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

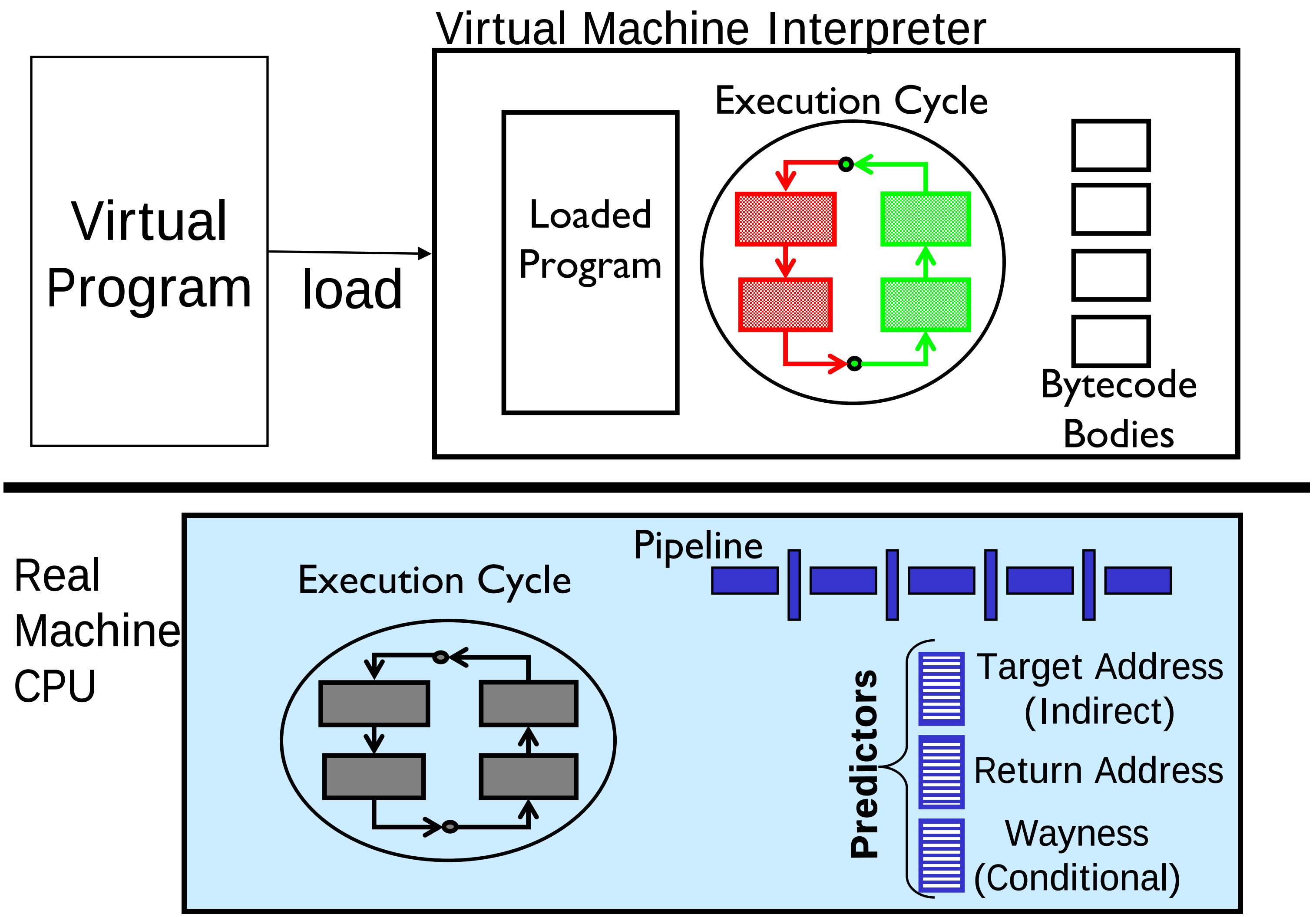
- Why not just JIT?
  - High performance JITs still interpret
  - People use interpreted languages that don't yet have JITs
  - They still want performance!

