Introduction to Performance Analysis on Intel® Core™ 2 Duo Processors

Software and Solutions Group

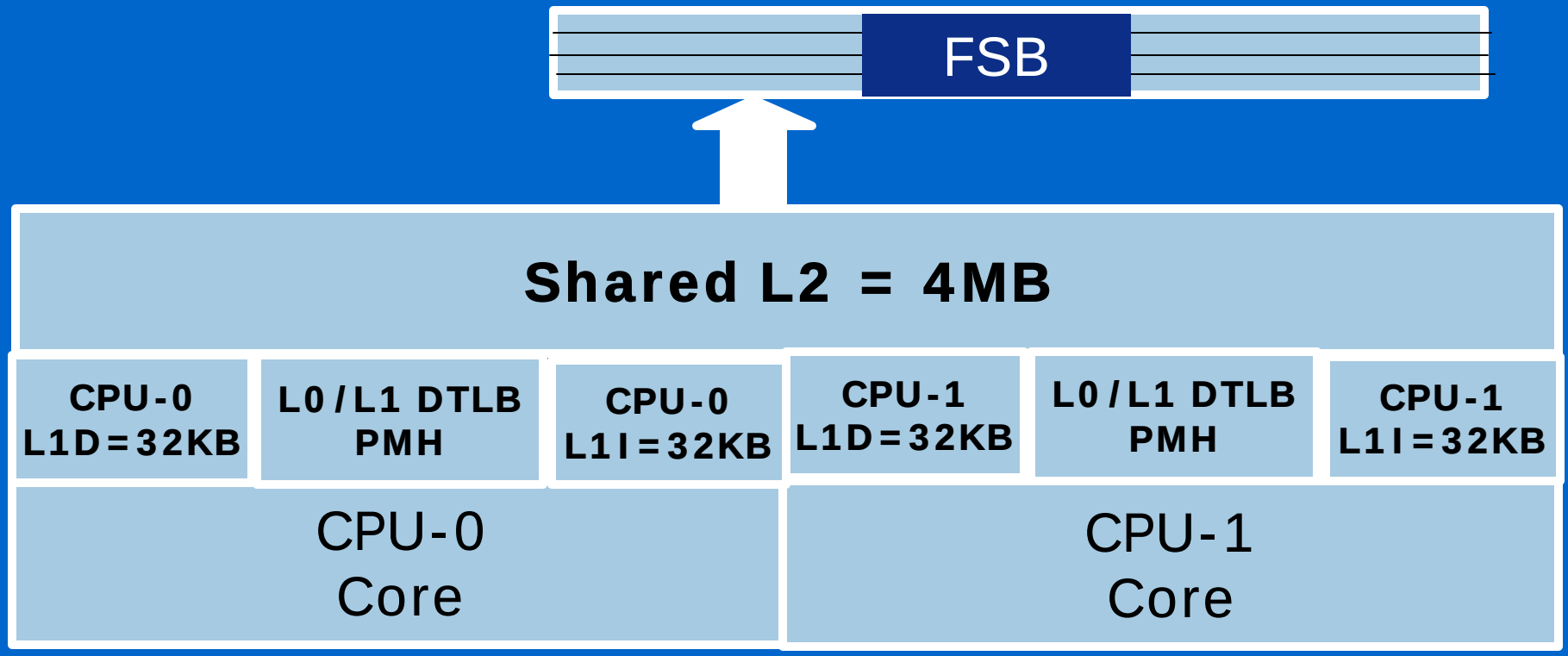
David Levinthal, Sr SW Engineer

Nov 30, 2006

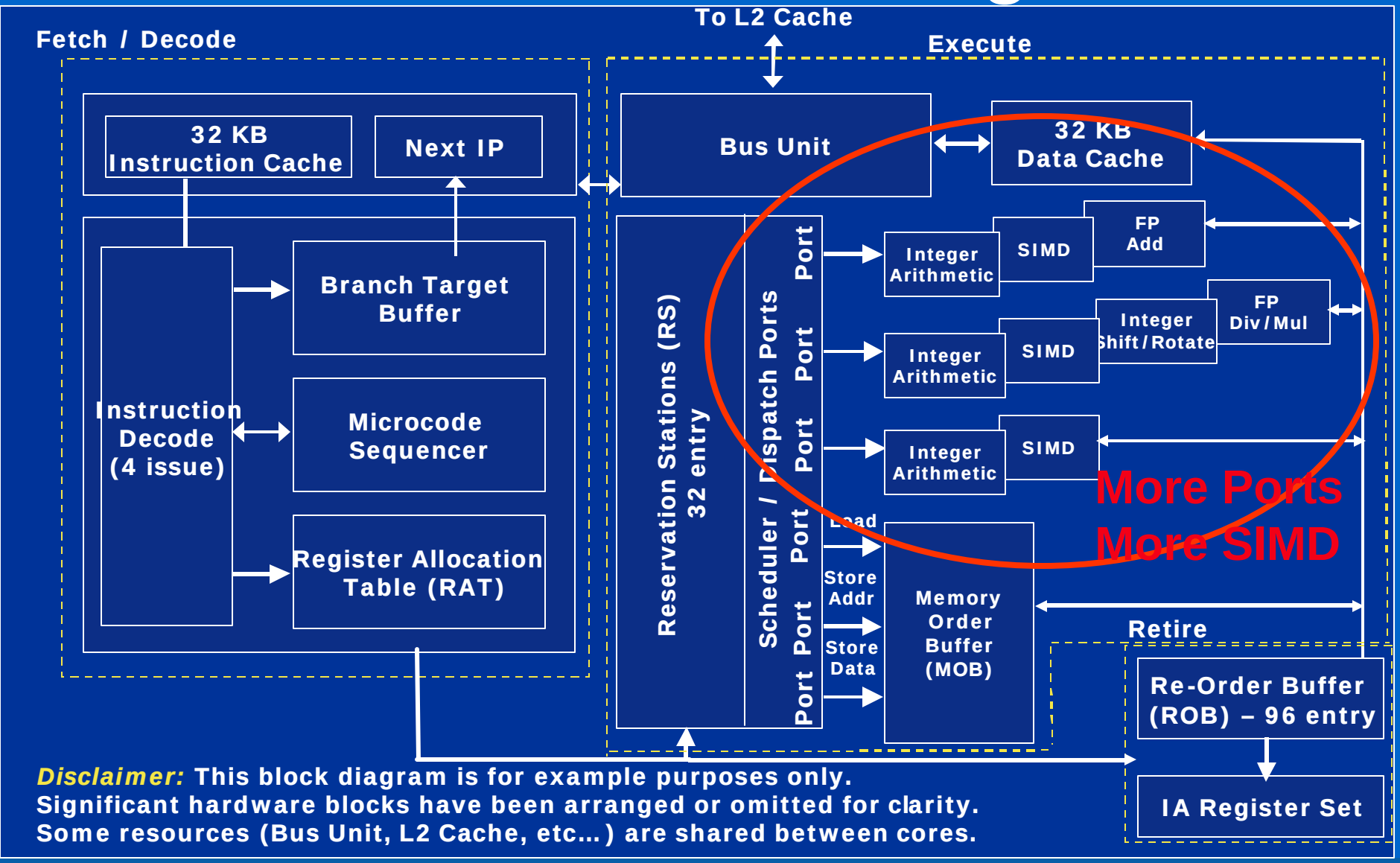
Agenda

- Tuning Overview
- Processor Overview
- Methodology

Next Generation Micro Architecture
Intel® Core™2 Duo Processor



Architecture Block Diagram



Disclaimer: **This block diagram is for example purposes only.**
Significant hardware blocks have been arranged or omitted for clarity.
Some resources (Bus Unit, L2 Cache, etc…) are shared between cores.

Four Component HW Prefetcher

- L1 Cache Prefetch (first in Intel® Core™ Duo Processor)
  - DCU or Streaming prefetcher
    - DCU = Data Cache Unit
  - IP prefetch
    - Repeated stride load at frequently executed IP
- L2 Prefetch (similar to Pentium® 4 processor)

A Few Basic Events

EVENT	P	Description	EVENT	P	Description
CPU_CLK_UNHALTED			BUS_DRDY_CLOCKS.ALL_AGENTS		all busy bus cycles
INST_RETIRED.ANY_P	P		BUS_DRDY_CLOCKS.THIS_AGENT		all busy bus cycles due to writes
INST_RETIRED.LOADS			MEM_LOAD_RETIRED.L2_LINE_MISS	P	L2 demand misses
INST_RETIRED.STORES			MMX2_PRE_MISS.T1		SW prefetch to L1 inst
BUS_TRANS_ANY		all bus transactions	MMX2_PRE_MISS.T2		SW prefetch to L2 inst
BUS_TRANS_MEM		bus trans to memory	MMX2_PRE_MISS.STORES		Non Temporal Stores executed
BUS_TRANS_BURST		whole $lines to mem	L2_LINES_IN.SELF.DEMAND		L2$lines in for rfo, load, sw prefetch
BUS_TRANS_BRD		whole line reads from mem	L2_LINES_IN.SELF.PREFETCH		L2$lines in for hw prefetch
BUS_TRANS_WB		writebacks (no NT writes)	L2_LINES_OUT.SELF.DEMAND		demanded L2$Lines evicted
BUS_TRANS_RFO		$lines in for RFO (no HW pref)	L2_LINES_OUT.SELF.PREFETCH		HW prefetch L2$lines evicted

