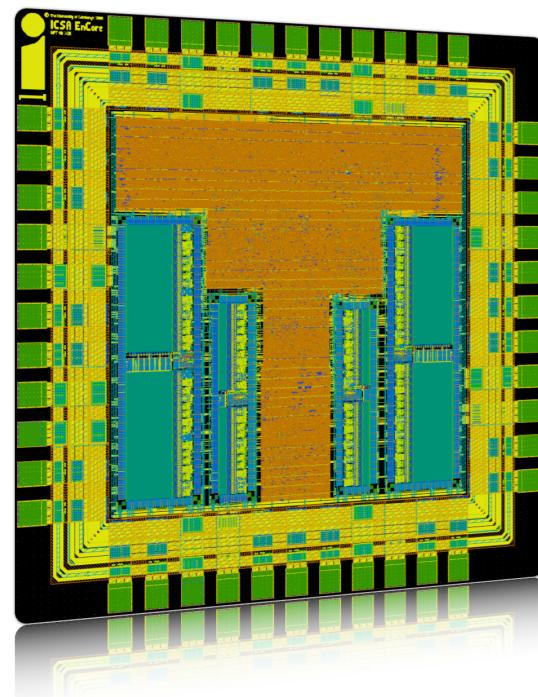


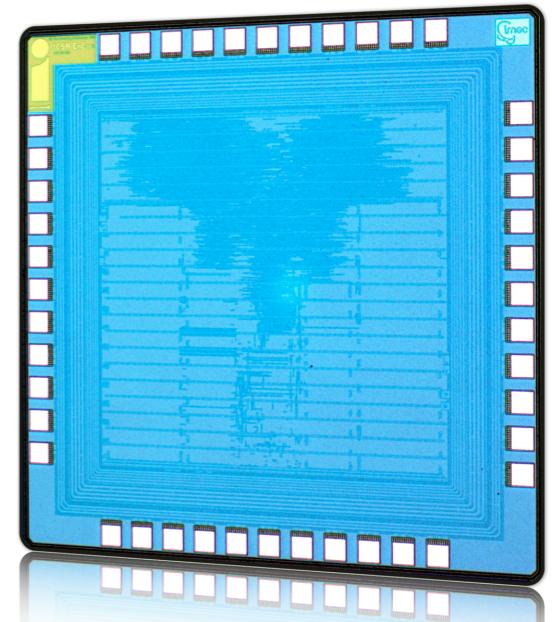


# Compact Code Generation

Tobias Edler von Koch  
Igor Böhm and Björn Franke



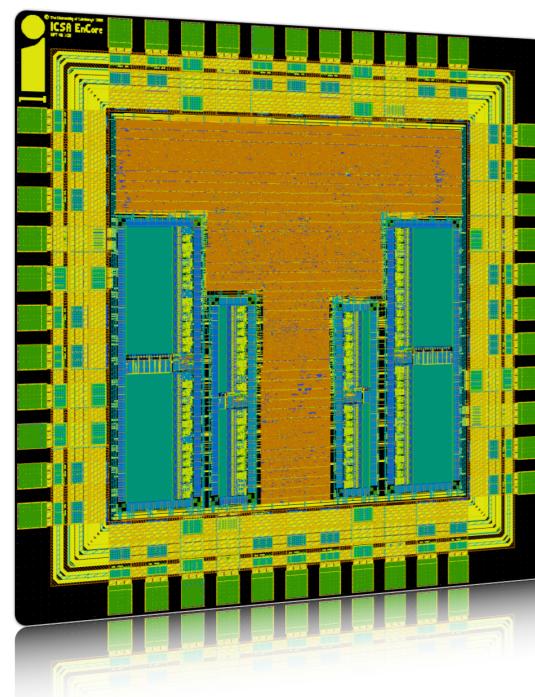
**PAsT**  
Processor **A**utomated **S**ynthesis  
by **i****T**erative **A**nalysis





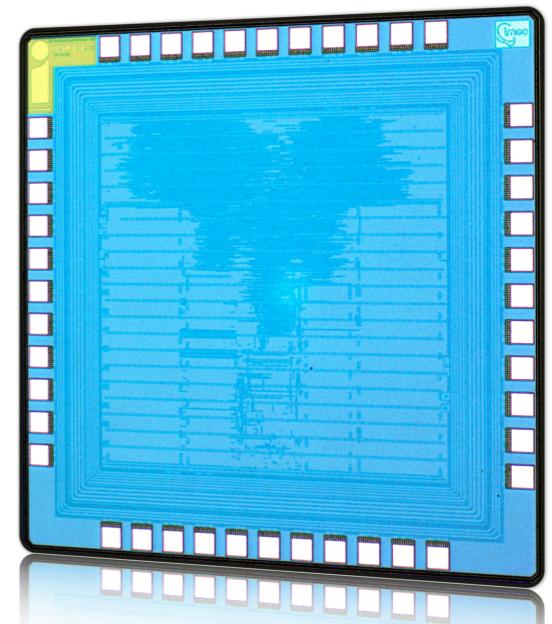
# Integrated Instruction Selection and Register Allocation for Compact Code Generation Exploiting Freeform Mixing of 16- and 32-bit Instructions

Tobias Edler von Koch  
Igor Böhm and Björn Franke



PAsT  
A  
S  
A

Processor **A**utomated **S**ynthesis  
by **i****T**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



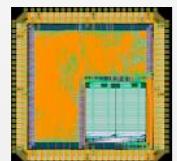
**Thumb-2**  
**ISA**

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



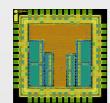
# Does Code Size Matter?

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?

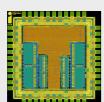
128-Mbit Flash  
27.3mm<sup>2</sup> at 0.13μm



**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
```

Basic Block

24 bytes



# 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
```

Basic Block

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
```

24 bytes

12 bytes



# 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
```

Basic Block

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

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
```

Basic Block

## 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
```

## Mixed Mode Integrated

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

24 bytes

20 bytes

16 bytes

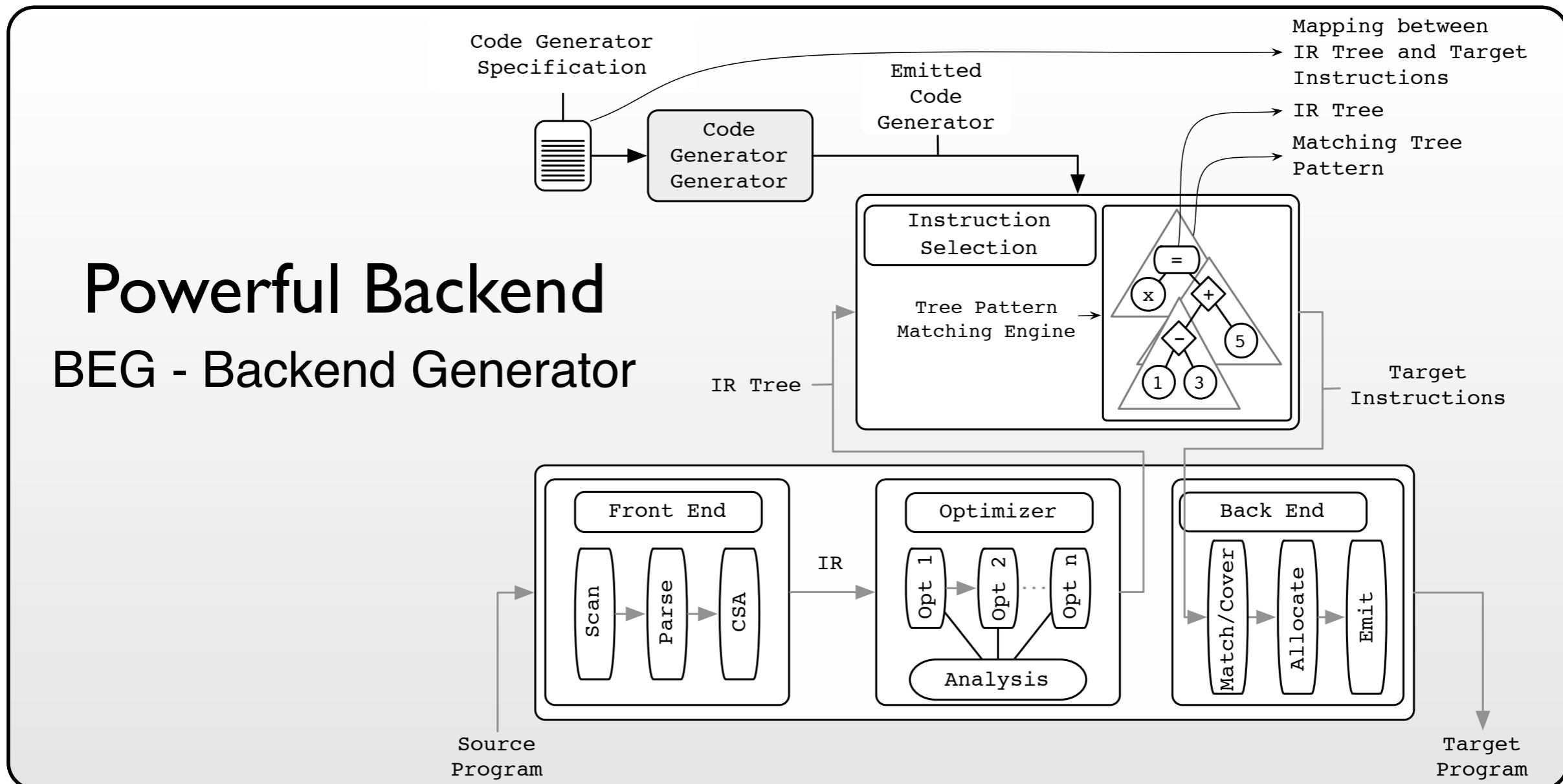


# Efficient Compact Code Generation is an **Integrated** Instruction Selection and Register Allocation Problem!



# 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<sup>©</sup>**.

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

Opportunistic

Feedback-Guided



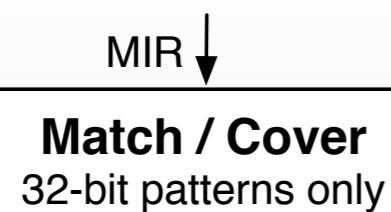
# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

Opportunistic

Feedback-Guided



Register Allocation

Code Emission

32-bit/16-bit instructions

ASM



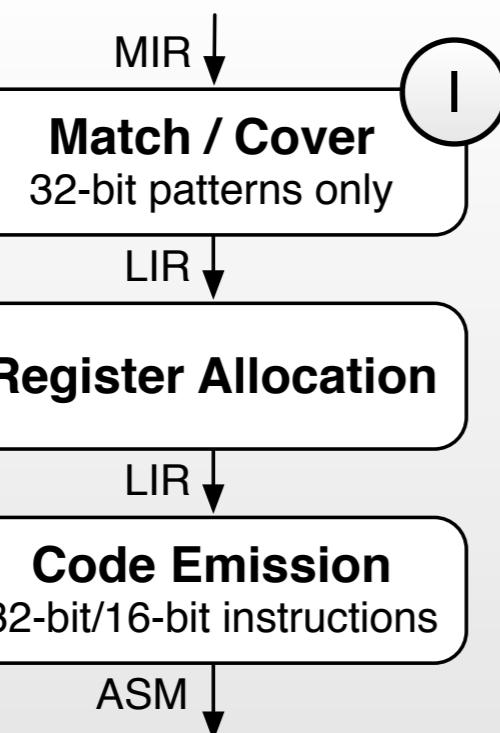
# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

Opportunistic

Feedback-Guided



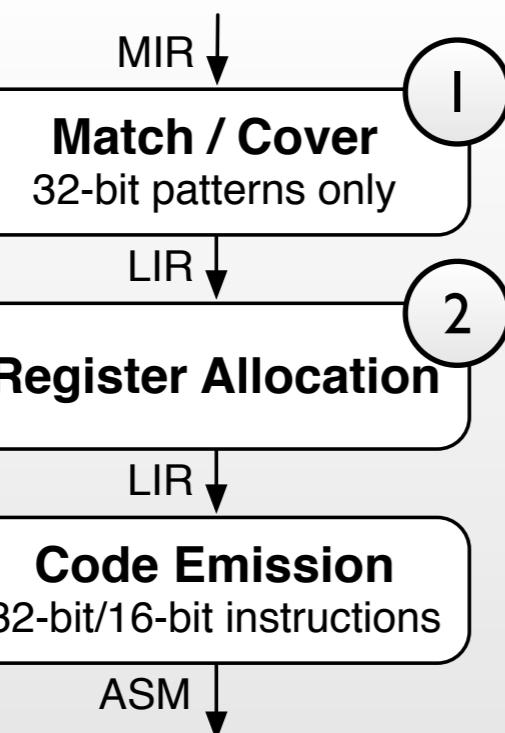


# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

## Opportunistic



## Feedback-Guided

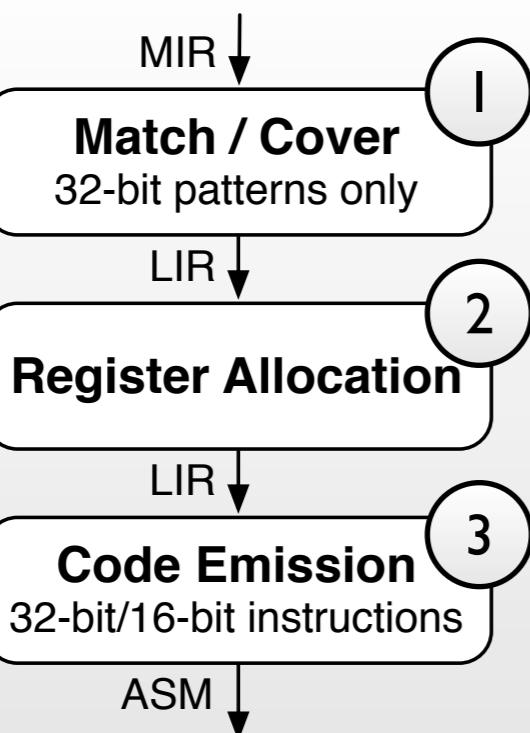


# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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<sup>©</sup>**.

