



Simulating Computer Systems

Jakob Engblom, PhD
Business Development Manager
Virtutech
jakob@virtutech.com

- Simulation: a way to study the world
- Build a **model** of a system
- **Try** scenarios on this model
 - **Experimental**, not analytical approach
- Understand the real system from the model



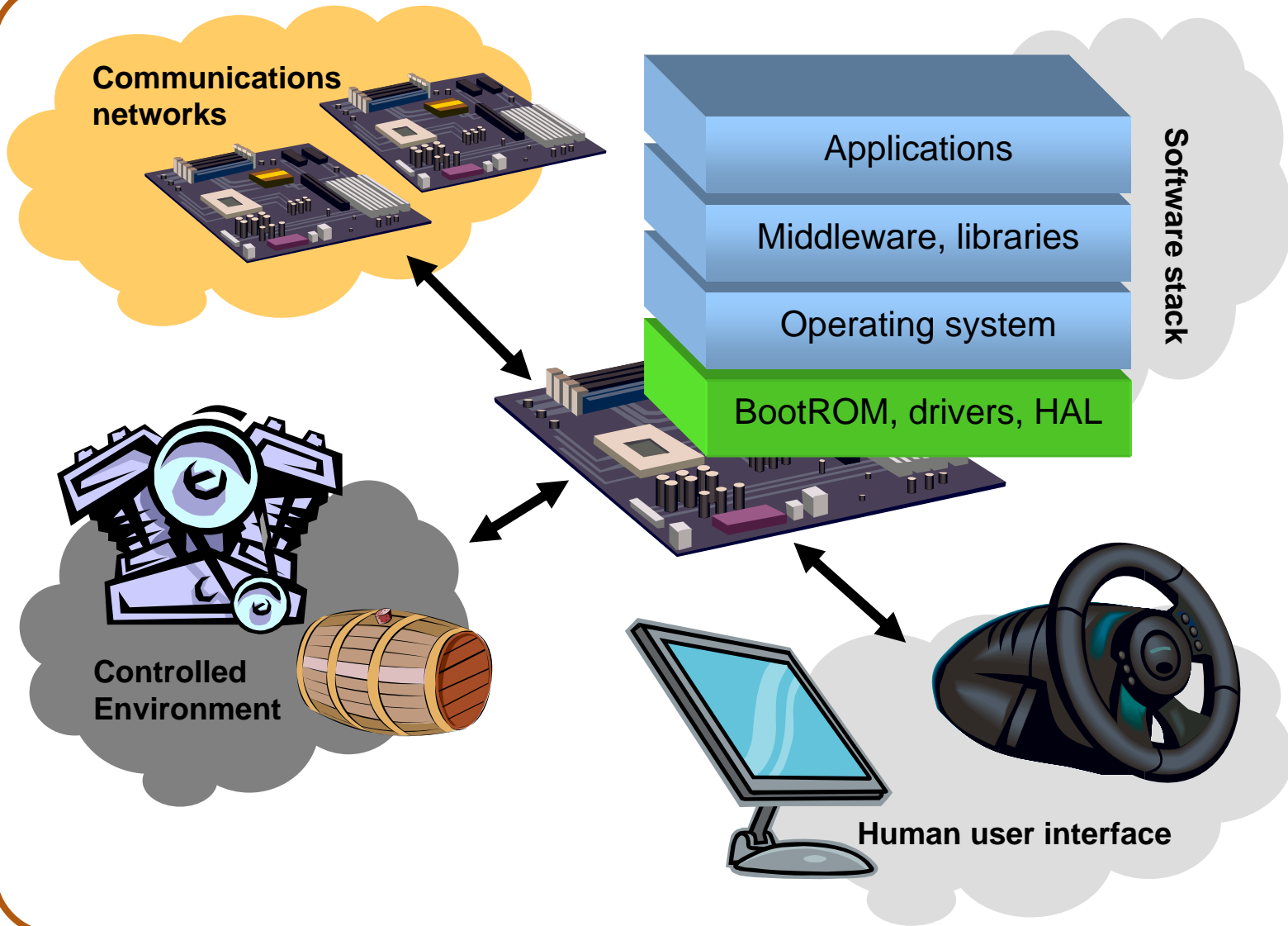
Simulation Advantages

- “Just software”
- Availability
 - Easy to copy & distribute
 - Requires no special lab
 - Good for global reach
- Flexibility
 - A computer can be “any” system, no fixed lab setup
- Turn-around time
 - Virtual car crashes
 - Virtual prototypes rather than physical prototypes
- Inspectability
 - Any variable or property can be observed, even if hidden in the real world
- Controllability
 - Any variable or property can be changed
 - Controlled experiments, not real-world random
- Configurability
 - Easy to change configuration and create new configurations
 - Easy to vary parameters

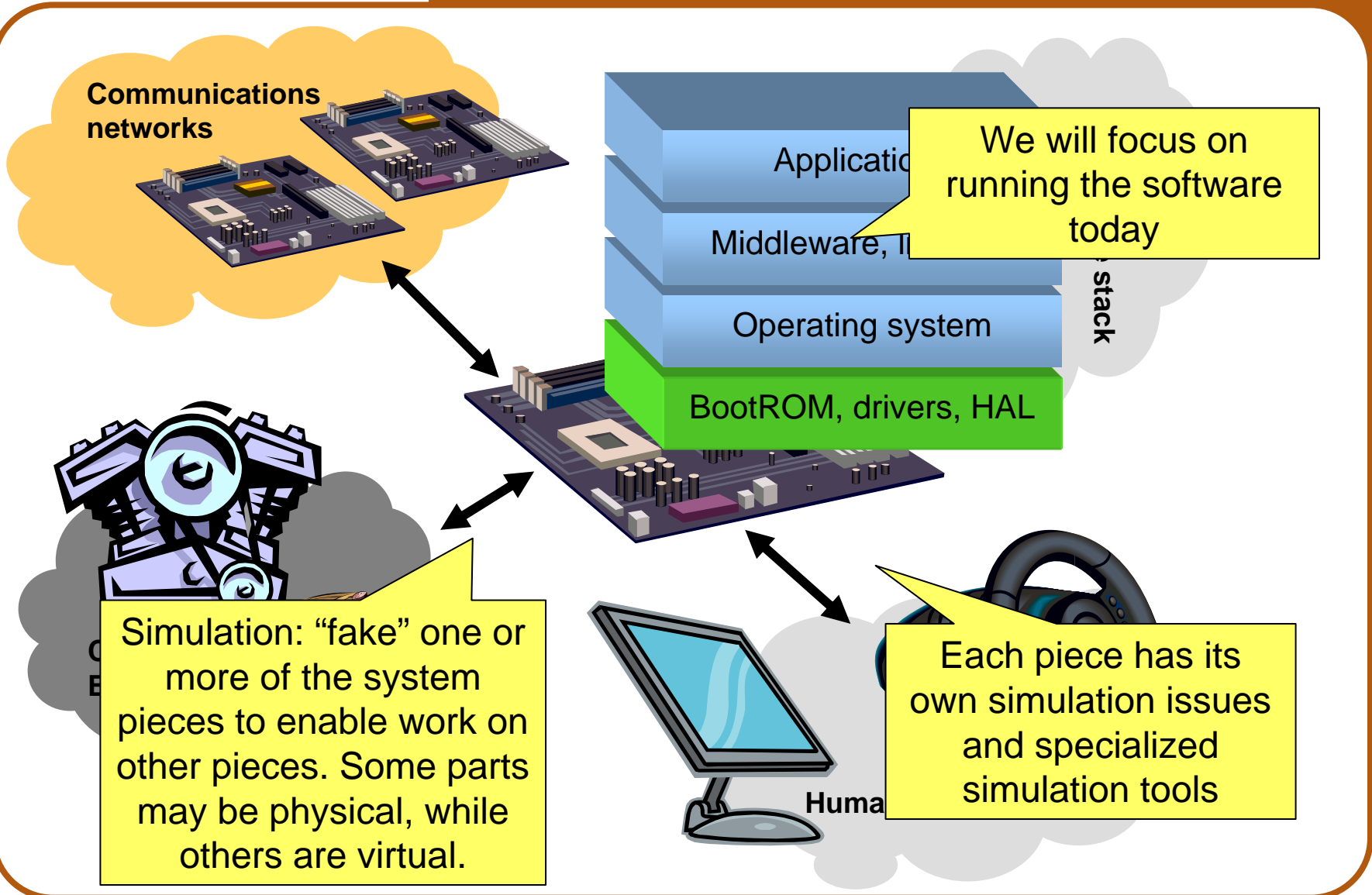


Simulating Computer Systems

Embedded Computer System



Simulating Embedded Computer System

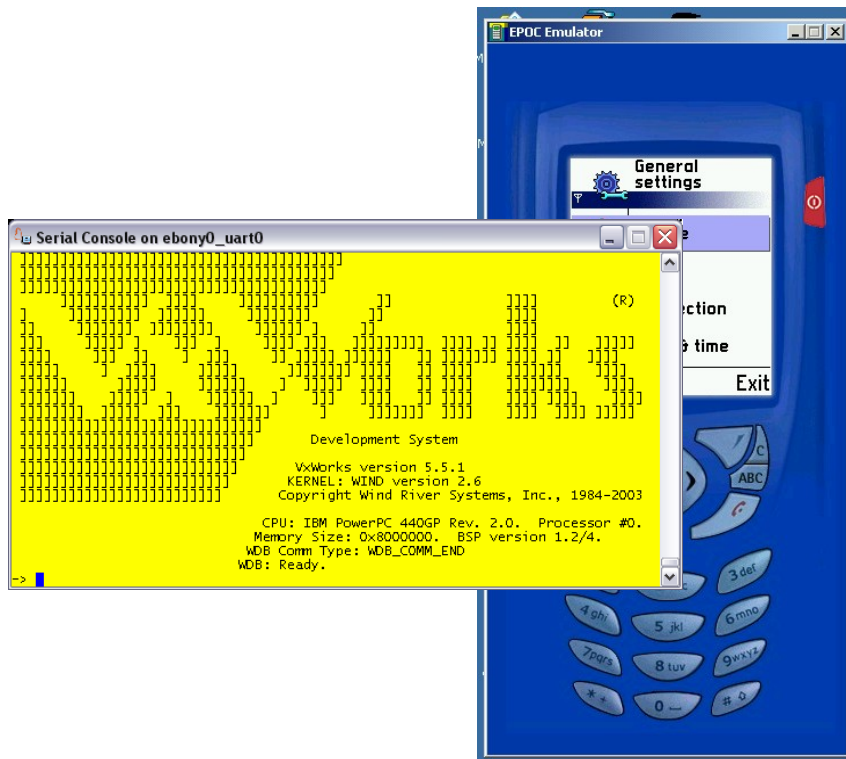


Why? Avoiding Hardware

- Some simulation work is motivated by the clear advantages of simulation over real hardware
 - Control, visibility, access
 - No damage to physical objects (testing a tank, for example)
 - Faster turn-around times for experiments
 - Early availability compresses development schedules
- Much simulation work is motivated by the necessity of avoiding physical hardware – even when hardware is considered the ideal solution
 - Avoid inconvenience
 - Save costs
 - Get around slipping hardware schedules
 - Insufficient number of real systems available

