Προηγμένη Αρχιτεκτονική Υπολογιστών

ILP Processor Complexity
Thread- & Data-Level Parallelism

Νεκτάριος Κοζύρης & Διονύσης Πνευματικάτος
{nkoziris,pnevmati}@cslab.ece.ntua.gr

8ο εξάμηνο ΣΗΜΜΥ – Ακαδημαϊκό Έτος: 2019-20

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

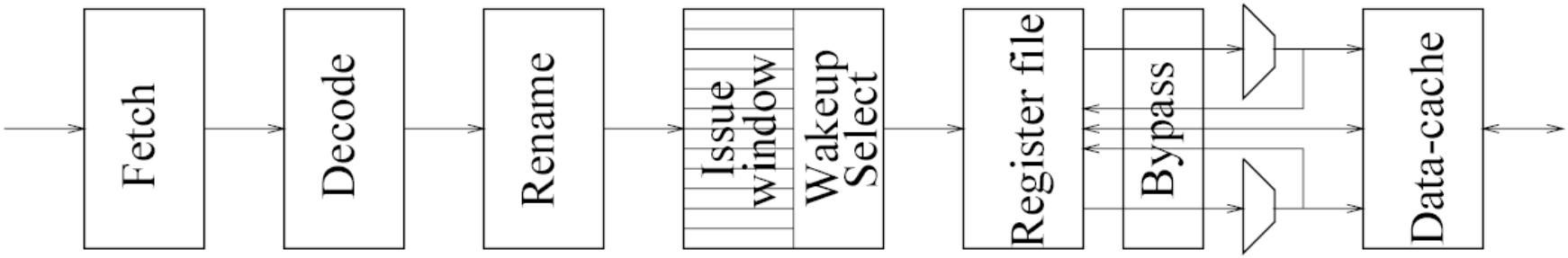
Μέρος πρώτο

Κυκλωματική Πολυπλοκότητα Επεξεργστών

Παραδείγματα σύγχρονων pipeline

Κυκλωματική Πολυπλοκότητα

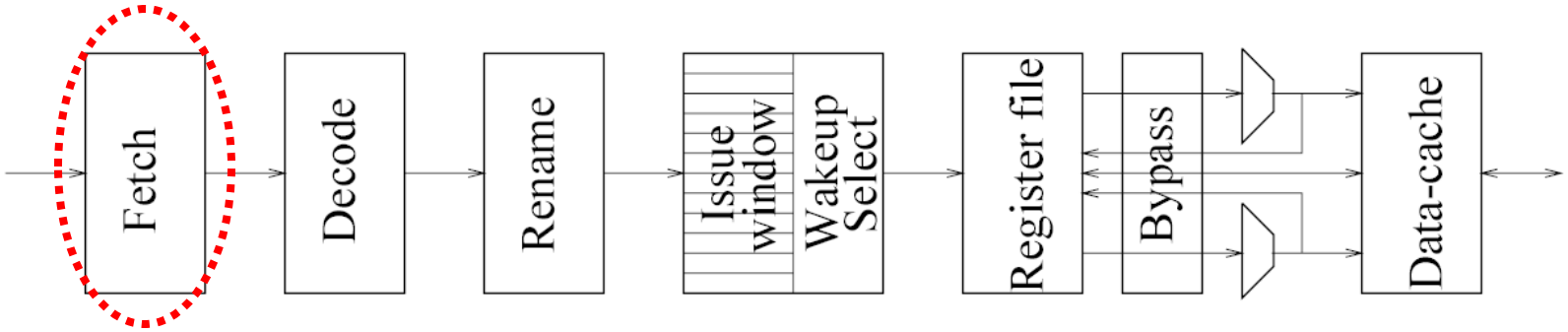
- **Πού βρίσκεται η (κυκλωματική) πολυπλοκότητα σε επεξεργαστές ILP;**
  - Το «ψάξιμο» είναι ακριβό:
    - CDB broadcast -> όλα τα RS + όλους τους καταχωρητές στην RF
    - Συγκριση με tag => καθυστέρηση
    - Μεγάλο “instruction window” => επιλογή «έτοιμων» εντολών ακριβή!



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Based on: Complexity-Effective Superscalar Processors
  - S. Palacharla, N. Jouppi and J. E. Smith, ISCA 97

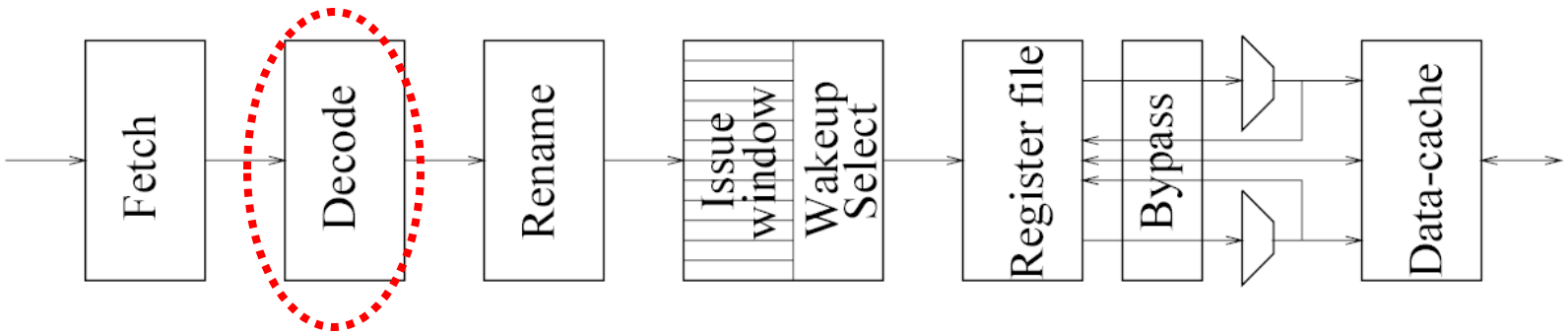
Fetch Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Fetch:
  - Read instructions from I-Cache
  - Predict Branches
  - Pass on to Decode phase

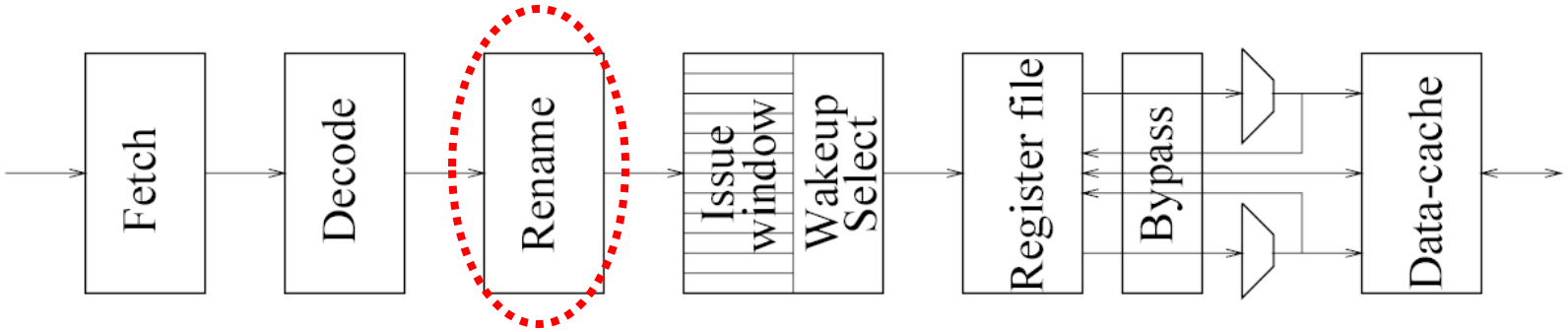
Decode Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Decode:
  - Parse instruction
  - Shuffle opcode parts to appropriate ports for rename

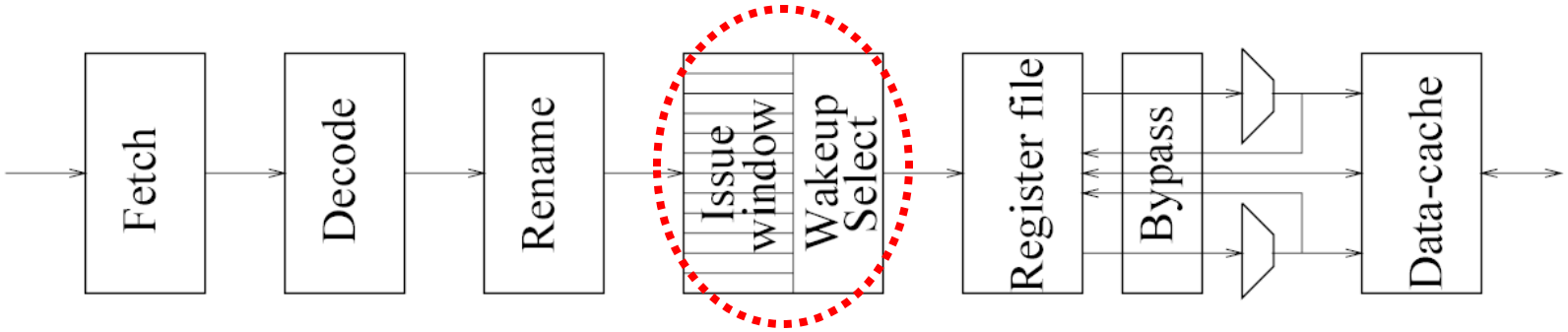
Renaming Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Rename:
  - Map Architectural registers to Physical
  - Eliminate False Dependences
  - Passes renamed instructions to **scheduler**
    - Called **Dispatch**

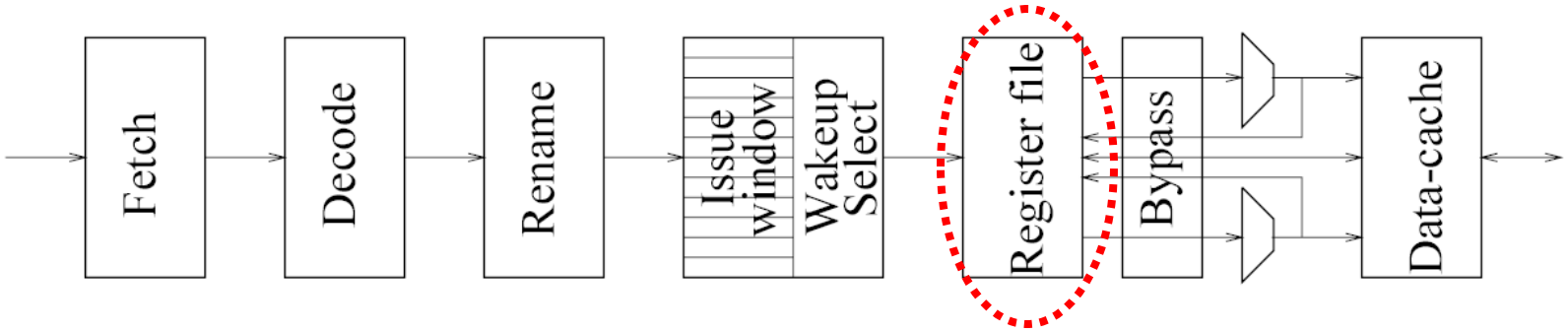
Scheduling Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Wakeup:
  - Instructions check whether they become ready
  - From Writeback: physical register names
- Select:
  - Amongst the **ready** select those to execute
  - **Structural hazards**

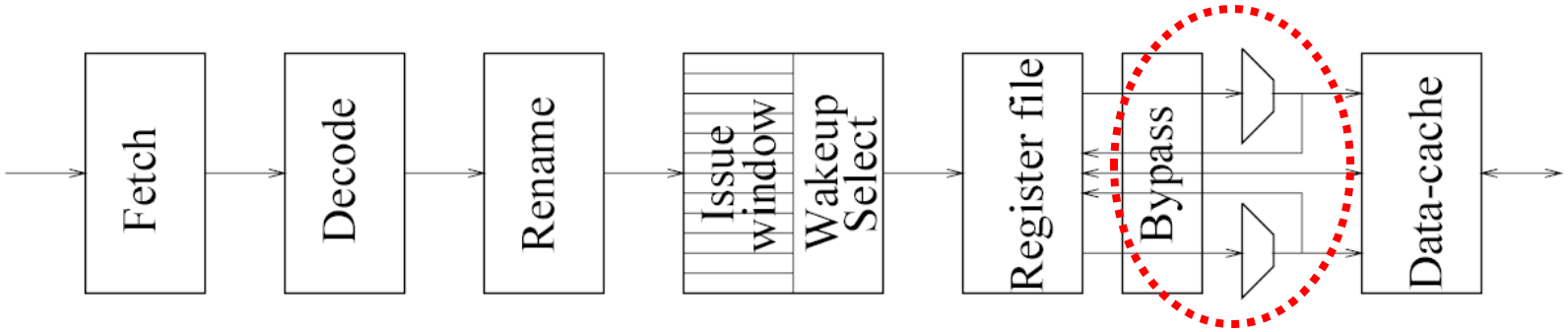
Register File Read Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

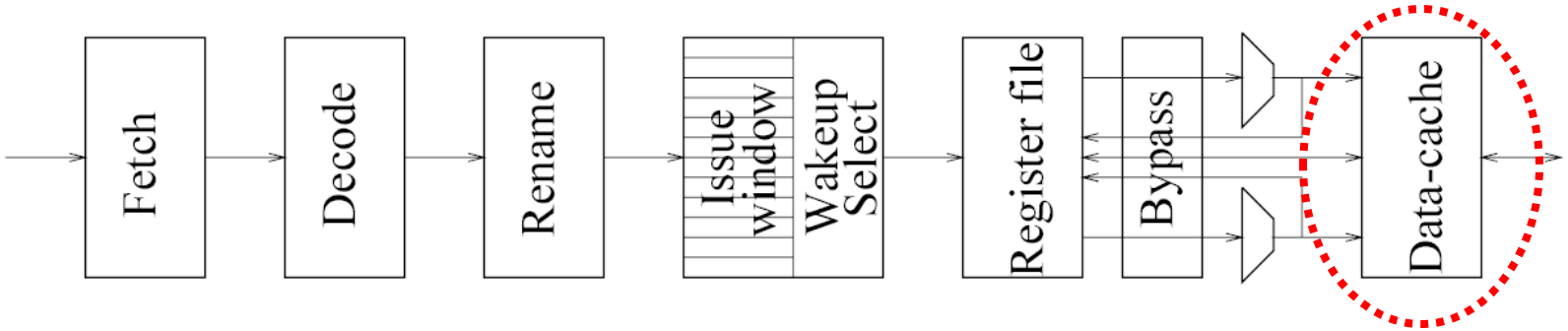
- Read source operands

Bypass and Execute Phase



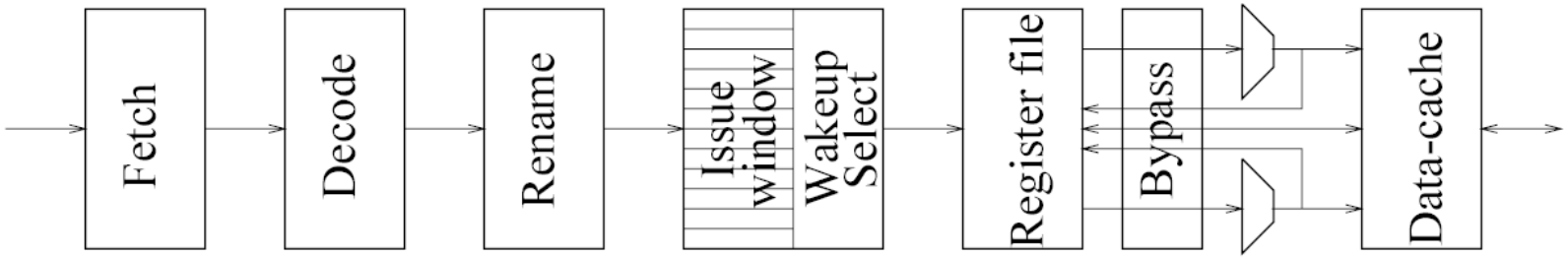
FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

Data Cache Access Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

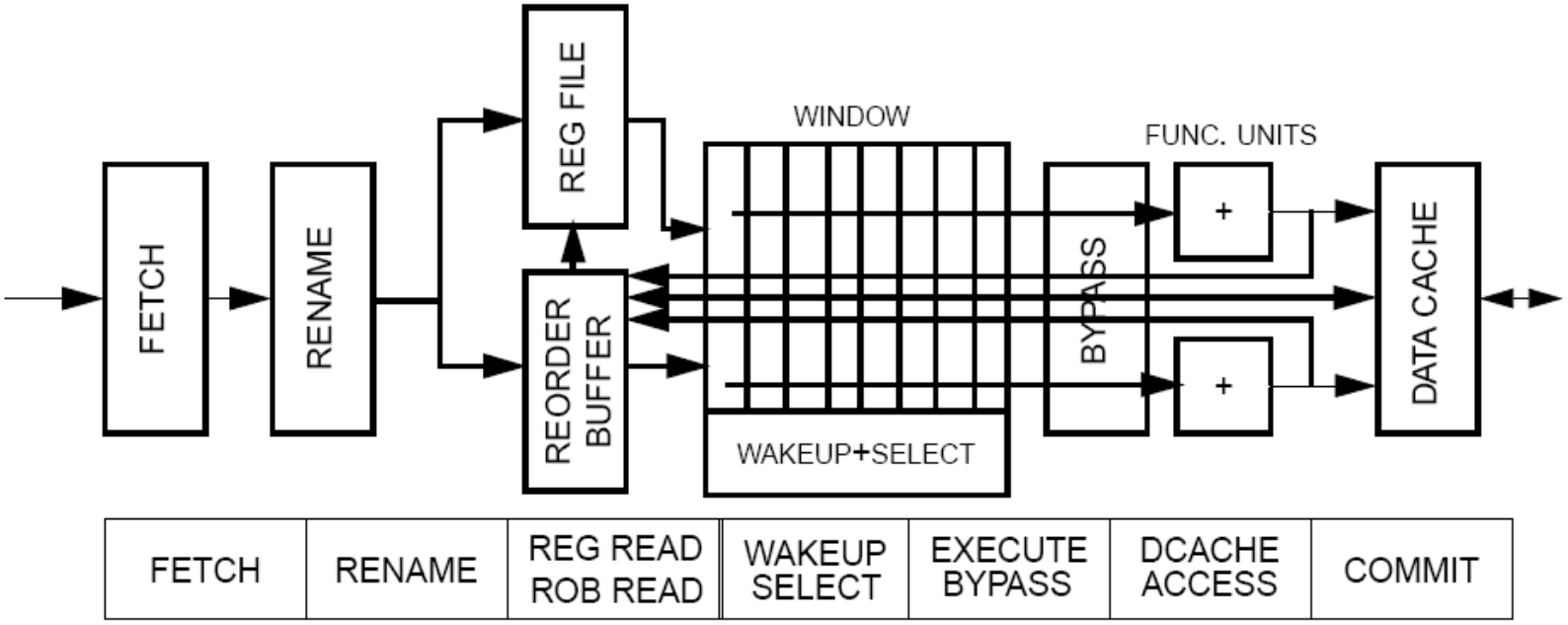
Writeback Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Write result to register file
- Broadcast tag in order to wakeup waiting instructions
  - Notice that the tag broadcast should happen ***TWO cycles*** in ***advance*** of the result production

Reservation Station Model



- Used by Pentium Pro, PowerPC 604
- Re-order buffer holds values
- Renaming points to re-order buffer entries
  - Tomasulo-like

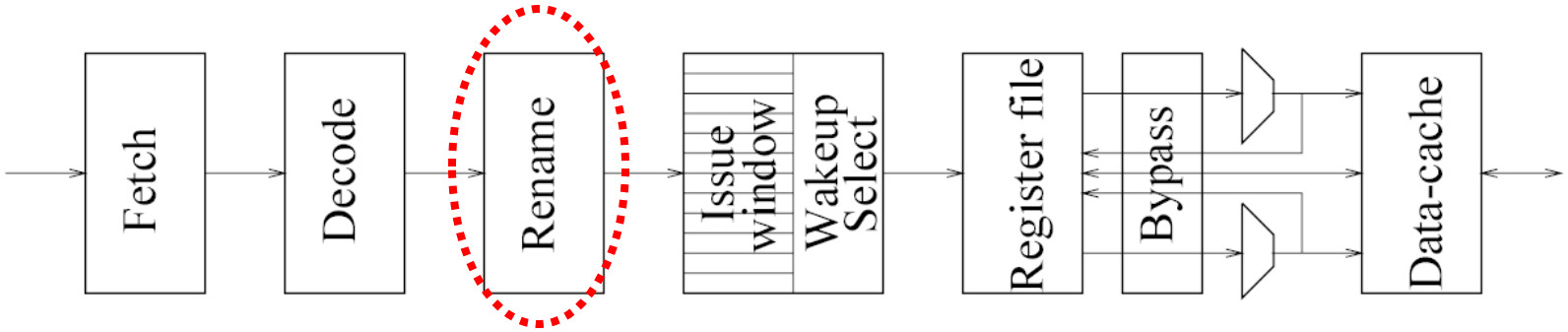
Physical Register File vs. Reservation Station

- Physical Register File
  - Values reside in the register file
  - At writeback instructions broadcast the register name
- Reservation Stations:
  - Values reside:
  - In the register file upon commit
    - Non-speculative
  - In reservation stations prior to commit
    - Speculative

Quantifying Complexity

- Critical Path Delay as a function of architectural parameters
  - **WinSize:** Instruction Window size δηλ. πόσες εντολές έχουν «μπει» στο σύστημα εκτέλεσης και περιμένουν είτε τελεσταίους είτε λειτουργική μονάδα
  - **IW:** Issue Width

Renaming Phase



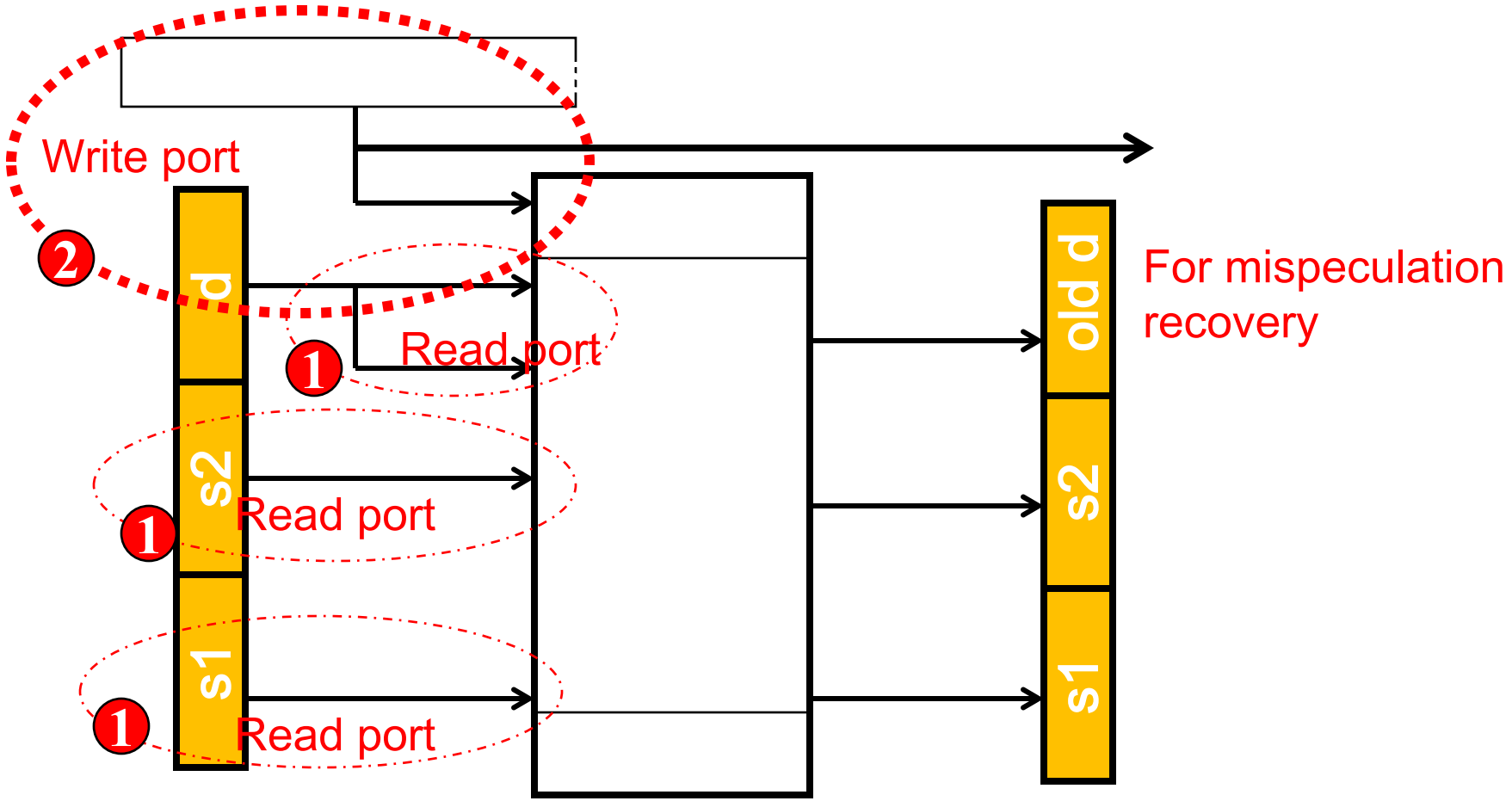
FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Rename:
  - Map Architectural registers to Physical
  - Eliminate False Dependences
  - Passes renamed instructions to **scheduler**
    - Also called **Dispatch**

Renaming

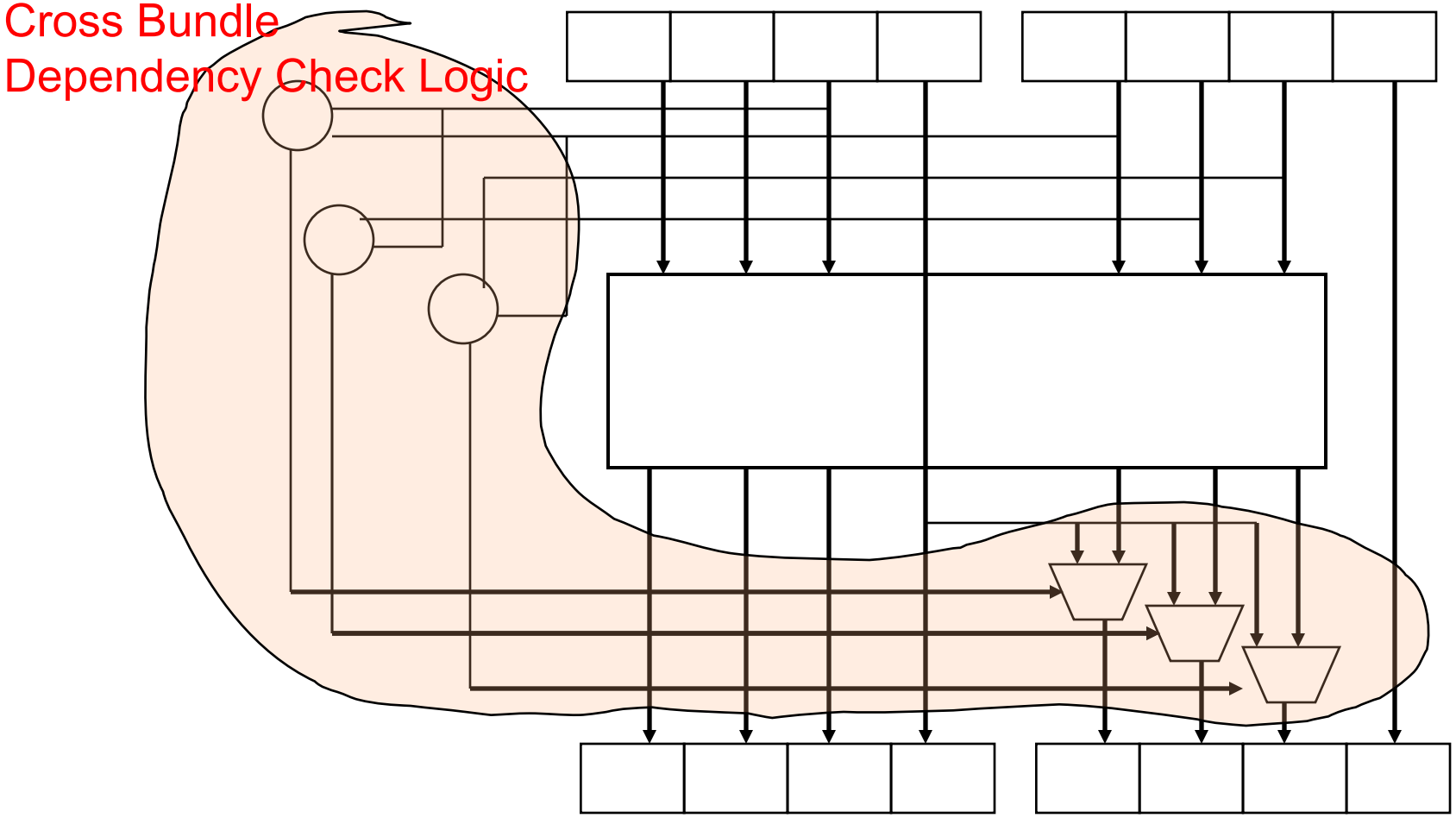
- Inputs:
  - IW instructions (IW = Issue Width)
  - Up to 2 x Input register names
  - Up to 1 x Output register name
- Outputs:
  - 2 x input physical registers
  - 1 x new output physical register
  - 1 x previous physical register name for checkpointing
  - Updated rename table
- Superscalar Issue complicates thing!

