Selfie: Sandboxed Concurrency

Christoph Kirsch, University of Salzburg, Austria

OPCT 2017, Maria Gugging, Austria

Joint Work

- Alireza Abyaneh
- Martin Aigner
- Sebastian Arming
- Christian Barthel
- Michael Lippautz
- Cornelia Mayer
- Simone Oblasser

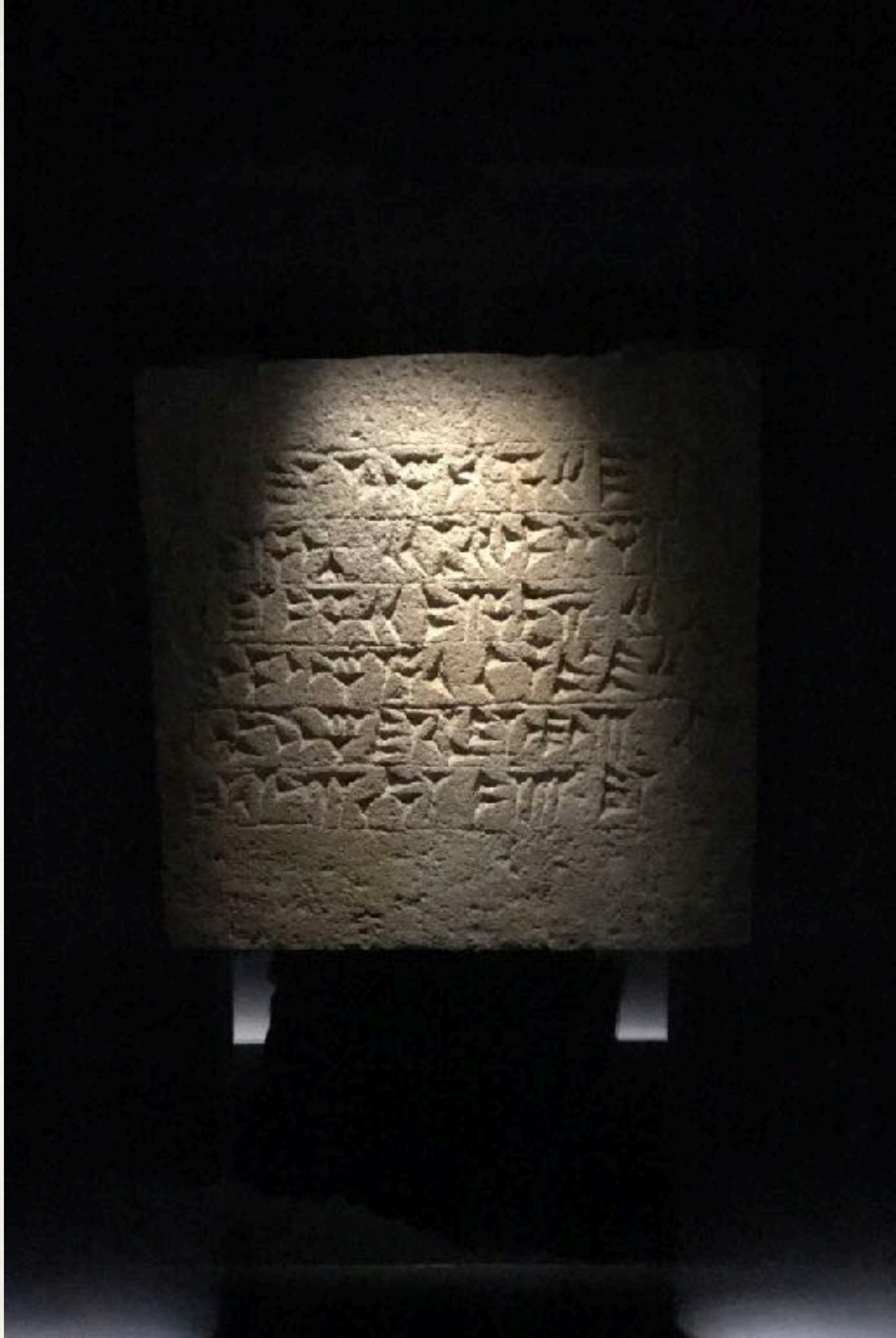
Inspiration

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

Teaching Computer Science from First Principles!

What is the meaning of this sentence?