Memory BW = 64*Bus_Trans_Mem*freq/Cpu_Clk_Unhalted

Methodology Overview

- The traditional view of performance tuning on X86 processors has focused on instruction retirement
- The OOO engine has always been viewed as an impenetrable and incomprehensible beast
- This is perhaps not the best perspective

Methodology Overview

- Methodology developed by studying FP loop dominated apps
  - Anything that is applicable beyond this limited range is an unintended benefit
  - But if you can't do these…you're toast
- This style of optimization has 2 components
  - Minimizing instruction count
    - A sort of "tree height" minimization
  - Minimizing deviations from ideal execution
    - Generically thought of as "stall cycles"
  - Treating both equally is critical

Stalls, Execution Imperfection and Performance Analysis

- Stall cycles are used to indicate less than perfect execution
  - An architectural decomposition of “stalls” can be used to guide the selection of architectural events
  - The IP correlation of “stalls” and arch events then guides the optimization effort
- Stalls have 4 basic components in an OOO Engine
  - Execution stalls
    - (Waiting on input/Scoreboard, L2 miss, BW, DTLB, etc)
  - Mispredicted branch pipeline flushing
  - FE stalls
    - Execution stage instruction starvation (Front End)
  - Cycles wasted executing instructions that are not retired

X86 Cycle Accounting and SW Optimization

- Cpu_clk_unhalted = “stalls” + non_ret_dispatch + ret_dispatch

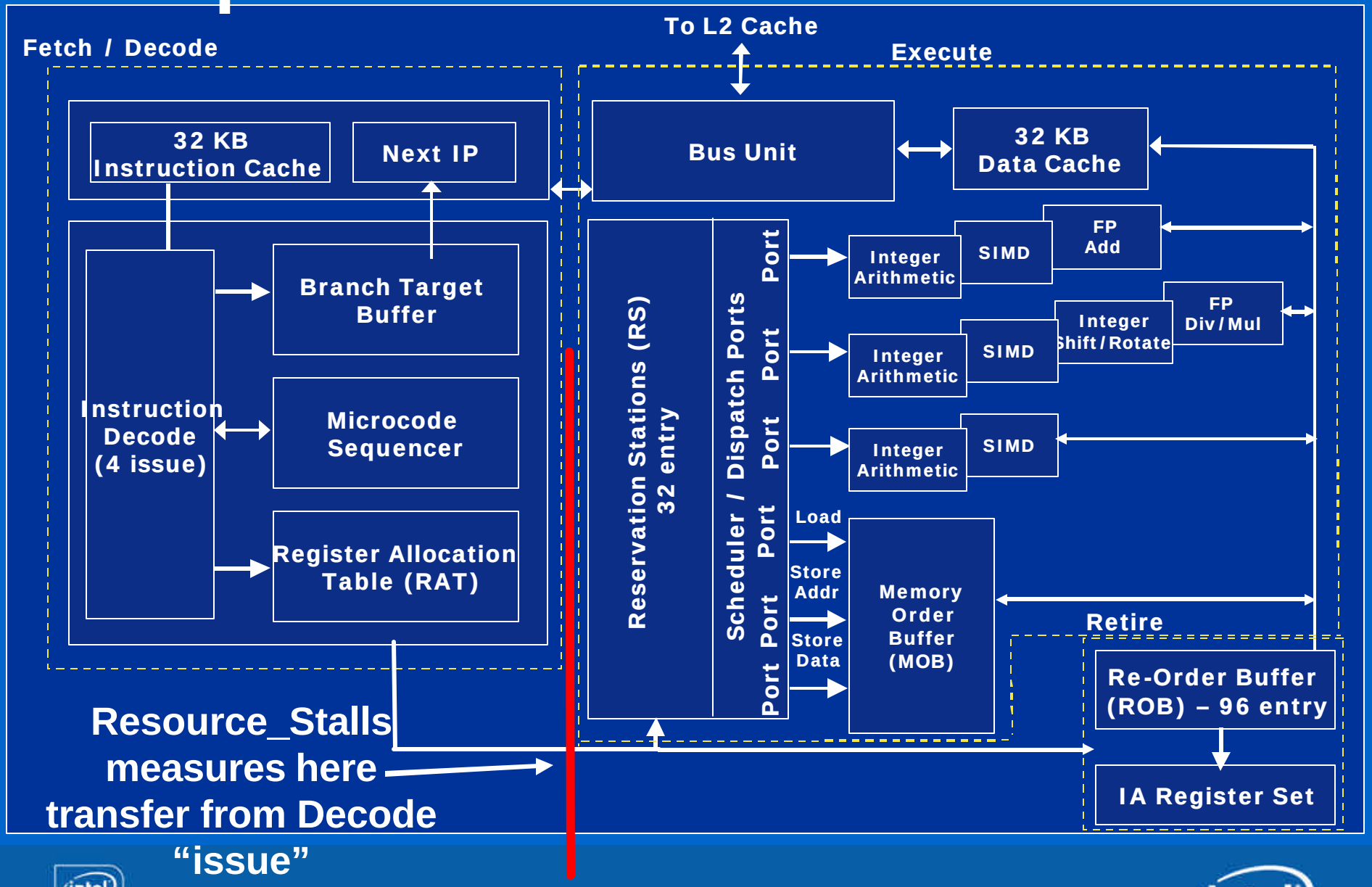
