Intel® Simics® Virtual Platforms in Industry

Dr. Jakob Engblom, Director, Simulation Technology Ecosystem

Intel, Stockholm, Sweden

jakob.engblom@intel.com

intel®

About Me: Jakob Engblom

Currently:

- Director (of Simulation Technology Ecosystem), Simics Core team, at Intel in Stockholm, Sweden

Education:

- MSc, Computer Science, and PhD, Real-Time Systems, **Uppsala**

Experience: virtual platforms, simulation, embedded systems

- Product management, product marketing, technical sales, technical marketing, business development, training development, demos, … At IAR Systems, Virtutech, Wind River, and Intel

My own blog, since 2007:

- https://jakob.engbloms.se

Intel software blog:

- https://community.intel.com/t5/Blogs/Products-and-Solutions/Software/bg-p/blog-software

Introducing Intel®

intel advantage

Software + Silicon & Platforms + Packaging & Process

At-Scale Manufacturing

The leading provider of silicon globally

Product Leadership

Client Computing	Datacenter & AI	Network & Edge	Accelerated Computing Systems & Graphics	Automotive	Foundry Services
Make PCs indispensable for work, play, learning, creating and collaborating with a vibrant ecosystem of innovation	Develop the best cloud and data center solutions for diverse customer workloads with leadership in AI	Provide cloud to intelligent edge network infrastructure by delivering 5/6G, Edge-AI, NIC/ Switch, SiPh and Software disruptions at scale	Become the No. 1 provider of accelerated compute through graphics, GPU and HPC – first to Zettascale	Be the leading provider of autonomous vehicle technology	Become the No. 2 foundry provider this decade

Open Platforms

700+
Foundations and standards bodies with Intel

450+ Software tools & resources
on the Intel Developer Catalog for developers to create and deploy solutions

150+ Software tools & resources
on the Intel Developers Catalog for AI workloads

300+
community managed projects contributed to and maintained

#1
Linux kernel corporate contributor since 2007

top 10
contributor to Kubernetes

Intel software powers much of the world's computing

Floating Point Standard

HI-SPEED CERTIFIED USB

Wi Fi

DDRx

PCI EXPRESS

THUNDERBOLT

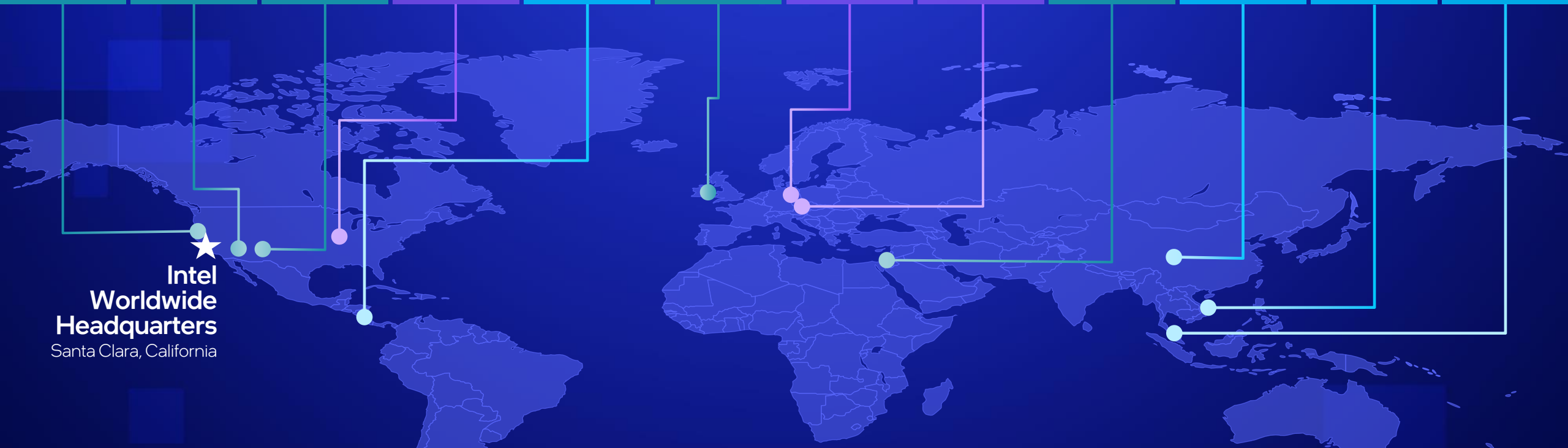
5G Standards

RSS Responsibility Sensitive Safety

CXL Compute Express Link

6G Network/Edge

Oregon	Arizona	New Mexico	Ohio	Costa Rica	Ireland	Germany	Poland	Israel	Chengdu	Vietnam	Malaysia



Geographically Diverse Manufacturing Capacity

Wafer Fabs · Assembly & Test · Future Site

The Intel® Simics® Simulator

The History of the Intel® Simics® Simulator

Development started in 1991

- Spin-off from research project
- Pre-silicon OS bring-up.

Virtutech founded in 1998

- Sun* & Ericsson* first customers

Acquired by Intel in 2010

Wide usage

- Intel-internal
- Intel ecosystem
- Commercial customers via Wind River*
- Academics and OSS via public release

Major milestones

- 2.0: Heterogeneous system models
- 3.0: Reverse execution & debug, 2005
- 3.2: Intel virtualization acceleration
- 4.0: Multi-threaded (coarse), 2008
- 4.2: Distributed simulation, 2009
- 5: Multicore multithreading, 2015
- 6: More threading & better support for model integrations, 2018
  - Integration with power, thermal, performance models
  - Continuously adding features
- 7: Clean up & modernizations, 2023
  - Removing older APIs and features to focus on the new

What Does Intel® Do and Where do We Fit?

Laptop and desktop
- Intel® Core®
- Intel® Atom™
- Chipsets
- Thunderbolt*
- Graphics Processors (GPU)

Data Center
- Intel® Xeon®
- Chipsets
- Infrastructure processing units (smart network)
- GPU

AI and ML
- Movidius
- Habana
- Intel® Xeon®
- GPU
- AI PC

Connectivity
- Ethernet
- WiFi
- Bluetooth
- GNSS

FPGA (Now Altera®)
- FPGA
- SoC-FPGA
- eASIC hard-copy

Intel Labs
- Quantum computing
- Neuromorphic computing
- Software

Foundry
- Intel Foundry Services

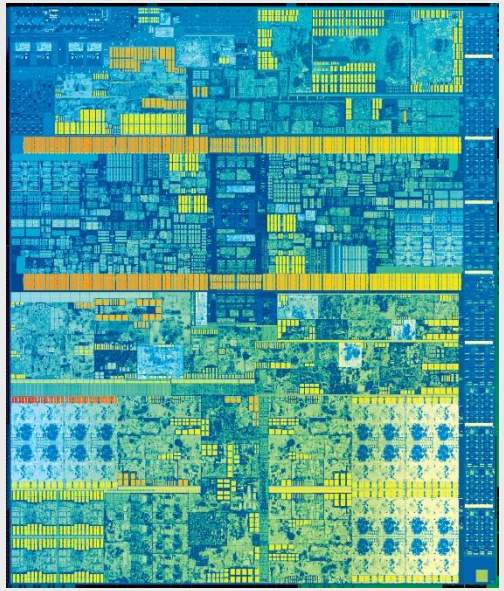
Software
- OneAPI
- Development tools
- Compilers
- Simulation solutions
- Linux & Windows drivers
- UEFI & BIOS

Virtual Platforms Why and What?

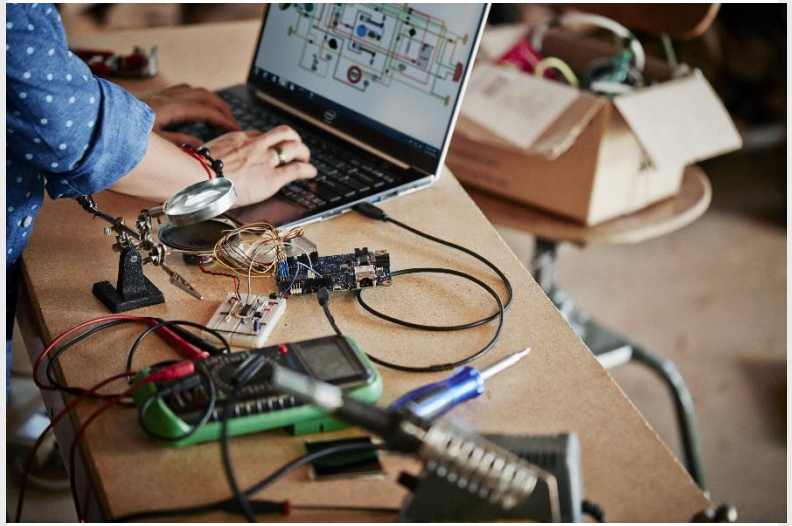
Hardware: A Hard Development Platform?

