It's the Memory, Stupid!

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MICRO-2026

TopDown Microarchitectural Analysis (TMA)

Based on "A Top-Down Method for Performance Analysis and Counters Architecture" paper by **Ahmad Yasin**

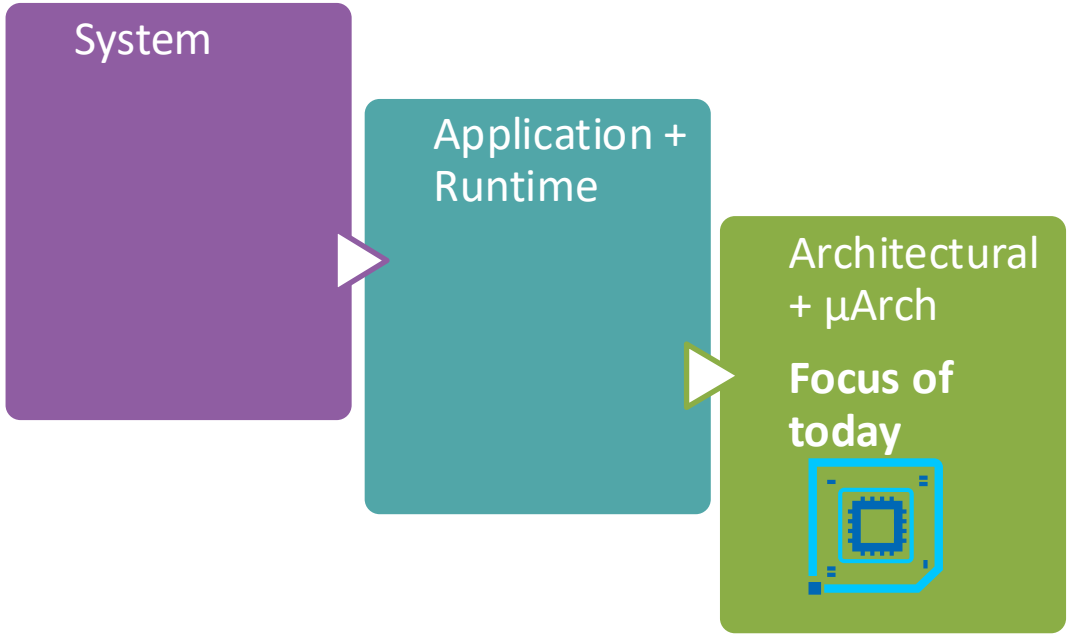
Anton Hanna
CPU Core Architect – Intel

Victor Xirau
BSC

intel

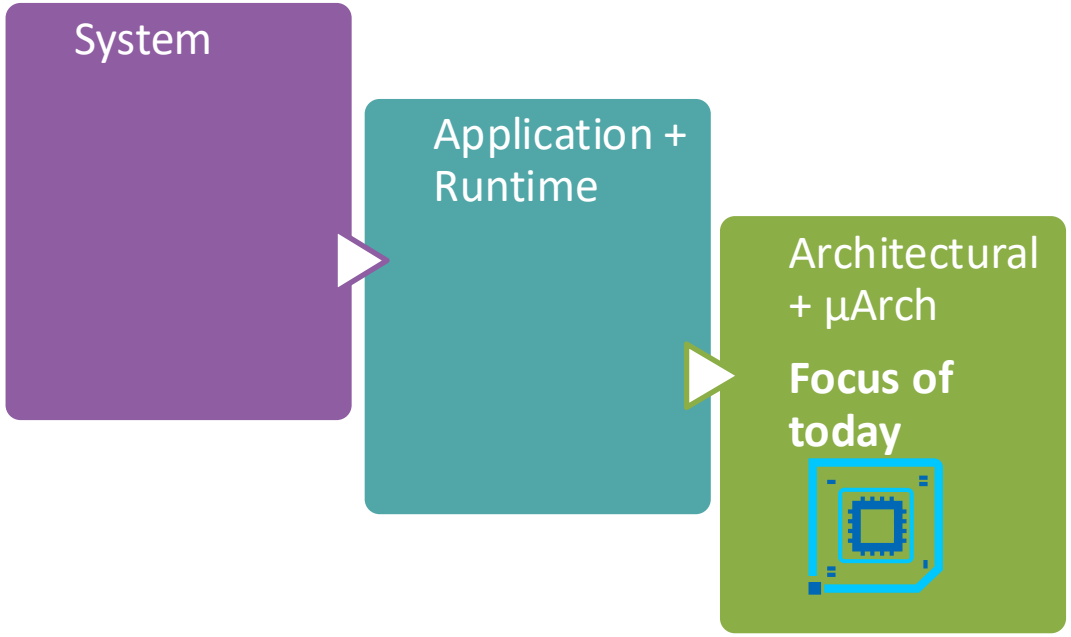
31st of October 2026.

Performance Analysis – Definition



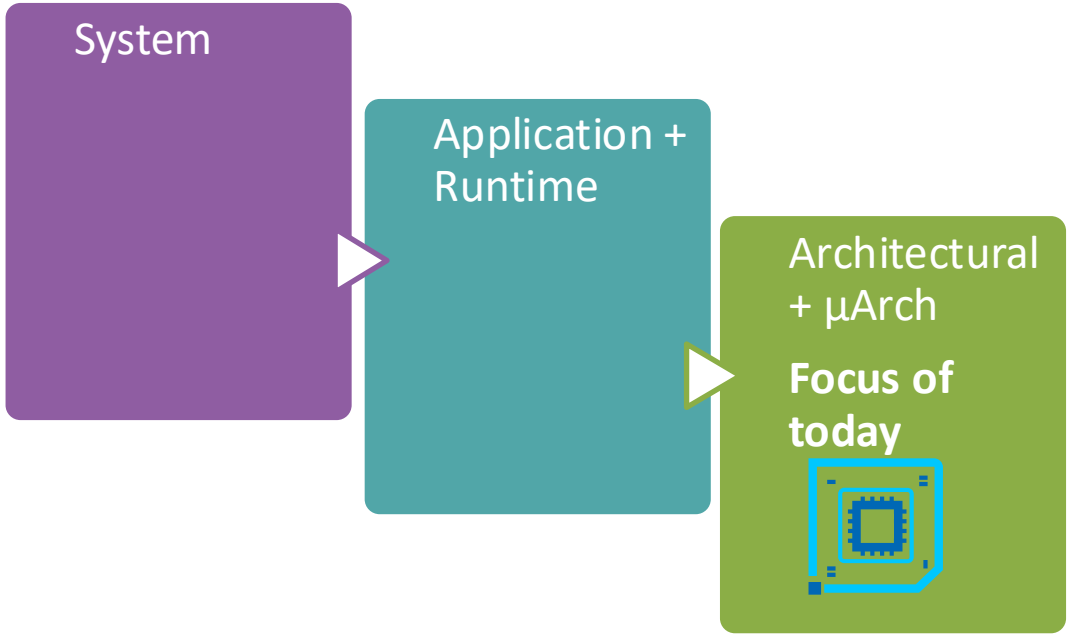
Performance Analysis – Definition

- A performance analysis methodology is a procedure that you can follow to analyze system or application performance.



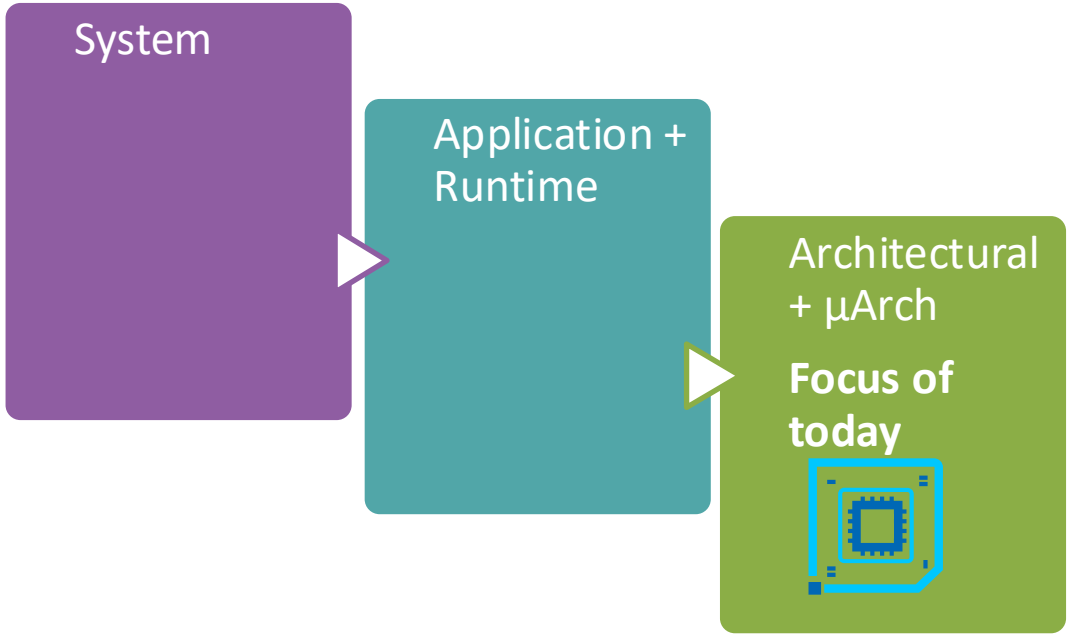
Performance Analysis – Definition

- A performance analysis methodology is a procedure that you can follow to analyze system or application performance.
- Generally, provides a starting point and then guidance to root cause, or causes.



Performance Analysis – Definition

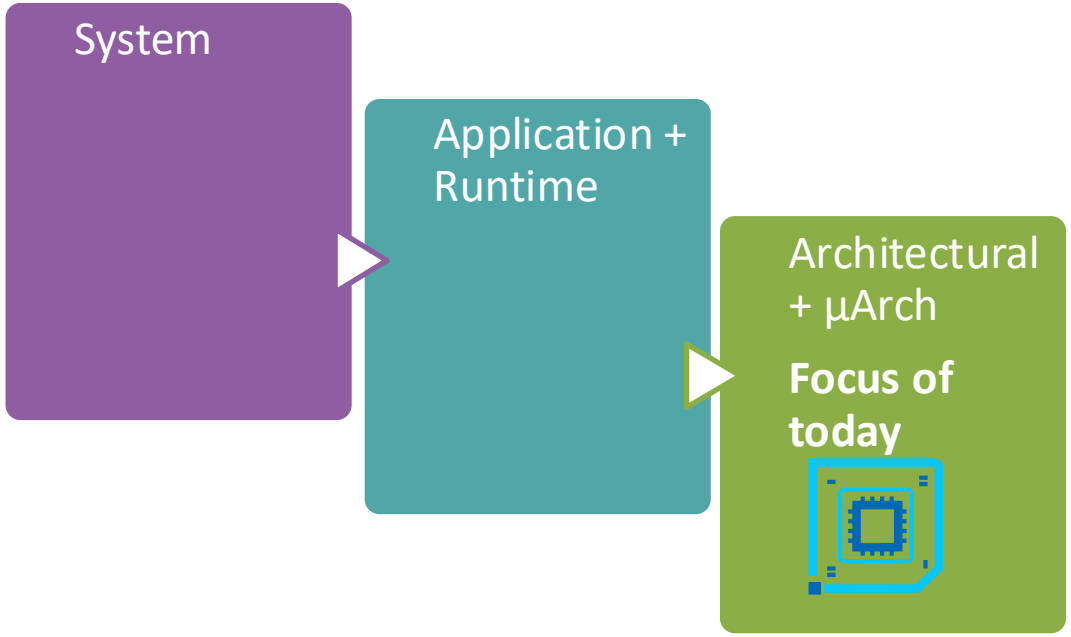
- A performance analysis methodology is a procedure that you can follow to analyze system or application performance.
- Generally, provides a starting point and then guidance to root cause, or causes.
- Different Methodologies are suited for solving different classes of issues
  - May try more than one before accomplishing your goal.



Performance Analysis – Definition

- A performance analysis methodology is a procedure that you can follow to analyze system or application performance.
- Generally, provides a starting point and then guidance to root cause, or causes.
- Different Methodologies are suited for solving different classes of issues
  - May try more than one before accomplishing your goal.
- Analysis without methodology can become a fishing expedition, where metrics are examined ad hoc, until the issue is found – if it is at all

from: Performance Analysis Methodology by Brendan Gregg



Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x		
Loop intercha			
Cache Blocking (L2 Cache			
Parallelization (x			
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	
Loop intercha			
Cache Blocking (L2 Cache			
Parallelization (x			
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop intercha			
Cache Blocking (L2 Cache			
Parallelization (x			
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15		
Cache Blocking (L2 Cache			
Parallelization (x			
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	
Cache Blocking (L2 Cache			
Parallelization (x			
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache			
Parallelization (x			
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x		
Parallelization (x			
Parallelization (x2			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	
Parallelization (x			
Parallelization (x2			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x			
Parallelization (x2			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x		
Parallelization (x24)			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24			
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x		
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x	Core Bound	
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x	Core Bound	Improve in-core parallelism
Vectorization (AVX512)			

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x	Core Bound	Improve in-core parallelism
Vectorization (AVX512)	1600X		

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x	Core Bound	Improve in-core parallelism
Vectorization (AVX512)	1600X	Cache Latency Bound	

Why should SW developers care about performance analysis ?

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x	Core Bound	Improve in-core parallelism
Vectorization (AVX512)	1600X	Cache Latency Bound	

Tuning Performance via Metrics with Expectations | IEEE Journals & Magazine | IEEE Xplore

Why should SW developers care about perfor…?

This is largely solved

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	Increase compute per byte
Loop interchange	15x	Memory Latency Bound	Block data into caches
Cache Blocking (L2 Cache)	21x	Compute Bound	Utilize Multi-Core
Parallelization (x4)	78x	Compute Bound	Utilize Multi-Core
Parallelization (x24)	430x	Core Bound	Improve in-core parallelism
Vectorization (AVX512)	1600X	Cache Latency Bound	

Tuning Performance via Metrics with Expectations | IEEE Journals & Magazine | IEEE Xplore

Why should SW developers care about perfor[illegible]?

This is largely solved

SW Version	Speedup	Bottleneck	Optimization Hint
Baseline	1x	External Memory	[illegible]crease compute per byte
Loop interchange	15x	Memory Latency Bound	
Cache Blocking (L2 Cache)	21x	Compute Bound	
Parallelization (x4)	78x	Compute Bound	[illegible] Core
Parallelization (x24)	430x	Core Bound	Improve in-core parallelism
Vectorization (AVX512)	1600X	Cache Latency Bound	

Very large number of potential optimizations

Tuning Performance via Metrics with Expectations | IEEE Journals & Magazine | IEEE Xplore

Why should SW developers care about performance analysis ?

Organization & Workload	Tools and Analysis Methods	Key PerfMon Metrics	Optimization Actions	Performance Gains
Netflix – Java microservice (AWS)	Intel *VTune Profiler*; Intel *PerfSpect* (PMU counter collection)[1] [2]; Top-Down microarchitecture analysis	**Cycles per Instruction (CPI)**, L1 cache misses, **MACHINE_CLEARS** (cache invalidations)[3]; *Front-End vs Back-End Bound*	Identified **false sharing** in JVM data structure and **true sharing** due to writes to a shared cache; inserted **cache-line padding** and **disabled a problematic JDK cache**[4] [5] to eliminate both false and true sharing	**3.5× throughput increase** (from ~150 RPS to ~350 RPS per instance) at same CPU utilization, plus **lower latency**[6] [7]
Taboola – AI recommendation engine (deep learning inference)	Intel *VTune Profiler* and analysis on new Intel Xeon Platinum CPUs[8]; Code profiling with PMU events guiding optimization	**Function hotspots** (using sampling via PMCs); HW utilization (vectorization, **AVX** usage); concurrency profile	Upgraded to new Intel Xeon with **wider vector units** & memory bandwidth; leveraged **Intel MKL** for math routines; **refactored hot code** identified by VTune's PMU-based analysis[9]	**Up to +149%** throughput in deep-learning inference (2.49× speedup in content recommendations per server)

Why should SW developers care about performance analysis ?

Organization & Workload	Tools and Analysis Methods	Key PerfMon Metrics	Optimization Actions	Performance Gains
Verizon – Enterprise server workloads (telecom apps)	Intel *VTune Amplifier* (including **cache and thread analysis**), Intel *Platform Profiler*[10] [11]	**Cache miss ratios**; **CPU concurrency and lock contention**; thread hotspots	Diagnosed **NUMA memory contention** and inefficient lock usage in legacy C++/Java/Node.js code[12]; applied **small code fixes** (e.g. improved memory locality, reduced lock holds) via PEG (Performance Engineering Group)	**Restored expected performance** (eliminated "X vs Y" gap) for critical services[13] [14]; *millions of dollars* saved by consolidating servers[15]
HPC – Scientific computing code (matrix math)	Linux *perf* + pmu-tools (Top-Down metrics); Spectral decomposition of cycles via **Top-Down PMCs**	**Retiring vs. stalled cycles** breakdown (Retiring, Front-End Bound, Back-End Bound)[16]; branch mispredictions, memory misses	Found code was **memory-bound** (e.g. many **stall cycles** waiting on memory) due to loop order; applied a **loop interchange** optimization and **vectorization** to reduce memory stalls	**Multi-fold speedups**: e.g. in a class demo, loop interchange + vectorization shifted a code's bottleneck and **yielded large speedups** (illustrating how Top-Down PMCs guide HPC code tuning)

Why should SW developers care about performance analysis ?

Organization & Workload	Tools and Analysis Methods	Key PerfMon Metrics	Optimization Actions	Performance Gains
Intel IBOT (Intel® Binary Optimization Tool: Enhanced Performance for Gaming**)**	Intel *EMON EDP and Vtune*	**Lock Contention Stalls, Forward Branch Mispredict Ratio** and many more…	Diagnosed **various AAA gaming title performance hiccups,** root-causing many issues, allowing to develop the binary translator	**Massive gains on some popular games performance on specific Intel Core Ultra (ARL) processors.**

PerfMon – Counters & More

Performance Monitoring Unit (PMU) capabilities on Intel Core

Performance Monitoring Unit

Performance Monitoring Unit

- PMU – Performance Monitoring Unit

Performance Monitoring Unit

- PMU – Performance Monitoring Unit
  - Brings a set of capabilities to retrieve info on CPU execution
  - Mostly on microarchitecture performance, but could also cover arch state, energy and more
  - **Non-intrusive** – very low over-head
  - **Isolation** - Per Logical Processor unit

Performance Monitoring Unit

- PMU – Performance Monitoring Unit
  - Brings a set of capabilities to retrieve info on CPU execution
  - Mostly on microarchitecture performance, but could also cover arch state, energy and more
  - **Non-intrusive** – very low over-head
  - **Isolation** - Per Logical Processor unit

- Basic Building Blocks:

Performance Monitoring Unit

- PMU – Performance Monitoring Unit
  - Brings a set of capabilities to retrieve info on CPU execution
  - Mostly on microarchitecture performance, but could also cover arch state, energy and more
  - **Non-intrusive** – very low over-head
  - **Isolation** - Per Logical Processor unit

- Basic Building Blocks:
  - **Performance Counters**: Fixed + Configurable
  - Interrupt/Assist Generation (on counter overflow)
  - HW Latency counting and logging mechanisms
  - Configuration MSRs (SW configurable Control Registers)
  - Micro-code (FW) support

Performance Monitoring Unit

- PMU – Performance Monitoring Unit
  - Brings a set of capabilities to retrieve info on CPU execution
  - Mostly on microarchitecture performance, but could also cover arch state, energy and more
  - **Non-intrusive** – very low over-head
  - **Isolation** - Per Logical Processor unit

- Basic Building Blocks:
  - **Performance Counters**: Fixed + Configurable
  - Interrupt/Assist Generation (on counter overflow)
  - HW Latency counting and logging mechanisms
  - Configuration MSRs (SW configurable Control Registers)
  - Micro-code (FW) support

- Terminology
  - General Purpose **Counter**
  - Predefined HW **Even**
  - **Counting vs. Sampling Modes**
  - **PMI** – Performance Monitoring Interrupt
  - **PEBS** – Processor Event Based Sampling
    - Samples
    - Reload Value (Sample After)
  - **LBR** – Last Branch Record

Performance Monitoring Unit

Performance Monitoring Unit

- Architectural PMU on Intel P-Core includes
  - 4 fixed counters: Instructions Retired, Core Unhalted Cycles, Reference Unhalted Cycles and TMA Slots
  - 8 general-purpose counters – configured to count events by SW
  - Global Status and Controls

Performance Monitoring Unit

- Architectural PMU on Intel P-Core includes
  - 4 fixed counters: Instructions Retired, Core Unhalted Cycles, Reference Unhalted Cycles and TMA Slots
  - 8 general-purpose counters – configured to count events by SW
  - Global Status and Controls
- Intel P-Core brings a rich list of performance monitoring events
  - Most events are model specific
  - 12 events are architectural (since RedwoodCove core – MTL/GNR)

Performance Monitoring Unit

- Architectural PMU on Intel P-Core includes
  - 4 fixed counters: Instructions Retired, Core Unhalted Cycles, Reference Unhalted Cycles and TMA Slots
  - 8 general-purpose counters – configured to count events by SW
  - Global Status and Controls
- Intel P-Core brings a rich list of performance monitoring events
  - Most events are model specific
  - 12 events are architectural (since RedwoodCove core – MTL/GNR)
- Advanced mechanisms
  - PEBS – precisely tags performance events and profiling data on software location
  - LBR – Non-intrusive branch recording facility

PerfMon Events

Links

PerfMon Events

Links

- Database of Intel PerfMon events and metrics: [intel/perfmon]

PerfMon Events

Links

- Database of Intel PerfMon events and metrics: [intel/perfmon]
- Categorized by CPU family and Core Type (for hybrid CPUs)

PerfMon Events

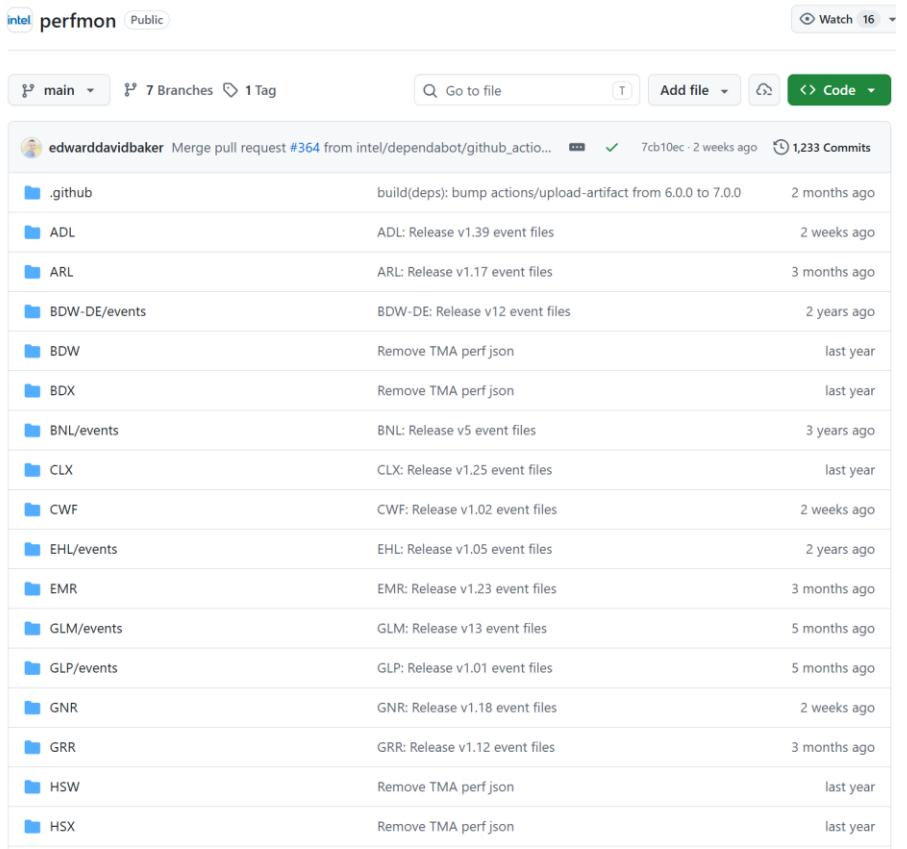
Links

- Database of Intel PerfMon events and metrics: [intel/perfmon]
- Categorized by CPU family and Core Type (for hybrid CPUs)
- Includes uncore (SoC level) events:
  - performance monitoring events counted at SoC level in various locations
  - Are not attributed to one core/process

