RAPIDO
2026

Virtual Platforms for System Architecture, Development, and Test

Dr Jakob Engblom, Cadence, engblom@cadence.com

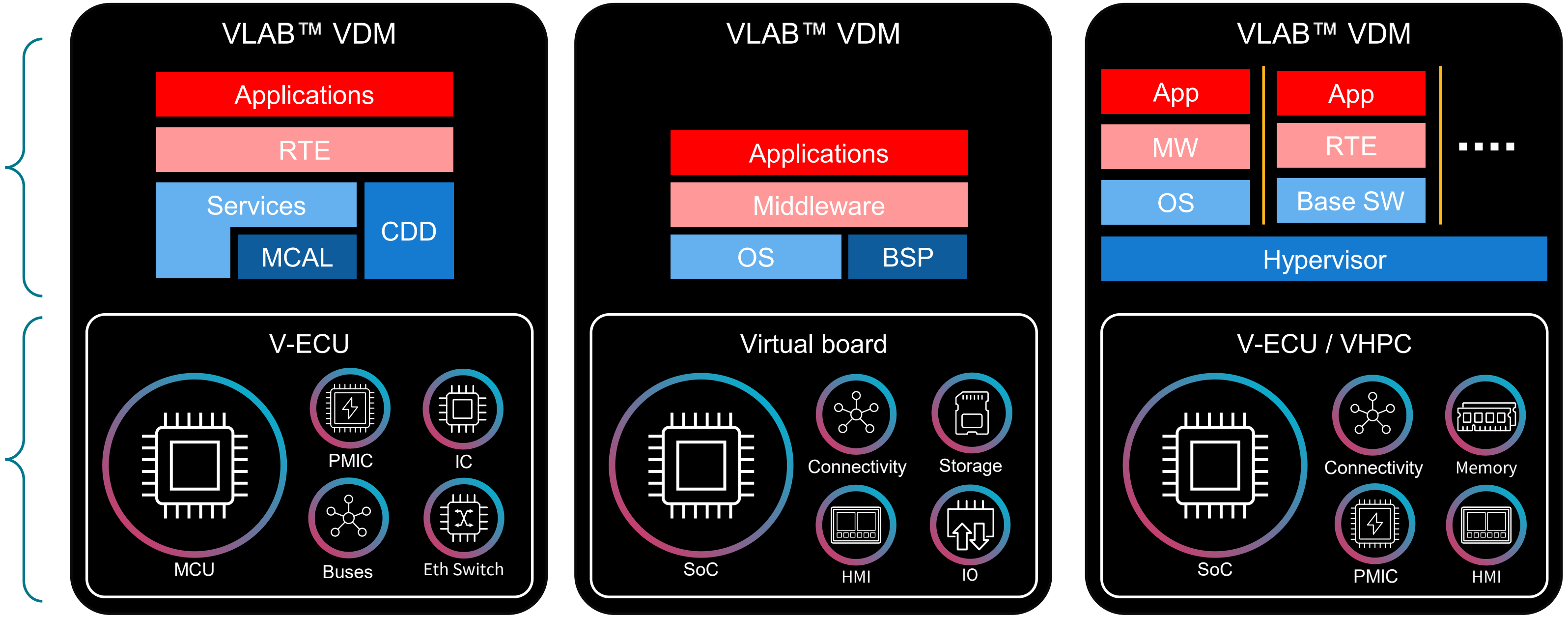
cādence®

VLAB™ - Virtualization for Software Development

Complete target-compiled software stack

Virtualization of the target hardware system – ECU, board, system level

Large library of models: cores, SoCs, buses, networks, components

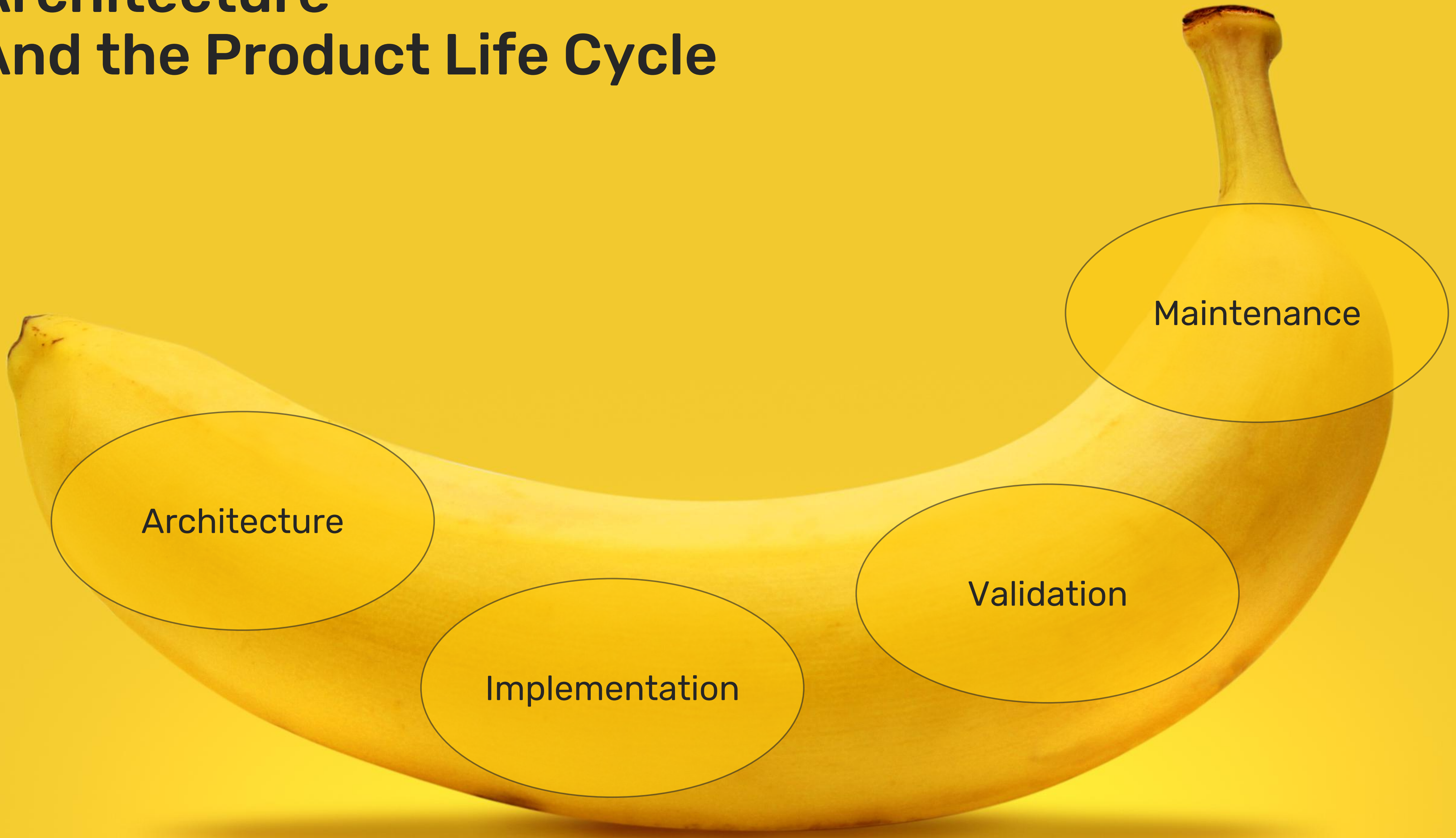


Ecosystem

- Debug using any standard debugger
- Connect to system simulators
- Embedded and automotive test tools
- Integrate 3rd party hardware models
- Script and automate in Python
- Export traces and logs for analysis

- Industry leading virtualization of embedded hardware for software developers
- **High-performance** hardware simulation (multiple GIPS/GHz)
- Multithreaded and multi-process execution for cores, IP models, and SystemC plugins
- Scalable, on-demand capacity for CD/CI/CT flows
- Access for local and global development teams
- Run on a laptop, desktop, server, or in the cloud

Architecture –
And the Product Life Cycle



What is a System?

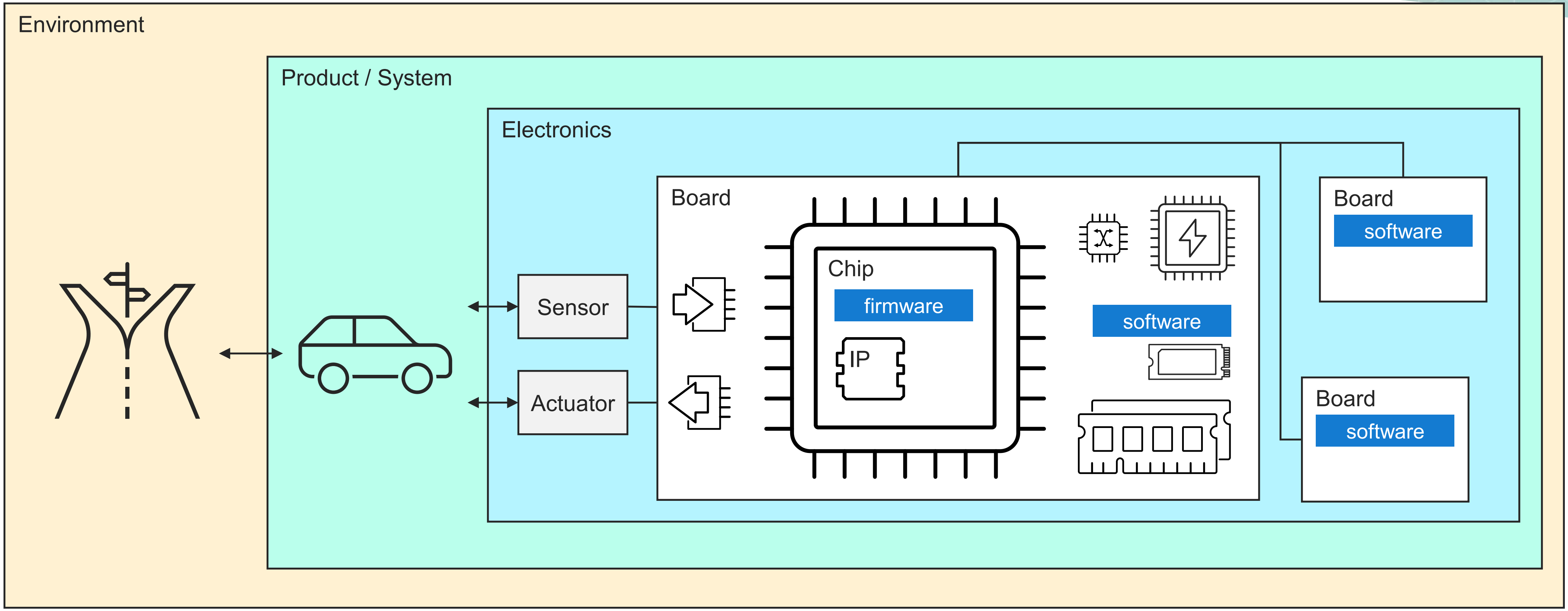
(Featuring Chips & Software)

RAPIDO 2026

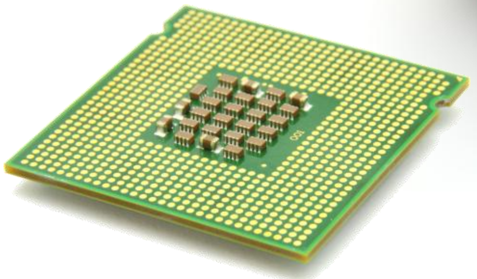
Virtual Platforms for System Architecture, Development, and Test

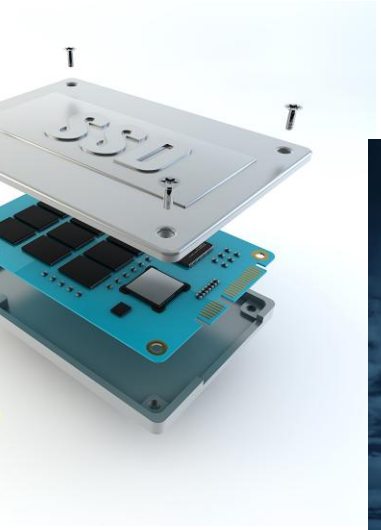
Dr Jakob Engblom, Cadence, engblom@cadence.com

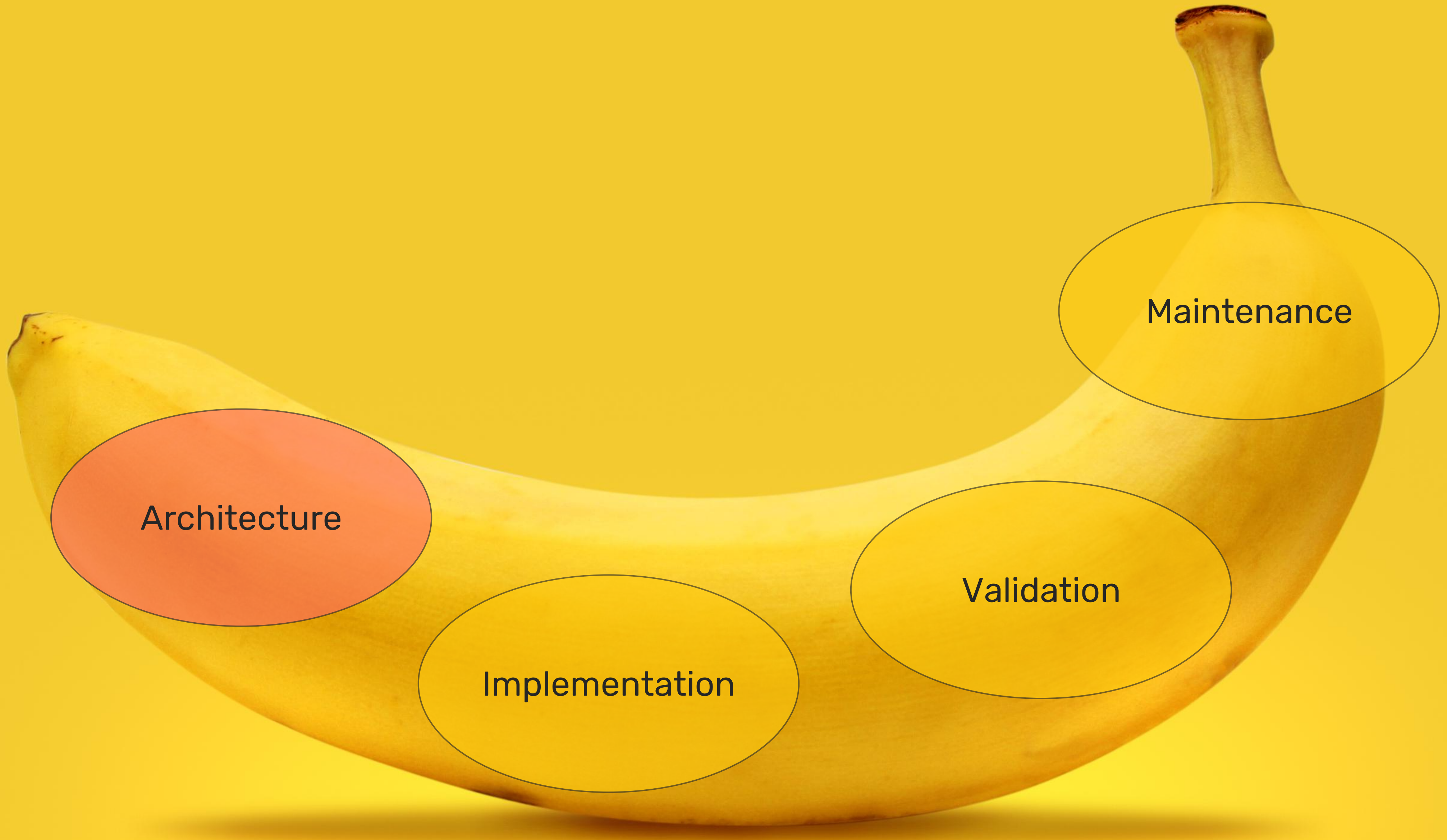
cadence



Systems Featuring Chips & Software







Architecture

Implementation

Validation

Maintenance

Architecting (Something) (With Models)

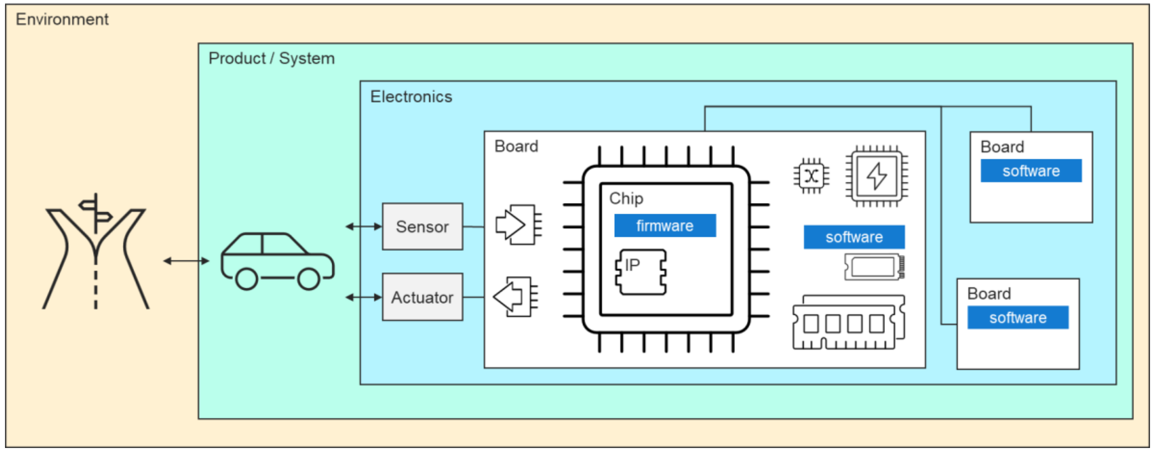
What are you architecting? → Where are the main risks and unknown? → Modeling!

What is important to model?

What is **not** important to model?

