How to Compile a Compiler

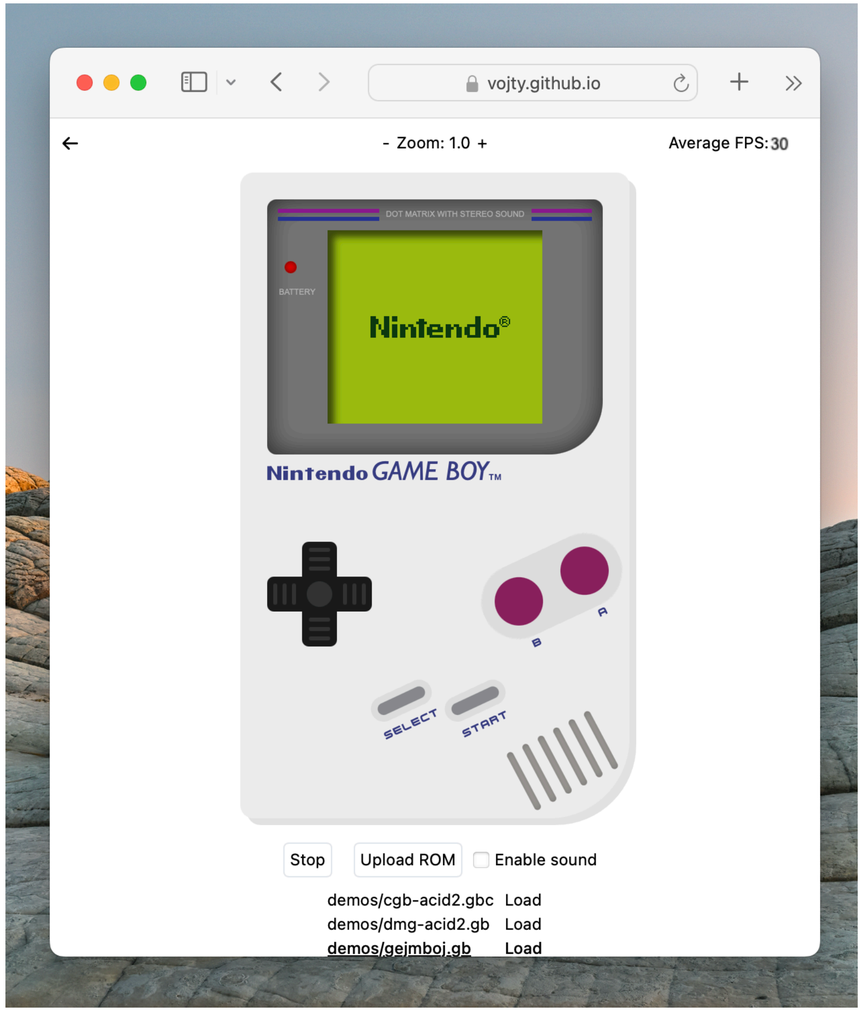
Producing Fast Full-system Emulators from Formal Specifications

Ferdia McKeogh, University of St Andrews

Background

Emulator

- “Run this ARM64 binary on an x86_64 machine”
- “Play this GameBoy ROM on my laptop”
- Useful:
  - Testing
  - Debugging
  - Legacy + Future



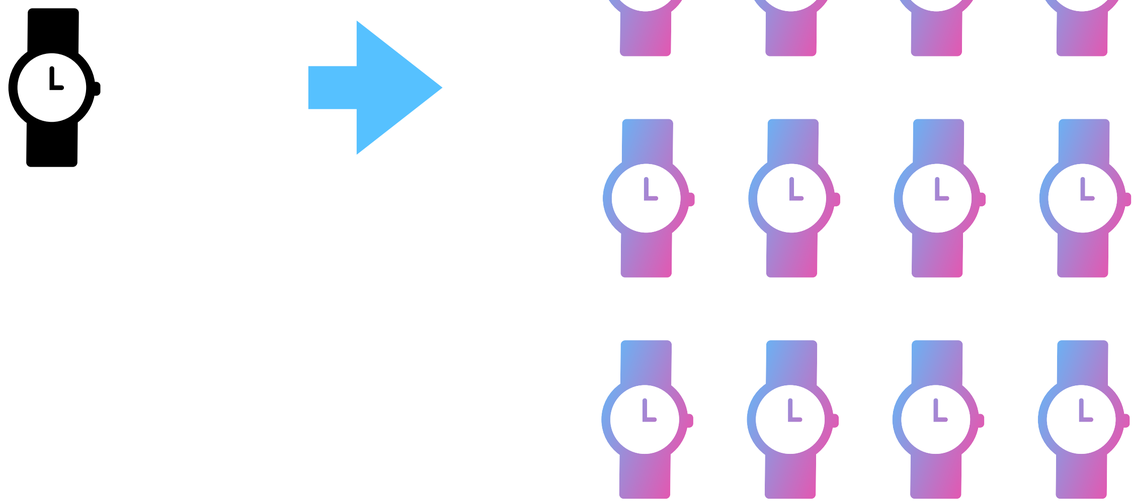
Problem

- Testing software for embedded devices is hard
  - Cross-compile and stub “real” things
  - Manual testing
  - Automatic testing

“Here's how I juggle embedded projects (home office/workspace tour)” - Jay Carlson

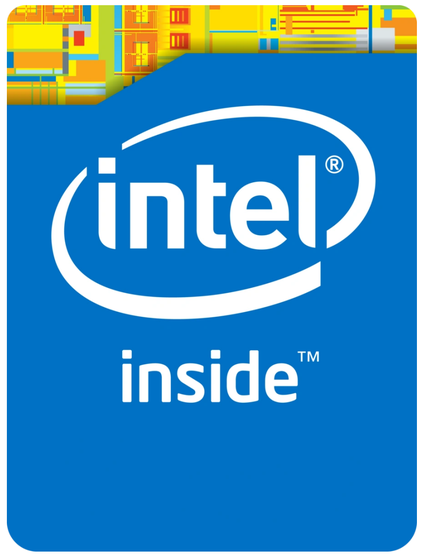
Solution

- Emulation
- Subject 1000s of emulated devices to different conditions
- Property-based + fuzz testing
- *real* unmodified firmware shipped to customers

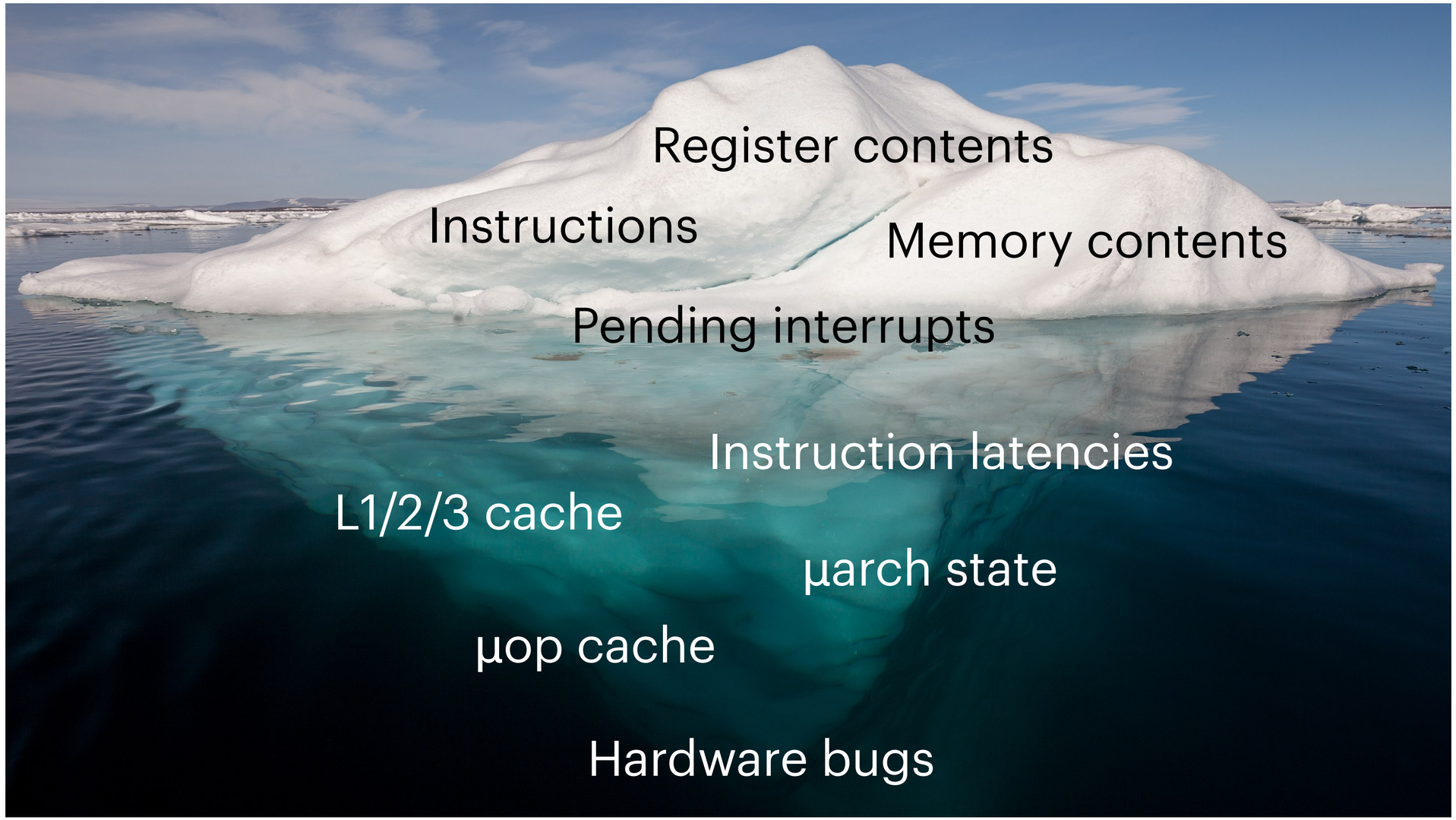


Instruction Set Architecture

- Interface between hardware and software
  - Instructions
    - How do you decode them?
    - How do you execute them?
  - Registers
  - Other misc. state and behaviours (exceptions, etc)



The ISA-berg



Andreas Weith, 2015

Sail

- Language for formally specifying an ISA
- Ergonomic, and expressive language
- ARM->Sail
- RISC-V
- Industry adoption!
- Slow*, 30 mins to boot Linux :(

```
register VBAR_EL1 : bits(64)

val get_VBAR_NS : unit -> bits(32) effect {rreg, undef}

function get_VBAR_NS () = {
    r : bits(32) = undefined : bits(32);
    let r = __SetSlice_bits(32, 32, r, 0, slice(VBAR_EL1, 0, 32));
    r
}
```

