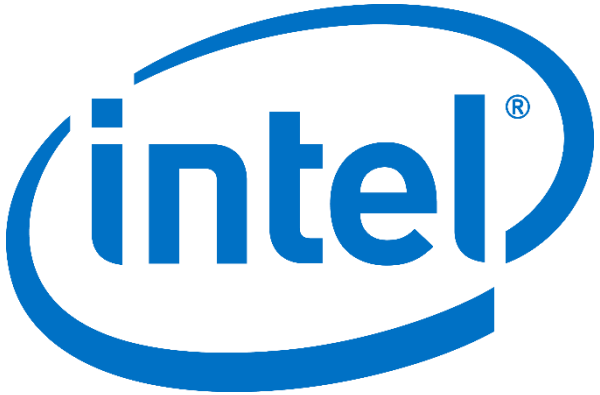
Temporal Decoupling – Are “Fast” and “Correct” Mutually Exclusive?

Dr. Jakob Engblom, Intel Stockholm, Sweden

jakob.engblom@intel.com

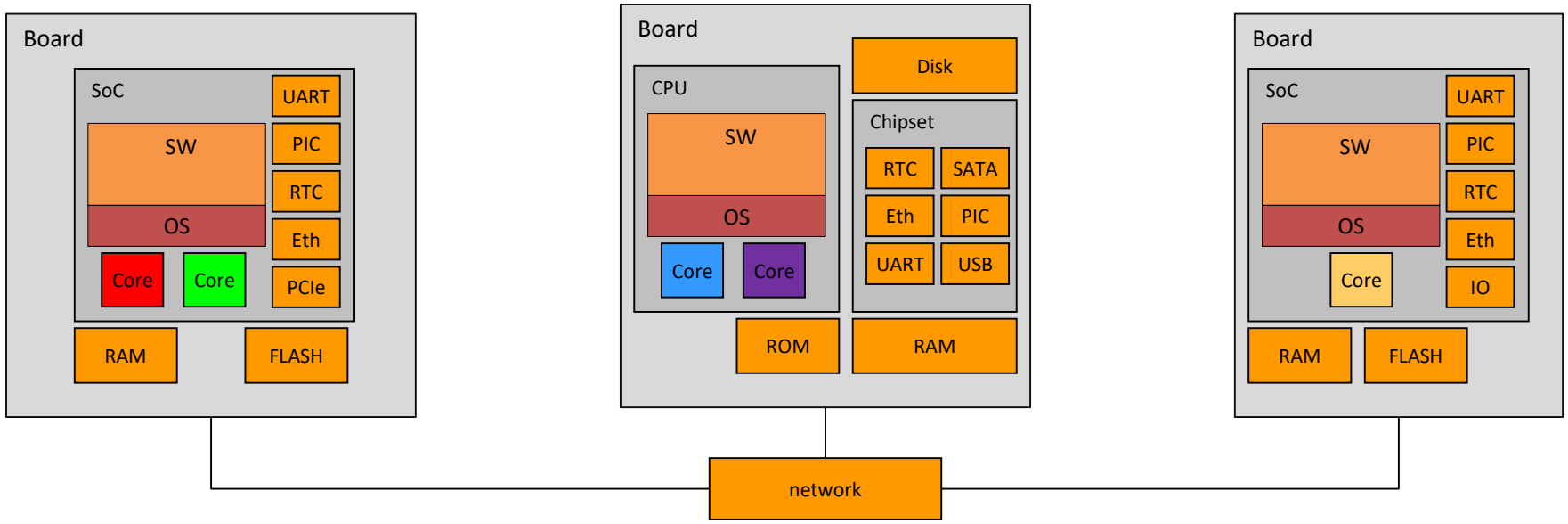


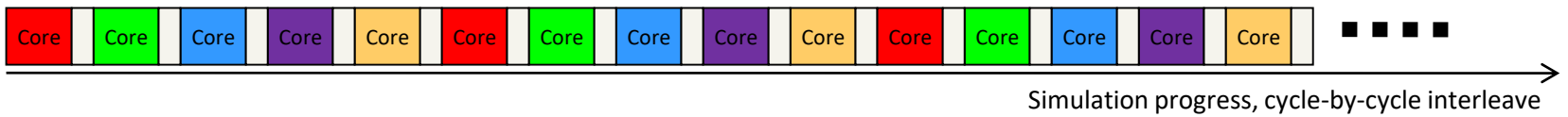
accellera SYSTEMS INITIATIVE

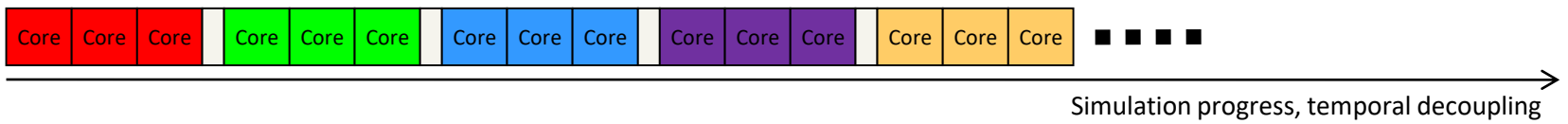
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

TEMPORAL DECOUPLING

Temporal Decoupling







Necessary Technique

Less overhead

- Less time spent in simulator kernel
- More time spent in models
- Fewer trips through sim kernel code

- A fast Virtual Platform (VP) has to get to much less than 10 host instructions per target instruction

More locality

- Data
- Code

- Better effect from Just-in-Time (JIT) compilers and Virtualization Technology (VT) acceleration

