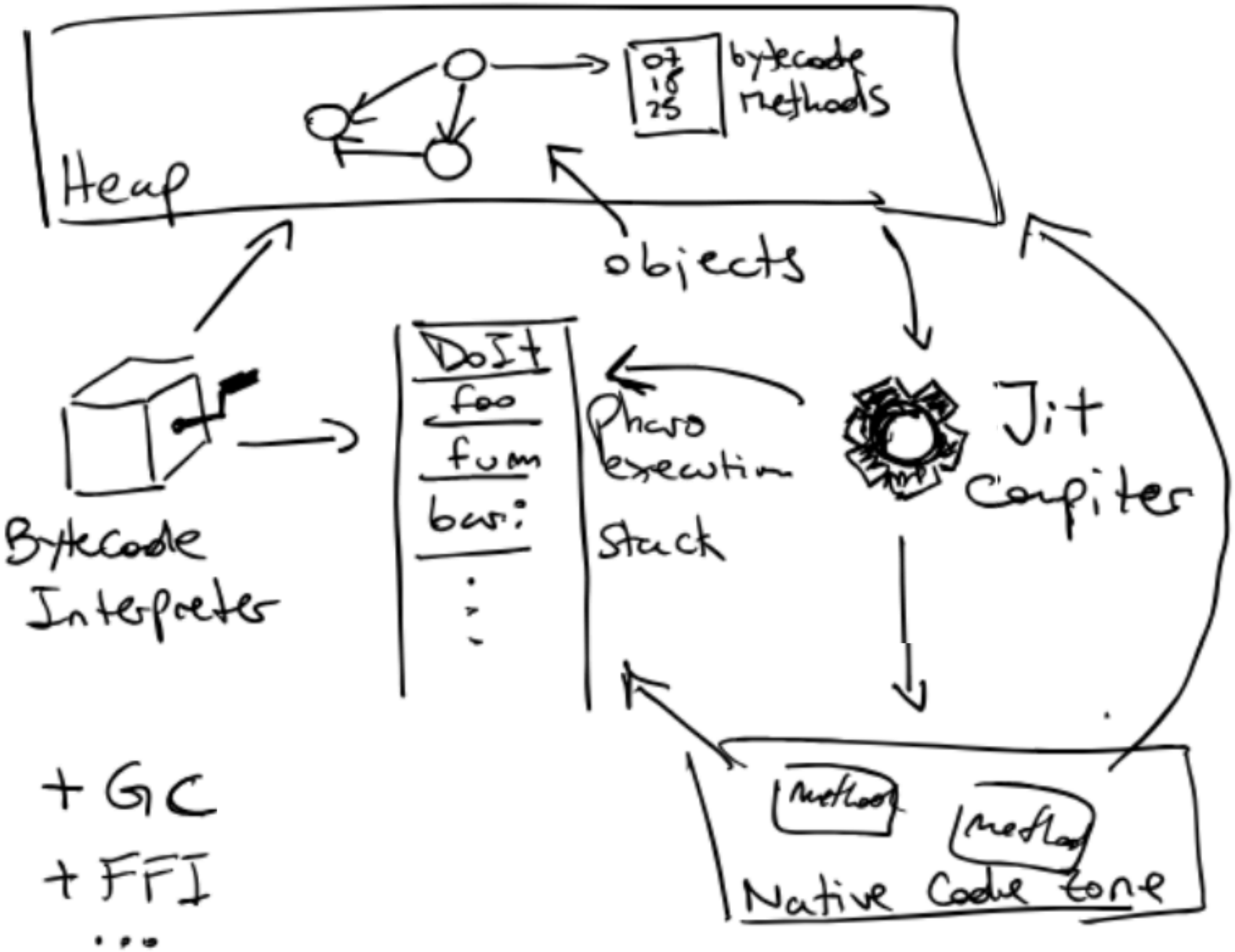
Engineering a Compiler

Writing an ARMv8 backend for a JIT compiler in 2 days per week

Guille Polito

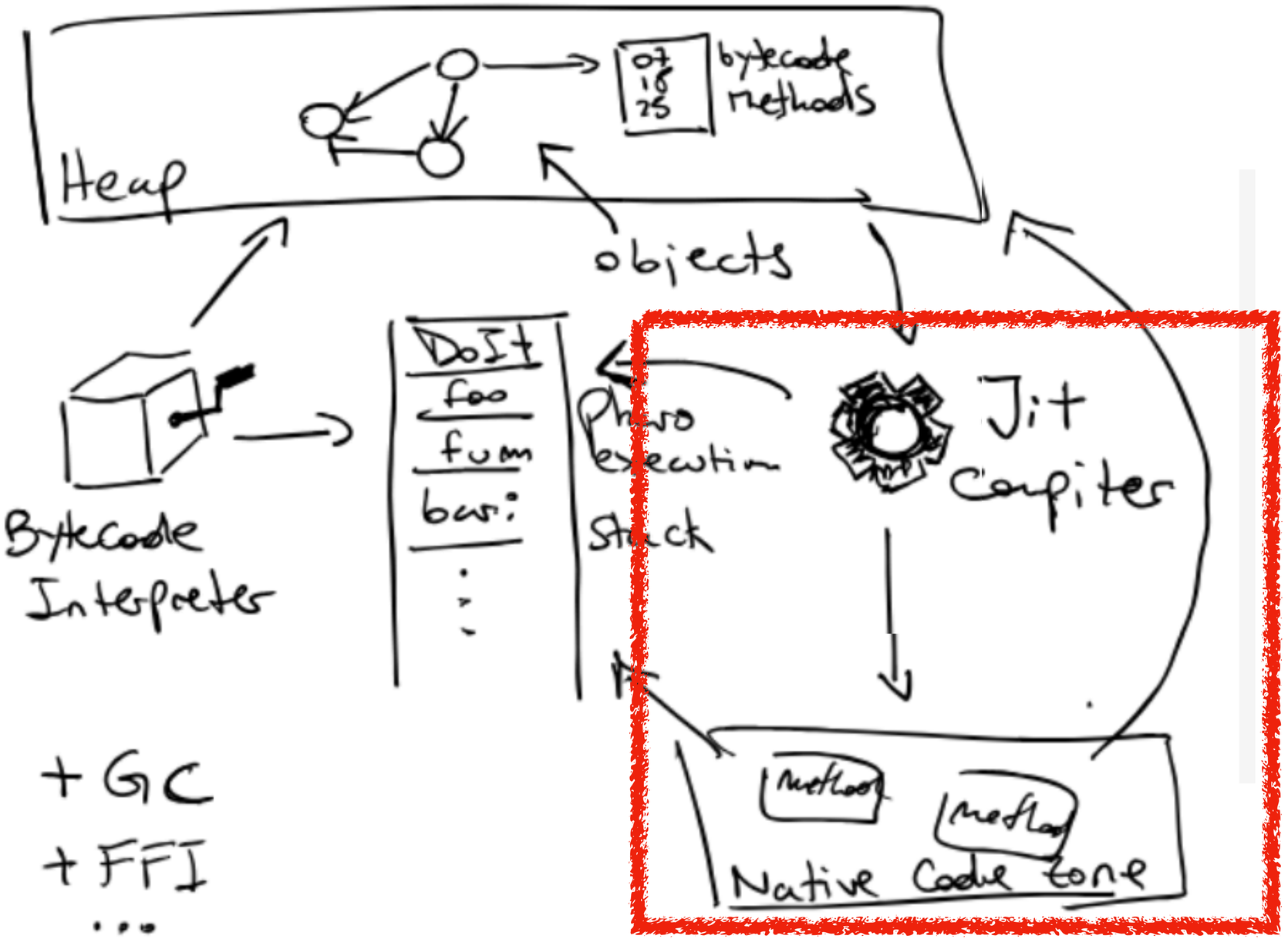
Context

The Pharo VM



