Lecture 14: Compiler Correctness

John Wickerson

This lecture

- How can we ensure that a compiler is **correct**?
- Compiler correctness is important because compilers are **prevalent** and **trusted**.

Coursework assessment

- List of language features to support is on Github.
- Compilers will be assessed solely on whether the generated assembly produces the correct output.
- Compilers will only be tested on valid inputs.
- I don't expect your compiler to work every input I've given you; this list is meant as an exhaustive upper bound for everyone!

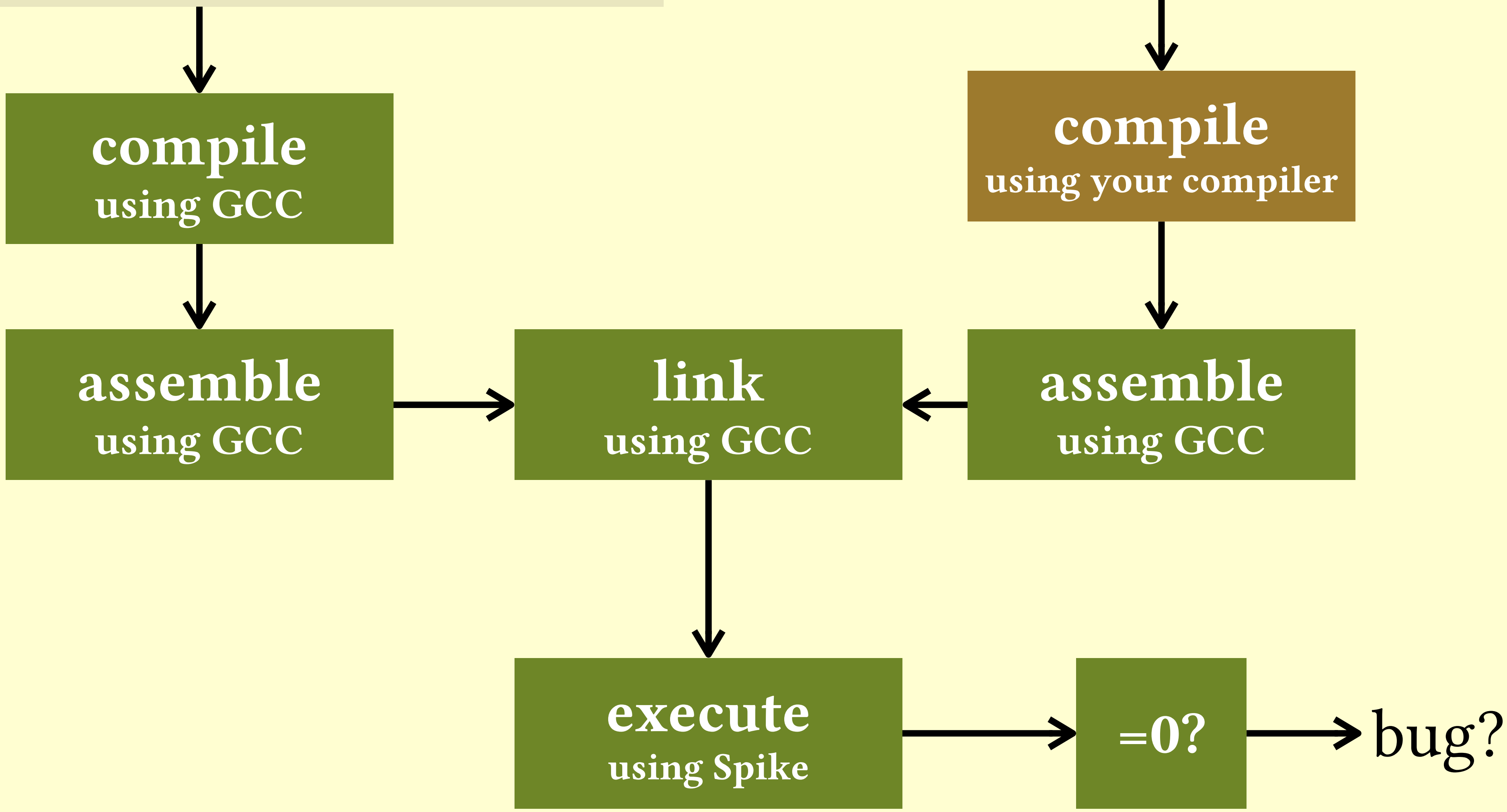
Coursework assessment

test_ADD0_driver.c

```
int f(int x, int y);
int main() {
  return !(40==f(30,10));
}
```