Renaming One Instruction



RAT = Register Alias Table

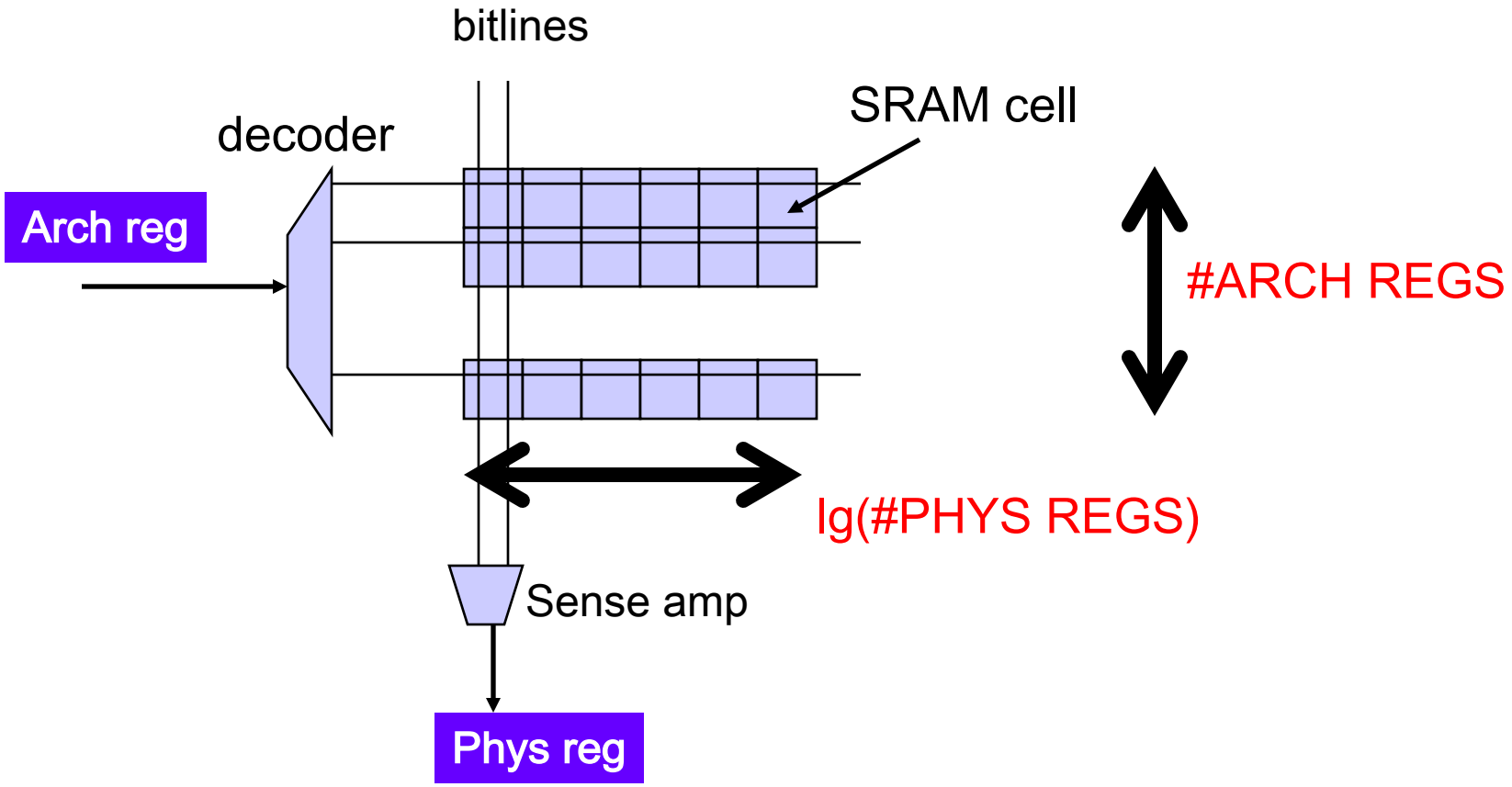
Renaming Two Instructions



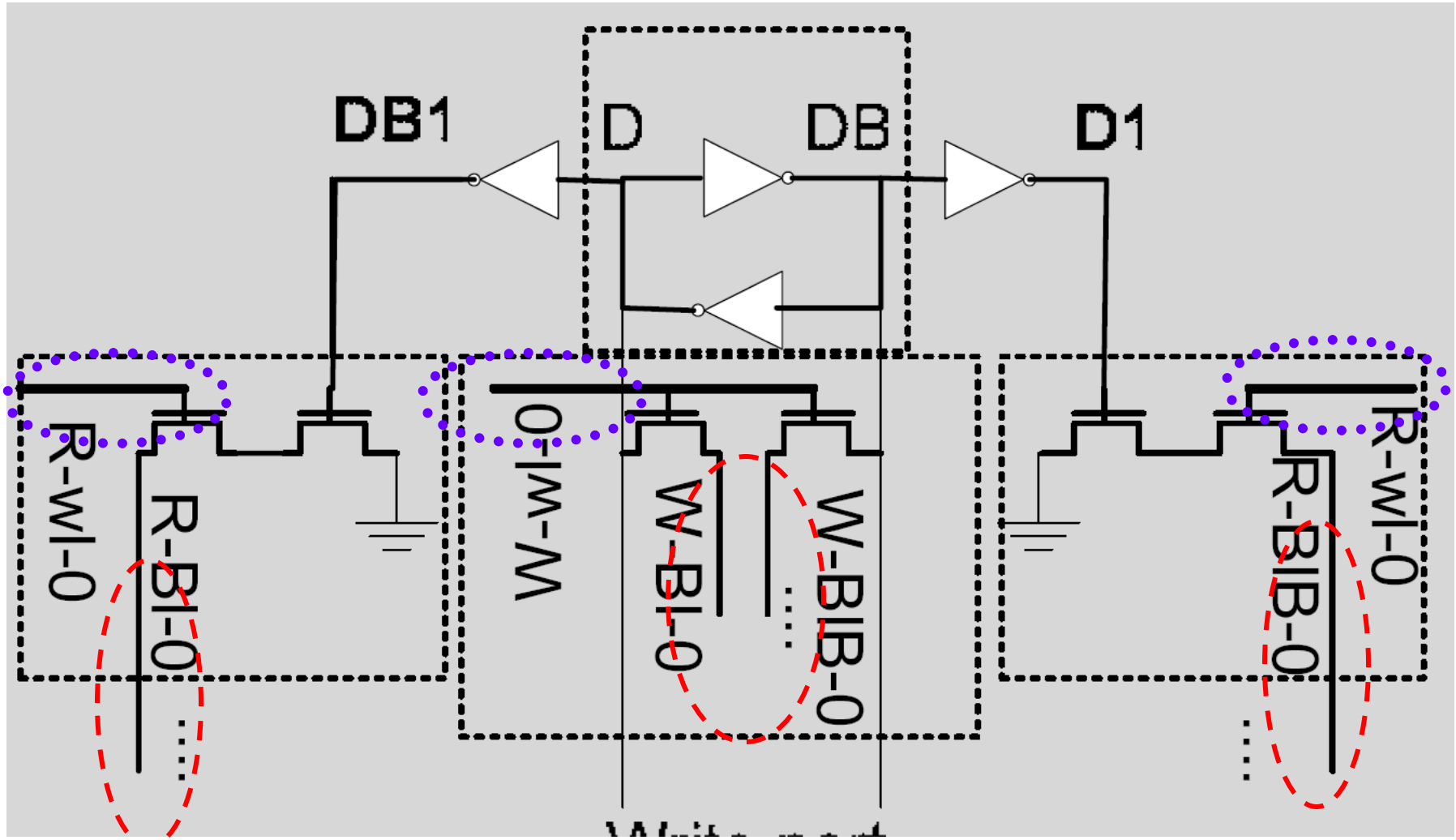
Renaming More Instructions

- Dependency Checking logic for instruction *i* must match against all preceding destinations
- If there are multiple matches it must enforce priority:
  - Pick the one closest to this instruction

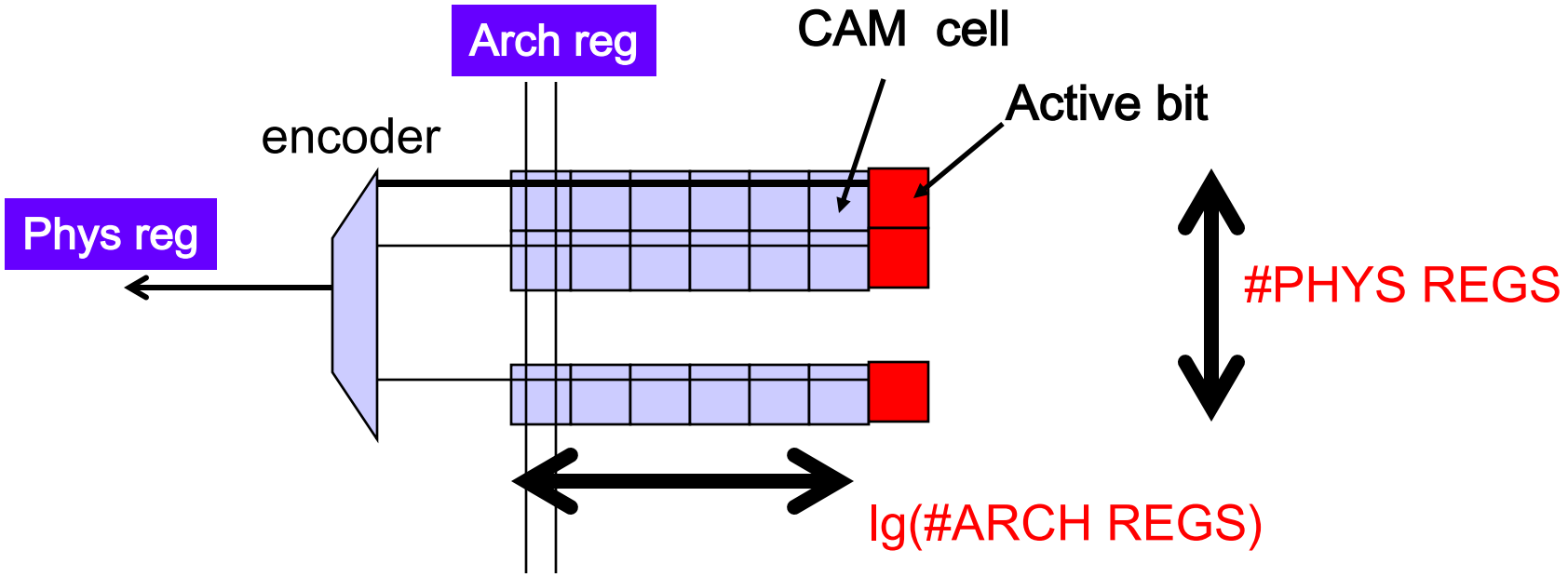
RAT: SRAM Implementation



SRAM RAT cell

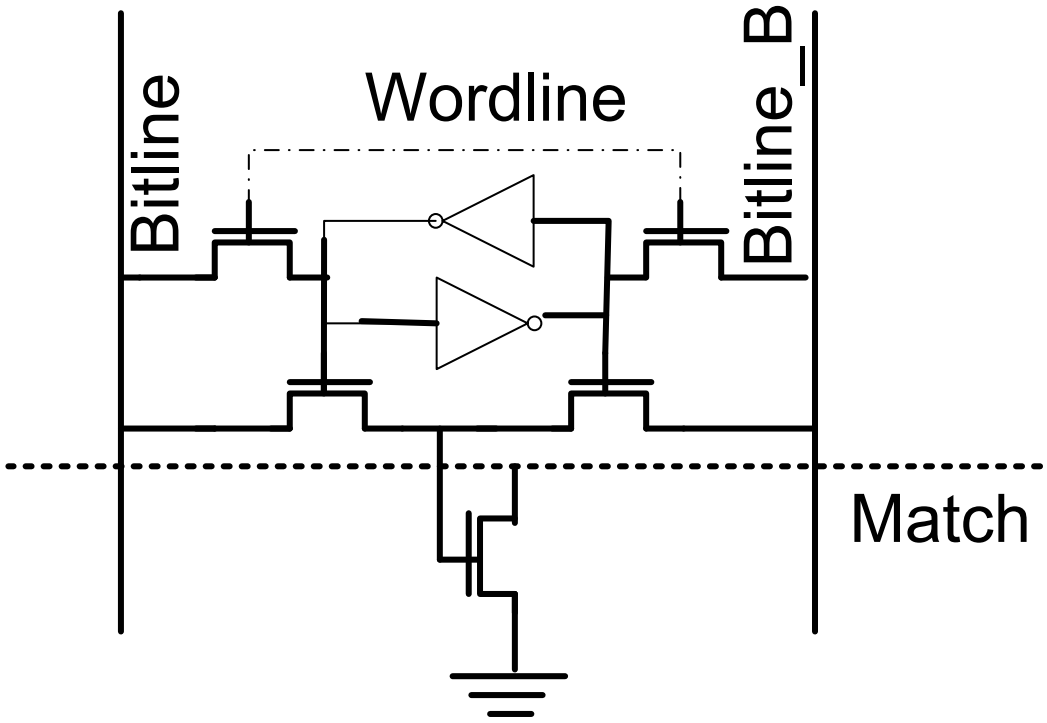


RAT: CAM Implementation



- One CAM per physical register
- Active bit indicates the current map
- New version by setting active bit

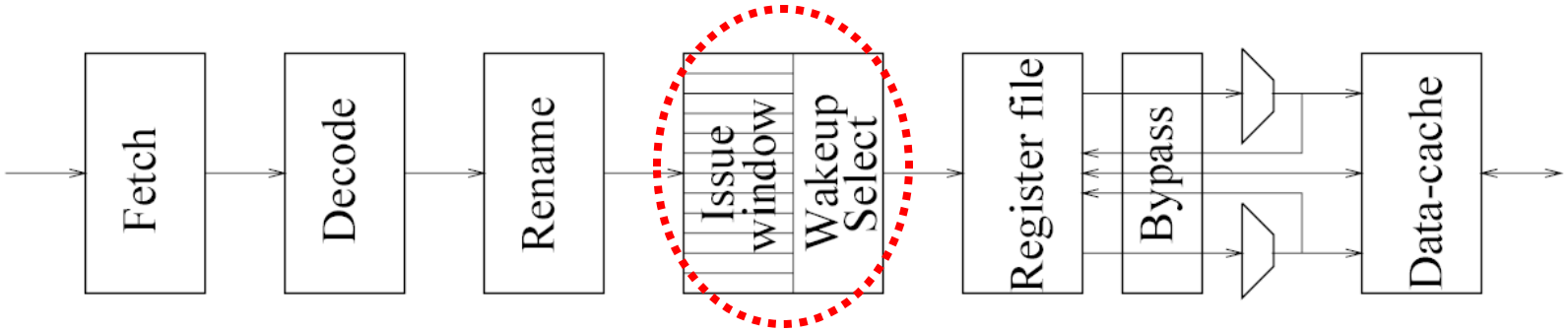
CAM Cell



SRAM vs. CAM

- SRAM:
  - Arch reg rows
  - Lg(phy reg) cols
  - SRAM read/write
- CAM:
  - Phy reg rows
  - Lg(arch reg) cols
  - CAM match
  - Update:
    - Reset previous valid bit
    - Set current valid bit

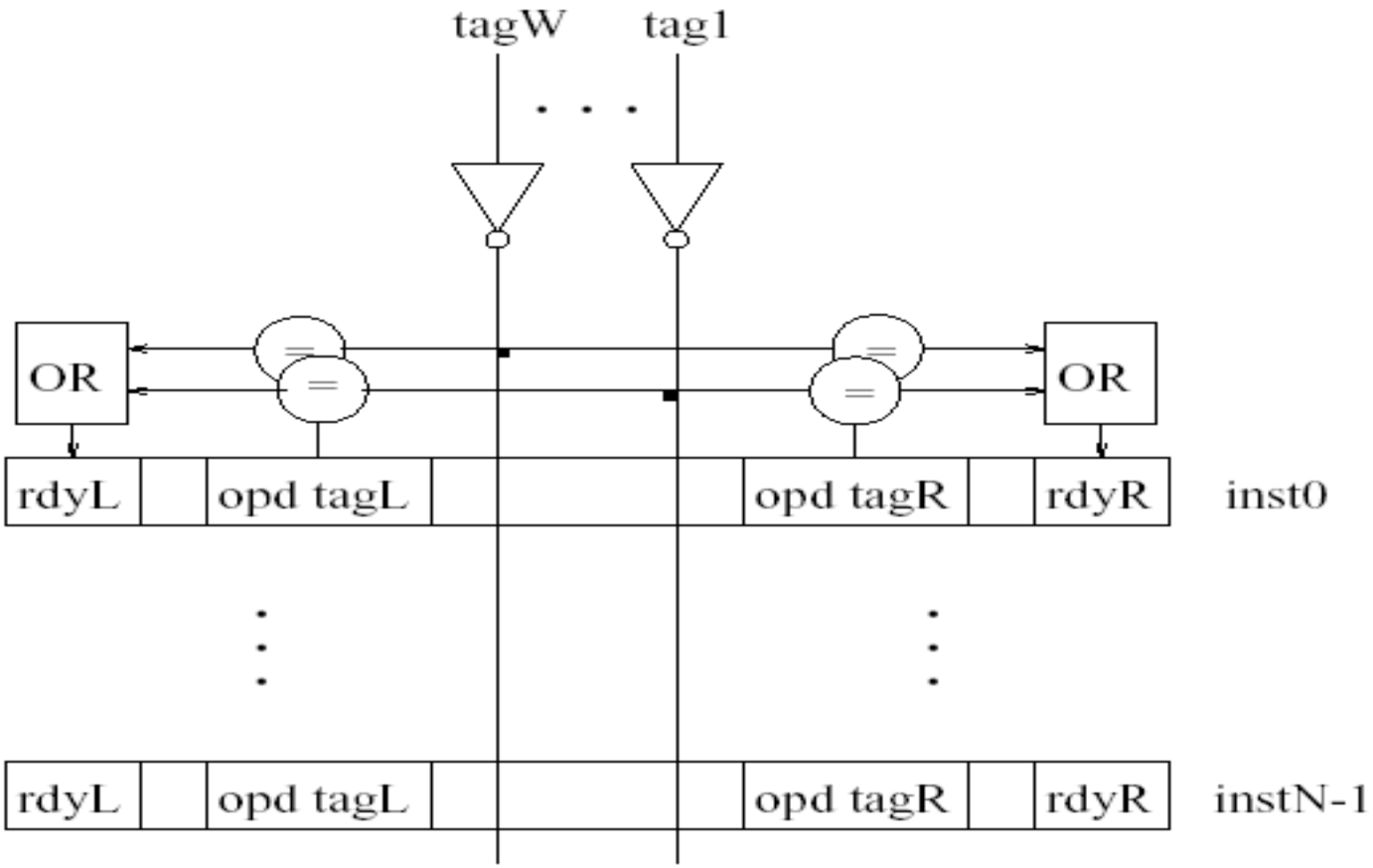
Scheduling Phase



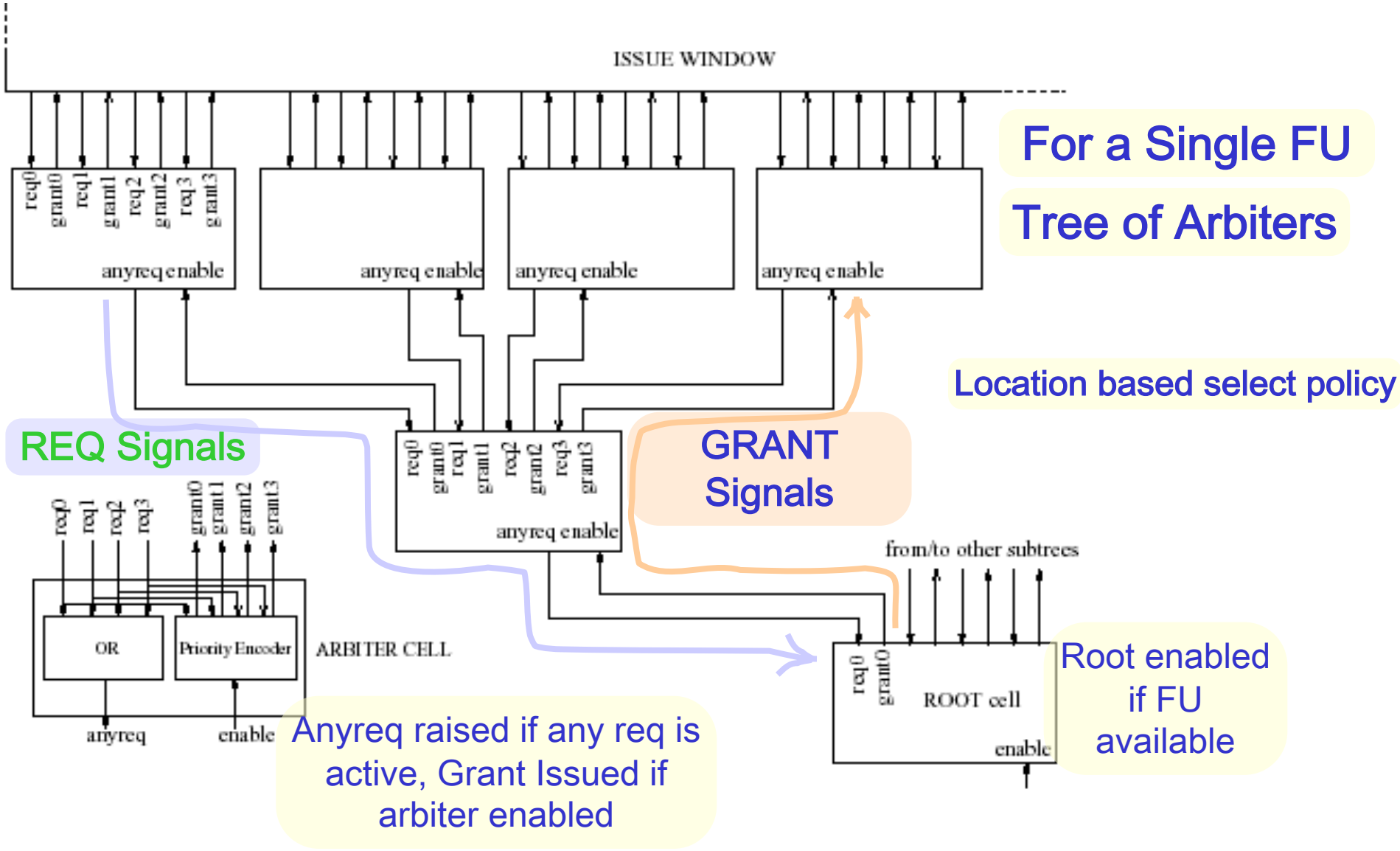
FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

- Wakeup:
  - Instructions check whether they become ready
  - From Writeback: physical register names
- Select:
  - Amongst the **ready** select those to execute
  - **Structural hazards**

Scheduler: Part #1 - Wakeup

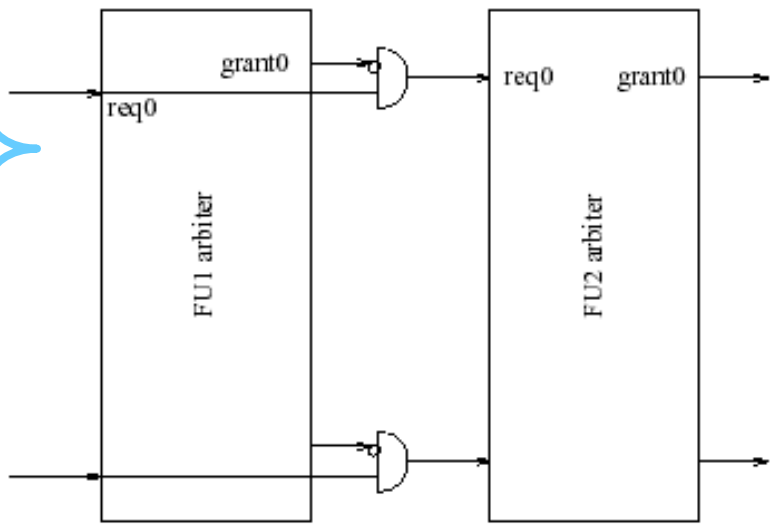


Scheduler: Part #2 - Select



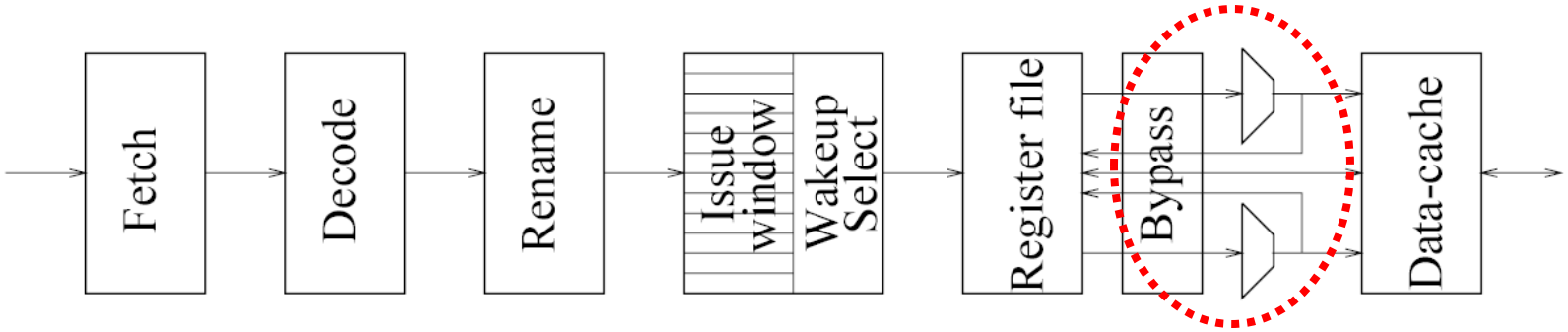
Select for more than one FUs

- Handling Multiple FUs of Same Type:
  - Stack Select logic blocks in series - hierarchy
  - Mask the Request granted to previous unit



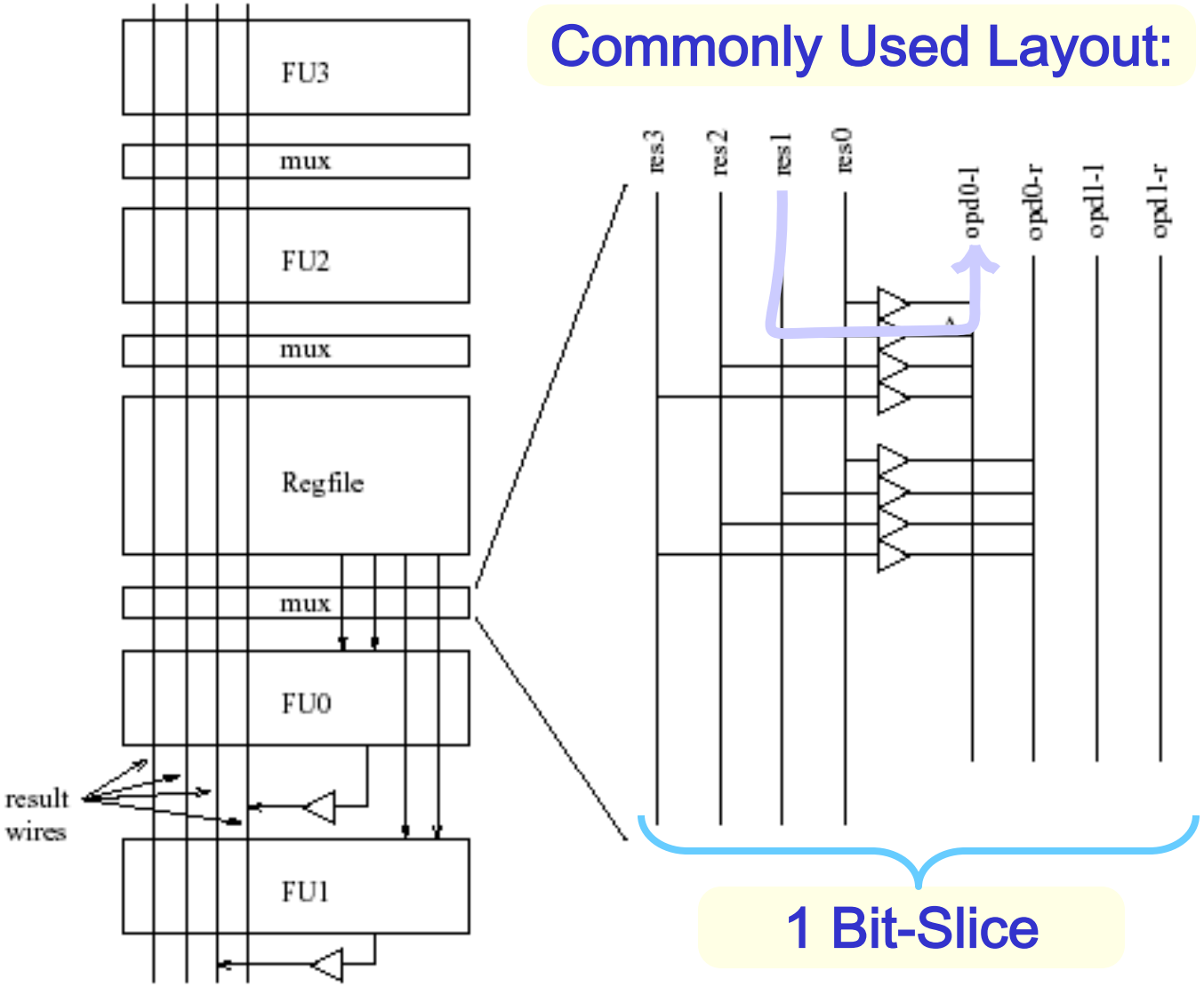
- NOT Feasible for More than 2 FUs
- Alternative:
  - statically partition issue window among FUs – MIPS R10000, HP PA 8000

Bypass and Execute Phase



FETCH	DECODE	RENAME	WAKEUP SELECT	REG READ	EXECUTE BYPASS	DCACHE ACCESS	REG WRITE COMMIT

Datapath and Bypass



Commonly Used Layout:

1 Bit-Slice

Turn on Tri-State A to pass result of FU1 to left operand of FU0

CHARACTERIZING COMPLEXITY

- Summary:

Issue width	Window size	Rename delay (ps)	Wakeup+Select delay (ps)	Bypass delay (ps)
0.8μm technology				
4	32	1577.9	2903.7	184.9
8	64	1710.5	3369.4	1056.4
0.35μm technology				
4	32	627.2	1248.4	184.9
8	64	726.6	1484.8	1056.4
0.18μm technology				
4	32	351.0	578.0	184.9
8	64	427.9	724.0	1056.4

- "Future"
  - 4 Way → Window Logic is bottleneck
  - 8 Way → Bypass Logic is bottleneck

Full Processor Pipeline

- Παραδείγματα από επεξεργαστές Intel x86
- Ιδιαιτερότητα: το σύνολο εντολών *δεν* είναι RISC
  - Πολλές απλές εντολές αλλά και λίγες πολύπλοκες (πρόσβαση μνημης + πράξη, >1 πρόσβαση μνήμης, κ.α.)
  - Οι σύνθετες εντολές απαιτούν πολλαπλά βήματα εκτέλεσης
- Λύση: σπάσιμο πολύπλοκων εντολών σε πολλές απλές
  - Πολύπλοκο front-end => μ-ops
  - Εκτέλεση μ-ops o-o-o, όπως έχουμε δει
  - Pipeline εκτέλεσης δεν αλλάζει, μόνο μεγαλύτερο κόστος διακλαδώσεων!

