Checkpointing

Jakob Engblom, Håkan Zeffer,
Eric Nilsson, Philipp Hartmann

Intel Sweden & Germany

SYSTEMC™ EVOLUTION DAY
OCT 18, 2017 | MUNICH | GERMANY

Presentation Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Intel** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

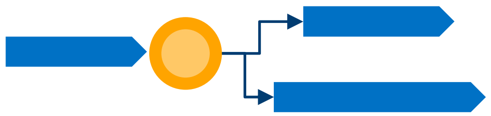
CHECKPOINTING BASICS

Q: What is Checkpointing in a Virtual Platform?

A: The ability to save the state of a simulation and later pick up at the exact same point

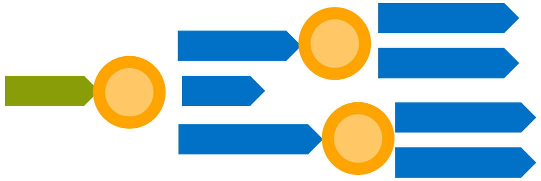
Checkpointing Use Cases

“Save the boot”



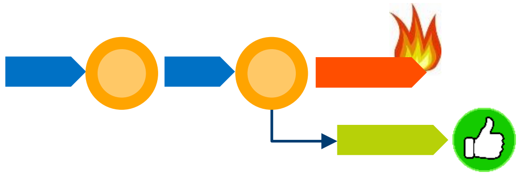
Avoid redoing work, save booted & configured system for reuse and distribution

Parallelize Test execution



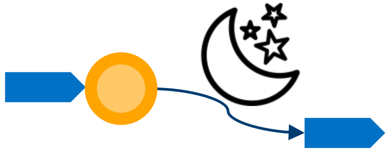
Save checkpoints and spin up additional parallel simulations during testing

Undo target actions



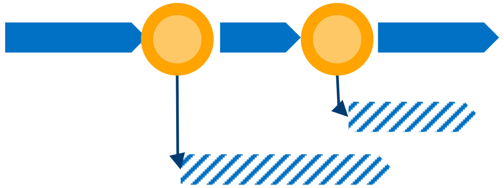
Get back to a previous good state after something went wrong

Save for the day



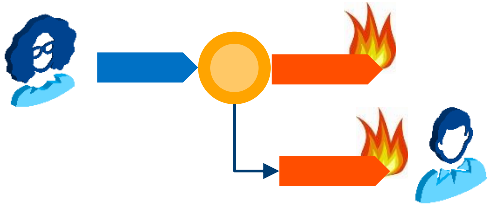
Save work, shut down simulation, continue the next day

Gear shifting



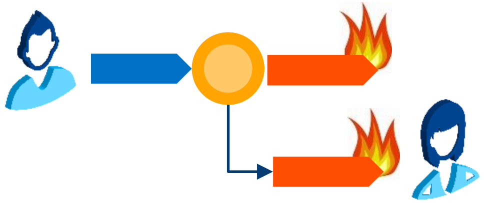
Save state from fast VP, open in a detailed separate model

Report software bugs



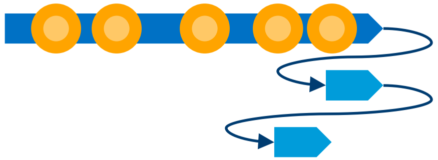
Save state of system when bug hits, transfer to SW developer for analysis

Report model bugs



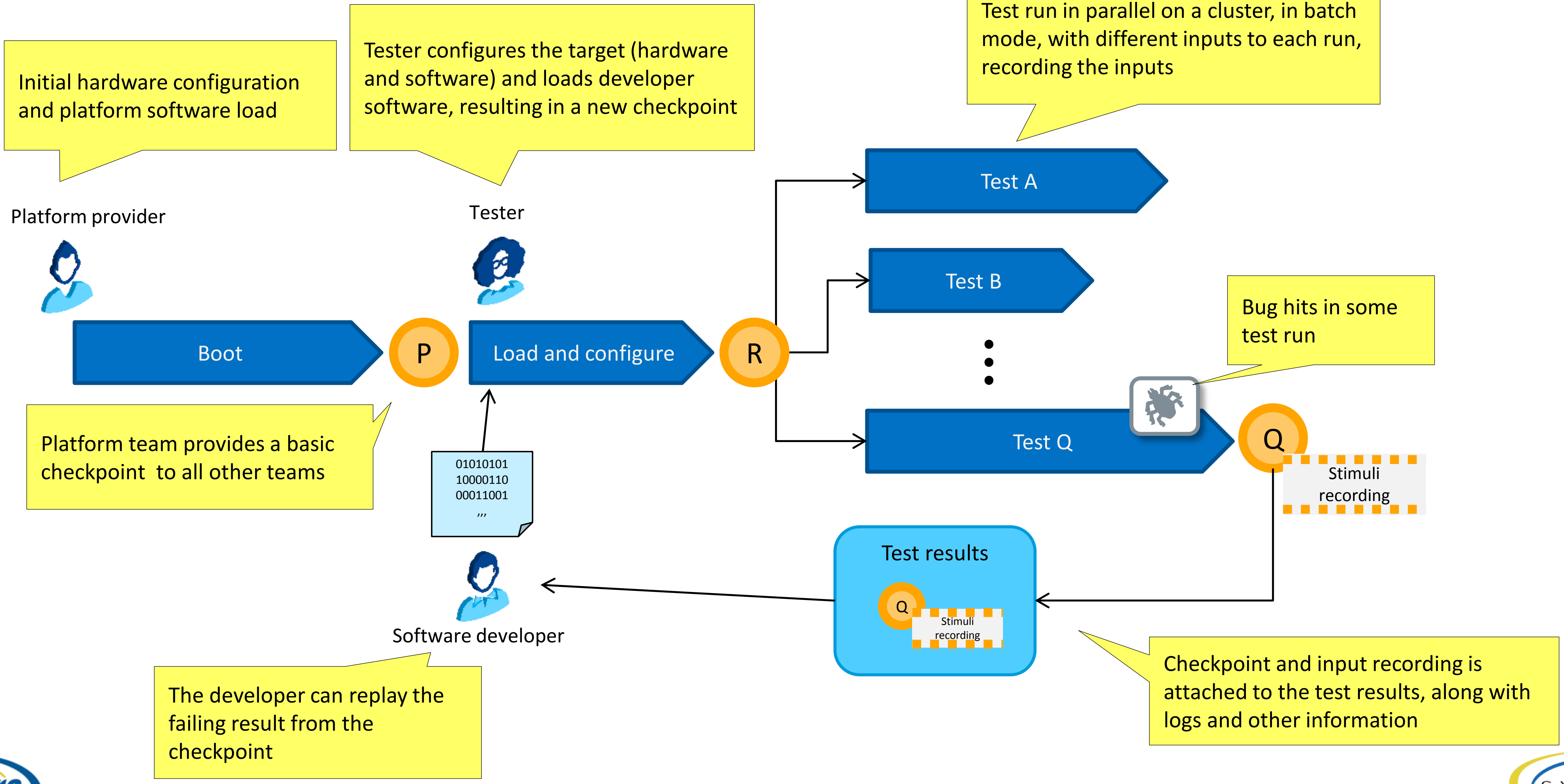
Provide HW + SW state when a model bug hits to model developer for analysis

