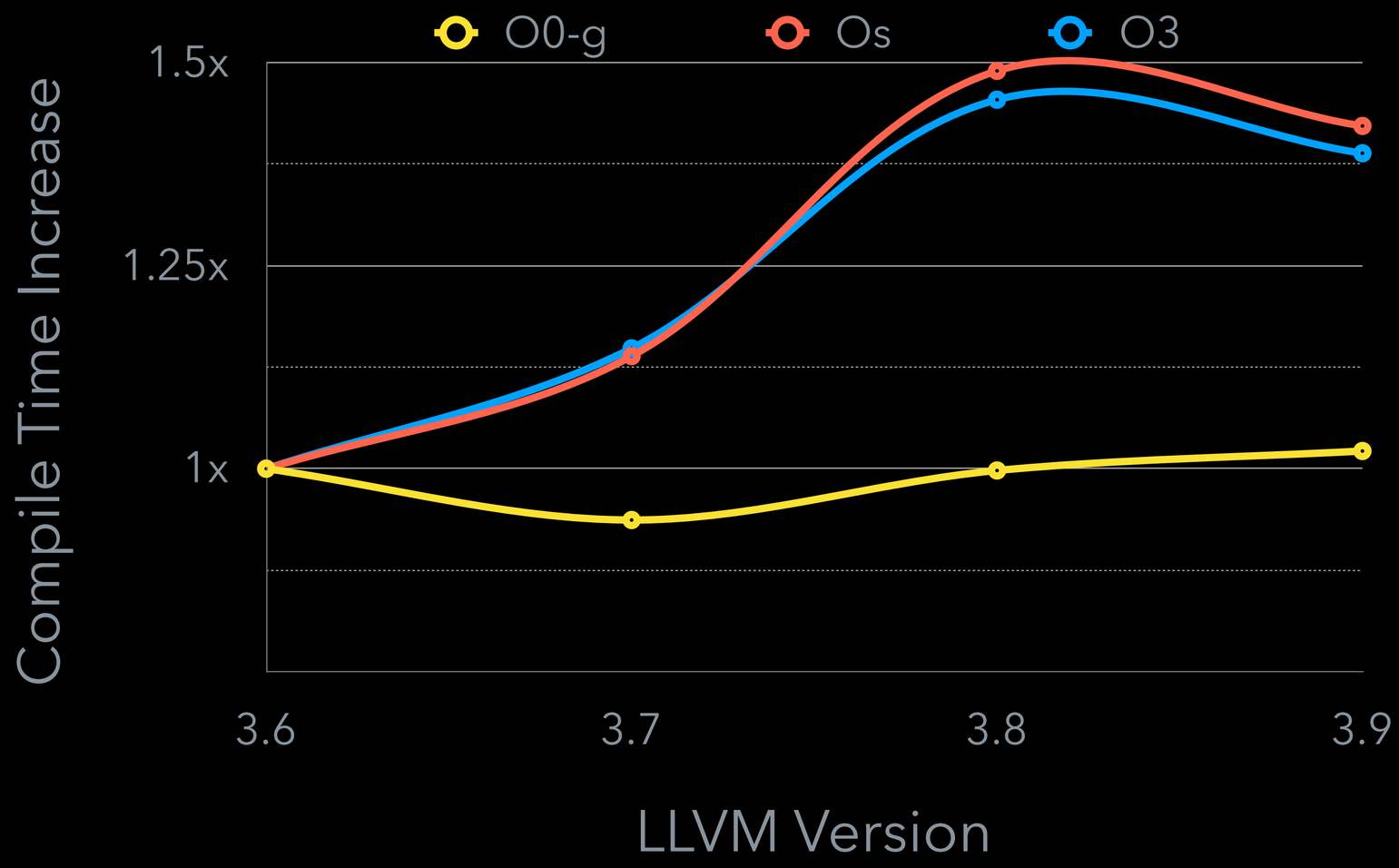
LLVM Compile Time.
Challenges. Improvements. Outlook.

Michael Zolotukhin, Apple

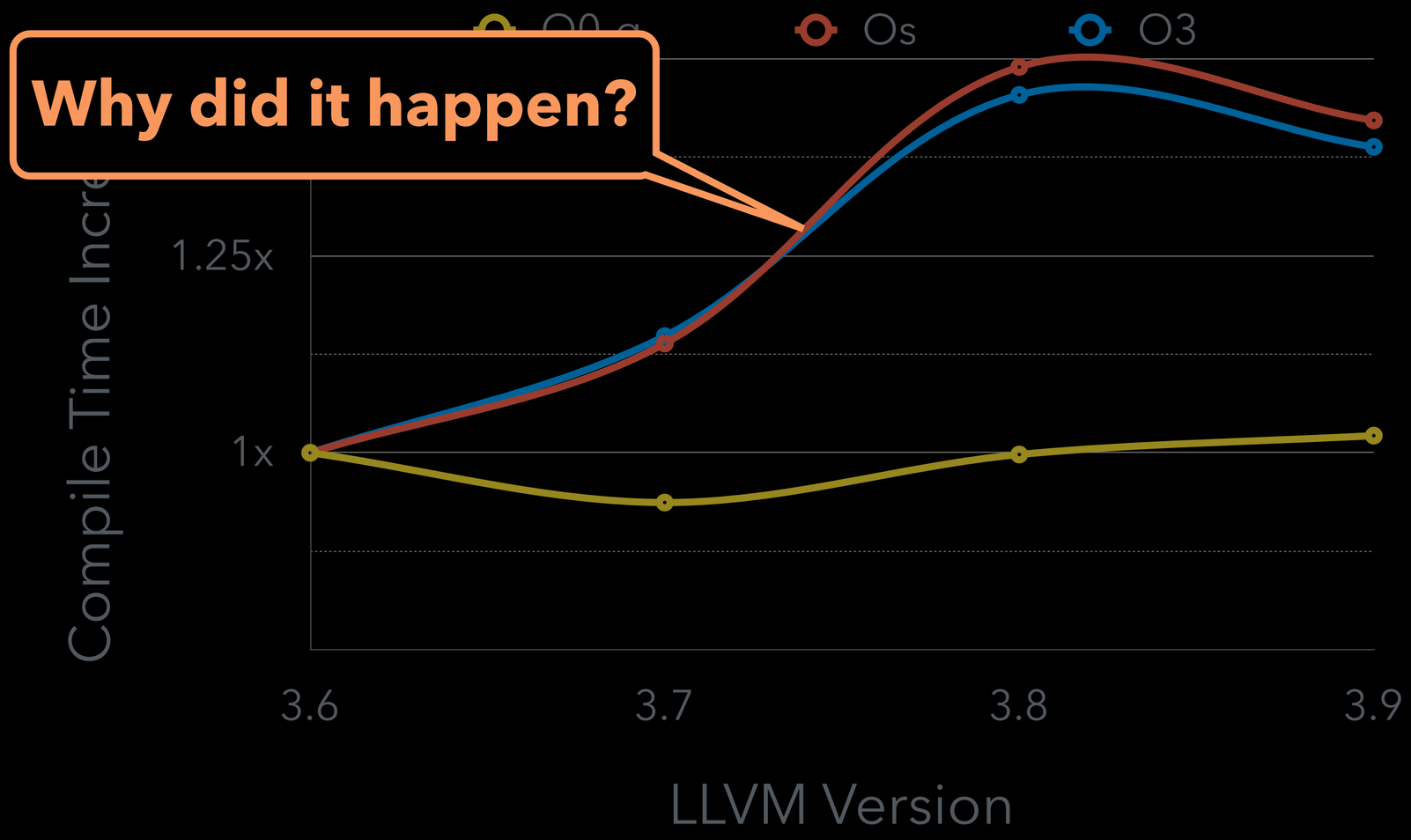
Agenda

- Benchmarking and tracking
- Historical findings
- Future work
- Tools and tricks

