DEVCON
Enabling the Smart Society
OCTOBER 22-25, 2012
HYATT REGENCY ORANGE COUNTY

RENESAS

MULTICORE DESIGN SIMPLIFIED
imperas

Virtual Platform Based Software Testing

Larry Lapides
Imperas Software Ltd

Class ID: 0C19B

Renesas Electronics America Inc.

Larry Lapides

- VP worldwide sales, responsible for sales and marketing at Imperas Software Ltd
- VP worldwide sales at Averant and Calypto Design Systems
- VP of worldwide sales at Verisity, including for the 2001 IPO
- Sales and marketing for Exemplar Logic and Mentor Graphics
- Entrepreneur-in-Residence at Clark University's Graduate School of Management
- BA in Physics from the University of California Berkeley
- MS in Applied and Engineering Physics from Cornell University
- MBA from Clark University
- Food and wine – contributor to ViciVino.com

Agenda

- Silicon without software is just sand...
  - Issues in embedded software development
  - Software quality / testing
- What is a virtual platform?
  - Building a virtual platform
  - Requirements for a virtual platform testing environment
- Case studies for virtual platform based software testing
  - System testing – integration with Simulink
  - Software regression testing
  - In depth software analysis: exception behavior
  - Fault injection
- Summary, Q&A

Agenda

- Silicon without software is just sand...
  - Issues in embedded software development
  - Software quality / testing
- What is a virtual platform?
- Case studies for virtual platform based software testing
- Summary, Q&A

Silicon Without Software Is Just Sand

Photo courtesy of Renesas Electronics Corporation

- ✓ Software development is costing more than chip development cost
- ✓ Embedded software is the critical path to system delivery
- ✓ Source code is doubling annually
- ✓ Software complexity is increasing dramatically with multi-core devices, multi-processor systems
- ✓ Products are defined by their software

Issues in Embedded Software Development

- Quality is critical
- Current development methodology breaks with increasing code complexity
- Time to market still counts!
- Management cannot manage the software development process: insufficient metrics
  - *You cannot manage what you cannot measure*

Focus for Today's Presentation: Software Quality / Testing

- Current test methodologies employ testing on hardware
  - Actual hardware
  - Prototypes
  - Hybrid methods involving simulation driving hardware: hardware in the loop (HIL)
- These methods lack visibility, controllability
  - Visibility: if an error occurs, will it be observed by the test environment?
  - Controllability:
    - Can corner cases be tested?
    - Can an error be made to happen?
- Virtual platforms – software simulation – provide a complementary technology to the current methodology
  - Simulation promises visibility, controllability
  - How to deliver on this promise?

Software Failures in Embedded Systems Are Bad!

This Car Runs on Code

February 5, 2010

The avionics system in the F-22 Raptor, the current U.S. Air Force frontline jet fighter, consists of about 1.7 million lines of software code. The F-35 Joint Strike Fighter, scheduled to become operational in 2010, will require about 5.7 million lines of code to operate its onboard systems. And Boeing's new 787 Dreamliner, scheduled to be delivered to customers in 2010, requires about 6.5 million lines of software code to operate its avionics and onboard support systems.

These are impressive amounts of software, yet if you bought a premium-class automobile recently, "it probably contains close to 100 million lines of software code," says Manfred Broy, a professor of informatics at Technical University, Munich, and a leading expert on software in cars. All that software executes on 70 to 100 microprocessor-based electronic control units (ECUs) networked throughout the body of your car.

http://news.discovery.com/autos/toyota-recall-software-code.html

ailures Responsible for cal Device Recalls

e behind 24 percent of all the medical device recalls in 2011, according to od and Drug Administration, which said it is gearing up its labs to spend he quality and security of software-based medical instruments and

cience and Engineering Laboratories (OSEL) released the data in its on June 15, amid reports of a compromise of a Web site used to distribute hospital respirators. The absence of solid architecture and "principled " in software development affects a wide range of medical devices, with potentially life-threatening consequences, the Agency said.

There is growing evidence that software security and integrity is a growing problem in the medical field. In October, for example, security researcher Barnaby Jack demonstrated a remote, wireless attack on an implantable insulin pump from the firm Medtronic.

https://threatpost.com/en_us/blogs/fda-software-failures-responsible-24-all-medical-device-recalls-062012

- **Systems are getting more complex**
  - **70-100 processors in cars**
- **Software failures can be life-threatening**
- **Software failures now include security breaches**

DEVCON Enabling the Smart Society

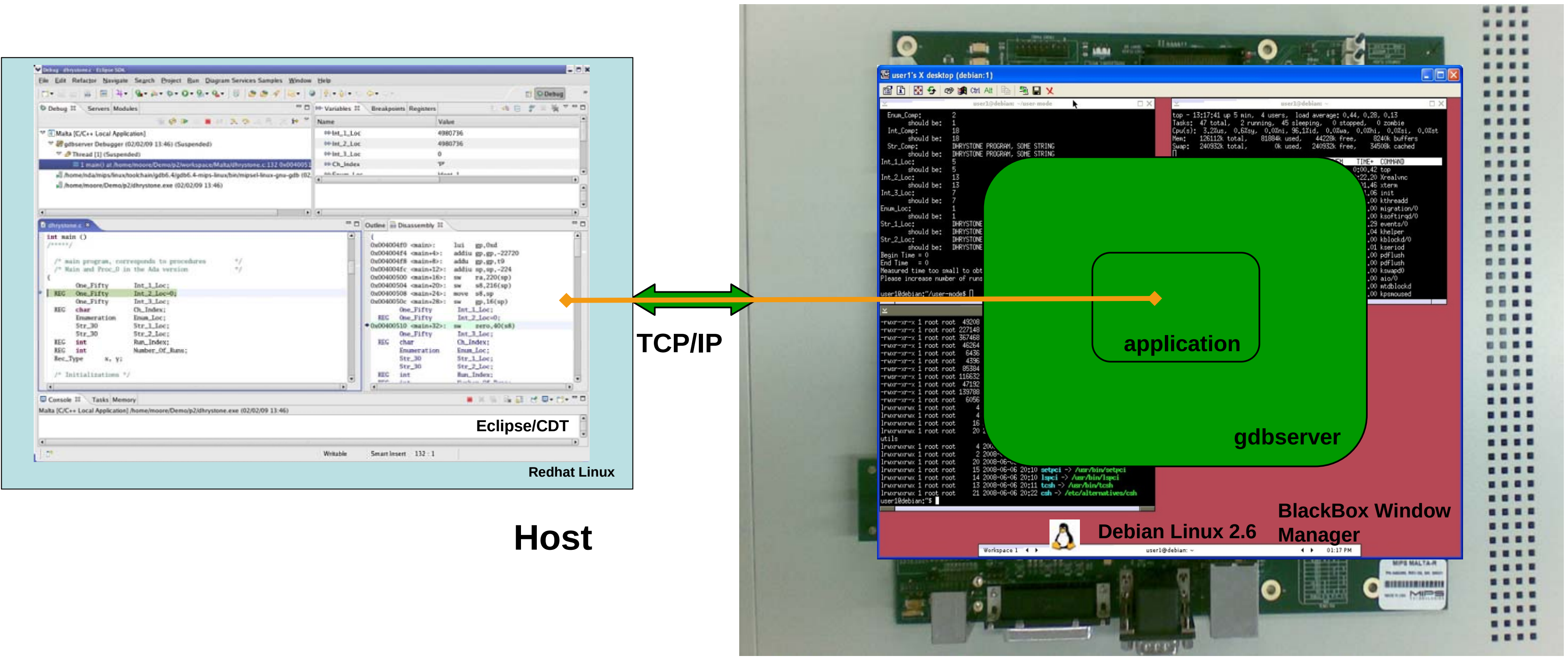
MULTICORE DESIGN SIMPLIFIED imperas

Agenda

- Silicon without software is just sand…
- What is a virtual platform?
  - Building a virtual platform
  - Requirements for a virtual platform testing environment
- Case studies for virtual platform based software testing
- Summary, Q&A

Current Methodology, Software Debug on Prototype:

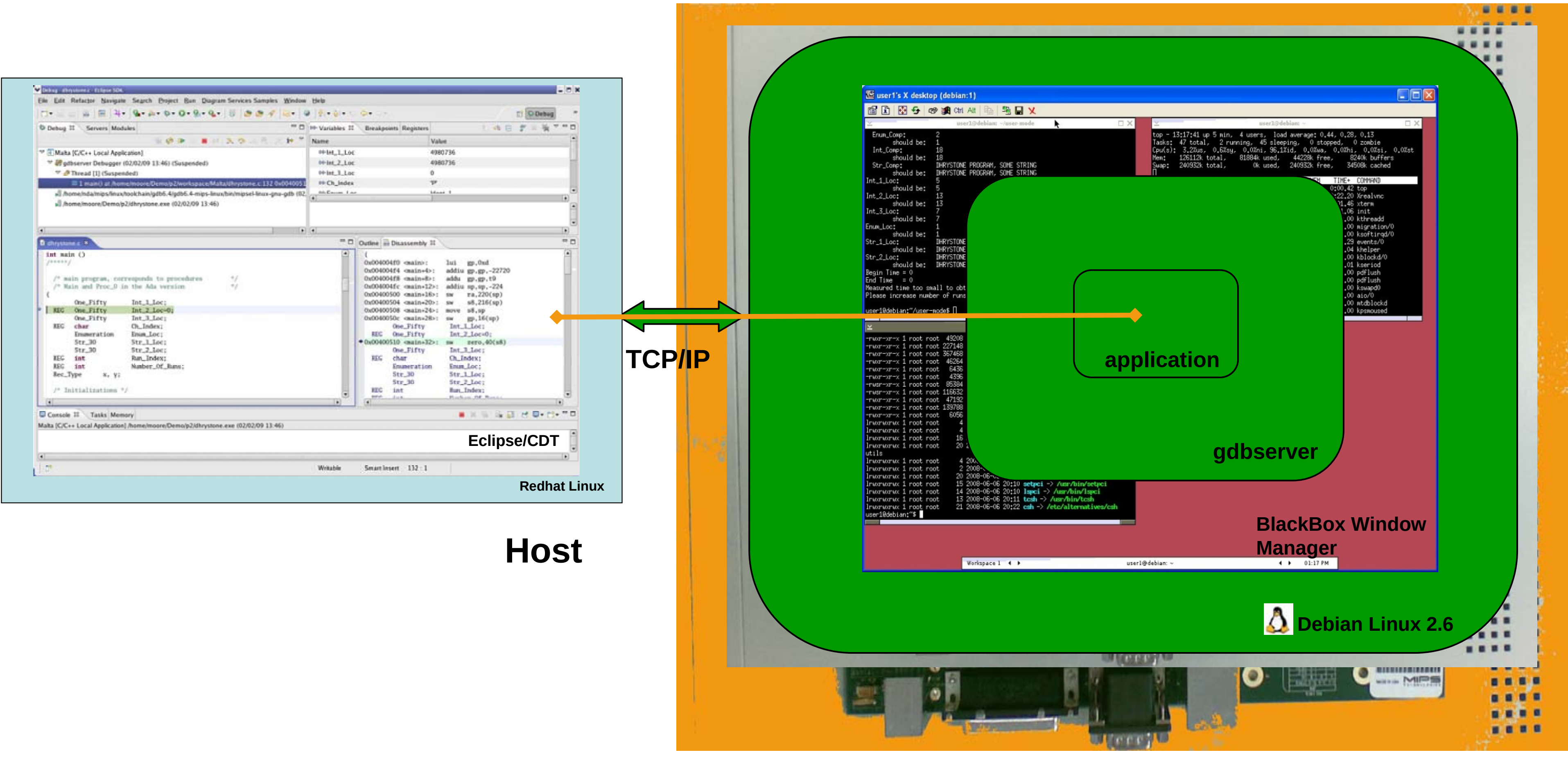
Run gdbserver on target and Eclipse on host to debug application on target



Host

Remote Target

Using a Virtual Platform Provides Exactly the Same Environment
(with many of the same limitations)



Virtual Platform as Remote Target

Building the Virtual Platform

- The virtual platform is a set of models that reflects the hardware on which the software will execute
  - Subset / subsystem of a single device
  - Processor chip
  - Board
  - System
- Models are typically written in C or SystemC
- Models for individual components – interrupt controller, UART, ethernet, … – are connected just like in the hardware
- Peripheral components can be connected to the real world by using the host workstation resources: keyboard, mouse, screen, ethernet, USB, …
- Models can be cycle accurate, cycle approximate, or instruction accurate, with instruction accurate models providing the highest simulation performance

Instruction Accurate Virtual Platforms Run at 100s of MIPS

- To get the high speed required for real usage, processor hardware is modeled only to the minimum necessary level for *correct or plausible instruction behavior* so that software cannot tell it is not running on real hardware*.* Other features are approximated or omitted. Some examples:
  - *Accurately modeled*
    - Most instructions
    - Exceptions
    - Structures, such as TLBs, required to allow OS boot
  - *Approximated*
    - Tick timers – one "tick" per instruction
    - Random number generators (can affect, for example, TLB replacement algorithms)
  - *Omitted*
    - Instruction pipelines
    - Speculative execution
    - Write buffers
    - Caches (can be added; not modeled by default)
- General rule – if a feature cannot be modeled with reasonable accuracy, don't model it at all (no bogus pretence of accuracy)

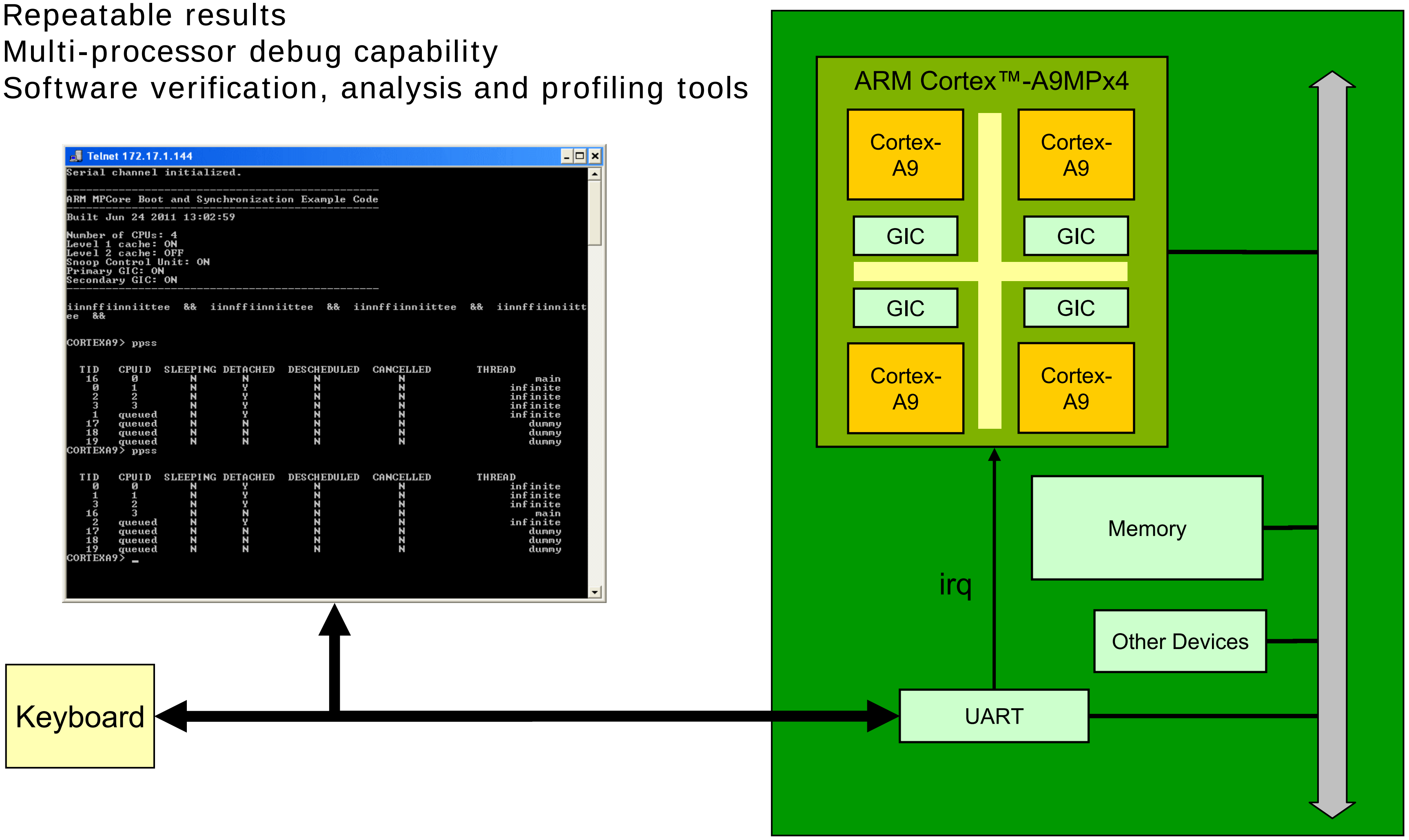
Open Virtual Platforms™ Provides the Modelling Infrastructure

- Website community/portal/forum
- Over 6,500 people registered on the website

- Modeling APIs for processor, peripheral, and platform modeling
- Open source library of models (Apache 2.0 open source license)
  - Fast Processor Models (100+ by end 2012): ARM, MIPS, **Renesas**, …
    - Current Renesas processor core models include V850 ES, E1, E1F; M16C
    - Short term roadmap includes V850 E2, G3M, G3K; RL78
  - Peripheral models: UART, timer, interrupt, ethernet, DMA, I/O, …
  - Working platforms: Linux, Nucleus, µC/OS II, FreeRTOS, bare metal applications, …
  - OVP™ and SystemC/TLM2.0 native interfaces for all models
- OVPsim™ simulator (models need the simulator to execute)
  - Runs processor models fast, 100s of mips
  - Interfaces to GDB via RSP
  - Encapsulation in Eclipse IDE for software and platform debug

