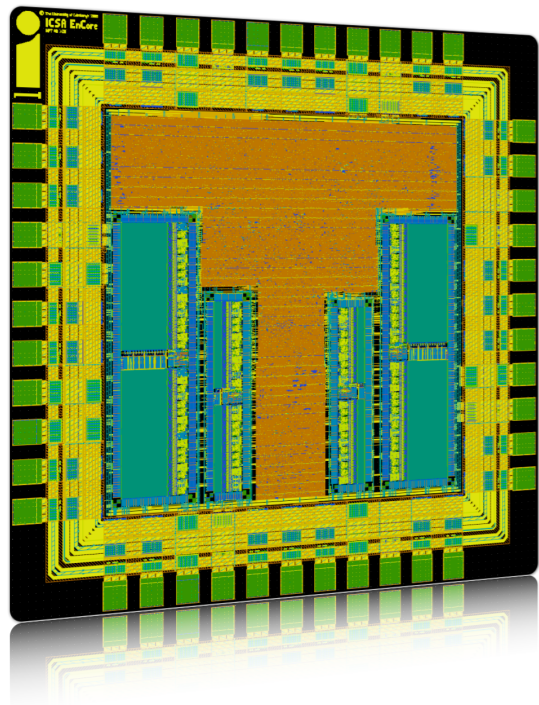




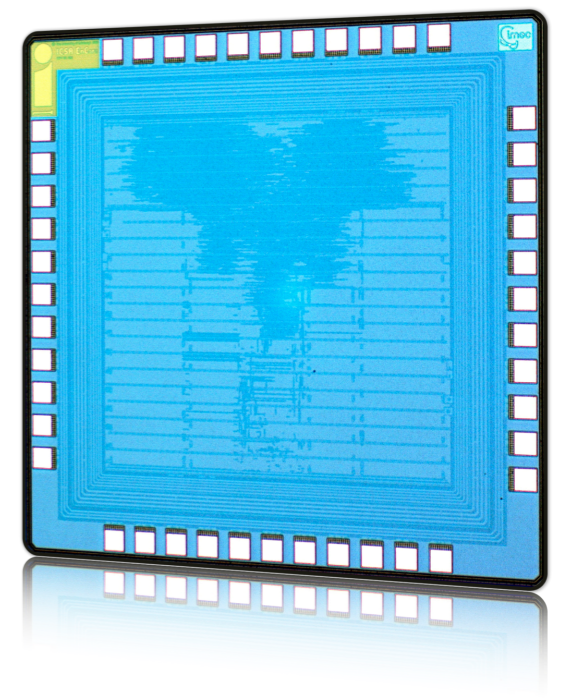
# Compact Code Generation

Tobias Edler von Koch  
Igor Böhm and Björn Franke



# PASTA

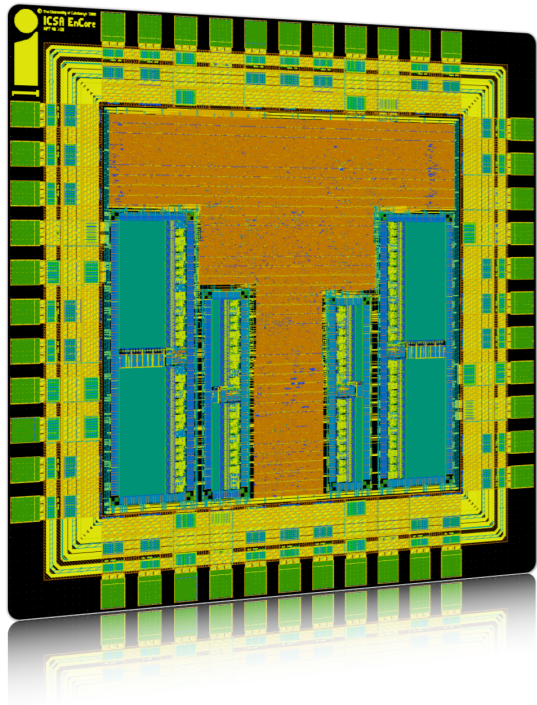
**P**rocessor **A**utomated **S**ynthesis  
by **iT**erative **A**nalysis





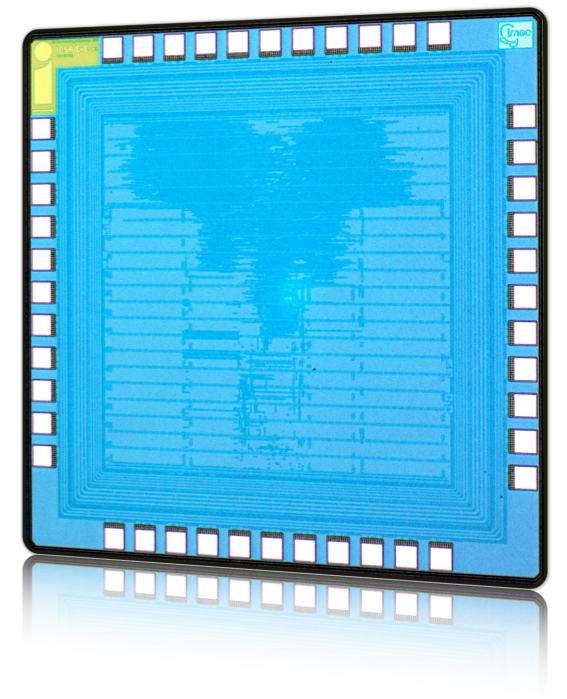
# Integrated Instruction Selection and Register Allocation for Compact Code Generation Exploiting Freeform Mixing of 16- and 32-bit Instructions

Tobias Edler von Koch  
Igor Böhm and Björn Franke



# PASTA

**P**rocessor **A**utomated **S**ynthesis  
by **iT**erative **A**nalysis





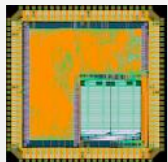
# Does Code Size Matter?

128-Mbit Flash  
27.3mm<sup>2</sup> at 0.13μm



# Does Code Size Matter?

128-Mbit Flash  
27.3mm<sup>2</sup> at 0.13μm



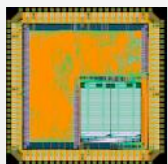
**Thumb-2  
ISA**

ARM Cortex M3  
0.43mm<sup>2</sup> at 0.13μm



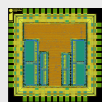
# Does Code Size Matter?

128-Mbit Flash  
27.3mm<sup>2</sup> at 0.13μm



**Thumb-2  
ISA**

ARM Cortex M3  
0.43mm<sup>2</sup> at 0.13μm

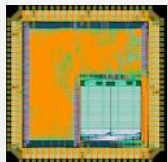


**ARCompact  
ISA**

ENCORE Calton  
0.15mm<sup>2</sup> at 0.13μm

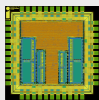
# Does Code Size Matter?

128-Mbit Flash  
27.3mm<sup>2</sup> at 0.13μm



**Thumb-2  
ISA**

ARM Cortex M3  
0.43mm<sup>2</sup> at 0.13μm



**ARCompact  
ISA**

ENCORE Calton  
0.15mm<sup>2</sup> at 0.13μm

## **RISC Architectures**

sacrifice code density  
in order to simplify  
implementation circuitry  
and decrease die area



# Solution to Code Size Problem

- Dual instruction sets providing 32-bit and 16-bit instruction encodings:



# Solution to Code Size Problem

- Dual instruction sets providing 32-bit and 16-bit instruction encodings:

***microMIPS***

**ARM Thumb-2**

**ARM Thumb**

**ARCompact**





# Solution to Code Size Problem

- Dual instruction sets providing 32-bit and 16-bit instruction encodings:

***microMIPS***

**ARM Thumb-2**

**ARM Thumb**

**ARCompact**

- There's no such thing as a free lunch!



# Solution to Code Size Problem

- Dual instruction sets providing 32-bit and 16-bit instruction encodings:

***microMIPS***

**ARM Thumb-2**

**ARM Thumb**

**ARCompact**

- There's no such thing as a free lunch!
- 16-bit instructions come with constraints!



# Common Compact Instruction Format Constraints

- Only subset of registers accessible



# Common Compact Instruction Format Constraints

- Only subset of registers accessible
- Explicit instructions necessary to switch between 16-bit and 32-bit modes



# Common Compact Instruction Format Constraints

- Only subset of registers accessible
- Explicit instructions necessary to switch between 16-bit and 32-bit modes
- Reduced size of immediate operands



# Common Compact Instruction Format Constraints

- Only subset of registers accessible
- Explicit instructions necessary to switch between 16-bit and 32-bit modes
- Reduced size of immediate operands
- Not every 32-bit instruction has a 16-bit counterpart



# Common Compact Instruction Format Constraints

- Only subset of registers accessible
- Explicit instructions necessary to switch between 16-bit and 32-bit modes
- Reduced size of immediate operands
- Not every 32-bit instruction has a 16-bit counterpart
- Free mixing of 16- and 32-bit encodings not always possible



# ARCompact 16-bit Instruction Format Constraints

- Only subset of registers accessible
- ~~Explicit instructions necessary to switch between 16-bit and 32-bit modes~~
- Reduced size of immediate operands
- Not every 32-bit instruction has a 16-bit counterpart
- **Free** mixing of 16- and 32-bit encodings ~~not~~  
~~always possible~~



# Motivating Example

32-Bit Only

```
ld  r2, [sp, 0]
ld  r3, [sp, 4]

ld  r4, [sp, 8]
add r2, r2, r3
asl r2, r2, 2
sub r2, r2, r4
```

24 bytes

Basic Block

# Motivating Example

32-Bit Only

```
ld  r2, [sp, 0]
ld  r3, [sp, 4]

ld  r4, [sp, 8]
add r2, r2, r3
asl r2, r2, 2
sub r2, r2, r4
```

24 bytes

Mixed Mode  
Aggressive

```
ld_s r2, [sp, 0]
ld_s r3, [sp, 4]

ld_s rX, [sp, 8]
add_s r2, r2, r3
asl_s r2, r2, 2
sub_s r2, r2, rX
```

12 bytes

Basic Block

# Motivating Example

32-Bit Only

```
ld  r2, [sp, 0]
ld  r3, [sp, 4]

ld  r4, [sp, 8]
add r2, r2, r3
asl r2, r2, 2
sub r2, r2, r4
```

24 bytes

Mixed Mode  
Aggressive

```
ld_s r2, [sp, 0]
ld_s r3, [sp, 4]
mov  r4, r1
ld_s r1, [sp, 8]
add_s r2, r2, r3
asl_s r2, r2, 2
sub_s r2, r2, r1
mov  r4, r1
```

20 bytes

Basic Block

Instruction Selection

Register Allocation

# Motivating Example

## 32-Bit Only

```
ld  r2, [sp, 0]
ld  r3, [sp, 4]

ld  r4, [sp, 8]
add r2, r2, r3
asl r2, r2, 2
sub r2, r2, r4
```

24 bytes

## Mixed Mode Aggressive

```
ld_s r2, [sp, 0]
ld_s r3, [sp, 4]
mov  r4, r1
ld_s r1, [sp, 8]
add_s r2, r2, r3
asl_s r2, r2, 2
sub_s r2, r2, r1
mov  r4, r1
```

20 bytes

## Mixed Mode Integrated

```
ld_s r2, [sp, 0]
ld_s r3, [sp, 4]

ld  r4, [sp, 8]
add_s r2, r2, r3
asl_s r2, r2, 2
sub  r2, r2, r4
```

