Context Threading: A flexible and efficient dispatch technique for virtual machine interpreters

Marc Berndl

Benjamin Vitale

Mathew Zaleski

Angela Demke Brown

Research supported by IBM CAS, NSERC, CITO

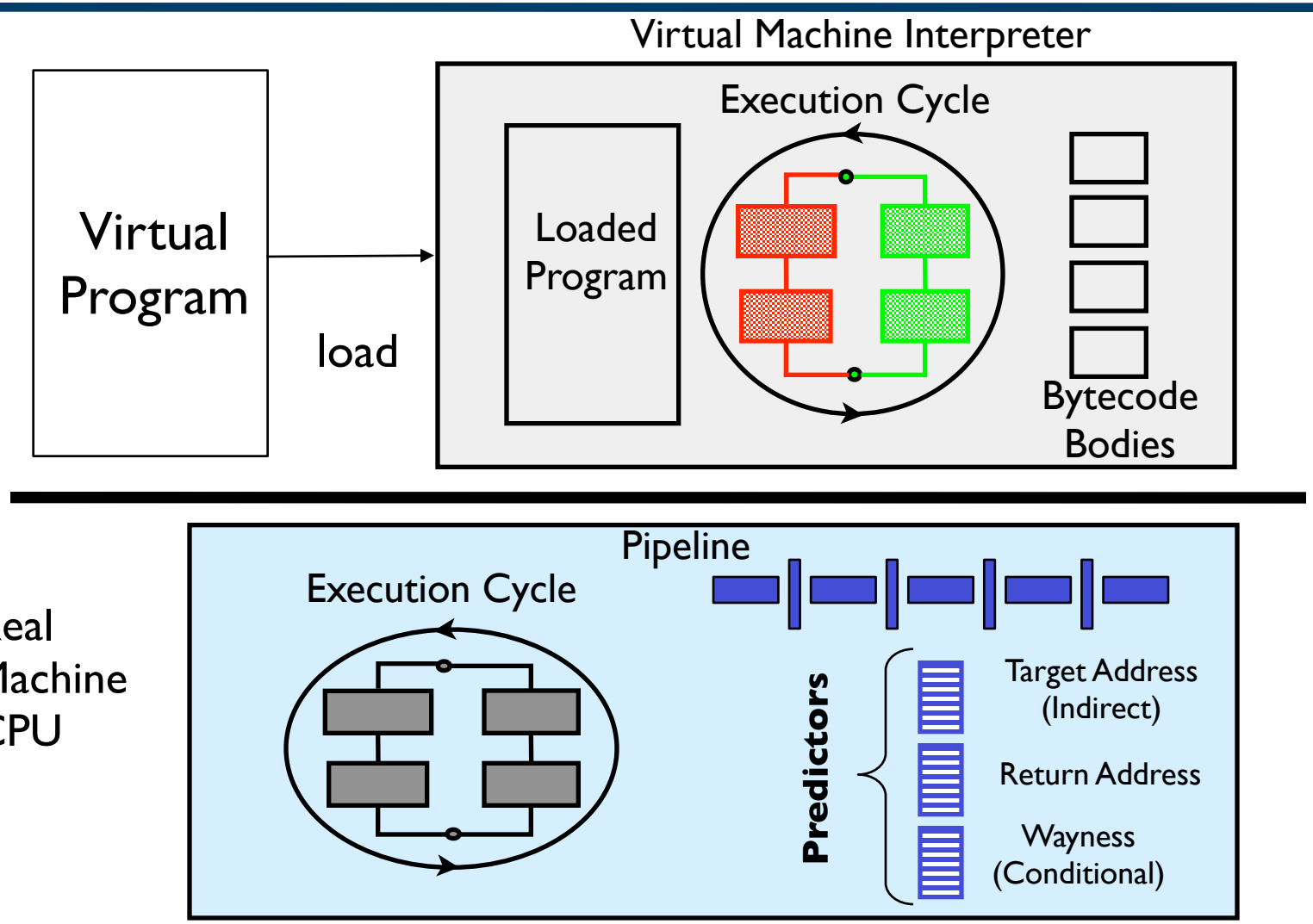
Interpreter performance

- Why not just in time (JIT) compile?
  - High performance JVMs still interpret
  - People use interpreted languages that don't yet have JITs
  - They still want performance!
- 30-40% of execution time is due to stalls caused by branch misprediction.
- Our technique eliminates 95% of branch mispredictions