Virtual Platform Requirements for Software Test

- ✓ Performance near real time
- ✓ Run target binaries without change
- ✓ Repeatable results
- ✓ Multi-processor debug capability
- ✓ Software verification, analysis and profiling tools



```
Serial channel initialized.

------------------------------------------------
ARM MPCore Boot and Synchronization Example Code
------------------------------------------------
Built Jun 24 2011 13:02:59

Number of CPUs: 4
Level 1 cache: ON
Level 2 cache: OFF
Snoop Control Unit: ON
Primary GIC: ON
Secondary GIC: ON
------------------------------------------------

iinnffiinniittee  &&  iinnffiinniittee  &&  iinnffiinniittee  &&  iinnffiinniitt
ee  &&


CORTEXA9> ppss


  TID   CPUID  SLEEPING DETACHED  DESCHEDULED  CANCELLED      THREAD
   16     0       N        N          N           N                 main
    0     1       N        Y          N           N             infinite
    2     2       N        Y          N           N             infinite
    3     3       N        Y          N           N             infinite
    1   queued    N        Y          N           N             infinite
   17   queued    N        N          N           N                dummy
   18   queued    N        N          N           N                dummy
   19   queued    N        N          N           N                dummy
CORTEXA9> ppss


  TID   CPUID  SLEEPING DETACHED  DESCHEDULED  CANCELLED      THREAD
    0     0       N        Y          N           N             infinite
    1     1       N        Y          N           N             infinite
    3     2       N        Y          N           N             infinite
   16     3       N        N          N           N                 main
    2   queued    N        Y          N           N             infinite
   17   queued    N        N          N           N                dummy
   18   queued    N        N          N           N                dummy
   19   queued    N        N          N           N                dummy
CORTEXA9> _
```

Virtual Platform Requirements for Software Test: Checklist

- Performance near real time
- ✓ Instruction accurate virtual platforms run at 100s of MIPS

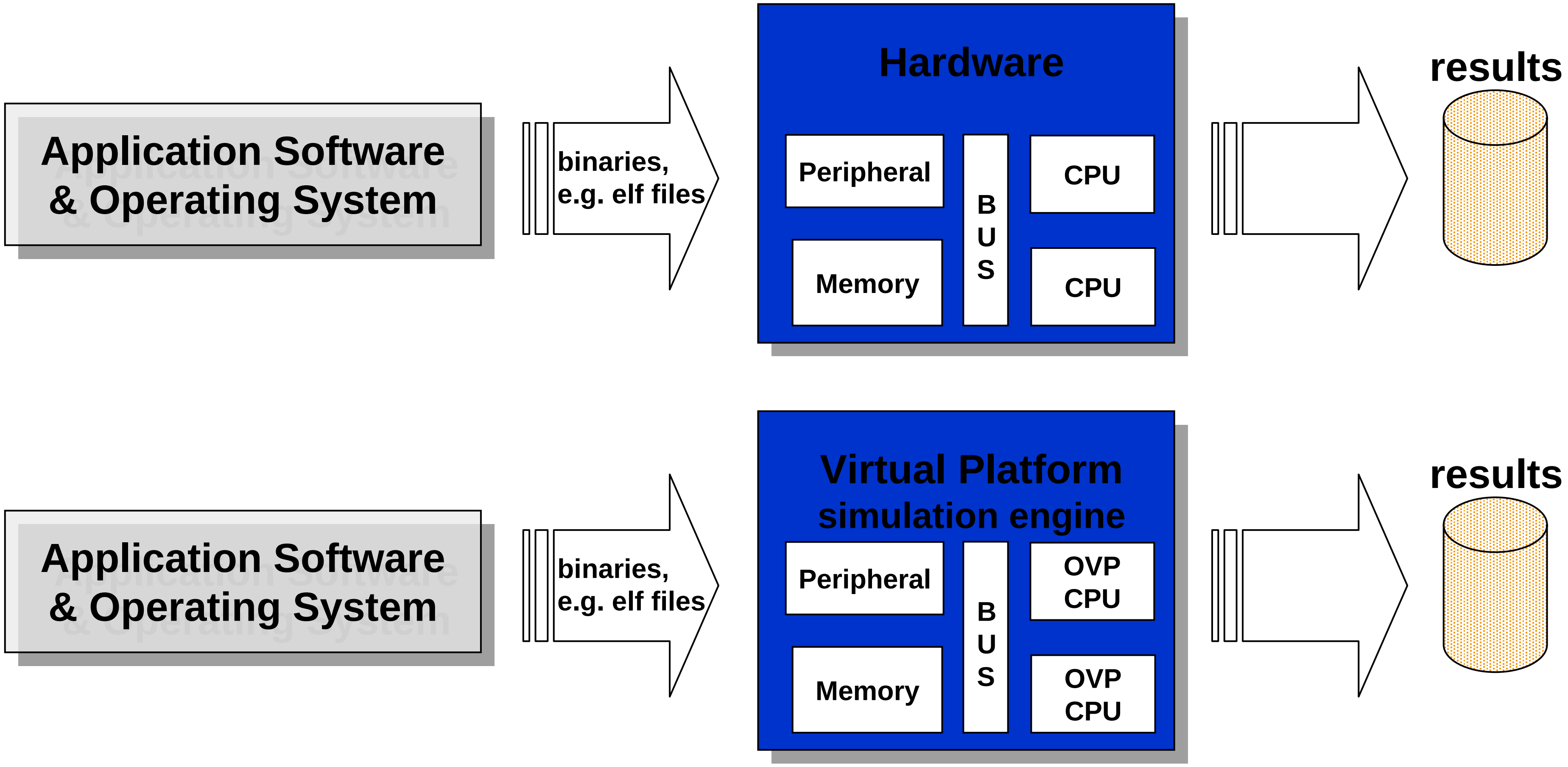
- Run target binaries without change
- ✓ Use the same tool chain for compiling as for the real hardware

- Repeatable results
- ✓ Simulation is a deterministic process, with repeatable results

- Multi-processor debug capability
  - Whether multiple processors on one device or board or system
- ✓ Available either from virtual platform tool vendor or tool chain (IDE) vendor

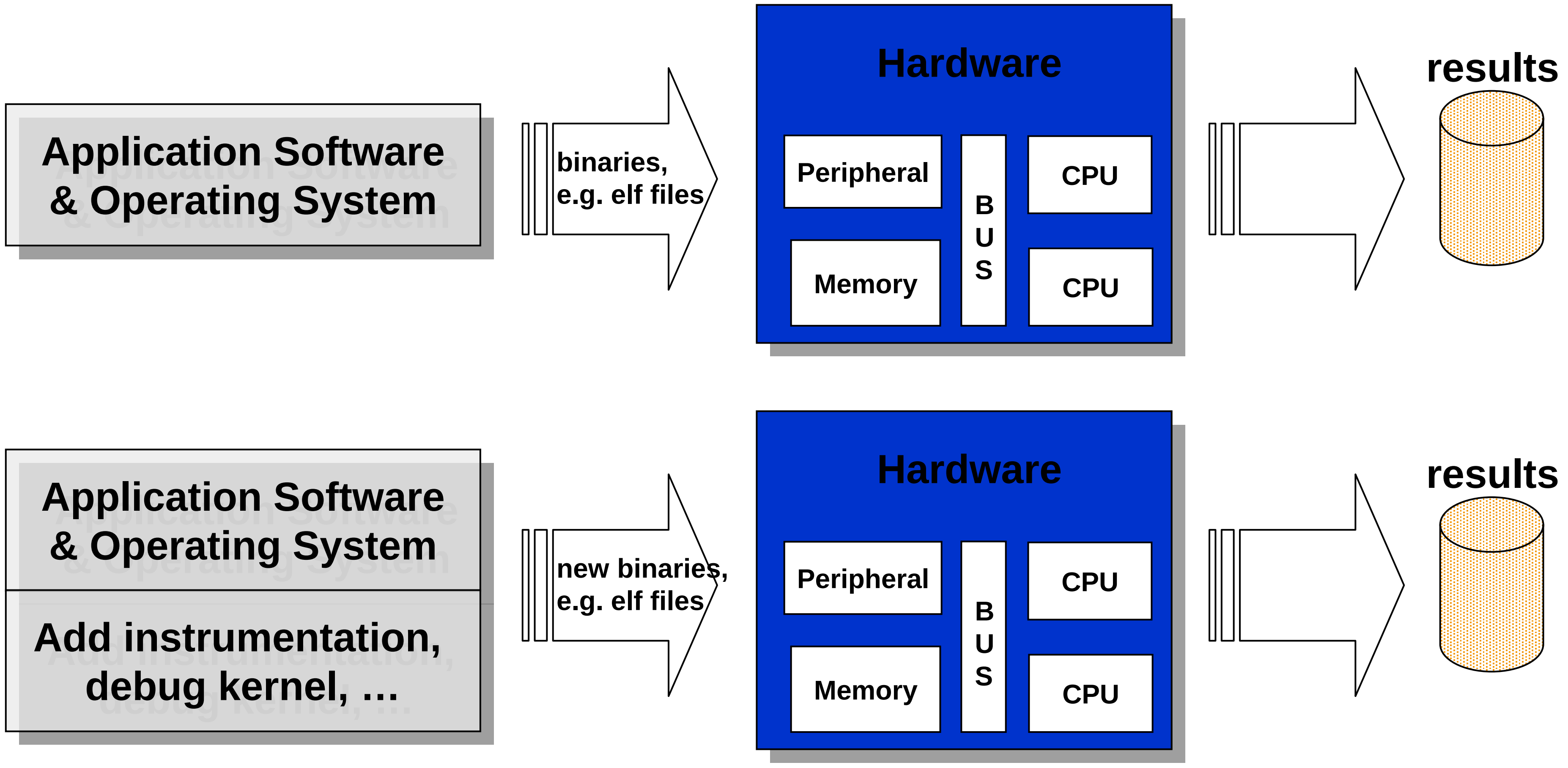
- Software verification, analysis and profiling tools
- Tools are needed so the virtual platforms can deliver on the simulation promise of complete controllability, visibility

