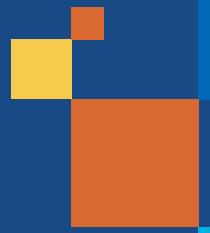
Intel® Simics® Virtual Platforms in Embedded Systems and Silicon Engineering

Jakob Engblom, Director, Simulation Technology Ecosystem



intel®

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

Intel Worldwide Headquarters
Santa Clara, California

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
  - Added features and improvements added within major
  - Integration with power, thermal, performance models
- Next: clean-up release to remove old features

Where do we Fit into Intel?

Get our software for free at https://developer.intel.com/intel-isim

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

Laptop and desktop

- Intel® Xeon®
- Chipsets
- Infrastructure processing units (smart network)
- GPU
- Bitcoin Miners (BZM)

Data Center

- Movidius
- Habana
- Intel® Xeon®
- GPU

AI and ML

- Ethernet
- WiFi
- Bluetooth
- GNSS

Connectivity

- SoC-FPGA
- FPGA
- eASIC hard-copy

FPGA (to be spun out)

- Quantum computing
- Neuromorphic computing
- Software

Intel Labs

- Intel Foundry Services

Foundry

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

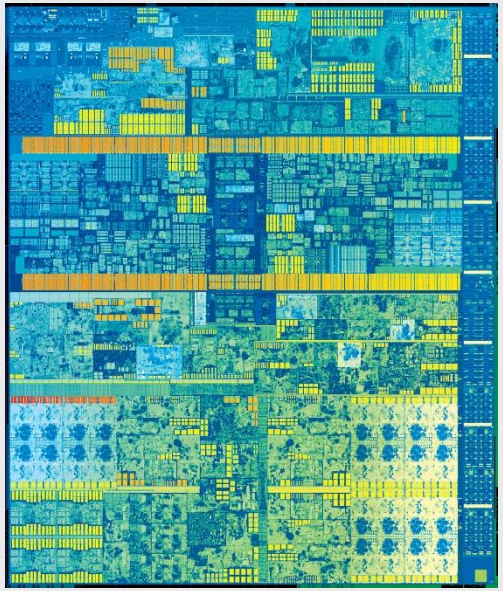
Software

Virtual Platforms Why and What?

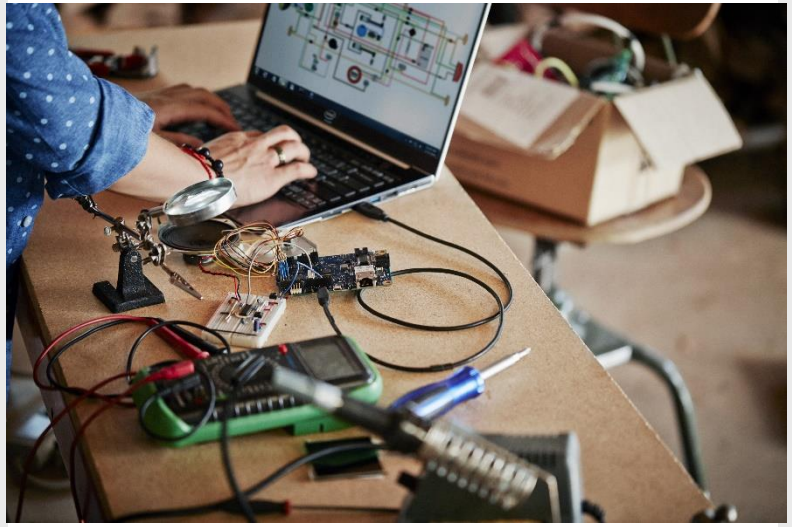
Hardware: A Hard Development Platform?

Hardware is Hard When it is in…

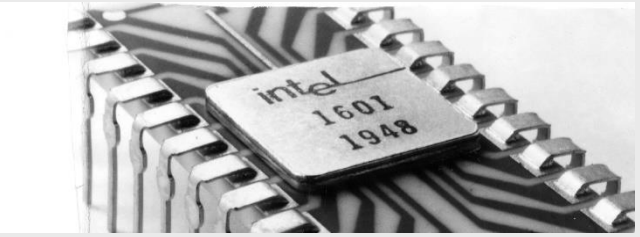
Not yet available



Flaky prototype stage

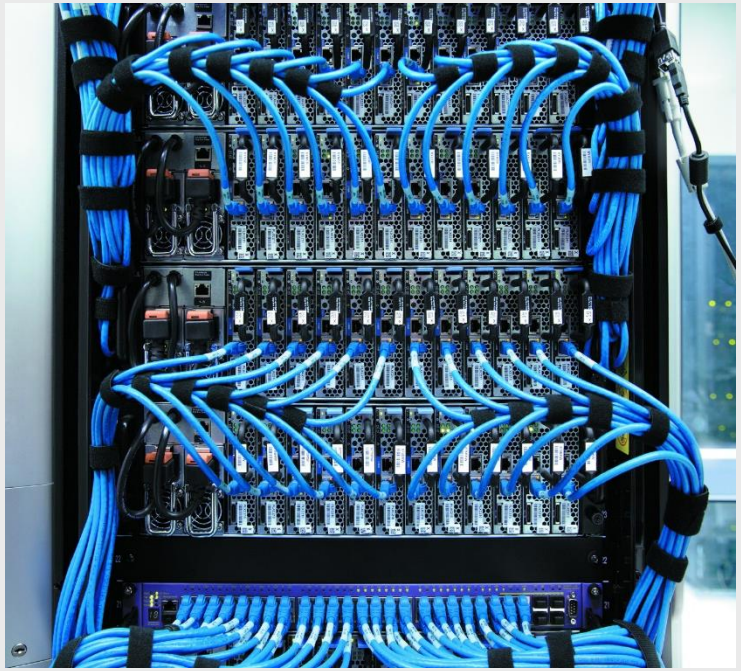


Not available anymore



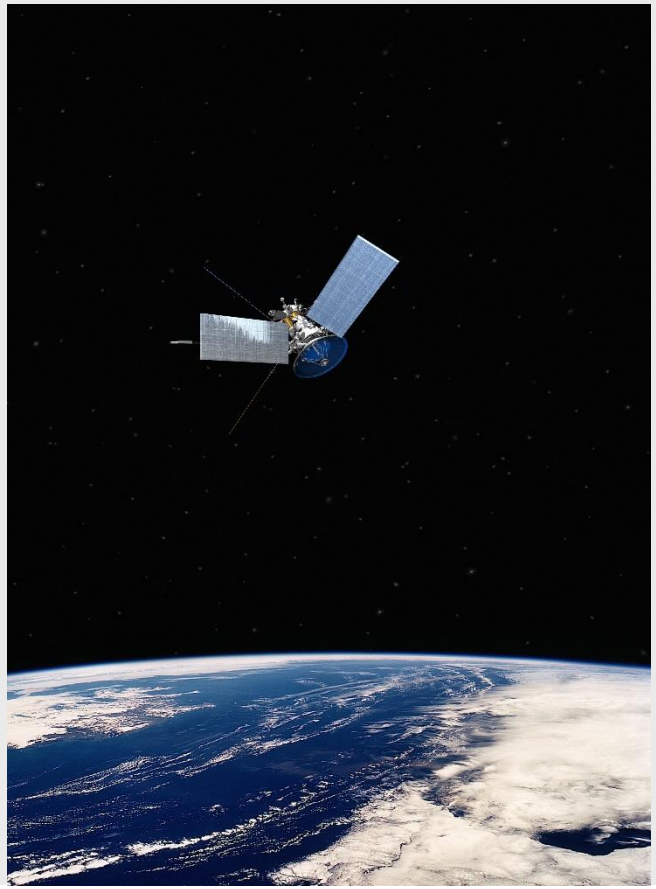
Hardware is Hard When it is...