Selfie as in self-referentiality



Interpretation

Translation

Teaching the Construction of <u>Semantics</u> of Formalisms

Virtualization

Verification

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

- *Selfie* is a self-referential 7k-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 MIPS32 called MIPSter,

  2. a self-executing emulator called *mipster* that executes MIPSter 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 tiny C* library called *libcstar* utilized by all of selfie, and

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

Website

selfie.cs.uni-salzburg.at

Book (Draft)

leanpub.com/selfie

Code

github.com/cksystemsteaching/selfie

Discussion of Selfie recently reached 3rd place on Hacker News

news.ycombinator.com

nsf.gov/csforall

code.org

computingatschool.org.uk

programbydesign.org

bootstrapworld.org

k12cs.org

csfieldguide.org.nz

5 statements:
assignment
while
if
return
procedure()

```
int atoi(int *s) {
    int i;
    int n;
    int 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 `int` and `int*` and dereferencing: the * operator

character literals
string literals

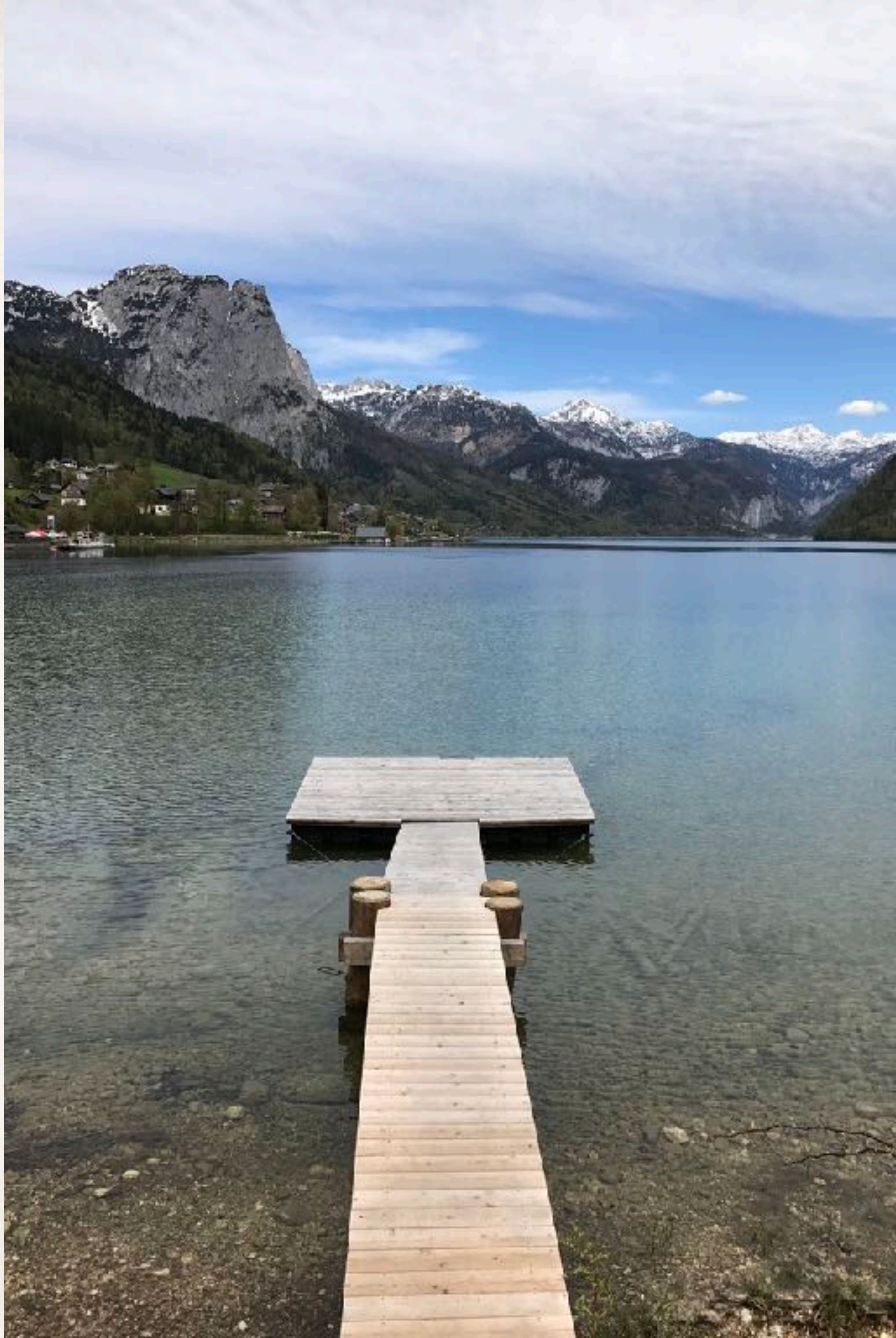
integer arithmetics
pointer arithmetics

no bitwise operators
no Boolean operators

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

Scarcity versus Abundance

If you want structs implement them!



