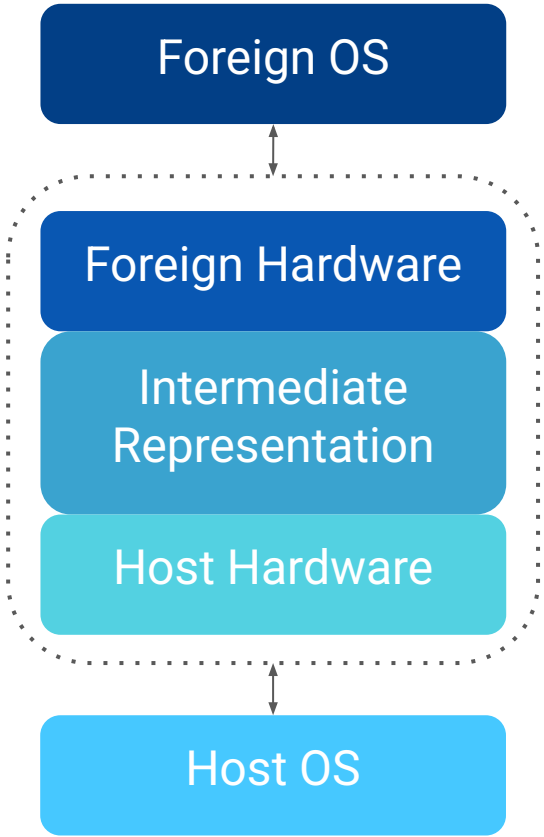
HABITAT: Hardware-Accelerated Binary Translator

Sing Hin To (Jason)[1], Tom Spink[1]
University of St Andrews[1]

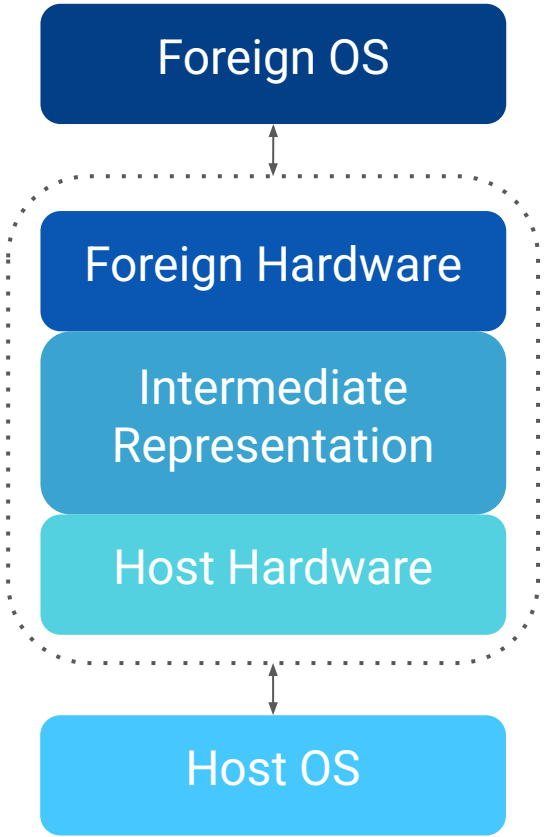
University of St Andrews | FOUNDED 1413

Background

Emulators


Foreign OS
Foreign Hardware
Intermediate Representation
Host Hardware
Host OS

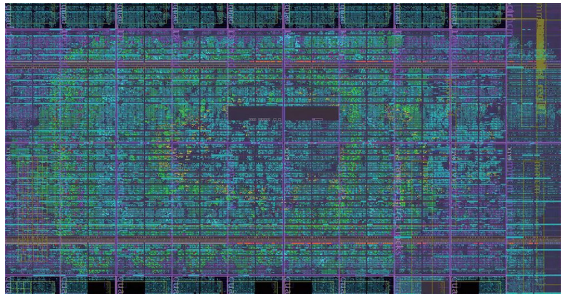
Emulators



	Commercial	Recreational
System Level	QEMU	
Application Level		BlueStacks by now.gg

