Improving the Performance of Object-Oriented Languages with Dynamic Predication of Indirect Jumps

José A. Joao[*‡]
Onur Mutlu[‡*]
Hyesoon Kim[§]
Rishi Agarwal[†‡]
Yale N. Patt[*]

*** HPS Research Group**
University of Texas at Austin

‡ Computer Architecture Group
Microsoft Research

§ College of Computing
Georgia Institute of Technology

† Dept. of Computer Science and Eng.
IIT Kanpur

Motivation

- Polymorphism is a key feature of Object-Oriented Languages
  - Allows modular, extensible, and flexible software design
- Object-Oriented Languages include virtual functions to support polymorphism
  - Dynamically dispatched function calls based on object type
- Virtual functions are usually implemented using indirect jump/call instructions in the ISA
- Other programming constructs are also implemented with indirect jumps/calls: switch statements, jump tables, interface calls

Indirect jumps are becoming more frequent with modern languages

Example from DaCapo *fop* (Java)

- Length
  - LinearCombinationLength
  - PercentLength
  - MixedLength

Length.class:

```
protected void computeValue() {}

public int mvalue() {
    if (!bIsComputed)
        computeValue();
    return millipoints;
}
```

This indirect call is hard to predict

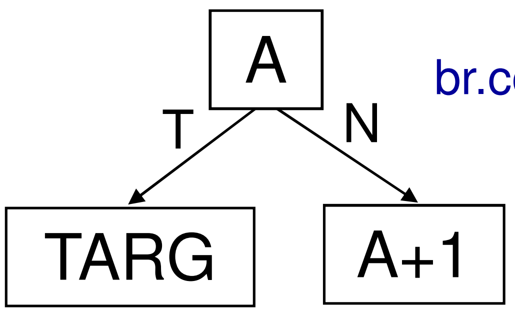
LinearCombinationLength.class:

```
protected void computeValue() {
    // …
    setComputedValue(result);
}
```

PercentLength.class:

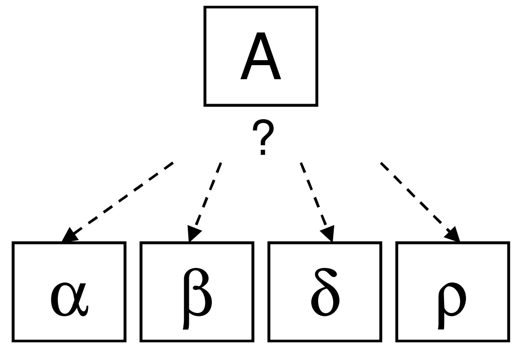
```
protected void computeValue() {
    // …
    setComputedValue(result1);
}
```

Predicting Direct Branches vs. Indirect Jumps



br.cond TARGET

Conditional (Direct) Branch



R1 = MEM[R2]
branch R1

Indirect Jump

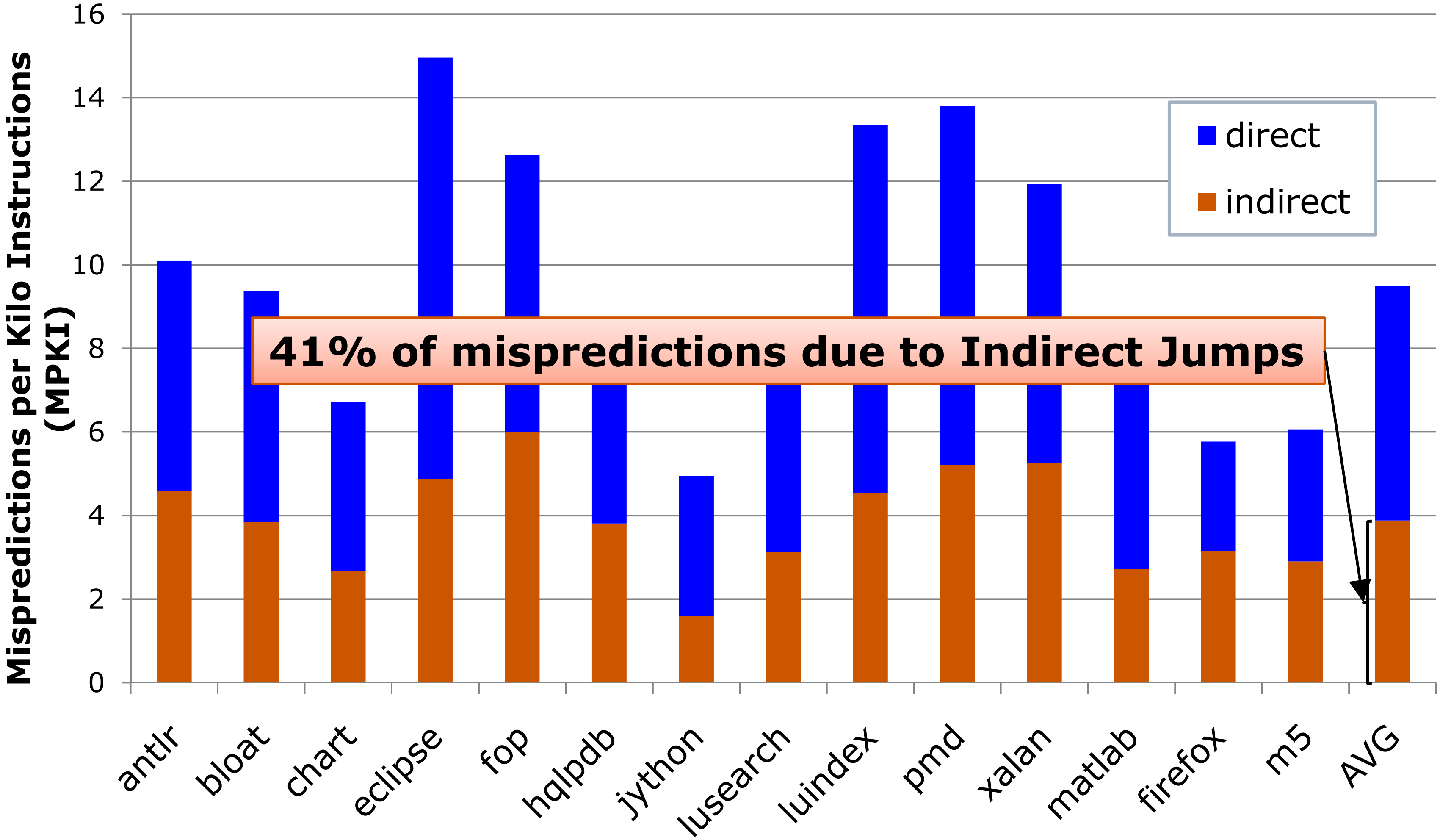
Indirect jumps:

- Multiple target addresses → More difficult to predict than conditional (direct) branches
- Can degrade performance

The Problem

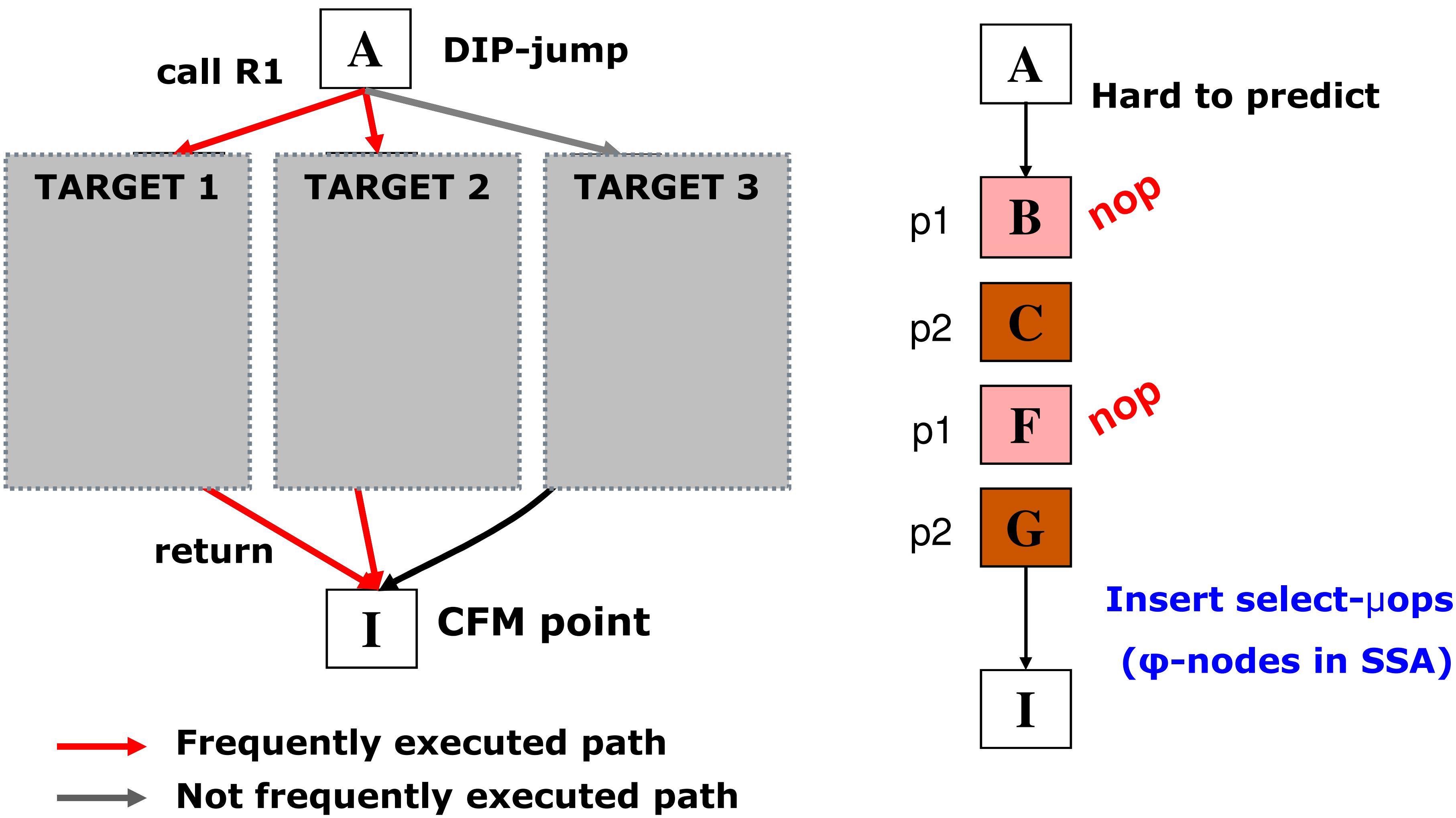
- Most processors predict using the BTB:
  target of indirect jump = target in previous execution
  - Stores only one target per jump
    (already done for conditional branches)
  - Inaccurate
    - Indirect jumps usually switch between multiple targets
    - ~50% of indirect jumps are mispredicted
- Most history-based indirect jump target predictors add large hardware resources for multiple targets

