

# Supporting Higher-Order Virtualization

Jeremy Singer, Chris Kirkham and Ian Watson

University of Manchester, UK

# Outline

- What is virtualization?
- What is higher-order virtualization?
- Why is HOV ...
  - interesting?
  - inefficient?
- How can HOV be improved?

# What is virtualization?

*a framework for partitioning a single (physical) resource into multiple (virtual) resources, or vice versa*

# virtualization examples

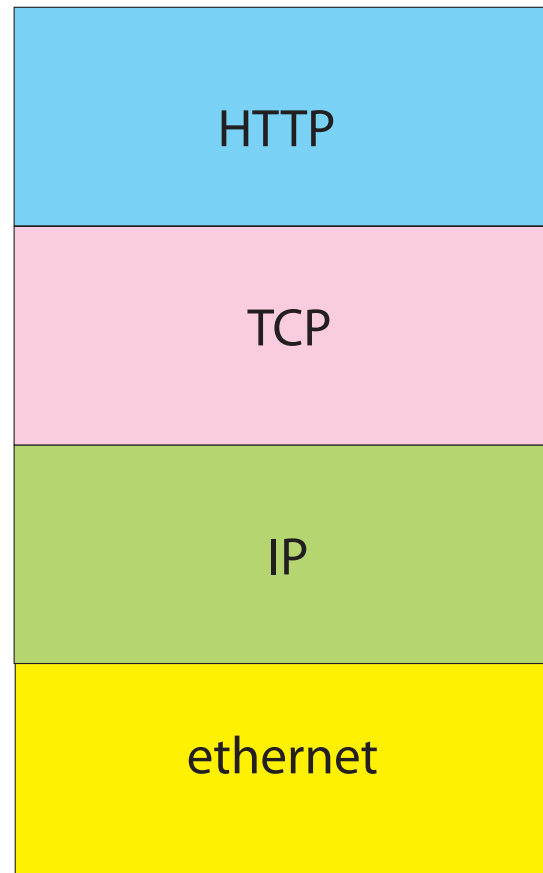
- discs
  - logical partitions (volumes)
  - RAID
- processors
  - VMWare
  - PVM

# What is HOV?

same as virtualization, only it partitions *virtual* resources rather than *physical* resources