Traditional Stall Removal

Resource_stalls.br_miss_clear will estimate stalls due to Pipeline Flush

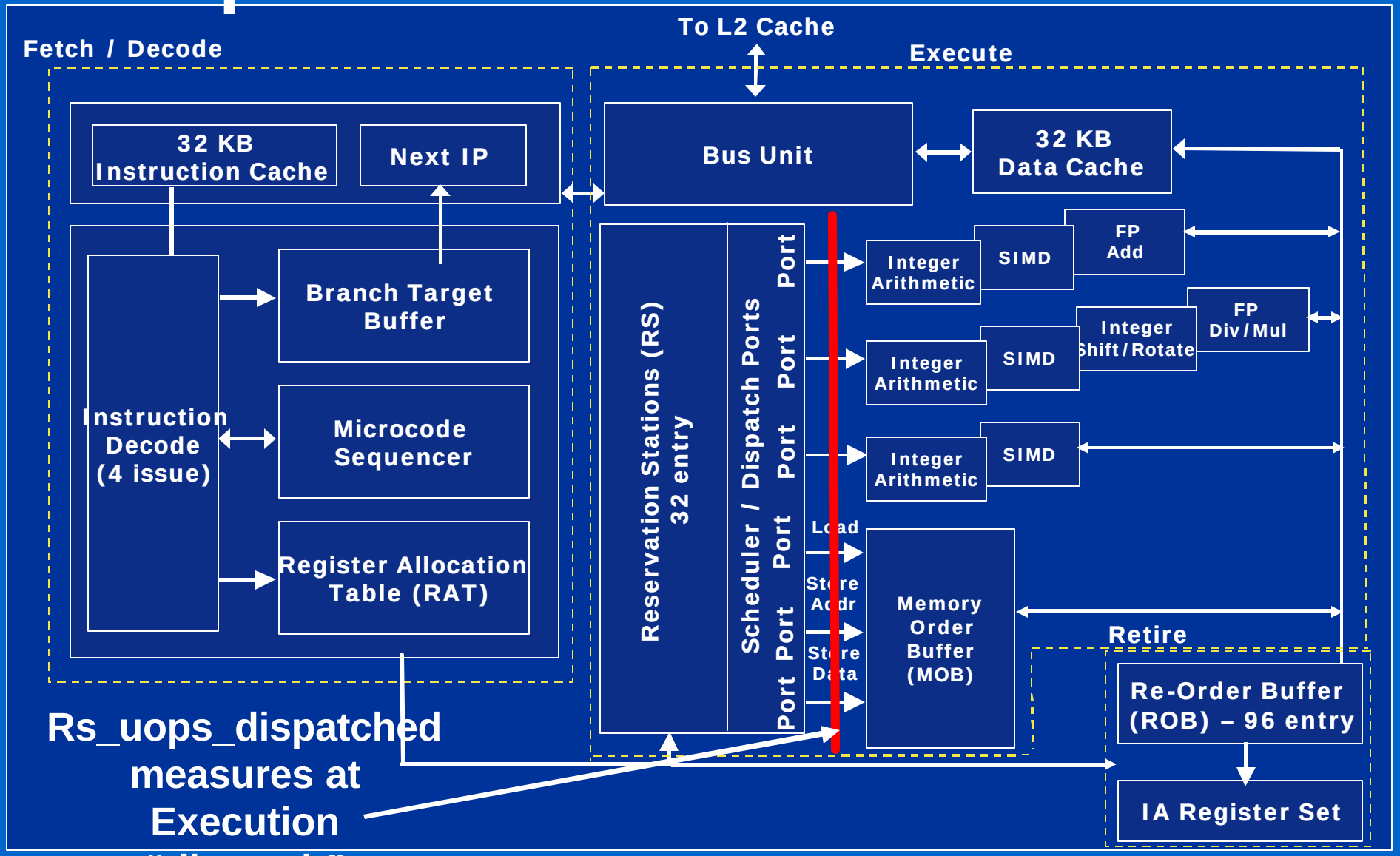
Reduce Branch Mispredictions PGO

Improve Optimization to Reduce Instruction Count, Split Loops to Increase ILP

Uop Flow

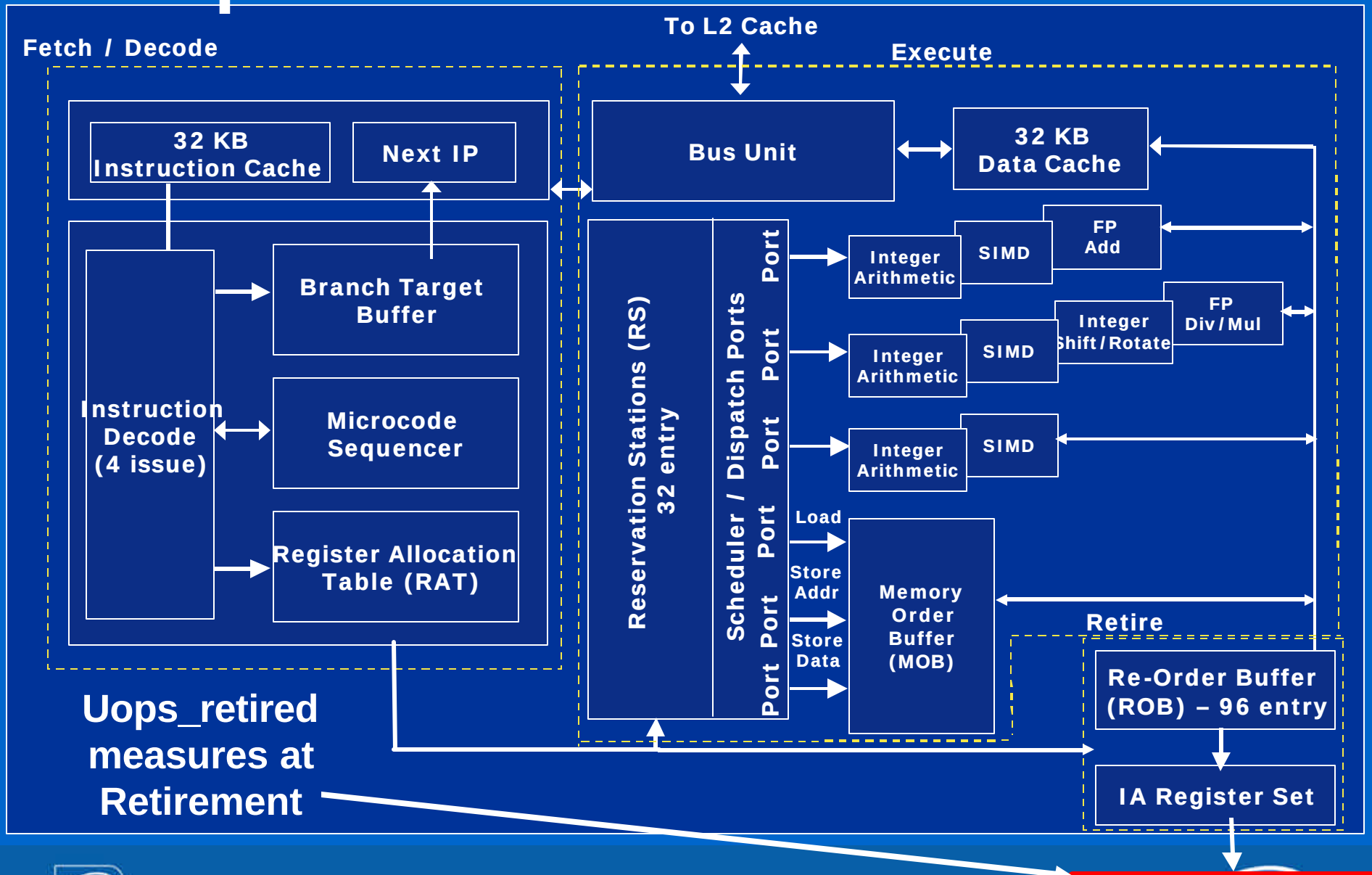

Fetch / Decode
To L2 Cache
Execute
32 KB Instruction Cache
Next IP
Bus Unit
32 KB Data Cache
Branch Target Buffer
Instruction Decode (4 issue)
Microcode Sequencer
Register Allocation Table (RAT)
Reservation Stations (RS) 32 entry
Scheduler / Dispatch Ports
Port
Port
Port
Port
Port
Port
Integer Arithmetic
SIMD
FP Add
Integer Arithmetic
SIMD
Integer Shift/Rotate
FP Div/Mul
Integer Arithmetic
SIMD
Load
Store Addr
Store Data
Memory Order Buffer (MOB)
Retire
Re-Order Buffer (ROB) – 96 entry
IA Register Set
Resource_Stalls measures here
transfer from Decode
"issue"

Uop Flow



Rs_uops_dispatched measures at Execution "dispatch"

Uop Flow

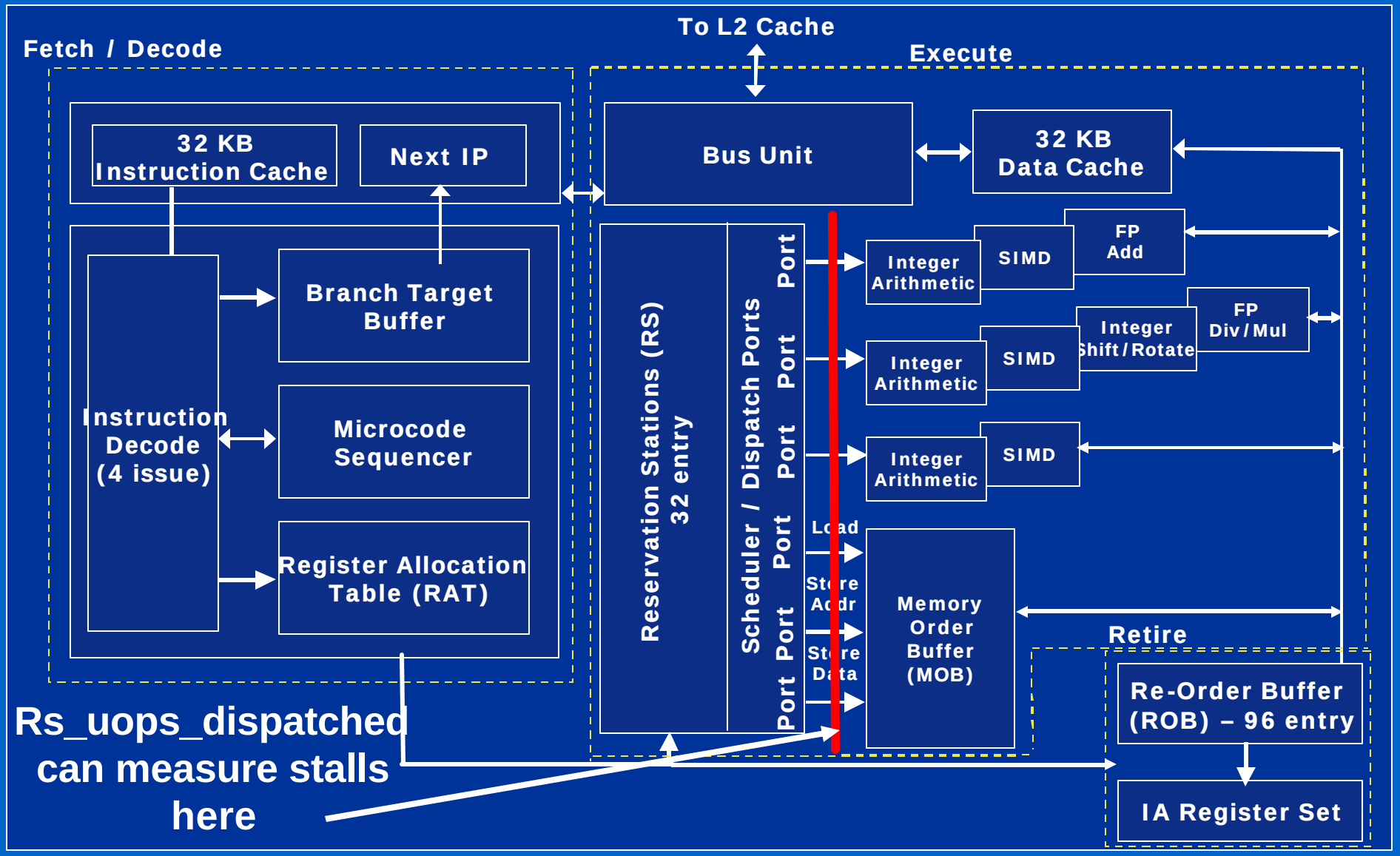

To L2 Cache
Fetch / Decode
Execute
32 KB Instruction Cache
Next IP
Bus Unit
32 KB Data Cache
Branch Target Buffer
Instruction Decode (4 issue)
Microcode Sequencer
Register Allocation Table (RAT)
Reservation Stations (RS) 32 entry
Scheduler / Dispatch Ports
Port
Port
Port
Port
Port
Port
Integer Arithmetic
SIMD
FP Add
Integer Arithmetic
SIMD
Integer Shift / Rotate
FP Div / Mul
Integer Arithmetic
SIMD
Load
Store Addr
Store Data
Memory Order Buffer (MOB)
Retire
Re-Order Buffer (ROB) – 96 entry
IA Register Set
Uops_retired measures at Retirement