16 bytes

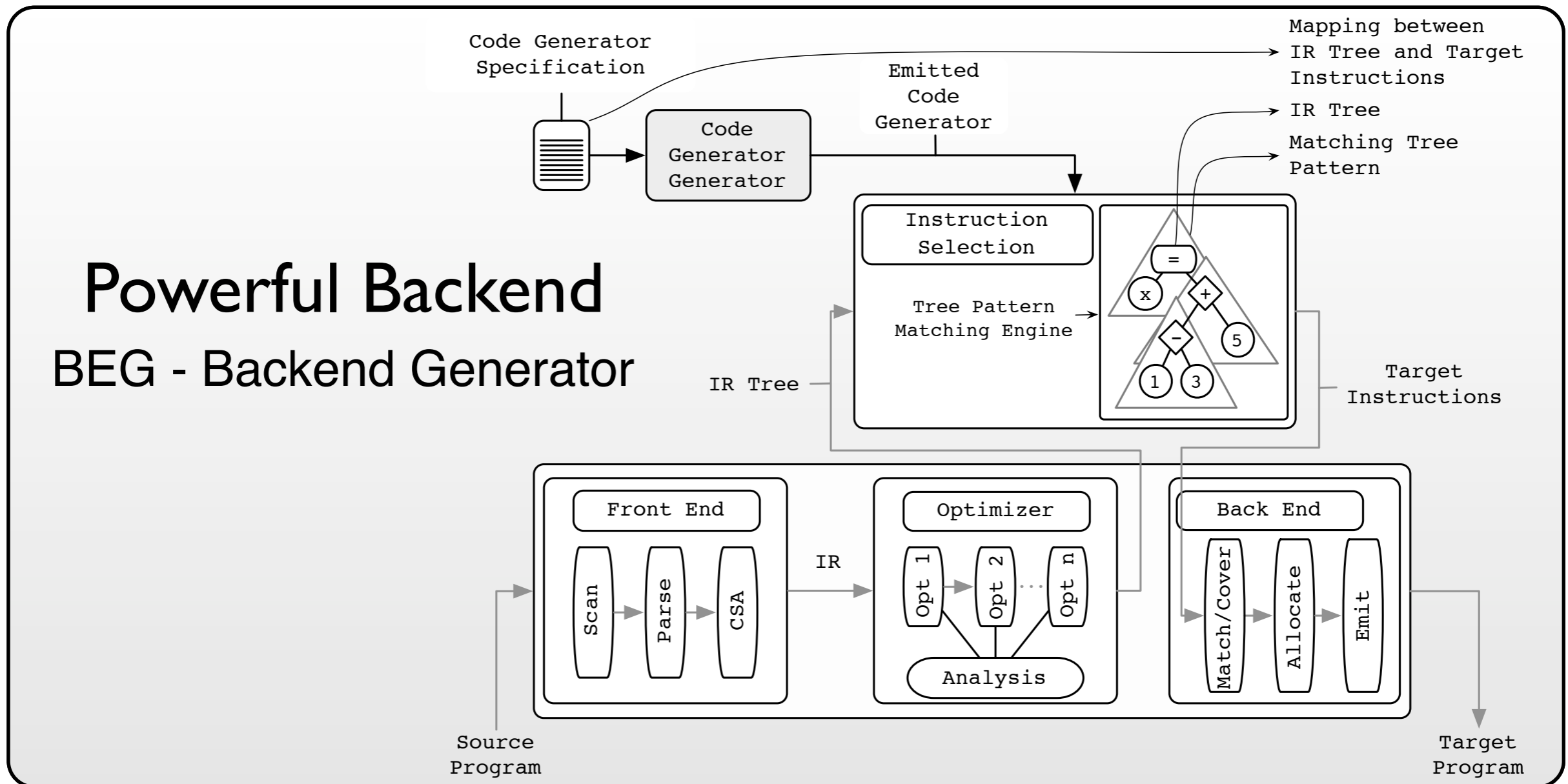
Basic Block



# Efficient Compact Code Generation is an **Integrated** Instruction Selection and Register Allocation Problem!

# Compact Code Generation Approach

**ECC** - EnCore **C** Compiler based on commercial **CoSy** compiler by **ACE**©.





# Compact Code Generation Approach

**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

Opportunistic

Feedback-Guided



# Compact Code Generation Approach

**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

Opportunistic

MIR ↓

**Match / Cover**  
32-bit patterns only

LIR ↓

**Register Allocation**

LIR ↓

**Code Emission**  
32-bit/16-bit instructions

ASM ↓

Feedback-Guided





# Compact Code Generation Approach

**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

Opportunistic

Feedback-Guided

MIR ↓

**Match / Cover**  
32-bit patterns only

LIR ↓

**Register Allocation**

LIR ↓

**Code Emission**  
32-bit/16-bit instructions

ASM ↓

# Compact Code Generation Approach

**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

Opportunistic

Feedback-Guided

MIR ↓

**Match / Cover**  
32-bit patterns only

1

LIR ↓

**Register Allocation**

2

LIR ↓

**Code Emission**  
32-bit/16-bit instructions

ASM ↓

# Compact Code Generation Approach

**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

Opportunistic

Feedback-Guided

MIR ↓

**Match / Cover**  
32-bit patterns only

1

LIR ↓

**Register Allocation**

2

LIR ↓

**Code Emission**  
32-bit/16-bit instructions

3

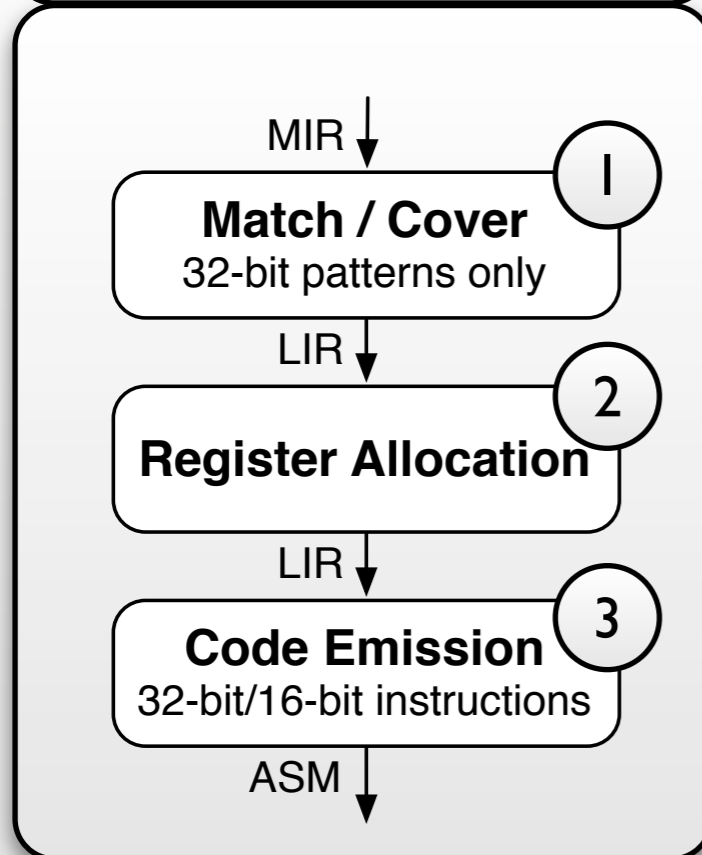
ASM ↓

# Compact Code Generation Approach

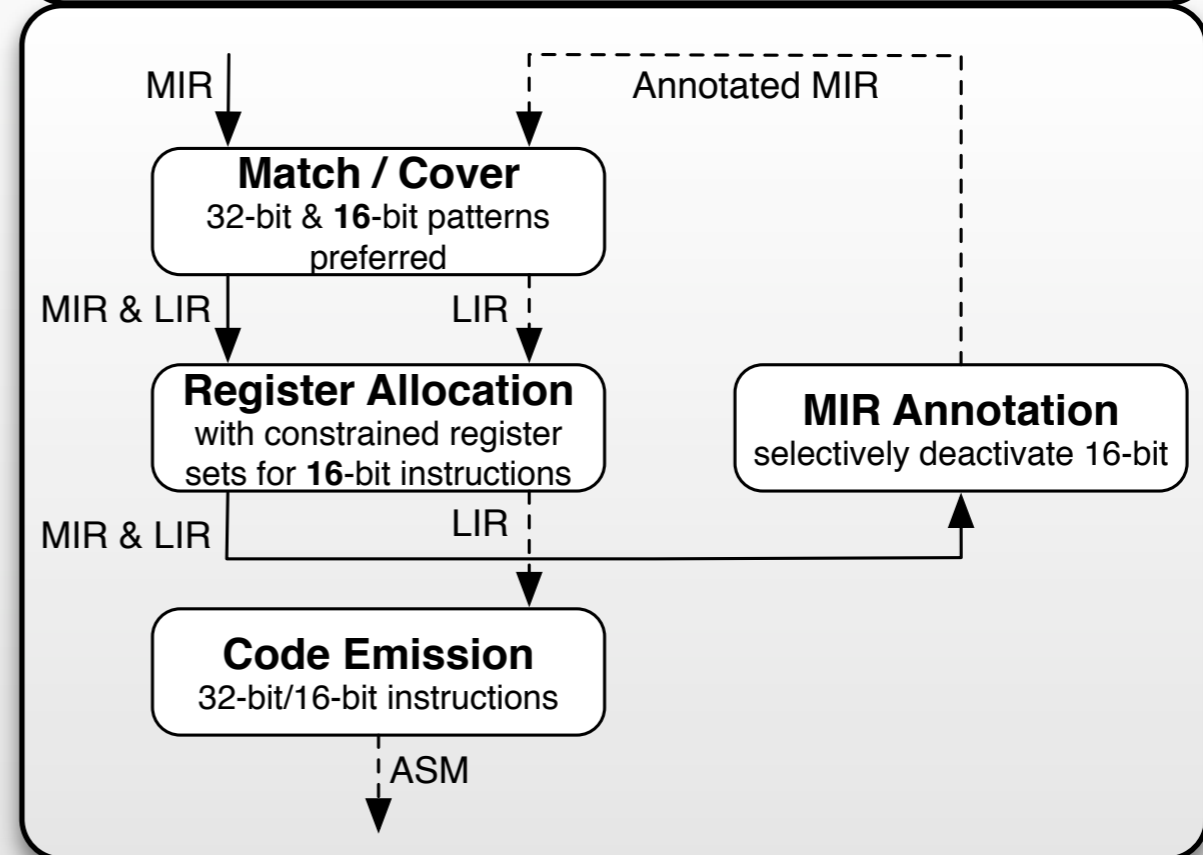
**ECC** - **EnCore C C**ompiler based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided

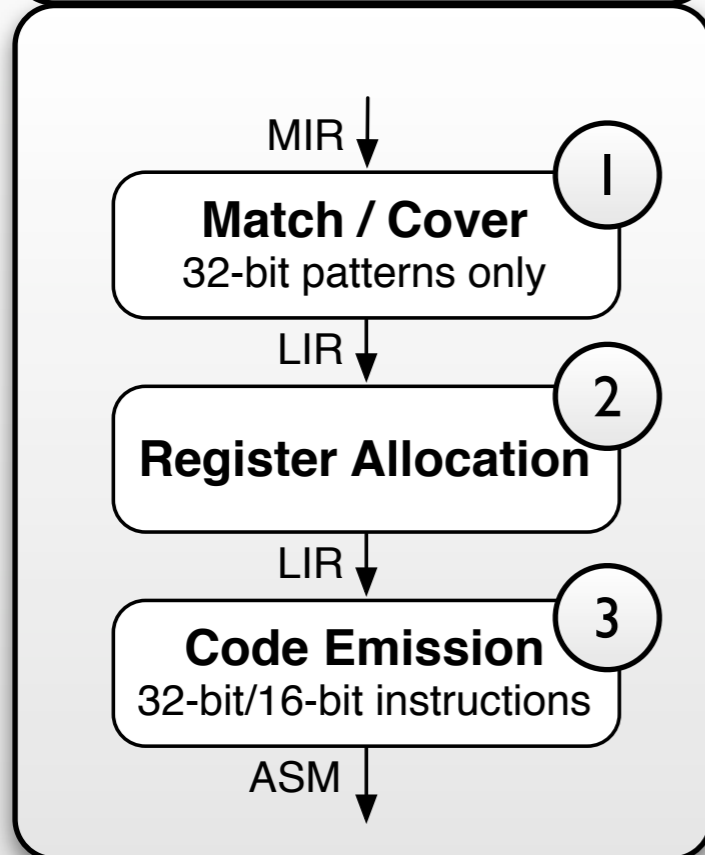


# Compact Code Generation Approach

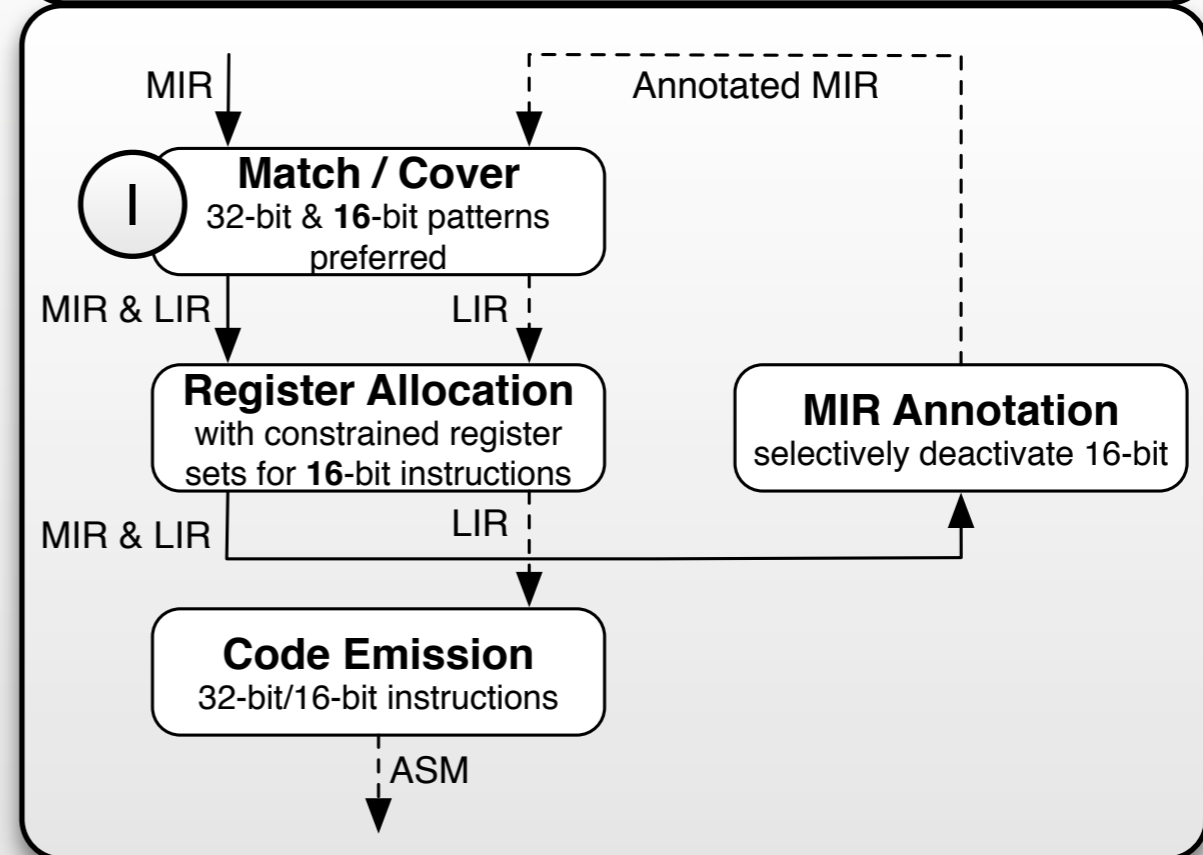
**ECC** - **EnCore C C**ompiler based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided

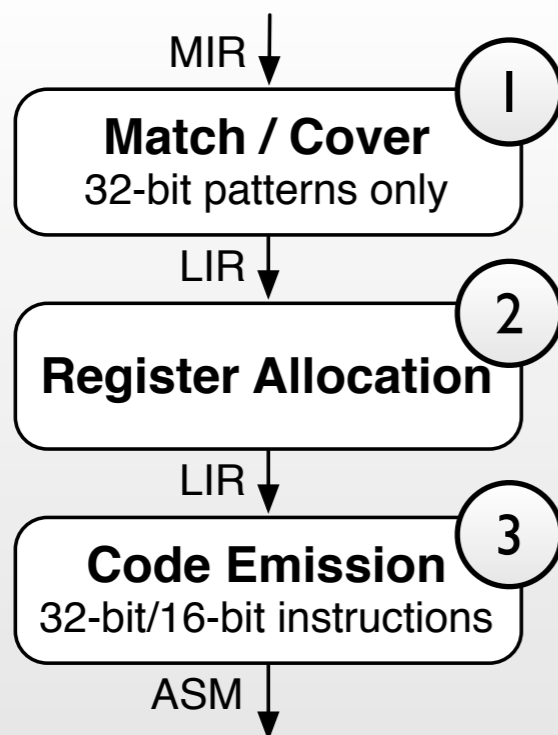


# Compact Code Generation Approach

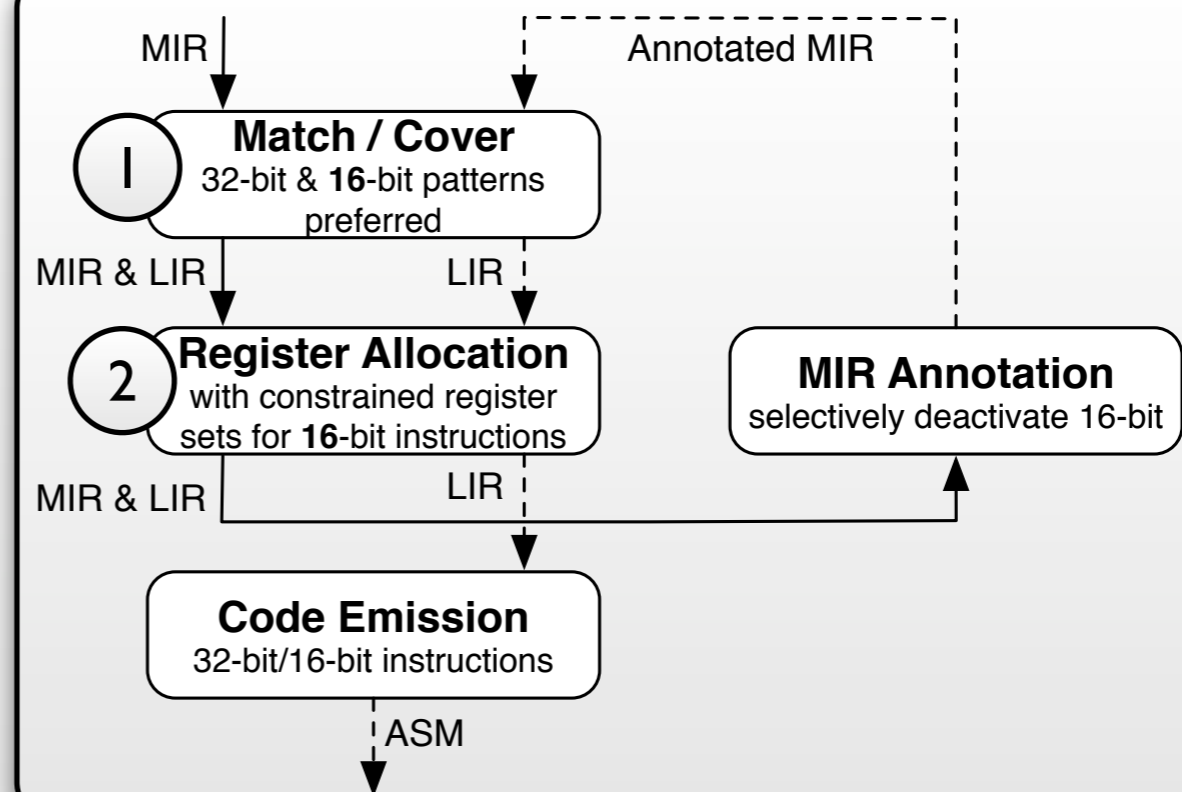
**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided

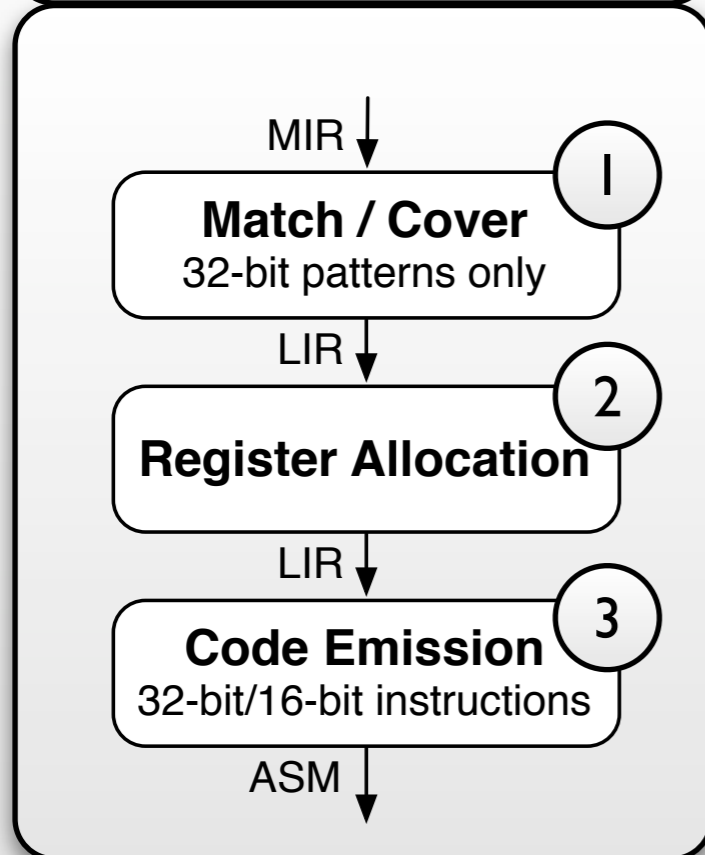


# Compact Code Generation Approach

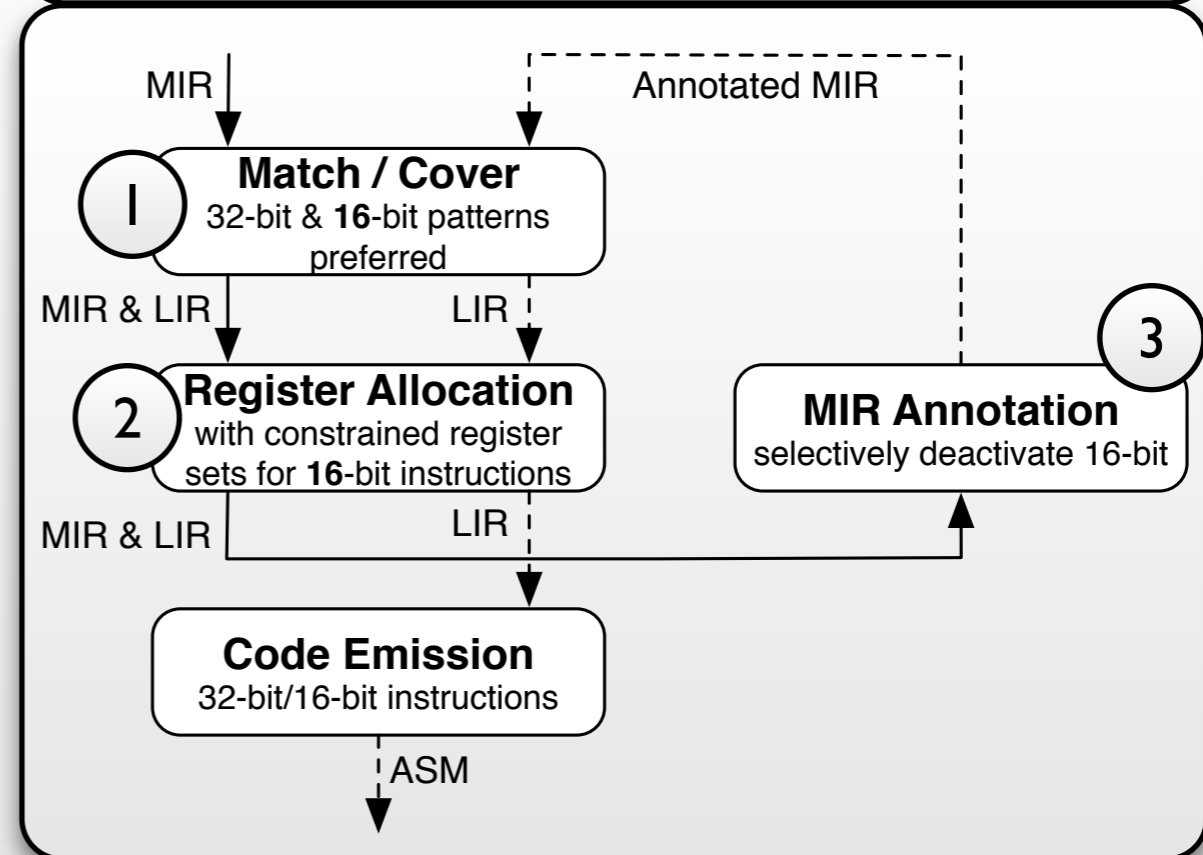
**ECC** - **EnCore C C**ompiler based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided

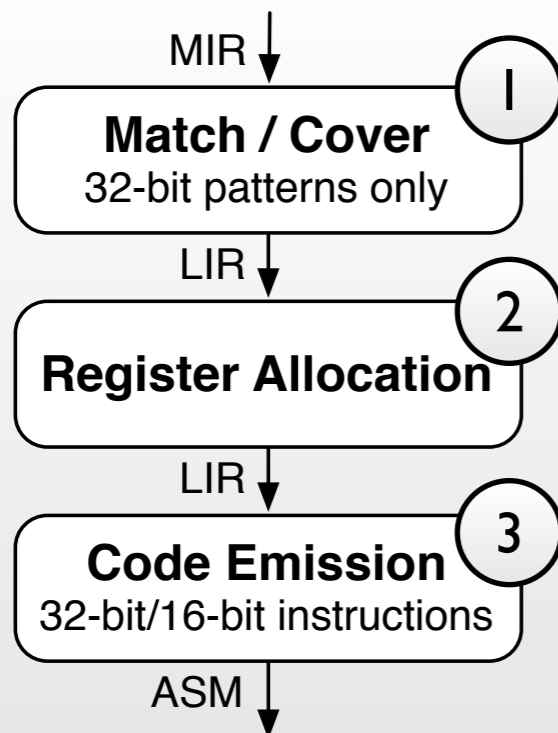


# Compact Code Generation Approach

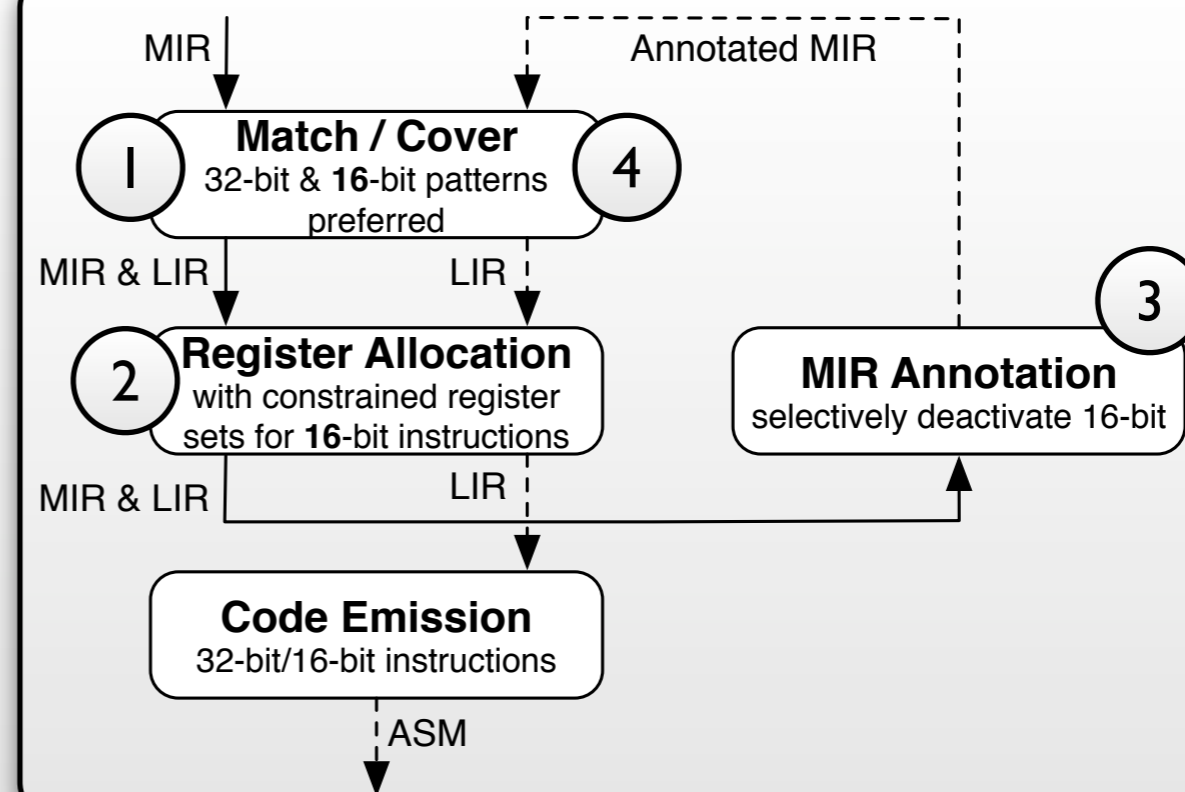
**ECC** - **EnCore C C**ompiler based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided



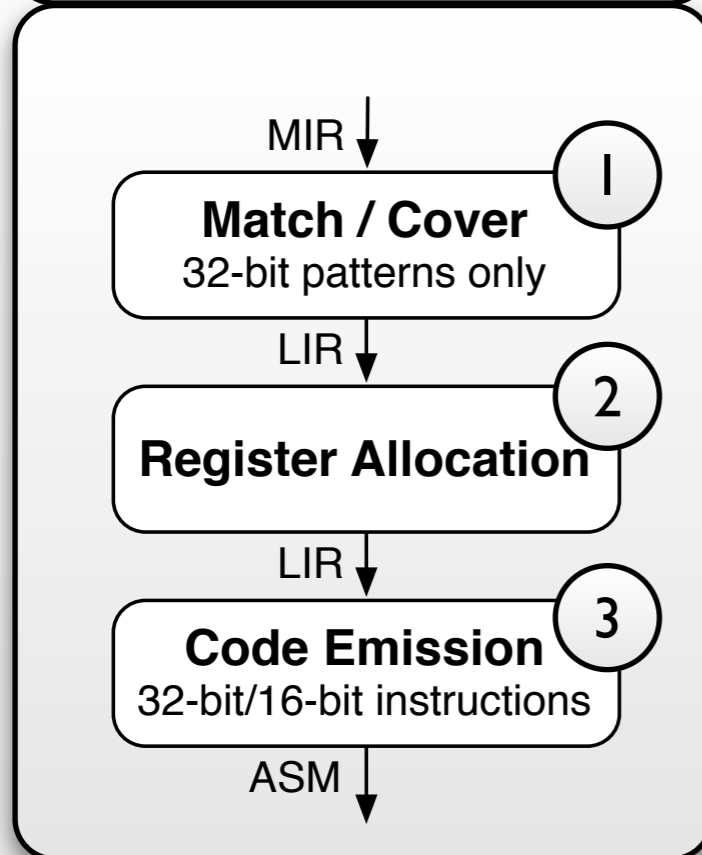


# Compact Code Generation Approach

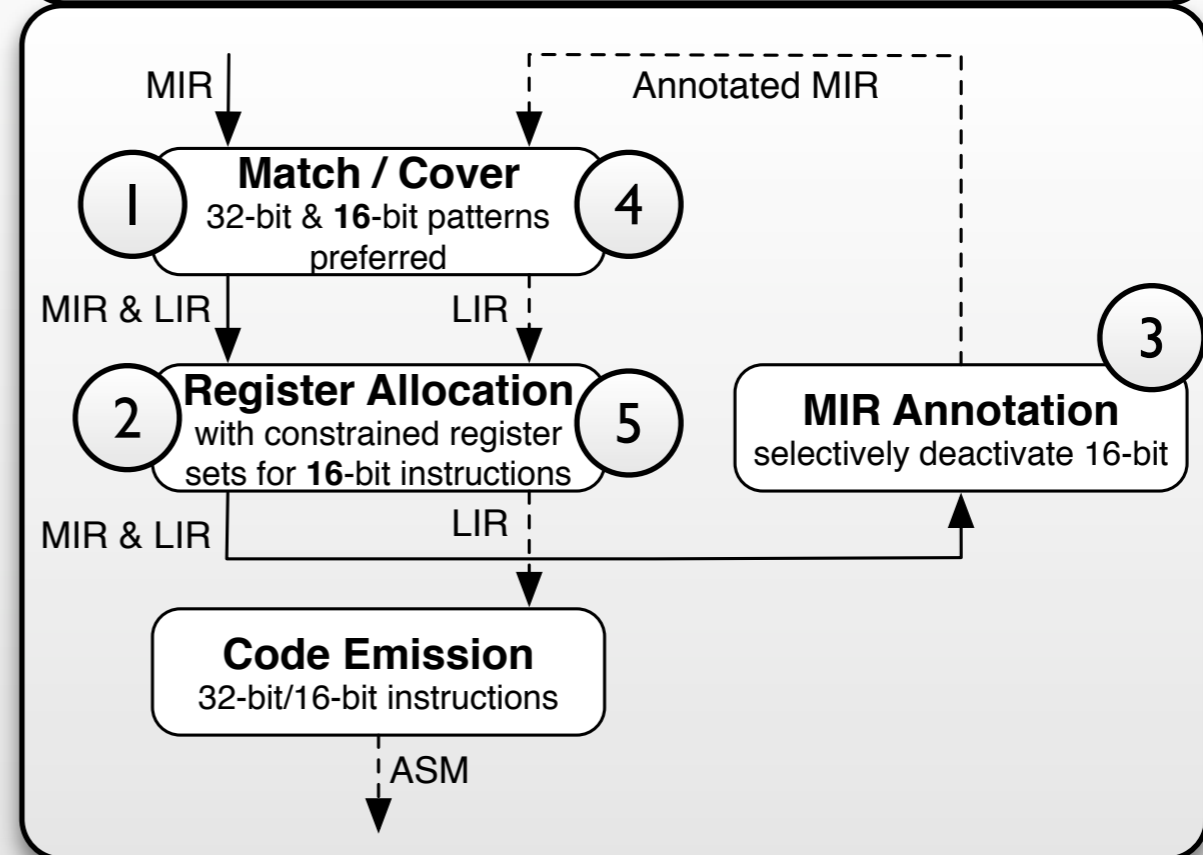
**ECC** - **EnCore C C**ompiler based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided

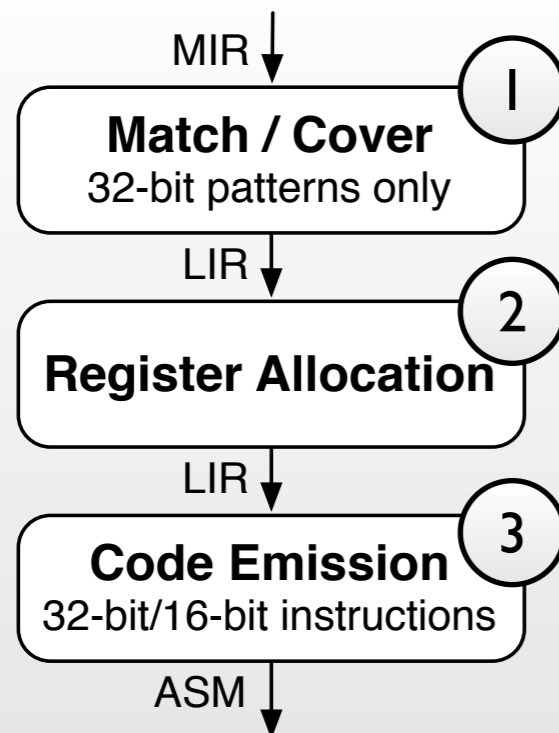


# Compact Code Generation Approach

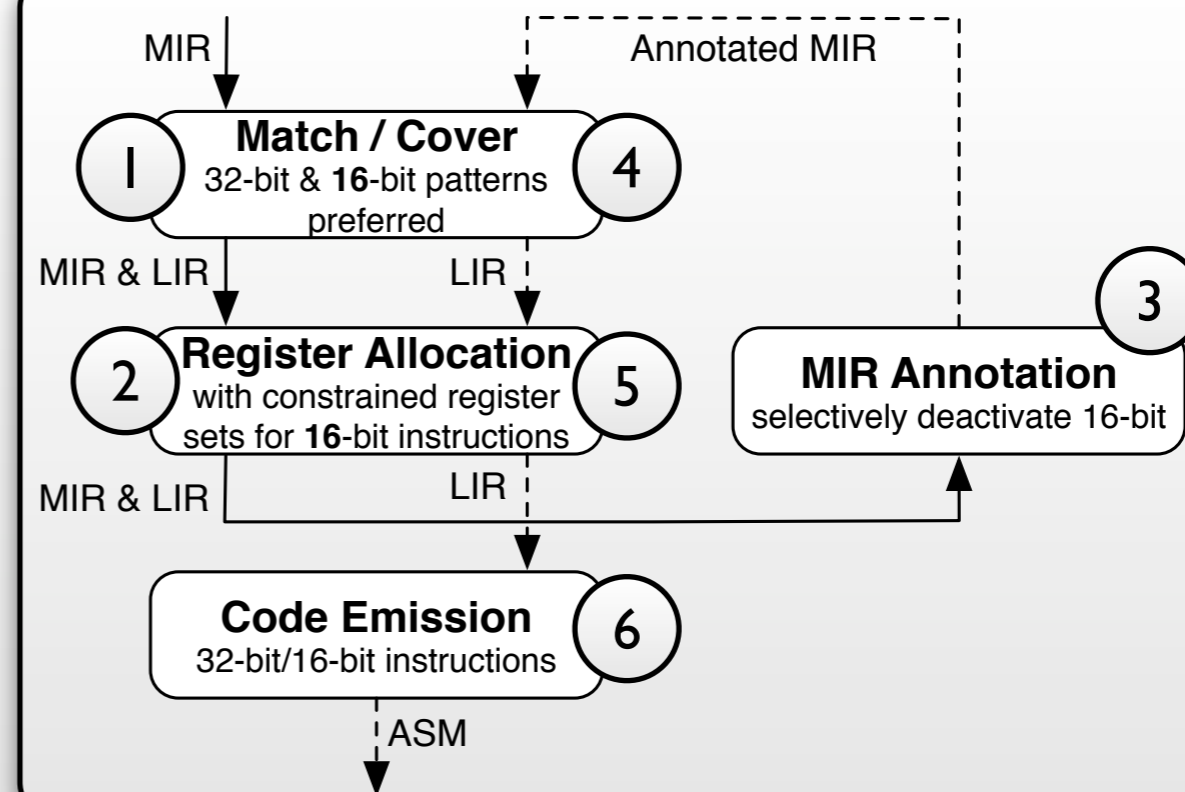
**ECC** - **EnCore C Compiler** based on commercial **CoSy** compiler by **ACE**©.

Compiler backend supports **two** compact code generation strategies

## Opportunistic



## Feedback-Guided





# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register

**MIR**

```
ld v2,x
ld v3,y

ld v4,z
add v5,v2,v3
asl v6,v5,2
sub v7,v6,v4
```

MIR →

# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register

## MIR

```
ld v2,x  
ld v3,y  
  
ld v4,z  
add v5,v2,v3  
asl v6,v5,2  
sub v7,v6,v4
```

MIR

## MIR Annotation

### Match/Cover

```
ld_s v2,[sp,0]  
ld_s v3,[sp,4]  
  
ld_s v4,[sp,8]  
add_s v5,v2,v3  
asl_s v6,v5,2  
sub_s v7,v6,v4
```

MIR  
LIR

### Register Allocation & MIR Annotation

aggressively select 16-bit  
instructions

# Feedback-Guided Instruction Selection

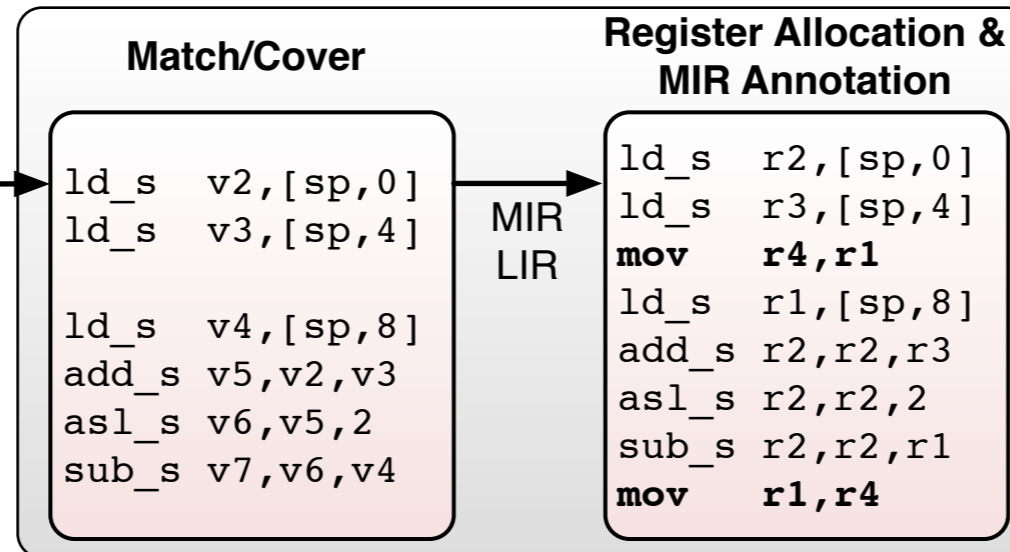
vX ... Virtual Register  
rX ... Physical Register