User Interface Simulation

- A category of tools of its own
- Part of many other simulation tools



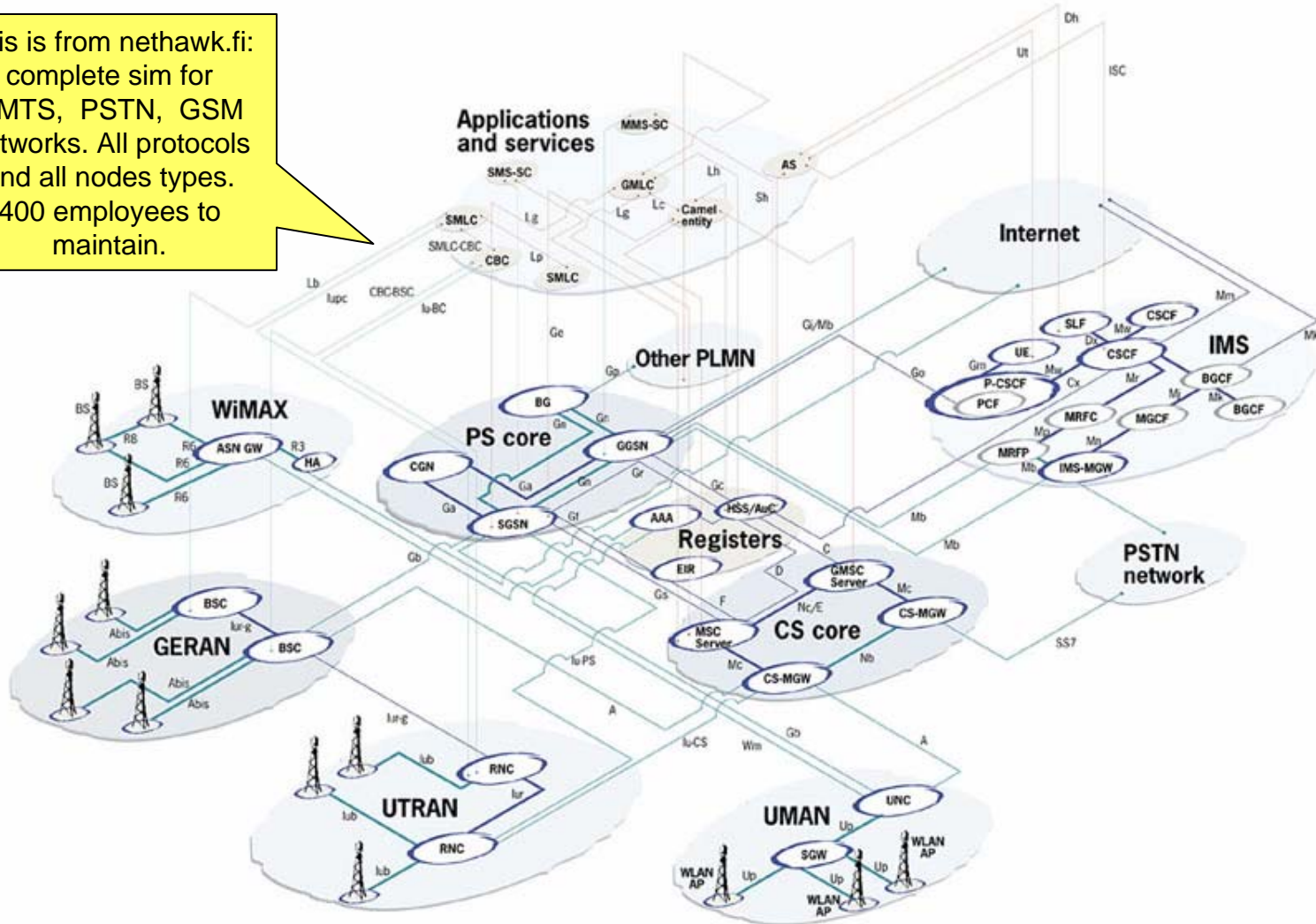
- Many different levels:
 - Virtual serial ports
 - Virtual graphical screen & mouse like VmWare
 - Clickable simulation of touch screens
 - Clickable panels of buttons
 - LED panels
 - Full hardware mockups connected over CAN bus to PC
- Software:
 - Simulated by scripts
 - Special code for special API
 - Actual target code in some form of other simulator

- Very large field in its own right, largely independent of computer systems
- Largest players:
 - Finite Element Methods
 - MatLab/Simulink
 - LabView/Matrxix
- Used in all industries
- History of 50 years
- Commonly used for control algorithm development
- Key part of the model-driven architecture/model-driven design paradigm in automotive

- Connections between abstracted nodes, to study communication patterns
 - Contains models of node behavior, no actual code
 - ns2
- “Rest of network simulation” to provide environment for a single node
 - Understands protocols
 - Vector CANOE
 - Nethawk tools
- Dumb traffic generation
- Move packets between simulated nodes
 - No understanding of protocols
 - Runs real code on all nodes
 - Virtutech Simics
- Integrate physical and simulated nodes
 - “Hardware in the loop”
- Networks:
 - Ethernet, AFDX, CAN, LIN, FlexRay, MOST, PCIe, I2C, LonWorks, ARINC 429, MIL-STD-1553, serial, RapidIO, VME, SpaceWire, USB, FireWire, ...

Rest-of-Network Example

This is from nethawk.fi:
complete sim for
UMTS, PSTN, GSM
networks. All protocols
and all nodes types.
400 employees to
maintain.





Simulating a Single Computer

System Simulation Use Cases

- **Processor Design**
 - Detailed models of processor pipeline, caches
 - Determine architecture of next-generation processors
 - Classic field of comp sim
- **Running benchmarks** is the primary goal
 - Needs system-level to handle modern workloads
 - Multithreading on multicore processors
 - Many techniques to increase speed to increase test breadth
 - Can get away with simplified system model outside processor
- **System-on-Chip Design**
 - Combining processors, accelerators, devices
 - Processors “fixed”
 - Architecture exploration
 - Sizing, performance, optimization of hardware
 - Focus on hardware designer needs
- **Fidelity to target** is primary driver for models
 - Timing
 - Bandwidth
 - Latency
 - Bus structure
- All components in the architectures are equals

- **Hardware Verification**
 - Devices in an SoC
 - Alternative to fixed tests
 - Use simulation to derive test cases and determine correct results
 - Run system simulation alongside VHDL/Verilog simulation of actual implementation
- **Test-Case Coverage and Realism** are drivers
 - Test cases generated should cover real-world usage scenarios
 - Has to use quite a large system to be true to real-world behavior
- **Timing-Sensitive Software Development**
 - Depends on detailed hardware timing
 - Small codes
 - Close to hardware
 - Optimized DSP kernels
- **Execution time accuracy** is the primary driver for model
 - Detailed timing model from processor and outwards
 - Execution speed secondary
 - Insight into performance characteristics

System Simulation Use Cases

- **General Software Development**
 - Focus on general software developer needs
 - Execute large workloads (OS, applications)
 - Debug & inspection features
- **Speed of execution with fidelity** is the primary driver for model
 - Abstract as far as possible
 - Approximate timing
 - JIT-compiling processor models
- Clear difference between processors and devices in the system, processors dominate execution time
- **Production Use**
 - Use simulated/virtual machines to run software in production
 - VmWare, Xen, Parallels, etc.
 - Mac PPC emulation
 - Little attempt so support debug
- **Raw speed of execution** is the key driver for simulator design
 - Usually no need to model a particular hardware, just something that runs the right OS and application
 - Typically not cross-target
 - Pseudo-devices acceptable

- The level of detail and what is being modeled varies greatly across use cases
- Different roles have different needs

Enough Detail is Enough

- Computer games masters of sufficient abstraction
- "Life-like" action:
 - Momentum
 - Friction
 - Steering
 - Engine torque
- But: not nuts & bolts of cars
- **Simulate only what is observable and relevant for the intended user**



Grand-Prix Legends

Simulation Detail Levels

- SystemC defines five levels of model abstraction
good map of abstraction levels for system mod

Note: production use is not on this map

	Meaning	Style	Comments
AL	<i>Algorithmic Level</i>	<i>UML, SDL, Matlab models</i>	<i>Not concrete enough to run software</i>
PV	Programmer's View	Transaction-level with simple time	100+ MIPS, software development
PVT	Programmer's View, with Timing	Transactions with precise time	10+ MIPS, Low-level software devel, some SoC architecting
CC	Cycle-Callable	Bit-level with precise timing	<1 MIPS, Hardware validation, comp.arch
RTL	Implementation	VHDL/Verilog implementation	<1 MIPS, Hardware validation



Simulating for Software Development

“Virtual Software Development”

Where are we?

- In “Simulation for General Software Development”
- For embedded systems mainly

Electronics Is Software

Electronics is software. Shipping a system is largely about identifying and removing defects from the software and keeping them from creeping back in as the product evolves.

St. Jude Medical Cardiac Defibrillators

St. Jude Medical Inc., a Canada-based medical technology company, announced in June 2005 that some of its implantable defibrillator models, or ICDs, have a software problem that could cause the heart-shocking device to malfunction.

Los Angeles Times
latimes.com
2:43 PM PDT, October 13, 2005

Software Glitch Triggers Toyota Prius Recall

By James F. Peltz, Times Staff Writer

In what's believed to be the first recall of hybrid cars for engine-related problems, Toyota Motor Corp. said today that it is notifying about 75,000 owners of its hot-selling Prius about a potential software glitch that could cause the car to stall or shut down.

The voluntary recall dented the good reliability record of the Prius, whose sales have jumped in the past two years as drivers sought better fuel economy in the face of soaring gasoline prices.

The problem involves the hybrid's computer software, rather than mechanical parts, and it first came to light in May when the National Highway Traffic Safety

The Explosion of the Ariane 5

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing \$7 billion. The destroyed rocket and its cargo were valued at \$500 million. A board of inquiry investigated the causes of the explosion and in two weeks issued a report. It turned out that the cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,767, the integer storable in a 16 bit signed integer, and thus the conversion failed.



The Register

Software glitch blamed for CryoSat loss

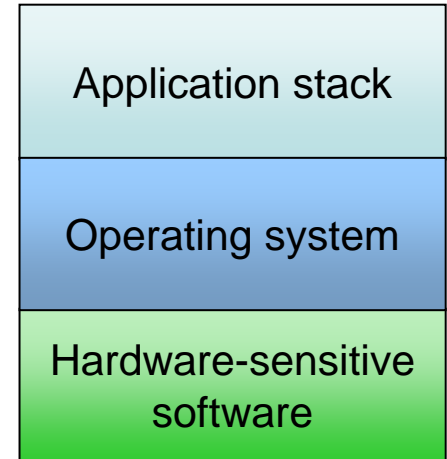
Rocket still ok to fly again
By Lucy Sherriff
Published Thursday 27th October 2005 14:30 GMT
Get breaking Reg news straight to your desktop - click here to find out how

Officials investigating the loss of the CryoSat mission have revealed that a software glitch in the on board flight control system on the new, upper stage of the rocket was to blame.

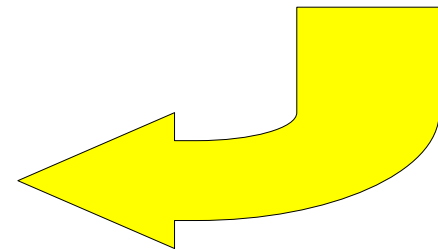
There is no fault with the Rockot launcher itself, Russian officials said, which means it has been cleared for future flights: the BBC reports

Traditional Software Development

- Software development methodology creates production binary
- Production binary runs on the real hardware

