Gathering Memory Hierarchy Statistics in QEMU

Clément Deschamps, Mark Burton, Eric Jenn, Frédéric Pétrot

GreenSocs, IRT Saint Exupéry,Univ. Grenoble

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

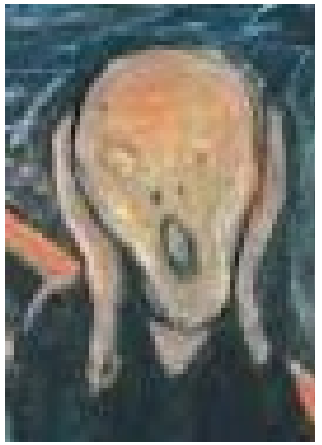
Can I optimize my code for the memory hierarchy?

- Just after “Does my code work”, this seems to be the biggest preoccupation for embedded software.

- And we have answered – till now, by counting accesses.

But now…

CACHE COHERENCY



Takeaway message:

QEMU + *PV+T* =

Qemu: defacto standard Virtualizer

Architectures	CPU's	Commits	Contributors	Lines of code
18	1100	43000	1000	989,863

- GreenSocs is a contributor:
  Xilinx components, power, reset, clocks, Multi-Core TCG Kernel…

Regular committers from many organizations

Qbox, Connecting Qemu and SystemC

QBox

In A Box

- Connect QEMU and SystemC
- Can use any QEMU model within SystemC

- How is synchronization done:
  - WHOLE OTHER DISCUSSION…
  - **Come to the SystemC Evolution Day !**

“PV+T” timewarp…..

- Sooo long ago, sooo far away, 3 people tried to convince the TLM WG (!) that they should support a modeling style they called “PV+T”
  - they failed!


Frank Ghenassia


me!


Jean-Michel Fernandez

- The idea was simple, simulate your device, and then add timing.
- Run a functional model, record all the information you need, and feed that into a model of the architecture to evaluate the time.

“PV+T” timewarp……

Jérôme Cornet, Florence Maraninchi, Laurent Maillet-Contoz: A Method for the Efficient Development of Timed and Untimed Transaction-Level Models of Systems-on-Chip. DATE 2008: 9-14

- Separation of Functional and Non-Functional Aspects in Transactional Level Models of Systems-on-Chip, Jerome Cornet, Apr 25, 2008, Univ. Grenoble

Function separate from time?

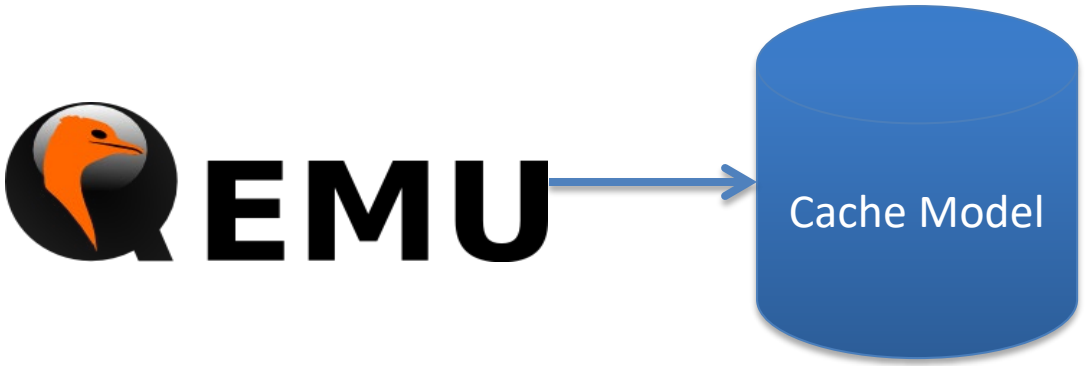
Functional Model (PV)

H/W Model (pipelines etc) - T

NOTICE:
Information flows one way !
(The timing model does not even have to execute at the same time, it can post-process)

The plan

- Capture enough information from QEMU to feed into a cache model
  - (No need to know the ‘actual’ data!)
  - Need to know all D-Side accesses
  - Need to know all I-Side accesses



- Run the cache model in a **separate host threads**

- Model L1, (and L2 Cache coherency).

QEMU - MTTCG

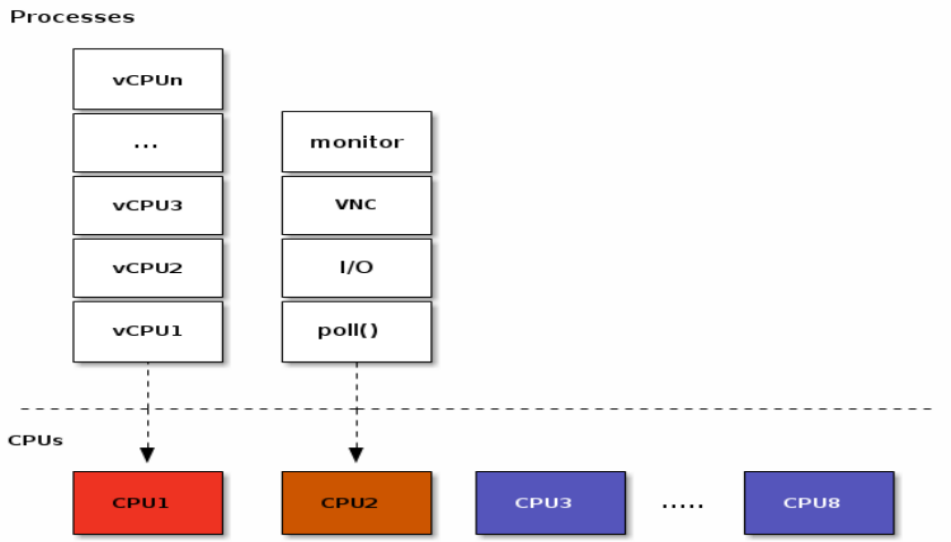
Multi Thread Tiny Code Generator

GreenSocs contribution to QEMU, started in 2014.