Hardware is Hard When it is in…

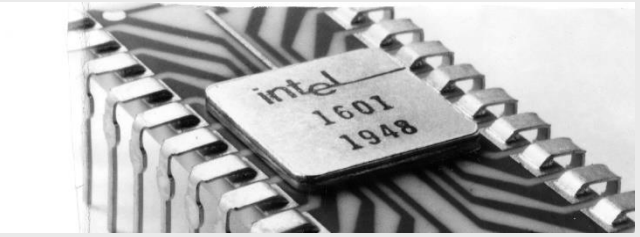
Not yet available



Flaky prototype stage

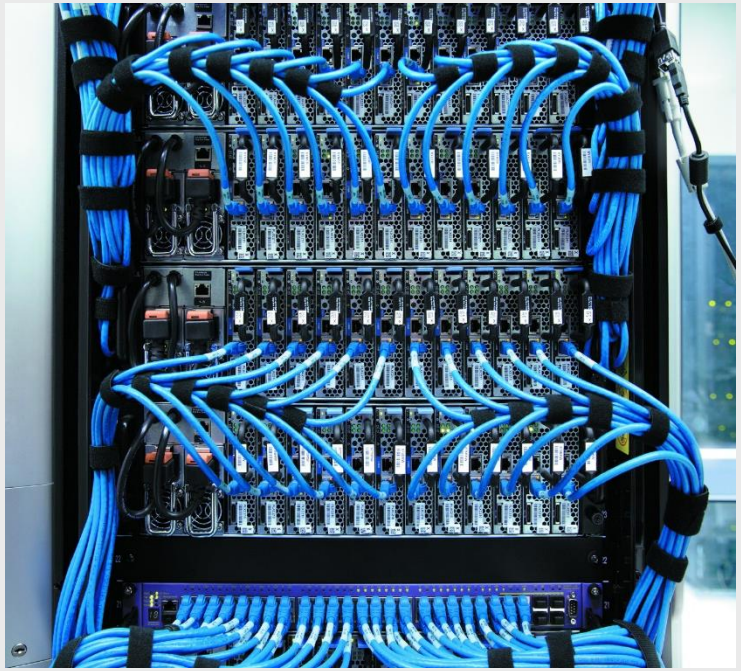


Not available anymore



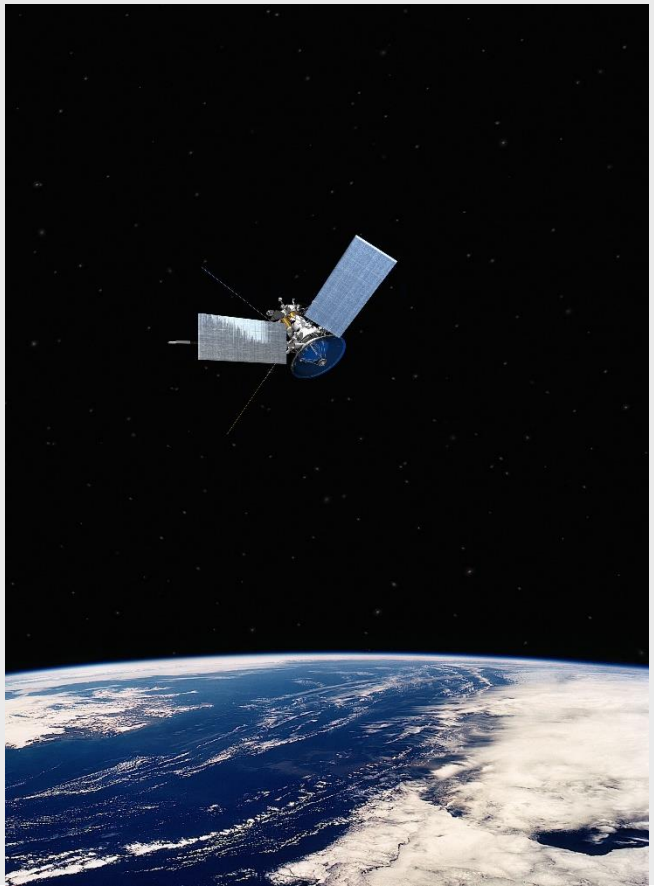
Hardware is Hard When it is…

Inconveniently large & complex

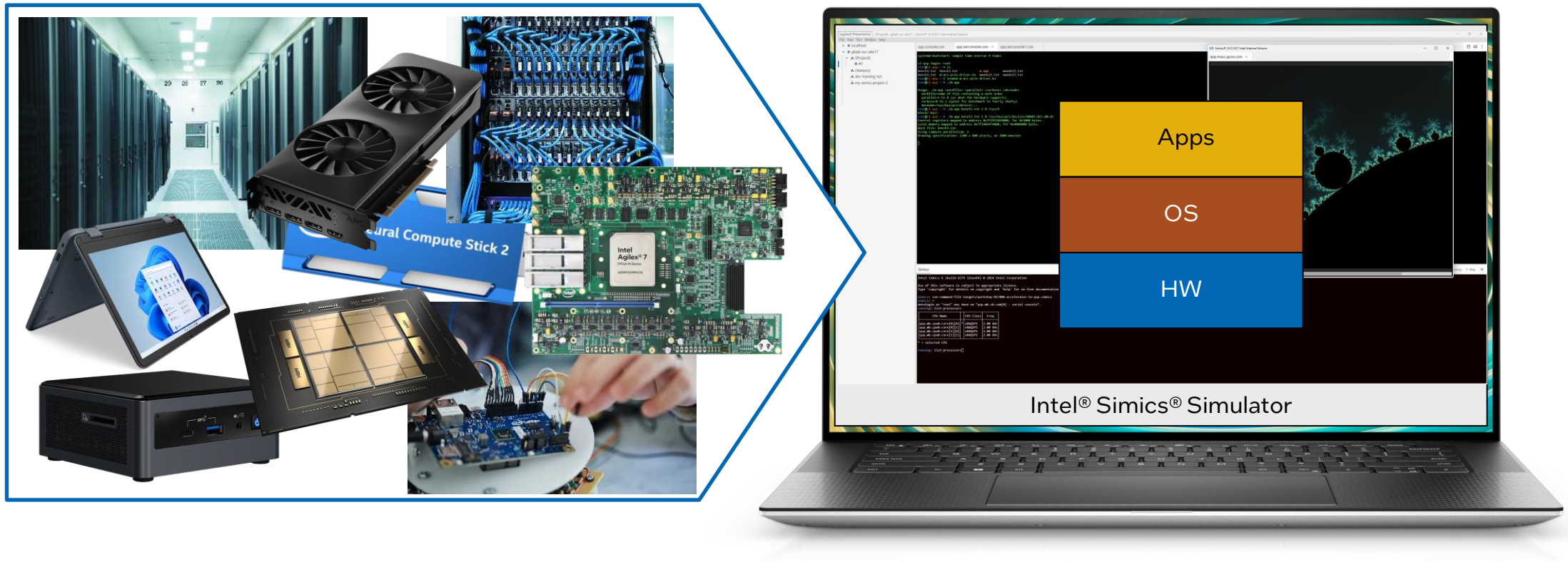


Dangerous to play with

Inaccessible & expensive



The Idea of a Virtual Platform



Run your software without the hardware – on a software model

Running the Real Software

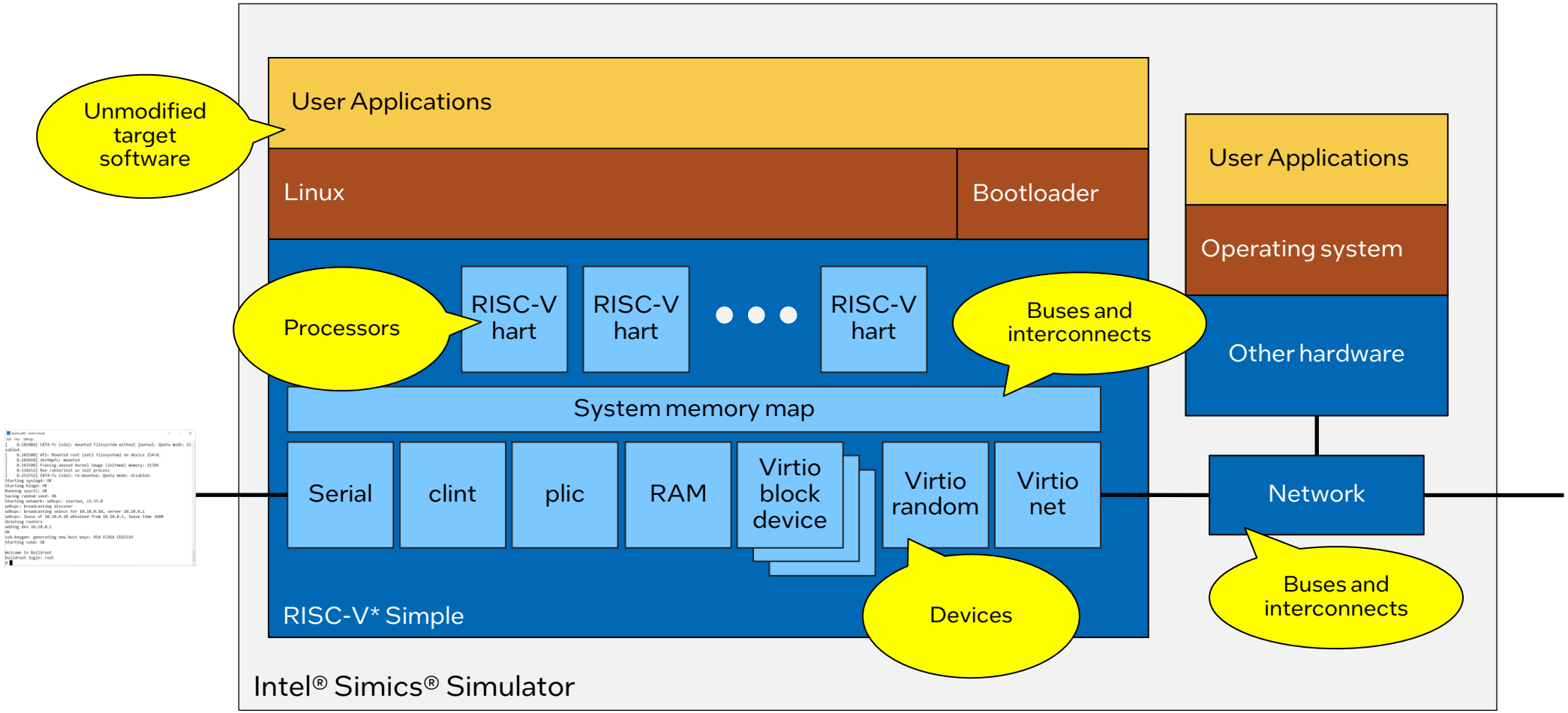
Purpose:

- Test & debug the **software** and the **software-visible** aspects of the hardware

"Software" can mean many things…

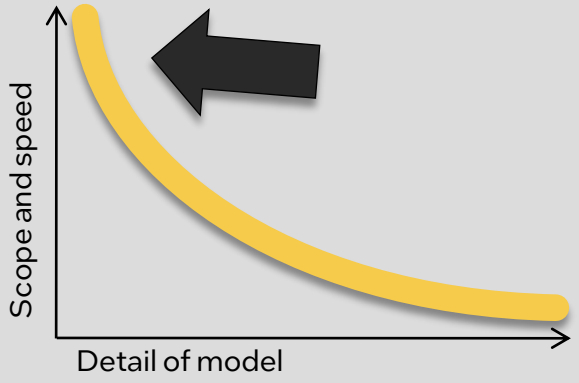
- **Firmware,** that is deeply hidden inside a chip
- **BIOS/Bootloader/UEFI,** that is used to boot the machine
- **Device drivers,** that manage hardware for an operating system
- **Operating systems**
- **Middleware**, providing services for other software
- **Applications,** that any programmer would write
- **Distributed systems**, software running across many separate machines
- From bytes to terabytes of code!

Inside a Typical Virtual Platform

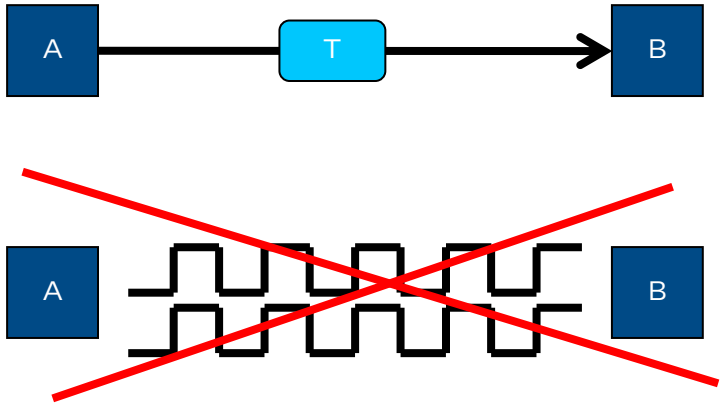


Intel® Simics® Simulation: Level of Abstraction

Goal: Fast & scalable simulation

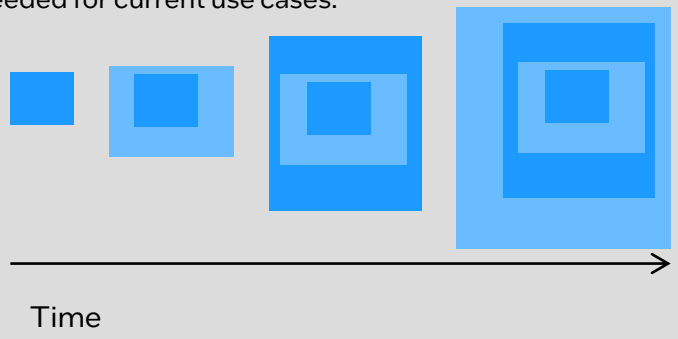


Transaction-level modeling (TLM)

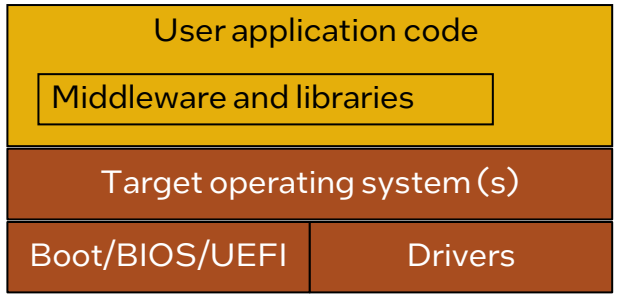


Lazy and agile modeling

Build up the model piece by piece over time, as use cases materialize or become possible. Only model what is needed for current use cases.



Goal: run the real software



Target model includes all software-visible functional aspects of hardware, such as processor instructions, supervisor modes, device registers, interrupts, etc.