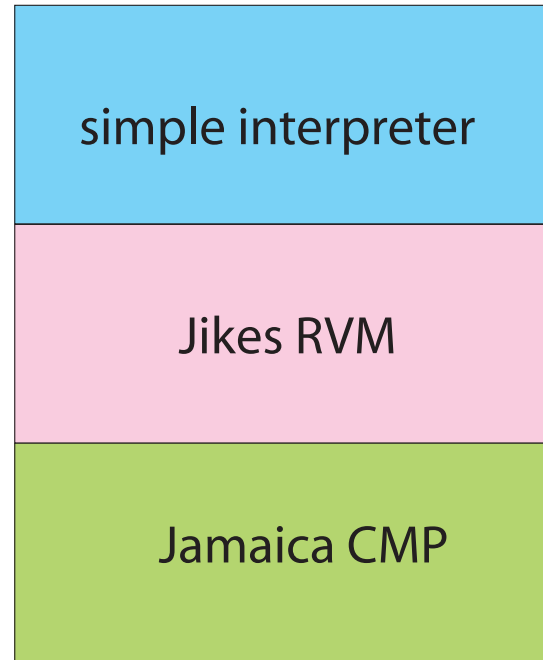
# HOV example 1

- network protocol stack



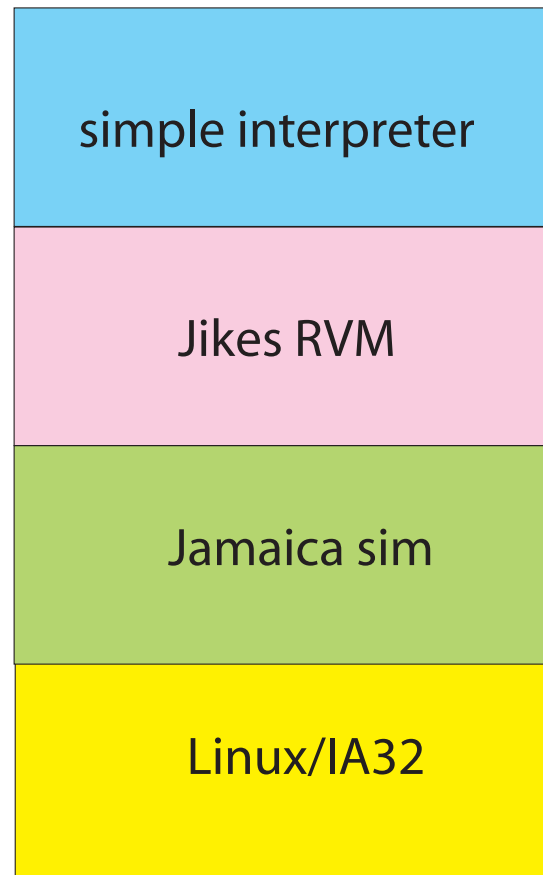
# HOV example 2

● stack of VMs



# Actually ...

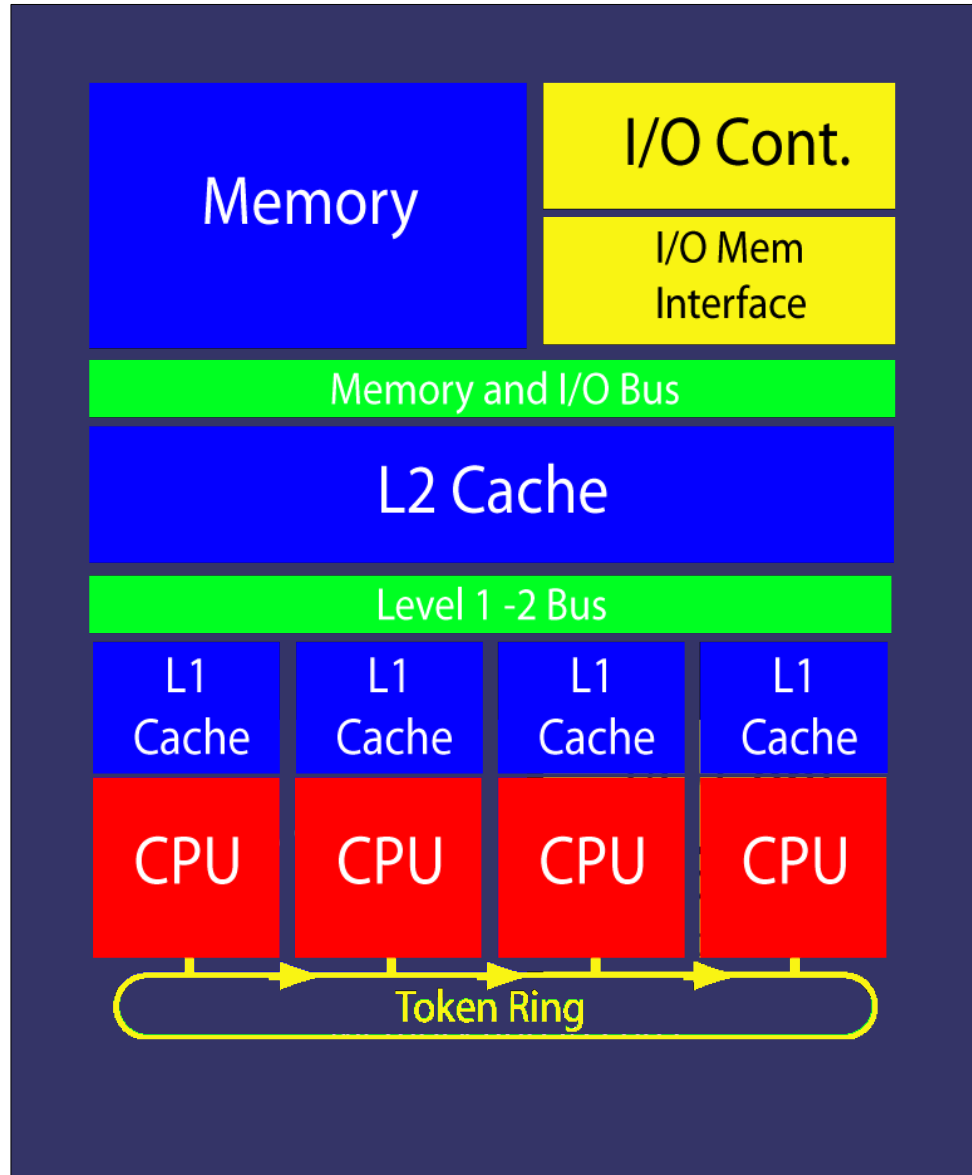
- stack of VMs with *simulated* layers



# Why is HOV interesting?

- current fashion: Java, .NET, Parrot, ...
- plenty of motivations
  - separation of concerns
  - portability
  - legacy code support
  - flexibility
- architectural support available: increased *parallelism*
  - Jamaica, prototype commodity chip-multiprocessor

# Jamaica architecture



# Why is HOV inefficient?

- overhead of translation from one instruction set to another
- reduce inefficiency by well-managed trapdoors through VM stack layers

# How can HOV be improved?

- standard interpreter techniques
  - switch-based interpretation
  - threaded interpretation
  - stack caching
  - method inlining
- speculative parallel interpretation
  - loop-level speculation
  - method-level speculation
  - alternatives (?) . . .

