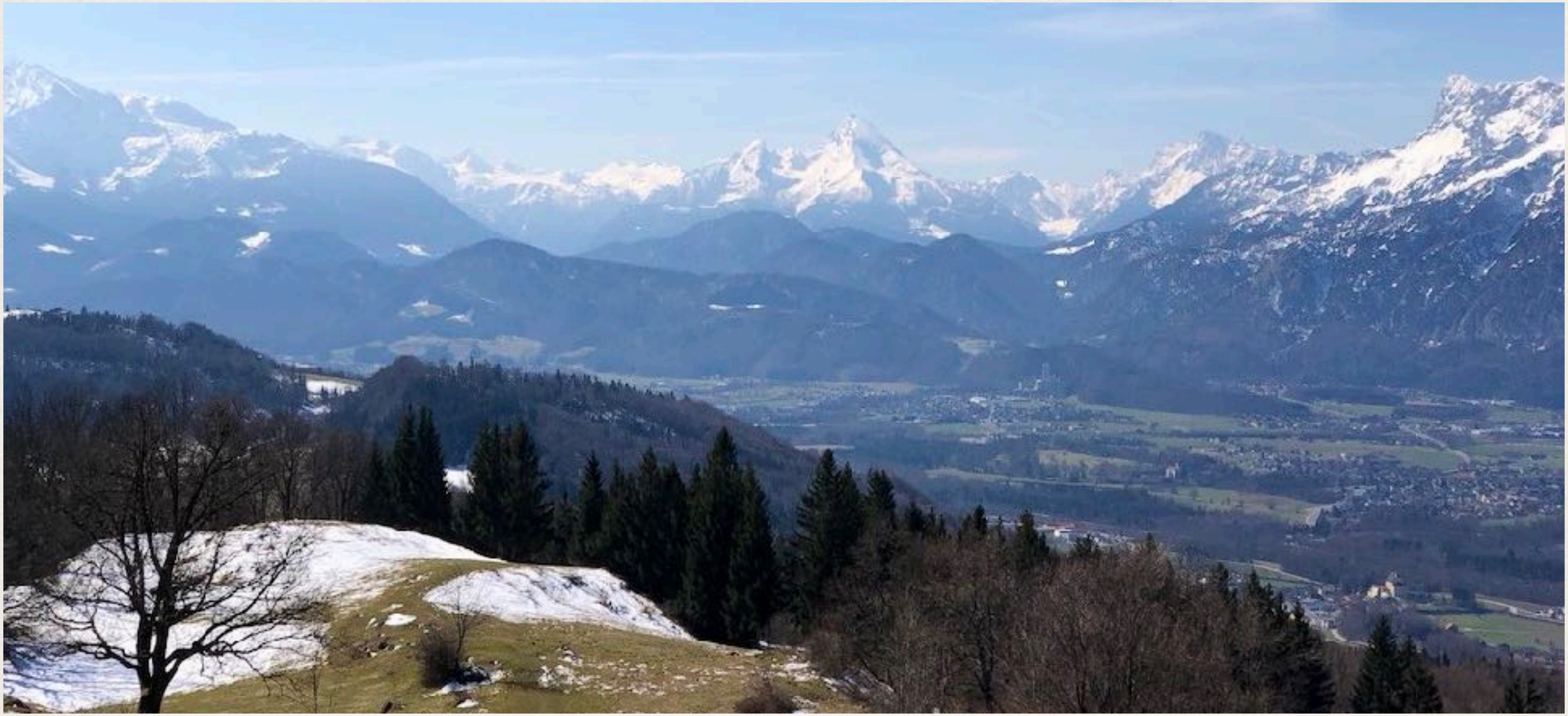


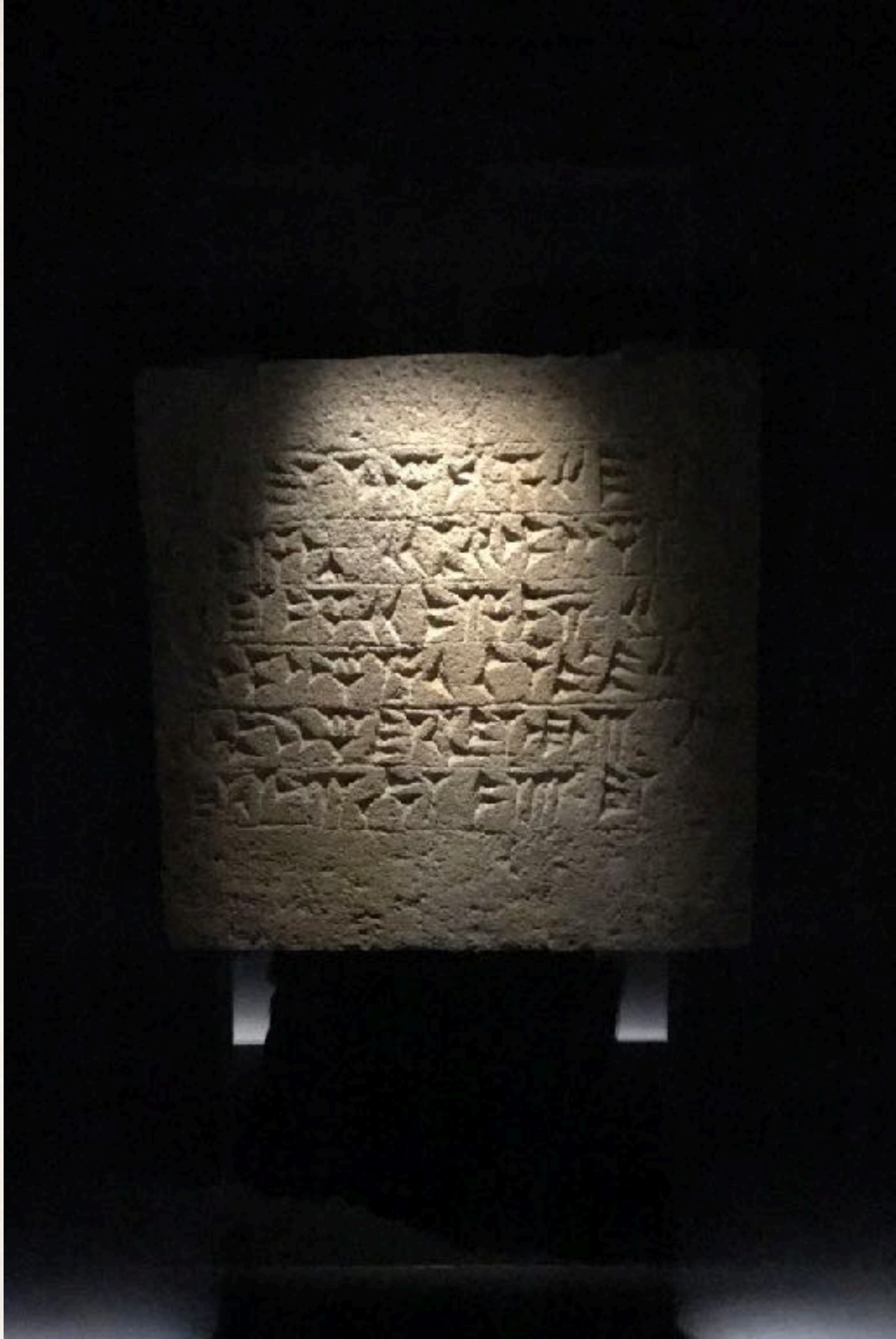
Selfie: Towards Minimal Symbolic Execution

Alireza S. Abyaneh, Simon Bauer, **Christoph M. Kirsch**, Philipp Mayer, Christian Mösl, Clément Poncelet, Sara Seidl, Ana Sokolova, and Manuel Widmoser, University of Salzburg, Austria

VDS Workshop, Essaouira, Morocco, May 2018 selfie.cs.uni-salzburg.at

What is the meaning of this sentence?