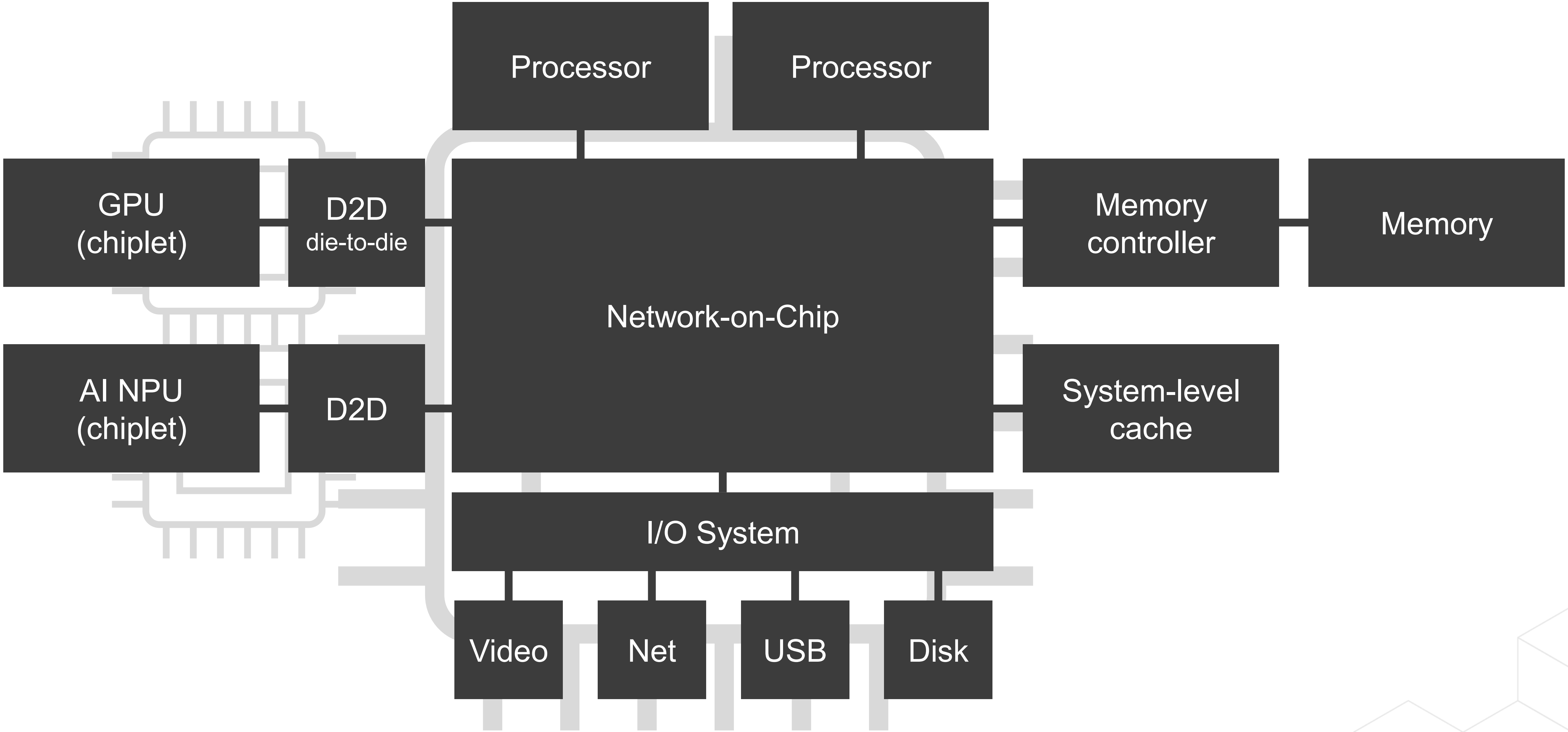
What is under your control?

What external inputs, factors, behaviors must be considered?

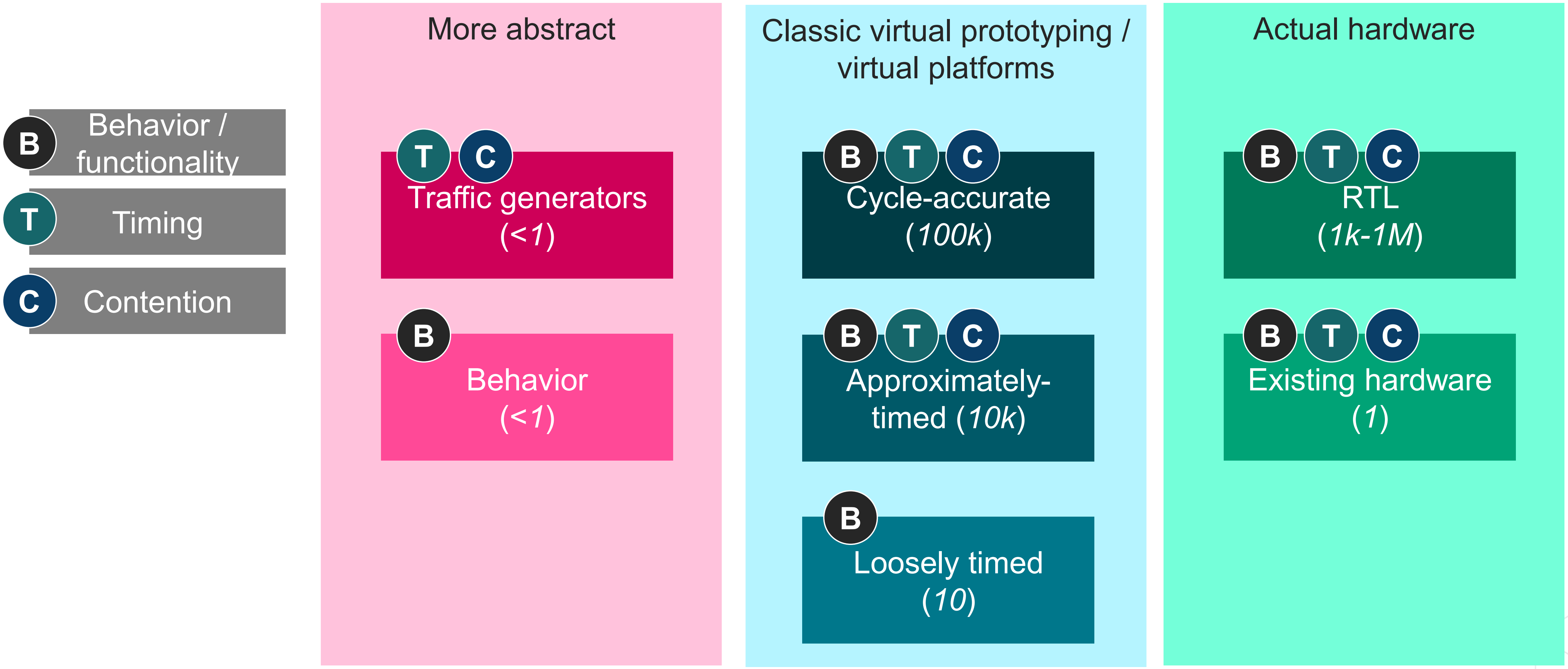


The **model** should be faster and easier to change than an actual **implementation** for modeling to make sense

Architecting a Chip (Generic SoC for Automotive, Mobile, etc.)



Modeling Hardware Units and Interconnects



B Behavior / functionality

T Timing

C Contention

More abstract

T C Traffic generators (*<1*)

B Behavior (*<1*)

Classic virtual prototyping / virtual platforms

B T C Cycle-accurate (*100k*)

B T C Approximately-timed (*10k*)

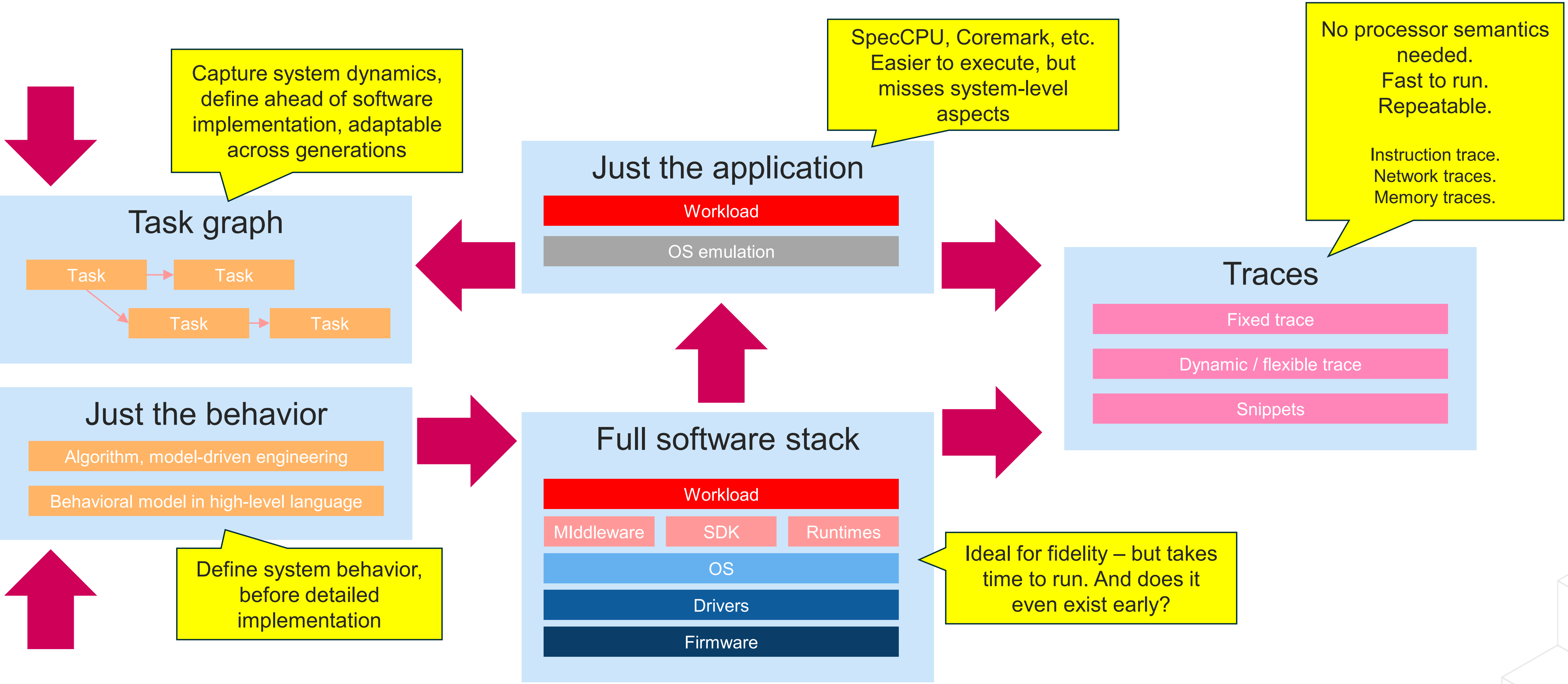
B Loosely timed (*10*)

Actual hardware

B T C RTL (*1k-1M*)

B T C Existing hardware (*1*)

Modeling Software



Capture system dynamics, define ahead of software implementation, adaptable across generations

Task graph

Task

Task

Task

Task

Just the behavior

Algorithm, model-driven engineering

Behavioral model in high-level language

Define system behavior, before detailed implementation

SpecCPU, Coremark, etc. Easier to execute, but misses system-level aspects

Just the application

Workload

OS emulation

Full software stack

Workload

MIddleware

SDK

Runtimes

OS

Drivers

Firmware

Ideal for fidelity – but takes time to run. And does it even exist early?

No processor semantics needed.
Fast to run.
Repeatable.

Instruction trace.
Network traces.
Memory traces.

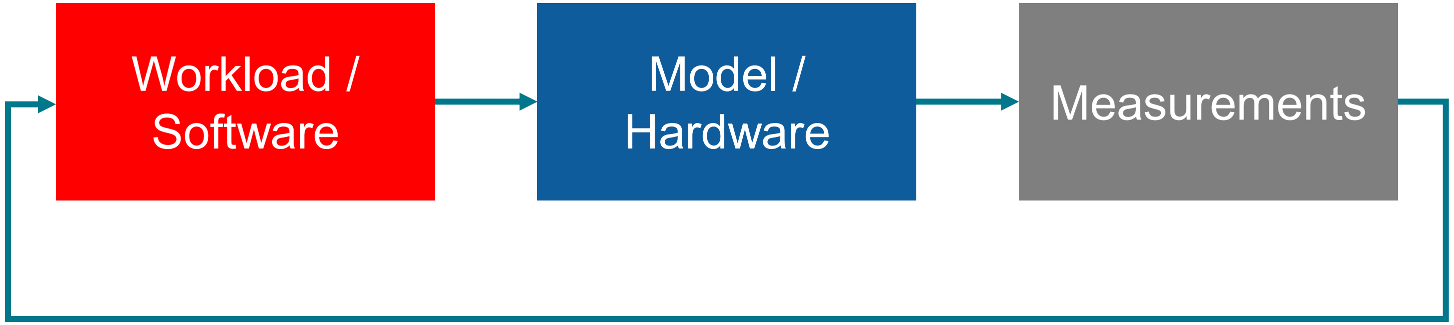
Traces

Fixed trace

Dynamic / flexible trace

Snippets

Attention! Feedback Loops!



Examples

Power management

- Application *is* a control loop
- Throttle on overheating
- Speed up when cool
- Speed up when plugged in
- …

Games

- Measure achieved frame rate
- Adjust quality settings to reach frame rate
- Measure input latencies, adjust handling

Streaming & video

- Adjust video quality based on network bandwidth
- Prioritize traffic

Interrupt timing

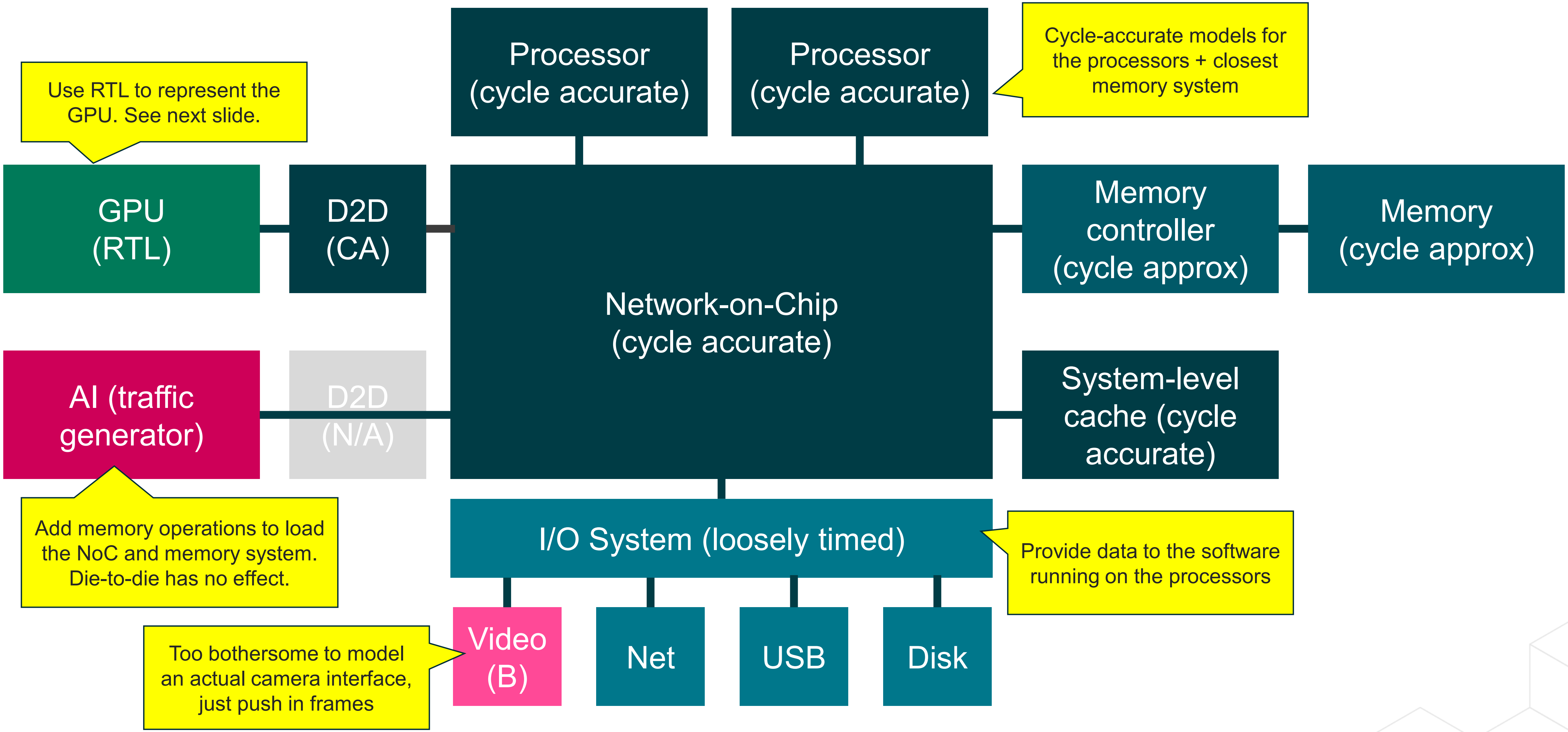
- Change to software timing
- Changes when OS schedules tasks
- Divergent behavior

JIT compilers

- Measures execution time using performance counters
- Decides what to compile dynamically

Fixed traces (often) fail to capture feedback loops in the system behavior and can be **directly misleading**

Example: For a Processor Core Architect



Processor (cycle accurate)
Processor (cycle accurate)
GPU (RTL)
D2D (CA)
Network-on-Chip (cycle accurate)
Memory controller (cycle approx)
Memory (cycle approx)
AI (traffic generator)
D2D (N/A)
System-level cache (cycle accurate)
I/O System (loosely timed)
Video (B)
Net
USB
Disk

Use RTL to represent the GPU. See next slide.