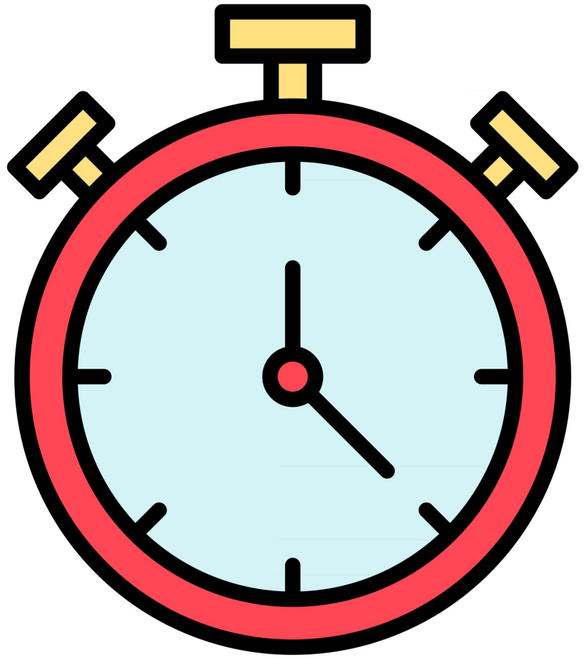
GenC

- C-like language
- GenSim toolchain, Captive DBT
- Very fast, boots Linux in seconds!
- Handwritten models:
  - Slow
  - Error prone

```
19  // 32-Base Instructions
20  execute(addi)
21  {
22          sint32 imm = inst.imm;
23          imm <<= 20;
24          imm >>= 20;
25
26
27          typename word_t rs = read_gpr(inst.rs1);
28
29          rs += (typename sword_t)imm;
30
31          write_register_bank(GPR, inst.rd, rs);
32
33  }
```

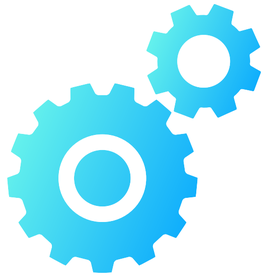
Why is Sail slow and GenC fast?

- Sail compiles to an interpreter:
  - Fetch, decode, execute
  - 1 guest instruction -> 100-1000s host instructions
- GenC compiles to a **dynamic binary translator**
  - Fetch, decode many instructions
  - Translate basic block using a Just-In-Time compiler to native machine code
  - Execute native basic block
- Captive DBT uses *hardware acceleration*



