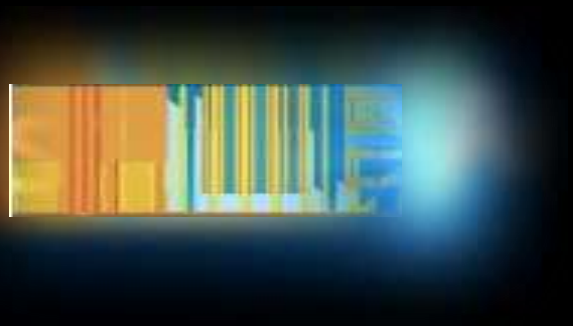


2014 IEEE International Symposium on Performance Analysis of Systems and Software. March 23-25, Monterey, CA

A Top-Down Method for Performance Analysis and Counters Architecture

Ahmad Yasin

Intel Corporation

Motivation

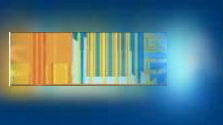
- Performance Optimization Is Difficult
  - Complicated micro-architectures
  - Application/workload diversity
  - Unmanageable data
  - Tougher constraints: Time, Resource, Priorities
- Opportunities once true bottleneck is identified
  - Software Tuning/Optimization
  - Workload Characterization
  - Profile-Guided Optimizations
  - Resource utilization in the Cloud

Top Down Analysis

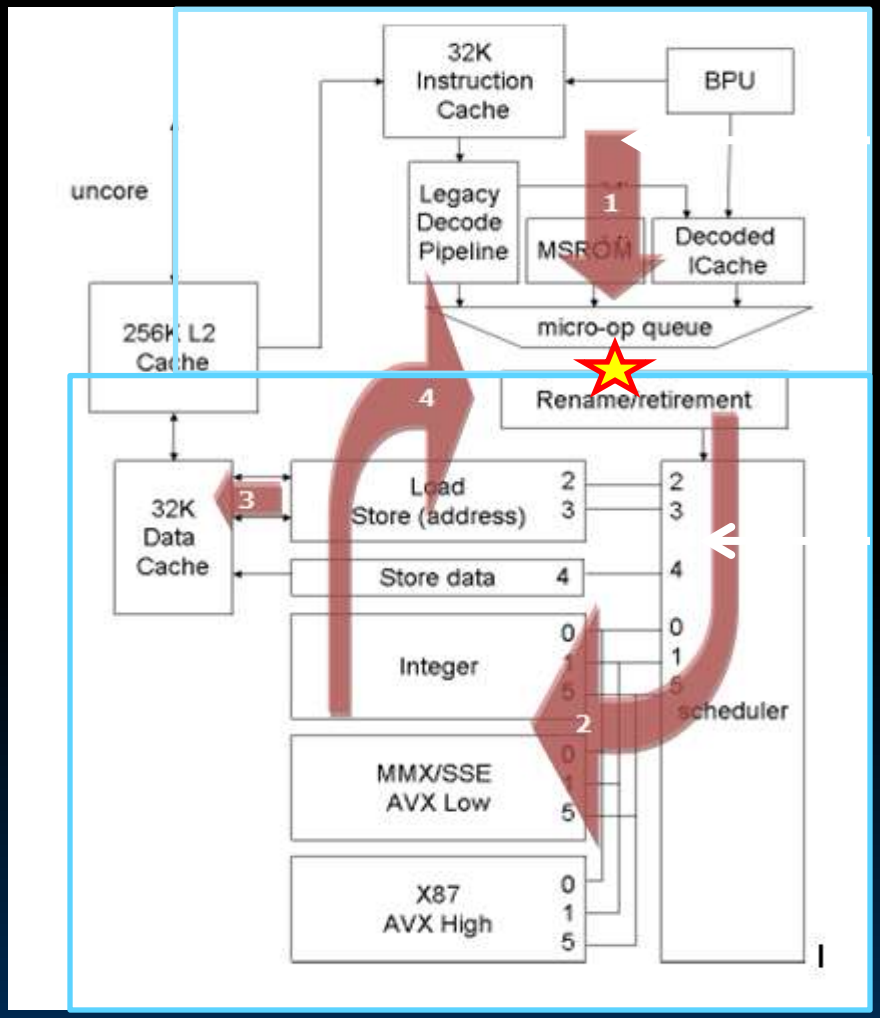
- A method to identify the true bottlenecks
  - **Simple:** a structured hierarchical approach
  - **Quick**: a few steps to get to a tree-leaf
  - **Practical:** adopted by in-production tools, e.g. VTune[2]
- Benefits
  - Analysis made easier for non-expert users
  - Simplicity avoids u-arch high-learning curve
- Assumptions
  - Goal: detect bottleneck; Not-a-goal: quantify speedup
  - A sub-level of Analysis Process: System, Application, **u/Architecture**