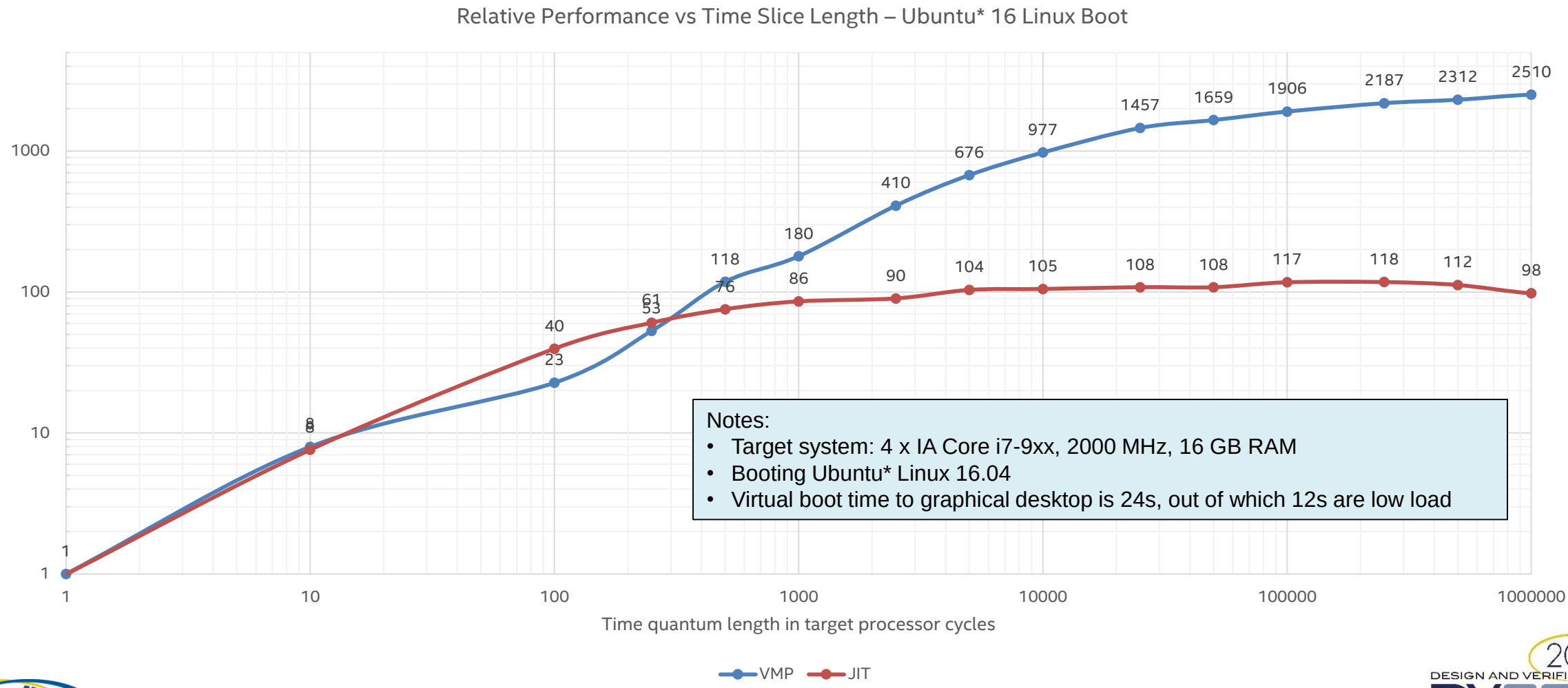
Experiments on Wind River Simics®

- All experiments performed using Simics

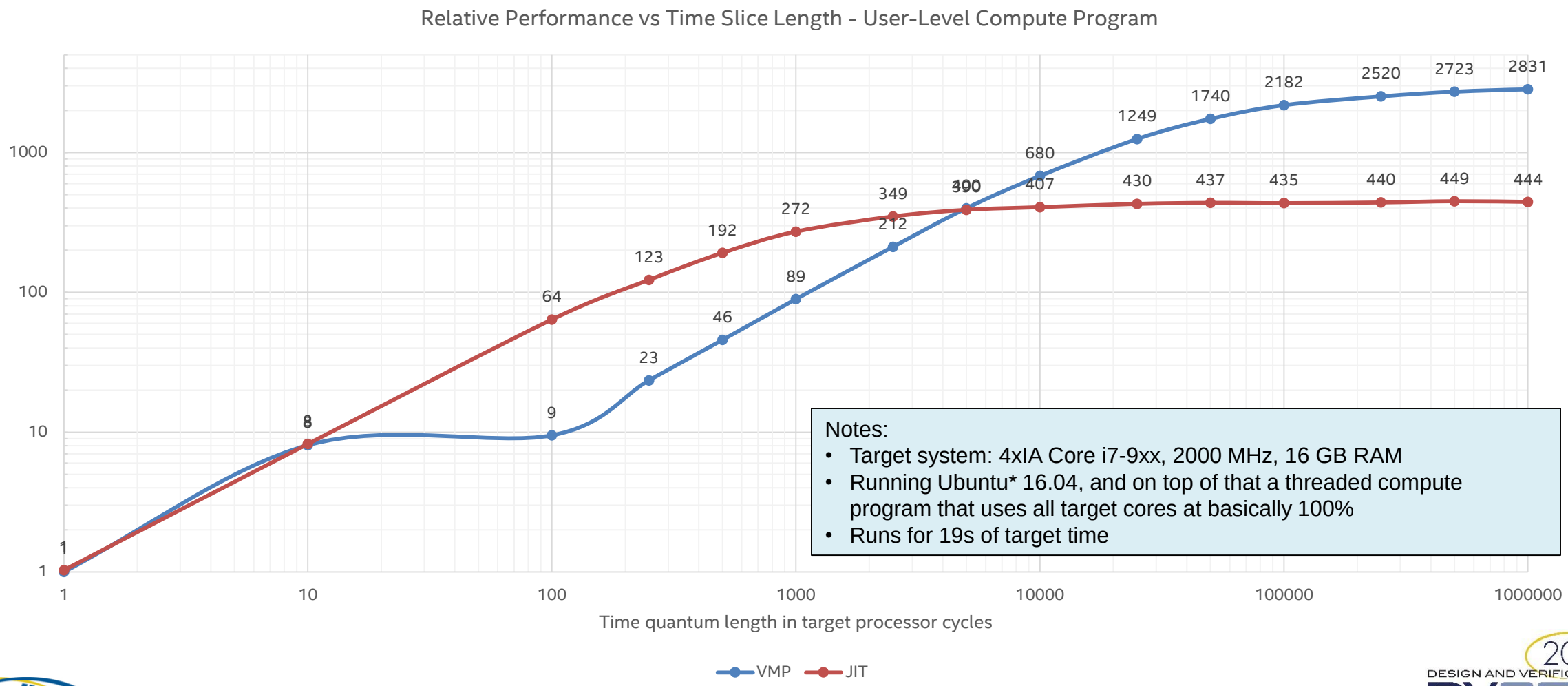
- Instruction Set Simulator (ISS) modes:
  - Interpreter
  - Just-in-time (JIT) compiler
    - Like all other fast simulators in the world
  - Direct execution (VMP)
    - Uses Intel® Virtualization Technology for Intel® 64 and IA-32 architectures (Intel® VT-x) to run IA code directly on the host

FAST?

