RFVP: Rollback-Free Value Prediction with Safe to Approximate Loads

Amir Yazdanbakhsh,
Bradley Thwaites,
Hadi Esmaeilzadeh

Gennady Pekhimenko,
Onur Mutlu,
Todd C. Mowry

Georgia Institute of Technology
Carnegie Mellon University

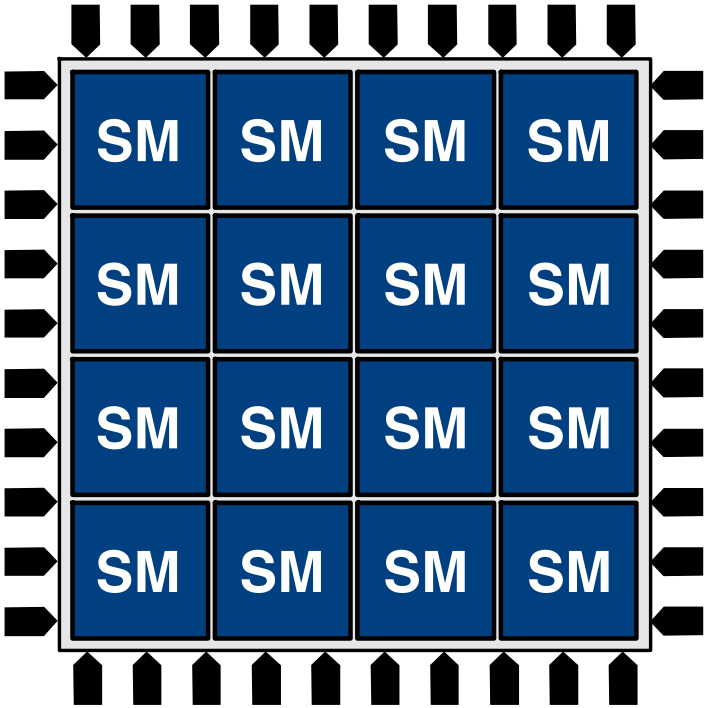
Carnegie Mellon University

Executive Summary

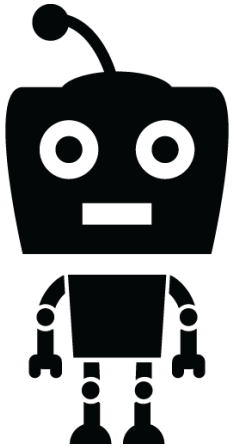
- **Problem**: Performance of modern GPUs significantly limited by the available off-chip bandwidth
- **Observations**:
  - Many GPU applications are amenable to approximation
  - Data value similarity allows to efficiently predict values of cache misses
- **Key Idea**: Use simple value prediction mechanisms to avoid accesses to main memory when it is safe
- **Results:**
  - Higher speedup (**36%** on average) with less than 10% quality loss
  - Lower energy consumption (**27%** on average)