Sail Architecture Description

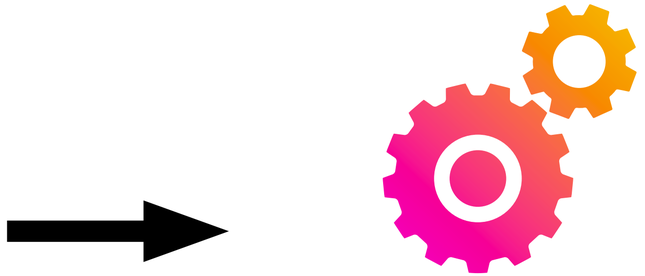
GenC Architecture Description



GenSim Toolchain

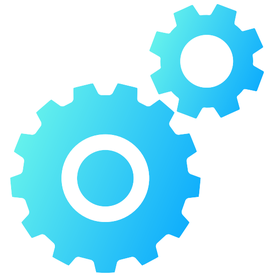
Dynamic Binary Translator

Sail Architecture Description



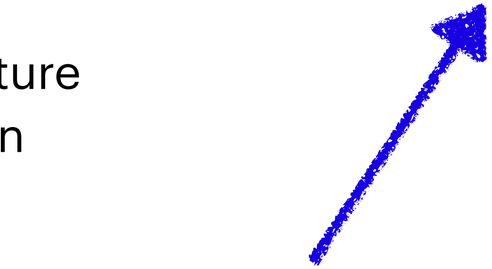
Borealis

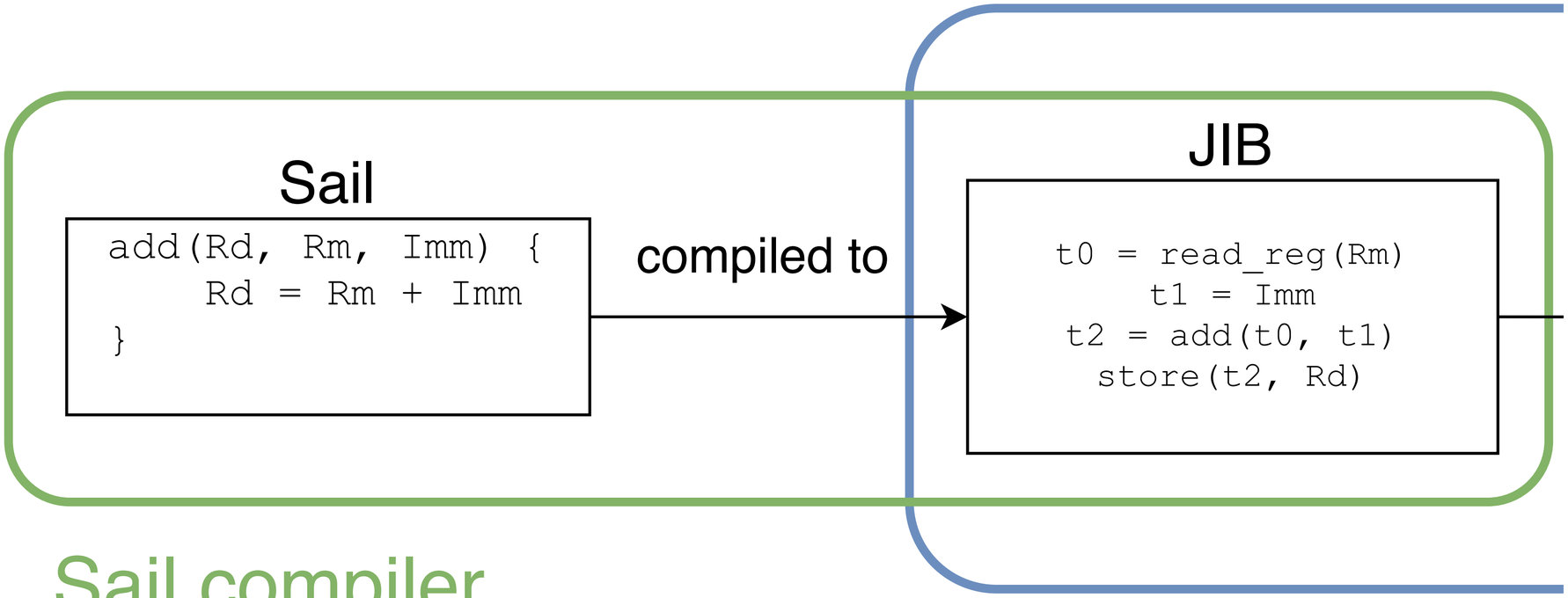
GenC Architecture Description

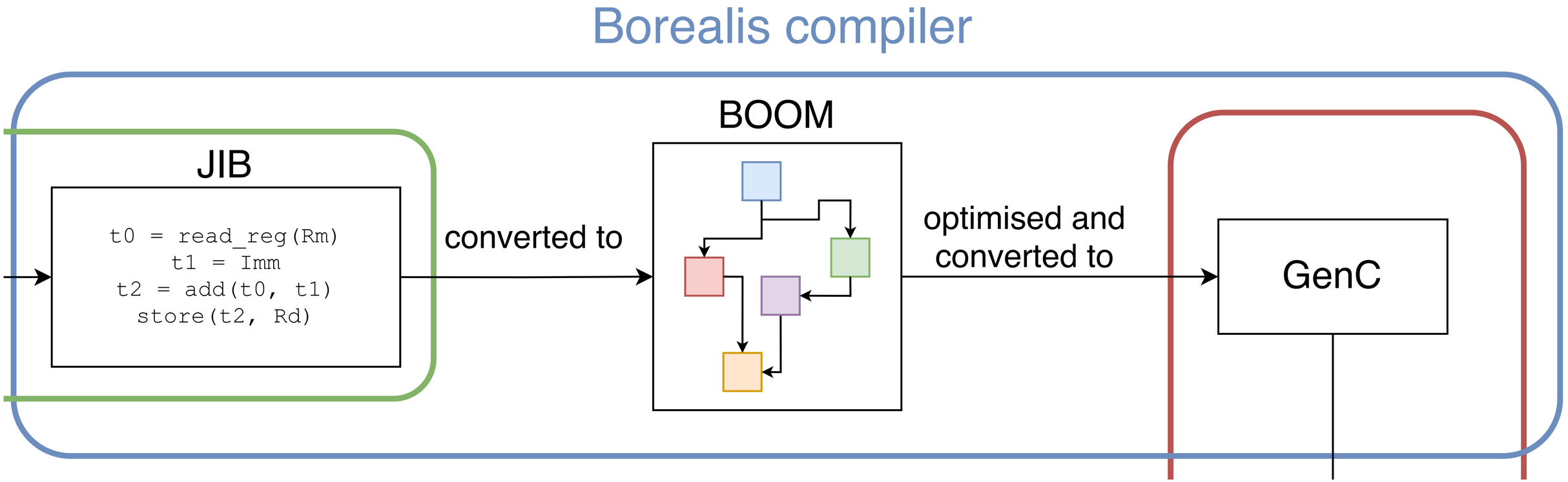


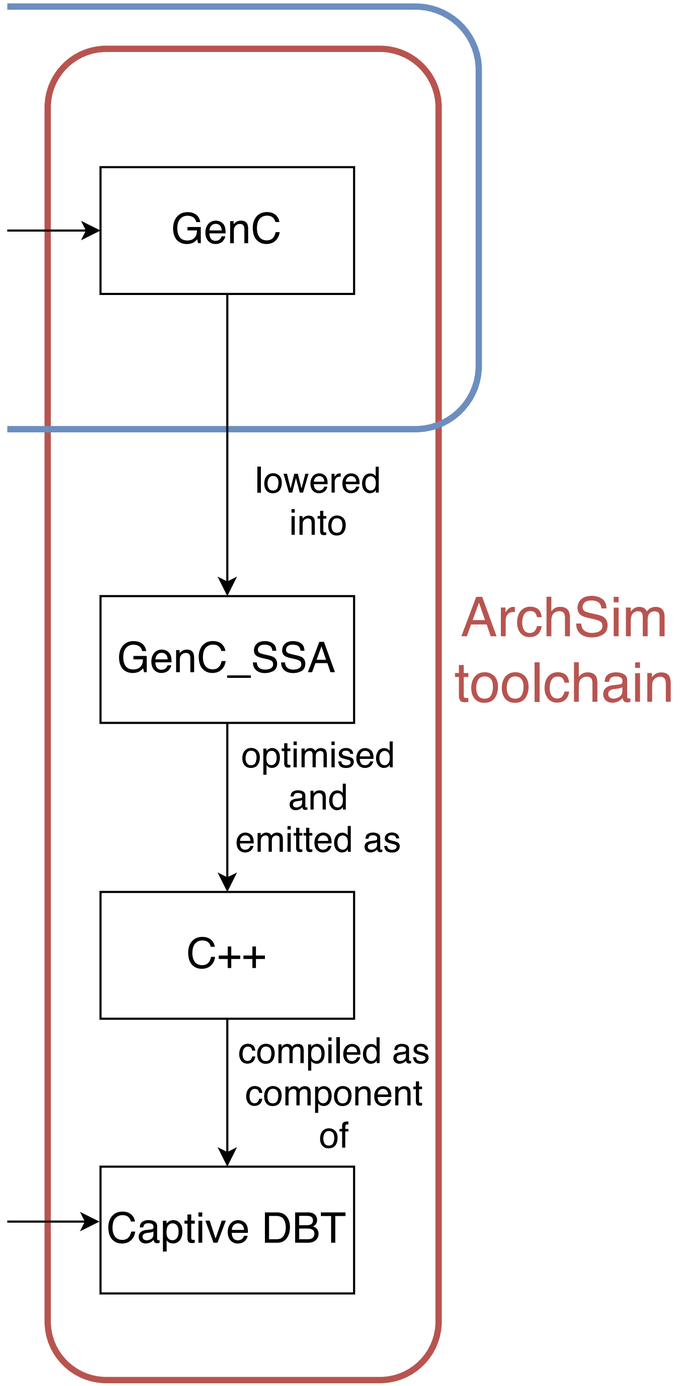
GenSim Toolchain

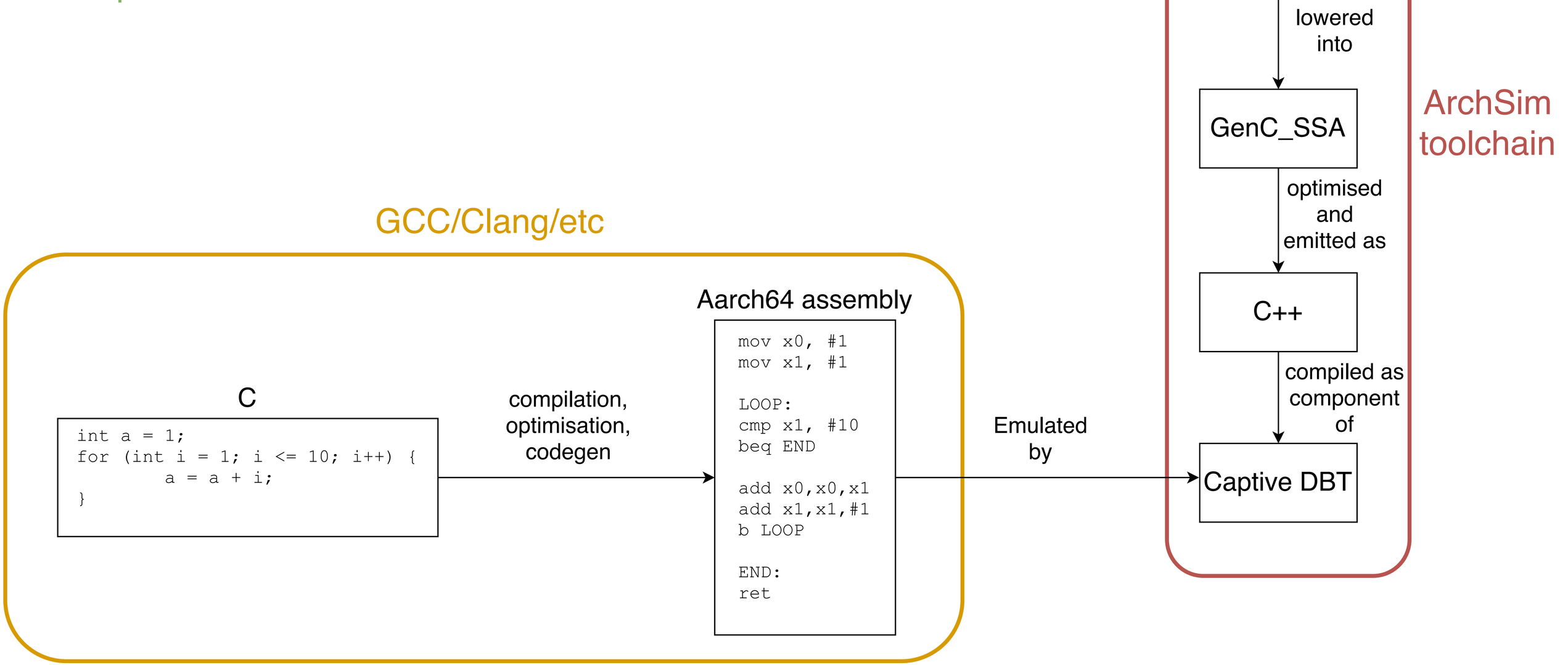
Dynamic Binary Translator

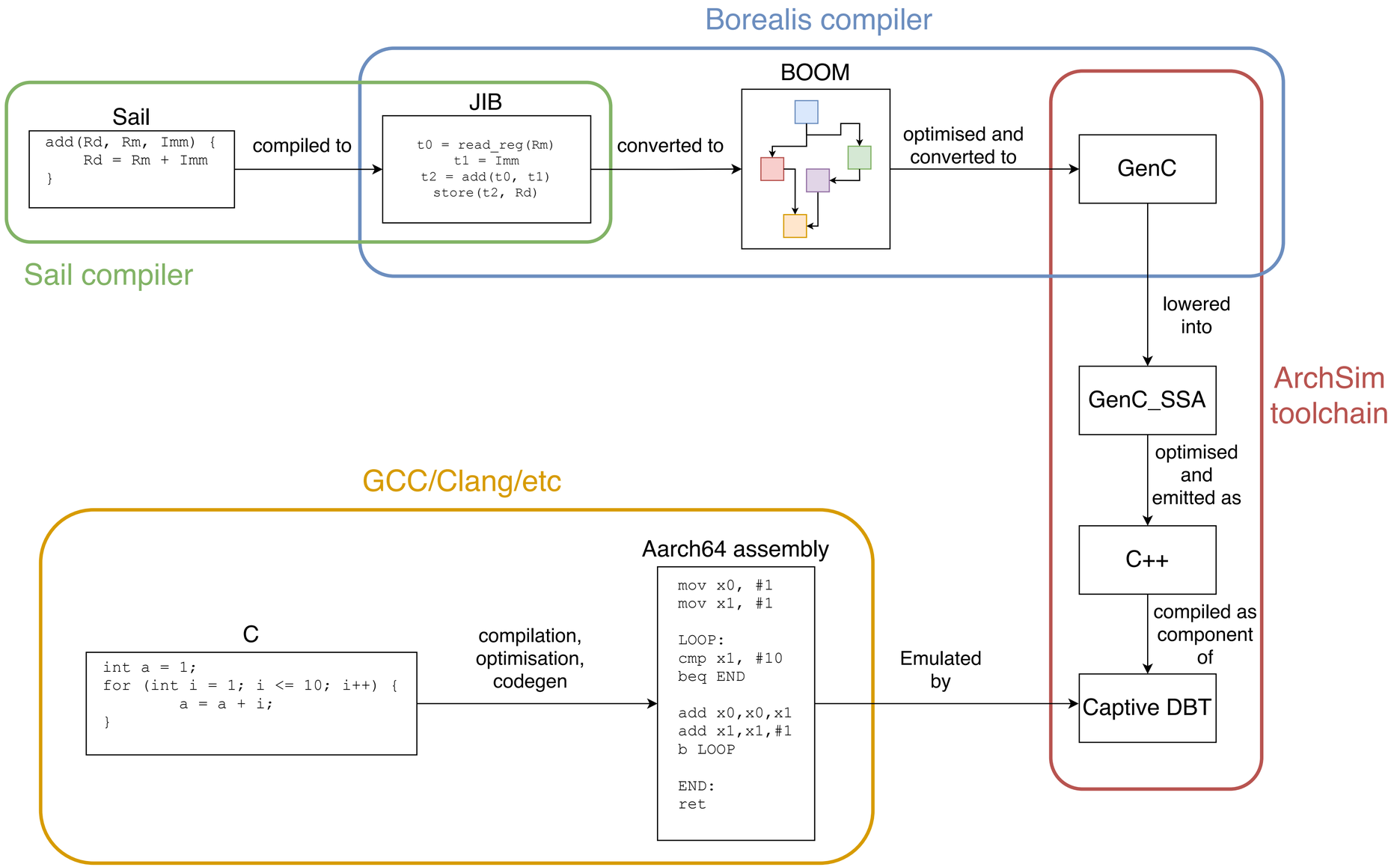



Sail
add(Rd, Rm, Imm) {
Rd = Rm + Imm
}
compiled to
JIB
t0 = read_reg(Rm)
t1 = Imm
t2 = add(t0, t1)
store(t2, Rd)
Sail compiler

