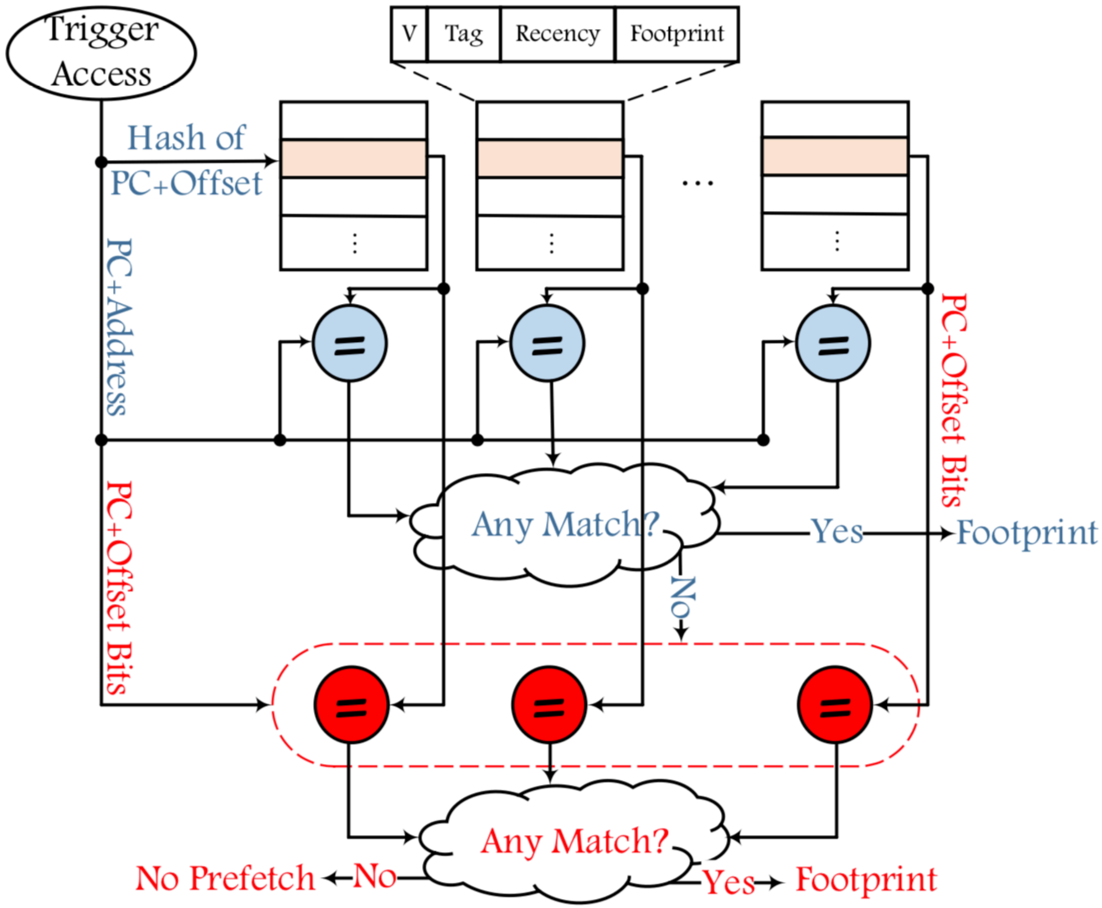
18-742:
Computer Architecture & Systems

Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

Prof. Phillip Gibbons

Spring 2025, Lecture 7

Last Lecture: Data Prefetcher



This Lecture

- **Prefetching “hard-to-predict” memory references**
  - Helper thread prefetching
  - Runahead execution

Helper Thread Prefetching [1999]

- Use <u>idle</u> threads to prefetch data for the main thread

The Actual Program

```
long refresh_potential
   (network_t * net) {
 node_t * node, * tmp;
 ... // some computation
 while (node != root) {
  while (node) {
   if(node->orientation
     == UP) {
    node->potential
     = node->basic_arc->cost
     + node->pred->potential;
   } else {
    node->potential
     = node->pred->potential
     - node->basic_arc->cost;
    checksum++;
   }
   tmp = node;
   node = node->child;
  }
  node = tmp;
  while (node->pred) {
   tmp = node->sibling;
   if(tmp) {
    node = tmp;
    break;
   } else
    node = node->pred;
  }
 }
}
```

(a)

The "Constructed" Program

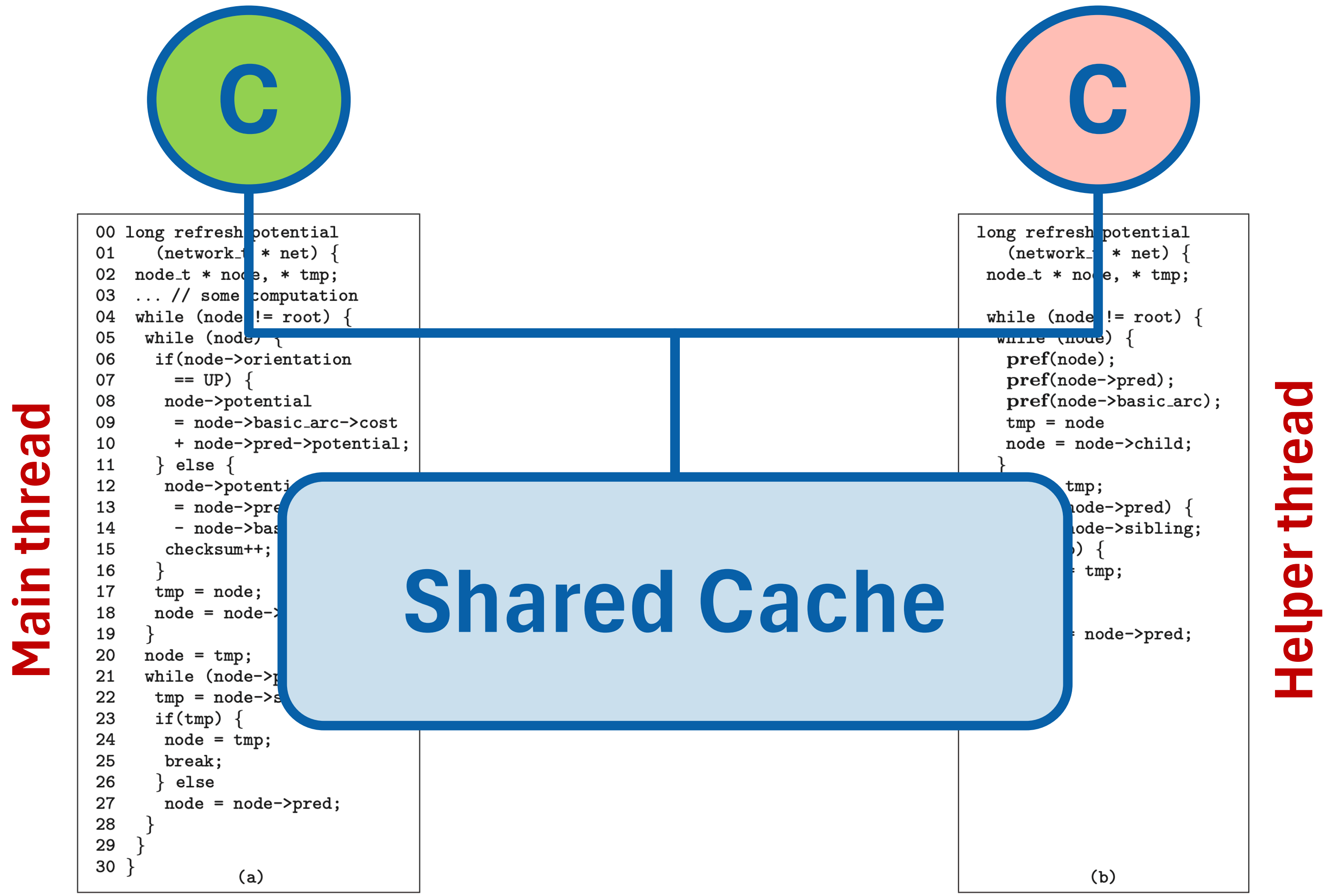
```
long refresh_potential
   (network_t * net) {
 node_t * node, * tmp;

 while (node != root) {
  while (node) {
   pref(node);
   pref(node->pred);
   pref(node->basic_arc);
   tmp = node
   node = node->child;
  }
  node = tmp;
  while (node->pred) {
   tmp = node->sibling;
   if (tmp) {
    node = tmp;
    break;
   } else
    node = node->pred;
  }
 }
}
```

