



Εθνικό Μετσόβιο Πολυτεχνείο  
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών  
Εργαστήριο Υπολογιστικών Συστημάτων

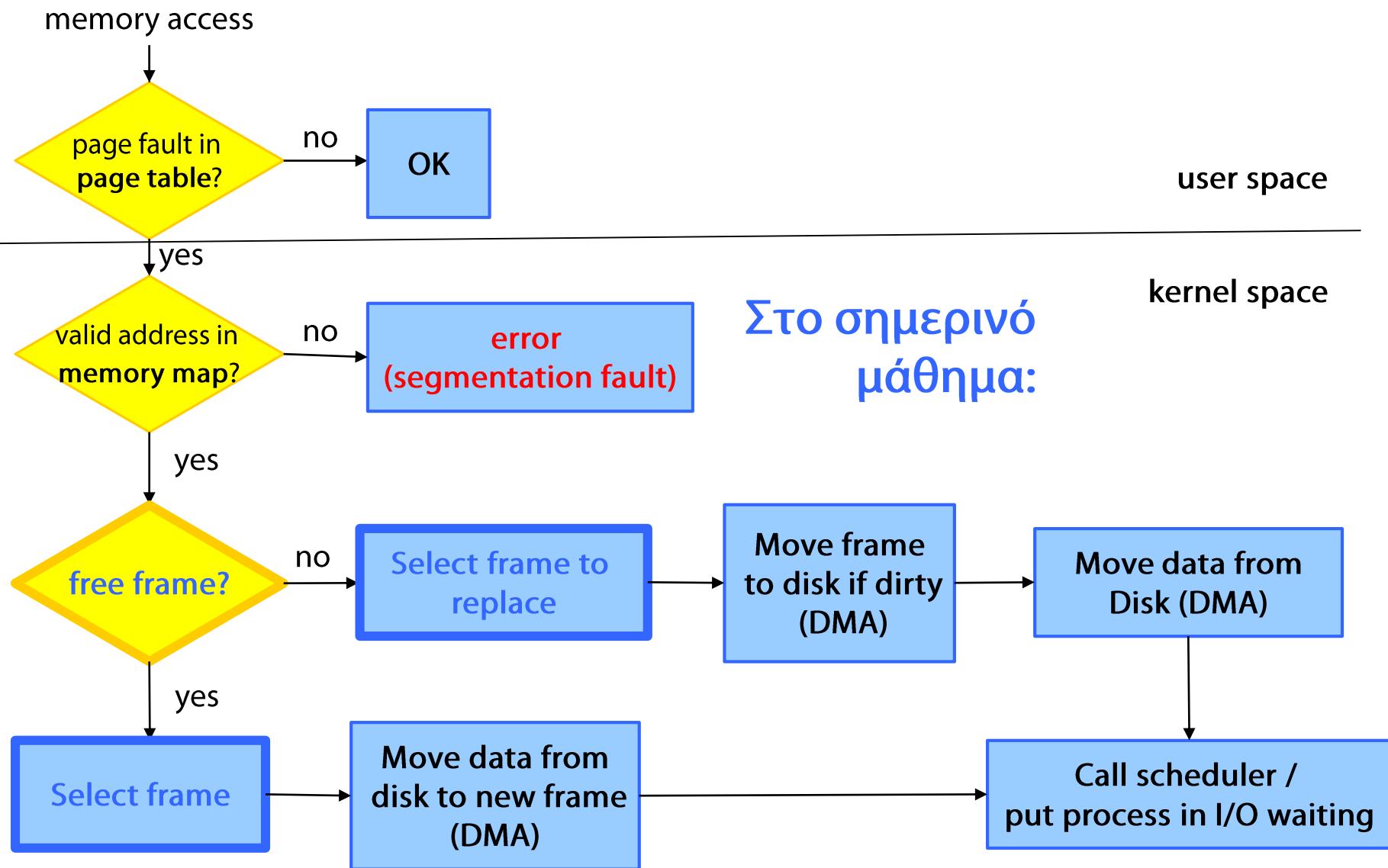
## Εικονική Μνήμη (2/2)

Λειτουργικά Συστήματα Υπολογιστών  
6ο Εξάμηνο, 2021-2022

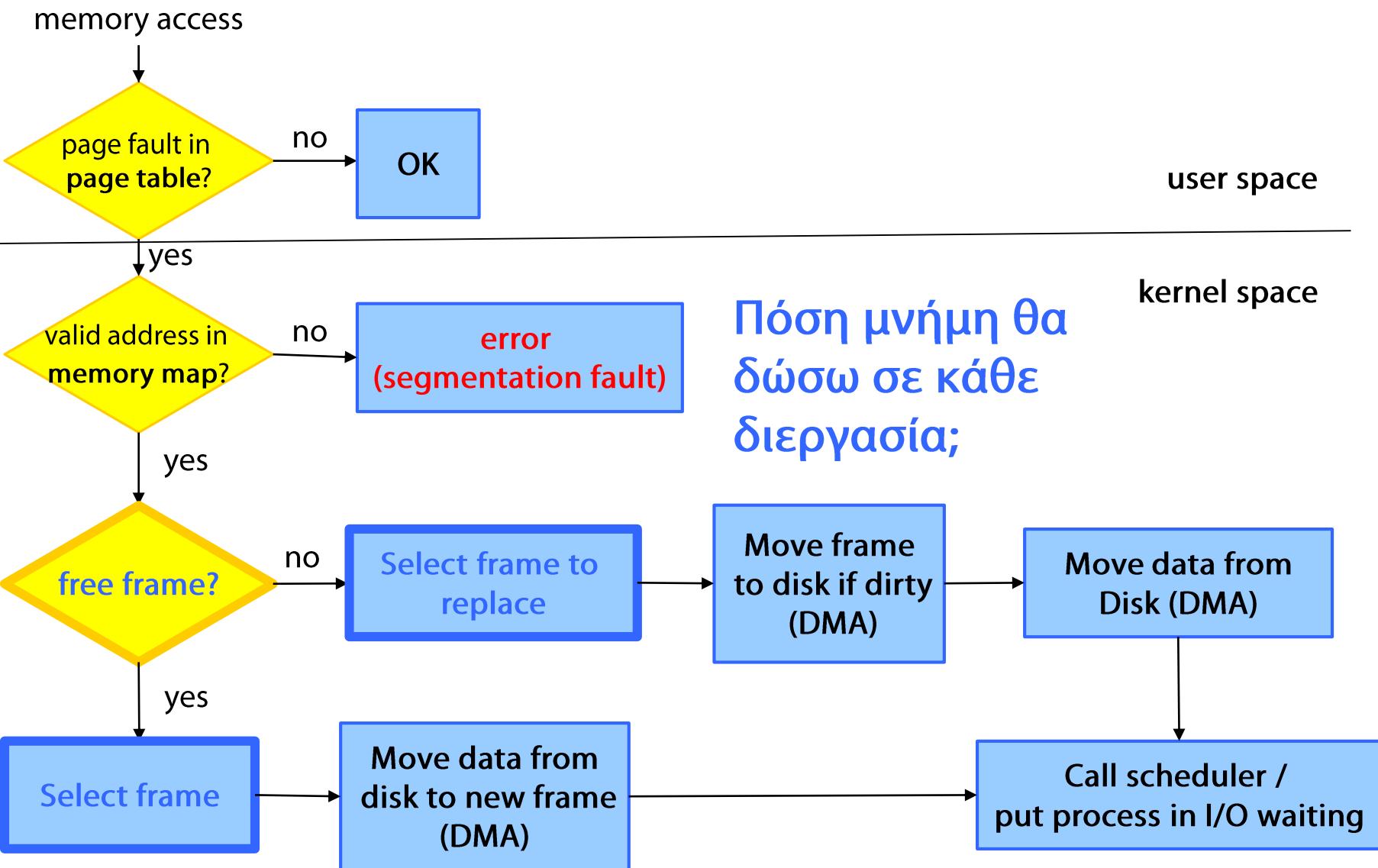
# Εικονική Μνήμη - Σύνοψη

- ◆ Εικονική Μνήμη με σελιδοποίηση
  - ➡ Σελιδοποίηση κατ' απαίτηση
  - ➡ Χειρισμός σφάλματος σελίδας
- ◆ Αντικατάσταση Σελίδων
  - ➡ Αλγόριθμοι αντικατάστασης FIFO, βέλτιστος, LRU
  - ➡ Γλοποίηση LRU, Προσέγγιση LRU
- ◆ Ανάθεση Πλαισίων
  - ➡ Thrashing, μοντέλο Συνόλου Εργασίας
- ◆ Buddy Allocators