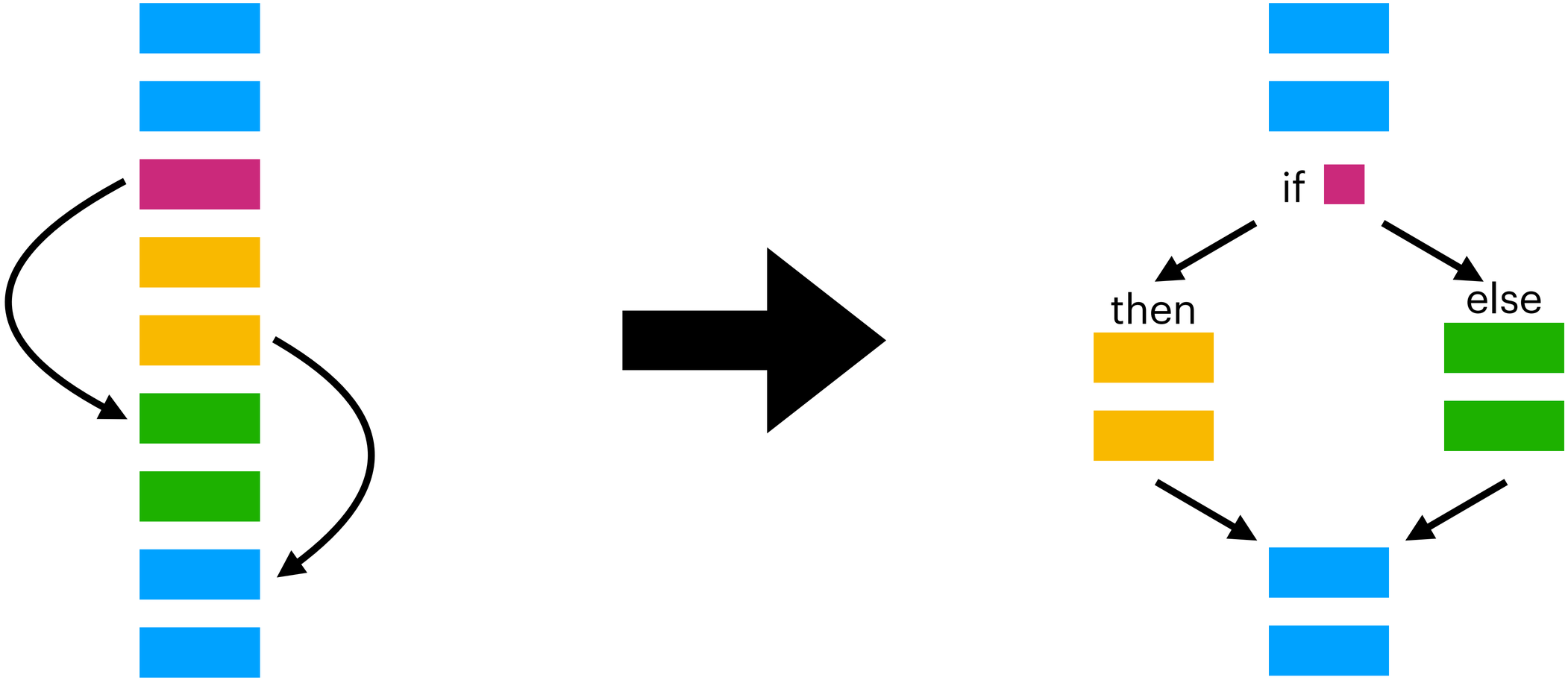

Borealis compiler
JIB
t0 = read_reg(Rm)
t1 = Imm
t2 = add(t0, t1)
store(t2, Rd)
converted to
BOOM
optimised and
converted to
GenC


GenC
lowered into
GenC_SSA
optimised and emitted as
C++
compiled as component of
Captive DBT
ArchSim toolchain


lowered
into
GenC_SSA
ArchSim
toolchain
optimised
and
emitted as
C++
compiled as
component
of
Captive DBT
GCC/Clang/etc
Aarch64 assembly
mov x0, #1
mov x1, #1
LOOP:
cmp x1, #10
beq END
add x0,x0,x1
add x1,x1,#1
b LOOP
END:
ret
C
int a = 1;
for (int i = 1; i <= 10; i++) {
a = a + i;
}
compilation,
optimisation,
codegen
Emulated
by


Borealis compiler
Sail
add(Rd, Rm, Imm) {
Rd = Rm + Imm
}
compiled to
JIB
t0 = read_reg(Rm)
t1 = Imm
t2 = add(t0, t1)
store(t2, Rd)
converted to
BOOM
optimised and
converted to
GenC
Sail compiler
lowered
into
GenC_SSA
optimised
and
emitted as
C++
compiled as
component
of
Captive DBT
ArchSim
toolchain
GCC/Clang/etc
C
int a = 1;
for (int i = 1; i <= 10; i++) {
a = a + i;
}
compilation,
optimisation,
codegen
Aarch64 assembly
mov x0, #1
mov x1, #1
LOOP:
cmp x1, #10
beq END
add x0,x0,x1
add x1,x1,#1
b LOOP
END:
ret
Emulated
by

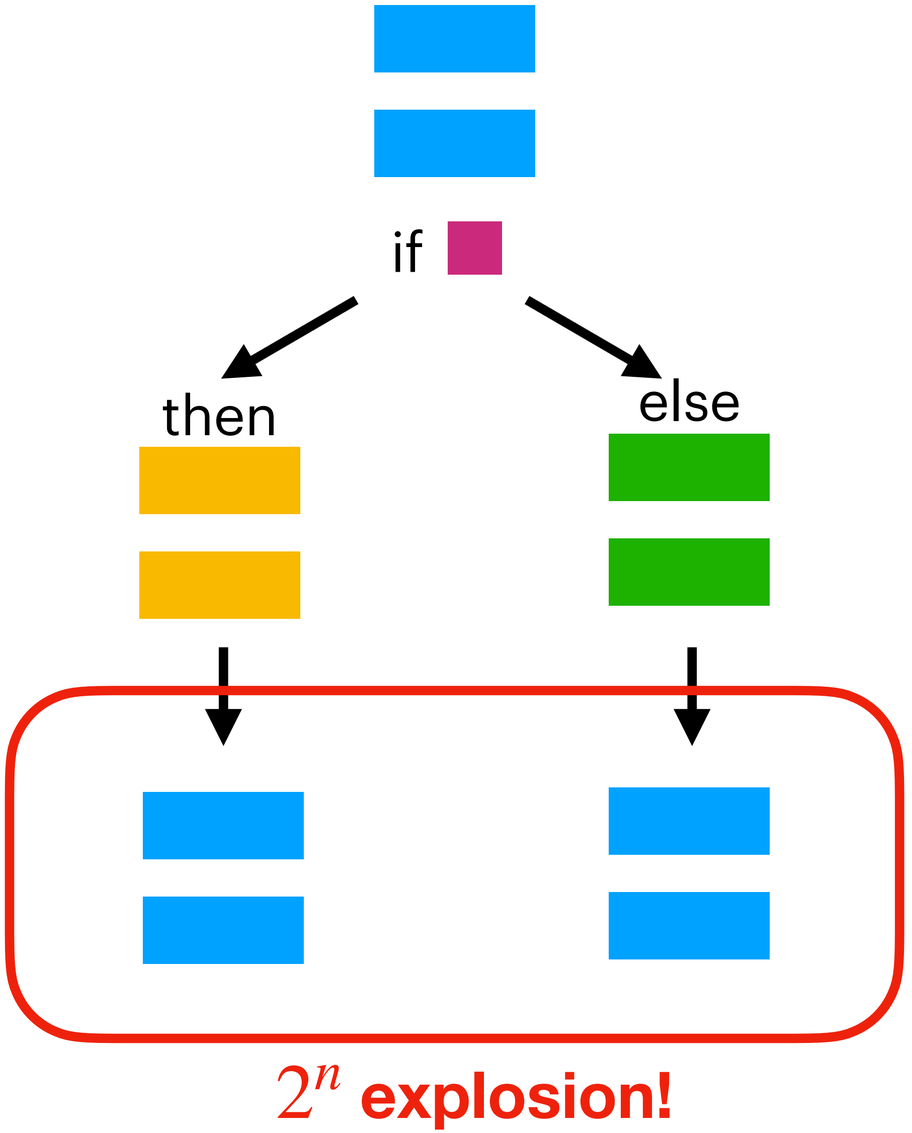
Raising Control Flow


if
then
else

Naive Approach

“Just recurse”

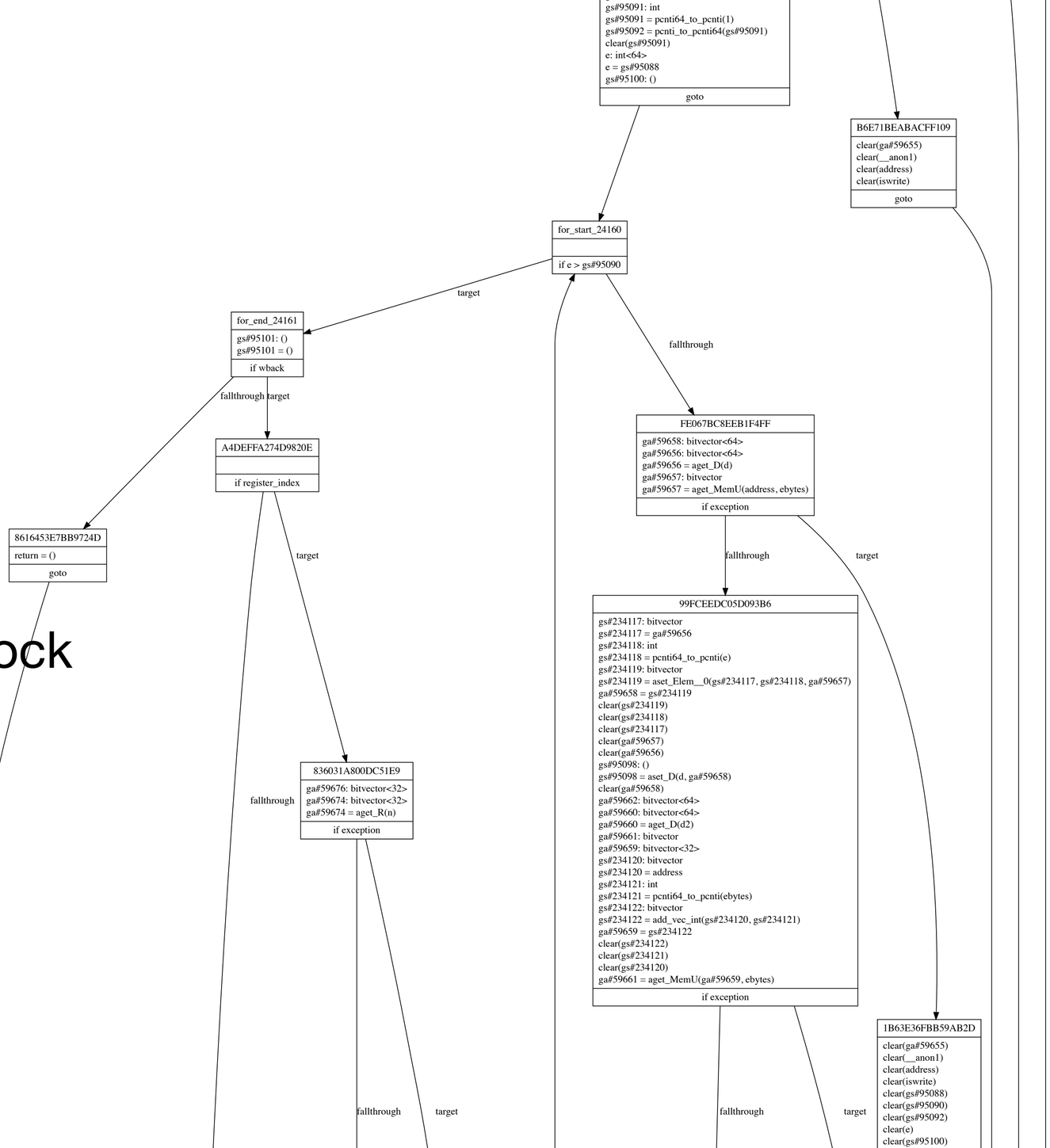
```
fn print_block(block) {
    print_statements(block.statements());

  match block.terminator() {
    Some(Condition(value, target, fallthrough)) => {
      print(“if {condition} {”);

      print_block(target);

      print(“} else {“);

      print_block(fallthrough);

      print(“}“);
    }

    Some(Unconditional(target)) => {
      print_block(block);
    }

    None => // do nothing
  }
}
```