Virtual Reality

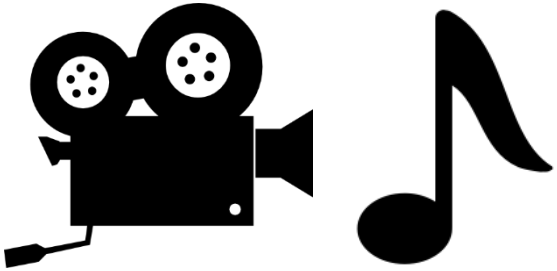
Data Analytics



GPU



Robotics



Multimedia

Virtual

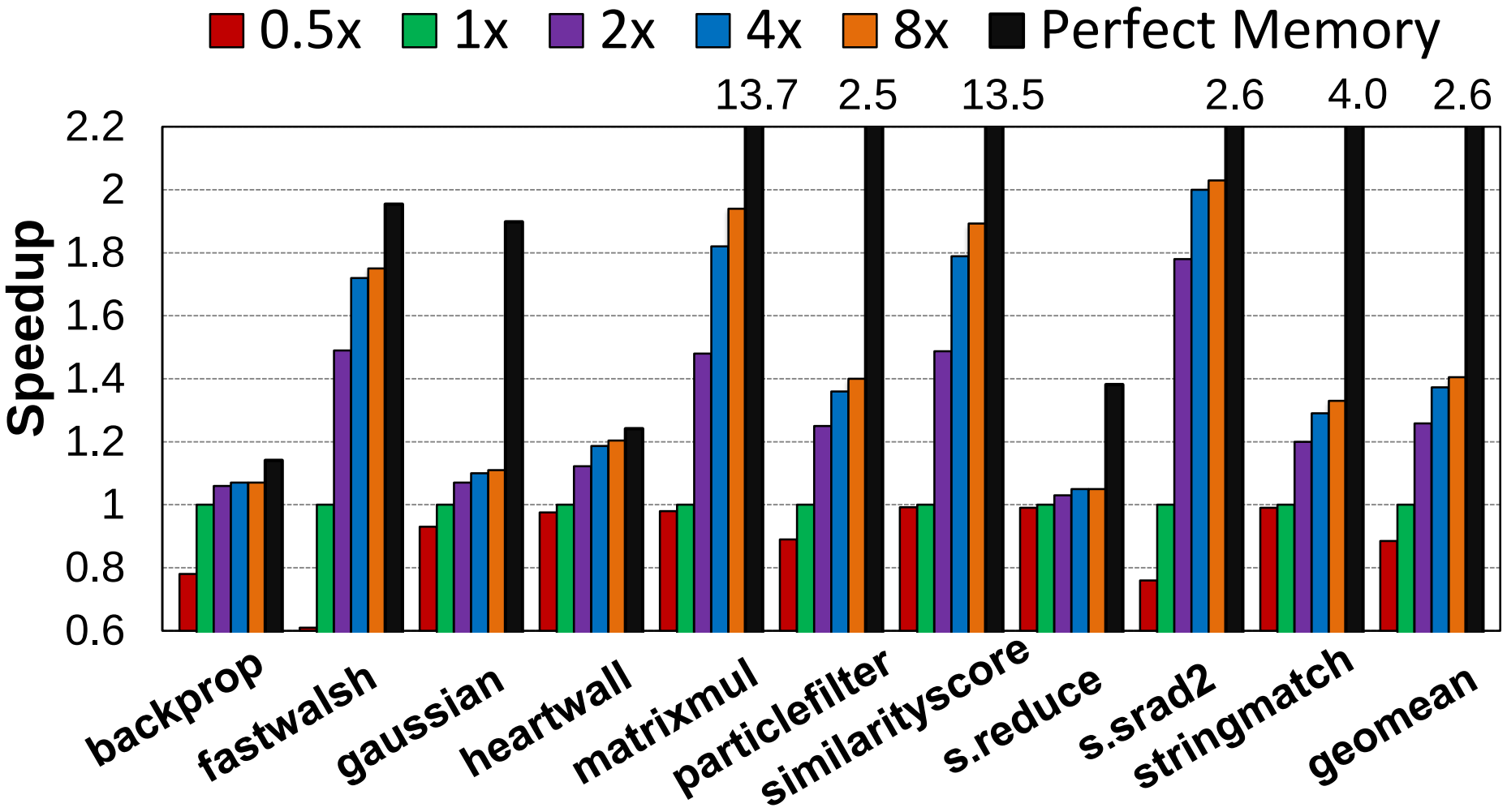
Data

Many GPU applications are limited by the off-chip bandwidth

Robotics

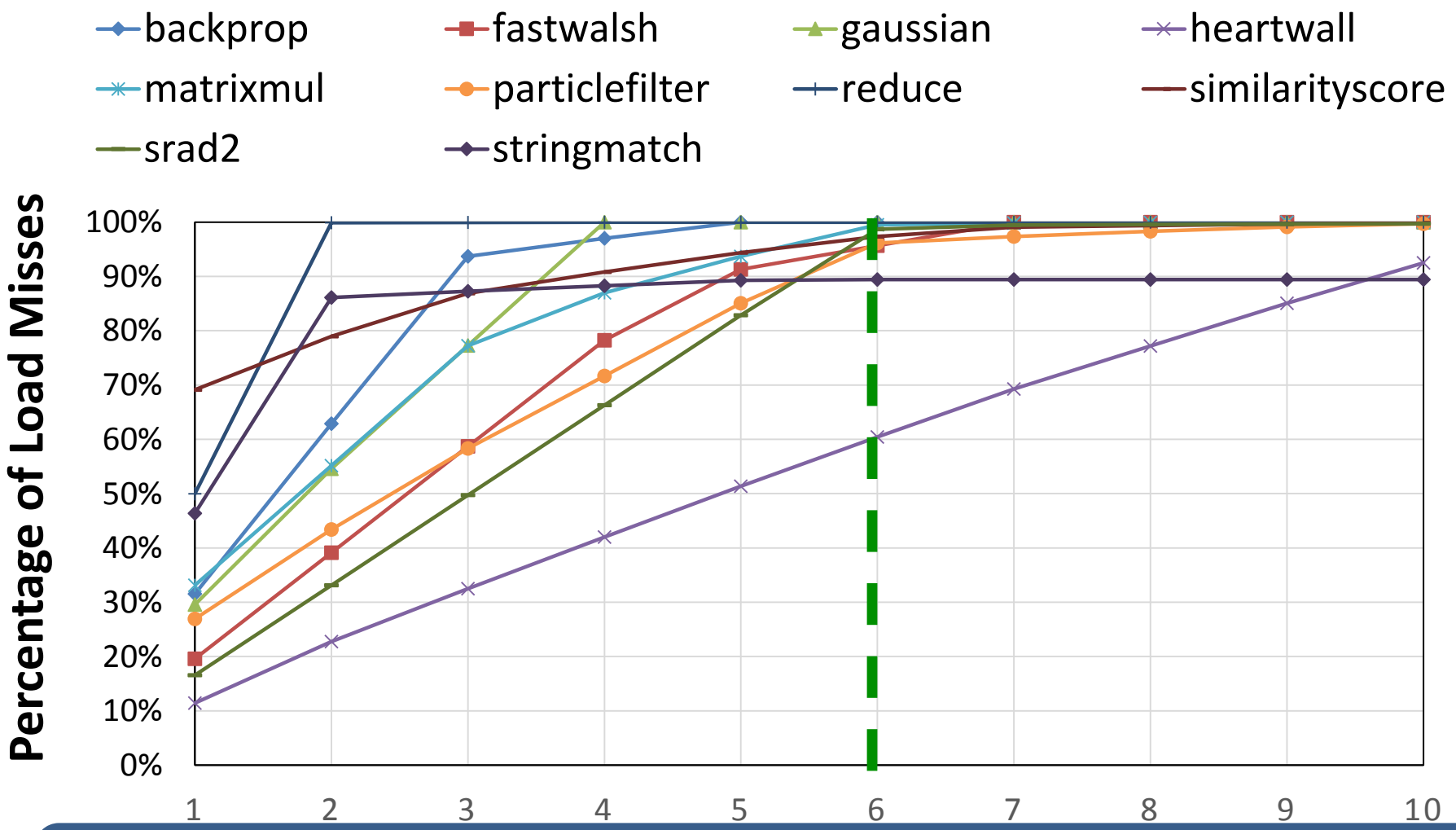
Multimedia

Motivation: Bandwidth Bottleneck



Off-chip bandwidth is a major performance bottleneck

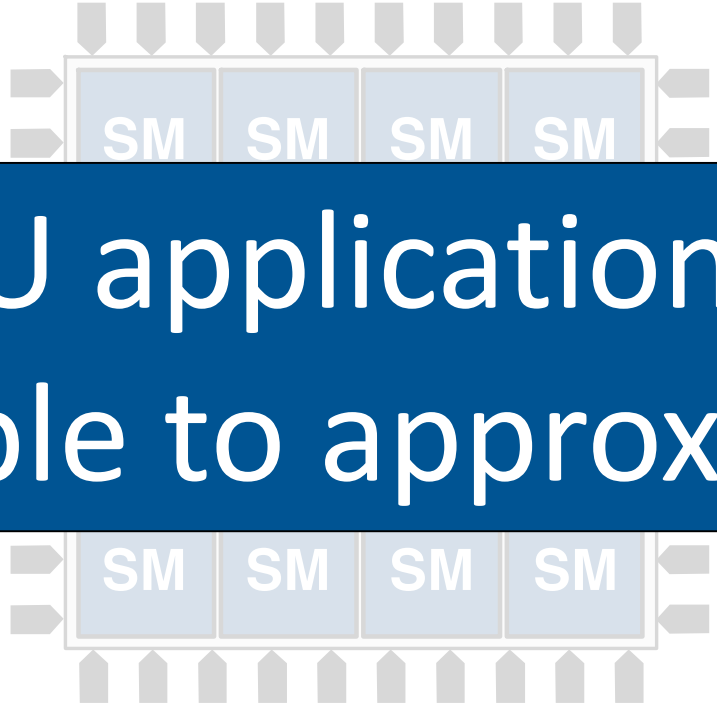
Only Few Loads Matters



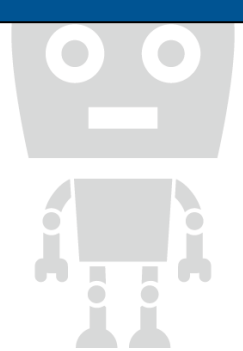
Few GPU instructions generate most of the cache misses

Virtual

Data



Many GPU applications are also amenable to approximation



Robotics

Multimedia

Rollback-Free Value Prediction

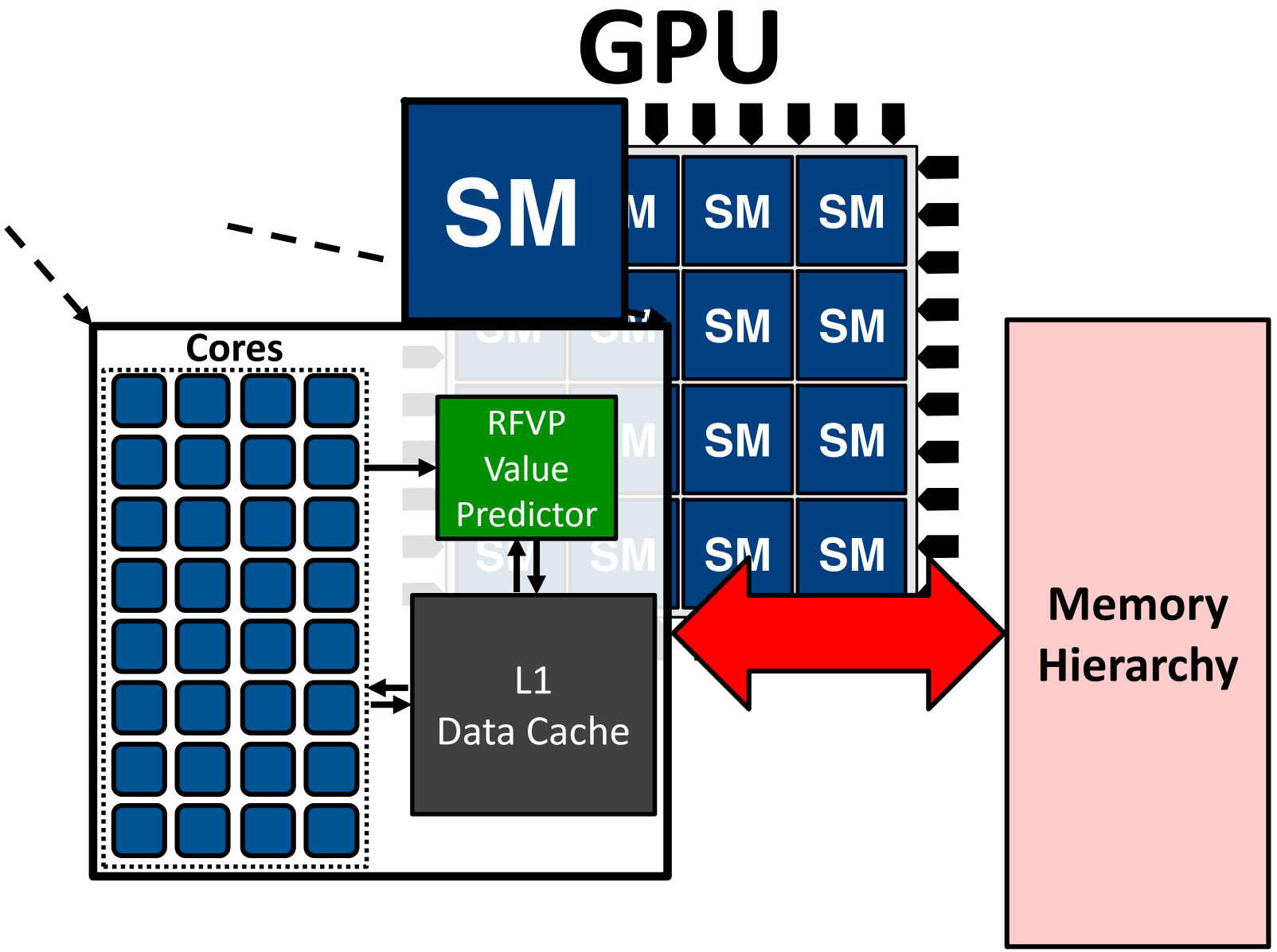
Key idea:

Predict values for safe-to-approximate loads when they miss in the cache

Design principles:

1. **No rollback/recovery**, only value prediction
2. **Drop rate** is a tuning knob
3. Other requests are serviced normally
4. Providing safety guarantees

RFVP: Diagram



Code Example to Support Intuition

Matrixmul:

```
float newVal = 0;
for (int i=0; i<N; i++) {
    float4 v1 = matrix1[i];
    float4 v2 = matrix2[i];
    newVal += v1.x * v2.x;
    newVal += v1.y * v2.y;
    newVal += v1.z * v2.z;
    newVal += v1.w * v2.w;
}
```

Code Example to Support Intuition (2)

s.srad2:

```
int d_cN, d_cS, d_cW, d_cE;

d_cN  = d_c[ei];
d_cS   = d_c[d_iS[row] + d_Nr * col];
d_cW = d_c[ei];
d_cE   = d_c[row + d_Nr * d_jE[col]];
```

Outline

- Motivation
- Key Idea
- RFVP Design and Operation
- Evaluation
- Conclusion

RFVP Architecture Design

- Instruction annotations by the programmer
- ISA changes
  - Approximate load instruction
  - Instruction for setting the drop rate
- Defining approximate load semantics
- Microarchitecture Integration

Programmer Annotations

- Safety is a **semantic property** of the program

- We rely on the programmer to **annotate** the code

ISA Support

- Approximate Loads

  load.approx Reg<id>, MEMORY<address>

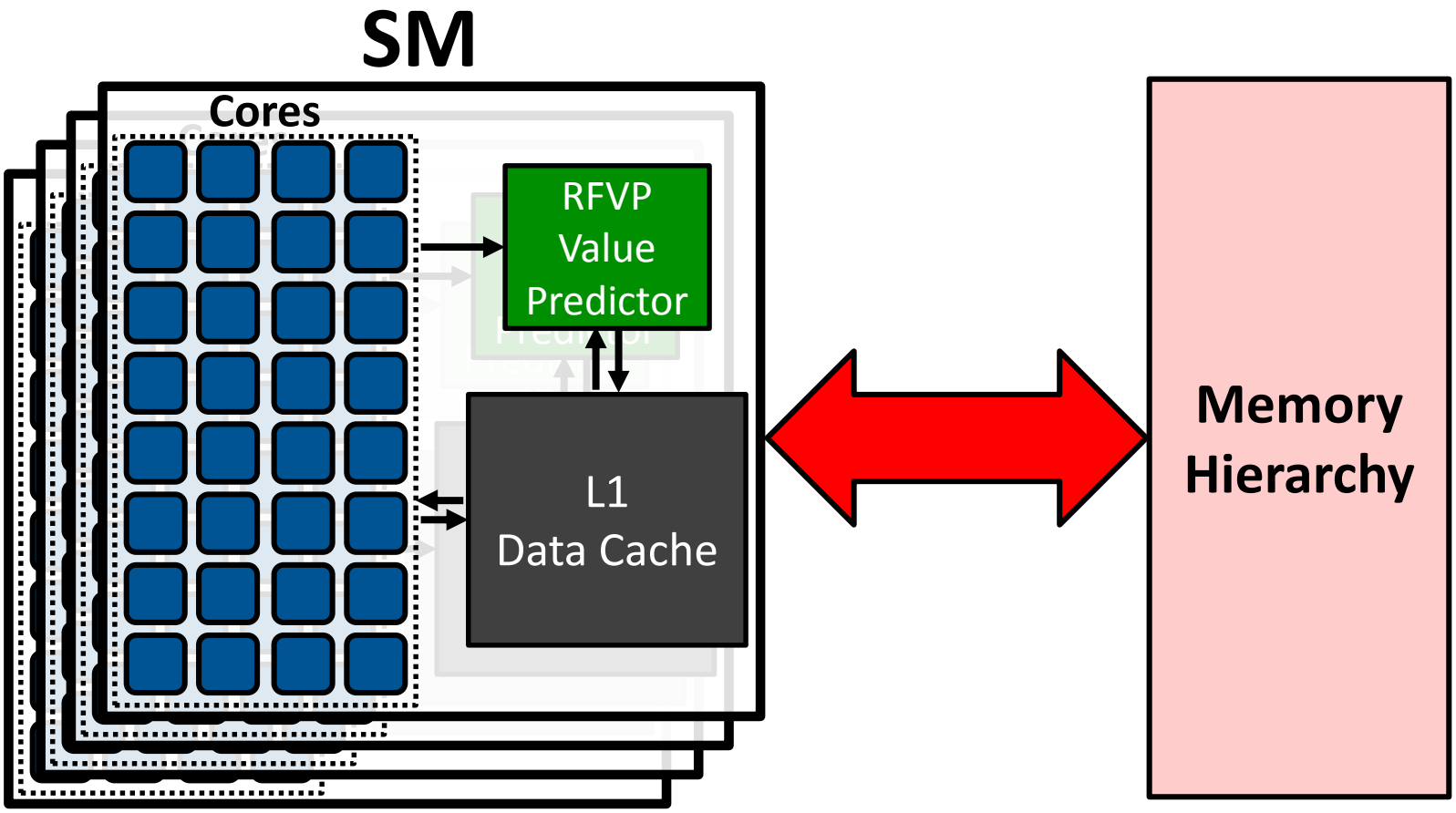
is a probabilistic load - can assign precise or imprecise value to Reg<id>