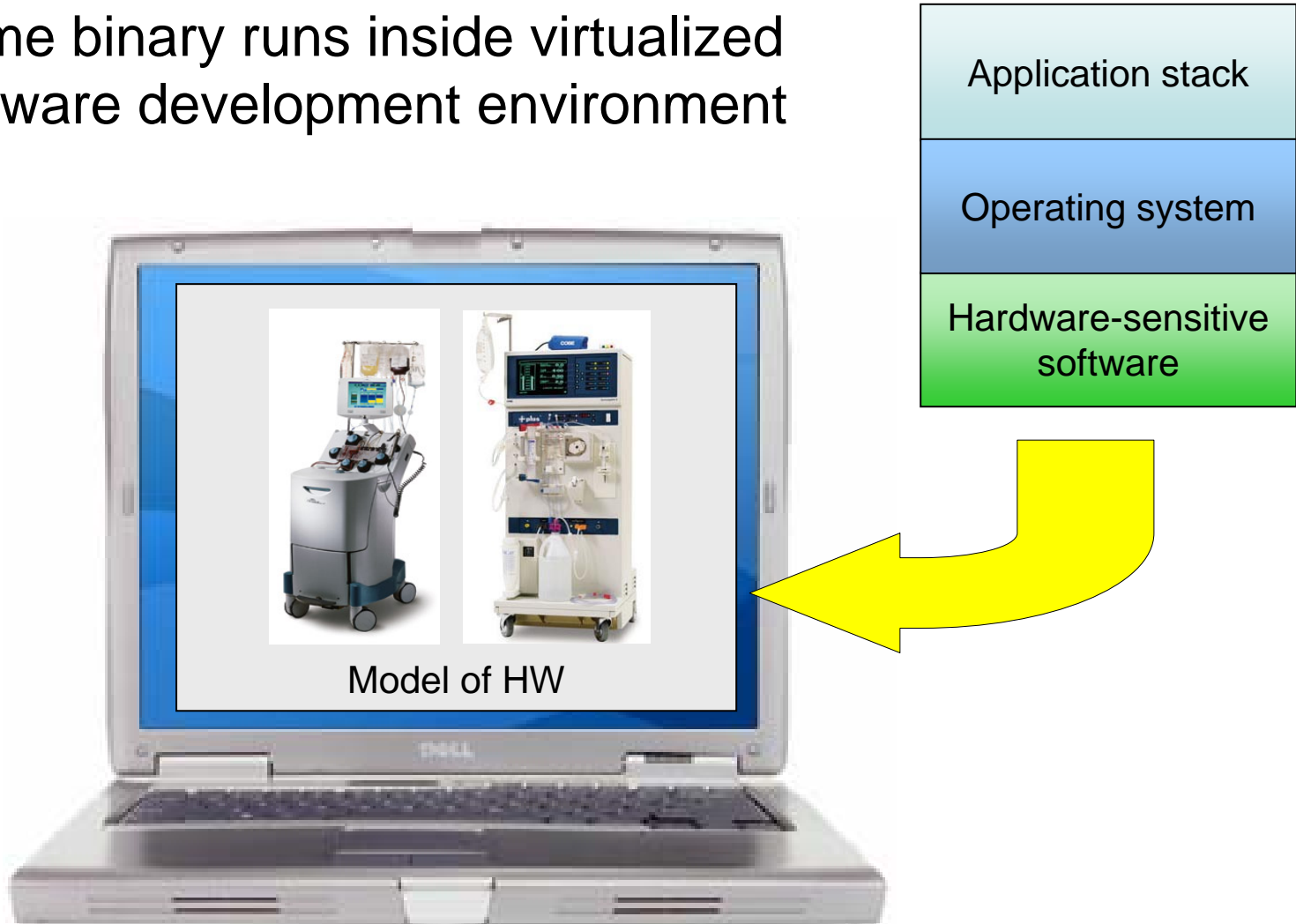


Actual hardware



Virtualized Software Development

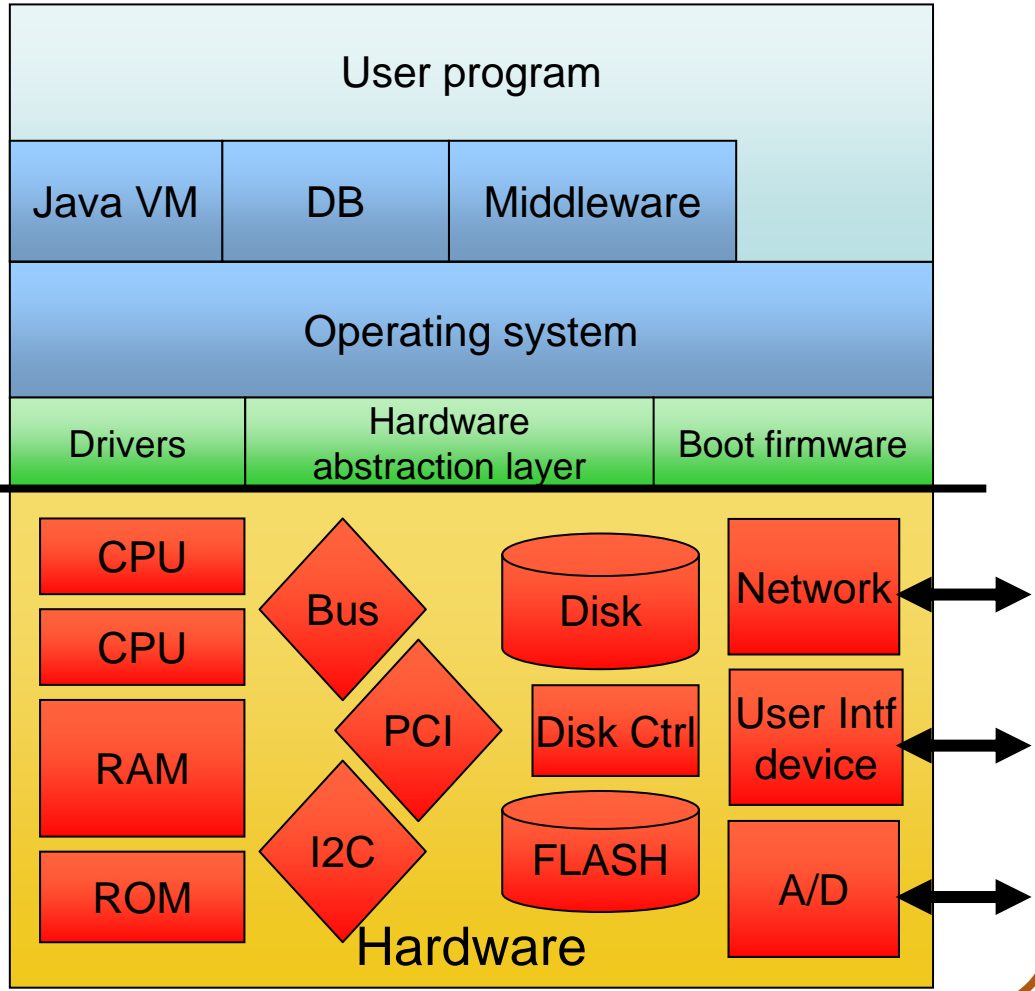
- Same binary runs inside virtualized software development environment



Simulating the Software Stack

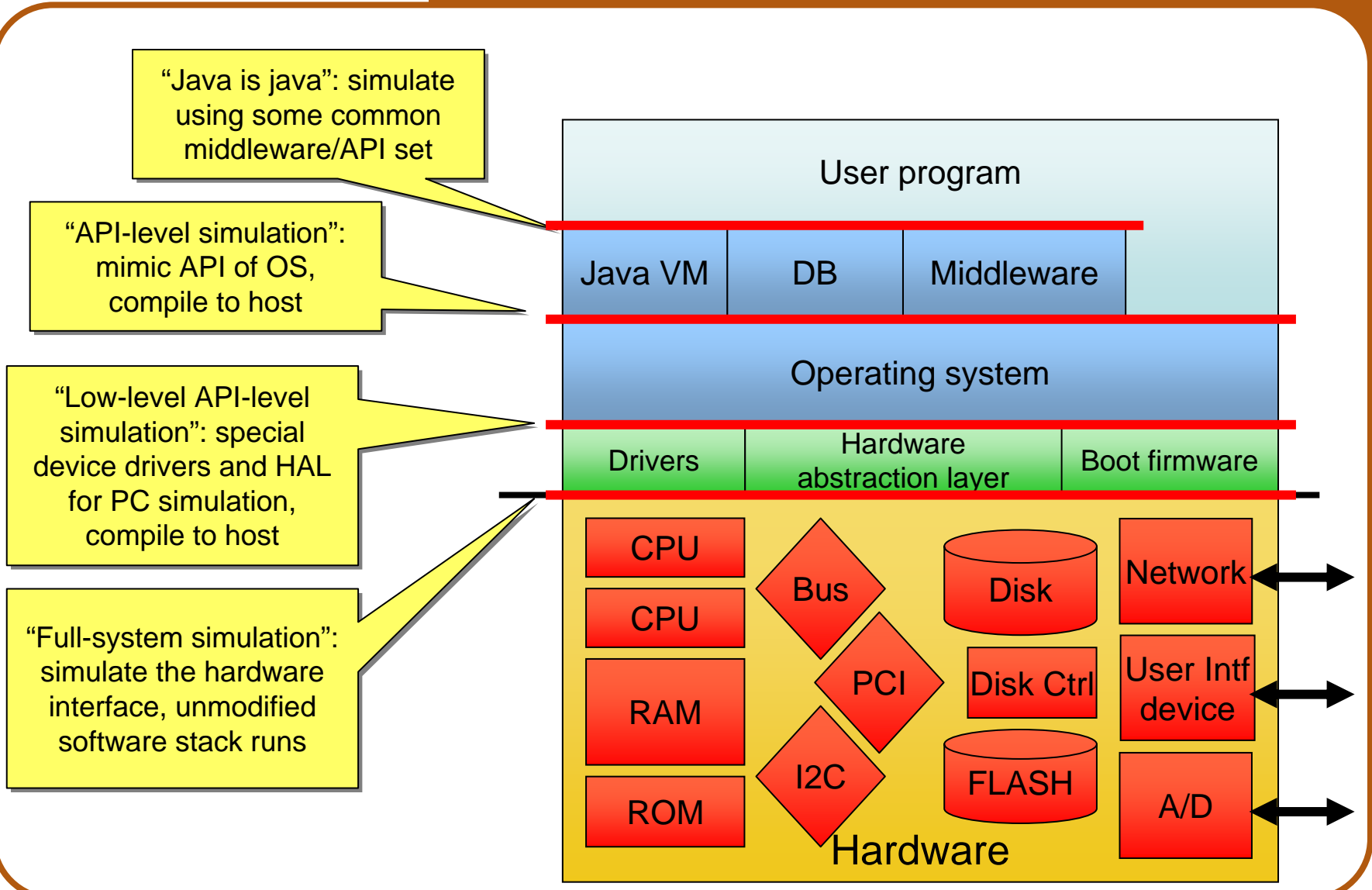
Goal: run your embedded software on a PC instead of on a physical target

HW/SW interface

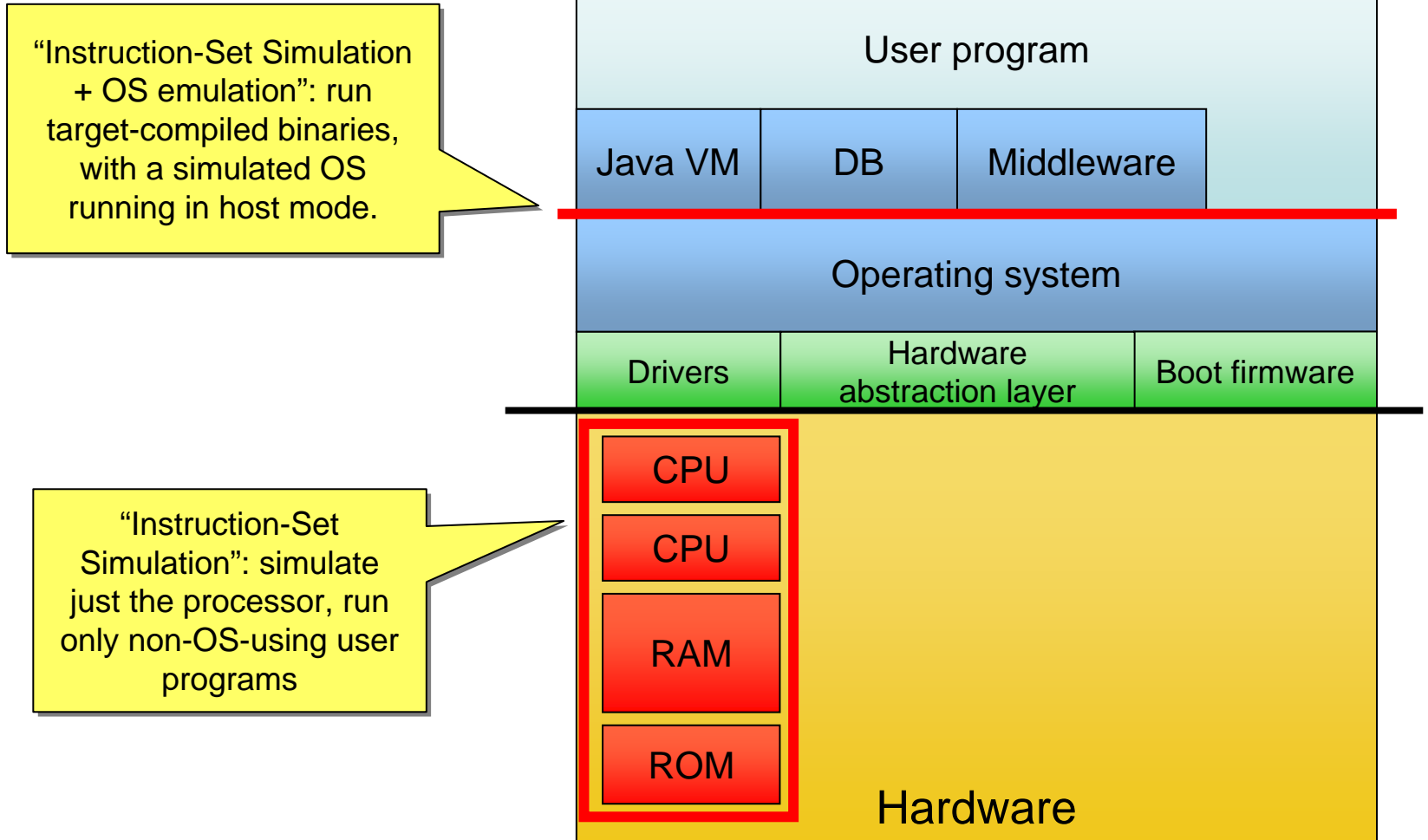


- **Host-compiled or target-compiled?**
 - Host: Compiled to x86 to run on PC, for simulation only
 - Target: Compiled to actual target code
- **Software stack completeness**
 - Which pieces of the software stack are the same as on target, which are faked-out or stubbed?
 - Which types of code can be tested in simulation?
- **OS Behavior**
 - Same scheduling as real target?
 - Same memory protection?
 - Same memory allocation and limitations?