test_ADD0.c

```
int f(int a, int b) {
  return a+b;
}
```

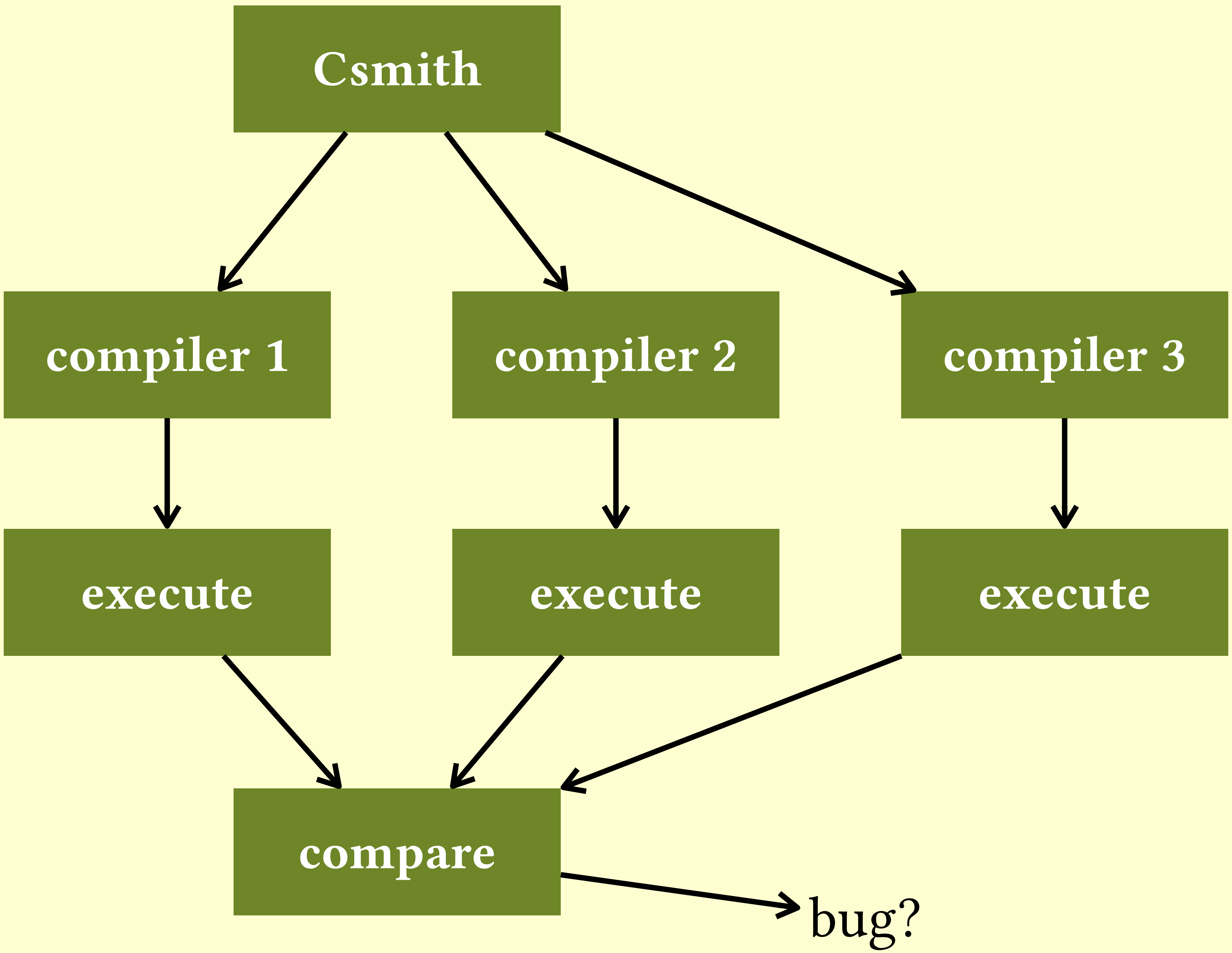


Random testing

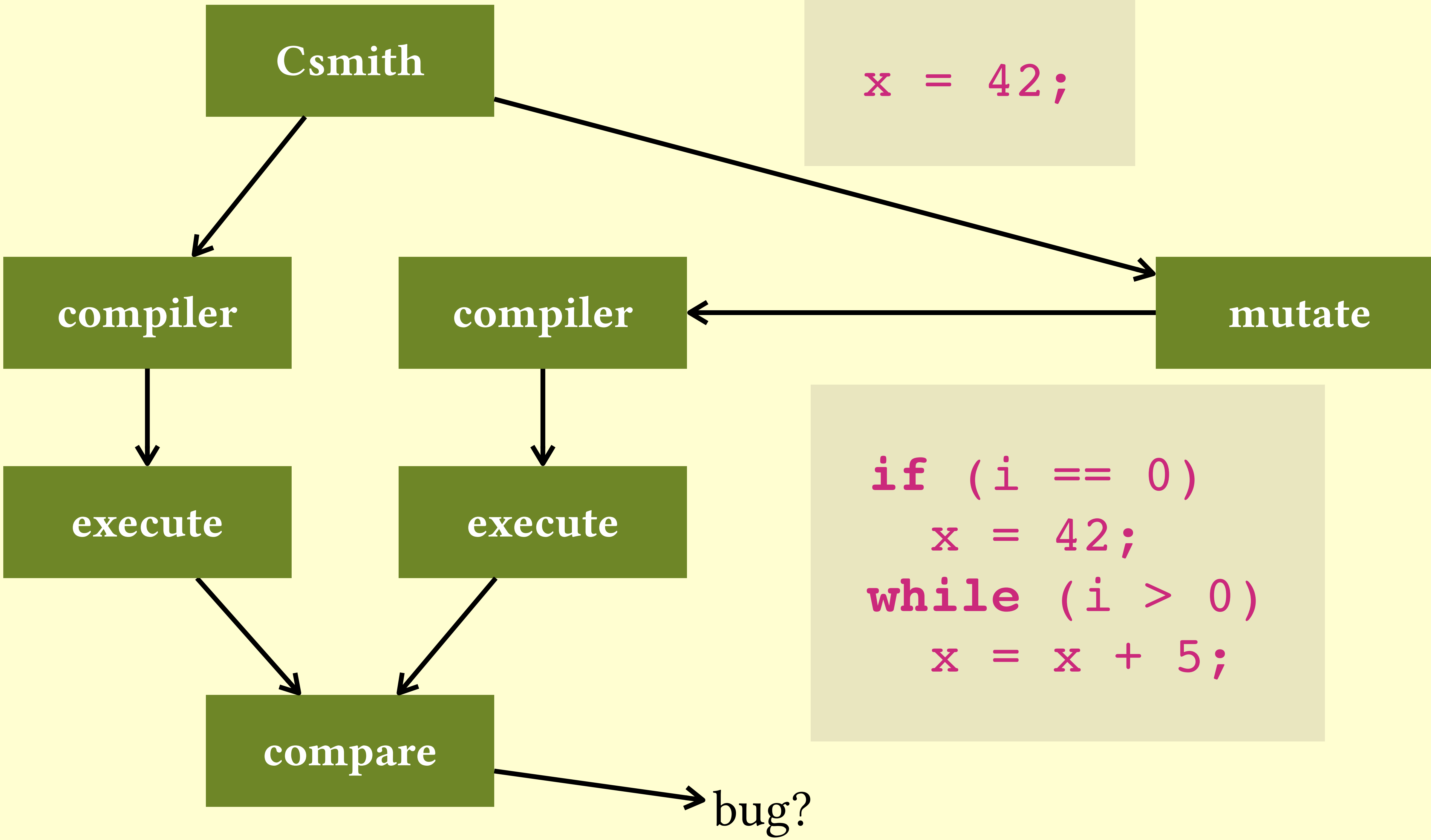
- **Csmith** is a tool from the University of Utah that generates random C programs.

- **Question.** How do we know if the random program has been compiled correctly?
- **Solution 1.** Random differential testing.
- **Solution 2.** EMI testing.

Random differential testing



EMI testing




```
x = 42;
```

```
if (i == 0)
  x = 42;
while (i > 0)
  x = x + 5;
```

Problem: non-determinism

- C99 has 52 **unspecified** behaviours (e.g. the order in which operands are evaluated) and 114 **implementation-defined** behaviours (e.g. the size of an `int`). Generated programs must produce output that does not depend on how a compiler implements these behaviours.