- Drop rate

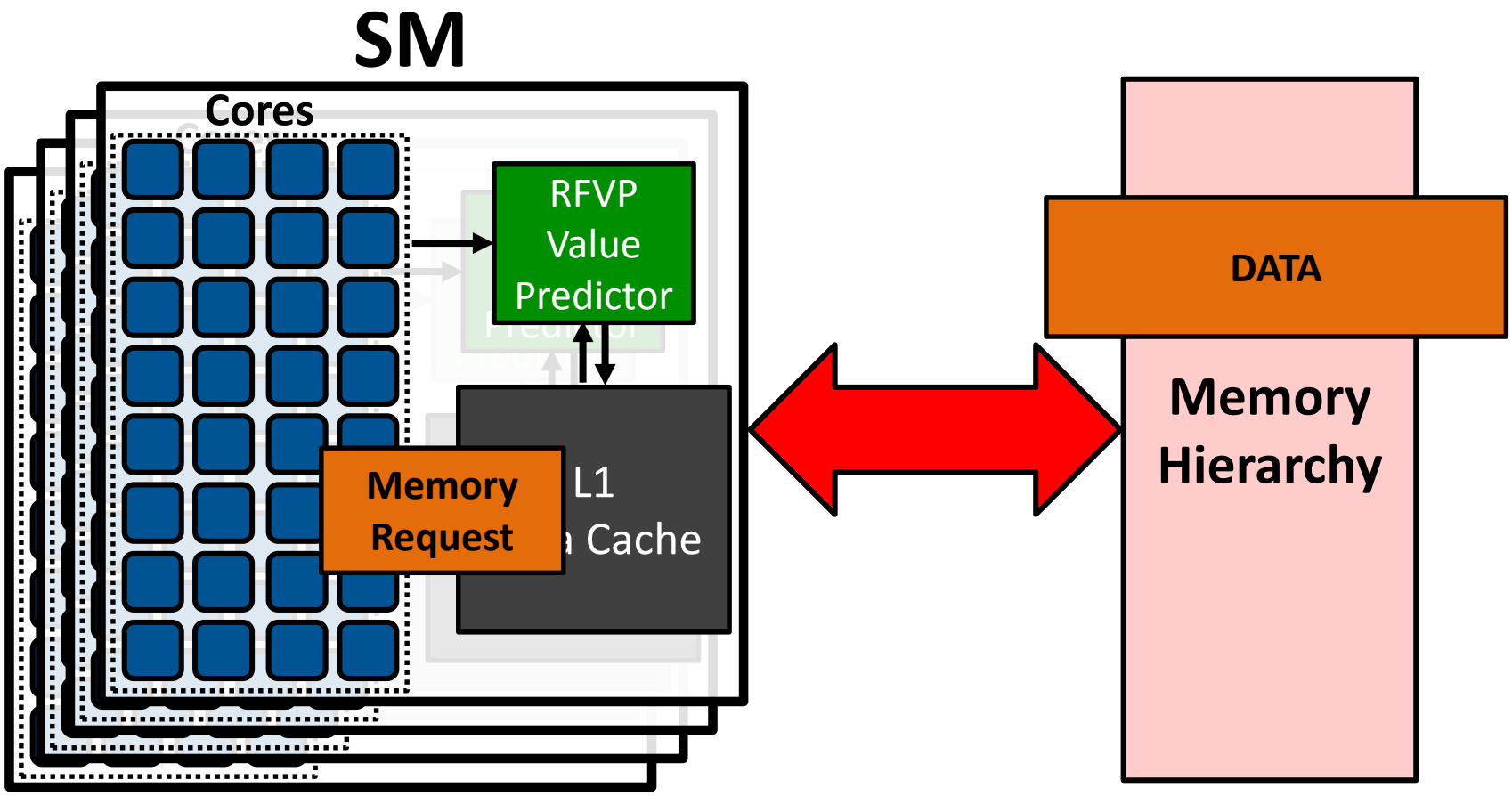
  set.rate DropRateReg

sets the fraction (e.g., 50%) of the approximate cache misses that do not initiate memory requests

