SimOS Machine Simulator

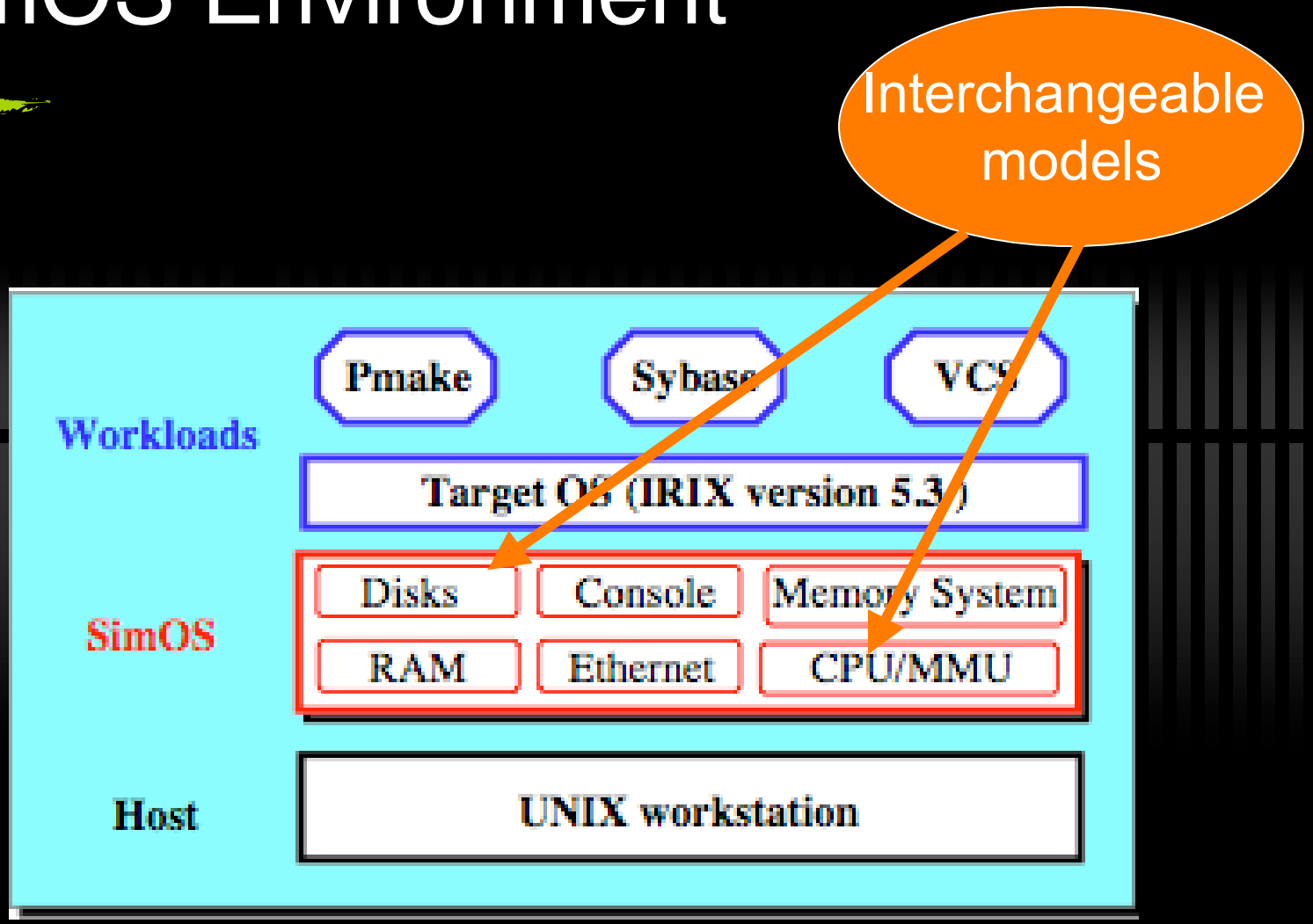
Reetuparna DAS

Adapted from SimOS tutorial www.cs.stanford.edu

SimOS

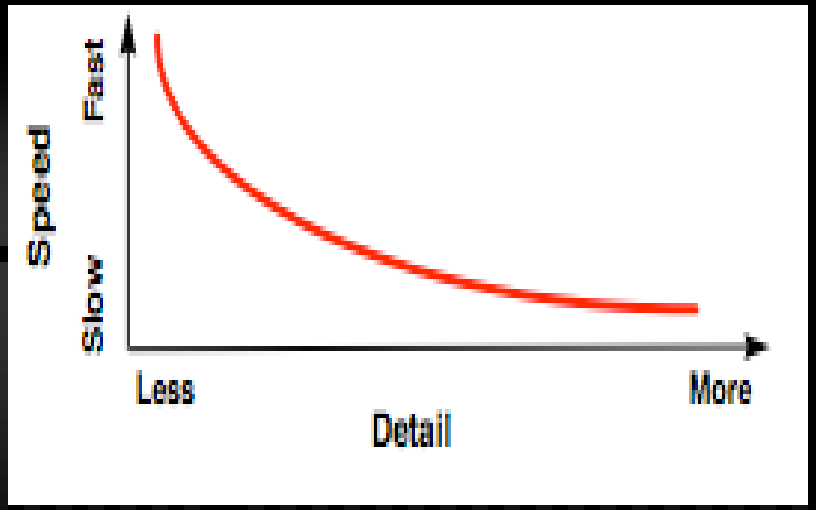
- Complete Hardware simulator
  - High Simulation speed
  - Simulation Data Characterization
- Can be used to study
  - New Architectural designs
    - E.g. FLASH Multiprocessor
  - New OS designs
    - E.g. HIVE Debugging
  - Complex work load Characterization
    - E.g. Relational Databases server tuning
  - Performance Evaluation

SimOS Environment



SimOS Speed-Detail Trade Off

- ✓ Embra
  (O(10x) slowdown)
- ✓ Mipsy
  (O(100x) slowdown)
- ✓ MXS
  (O(1000x) slowdown)



SimOS Modes

- **Emulation mode**
  - Run workload as fast as possible,
  - No concern for timing accuracy.
  - Simulation slowdown < 10x
- **Rough Characterization mode**
  - Keep speed of emulation but add timing model
  - Capture first-order effects
  - Instruction execution, memory stall, I/O, etc.
  - Simulation slowdown < 25x ·
- **Detailed Characterization mode**
  - Arbitrary accuracy and simulation slowdown