Intel Sandy Bridge (2nd gen i3/i5/i7)

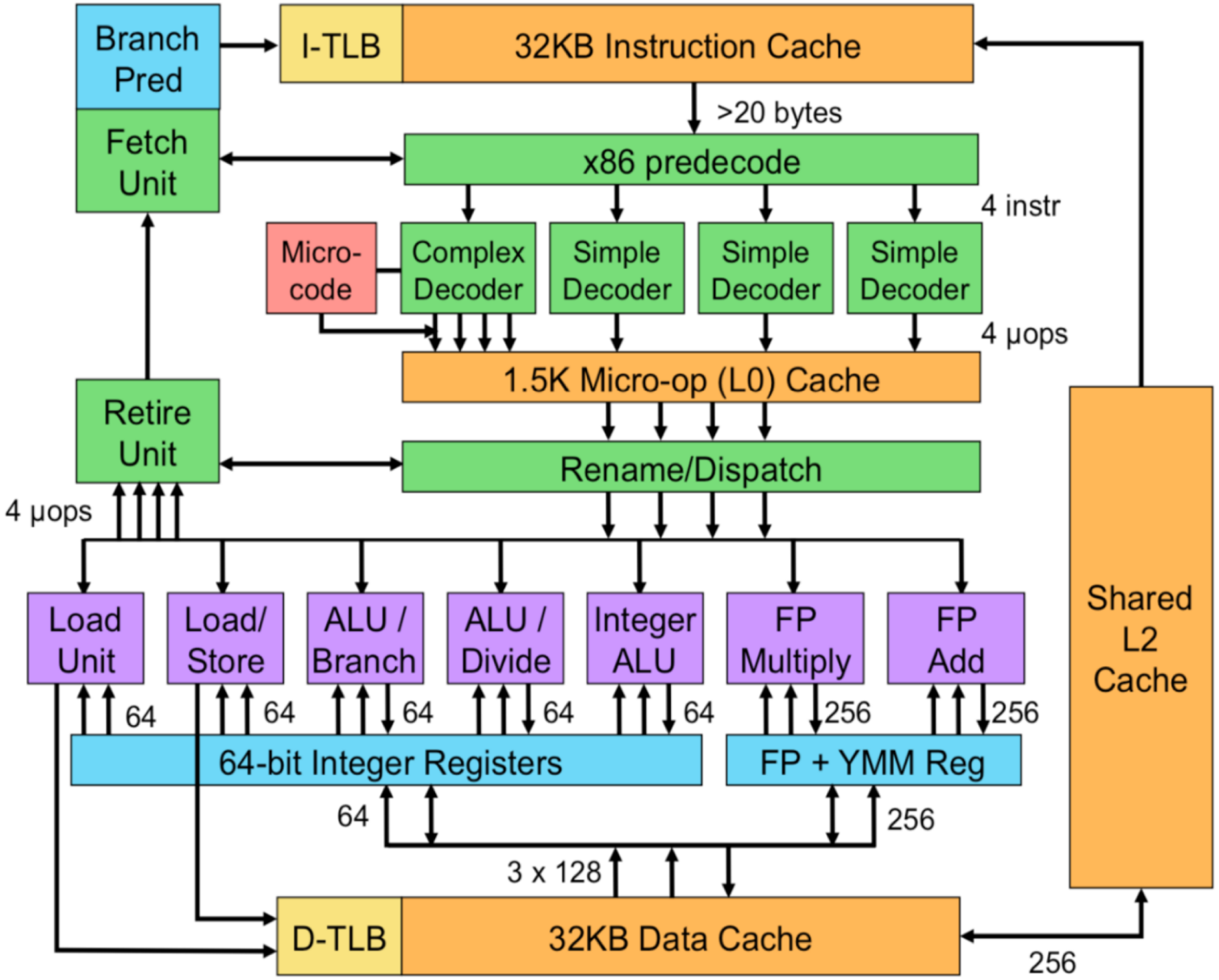
Multi-level decoder

Micro-code (?)

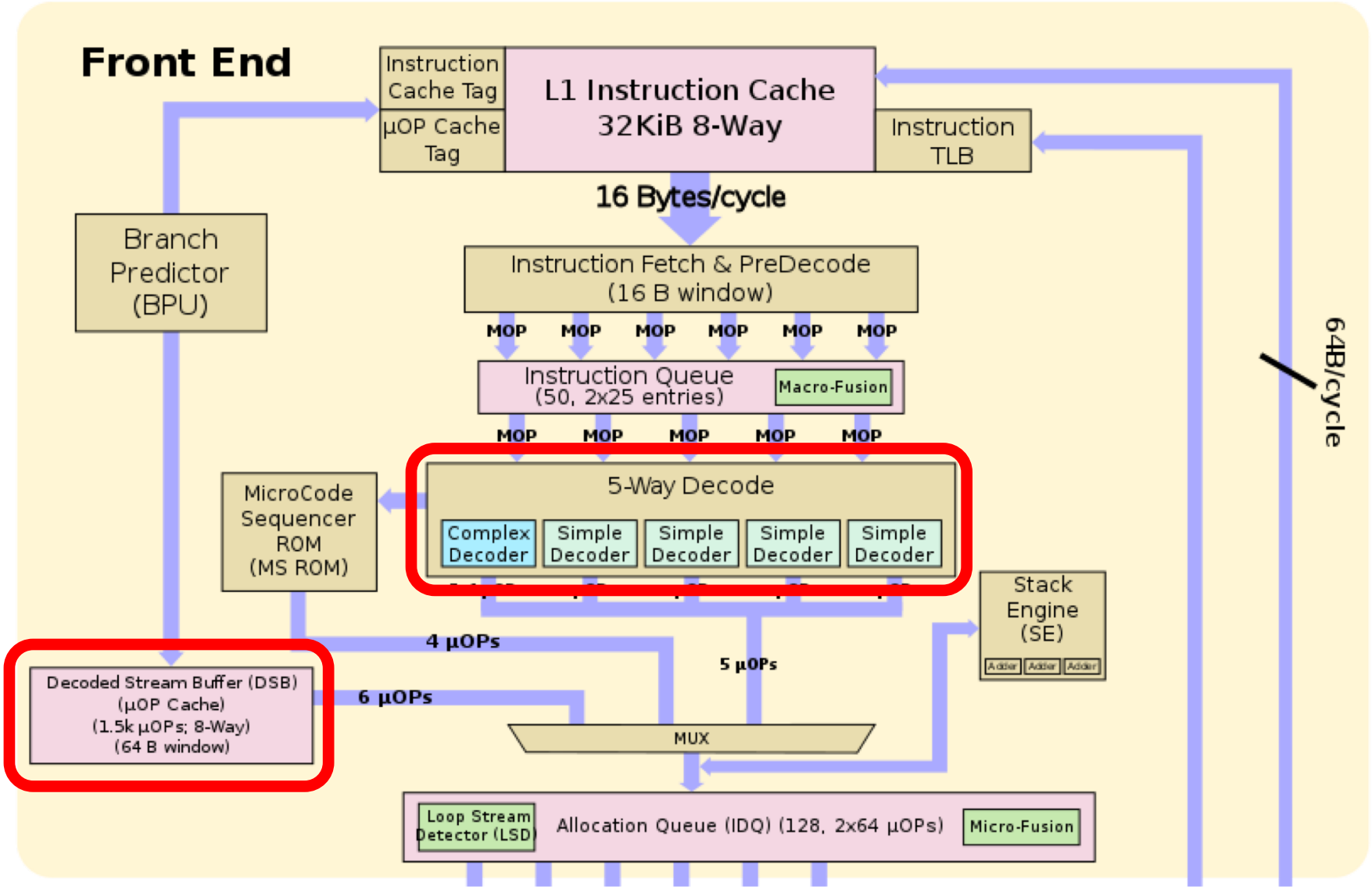
µ-op cache

Rename w/64 physical registers (not w/ #RS)

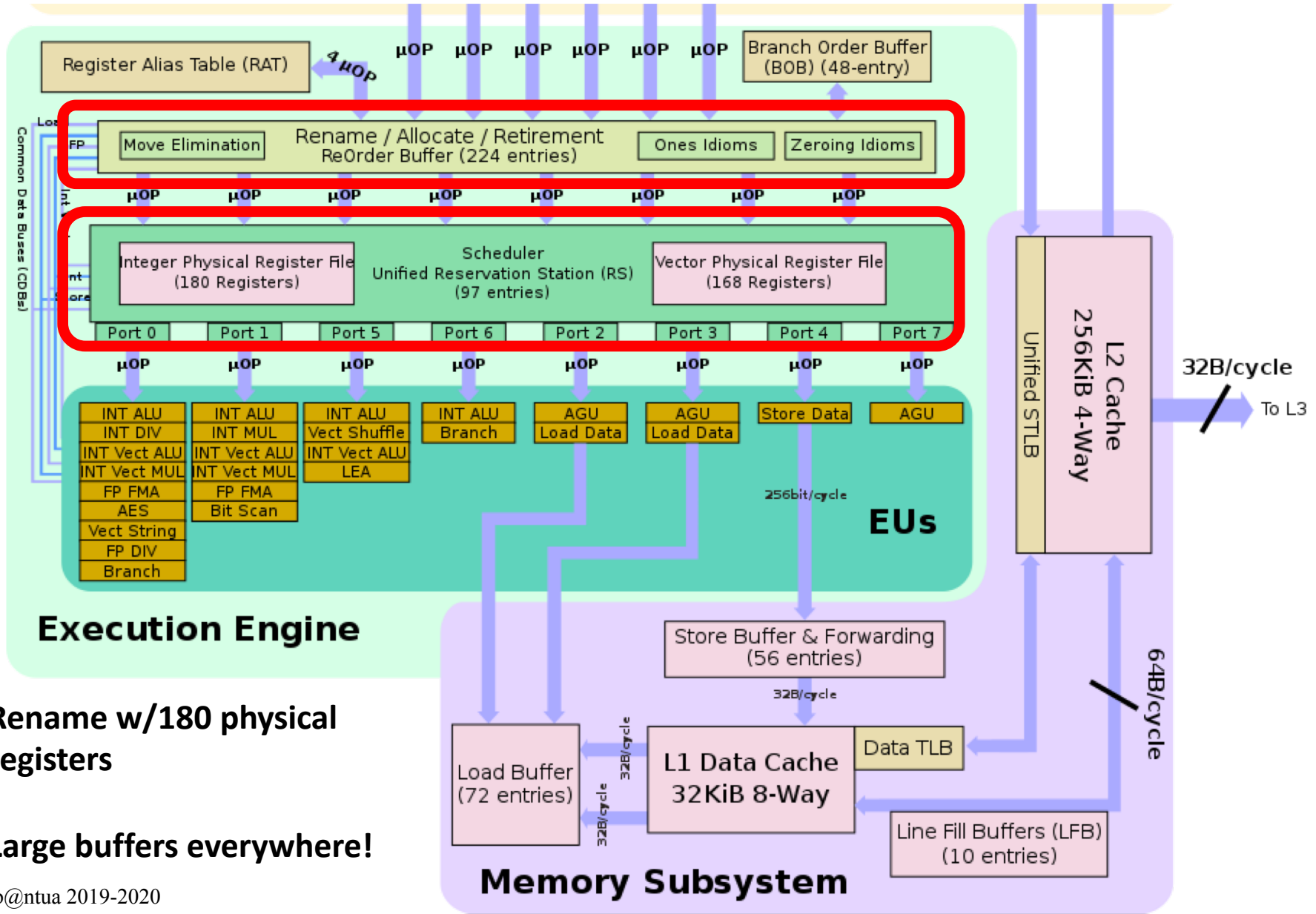
x86-64 arch registers = 16



Intel Skylake (6-9th gen i3/i5/i7) Front End



Intel Skylake (6-9th gen i3/i5/i7) Exec/Mem



Rename w/180 physical registers

Large buffers everywhere!

Μέρος Δεύτερο

Αύξηση απόδοσης με πολυνηματική επεξεργασία

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)

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

Αύξηση απόδοσης

- clock speed (↑Hz)
- αρχιτεκτονικές βελτιστοποιήσεις (↑IPC):
  - pipelining, superscalar execution, branch prediction, out-of-order execution, caches
- αλγόριθμοι (↓instr.count)

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

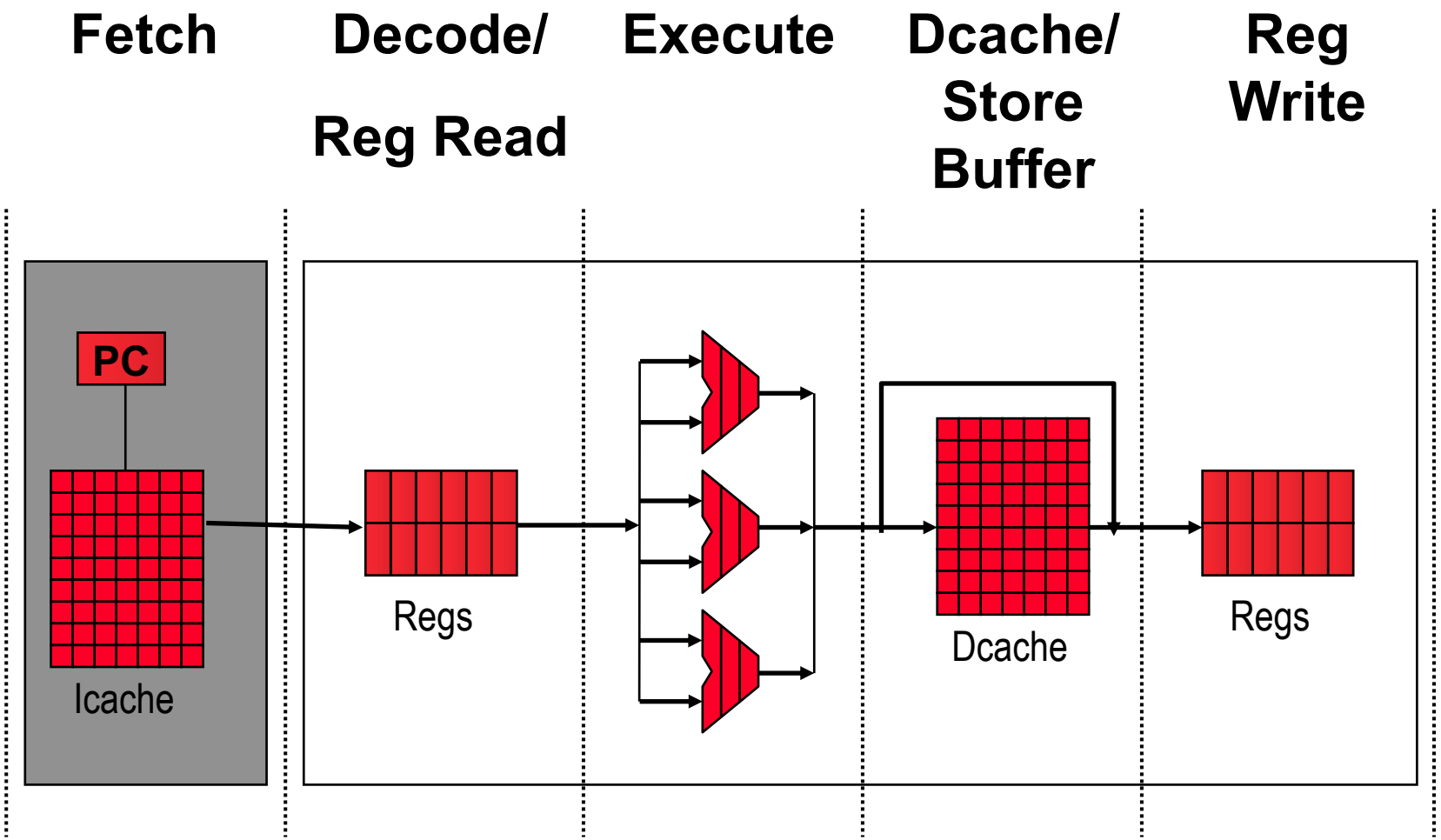
  → ανεκμετάλλευτες πολλές από τις μονάδες ενός superscalar επεξεργαστή
- Συχνότητα ρολογιού: φυσικά εμπόδια στη συνεχόμενη αύξησή της
  - μεγάλη έκλυση θερμότητας, μεγάλη κατανάλωση ισχύος, διαρροή ρεύματος

Πρέπει να βρούμε άλλον τρόπο πέρα από τον ILP + συχνότητα ρολογιού για να βελτιώσουμε την απόδοση

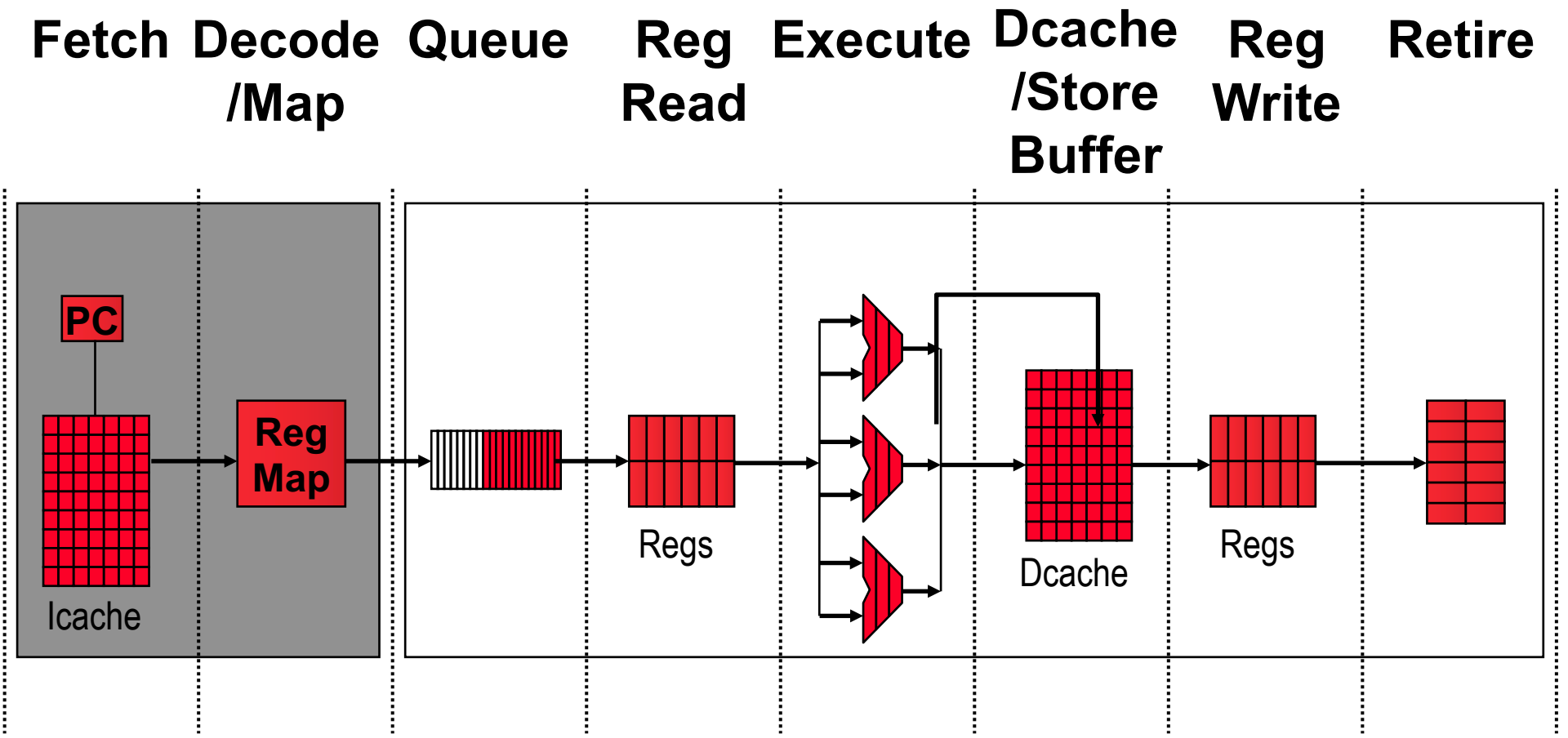
Προσεγγίσεις Αύξησης Απόδοσης

- **Παραλληλισμός σε επίπεδο εντολής (Instruction Level Parallelism – ILP)**
  - Εξαρτάται από τις πραγματικές εξαρτήσεις δεδομένων που υφίστανται ανάμεσα στις εντολές.
- **Παραλληλισμός σε επίπεδο νήματος (Thread-Level Parallelism – TLP)**
  - Αναπαρίσταται ρητά από τον προγραμματιστή, χρησιμοποιώντας πολλαπλά νήματα εκτέλεσης τα οποία είναι εκ κατασκευής παράλληλα.
- **Παραλληλισμός σε επίπεδο δεδομένων (Data-Level Parallelism – DLP)**
  - Αναπαρίσταται ρητά από τον προγραμματιστή ή δημιουργείται αυτόματα από τον μεταγλωττιστή.
  - Οι ίδιες εντολές επεξεργάζονται πολλαπλά δεδομένα ταυτόχρονα