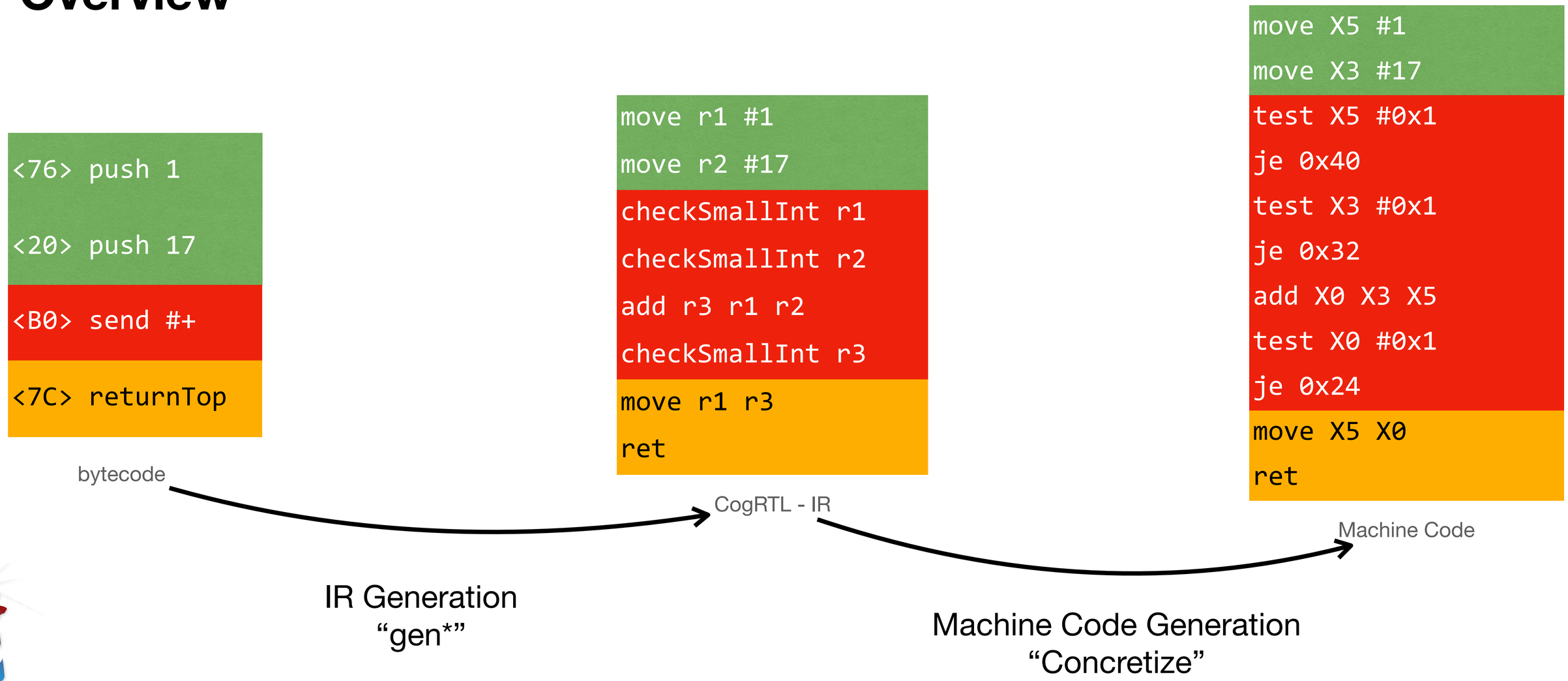
Context

The Pharo VM



Pharo VM JIT Compiler

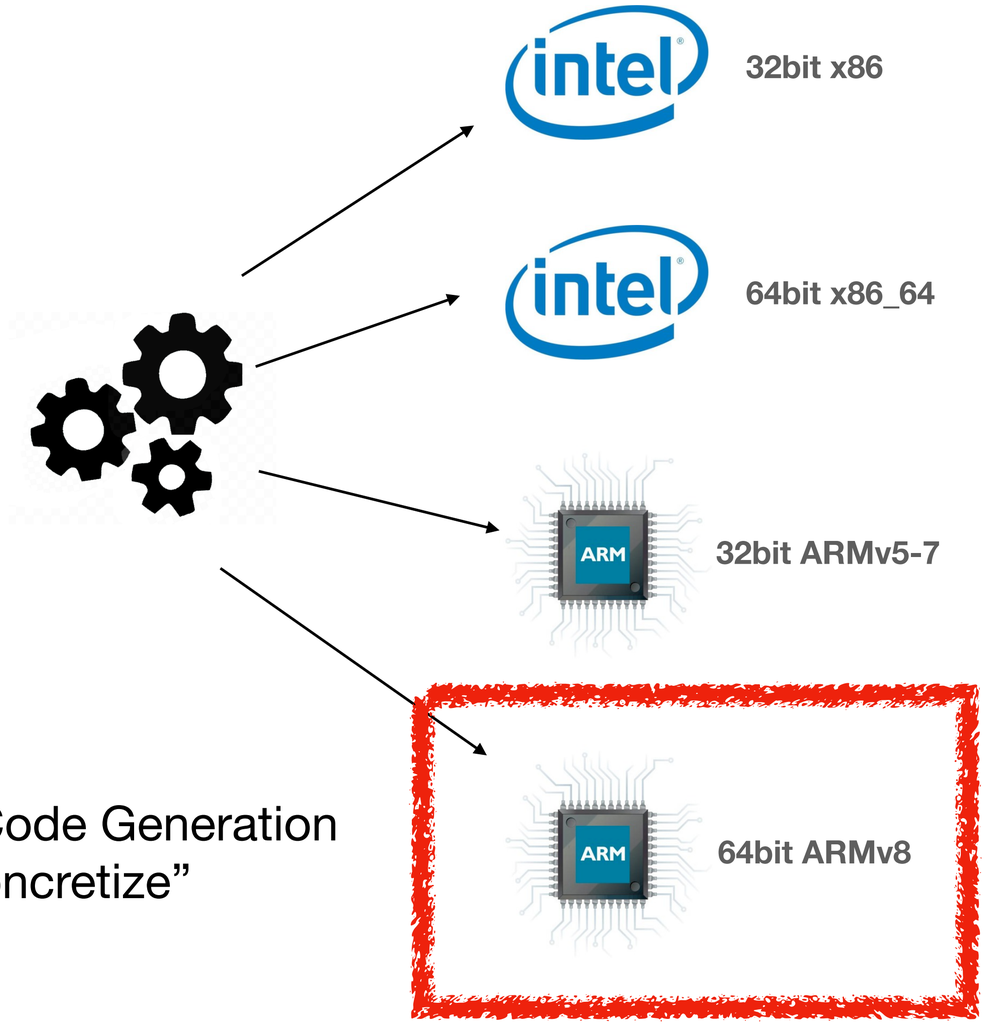
Overview



Implementing an ARMv8 Backend

- ARMV8 is now pervasive:
  - New Apple Silicon
  - Raspberry Pi 4
  - Microsoft Surface Pro X
  - PineBook Pro
  - …

```
move r1 #1
move r2 #17
checkSmallInt r1
checkSmallInt r2
add r3 r1 r2
checkSmallInt r3
move r1 r3
ret
```



