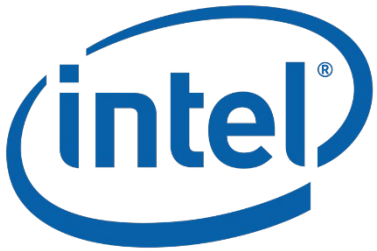
# Integrating Different Types of Models into a Complete Virtual System

Jakob Engblom, Andreas Hedström,
Xiuliang Wang, Håkan Zeffer

Intel, Stockholm, Sweden




accellera SYSTEMS INITIATIVE

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# Heterogeneous VPs

## Groups

VP team 1 | Architecture team | External IP vendor | Legacy models | VP team 2

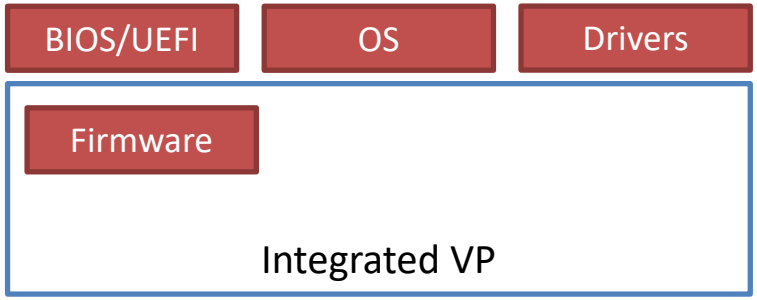
## Technologies

SystemC | C | TLM | DSL | Detailed models | PV | ISS type 1

TLM Modeling library 1 | TLM Modeling library 2 | TLM Modeling library 3 | No modeling library at all | C++ | ISS type 2

## Goal

Run firmware, boot code, OS, drivers, software loads




By combining many pieces from several sources into a single coherent platform


© Accellera Systems Initiative

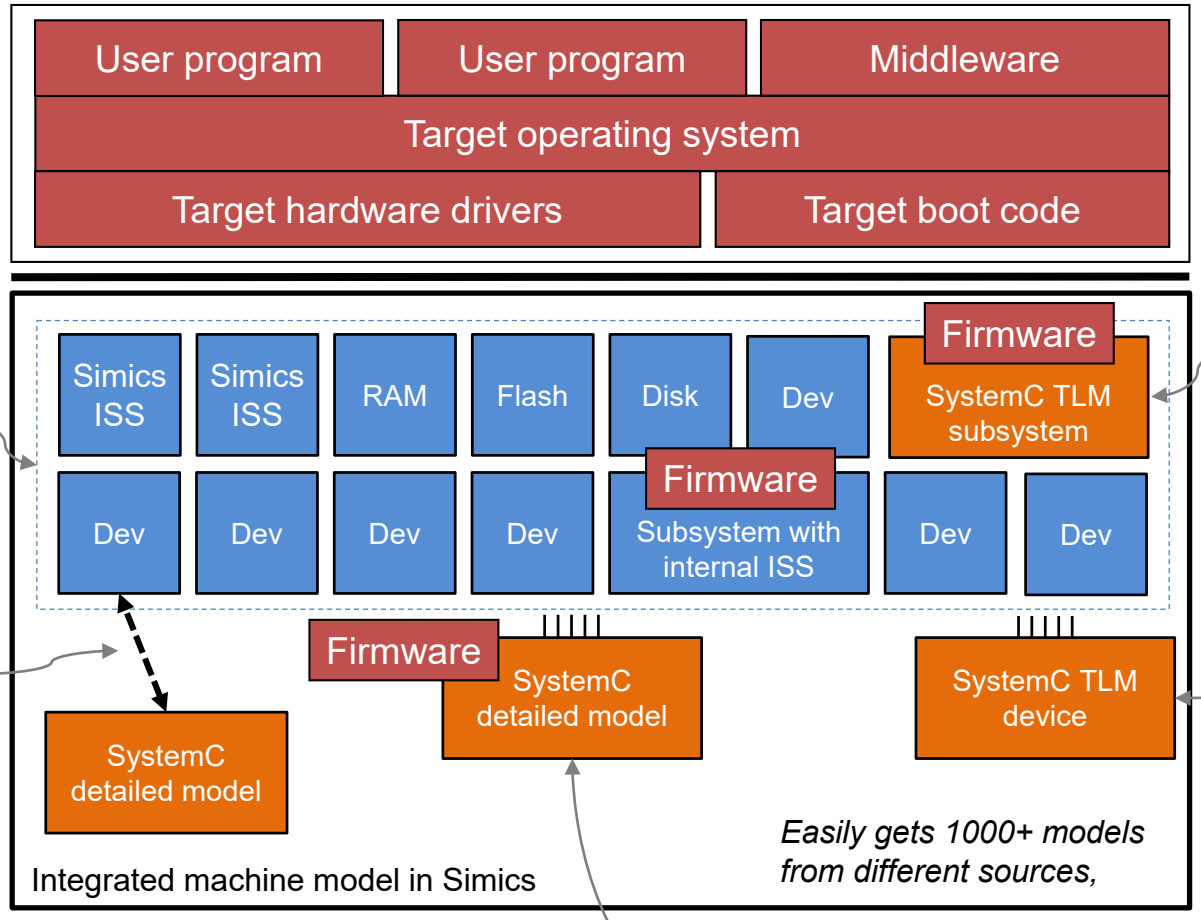
# Simics

- Virtual platform framework
- Designed to run large software stacks quickly
  - Run SW the primary purpose
- Designed to handle large targets and scale up
- Streamlined TLM semantics
  - "Software lineage" in the style of Qemu, IBM Mambo, ARM Fastsim, Mame, SPIM, ...
  - *Similar* to SystemC TLM LT
  - Different from SystemC in the details

- History
  - First code in early 1990s
  - **Virtutech** founded 1998
  - Acquired by **Intel** in 2010
  - Intel subsidiary **Wind River** sells Simics to general market
  - Large user base inside of Intel
- Long history of integrating various other simulators
  - Cycle-accurate
  - Different languages
  - Mechanics & physics
  - ...