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

MIR  
↓  
**Match / Cover**  
32-bit & 16-bit patterns  
preferred

MIR & LIR  
↓  
**Register Allocation**  
with constrained register  
sets for 16-bit instructions

MIR & LIR  
↓  
**Code Emission**  
32-bit/16-bit instructions

Annotated MIR  
↓

**MIR Annotation**  
selectively deactivate 16-bit

ASM  
↓



# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

MIR  
↓  
**I**  
**Match / Cover**  
32-bit & 16-bit patterns preferred

MIR & LIR  
↓  
**Register Allocation**  
with constrained register sets for 16-bit instructions

MIR & LIR  
↓  
**Code Emission**  
32-bit/16-bit instructions

Annotated MIR

**MIR Annotation**  
selectively deactivate 16-bit

ASM



# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

MIR  
↓  
**Match / Cover**  
32-bit & 16-bit patterns preferred

MIR & LIR  
↓  
**Register Allocation**  
with constrained register sets for 16-bit instructions

MIR & LIR  
↓  
**Code Emission**  
32-bit/16-bit instructions

Annotated MIR

MIR Annotation  
selectively deactivate 16-bit

ASM



# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

MIR  
Match / Cover  
32-bit & 16-bit patterns preferred

MIR & LIR  
Register Allocation  
with constrained register sets for 16-bit instructions

MIR & LIR  
Code Emission  
32-bit/16-bit instructions

Annotated MIR

MIR Annotation  
selectively deactivate 16-bit

ASM



# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

MIR  
↓  
**Match / Cover**  
32-bit & 16-bit patterns  
preferred

MIR & LIR  
↓  
**Register Allocation**  
with constrained register  
sets for 16-bit instructions

MIR & LIR  
↓  
**Code Emission**  
32-bit/16-bit instructions

Annotated MIR  
↓  
3

**MIR Annotation**  
selectively deactivate 16-bit

1

4

2

3

6



# Compact Code Generation Approach

**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

MIR  
↓  
**Match / Cover**  
32-bit & 16-bit patterns preferred

MIR & LIR  
↓  
**Register Allocation**  
with constrained register sets for 16-bit instructions

MIR & LIR  
↓  
**Code Emission**  
32-bit/16-bit instructions

Annotated MIR

4

5

3

**MIR Annotation**  
selectively deactivate 16-bit

2

6

ASM

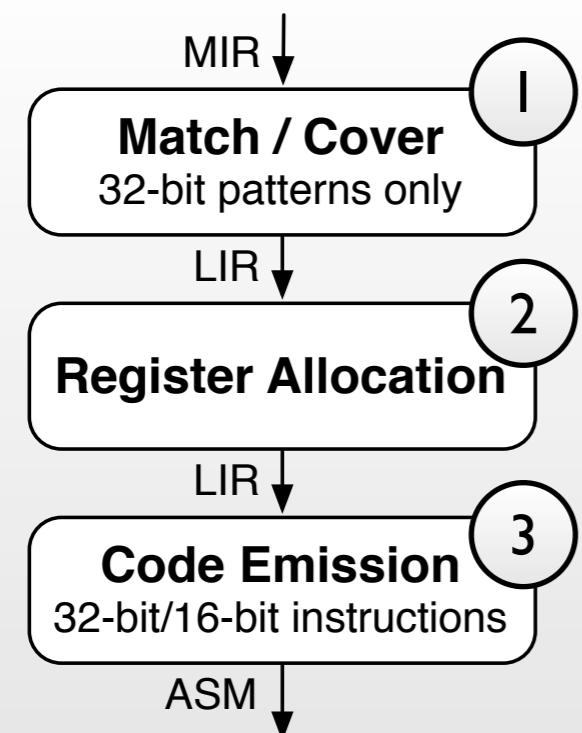


# Compact Code Generation Approach

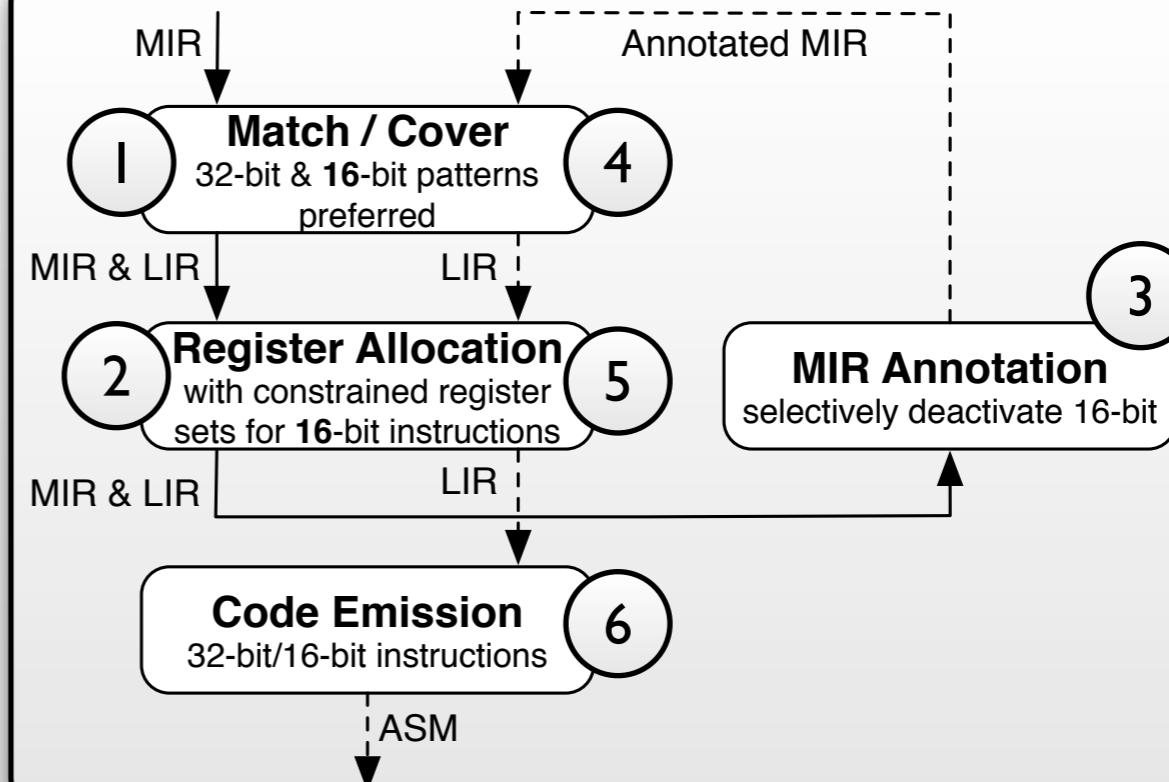
**ECC** - EnCore C Compiler based on commercial **CoSy** compiler by **ACE<sup>©</sup>**.