PerfMon Events

Links

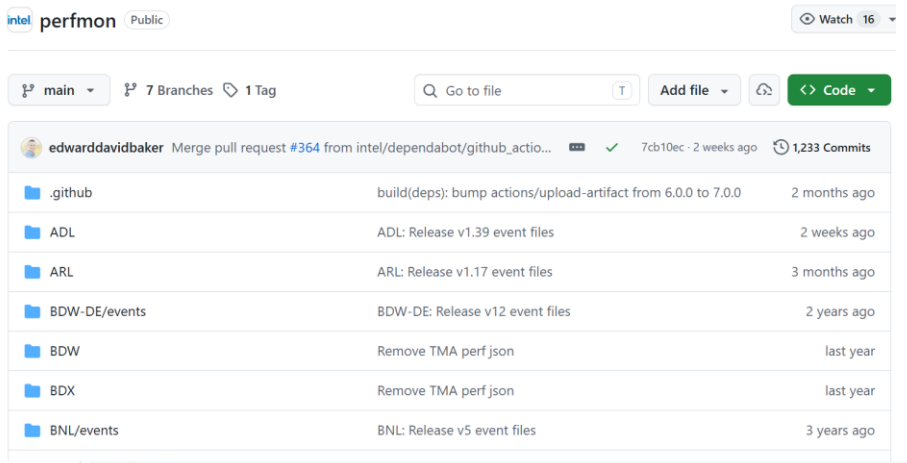
- Database of Intel PerfMon events and metrics: [intel/perfmon]
- Categorized by CPU family and Core Type (for hybrid CPUs)
- Includes uncore (SoC level) events:
  - performance monitoring events counted at SoC level in various locations
  - Are not attributed to one core/process

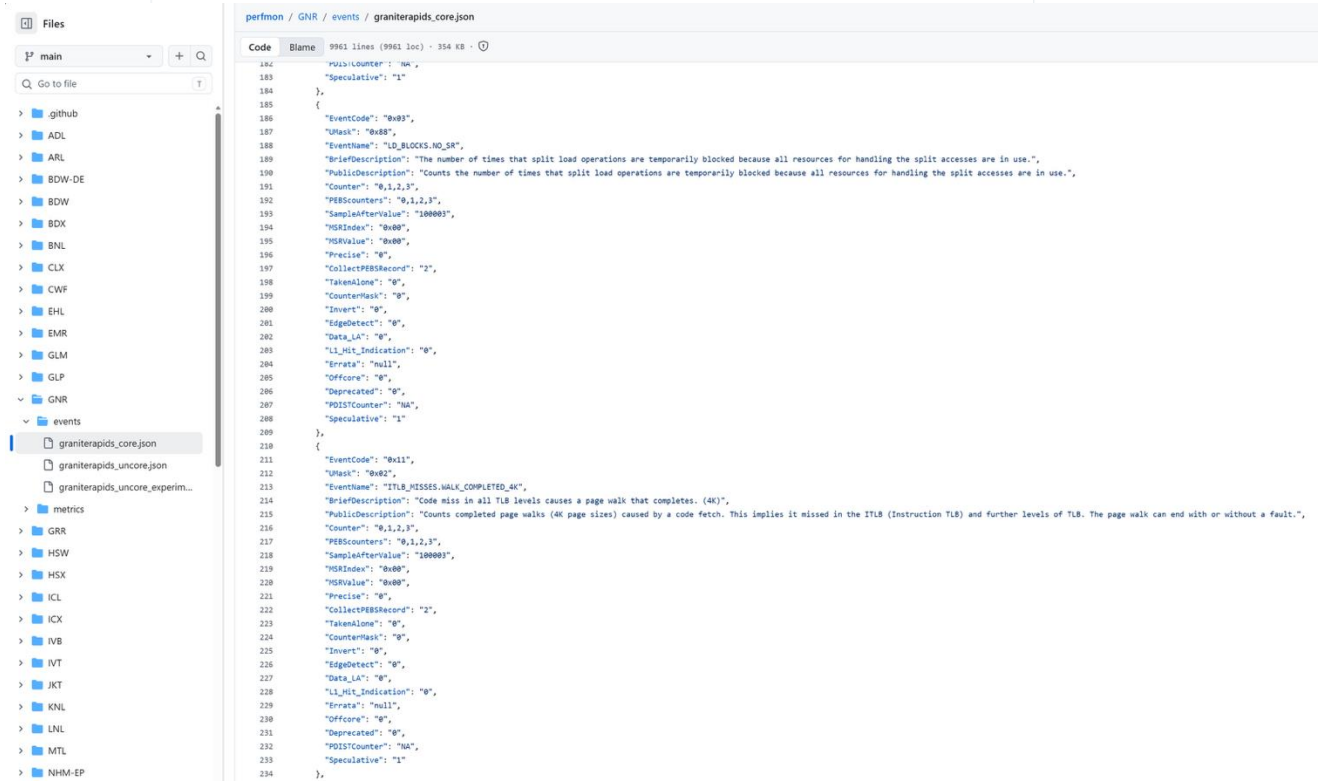


PerfMon Events

Links

- Database of Intel PerfMon events and metrics: intel/perfmon
- Categorized by CPU family and Core Type (for hybrid CPUs)
- Includes uncore (SoC level) events:
  - performance monitoring events counted at SoC level in various locations
  - Are not attributed to one core/process





Challenges in performance analysis

Out-Of-Order (OOO) execution cores are complex monsters!

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis

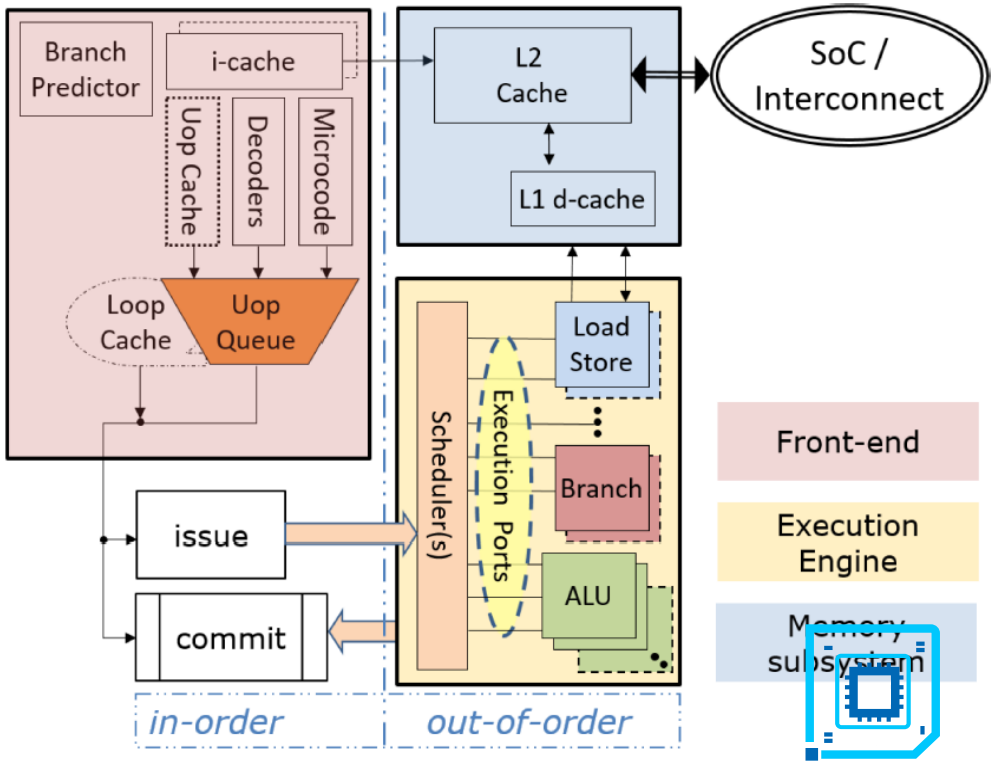


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:

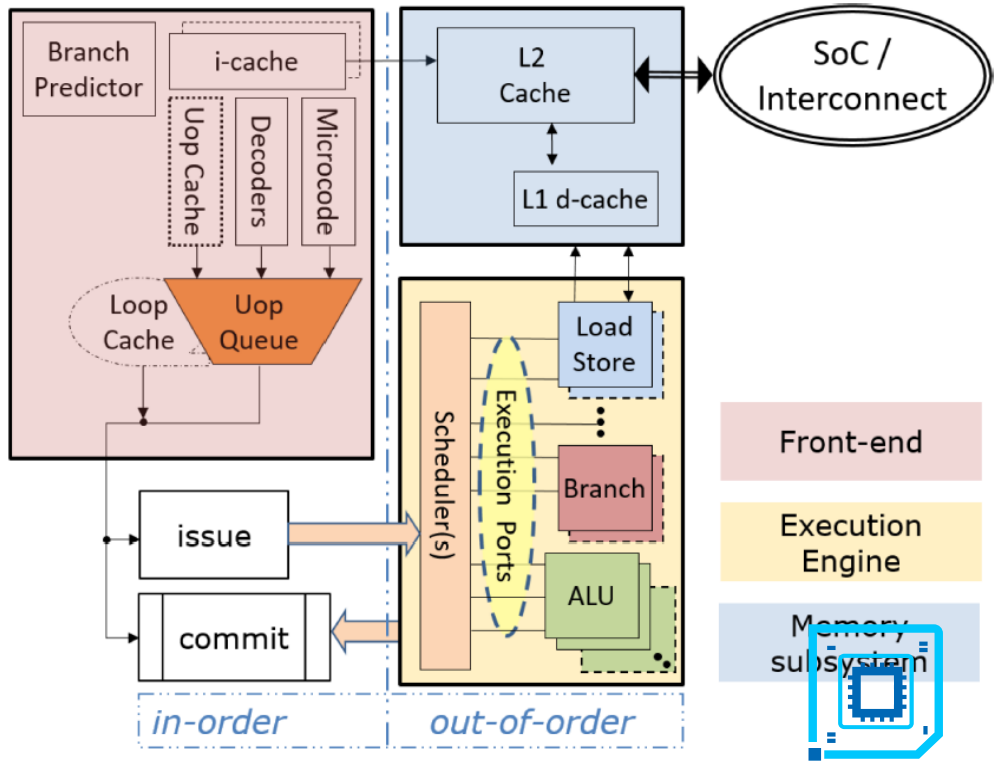


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:
  - Pipelined Superscalars

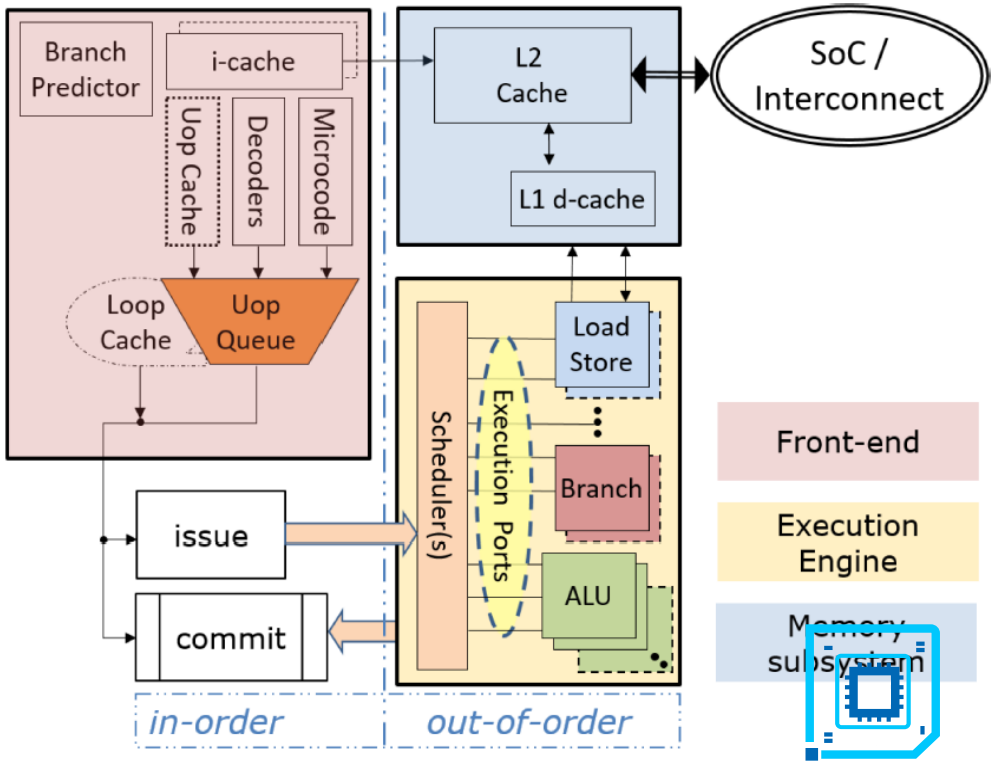


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:
  - Pipelined Superscalars
  - OOO Execution

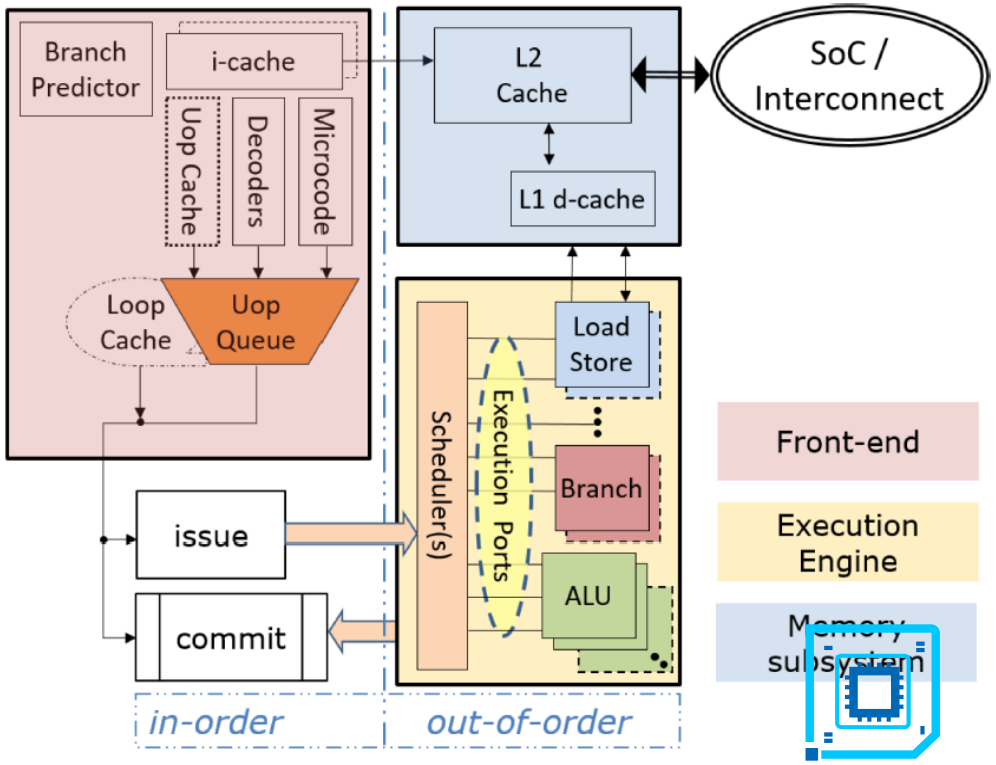


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:
  - Pipelined Superscalars
  - OOO Execution
  - Speculative Execution

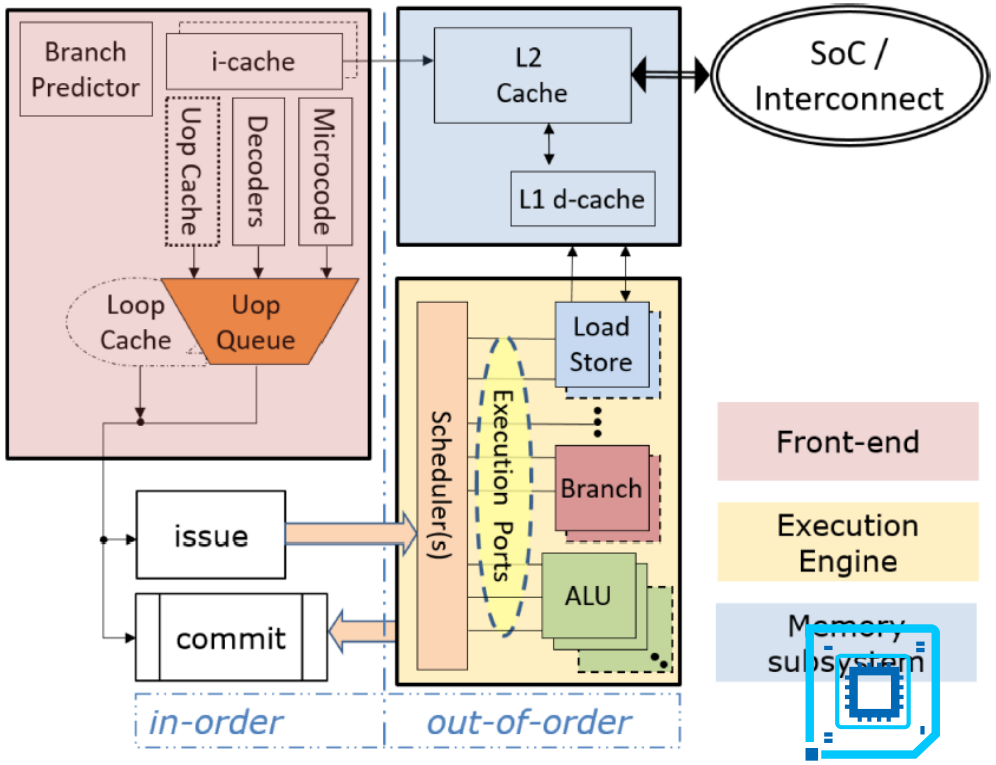


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:
  - Pipelined Superscalars
  - OOO Execution
  - Speculative Execution
  - Multi-Level Hierarchical Caches

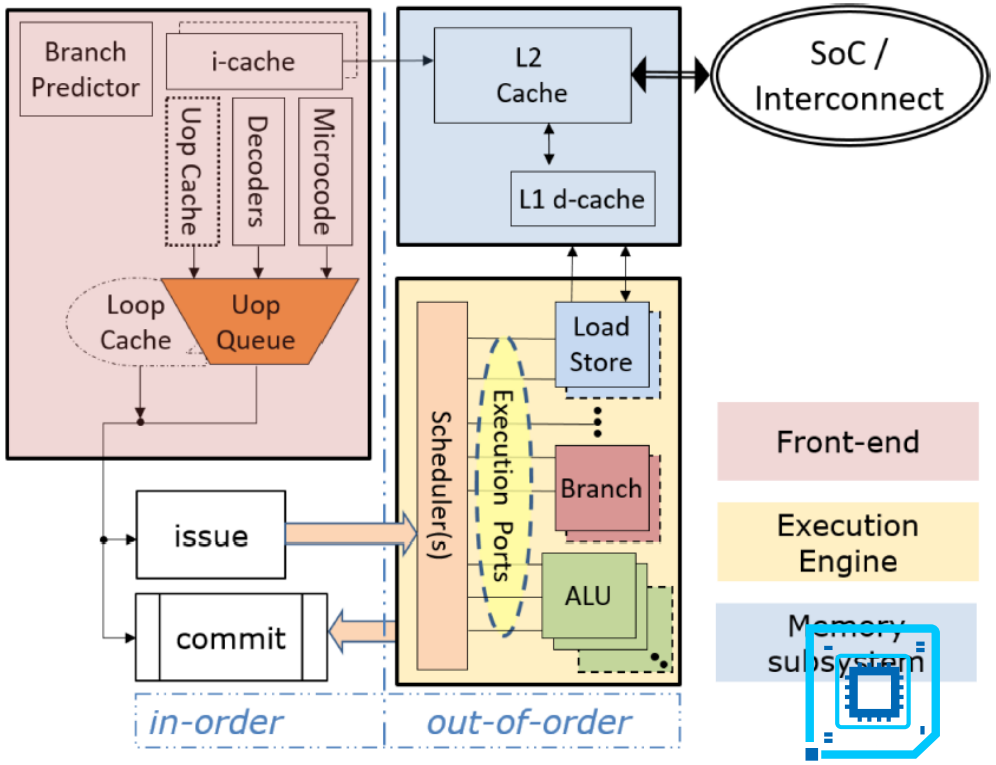


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:
  - Pipelined Superscalars
  - OOO Execution
  - Speculative Execution
  - Multi-Level Hierarchical Caches
  - OOO Memory Accesses

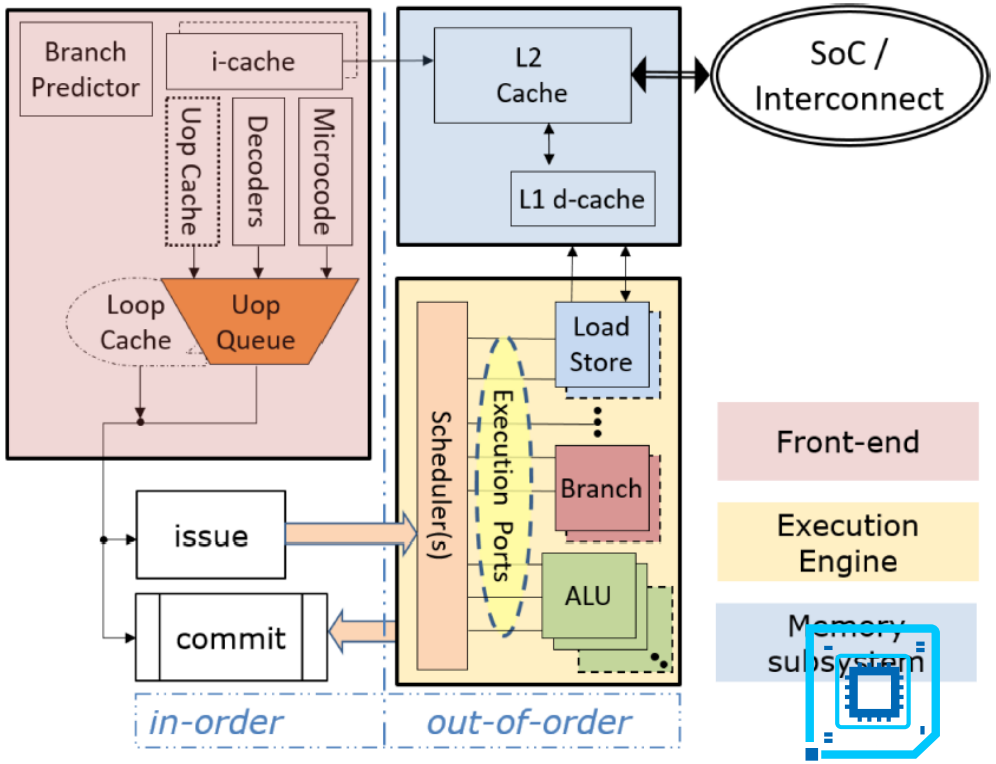


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Challenges in CPU Core Performance Analysis

- Modern CPUs are very challenging for performance analysis
- Utilizes very complicated mechanisms:
  - Pipelined Superscalars
  - OOO Execution
  - Speculative Execution
  - Multi-Level Hierarchical Caches
  - OOO Memory Accesses
  - Various execution stacks (Integer, Vector and Matrix)

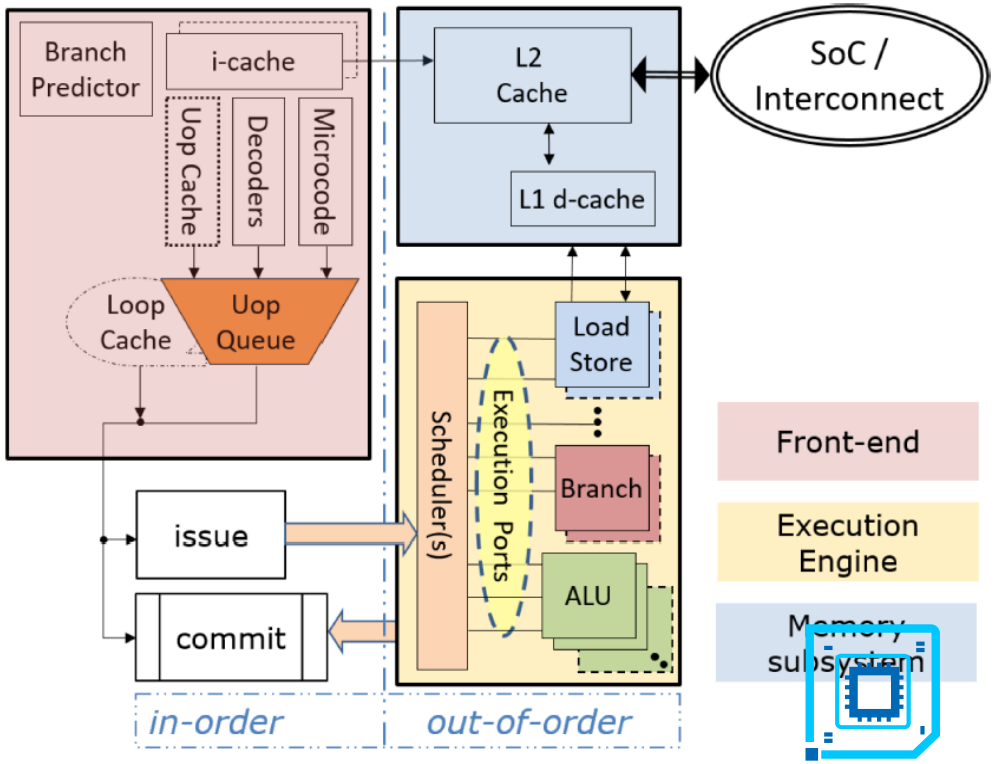


Fig. 3. Abstraction of processor core microarchitecture.