Inconveniently large & complex

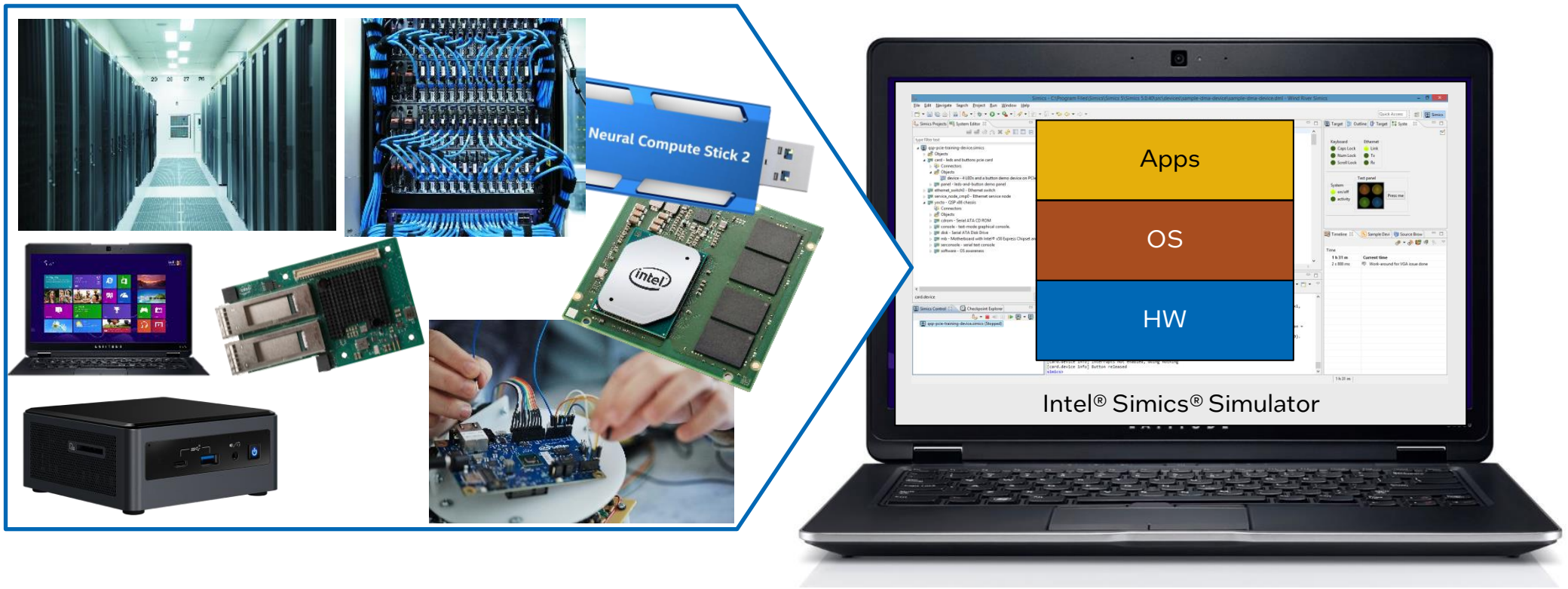


Dangerous to play with

Inaccessible & expensive

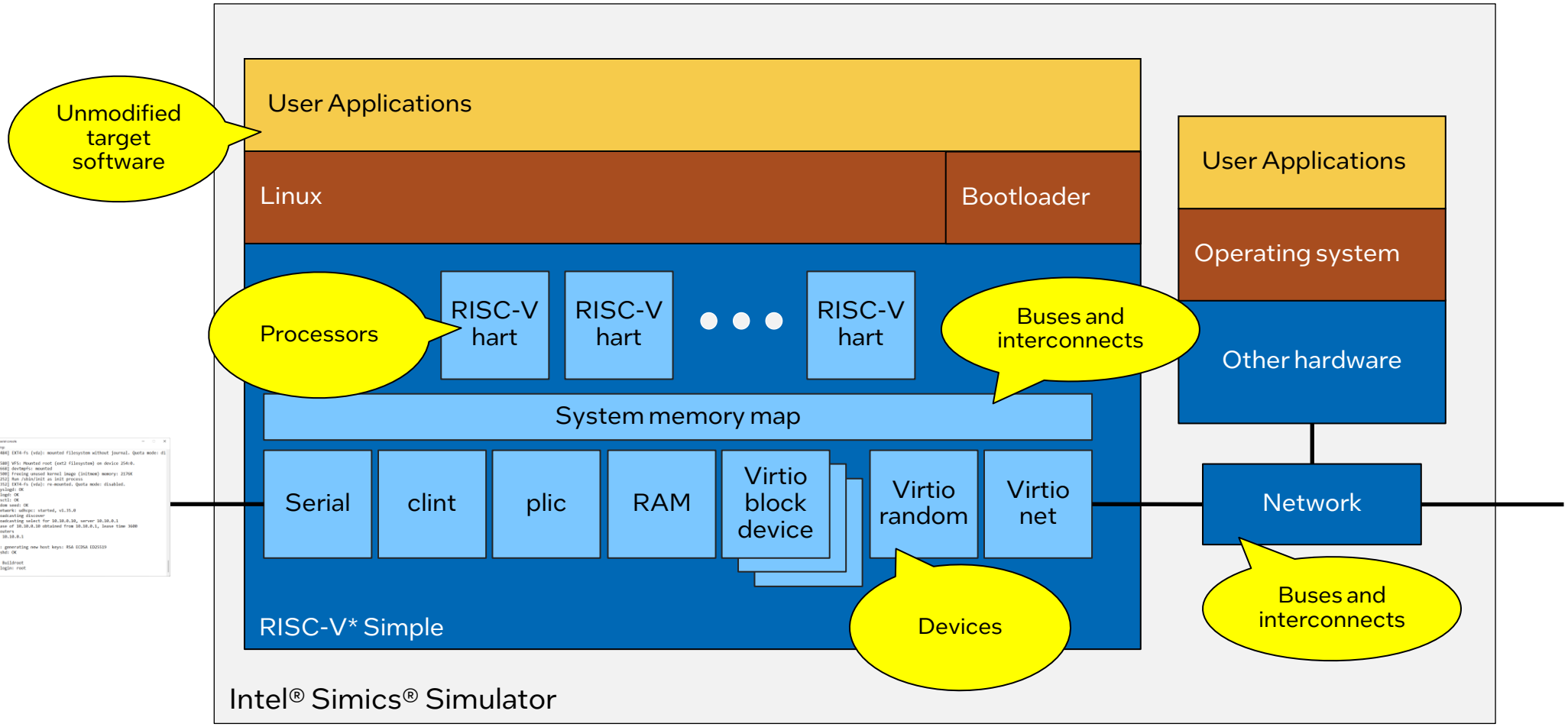


The Idea of a Virtual Platform



Run your software without the hardware – on a software model

Inside a Typical Virtual Platform



Running the Real Software

Purpose:

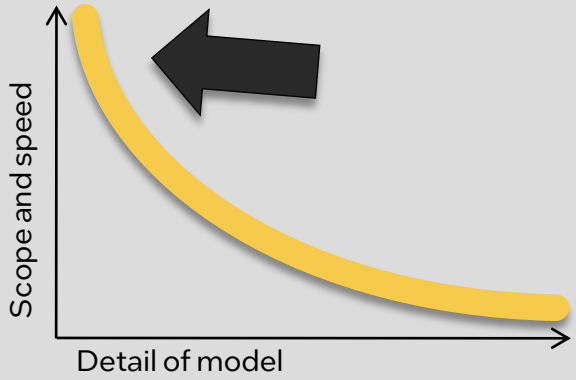
- Test & debug the **software** and the **software-visible** aspects of the hardware

"Software" can mean many things…

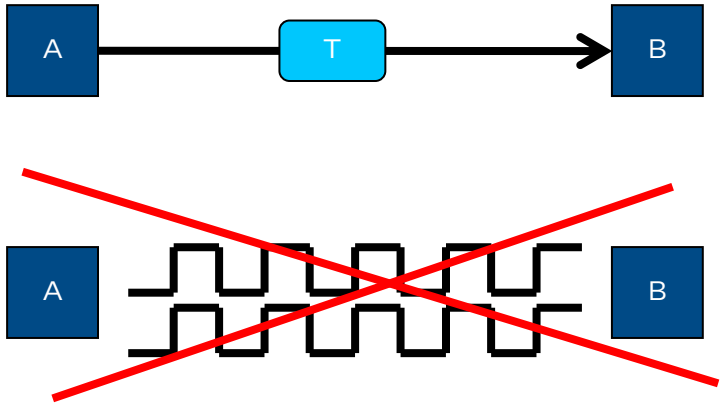
- **Firmware,** that is deeply hidden inside a chip
- **BIOS/Bootloader/UEFI,** that is used to boot the machine
- **Device drivers,** that manage hardware for an operating system
- **Operating systems**
- **Middleware**, providing services for other software
- **Applications,** that any programmer would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

Intel® Simics® Simulator: Level of Abstraction

Goal: Fast & scalable simulation

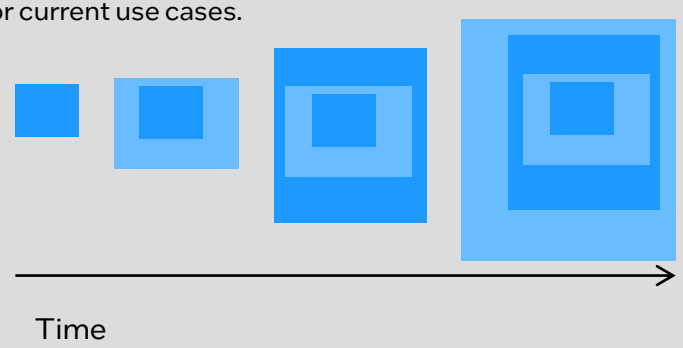


Transaction-level modeling (TLM)

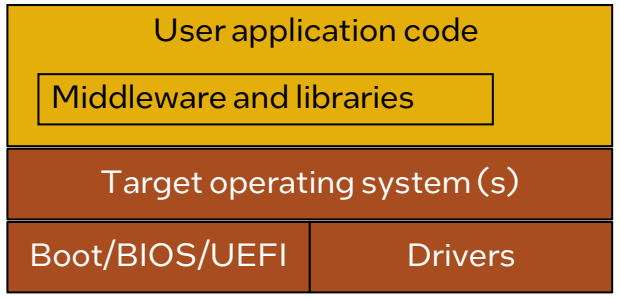


Lazy and agile modeling

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

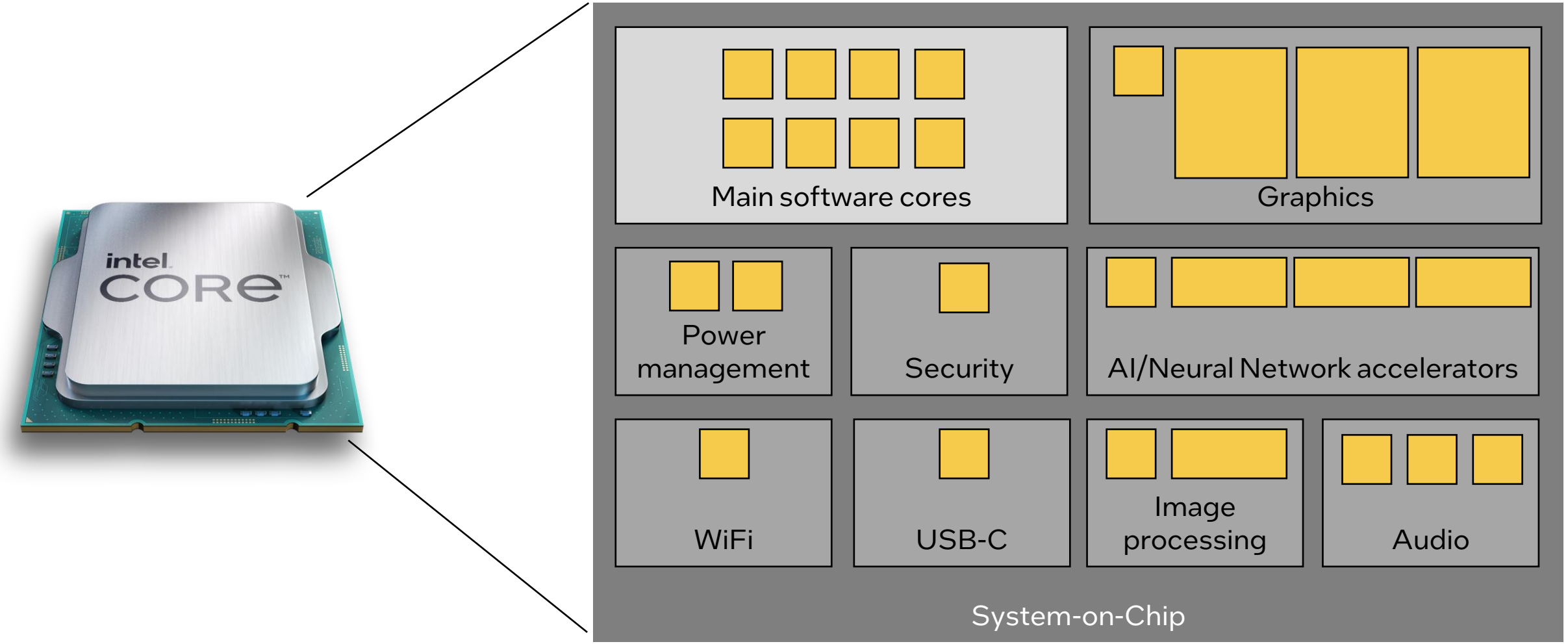
Processor instruction set	System memory map (not bus system)	Device register interface
Loose timing model	Packet-level models of networks	Event-driven simulation, not cycle-driven