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

## Opportunistic



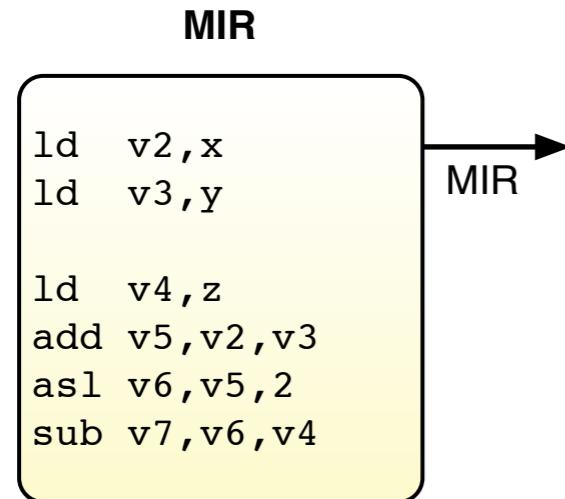
## Feedback-Guided





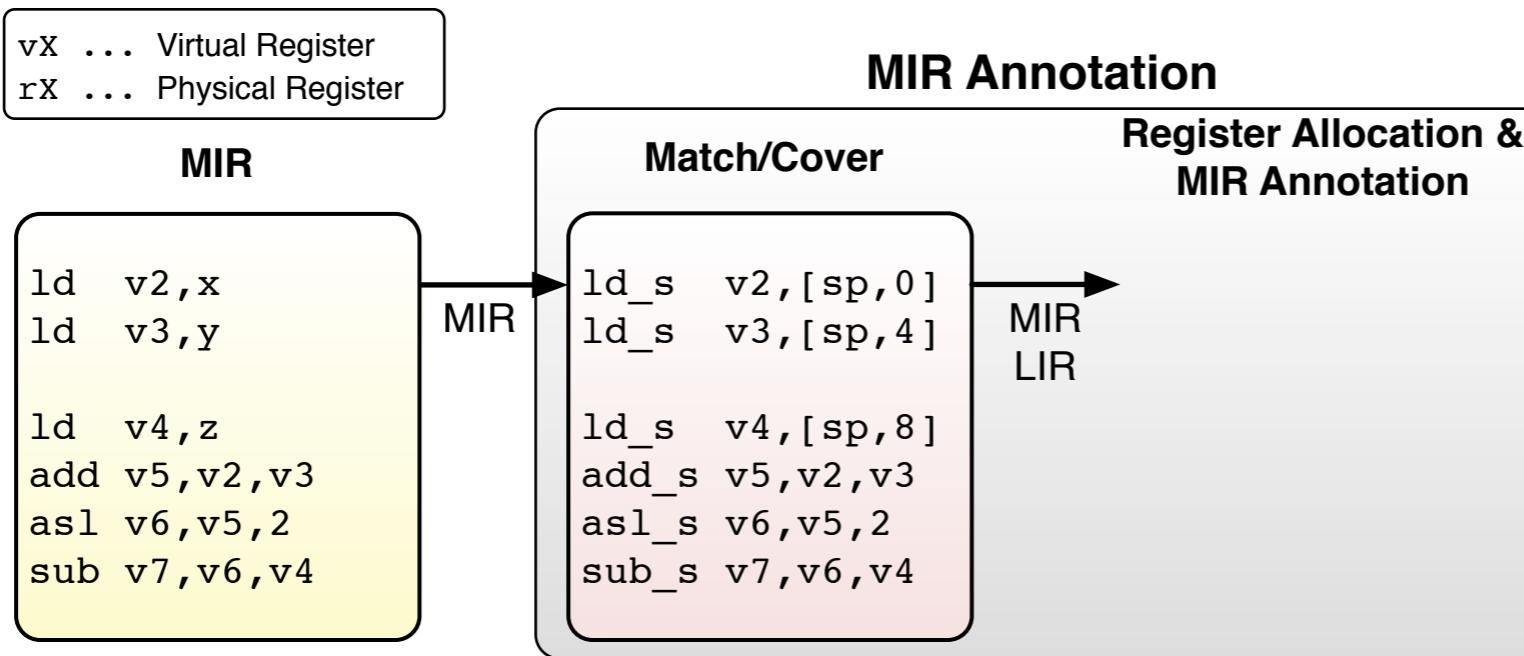
# Feedback-Guided Instruction Selection

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





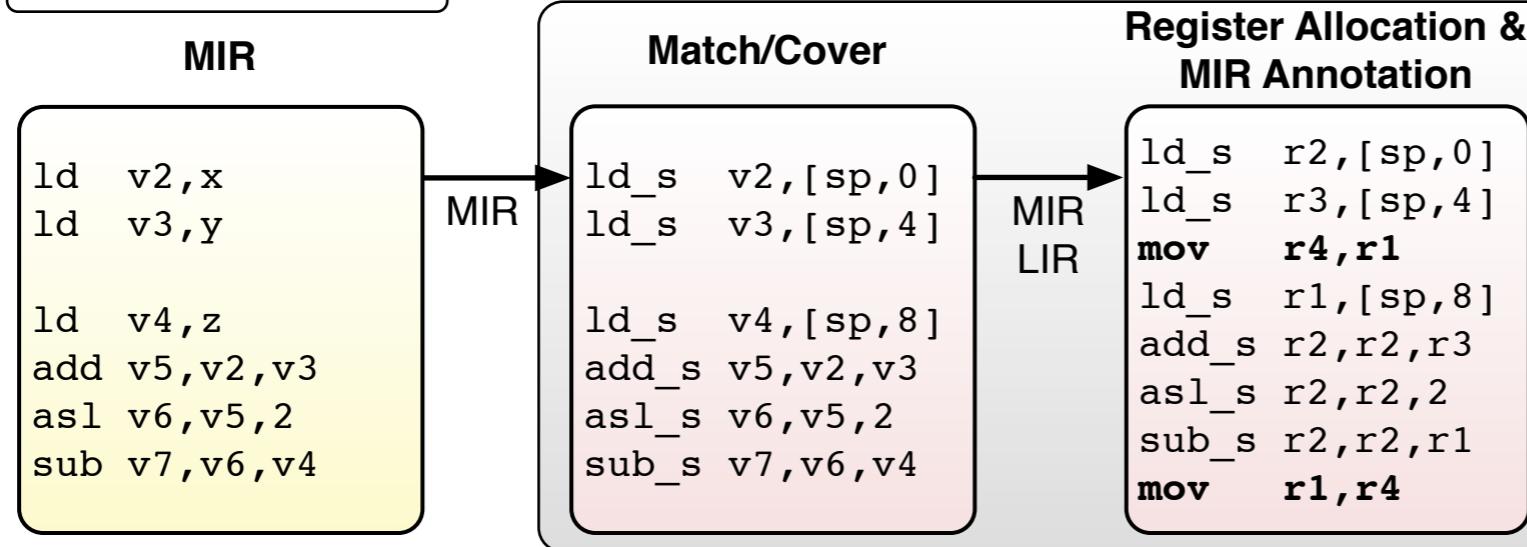
# Feedback-Guided Instruction Selection





# Feedback-Guided Instruction Selection

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

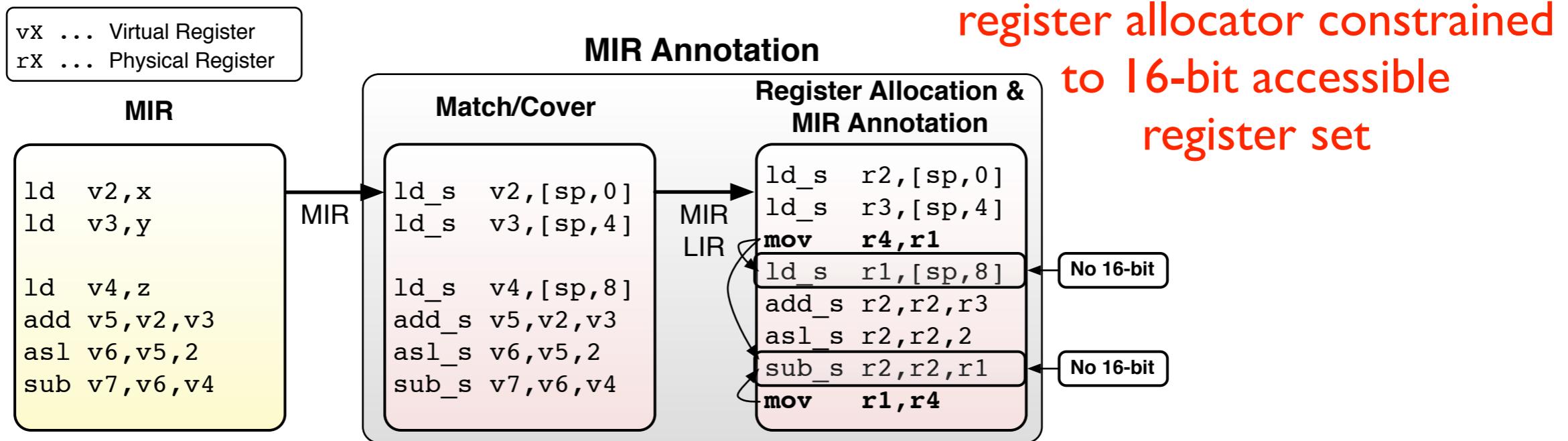


register allocator constrained  
to 16-bit accessible  
register set

aggressively select 16-bit  
instructions

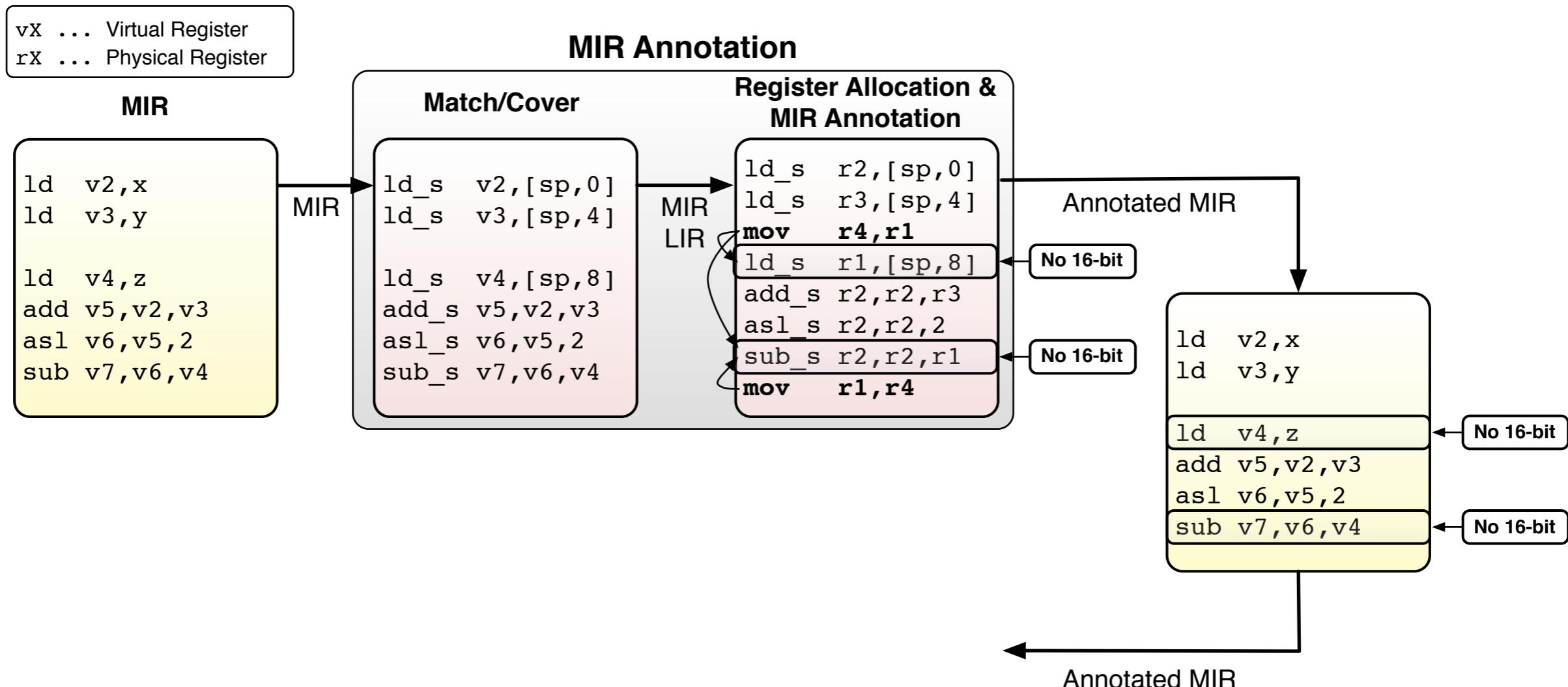


