) supernode αρκεί να έχει καλή φήμη (reputation)
  - Kazaalite: participation level (=φήμη) ενός χρήστη από 0 έως 1000, επηρεάζεται από τη διάρκεια που ο χρήστης είναι συνδεδεμένος και τον αριθμό uploads
  - Υπάρχουν και πιο πολύπλοκες μέθοδοι υπολογισμού του reputation
- Ένας κόμβος αναζητά πάντα μέσω του πιο κοντινού του supernode

# Ειδική περίπτωση: BitTorrent

---

- Κίνητρο: δημοφιλή αρχεία
  - Η δημοφιλία παρουσιάζει temporal locality (Flash Crowds)
  - Π.χ., Slashdot effect, CNN Web site την 9/11, κυκλοφορία νέας ταινίας ή παιχνιδιού
- Στοχεύει στην αποδοτική μεταφόρτωση του αρχείου παρά στην αναζήτηση
  - Κατανομή του αρχείου σε πολλούς κόμβους ταυτόχρονα
  - Ένας εκδότης, πολλοί downloaders
- Αποφυγή free-riding
  - Μόνο κατεβάζω αρχεία αλλά δεν διαμοιράζω

# Βασική αρχή: Παράλληλο Downloading

---

- Διαίρεση αρχείου σε πολλά κομμάτια
  - Replication διαφορετικών κομματιών σε διαφορετικούς κόμβους
  - Ένας κόμβος μπορεί να ανταλλάξει τα κομμάτια που έχει με κομμάτια που δεν έχει από άλλους κόμβους
  - Οι κόμβοι μπορούν να ανακατασκευάσουν όλο το αρχείο από τα κομμάτια του
- Επιτρέπει ταυτόχρονα downloads
  - Διαφορετικά μέρη του αρχείου από διαφορετικούς κόμβους ταυτόχρονα
  - Ιδιαίτερα αποδοτικό για μεγάλα αρχεία
- System Components
  - Web server
  - Tracker
  - Peers

# Tracker

---

- Κόμβος με ειδικό ρόλο
  - Καταγράφει τους κόμβους που συμμετέχουν στο δίκτυο
- Οι κόμβοι εγγράφονται στον tracker
  - Κατά την άφιξή τους
  - Περιοδικά ενημερώνουν ότι είναι ακόμα στο δίκτυο
- Ο tracker επιλέγει τους κόμβους για downloading
  - Επιστρέφει τυχαίο υποσύνολο (με IP addresses)
  - Έτσι ένας εισερχόμενος κόμβος ξέρει με ποιους άλλους να συνδεθεί για να κατεβάσει ένα αρχείο
- Αντί για κεντρικό tracker μπορεί να χρησιμοποιήσει P2P δίκτυο (DHT)

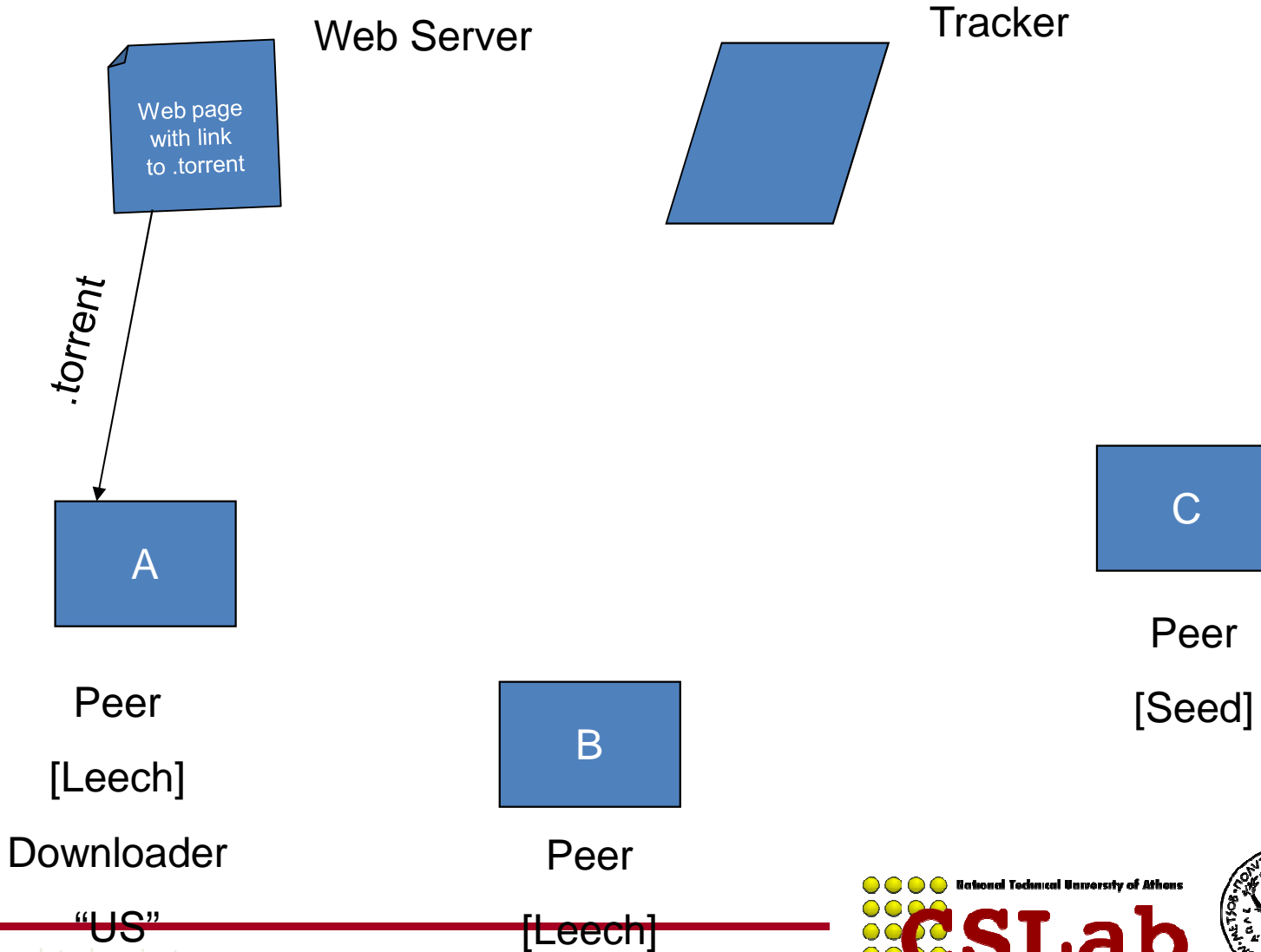
# Chunks

---

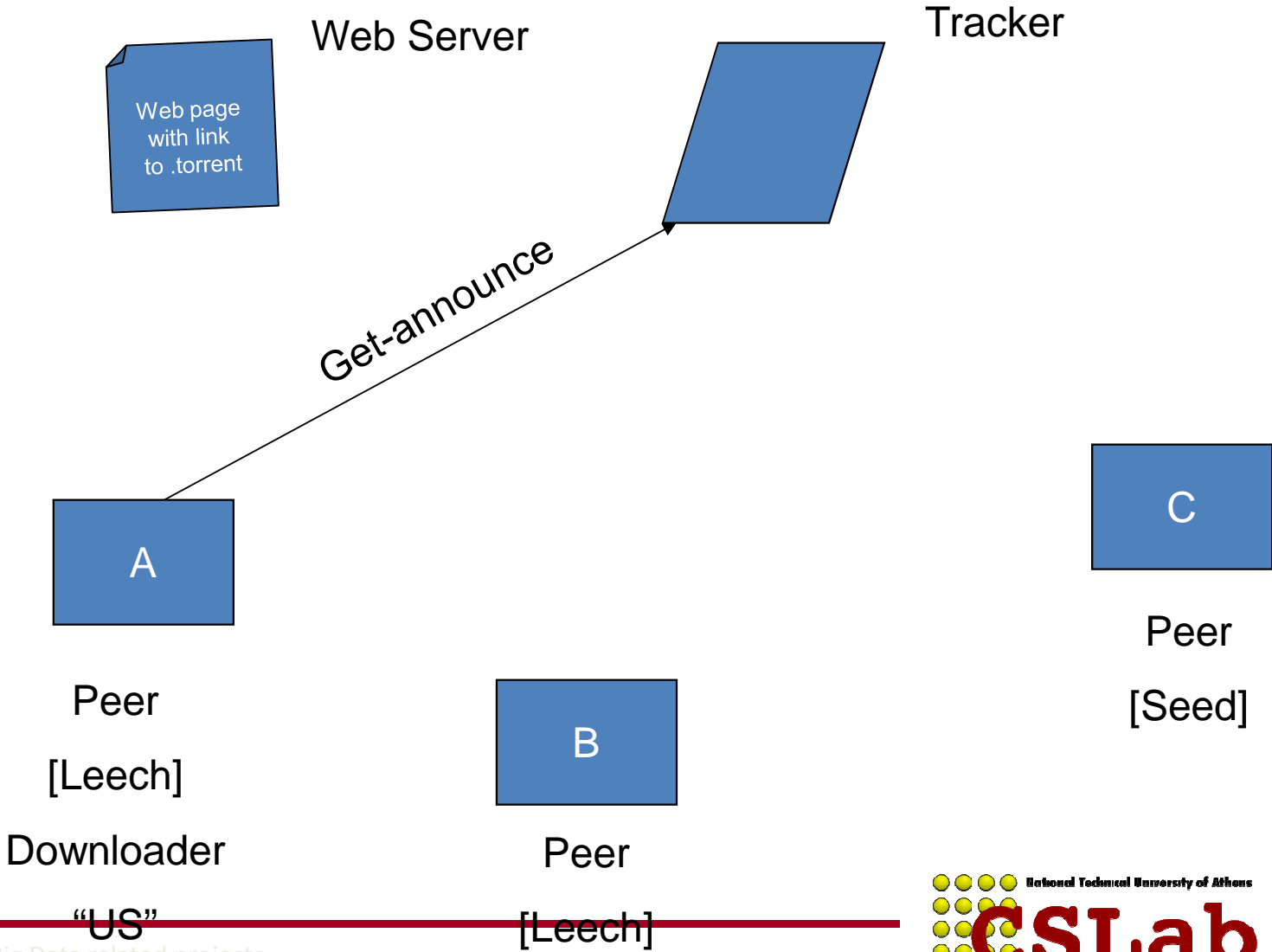
- Τα αρχεία διαιρούνται σε μικρότερα κομμάτια
  - Σταθερού μεγέθους, συνήθως 256 Kbytes
- Επιτρέπει ταυτόχρονη μεταφορά
  - Κατέβασμα chunks από διαφορετικούς κόμβους
  - Ανέβασμα chunks σε άλλους
- Γνώση για τα chunks που έχουν οι συνδεδεμένοι κόμβοι
  - Περιοδική μεταφορά λίστας με chunks
- Η μεταφόρτωση ολοκληρώνεται όταν κατέβουν όλα τα chunks ενός αρχείου



