

Optimizing Precision Overhead for x86 Processors

Takeshi Ogasawara

Hideaki Komatsu

Toshio Nakatani

IBM Tokyo Research Laboratory



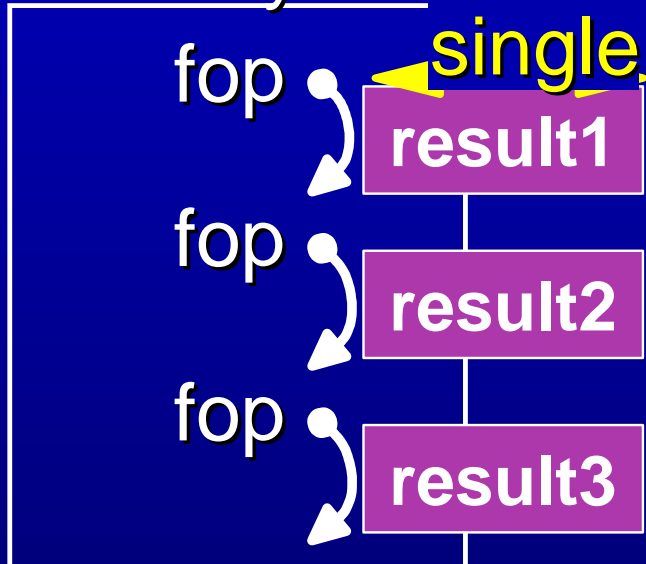
Outline

- **Background**
- **Our approach**
- **Experimental results**
- **Conclusion**

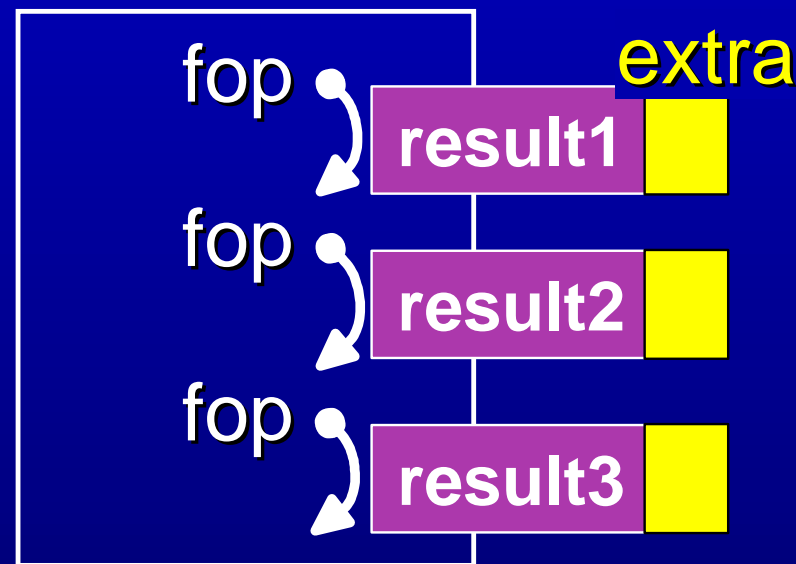
Floating-point operations in Java

- Each operation must produce the result of **the correct precision**.
 - ▶ **fop** (fadd, fmul, etc.) for *single-precision* bytecode
 - ▶ **dop** (dadd, dmul, etc.) for *double-precision* bytecode
- Example