Microarchitecture Integration

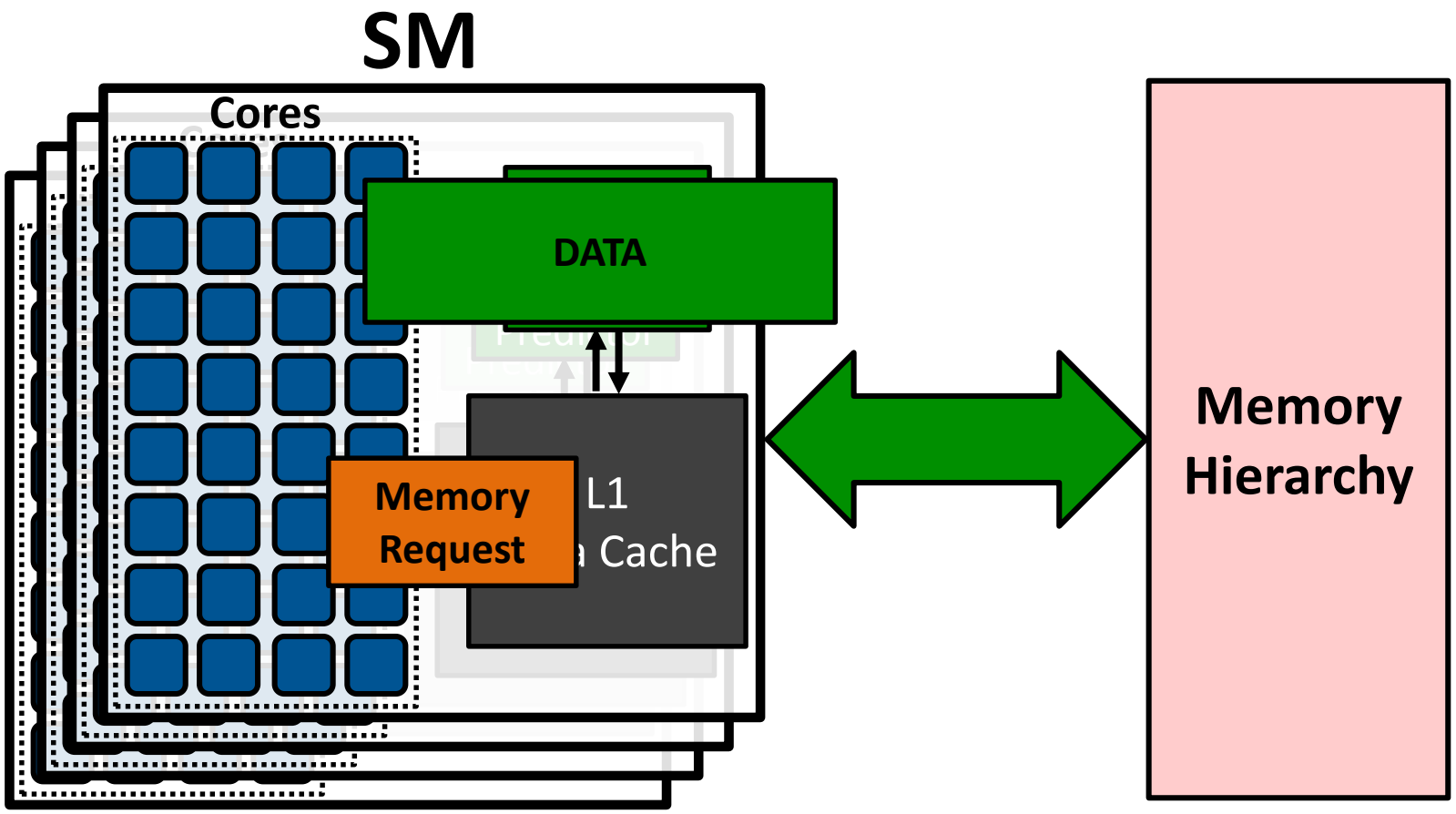


Microarchitecture Integration



All the L1 misses are sent to the memory subsystem

Microarchitecture Integration

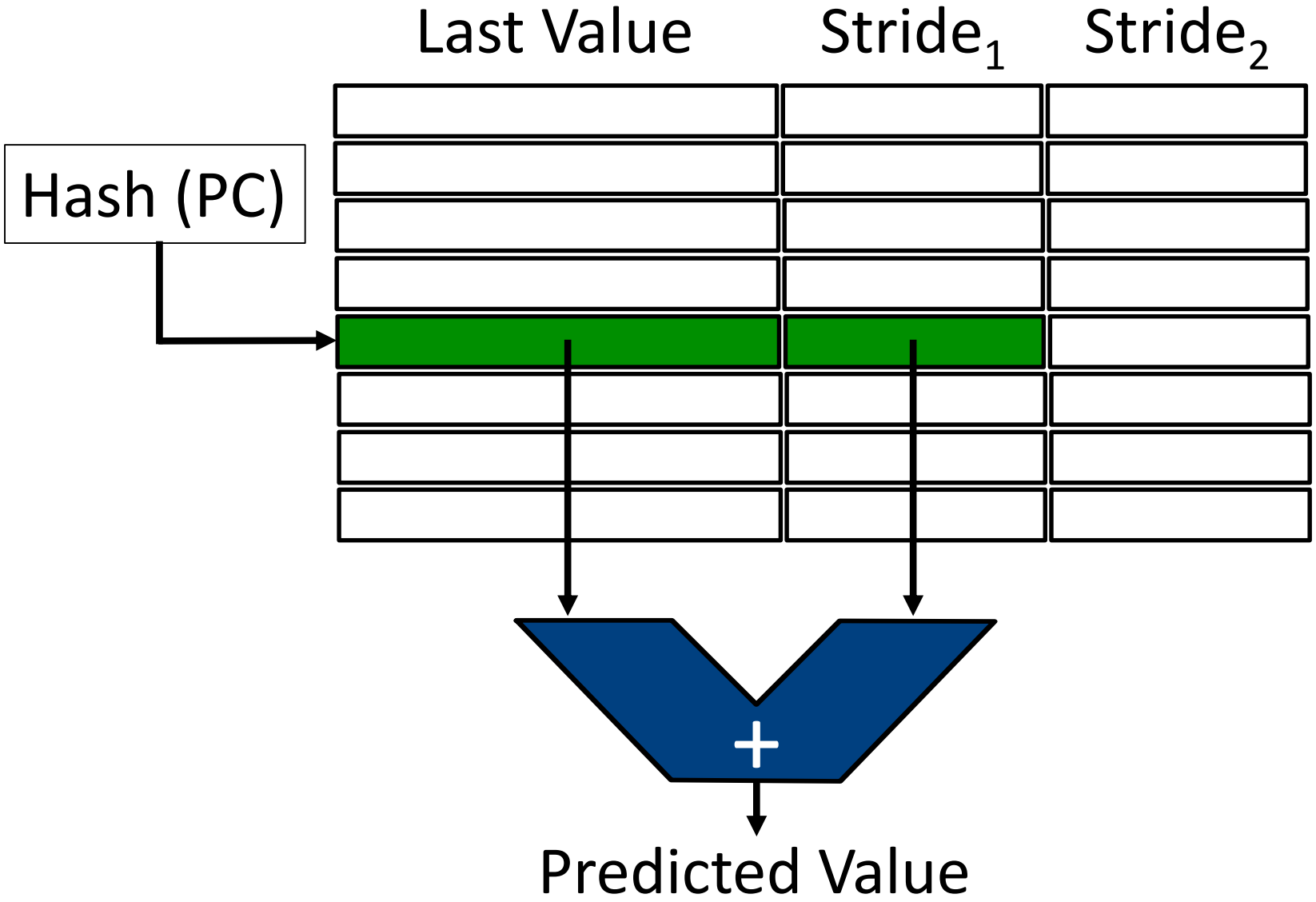


A fraction of the requests will be handled by RFVP

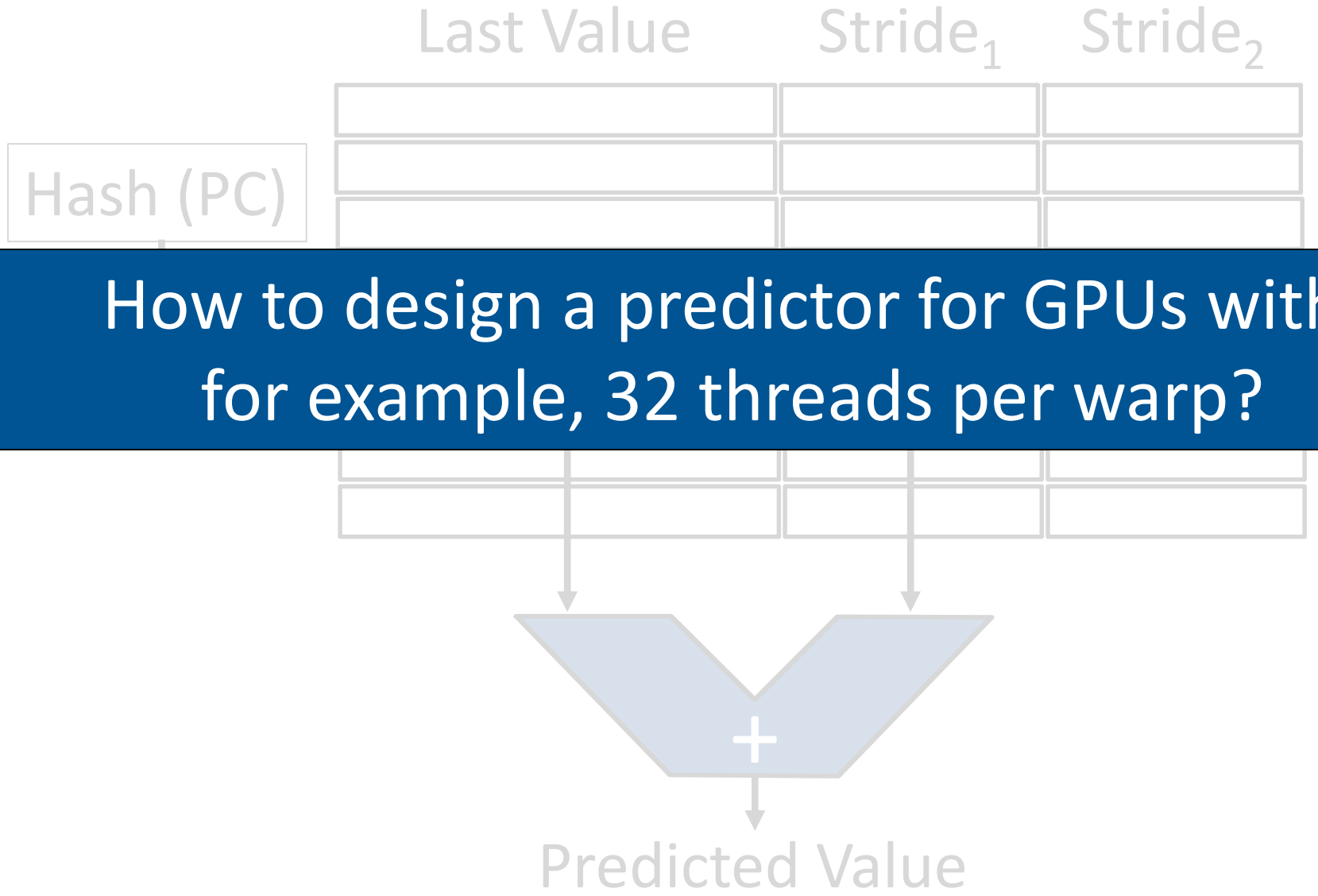
Language and Software Support

- Targeting performance critical loads
  - Only a few critical instructions matter for value prediction
- Providing safety guarantees
  - Programmer annotations and compiler passes
- Drop-rate selection
  - A new **knob** that allows to control quality vs. performance tradeoffs

Base Value Predictor: Two-Delta Stride

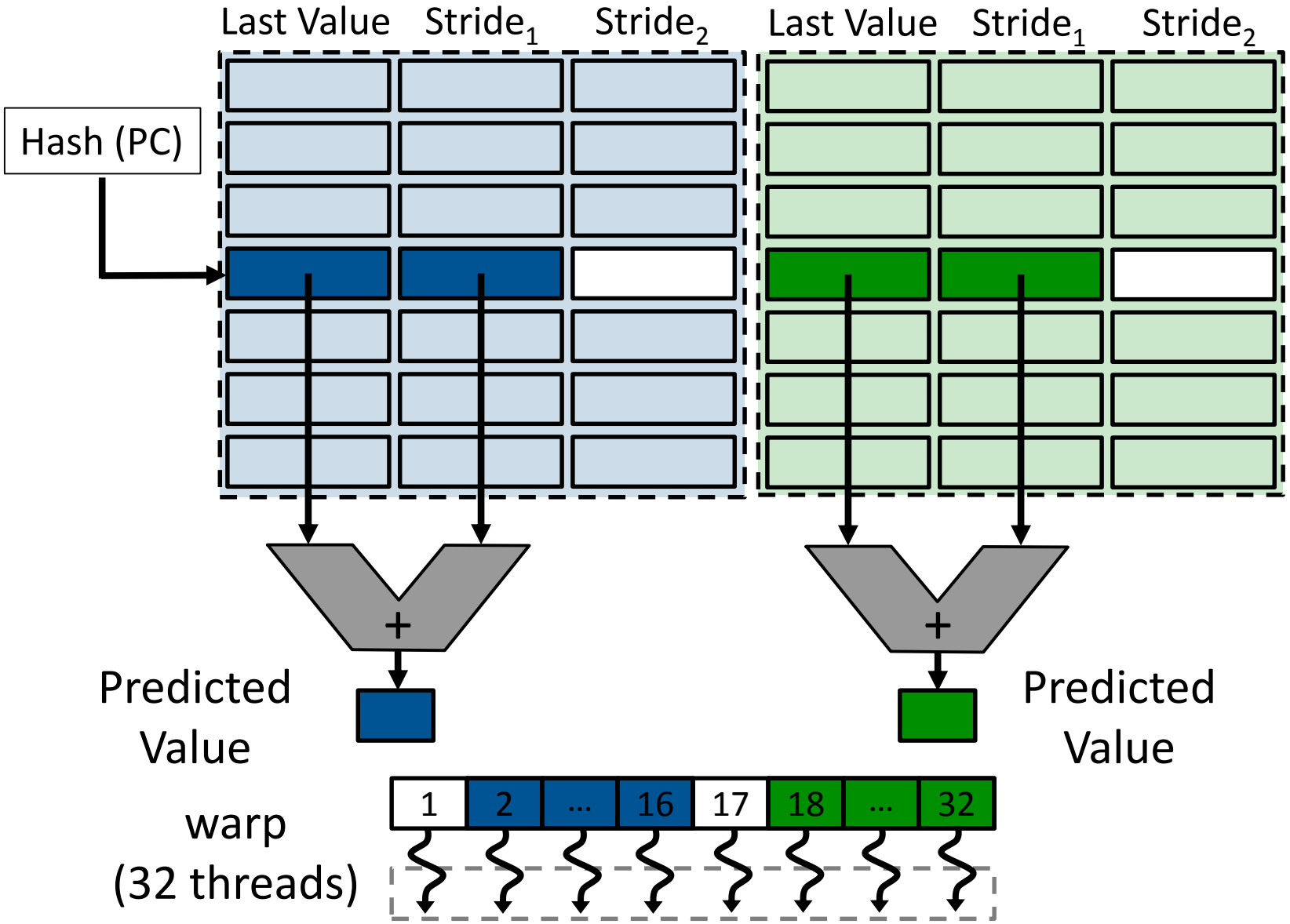


Designing RFVP predictor for GPUs



How to design a predictor for GPUs with, for example, 32 threads per warp?

