18-747 Lecture 13: Memory Prefetching and Speculation

James C. Hoe
Dept of ECE, CMU
October 10, 2001

Reading Assignments: (Optional MJ Ch9)

Announcements: Quiz on Monday 10/15
HW 2 due now
Project 1 due Friday before recitation

Handouts: Handout 09 HW2 Solution

Format of the Quiz

- Coverage
  - Lectures (upto L12), projects, HWs, assigned readings (textbooks and papers)
- Types of questions
  - Freebies: can you remember the materials
  - Probing: did you understand the materials
  - Design: can you apply the materials
- Open Book (course textbooks, course handouts, any materials written by yourself, **no outside materials, no electronics, no sharing, no borrowing**)
- Designed to be completed by the average student in 100 minutes
  - Someone who understands the material solidly should take substantially less time

How to Prepare

- Start early
- Review the lecture notes and reading assignments
  - Pay attention to *high-lighted* items
  - Redo all of the examples in the lecture handouts
- Do the practice exam
- Review the problem sets and projects
  - (especially if you didn't do them the first time)
- Go to office hours with questions
- Rest well the night before and show up on-time

During the Quiz

- Think before you answer. Is there an easier way to solve the problem?
- Pay attention to time allocation
  - 100 minutes, 100 points
  - *if a question is worth 5 points, don't spend 20 minutes on it*
  - What does it mean if a problem is worth [X/Y points]?
- Skip questions you can't do and come back to them later
- Don't expect to have time to read Chapter 3 for the very first time during the quiz

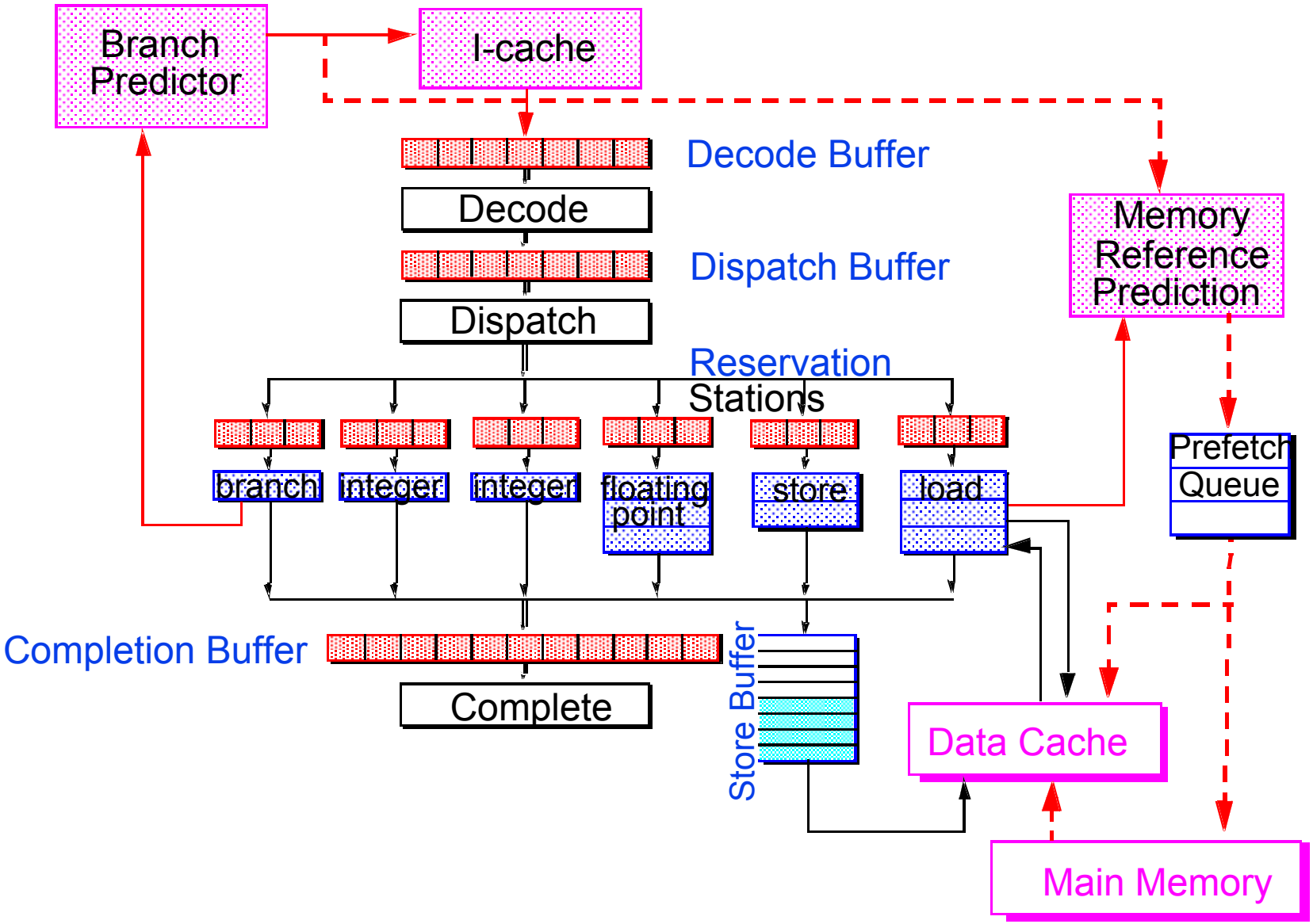
- Reminder: Use pencil or black/blue ink only

Prefetching

- Cache miss is either expensive or very expensive
- If we can foretell which address the program will reference in the future then we can ensure the location is in the cache ahead of time $\Rightarrow$ *No cache miss!!*
- Like branch prediction, prefetching takes advantage of regular/repeatable program behavior, in this case the memory reference pattern)
- Unlike branch prediction, prefetching is quite safe
- Only involves prediction but no "speculative execution"

Prefetching the wrong location can only affect processor performance (more misses) but not correctness

Prefetching Data Cache



What is Hard about Prefetching

- Must correctly guess both *when* to prefetch and *which address* to prefetch

- What if you prefetch the wrong lines
  - cache pollution
  - waste of bandwidth

    both bus bandwidth & cache port bandwidth

- What if you prefetch at the wrong time
  - too early, cache pollution
  - too late, ineffective

Static Prefetch

- PowerPC *Data Cache Block Touch* Instruction (dcbt *EA*)

  "a hint that performance will probably be improved if the block containing the byte addressed by EA is fetched into the data cache"
- A correct implementation of dcbt is to do nothing
- Or, as a load instruction with no destination register

  except it should not trigger page or protection faults
- Where should compilers insert dcbt?
  - In front of every load: *wastes I-cache and D-cache bandwidth*
  - Where are loads likely to miss
    - When traversing large data sets (arrays in scientific code)
  - Where load misses would really hurt performance
    - pointer arguments to functions
    - linked-list traversal - find loads who data address is itself the result of a previous load

Spatial Locality and Sequential Prefetching

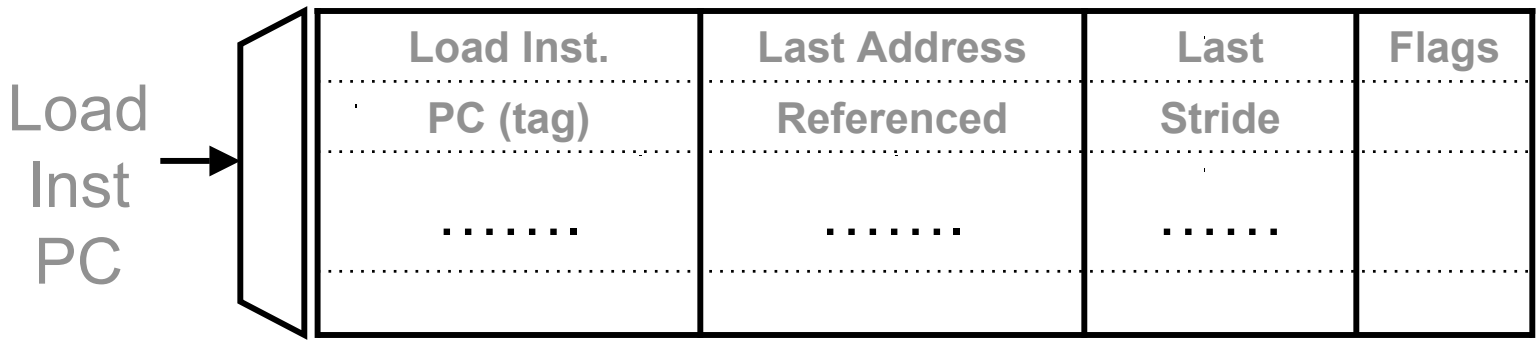
- Works well for I-cache
  - Instruction fetching tend to access memory sequentially
- Doesn't work very well for D-cache
  - More irregular access pattern
  - regular patterns may have non-unit stride (e.g. matrix code)
- Relatively easy to implement
  - Large cache block size already have the effect of prefetching
  - After loading one-cache line, start loading the next line automatically if the line is not in cache and the bus is not busy
- What if you fetch at the wrong time

Imagine if you started sequential prefetching of a long cache line and so happens you get a load miss to the middle of that line?

A critical-word-first reload triggered by the load miss itself may actually have restarted computation sooner!!