- C99 also has 191 **undefined** behaviours (e.g. `x = *NULL`, or overflowing a `signed int`). Generated programs must not contain undefined behaviours at all.

```
foo(*p);
```

✗

```
if (p!=NULL)
  foo(*p);
```

✓

Features of Csmith

- **Included:** global variables, local variables, if-then-else statements, for loops, break/continue statements, goto statements, function calls, signed and unsigned integers of various sizes, arithmetic and logical operations, structs, arrays, pointers.

- **Excluded:** strings, malloc, floating point types, unions, recursion, function pointers.

Example bug

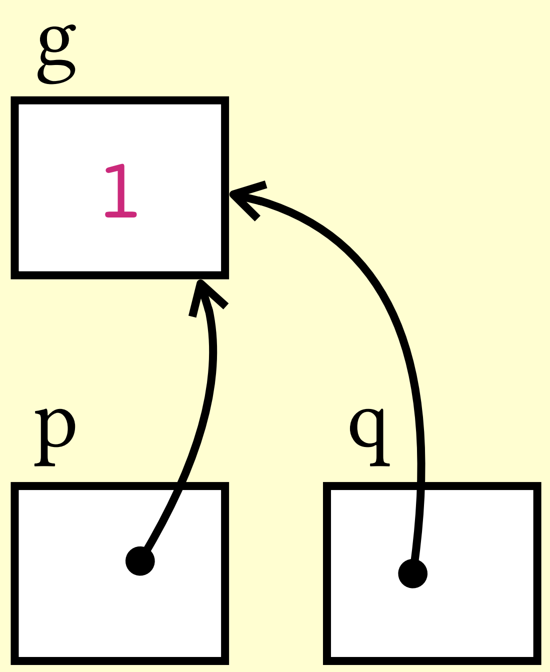
```
int foo () {
  signed char x = 1;
  unsigned char y = 255;
  return x > y;
}
```

- When operands have different types, both are promoted to `int`. So this function should return `0`.

- However, when compiled with an old version of GCC, this function returns `1`.

