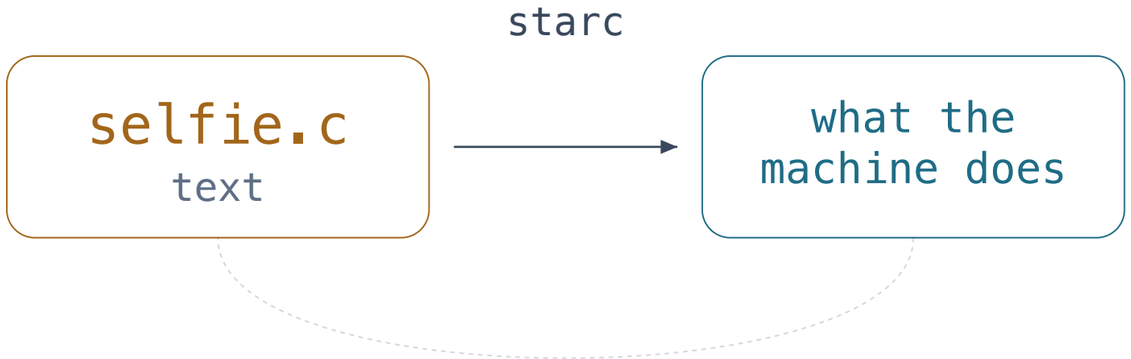
INTRODUCTION TO COMPUTER SCIENCE · WEEK 8 · STATION III

Self-Reference I

A compiler defines the meaning of the language it is written in.
The fixed point, what it cannot prove — and Gödel.

WHAT A COMPILER REALLY IS

A compiler defines the meaning of the language it is written in.



notation → meaning · and the arrow lives inside the box

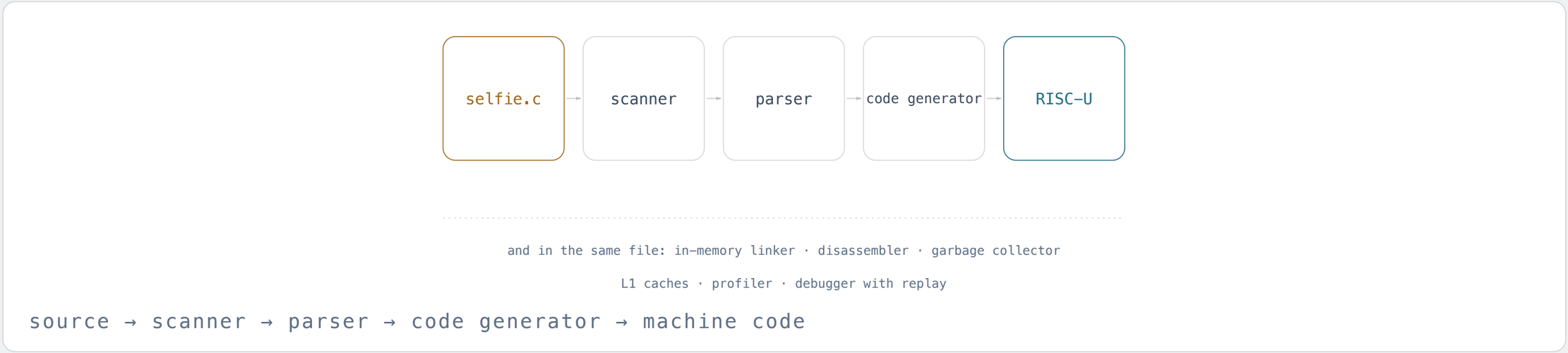
On the left, `selfie.c`: text, 365,784 characters. On the right, what the machine does when it runs. The arrow between them is `starc`, the compiler — and starc's own source code is in the left box.

That is a dictionary written in the language it defines. In English it would be a paradox. Here it is not, and the difference is that the definition is *executable*: a machine can run the dictionary.

Week 4 fixed meaning by decree. This is the decree, and it is a program.

NOTHING IN THE MIDDLE IS LEFT OUT

A parse is a proof. Code generation constructs semantics.