# Feedback-Guided Instruction Selection



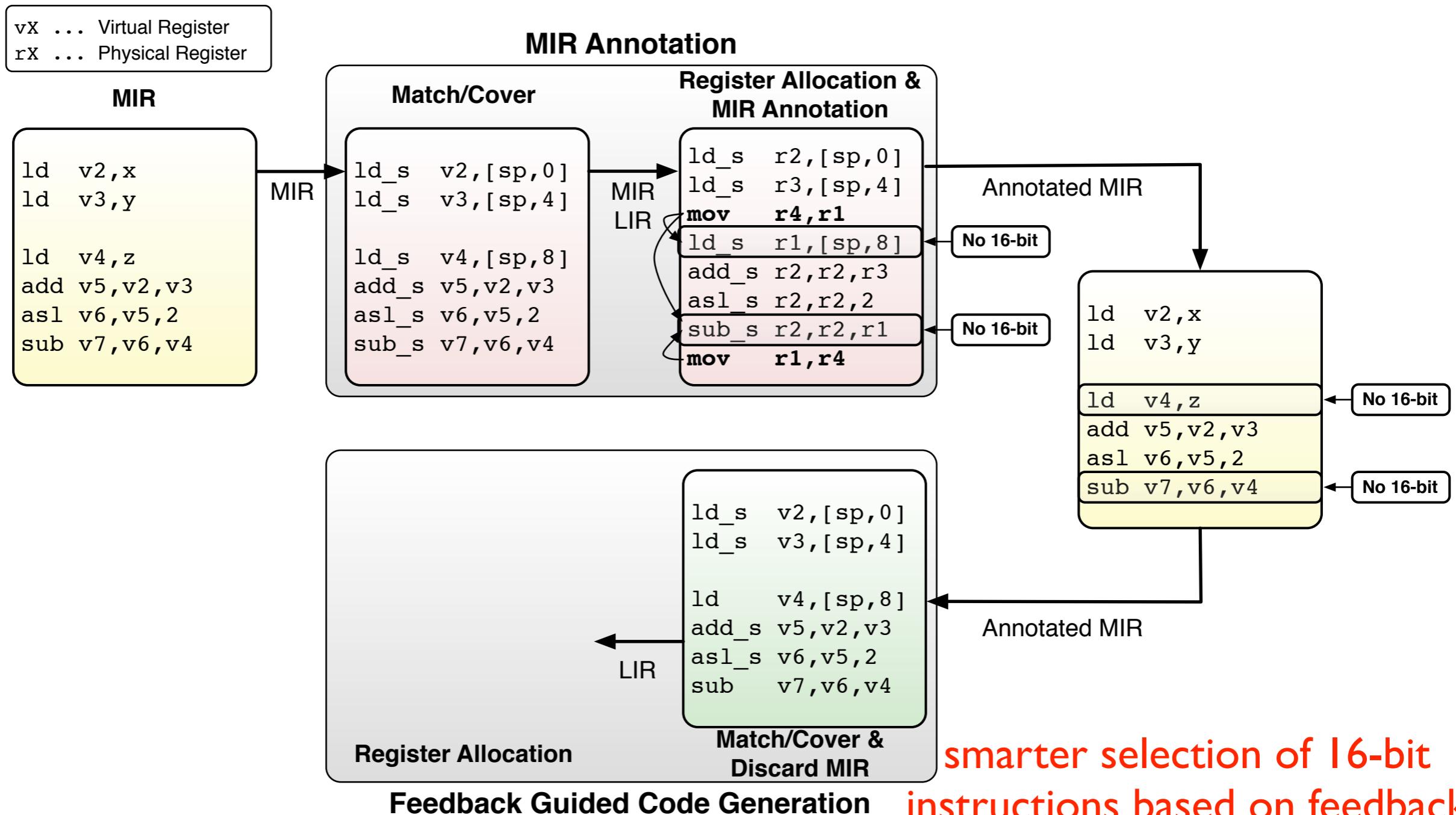


# Feedback-Guided Instruction Selection



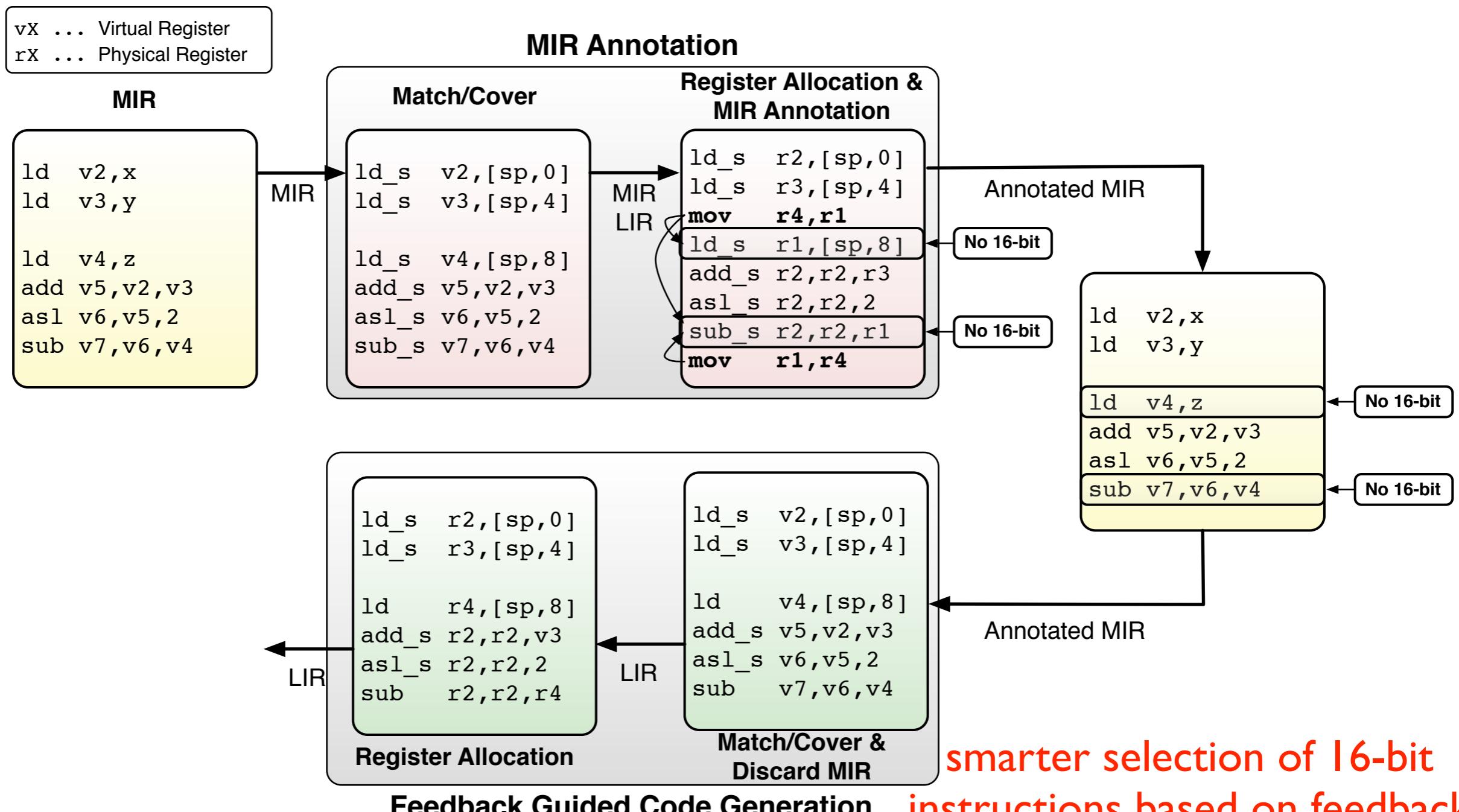


# Feedback-Guided Instruction Selection



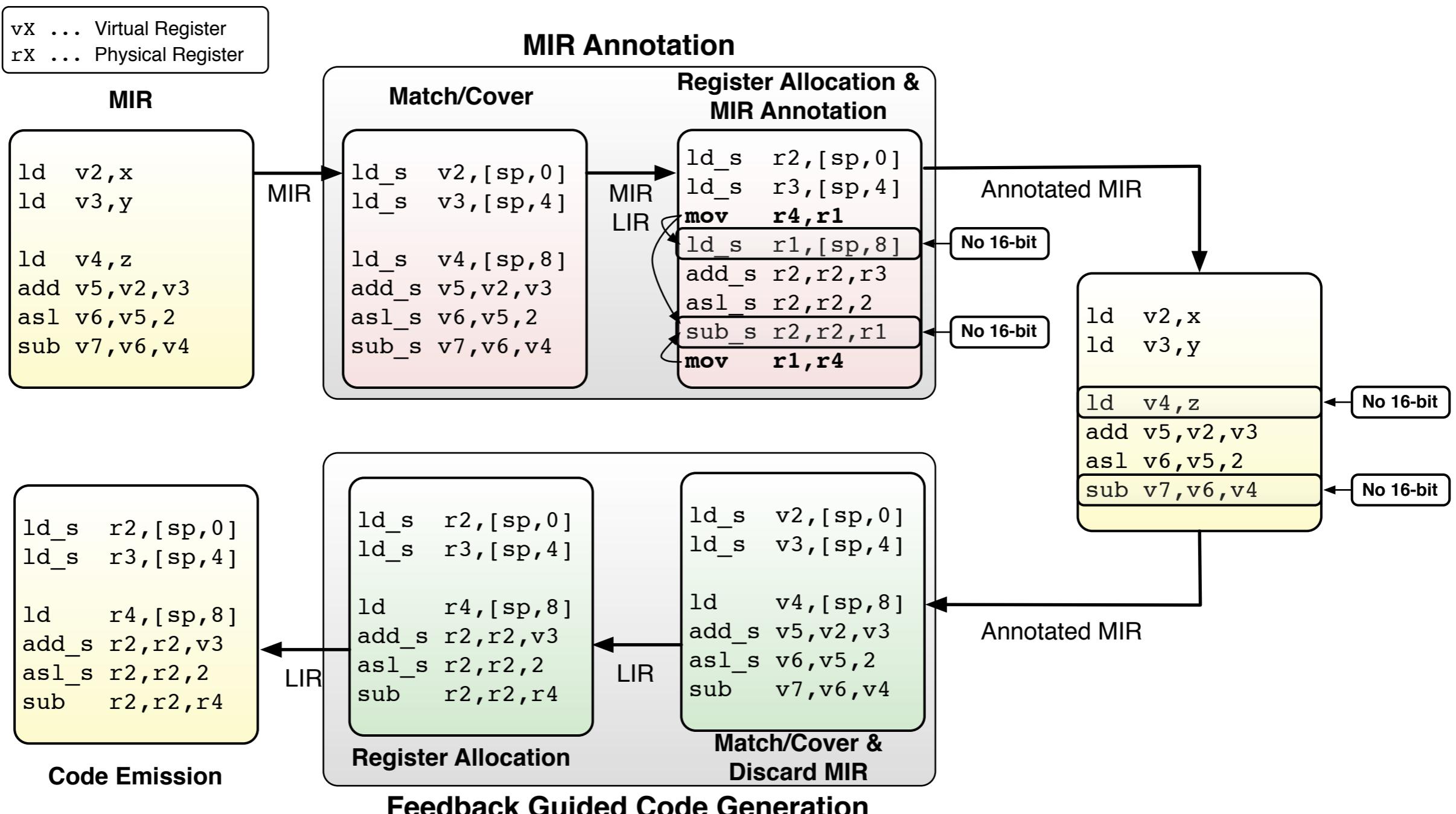


# Feedback-Guided Instruction Selection





# Feedback-Guided Instruction Selection





# Evaluation - Experimental Setup

BenchMarks	EEMBC I.I
------------	-----------

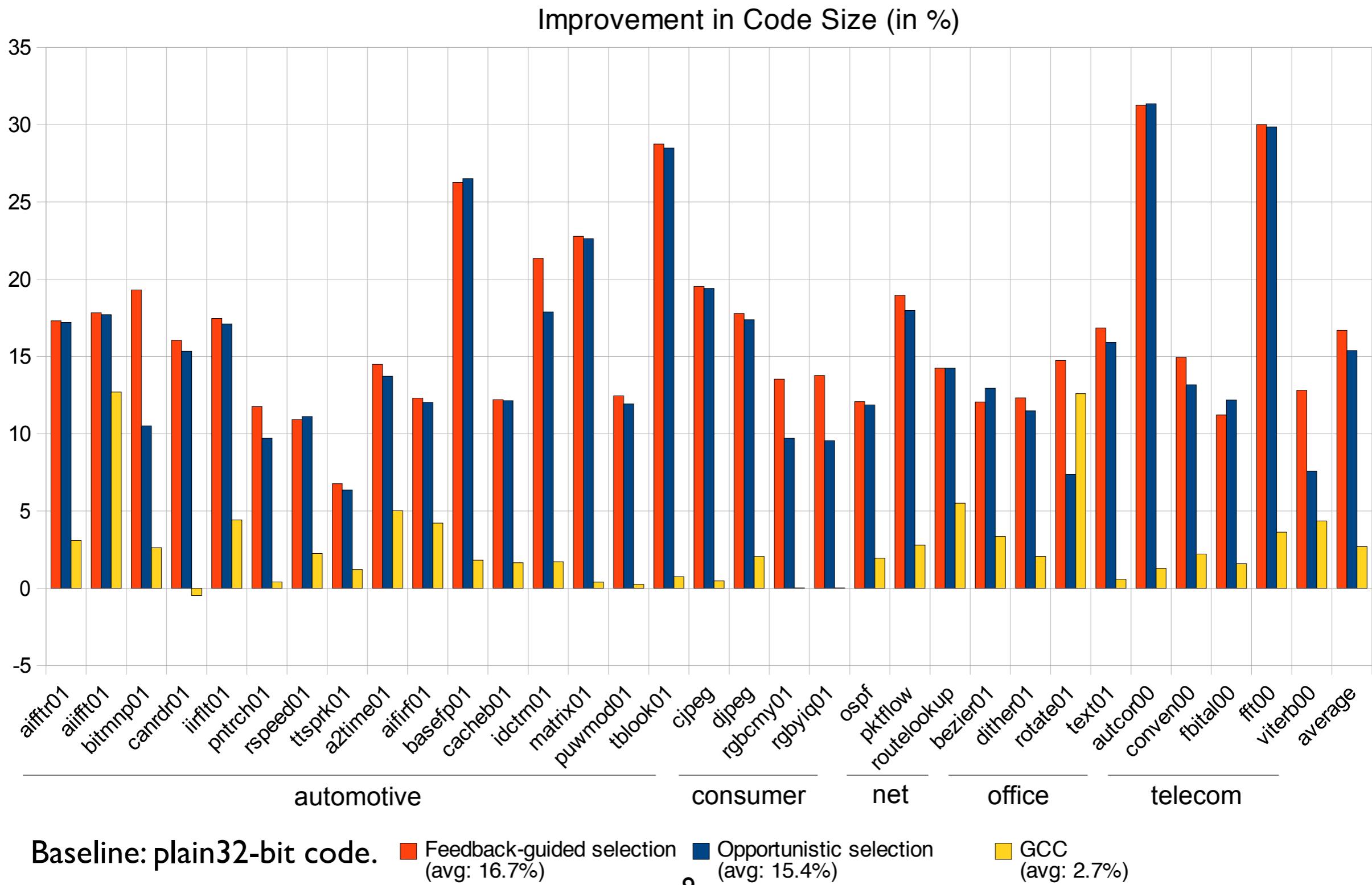
Core	ARC750D
Pipeline	7-stage interlocked
Execution Order	In-Order
Branch Prediction	Yes
ISA	ARCompact
Floating Point	Hardware
Memory Subsystem	
L1 Cache	Yes
Instruction	8k/2-way associative
Data	8k/2-way associative
L2 Cache	No



