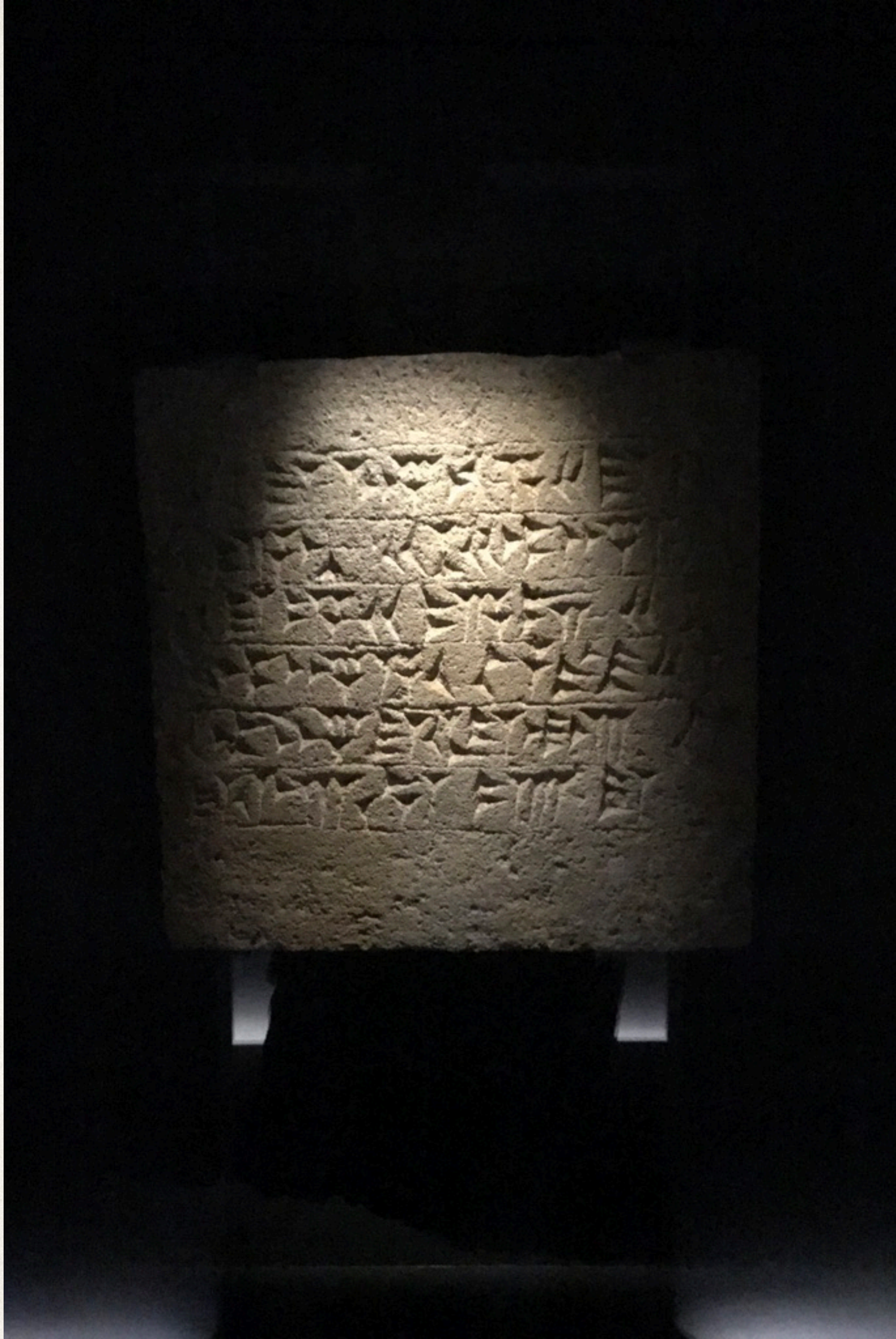
On the Self in Selfie

Christoph M. Kirsch

INESC-ID 2018 Lisbon, Portugal, October 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
- Hennessy/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

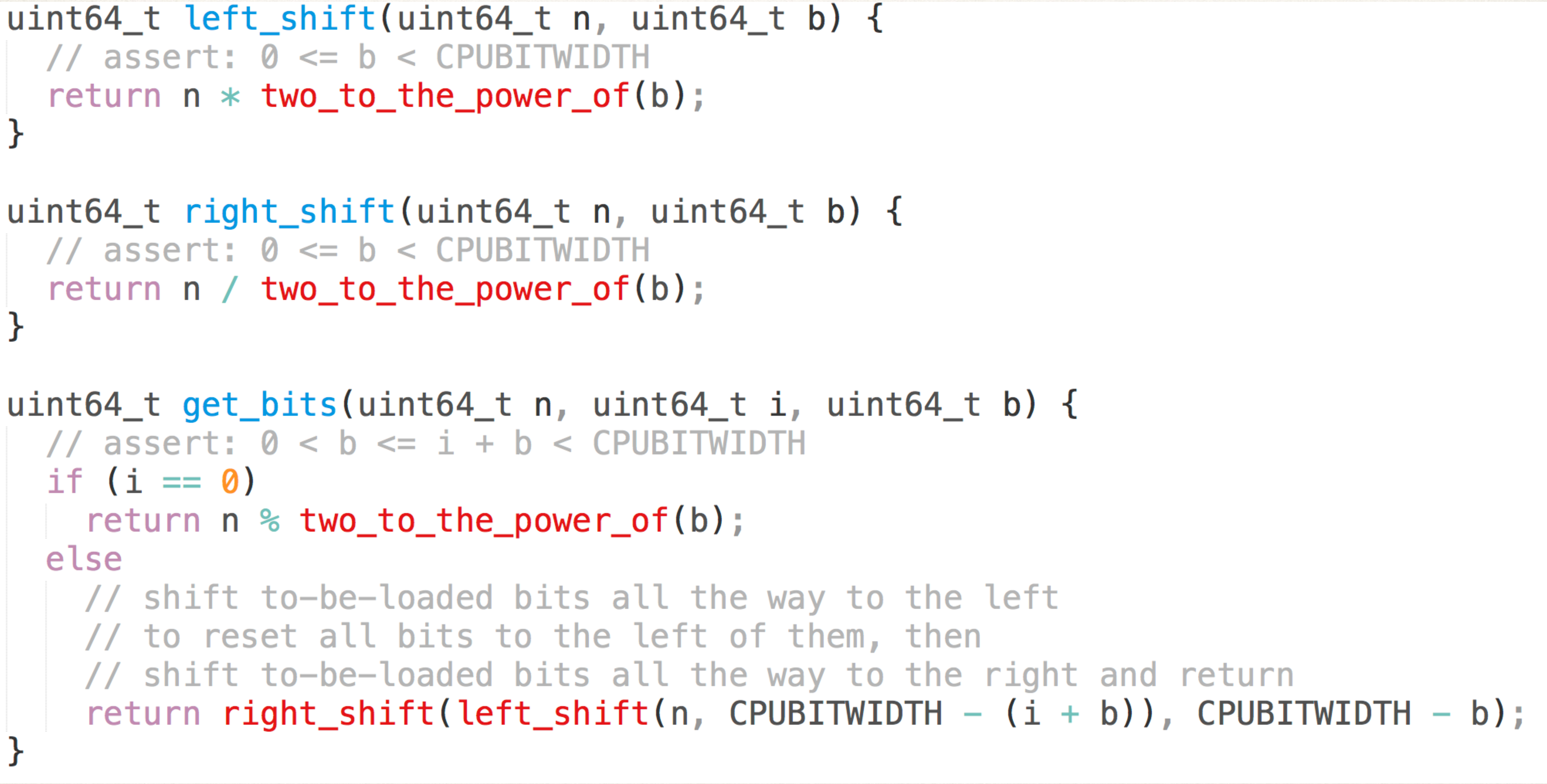
    6. a tiny, experimental SAT solver called ***babysat***.