Add timing and μarch when needed

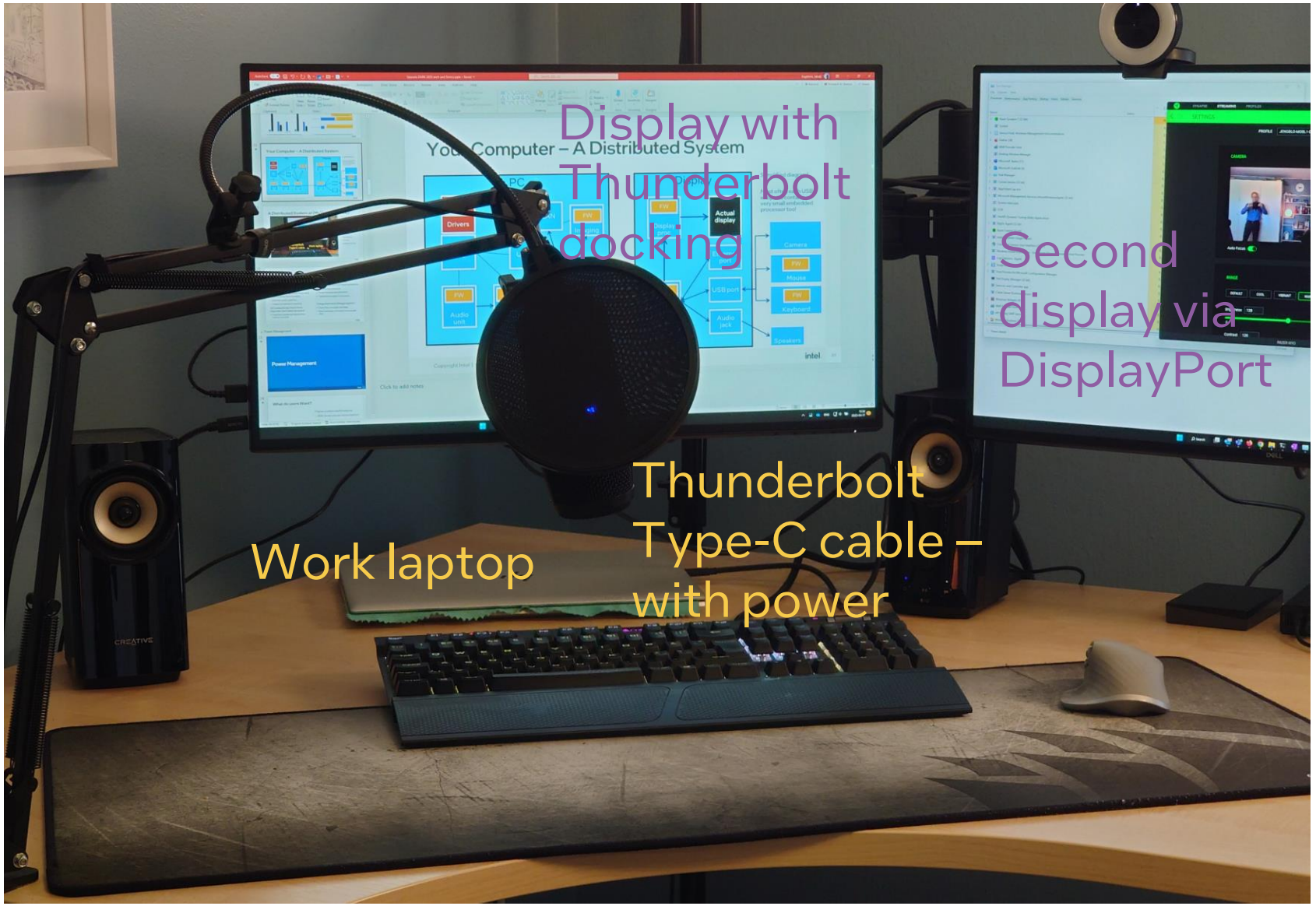
Processor simulators from designers	Cycle-accurate hardware models	Cache model (timing)
Processor timing models	Power models	

Note:
Current Hardware and Modeling Current Hardware

Inside a Modern System-on-Chip (SoC)



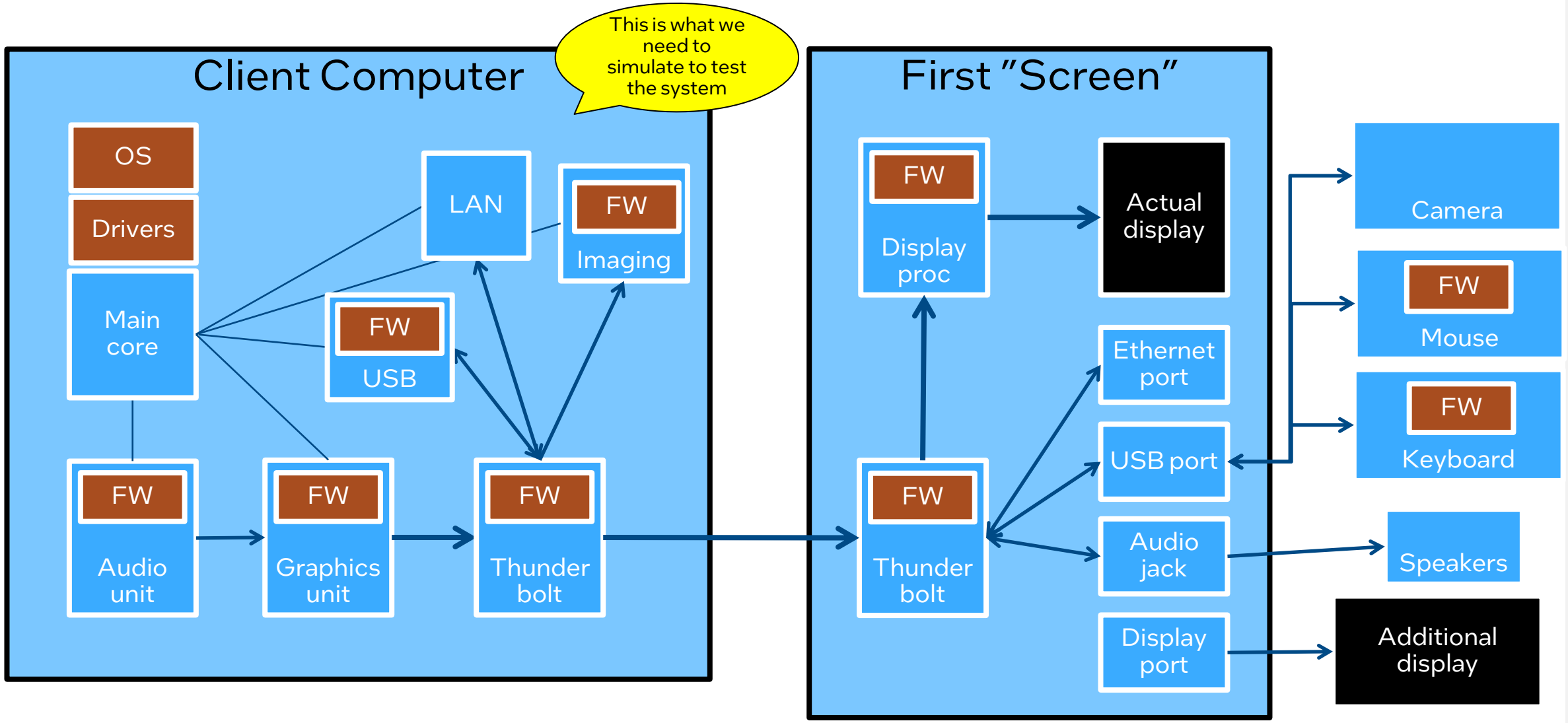
A Distributed System at (My) Home (Office)