Compile Time Trend



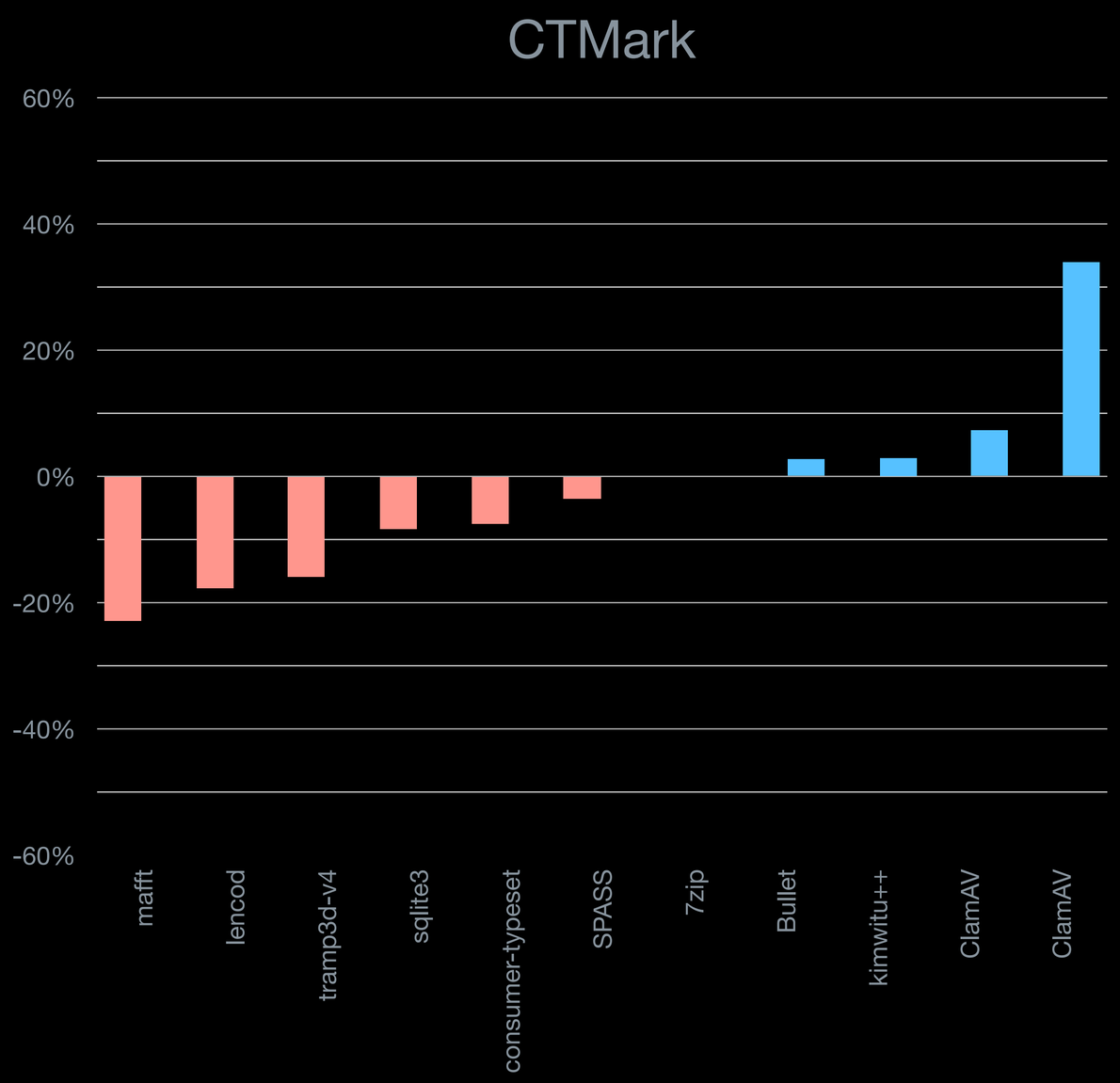
Compile Time Trend

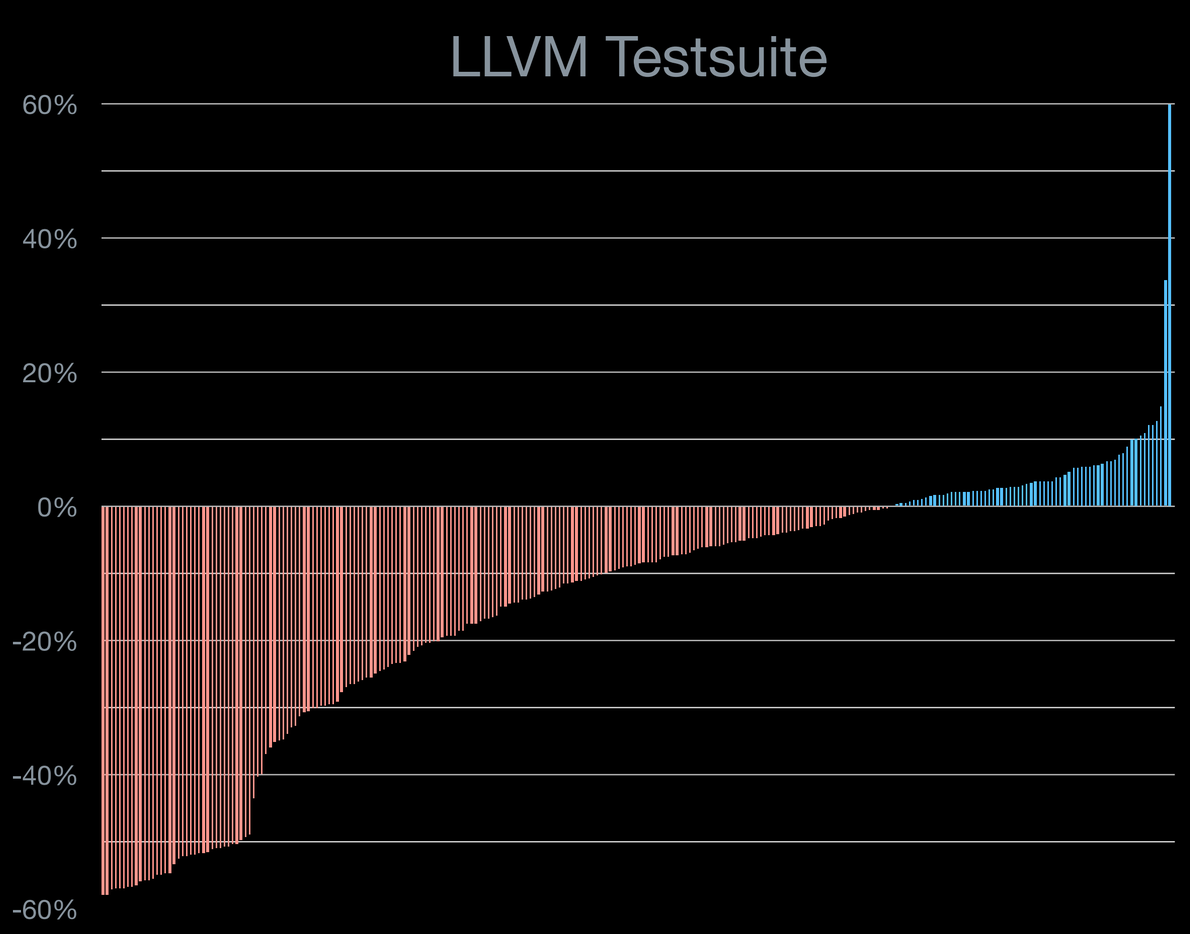


CTMark

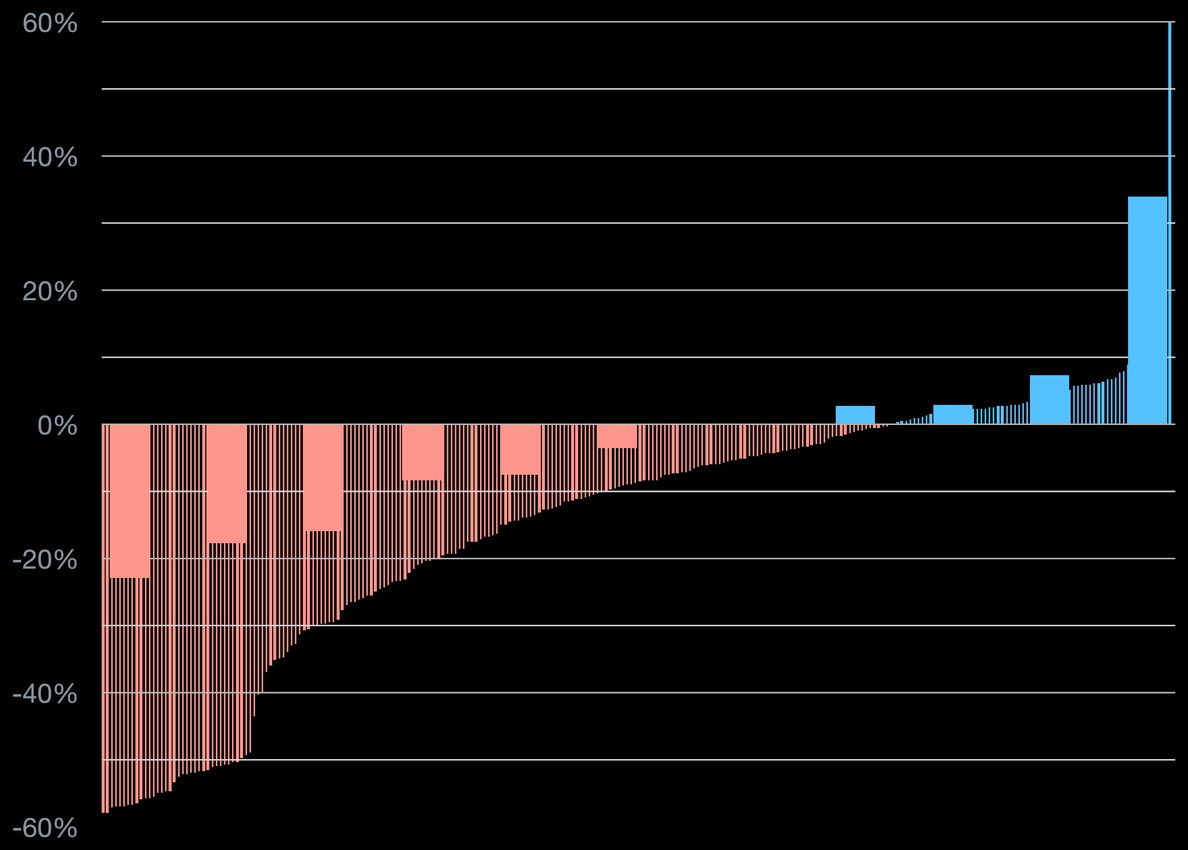
- Easy to use
- Reliable
- Fast
- Representative

CTMark


CTMark
60%
40%
20%
0%
-20%
-40%
-60%
mafft
lencod
tramp3d-v4
sqlite3
consumer-typeset
SPASS
7zip
Bullet
kimwitu++
ClamAV
ClamAV