```
> make
cc -w -m32 -D'main(a,b)=main(a,char**argv)' selfie.c -o selfie
```

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

(now also available for RISC-V machines)

```
> ./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 mallocated 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 MIPSter executable*

and

then running that MIPSter 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 MIPSter executable* `selfie1.m`

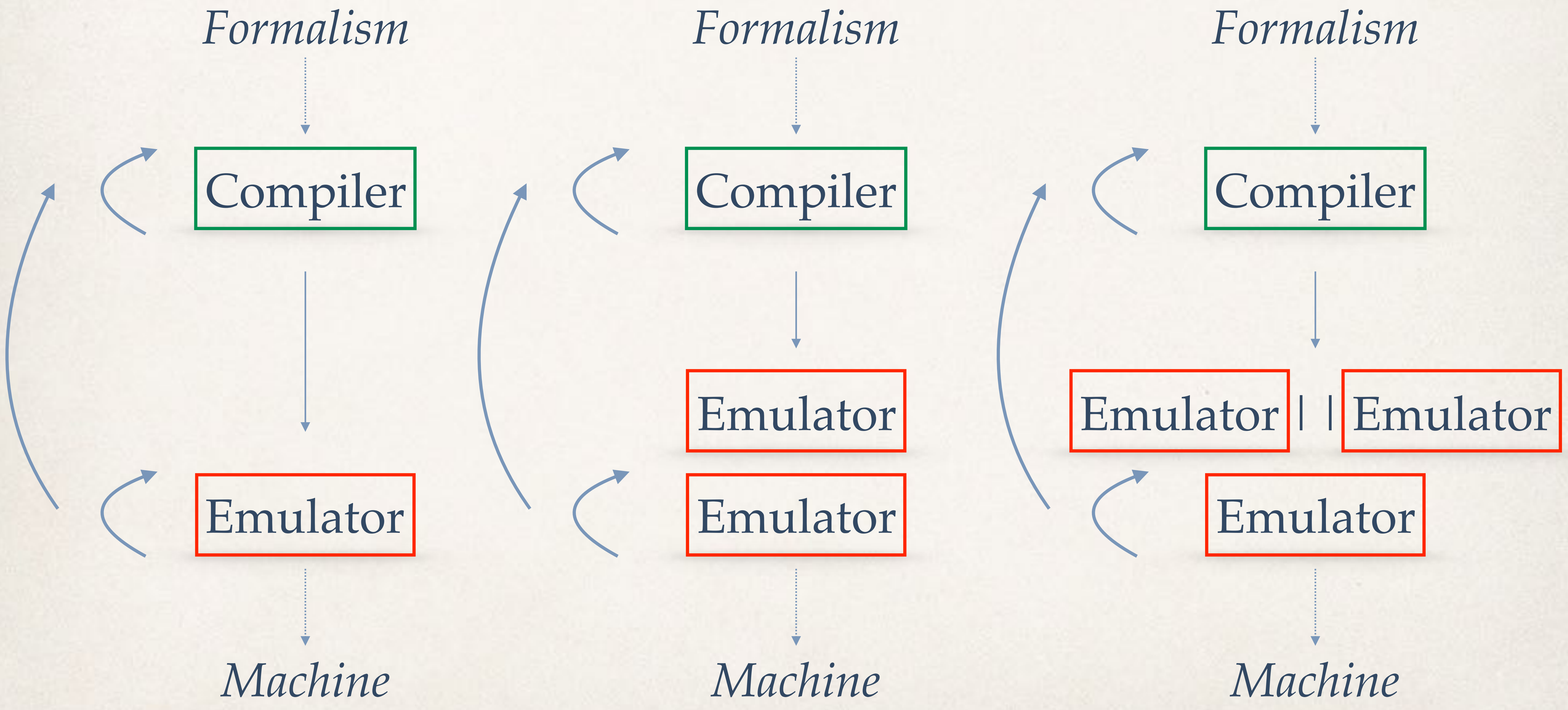
and

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

into another MIPSter executable `selfie2.m`