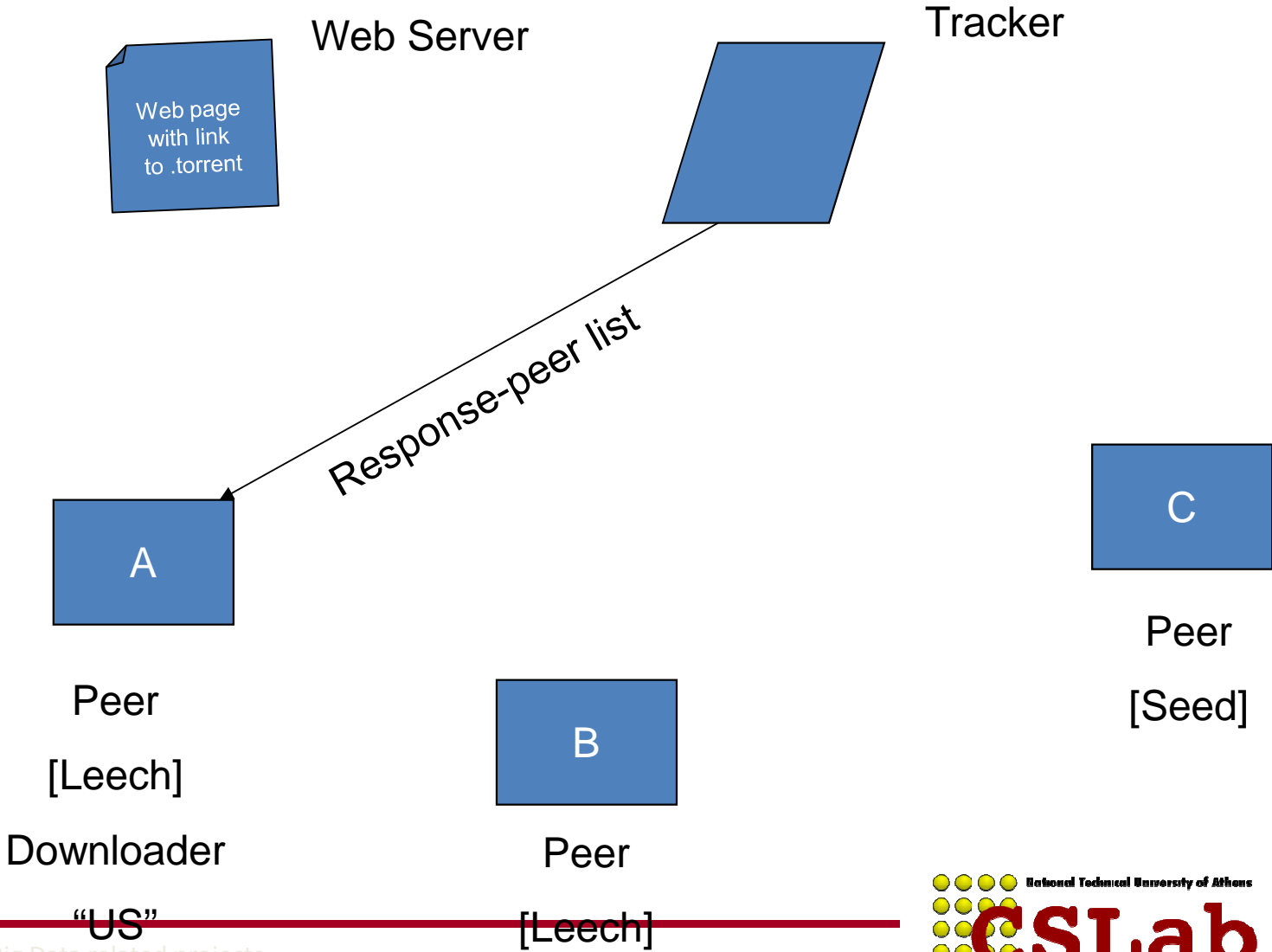
# BitTorrent



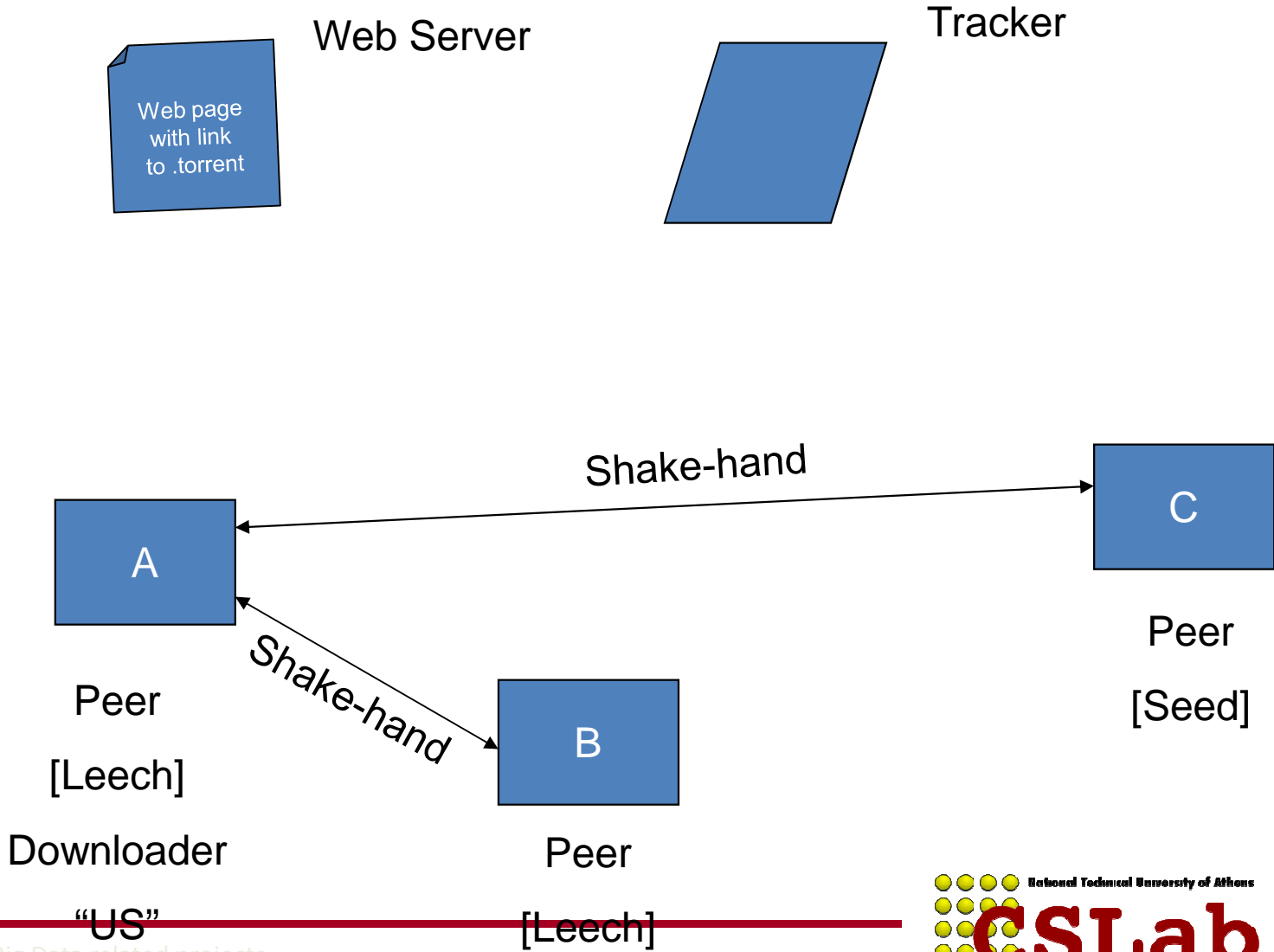
# BitTorrent



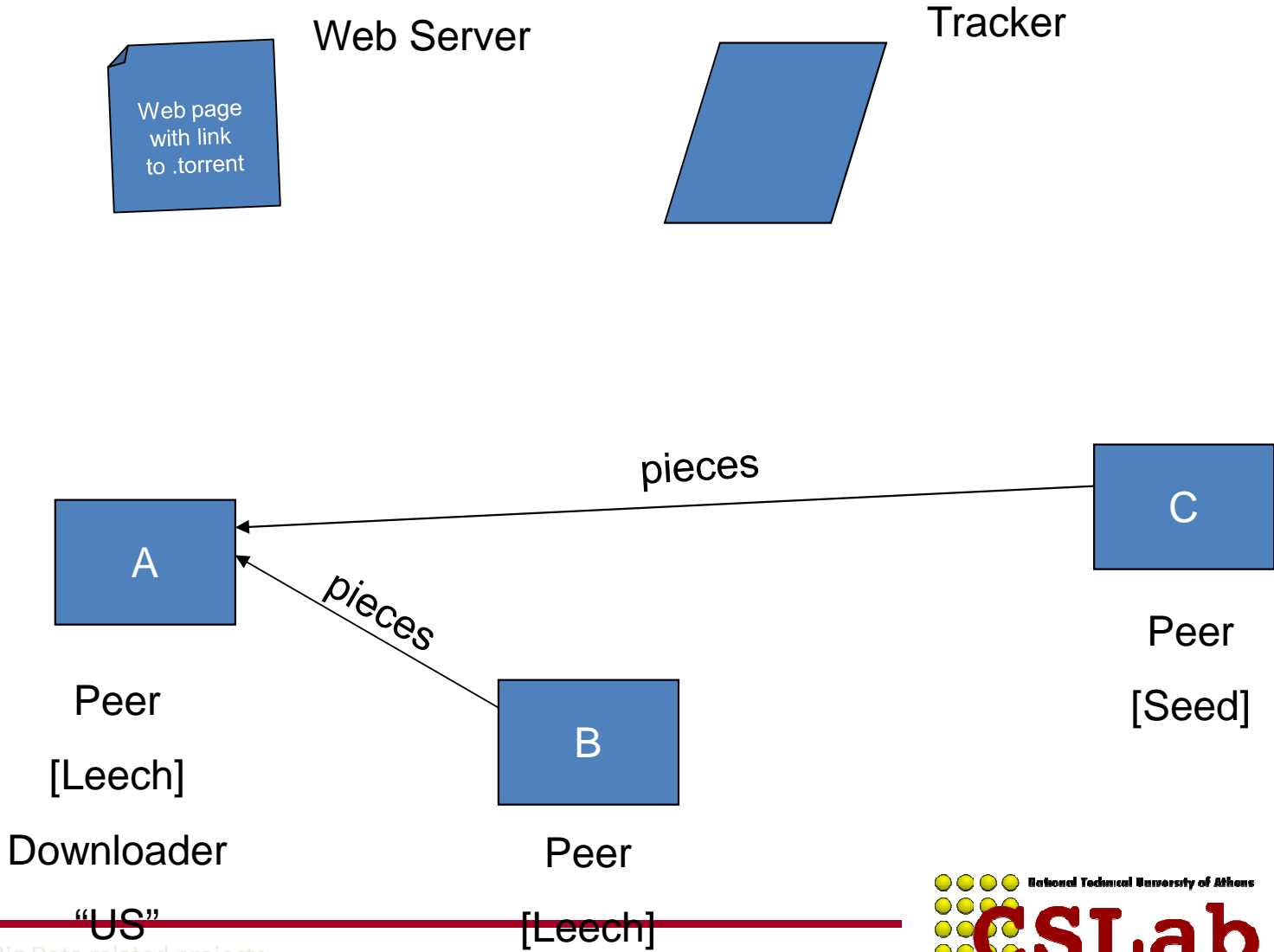
# BitTorrent



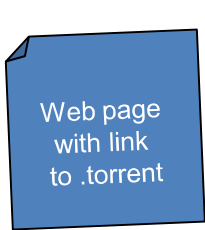
# BitTorrent



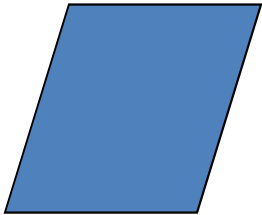
# BitTorrent



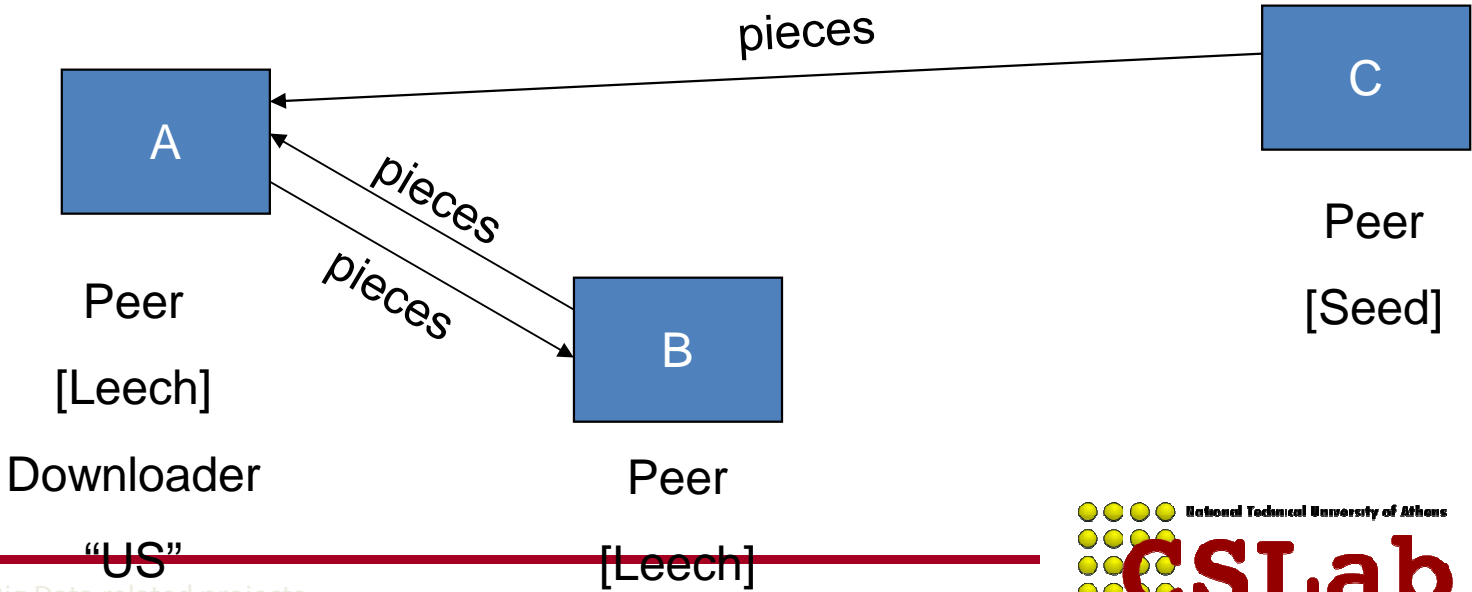
# BitTorrent



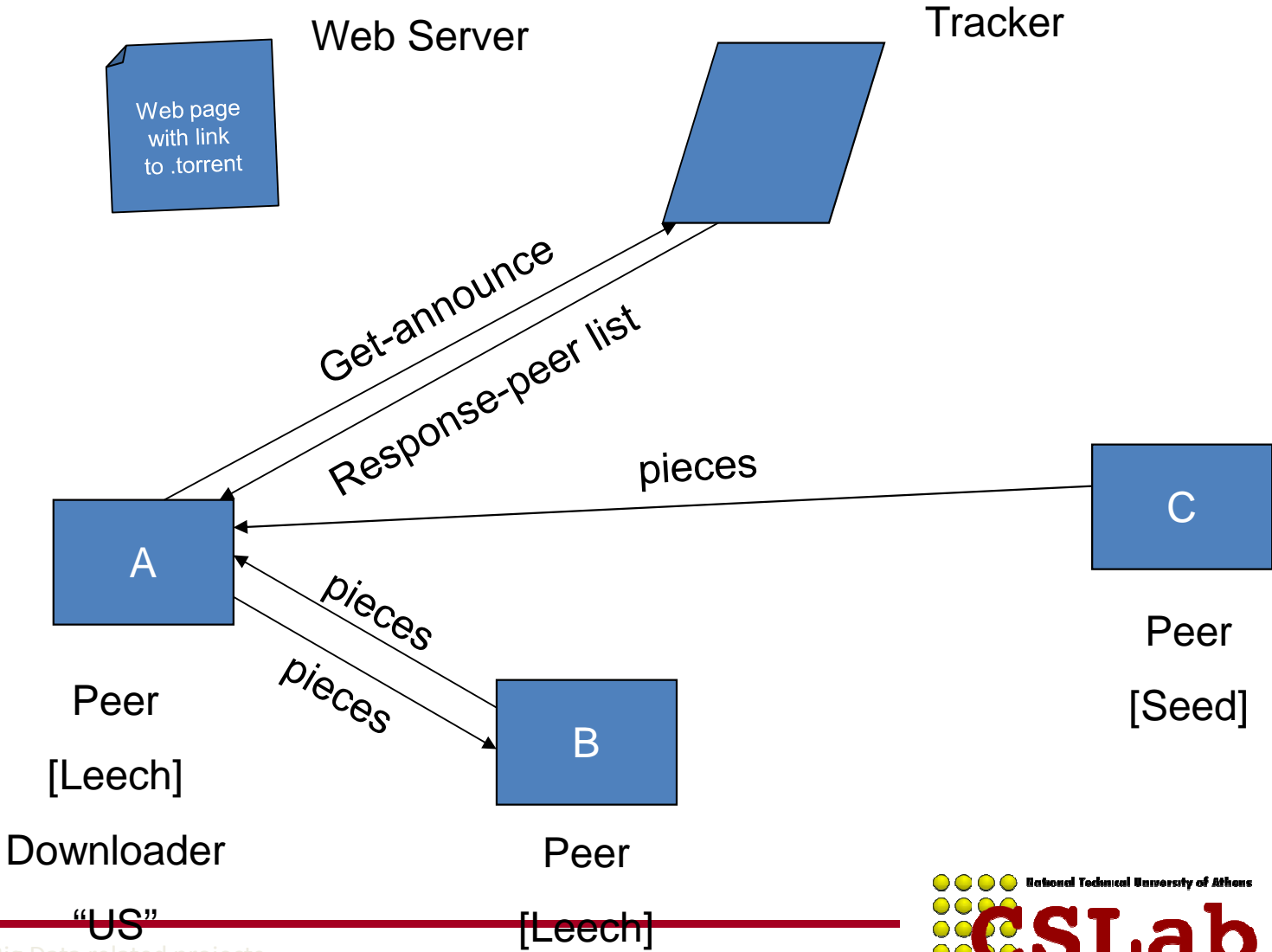
Web Server



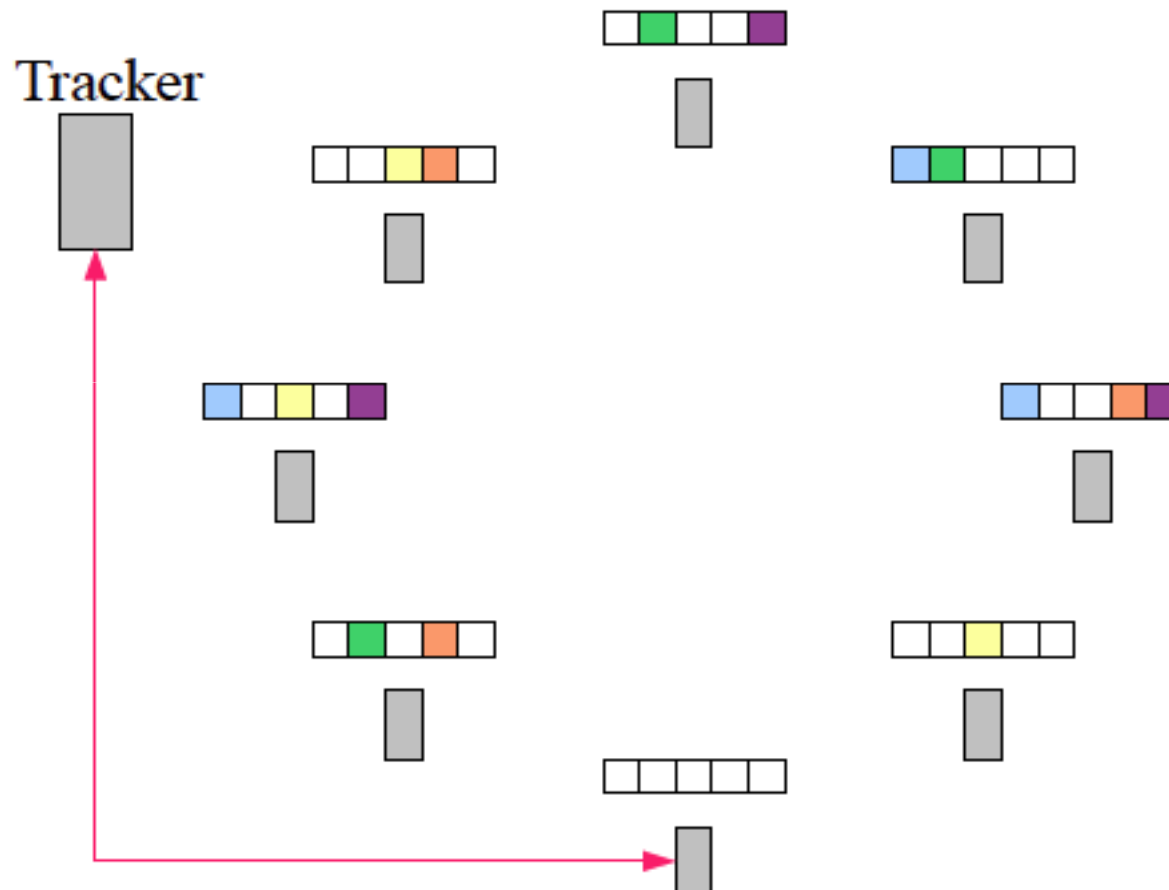
Tracker



# BitTorrent

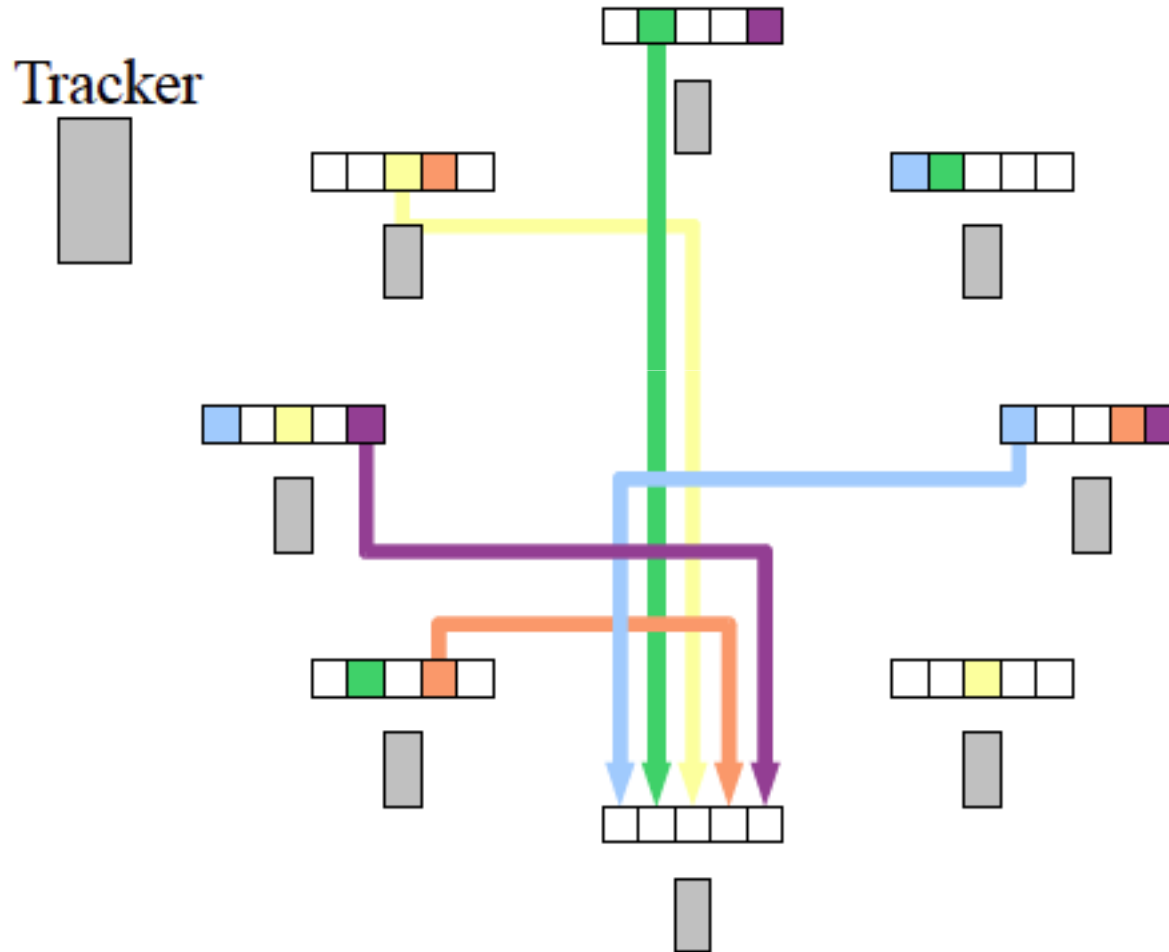


# Παράδειγμα





# Παράδειγμα



# Σειρά επιλογής chunk

---

- Ποια chunks να ζητήσω;
  - Μια λύση: με τη σειρά, όπως ένας HTTP client
- Πρόβλημα 1: πολλοί κόμβοι θα έχουν τα πρώτα chunks
  - Οι κόμβοι δε θα έχουν να μοιραστούν πολλά
  - Περιορίζεται η κλιμακωσιμότητα του συστήματος
- Πρόβλημα 2: Τελικά κανείς δεν έχει σπάνια chunks
  - Π.χ., τα chunks στο τέλος του αρχείου
  - Περιορίζεται η ικανότητα να κατέβει όλο το αρχείο
- Λύσεις: τυχαία επιλογή και επιλογή του σπανιότερου

# Προτεραιότητα στο σπανιότερο chunk

---

- Ποιο είναι το σπανιότερο;
  - Το chunk με τα λιγότερα διαθέσιμα αντίγραφα
  - Αυτό ζητείται πρώτα
- Πλεονέκτημα για τον κόμβο
  - Αποφεύγεται η λιμοκτονία όταν αποχωρούν πολλοί κόμβοι
- Πλεονέκτημα για το σύστημα
  - Διασφαλίζεται ότι δε θα χαθεί κάποιο chunk
  - Εξισορροπείται το φορτίο με την εξισορρόπηση του # των αντιγράφων των chunks

# Αποφυγή Free-Riding

---

- Η πλειονότητα των χρηστών είναι free-riders
  - Δε μοιράζονται αρχεία και δεν απαντούν σε ερωτήματα
  - Περιορίζουν τον αριθμό των συνδέσεων ή την ταχύτητα upload
- Λίγοι κόμβοι δρουν ως servers
  - Συνεισφέροντας στο «κοινό καλό»
- Το πρωτόκολλο BitTorrent σε έναν κόμβο
  - Επιτρέπει στους πιο γρήγορους κόμβους να κατεβάσουν από αυτόν
  - Κάποιες φορές επιτρέπουν σε τυχαίους κόμβους (και πιθανόν free riders) να κατεβάσουν