Stride Prefetchers

- Access pattern for a particular static load is more predictable
- Reference Prediction Table



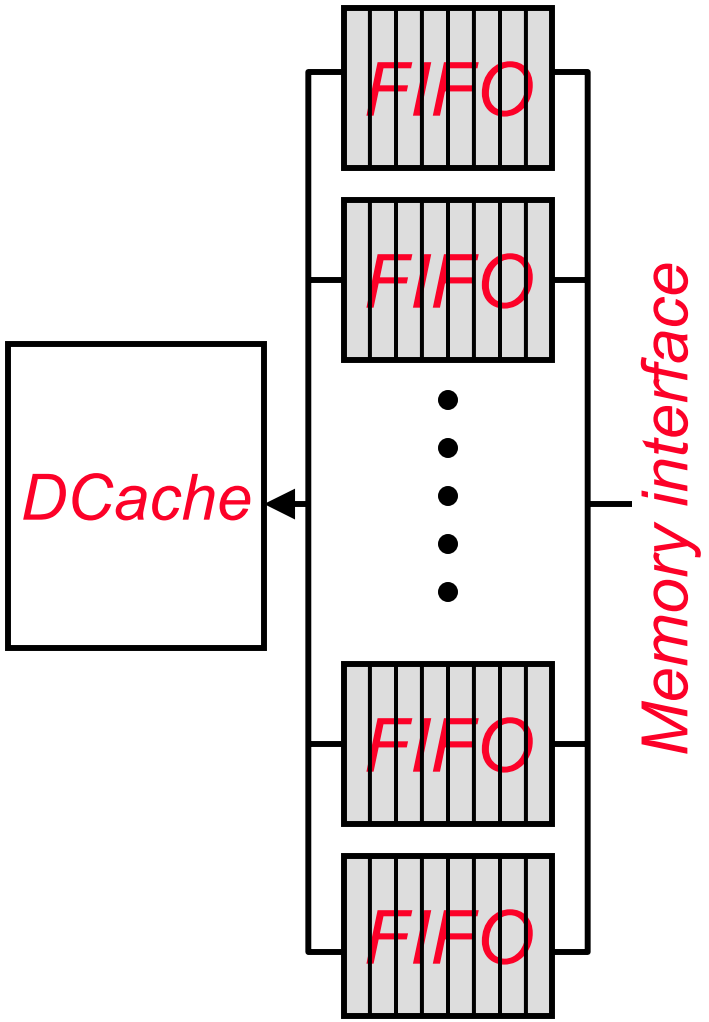
- Remembers previously executed loads, their PC, the last address referenced, stride between the last two references
- When executing a load, look up in RPT and compute the distance between the current data addr and the last addr
  - if the new distance matches the old stride
    $\Rightarrow$ found a pattern, go ahead and prefetch "current addr+stride"
  - update "last addr" and "last stride" for next lookup

Stream Buffers

- Each stream buffer holds one stream of sequentially prefetched cache lines

  No cache pollution

- On a load miss check the head of all stream buffers for an address match
  - if hit, pop the entry from FIFO, update the cache with data
  - if not, allocate a new stream buffer to the new miss address (may have to recycle a stream buffer following LRU policy)
- Stream buffer FIFOs are continuously topped-off with subsequent cache lines whenever there is room and the bus is not busy
- Stream buffers can incorporate stride prediction mechanisms to support non-unit-stride streams

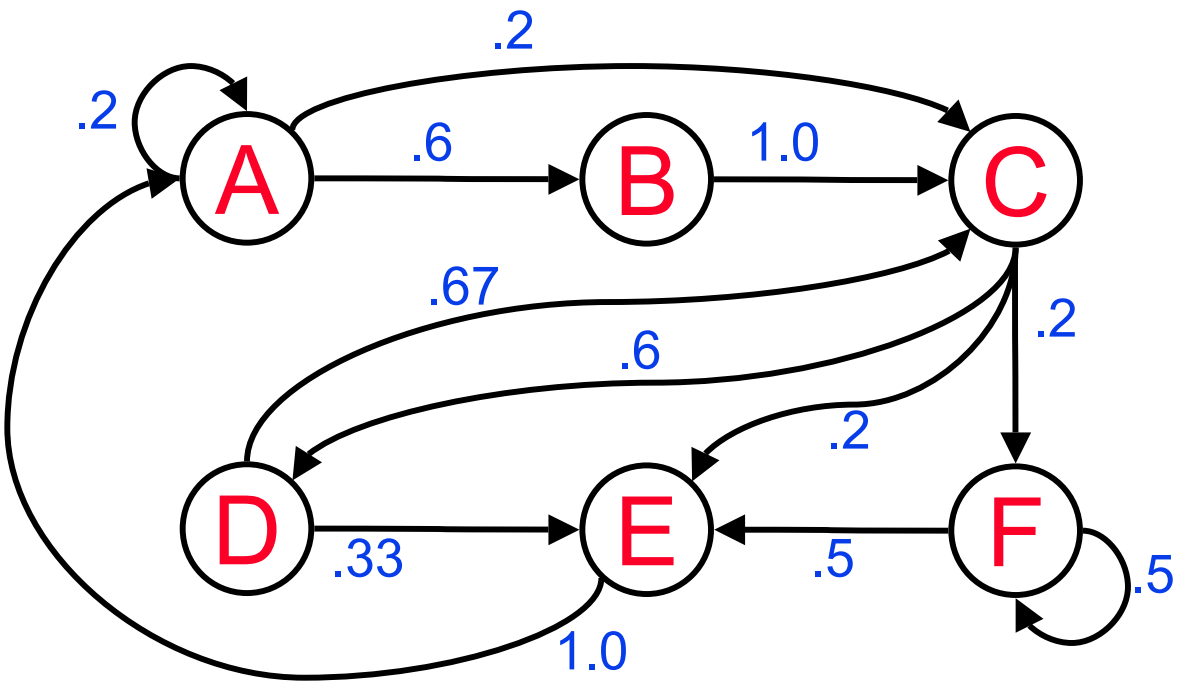


Correlation-Based Prefetching

- Consider the following history of Load addresses emitted by a processor

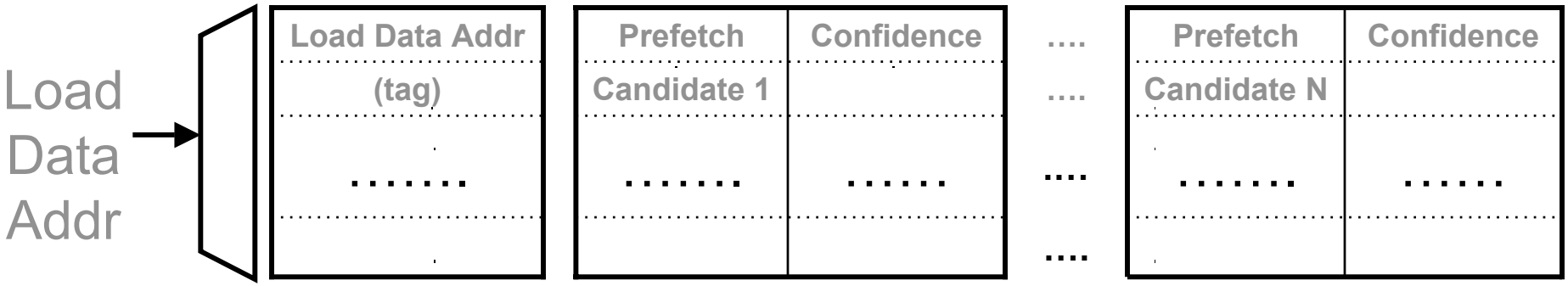
  A, B, C, D, C, E, A, C, F, F, E, A, A, B, C, D, E, A ,B C, D, C

- After referencing a particular address (say A or E), are some addresses more likely to be referenced next



Markov Model

Correlation-Based Prefetching



- Track the likely next addresses after seeing a particular addr.
- Prefetch *accuracy* is generally low so prefetch up to N next addresses to increase *coverage* *(but this wastes bandwidth)*
- Prefetch accuracy can be improved by using longer history
  - Decide which address to prefetch next by looking at the last K load addresses instead of just the current one
  - e.g. index with the XOR of the data addresses from the last K loads

    Very similar to the idea behind global branch prediction
  - Using history of a couple loads can increase accuracy dramatically

This technique can also be applied to just the load miss stream

Software Controlled Memory Hierarchy

- Expose cache hierarchy to the programmer
- Controlling the size of the cache
  - If the program is known to have a small footprint, turn off half of the L1 or turn off the entire L2 to save power
- Controlling the associativity
  - Assign individual L1 banks to different software threads so they don't thrash each other
  - Designate a specific L1 bank for streaming references only so you don't displace the rest of the cache contents
- Fine-grain cache control instructions, e.g.
  - Lock a particular cache line from displacement
  - Prefetch an address up to just L2 but not L1
  - Start a new stream buffer prefetch

