Selections from the Simics Users Workshop

Stockholm, June 28th, 2001

Peter S. Magnusson
CEO, Virtutech AB

virtutech

Digital Challenges

- Software complexity
  - Even ”low-end” products
- Multi-sourcing of HW & SW
- Realism of HW evaluation/design/test
- Time-to-market pressure
  - Need to parallelize/accelerate development

virtutech

slide 3

Virtutech's Vision

"In the future, full-system simulation will be a core technology for the design, development, and testing of computer hardware and software.

"It will revolutionize the quality and time-to-market of complex digital systems.

"Virtutech intends to be the leading player in that development."

  virtutech

slide 4

Background

- Analyzing Digital Systems
- Simulation
- Simics Objectives
- Simics Internals

  virtutech

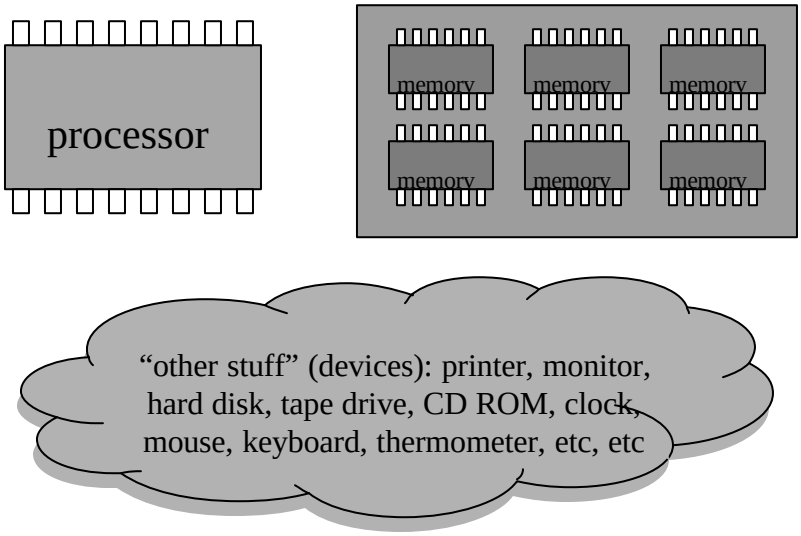
slide 5

What is a Computer?

 Simics Overview, June 2001 virtutech

slide 6

What is a Computer?



 Simics Overview, June 2001 virtutech

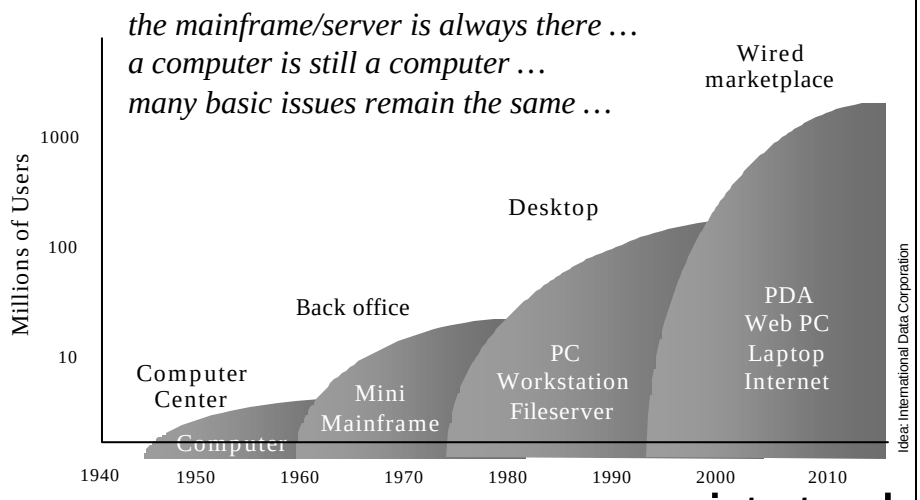
slide 7

Everything is going digital

 Simics Overview, June 2001 virtutech

slide 8

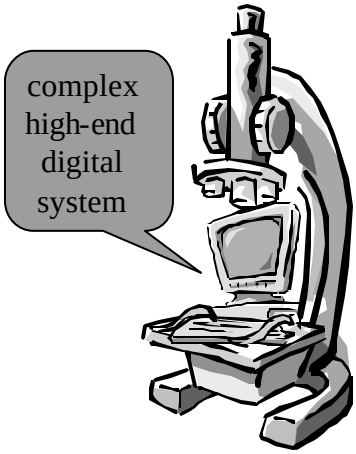
Computer Evolution



 Simics Overview, June 2001 virtutech

slide 9

The Problem



- Design
- Implement
- Test
- Deploy
- Analyze

 Simics Overview, June 2001 virtutech

slide 10

Requirements

- Generality
  - can the tool/method analyze my workloads?
  - typical stumbling blocks: parallel systems, multithreading, multiple address spaces, operating system code, self-modifying code, networked systems, erroneous code, binary-only code, large code/data set volumes
- Practicality
  - can the tool/method be used in an effective manner?
  - typical stumbling blocks: host assumptions, compiler assumptions, operating system modifications, workload language assumptions
- Applicability
  - can the tool/method answer my questions?
  - typical stumbling blocks: restricted state that can be monitored, restrictions on parameter variability, restricted length of observations

 Simics Overview, June 2001 virtutech

slide 11

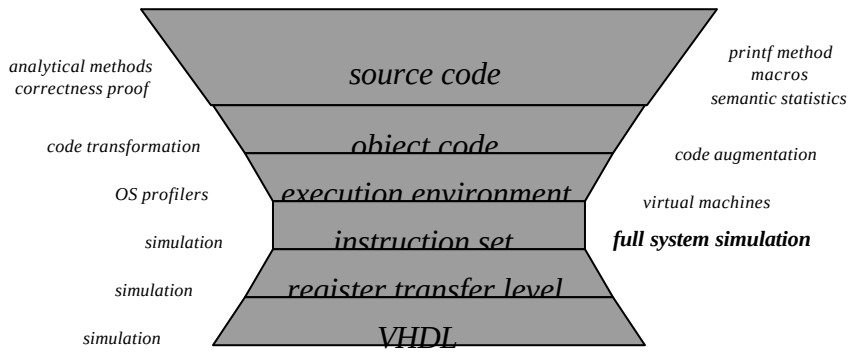
Analyzing complex digital systems

- The real thing
  - Hardware
    - built-in counters
    - probe
    - ...
  - Software
    - modify executable
    - "supervise" executable
    - ...
- Virtualize
  - Math
    - analytical models
    - statistical models
    - ...
  - Simulation
    - emulate user-level
    - full system simulation
    - ...

 Simics Overview, June 2001 virtutech

slide 12

Simulation: SW studying SW



Instruction set level is essentially the level where software engineers interface with hardware engineers. Therefore, it is the lowest level accessible to software, and at the same time it is the best defined, least complicated, and most stable level.

 Simics Overview, June 2001 virtutech

slide 13

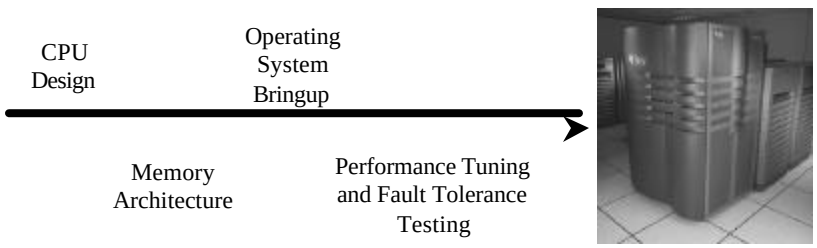
Full system simulation

- "Adequate" level of detail
  - complete functional model of instruction set, interrupt model, MMUs, etc
  - simple memory model
  - simple I/O model
- Accurate
  - all hardware interfaces are represented
  - timing accuracy can be added as needed
- Non-intrusive
  - measuring does not affect measurement
- Explicit
  - target is represented in software constructs
  - automation, scripting
  - modify, instrument, control, freeze, save state to file, etc
- Scales
  - large systems, SMPs, clusters, networks

virtutech

slide 14

Server Development I

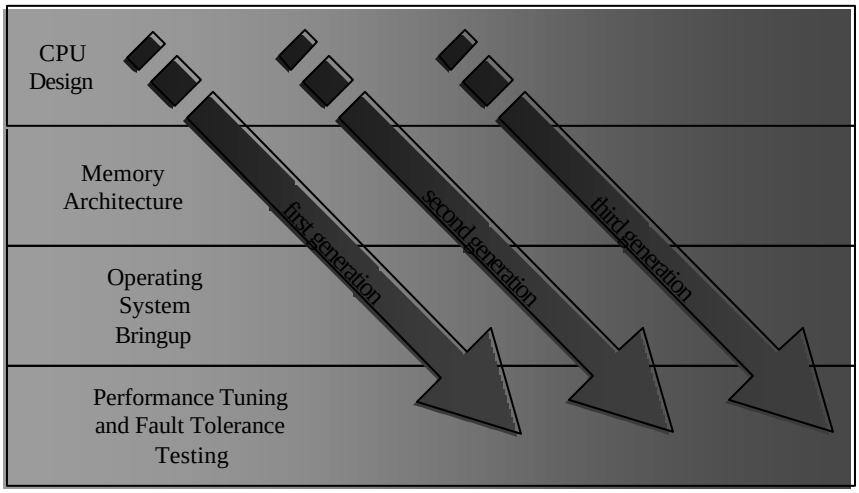


- *Multiple user communities for full system modeling*
- *Overlapping requirements*
- *Natural flow*

virtutech

slide 15

Server Development II



 Simics Overview, June 2001 virtutech

slide 16

Simics Objectives

- Complete: model the entire system
- Accurate: real binaries
- Fast: realistic workloads
- Scalable: large workloads
- Useful: deterministic, extensible, portable
- Tolerant: basic tasks should not slow it down
- Flexible: multiple tasks
- Practical: good at getting typical stats