Agenda

- ✓Motivation
- Challenges
- Top Down Analysis Hierarchy
- Top Level heuristics
- Counters Architecture
- Results
- Summary



Intel Core™ µarch



Front-end
of processor pipeline

Back-end
of processor pipeline

Where and How to start in this Complex microarchitecture?

Challenges

Traditional Methods

- Naïve approach

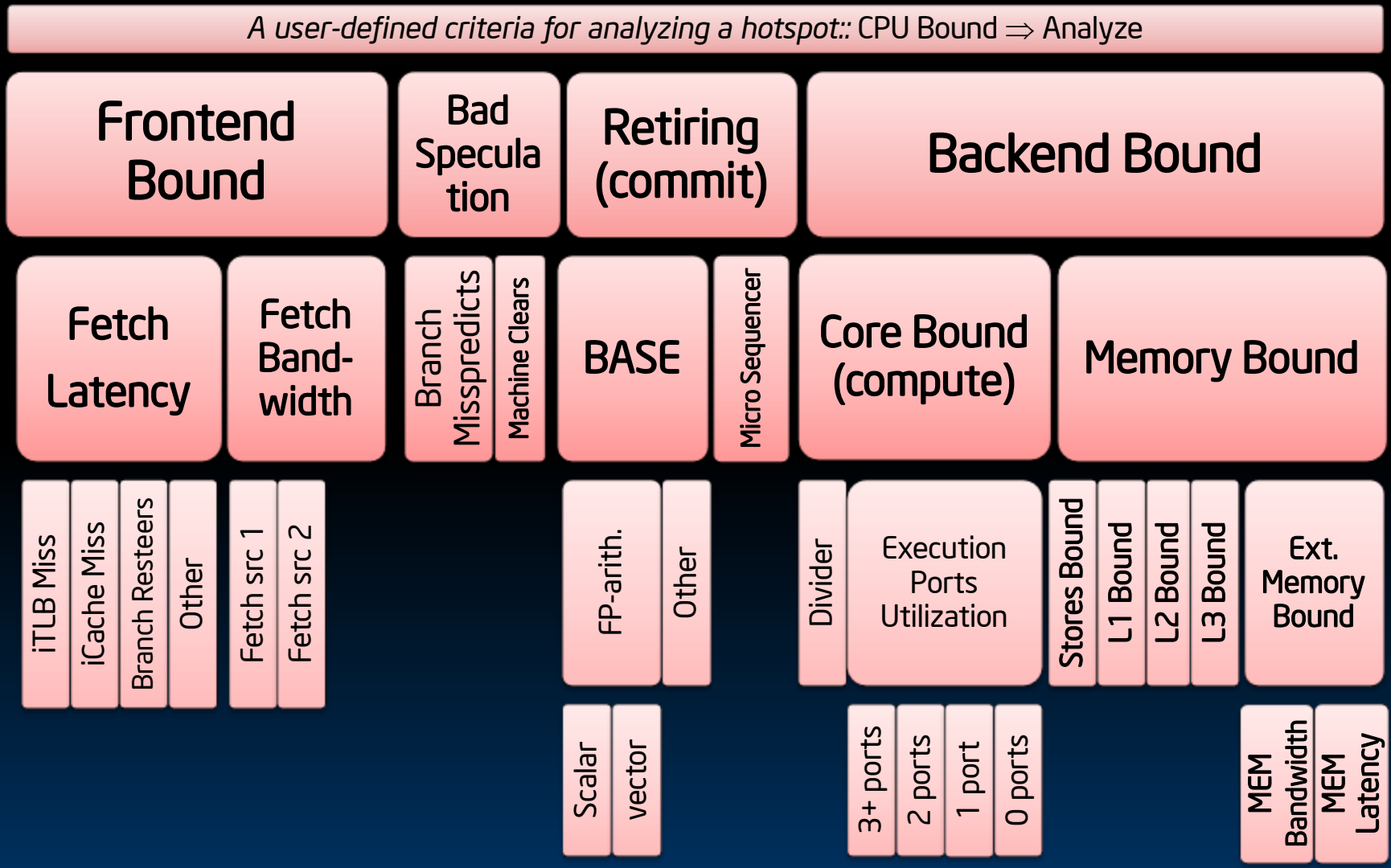
  $Stall\_Cycles = \Sigma\ Penalty_i * MissEvent_i$