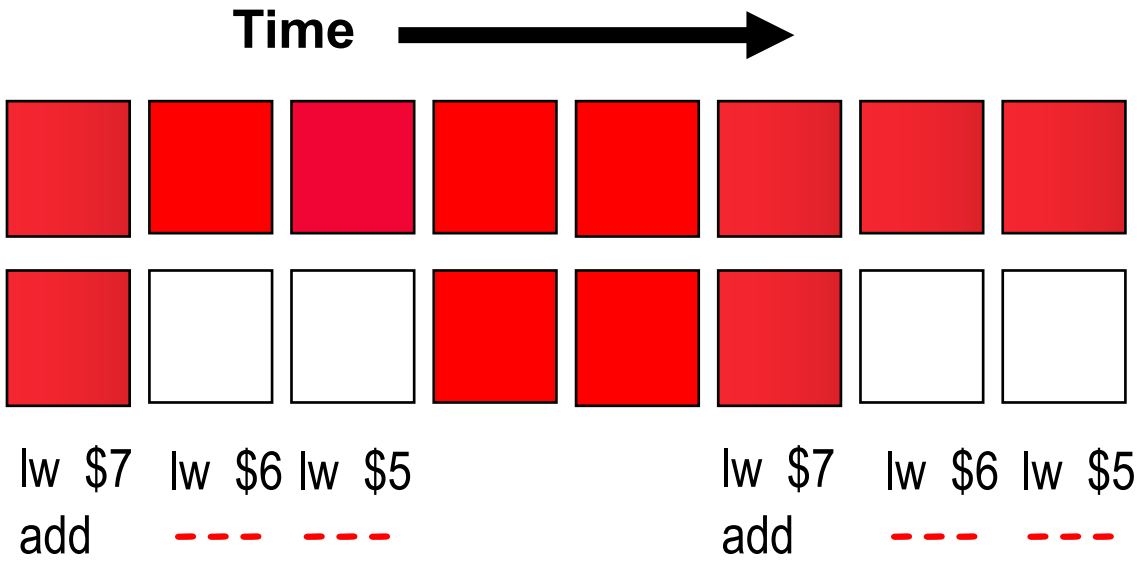
Review: Basic Scalar Pipeline



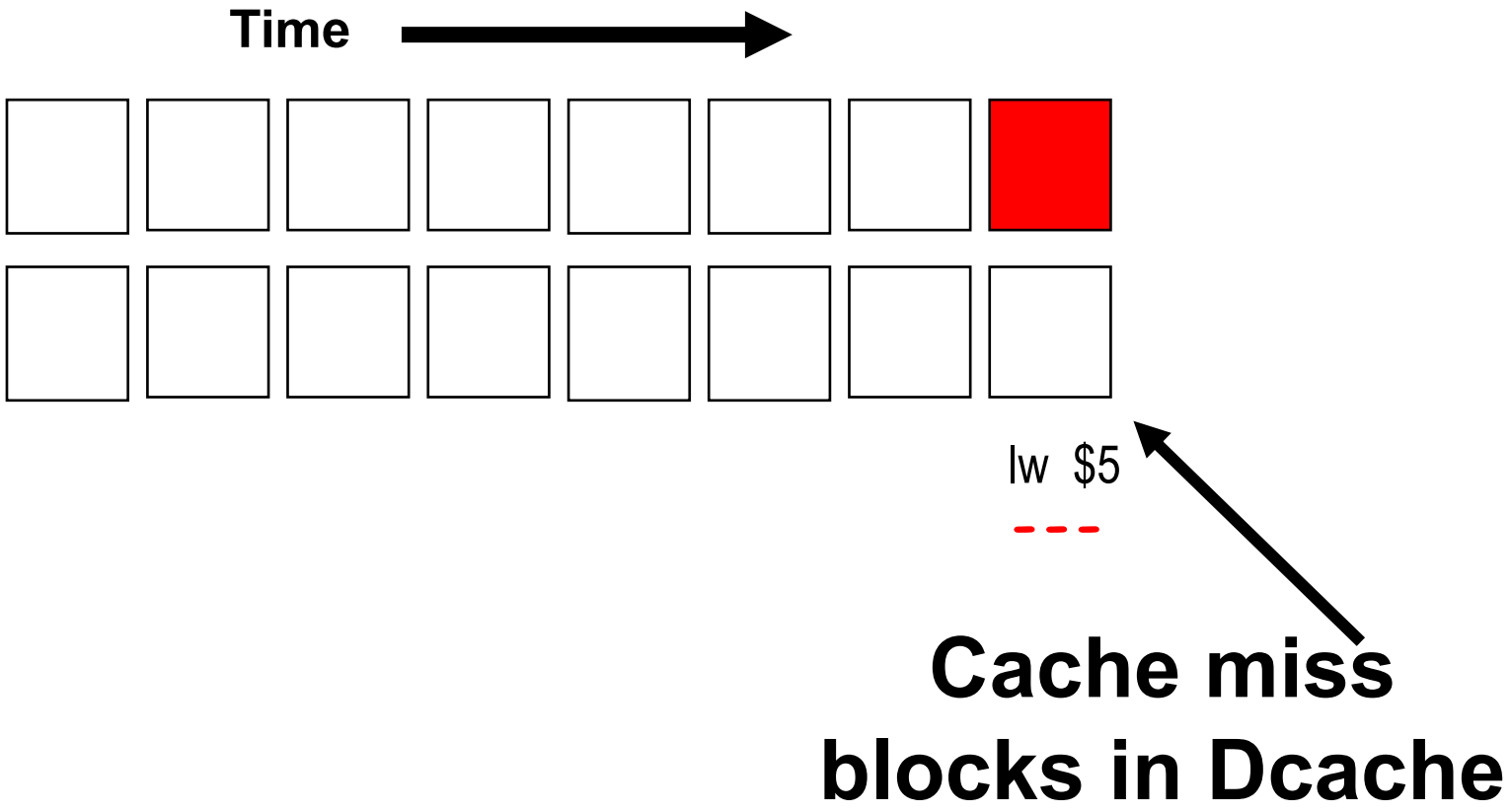
Review: Basic Superscalar Pipeline



High-Level Overview of the pipeline

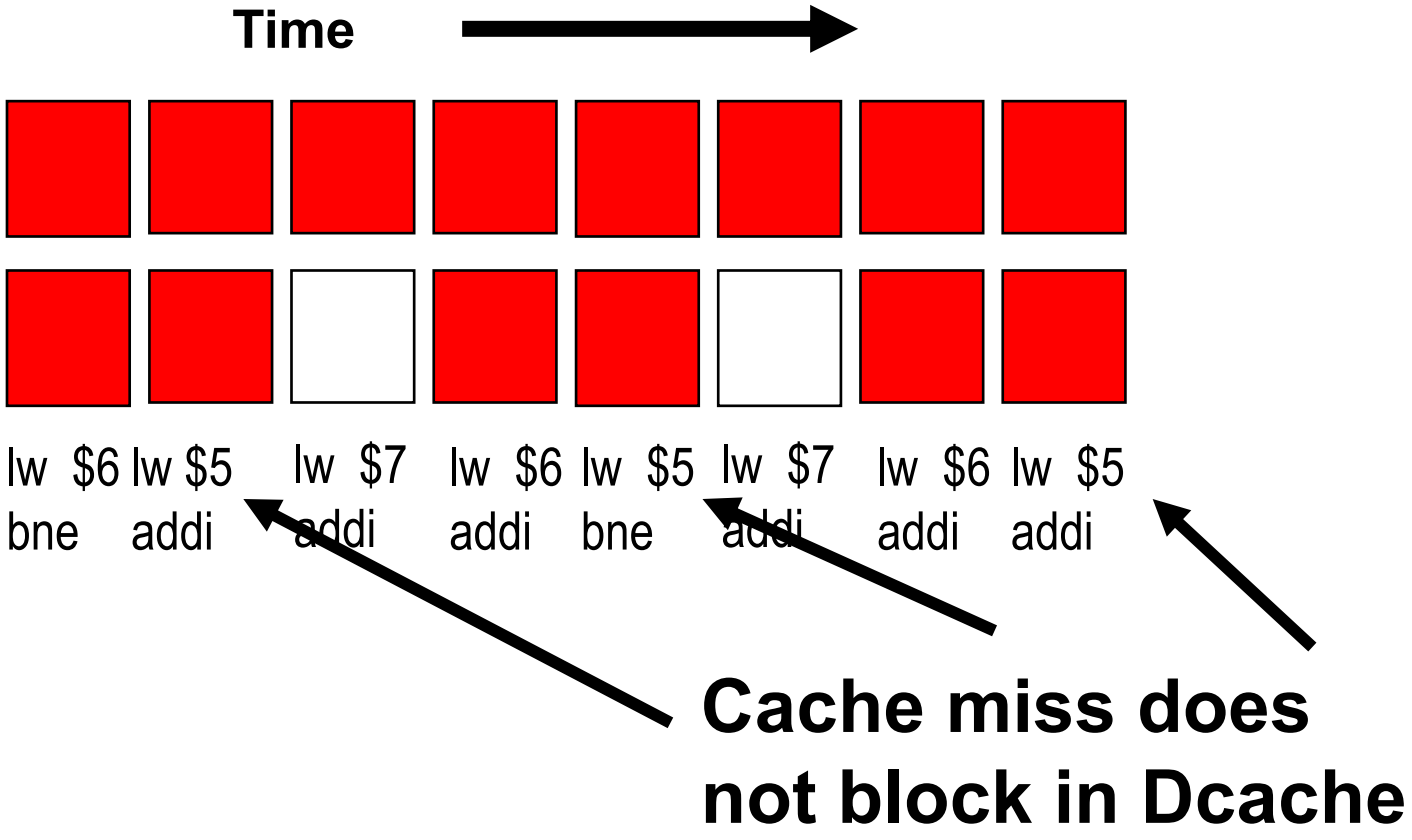


In-order pipeline with cache misses



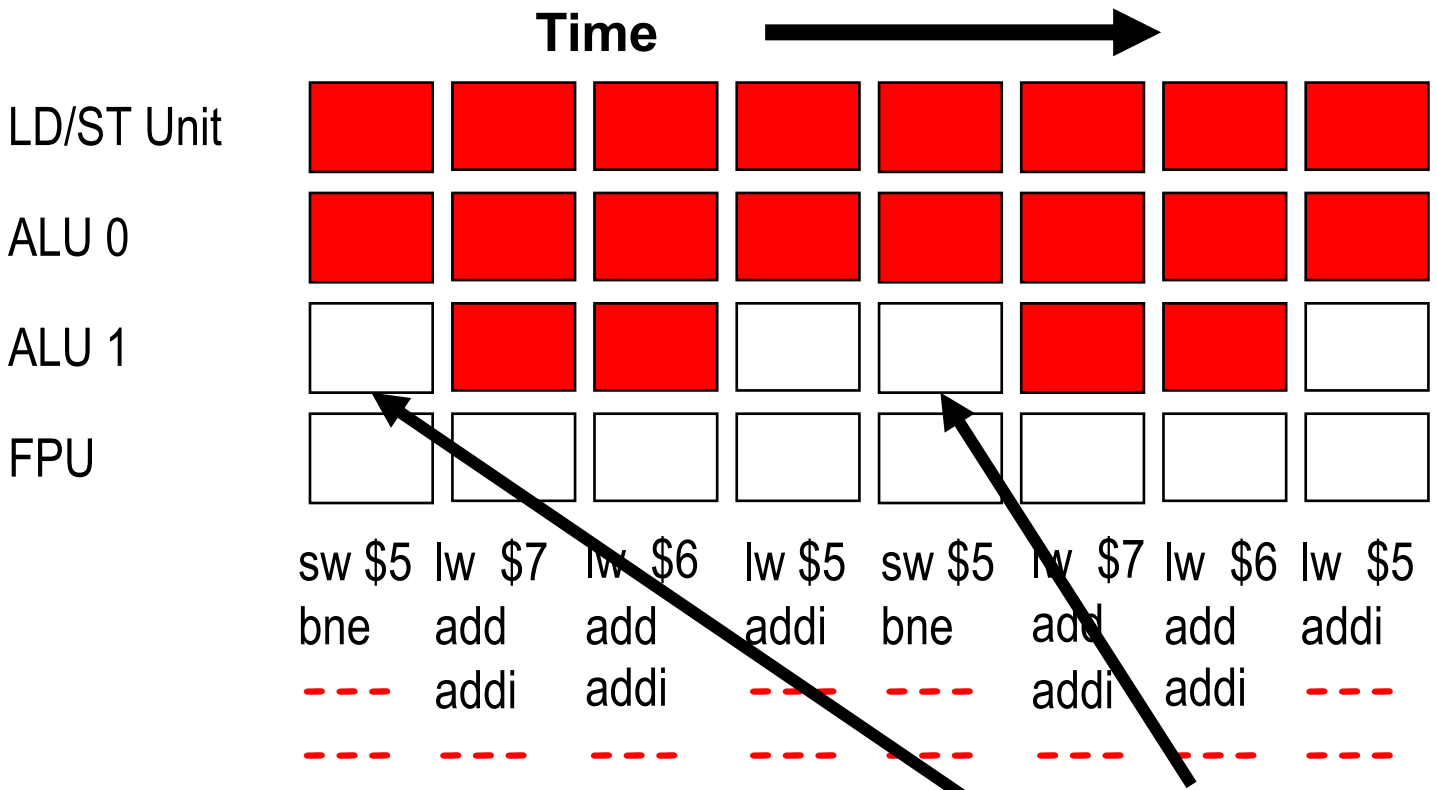
Out-of-order pipeline with cache misses

- Other dependent instructions sitting in instruction queue



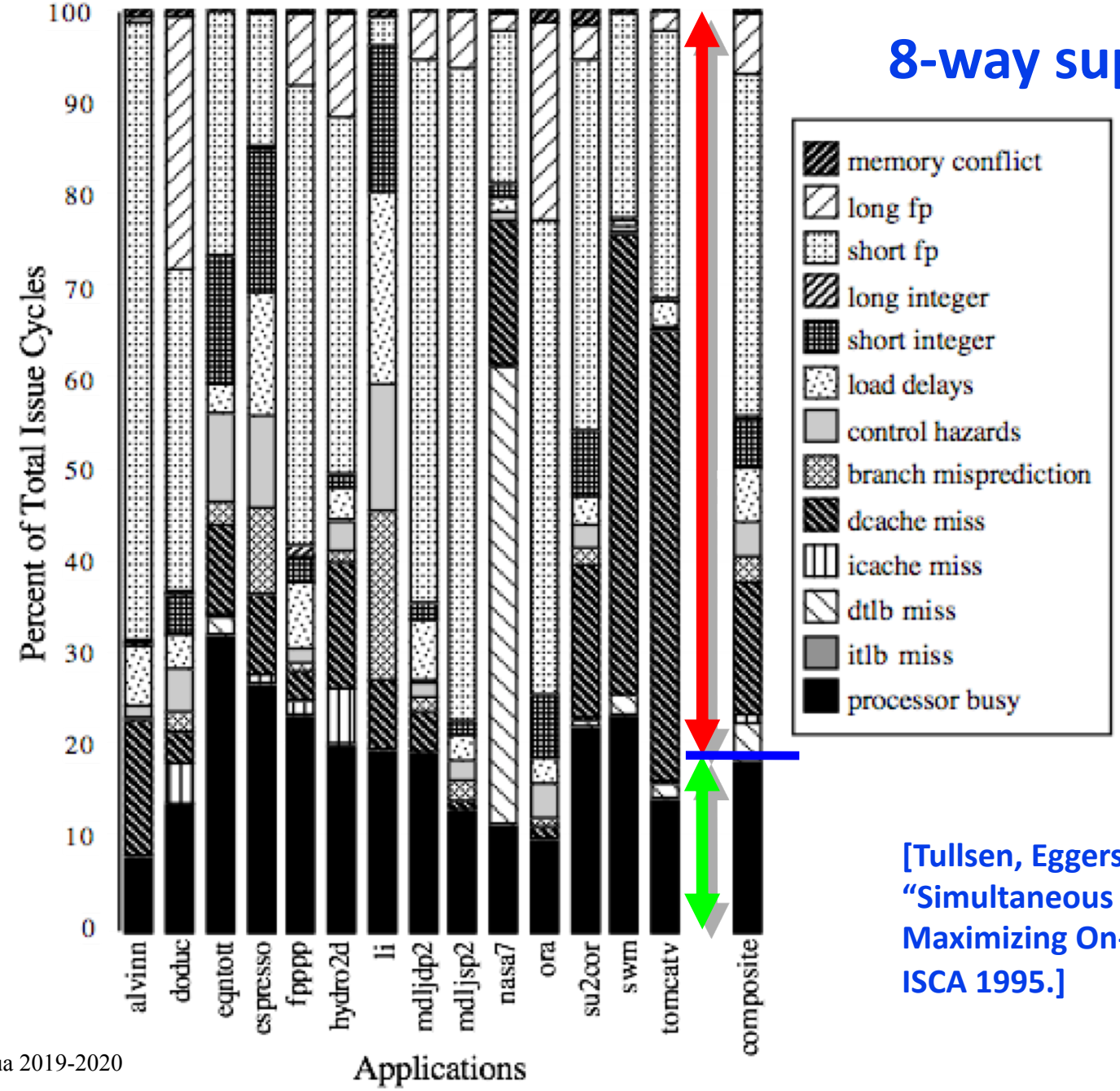
4-way Superscalar pipeline

- Other dependent instructions sitting in instruction queue



Not enough independent instructions to keep pipeline full => low utilization!

Ανεκμετάλλευτοι πόροι σε OoO superscalar processor



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

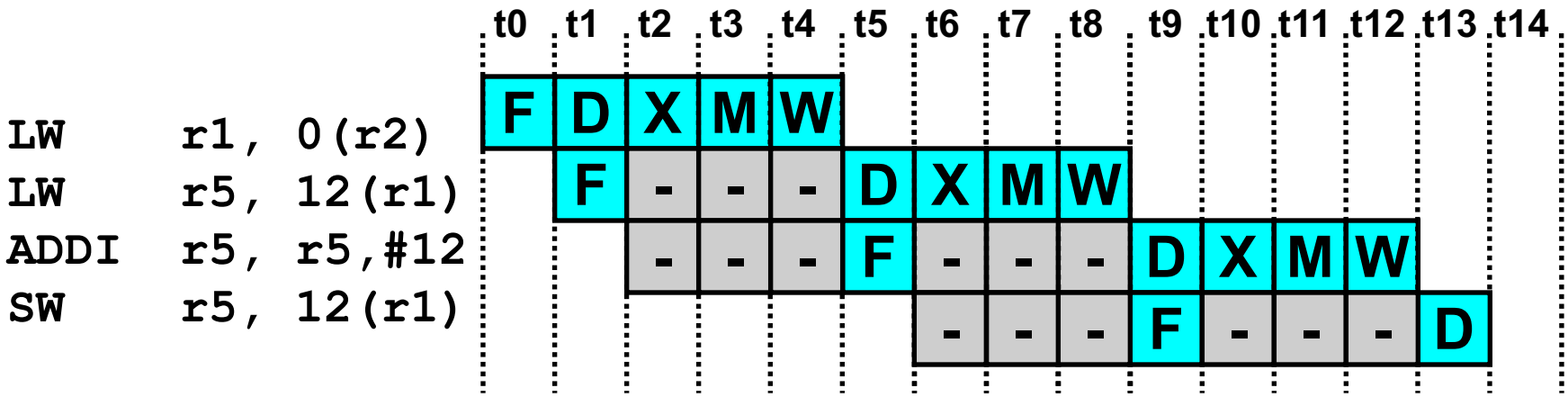
Πολυνηματικές Αρχιτεκτονικές

Πολυνηματισμός

- Σκοπός: χρήση πολλών ανεξάρτητων instruction streams από πολλαπλά νήματα εκτέλεσης
- **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



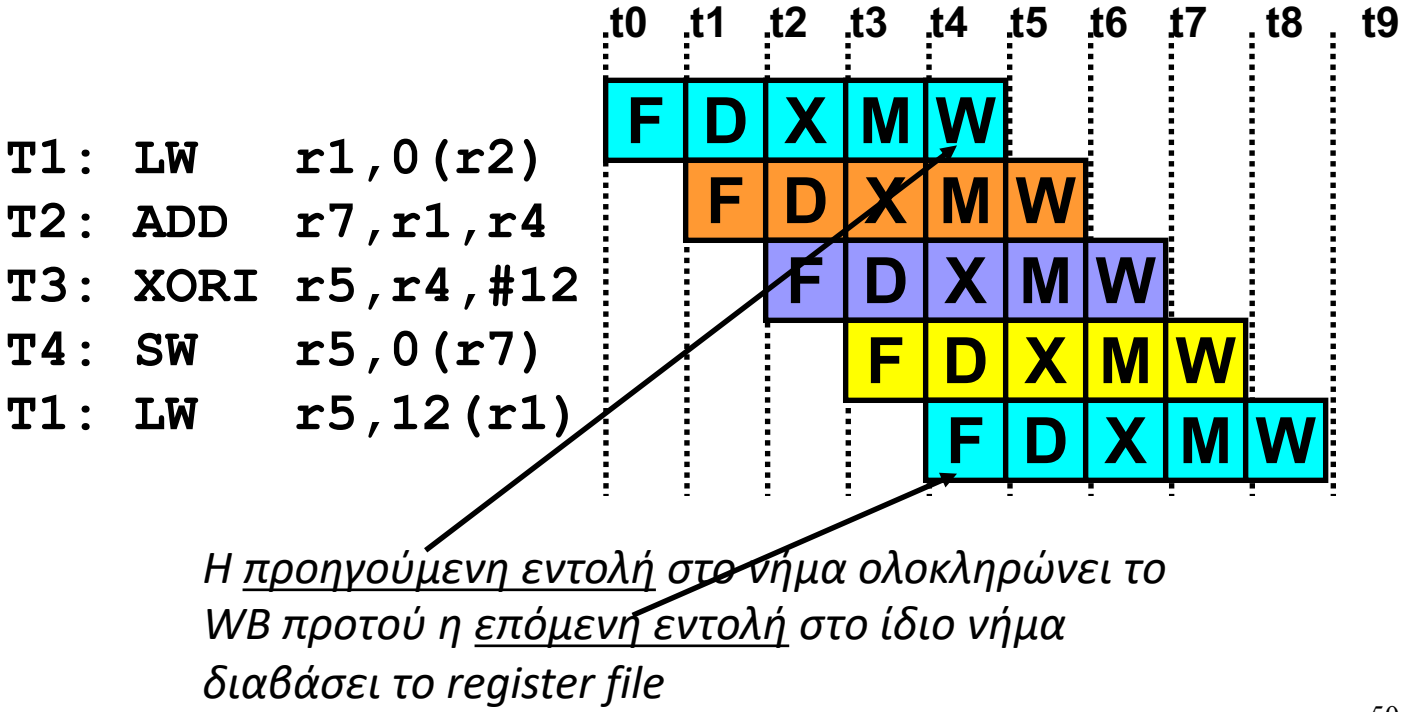
- Οι εξαρτήσεις μεταξύ των εντολών αποτελούν περιοριστικό παράγοντα για την εξαγωγή παραλληλισμού
- Τι μπορεί να γίνει προς αυτή τη κατεύθυνση;

Αντιμετώπιση με πολυνηματισμό

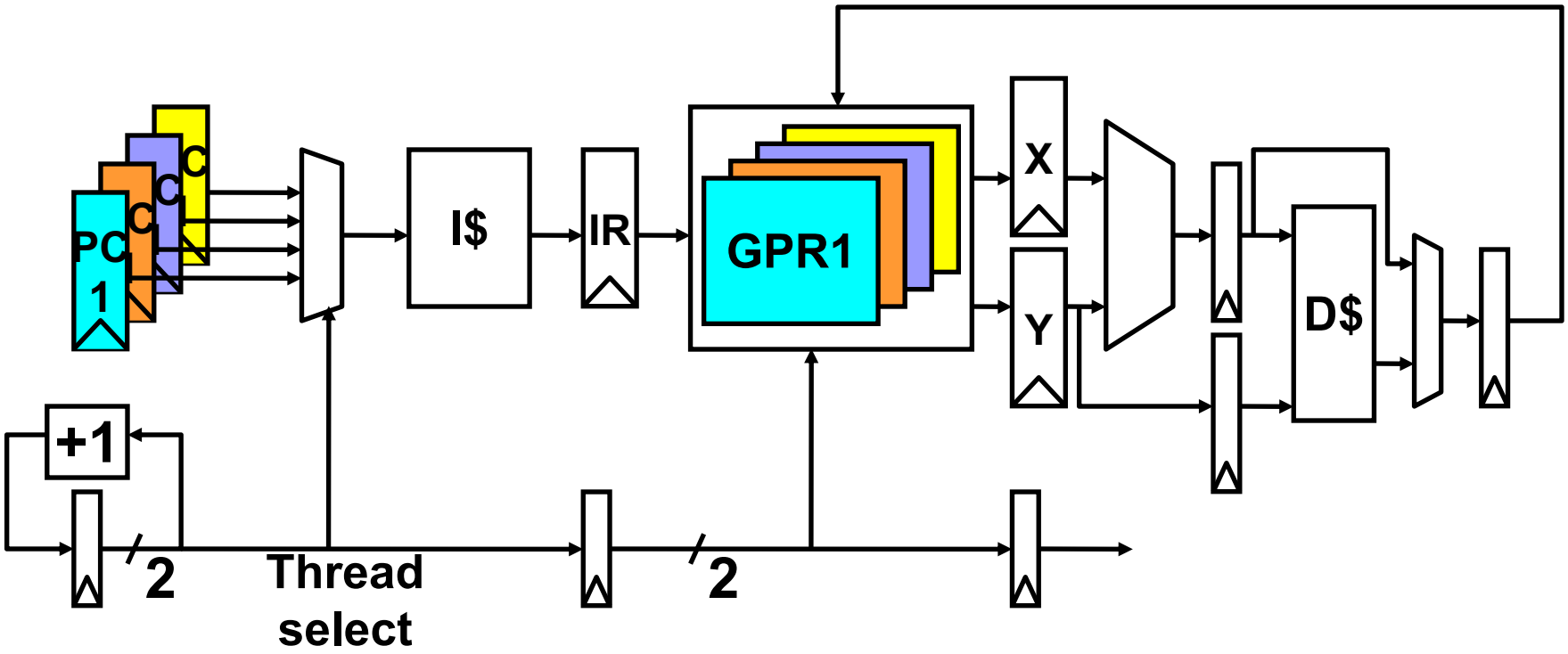
Πώς μπορούμε να μειώσουμε τις εξαρτήσεις μεταξύ των εντολών σε ένα pipeline?

- 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



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 (≈ 100s - 1000s κύκλων)

Πολυνηματισμός

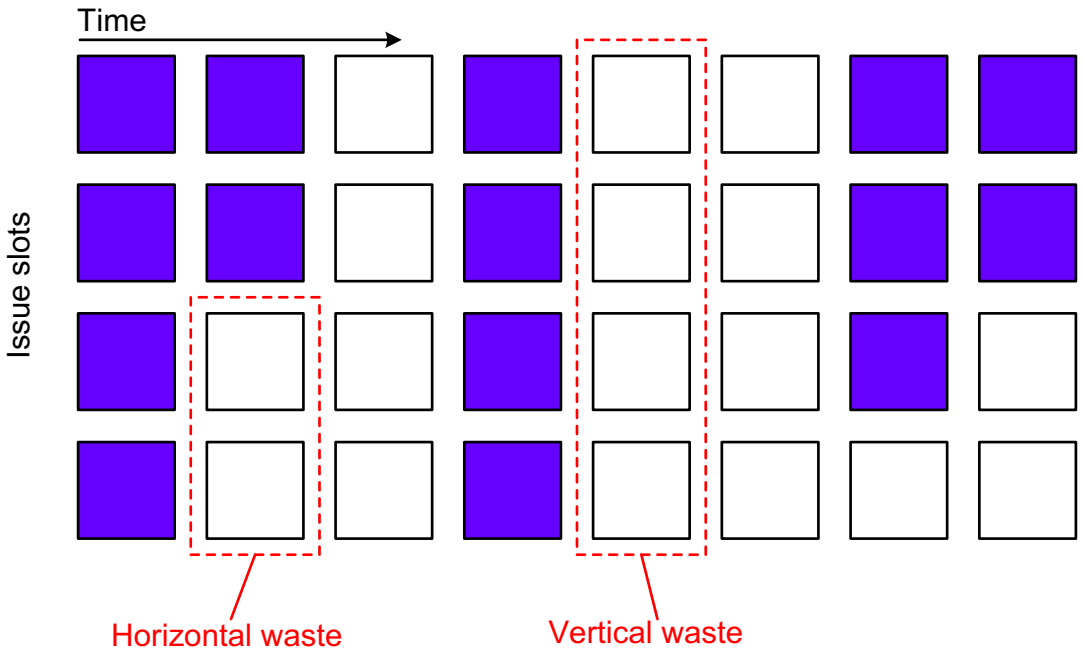
- Πλεονεκτήματα
  - Δε χρειάζεται dependency checking μεταξύ των εντολών
  - Δε χρειάζεται branch prediction
  - Αποφυγή bubbles πραγματοποιώντας χρήσιμη δουλειά από άλλα threads
  - Βελτίωση system throughput, utilization, latency tolerance

- 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)

Επιπλέον υλικό και πληροφορίες:

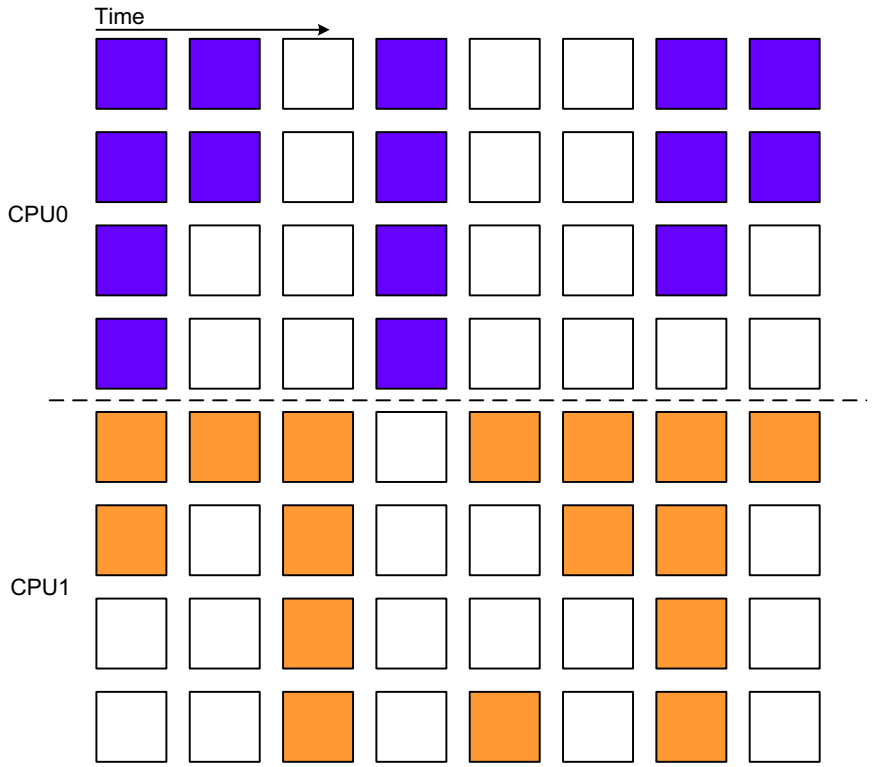
- Mario Nemirovsky, Dean M. Tullsen. **Multithreading Architecture**. Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers 2013, ISBN 9781608458554.

OoO superscalar



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

Chip Multi-Processor

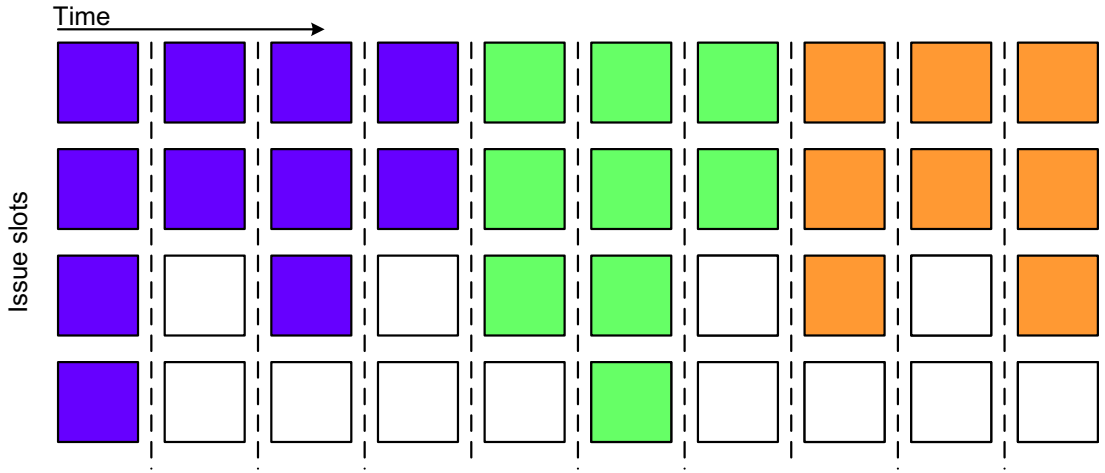


- τα προβλήματα εξακολουθούν να υφίστανται...

