Cross-architecture Virtualisation

Tom Spink
Harry Wagstaff, Björn Franke

School of Informatics
University of Edinburgh

Virtualisation

Many of you will be familiar with same-architecture virtualisation (e.g. VirtualBox, Parallels, Xen, etc)

You may think of cross-architecture virtualisation as “simulation” or “emulation”

But, the goals for simulation are different to virtualisation.

Cross-architecture Virtualisation

Virtualising a guest architecture on a different host architecture, for example **AArch64** on **x86**.

Less interested in accurate simulation of architectural components, whilst remaining architecturally correct.

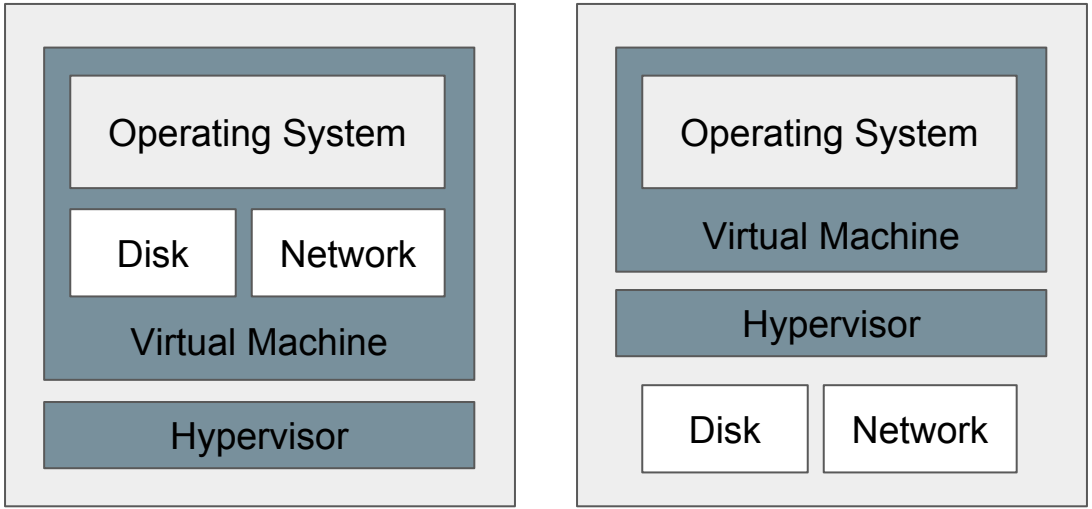
More interested in fast execution of guest architecture, whilst remaining functionally correct.

Useful for design space exploration, embedded system development, security research, running unmodified guest environments on faster host machines...

Cross-architecture Virtualisation

We still need to emulate the behaviour of guest architecture.

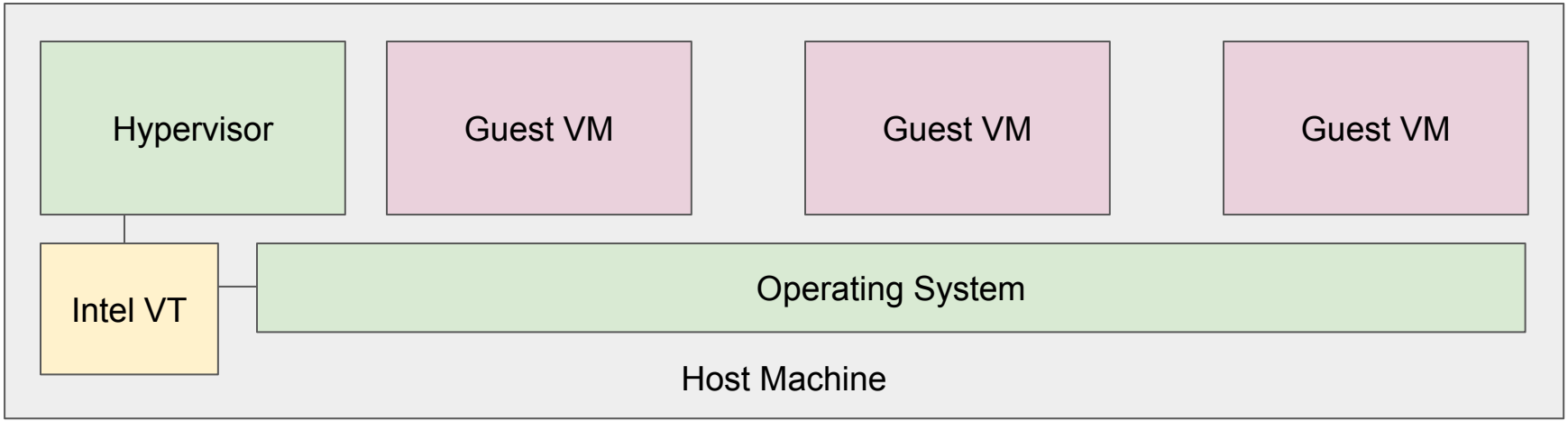
We could use para-virtualisation, e.g. VirtIO, but virtualising an unmodified system needs architectural components to behave correctly.



