Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

February 25th, 2026

TopDown Methodology

Harald Servat, PhD (Intel)

Victor Xirau (BSC)

It's the Memory, Stupid!

intel

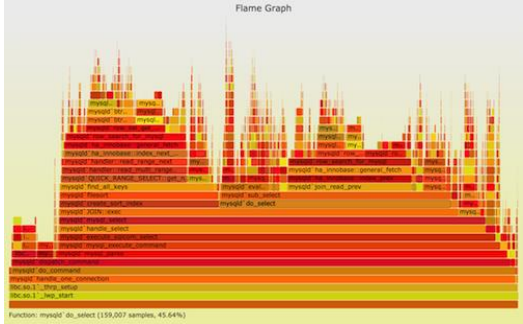
Performance analysis is (inherently) complex

- Performance analysis at multiple levels / dimensions among
  - System-wide: memory, I/O (network, disk,…), uncore/offcore
  - Application-wide
    - Programming models (OpenMP, MPI, CUDA, HIP, OpenCL, L0,…)
    - Application performance specifics (e.g. FoM)

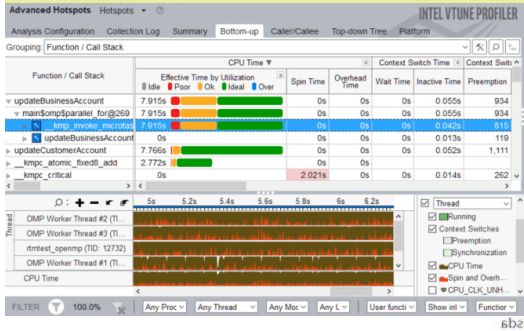
- CPU core-focused performance analyses – for quickly identifying bottlenecks in processors
  - Itanium (~2001 – Chamberlain et al.), Itanium2 (~2006 – Levinthal et al.)
  - Core2Duo (2005 – Levinthal et al.)
  - IBM Power 5 / CPIstack (2005 – Maron et al.)
  - Top Down Methodology [TMA] (2014 – Yasin)

- Ideally, CPU core-focused analysis shall be used once:
  - Tuned hardware (production system, performance configuration)
  - Completed algorithmic tuning (parallelism, …)

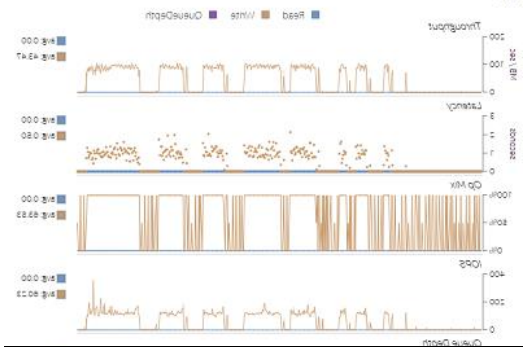
flamegraph/brendangregg



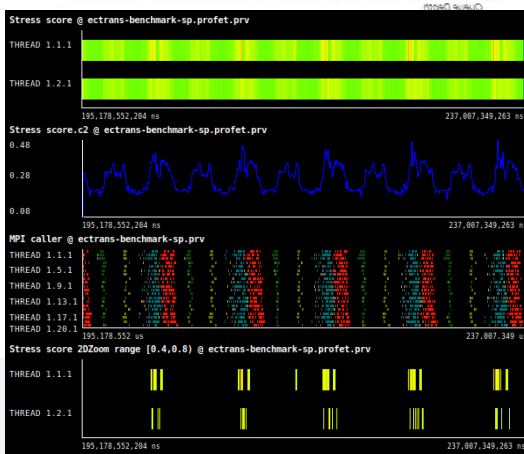
Intel® Vtune™



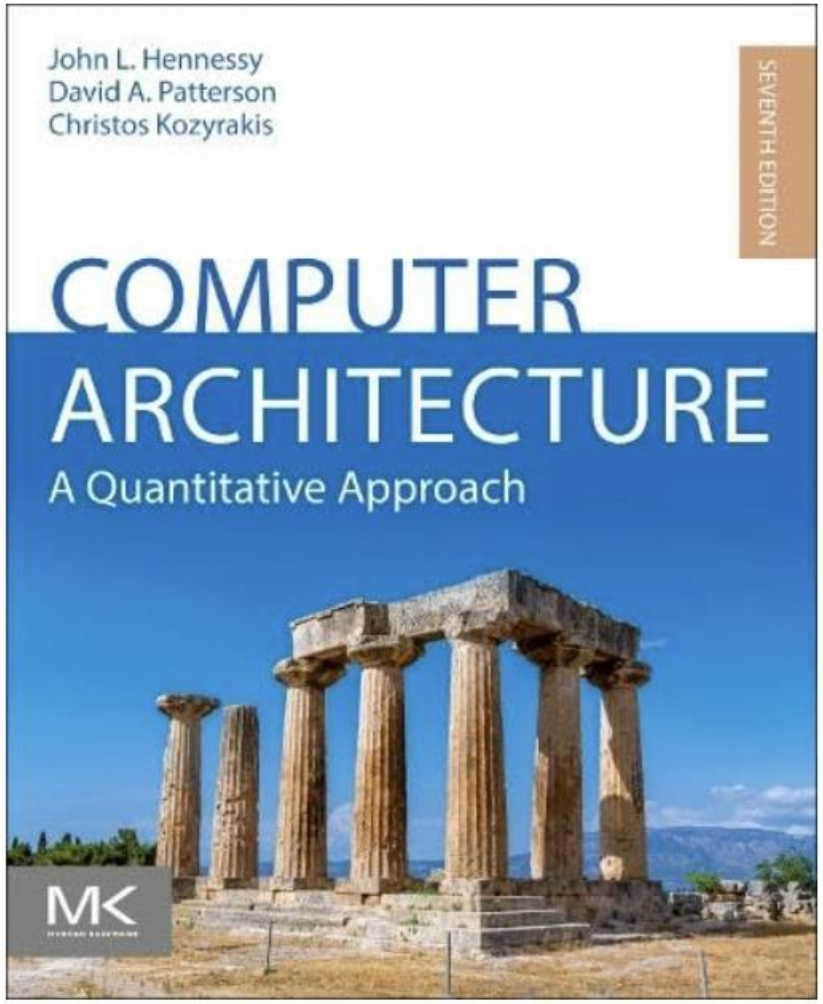
Intel® Platform Profiler

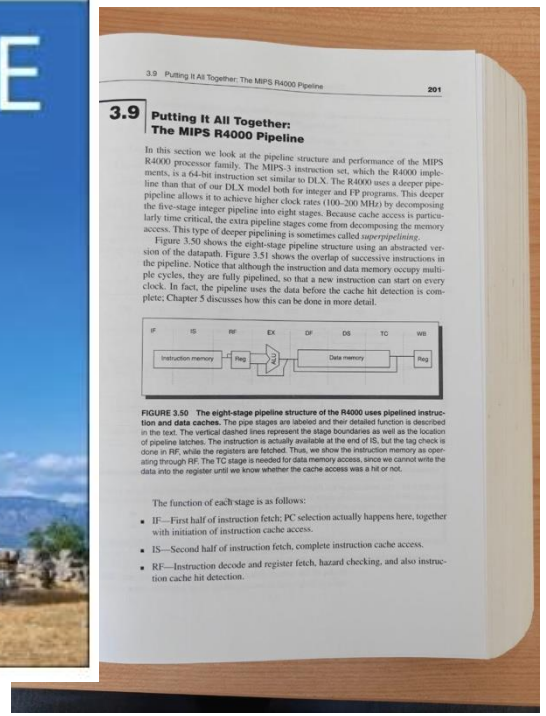


Paraver



Let’s go to the basics

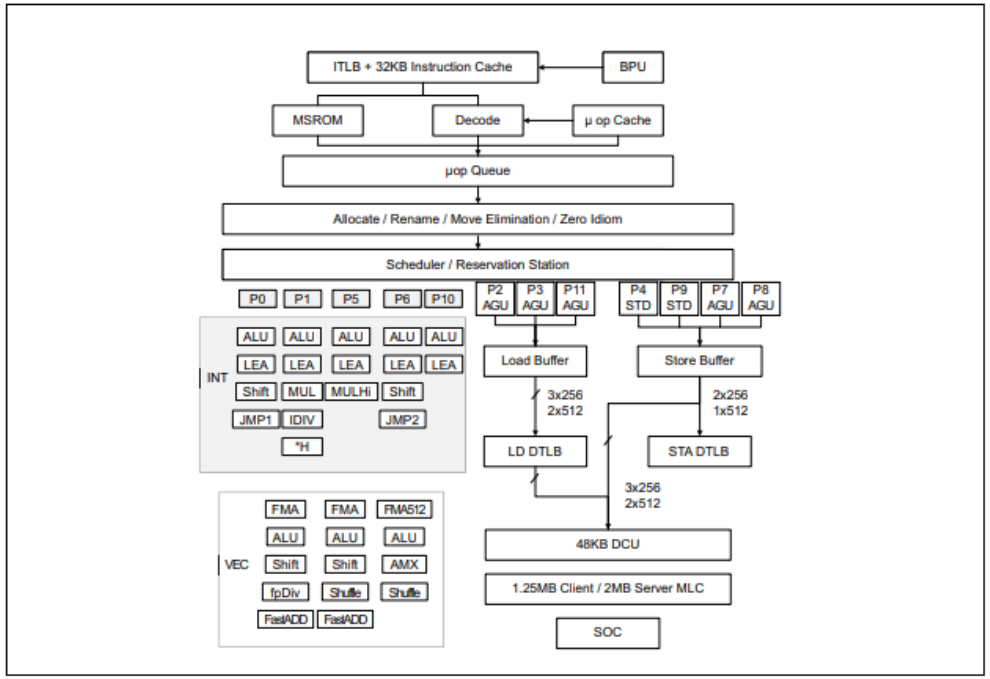




Picture taken from 2nd edition

2.3.1 Golden Cove Microarchitecture Overview

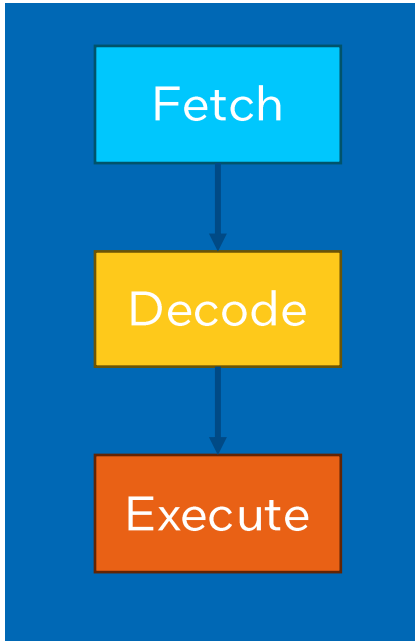
The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



https://www.intel.com/content/www/us/en/content-details/671488/intel-64-and-ia-32-architectures-optimization-reference-manual-volume-1.html

CPU uarchitecture background

Simplified
CISC CPU operation



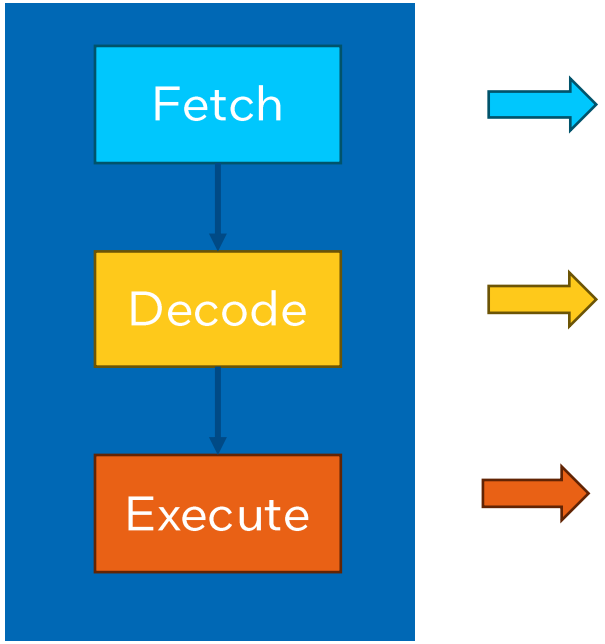
Retrieve the next block of code / instructions

Break blocks of code into instructions and
if instructions are complex further break into uops