**MIR**

```
ld v2,x
ld v3,y

ld v4,z
add v5,v2,v3
asl v6,v5,2
sub v7,v6,v4
```

MIR



register allocator constrained  
to 16-bit accessible  
register set

aggressively select 16-bit  
instructions

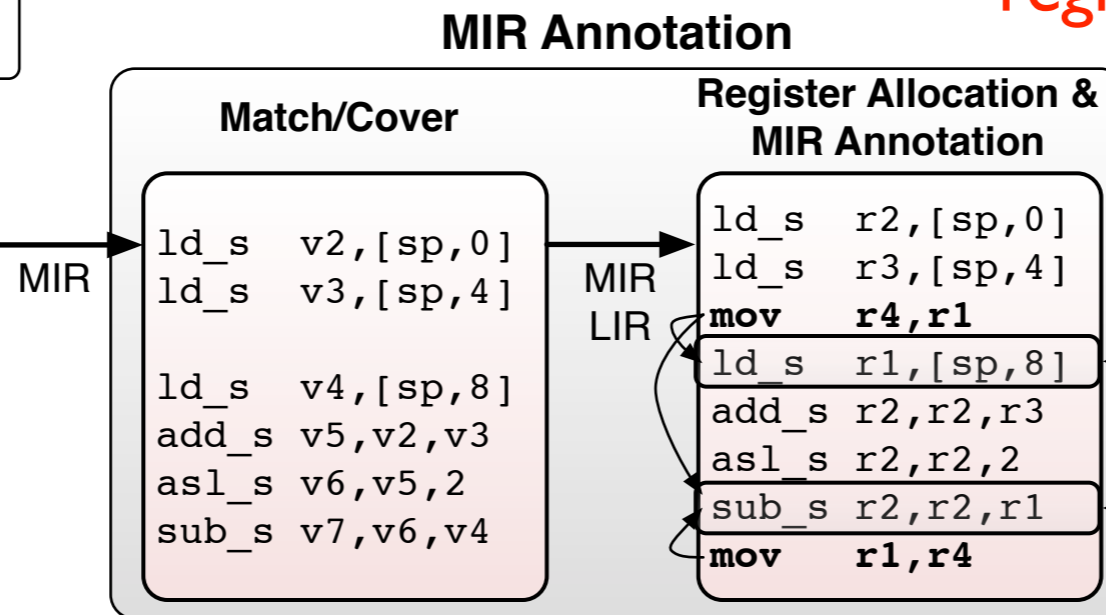
# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register

**MIR**

```
ld v2,x
ld v3,y

ld v4,z
add v5,v2,v3
asl v6,v5,2
sub v7,v6,v4
```



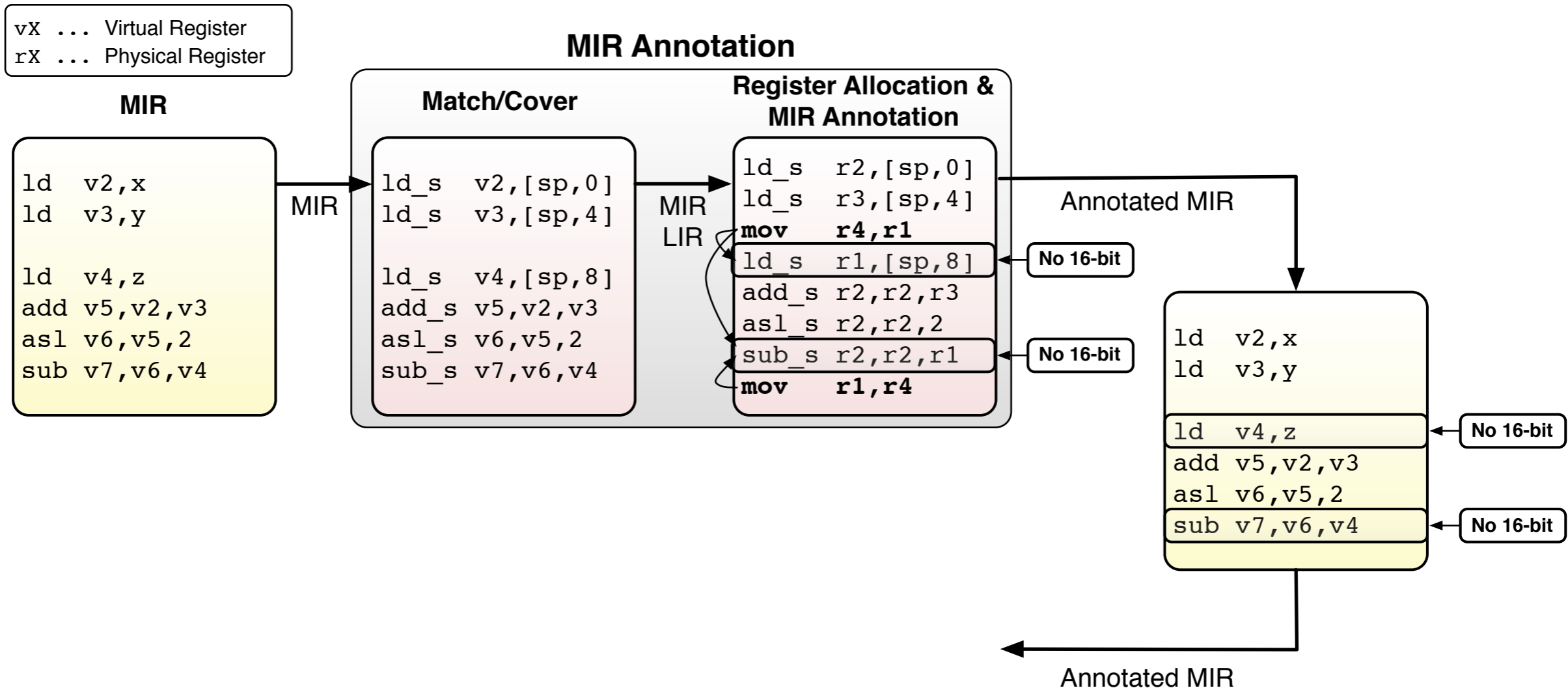
register allocator constrained to 16-bit accessible register set

No 16-bit

No 16-bit

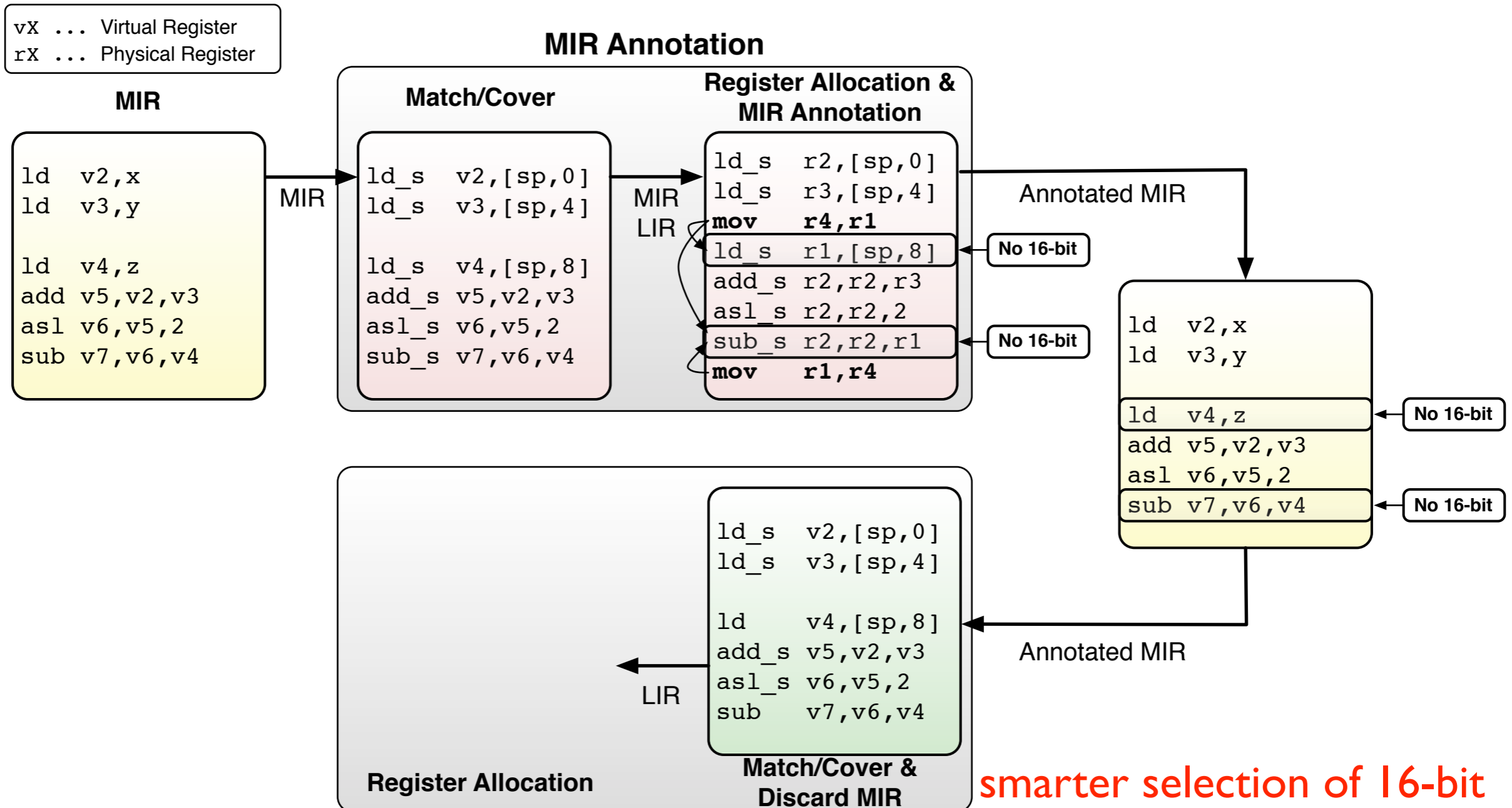
# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register



# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register

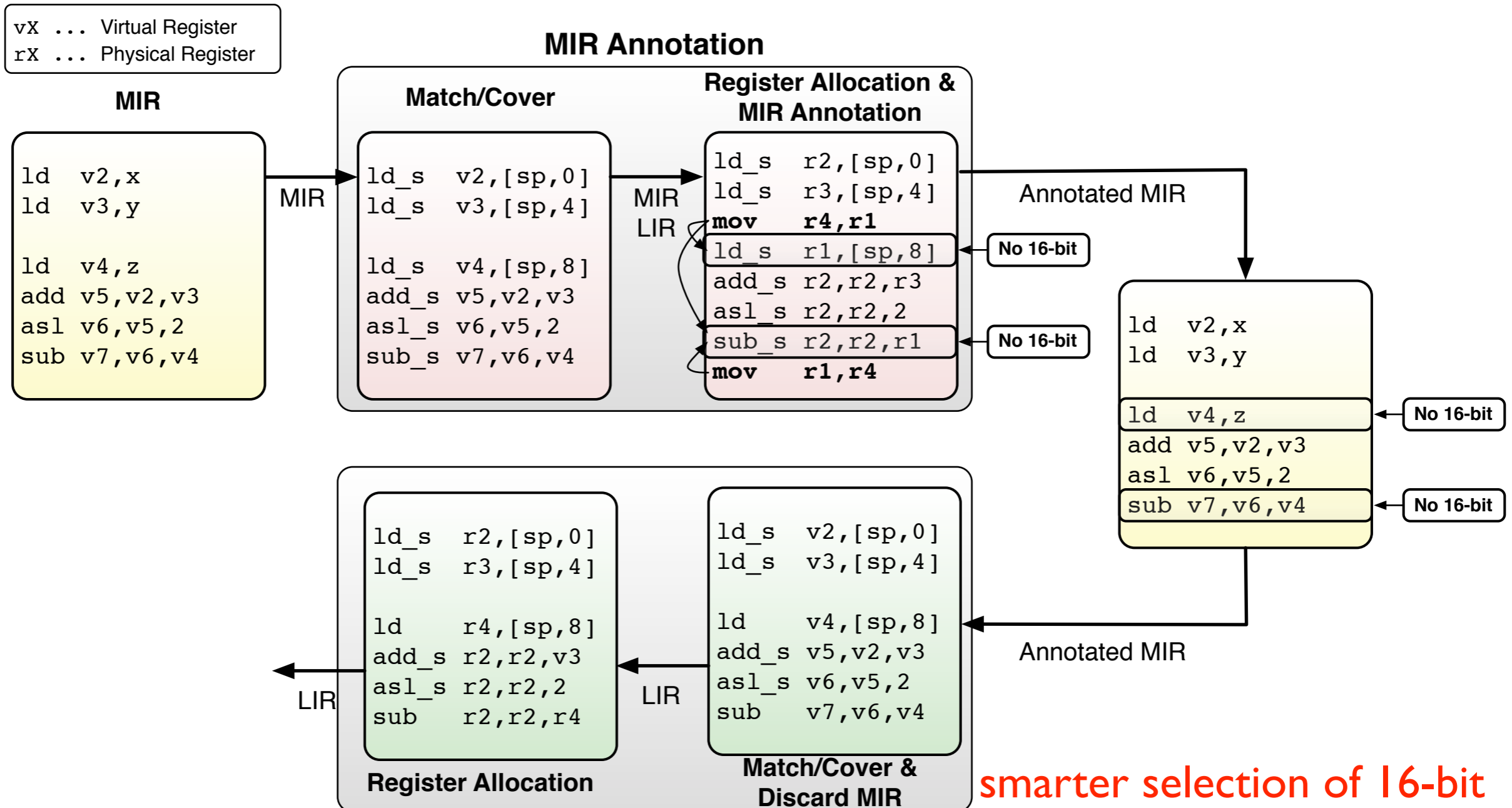


smarter selection of 16-bit instructions based on feedback



# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register

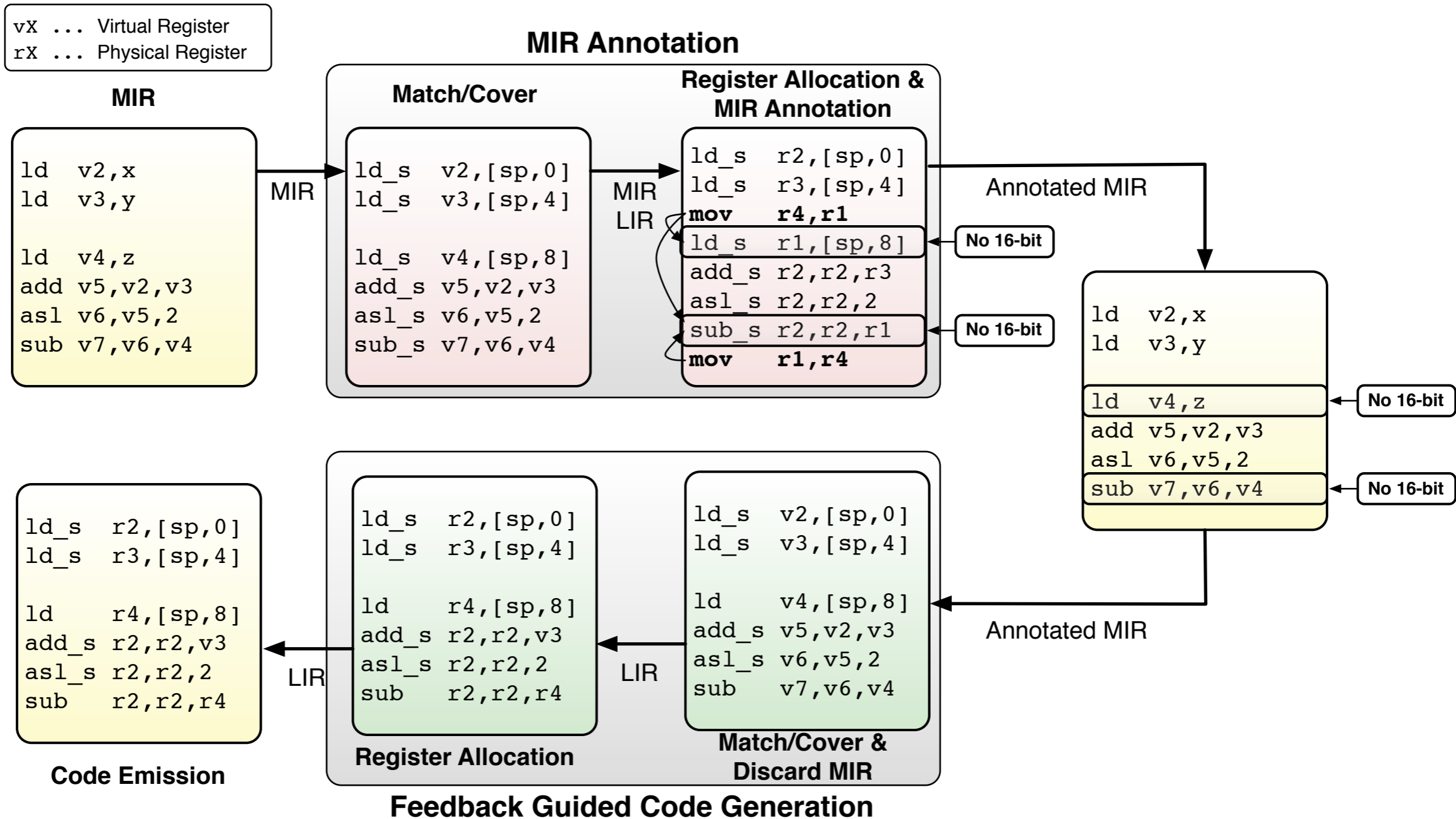


fewer constraints for  
register allocator

smarter selection of 16-bit  
instructions based on feedback

# Feedback-Guided Instruction Selection

vX ... Virtual Register  
rX ... Physical Register



fewer constraints for  
register allocator

# Evaluation - Experimental Setup

<b>BenchMarks</b>	<b>EEMBC I.I</b>
-------------------	------------------

<b>Core</b>	<b>ARC750D</b>
Pipeline	7-stage interlocked
Execution Order	In-Order
Branch Prediction	Yes
ISA	ARCompact
Floating Point	Hardware
<b>Memory Subsystem</b>	
L1 Cache	Yes
Instruction	8k/2-way associative
Data	8k/2-way associative
L2 Cache	No

