

# Πολυνηματικές Αρχιτεκτονικές

# Ο “Νόμος” της απόδοσης των μικροεπεξεργαστών

$$\frac{1}{\text{Performance}} = \frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(instr. count)      (CPI)      (cycle time)

$$\rightarrow \text{Performance} = \frac{\text{IPC} \times \text{Hz}}{\text{instr. count}}$$

- clock speed ( $\uparrow$ Hz)
- αρχιτεκτονικές βελτιστοποιήσεις ( $\uparrow$ IPC): pipelining, superscalar execution, branch prediction, out-of-order execution
- cache ( $\uparrow$ IPC)

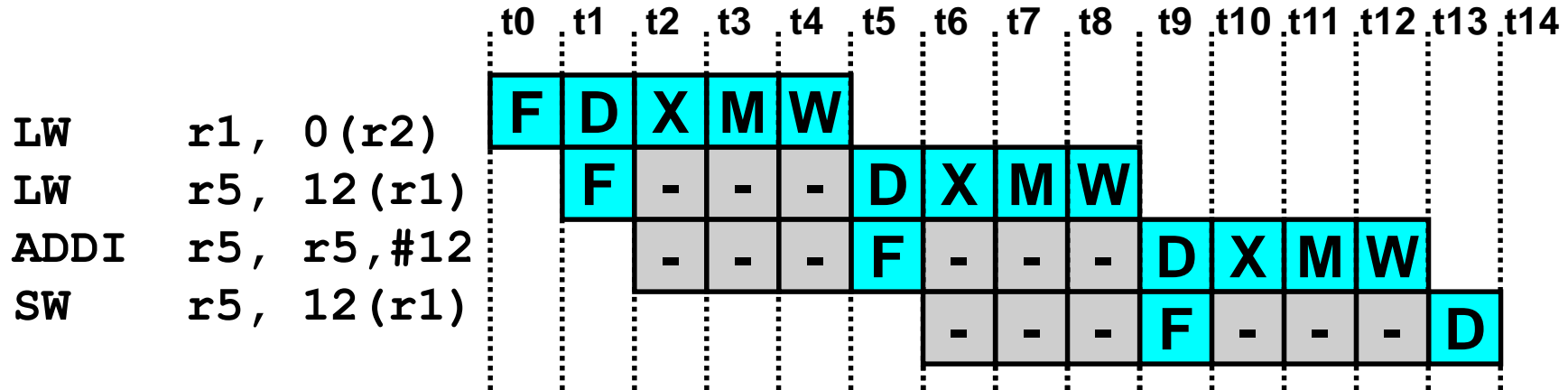
# Περιορισμοί Αύξησης Απόδοσης

- ο ILP εκμεταλλεύεται παράλληλες λειτουργίες, συνήθως μη οριζόμενες από τον προγραμματιστή
  - σε μια «ευθεία» ακολουθία εντολών (χωρίς branches)
  - ανάμεσα σε διαδοχικές επαναλήψεις ενός loop
- δύσκολο το να εξάγουμε ολοένα και περισσότερο ILP από ένα και μόνο νήμα εκτέλεσης
  - εγγενώς χαμηλός ILP σε πολλές εφαρμογές
  - ανεκμετάλλευτες πολλές από τις μονάδες ενός superscalar επεξεργαστή
- συχνότητα ρολογιού: φυσικά εμπόδια στη συνεχόμενη αύξησή της
  - μεγάλη έκλυση θερμότητας, μεγάλη κατανάλωση ισχύος, διαρροή ρεύματος
- *πρέπει να βρούμε άλλον τρόπο πέρα από τον ILP + συχνότητα ρολογιού για να βελτιώσουμε την απόδοση*

# Πολυνηματισμός

- Σκοπός: χρήση πολλών ανεξάρτητων instruction streams από πολλαπλά νήματα εκτέλεσης
- ο παραλληλισμός σε επίπεδο νήματος (Thread-Level Parallelism – TLP) αναπαρίσταται ρητά από τον προγραμματιστή, χρησιμοποιώντας πολλαπλά νήματα εκτέλεσης τα οποία είναι εκ κατασκευής παράλληλα
- πολλά φορτία εργασίας έχουν σαν χαρακτηριστικό τους τον TLP:
  - TLP σε πολυπρογραμματιζόμενα φορτία (εκτέλεση ανεξάρτητων σειριακών εφαρμογών)
  - TLP σε πολυνηματικές εφαρμογές (επιτάχυνση μιας εφαρμογής διαχωρίζοντάς την σε νήματα και εκτελώντας τα παράλληλα)
- οι πολυνηματικές αρχιτεκτονικές χρησιμοποιούν τον TLP σε τέτοια φορτία εργασίας για να βελτιώσουν τα επίπεδα χρησιμοποίησης των μονάδων του επεξεργαστή
  - βελτίωση του throughput πολυπρογραμματιζόμενων φορτίων
  - βελτίωση του χρόνου εκτέλεσης πολυνηματικών εφαρμογών

## Παράδειγμα: κίνδυνοι δεδομένων



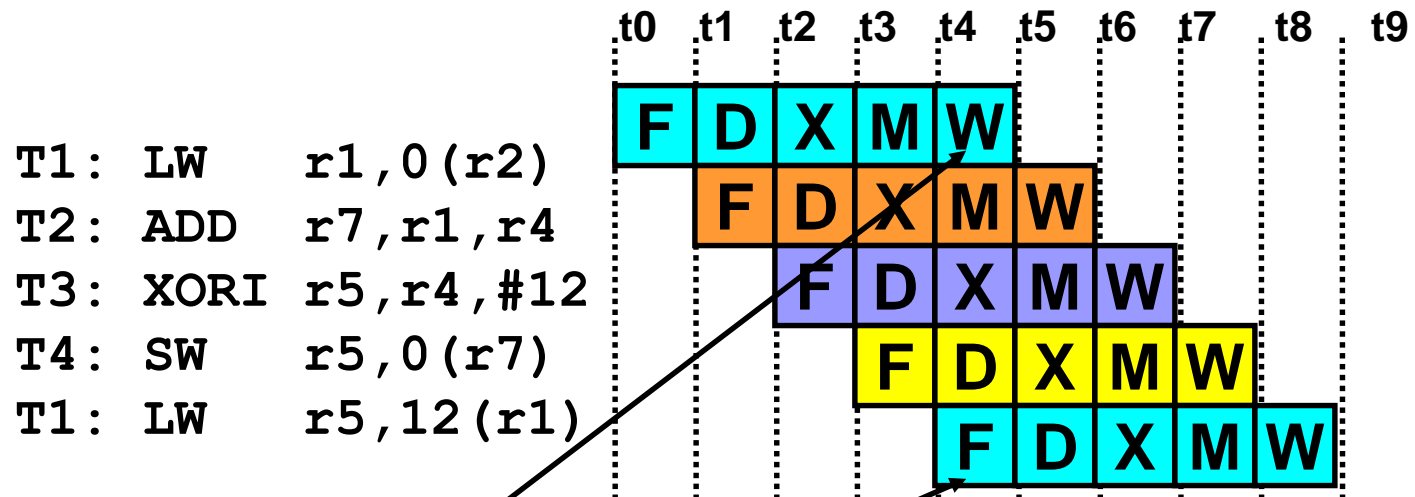
- Οι εξαρτήσεις μεταξύ των εντολών αποτελούν περιοριστικό παράγοντα για την εξαγωγή παραλληλισμού
- Τι μπορεί να γίνει προς αυτή τη κατεύθυνση;

# Αντιμετώπιση με πολυνηματισμό

Πώς μπορούμε να μειώσουμε τις εξαρτήσεις μεταξύ των εντολών σε ένα pipeline?

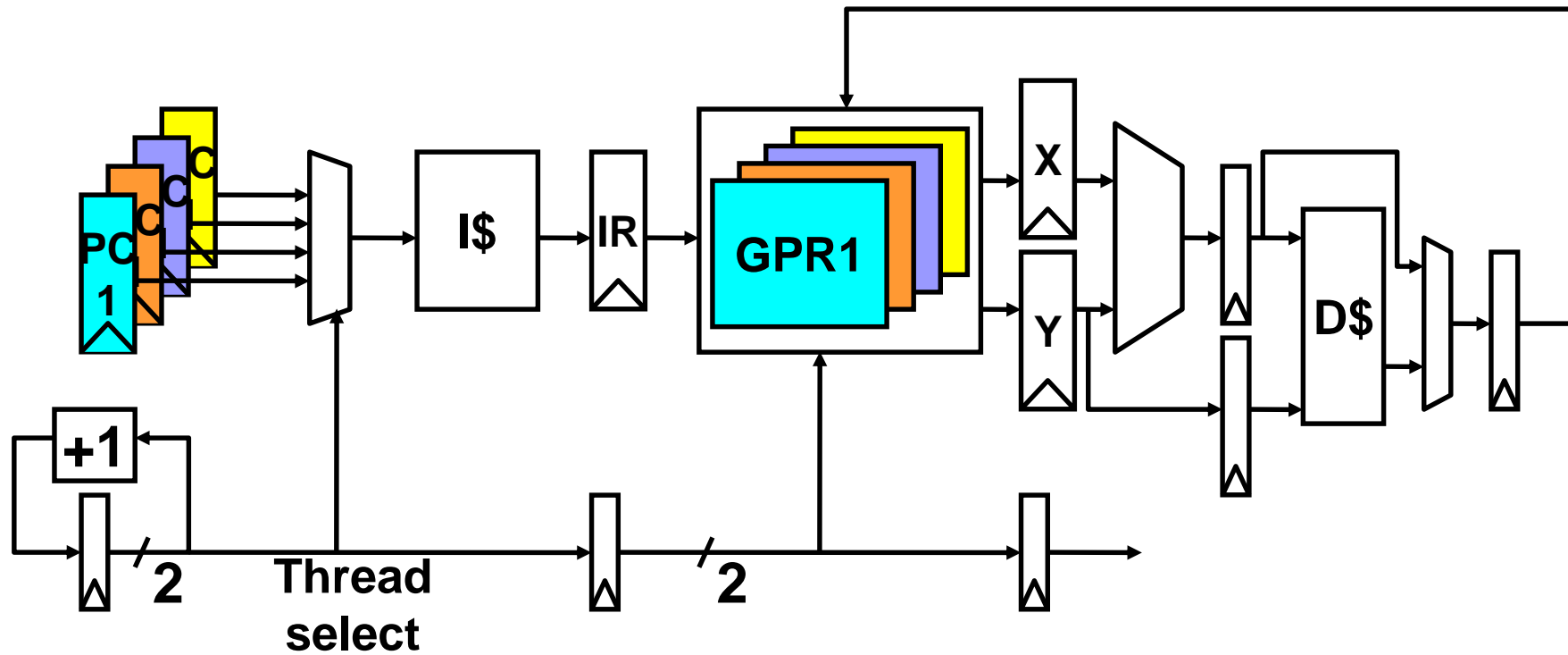
- Ένας τρόπος είναι να επικαλύψουμε την εκτέλεση εντολών από διαφορετικά νήματα στο ίδιο pipeline...

*Επικάλυψη εκτέλεσης 4 νημάτων, T1-T4, στο απλό 5-stage pipeline*



*Η προηγούμενη εντολή στο νήμα ολοκληρώνει το WB προτού η επόμενη εντολή στο ίδιο νήμα διαβάσει το register file*

# Απλή μορφή πολυνηματικού pipeline



- το software «βλέπει» πολλαπλές CPUs
- κάθε νήμα χρειάζεται να διατηρεί τη δική του αρχιτεκτονική κατάσταση
  - program counter
  - general purpose registers
- τα νήματα μοιράζονται τις ίδιες μονάδες εκτέλεσης
- hardware για γρήγορη εναλλαγή των threads
  - πρέπει να είναι πολύ πιο γρήγορη από ένα software-based process switch ( $\approx 100s - 1000s$  κύκλων)

# Πολυνηματισμός

- Πλεονεκτήματα

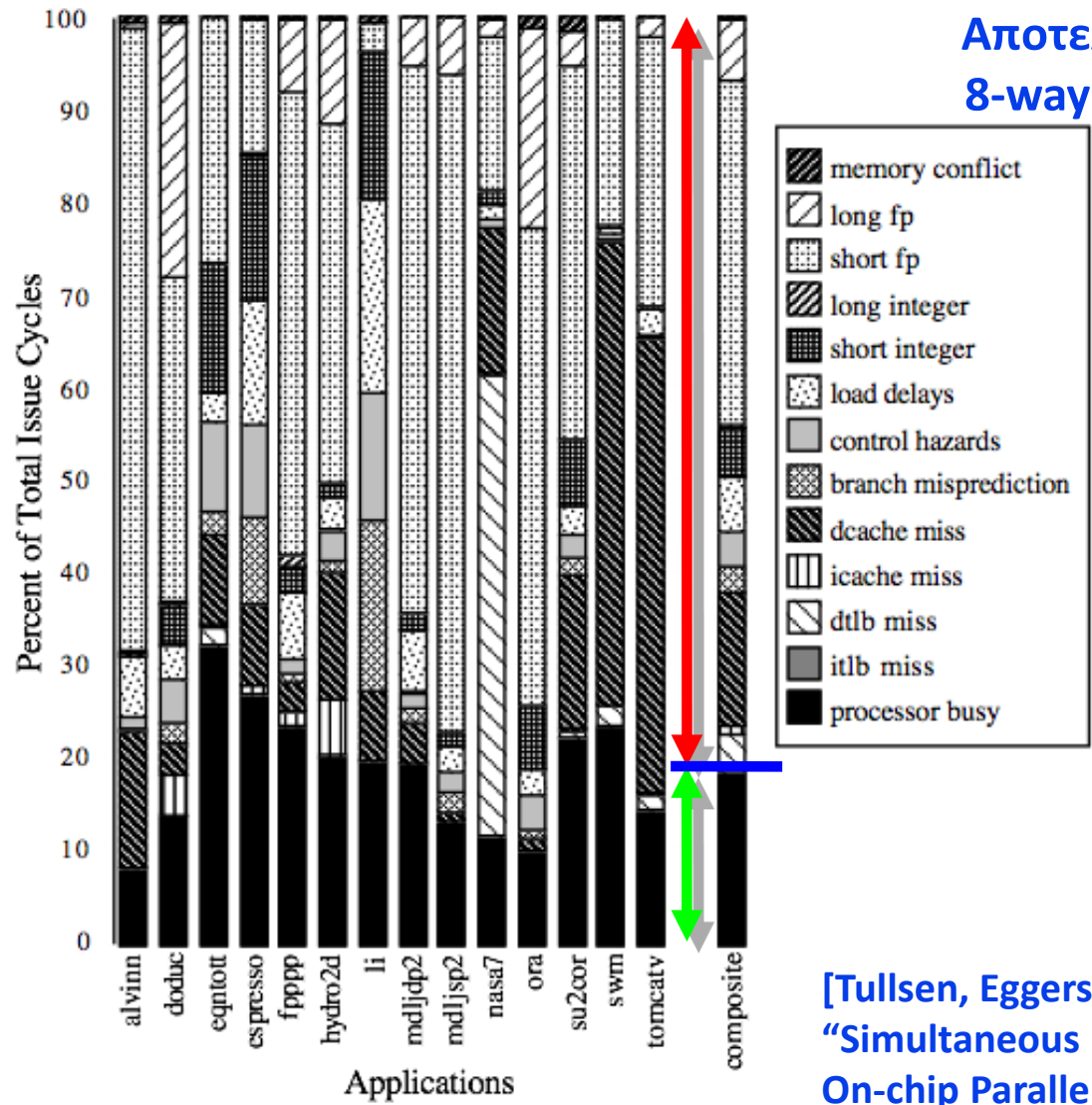
- Δε χρειάζεται dependency checking μεταξύ των εντολών
- Δε χρειάζεται branch prediction
- Αποφυγή bubbles πραγματοποιώντας χρήσιμη δουλειά από άλλα threads
- Βελτίωση system throughput, utilization, latency tolerance

- Μειονεκτήματα

- Πολυπλοκότητα hardware (PCs, register files, thread selection logic, ...)
- Χειρότερο single thread performance (1 instruction every N cycles)
- Υψηλός ανταγωνισμός για πόρους (resource contention for caches & memory)



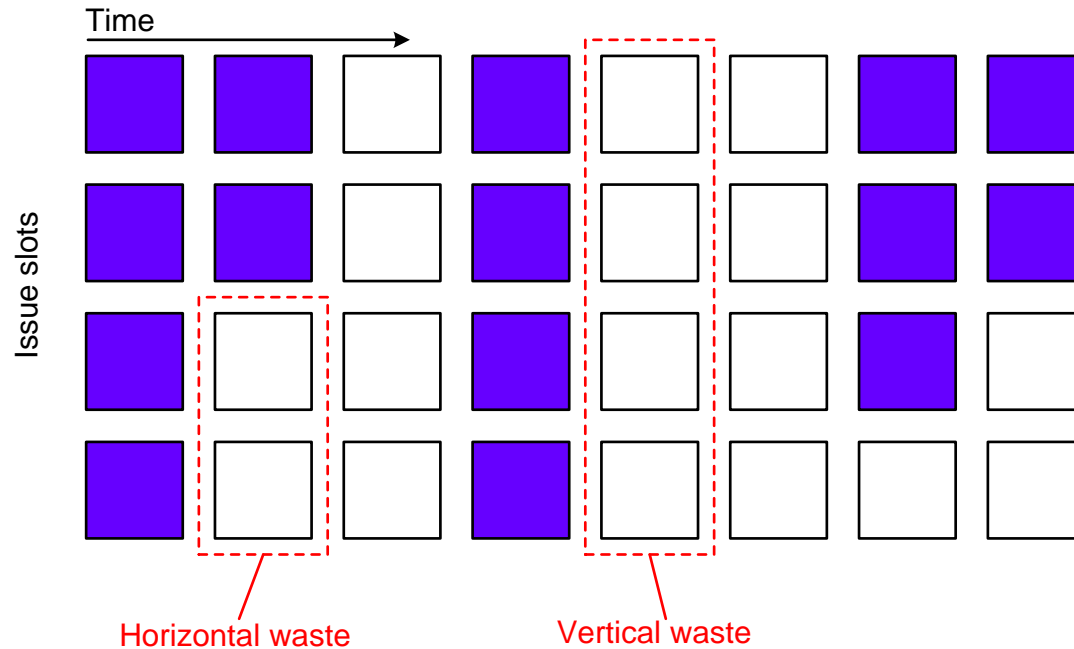
# Για αρκετές εφαρμογές, οι περισσότερες μονάδες εκτέλεσης σε έναν ΟοΟ superscalar μένουν ανεκμετάλλευτες



Αποτελέσματα για 8-way superscalar.