- 30-40% of execution time is due to branch misprediction
- Our technique eliminates 95% of branch mispredictions

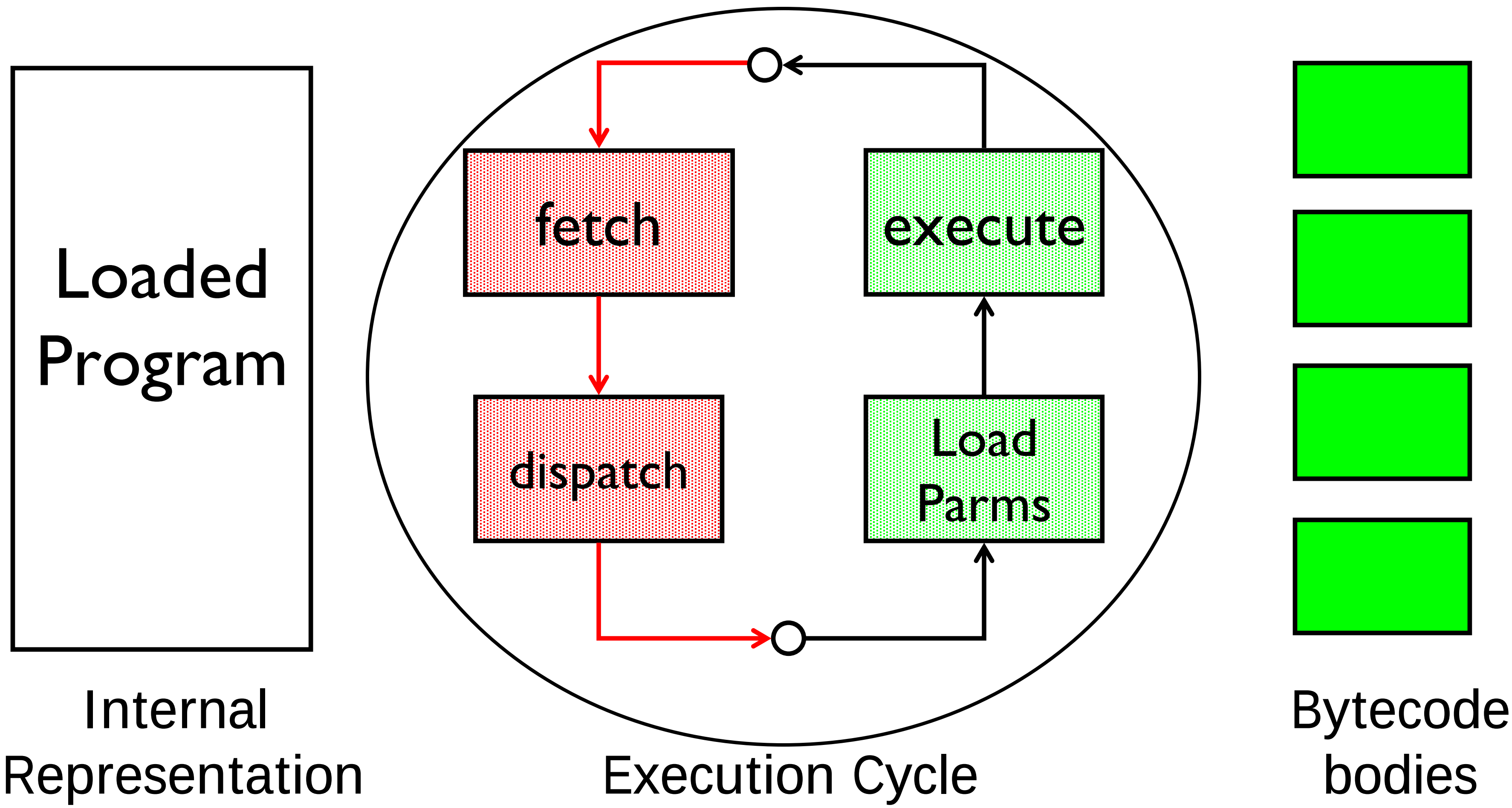
Overview

- ✓ 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);
}
```

Javac compiler

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

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

Virtual Program

vPC

```
<iload_1>
<iload_1>
<iadd>
<istore_1>
<iload_1>
<bipush>
64
<if_icmplt>
-7
```

Internal Representation

```
while(1){
 switch(*vPC++){