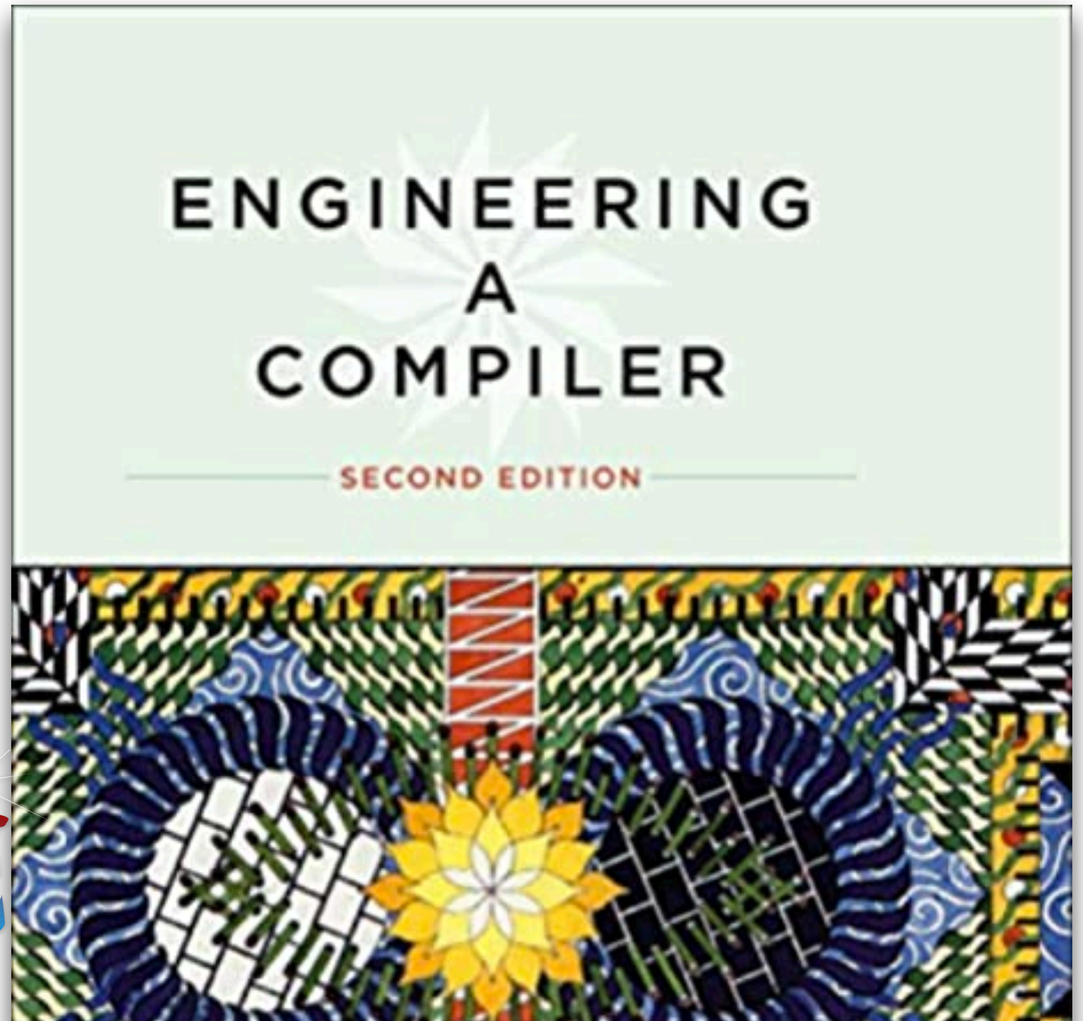
Machine Code Generation
“Concretize”

Some other numbers to contextualise

- 255 bytecode (77 different) + ~340 primitives
- 146 different IR instructions
- Instruction patching (+runtime disassemble)
  - optimizations such as inline caches
  - unused generated machine code is garbage collected
    - (thus moved, and its callers need to be re-linked)

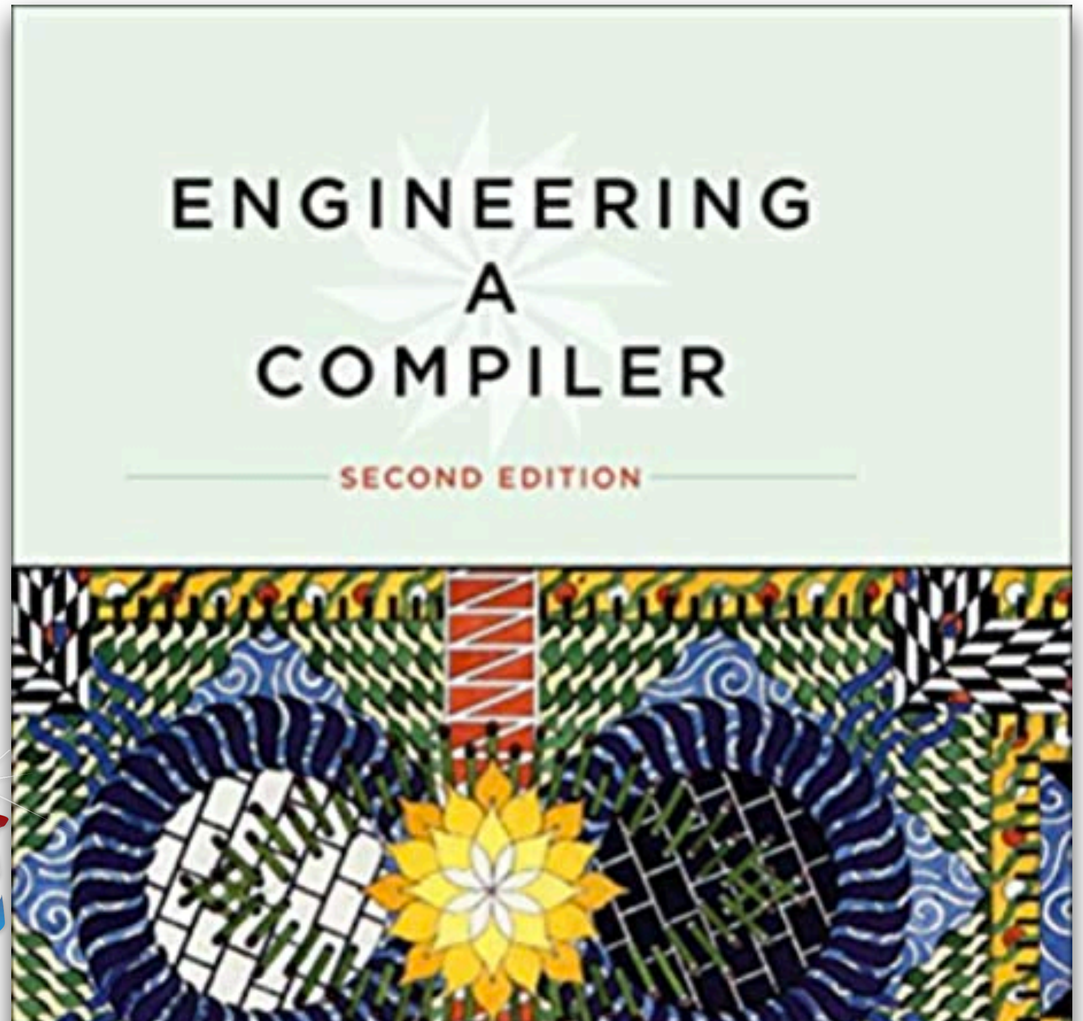
Lots of combinations!