Hardware Assisted Virtualisation

Available for same-architecture virtualisation (e.g. Intel VT, AMD-V, ARM Virtualization Extensions)

Can we exploit these features for cross-architecture virtualisation?

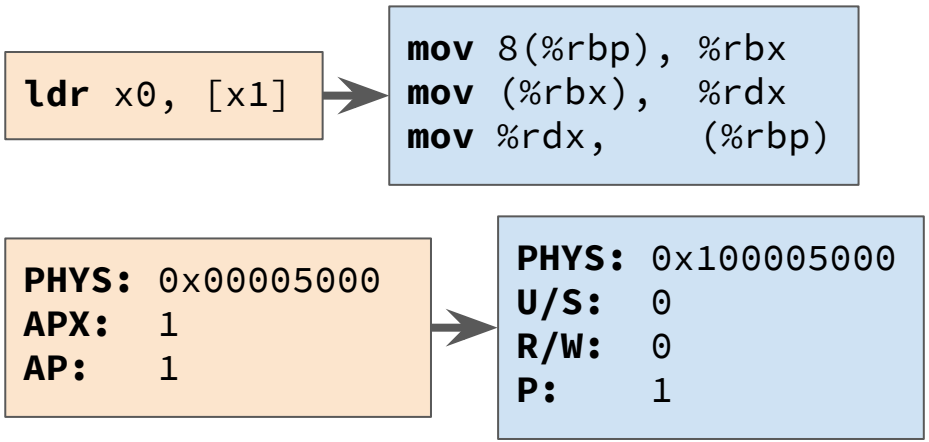


Captive

Captive is an implementation of a cross-architecture virtualisation hypervisor.

Its core goal is to map guest architectural behaviour to host architectural behaviour.

- Instruction mapping
- MMU mapping
- Device/IRQ mapping



Captive

- Both host and guest architecture are pluggable
  - Host machine requires KVM support
    - Also need to write a JIT backend!
  - Guest architecture requires patience!
- Architectural components are modular
  - Processor
  - Devices
  - (MMU)
- Supports instruction tracing
  - Quite fast!
  - Very useful for debugging
- Memory access tracing/cache simulation

Structure

Hypervisor (KVM)