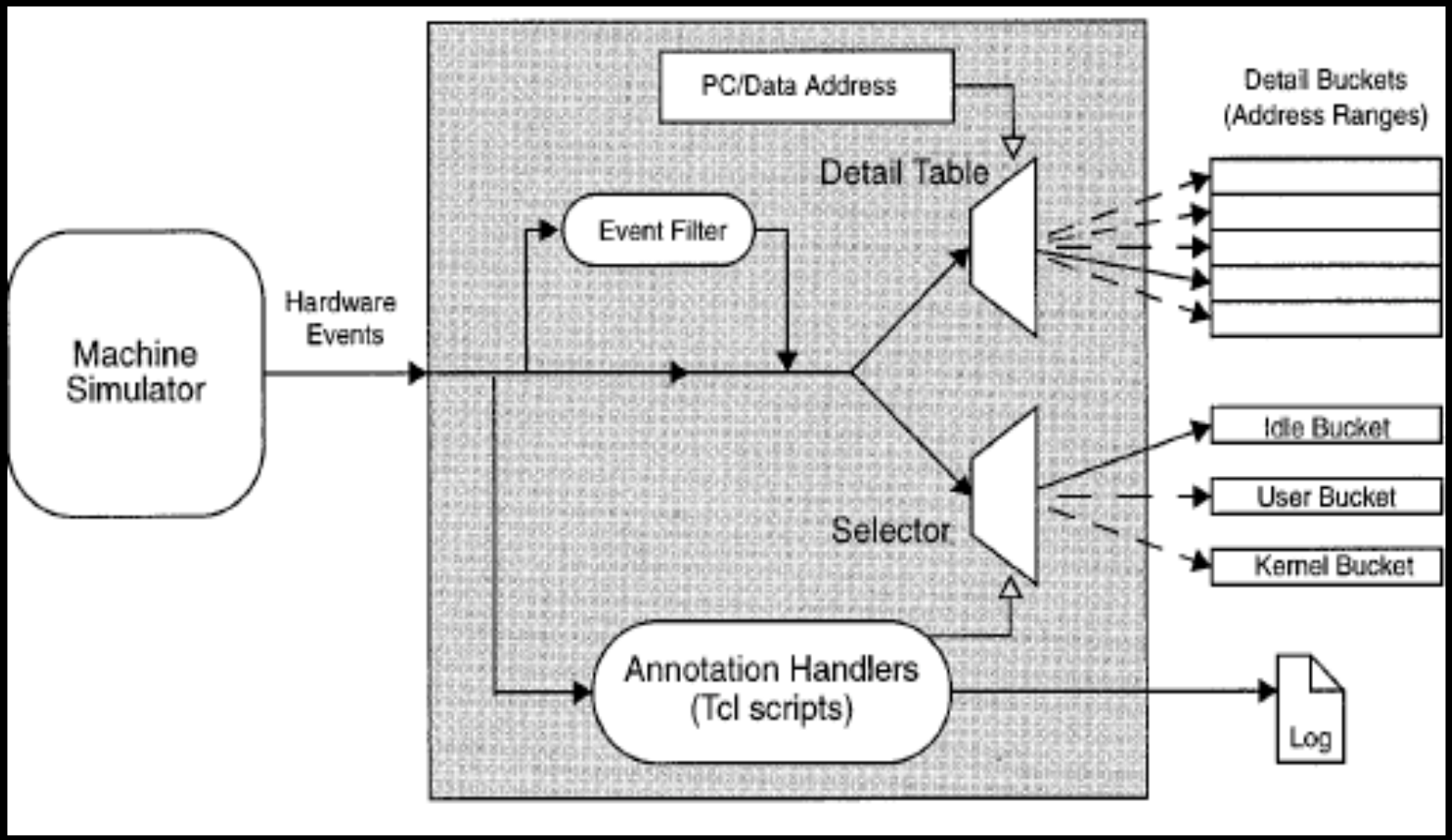
Data Collection

- Three major Challenges
  - Large amount of data
  - High rate at which data is generated
  - Mapping hardware events to more meaningful higher level system metrics

Data Collection

- SimOS Data collection Techniques
  - **Buckets: Places where events can be stored** Defined by the user of SimOS
  - **Annotations: Tcl scripts that run on events** Allows user to control the processing of events
  - **Selectors & Detail Tables: Control event recording into Buckets.** Supports efficient and flexible recording of events

Data Collection Architecture



Data Collection Mechanisms

- Implementation through Tool command language (Tcl) scripting language Interpreter
- We now look in more detail at
  - Event actions
  - Event classification
  - Event filter

Annotations

- Annotations are Tcl scripts that user can attach to specific hardware events or software events
- These Annotations may freely access internal state of the simulator, Kernel, and depend on the OS but SimOS is itself totally independent of any operating system
- They are non intrusive

Example Events

- ✓PC virtual address
- ✓Data reference virtual address
- ✓Traps or interrupts
- ✓Instruction opcodes
- ✓Cache misses
- ✓Cycle count

A Simple Annotation

```
# Define a new annotation type for process events
annotation type process enum {switchOut switchIn}

# Program Counter annotation at the end of the exec system call
annotation set pc kernel:exece:END {
        # On an exec, only the name of the process changes, not the pid
        set PROCESS($CPU) [symbol read kernel:u.u_comm]
}

# Program Counter annotation at the end of the context-switching code
annotation set pc kernel:resume:END {
        # Execute higher-level annotation
        annotation exec process switchOut
        # Update executable name and pid
        set PID($CPU) [symbol read kernel:u.u_procp->pid]
        set PROCESS($CPU) [symbol read kernel:u.u_ucomm]
        annotation exec process switchIn
}

# Program Counter annotation at the beginning of the idle loop
annotation set pc kernel:idle:START {
        annotation exec process switchOut
        set PID($CPU) -1
        set PROCESS($CPU) "Idle"
        annotation exec process switchIn
}
```