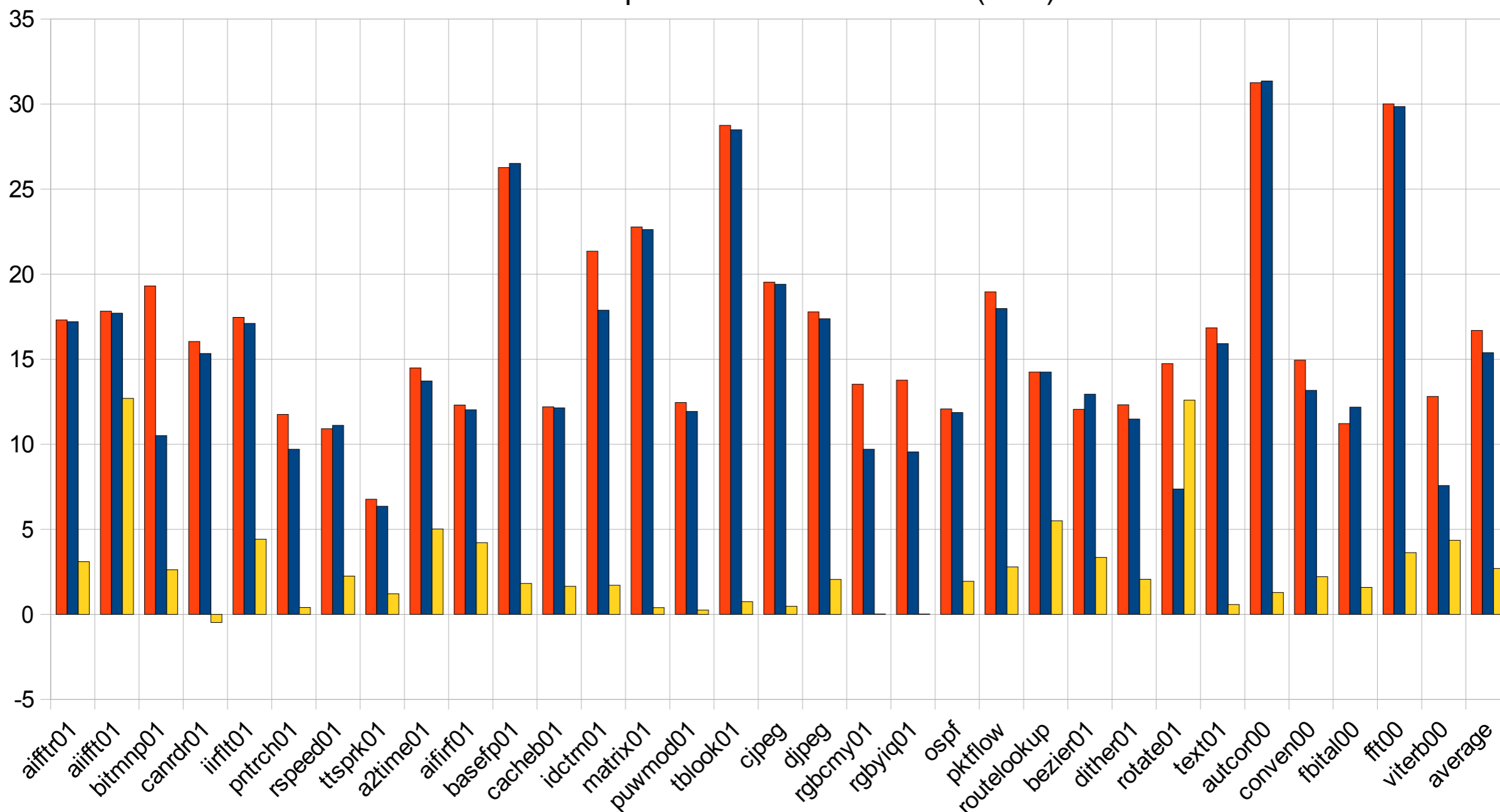


<b>Simulator</b>	<b>ArcSim</b>
Simulation Mode	Full System, Cycle Accurate
Accuracy	Cycle accurate mode validated against real HW
Options	Default
I/O & System Calls	Emulated



# Evaluation - Code Size Reduction

Improvement in Code Size (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 16.7%)

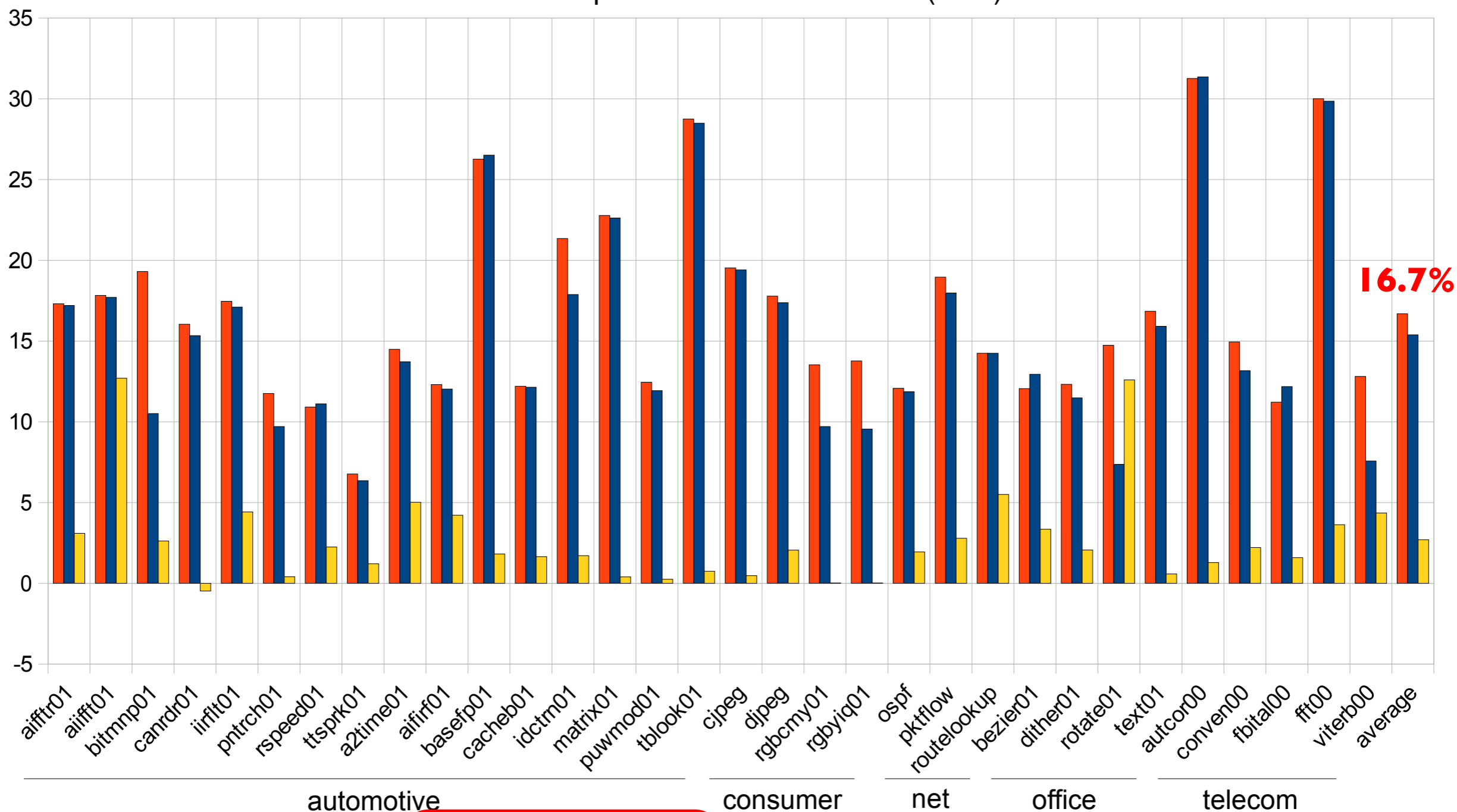
Opportunistic selection (avg: 15.4%)

GCC (avg: 2.7%)



# Evaluation - Code Size Reduction

Improvement in Code Size (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 16.7%)

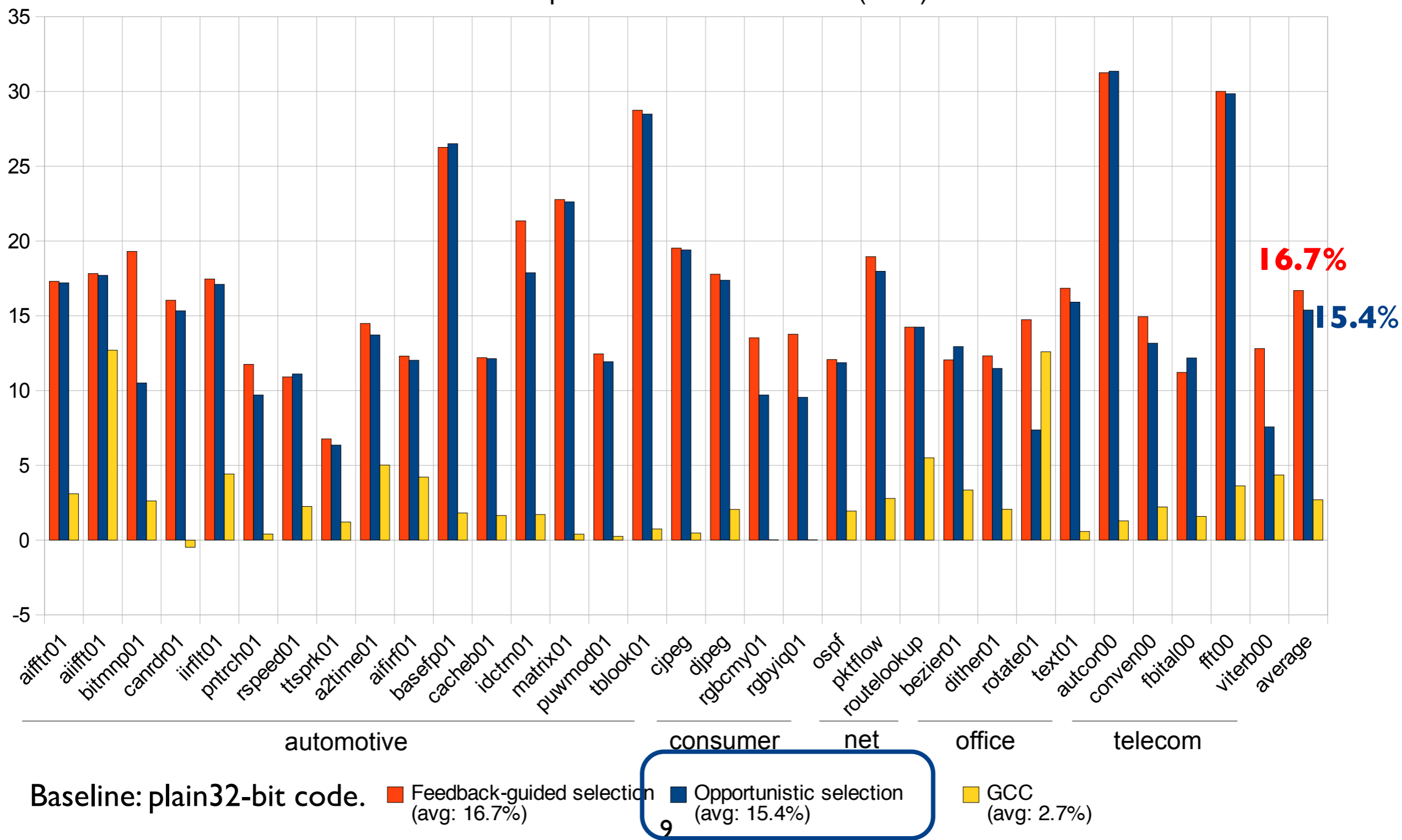
Opportunistic selection (avg: 15.4%)

GCC (avg: 2.7%)



# Evaluation - Code Size Reduction

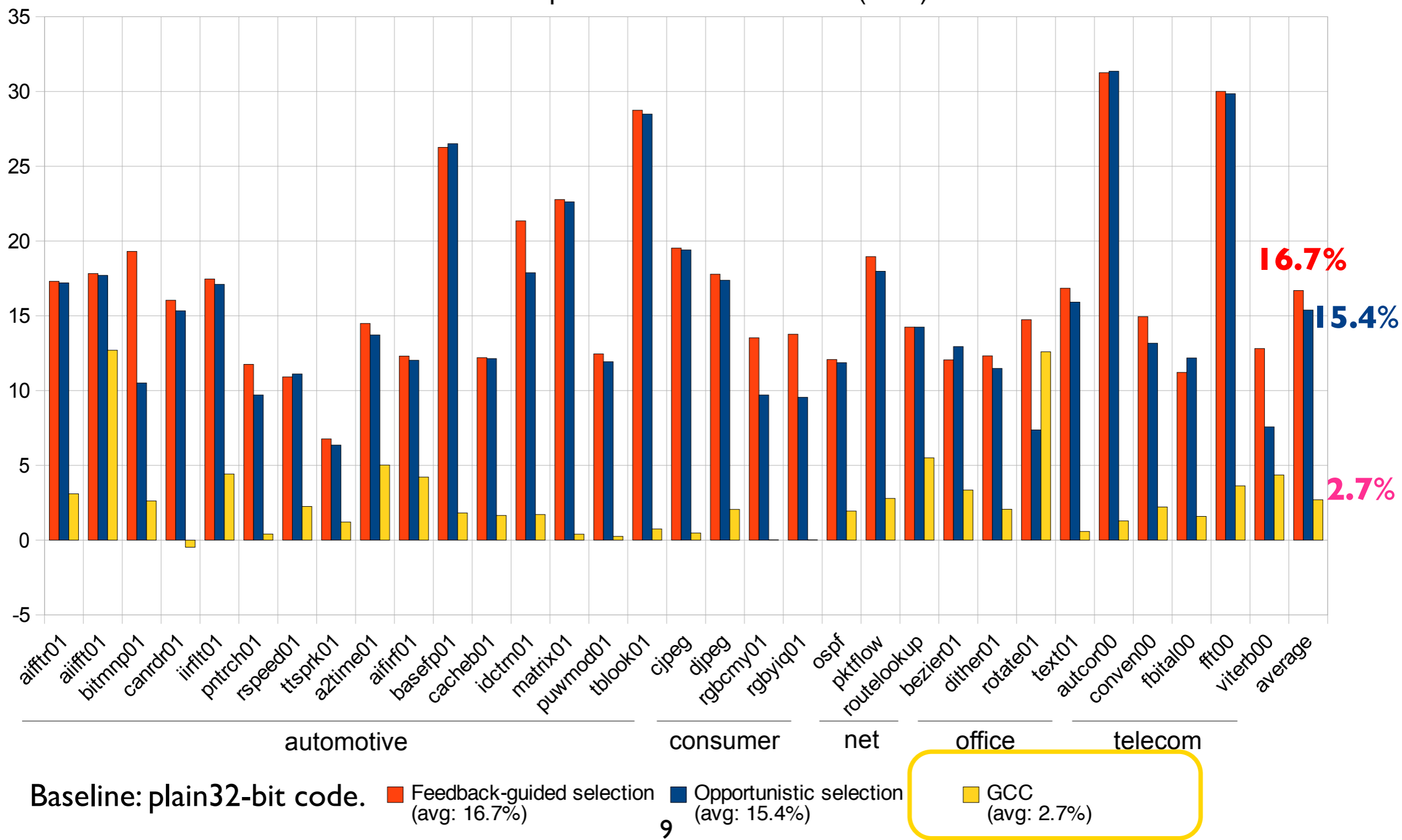
Improvement in Code Size (in %)





# Evaluation - Code Size Reduction

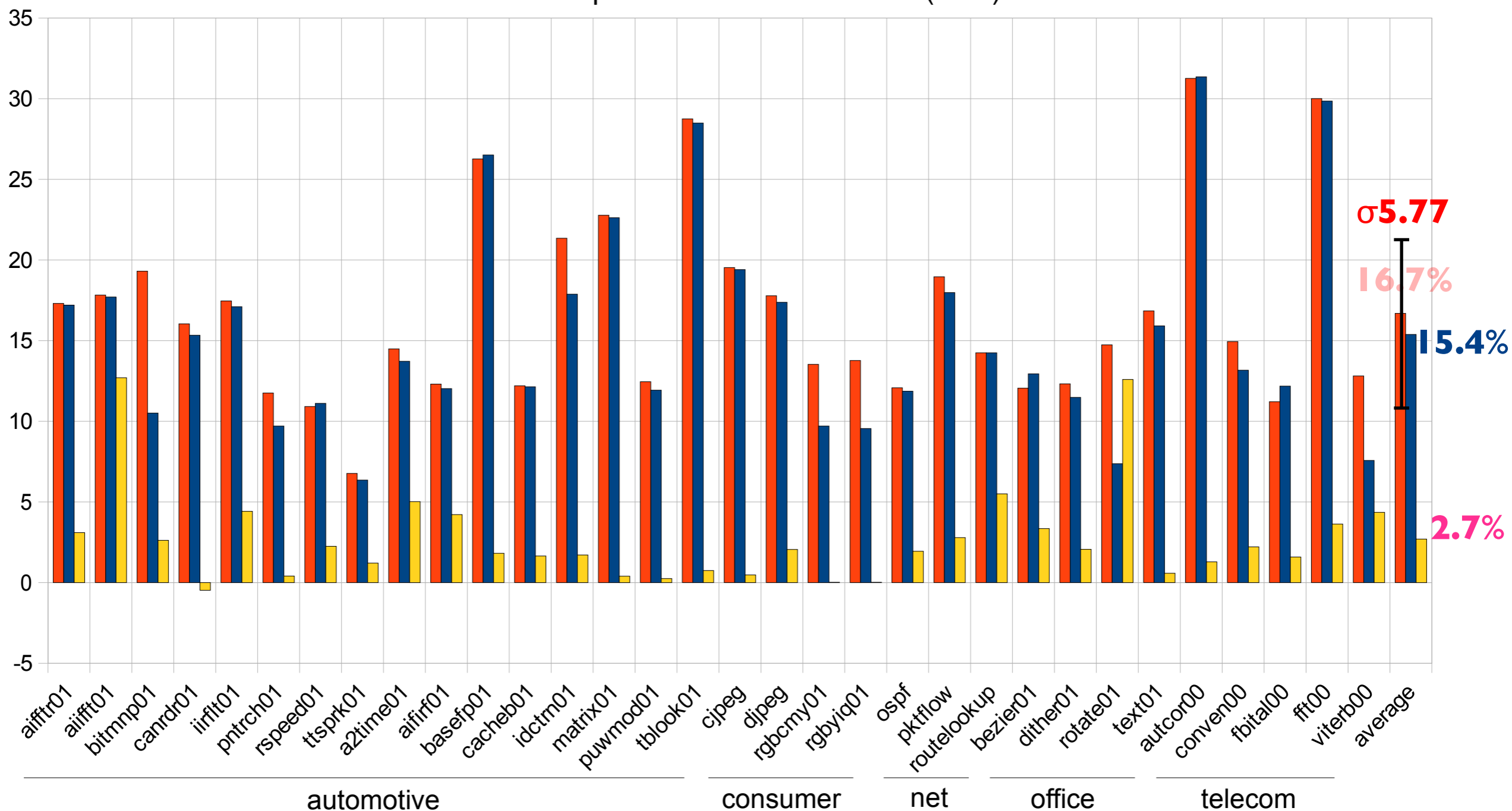
Improvement in Code Size (in %)





