Introduction to the LLVM Compiler System

Chris Lattner
llvm.org Architect

November 4, 2008
ACAT'08 - Erice, Sicily

What is the LLVM Project?

- **Collection of industrial strength compiler technology**
  - Optimizer and Code Generator
  - llvm-gcc and Clang Front-ends
  - MSIL and .NET Virtual Machines

http://llvm.org

What is the LLVM Project?

- **Collection of industrial strength compiler technology**
  - Optimizer and Code Generator
  - llvm-gcc and Clang Front-ends
  - MSIL and .NET Virtual Machines
- **Open Source Project with many contributors**
  - Industry, Research Groups, Individuals

http://llvm.org/

Why New Compilers?

http://llvm.org

Why New Compilers?

- Existing Open Source C Compilers have Stagnated!

http://llvm.org

Why New Compilers?

- Existing Open Source C Compilers have Stagnated!

- How?
  - Based on decades old code generation technology
  - No modern techniques like cross-file optimization and JIT codegen
  - Aging code bases: difficult to learn, hard to change substantially
  - Can’t be reused in other applications
  - Keep getting slower with every release

What I want!

http://llvm.org

What I want!

- **A set of production-quality reusable libraries:**
  - ... which implement the best known techniques drawing from modern literature
  - ... which focus on compile time
  - ... and performance of the generated code

- **Ideally support many different languages and applications!**

LLVM Vision and Approach

- **Primary mission: build a set of modular compiler components:**
  - *Reduces the time & cost* to construct a particular compiler
  - Components are *shared* across different compilers
  - Allows choice of the *right component for the job*

http://llvm.org

LLVM Vision and Approach

- **Primary mission: build a set of modular compiler components:**
  - *Reduces the time & cost* to construct a particular compiler
  - Components are *shared* across different compilers
  - Allows choice of the *right component for the job*
- **Secondary mission: Build compilers out of these components**
  - ... for example, a truly great C compiler
  - ... for example, a runtime specialization engine

http://llvm.org

Talk Overview

- Intro and Motivation
- LLVM as a C and C++ Compiler
- Other LLVM Capabilities
- Going Forward

LLVM-GCC 4.2

- C, C++, Objective C, Ada and Fortran
- Standard GCC command line options
- Supports almost all GCC language features and extensions
- Supports many targets, including X86, X86-64, PowerPC, etc.
- Extremely compatible with GCC 4.2

http://llvm.org

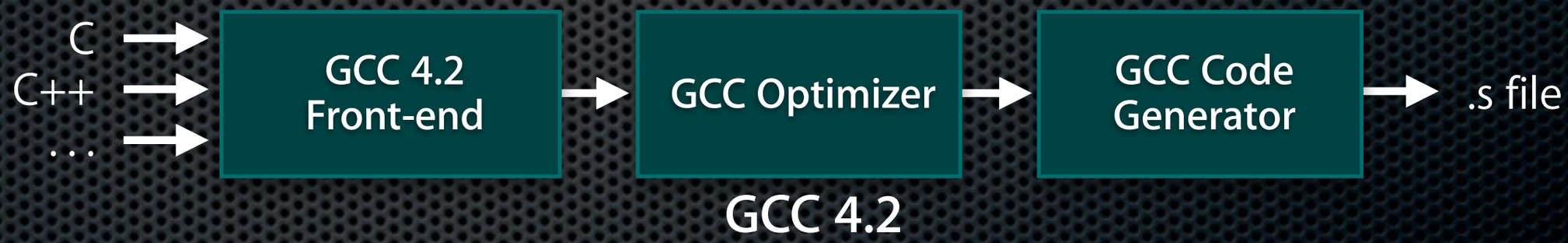
LLVM-GCC 4.2

- C, C++, Objective C, Ada and Fortran
- Standard GCC command line options
- Supports almost all GCC language features and extensions
- Supports many targets, including X86, X86-64, PowerPC, etc.
- Extremely compatible with GCC 4.2

What does it mean to be both LLVM and GCC?

LLVM GCC 4.2 Design

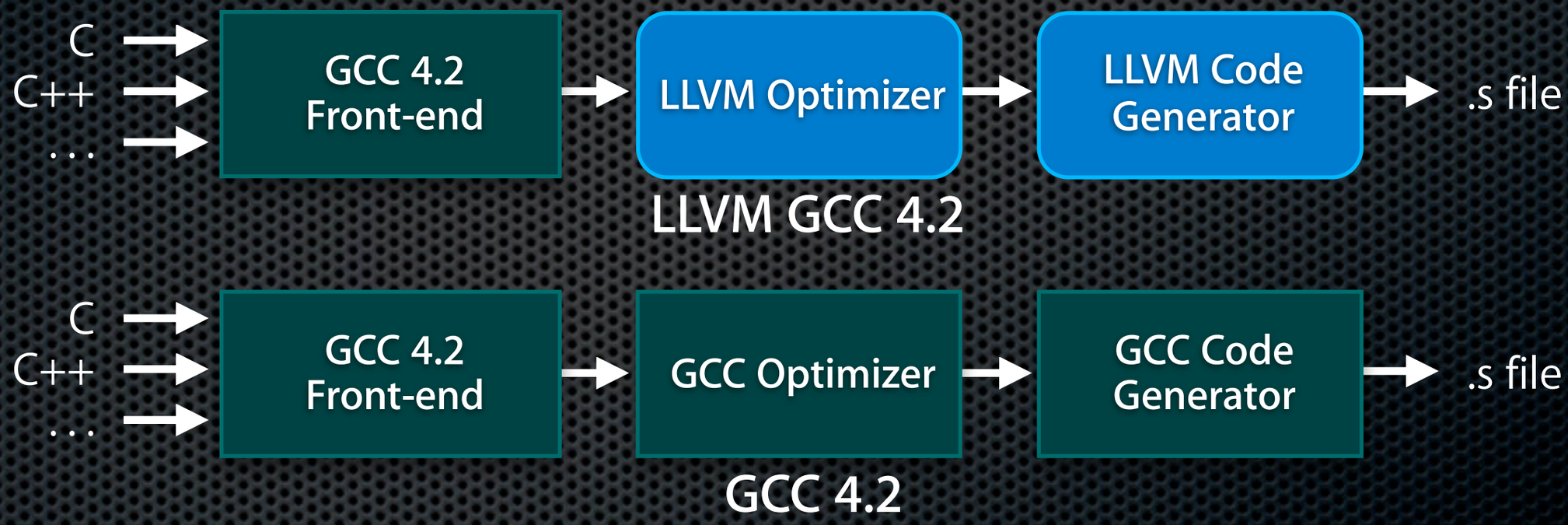
- **Replace GCC optimizer and code generator with LLVM**
  - Reuses GCC parser and runtime libraries