Perform the instructions/uops

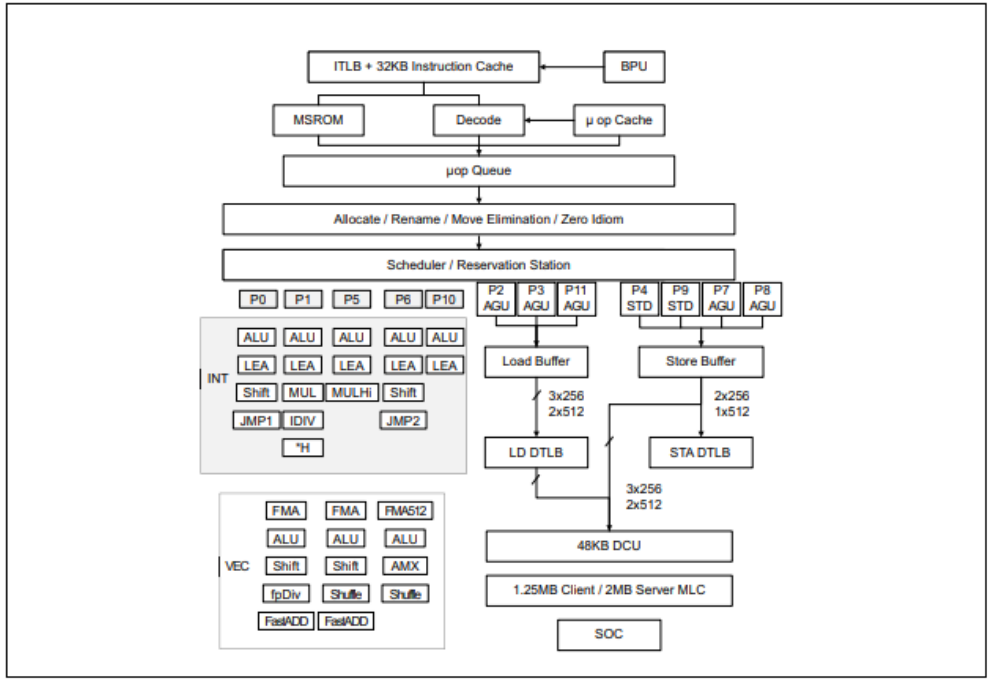
CPU uarchitecture background

Simplified
CISC CPU operation



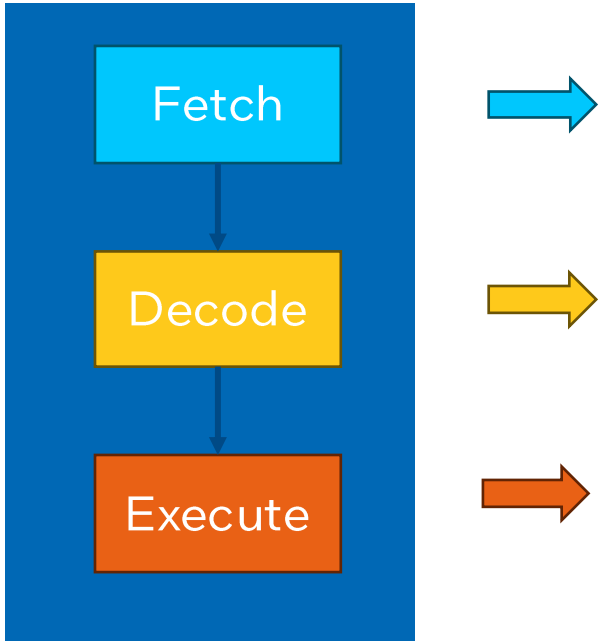
2.3.1 Golden Cove Microarchitecture Overview

The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



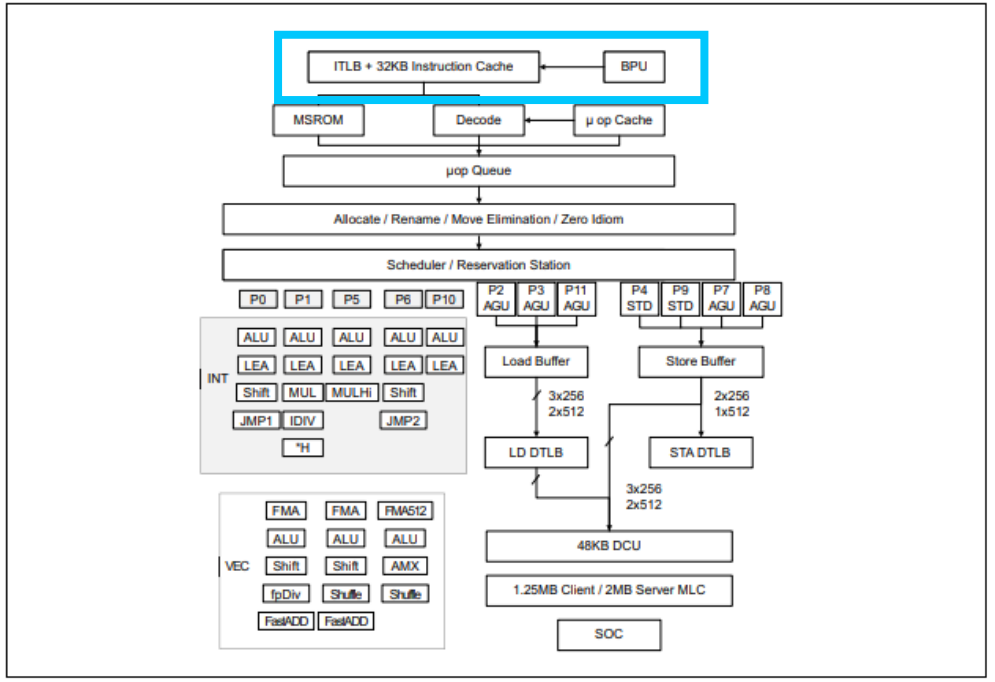
CPU uarchitecture background

Simplified
CISC CPU operation



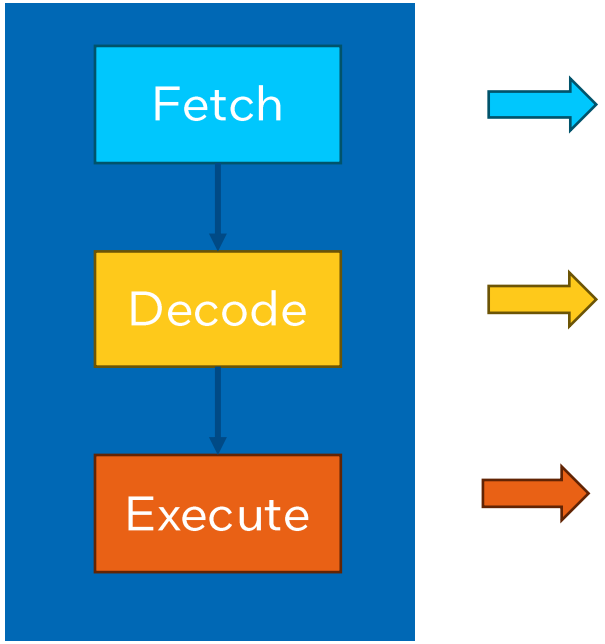
2.3.1 Golden Cove Microarchitecture Overview

The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



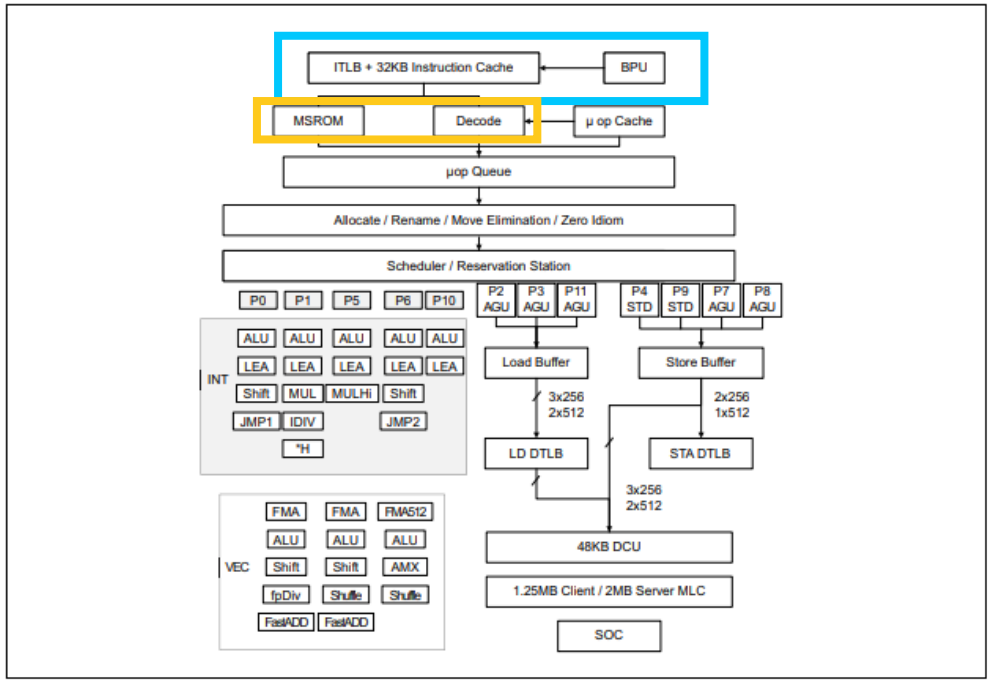
CPU uarchitecture background

Simplified
CISC CPU operation



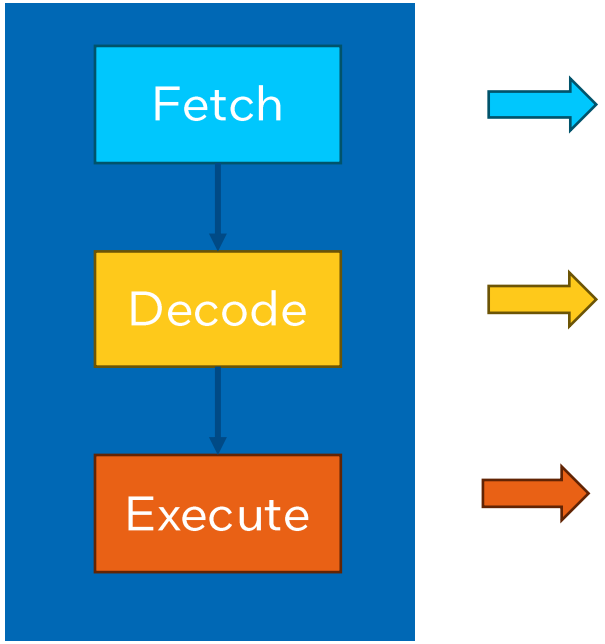
2.3.1 Golden Cove Microarchitecture Overview

The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



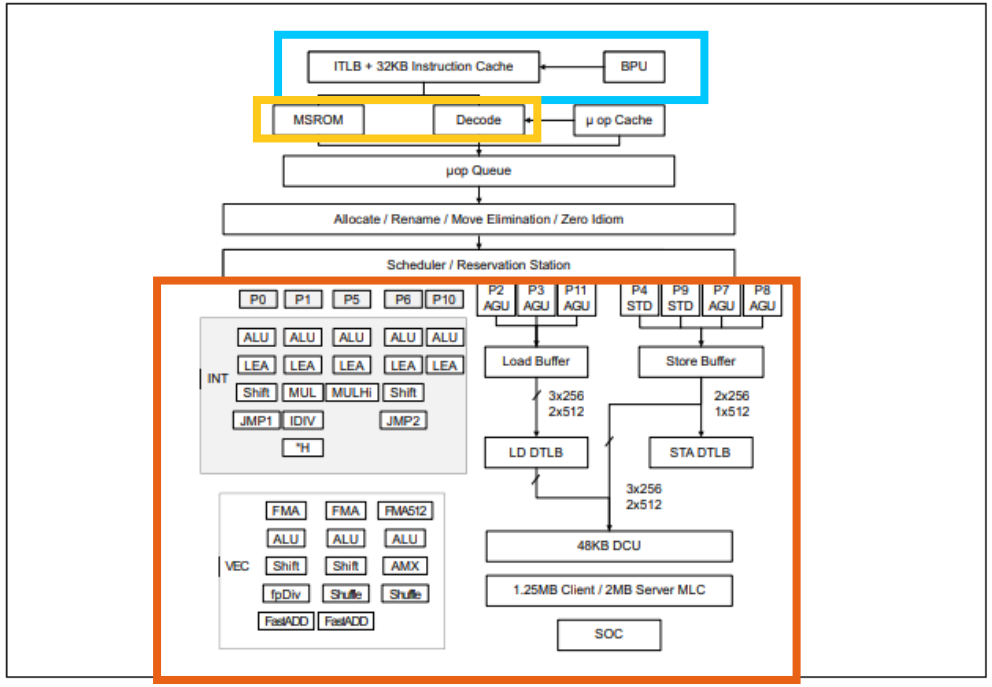
CPU uarchitecture background

Simplified
CISC CPU operation



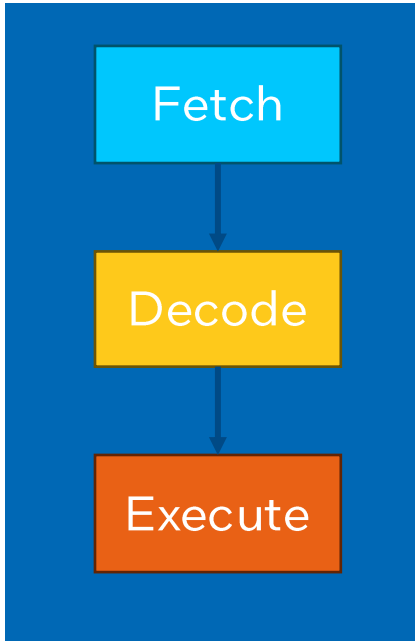
2.3.1 Golden Cove Microarchitecture Overview

The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



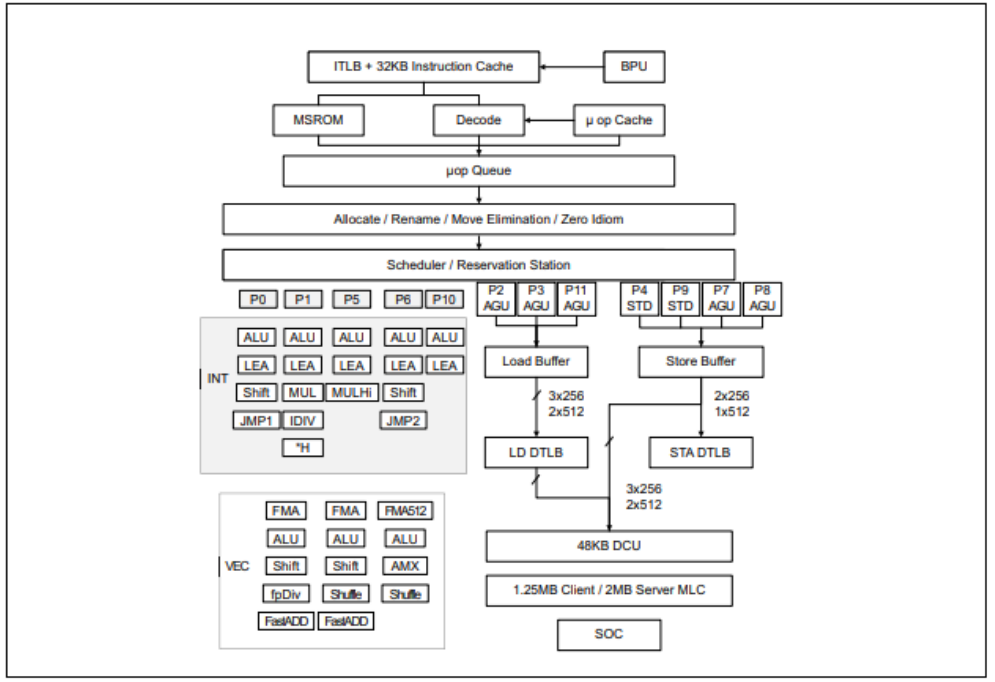
Making the HW faster!