Java bytecode

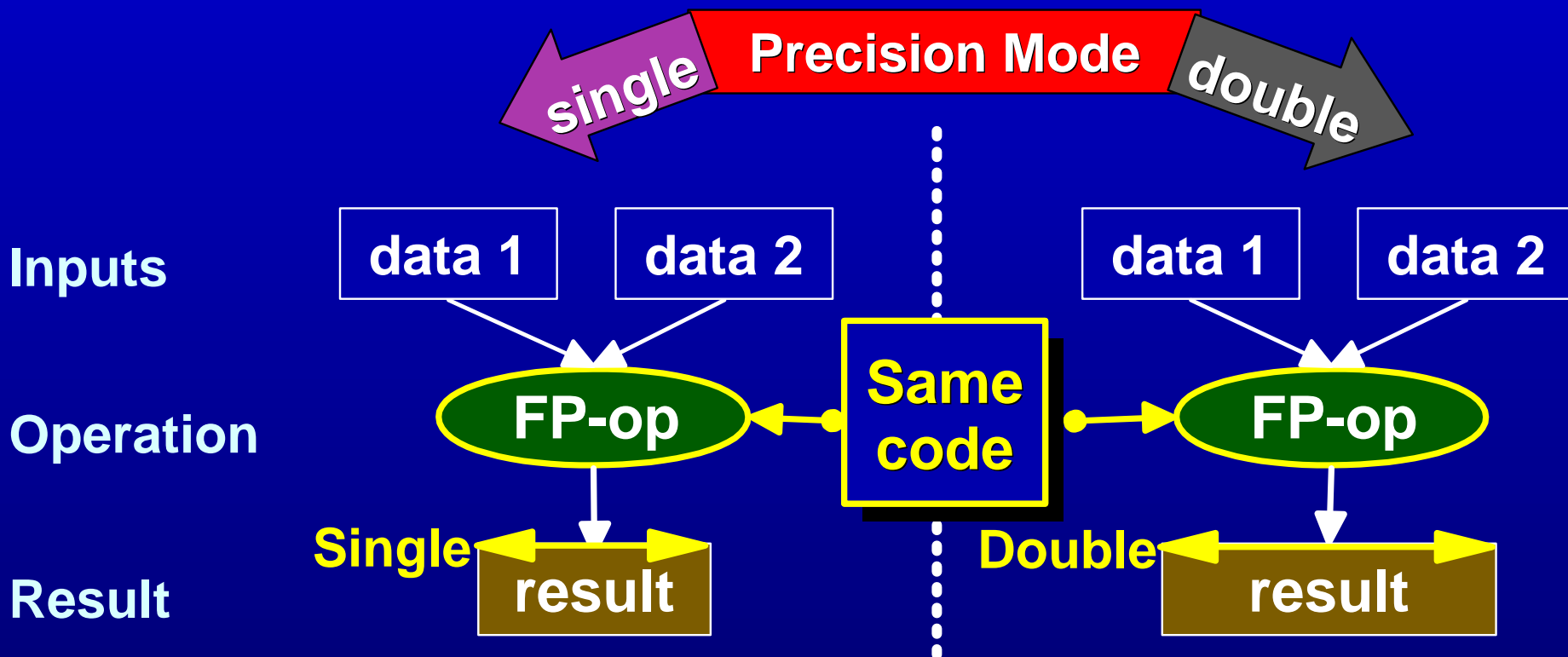


Not Java



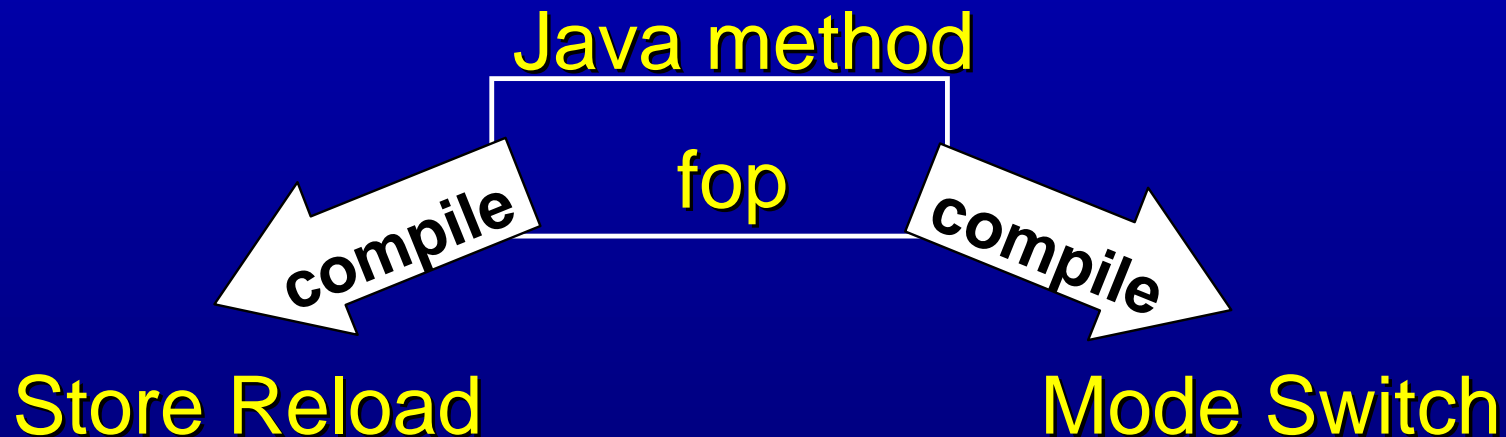
X86 specific feature: precision mode

- X86 does not have instructions corresponding to each of **fop** and **dop** bytecodes.
- **The precision mode** controls the results of instructions.