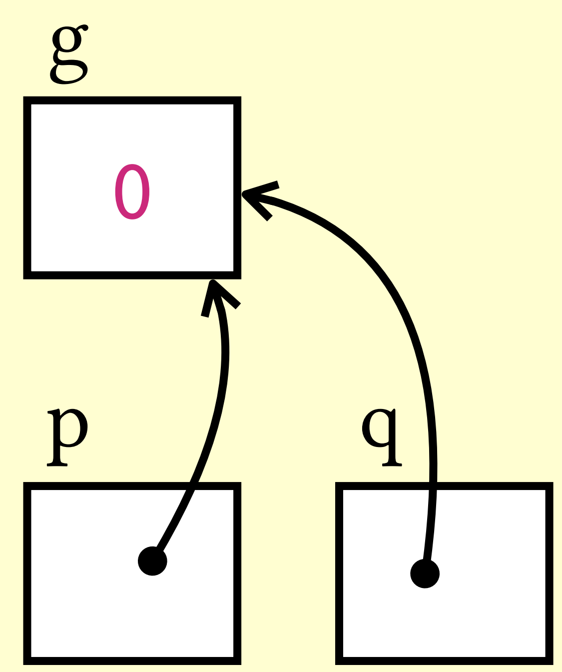
Another example bug

```
int g[1];
int *p = &g[0];
int *q = &g[0];

int foo () {
  g[0] = 1;
  *p = 0;
  *p = *q;
  return g[0];
}
```



Another example bug

```
int g[1];
int *p = &g[0];
int *q = &g[0];

int foo () {
  g[0] = 1;
  *p = 0;
  *p = *q;
  return g[0];
}
```

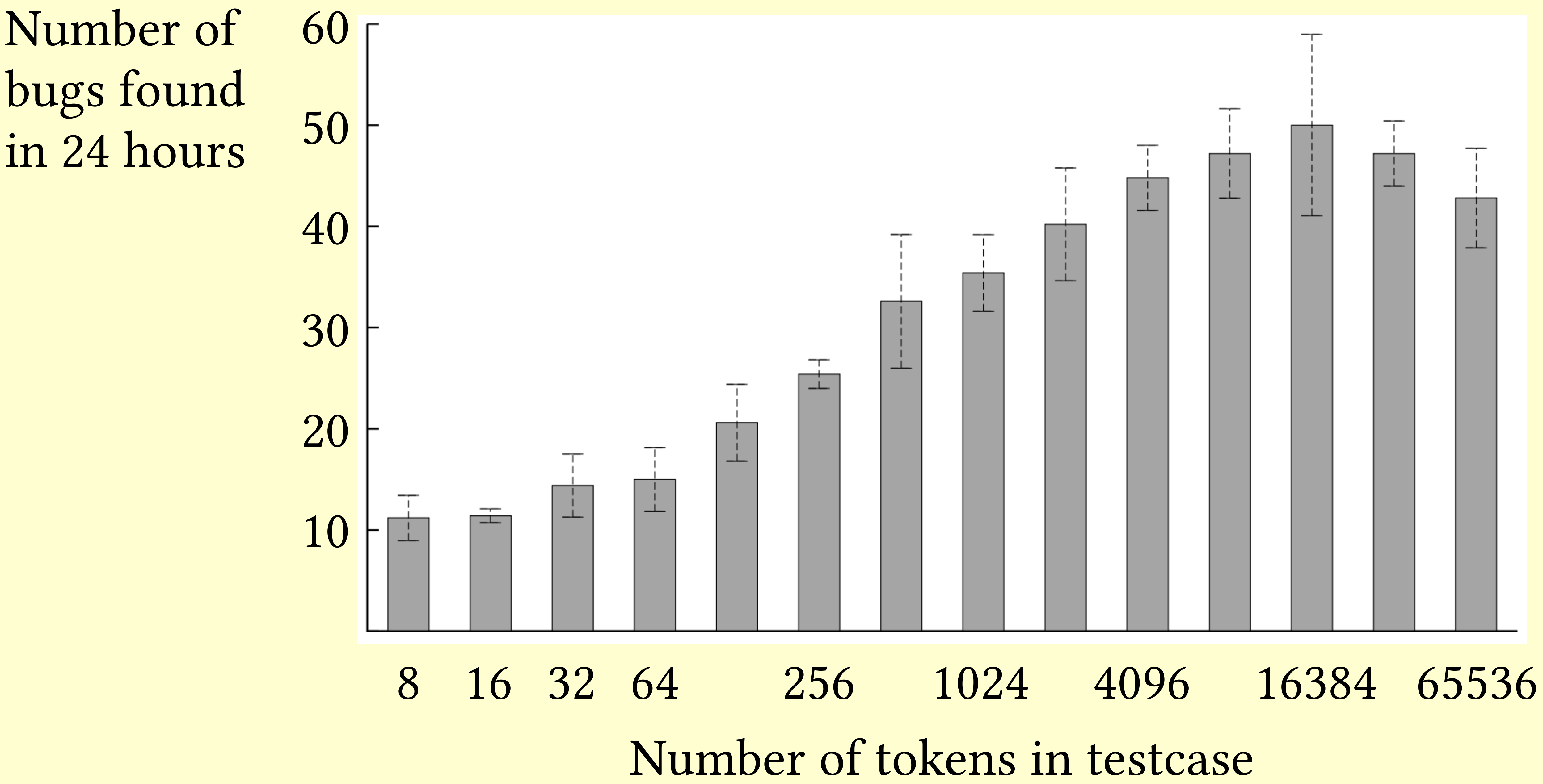


- Should return `0`, but an old version of GCC did faulty dead-code elimination (removing `*p = 0`) because it didn't realise that `p` and `q` are aliases, and thus returned `1`.