(b)

Helper Thread Prefetching

- Use idle threads to prefetch data for the main thread



Helper Thread Prefetching

- **Use <u>idle</u> threads to prefetch data for the main thread**

The helper thread prefetches the data of the main thread into the shared cache

- **How does Helper Thread stay ahead of Main Thread?**
  - Only waits for prefetches when result needed for its next prefetch

- **Main Drawback?**
  - An entire execution unit (core) is dedicated for prefetching

“Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors”

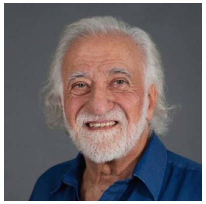
Onur Mutlu, Jared Stark, Chris Wilkerson, Yale N. Patt 2003

- **Onur:** UT Austin PhD, CMU prof, now ETH; Young Architect Award, Maurice Wilkes Award, ACM/IEEE Fellow

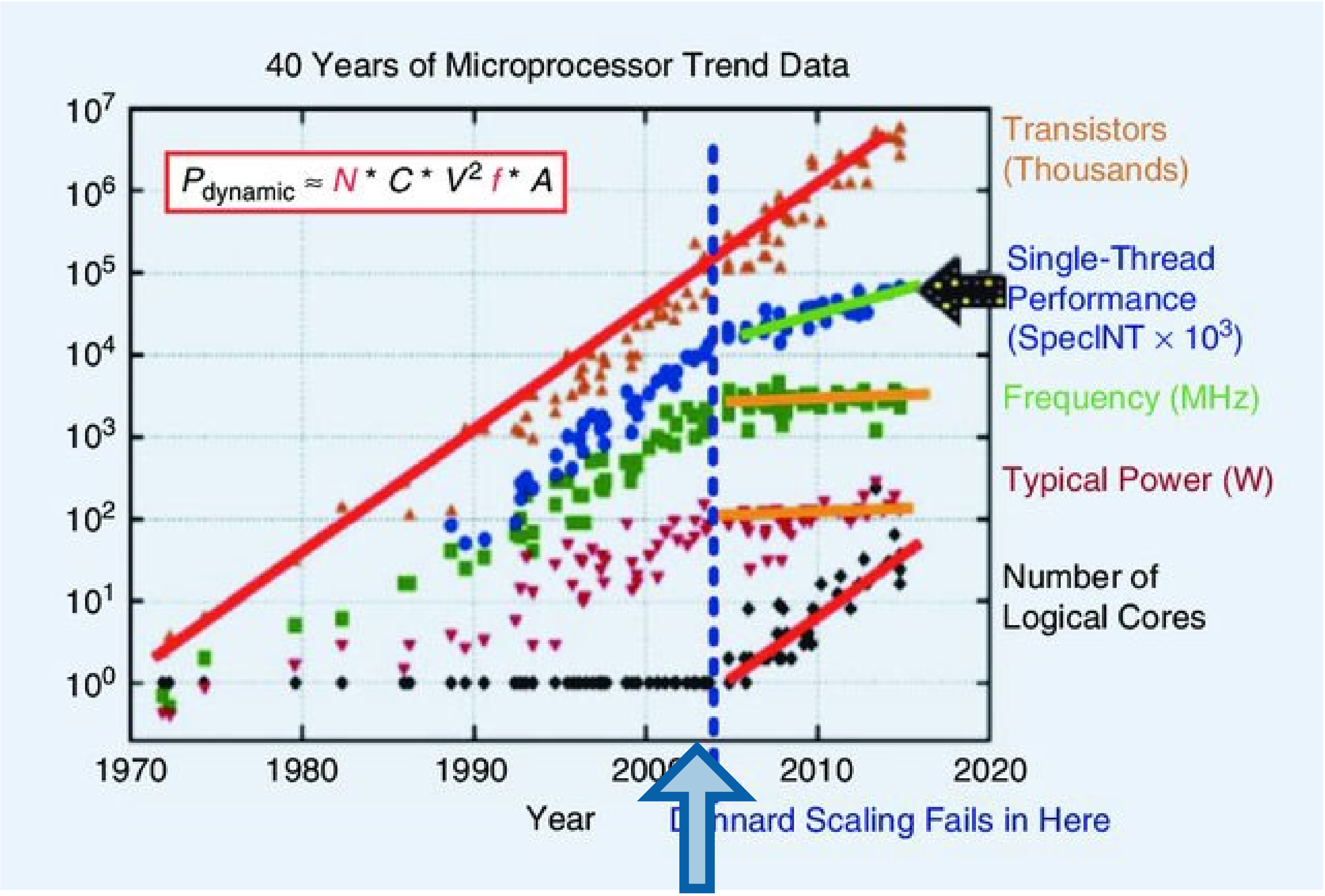
- **Jared:** Intel Processor Architect; branch predictors for Sandy Bridge and Ivy Bridge processors

- **Chris:** Intel Principal Engineer, CMU MS

- **Yale:** UT Austin Prof; NAE, ACM/IEEE Fellow, Eckert-Mauchly Award, Charles Babbage Award



Moore's Law w/o Dennard Scaling



We are here

"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"

Onur Mutlu, Jared Stark, Chris Wilkerson, Yale N. Patt 2003

- An alternative architecture for better tolerating long-latency cache misses

- Integrated into Sun ROCK, IBM POWER6, NVIDIA Denver

The slides presented hereafter are adapted from the original materials developed by Professor Onur Mutlu.

Small Windows: Full-window Stalls

8-entry instruction window:

Oldest →

LOAD R1 ← mem[R5]
BEQ R1, R0, target
ADD R2 ← R2, 8
LOAD R3 ← mem[R2]
MUL R4 ← R4, R3
ADD R4 ← R4, R5
STOR mem[R2] ← R4
ADD R2 ← R2, 64

L2 Miss! Takes 100s of cycles.

Independent of the L2 miss, executed out of program order, but cannot be retired.