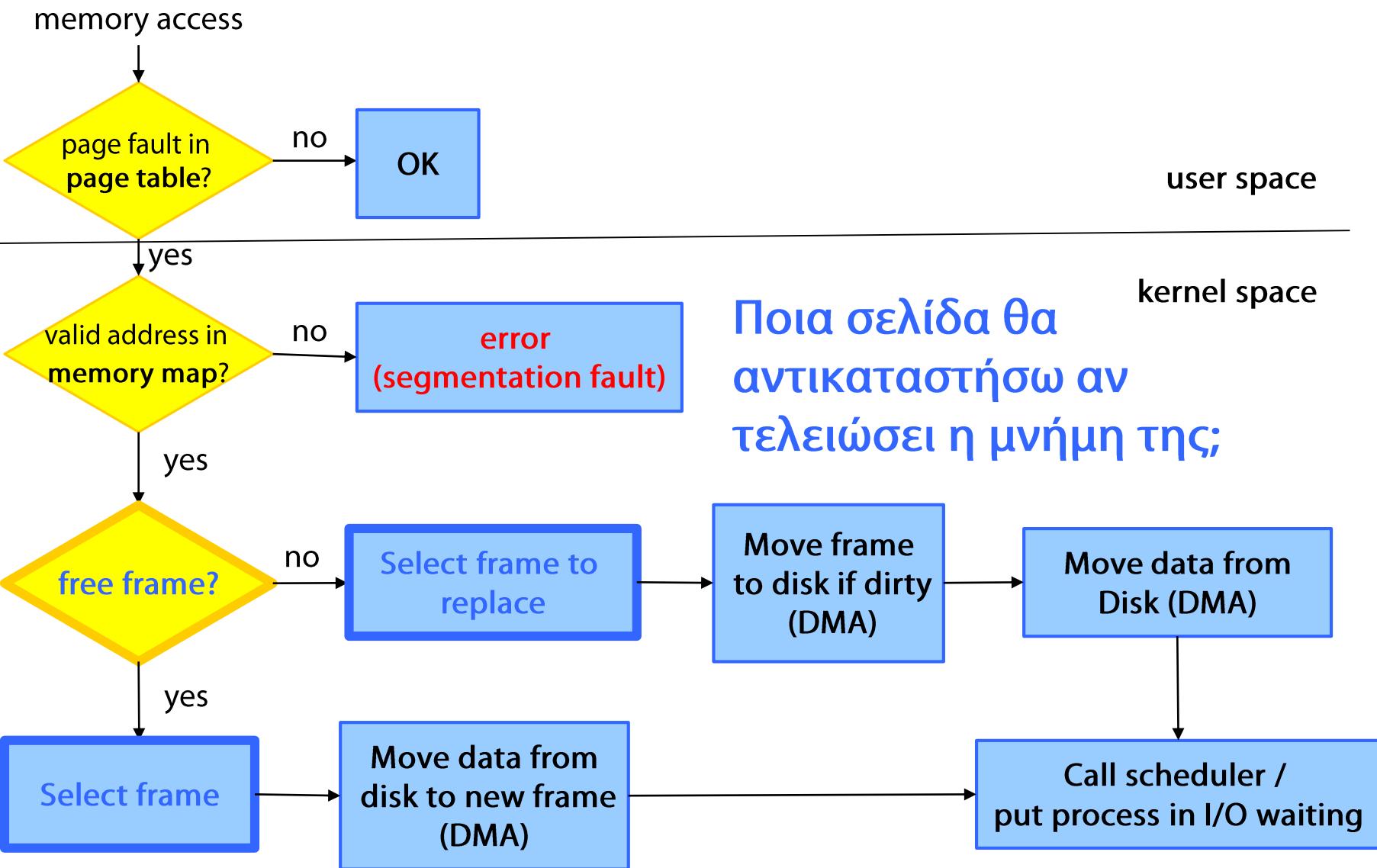
# Εικονική Μνήμη με Σελιδοποίηση: Η συνολική εικόνα



# Εικονική Μνήμη με Σελιδοποίηση: Η συνολική εικόνα



# Εικονική Μνήμη με Σελιδοποίηση: Η συνολική εικόνα



# Εικονική Μνήμη - Σύνοψη

- ◆ Αντικατάσταση Σελίδων
  - ➡ Αλγόριθμοι αντικατάστασης FIFO, βέλτιστος, LRU
  - ➡ Γλοποίηση LRU, Προσέγγιση LRU
- ◆ Ανάθεση Πλαισίων
  - ➡ Thrashing, μοντέλο Συνόλου Εργασίας
- ◆ Buddy Allocators

# Εικονική Μνήμη - Σύνοψη

- ◆ Αντικατάσταση Σελίδων
  - ➔ Αλγόριθμοι αντικατάστασης FIFO, βέλτιστος, LRU
  - ➔ Γλοποίηση LRU, Προσέγγιση LRU
- ◆ Ανάθεση Πλαισίων
  - ➔ Thrashing, μοντέλο Συνόλου Εργασίας
- ◆ Buddy Allocators

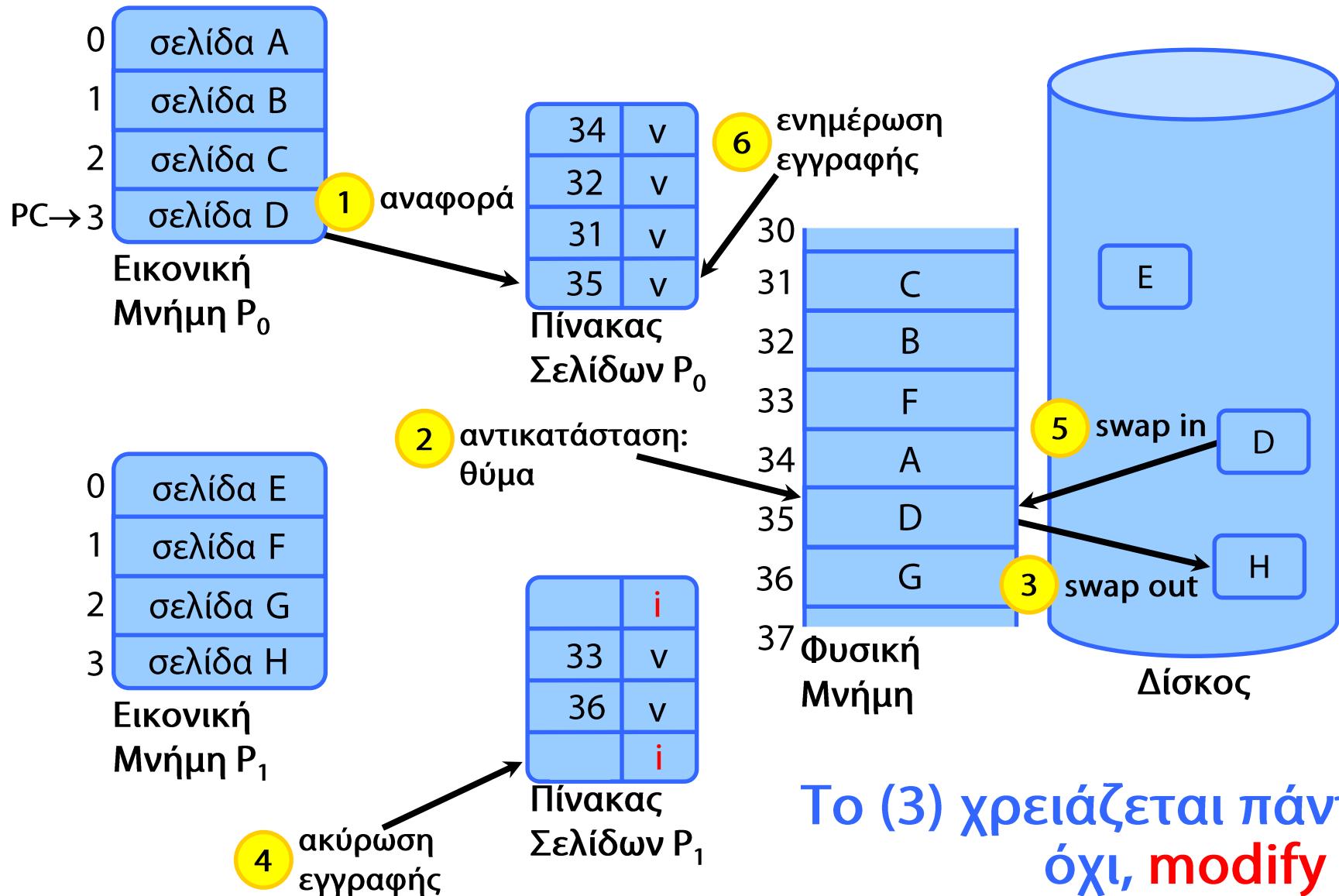
# Υπάρχει πάντα ελεύθερο πλαίσιο;