Simulator	ArcSim
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



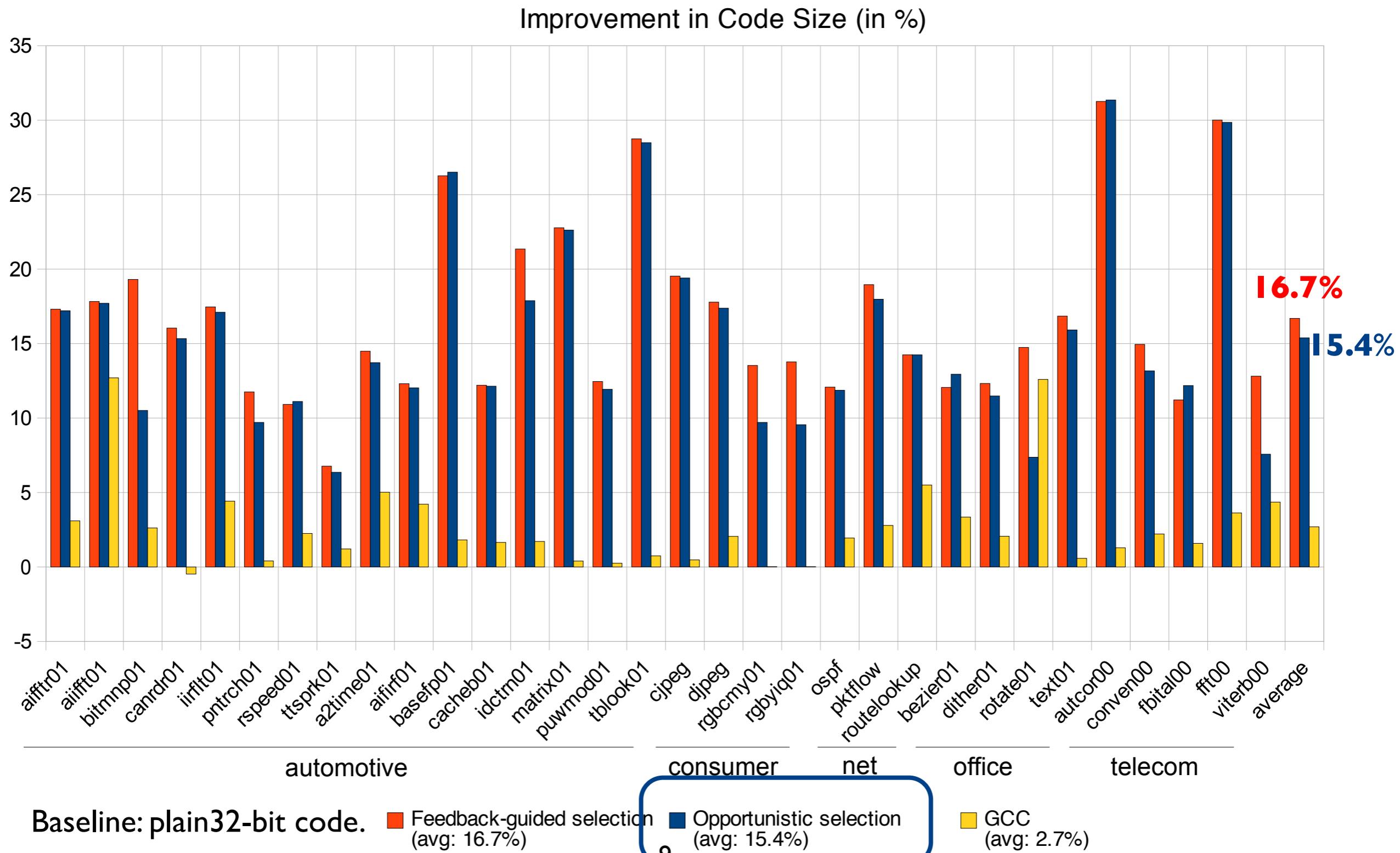


# Evaluation - Code Size Reduction



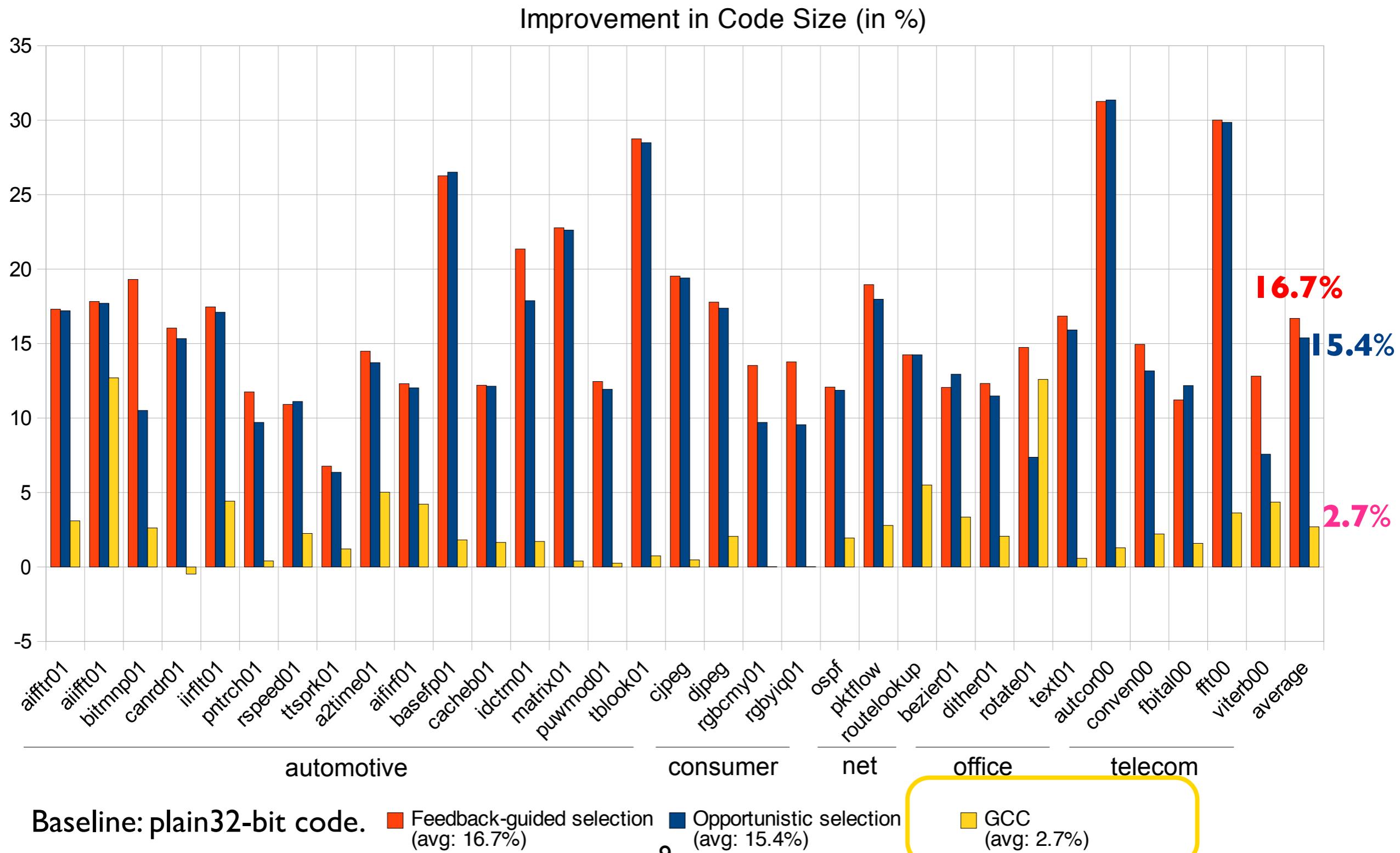


# Evaluation - Code Size Reduction



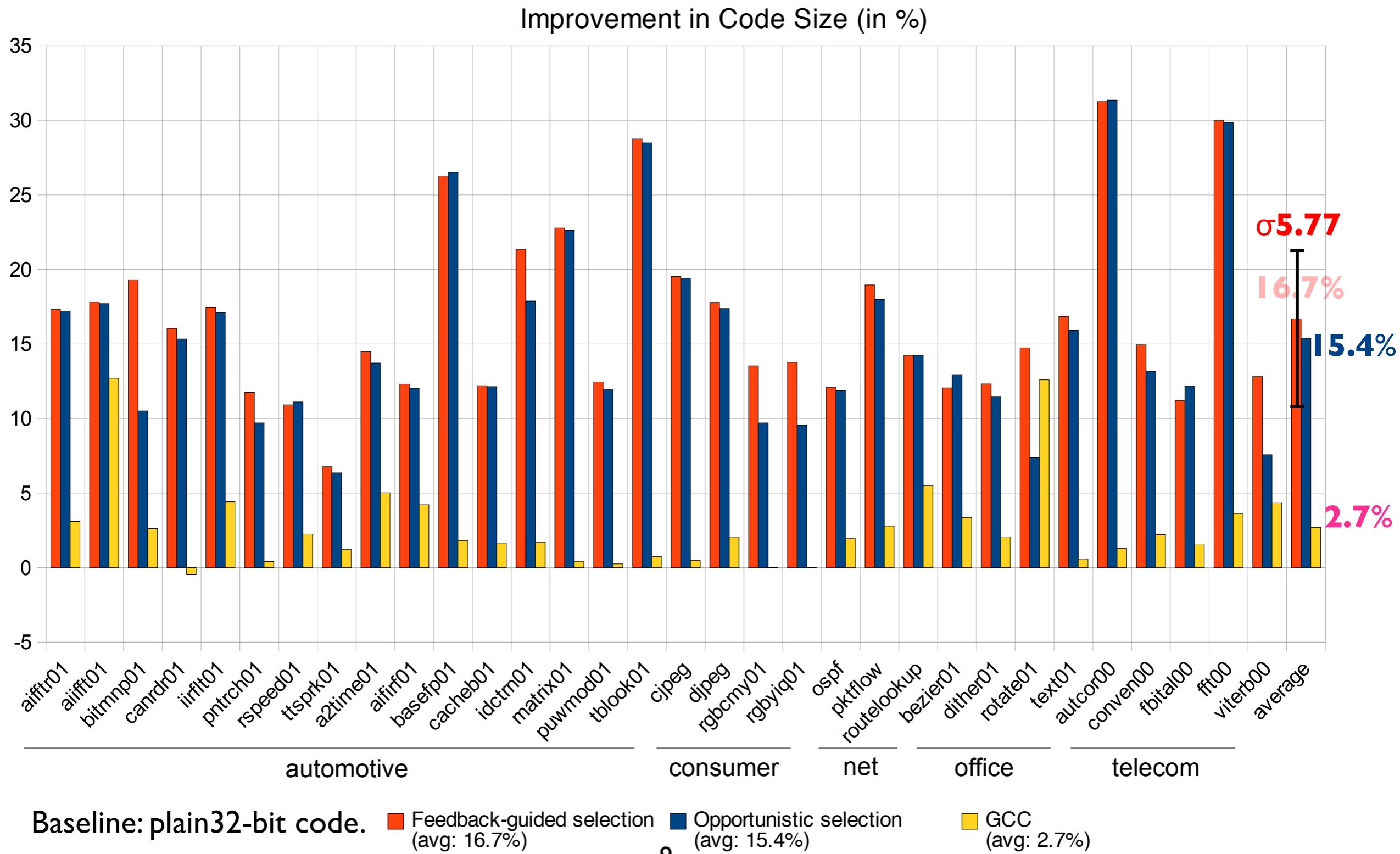


# Evaluation - Code Size Reduction



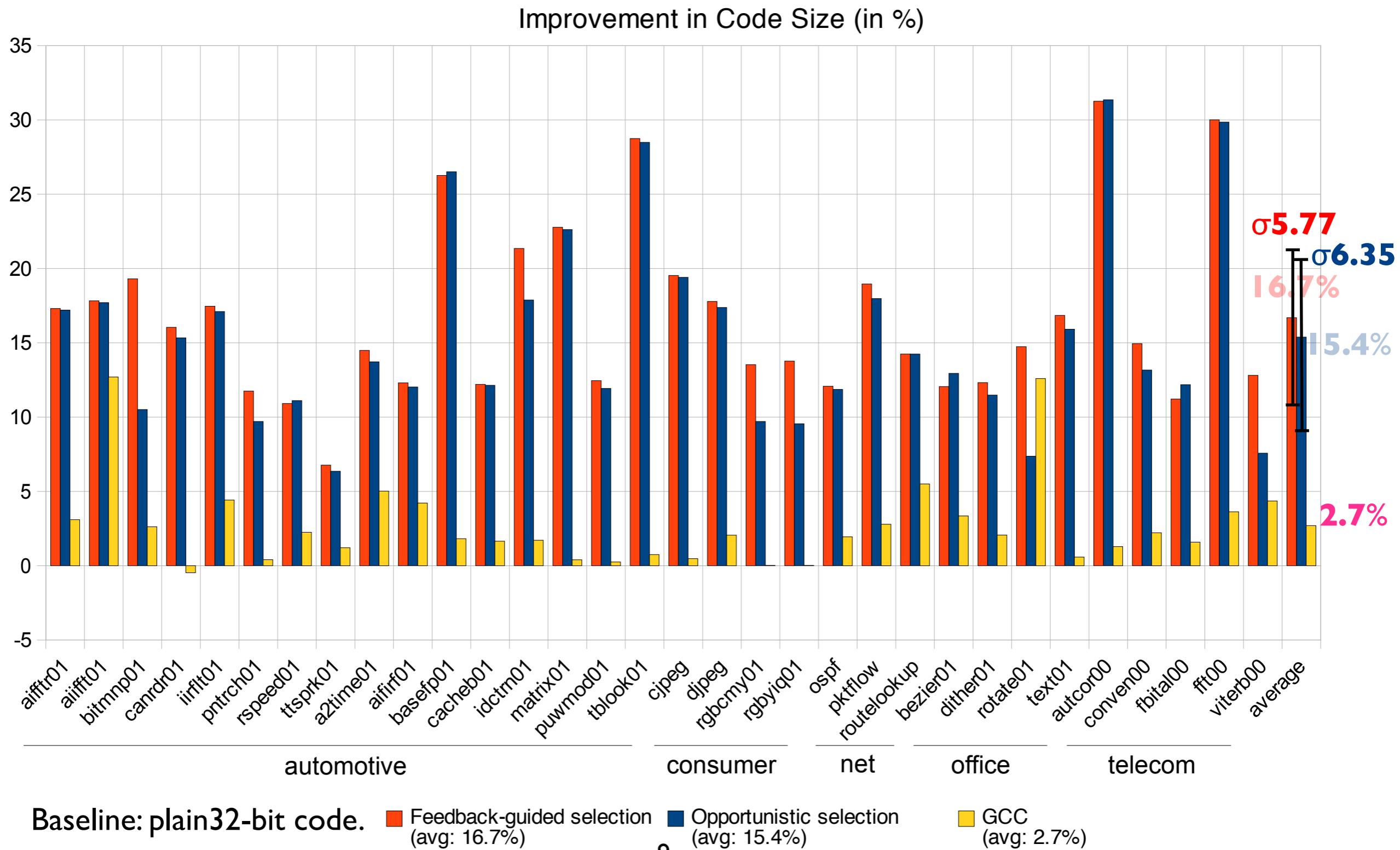


# Evaluation - Code Size Reduction



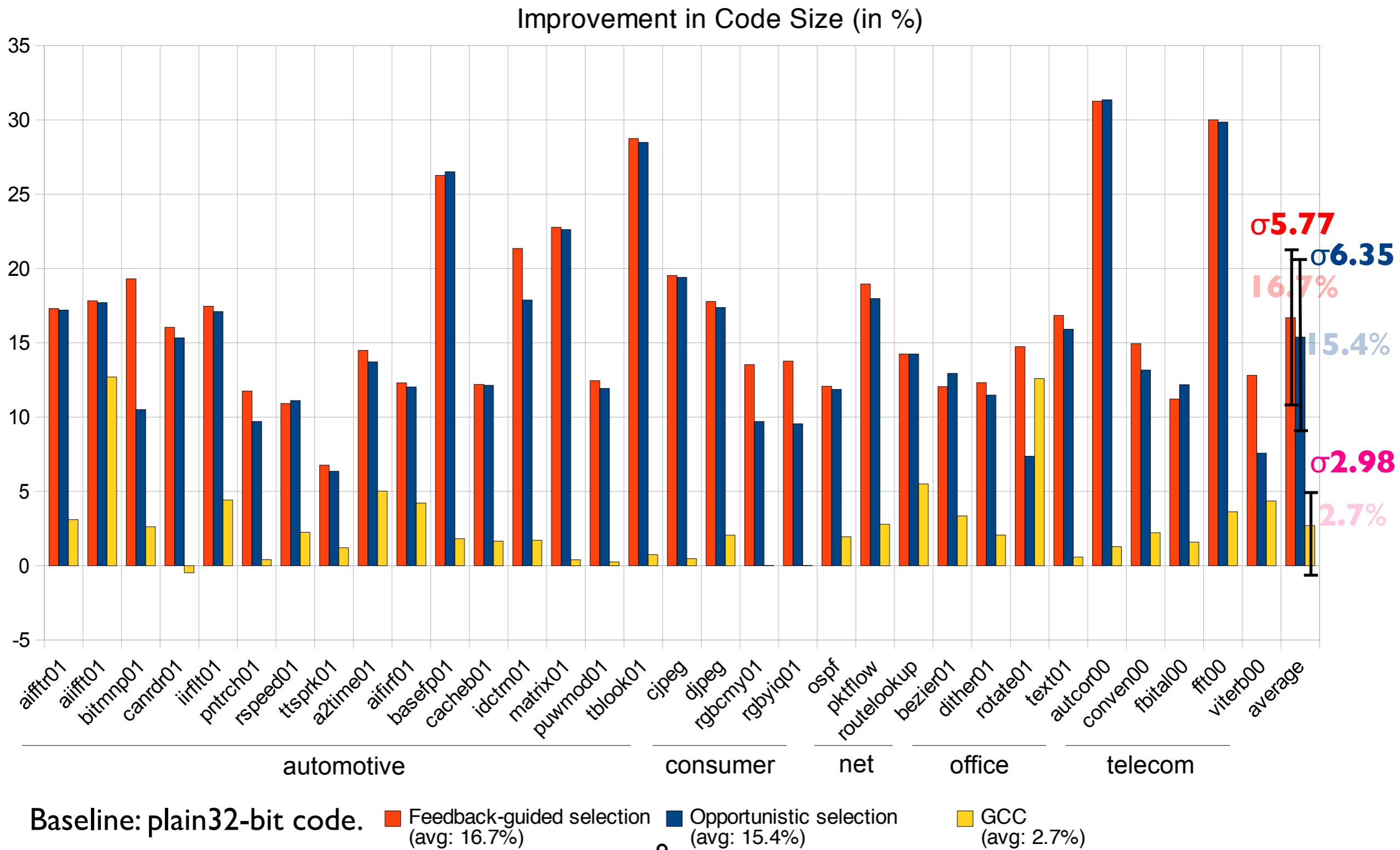