Bugs found

	GCC	LLVM COMPILER INFRASTRUCTURE
crashes	50	136
wrong code bugs	29	66

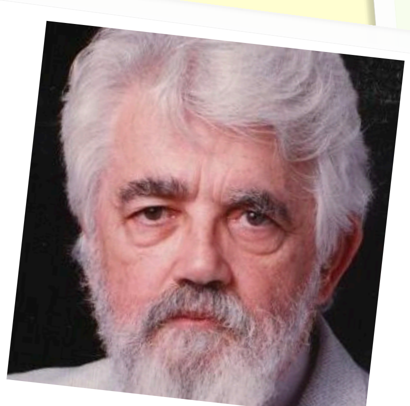
How large should the random programs be?



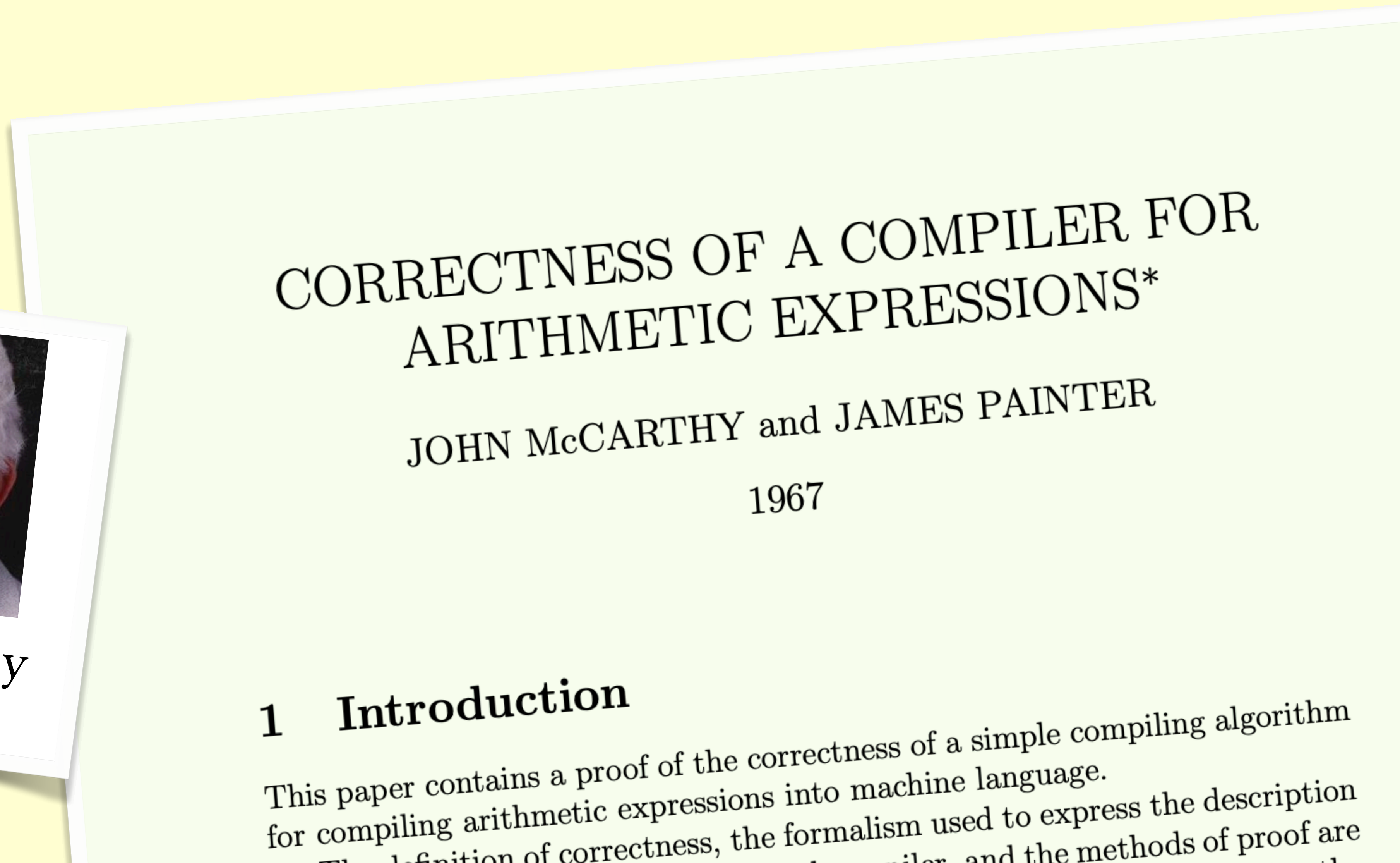
Can we do better?