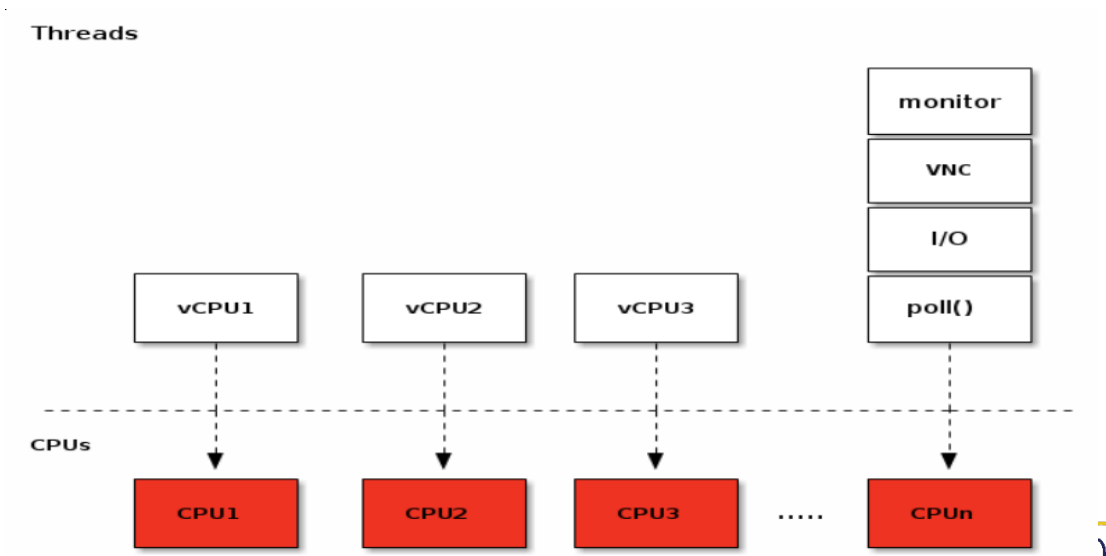
Can potentially generate a lot of *concurrent* memory accesses.

(just like real hardware)

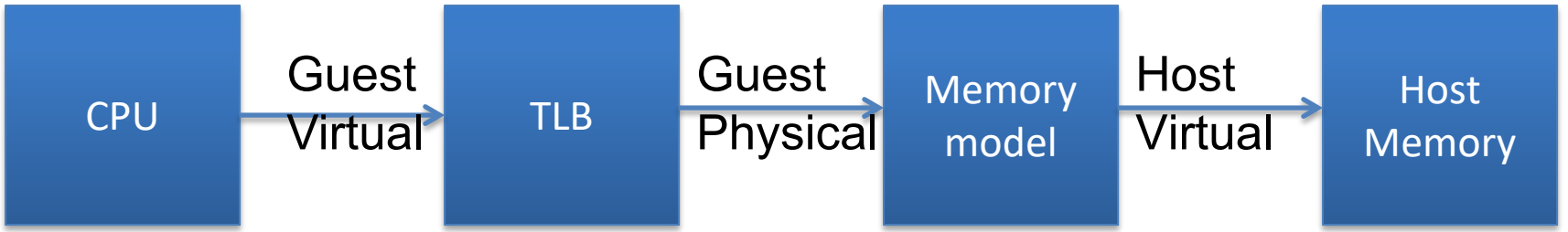
Single TCG (Round Robin)



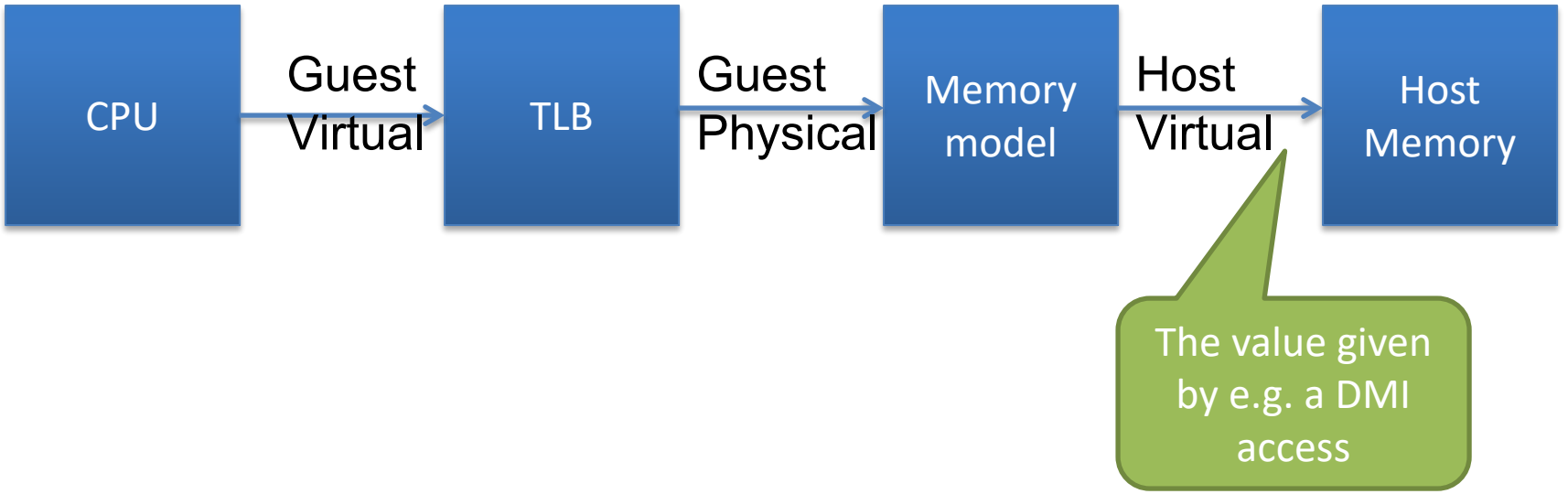
MTTCG (1 vCPU per host core)