Better Approach

“Nearest common block”

- Walk both sides of conditional
- Compare paths to find first shared block
- Print each side until that shared block is reached
- Massive reduction in code size
- Linear time



Propagating Bitvector Lengths

Bitvector Sizes

```
val bitvector_concat : forall ('n : Int) ('m : Int).
  (bits('n), bits('m)) -> bits('n + 'm)
```

- Sail has dependent types, but this information is lost in JIB
- JIB has variable-length bit vectors
- GenC only has 1, 2, 4, 8-byte integers and no allocation

```
fn bitvector_concat(n: bv, m: bv) -> bv
```

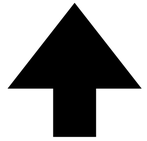
```
c = a + b
```
✅

```
c = concat(a, b)
```
❌

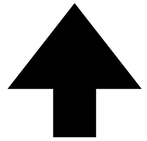
Bitvector Sizes

- Compiler pass to propagate known sizes
- Emit different code for static vs runtime
- Hope we never need to concat an unknown

Static(size = 32 bits)



Runtime(size = local var $foo)



Unknown

Does it work?

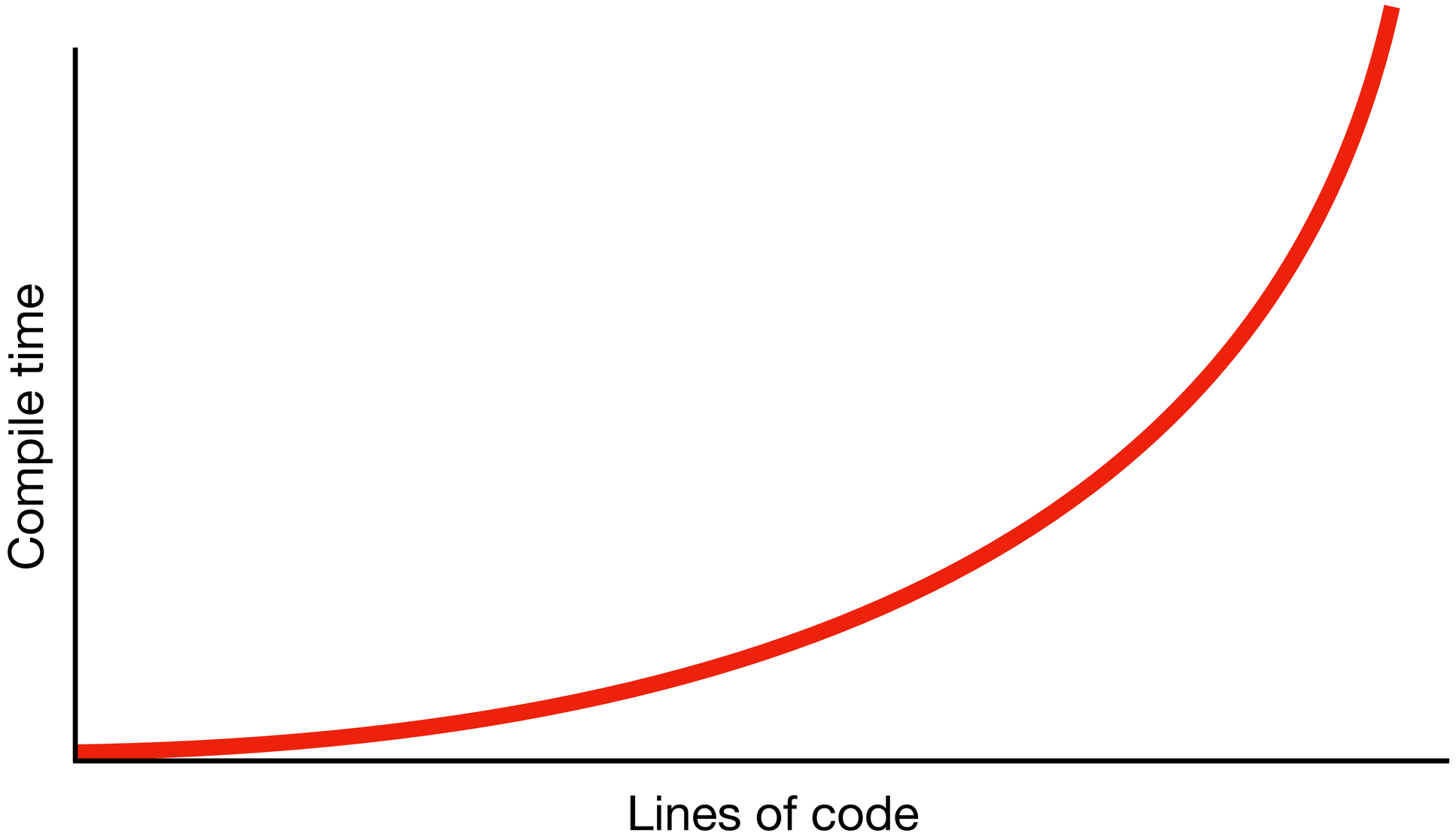
```
.globl _start

_start:
    mov x0, #0  // x
    mov x1, #1  // y
    mov x2, #0  // z
    mov x3, #0  // i
    mov x4, #10  // num

loop:
    cmp x3, x4
    bge done

    add x2, x0, x1  // z = x + y
    mov x0, x1      // x = y
    mov x1, x2      // y = z

    add x3, x3, #1
    b loop

done:
    mov x0, x2
    mov x0, #0
    mov w8, #93
    svc 0
```