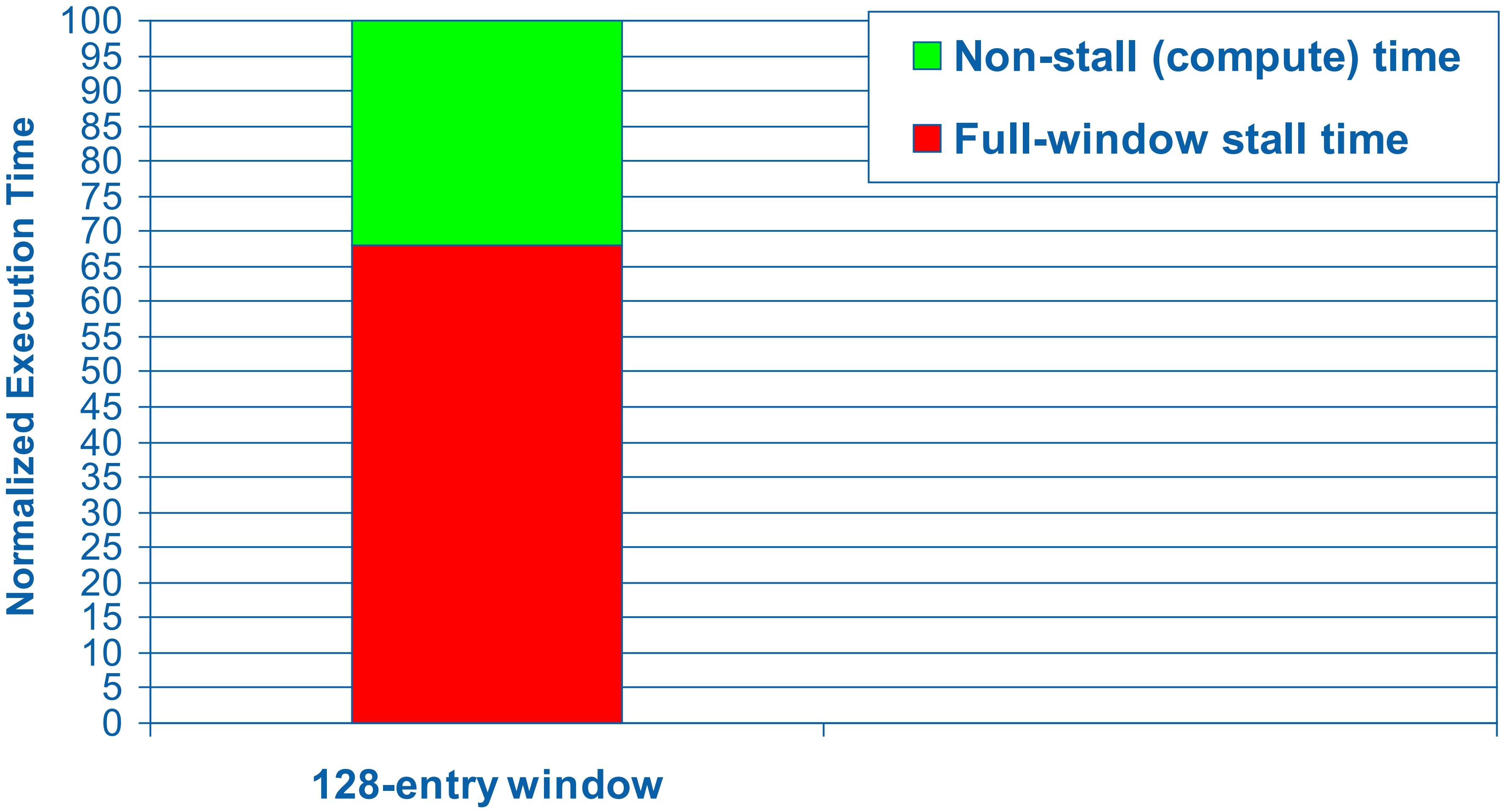
~~LOAD R3 ← mem[R2]~~

Younger instructions can't be executed because there is no space in the instruction window.

The processor stalls until the L2 Miss is serviced.

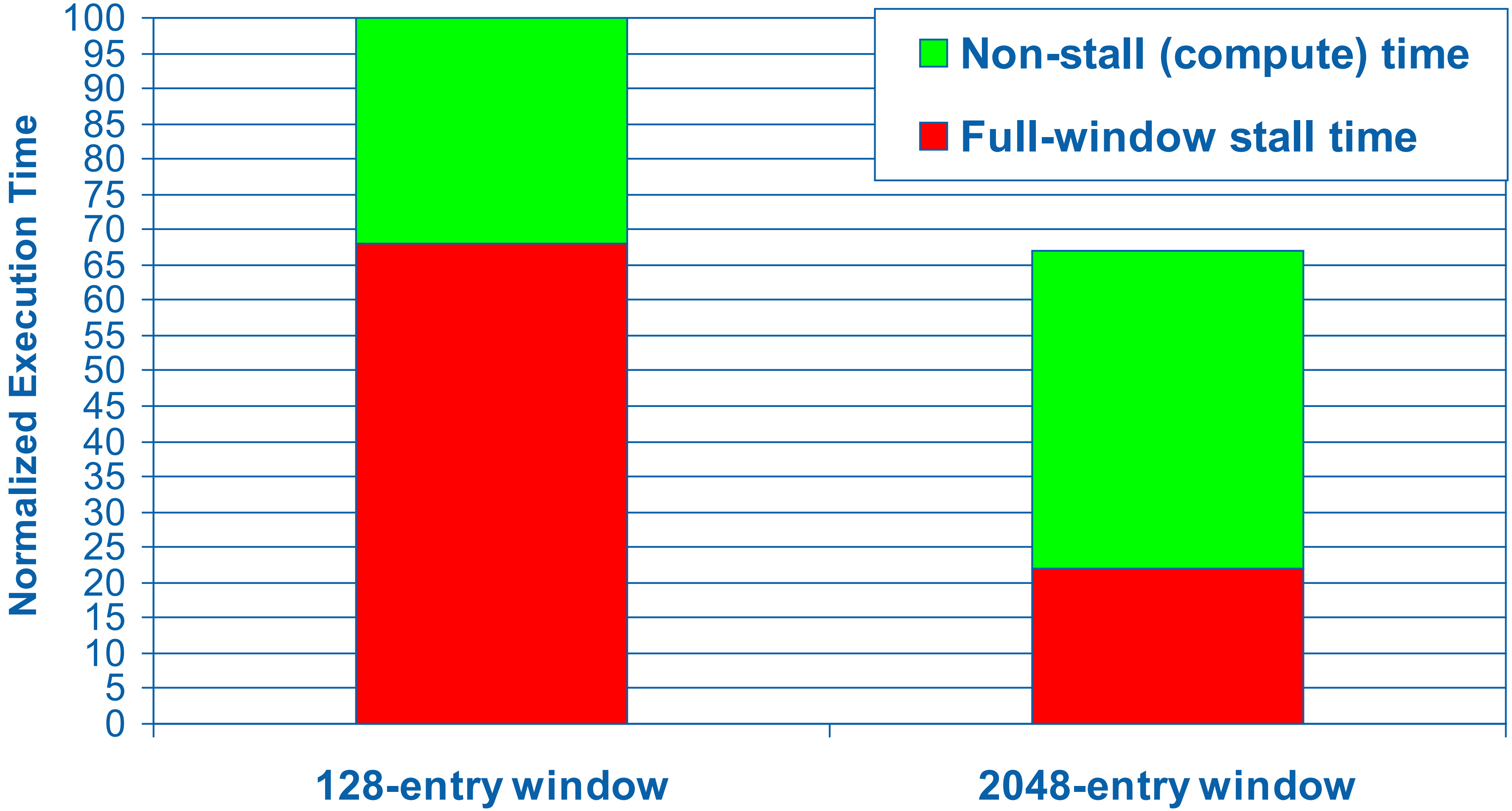
Long-latency cache misses are responsible for most full-window stall

Impact of Long-Latency Cache Misses



500-cycle DRAM latency, aggressive stream-based prefetcher
Data averaged over 147 memory-intensive benchmarks on a high-end x86 processor model

Impact of Long-Latency Cache Misses



500-cycle DRAM latency, aggressive stream-based prefetcher
Data averaged over 147 memory-intensive benchmarks on a high-end x86 processor model

The Problem

- **Out-of-order execution requires large instruction windows to tolerate today's main memory latencies.**

- **As main memory latency increases, instruction window size should also increase to fully tolerate the memory latency.**

- **Building a large instruction window is a challenging task if we would like to achieve:**
  - Low power/energy consumption (tag matching logic, ld/st buffers)
  - Short cycle time (access, wakeup/select latencies)
  - Low design and verification complexity

Efficient Scaling of Instruction Window Size

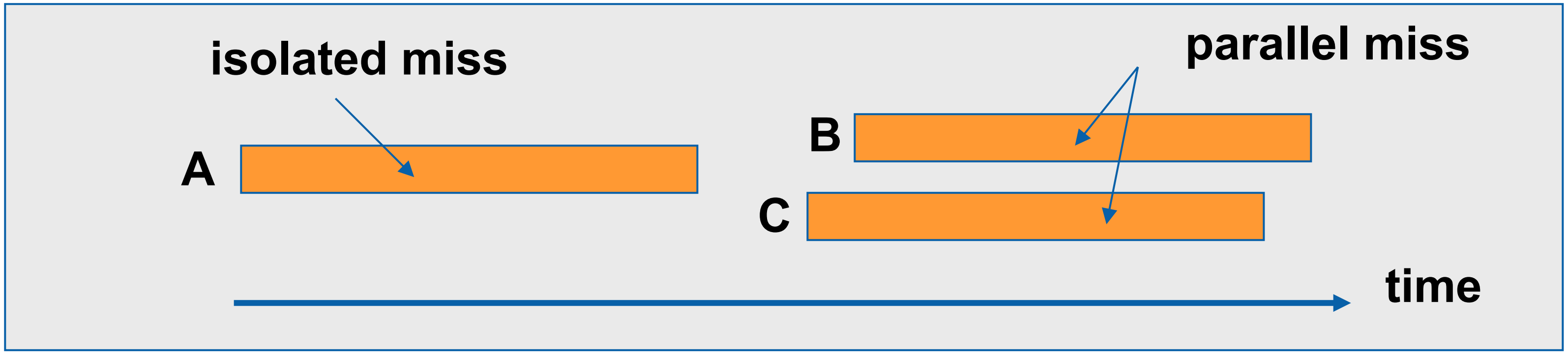
One of the major research issues in out-of-order execution

- **How to achieve the benefits of a large window with a small one (or in a simpler way)?**

- **How to efficiently tolerate memory latency using the machinery of out-of-order execution (and a small instruction window)?**

Memory Level Parallelism (MLP)

- **Idea: Find/service multiple cache misses in parallel**
  - Processor stalls only once for all misses