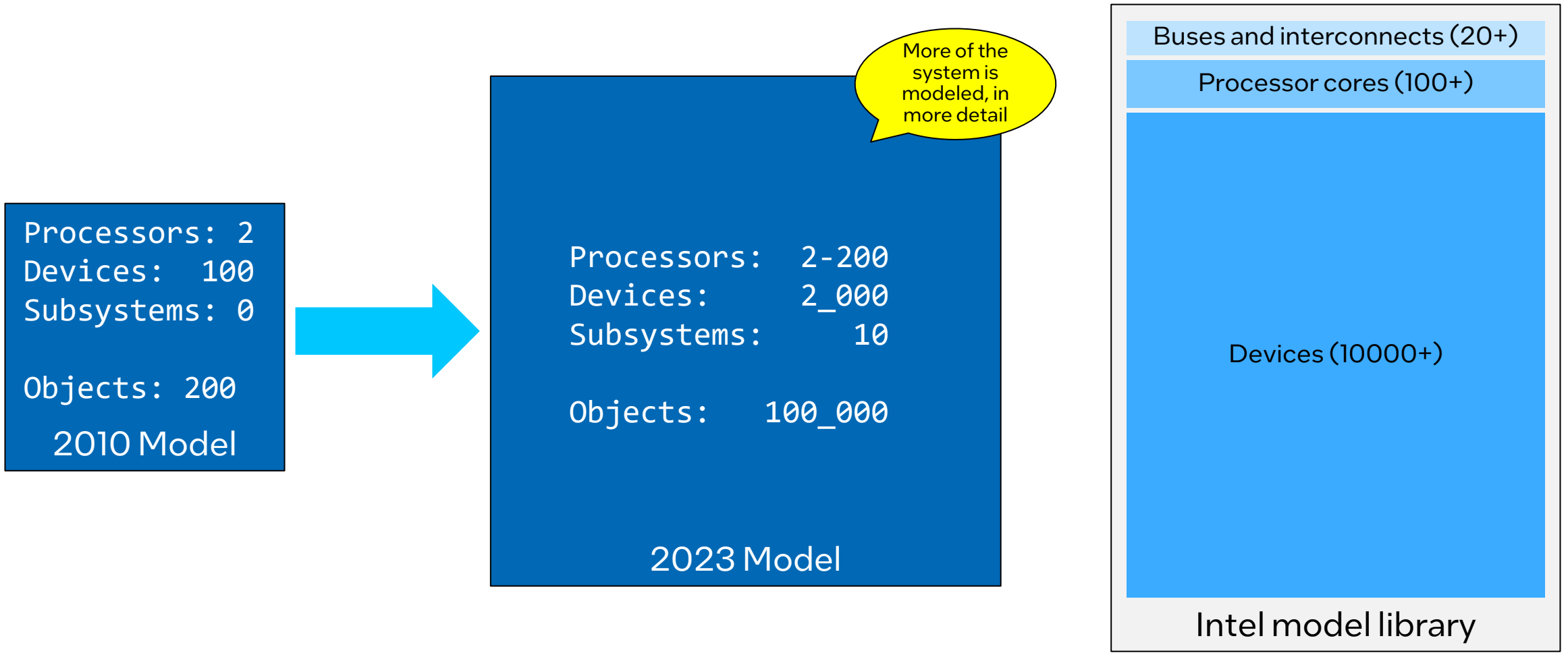
Attached to screen:

- Power – which is fed to laptop over USB Type-C
- Keyboard
- Mouse receiver
- Speakers
- Microphone
- Camera
- Headset receiver
- Second display

Computer Setup = Distributed Systems



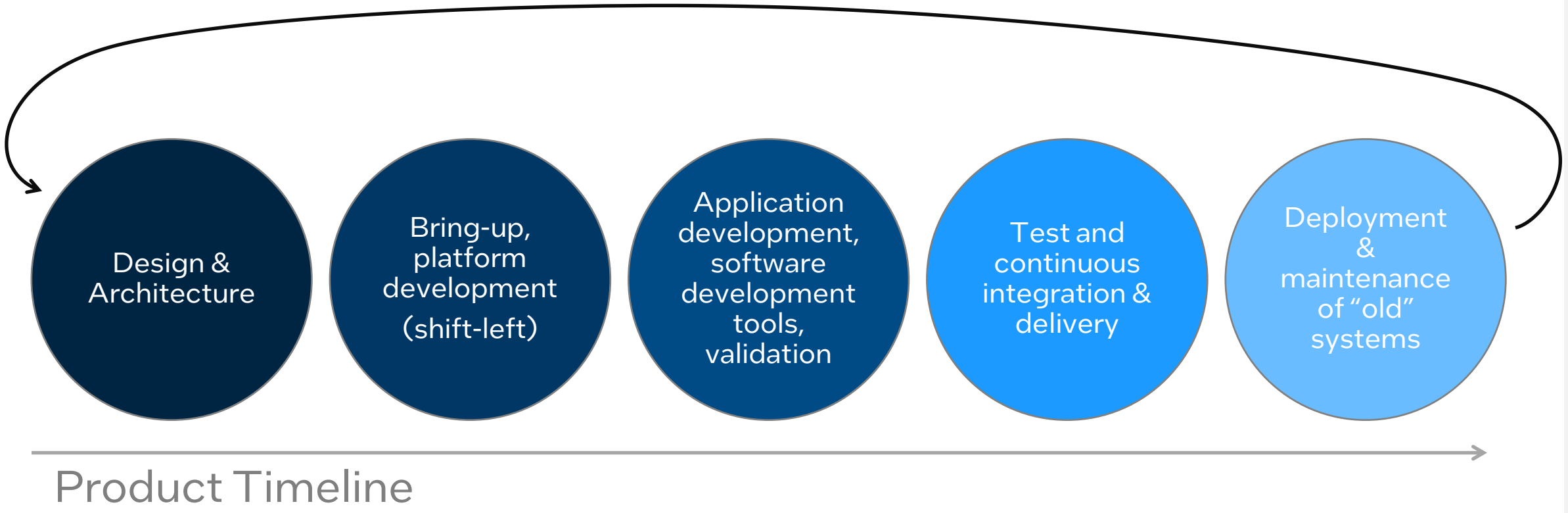
Result: Virtual Platforms get Bigger Over Time



The Intel® Simics® Simulator

Use Cases

Virtual Platforms & the Product Lifecycle

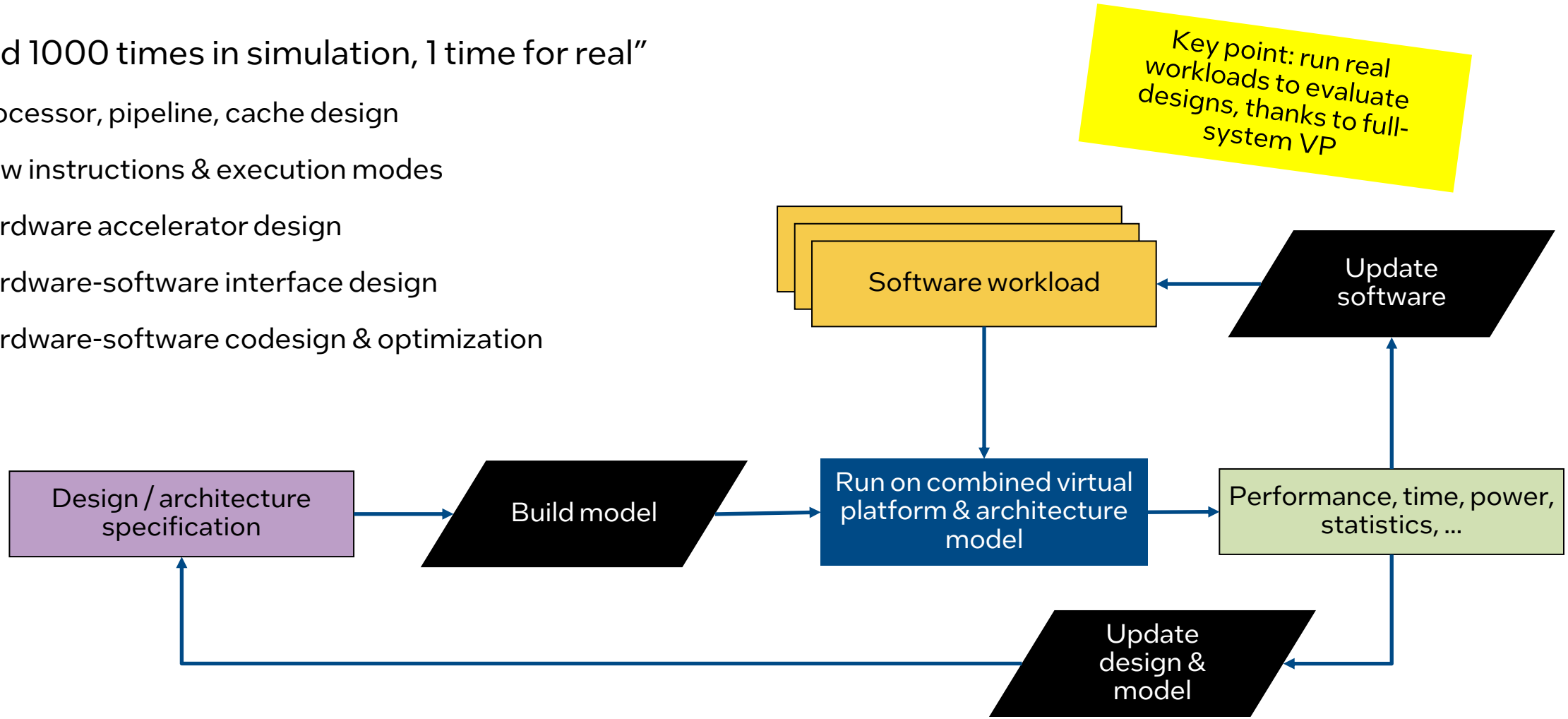


Getting the Architecture Right

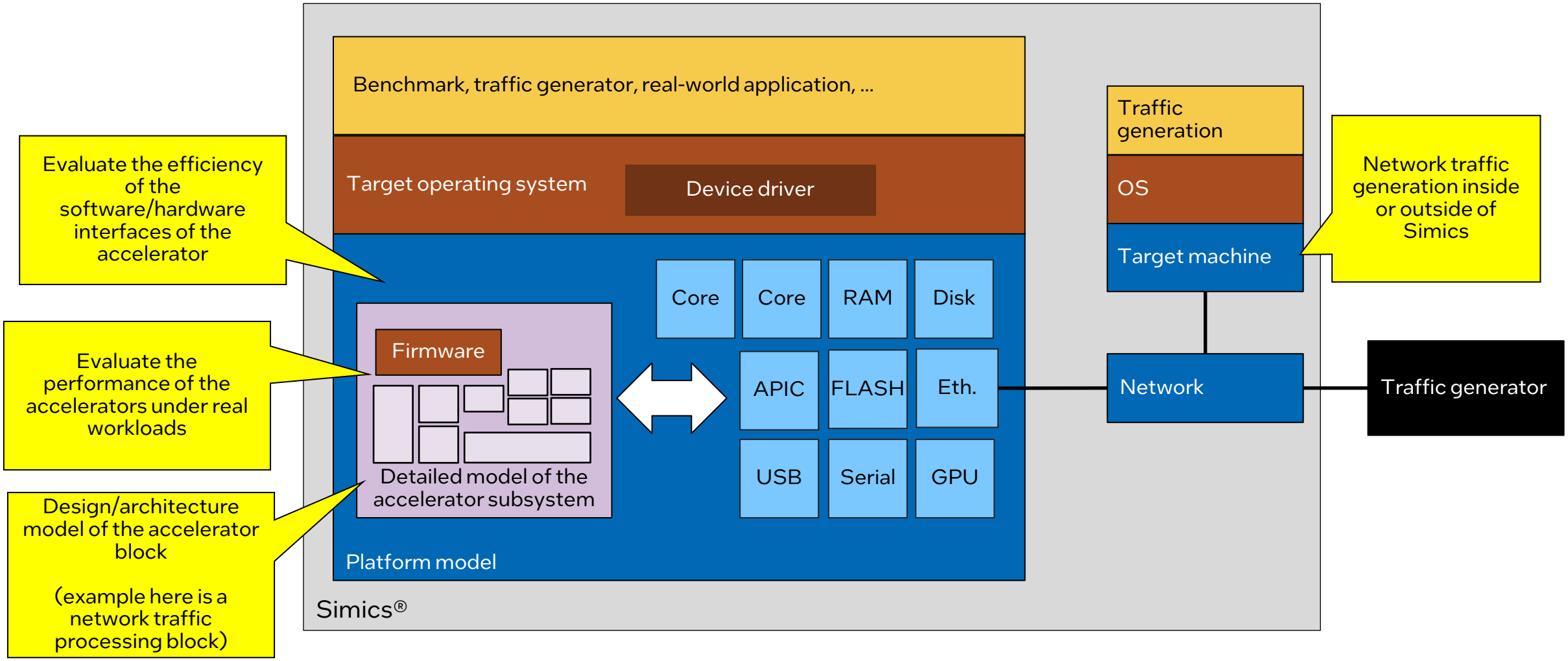
Computer Architecture (on Virtual Platform)