Virtual Platforms Simulate the Software Running on the Hardware



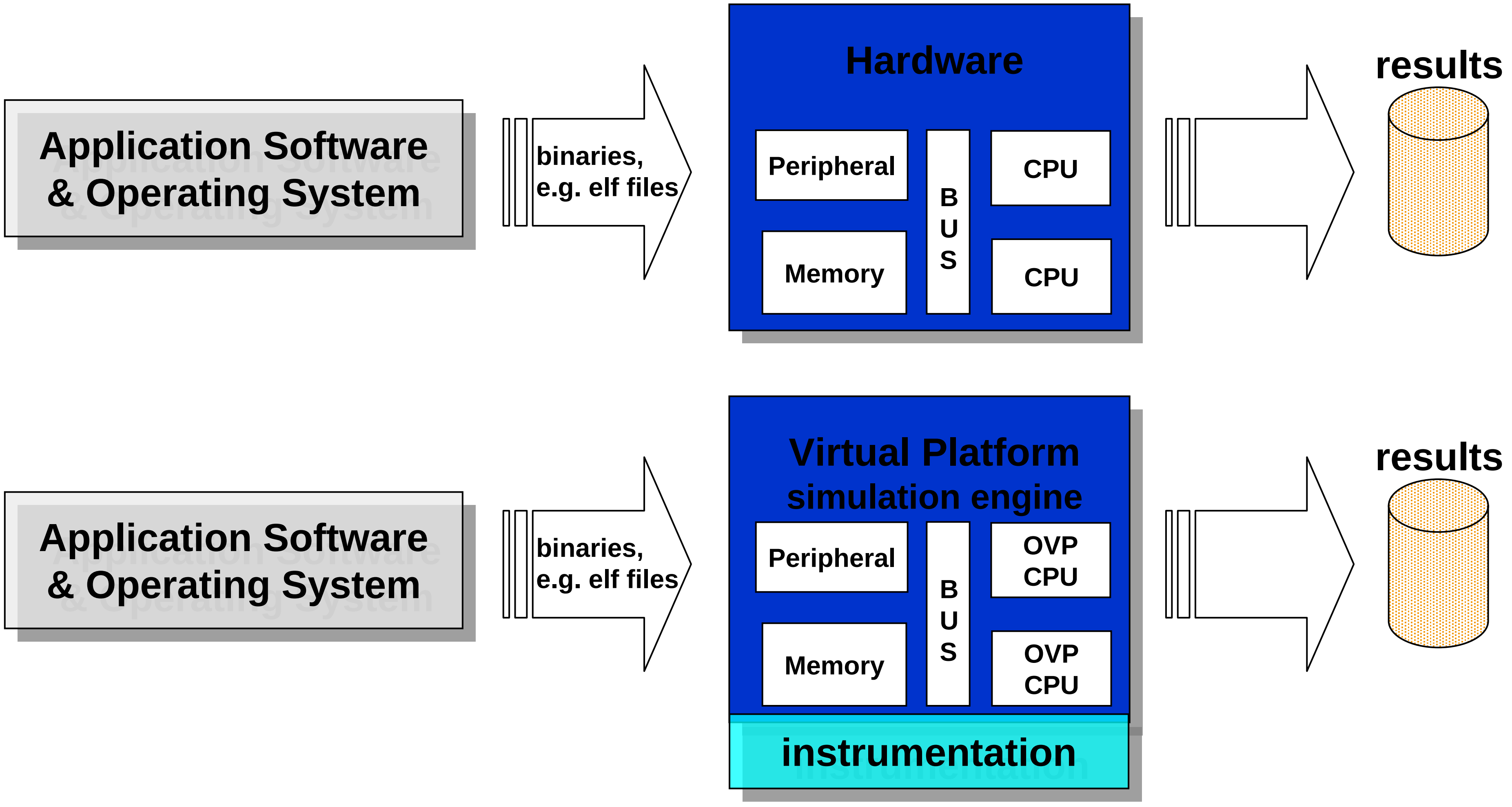
results(HW) = results(VP)

Software Analysis on Hardware Has Accuracy Questions
(code coverage, profiling, …)


Application Software & Operating System
binaries, e.g. elf files
Hardware
Peripheral
BUS
CPU
Memory
CPU
results
Application Software & Operating System
Add instrumentation, debug kernel, …
new binaries, e.g. elf files
Hardware
Peripheral
BUS
CPU
Memory
CPU
results

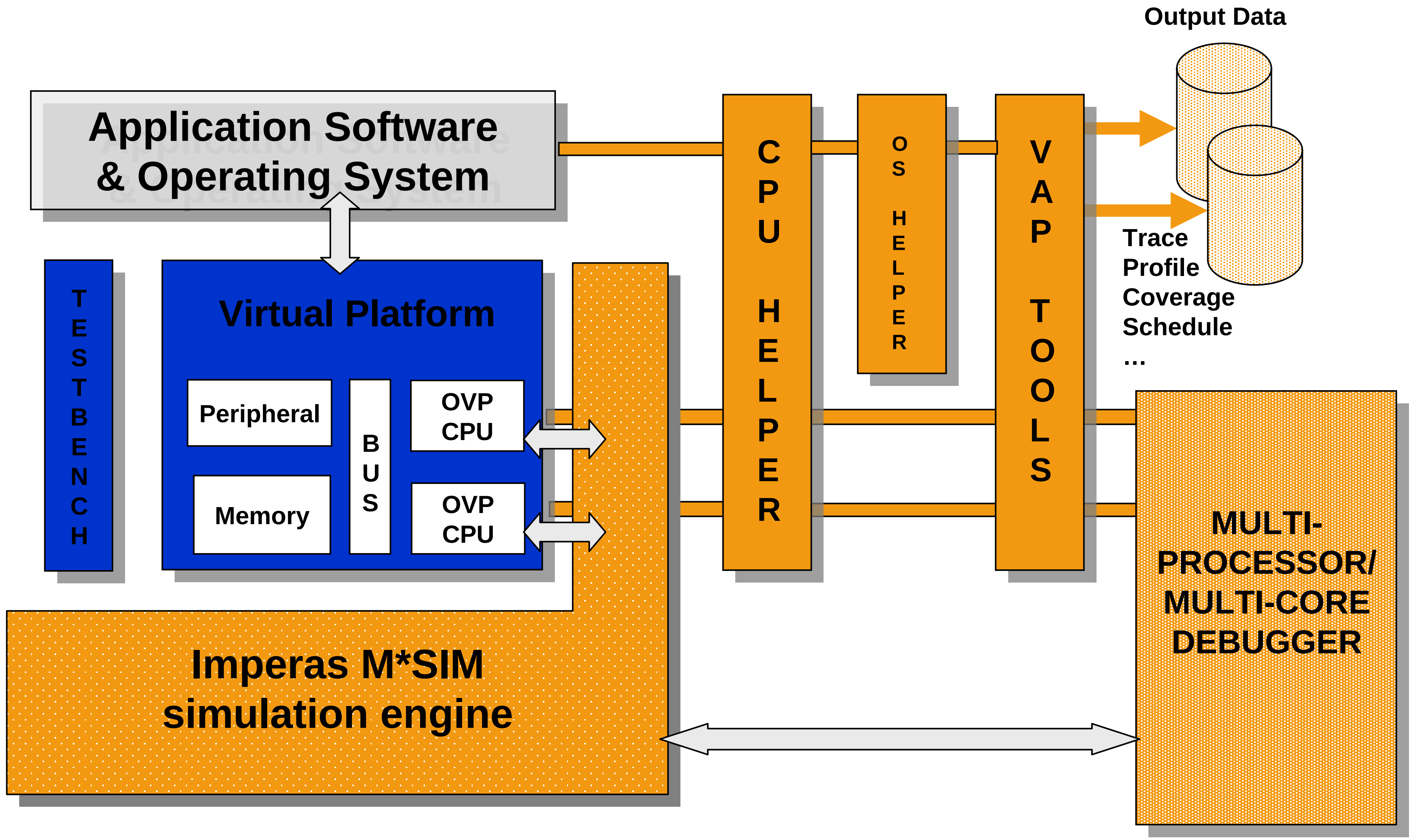
$$\text{results(HW)} \overset{?}{=} \text{results(HW + instrumentation)}$$