Simplified
CISC CPU operation



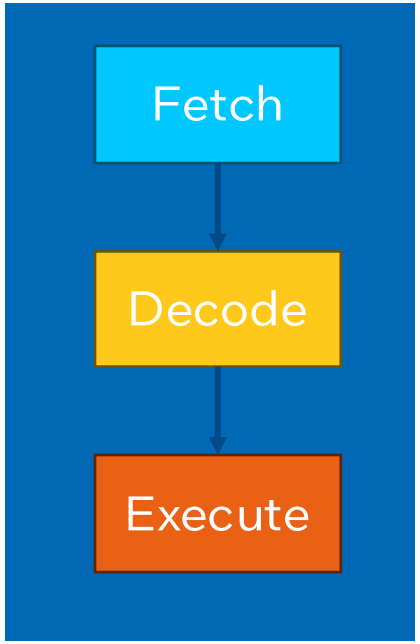
2.3.1 Golden Cove Microarchitecture Overview

The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



Making the HW faster!

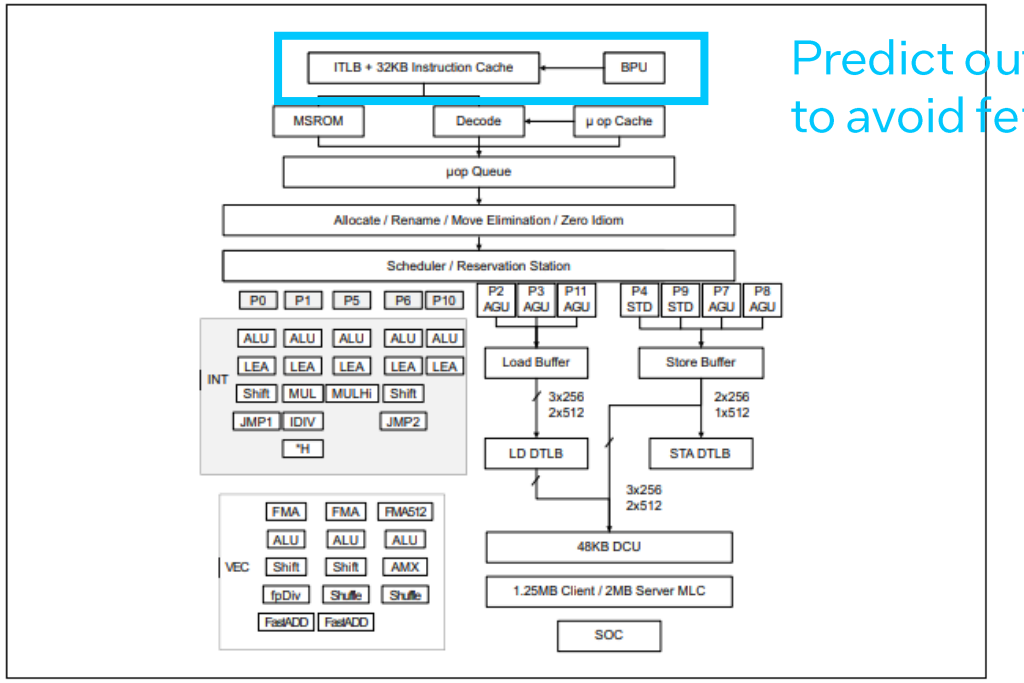
Simplified
CISC CPU operation



Keep instructions in a cache to prevent fetching from main memory in the future

2.3.1 Golden Cove Microarchitecture Overview

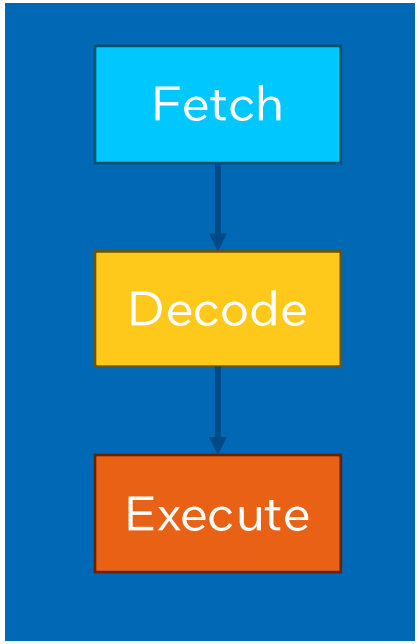
The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



Predict outcome of branches to avoid fetching to stall

Making the HW faster!

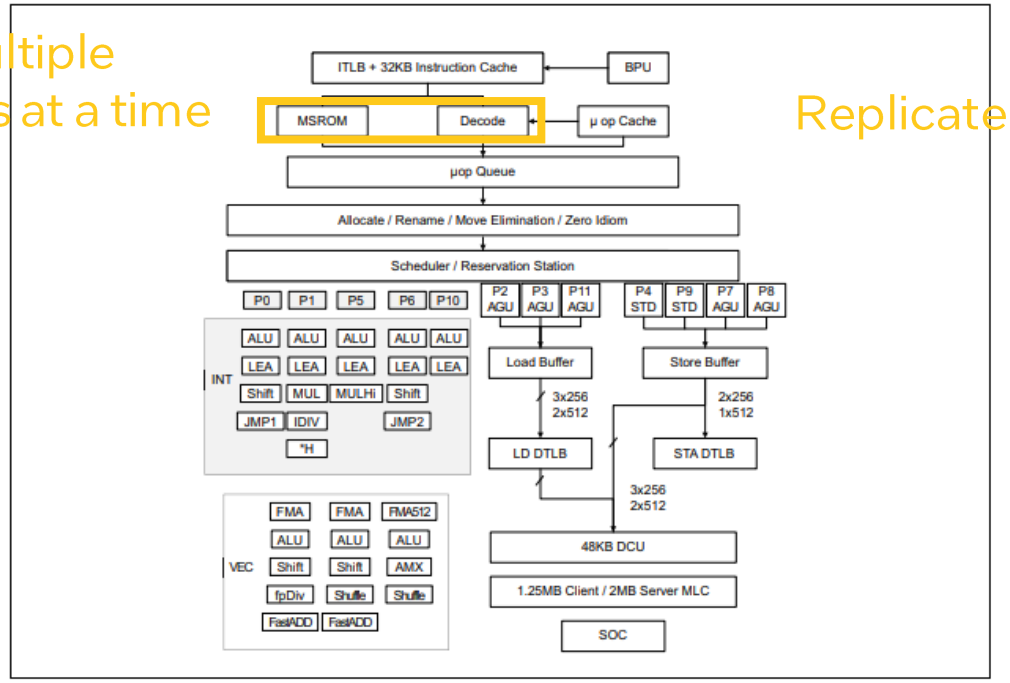
Simplified
CISC CPU operation



Decode multiple instructions at a time

2.3.1 Golden Cove Microarchitecture Overview

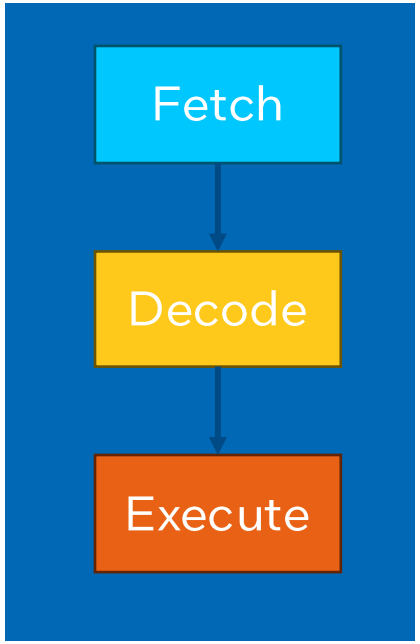
The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



Replicate decode logic

Making the HW faster!

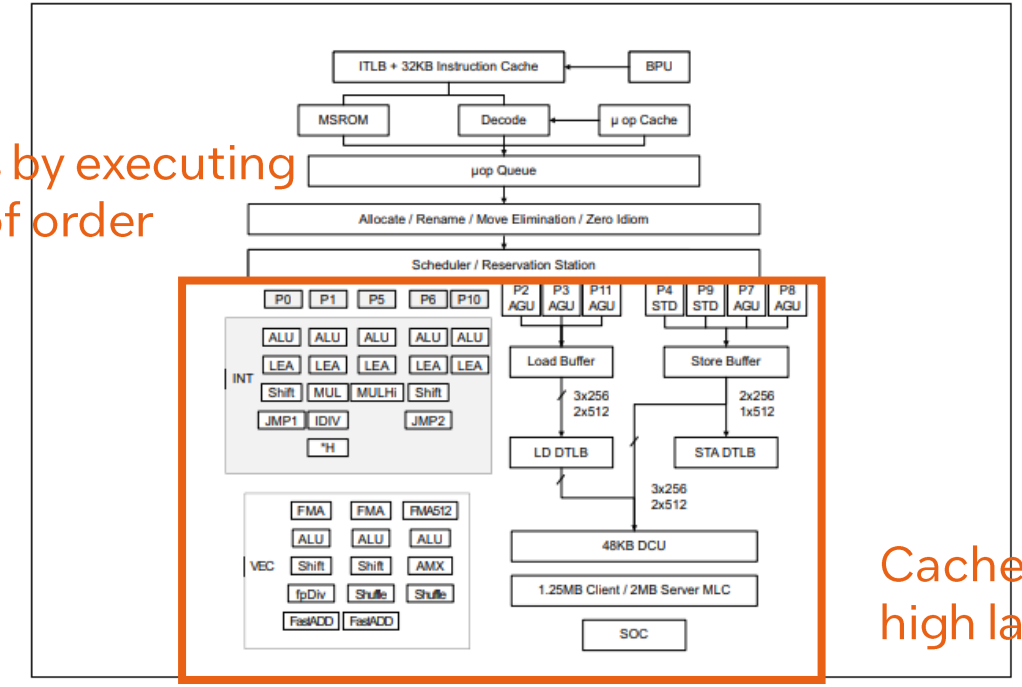
Simplified
CISC CPU operation



Hide stalls by executing uops out of order

2.3.1 Golden Cove Microarchitecture Overview

The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.

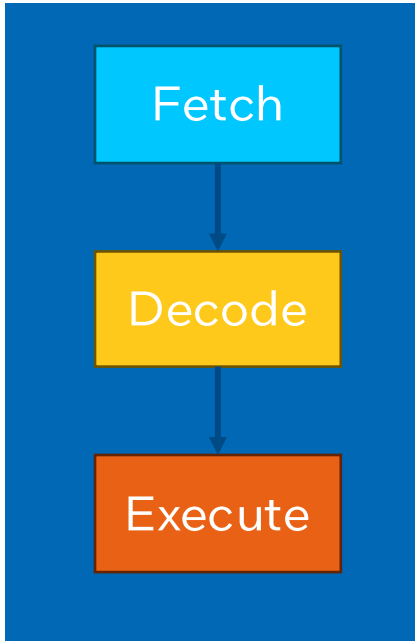


Cache data to avoid high latency accesses

Replicate hardware logic
to execute many instructions/uops simultaneously

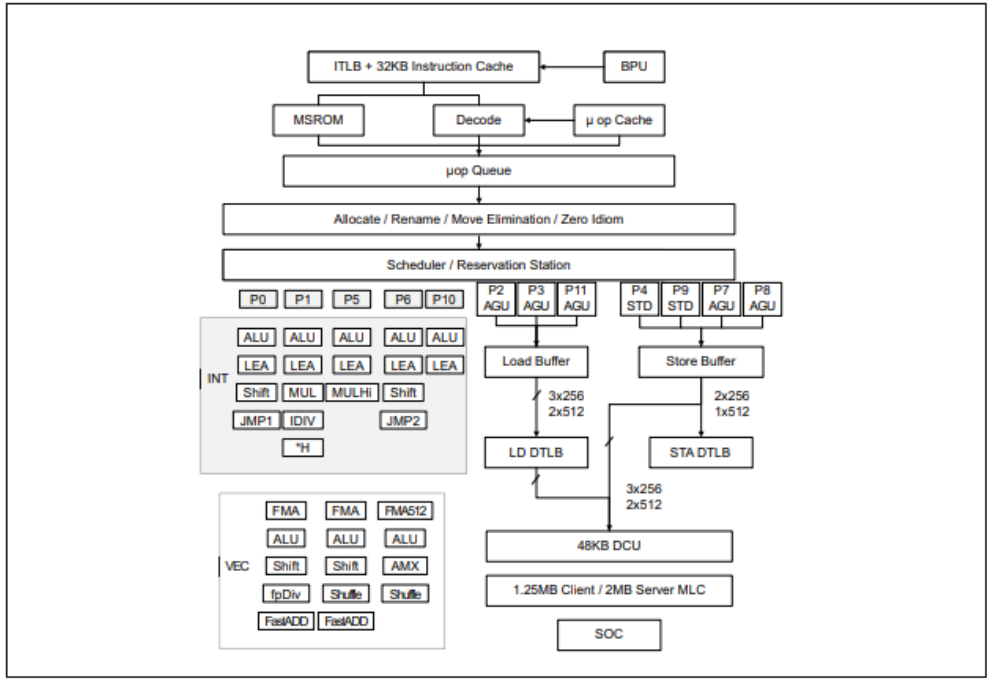
…. what about SW?