# Evaluation - Code Size Reduction

Improvement in Code Size (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 16.7%)

Opportunistic selection (avg: 15.4%)

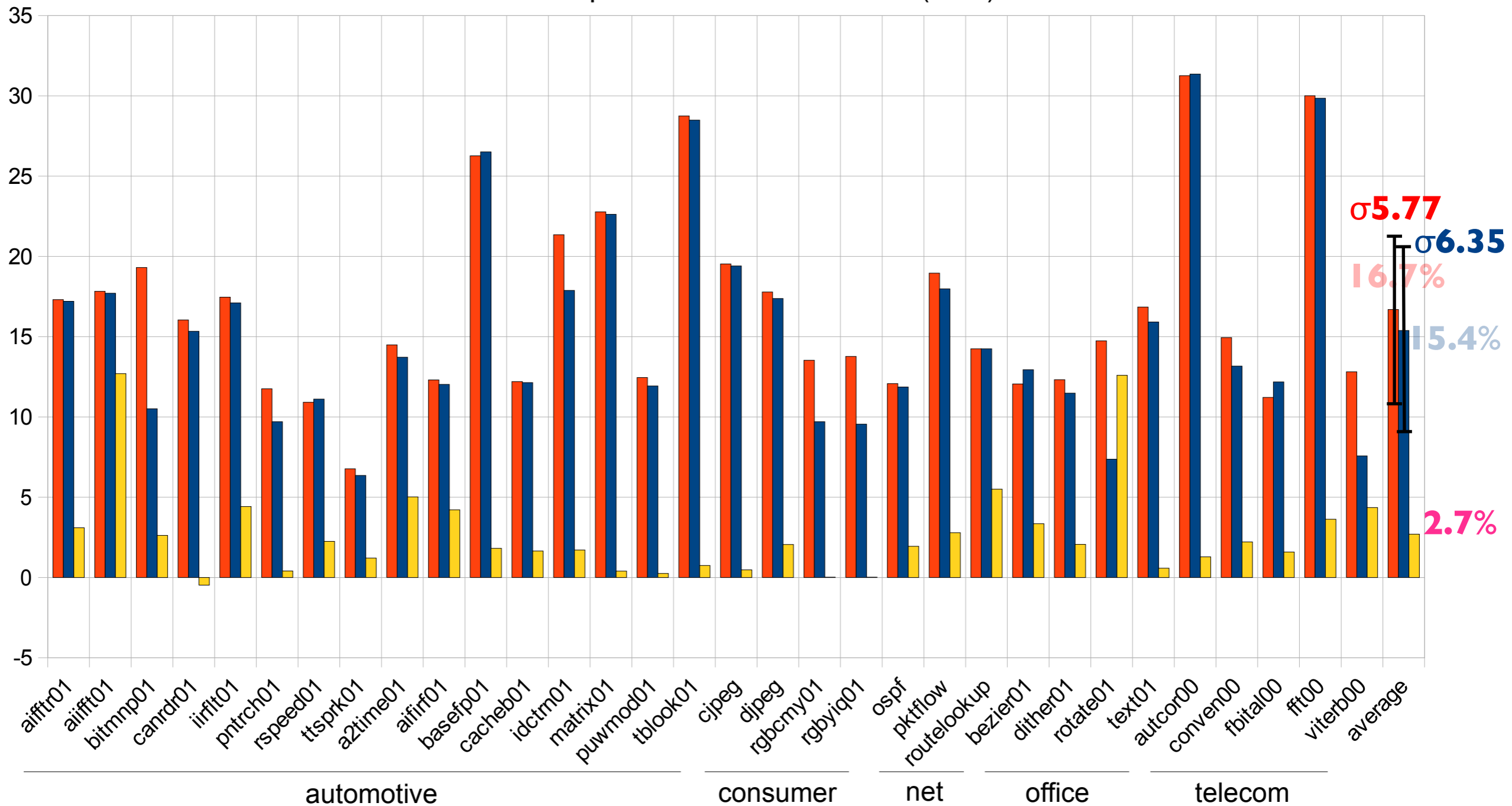
GCC (avg: 2.7%)





# Evaluation - Code Size Reduction

Improvement in Code Size (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 16.7%)

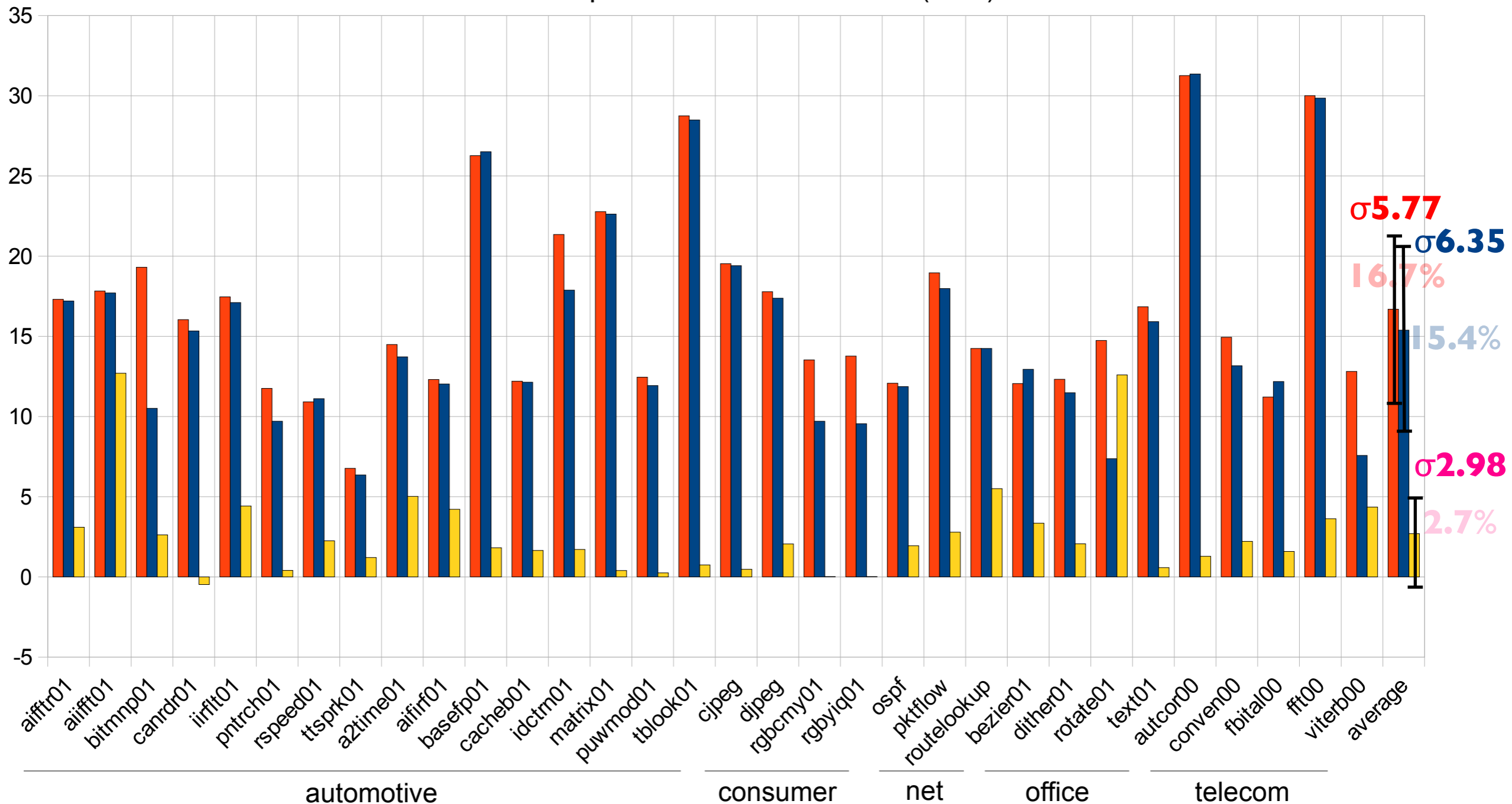
Opportunistic selection (avg: 15.4%)

GCC (avg: 2.7%)



# Evaluation - Code Size Reduction

Improvement in Code Size (in %)



Baseline: plain32-bit code.

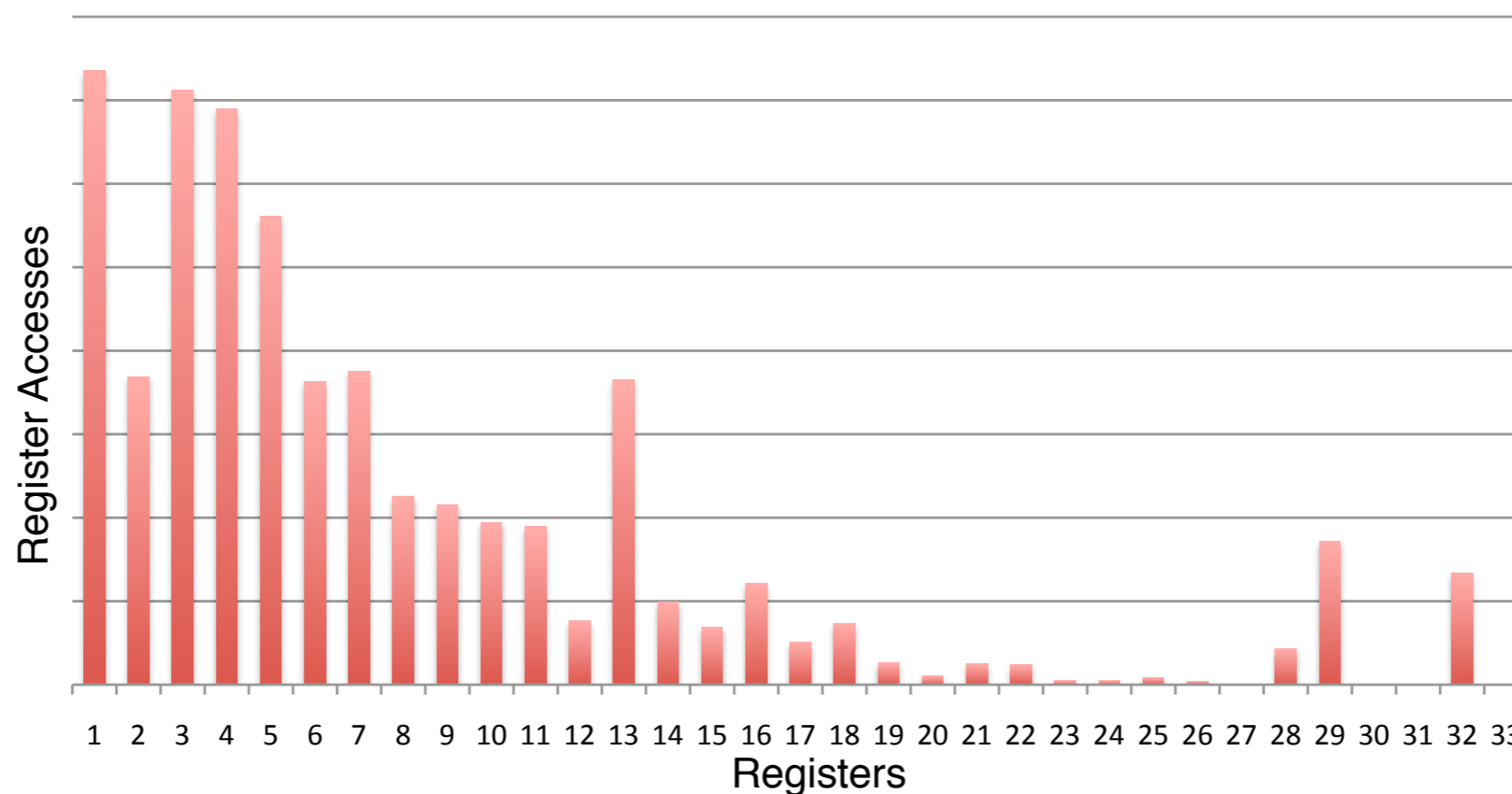
Feedback-guided selection (avg: 16.7%)

Opportunistic selection (avg: 15.4%)

GCC (avg: 2.7%)