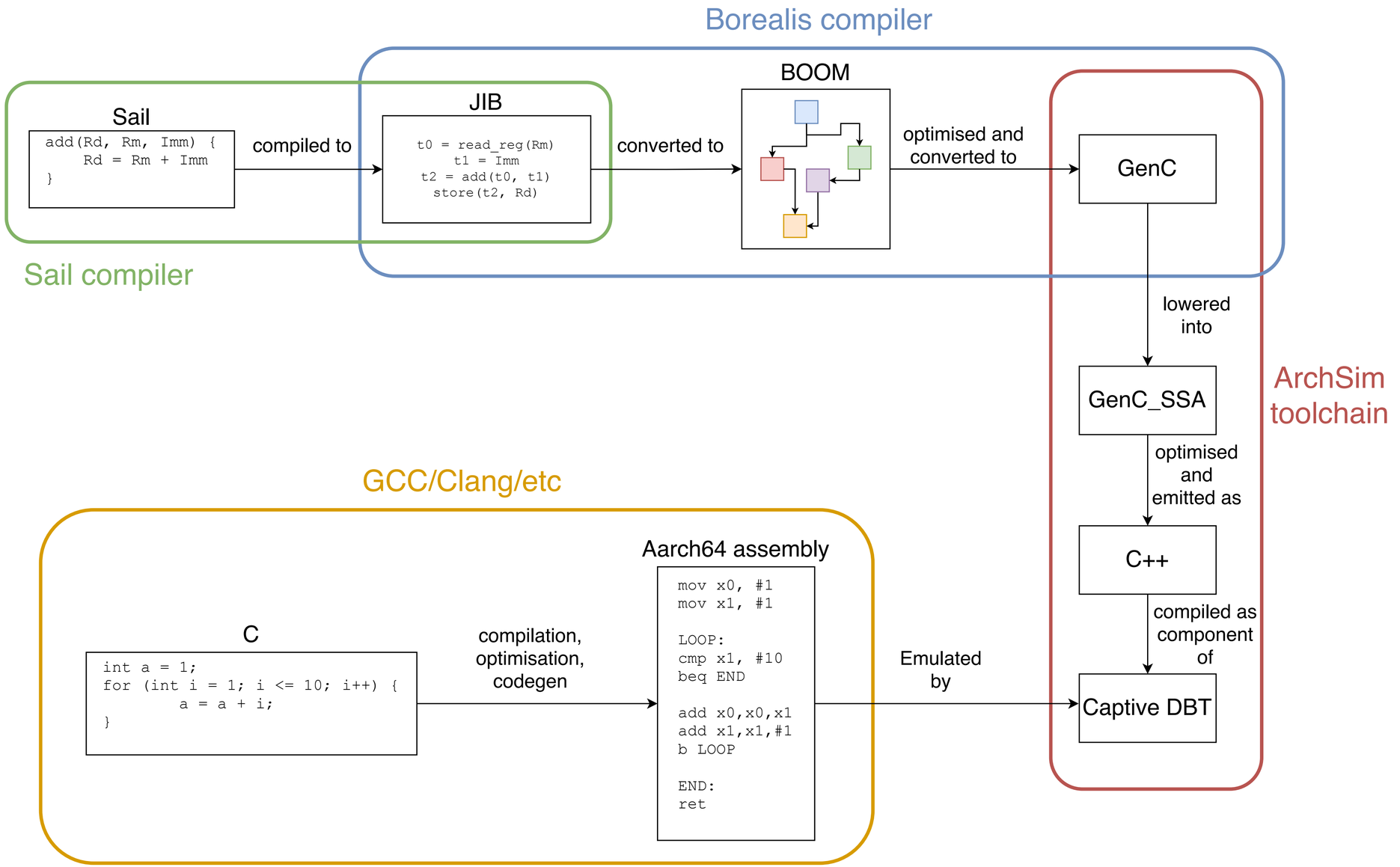
Guest program

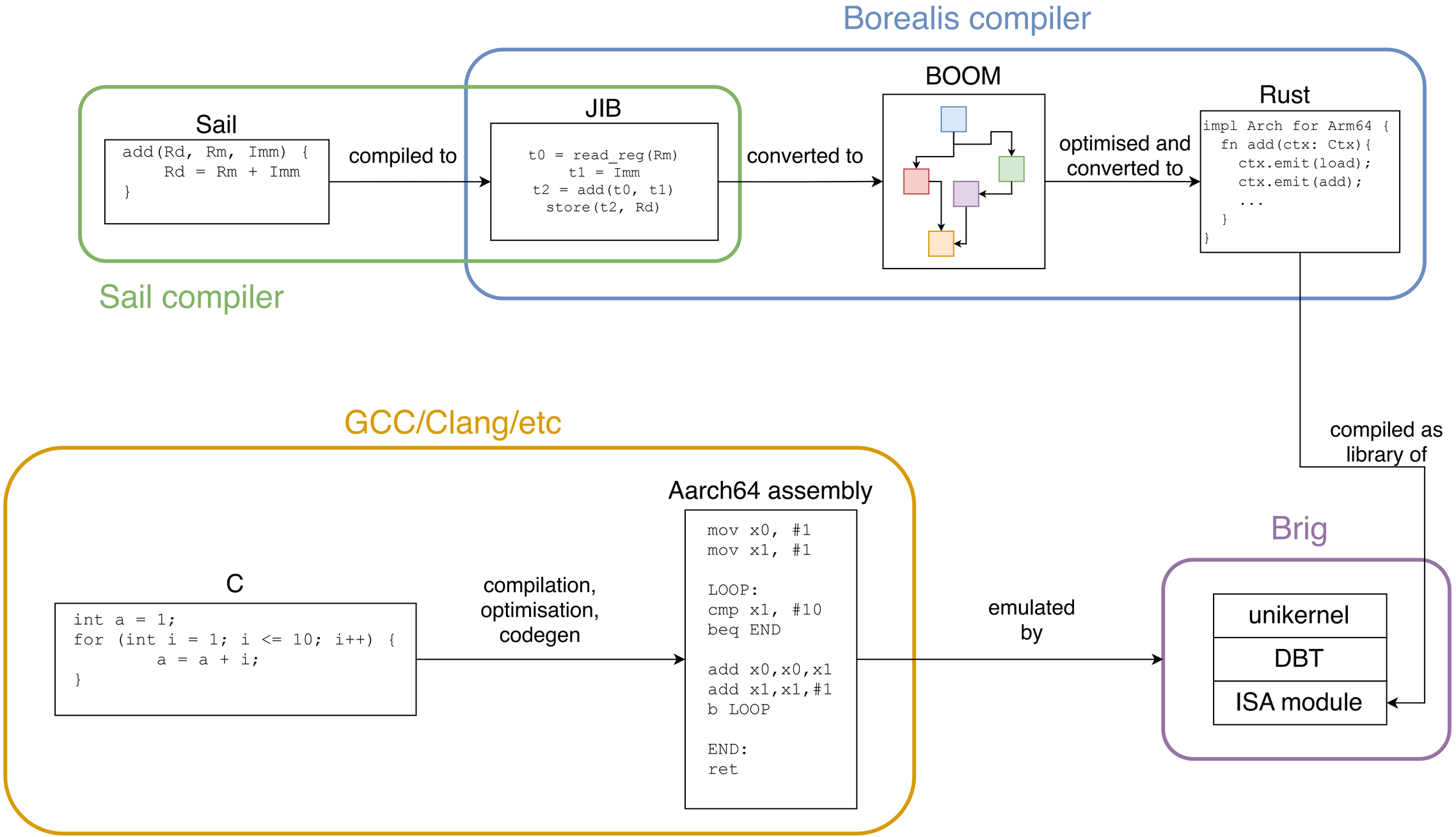
```
[0x00000000004000d4] d2800000 (R[0][0]
[0x00000000004000d8] d2800021 (R[0][1]
[0x00000000004000dc] d2800002 (R[0][2]
[0x00000000004000e0] d2800003 (R[0][3]
[0x00000000004000e4] d2800144 (R[0][4]
[0x00000000004000e8] eb04007f (R[0][3]
[0x00000000004000ec] 540000ca (R[2] <-
[0x00000000004000f0] 8b010002 (R[0][0]
[0x00000000004000f4] aa0103e0 (R[0][1]
[0x00000000004000f8] aa0203e1 (R[0][2]
[0x00000000004000fc] 91000463 (R[0][3]
[0x0000000000400100] 17fffffa (R[2] <-
[0x00000000004000e8] eb04007f (R[0][3]
[0x00000000004000ec] 540000ca (R[2] <-
[0x00000000004000f0] 8b010002 (R[0][0]
[0x00000000004000f4] aa0103e0 (R[0][1]
[0x00000000004000f8] aa0203e1 (R[0][2]
[0x00000000004000fc] 91000463 (R[0][3]
```

Execution trace

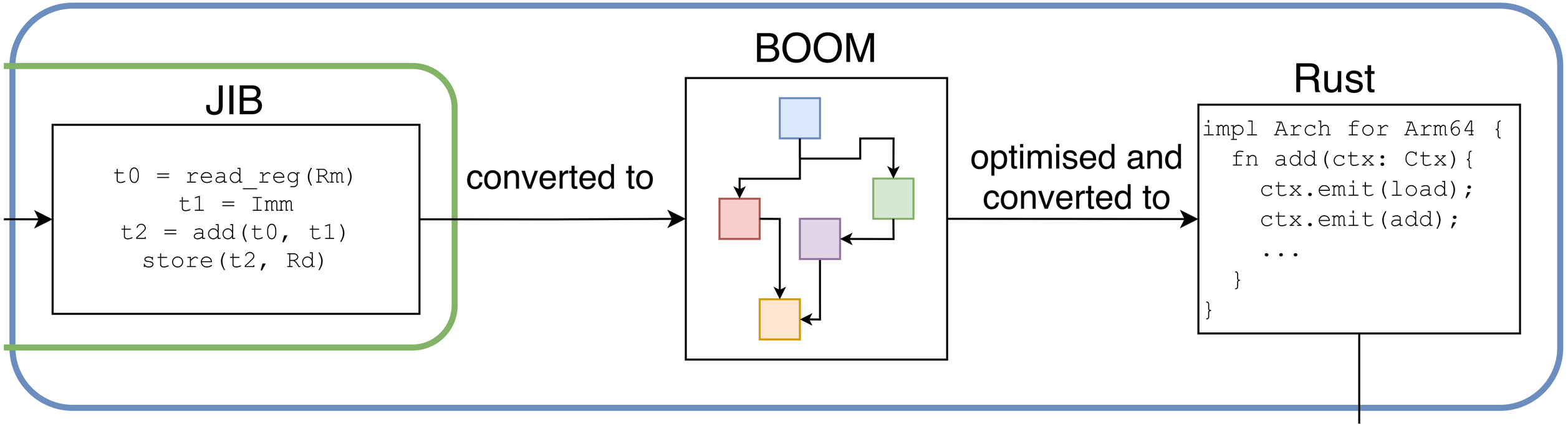
Scalability Issues

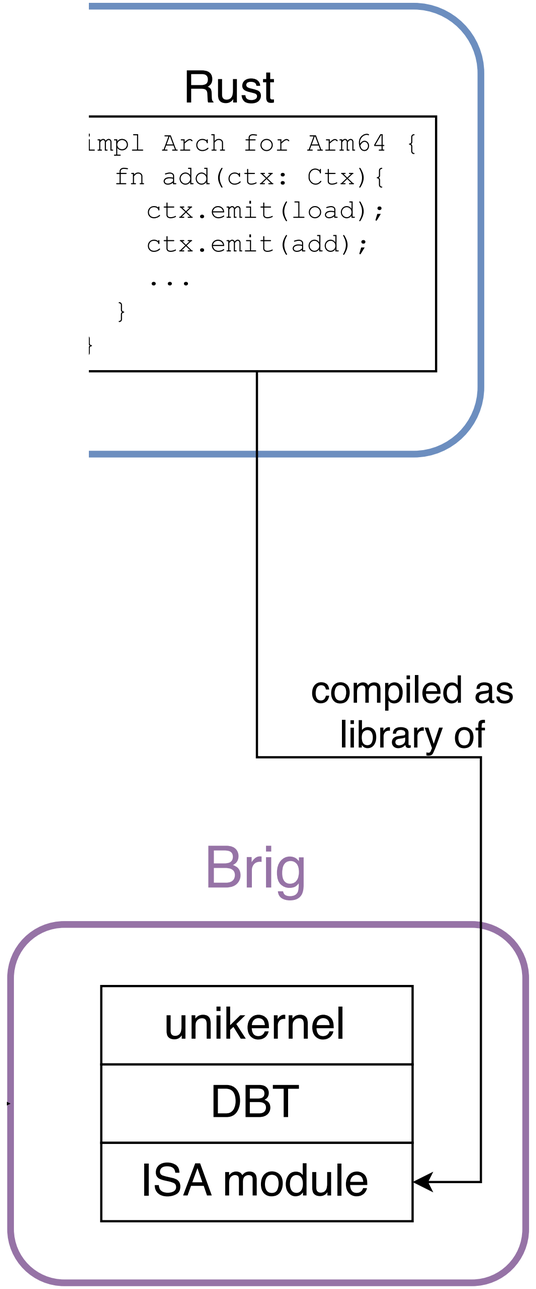

Compile time
Lines of code


Borealis compiler
Sail
add(Rd, Rm, Imm) {
Rd = Rm + Imm
}
compiled to
JIB
t0 = read_reg(Rm)
t1 = Imm
t2 = add(t0, t1)
store(t2, Rd)
converted to
BOOM
optimised and
converted to
GenC
Sail compiler
lowered
into
GenC_SSA
optimised
and
emitted as
C++
compiled as
component
of
Captive DBT
ArchSim
toolchain
GCC/Clang/etc
C
int a = 1;
for (int i = 1; i <= 10; i++) {
a = a + i;
}
compilation,
optimisation,
codegen
Aarch64 assembly
mov x0, #1
mov x1, #1
LOOP:
cmp x1, #10
beq END
add x0,x0,x1
add x1,x1,#1
b LOOP
END:
ret
Emulated
by