Selfie as in self-referentiality



Interpretation

Compilation

Teaching the Construction of Semantics of Formalisms

Virtualization

Verification

Joint Work

- Alireza Abyaneh
- Martin Aigner
- Sebastian Arming
- Christian Barthel
- Simon Bauer
- Thomas Hütter
- Alexander Kollert
- Michael Lippautz
- Cornelia Mayer
- Philipp Mayer
- Christian Moesl
- Simone Oblasser
- Clement Poncelet
- Sara Seidl
- Ana Sokolova
- Manuel Widmoser

Inspiration

- Armin Biere: SAT/SMT Solvers
- Donald Knuth: Art
- Jochen Liedtke: Microkernels
- David Patterson: RISC
- Niklaus Wirth: Compilers

Selfie: Teaching Computer Science [selfie.cs.uni-salzburg.at]

- *Selfie* is a self-referential 10k-line C implementation (in a single file) of:

1. a self-compiling compiler called ***starc*** that compiles a tiny subset of C called C Star (C*) to a tiny subset of RISC-V called RISC-U,

2. a self-executing emulator called ***mipster*** that executes RISC-U code including itself when compiled with starc,

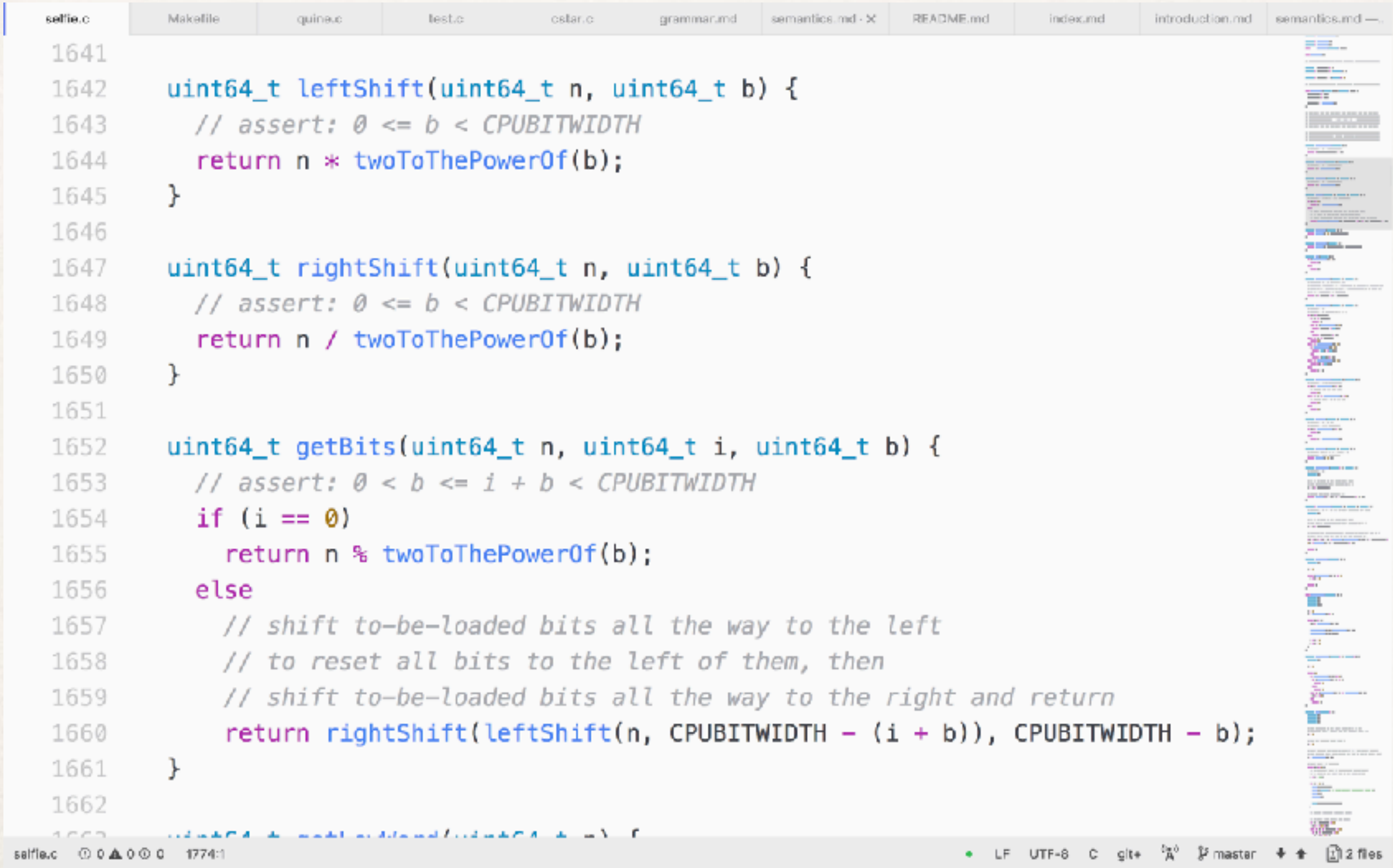
3. a self-hosting hypervisor called ***hypster*** that virtualizes mipster and can host all of selfie including itself,