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

Sandboxed Concurrency: 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

and

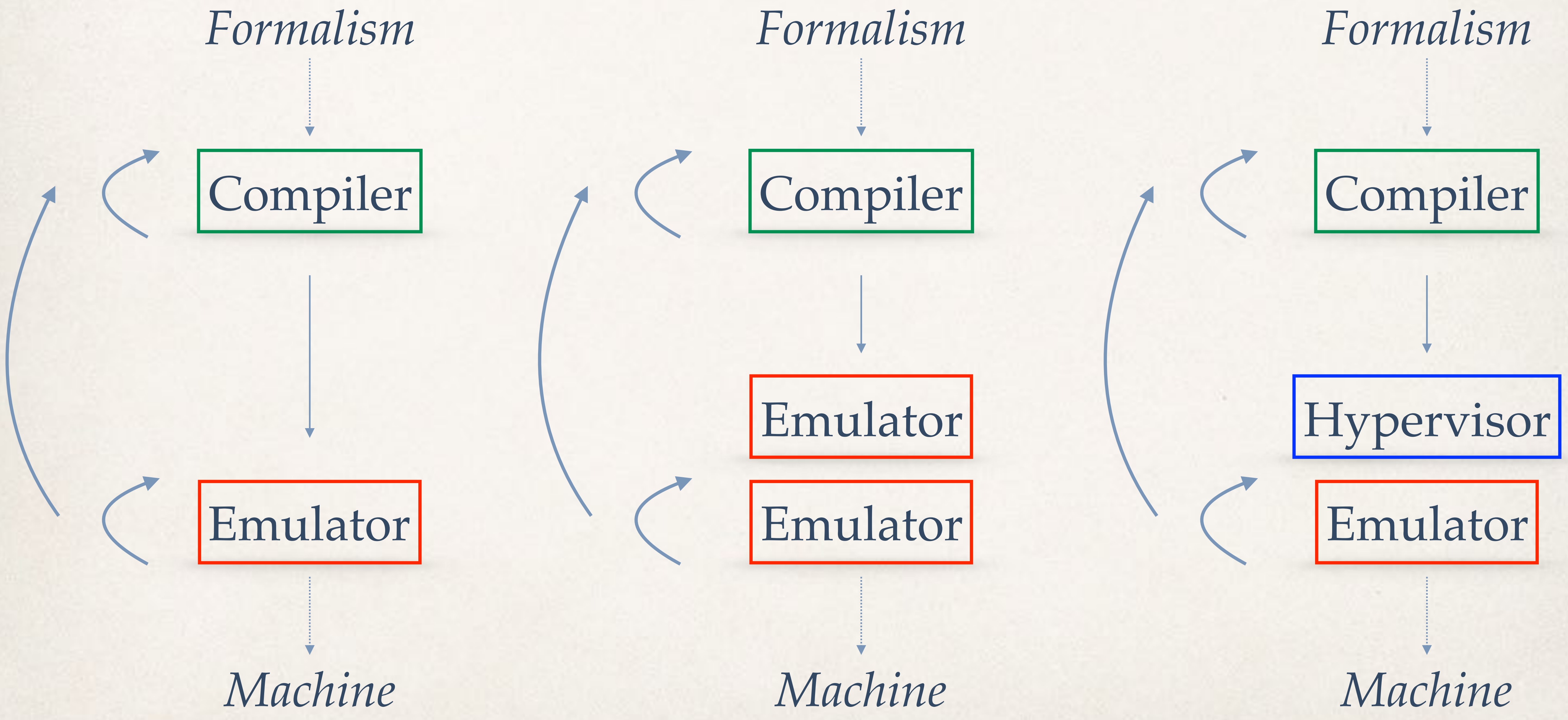
then running that executable to compile `selfie.c` *again*

and

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*

<u>and</u>

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

<u>and</u>

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

(takes ~12 minutes)

"How do we introduce self-model-checking and maybe even self-verification into Selfie?"