 Simics Overview, June 2001 virtutech

slide 17

Virtutech Simics™

- Full system simulator at instruction set level
- Advanced instrumentation capabilities
- Very fast: 20x-150x “slowdown”
- Multi-target: Sparc V9, Alpha, x86, Hammer, PPC
- Multi-host: Solaris, Linux, Tru64, Win2K
- Platform toolbox: broad range of uses
- Leading technology

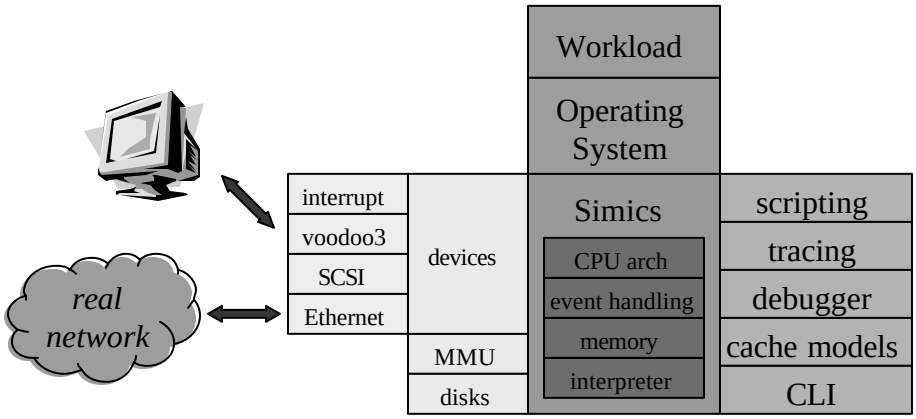
  virtutech

Simics

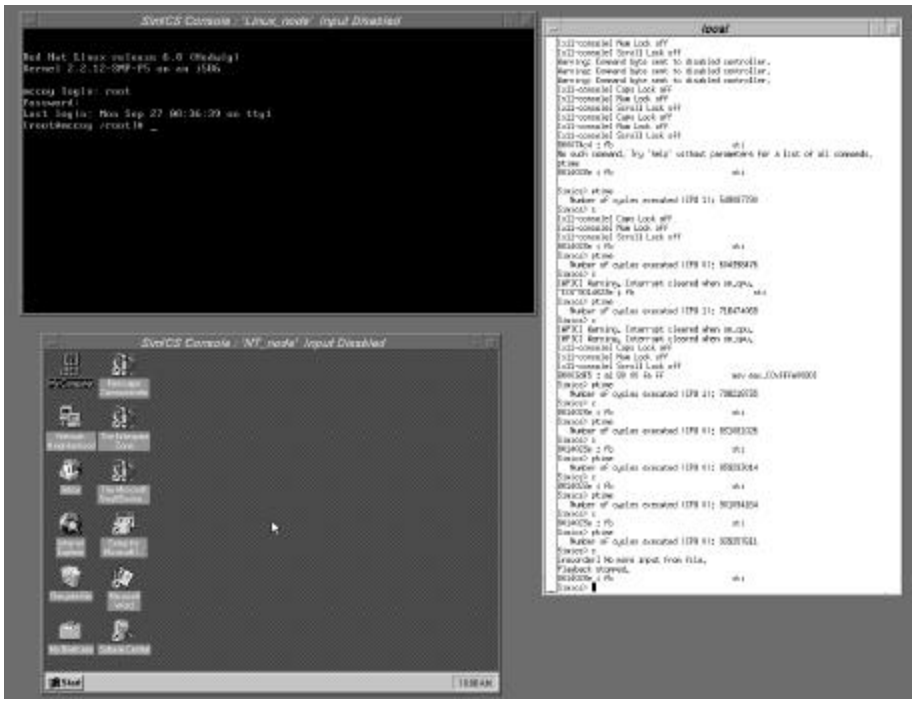
virtutech

slide 19

Virtutech Simics Architecture

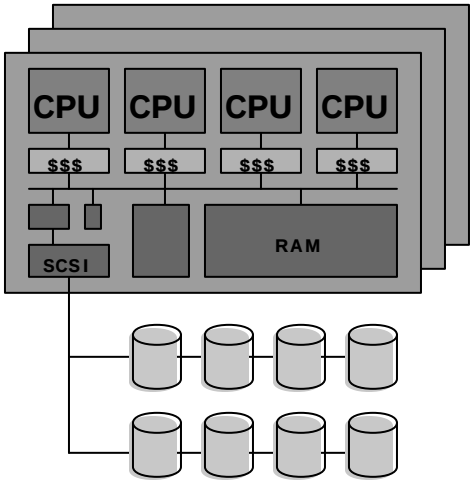


Simics Overview, June 2001
virtutech



slide 21

Configurable Target

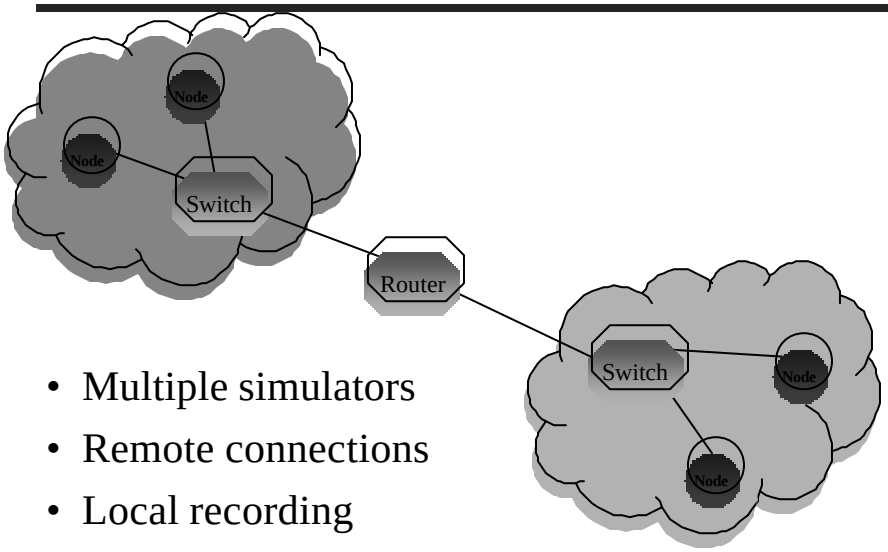


- Arbitrary configurations
- Multiple processors
- Multiple address spaces
- Arbitrary device models
- Relatively easy to add new devices
- Networking, clusters, etc

 Simics Overview, June 2001 virtutech

slide 22

Multi-Node/Multi-Net Simulation



- Multiple simulators
- Remote connections
- Local recording

 Simics Overview, June 2001 virtutech

slide 24

Simics Design Philosophy

- Careful design to optimize common case
- Very fast core algorithms for key areas
- Tools to facilitate implementation
- Support for efficient instrumentation permeates design
- Flexible framework for adding devices, support for new processors, end-user extensions, etc
- Highly dynamic and run-time configurable
- Effectiveness preferred over efficiency

 Simics Overview, June 2001 virtutech

slide 25

System requirements (host)

- Fast CPU, ½ Gig memory, lots of disk
- Currently supported platforms:
  - Linux on x86
  - Solaris 7 / 8 on Sparc
- Real soon now support
  - Windows 2000 (x86)
- Other ports performed:
  - Tru64 Alpha
  - Yellowdog Linux 1.2.1 (Mac Cube!)
  - Linux Alpha

  virtutech

slide 26

Simics Targets

- Simics/UII (SunFire)
- Simics/x86 (PC)
- Simics/x86-64 (“Hammer”)
- Simics/Alpha
- Simics/PPC
- Simics/UIII (Serengeti)
- *Coming soon: Simics/IA64, Simics/ARM, Simics/MIPS*

  virtutech

slide 27

Part III: Applications

- Overview of Current Applications
- Debugging
- Performance Analysis
- High-Availability Testing
- Fault Injection with Checkpointing
- HW/SW Co-development