Borealis compiler
Sail
add(Rd, Rm, Imm) {
Rd = Rm + Imm
}
compiled to
JIB
t0 = read_reg(Rm)
t1 = Imm
t2 = add(t0, t1)
store(t2, Rd)
converted to
BOOM
optimised and
converted to
Rust
impl Arch for Arm64 {
fn add(ctx: Ctx){
ctx.emit(load);
ctx.emit(add);
...
}
}
Sail compiler
GCC/Clang/etc
C
int a = 1;
for (int i = 1; i <= 10; i++) {
a = a + i;
}
compilation,
optimisation,
codegen
Aarch64 assembly
mov x0, #1
mov x1, #1
LOOP:
cmp x1, #10
beq END
add x0,x0,x1
add x1,x1,#1
b LOOP
END:
ret
emulated
by
compiled as
library of
Brig
unikernel
DBT
ISA module

Borealis compiler




Rust
impl Arch for Arm64 {
fn add(ctx: Ctx){
ctx.emit(load);
ctx.emit(add);
...
}
}
compiled as
library of
Brig
unikernel
DBT
ISA module

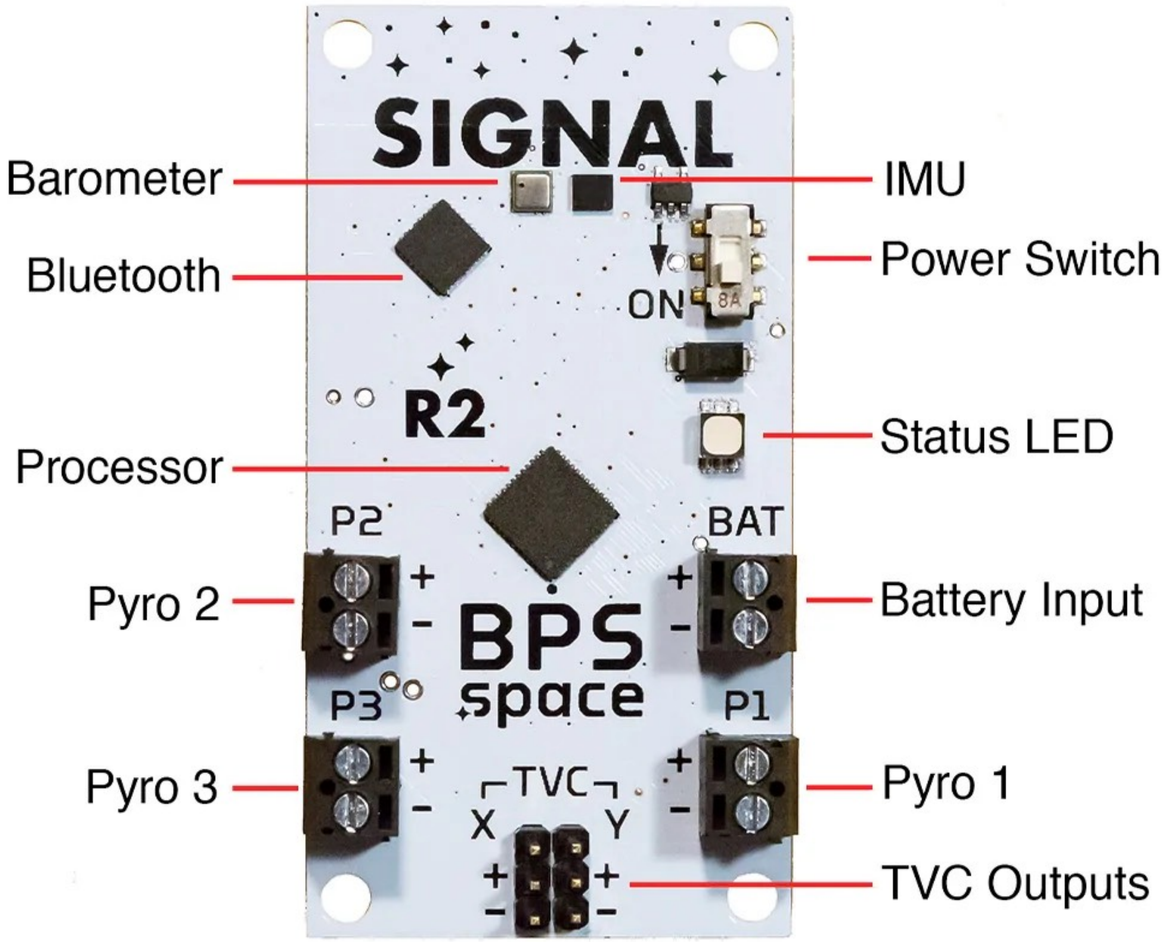

INFO [brig::logger]
starting...
TRACE [brig::arch] initializing core 0
TRACE [brig::arch::x86] cr0=CR0_ENABLE_PAGING | CR0_WRITE_PROTECT | CR0_NUMERIC_ERROR | CR0
TRACE [brig::arch::x86] cr4=CR4_ENABLE_FSGSBASE | CR4_UNMASKED_SSE | CR4_ENABLE_SSE | CR4_E
TRACE [brig::arch::x86] efer=EferFlags(SYSTEM_CALL_EXTENSIONS | LONG_MODE_ENABLE | LONG_MOD
INFO [brig::arch::x86::memory] heap allocator initialized, 7.84 GiB available
DEBUG [brig::devices::pcie] enumerating pcie 0..=255 b0000000
WARN [brig::devices::pcie] unsupported pcie device 8086:29c0
WARN [brig::devices::pcie] unsupported pcie device 1234:1111
WARN [brig::devices::pcie] unsupported pcie device 8086:10d3
TRACE [brig::devices::virtio] probing virtio block device
DEBUG [brig::devices::manager] registering device 6e534ac8-57df-42d0-bd9b-c8990fca18a2: Sha
TRACE [brig::devices::manager] added alias "disk00:03.0" for 6e534ac8-57df-42d0-bd9b-c8990f
TRACE [brig::devices::virtio] probing virtio block device
DEBUG [brig::devices::manager] registering device c9db19da-51fb-4596-b3ff-11f4b3d439b5: Sha
TRACE [brig::devices::manager] added alias "disk00:04.0" for c9db19da-51fb-4596-b3ff-11f4b3
WARN [brig::devices::pcie] unsupported pcie device 8086:2918
TRACE [brig::devices::pcie] probing sata controller
WARN [brig::devices::pcie] unsupported pcie device 8086:2930
TRACE [brig::devices::lapic] ticks-per-period=10001808, freq=1000180800
TRACE [brig::devices::lapic] lapic frequency=1000180800
TRACE [brig::scheduler] scheduler started
TRACE [brig::guest] starting guest
TRACE [brig::arch::x86::memory] translating ffff81807ff2e3a0 to Some(PhysAddr(0x7ff2e3a0))
TRACE [brig::arch::x86::memory] translating ffff8181440000000 to Some(PhysAddr(0x144000000))
TRACE [brig::arch::x86::memory] translating ffff81807ff2e1a7 to Some(PhysAddr(0x7ff2e1a7))
TRACE [brig::arch::x86::memory] translating ffff81807bf75f80 to Some(PhysAddr(0x7bf75f80))
TRACE [brig::guest] kernel len: 0x27caa00, got config: Config {

Future Work

What’s in an embedded system?

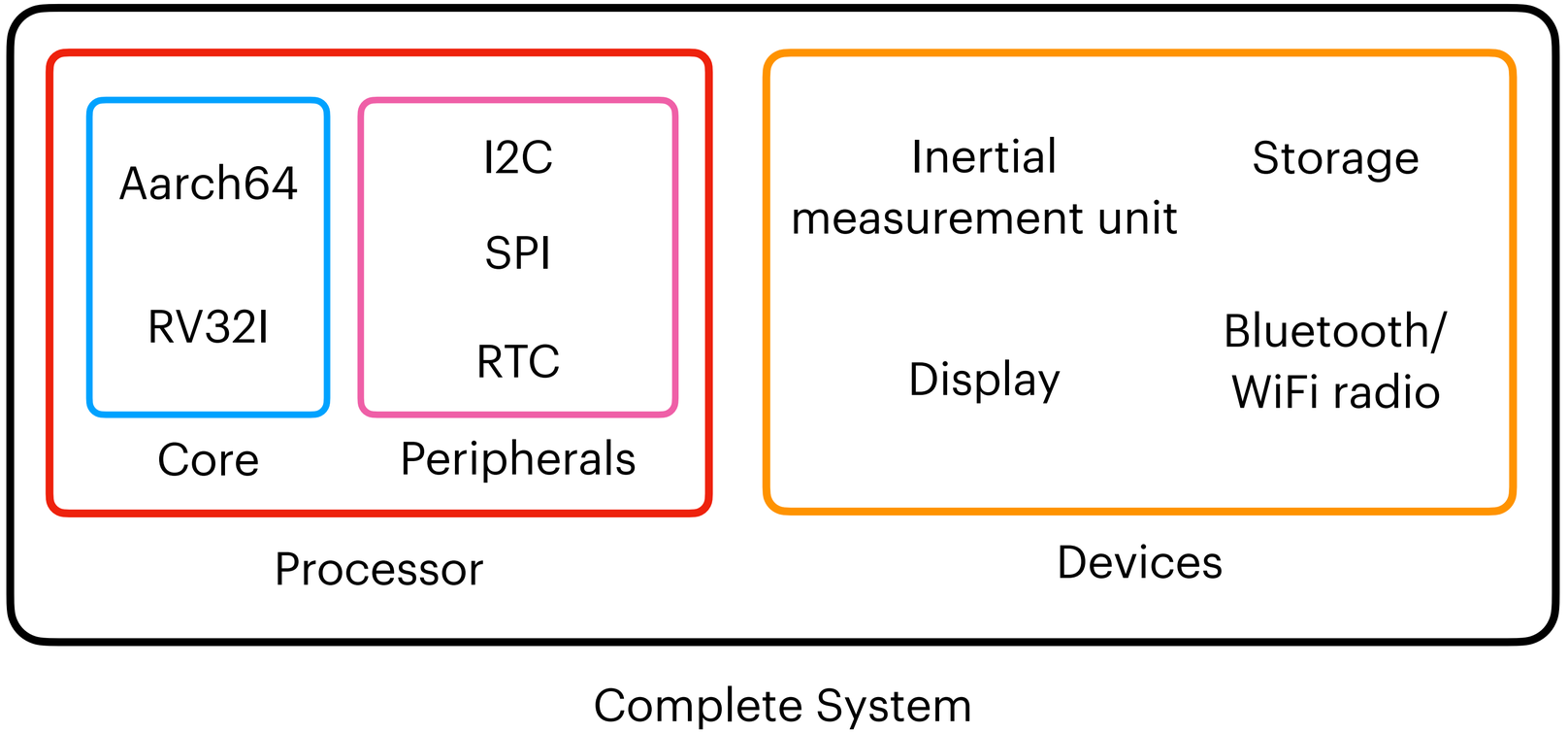
STMicroelectronics

What’s in an embedded system?