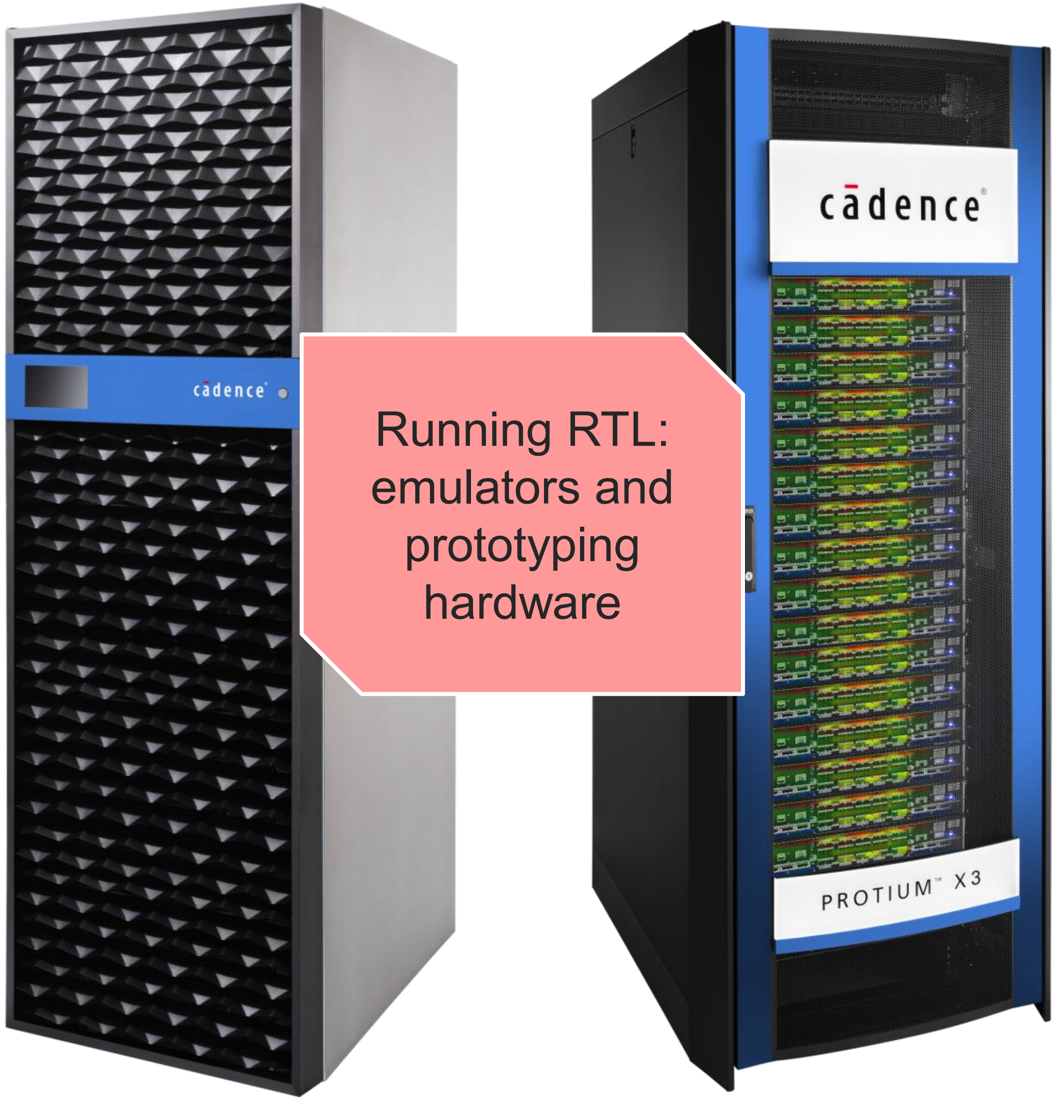
Cycle-accurate models for the processors + closest memory system

Add memory operations to load the NoC and memory system. Die-to-die has no effect.

Provide data to the software running on the processors

Too bothersome to model an actual camera interface, just push in frames

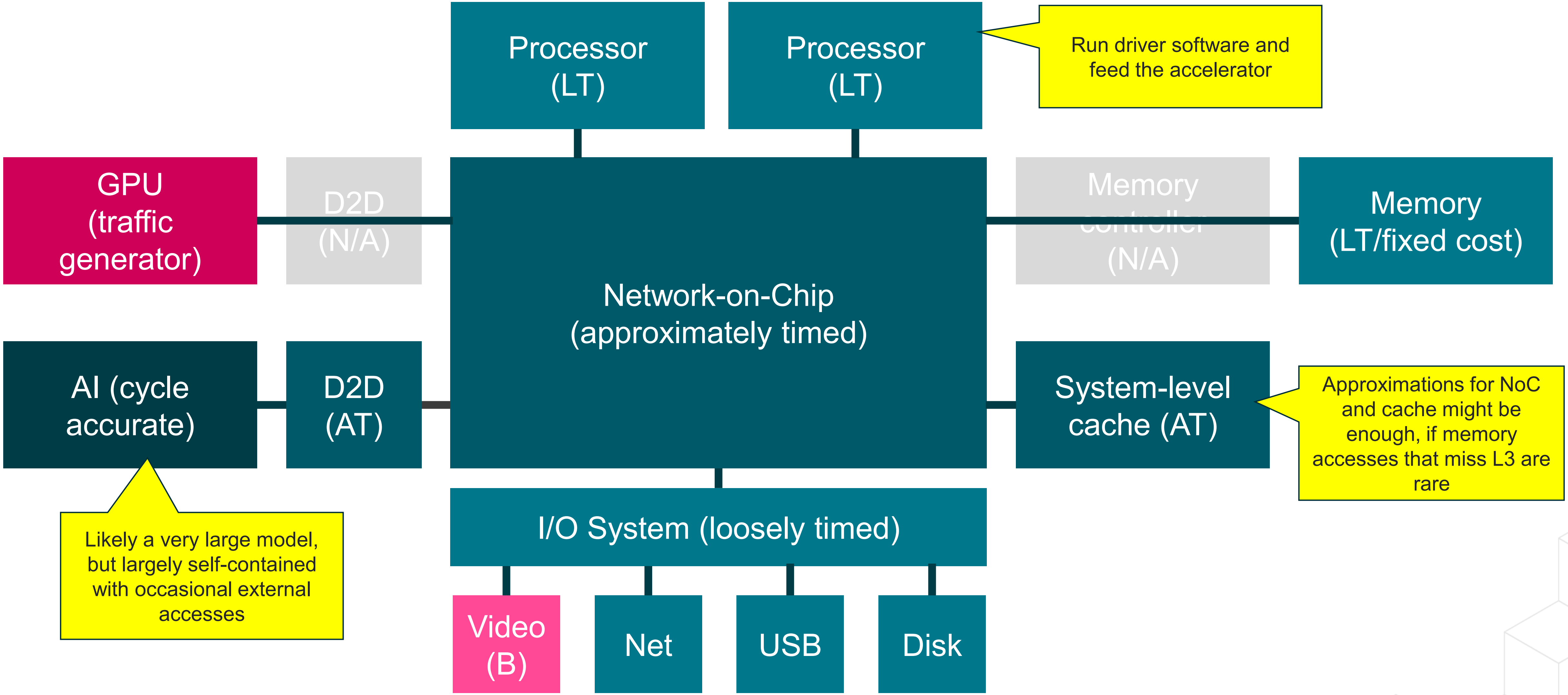
Using the RTL Implementation as Model



Running RTL: emulators and prototyping hardware

- Architecting a **chip**
- Buying **components**
  - = the components are done
- Architecture = configuration & interactions
  - How many processor cores?
  - How many GPU slices?
  - Clock speed?
  - Power budget?
  - Bandwidth of the interconnect?
  - Structure of the interconnect?
- Running pieces as RTL makes sense
  - “Fast” with emulators and prototypes
  - Modeling someone else’s design is *hard*

Example: For an AI Accelerator Architect

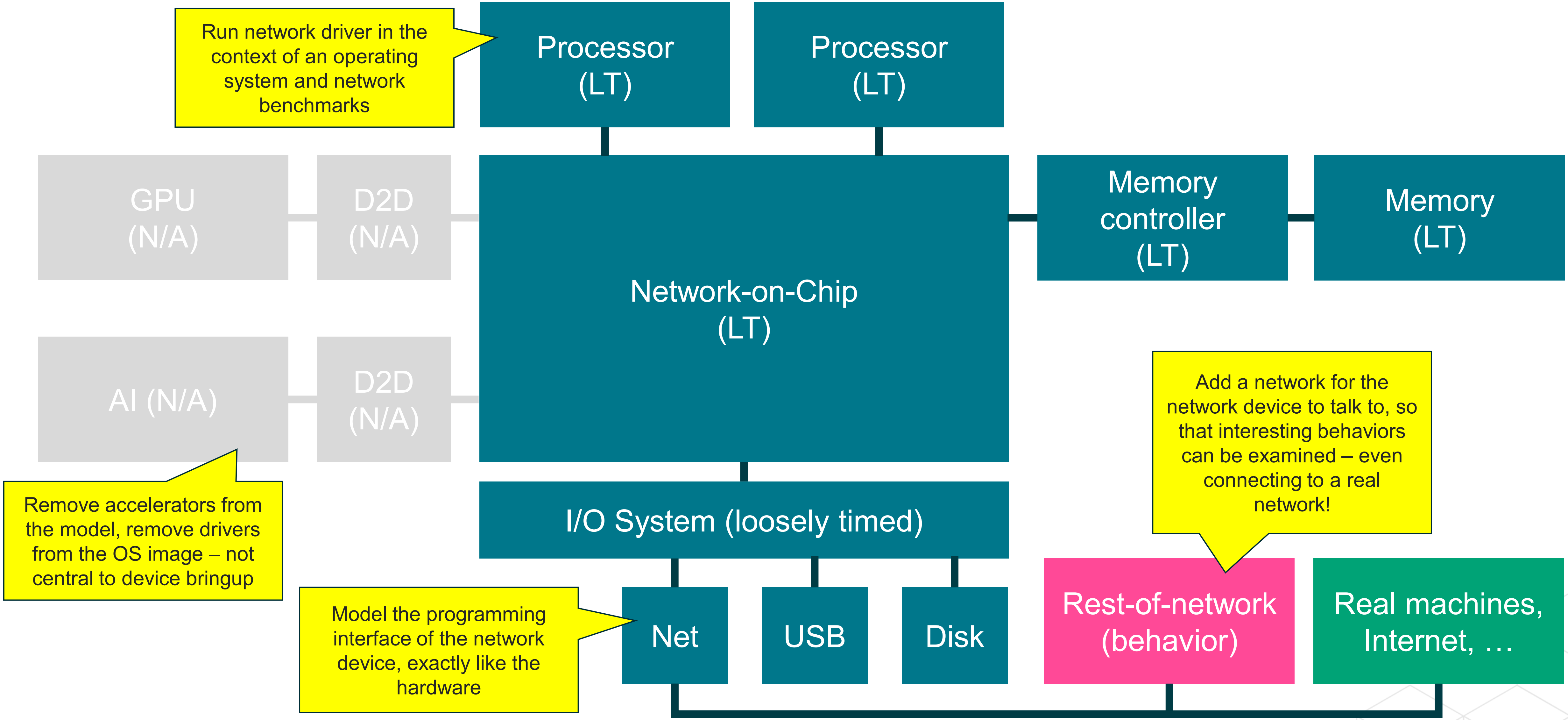


Run driver software and feed the accelerator

Approximations for NoC and cache might be enough, if memory accesses that miss L3 are rare

Likely a very large model, but largely self-contained with occasional external accesses

Example: For a Device Interface Designer / Architect



Run network driver in the context of an operating system and network benchmarks

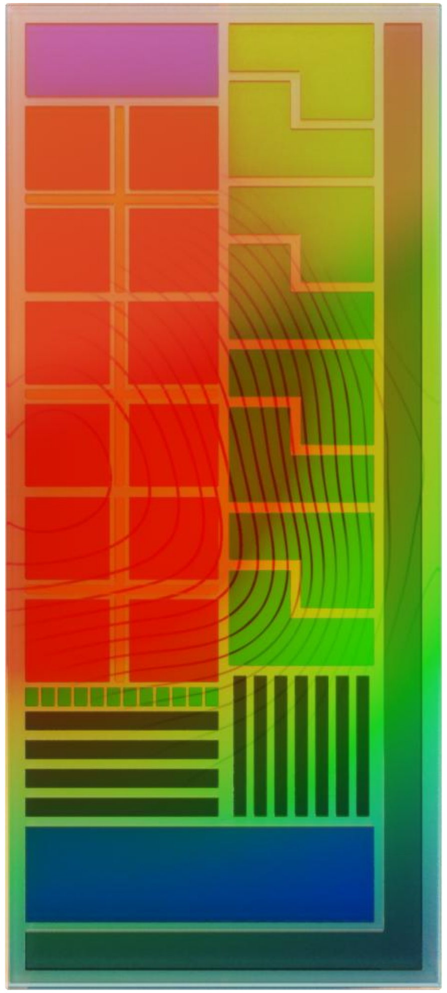
Remove accelerators from the model, remove drivers from the OS image – not central to device bringup

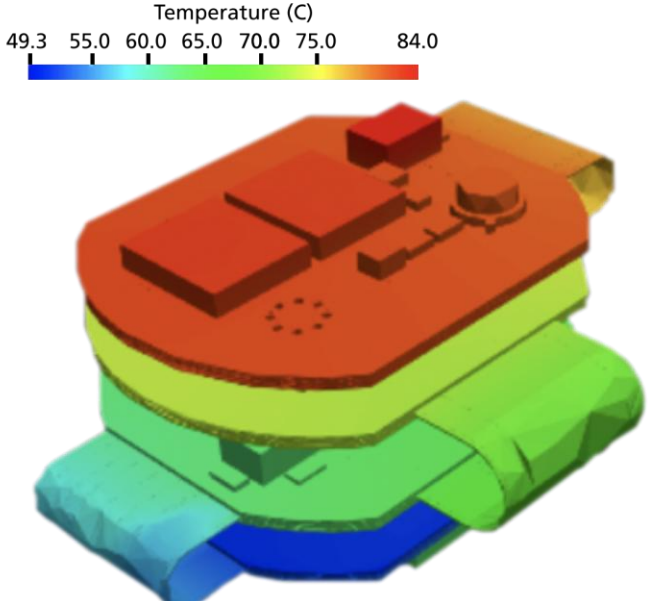
Model the programming interface of the network device, exactly like the hardware

Add a network for the network device to talk to, so that interesting behaviors can be examined – even connecting to a real network!

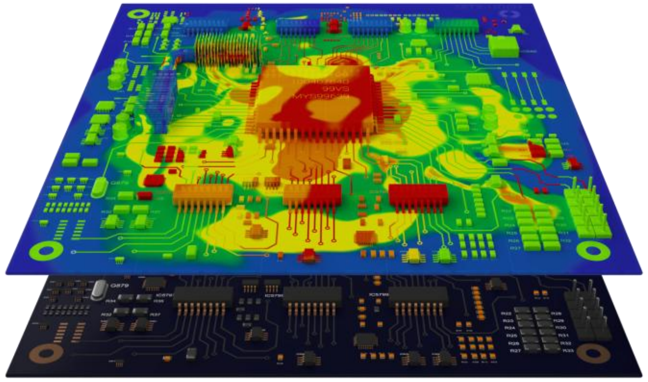
Modeling Power, Energy, Thermals

- Power is “easy”
  - Power state, clock speed, voltage, …
  - Essentially, functional aspects that are set by software and firmware
- Energy is trickier
  - Energy = Power * Time
  - Requires a precise timing model
- Thermals are even trickier
  - Requires a good energy model
  - Energy to heat conversion model
  - Geography of the chip
  - Thermal properties of materials
  - Heat propagation and cooling
  - = complex physics simulation