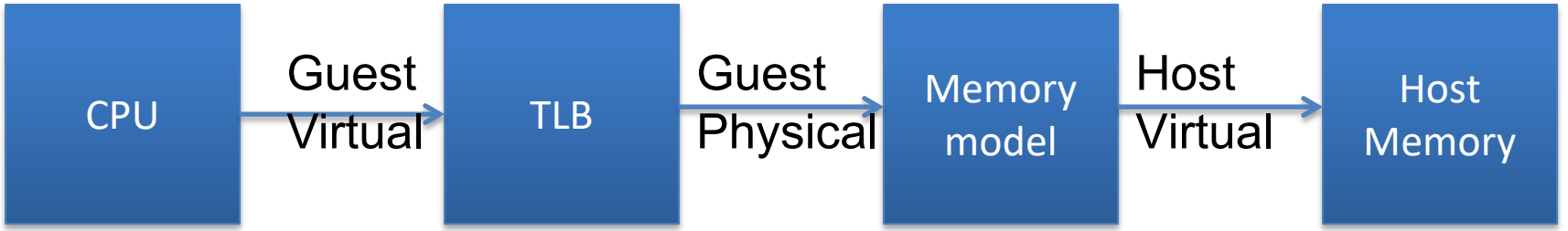
Memory paths



Memory paths

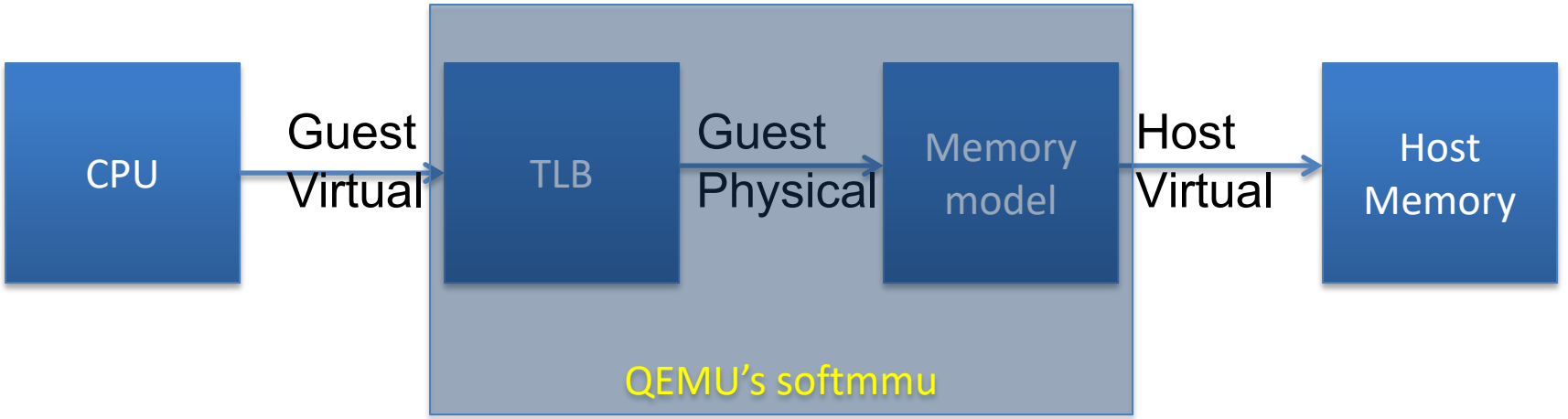


Memory paths



Same for both I side and D side

Memory paths



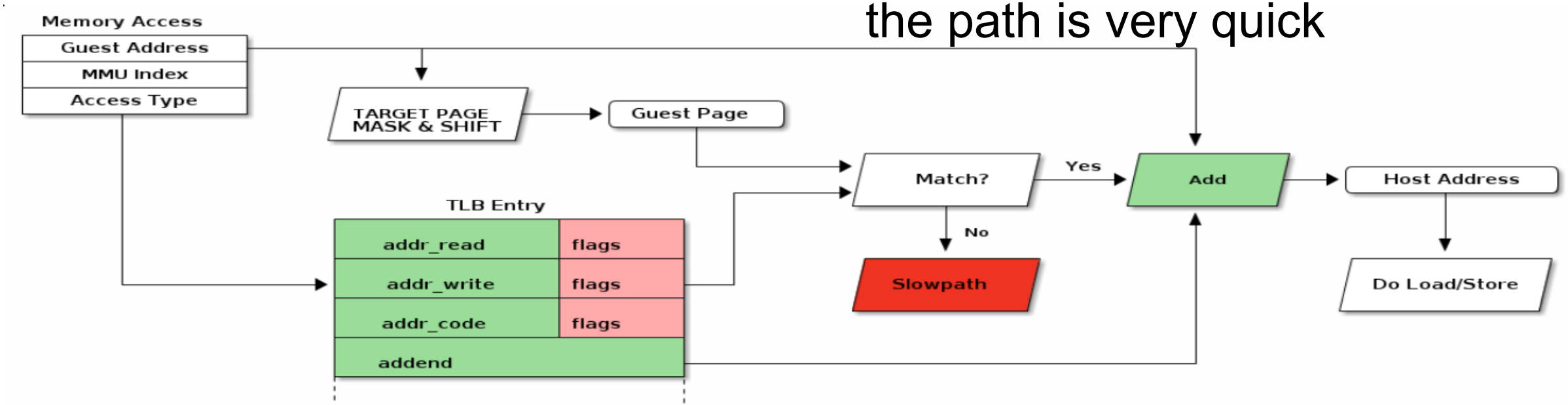
QEMU - Memory Access

ARM Load Example:

Basic Block

IN: ldr r1, [fp]

If the entry is in the Softmmu TLB, the path is very quick

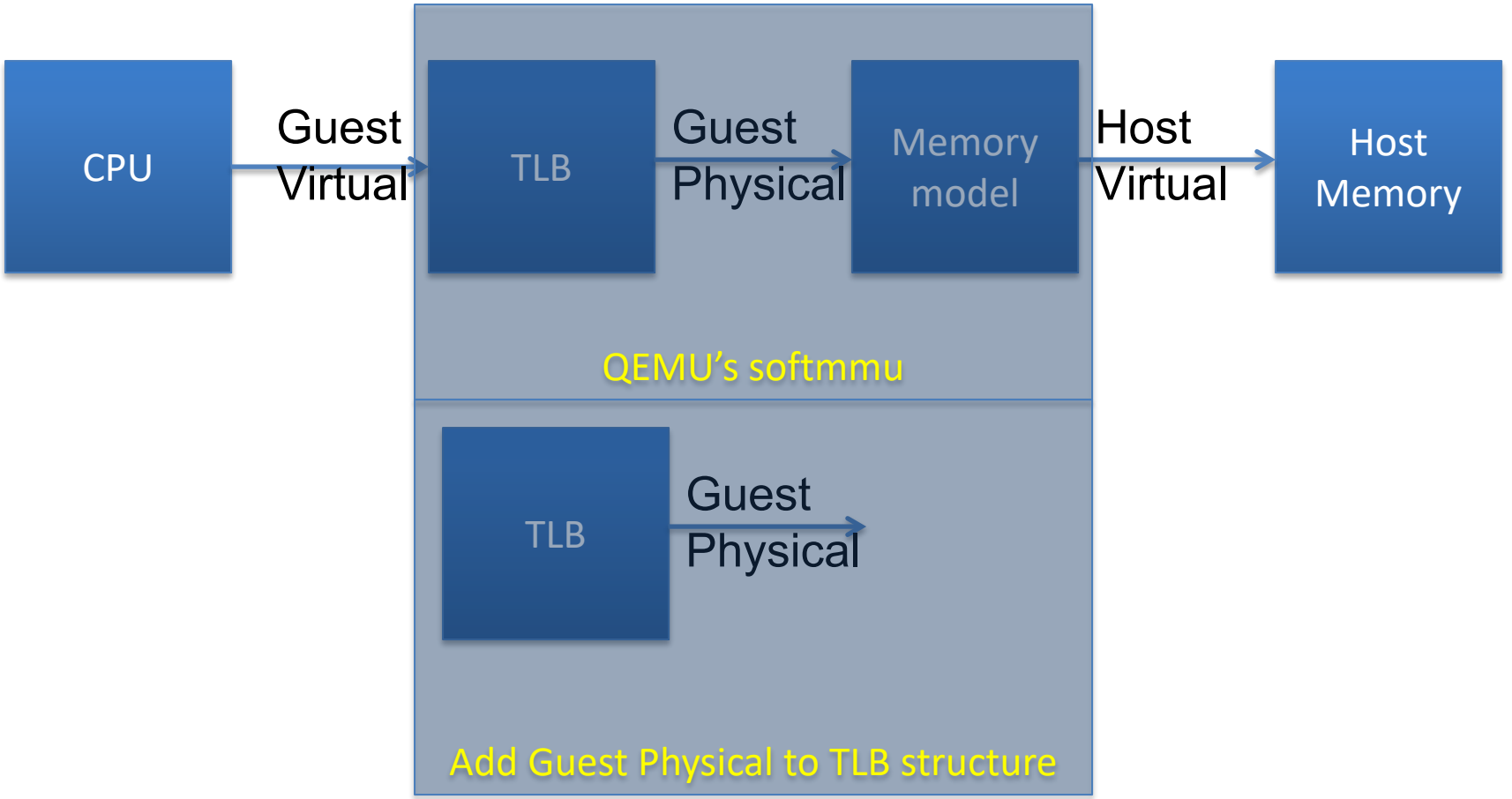


Alex Bennée, Towards multi-threaded TCG, KVM Forum 2015

Path specifics

- On the I side, we need both Virtual and Physical addresses (e.g. A53 I-side caches are virtually indexed and physically tagged!).
  - Notice that on the I-Side, the QEMU softMMU does a direct translation from Guest virtual to Host virtual. We have added Guest physical addresses to this structure.
  - But, this can be calculated as 'JIT Compile time', rather than for each access – so we generate I side accesses per TB. (Handled by a new field in TLB structure).
- D-side slow : simply modify slow path helper.
- D-side fast path we have to insert code – this is harder . . .