Model function & basic timing

- Processor instruction set
- System memory map (not bus system)
- Device register interface
- Loose timing model
- Packet-level models of networks
- Event-driven simulation, not cycle-driven

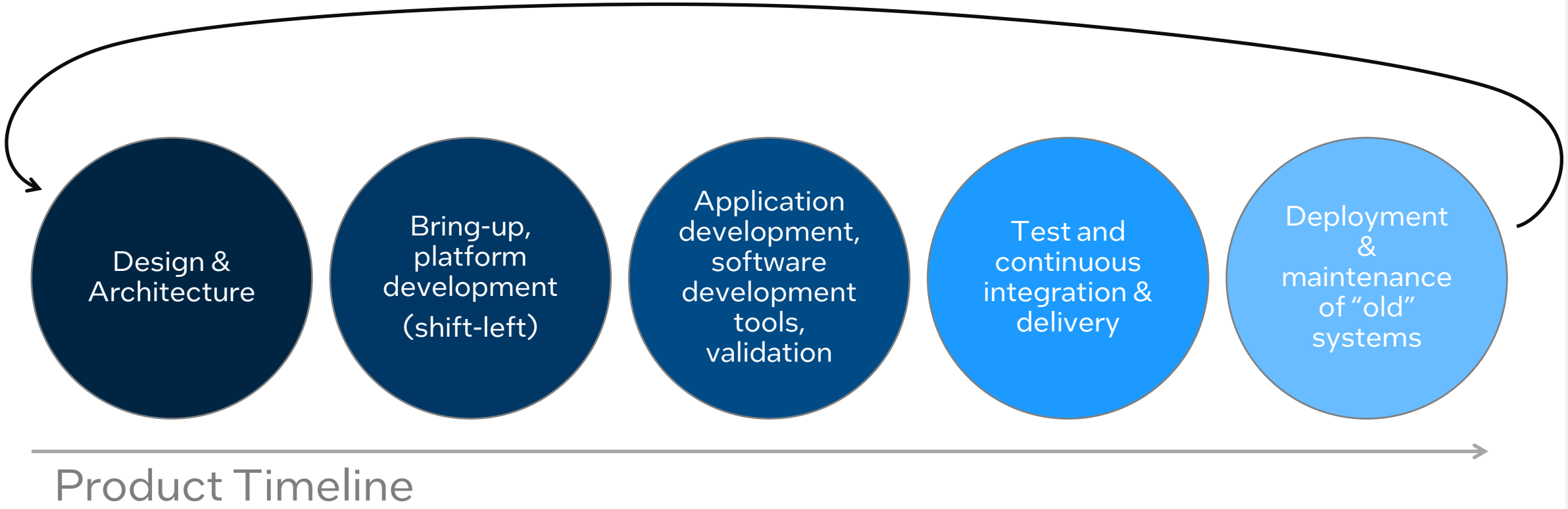
Add details when and where needed

- Processor performance models
- Device performance models
- Cache and memory system models
- Hardware RTL
- Power and thermal models

The Intel® Simics® Simulator

Use Cases

Virtual Platforms & the Product Lifecycle

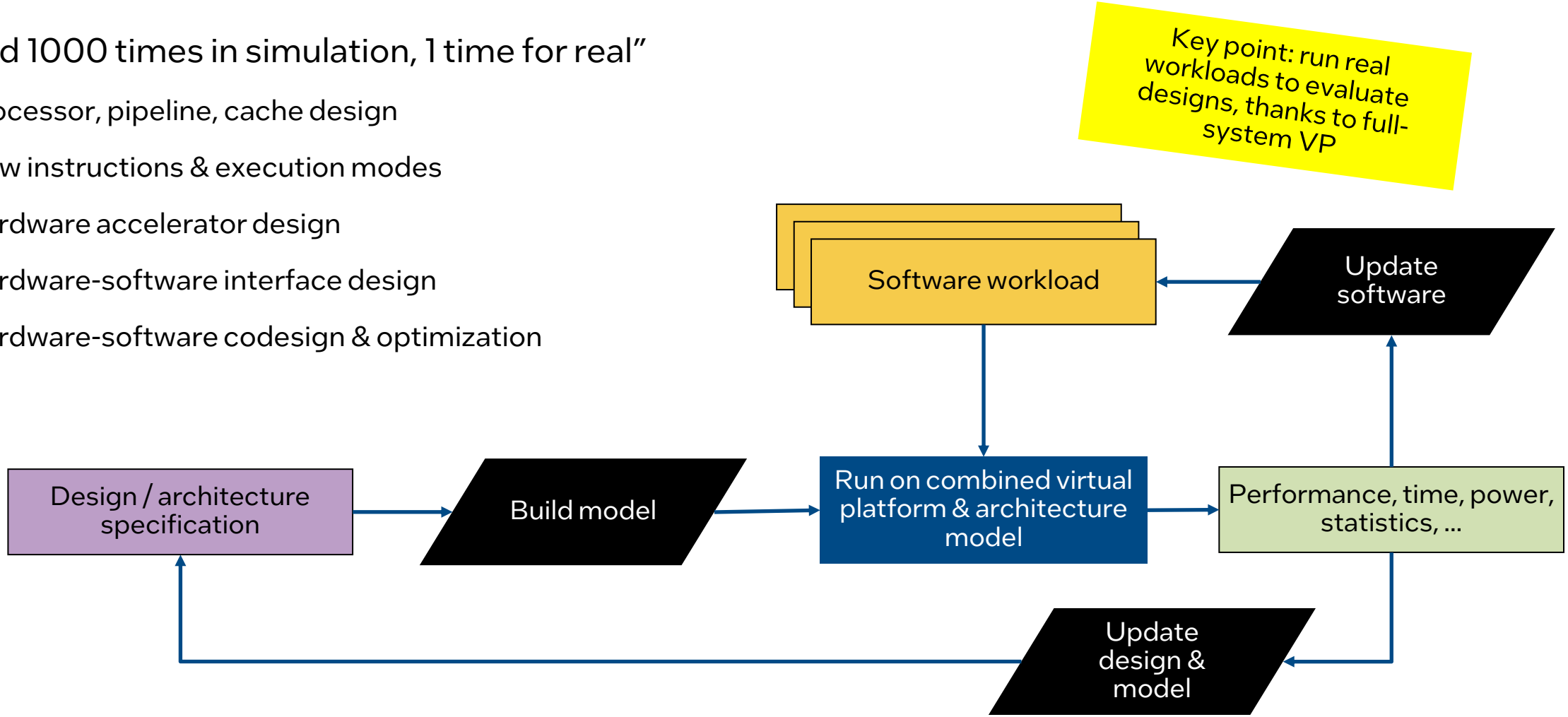


Getting the Architecture Right

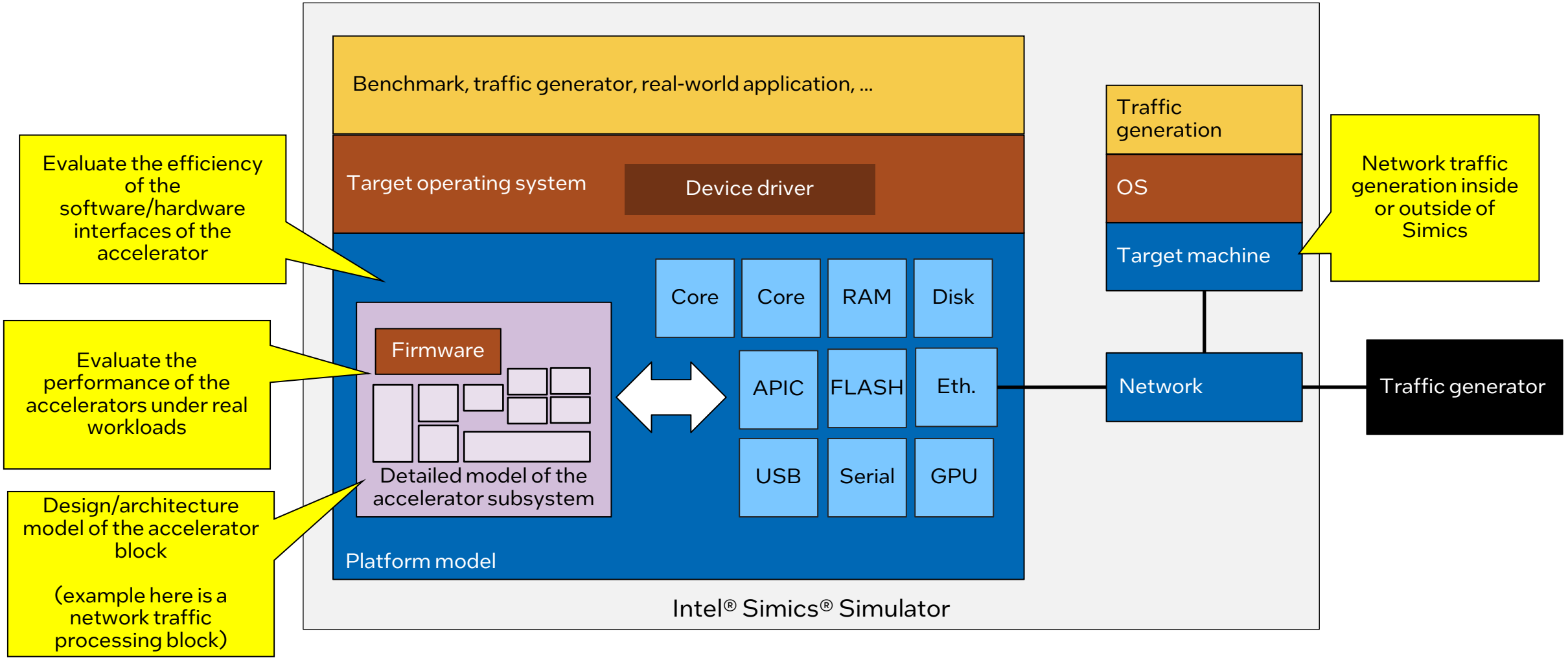
Computer Architecture (on Virtual Platform)

"Build 1000 times in simulation, 1 time for real"

- Processor, pipeline, cache design
- New instructions & execution modes
- Hardware accelerator design
- Hardware-software interface design
- Hardware-software codesign & optimization



Computer Architecture: for Subsystem



Instruction-Set-Level Computer Architecture

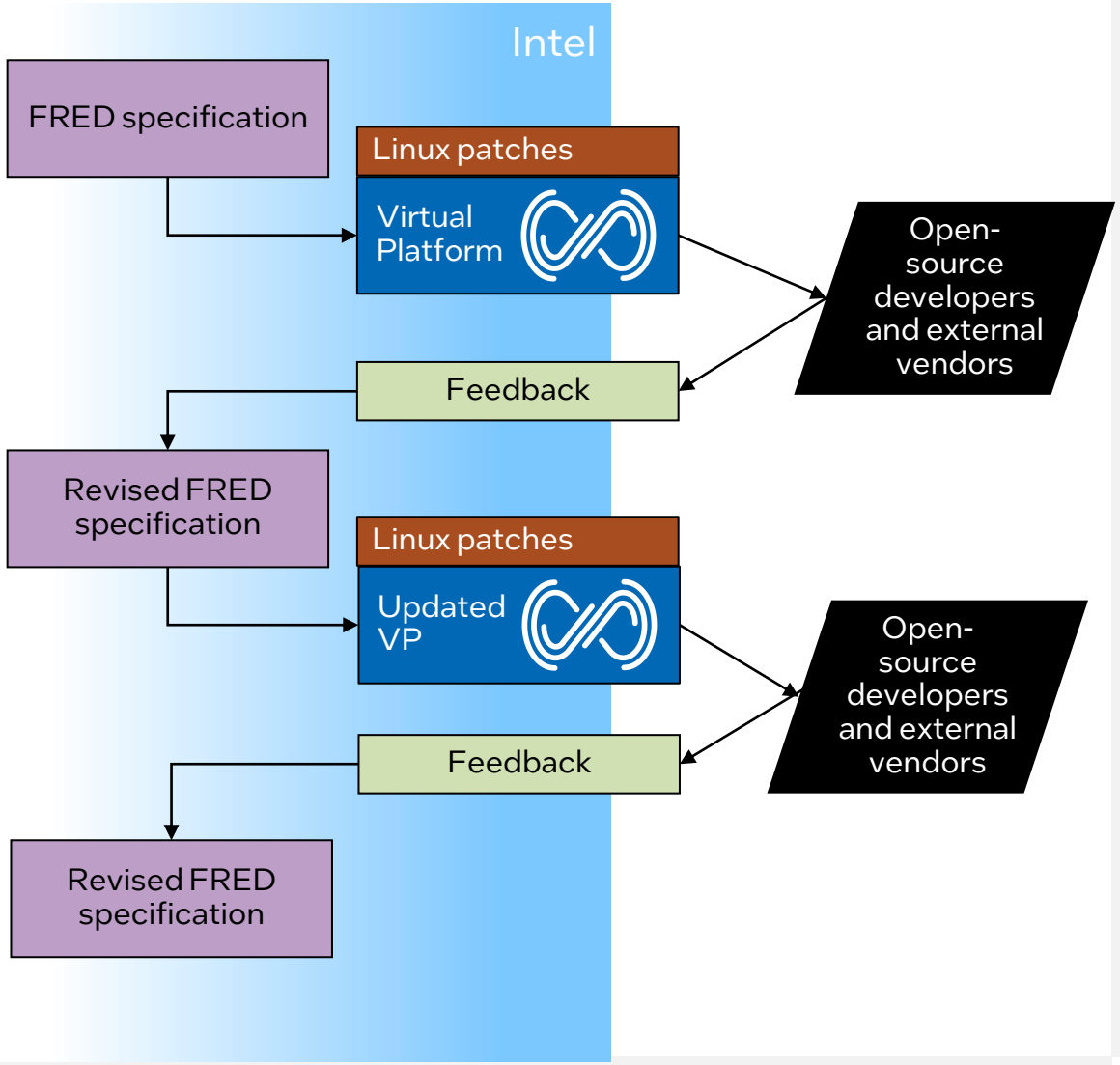
Architecture happens at the instruction-set level as well as microarchitecture