http://llvm.org

LLVM GCC 4.2 Design

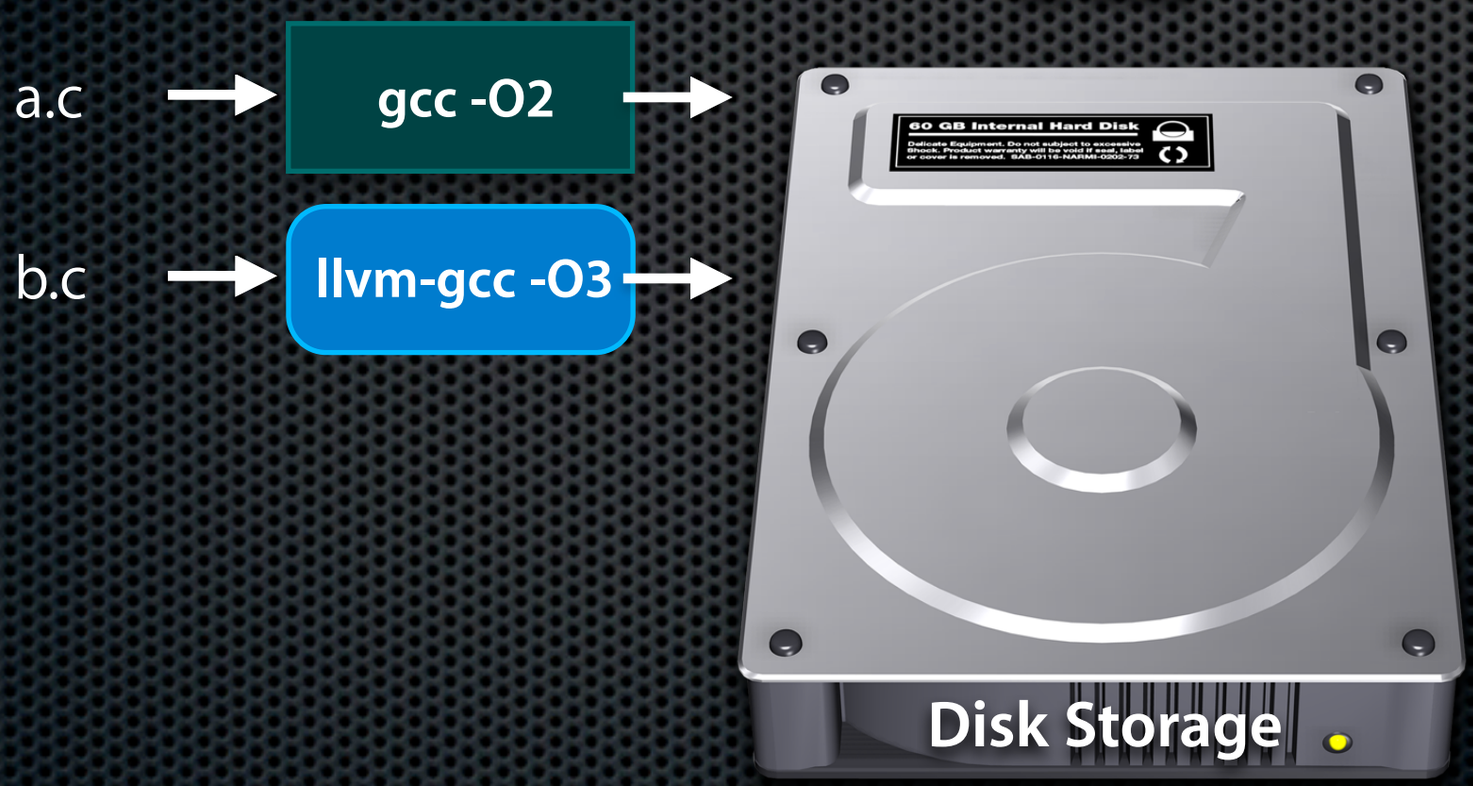
- **Replace GCC optimizer and code generator with LLVM**
  - Reuses GCC parser and runtime libraries