   case iload_1:
     ..
   break;
   case iadd:
     ..
   break;
  }
};
```

Switched Body Implementation

☞ Simple, portable and extremely slow

Direct Threaded Interpreter

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

Virtual Program

vPC

&&iload_1
&&iload_1
&&iadd
&&istore_1
&&iload_1
&&bipush
64
&&if_icmplt
-7

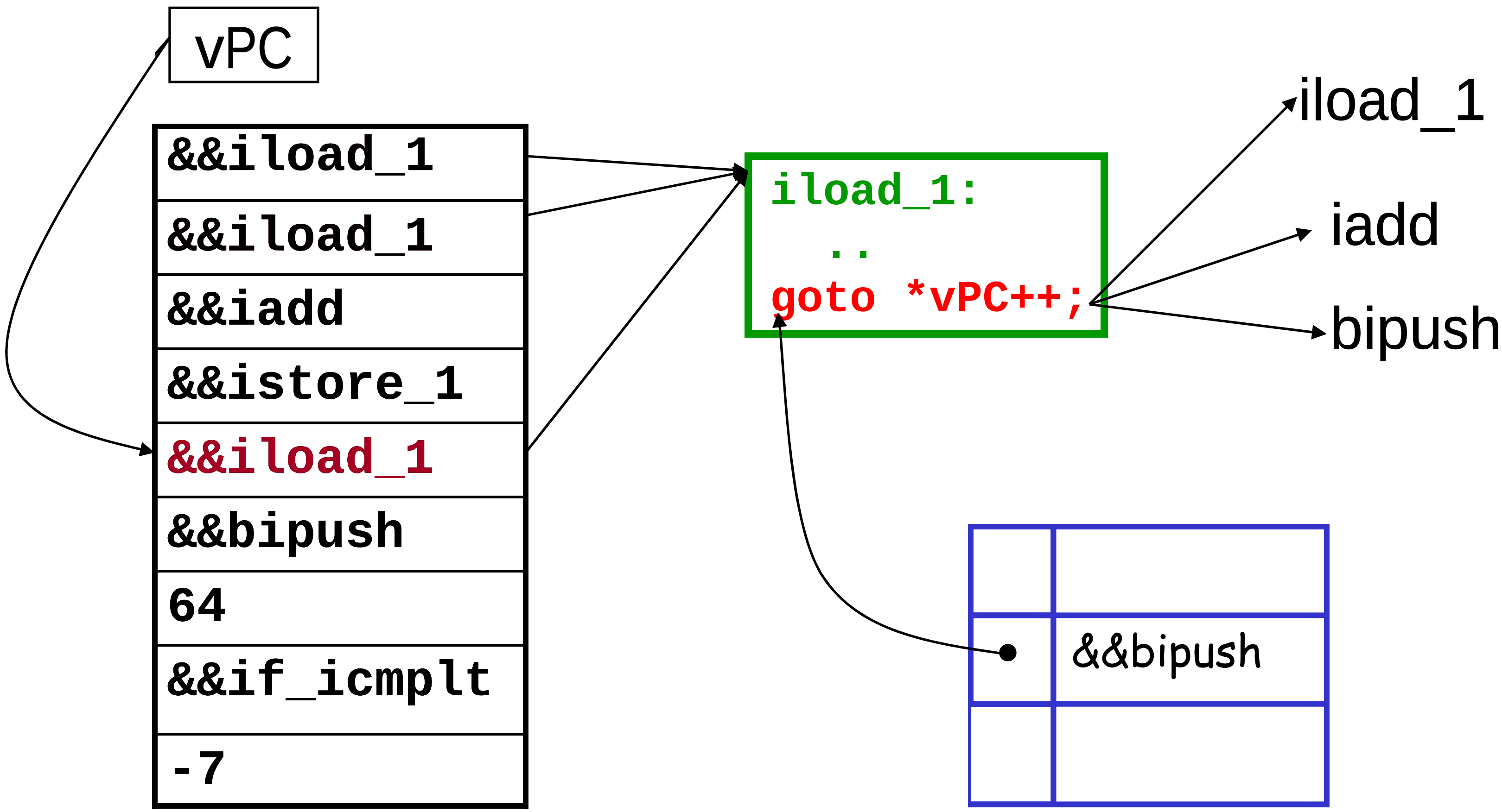
DTT - Direct Threading Table

```
iload_1:
   ..
goto *vPC++;
```

```
iadd:
   ..
goto *vPC++;
```

☞ Target of computed goto is data-driven