Example: “Flexible Return and Event Delivery” (FRED)

- New way to handle exceptions and interrupts in the Intel Architecture

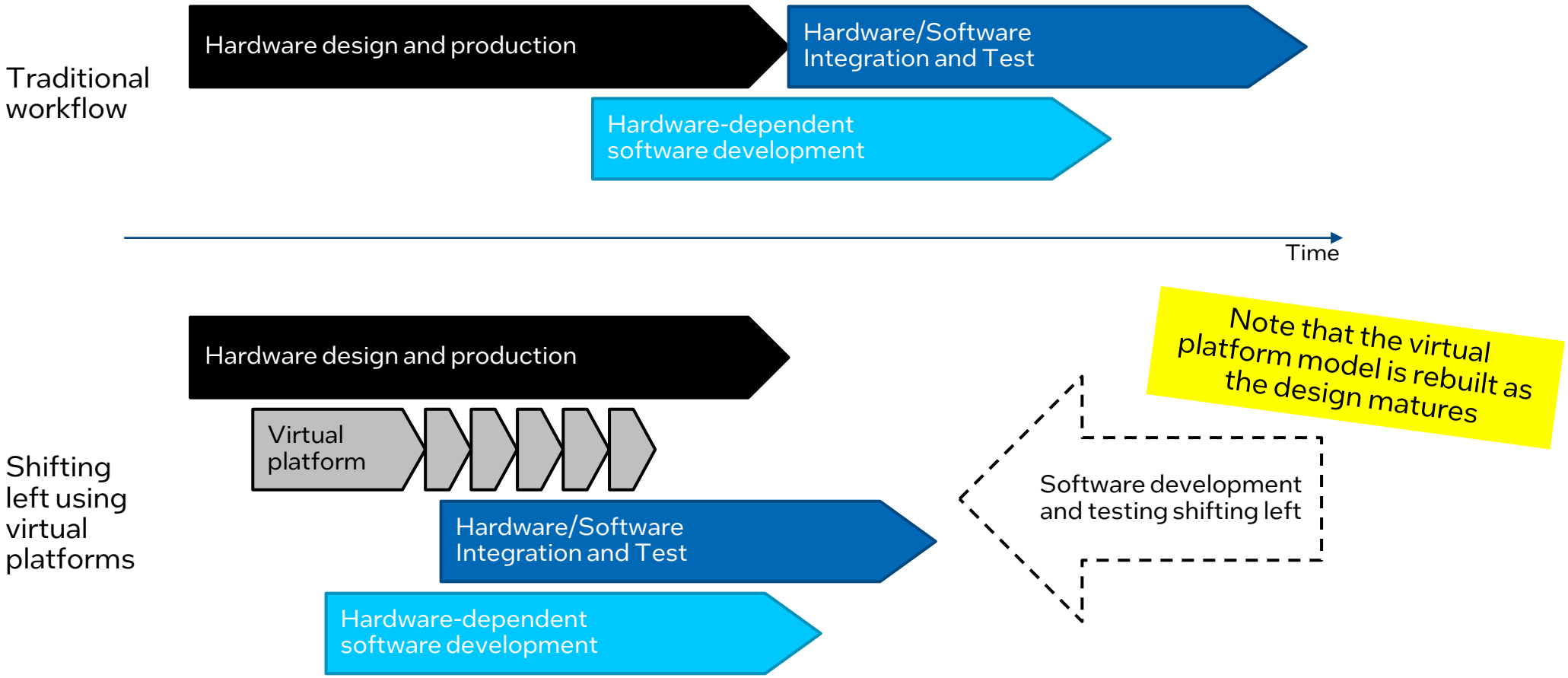
Intel® Simics® virtual platforms used as “test hardware” for external software developers

- For Linux developers, provided together with Linux kernel patches from Intel’s Linux developers – software is needed
- Collect feedback from external (operating-system) developers, improve the design



Getting Software Done Early

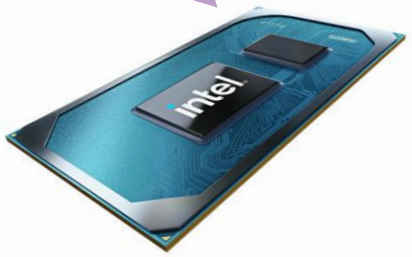
Shift-Left / Early Software Development



Classic use case – Earliest examples from the 1950s

Shift-Left: With the Ecosystem

Silicon vendor builds basic code, makes sure the platform works



Chip/Platform

Board designer adds more components, ports more operating systems, validates additional functionality

(Custom) Board

Typically, this is a customer of the silicon vendor, a separate company

OEMs build on the boards to build complete products.

Digital twins

Virtual system integration

OEM Product

Could be the same company as the board designer, or yet another company

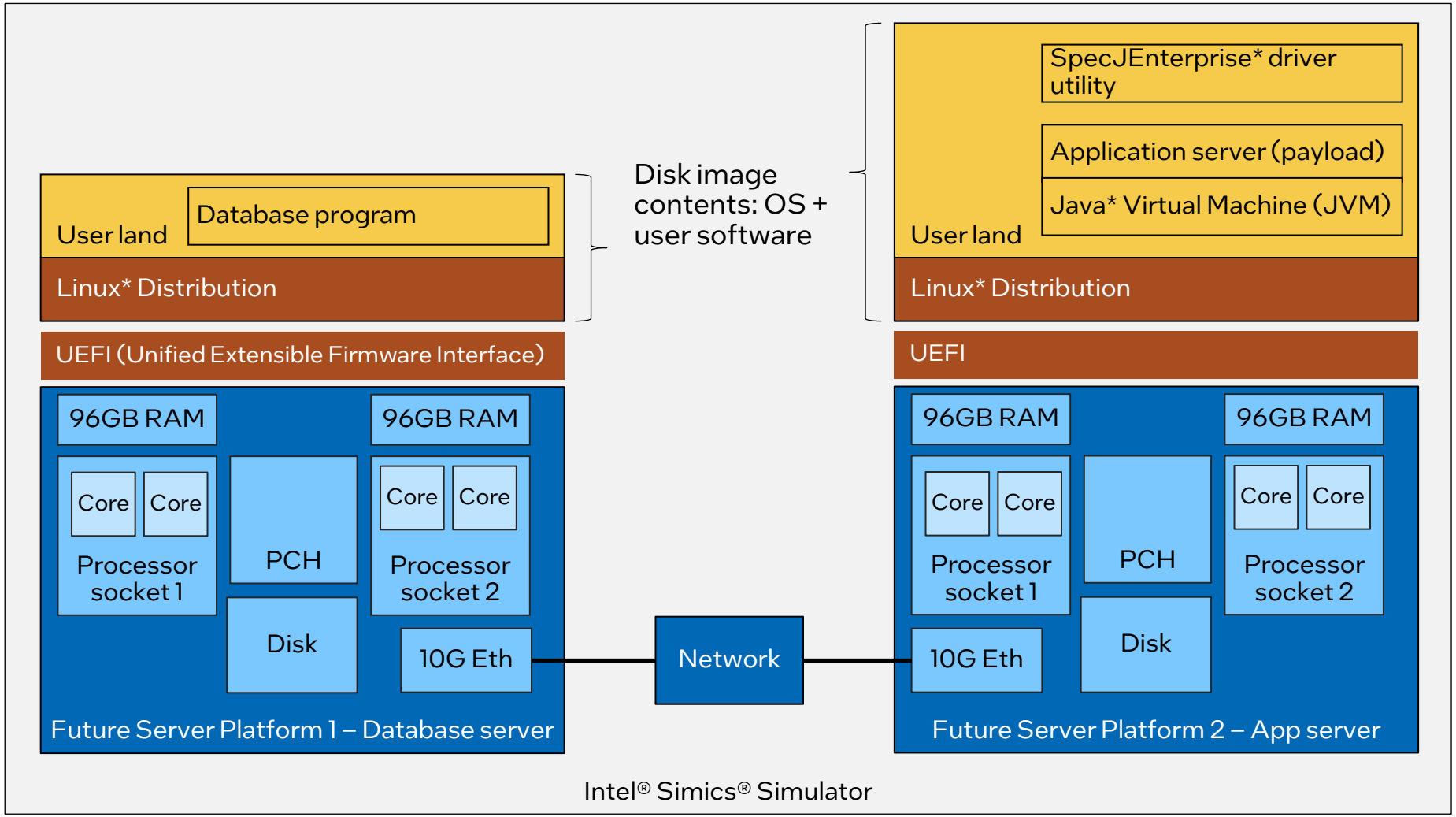
Testing Software at the System Level

System-Scale Simulation Example

Update software stack to use latest hardware instruction sets and features

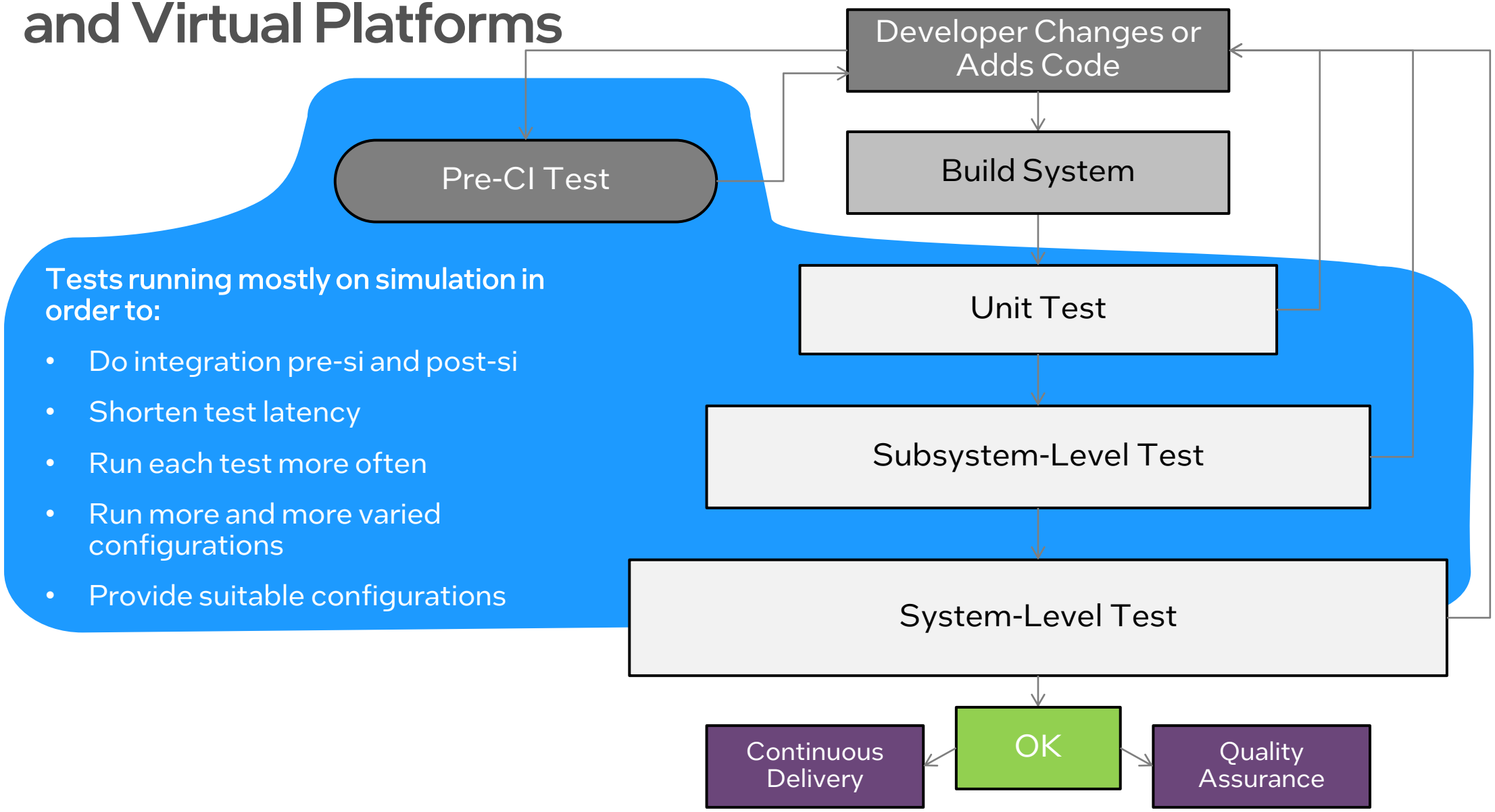
Ensure integration of hardware, boot code, drivers, OS, and applications work – before the silicon arrives