http://llvm.org

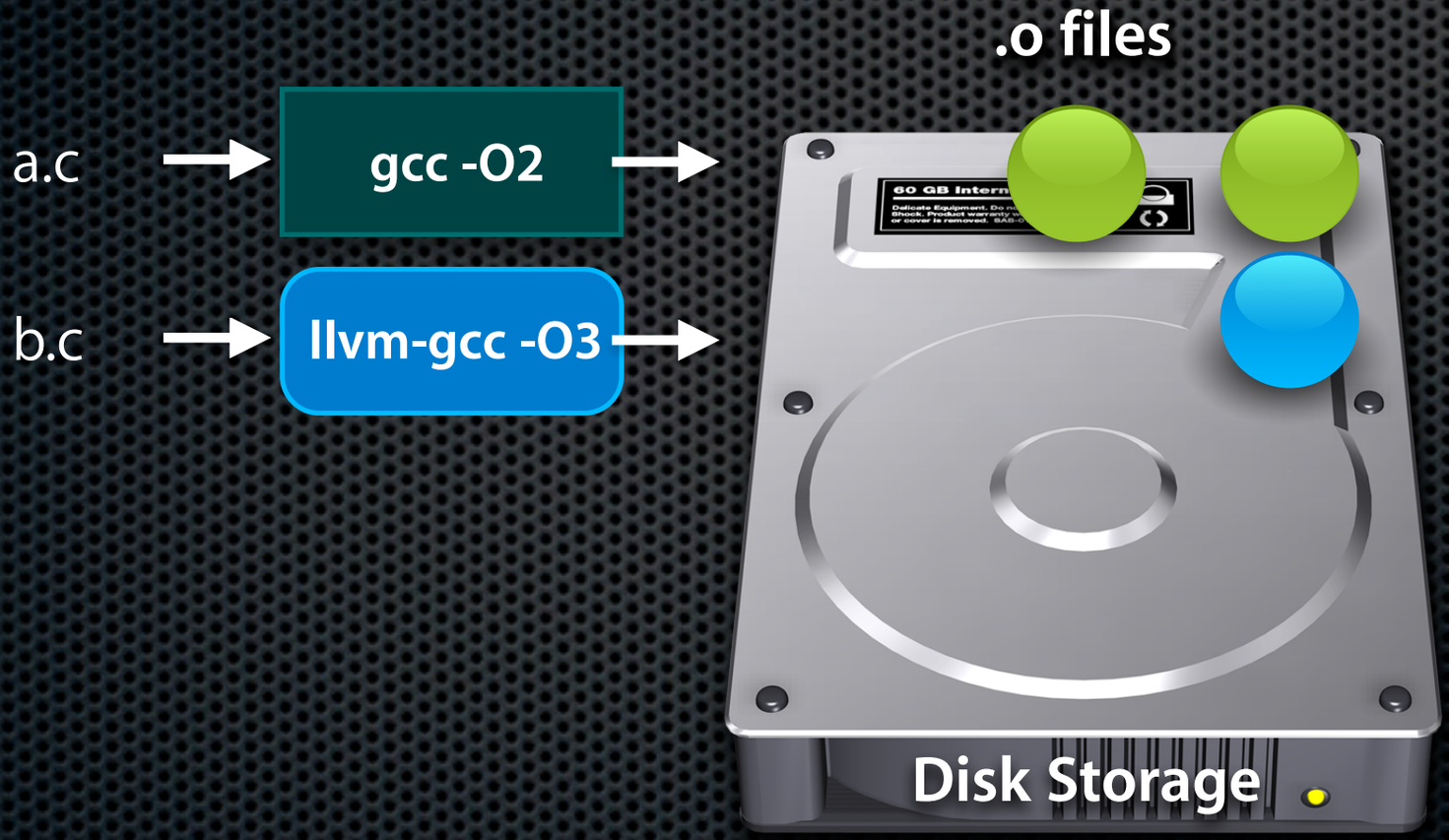
Linking LLVM and GCC compiled code

- Safe to mix and match .o files between compilers
- Safe to call into libraries built with other compilers



Linking LLVM and GCC compiled code

- Safe to mix and match .o files between compilers
- Safe to call into libraries built with other compilers



Linking LLVM and GCC compiled code

- Safe to mix and match .o files between compilers
- Safe to call into libraries built with other compilers

Potential Impact of LLVM Optimizer

- **Generated Code**
  - How fast does the code run?

http://llvm.org

Potential Impact of LLVM Optimizer

- **Generated Code**
  - How fast does the code run?
- **Compile Times**
  - How fast can we get code from the compiler?

Potential Impact of LLVM Optimizer

- **Generated Code**
  - How fast does the code run?
- **Compile Times**
  - How fast can we get code from the compiler?
- **New Features**

Potential Impact of LLVM Optimizer

- **Generated Code**
  - How fast does the code run?

- **Compile Times**
  - How fast can we get code from the compiler?

- New Features

Link Time Optimization

New Feature: Link Time Optimization

- Optimize (e.g. inline, constant fold, etc) across files with -O4
- Optimize across language boundaries too!

New Feature: Link Time Optimization

- Optimize (e.g. inline, constant fold, etc) across files with -O4
- Optimize across language boundaries too!

New Feature: Link Time Optimization

- Optimize (e.g. inline, constant fold, etc) across files with -O4
- Optimize across language boundaries too!

New Feature: Link Time Optimization

- Optimize (e.g. inline, constant fold, etc) across files with -O4
- Optimize across language boundaries too!