Selfie supports the official 64-bit RISC-V toolchain and runs on the spike emulator and the pk kernel

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)

Code as Prose



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

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

uint64_t get_bits(uint64_t n, uint64_t i, uint64_t b) {
  // assert: 0 < b <= i + b < CPUBITWIDTH
  if (i == 0)
    return n % two_to_the_power_of(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 right_shift(left_shift(n, CPUBITWIDTH - (i + b)), CPUBITWIDTH - b);
}
```

Discussion of Selfie reached 3rd place on Hacker News

news.ycombinator.com

Website

selfie.cs.uni-salzburg.at

Code

github.com/cksystemsteaching/selfie

Slides (250 done, ~200 todo)

selfie.cs.uni-salzburg.at/slides

Book (draft)

leanpub.com/selfie

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

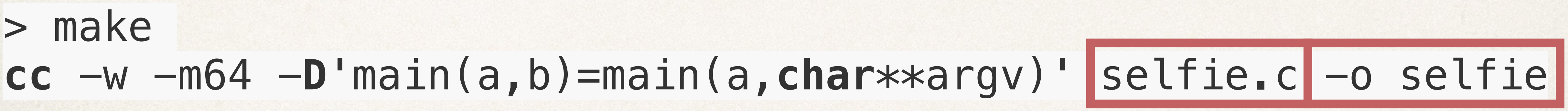
integer arithmetics
pointer arithmetics

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 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 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**)*

```
> ./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)

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

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

and

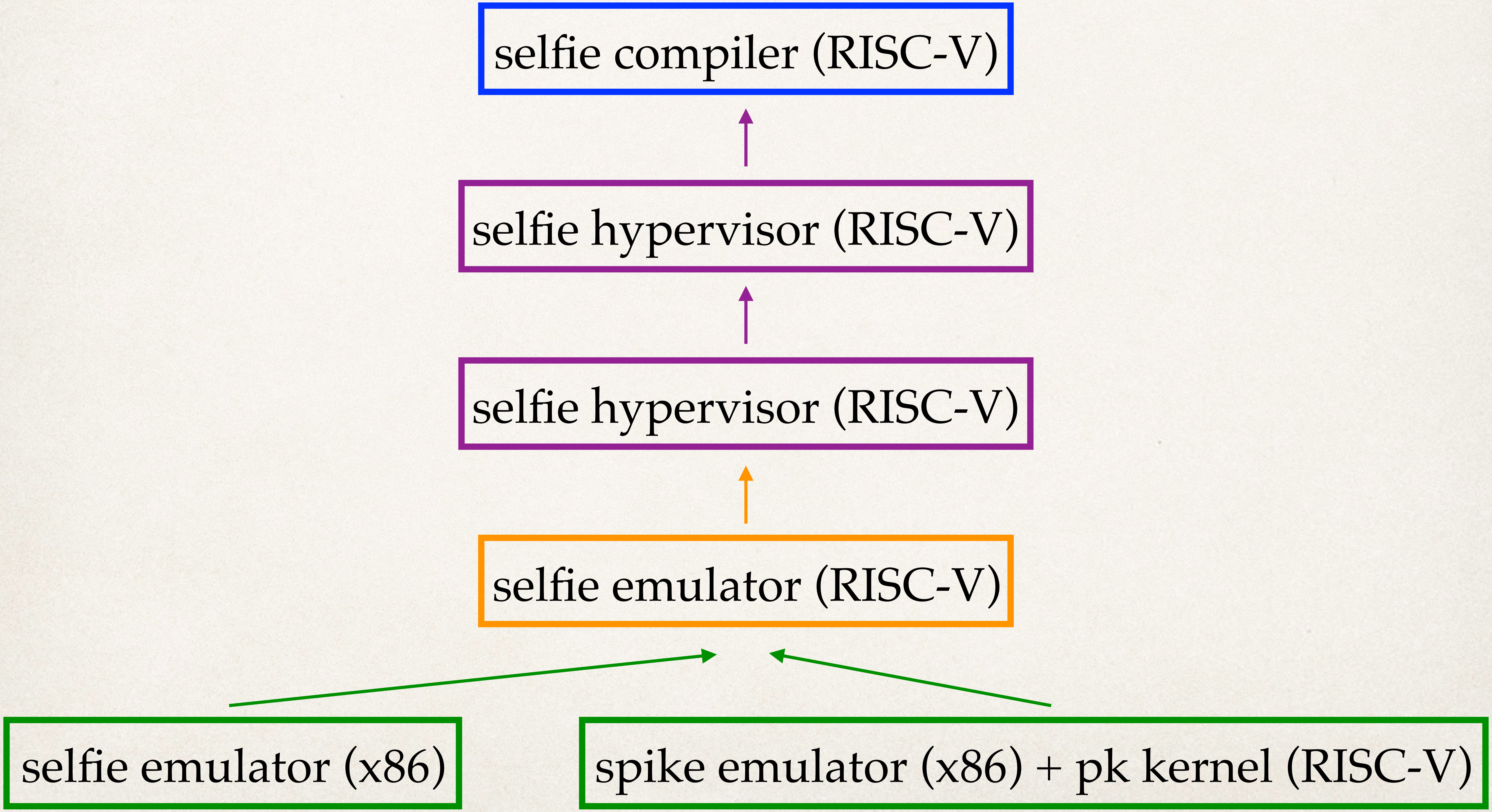
then running that executable to compile `selfie.c` *again*

and

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

(takes ~12 minutes)

Now That's a Selfie!



selfie compiler (RISC-V)

selfie hypervisor (RISC-V)

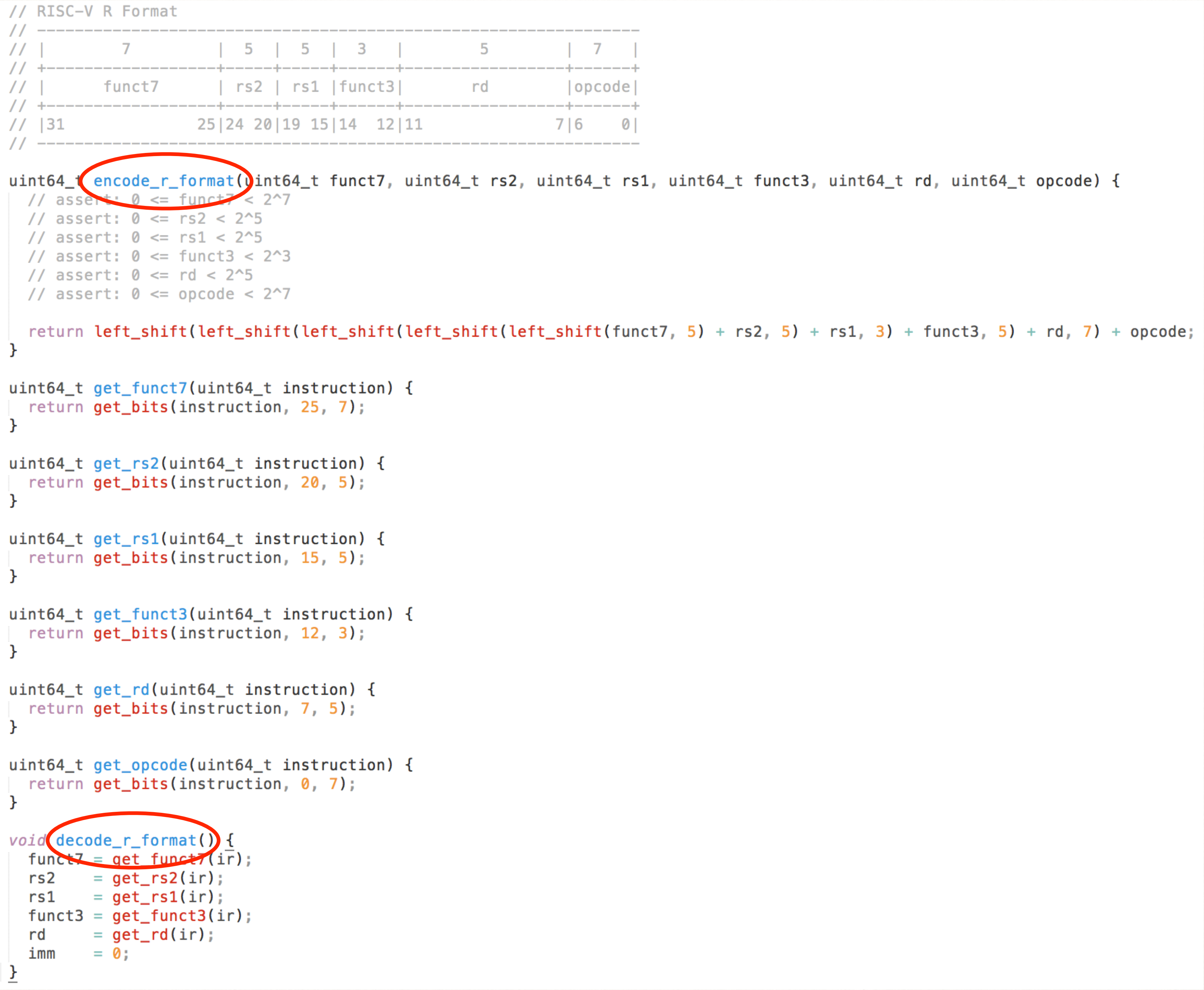
selfie hypervisor (RISC-V)

selfie emulator (RISC-V)

selfie emulator (x86)

spike emulator (x86) + pk kernel (RISC-V)