Verified compilation

- Can we prove (mathematically) that our compiler will never generate wrong code?



John McCarthy
1927–2011



CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

1967

1 Introduction

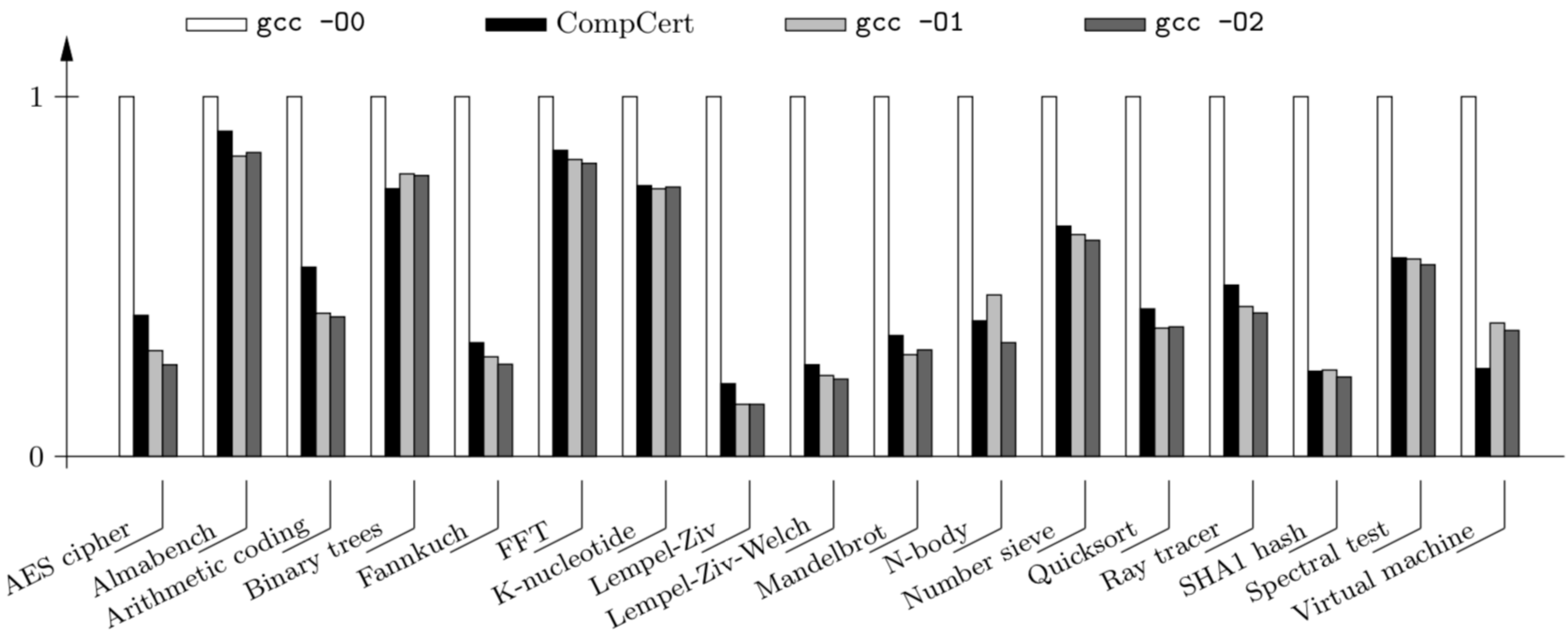
This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

CompCert

- A verified C compiler
- Development started around 2005;
  first release was in 2008.
- Programmed (and proved correct) in Coq.
- Performance is comparable to `gcc -O1`.