- ◆ Αν δεν υπάρχει ελεύθερο πλαίσιο μνήμης
  - ➡ Ανάγκη για *αντικατάσταση σελίδων*
  - ➡ Page Replacement
- ◆ Αντικατάσταση Σελίδων
  - ➡ Βρες μια σελίδα που δεν χρησιμοποιείται
    - και στείλε τη στο δίσκο
  - ➡ Με ποιον αλγόριθμο;
  - ➡ *Στόχος:* ελαχιστοποίηση σφαλμάτων σελίδας
- ◆ Η ίδια σελίδα μπορεί να ξαναέρθει αργότερα στη μνήμη

# Αντικατάσταση Σελίδων

- ◆ Επέκταση ρουτίνας χειρισμού σφαλμάτων σελίδας
  - ➡ Αν δεν υπάρχει ελεύθερο πλαίσιο, κάποια σελίδα πρέπει να πάει στο δίσκο
- ◆ Αντικατάσταση Σελίδων
  - ➡ Βρες την επιθυμητή σελίδα στο δίσκο
  - ➡ Βρες ελεύθερο πλαίσιο
    - Αν υπάρχει, εντάξει, χρησιμοποίησέ το
    - Αν όχι, βρες μια σελίδα *Θύμα*, με βάση τον αλγόριθμο αντικατάστασης και στείλε τη στο δίσκο (*πάντα;*)
  - ➡ Πλέον υπάρχει ελεύθερο πλαίσιο
    - Μεταφορά επιθυμητής σελίδας από δίσκο
    - Ανανέωση πίνακα σελίδων

# Δεν υπάρχει ελεύθερο πλαίσιο - Αντικατάσταση

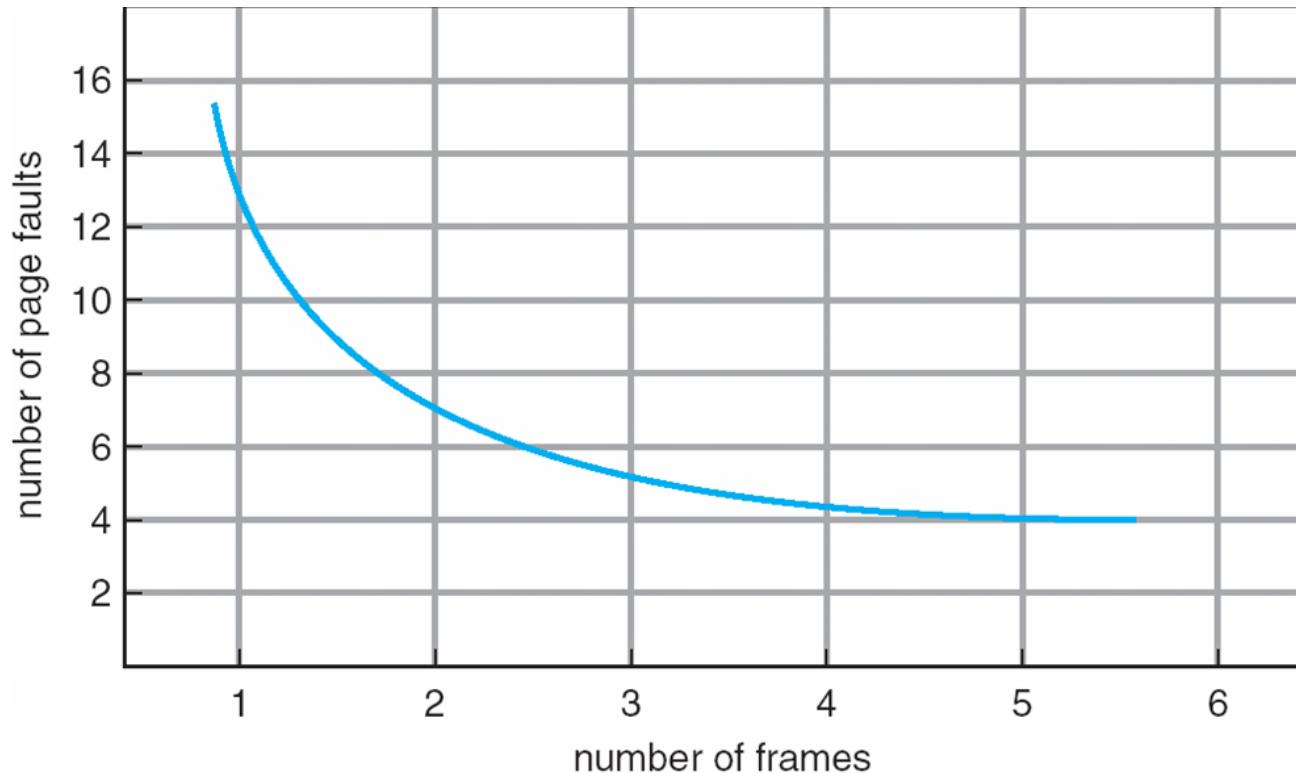


# Αλγόριθμοι αντικατάστασης σελίδων

- ◆ **Στόχος:** ελαχιστοποίηση ρυθμού σφαλμάτων σελίδας
- ◆ Αποτίμηση διαφορετικών αλγορίθμων
  - ➡ εκτέλεση για δεδομένη ακολουθία αναφορών
  - ➡ εξαρτάται από την εφαρμογή
  - ➡ πόσα σφάλματα σελίδας γίνονται;
- ◆ Οι προσβάσεις της διεργασίας σχηματίζουν μια ακολουθία αναφορών, π.χ.:
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

# Σφάλματα σε σχέση με διαθέσιμα πλαίσια

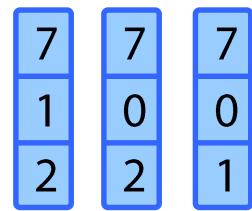
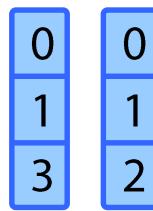
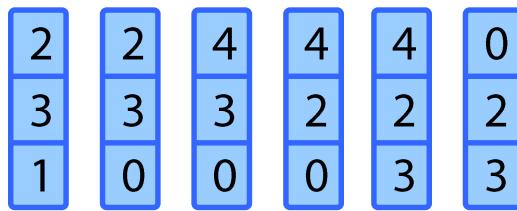
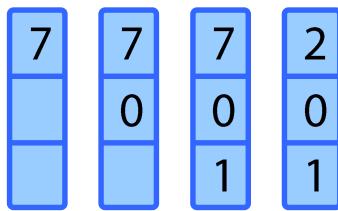


# Αλγόριθμος αντικατάστασης FIFO

Ακολουθία αναφορών

15 σφάλματα

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



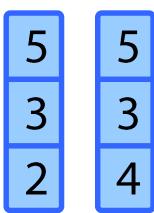
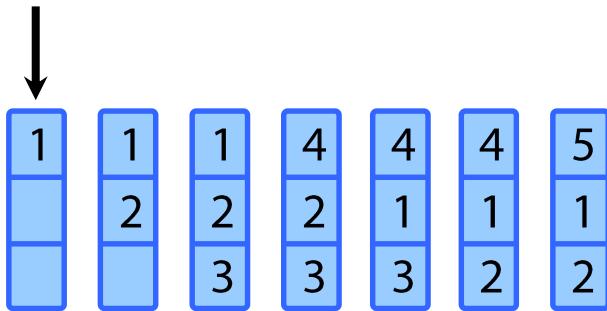
Πλαίσια μνήμης

- ◆ Πόσα σφάλματα σελίδας προκλήθηκαν;  
→ 15.