4. a self-executing symbolic execution engine called ***monster*** that executes RISC-U code symbolically when compiled with starc which includes all of selfie,

5. a tiny C* library called ***libcstar*** utilized by all of selfie, and

6. a tiny, experimental SAT solver called ***babysat***.

Code as Prose (yeah, right, but still)



```
uint64_t leftShift(uint64_t n, uint64_t b) {
  // assert: 0 <= b < CPUBITWIDTH
  return n * twoToThePowerOf(b);
}

uint64_t rightShift(uint64_t n, uint64_t b) {
  // assert: 0 <= b < CPUBITWIDTH
  return n / twoToThePowerOf(b);
}

uint64_t getBits(uint64_t n, uint64_t i, uint64_t b) {
  // assert: 0 < b <= i + b < CPUBITWIDTH
  if (i == 0)
    return n % twoToThePowerOf(b);
  else
    // shift to-be-loaded bits all the way to the left
    // to reset all bits to the left of them, then
    // shift to-be-loaded bits all the way to the right and return
    return rightShift(leftShift(n, CPUBITWIDTH - (i + b)), CPUBITWIDTH - b);
}

uint64_t getLowWord(uint64_t n) {
```

Also, there is a…

- linker (in-memory only)
- disassembler (w/ source code line numbers)
- debugger (tracks full machine state w/ rollback)
- profiler (#proc-calls, #loop-iterations, #loads, #stores)
- ELF boot loader (same code for mipster/hypster)

Discussion of Selfie reached 3rd place on Hacker News

news.ycombinator.com

Website

selfie.cs.uni-salzburg.at

Code

github.com/cksystemsteaching/selfie

Slides (incomplete)

selfie.cs.uni-salzburg.at/slides

Book (draft)

leanpub.com/selfie

nsf.gov/csforall

code.org

computingatschool.org.uk

programbydesign.org

bootstrapworld.org

k12cs.org

csfieldguide.org.nz

5 statements:
assignment
while
if
return
procedure()

```
uint64_t atoi(uint64_t *s) {
    uint64_t i;
    uint64_t n;
    uint64_t c;

    i = 0;
    n = 0;
    c = *(s+i);

    while (c != 0) {
        n = n * 10 + c - '0';
        if (n < 0)
            return -1;

        i = i + 1;
        c = *(s+i);
    }

    return n;
}
```

no data types other than `uint64_t` and `uint64_t*` and dereferencing: the * operator

character literals
string literals

integer arithmetics
pointer arithmetics

no bitwise operators
no Boolean operators

library: `exit`, `malloc`, `open`, `read`, `write`

Minimally complex, maximally self-contained system

Programming languages vs systems engineering?

```
> make
cc -w -m64 -D'main(a,b)=main(a,char**argv)' selfie.c -o selfie
```

bootstrapping `selfie.c` *into x86* `selfie` *executable*
using standard C compiler

```
> ./selfie
./selfie: usage: selfie { -c { source } | -o binary | -s assembly
| -l binary } [ ( -m | -d | -y | -min | -mob ) size ... ]
```