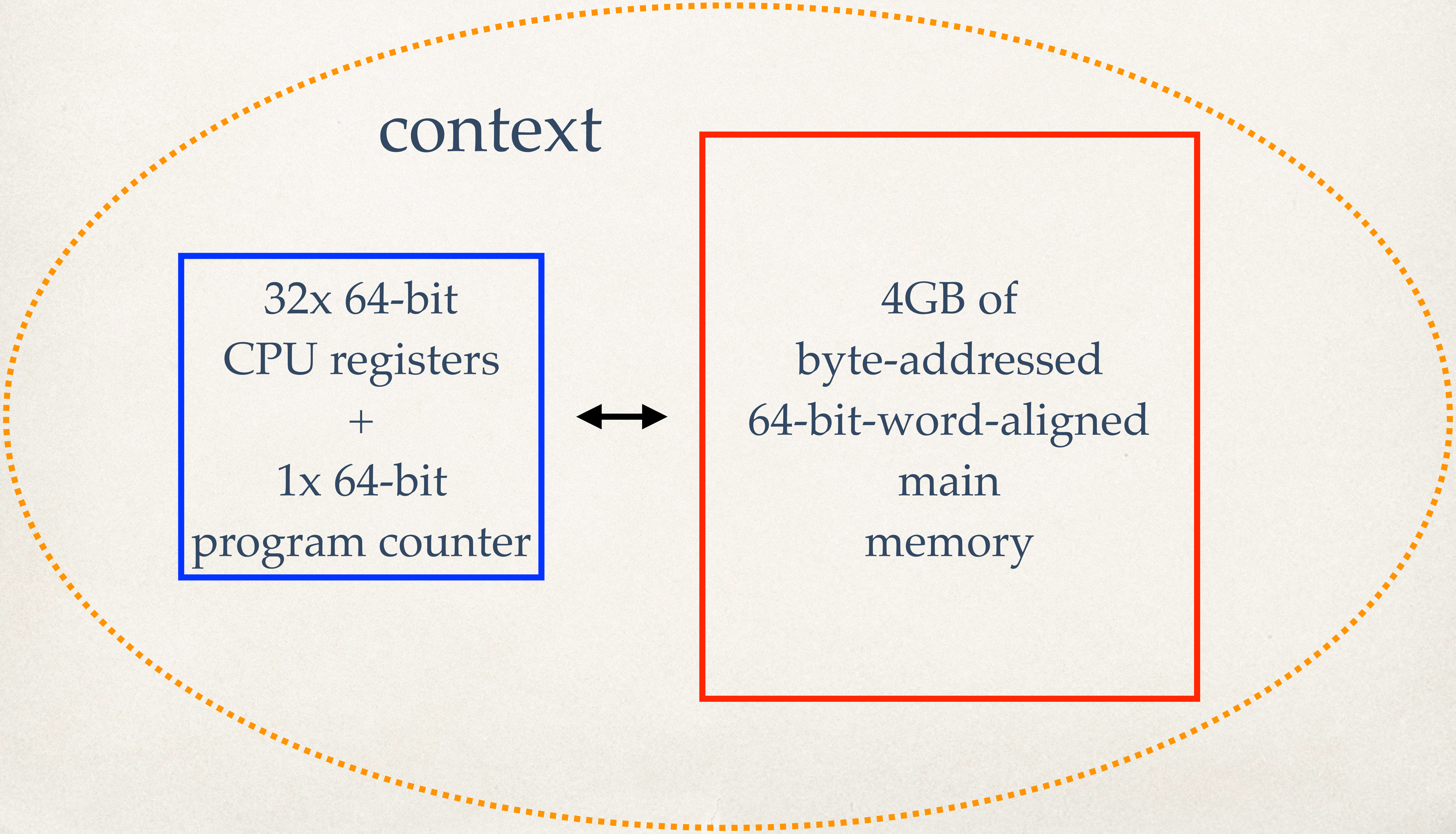
Self-Execution: works out of the box!



```
// RISC-V R Format
// -----------------------------------------------------------------
// |        7         |  5  |  5  |  3   |        5         |  7   |
// +------------------+-----+-----+------+------------------+------+
// |      funct7      | rs2 | rs1 |funct3|       rd         |opcode|
// +------------------+-----+-----+------+------------------+------+
// |31              25|24 20|19 15|14  12|11               7|6    0|
// -----------------------------------------------------------------

uint64_t encode_r_format(uint64_t funct7, uint64_t rs2, uint64_t rs1, uint64_t funct3, uint64_t rd, uint64_t opcode) {
  // assert: 0 <= funct7 < 2^7
  // assert: 0 <= rs2 < 2^5
  // assert: 0 <= rs1 < 2^5
  // assert: 0 <= funct3 < 2^3
  // assert: 0 <= rd < 2^5
  // assert: 0 <= opcode < 2^7

  return left_shift(left_shift(left_shift(left_shift(left_shift(funct7, 5) + rs2, 5) + rs1, 3) + funct3, 5) + rd, 7) + opcode;
}

uint64_t get_funct7(uint64_t instruction) {
  return get_bits(instruction, 25, 7);
}

uint64_t get_rs2(uint64_t instruction) {
  return get_bits(instruction, 20, 5);
}

uint64_t get_rs1(uint64_t instruction) {
  return get_bits(instruction, 15, 5);
}

uint64_t get_funct3(uint64_t instruction) {
  return get_bits(instruction, 12, 3);
}

uint64_t get_rd(uint64_t instruction) {
  return get_bits(instruction, 7, 5);
}

uint64_t get_opcode(uint64_t instruction) {
  return get_bits(instruction, 0, 7);
}

void decode_r_format() {
  funct7 = get_funct7(ir);
  rs2    = get_rs2(ir);
  rs1    = get_rs1(ir);
  funct3 = get_funct3(ir);
  rd     = get_rd(ir);
  imm    = 0;
}
```

synergy with compiler
in the same file
is still surprisingly
cool!