SPEC INT 2000 Compile Time

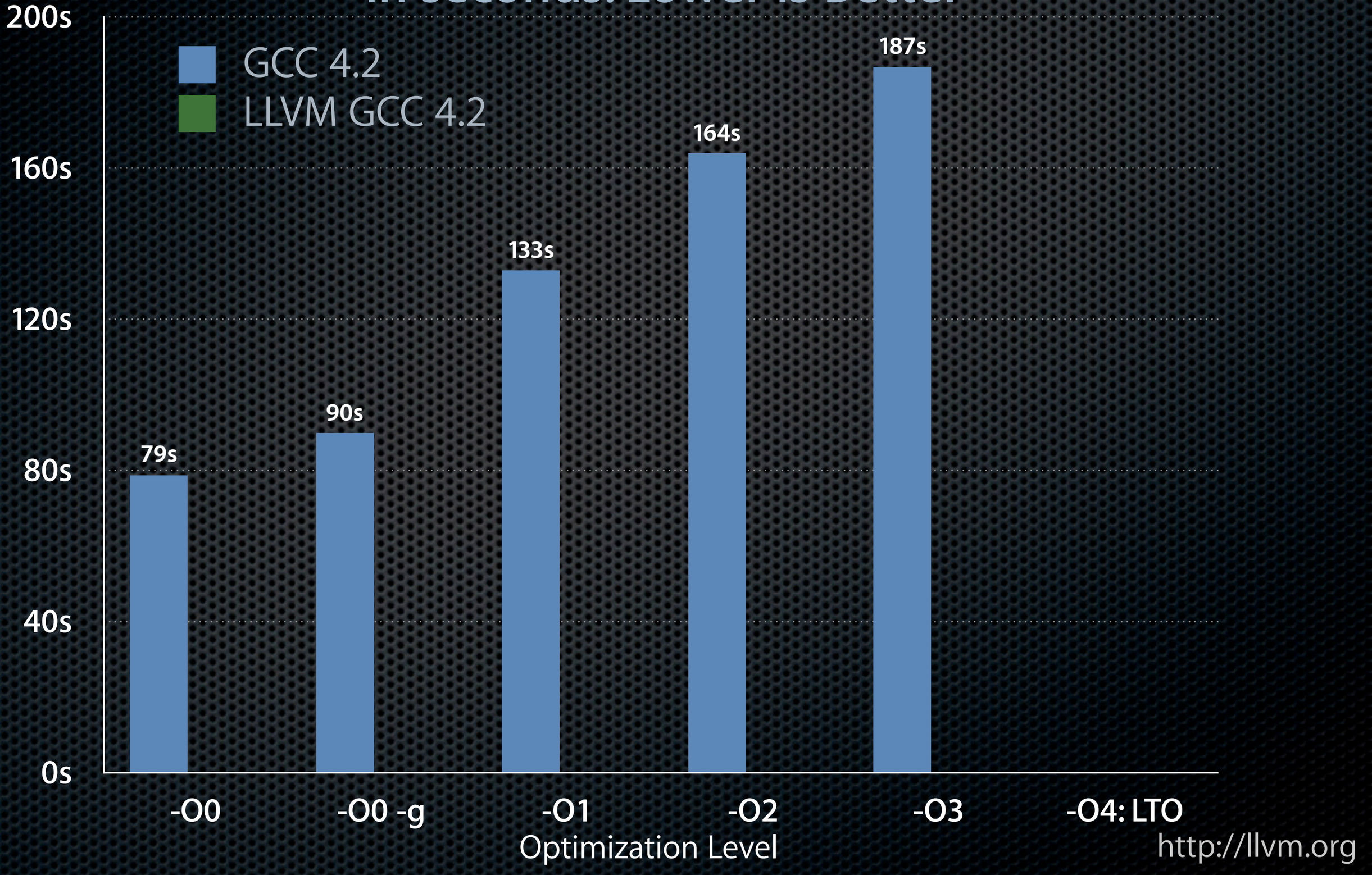
In seconds: Lower is Better

Optimization Level

http://llvm.org

SPEC INT 2000 Compile Time

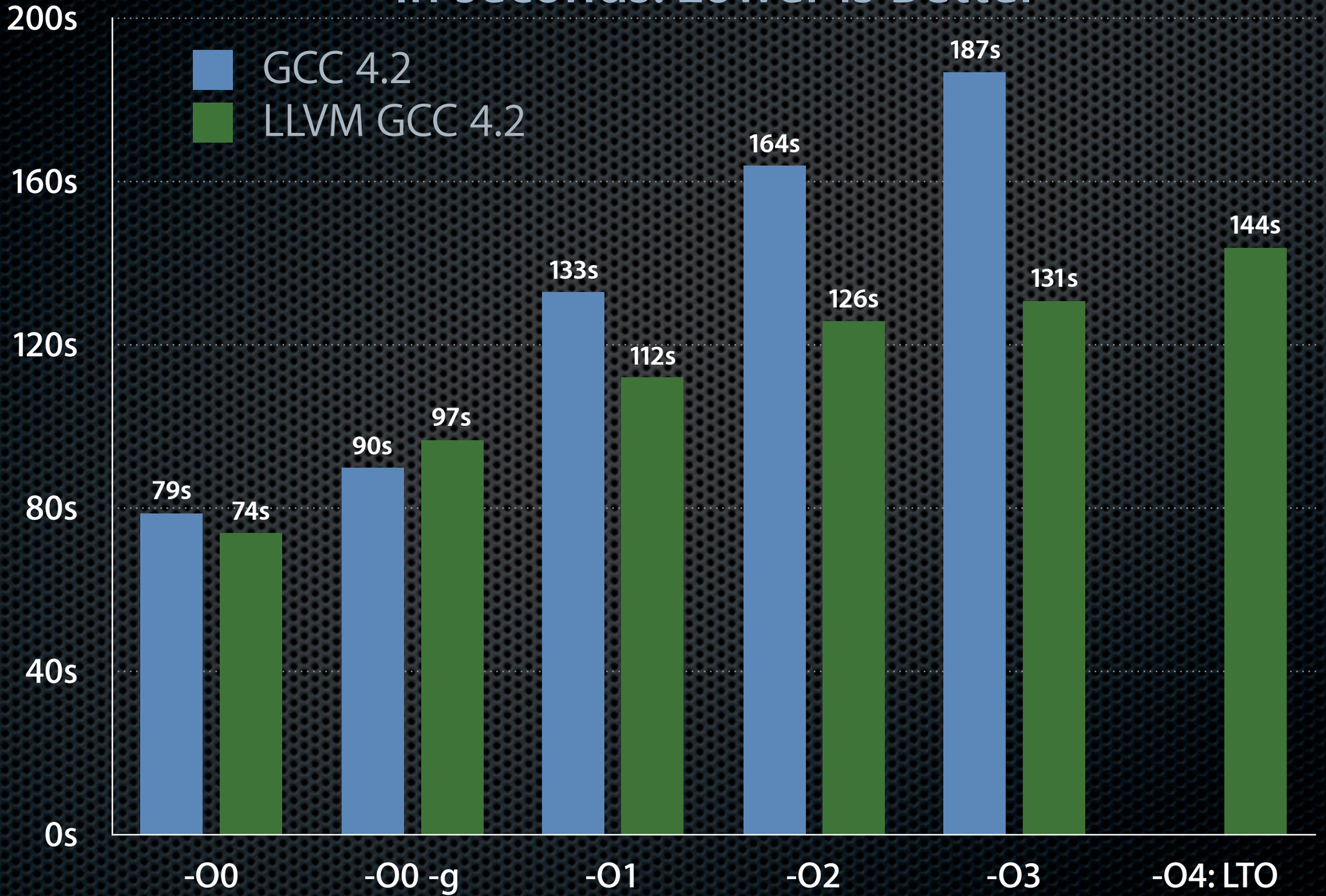
In seconds: Lower is Better



http://llvm.org

SPEC INT 2000 Compile Time

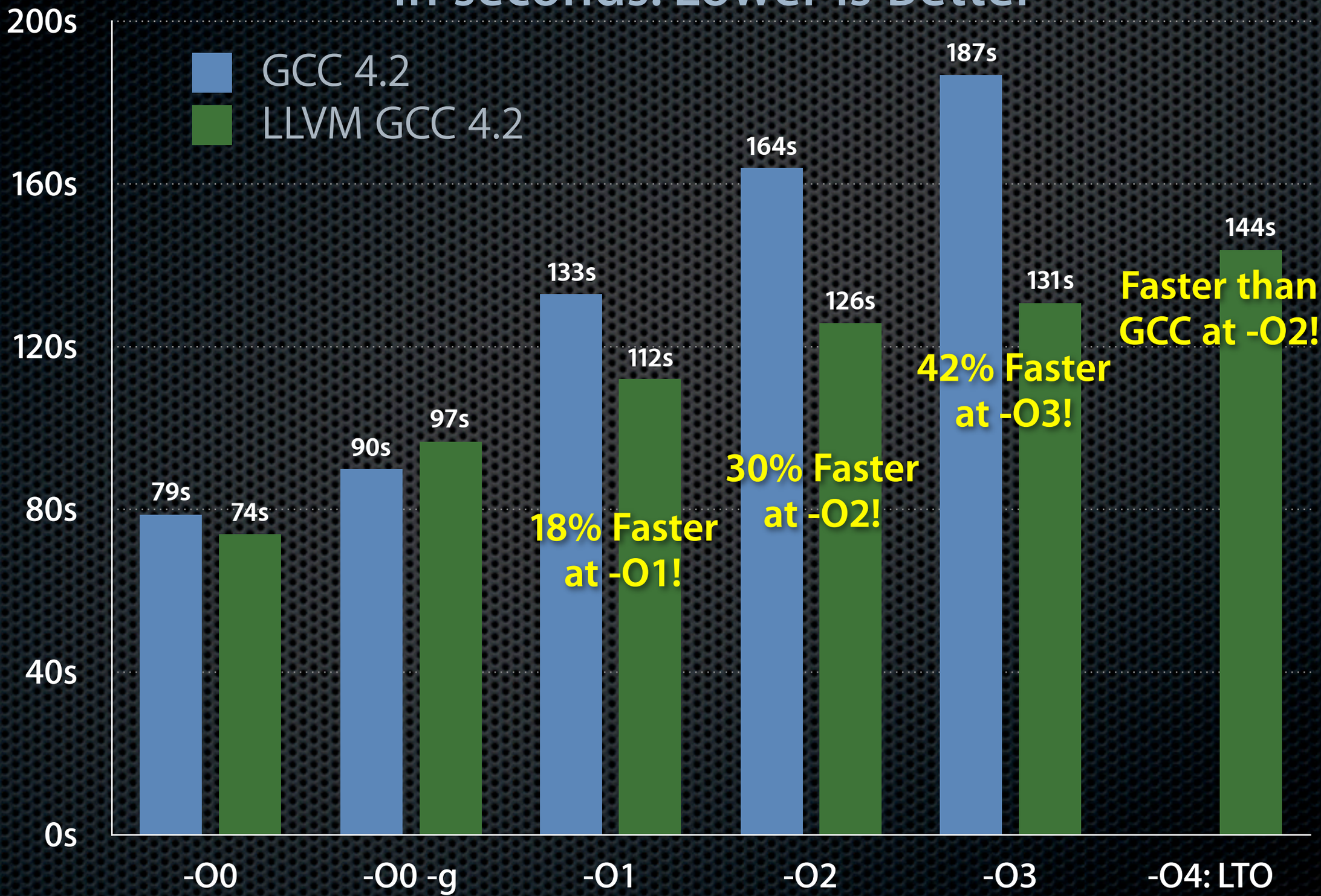
In seconds: Lower is Better



http://llvm.org

SPEC INT 2000 Compile Time

In seconds: Lower is Better



http://llvm.org

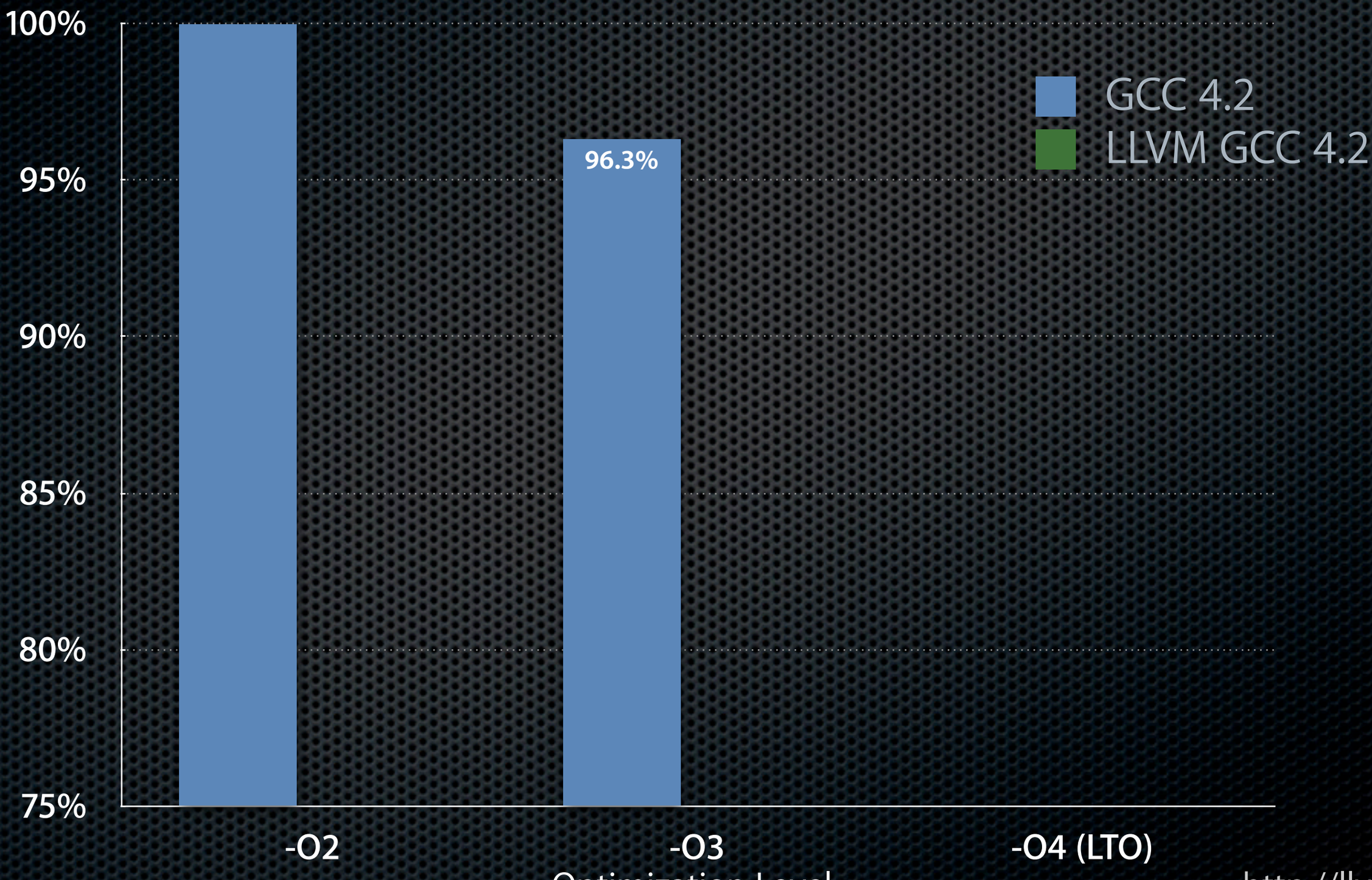
SPEC 2000 Execution Time

Relative to GCC -O2: Lower is Faster