This particular example: silicon vendor + software vendor cooperating on next-gen hardware tuning



https://software.intel.com/en-us/blogs/2018/03/15/software-on-wind-river-simics-virtual-platforms-then-and-now

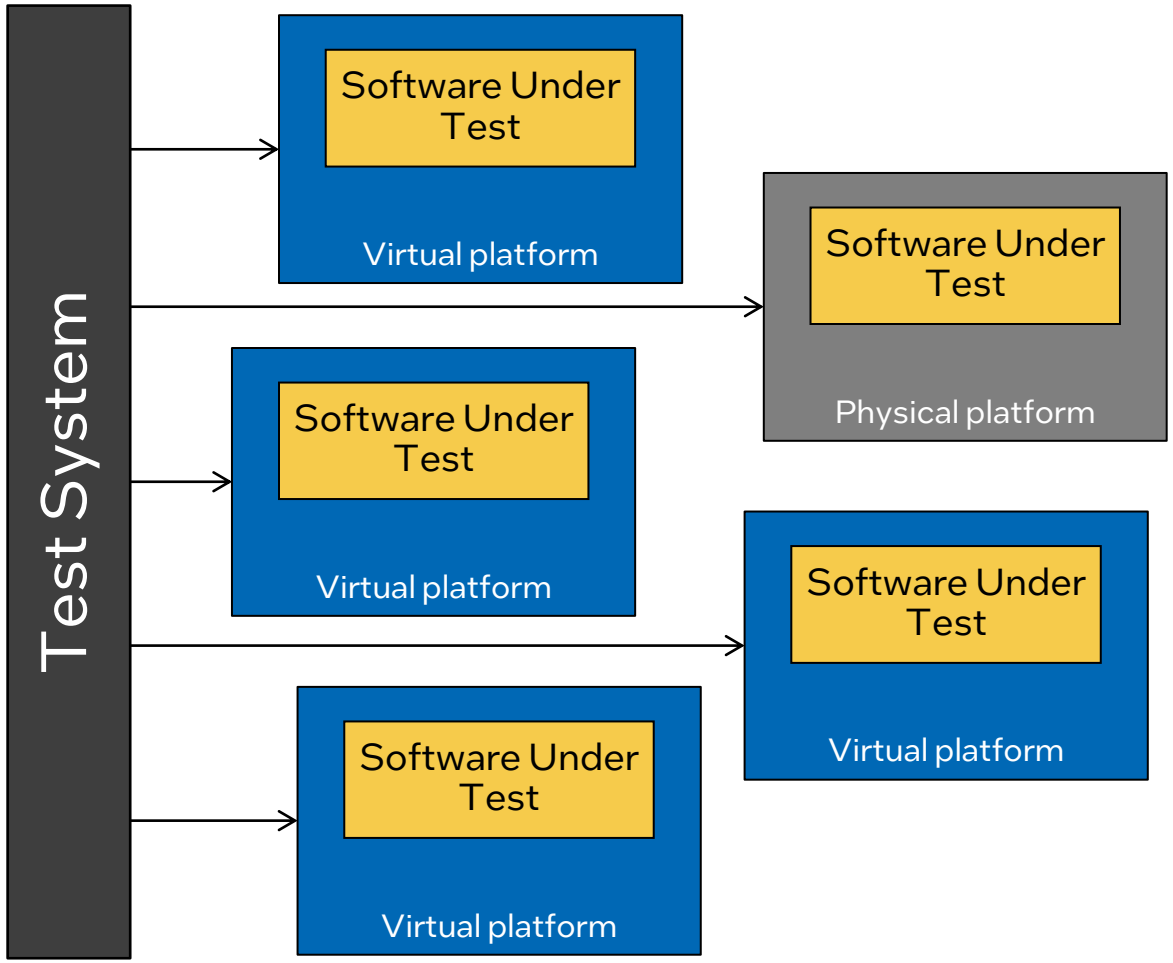
Continuous Integration and Virtual Platforms



Tests running mostly on simulation in order to:

- Do integration pre-si and post-si
- Shorten test latency
- Run each test more often
- Run more and more varied configurations
- Provide suitable configurations

Allowing More Tests for Difficult Hardware



Hardware availability is often a bottleneck in embedded systems testing

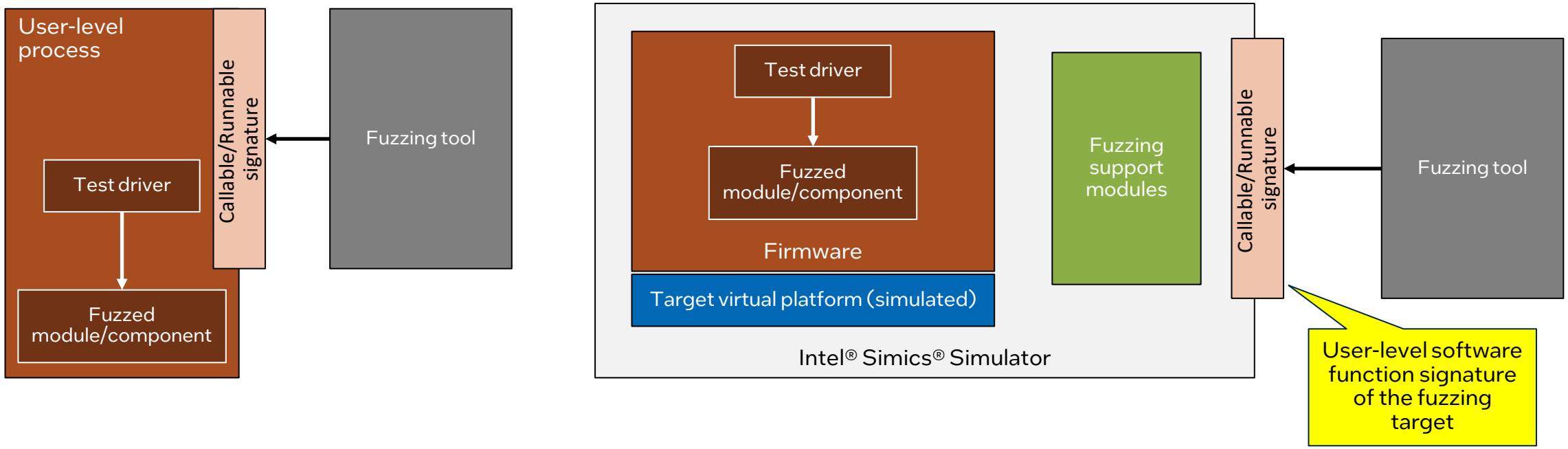
Classic customer case

- Hardware = Integration testing every *week*
  - Bugs creep back in
  - Impossible to go Agile
- Virtual platforms provided more targets
  - Integration testing *daily*
- = Higher quality, less rework, more agile development flow

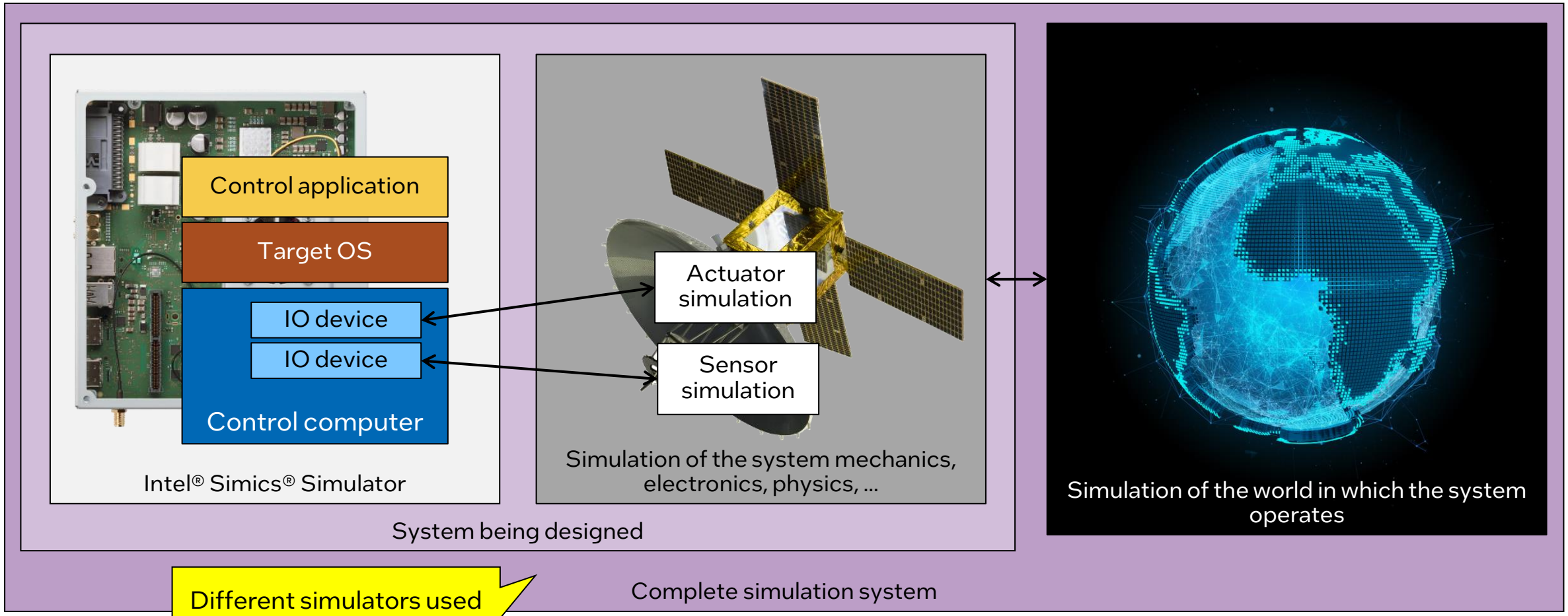
Virtual-Platform-Based Guided Fuzzing

Concept: Make the virtual platform look like a user-level program

- Reuse existing fuzzers and their fuzzing logic as-is…
- … while facilitating access to the firmware using virtual-platform techniques



Integrating Environment Simulation



Different simulators used for different parts of the complete simulation

Intel® Simics® Simulator Technology

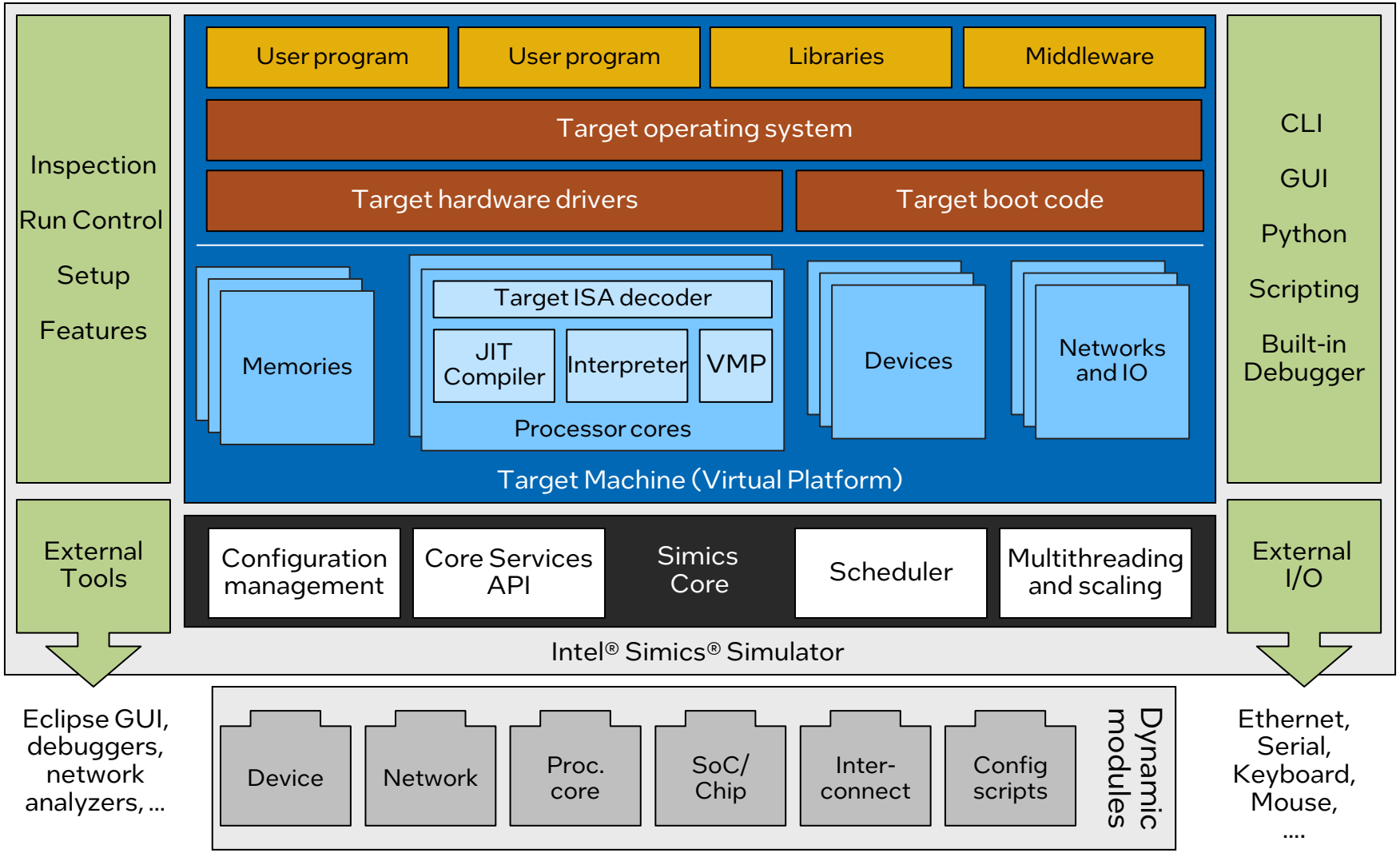
Intel® Simics® Simulator Architecture

All devices, processors, etc., are loaded dynamically from modules.