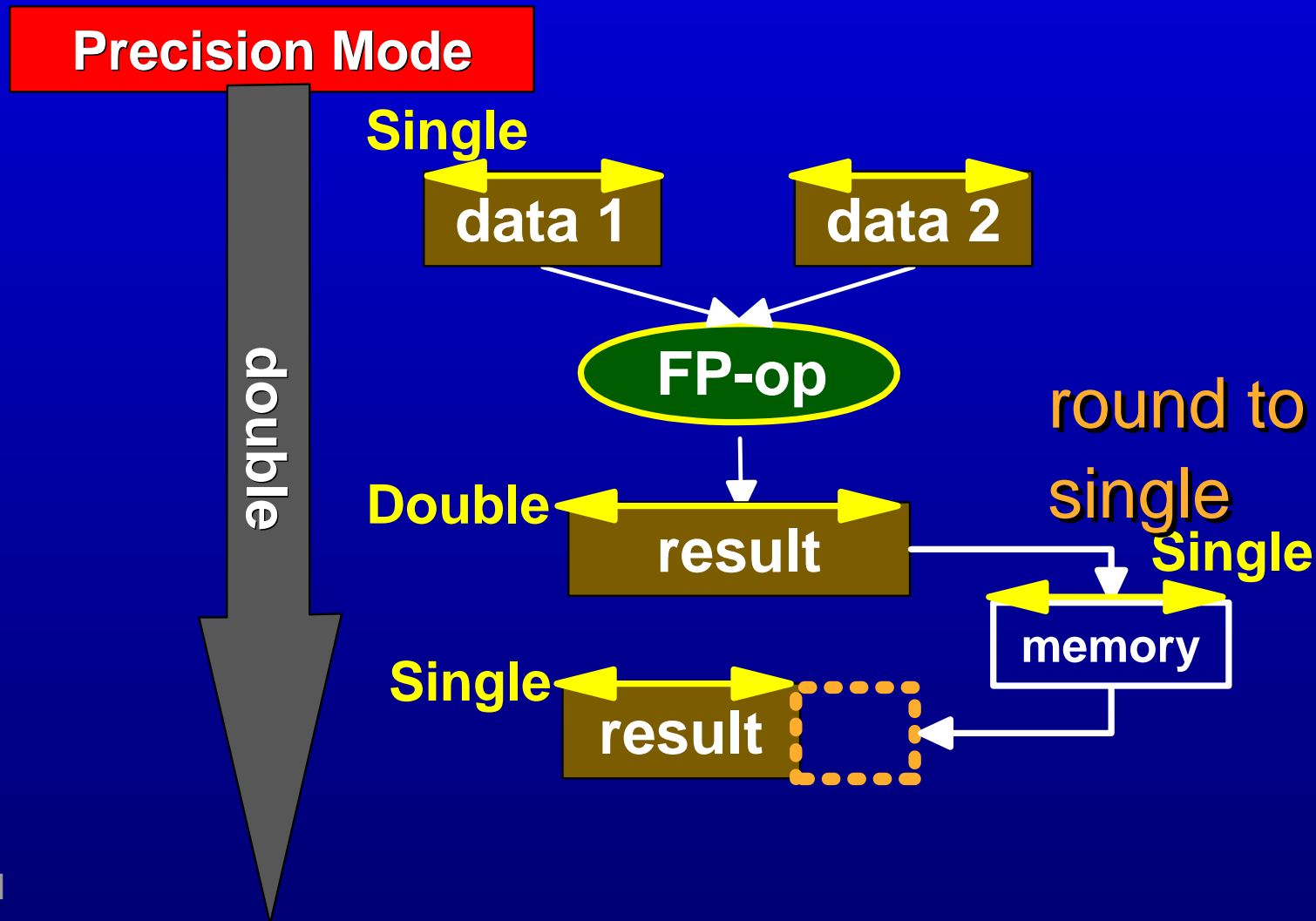
Single-precision operations under the default mode

- The default precision mode is *double*.
 - ▶ Assume that methods are called in this mode.
- Each Java method is compiled using either
 - ▶ **Store Reload** [Java Grande Report] or
 - ▶ **Mode Switch**