Optimization Level

SPEC 2000 Execution Time

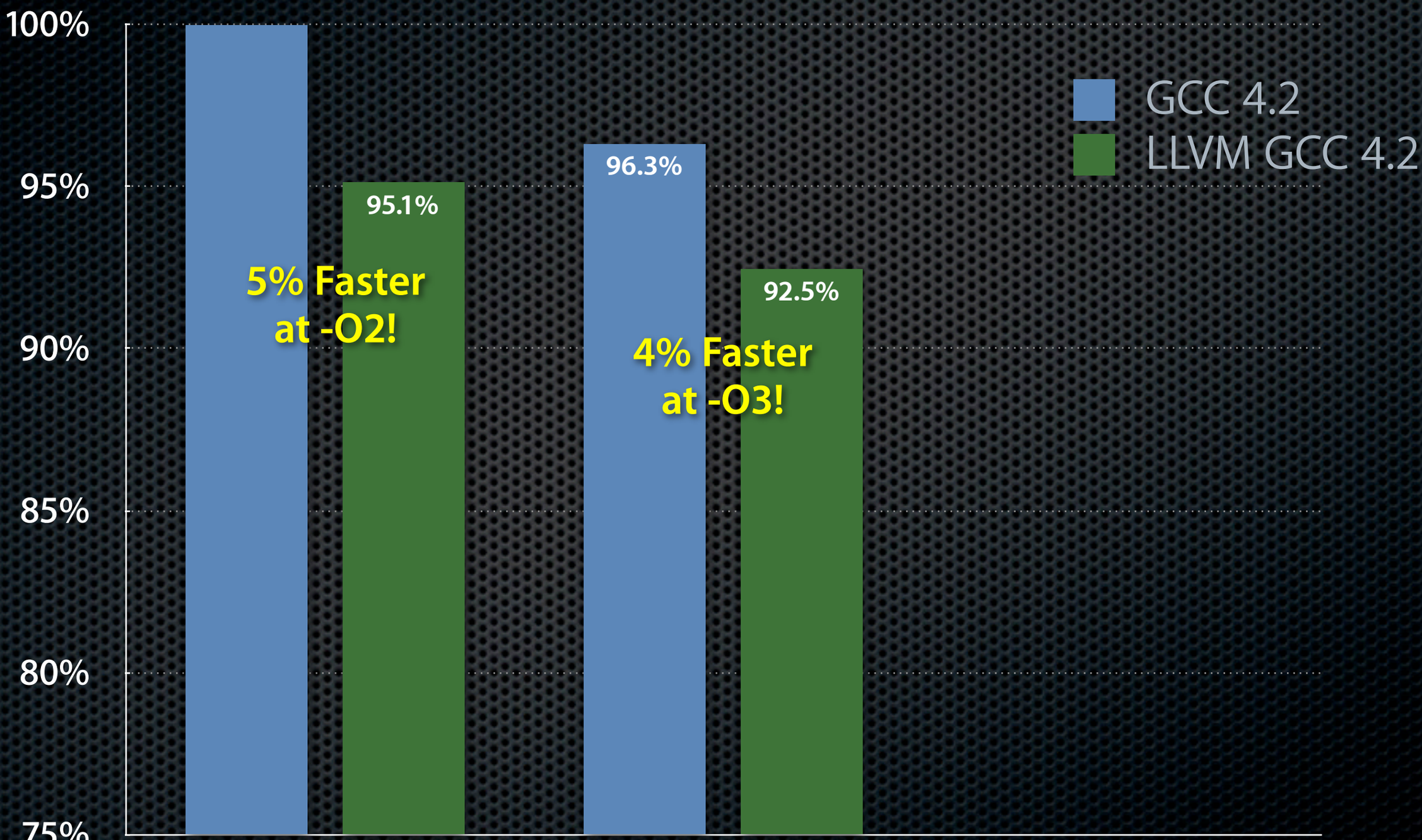
Relative to GCC -O2: Lower is Faster



http://llvm.org

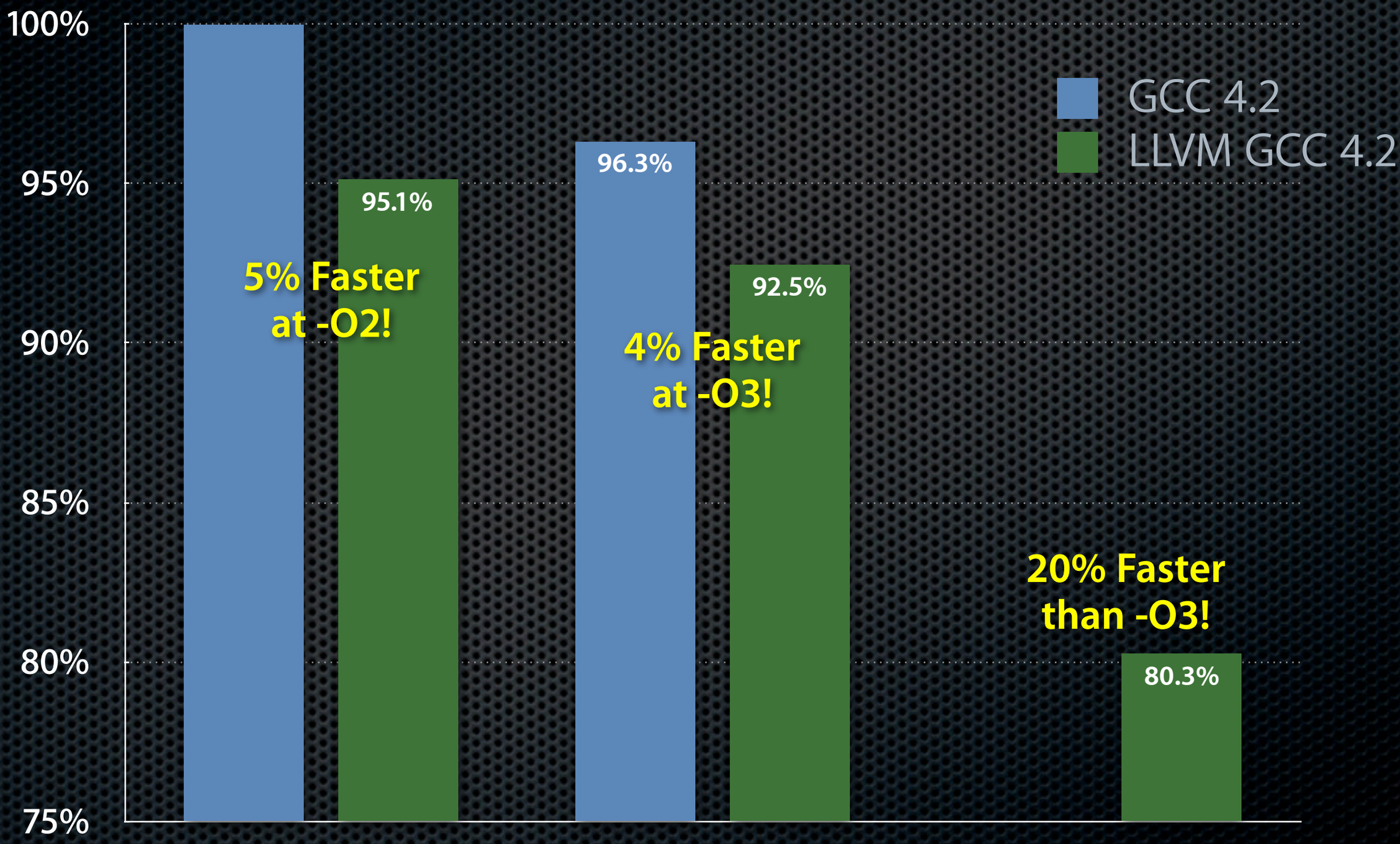
SPEC 2000 Execution Time

Relative to GCC -O2: Lower is Faster



SPEC 2000 Execution Time

Relative to GCC -O2: Lower is Faster



http://llvm.org

llvm-gcc 4.2 Summary

- **Drop in replacement for GCC 4.2**
  - Compatible with GCC command line options and languages
  - Works with existing makefiles (e.g. "make CC=llvm-gcc")

- **Benefits of LLVM Optimizer and Code Generator**
  - Much faster optimizer: ~30-40% at -O3 in most cases
  - Slightly better codegen at a given level: ~5-10% on x86/x86-64
  - Link-Time Optimization at -O4: optimize across source files

Talk Overview

- Intro and Motivation
- LLVM as a C and C++ Compiler
- Other LLVM Capabilities
- LLVM Going Forward

LLVM For Compiler Hackers

- **LLVM is a great target for new languages**
  - Well defined, simple to program for
  - Easy to retarget existing compiler to use LLVM backend