Running simulation consists of a large number of objects (10000-100000 for an Intel platform)

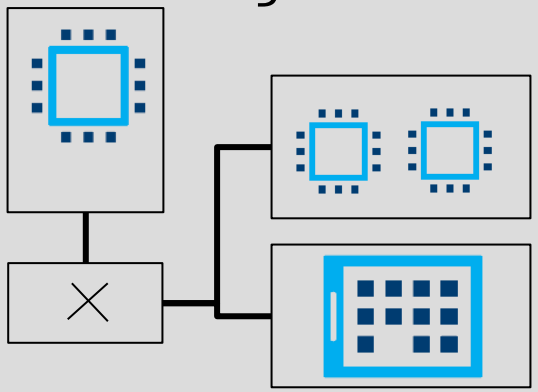
All connections with the outside world should pass through dedicated interface objects – models should **never** directly talk to the host.

Features are built outside of the target system and should apply across different system models.



System Level Features

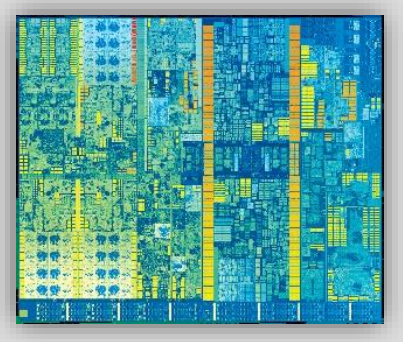
Scalable & heterogeneous



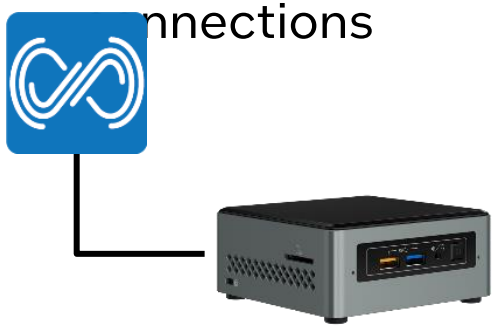
Scripting

```
con0.wait-for-string "$“
con0.record-start
con0.input "./ptest.elf 5\n"
con0.wait-for-string "."
$r = con0.record-stop
if ($r == "fail.”) {
   echo ”test failed”
}
```

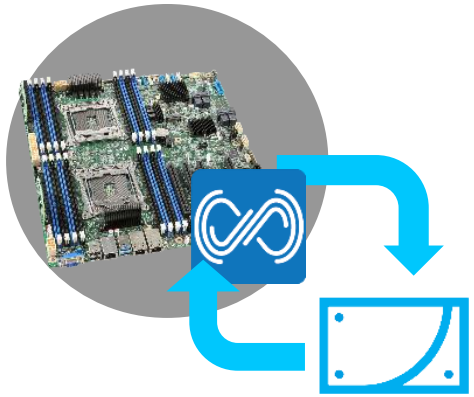
Bit-exact function



Real-world connections

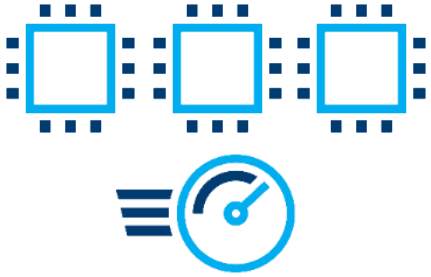


Checkpoint and restore

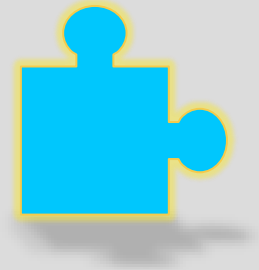


Fault injection & control

Multicore & -machine multithreading

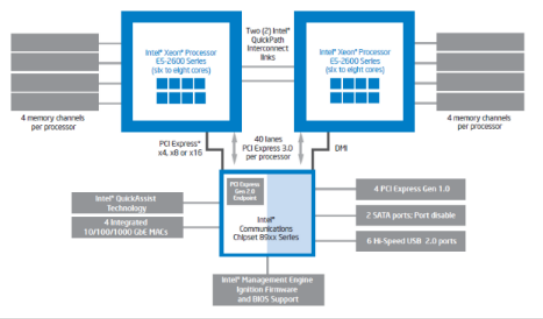


Modular & user-extensible

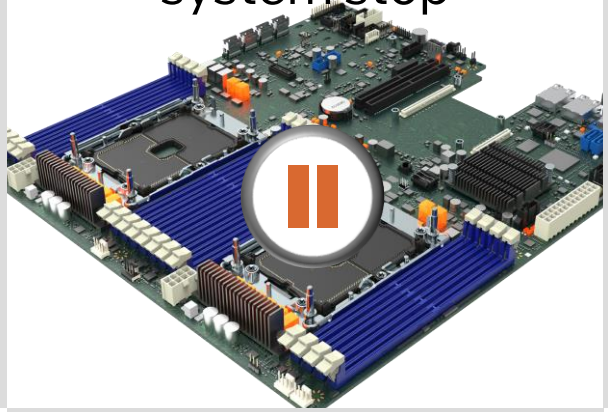


Debugging Features

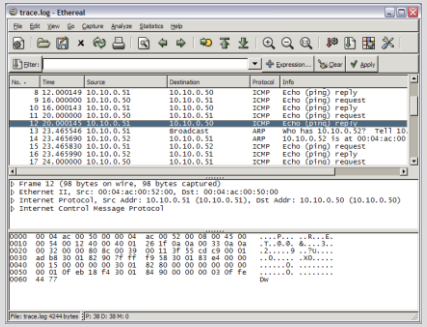
Insight into all components



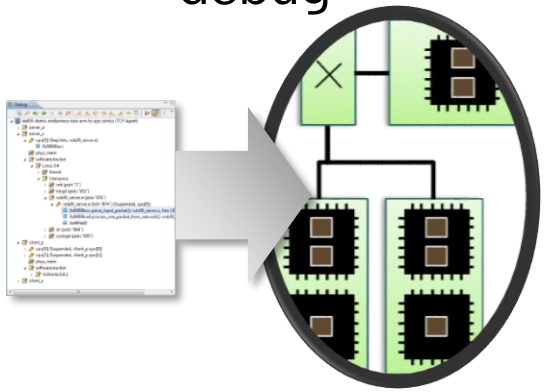
Synchronous entire-system stop



Trace anything



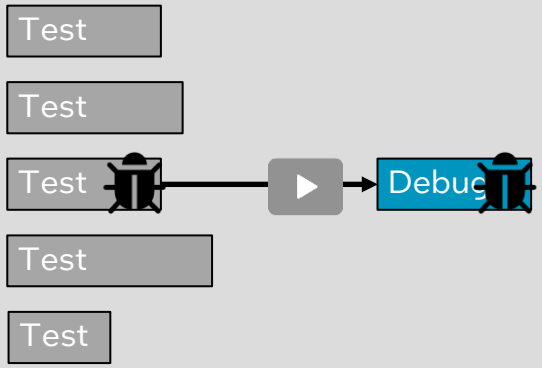
System-level symbolic debug



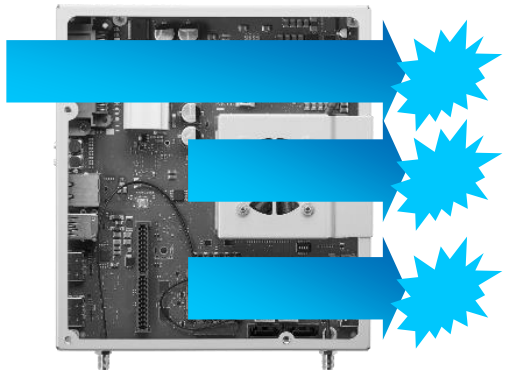
Unlimited powerful breakpoints

```
Code execution
Exceptions
Register changes
Memory accesses
  On arbitrary areas
Hardware device accesses
Time
Simulator events
```

Record-replay debug



Repeatability & snapshots



Collaboration between developers

How to build a fast virtual platform

Fast Instruction-Set Simulator (ISS)

Functional abstraction level
Just-in-time compilation (JIT)
Virtualization
Simplified timing
Temporal decoupling

Fast Device Models

Transaction-Level Modeling (TLM)

Event-driven simulation

Simplified timing

Efficient Framework

Reduce overheads

Multithreading

Optimize, optimize, optimize

Tailored Configurations

Configurations optimized for each use case

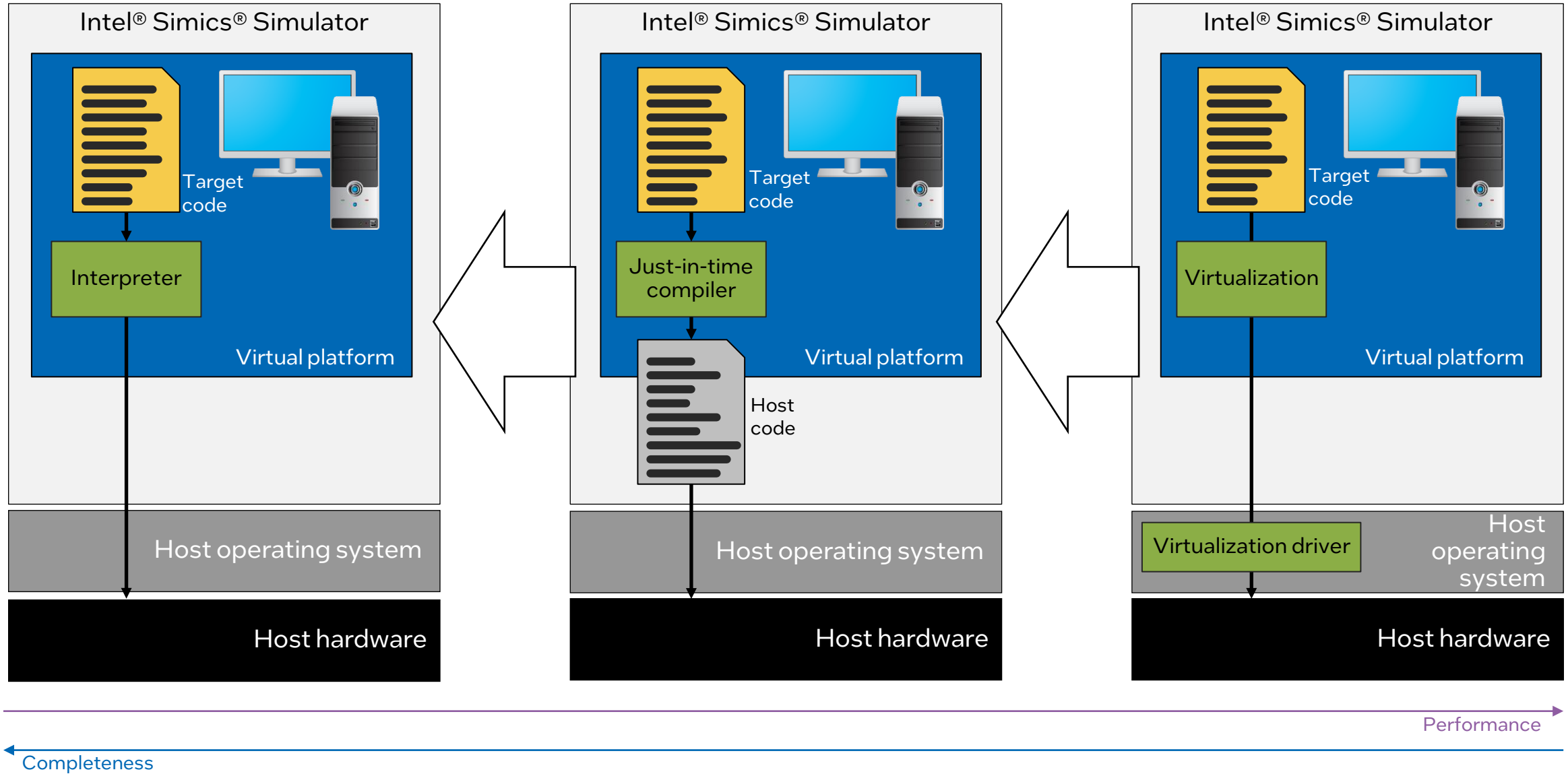
Highest-possible level of abstraction

Use different models in different cases

What Performance do you Need for a Particular Use Case?