selfie usage

```
> ./selfie -c selfie.c

./selfie: this is selfie's starc compiling selfie.c

./selfie: 176408 characters read in 7083 lines and 969 comments
./selfie: with 97779(55.55%) characters in 28914 actual symbols
./selfie: 261 global variables, 289 procedures, 450 string literals
./selfie: 1958 calls, 723 assignments, 57 while, 572 if, 243 return
./selfie: 121660 bytes generated with 28779 instructions and 6544
bytes of data
```

compiling `selfie.c` *with x86* `selfie` *executable*

(takes seconds)

```
> ./selfie -c selfie.c -m 2 -c selfie.c

./selfie: this is selfie's starc compiling selfie.c

./selfie: this is selfie's mipster executing selfie.c with 2MB of
physical memory

selfie.c: this is selfie's starc compiling selfie.c

selfie.c: exiting with exit code 0 and 1.05MB of malloced memory

./selfie: this is selfie's mipster terminating selfie.c with exit code
0 and 1.16MB of mapped memory
```

compiling `selfie.c` *with x86* `selfie` *executable into a RISC-U executable*

and

then running that RISC-U executable to compile `selfie.c` *again*

(takes ~6 minutes)

```
> ./selfie -c selfie.c -o selfie1.m -m 2 -c selfie.c -o selfie2.m

./selfie: this is selfie's starc compiling selfie.c
./selfie: 121660 bytes with 28779 instructions and 6544 bytes of data
written into selfie1.m

./selfie: this is selfie's mipster executing selfie1.m with 2MB of
physical memory

selfie1.m: this is selfie's starc compiling selfie.c
selfie1.m: 121660 bytes with 28779 instructions and 6544 bytes of data
written into selfie2.m

selfie1.m: exiting with exit code 0 and 1.05MB of mallocated memory

./selfie: this is selfie's mipster terminating selfie1.m with exit
code 0 and 1.16MB of mapped memory
```