GPU Predictor Design and Operation



Outline

- Motivation
- Key Idea
- RFVP Design and Operation
- Evaluation
- Conclusion

Methodology

- **Simulator**

GPGPU-Sim simulator (cycle-accurate) ver. 3.1

- **Workloads**

GPU benchmarks from **Rodinia, Nvidia SDK**, and **Mars** benchmark suites

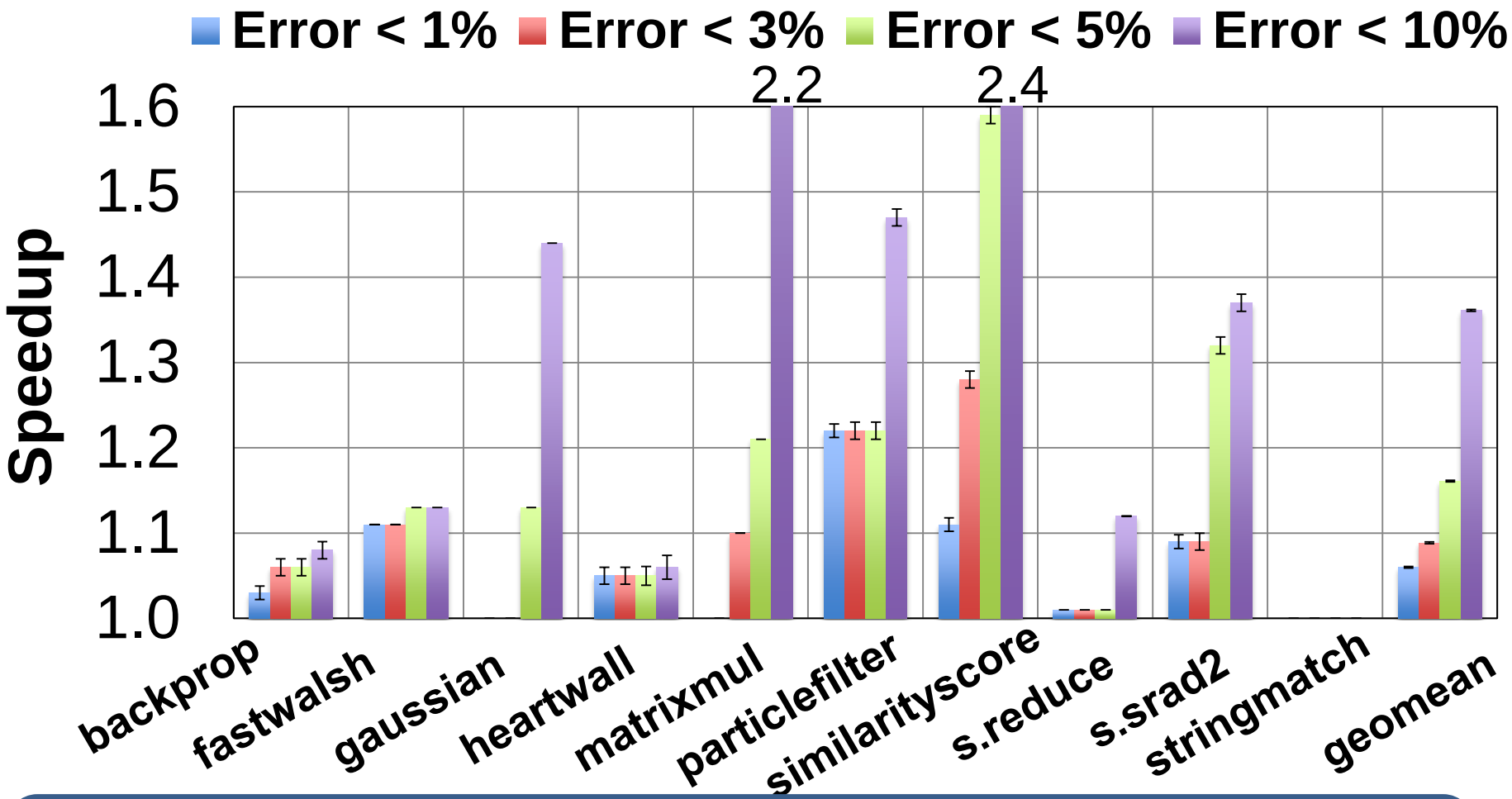
- **System Parameters**

GPU with **15 SMs**, **32 threads/warp**, 6 memory channels,

48 warps/SM, 32KB shared memory, 768KB LLC, GDDR5

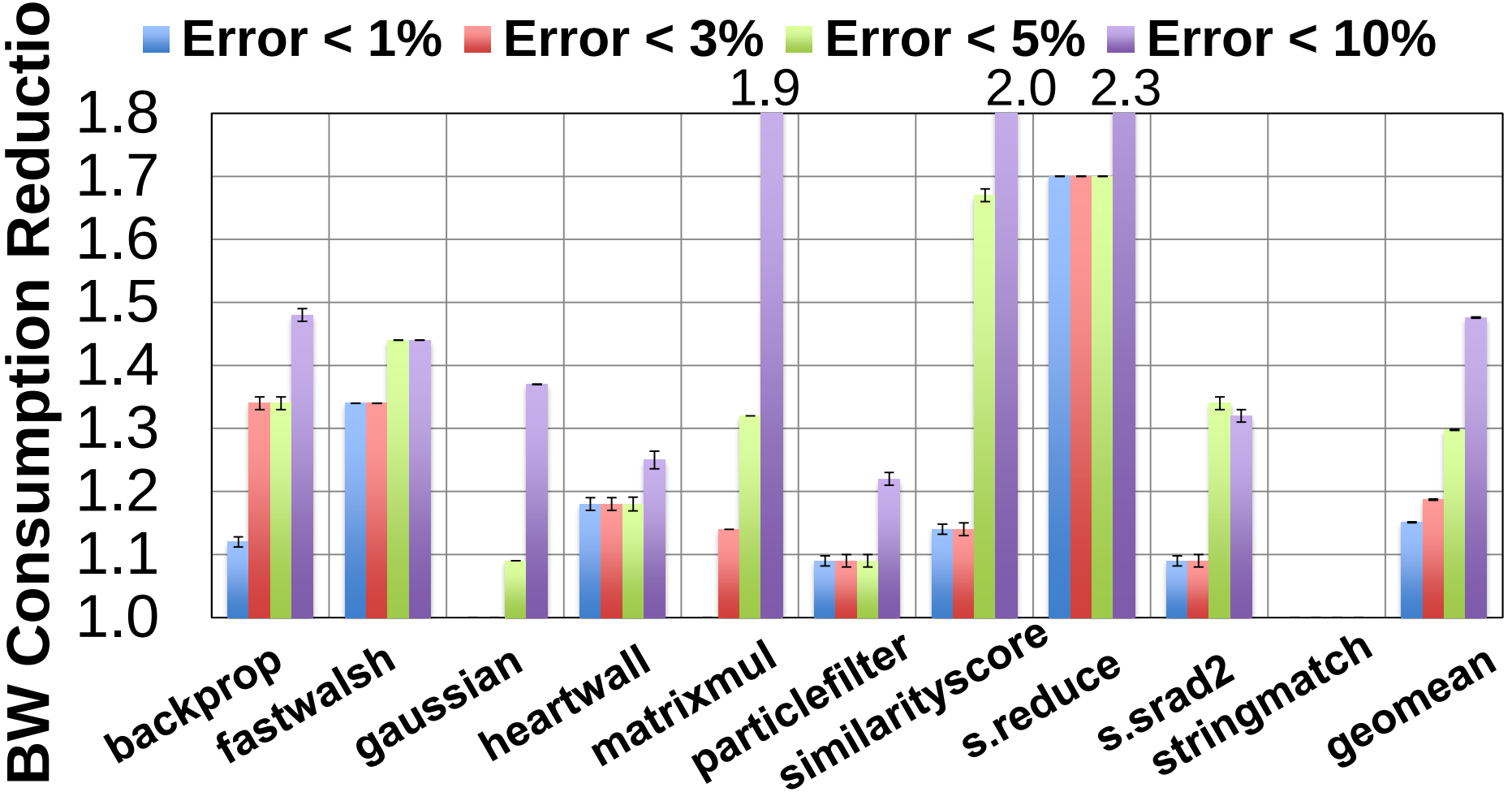
177.4 GB/sec off-chip bandwidth

RFVP Performance



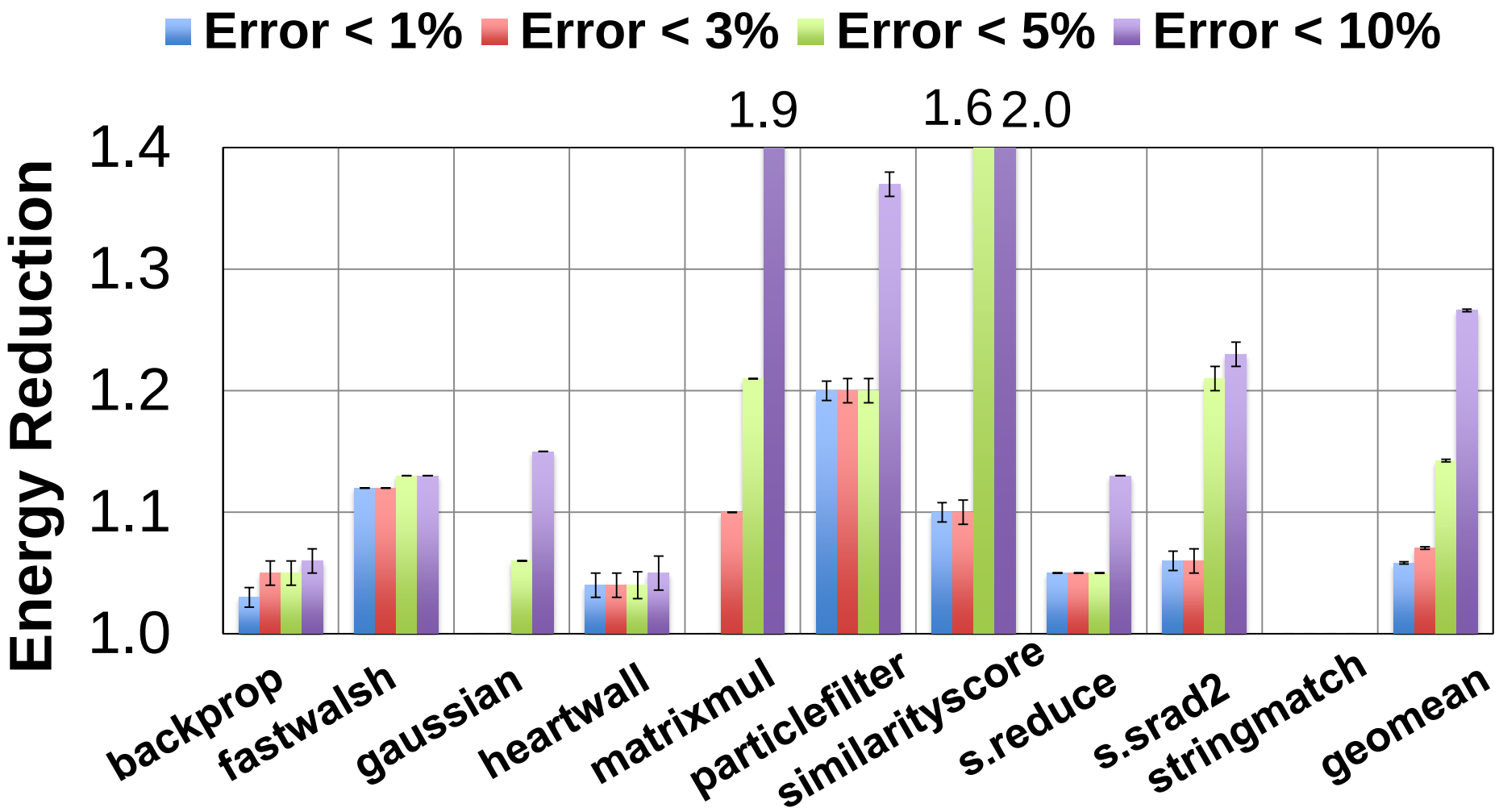
