Incremental Compilers with Internal Build Systems

Jeff Smits, Gabriël Konat, Eelco Visser

Constructing Hybrid Incremental Compilers for Cross-Module Extensibility with an Internal Build System. <Programming>'20.

Incremental Compiler Problem

Initial issue Compiler is slow for large projects

Goal Compilation time proportional to the size of the change

Problem Influenced by language features

Problem Existing compiler was not designed to be incremental

Problem Expense of building new incremental compiler

Problem Writing your own incremental system is difficult and error-prone

<u>Our paper</u> Rework existing compiler to be incremental anyway

- Internal use of incremental build system
- Make cross-module language features incrementally compilable
- Demonstrated on critical case: Stratego

Stratego

Rewriting with programmable strategies [ICFP'98]

- Generic traversals
- Dynamically typed
- Used in practice
  - Stratego/XT, Spoofax Language Workbench
  - At Oracle Labs, Canon
  - Indirectly for researchr conference websites

Stratego

Rewriting with programmable strategies [ICFP'98]

- Generic traversals
- ~~Dynamically~~ Gradually typed
- Used in practice
  - Stratego/XT, Spoofax Language Workbench
  - At Oracle Labs, Canon
  - Indirectly for researchr conference websites

Cross-Module Extensibility

Statements
Expressions
Function calls
Void type

```
module desugar/core

strategies

desugar-all = innermost(Desugar)

Desugar = fail
```

Integer literals
Integer operations
Integer type

```
module desugar/int
imports desugar/core

strategies

Desugar = BinOpToCall

is-bin-op = ?"Add" <+ ?"Mul" // etc.

rules

BinOpToCall :
  f#([e1, e2]) → |[ f(e1, e2) ]|
  where <is-bin-op> f
```

If-then
If-then-else
While
For

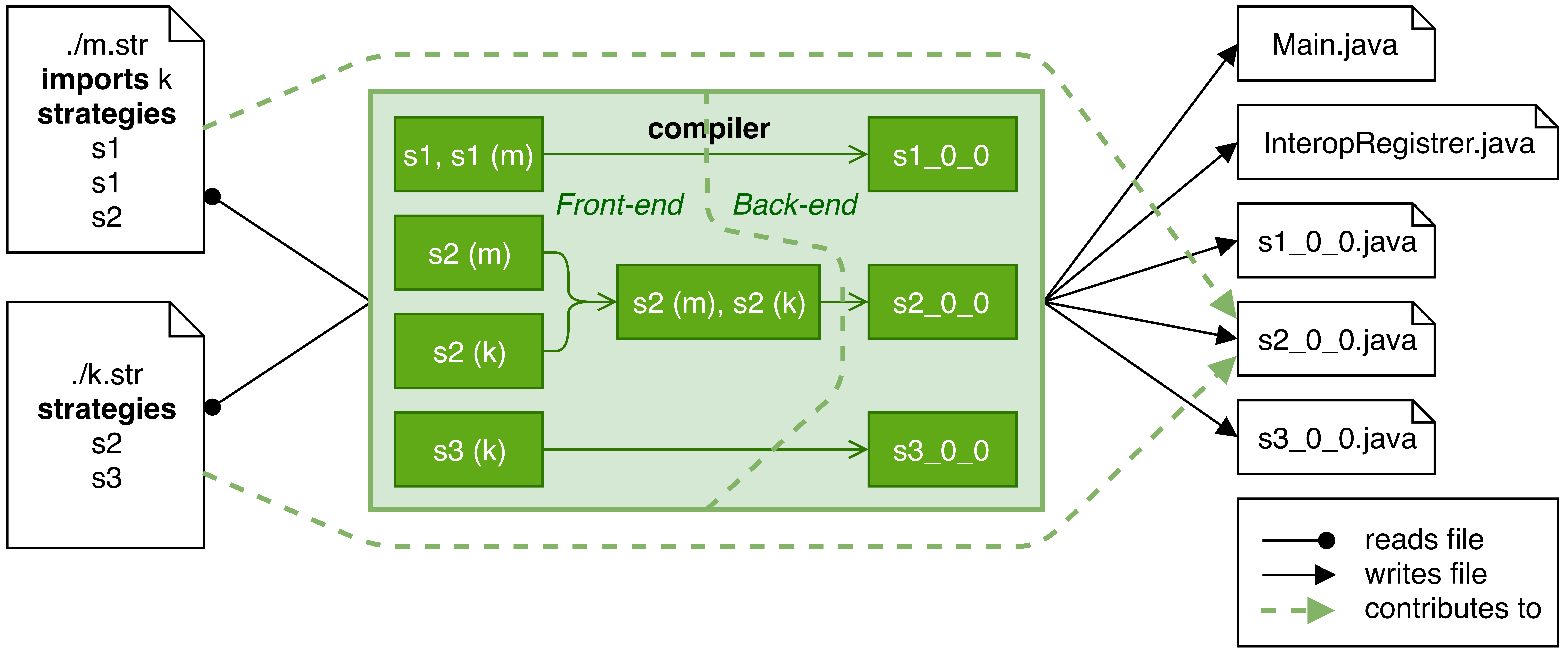
```
module desugar/control
imports desugar/core

strategies

Desugar =
  ForToWhile <+ IfThenToIfElse

rules

ForToWhile :
  |[ for x := e1 to e2 do st* end ]| →
  |[ begin
    var x : int; var y : int;
    x := e1; y := e2;
    while x ⩽ y do
    st* x := x + 1; end
    end ]|
  where new ⇒ y

IfThenToIfElse :
  |[ if e then st* end ]| →
  |[ if e then st* else end ]|
```