 Simics Overview, June 2001 virtutech

slide 28

Current Simics™ Applications

- Microprocessor design work
  - *generating realistic traces, interfacing a detailed processor model*
- System design work
  - *memory hierarchy, coherency protocols, device design, firmware development*
- System software
  - *OS porting, firmware, drivers, bring-up, debugging*
- Performance tuning
  - *hardware/software interaction, event profiling*
- Reliability, Availability, Serviceability
  - *fault injection framework, stress testing*
- Hardware/Software co-development
  - *combining different levels of fidelity; superset of other apps*

 Simics Overview, June 2001 virtutech

slide 29

App: Debugging

- Arbitrary code can run
  - independent of language, compilers, etc
- Checkpointing allows repeatability
  - entire state saved to file
- Cross-platform
  - multi-host/multi-platform
- Breakpoints
  - memory, control registers, time, hw events

  virtutech

slide 30

App: Performance Analysis

- Relate hardware events to code
  - data caches
  - instruction caches
  - device operations
  - user-defined (open API)
- Vary hardware parameters
- Timing accuracy can be varied
  - coarse accuracy is valuable

  virtutech

slide 31

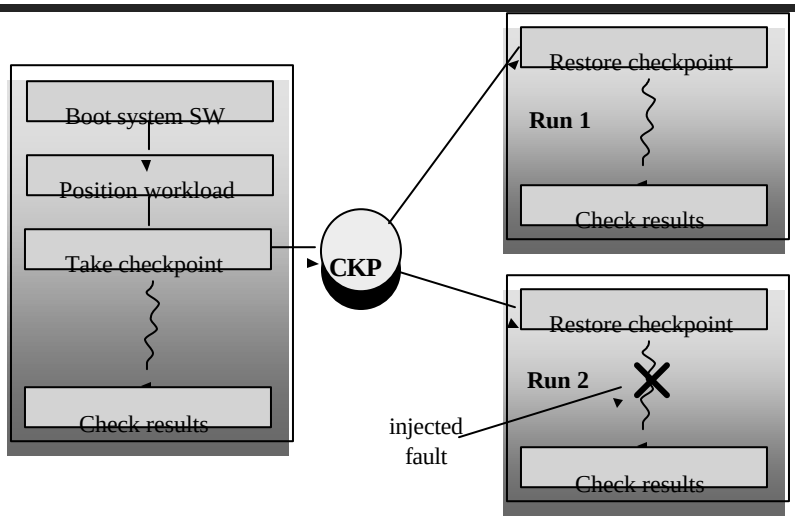
App: High-Availability Testing

- Virtualization of complex, heterogeneous, systems
- Decouples host from target
- Systematically inject errors
- Inject rare but valid events
- Snapshot-error-state check cycles
- Explore valid HW parameter space
- Farm out tests across parameter space
- Fully replicateable

 Simics Overview, June 2001 virtutech

slide 32

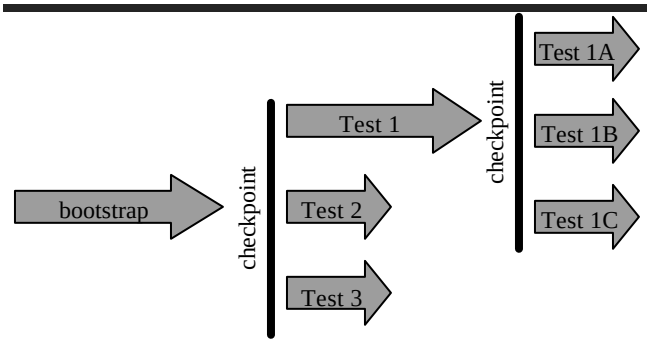
App: Fault Injection with Checkpointing



 Simics Overview, June 2001 virtutech

slide 33

Exploiting Parallelism



- Leverage determinism & ”wrappability”
- Tests can be automated etc
- Re-use same framework throughout development

 Simics Overview, June 2001 virtutech

slide 34

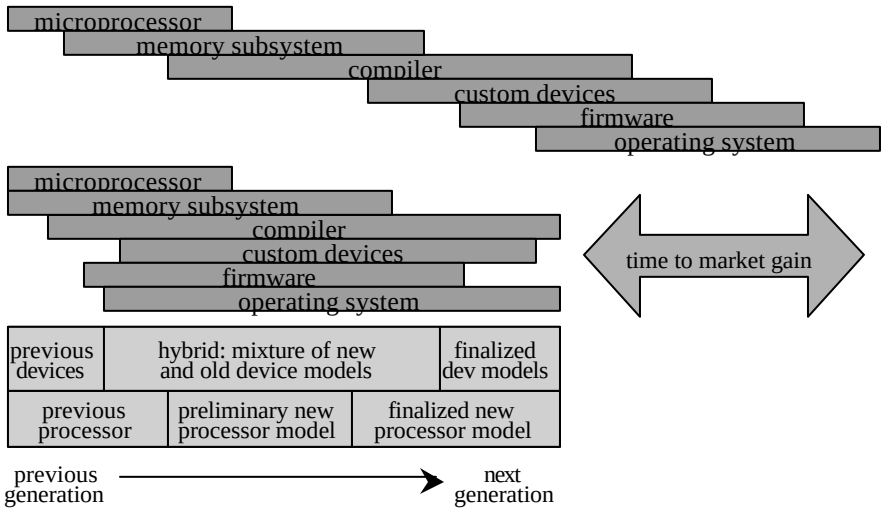
App: HW/SW Co-development

- Leverage maleability of simulator
- Allow software development to begin early
- Provide a common framework for a variety of simulation tools and techniques

 Simics Overview, June 2001 virtutech

slide 35

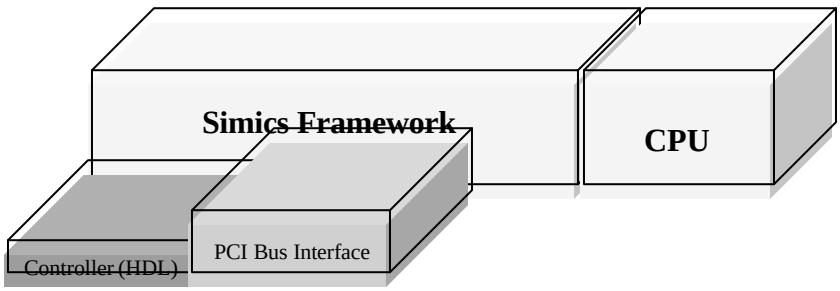
Parallelizing system design



 Simics Overview, June 2001 virtutech

slide 36

Basic Philosophy, part II

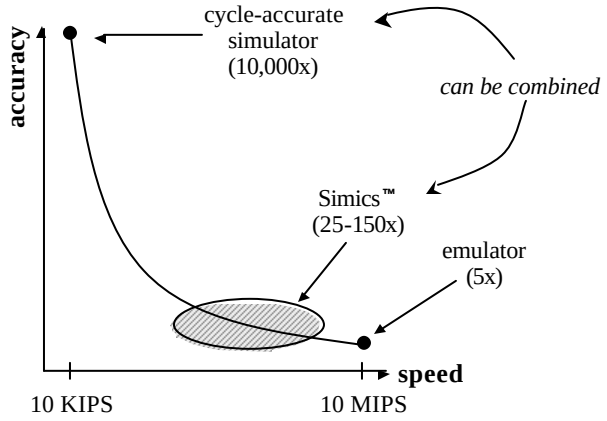


- Modular structure
- Baseline full-system instruction-level model
- Vary abstraction level on a per-module basis
- Vary composition of module types on per-test basis

 Simics Overview, June 2001 virtutech

slide 37

Simulation speed vs. accuracy



 Simics Overview, June 2001 virtutech

slide 38

The rest of this talk ...