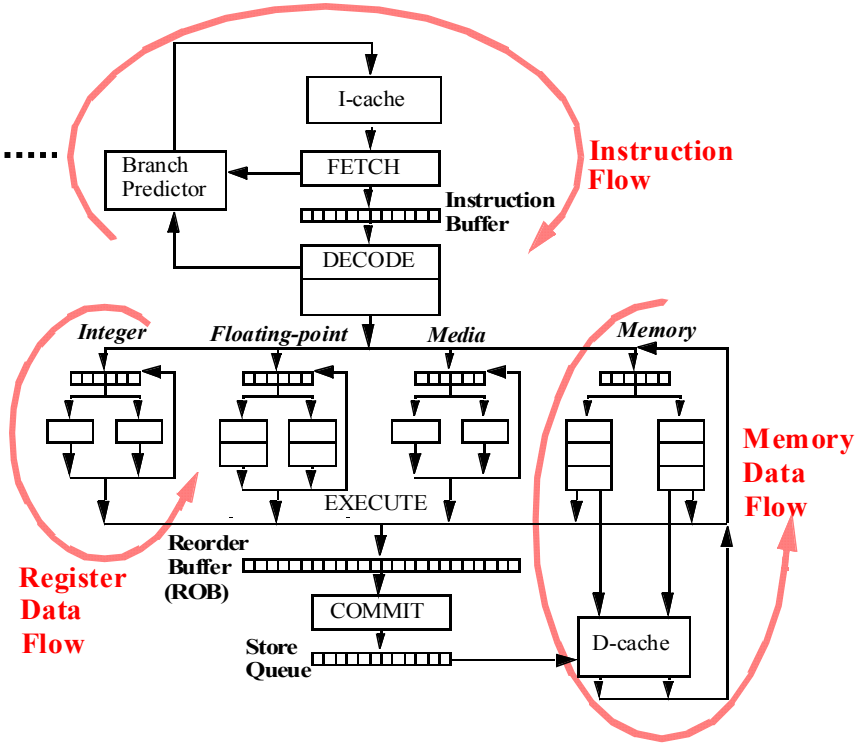
Our Picture of Superscalar Micros

- First half of the course

  Single-instruction stream

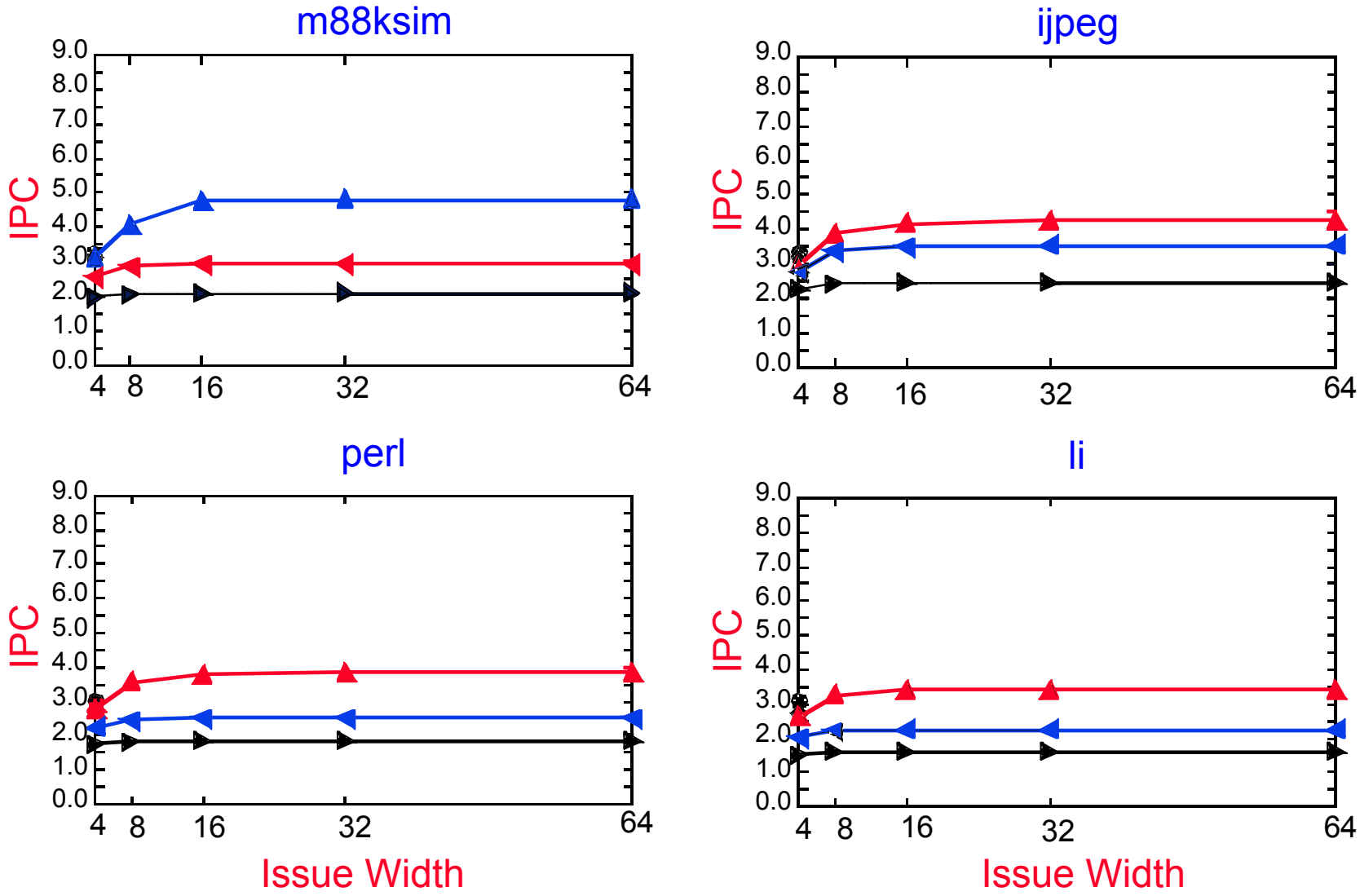
  shrink-wrap binary

- What's to come
  - Software/compilation based techniques
  - Heavy-duty speculation
  - New ISA design and support
  - New microarchitecture style
  - System-level parallelism



Higher performance, lower power, anything goes

Dataflow Limit on Superscalar Micros



Cycles before dispatch: ▲1 ◀2 ▶3

Assume infinite functional units

Breaking Dataflow Dependence:

Prediction and Speculation

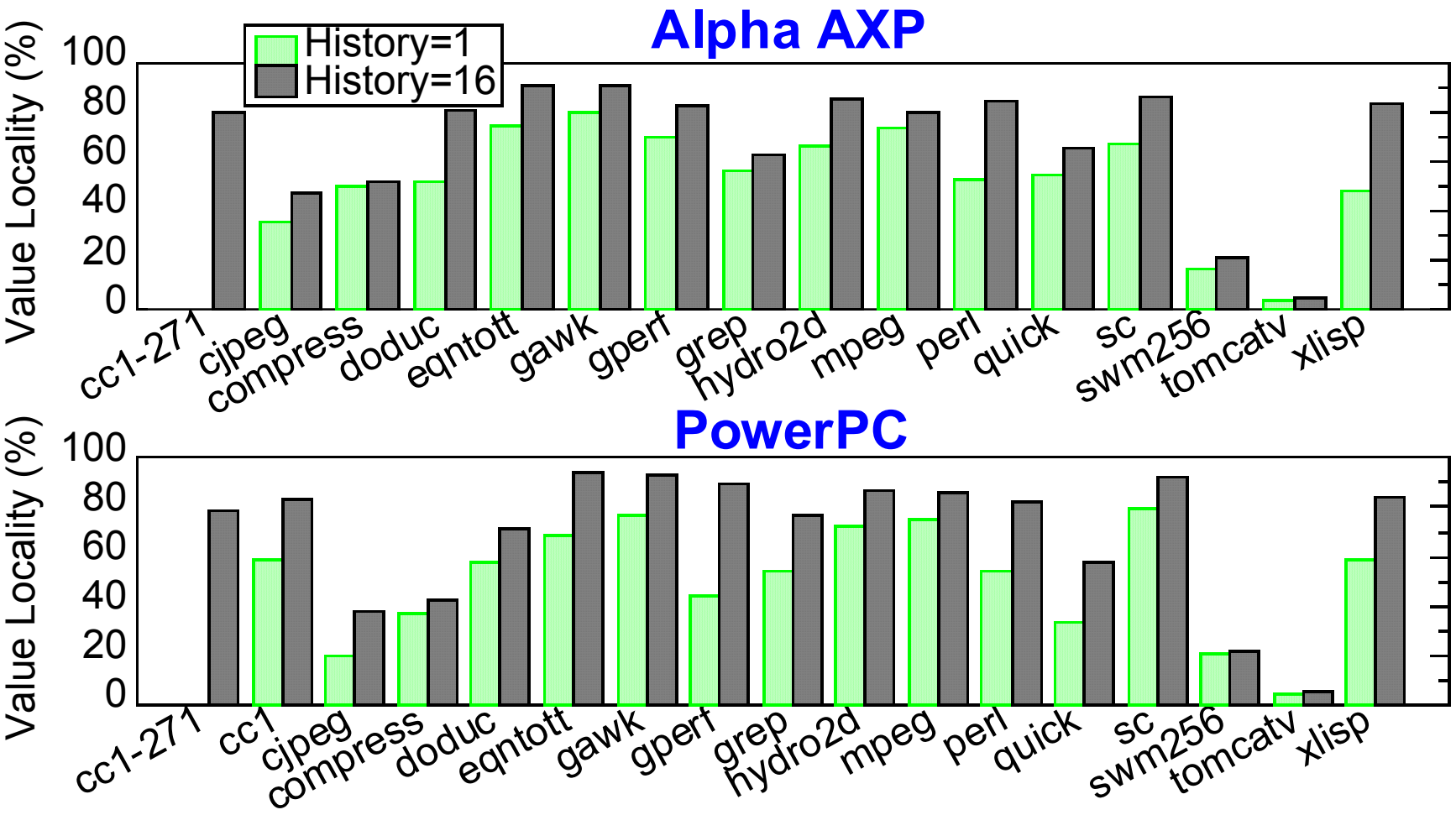
- Branch prediction:
  - Branch target history
  - Branch direction history
- Load value prediction:
  - Value history for each static load
- Register value prediction:
  - Source or destination value history per static instruction operand
- Dependence and alias prediction:
  - Source or destination dependence distance history