Significant speedup for various acceptable quality rates

RFVP Bandwidth Consumption



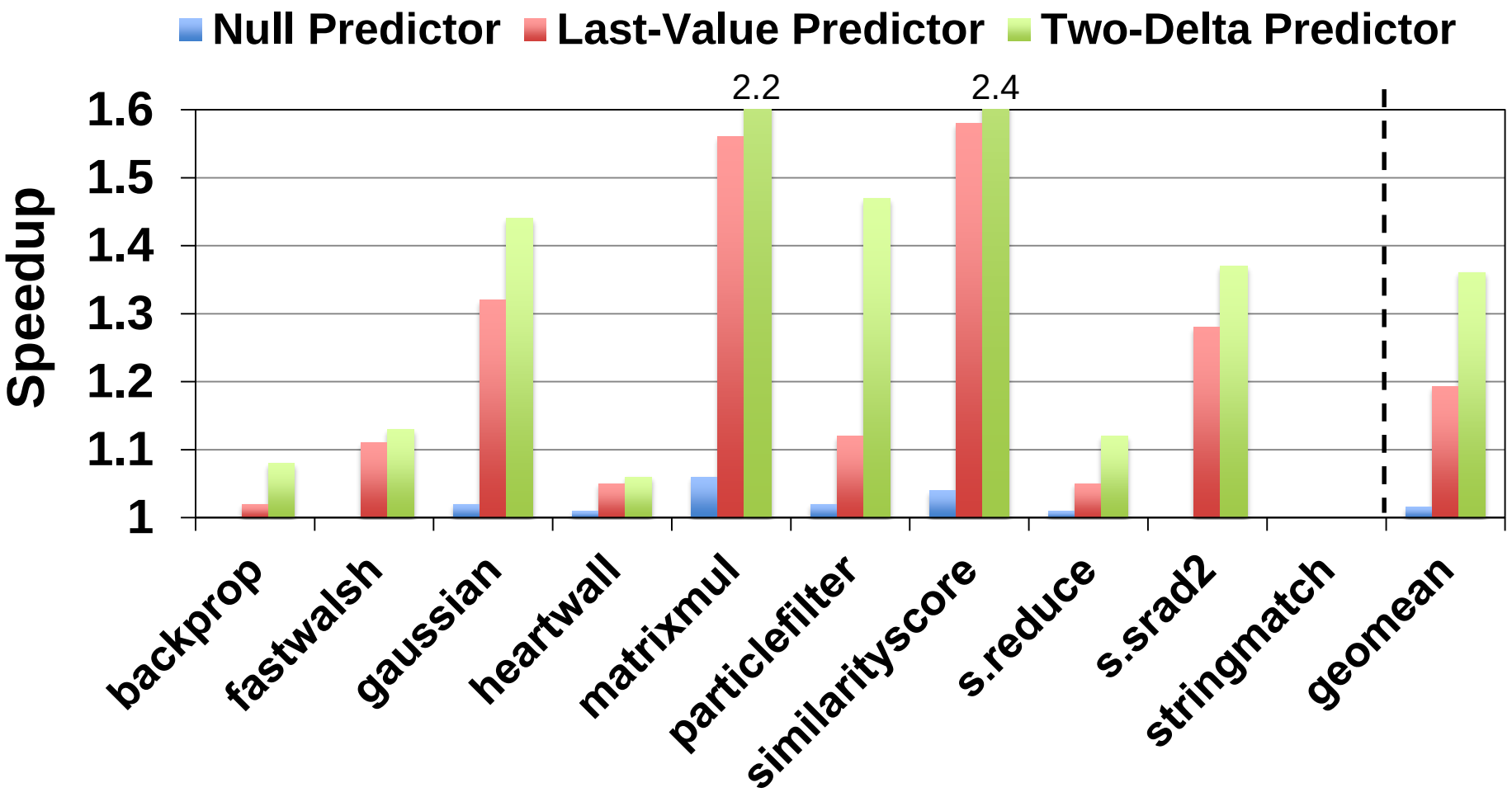
Reduction in consumed bandwidth (up to **1.5X average**)

RFVP Energy Reduction



Reduction in consumed energy (**27% on average**)

Sensitivity to the Value Prediction



Two-Delta predictor was the best option

Other Results and Analyses in the Paper

- Sensitivity to the drop rate (energy and quality)
- Precise vs. imprecise value distributions
- RFVP for memory latency wall
  - CPU performance
  - CPU energy reduction
  - CPU quality vs. performance tradeoff