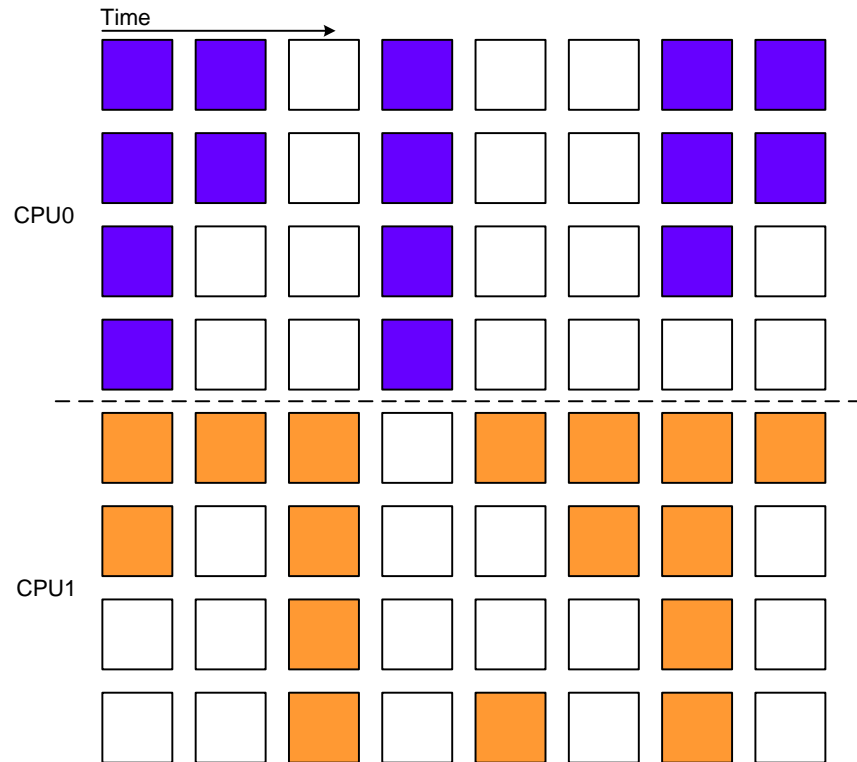
[Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.]

# OoO superscalar



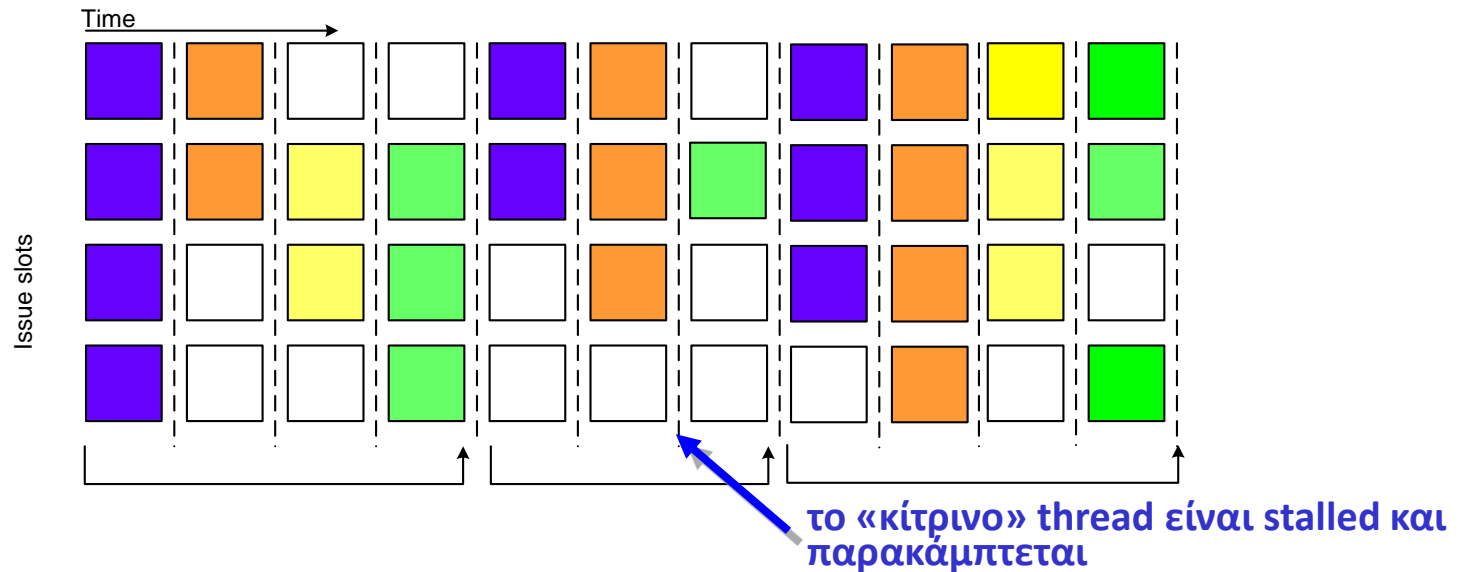
- horizontal waste: εξαιτίας χαμηλού ILP
- vertical waste: εξαιτίας long-latency γεγονότων
  - cache misses
  - pipeline flushes λόγω branch mispredictions

# Chip Multi-Processor



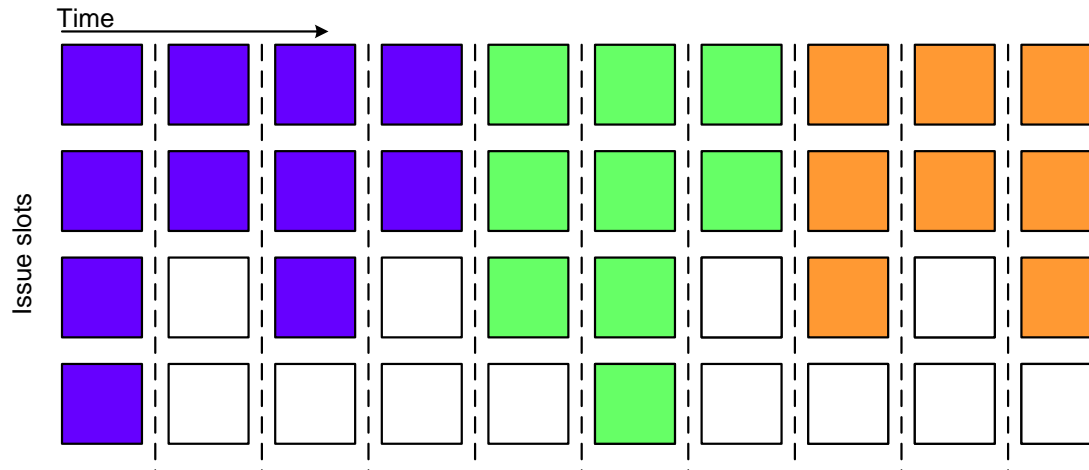
- τα προβλήματα εξακολουθούν να υφίστανται...

# Fine-grained multithreading



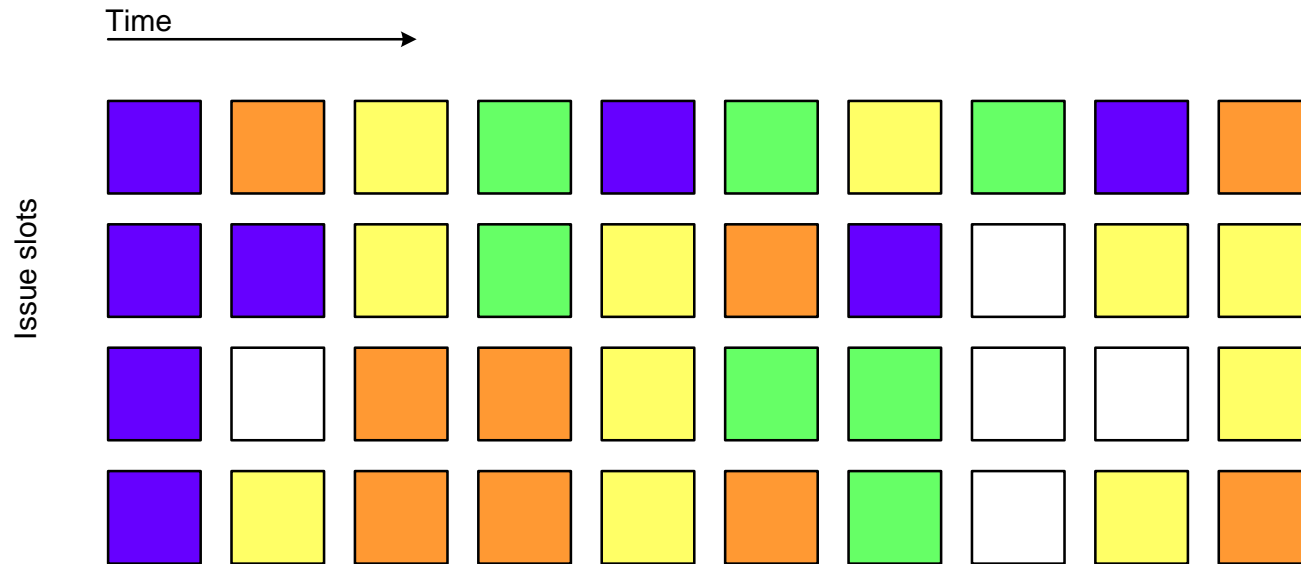
- εναλλαγή μεταξύ των threads σε κάθε κύκλο, με αποτέλεσμα την επικάλυψη της εκτέλεσης των threads
  - η CPU είναι αυτή που κάνει την εναλλαγή σε κάθε κύκλο
- γίνεται με round-robin τρόπο (κυκλικά), παρακάμπτοντας threads τα οποία είναι stalled σε κάποιο long-latency γεγονός
- αντιμετωπίζει το vertical waste, τόσο για μικρά όσο και για μεγάλα stalls, αφού όταν ένα thread είναι stalled το επόμενο μπορεί να γίνει issue
- μειονεκτήματα:
  - δεν αντιμετωπίζει το horizontal waste
  - καθυστερεί την εκτέλεση ενός thread το οποίο είναι έτοιμο να εκτελεστεί, χωρίς stalls, αφού ανάμεσα σε διαδοχικούς κύκλους αυτού του thread παρεμβάλλονται κύκλοι εκτέλεσης από όλα τα υπόλοιπα threads
- e.g., UltraSPARC T1 (“Niagara”), Cray MTA

# Coarse-grained multithreading (“switch-on-event”)



- εναλλαγή thread μόνο μετά από stall του thread που εκτελείται, π.χ. λόγω L2 cache miss
- πλεονεκτήματα:
  - δε χρειάζεται να έχει πολύ γρήγορο μηχανισμό εναλλαγής των threads
  - δεν καθυστερεί την εκτέλεση ενός thread, αφού οι εντολές από άλλα threads γίνονται issue μόνο όταν το thread αντιμετωπίσει κάποιο stall
- μειονεκτήματα:
  - δεν αντιμετωπίζει το horizontal waste
  - σε μικρά stalls, η εναλλαγή του stalled thread και η δρομολόγηση στο pipeline κάποιου έτοιμου thread μπορεί να έχει απώλειες στην απόδοση του πρώτου thread αν τελικά οι κύκλοι που stall-άρει είναι λιγότεροι από τους κόστος εκκίνησης του pipeline με το νέο thread
- εξαιτίας αυτού του start-up κόστους, το coarse-grained multithreading είναι καλύτερο για την μείωση του κόστους από *μεγάλα stalls*, για τα οποία το stall time  $\gg$  pipeline refill time
- e.g. IBM AS/400

# Simultaneous Multithreading (SMT)

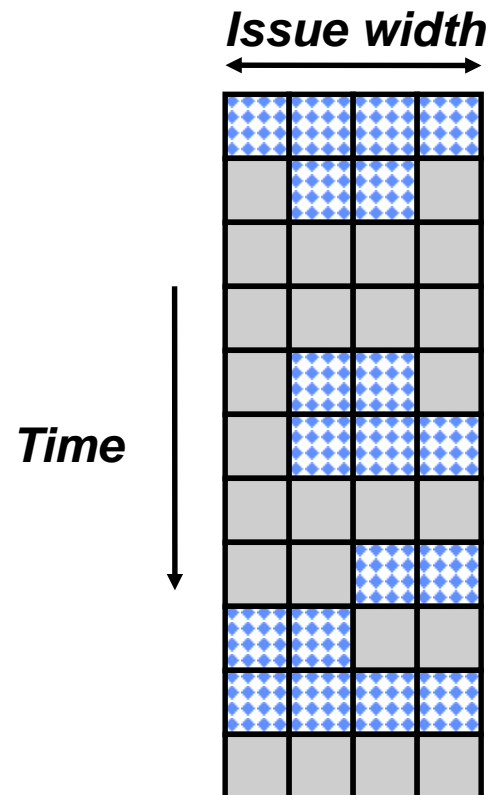
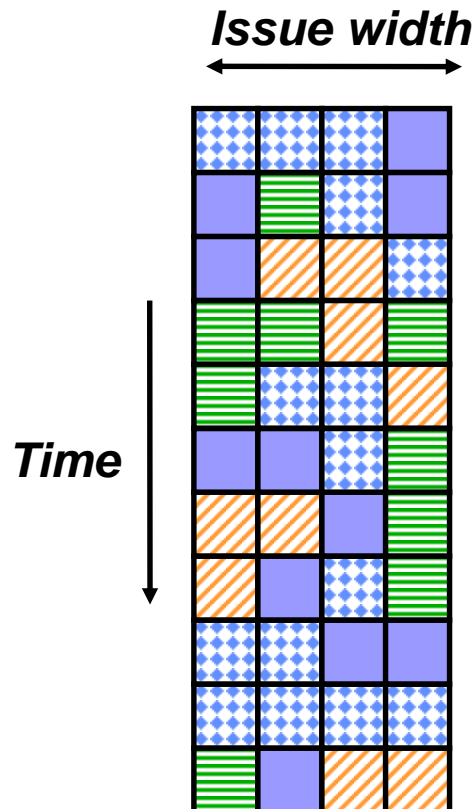


- γίνονται issue εντολές από πολλαπλά νήματα ταυτόχρονα
  - αντιμετωπίζεται το horizontal waste
- όταν ένα νήμα stall-άρει λόγω ενός long-latency γεγονότος, τα υπόλοιπα νήματα μπορούν να δρομολογηθούν και να χρησιμοποιήσουν τις διαθέσιμες μονάδες εκτέλεσης
  - αντιμετωπίζεται το vertical waste
- Μέγιστη χρησιμοποίηση των επεξεργαστικών πόρων από ανεξάρτητες λειτουργίες

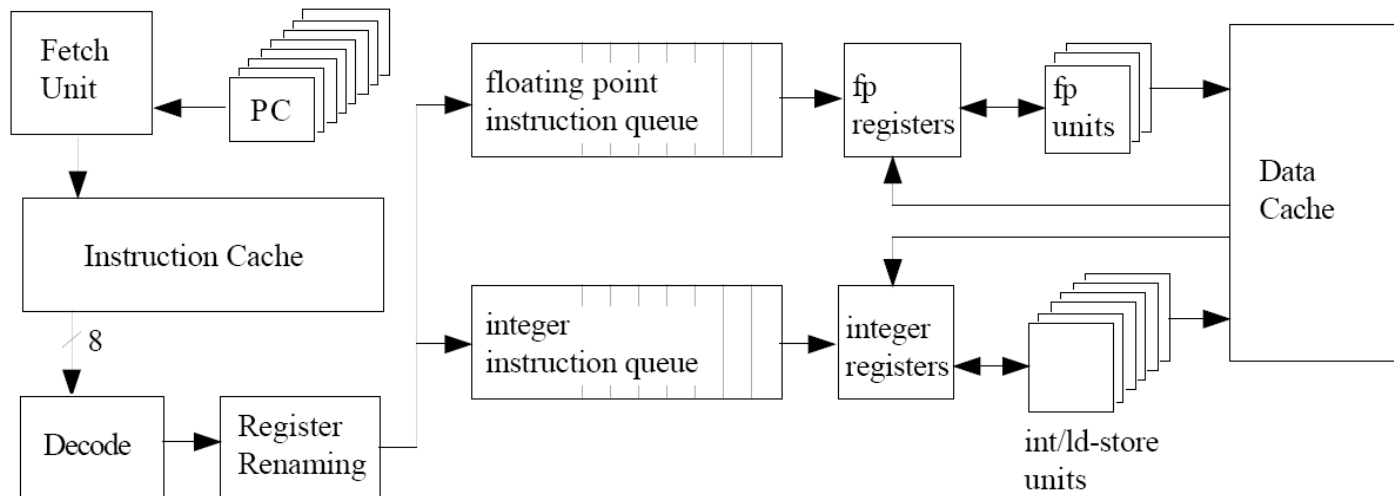
# Προσαρμοστικότητα του SMT στο είδος του διαθέσιμου παραλληλισμού