Simulating the Software Stack

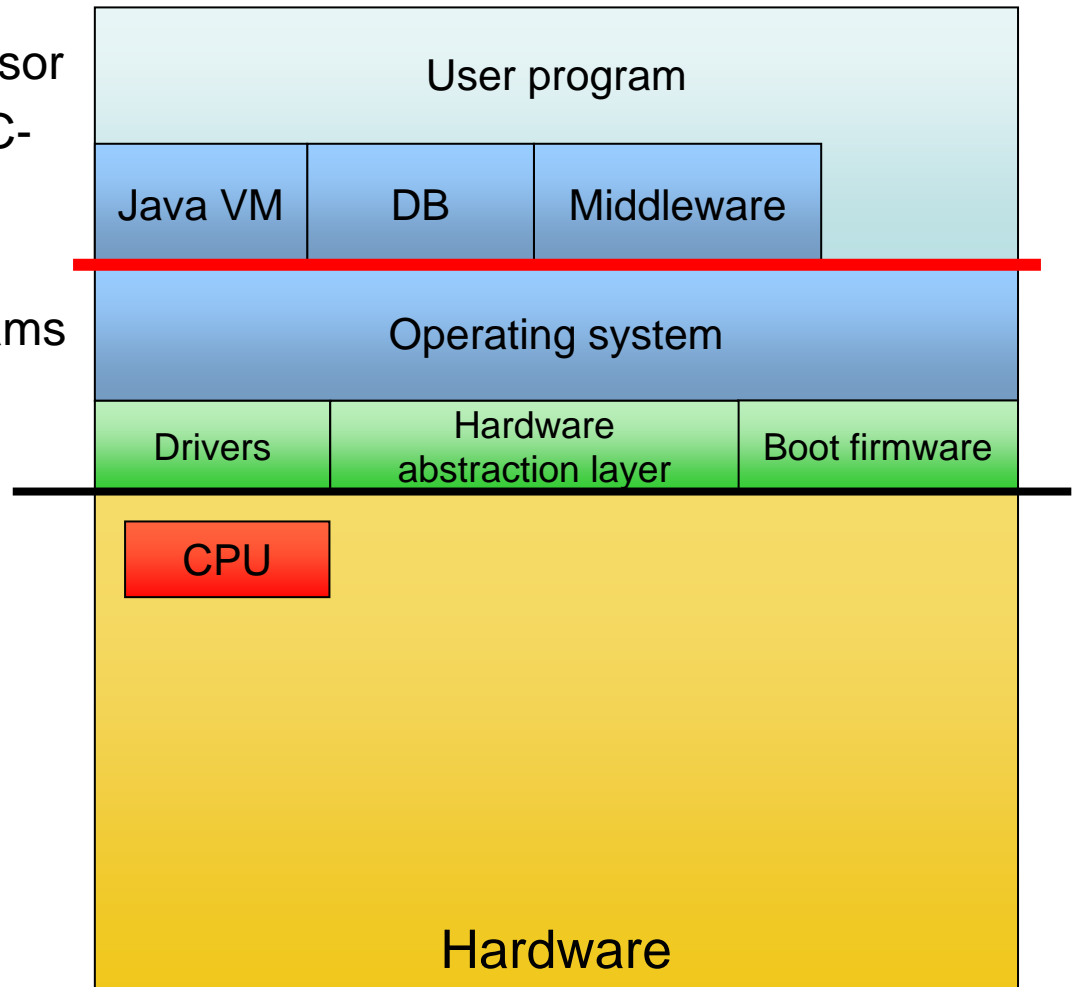


ISS-OS Hybrid



Emulation

- Goal: run software for the same OS but a different target processor
- Example: MacOS PPC->x86 migration
- OS runs natively
- Only user-level programs and libraries affected

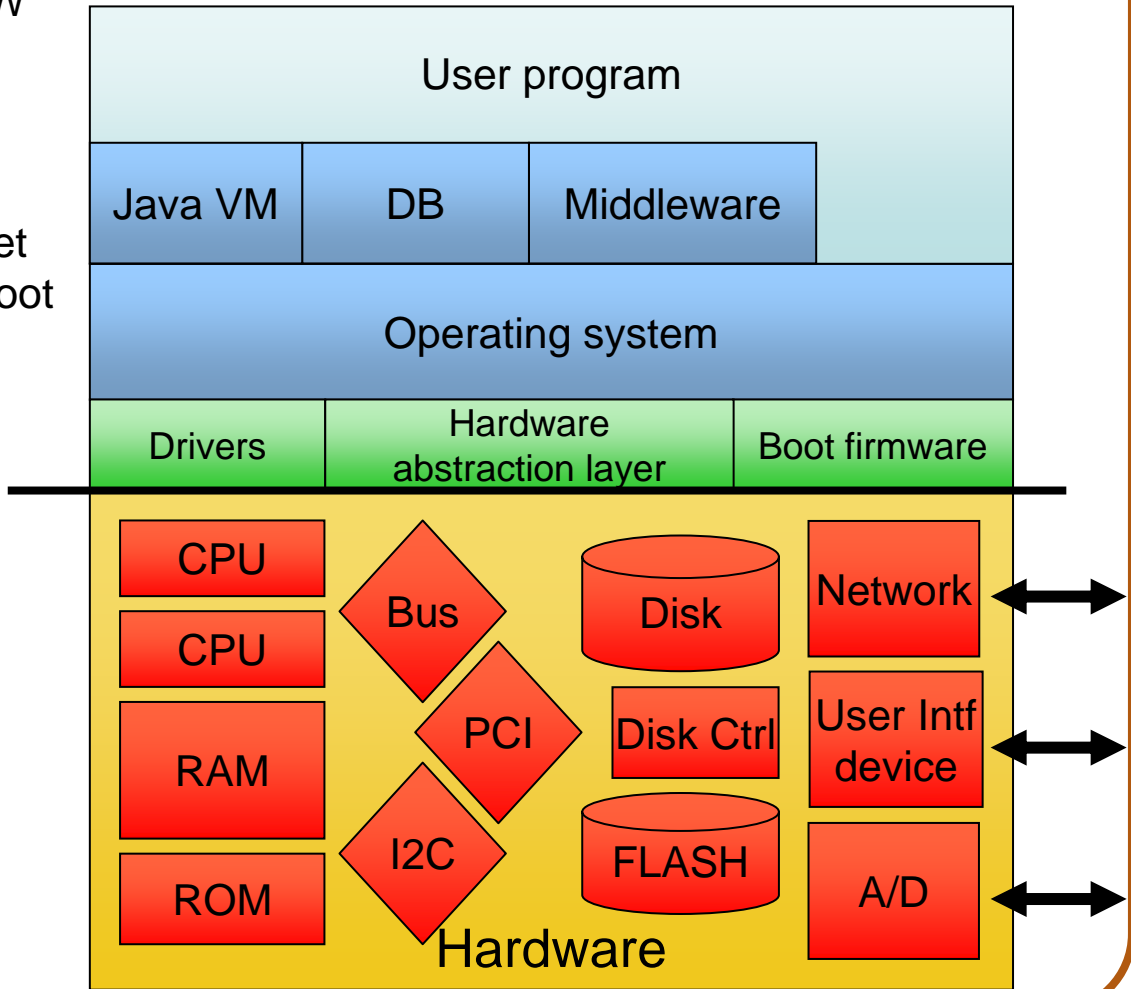




Virtual Software Development Benefits

Virtualized Software Development

- Simulate the hardware
 - We cut at the HW/SW interface
- Run the real software
 - Same binary as target
 - Same OS, drivers, boot process as target
 - Same compiler
 - Same build setup
 - “One-track development”
- This is what we do at Virtutech with Simics