Υλοποιήσεις Πολυνηματισμού

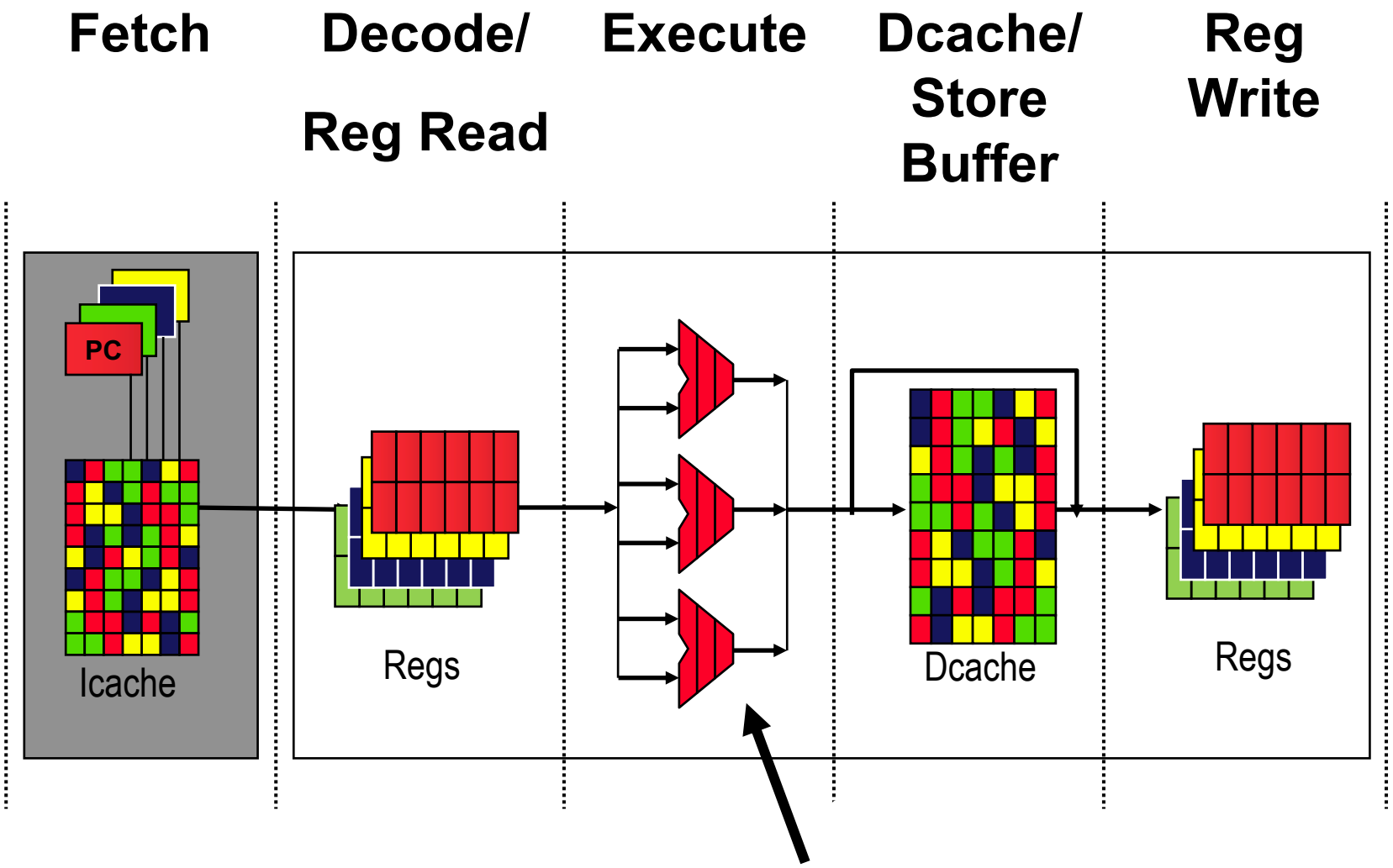
1. Coarse-grained multithreading
2. Fine-grained multithreading
3. Simultaneous multithreading (SMT)

Coarse-grain multithreading ("switch-on-event")

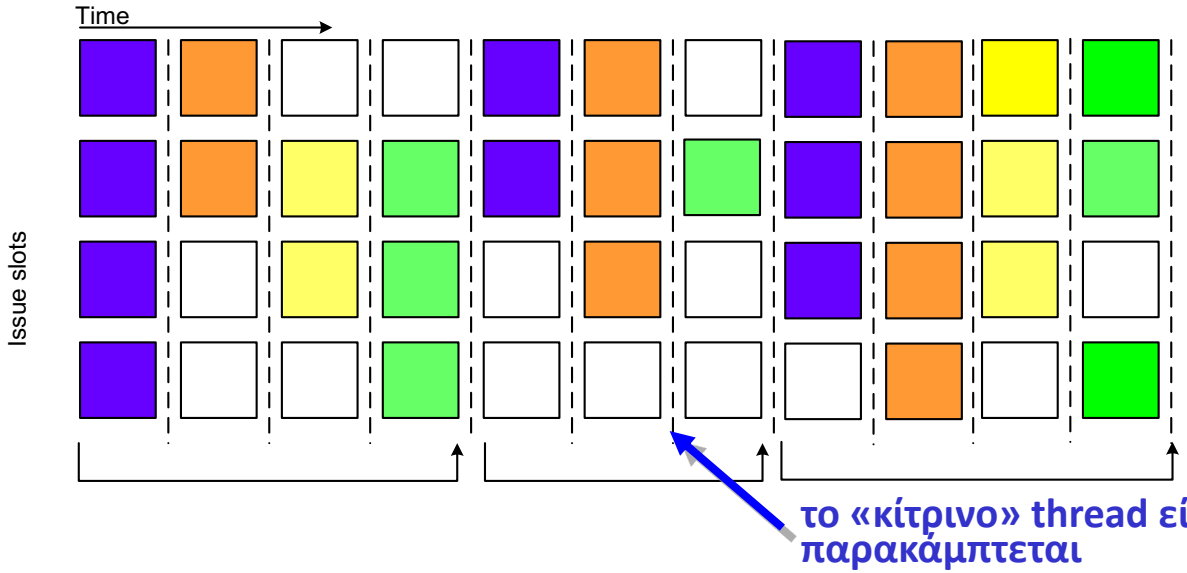


- Εναλλαγή thread μόνο μετά από stall του thread που εκτελείται, π.χ. λόγω L2 cache miss
- Πλεονεκτήματα:
  - δε χρειάζεται να έχει πολύ γρήγορο μηχανισμό εναλλαγής των threads
  - δεν καθυστερεί την εκτέλεση ενός thread, αφού οι εντολές από άλλα threads γίνονται issue μόνο όταν το thread αντιμετωπίσει 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 >> pipeline refill time
- e.g., IBM AS/400, DYSEAC, TX-2

Coarse-grain Multithreading

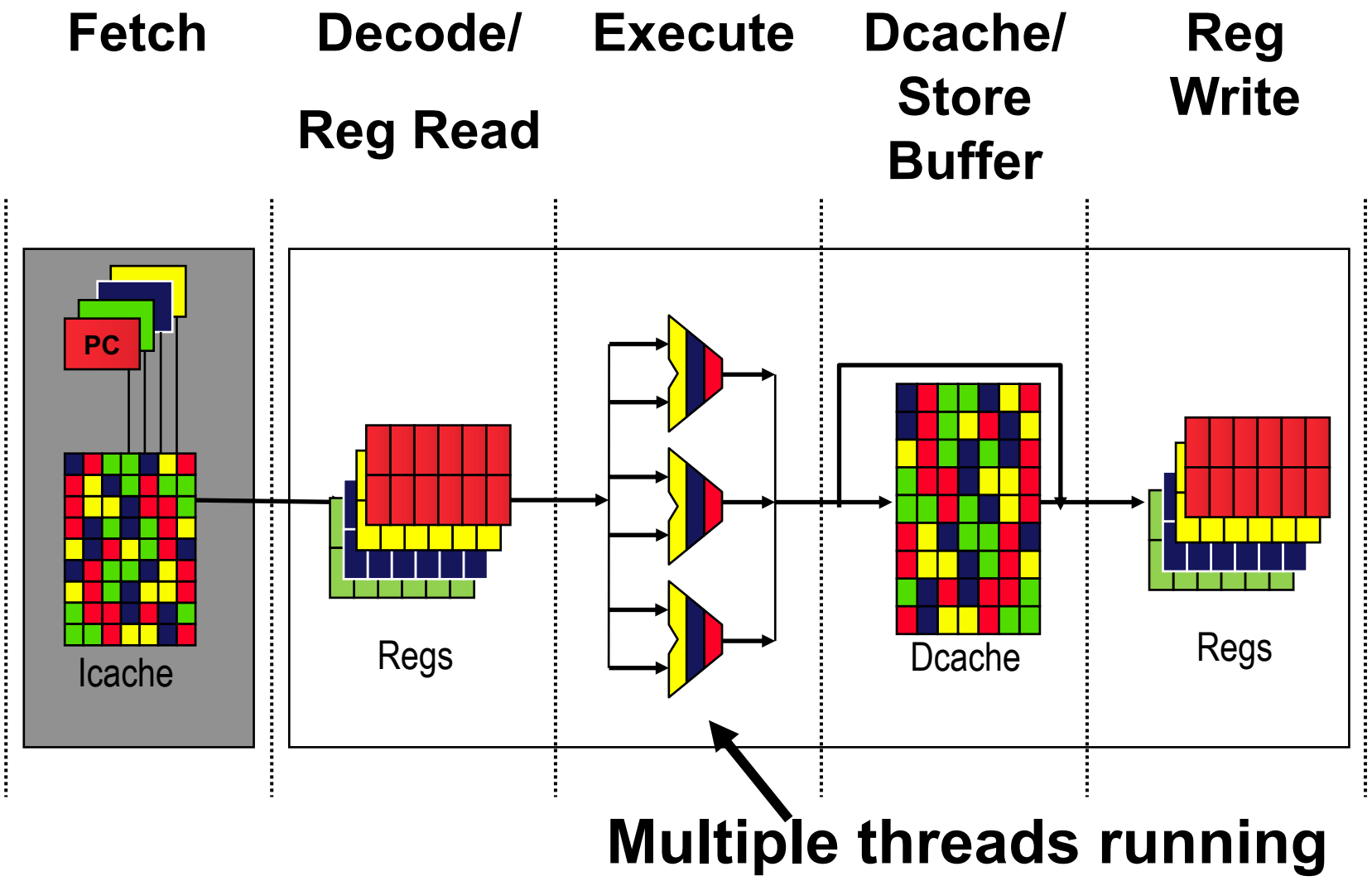


Fine-grained multithreading

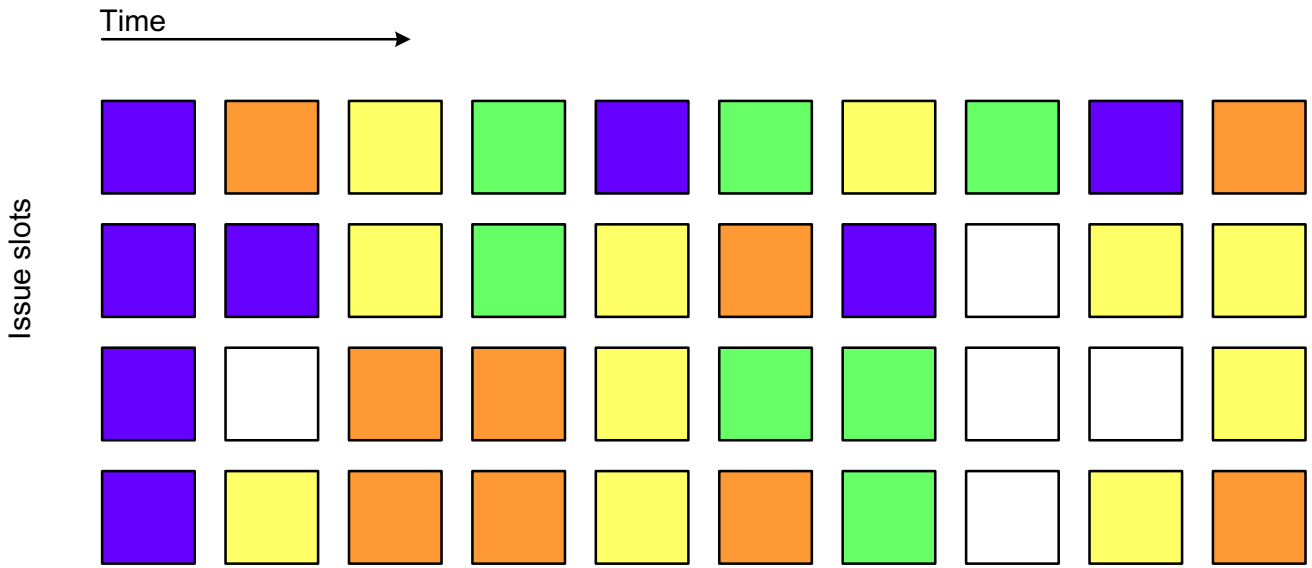


- Εναλλαγή μεταξύ των threads σε κάθε κύκλο, με αποτέλεσμα την επικάλυψη της εκτέλεσης των threads
  - η CPU είναι αυτή που κάνει την εναλλαγή σε κάθε κύκλο
- Γίνεται με round-robin τρόπο (κυκλικά), παρακάμπτοντας threads τα οποία είναι stalled σε κάποιο long-latency γεγονός
- Αντιμετωπίζει το vertical 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, CDC 6600, Delco TIO

Fine-Grained Multithreading

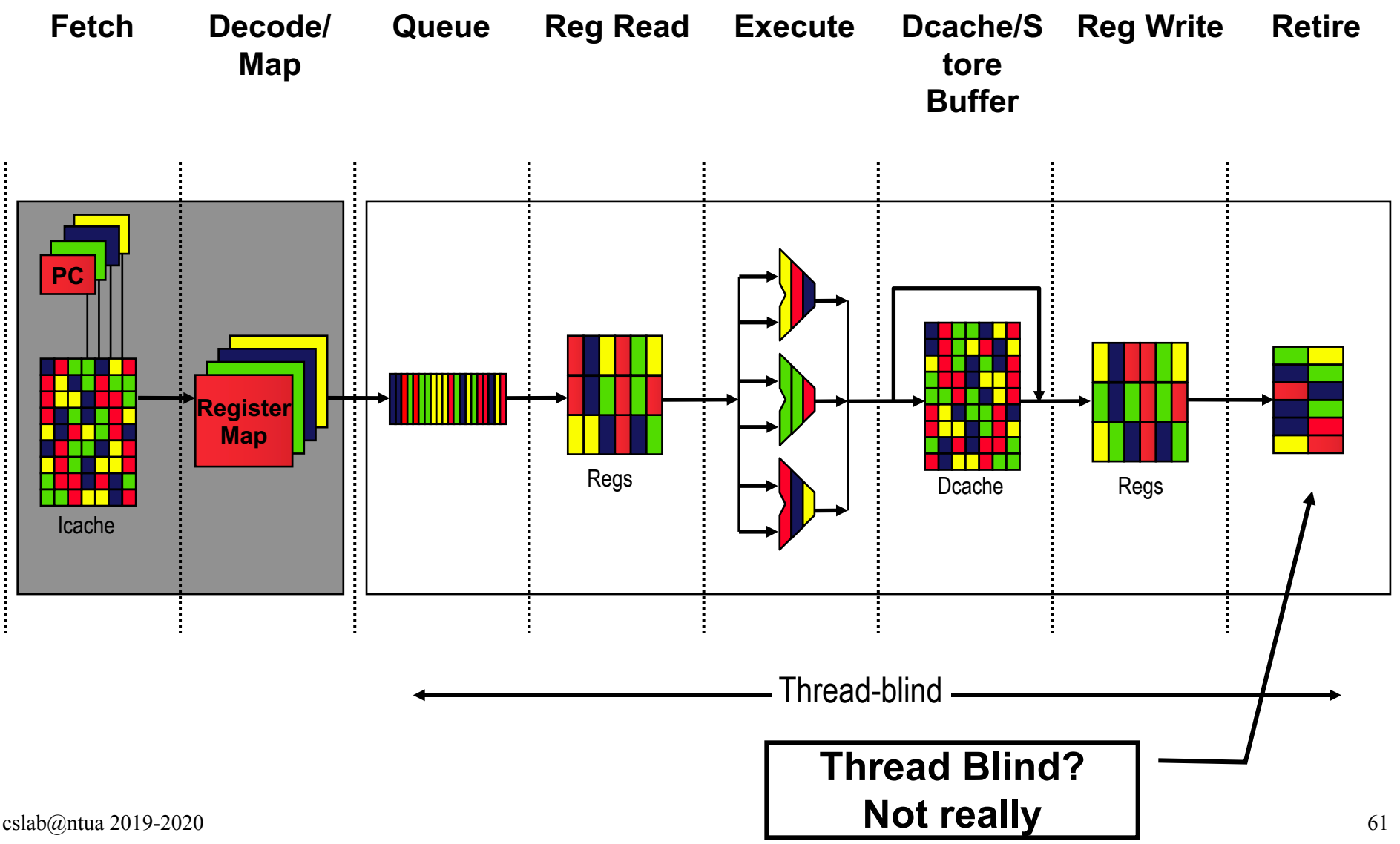


Simultaneous Multithreading (SMT)



- 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

Simultaneous Multithreading

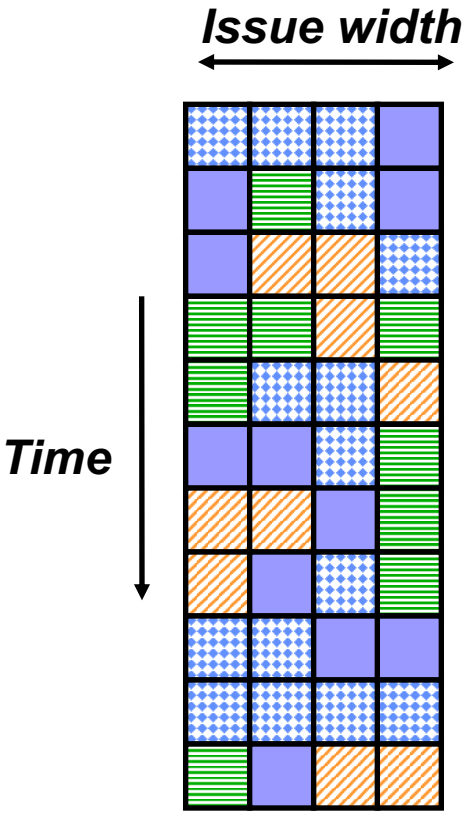


Αρχιτεκτονική SMT

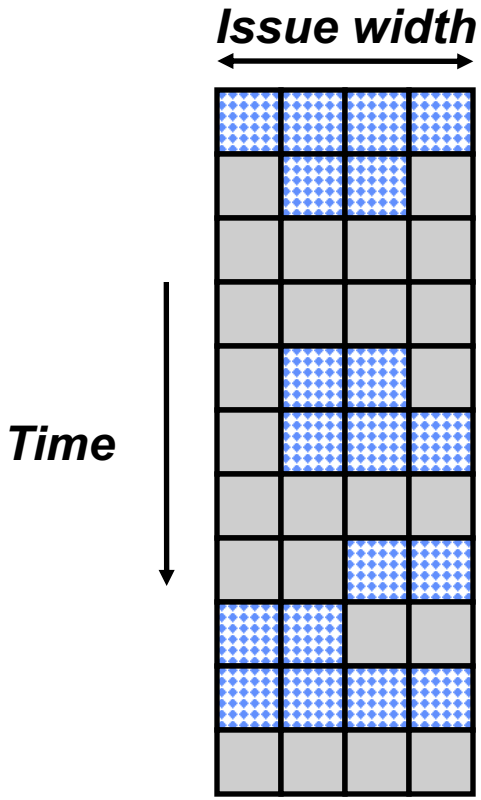
- Βασικές επεκτάσεις σε σχέση με μια συμβατική superscalar αρχιτεκτονική
  - **πολλαπλοί program counters** και κατάλληλος μηχανισμός μέσω του οποίου η fetch unit επιλέγει κάποιον από αυτούς σε κάθε κύκλο (π.χ. με βάση κάποια συγκεκριμένη πολιτική)
  - **thread-id σε κάθε BTB entry** για την αποφυγή 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*
  - **Κρίσιμη επιλογή:** fetch-interleaving policy

Προσαρμοστικότητα του SMT στο είδος του διαθέσιμου παραλληλισμού

Για περιοχές με υψηλά επίπεδα TLP, ολόκληρο το έυρος 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



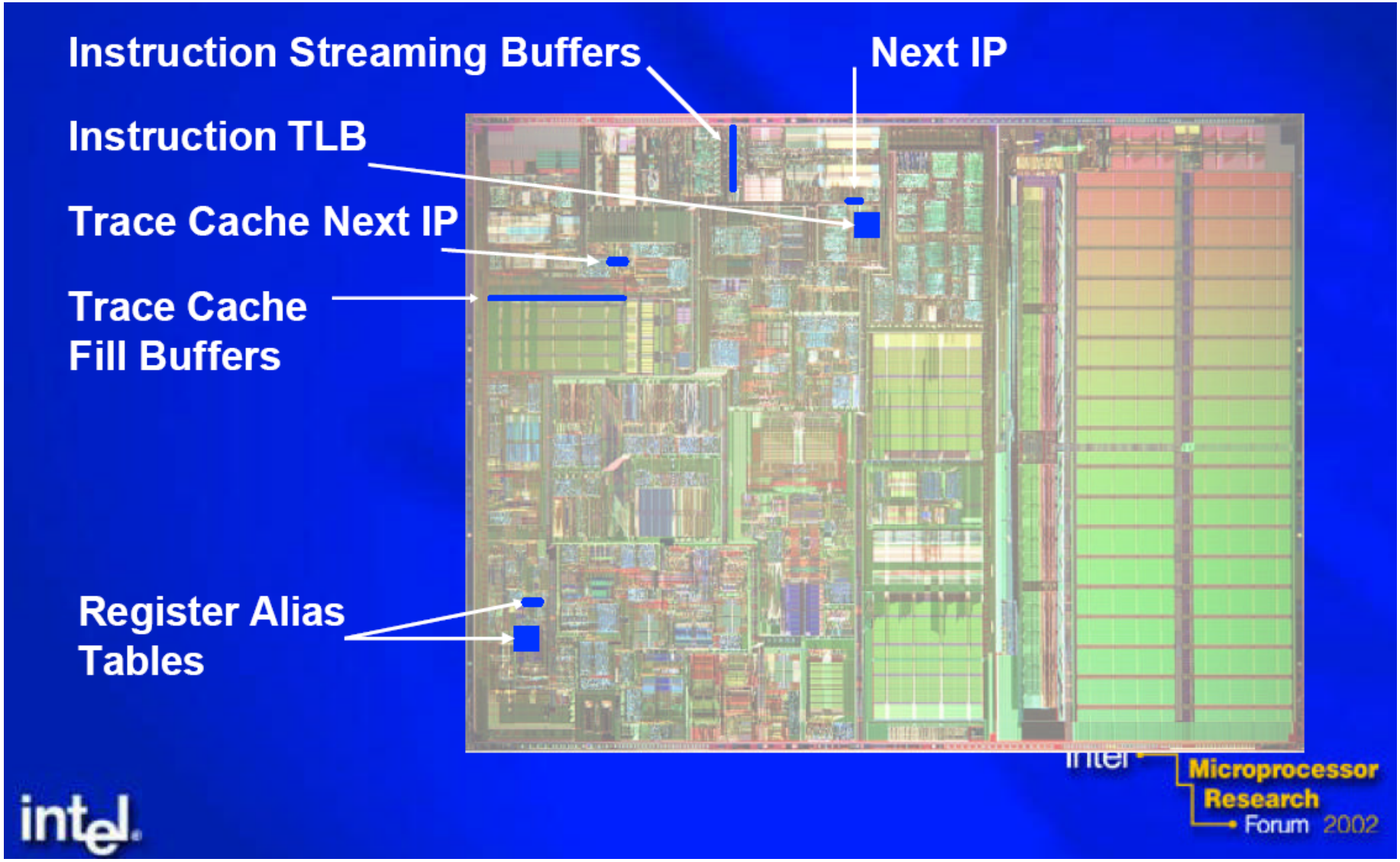
Case Studies

1. Pentium 4
2. Power5
3. UltraSPARC T1

Case-study 1: Pentium 4
SMT in Hyperthreading technology (~2002)

- 2-way SMT (Hyperthreading στην ορολογία της Intel)
  - 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

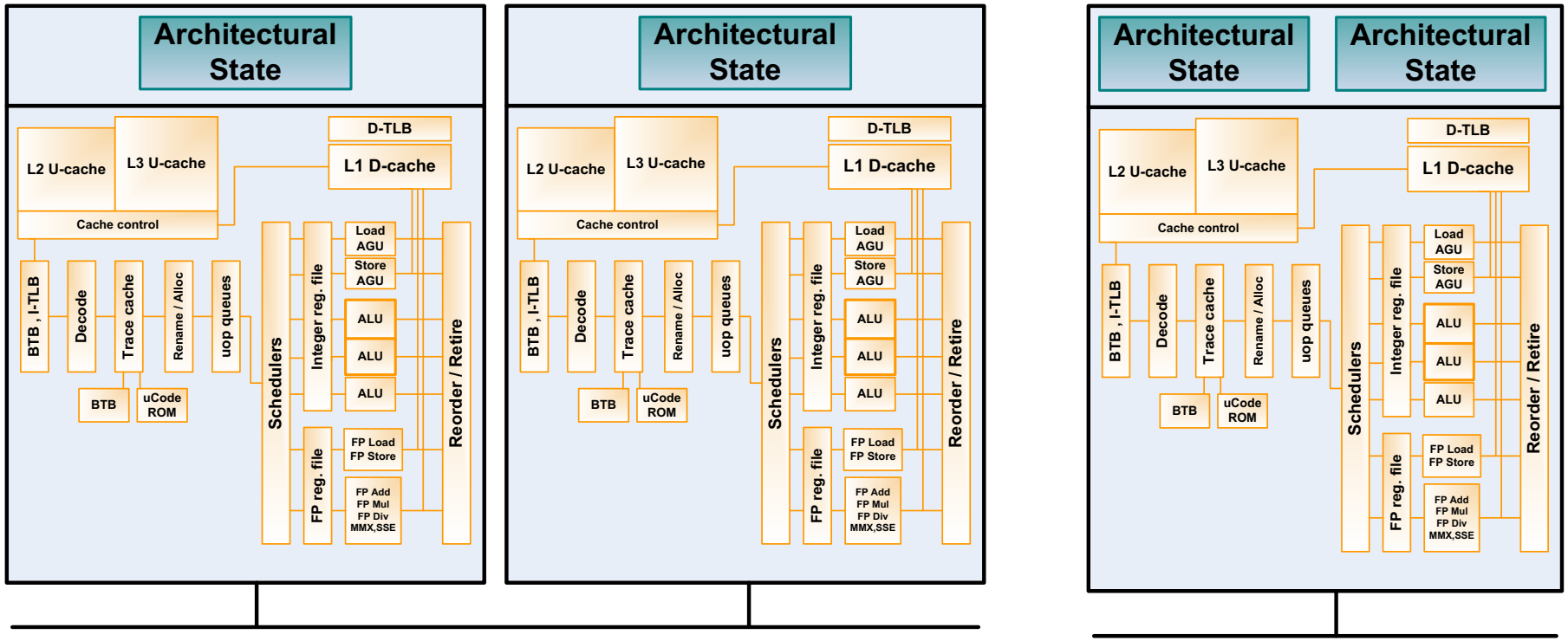
Τι προστέθηκε...



Επεξεργαστικοί πόροι: πολλαπλά αντίγραφα vs. διαμοιρασμός

Multiprocessor

Hyperthreading

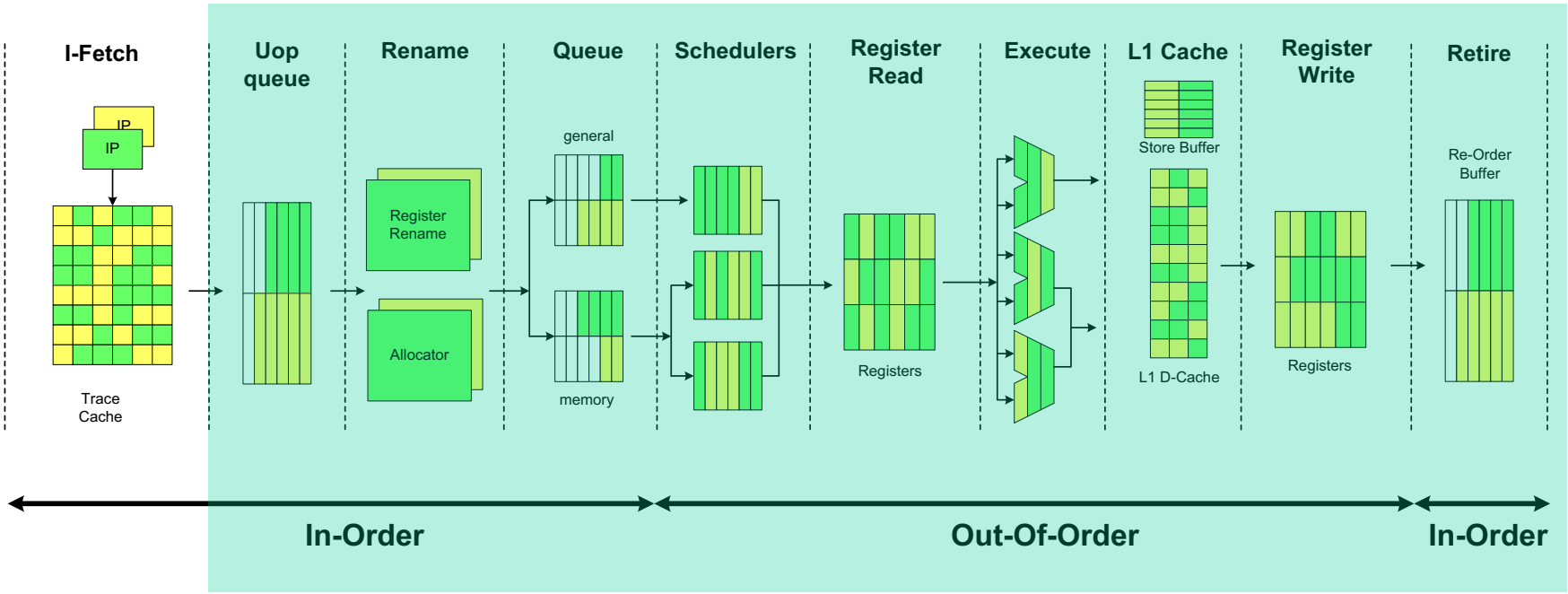


- στους πολυεπεξεργαστές τα resources βρίσκονται σε πολλαπλά αντίγραφα
- στο Hyper-threading τα resources διαμοιράζονται

Διαχείριση επεξεργαστικών πόρων

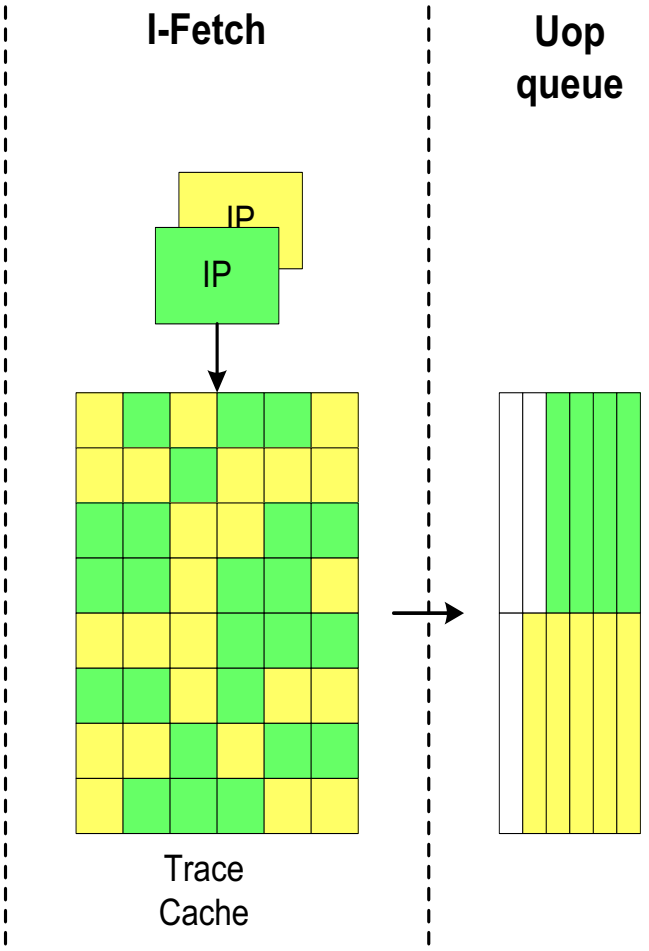
- Πόροι σε πολλαπλά αντίγραφα:
  - αρχιτεκτονική κατάσταση: GPRs, control registers, APICs
  - instruction pointers, renaming hardware
  - smaller resources: ITLBs, branch target buffer, return address stack
- Στατικά διαχωρισμένοι πόροι:
  - ROB, Load/Store queues, instruction queues
  - κάθε νήμα μπορεί 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