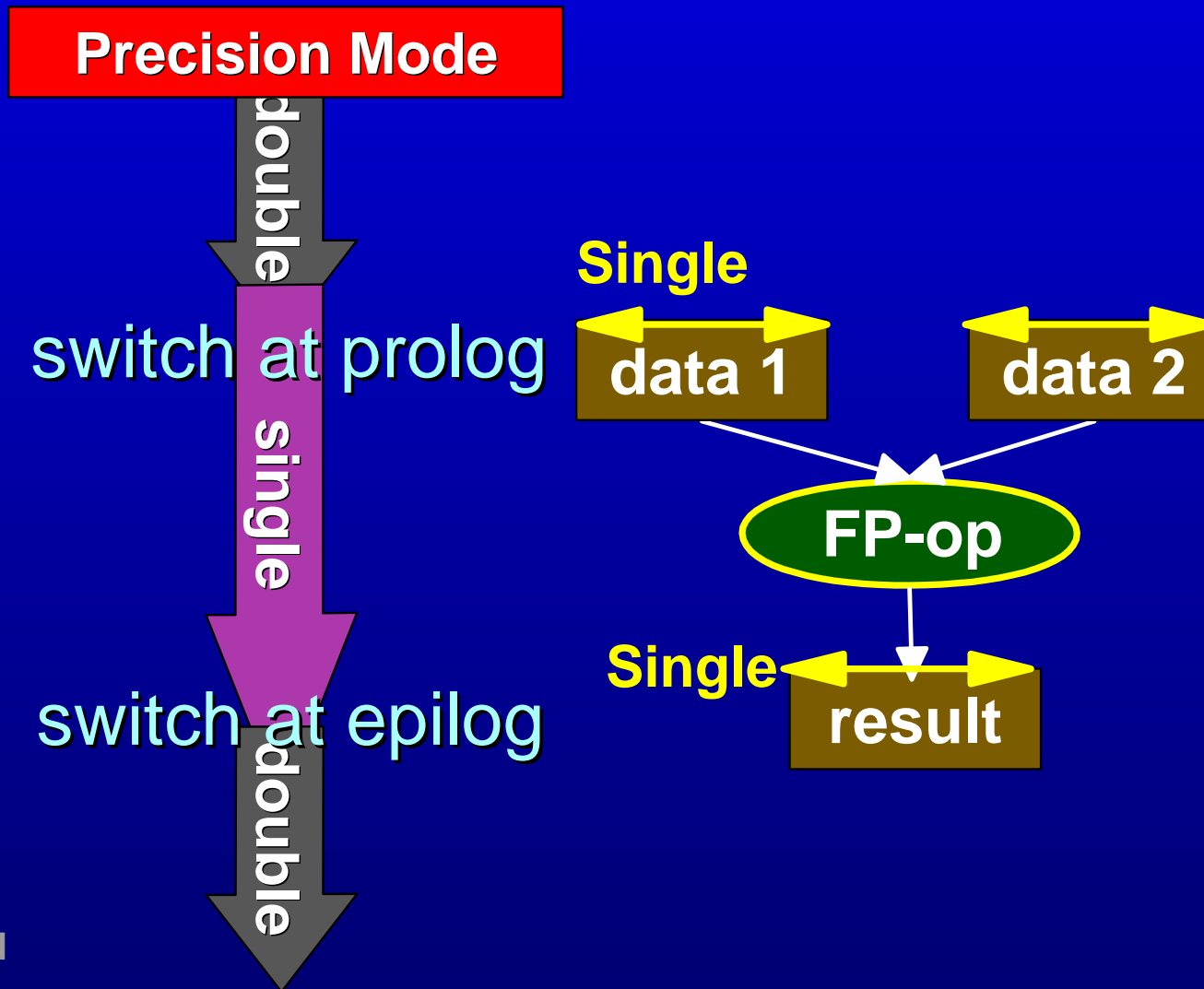
Store Reload

- Executes a single-prec op **in the double mode**
- **Rounds** the result by using the memory



Mode Switch

- Executes a single-prec op **in the single mode**
- **Switches** the mode at the prolog/epilog/call sites



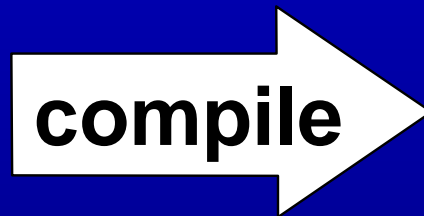
Code generation for a method

■ General case

- ▶ **Store Reload** is used for *mixed-precision methods*.
- ▶ Single- and double-precision ops are included.

Java bytecode

fop
dop
fop



x86 code

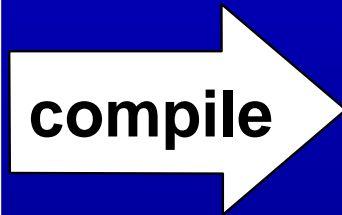
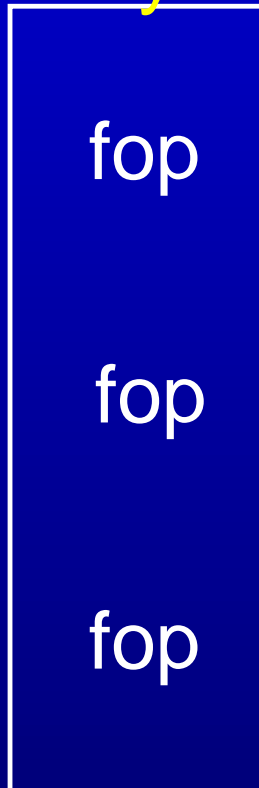
FP-op (in double)
round to single
FP-op (in double)

FP-op (in double)
round to single

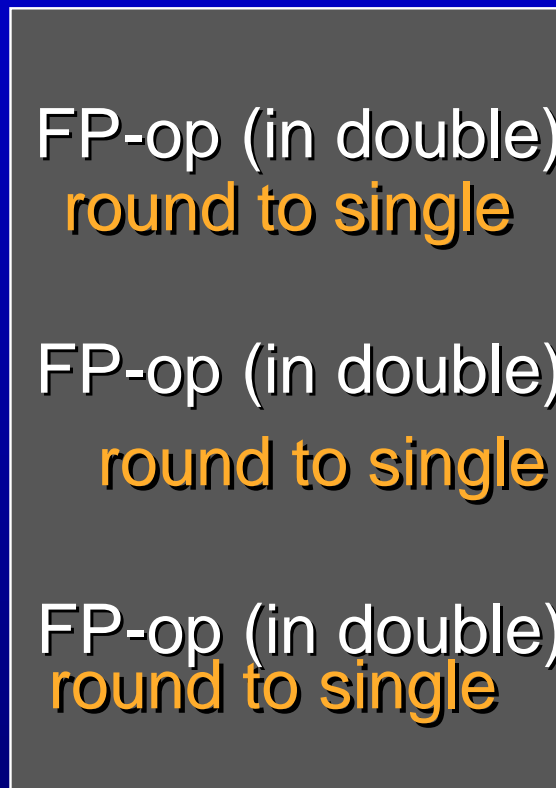
Code generation for a method (contd.)

- **Single-precision-only case**
 - ▶ **Store Reload or Mode Switch**
 - ▶ **Depending on the costs of rounds and switches**

Java bytecode



x86 code



OR



Heuristics of single-precision technique selection

- [JVM01] Placzny, et al.
 - ▶ Mode Switch if
 - # doubleOps == 0 &&
 - # floatOps > 32 &&
 - # floatOps > 10 * # callSites
 - ▶ Otherwise, Store Reload

Problems

- *Mostly-single-precision* methods
- *Useless mode switches*

Problem 1: Mostly-single-precision methods

- **Much fewer** *double*-precision operations are performed than *single*-precision operations.
- Suffer from the rounding cost