Software Analysis on Virtual Platform is Non-Intrusive
(code coverage, profiling, tracing, memory analysis, …)



results(HW) = results(VP + instrumentation)

Virtual Platform with Verification, Analysis and Profiling (VAP) Tools Plus Debugger



Requirements for VAP (Verification, Analysis & Profiling) Tools

- Non-intrusive: no modification of application source code
- Minimal overhead: simulations should still run *fast*
- Modular: can run one or more without tools stepping on each other
- Flexible: interactive or scripted use models
- Configurable: adjust for specific platform and focus
- Distributable: need to be shipped with virtual platform as integral part of SDK for specific platform/chip

Agenda

- Silicon without software is just sand…
- What is a virtual platform?
- Case studies for virtual platform based software testing
  1) System testing – integration with Simulink
  2) Software regression testing
  3) In depth software analysis: exception behavior
  4) Fault injection
- Summary, Q&A

Example 1: Simulink Integration

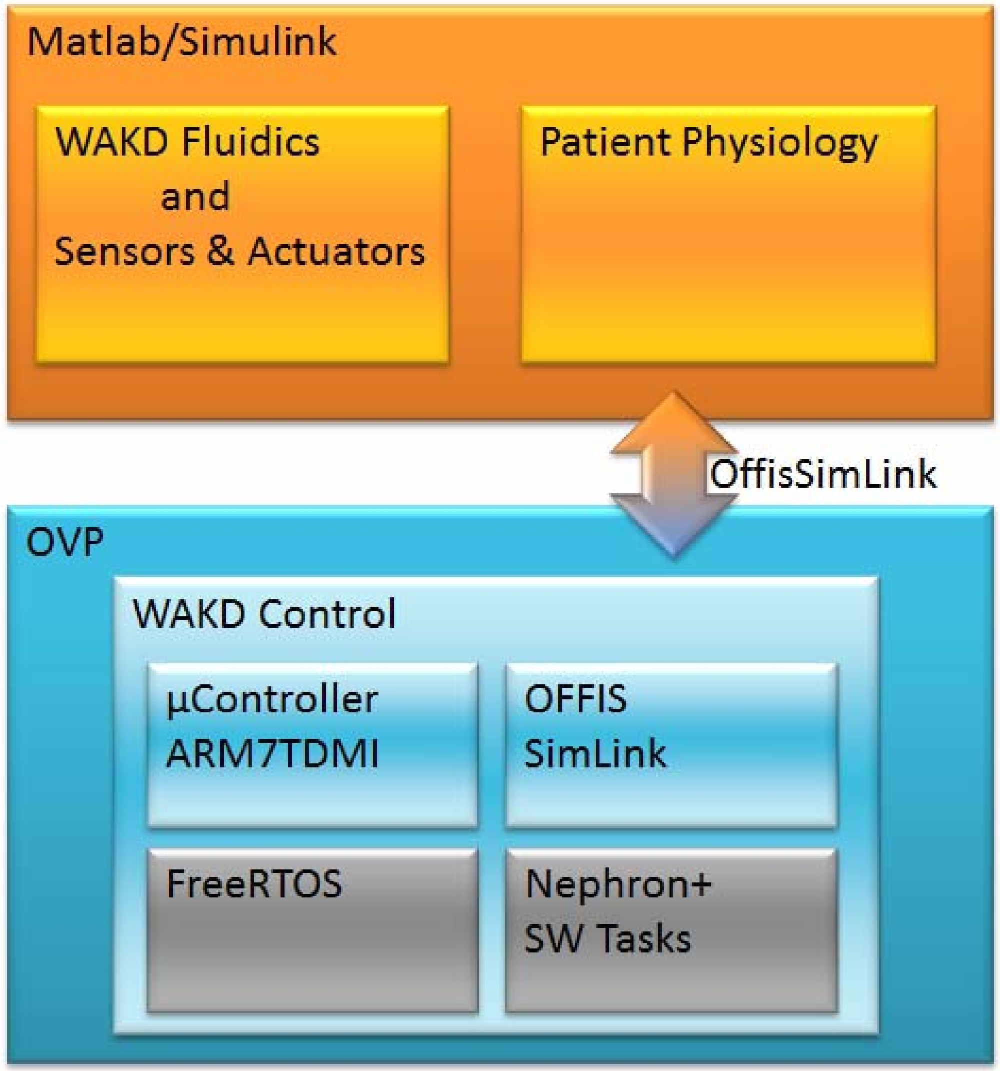
- Simulink and Matlab (Mathworks) are well established tools for system level simulation and analysis
  - At algorithmic level
  - No processor/software detail
- Would like to integrate OVPsim and Simulink
  - OVPsim typically only needed for Fast Processor Model

  a) Use Simulink as test stimuli for software, with OVPsim as master simulator (OFFIS)
     - Co-Simulation of C-Based SoC Simulators and Matlab Simulink
     - Frank Poppen, Kim Gruettner, OFFIS, Oldenberg, Germany
     - Proceedings of the Operation Research Society Simulation Workshop 2012

  b) Need more detailed system analysis, use Simulink as master simulator (FZI)
     - Scalable Problem-Oriented Approach for Dynamic Verification of Embedded Systems
     - Francisco Mendoza et al, FZI, Karlsruhe, Germany
     - 1st IFAC Conference on Embedded Systems, CESCIT 2012

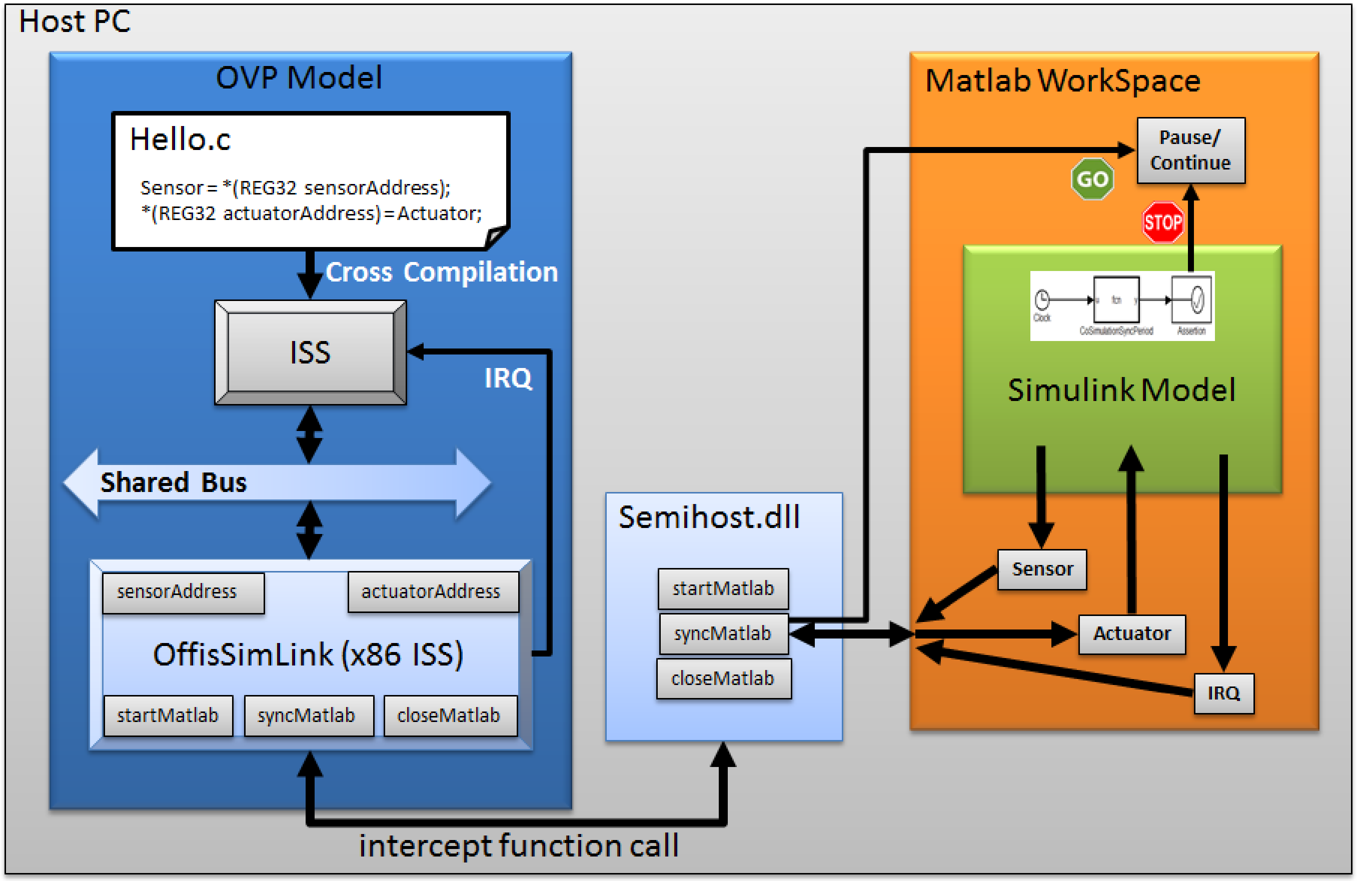
OFFIS Nephron+ System (Medical Electronics)