© Accellera Systems Initiative

# Use Cases: SystemC Models in Simics System Context




Simics model is sufficient to boot and run (basic) software for the platform

Model of part of base platform, necessary for boot

Add additional hardware components to the base platform

Exchange fast model for detailed model for performance studies

Arbitary models: TLM, CCA, processor cores, ...

Explore architecture and performance of new hardware

© Accellera Systems Initiative

# Simics vs SystemC Semantics

| | Simics | SystemC |
|---|---|---|
| Model abstraction level | TLM | TLM + AV + PV + CCA + ... |
| TLM transaction phases | Single phase synchronous | Single phase synchronous, Multi-phase asynchronous, Cycle-driven |
| Memory transaction time | Zero time | Zero, fixed delay, dynamically computed |
| Time model | Local time with multiple clocks | Global time + temporal decoupling |
| Deltacycles | No | Yes |
| Asynchronous events | Inside time quantum | End of quantum or breaks time quantum |
| Interfaces | Unidirectional, Simics-defined | Bidirectional, TLM 2.0 MMB + custom |
| Threaded device model | No (passive reactive run-to-completion) | Yes, available and used |
| Host multithreading | Built-in | Not available |
| System configuration | Dynamic, runtime reconfigurable | Static after elaboration |
| Module packaging | Dynamic library (.so/.dll) | Static (.a), some.dll/.so |


© Accellera Systems Initiative

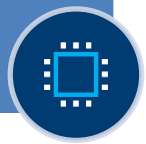
# Integration Design Issues

**Execution**

- ~~Multiple processes (co-simulation)~~
- Single process
- Exchange transactions at SystemC subsystem boundary

**Configuration**

- ~~Each SystemC module as a separate unit~~
- Top-level system in Simics
- SystemC subsystem as configuration unit



**Synchronization**

- ~~Unsynchronized execution~~
- Simics time as basis
- Run control
- Lazy time synchronization

**Communication**

- Simics interfaces as standard across system
- TLM transactions
- Synchronous transactions

**Adapter code**

- ~~Code on Simics-side~~
- User codes the adapter entirely in SystemC

**Model code**

- ~~Rewrite model~~
- No changes to SystemC model code required
- Allow binary-only models inside adapter

**User interface**

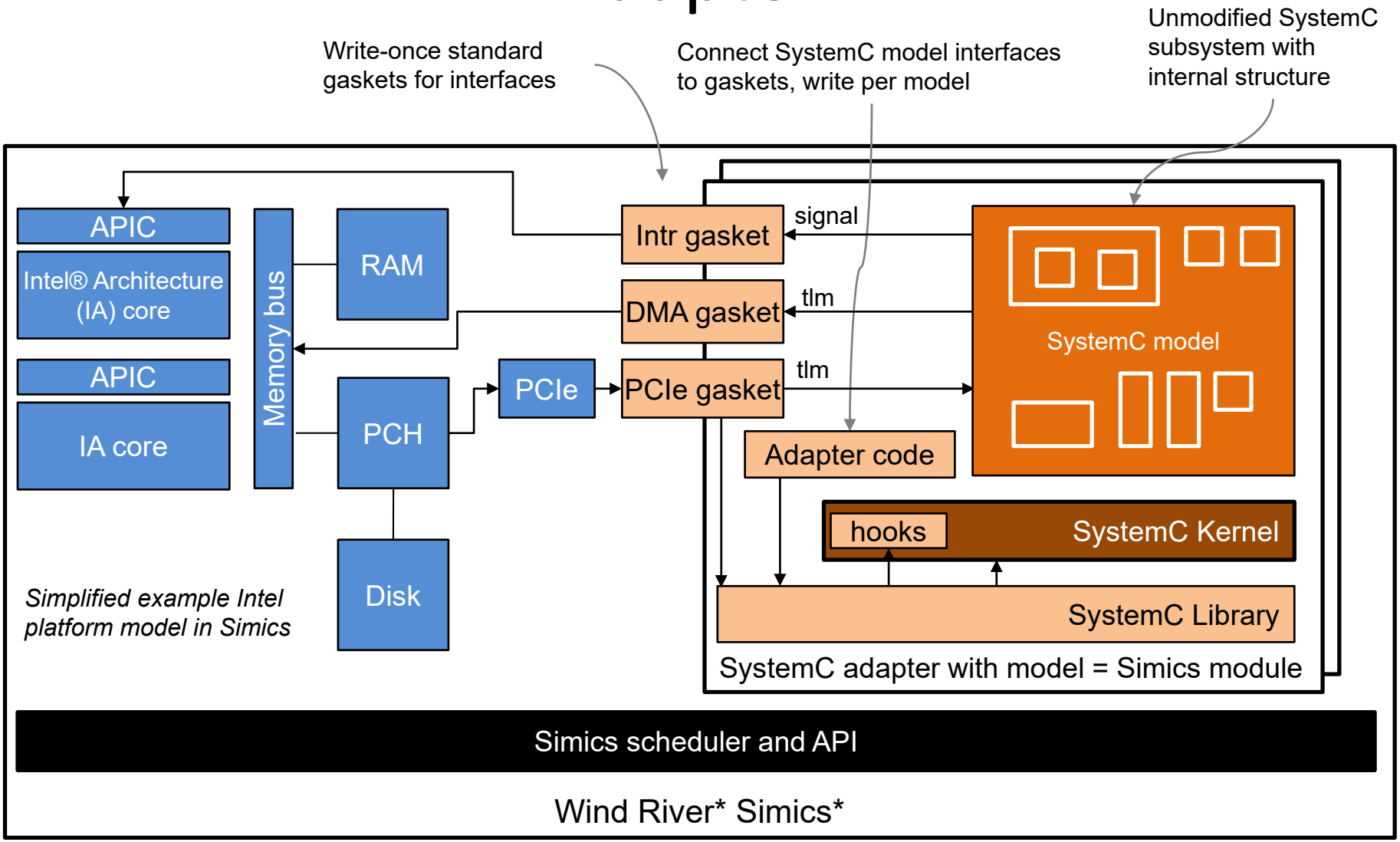
- ~~Separate user interface~~
- Common user interface
- Provide UI for SystemC models inside of Simics