- Unsuitable for out-of-orders (Gaps)
  1) Stalls Overlap
  2) Speculative Execution
  3) Workload-dependent penalties
  4) Predefined set of miss-events
  5) Superscalar inaccuracy

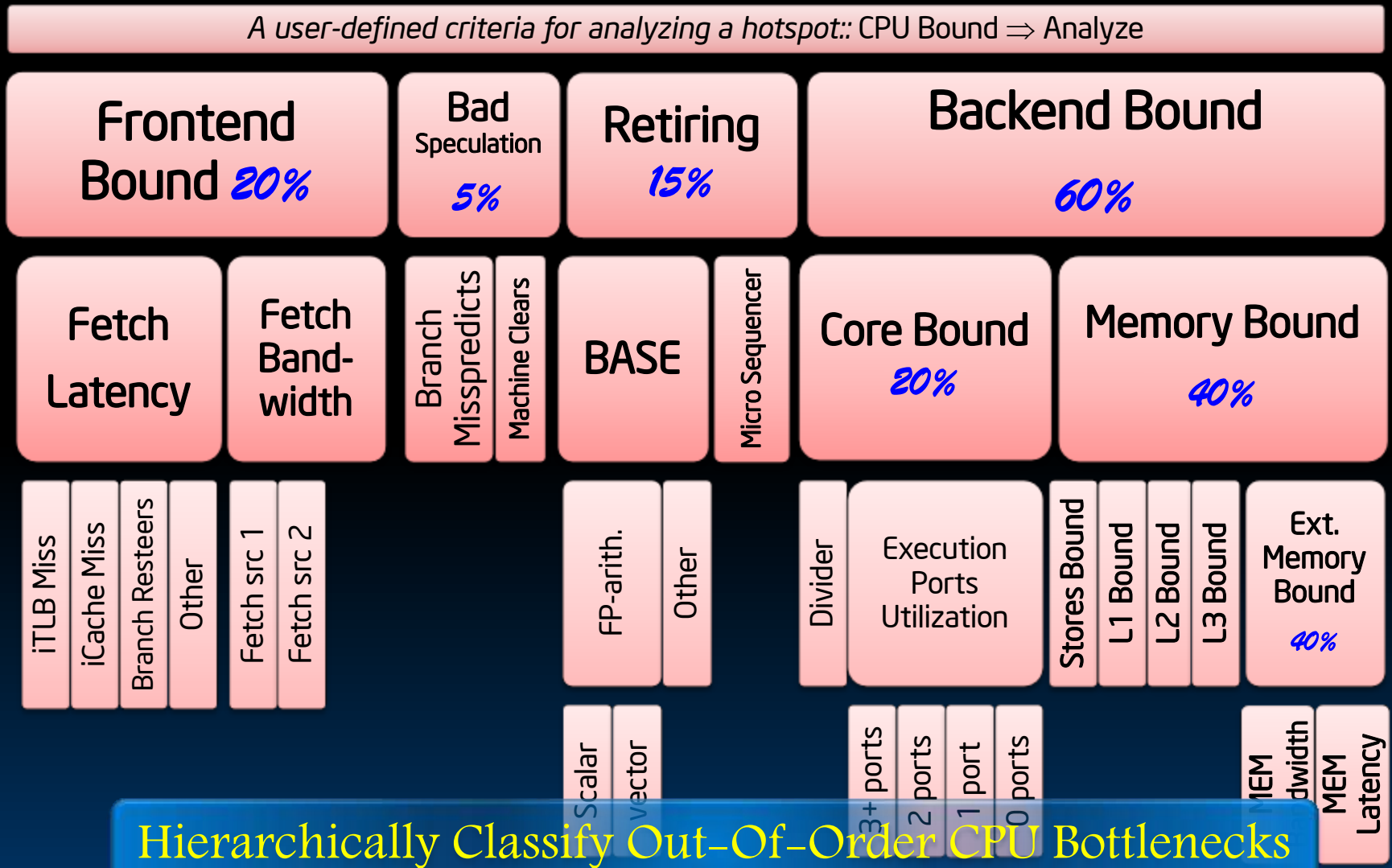
Top Down Analysis

- A hierarchy
  - Top-Down designated events at appropriate pipeline stages
  - "Hierarchical safety property"
- Addressing Gaps
  - Bad Speculation at the top
  - Generic top-down events, who
  - count when matters, and
  - count where matters
  - Occupancy events