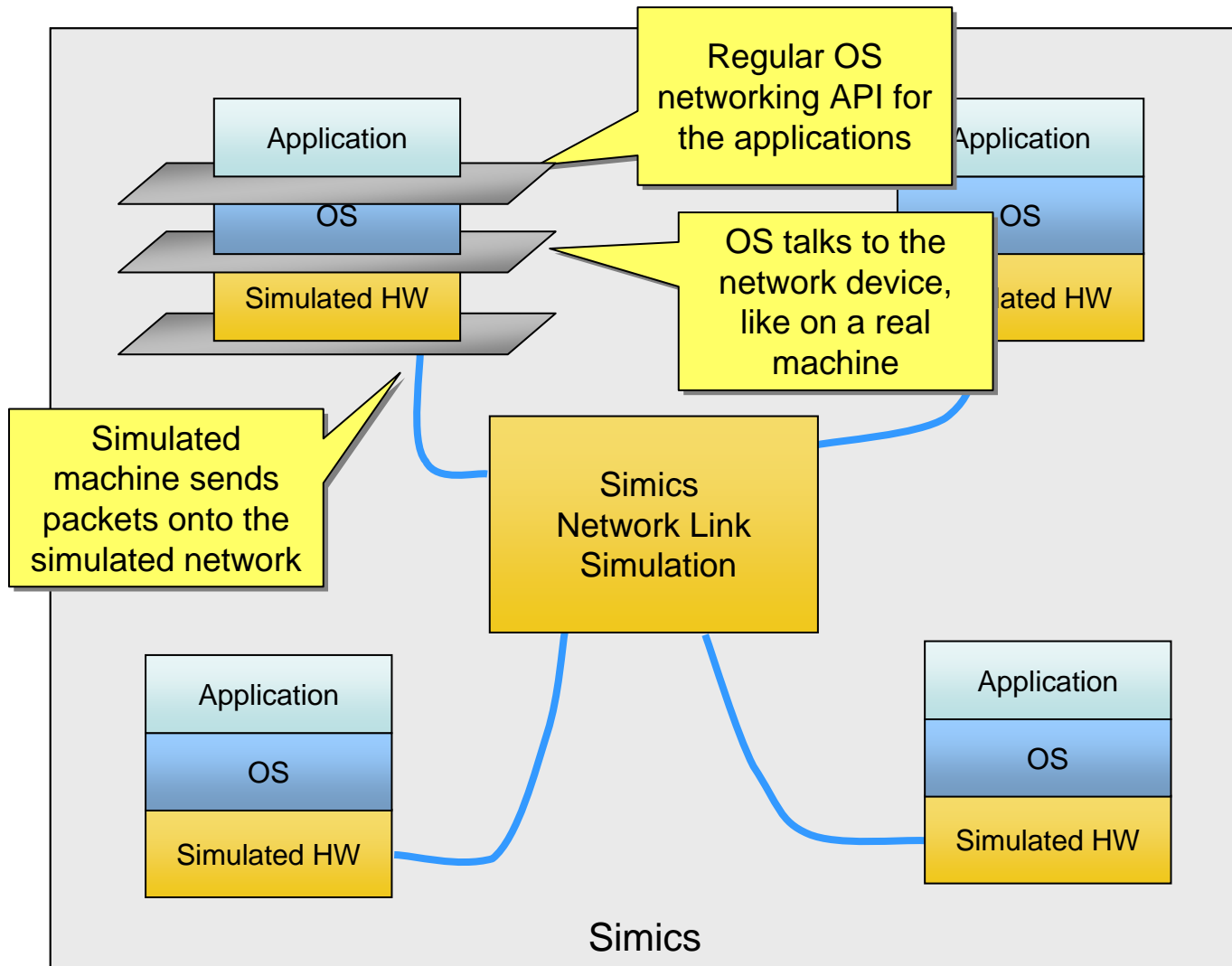


The Scope of Interesting Systems

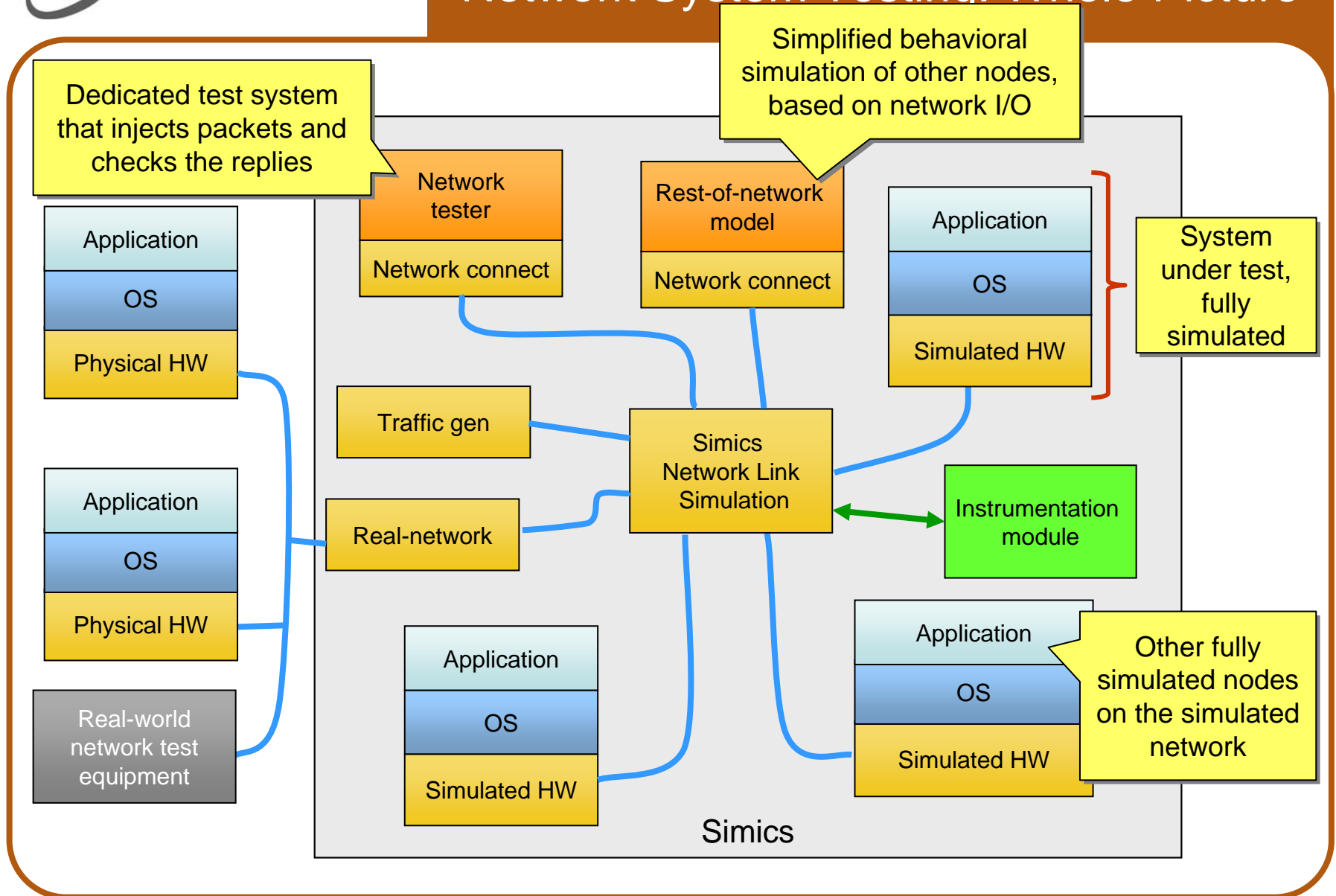
- Interesting systems not just single-CPU stand-alone
- Multiprocessors
 - Homogeneous like servers
 - Heterogeneous like mobile phones
- Distributed systems
 - Local-Area Networks
 - Embedded Systems
 - Network-on-Chip
- Need to simulate
 - Shared memory computers
 - Multiple computers
 - Networks of computers
 - Computers in their environment



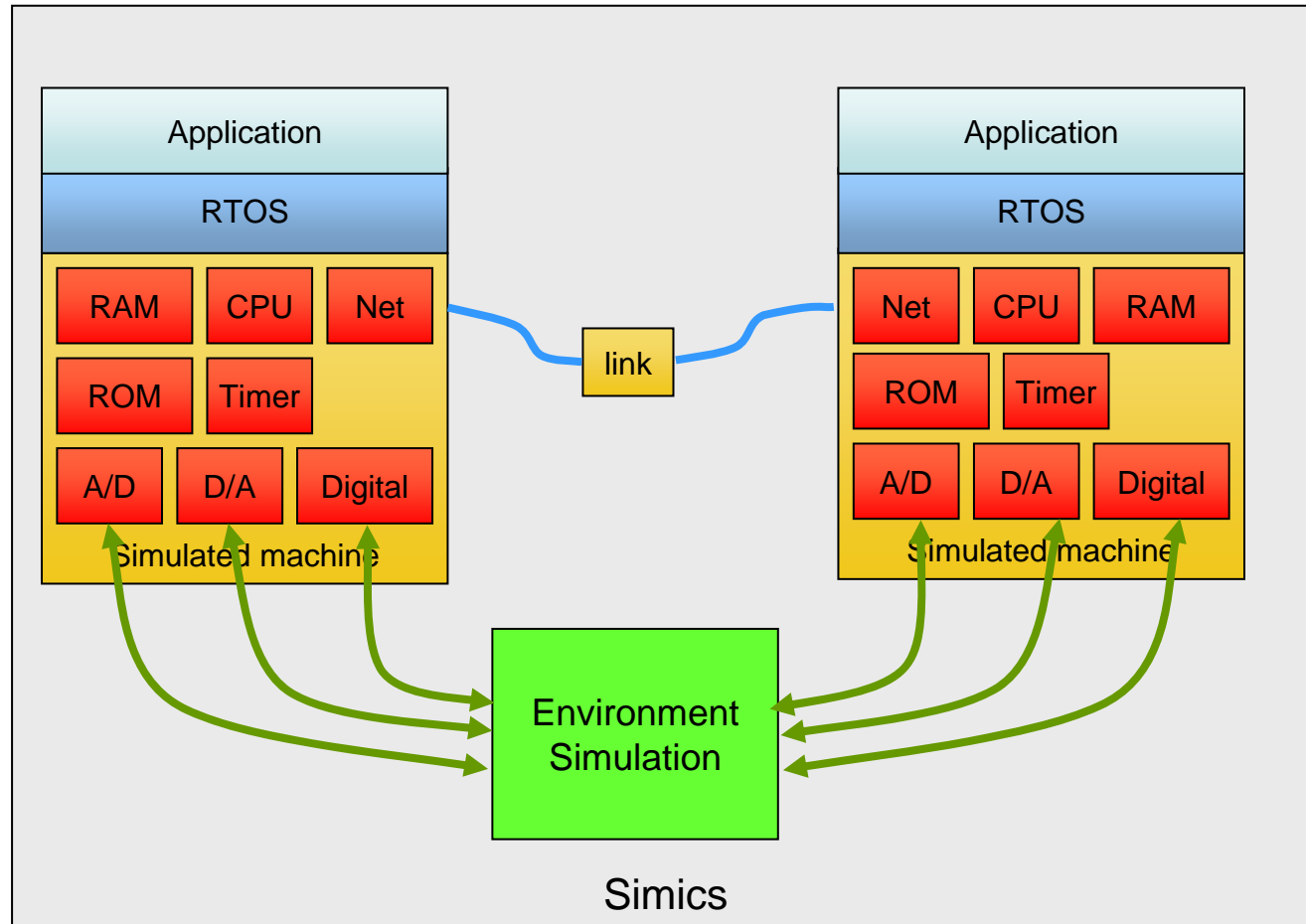
Network Simulation



Network System Testing: Whole Picture



Simulating Computers in an Environment



Demo: Networked Systems



- Checkpointing
 - Store current state; pick up and continue later
 - Position workload once, use many times
 - Spread a system state to multiple developers
 - Can checkpoint an entire network of machines
- Determinism
 - Same initial state gives same execution;
 - Repeat the same execution any number of times
 - Investigate a problem time after time
 - For multiprocessor systems & network systems

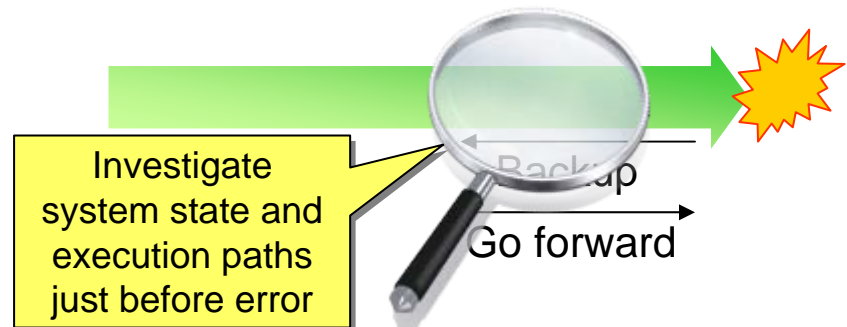
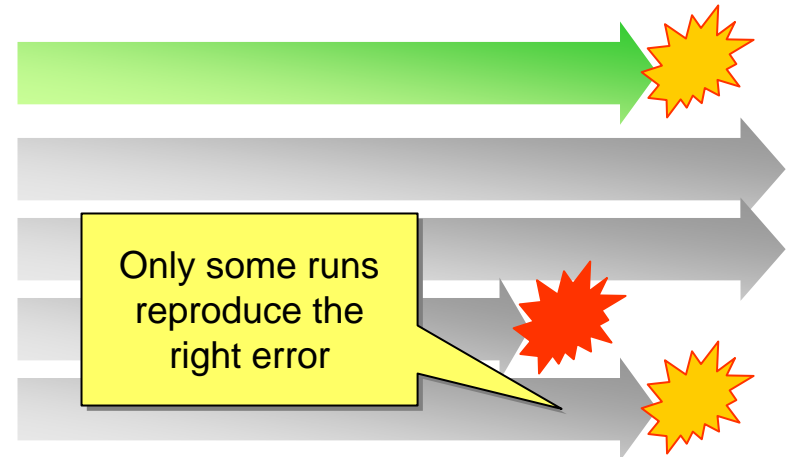
- **Visibility (insight without intrusion)**
 - All state can be observed
 - All events can be traced and logged
- **Controllability**
 - Any part of machine or state can be changed
 - Fault injection
- **Virtual time**
 - Time is completely virtual
 - Global synchronization across all machines in a network
 - Global stop across all processors in a multiprocessor

- **Configurability**
 - Any parameter of system can be changed
- **Sandboxing**
 - Allows investigating "nasty code"
 - Simulated machine complete isolated
 - Network can be isolated
 - Simics undetectable by malware
 - Complete hardware simulation, no virtualization tricks
- **Backwards debugging**
 - Roll back execution to previous state
 - Reverse breakpoints
 - Investigate details of program errors