- Simics Time
- Distributed Simics
- ”Advanced Usage”

 Simics Overview, June 2001 virtutech

Simics Time

virtutech

Introduction

- Simics default model has a simple view of time:
  - One instruction consumes one clock cycle
  - Some devices have delays
  - Each processor simulates a fixed number of cycles in a round robin fashion

virtutech

slide 41

Methods to add timing fidelity

- Feed Simics execution into a cache/cpu model
- Allow feedback by stalling instructions
- Extend Simics
  - Simultaneous instructions
  - Partial instructions
  - Speculation
- Combine techniques above
- Accuracy has a performance penalty

  virtutech

slide 42

Instr. Set Simulation State

- Model all software observable state
- Ignore redundant state (cached)
- Model state changes exactly
- Instructions are atomic
- Device state changes as big as possible

  virtutech

slide 43

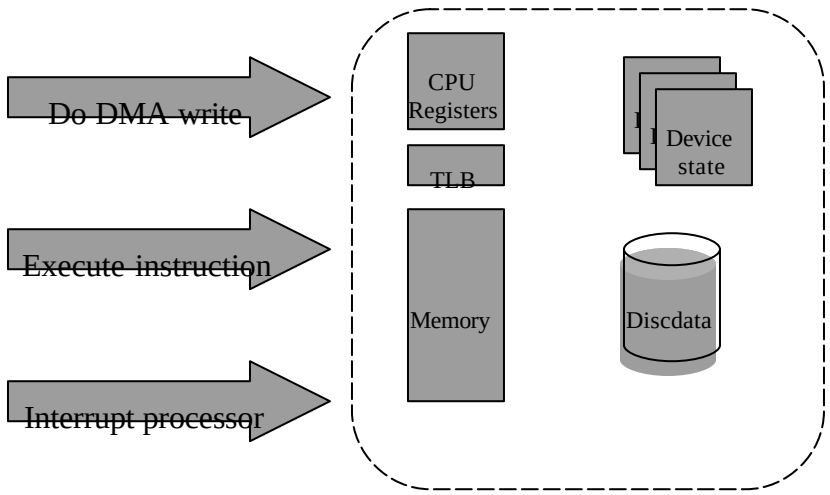
Time Models

- Time is approximated
- Order of events is approximated
- All interesting software tolerates this
- *Keep approximation in a separate part*

 Simics Overview, June 2001 virtutech

slide 44

ISS example



 Simics Overview, June 2001 virtutech

1G: Simple Cache Models

virtutech

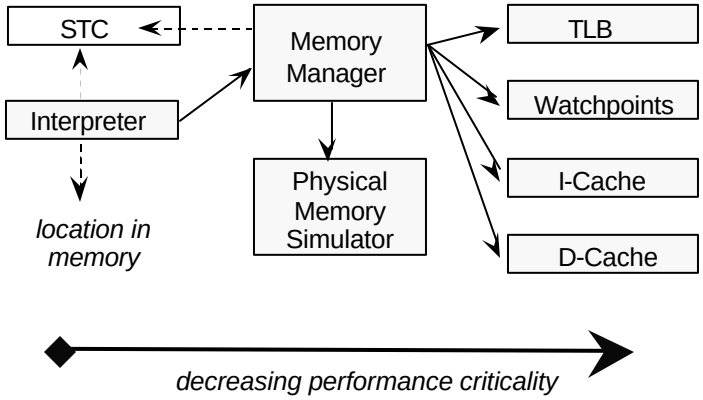
Simple Model

- Easy to add data and instruction caches
- Count cache accesses/hits/misses
- Provides approximation of event interleaving for multipro systems
- Takes advantage of STC to boost performance

virtutech

slide 47

Simulating Memory



 Simics Overview, June 2001 virtutech

slide 48

Memory Hierarchy

- Callback when a memory access is performed
- Based on address, accesses can be filtered for simulation efficiency
- Info about the memory access passed in a data structure
- May return ‘stall operation’ (used in 3G)

 Simics Overview, June 2001 virtutech

slide 49

Memory Hierarchy

```
struct generic_transaction {
  pa_t                physical_address;
  la_t                logical_address;
  char               *real_address;    /* pointer to data to be stored/where data will be loaded to */
  char               *host_address;    /* pointer to start of accessed memory */
  read_or_write_t     read_or_write;
  processor_mode_t    mode;
  data_or_instr_t     data_or_instr;
  unsigned            size;            /* size in bytes */
  mem_op_type_t       type;
  io_val_t            value;           /* value being passed, used for I/O, corresponds to register. */
                                       /* also overloaded in some cases, so must be at least 32b. */
  struct processor   *cpu_ptr;         /* processor on which operation is performed */
  unsigned            snoop_bit:1;     /* should it be snooped? */
  unsigned            cache_bit:1;     /* should it be cached? */
  unsigned            writethrough_bit:1; /* write through? */
  unsigned            block_STC:1;     /* set to 1 iff anybody (MMU, memory hierarchy, etc) wants to */
                                       /* see future accesses of this type */
  unsigned            may_stall:1;     /* when false the memory system must return a zero stall time */
  unsigned            ignore:1;        /* set to 1 to signal no-op (don't do access) */
  unsigned            inverse_endian:1;/* rev. endian of data in regs when moving data between mem and reg. */
  unsigned            atomic:1;        /* trans is part of an atomic sequence of mem ops */
  unsigned            atomic_last:1;   /* last trans in an atomic sequence (may have size 0). */
  unsigned            reserved1:1;     /* reserved field, typically defined to an arch dep type */
  unsigned            inv_mem_endian:1;/* the memory pointed out by host_address has inverted endianness */
                                       /* 'size' is the number of endian inverted bytes. */
  unsigned            inquiry:1;       /* set to 1 to indicate inquiry access */
  unsigned            use_page_cache:1;
  uint16              STC_choice;      /* internal: choice of STC table for this transaction, this */
                                       /* is numbered from 0 and up to STC_CHOICE_MAX */
  conf_object_t      *bus_master;      /* Bus master (PCI only) */
};
```

 Simics Overview, June 2001 virtutech

slide 50

il_quantize_fs_bdither()

	icache (a)	dcache write (b)	dcache read (c)	branch to (d)	branch from (e)	prof (f)	count (g)	
244	179	0	0	282	0	564	2	dir = 1;
245								/* => entry before first column */
246	272	0	0	0	0	1974	7	r_errorptr = cquantize->fserrors[0] + x_offset;
247	0	0	0	0	0	1974	7	g_errorptr = cquantize->fserrors[1] + x_offset;
248	70	0	0	0	0	1974	7	b_errorptr = cquantize->fserrors[2] + x_offset;
249								}
250								
251								/* Preset error values: no error propagated to first pixel ...
252	54	0	0	281	0	1689	3	r_cur = g_cur = b_cur = 0;
253								
254								/* and no error propagated to row below yet */
255	0	13	0	0	0	1689	3	r_belowerr = g_belowerr = b_belowerr = 0;
256	0	0	0	0	0	1689	3	r_bpreverr = g_bpreverr = b_bpreverr = 0;
257								
258	325	0	0	270261	270803	1085464	8	for (col = width; col > 0; col--) {
...								
...								
267	1	0	3900	270283	0	2432160	9	r_cur = RIGHT_SHIFT(r_cur + r_errorptr[dir] + 8, 4);
268	43	0	7987	60	0	2432160	9	g_cur = RIGHT_SHIFT(g_cur + g_errorptr[dir] + 8, 4);
269	1	0	2841	55	0	2432160	9	b_cur = RIGHT_SHIFT(b_cur + b_errorptr[dir] + 8, 4);

 Simics Overview, June 2001 virtutech

2G: Hybrid Simulation

virtutech

Hybrid simulation

- A combination of an instruction set simulator and a clock-cycle simulator
- Only timing of *subtraces* in clock-cycle simulator
- Some timing state in instruction simulator

virtutech

slide 53

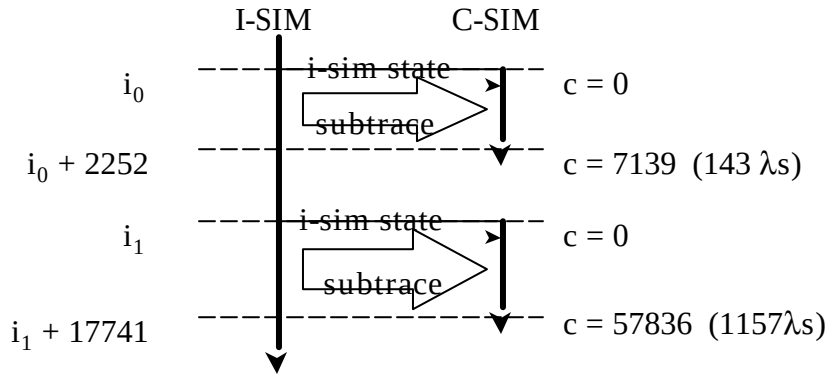
Transient timing state

- Affects timing within a few instructions
- Example:
  - FPU busy - transient,
  - Address *A* in cache - non transient
- In general:
  - Transient state delays instructions
  - Non-transient state speeds up instructions

 Simics Overview, June 2001 virtutech

slide 54

A hybrid simulation example



 Simics Overview, June 2001 virtutech

slide 55

Case study

- Commercially developed real-time system
- Motorola 88110 based hardware
- Proprietary operating system
- Multiple application processes
- Three critical functions were identified

 Simics Overview, June 2001 virtutech

slide 56

Model of MC 88110

- Dual-issue super scalar processor
- Register scoreboard
- Multiple functional units
- Load and store buffers
- 4 Kbytes, 4-way set associative split instruction and data caches