Context Problem

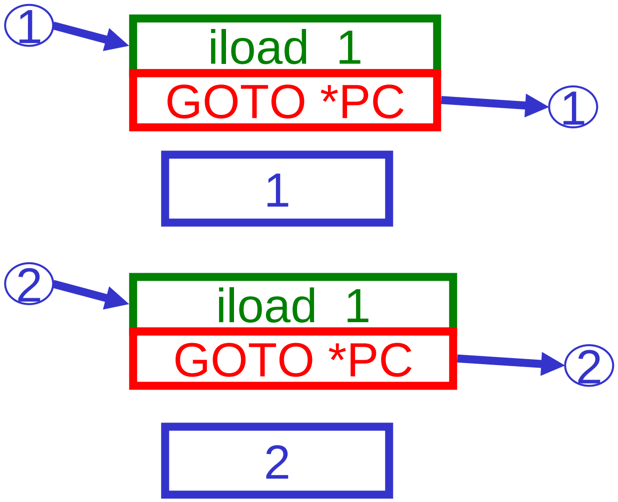


DTT - Direct Threading Table

Indirect Branch Predictors

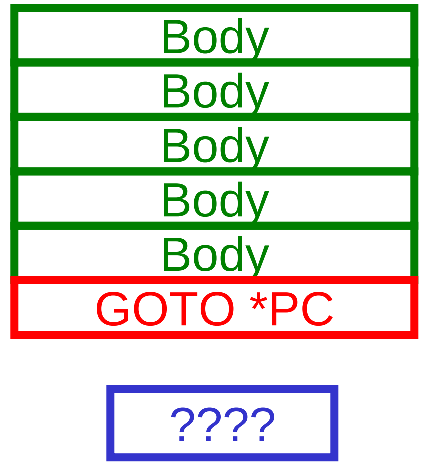
Existing Solutions

Replicate



Ertl & Gregg:
Bodies and Dispatch Replicated

Super Instruction



Piumarta & Ricardi :
Bodies Replicated

☞ Limited to relocatable virtual instructions

Overview

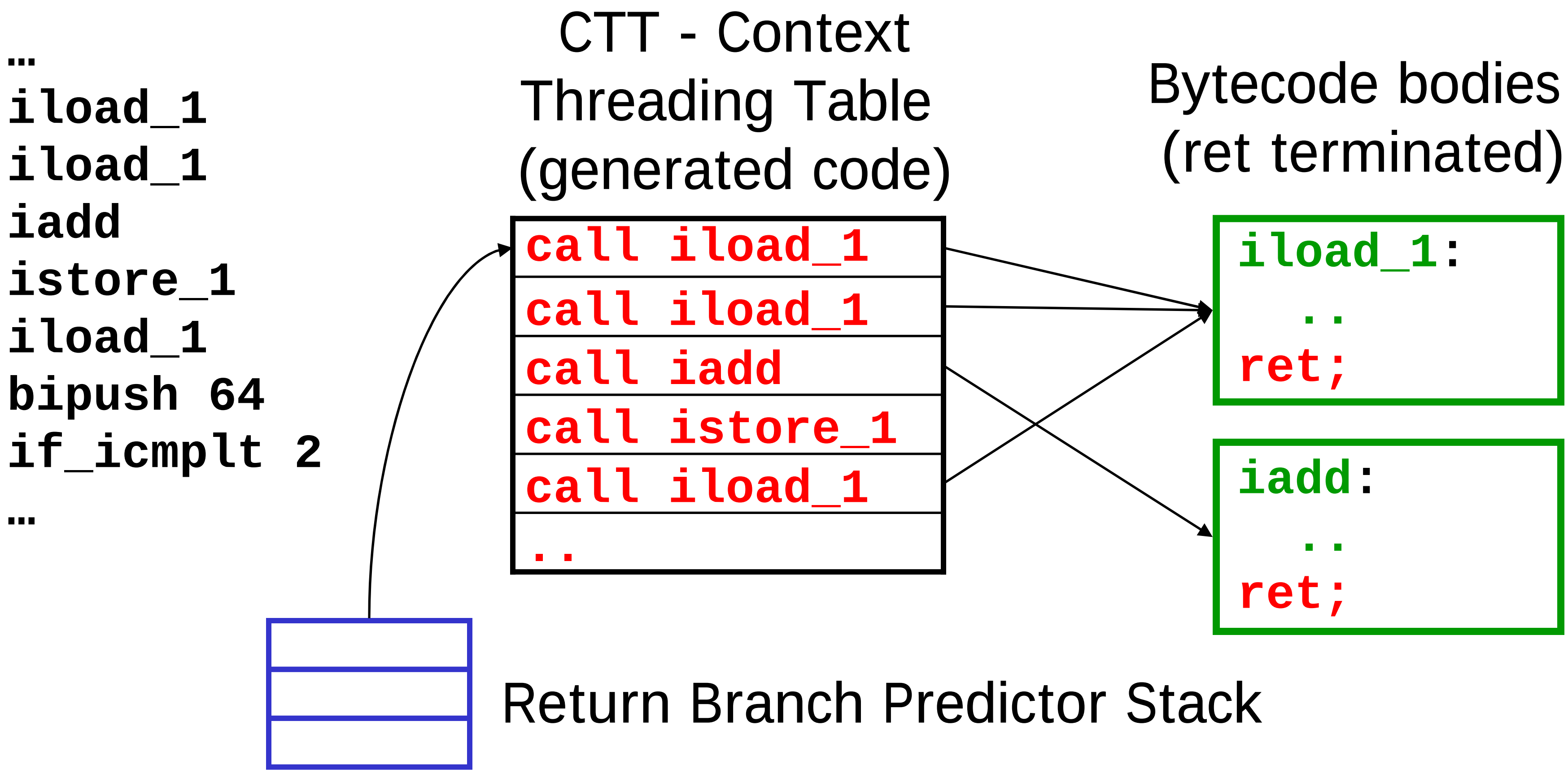
- ✓ Motivation
- ✓ Background: The Context Problem
- ✓ Existing Solutions
- Our Approach
- Inlining
- Results

Key Observation