Indirect Jump Mispredictions

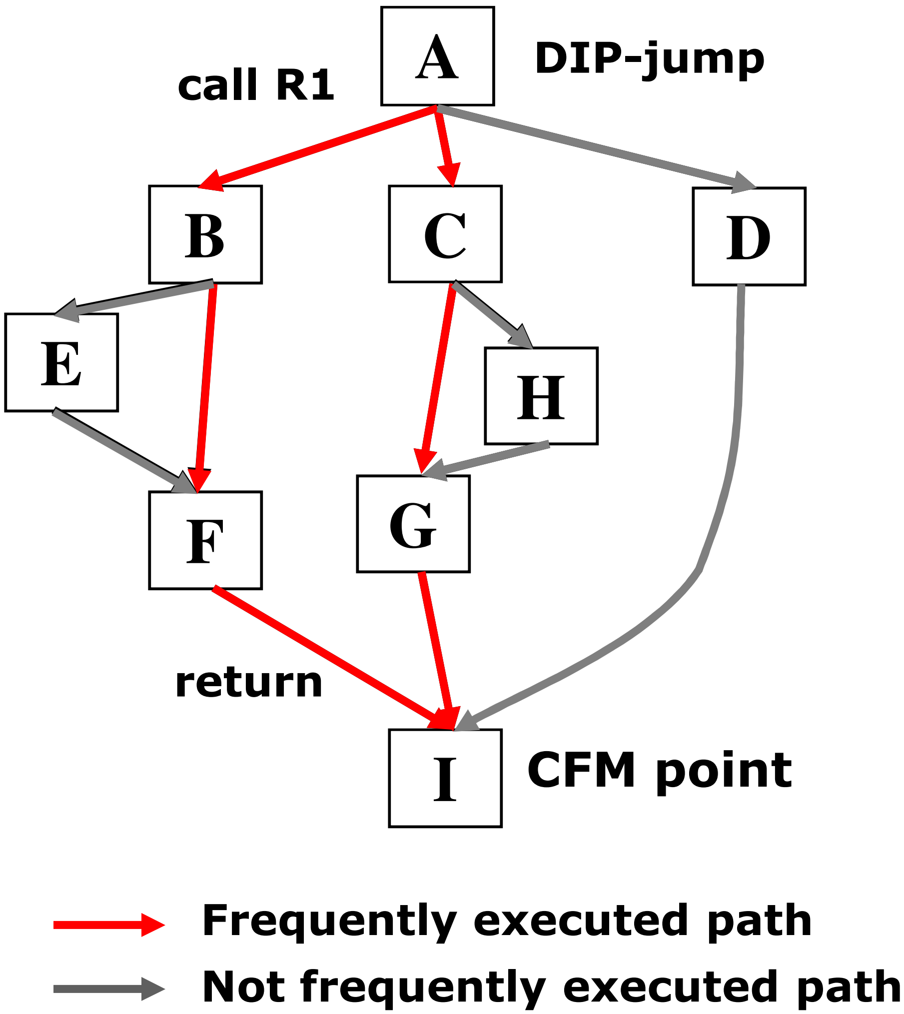


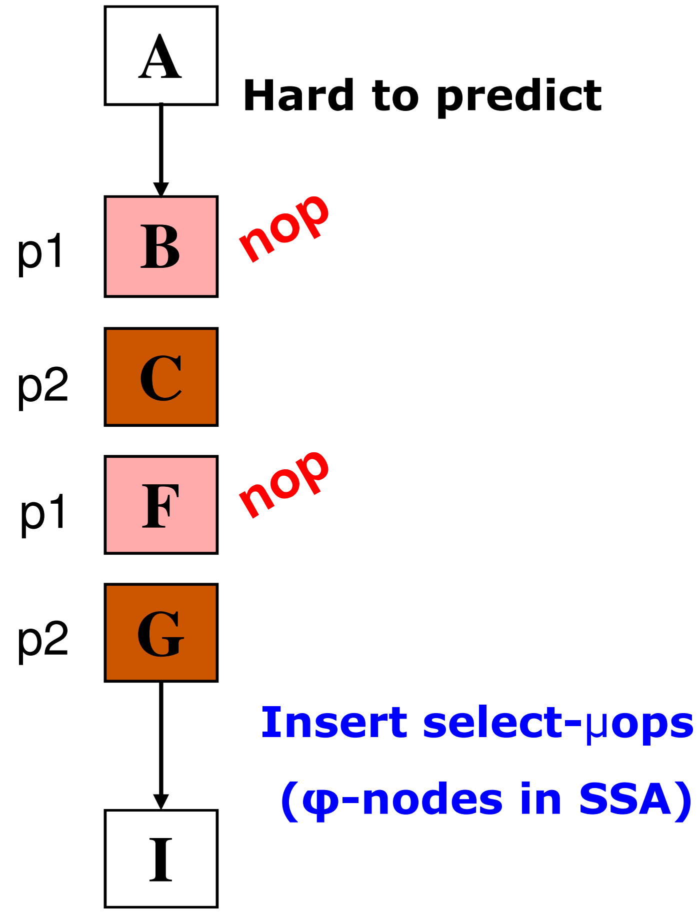
Data from Intel Core2 Duo processor

Dynamic Indirect Jump Predication (DIP)



Dynamic Indirect Jump Predication (DIP)