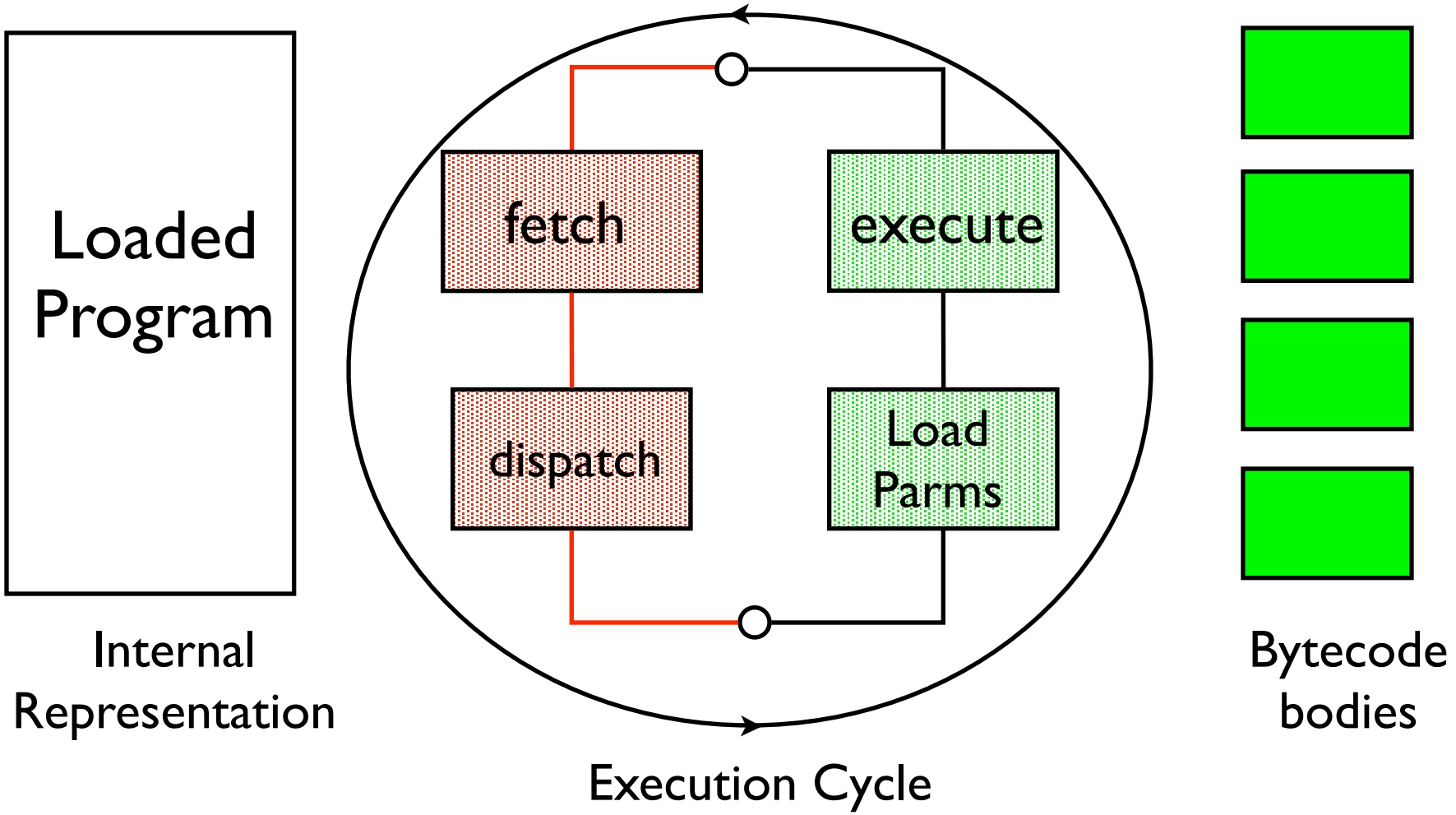
Overview

- ✔ Motivation
- Background: The Context Problem
- Existing Solutions
- Our Approach
- Inlining
- Results

A Tale of Two Machines



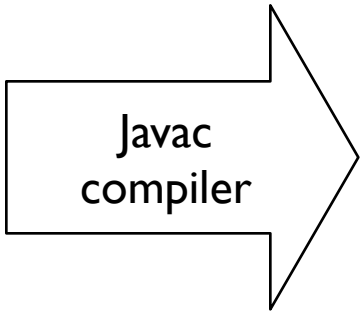
Interpreter



Running Java Example

Java Source

```
void foo(){
  int i=1;
  do{
    i+=i;
    } while(i<64);
  }
```



Java Bytecode

```
0:   iconst_0
1:   istore_1
2:   iload_1
3:   iload_1
4:   iadd
5:   istore_1
6:   iload_1
7:   bipush 64
9:   if_icmplt 2
12:  return
```

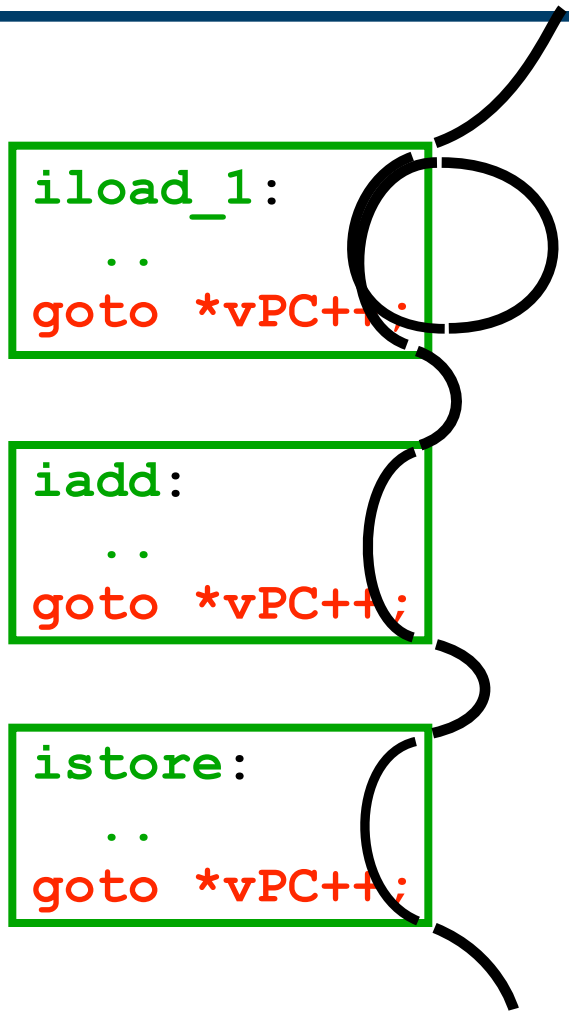
Switched Interpreter

```
while(1){
 opcode = *vPC++;
 switch(opcode){

   case iload_1:
        ..
     break;

   case iadd:
        ..
     break;

  //and many more..

   }
};
```

- slow. burdened by switch and loop overhead

"Threading" Dispatch

```
0:    iconst_0
1:    istore_1
2:    iload_1
3:    iload_1
4:    iadd
5:    istore_1
6:    iload_1
7:    bipush 64
9:    if_icmplt 2
12:   return
```