**Inspection**

- Fit SystemC subsystem into Simics hierarchy
- Expose SystemC devices in Simics UI
- Special tools for SystemC


© Accellera Systems Initiative

# Adapter


Write-once standard gaskets for interfaces
Connect SystemC model interfaces to gaskets, write per model
Unmodified SystemC subsystem with internal structure
APIC
Intel® Architecture (IA) core
APIC
IA core
Memory bus
RAM
PCH
PCIe
Disk
Intr gasket
DMA gasket
PCIe gasket
signal
tlm
tlm
SystemC model
Adapter code
hooks
SystemC Kernel
SystemC Library
SystemC adapter with model = Simics module
Simplified example Intel platform model in Simics
Simics scheduler and API
Wind River* Simics*

# Gaskets

- Write once for each interface, reuse
  - One gasket per direction
- Outside the Adapter, simulation uses Simics interfaces
  - All modules are equivalent from an external view
  - Models from different sources use common interfaces
- Gasket encapsulates the protocol/interface translation
  - Data format and encoding, metadata, etc.
  - Timing, including SystemC AT to Simics synchronous
- Notes:
  - Real-world connections are handled via Simics
  - The gasket concept is not unique to Simics-SystemC. When models from different frameworks integrate, you always need a translation layer


© Accellera Systems Initiative

# Standardized Interfaces

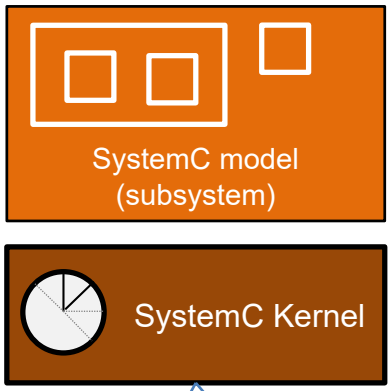
- Gaskets allow reuse of translations
  - Efficient if the same interfaces recur
- Requires modeling standards – which are lacking
  - TLM2 Base Protocol is really just a memory-mapped bus
- Examples where we need TLM standards:
  - Interrupt – sc_signal destroys scheduling
  - PCIe – more than just MMB
  - Ethernet – example of unidirectional interface

More interface standards are needed – or at least some shared repository where you can find what other people have done for reuse


© Accellera Systems Initiative

# Time Management

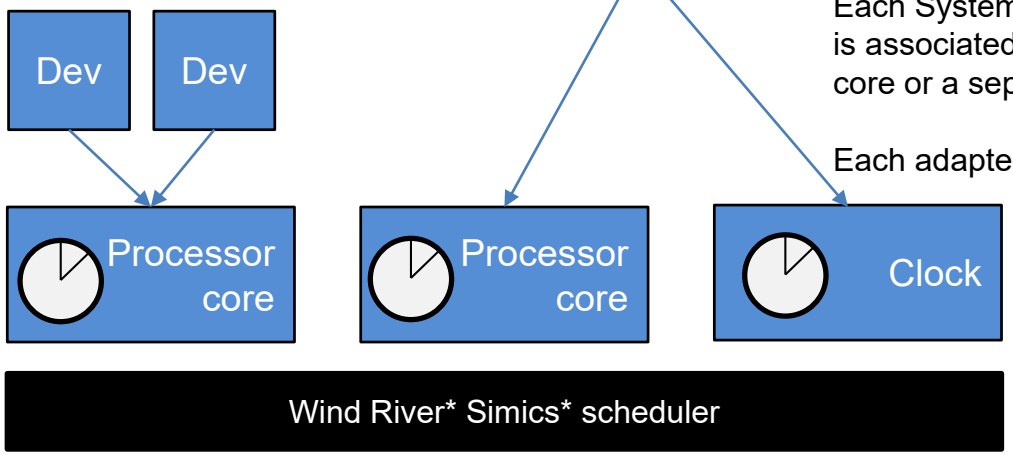
SystemC model gets its time from the SystemC kernel




SystemC time is usually behind Simics time. SystemC time is brought up to Simics time when an interaction or event happens (**Lazy synchronization**).

