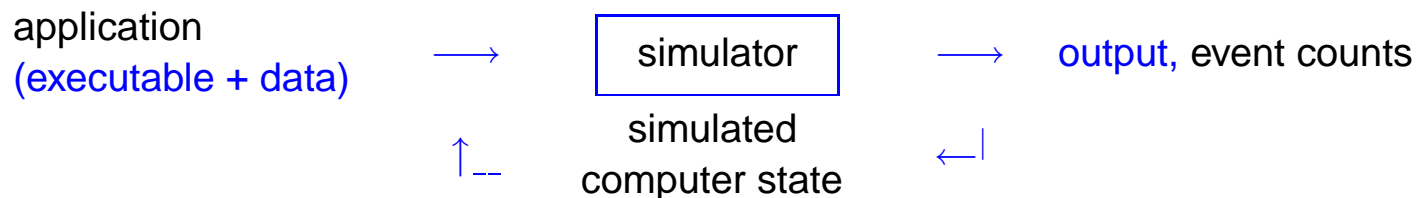


Background: why use computer simulation?

- or, why not just run and time the application on the computer itself?
- advantages of simulating applications:
 - can have full visibility: actual H/W may not count all events of interest
 - the simulated computer may not even exist! (essential for design!)
 - ◆ can use any suitable host computer to run the simulator
 - can vary parameters for architectural studies (e.g. cache size)
- user-level computer simulation: run executable program; when a trap instruction is simulated, use OS on the host computer to perform the corresponding action
 - PeANUt is a user-level simulator!
 - for trap #2, the execute operands stage involves `AC = getchar();`
 - ◆ `getchar()` calls the `read()` system call on the host computer
- complete computer simulation: boot the simulated computer from its OS's kernel image; can then run applications from the (simulated) shell

Simulation: basic concepts

- simulator runs as a normal process on the host computer:



- interpret each instruction; (c.f. Lects P1-10)

update (functional & timing-related aspects of) state, count desired events

- e.g. PeANUt! (iwaki:/usr/local/src/DCS/peanut/execute, 4625 lines of C code)

■ code path for load 99 (001 001 0001100011)

```
/* fetch instruction: */
MAR = GetBits(PC-1,0,10);
MDR = memory[MAR]; CI = MDR;
...
/* decode instruction: */
mode = GetBits(CI,13,3)>>13;
switch(mode) { ...
  case 1: /* 001 */
    opCode = GetBits(CI,10,3)>>10;
  ... }
... }
... }

/* evaluate operands: */
switch(mode) { ...
  case 1: /* 001 */
    MAR = opSpec = GetBits(CI,0,10);
    MDR = memory[MAR]; OP = MDR;
  ... }
... }

/* execute operation: */
switch(opCode) {
  case LOAD_OC:
    AC = OP;
  ... }
... }
```

The Sparc-Sulima Simulator

- an (almost-) complete UltraSPARC simulator: Sparc-Sulima (50K lines C++ code) (multiple-CPU)
- reasonably (?) fast (eg. 200 – 500 × slowdown)
- ‘boot from `main()`’ mode: (runs modified, statically linked `exe`)
 - MM: Phys. Address = Virt. Address (+ offset); all pages must be in RAM
 - load ELF executable `exe` into simulated RAM, section by section (`load_elf.cc`)
 - set `%pc` to `_start`, `%sp` to ELF-specified value in `exe`; go!
 - ◆ can optionally install trap handlers, e.g. for TLB misses, register window overflow
 - simulator can also read symbol table from ELF `exe`
 - ◆ provides mapping between program symbols and (e.g. `main = 0x1800000`)
 - ◆ useful, e.g. trace function calls
 - ◆ demo: `factorial` program; demonstrates register window spilling ($n \geq 5$)
 - ◆ note: SPARC stores the return address in `%o7` (not on the stack!)
 - ★ does this make it immune from buffer overflow attack?

The Sparc-Sulima Simulator: Solaris Emulation mode

- SolEmn (chief architect: Bill Clarke)
- runs *un*-modified, statically or dynamically linked Solaris `exe`
 - boots from a small nucleus
 - emulates *most* system calls, e.g. I/O, threads, memory management (like a modern OS)
- MM: full simulation of demand paging
 - simulated RAM can be much smaller than program size!
 - memory-map system calls must also be emulated
- loads ELF `exe`, then loads the Solaris dynamic loader `ld` (also an ELF executable)
- `ld` is then run to perform dynamic linking (when finished, jumps to `exe`'s `_start`)
- demo: `testtrans`
 - note: simulator has a cycle-accurate timing model for the UltraSPARC III Cu