FPGA

Transform logic into hardware designs

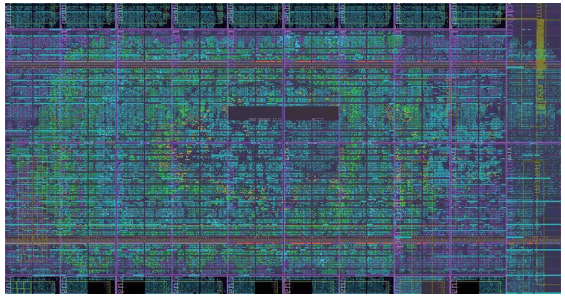


Specialised

Fast

FPGA

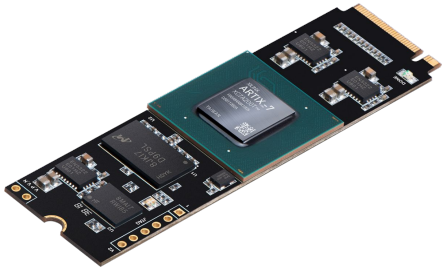
Transform logic into hardware designs



Specialised

Fast

Multiple sizes / mediums (USB, PCIe, M.2 …)

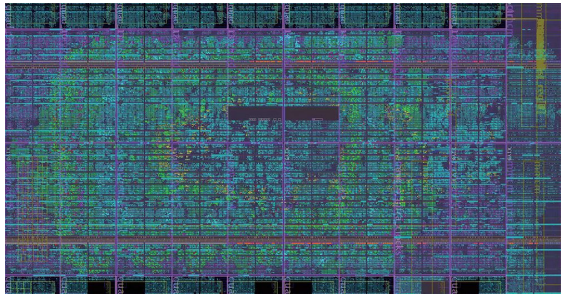


Accessible

Compact

FPGA

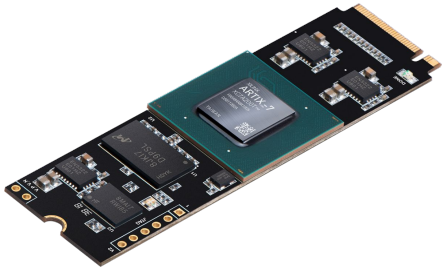
Transform logic into hardware designs



Specialised

Fast

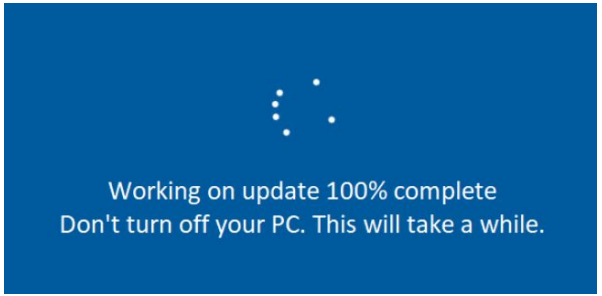
Multiple sizes / mediums (USB, PCIe, M.2 …)