Simplified
CISC CPU operation



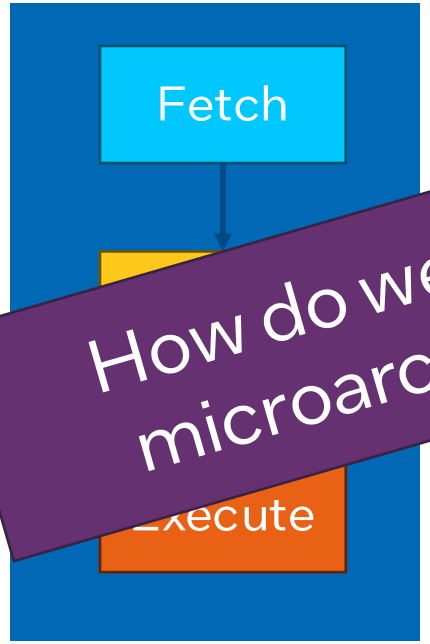
2.3.1 Golden Cove Microarchitecture Overview

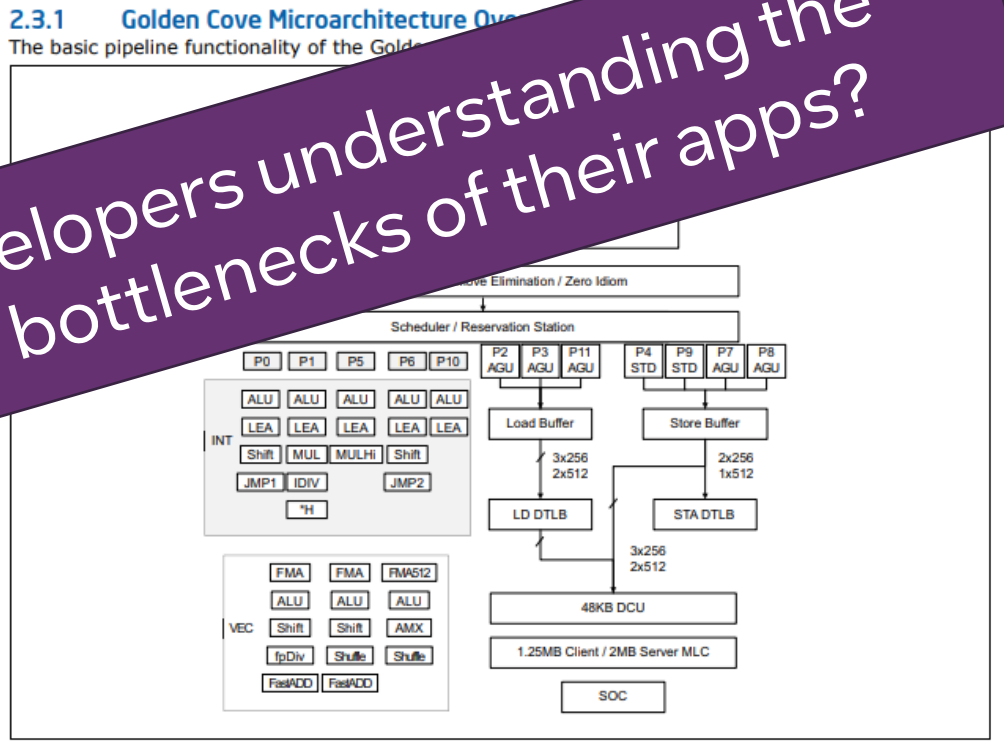
The basic pipeline functionality of the Golden Cove microarchitecture is depicted in Figure 2-1.



…. what about SW?

Simplified
CISC CPU operation

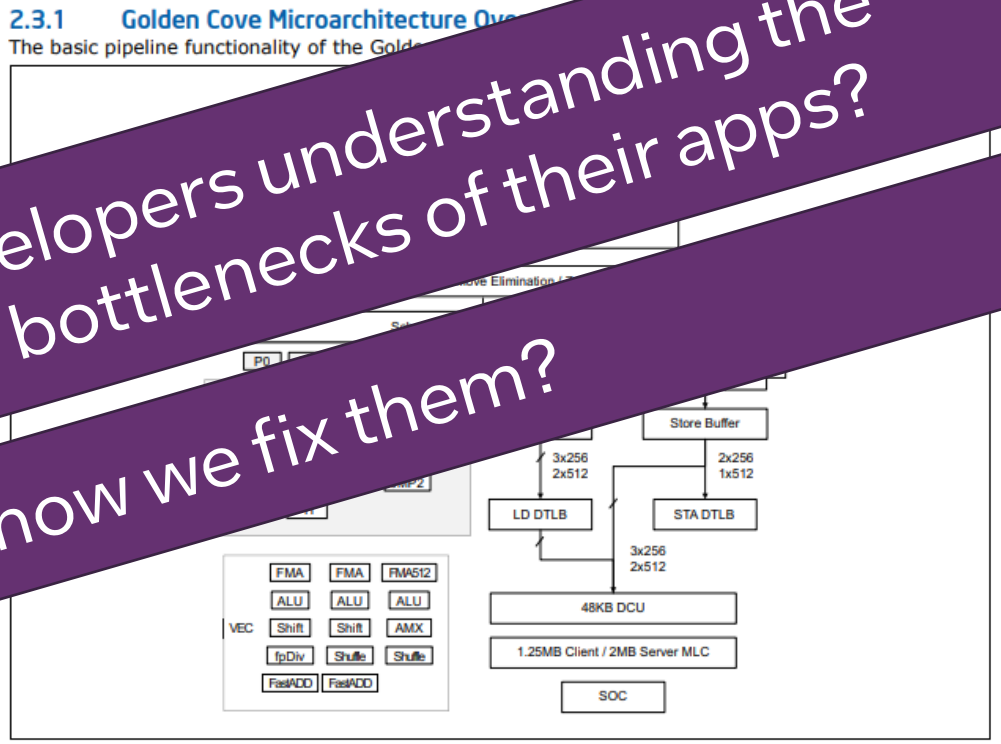




How do we help developers understanding the microarchitectural bottlenecks of their apps?

…. what about SW?

Simplified
CISC CPU operation



How do we help developers understanding the microarchitectural bottlenecks of their apps?

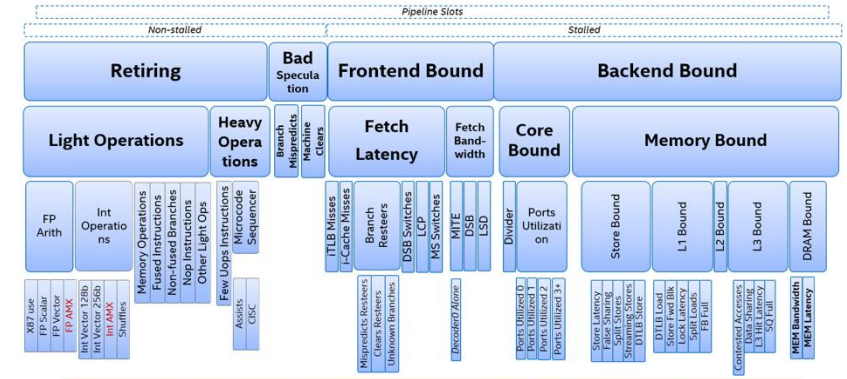
… and how we fix them?

What is Top Down?

Top Down Methodology (TMA)

- Top Down (TMA) is a hierarchical performance analysis technique for identifying software bottlenecks that occur in the core
- TMA categorizes CPU pipeline slots into high-level categories
  - **Retiring**: Useful work slots (maximizes IPC, ideal ~100% on peak throughput)
  - **Bad Speculation**: Wasted slots from branch mispredictions or exceptions
  - **Frontend Bound**: Delays fetching/decoding instructions (e.g., I-cache misses)
  - **Backend Bound**: Execution stalls (e.g., memory, core bound subcategories)
- TMA drills the hierarchy down iteratively using performance counters until narrowing the bottleneck
- Low-cost & Comprehensive

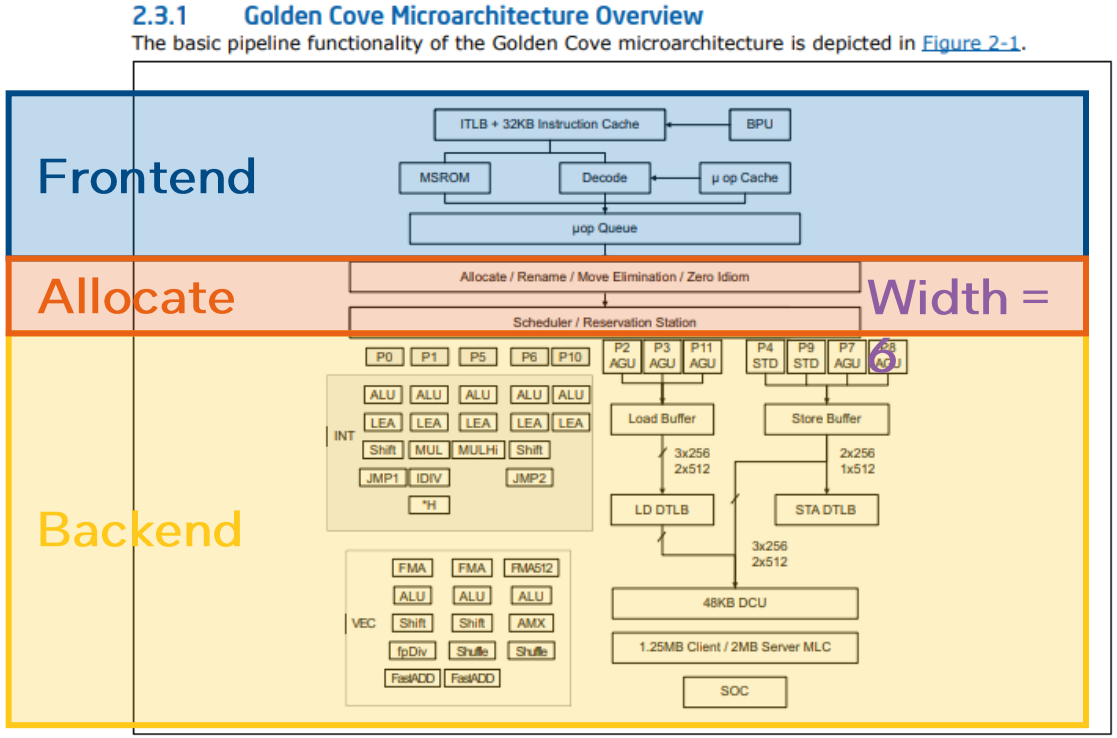
Top-down* μarch Analysis: TMA 4.4 tree. 4-levels [ADL, SPR]



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

Top Down 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
  - Width: how many uops/instructions can be operated at once (either Allocation or Retirement stages)
  - Backend: includes all the execution and retirement logic



Top Down Key Concepts

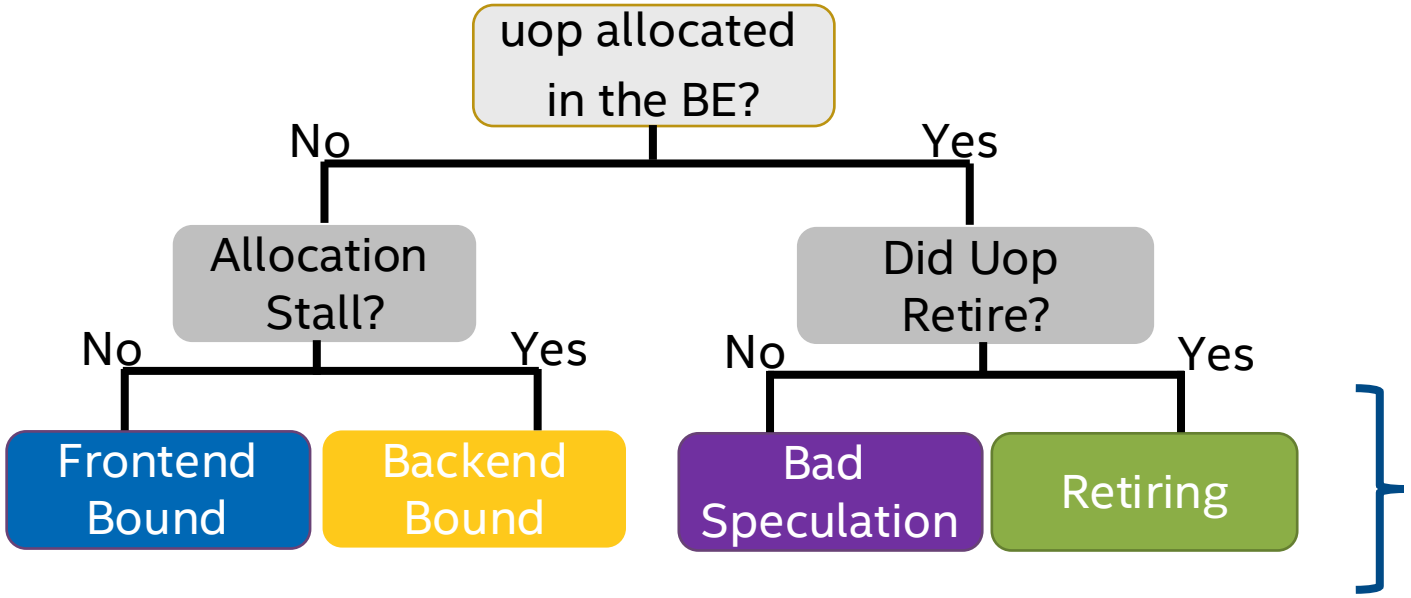
- The novel concept that Top Down introduces is called Pipeline Slot