Example: Boot



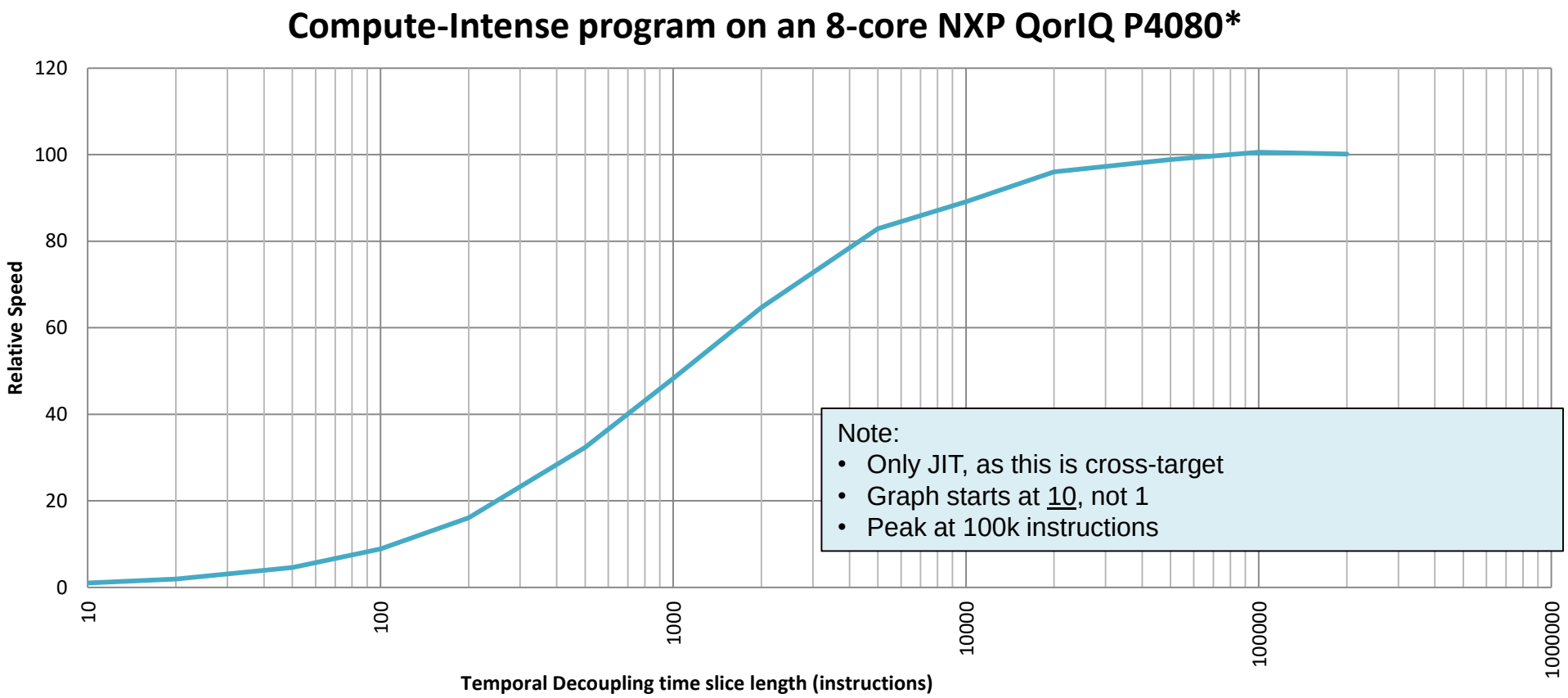
Example: Compute Program



Notes:

- Target system: 4xIA Core i7-9xx, 2000 MHz, 16 GB RAM
- Running Ubuntu* 16.04, and on top of that a threaded compute program that uses all target cores at basically 100%
- Runs for 19s of target time

Example/Old: Compute Program (2008)



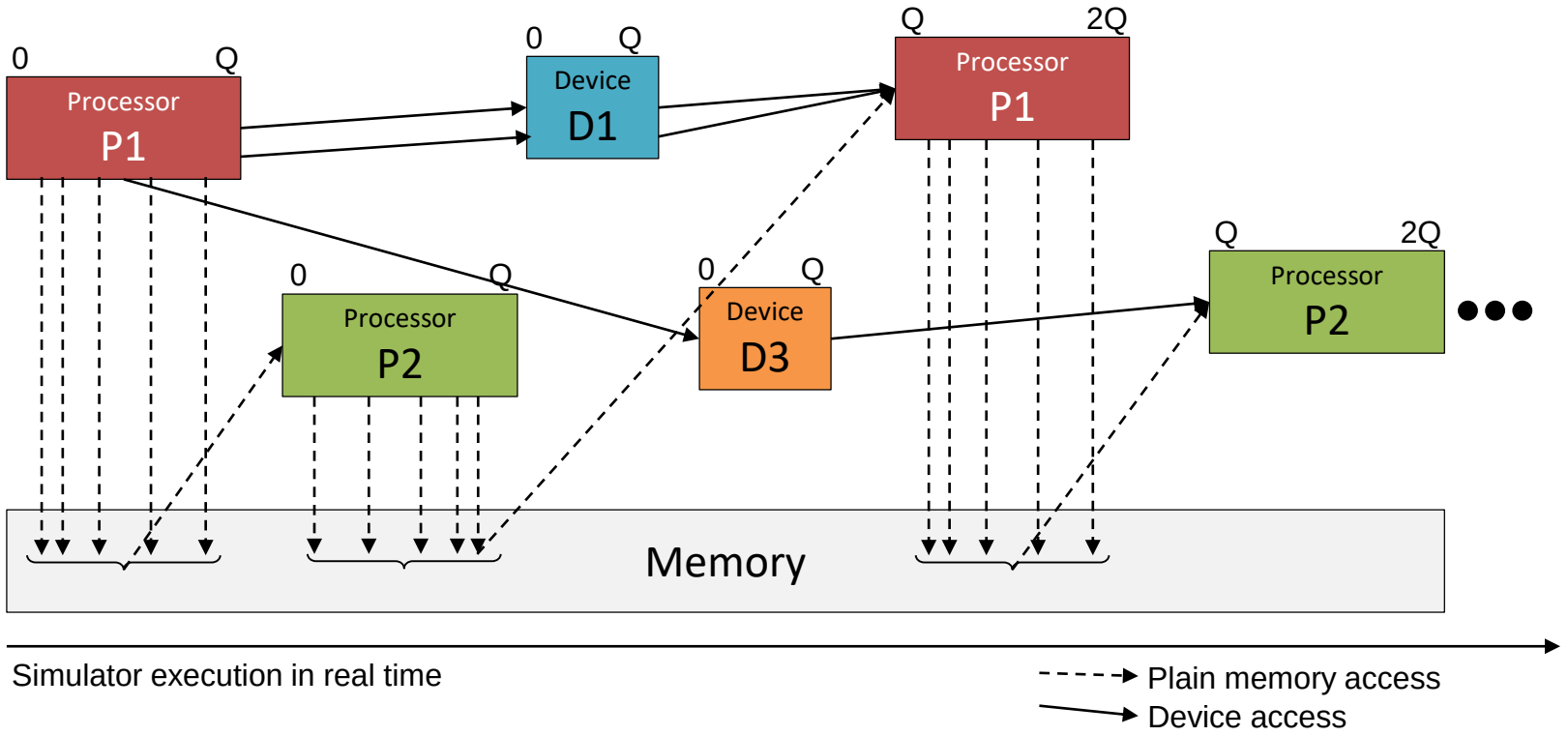
Observations

- VMP benefits from longer time slices – up to 1M instructions
  - Still less than 1/1000 of a second for a typical 2-4GHz processor core
- JIT plateaus around 10k instructions
  - Standard observation going back to mid-2000s
- Good default is 100k instructions, might increase from there

CORRECT?