If SystemC needs to process time in order to compute the result to a transaction, SystemC time will be advanced ahead of Simics time

Simics devices typically get their time from a particular processor core, and drive events using that core




Each SystemC subsystem (adapter) is associated with a Simics processor core or a separate clock – up to user

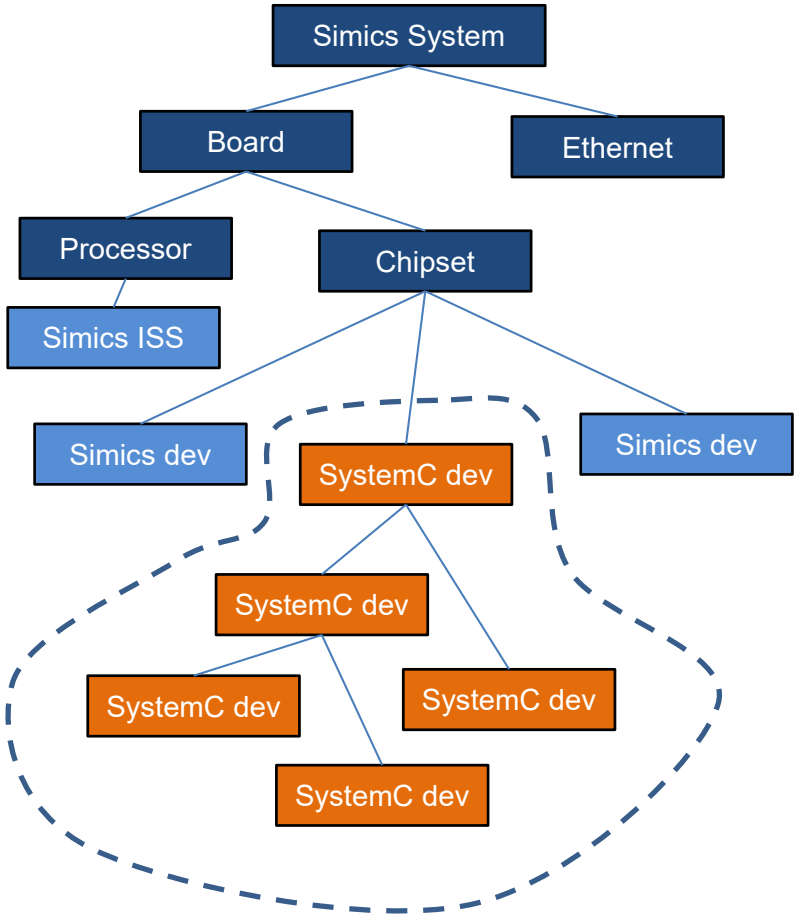
Each adapter can have its own time

Simics-level clocks are kept in synch (within time quanta)


© Accellera Systems Initiative

# Inspection & Hierachy

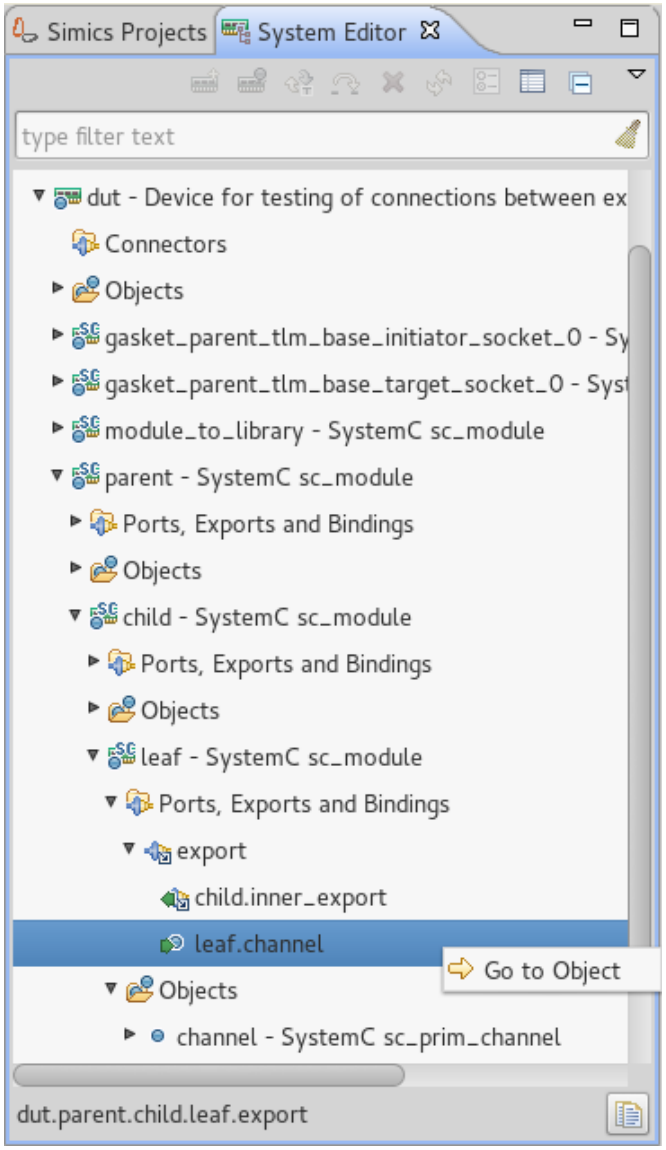
- SystemC model hierarchy integrated in Simics system hiearchy
  - Simics Eclipse GUI inspects SystemC seamlessly, with SystemC-specific features
  - Simics provides CLI + Python system for scripting, including SystemC
  - *sc_report* to Simics logs
- Command-line tools to break, inspect, trace, profile
  - On process, socket, event, signal
  - Time and memory usage of the SystemC subsystem
  - VCD output
  - TLM protocol checker
- Using specific modeling libraries in the SystemC models adds:
  - Registers, attributes, properties, back-door access
- SystemC editor & debugger