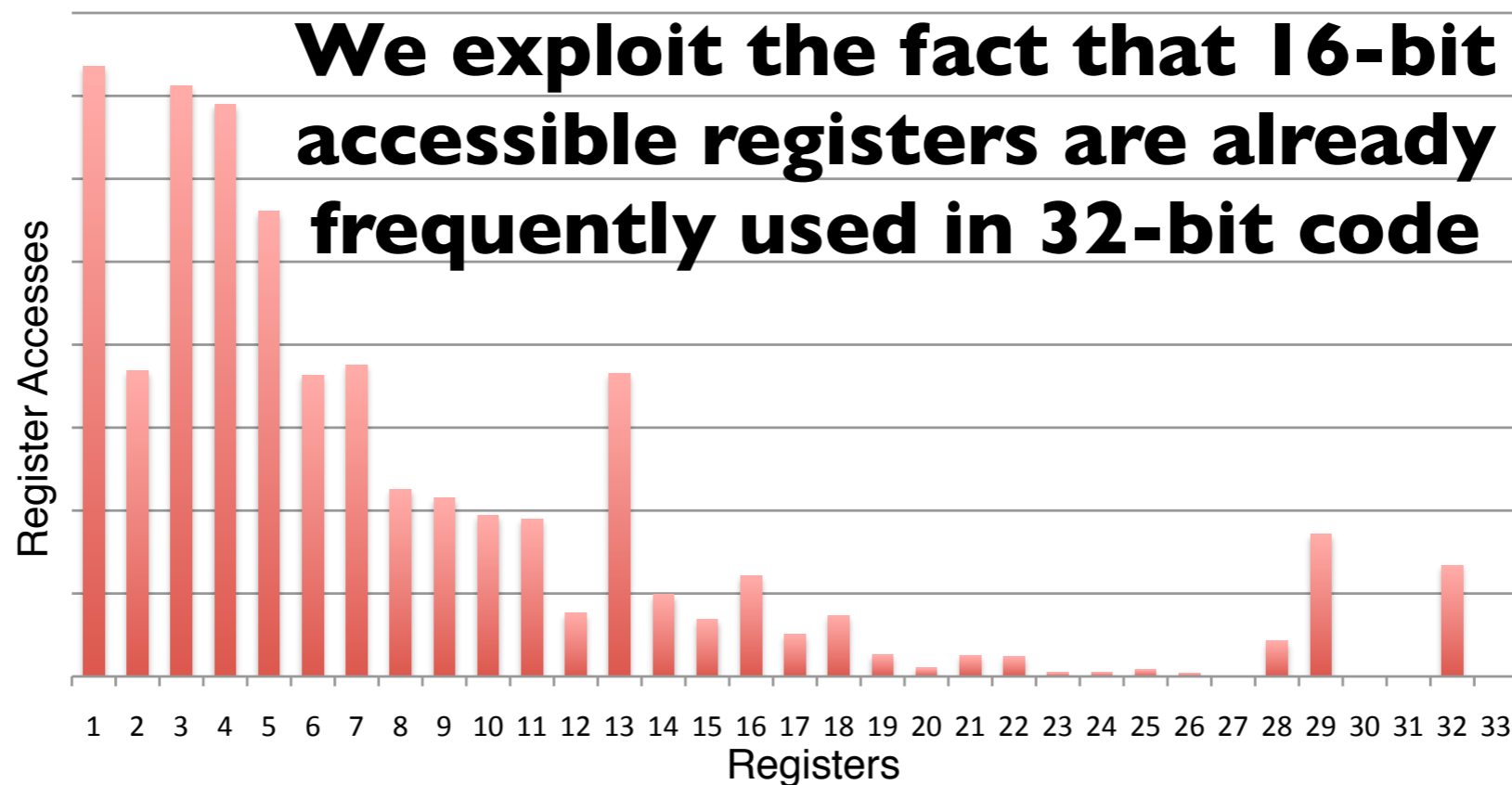


# Why does the simple Opportunistic Mode perform so well?



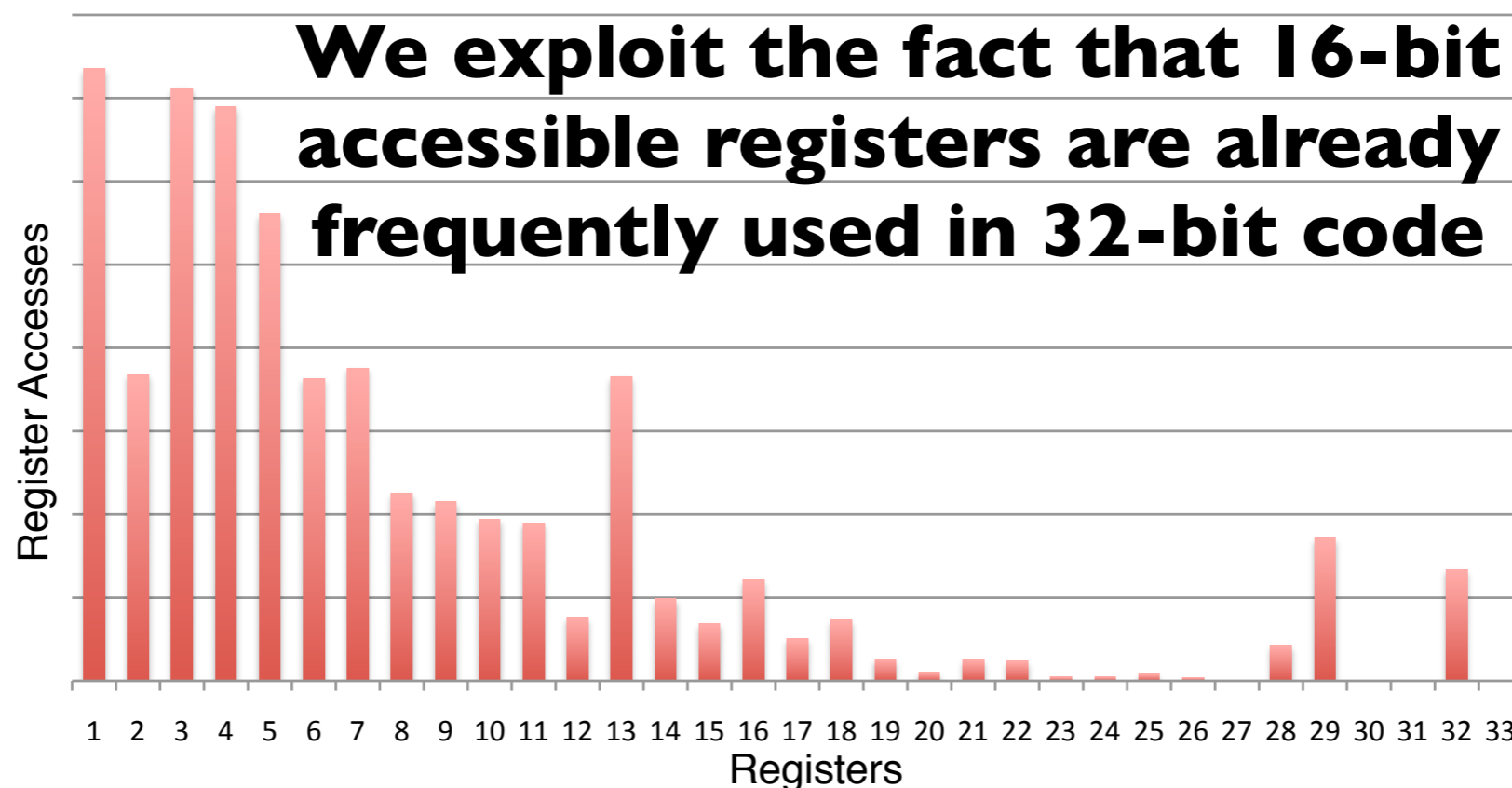


# Why does the simple Opportunistic Mode perform so well?



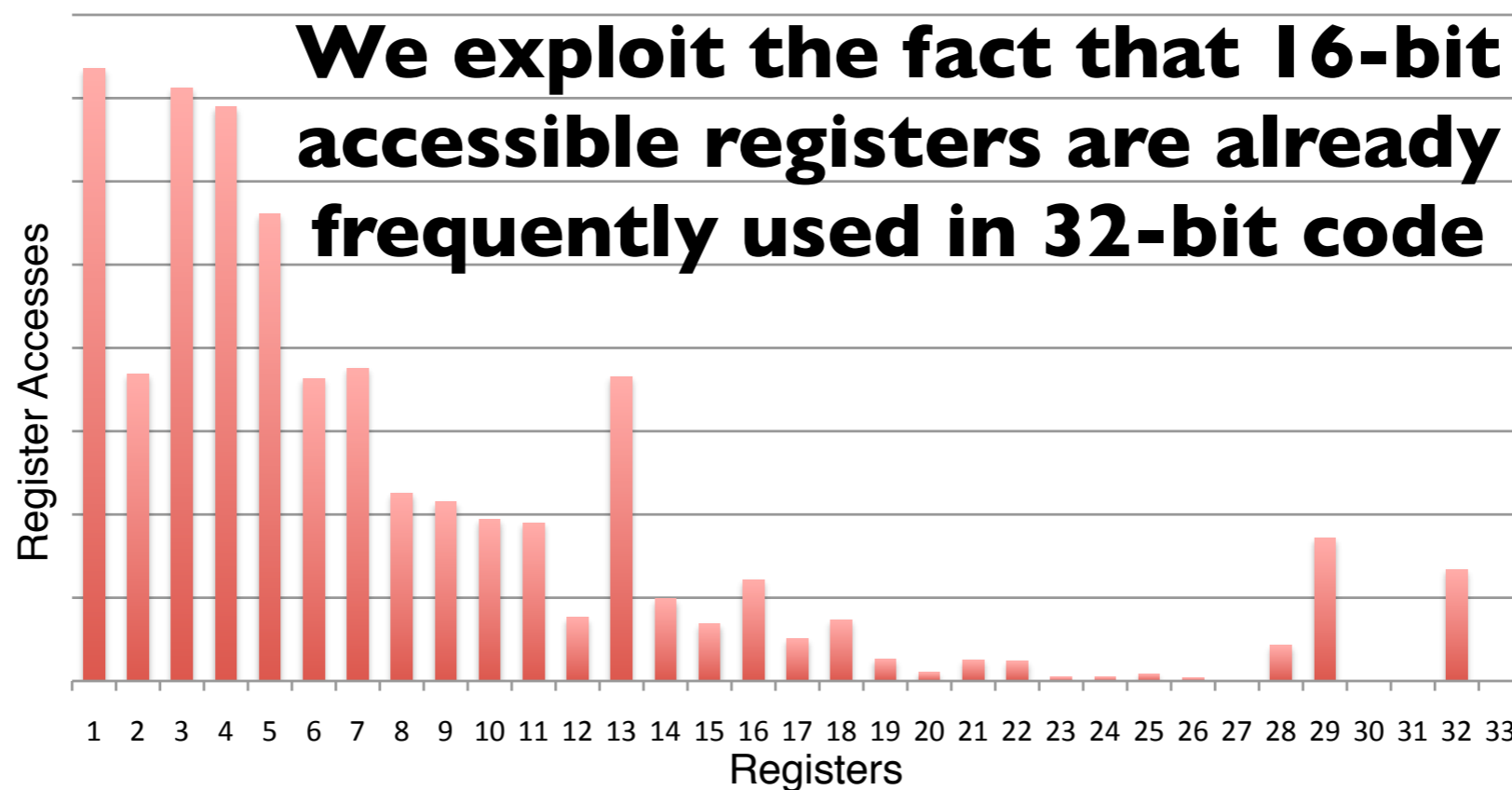
# Why does the simple Opportunistic Mode perform so well?

- register allocator selects registers with lower ID from set of possible registers



# Why does the simple Opportunistic Mode perform so well?

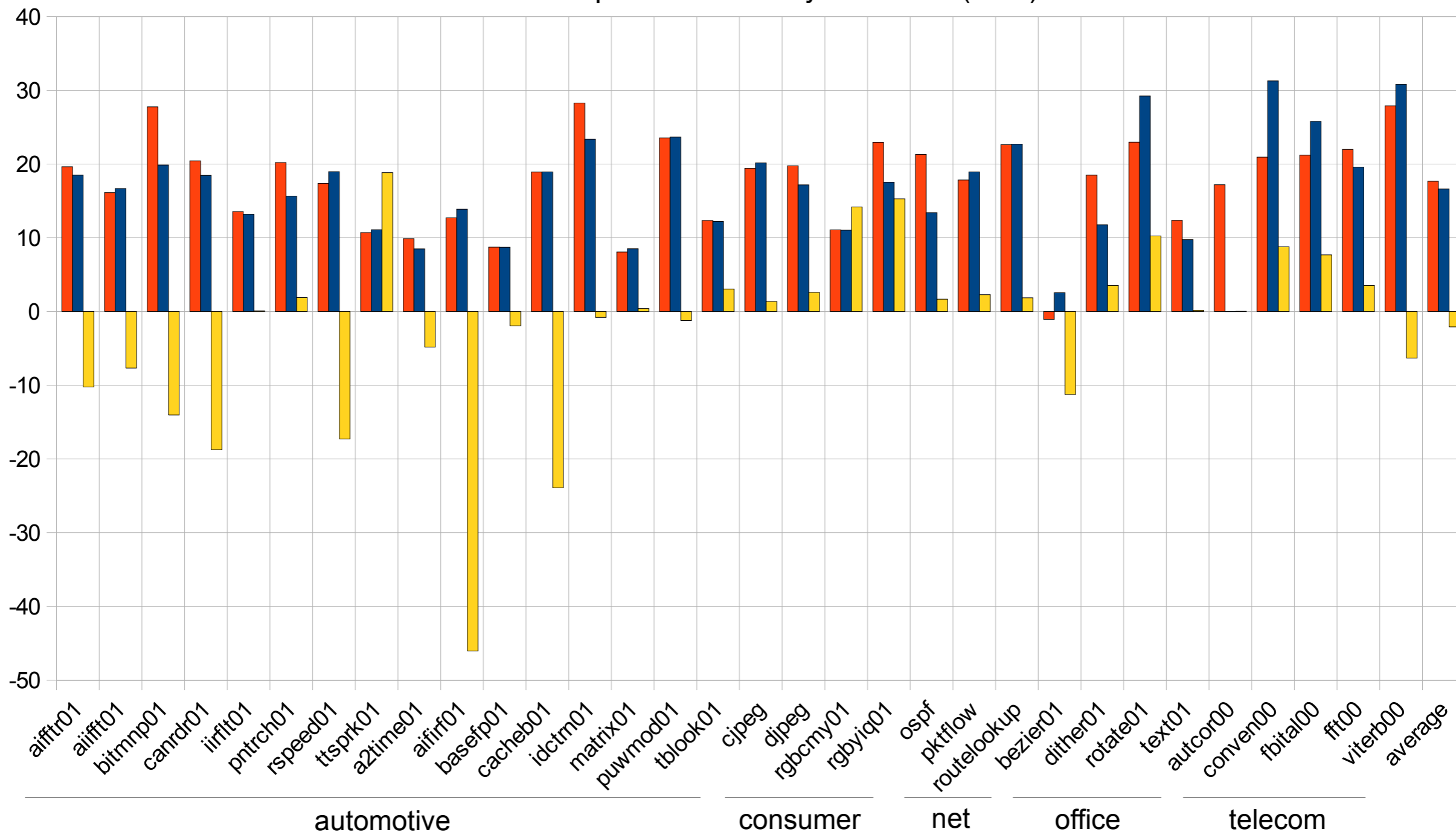
- register allocator selects registers with lower ID from set of possible registers
- calling conventions constrain register allocator





# Evaluation - Performance Improvements

Improvement in Cycle Count (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 17.7%)

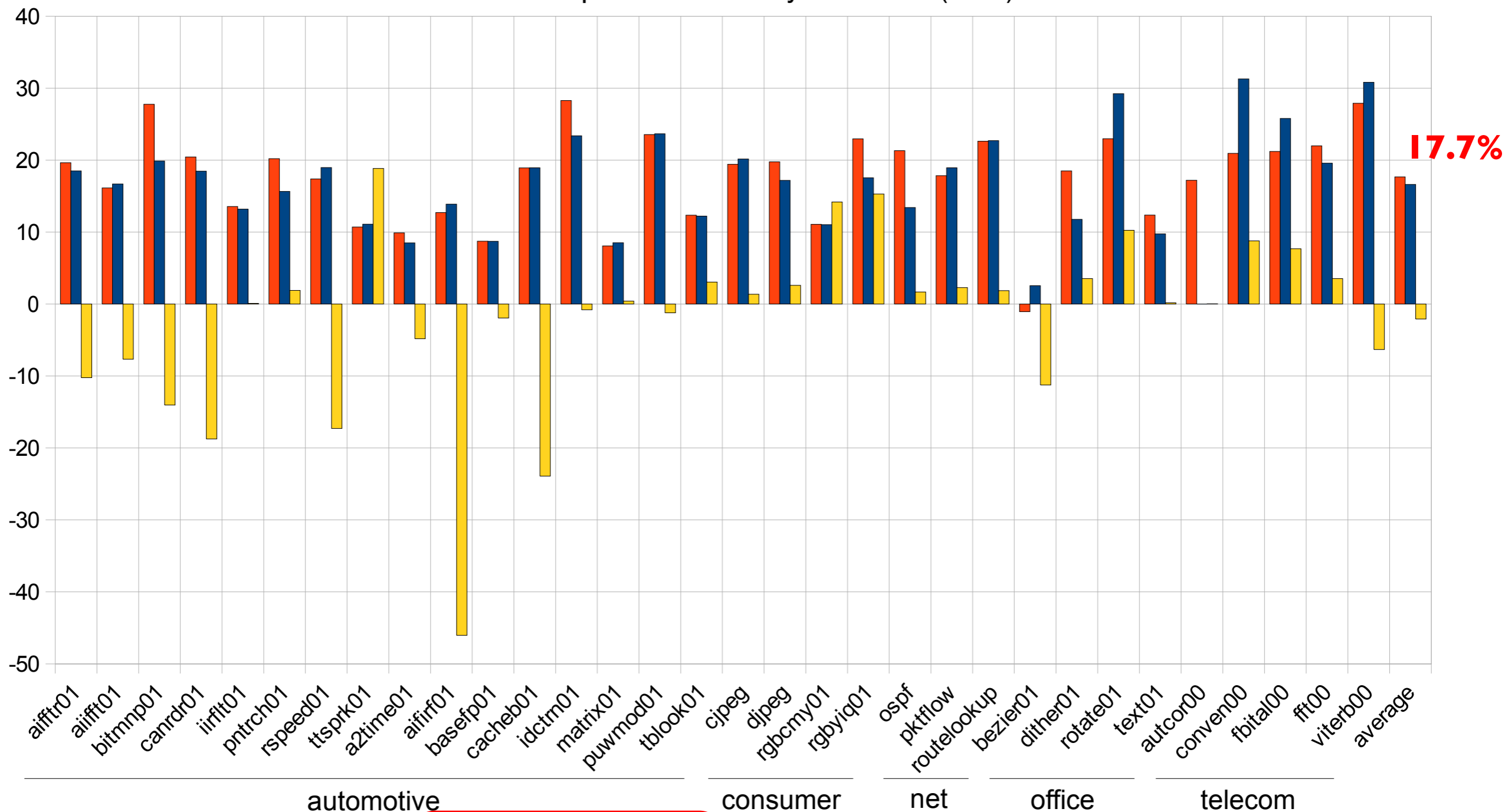
Opportunistic selection (avg: 16.6%)

GCC (avg: -2.08%)



# Evaluation - Performance Improvements

Improvement in Cycle Count (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 17.7%)

Opportunistic selection (avg: 16.6%)

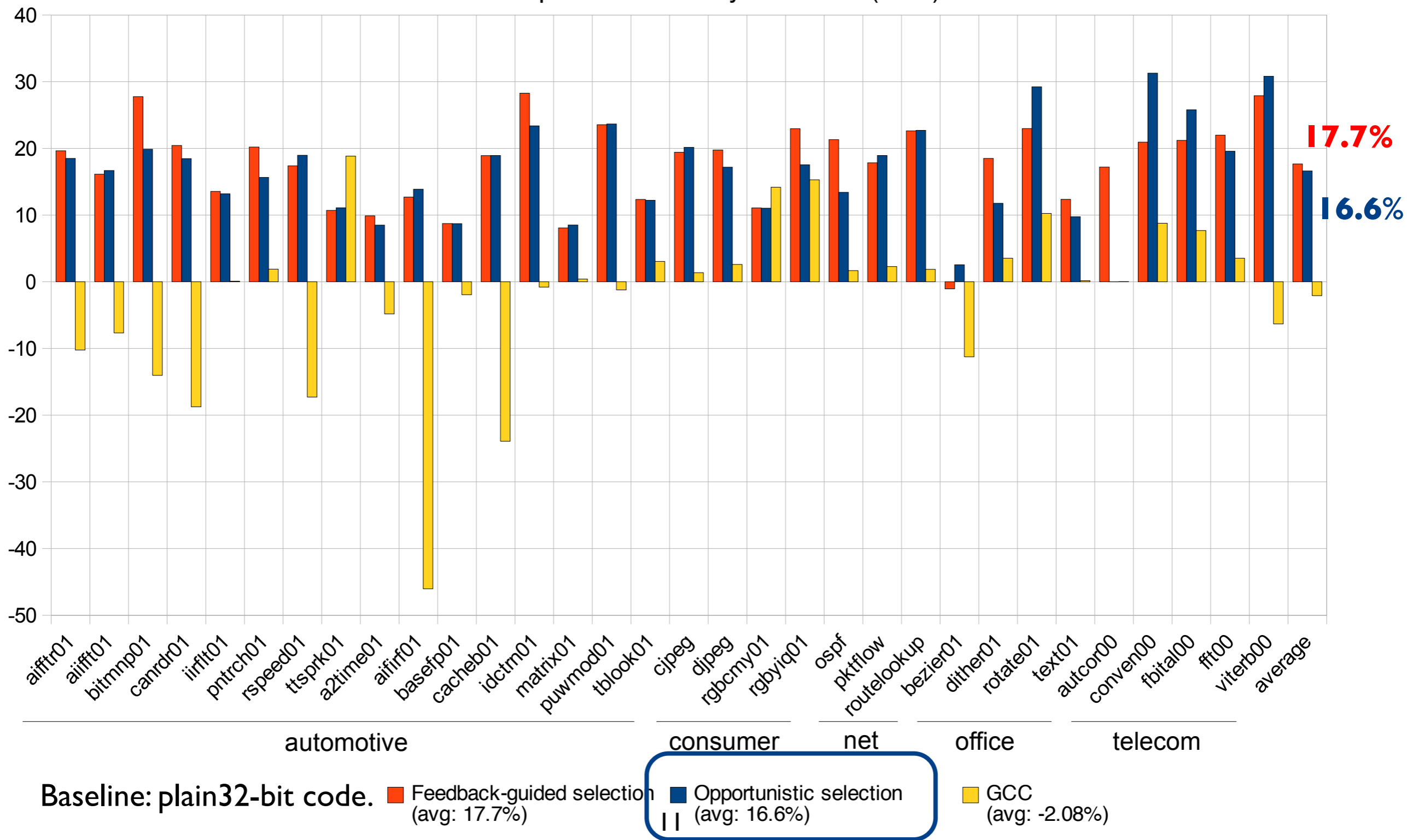
GCC (avg: -2.08%)