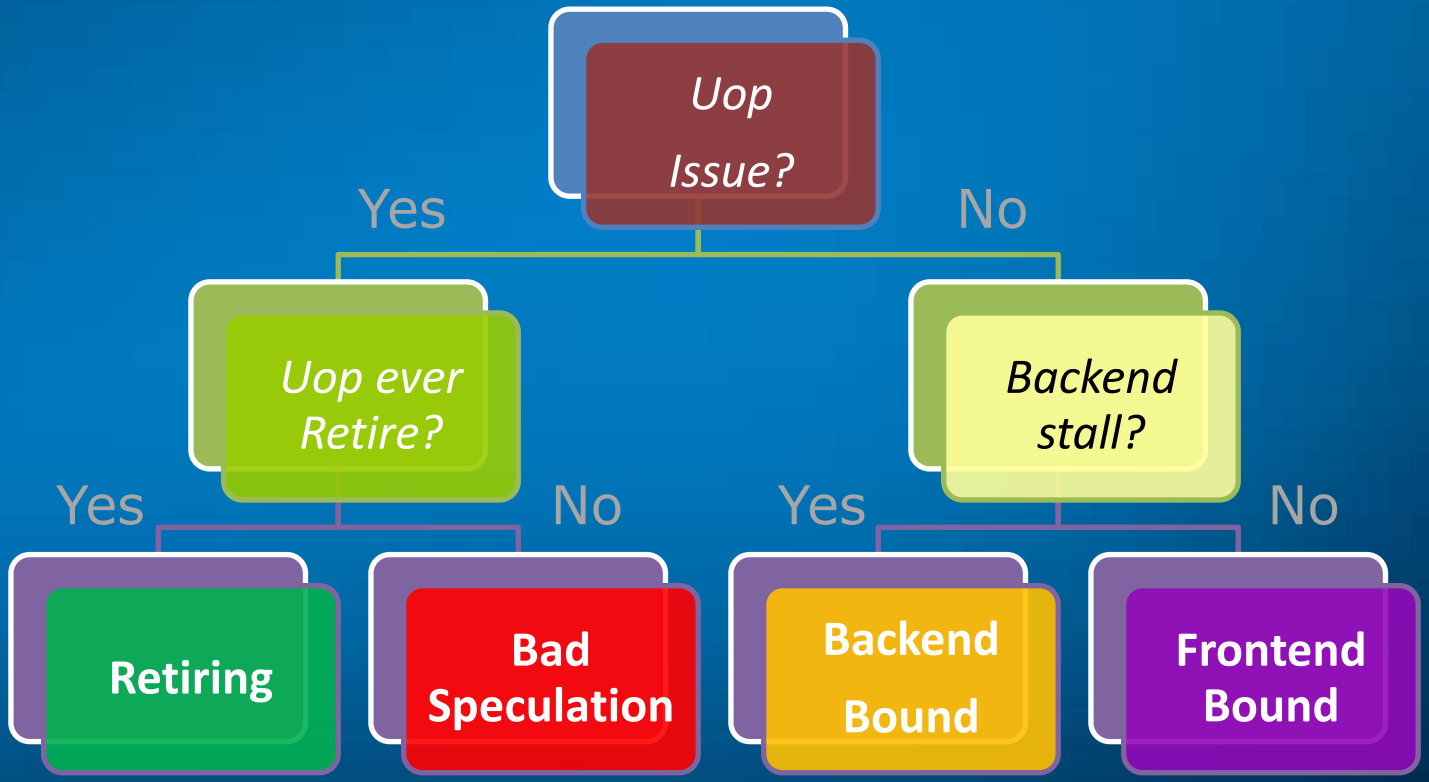
The Hierarchy



The Hierarchy w/ weights



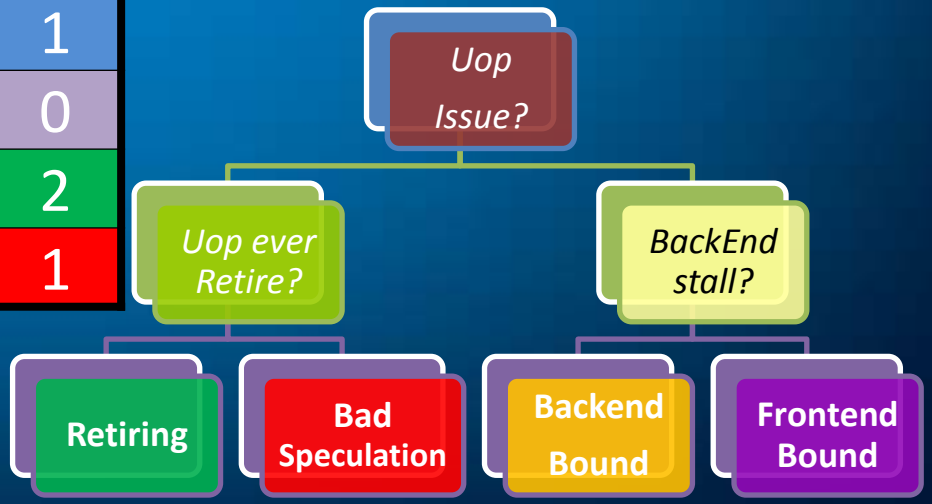
Top Level Breakdown – the idea



Ahmad Yasin -- A Top-Down Method for Performance Analysis and Counters Architecture (ISPASS 2014)

Top Level Breakdown