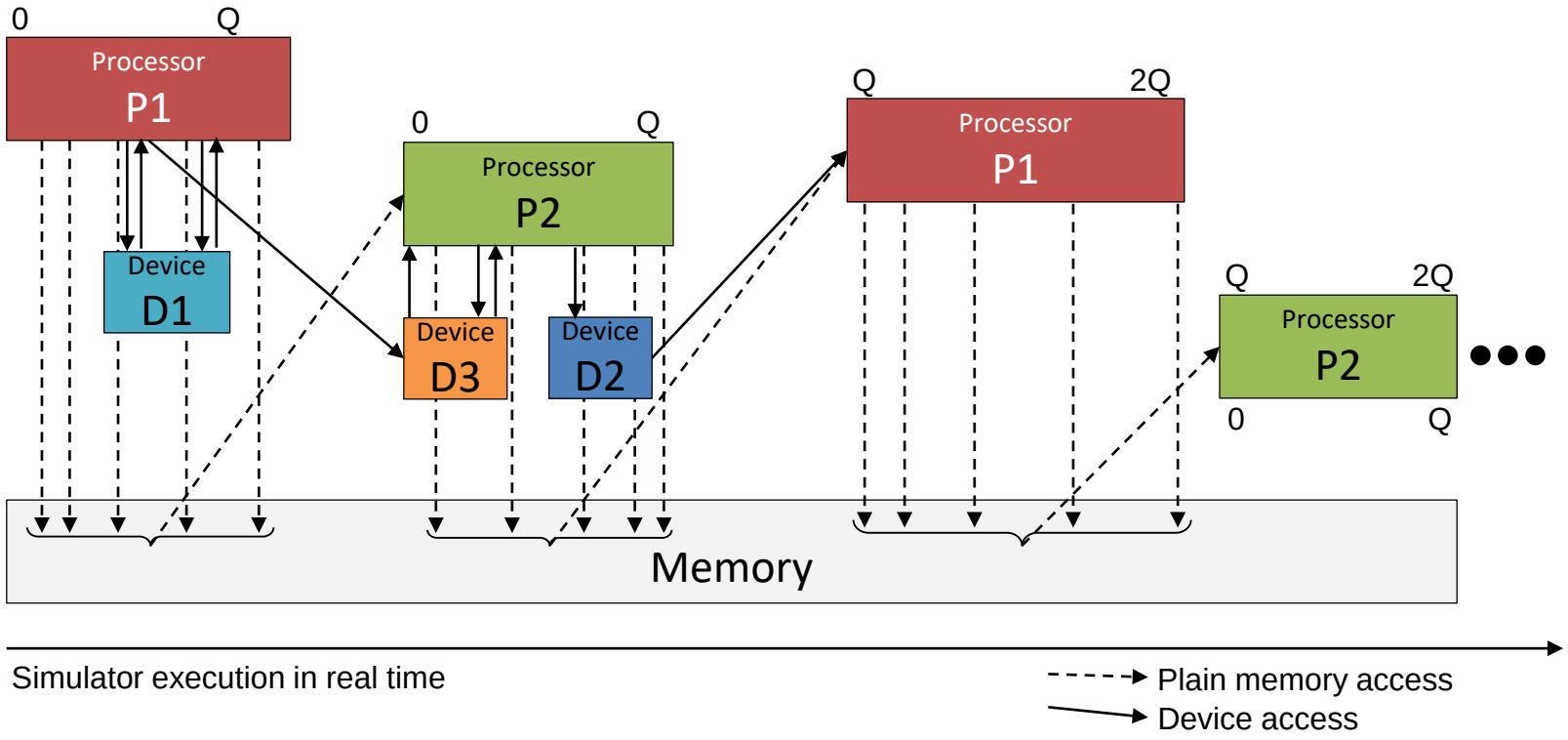
Temporal Decoupling = Information Latency

"Variant 1" in the paper, with devices running in their own time quanta

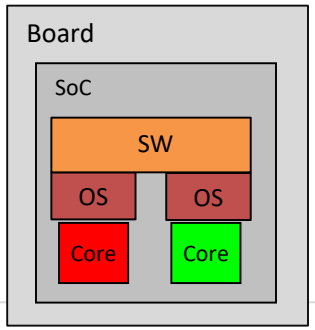


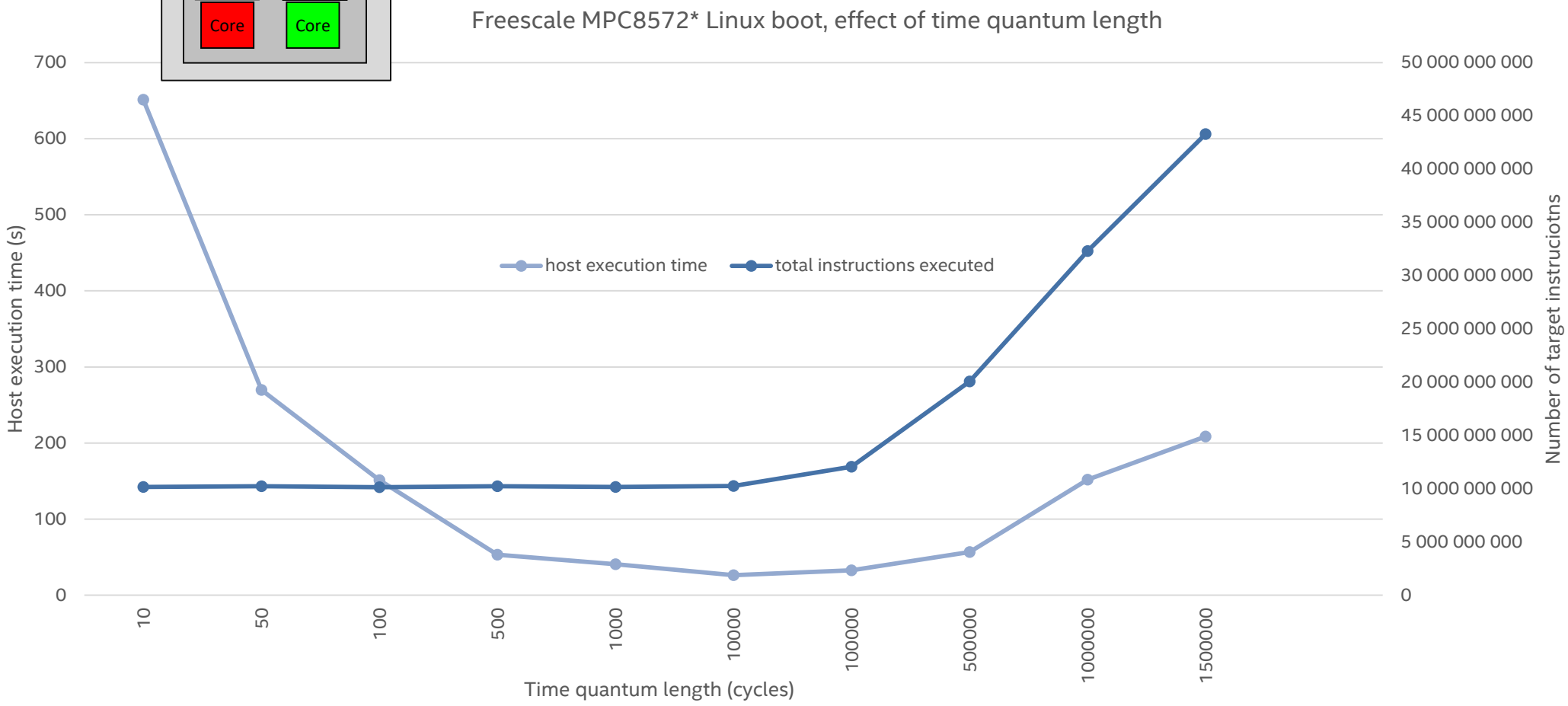
Temporal Decoupling = Information Latency