LLVM Testsuite
60%
40%
20%
0%
-20%
-40%
-60%

CTMark

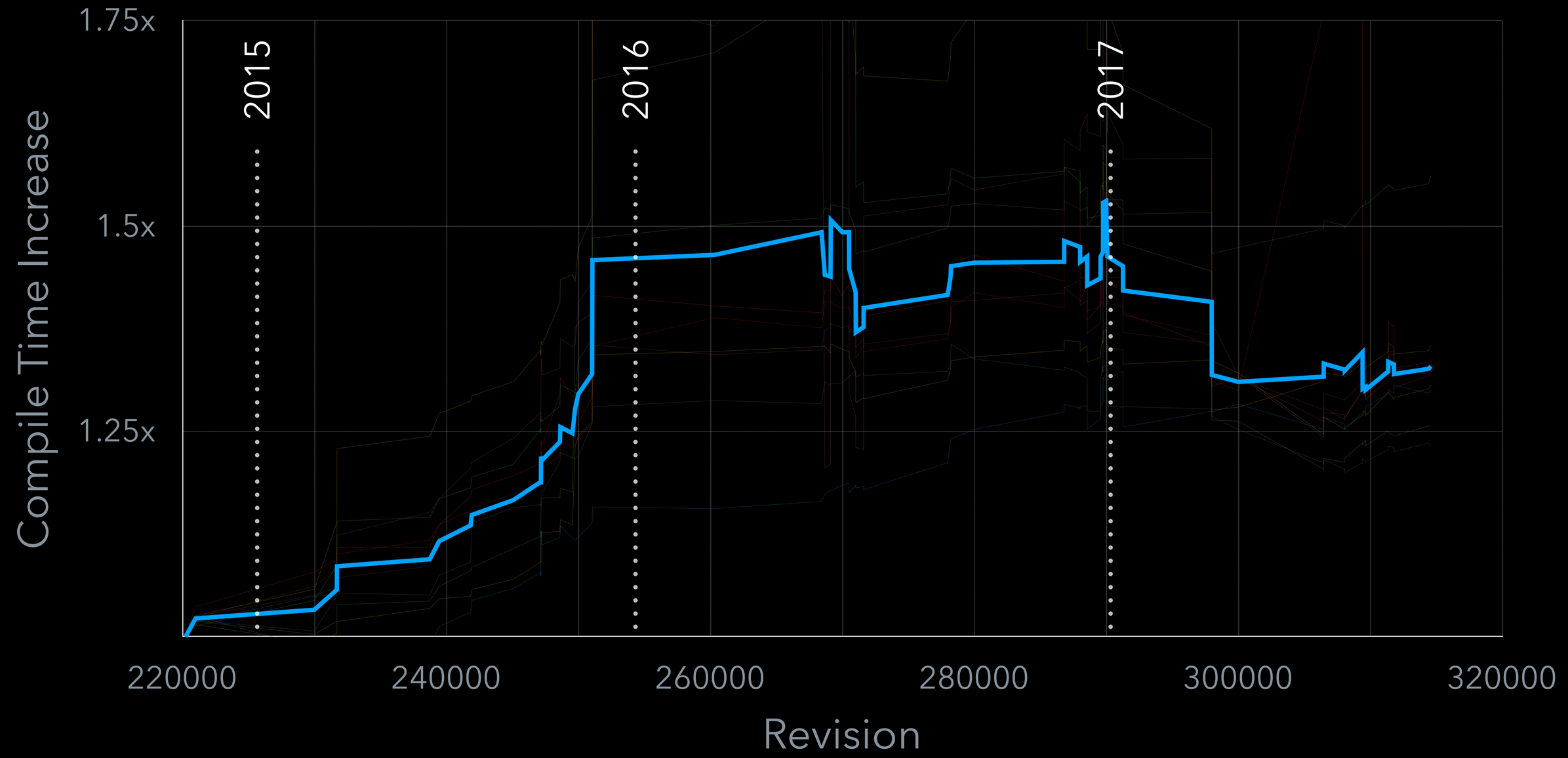


Regular Tracking

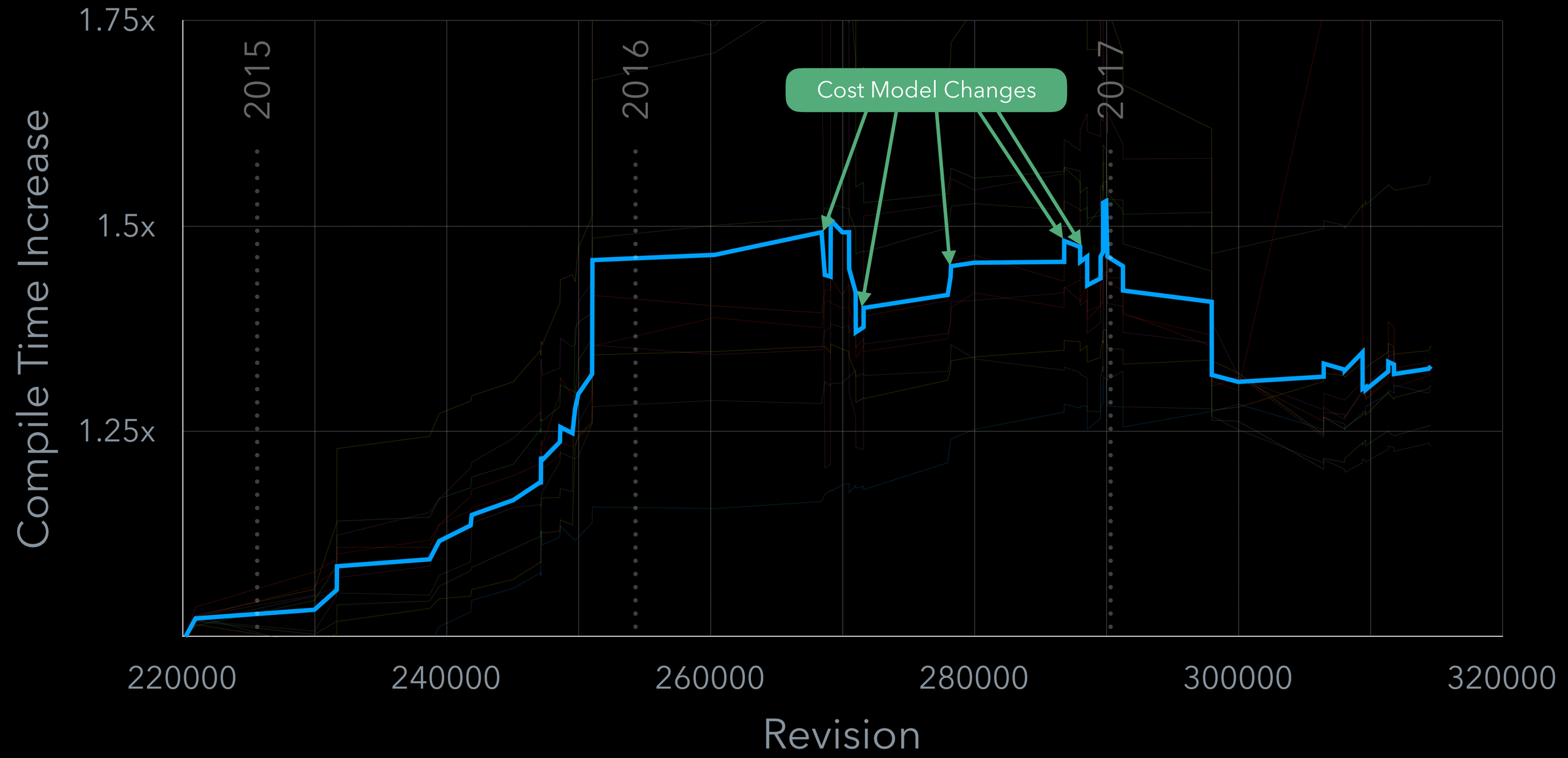
- Test every compiler build on several optlevels
- Detect and analyze incoming regressions
- Publish results on Green Dragon
- Raise awareness with reports