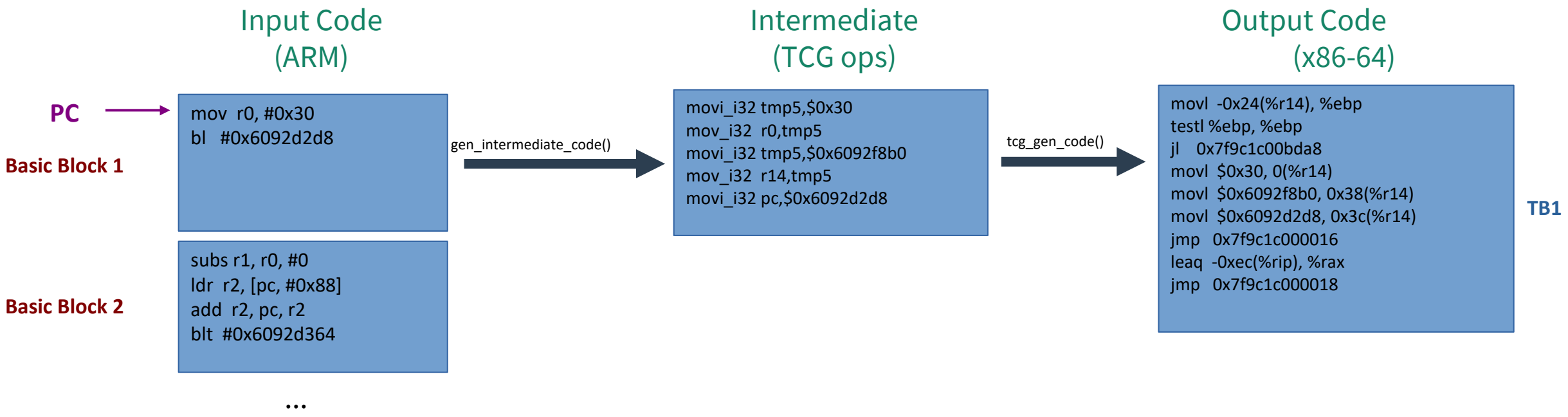
Memory paths



Path specifics

- On the I side, we need both Virtual and Physical addresses (e.g. A53 I-side caches are virtually indexed and physically tagged!).
  - Notice that on the I-Side, the QEMU softMMU does a direct translation from Guest virtual to Host virtual. We have added Guest physical addresses to this structure.
  - But, this can be calculated as 'JIT Compile time', rather than for each access – so we generate I side accesses per TB. (Handled by a new field in TLB structure).
- D-side slow : simply modify slow path helper.
- D-side fast path we have to insert code – this is harder . . .

QEMU – DBT (Dynamic Binary Translation)



A **Basic Block** corresponds to a list of instructions terminated by a branch instruction.

A Translation Block (**TB**) is the translation of a basic block into host instructions.

QEMU - Fast Path / Slow Path

Input Code (ARM)

Basic Block

IN: ldr r1, [fp]

Output Code (x86-64)

```
OUT: [size=128]
0x7f70c0012300:  41 8b 6e dc           movl    -0x24(%r14), %ebp
0x7f70c0012304:  85 ed                 testl   %ebp, %ebp
0x7f70c0012306:  0f 8c 41 00 00 00     jl      0x7f70c001234d
0x7f70c001230c:  41 8b 6e 2c           movl    0x2c(%r14), %ebp
0x7f70c0012310:  8b fd                 movl    %ebp, %edi
0x7f70c0012312:  8d 75 03              leal    3(%rbp), %esi
0x7f70c0012315:  c1 ef 05              shrl    $5, %edi
0x7f70c0012318:  81 e6 00 fc ff ff     andl    $0xfffffc00, %esi
0x7f70c001231e:  81 e7 e0 1f 00 00     andl    $0x1fe0, %edi
0x7f70c0012324:  49 8d bc 3e 00 6e 00 00  leaq    0x6e00(%r14, %rdi), %rdi
0x7f70c001232c:  3b 37                 cmpl    0(%rdi), %esi
0x7f70c001232e:  8b f5                 movl    %ebp, %esi
0x7f70c0012330:  0f 85 23 00 00 00     jne     0x7f70c0012359
0x7f70c0012336:  48 03 77 10           addq    0x10(%rdi), %rsi
0x7f70c001233a:  8b 2e                 movl    0(%rsi), %ebp
0x7f70c001233c:  41 89 6e 04           movl    %ebp, 4(%r14)
0x7f70c0012340:  41 c7 46 3c ec c9 92 60  movl    $0x6092c9ec, 0x3c(%r14)
0x7f70c0012348:  e9 c9 dc fe ff        jmp     0x7f70c0000016
0x7f70c001234d:  48 8d 05 ef fe ff ff  leaq    -0x111(%rip), %rax
0x7f70c0012354:  e9 bf dc fe ff        jmp     0x7f70c0000018
0x7f70c0012359:  49 8b fe              movq    %r14, %rdi
0x7f70c001235c:  ba 23 00 00 00        movl    $0x23, %edx
0x7f70c0012361:  48 8d 0d d4 ff ff ff  leaq    -0x2c(%rip), %rcx
0x7f70c0012368:  ff 15 0a 00 00 00     callq   *0xa(%rip)
0x7f70c001236e:  8b e8                 movl    %eax, %ebp
0x7f70c0012370:  e9 c7 ff ff ff        jmp     0x7f70c001233c
```

QEMU - Fast Path / Slow Path

Input Code (ARM)

Basic Block

IN: ldr r1, [fp]

Output Code (x86-64)

```
OUT: [size=128]
Prologue:
0x7f70c0012300:  41 8b 6e dc          movl   -0x24(%r14), %ebp
0x7f70c0012304:  85 ed                testl  %ebp, %ebp
0x7f70c0012306:  0f 8c 41 00 00 00    jl     0x7f70c001234d
0x7f70c001230c:  41 8b 6e 2c          movl   0x2c(%r14), %ebp

TLB Load:
0x7f70c0012310:  8b fd                movl   %ebp, %edi
0x7f70c0012312:  8d 75 03             leal   3(%rbp), %esi
0x7f70c0012315:  c1 ef 05             shrl   $5, %edi
0x7f70c0012318:  81 e6 00 fc ff ff    andl   $0xfffffc00, %esi
0x7f70c001231e:  81 e7 e0 1f 00 00    andl   $0x1fe0, %edi
0x7f70c0012324:  49 8d bc 3e 00 6e 00 00  leaq  0x6e00(%r14, %rdi), %rdi
0x7f70c001232c:  3b 37                cmpl   0(%rdi), %esi
0x7f70c001232e:  8b f5                movl   %ebp, %esi
0x7f70c0012330:  0f 85 23 00 00 00    jne    0x7f70c0012359

Fast Path:
0x7f70c0012336:  48 03 77 10          addq   0x10(%rdi), %rsi
0x7f70c001233a:  8b 2e                movl   0(%rsi), %ebp

Epilogue:
0x7f70c001233c:  41 89 6e 04          movl   %ebp, 4(%r14)
0x7f70c0012340:  41 c7 46 3c ec c9 92 60  movl  $0x6092c9ec, 0x3c(%r14)
0x7f70c0012348:  e9 c9 dc fe ff       jmp    0x7f70c0000016
0x7f70c001234d:  48 8d 05 ef fe ff ff  leaq   -0x111(%rip), %rax
0x7f70c0012354:  e9 bf dc fe ff       jmp    exit_tb

Slow Path:
0x7f70c0012359:  49 8b fe             movq   %r14, %rdi
0x7f70c001235c:  ba 23 00 00 00       movl   $0x23, %edx
0x7f70c0012361:  48 8d 0d d4 ff ff ff  leaq   -0x2c(%rip), %rcx
0x7f70c0012368:  ff 15 0a 00 00 00    callq  *0xa(%rip)
0x7f70c001236e:  8b e8                movl   %eax, %ebp
0x7f70c0012370:  e9 c7 ff ff ff       jmp    0x7f70c001233c
```