# Αποφυγή Free-Riding

---

- Ένας κόμβος έχει περιορισμένο upload bandwidth
  - Και πρέπει να το μοιράζεται με πολλούς
- Tit-for-tat
  - Δίνει προτεραιότητα σε κόμβους με μεγάλο upload rate
- Επιβραβεύει τους top 4 γείτονες
  - Μετράει το download bit rate του κάθε γείτονα
  - Ανταποδίδει στέλνοντας chunks στους 4 καλύτερους
  - Επαναλαμβάνει κάθε 10 sec
- Optimistic unchoking
  - Επιλέγει τυχαίο γείτονα για upload κάθε 30 sec
  - Δίνει τη δυνατότητα και σε κάποιον άλλον να κατεβάσει chunks

# Πειράζοντας το BitTorrent

---

- BitTorrent can be gamed, too
  - Οι κόμβοι ανεβάζουν στους top N peers με ρυθμό  $1/N$
  - Π.χ., αν  $N=4$  και οι κόμβοι ανεβάζουν με ρυθμό 15, 12, 10, 9, 8, 3
  - ... Τότε ο 4<sup>ος</sup> κόμβος έχει πλεονέκτημα
- Καλύτερα  $N^{\text{ος}}$  κόμβος στη λίστα από πρώτος

# BitTorrent σήμερα

---

- Σημαντικό μέρος της κίνησης στο Internet
  - Εκτίμηση: 30%
- Πρόβλημα για λιγότερο δημοφιλή αρχεία
  - Οι κόμβοι αποχωρούν όταν τελειώσουν το κατέβασμα
  - Κάποια αρχεία χάνονται ή δεν υπάρχουν ολόκληρα
- Παραμένουν νομικά θέματα

# Εντοπισμός δεδομένων

---

- Θέλουμε
  - Χαμηλό κόστος
  - Κλιμακωσιμότητα
  - Εγγύηση για το lookup

Napster	Central server
Gnutella & Kazaa	Network flooding Optimized to flood supernodes ... but it's still flooding
BitTorrent	Nothing! It's somebody else's problem



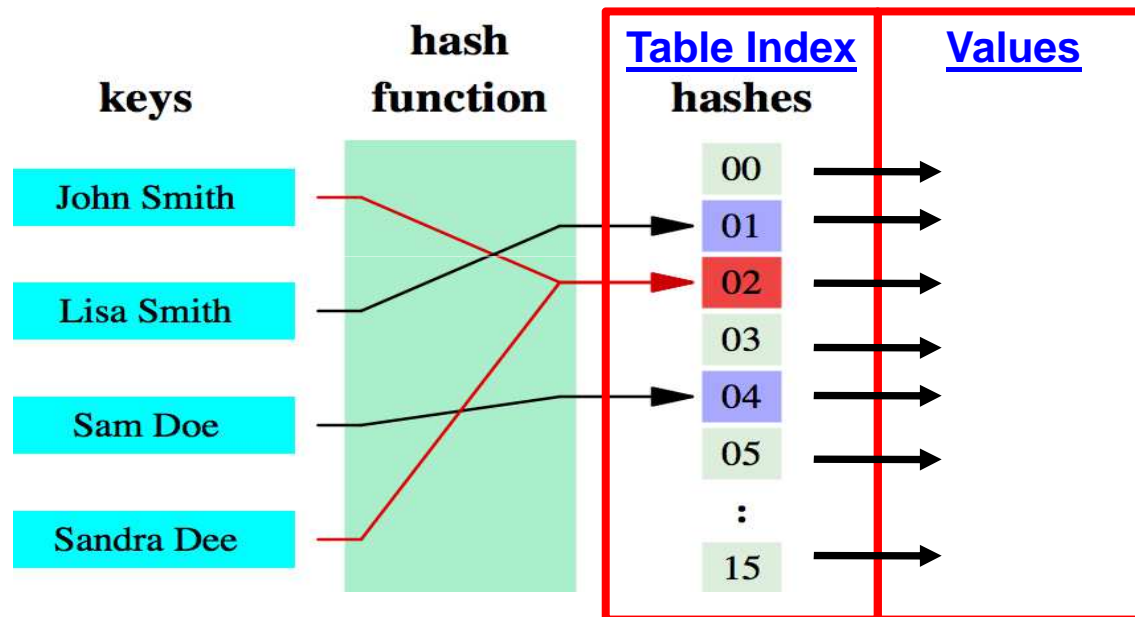
# Τι κακό έχει το flooding;

---

- Κάποιοι κόμβοι δεν είναι πάντα συνδεδεμένοι και κάποιοι είναι πιο αργοί από άλλους
  - Το Kazaa το αντιμετωπίζει κατηγοριοποιώντας τους κόμβους σαν supernodes
- Κακή χρήση των δικτυακών πόρων
- Μεγάλο latency
  - Τα αιτήματα προωθούνται από κόμβο σε κόμβο
  - Επιστροφή της απάντησης από το ίδιο μονοπάτι που πέρασε το ερώτημα στο Gnutella