"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



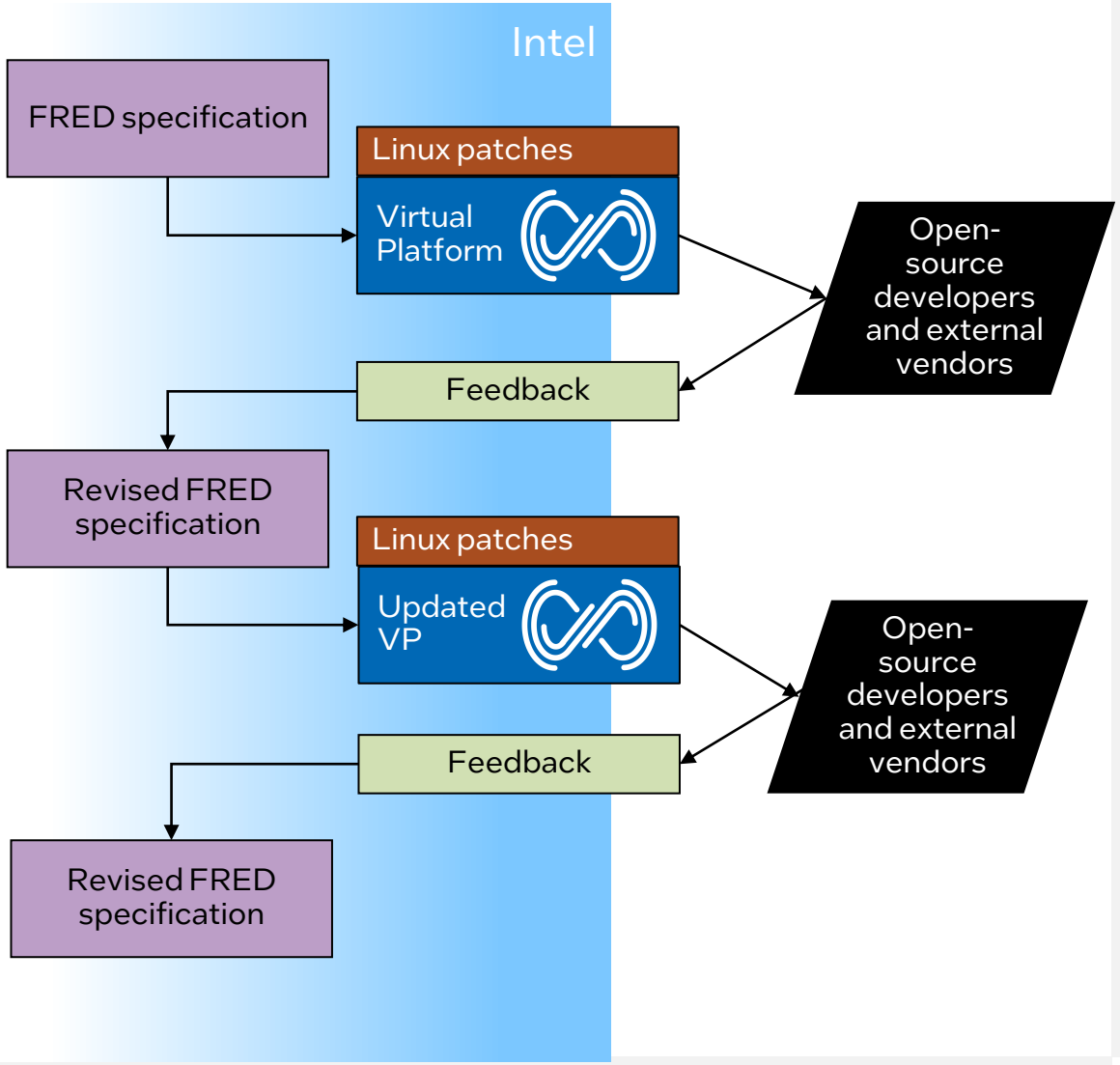
High-Level Computer Architecture

Example of architecture at the instruction-set specification level

- "Flexible Return and Event Delivery" (FRED)
- New way to handle exceptions and interrupts in the Intel Architecture

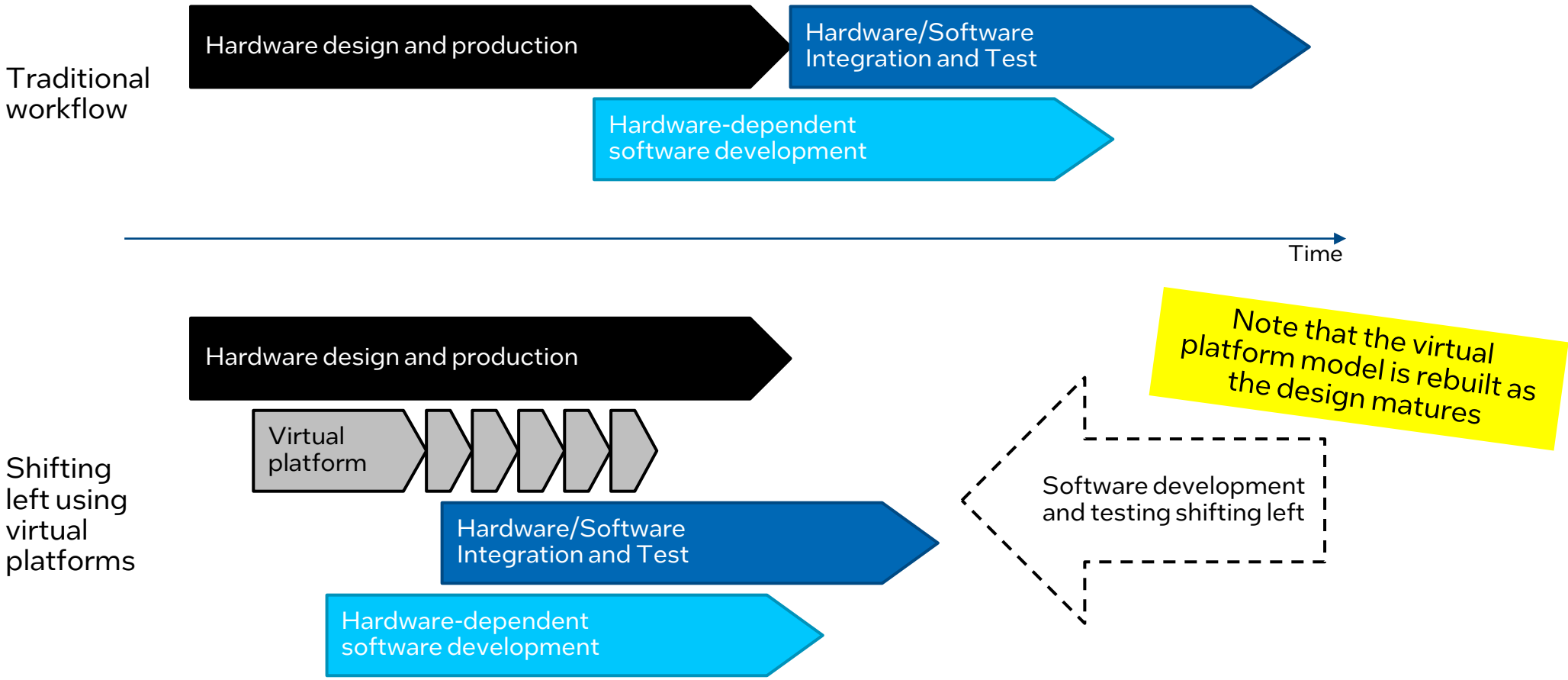
Intel® Simics® virtual platforms used as "test hardware" for external software developers