# Interpreter source code

```
while (pc < MAX_ADDRESS) {  
    byte instr = getByteAt(pc++);  
    switch(instr) {
```

fetch/decode

```
    case Bytecodes.PCB:  
        // push constant byte from instr stream  
        dataB = getByteAt(pc++);  
        sp += 4;  
        push(sp, (int)dataB);  
        break;
```

execute

```
    case Bytecodes.ADD:  
        // standard integer arithmetic  
        second = pop(sp);  
        sp -= 4;  
        first = pop(sp);  
        sp -= 4;  
        answer = first + second;  
        sp += 4;  
        push(sp, answer);  
        break;
```

execute

```
    case ...:
```

execute  
execute  
execute

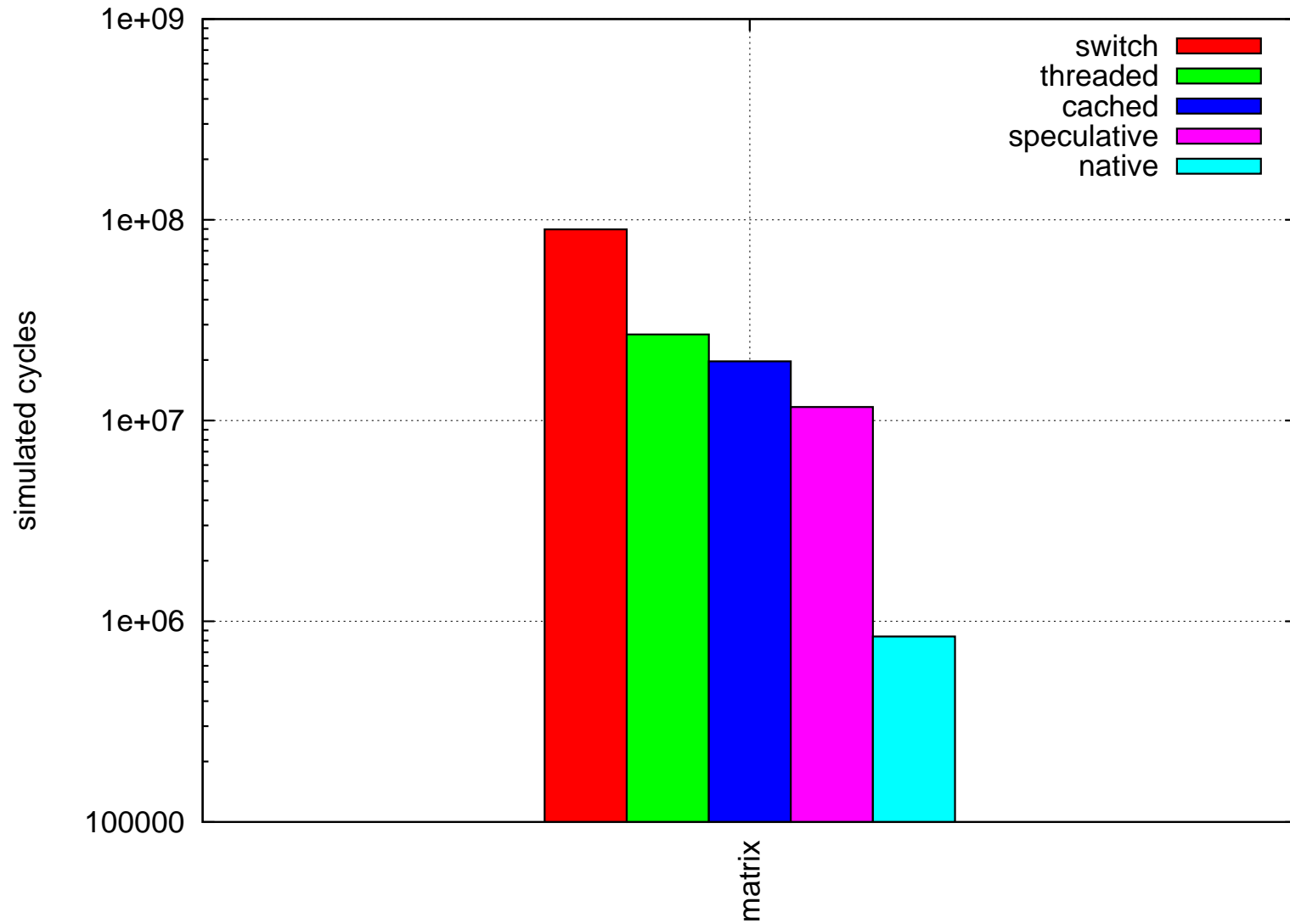
```
    case ...:
```

```
    case ...:
```

```
    }
```

```
}
```

# Results



# Future Work

- compiler to target simple bytecode
- improve speculation support
- add speculation at lowest level, in Jamaica hardware

# Conclusions

- Disadvantages of this work
  - software inefficiency
  - cannot run Java programs speculatively
- Advantages
  - presents HOV paradigm
  - shows how to speculate on non-spec hardware
  - simple system to prototype ideas