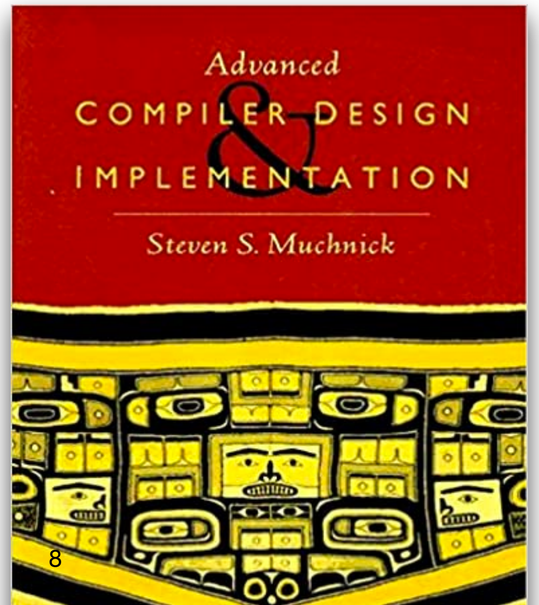
Engineering a Compiler

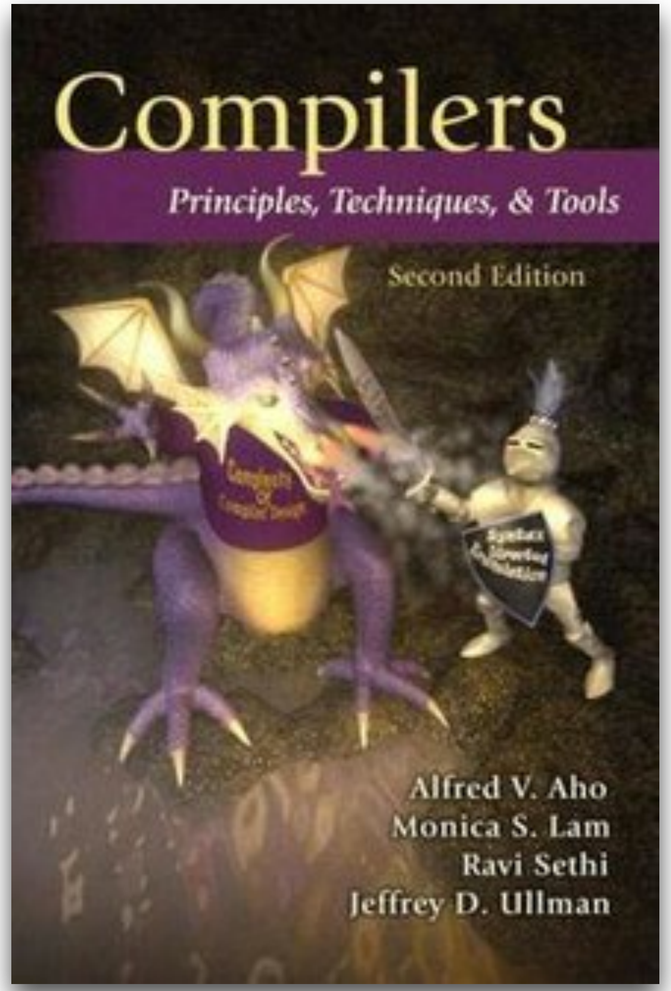


Engineering a Compiler

Good books, but centered on the *underlying algorithms*





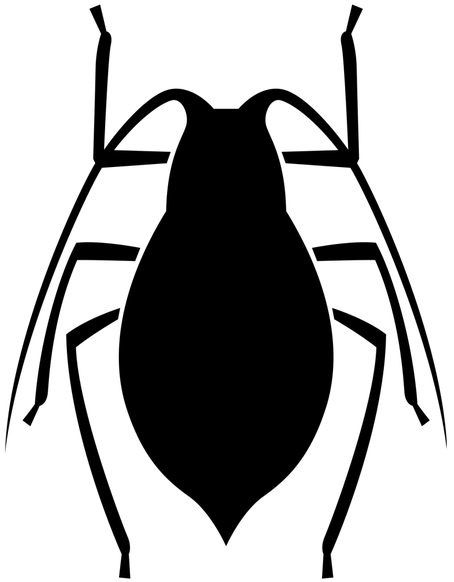


~~Engineering~~ a Compiler

Software Engineering

Testable

Incremental



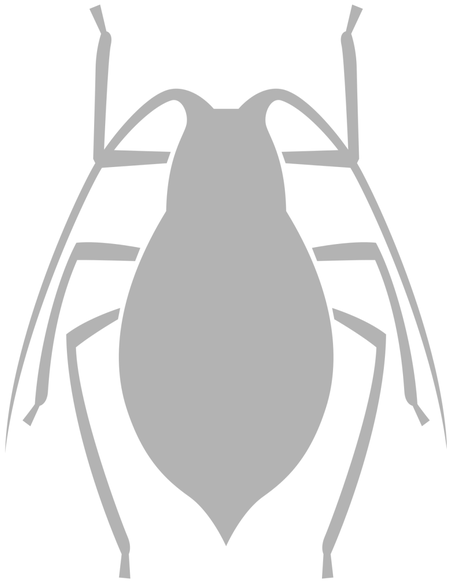
Easy Debugging

~~Engineering~~ a Compiler

Software Engineering

Testable

Incremental



Easy Debugging

Testing a Compiler

Insights: Black box testing

Black box testing

=> depend only on **observable behaviour**

=> **reusable** in different backends

=> more **resistant to changes** in the implementation

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Testing a Compiler

Insights: Cross-compile / Cross-execute

http://www.unicorn-engine.org

Use a machine simulator

=> **hardware independent:** test and debug in any machine any backend

=> **parametrizable tests** run the same test with multiple backends

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Intermezzo: Generating machine code

To LLVM or not to LLVM

- LLVM is a compiler infrastructure: its own IR, its assemblers and disassemblers
- It even has a JIT module
- But
  - **Not observable**: not straightforward to extract the generated code to test it
    => not testable!
  - **Lock-in**: it leaves **no control** on how machine code is generated, executed and managed
    => not compatible with the rest of our infrastructure

Intermezzo: Generating machine code

The backend **is** an *assembler*

Generates *machine code* from the intermediate representation

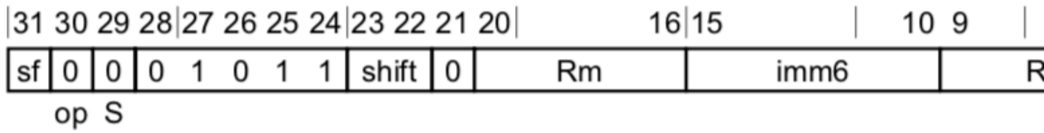
```
(is64Bits bitAnd: 1) << 31
      bitOr: ((substractionFlag bitAnd: 1) << 30
      bitOr: ((setFlagsFlag bitAnd: 1) << 29
      bitOr: (2r01011 << 24
      bitOr: ((shiftType bitAnd: 2r11) << 22
      bitOr: ((rightRegister bitAnd: 2r11111) << 16
      bitOr: ((immediate6bitValue bitAnd: 2r111111) << 10
      bitOr: ((leftRegister bitAnd: 2r11111) << 5
      bitOr: (destinationRegister bitAnd: 2r11111)))))))))
```

Arm® Architecture Reference Manual

Armv8, for Armv8-A architecture profile

C6.2.5 ADD (shifted register)

Add (shifted register) adds a register value and an optionally-shifted register va
destination register.



32-bit variant

Applies when sf == 0.

```
ADD <Wd>, <Wn>, <Wm>{, <shift> #<amount>}
```

64-bit variant

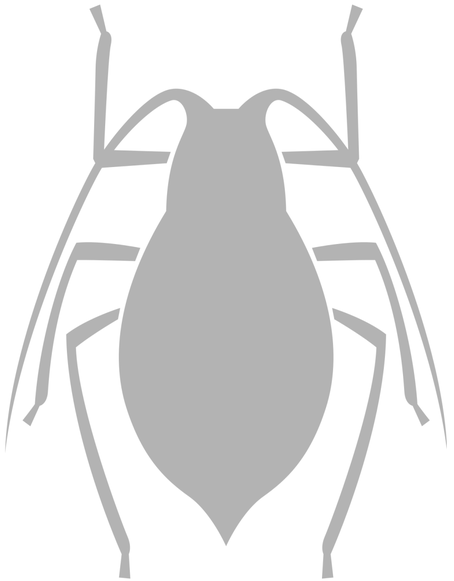
Applies when sf == 1.

