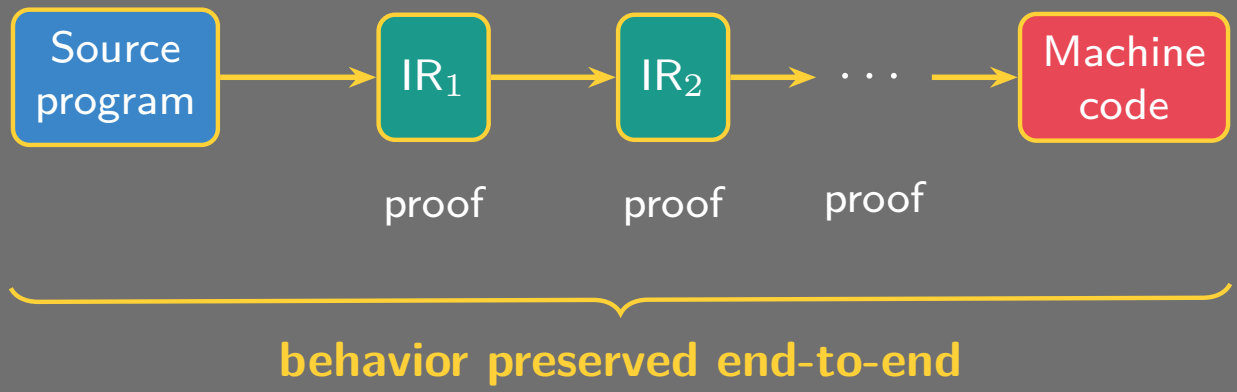
Certified Compilers

Charles Averill

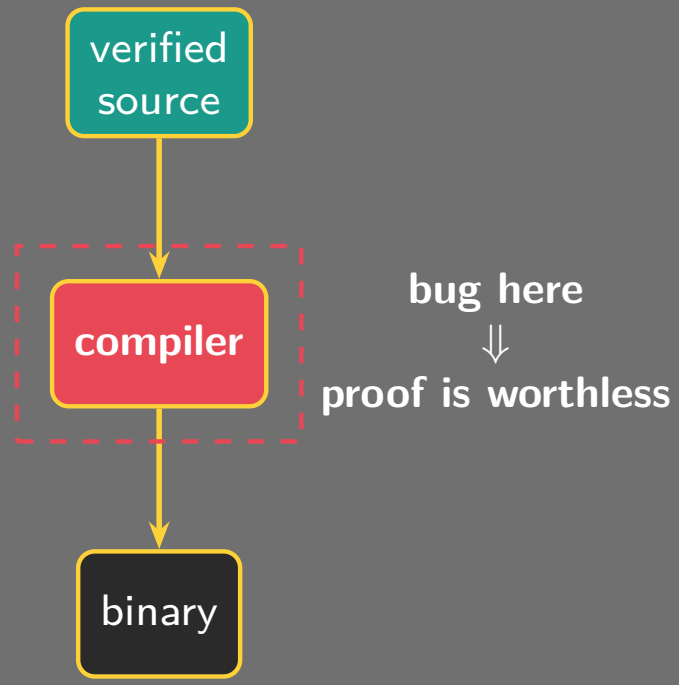
Dartmouth College

June 12, 2026



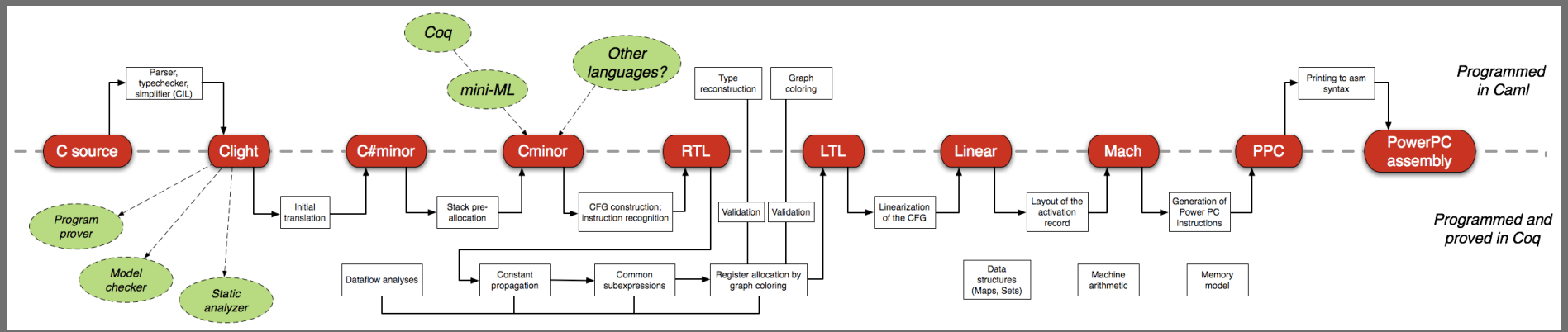
Why verify a compiler?