Assumes a very large transistor budget

1. Large complicated prediction logic for accuracy

2. Spare resources to spend on speculated computation

Load Value Locality



Optimized, two ISAs, SPECish benchmarks (720M instr)

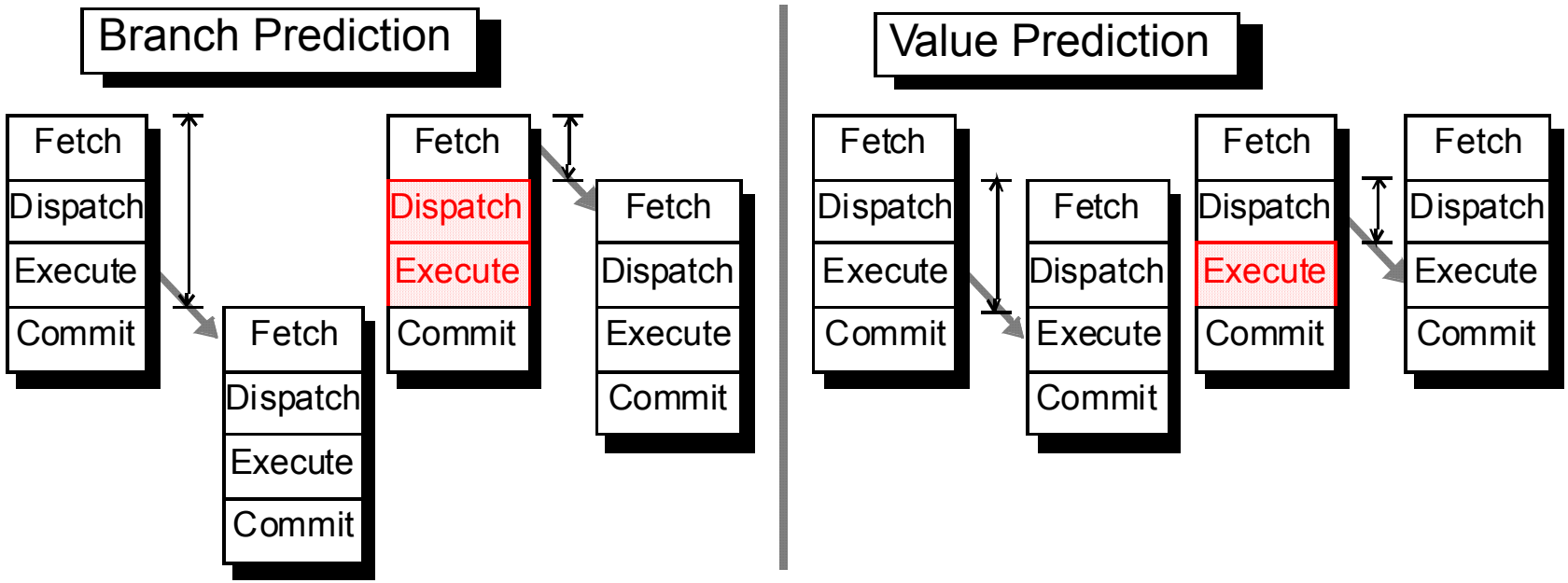
Frequently >= **50%** with history depth of 1