Measuring Efficiency in the Execution Stage

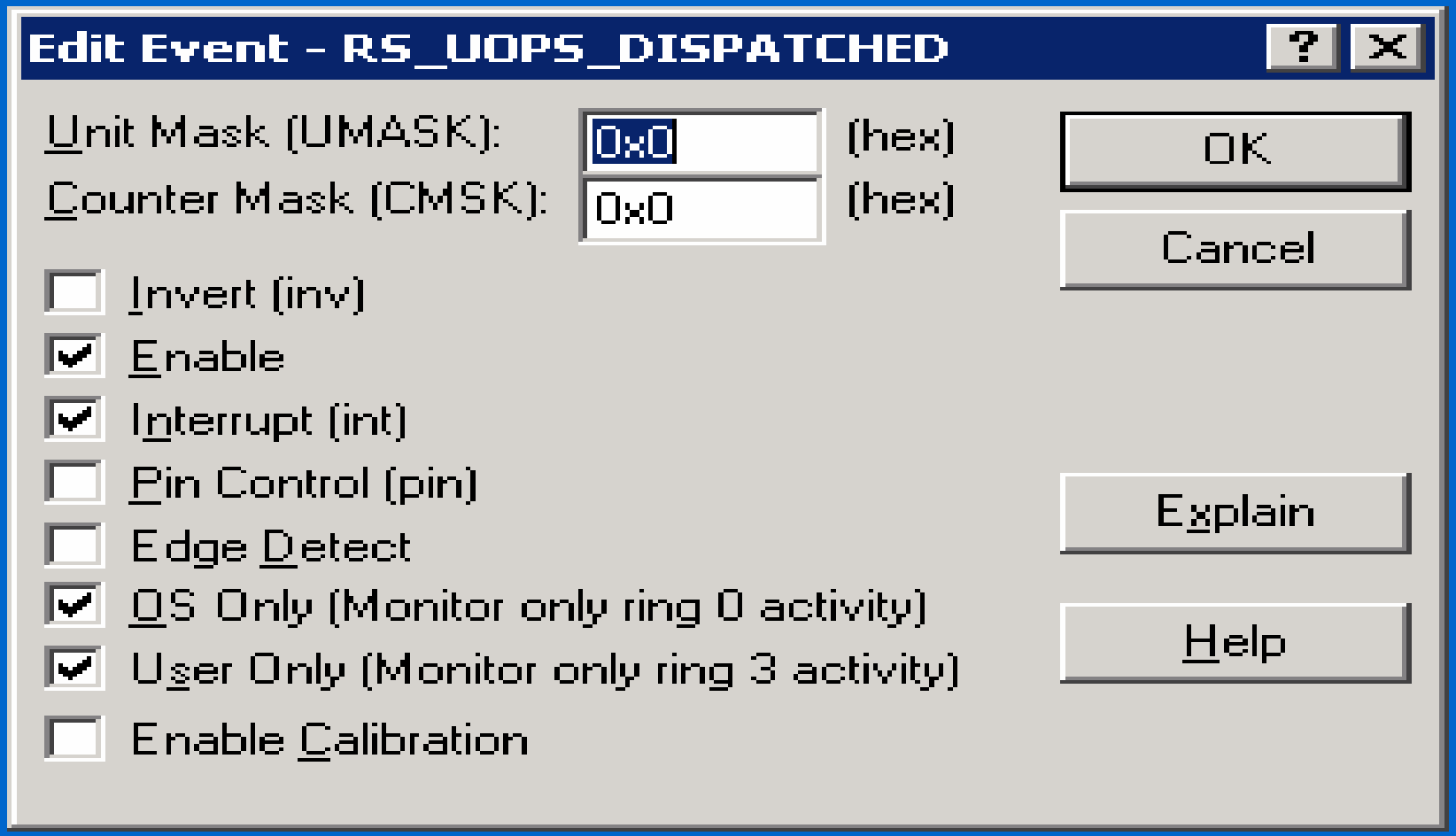
- OOO engine optimizes instruction dispatch to execution units from Reservation Station
  - They wait there until their inputs are available
- RS_UOPS_DISPATCHED measures number of uops dispatched from RS on each cycle

The Single Most Important PMU Event!

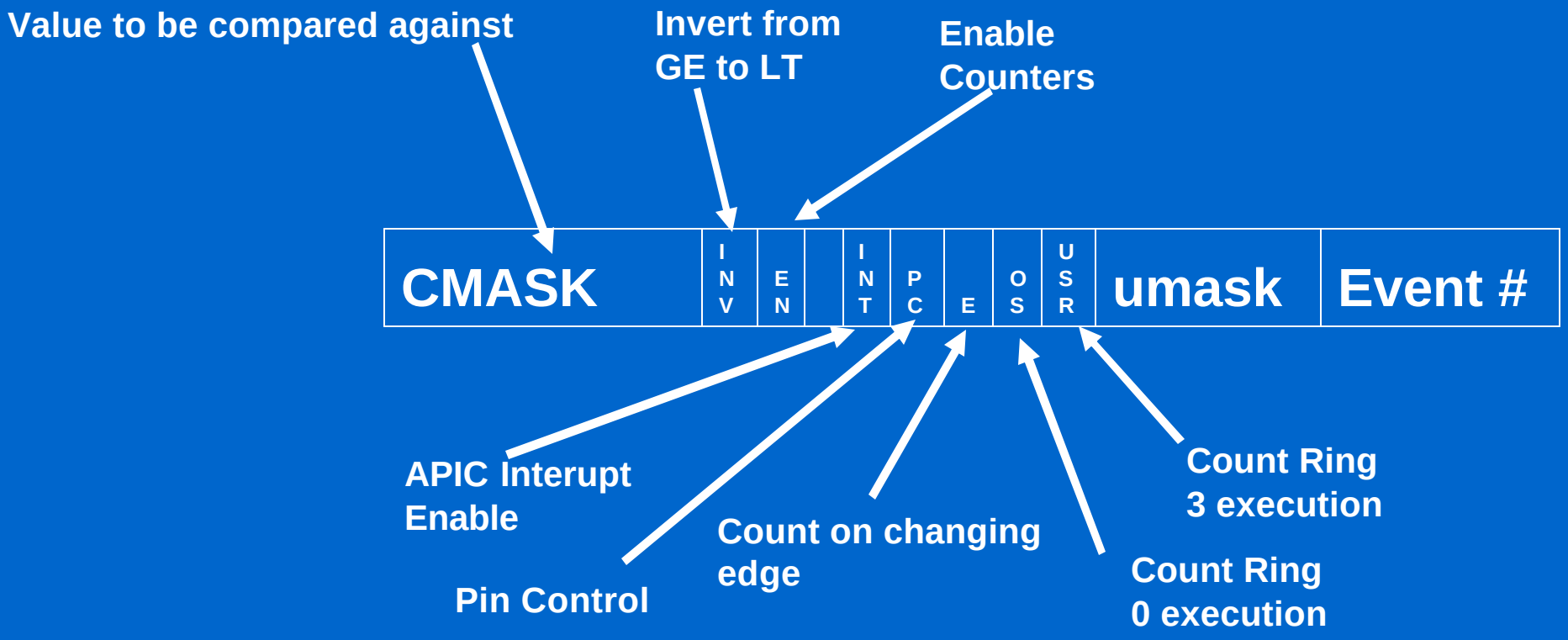
Execution is What Matters



VTune™ Analyzer Edit Event

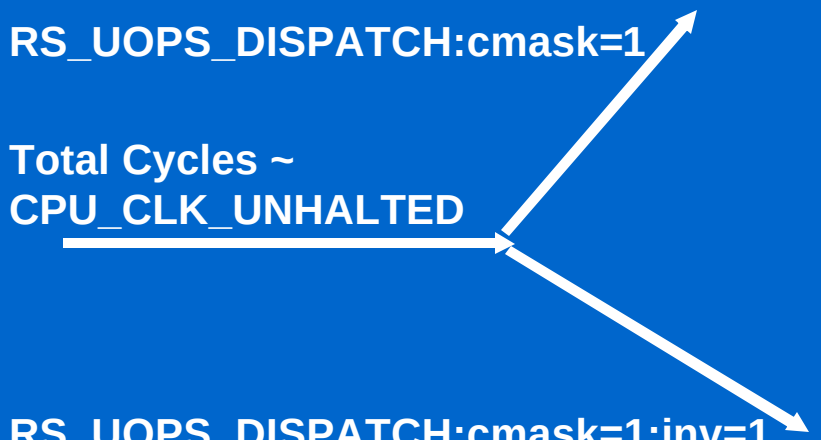


Some Features of the PMU



Setting CMASK = 1 and INV = 1 for RS_uops_dispatched
Counts Cycles Where
NO UOPS WERE DISPATCHED == Stalls
RS_UOPS_DISPATCHED.CYCLES_NONE

A Methodology?