Java bytecode

```
fop  
fop  
dop  
fop  
fop  
fop
```

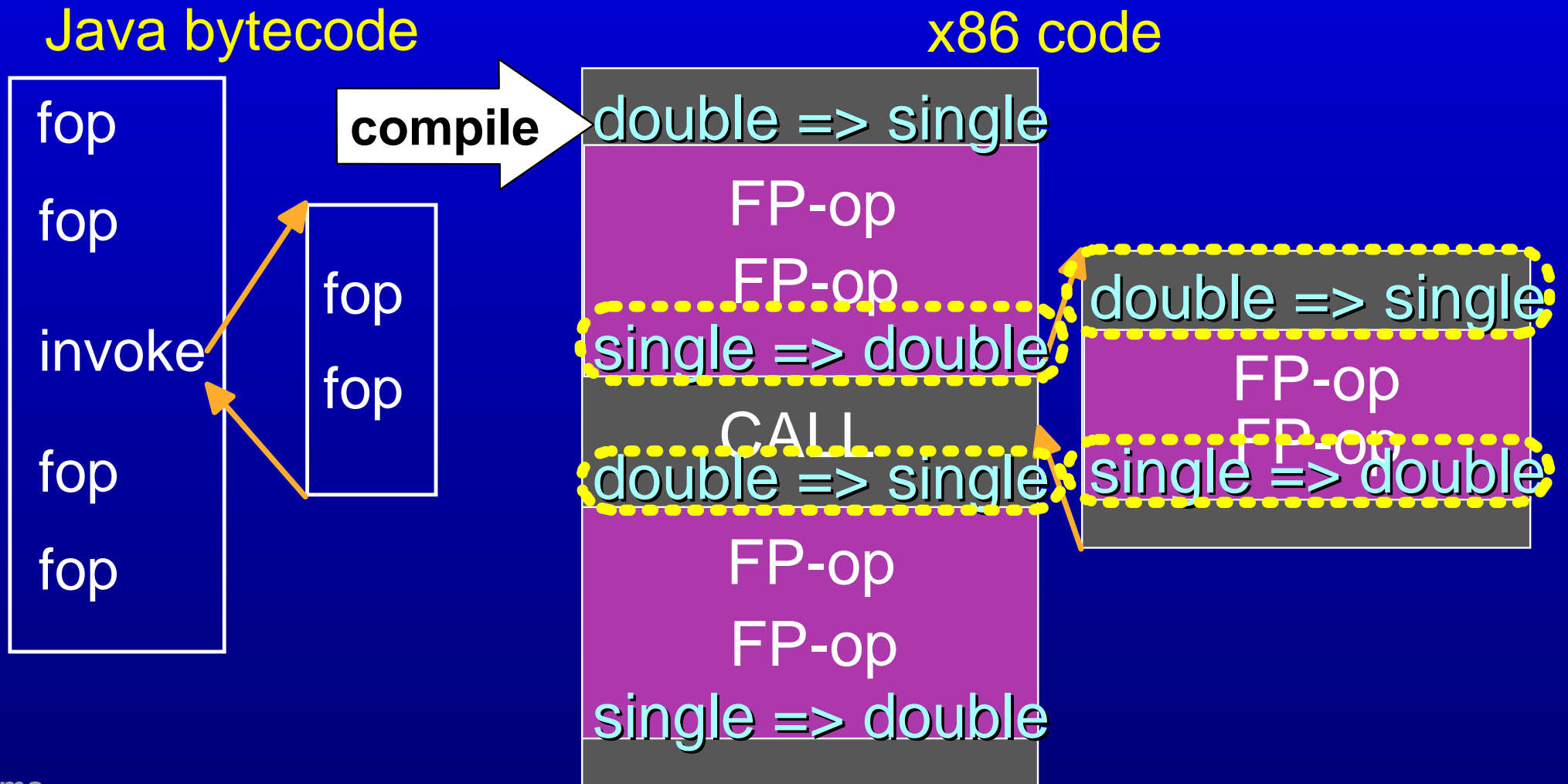
compile

x86 code

```
FP-op  
round  
FP-op  
round  
FP-op  
FP-op  
round  
FP-op  
round  
FP-op  
round
```

Problem 2: Useless mode switches

- Single-double-single switches during invocations



Our Approach

- For mostly-single-precision methods:
 - ▶ *Excessive precision optimization*
 - ▶ *Precision region analysis*
- For useless mode switches:
 - ▶ *Precision-aware code generation*

Excessive precision optimization

- Replaces a **double**-prec code sequence by a **single**-prec code that **calculates the same result**

▶ E.g. " f *= 0.5 "

– Compiled to bytecodes **f = D2F(F2D(f) * 0.5D)**

– Equal to **f = f * 0.5F**

Java bytecode

```
fop
fop
dop
fop
fop
```

transform

Java bytecode

```
fop
fop
fop
fop
fop
```

Precision region analysis

- Splits a method into *precision regions*
 - ▶ Consists of either single or double operations
- **Switches** the precision mode **region by region**