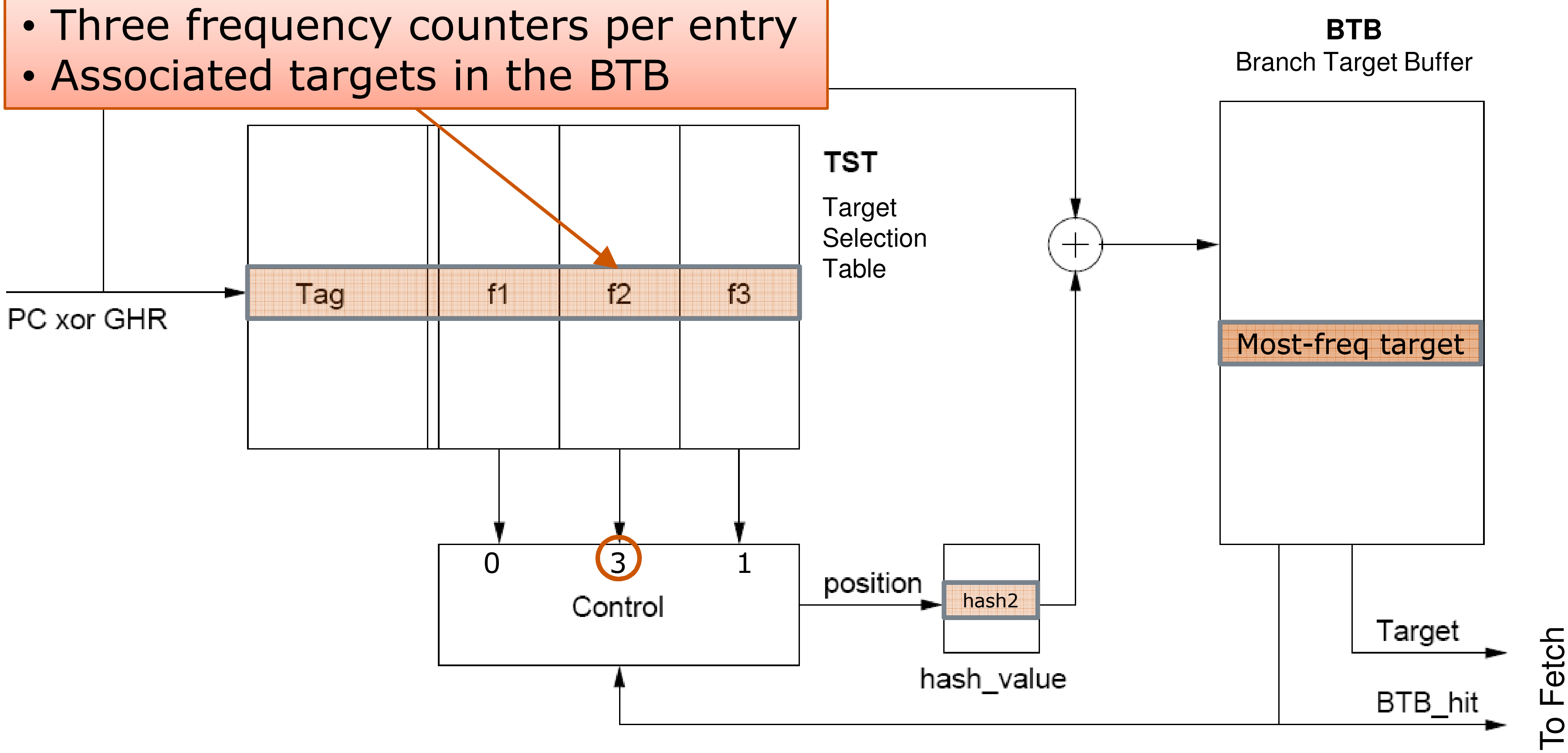


Dynamic Predication of Indirect Jumps

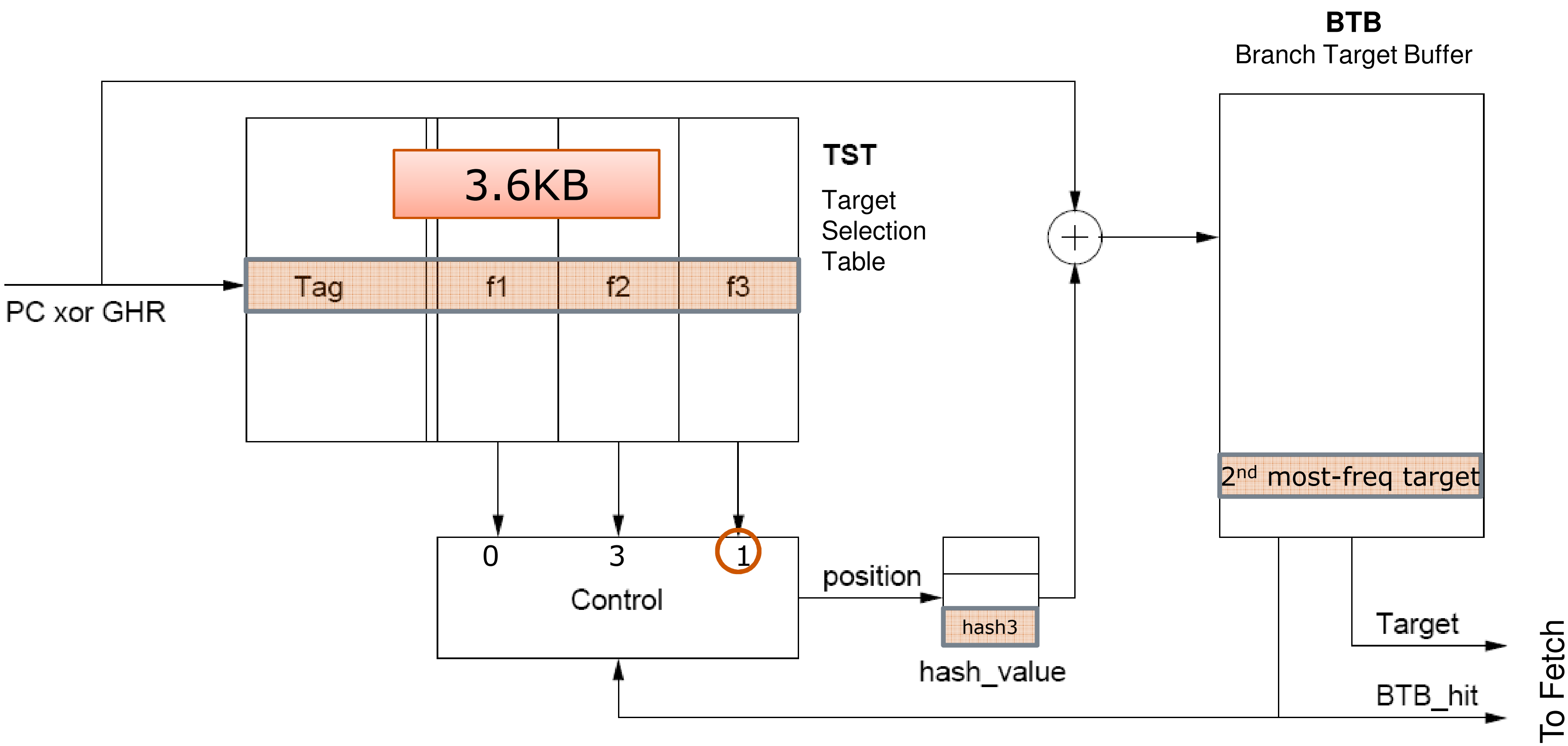
- The compiler uses control-flow analysis and profiling to identify
  - DIP-jumps: highly-mispredicted indirect jumps
  - Control-flow merge (CFM) points
- The microarchitecture decides when and what to predicate dynamically
  - Dynamic target selection