Frequently >= **80%** with history depth of 16

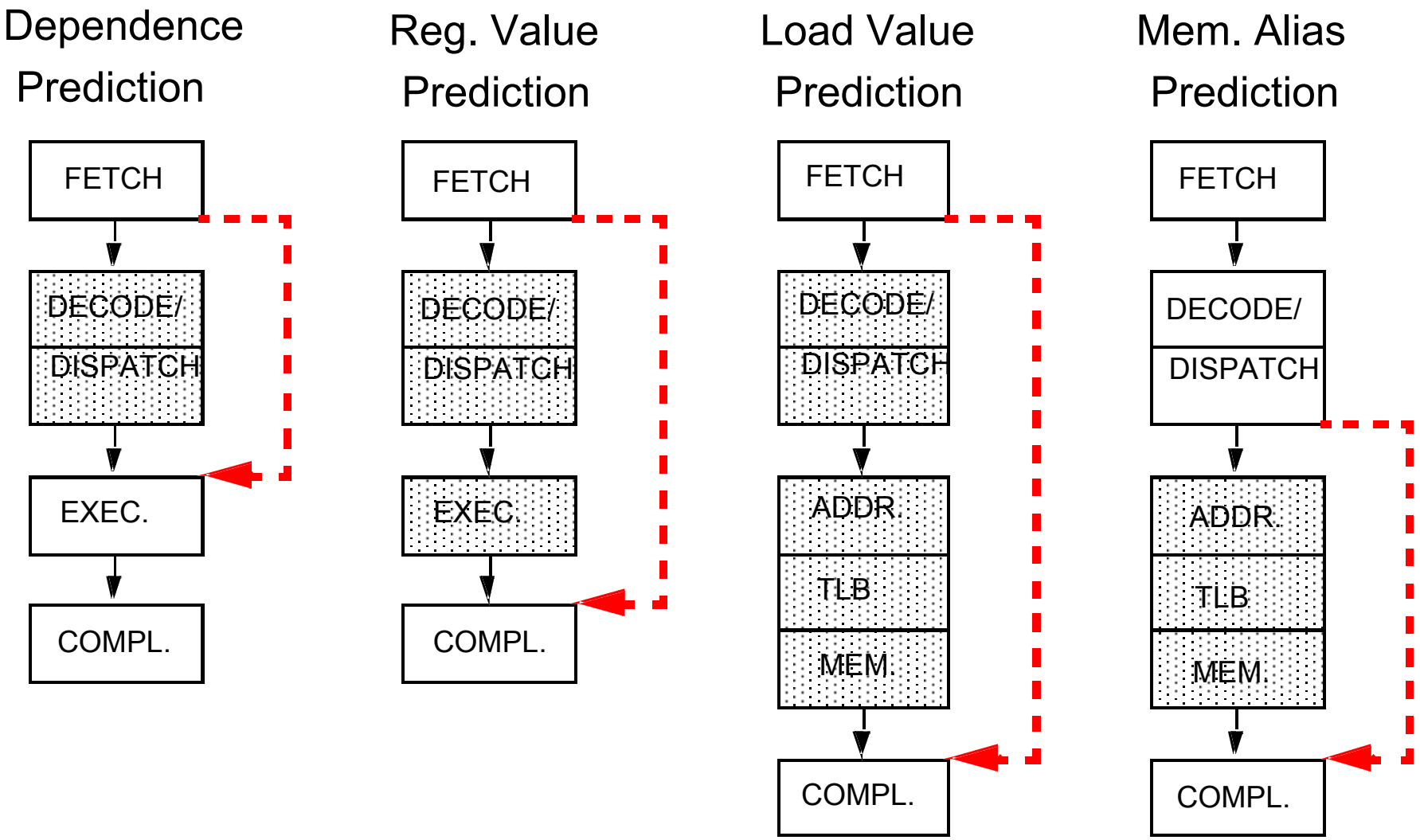
Generalized Value Locality

Dynamic Pipeline Contraction

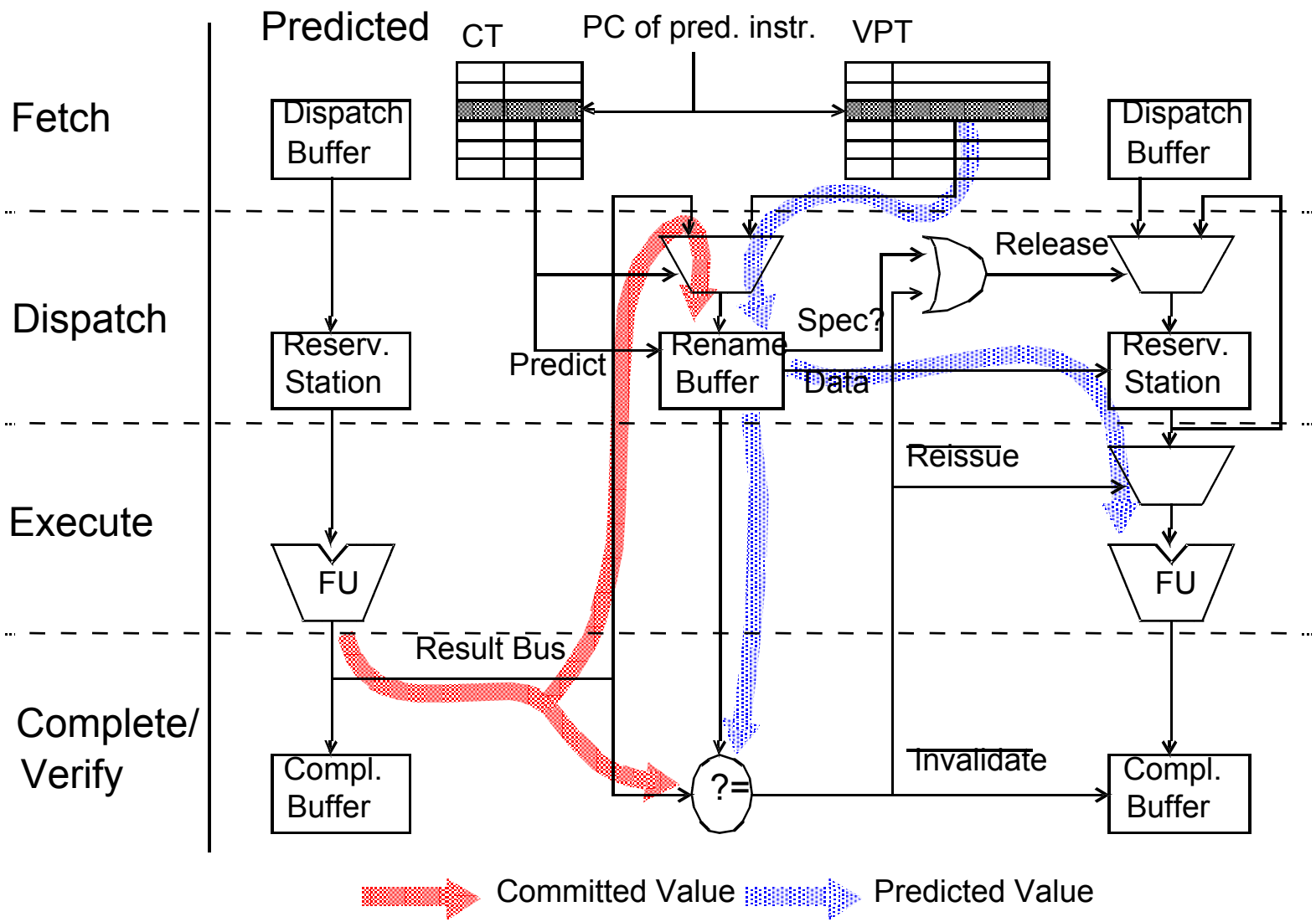
- Fold away pipeline stages via speculation:
  - Predict: obtain semantic outcome of instruction early
  - Speculate: allow dependent instr. to execute in parallel
  - Recover: Perform fix-up when mis-speculation occurs