  - Enables latency tolerance: **overlaps latency of different misses**

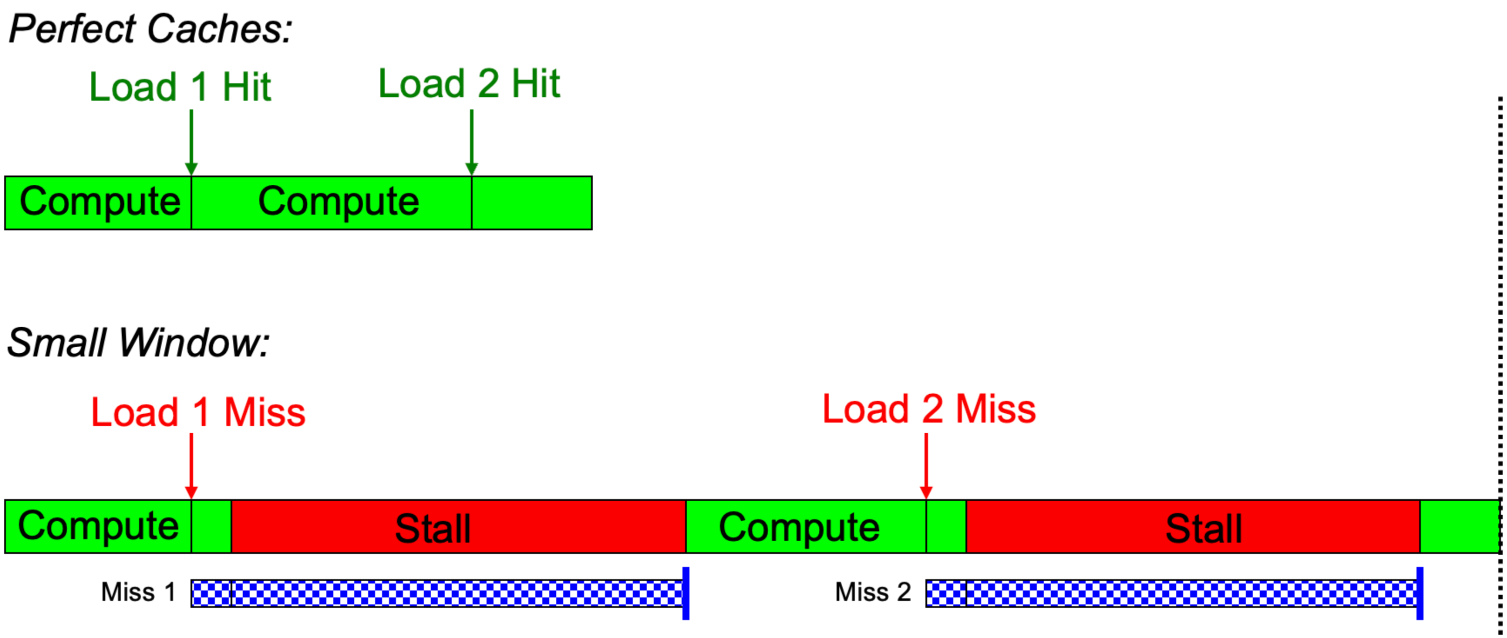
- **How to generate multiple misses?**
  - Out-of-order execution, multithreading, prefetching, runahead

Runahead Execution

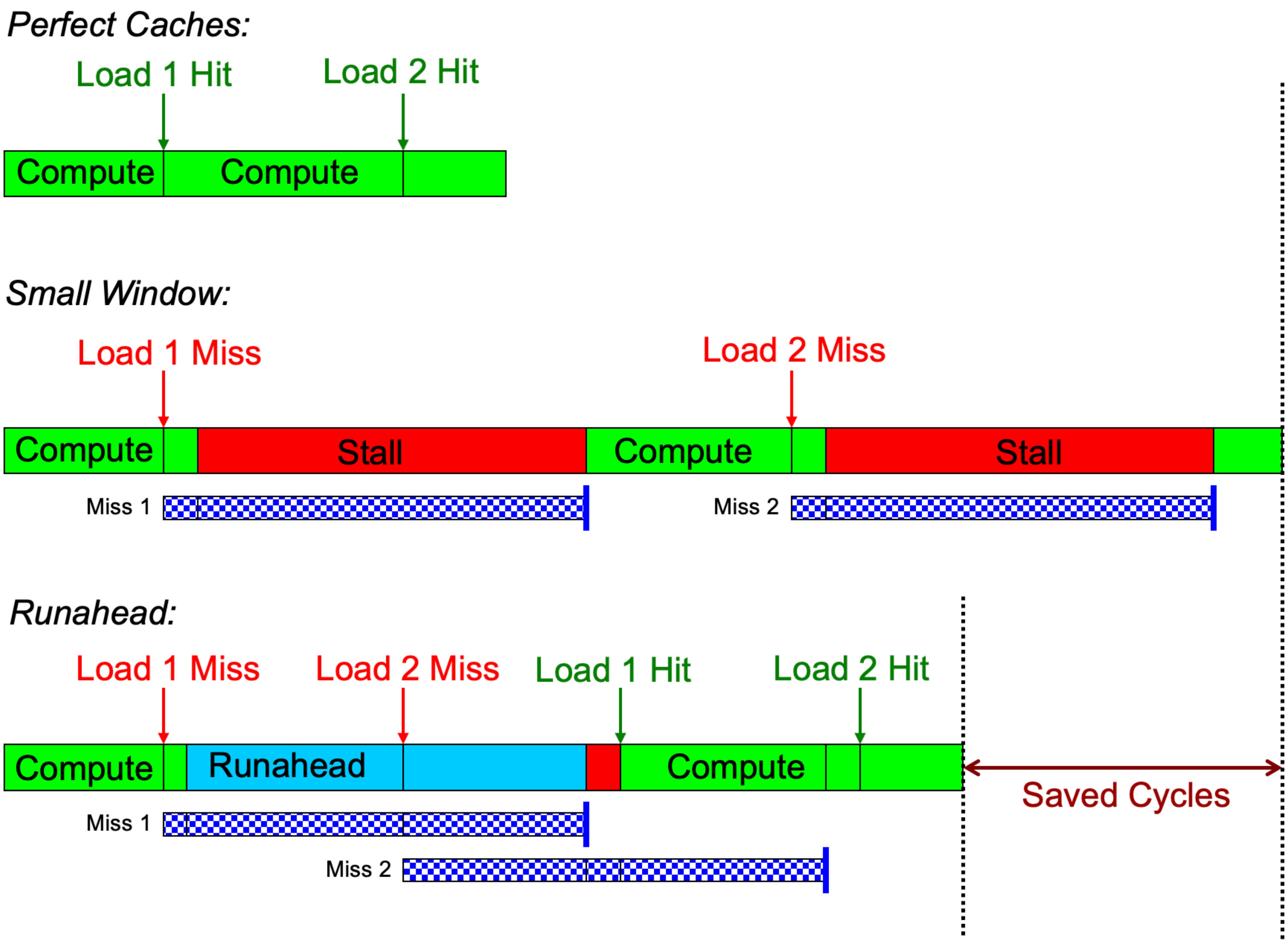
A technique to obtain the memory-level parallelism benefits of a large instruction window

- **When the oldest instruction is a long-latency cache miss:**
  - Checkpoint architectural state and enter runahead mode
- **In runahead mode:**
  - Speculatively pre-execute instructions (generates prefetches)
  - L2-miss dependent instructions are marked INV and dropped
- **Runahead mode ends when the original miss returns**
  - Checkpoint is restored and normal execution resumes

Runahead Example



Runahead Example



Discussion: Summary Question #1

What Did the Paper Get Right?

State the 3 most important things the paper says.

These could be some combination of the motivations, observations, interesting parts of the design, or clever parts of the implementation.

Benefits of Runahead Execution

Instead of stalling during an L2 cache miss:

- Pre-executed loads/stores (independent of L2-miss instructions) generate very accurate data prefetches
  - For both regular and irregular access patterns

- Instructions on the predicted program path are prefetched into the instruction/trace cache and L2.

- Hardware prefetcher and branch predictor tables are trained using future access information.