Java bytecode

```
fop
fop
dop
fop
fop
fop
```

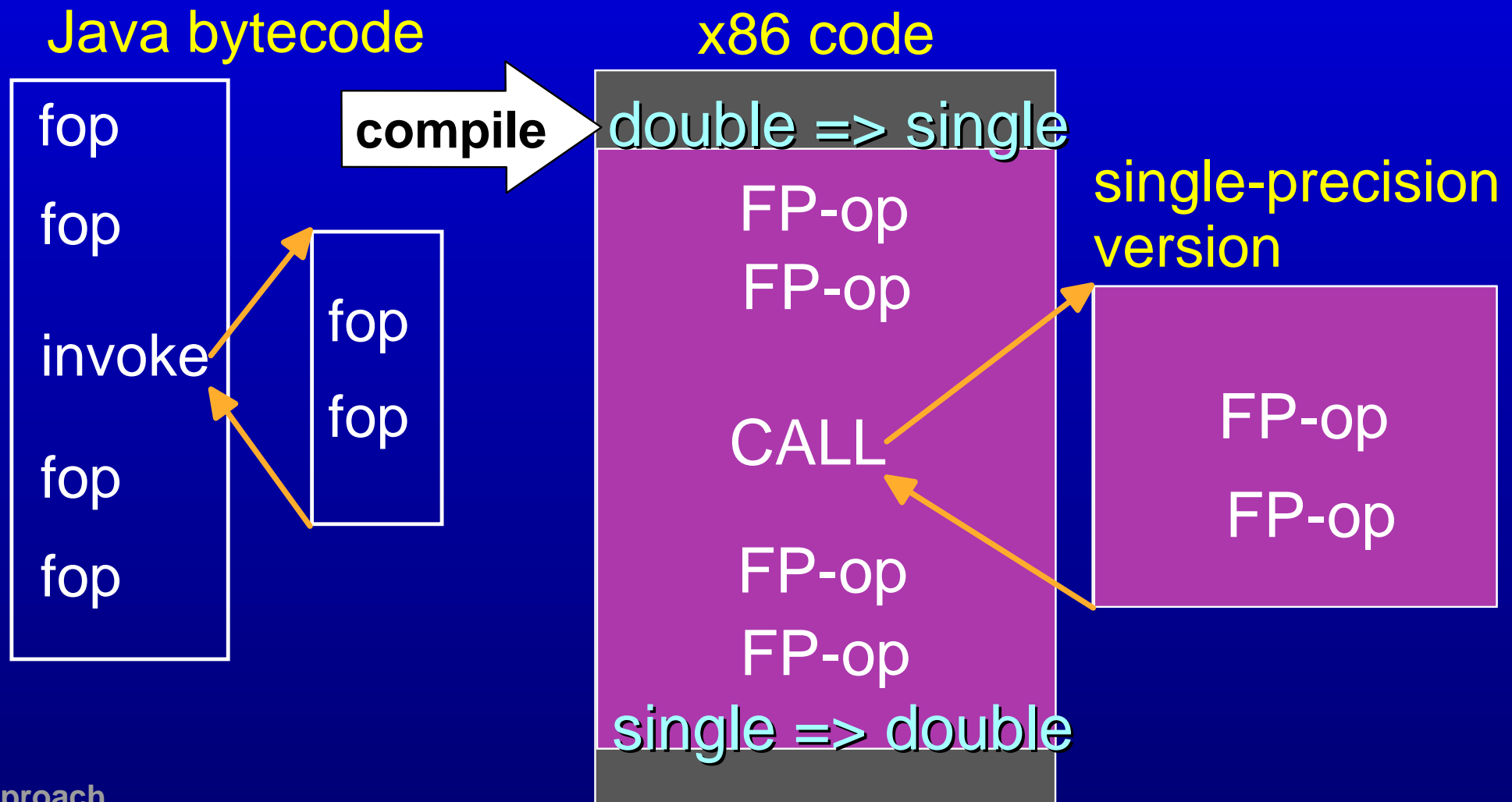
compile

x86 code

```
FP-op
FP-op
single => double
FP-op
double => single
FP-op
FP-op
FP-op
```