Fig. 3 source: [2] A Metric-Guided Method for Discovering Impactful Features and Architectural Insights for Skylake-Based Processors. Ahmad Yasin, Jawad Haj-Yahya, Yosi Ben-Asher, Avi Mendelson. In ACM Transactions on Architecture and Code Optimization (TACO), Dec. 2019

Good Ol’ Metrics

- Having a lot of events to count is useful, however, event counts on their own are mostly not indicative of performance

Good Ol’ Metrics

- Having a lot of events to count is useful, however, event counts on their own are mostly not indicative of performance
  - My workload did 200B instructions!! Wait is that good ?

Good Ol’ Metrics

- Having a lot of events to count is useful, however, event counts on their own are mostly not indicative of performance
  - My workload did 200B instructions!! Wait is that good ?
  - My workload took 100 secs to finish !! Is that good ?

Good Ol' Metrics

- Having a lot of events to count is useful, however, event counts on their own are mostly not indicative of performance
  - My workload did 200B instructions!! Wait is that good ?
  - My workload took 100 secs to finish !! Is that good ?
- Hence metrics are needed!
  - Instructions Per Cycle – IPC = INST_RETIRED.ANY / CPU_CLK_UNHALTED.THREAD
  - Instructions per Mispredict – IpMisp = INST_RETIRED.ANY / BR_MISP_RETIRED.ALL_BRANCHES
  - And many more

Good Ol' Metrics

- Having a lot of events to count is useful, however, event counts on their own are mostly not indicative of performance
  - My workload did 200B instructions!! Wait is that good ?
  - My workload took 100 secs to finish !! Is that good ?
- Hence metrics are needed!
  - Instructions Per Cycle – IPC = INST_RETIRED.ANY / CPU_CLK_UNHALTED.THREAD
  - Instructions per Mispredict – IpMisp = INST_RETIRED.ANY / BR_MISP_RETIRED.ALL_BRANCHES
  - And many more
- Metrics are great, but without microarchitectural context, could be misleading

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

$$STALL = \Sigma\ event_i * penalty_i$$

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

$$STALL = \Sigma\ event_i * penalty_i$$

  - Example: L2 Cache Miss Penalty = 80 cycles * Load L2 Misses.

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

$$STALL = \Sigma\ event_i * penalty_i$$

  - Example: L2 Cache Miss Penalty = 80 cycles * Load L2 Misses.
- This approach does not work for moder OOO cores

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

$$STALL = \Sigma\ event_i * penalty_i$$

  - Example: L2 Cache Miss Penalty = 80 cycles * Load L2 Misses.
- This approach does not work for moder OOO cores
  - Overlapping stalls

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

$$STALL = \Sigma\ event_i * penalty_i$$

  - Example: L2 Cache Miss Penalty = 80 cycles * Load L2 Misses.
- This approach does not work for moder OOO cores
  - Overlapping stalls
  - Variable penalties

Stall Estimation

- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

$$STALL = \Sigma\ event_i * penalty_i$$

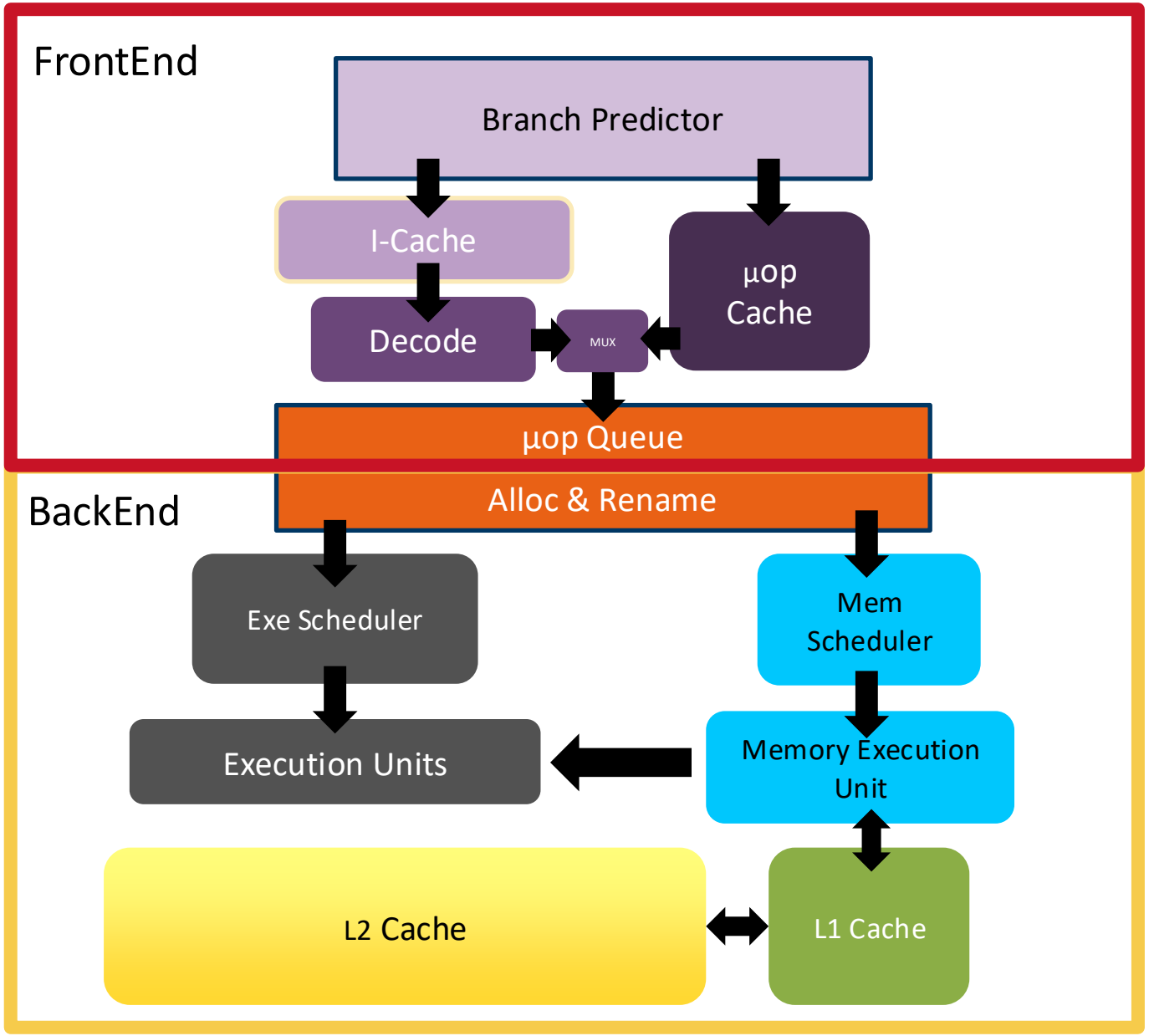
  - Example: L2 Cache Miss Penalty = 80 cycles * Load L2 Misses.
- This approach does not work for moder OOO cores
  - Overlapping stalls
  - Variable penalties
  - Speculative execution

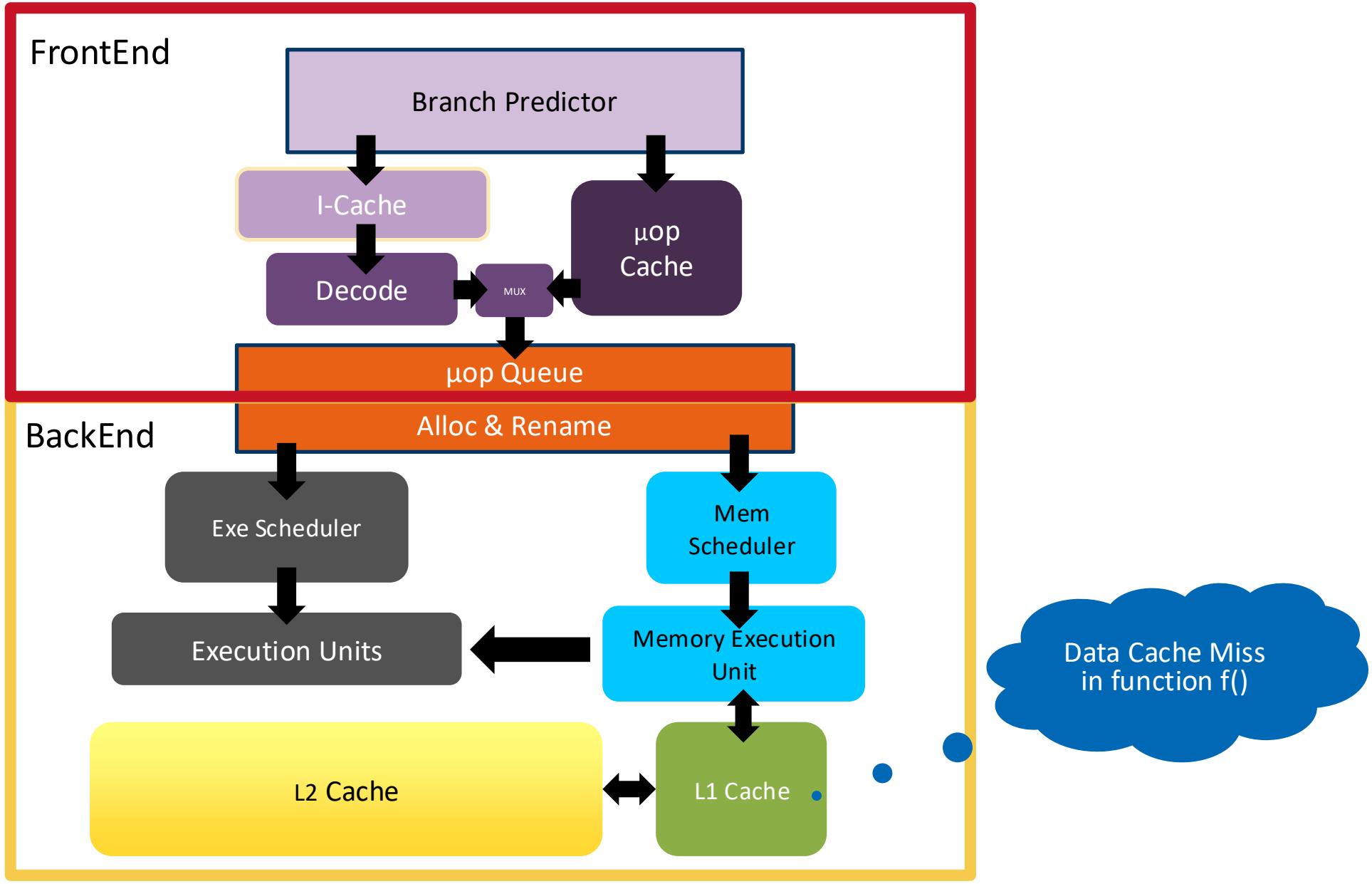
Stall Estimation

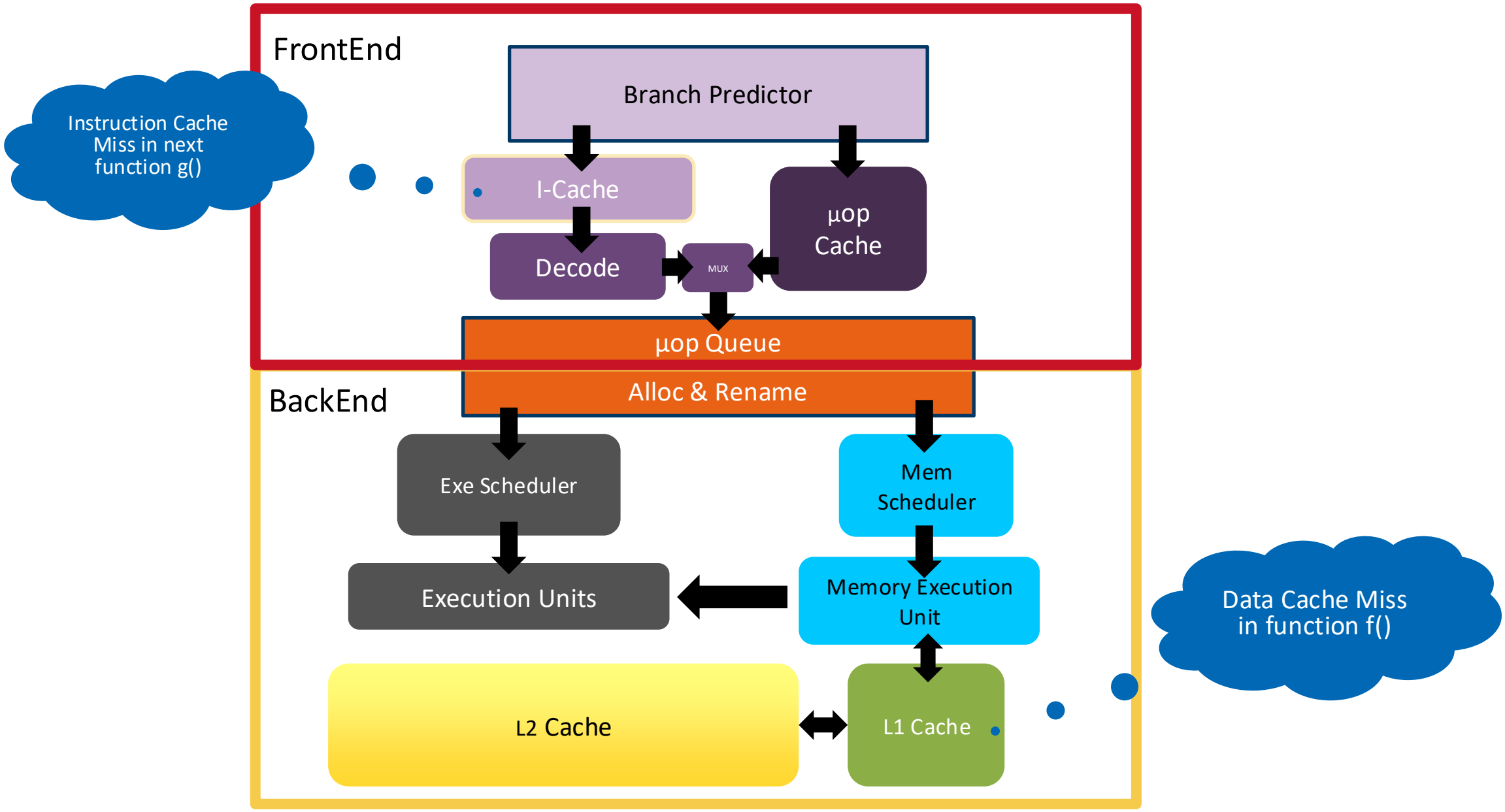
- With in-order non scalar pipelines stall accounting was simple, for each event that cause a stall one could attribute a penalty cost for the stall (e.g. L1 Data Cache Miss costs X cycles of stall)

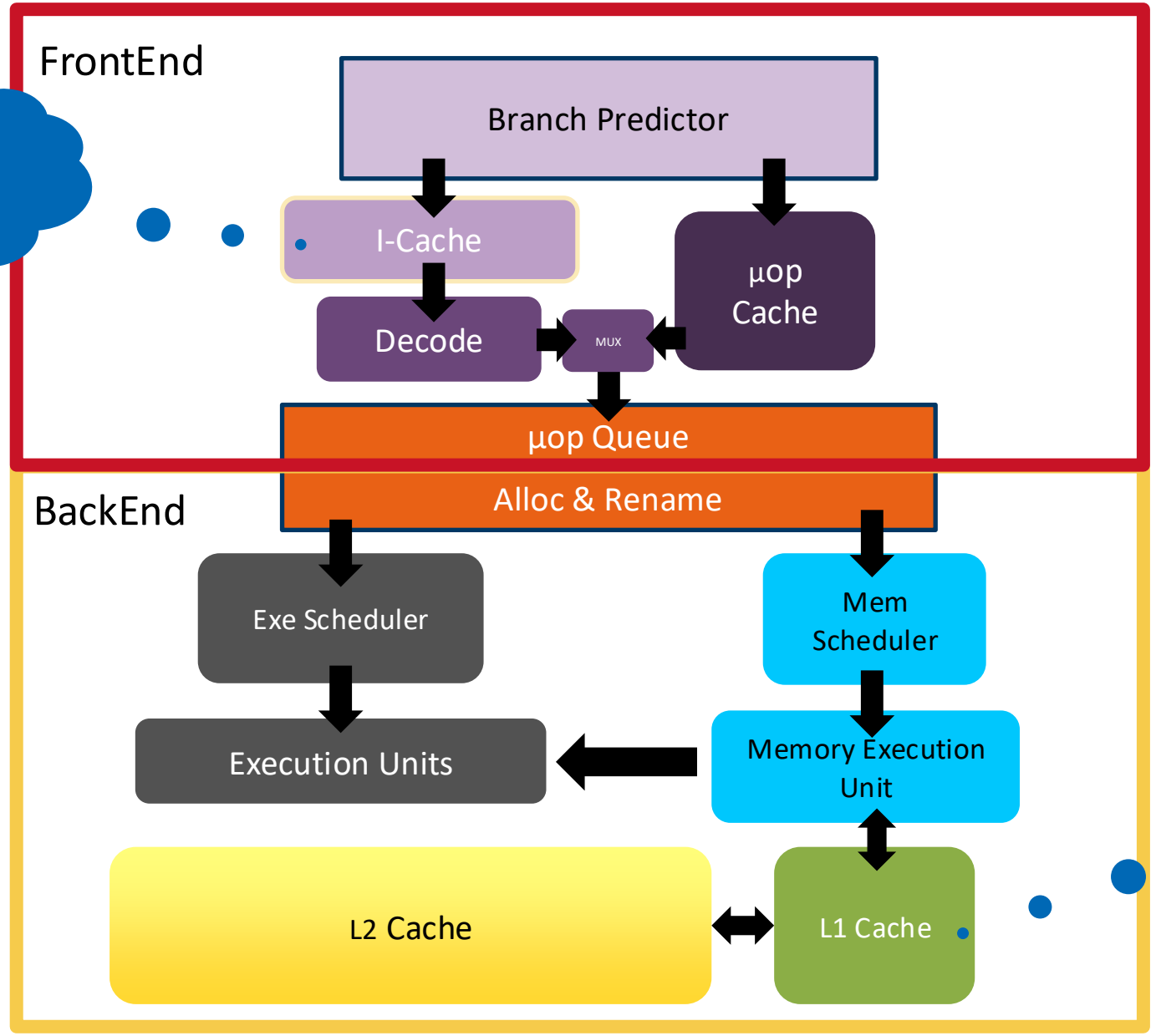
$$STALL = \Sigma\ event_i * penalty_i$$

  - Example: L2 Cache Miss Penalty = 80 cycles * Load L2 Misses.
- This approach does not work for moder OOO cores
  - Overlapping stalls
  - Variable penalties
  - Speculative execution
  - Superscalar inaccuracies


FrontEnd
Branch Predictor
I-Cache
µop Cache
Decode
MUX
µop Queue
BackEnd
Alloc & Rename
Exe Scheduler
Mem Scheduler
Execution Units
Memory Execution Unit
L2 Cache
L1 Cache


FrontEnd
Branch Predictor
I-Cache
µop Cache
Decode
MUX
µop Queue
BackEnd
Alloc & Rename
Exe Scheduler
Mem Scheduler
Execution Units
Memory Execution Unit
L2 Cache
L1 Cache
Data Cache Miss in function f()


FrontEnd
Branch Predictor
Instruction Cache Miss in next function g()
I-Cache
µop Cache
Decode
MUX
µop Queue
BackEnd
Alloc & Rename
Exe Scheduler
Mem Scheduler
Execution Units
Memory Execution Unit
L2 Cache
L1 Cache
Data Cache Miss in function f()



Instruction Cache Miss in next function g()

Which is hurting performance more ?