# Evaluation - Code Size Reduction



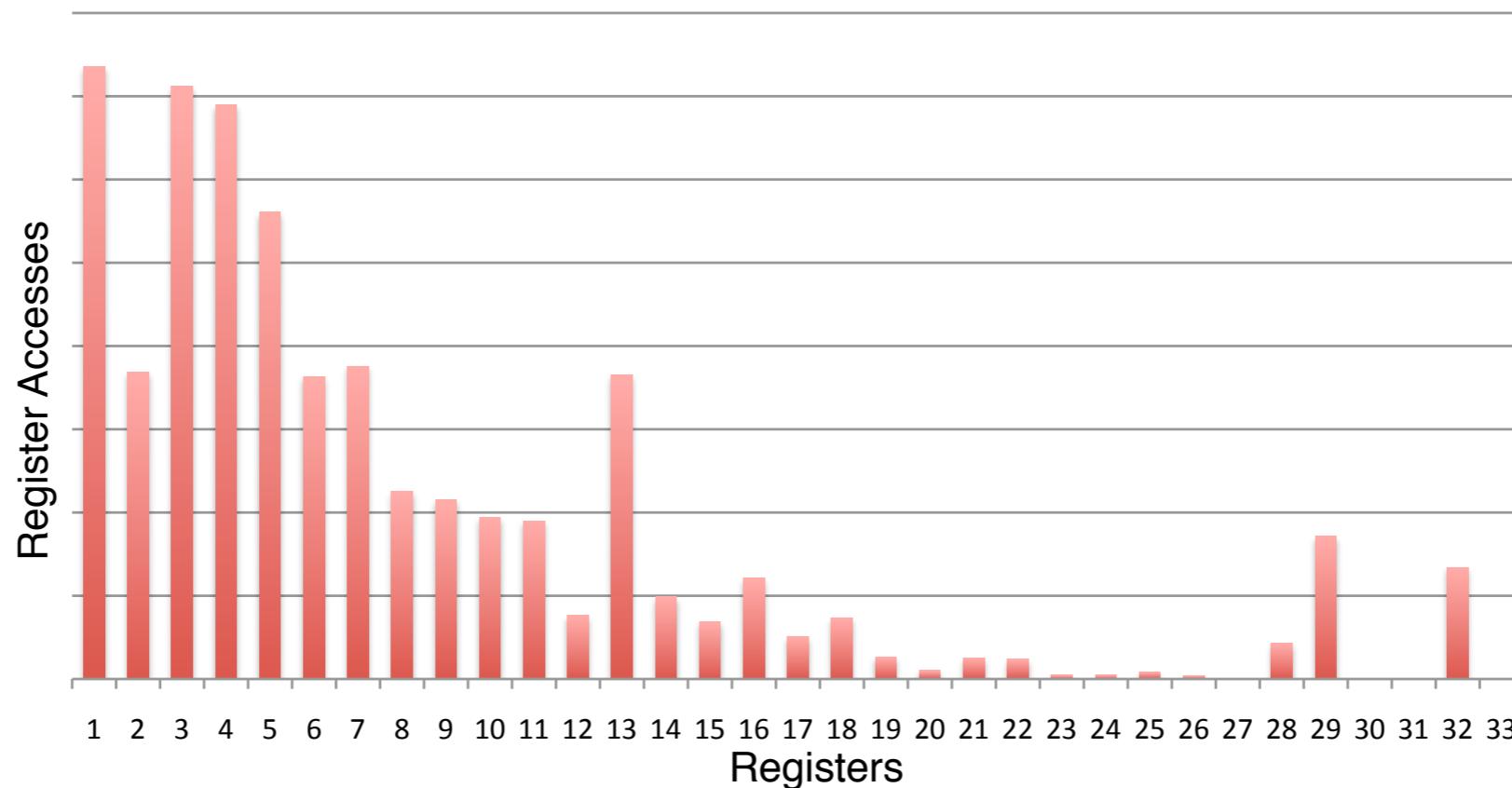


# Evaluation - Code Size Reduction



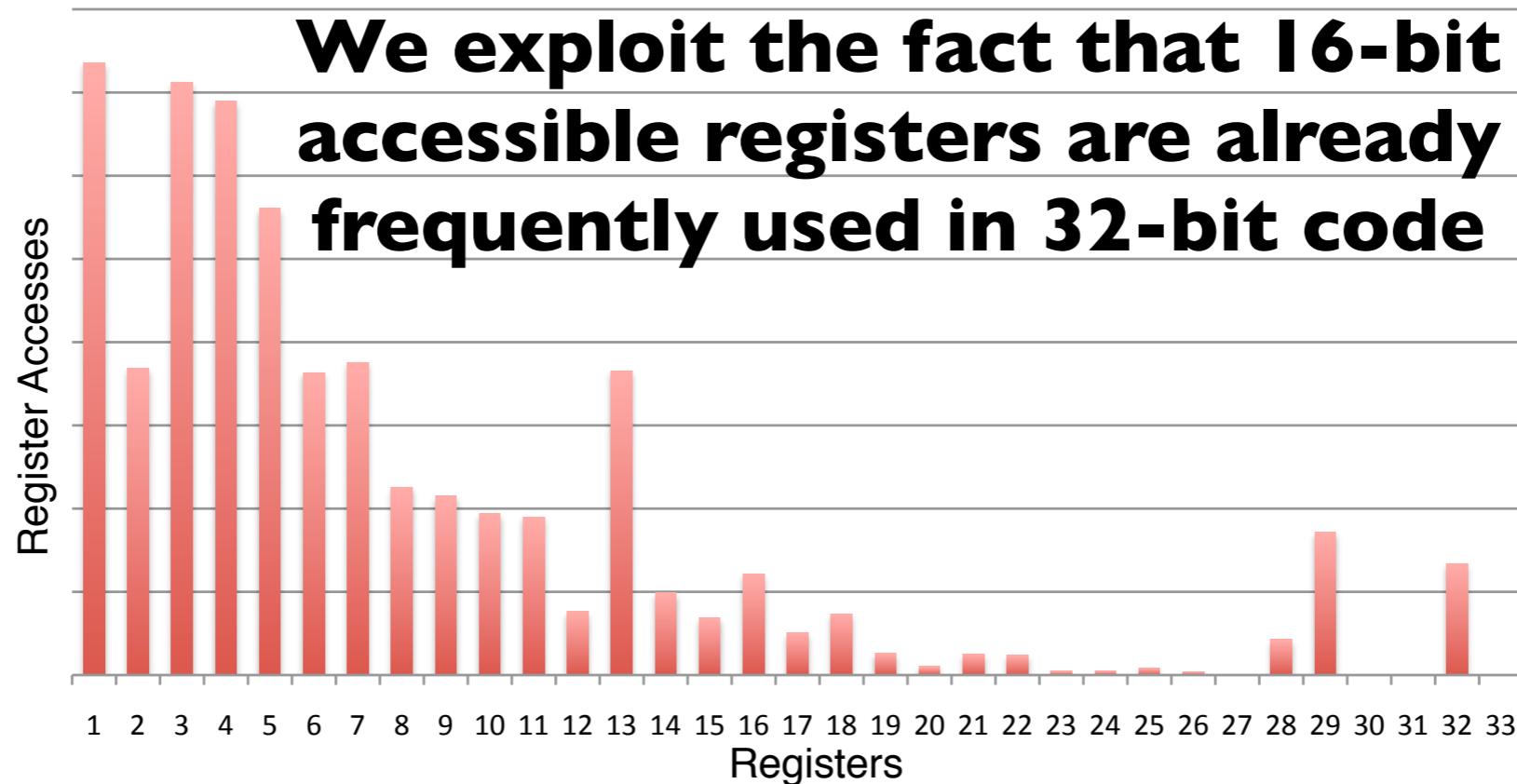


# Why does the simple Opportunistic Mode perform so well?





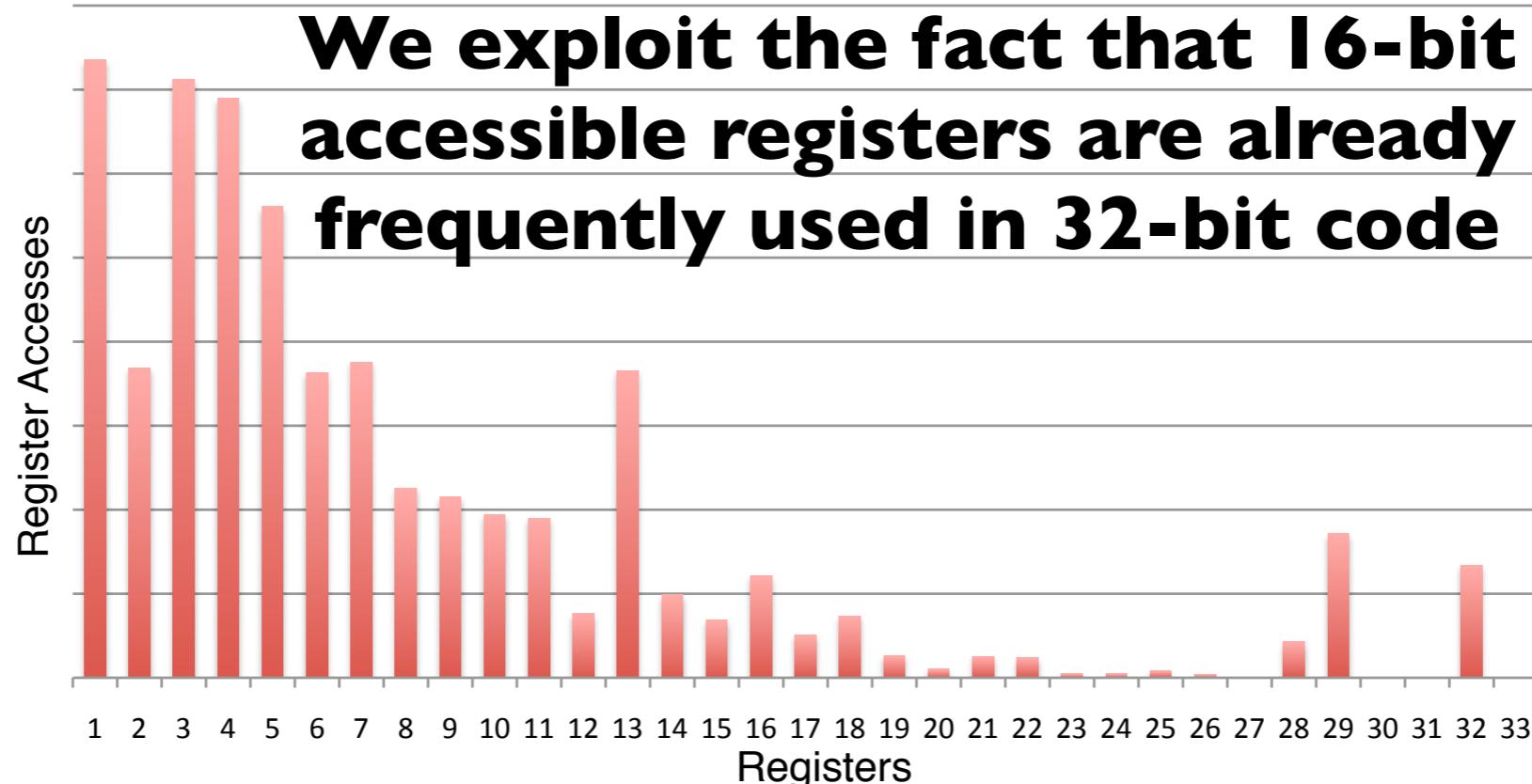
# Why does the simple Opportunistic Mode perform so well?





# Why does the simple Opportunistic Mode perform so well?

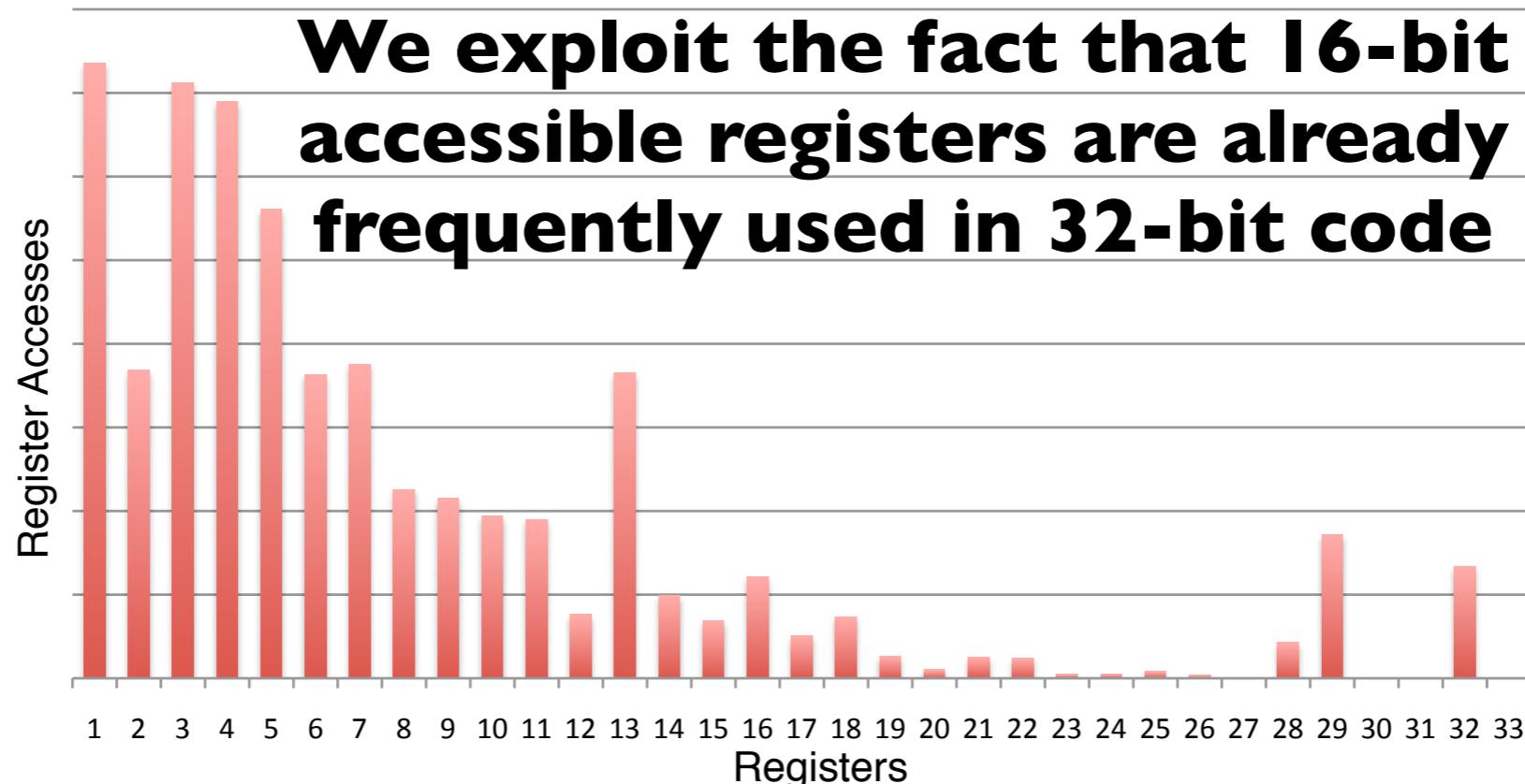
- register allocator selects registers with lower ID from set of possible registers