Precision-aware code generation

- Generates the code of **the caller's precision mode**

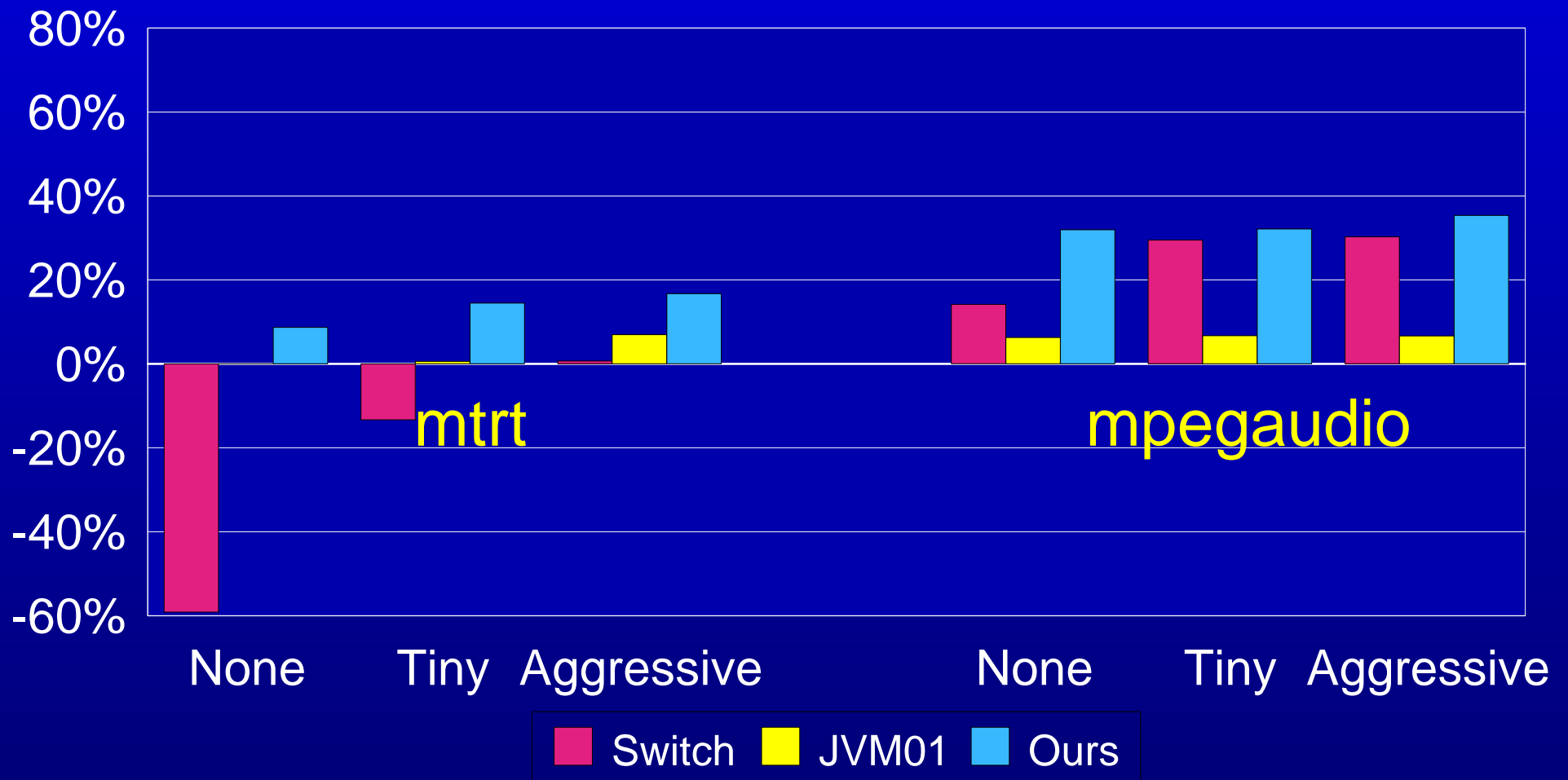


Performance Measurements: Methodology

- JIT compiler for IBM DK 1.3.1 for Windows
- Compare four single-prec operation techniques:
Store reload, Switch, JVM01, Ours
 - ▶ Excessive-precision opt is commonly applied.
- Two FP-intensive programs from SPECjvm98:
_227_mtrt, _222_mpegaudio
- Three inlining policies (the problems are sensitive to #calls and #FPOpsInMethod):
No, Tiny methods, Aggressive
- Three x86 implementations:
Pentium III, Pentium 4, Athlon MP

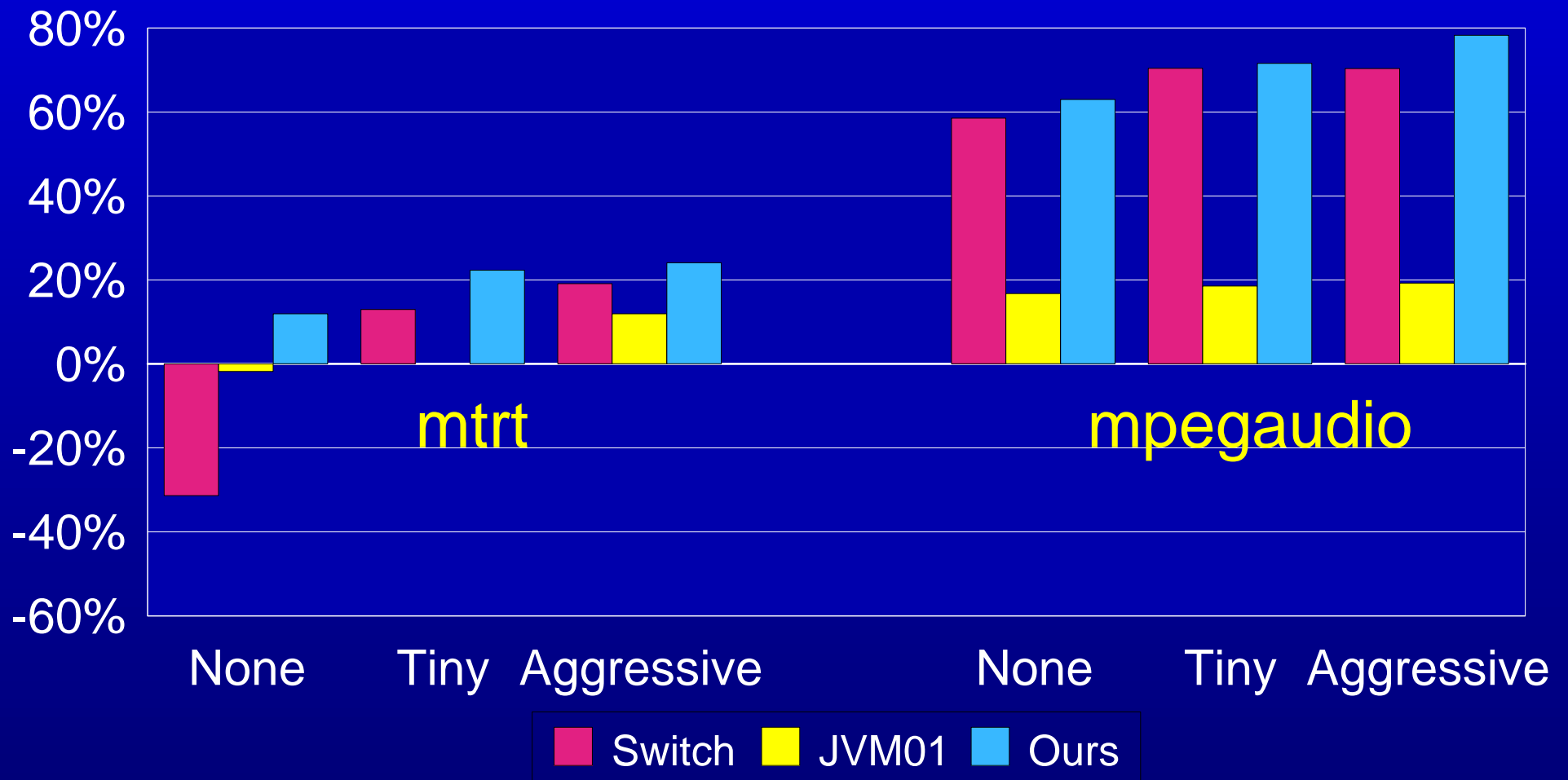
Performance Measurements: Pentium III

- Switch is worse than Store Reload for mtrt.
- Ours consistently improves mtrt and mpegaudio.