TLB Hit

TLB Miss

Go back to execution

Implementation: Record Load/Store

Extra code for each TLB lookup (even on the fast path!)

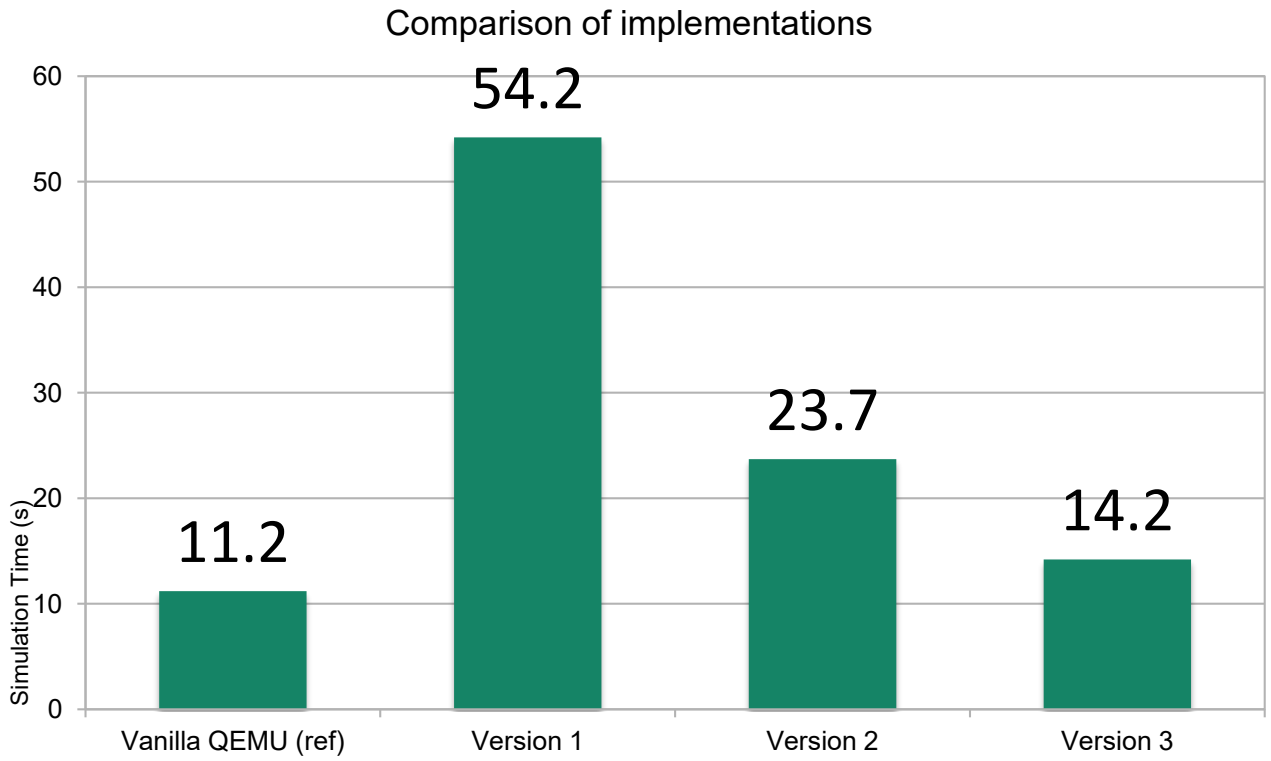
```
[...]

TB Load:
0x7f0404013690:  8b fd              movl    %ebp, %edi
0x7f0404013692:  8d 75 03           leal    3(%rbp), %esi
0x7f0404013695:  c1 ef 05           shrl    $5, %edi
0x7f0404013698:  81 e6 00 fc ff ff  andl    $0xfffffc00, %esi
0x7f040401369e:  81 e7 e0 1f 00 00  andl    $0x1fe0, %edi
0x7f04040136a4:  49 8d bc 3e 00 6e 00 00  leaq    0x6e00(%r14, %rdi), %rdi
0x7f04040136ac:  3b 37              cmpl    0(%rdi), %esi
0x7f04040136ae:  8b f5              movl    %ebp, %esi
0x7f04040136b0:  0f 85 67 00 00 00  jne     0x7f040401371d

Fast Path:
0x7f04040136b6:  48 03 77 10        addq    0x10(%rdi), %rsi
0x7f04040136ba:  41 51              pushq   %r9
0x7f04040136bc:  41 50              pushq   %r8
0x7f04040136be:  4d 8b 8e f8 9a 01 00  movq    0x19af8(%r14), %r9
0x7f04040136c5:  4d 6b c1 09        imulq   $9, %r9, %r8
0x7f04040136c9:  4d 03 86 00 9b 01 00  addq    0x19b00(%r14), %r8
0x7f04040136d0:  49 ff c1           incq    %r9
0x7f04040136d3:  4d 89 8e f8 9a 01 00  movq    %r9, 0x19af8(%r14)
0x7f04040136da:  49 89 30           movq    %rsi, 0(%r8)
0x7f04040136dd:  41 c6 40 08 12     movb    $0x12, 8(%r8)
0x7f04040136e2:  41 58              popq    %r8
0x7f04040136e4:  49 81 c1 80 7b e1 ff  addq    $-0x1e8480, %r9
0x7f04040136eb:  75 0f              jne     0x7f04040136fc
0x7f04040136ed:  57                 pushq   %rdi
0x7f04040136ee:  56                 pushq   %rsi
0x7f04040136ef:  50                 pushq   %rax
0x7f04040136f0:  49 8b fe           movq    %r14, %rdi
0x7f04040136f3:  ff 15 4f 00 00 00  callq   *0x4f(%rip)
0x7f04040136f9:  58                 popq    %rax
0x7f04040136fa:  5e                 popq    %rsi
0x7f04040136fb:  5f                 popq    %rdi
0x7f04040136fc:  41 59              popq    %r9
0x7f04040136fe:  8b 2e              movl    0(%rsi), %ebp

Epilogue: [...]
```