RISC-U Machine State

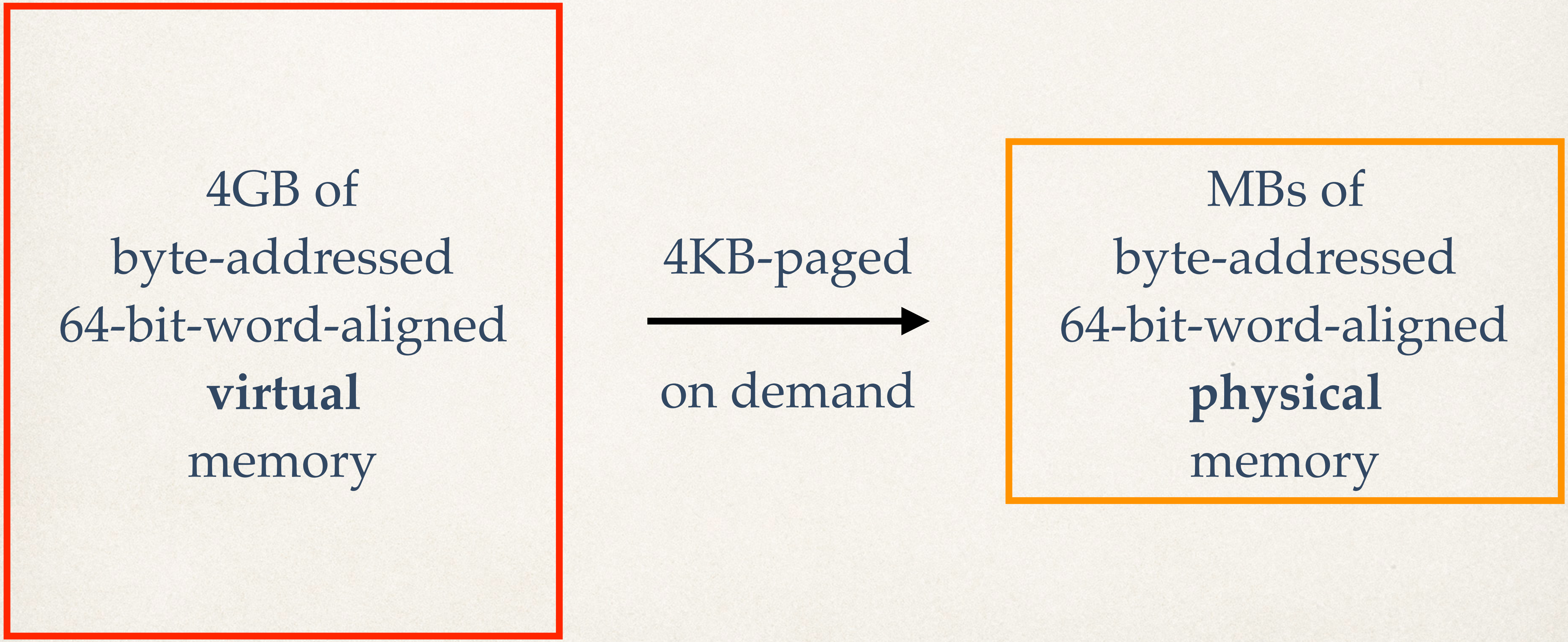


context

32x 64-bit
CPU registers
+
1x 64-bit
program counter

4GB of
byte-addressed
64-bit-word-aligned
main
memory

Virtual Memory in Selfie



4GB of
byte-addressed
64-bit-word-aligned
virtual
memory

4KB-paged

on demand

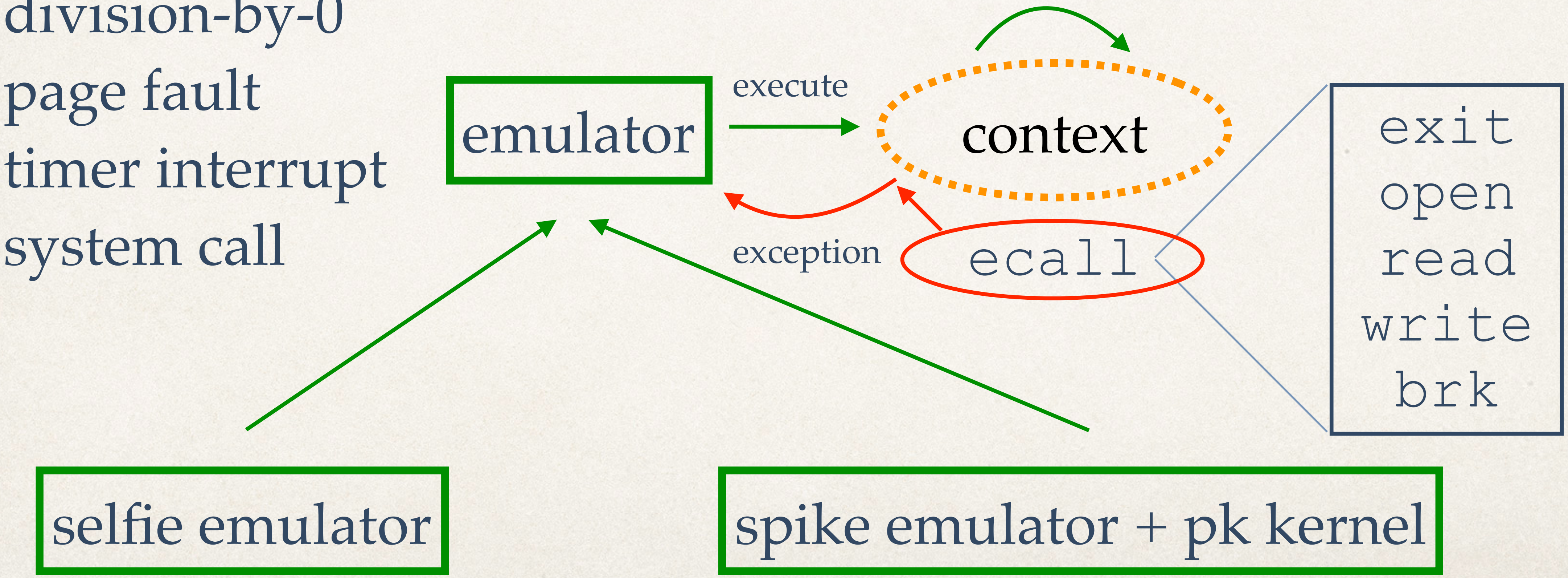
MBs of
byte-addressed
64-bit-word-aligned
physical
memory

Code Execution and Exceptions

13+1 instructions:

lui addi
add sub mul
divu remu
ld sd
sltu beq
jal jalr

1. division-by-0
2. page fault
3. timer interrupt
4. system call



emulator

execute

context

exception

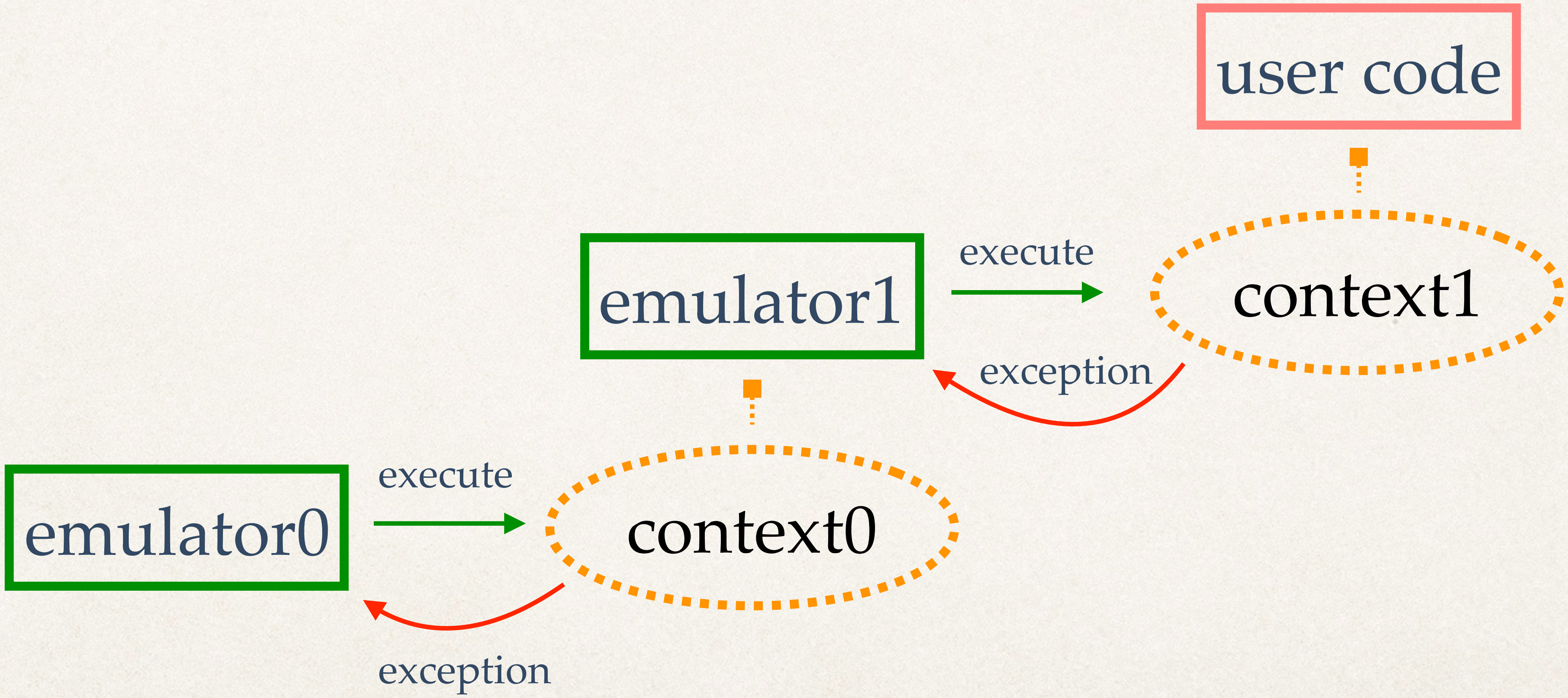
ecall

exit
open
read
write
brk

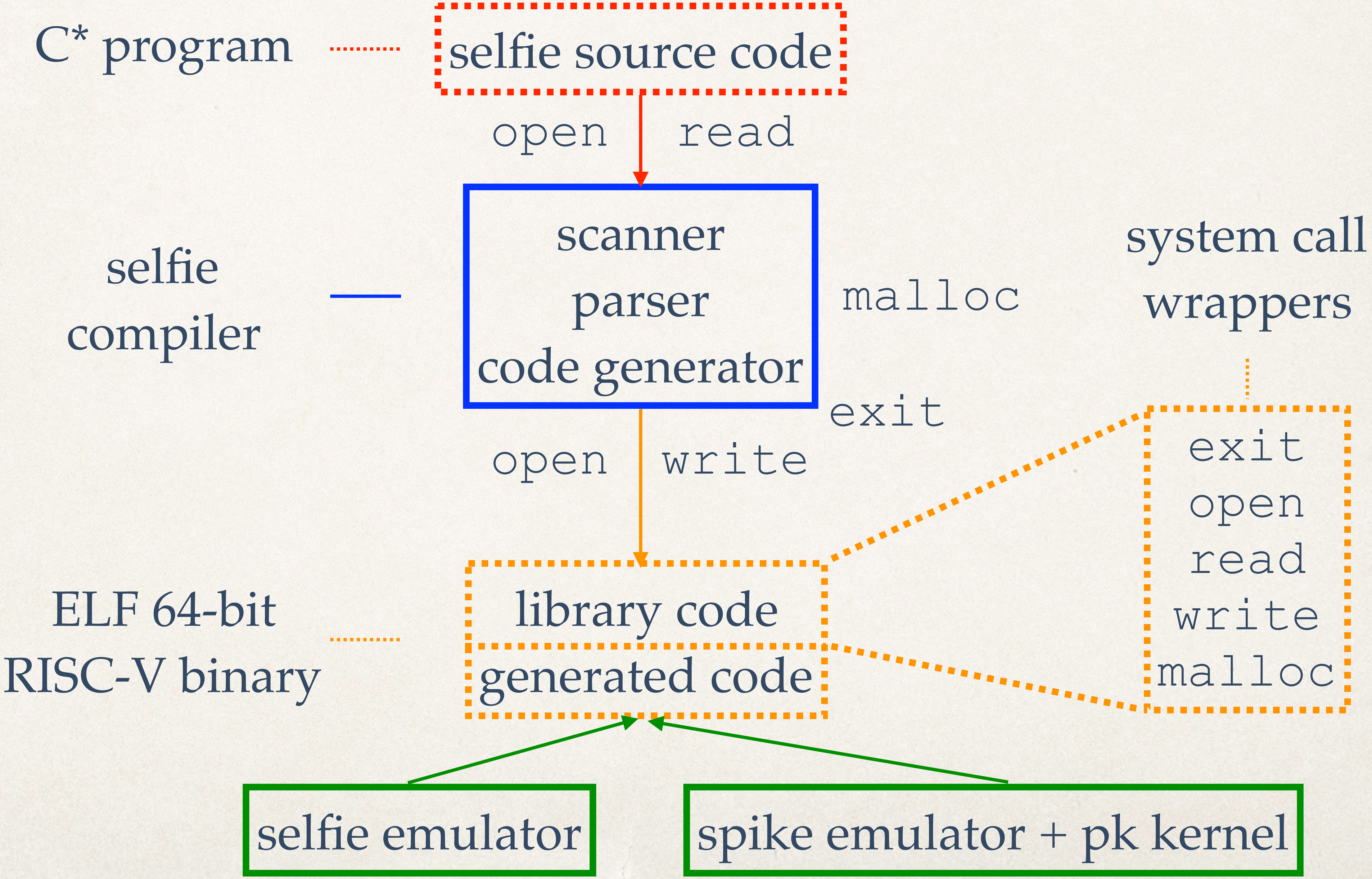
selfie emulator

spike emulator + pk kernel

Self-Execution



Self-Compilation



C* program

selfie source code

open read

selfie compiler

scanner
parser
code generator

malloc

system call wrappers

exit

open write

exit
open
read
write
malloc

ELF 64-bit RISC-V binary

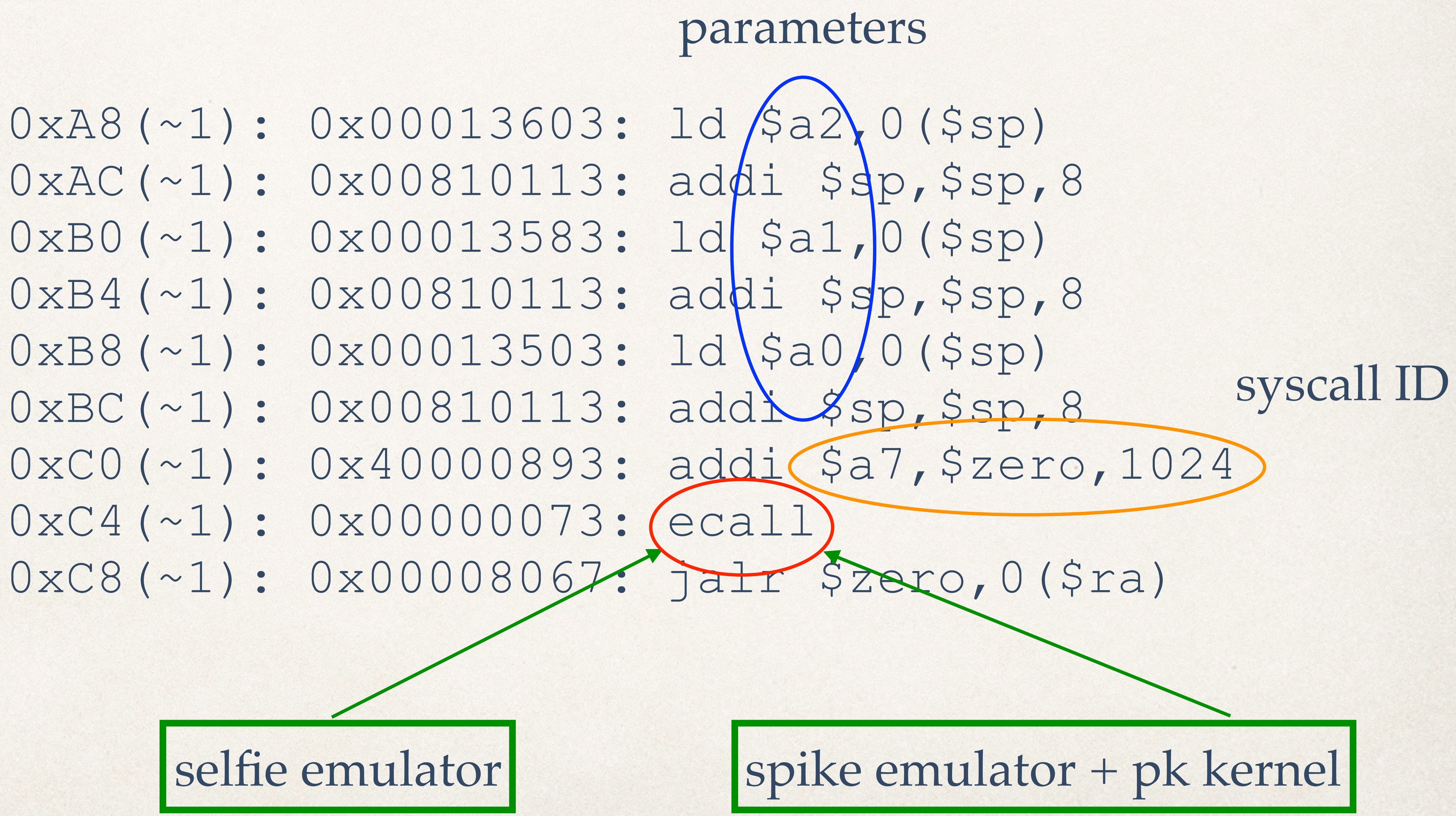
library code

generated code

selfie emulator

spike emulator + pk kernel

Library Code: open wrapper



parameters

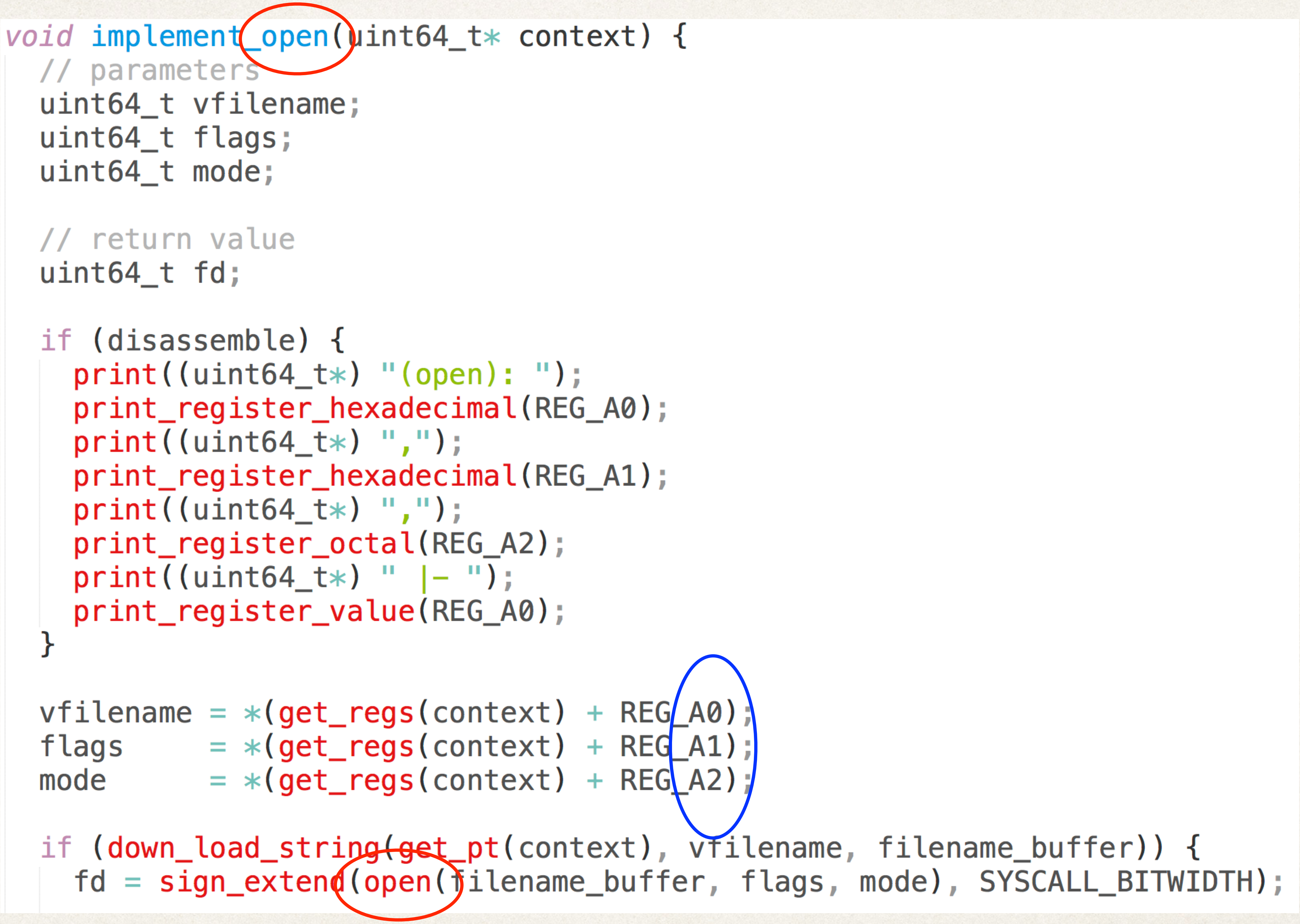
```
0xA8(~1): 0x00013603: ld $a2,0($sp)
0xAC(~1): 0x00810113: addi $sp,$sp,8
0xB0(~1): 0x00013583: ld $a1,0($sp)
0xB4(~1): 0x00810113: addi $sp,$sp,8
0xB8(~1): 0x00013503: ld $a0,0($sp)
0xBC(~1): 0x00810113: addi $sp,$sp,8
0xC0(~1): 0x40000893: addi $a7,$zero,1024
0xC4(~1): 0x00000073: ecall
0xC8(~1): 0x00008067: jalr $zero,0($ra)
```