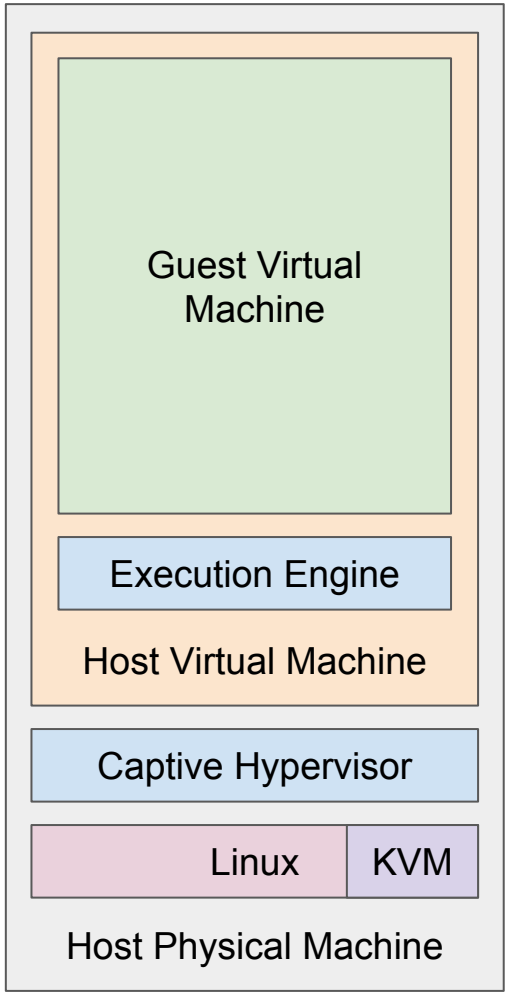
Runs as a user-mode program on a Linux host. Uses KVM to create a virtual machine instance.

Engine (Bare Metal)

A unikernel that implements the architectural mapping.

Guest (Unmodified)

The guest platform - typically a kernel to boot, and a file-system.

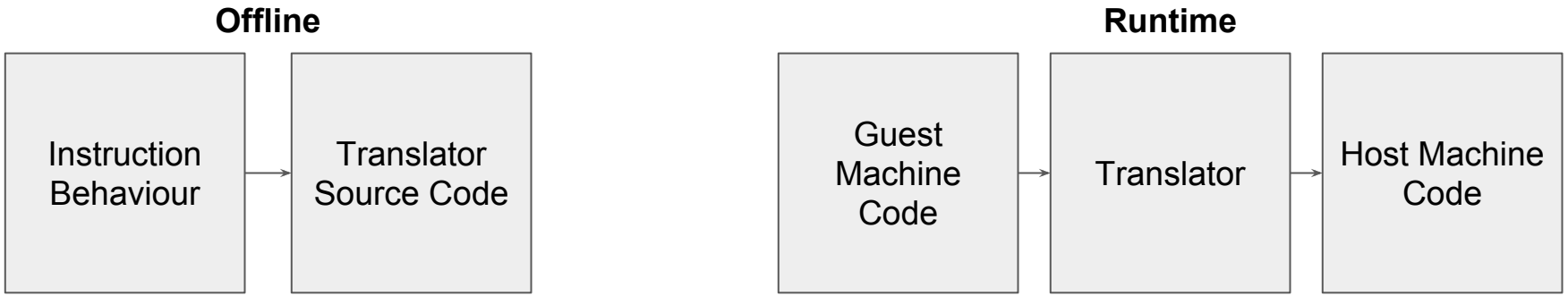


Instruction Virtualisation

Interpretation is **far** too slow; use Dynamic Binary Translation (DBT)

Captive implements a high-speed domain-specific JIT compiler, generated from a high-level architecture description language. Very fast translation, compilation and optimisation.

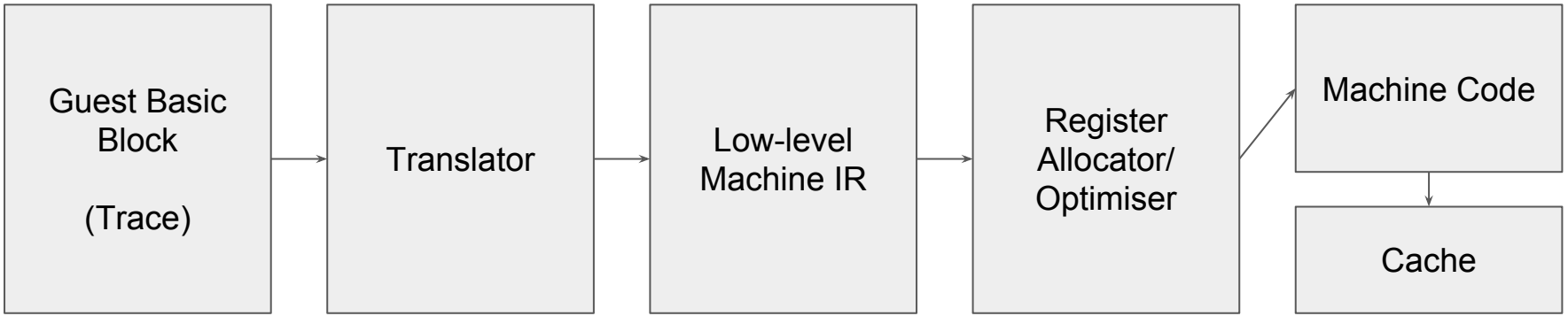
High-level C-like descriptions are used to describe guest instruction behaviours.