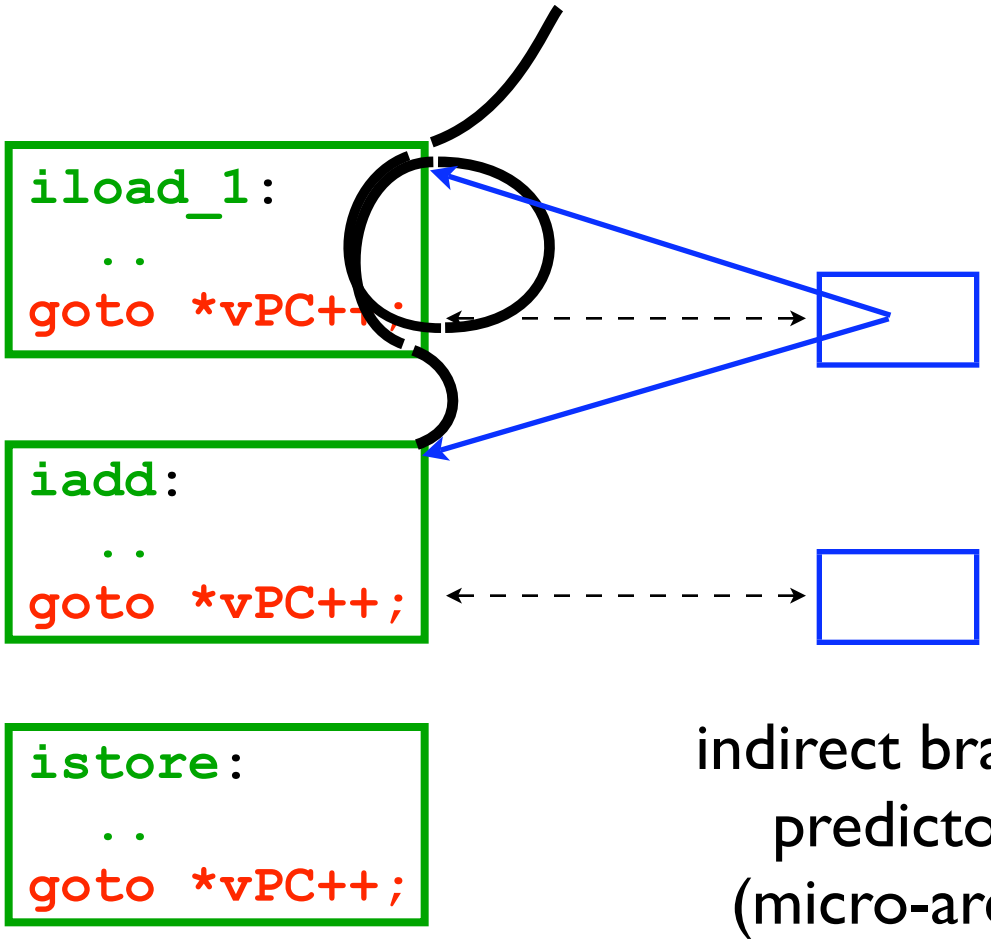
execution of virtual program "threads" through bodies

(as in needle & thread)

- No switch overhead. Data driven indirect branch.

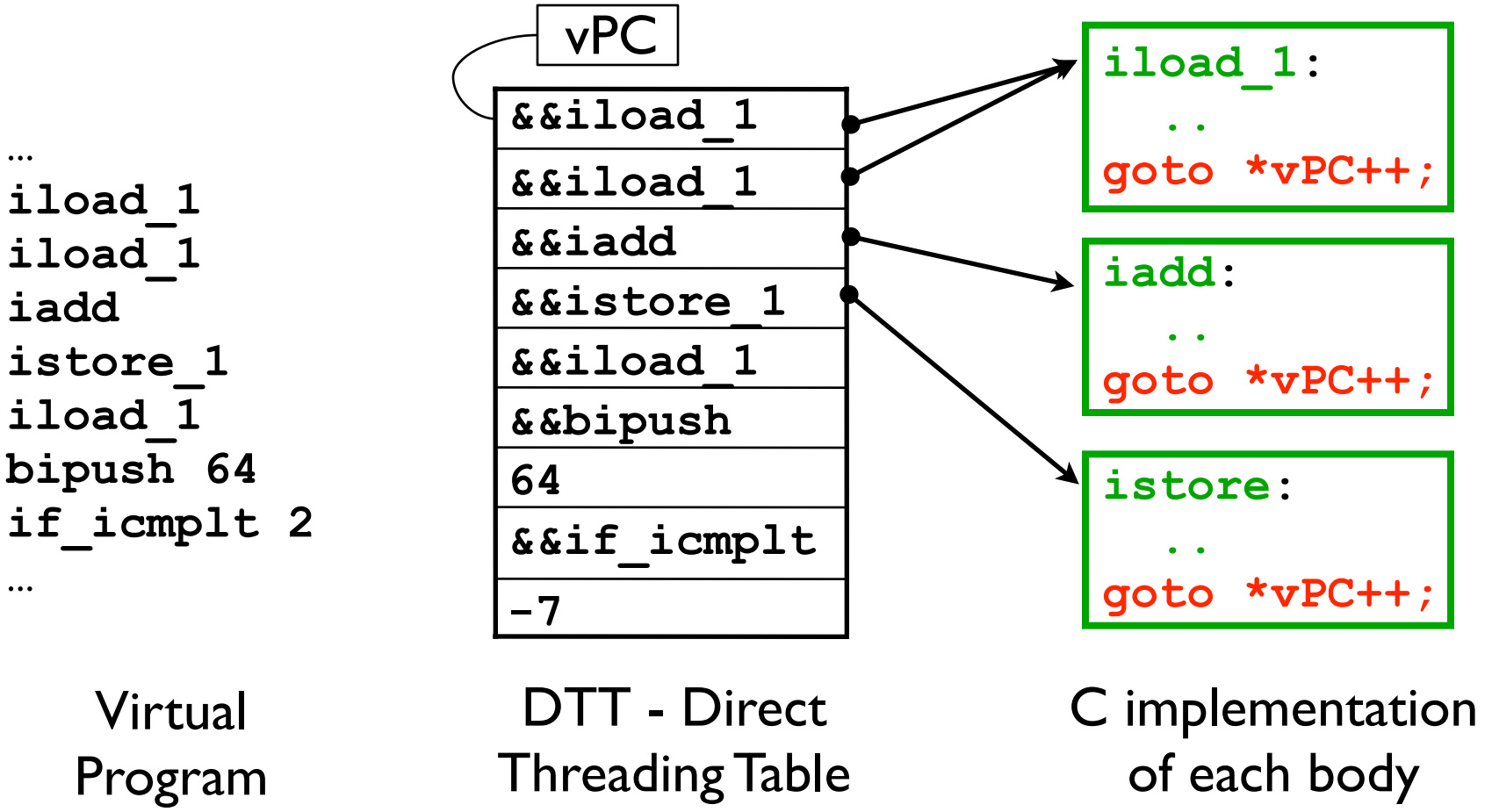
Context Problem

```
0:    iconst_0
1:    istore_1
2:    iload_1
3:    iload_1
4:    iadd
5:    istore_1
6:    iload_1
7:    bipush 64
9:    if_icmplt 2
12:   return
```