syscall ID

selfie emulator

spike emulator + pk kernel

open implementation in selfie emulator



```
void implement_open(uint64_t* context) {
  // parameters
  uint64_t vfilename;
  uint64_t flags;
  uint64_t mode;

  // return value
  uint64_t fd;

  if (disassemble) {
    print((uint64_t*) "(open): ");
    print_register_hexadecimal(REG_A0);
    print((uint64_t*) ",");
    print_register_hexadecimal(REG_A1);
    print((uint64_t*) ",");
    print_register_octal(REG_A2);
    print((uint64_t*) " |- ");
    print_register_value(REG_A0);
  }

  vfilename = *(get_regs(context) + REG_A0);
  flags     = *(get_regs(context) + REG_A1);
  mode      = *(get_regs(context) + REG_A2);

  if (down_load_string(get_pt(context), vfilename, filename_buffer)) {
    fd = sign_extend(open(filename_buffer, flags, mode), SYSCALL_BITWIDTH);
```

selfie compiler

C library call

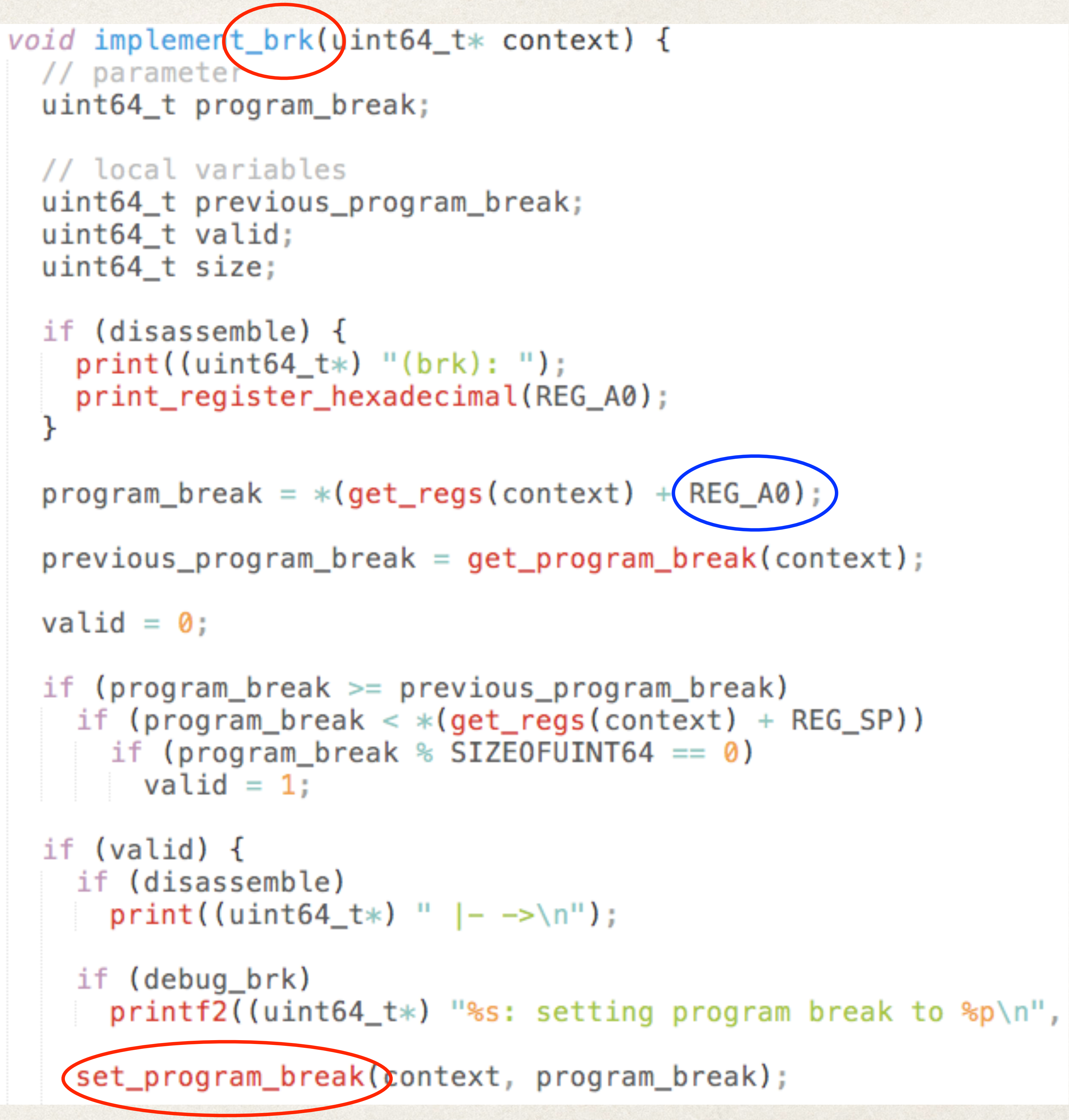
gcc / clang

`malloc` is different!

`malloc` invokes
the `brk` system call

both manage pure
address spaces

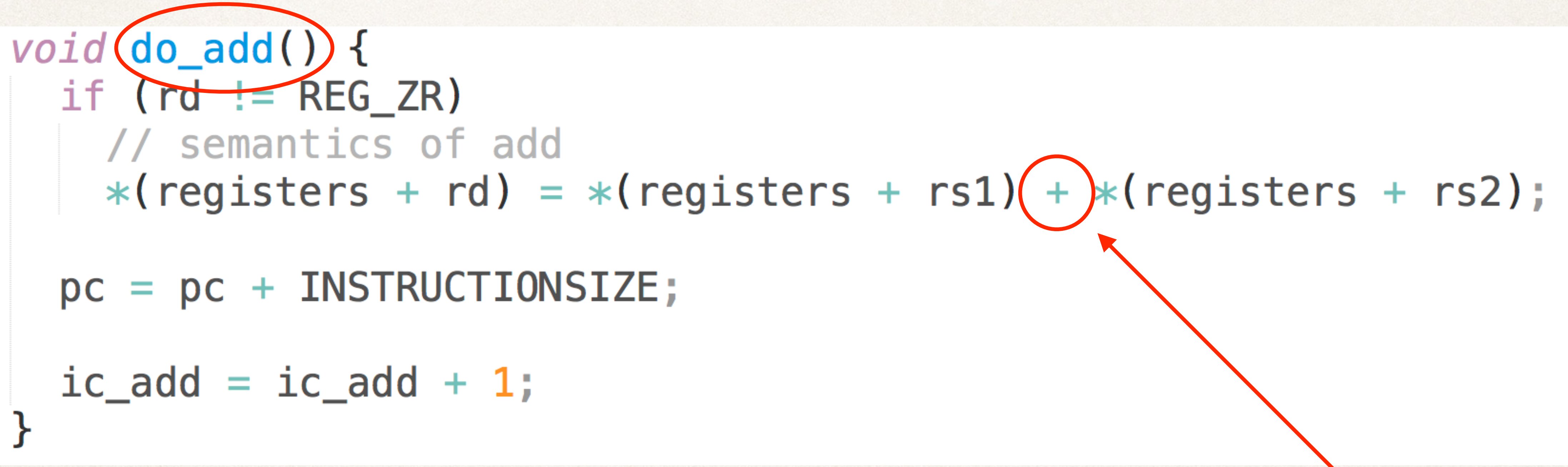
actual memory
storage is done
in the paging system



```
void implement_brk(uint64_t* context) {
  // parameter
  uint64_t program_break;

  // local variables
  uint64_t previous_program_break;
  uint64_t valid;
  uint64_t size;

  if (disassemble) {
    print((uint64_t*) "(brk): ");
    print_register_hexadecimal(REG_A0);
  }

  program_break = *(get_regs(context) + REG_A0);

  previous_program_break = get_program_break(context);

  valid = 0;

  if (program_break >= previous_program_break)
    if (program_break < *(get_regs(context) + REG_SP))
      if (program_break % SIZEOFUINT64 == 0)
        valid = 1;

  if (valid) {
    if (disassemble)
      print((uint64_t*) " |- ->\n");

    if (debug_brk)
      printf2((uint64_t*) "%s: setting program break to %p\n",

    set_program_break(context, program_break);
```