- 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
  - Allocation width = 6 on GoldenCove / SapphireRapids

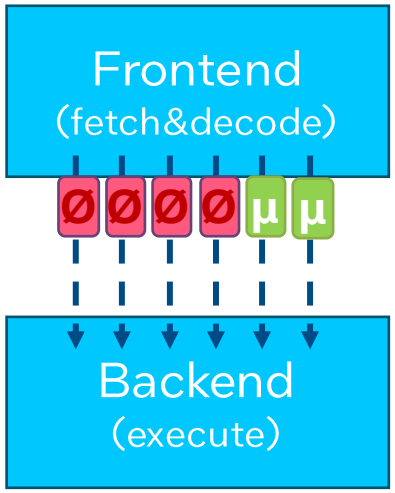
Top down issue classification

- Performance issues are classified to what happened to each pipeline slot



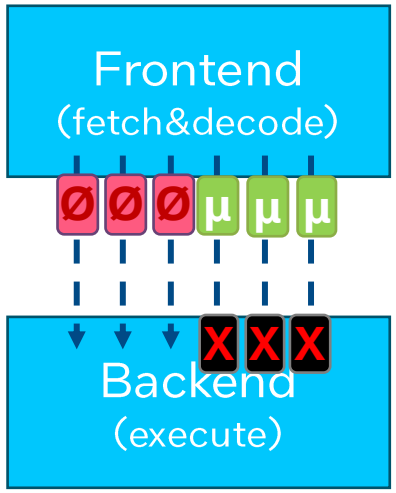
The top 4 categories of Top Down (TMA Level 1)

Visualizing Top Down (L1)



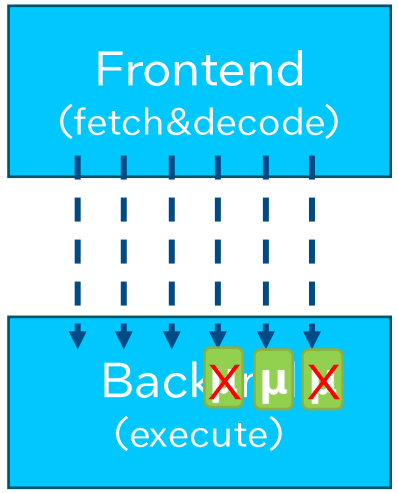
Frontend Bound

Frontend does not provide a uop while backend is not stalled



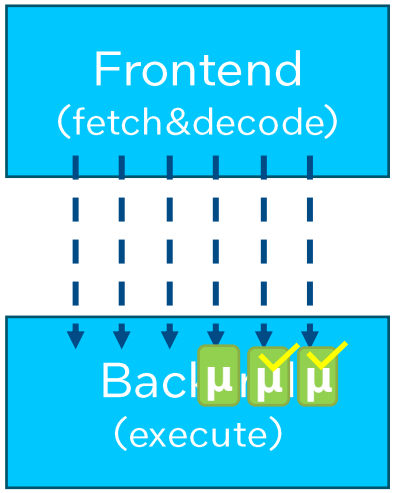
Backend Bound

Frontend has 1+ uops ready but backend is stalled



Bad Speculation

uops are allocated, but later cancelled

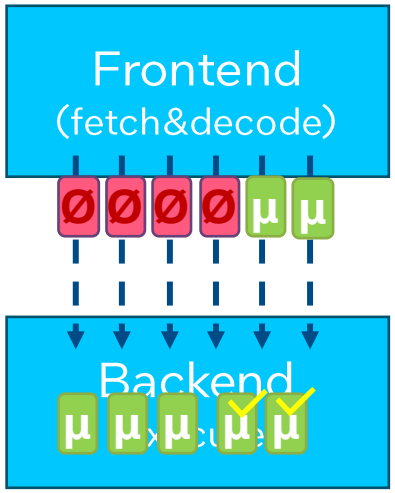


Retiring
(good!)

uops complete execution and instructions retire

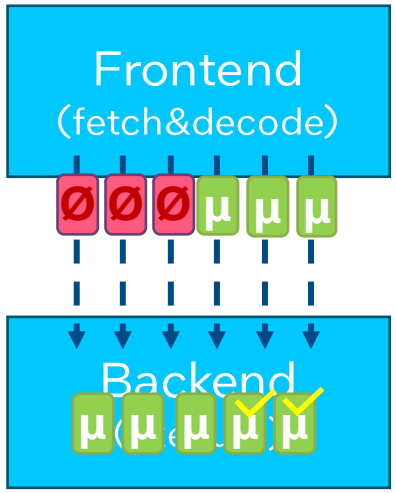
Visualizing Top Down (L1), 4 hypothetical cycles

Cycle N



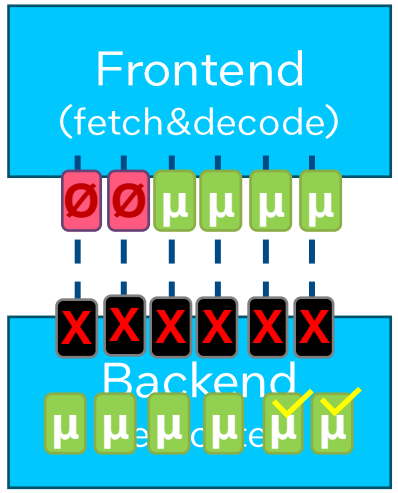
4 slots FE bound
2 slots Retiring

Cycle N+1



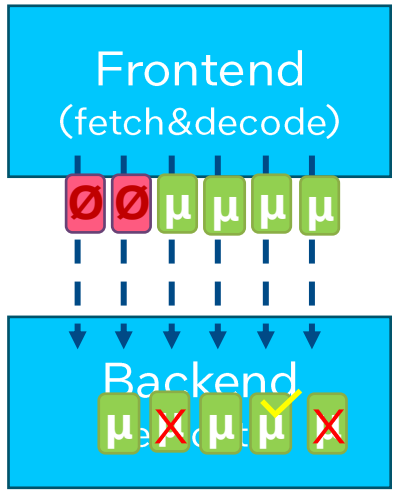
3 slots FE bound
2 slots Retiring

Cycle N+2



6 slots BE bound
2 slots Retiring

Cycle N+3



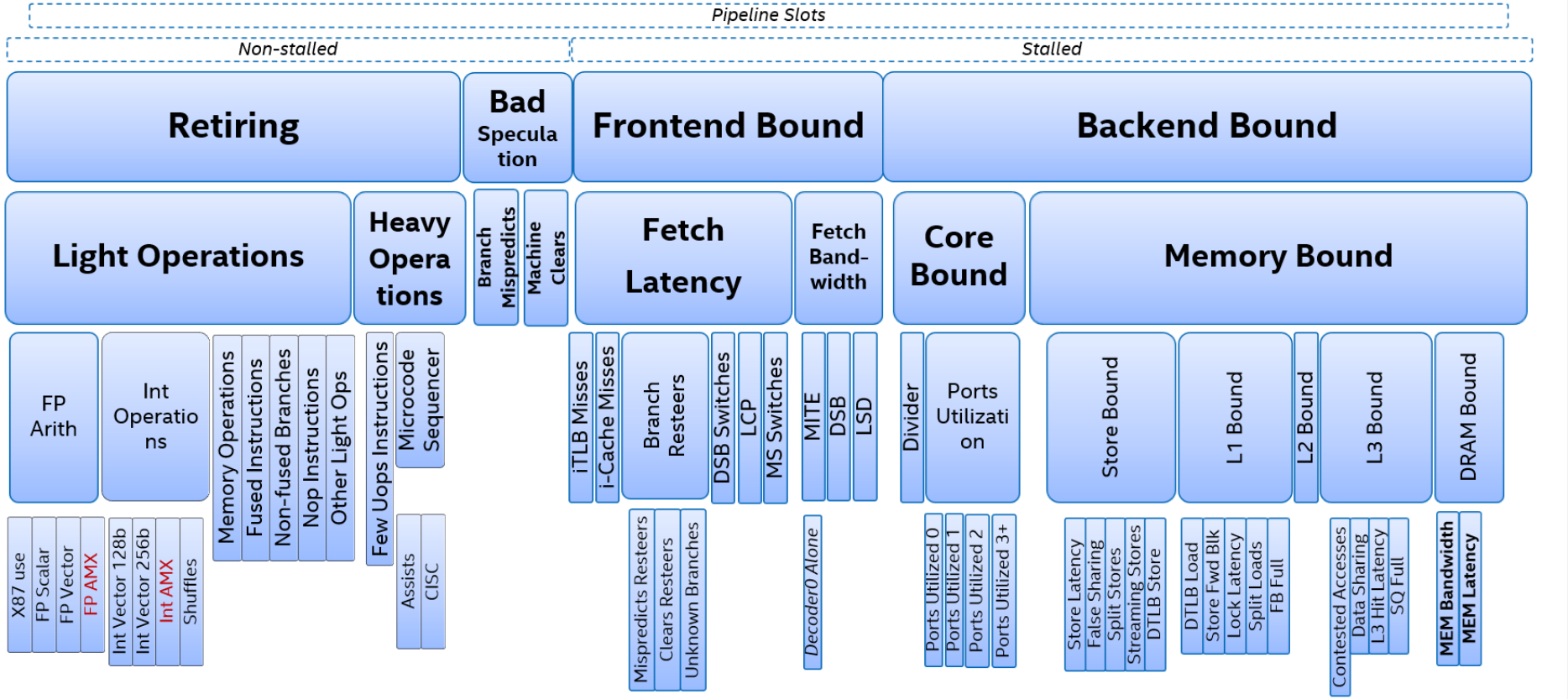
2 slots FE bound
2 slots Bad Speculation
1 slot Retiring

Summary:
Allocation width = 6
9 slots FE bound
6 slots BE bound
2 slots BS
7 slots Retiring

How is Top Down used?

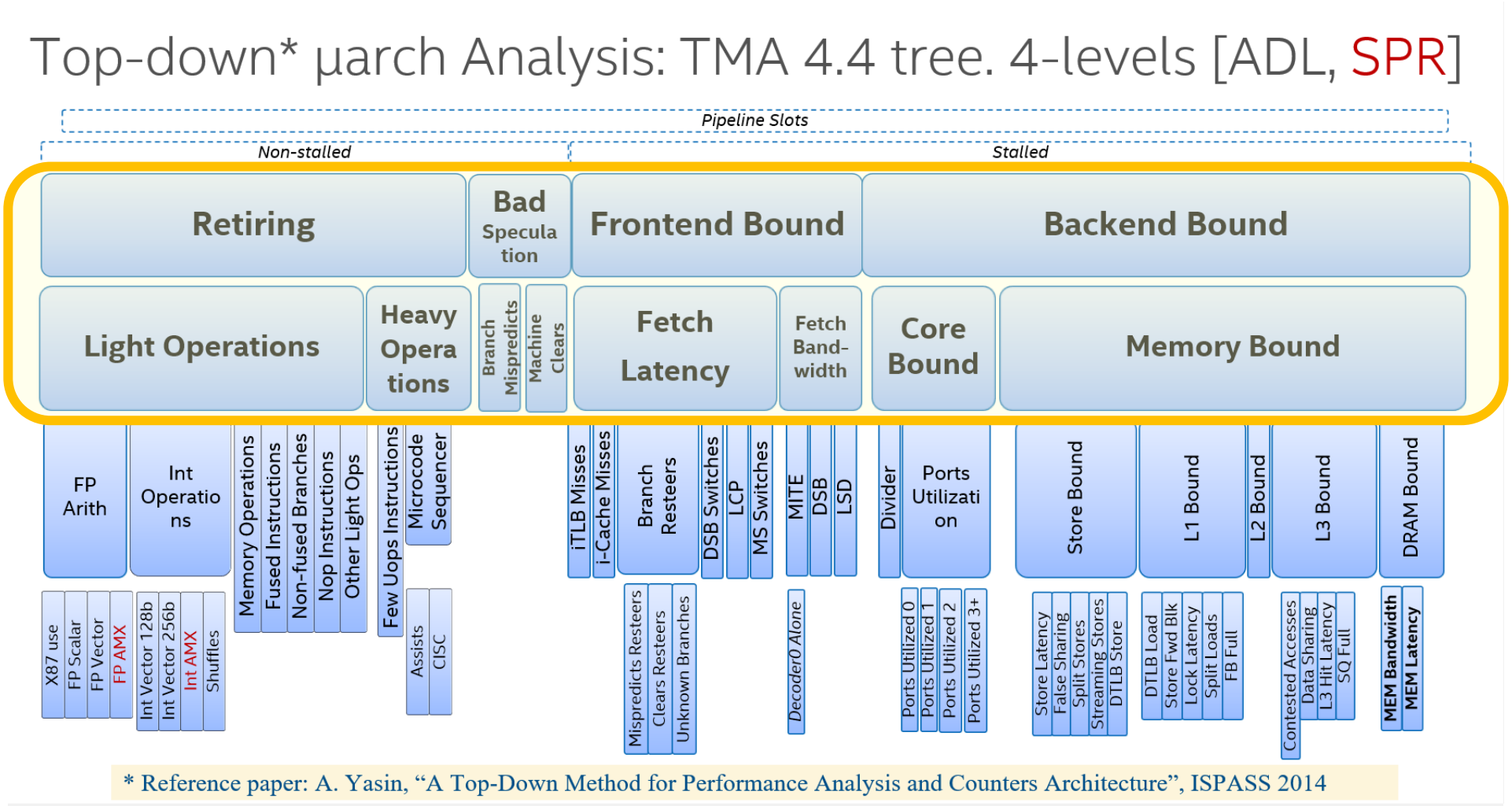
The complete TMA "tree"

Top-down* µarch Analysis: TMA 4.4 tree. 4-levels [ADL, SPR]



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

The complete TMA "tree"