- Data driven indirect branches hard to predict

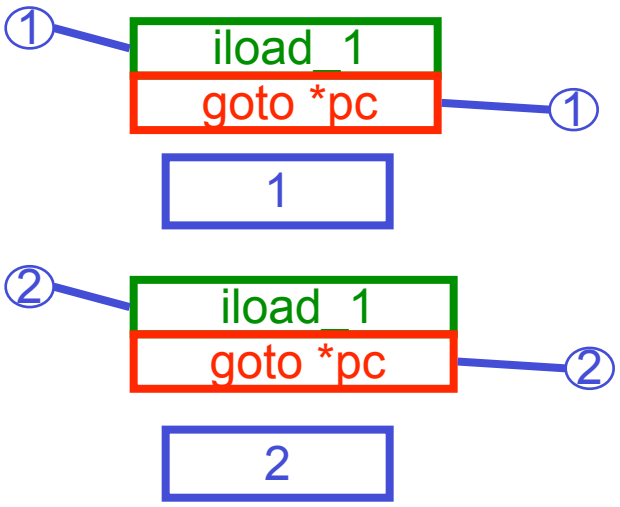
Direct Threaded Interpreter



- Target of computed goto is data-driven

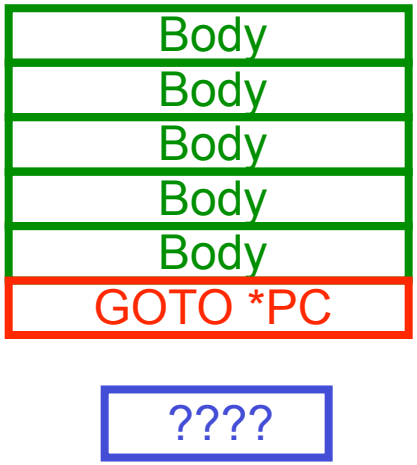
Existing Solutions

Replicate



Ertl & Gregg:
Bodies and Dispatch Replicated

Super Instruction



Piumarta & Ricardi :
Bodies Replicated

- Limited to relocatable virtual instructions

Overview

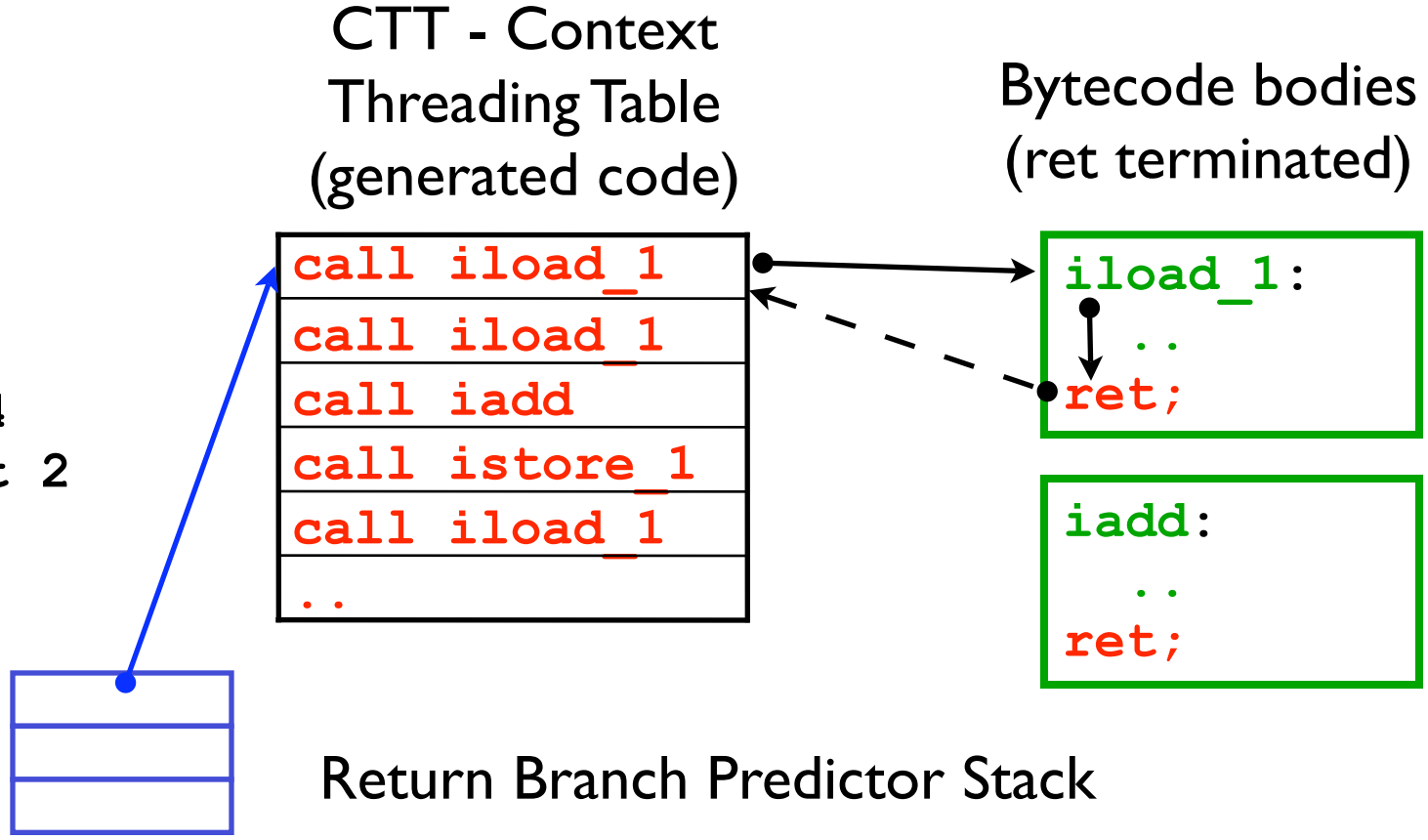
- ✔ Motivation
- ✔ Background: The Context Problem
- ✔ Existing Solutions
- Our Approach
- Inlining
- Results