Data Cache Miss in function f()

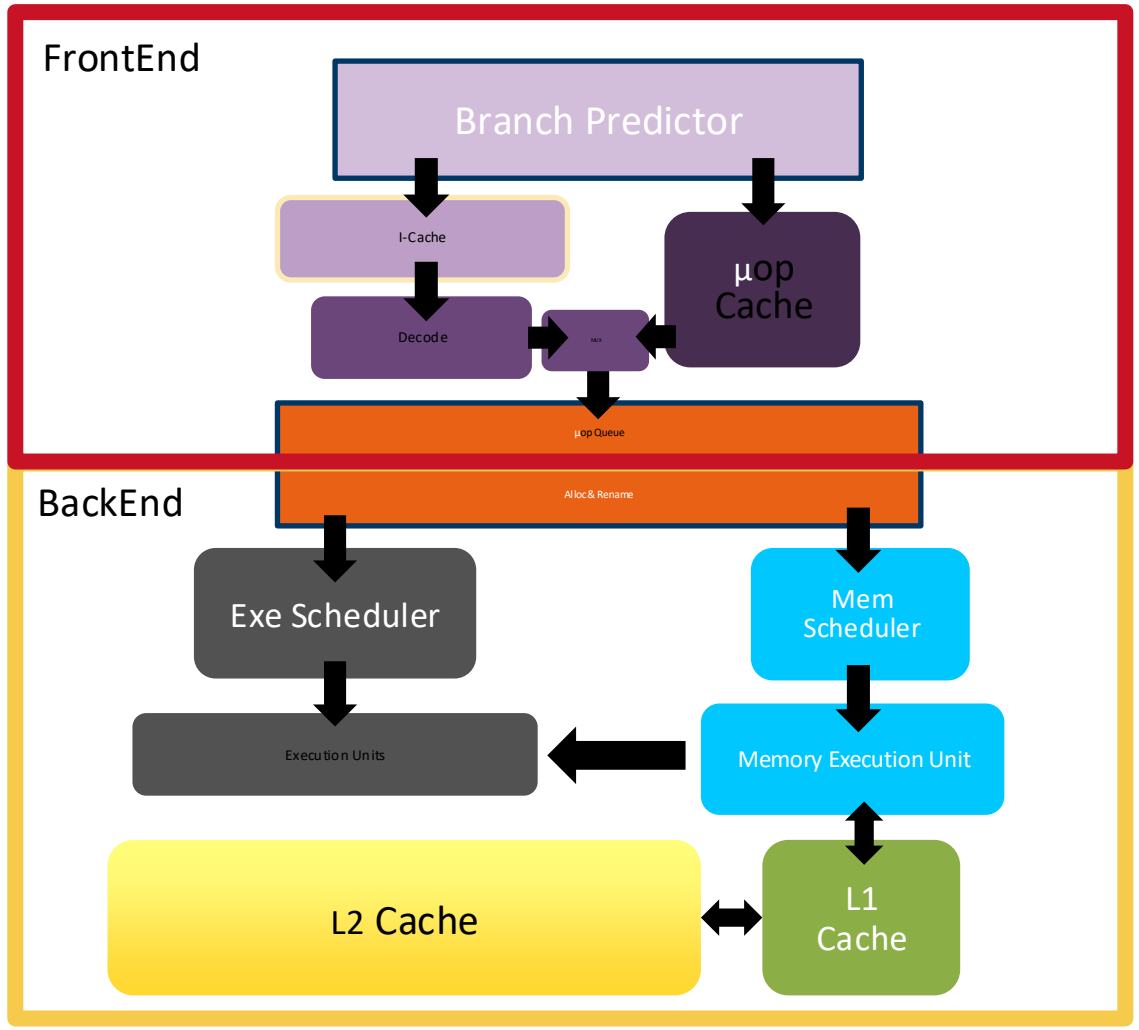
TopDown Microarchitectural Analysis - TMA

From 100’s of PerfMon events to abstract metrics

Based on “A Top-Down Method for Performance Analysis and Counters Architecture” paper by **Ahmad Yasin**

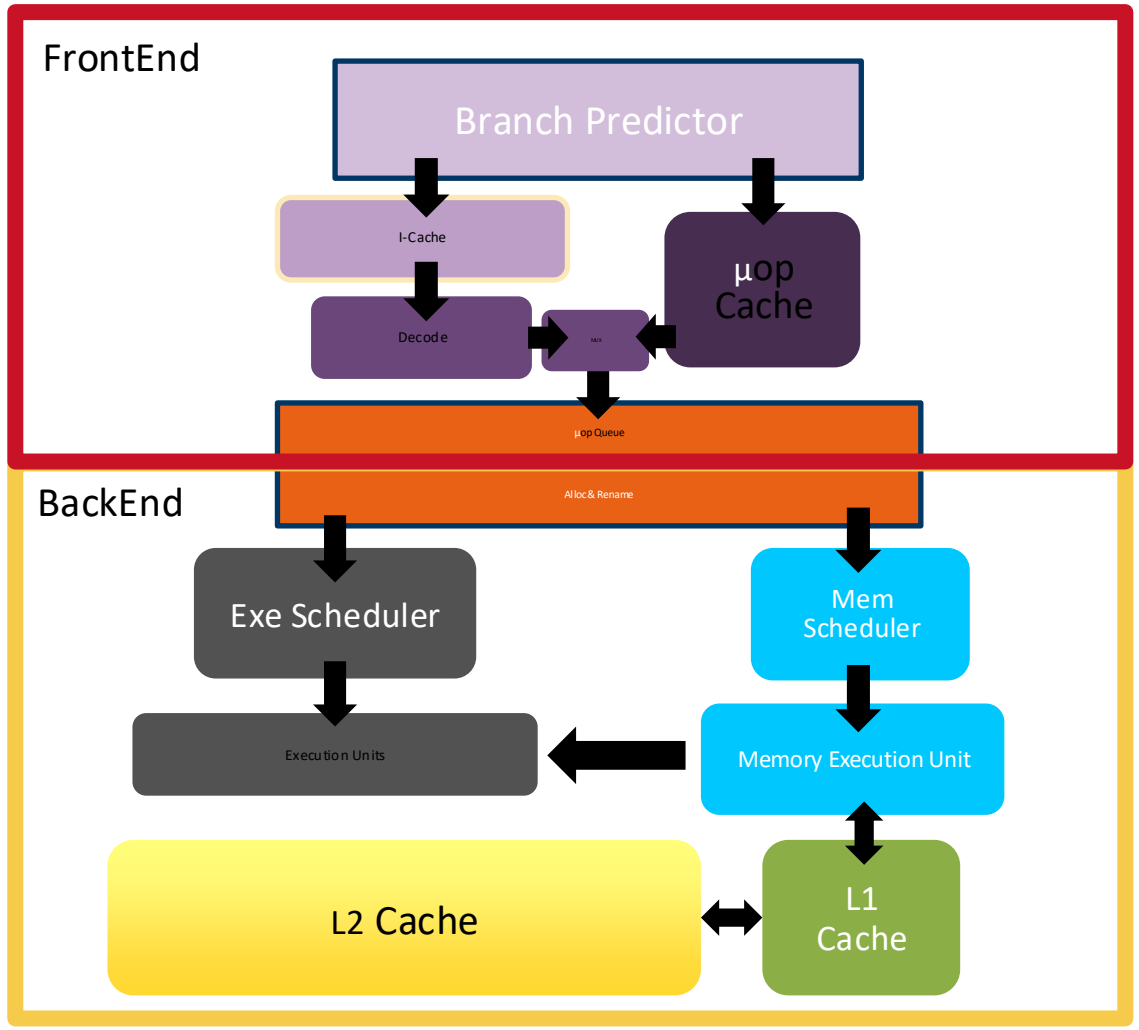
TMA Key Concepts

- In TMA, the core is abstracted into



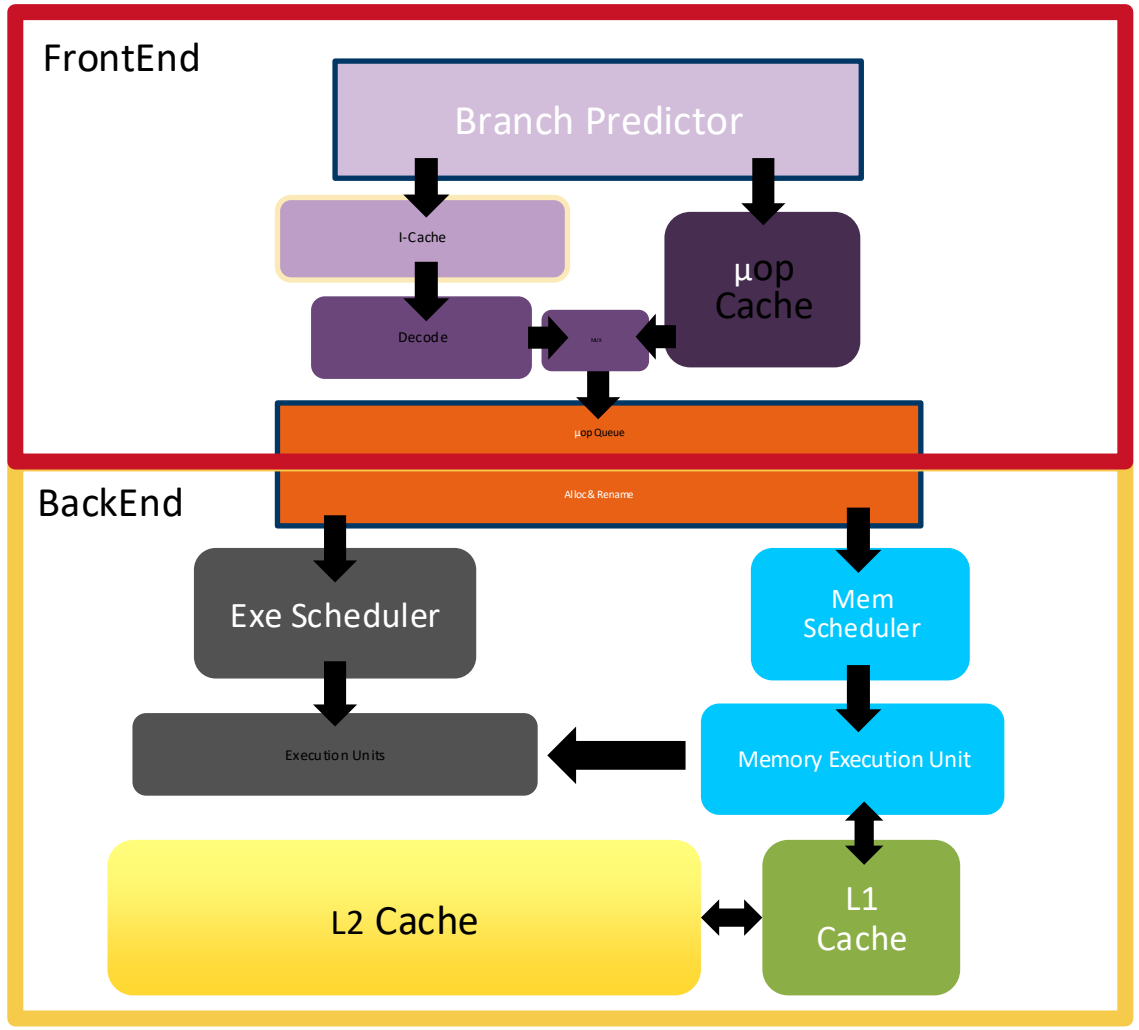
TMA Key Concepts

- In TMA, the core is abstracted into
  - **Frontend**: includes all the logic for fetch and decode



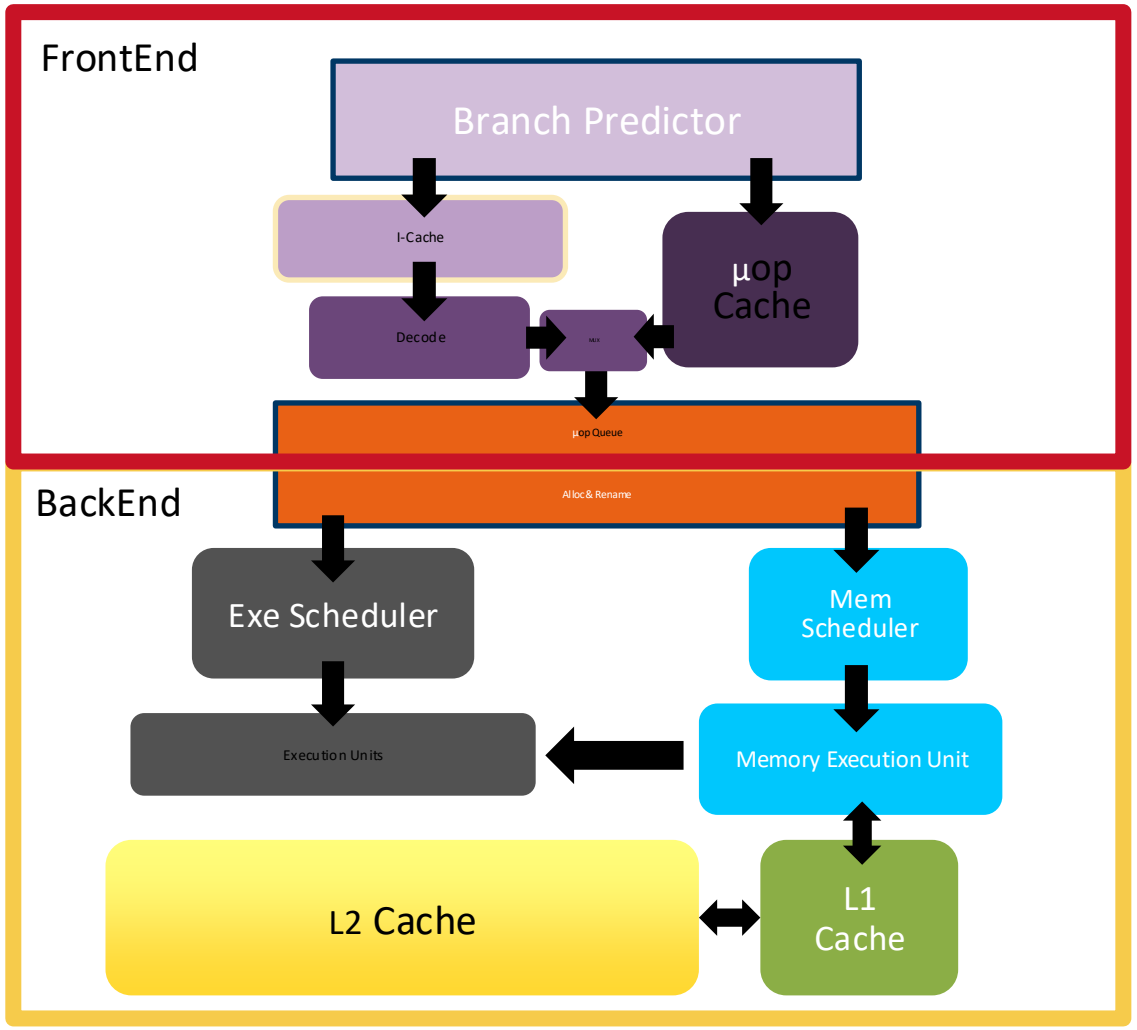
TMA Key Concepts

- In TMA, the core is abstracted into
  - **Frontend**: includes all the logic for fetch and decode
  - **Allocation/issue**: logic sitting in between the frontend and backend



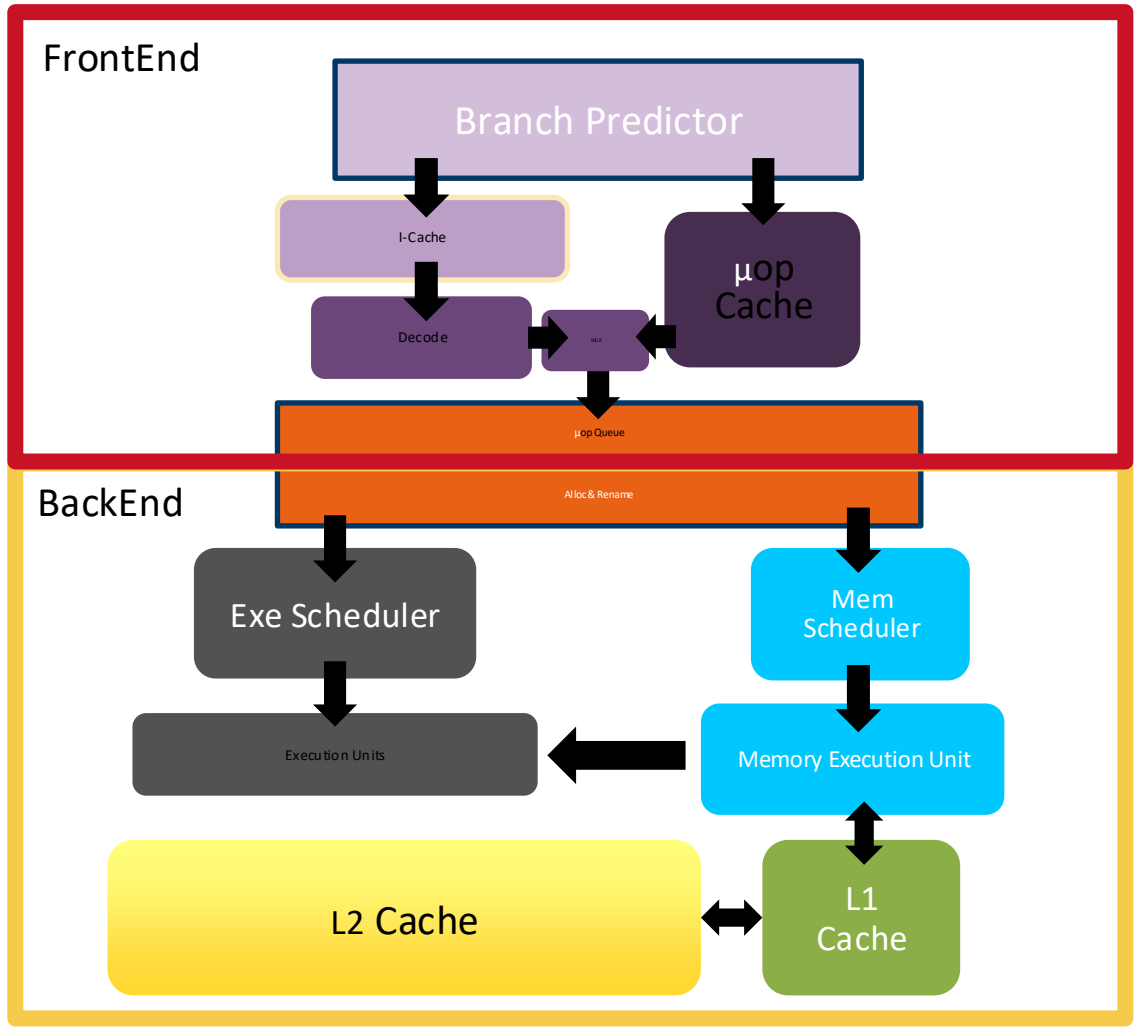
TMA Key Concepts

- In TMA, the core is abstracted into
  - **Frontend**: includes all the logic for fetch and decode
  - **Allocation/issue**: logic sitting in between the frontend and backend
  - **Backend**: includes all the execution, memory and retirement logic



TMA Key Concepts

- In TMA, the core is abstracted into
  - **Frontend**: includes all the logic for fetch and decode
  - **Allocation/issue**: logic sitting in between the frontend and backend
  - **Backend**: includes all the execution, memory and retirement logic
- Allocation **Width**: how many uops/instructions can be operated at once (Allocation stages)



TMA Key Concepts

- Top Down (TMA) is a hierarchical performance analysis technique for identifying software bottlenecks that occur in the core
- The novel concept that Top Down introduces is called Pipeline Slot

TMA Key Concepts

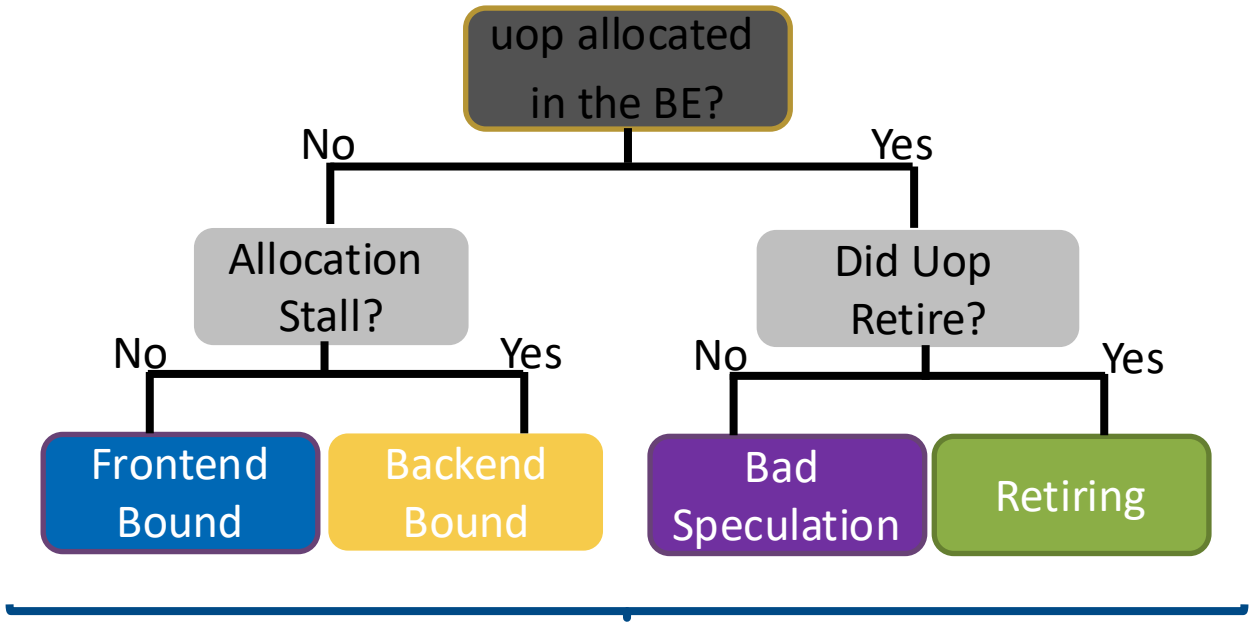
- Top Down (TMA) is a hierarchical performance analysis technique for identifying software bottlenecks that occur in the core
- The novel concept that Top Down introduces is called Pipeline Slot
  - A pipeline slot is an abstraction representing the HW resources available to one uop as it travels through the pipeline, per cycle

TMA Key Concepts

- Top Down (TMA) is a hierarchical performance analysis technique for identifying software bottlenecks that occur in the core
- The novel concept that Top Down introduces is called Pipeline Slot
  - A pipeline slot is an abstraction representing the HW resources available to one uop as it travels through the pipeline, per cycle
- Allocation width determines the number of Pipeline Slots per cycle
  - Allocation width = 4 on SkyLake, 5 on IceLake, 6 on GoldenCove(ADL P-Core) / SapphireRapids and 8 on LionCove (LNL/ARL P-Core) and CougarCove (PTL P-Core)
- TMA drills the hierarchy down iteratively using performance counters until narrowing the bottleneck
- Low-cost & Comprehensive

TMA issue classification

- Performance issues are classified to what happened to each pipeline slot
- TMA categorizes CPU pipeline slots into high-level categories -