- Nephron+ research project goal is to develop a wearable kidney filter
  - Reduce kidney dialysis visits by 10x (cost savings)
  - Improved patient lifestyle
- Use Simulink to model patient-related physiology parameters, wearable kidney system
- Use OVPsim to simulate control device (ARM-based microcontroller)
- Simulink provides test stimulus for control software (Nephron+ SW Tasks) running on OVPsim



OFFIS Integration Detail

- OFFIS developed OffisSimLink OVP peripheral model to provide interface to Simulink



Example 2: Software Regression Testing
(NIRA Dynamics AB, subsidiary of Audi)

- Tire pressure sensors
  - Software application only; runs on anti-lock braking system
    - After calibration, detects changes in tire pressure by changes in braking data
- Use different processors (from different ABS systems)
  - ARM 7, Cortex-M3, Cortex-R4
  - Renesas V850
  - Need to run the same application on each processor, so memory usage is key
- Software test and analysis
  - Collect weeks/months of road test data
  - Want to run road test data as software regression suite, 1,000s of tests each night
  - Want to ensure that stack and heap behave properly (memory analysis tools)
- Imperas M*SDK and OVP Fast Processor Models
  - Meets their speed, processor support, memory analysis requirements
  - Virtual platform value is enhanced software testing capabilities

Audi
Vorsprung durch Technik

- Peter Lindskog, head of development:
  - "Imperas M*SDK helps us not only to find bugs in our code, but also in the compilers we use."
  - "We will not ship software without testing with Imperas tools."

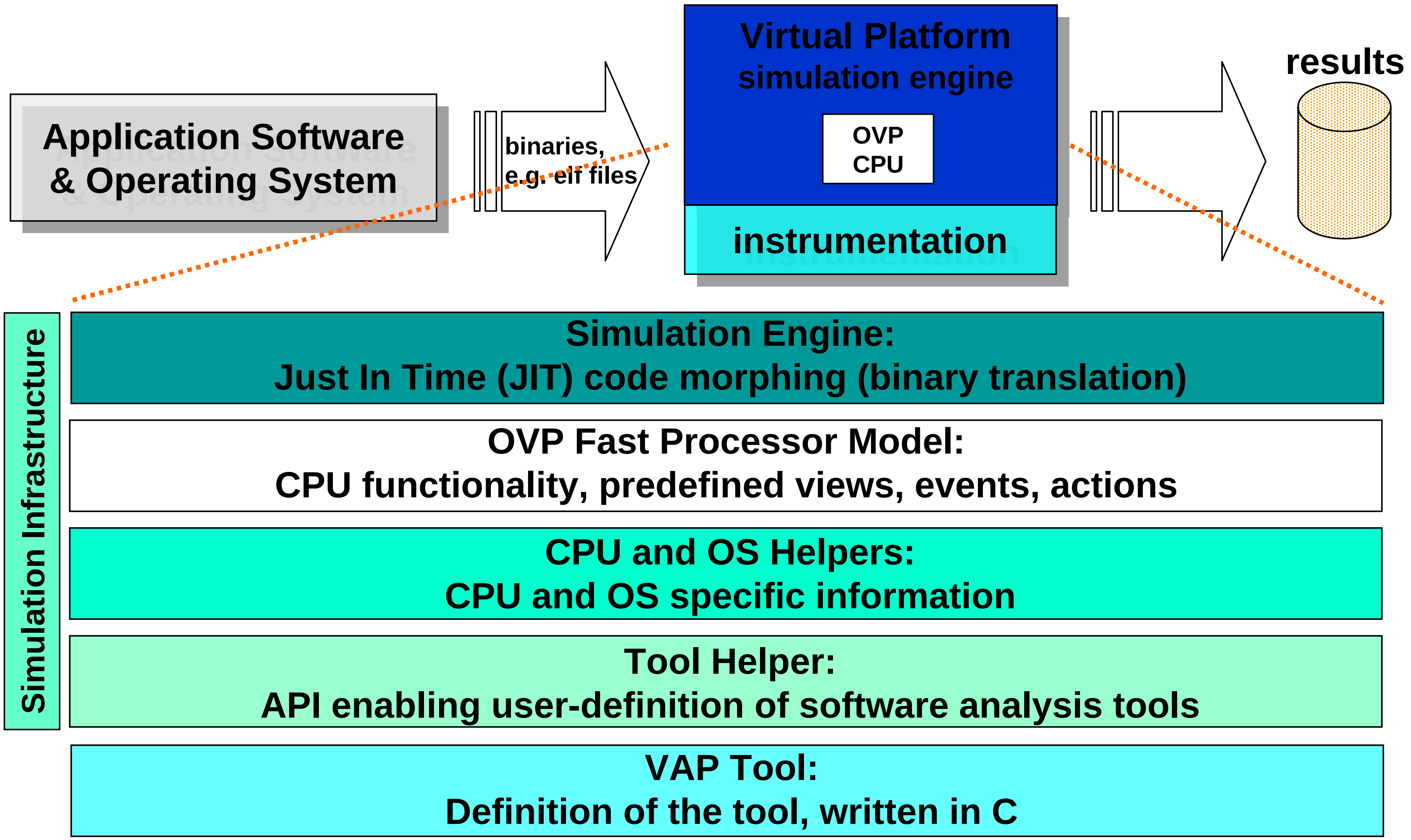
Imperas VAP Tools

- OVP Fast Processor Models enable use of VAP tools
- CPU and OS aware
  - 90+ CPU cores supported
  - OS support: Linux, Nucleus, uCLinux, FreeRTOS, µC/OS II, eCoS, µItron, proprietary
  - Used for hardware-dependent software development
    - Early software development
    - Software testing
    - System analysis
- 25+ M*VAP™ tools: code coverage, profiling (function, OS events), tracing (instruction, function, event, OS task, OS kernel), memory analysis, …
- Non-intrusive
  - No instrumentation or modification of application code
  - No change to instruction ordering
- Executes as native host code for minimal overhead
- Can be used interactively or scripted
- Multiple tools can be loaded simultaneously
- User defined tools enabled: fault injection, protocol verification, software behavior analysis, …
  - Users write tools in C
  - Documented API