Generated Code: add and +

64-bit RISC-V add instruction



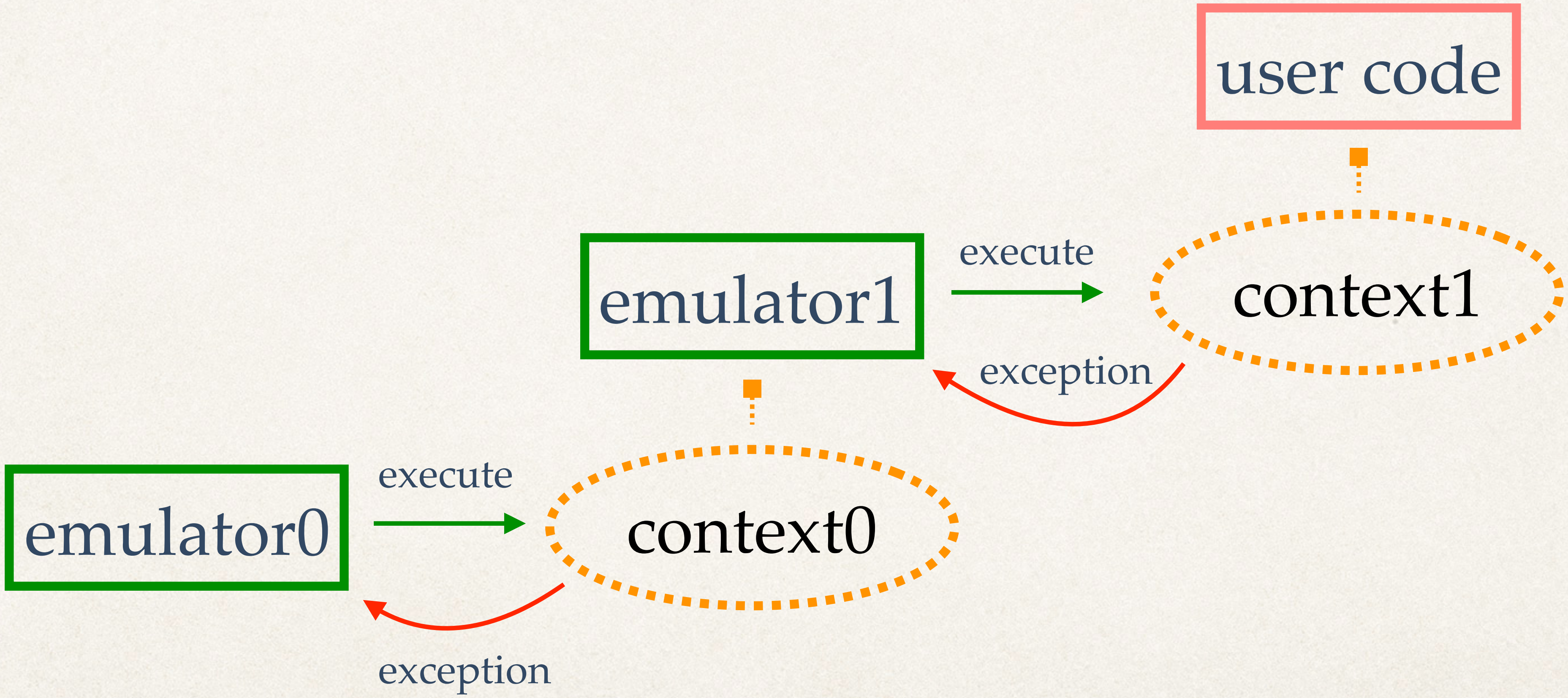
```
void do_add() {
  if (rd != REG_ZR)
    // semantics of add
    *(registers + rd) = *(registers + rs1) + *(registers + rs2);

  pc = pc + INSTRUCTIONSIZE;

  ic_add = ic_add + 1;
}
```