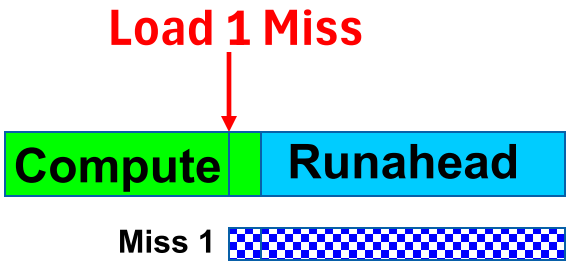
Runahead Execution Mechanism

- **Entry into runahead mode**
  - Checkpoint architectural register state

- **Instruction processing in runahead mode**

- **Exit from runahead mode**
  - Restore architectural register state from checkpoint

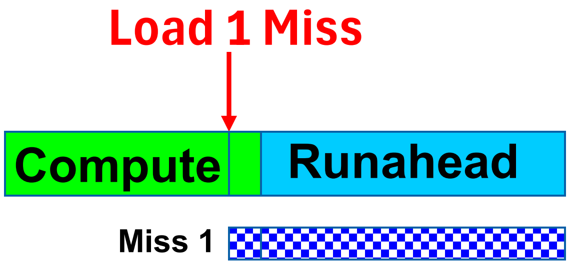
Instruction Processing in Runahead Mode



Runahead mode processing is the same as normal processing, EXCEPT:

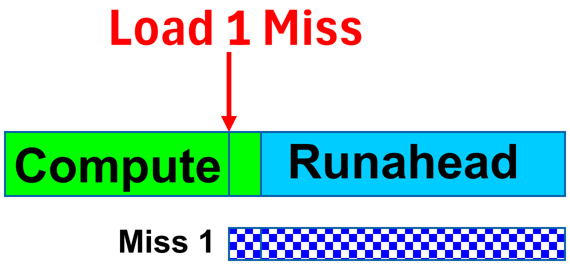
- **It is purely speculative:** Architectural (software-visible) register/memory state is NOT updated in runahead mode.
- **L2-miss dependent instructions are identified and treated specially.**
  - They are quickly removed from the instruction window.
  - Their results are not trusted.

L2-Miss Dependent Instructions



- **Two types of results produced: INV and VALID**
  - INV = Dependent on an L2 miss
- **INV results are marked using INV bits in register file & store buffer**
- **INV values are not used for prefetching/branch resolution**

Removal of Instructions from Window

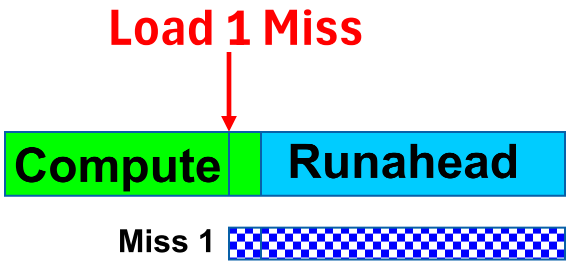


- **Oldest instruction is examined for pseudo-retirement**
  - An INV instruction is removed from window immediately.
  - A VALID instruction is removed when it completes execution.

- **Pseudo-retired instructions free their allocated resources.**
  - This allows the processing of later instructions.

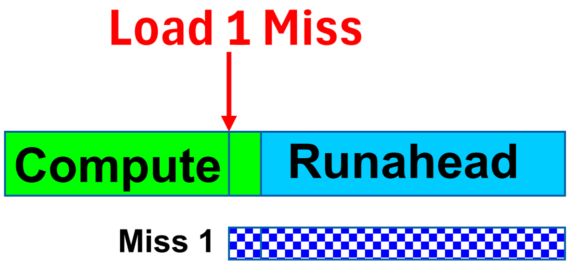
- **Pseudo-retired stores communicate their data to dependent loads.**

Store/Load Handling in Runahead Mode



- **A pseudo-retired store writes its data and INV status to a dedicated memory, called a runahead cache.**
  - ➢ Purpose: Data communication thru memory in runahead mode.
- **A dependent load reads its data from the runahead cache.**
- **Need not be always correct → Size of runahead cache is very small.**