Historical Data

Historical Data



Historical Data



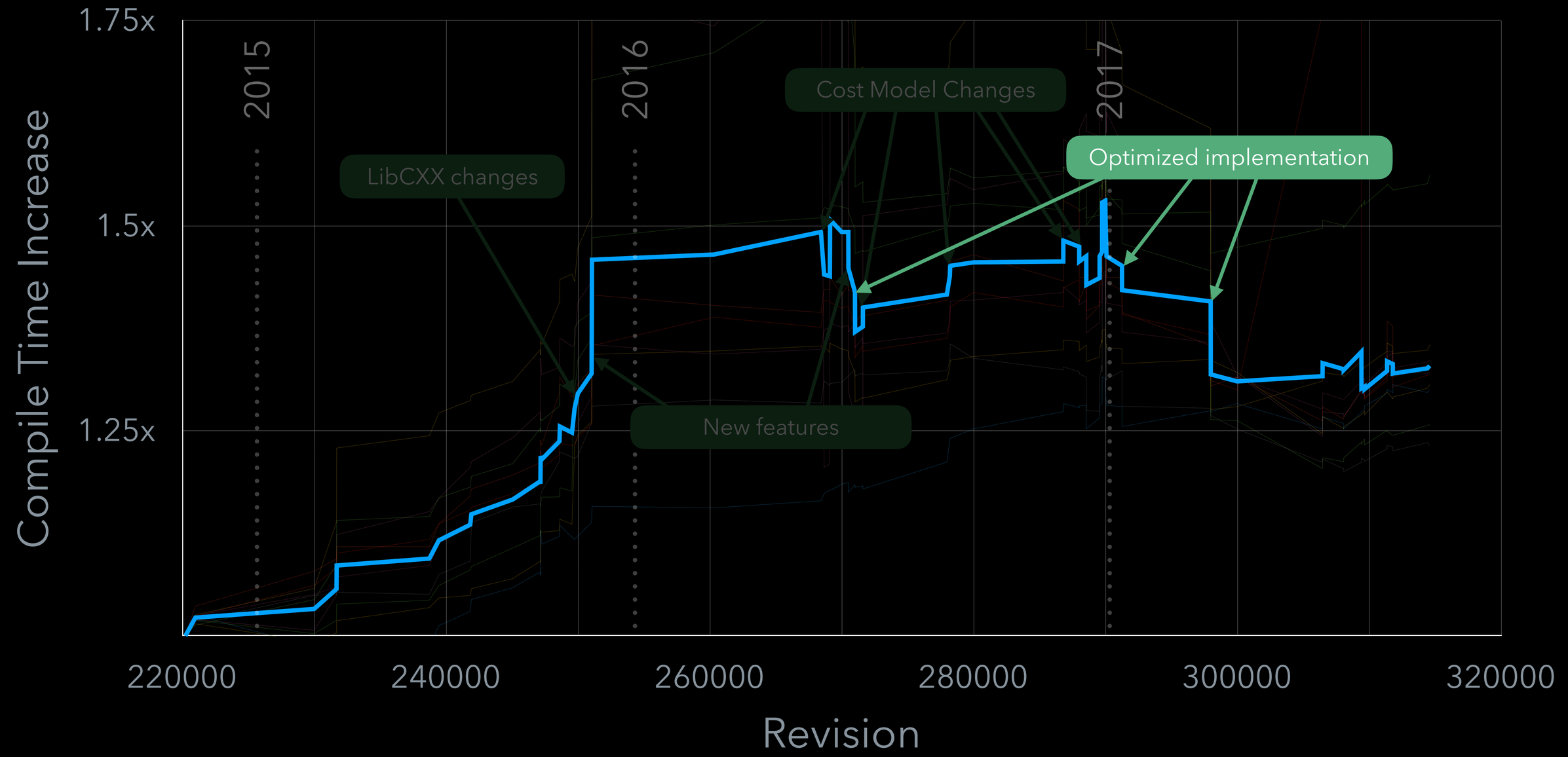
Historical Data

Historical Data

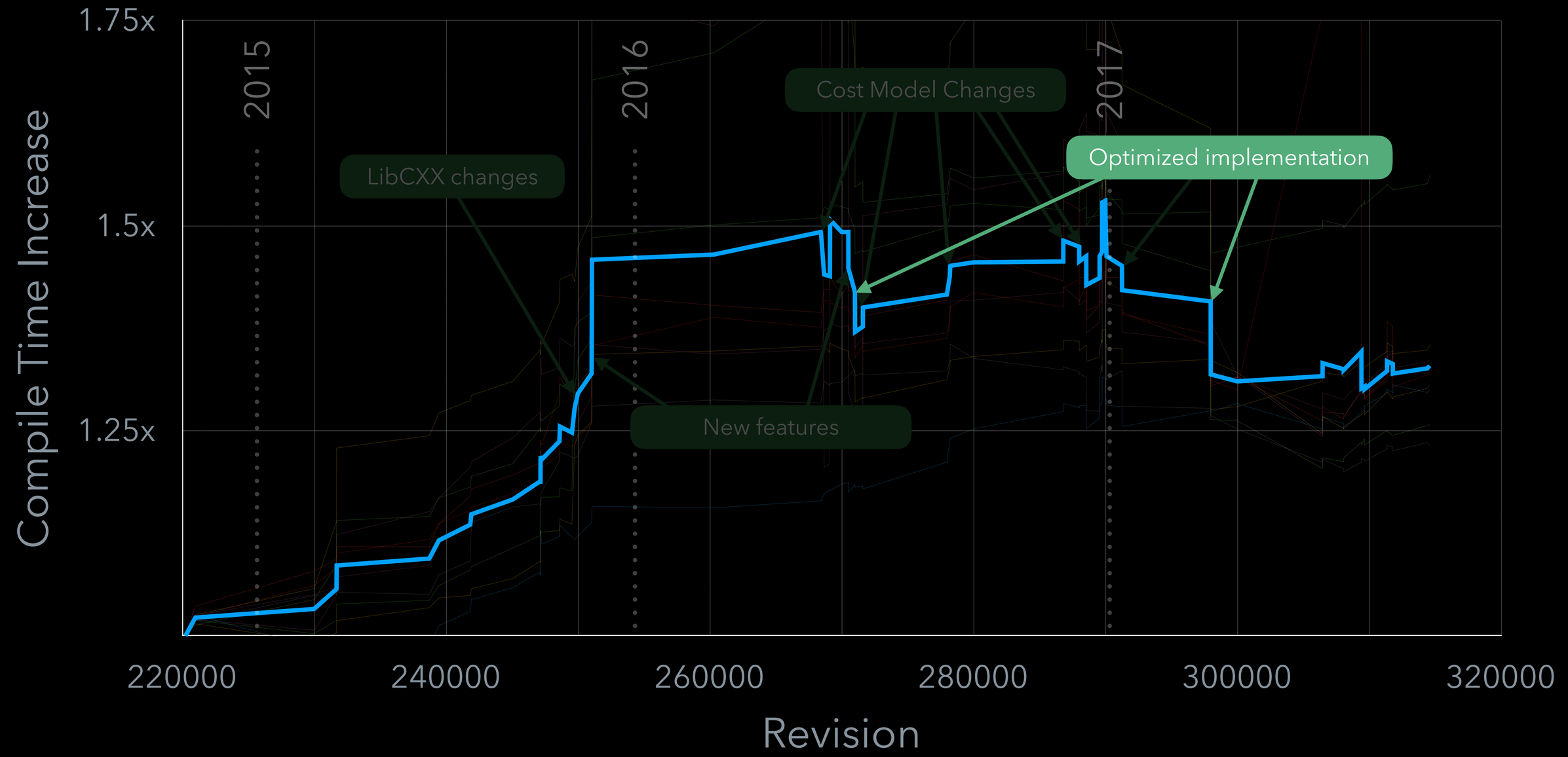
Historical Data

Historical Data

Historical Data



Historical Data

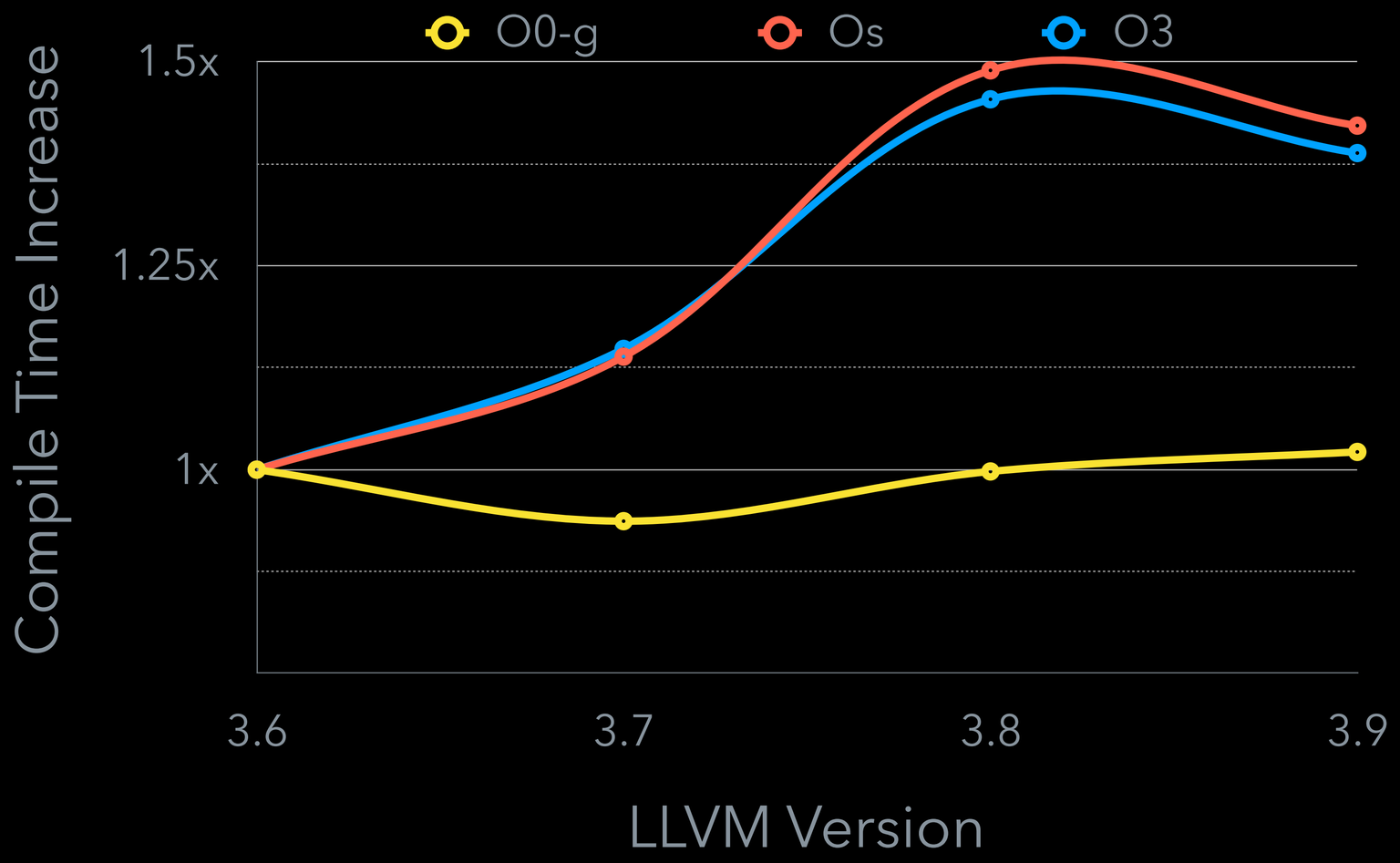


Historical Data