- **LLVM supports Just-In-Time optimization and compilation**
  - Optimize code at runtime based on dynamic information
  - Easy to retarget existing bytecode interpreter to LLVM JIT
  - Great for performance, not just for traditional "compilers"

http://llvm.org

Colorspace Conversion JIT Optimization

- Code to convert from one color format to another:
  - e.g. BGRA 444R -> RGBA 8888
  - Hundreds of combinations, importance depends on input

```
for each pixel {
  switch (infmt) {
  case RGBA 5551:
    R = (*in >> 11) & C
    G = (*in >> 6) & C
    B = (*in >> 1) & C
  ... }
  switch (outfmt) {
  case RGB888:
    *outptr = R << 16 |
              G << 8 ...
  }
}
```

Colorspace Conversion JIT Optimization

- Code to convert from one color format to another:
  - e.g. BGRA 444R -> RGBA 8888
  - Hundreds of combinations, importance depends on input

```
for each pixel {
  switch (infmt) {
  case RGBA 5551:
    R = (*in >> 11) & C
    G = (*in >> 6) & C
    B = (*in >> 1) & C
  ... }
  switch (outfmt) {
  case RGB888:
    *outptr = R << 16 |
              G << 8 ...
  }
}
```

Run-time specialize

```
for each pixel {
  R = (*in >> 11) & C;
  G = (*in >> 6) & C;
  B = (*in >> 1) & C;
  *outptr = R << 16 |
            G << 8 ...
}
```

Compiler optimizes shifts and masking

Colorspace Conversion JIT Optimization

- Code to convert from one color format to another:
  - e.g. BGRA 444R -> RGBA 8888
  - Hundreds of combinations, importance depends on input

```
for each pixel {
  switch (infmt) {
  case RGBA 5551:
    R = (*in >> 11) & C
    G = (*in >> 6) & C
    B = (*in >> 1) & C
  ... }
  switch (outfmt) {
  case RGB888:
    *outptr = R << 16 |
              G << 8 ...
  }
}
```

Run-time specialize

```
for each pixel {
  R = (*in >> 11) & C;
  G = (*in >> 6) & C;
  B = (*in >> 1) & C;
  *outptr = R << 16 |
            G << 8 ...
}
```

Compiler optimizes shifts and masking

Speedup depends on src/dest format:

5.4x speedup on average, 19.3x max speedup: (13.3MB/s to 257.7MB/s)

Another example: RegEx Compilation

- **Many regex's are matched millions of times:**
  - Match time is critical
- **Common regex engines 'compile' to 'bytecode' and interpret:**
  - regcomp/regexec
- **Why not compile to native code? Partial Evaluation!**
  - regcomp compiles regex to a native function
  - Much faster matching, could even vectorize common idioms
- **Excellent way to handle multiple different Unicode encodings**

Talk Overview

- Intro and Motivation
- LLVM as a C and C++ Compiler
- Other LLVM Capabilities
- LLVM Going Forward

LLVM Going Forward

- More of the same...

http://llvm.org

LLVM Going Forward

- **More of the same...**
  - Even faster optimizer
  - Even better optimizations
  - More features for non-C languages
  - Debug Info Improvements
  - Many others...

http://llvm.org

LLVM Going Forward

- **More of the same...**
  - Even faster optimizer
  - Even better optimizations
  - More features for non-C languages
  - Debug Info Improvements
  - Many others...

Better tools for source level analysis of C/C++ programs!

Clang Frontend: What is it?

- C, Objective-C, and C++ front-end

- Aggressive project with many goals...
  - Compatibility with GCC
  - Fast compilation
  - Expressive error messages

- Host for a broad range of source-level tools

Clang Frontend: What is it?

- C, Objective-C, and C++ front-end

- Aggressive project with many goals...
  - Compatibility with GCC
  - Fast compilation
  - Expressive error messages

```
t.c:6:49: error: invalid operands to binary expression ('int' and 'struct A')
  return intArg + func(intArg ? ((someA.X+40) + someA) / 42 : someA.X));
                               ~~~~~~~~~~~~~ ^ ~~~~~
```

- Host for a broad range of source-level tools

Clang Frontend: What is it?

- C, Objective-C, and C++ front-end

- Aggressive project with many goals...
  - Compatibility with GCC
  - Fast compilation
  - Expressive error messages

```
t.c:6:49: error: invalid operands to binary expression ('int' and 'struct A')
  return intArg + func(intArg ? ((someA.X+40) + someA) / 42 : someA.X));
                               ~~~~~~~~~~~~~ ^ ~~~~~
```

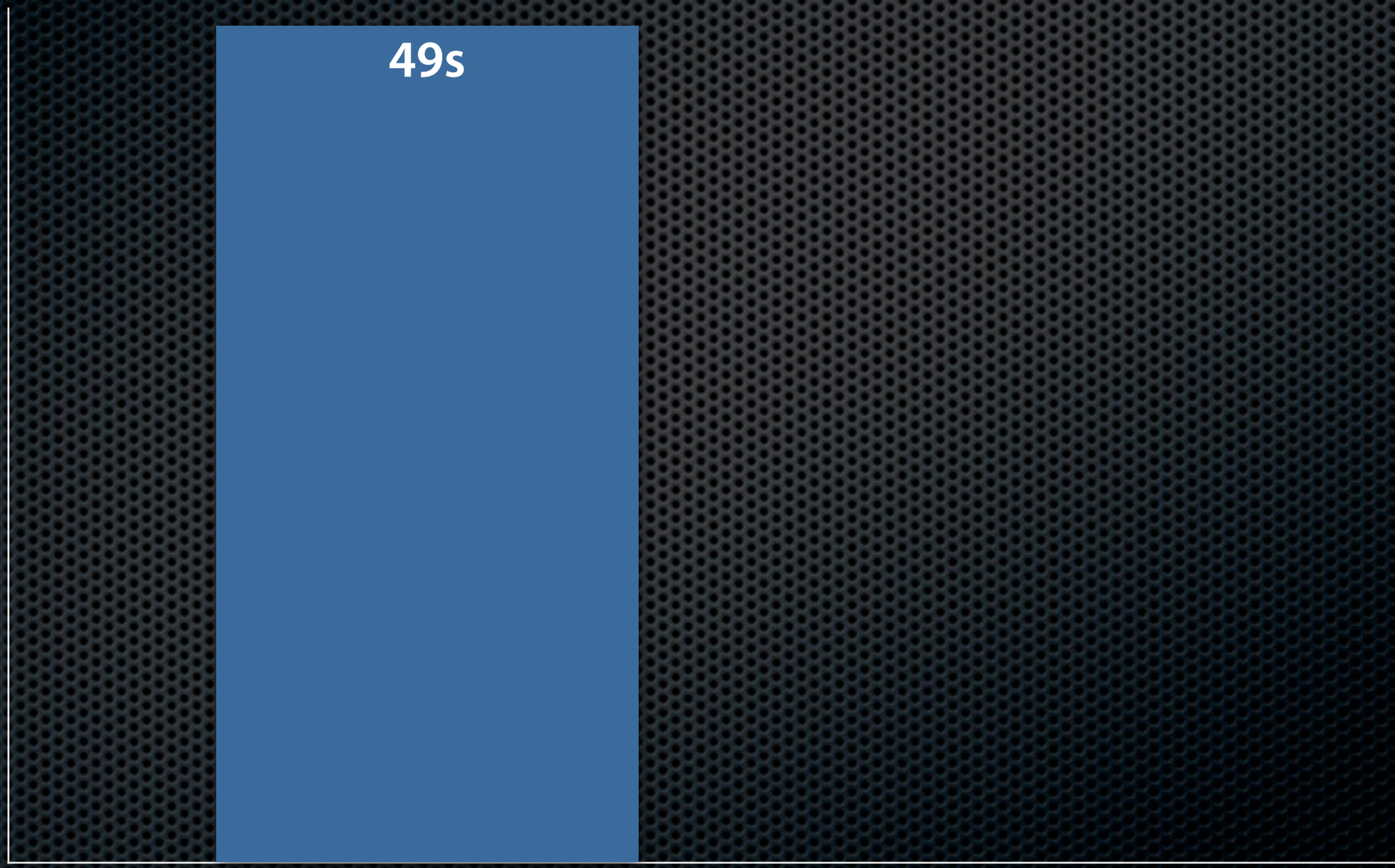
- Host for a broad range of source-level tools