Results



Version 1:

Call a C helper from TB

Version 2:

Generate assembly code in TB to fill a shared buffer.

Version 3:

code in TB, to fill a buffer (lock free).

Benchmark:

- Modeling an ARM VExpress Board, with a single core cortex-a9
- Running dhrystone for ARM (5000000 runs)
- Simulation takes around 11 seconds without instrumentation
- 1.2 Billion guest memory accesses (800M reads, 400M writes)

The Cache model

Model gets addresses that are accessed (for read or write).

It can model anything.

Model follows real H/W,

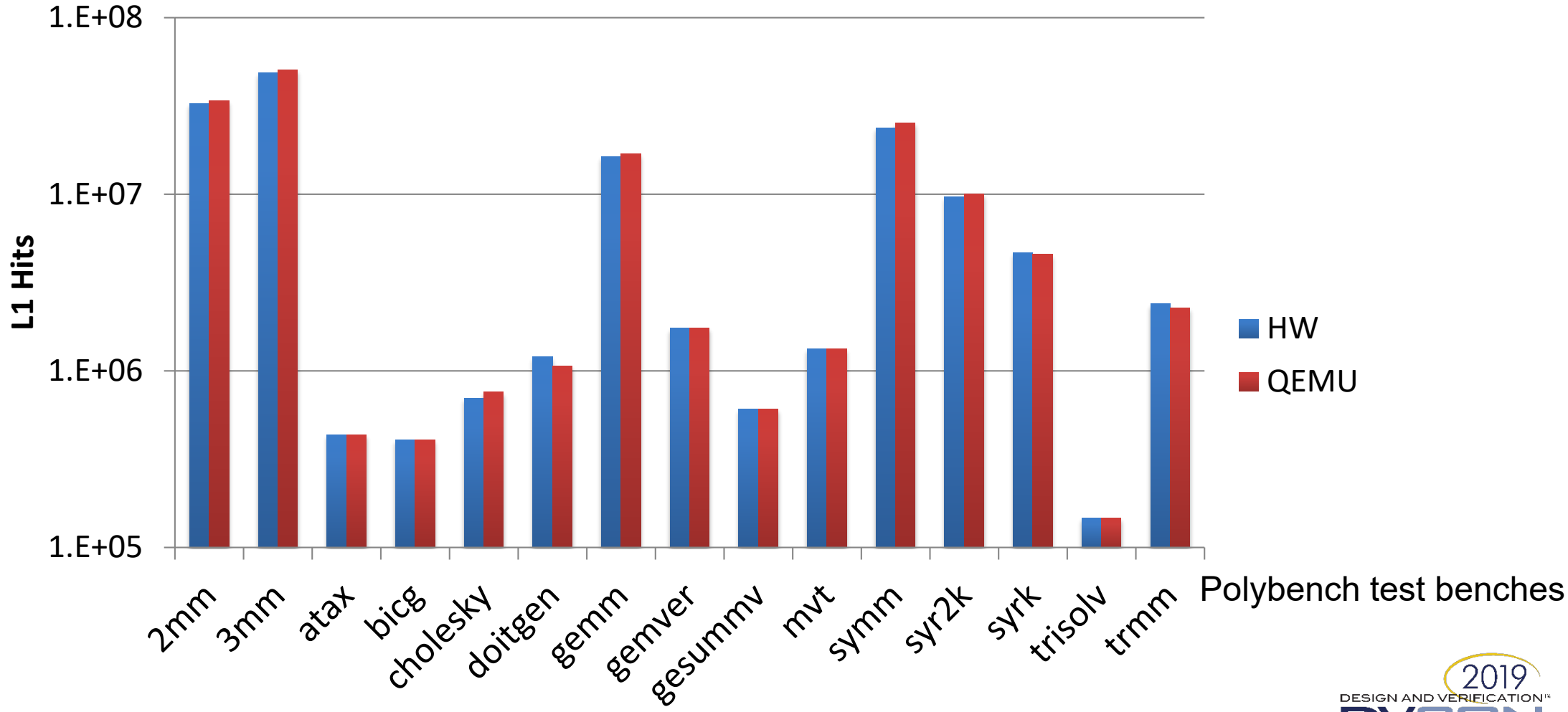
Complete with Linefill buffers, etc.