Historical Data

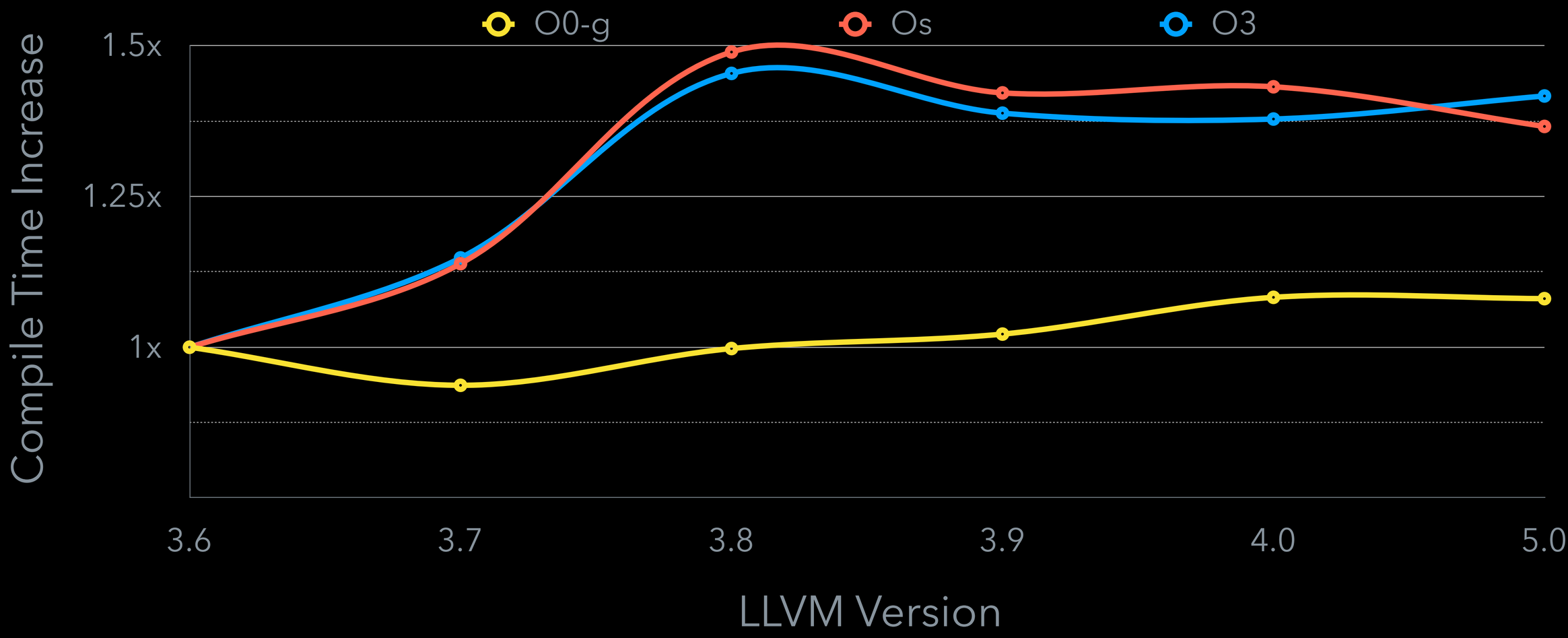
Historical Data

Compile Time Trend



Compile Time Trend

Compile Time Trend



Future Outlook

- Look at other components: front-end, libraries
- Find new ways to speed-up LLVM
- Add more benchmarks
- Speed-up your favorite pass