# Με περισσότερα πλαίσια, καλύτερα;

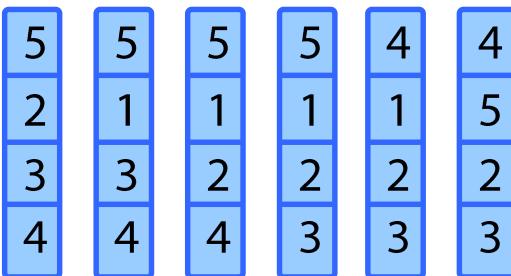
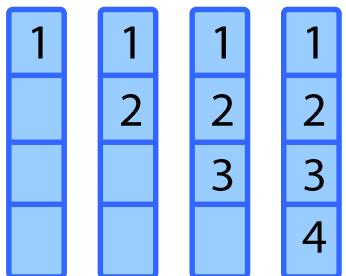
Ακολουθία αναφορών

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



3 πλαίσια  
μνήμης

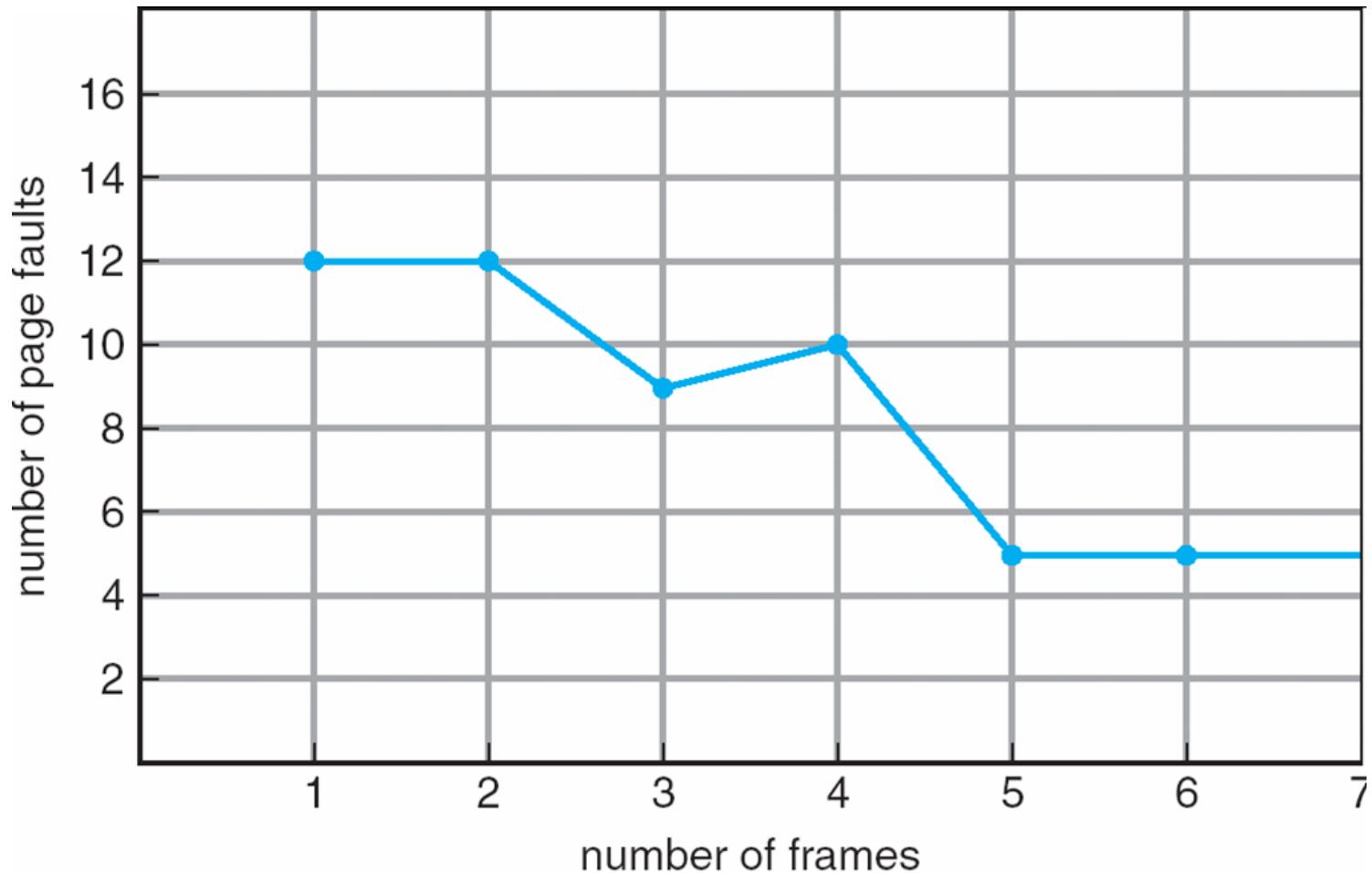
◆ 9 σφάλματα σελίδας.



4 πλαίσια  
μνήμης

◆ 10 σφάλματα σελίδας!

# Το παράδοξο του Belady για FIFO

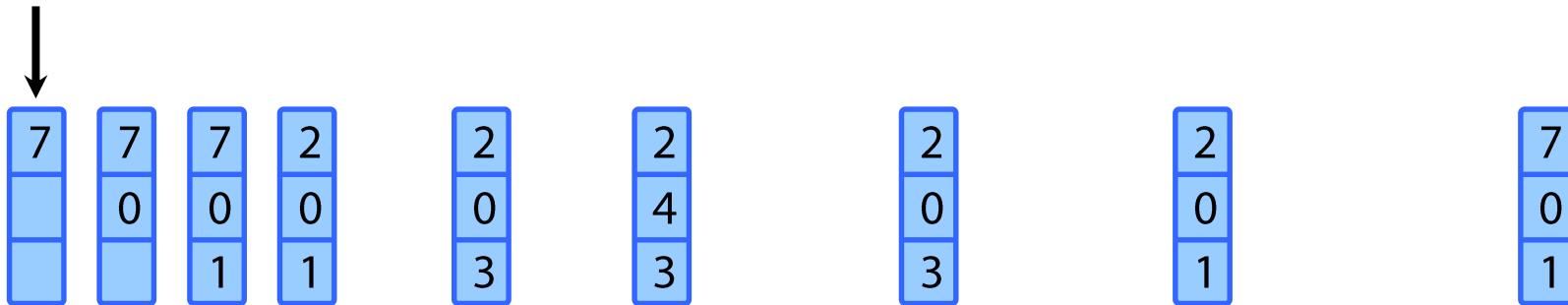


# Βέλτιστος Αλγόριθμος Αντικατάστασης

## Ακολουθία αναφορών

## 9 σφάλματα

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



## Πλαίσια μνήμης

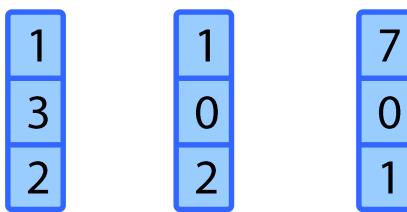
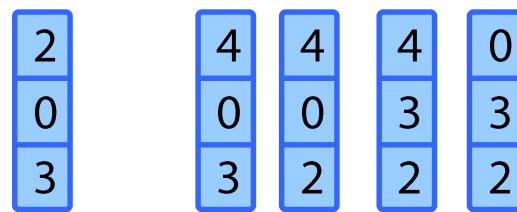
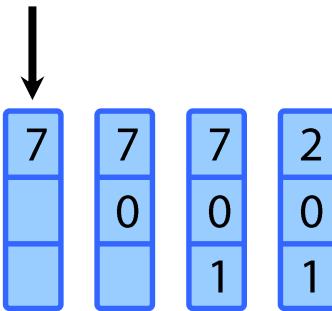
- ◆ Αντικατάστησε τη σελίδα στην οποία δεν θα γίνει αναφορά για το **μεγαλύτερο χρονικό διάστημα**.
    - ➡ ...στο μέλλον.
    - ➡ **Ιδανική** στρατηγική, χρήσιμη ως μέτρο σύγκρισης.
    - ➡ κατάσταση ανάλογη του SJF.

# Στρατηγική Least Recently Used (LRU)

Ακολουθία αναφορών

12 σφάλματα

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



- ◆ Αντικατάστησε τη σελίδα στην οποία δεν έγινε αναφορά για το *μεγαλύτερο χρονικό διάστημα στο παρελθόν*.
  - ➡ εκτίμηση της μελλοντικής συμπεριφοράς της διεργασίας, με βάση το παρελθόν της
- ◆ Πώς υλοποιείται;
  - ➡ Απαιτείται σημαντική υποστήριξη από το *υλικό*.