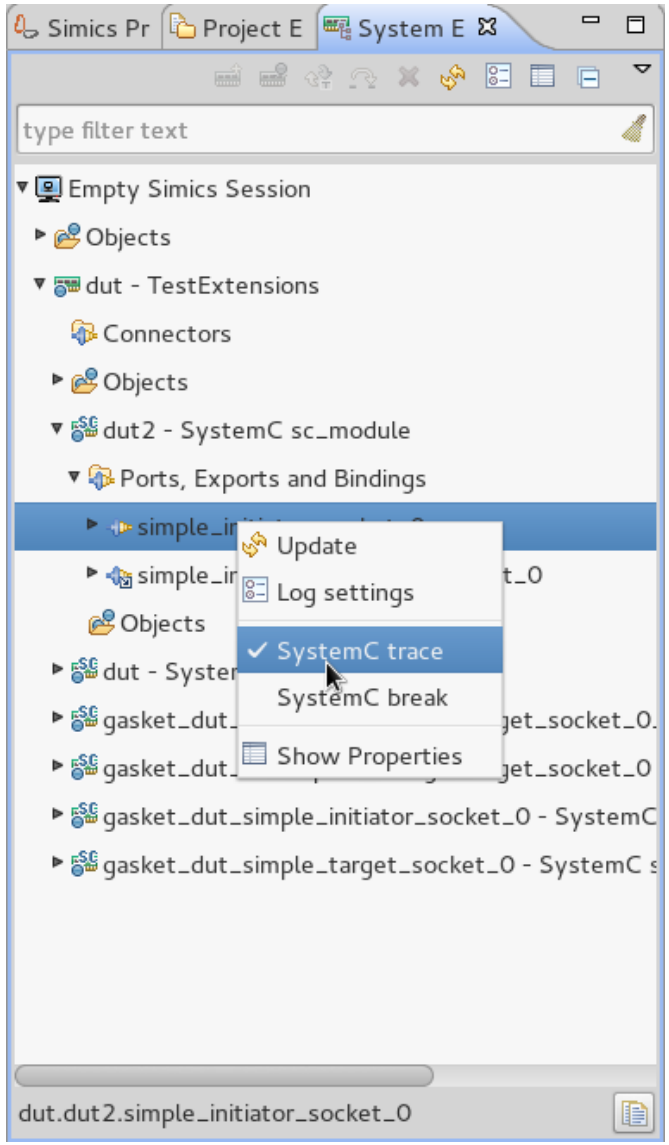

SystemC subsystem loaded as a single Simics module


© Accellera Systems Initiative

# GUI




SystemC models shown in the Simics Eclipse GUI, as part of a Simics system in the System Editor



# Performance

- Gaskets do not induce noticeable overhead
- Lazy synchronization increases performance
  - If a model is not used, it should not be activated
  - Well-written SystemC models should not impact speed
- Models *in use* will cause slower simulation
  - They are **added** to a base system and thus add more work

- Compared to native Simics models?
  - Typically slower... Since the SystemC models are more detailed
  - ”Apples and Oranges”