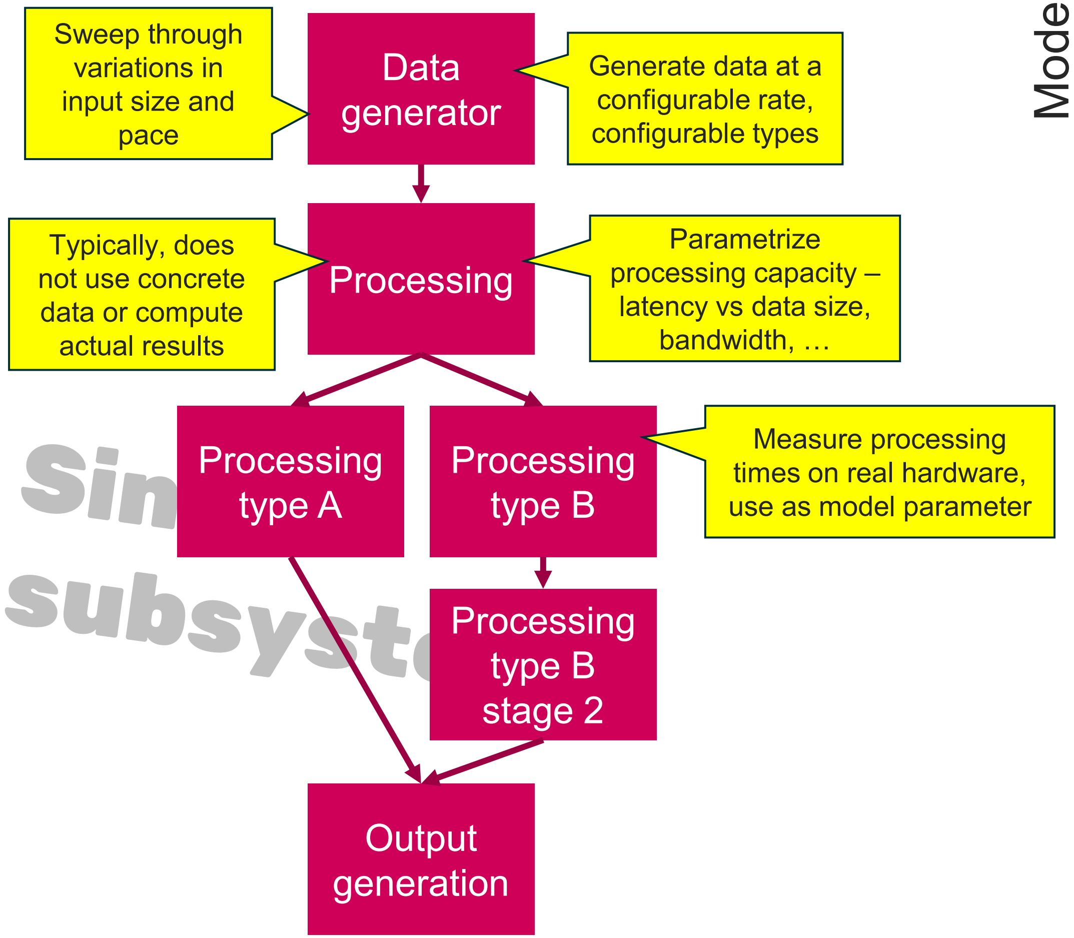


Screenshot from Cadence Celsius



Analysis by Cadence Celsius

More Abstract Models – “Boxes and Lines” – Examples



Data generator

Processing

Processing type A

Processing type B

Processing type B stage 2

Output generation

Sweep through variations in input size and pace

Generate data at a configurable rate, configurable types

Typically, does not use concrete data or compute actual results

Parametrize processing capacity – latency vs data size, bandwidth, …

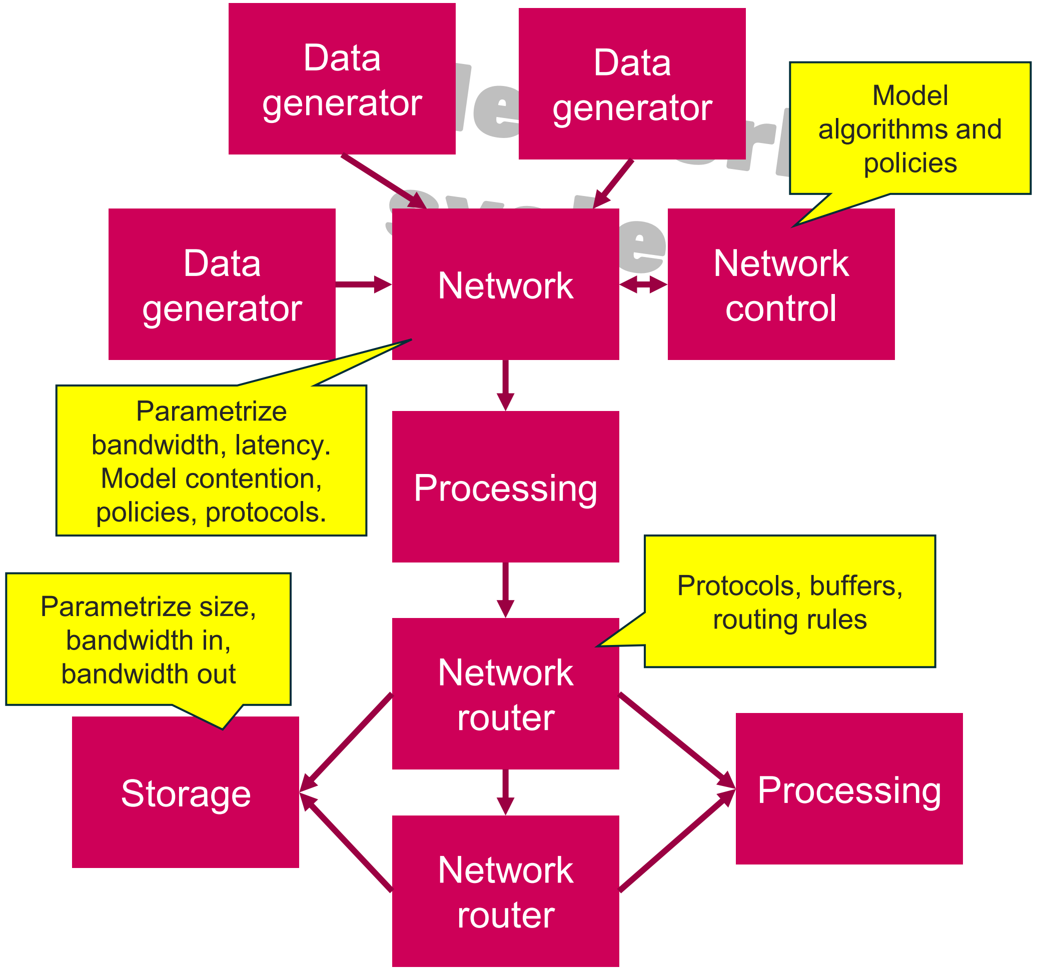
Measure processing times on real hardware, use as model parameter

Model

Unit behavior (“box”)

Connection & network behavior (“box”)

Unit connections (“lines”)



Data generator

Data generator

Model algorithms and policies

Data generator

Network

Network control

Parametrize bandwidth, latency. Model contention, policies, protocols.

Processing

Protocols, buffers, routing rules

Parametrize size, bandwidth in, bandwidth out

Network router

Storage

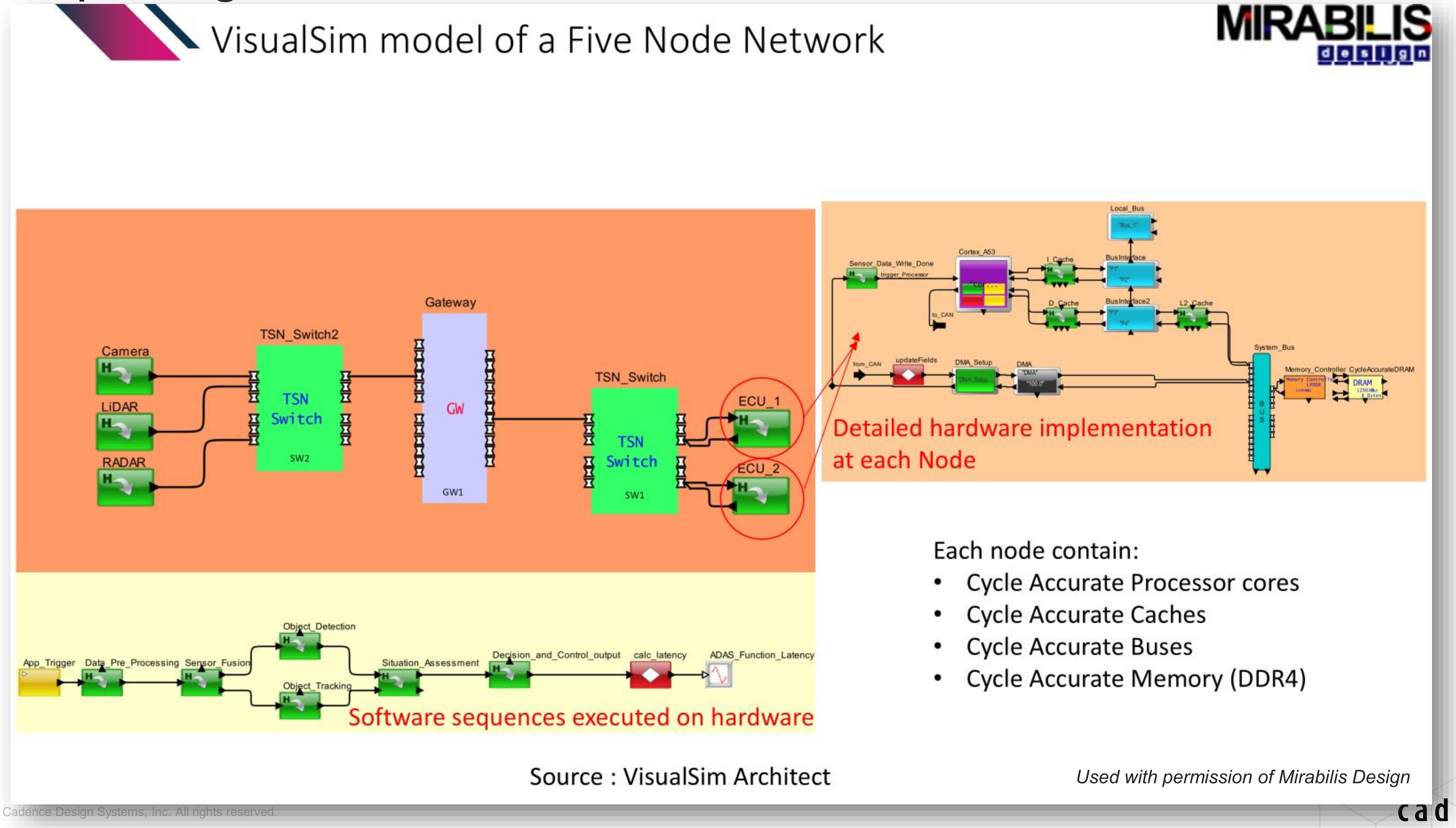
Processing

Network router

Abstract models can give surprisingly accurate results – turn Excel sheet guesses into executable model

Very suitable for complete system models beyond a single IP or chip

Example Higher-Level Model



VisualSim model of a Five Node Network

Detailed hardware implementation at each Node

Software sequences executed on hardware

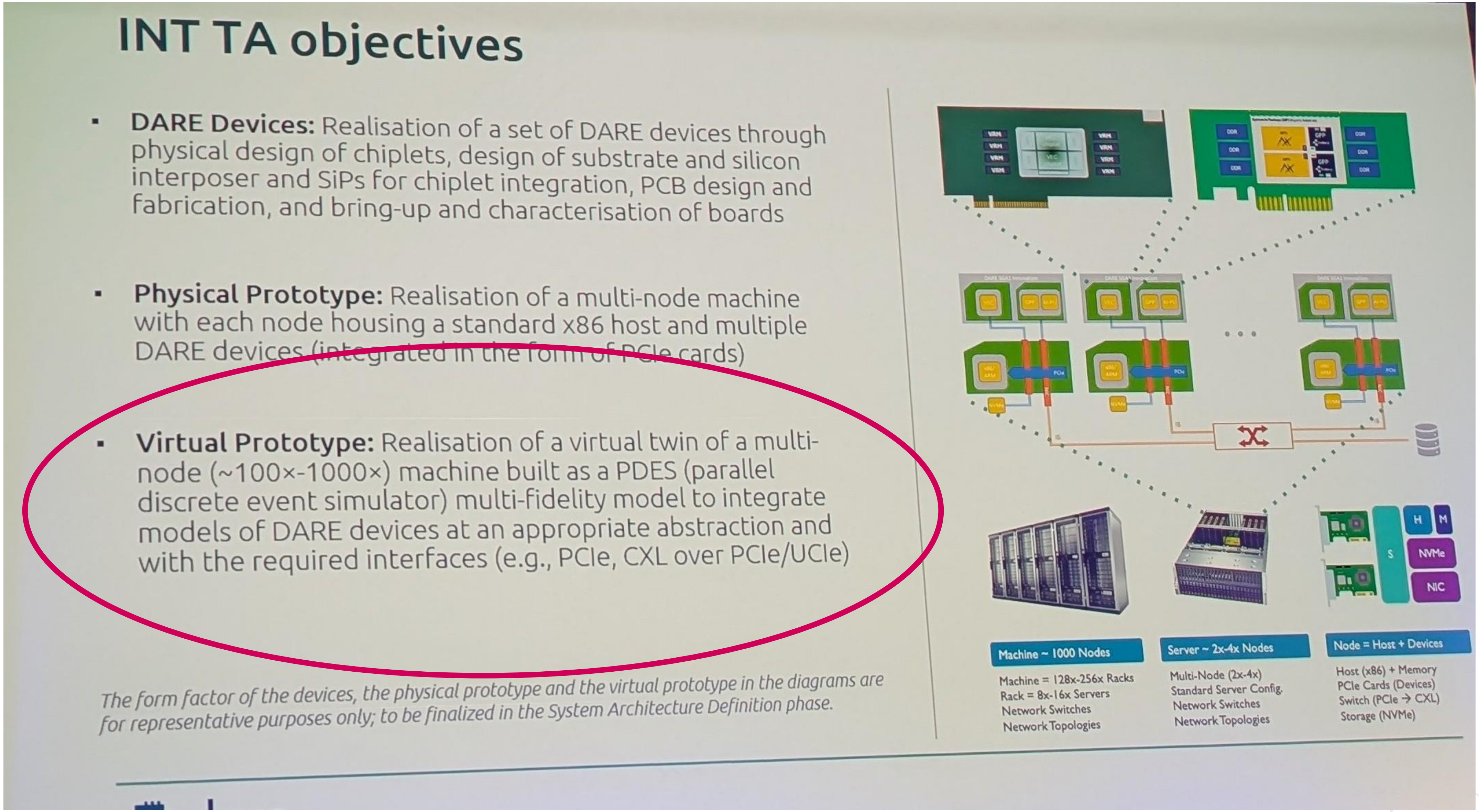
Each node contain:

- Cycle Accurate Processor cores
- Cycle Accurate Caches
- Cycle Accurate Buses
- Cycle Accurate Memory (DDR4)

Source : VisualSim Architect

Used with permission of Mirabilis Design

Example: From DARE Project (HiPEAC RISC-V Workshop)


H
M
S
NVMe
NIC

INT TA objectives

- **DARE Devices:** Realisation of a set of DARE devices through physical design of chiplets, design of substrate and silicon interposer and SiPs for chiplet integration, PCB design and fabrication, and bring-up and characterisation of boards
- **Physical Prototype:** Realisation of a multi-node machine with each node housing a standard x86 host and multiple DARE devices (integrated in the form of PCIe cards)
- **Virtual Prototype:** Realisation of a virtual twin of a multi-node (~100×-1000×) machine built as a PDES (parallel discrete event simulator) multi-fidelity model to integrate models of DARE devices at an appropriate abstraction and with the required interfaces (e.g., PCIe, CXL over PCIe/UCIe)

Machine ~ 1000 Nodes

Machine = 128x-256x Racks
Rack = 8x-16x Servers
Network Switches
Network Topologies

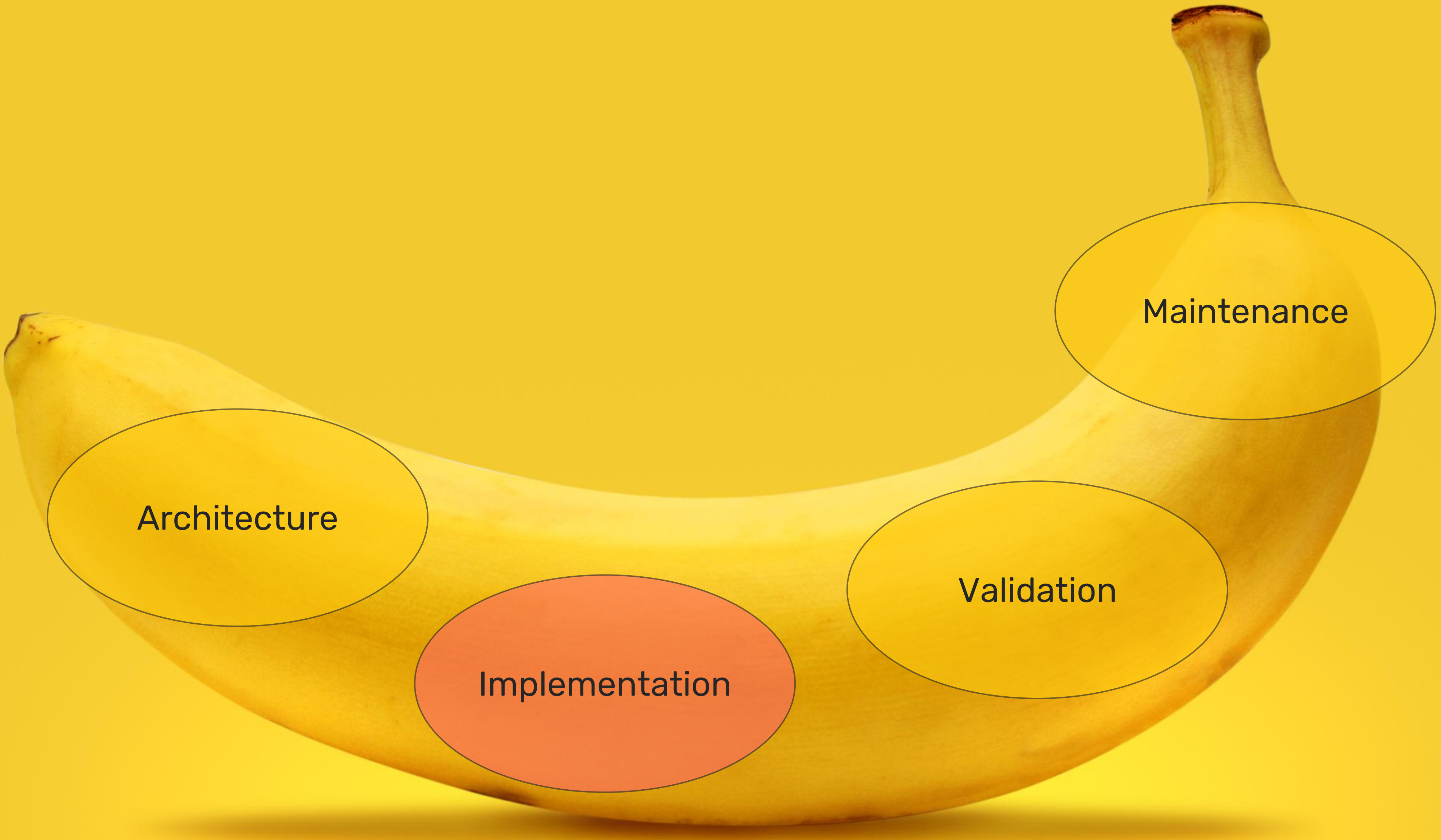
Server ~ 2x-4x Nodes

Multi-Node (2x-4x)
Standard Server Config.
Network Switches
Network Topologies

Node = Host + Devices

Host (x86) + Memory
PCIe Cards (Devices)
Switch (PCIe → CXL)
Storage (NVMe)

The form factor of the devices, the physical prototype and the virtual prototype in the diagrams are for representative purposes only; to be finalized in the System Architecture Definition phase.