- Everything gets compiled
- Securing high level code and letting an untrustworthy transformation un-secure it = bad
- Safety-critical code can't tolerate this
- Solution: prove that the compiler preserves the behavior of input programs

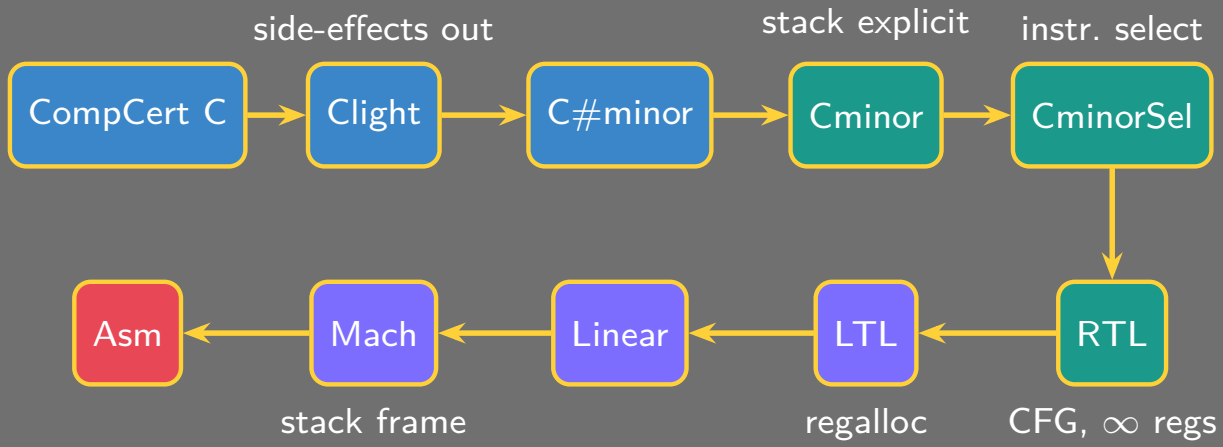


CompCert

- Xavier Leroy et al. (INRIA), in development since ∼2005
- Compiles a *large subset of C* to PowerPC / ARM / x86 / RISC-V
- *∼16 passes, ∼10 intermediate languages*
- Written in Rocq, executable code *extracted* to OCaml



IR Pipeline



- Top row: *structured* languages
- Bottom row: *unstructured* assembly-like languages
- Most optimization happens at RTL

IR Transformations

Lowering

- Clight: pull side-effects out of expressions
- C#minor: resolve C types, overloading → explicit ops
- Cminor: stack-allocate address-taken locals
- RTL gen: flatten to a *control-flow graph*
- Lowering to Mach: build the activation record

Optimization

- Constant propagation
- Common-subexpression elimination
- Inlining
- Dead-code elimination
- Register allocation

IR Semantics

- Uniform framework among all IRs: *labeled transition systems*
- *Small-step*: states $\rightarrow$ states, emitting events
- Events/observable behavior: I/O, volatile, external calls
- Behavior: trace of events a program produces
- Same memory model shared across all IRs

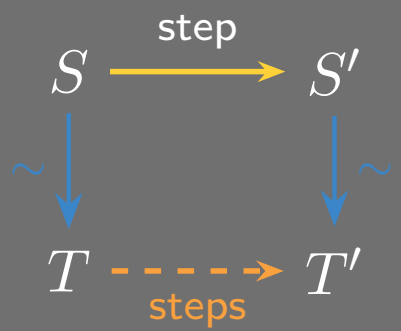
trace: [write 5]

Semantic Preservation

Per-pass theorem (informal)

If pass T maps program P to P' without error, and P has observable behavior B, then P' has behavior B — unless P had *undefined behavior* (then no guarantee).

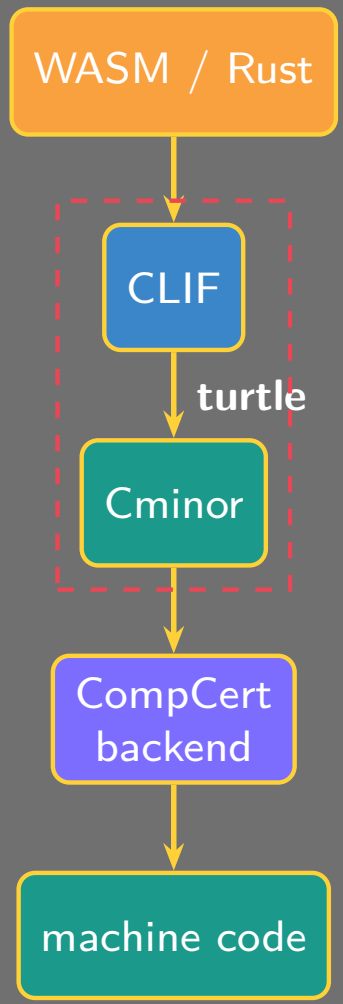
- **Forward Simulation**: Source step $\Rightarrow$ matching target steps (existential)
- Easy to prove, but does not rule out unwanted target behaviors
- **Deterministic Target**: Assembly execution has exactly one path
- Determinism guarantees that any target behavior is a valid source behavior (universal)



simulation relation $\sim$

turtle-rocq

- Translates *CLIF* → *Cminor*, pass to CompCert
- Reuses verified back-end for free
- New obligation shrinks to verifying CLIF → Cminor translation



Fibonacci example

From a CLIF `fib` (iterative, two accumulators), through turtle → Cminor → CompCert, we get correct, lightly-optimized x86-64:

```
turtle_fib:
    movl   $1, %eax       ; a = 1
    movl   $1, %edx       ; b = 1
.L100:
    testq  %rdi, %rdi     ; n<=0?
    jle    .L101
    leaq   -1(%rdi), %rdi ; n-
    leaq   0(%rax,%rdx,1), %rcx ; tmp = a + b
    movq   %rdx, %rax     ; a = b
    movq   %rcx, %rdx     ; b = tmp
    jmp    .L100
.L101:
    ret                   ; return a
```