Key Observation

- Virtual and native control flow similar
  - Linear or straight-line code
  - Conditional branches
  - Calls and Returns
  - Indirect branches
- Hardware has predictors for each type
  - Direct uses indirect branch for everything!
- Solution: Leverage hardware predictors

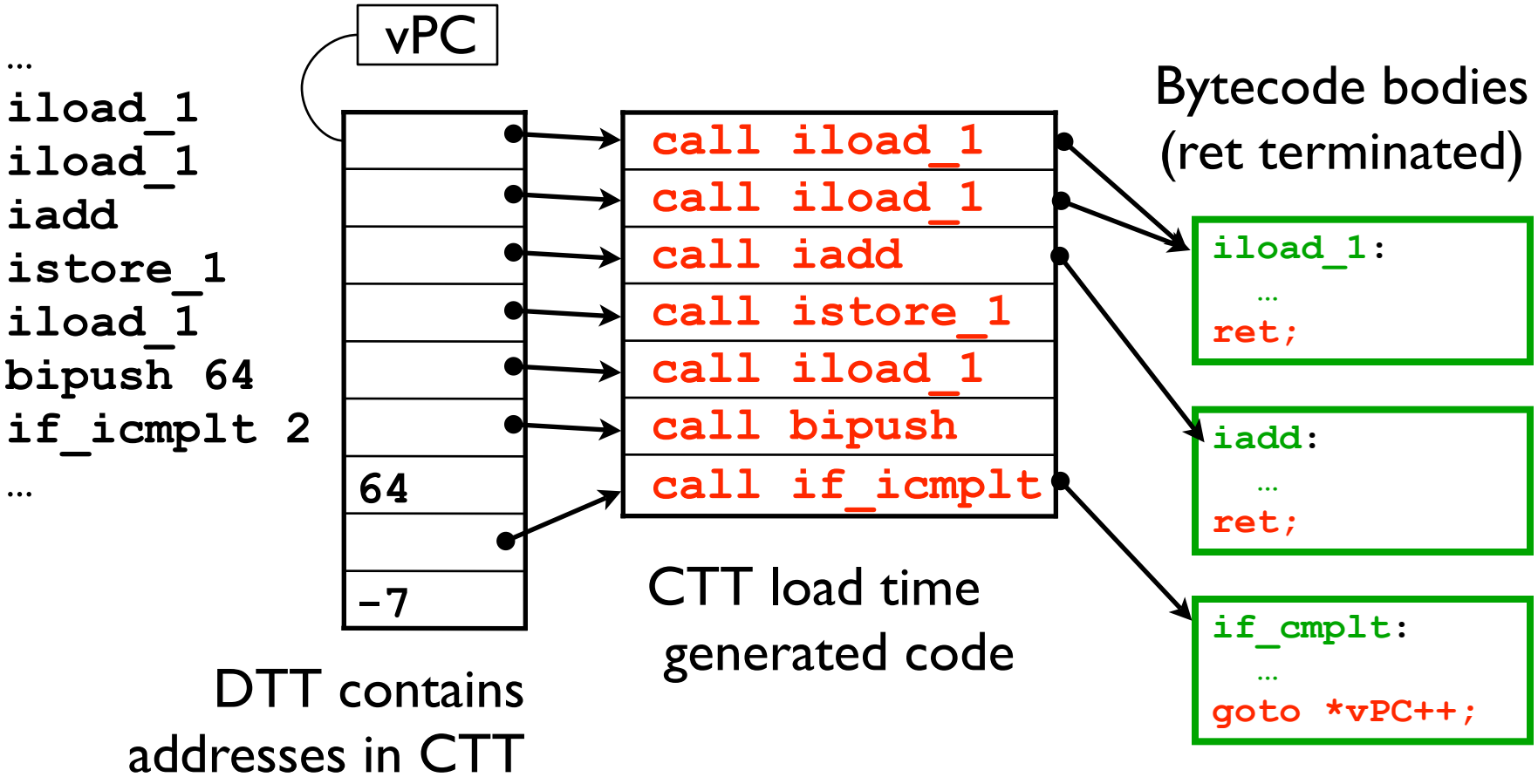
Essence of our Solution

```
…
iload_1
iload_1
iadd
istore_1
iload_1
bipush 64
if_icmplt 2
…
```