CPU_CLK_UNHALTED can be decomposed into execution and stall cycles in the OOO engine

Requires >99% CPU Utilization

OR User PL only/sampling

EVENTS COUNT EVEN DURING HALTED CYCLES

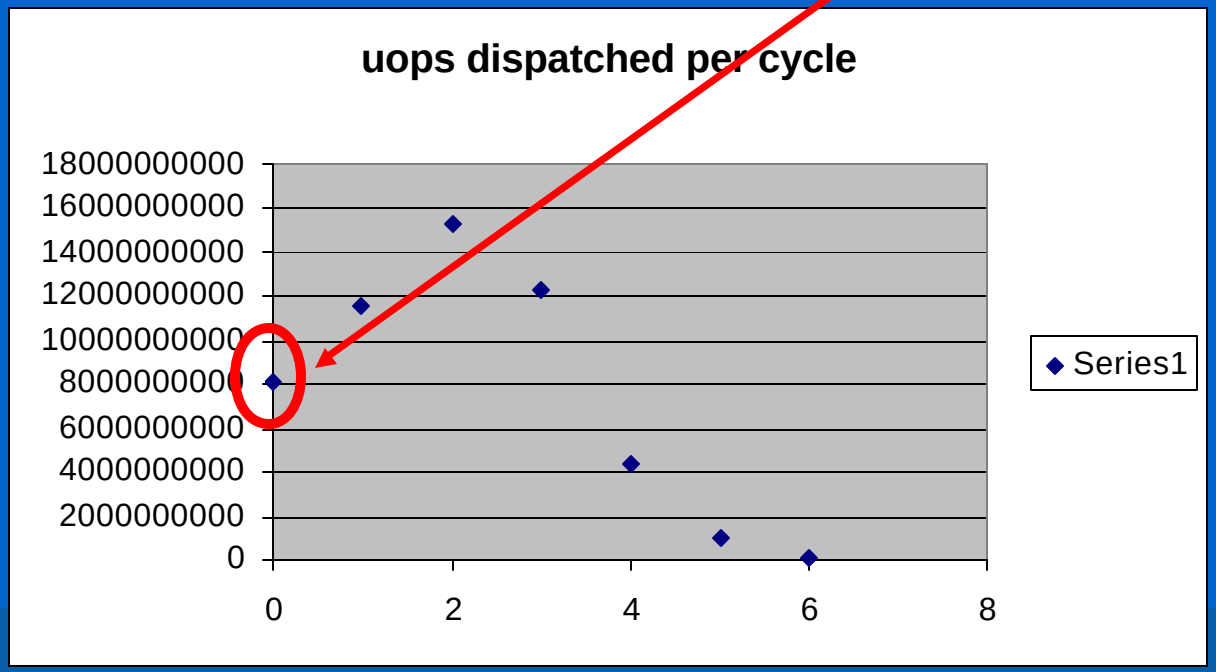
The Distribution of uops/cycle

RS_UOPS_DISPATCHED:cmask=1:inv=1
RS_UOPS_DISPATCHED:cmask=2:inv=1
RS_UOPS_DISPATCHED:cmask=3:inv=1
RS_UOPS_DISPATCHED:cmask=4:inv=1
RS_UOPS_DISPATCHED:cmask=5:inv=1
RS_UOPS_DISPATCHED:cmask=6:inv=1
RS_UOPS_DISPATCHED:cmask=7:inv=1

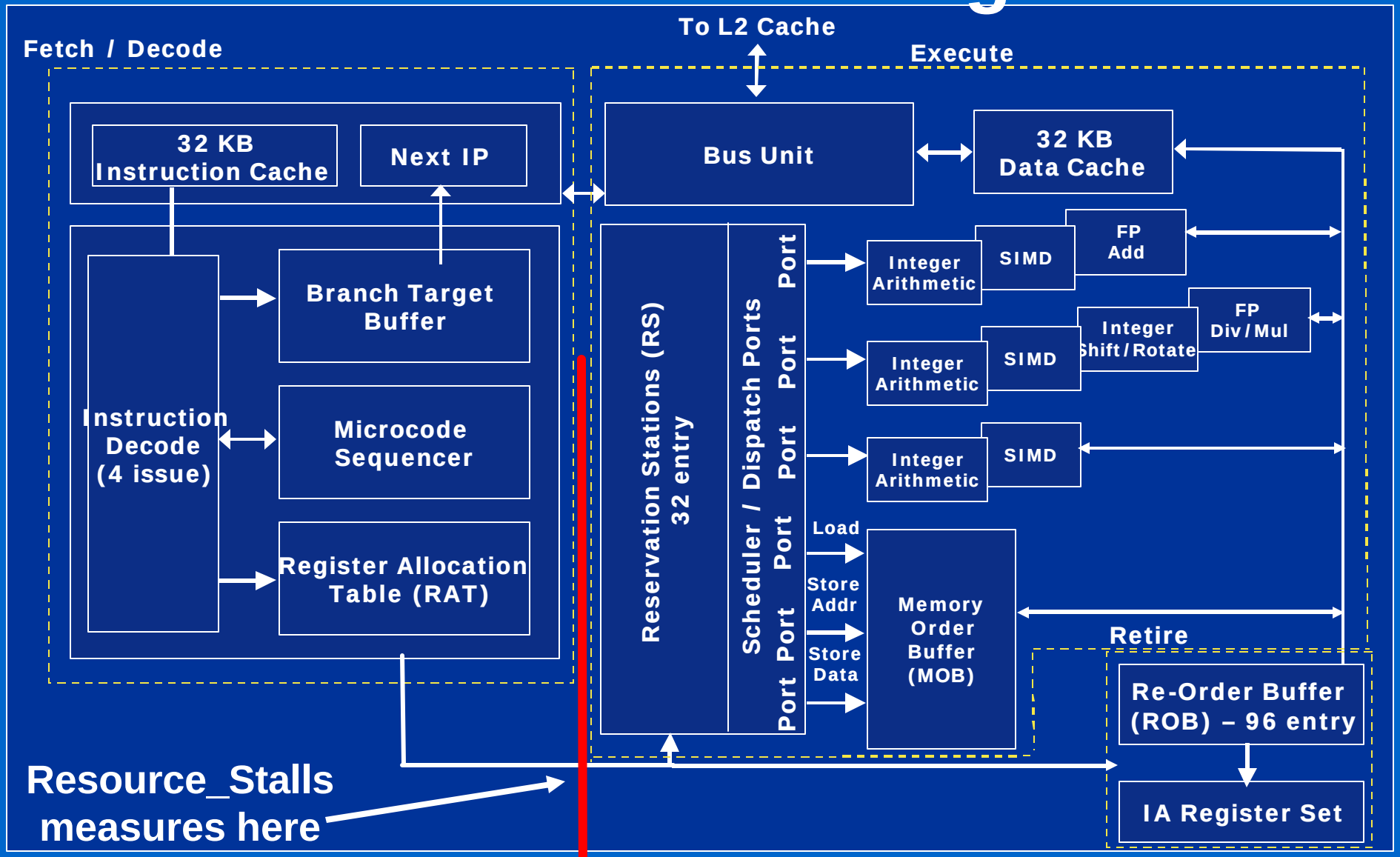
Subtract the N-1 value

Stall Cycles

Distribution of the Instruction Level Parallelism
(example:
a[i] = exp(x[i]);
in a simple loop)