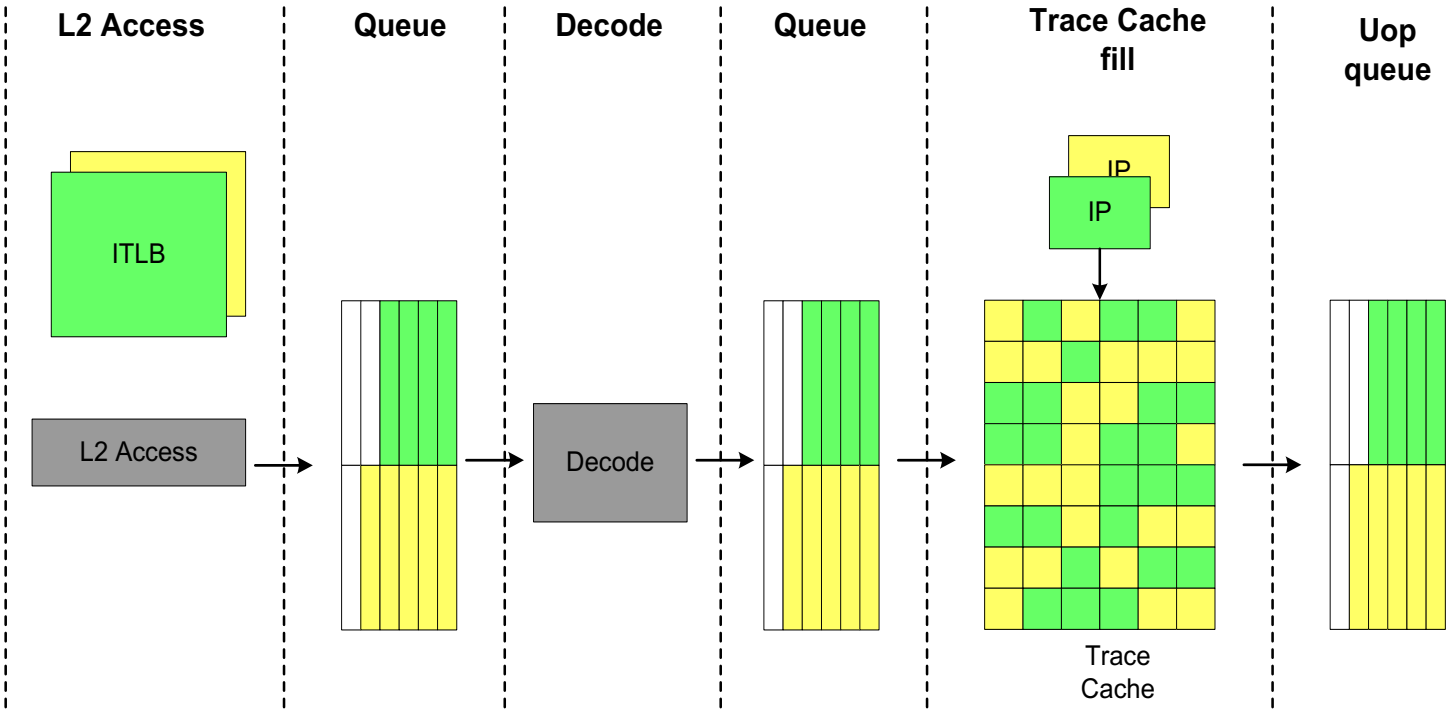


Pentium 4 w/ Hyper-Threading: Front End – trace cache hit

- 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 (Logical Processors), η πρόσβαση εναλλάσσεται κύκλο-ανά-κύκλο
- Όταν μόνο 1 LP ζητάει πρόσβαση, μπορεί να χρησιμοποιήσει την TC στο μέγιστο δυνατό fetch bandwidth (3 μIPC)

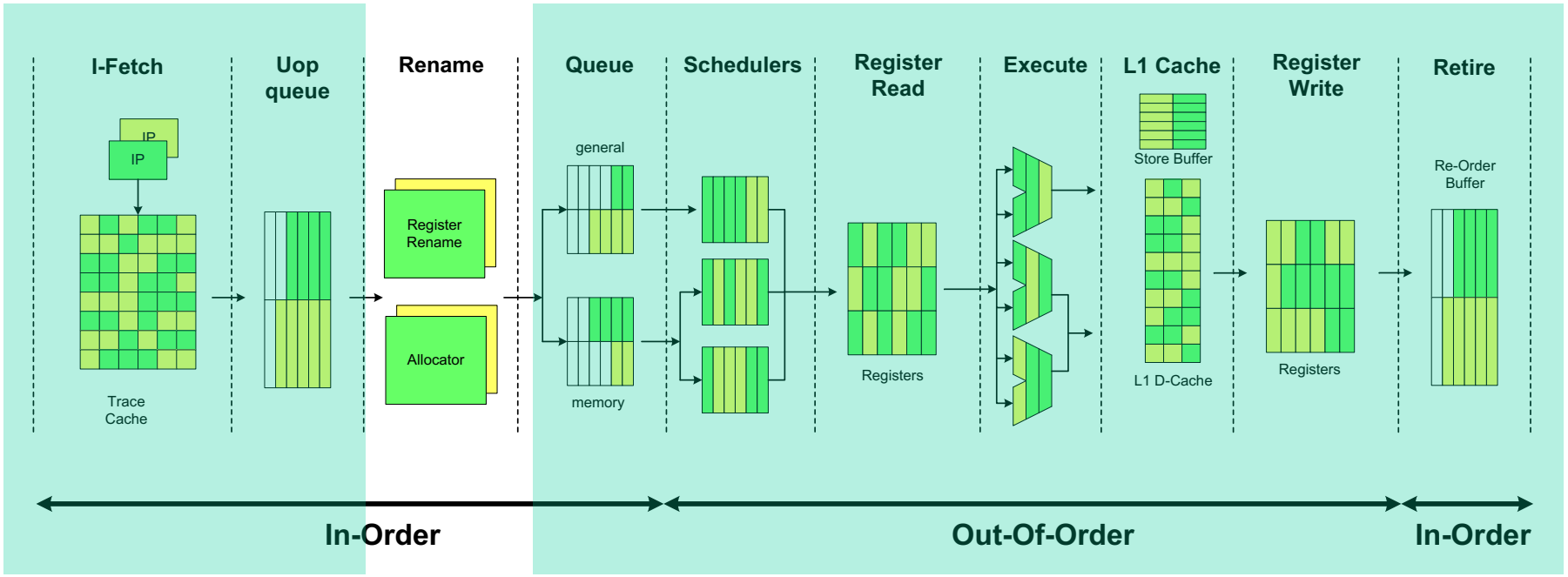


Pentium 4 w/ Hyper-Threading: Front End – trace cache miss



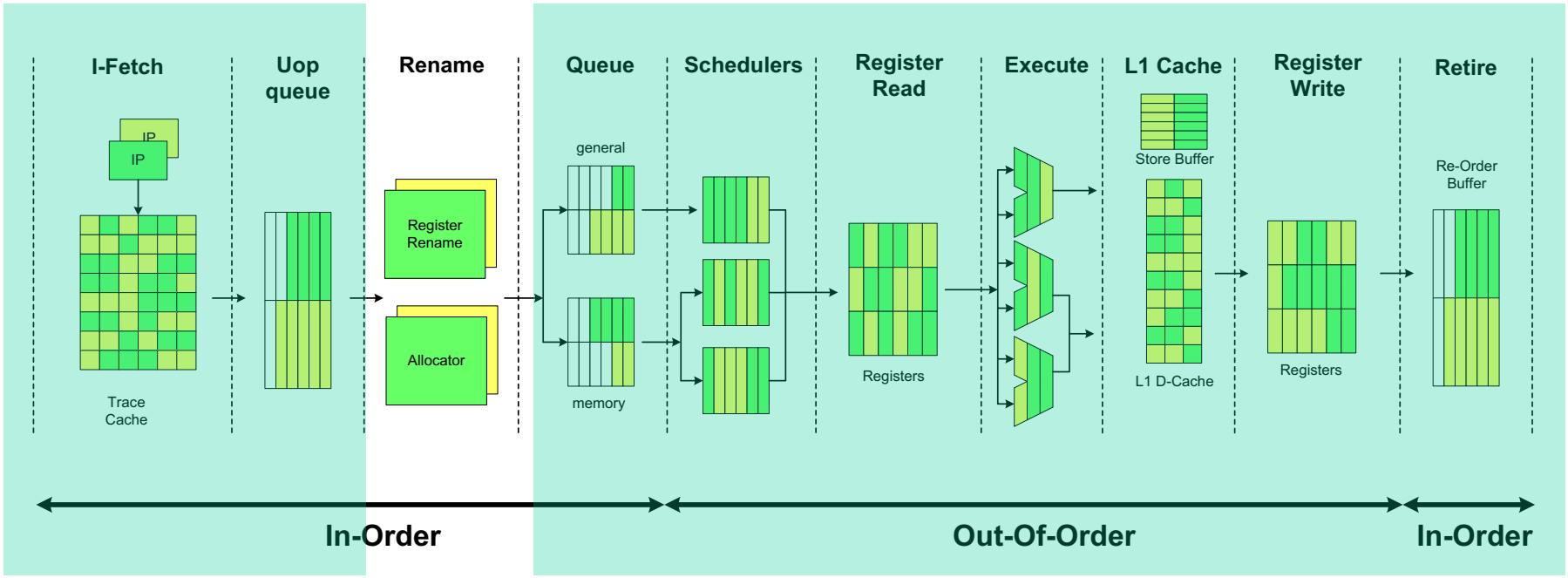
- L2 cache
  - η πρόσβαση γίνεται με FCFS τρόπο
- Decode
  - σε ταυτόχρονη ζήτηση από τους 2 LPs, η πρόσβαση εναλλάσσεται, αλλά με πιο coarse-grained τρόπο (δηλ. όχι κύκλο-ανά-κύκλο, αλλά k κύκλους-ανά-k κύκλους)

Pentium 4 w/ Hyper-Threading: Execution engine



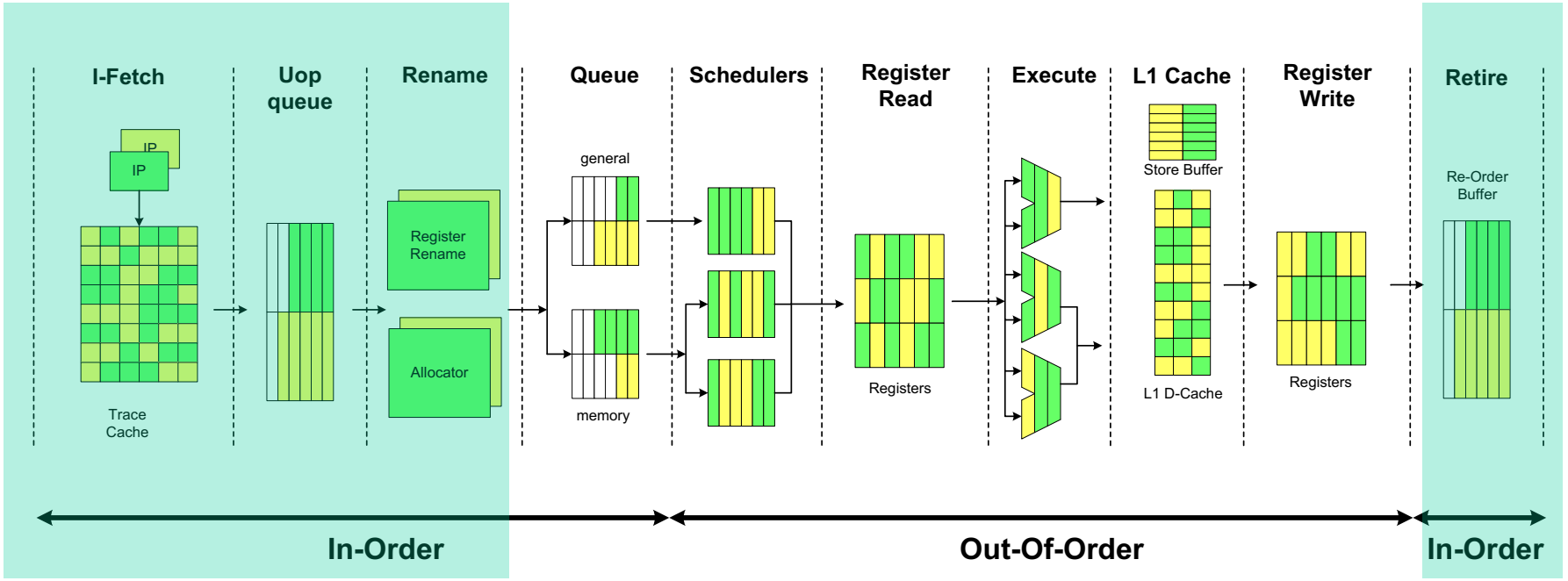
- Allocator: εκχωρεί entries σε κάθε LP
  - 63/126 ROB entries
  - 24/48 Load buffer entries
  - 12/24 Store buffer entries
  - 128/128 Integer physical registers
  - 128/128 FP physical registers
- Σε ταυτόχρονη ζήτηση από τους 2 LPs, η πρόσβαση εναλλάσσεται κύκλο-ανά-κύκλο
- stall-άρει έναν LP όταν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

Pentium 4 w/ Hyper-Threading: Execution engine



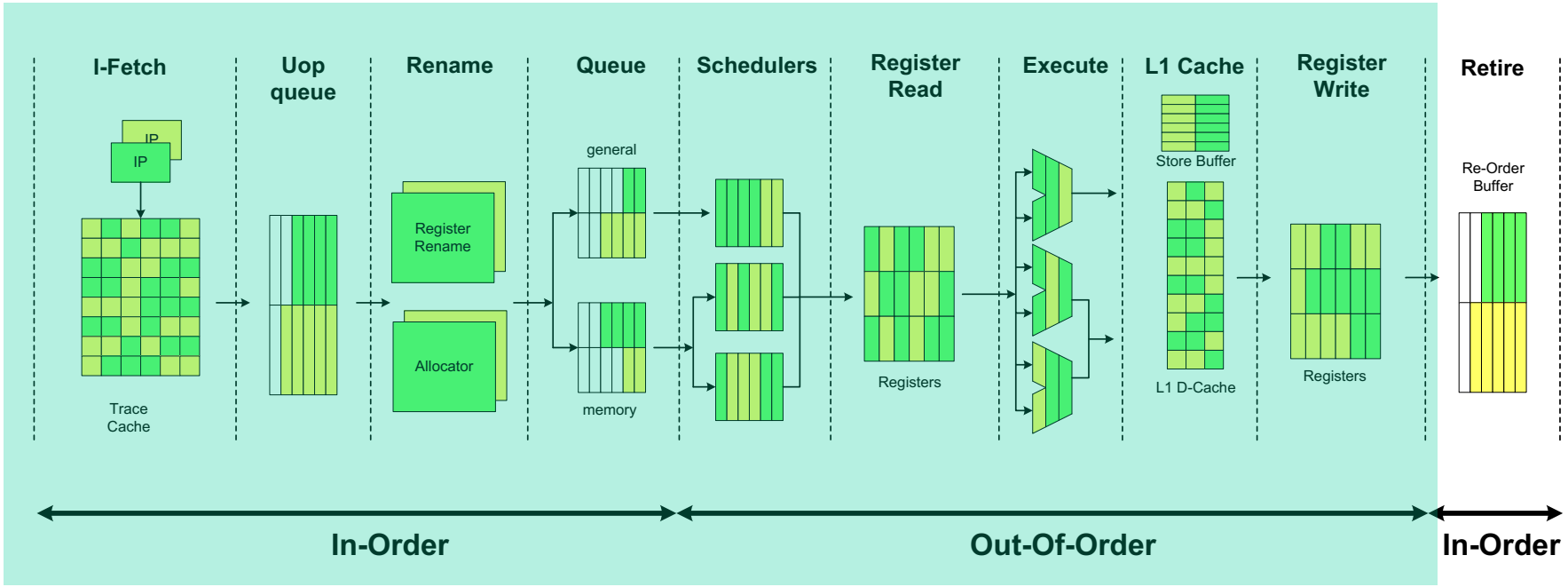
- Register renaming unit
  - επεκτείνει δυναμικά τους architectural registers απεικονίζοντάς τους σε ένα μεγαλύτερο σύνολο από physical registers
  - ξεχωριστό register map table για κάθε LP

Pentium 4 w/ Hyper-Threading: Execution engine



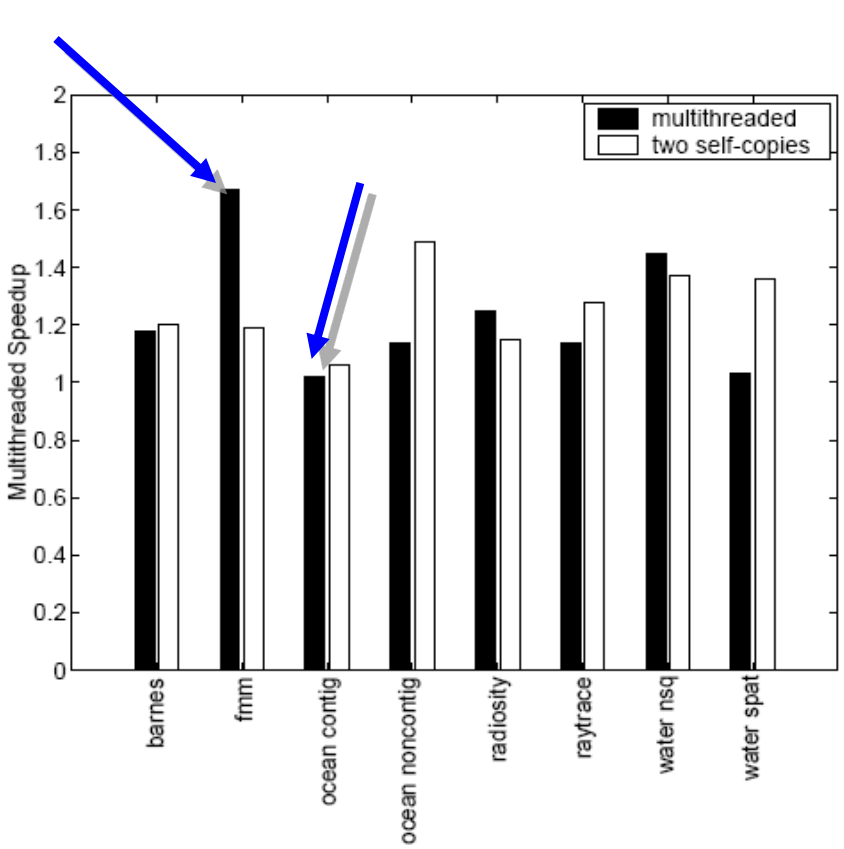
- Schedulers / Execution units
  - στην πραγματικότητα δε (χρειάζεται να) ξέρουν σε ποιον LP ανήκει η εντολή που εκτελούν
  - general+memory queues στέλνουν μικρο-εντολές στους δρομολογητές, με την πρόσβαση να εναλλάσσεται κύκλο-ανά-κύκλο ανάμεσα στους 2 LPs
  - 6 μIPC dispatch/execute bandwidth (→ 3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

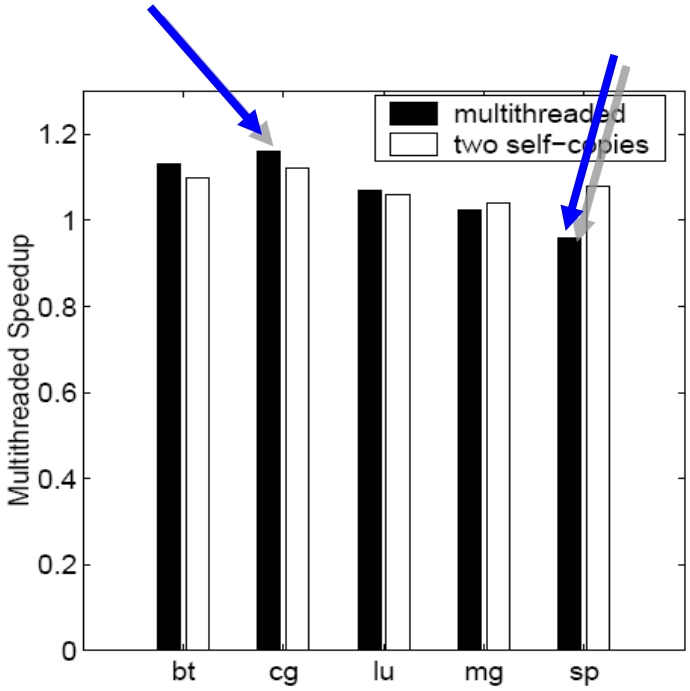
Pentium 4 w/ Hyper-Threading: Retirement



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

Multithreaded speedup

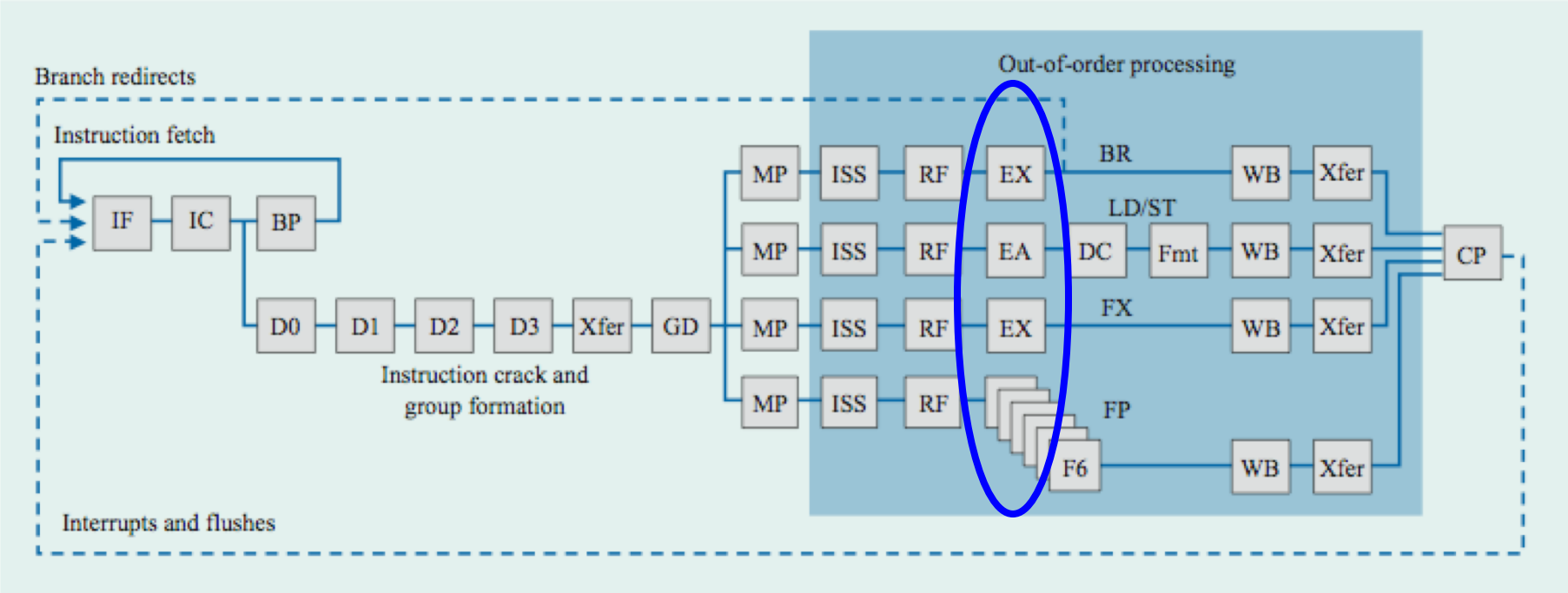




From: Tuck and Tullsen, "Initial Observations of the Simultaneous Multithreading Pentium 4 Processor", PACT 2003.

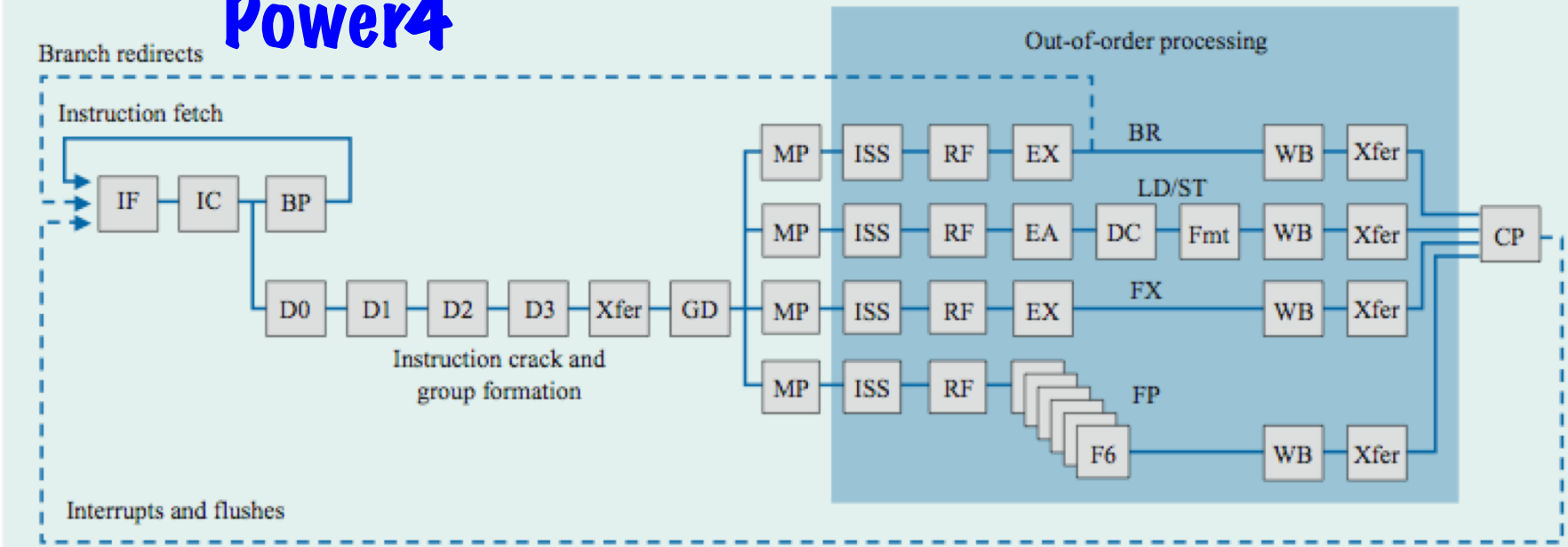
- SPLASH2 Benchmarks: 1.02 – 1.67
- NAS Parallel Benchmarks: 0.96 – 1.16

Case-study 2: Power5
Επέκταση του Power4 για υποστήριξη SMT (2004)

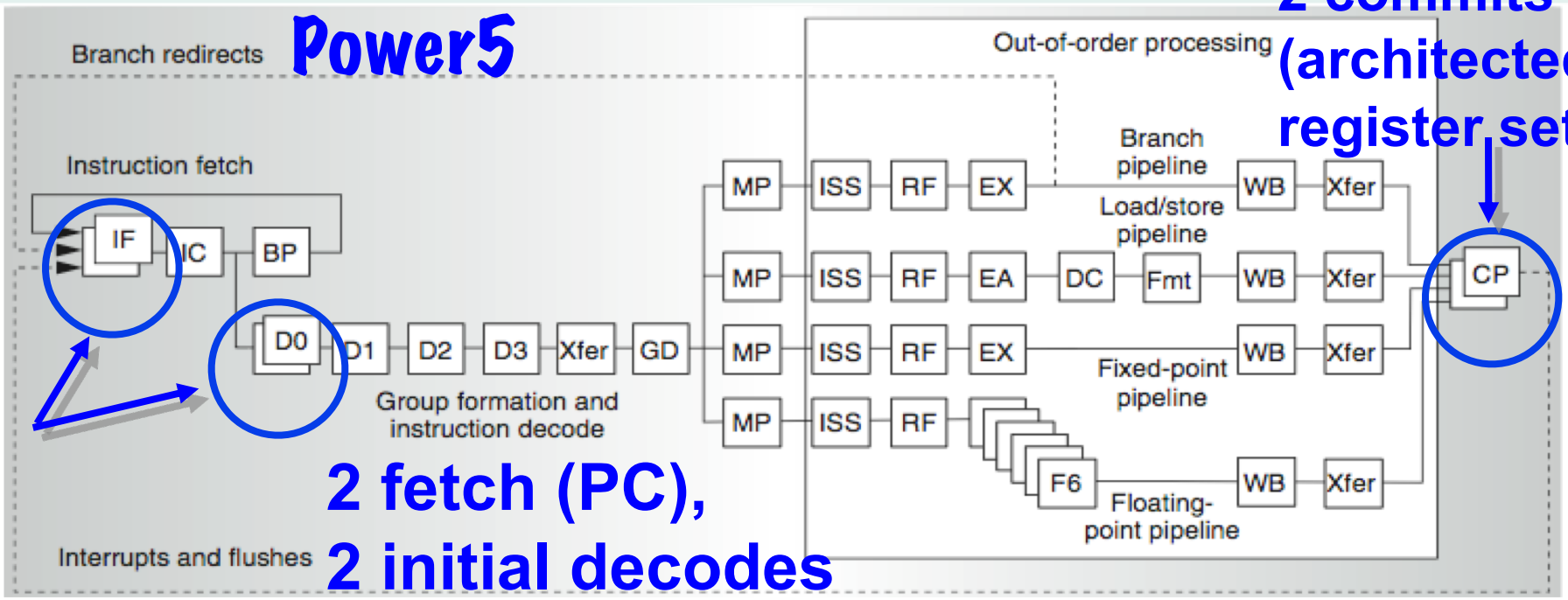


- Power4: Single-threaded «προκάτοχος» του Power5
- 8 execution units
  - 2 Float. Point, 2 Load/Store, 2 Fixed Point, 1 Branch, 1 Conditional Reg. unit
  - κάθε μία μπορεί να κάνει issue 1 εντολή ανά κύκλο
- Execution bandwidth: 8 operations ανά κύκλο
  - (1 fpadd + 1 fpmult) x 2FP + 1 load/store x 2 LD/ST + 1 integer x 2 FX