Για περιοχές με υψηλά επίπεδα TLP, ολόκληρο το εύρος του επεξεργαστή μοιράζεται από όλα τα threads

Για περιοχές με χαμηλά επίπεδα TLP, ολόκληρο το εύρος του επεξεργαστή είναι διαθέσιμο για την εκμετάλλευση του (όποιου) ILP



# Αρχιτεκτονική SMT



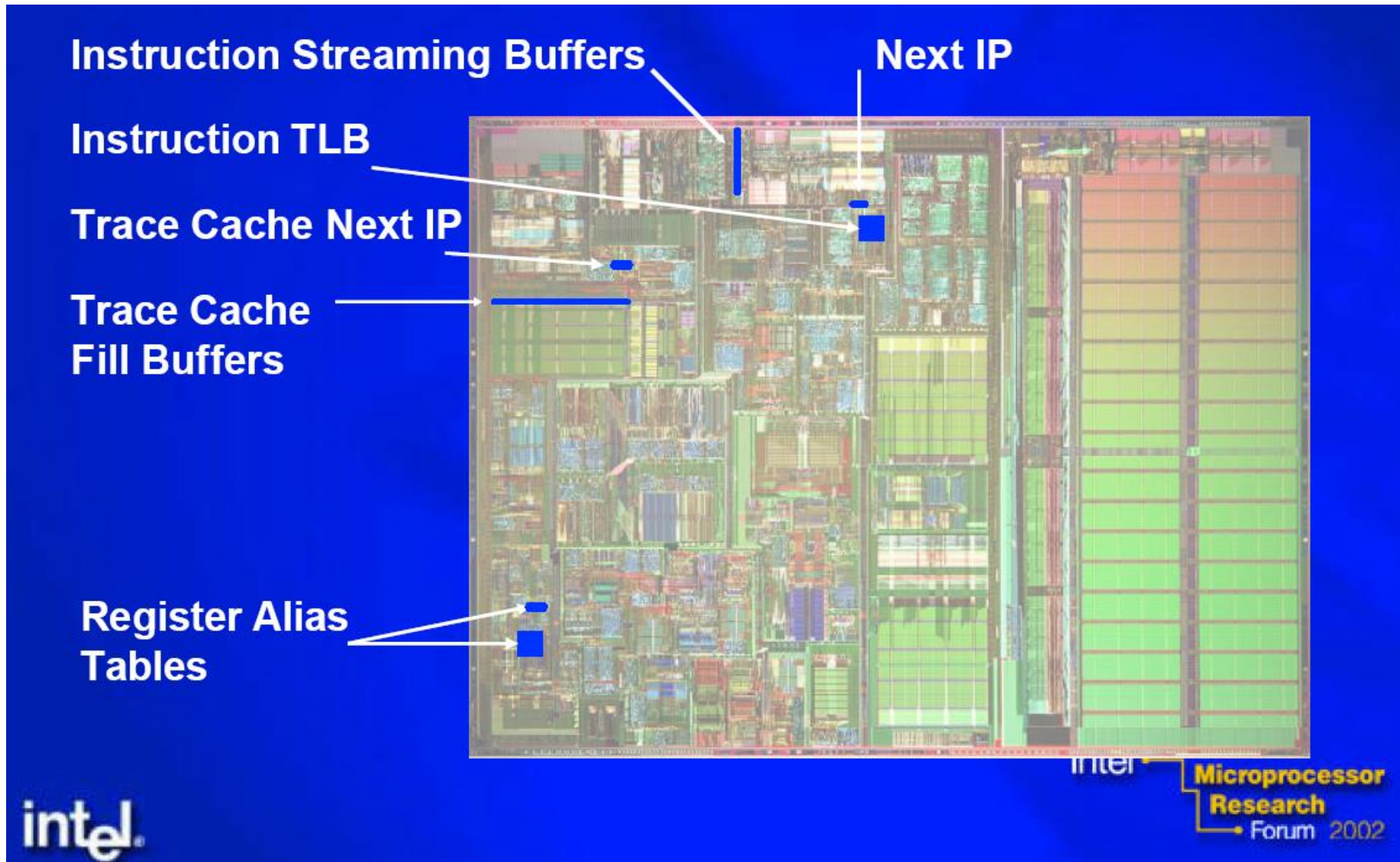
- Βασικές επεκτάσεις σε σχέση με μια συμβατική superscalar αρχιτεκτονική
  - **πολλαπλοί program counters** και κατάλληλος μηχανισμός μέσω του οποίου η fetch unit επιλέγει κάποιον από αυτούς σε κάθε κύκλο (π.χ. με βάση κάποια συγκεκριμένη πολιτική)
  - **thread-id σε κάθε BTB entry** για την αποφυγή πρόβλεψης branches που ανήκουν σε άλλα threads
  - **ξεχωριστή RAS για κάθε thread** για την πρόβλεψη της διεύθυνσης επιστροφής μετά από κλήση υπορουτίνας σε κάθε thread
  - **ξεχωριστός ROB για κάθε thread** προκειμένου το commit και η διαχείριση των mispredicted branches + των exceptions να γίνεται ανεξάρτητα για κάθε thread
  - **μεγαλύτερο register file**, για να υποστηρίζει λογικούς καταχωρητές για όλα τα threads + επιπλέον καταχωρητές για register renaming
  - **ξεχωριστό renaming table για κάθε thread**
    - » εφόσον οι λογικοί καταχωρητές για όλα τα threads απεικονίζονται όλοι σε διαφορετικούς φυσικούς καταχωρητές, οι εντολές από διαφορετικά threads μπορούν να αναμιχθούν μετά το renaming χωρίς να συγχέονται οι source και target operands ανάμεσα στα threads



# Case-study 1: SMT in Hyperthreading technology (~2002)

- 2-way SMT
  - το λειτουργικό «βλέπει» 2 CPUs
  - εκτελεί δύο διεργασίες ταυτόχρονα
    - » πολυπρογραμματιζόμενα φορτία
    - » πολυνηματικές εφαρμογές
- ο φυσικός επεξεργαστής διατηρεί την αρχιτεκτονική για 2 λογικούς επεξεργαστές
- επιπλέον κόστος για υποστήριξη 2 ταυτόχρονων νημάτων εκτέλεσης < 5% του αρχικού hardware

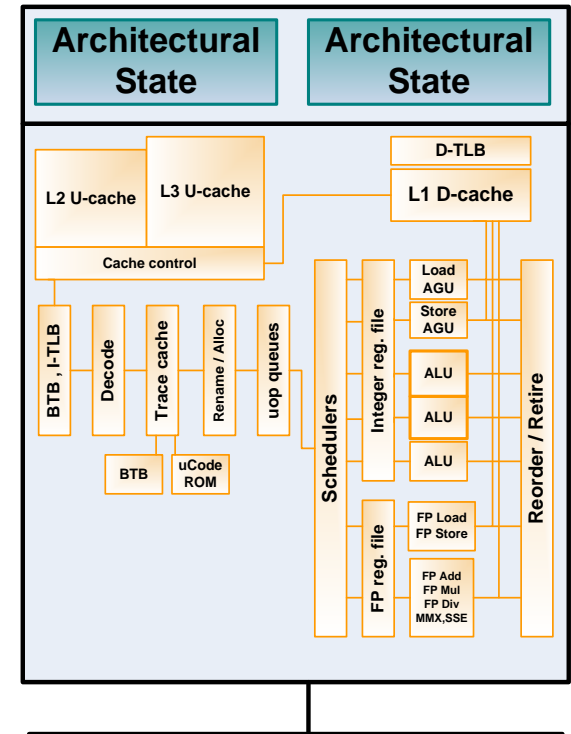
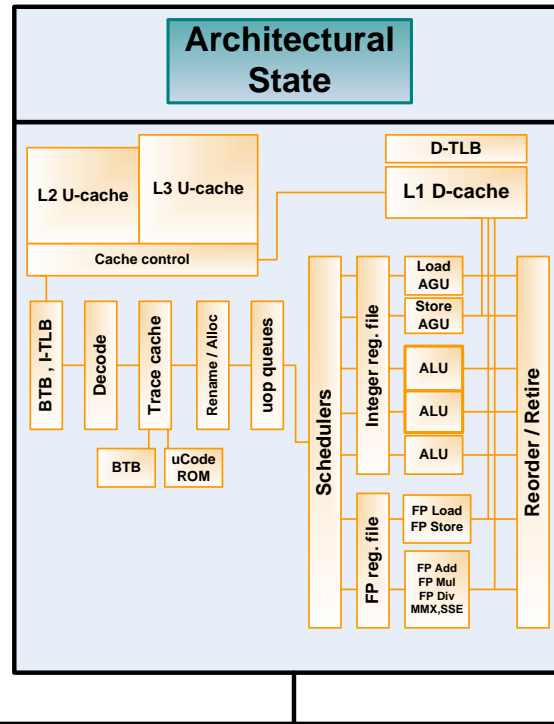
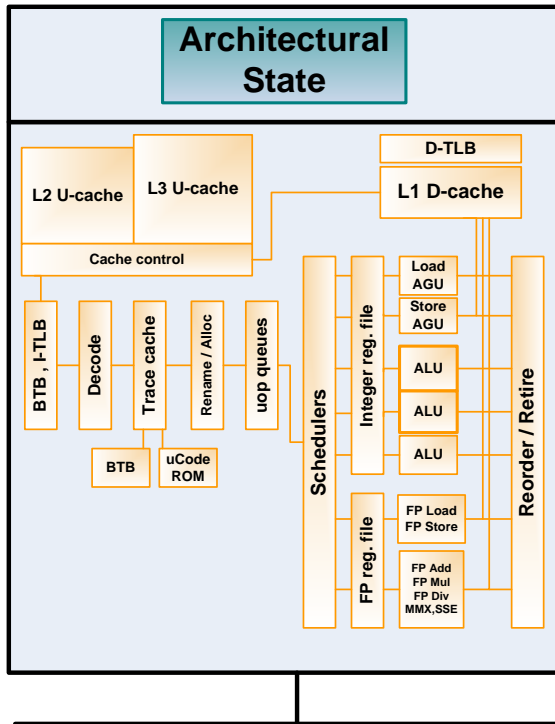
# Τι προστέθηκε...



# Επεξεργαστικοί πόροι: πολλαπλά αντίγραφα vs. διαμοιρασμός

## Multiprocessor

## Hyperthreading



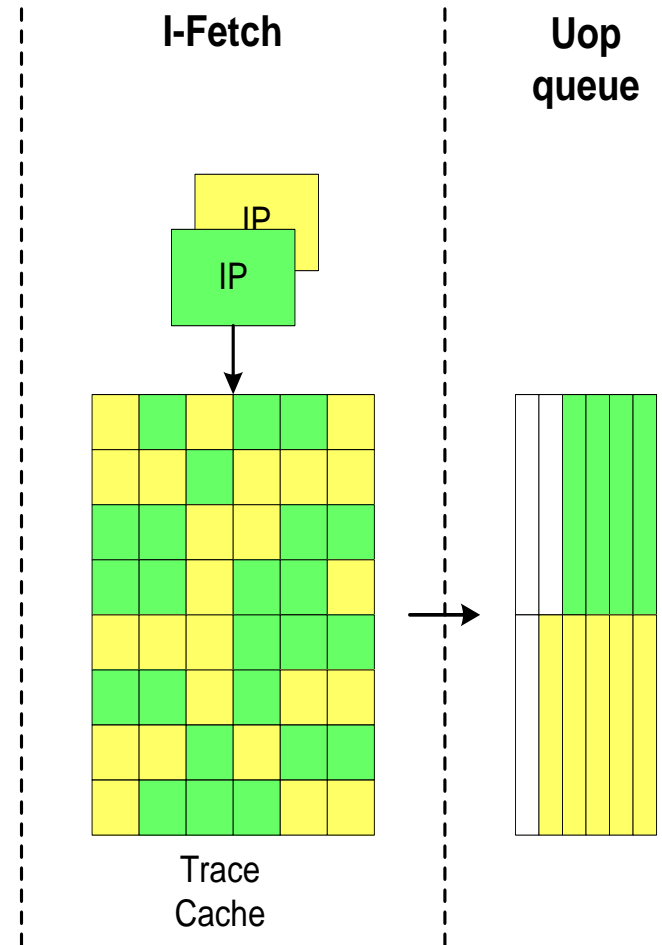
- στους πολυεπεξεργαστές τα resources βρίσκονται σε πολλαπλά αντίγραφα
- στο Hyper-threading τα resources διαμοιράζονται

# Διαχείριση επεξεργαστικών πόρων

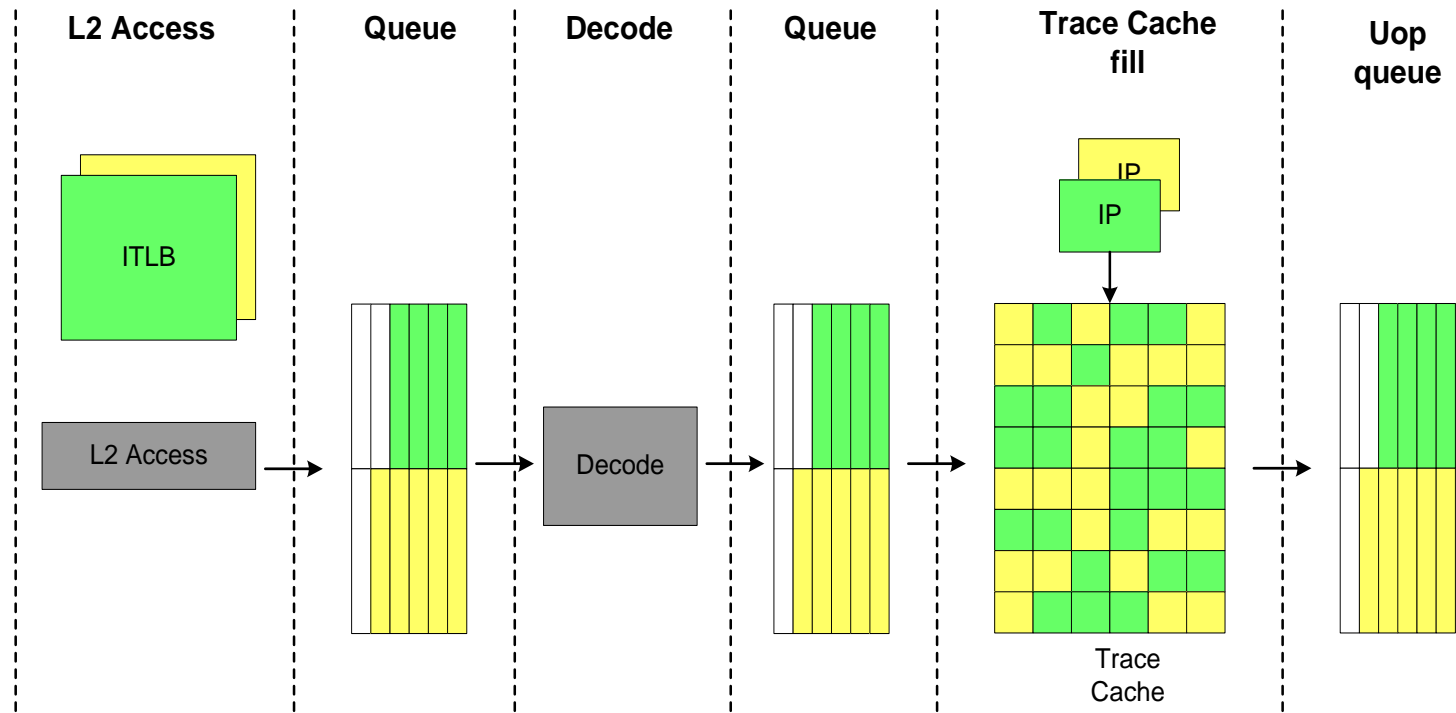
- Πόροι σε πολλαπλά αντίγραφα:
  - αρχιτεκτονική κατάσταση: GPRs, control registers, APICs
  - instruction pointers, renaming hardware
  - smaller resources: ITLBs, branch target buffer, return address stack
- Στατικά διαχωρισμένοι πόροι:
  - ROB, Load/Store queues, instruction queues
  - κάθε νήμα μπορεί να χρησιμοποιήσει έως τα μισά (το πολύ) entries κάθε τέτοιου πόρου
    - » ένα νήμα δεν μπορεί να οικειοποιηθεί το σύνολο των entries, στερώντας τη δυνατότητα από το άλλο νήμα να συνεχίσει την εκτέλεσή του
    - » εξασφαλίζεται η απρόσκοπτη πρόοδος ενός νήματος, ανεξάρτητα από την πρόοδο του άλλου νήματος
- Δυναμικά διαμοιραζόμενοι πόροι (κατ' απαίτηση):
  - out-of-order execution engine, caches