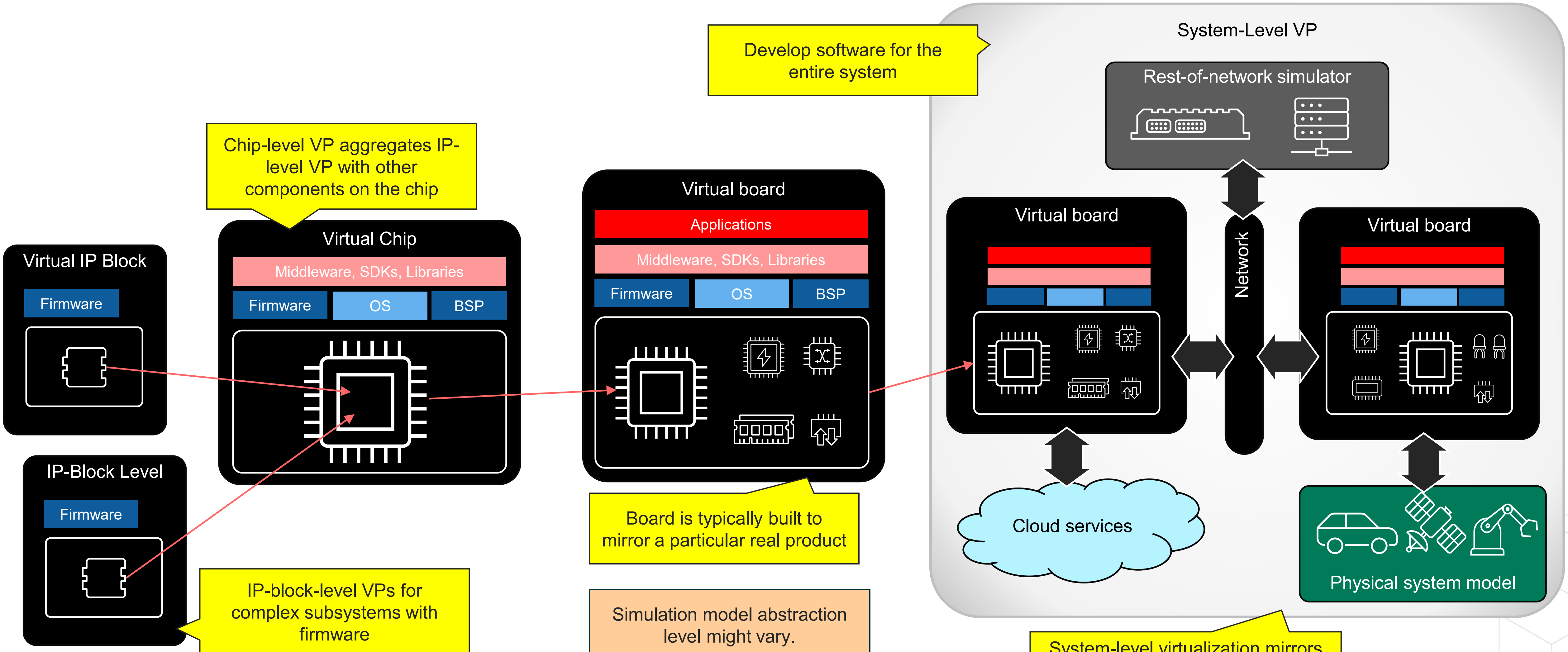
Architecture

Implementation

Validation

Maintenance

Virtual Platforms – From IP to System, Developing Software



Develop software for the entire system

Chip-level VP aggregates IP-level VP with other components on the chip

IP-block-level VPs for complex subsystems with firmware

Board is typically built to mirror a particular real product

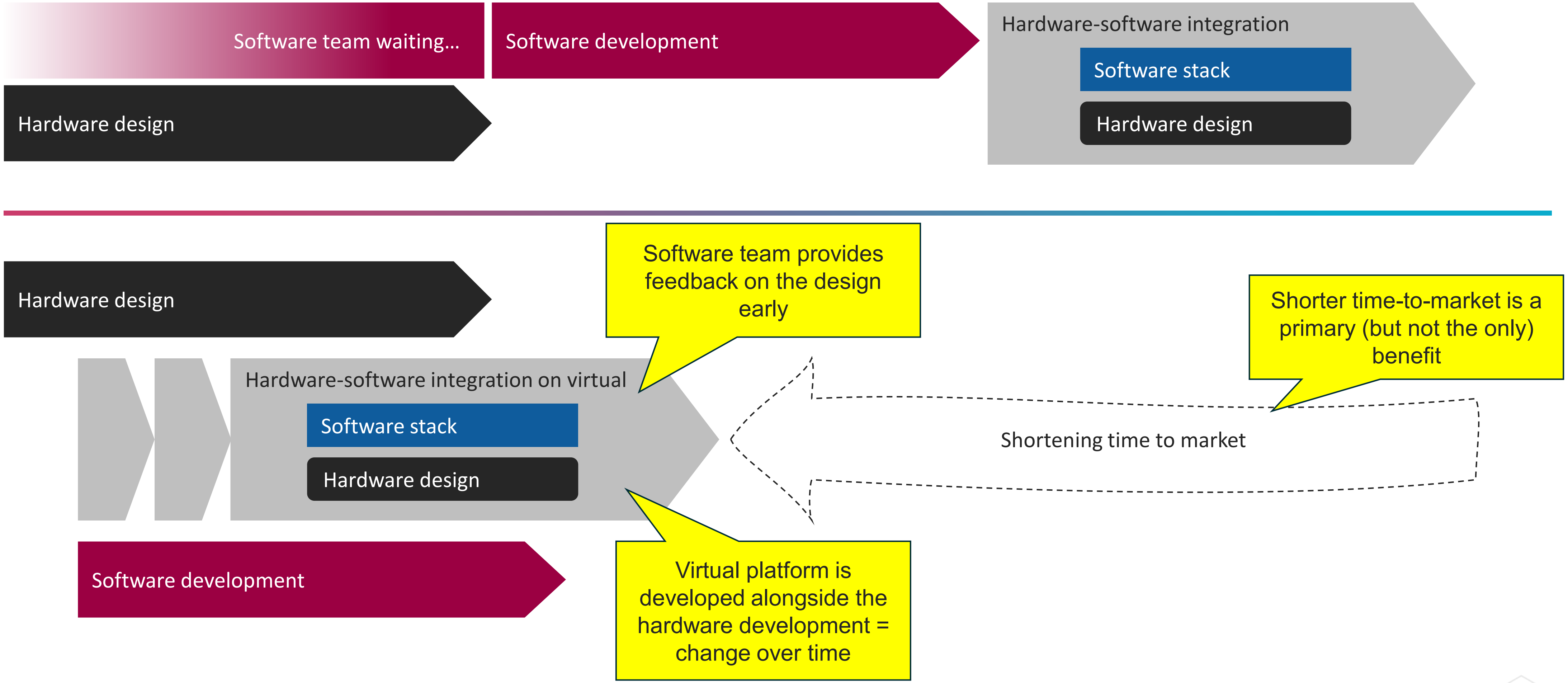
Simulation model abstraction level might vary.

More details and timing accuracy for IP blocks

Fast functional for system flows

System-level virtualization mirrors an actual system with multiple boards, interacting with the real world, the Internet, etc.

Using Models at the Implementation Stage: Shift Left

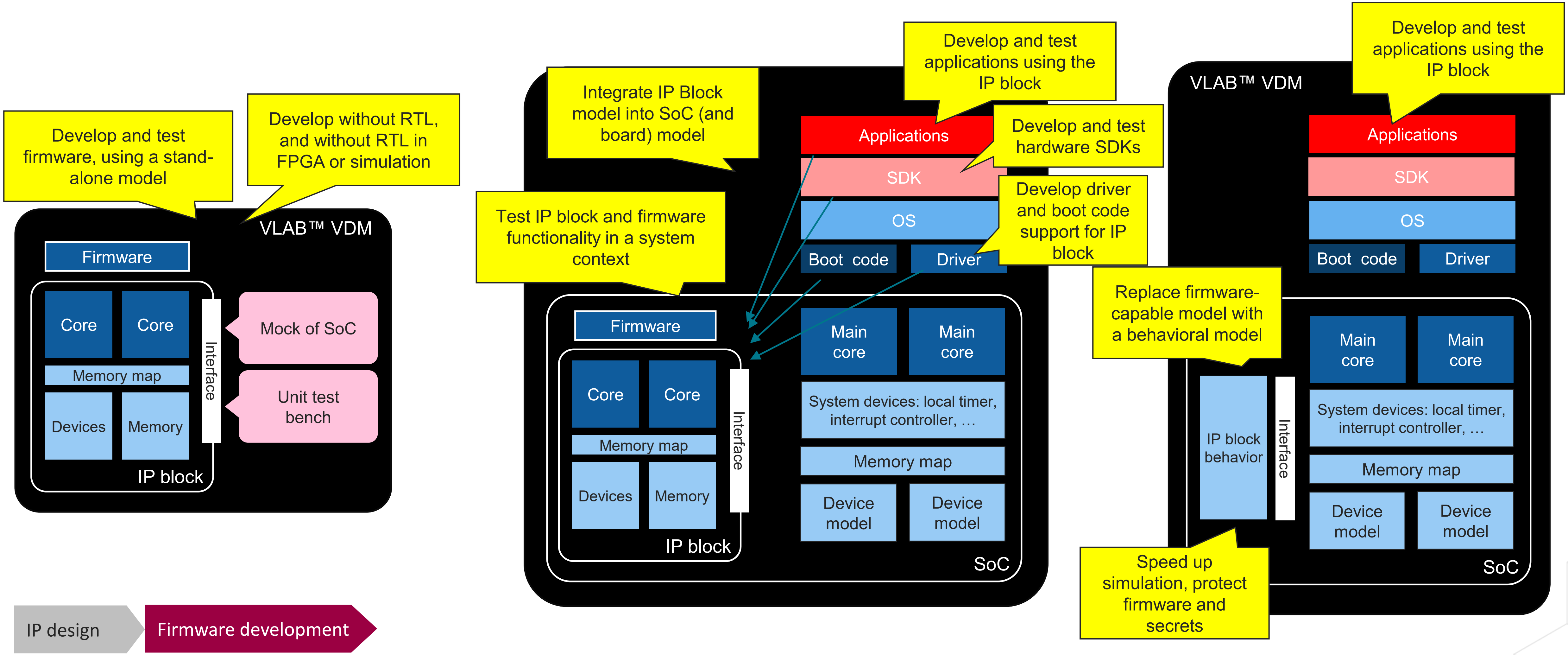


Software team provides feedback on the design early

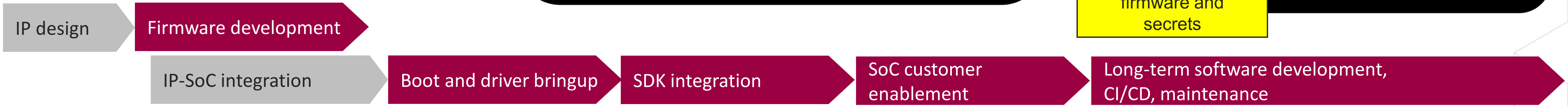
Shorter time-to-market is a primary (but not the only) benefit

Virtual platform is developed alongside the hardware development = change over time

IP Block Development: Software on VP



Develop and test firmware, using a stand-alone model
Develop without RTL, and without RTL in FPGA or simulation
VLAB™ VDM
Firmware
Core
Core
Memory map
Devices
Memory
Interface
IP block
Mock of SoC
Unit test bench
Integrate IP Block model into SoC (and board) model
Develop and test applications using the IP block
Applications
Develop and test hardware SDKs
SDK
OS
Develop driver and boot code support for IP block
Boot code
Driver
Test IP block and firmware functionality in a system context
Firmware
Core
Core
Memory map
Devices
Memory
Interface
IP block
Main core
Main core
System devices: local timer, interrupt controller, …
Memory map
Device model
Device model
SoC
VLAB™ VDM
Develop and test applications using the IP block
Applications
SDK
OS
Boot code
Driver
Replace firmware-capable model with a behavioral model
IP block behavior
Interface
Main core
Main core
System devices: local timer, interrupt controller, …
Memory map
Device model
Device model
SoC
Speed up simulation, protect firmware and secrets



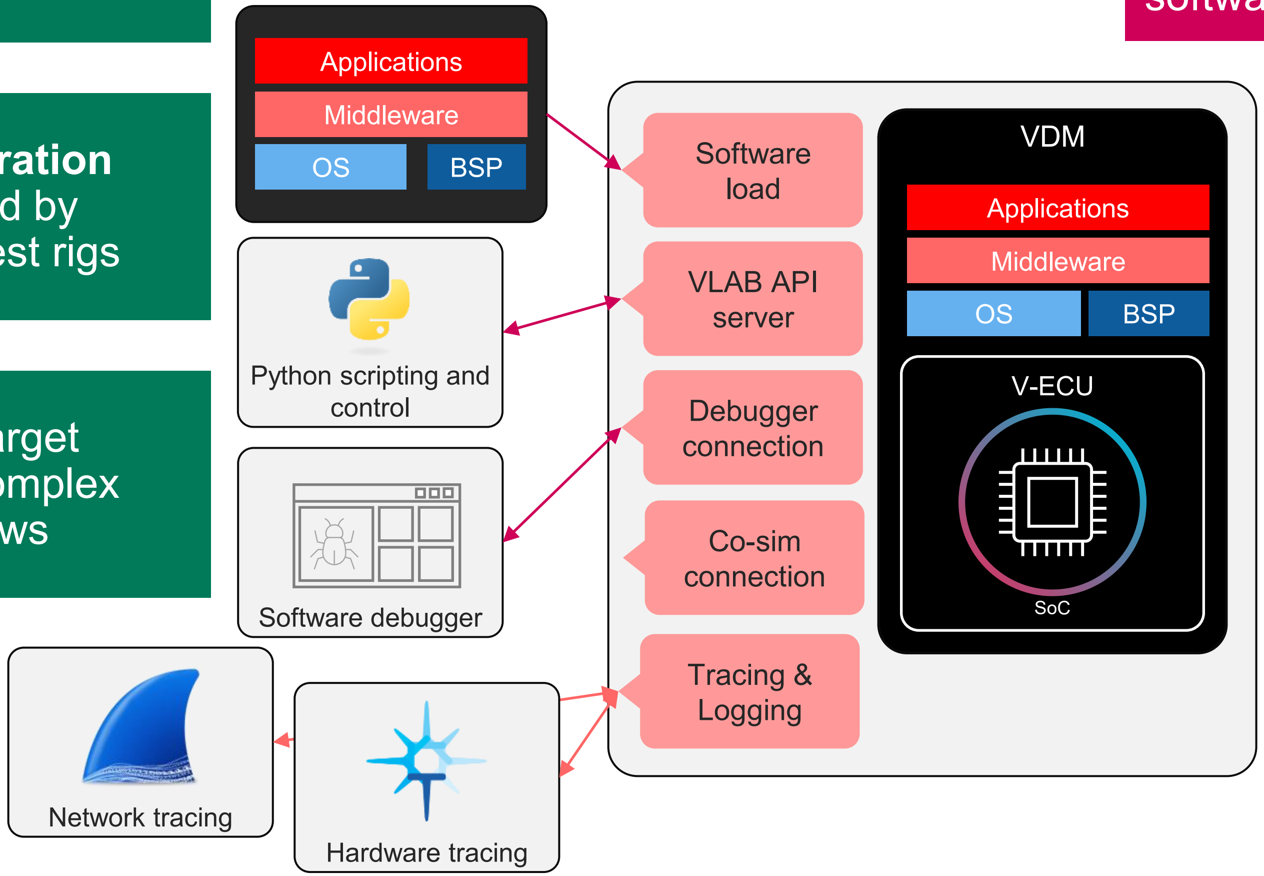
cādence®

Pre- or Post-Silicon: Free Developers from Hardware Limitations

Unlimited access to targets for testing - use any PC, server, or cloud VM to run tests

Apply **any test configuration on-demand** – not limited by availability of physical test rigs

Load software to the target system instantly – no complex flashing or download flows