 Simics Overview, June 2001 virtutech

slide 57

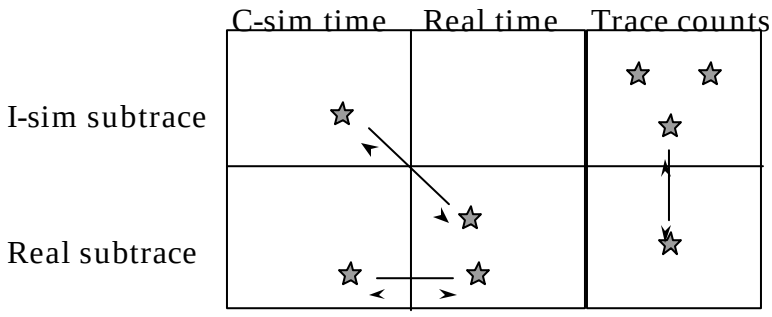
Clock-cycle simulator

- Models time producing a CPI breakdown
- Event-driven simulator modeling system resources
- Validated against logic analyzer trace

 Simics Overview, June 2001 virtutech

slide 58

Measurement domains



 Simics Overview, June 2001 virtutech

slide 59

CPI averages

	Program A		Program B		Program C	
	CPI		CPI		CPI	
Ideal	0.50	16%	0.50	15%	0.50	16%
Pipe	0.40	13%	0.44	13%	0.4	13%
Fetch	1.47	46%	1.13	35%	1.08	34%
Load	0.13	4%	0.22	7%	0.2	6%
Store	0.15	5%	0.09	3%	0.1	3%
Explained	2.65	84%	2.38	73%	2.28	72%
Measured	3.17	100%	3.26	100%	3.15	100%

virtutech

3G: Retire-Engine Model

virtutech

slide 61

3G features

- Memory operations stall
- Instructions may take zero or more cycles
- Multiprocessor interleaving more correct

 Simics Overview, June 2001 virtutech

slide 62

Event Queues

- Two queues: steps, cycles
- Step: each time the program counter changes
  - A completed instruction
  - An instruction exception, trap
  - An external interrupt
- Cycle: A time unit set at simulation start.
  - Typically the processor clock cycle.
- Absolute time can be used in cycle queue, rounded down to nearest cycle.
- Events can be posted to appear in a future cycle or step.
- Cycle is the highest resolution of time simics handles
- Events within a cycle are handled in fifo.

 Simics Overview, June 2001 virtutech

slide 63

Mapping time to steps

Time	1	2	3	4	5	6	7	8
Cycle queue	1	2	3	4	5	6	7	8
Step queue	1		2	3	4			5

- Default timing is one cycle per instruction.
- Memory accesses can stall instruction execution.
- Scheduling is a quantum of cycles per processor in a round-robin fashion.
- Executing an event takes zero time.

 Simics Overview, June 2001 virtutech

slide 64

Advanced Timing

Time (ns)	1	2	3	4	5	6	7	8
Cycles CPU1	1		2		3		4	
Step CPU1	i1				i2,i3		i5	
Cycles CPU2	1			2			3	
Steps CPU2	i1,i2			i3			i4	

- Cycle queue event may sync time on all CPUs.
- Synced event minimum delay is one quantum.
- Instruction may take zero cycles.
- CPUs may have different clock frequencies.

 Simics Overview, June 2001 virtutech

slide 65

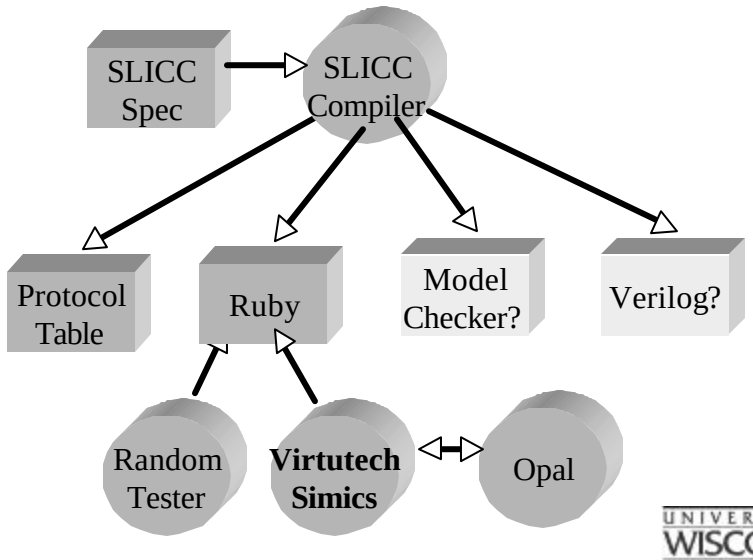
Ruby & Opal

- Ruby Memory Hierarchy Simulator
  - Simulates cache hierarchy & protocol specified in SLICC
  - Loadable Simics module
  - Informs Simics of variable memory reference latency
- Opal Out-of-Order Processor Simulator
  - Simulates R10K-like SPARC processor
  - Uses Simics as retirement engine
    - Opal need *not* get every case correct
    - Must advance Simics processor by #instructions retired
  - Uses Ruby for memory hierarchy simulation

UNIVERSITY OF WISCONSIN MADISON

slide 66

Wisconsin Multifacet's Simulation Infrastructure



UNIVERSITY OF WISCONSIN MADISON

4G: Generic Middleware

virtutech

slide 68

Bridge gap to more detail

- Multiple instructions
- Non atomic execution
- Speculative execution
- Cycle accuracy

 Simics Overview, June 2001 virtutech

slide 69

Desired Enhancements

- Interface to other simulators
  - Both timing and functional modles
  - Memory, CPU and device models
- Multiple memory transactions
- Speculation
- Memory coherency (MP) and consistency
- Multiple instructions per cycle

  virtutech

slide 70

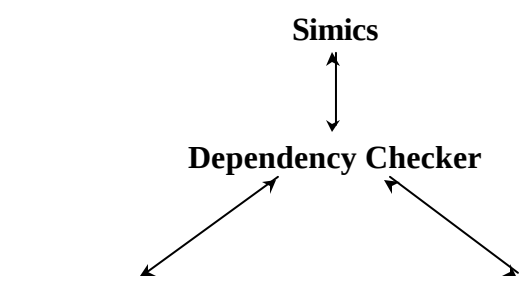
Simics Requirements

- Execute partial instructions
  - I-TLB, fetch, execute, D-TLB, mem access, commit
- Multiple outstanding instructions
- Check dependencies between instructions & parts
- Keep per-instruction view/change of machine state
  - Speculation, out-of-order execution
- Clock cycle-driven simulation
  - Multithreading

  virtutech

slide 71

Simics Architecture Overview



CPU functionality, devices, basic framework

Checks timing model
Dictated by architecture
May drive simulation

CPU Timing Model

Microarchitecture details
Begin, ITLB, fetch, exec,
DTLB, d-access, commit

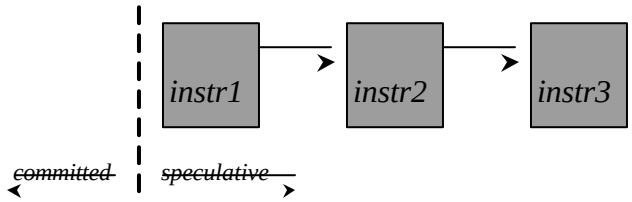
Memory Hierarchy

Cache, MP Coherency
mem_hier_operate
data_ready

 Simics Overview, June 2001 virtutech

slide 72

Implementation: ”in progress” buffer

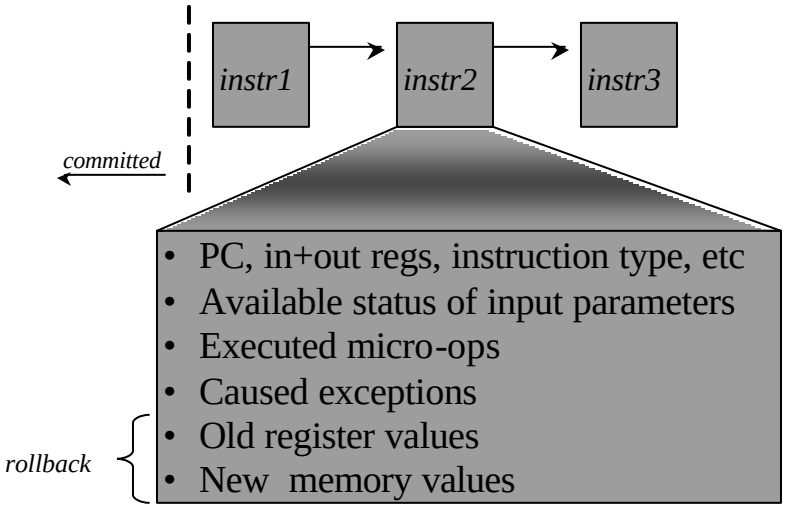


- One state packet per instruction
  - A sequence of expected state changes
- Buffer of state packets for *in progress* ops
  - I.e. Fetched but not committed
- Metadata in state packet supports:
  - Dependency checking
  - Split of instructions into micro-ops
  - Exceptions delayed until commit
  - Rollback