C code for unsigned 64-bit integer addition

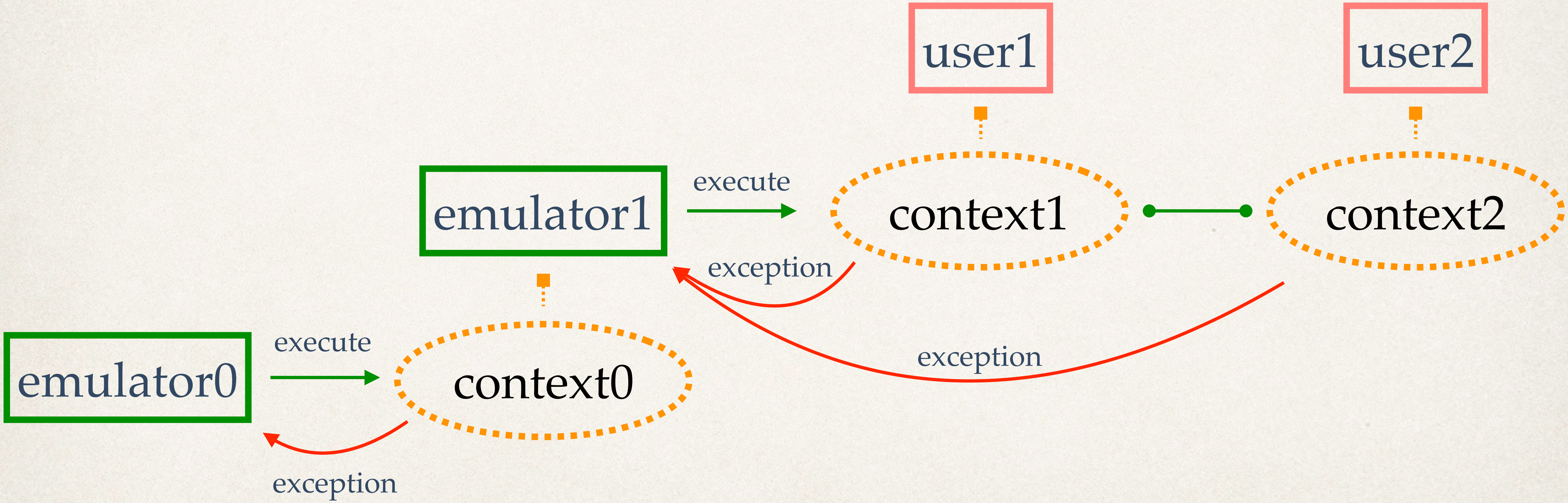
selfie compiler

gcc / clang

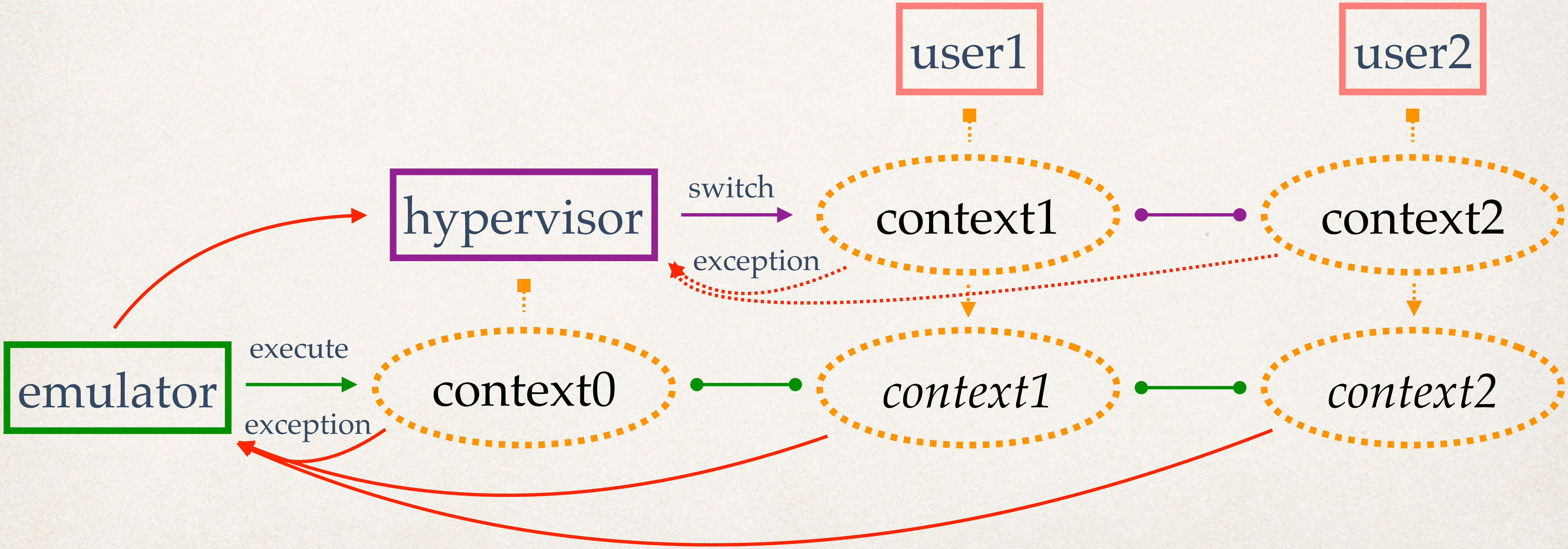
Self-Execution Revisited



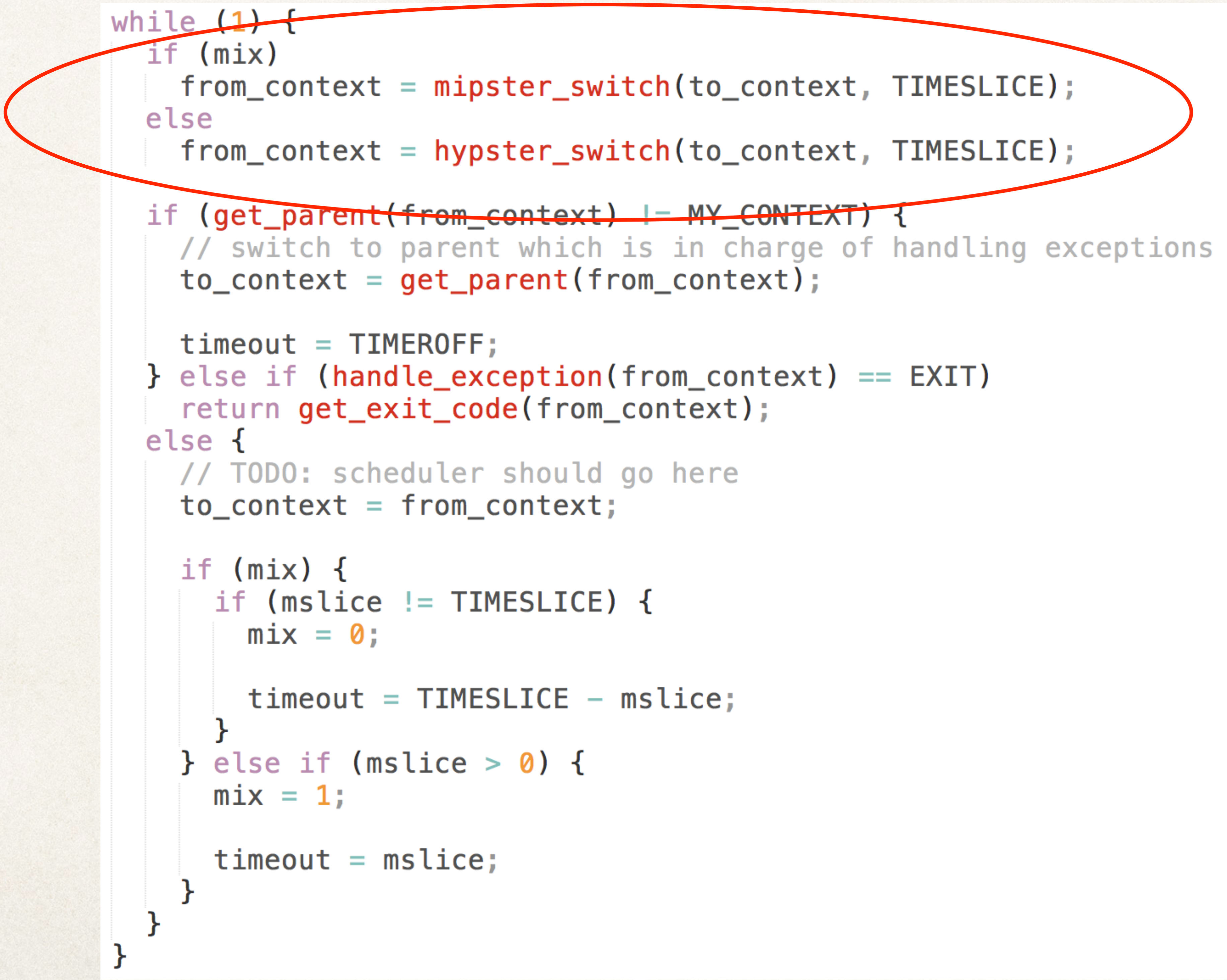
Self-Execution: Concurrency



Hosting: Concurrency

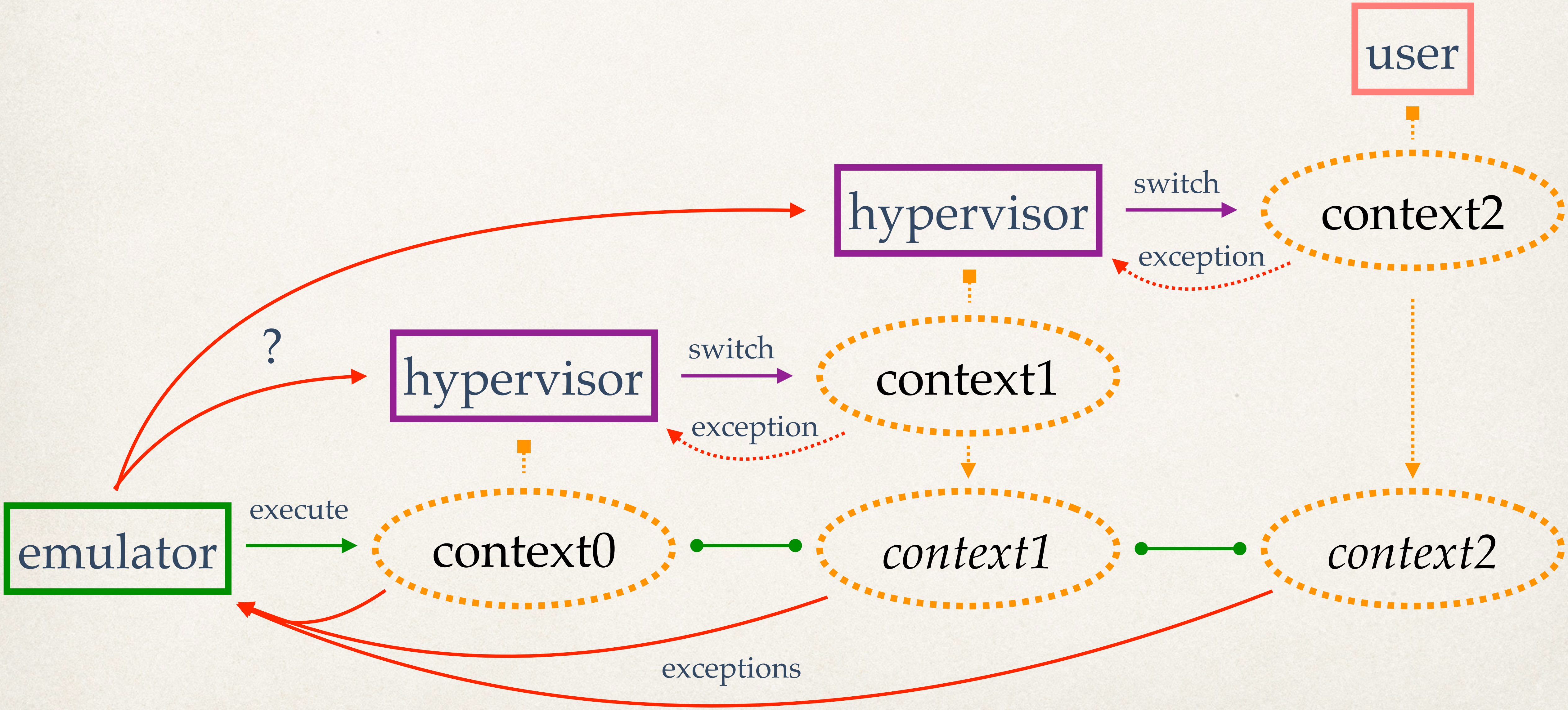


Emulation versus Virtualization



```
while (1) {
  if (mix)
    from_context = mipster_switch(to_context, TIMESLICE);
  else
    from_context = hypster_switch(to_context, TIMESLICE);

  if (get_parent(from_context) != MY_CONTEXT) {
    // switch to parent which is in charge of handling exceptions
    to_context = get_parent(from_context);

    timeout = TIMEROFF;
  } else if (handle_exception(from_context) == EXIT)
    return get_exit_code(from_context);
  else {
    // TODO: scheduler should go here
    to_context = from_context;

    if (mix) {
      if (mslice != TIMESLICE) {
        mix = 0;

        timeout = TIMESLICE - mslice;
      }
    } else if (mslice > 0) {
      mix = 1;

      timeout = mslice;
    }
  }
}
```

Self-Hosting: Hierarchy



user

hypervisor

switch

context2

exception

?

hypervisor

switch

context1

exception

emulator

execute

context0

context1

context2

exceptions

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

Minimal Symbolic Execution?

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

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

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!