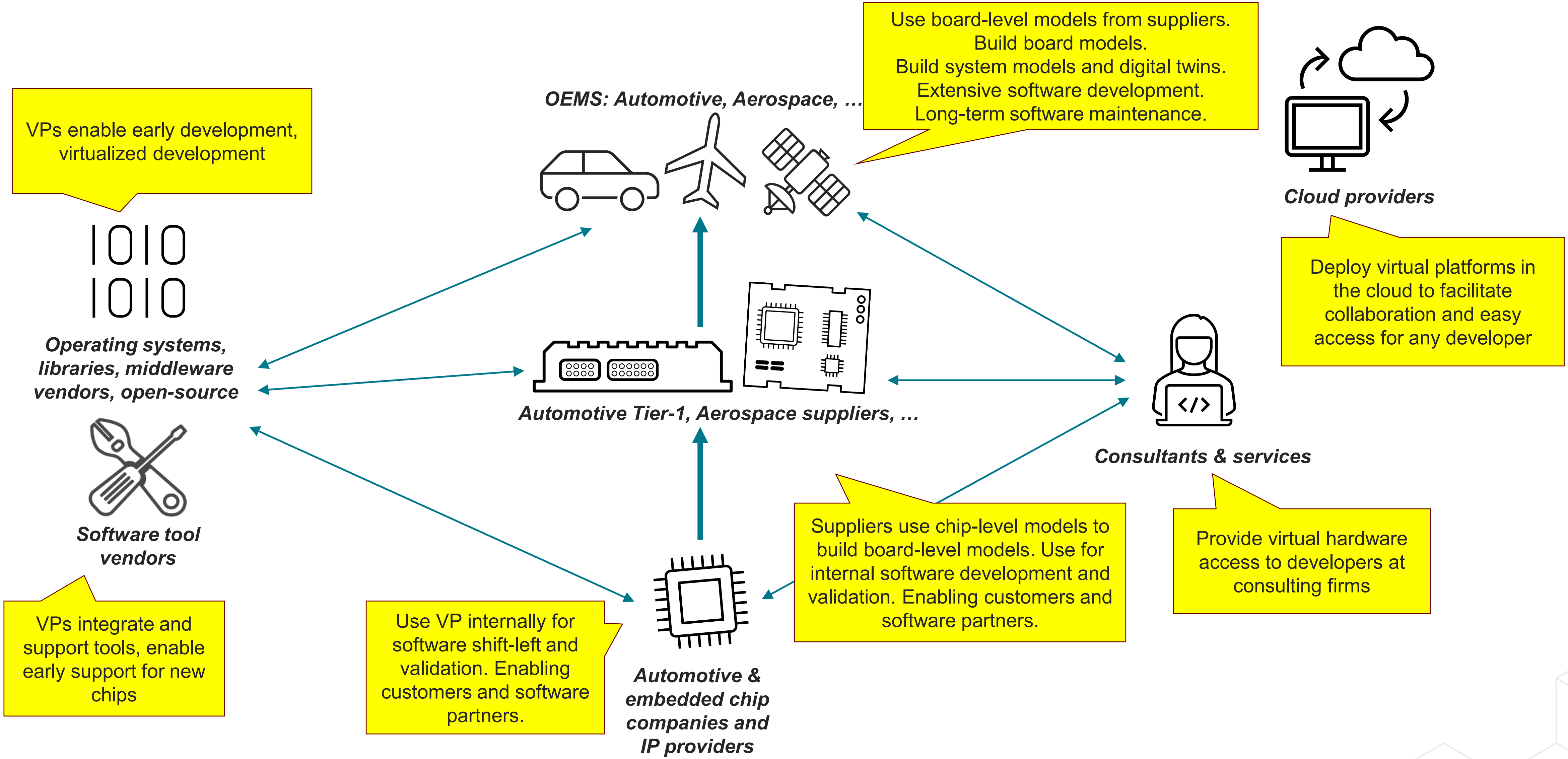
Automate and script any action —the simulator has perfect insight and control, control over time, react to hardware and software events

Debug better than hardware – powerful breakpoints, complete system inspection, debug across all cores, …

Trace and log everything– software, hardware, networks, …

Inject faults, force boundary conditions, …

Virtual Platforms as an Ecosystem Collaboration Tool



VPs enable early development, virtualized development

Operating systems, libraries, middleware vendors, open-source

Software tool vendors

VPs integrate and support tools, enable early support for new chips

OEMS: Automotive, Aerospace, …

Use board-level models from suppliers.
Build board models.
Build system models and digital twins.
Extensive software development.
Long-term software maintenance.

Automotive Tier-1, Aerospace suppliers, …

Automotive & embedded chip companies and IP providers

Use VP internally for software shift-left and validation. Enabling customers and software partners.

Suppliers use chip-level models to build board-level models. Use for internal software development and validation. Enabling customers and software partners.

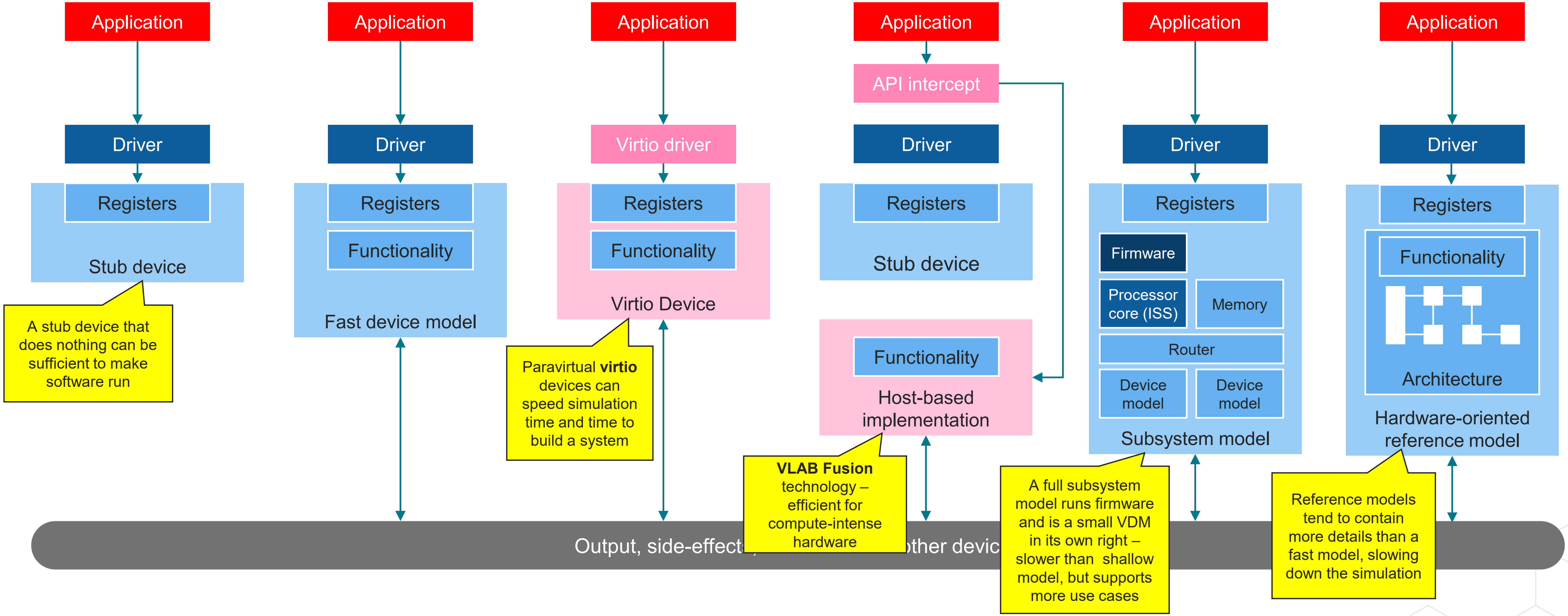
Cloud providers

Deploy virtual platforms in the cloud to facilitate collaboration and easy access for any developer

Consultants & services

Provide virtual hardware access to developers at consulting firms

Fast Functional Model Variants – Optimize for Performance



Application
Driver
Registers
Stub device

A stub device that does nothing can be sufficient to make software run

Application
Driver
Registers
Functionality
Fast device model

Application
Virtio driver
Registers
Functionality
Virtio Device

Paravirtual **virtio** devices can speed simulation time and time to build a system

Application
API intercept
Driver
Registers
Stub device
Functionality
Host-based implementation

VLAB Fusion technology – efficient for compute-intense hardware

Application
Driver
Registers
Firmware
Processor core (ISS)
Memory
Router
Device model
Device model
Subsystem model

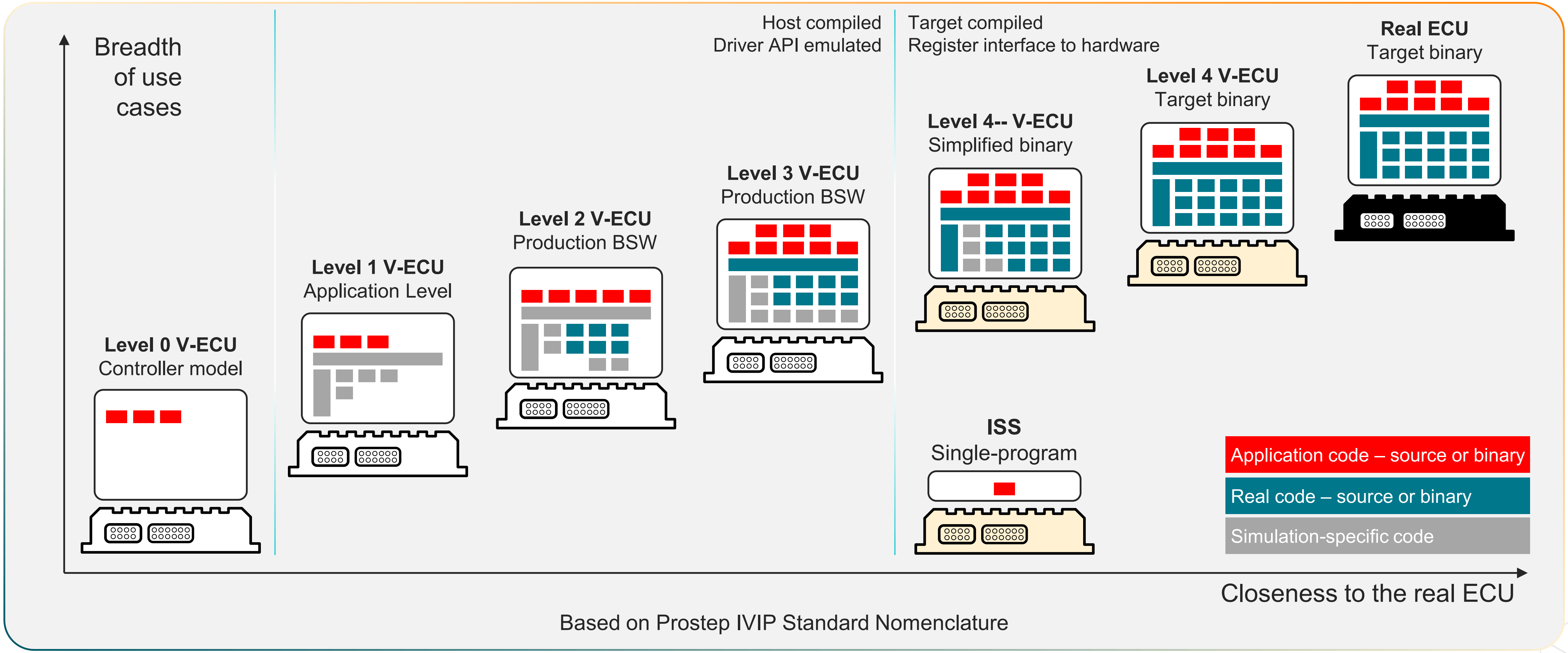
A full subsystem model runs firmware and is a small VDM in its own right – slower than shallow model, but supports more use cases

Application
Driver
Registers
Functionality
Architecture
Hardware-oriented reference model

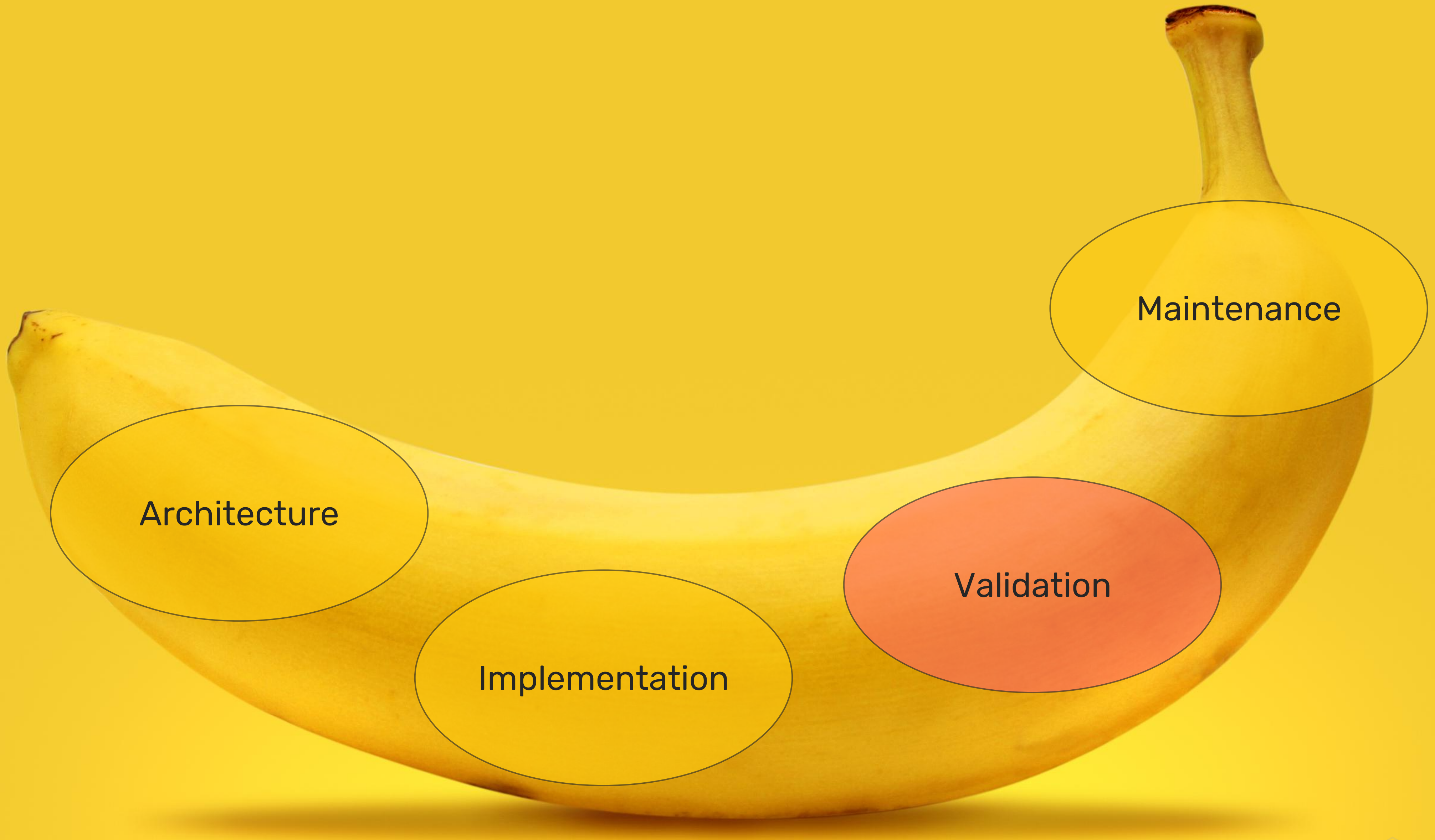
Reference models tend to contain more details than a fast model, slowing down the simulation

Output, side-effects ther devic

Automotive Variants Simpler Than Traditional Virtual Platforms



Based on Prostep IVIP Standard Nomenclature



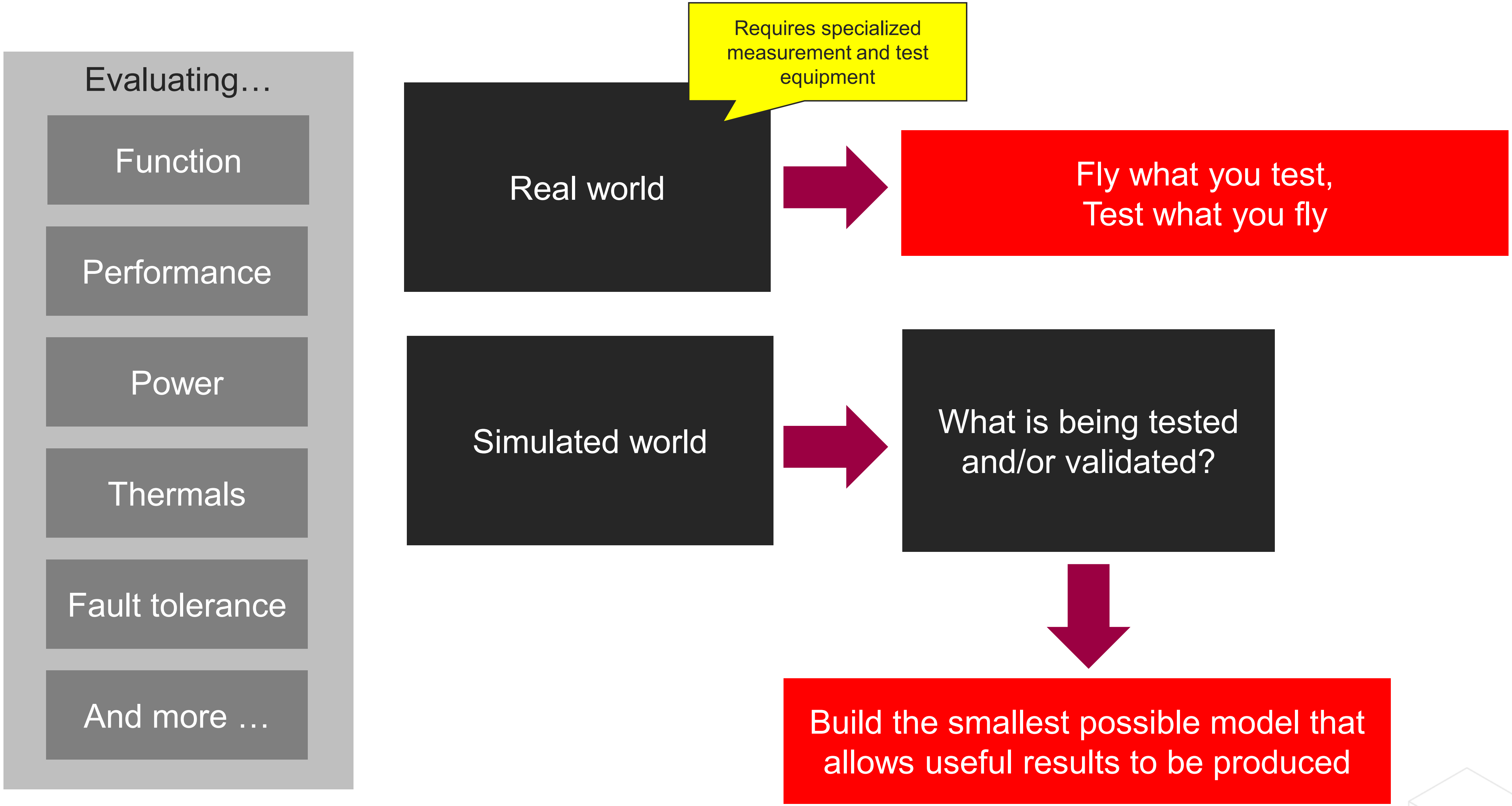
Maintenance

Architecture

Validation

Implementation

Validation – Does the System Work Right in Practice?