Time is **not** modeled

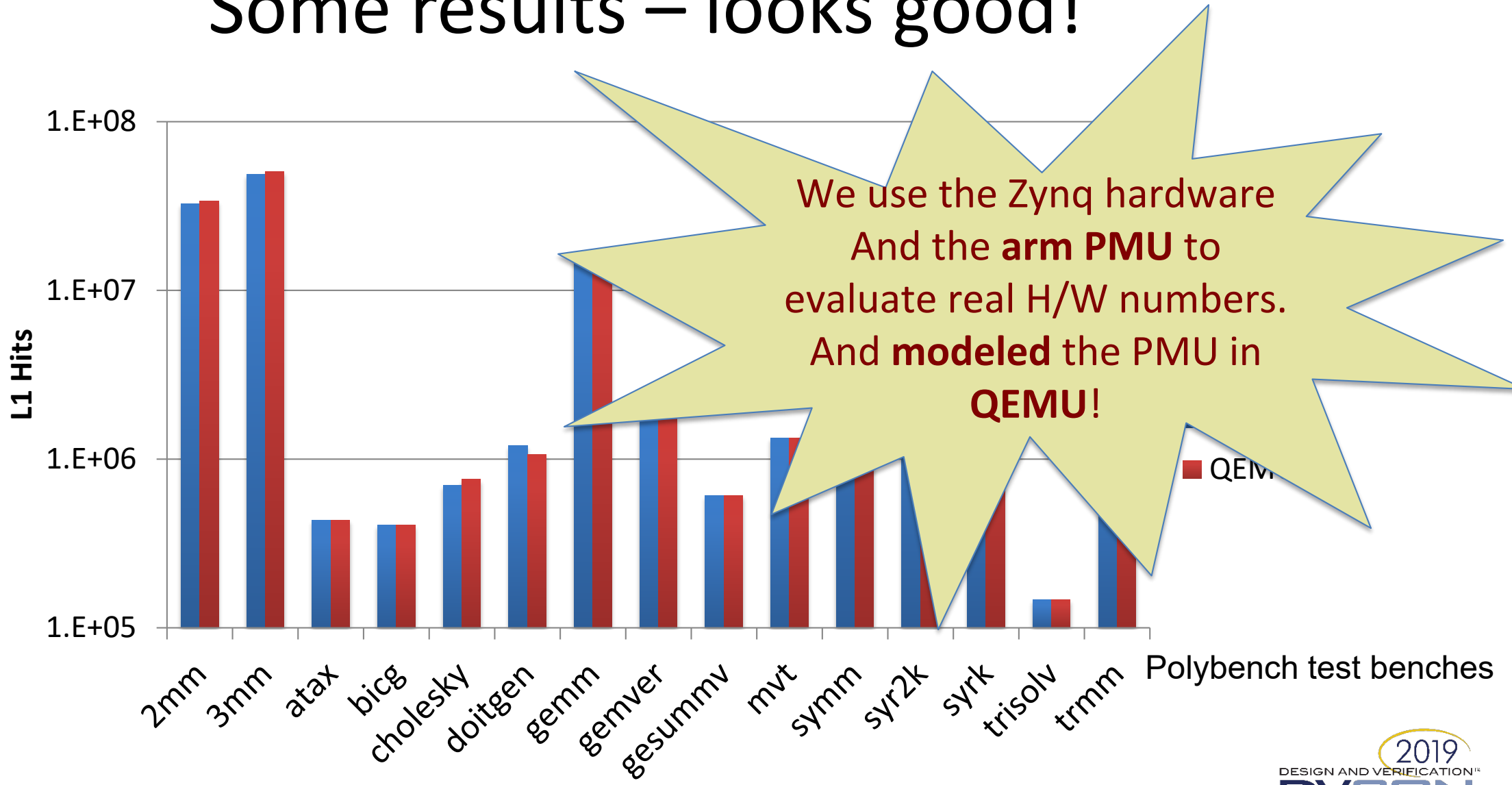
(e.g. sometime it's easier to count the statistics at different 'times' than the real h/w)

Model ONLY holds addresses, **not data**

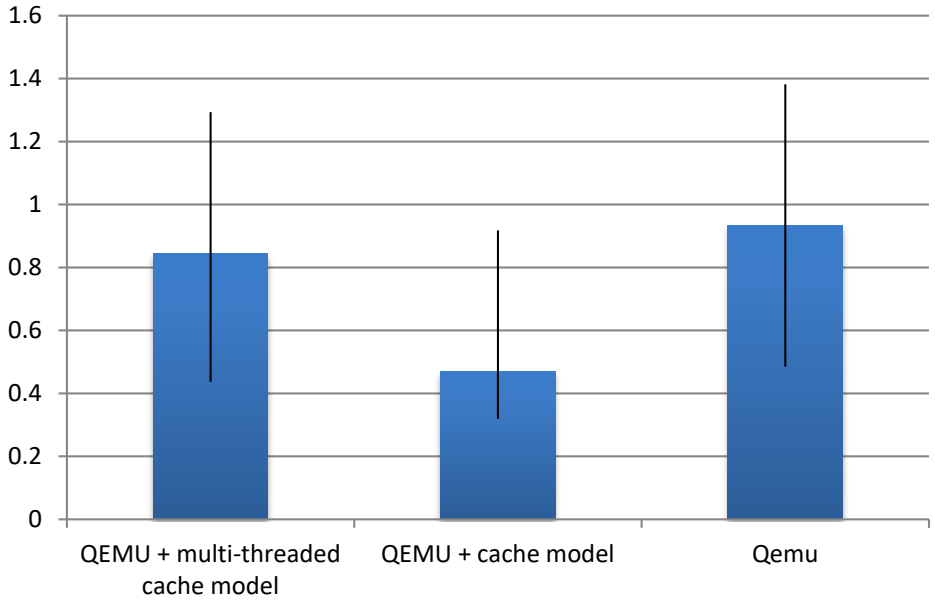
Some results – looks good!



Some results – looks good!



SPEED Results



Speed compared to HW

- QEMU runs at about real-time for a 600 mHz Zynq board.
- Adding a multi-threaded cache model adds ONLY about 10% overhead

This is for an R5 single core target, across all polybench benchmarks.

Questions

Mark @ GreenSocs.com