# Pentium 4 w/ Hyper-Threading: Front End – trace cache hit

- Trace cache
  - «ειδική» instruction cache που κρατάει αποκωδικοποιημένες μικρο-εντολές
  - σε κάθε cache line αποθηκεύονται οι μικρο-εντολές στη σειρά με την οποία εκτελούνται (π.χ. εντολή άλματος μαζί με την ακολουθία εντολών στην προβλεφθείσα κατεύθυνση)
- Σε ταυτόχρονη ζήτηση από τους 2 LPs, η πρόσβαση εναλλάσσεται κύκλο-ανά-κύκλο
- Όταν μόνο 1 LP ζητάει πρόσβαση, μπορεί να χρησιμοποιήσει την TC στο μέγιστο δυνατό fetch bandwidth (3 μIPC)

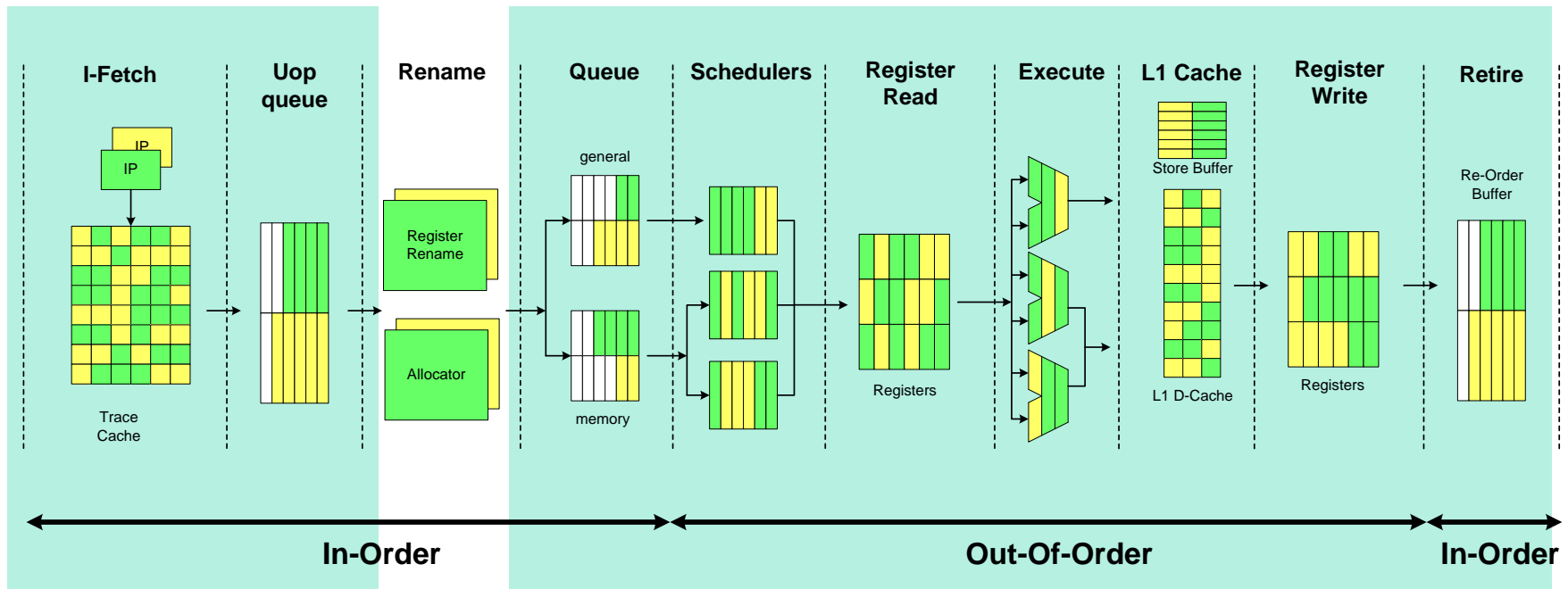


# Pentium 4 w/ Hyper-Threading: Front End – trace cache miss



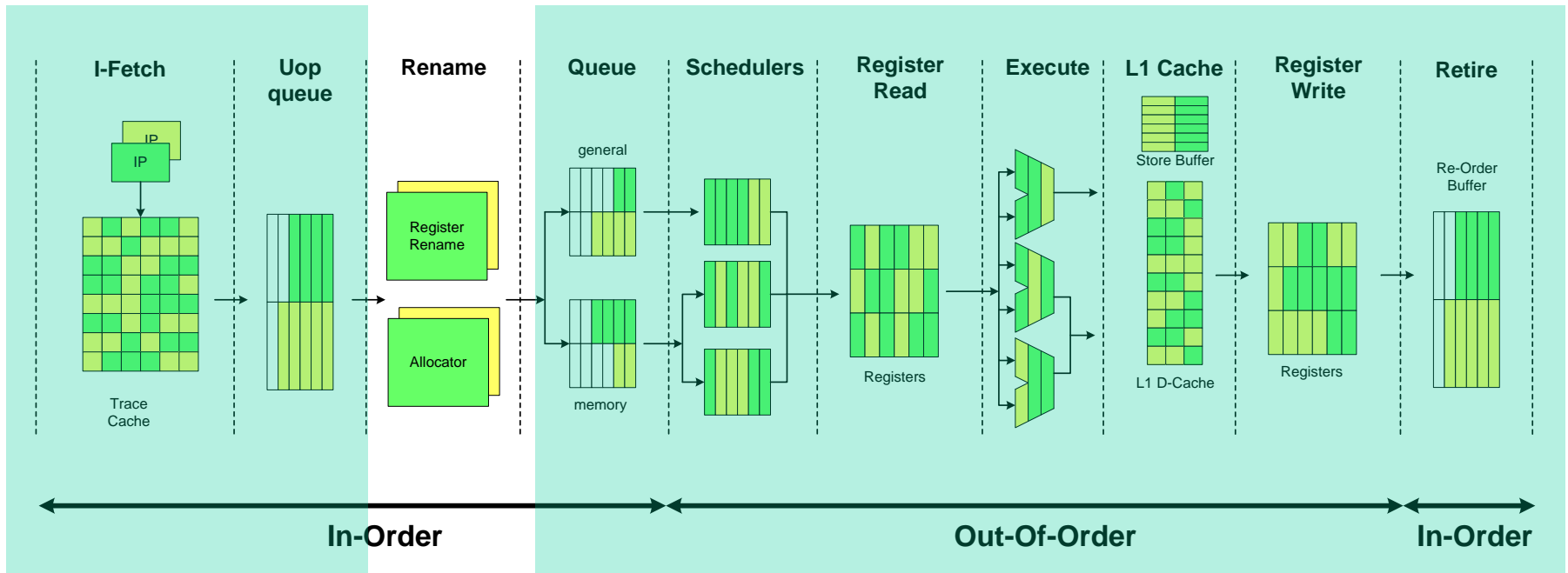
- L2 cache
  - η πρόσβαση γίνεται με FCFS τρόπο
- Decode
  - σε ταυτόχρονη ζήτηση από τους 2 LPs, η πρόσβαση εναλλάσσεται, αλλά με πιο coarse-grained τρόπο (δηλ. όχι κύκλο-ανά-κύκλο, αλλά k κύκλους-ανά-k κύκλους)

# Pentium 4 w/ Hyper-Threading: Execution engine



- **Allocator:** εκχωρεί entries σε κάθε LP
  - 63/126 ROB entries
  - 24/48 Load buffer entries
  - 12/24 Store buffer entries
  - 128/128 Integer physical registers
  - 128/128 FP physical registers
- Σε ταυτόχρονη ζήτηση από τους 2 LPs, η πρόσβαση εναλλάσσεται κύκλο-ανά-κύκλο
- stall-άρει έναν LP όταν επιχειρεί να χρησιμοποιήσει περισσότερα από τα μισά entries των στατικά διαχωρισμένων πόρων

# Pentium 4 w/ Hyper-Threading: Execution engine

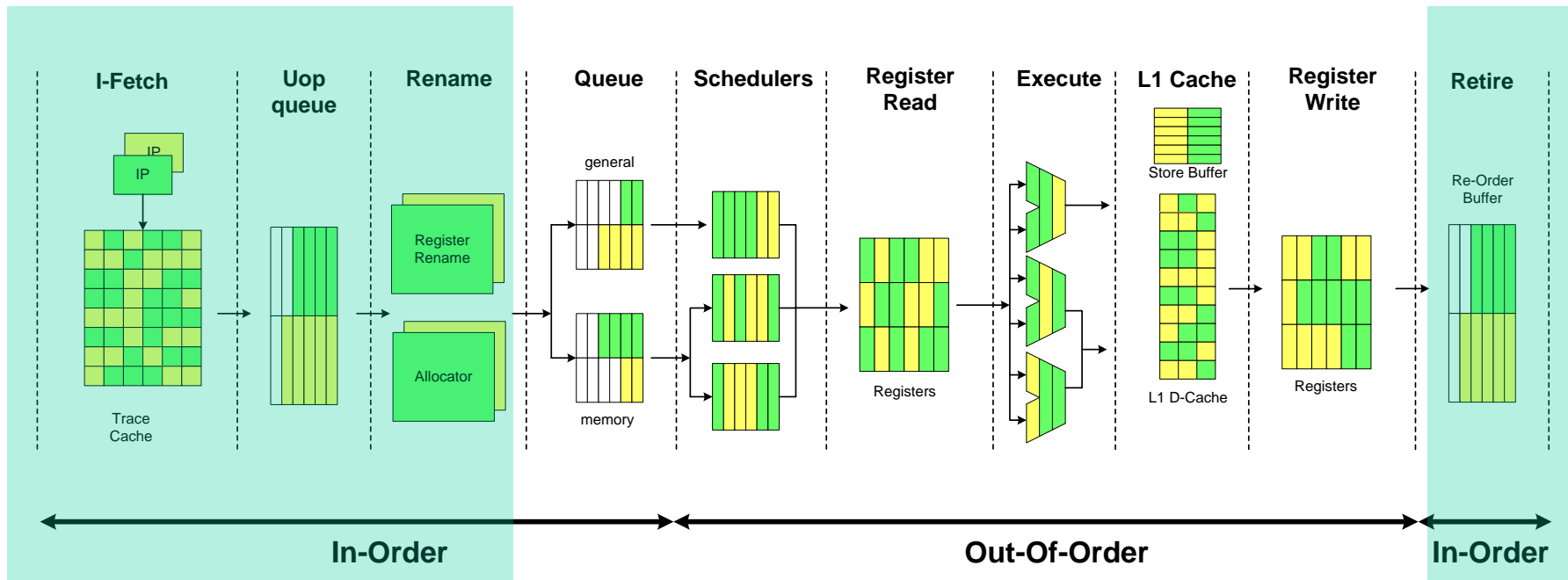


- Register renaming unit

- επεκτείνει δυναμικά τους architectural registers απεικονίζοντάς τους σε ένα μεγαλύτερο σύνολο από physical registers
- ξεχωριστό register map table για κάθε LP



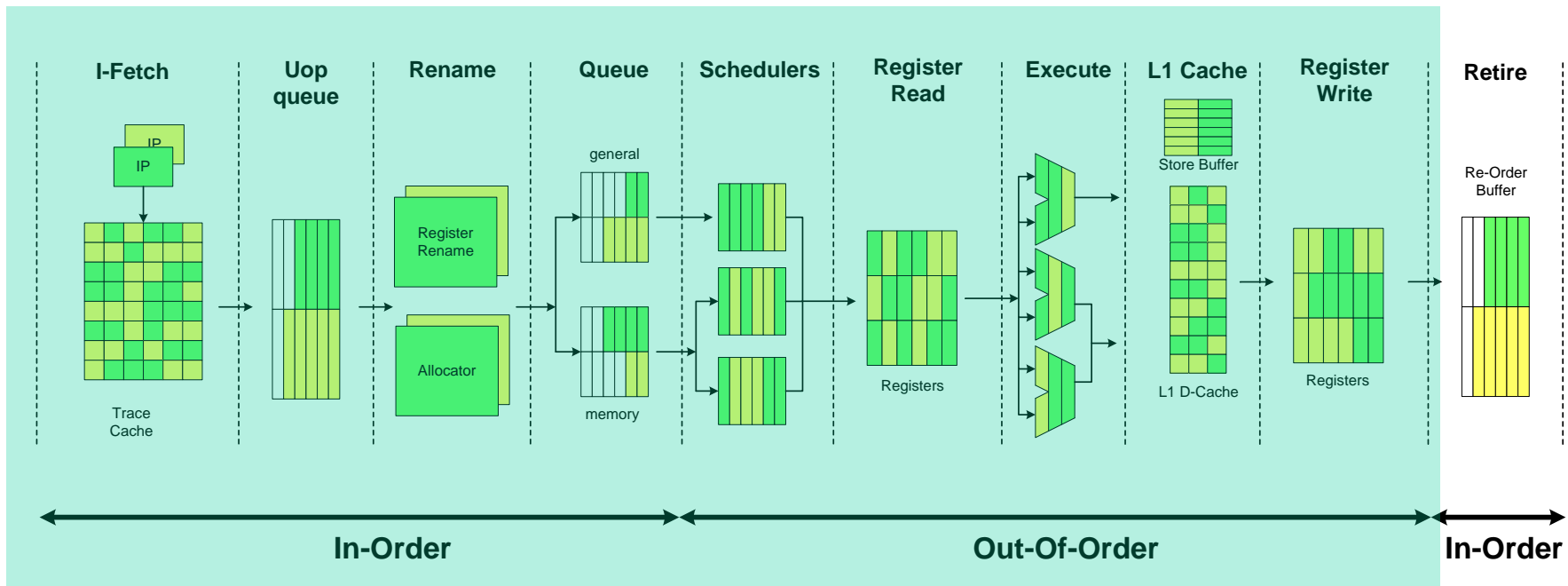
# Pentium 4 w/ Hyper-Threading: Execution engine



- Schedulers / Execution units

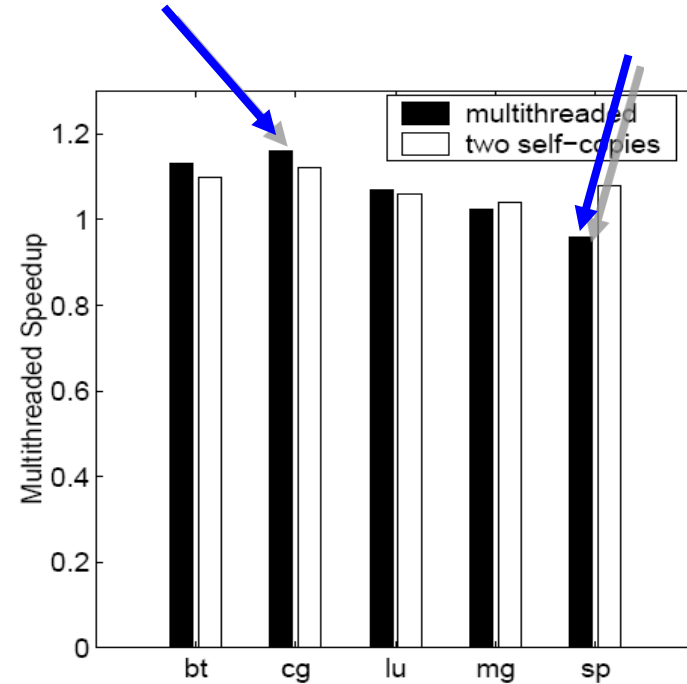
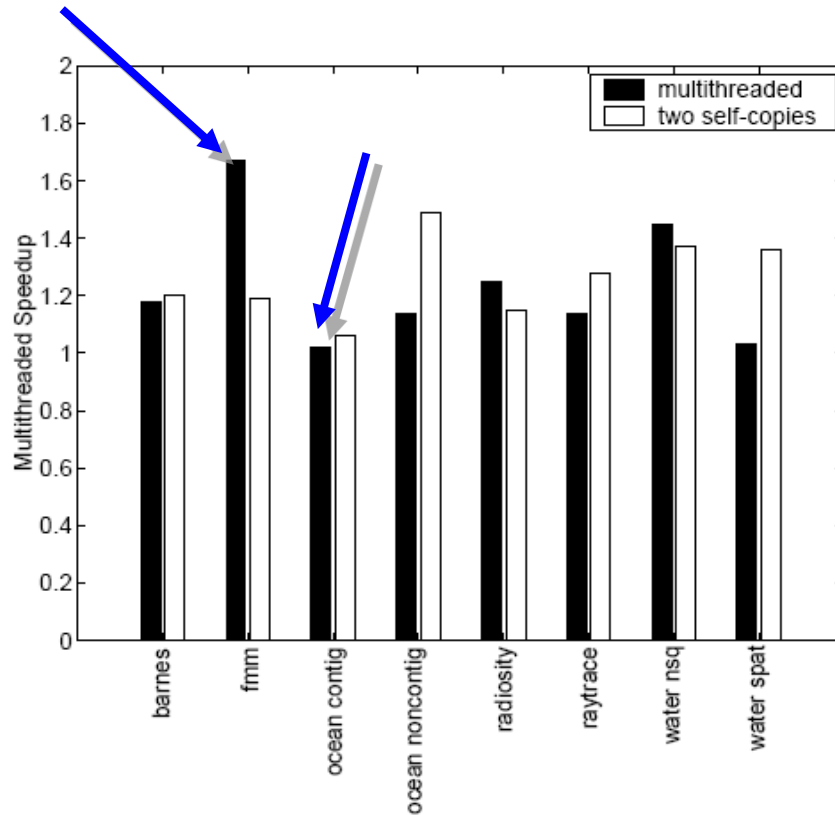
- στην πραγματικότητα δε (χρειάζεται να) ξέρουν σε ποιον LP ανήκει η εντολή που εκτελούν
- general+memory queues στέλνουν μικρο-εντολές στους δρομολογητές, με την πρόσβαση να εναλλάσσεται κύκλο-ανά-κύκλο ανάμεσα στους 2 LPs
- 6  $\mu$ IPC dispatch/execute bandwidth ( $\rightarrow$  3  $\mu$ IPC per-LP effective bandwidth, όταν και οι 2 LPs είναι ενεργοί)