~~Engineering~~ a Compiler

Software Engineering

Testable

Incremental



Easy Debugging

Incremental Compiler Engineering

Insights: Start Small

- First: The **simplest** test you can write for the **simplest** functionality
- Second: The **next simplest** test you can write for the **next simplest** functionality

=> The first focus is in understanding **how to** better write the **tests**

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Incremental Compiler Engineering

Insights: Invest in infrastructure

- **Refactor**
- **Clean**
- **Create Reusable Components**

```
testPushConstantZeroBytecodePushesASmallIntegerZero

  self compile: [ compiler genPushConstantZeroBytecode ].
  self runGeneratedCode.

  self assert: self popAddress equals: (memory integerObjectOf: 0)
```

Incremental Compiler Engineering

Insights: Don’t hesitate to step back

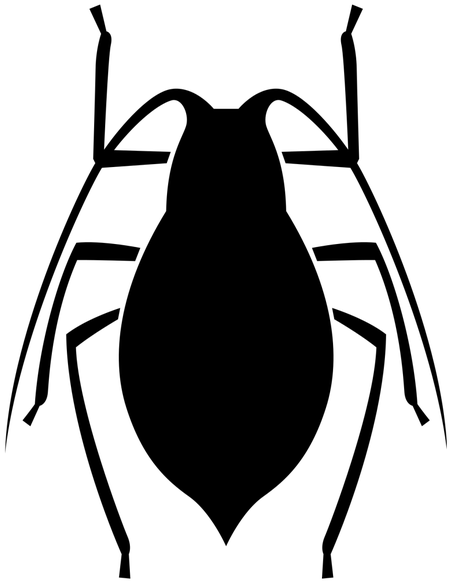
- If a test cannot be tamed with the current infrastructure
  - The infrastructure is not good!
  - Step back as soon as possible => **lose minutes, not weeks**
  - Choose a simpler test that could help you develop missing points