Accessible

Compact

(Re)load configuration during runtime

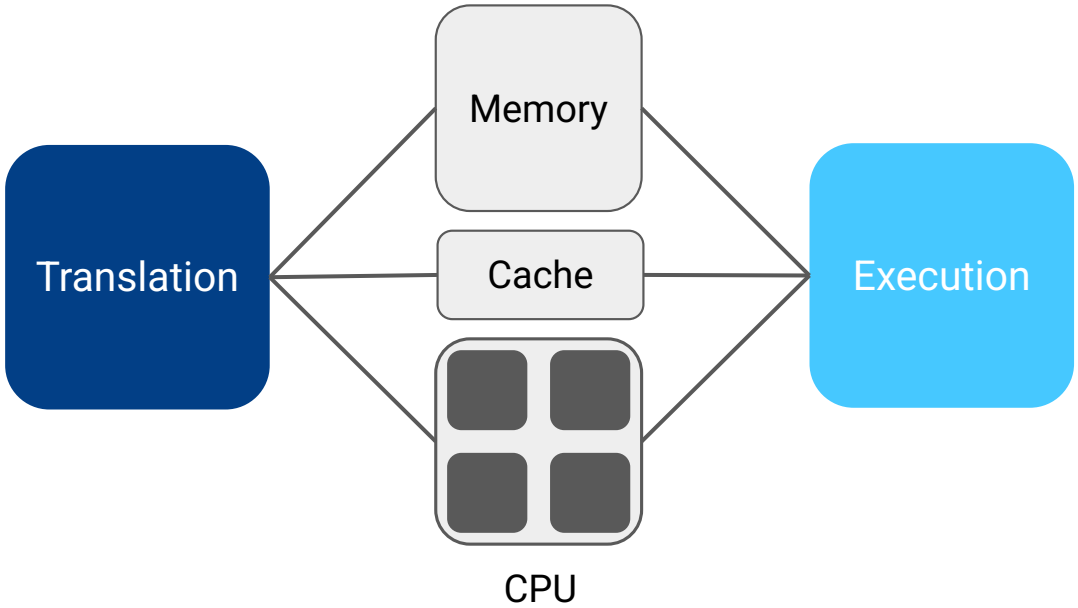


Flexible

Upgradable

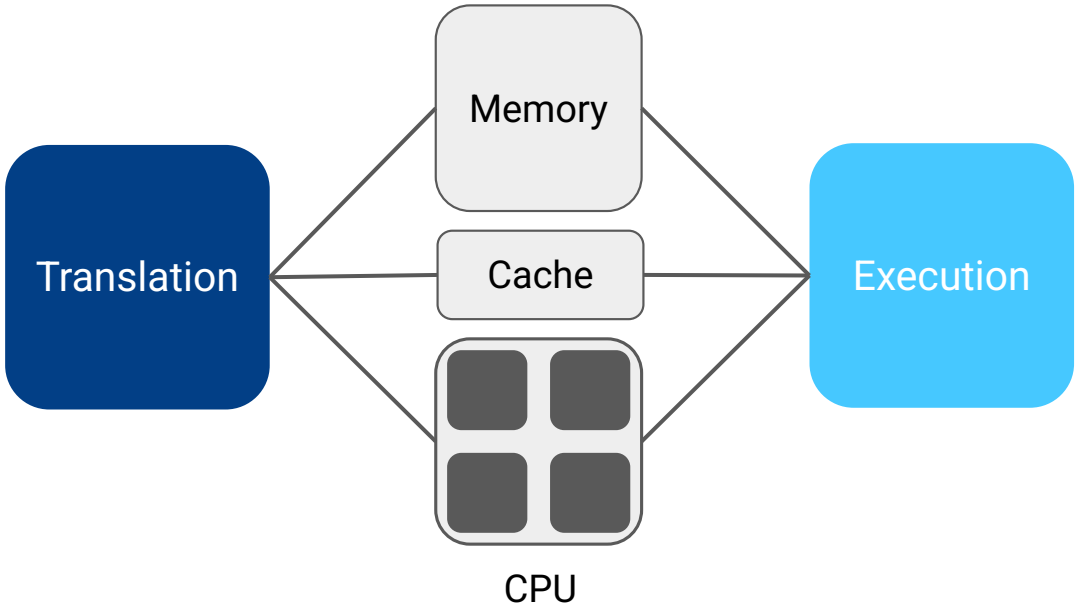
Current Situation

Software-Based Approach



Shared infrastructure

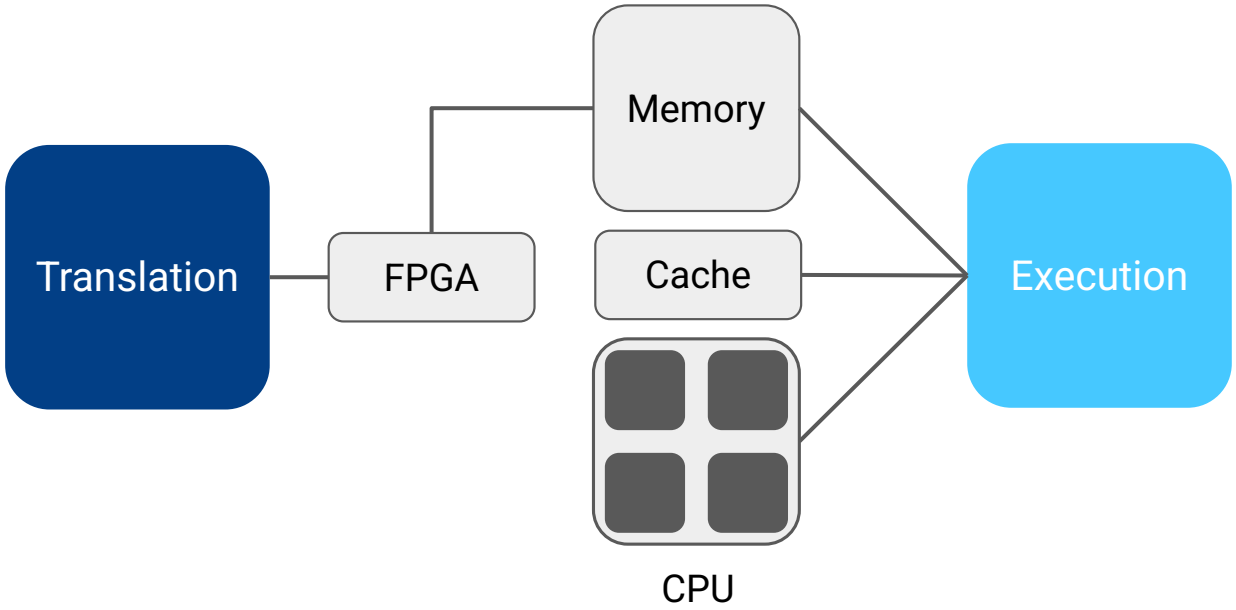
Software-Based Approach



- **Scalability** issue
- **Performance** bottleneck
- Hard to **optimise**