Instruction Virtualisation

At JIT compilation time, each guest instruction in a basic-block is asked to translate itself directly into x86.

Block translations are cached by physical address -- no need for invalidation on page table changes, only need to beware of self-modifying code.



Instruction Virtualisation - Self-modifying Code

Since translations are cached, changes to the guest executable code need to be detected, and translations invalidated.

Captive marks all guest memory pages as read-only after compiling a block (via the top-level page table entry), so any writes will trigger self-modifying code detection.

Unfortunately, the entire guest VA space must be protected, since multiple VAs can point to the same PA.

SMC is practically non-existent in early (Linux) system runtime, only appears later when OS needs memory for newer programs.

Memory Virtualisation

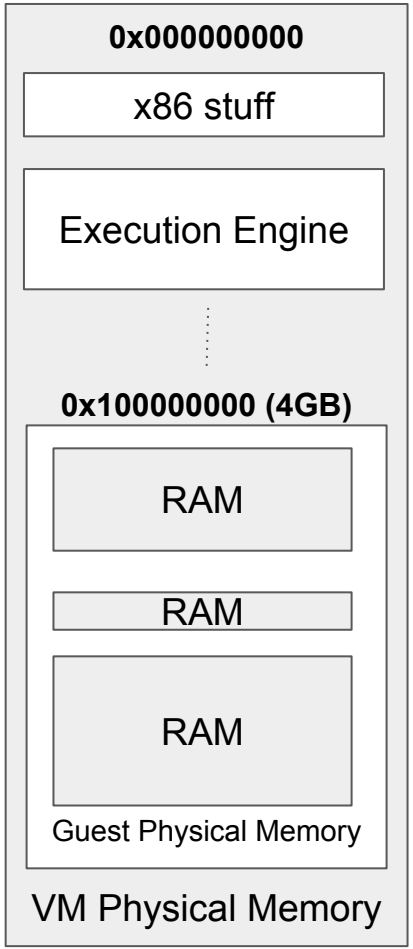
The guest platform defines it's physical memory layout.

Physical memory regions are instantiated that mirror guest RAM areas.

No MMU: Build 1-1 mapping of Guest Physical Memory to VM Virtual Memory.

MMU: Maintain virtual-to-physical mappings in VM that represent virtual-to-physical mappings in guest.

An ARM page table entry gets turned into a similar x86 page table entry. Hardware MMU virtualisation!