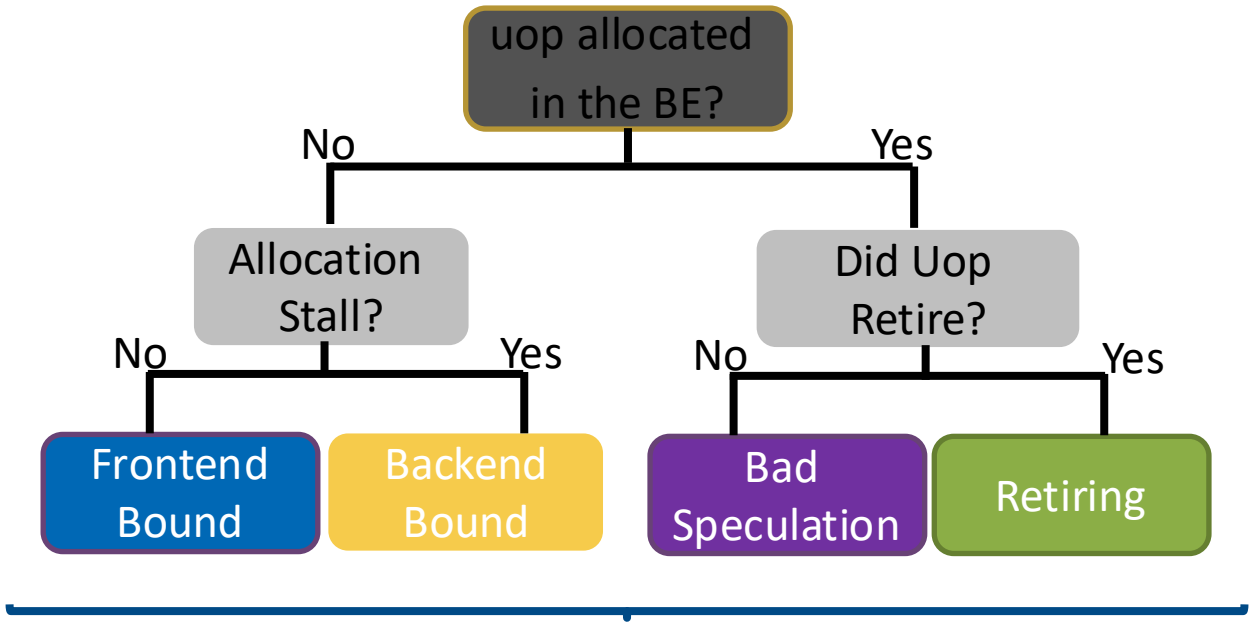


The top 4 categories of Top Down (TMA Level 1)

- **Retiring**: Useful work slots (maximizes IPC, ideal ~100% on peak throughput)

TMA issue classification

- Performance issues are classified to what happened to each pipeline slot
- TMA categorizes CPU pipeline slots into high-level categories -

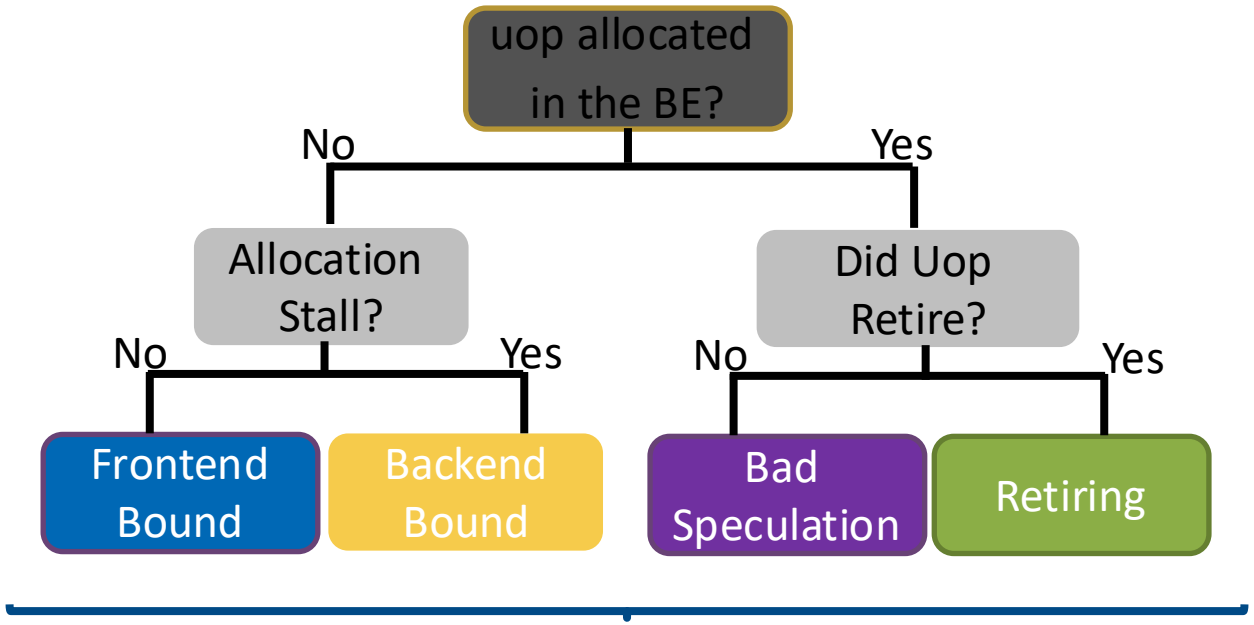


The top 4 categories of Top Down (TMA Level 1)

- **Retiring**: Useful work slots (maximizes IPC, ideal ~100% on peak throughput)
- **Bad Speculation**: Wasted slots from branch mispredictions or memory ordering clears

TMA issue classification

- Performance issues are classified to what happened to each pipeline slot
- TMA categorizes CPU pipeline slots into high-level categories -

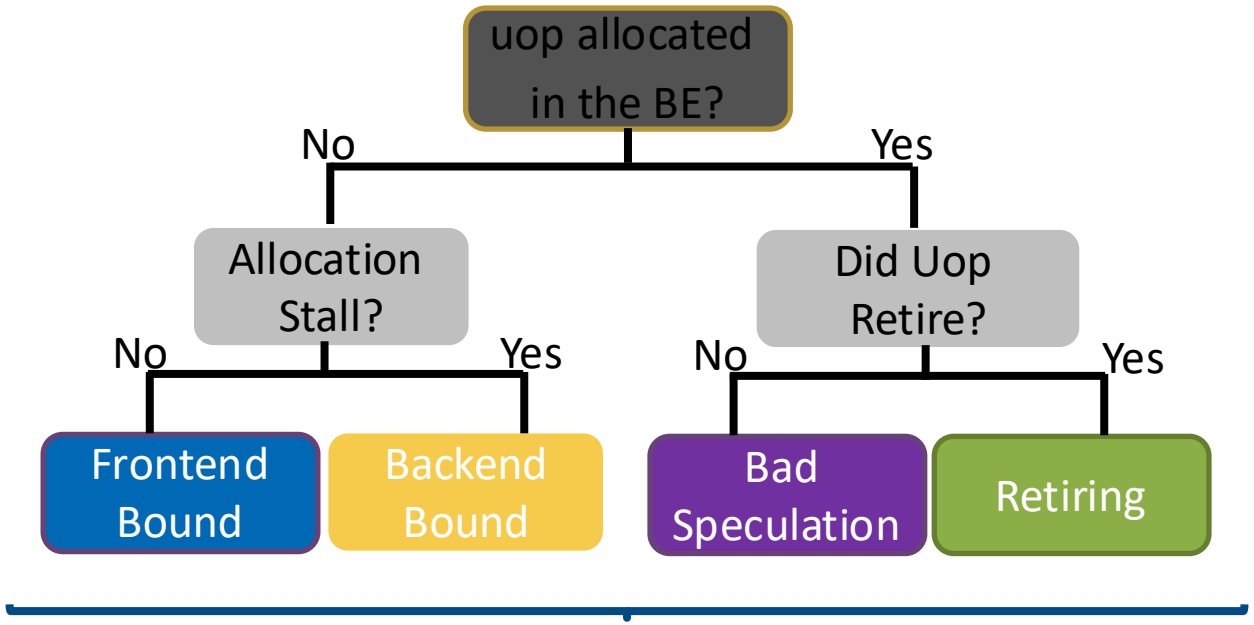


The top 4 categories of Top Down (TMA Level 1)

- **Retiring**: Useful work slots (maximizes IPC, ideal ~100% on peak throughput)
- **Bad Speculation**: Wasted slots from branch mispredictions or memory ordering clears
- **Frontend Bound**: Delays or non-optimal utilization in fetching/decoding instructions (e.g., I-cache misses)

TMA issue classification

- Performance issues are classified to what happened to each pipeline slot
- TMA categorizes CPU pipeline slots into high-level categories -

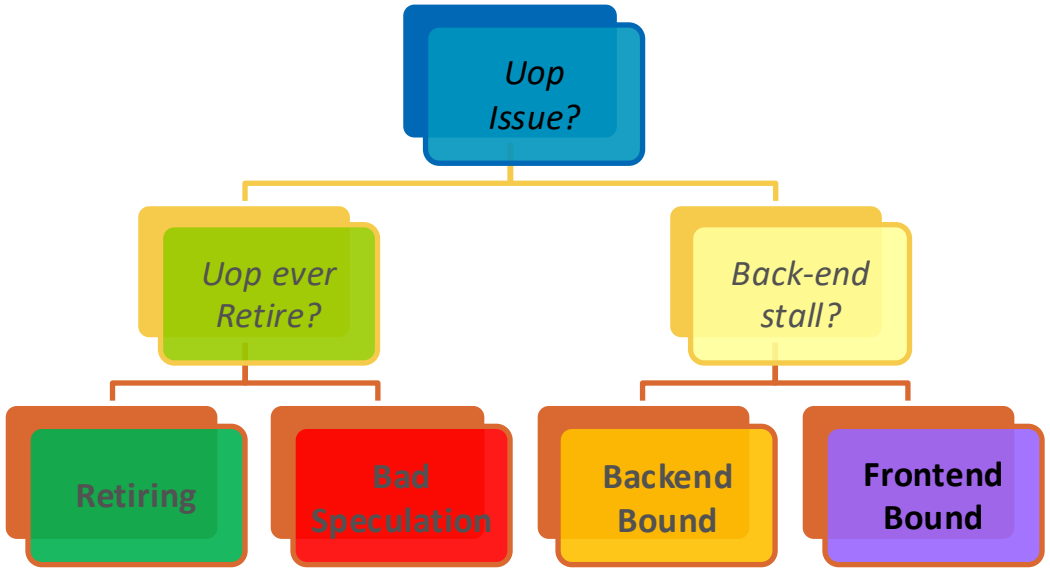


The top 4 categories of Top Down (TMA Level 1)

- **Retiring**: Useful work slots (maximizes IPC, ideal ~100% on peak throughput)
- **Bad Speculation**: Wasted slots from branch mispredictions or memory ordering clears
- **Frontend Bound**: Delays or non-optimal utilization in fetching/decoding instructions (e.g., I-cache misses)
- **Backend Bound**: Allocation (issue) and Execution (dispatch) stalls (e.g., memory, core bound subcategories)

Top Level Illustration on a 4-wide core

Top Level Illustration on a 4-wide core

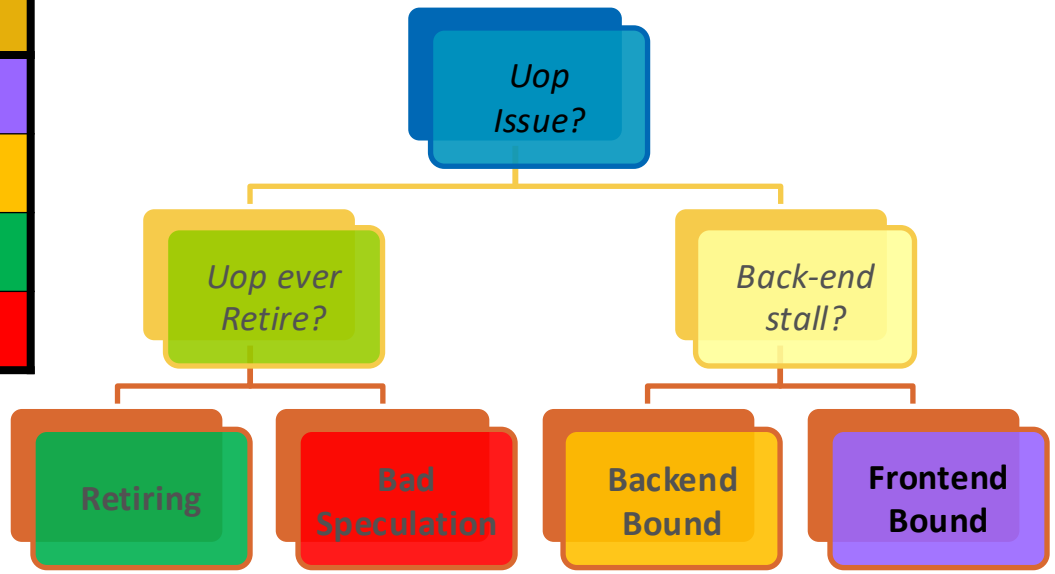


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	…	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	**v**		v
Issue Slot 2	-	-	-	**v**		**v**
Issue Slot 3	-	-	-	**v**		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

Bubble := An un-utilized Slot.

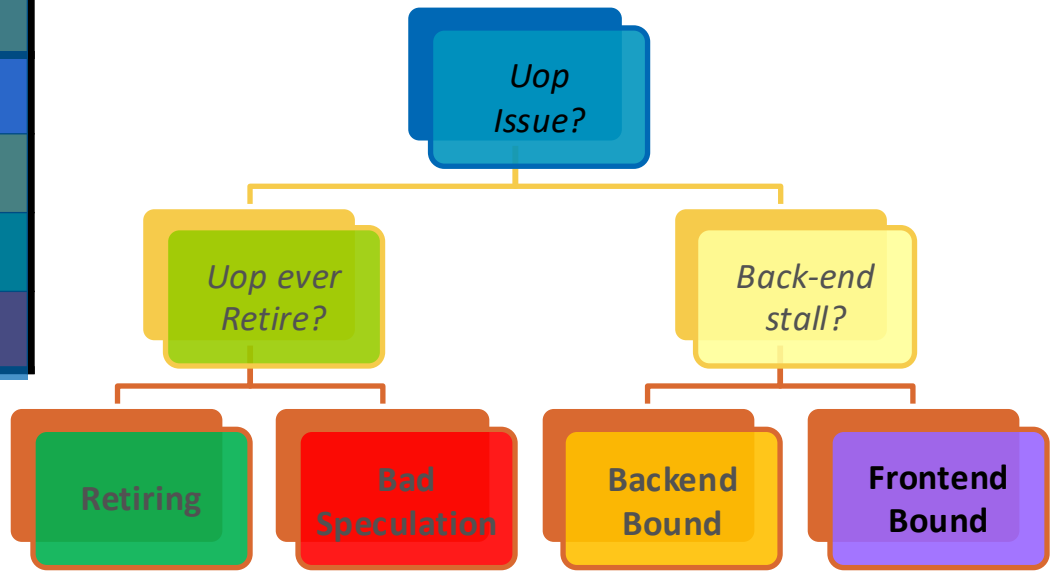


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	…	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	v		v
Issue Slot 2	-	-	-	v		v
Issue Slot 3	-	-	-	v		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

Bubble := An un-utilized Slot.

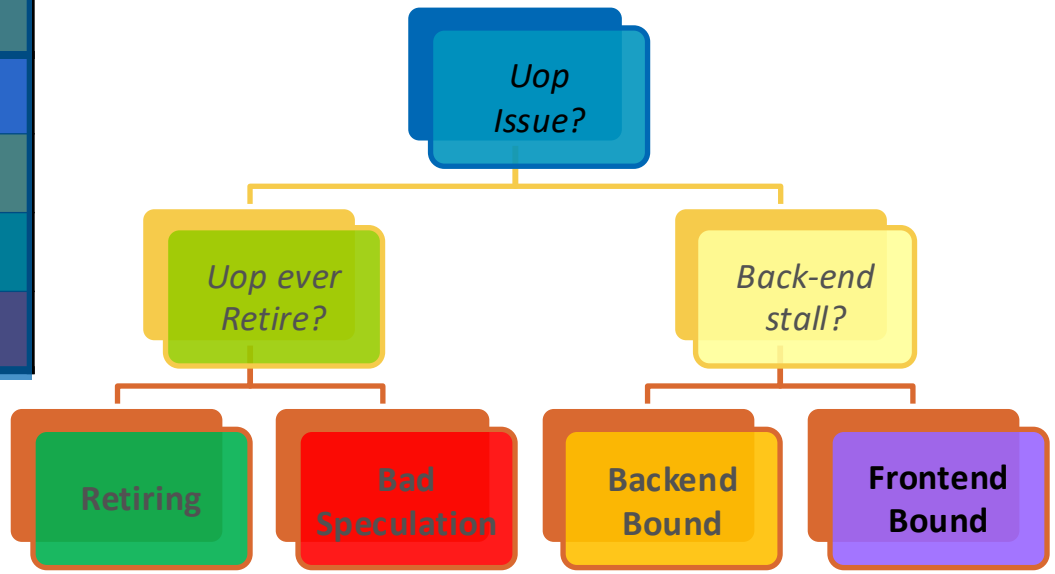


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	…	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	v		v
Issue Slot 2	-	-	-	v		v
Issue Slot 3	-	-	-	v		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

Bubble := An un-utilized Slot.

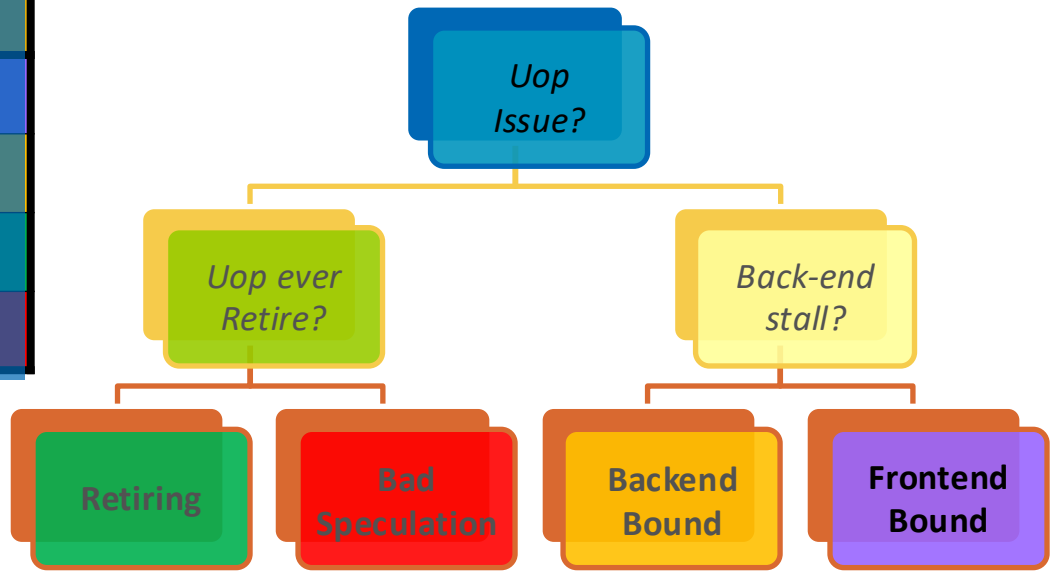


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	…	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	v		v
Issue Slot 2	-	-	-	v		v
Issue Slot 3	-	-	-	v		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

Bubble := An un-utilized Slot.

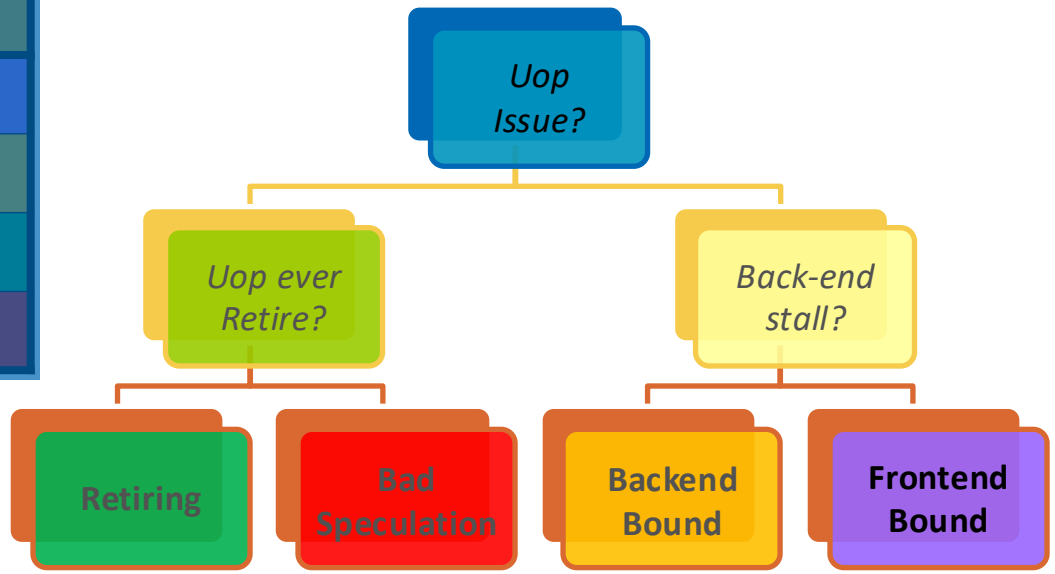


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	...	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	**v**		v
Issue Slot 2	-	-	-	**v**		**v**
Issue Slot 3	-	-	-	**v**		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

Bubble := An un-utilized Slot.

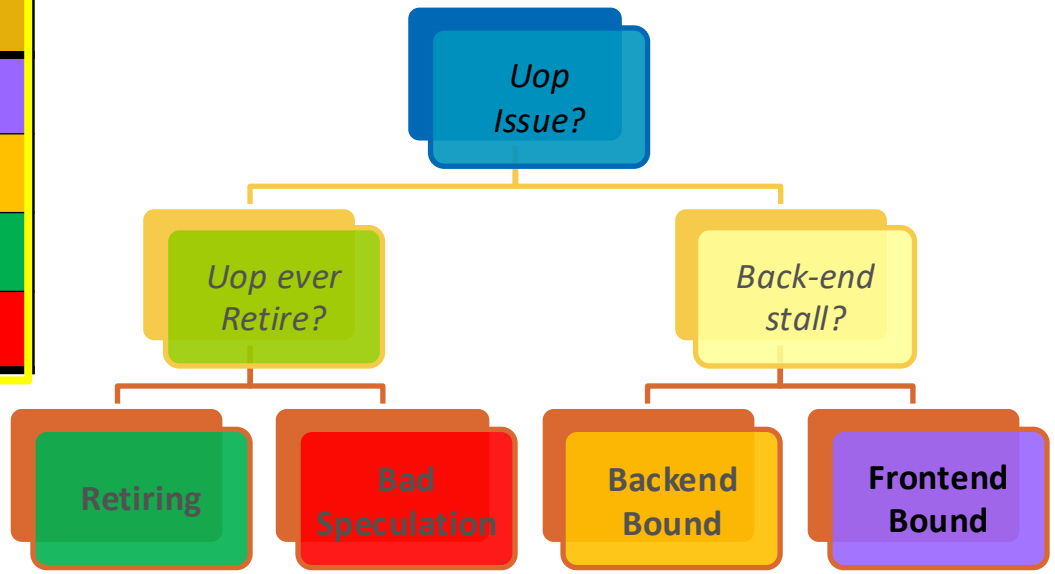


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	…	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	**v**		v
Issue Slot 2	-	-	-	**v**		**v**
Issue Slot 3	-	-	-	**v**		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

Bubble := An un-utilized Slot.