- Virtual and native control flow have same branch types
  - Linear (not really a branch)
  - Conditional
  - Calls and Returns
  - Indirect
- Hardware has predictors for each type

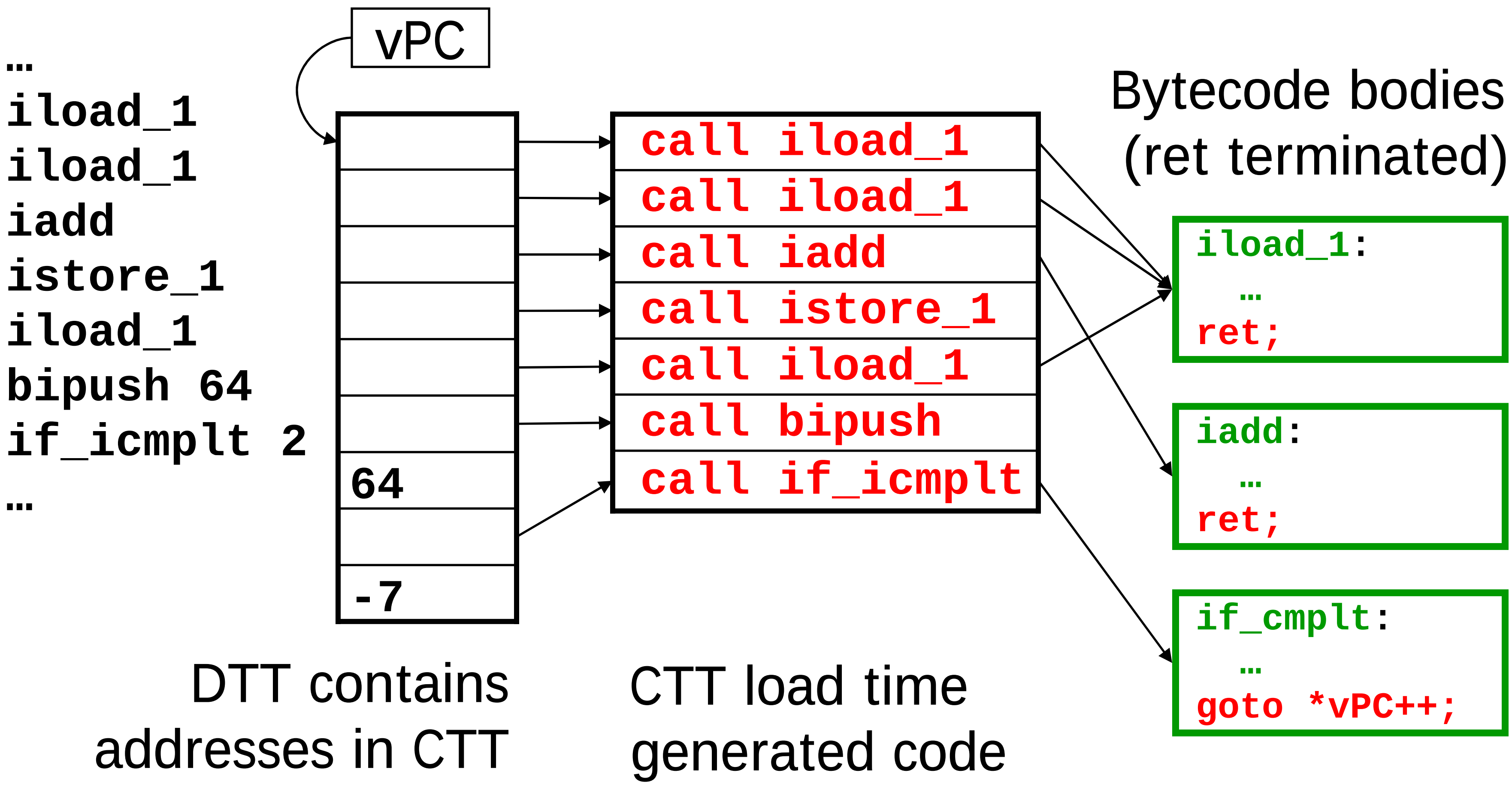
☞ Solution: Leverage hardware predictors

Essence of our Solution



☞ Package bodies as subroutines and call them

Context Threading



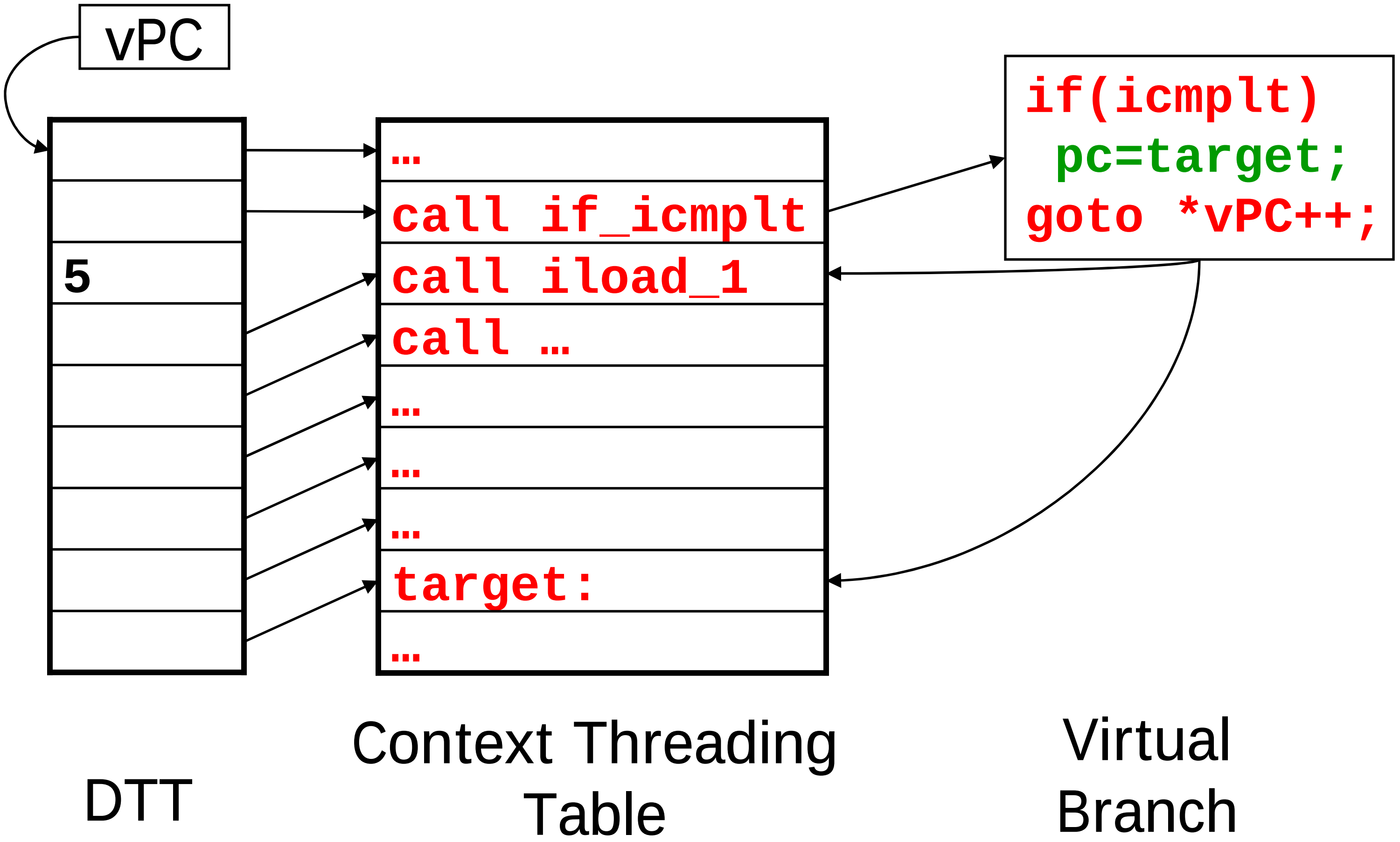
☞ Generate calls into the CTT at load time

The Context Threading Table

- A sequence of calls
- A sequence of generated instructions