https://github.com/cksystemsteaching/selfie/tree/vipster

SMT Solver

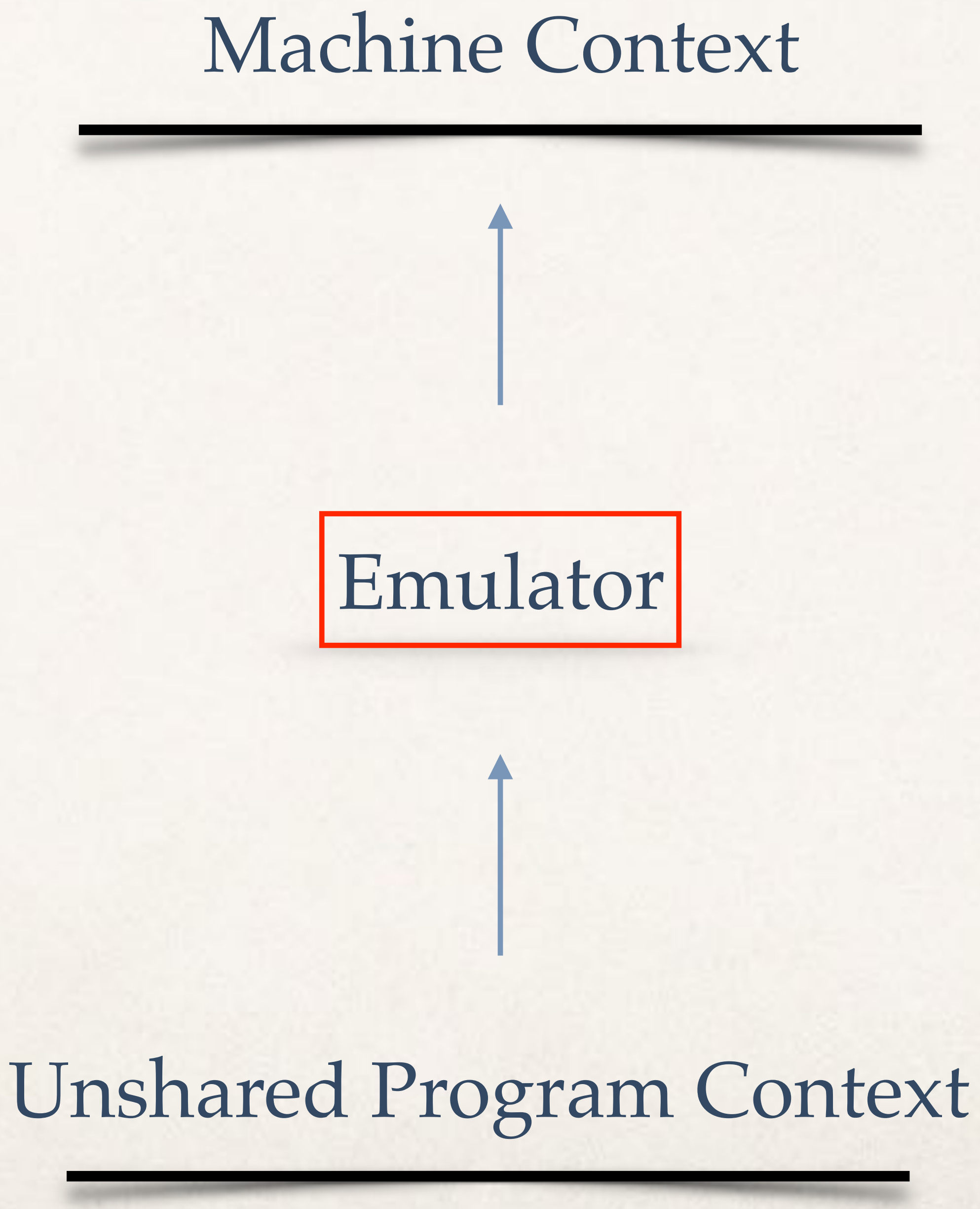
SAT Solver

What is the absolute simplest way of proving non-trivial properties of Selfie using Selfie?