# Pentium 4 w/ Hyper-Threading: Retirement



- Η αρχιτεκτονική κατάσταση κάθε LP γίνεται commit με τη σειρά προγράμματος, εναλλάσσοντας την πρόσβαση στον ROB ανάμεσα στους 2 LPs κύκλο-ανά-κύκλο
- 3  $\mu$ IPC retirement bandwidth

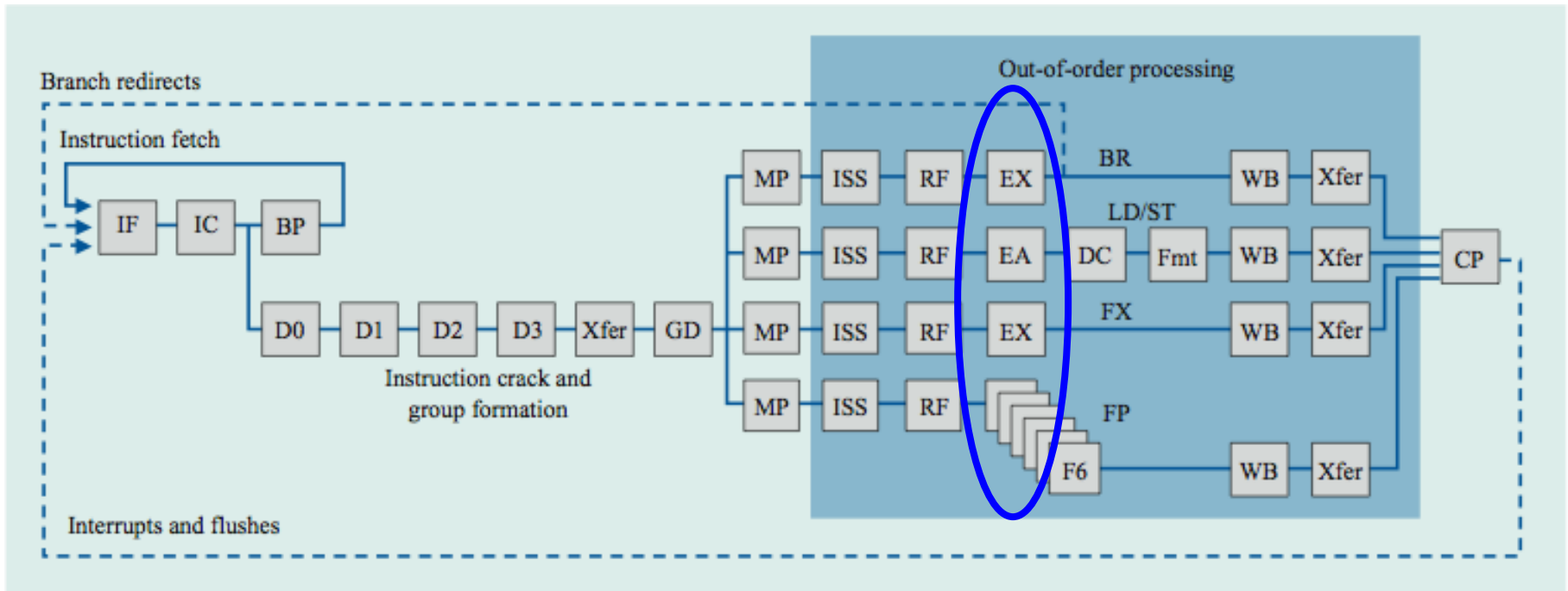
# Multithreaded speedup



From: Tuck and Tullsen, "Initial Observations of the Simultaneous Multithreading Pentium 4 Processor", PACT 2003.

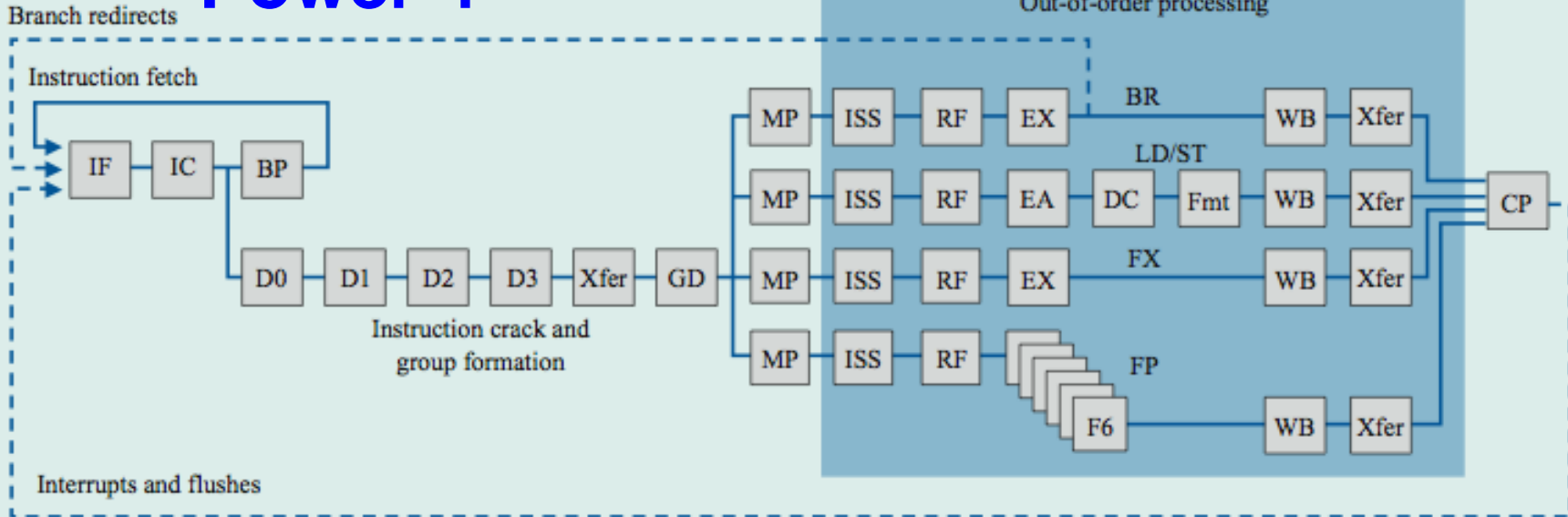
- SPLASH2 Benchmarks: 1.02 – 1.67
- NAS Parallel Benchmarks: 0.96 – 1.16

# Case-study 2 Power 5 : επέκταση του Power 4 για υποστήριξη SMT (2004)



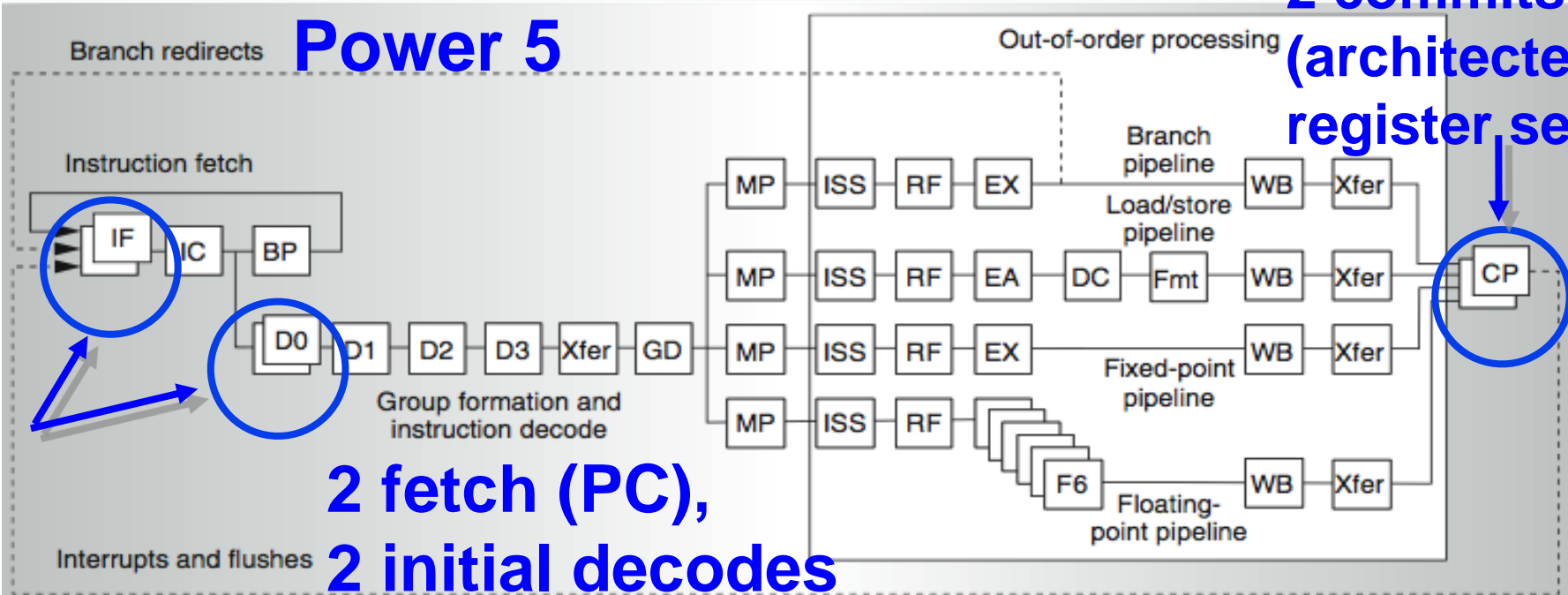
- Single-threaded «προκάτοχος» του Power5
- 8 execution units
  - 2 Float. Point, 2 Load/Store, 2 Fixed Point, 1 Branch, 1 Conditional Reg. unit
  - κάθε μία μπορεί να κάνει issue 1 εντολή ανά κύκλο
- Execution bandwidth: 8 operations ανά κύκλο
  - $(1 \text{ fpadd} + 1 \text{ fpmult}) \times 2\text{FP} + 1 \text{ load/store} \times 2 \text{ LD/ST} + 1 \text{ integer} \times 2 \text{ FX}$

# Power 4



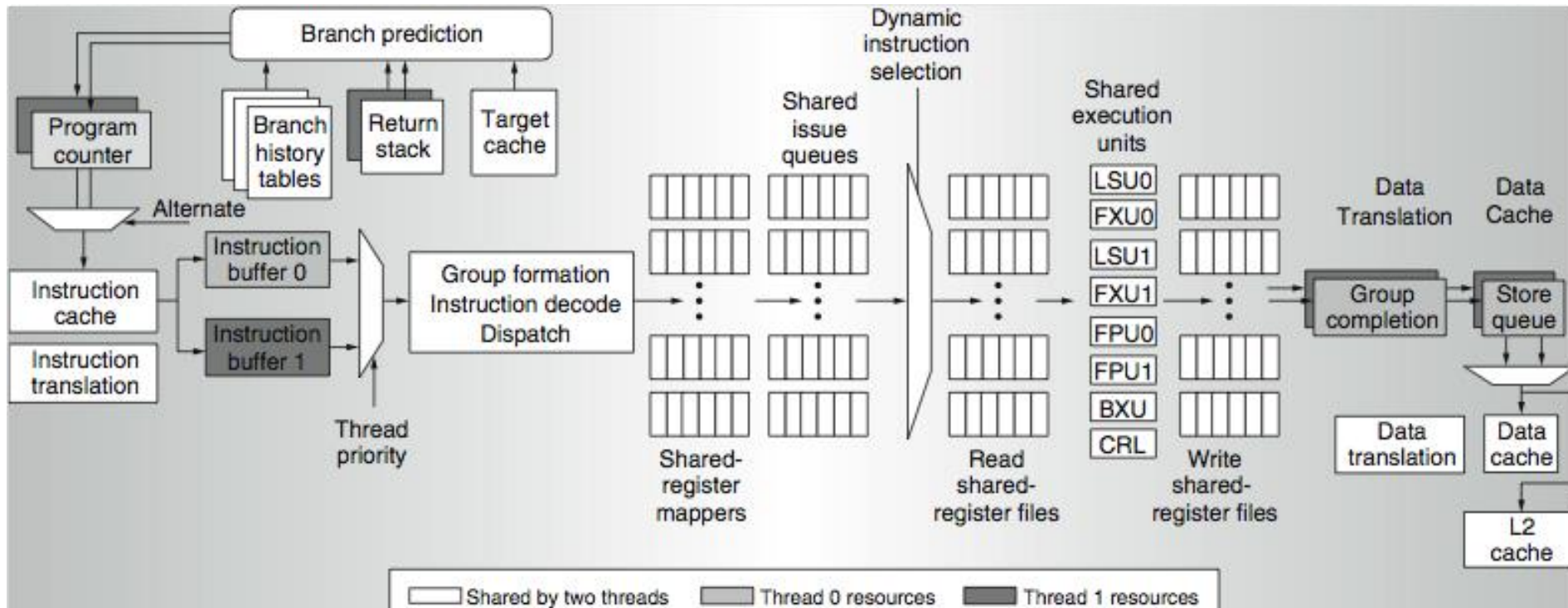
**2 commits  
(architected  
register sets)**

# Power 5



**2 fetch (PC),  
2 initial decodes**

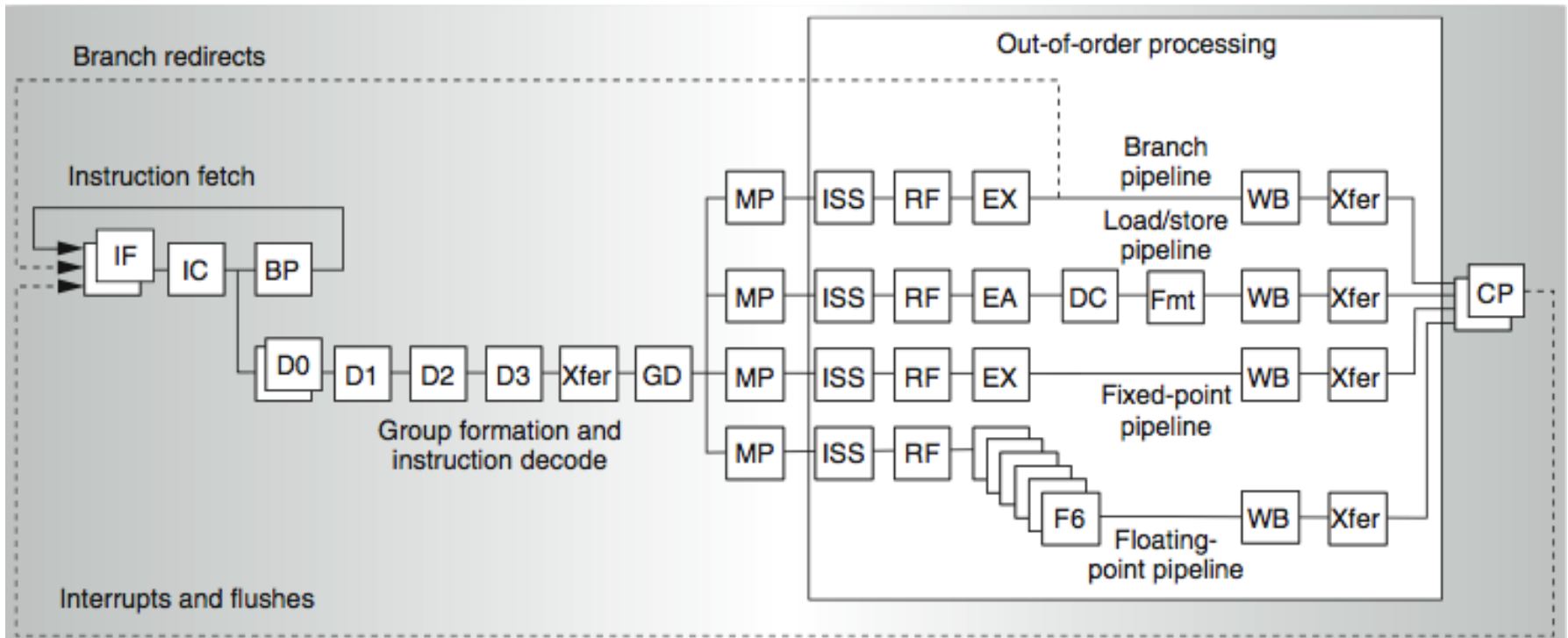
# SMT resource management



## Αλλαγές στον Power 5 για να υποστηρίζεται το SMT

- Αύξηση συσχετιστικότητας της L1 instruction cache και των ITLBs
- Ξεχωριστές Load/Store queues για κάθε νήμα
- Αύξηση μεγέθους των L2 (1.92 vs. 1.44 MB) και L3 caches
- Ξεχωριστό instruction prefetch hardware και instruction buffers για κάθε νήμα
- Αύξηση των registers από 152 σε 240
- Αύξηση του μεγέθους των issue queues
- Αύξηση μεγέθους κατά 24% σε σχέση με τον Power4 εξαιτίας της προσθήκης hardware για υποστήριξη SMT