Clang Compile Time

PostgreSQL -fsyntax-only Time: 665K lines of C code in 619 files

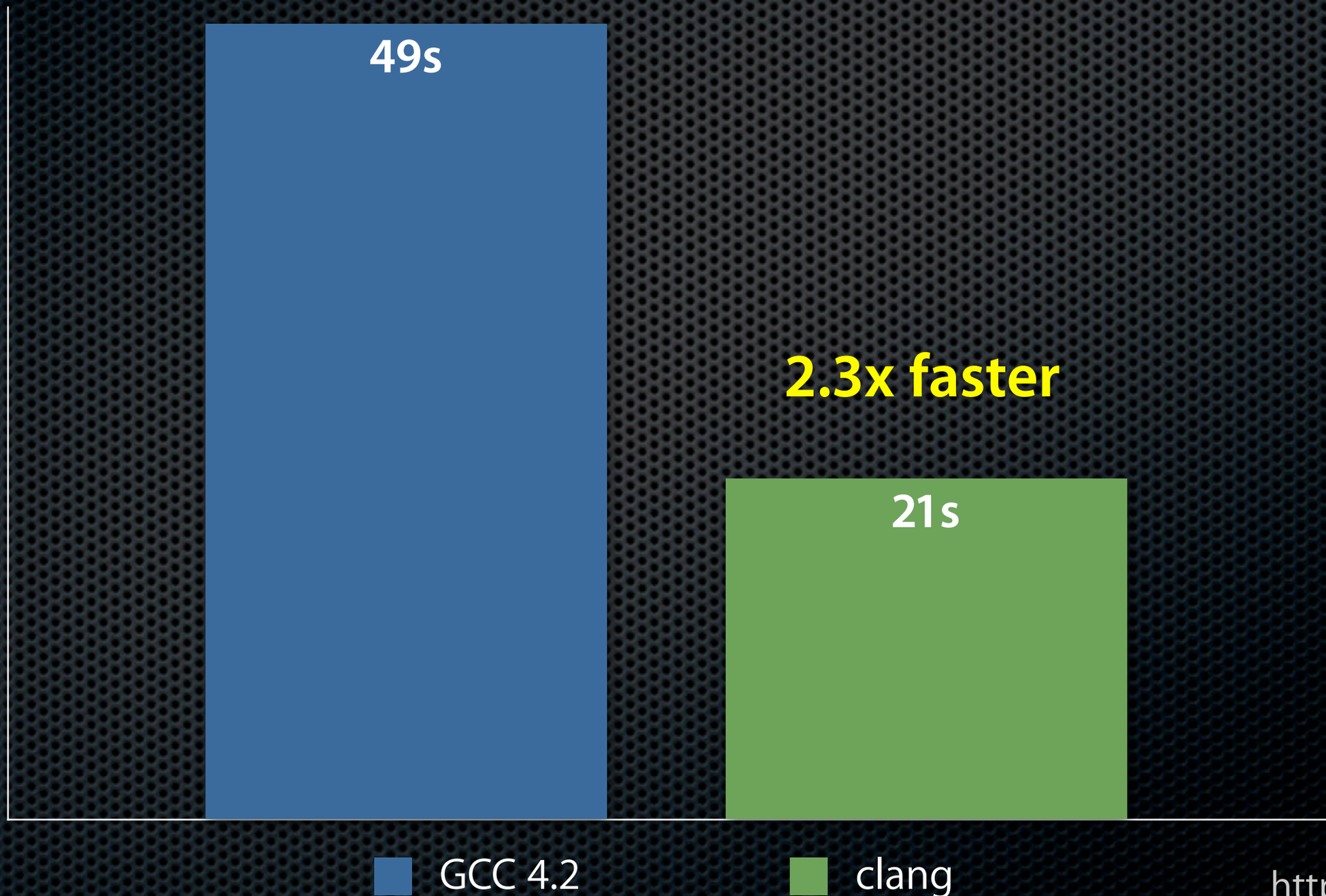
Clang Compile Time

PostgreSQL -fsyntax-only Time: 665K lines of C code in 619 files



Clang Compile Time

PostgreSQL -fsyntax-only Time: 665K lines of C code in 619 files



http://llvm.org

LLVM Overview

- New compiler architecture built with reusable components
  - Retarget existing languages to JIT or static compilation
  - Many optimizations and supported targets

http://llvm.org

LLVM Overview

- New compiler architecture built with reusable components
  - Retarget existing languages to JIT or static compilation
  - Many optimizations and supported targets

- llvm-gcc: drop in GCC-compatible compiler
  - Better & faster optimizer
  - Production quality

http://llvm.org

LLVM Overview

- New compiler architecture built with reusable components
  - Retarget existing languages to JIT or static compilation
  - Many optimizations and supported targets

- llvm-gcc: drop in GCC-compatible compiler
  - Better & faster optimizer
  - Production quality

- Clang front-end: C/ObjC/C++ front-end
  - Several times faster than GCC
  - Much better end-user features (warnings/errors)

http://llvm.org

LLVM Overview

- New compiler architecture built with reusable components
  - Retarget existing languages to JIT or static compilation
  - Many optimizations and supported targets

- llvm-gcc: drop in GCC-compatible compiler
  - Better & faster optimizer
  - Production quality

- Clang front-end: C/ObjC/C++ front-end
  - Several times faster than GCC
  - Much better end-user features (warnings/errors)

- LLVM 2.4 release this week!

Come join us at:
http://llvm.org
http://clang.llvm.org