Reverse Debugging

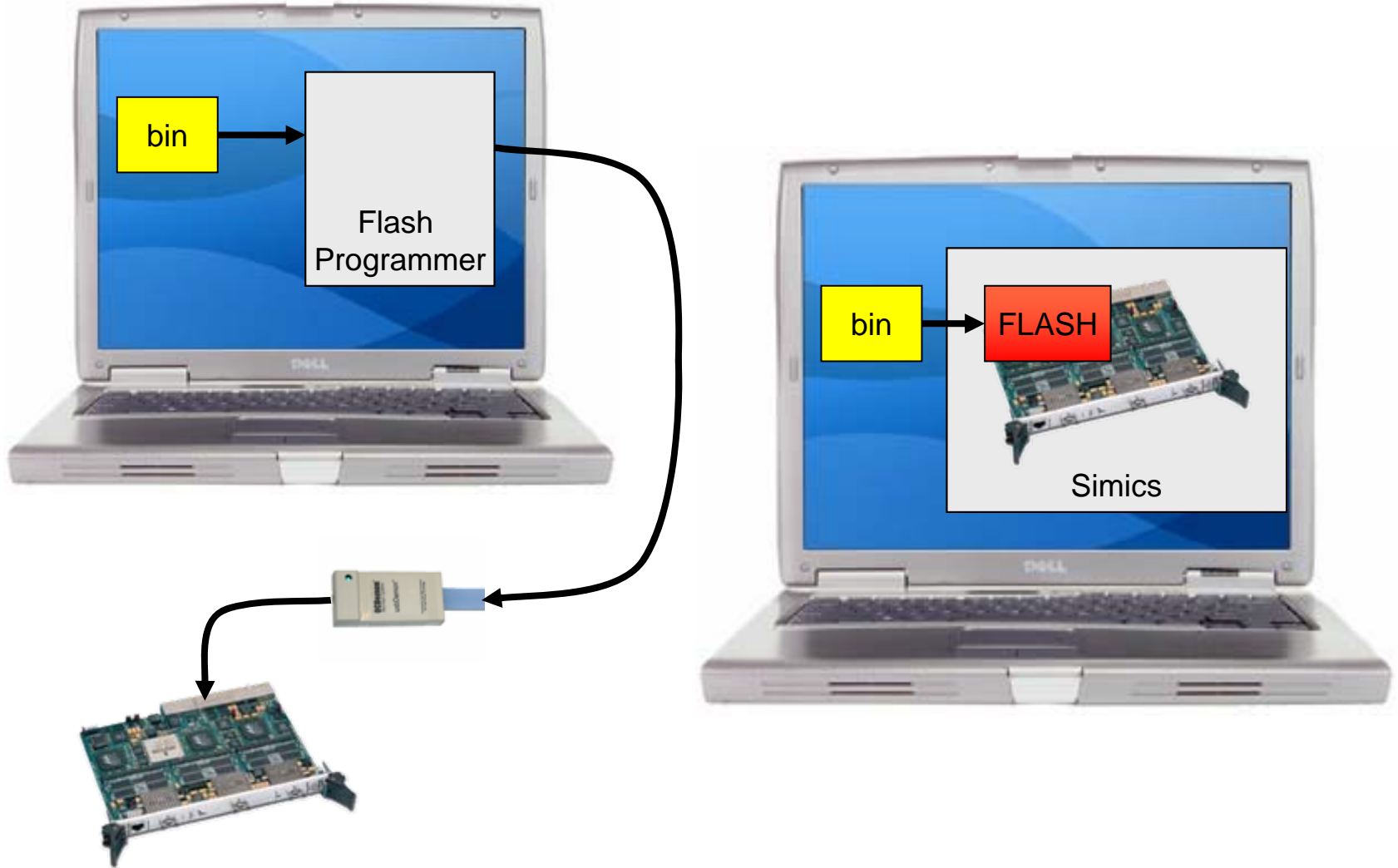
- Stop & go back in time
 - Instead of rerunning program from start
 - No need to rerun and hope for bug to reoccur
 - Investigate exactly what happened this time
 - Breakpoints & watchpoints backwards in time
 - Very powerful for parallel programs
 - Very hard to do for multiple processors using trace on physical hardware



Demo: Backwards in Time



Convenience: Flash programming



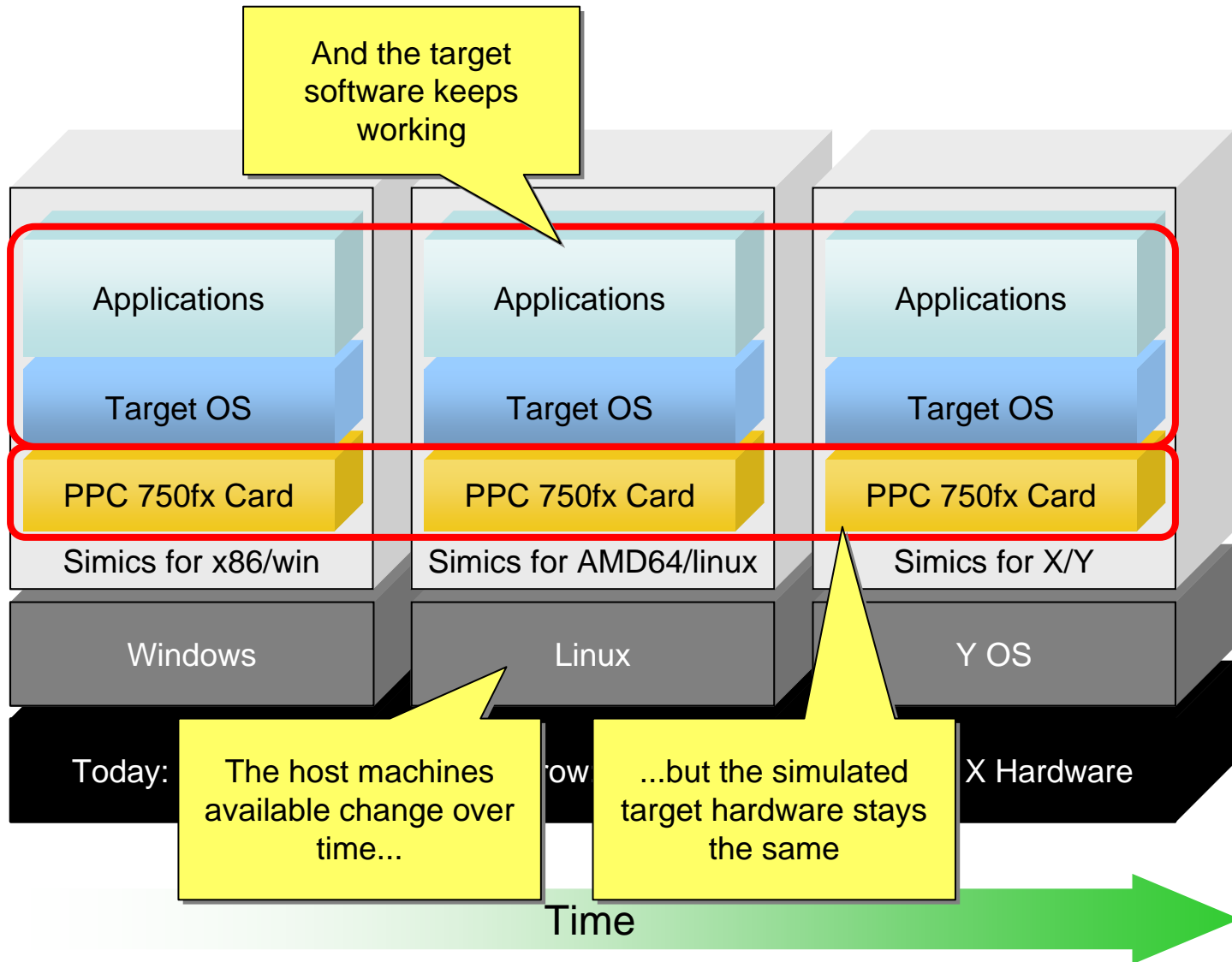
Legacy



Simulating the Legacy System



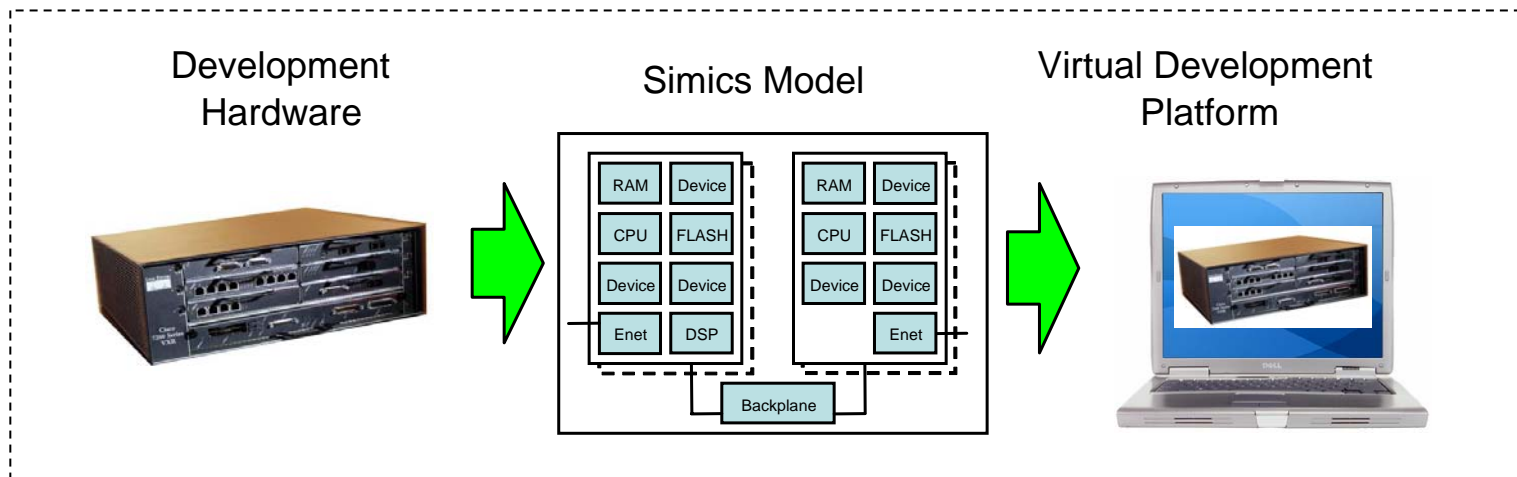
Virtualization = Infinite Longevity





Modeling Hardware for Virtual Software Development

- Prerequisite to obtaining benefits of virtualized software development

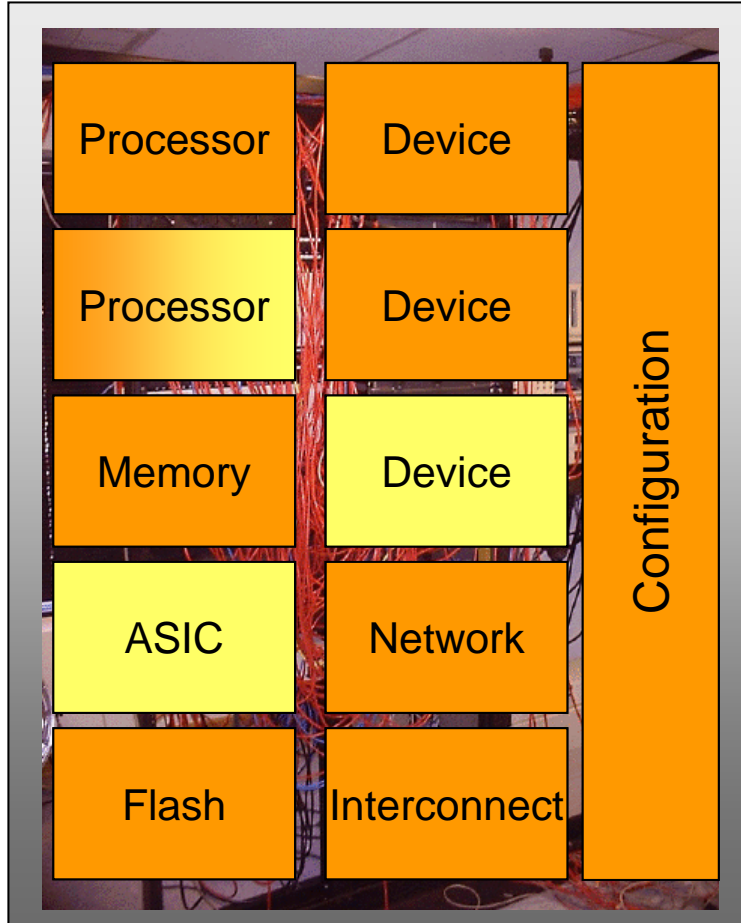


- How do we achieve this?