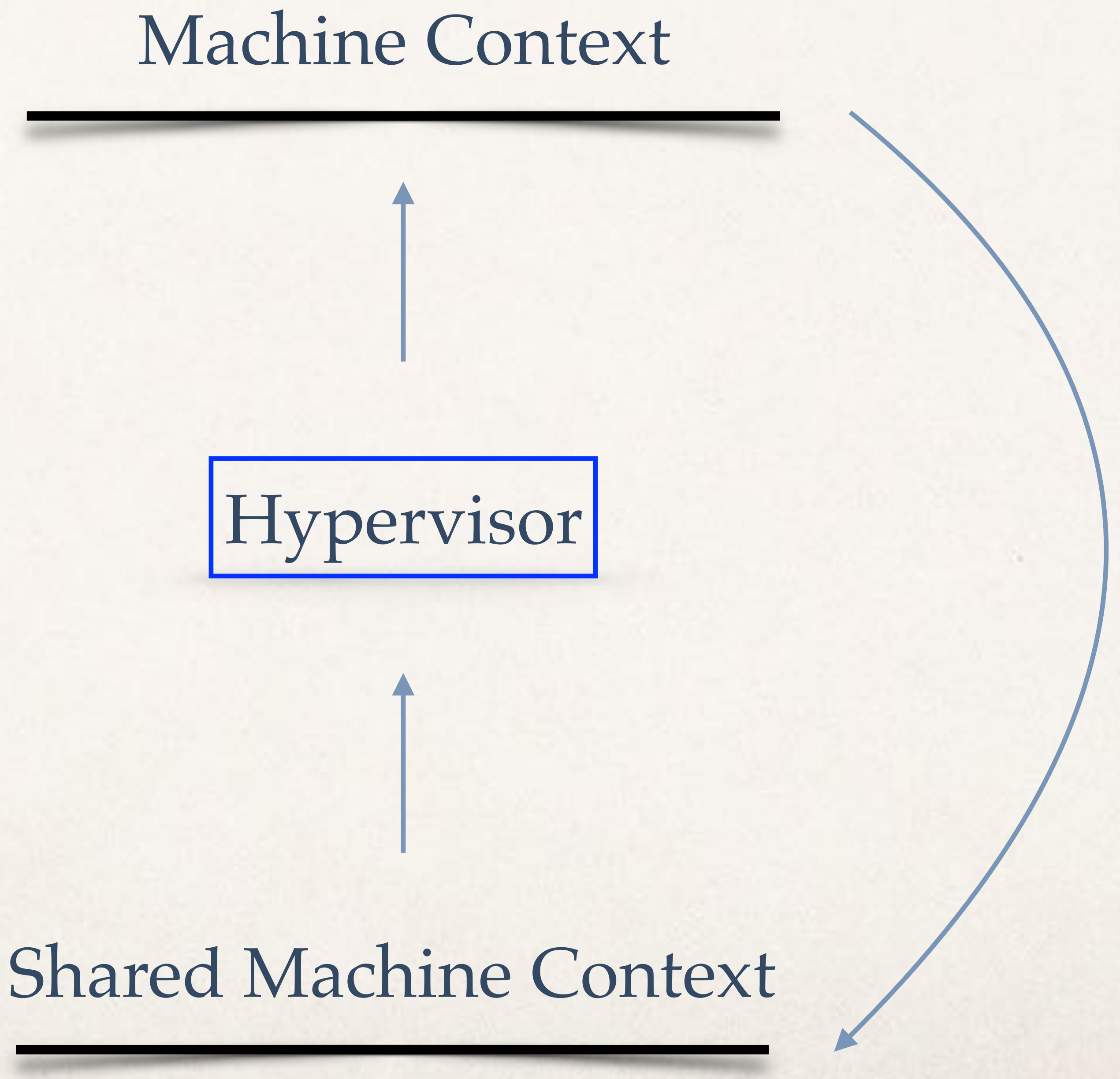
Bounded Model Checker

Inductive Theorem Prover

Emulation



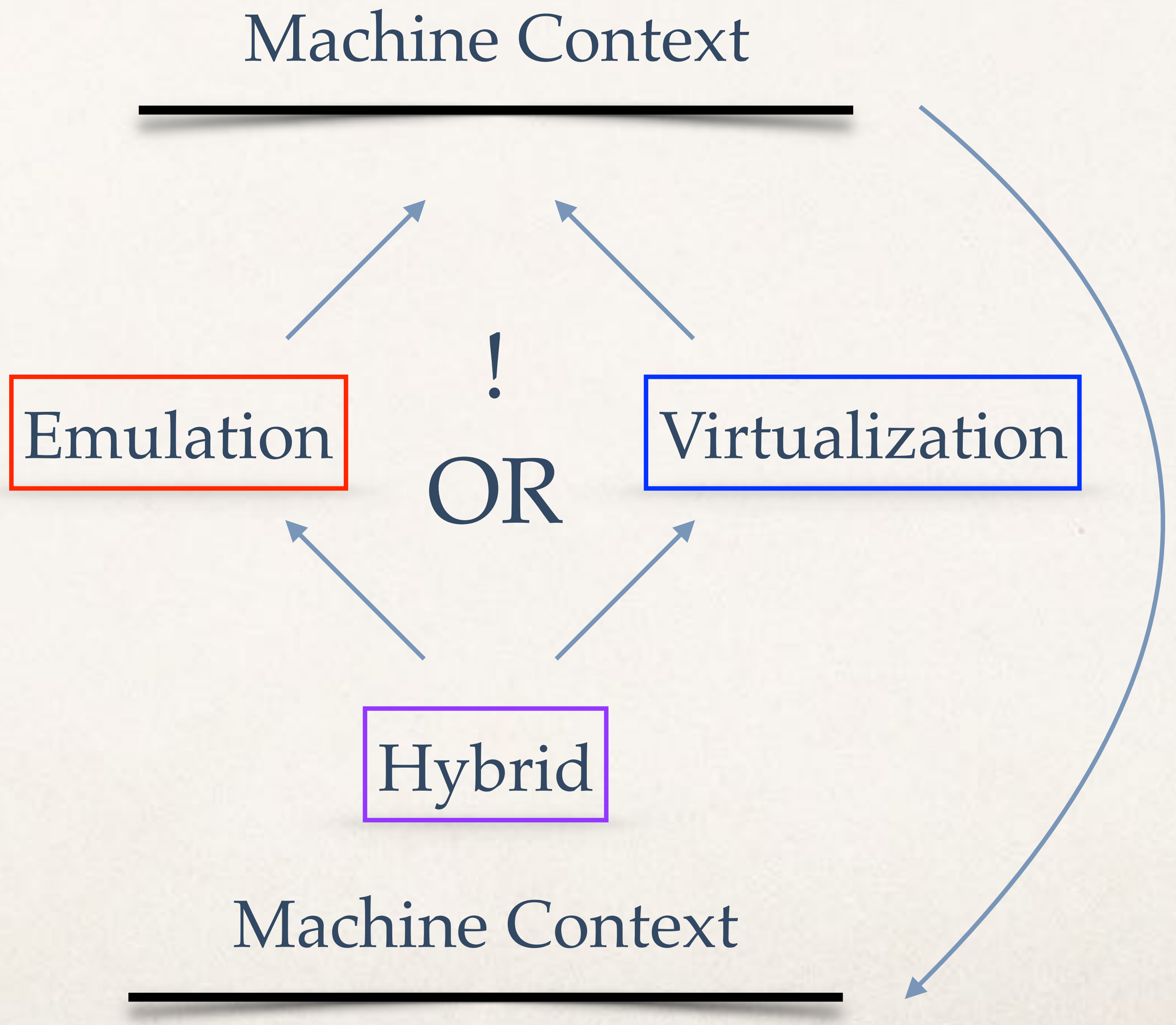
Virtualization