Dynamic Target Selection

- Three frequency counters per entry
- Associated targets in the BTB



Dynamic Target Selection



Additional DIP Entry/Exit Policies

- Single predominant target in the TST
  - TST has more accurate information

    → Override the target prediction

- Nested low confidence DIP-jumps

  → Exit dynamic predication for the earlier jump and re-enter for the later one

- Return instructions inside switch statements
  - Merging address varies with calling site

    → Return CFM points

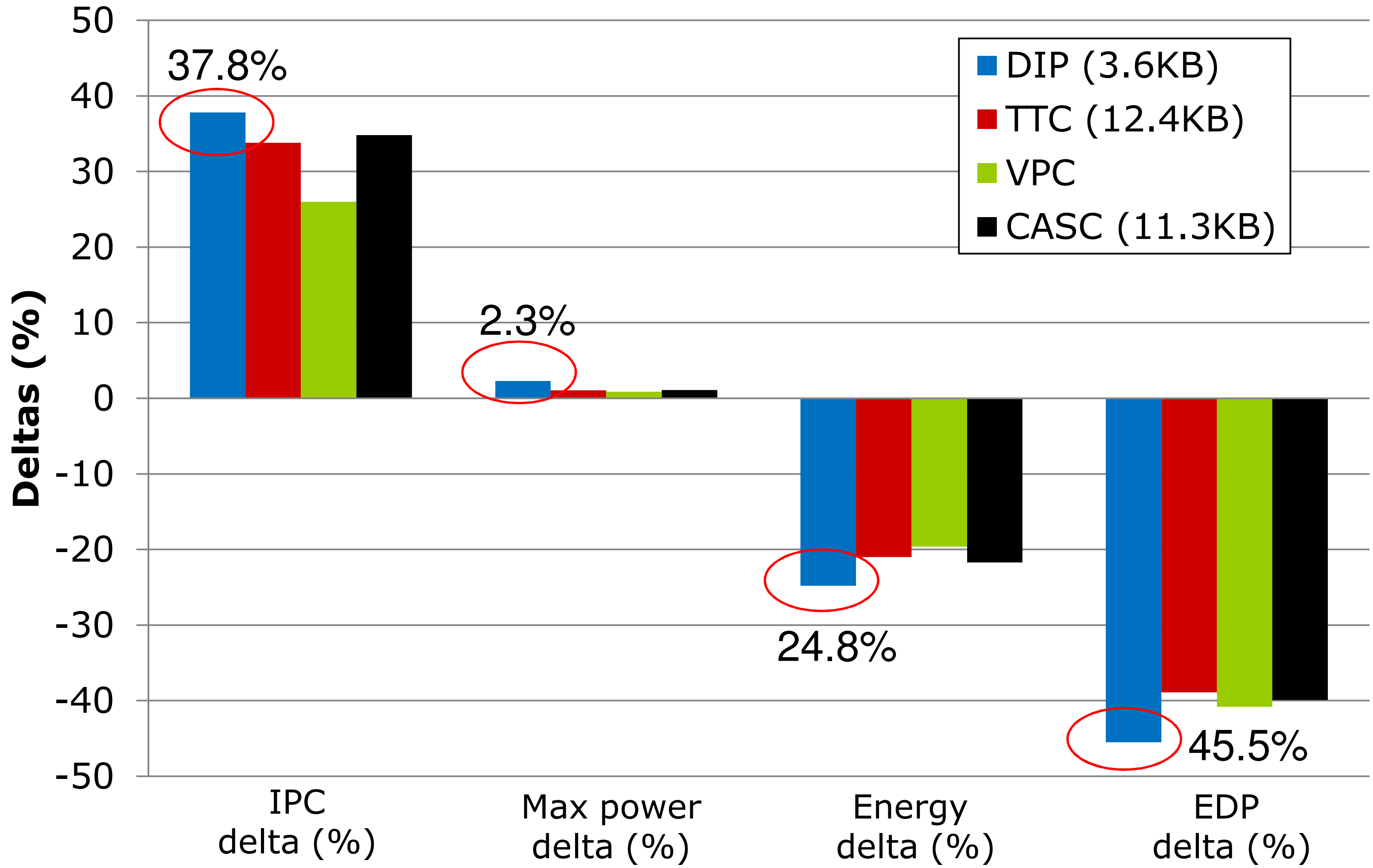
Methodology

- Dynamic profiling tool for DIP-jump and CFM point selection
- Cycle-accurate x86 simulator:
  - Processor configuration
    - 64KB perceptron predictor
    - 4K-entry, 4-way BTB (baseline indirect jump predictor)
    - Minimum 30-cycle branch misprediction penalty
    - 8-wide, 512-entry instruction window
    - 300-cycle minimum memory latency
    - 2KB 12-bit history enhanced JRS confidence estimator
    - 32 predicate registers, 1 CFM register
  - Also less aggressive processor (in paper)
- Benchmarks: DaCapo suite (Java), matlab, m5, perl
  - Also evaluated SPEC CPU 2000 and 2006