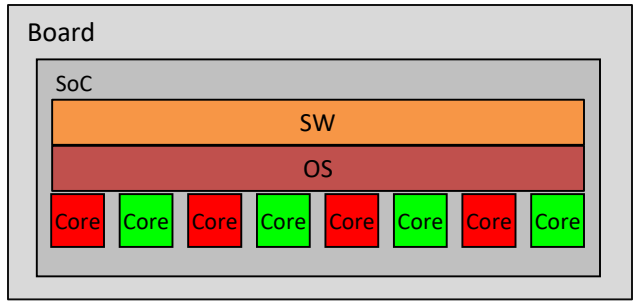
"Variant 2" in the paper, with devices running inside the time quanta of the processors accessing them



The Classic Bathtub

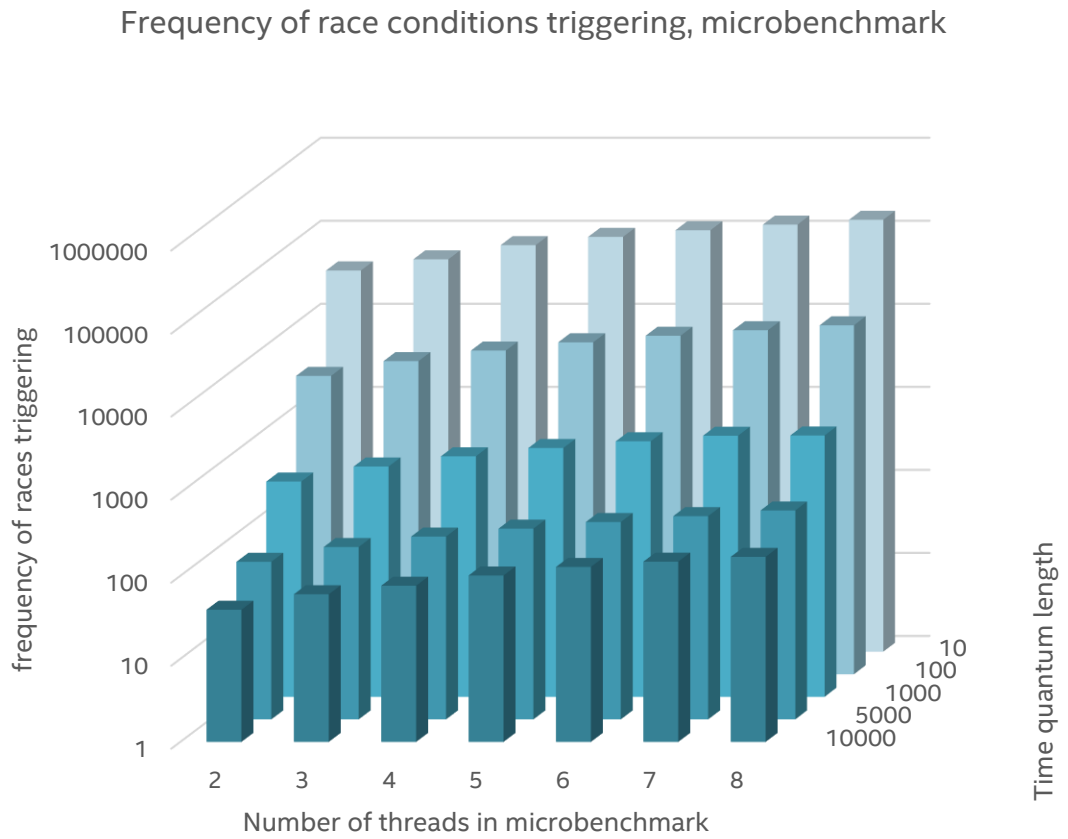




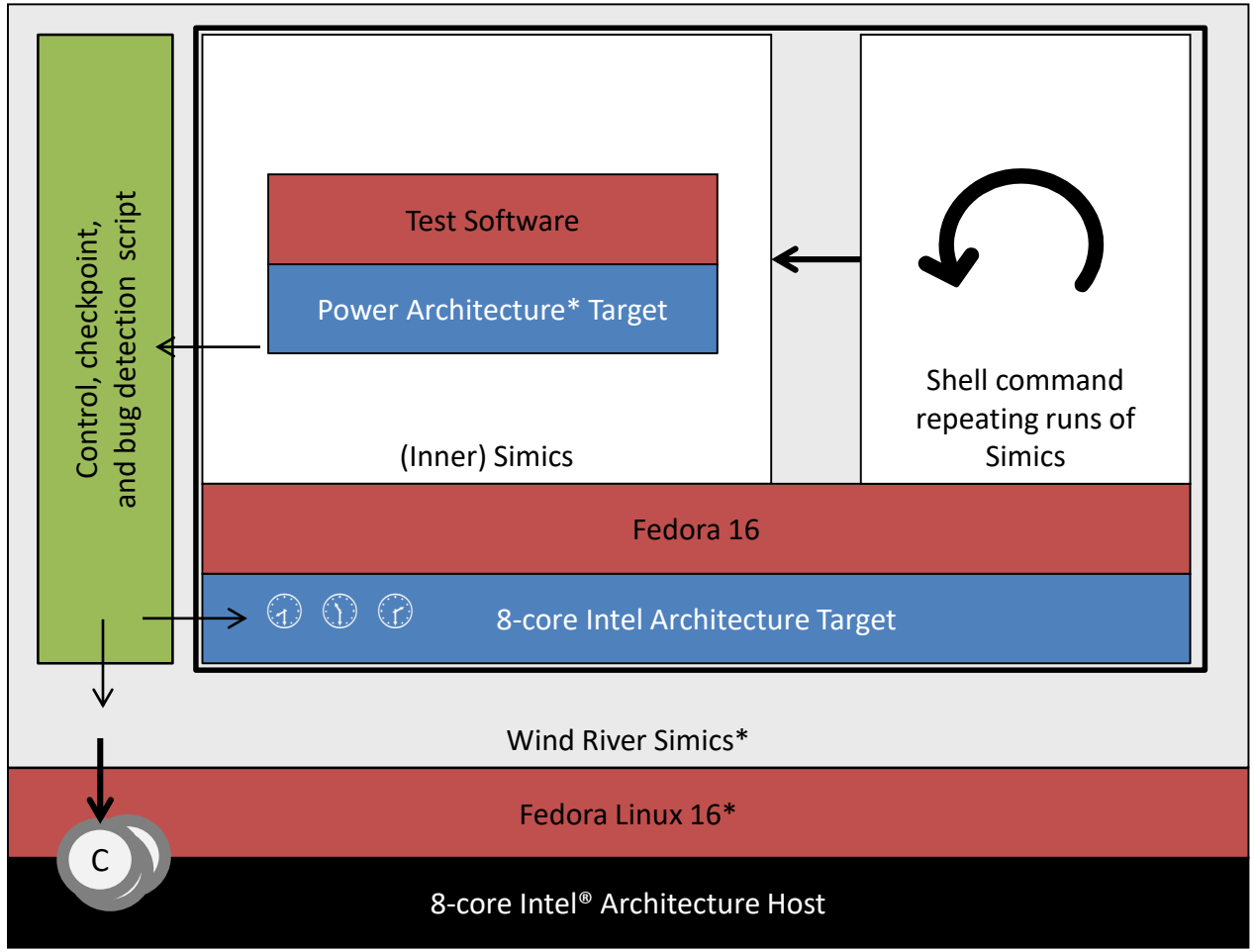


Race Conditions - Micro

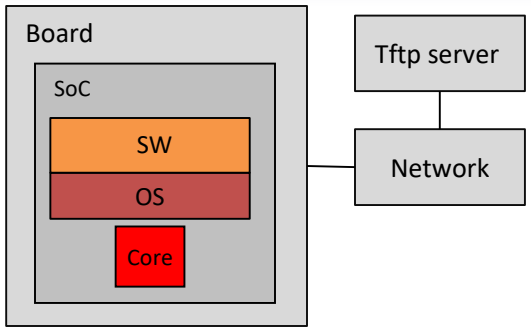
- Microbenchmark
  - Each thread loops, load-modify-write as quick as possible on a shared variable
- Longer time quantum =
  - Fewer races seen