# Υλοποίηση LRU βασισμένη σε μετρητή

- ◆ Για κάθε σελίδα, καταγράφεται **πότε** έγινε η τελευταία αναφορά σε αυτή
  - ➡ επιπλέον πεδίο σε εγγραφή του πίνακα σελίδων
- ◆ Η CPU απαριθμεί τις αναφορές στη μνήμη
- ◆ Για κάθε προσπέλαση της σελίδας, ανανεώνεται το πεδίο-χρονοσφραγίδα
- ◆ Αντικατάσταση σελίδας
  - ➡ διώξε τη σελίδα με την ελάχιστη τιμή του μετρητή
- ◆ Προβλήματα;
  - ➡ Κόστος αναζήτησης, κόστος ανανέωσης πεδίων
  - ➡ Πιθανή υπερχείλιση μετρητή

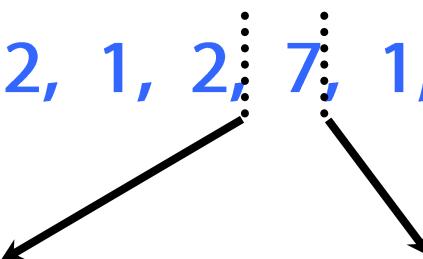
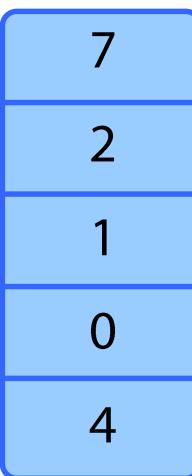
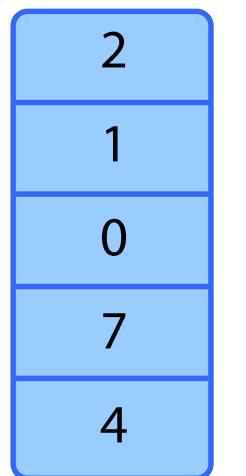
# Υλοποίηση LRU βασισμένη σε στοίβα (1)

- ◆ Διπλά συνδεδεμένη λίστα με αριθμούς σελίδων
- ◆ Αναφορά σε σελίδα
  - ➔ μετακίνησέ τη στην κορυφή της στοίβας
  - ➔ **κόστος:** αλλαγή 6 δεικτών
- ◆ Αντικατάσταση σελίδας
  - ➔ χωρίς αναζήτηση, φεύγει η τελευταία στη λίστα

# Υλοποίηση LRU βασισμένη σε στοίβα (2)

Ακολουθία αναφορών

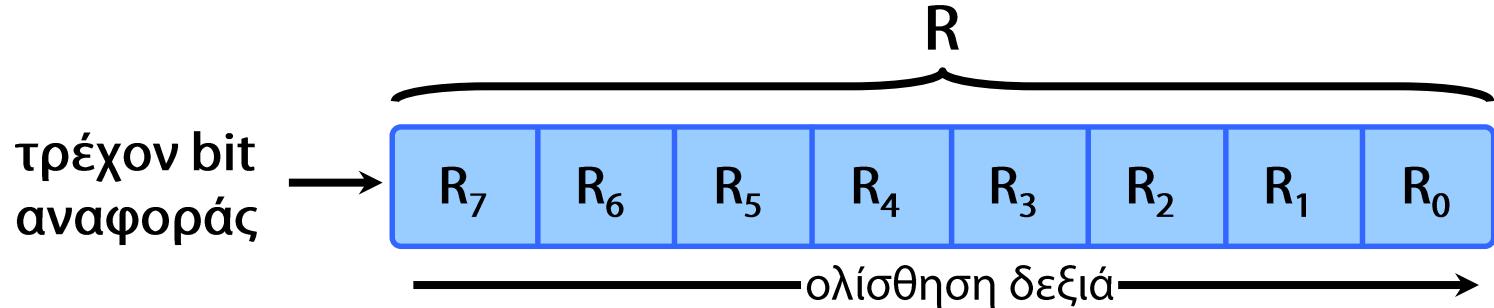
4, 7, 0, 7, 1, 0, 1, 2, 1, 2, 7, 1, 2



# Αλγόριθμοι προσέγγισης LRU

- ◆ Bit αναφοράς
  - ➔ Για κάθε σελίδα στον πίνακα σελίδων, αρχικά 0
  - ➔ Οποτεδήποτε γίνεται *αναφορά* σε αυτή, τίθεται 1
- ◆ Ξέρουμε σε ποιες σελίδες έγιναν αναφορές
  - ➔ δεν τηρείται όμως η σειρά των προσβάσεων
- ◆ Όταν πρέπει να φύγει μια σελίδα
  - ➔ διώξε αυτή που έχει bit αναφοράς μηδενικό

# Επιπλέον bits αναφοράς



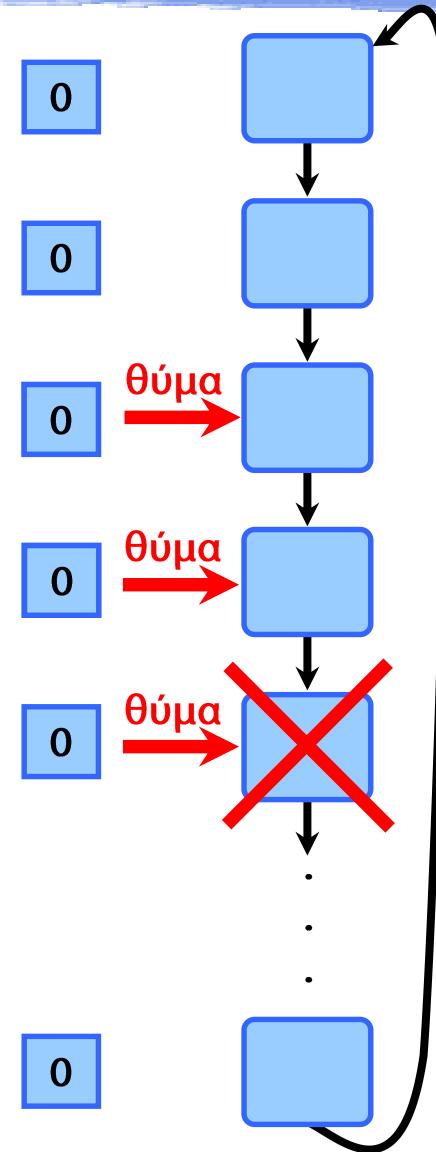
- ◆ Επιπλέον bits αναφοράς
- ◆ Για κάθε σελίδα, ένας αριθμός από bits, π.χ. 8
- ◆ Σε τακτά χρονικά διαστήματα (π.χ. κάθε 100ms), ολίσθηση του bit αναφοράς κάθε σελίδας στο MSB του καταχωρητή  $R$
- ◆ Αντικατάσταση σελίδας
  - Θεωρώντας τον  $R$  θετικό ακέραιο, η σελίδα με τη μικρότερη τιμή φεύγει

# Αλγόριθμος Δεύτερης Ευκαιρίας (1)

- ◆ Η βάση του είναι ένας αλγόριθμος FIFO
- ◆ αν μια σελίδα είναι να φύγει
  - ➡ έχει μια *δεύτερη ευκαιρία* αν το bit είναι αναμμένο
  - ➡ το bit μηδενίζεται
- ◆ και ελέγχεται η επόμενη
- ◆ Υπάρχει περίπτωση να φύγει παρόλο που το bit είναι αναμμένο;
  - ➡ Μόνο όταν σε όλες έχει δοθεί δεύτερη ευκαιρία
- ◆ Οργάνωση των σελίδων σε ρολόι

# Αλγόριθμος Δεύτερης Ευκαιρίας

- ◆ Οργάνωση σε ρολόι
  - ➡ κυκλική λίστα
- ◆ 'Όσο το bit αναφοράς είναι '1'
  - ➡ το μηδενίζει και συνεχίζει
- ◆ Μέχρι να βρει μηδενικό bit αναφοράς
- ◆ Αν όλα είναι αναμμένα;
  - ➡ εκφυλίζεται σε FIFO



# Εικονική Μνήμη - Σύνοψη

- ◆ Αντικατάσταση Σελίδων
  - ➔ Αλγόριθμοι αντικατάστασης FIFO, βέλτιστος, LRU
  - ➔ Γλοποίηση LRU, Προσέγγιση LRU
- ◆ Ανάθεση Πλαισίων
  - ➔ Thrashing, μοντέλο Συνόλου Εργασίας
- ◆ Buddy Allocators

# Ανάθεση πλαισίων

- ◆ Πώς ανατίθενται ελεύθερα πλαίσια σε διεργασίες;
  - ➡ Ή αλλιώς, πότε αποφαινόμαστε ότι δεν υπάρχει ελεύθερο πλαίσιο για μία διεργασία;
- ◆ Διαφορετικά σχήματα ανάθεσης
  - ➡ Ισόποση ανάθεση
  - ➡ Αναλογική ανάθεση
  - ➡ Ανάθεση με προτεραιότητες