Reverse Debugging



Support the implementation of reverse debugging (reproduction-based)

Example: Feedback from Automatic Testing



Initial hardware configuration and platform software load

Tester configures the target (hardware and software) and loads developer software, resulting in a new checkpoint

Test run in parallel on a cluster, in batch mode, with different inputs to each run, recording the inputs

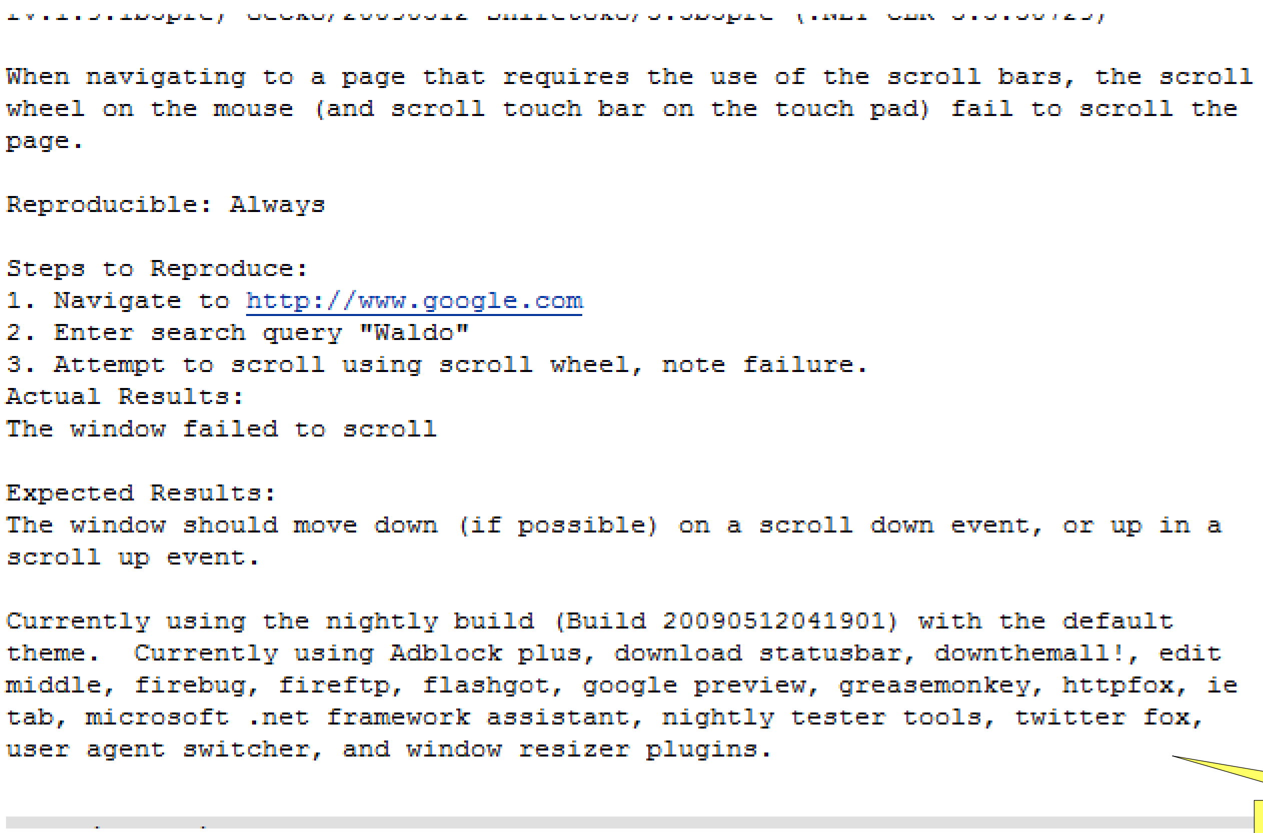
Bug hits in some test run

Platform team provides a basic checkpoint to all other teams

Checkpoint and input recording is attached to the test results, along with logs and other information