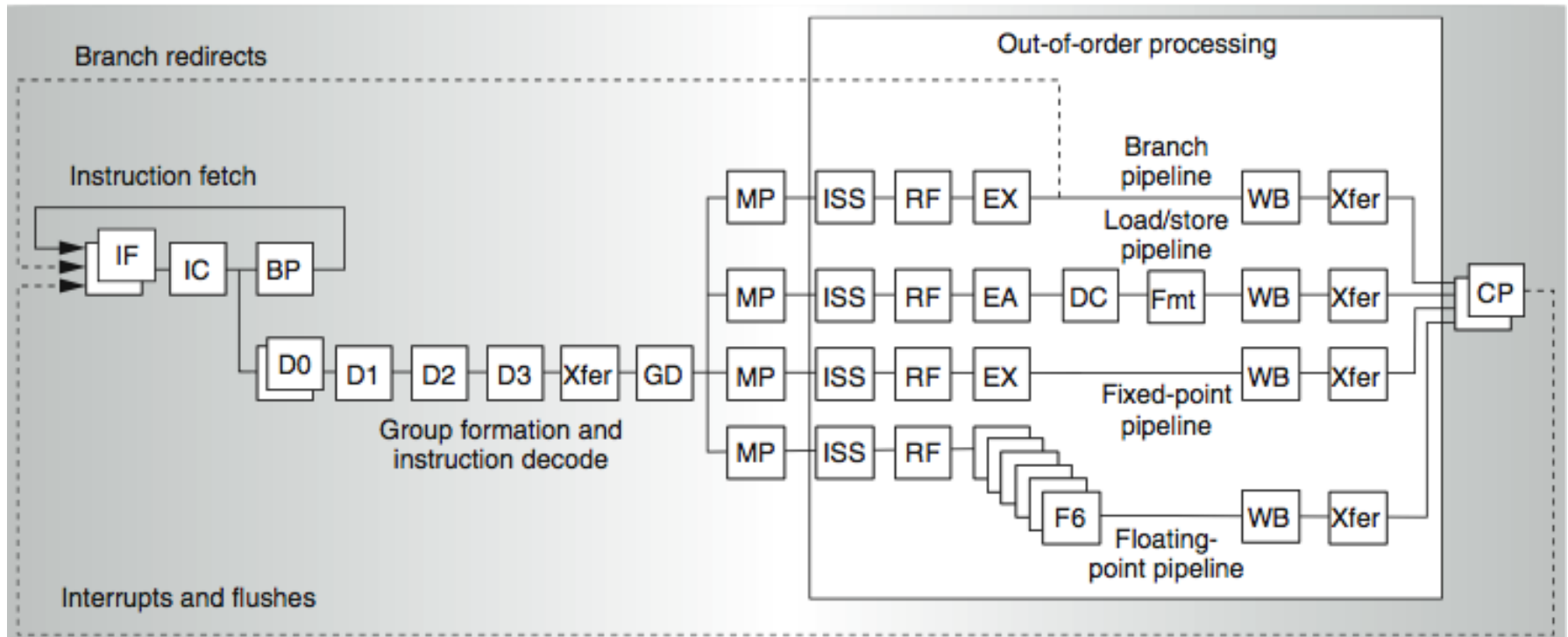
# Power 5 datapath



- στο **IF στάδιο** η πρόσβαση εναλλάσσεται κύκλο-ανά-κύκλο ανάμεσα στα 2 threads
  - 2 instruction fetch address registers, 1 για κάθε νήμα
- μπορούν να φορτωθούν 8 instructions σε κάθε κύκλο (**στάδιο IC**) από την I-Cache
  - σε έναν συγκεκριμένο κύκλο, οι εντολές που φορτώνονται προέρχονται όλες από το ίδιο thread
  - και τοποθετούνται στο instruction buffer του thread αυτού (**στάδιο D0**)

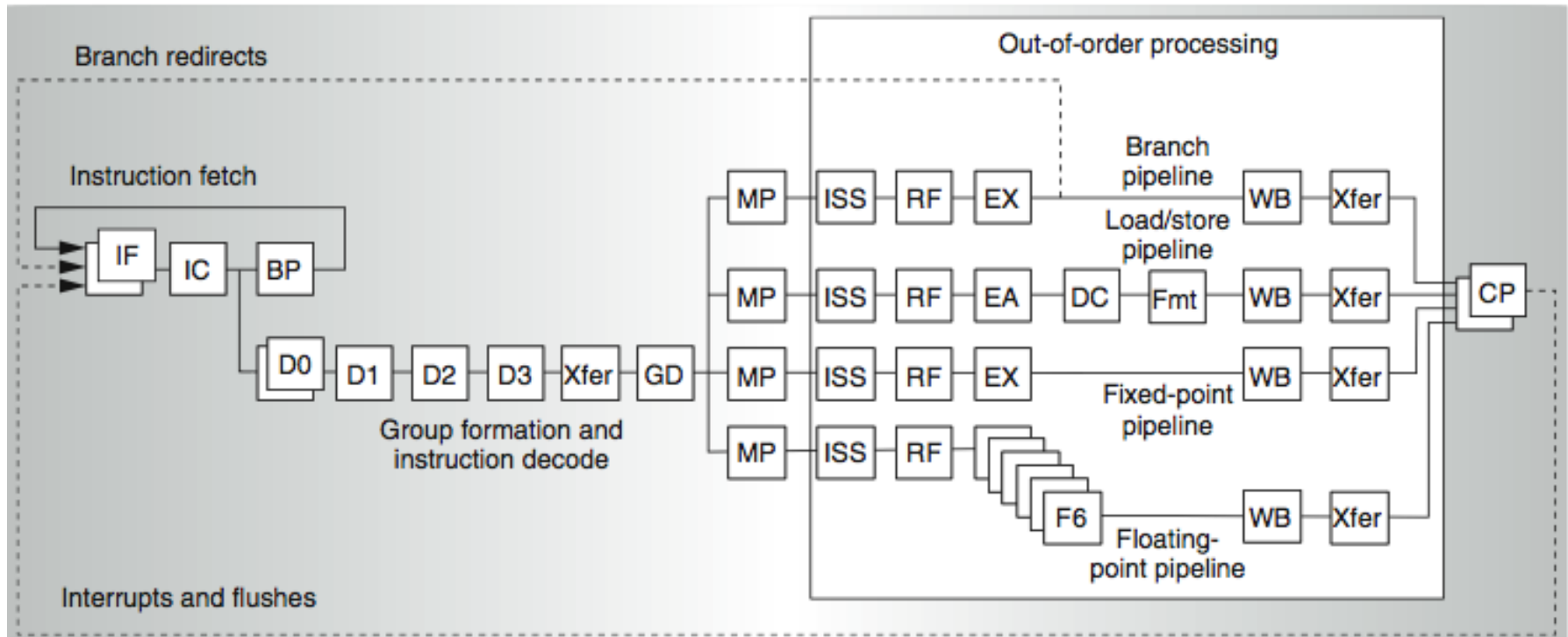


# Power 5 datapath



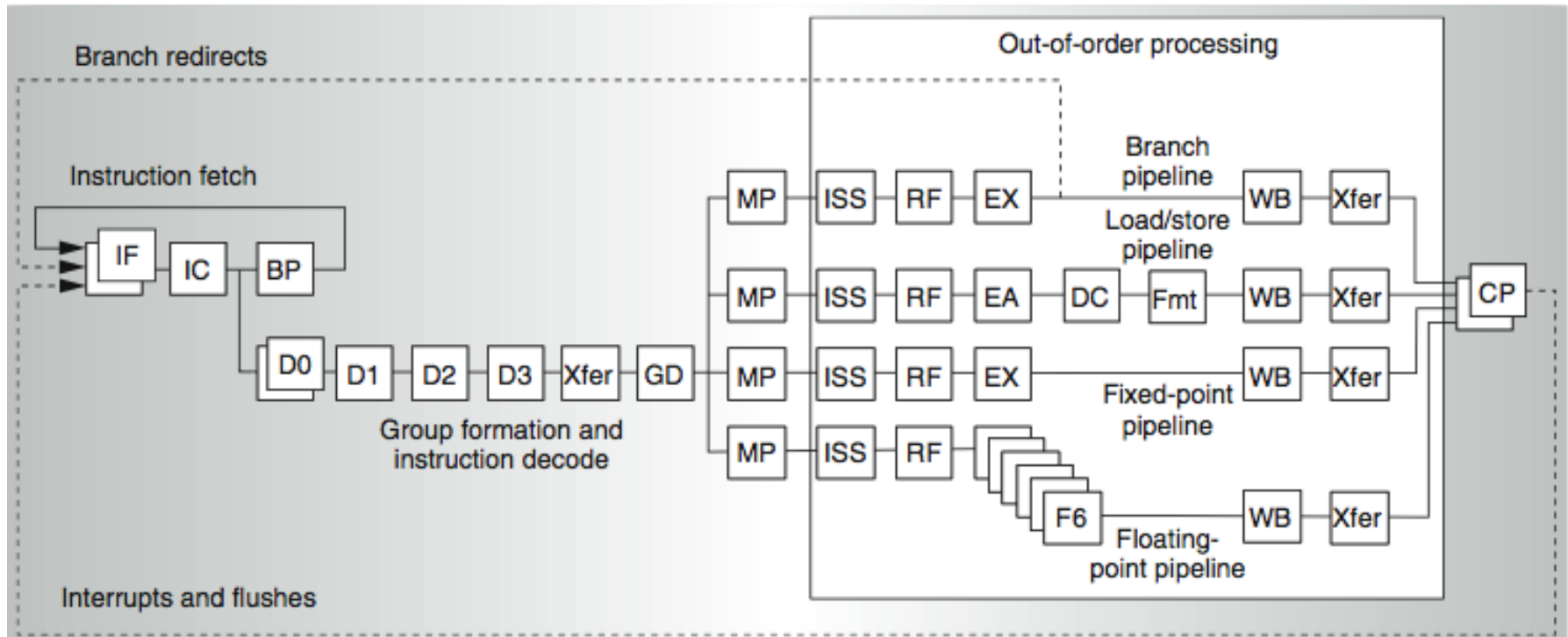
- στα **στάδια D1-D3**, ανάλογα με την προτεραιότητα κάθε thread, ο επεξεργαστής διαλέγει εντολές από έναν από τους δύο instruction buffers και σχηματίζει ένα group
- όλες οι εντολές σε ένα group προέρχονται από το ίδιο thread και αποκωδικοποιούνται παράλληλα
- κάθε group μπορεί να περιέχει το πολύ 5 εντολές

# Power 5 datapath



- όταν όλοι οι απαιτούμενοι πόροι γίνονται διαθέσιμοι για τις εντολές ενός group, τότε το group μπορεί να γίνει dispatch (στάδιο GD)
  - το group τοποθετείται στο Global Completion Table (ROB)
  - τα entries στο GCT εκχωρούνται με τη σειρά προγράμματος για κάθε thread, και απελευθερώνονται (πάλι με τη σειρά προγράμματος) μόλις το group γίνει commit
- μετά το dispatch, κάθε εντολή του group διέρχεται μέσα από το register renaming στάδιο (MP)
  - 120 physical GPRs, 120 physical FPRs
  - τα 2 νήματα διαμοιράζονται δυναμικά registers

# Power 5 datapath



- Issue, execute, write-back
  - δε γίνεται διάκριση ανάμεσα στα 2 threads
- Group completion (στάδιο CP)
  - 1 group commit ανά κύκλο για κάθε thread
  - στη σειρά προγράμματος του κάθε thread

# Δυναμική εξισορρόπηση επεξεργαστικών πόρων

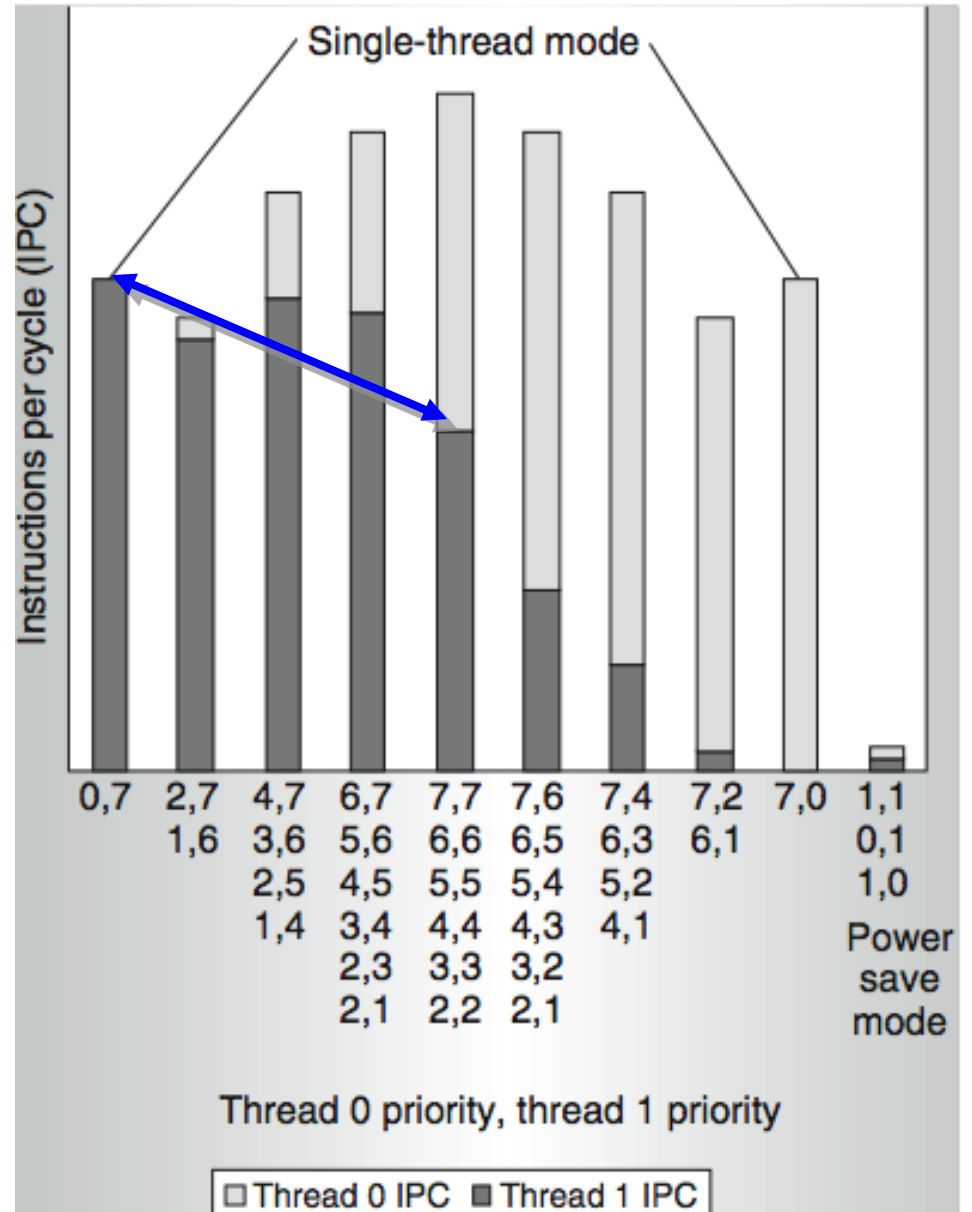
- Πρέπει να εξασφαλιστεί η απρόσκοπτη και ομαλή ροή των threads στο pipeline, ανεξάρτητα από τις απαιτήσεις του καθενός σε επεξεργαστικούς πόρους
- Μηχανισμοί περιορισμού ενός πολύ απαιτητικού thread:
  - μείωση προτεραιότητας του thread, όταν διαπιστώνεται ότι τα **GCT entries** που χρησιμοποιεί ξεπερνούν ένα καθορισμένο όριο
  - όταν ένα thread έχει πολλά **L2 misses**, τότε οι μεταγενέστερες εξαρτώμενες εντολές του μπορούν να γεμίσουν τις issue queues, εμποδίζοντας να γίνουν dispatch εντολές από το άλλο thread
    - » παρακολούθηση της *Load Miss Queue* ενός thread έτσι ώστε όταν τα misses του υπερβαίνουν κάποιο όριο, η αποκωδικοποίηση εντολών του να σταματάει μέχρι να αποσυμφορηθούν οι issue queues
  - συμφόρηση στις issue queues μπορεί να συμβεί και όταν ένα thread εκτελεί μια **εντολή που απαιτεί πολύ χρόνο**
    - » *flushing* των εντολών του thread που περιμένουν να γίνουν dispatch και προσωρινή διακοπή της αποκωδικοποίησης εντολών του μέχρι να αποσυμφορηθούν οι issue queues