# Ισόποση - αναλογική ανάθεση

- ◆ **Ισόποση ανάθεση:** ίδιος αριθμός πλαισίων για κάθε διεργασία, ανάλογα με τον αριθμό τους  
→ 100 πλαίσια, 5 διεργασίες → 20 πλαίσια / διεργασία
- ◆ **Αναλογική ανάθεση:** κατανομή αναλογικά με το μέγεθος της διεργασίας
  - αν  $s_i$  το μέγεθος του χώρου εικονικών διευθύνσεων που χρησιμοποιεί η  $P_i$
  - και υπάρχουν συνολικά  $m$  πλαίσια, η  $P_i$  παίρνει  $a_i$ :

$$a_i = \frac{s_i}{\sum s_i} m$$

# Ανάθεση με προτεραιότητες

- ◆ Αναλογικό σχήμα με βάση την προτεραιότητα των διεργασιών
- ◆ Όταν μια διεργασία προκαλέσει σφάλμα σελίδας
  - ➡ αντικατάσταση δικού της πλαισίου
  - ➡ αντικατάσταση πλαισίου κάποιας με χαμηλότερη προτεραιότητα
- ◆ **Μειονέκτημα:** η διεργασία δεν μπορεί να ελέγξει το ρυθμό σφαλμάτων της

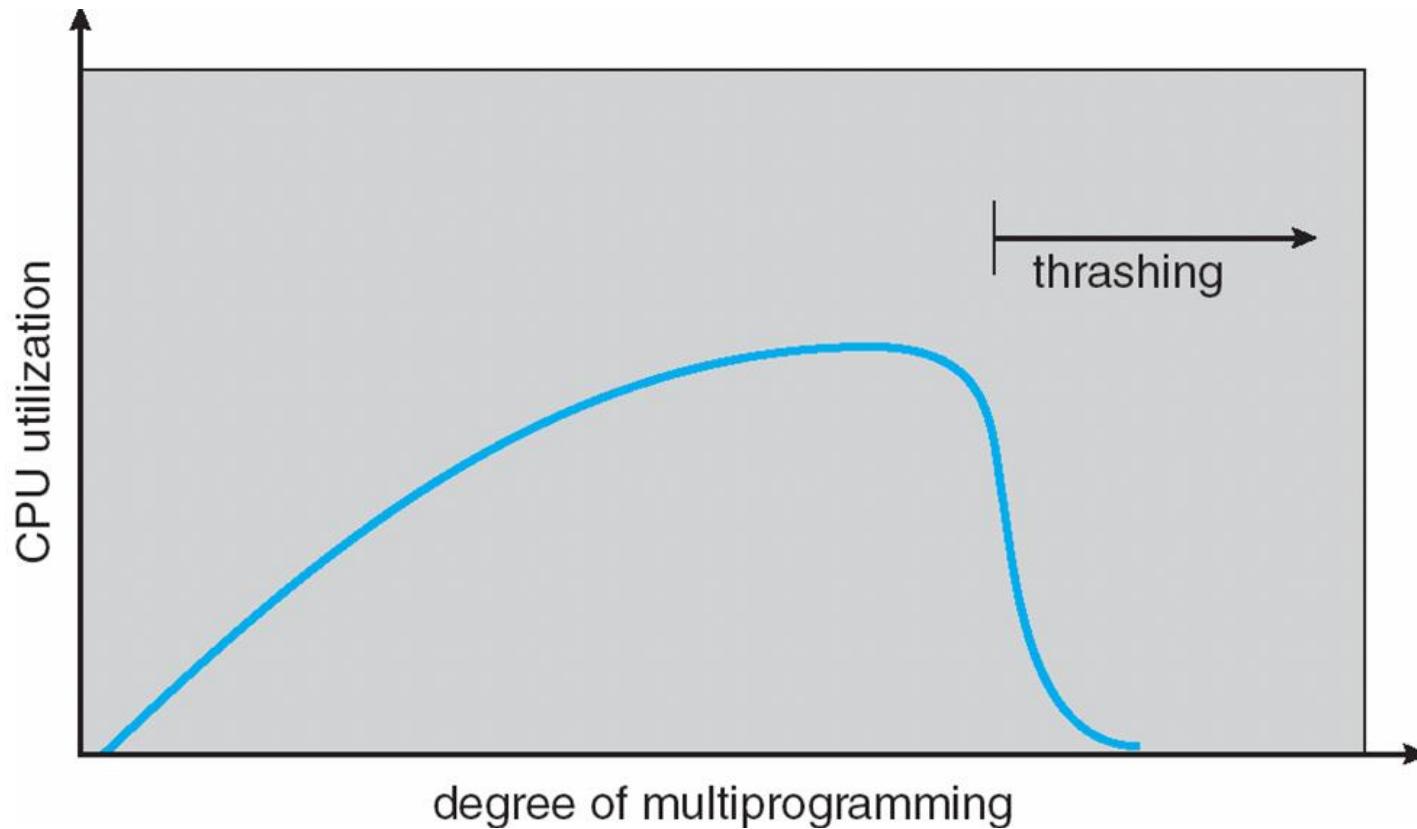
# Καθολική και τοπική αντικατάσταση

- ◆ **Καθολική αντικατάσταση:** σφάλμα σελίδας σε διεργασία προκαλεί αντικατάσταση πλαισίου οποιασδήποτε διεργασίας
- ◆ **τοπική αντικατάσταση:** σφάλμα σελίδας προκαλεί αντικατάσταση πλαισίου της συγκεκριμένης διεργασίας
  - ➡ Ο αριθμός των πλαισίων της παραμένει σταθερός
- ◆ Από τι εξαρτάται ο ρυθμός σφαλμάτων σελίδας σε κάθε περίπτωση;

# Thrashing (1)

- ◆ Υπερβολικά υψηλός ρυθμός αντικατάστασης
  - ➡ λόγω υπερβολικά χαμηλού αριθμού πλαισίων
- ◆ Αντί να γίνεται χρήσιμος υπολογισμός
  - ➡ οι διεργασίες προκαλούν συνεχώς Ε/Ε για σελίδες
  - ➡ η CPU παραμένει άεργη
  - ➡ καμία διεργασία δεν κάνει πρόοδο
- ◆ Χαμηλή χρησιμοποίηση CPU
  - ➡ το ΛΣ μπορεί εσφαλμένα να θεωρήσει ότι πρέπει να αυξήσει το βαθμό πολυπρογραμματισμού
  - ➡ ενώ πρέπει να τον μειώσει

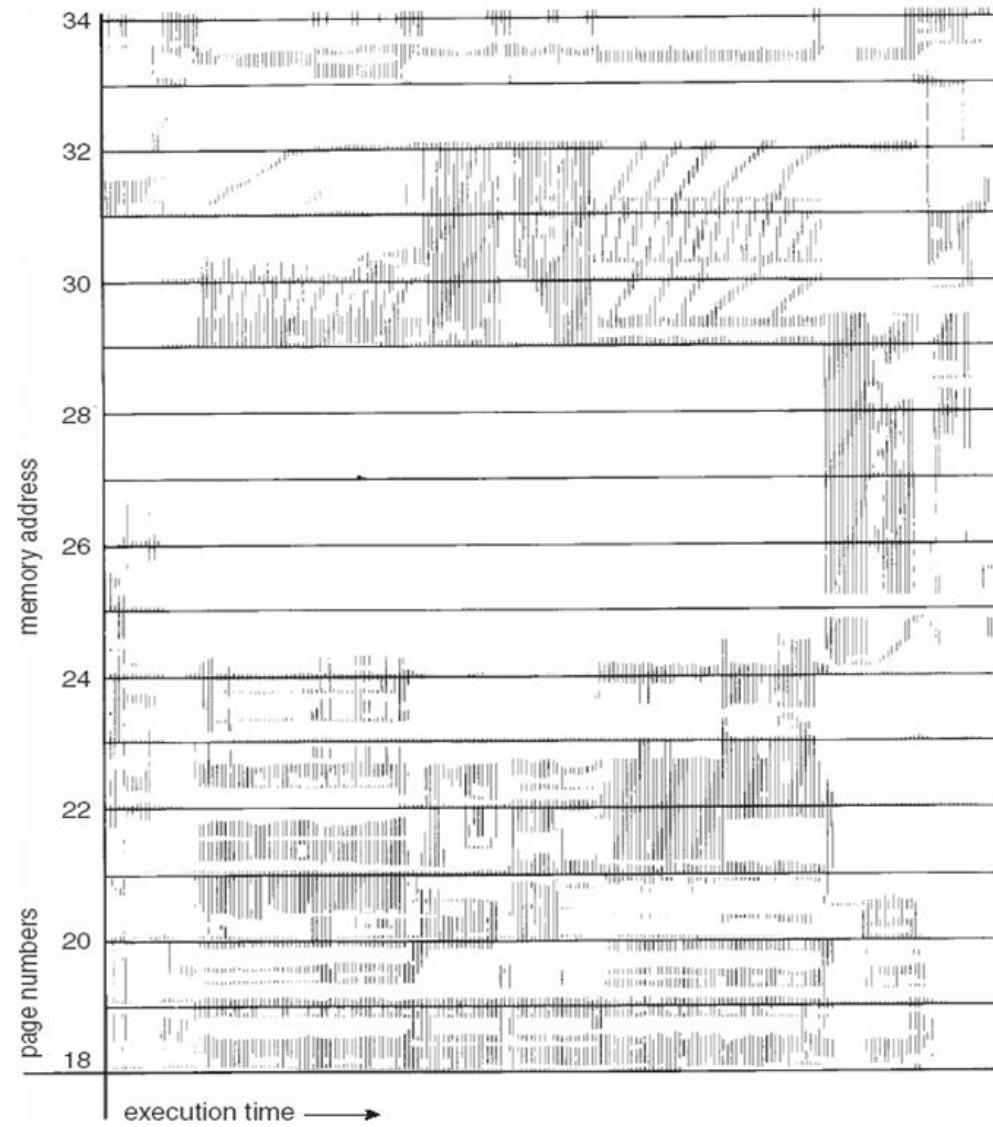
# Thrashing (2)