  - But: Races still happen
- Not really representative of real software



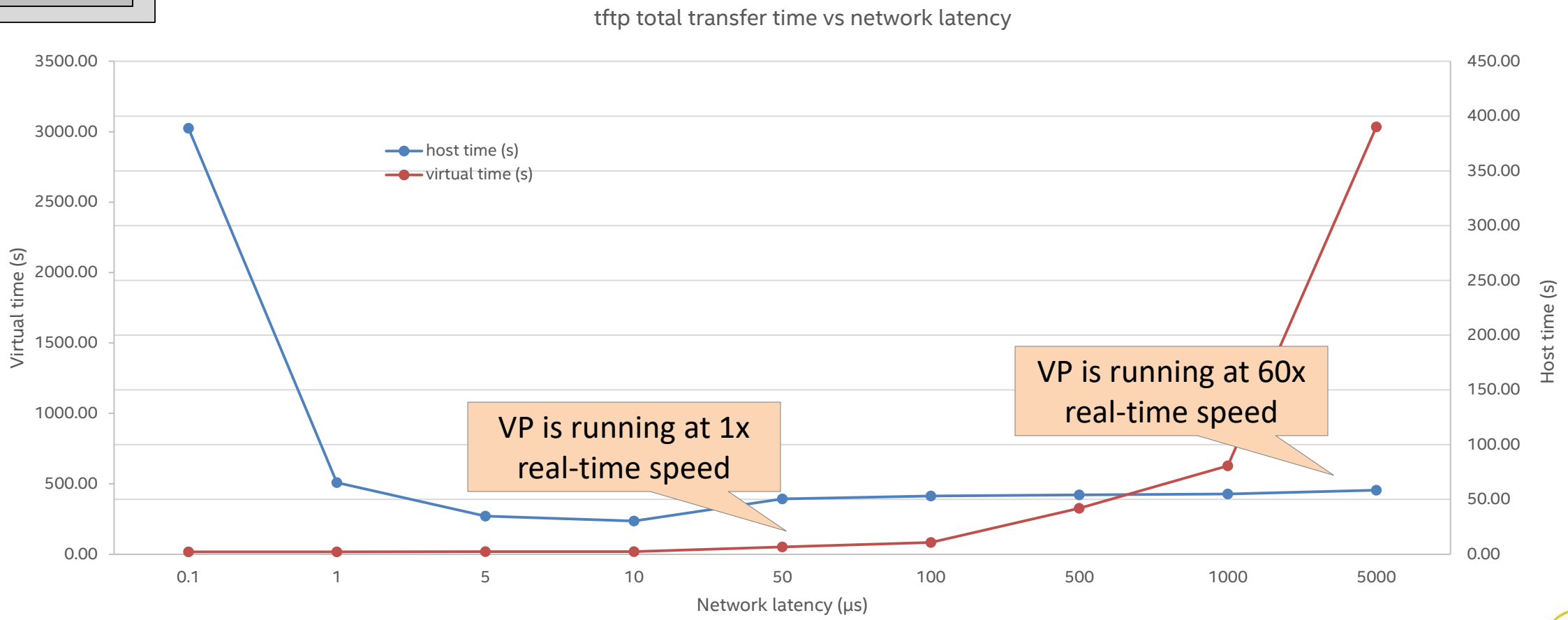
Race Conditions - Macro



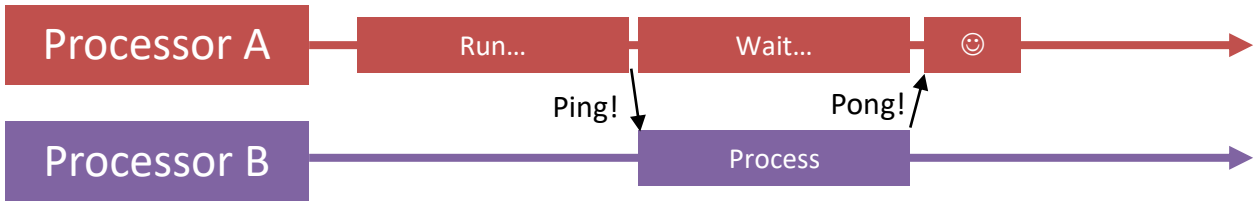
- Varying time quantum length proven way to find errors
  - Order of events
  - Interleave of software operations
- Scriptable & deterministic
  - Example: http://blogs.windriver.com/tools/2012/12/debugging-simics-on-simics.html
  - Some 30 tests to replicate error