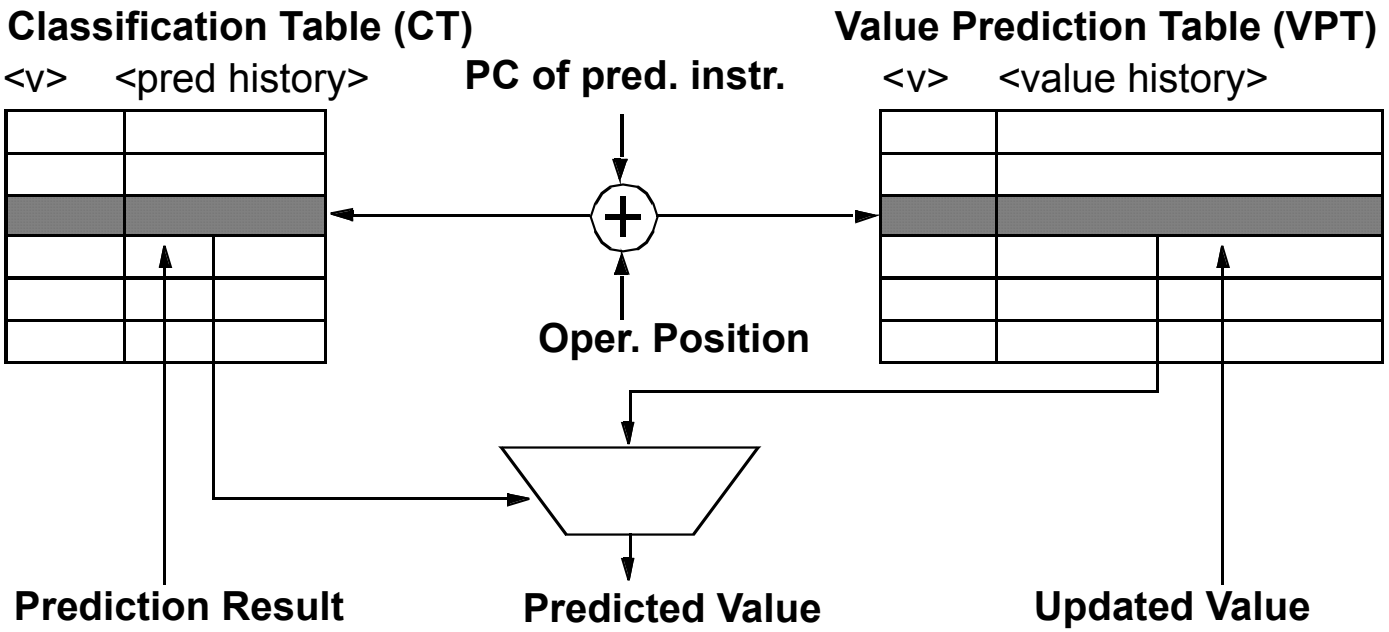
Superspeculation Pipeline Contractions



Value Prediction Mechanism

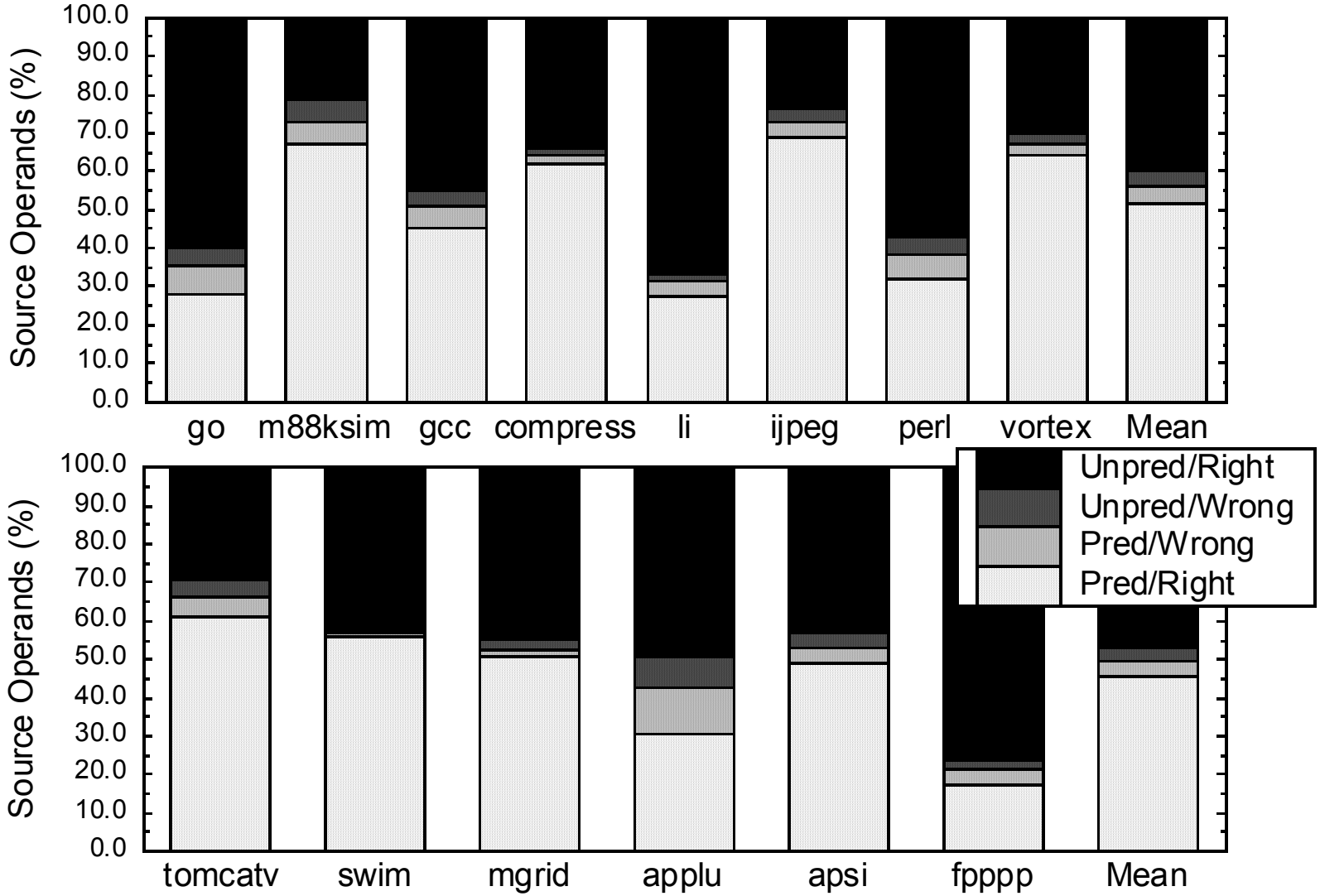


Source Operand Value Prediction



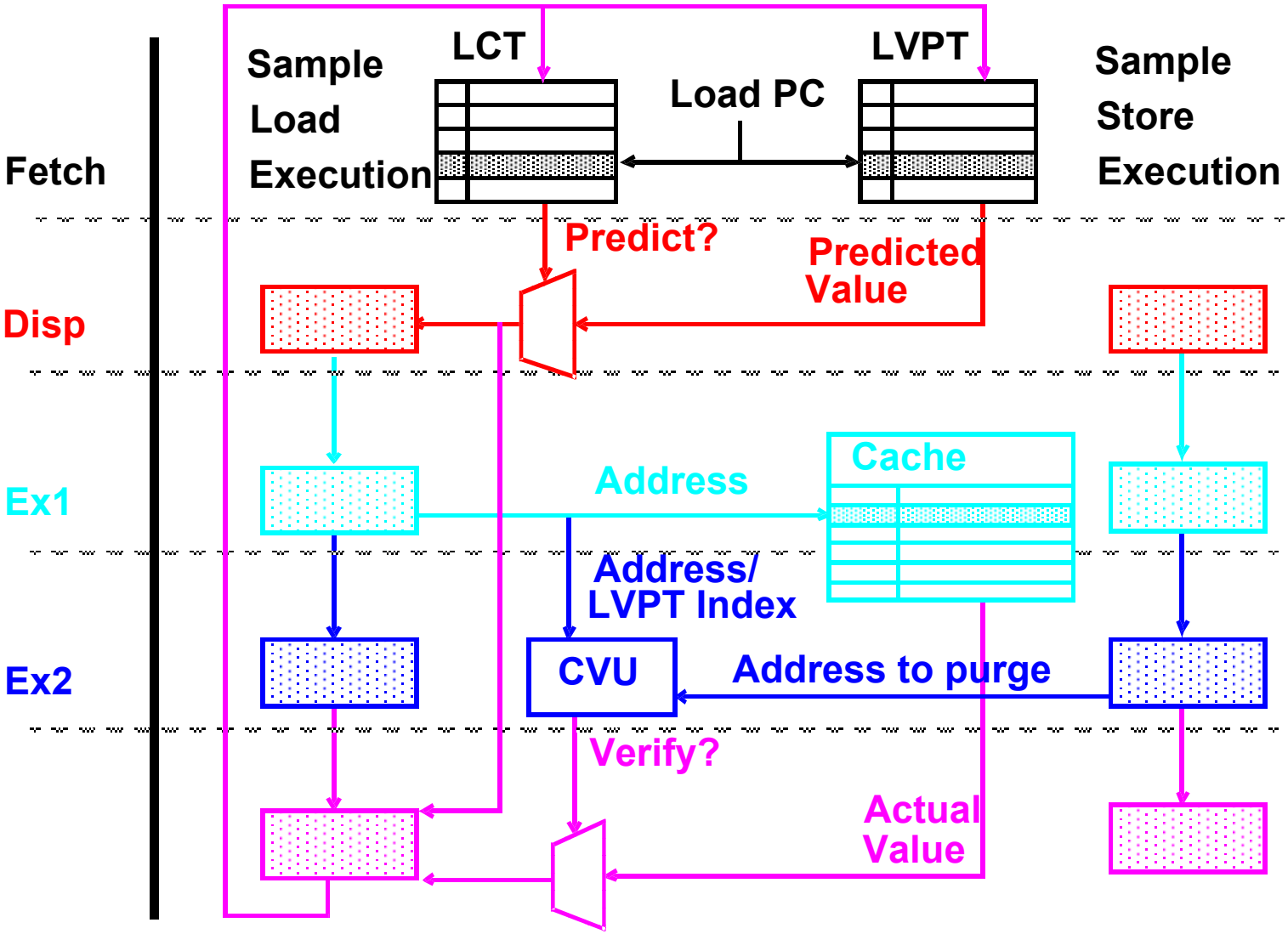
- Similar to earlier work on value prediction, but predicts source operands:
  - Decouples execution from dependence checking
  - Don't care where value is coming from until validation
- Confidence mechanism (CT) filters out wrong predictions