© Accellera Systems Initiative

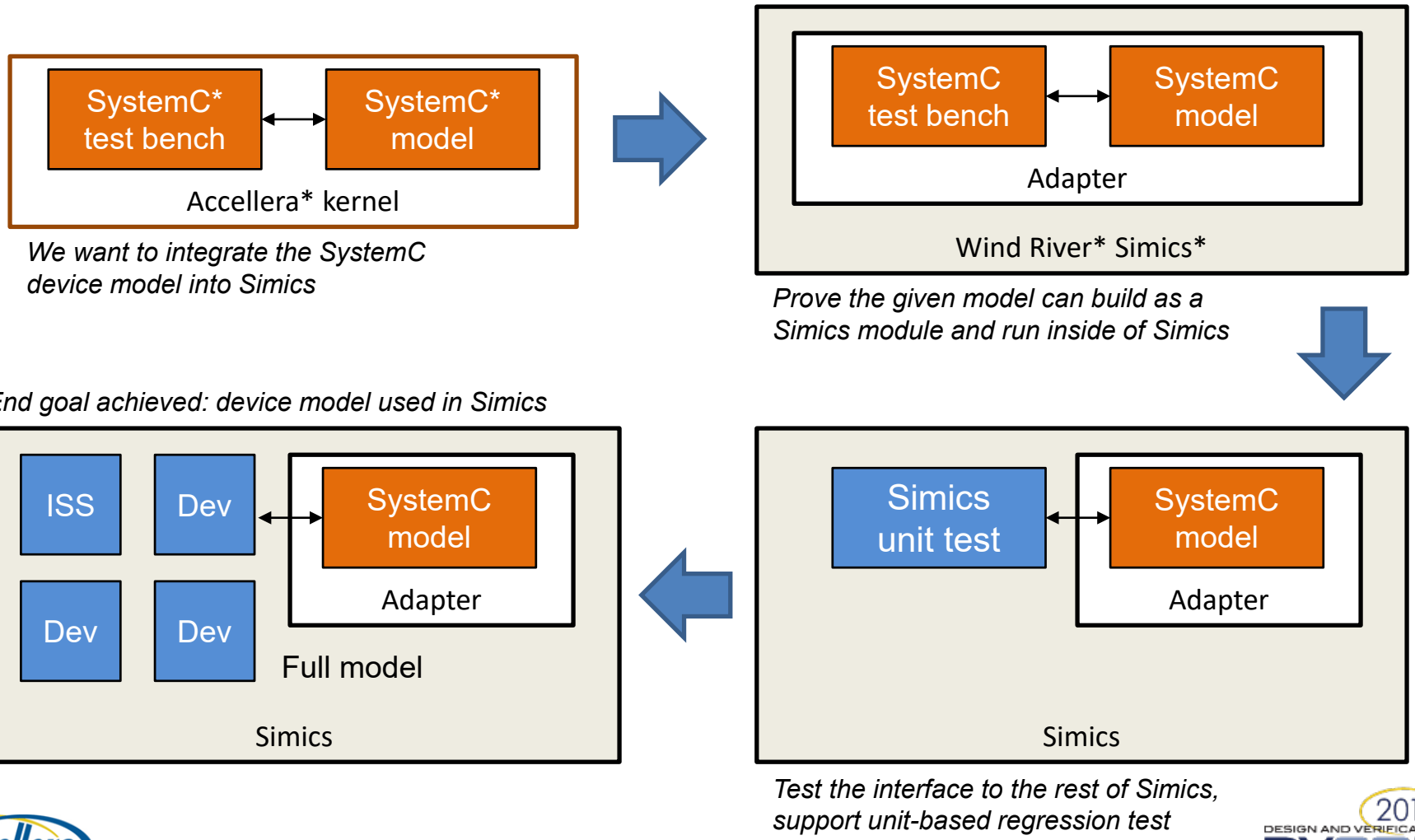
# Performance: Too many Events

- *(Observation from real engagements)*
- SystemC models often too detailed – even when TLM
- Models post frequent events to drive themselves
- Frequent events break Simics processors out of JIT or virtualized execution: loss of performance
  - Effect of a poorly written model is amplified in a non-linear way in a system context
  - Putting SystemC models on their own clock isolates the noise and can give 10x performance increase

SystemC TLM modeling needs to move towards a reactive passive style, rather than active threads