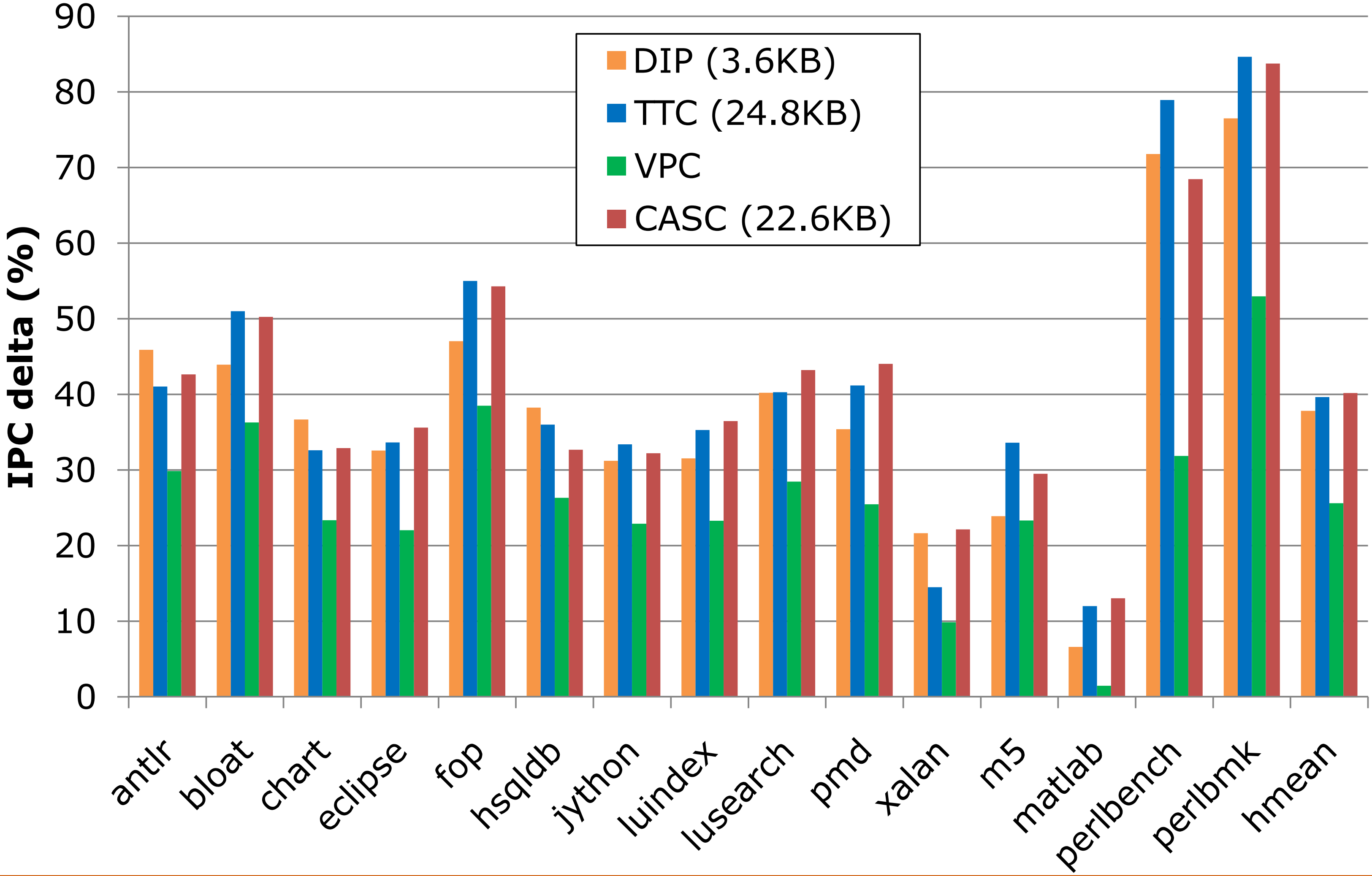
Indirect Jump Predictors

- Tagged Target Cache Predictor (TTC) [P. Chang et al., ISCA 97]
  - 4-way set associative fully-tagged target table
  - Our version does not store easy-to-predict indirect jumps
- Cascaded Predictor [Driesen and Hölzle, MICRO 98, Euro-Par 99]
  - Hybrid predictor with tables of increasing complexity
  - 3-stage predictor performs best
- Virtual Program Counter (VPC) Predictor [Kim et al., ISCA 07]
  - Predicts indirect jumps using the conditional branch predictor
  - Stores multiple targets on the BTB, as our target selection logic does