Shared infrastructure

Hybrid Approach

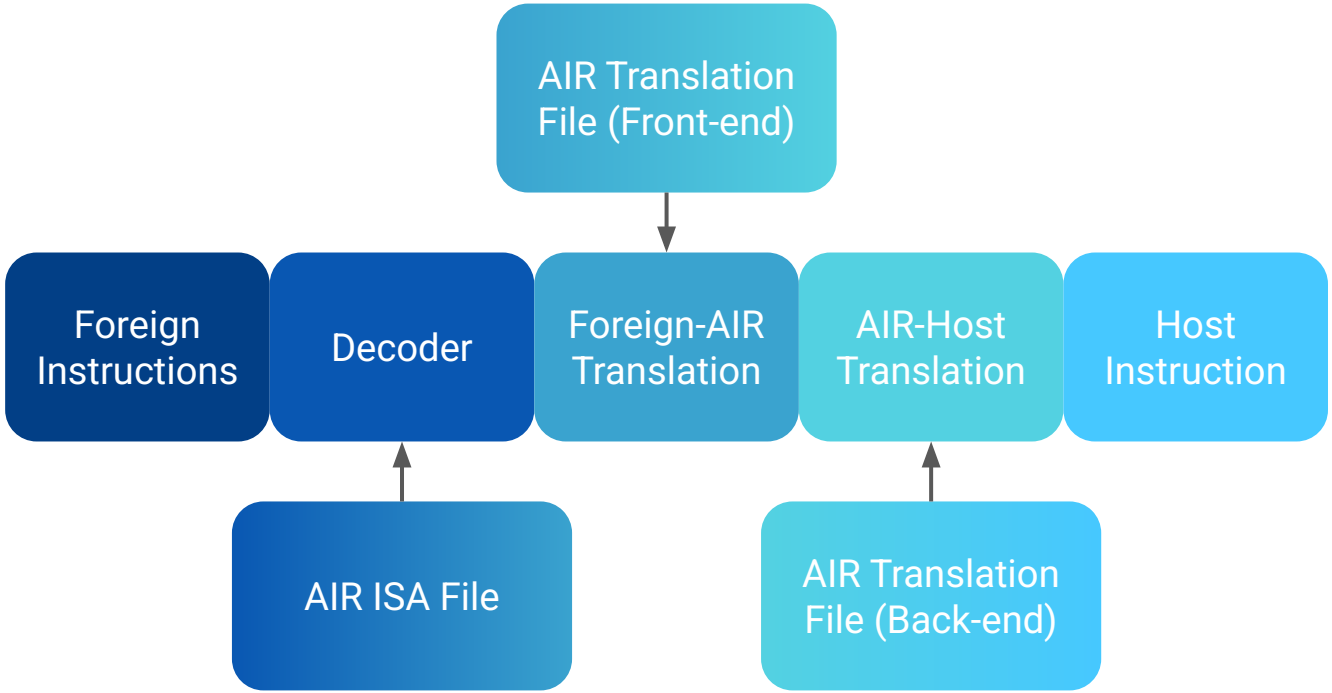


- **One** chip for all
- **Performance** decoupling
- Intensive **optimisation**

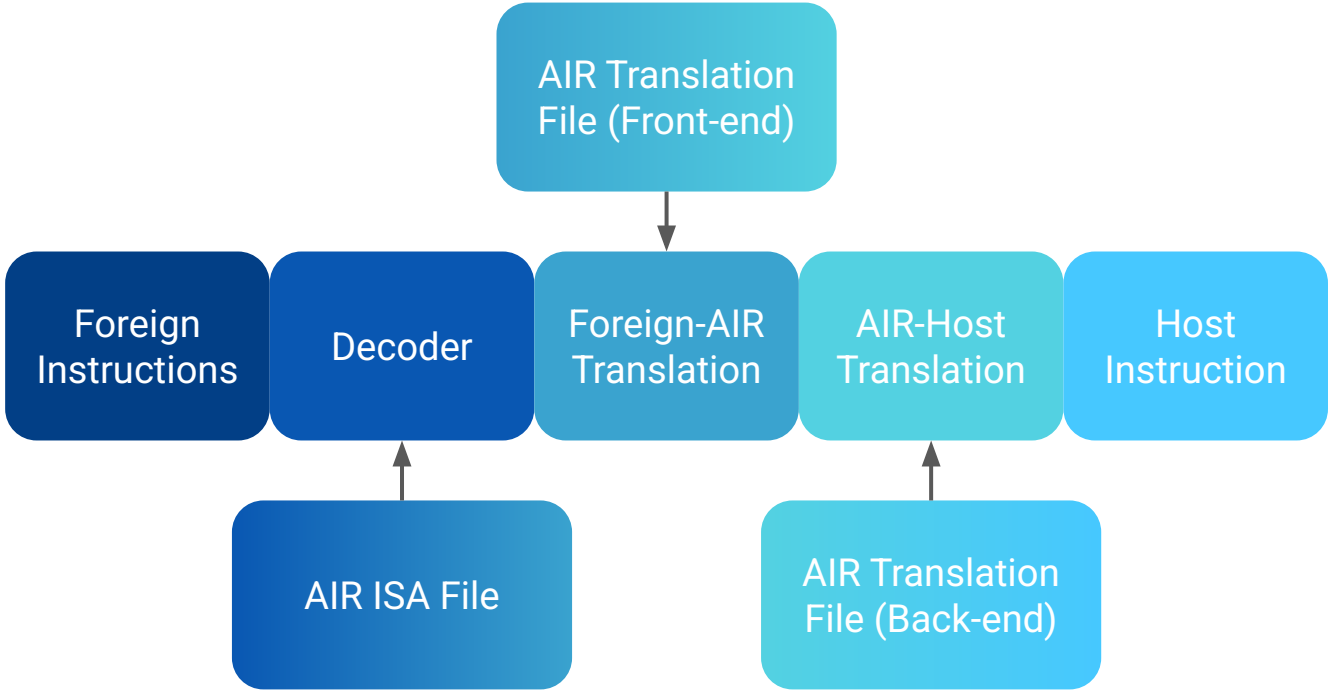
Specialised translator

HABITAT & AIR

HABITAT


AIR Translation File (Front-end)
Foreign Instructions
Decoder
Foreign-AIR Translation
AIR-Host Translation
Host Instruction
AIR ISA File
AIR Translation File (Back-end)

HABITAT



- **Fine-grained**
- **Lightweight**
- **Extensible**

Decoder - Effective Input

101	simm[11|4|9:8|10|6|7|3:1|5]			01	C.J rd, offset
110	simm[8|4:3]	rs1'	simm[7:6|2:1|5]	01	C.BEQZ rs1, rs2, offset
111	simm[8|4:3]	rs1'	simm[7:6|2:1|5]	01	C.BNEZ rs1, rs2, offset
000	0	rs1/rd≠ 0	nzuimm[4:0]	10	C.SLLI rd, rs1, imm