# Τι θα θέλαμε;

- Hash table: Δομή που συνδέει κλειδιά με τιμές
- Κόστος αναζήτησης:  $O(1)$



- Ζεύγη (key, value)
  - (<http://www.cnn.com/foo.html>, Web page)
  - (Help!.mp3, 12.78.183.2)

# Τι είναι hash function

---

- Hash function

- Μια συνάρτηση που παίρνει είσοδο μεταβλητού μήκους (π.χ., ένα string) και παράγει ένα αποτέλεσμα (συνήθως μικρότερου) σταθερού μεγέθους (π.χ., έναν integer)
- Παράδειγμα: hash strings σε ακέραιους 0-7:
  - $hash("Αθήνα") \rightarrow 1$
  - $hash("Θεσσαλονίκη") \rightarrow 6$
  - $hash("Πάτρα") \rightarrow 2$

- Hash table

- Πίνακας από  $(key, value)$  ζεύγη
- Αναζήτηση κλειδιού:
  - Η hash function αντιστοιχίζει κλειδιά στο εύρος  $0 \dots N-1$   
table of  $N$  elements  
 $i = hash(key)$   
table[i] contains the item
- Αναζήτηση σε  $O(1)$

# Τι πρέπει να προσέξω

---

- Επιλογή καλού hash function
  - Θέλουμε ομοιόμορφη κατανομή όλων των κλειδιών στο εύρος  $0 \dots N-1$
- Συγκρούσεις
  - Πολλά κλειδιά μπορεί να αντιστοιχιστούν στην ίδια τιμή
    - hash("Paterson")  $\rightarrow 2$
    - hash("Edison")  $\rightarrow 2$
  - Η θέση table[i] είναι ένα bucket για όλα αυτά τα ζεύγη (*key, value*)
  - Μέσα στο table[i] χρησιμοποιούμε linked list ή άλλο ένα επίπεδο hashing
- Τι γίνεται όταν το hash table μεγαλώνει ή μικραίνει;
  - Αν προσθέσουμε ή αφαιρέσουμε buckets  $\rightarrow$  πρέπει να ξαναπεράσουμε τα κλειδιά από hash function και να μετακινήσουμε αντικείμενα

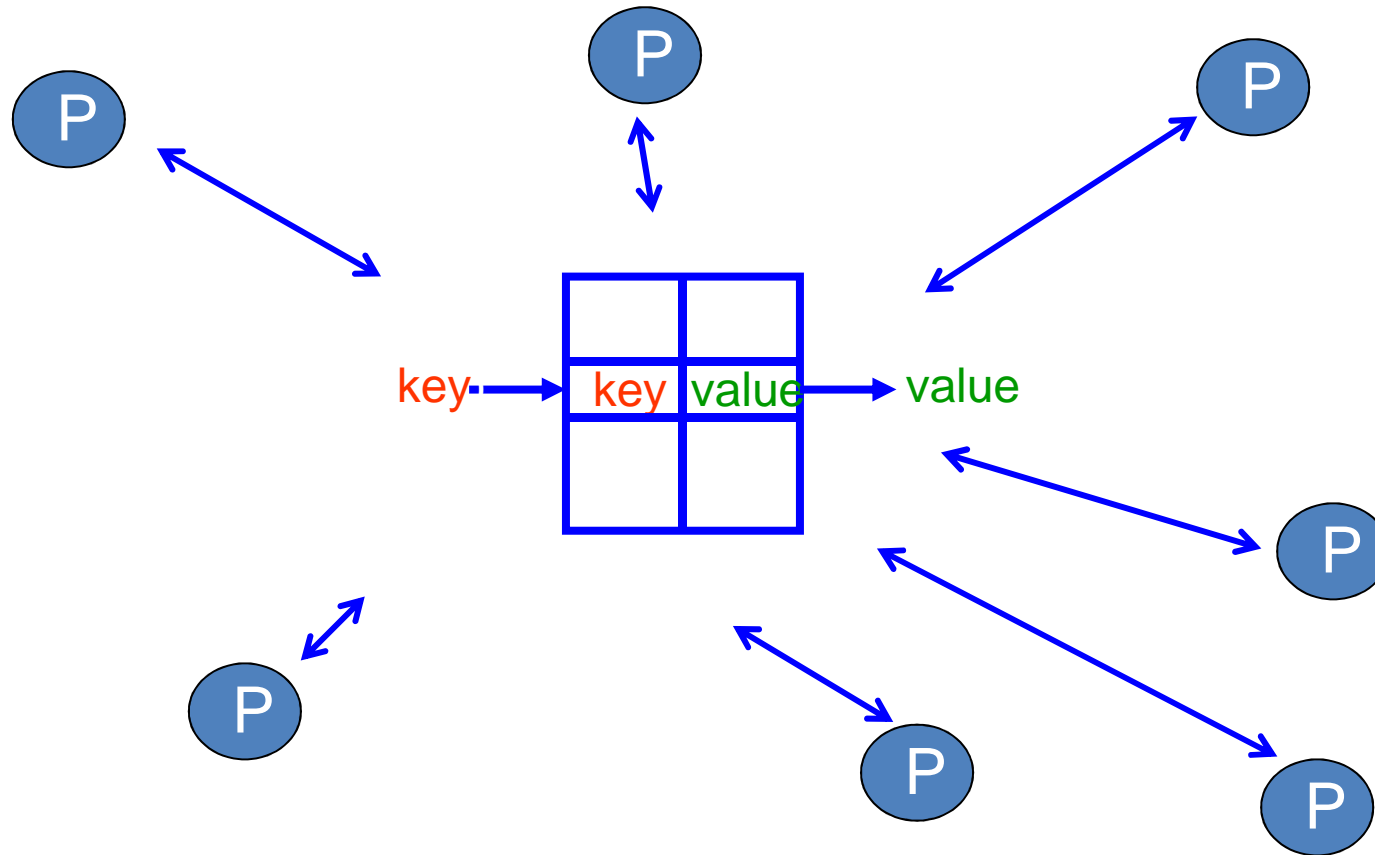
# Distributed Hash Tables (DHT)

---

- Δημιουργία peer-to-peer εκδοχής μιας βάσης (*key, value*)
- Πώς θέλουμε να λειτουργεί
  1. Ένας κόμβος (*A*) ρωτάει τη βάση με ένα κλειδί
  2. Η βάση βρίσκει τον κόμβο (*B*) που έχει την τιμή
  3. Ο κόμβος (*B*) επιστρέφει το ζεύγος (*key, value*) στον (*A*)
- Πρέπει να γίνει αποδοτικά
  - Όχι με flooding

# DHT

---



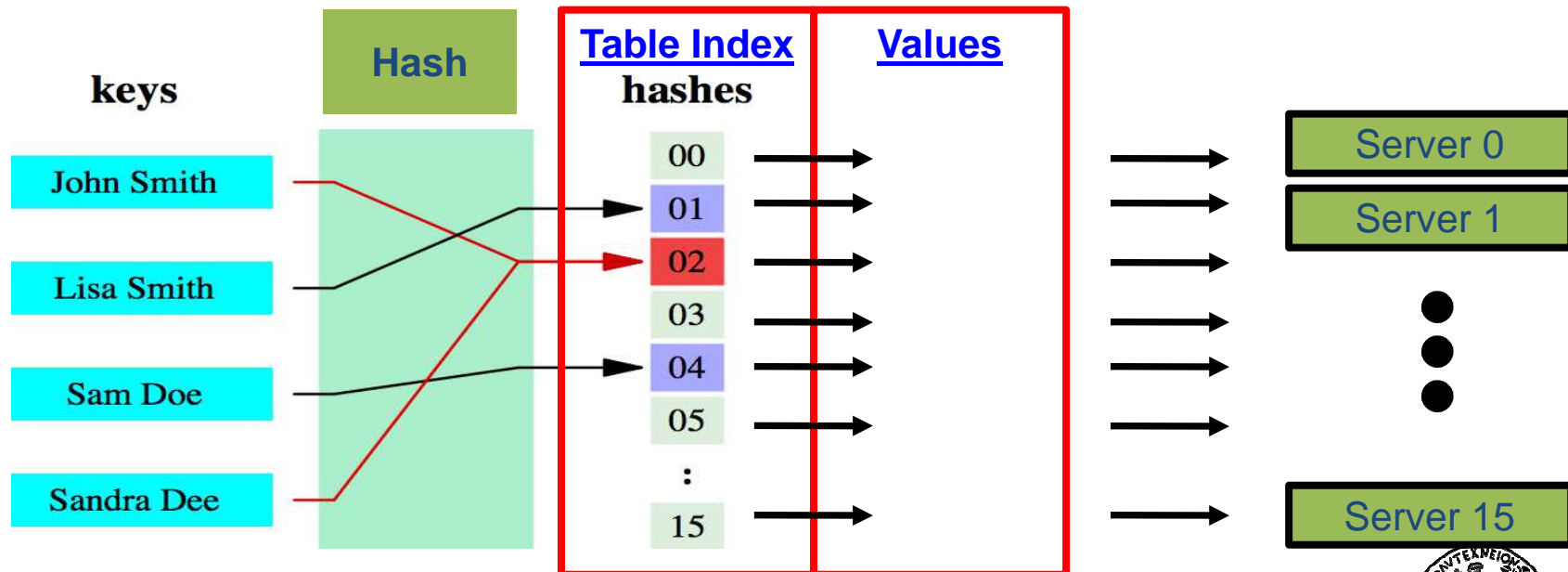
# DHT

---

- Ποιες είναι οι απαιτήσεις
  - Ντετερμινιστικό lookup
  - Μικρός χρόνος lookup (δεν πρέπει να αυξάνεται γραμμικά με το μέγεθος του συστήματος)
  - Εξισορρόπηση φορτίου ακόμα και με εισόδους/αποχωρήσεις κόμβων
- Τι κάνουμε: **διαιρούμε το hash table και το κατανέμουμε στους κόμβους του συστήματος**
- Πρέπει να διαλέξουμε το σωστό **hash function**
- Πρέπει να χωρίσουμε τον πίνακα και να κατανείμουμε τα κομμάτια με το ελάχιστο κόστος επανατοποθέτησης σε περίπτωση εισόδου/αποχώρησης κόμβου

# Χρήση βασικού Hashing

- Τοποθέτηση  $X$  στον server  $i = \text{hash}(X) \bmod k$
- Πρόβλημα;
  - Τι συμβαίνει όταν ένας κόμβος πεθάνει ή μπει στο σύστημα ( $k \rightarrow k \pm 1$ )?
  - Όλες οι τιμές αντιστοιχίζονται ξανά σε νέους κόμβους!

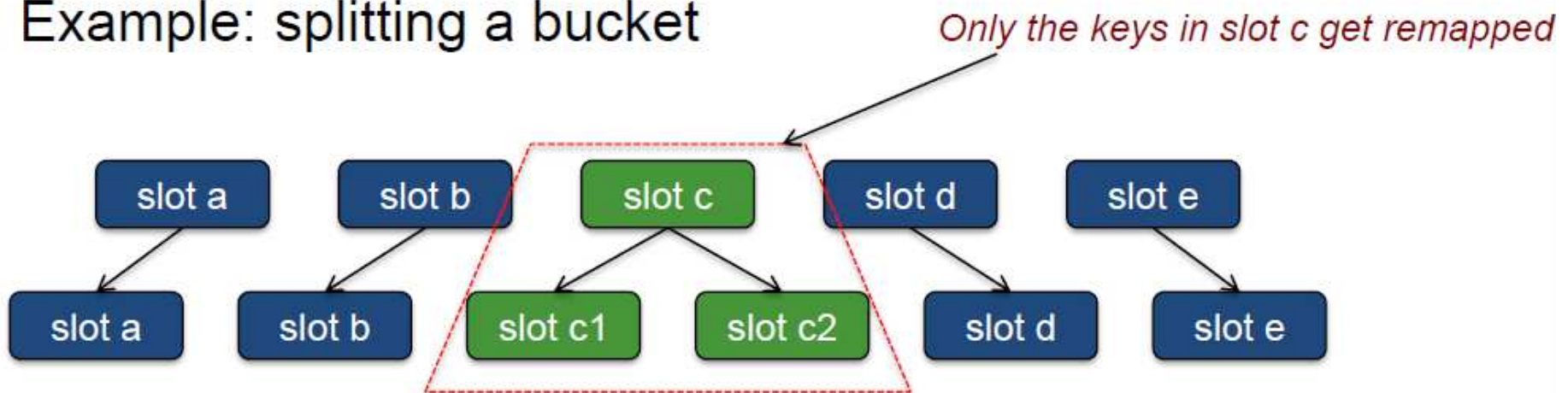




# Consistent hashing

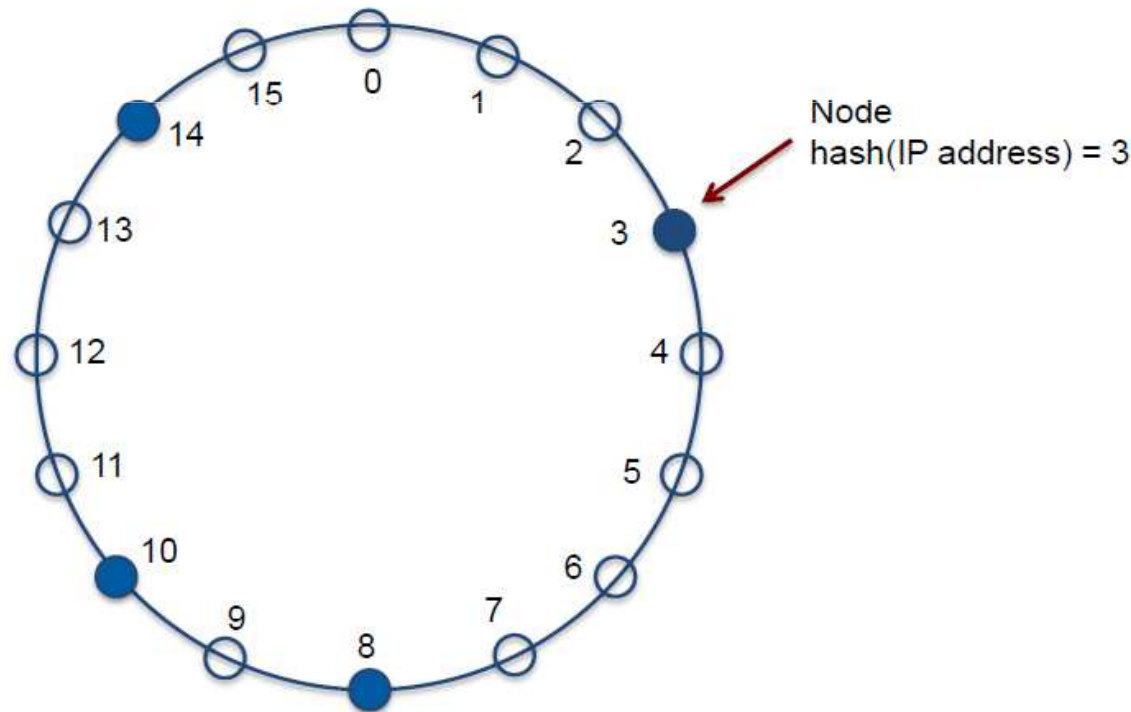
- Consistent hashing
  - Τα περισσότερα κλειδιά θα αντιστοιχιστούν στην ίδια τιμή όπως πριν
  - Κατά μέσο όρο  $K/n$  κλειδιά θα πρέπει να αντιστοιχιστούν ξανά
- $K = \# \text{ keys}$ ,  $n = \# \text{ of buckets}$