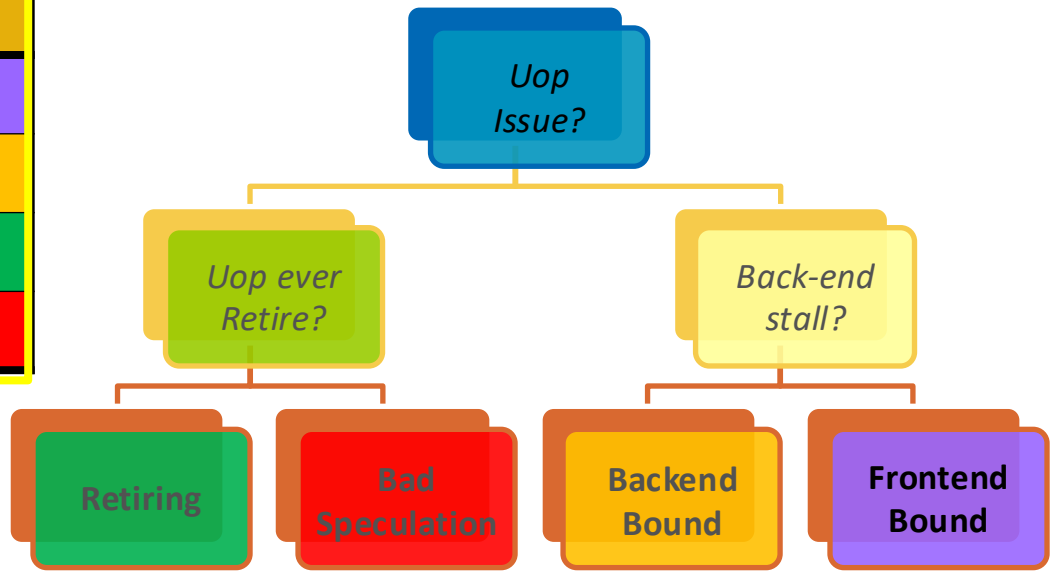


Top Level Illustration on a 4-wide core

Cycle	1	2	3	4	…	x
Back-end Stall	0	0	1	0		0
Issue Slot 0	-	v	-	v		v
Issue Slot 1	-	v	-	**v**		v
Issue Slot 2	-	-	-	**v**		**v**
Issue Slot 3	-	-	-	**v**		-
Frontend Bound	4	2		0		1
Backend Bound			4	0		0
Retiring		2		1		2
Bad Speculation				3		1

Issue Slots := number of uops the front-end pipeline can issue to the back-end in a cycle.

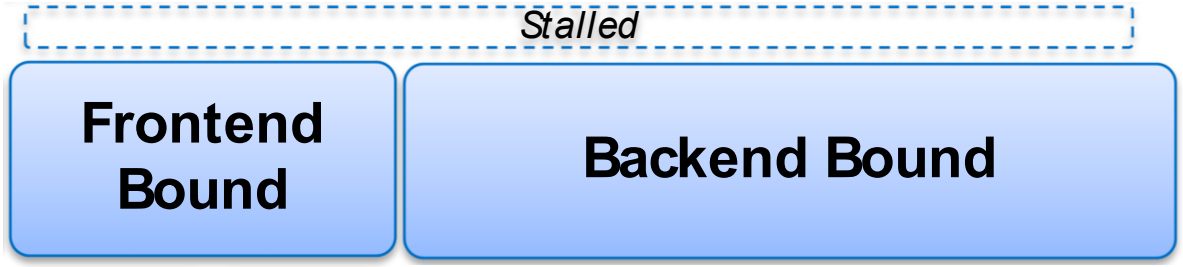
Bubble := An un-utilized Slot.



Classify each Pipeline Slot into 1 of 4 Categories

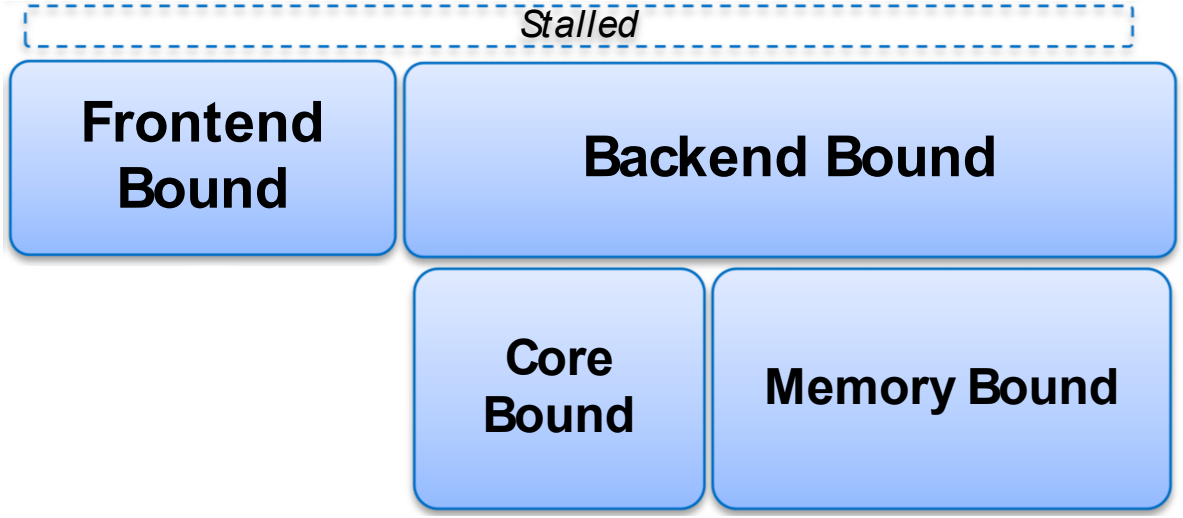
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*



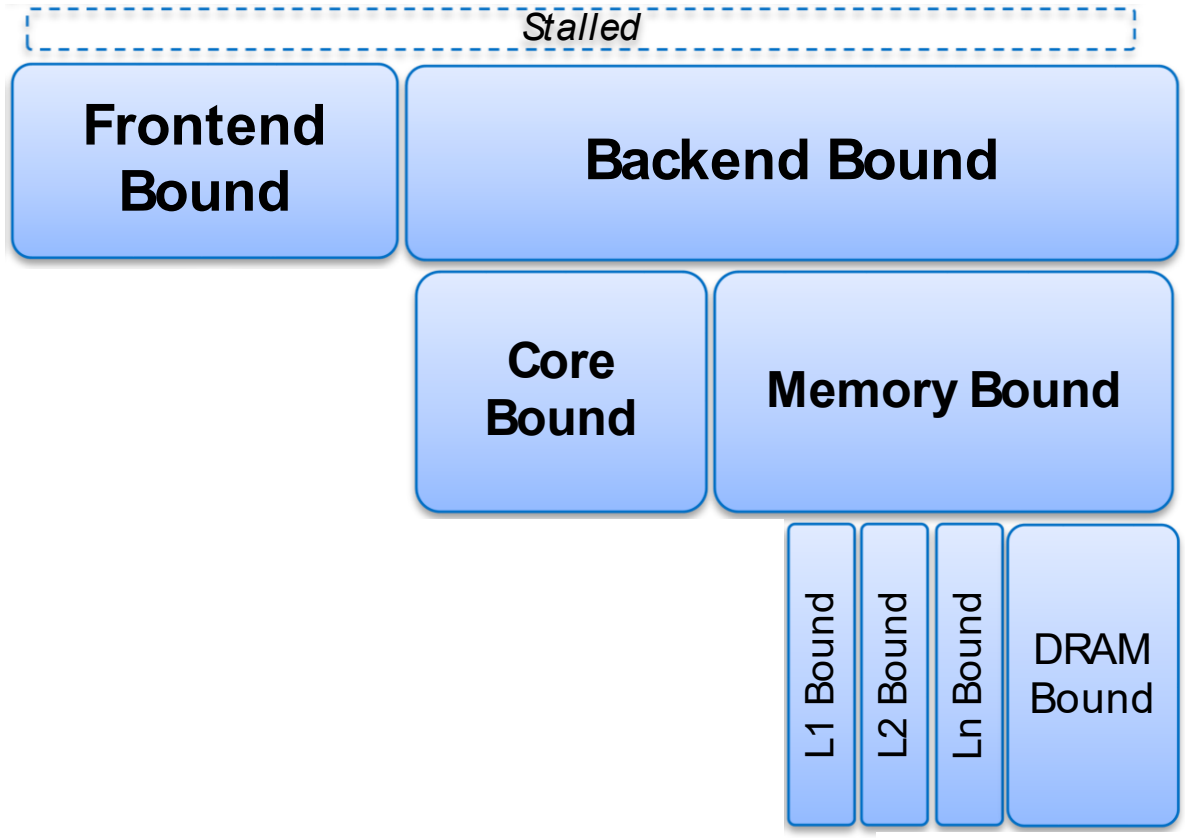
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*



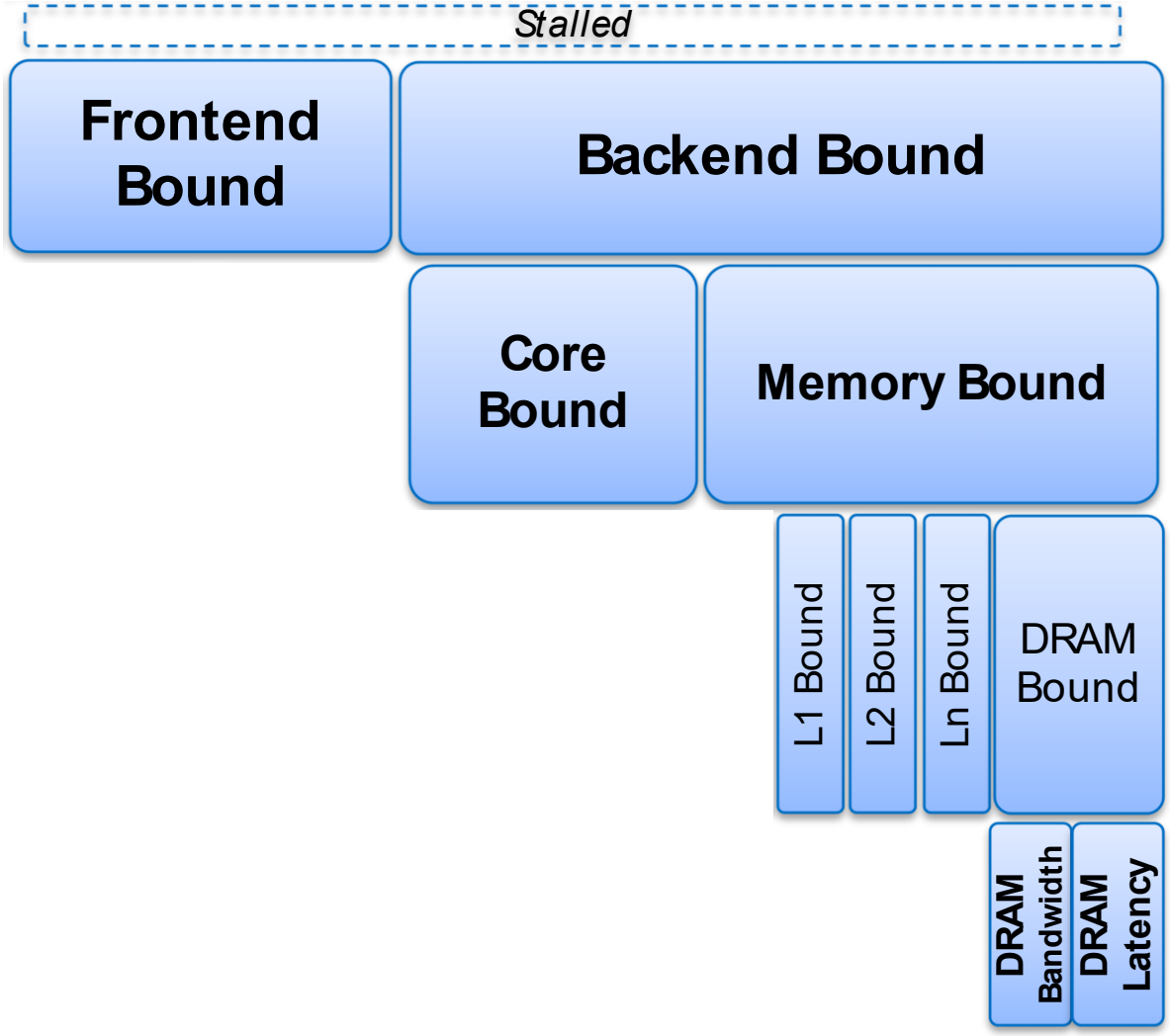
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*



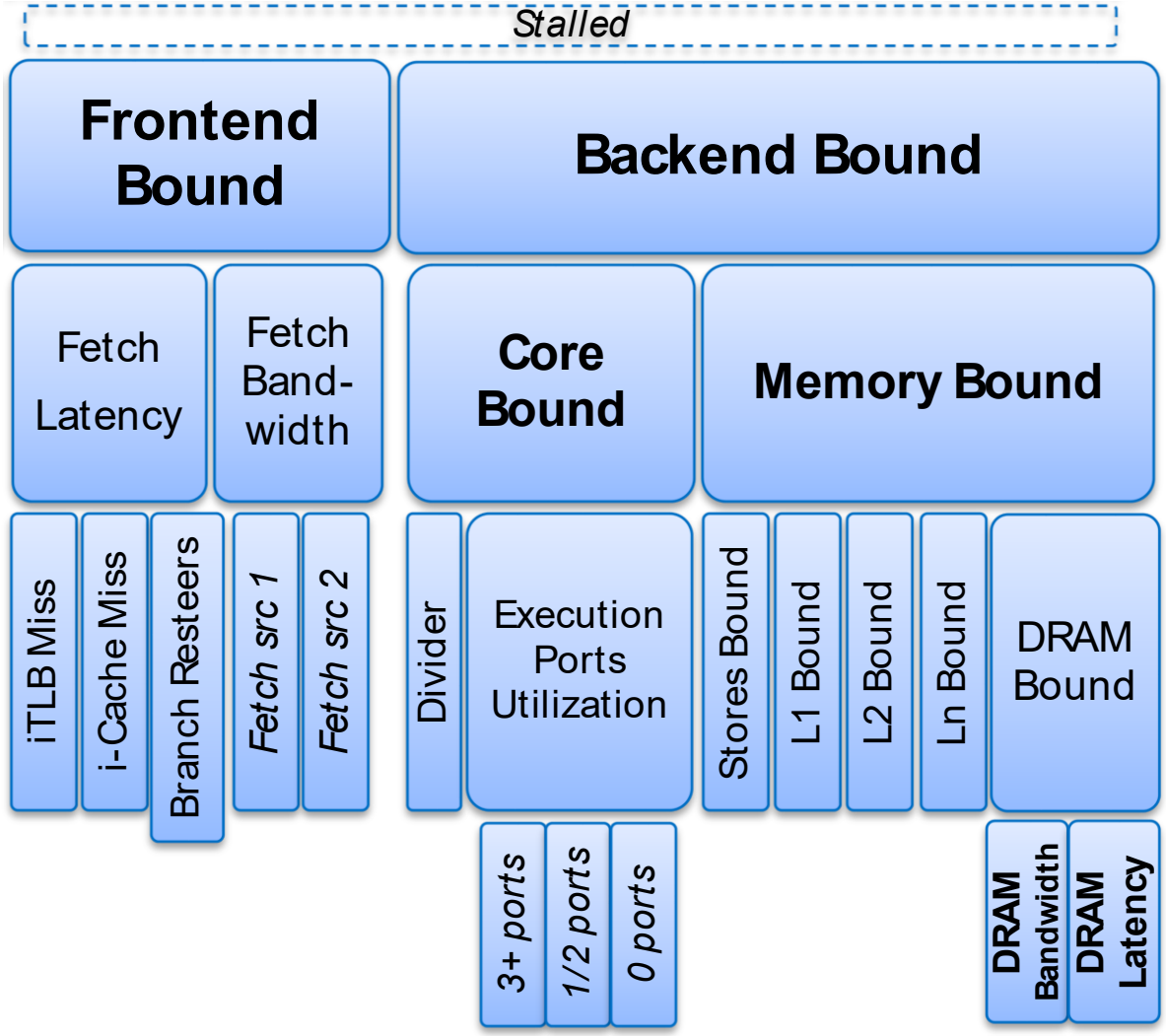
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*



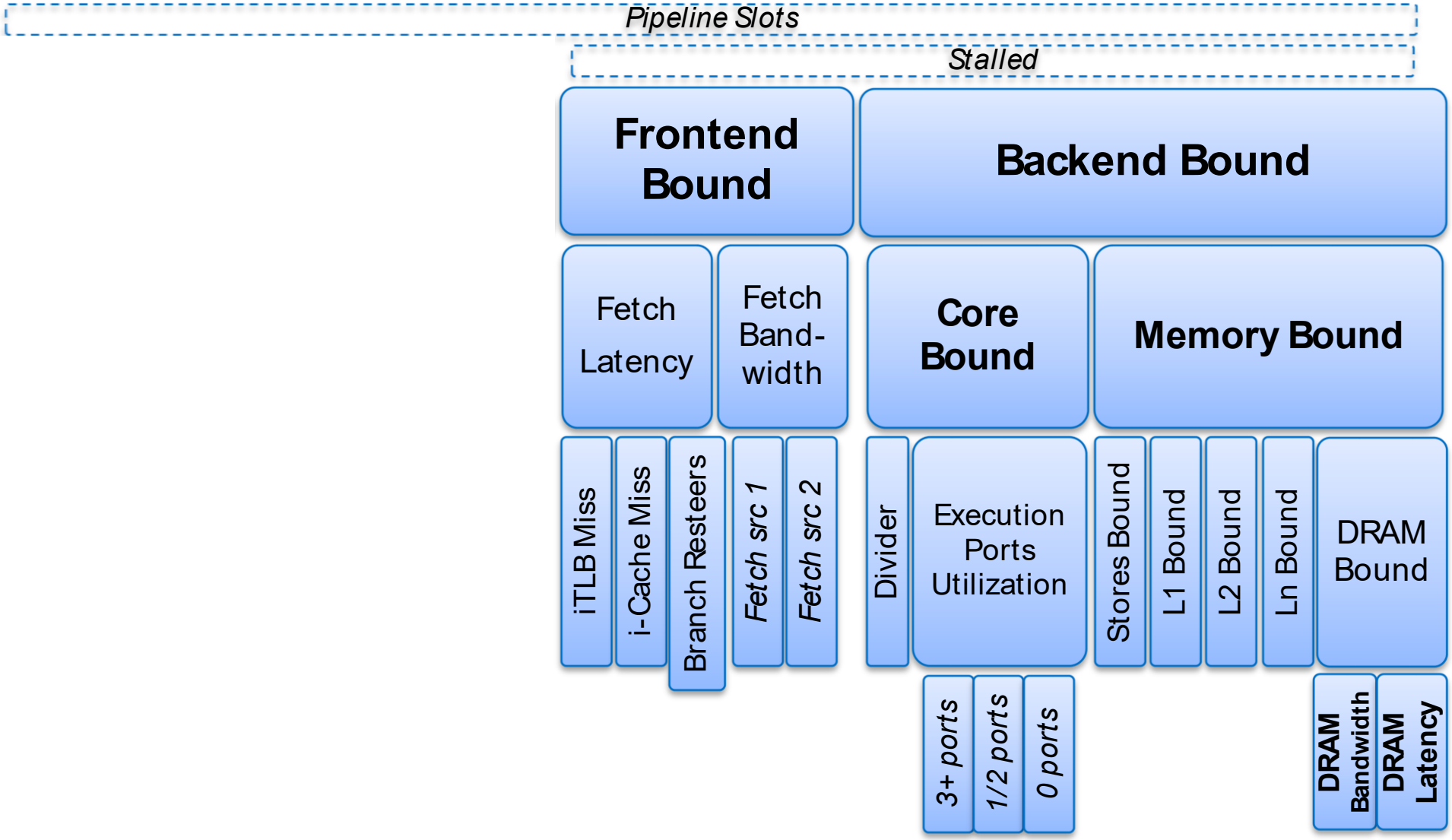
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*



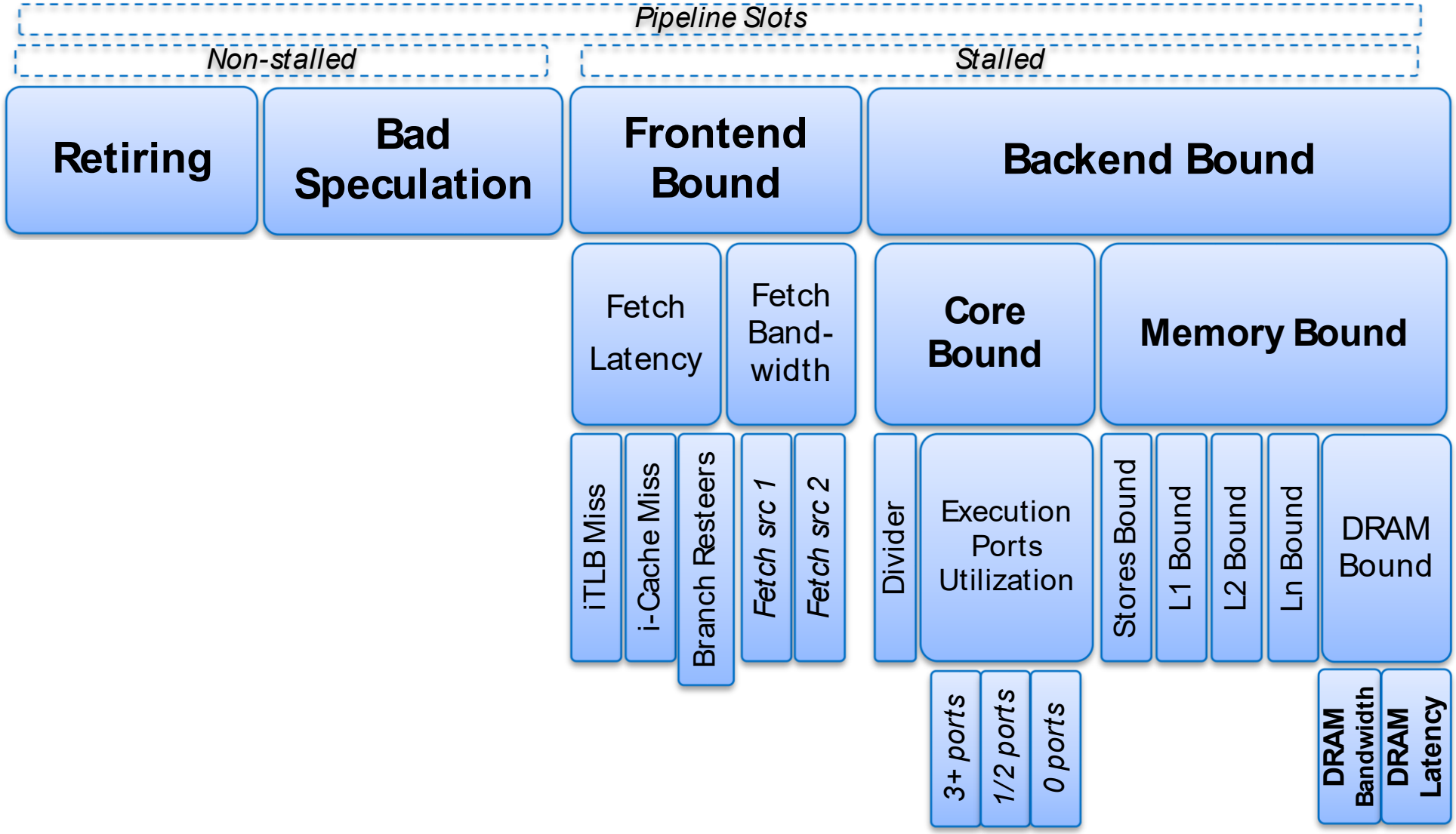
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*