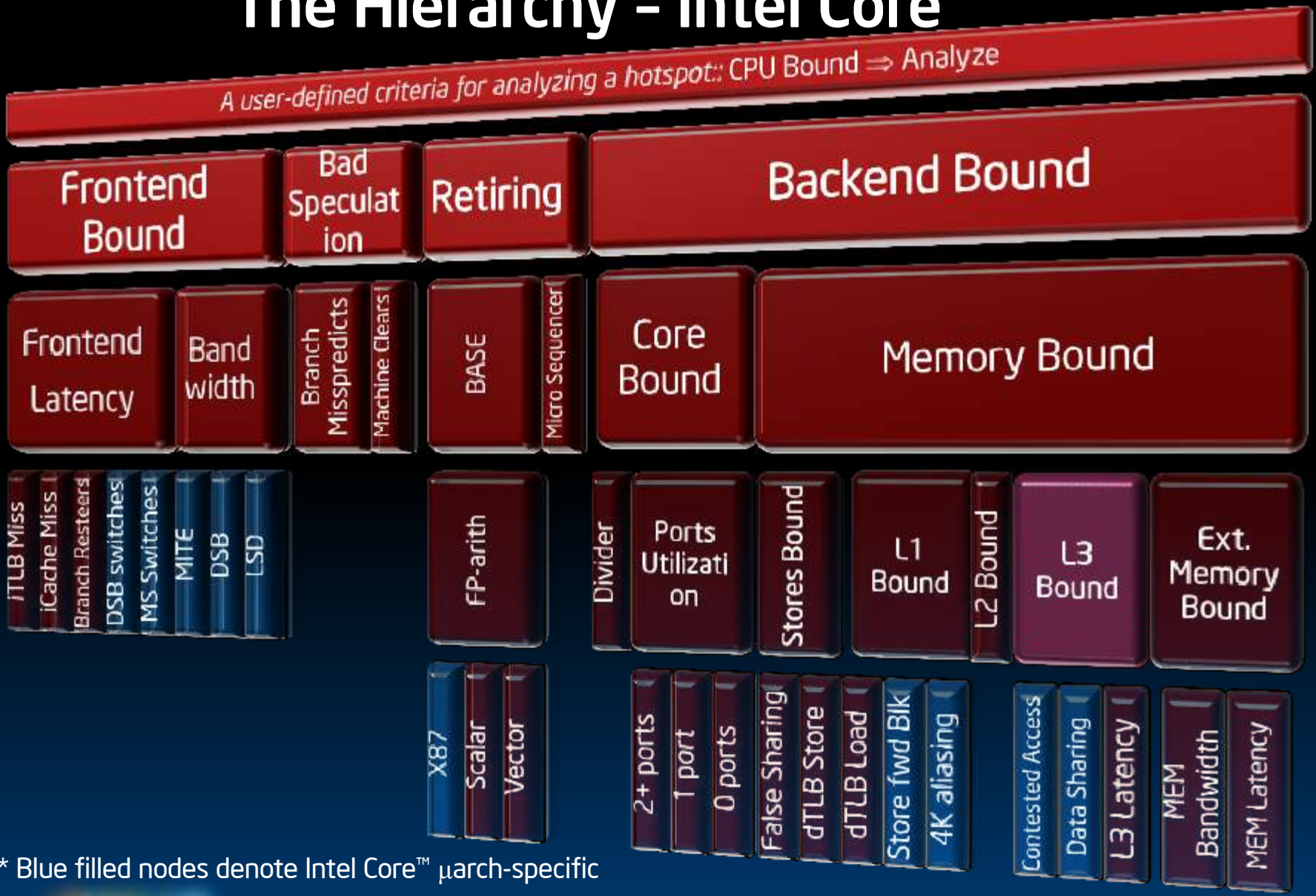
Cycle	1	2	3	4	5
Backend Stall	*0*	*0*	*1*	*0*	*0*
Issue Slot 0	-	v	-	v	v
Issue Slot 1	-	v	-	v	v
Issue Slot 2	-	v	-	v	v
Issue Slot 3	-	-	-	v	-
Frontend Bound	4	1		0	1
Backend Bound			4	0	0
Retiring		3		1	2
Bad Speculation				3	1