Example: splitting a bucket



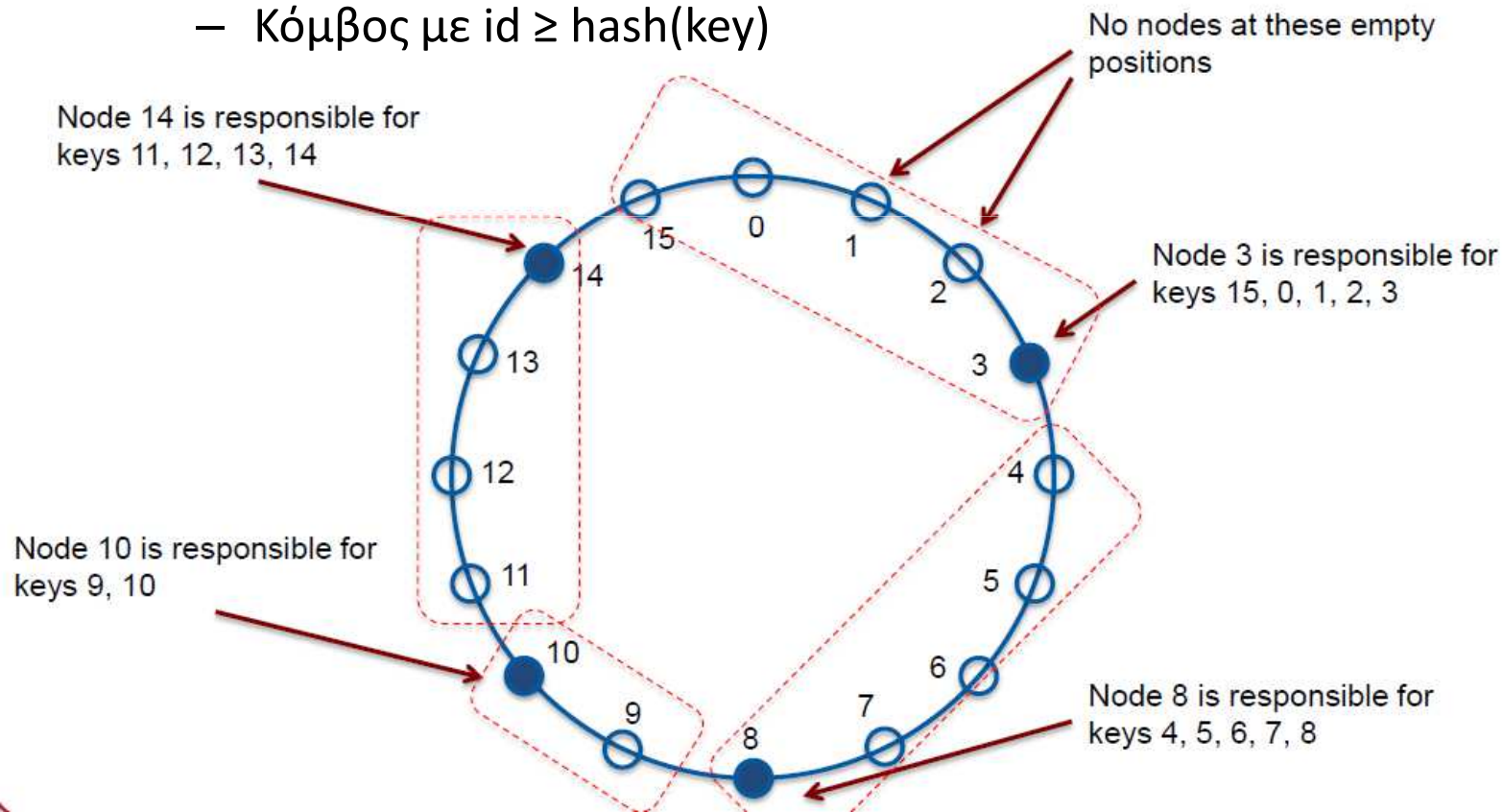
# Παράδειγμα DHT: Chord

- Ένα key γίνεται hash σε μια τιμή με  $m$ -bits :  $0 \dots (2^m-1)$
- Κατασκευάζεται ένας λογικός δακτύλιος για τις τιμές  $0 \dots (2^m-1)$
- Οι κόμβοι τοποθετούνται στον δακτύλιο στη θέση  $hash(IP)$



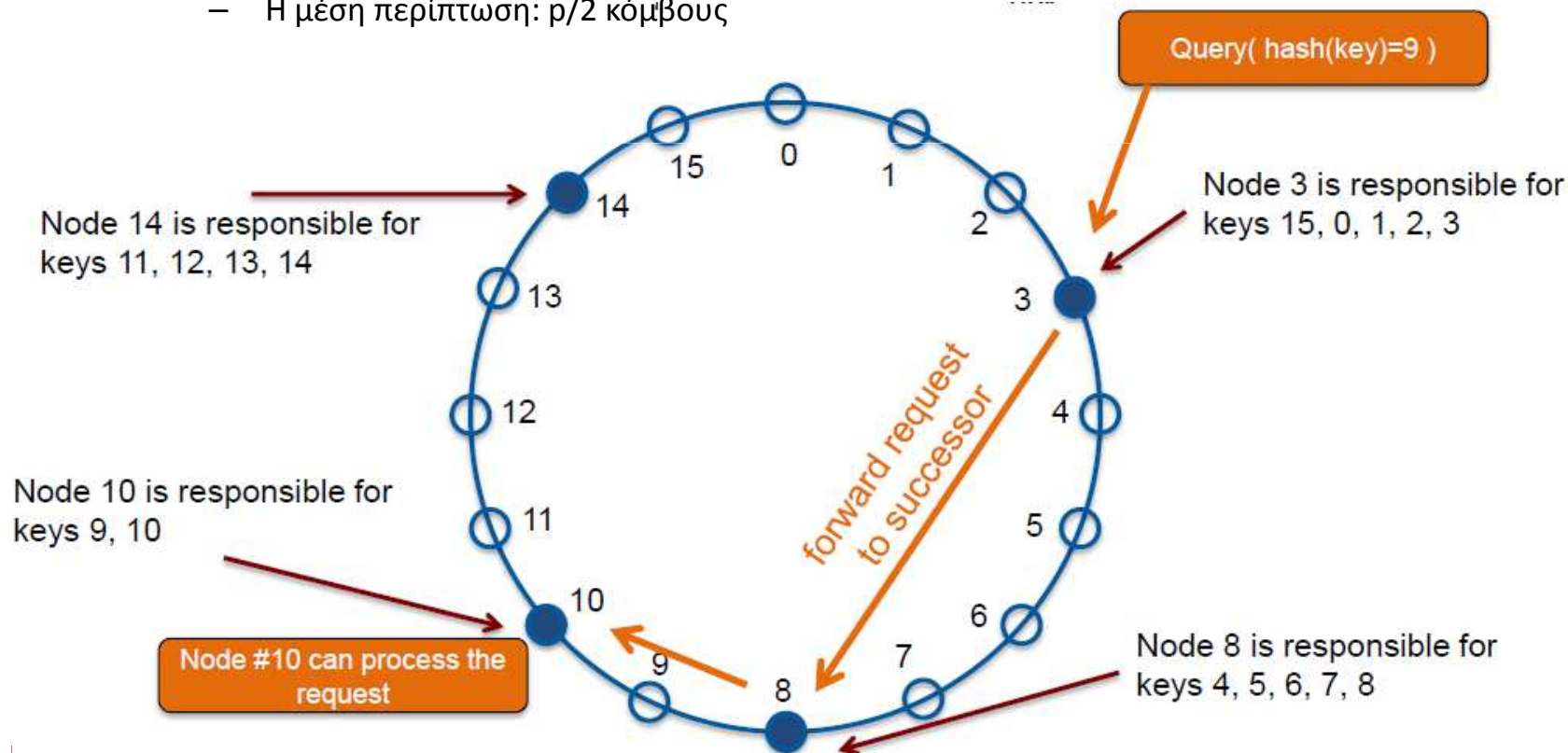
# Τοποθέτηση κλειδιών

- Παράδειγμα:  $n=16$ ; Σύστημα με 4 κόμβους (μέχρι τώρα)
- Τα δεδομένα αποθηκεύονται στον διάδοχο κόμβο (**successor**)
  - Κόμβος με  $id \geq \text{hash}(\text{key})$



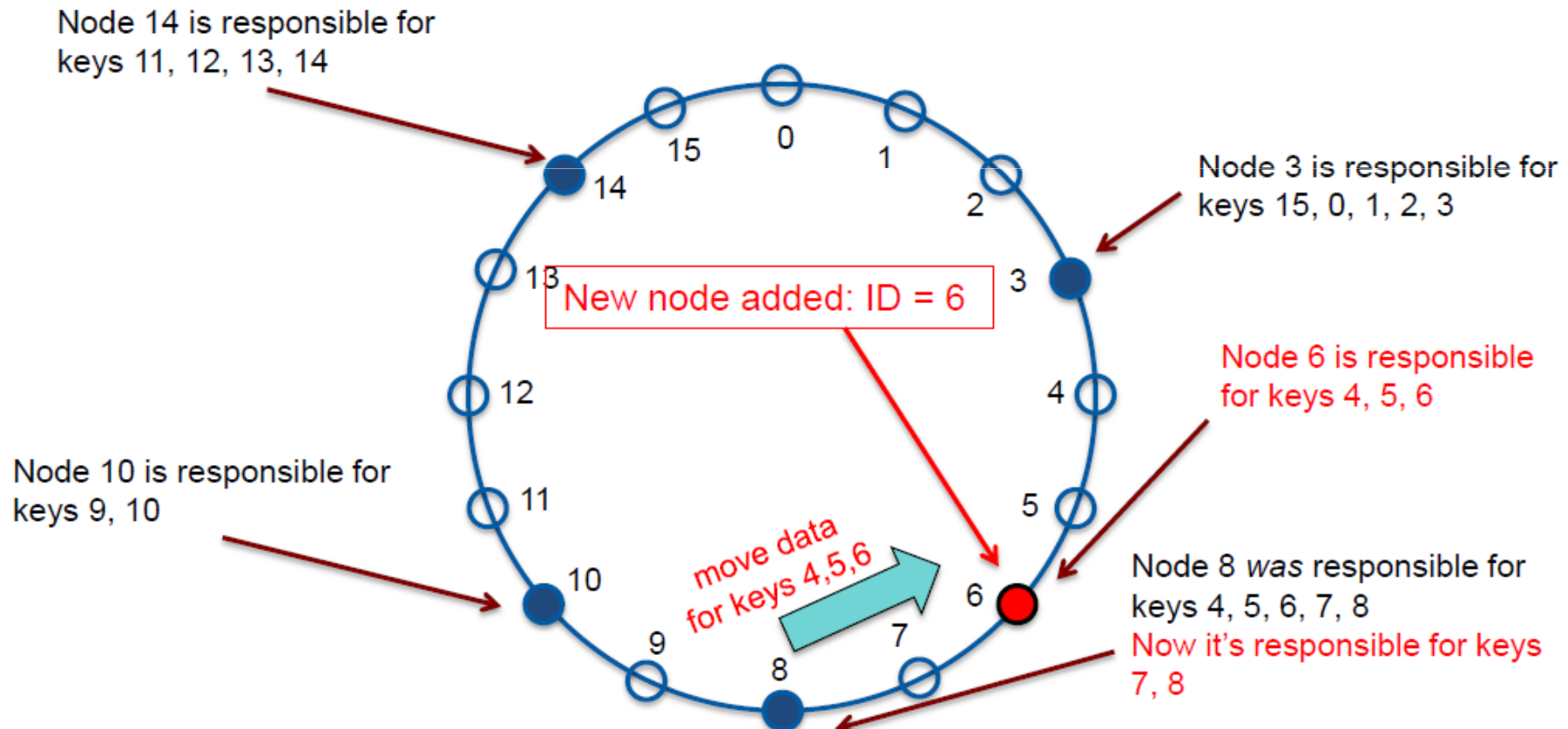
# Χειρισμός ερωτημάτων

- Οποιοσδήποτε κόμβος μπορεί να λάβει ερώτημα για ένα key (insert ή query). Αν το  $\text{hash}(\text{key})$  δεν ανήκει στο εύρος κλειδιών του, το προωθεί στον διάδοχο
- Η διαδικασία συνεχίζεται μέχρι να βρεθεί ο υπεύθυνος κόμβος
  - Η χειρότερη περίπτωση: με  $p$  κόμβους, διασχίζει  $p-1 \rightarrow O(N)$
  - Η μέση περίπτωση:  $p/2$  κόμβους



# Εισαγωγή κόμβου

- Κάποια κλειδιά ανατίθενται σε νέο κόμβο
- Τα δεδομένα για αυτά τα ζεύγη (key, value) πρέπει να μεταφερθούν στον νέο κόμβο



# Αποχώρηση κόμβου

- Τα κλειδιά ανατίθενται στον διάδοχο του κόμβου
- Τα δεδομένα για αυτά τα ζεύγη (key, value) πρέπει να μεταφερθούν