~~Engineering~~ a Compiler

Software Engineering

Testable

Incremental



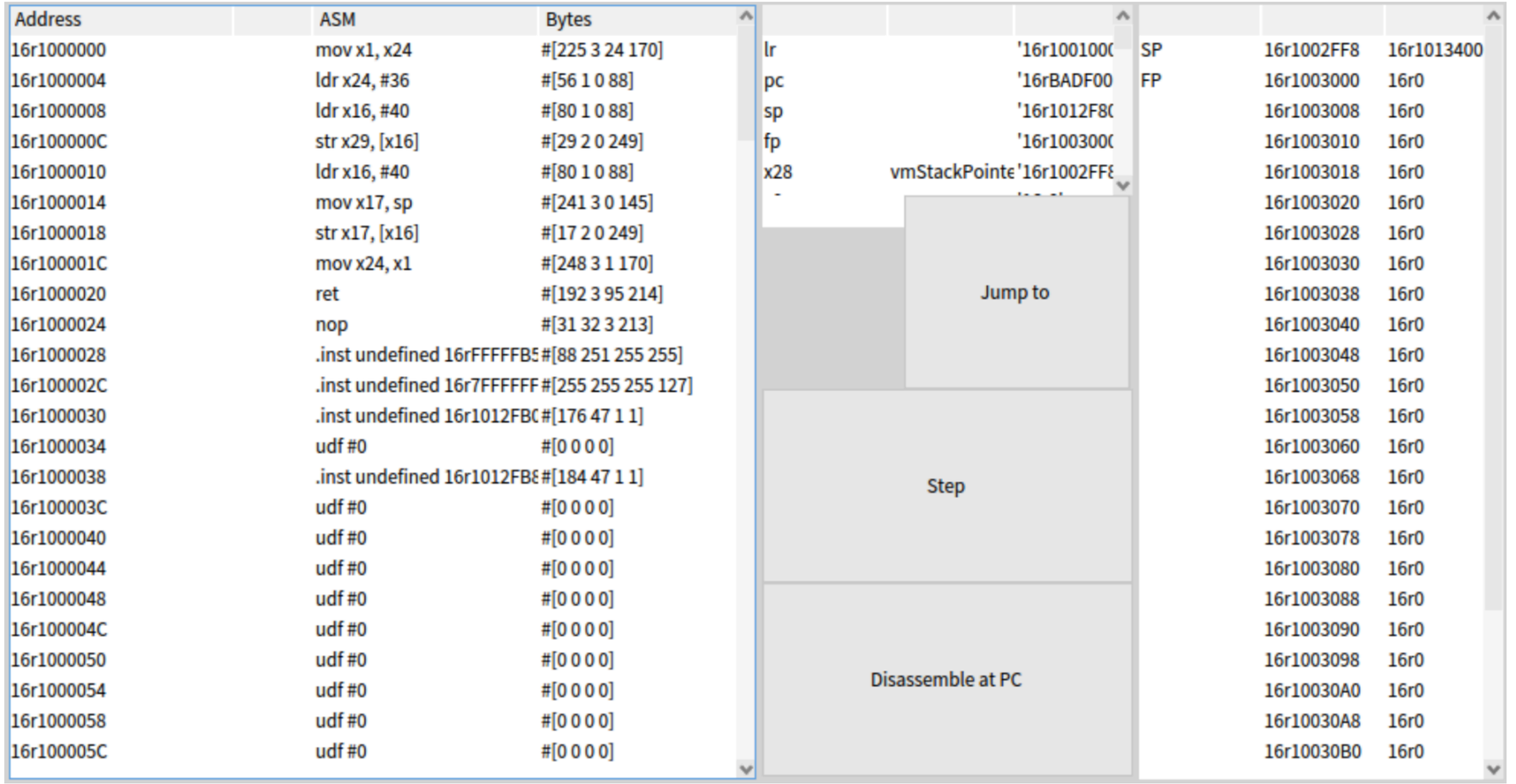
Easy Debugging

Debugging a compiler

Insights: prototype your own tools

- Prototype **fast**
- Prototype based on **needs**, not desires

Examples:
- Machine debugger
- Bytecode-IR visualization
- Disassembler DSL



Debugging a compiler

Insights: Get real execution feedback

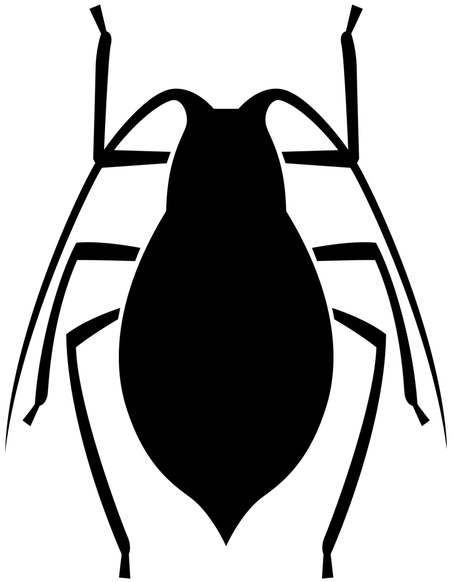
- Simulators are cheap, but not 100% trustworthy
- Full execution (simulated and real HW)
  - more expensive to run
  - cannot unit-test it (less controllable)
- Turn failures into tests
  - If you can reproduce it in a test, **you understand the bug**