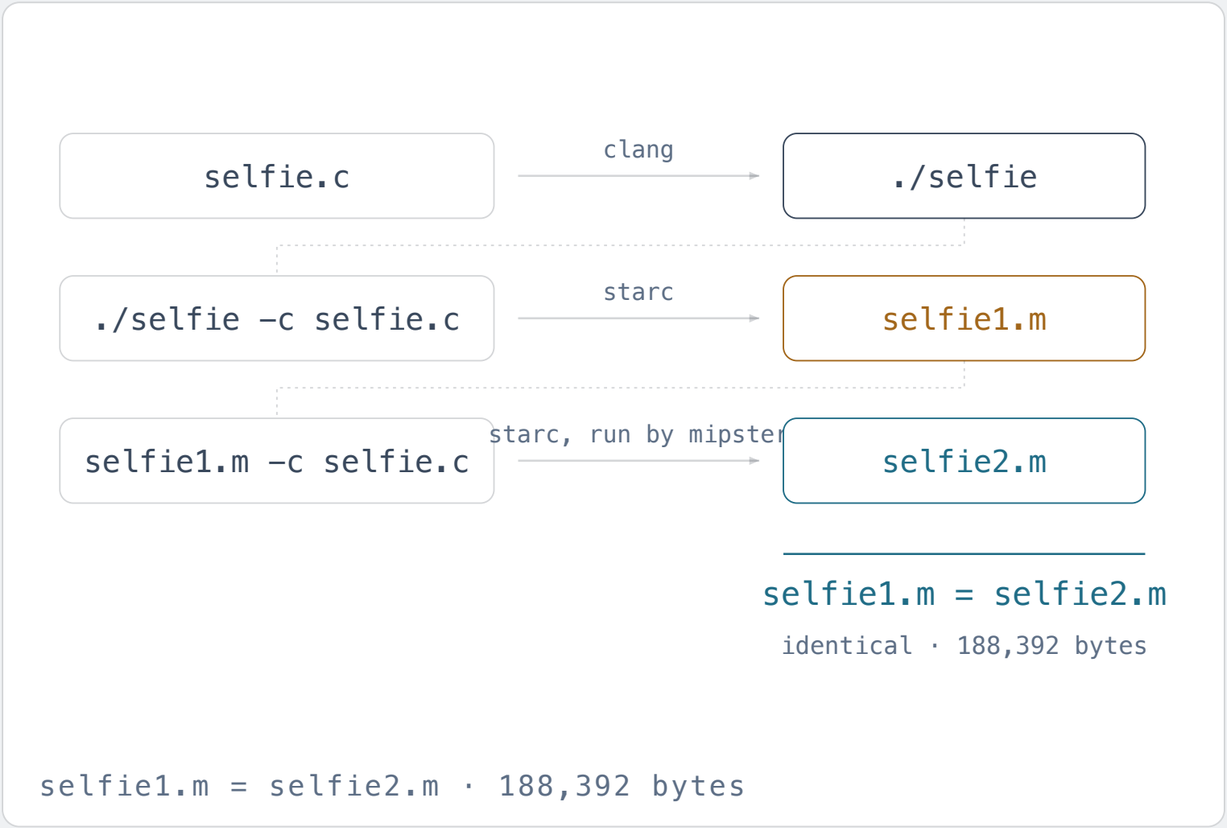
Scanner. Characters into symbols — the 22 symbols and 7 keywords of week 4. A finite state machine, one per kind of symbol.

Parser. Symbols against the grammar. A successful parse is a derivation, which is a proof that the text is a C* program. It always terminates and always answers. Syntax is *decidable*.

Code generator. For each construct, the RISC-U instructions that mean it: the nine lines of week 4's loop. This is where meaning is made, and it is the only place.

THE FIXED POINT

Selfie compiles selfie and gets the same bytes twice. What does that prove?



Exactly one thing: that selfie agrees with itself. A compiler that consistently mistranslated one construct would reproduce the mistake perfectly, and pass.