Power4



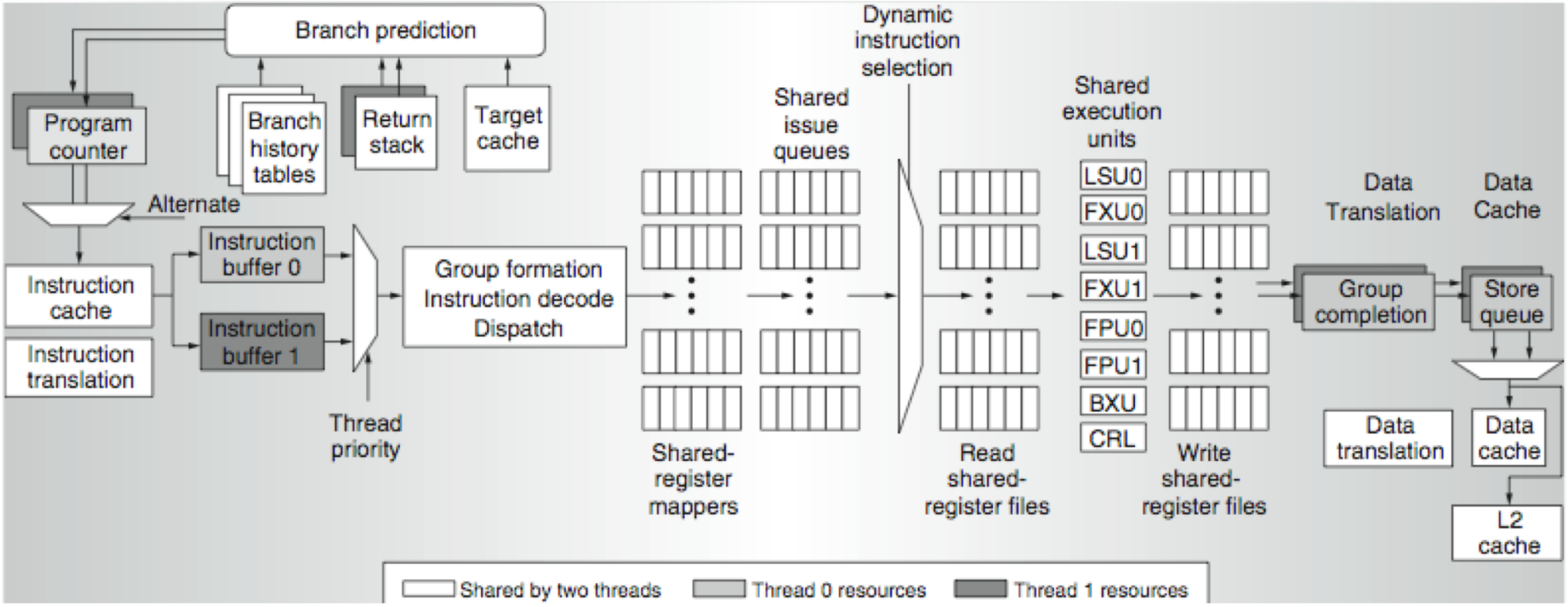
Power5



2 commits (architected register sets)

2 fetch (PC), 2 initial decodes

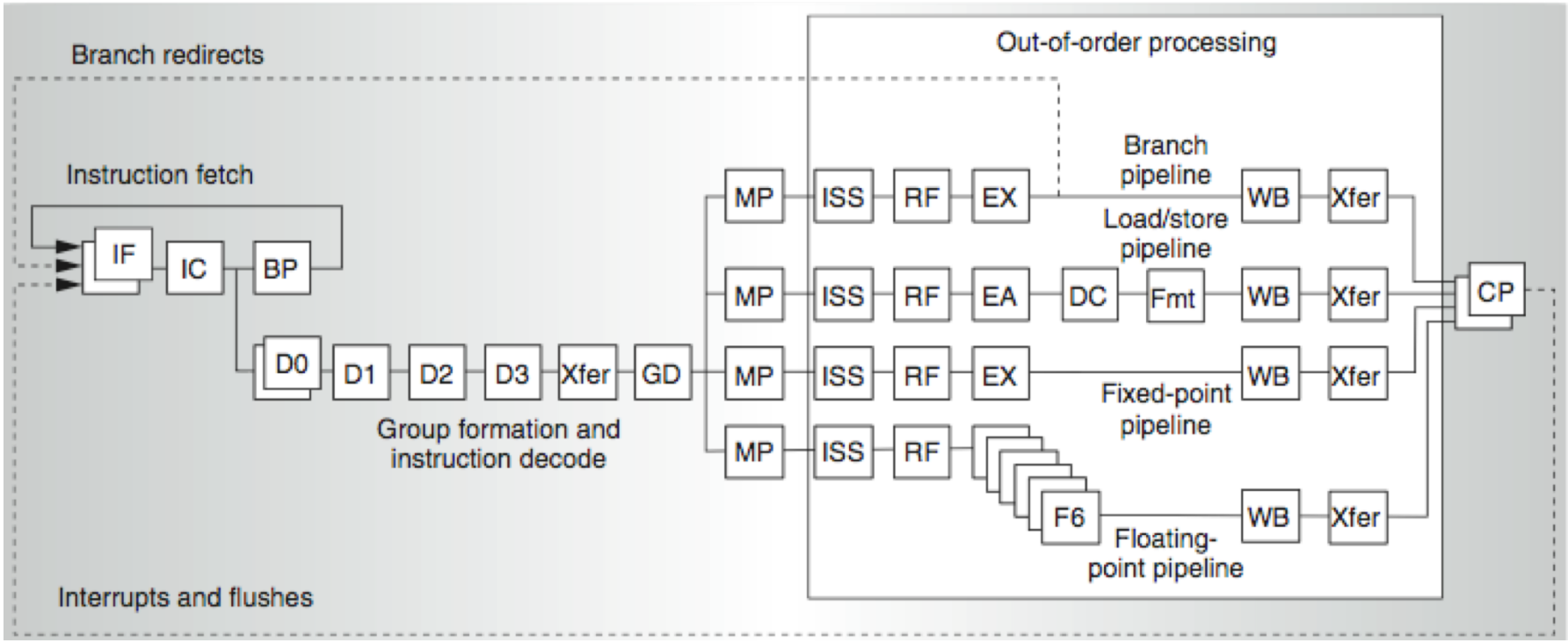
SMT resource management



Αλλαγές στον Power5 για να υποστηρίζεται το SMT

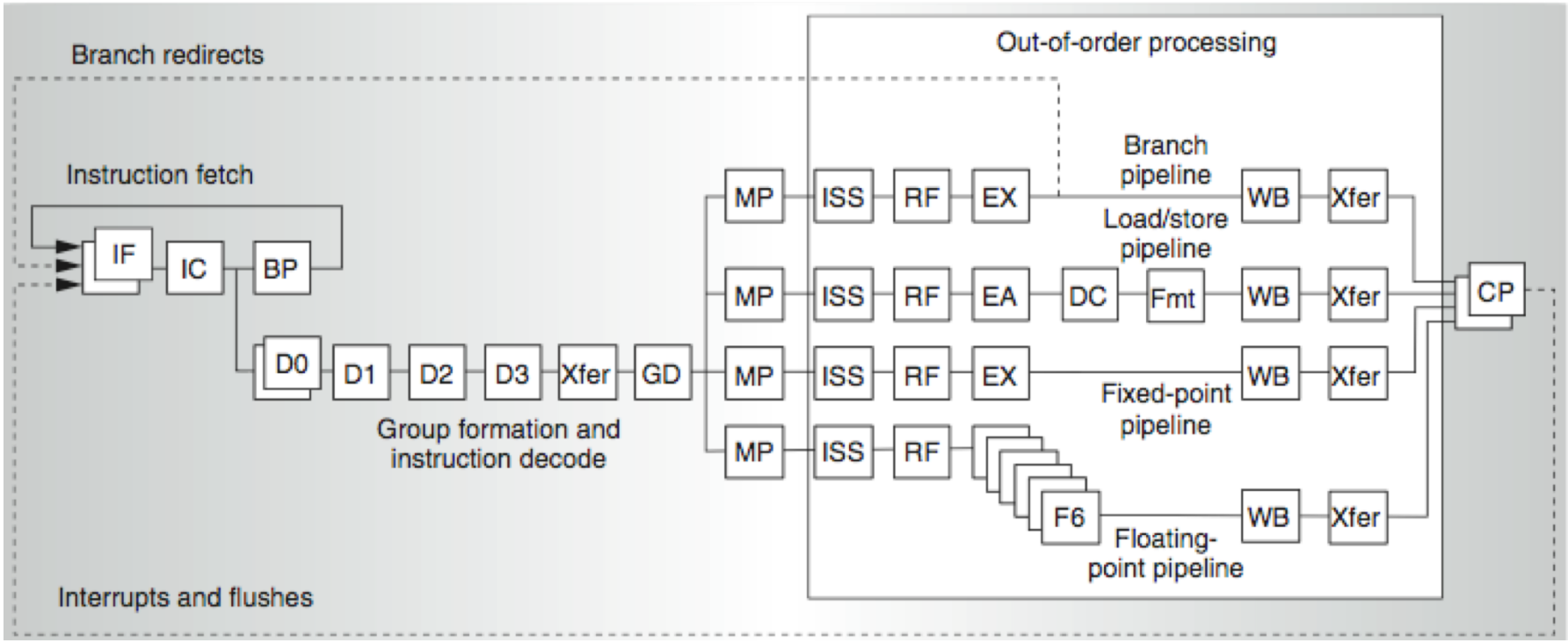
- Αύξηση συσχετιστικότητας της L1 instruction cache και των ITLBs
- Ξεχωριστές Load/Store queues για κάθε νήμα
- Αύξηση μεγέθους των L2 (1.92 vs. 1.44 MB) και L3 caches
- Ξεχωριστό instruction prefetch hardware και instruction buffers για κάθε νήμα
- Αύξηση των registers από 152 σε 240
- Αύξηση του μεγέθους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

Power 5 datapath



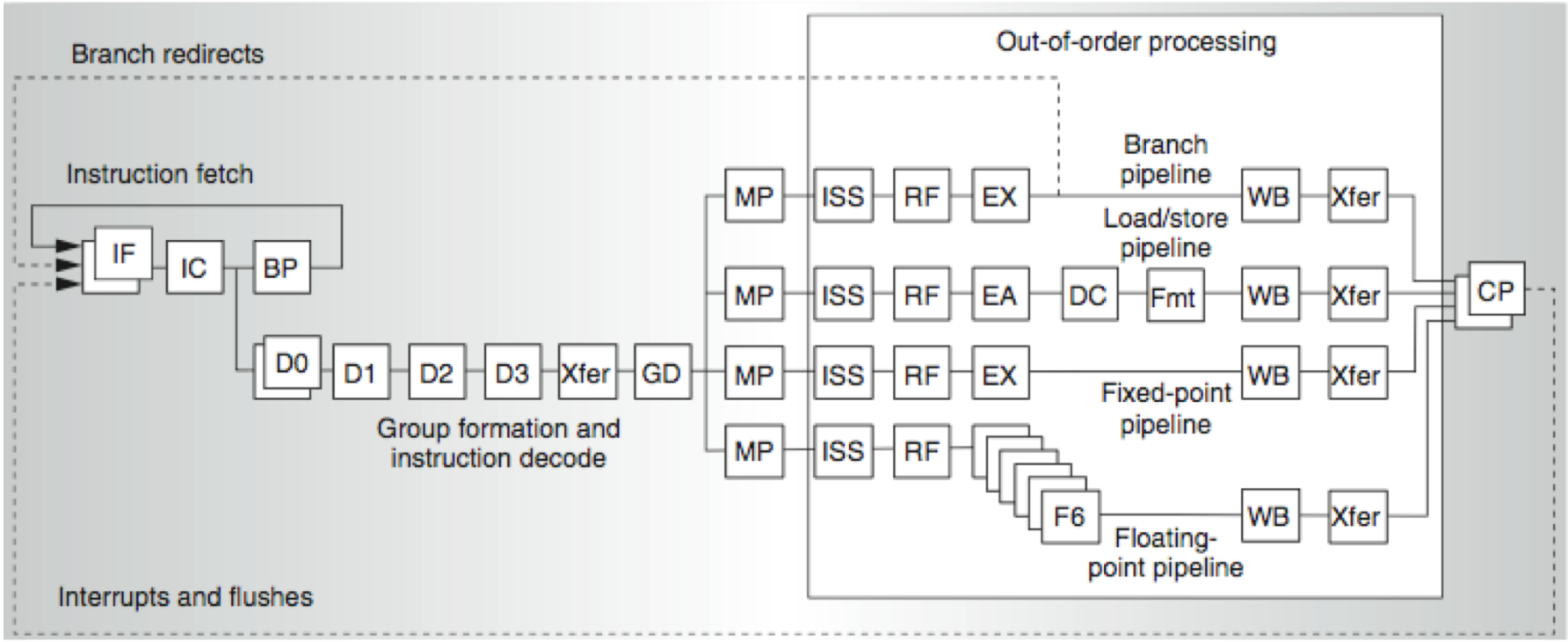
- Στο IF στάδιο η πρόσβαση εναλλάσσεται κύκλο-ανά-κύκλο ανάμεσα στα 2 threads
  - 2 instruction fetch address registers, 1 για κάθε νήμα
- Μπορούν να φορτωθούν 8 instructions σε κάθε κύκλο (στάδιο IC) από την I-Cache
  - σε έναν συγκεκριμένο κύκλο, οι εντολές που φορτώνονται προέρχονται όλες από το ίδιο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

Power 5 datapath



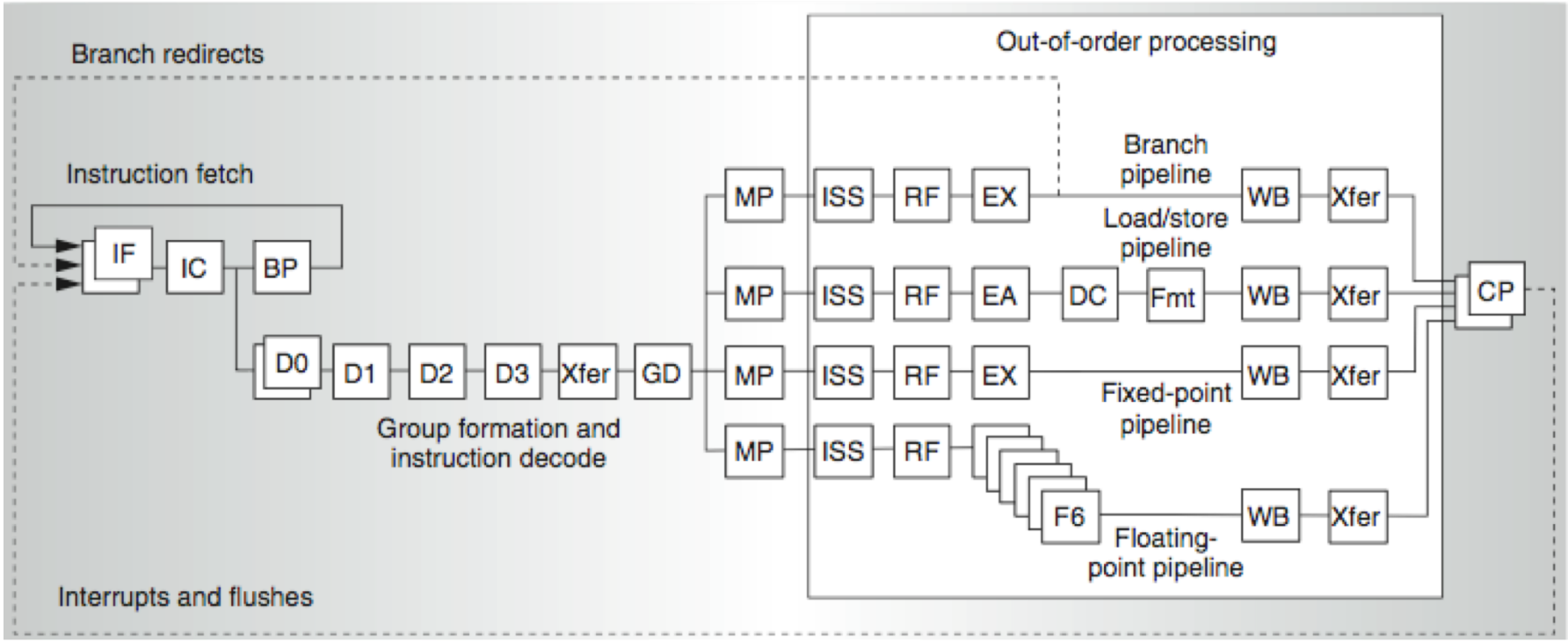
- Στα στάδια D1-D3, ανάλογα με την προτεραιότητα κάθε thread, ο επεξεργαστής διαλέγει εντολές από έναν από τους δύο instruction buffers και σχηματίζει ένα group
- Όλες οι εντολές σε ένα group προέρχονται από το ίδιο thread και αποκωδικοποιούνται παράλληλα
- Κάθε group μπορεί να περιέχει το πολύ 5 εντολές

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

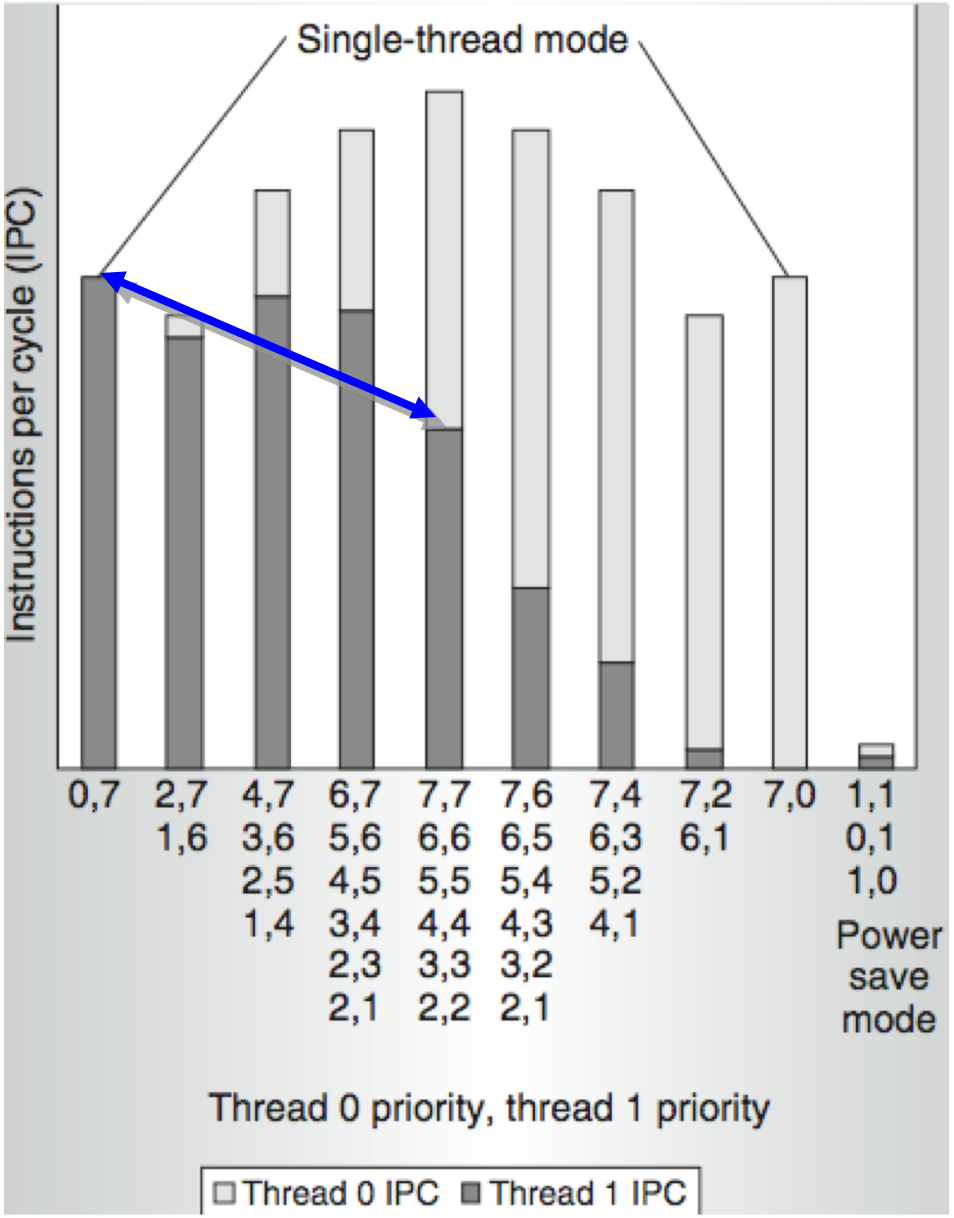
- Μηχανισμοί περιορισμού ενός πολύ απαιτητικού thread:
  - *μείωση προτεραιότητας του thread*, όταν διαπιστώνεται ότι τα GCT entries που χρησιμοποιεί ξεπερνούν ένα καθορισμένο όριο
  - όταν ένα thread έχει πολλά L2 misses, τότε οι μεταγενέστερες εξαρτώμενες εντολές του μπορούν να γεμίσουν τις issue queues, 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 προτεραιτότητες, εκτελούνται πιο αργά απ'ό,τι αν κάθε thread είχε διαθέσιμα όλα τα resources του επεξεργαστή (single-thread mode)



"Large" vs. "Small" Cores

Large Core

- Out-of-order
- Wide fetch e.g. 4-wide
- Deeper pipeline
- Aggressive branch predictor (e.g. hybrid)
- Multiple functional units
- Trace cache
- Memory dependence speculation

Small Core

- In-order
- Narrow Fetch e.g. 2-wide
- Shallow pipeline
- Simple branch predictor (e.g. Gshare)
- Few functional units

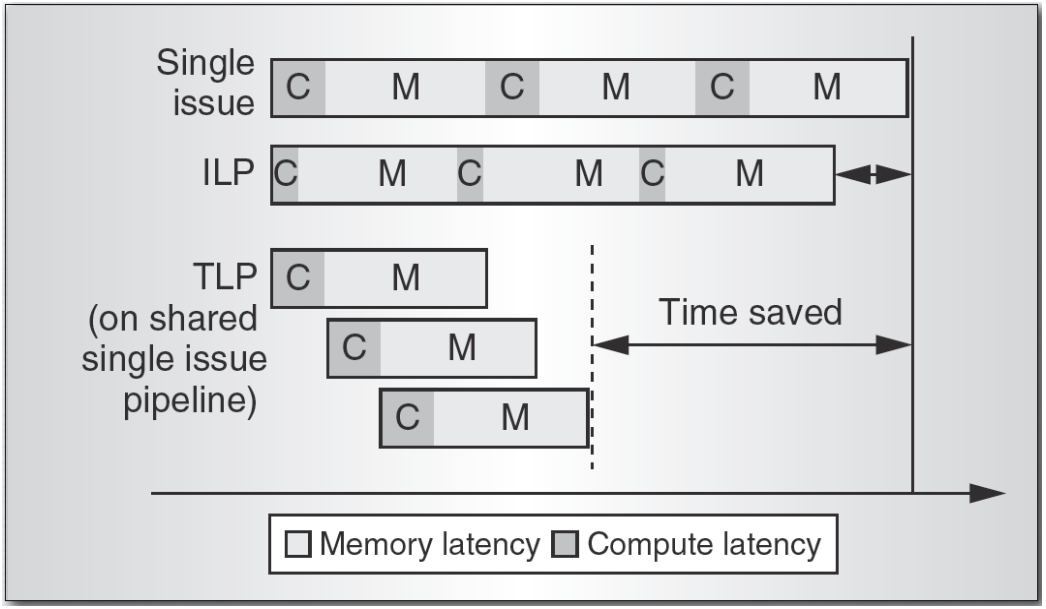
Large Cores are power inefficient:
e.g., 2x performance for 4x area (power)

Large vs. Small Cores

- Grochowski et al., "*Best of both Latency and Throughput*," ICCD 2004.

	Large core	Small core
Microarchitecture	Out-of-order, 128-256 entry ROB	In-order
Width	3-4	1
Pipeline depth	20-30	5
Normalized performance	5-8x	1x
Normalized power	20-50x	1x
Normalized energy/instruction	4-6x	1x

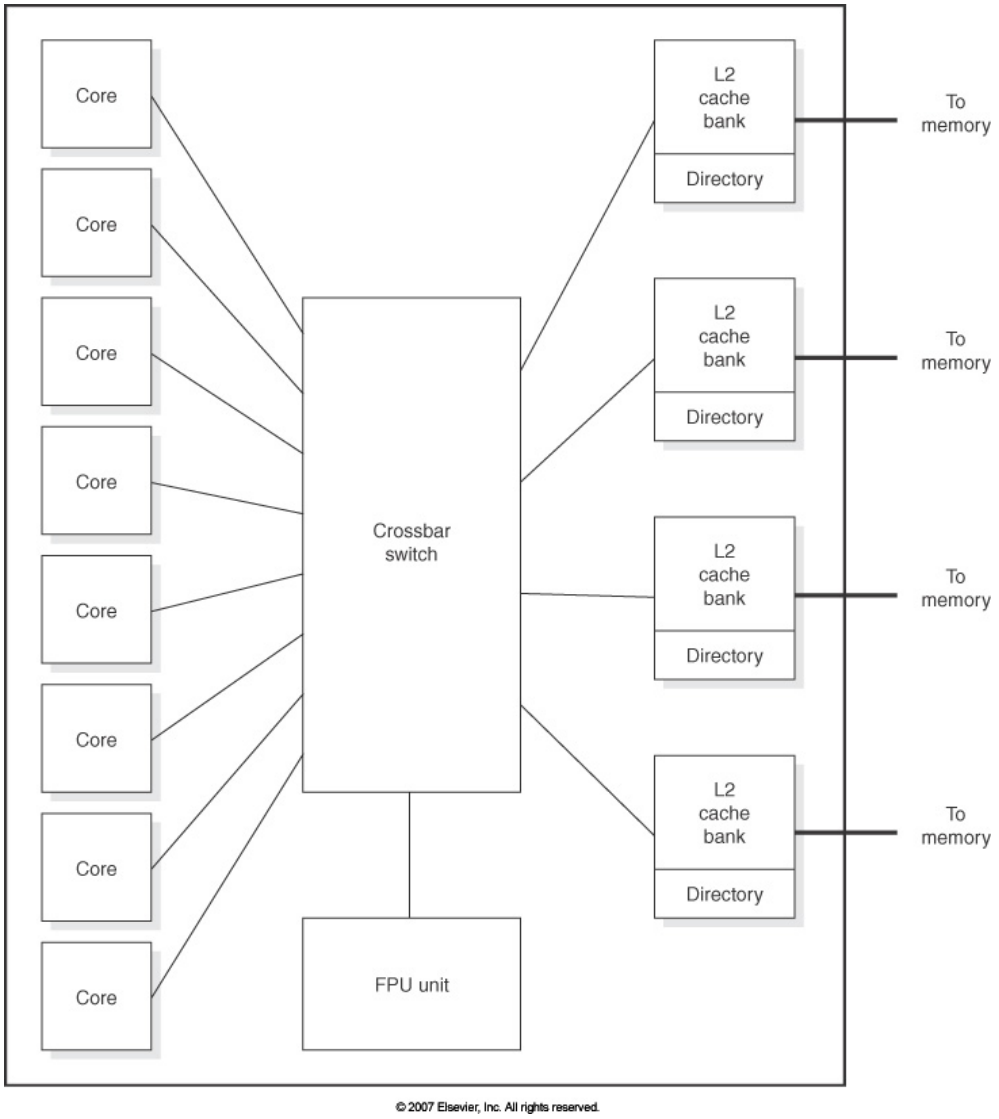
Case-study 3: UltraSPARC T1 (“Niagara”) (2005)