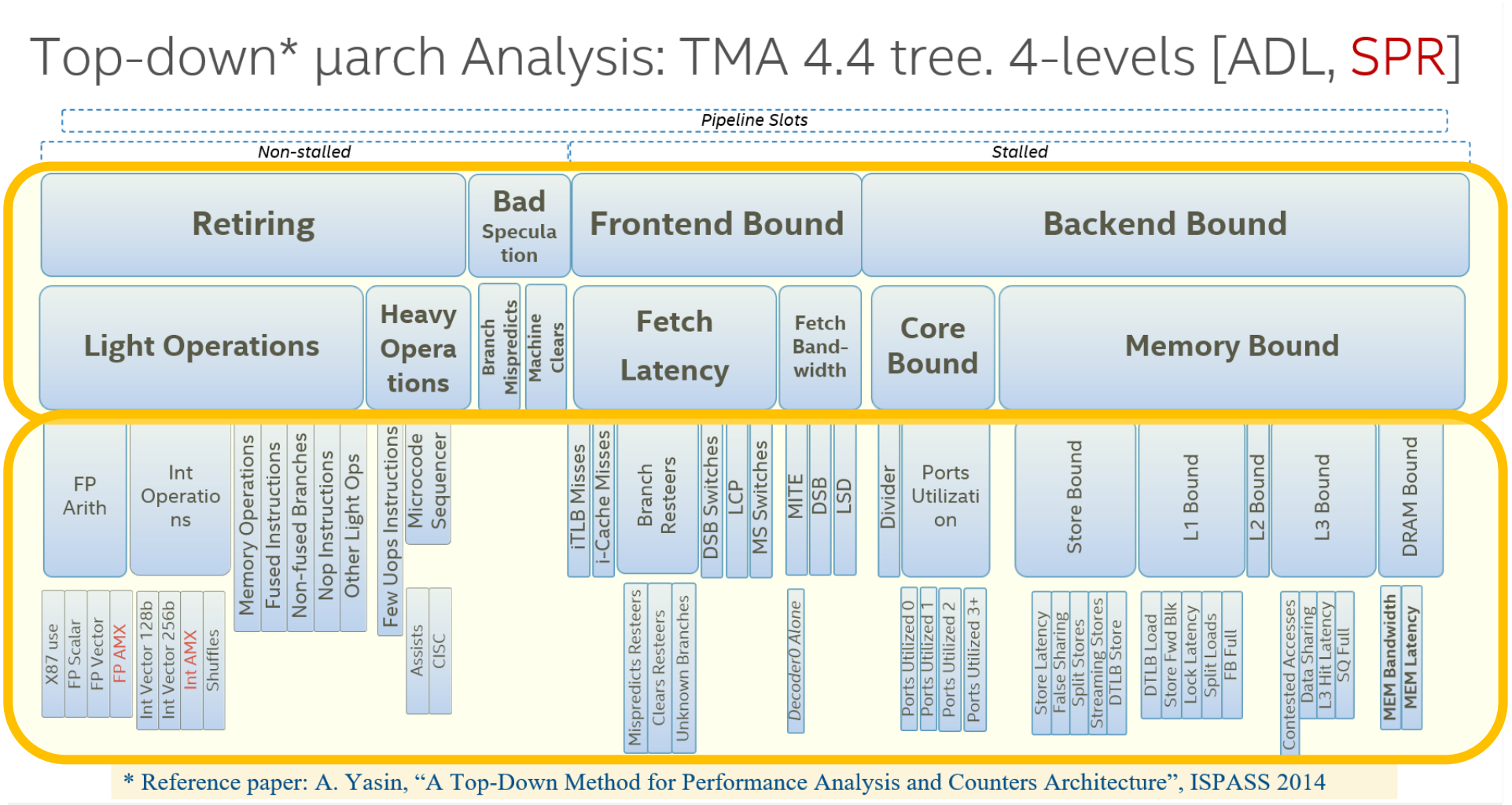


* Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

MostlyConverged;
Sum to 100

Count domain: slots

The complete TMA "tree"