 Simics Overview, June 2001 virtutech

slide 73

State Packet Contents



 Simics Overview, June 2001 virtutech

slide 74

Dependency Checker

- Checks for dependencies between instr
  - Determine when instr may exec/commit
- Operates on state packets
- Gets timing info from external model
- Handles issues not of interest to timing mod
  - e.g. the TM may only want memory ops

 Simics Overview, June 2001 virtutech

slide 75

Simics Interface

- Same for both TM and DC (filter)
- Data in state packets
- Core functions:
  - sp = begin()
  - execute(sp, part)
  - commit(sp)
  - rollback(sp)

  virtutech

slide 76

Example: A simple CPU

- Setup: begin, end, mem access
- Loop:
  - sp = begin()
  - end(sp)
  - sp = begin()
  - end(sp)
  - advance_one_clockcycle()

  virtutech

slide 77

Example: A speculating CPU

- Setup: begin, end, rollback, mem access
- Loop:
  - sp = begin()
  - Set_reg(”PC”, 0x4000)
  - sp2 = begin()
  - end(sp)
  - If (speculation(sp2))
    - rollback(sp2) else end(sp2)
  - advance_one_clockcycle()

 Simics Overview, June 2001 virtutech

slide 78

Example

```
ld r1 <- [0x10]
or r1,5 -> r2
and r4,7 -> r3
nop
```

- Clock 1:
  - sp1 = begin(”ld”)
  - execute(sp1)
- Clock 2:
  - sp2 = begin(”or”)
  - [can’t exec, missing r1]
  - sp3 = begin(”and”)
  - execute(sp3) ; no deps
- Clock 3:
  - sp4 = begin(”nop”)
  - execute(sp4)
- Clock 4:
  - [mem sys return value for load]
  - end(sp1) ; load
  - execute(sp2) ; or
  - end(sp2)

 Simics Overview, June 2001 virtutech

slide 79

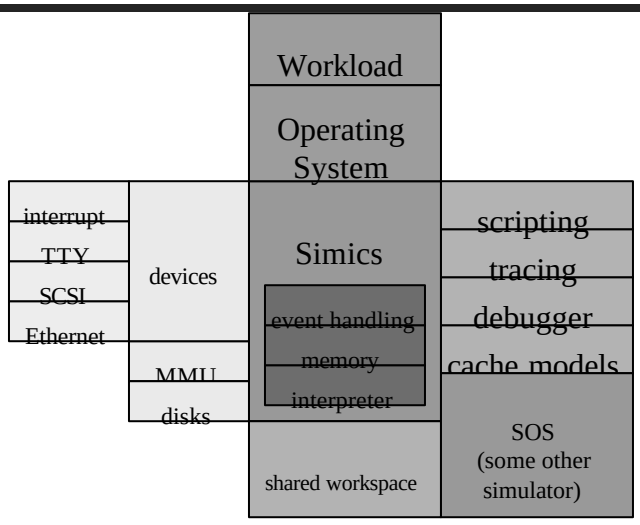
Next step: other simulators

- Direct integration with other models poses several challenges
  - Perspective on time (multiple masters)
  - Communication overhead (task switching)
  - Interface design (multiple perspectives on the same module)
- Work in progress
  - Basic model is a multithreaded co-execution platform

 Simics Overview, June 2001 virtutech

slide 80

Multithreaded co-exec platform



 Simics Overview, June 2001 virtutech

Distributed Simulation

virtutech

Outline

- The basics
- Ethernet support
- Usage
- Implementation
- Proxy Objects

virtutech

slide 83

Simics-Central tasks

- Handle all Simics-Simics communication
- Run multiple Simics simulations synchronized
  - Keep simulation deterministic!
- Provide framework for simulated networks

 Simics Overview, June 2001 virtutech

slide 84

Uses

- Virtual networks (Ethernet)
- Real network connection
- Access objects in other Simics
- Large configuration split over several Simics
- Remote control of Simics
  - E.g. Collect information from many simulations

 Simics Overview, June 2001 virtutech

slide 85

What is Simics-Central?

- Simics-Central is a stripped-down Simics
  - Same front-end, loadable modules
  - Same or similar API
  - No CPU, memory, devices
- Loadable modules does most of the work
  - E.g. Ethernet module

  virtutech

slide 86

Communication Provided

- Timestamped communication
  - Between Simics modules and Central modules
- Also non-timestamped messages
  - Called "meta" messages
- Minimum latency
  - Determines how far a Simics may run without synchronization

  virtutech

slide 87

Overview

- First central module: "ethernet-central"
- Support several simulated networks (LANs)
- Connect to the real network
- Standard network services

 Simics Overview, June 2001 virtutech

slide 88

Network Setup

- All target machines on simulated networks
- Use own IP ranges and Ethernet addresses
  - Same target configuration everywhere
  - DHCP can be used to boot targets
- Drawback: Host system configuration needed
  - Must have a route to the simulated network

 Simics Overview, June 2001 virtutech

slide 89

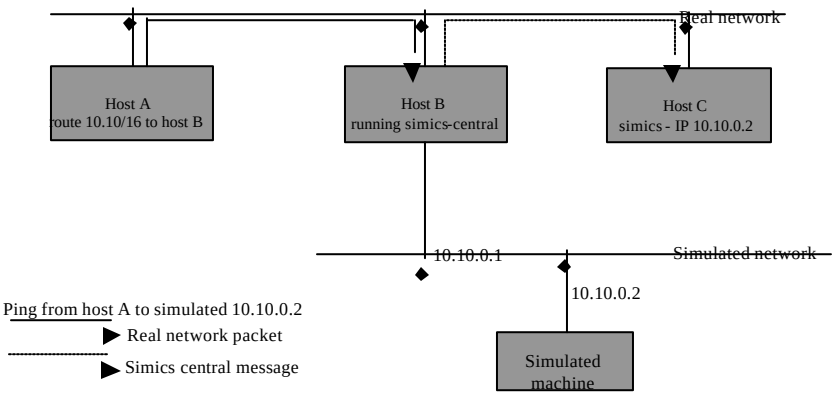
Network Support

- The ethernet-central acts as a router
  - Between simulated networks
  - Between simulated networks and real network
- Has own IP and ethernet address
  - Used to handle network services

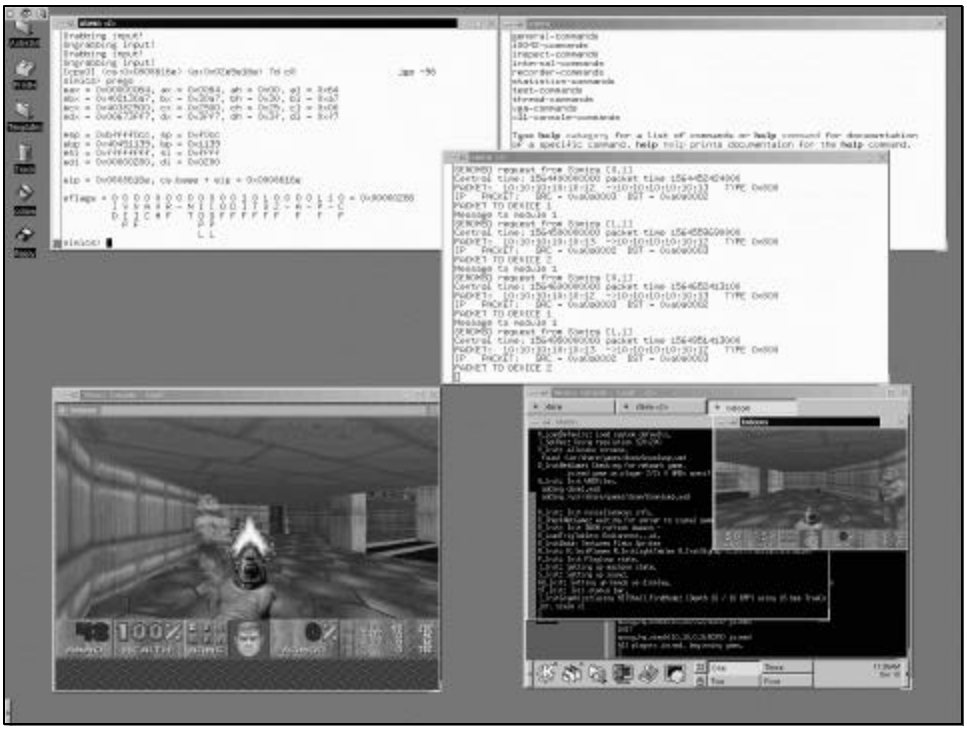
 Simics Overview, June 2001 virtutech

slide 90

Network Example



 Simics Overview, June 2001 virtutech



slide 92

Standard TCP/IP Services

- RARP
- BOOTP/DHCP
- DNS
- Bootparam
- Tftp
- Ping

virtutech

slide 93

Misc.