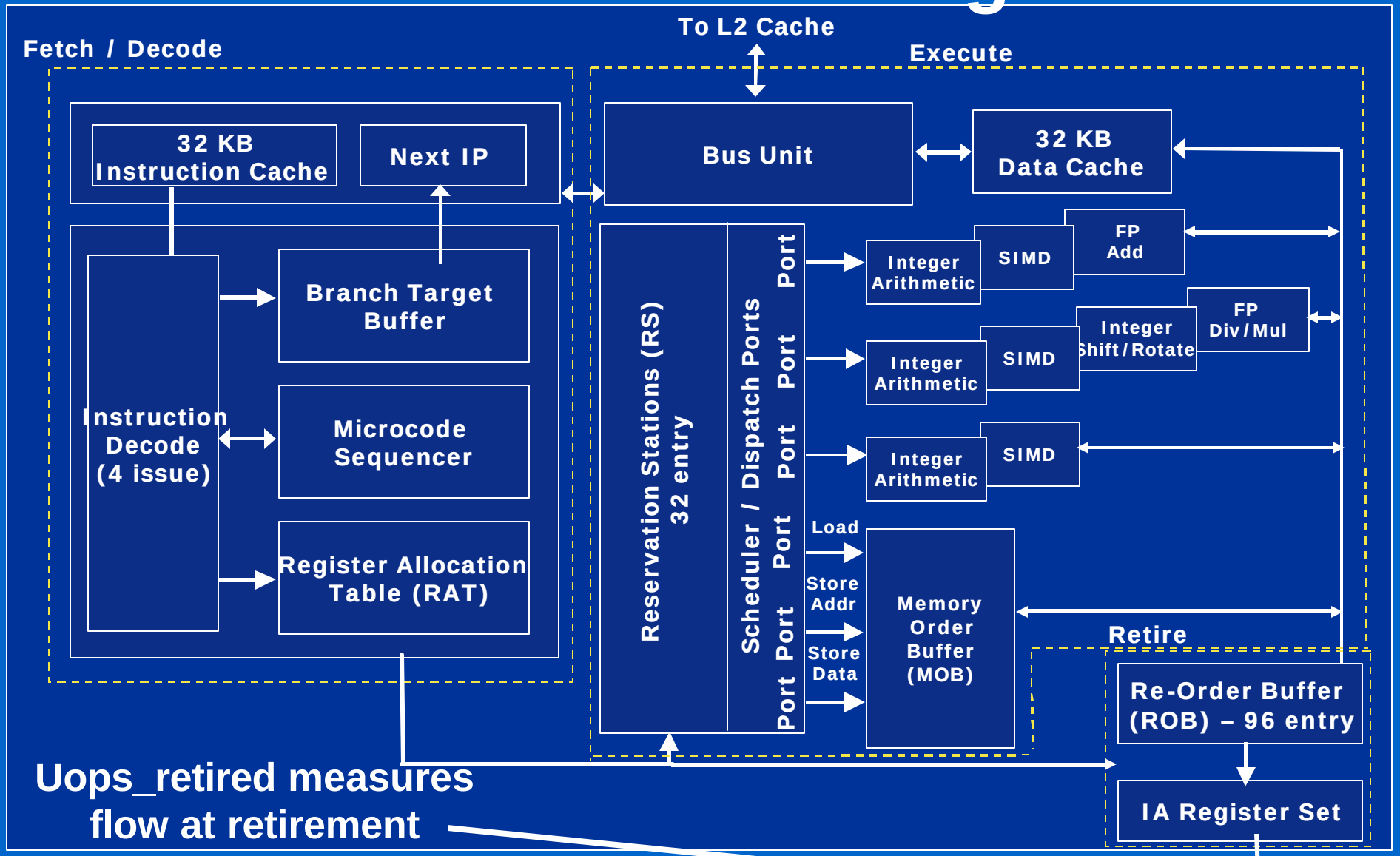
Architecture Block Diagram



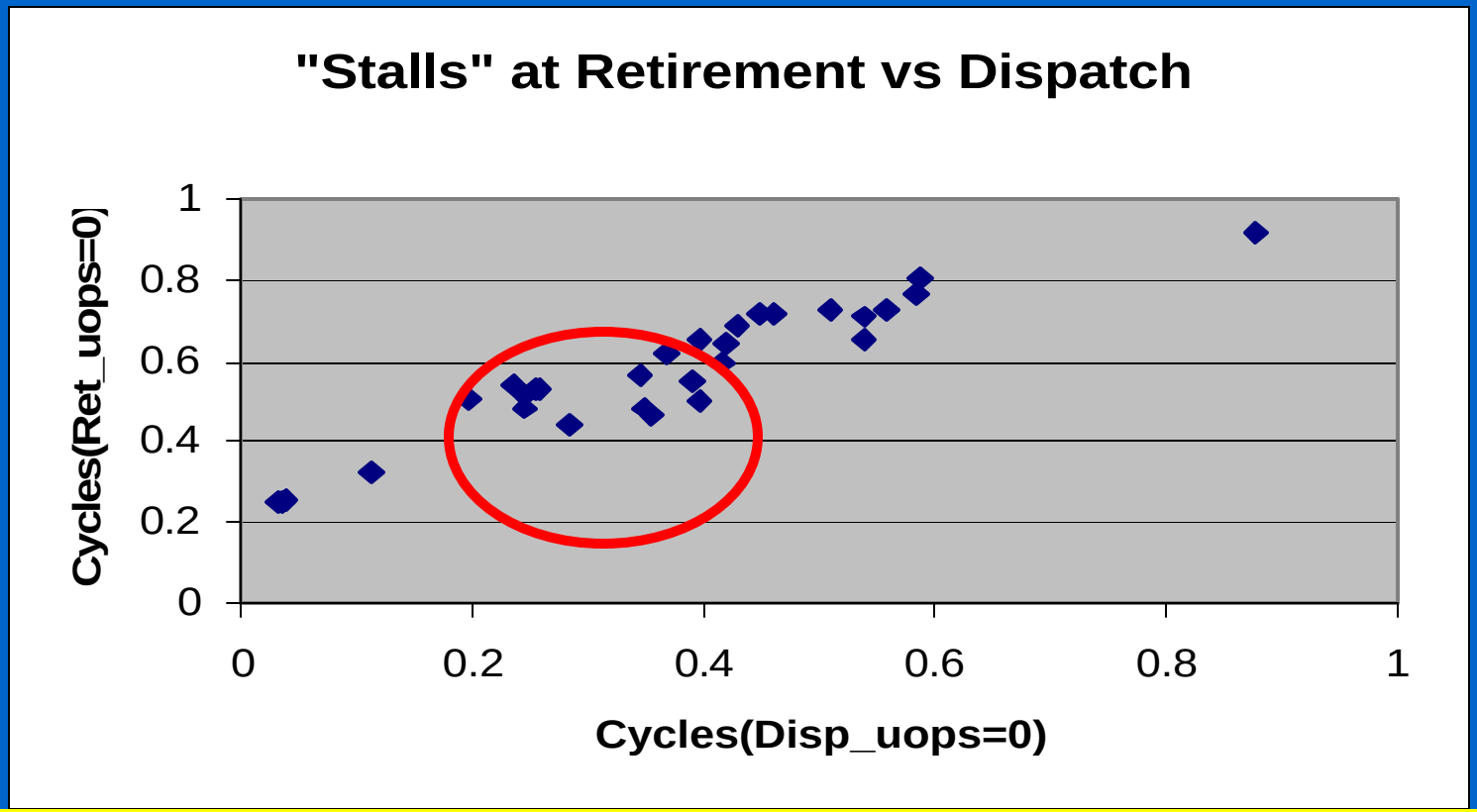
Components of Resource _Stalls

- Uop flow to OOO engine blocked by downstream cause
- Resource_Stalls.BR_MISS_CLEAR
  - pipeline stalls due to flushing mispredicted branches
    Combine in Resource_stalls.CLEAR
    mispredicted branch followed by fp inst
- Resource_Stalls.ROB_FULL
  96 instructions in ROB
- Resource_Stalls.LD_ST
  All Store or Load buffers in use
- Resource_Stalls.RS_FULL
  32 instructions waiting for inputs
  in Reservation Station

Architecture Block Diagram



Retirement or Dispatch?
Which Function to work on first?



For loops, difference is due to OOO execution
Fewer False Positives When "Stalls" Are Measured at Dispatch