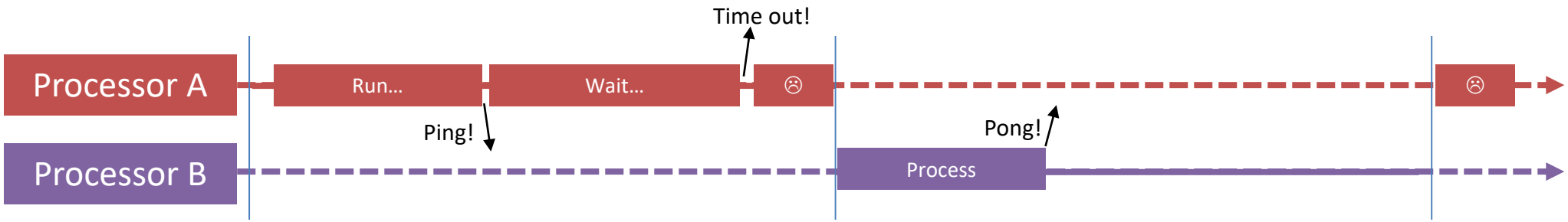
Ping-Pong Protocol, no Time-Out



Ping-Pong Protocols with Time-Out



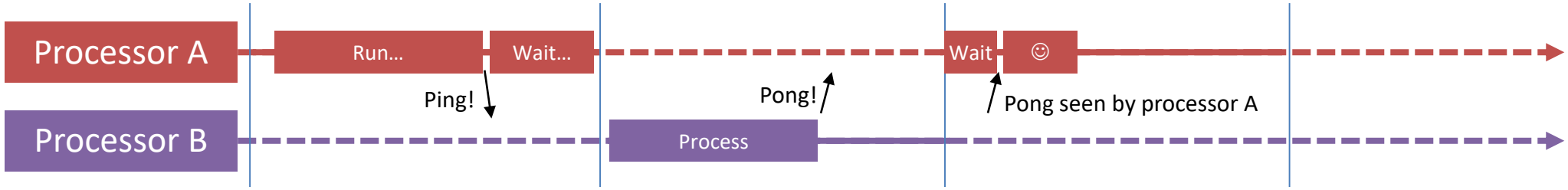
On hardware, this is expected path



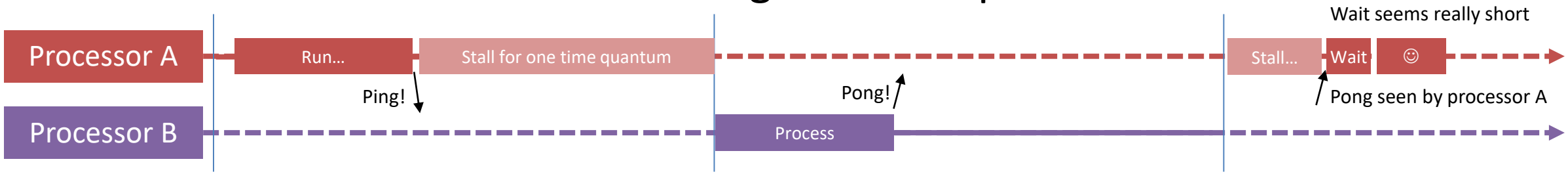
If the time slice is longer than the time-out

Ping-Pong Protocols with Time-Out (2)