Source Operand Value Locality

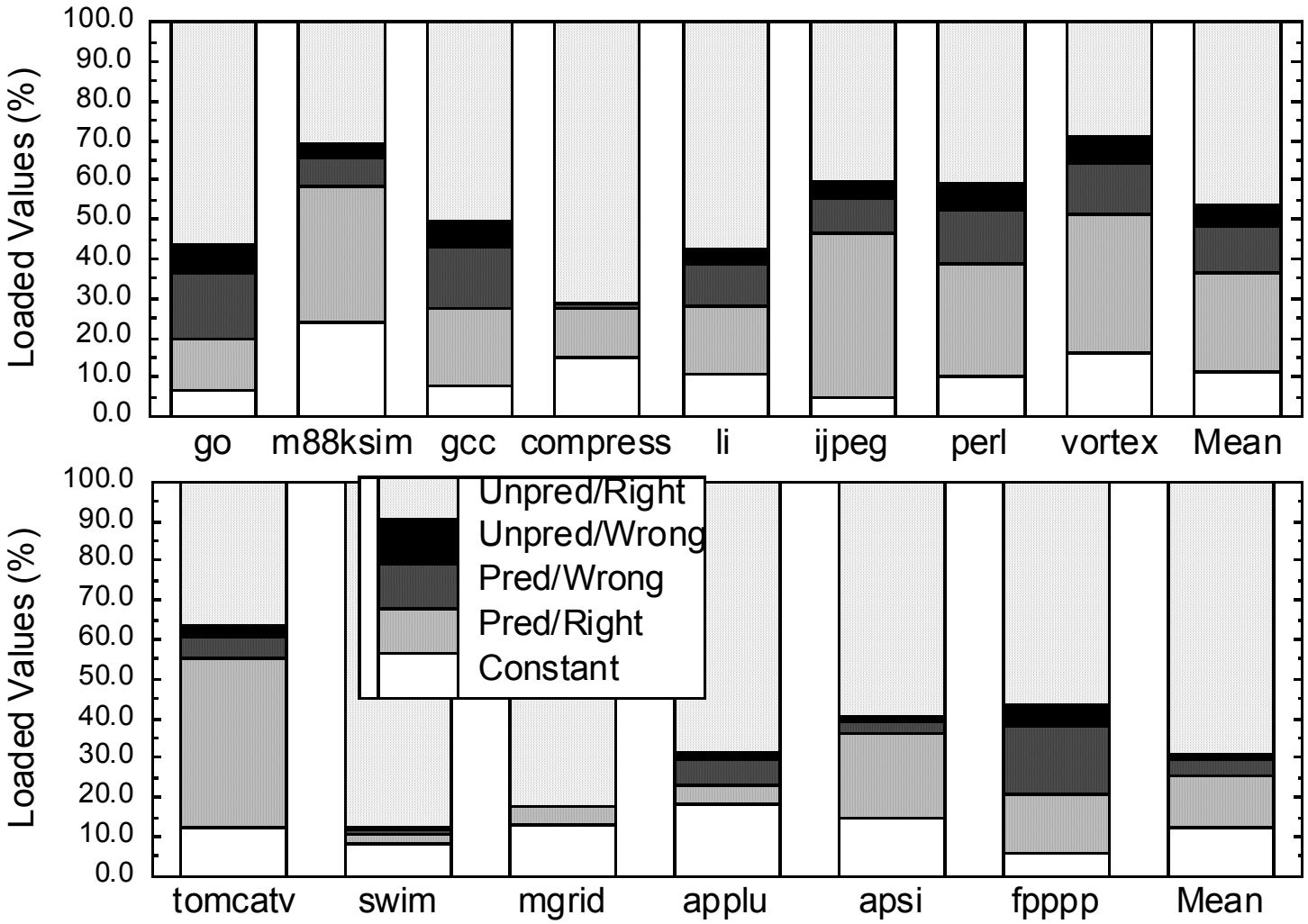


~50% of source operands can be correctly predicted

Load Value Prediction and Classification



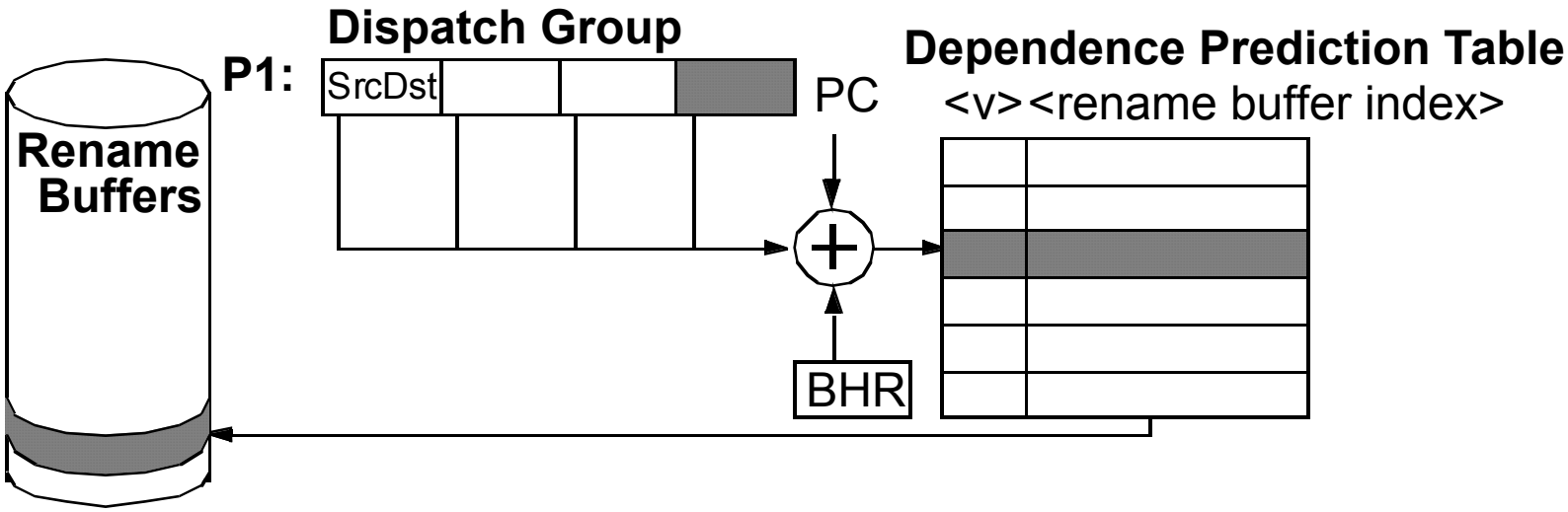
Load Value Locality



~10% of loaded values are constant-promoted

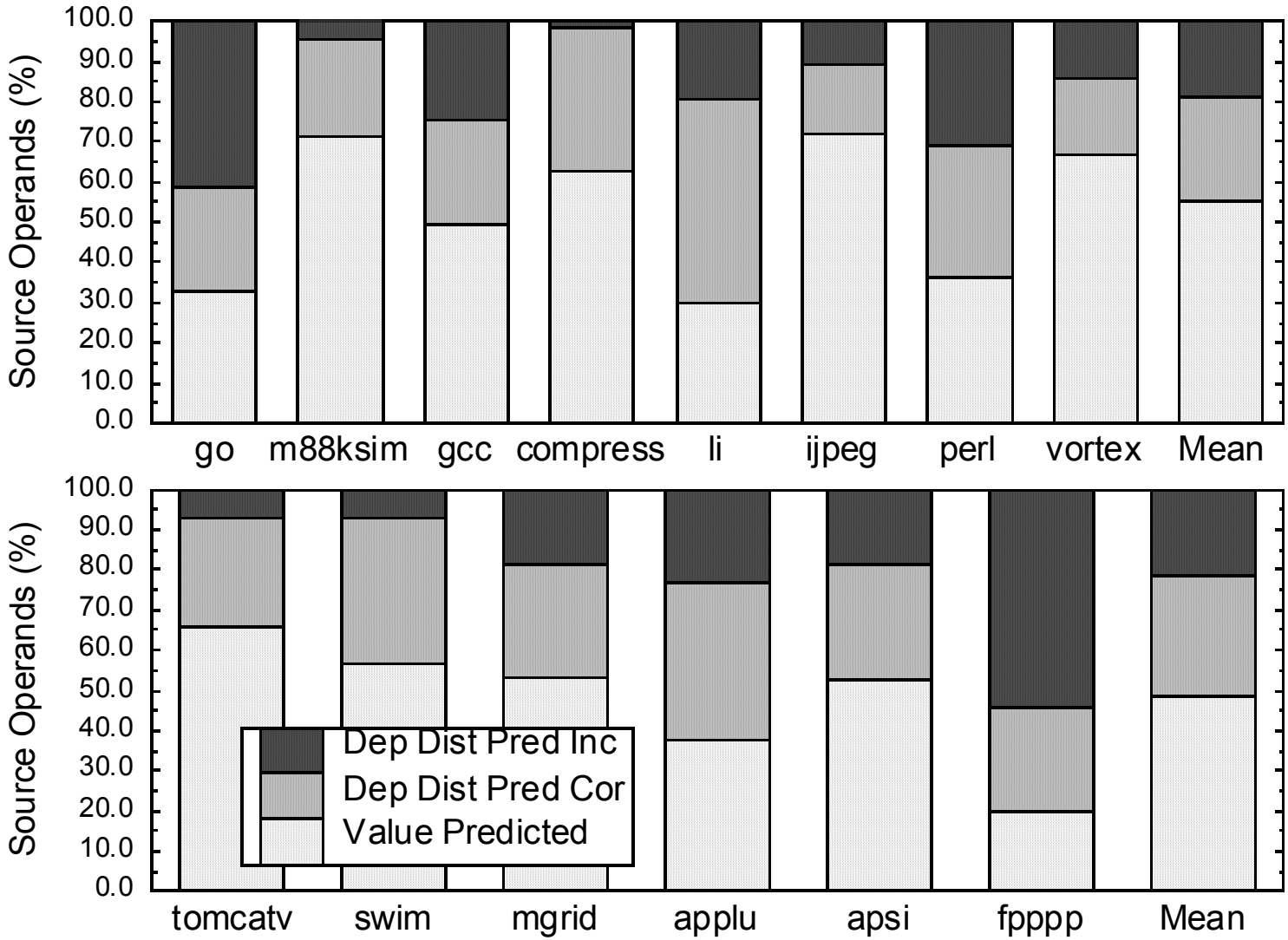
~35% (int) and 25% (FP) of loaded values are predicted

Dependence Prediction Mechanism



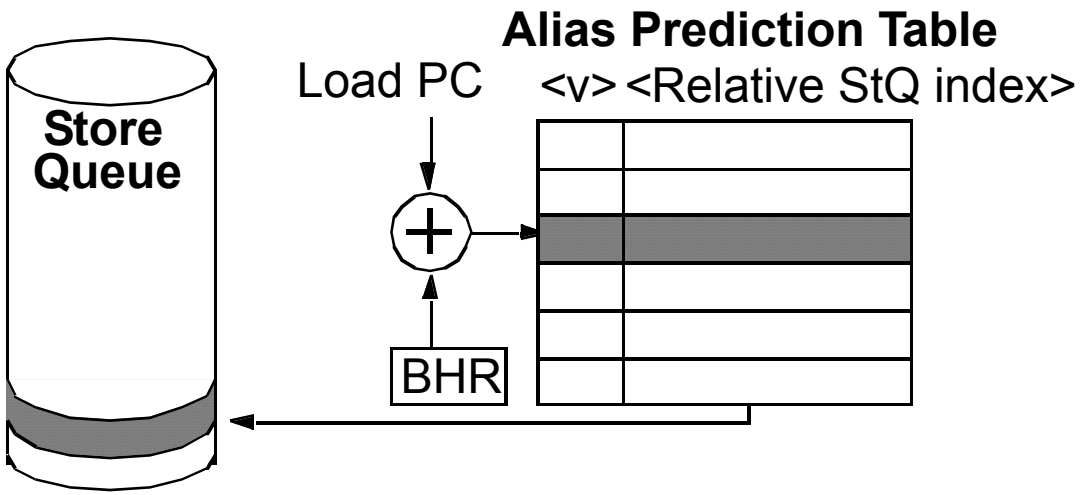
- Keeps dependence distance history per instruction
- Bypasses dependence detection if dependence distance stays constant
- Allows execution to begin in parallel with exact dependence checking
- Incorrect prediction results in execution restart

Dependence Predictability



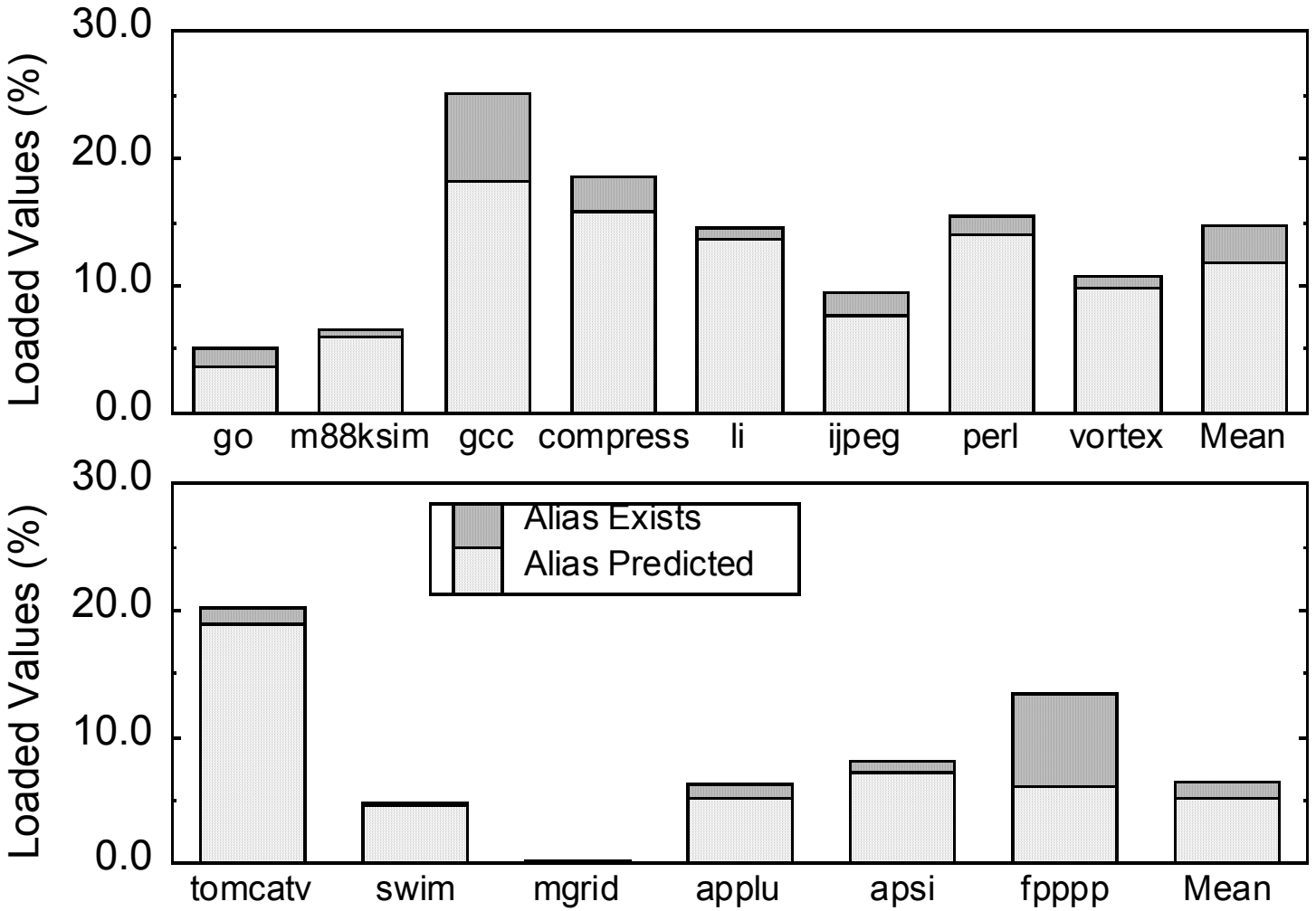
~80% of source operands can be value- or dependence predicted