Cycle Accounting on X86

- **Cycles =**

  rs_uops_dispatched.cycles_none + rs_uops_dispatched:cmask=1

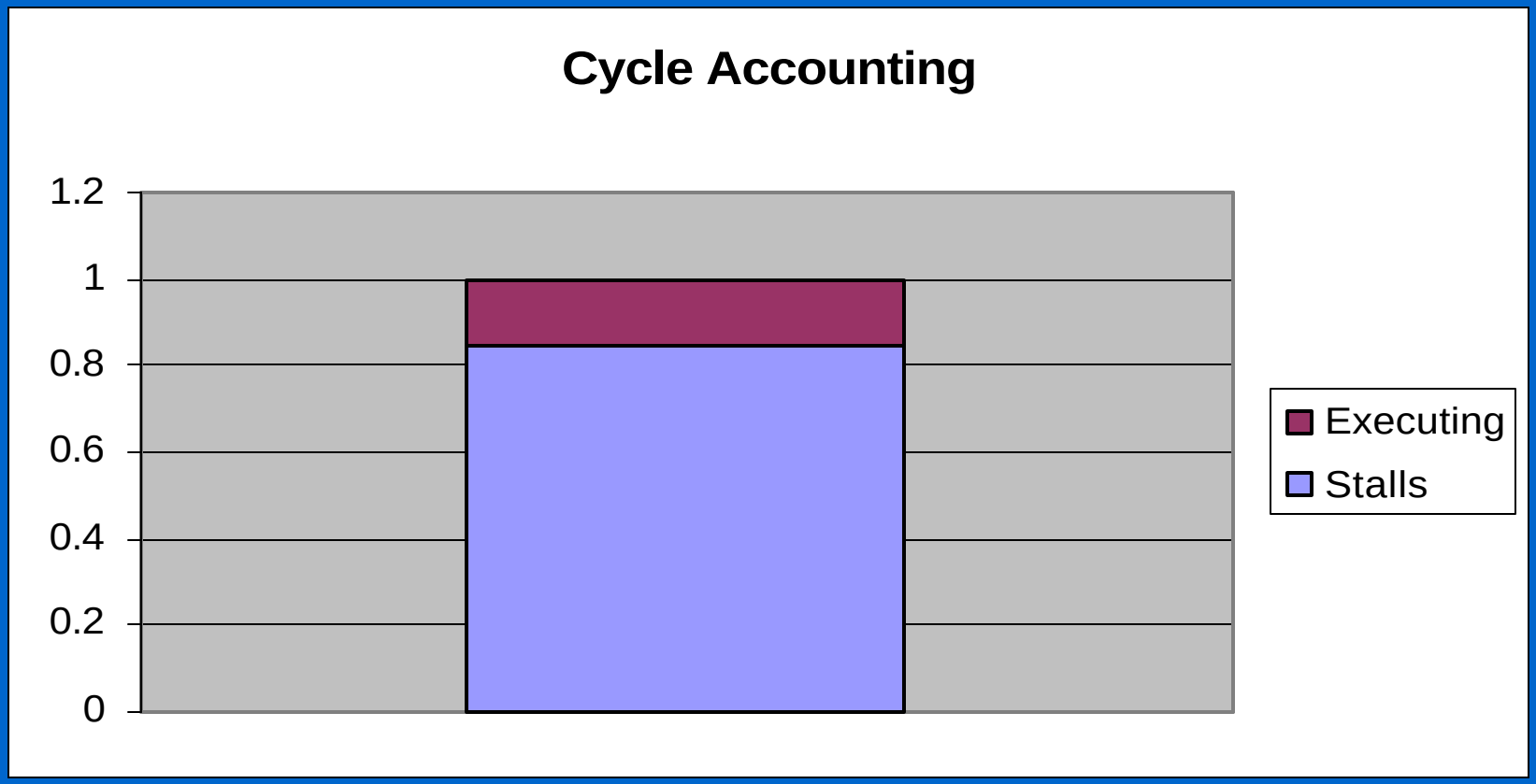
  “stalls” + dispatch

  - **An equality by definition**
- **Cycles ~ CPU_CLK_UNHALTED.CORE**
  - **For cpu intensive applications/sampling**

Cycle Accounting on X86

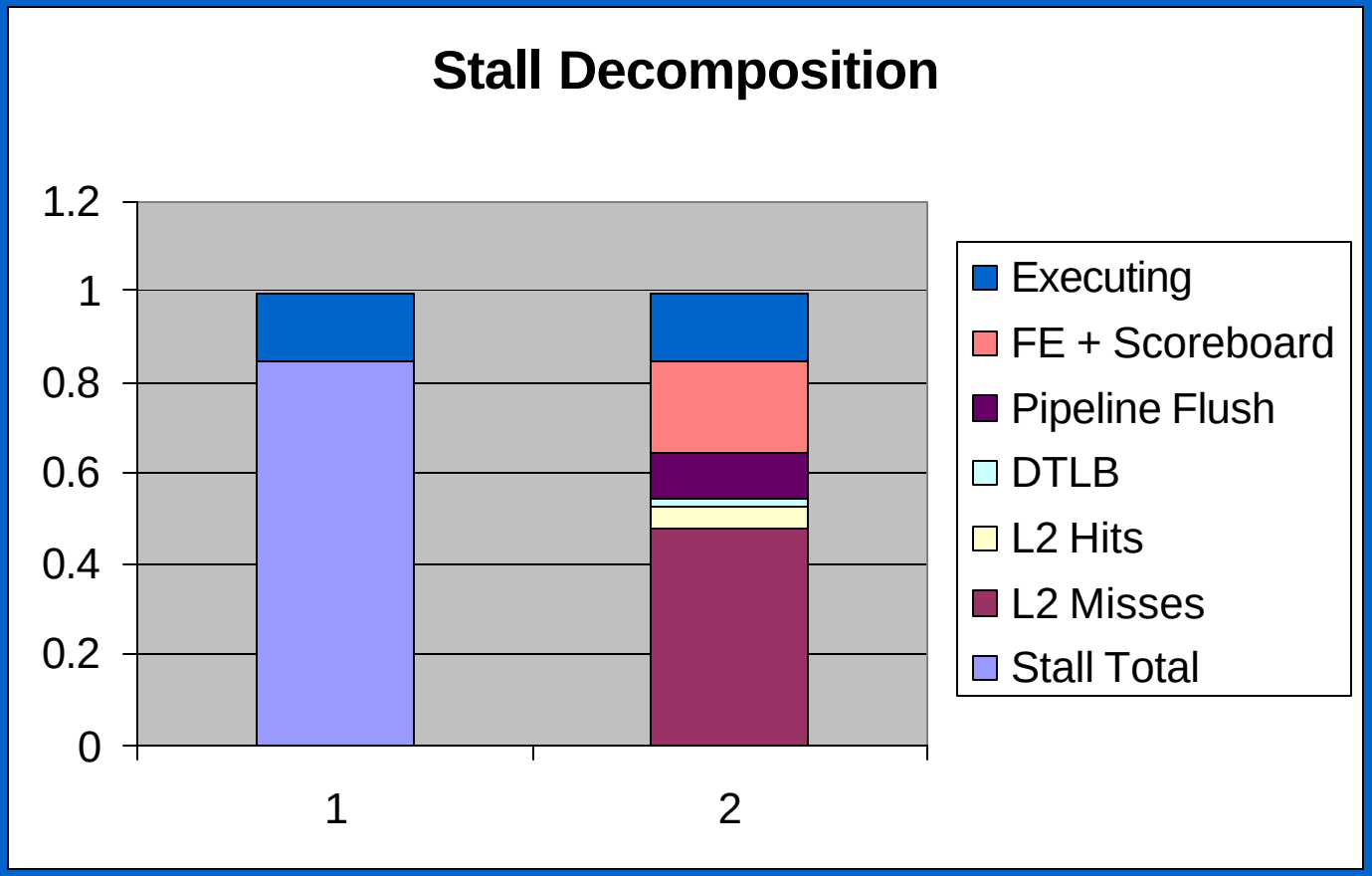
- Dispatch ~ cycles_dispatch_retiring_uops + cycles_dispatch_non_retiring_uops
  - Assumes no overlap of retired/non retired uops
    - Worst Case Senario
- Non retired uops = rs_uops_dispatched – (uops_retired.any + Uops_retired.fused)
  - Non retired uop cycles ~ non retired uops/avg_uops_per_cycle
- Fractional Wasted Work = rs_uops_dispatched / (uops_retired.any + uops_retired.fused) - 1

Pulling Cycle Accounting Together



Stall Cycles = Cycles with NO uops Dispatched
= RS_UOPS_DISPATCH.CYCLES_NONE

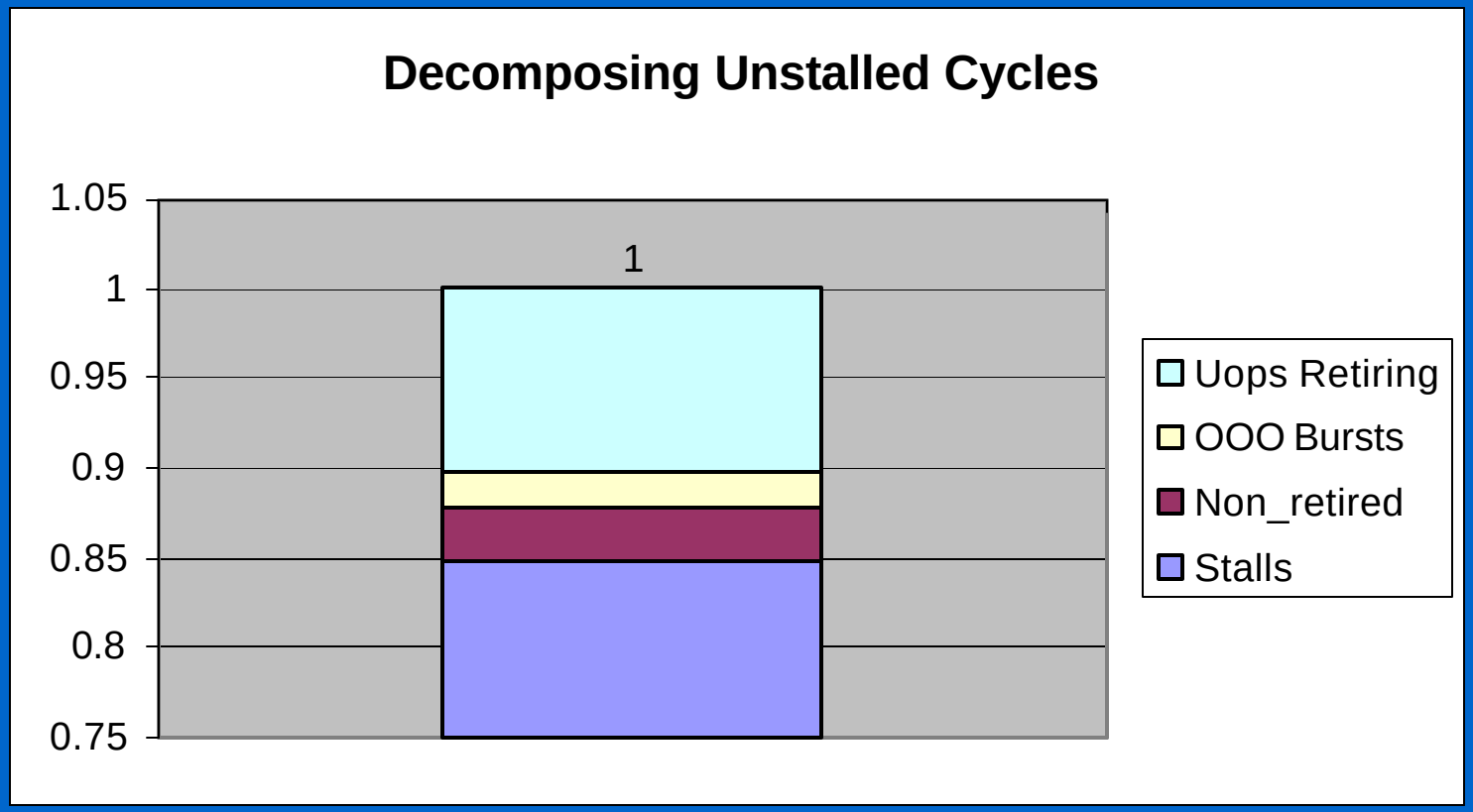
Decomposing Stalls: Elephants First



Pipeline Flush = Resource_Stalls.Br_Miss_Clear/cycles
L2 Hits = (MEM_LOAD_RETIRED.L1D_LINE_MISS - MEM_LOAD_RETIRED.L2_LINE_MISS)* 10/cycles
DTLB/L2 Miss = event count* penalty/cycles
FE + Scoreboard = Stalls – all of the above