Event Classification

- Selectors
- Selector is a script which determines how frequent events are recorded rather than what action needs to be taken for the event
- Selectors control which of a set predefined bucket is used to record events
- Annotations need to be placed on events that change the selector to point to a new bucket

Event Classification

- Another Mechanism, Detail Tables
- Detail Tables are like selectors except that target bucket is based on the address of the event

A Selector Example

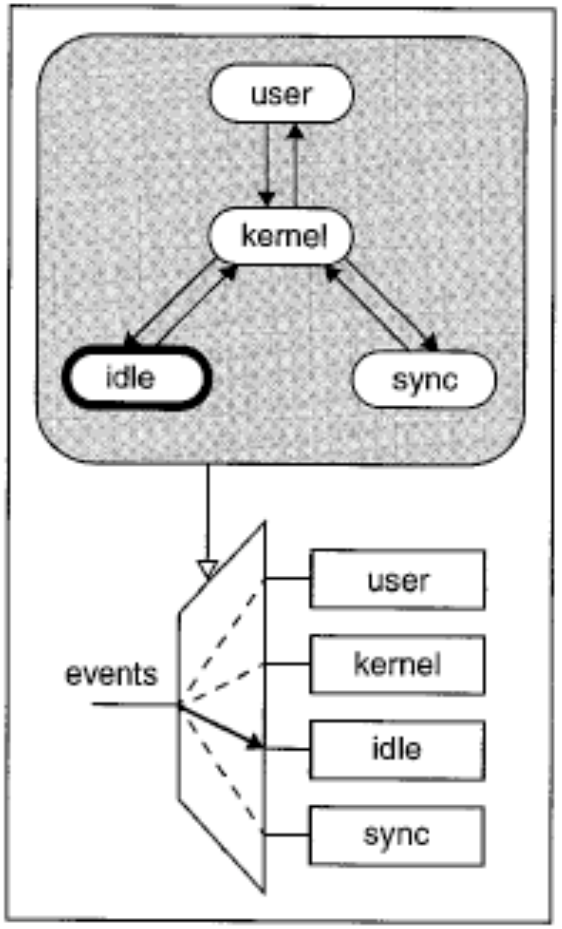
```
# Initialization: create and initialize selector
# Initial mode is kernel
selector create modes
for {set i 0} {$i < $numCPUs} {incr i} {
        set currentState($i) kernel
        selector set modes $i kernel
        stackInit "stateStack_$i"
}

# Save current state on stack when an exception occurs
annotation set exc {
       stackPush "stateStack_$CPU" $currentState($CPU)
       set currentState($CPU) kernel
       selector set modes $CPU kernel
}

# Restore saved state when processor executes "return from exception" opcode
annotation set inst rfe {
       set currentState($CPU) [stackPop "stateStack_$CPU"]
       selector set modes $CPU $currentState($CPU)
}

# transition to idle when PC reaches the kernel function named "idle"
annotation set pc kernel:idle:START {
       set currentState($CPU) idle
       selector set modes $CPU idle
}

# Return to kernel state when PC reaches end of the kernel function named "idle"
annotation set pc kernel:idle:END {
       set currentState($CPU) kernel
       selector set modes $CPU kernel
}
```