Stratego Compilation

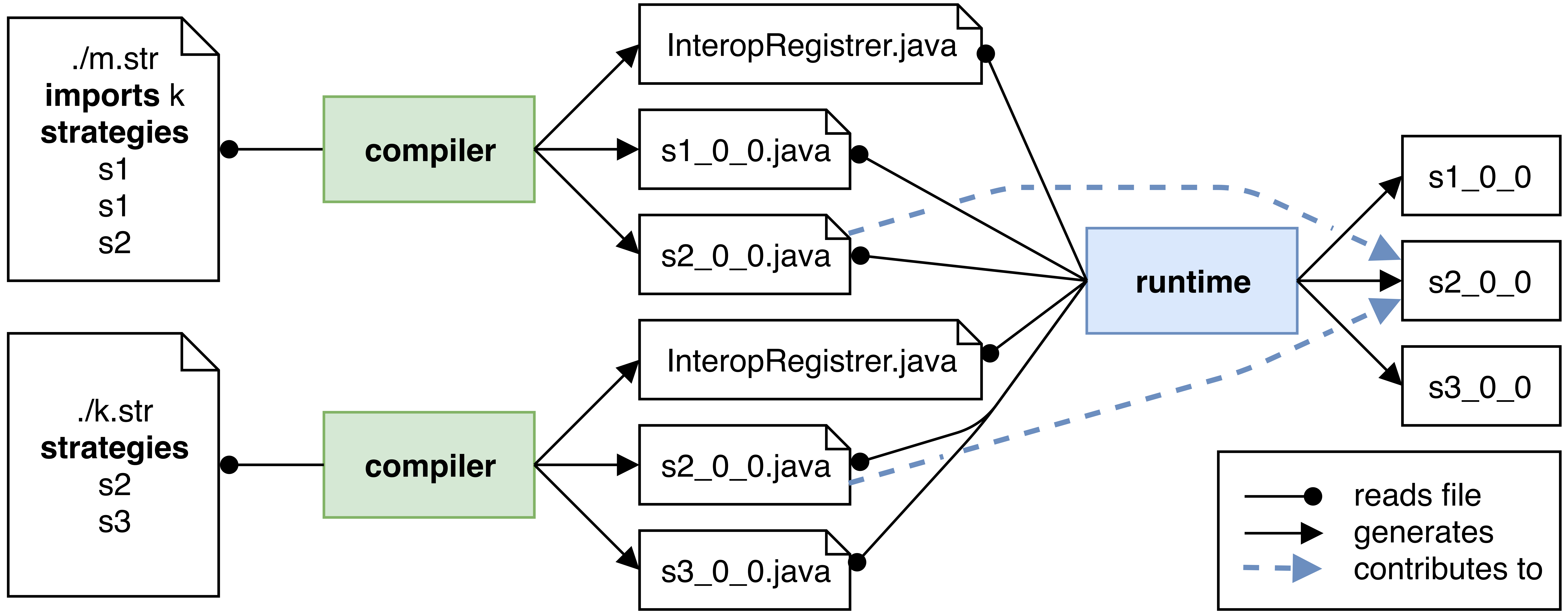
The (main) problem with incrementally compiling Stratego:

- There can be *multiple* definitions of rules and strategies with the same name
- These can exist in *different modules*
- Definitions with the same name are merged into one

Existing Stratego Compiler



Dynamic Linking



Static Linking (with Gradual Types)

./m.str
imports k
strategies
s1
s1
s2
./k.str
strategies
s2
s3
Front
Front
imports: [k],
definitions: [s1, s2]
imports: [],
definitions: [s2, s3]
Resolve
ast,
imports: [k],
definitions: [s1, s2],
references: [s3]
CheckModule
CheckModule
ast,
imports: []
definitions: [s2, s3],
references
referenced defs: [s3]
moduleList: [m, k]
success,
messages
messages
messages
Check
messages
success,
messages,
files written
Compile
modules defining s2: [m,k]
modules defining s1: [m]
modules defining s3: [k]
ast of s1
ast of s2
ast of s2
ast of s3
definitions: [s1,s2,s3]
definitions: [s1,s2,s3]
Back
Back
Back
Back
s1_0_0.java
s2_0_0.java
s3_0_0.java
Main.java
InteropRegistrer.java
reads
generates
requires task

Scalable Incremental Building with Dynamic Task Dependencies
Gabriël Konat, Sebastian Erdweg, and Eelco Visser. ASE'18.