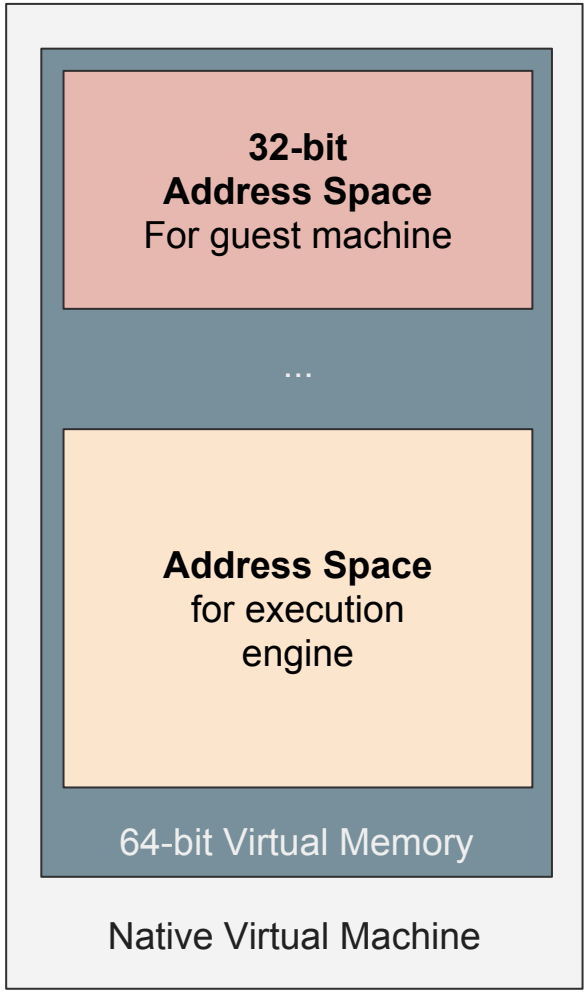
32-bit on 64-bit

Easy, 32-bit virtual memory area fits within lower portion of 64-bit address space.

Efficient mapping of guest MMU to host MMU, translate guest page table entries almost directly to host page table entries.

Lazily perform translations (on guest memory accesses). Trap guest TLB flushes to invalidate mapped entries.

Fast invalidation by clearing top-level page table entry (PML4E)



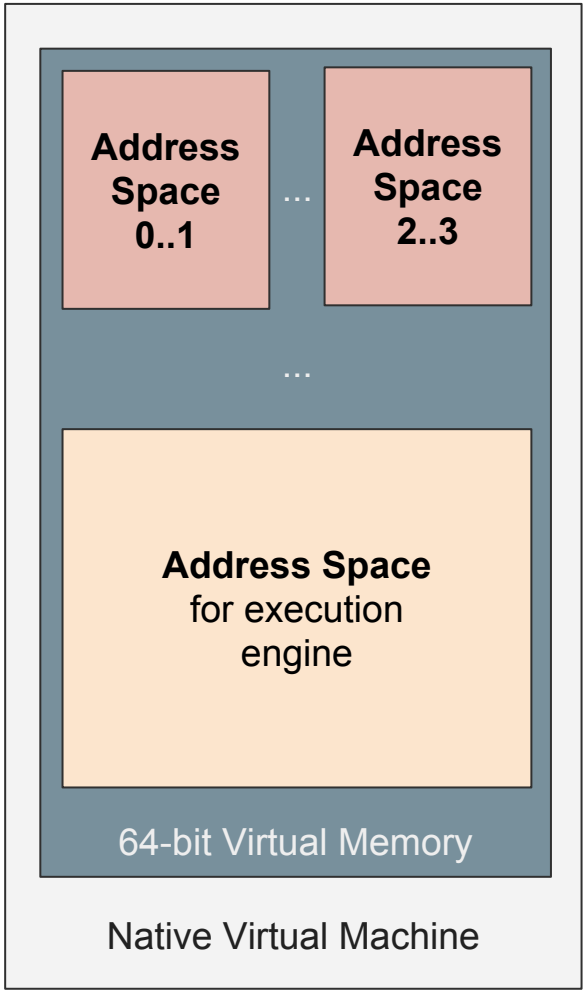
64-bit on 64-bit

Tricky, we need some 64-bit address space for the execution engine, JITted code, book-keeping, etc.

Also, 64-bit isn't really 64-bit - it's 48-bit on x86 and on ARMv8 the situation is complicated by having two address spaces.

Possible solution: Have four page tables, representing slices of the guest 64-bit address space, and switch these depending on the memory address being accessed.

TLB penalty significant, although PCIDs help here.



Memory Instruction Virtualisation

ARMv7 Instruction:

```
ldr r3, [r4]
```

x86 Translated Sequence:

```
mov 0x10(%rdi), %eax
mov (%rax), %eax
mov %eax, 0xc(%rdi)
add $0x4, %r15
```