  - **Fix with the aid of the test:**
    => the test is faster to run
    => and easier to debug than the real execution

~~Engineering~~ a Compiler

Software Engineering

Testable

Incremental

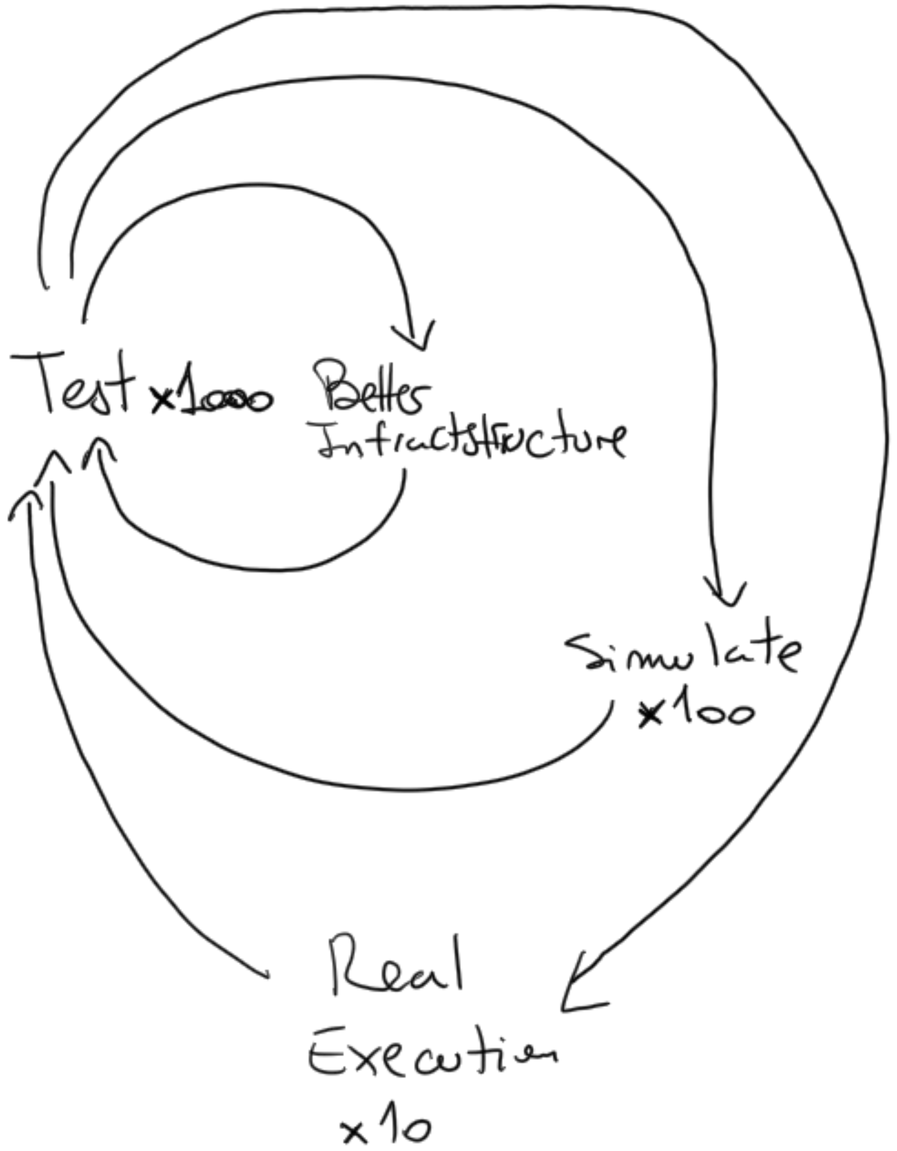


Easy Debugging

Software Engineering a Compiler

Our final workflow

- Test as much as you can
- Enhance your infrastructure as much too



- Simulate the execution, less than you run tests
- Run the real app, less than you simulate

- Turn failures into tests

Software Engineering a Compiler

Some Final Ideas/Insights

Tests are code

- **Design** your tests
- **Refactor** your tests
- Make them **modular and reusable**