Tools and Tricks

Tools and Tricks

llvmlab

Tools and Tricks

llvmlab

Installation:

```
$ virtualenv venv && . venv/bin/activate
(venv)$ git clone http://llvm.org/git/zorg
(venv)$ pip install requests
(venv)$ python zorg/llvmbisect/setup.py install
(venv)$ which llvmlab
   /path/to/venv/bin/llvmlab
```

Tools and Tricks

llvmlab

Usage:

```
$ ### List available builders:
$ llvmlab ls
   clang-cmake-aarch64
   clang-cmake-armv7a
   clang-cmake-mips
   clang-cmake-mipsel
   clang-stage1-configure-RA
   clang-stage1-configure-RA_build
   clang-stage2-cmake-RgTSan
   clang-stage2-configure-Rlto
   clang-stage2-configure-Rlto_build
   clang-stage2-configure-Rthinlto_build
```

Tools and Tricks

llvmlab

Usage:

```
$ ### List available artifacts:
$ llvmlab ls clang-stage2-configure-Rlto
   clang-r314805-b21519
   clang-r314804-b21518
   clang-r314803-b21517
   clang-r314799-b21516
   clang-r314798-b21515
   clang-r314795-b21514
   clang-r314793-b21513
   ...
```

Tools and Tricks

llvmlab

Usage:

```
$ ### Download specified artifact:
$ llvmlab fetch clang-stage2-configure-Rlto clang-r314805-b21519
   downloaded root: clang-r314805-b21519.tar.gz
   extracted path : clang-r314805-b21519
$ clang-r314805-b21519/bin/clang -v
   Apple clang version 6.0.99 (master 314805) (based on LLVM 6.0.99)
   Target: x86_64-apple-darwin16.7.0
   Thread model: posix
   InstalledDir: /tmp/clang-r314805-b21519/bin
```

Tools and Tricks

Test-suite + LNT

Tools and Tricks

Test-suite + LNT

Installation:

```
$ virtualenv venv && . venv/bin/activate
(venv)$ git clone http://llvm.org/git/test-suite
(venv)$ git clone http://llvm.org/git/lnt
(venv)$ pip install -r lnt/requirements.client.txt
(venv)$ python lnt/setup.py install
(venv)$ pip install svn+http://llvm.org/svn/llvm-project/llvm/trunk/utils/lit
(venv)$ which lit
   /path/to/venv/bin/lit
(venv)$ which lnt
   /path/to/venv/bin/lnt
```

Tools and Tricks

Test-suite + LNT

Running CTMark:

```
$ lnt runtest test-suite --sandbox /Path/To/Sandbox                    \
                         --use-lit=lit                                 \
                         --cc  /Path/To/Compiler/bin/clang             \
                         --test-suite=/Path/To/test-suite              \
                         -C target-x86_64-macosx                       \
                         -C Os                                         \
                         --cmake-define TEST_SUITE_RUN_BENCHMARKS=Off  \
                         --cmake-define TEST_SUITE_SUBDIRS=CTMark
```

Tools and Tricks

Test-suite + LNT

Running full LLVM test-suite:

```
$ lnt runtest test-suite --sandbox /Path/To/Sandbox                    \
                         --use-lit=lit                                 \
                         --cc  /Path/To/Compiler/bin/clang             \
                         --test-suite=/Path/To/test-suite              \
                         -C target-x86_64-macosx                       \
                         -C Os                                         \
                         --cmake-define TEST_SUITE_BENCHMARKING_ONLY=On
```

Tools and Tricks

Test-suite + LNT

Running a subset of LLVM test-suite:

```
$ lnt runtest test-suite --sandbox /Path/To/Sandbox                    \
                         --use-lit=lit                                 \
                         --cc  /Path/To/Compiler/bin/clang             \
                         --test-suite=/Path/To/test-suite              \
                         -C target-x86_64-macosx                       \
                         -C Os                                         \
                         --cmake-define TEST_SUITE_BENCHMARKING_ONLY=On \
                         -only-test MultiSource/Applications
```

Tools and Tricks

Test-suite + LNT

Running a test-suite:

```
$ lnt runtest test-suite --sandbox /Path/To/Sandbox                    \
                         --use-lit=lit                                 \
                         --cc  /Path/To/Compiler/bin/clang             \
                         --test-suite=/Path/To/test-suite              \
                         -C target-x86_64-macosx                       \
                         -C Os                                         \
                         --cmake-define TEST_SUITE_BENCHMARKING_ONLY=On \
                         -only-test MultiSource/Applications

$ ### Results will be in /Path/To/Sandbox/test-DATETIME/output*.json
```

Tools and Tricks

Test-suite + LNT

Comparing results:

```
$ pip install pandas
$ /Path/To/test-suite/utils/compare.py -m compile_time output1.json output2.json
   Tests: 27
   Metric: compile_time

   Program                                 output1  output2 diff

   flops-3.test                            0.03     0.03   14.9%
   flops-2.test                            0.03     0.03   -13.7%
   himenobmtxpa.test                       0.10     0.12   11.6%
   ffbench.test                            0.06     0.07    9.3%
   …
$ ### The tool also has many useful options, see ‘--help’ for details
```

Tools and Tricks

Test-suite + cmake + lit

Tools and Tricks

Test-suite + cmake + lit

Building and running:

```
$ cd /Path/To/Sandbox

$ cmake   -DCMAKE_C_COMPILER=/Path/To/Compiler/bin/clang                  \
          -C /Path/To/test-suite/cmake/caches/target-x86_64-macosx.cmake \
          -C /Path/To/test-suite/cmake/caches/Os.cmake                   \
          -DTEST_SUITE_RUN_BENCHMARKS=Off                                \
          -DTEST_SUITE_SUBDIRS=CTMark                                    \
          /Path/To/test-suite
```

Tools and Tricks

Test-suite + cmake + lit

Building and running:

```
$ cd /Path/To/Sandbox

$ cmake   -DCMAKE_C_COMPILER=/Path/To/Compiler/bin/clang                \
          -C /Path/To/test-suite/cmake/caches/target-x86_64-macosx.cmake \
          -C /Path/To/test-suite/cmake/caches/Os.cmake                  \
          -DTEST_SUITE_RUN_BENCHMARKS=Off                               \
          -DTEST_SUITE_SUBDIRS=CTMark                                   \
          /Path/To/test-suite

$ ### Go to a subfolder if we don’t want to build all the tests:

$ cd CTMark/bullet

$ make -k -j 1 VERBOSE=1 all
```