Example 3: In Depth Software Behavior Analysis

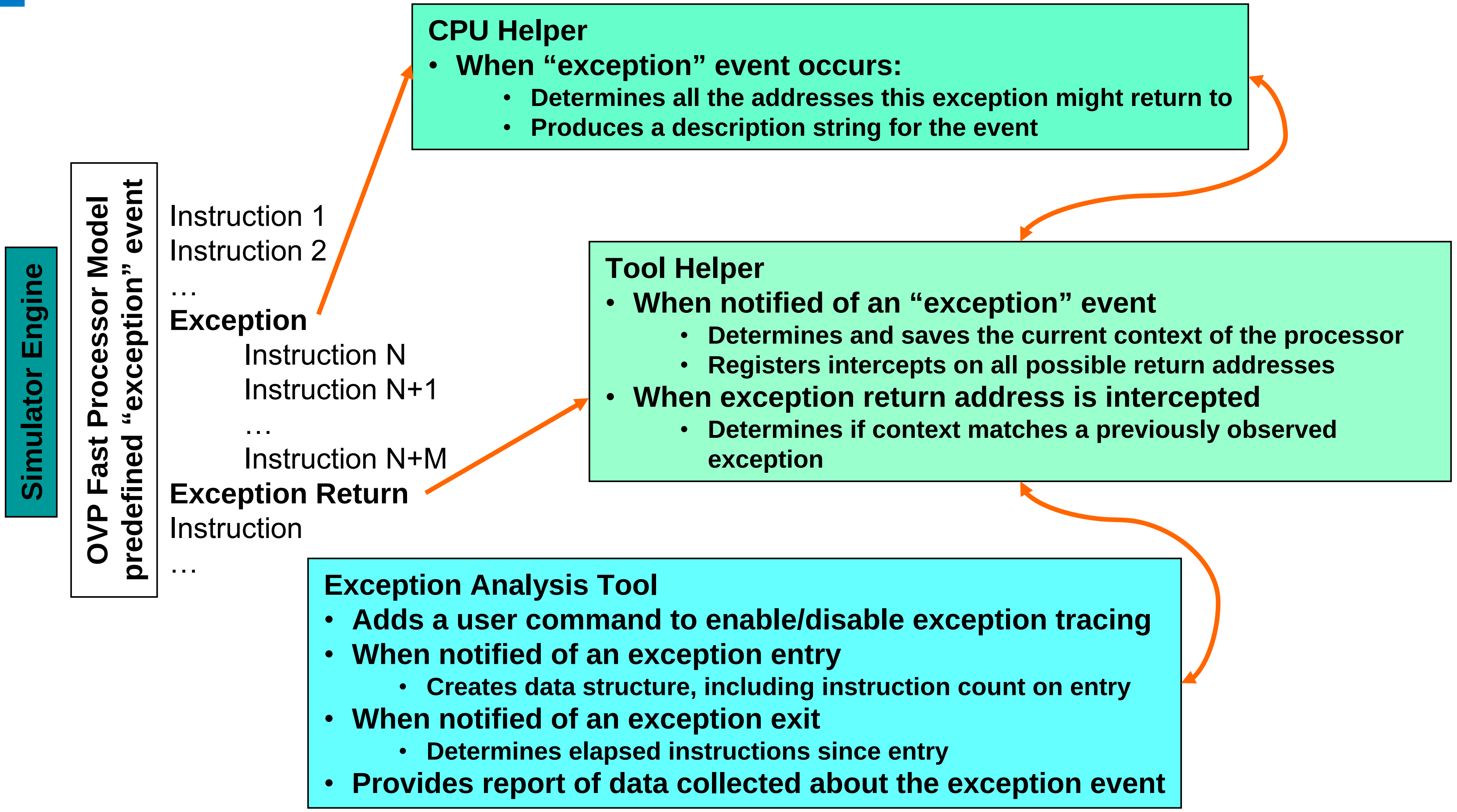
- Goal: Use virtual platform visibility to measure the number of instructions from exception entry to return
- Custom tool developed for analyzing exception handler instruction counts
  - Utilizes VAP Tools infrastructure
    - Registers for callbacks on exceptions and their returns
      - VapHelper provides callbacks on entry and return from exception
      - CpuHelper detects and provides details of exceptions
  - Adds new command to simulation environment to turn on/off tracing
  - Simply reports entries and returns with elapsed instruction counts
  - Could easily be enhanced to provide statistical analysis, report worst case occurrences, provide call stack snapshot at exception, provide RTOS process information, etc.

Simulation Infrastructure Enables Tool Definition



Application Software & Operating System
binaries, e.g. elf files
Virtual Platform simulation engine
OVP CPU
instrumentation
results
Simulation Infrastructure
Simulation Engine: Just In Time (JIT) code morphing (binary translation)
OVP Fast Processor Model: CPU functionality, predefined views, events, actions
CPU and OS Helpers: CPU and OS specific information
Tool Helper: API enabling user-definition of software analysis tools
VAP Tool: Definition of the tool, written in C

Exception Analysis Tool



Instruction 1
Instruction 2
…
Exception
Instruction N
Instruction N+1
…
Instruction N+M
Exception Return
Instruction
…

CPU Helper

- **When “exception” event occurs:**
  - **Determines all the addresses this exception might return to**
  - **Produces a description string for the event**

Tool Helper

- **When notified of an “exception” event**
  - **Determines and saves the current context of the processor**
  - **Registers intercepts on all possible return addresses**
- **When exception return address is intercepted**
  - **Determines if context matches a previously observed exception**

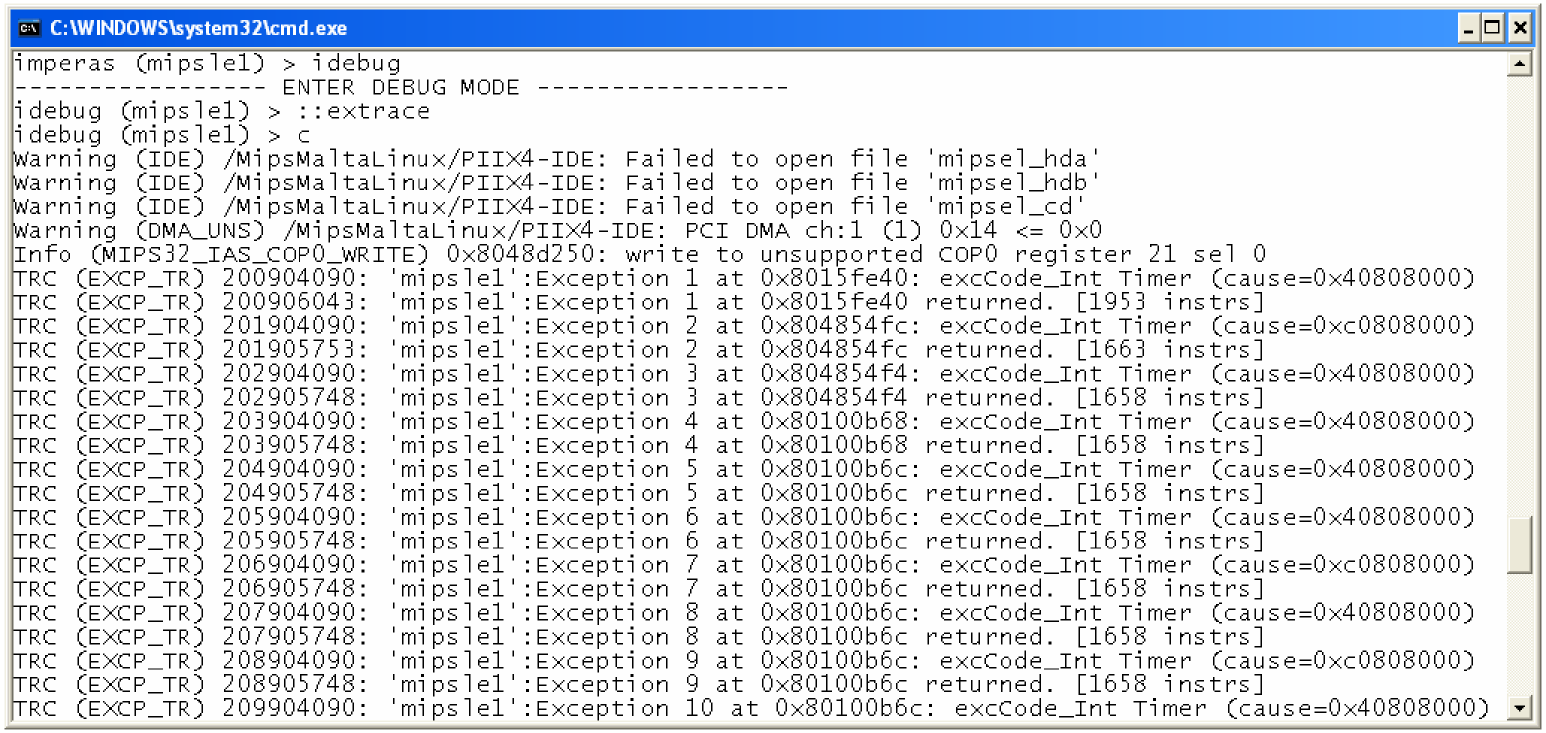
Exception Analysis Tool

- **Adds a user command to enable/disable exception tracing**
- **When notified of an exception entry**
  - **Creates data structure, including instruction count on entry**
- **When notified of an exception exit**
  - **Determines elapsed instructions since entry**
- **Provides report of data collected about the exception event**

Show Exception Analysis Tool

Exception Analysis Tool: Results

- Exception analysis tool is used interactively as application is running
  - Could be used in script
- Reports where exception was taken and returned
- Calculates instructions between exception entry and return