Classify Each Pipeline Slot Into 1 of 4 Categories

Ahmad Yasin -- A Top-Down Method for Performance Analysis and Counters Architecture (ISPASS 2014)

The Hierarchy – Intel Core™



* Blue filled nodes denote Intel Core™ μarch-specific

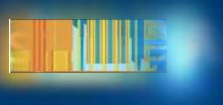
Counters Architecture

- Assume a baseline common PMU
  - Few general counters can count many perf events
- A dozen perf. events are required to feature *key* hierarchy nodes
  - Just 8 are new TopDown events, rest are in PMU already
- Example: Frontend Bound
  - TopDown Events
    - `FetchBubbles`: Unutilized issue-pipeline slots AND there is no Backend-stall
    - `TotalSlots`: Total number of issue-pipeline slots (e.g. Intel: `4*Clockticks`)
  - TopDown Metric
    - `Frontend Bound = FetchBubbles / TotalSlots`

Top Level Events and Metrics

Event Name	Definition	Intel Core™ PMU event name
TotalSlots*	Total number of issue-pipeline slots.	4*CPU_CLK_UNHALTED.THREAD
SlotsIssued*	Utilized issue-pipeline slots to issue operations	UOPS_ISSUED.ANY
SlotsRetired*	Utilized issue-pipeline slots to retire (complete) operations	UOPS_RETIRED.RETIRE_SLOTS
FetchBubbles	Unutilized issue-pipeline slots while there is no backend-stall	IDQ_UOPS_NOT_DELIVERED.CORE
RecoveryBubbles	Unutilized issue-pipeline slots due to recovery from earlier miss-speculation	4*INT_MISC.RECOVERY_CYCLES