Decoder - Effective Input

101	simm[11|4|9:8|10|6|7|3:1|5]			01	C.J rd, offset
110	simm[8|4:3]	rs1'	simm[7:6|2:1|5]	01	C.BEQZ rs1, rs2, offset
111	simm[8|4:3]	rs1'	simm[7:6|2:1|5]	01	C.BNEZ rs1, rs2, offset
000	0	rs1/rd≠ 0	nzuimm[4:0]	10	C.SLLI rd, rs1, imm

```
{
    "name": "C.J",
    "format": "CJ",
    "pattern": [{"opcode": 1}, {"funct1": 5}],
    "immediate pattern": [[5, 1], [1, 3], [7, 1], [6, 1], [10, 1], [8, 2], [4, 1], [11, 1]],
    "signed immediate": 12
},
{
    "name": "C.BEQZ",
    "format": "CB",
    "pattern": [{"opcode": 1}, {"funct1": 6}],
    "immediate pattern": [[5, 1], [1, 2],  [6, 2], [3, 2], [8, 1]],
    "signed immediate": 9,
    "reg shift": 8
},
 {
    "name": "C.SLLI",
    "format": "CI",
    "pattern": [{"opcode": 2}, {"funct1": 0}],
    "condition": [{"rs1": 0, "op": "NE"}],
    "immediate pattern": [[0, 5], [5, 1]]
},
```

Decoder's Responsibility

- Pattern Matching
- Reordering
- Preprocessing

Generator's feature[1]

- Arbitrary Instruction Length
- Support Irregular ISA

[1] Katsumi Okuda and Haruhiko Takeyama. Decision tree generation for decoding irregular instructions. In 2016 Design, Automation Test in Europe Conference Exhibition (DATE), pages 1592–1597, 2016.

Foreign-AIR Translation - Effective Behaviour

```
WriteBack(
    (
        Literal(Register(2, 64, INT, true)),
        (
            BinaryOperator(
                ADD,
                Register(2, 64, INT, true),
                INT(-112, 32)),
            64, SignExtend
        )
    )
)
WriteBack(
    (
        Literal(Register(32, 64, INT, false)),
        (BinaryOperator(
            ADD,
            Register(32, 64, INT, false),
            INT(2, 64)),
        64, ZeroExtend)
    )
)
```

c.addi16sp

Description	Add the non-zero sign-extended 6-bit immediate to the value in the stack pointer (sp=x2), where the immediate is scaled to represent multiples of 16 in the range (-512,496).

AIR Intermediate Representation (IR)

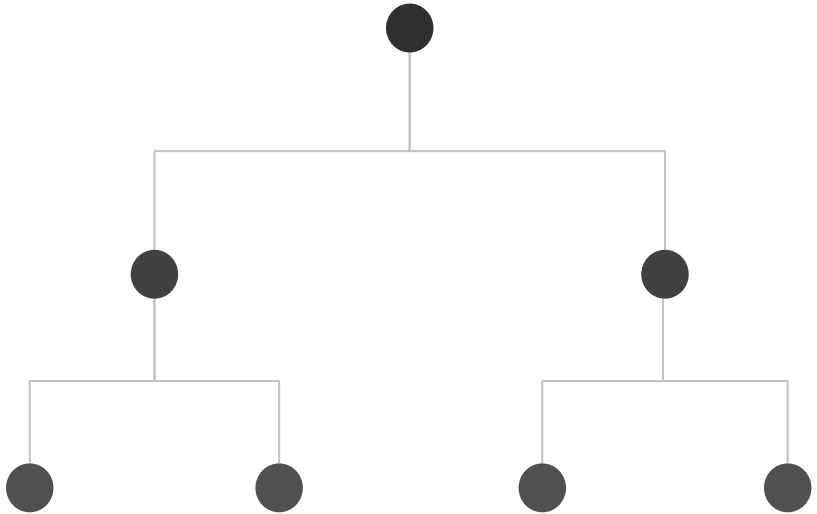
- **Tree-based IR**
- **0-5 Operands**
- **Architectural Information Extraction**
- **Unified Control / Data Flow representation**
- **Conditional Overwriting / Casting**