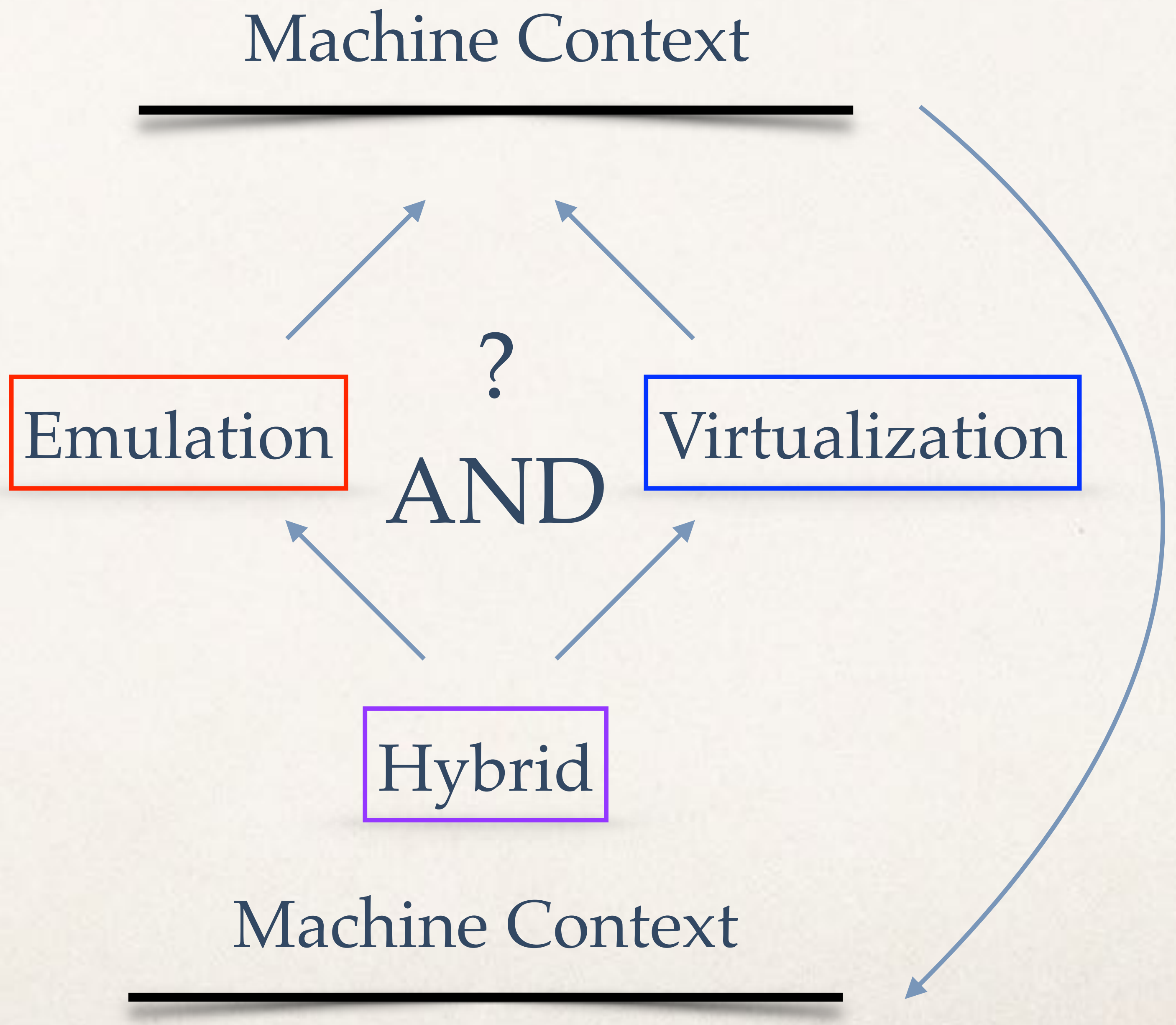
Proof Obligation

$$\frac{\text{Machine Context}}{\boxed{\text{Emulator}}} \overset{?}{=} \frac{\text{Machine Context}}{\boxed{\text{Hypervisor}}}$$