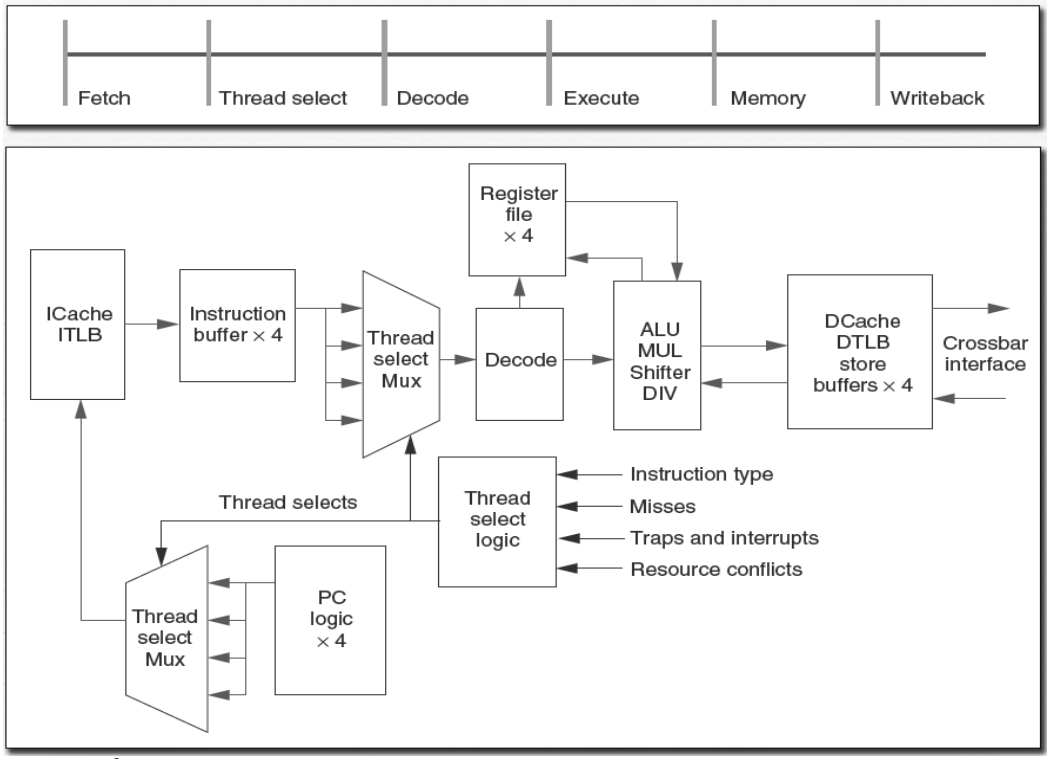
- Συμπεριφορά επεξεργαστών βελτιστοποιημένων για TLP και ILP σε server workloads:
  - server workloads:
    - » υψηλός TLP (μεγάλος αριθμός παράλληλων client requests)
    - » χαμηλός ILP (υψηλά miss rates, πολλά unpredictable branches, συχνές load-load εξαρτήσεις)
    - » το memory access time κυριαρχεί 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

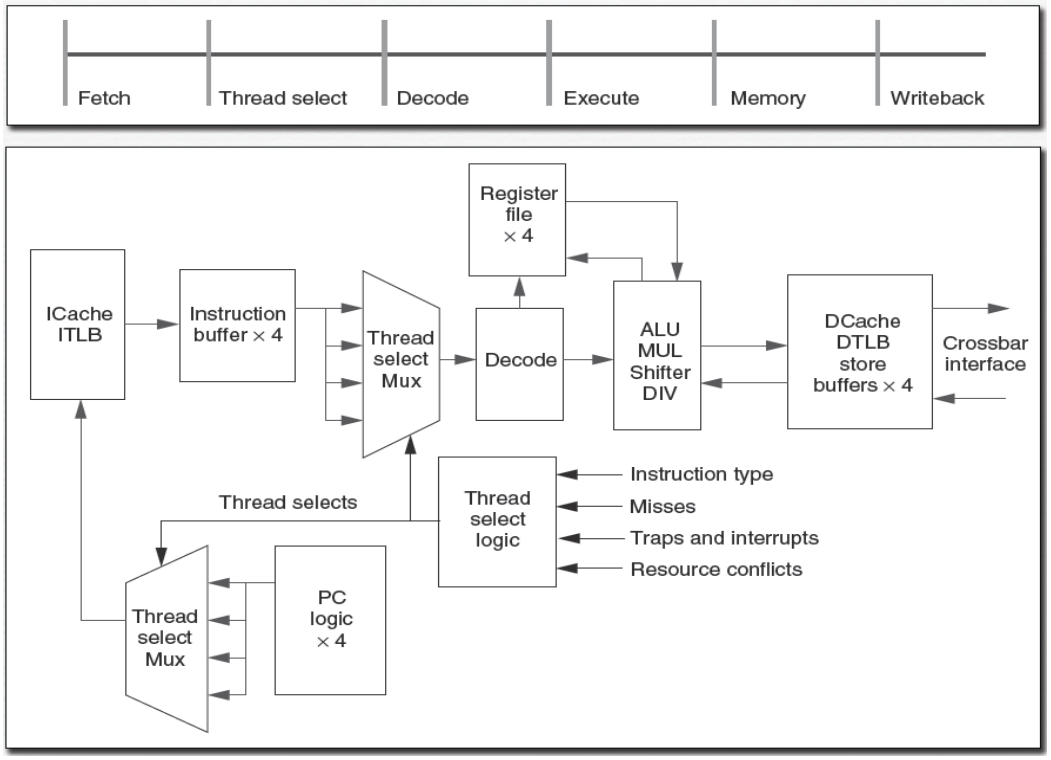


UltraSPARC T1 pipeline



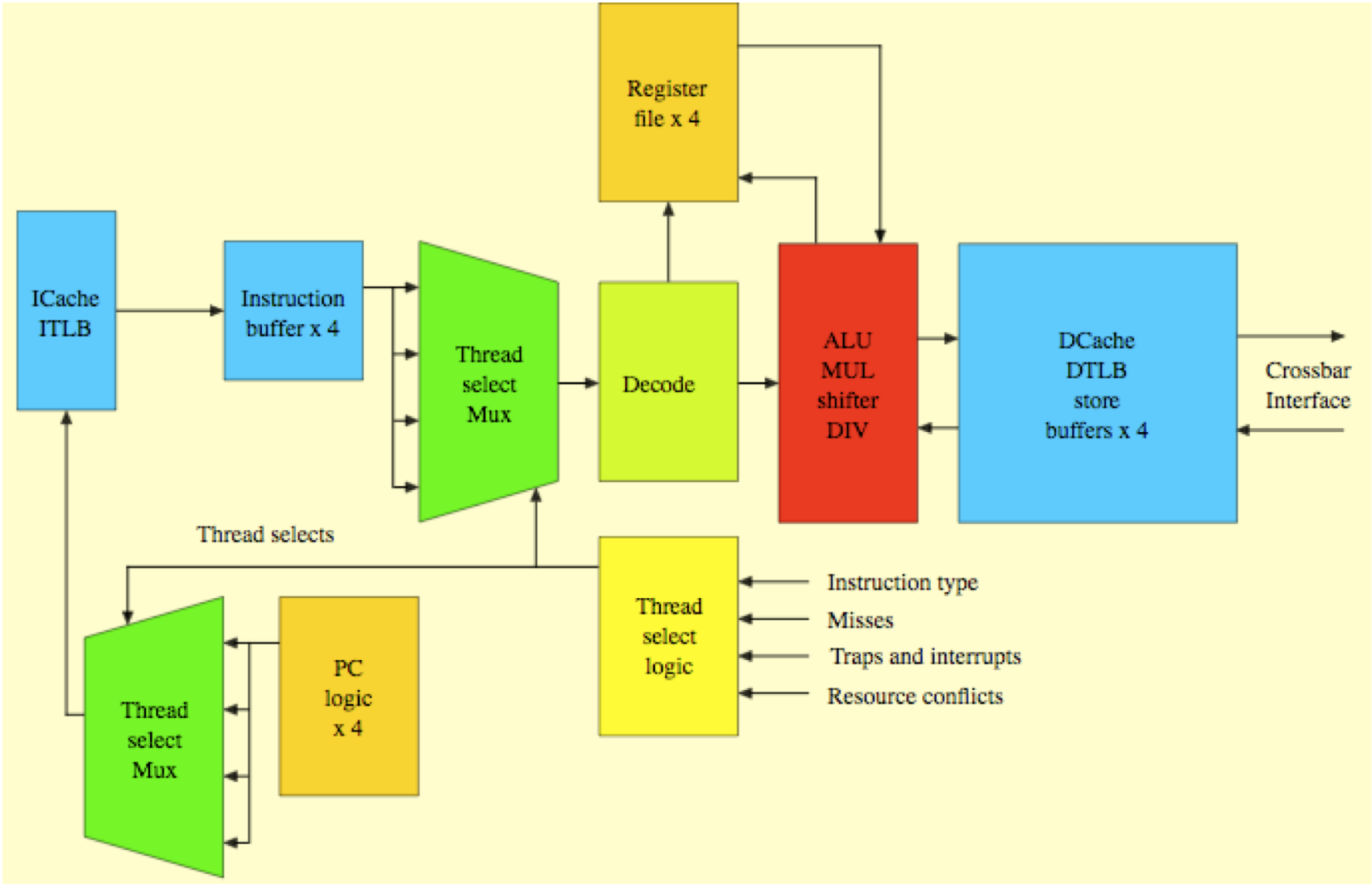
- Όπως το κλασικό 5-stage pipeline + thread select stage
- Fetch stage: ο thread select mux επιλέγει ποιος από τους 4 PCs θα πρέπει να προσπελάσει την ICache και το ITLB
- Thread select stage: αποφασίζει σε κάθε κύκλο ποιος από τους 4 instruction buffers θα τροφοδοτήσει με εντολές τα επόμενα στάδια
  - 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

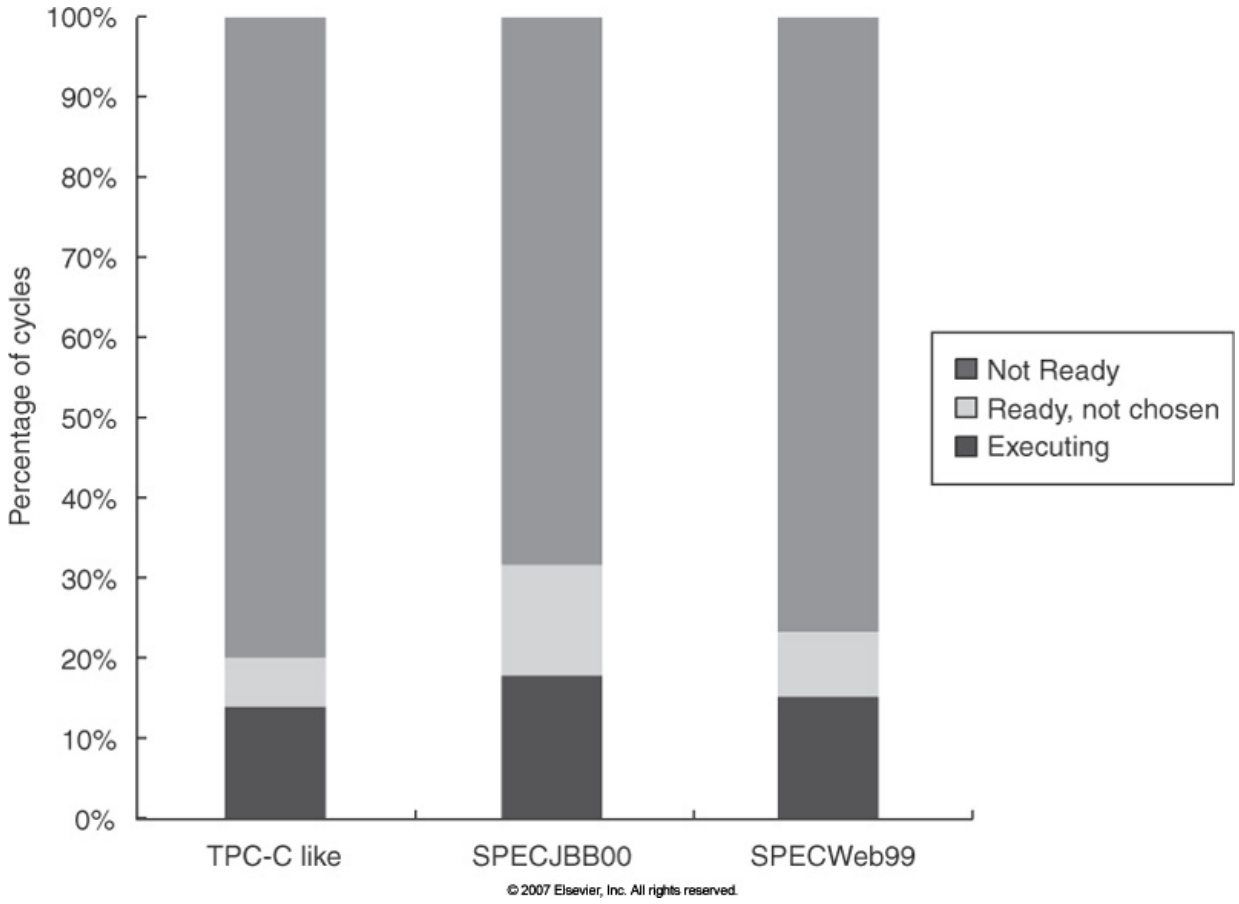


UltraSPARC T1 performance

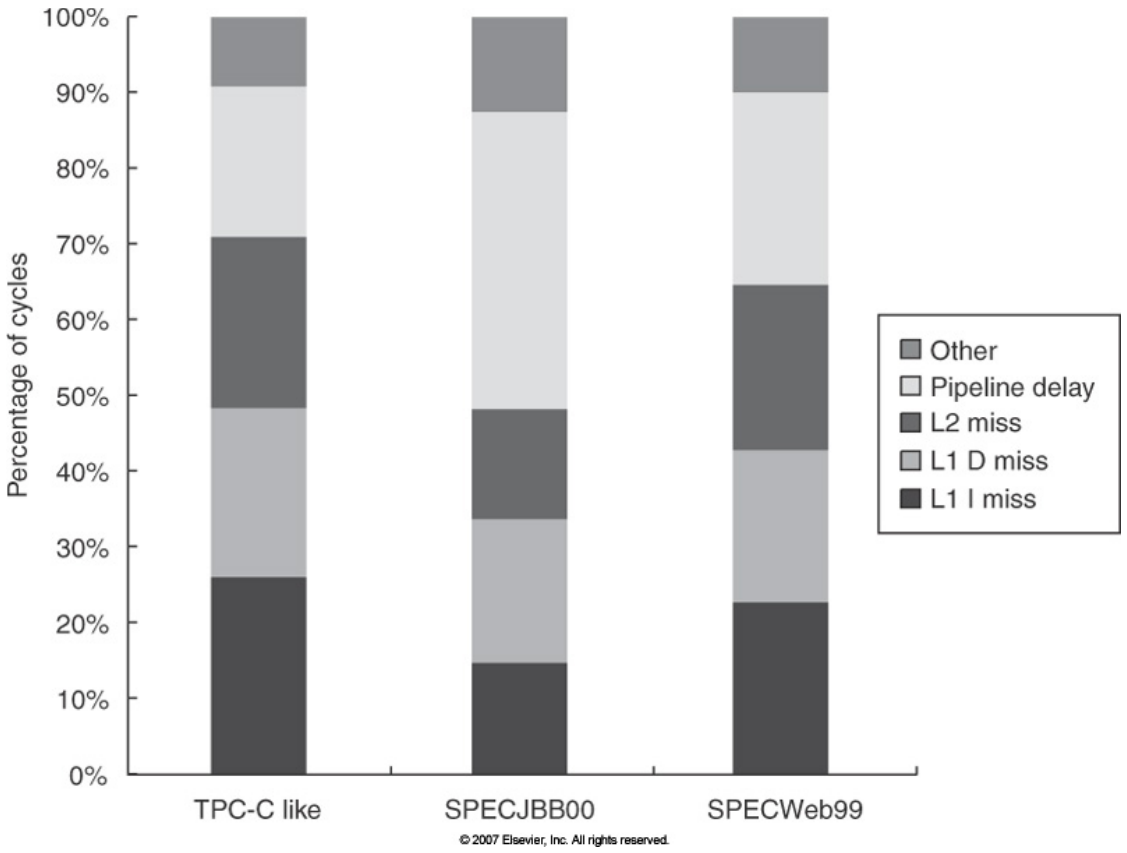
Benchmark	Per-thread CPI	Per core CPI	Effective CPI for eight cores	Effective IPC for eight cores
TPC-C	7.2	1.8	0.225	4.4
SPECJBB	5.6	1.40	0.175	5.7
SPECWeb99	6.6	1.65	0.206	4.8

- 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



Λόγοι για τη μη διαθεσιμότητα ενός thread



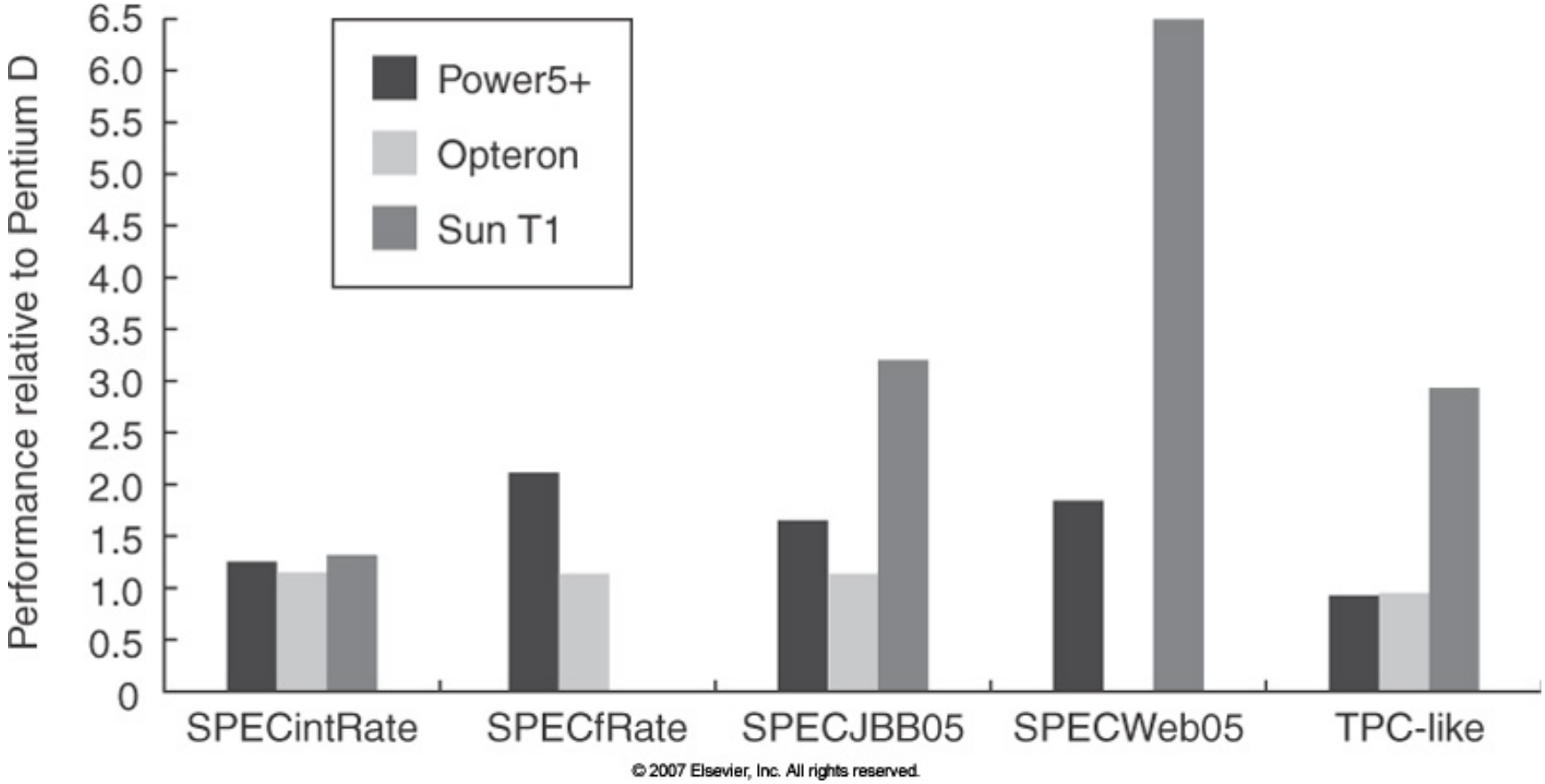
- Pipeline delay: long-latency εντολές όπως branches, loads, fp, int mult/div

«Crash-test» multicore επεξεργαστών (~2005)

Characteristic	SUN T1	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^2)	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

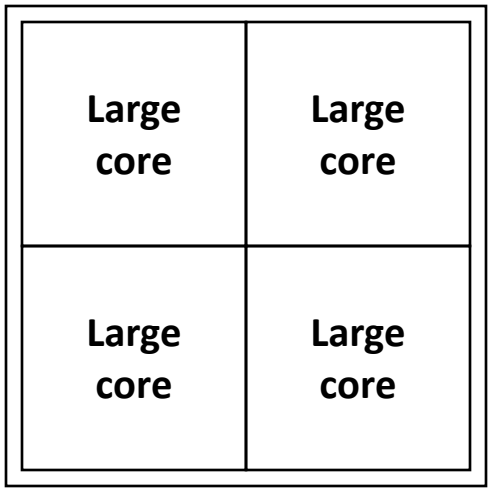
«Crash-test» multicore επεξεργαστών



«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
Mutlithreading	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)

Tile-Large Approach



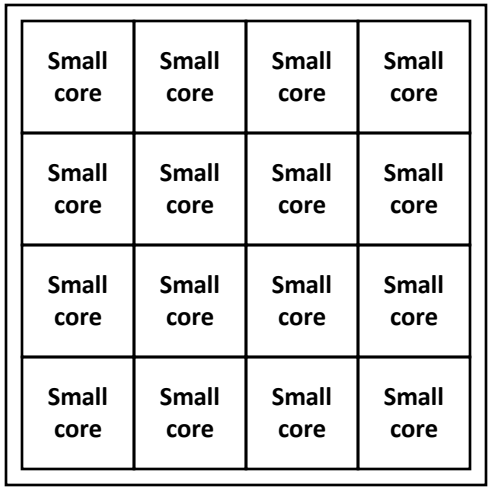
“Tile-Large”

- Tile a few large cores
- IBM Power 5, AMD Barcelona, Intel Core2Quad, Intel Nehalem

+ High performance on single thread, serial code sections

- Low throughput on parallel program portions

Tile-Small Approach



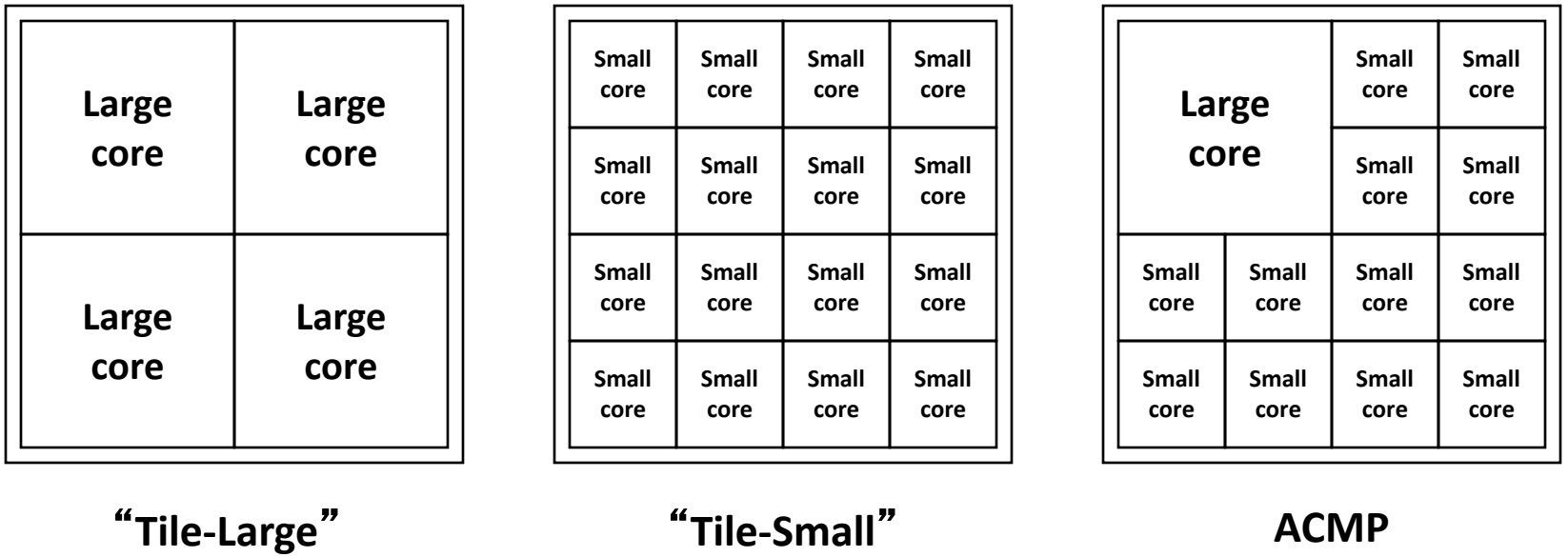
"Tile-Small"

- Tile many small cores
- Sun Niagara, Intel Larrabee, Tilera TILE (tile ultra-small)

\+ High throughput on the parallel part (16 units)

\- Low performance on the serial part, single thread (1 unit)

Asymmetric Chip Multiprocessor (ACMP)

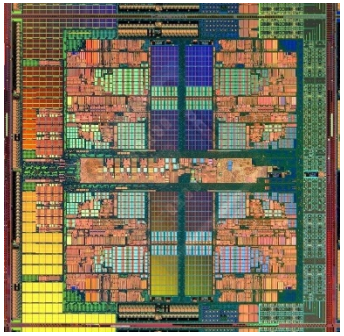


- Provide one large core and many small cores
- ARM big.LITTLE

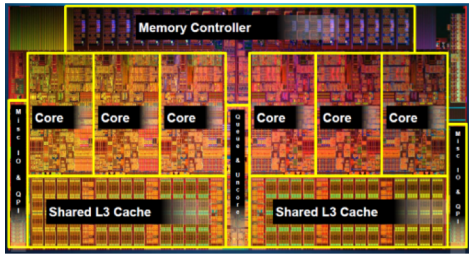
+ Accelerate serial part using the large core
+ Execute parallel part on small cores and large core for high throughput

Today: Many Cores on Chip

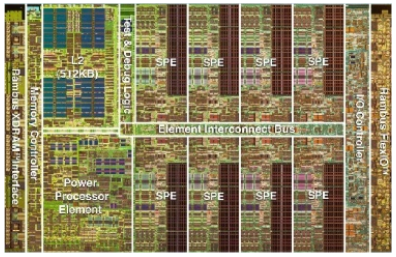
- Simpler and lower power than a single large core
- Large scale parallelism on chip



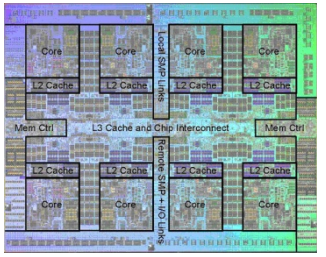
AMD Barcelona
4 cores



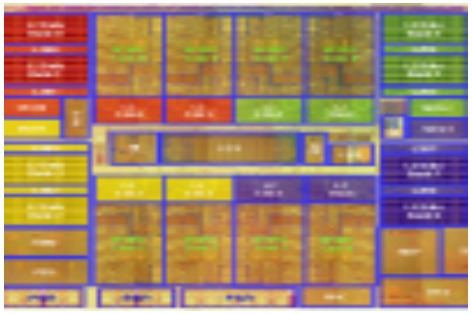
Intel Core i7
8 cores



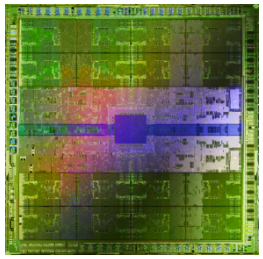
IBM Cell BE
8+1 cores



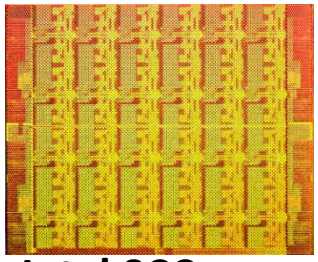
IBM POWER7
8 cores



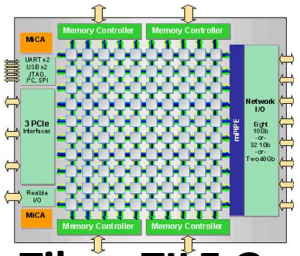
Sun Niagara II
8 cores



Nvidia Fermi
448 "cores"



Intel SCC
48 cores, networked



Tilera 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 3rd 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 4th 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)

Chips Today (2015-2017)

- HotChips 2015 (HC27 http://www.hotchips.org/archives/hc27/)
  - ARM Mali-T880GPU (link)
  - Intel's Knight Landing: 2nd Generation Xeon Phi Processor (link)
  - AMD's Carrizo APU (link)
  - Oracle's Sonoma Processor (link)
- HotChips 2016 (HC28 http://www.hotchips.org/archives/hc28/)
  - ARM v8-A (link)
  - Samsung Exynos-M1 (link)
  - NVIDIA Tegra SoC (link)
  - Oracle SPARC M7 (link)
  - Intel Skylake (link)
  - POWER9 (link)
- HotChips 2017 (HC29 http://www.hotchips.org/archives/hc29/)
  - Knights Mill: Intel Xeon Phi Processor for Machine Learning (link)
  - Celerity: An Open Source RISC-V Tiered Accelerator Fabric (link)
  - Graph Streaming Processor (GSP) A Next-Generation Computing Arch. (link)
  - The New Intel Xeon Processor Scalable Family (link)
  - And many more..

Chips Today (2018-)

- HotChips 2018 (HC30 https://www.hotchips.org/archives/2010s/hc30/)
  - Samsung M3 processor (link)
  - BROOM open-source OoO processor (link)
  - NVIDIA Xavier SoC (link)
  - ARM ML processor (link)
  - IBM Power9 Scale Up processor (link)
  - Xilinx DNN processor (link)
  - Vector Engine Processor of NEC's Aurora (link)
  - And many more…