The developer can replay the failing result from the checkpoint

Checkpoint vs Bug Report – Reproducibility!



```
When navigating to a page that requires the use of the scroll bars, the scroll
wheel on the mouse (and scroll touch bar on the touch pad) fail to scroll the
page.

Reproducible: Always

Steps to Reproduce:
1. Navigate to http://www.google.com
2. Enter search query "Waldo"
3. Attempt to scroll using scroll wheel, note failure.
Actual Results:
The window failed to scroll

Expected Results:
The window should move down (if possible) on a scroll down event, or up in a
scroll up event.

Currently using the nightly build (Build 20090512041901) with the default
theme.  Currently using Adblock plus, download statusbar, downthemall!, edit
middle, firebug, fireftp, flashgot, google preview, greasemonkey, httpfox, ie
tab, microsoft .net framework assistant, nightly tester tools, twitter fox,
user agent switcher, and window resizer plugins.
```

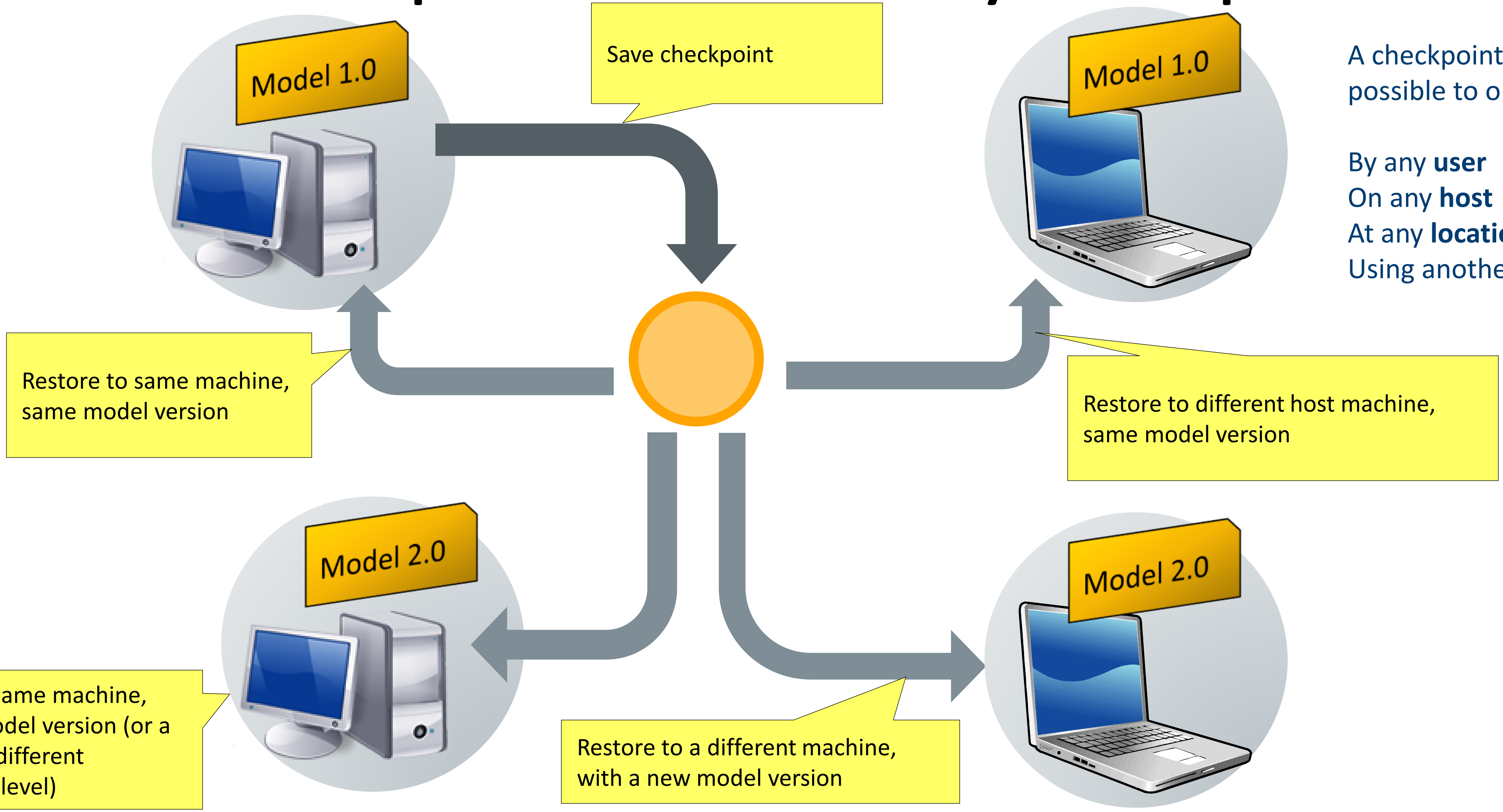
Provide steps to reproduce the bug

Try to capture all relevant aspects of the software environment…

Pity the developer who tries to reproduce this

https://bugzilla.mozilla.org/show_bug.cgi?id=492885

Note: Checkpoint Portability is Important


Model 1.0
Save checkpoint
Model 1.0
Restore to same machine, same model version
Restore to different host machine, same model version
Model 2.0
Model 2.0
Restore to same machine, updated model version (or a model at a different abstraction level)
Restore to a different machine, with a new model version

A checkpoint should be possible to open...