compiling `selfie.c` *into a RISC-U executable* `selfie1.m`

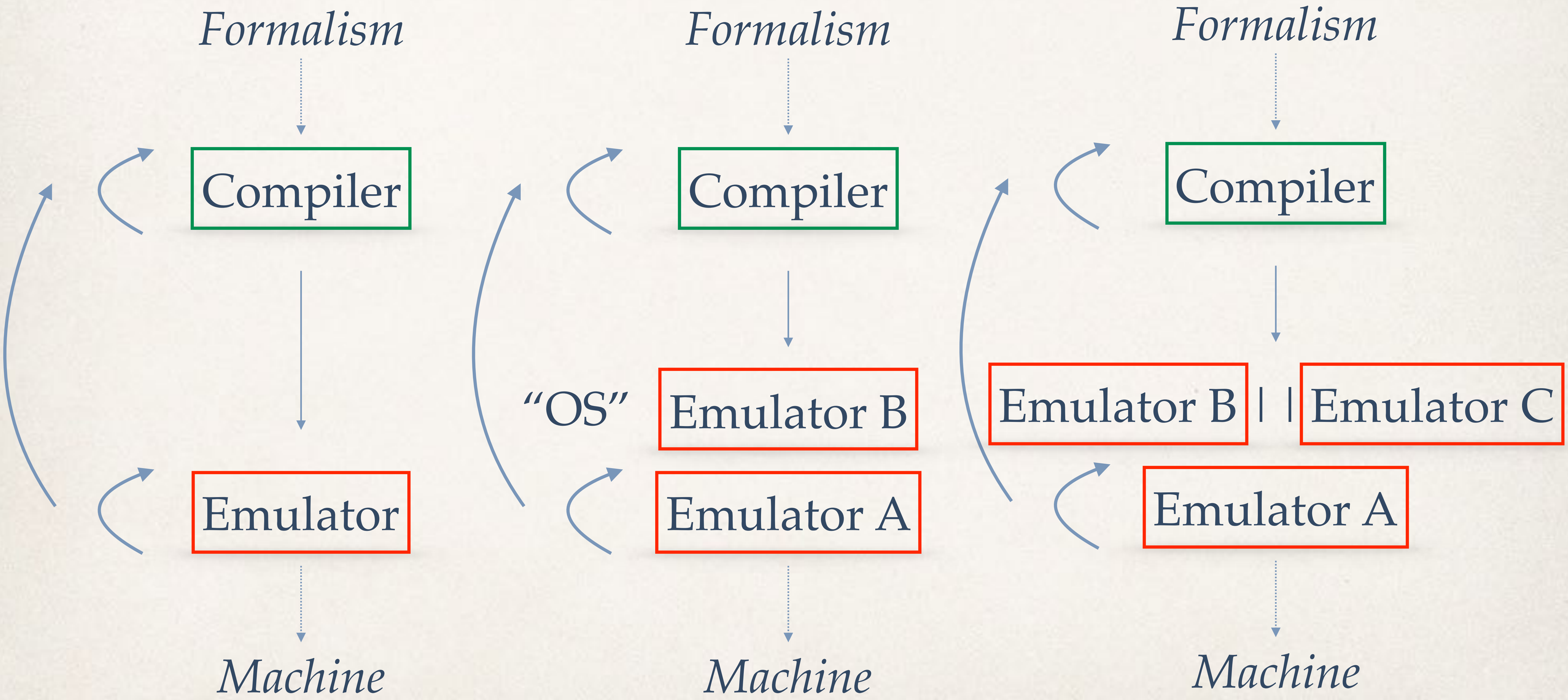
and

then running `selfie1.m` *to compile* `selfie.c`

into another RISC-U executable `selfie2.m`

*(**takes ~6 minutes**)*

Implementing an OS Kernel: 1-Week Homework Assignment



```
> ./selfie -c selfie.c -m 2 -c selfie.c -m 2 -c selfie.c
```

compiling `selfie.c` *with x86* `selfie` executable

<u>and</u>

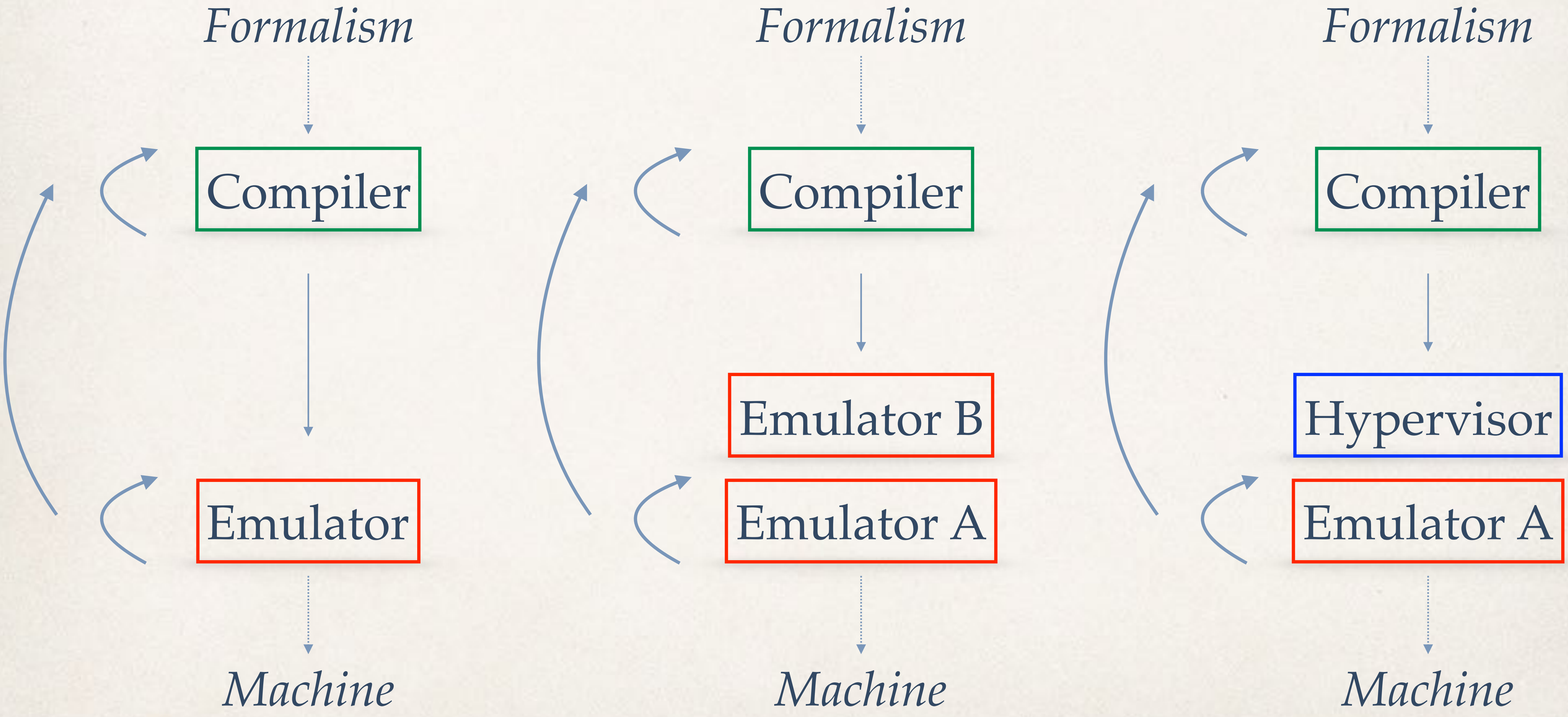
then running that executable to compile `selfie.c` *again*