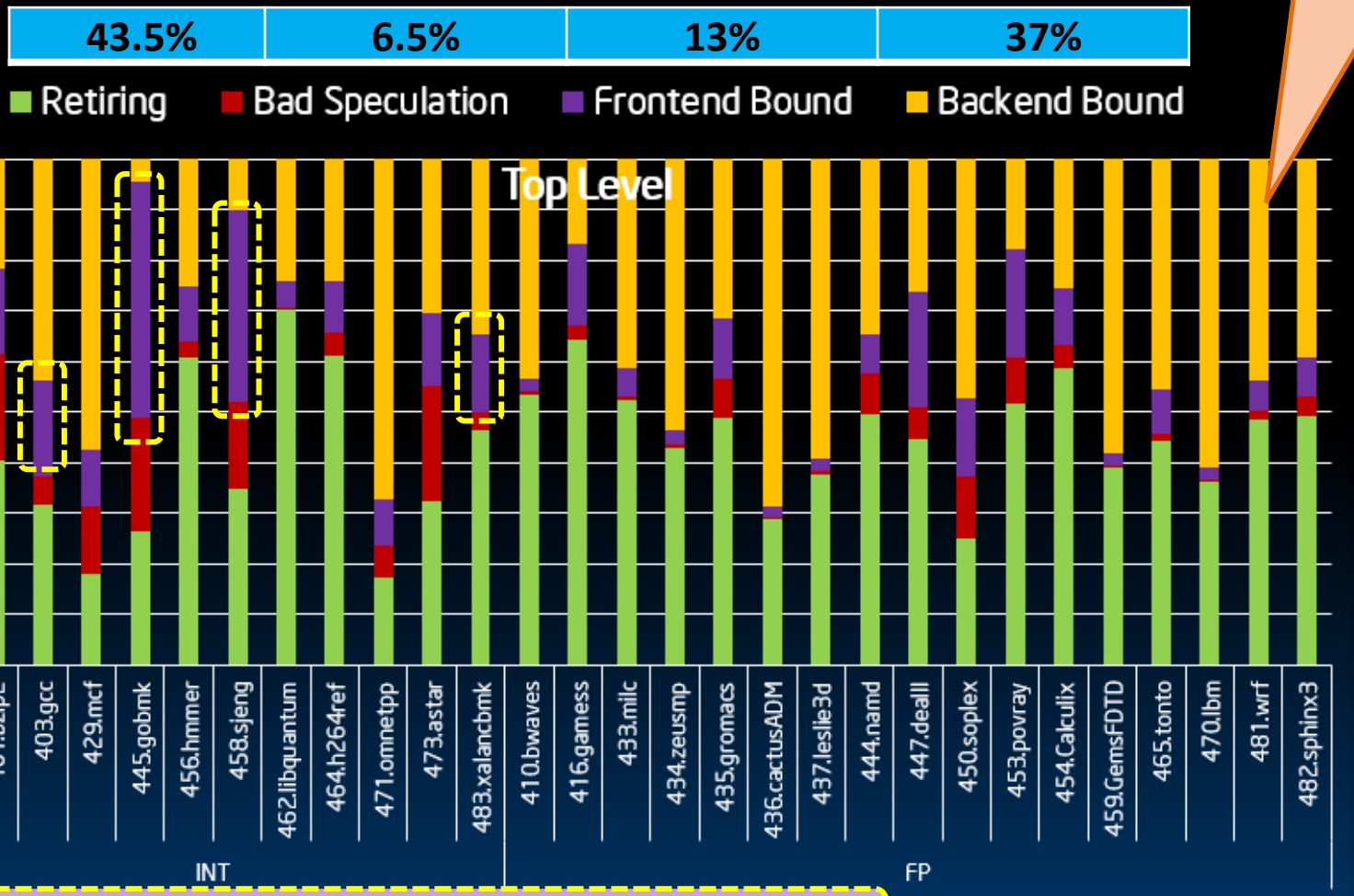
Metric Name	Definition	Formula
Frontend Bound	Frontend delivers < 4 uops per cycle while Backend is ready to accept uops	FetchBubbles / TotalSlots
Bad Speculation	Tracks uops that never retire or slots wasted due to recovery from clears	(SlotsIssued – SlotsRetired + RecoveryBubbles) / TotalSlots
Retiring	Successfully delivered uops who eventually do retire	SlotsRetired / TotalSlots
Backend Bound	No uops were delivered due to lack of Backend resources	1 – (Frontend Bound + Bad Speculation + Retiring)