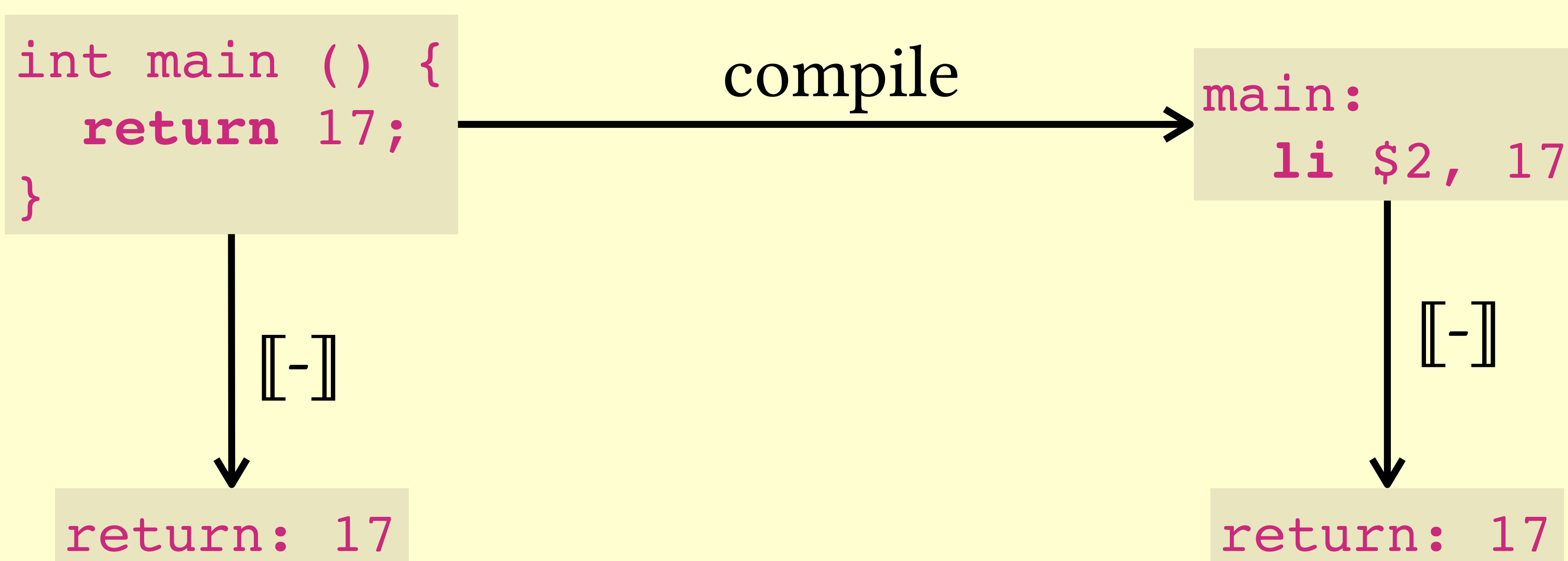
Xavier Leroy
1968–

CompCert performance



CompCert correctness

- **Main theorem:**
  For any C program P,
  $[\![P]\!] = [\![\text{compile}(P)]\!]$.



CompCert correctness

- **Main theorem:**
  For any C program P,
  $[\![P]\!] = [\![\mathrm{compile}(P)]\!]$.

```c
int main () {
  return (printf("a"), 1) +
    (printf("b"), 2);
}
```

compile

```
main:
  li $4, 'a'
  jal putchar
  li $4, 'b'
  jal putchar
  li $2, 3
```

$[\![-]\!]$

```
stdout: "ab"
return: 3
```

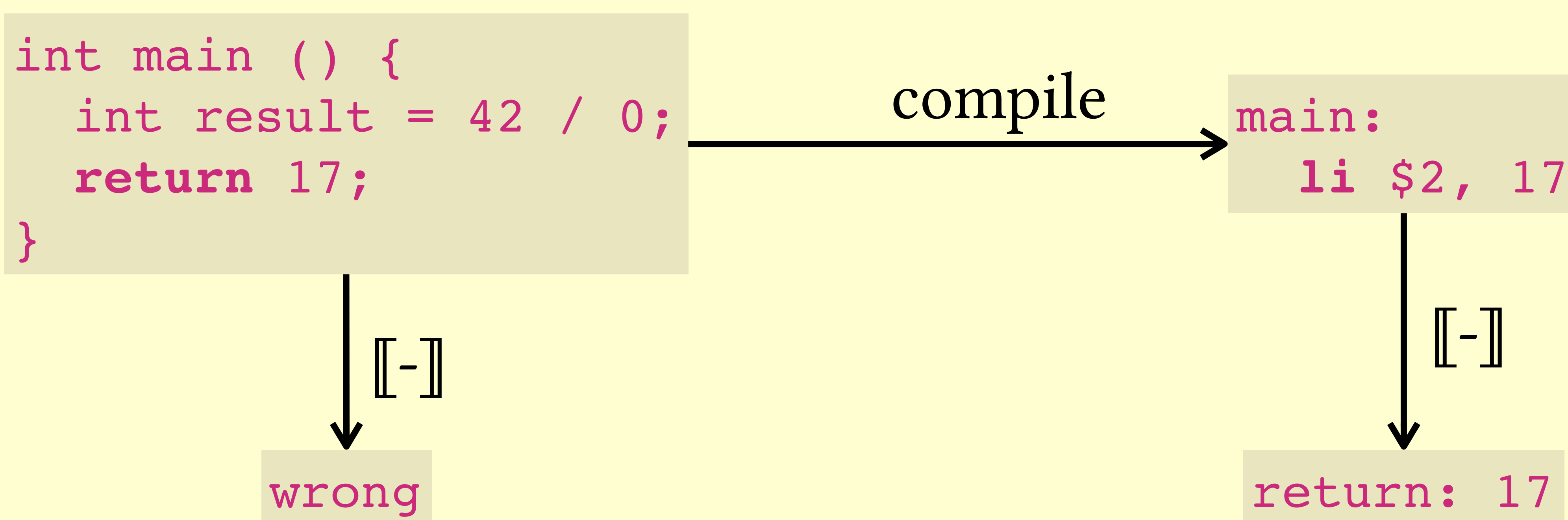
```
stdout: "ba"
return: 3
```

$[\![-]\!]$

```
stdout: "ab"
return: 3
```

CompCert correctness

- **Main theorem:**
  For any C_{light} program P,
  $[\![P]\!] = [\![compile(P)]\!]$.