- 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



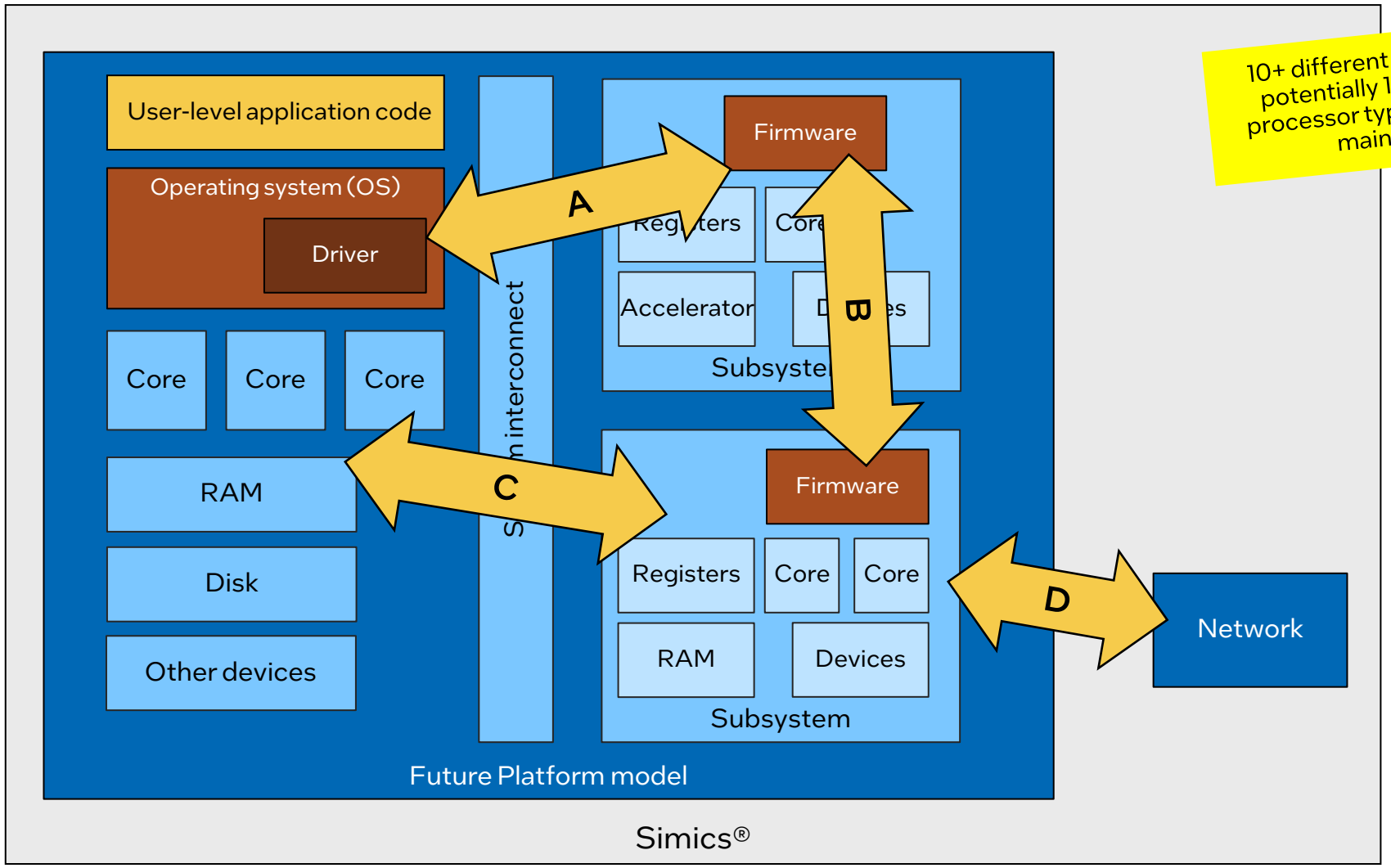
Getting Software Done Early

Shift-Left / Early Software Development



Classic use case – Earliest examples from the 1950s

Shift-Left: Going into Details with Firmware

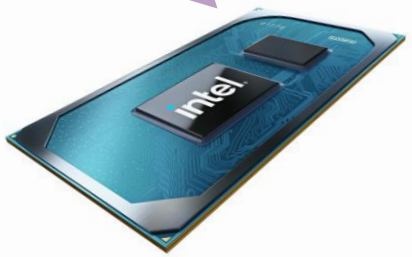


10+ different subsystems, potentially 10+ separate processor types, alongside main cores

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

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

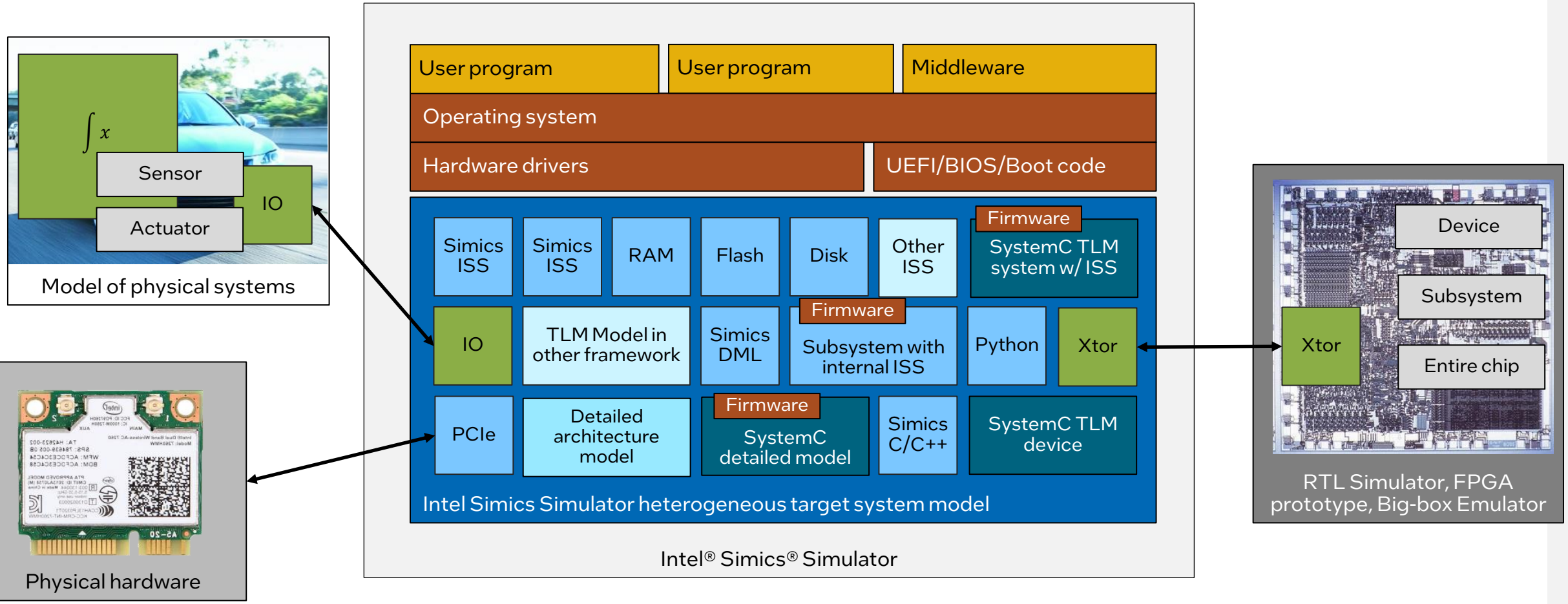
Virtual system integration

OEM Product

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

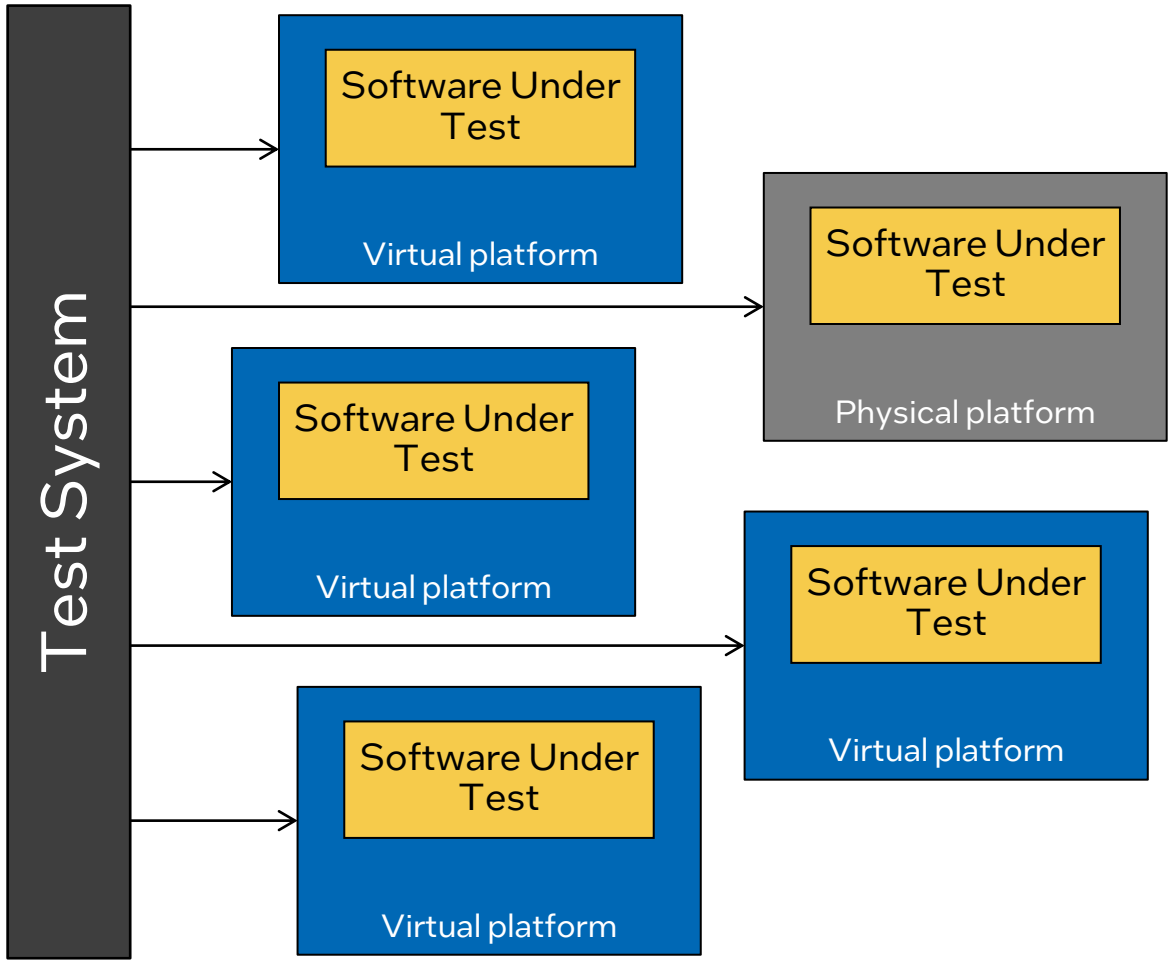
Testing Software at the System Level

Using the Simulator as an Integration Platform



https://www.intel.com/content/www/us/en/developer/articles/technical/the-more-the-merrier-building-virtual-platforms-for-integration.html