# Το μοντέλο τοπικότητας (1)

- ◆ Οι διεργασίες μετακινούνται από **τοπικότητα** σε τοπικότητα
- ◆ Τοπικότητα – Locality
  - ➡ σύνολο σελίδων που χρησιμοποιούνται μαζί
  - ➡ π.χ. το πρόγραμμα εκτελεί μια κλήση συνάρτησης
  - ➡ οι τοπικότητες μπορεί να επικαλύπτονται
- ◆ 'Όταν Σμέγεθος τοπικότητας > διαθέσιμη μνήμη
  - ➡ Thrashing

# Το μοντέλο τοπικότητας (2)



# Σύνολο Εργασίας (1)

- ◆ Δ: *παράθυρο* του συνόλου εργασίας
- ◆ Σύνολο εργασίας – Working Set
  - ➡ Το σύνολο των σελίδων όπου γίνεται αναφορά μέσα στις Δ τελευταίες χρονικές μονάδες
- ◆ Επιλογή του Δ
  - ➡ πολύ μικρό: δεν συμπεριλαμβάνει όλη την τοπικότητα
  - ➡ πολύ μεγάλο: συμπεριλαμβάνει πολλές τοπικότητες
- ◆  $WSS_i$ : μέγεθος του συνόλου εργασίας
- ◆  $D = \sum WSS_i$ : συνολική απαίτηση σε πλαίσια
- ◆  $A_v D > m$ 
  - ➡ Thrashing! Το ΛΣ πρέπει να *αναστείλει* μια διεργασία

# Σύνολο εργασίας (2)

Ακολουθία αναφορών

2, 6, 1, 5, 7, 7, 7, 7, 5, 1, 6, 2, 3, 4, 1, 2, 3, 4, 4, 4, 3, 4, 3, 4, 4, 4, 1, 3, 2, 3, 4, 4, 4, 3, 4, 4, 4, ...