Tools and Tricks

Test-suite + cmake + lit

Building and running:

```
$ cd /Path/To/Sandbox

$ cmake   -DCMAKE_C_COMPILER=/Path/To/Compiler/bin/clang                \
          -C /Path/To/test-suite/cmake/caches/target-x86_64-macosx.cmake \
          -C /Path/To/test-suite/cmake/caches/Os.cmake                  \
          -DTEST_SUITE_RUN_BENCHMARKS=Off                               \
          -DTEST_SUITE_SUBDIRS=CTMark                                   \
          /Path/To/test-suite

$ ### Go to a subfolder if we don’t want to build all the tests:

$ cd CTMark/bullet

$ make -k -j 1 VERBOSE=1 all

$ /Path/To/LLVM-Repo/utils/lit/lit.py -v -j 1 /Path/To/Sandbox/CTMark/bullet -o output.json
```

Tools and Tricks

Test-suite + cmake + lit

Per file stats:

```
$ find CTMark/Bullet -name “*.time"
   CTMark/Bullet/bullet.link.time
   CTMark/Bullet/CMakeFiles/bullet.dir/BenchmarkDemo.cpp.o.time
   …
   CTMark/Bullet/CMakeFiles/bullet.dir/SphereTriangleDetector.cpp.o.time
$ cat CTMark/Bullet/CMakeFiles/bullet.dir/SphereTriangleDetector.cpp.o.time
   exit 0
   real        0.1942
   user        0.1676
   sys         0.0172
$ size CTMark/Bullet/CMakeFiles/bullet.dir/SphereTriangleDetector.cpp.o
   __TEXT   __DATA   __OBJC   others   dec    hex
   3755     88       0        288      4131   1023
```

Tools and Tricks

Compiler Options

Tools and Tricks

Compiler Options

Collecting pass timings:

```
$ clang myfile.c -O3 -c -ftime-report
```

```
===-------------------------------------------------------------------------===
                         Miscellaneous Ungrouped Timers
===-------------------------------------------------------------------------===
  ---User Time---   --System Time--   --User+System--   ---Wall Time---  --- Name ---
   6.7877 ( 97.5%)   0.5060 ( 96.7%)   7.2938 ( 97.5%)   7.4313 ( 97.3%)  Code Generation Time
   0.1720 (  2.5%)   0.0172 (  3.3%)   0.1892 (  2.5%)   0.2075 (  2.7%)  LLVM IR Generation Time
   6.9598 (100.0%)   0.5232 (100.0%)   7.4830 (100.0%)   7.6388 (100.0%)  Total

===-------------------------------------------------------------------------===
                              Register Allocation
===-------------------------------------------------------------------------===
  Total Execution Time: 0.2353 seconds (0.2364 wall clock)
   ---User Time---   --System Time--   --User+System--   ---Wall Time---  --- Name ---
   0.1907 ( 83.5%)   0.0024 ( 34.6%)   0.1931 ( 82.1%)   0.1940 ( 82.0%)  Global Splitting
   0.0213 (  9.3%)   0.0010 ( 13.7%)   0.0222 (  9.4%)   0.0222 (  9.4%)  Spiller
    ...
```

Tools and Tricks

Compiler Options

Collecting pass timings:

```
$ clang myfile.c -O3 -c -ftime-report -save-stats=obj
$ cat myfile.stats
   {
          "time.regalloc.local_split.wall": 1.283169e-03,
          "time.regalloc.local_split.user": 1.176000e-03,
          "time.regalloc.local_split.sys": 9.700000e-05,
          "time.regalloc.spill.wall": 2.174950e-02,
          "time.regalloc.spill.user": 2.083200e-02,
          "time.regalloc.spill.sys": 8.420000e-04,
           ...
   }
```

Tools and Tricks

Compiler Options

Collecting pass timings:

```
$ clang myfile.c -O3 -c -disable-llvm-passes -emit-llvm
$ opt -O3 myfile.bc -o myfile.o -time-passes
```

```
   ===-------------------------------------------------------------------===
                         ... Pass execution timing report ...
   ===-------------------------------------------------------------------===
     Total Execution Time: 5.9781 seconds (6.0340 wall clock)
      ---User Time---   --System Time--   --User+System--   ---Wall Time---  --- Name ---
      0.5257 (  9.2%)   0.0044 (  1.7%)   0.5302 (  8.9%)   0.5351 (  8.9%)  Global Value Numbering
      0.3432 (  6.0%)   0.0035 (  1.3%)   0.3467 (  5.8%)   0.3494 (  5.8%)  Function Integration/Inlining
      0.2423 (  4.2%)   0.0019 (  0.7%)   0.2443 (  4.1%)   0.2458 (  4.1%)  Combine redundant instructions
       ...
```

Tools and Tricks

Compiler Options

Collecting optimization remarks:

```
$ clang myfile.c -O3 -c -fsave-optimization-record
$ cat myfile.opt.yaml
```

```
--- !Passed
Pass:            loop-unroll
Name:            FullyUnrolled
DebugLoc:        { File: myfile.c, Line: 3, Column: 3 }
Function:        foo
Args:
  - String:          'completely unrolled loop with '
  - UnrollCount:     '4'
  - String:          ' iterations'
...
```

Tools and Tricks

Compiler Options

Inspecting optimization remarks:

```
$ opt-viewer.py myfile.opt.yaml
$ open html/myfile.c.html
```

Line	Hotness	Optimization	Source	Inline Context
1			`int foo(int *a) {`	
		prologepilog	8 stack bytes in function	foo
		asm-printer	8 instructions in function	foo
2			`  int r = 0;`	
3			`  for (int i = 0; i < 4; i++)`	
		loop-unroll	completely unrolled loop with 4 iterations	foo
4			`    r += a[i];`	
5			`  return r;`	
6			`}`	

Tools and Tricks

Compiler Options

Inspecting optimization remarks:

```
$ opt-viewer.py myfile.opt.yaml
$ open html/myfile.c.html
```

Line	Hotness	Optimization	Source	Inline Context
1			`int foo(int *a) {`	
		prologepilog	8 stack bytes in function	foo
		asm-printer	8 instructions in function	foo
2			`  int r = 0;`	
3			`  for (int i = 0; i < 4; i++)`	
		loop-unroll	completely unrolled loop with 4 iterations	foo
4			`    r += a[i];`	
5			`  return r;`	
6			`}`	

```
$ opt-diff.py myfile1.opt.yaml myfile2.opt.yaml
$ opt-viewer.py diff.opt.yaml
```

Tools and Tricks

Compiler Options

Inspecting optimization remarks:

```
$ opt-viewer.py myfile.opt.yaml
$ open html/myfile.c.html
```

Line	Hotness	Optimization	Source	Inline Context
1			`int foo(int *a) {`	
		prologepilog	8 stack bytes in function	foo
		asm-printer	8 instructions in function	foo
2			`  int r = 0;`	
3			`  for (int i = 0; i < 4; i++)`	
		loop-unroll	completely unrolled loop with 4 iterations	foo
4			`    r += a[i];`	
5			`  return r;`	
6			`}`	

```
$ opt-diff.py myfile1.opt.yaml myfile2.opt.yaml
$ opt-viewer.py diff.opt.yaml
```

More details: LLVM Dev 2016 Talk: Compiler-assisted Performance Analysis

Summary

- Test your patches for compile time
- Use CTMark
- Know your tools
- Make LLVM faster!