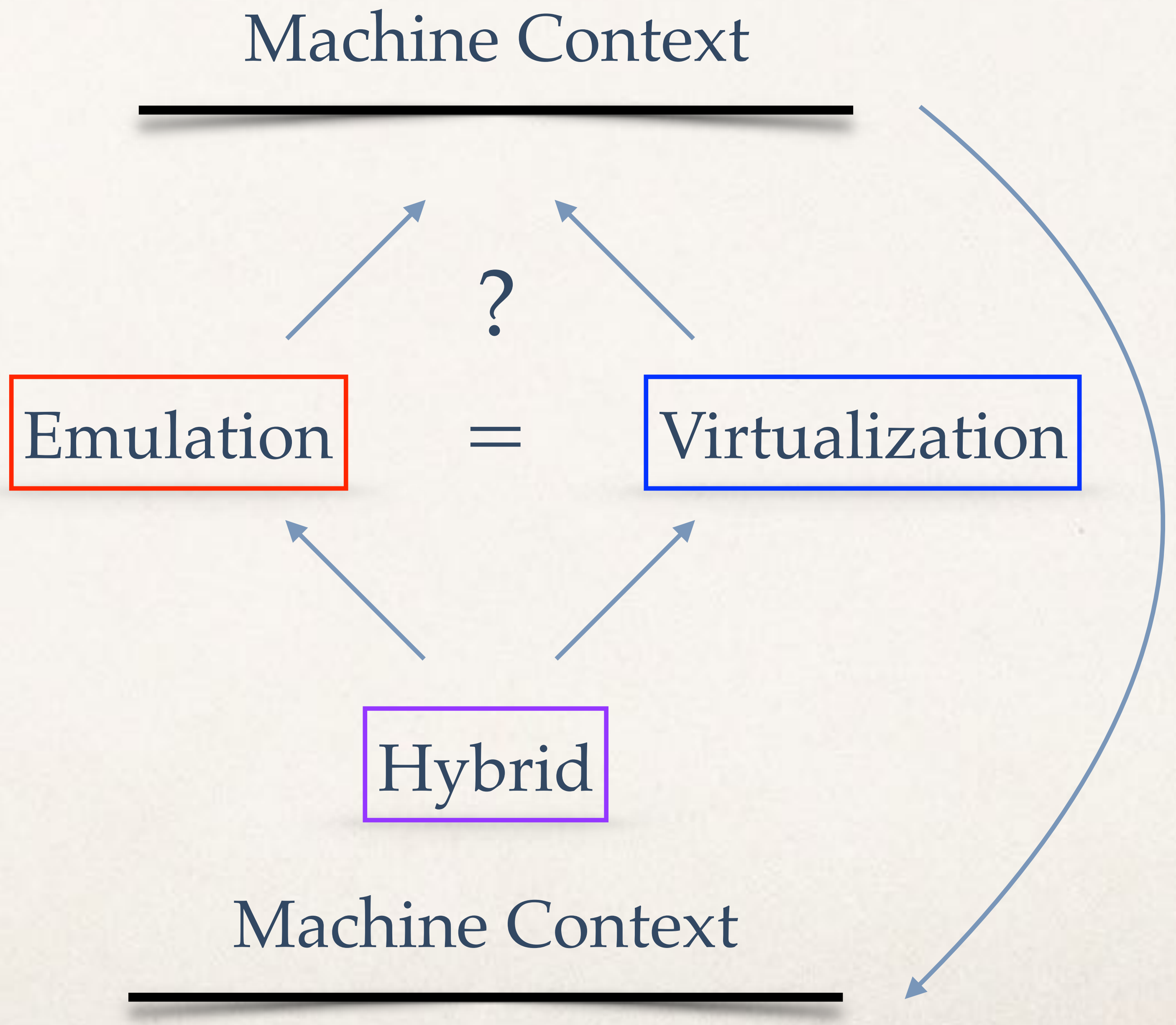
Hybrid of Emulator & Hypervisor



Validation of Functional Equivalence?



Verification of Functional Equivalence?



Questions

- What are the <u>benefits</u> of the hybrid design in Selfie?
- Will these benefits change the design of real kernels, that is, is the hybrid design <u>realistic</u>?
- Can we develop C* into a <u>useful</u> specification language, cf. ACL2?
- Can we prove <u>interesting</u> properties with a, say, ~10k-line system?
- Will this help teaching <u>rigorous</u> systems and software engineering at bachelor level?
- Will this help identifying <u>basic principles</u> that can be taught to everyone?

Thank you!