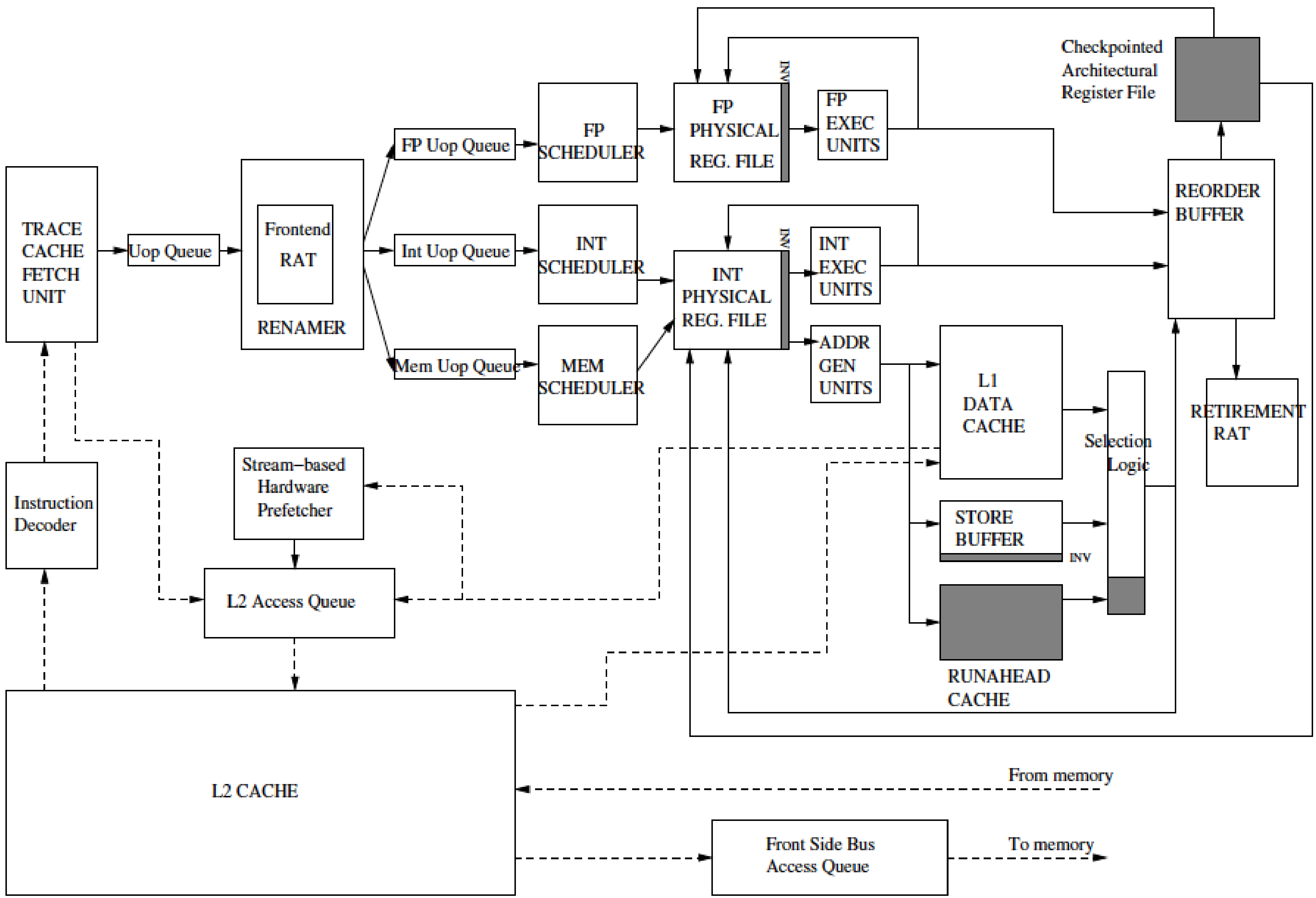
Branch Handling in Runahead Mode



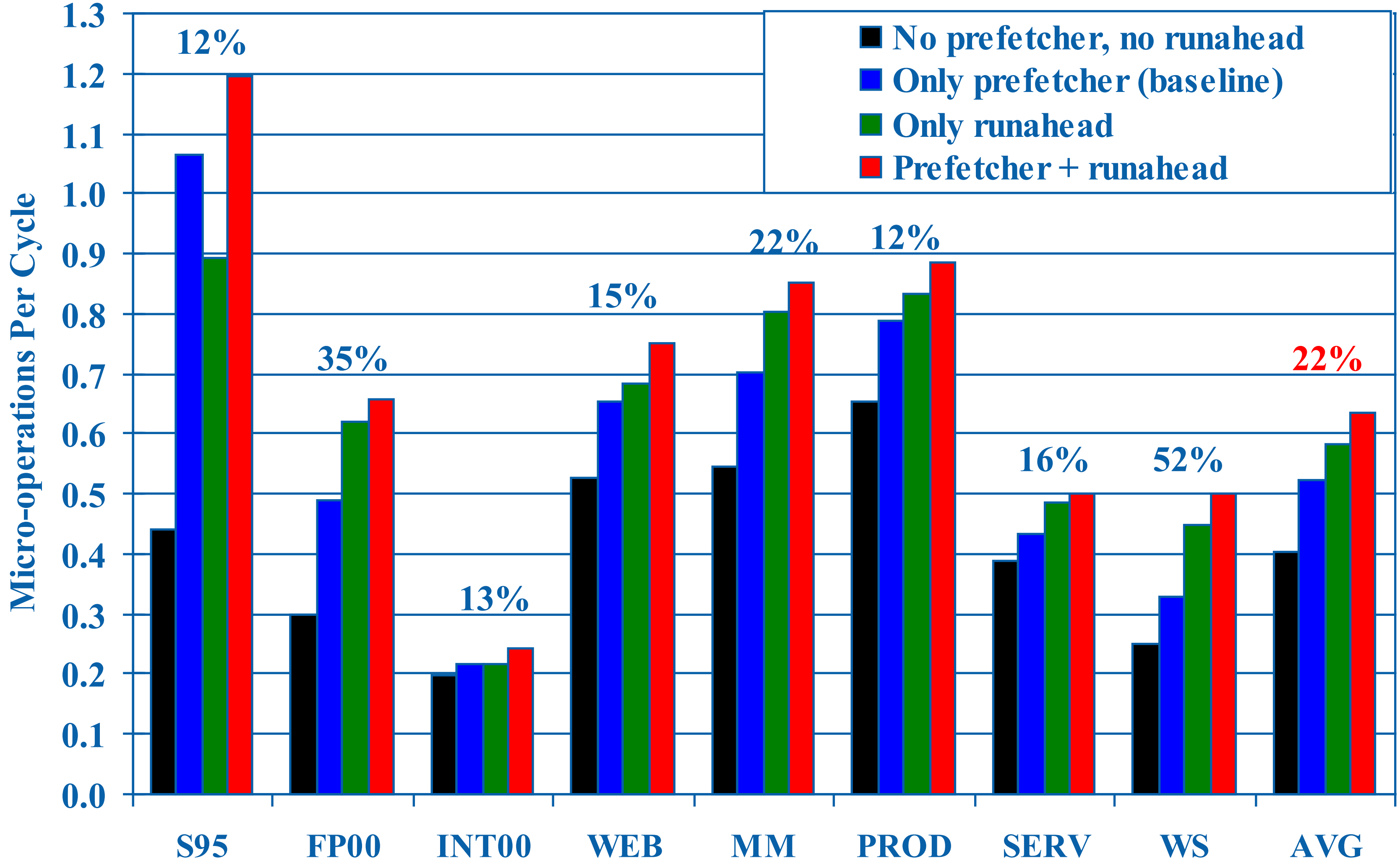
- **INV branches cannot be resolved**
  - A mispredicted INV branch causes the processor to stay on the wrong program path until the end of runahead execution.

- **VALID branches are resolved and initiate recovery if mispredicted.**

A Runahead Processor Diagram

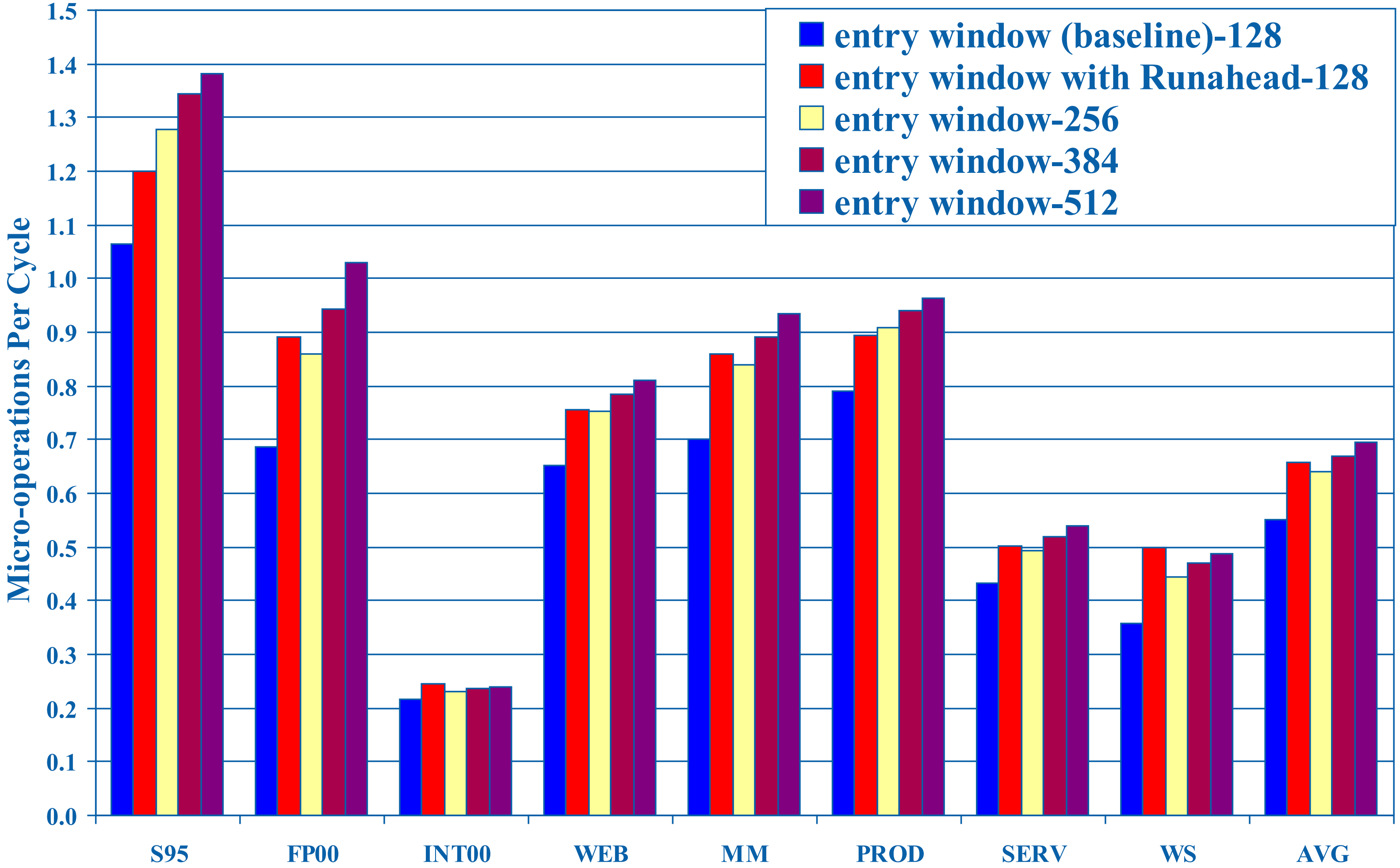


Performance of Runahead Execution



Runahead+Prefetcher is 22% better than Prefetcher alone

Runahead Execution vs. Large Windows



Runahead Window-128 is within 1% of Baseline Window-384

Discussion: Summary Question #2

What Did the Paper Get Wrong?

Describe the paper's single most glaring deficiency.

Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Runahead Execution: Pros and Cons

Advantages:

+ Very accurate prefetches for data/instructions (all cache levels)
- Follows the program path

+ Simple to implement, most of the hardware is already built in

+ Versus other pre-execution-based prefetching mechanisms:
- Uses the same thread context as main thread, no waste of context
- No need to construct a pre-execution thread

Disadvantages/Limitations:

-- Extra executed instructions

-- Limited by branch prediction accuracy

-- Cannot prefetch dependent cache misses

-- Effectiveness limited by available "memory-level parallelism" (MLP)

-- Prefetch distance (how far ahead to prefetch) limited by memory latency

Current and Future Processors

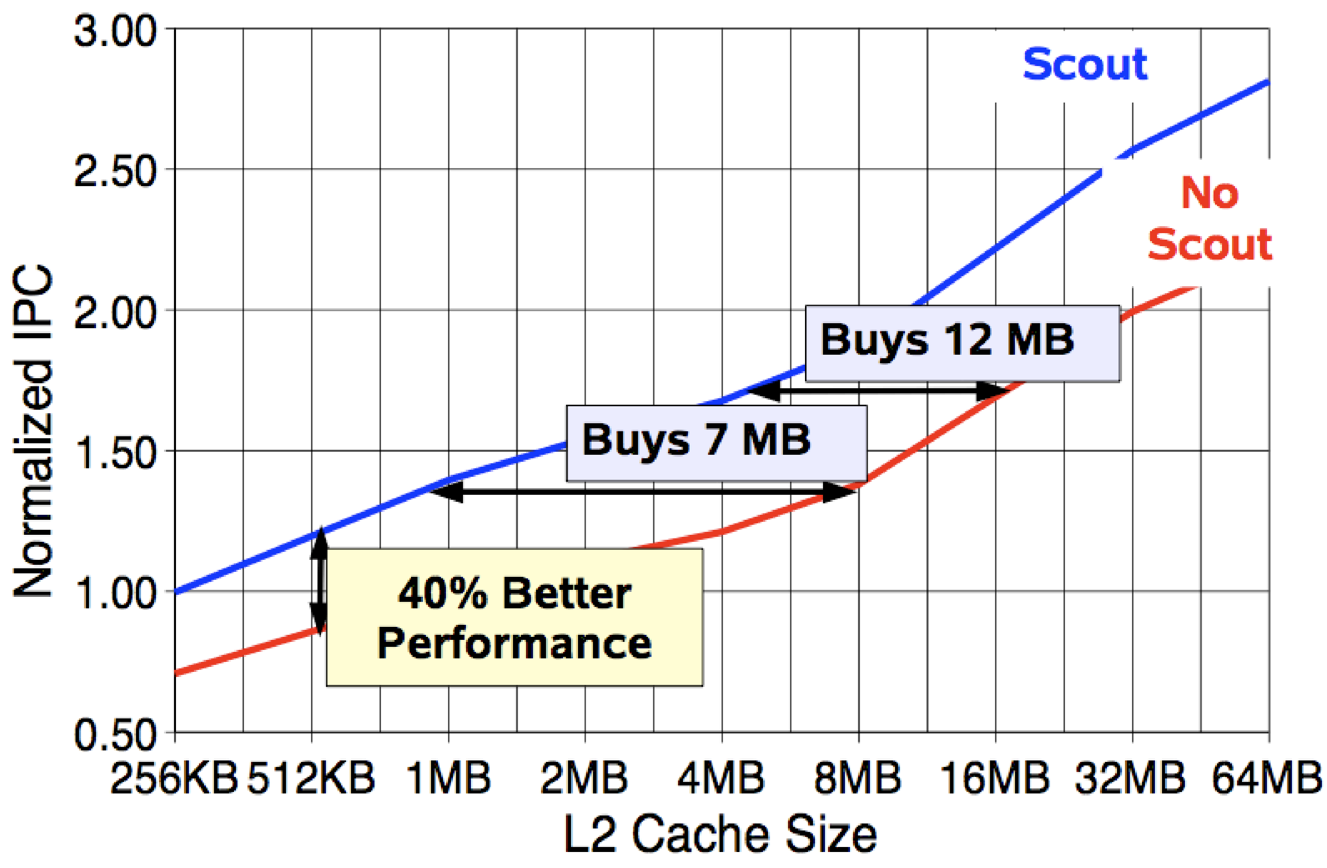
PARAMETER	CURRENT	FUTURE
Processor Frequency	4 GHz	8 GHz
Fetch/Issue/Retire Width	3	6
Branch Misprediction Penalty	29 stages	58 stages
Instruction window size	128	512
Scheduling window size	16 int, 8 mem, 24 fp	64 int, 32 mem, 96 fp
Load and store buffer sizes	48 load, 32 store	192 load, 128 store
Functional units	3 int, 2 mem, 1 fp	6 int, 4 mem, 2 fp
Branch predictor	1000-entry 32-bit history perceptron [15]	3000-entry 32-bit history perceptron
Hardware Data Prefetcher	Stream-based (16 streams)	Stream-based (16 streams)
Trace Cache	12k-uops, 8-way	64k-uops, 8-way
Memory Disambiguation	Perfect	Perfect

Memory Subsystem

L1 Data Cache	32 KB, 8-way, 64-byte line size	64 KB, 8-way, 64-byte line size
L1 Data Cache Hit Latency	3 cycles	6 cycles
L1 Data Cache Bandwidth	512 GB/s, 2 accesses/cycle	4 TB/s, 4 accesses/cycle
L2 Unified Cache	512 KB, 8-way, 64-byte line size	1 MB, 8-way, 64-byte line size
L2 Unified Cache Hit Latency	16 cycles	32 cycles
L2 Unified Cache Bandwidth	128 GB/s	256 GB/s
Bus Latency	495 processor cycles	1008 processor cycles
Bus Bandwidth	4.25 GB/s	8.5 GB/s
Max Pending Bus Transactions	10	20

Effect of Runahead in Sun ROCK

Shailender Chaudhry talk, Aug 2008



To Read for Friday

“Decoupled Vector Runahead”

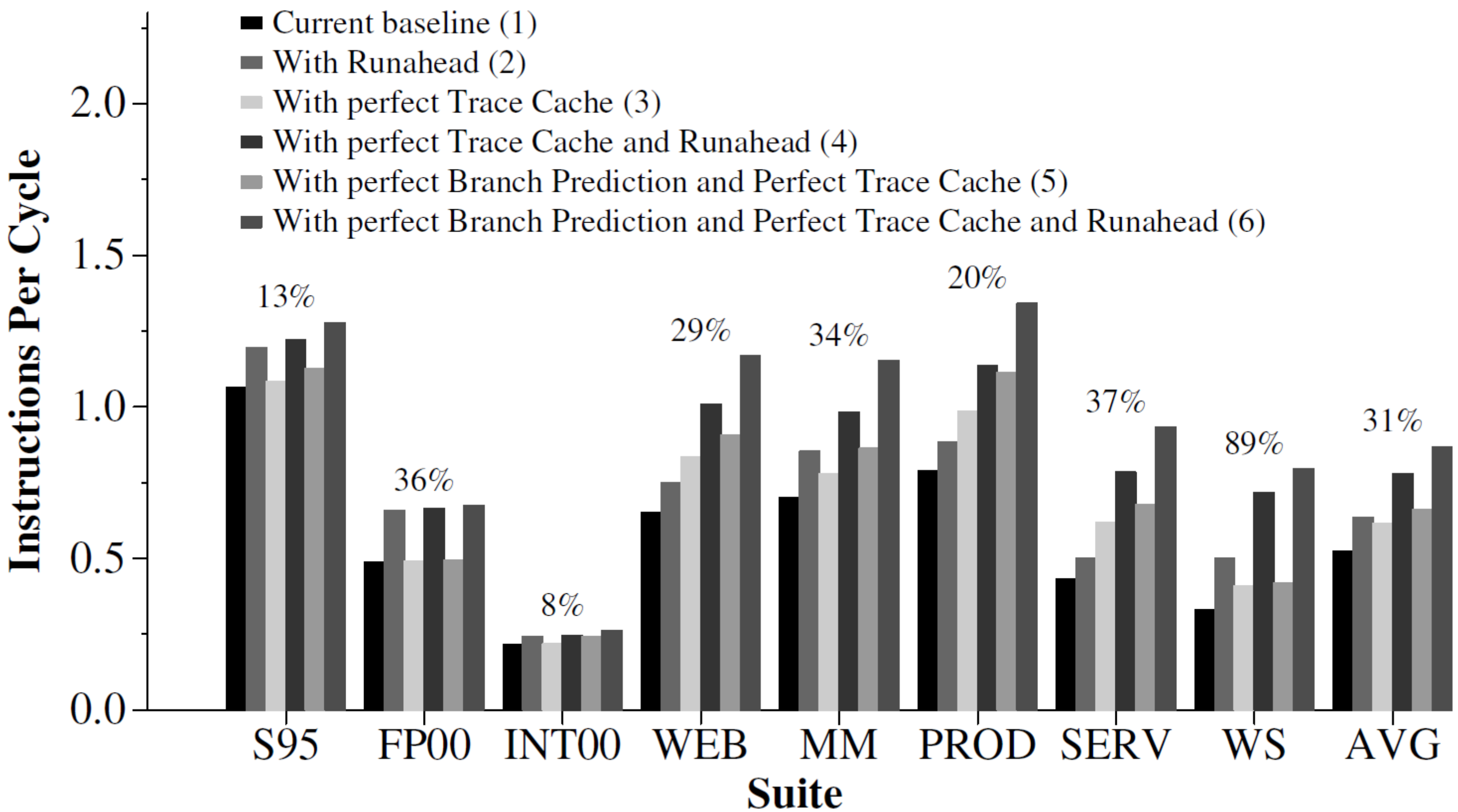
Ajeya Naithani, Jaime Roelandts, Sam Ainsworth, Timothy M. Jones, Lieven Eeckhout 2023