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

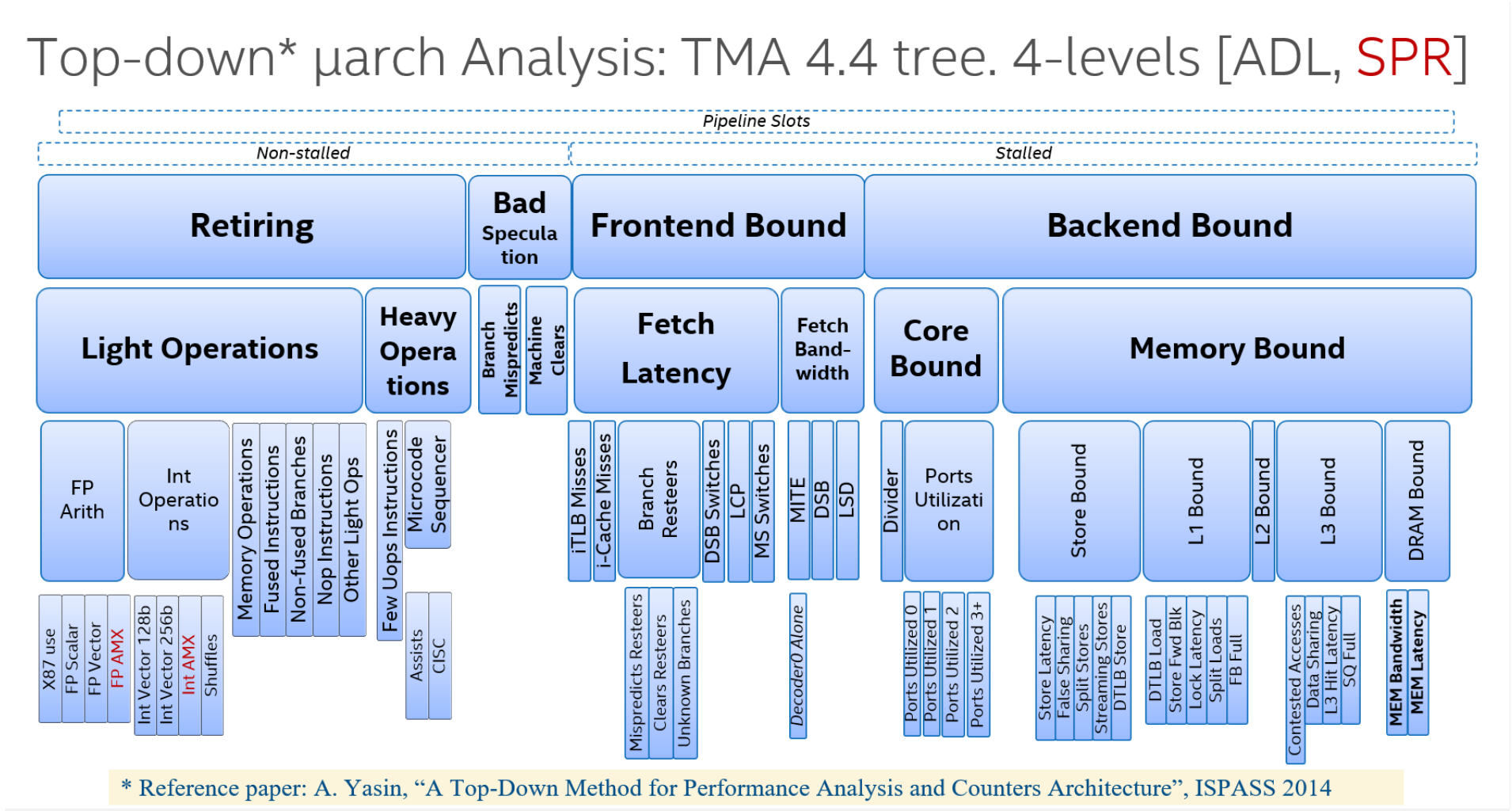
MostlyConverged; Sum to 100

Count domain: slots

More uArch-Specific, "leaf" nodes that don't necessarily sum to 100

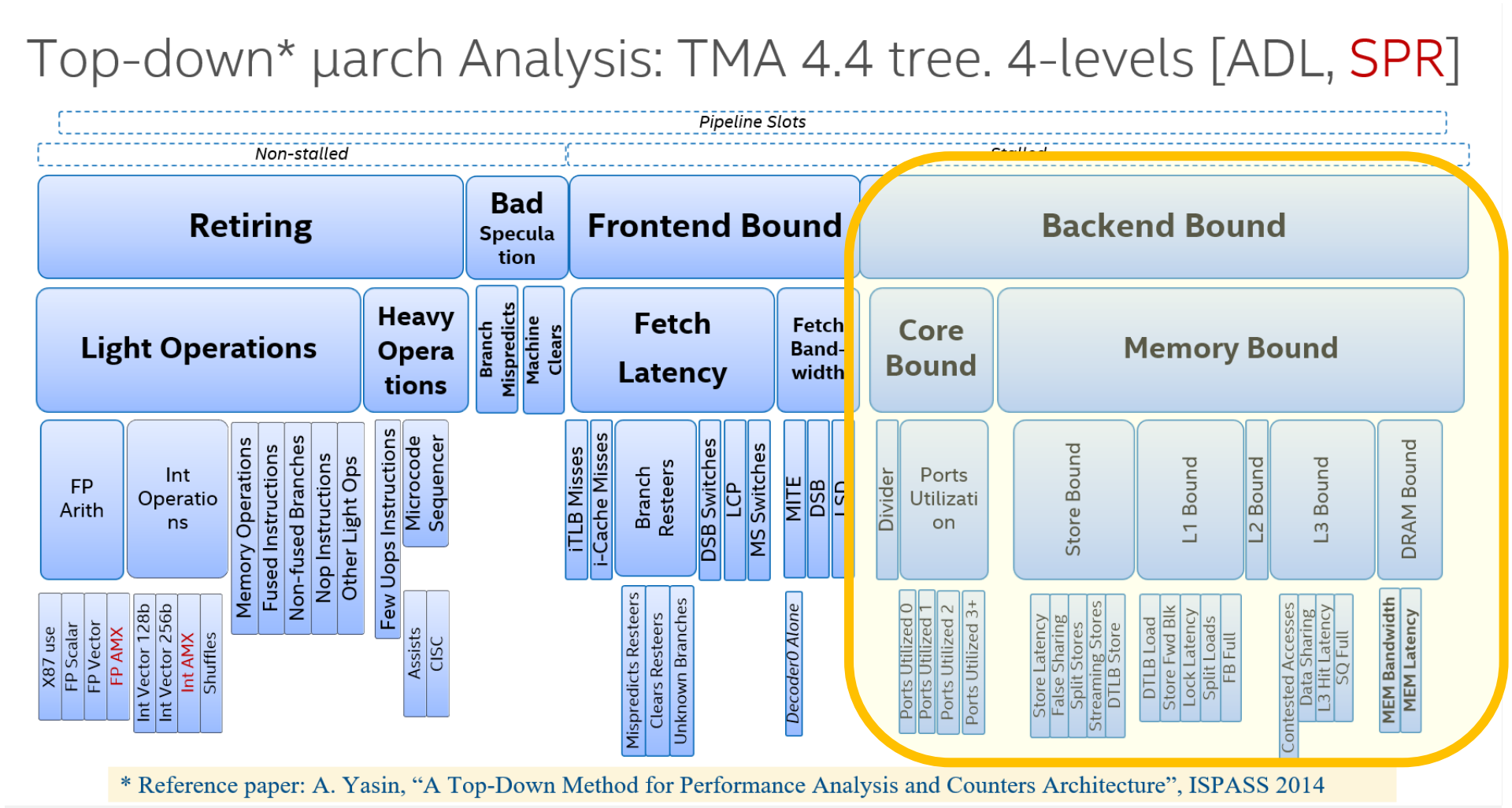
Count domain: clock ticks

The complete TMA "tree" – now focusing on Memory Boundness

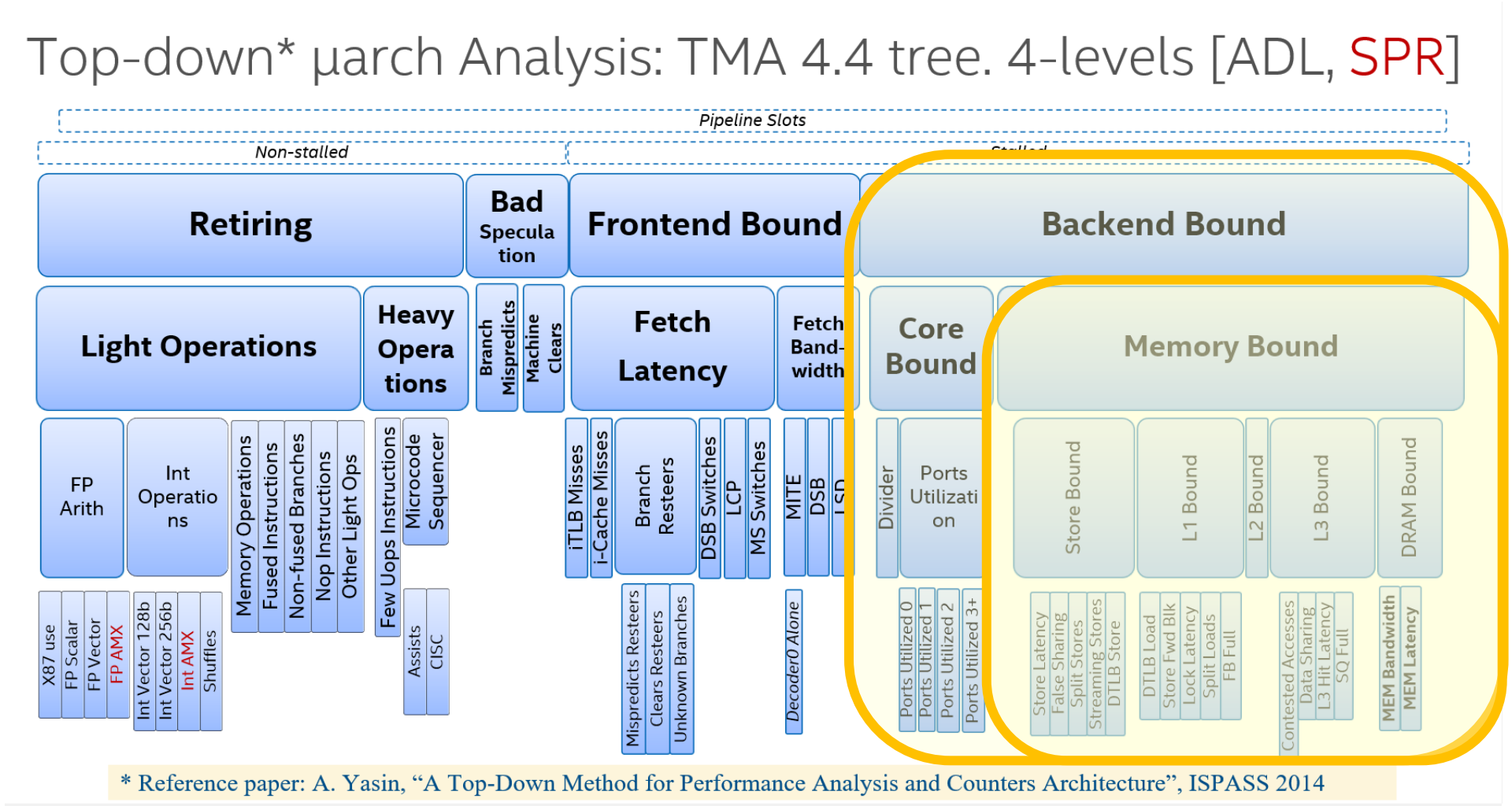


* Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

The complete TMA "tree" – now focusing on Memory Boundness

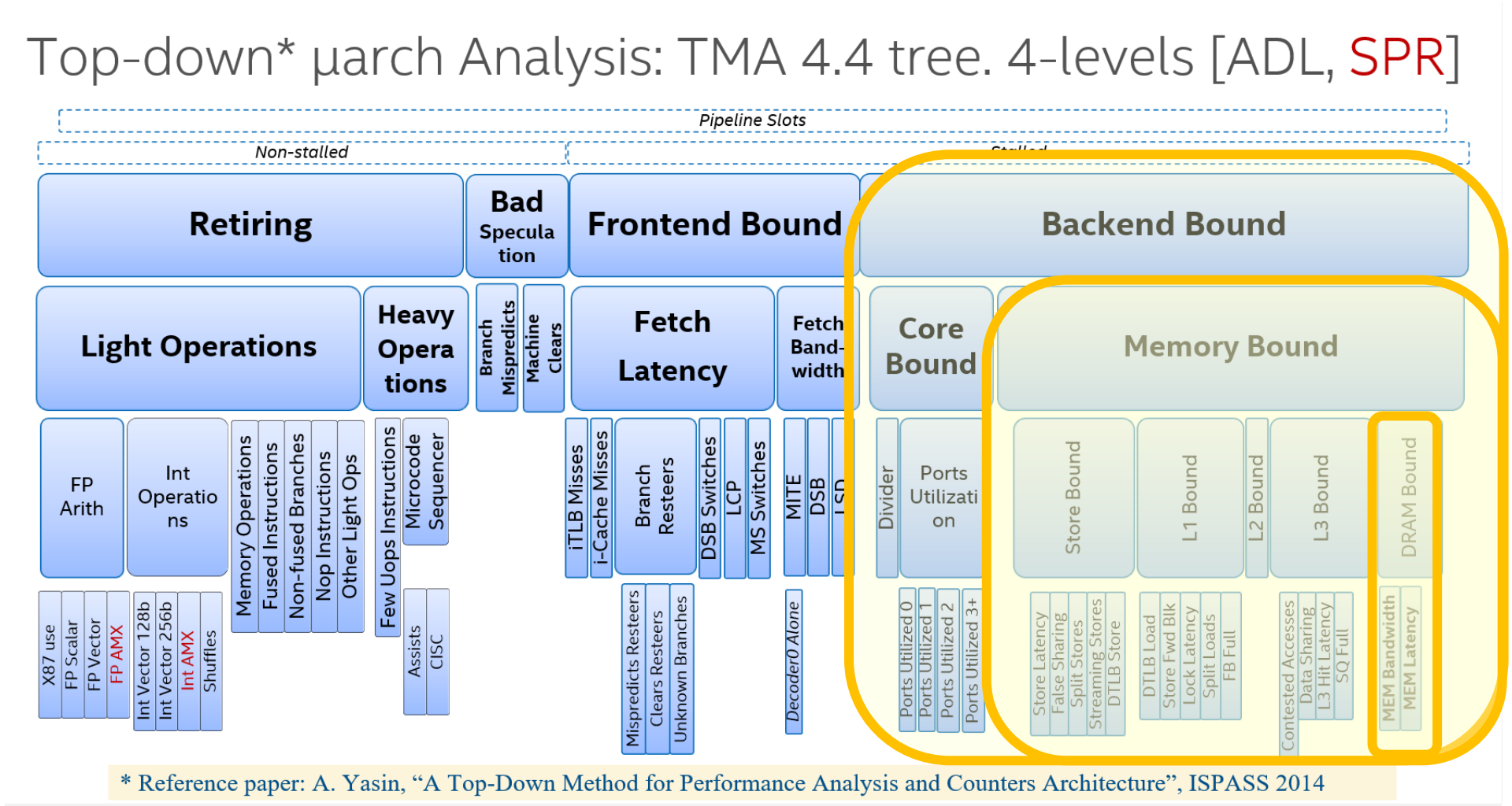


The complete TMA "tree" – now focusing on Memory Boundness



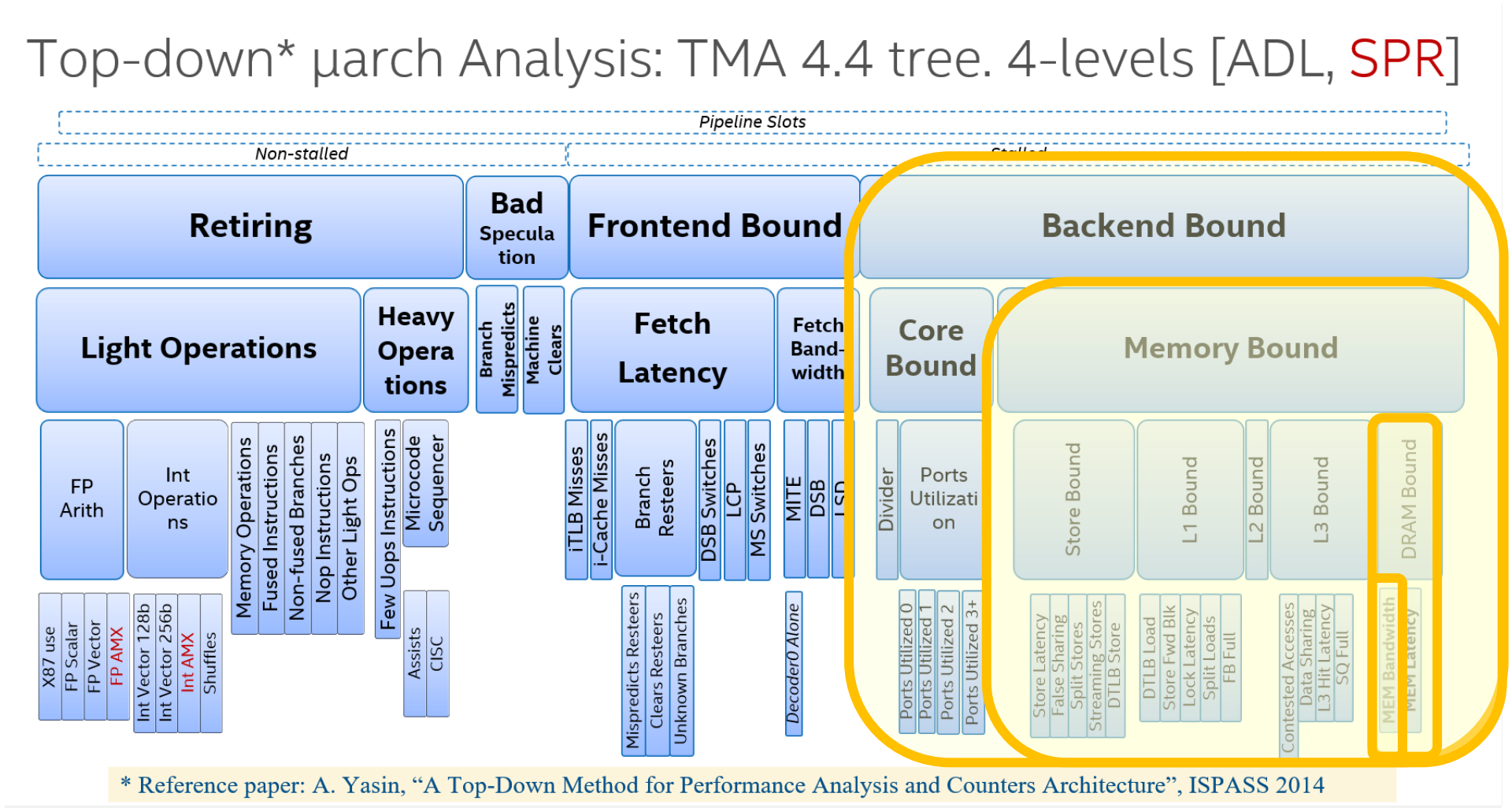
* Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

The complete TMA "tree" – now focusing on Memory Boundness



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

The complete TMA "tree" – now focusing on Memory Boundness



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

Workload characterization

Top Down L1 applied to real workloads

TMA Category	Retiring	Bad Speculation	Frontend Bound	Backend Bound
What to expect	Ideally the highest percentage category; ~25-50%	Ideally below 5%; can be up to 10%	Client: <5% Server: <20%	25-50%
What kinds of workloads will have high % in this category	Small data footprint workloads with good locality, optimized binaries	Poor usage of compiler optimizations, interpreted workloads, many small tight loops	Web-, containerized or virtualized workloads, large code footprints	Large data footprint, poor locality, I/O

Insights on Retiring

What it tells us about the Software:
- CPU resources are being used efficiently
- Optimization potential may exist: Vectorization, Power Reduction (for OS)

What it tells us about the Software:
- Optimization potential may exist: Find source of instructions triggering microcode assist

What it tells us about the Hardware:
- CPU resources are being used efficiently
- Consider hardware optimizations favoring 1uop instructions

What it tells us about the Hardware:
- If seeing significant heavy ops patterns increasing across workloads, consider accelerating that instruction

Insights on Bad Speculation

What it tells us about the Software:

- Some of the CPU's resources are wasted due to mispredicted branches
- Optimization opportunities exist: Use PGO/advanced compiling techniques, reduce branches, use predicated instructions

What it tells us about the Software:

- Software causes some undesired behavior for the CPU, investigate it Machine Clears and Bad Spec meet thresholds
  - Potentially false sharing, SMC

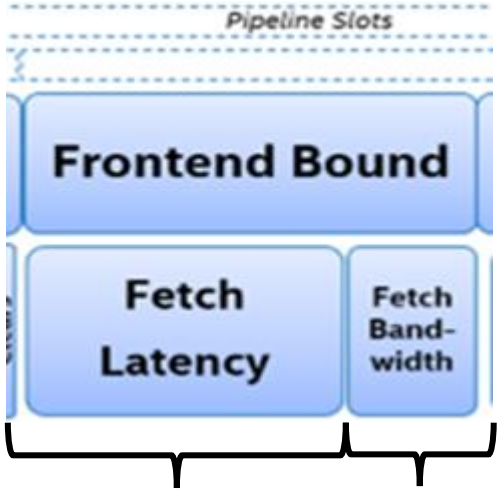
What it tells us about the Hardware:

- Potential exists to improve branch prediction

What it tells us about the Hardware:

- More actionable on the software side

Insights on Frontend Bound



What it tells us about the Software:

- Optimization opportunities exist: reduce code footprint, reduce iTLB footprint, investigate source of resteers / switches if high

What it tells us about the Software:

- Optimization opportunities exist: reduce code footprint, reduce iTLB footprint

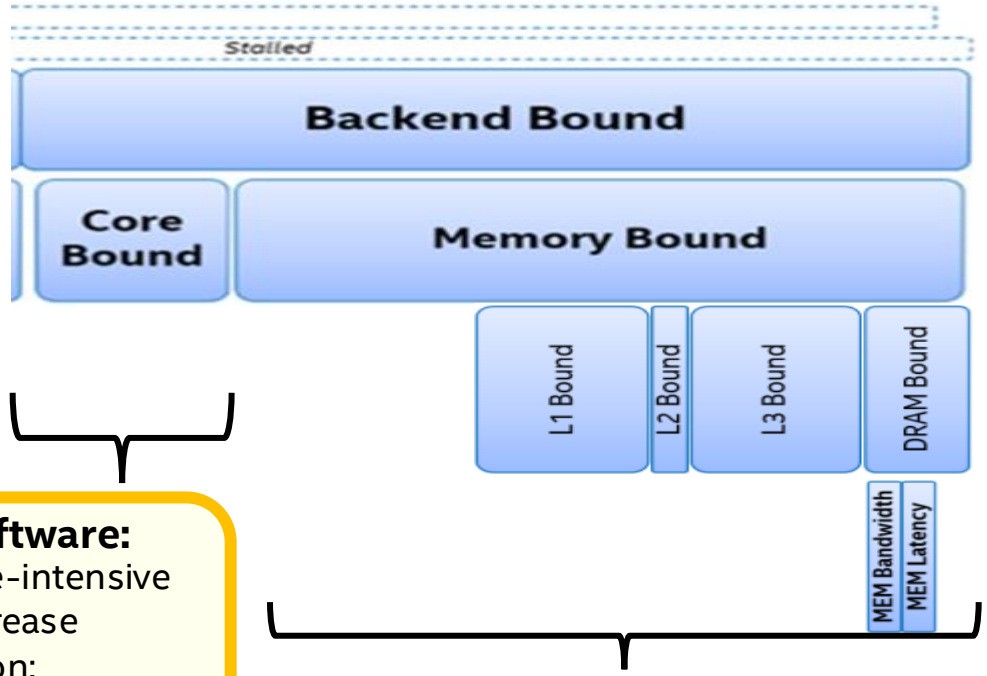
What it tells us about the Hardware:

- If increasing across workloads, consider increasing icache or iTLB sizes

What it tells us about the Hardware:

- If increasing across workloads, consider increasing icache or iTLB sizes
- Consider improvements in decode/uop pipelines

Insights on Backend Bound



What it tells us about the Software:

- If high, software is compute-intensive
- Optimization potential: increase parallelism and vectorization; investigate leaf issues if high

What it tells us about the Hardware:

- If increasing across workloads, consider adding execution units/ports

What it tells us about the Software:

- Use L3 to determine the memory bottleneck, try to reduce code or data footprint/spatial locality/temporal locality accordingly

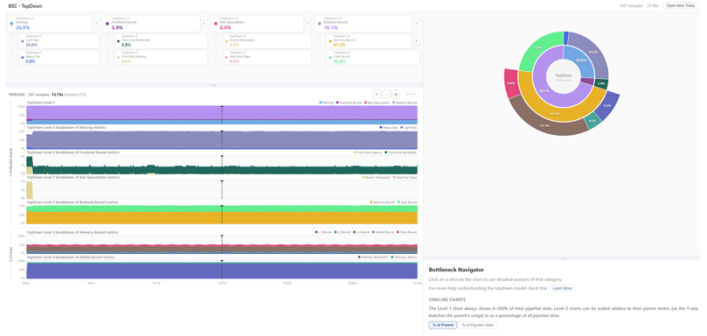
What it tells us about the Hardware:

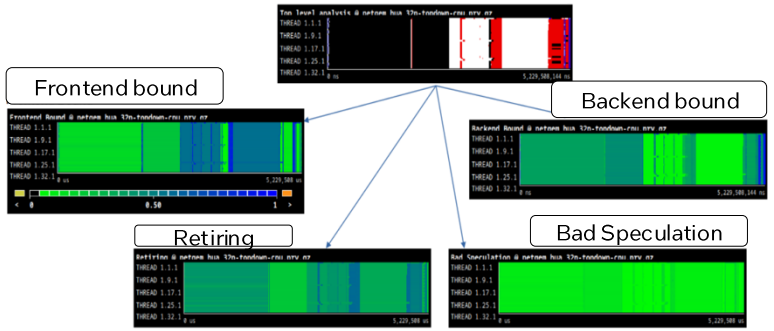
- Use L3 to determine the memory bottleneck, consider increasing cache sizes
- If workload is DRAM bound, use L4 to understand if workload would scale with more memory/more channels or would be better to have low latency

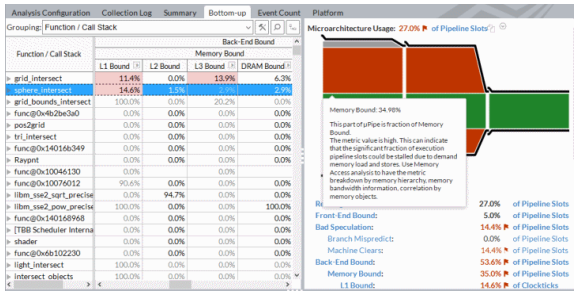
Good, I buy it. How do I use TopDown?

TMA in MN5?

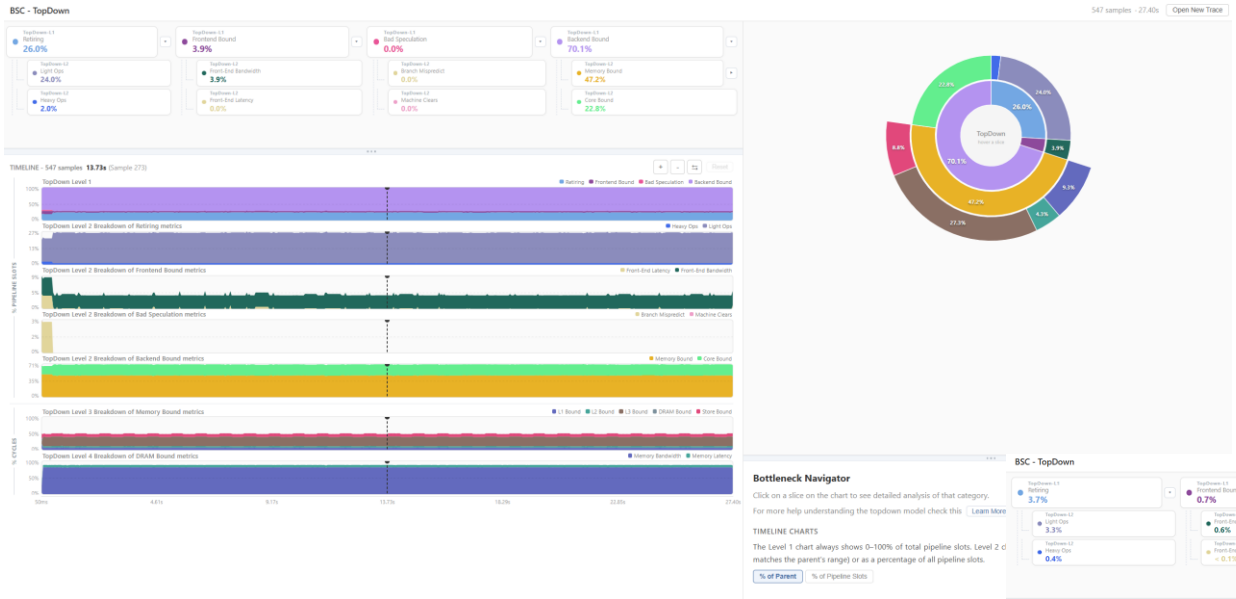
- BSC / Performance tools
  - BSC – TopDown Visualizer
  - Extrae/Paraver (WIP on MN5, already available for other architectures)
- Intel® Vtune™ Profiler
- Linux/perf

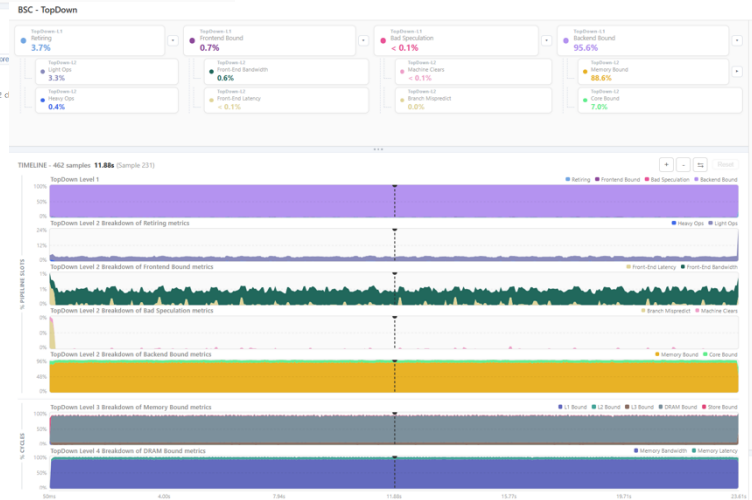


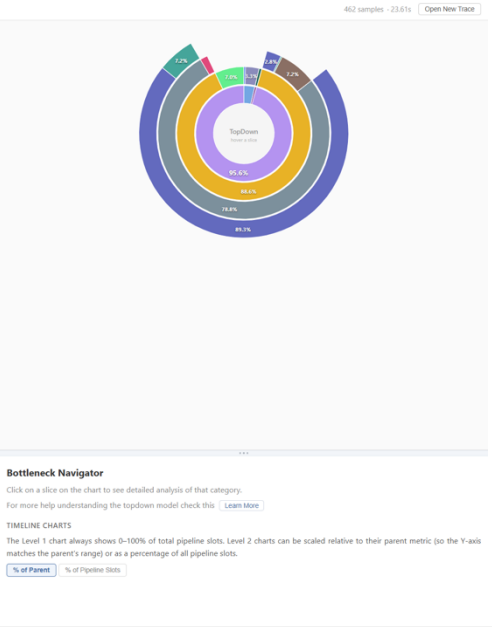




Demo / Hands-on session on BSC tools

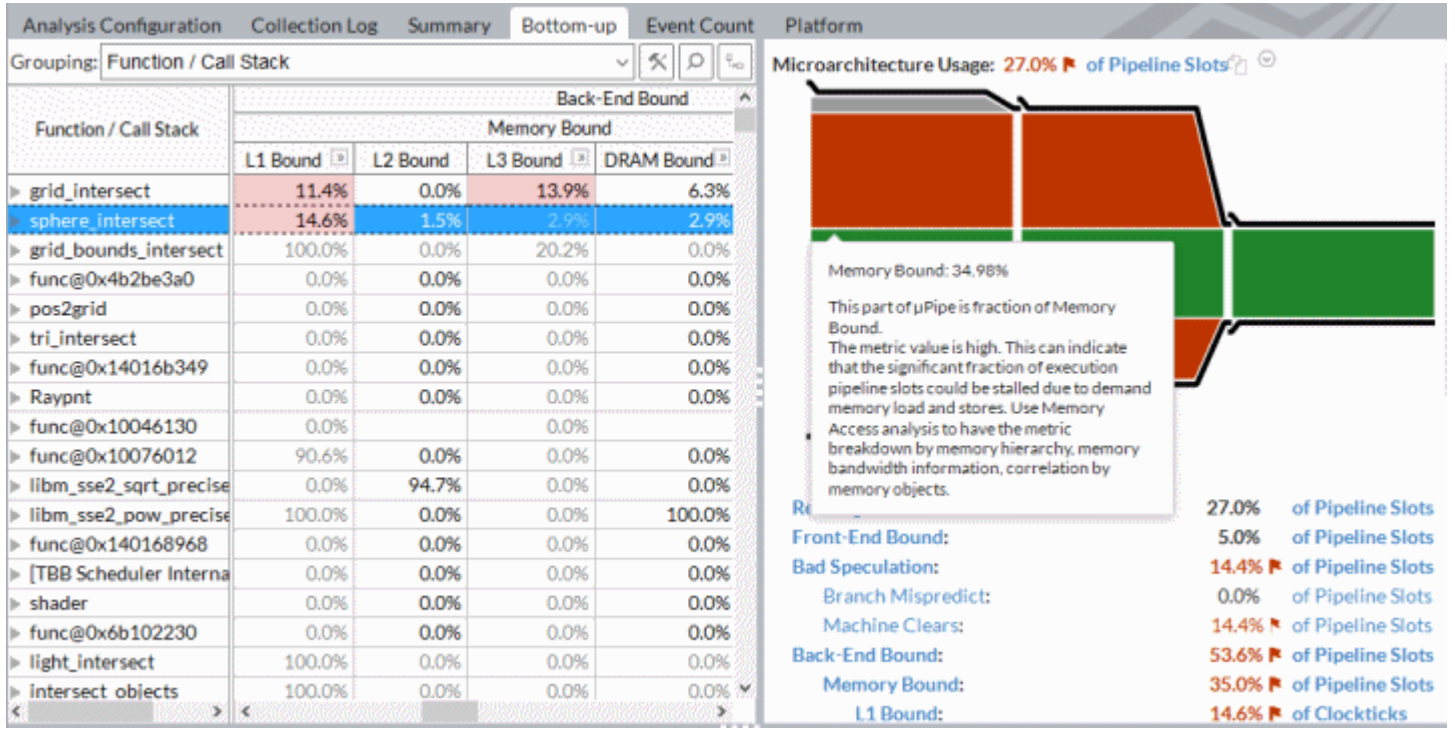






Also available in Intel tools

- Top Down Methodology is also included in Intel® Vtune™ Profiler
  - https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2025-4/top-down-microarchitecture-analysis-method.html



Software optimization examples

- Netflix VTune Case Study
  - https://netflixtechblog.com/seeing-through-hardware-counters-a-journey-to-threefold-performance-increase-2721924a2822
  - Vadim Filanovsky and Harshad Sane
- PyTorch Case Study
  - https://pytorch-cn.com/tutorials/intermediate/torchserve_with_ipex_2
  - Ming Jean Cho, Jing Xu, Mark Saroufim

Pointers, contacts, documents and other architectures

- TMA metrics v5.1 (https://github.com/intel/perfmon/blob/main/TMA_Metrics-full.xlsx)
- BSC – TopDown Visualizer (mess@bsc.es)
- Extrae/Paraver (tools@bsc.es)
- Intel® Vtune™ Profiler (https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler-download.html)
- perf-tools (https://github.com/aayasin/perf-tools)
- toplev (https://github.com/andikleen/pmu-tools)
- perf (https://perfwiki.github.io/main/top-down-analysis/)

- Intel® 64 and IA-32 Architectures Software Developer Manuals (https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html)
- Agner Fog's optimization manuals (https://agner.org/optimize/#manuals)
- uops.info (https://uops.info/)

- Also implemented in AMD Zen4, ARM Neoverse, and at least one RISC-V vendor (Ventana Micro)

Linux 6.2 Adds AMD Zen 4 Pipeline Utilization Data To Help Find Performance Bottlenecks

Written by Michael Larabel in AMD on 22 December 2022 at 01:00 PM EST. 1 Comment

Thank you!

Questions?