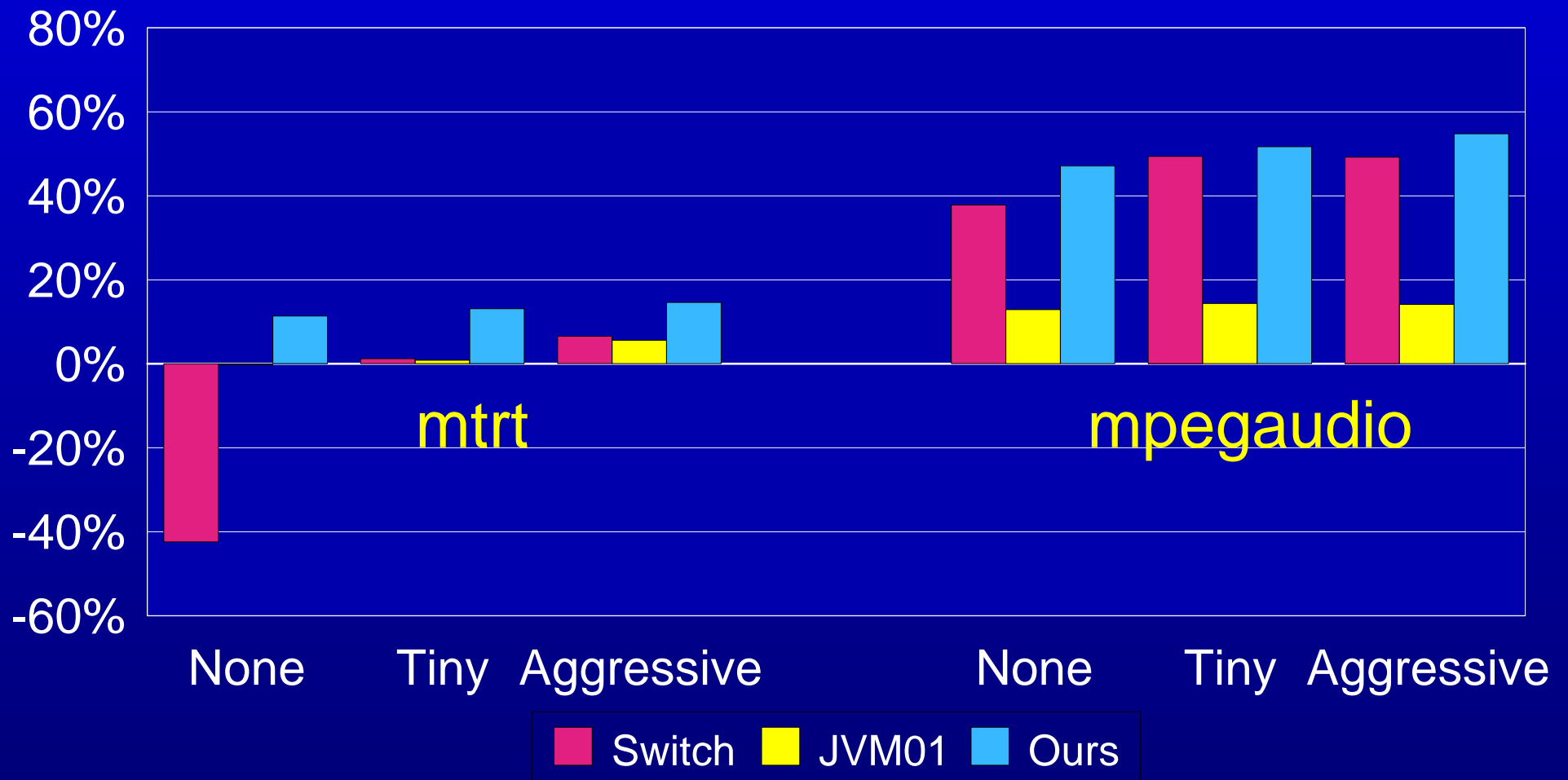
Performance Measurements: Pentium 4

- Switch is still worse for mtrt w/o inlining.
- Useless switches improved by hardware still costs.



Performance Measurements: Athlon MP

- Results are similar to that of Pentium 4.



Performance Measurements: Summary

- **Our approach shows the best performance in any configuration of:**
 - ▶ **benchmark programs**
 - ▶ **the ranges of compile scope (inlining policies)**
 - ▶ **processor implementations**

Conclusion

- **Problems for single-precision ops on x86**
 - ▶ **Mostly-single-precision methods**
 - ▶ **Useless mode switches**
- **A novel approach to optimization of Java floating-point operations for x86**
- **Three optimization techniques:**
 - ▶ **Excessive precision optimization**
 - ▶ **Precision region analysis**
 - ▶ **Precision-aware code generation**