# Evaluation - Performance Improvements

Improvement in Cycle Count (in %)

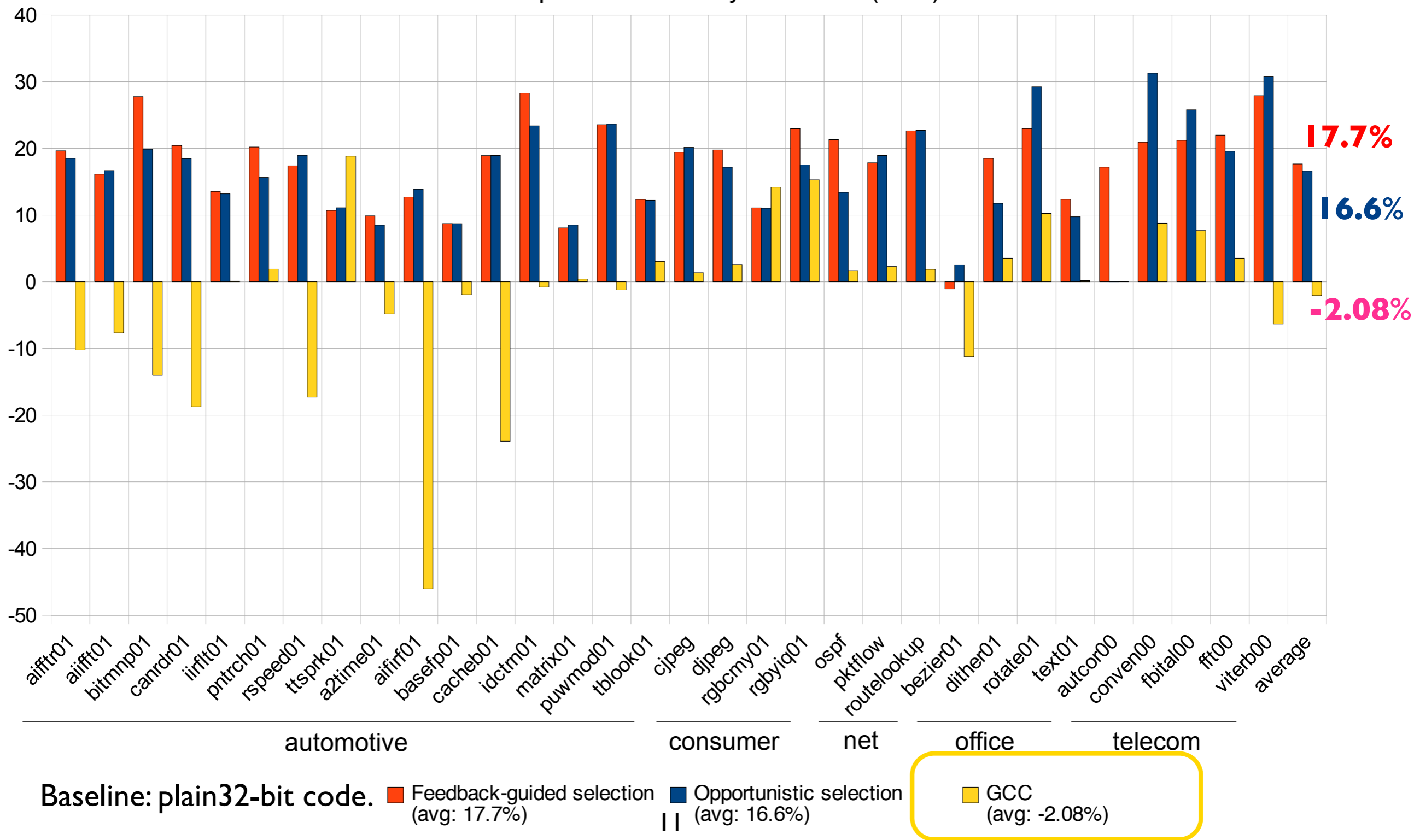


Baseline: plain32-bit code.



# Evaluation - Performance Improvements

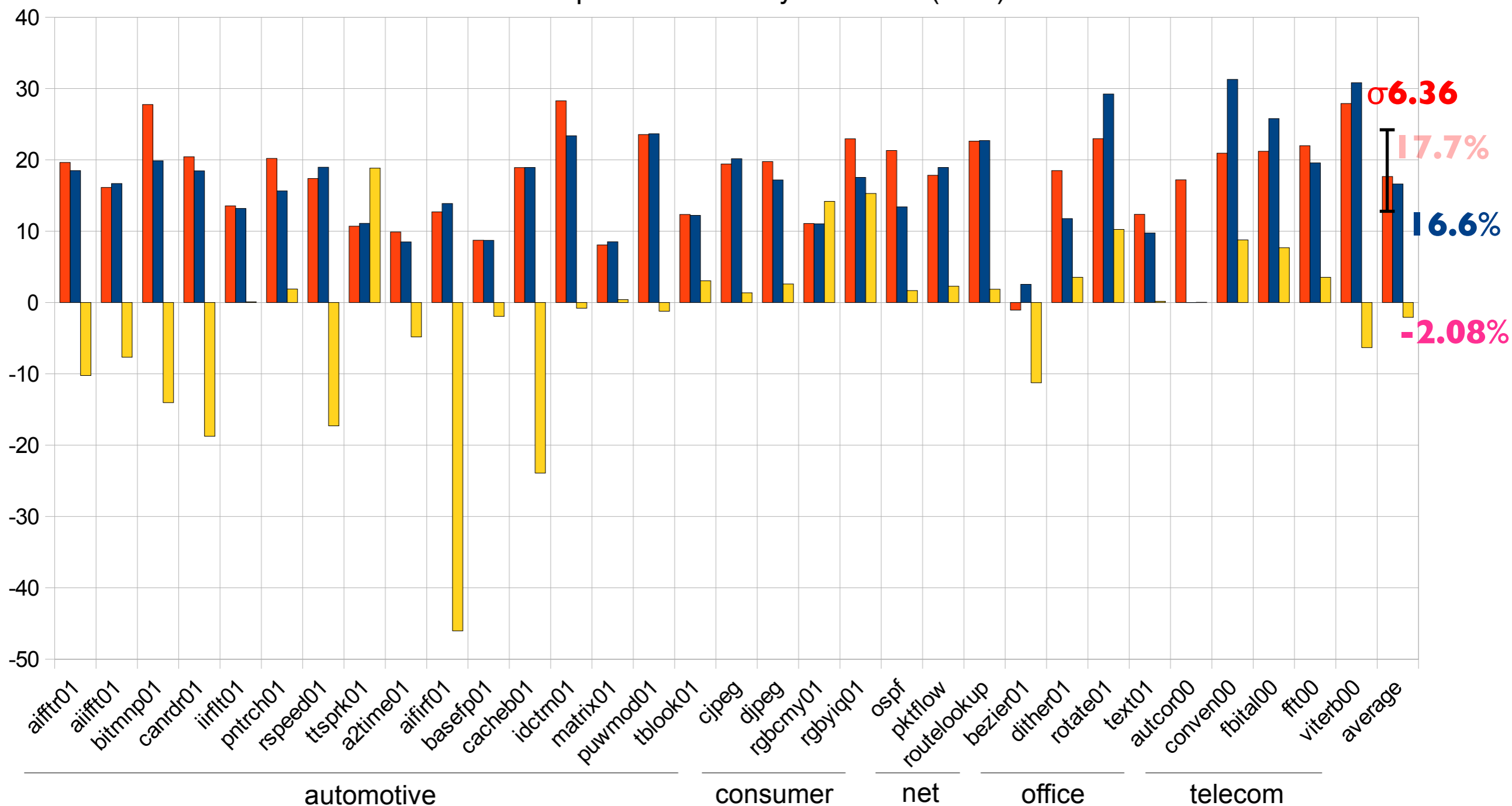
Improvement in Cycle Count (in %)





# Evaluation - Performance Improvements

Improvement in Cycle Count (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 17.7%)

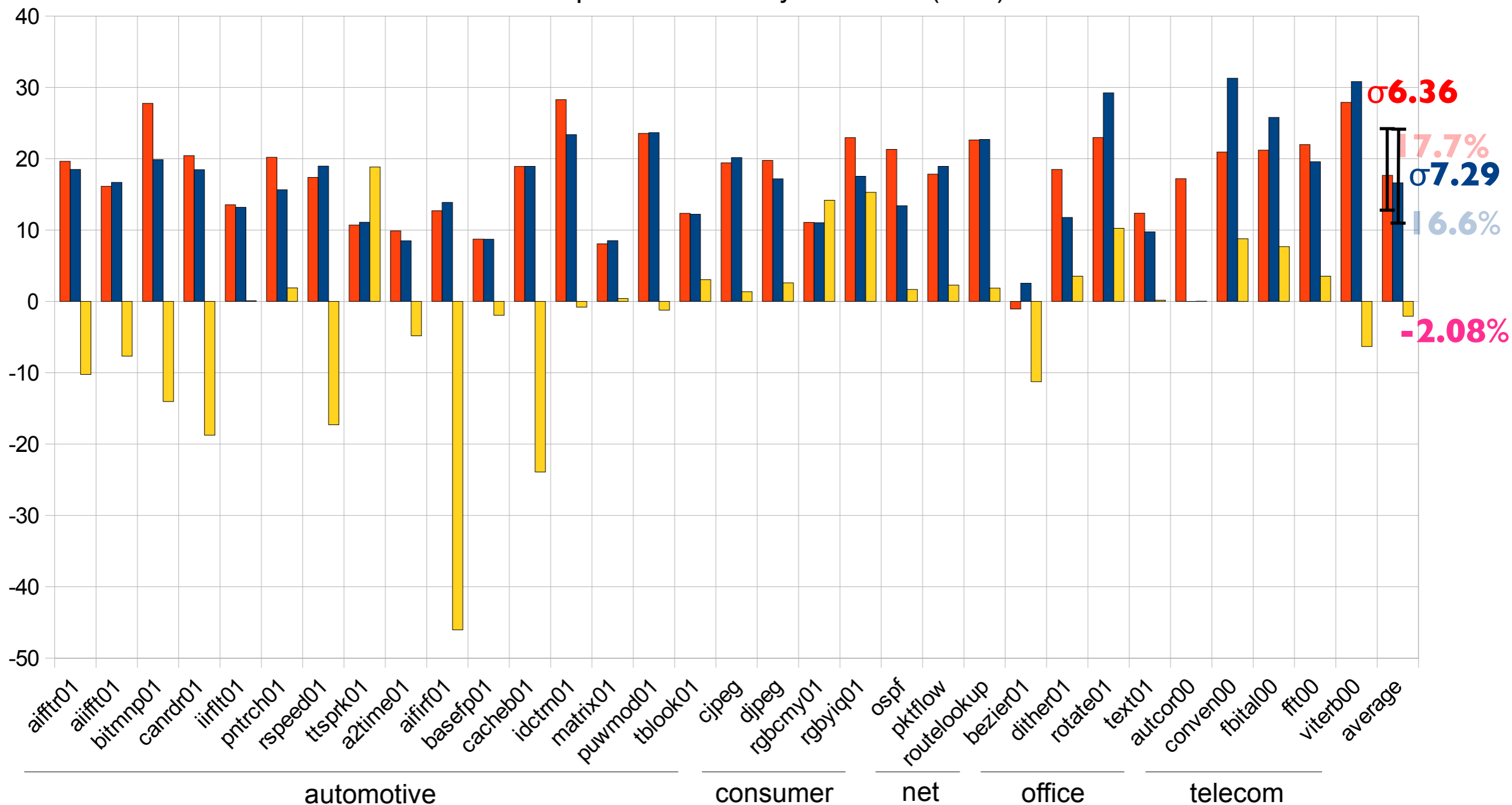
Opportunistic selection (avg: 16.6%)

GCC (avg: -2.08%)



# Evaluation - Performance Improvements

Improvement in Cycle Count (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 17.7%)

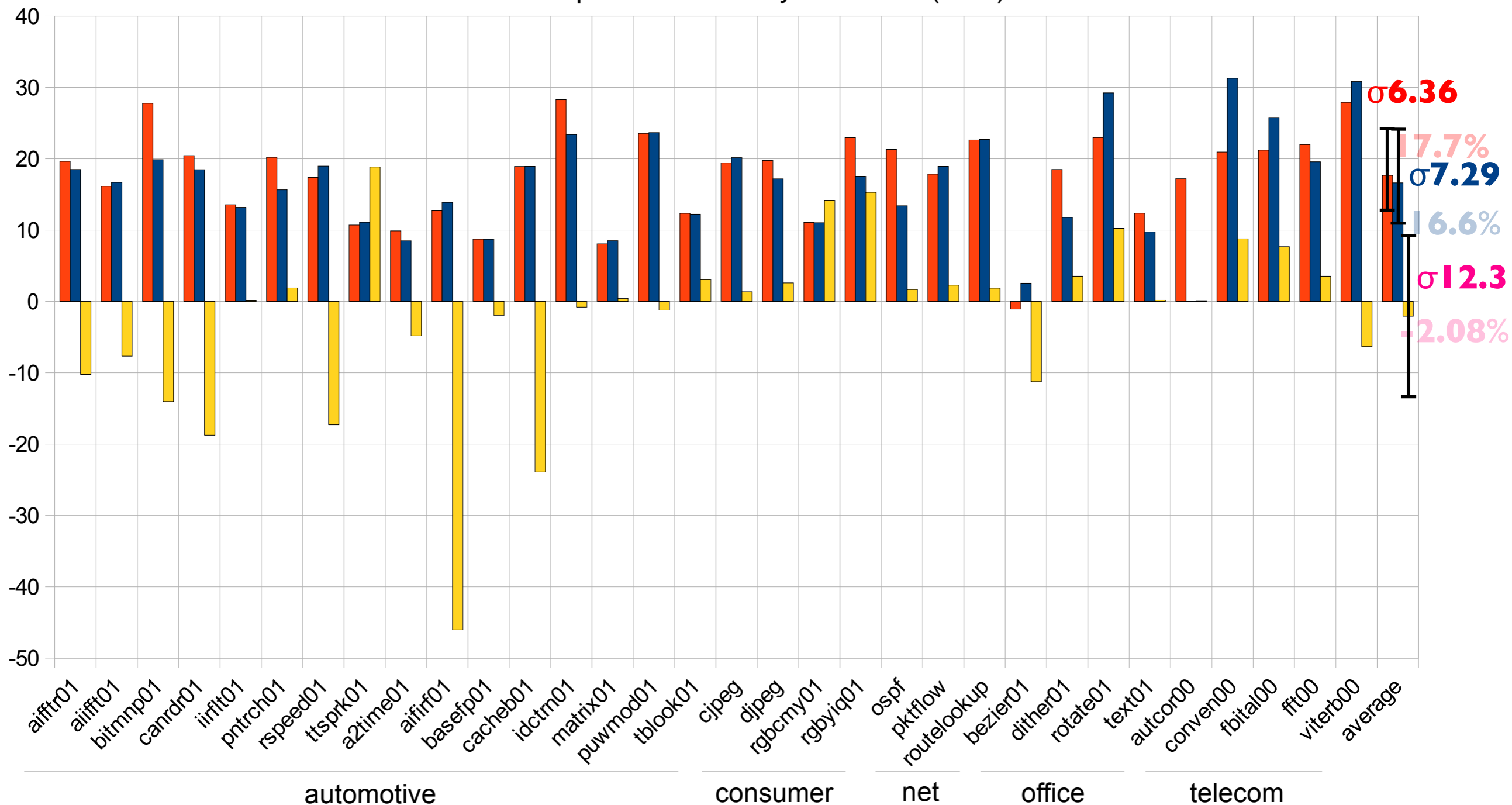
Opportunistic selection (avg: 16.6%)

GCC (avg: -2.08%)



# Evaluation - Performance Improvements

Improvement in Cycle Count (in %)



Baseline: plain32-bit code.

Feedback-guided selection (avg: 17.7%)

Opportunistic selection (avg: 16.6%)

GCC (avg: -2.08%)



# Conclusions

- Compact Code Generation is an *integrated* Instruction Selection and Register Allocation problem.



# Conclusions

- Compact Code Generation is an *integrated* Instruction Selection and Register Allocation problem.
- While our simple *opportunistic* mode works well, our *feedback-directed* mode produces more consistent results and does not rely on calling conventions or register-allocation implementations.



# Conclusions

- Compact Code Generation is an *integrated* Instruction Selection and Register Allocation problem.
- While our simple *opportunistic* mode works well, our *feedback-directed* mode produces more consistent results and does not rely on calling conventions or register-allocation implementations.
- Our scheme is the first one demonstrating that small code size can be achieved whilst improving performance.





Thanks!

For more information  
visit our website or  
search for the term  
'PASTA project'.



The screenshot shows the PASTA project website. At the top, there are navigation links for University Homepage, School Homepage, School Contacts, and School Search. The main header includes the University of Edinburgh logo and the text 'THE UNIVERSITY of EDINBURGH informatics'. Below this is the PASTA logo and the full name 'Processor Automated Synthesis by iTerative Analysis Project'. The main content area features a paragraph about the project's goal to automate the design and optimisation of customisable embedded processors. To the right of this text is a small image of a processor chip. Below the paragraph is a list of three main areas for automated synthesis: processor architecture, micro-architecture, and compiler. Further down is another paragraph discussing the challenge of inter-dependent design decisions and the need for automated trade-off optimization. To the left of this paragraph is a photo of a group of people standing in front of the Informatics Forum building. On the right side of the page, there is a search bar and a navigation menu with links to Home, News, Publications, People, Seminars, Contact, and Internal Area. Below the menu are sections for 'PASTA Activities', 'EnCore Tools' (listing ECC Compiler, GCC Compiler, EnCore Simulator, and Verification/Co-Simulation), 'HW Systems' (listing EnCore Calton, FPGA Platform, ASIC Flow, and Chips & Boards), 'M.Sc. and UG Projects', and 'Research Areas' (listing Resource Sharing, Configurable Flow Accelerators, Automated ISE, Mapping ISE, Power Prediction & Optimisation, Interconnect Synthesis, Cache Optimisation for Embedded Systems, Compilation for Dual Memory Banks, and Fast Cycle-Approximate Simulation).