© Accellera Systems Initiative

# Integration Methodology

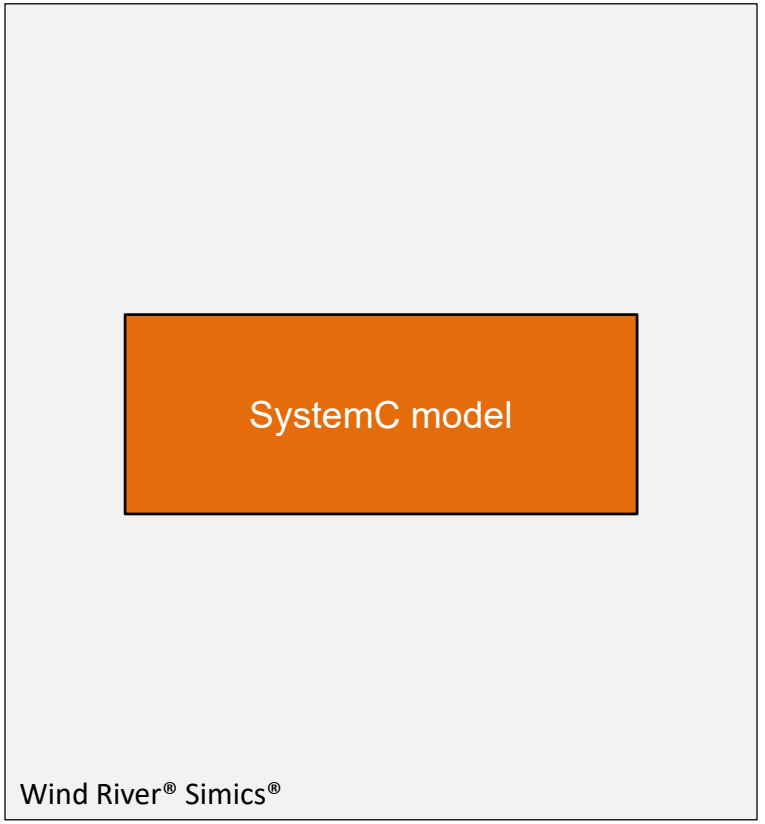




© Accellera Systems Initiative

# Example: Pure SystemC

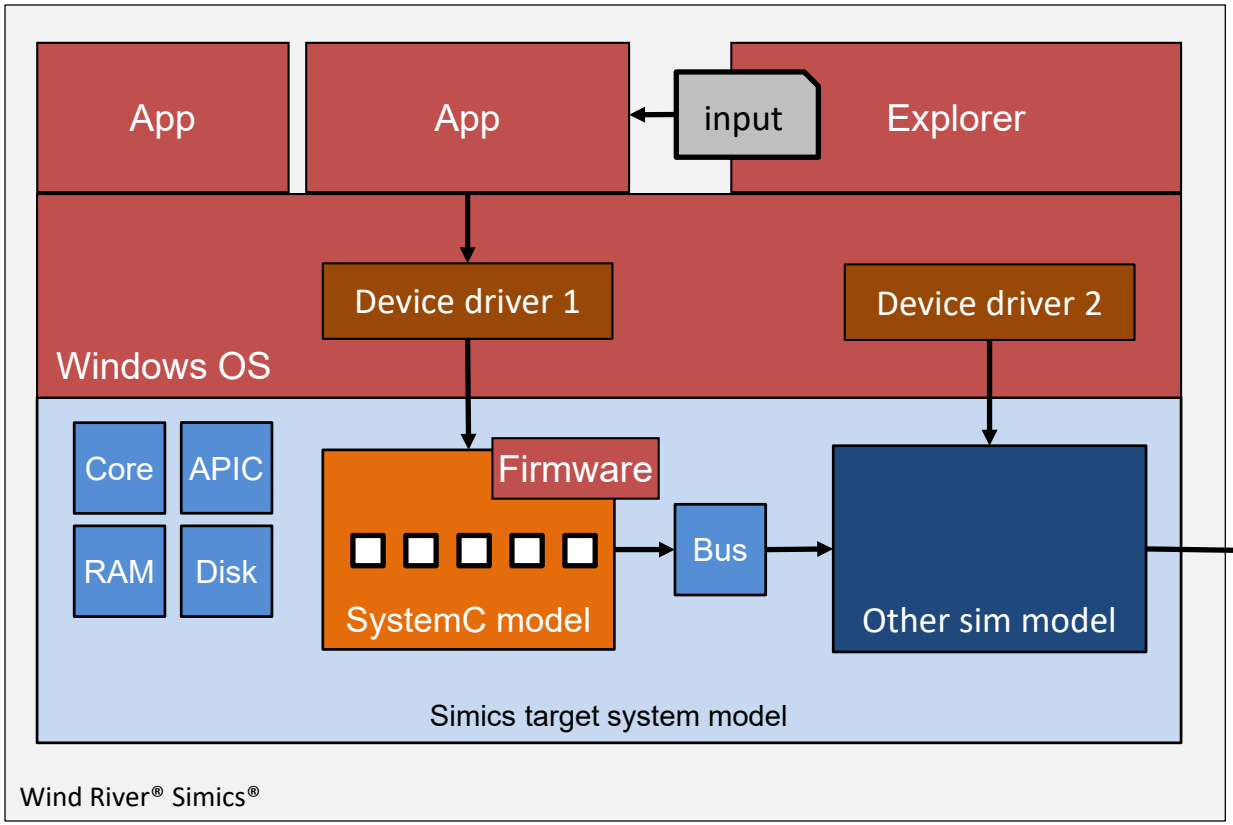
- Use Simics as a SystemC simulator for a SystemC model
- Better than using a plain Accellera kernel
- Benefits:
  - Binary delivery
  - Tooling for inspection, debug, profiling, …
  - Scripting & setup system
- Part of step-by-step integration strategy





© Accellera Systems Initiative

# Example: Cross-IP & Heterogeneous




Simics provides the System context, including the ability to boot an operating system like Windows

SystemC model and other models integrate into Simics – and can thus communicate with each other

Complete test passes data from user application to driver to device, and then through the other device – system-level flow