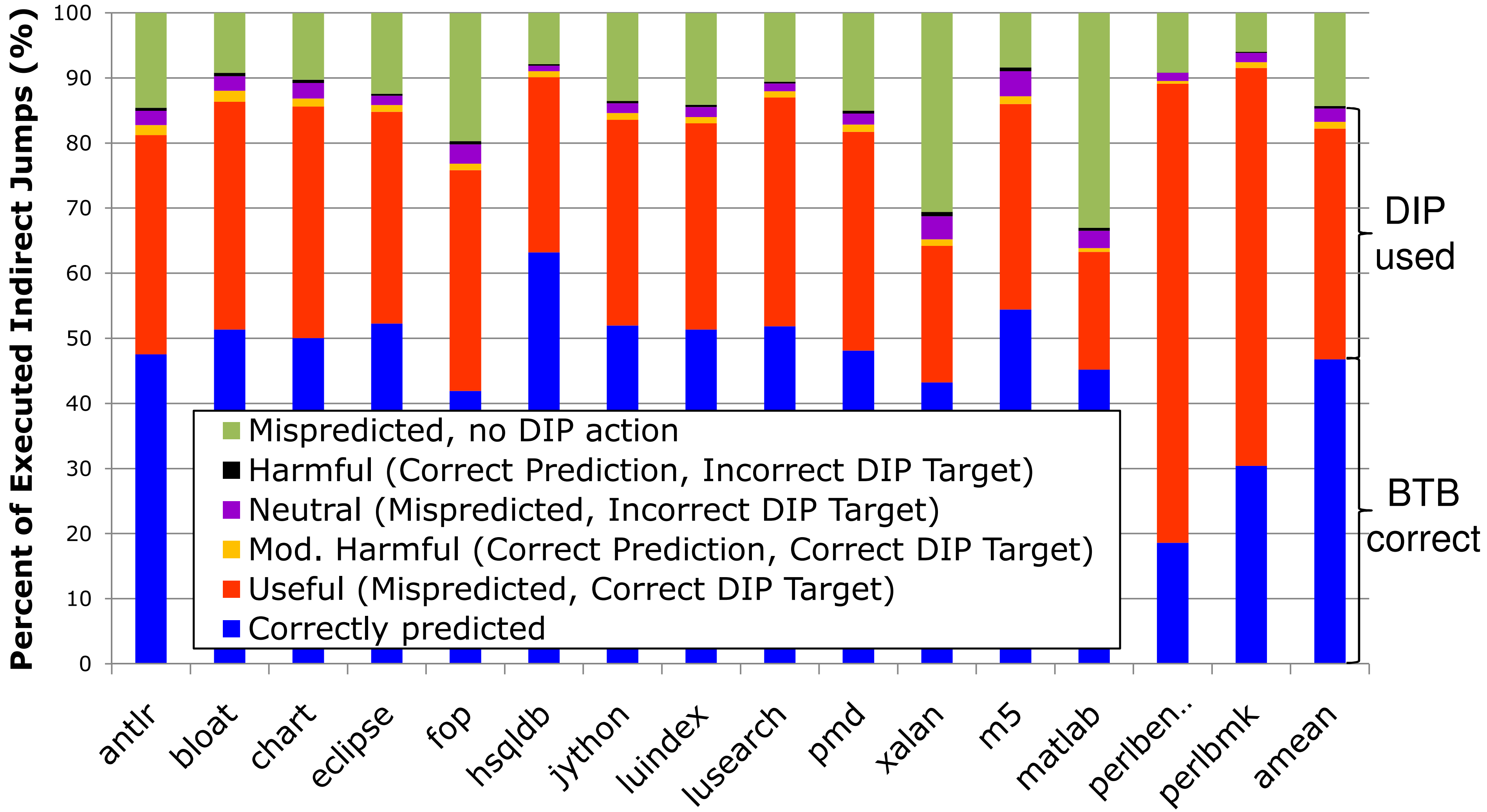
Performance, Power, and Energy



DIP vs. Indirect Jump Predictors



Outcome of Executed Indirect Jumps



Additional Evaluation (in paper)

- Static vs. dynamic target selection policies
- DIP with more than 2 targets → 2 dynamic targets is best
- DIP on top of a baseline with TTC, VPC or Cascaded predictors
- Sensitivity to:
  - Processor configuration
  - BTB size
  - TST size and structure
- More benchmarks (SPEC CPU 2000 and 2006)

Conclusion

- Object-oriented languages use more indirect jumps
  - Indirect jumps are hard to predict and have already become an important performance limiter

- We propose DIP, a cooperative hardware-software technique
  - Improves performance by 37.8%
  - Reduces energy by 24.8%
  - Provides better performance and energy-efficiency than three indirect jump predictors
  - Incurs low hardware cost (3.6KB) if dynamic predication is already used for conditional branches
  - Can be an enabler encouraging developers to use object-oriented programming

Thank You!

Questions?