- Package bodies as subroutines and call them

Subroutine Threading



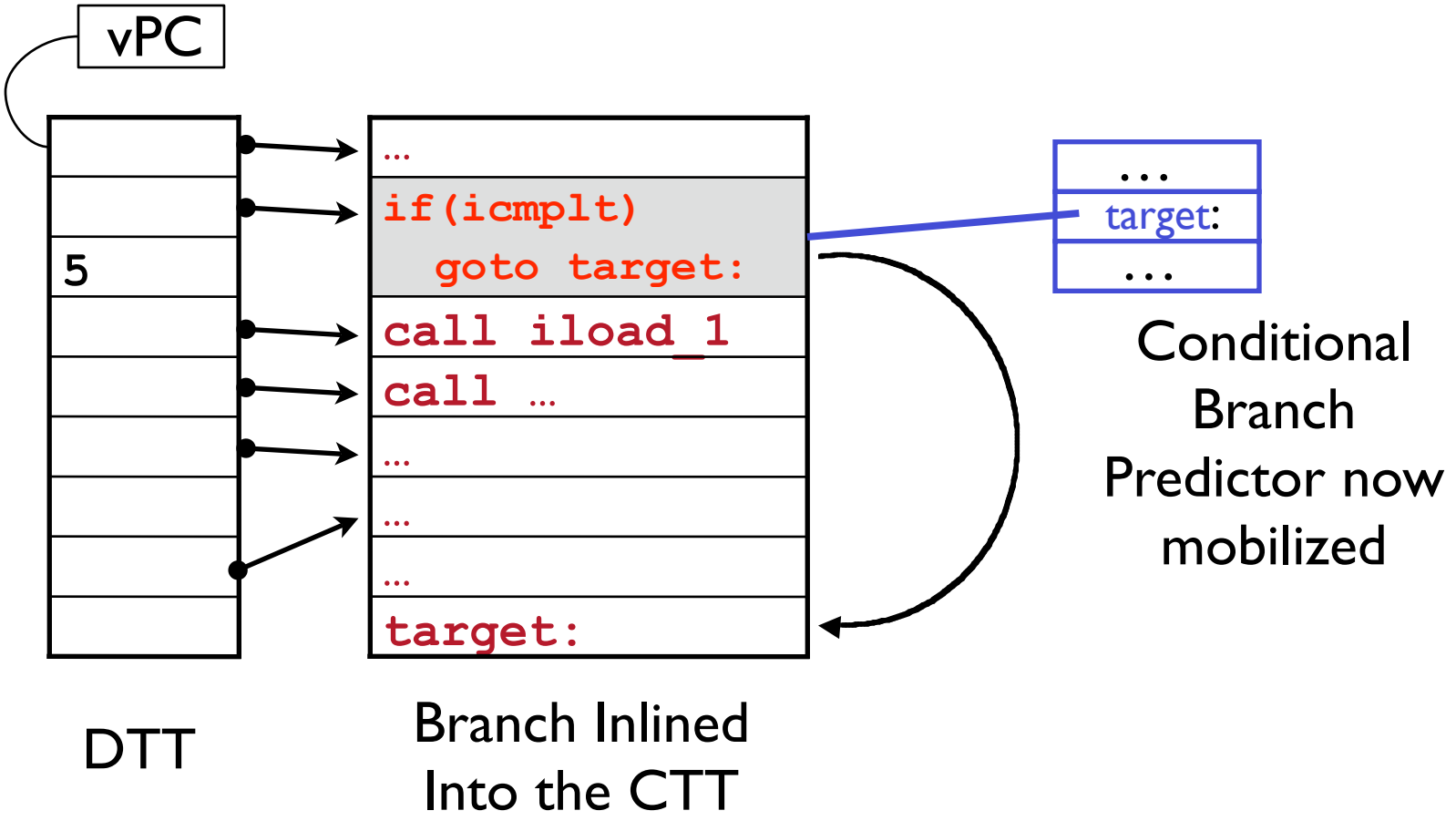
- virtual branch instructions as before

The Context Threading Table

- A sequence of generated call instructions
- Good alignment of virtual and hardware control flow for straight-line code.

- Can virtual branches go into the CTT?

Specialized Branch Inlining



- Inlining conditional branches provides context

Tiny Inlining

- Context Threading is a dispatch technique
  - But, we inline branches
- Some non-branching bodies are very small
  - Why not inline those?

▸ Inline all tiny linear bodies into the CTT

Overview

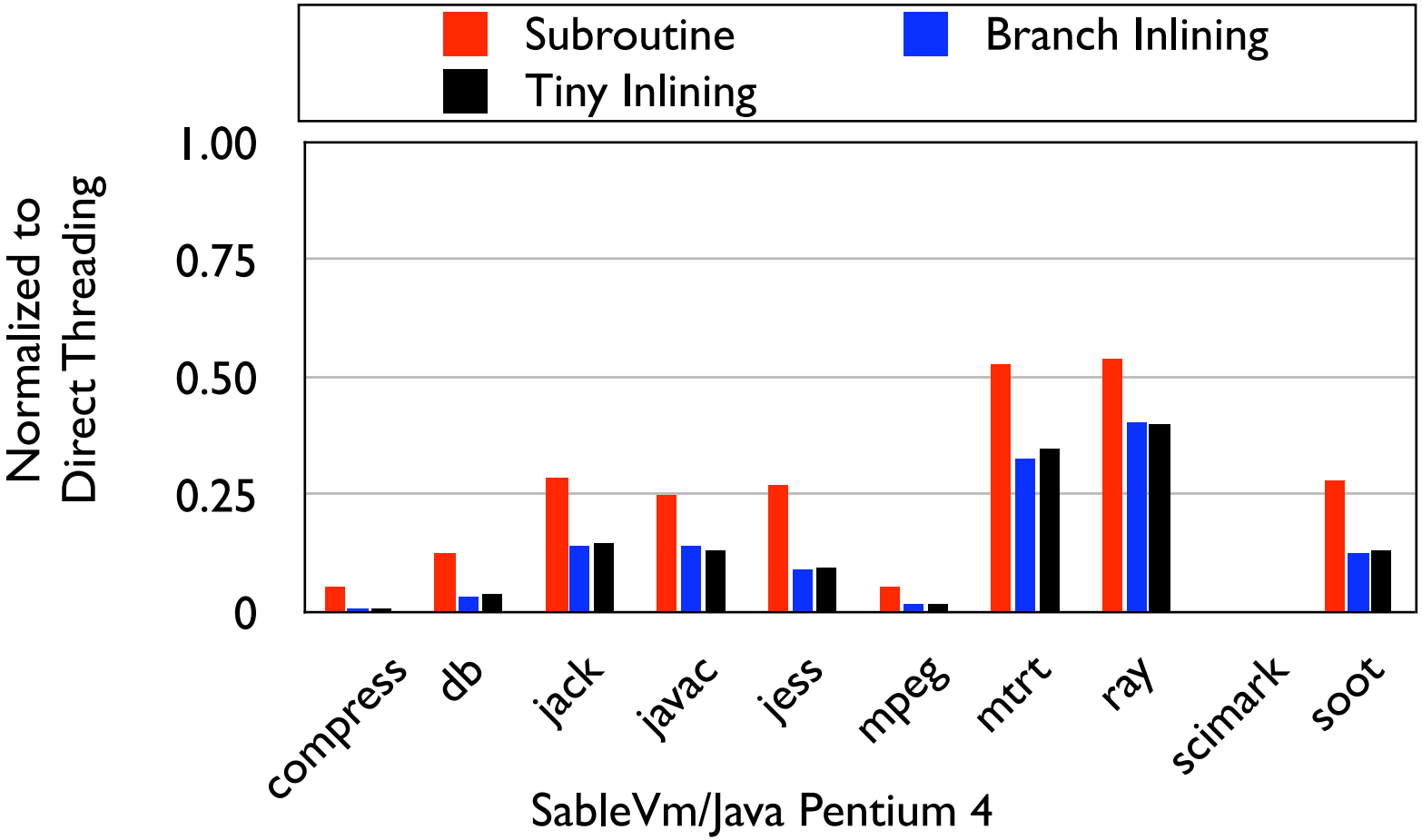
- ✔ Motivation
- ✔ Background: The Context Problem
- ✔ Existing Solutions
- ✔ Our Approach
- ✔ Inlining
- Results