# Ρυθμιζόμενη προτεραιότητα threads

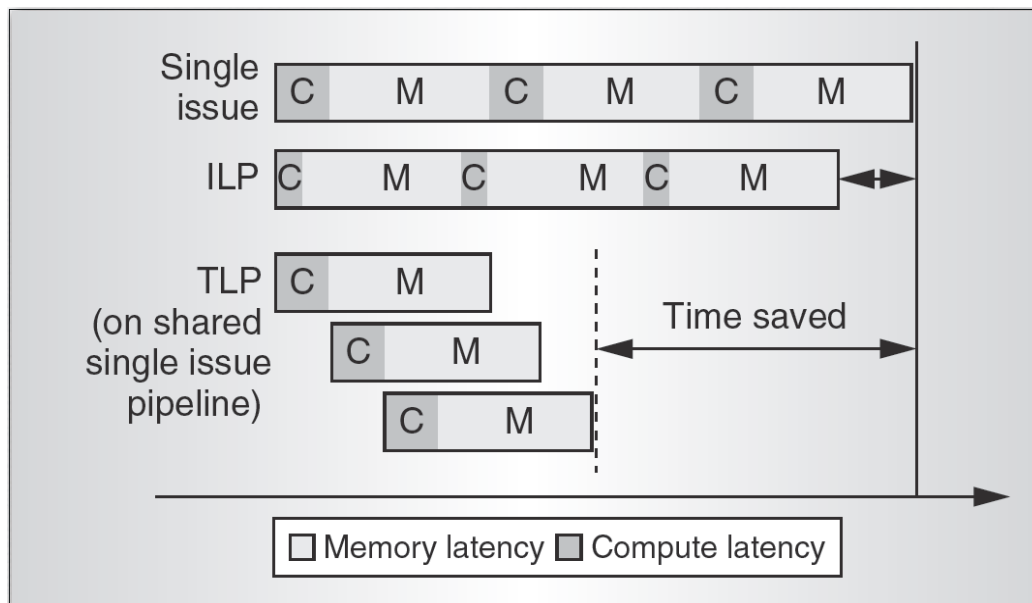
- επιτρέπει στο software να καθορίσει πότε ένα thread θα πρέπει να έχει μεγαλύτερο ή μικρότερο μερίδιο επεξεργαστικών πόρων
- πιθανές αιτίες για διαφορετικές προτεραιότητες:
  - ένα thread εκτελεί ένα spin-loop περιμένοντας να πάρει κάποιο lock → δεν κάνει χρήσιμη δουλειά όσο spin-άρει
  - ένα thread δεν έχει δουλειά να εκτελέσει και περιμένει να του ανατεθεί δουλειά σε ένα idle loop
  - μία εφαρμογή πρέπει να τρέχει πιο γρήγορα σε σχέση με μία άλλη (π.χ. real-time application vs. background application)
- 8 software-controlled επίπεδα προτεραιότητας για κάθε thread
- ο επεξεργαστής παρατηρεί τη διαφορά των επιπέδων προτεραιότητας των threads, και δίνει στο thread με τη μεγαλύτερη προτεραιότητα περισσότερους διαδοχικούς κύκλους για αποκωδικοποίηση εντολών του

# Επίδραση προτεραιότητας στην απόδοση κάθε thread

- Όταν τα threads έχουν ίδιες προτεραιότητες, εκτελούνται πιο αργά από,τι αν κάθε thread είχε διαθέσιμα όλα τα resources του επεξεργαστή (single-thread mode)



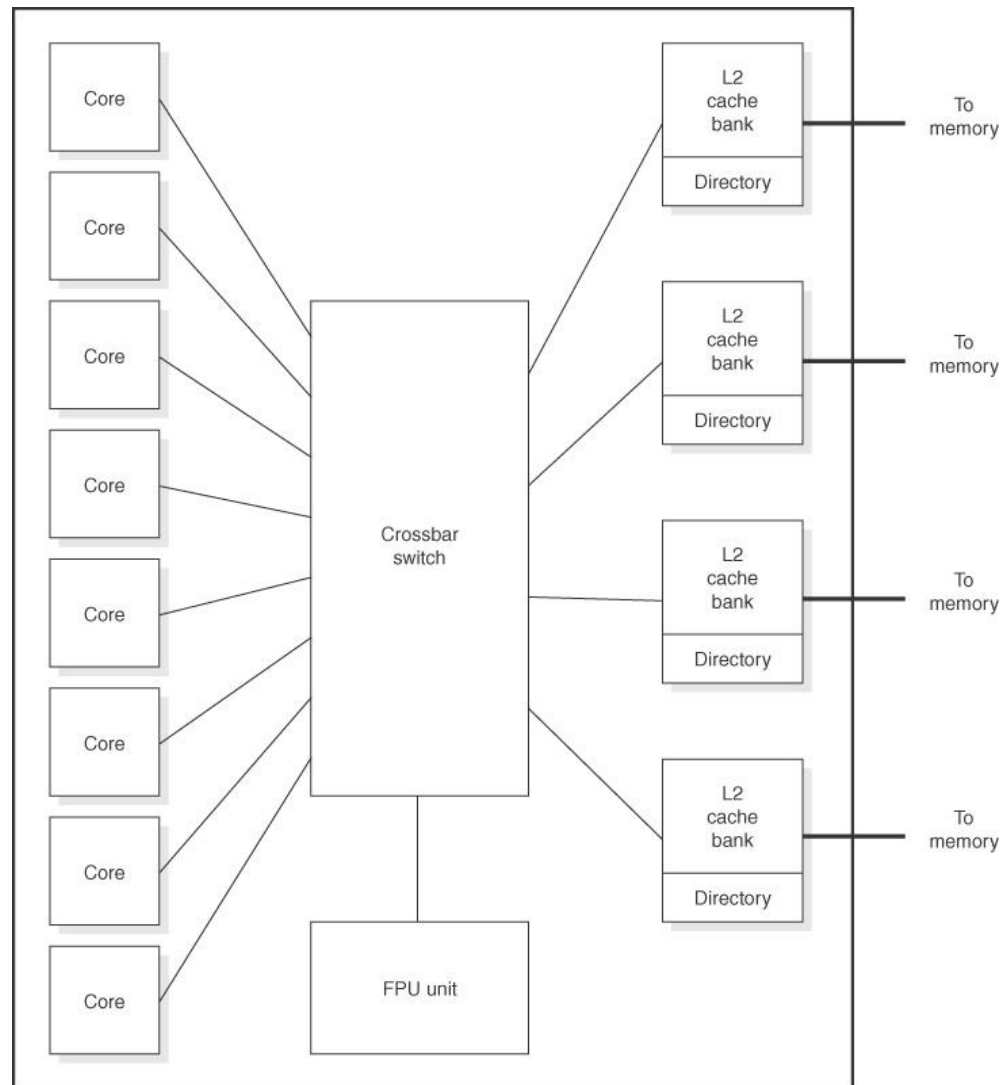
## Case-study 3: UltraSPARC T1 (“Niagara”) (2005)



- Συμπεριφορά επεξεργαστών βελτιστοποιημένων για TLP και ILP σε server workloads:
  - server workloads:
    - » υψηλός TLP (μεγάλος αριθμός παράλληλων client requests)
    - » χαμηλός ILP (υψηλά miss rates, πολλά unpredictable branches, συχνές load-load εξαρτήσεις)
    - » το memory access time κυριαρχεί στο συνολικό χρόνο εκτέλεσης
- ο “ILP” επεξεργαστής μειώνει μόνο το computation time
  - το memory access time κυριαρχεί σε ακόμα μεγαλύτερο ποσοστό
- στον “TLP” επεξεργαστή, το memory access ενός thread επικαλύπτεται από computations από άλλα threads
  - η απόδοση αυξάνεται για μια memory-bound multithreaded εφαρμογή

# UltraSPARC T1

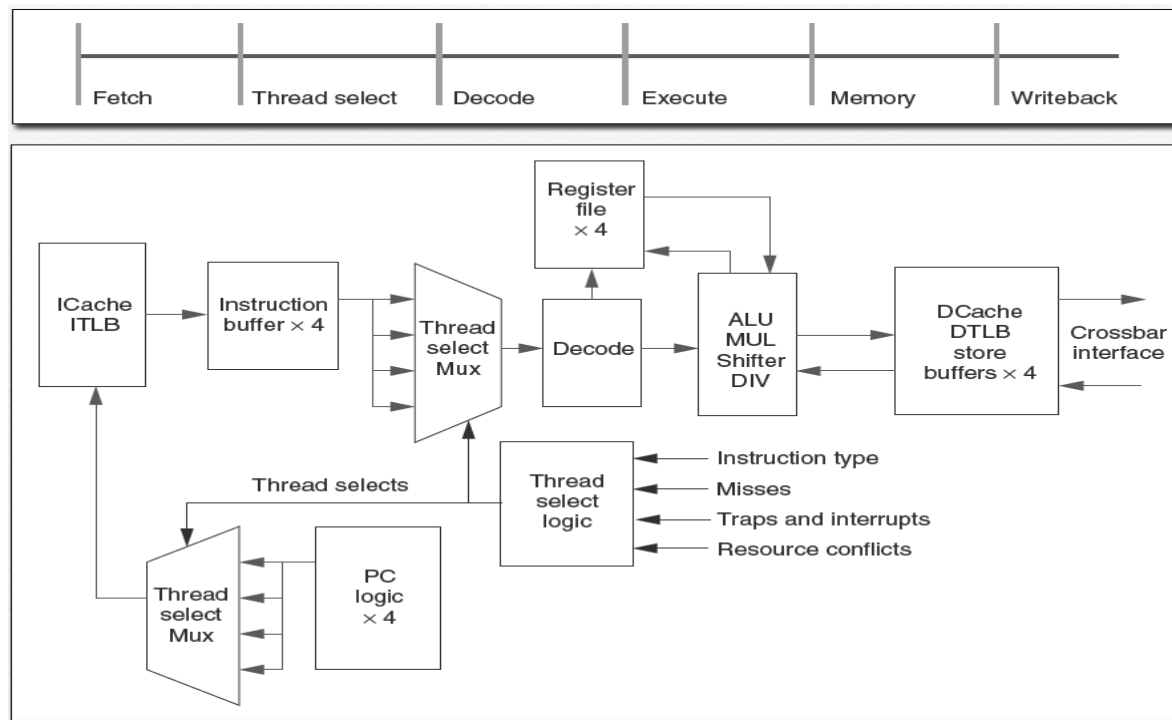
- in-order, single-issue
  - επικεντρώνεται πλήρως στην εκμετάλλευση του TLP
- 4-8 cores, 4 threads ανά core
  - max 32 threads
  - fine-grained multithreading
- L1D + L1I μοιραζόμενες από τα 4 threads
- L2 cache + FPU μοιραζόμενη από όλα τα threads
- ξεχωριστό register set + instruction buffers + store buffers για κάθε thread



© 2007 Elsevier, Inc. All rights reserved.

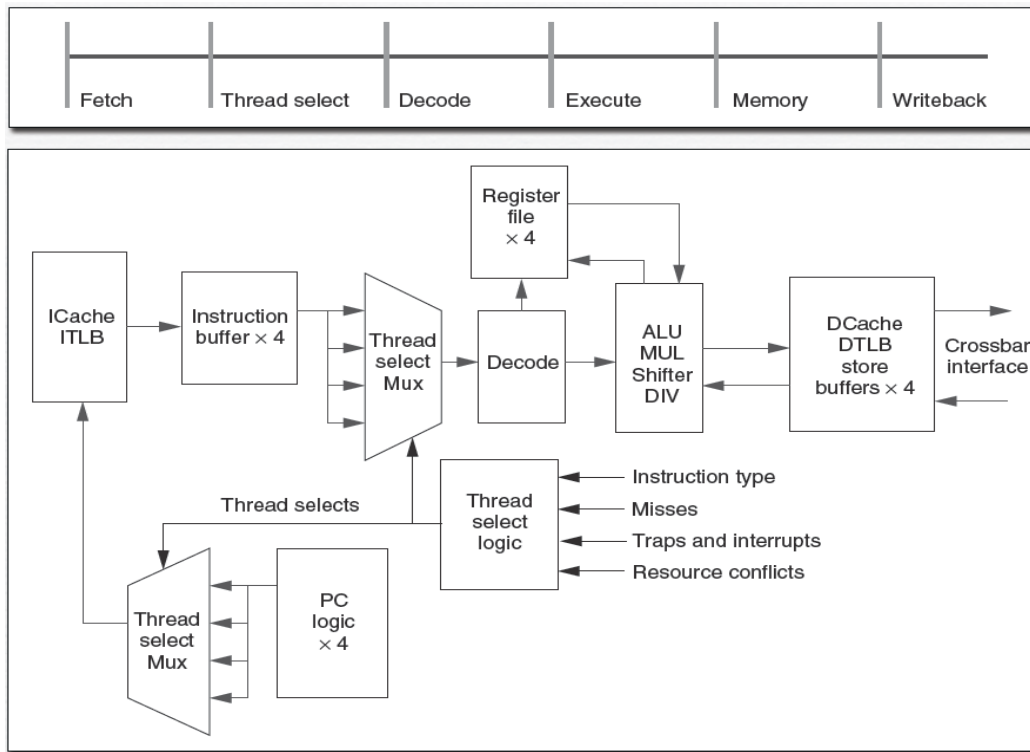


# UltraSPARC T1 pipeline



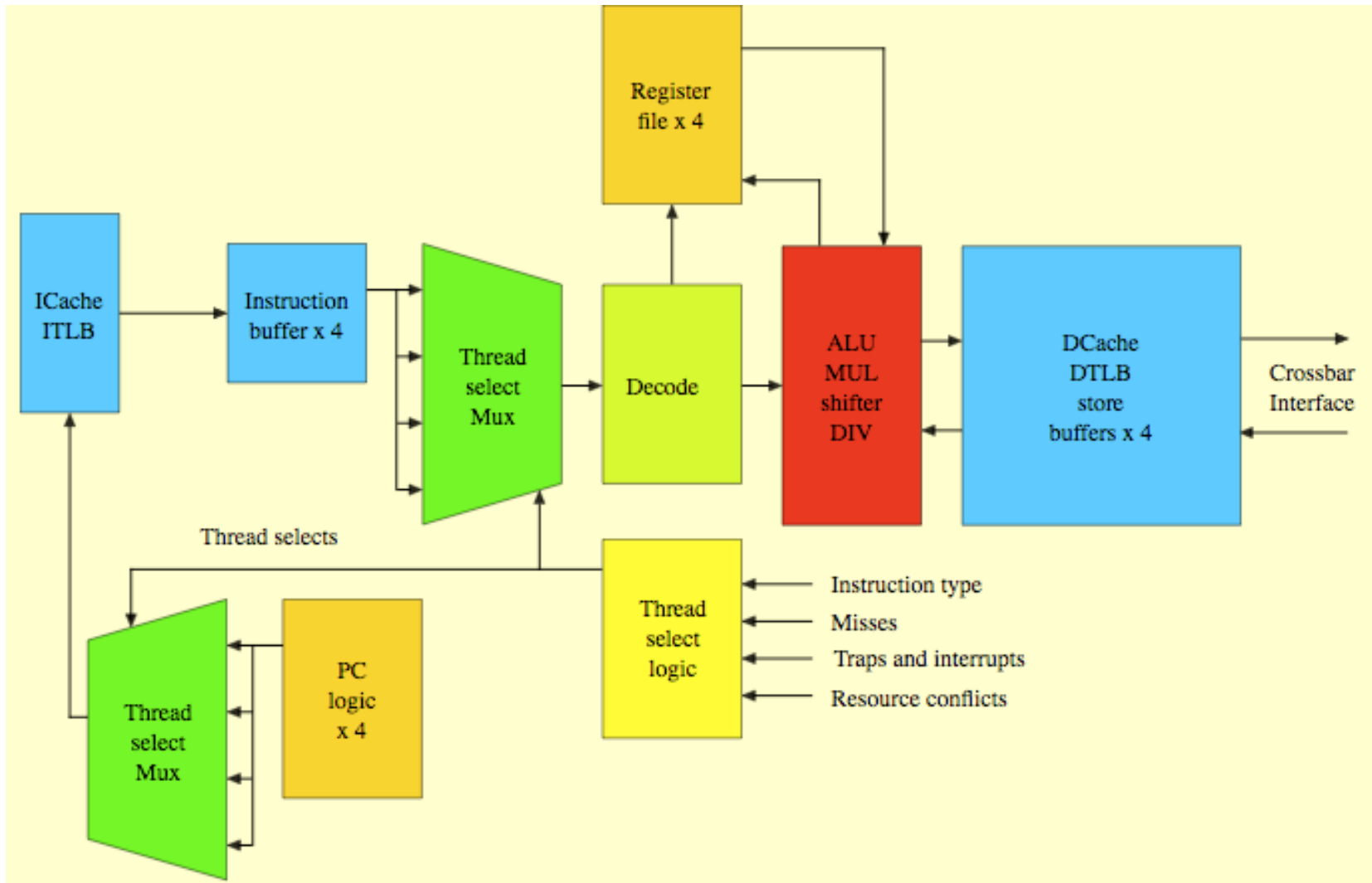
- όπως το κλασικό 5-stage pipeline + thread select stage
- **Fetch stage:** ο thread select mux επιλέγει ποιος από τους 4 PCs θα πρέπει να προσπελάσει την ICache και το ITLB
- **Thread select stage:** αποφασίζει σε κάθε κύκλο ποιος από τους 4 instruction buffers θα τροφοδοτήσει με εντολές τα επόμενα στάδια
  - αν το thread-select στάδιο επιλέξει ένα νήμα από το οποίο θα στείλει εντολές, το fetch στάδιο θα επιλέξει το ίδιο thread για να προσπελάσει την ICache

# UltraSPARC T1 pipeline



- πολιτική επιλογής thread:
  - εναλλαγή μεταξύ διαθέσιμων threads σε κάθε κύκλο
  - προτεραιότητα στο least recently used thread (round-robin)
- λόγοι μη διαθεσιμότητας (και μη επιλογής) ενός thread
  - long-latency εντολές (π.χ. branches, mult/div) οδηγούν στη μη-επιλογή του αντίστοιχου thread για όσους κύκλους διαρκούν
  - stalls λόγω cache misses
  - stalls λόγω structural hazards για μια non-pipelined δομή που χρησιμοποιείται ήδη από κάποιο άλλο thread (π.χ. divider)