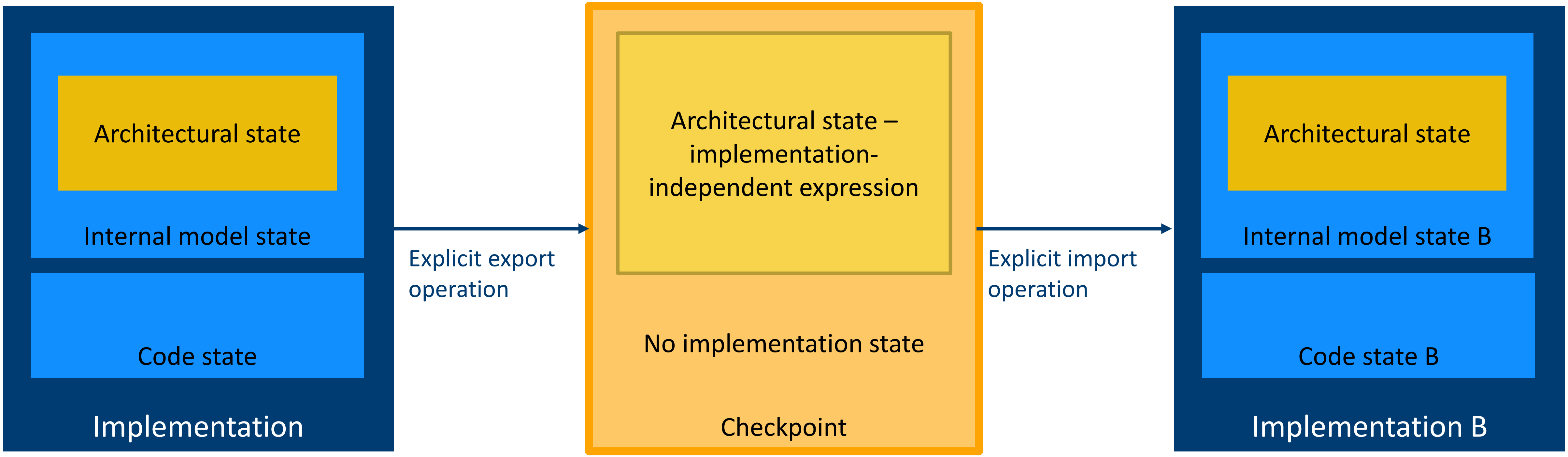
By any **user**
On any **host**
At any **location**
Using another **model version**

Note: Repeatability, Determinism, Variability

- For most use cases, repeatability provides great value
  - Requires checkpointing + determinism + a way to record inputs
- Determinism vs variability
  - Determinism is the key simulator implementation property
  - Note that determinism does not preclude variability – just vary the inputs
- Note: Checkpoints and determinism are independent concepts
  - A deterministic simulation might not be possible to checkpoint
  - A checkpoint might lead to another execution if simulator is not deterministic

CHECKPOINTING CORE CONCEPTS

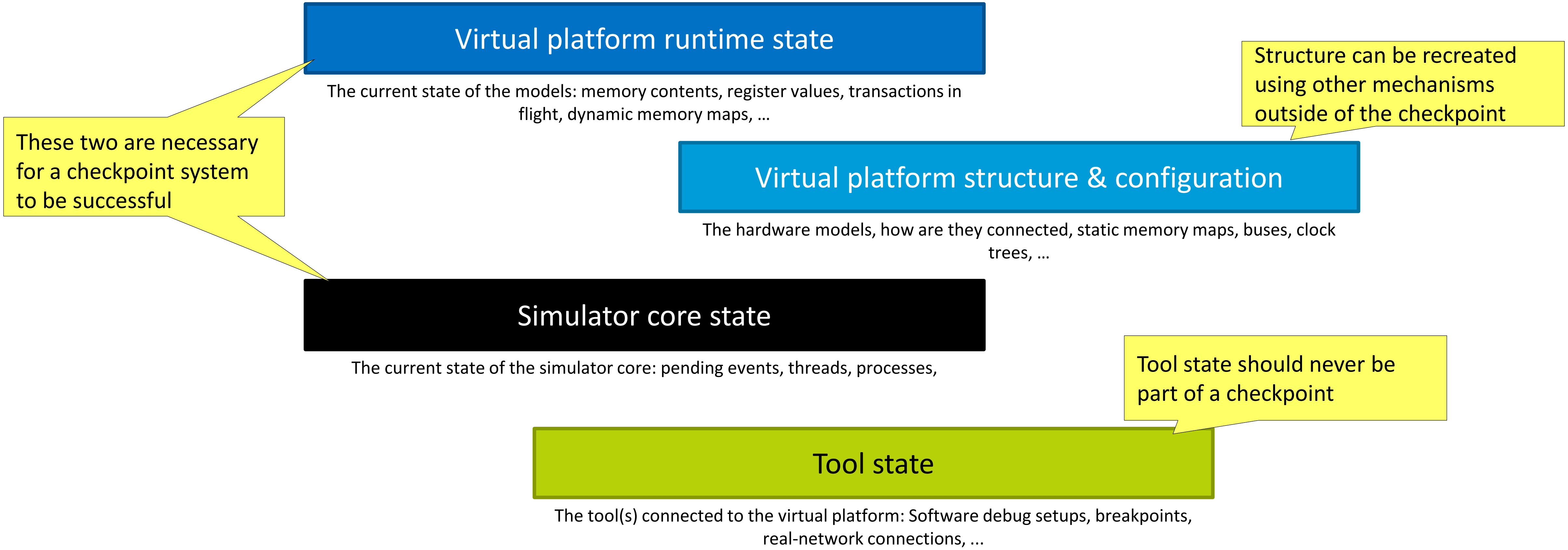
Core: Implementation State ≠ Checkpoint State