Experimental Setup

- Two Virtual Machines on two hardware architectures.
  - VM: Java/SableVM, OCaml interpreter
    - Compare against direct threaded SableVM
    - SableVM distro uses selective inlining
  - Arch: P4, PPC
- Branch Misprediction
- Execution Time

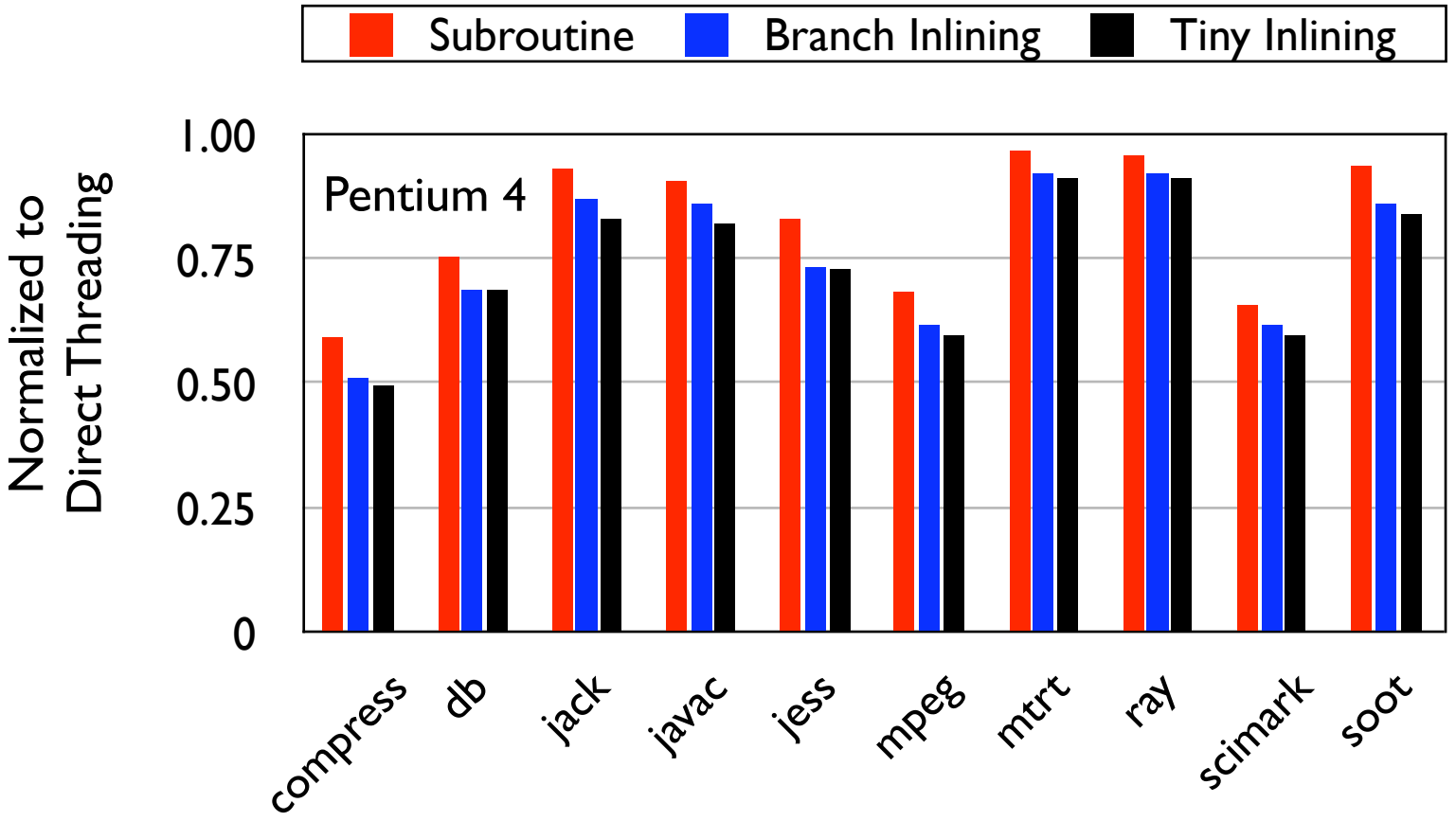
▸ Is our technique effective and general?