ARMv8 Instruction:

```
ldr x0, [x1]
```

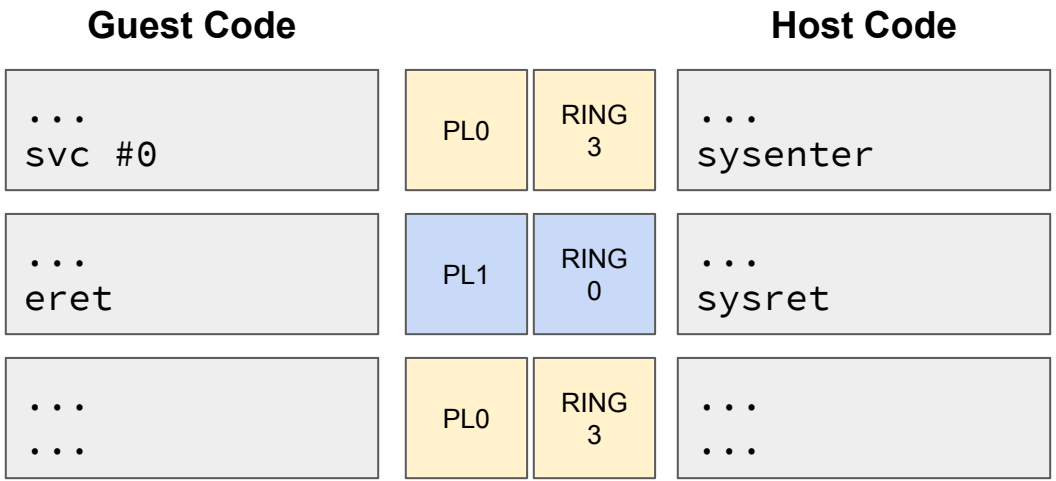
x86 Translated Sequence:

```
    mov    0x8(%rbp), %rcx
    movabs $0x7fffffffffff, %rax
    and    %rcx, %rax
    shr    $0x2f, %rcx
    and    $0x3, %ecx
    cmp    %cl, %fs:0x68
    je     1f
    mov    %cl, %fs:0x68
    mov    %fs:0x70(%riz,%rcx,8), %rcx
    mov    %rcx, %cr3
1:  mov    (%rax), %rax
    mov    %rax, (%rbp)
    lea    0x4(%r15), %r15
```

Privilege Levels

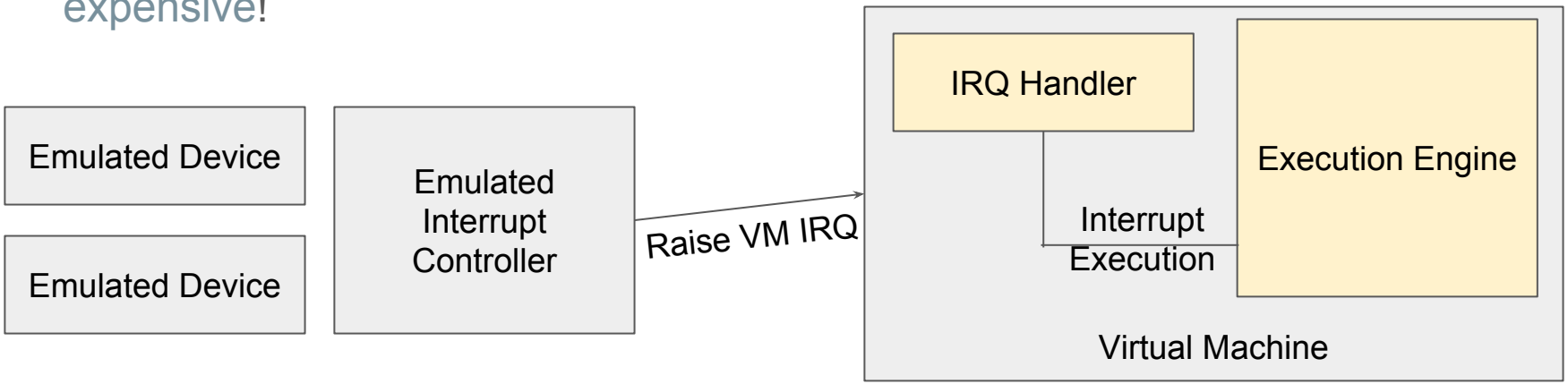
Take advantage of x86 privilege levels (only ring 0 and ring 3 in long mode!) and track guest execution state with these.