The architectural state is part of the internal model state. More or less explicit, depends on the model, the model can do pretty much anything it wants to store it.

When opening the checkpoint, the model state is built from the architectural state in the checkpoint. The model state is usually bigger than the architectural state.

Aspects of Checkpoint State



Virtual platform runtime state

The current state of the models: memory contents, register values, transactions in flight, dynamic memory maps, …

Structure can be recreated using other mechanisms outside of the checkpoint

These two are necessary for a checkpoint system to be successful

Virtual platform structure & configuration

The hardware models, how are they connected, static memory maps, buses, clock trees, …

Simulator core state

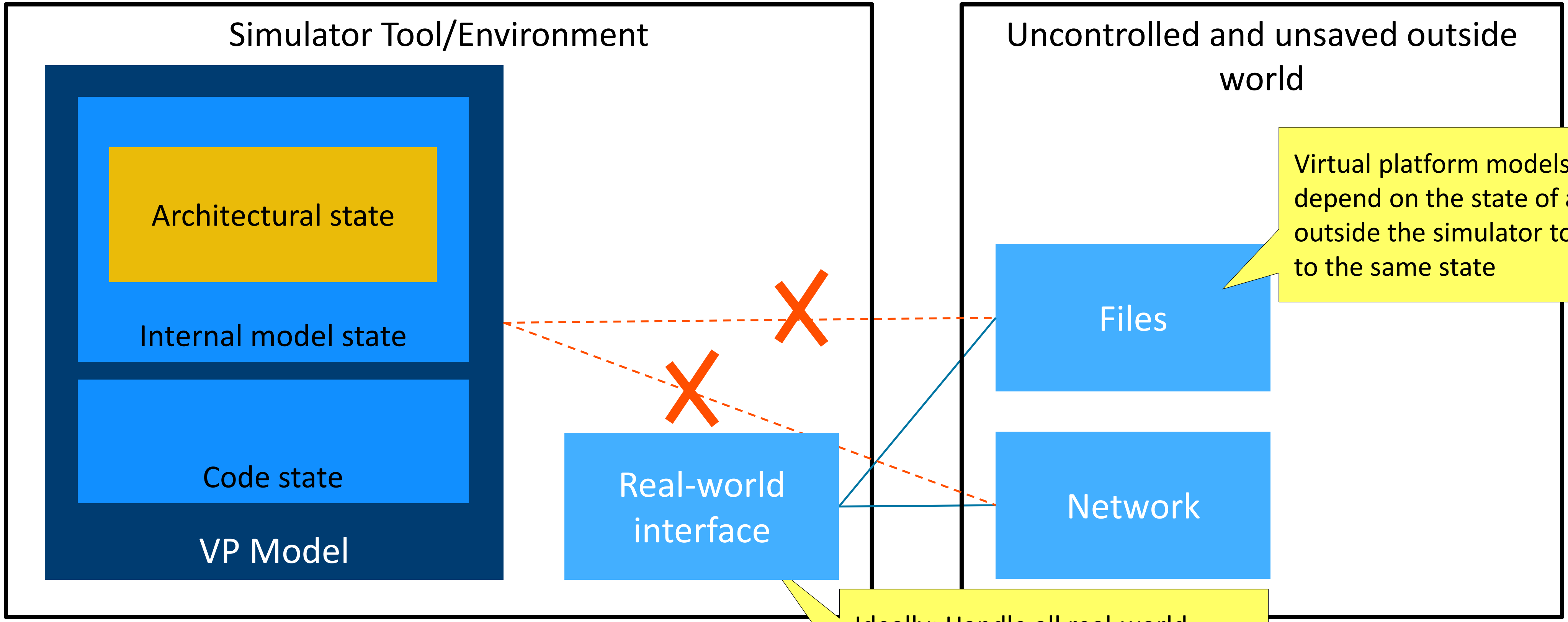
The current state of the simulator core: pending events, threads, processes,

Tool state should never be part of a checkpoint

Tool state

The tool(s) connected to the virtual platform: Software debug setups, breakpoints, real-network connections, ...