© Accellera Systems Initiative

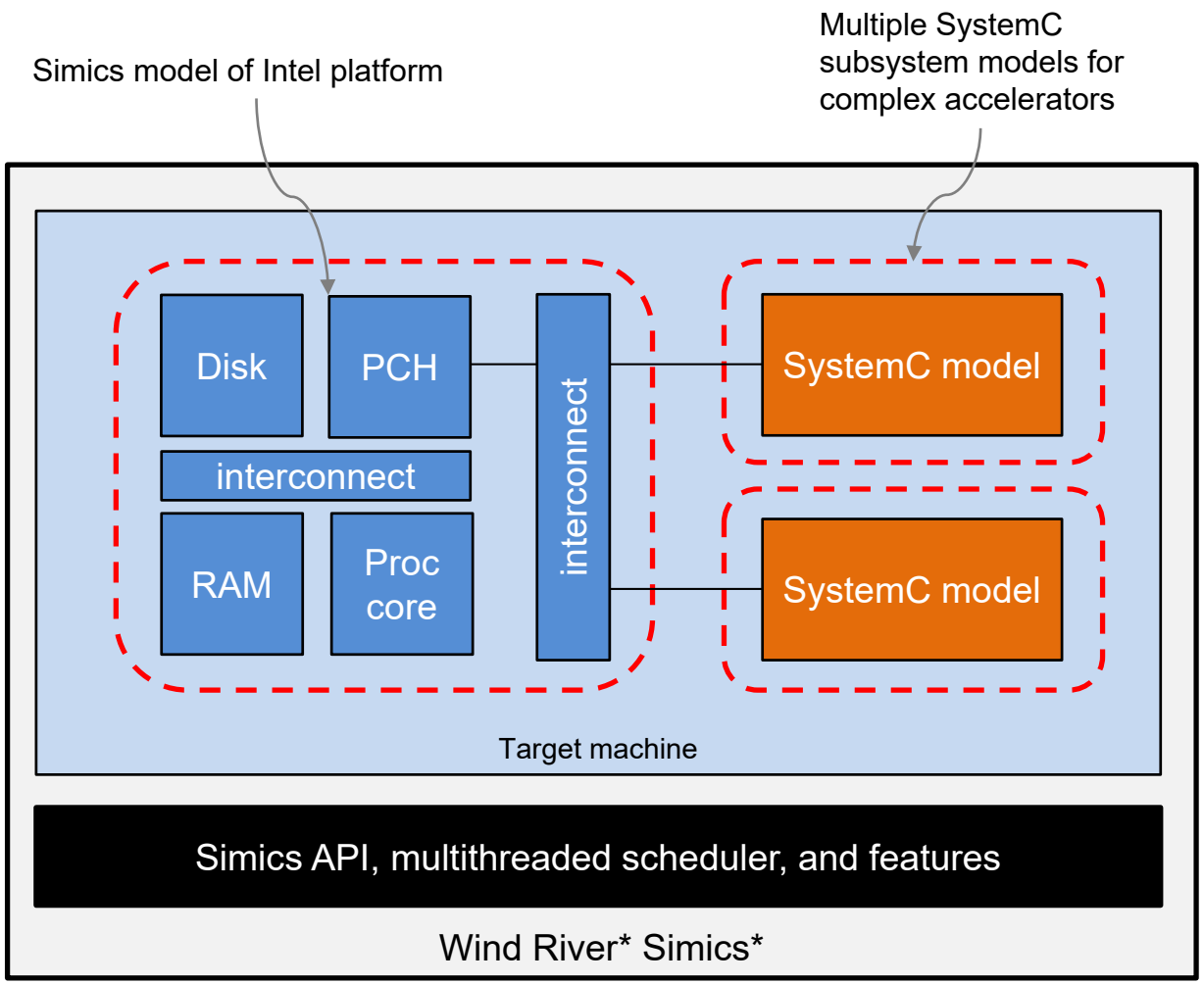
# Example: Multithreading

Simics provides the System context, including the ability to boot operating systems and run drivers and test code. As well as inject network traffic.

Using Simics multithreading, we can run the complex models in parallel, improving performance. Simics makes parallel SystemC simulation possible (using multiple-kernel model)

*Khan et al. "Multi-Threaded Simics SystemC Virtual Platform", ICCAD 2015*

Simics simulation cell



# Questions?

You can always reach us later:

jakob.engblom@intel.com

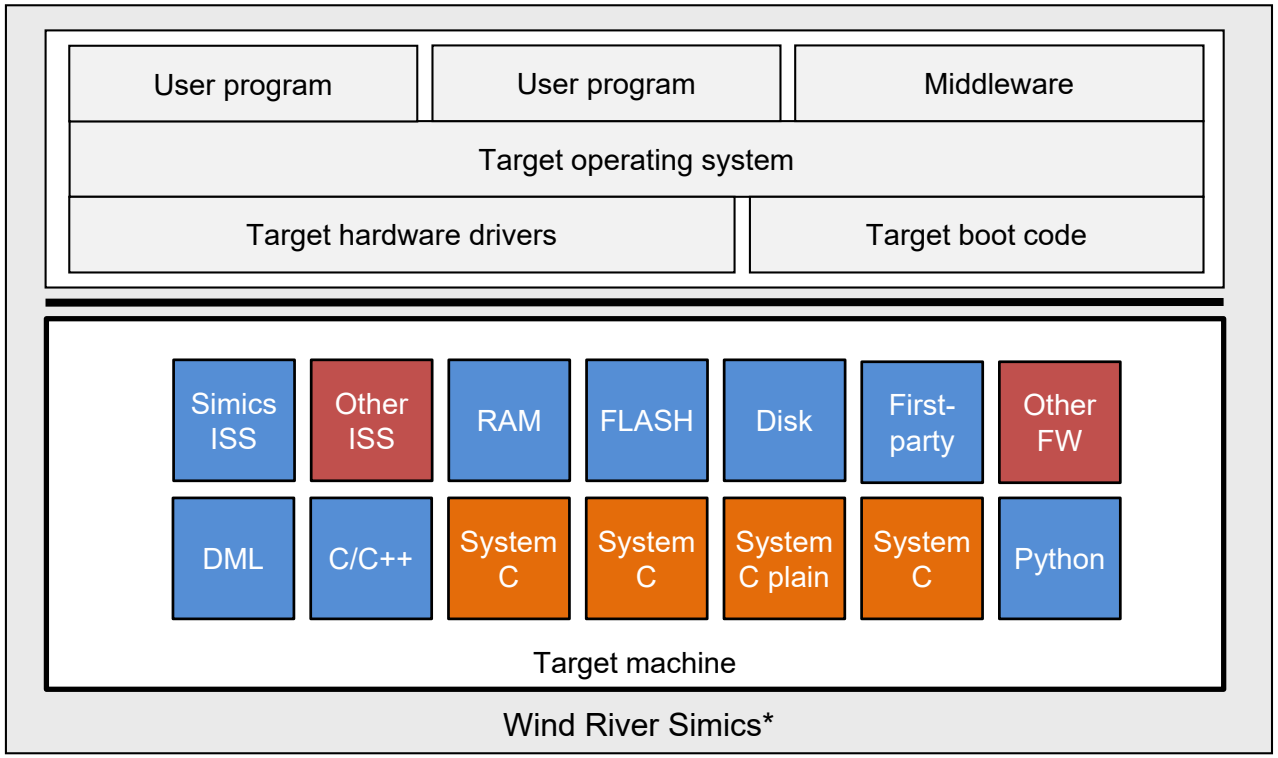
hakan.zeffer@intel.com

accellera SYSTEMS INITIATIVE

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# Thank You!

# BACKUP


User program
User program
Middleware
Target operating system
Target hardware drivers
Target boot code
Simics ISS
Other ISS
RAM
FLASH
Disk
First-party
Other FW
DML
C/C++
System C
System C
System C plain
System C
Python
Target machine
Wind River Simics*