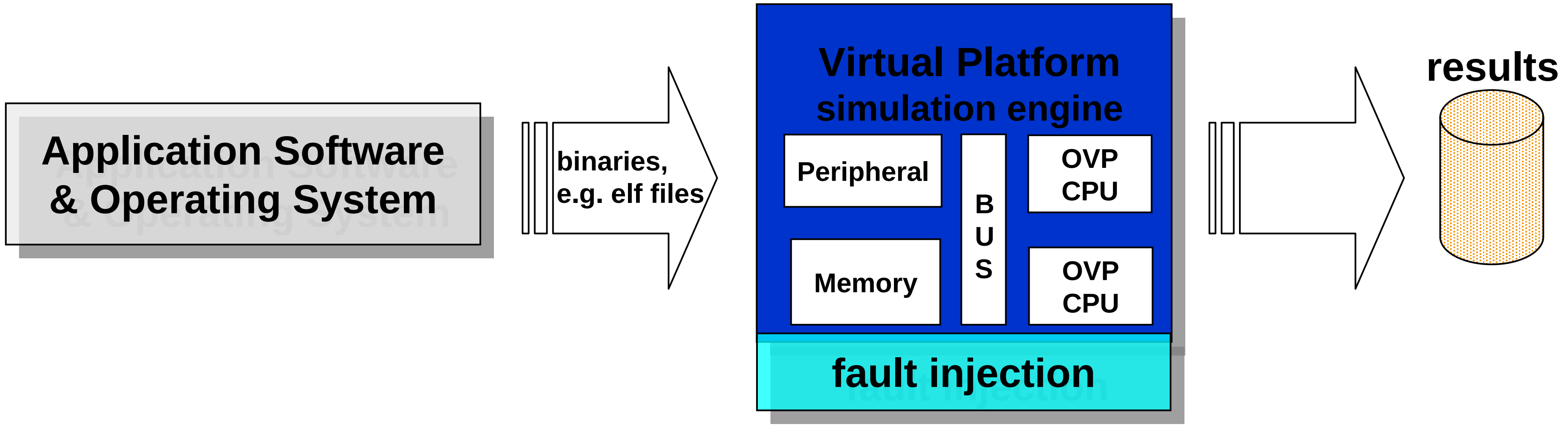


```
imperas (mipsle1) > idebug
----------------- ENTER DEBUG MODE -----------------
idebug (mipsle1) > ::extrace
idebug (mipsle1) > c
Warning (IDE) /MipsMaltaLinux/PIIX4-IDE: Failed to open file 'mipsel_hda'
Warning (IDE) /MipsMaltaLinux/PIIX4-IDE: Failed to open file 'mipsel_hdb'
Warning (IDE) /MipsMaltaLinux/PIIX4-IDE: Failed to open file 'mipsel_cd'
Warning (DMA_UNS) /MipsMaltaLinux/PIIX4-IDE: PCI DMA ch:1 (1) 0x14 <= 0x0
Info (MIPS32_IAS_COP0_WRITE) 0x8048d250: write to unsupported COP0 register 21 sel 0
TRC (EXCP_TR) 200904090: 'mipsle1':Exception 1 at 0x8015fe40: excCode_Int Timer (cause=0x40808000)
TRC (EXCP_TR) 200906043: 'mipsle1':Exception 1 at 0x8015fe40 returned. [1953 instrs]
TRC (EXCP_TR) 201904090: 'mipsle1':Exception 2 at 0x804854fc: excCode_Int Timer (cause=0xc0808000)
TRC (EXCP_TR) 201905753: 'mipsle1':Exception 2 at 0x804854fc returned. [1663 instrs]
TRC (EXCP_TR) 202904090: 'mipsle1':Exception 3 at 0x804854f4: excCode_Int Timer (cause=0x40808000)
TRC (EXCP_TR) 202905748: 'mipsle1':Exception 3 at 0x804854f4 returned. [1658 instrs]
TRC (EXCP_TR) 203904090: 'mipsle1':Exception 4 at 0x80100b68: excCode_Int Timer (cause=0x40808000)
TRC (EXCP_TR) 203905748: 'mipsle1':Exception 4 at 0x80100b68 returned. [1658 instrs]
TRC (EXCP_TR) 204904090: 'mipsle1':Exception 5 at 0x80100b6c: excCode_Int Timer (cause=0x40808000)
TRC (EXCP_TR) 204905748: 'mipsle1':Exception 5 at 0x80100b6c returned. [1658 instrs]
TRC (EXCP_TR) 205904090: 'mipsle1':Exception 6 at 0x80100b6c: excCode_Int Timer (cause=0x40808000)
TRC (EXCP_TR) 205905748: 'mipsle1':Exception 6 at 0x80100b6c returned. [1658 instrs]
TRC (EXCP_TR) 206904090: 'mipsle1':Exception 7 at 0x80100b6c: excCode_Int Timer (cause=0xc0808000)
TRC (EXCP_TR) 206905748: 'mipsle1':Exception 7 at 0x80100b6c returned. [1658 instrs]
TRC (EXCP_TR) 207904090: 'mipsle1':Exception 8 at 0x80100b6c: excCode_Int Timer (cause=0x40808000)
TRC (EXCP_TR) 207905748: 'mipsle1':Exception 8 at 0x80100b6c returned. [1658 instrs]
TRC (EXCP_TR) 208904090: 'mipsle1':Exception 9 at 0x80100b6c: excCode_Int Timer (cause=0xc0808000)
TRC (EXCP_TR) 208905748: 'mipsle1':Exception 9 at 0x80100b6c returned. [1658 instrs]
TRC (EXCP_TR) 209904090: 'mipsle1':Exception 10 at 0x80100b6c: excCode_Int Timer (cause=0x40808000)
```

Run Exception Analysis Demo

Example 4: Fault Injection

- Goal: Use virtual platform controllability to analyze failure modes
  - Virtual platform is able to inject failures that are difficult or impossible to recreate deliberately in the actual hardware
- Fault injection first used for software/system testing around 1970
- Recently elevated in importance in automotive electronics due to ISO 26262 failsafe requirements
  - Also applicable to requirements in aerospace, medical and other applications with critical systems
  - Should be an important tool for testing security in all embedded systems
- Custom tool developed for changing instructions prior to execution on simulator engine



Key Concepts in Fault Injection

- Fault injection: *the deliberate insertion of faults into an operational system to determine its response*
- Triggers: when and where faults are inserted
- Failure modes
  - What the fault does to system to cause an error to occur
  - For example: locking, drifting, oscillation, delay, …
- Observation: how the failure response is determined

- Fault injection in virtual platform environments can be black box or white box
  - Black box
    - Treat the system as a black box, and just perturb from the outside
    - This is the way hardware based fault injection works
  - White box
    - Open up the system
    - Enables full control over triggering
      - Can be predetermined/random, event/timing/instruction triggered, …
      - Can focus on specific failure modes if needed
    - Important to perturb only what you want to perturb
      - By implementing changes (faults) at the simulator engine, and not in the application source, full controllability is achieved
- Virtual platforms also enable complete observability of failure caused responses

Fault Injection Custom Tool

- Instructions are "intercepted" at simulation engine prior to execution
  - Instructions can be changed before execution
  - Complete control over generation of faults
- White box fault injection
  - Randomly trigger corruption of an instruction
  - Randomly corrupt an individual bit of an instruction
  - Reports when / where fault was injected

```
Execution starts, 2000000 runs through Dhrystone
FAULT 0x001002a4(  3004905): 0x0000580d -> 0x00005805(^bit= 3) (     mov) 16 Bits
FAULT 0x00100b8c( 12312824): 0x000059e8 -> 0x000059f8(^bit= 4) (     cmp) 16 Bits
FAULT 0x00100b8c( 17912768): 0x000059f8 -> 0x00005bf8(^bit= 9) (     cmp) 16 Bits
FAULT 0x00100b7e( 19676187): 0x00006f0c -> 0x00016f0c(^bit=16) (    ld.b) 32 Bits
FAULT 0x00100b86( 26529726): 0x00006f4c -> 0x20006f4c(^bit=29) (    st.b) 32 Bits
FAULT 0x00100b7a( 34399330): 0x00006007 -> 0x00006087(^bit= 7) (     mov) 16 Bits
FAULT 0x00100b8e( 38537367): 0x0000f5ea -> 0x0000f5e2(^bit= 3) (       b) 16 Bits
FAULT 0x00100a94( 47901218): 0x0015e763 -> 0x001de763(^bit=19) (    st.w) 32 Bits
FAULT 0x0010021a( 52556124): 0x0001ef6a -> 0x0000ef6a(^bit=16) (    st.w) 32 Bits
FAULT 0x001000e6( 60904500): 0x00115640 -> 0x00515640(^bit=22) (   movhi) 32 Bits
```

Virtual Platform Based Software Testing Enhances Current Methodology for Automotive and Other Embedded Systems

- Simulation (virtual platforms) enables full visibility, controllability of software
- Tools are needed – more than just simulation – to deliver on the promise of visibility, controllability
- Verification, analysis and profiling tools for virtual platforms provide complementary capability (white box testing) to existing test methodology

DEVCON
Enabling the Smart Society

MULTICORE DESIGN SIMPLIFIED
imperas

Questions?

Please Provide Your Feedback…

- Please utilize the 'Guidebook' application to leave feedback

Or

- Ask me for the paper feedback form for you to use…

DEVCON
Enabling the Smart Society

MULTICORE DESIGN SIMPLIFIED
imperas

RENESAS

Renesas Electronics America Inc.