It does not prove that selfie is correct. It does not prove that selfie.c is what compiled selfie. The source does not determine the binary.

```
$ make self
./selfie -c selfie.c -o selfie.m -s selfie.s -m 1
…
```

TRUSTING TRUST · 1984

The source does not determine the binary.

Ken Thompson's Turing award lecture. A compiler that recognises when it is compiling the login program and inserts a back door — and recognises when it is compiling a compiler, and inserts the code that does both.

Remove the trick from the compiler's source. Recompile with the infected compiler. The back door survives, because the source is not what compiled the compiler. Every line inspected and found clean; the binary still lies.

> *You can't trust code that you did not totally create yourself.*
>
> KEN THOMPSON, REFLECTIONS ON TRUSTING TRUST

THE OUTSIDE CHECK

Compile the compiler with two unrelated compilers and compare the results — David Wheeler, 2005. The check that the fixed point cannot supply comes from *outside* the loop. Remember the shape.

THEOREM

There are truths with no proof.

GÖDEL · FIRST INCOMPLETENESS · 1931

> *Any consistent formal system rich enough to describe arithmetic contains statements that are true but not provable within it.*

How. Build, by Gödel numbering, a sentence that says: "this sentence has no proof in this system." If it were provable, the system proves a falsehood. So it is unprovable — and therefore true.

The same diagonal. Cantor built a row not on the list. Gödel builds a truth not on the list of provable things. Adding it as a new axiom does not help: the construction simply runs again.

Proof is a finite object you can check. Truth is not. Proof is syntax. Truth is semantics. They are not the same size — and week 7 already knew that. Gödel says where to look.

THEOREM

And no strong system can certify itself.

GÖDEL · SECOND INCOMPLETENESS · 1931

Such a system cannot prove its own consistency — unless it is inconsistent, in which case it proves everything.

READ IT AS ENGINEERING

A trustworthy system cannot be the source of its own trust. Confidence has to come from *outside*: a stronger theory, an experiment, an independent compiler, reality.

READ IT AS ADVICE

When a machine explains why its own answer is correct, you have received more output from the same machine — fluent, plausible, and not a certificate. Introspection is not audit.

Tarski, 1936, closes the circle: no sufficiently expressive language can define truth for its own sentences. Truth always lives one level up.

IN THE TERMINAL

What the compiler decides, and what it only warns about.

```
$ printf 'uint64_t main() { return 1 + ; }' > bad.c; ./selfie -c bad.c
./selfie: syntax error in bad.c in line 1: unexpected symbol ";" found
$ printf 'uint64_t* p; uint64_t main() { p = 42; return 0; }' > warn.c; ./selfie -c warn.c
./selfie: warning in warn.c in line 1: type mismatch, uint64_t* expected but uint64_t found
$ printf 'uint64_t main() { return 1 / 0; }' > div.c; ./selfie -c div.c -m 1
./selfie: division by zero
./selfie: 64-bit mipster terminating 64-bit RISC-U binary div.c with exit code 22
```

The first is about the *text*: the parser decided it, always, in finite time. The second is about the text too — a type is a label the compiler attached — so it is decided, and only a warning because C* is permissive.

The third is about *behaviour*. The compiler said nothing. The machine found it, on this run, on this input. Next week: why no compiler can ever say it in general — and why that is fine.

BEFORE NEXT WEEK

Recommended exercises.

1. Read the Meaning chapter from *Syntax and Semantics* to *Trusting Trust*, and the two station callouts in the Programming chapter.
2. Read Thompson's lecture. It is three pages. Write down, in one sentence, what an outside check for it looks like.
3. Run `make self-self-check` once more. Then state, in one sentence each, what the identical binaries prove and what they do not.
4. Write three C* programs: one with a syntax error, one with a type warning, one that divides by zero at runtime. Say for each whether the compiler could have known.
5. In your own words: why is a dictionary written in the language it defines a problem in English and not in C*?

NEXT WEEK

Self-reference II: will this program ever stop? No program can always tell. Then the same move run forwards — one machine that can be any machine — and Rice: every interesting question about behaviour.