Isolation from Host and Identification of State



Virtual platform models must not depend on the state of anything outside the simulator to get back to the same state

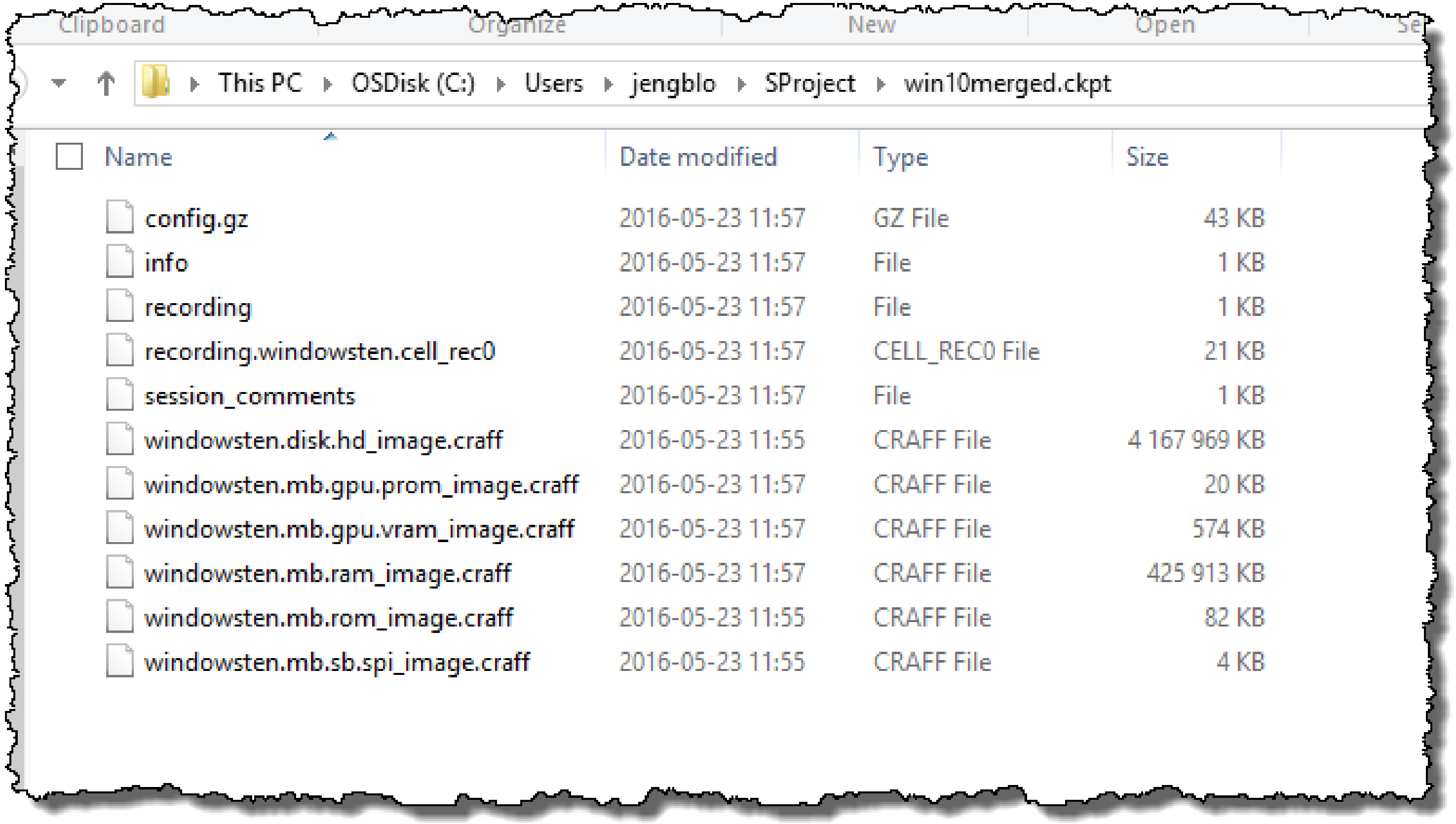
Ideally: Handle all real-world interactions using specialized modules that isolate the model from the outside world

(Simics supports all the use cases presented initially)

WIND RIVER® SIMICS® AS AN EXAMPLE IMPLEMENTATION

Wind River® Simics®: Checkpoints = Folder

- config[.gz]
  - Contains Simics configuration, paths to previous, checkpoints, names of image files, ...
  - Compressed by default
- info
  - Metadata
- recording
  - optional
- session_comments
  - optional
- *.craff files for all images



This PC ▸ OSDisk (C:) ▸ Users ▸ jengblo ▸ SProject ▸ win10merged.ckpt

Name	Date modified	Type	Size
config.gz	2016-05-23 11:57	GZ File	43 KB
info	2016-05-23 11:57	File	1 KB
recording	2016-05-23 11:57	File	1 KB
recording.windowsten.cell_rec0	2016-05-23 11:57	CELL_REC0 File	21 KB
session_comments	2016-05-23 11:57	File	1 KB
windowsten.disk.hd_image.craff	2016-05-23 11:55	CRAFF File	4 167 969 KB
windowsten.mb.gpu.prom_image.craff	2016-05-23 11:57	CRAFF File	20 KB
windowsten.mb.gpu.vram_image.craff	2016-05-23 11:57	CRAFF File	574 KB
windowsten.mb.ram_image.craff	2016-05-23 11:57	CRAFF File	425 913 KB
windowsten.mb.rom_image.craff	2016-05-23 11:55	CRAFF File	82 KB
windowsten.mb.sb.spi_image.craff	2016-05-23 11:55	CRAFF File	4 KB