- An internal representation of the program's control flow
  - Virtual branches are also control flow

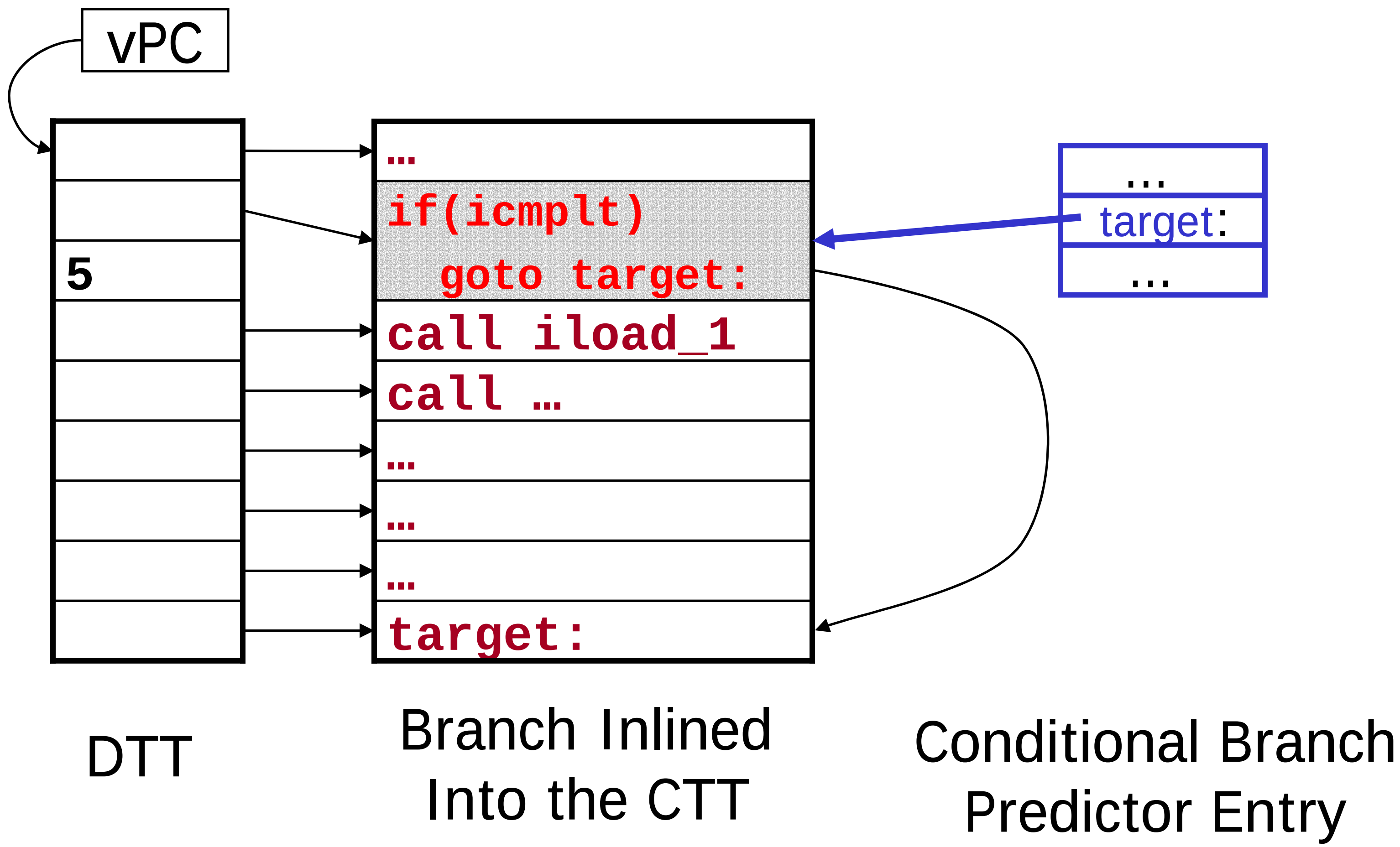
☞ Can virtual branches go into the CTT?

Virtual Branches



☞ Context problem is worse for virtual branches

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

What can go in the CTT?

- Calls to bodies
- Inlined bodies
- Mixed-Mode virtual machine?
- Partially Inlined bodies
- Calls to subroutines that aren't bytecodes
- Generated code

☞ Performance?