- To validate connection to a real network, use ping or a DNS lookup to the server itself.
- The central module will use the first IP on a simulated network, typically 10.10.0.1
- The 'run-loose' command:
  - breaks determinism
  - May be included in DIST13

  virtutech

slide 94

Central Setup

- Simics connects to central at startup
  - But may connect at anytime
- All "central aware" modules lookup the corresponding central module
  - Central modules are identified by a name
  - Modules register and specify their minimum latencies
- Central determines when a Simics may run

  virtutech

slide 95

Central Configuration

```
OBJECT ether0 TYPE ethernet-central {

        networks: ((net0, "10.10.0.0", "255.255.255.0"))

        dns: (("10.10.0.1", central, "network.sim"),
              ("10.10.0.5", donut, "network.sim"),
              ("10.10.0.6", bagle, "network.sim"),
              ("10.10.0.7", helicoid, "network.sim"),
              ("10.10.0.8", kawasaki, "network.sim"))

        arp: (("10.10.0.1", "10:10:10:10:10:10"),
              ("10.10.0.5", "10:10:10:10:10:12"),
              ("10.10.0.6", "10:10:10:10:10:14"),
              ("10.10.0.7", "10:10:10:10:10:16"),
              ("10.10.0.8", "10:10:10:10:10:18"))
}
```

 Simics Overview, June 2001 virtutech

slide 96

Example Run

```
[am@leeloo central]$ ./simics -x central.simics
Reading configuration from 'central.conf'...
New network (net0), address 10.10.0.0 netmask 255.255.255.0 (bcast 10.10.0.255)
central-ip 10.10.0.1
Device 10:10:10:10:10:10 registering
Setting minimum number of processes to 1

[Simics connects...]

Connecting: /tmp/simics-03268
Connection established with local Simics
Installing new connection 2

[Network device connects...]

Got GETMOD message: ethernet-central
REGISTER request from Simics [0,1] (dev 1) lat: 50000000
Device 10:10:10:10:10:24 registering
```

```
simics> connect-central leeloo
Resolving Simics Central host name `leeloo'
Connecting to Simics Central: (local)
Connected to Simics-central
simics> hme0.connect net0
simics>
```

 Simics Overview, June 2001 virtutech

slide 97

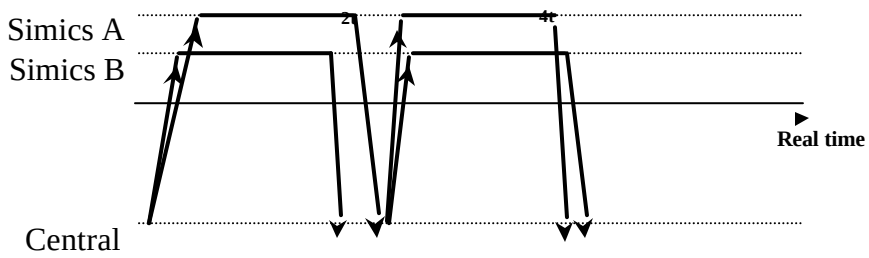
Protocol

- File or inet sockets are used for communication
- A two-phase clocking scheme is used
  - Hides some of the network latency
- Very simple message format
  - To, from, timestamp, meta, request-id, reply-id

 Simics Overview, June 2001 virtutech

slide 98

Simple clocking - NOT used



Green = messages + run 2t

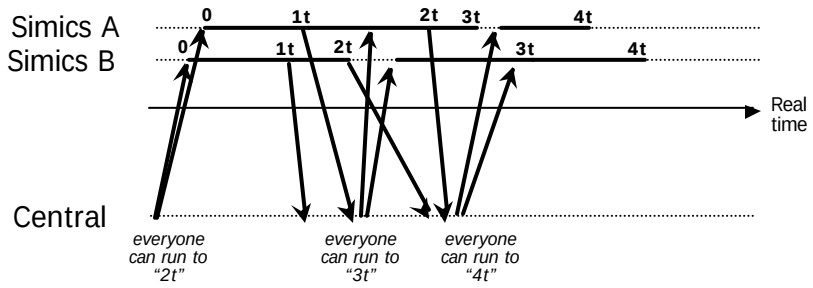
Red = ready

Minimum latency = 2t !

 Simics Overview, June 2001 virtutech

slide 99

Two-phase clocking



Green = messages + run 2t

Red = ready

Minimum latency = 2t

virtutech

slide 100

Keeping Determinism

- When packet received put it in event queue at correct time
- Packet received at same time sorted by sender
  - Not yet!
- Receive handling in own thread
  - Needed to handle meta (asychronous) messages

virtutech

slide 101

Addressing

- At startup Simics gets a unique *simics-id*
- Modules in each Simics get unique *sub-ids*
- *To address specific Simics, a global-id* is used
- *Set in the configuration*
  - *Translated by central to simics-id*
  - *Usually not needed by network modules*

  virtutech

slide 102

Simics API

- The API is still under development
- Similar interface in Simics-Central

```
int SIM_register_central_device(cb_func_ncsII_t get_message,
                                conf_object_t *obj,
                                nano_secs_t latency);

int SIM_get_central_module(const char *mod_name);

int SIM_central_send(int mod_id, int sub_id,
                     char *obuf, int olen);

int SIM_central_send_meta(int mod_id, int sub_id,
                          char *obuf, int olen);
```

  virtutech

slide 103

Simics API, cont

```
char *SIM_central_request(int mod_id, int sub_id,
                          char *obuf, int olen);

char *SIM_central_request_meta(int mod_id, int sub_id,
                               char *obuf, int olen);

int SIM_central_reply(int mod_id, int sub_id,
                      char *obuf, int olen,
                      int reply_id);
```


virtutech

slide 104

General

- Shadow objects from remote Simics
  - Implement the same *interfaces*
- Proxy server performs action on behalf of remote object
- Proxy object is transparent to other objects
- Can be used for all kinds of configuration objects

virtutech

slide 105

Typical Usage

- Split multiprocessor configurations
  - Some cpus in each Simics
  - Improve performance without changing Simics
  - MP host needed for performance
- Access device in other Simics
  - Useful for systems with different cpu architectures

 Simics Overview, June 2001 virtutech

slide 106

Example Configuration

Simics 1

```
OBJECT remote-pcibus25B TYPE proxy-class {
        remote_object_name: remote-pcibus25B
        proxy_server: proxy1
}
OBJECT glm0 TYPE proxy-class {
        remote_object_name: Xglm0
        proxy_server: proxy1
}
OBJECT proxy1 TYPE proxy-server {
        remote_simics_id: 2
        remote_server_name: proxy2
        min_latency: 100000
}
```

Simics 2

```
OBJECT remote-schizo25 TYPE proxy-class {
        remote_object_name: schizo25
        proxy_server: proxy2
}
OBJECT remote-cpu0 TYPE proxy-class {
        remote_object_name: cpu0
        proxy_server: proxy2
}
OBJECT proxy2 TYPE proxy-server {
        remote_simics_id: 1
        remote_server_name: proxy1
        min_latency: 100000
}
```

 Simics Overview, June 2001 virtutech

slide 107

Proxy Communication

- Simple: Block request until reply is back
  - Slows down the simulation
  - Lower latency might help, but not too low
- Still have to stall the time
  - To keep determinism
  - To avoid dead-lock
  - Solution: Only stall local time relative central

  virtutech

slide 108

Proxy Communication, cont

- Utilize *stall* for memory accesses
  - Most important call when distributing memory
- Other CPUs may run when one is stalling
  - Hides (some of the) communication latency
- Only works with stallable interface calls
- Calls without return value does not have to stall

  virtutech

slide 109

Current Proxy Status

- Still under development
- Successfully booted machine with a PCI bus in another Simics
- Expected to be included in DIST13B
  - Fall of 2001

 Simics Overview, June 2001 virtutech

Simics Advanced Usage

virtutech

slide 111

Topics

- Scripting
- Simics' API
- C vs. Python
- Communicating with the target
- Haps, events, and breakpoints
- The configuration system
- Source code debugging
- OS emulation

  virtutech

slide 112

Scripting

- CLI – command line interface

```
./simics –x <file>
simics> include "myfile.simics"
```

- Python at the command line

```
simics> @conf.phys_mem0.memory[conf.cpu0.esp] = 0x12
```

- Python source files

```
simics> source "myfile.py"
```

  virtutech

slide 113

Adding Frontend Commands

- Commands are defined in Python
- Example:

```
from cli import *

def mycommand(arg0, arg1):
  print ”mycommand arg0 %s and arg1 %s” % (arg0, arg1)

new_command(”mycommand”, mycommand,
            [ arg(int, ”arg0”),
              arg(str, ”arg1”, ”?”, ”defaultstring”) ])
```

  virtutech

slide 114

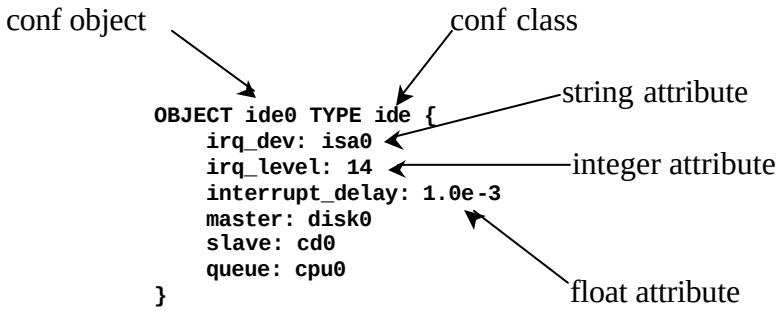
Simics’ API

- Functions listed in the reference guide
  - C: `#include simics_api.h`
  - Python: `from sim_core import *`
- Simics datatypes
  - Configuration classes – `conf_class_t`
  - Configuration objects – `conf_object_t`
  - Configuration attributes – `attr_value_t`
  - Interfaces

  virtutech

slide 115

Configuration



 Simics Overview, June 2001 virtutech

slide 116

Configuration (kirk-1p.conf)