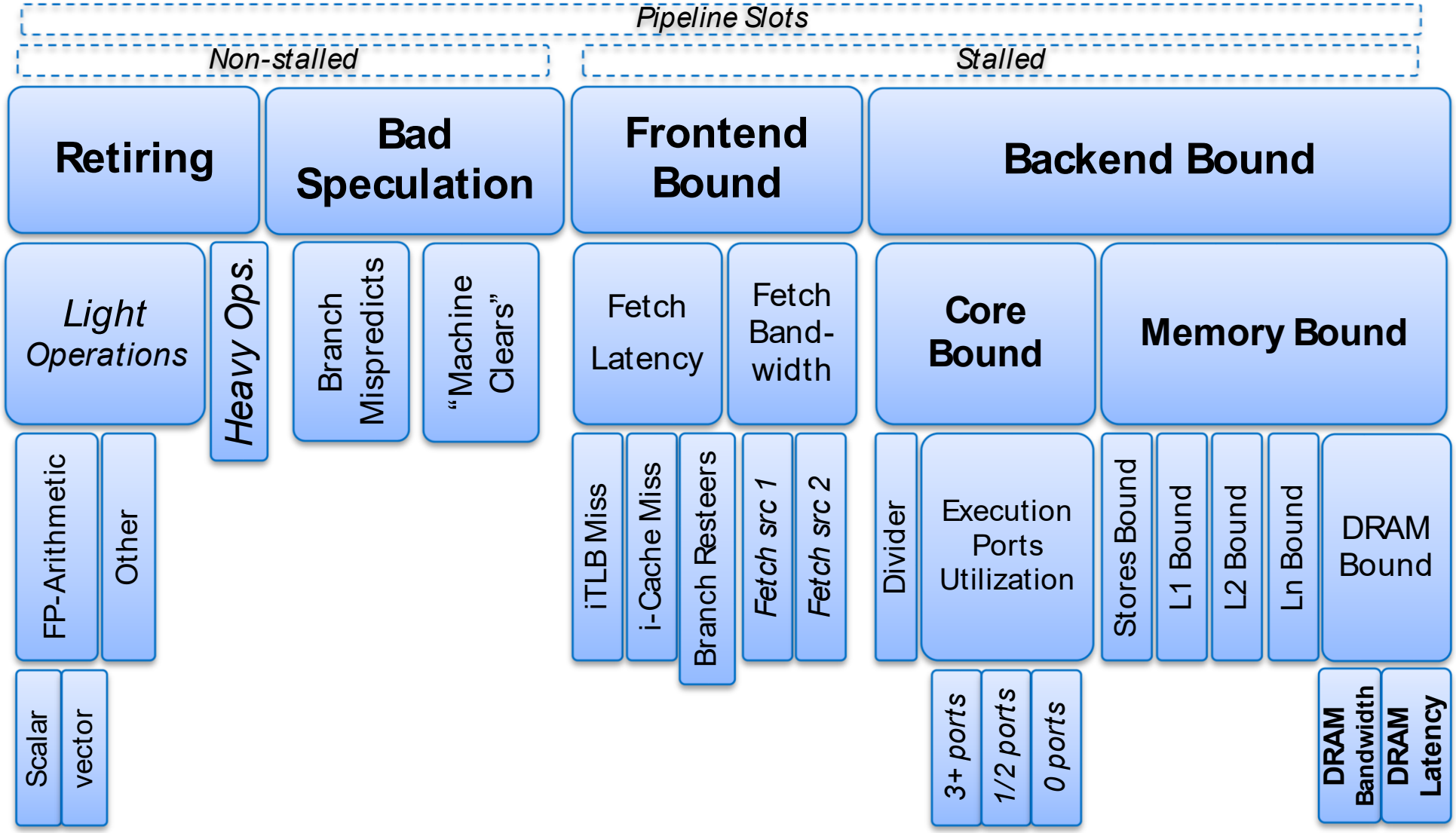
*Reference paper: A. Yasin, “A Top-Down Method for Performance Analysis and Counters Architecture”, ISPASS 2014

One Bottlenecks Hierarchy*

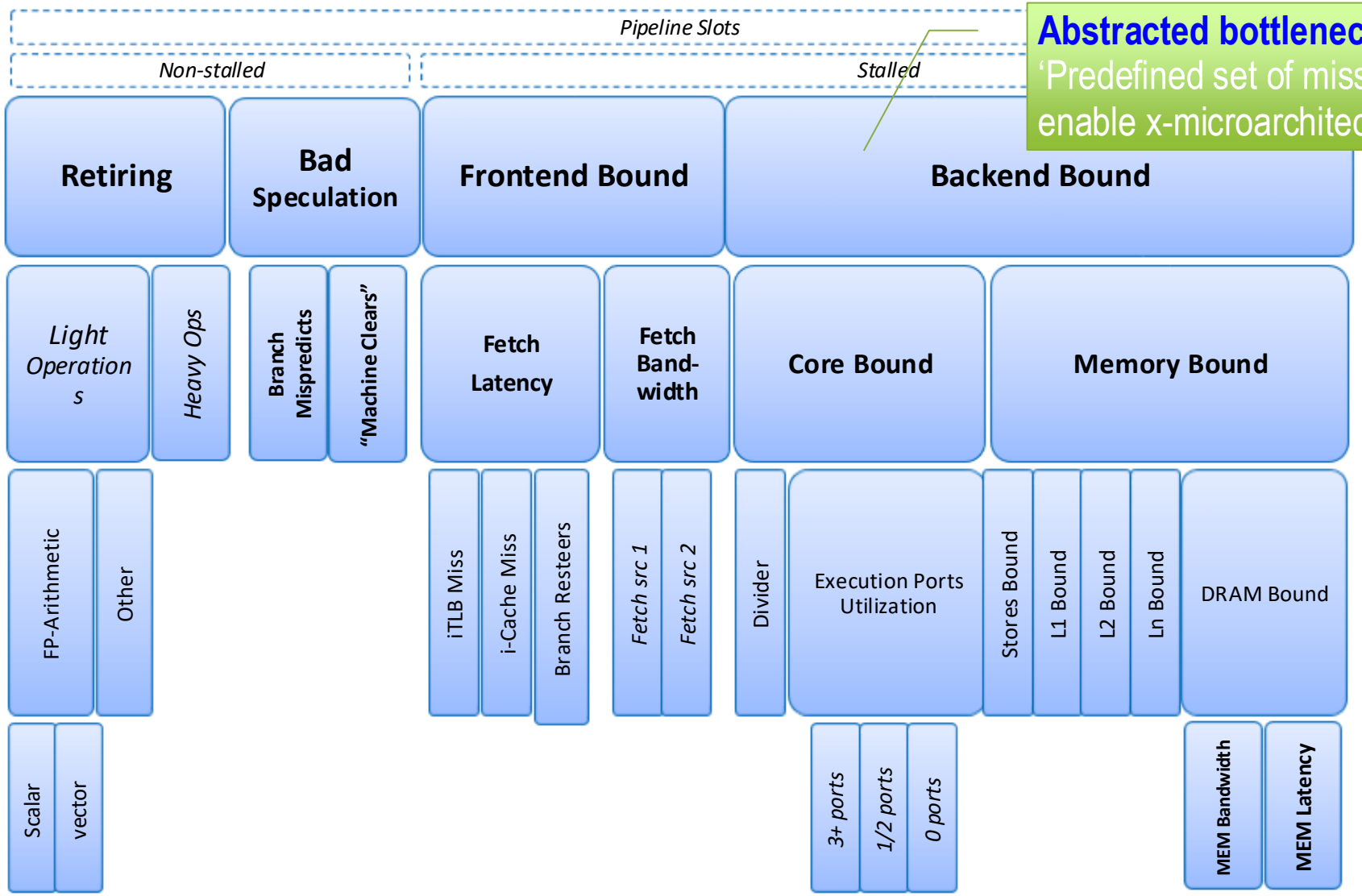


*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

One Bottlenecks Hierarchy*



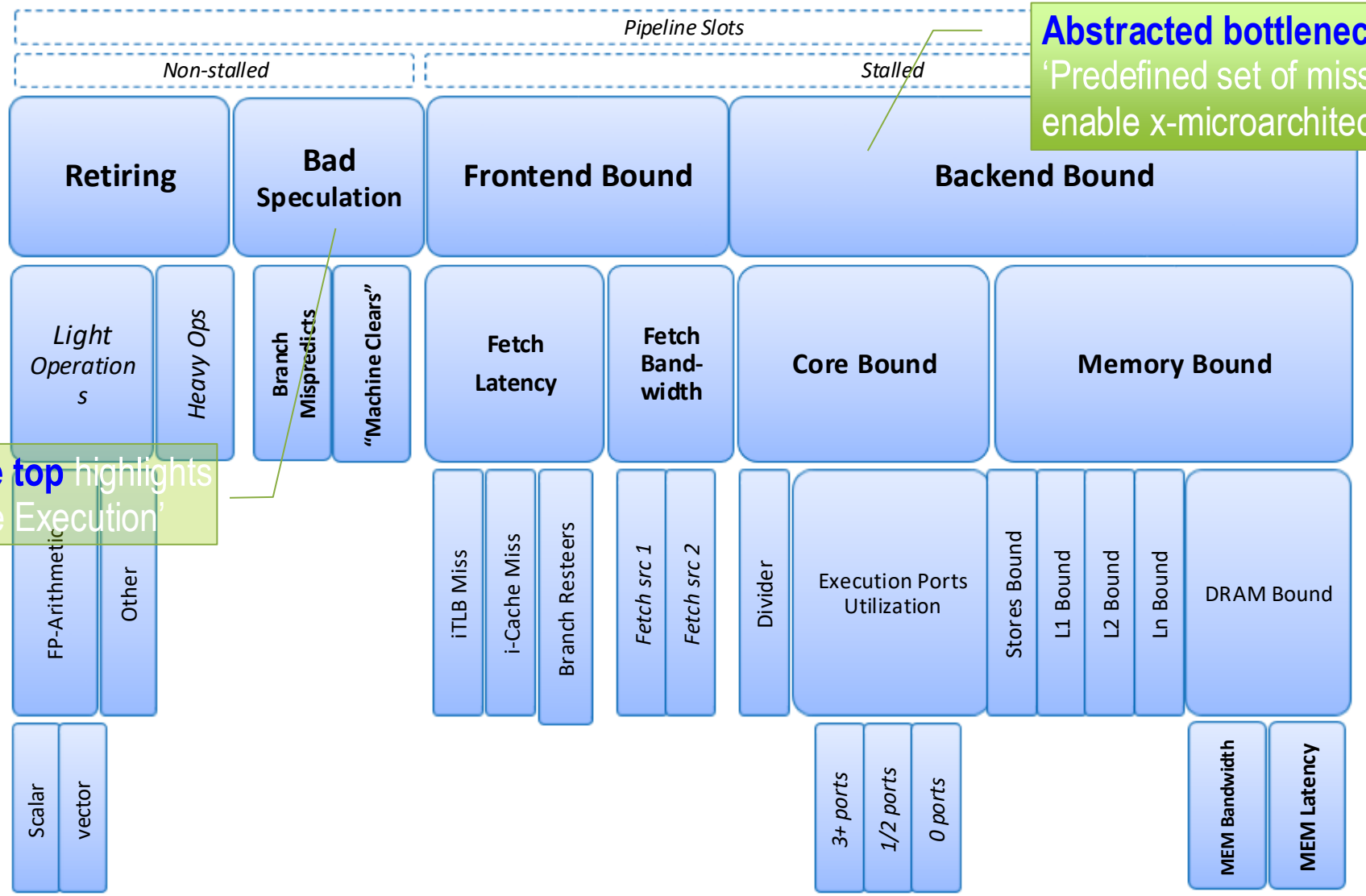
Addressing the 5 key perf analysis challenges



Abstracted bottlenecks displace 'Predefined set of miss-events' & enable x-microarchitecture comparison

*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

Addressing the 5 key perf analysis challenges

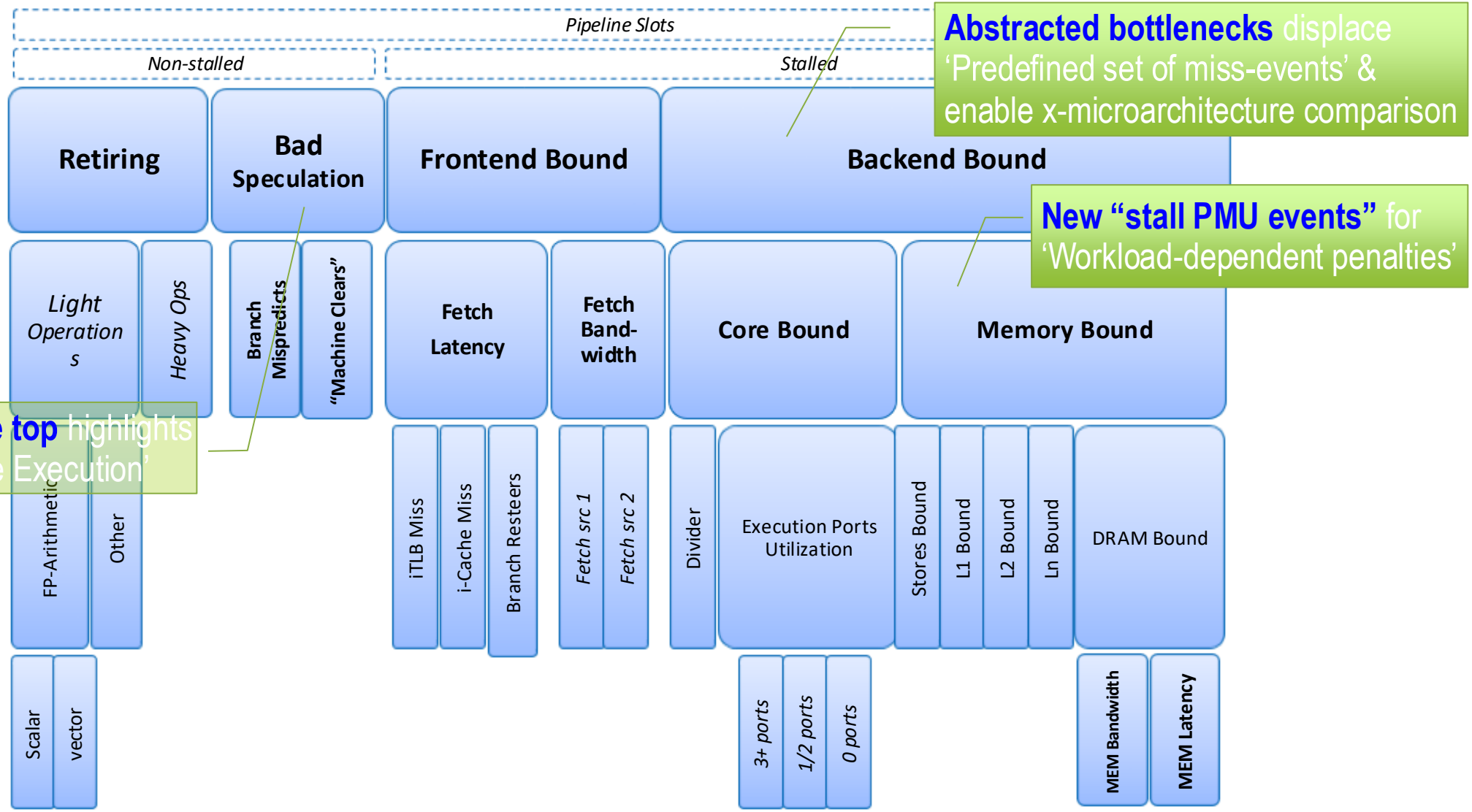


Abstracted bottlenecks displace 'Predefined set of miss-events' & enable x-microarchitecture comparison

Bad Speculation at the top highlights sensitivity to 'Speculative Execution'

*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

Addressing the 5 key perf analysis challenges



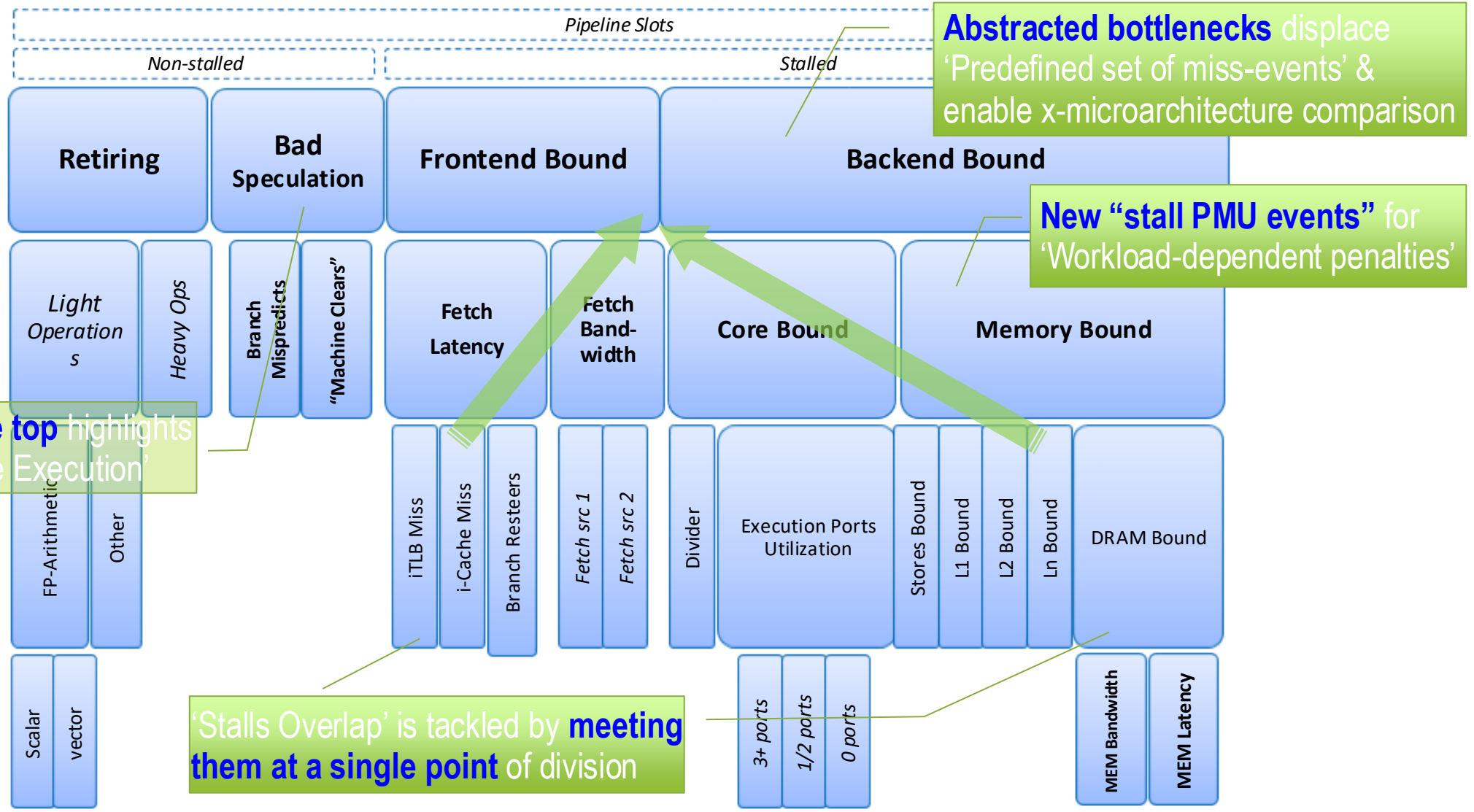
Abstracted bottlenecks displace 'Predefined set of miss-events' & enable x-microarchitecture comparison

New "stall PMU events" for 'Workload-dependent penalties'

Bad Speculation at the top highlights sensitivity to 'Speculative Execution'

*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

Addressing the 5 key perf analysis challenges



Abstracted bottlenecks displace 'Predefined set of miss-events' & enable x-microarchitecture comparison

New "stall PMU events" for 'Workload-dependent penalties'

Bad Speculation at the top highlights sensitivity to 'Speculative Execution'

'Stalls Overlap' is tackled by **meeting them at a single point** of division

*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

Addressing the 5 key perf analysis challenges

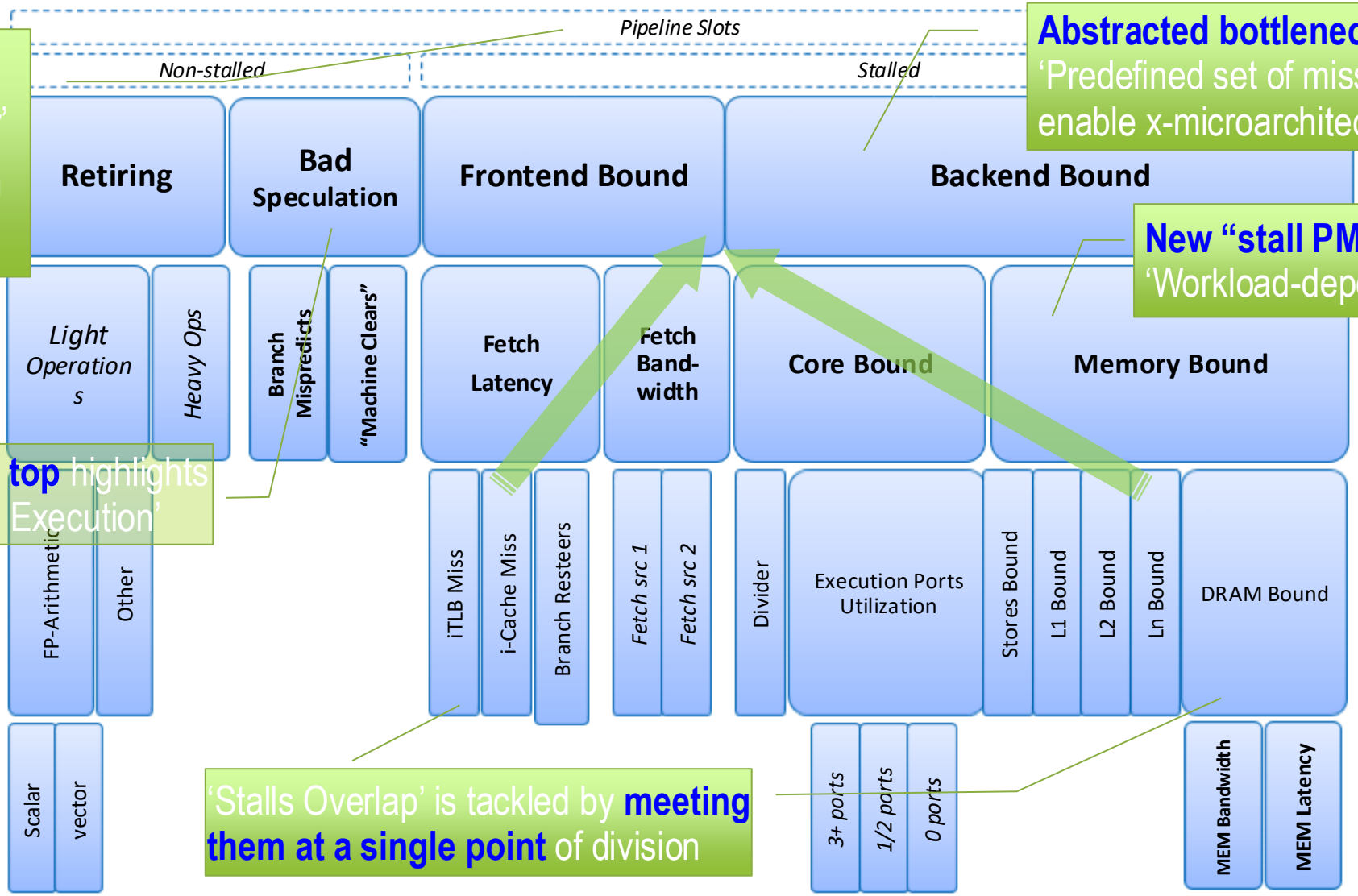
Slot-granularity solves 'Superscalar inaccuracy'