Overview

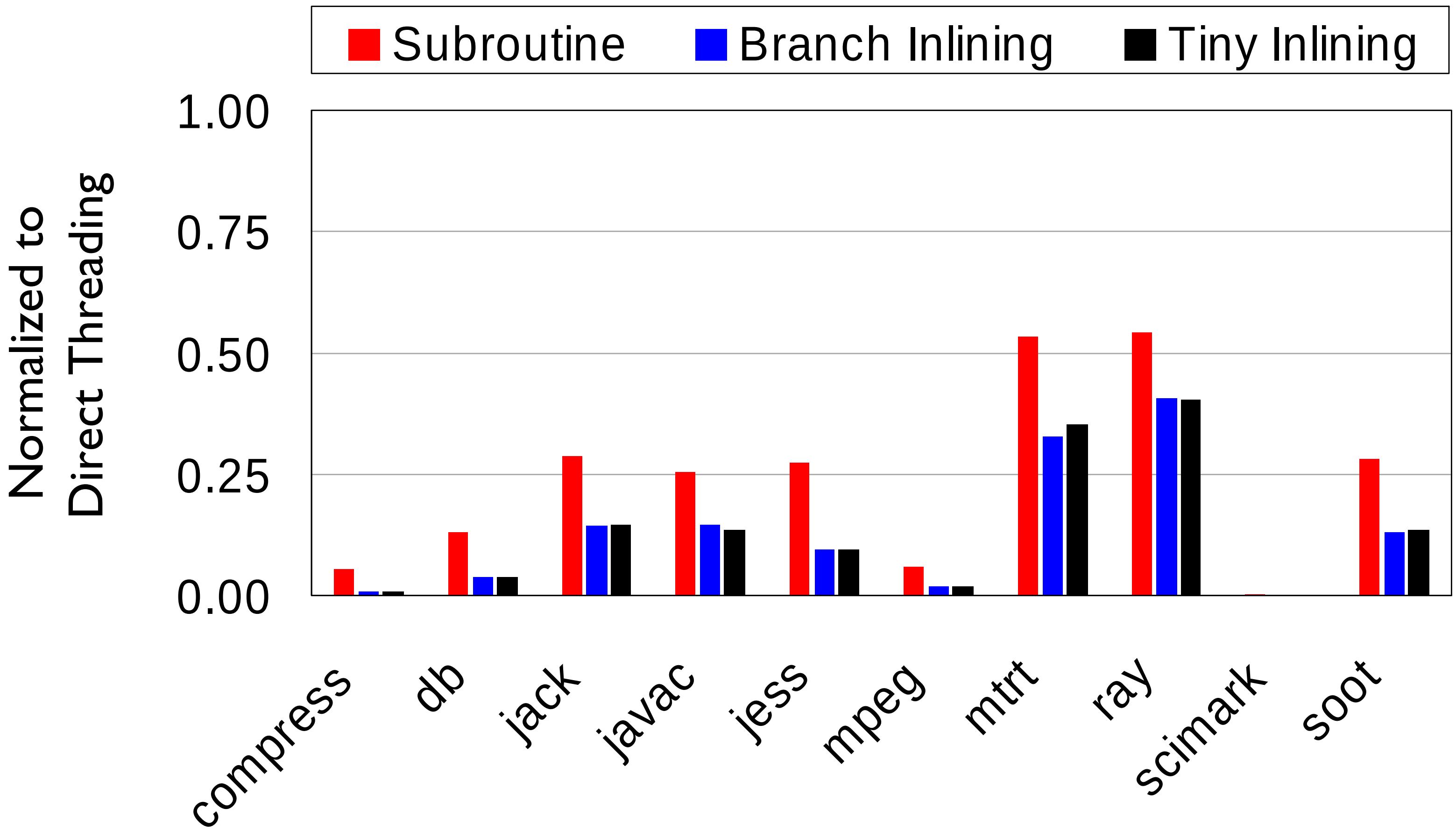
- ✓ Motivation
- ✓ Background: The Context Problem
- ✓ Existing Solutions
- ✓ Our Approach
- ✓ Inlining
- Results

Experimental Setup

- Two Virtual Machines on two hardware architectures.
  - VM: Java/SableVM, OCaml interpreter
  - Arch: P4, PPC
- Branch Misprediction
- Execution Time

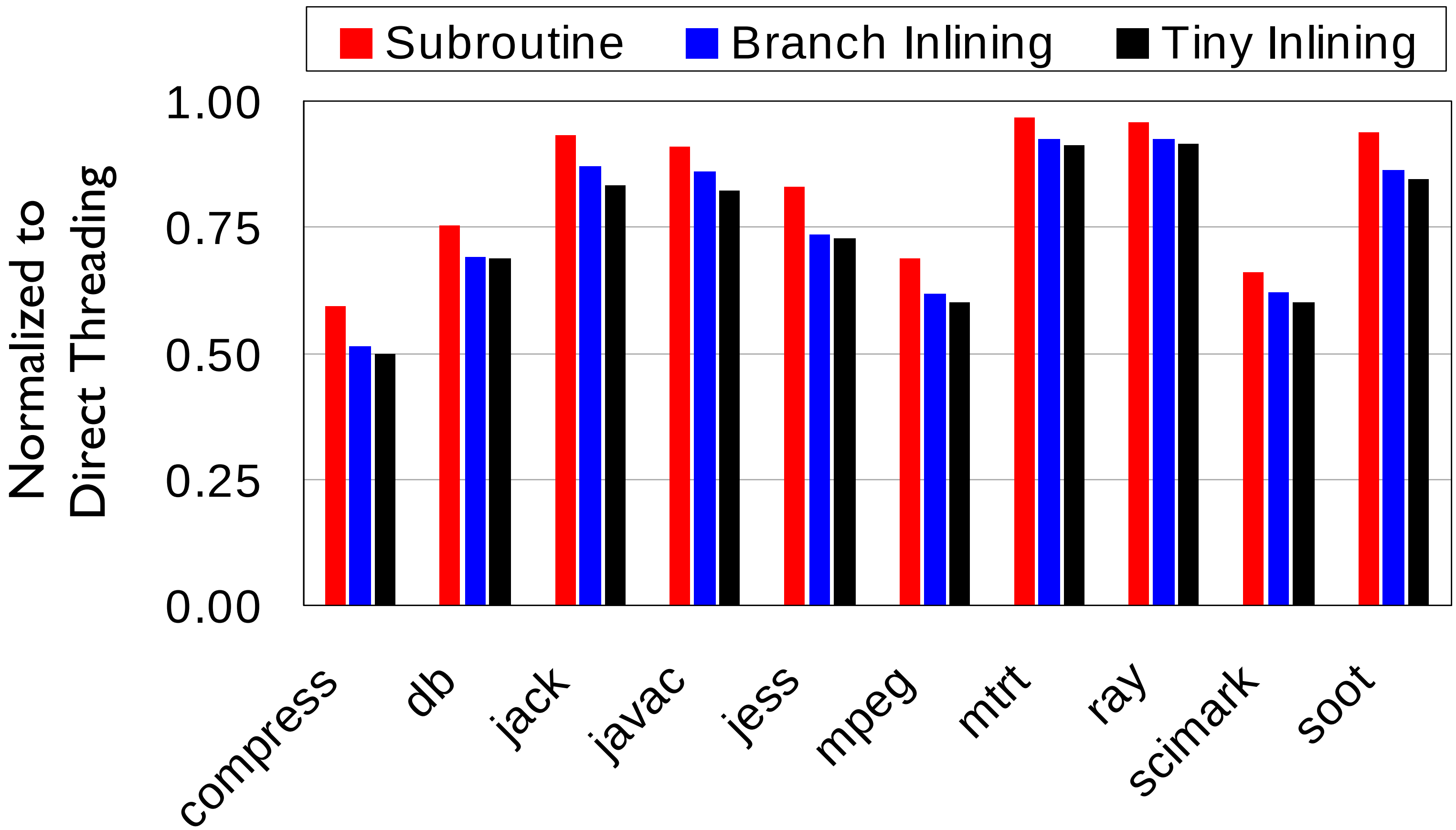
☞ Is our technique effective and general?