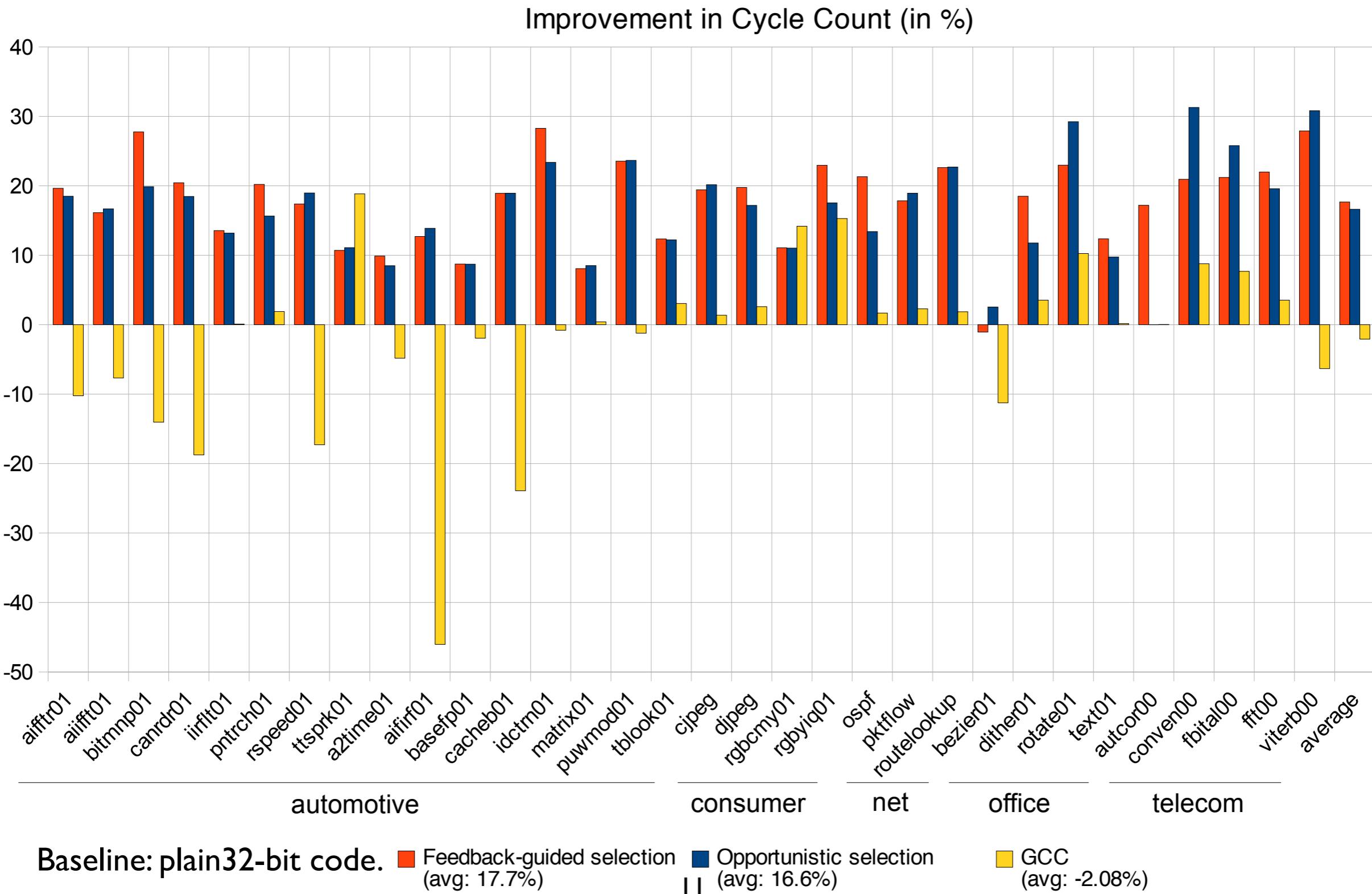
# Why does the simple Opportunistic Mode perform so well?