Top Level for SPEC CPU2006

INT apps have quiet some Frontend/Bad Spec. issues

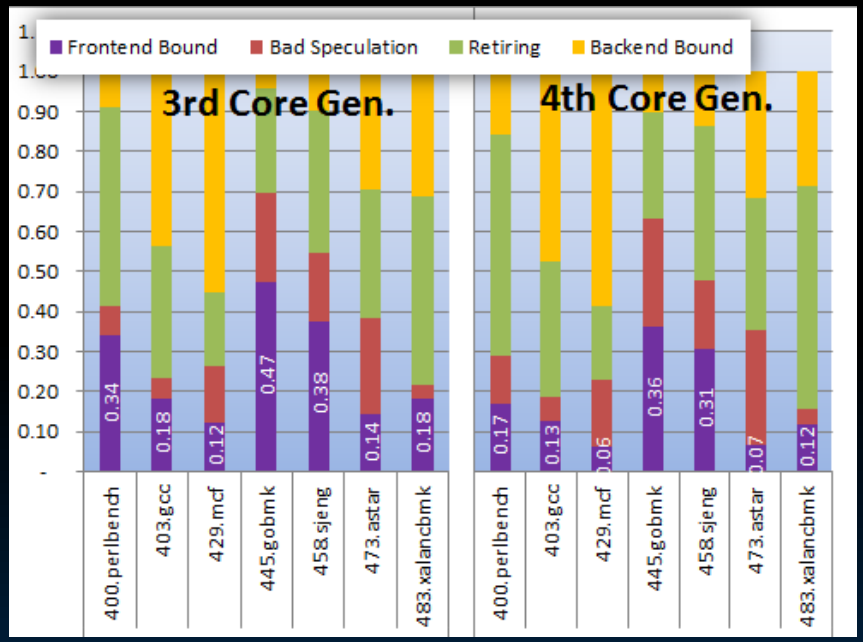
Most apps are Backend Bound, esp. FP



e.g. [11] reported perlbench, gcc, xalancbmk, gobmk, sjeng have >32KB code footprint

Across microarchitecture's support

- Haswell (4th Core gen) has improved front-end
  - Speculative iTLB and cache accesses with better timing to improve the benefits of prefetching
- Benefiting benchmarks clearly show reduction in Frontend Bound



Top Down Analysis forward compatibility on Intel Core™

Memory Bound (1-core vs 4-core)

Case Study: Matrix Multiply

- A kernel is *iteratively* analyzed with Top-Down
  - Big matrices in memory
    - multiply1 MEM Bound
  - Loop Interchange: 12x
    - multiply2 becomes CoreBound due to execution ports utilization
  - Vectorization: 17x
    - multiply3 mitigated CoreBound.

Metric	multiply1	multiply2	multiply3
Speedup	1.0x	11.8x	16.5x
IPC	0.17	1.19	0.80
Frontend Bound	0.00	0.07	0.02
Retiring	0.05	0.41	0.28
Bad Speculation	0.00	0.00	0.00
Backend Bound	**0.95**	**0.52**	**0.70**
├─+ Memory Bound	0.84	0.12	0.31
│ ├── L1 Bound	0.05	0.07	0.03
│ ├── L2 Bound	0.03	-	0.05
│ ├── L3 Bound	0.05	-	0.01
│ ├── MEM Bound	0.71	0.07	0.21
│ └── Stores Bound	-	-	-
└─+ Core Bound	0.15	0.64	0.55
├── Divider	-	-	-
└── Ports Utiliz.	0.15	0.64	0.55

Related Work

- [4][5] use naïve-approach
- [6] IBM POWER5
  - CPI Breakdown at commit-stage
  - Stalls-periods counted per type of next instruction
- [5] Cycle Accounting (x-Intel)
  - A flat breakdown at execution-stage
- [1][6][8] CPI stacks
  - A simulation-based interval analysis improves over [4][6]
  - High hardware cost as authors admit in [8]
  - [8] requires extra logic for penalty calculation & aggregation in dedicated counters
- [12][13] data-locality and scalability bottlenecks
  - Use instrumentation- and simulation-based tools
  - Advanced optimization-specific techniques; could be invoked from Top Down once Memory Bound is flagged

Summary

- Top Down Analysis Method
  - Identifies critical bottlenecks
  - Simple, Structured, Quick
- Demonstrated results
  - On many workloads
  - In-production. e.g. VTune™, perf *
  - Forward compatibility in Intel cores
- Counters Architecture
  - For a generic out-of-orders
  - Low cost: 8 simple events
  - Standardization across platforms

Check out the paper and send us your feedback