- Instruction-set simulation (ISS)
- **Complete** and **correct** processor functionality
 - All instructions semantics bit-correct vs real machine
 - Supervisor-mode & user-mode
 - Runs the complete target instruction set
 - Including AltiVec, SSE, 3dNow, VIS, etc. extensions
 - All accessible values represented
 - User-level registers
 - Supervisor-level registers
 - Model-specific registers, ASIs, debug registers, etc.
- Memory-management unit
- Timing abstracted
 - Add details if required

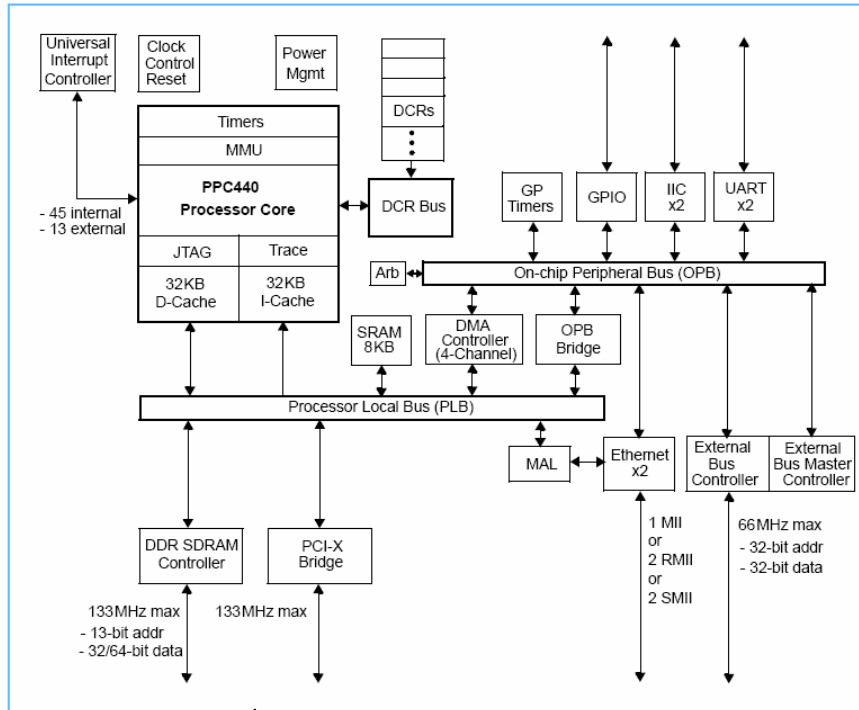
- Hardware modeled as a **set of devices**
 - Memory map of machine (as seen by processor)
 - At the programming register level
- Model the program-visible behavior
 - Configuration registers
 - Control registers
 - Buffers in memory, etc.
- Transaction-level modeling
 - Reads, writes, DMA transfers, network packets
- ASICs & FPGAs
 - Model programming interface behavior
 - Not detailed implementation
 - Unless that is needed, we'll get back to that later
- Detailed timing can be added if required

Modeling Your System

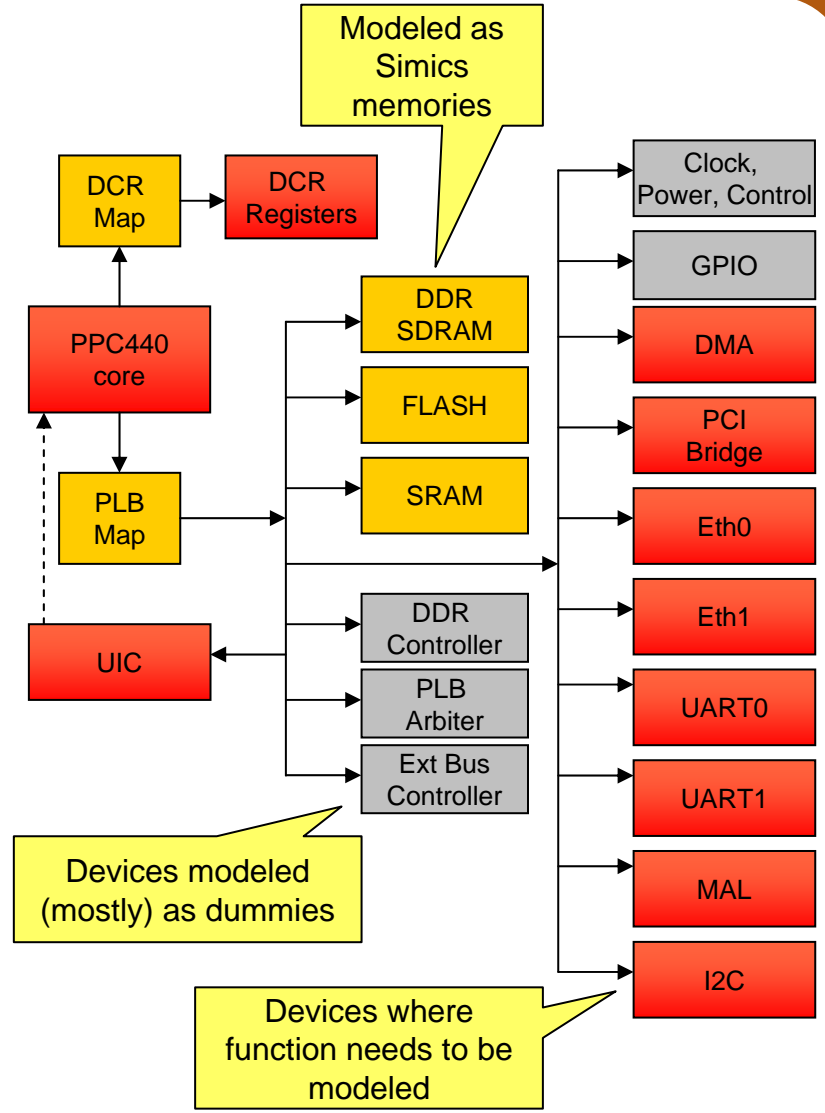


- Use Simics framework
- Reuse VT components
 - Large library available
- Adapt VT components
- Model custom parts
 - DML
 - C, C++
 - Python
- Device modeling by
 - Virtutech
 - Customer
 - Partner
 - Consultant

Mapping a System For Modeling



PPC 440 GP block diagram



Devices modeled (mostly) as dummies

Devices where function needs to be modeled

- Declarative
- Domain-specific for Simics
- Efficient coding
 - 5 times smaller than C
 - Quick start modeling
 - Iterative lazy development
- Fast compiled models
- Models redistributable as binaries
- Modeling time:
 - Days to weeks
 - Depends on model complexity

```
bank b {
  register DMA_control size 4 @ 0x20 {
    field EN [31] "Enable DMA";
    field SWT [30] "Software Trigger";
    field TS [15:0] "Transfer size";
    method after_write(memop) {
      inline $do_dma_transfer();
    }
  }
  register DMA_source size 4 @ 0x24;
  register DMA_dest size 4 @ 0x28;

  method do_dma_transfer() {
    if ($DMA_control.EN==1) {
      local uint16 count = $DMA_control.TS;
      local uint8 local_buf[4];
      local exception_type_t result;

      while(count>0) {
        // copy memory details elided...
        $DMA_source += 4;
        $DMA_dest += 4;
        count -= 1;
      }
      // clear SWT bit, update TS
      $DMA_control.SWT = 0;
      $DMA_control.TS = count;
    } else {
      if($DMA_control.SWT==1) {
        // enable bit not set, so we cannot transfer
        log "info", 2 : "EN bit not set, SWT=1 has
no effect";
      }
    }
  }
  ...
}
```



Simics Uses and Examples

“Simulation is the key to advanced microprocessor development, and Simics is by far the most advanced realization of this technology available”

Kevin Collins, Director Global Firmware Development



“There are things you can do in a simulation environment that you can't in the real world”

Mike Turnlund, Director Software Tools



“With Simics we were able to locate defects in the flight software within hours versus days using traditional hardware in the loop testing”

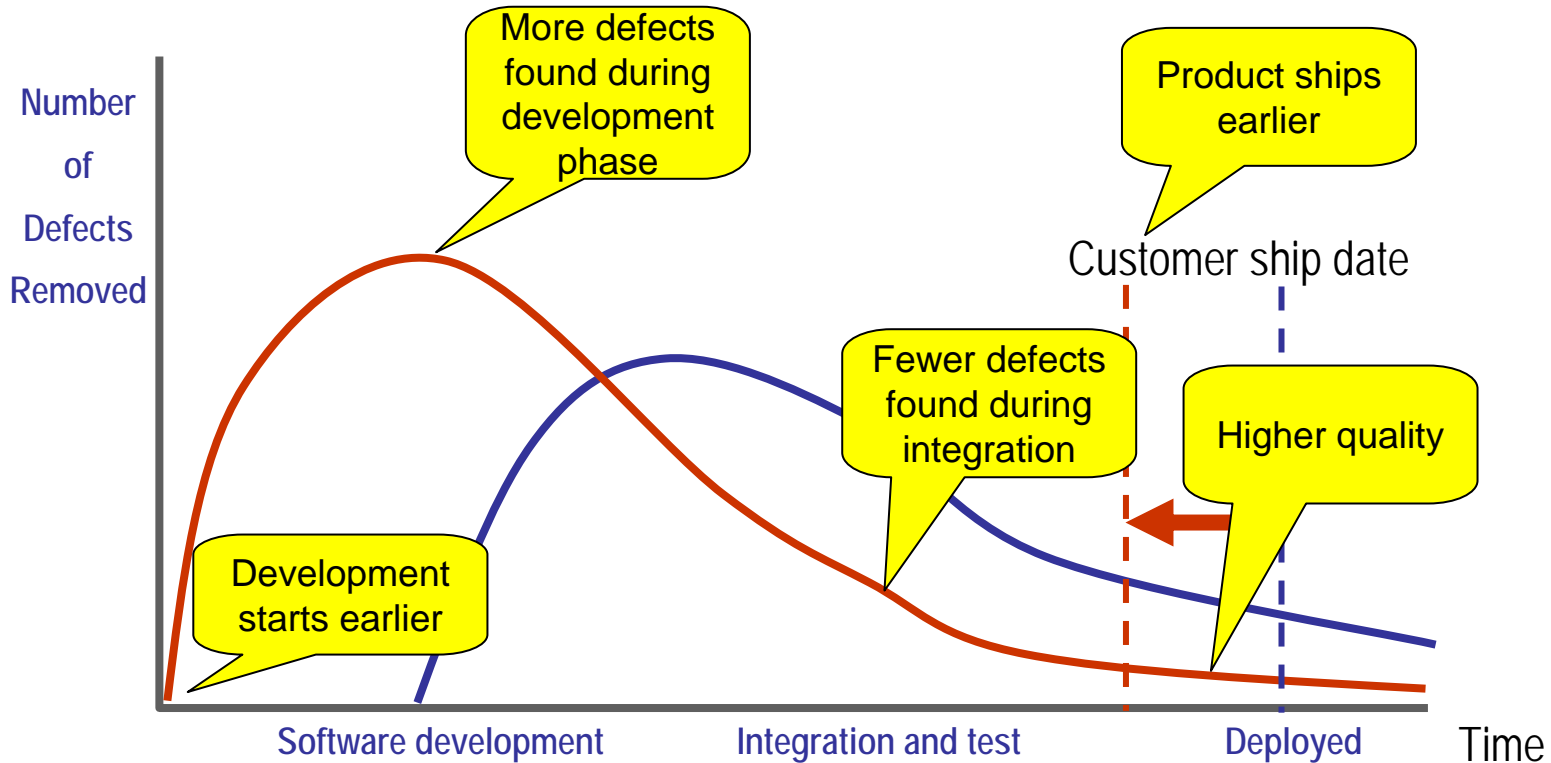
Joe Pizzicaroli, Vice President Satellite and Launch Operations