# UltraSPARC T1 pipeline

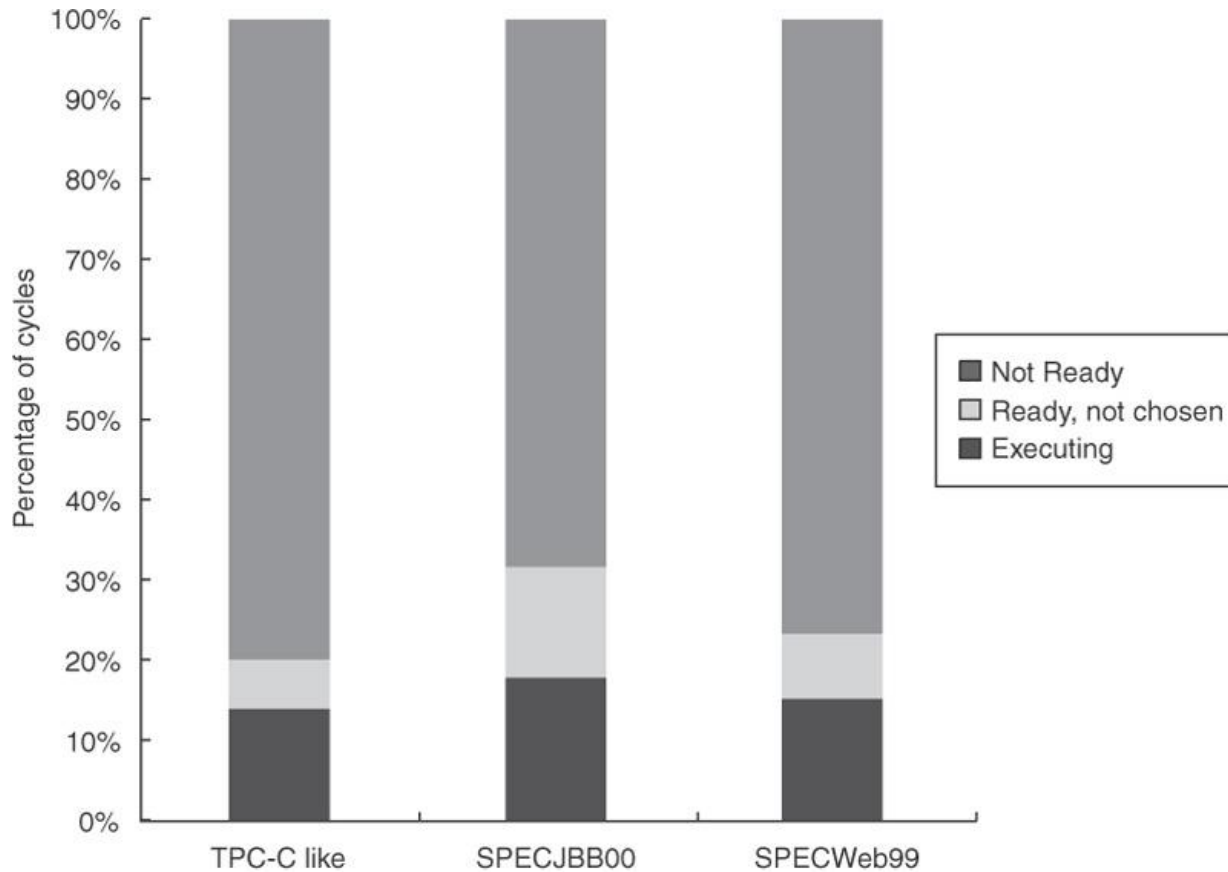


# UltraSPARC T1 performance

Benchmark	Per-thread CPI	Per core CPI	Effective CPI for eight cores	Effective IPC for eight cores
TPC-C	7.2	1.8	0.225	4.4
SPECJBB	5.6	1.40	0.175	5.7
SPECWeb99	6.6	1.65	0.206	4.8

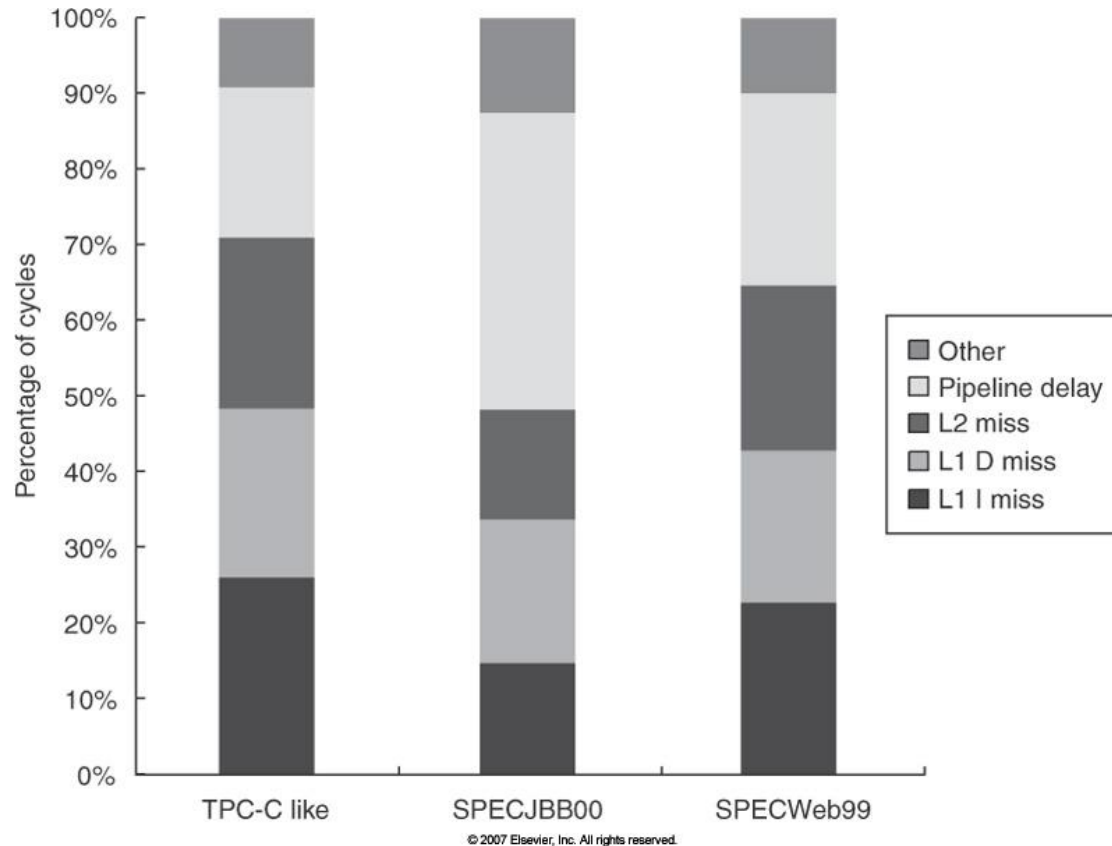
- fine-grained multithreading μεταξύ 4 threads → ιδανικό per-thread CPI = 4
- ιδανικό per-core CPI = 1
- effective CPI = per-core CPI / #cores
- effective throughput: μεταξύ 56% και 71% του ιδανικού

# Προφίλ εκτέλεσης ενός μέσου thread



© 2007 Elsevier, Inc. All rights reserved.

# Λόγοι για τη μη διαθεσιμότητα ενός thread



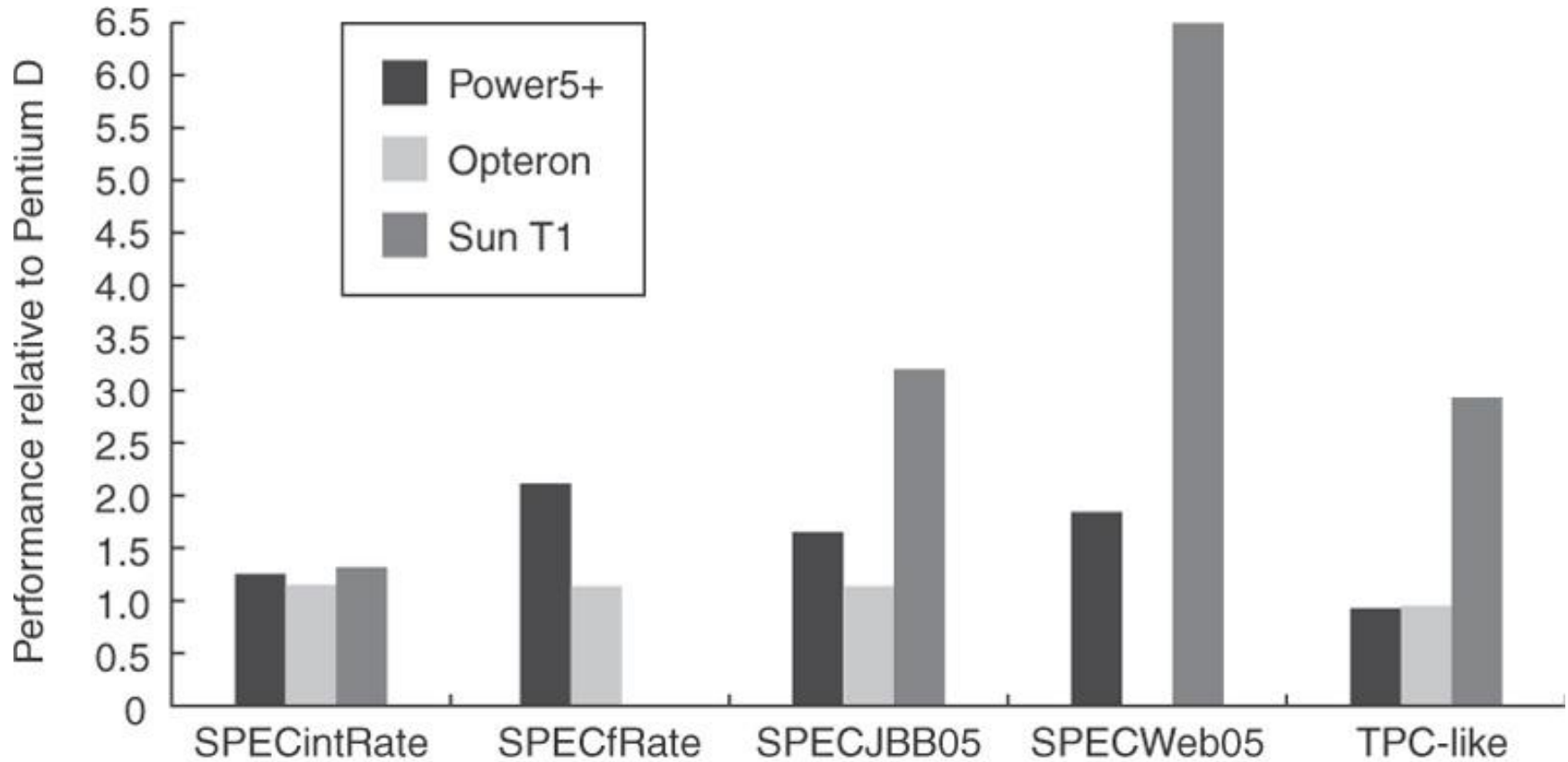
- pipeline delay: long-latency εντολές όπως branches, loads, fp, int mult/div

# «Crash-test» multicore επεξεργαστών (~2005)

Characteristic	SUNT1	AMD Opteron	Intel Pentium D	IBM Power5
Cores	8	2	2	2
Instruction issues per clock per core	1	3	3	4
Multithreading	Fine-grained	No	SMT	SMT
Caches L1 I/D in KB per core L2 per core/shared L3 (off-chip)	16/8 3 MB shared	64/64 1 MB/core	12K uops/16 1 MB/core	64/32 L2: 1.9 MB shared L3: 36 MB
Peak memory bandwidth (DDR2 DRAMs)	34.4 GB/sec	8.6 GB/sec	4.3 GB/sec	17.2 GB/sec
Peak MIPS FLOPS	9600 1200	7200 4800 (w. SSE)	9600 6400 (w. SSE)	7600 7600
Clock rate (GHz)	1.2	2.4	3.2	1.9
Transistor count (M)	300	233	230	276
Die size (mm <sup>2</sup> )	379	199	206	389
Power (W)	79	110	130	125

- βασικές διαφορές:
  - εκμετάλλευση ILP vs. TLP (Power5 → Opteron, Pentium D → T1)
  - floating point performance (Power5 → Opteron, Pentium D → T1)
  - memory bandwidth (T1 → Power5 → Opteron → Pentium D)
    - » επηρεάζει την απόδοση εφαρμογών με μεγάλο miss rate

# «Crash-test» multicore επεξεργαστών



© 2007 Elsevier, Inc. All rights reserved.

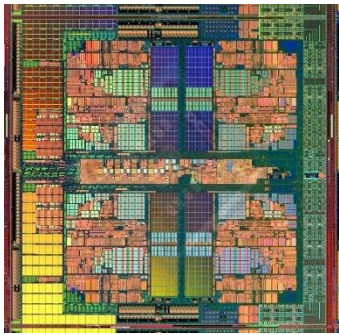


# «Crash-test» multicore επεξεργαστών (~2010)

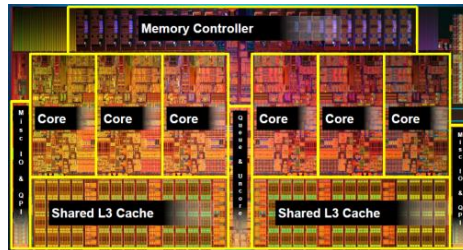
	AMD Opteron 8439	IBM Power 7	Intel Xenon 7560	Sun T2
Transistors (M)	904	1200	2300	500
Power (W)	137	140	130	95
Max cores/chip	6	8	8	8
Multithreading	No	SMT	SMT	Fine-grained
Threads/ core	1	4	2	8
Instr. issue/clock	3 from 1 thread	6 from 1 thread	4 from 1 thread	2 from 2 threads
Clock rate (GHz)	2.8	4.1	2.7	1.6
Outermost cache	L3, 6MB, shared	L3, 32MB, shared or private/core	L3, 24MB shared	L2, 4MB, shared
Inclusion	No	Yes	Yes	Yes
Coherence protocol	MOESI	Extended MESI	MESIF	MOESI
Coherence implementation	Snooping	L3 Directory	L3 Directory	L2 Directory
Extended coherence support	Up to 8 processor chips (NUMA)	Up to 32 processor chips (UMA)	Up to 8 processor cores	Up to 2/4 chips (directly/external ASICs)

# Today: Many Cores on Chip

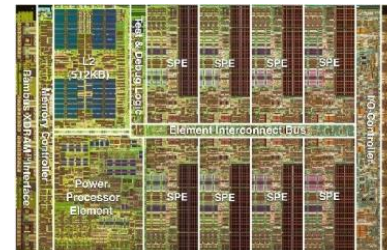
- Simpler and lower power than a single large core
- Large scale parallelism on chip



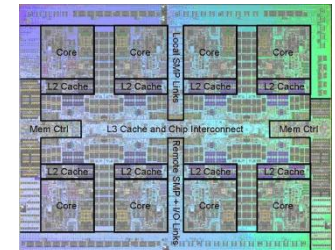
**AMD Barcelona**  
4 cores



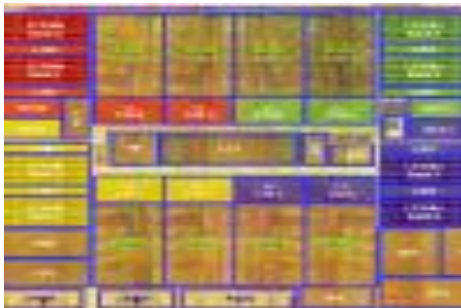
**Intel Core i7**  
8 cores



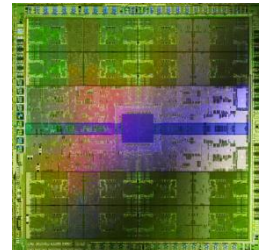
**IBM Cell BE**  
8+1 cores



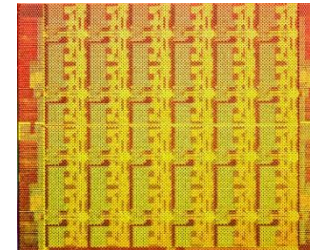
**IBM POWER7**  
8 cores



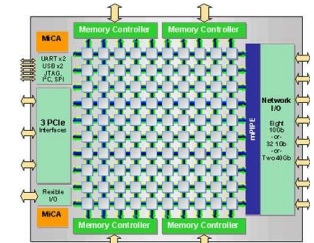
**Sun Niagara II**  
8 cores



**Nvidia Fermi**  
448 "cores"



**Intel SCC**  
48 cores, networked



**Tiler TILE Gx**  
100 cores, networked

# Chips Today (2010-2014)

- Intel Nehalem (~2010)
  - slides by Krste Asanovic (Berkeley, CS152 - [link](#))
- HotChips 2012 (HC24 <http://www.hotchips.org/archives/hc24/> )
  - Intel's 3<sup>rd</sup> generation processors Ivy Bridge ([link](#))
  - AMD's Jaguar next generation low power x86 core ([link](#))
  - Knight's Corner - Intel's MIC ([link](#))
  - Power 7+ ([link](#))
- HotChips 2013 (HC25 <http://www.hotchips.org/archives/hc25/> )
  - Power 8 ([link](#))
  - Intel's 4<sup>th</sup> generation processors Haswell ([link](#))
- HotChips 2014 (HC26 <http://www.hotchips.org/archives/hc26/> )
  - AMD's Kaveri APU ([link](#))
  - AMD's Opteron A1100 ([link](#))
  - Next generation SPARC Processor Cache Hierarchy ([link](#))
  - Intel C2000 Atom Microserver ([link](#))
  - NVIDIA's Denver Processor ([link](#))
  - MIT Scorpio ([link](#))
  - Powering the IoT ([link](#))