Node 14 was responsible for keys 11, 12, 13, 14

Node 14 is now responsible for keys 9, 10, 11, 12, 13, 14

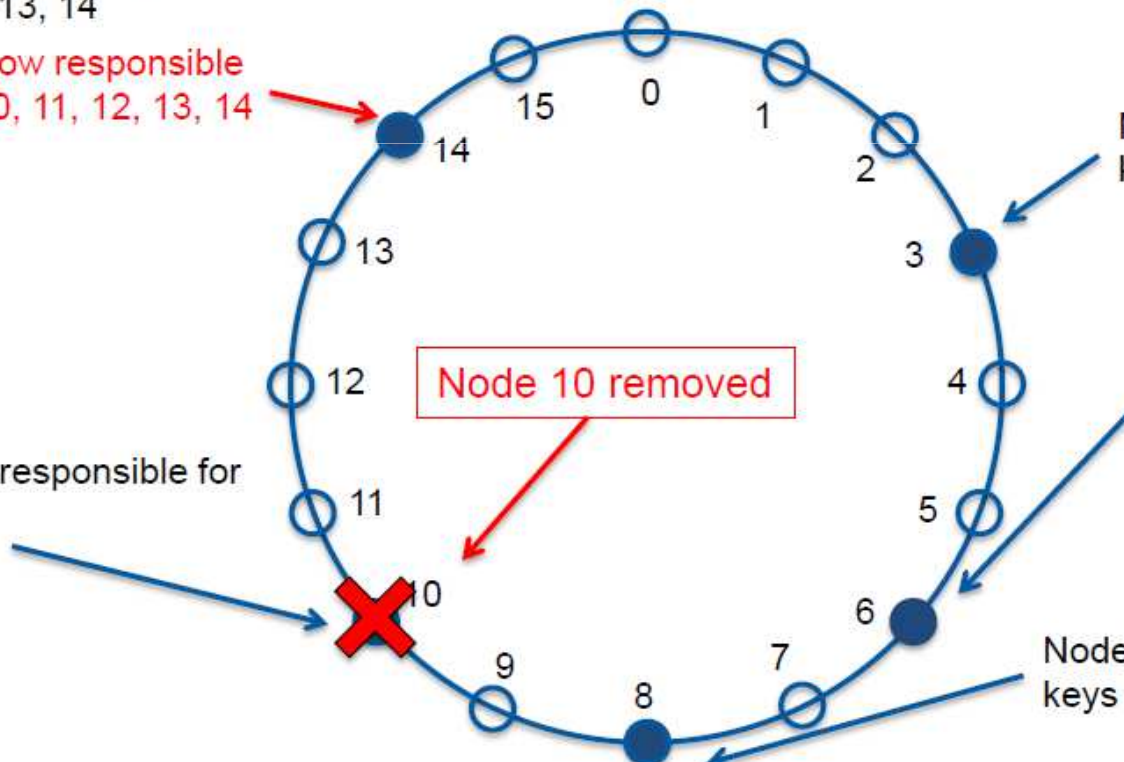
Node 3 is responsible for keys 15, 0, 1, 2, 3

Node 6 is responsible for keys 4, 5, 6

Node 8 is responsible for keys 7, 8

Node 10 was responsible for keys 9, 10

Node 10 removed



# Ανοχή σε σφάλματα

---

- Κόμβοι μπορεί να πεθάνουν
  - (key, value) δεδομένα replicated
  - Δημιουργία R replicas που αποθηκεύονται στους R-1 διαδοχικούς κόμβους στον δακτύλιο
- Γίνεται λίγο πιο περίπλοκο
  - Κάθε κόμβος πρέπει να ξέρει και τον διάδοχο του διαδόχου του (ή και παραπάνω από έναν)
- Εύκολο αν γνωρίζει και τους R-1
  - Οποιαδήποτε αλλαγή πρέπει να εξαπλωθεί σε όλα τα replicas

# Απόδοση

---

- Δεν μας αρέσει το  $O(N)$  κόστος του lookup
- Απλή προσέγγιση για καλή απόδοση
  - Όλοι οι κόμβοι γνωρίζονται μεταξύ τους
  - Όταν ένας κόμβος λάβει ερώτημα για key ψάχνει στον πίνακα δρομολόγησης να βρει τον κόμβο που είναι υπεύθυνος για το key
  - Απόδοση  $O(1)$
  - Κατά την εισαγωγή/αποχώρηση κόμβων πρέπει να ενημερωθούν όλοι
  - Όχι πολύ καλή λύση για τεράστια δίκτυα (πολλοί κόμβοι -> μεγάλοι πίνακες δρομολόγησης)



# Finger Tables

---

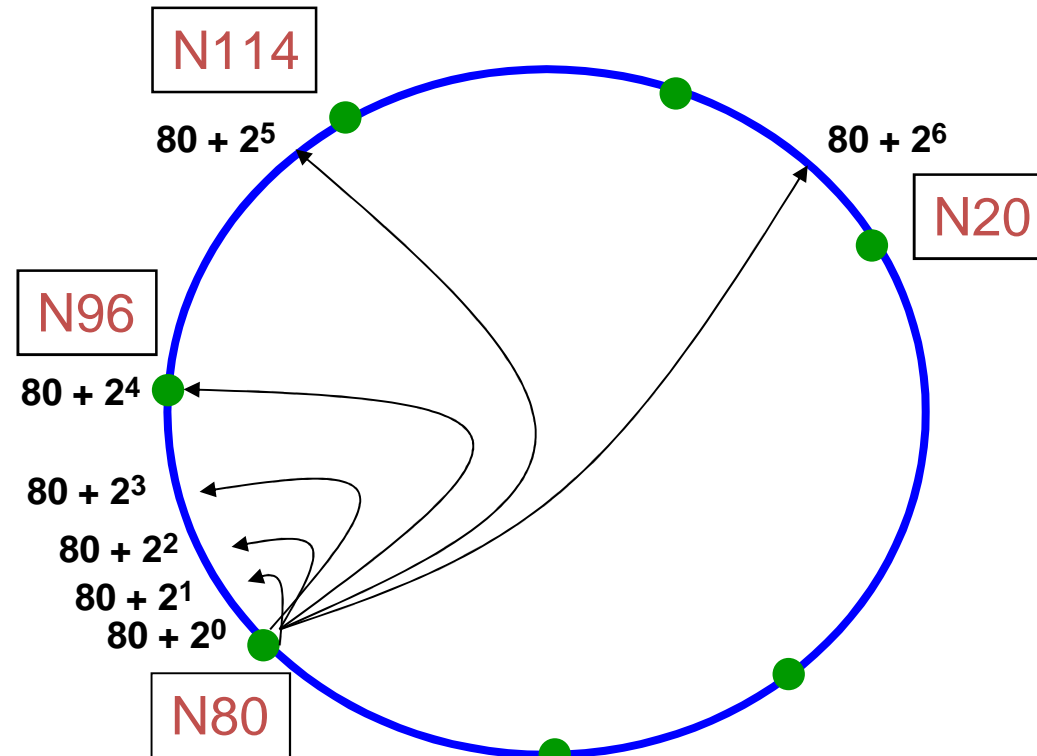
- Συμβιβασμός για αποφυγή μεγάλων πινάκων δρομολόγησης σε κάθε κόμβο
  - Χρήση finger tables για άνω όριο στο μέγεθος του πίνακα
- Finger table = μερική λίστα κόμβων
- Σε κάθε κόμβο το  $i^{\text{th}}$  στοιχείο είναι ο κόμβος που ακολουθεί κατά τουλάχιστον  $2^i-1$  στον κύκλο
  - finger\_table[0]: immediate (1st) successor
  - finger\_table[1]: successor after that (2nd)
  - finger\_table[2]: 4th successor
  - finger\_table[3]: 8th successor
- $O(\log N)$  κόμβοι προσπελάζονται για ένα lookup

# Παράδειγμα finger table

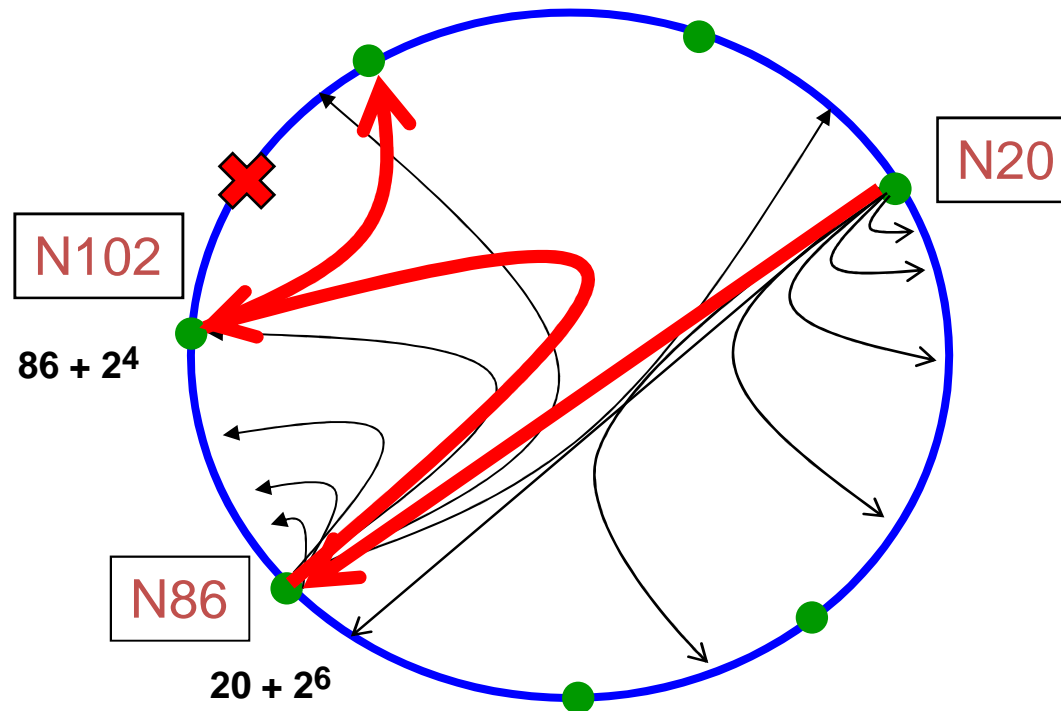
- Ids 0-120  $\rightarrow 2^7 = 128 > 120$  οπότε 7 fingers
- $f_n(i) = \text{Successor}(n + 2^i \text{ mod } 120)$

Finger Table at N80

$i$	$ft[i]$
0	96
1	96
2	96
3	96
4	96
5	114
6	20



# Εύρεση key με finger table



# Δομημένα P2P

---

- Τα DHT συστήματα είναι δομημένα (structured) P2P συστήματα σε αντίθεση με τα αδόμητα (unstructured) P2P όπως το Napster, Gnutella, Kazaa κλπ.
- Χρησιμοποιούνται ως βάση για άλλα συστήματα, όπως “trackerless” BitTorrent, Amazon Dynamo, κατανεμημένα file systems, κατανεμημένα repositories κλπ.

# Amazon Dynamo

---

- Κατανεμημένο key-value storage
  - Προσβάσιμο μόνο με το key
  - put(key, value) & get(key)
- Χρησιμοποιείται για πολλά Amazon services
  - Shopping cart, best seller lists, customer preferences, product catalog...
  - Δεν υπάρχει ανάγκη για περίπλοκα ερωτήματα όπως αυτά που απαντάει ένα RDBMS
    - Θα περιορίζει κλιμακωσιμότητα και διαθεσιμότητα
  - Πλέον προσφέρεται και ως AWS (DynamoDB)
- Μαζί με άλλα συστήματα της Google (GFS & Bigtable) το Dynamo είναι από τα πρώτα non-relational storage systems (a.k.a. NoSQL)

# Κίνητρο

---

- Shopping cart service
  - 3 million checkouts κάθε μέρα
  - Εκατοντάδες χιλιάδες ταυτόχρονα sessions
- Οι εφαρμογές πρέπει να μπορούν να παραμετροποιήσουν το Dynamo για επιθυμητό latency & throughput
  - Τουλάχιστον 99.9% των read/write λειτουργιών σε λιγότερο από 1 sec
  - Ισορροπία ανάμεσα σε performance, cost, availability, durability guarantees.
- Ανάγκη για availability (θυμήσου το θεώρημα CAP)
  - Eventual consistency
  - Partition tolerance
  - Availability (“always-on” experience)

# Απαιτούμενα κομμάτια

---

- Θέλουμε να σχεδιάσουμε μια υπηρεσία storage σε έναν cluster από servers
- Δύο λειτουργίες: **get(key)** και **put(key, data)**
  - Data μικρού μεγέθους (< 1MB)
- Τι χρειαζόμαστε;
  - Διαχείριση μελών
  - Εισαγωγή/αναζήτηση/διαγραφή δεδομένων
  - Consistency με replication
  - Partition tolerance

# Σχεδιαστικές τεχνικές

---

- Gossiping για διαχείριση μελών και ανίχνευση σφαλμάτων
  - Eventually-consistent membership
- Consistent hashing για κατανομή κλειδιών και κόμβων
  - Παρόμοιο με το Chord
  - Αλλά δεν υπάρχει δρομολόγηση δακτυλίου, όλοι οι κόμβοι γνωρίζονται
- Object versioning για eventually-consistent δεδομένα
  - Ένα vector clock για κάθε δεδομένο
- Quorums για partition tolerance
  - “Sloppy” quorum
- Merkle tree για συγχρονισμό των replicas μετά από σφάλματα ή partitions