A must have for modern wider-issue cores

Abstracted bottlenecks displace 'Predefined set of miss-events' & enable x-microarchitecture comparison

New "stall PMU events" for 'Workload-dependent penalties'

Bad Speculation at the top highlights sensitivity to 'Speculative Execution'



'Stalls Overlap' is tackled by **meeting them at a single point** of division

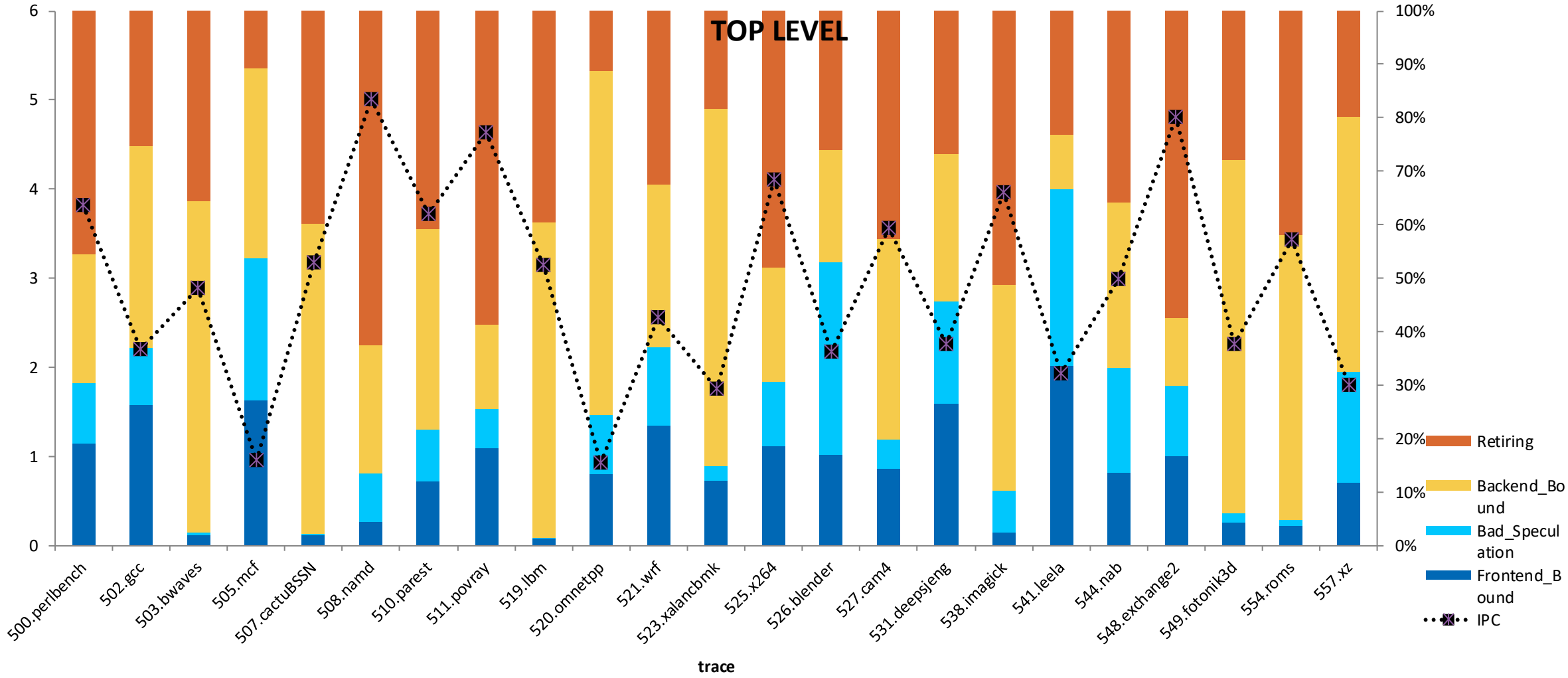
*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

Informative Metrics

- Non-tree metrics that helps to understand performance
- Also posted in:
  - perfmon/TMA_Metrics-full.xlsx at mai intel/perfmon
  - Samples listed in table

Categoy	Example Metric	Brief Description
System	CPU_Utilization	# CPUs utilized
	GFLOPs	Floating Point performance
	MEM_BW_Use	Total Memory bandwidth
	SMT_2T_Utilization	Hyperthreading usage
Memory	Load_Miss_Real_Latency	# cycles per demand load miss
	MLP	Miss Level Parallelism
	PMM_Read/Write_BW	Persistent Memory Mode
Thread	IPC (CPI)	Instructions Per Cycle
i-mix	IpTB	Instructions per Taken Branches
Core	CoreIPC	Physical core IPC
	ILP	Instruction Level Parallelism

TMA Results on SPEC17 Benchmark



TMA Adoption across architectures

	Intel		AMD	ARM	HiSilicon	Ventana
# Levels	6	4	2	1	3	2
# Metrics	>100	42	12	4	21	
ISA	X86			ARM		RISC-V
uarch (family)	P-Core	E-Core	Zen4	Neoverse		

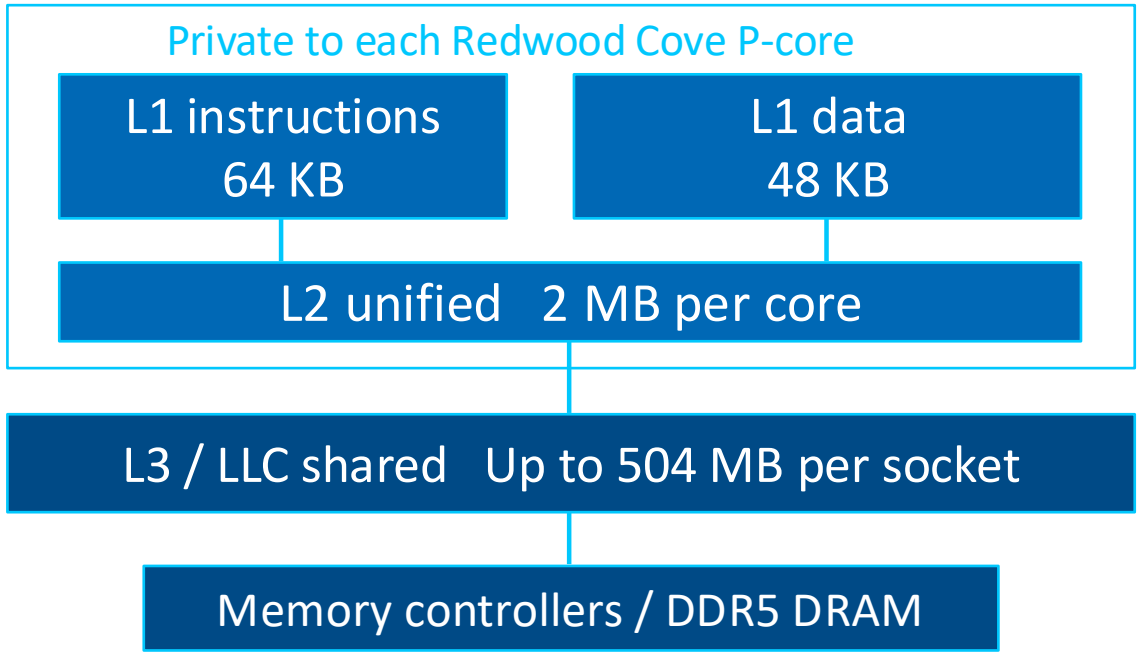
Case Study – Memory Bound Analysis

Memory Bound Breakdown

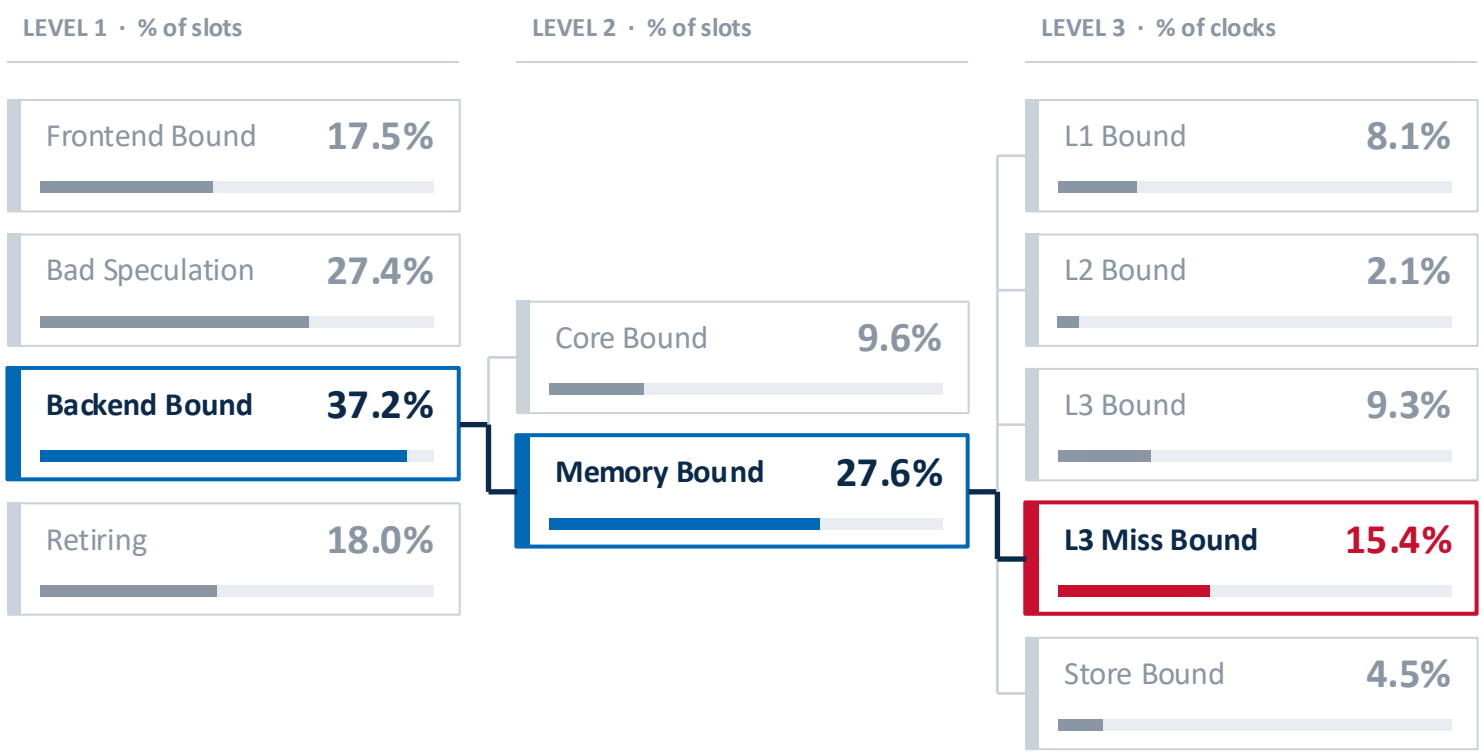
- Memory Bound in TMA is a heuristic that estimates how much of your resource stalls is due to memory operation when one or more of the following applies -
  - Load Bound - At least one load is in-flight & Execution Stall & Allocation (Resource) stall
  - Store Bound – !Load Bound & Store Buffer is full
  - Memory BW Allocation – reached the allowed BW limit.

Memory Bound Breakdown

- TMA Level-3 under Memory Bound breaks down the load bottleneck blame to the various levels of data sources.
  - L1_Bound - no in-flight load missed L1 & execution stall
  - L2_Bound – at least one load missed L1 & no load missed L2 & execution stall
  - L3_Bound - at least one load missed L2 & no load missed L3 & execution stall
  - L3Miss_Bound – at least one load missed L3 & execution stall (DRAM Bound)



TMA: the workload is Memory Bound, dominated by L3 misses



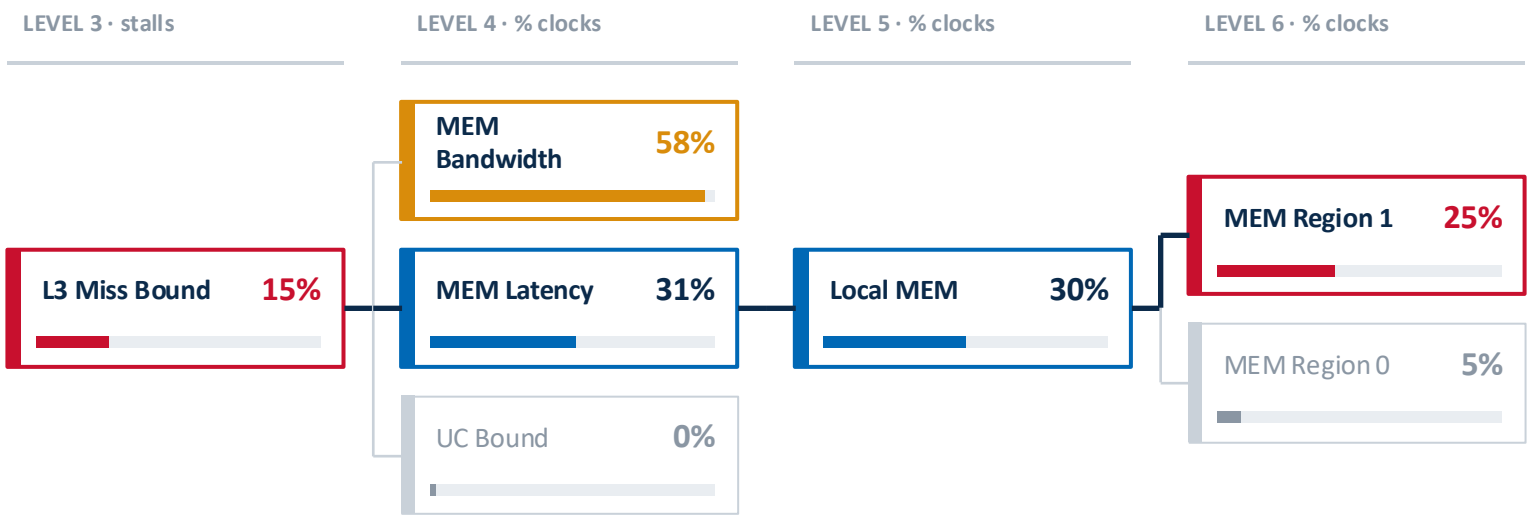
Drill-down: Backend Bound 37.2% → Memory Bound 27.6% → L3 Miss Bound 15.4%

Core 4, TMA 5.31. Bars share one 0–40% scale. Frontend, Bad Speculation and Retiring sub-trees not expanded. IPC 1.91; 3.8M L3-miss vs 22.7M L3-hit loads/s.

Reading

- **Backend Bound 37.2% → Memory Bound 27.6%**
- **L3 Miss Bound 15.4%** is the largest level-3 node
- IPC 1.91;
- 8M L3-miss loads/s vs 22.7M L3-hit loads/s

TMA levels 4–6 under L3 Miss Bound: latency, not bandwidth



critical path: **Backend → Memory → L3_Miss_Bound → MEM_Bandwidth** | levels 4–6 in % of clocks, may overlap siblings

Informative metric	Value	Meaning (tool description)
Load_L2_Miss_Latency	109 cyc	avg latency of L2-miss demand loads
Load_L2_MLP	2.5	demand L2 misses in flight
Data_L2_MLP	10.3	L2-miss data reads in flight (+HW prefetch)
Useless_HWPF	47%	L2 HW-prefetched lines never used
DRAM_BW_Use	3.7 GB/s	DRAM bandwidth, reads + writes
L2MPKI / L3MPKI	3.8 / 0.52	true misses per kilo-inst. (demand)
FB_HPKI	10.1	L1D misses merging into in-flight miss
MLP / Load_Miss_Real_Lat	2.35 / 22	L1-miss loads in flight / avg cyc

Next question:
which loads miss L3, and do they actually cost cycles?

Memory Bound Breakdown – next potential steps

- Once bottleneck is identified there are various methods of getting more info into which loads might be causing the performance bottleneck seen with TMA –
  - Sampling (PEBS) – sampling on precise load events based on data-source or using PEBS Load Latency Facility.
  - Last Branch Records (LBRs) – on top of PEBS (or as standalone) LBR with Timed LBR and Event Logging can be utilized with precise events to gain even more info surrounding the circumstances leading to the event

PEBS on L3-hit / L3-miss loads: rank loads by exposed (retire) latency

sep -ec "MEM_LOAD_L2_MISS_RETIRED.L3_MISS:sa=10007, MEM_LOAD_L2_MISS_RETIRED.L3_HIT*:sa=50021, MEM_INST_RETIRED.ALL_LOADS:sa=5000011" -apc basic

Load IP	Function	Instruction (XED)	% of L3-miss exposed cycles	Retire lat. miss / hit	L3 miss ratio
0x478698	cOutVector::recordWithTimestamp	movzx eax, byte ptr [rcx+0x49]	**25.6**	235 / 5	18%
0x4c12ef	EtherMAC::startFrameTransmission	mov rax, qword ptr [r14]	**12.9**	406 / 41	64%
0x459e7d	cNamedObject::setName	cmp byte ptr [r14], 0x0	**10.9**	369 / 9	49%
0x4141ef	cGate::deliver	mov rcx, qword ptr [rdi+0x28]	6.2	328 / 10	10%
0x479410	cQueue::pop	mov rax, qword ptr [rbx]	6.1	341 / 39	22%
0x4b43ea	MACRelayUnitBase::getPortForAddress	movbe r11d, dword ptr [rbx+0x20]	3.8	499 / 65	26%

Key numbers

- L3-miss load: **200 cyc** avg retire latency
- L3-hit load: **12.7 cyc**; all loads: 3.9
- 10 load IPs = **76.8%** of all exposed L3-miss cycles
- 96.8% in the application binary (not libc / libstdc++)

How it is computed

- Exposed cycles ≈ samples × sample-after × avg retire latency
- Retire latency = pebs_retire_latency[15:0] (bit 16 = valid)
- L3 miss ratio = L3-miss / (L3-miss + L3-hit) instances at that IP

LBRs - same load, two arrival paths: miss vs hit

Load 0x4BC3CC cmp qword ptr [rdi+0x18], rbx in EtherFrameWithLLC::~EtherFrameWithLLC (macro-fused with jnz → PEBS IP = 0x4BC3D0)

L3 MISS sample event L3_MISS | retire latency **241 cyc**

destructor tail-called from EtherMACBase::handleEndTxPeriod (frame done transmitting)

#	from → to	type	cyc	L3M	L3H	L1H
0	**EtherFrameWithLLC::~EtherFrameWithLLC+0x30 → EtherFrameWithLLC::~EtherFrameWithLLC+0x4a**	COND	**457**	**2**	0	1
1	**handleEndTxPeriod+0x19a → EtherFrameWithLLC::~EtherFrameWithLLC+0x0**	IND_JMP	**2**	**1**	0	2
2	handleEndTxPeriod+0x158 → handleEndTxPeriod+0x18d	COND	2	0	0	2
3	handleEndTxPeriod+0xf5 → handleEndTxPeriod+0x14d	COND	2	0	0	2

```
; caller block (LBR #2.to .. #1.from) - EtherMACBase::handleEndTxPeriod
4c713d: mov rax, qword ptr [r14]   ; frame vtable: L3 miss logged (#1)
4c7140: mov rdi, r14 ; add rsp, 0x8 ; pop rbx ; pop r14
4c714a: jmp qword ptr [rax+0x20]   ; virtual ~EtherFrameWithLLC (tail call)
; load block (LBR #1.to .. #0.from) - 457 cyc, 2 L3 misses logged
4bc3a0: push rbx ; mov rbx, rdi
4bc3a4: mov qword ptr [rdi], 0x5a3490
4bc3ab: mov rdi, qword ptr [rdi+0xb0]   ; frame->owner
4bc3b2: test rdi, rdi ; jz 0x4bc3ea
4bc3b7: movzx eax, word ptr [rdi+0xb8]
4bc3be: test ax, ax ; jz 0x4bc3dc
4bc3c3: dec eax ; mov word ptr [rdi+0xb8], ax
4bc3cc: cmp qword ptr [rdi+0x18], rbx ; <== L3 MISS (241 cyc retire lat.)
4bc3d0: jnz 0x4bc3ea              ; fused with the cmp = LBR #0.from
```

HIT sample event L1_HIT | retire latency **3 cyc**

destructor called from EtherMACBase::checkDestinationAddress (received frame dropped)

#	from → to	type	cyc	L3M	L3H	L1H
0	**EtherFrameWithLLC::~EtherFrameWithLLC+0x30 → EtherFrameWithLLC::~EtherFrameWithLLC+0x4a**	COND	40	0	0	2
1	**checkDestinationAddress+0x130 → EtherFrameWithLLC::~EtherFrameWithLLC+0x0**	IND_CALL	2	0	0	2
2	cOutVector::recordWithTimestamp+0x19c → checkDestinationAddress+0x129	RET	2	0	0	3
3	cOutVector::recordWithTimestamp+0x140 → cOutVector::recordWithTimestamp+0x191	COND	4	0	1	3

```
; caller block (LBR #2.to .. #1.from) - EtherMACBase::checkDestinationAddress
4c5d79: mov rdi, qword ptr [rsp]   ; the frame just received
4c5d7d: mov rax, qword ptr [rdi]
4c5d80: call qword ptr [rax+0x20]  ; virtual ~EtherFrameWithLLC
; load block (LBR #1.to .. #0.from) - 40 cyc, no L3 miss logged
4bc3a0: push rbx ; mov rbx, rdi
4bc3a4: mov qword ptr [rdi], 0x5a3490
4bc3ab: mov rdi, qword ptr [rdi+0xb0]   ; frame->owner
4bc3b2: test rdi, rdi ; jz 0x4bc3ea
4bc3b7: movzx eax, word ptr [rdi+0xb8]
4bc3be: test ax, ax ; jz 0x4bc3dc
4bc3c3: dec eax ; mov word ptr [rdi+0xb8], ax
4bc3cc: cmp qword ptr [rdi+0x18], rbx ; <== L1 HIT (3 cyc retire lat.)
4bc3d0: jnz 0x4bc3ea              ; fused with the cmp = LBR #0.from
```

Pattern: same instruction, different way in. Via TX completion the frame sat in the queue → frame and its owner are cold: event logging shows an L3 miss in the caller block (frame vtable) and 2 L3 misses in the destructor block, which took 457 cycles. Via RX drop the frame was touched just before → 40-cycle block, L1 hit. All samples of this load: 103 of its 110 L3-miss samples arrive via handleEndTxPeriod; all 185 samples arriving via checkDestinationAddress are L1 hits.

How do I use TMA ?

- TMA has been integrated in many performance analysis tools, being part of the basic uarch exploration analysis
  - Intel® VTune™ Profiler
  - Linux perf
  - perf-tools (on top of Linux perf and pmu-tools) - https://github.com/aayasin/perf-tools
  - PerfSpect - intel/PerfSpect
  - pmu-tools (on top of Linux Perf) - andikleen/pmu-tools: Intel PMU profiling tools
  - Intel EMON Data Processing (EDP) (over SEP, bundled in VTune Package)
  - And many others…


intel
Thank You!