Benchmark

The WebDSL compiler

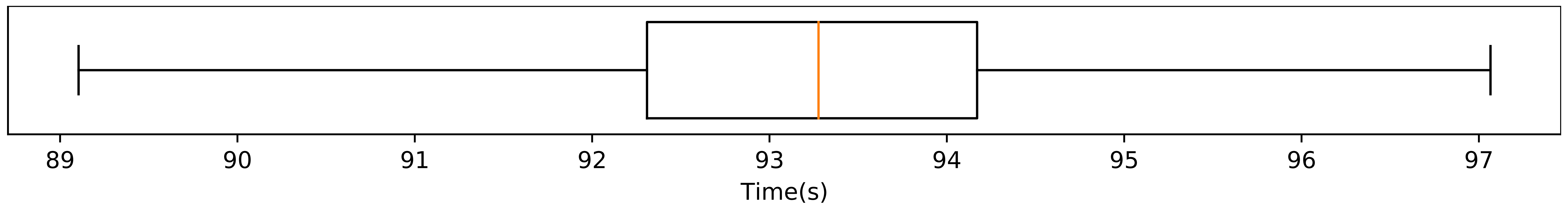
>27,000 LOC excluding whitespace and comments

~400 files

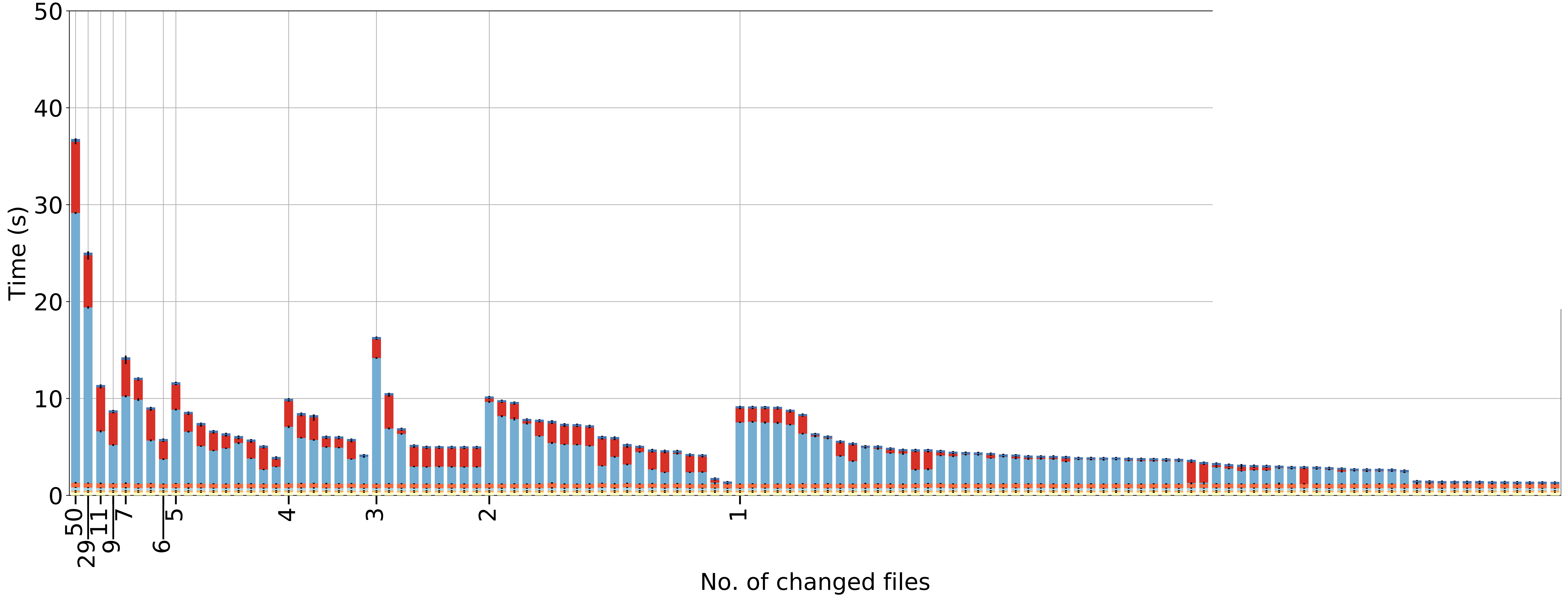
>10,000 distinct named strategies

The latest 200 commits of Git history

Original Compiler Performance



Incremental Compiler Performance



Clean build: 168.179 seconds (1.8x slower)

Conclusion

An incremental compiler for a critical case

- Reused most of the original whole-program compiler
- Backward compatible compiler output
- Created separate processing tasks out of compiler pieces
- Using an incremental build system *internally* to wire these tasks together