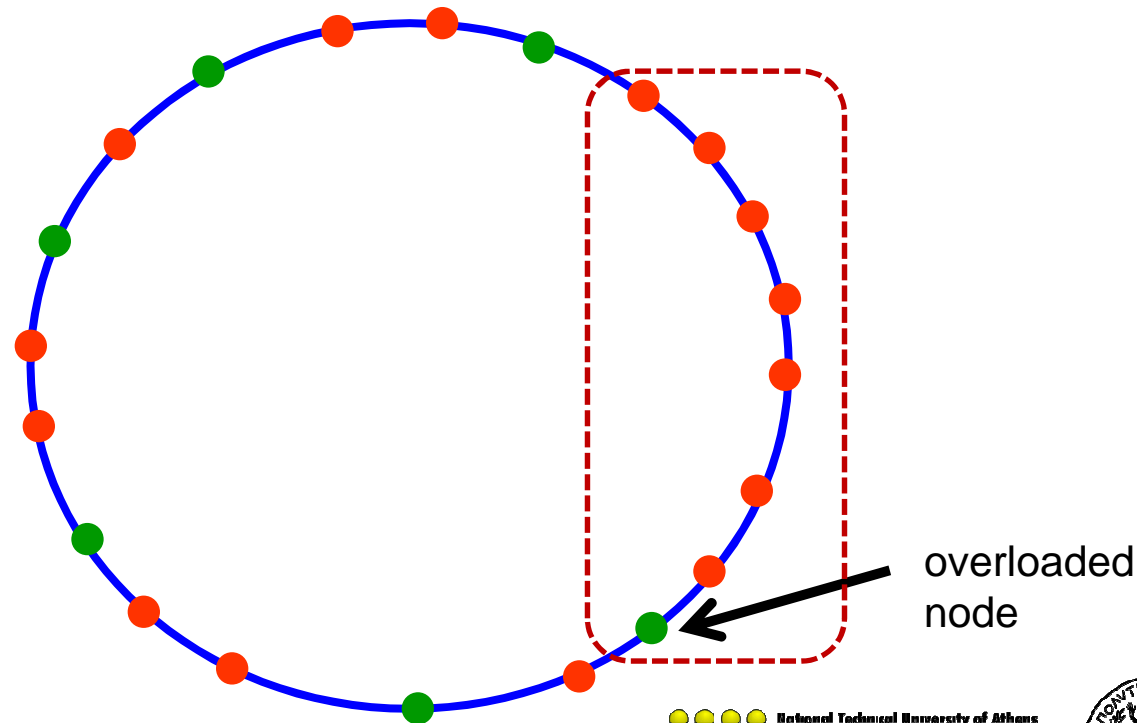
# Διαχείριση μελών

---

- Οι κόμβοι οργανώνονται σε δακτύλιο Chord με χρήση consistent hashing
  - Αλλά όλοι ξέρουν όλους
- Εισαγωγή/αποχώρηση κόμβων
  - Γίνεται χειροκίνητα
  - Κάποιος διαχειριστής χρησιμοποιεί κονσόλα για να προσθέσει ή να αφαιρέσει έναν κόμβο
  - Λόγος: η φύση των εφαρμογών που υποστηρίζει
    - Οι κόμβοι δεν αποχωρούν συχνά, μόνο λόγω σφάλματος και επανέρχονται γρήγορα
- Διάδοση της αλλαγής στα μέλη
  - Κάθε κόμβος διατηρεί δικό του group view και ιστορικό αλλαγών
  - Διάδοση μέσω gossiping (ανά δευτερόλεπτο, σε τυχαίους κόμβους)
- Eventually-consistent membership protocol

# Κατανομή κόμβων και κλειδιών

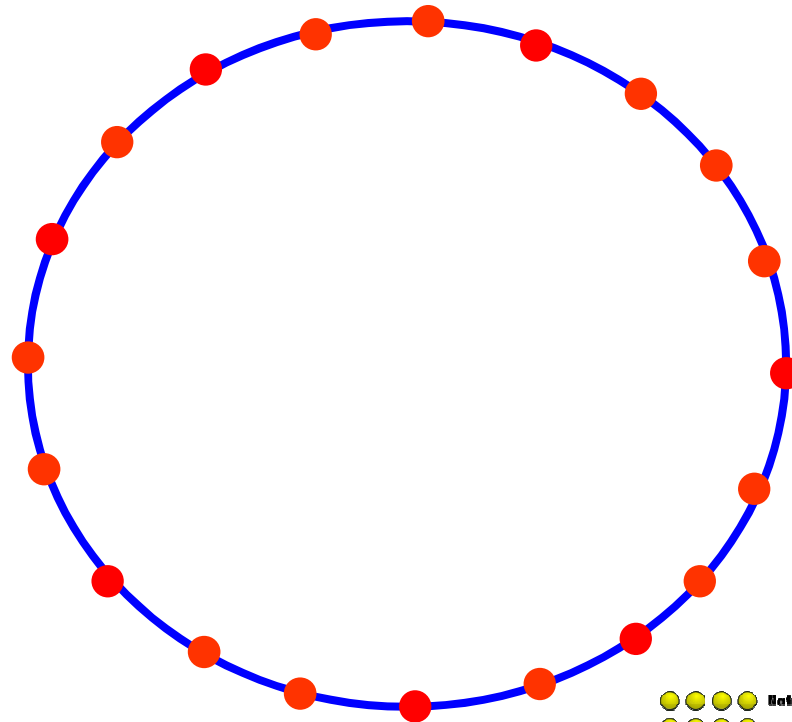
- Με παραδοσιακό consistent hashing έχουμε ανισορροπία φόρτου
  - Ένας κόμβος μπορεί να έχει περισσότερα κλειδιά από άλλους
  - Κάποια κλειδιά είναι πιο δημοφιλή από άλλα



# Κατανομή κόμβων και κλειδιών

---

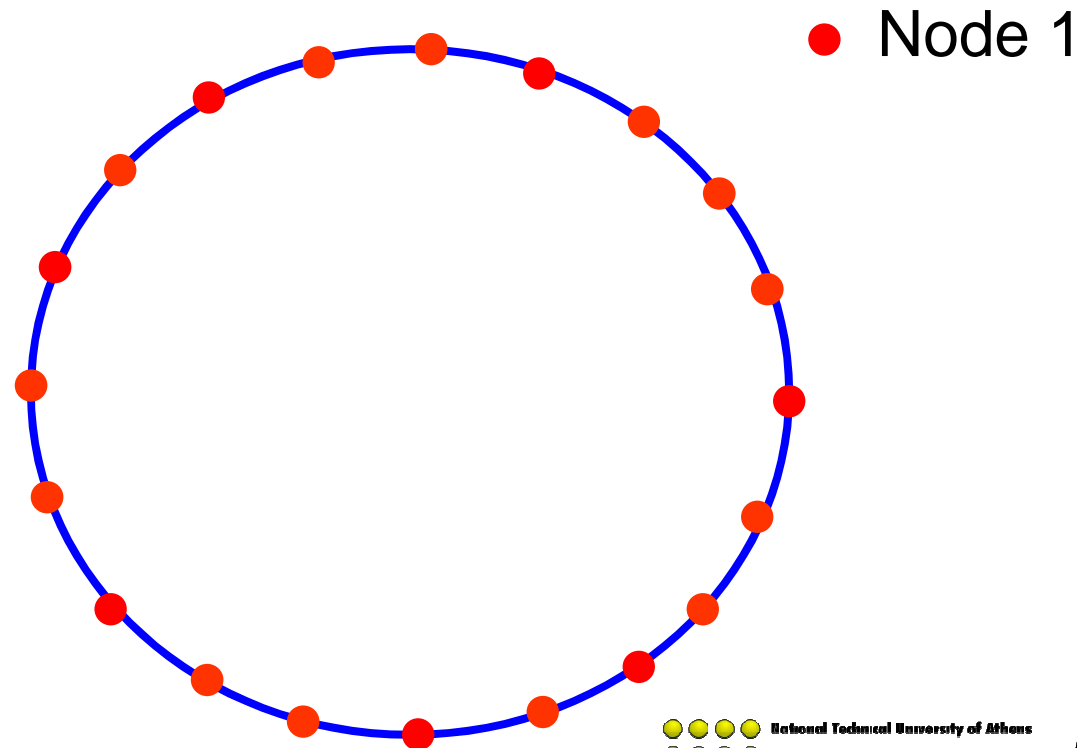
- Κάθε κόμβος ανατίθεται σε πολλά σημεία στον δακτύλιο
  - Κάθε σημείο είναι ένας εικονικός κόμβος (virtual node)
- Εκκίνηση με στατικό αριθμό virtual nodes ομοιόμορφα κατανεμημένων στον δακτύλιο