Wind River® Simics®: Configuration Files

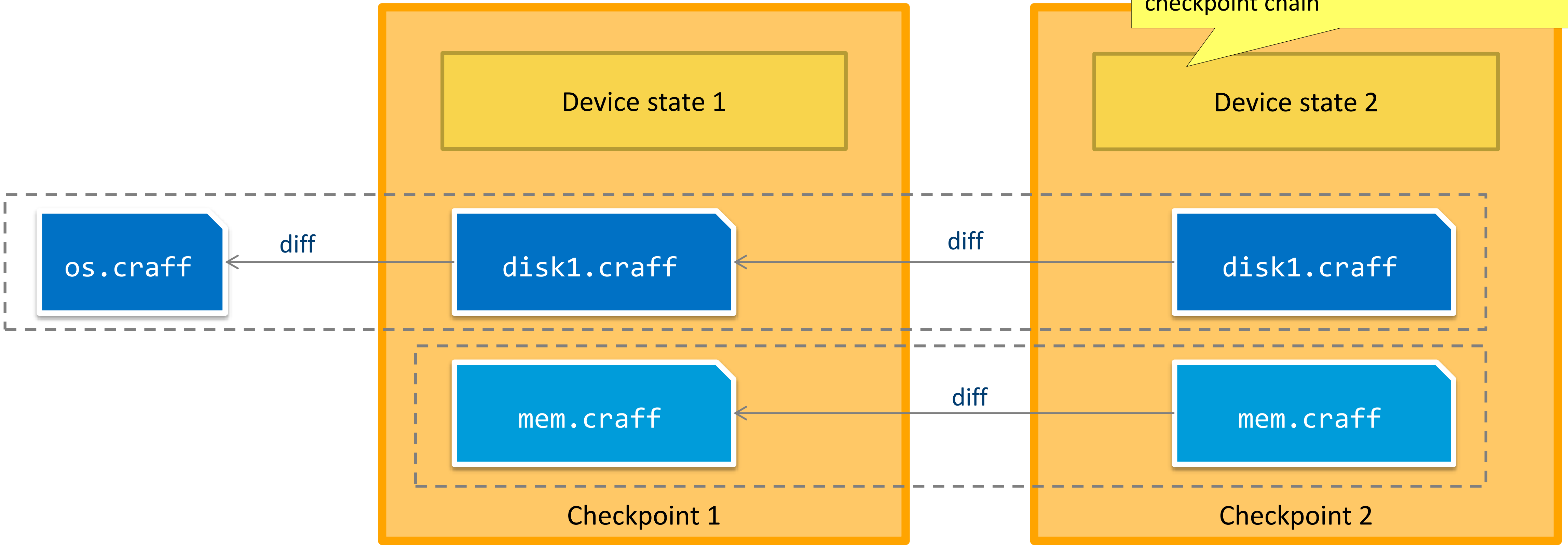
- Captures structure & state & simulator core state
- Human-editable
  - Checkpoint files are plain text files
  - Can be edited for fixing & experiments
- Data stored as Simics attributes
  - Key-value pairs
  - Similar set of types as CCI configuration parameters
  - Based on Simics object system

Example from a DMA controller

```
OBJECT ubuntu.mb.sb.dma TYPE i8237x2 {
         queue: ubuntu.mb.cpu0.core[0][0]
         object_id: "obj_0000018183f6616d"
         build_id: 0x13f0
         memory: ubuntu.mb.nb.pci_mem
         current_addr: ((0,0,0,0),(0,0,0,0))
         base_addr: ((0,0,0,0),(0,0,0,0))
         current_count: ((0,0,0,0),(0,0,0,0))
         base_count: ((0,0,0,0),(0,0,0,0))
         disabled: (0,0)
         mask: ((1,1,1,1),(0,1,1,1))
         flip_flop: (0,0)
         dec_address: ((0,0,0,0),(0,0,0,0))
         auto_init: ((0,0,0,0),(0,0,0,0))
         dma_type: ((0,0,0,0),(0,0,0,0))
         dma_mode: ((0,0,0,0),(3,0,0,0))
         request: ((0,0,0,0),(0,0,0,0))
         tc: ((0,0,0,0),(0,0,0,0))
         page_addr: ((0,0,0,0),(0,0,0,0))
         page_size: (0x10000,0x20000)
         extra_page_addr: ((0,0,0,0),(0,0,0,0))
}
```

Wind River® Simics®: Images handled using diffs

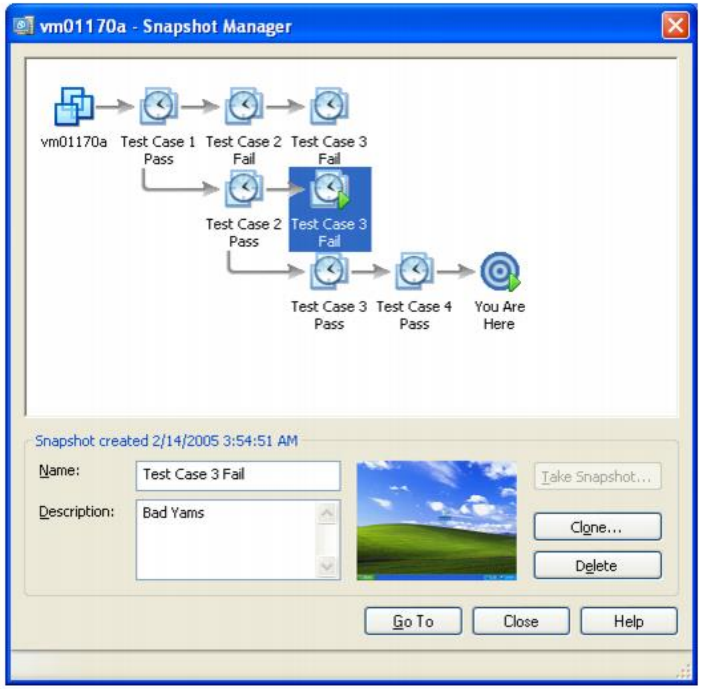
When opening checkpoint 2, Simics uses the device state from Checkpoint 2 + the cumulative disk and memory state from the checkpoint chain