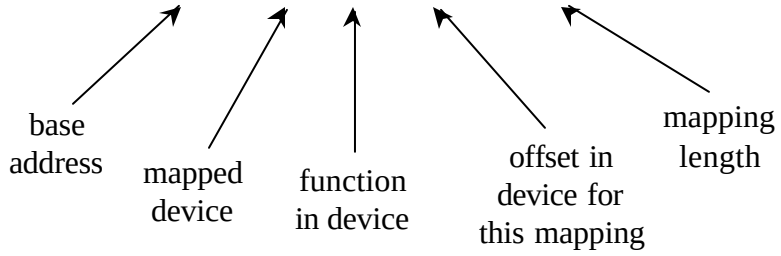
```
OBJECT cpu0 TYPE x86 {
          queue: cpu0
          freq_mhz: 20
          physical_memory: phys_mem0
          port_space: port_mem0
          apic: apic0
}
OBJECT phys_mem0 TYPE memory-space {
          map: ((0xa0000, vga0, 1, 0, 0x20000),
               (0x00000, mem0, 0, 0x00000, 0xA0000),
               (0xc0000, mem0, 0, 0xc0000, 0x8000),
               (0xc8000, setmem0, 0, 0, 0x28000),
               (0xf0000, mem0, 0, 0xf0000, 0x10000),
               (0x100000, mem0, 0, 0x100000, 0x1f00000),
               (0xfec00000, ioapic0, 0, 0, 0x20),
               (0xfee00000, apic0, 0, 0, 0x4000))
}
OBJECT apic0 TYPE apic {
          apic_id: 0
          apic_bus: apic_bus0
          cpu: cpu0
          queue: cpu0
}
...
```

 Simics Overview, June 2001 virtutech

slide 117

Configuration (maps)

```
OBJECT phys_mem0 TYPE memory-space {
    map: ((0xa0000, vga0, 1, 0, 0x20000),
          ...
          ...
          (0xf0000, mem0, 0, 0xf0000, 0x10000),
```



 Simics Overview, June 2001 virtutech

slide 118

Interfaces

- Inter-configurationobject communication
- structs containing functionpointers

```
typedef struct interrupt_interface {
    void (*raise_interrupt)(conf_object_t *ic, int level);
    void (*lower_interrupt)(conf_object_t *ic, int level);
} interrupt_interface_t;
#define INTERRUPT_INTERFACE_T "interrupt"
```

- Install in class with SIM_register_interface
- Get from object with SIM_get_interface
  - `interrupt_interface_t *intr;`
  - `intr = SIM_get_interface(obj, INTERRUPT_INTERFACE);`

 Simics Overview, June 2001 virtutech

slide 119

C/C++ vs. Python

- API function(pointer)s are the same
- Performance issues
- Data is converted automatically

C/C++	Python
`conf_object_t *`	`confobj` instance (`conf` module)
`conf_class_t *`	`string` (name of class)
`attr_value_t, kind = Val_List`	`list`
`attr_value_t, kind = Val_Data`	`tuple`

- No enums

  virtutech

slide 120

Moving Data in/out of Simics

- Magic Instructions
- Edit diskdumps through loopback mounts
- file-cdrom
  - Mount host CD (-image) in target drive
- /hostfs
  - Pseudo device and target OS driver
  - Allows the target to mount host disks
- Simics Central

  virtutech

slide 121

Haps, events, and breakpoints

- Events are scheduled in a time queue
- Haps are occurrences, e.g. state changes in the simulator
- Breakpoints are set by `SIM_breakpoint` and trigger the `Core_Breakpoint` hap

  virtutech

slide 122

More on Haps

- Hap functions:
  - `SIM_hap_register_callback` – catch hap by name
  - `SIM_hap_get_number` – faster access
  - `SIM_hap_install_callback` – catch hap
  - `SIM_hap_new_type` – create new
- Some haps have arguments
  - `SIM_hap_install_callback_idx`, `SIM_hap_install_callback_range`

  virtutech

slide 123

Hap Example

```
from sim_core import *

def printk_handler(dummy0, type, bp_id, dummy1, dummy2):
    cpu = SIM_current_processor()
    ea = cpu.gprs[31]
    while 1:
        pa = SIM_logical_to_physical(cpu, 0, ea)
        c = SIM_read_phys_memory(conf.cpu0, ea, 1)
        if (c == 0):
            return
        pr("%c" % c)
        ea = ea + 1

id = SIM_breakpoint(conf.phys_mem0, Break_Physical,
                    4, 0x0001715c, 1, 0)
SIM_hap_register_callback_idx("Core_Breakpoint",
                              printk_handler, id, 0)
```

  virtutech

slide 124

Magic Breakpoints

- Interface between taget software and the simulator
- Magic instructions trigger the `Core_Magic_Breakpoint` hap
  - alpha: `sextb r0,r<n>,r<m>`
  - sparc: `sethi <arg>, %g0`
  - x86: `xchg bx,bx`

  virtutech

slide 125

Magic Breakpoint Example

```
void *global_data;

int modify_data_area()
{
   MAGIC(0);  /* disable locks */
   /* modify data area here */
   MAGIC(1);  /* enable locks */
}

int main()
{
   for (;;) {
        modify_data_area();
        run_program(global_data);
       }
}
```

 Simics Overview, June 2001 virtutech

slide 126

Magic Breakpoint Example Ctd.

```
bp = -1
def magic_handler(obj, enable):
  global bp
  if enable:
    ea = SIM_logical_to_physical(conf.cpu0, 0, 0x12349abc)
    datav = SIM_read_phys_memory(conf.cpu0, ea, 4)
    datap = SIM_logical_to_physical(conf.cpu0, 0, datav)
    bp = SIM_breakpoint(conf.phys_mem0, Break_Physical,
                        2 /* write */, datap, 4096 /* size */, 0)
  else:
    if bp >= 0:
      SIM_delete_breakpoint(bp)
      bp = -1

SIM_hap_register_callback("Core_Magic_Instruction“,
                          magic_handler, 0)
```

 Simics Overview, June 2001 virtutech

slide 127

Script Branching

- Forked Python threads
- Created by `start_branch(<func>)`
- Sleep by calling `wait_for_hap`, `wait_for_hap_idx`, or `wait_for_hap_range`

 Simics Overview, June 2001 virtutech

slide 128

(Silly) Script Branch Example

```
@def test_branch():
    print "This is a test - going to sleep"
    ret = wait_for_hap_idx("Core_Step_Count", 10)
    print "Woke up with return: %d" % ret
    print "Leaving branch"

@start_branch(test_branch)
```

 Simics Overview, June 2001 virtutech

slide 129

Configurations

- Configuration objects appear in the `conf` module
  - `simics> list-attributes cpu0`
  - `simics> @conf.cpu0.ebx = conf.cpu0.eip`
- Memory limit
  - `<image>.memory-limit <max MBs>`

 Simics Overview, June 2001 virtutech

slide 130

Configuration via Python

- Allows parameterization of configuration files

```
def create_machine(memory_size):
  SIM_set_configuration([
    [”cpu0”, ”x86”,
      [ ”queue”, ”cpu0” ],
      [ ”freq_mhz”, 20 ],
      [ ”physical_memory”, ”phys_mem0” ] ],
    [”phys_mem0”, ”memory-space”,
      [ ”map”,  [ [ 0, ”vga0”, 0, 0, memory_size ] ]
  ])
```

 Simics Overview, June 2001 virtutech

slide 131

Source Code Debugging

- Remote gdb
  - Uses gdb's remote debugging protocol over TCP/IP

```
simics> load-module gdb-remote
(gdb) target remote localhost:9123
```

  - Works cross-platform
- Symbol support in Simics
  - Work in progress; planned for dist13

  virtutech

slide 132

Running without an OS

- Set up "dummy" machine with the configuration system
- Catch syscalls and exceptions

  virtutech

slide 133

User Decoders

- Add or change instructions
- Callback on each attempt to decode an instruction

 Simics Overview, June 2001 virtutech

thank you

www.simics.com
info@virtutech.se
www.virtutech.{com,se}

virtutech