Conclusion

- **Problem**: Performance of modern GPUs significantly limited by the available off-chip bandwidth
- **Observations**:
  - Many GPU applications are amenable to approximation
  - Data value similarity allows to efficiently predict values of cache misses
- **Key Idea**: Use simple rollback-free value prediction mechanism to avoid accesses to main memory
- **Results:**
  - Higher speedup (**36%** on average) with less than 10% quality loss
  - Lower energy consumption (**27%** on average)

RFVP: Rollback-Free Value Prediction with Safe to Approximate Loads

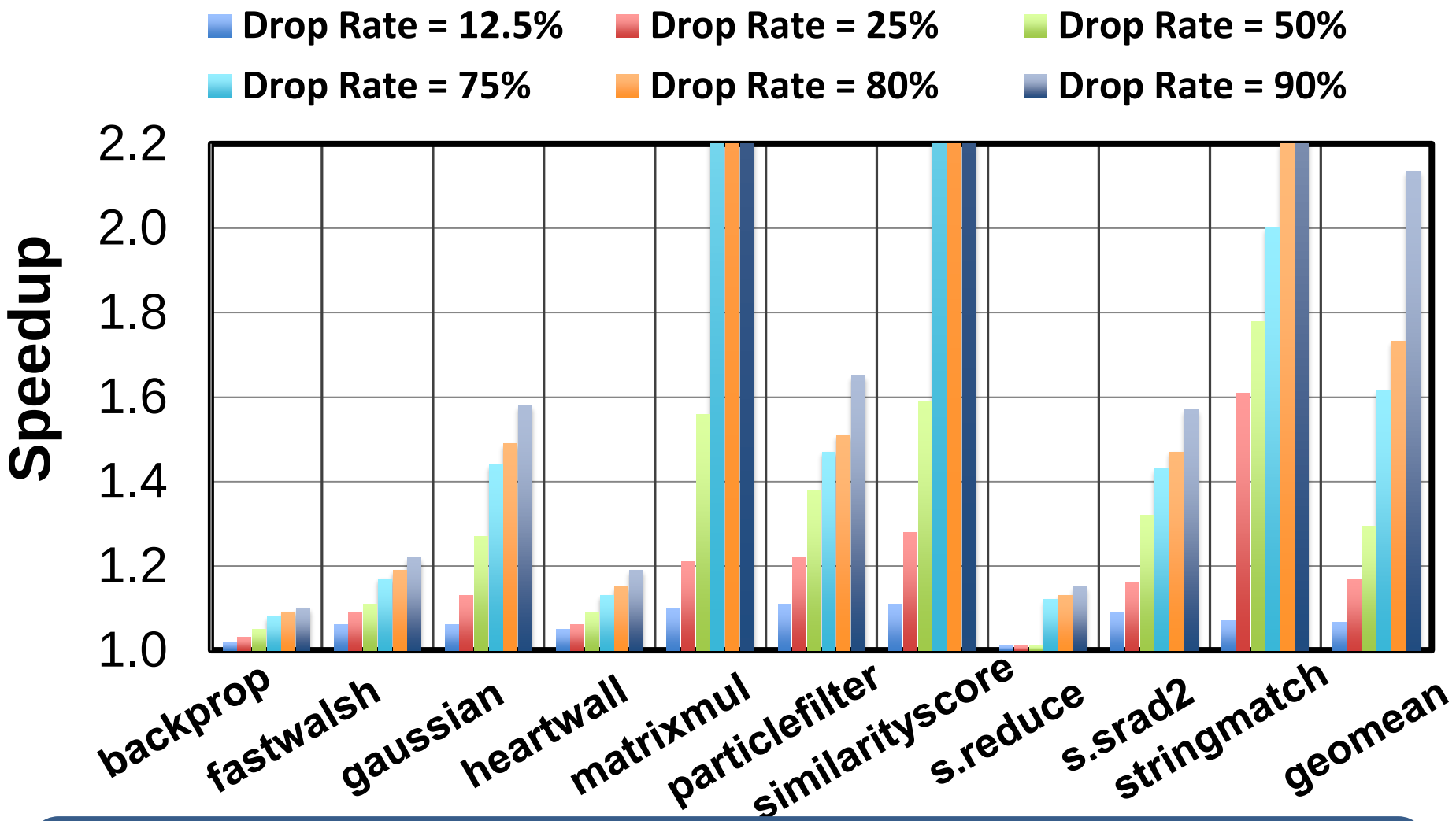
Amir Yazdanbakhsh,
Bradley Thwaites,
Hadi Esmaeilzadeh

Gennady Pekhimenko,
Onur Mutlu,
Todd C. Mowry

Georgia Institute of Technology
Carnegie Mellon University

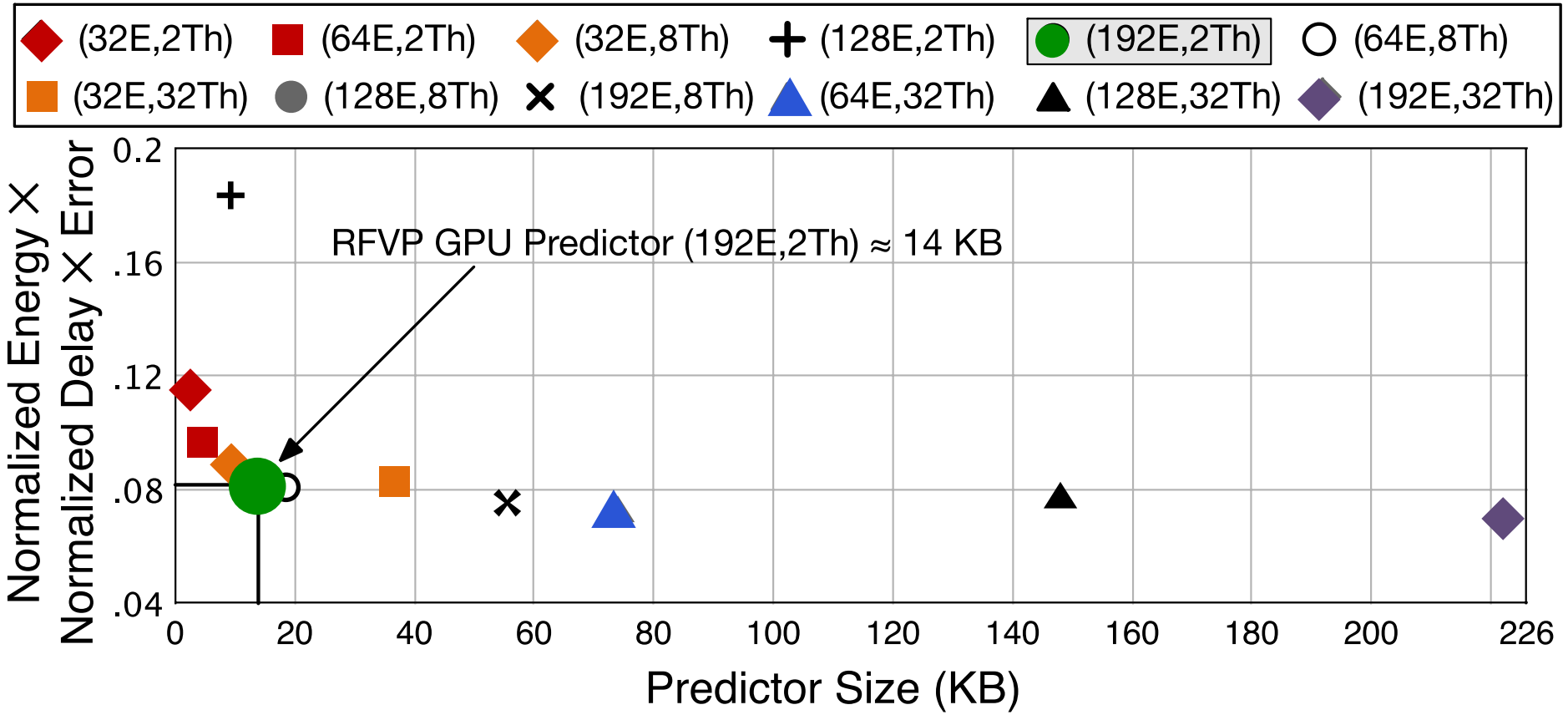
Carnegie Mellon University

Sensitivity to the Drop Rate



Speedup varies significantly with different drop rates

Pareto Analysis



Pareto-optimal is the configuration with 192 entries and 2 independent predictors for 32 threads