- Reduce the time quantum…

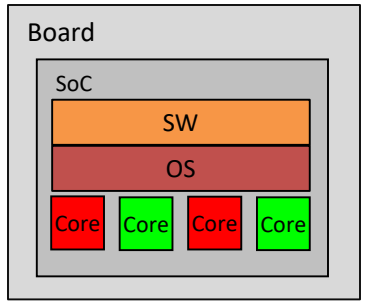


- … or insert stalls to avoid reducing the time quantum

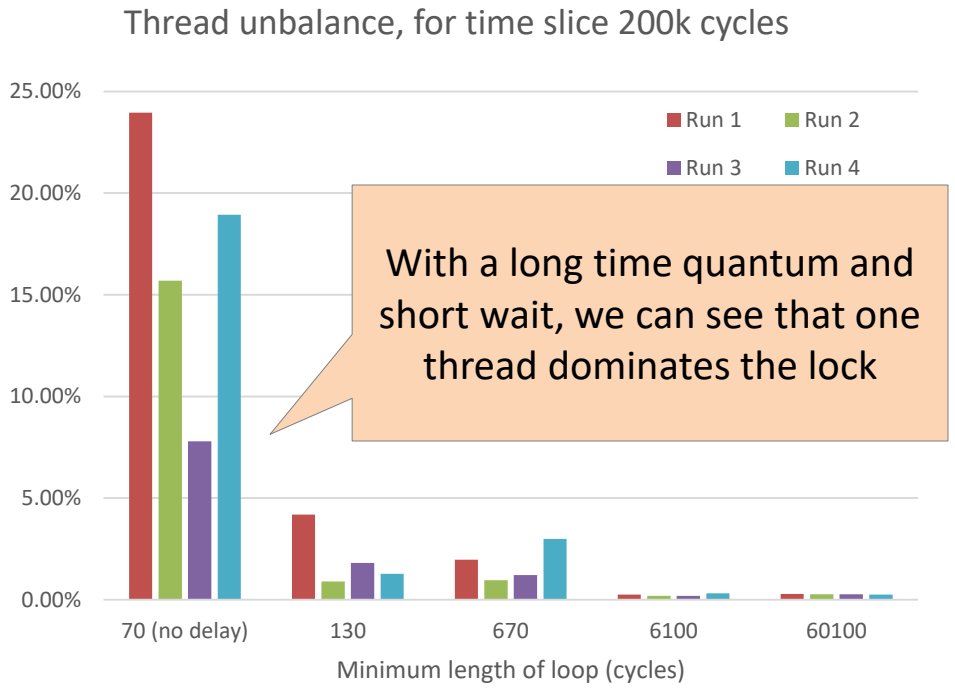


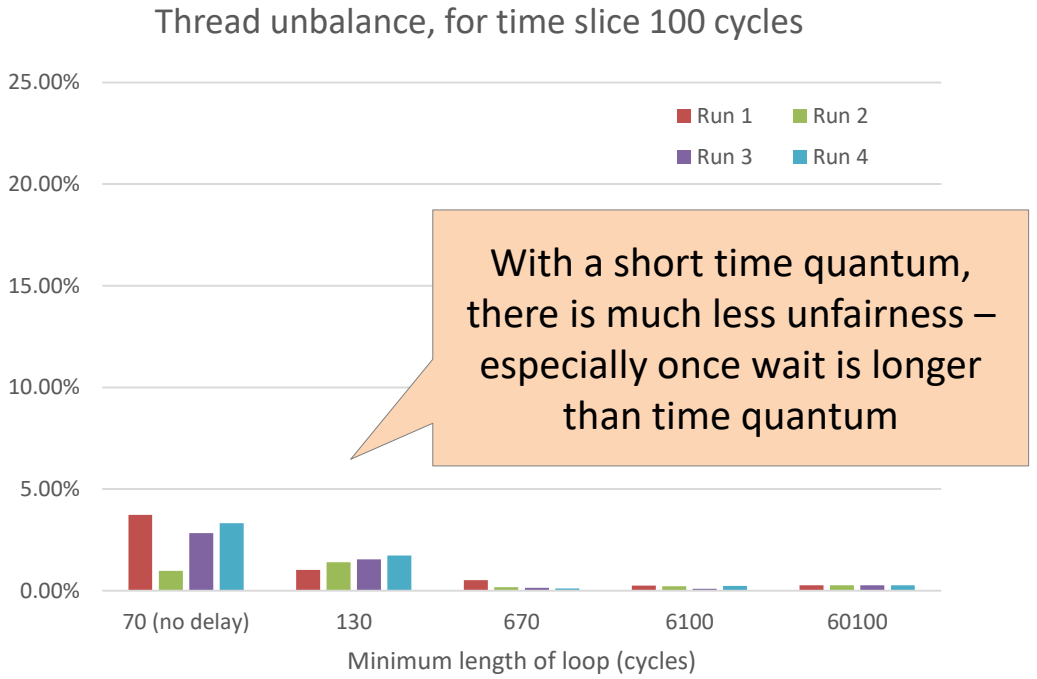
Make the ping operation take a long time so that processor B sees it and replies before the check for the reply & the time-out starts

Example: Fairness Test




```
4 x threads: Loop:
     Take lock
     Inc counter
     Release lock
     Wait N
```

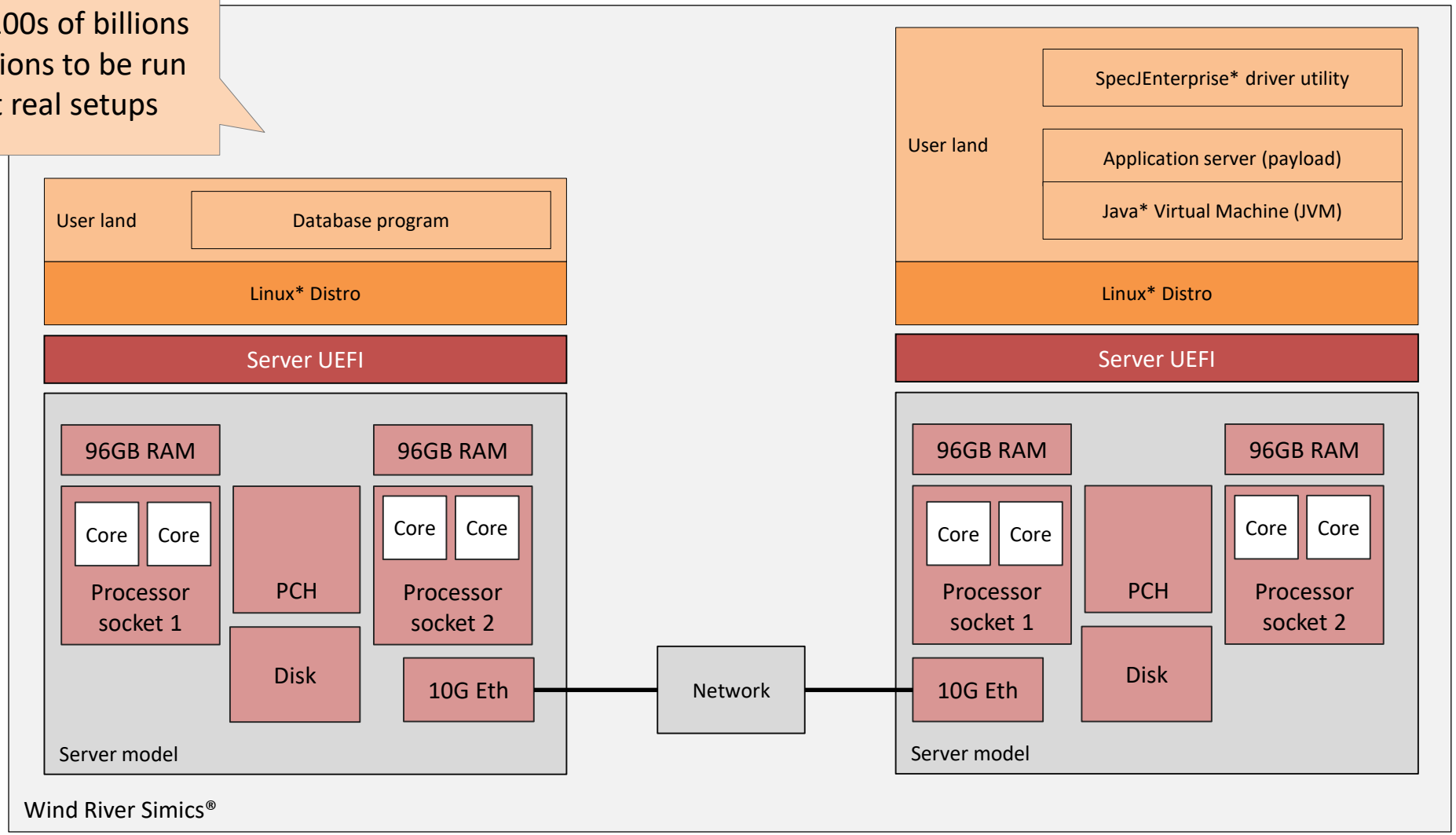




SUMMARY

Most of the Time, Speed is Key

We need 100s of billions of instructions to be run for most real setups



When do we Need to Dial Down?

- Results (timing) skew when accessing shared resources:
  - *Time quantum < shortest relevant observable delay*
  - Example: Cache studies: time slice lower than last-level cache penalty is ”OK”

- Software time-out:
  - Unit reports time-out unless a reply is seen within a short time
  - … unless we can solve it some other way (stall)
- Software expecting close timing between closely-coupled cores

Questions?