Slowdown	Use cases where this works	Notes
1/10	Long-term testing of a mostly idle system	Depends on system being idle, or very slow in the real world
1	Hardware-in-the-loop	Simulators trying to stay locked to real-world time
10	Edit-compile-test, volume software testing, interactive usage of virtual platform	
100	Automatic testing of complex setups	Slow execution due to large models or some details in the models
100000	Computer architecture, detailed performance modeling, run short segments of code	Design models run at 100k slowdown or more

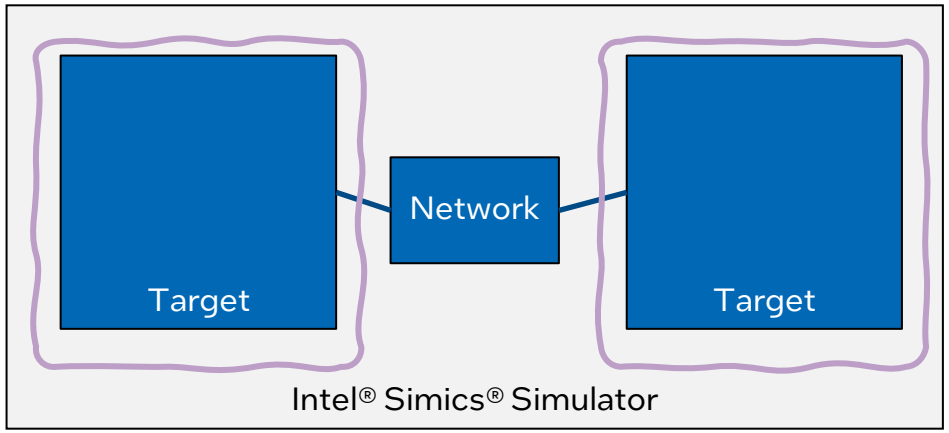
Intel® Simics® Simulator Instruction Execution



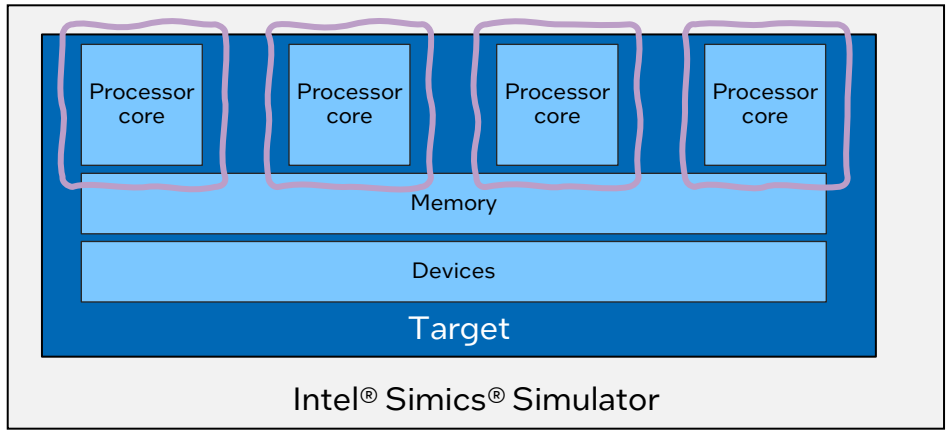
Threading in the Simics® Simulator – Use Cases

List is not exhaustive, and modes are often combined

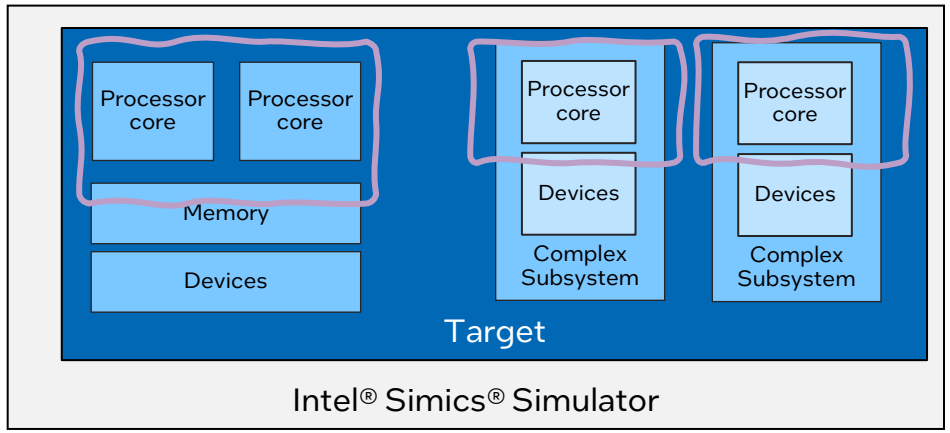
"Multimachine Accelerator"
Thread across long-latency networks



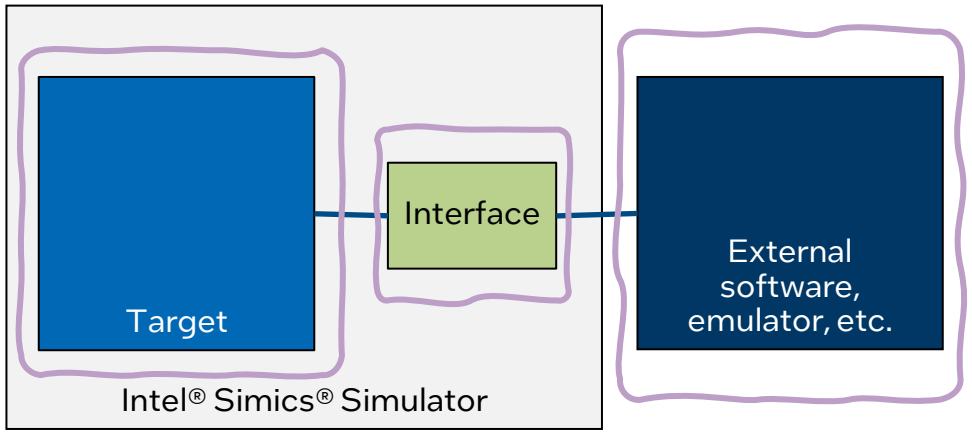
"Multicore Accelerator" (MCA)
Thread between processor cores sharing memory



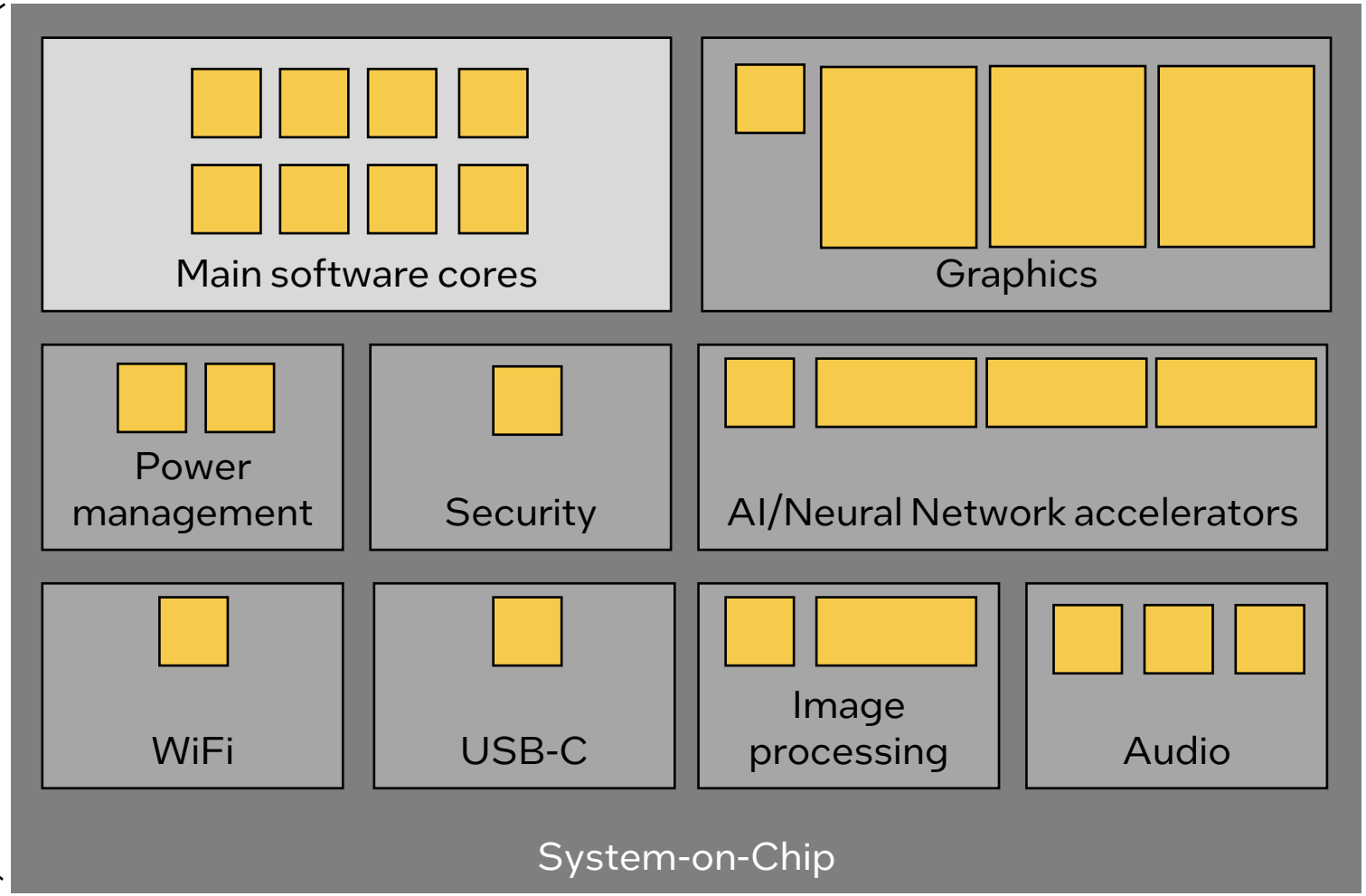
"Subsystem multithreading"
Run separate (definition) subsystems on their own threads



Multithreading as coding pattern
Use a thread to interact with the outside asynchronous world



Inside a Modern System-on-Chip (SoC)



Result: Virtual Platforms get Bigger Over Time

```
Processors: 2
Devices:  100
Subsystems: 0

Objects: 200
```