# Κατανομή κόμβων και κλειδιών

---

- Ο πρώτος φυσικός κόμβος που εισέρχεται αναλαμβάνει όλους τους virtual nodes

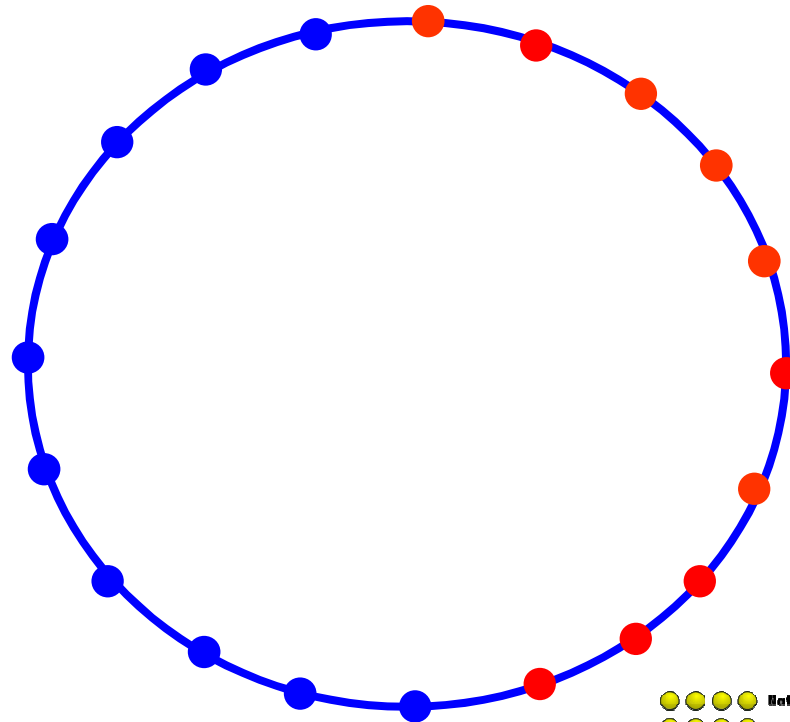


# Κατανομή κόμβων και κλειδιών

---

- Ο δεύτερος παίρνει το 1/2

● Node 1  
● Node 2

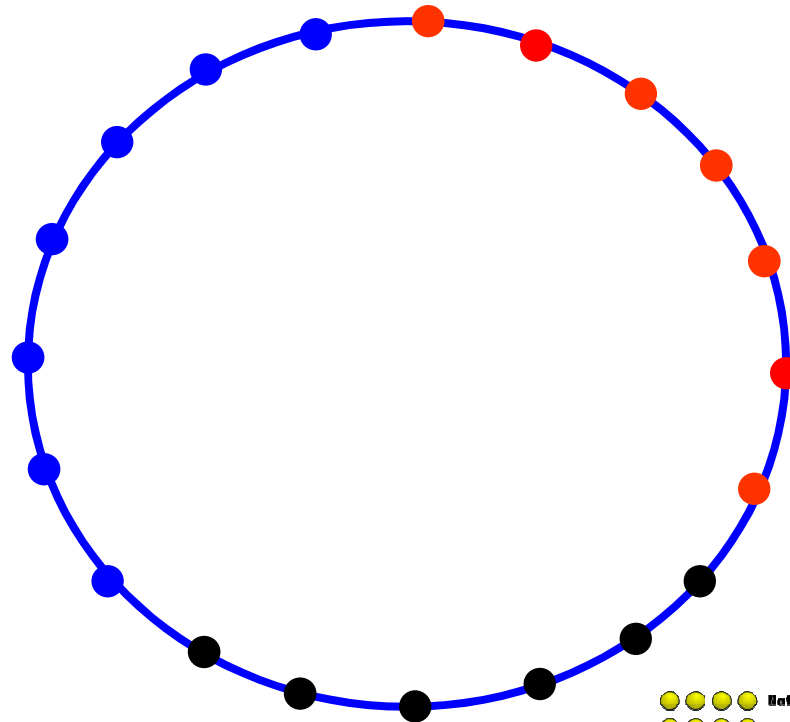


# Κατανομή κόμβων και κλειδιών

---

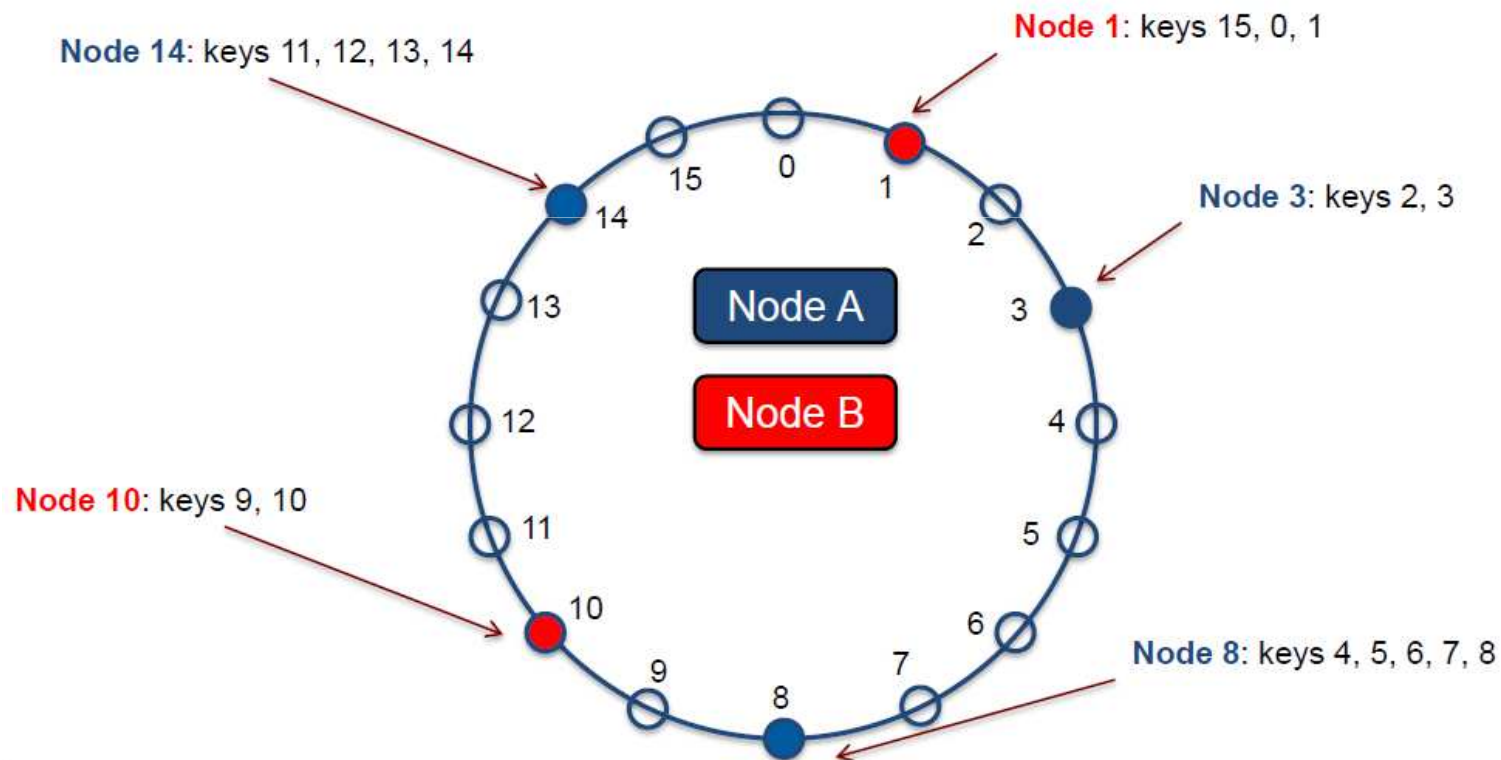
- Ο τρίτος περίπου το 1/3

- Node 1
- Node 2
- Node 3



# Παράδειγμα

- 2 physical nodes, 5 virtual nodes



# Πλεονεκτήματα

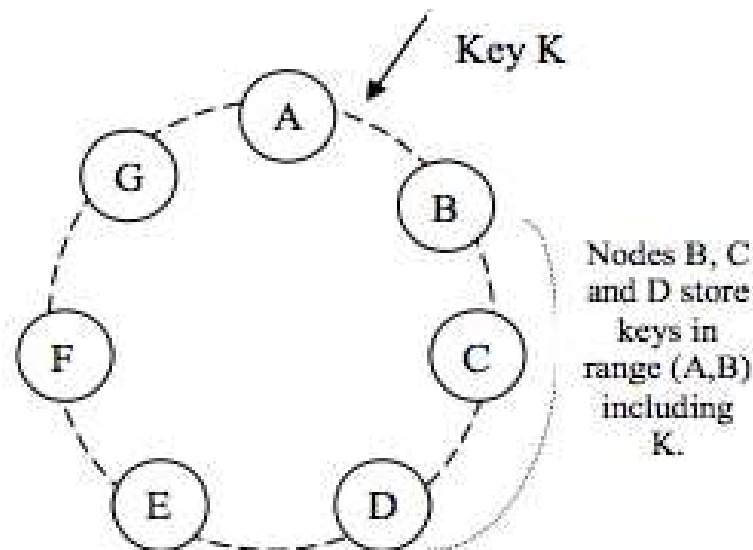
---

- Εξισορρόπηση φόρτου
  - Αν ένας κόμβος αποχωρήσει, ο φόρτος του κατανέμεται στους εναπομείναντες κόμβους
  - Αν ένας κόμβος προστεθεί, δέχεται παρόμοιο φορτίο από τους υπόλοιπους κόμβους
- Ο αριθμός από virtual nodes ανά κόμβο καθορίζεται από τις δυνατότητες του κόμβου



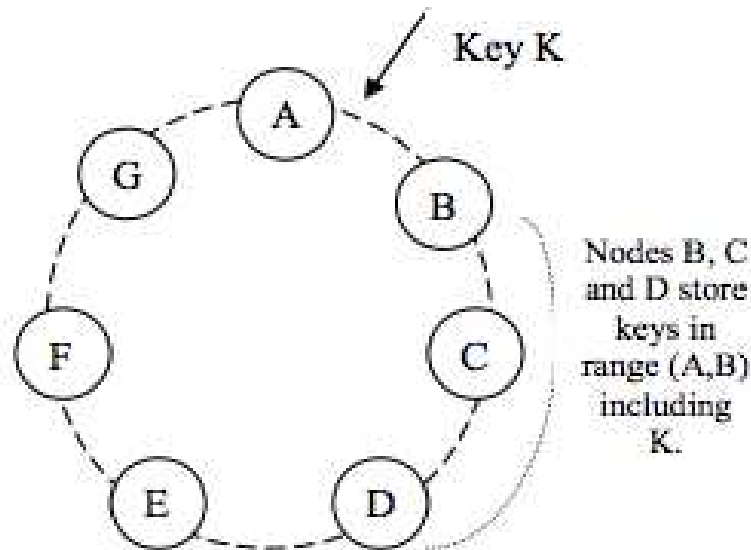
# Replication

- N: # of replicas
- Το πρώτο αποθηκεύεται με βάση το consistent hashing
- Τα υπόλοιπα N-1 αποθηκεύονται στους επόμενους N-1 (φυσικούς) διάδοχους κόμβους (preference list)



# Replication

- Lazy replication (eventual consistency)
  - Ένα αίτημα put() επιστρέφει αμέσως (high write throughput)
    - Δεν περιμένει να διαδοθεί το update σε όλα τα replicas
- Οδηγεί σε inconsistency, που λύνεται με object versioning



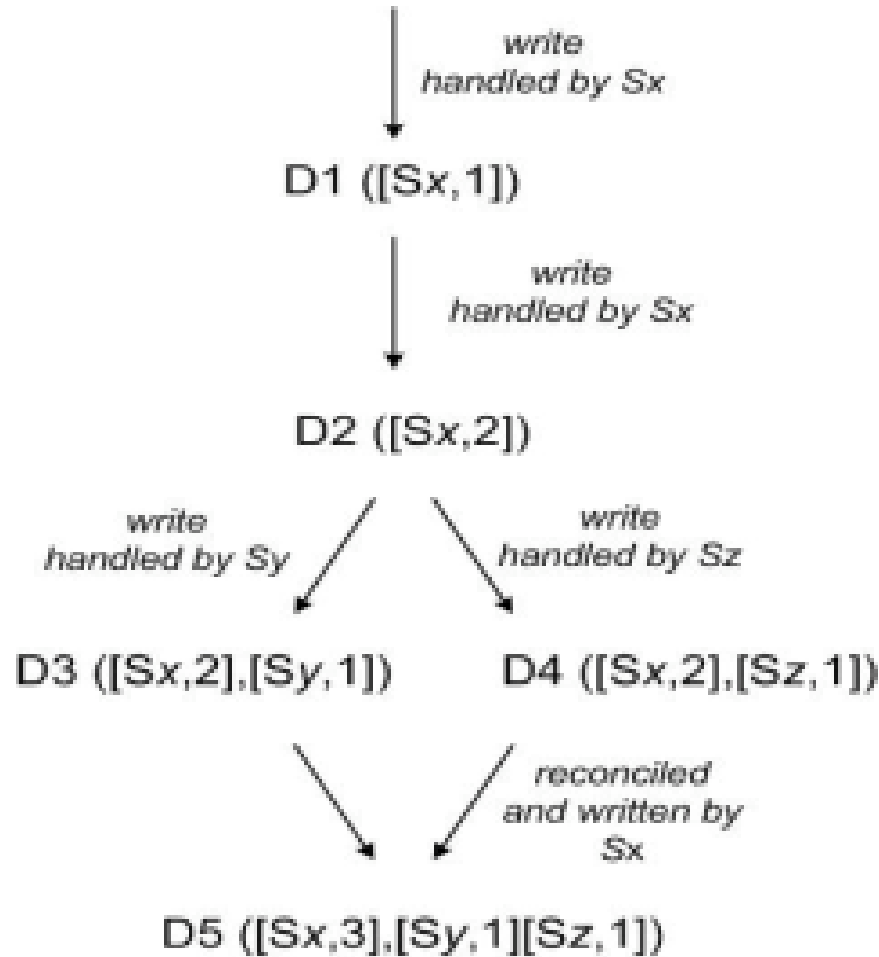
# Object Versioning

---

- Τα writes πρέπει πάντα να επιτυγχάνουν
  - π.χ., “Add to Cart”
- Χρησιμοποιούνται versions για επίλυση ασυνεπειών μεταξύ replicas
- Κάθε αντικείμενο έχει ένα vector clock
  - π.χ,  $D_1 ([S_x, 1], [S_y, 1])$ : Το  $D_1$  ενημερώθηκε μια φορά από τον  $S_x$  και μια από τον  $S_y$
  - Κάθε κόμβος διατηρεί όλα τα versions μέχρι τα δεδομένα να γίνουν συνεπή
- Ασυνέπεια έχουμε όταν εμφανίζονται ταυτόχρονες versions
- Αν υπάρχει ασυνέπεια γίνεται επίλυση αργότερα
  - Π.χ. μπορεί να εμφανιστούν πάλι προϊόντα που έχουν σβηστεί από το shopping cart

# Object Versioning

---



# Object Versioning

---

- Consistency revisited
  - Linearizability: οποιοδήποτε read διαβάζει το τελευταίο write
  - Eventual consistency: ένα read operations μπορεί να μη διαβάσει το τελευταίο write και κάποιες φορές χρειάζεται επίλυση ασυνεπειών
- Στο Dynamo επίλυση κατά τα reads
  - highly available, read/write συνεχίζουν ακόμα και σε περίπτωση network partition
  - Ο πελάτης θα πρέπει πάντα να μπορεί να προσθέτει ή να αφαιρεί προϊόντα από το shopping cart
- Απλή επίλυση από το σύστημα
  - last-write-wins policy
- Πολύπλοκη επίλυση από την εφαρμογή
  - Η εφαρμογή γνωρίζει καλύτερα τη σημασία των δεδομένων
  - Χρειάζεται όλα τα versions των δεδομένων
  - Π.χ. συγχώνευση των shopping cart versions

# Στατιστικά για object Versioning

---

- Over a 24-hour period
- 99.94% of requests saw exactly one version
- 0.00057% saw 2 versions
- 0.00047% saw 3 versions
- 0.00009% saw 4 versions
- Usually triggered by many concurrent requests issued by busy robots, not human clients

# Quorums

---

- Παράμετροι
  - N replicas
  - R readers
  - W writers
- Στατικό quorum:  $R + W > N$
- Τυπικές τιμές για Dynamo:  $(N, R, W) == (3, 2, 2)$

# Συγχρονισμός replicas

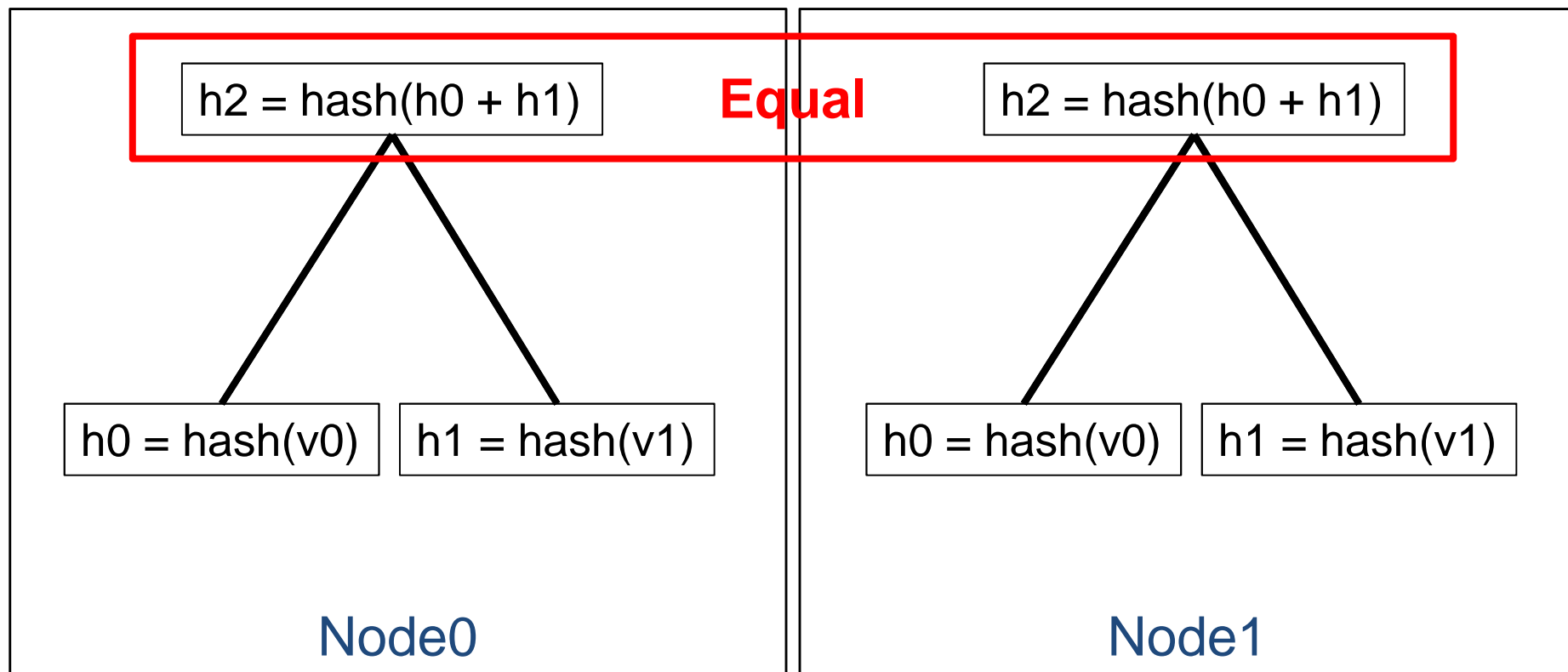
---

- Αν ένας κόμβος πεθάνει και ανανήψει, πρέπει να μπορεί γρήγορα να καταλάβει αν πρέπει να συγχρονίσει τα αντίγραφά του ή όχι
  - Η μεταφορά όλων των ζευγών (key, value) για σύγκριση δε συμφέρει
- Merkle trees
  - Τα φύλλα είναι hashes των τιμών του κάθε κλειδιού
  - Οι γονείς είναι hashes των παιδιών τους
  - Η σύγκριση γονέων στο ίδιο επίπεδο δείχνει διαφορά στα παιδιά
  - Μεταφορά μόνο των (key, value) που έχουν αλλάξει



# Συγχρονισμός replicas

- Σύγκριση 2 κόμβων που είναι συγχρονισμένοι
  - Two (key, value) pairs: (k0, v0) & (k1, v1)



# Συγχρονισμός replicas

- Σύγκριση 2 κόμβων που δεν είναι συγχρονισμένοι
  - One:  $(k_0, v_2)$  &  $(k_1, v_1)$
  - The other:  $(k_0, v_0)$  &  $(k_1, v_1)$