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



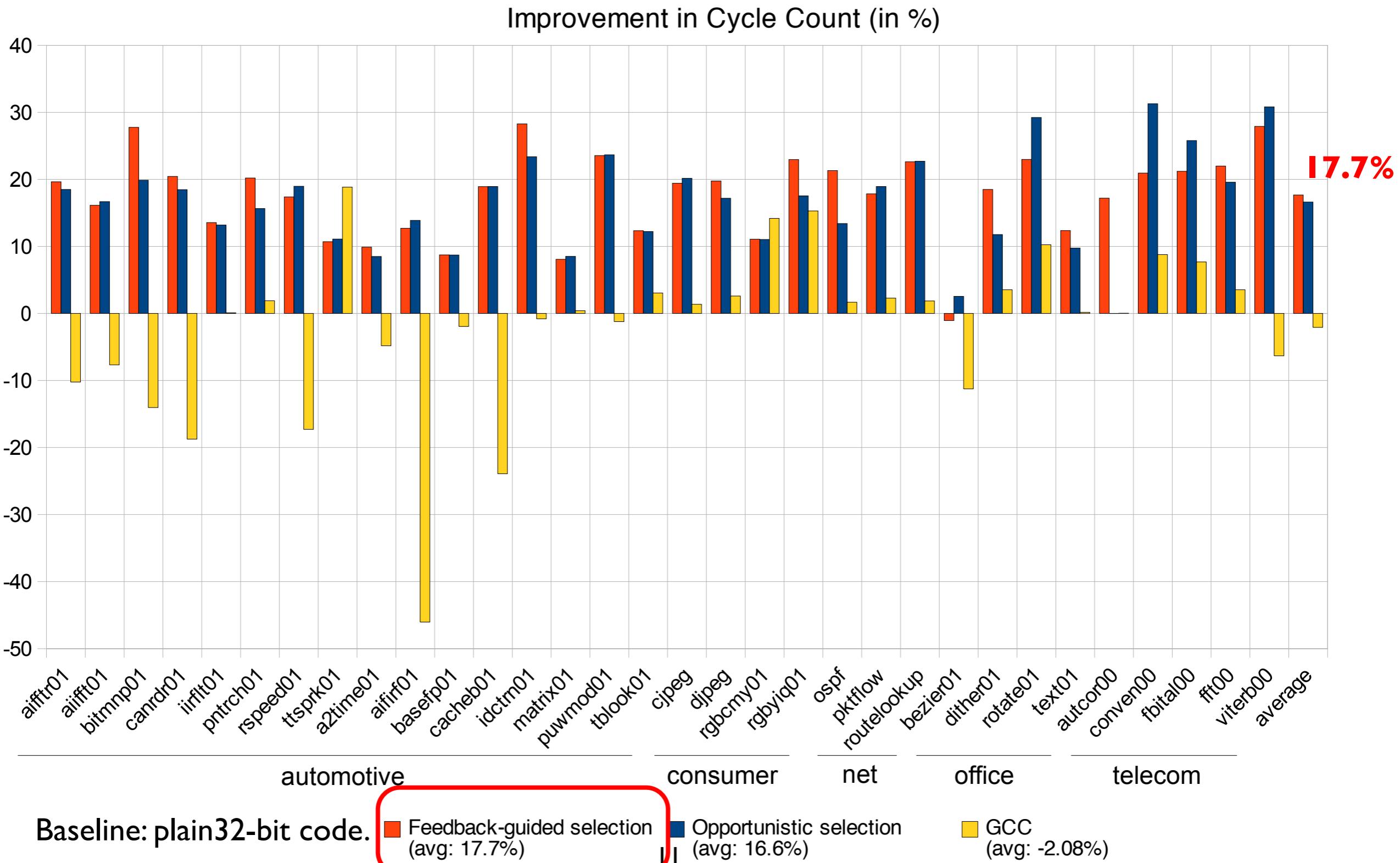


# Evaluation - Performance Improvements



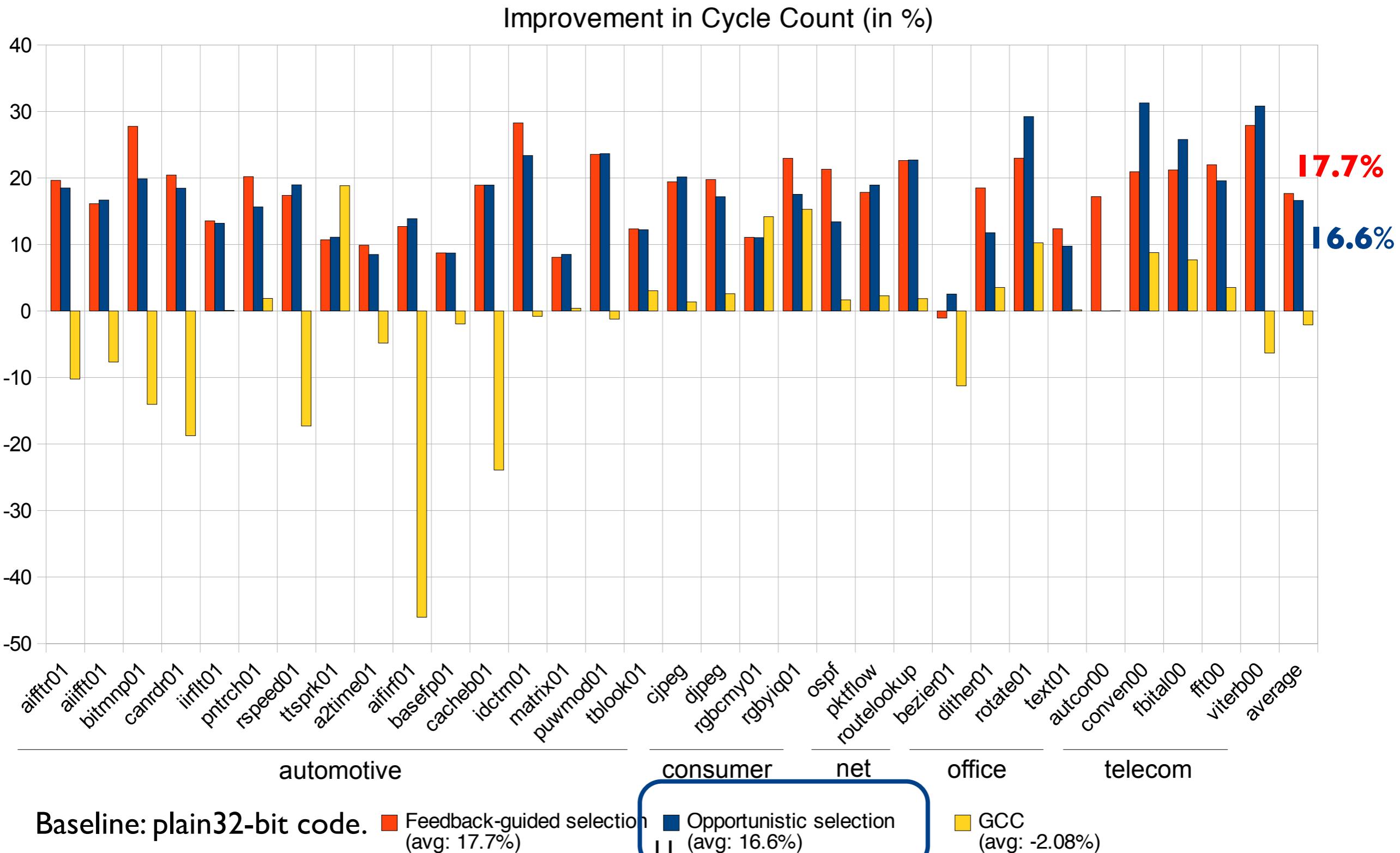


# Evaluation - Performance Improvements



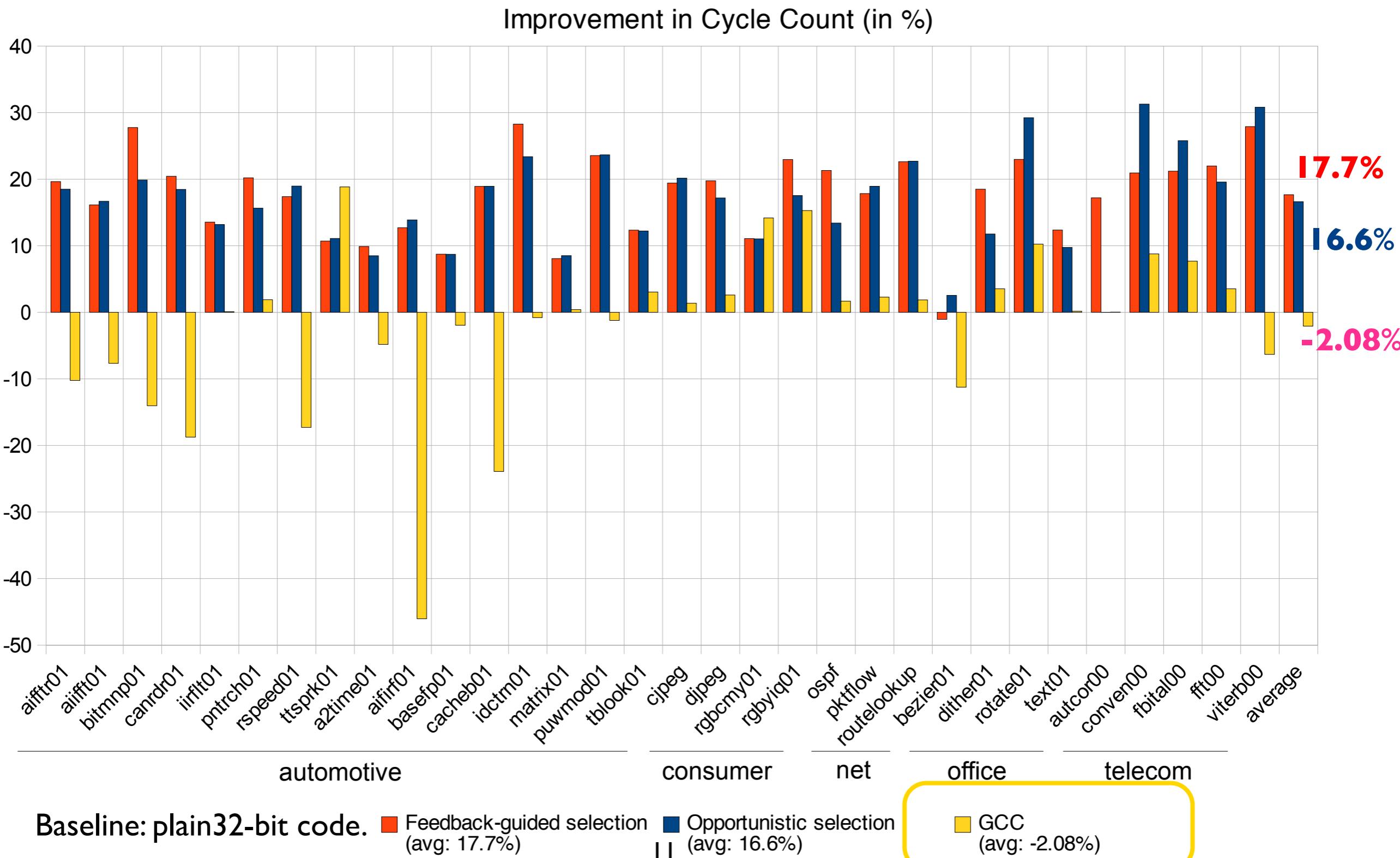


# Evaluation - Performance Improvements



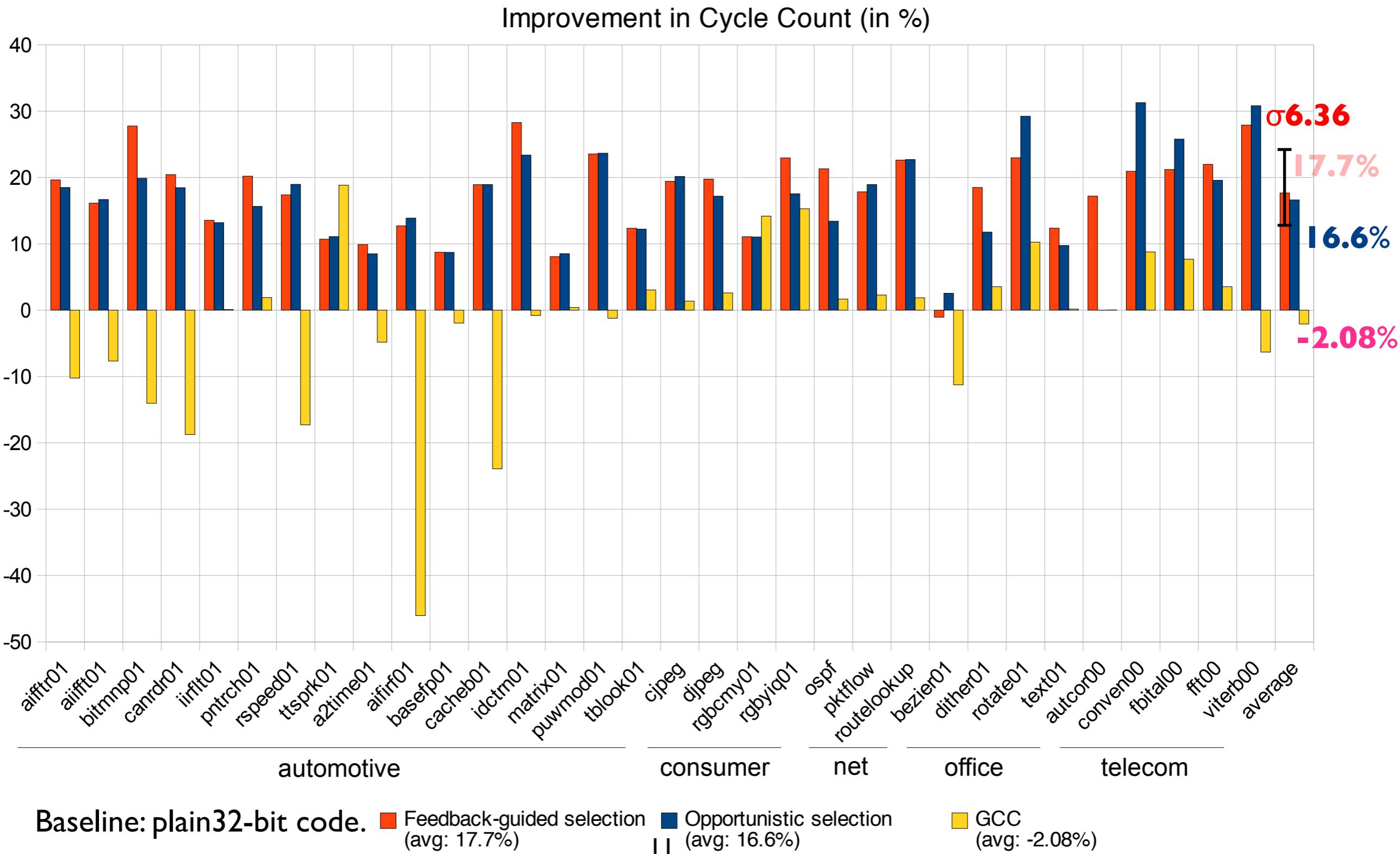


# Evaluation - Performance Improvements



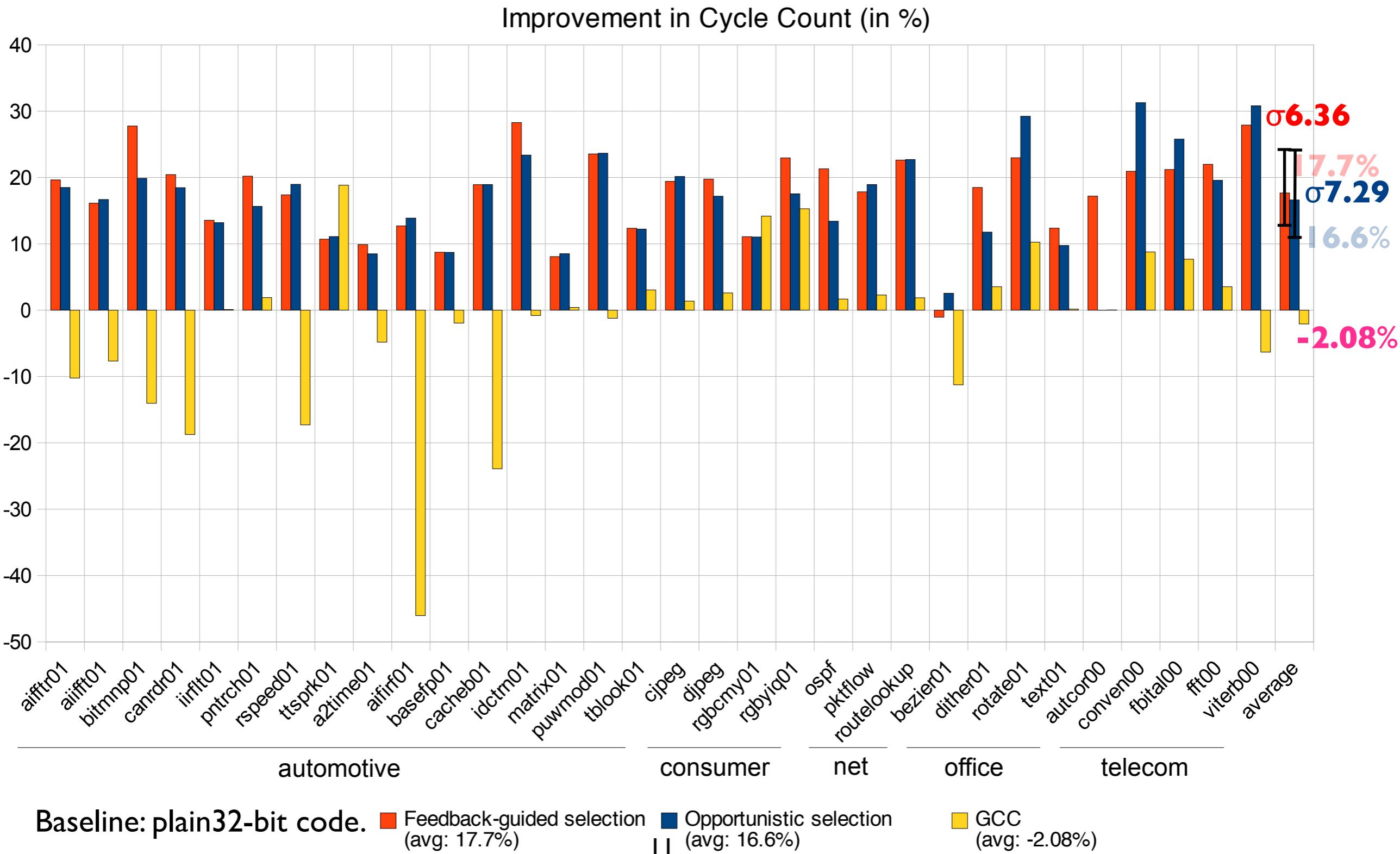


# Evaluation - Performance Improvements



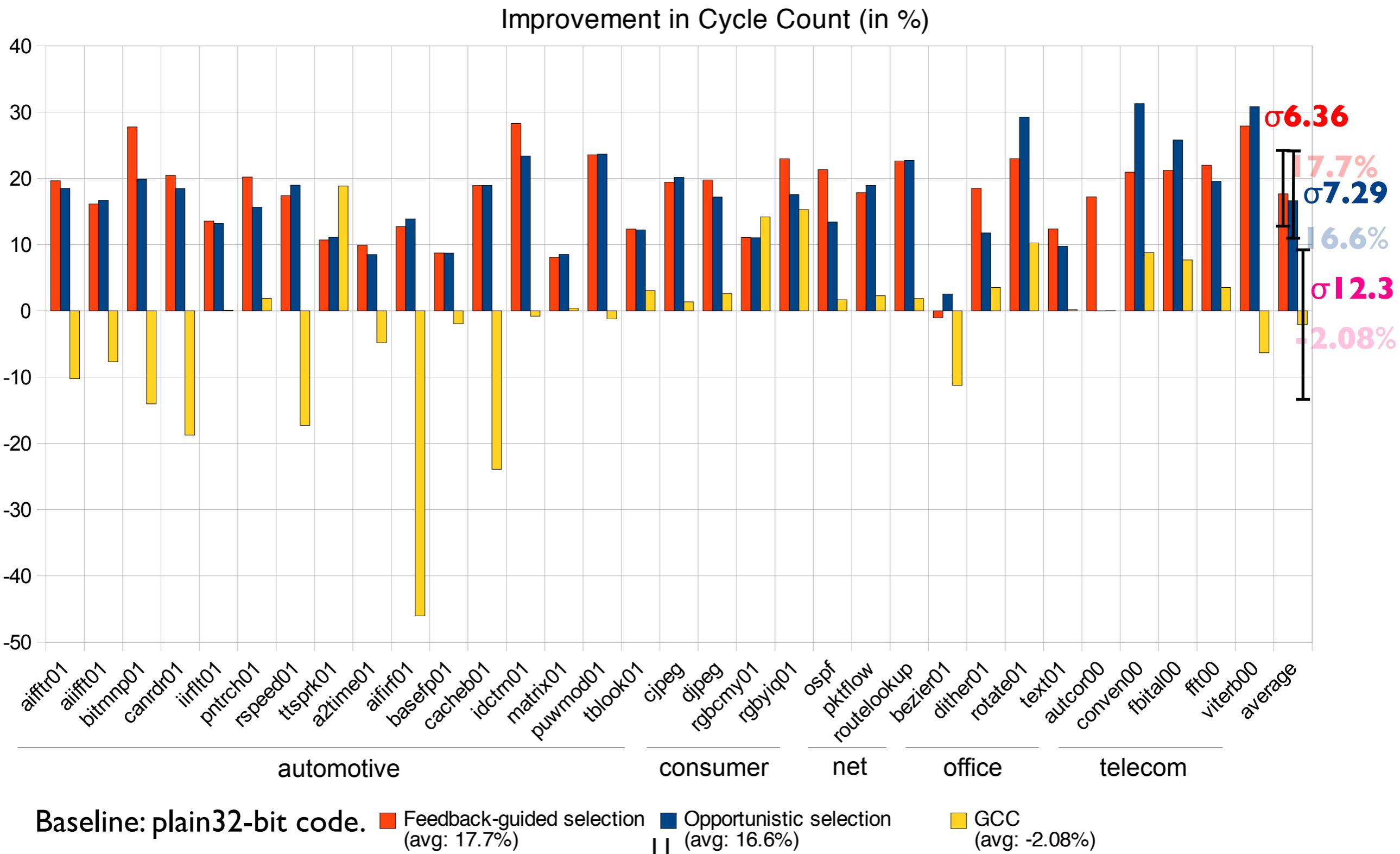


# Evaluation - Performance Improvements





# Evaluation - Performance Improvements





# 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.



<http://groups.inf.ed.ac.uk/pasta/>

Thanks!

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



Engineering and Physical Sciences  
Research Council

<http://groups.inf.ed.ac.uk/pasta/>

 THE UNIVERSITY of EDINBURGH  
**informatics**

**PASTA** Processor Automated Synthesis by iTerative Analysis Project

In the **PASTA** project we seek to *automate* the design and optimisation of *customisable* embedded processors. We do this by creating tools that are able to *learn* about the physical characteristics of the underlying silicon technology and use that knowledge to synthesise the structure of an embedded processor. As the processor is now a flexible entity without a pre-defined instruction set, the compiler for that processor must also be synthesised. Furthermore, the code optimisations that the compiler performs when translating from source code to the synthetic architecture, must also be synthesised.

The three main areas for automated synthesis are:

- The processor architecture,
- the micro-architecture, and
- the compiler.



However, the information on which to make automated decisions in each case will be different. At the micro-architecture level we need to know how each micro-architecture option translates into speed, energy and die area. At the architectural level we need to know how each instruction set option translates into clock cycles of execution time, and at the compiler level we need to know how each optimisation reduces the overall number of instructions executed and maximises the effectiveness of the memory system.

The challenge of our research is that all three areas are *inter-dependent* and ultimately depend on the characteristics of the silicon on which the system is based. By blurring the boundary between hardware and software, and by automating the process of adjusting that boundary, we hope to create a system that can perform design trade-offs in seconds when currently it takes an experienced designer several days.

**icSA** Institute for Computing Systems Architecture

University Homepage  
School Homepage  
School Contacts  
School Search

**icSA**  
Institute for Computing Systems Architecture

Search...  
Home  
News  
Publications  
People  
Seminars  
Contact  
Internal Area

**PASTA Activities**  
EnCore Tools  
ECC Compiler  
GCC Compiler  
EnCore Simulator  
Verification/Co-Simulation

**HW Systems**  
EnCore Processor  
EnCore Calton  
FPGA Platform  
ASIC Flow  
Chips & Boards

**M.Sc. and UG Projects**

**Research Areas**  
Resource Sharing  
Configurable Flow Accelerators  
Automated ISE  
Mapping ISE  
Power Prediction & Optimisation  
Interconnect Synthesis  
Cache Optimisation for Embedded Systems  
Compilation for Dual Memory Banks  
Fast Cycle-Approximate Simulation