Extra memory protection flags for free!



Device/IRQ Virtualisation

- Para-virtualisation with VirtIO - easy, just pass it through.
- Real device virtualisation - not so easy, must emulate devices in software.
- Devices that are similar (e.g. timers) can map behaviour.
- Devices that do not exist must be implemented in software.
- Possibly significant communication overhead - hypercalls/MMIO traps are expensive!



Current/Future Work

Floating-point implementation - tricky because ARM and x86 behave differently.

(Some problems are in fact philosophical rather than technical, but still a lot of engineering required to implement this mapping)

Intel MPX, interesting feature for fast address range checking, but doesn't quite solve the problem of switching page tables.

Intel MPK, could potentially solve problem (but need access to hardware to evaluate)

Intel PT, possibly exploit this to accelerate guest tracing

GDB integration, engine side for debugging core + JIT, guest side for debugging guest execution

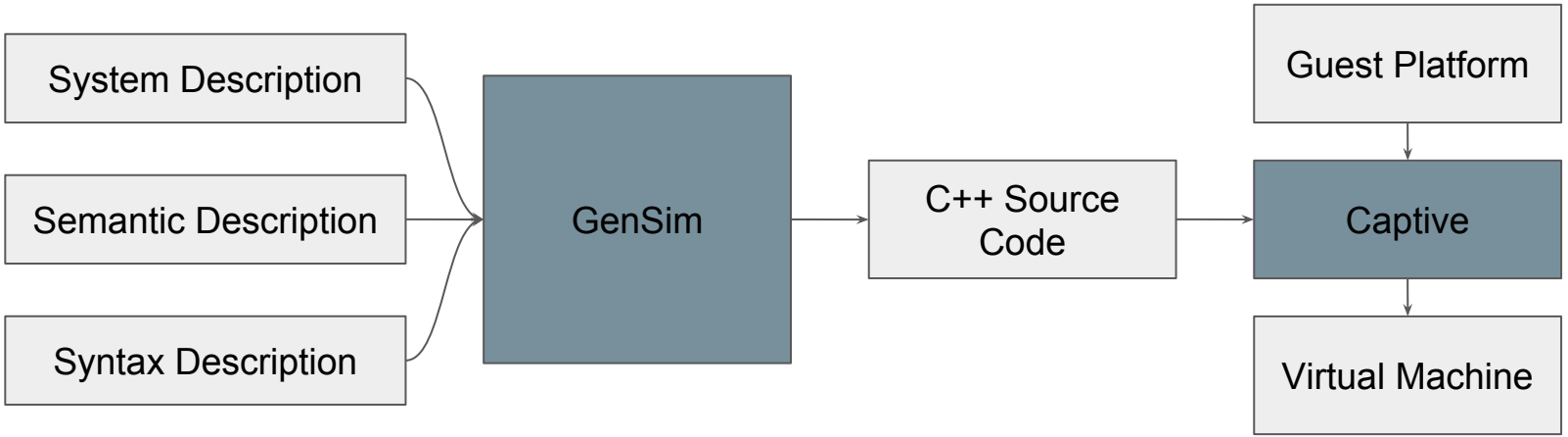
Removal of custom hypervisor - push device emulation into execution engine, to keep engine self-contained.

GenSim

GenSim is our flagship Simulator Generation tool

https://gensim.org

It is now available as open-source software, with Captive soon to follow.



Edinburgh Simulation Group

Tom Spink, Harry Wagstaff, Kuba Kaszyk, Bruno Bodin, Bjoern Franke

GenSim - Simulator Generation Tool

ArchSim - High Speed Simulation

Captive - High Speed Virtualisation

GPUSim - Unmodified GPU Simulation

SimBench - Simulator Benchmarking

ISPASS'18 Tutorial

Any Questions?

Dr Tom Spink

tspink@inf.ed.ac.uk
https://homepages.inf.ed.ac.uk/tspink

School of Informatics, University of Edinburgh

With Dr Harry Wagstaff & Dr Bjoern Franke

hwagstaf@inf.ed.ac.uk, bfranke@inf.ed.ac.uk