Optional Further Reading:

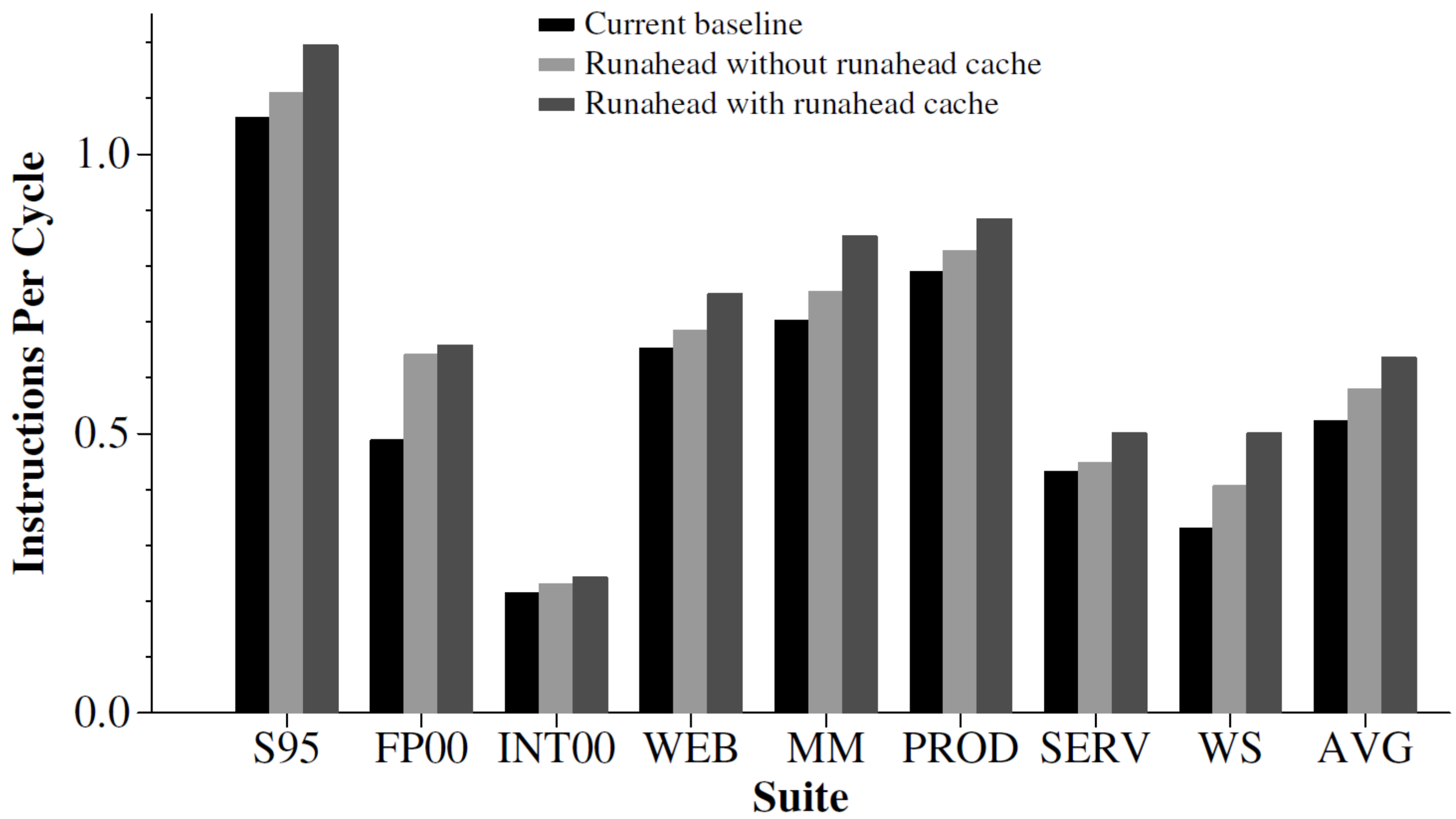
“Accelerating Dependent Cache Misses with an Enhanced Memory Controller”

Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, Yale N. Patt 2016

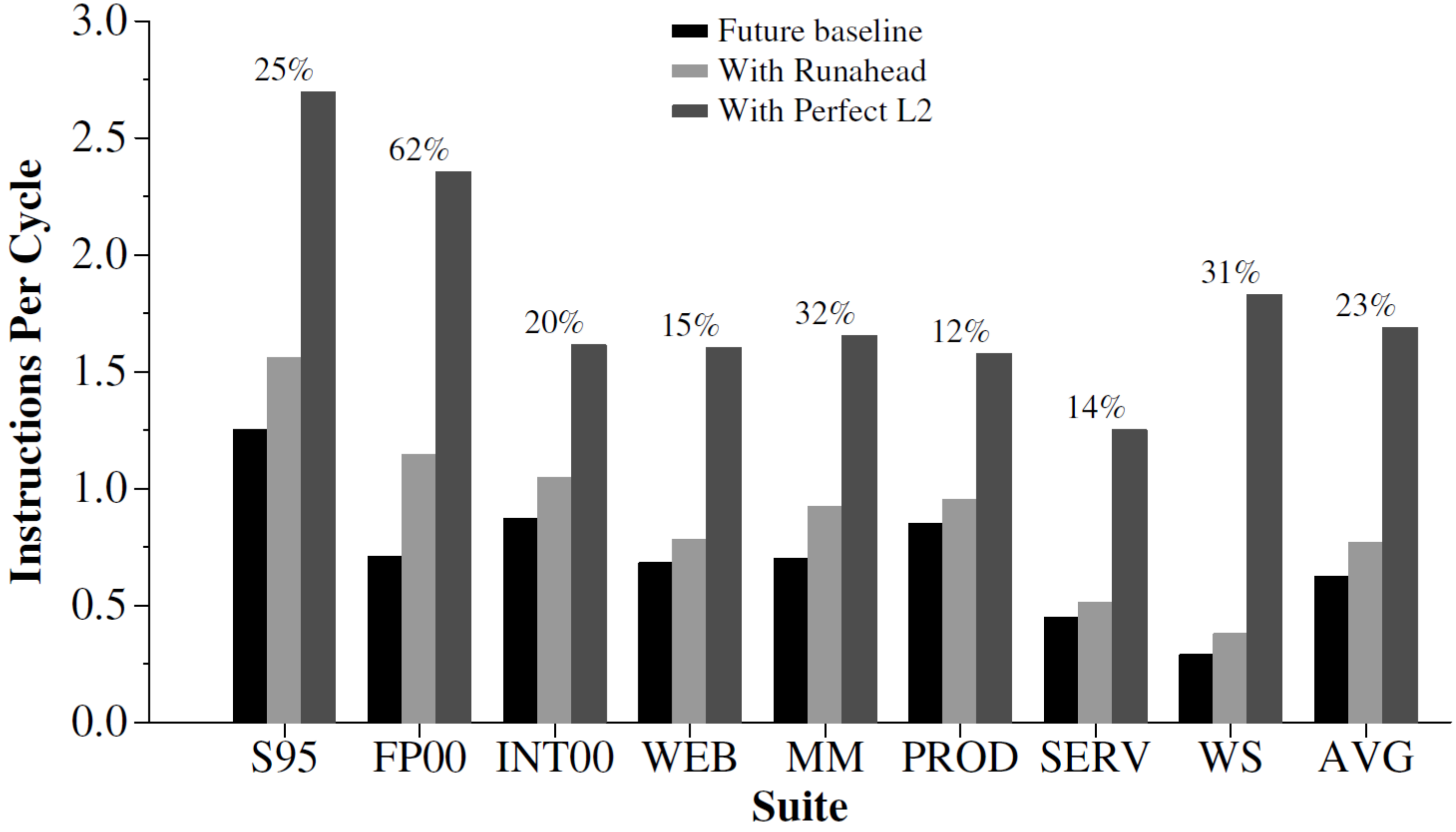
Performance on Improved Frontend



Impact of Runahead Cache



Runahead on Future Processor



Perfect Frontend on Future Processor