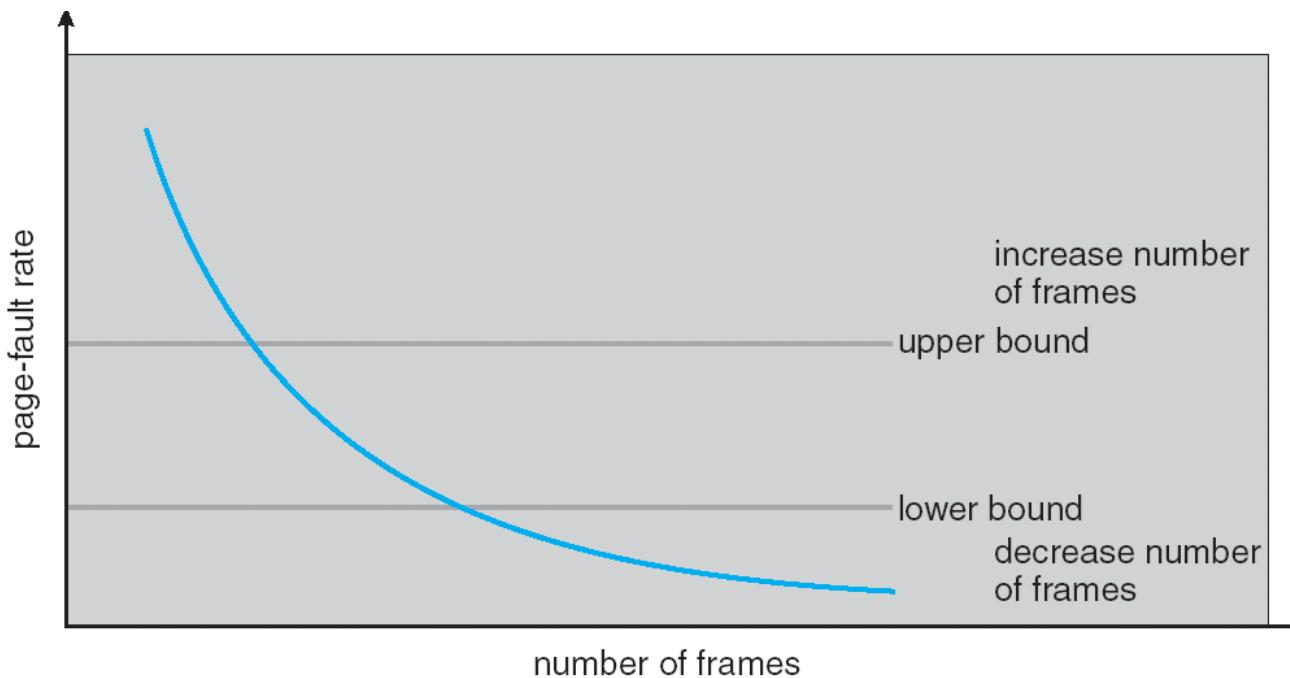
$$WS(t_1) = \{1, 2, 5, 6, 7\}$$

$$WS(t_2) = \{3, 4\}$$

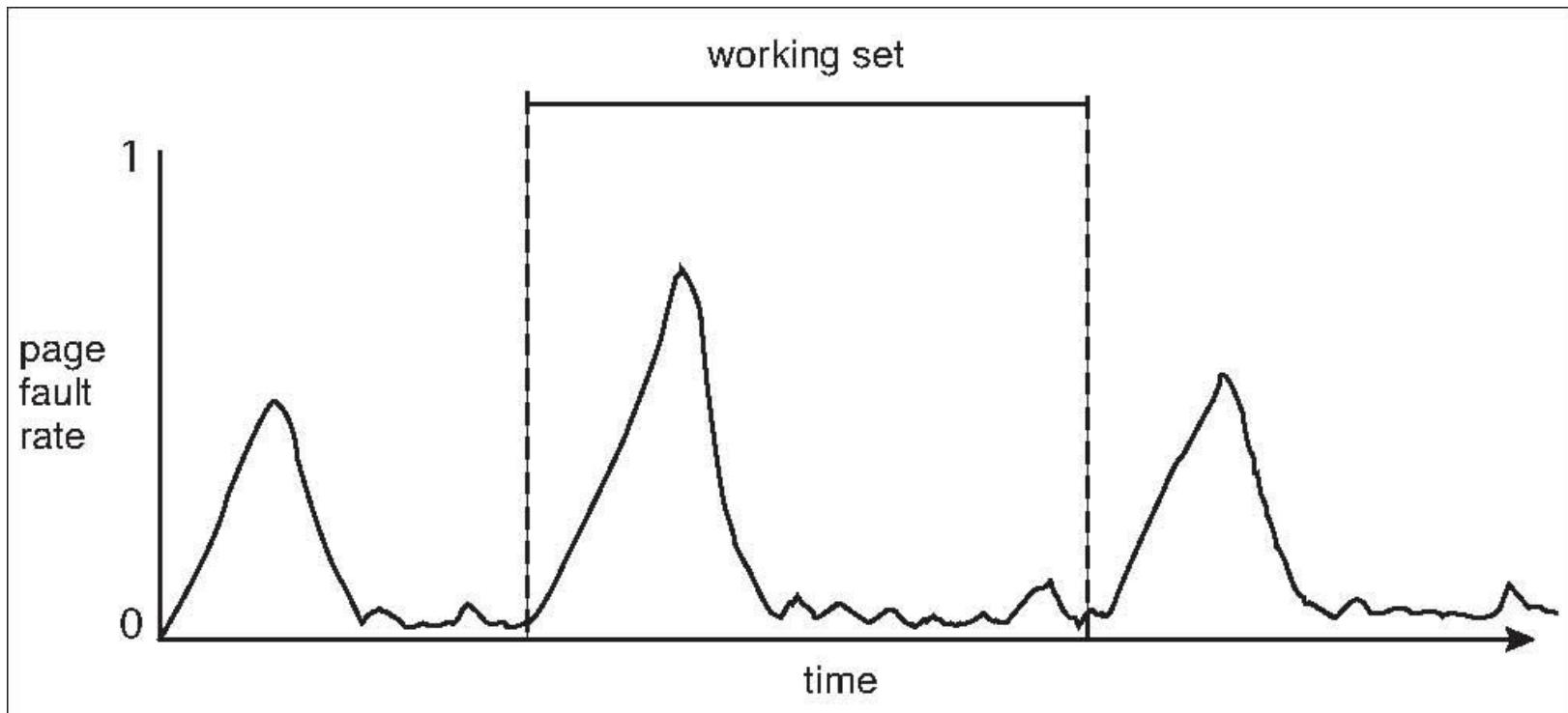
- ◆ Υλοποίηση παρακολούθησης του συνόλου εργασίας
  - ➡ με διακοπές χρονιστή και bit αναφοράς για κάθε σελίδα
  - ➡ αν το bit αναφοράς είναι αναμμένο, η σελίδα ανήκει στο σύνολο

# Ρυθμός σφαλμάτων σελίδας

- ◆ Το ΛΣ παρακολουθεί το ρυθμό σφαλμάτων για να περιορίσει το thrashing
  - ⇒ Χαμηλός ρυθμός: η διεργασία *χάνει* ένα πλαίσιο
  - ⇒ Υψηλός ρυθμός: η διεργασία *κερδίζει* ένα πλαίσιο



# Σχέση συνόλου εργασίας - ρυθμού σφαλμάτων



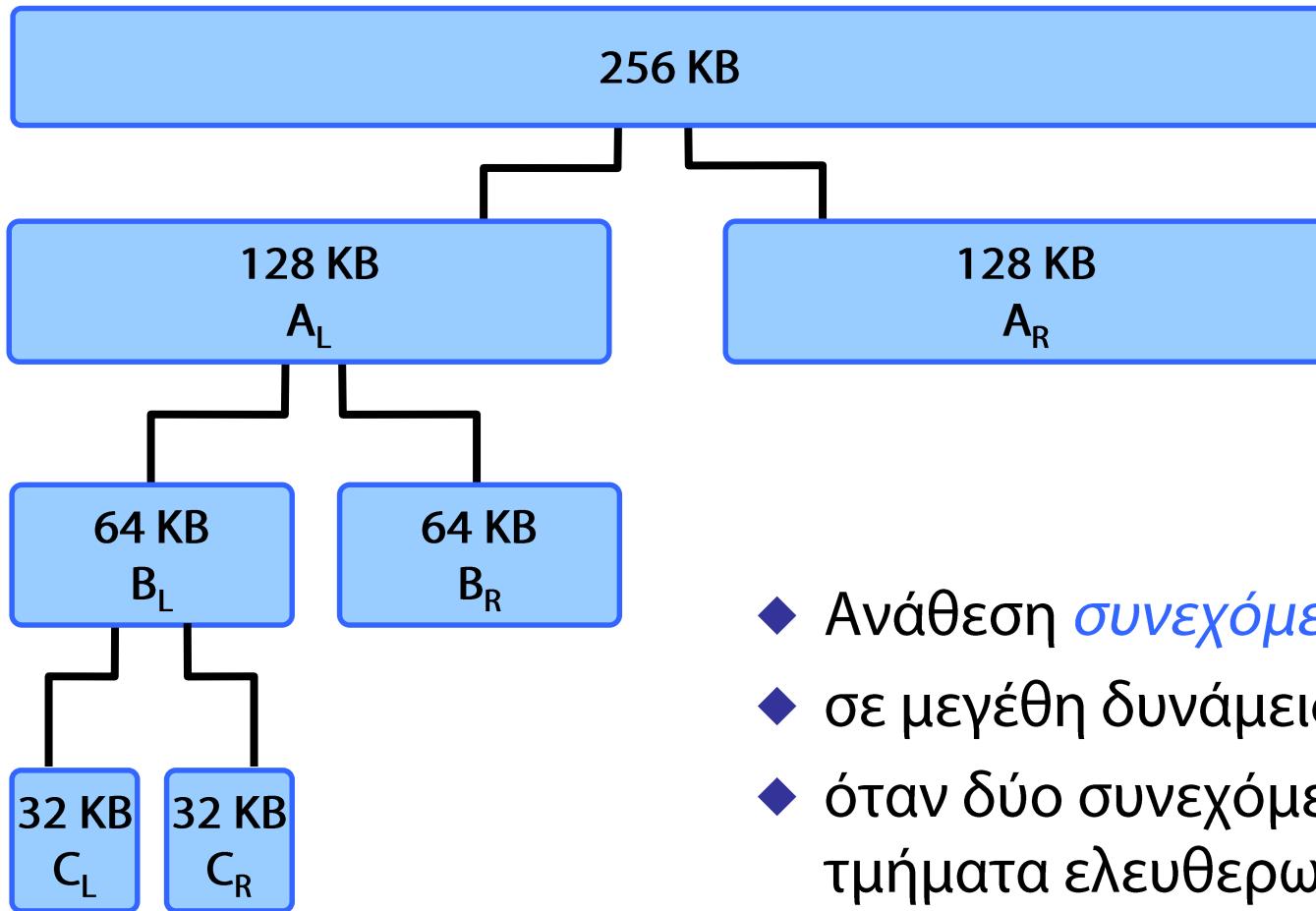
# Εικονική Μνήμη - Σύνοψη

- ◆ Αντικατάσταση Σελίδων
  - ➡ Αλγόριθμοι αντικατάστασης FIFO, βέλτιστος, LRU
  - ➡ Γλοποίηση LRU, Προσέγγιση LRU
- ◆ Ανάθεση Πλαισίων
  - ➡ Thrashing, μοντέλο Συνόλου Εργασίας
- ◆ Buddy Allocators

# Πλαίσια μνήμης για χρήστες και για ΛΣ

- ◆ Το ΛΣ τηρεί μια λίστα για ελεύθερα πλαίσια προς ανάθεση στις διεργασίες χρηστών.
  - ➡ Βασική ιδέα του (demand) paging είναι ότι οι διεργασίες των χρηστών παίρνουν διάσπαρτα πλαίσια στη μνήμη
- ◆ Όμως και το ΛΣ χρειάζεται μνήμη
  - ➡ Επιλέγει από λίστα πλαισίων διαφορετική από αυτή των χρηστών, γιατί:
    - Συχνά χρειάζεται μνήμη πολύ μικρότερη από το μέγεθος της σελίδας, άρα για λόγους αποφυγής κατάτμησης το στοιχειώδες μέγεθος ανάθεσης μπορεί να είναι μικρότερο της σελίδας
    - Σε πολλές περιπτώσεις η φυσική μνήμη που χειρίζεται το ΛΣ πρέπει να είναι συνεχόμενη (π.χ. λόγω αλληλεπίδρασης με κάποια συσκευή που αναμένει δεδομένα σε συνεχόμενες θέσεις φυσικής μνήμης)

# Buddy Allocator



- ◆ Ανάθεση *συνεχόμενης* μνήμης
- ◆ σε μεγέθη δυνάμεις του 2
- ◆ όταν δύο συνεχόμενα τμήματα ελευθερωθούν,  
*ενώνονται*

# Ερωτήσεις;



και στη λίστα:

[OS@lists.cslab.ece.ntua.gr](mailto:OS@lists.cslab.ece.ntua.gr)