Allowing More Tests for Difficult Hardware

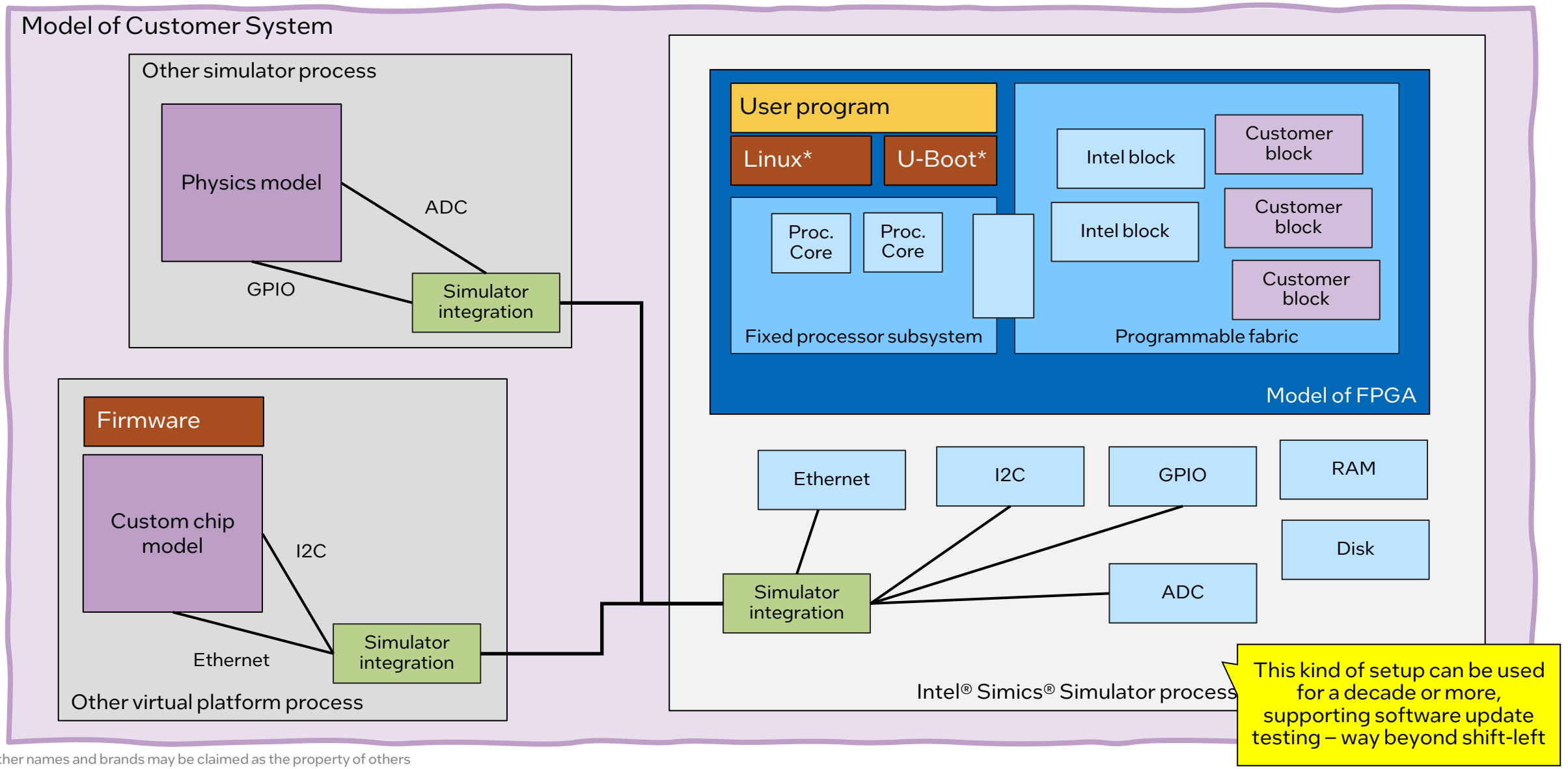


Hardware availability is often a bottleneck in embedded systems testing

Classic customer case

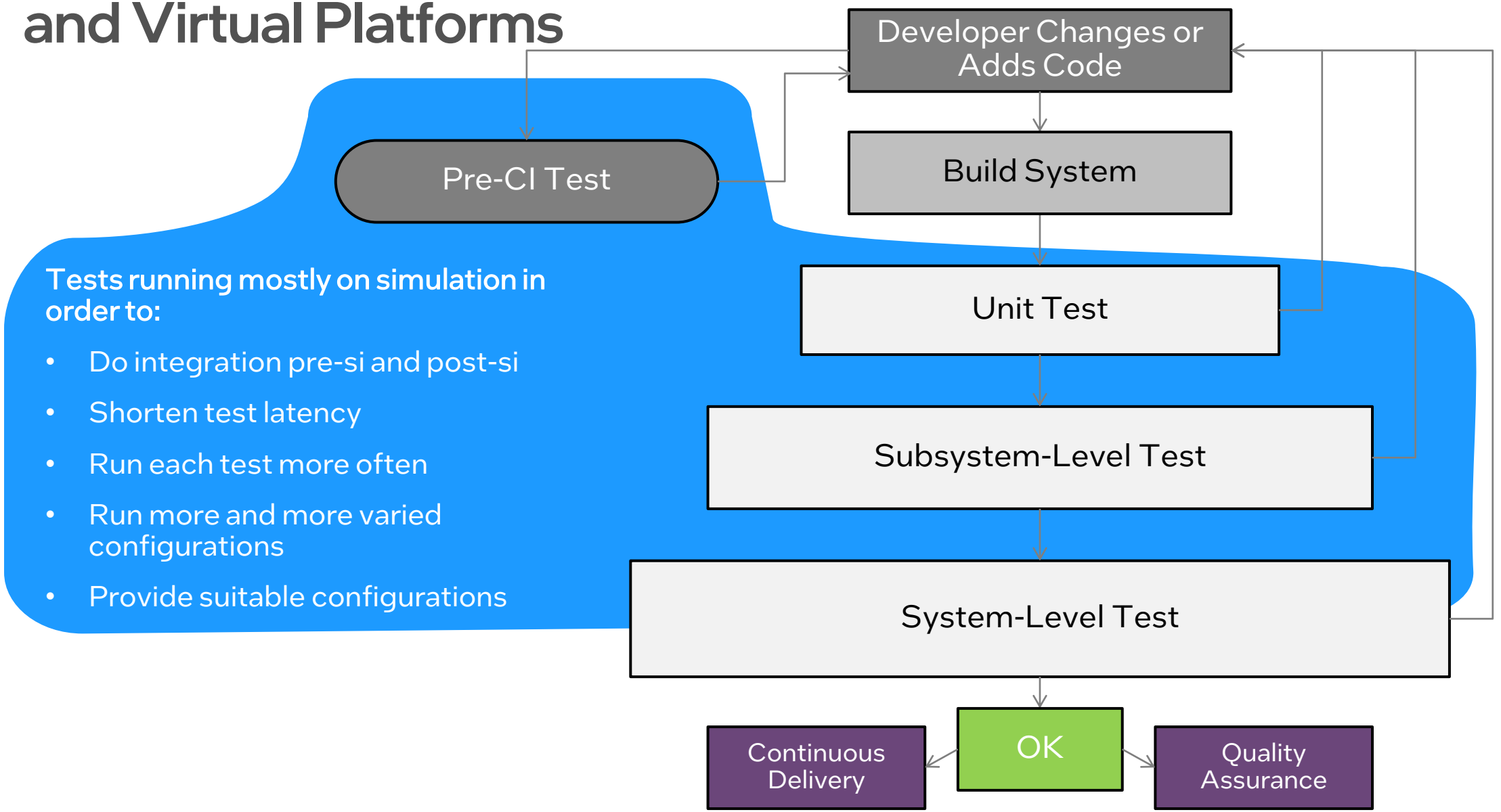
- Hardware = Integration testing every *week*
  - Bugs creep back in
  - Impossible to go Agile
- Virtual platforms provided more targets
  - Integration testing *daily*
- = Higher quality, less rework, more agile development flow

Testing with Many Models



*Other names and brands may be claimed as the property of others

Continuous Integration and Virtual Platforms



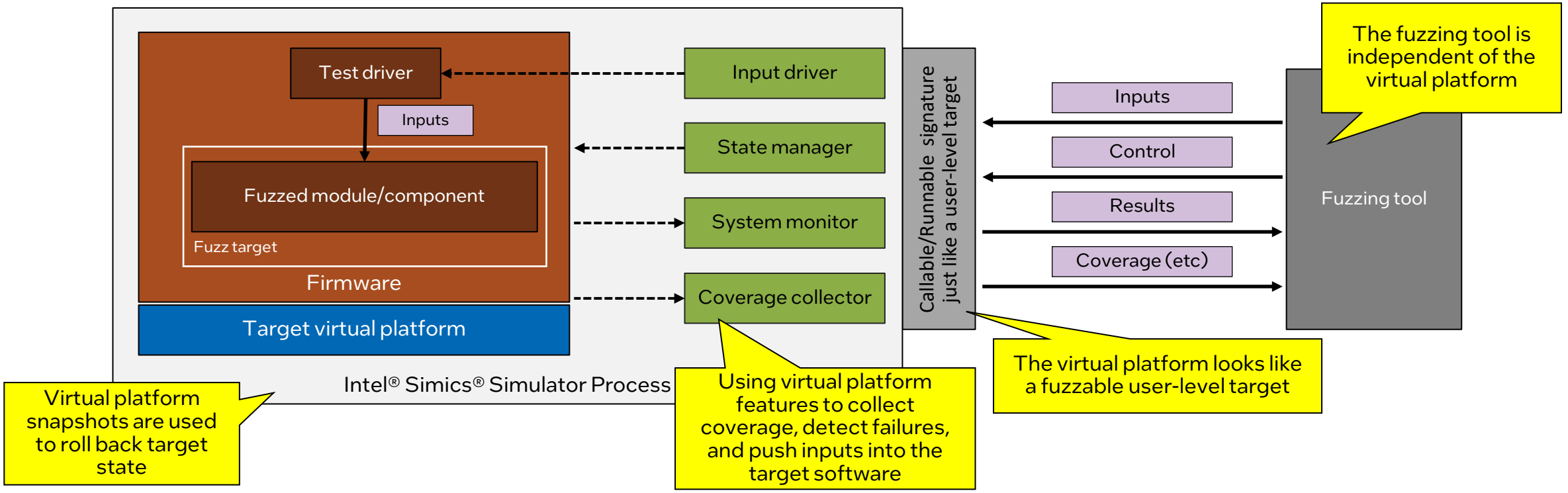
Tests running mostly on simulation in order to:

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

Fuzzing Firmware using a Virtual Platform

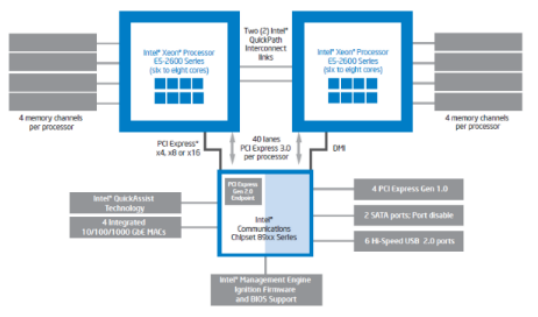
Applying standard fuzzing tools to hard-to-get-at software

- Drivers, embedded firmware, bare-metal code, bootloaders, …



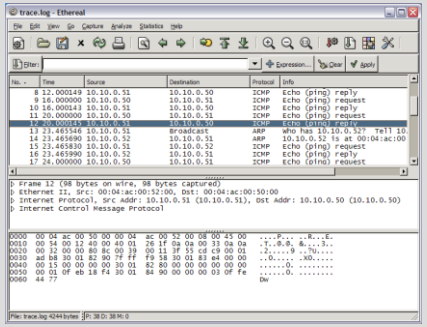
Virtual Platform Debug Features

Insight into all components

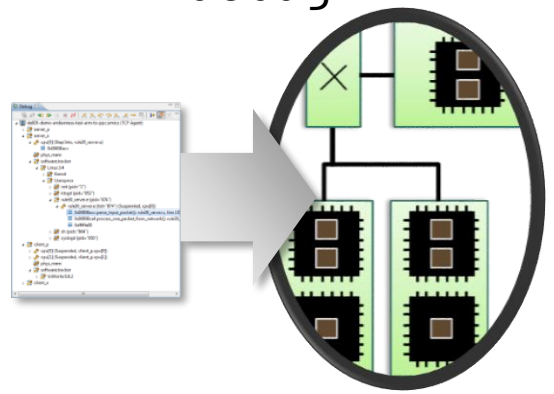


Synchronous entire-system stop

Trace anything



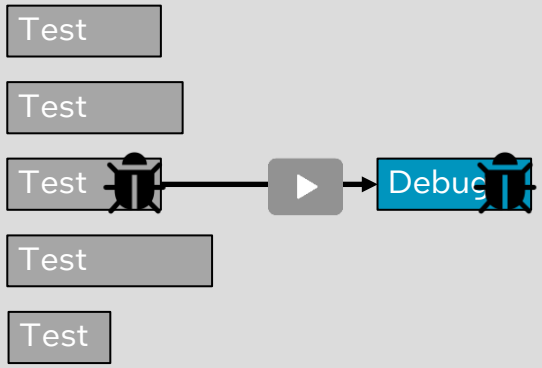
System-level symbolic debug



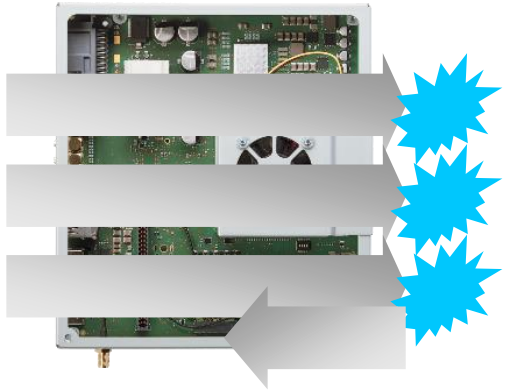
Unlimited powerful breakpoints

```
break -x 0x0000 length=0x1F00
break-io board.mb.sb.lan
break-exception int13
break-log "spec violation"
```

Record-replay debug



Repeatability & Reverse debug



Collaboration between developers

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.

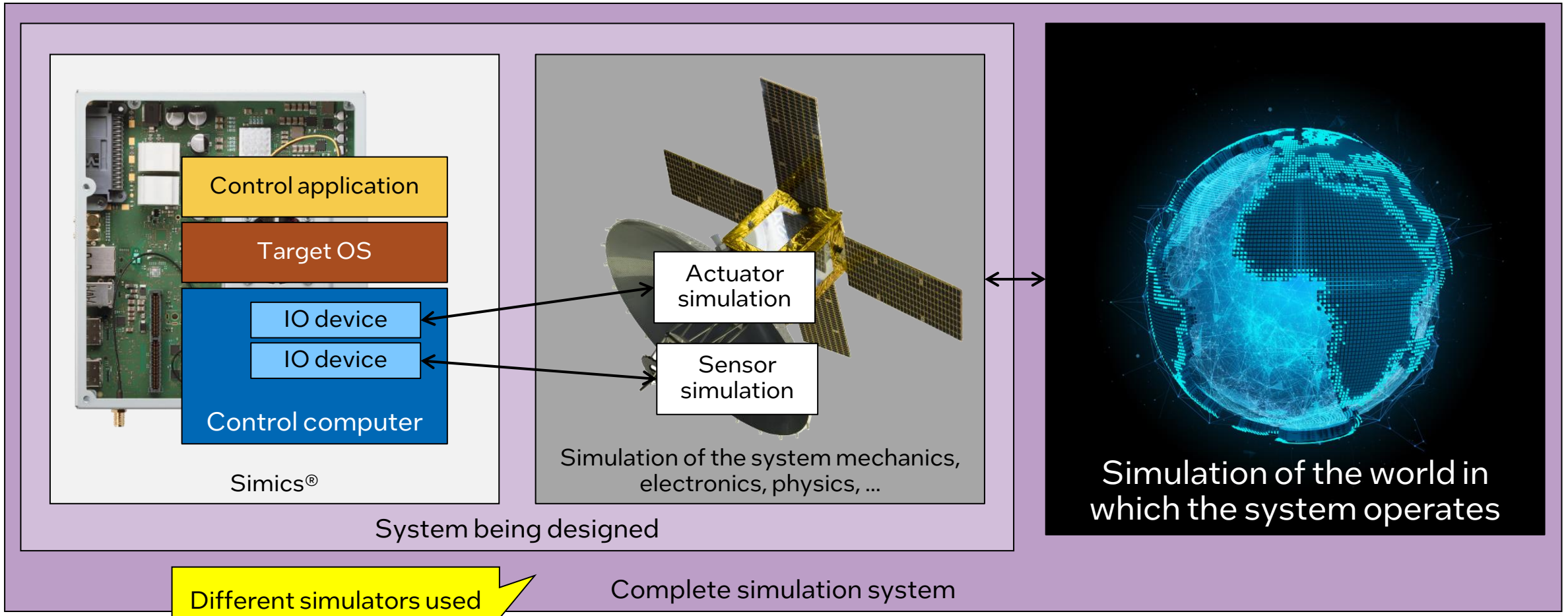
No computer system can be absolutely secure.

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.

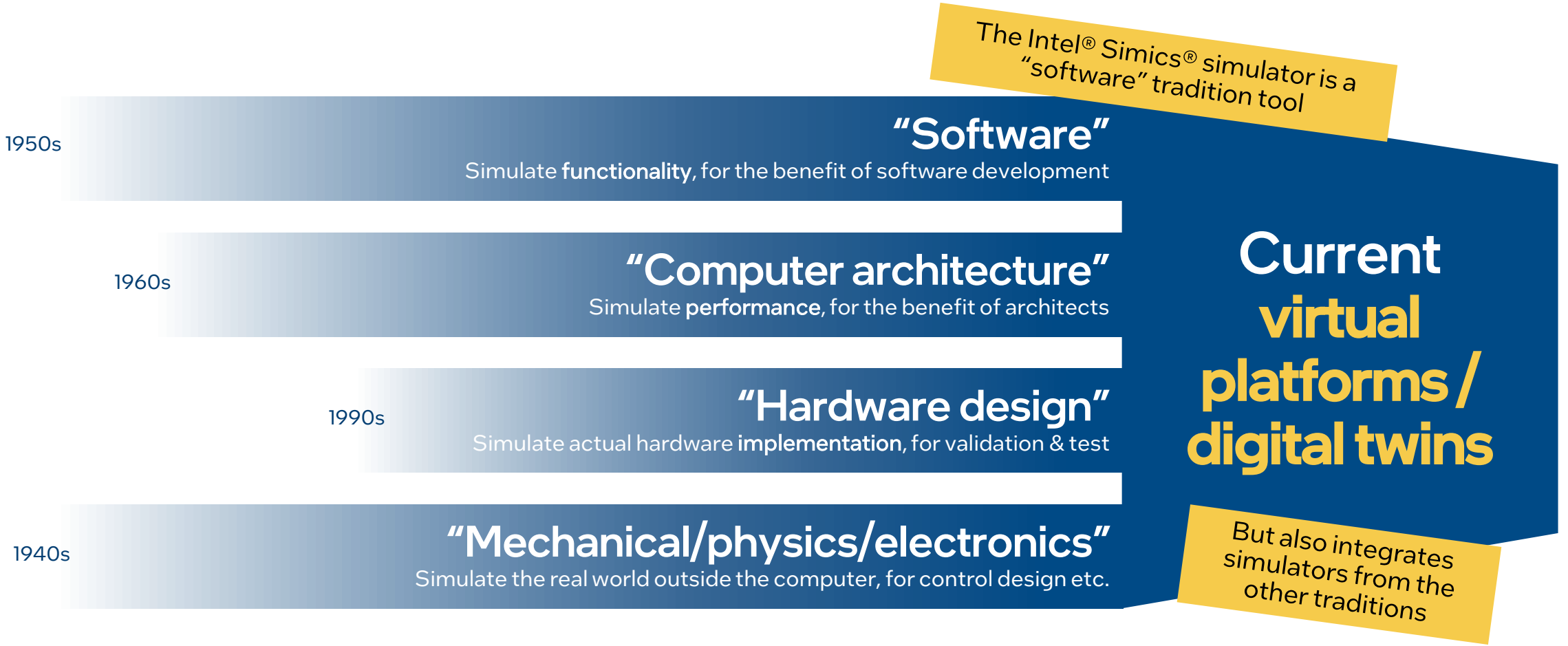
*Other names and brands may be claimed as the property of others.


intel

Integrating Environment Simulation



Virtual Platform and Simulator Traditions



*Other names and brands may be claimed as the property of others

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);
        ...
```