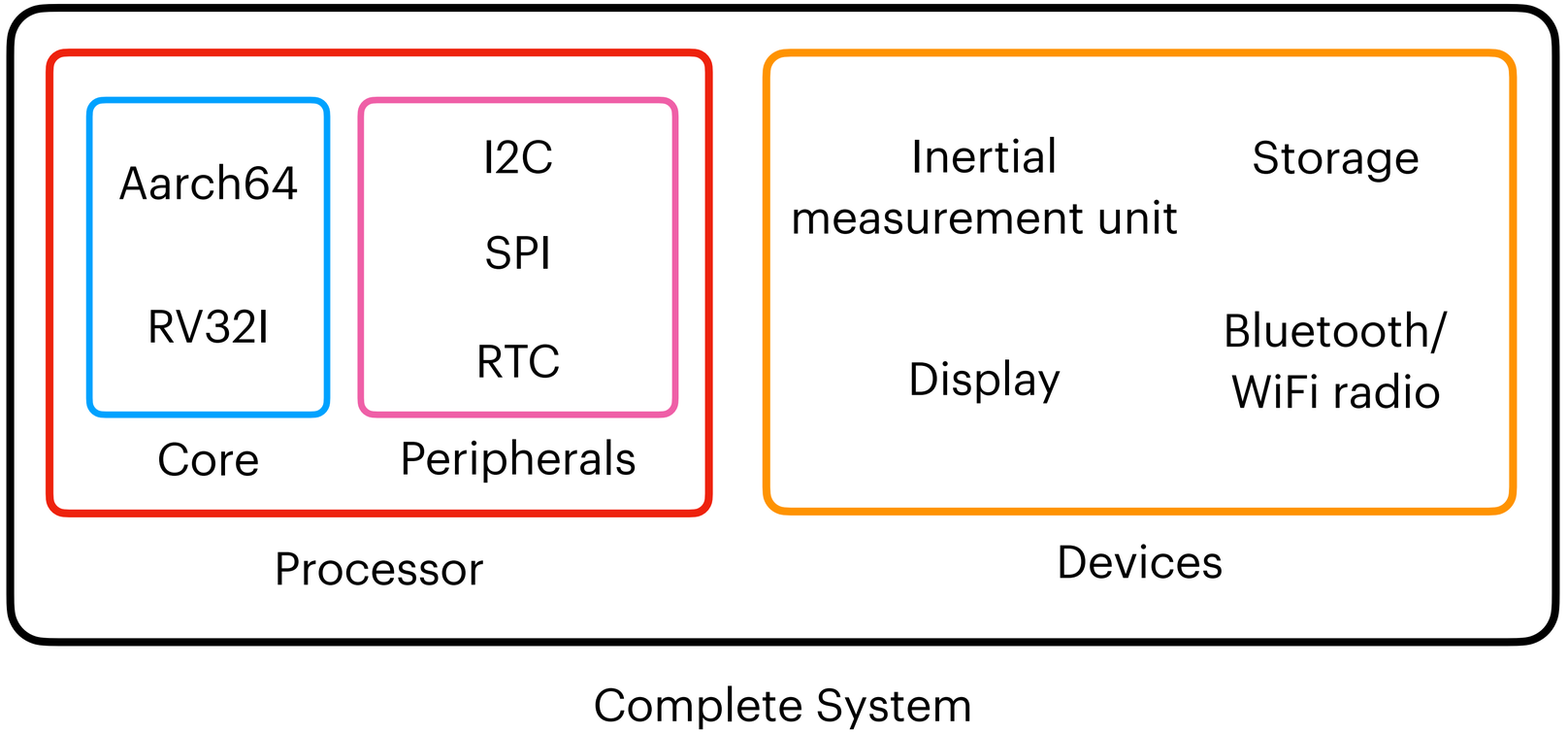


Signal R2 rocket control board, Joe Barnard, bps.space

What’s in an embedded system?



How do we easily describe this?



Parts Library

DigiKey | All Products | Enter keyword or part # | Login or REGISTER | 0 item(s)

Products | Manufacturers | Resources | FREE DELIVERY on Orders over £33!*

Product Index > Sensors, Transducers > Motion Sensors > IMUs (Inertial Measurement Units) > TDK InvenSense MPU-6050

Dark Mode | Share

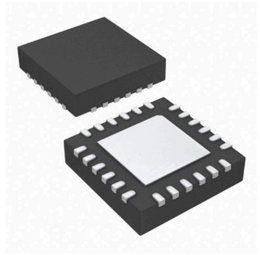


Image shown is a representation only. Exact specifications should be obtained from the product data sheet.

MPU-6050

DigiKey Part Number	1428-1007-2-ND - Tape & Reel (TR) 1428-1007-1-ND - Cut Tape (CT)
Manufacturer	TDK InvenSense
Manufacturer Product Number	MPU-6050
Description	IMU ACCEL/GYRO 3-AXIS I2C 24QFN
Customer Reference	
Detailed Description	Accelerometer, Gyroscope, 6 Axis Sensor I2C Output
Datasheet	Datasheet
EDA/CAD Models	MPU-6050 Models

In-Stock: 31,195

Can ship immediately

This product is no longer manufactured and will no longer be stocked once stock is depleted. ***View Substitutes***

QUANTITY

Add to List | Add to Basket

Download simulation component

10	£5.53000	£55.30
25	£5.11520	£127.88
100	£4.42390	£442.39
500	£4.14744	£2,073.72
1,000	£3.87093	£3,870.93

Tape & Reel (TR)

Product Attributes

TYPE	DESCRIPTION	SELECT ALL
Category	Sensors, Transducers Motion Sensors IMUs (Inertial Measurement Units)	
Manufacturer	TDK InvenSense	
Series	-	

Feedback | Need Help?

What's the temperature?

STM32F3 ⟷ BMP280

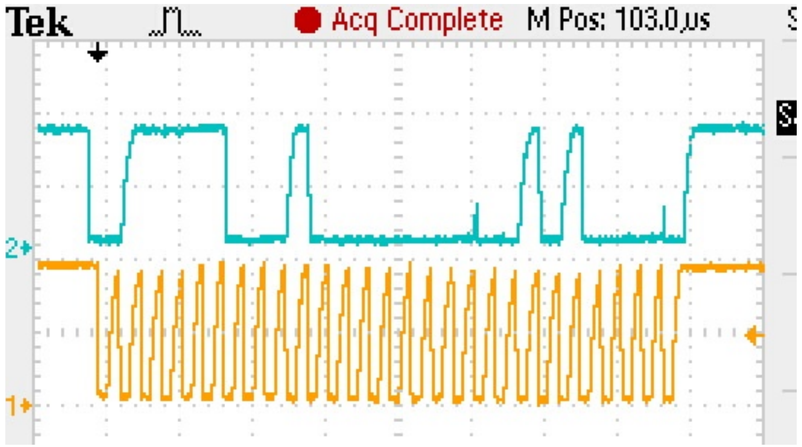
28.3°C

0x3E 0x10

I2C interface ⟷ I2C interface

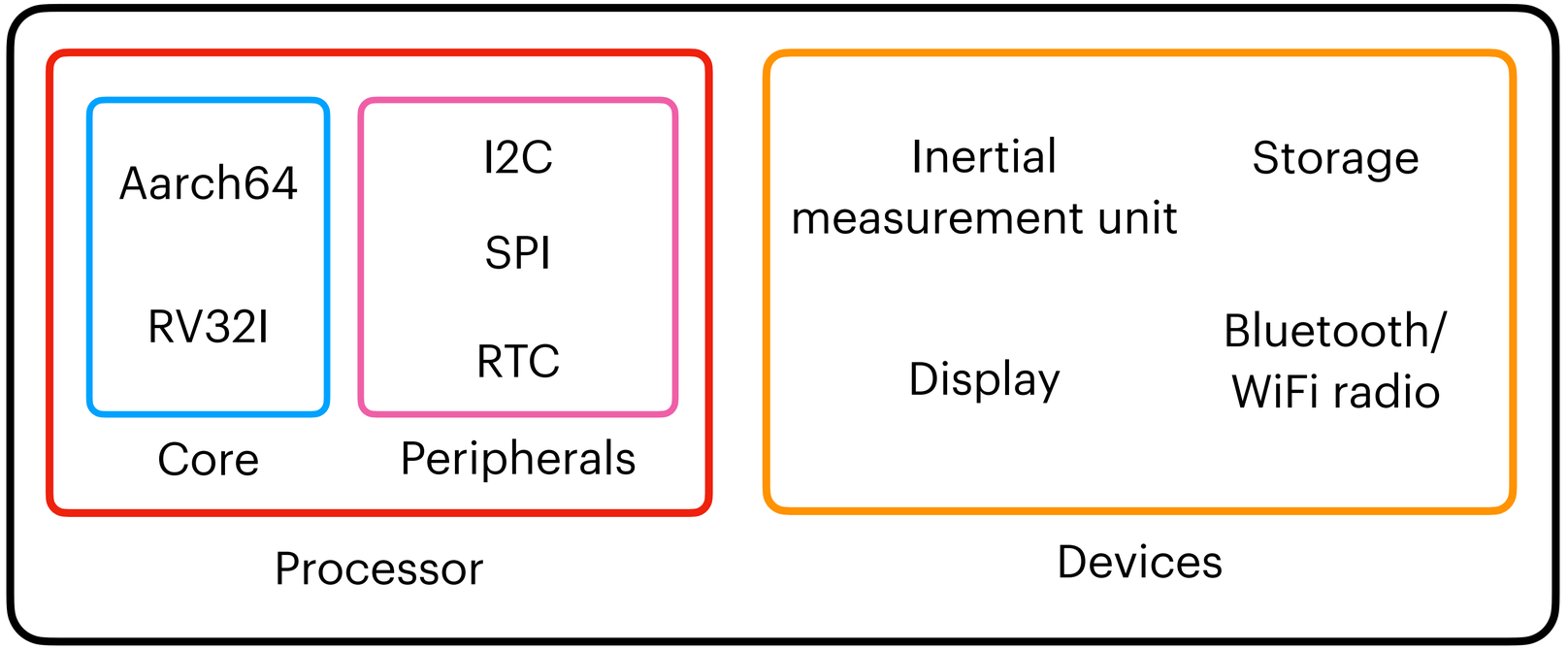
0x3E 0x4E0F

Physical



Physical

How do we *performantly* describe this?



How to Compile a Compiler

Producing Fast Full-system Emulators from Formal Specifications

Ferdia McKeogh, University of St Andrews