AIR-Host - Effective Implementation

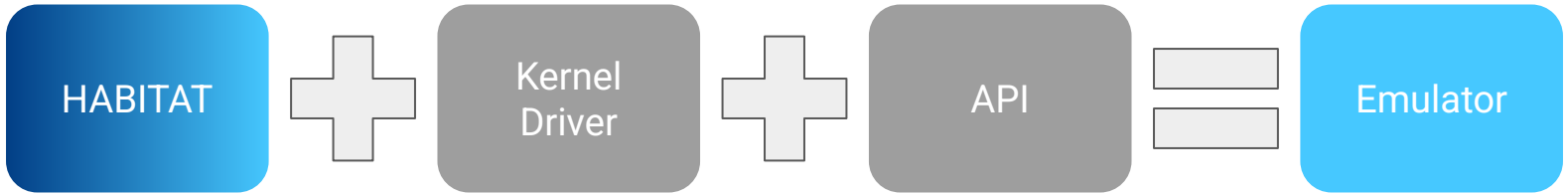
Translation Process

- Node-Instruction Mapping
- Register Allocation
- Optimisation through Bulk (Subtree) Mapping

```
WriteBack(
    (
        Literal(Register(32, 64, INT, false)),
        (BinaryOperator(
            ADD,
            Register(32, 64, INT, false),
            INT(2, 64)),
        64, ZeroExtend)
    )
)
```

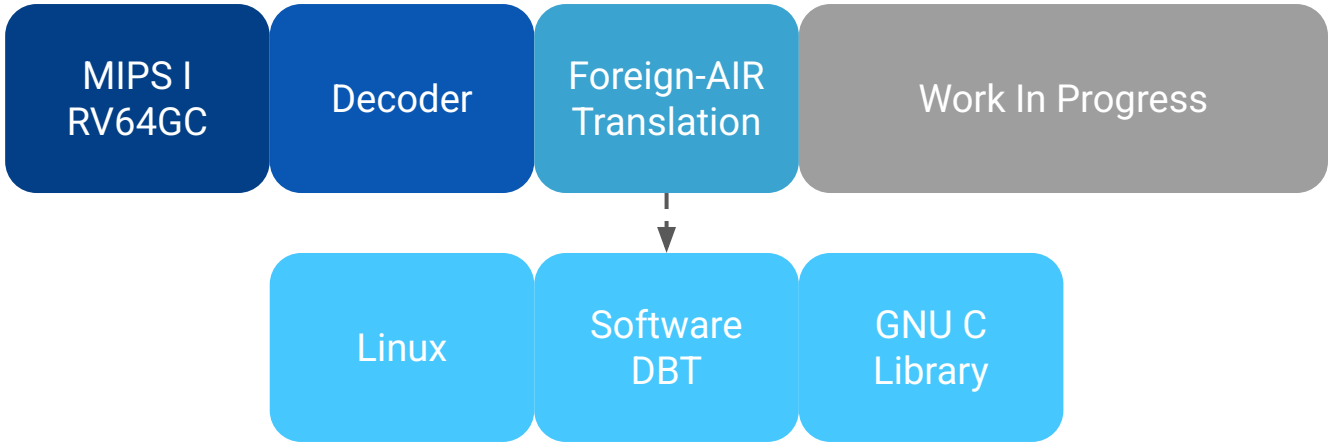


Runtime - Bring Everything Together



Result

One Small Step



- **Statically Linked**
- **Unprivileged Code**
- **Single-Threaded**

Challenges

Interesting Questions

Why not emulating an entire OS ?

- Accuracy of result > Accuracy of implementation

What if we want to support cross-OS emulation ?

- Application binary interface (ABI) differs a lot, even within the same OS

How about multi-threaded programs ?

- Theoretically possible
- Hard to model (old) memory models
- Harder to verify them

Next Steps

One Giant Leap

Hardware

- Procedural generation of translation pipeline
- Software-Hardware cross-validation
- Optimisation

Software

- Full RV64-x86 pipeline
- Linux runtime
- Position-Independent Code

Thank you