<u>and</u>

then running that executable to compile `selfie.c` *again*

*(**takes ~24 hours**)*

Emulation versus Virtualization



```
> ./selfie -c selfie.c -m 2 -c selfie.c -y 2 -c selfie.c
```

compiling `selfie.c` *with x86* `selfie` *executable*

and

then running that executable to compile `selfie.c` *again*

and

*then **hosting** that executable in a virtual machine to compile* `selfie.c` *again*

*(**takes ~12 minutes**)*

Homework Ideas

- Implement bitwise shifting (<<, >> as well as SLL, SRL)
- Multi-dimensional arrays and recursive structs
- Lazy evaluation of Boolean operators
- Conservative garbage collection
- Processes and threads, multicore support
- Locking and scheduling
- Atomic instructions and lock-free data structures

Recep
Resta
Sp

Ongoing Work

Verification

- SAT/SMT Solvers (microsat/boolector)
- Symbolic Execution Engine (KLEE/SAGE)
- Inductive Theorem Prover (ACL2)

-> microsat in C* is as fast as in C (forget structs, arrays, &&, ||, goto)

ISAs

1. Large memory and multicore support
2. x86 support through binary translation
3. ARM support?

Replay vs. Symbolic Execution

- Selfie supports replay of RISC-U execution upon detecting runtime errors such as division by zero

- Selfie first rolls back *n* instructions (undo (!) semantics, system calls?) and then re-executes them but this time printed on the console

- We use a cyclic buffer for replaying *n* instructions

- That buffer is also used in symbolic execution but then for recording symbolic execution of up to *n* instructions

Minimal Symbolic Execution?

What exactly is needed to execute systems code like selfie's symbolically?

Symbolic Execution: Status

- We fuzz input read from files
- Symbolic execution proceeds by computing integer interval constraints, only recording memory stores
- Sound but only complete for a subset of all programs
- Selfie compiler falls into that subset, so far…
- We detect division by zero, (some) unsafe memory access

Symbolic Execution: Future

- Witness generation and on-the-fly validation
- Loop termination through manually crafted invariants
- Parallelization on our 64-core machine
- And support for utilizing 0.5TB of physical memory

Got Research Ideas?

- Selfie is a simple but still realistic sandbox
- You control everything!
- Want to play with an idea that requires compiler / operating systems / architecture support?
- We are glad to help you get started!

Thank you!

acsd2018.cs.uni-salzburg.at