Illustrative Example Only, Not Real Data

Decomposing Unstalled Cycles



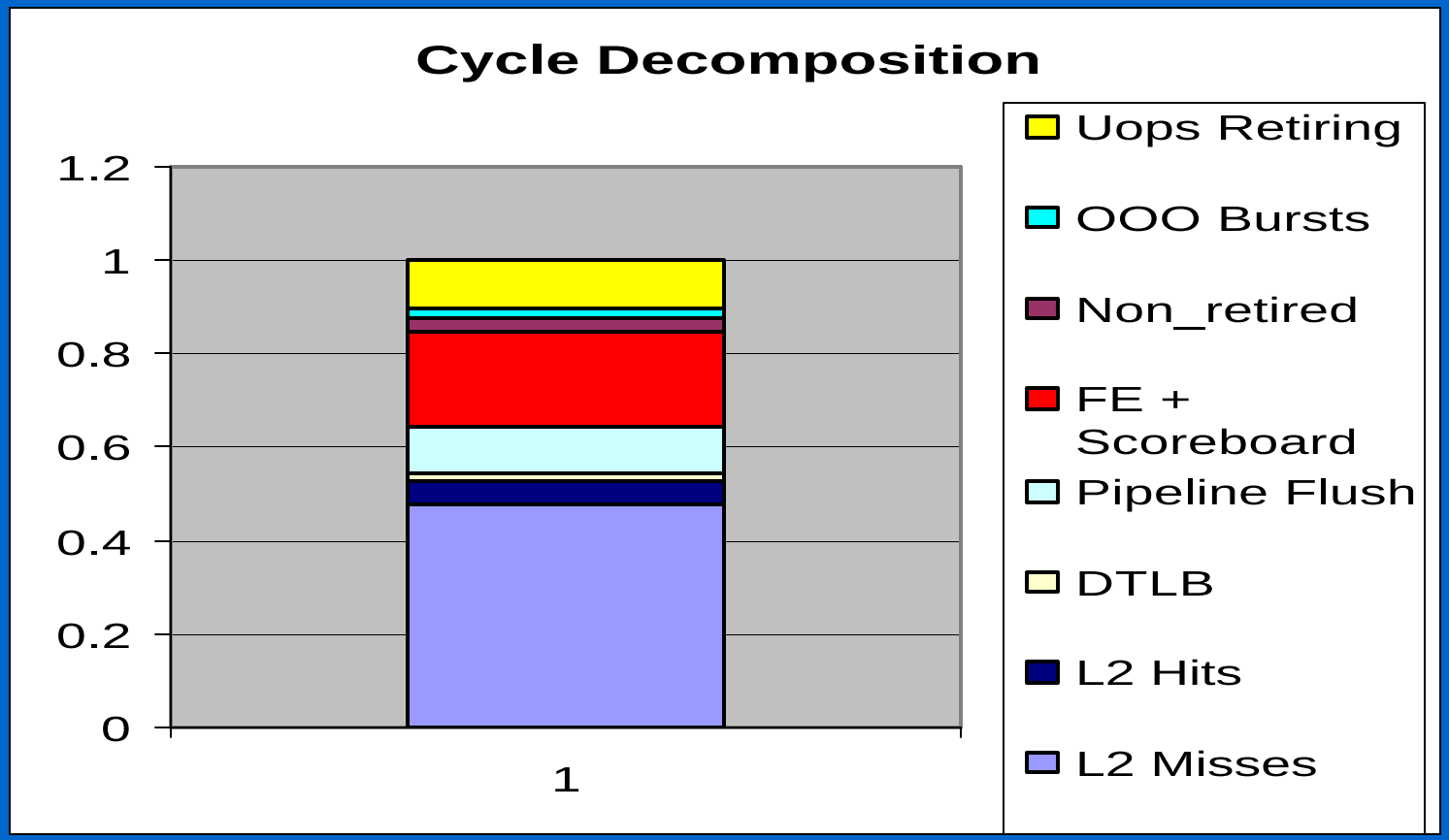
Non_Retired =

((1 – (Uops_retired.any+Uops_retired.fused)/RS_Uops_Dispatched) * RS_Uops_Dispatched:cmask=1 / CPU_CLK_UNHALTED.CORE

OOO Bursts = Uops_Retired.Any.cycles_none- Stalls – Non_Retired

Illustrative Example Only, Not Real Data

Pulling it All Together



Risks Over-counting / Minimizing FE + Scoreboard

But Offers a Guide to Execution Inefficiencies

Illustrative Example Only, Not Real Data

Architectural Pitfalls: The Ants

Issue	Performance Counter	Approx. Penalty
store to unknown addr preceeds load	Load_Blocks.ADR	~5
store forwarding 4 bytes from middle of 8	Load_Blocks.Overlap_Store	~6
store to known address precedes load offset by N*4096	Load_Blocks. Overlap_Store	~6
load from 2 cachelines (not in L1D)	Load_Blocks.UNTIL_RETIRE	~22
load from 2 cachelines with preceding store(not in L1D	Load_Blocks.UNTIL_RETIRE	~20
Length Changing Prefix (16 bit imm)	ILD_STALLS	~6

Contribute to “FE + Scoreboard”

The Big 4 Revisited

- CPU_CYCLES ->
  CPU_CLK_UNHALTED.CORE
- BACK_END_BUBBLE.ALL ->
  RS_UOPS_DISPATCHED.CYCLES_NONE
- BUS_MEMORY.ALL.SELF ->
  BUS_TRANS_ANY.SELF
- DEAR_LATENCY_GT_64 ->
  MEM_LOAD_RETIRED.L2_LINE_MISS

CYCLES, STALLS, UNPREFETCHED LOADS and BANDWIDTH

More Detailed Event Selection Hierarchy

FIRST PASS EVENTS	Sample After Value
CPU_CLK_UNHALTED.CORE	2,000,000
RS_UOPS_DISPATCHED.CYCLES_NONE	2,000,000
UOPS_RETIRED.ANY + UOPS_RETIRED.FUSED	2,000,000
RS_UOPS_DISPATCHED	2,000,000
MEM_LOAD_RETIRED.L2_LINE_MISS	10,000
INST_RETIRED.ANY_P	2,000,000
Loops	
BUS_TRANS_ANY.SELF	100,000
BUS_TRANS_ANY.ALL_AGENTS	100,000
Branch Dominated	
RESOURCE_STALLS.BR_MISS_CLEAR	2,000,000

SAV values selected so ratio of samples ~ absorbs penalty

Event Hierarchy

SECOND LEVEL EVENTS	Sample After Value
MEM_LOAD_RETIRED.DTLB_MISS	20,000
MEM_LOAD_RETIRED.L2_MISS	10,000
MEM_LOAD_RETIRED.L1_LINE_MISS	200,000
BR_CND_MISSP_EXEC	2,000,000
BR_CND_EXEC	2,000,000
BR_CALL_EXEC	200,000
BR_CALL_MISSP_EXEC	200,000
ILD_STALLS	200,000
LOAD_BLOCK.STORE_OVERLAP	200,000

SAV values selected so ratio of samples ~ absorbs penalty

EX: L1 miss/L2_hit penalty is 10 cycles

A loop Methodology?

- Identify hot functions and raise optimization
  - Fix alignments, split loops to enhance vectorization
- Identify BW limited functions
  - Merge BW loops with FP limited loops
- Identify L2 misses and add sw prefetch
- Optimize flow through OOO Engine
  - Use loop splitting to assist here

Summary

- Intel® Core™2 processor is very good for loops
  - 2 simd instr/cycle
  - Short pipeline
- Intel® Core™2 processor PMU is very good
  - Bandwidth is simple and clear
    - Though address bus utilization is perhaps not
  - More Precise Events
    - Instr_retired and its components
    - LLC miss
  - Execution Inefficiency for loops can be estimated


intel®
Leap ahead™