CompCert correctness

- **Main theorem:**
  For any C_{light} program P,
  if $[\![P]\!] \neq$ `wrong` then
  $[\![P]\!] = [\![\mathrm{compile}(P)]\!]$.

```
int main () {
  return 1 * 1 * ... * 1;
}
```

$\xrightarrow{\text{compile}}$

```
Compilation error!
```

CompCert correctness

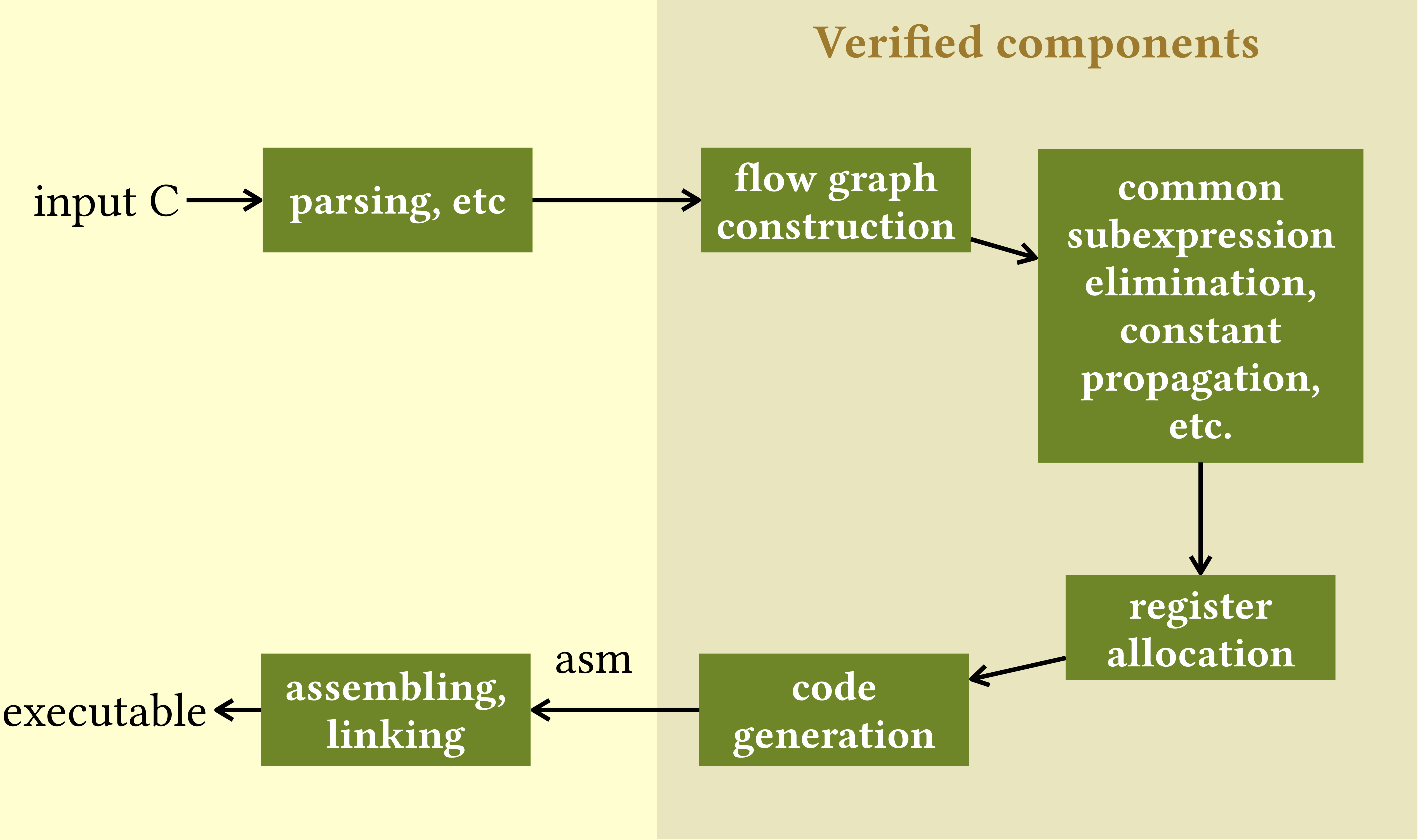
- **Main theorem:**
  For any C_{light} program P,
  if $[\![P]\!] \neq$ `wrong` and the compiler finishes then
  $[\![P]\!] = [\![\text{compile}(P)]\!]$.

`https://compcert.org`

CompCert vs. Csmith

	GCC	LLVM COMPILER INFRASTRUCTURE	CompCert
crashes	50	136	~5
wrong code bugs	29	66	~8

CompCert structure



Summary

- Compiler correctness is crucial because compilers are **prevalent** and **trusted**.

- Csmith generates random, deterministic C programs, which can be used for **differential testing** or **EMI testing** of compilers.

- CompCert is an optimising C compiler that is **proven** correct.