2010 Model

```
Processors:  2-200
Devices:     2_000
Subsystems:     10

Objects:   100_000
```

2024 Model

More of the system is modeled, in more detail

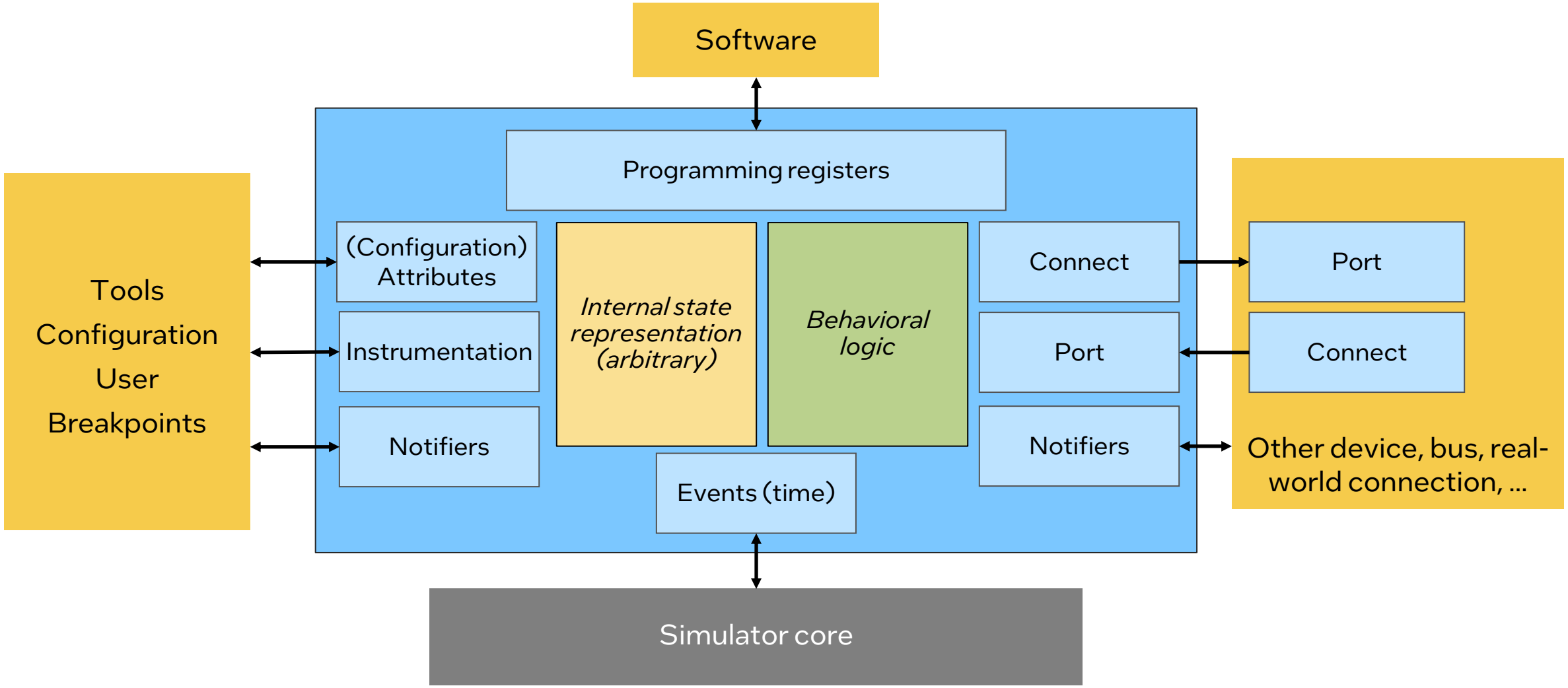
Buses and interconnects (20+)

Processor cores (100+)

Devices (10000+)

Intel model library

Parts of a Device Model – More Details



The Device Modeling Language

https://github.com/intel/device-modeling-language

Make device modeling easy

- Make it hard to write bad models

Provide natural modeling constructs

- Register, bit field, banks, connects, …
- Readability and maintainability
- Easy to generate register layouts from machine-readable specifications

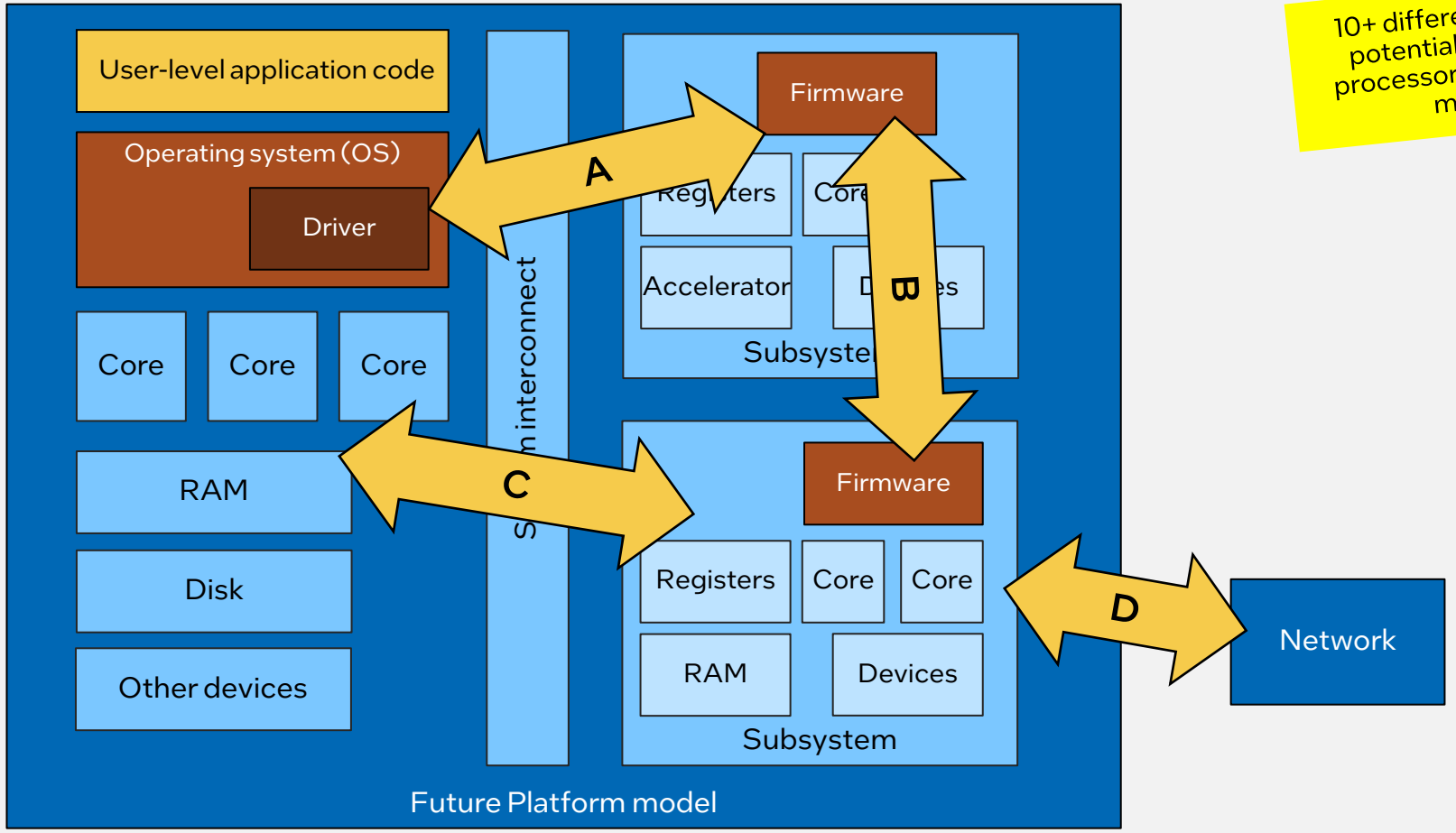
Powerful templating mechanisms

- Common behaviors
- Common types of devices
- Support library behind code generation
- …

Generates C code with Intel® Simics® Simulator API calls

```
// Example device model
dml 1.4;
device example_i2c_device;

import "simics/devs/i2c.dml";  // generic i2c
import "platform-i2c.dml";     // i2c logic shared with other platforms
import "fuse-common.dml";      // common platform fuse mechanisms

// generated code with register declarations
import "DevBank_gen_code.dml";

// instantiate the register bank from the file
bank regs is i2c_ctrl_reg_bank {
  register hst_cnt {           // Added manual code
        method write_action() {
            if (START.get() != 0) {
                START.set(0);
                send_start();
} } } }


// Generated file DevBank_gen_code.dml
dml 1.4;
import "access_templates_14.dml";

template i2c_ctrl_reg_bank {
    param    bank_reset_signal default undefined;

    register hst_sts        @ 0x00 is (read_write) "Host Status";
    register hst_cnt        @ 0x02 is (read_write) "Host Control";
    // array of registers
    register tx[i < 8]      @ 0x08 + i is (read_write) "Transmit data";

    // flesh out fields in hst_sts
    register hst_sts {
        field BYTE_DONE_STS @ [7:7] is (write_1_clears);
        field INUSE_STS     @ [6:6] is (write_1_clears);
        ...
```

Working with Firmware and Subsystems

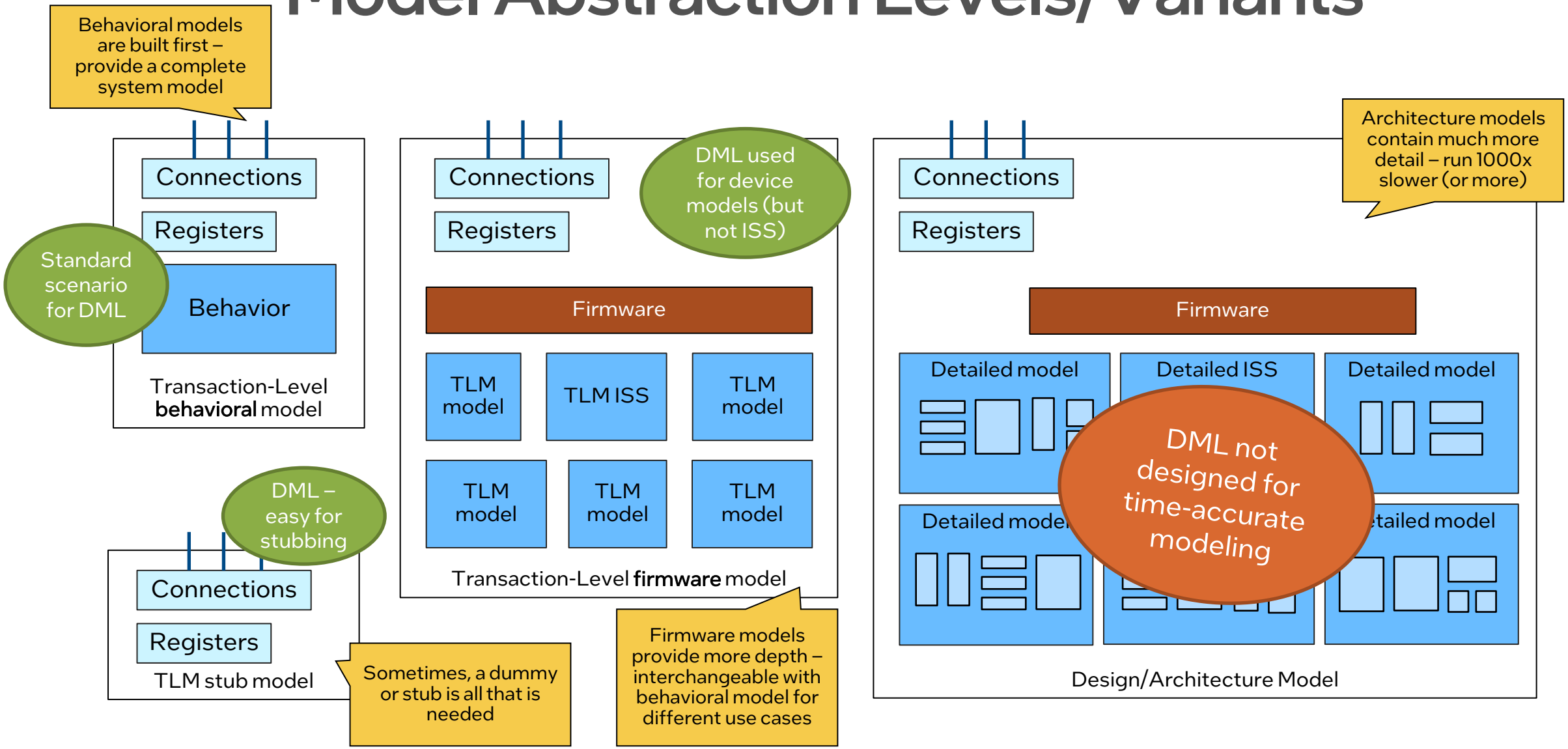


Intel® Simics® Simulator

10+ different subsystems, potentially 10+ separate processor types, alongside main cores

A. Test driver interaction with subsystem firmware
B. Test how different subsystems interact – integration is always "fun"
C. Test firmware interaction with other hardware
D. Test firmware interaction with external world

Model Abstraction Levels/Variants



Coding in All the Languages ☺

	C/C++	Python	Device Modeling Language	SystemC	Javascript	Rust	Simgen	Scripts of various types
Simulator core	X	X						X
User features	X	X						X
User interfaces		X			X			X
Utilities and tools	X	X			X	X		X
Processor models							X	X
Device models	X	X	X	X				X

Open for questions and discussions!

Public Release of Intel® Simics® and Intel® Integrated Simulation Infrastructure with Modeling (Intel® ISIM)

Download and Learn More at

https://software.intel.com/intel-isim

Legal Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete http://www.intel.com/performance.

intel®