Evaluating…

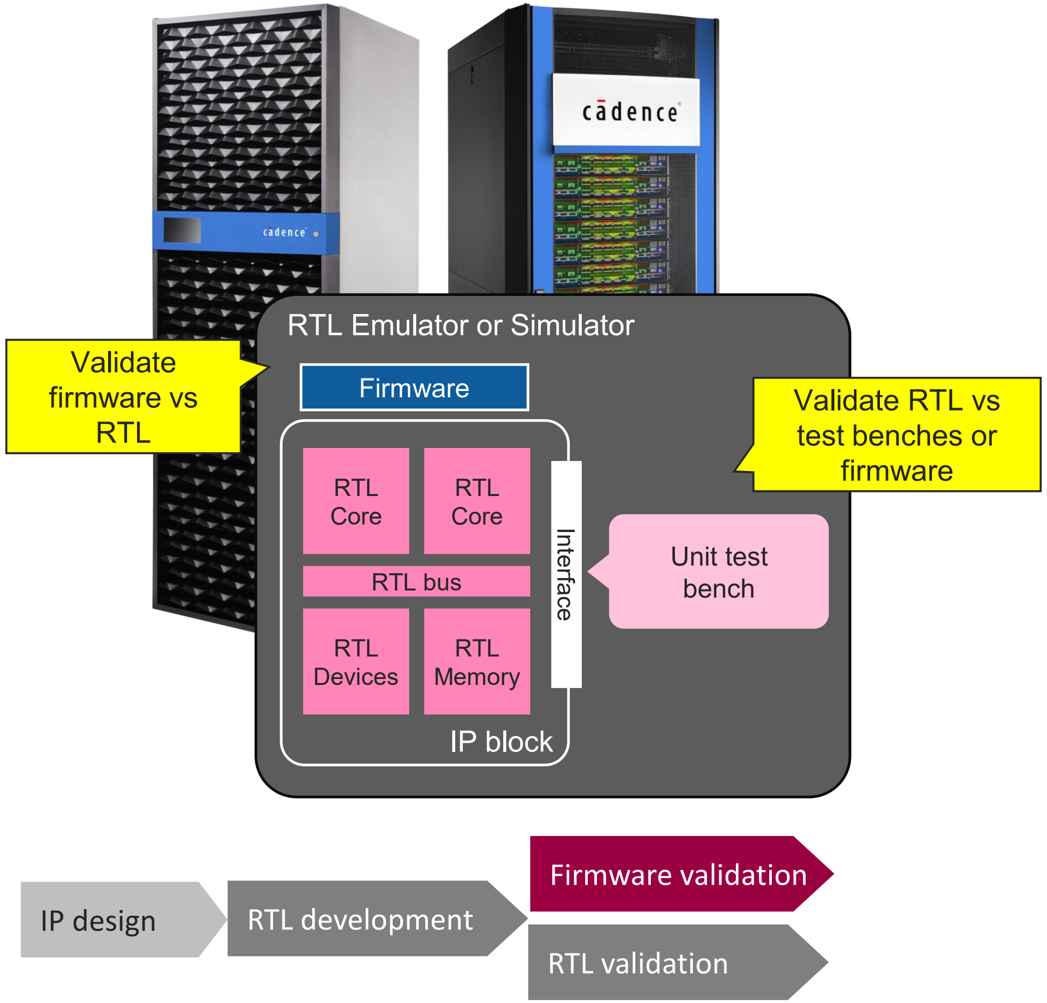
- Function
- Performance
- Power
- Thermals
- Fault tolerance
- And more …

Real world → Fly what you test, Test what you fly

Requires specialized measurement and test equipment

Simulated world → What is being tested and/or validated? → Build the smallest possible model that allows useful results to be produced

Example: IP Block Development: Validation of the RTL



Validate firmware vs RTL

RTL Emulator or Simulator

Firmware

RTL Core

RTL Core

RTL bus

RTL Devices

RTL Memory

IP block

Interface

Unit test bench

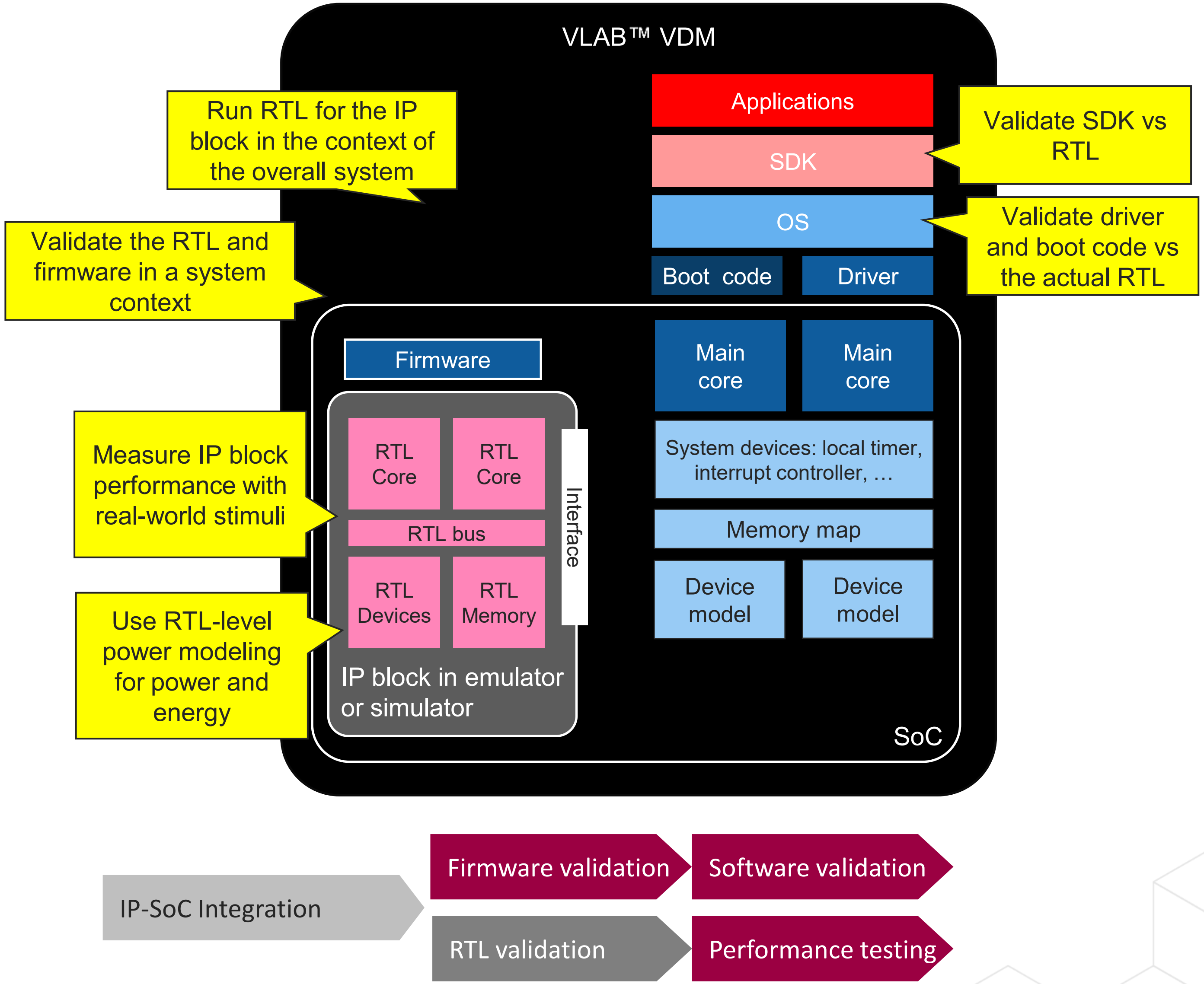
Validate RTL vs test benches or firmware

IP design

RTL development

Firmware validation

RTL validation



VLAB™ VDM

Run RTL for the IP block in the context of the overall system

Validate the RTL and firmware in a system context

Measure IP block performance with real-world stimuli

Use RTL-level power modeling for power and energy

Applications

SDK

OS

Boot code

Driver

Validate SDK vs RTL

Validate driver and boot code vs the actual RTL

Firmware

RTL Core

RTL Core

RTL bus

RTL Devices

RTL Memory

Interface

IP block in emulator or simulator

Main core

Main core

System devices: local timer, interrupt controller, …

Memory map

Device model

Device model

SoC

IP-SoC Integration

Firmware validation

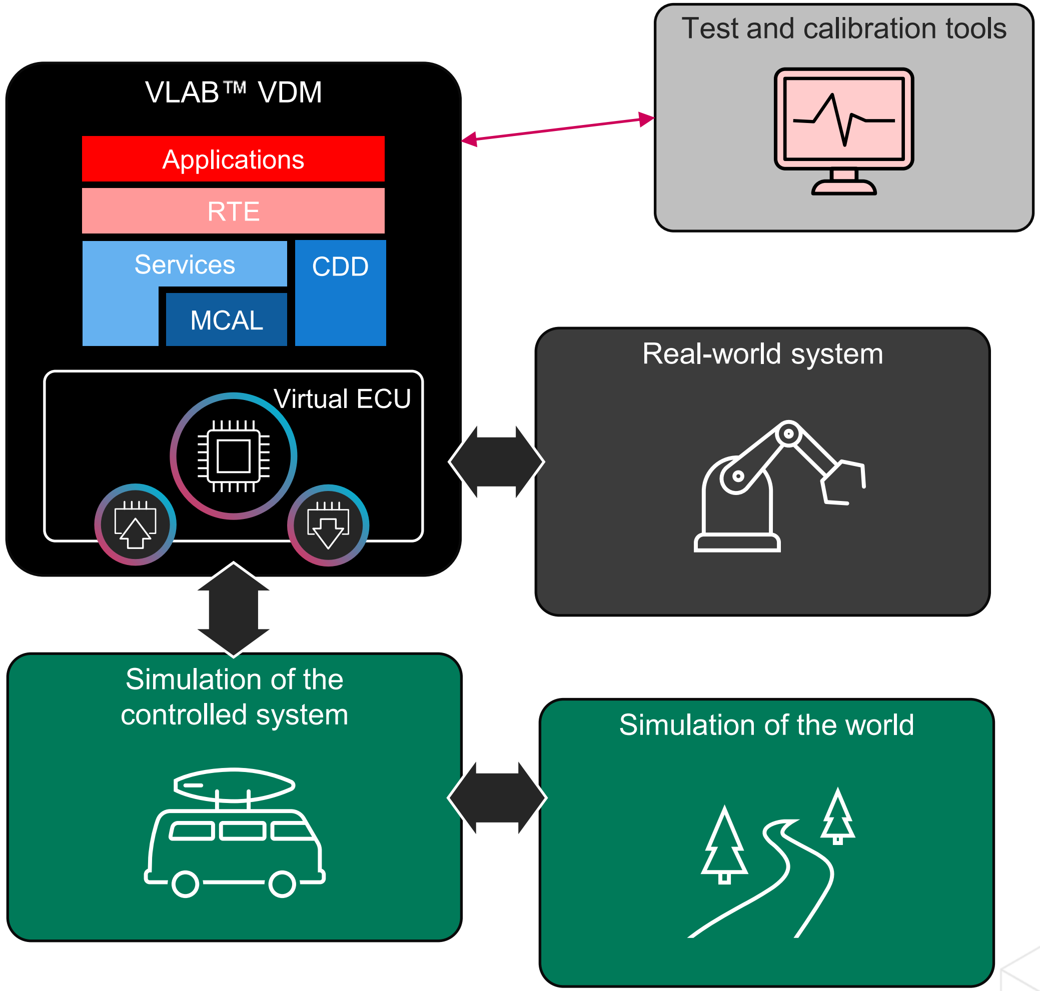
Software validation

RTL validation

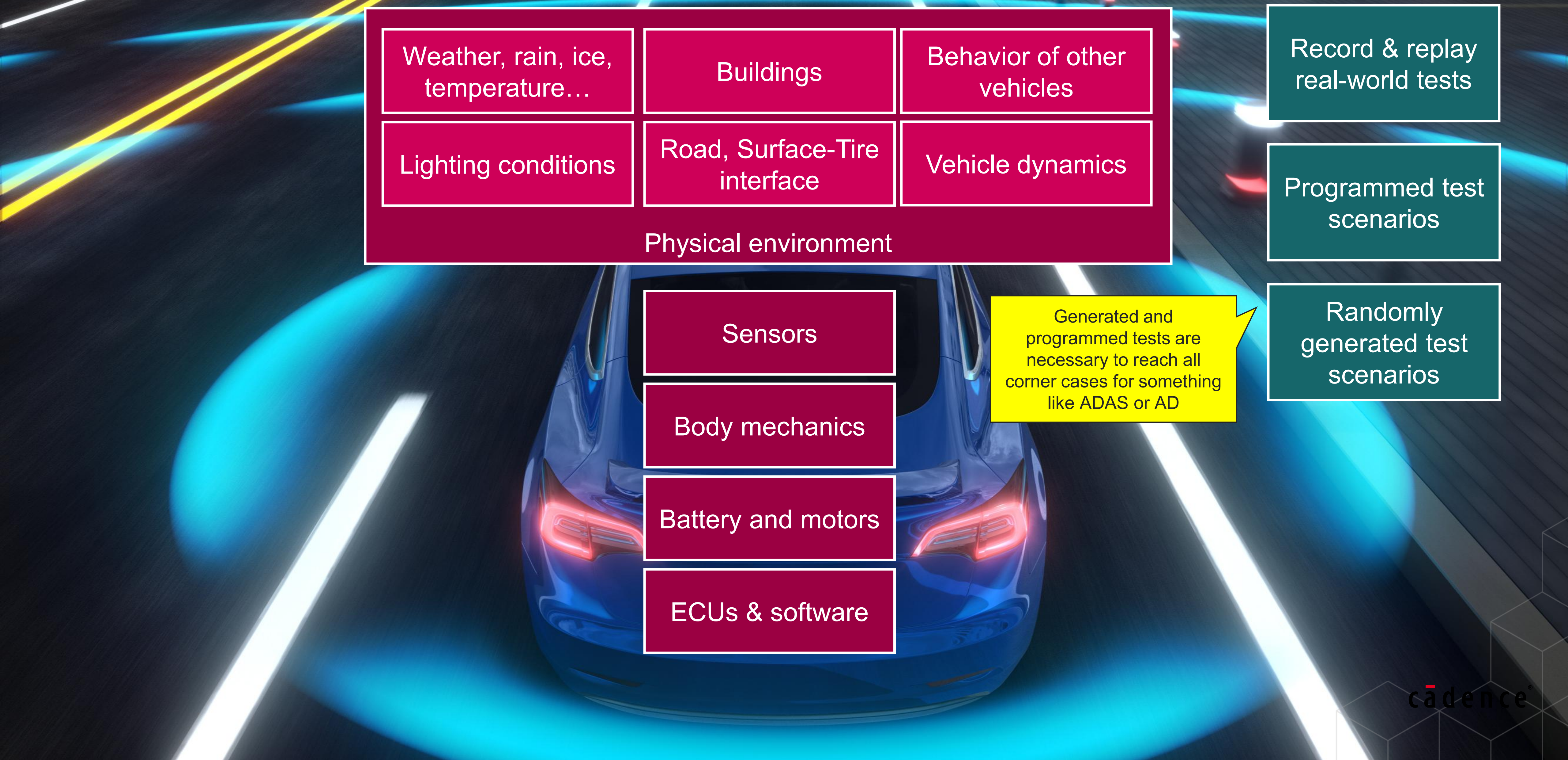
Performance testing

Virtual Hardware-in-the-Loop Testing

- Hardware-in-the-Loop is the “gold standard” for software testing
  - ECU or network of ECUs
  - Connected to physical simulation setups
  - *(whole ecosystem of solutions for this)*
- Replicate with virtual platforms for virtual testing
  - Real software stack
  - Simulation setup matching the real world
  - Closed-loop simulation including real-world behavior
  - Real systems attached to real-time sim
- Attach test and calibration tools to manage tests