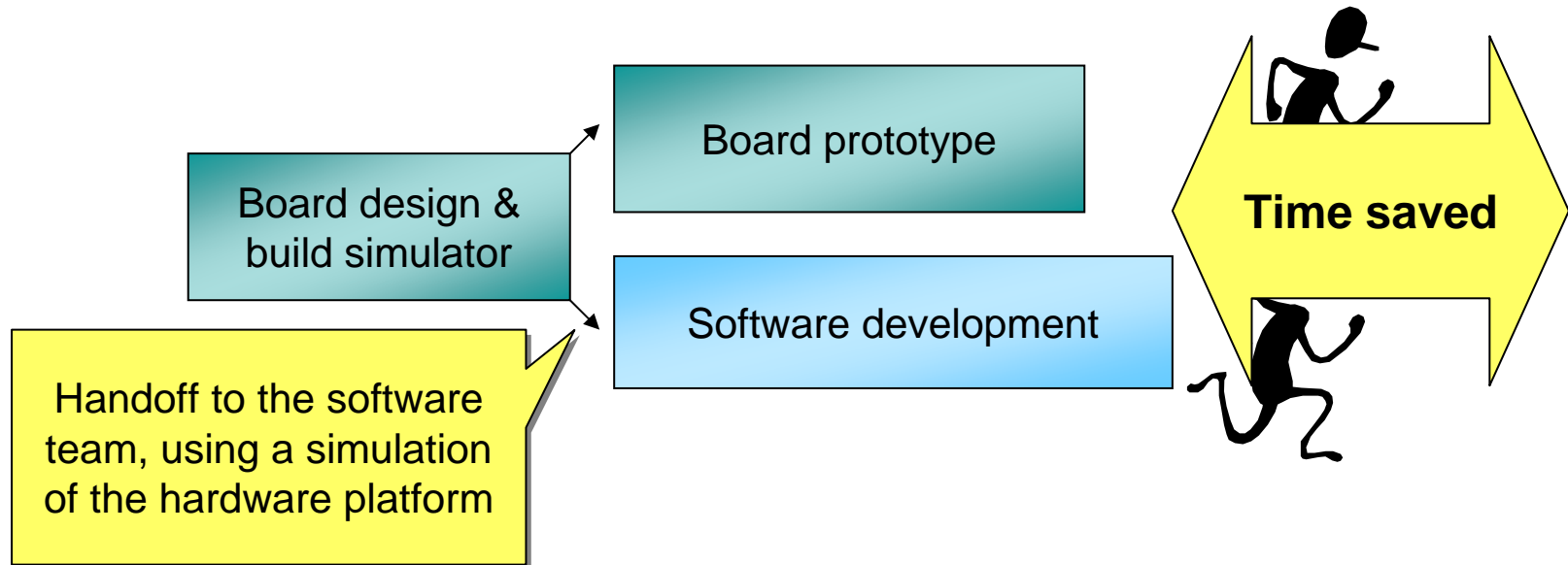
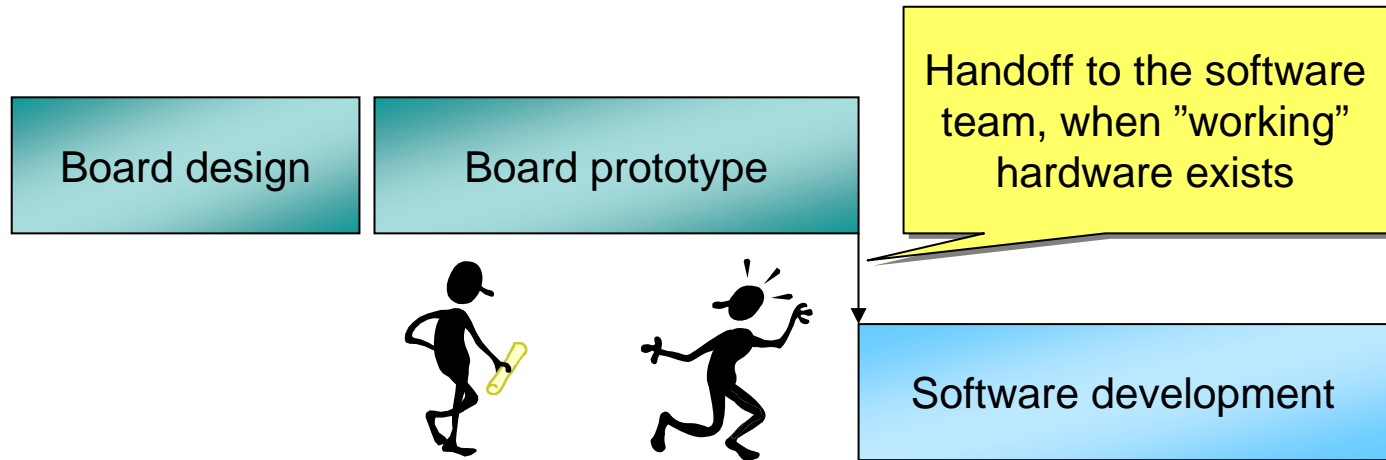
Driving Quality Sooner

With hardware only —

With virtualized software development —



Parallellize SW & HW Development

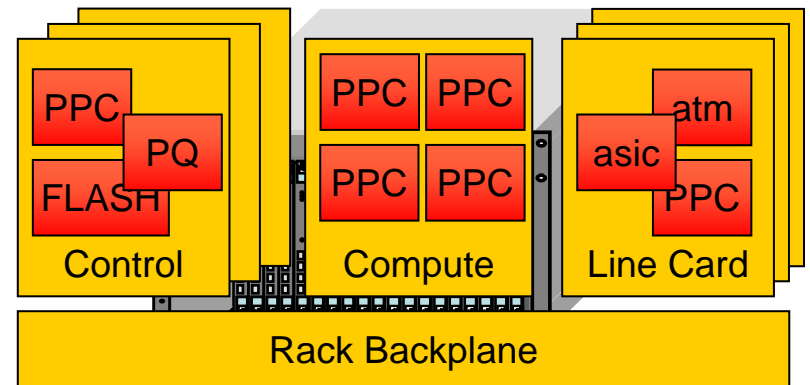


- Problem: Ericsson needed to test software on a large range of base-station configurations
- Challenges:
 - Hardware is expensive and takes 2-14 days to reconfigure before testing
 - Systems can have up to 66 boards and 700 processors
 - Test teams are geographically distributed
- Solution: Create Simics models for each board and handle all re-configuration through scripts
- Benefits:
 - Enormous reduction in cost of capital equipment used for testing
 - Can reconfigure a system almost instantly
 - Can handle even a fully populated system

Ericsson CPP Emulator

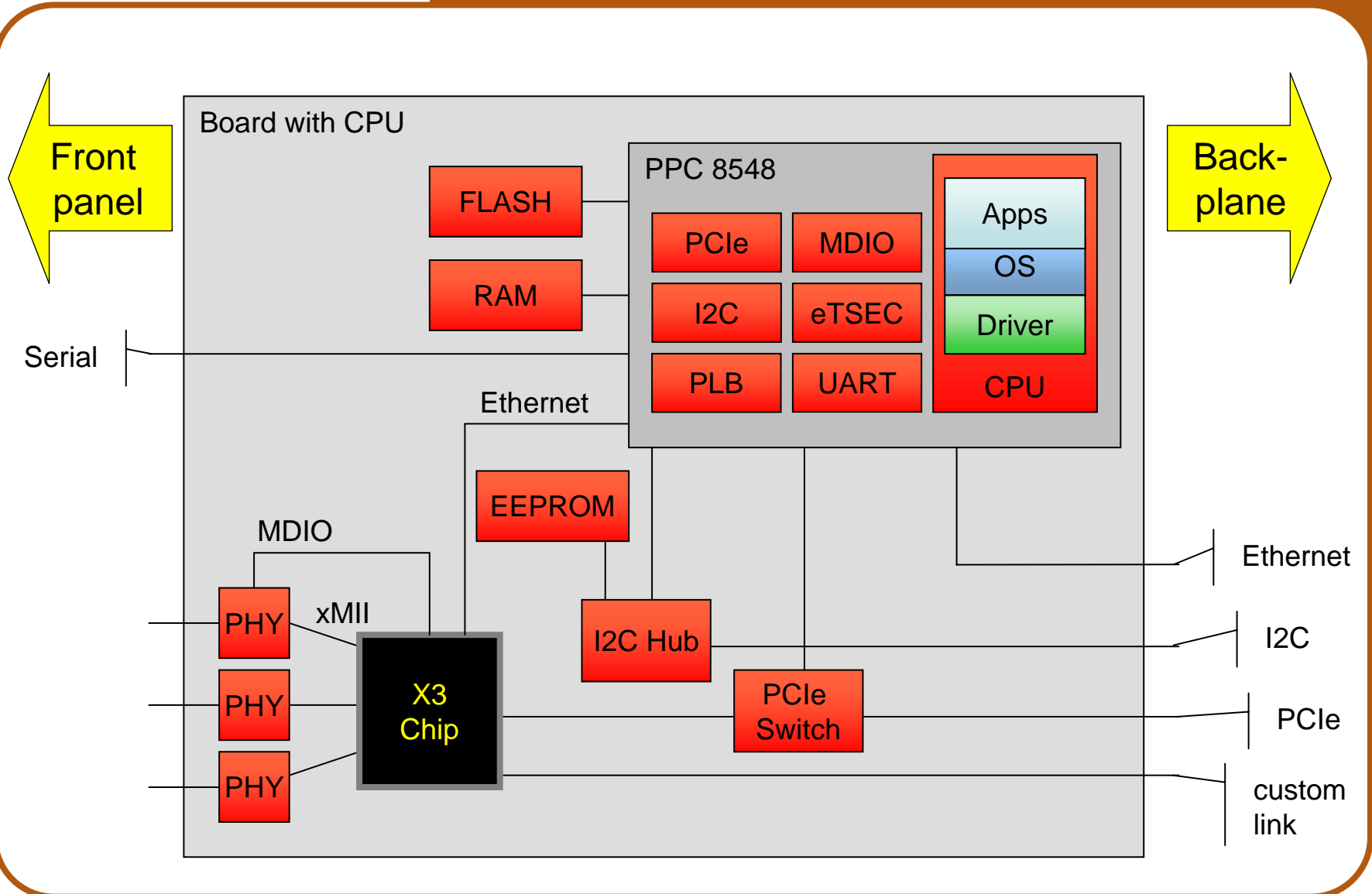
- Telecom Switch
 - ATM Backplane
 - Ethernet frontside
 - Serial
 - 20+ different card types
 - Control cards
 - Timer units
 - Line cards
 - Backplane switch cards
 - Multipro compute cards
 - DSP processing cards
 - 20+ cards in a rack
 - Combined arbitrarily

 - Extreme system size
 - 10-100s processors
 - 10+ of GB target RAM
-
- Multiple processor types
 - PowerPC 750, 750fx, 750gx
 - PowerPC 403, 405, 440
 - PowerQUICC II 8260, 8270, 8280
 - TI C64 DSP



- Problem: SwitchCore needed to develop and test drivers and protocol stacks for their next generation Xpeedium3 chips
- Challenges:
 - Silicon not yet available
 - SwitchCore customers need to evaluate performance and to develop their own software layers
 - Previously had used an internal simulator but slow and expensive to maintain
- Solution: Model Xpeedium3 using Simics
- Benefits:
 - Internal software development (including offshore)
 - Customers can develop their own software using the same model
 - Reduced delay between prototype availability and production orders

Switchcore Board Components



- Problem: Wind River needed to develop software for the Freescale MPC8641D

- Challenges:
 - No prototype silicon was available
 - Silicon schedule was slipping but customers still required Wind River support on schedule
 - 8641D is a dual-core chip - this is not a straightforward port

- Solution:
 - Wind River's engineering organization ported VxWorks using Virtutech Simics with the 8641D processor model

- Benefits:
 - Development could start ahead of silicon
 - Improved productivity, improved software quality, earlier availability of 8641D software to Wind River's customers

- “By using Virtutech Simics, Wind River has been able to accelerate the development of its products for the MPC8641D. With this simulation model, our engineers not only have pre-silicon access to chips, they can tap into Simics’ unique debugging tool, Hindsight, to move forward and backward in the code with the ability to stop at any point.”
 - Tomas Evensen, Chief Technology Officer
Wind River

- Operating-system kernel crash in virtual model
 - Divide-by-zero right in the kernel
 - Algorithm to determine and compensate for clock skew
 - Division by difference in time between two processors
- Virtual model had zero clock skew = provoked error
 - Could have happened on a real system
 - Just not very likely
 - Typical rare problem in the field
 - Essentially testing a rare corner case in system state

- Changed clock frequency of virtual MPC8641D
 - From 800 to 833 Mhz
 - OS froze on startup – quite unexpectedly
- Investigation:
 - Only happened at 832.9 to 833.3 MHz
 - Determinism: 100% reproduction of error trivial
 - Time control: single-step code feasible
 - Insight: look at complete system state, log interrupts, check the call stack at the point of the freeze, check lock state
- What we found:
 - ISR takes a lock on entry, and then expect a second external interrupt to occur to unlock the data structure. But this interrupt arrives before interrupts are reenabled, and thus we are stuck in deadlock. Took a few hours.