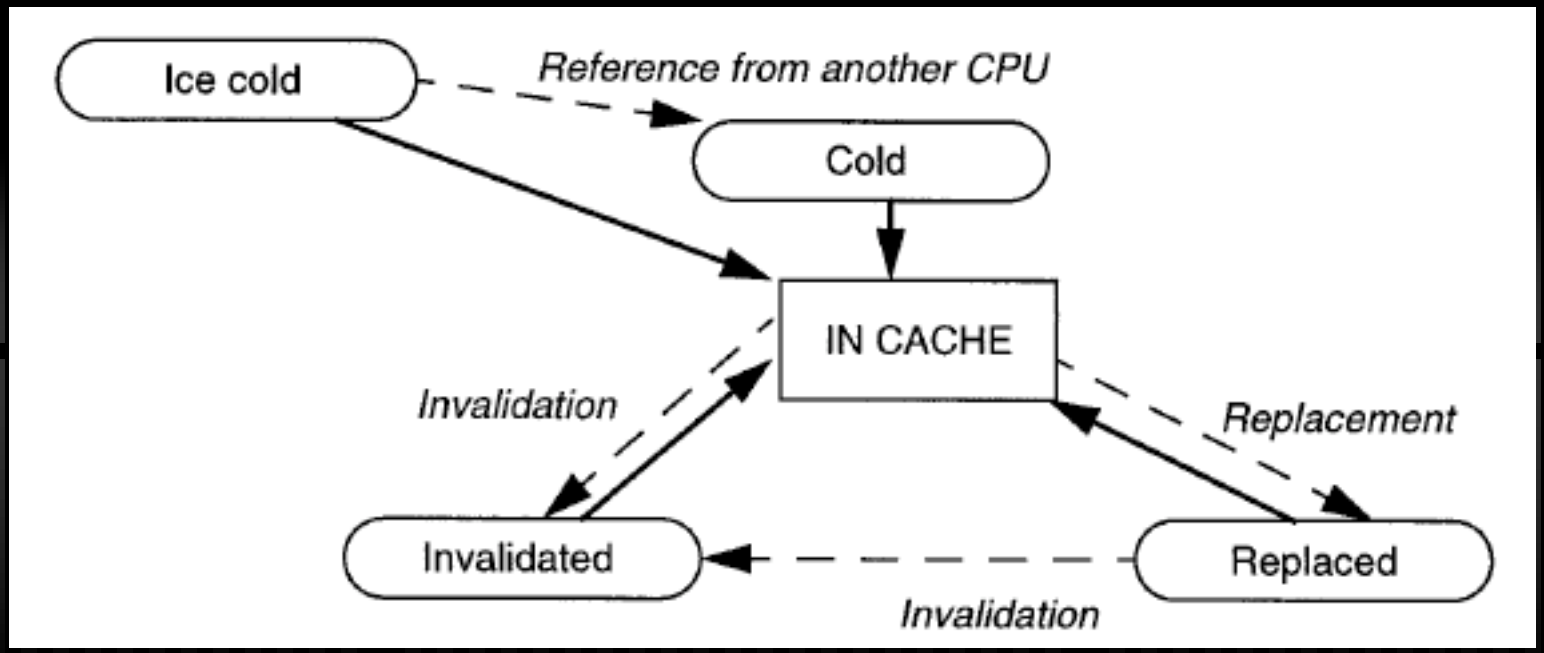


Process mode Tracking

Event filters

- Not necessary to know occurrence of an event, necessary to know cause
- Even filters are basically state machines associated with hardware events which give idea about cause of events
- Filter can be used in conjunction with detail tables

Event Filter Example



Cache miss Classification

Comparison with other Simulators

- Advantages over Instrumentation approach
  - Complete event coverage and Non intrusiveness
- Other approaches are more specific, lack complete system simulation
- Some approaches e.g. Shade, SimICS do not model complete hardware of the machine, thus cannot be used for studying OS or workloads
- Superior event classification and mapping mechanisms

SimOS Advantages

- **Realistic workloads**
  - SimOS can study almost any workload
  - Develop workloads on real machine
  - Copy workloads on to SimOS disks
- **Great visibility**
  - Observe **all** behavior: application, OS, hardware
- **Non-intrusive**
  - Observation does not perturb system
- **Consider alternatives**
  - Hardware/software instrumentation
  - Application-level simulation