Validating Automotive Systems



Weather, rain, ice, temperature…

Buildings

Behavior of other vehicles

Lighting conditions

Road, Surface-Tire interface

Vehicle dynamics

Physical environment

Sensors

Body mechanics

Battery and motors

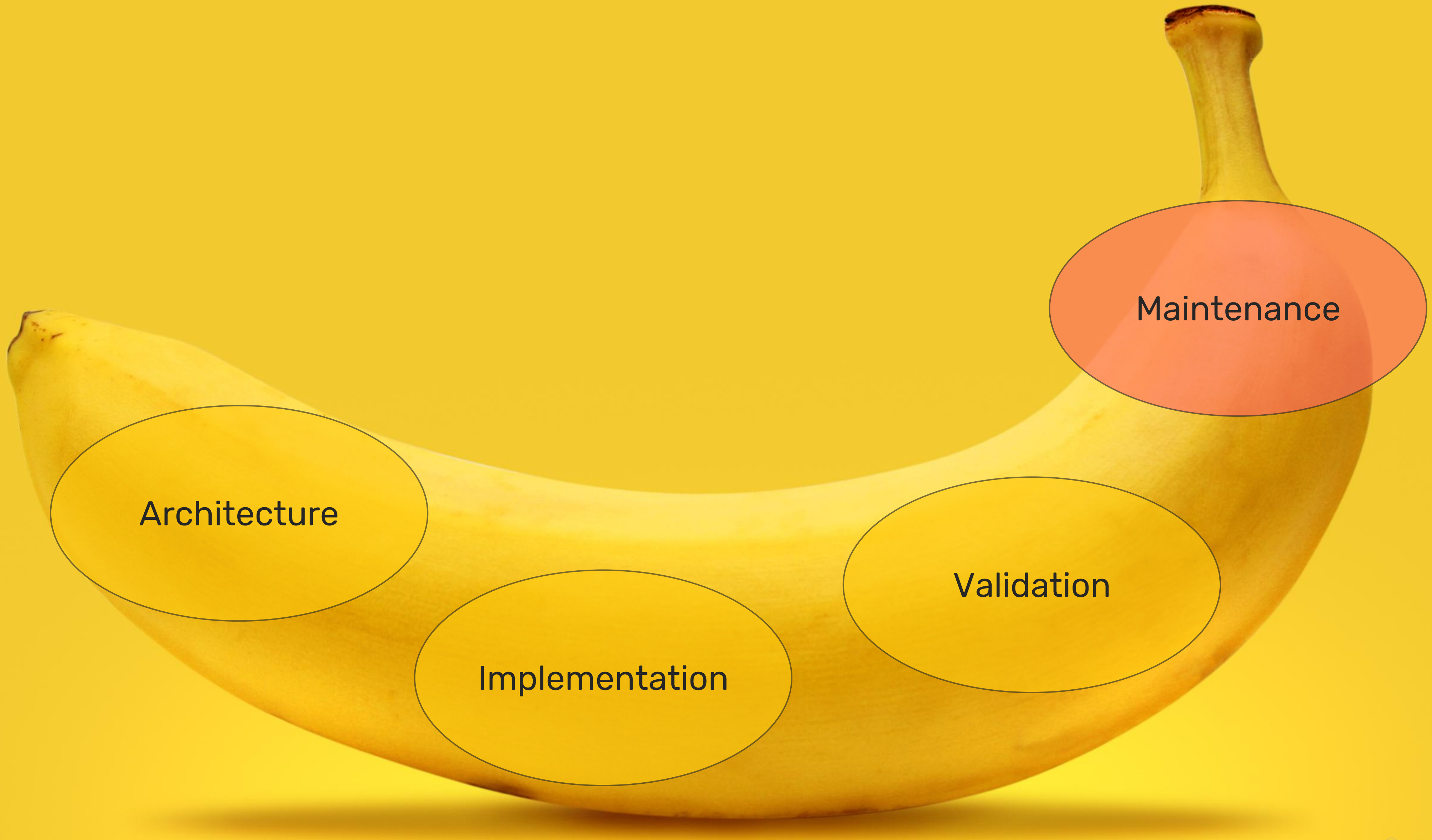
ECUs & software

Generated and programmed tests are necessary to reach all corner cases for something like ADAS or AD

Record & replay real-world tests

Programmed test scenarios

Randomly generated test scenarios



Architecture

Implementation

Validation

Maintenance

Deployed Systems Still Require Software Updates

New standards

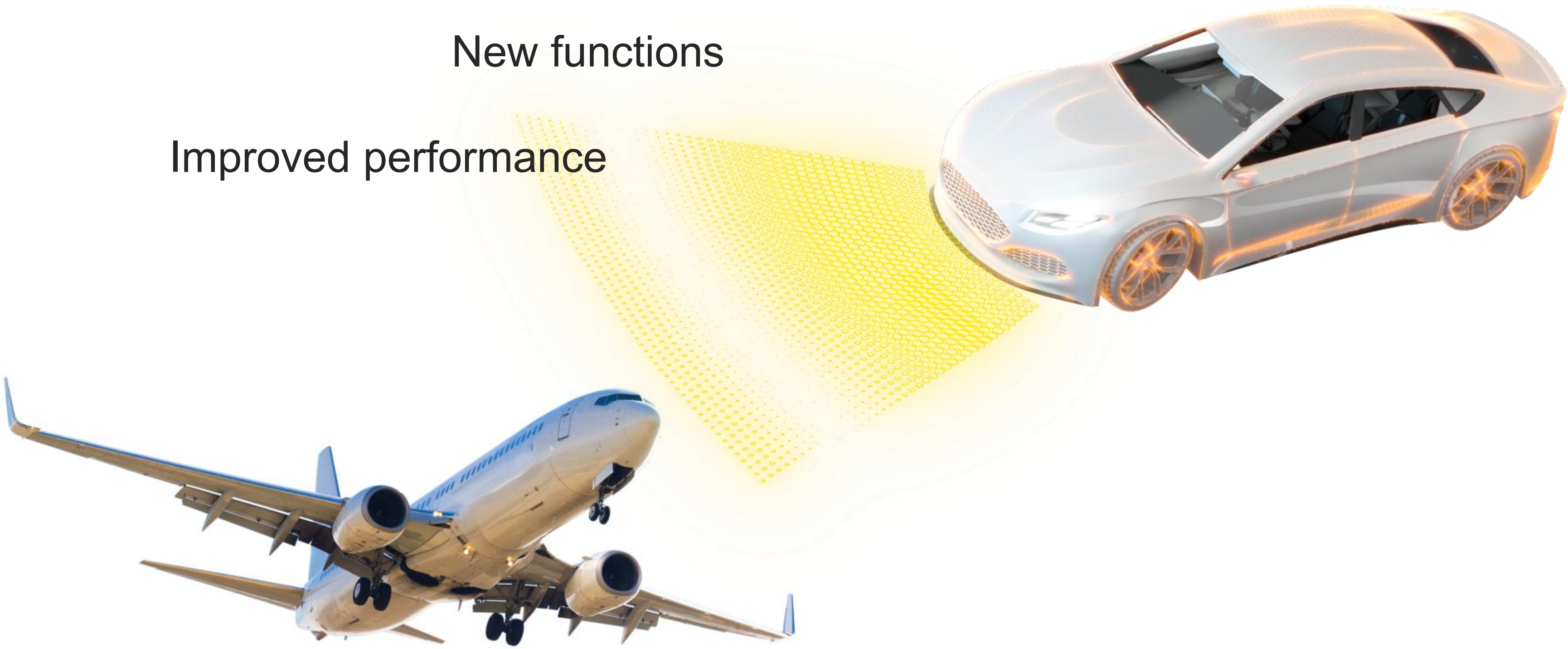
Look and feel updates

New operating modes

Updated functionality

New functions

Improved performance

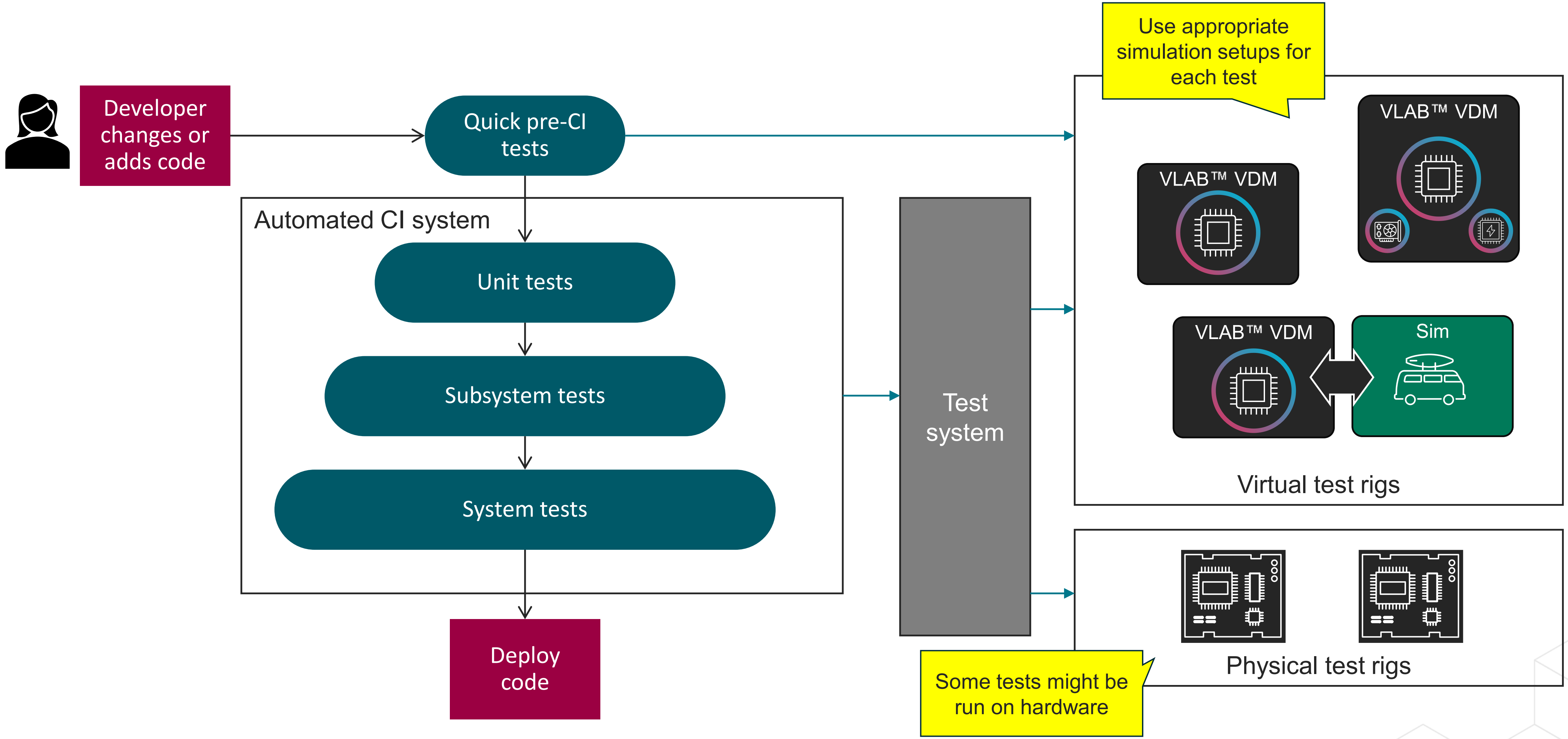


Security fixes

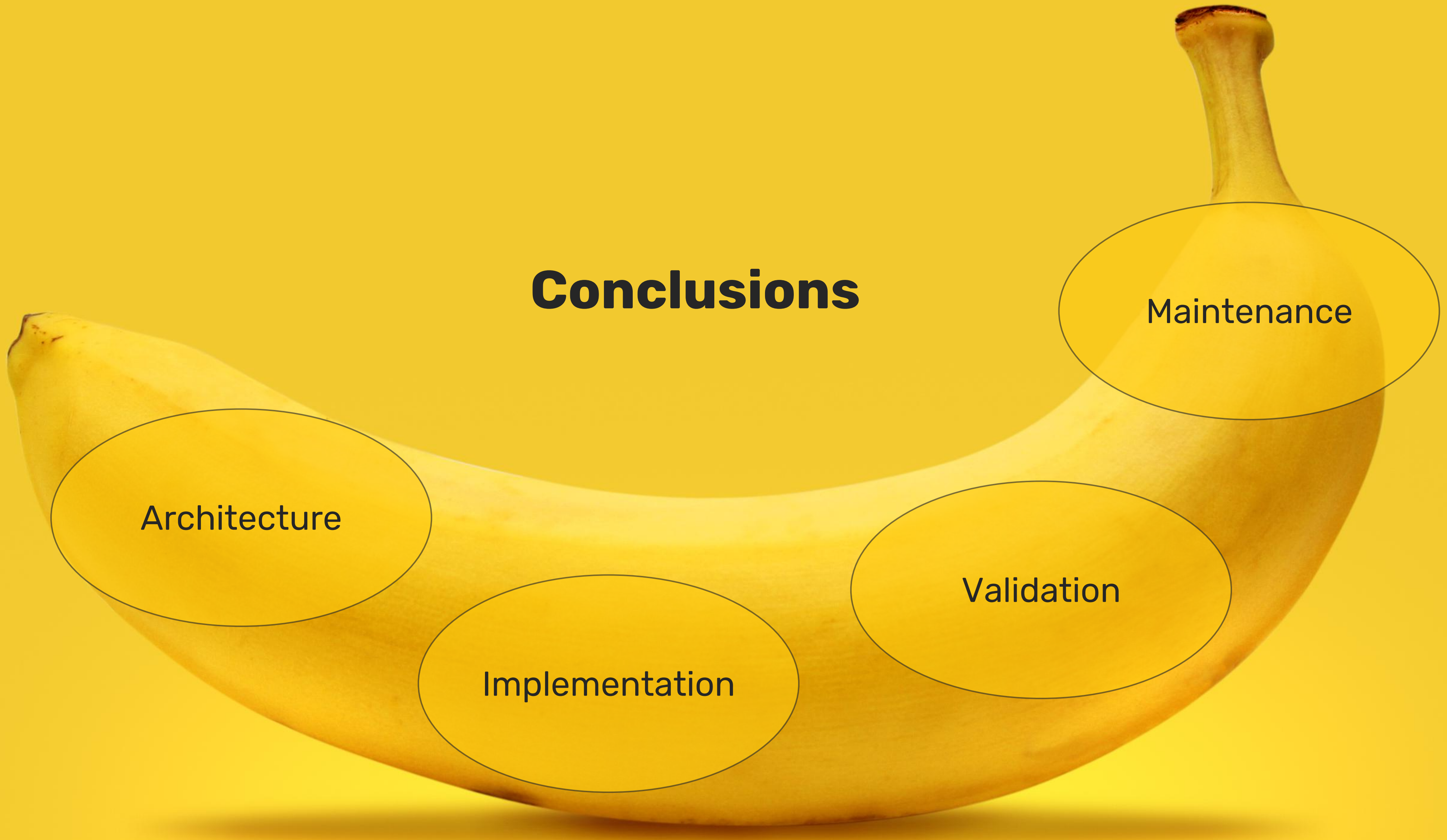
Safety fixes

Actually shipping what was promised…

Continuous Integration and Deployment



Conclusions



Architecture

Implementation

Validation

Maintenance

Conclusions and Summary

Simulation and architecture can be done at many levels

System models can extend to networks and networks-of-networks

Use cases drive simulation setups

Expect to mix models from different sources and of different types

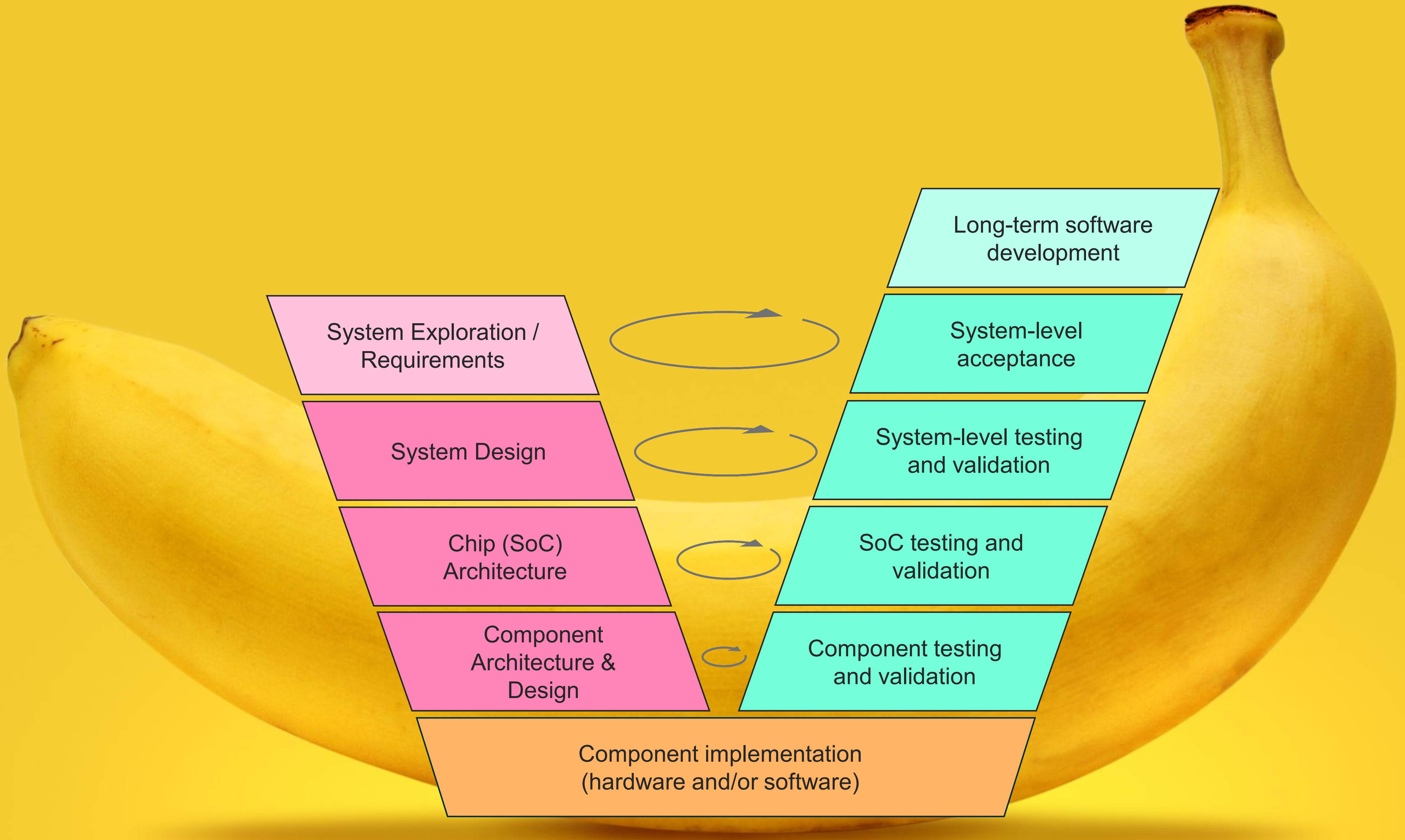
Use the right models for the right purpose!

Simulation models should always be more flexible than reality

Watch out for feedback loops and control loops in your workloads

You can only model what you have information about!

Model usage continues through the product lifecycle



System Exploration / Requirements

System Design

Chip (SoC) Architecture

Component Architecture & Design

Component implementation (hardware and/or software)

Component testing and validation

SoC testing and validation

System-level testing and validation

System-level acceptance

Long-term software development

cadence®

See www.vlabworks.com and www.cadence.com for more about us!