Alias Prediction



- Speculatively forwards values from store queue
- Dependence distance history used to predict the right store queue entry
- Address generation is taken off the critical path

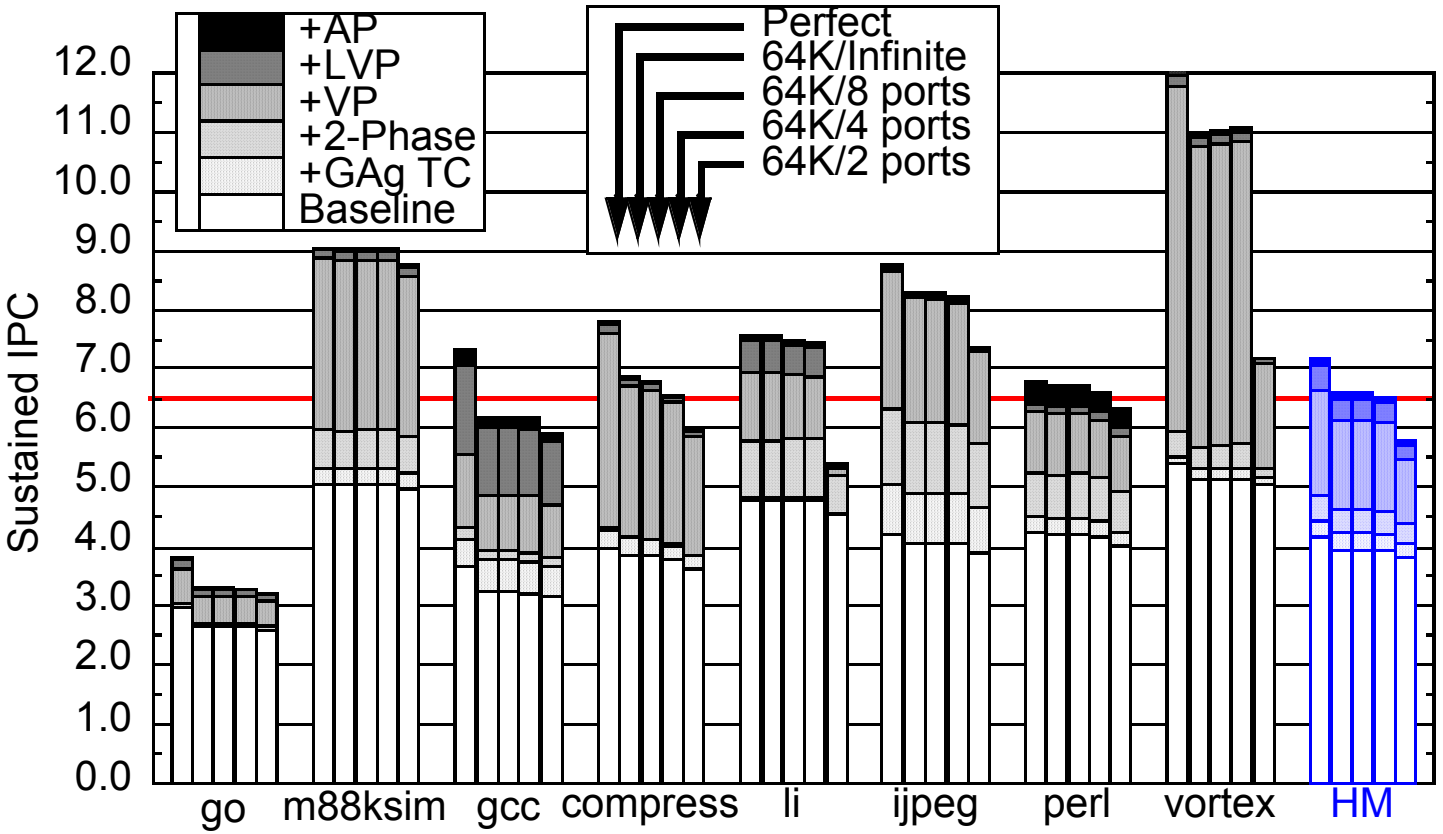
Alias Predictability



~15% (int) and 6% (FP) of loads are aliased

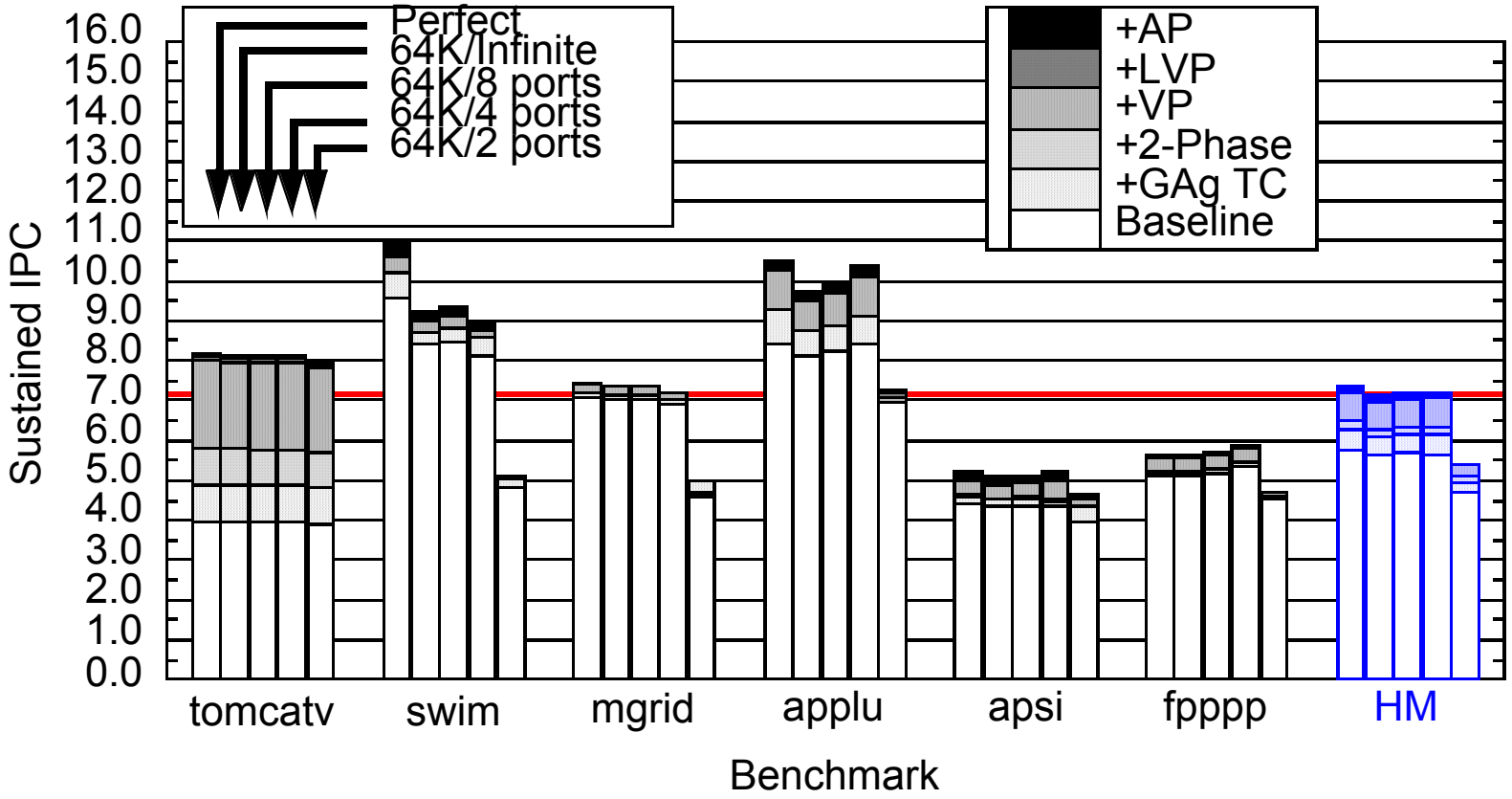
~80% of aliases are predicted correctly (no AGI)

SPECInt95 Performance (16-wide)



- Finite memory bandwidth only slightly degrades performance down to 4 cache ports
- At 16-wide, 4 cache ports, ROB=128, ---> IPC = 6.7

SPECFP95 Performance (16-wide)



- FP more sensitive to memory bandwidth
- Fewer aliases, more cache misses, limited by ROB size
- At 16-wide, 4 cache ports, ROB=256, ---> IPC = 7.2

Performance Potential of Superspeculation