Mispredicted Taken Branches



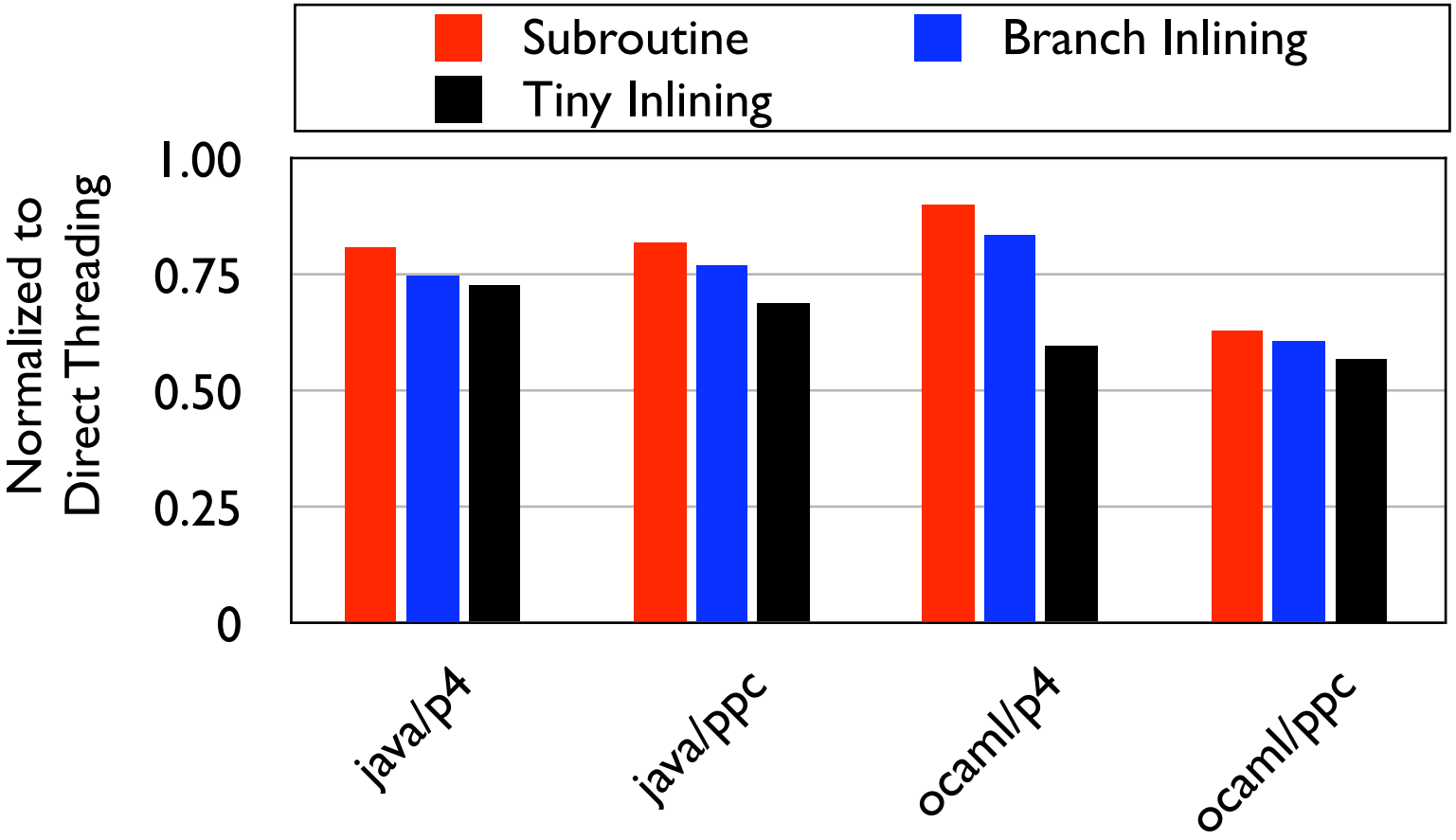
- 95% mispredictions eliminated on average

Execution time



- 27% average reduction in execution time

Execution Time (geomean)



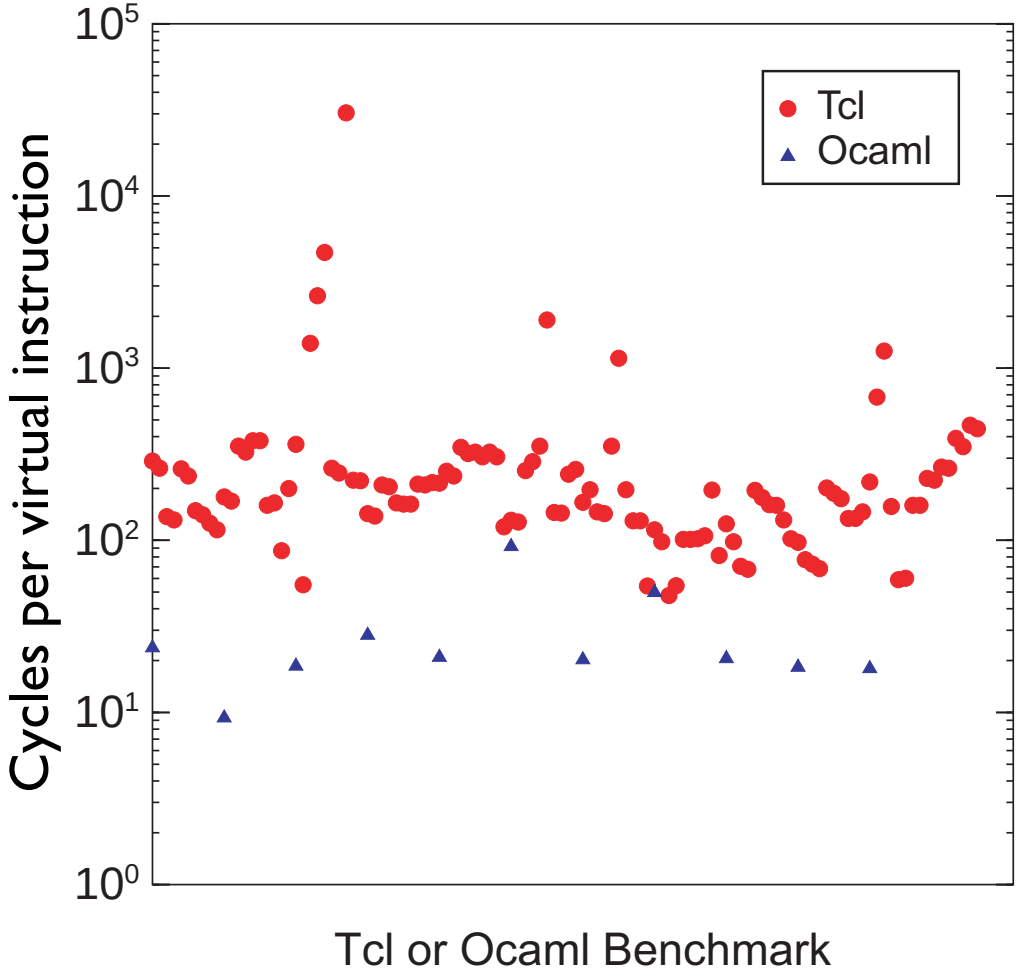
- Our technique is effective and general

Conclusions

- Context Problem: branch mispredictions due to mismatch between native and virtual control flow
- Solution: Generate control flow code into the Context Threading Table
- Results
  - Eliminate 95% of branch mispredictions
  - Reduce execution time by 30-40%

▸ recent, post CGO 2005, work follows

What about Scripting Languages?



- Recently ported context threading to TCL.
- 10x cycles executed per bytecode dispatched.
- Much lower dispatch overhead.
- Speedup due to subroutine threading, approx. 5%.
- TCL conference 2005