Mispredicted Taken Branches



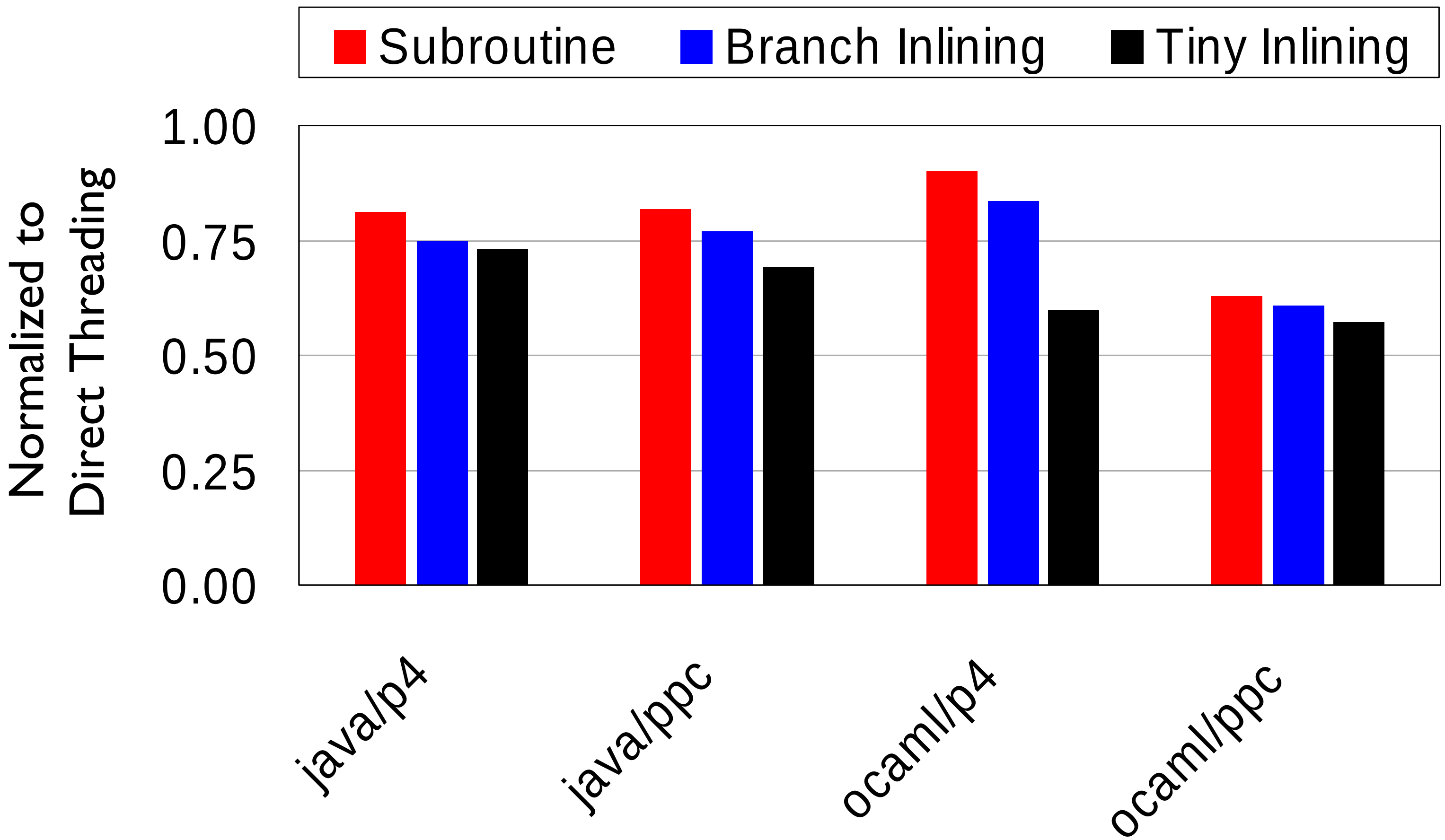
☞ 95% mispredictions eliminated on average

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%