RELATED TECHNOLOGIES

Virtual Machine Snapshots

- VMWare*, Virtualbox*, etc. “snapshots” are equivalent to checkpoints
  - (Mostly) portable
  - Typically heavier than Wind River® Simics® checkpoints (longer save, bigger size)
- Sometimes used to checkpoint simulators
  - Run inside of VM, save the VP along with its engine and the OS it is running on
  - Supports “save”, “undo”, “reproduce”



Snapshot created 2/14/2005 3:54:51 AM

Name: Test Case 3 Fail

Description: Bad Yams

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

Process checkpointing

- Used in high-performance computing (HPC) to save process state for long runs (allow recovery)
  - Save entire process space as maintained by the operating system (OS) to disk and reload
  - Expects same host, same precise host configuration, same binaries
  - Interesting effects if network connections are left open during save
  - Supports ”save your work” and ”undo bad actions”, but:
    - No portability across hosts
    - No compatibility across versions
    - No opportunity for gear-shifting
- Used in some commercial tools & virtual platform setups

Persistence and Serialization

- For example, C++ Boost* serialize
- Save state of a set of [C++, Java*, ...] objects to disk for later reloading into memory
  - Explicit export and import steps
  - Execution threads and variables not saved – only contents of objects
  - No pointers saved – only symbolic reference to other objects
  - Allows portability across hosts
    - Provided data is not saved in “dumb” ways, like binary blobs with data endianness

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

Reverse Execution & Record/Replay Systems

- Save traces of execution to disk for replay & review in debugger
  - Saves a particular concrete execution
  - Cannot continue execution from saves file
  - Supports bug reporting and replay use cases

- For example: Undo* LiveRecorder*, Microsoft* WinDbg* Time Travel Debug, Mozilla* RR*, Intel® PinPlay*

A current proof-of-concept implementation

WIND RIVER® SIMICS®
SYSTEMC CHECKPOINT LIBRARY

Tricky Stuff in SystemC

- Threads
  - Implicit state: the current point in the code (current wait())
- Stack-based storage of state
  - Tightly coupled to a particular compilation of a particular code version (and often implicit)
- Identification of state
  - Which class members and other variables constititute the state?
- Pointers between objects
  - Depends on the details of the machine state when program runs
- Endianness & word length
  - Data has to be neutral to host endianness, word length, and compiler data size choices
- Target system structure
  - Embedded in code or data-driven?

SystemC* Checkpoint Library

- Simulator-independent design
  - Gives the user the tools to write checkpointable models
  - Gives the user the tools to write a checkpointable SystemC kernel

- Implementations:
  - Checkpoint support added to Intel-internal SystemC kernel
  - Used in stand-alone execution (standard SystemC executable)
    - Call to save checkpoints integrated into the model code (like sc_main())
  - Used by the Wind River® Simics® SystemC* Library
    - Saves state into Simics checkpoint file structure, invoked from Simics

Checkpoint Library Design: Serializer

- Based on Boost serialize
  - Additional mechanisms available to deal with large data images
- SystemC modules:
  - Instantiate a **Serializer** class
    - Automatically found by framework
    - Provides the **serialize()** function
    - **serialize()** function lists all non-SystemC-module-children to include them
- Non-SystemC modules
  - Add a **serialize()** function to expose the state

- SystemC threads:
  - When a thread is (re)started, check current state in the module and somehow get to the right wait()
  - Note: Restart can happen multiple times in a session in case of reverse execution
  - Note: Restarting a feature of the patches SystemC kernel

Checkpoint Library Design: Loading

- When loading a checkpoint, the following happens:
  1. Create structure (just like when the model is created normally)
  2. Set the state (as on previous slide)
  3. Restart all SC_THREADs
- Note that for reverse execution, 2 & 3 can happen many times within a simulation run!

Checkpointing & Modeling Libraries

- Using a [TLM] modeling library can make checkpointing easy
  - Code registers, attributes (or other state container), etc., using library constructs
  - Write SC_METHOD & SC_THREAD according to guidelines

  - Library automatically creates the infrastructure for save and restore of state
  - Library automatically hooks models into checkpointing central driver

Checkpoint Library Notes

- All stacks of all threads are gone
  - State has to be kept in variables that can be serialized
- State of SystemC modules is saved in JSON format files
  - Inside of Simics checkpoint directories for the Simics case
- SystemC kernel patches currently not in mainline kernel
- Branched boost library used to avoid name collisions

GOING FORWARD

SystemC CCI – Basic Mechanisms

- CCI parameters essentially provide the core mechanism needed to represent state outside of a running model
  - Host-independent expression of data values with a name
  - Name-value mappings and a type system for values

- Still need to deal with kernel, structure, and making sure all state gets saved and restored

To Do (or at least Discuss)

- **There is no magic!**
  - Models will have to be adapted to support checkpointing

- In particular:
  - Keep model state explicit and separate from implementation
  - Deal with thread positions
  - Deal with in-place state changes
  - Avoid keeping things on the stack

THE END