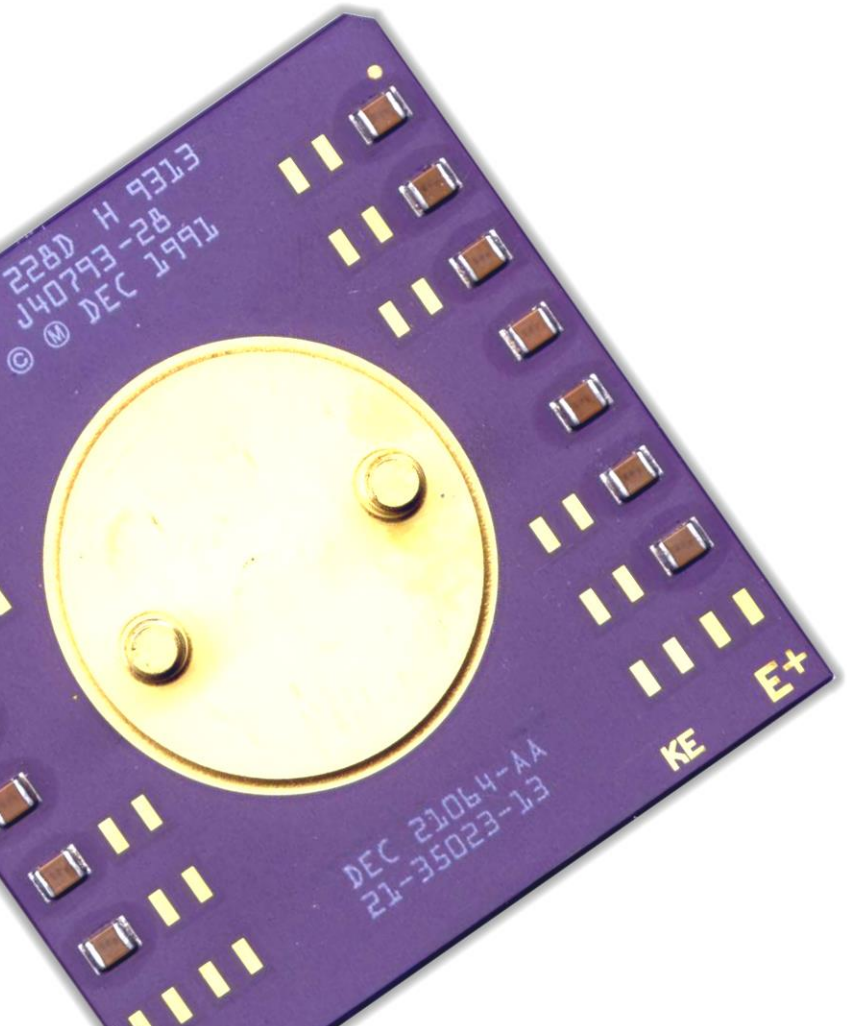
SimpleScalar MP

multiprocess simulator



developed by **Pau Erola**

under guidance of **Carles Aliagas**

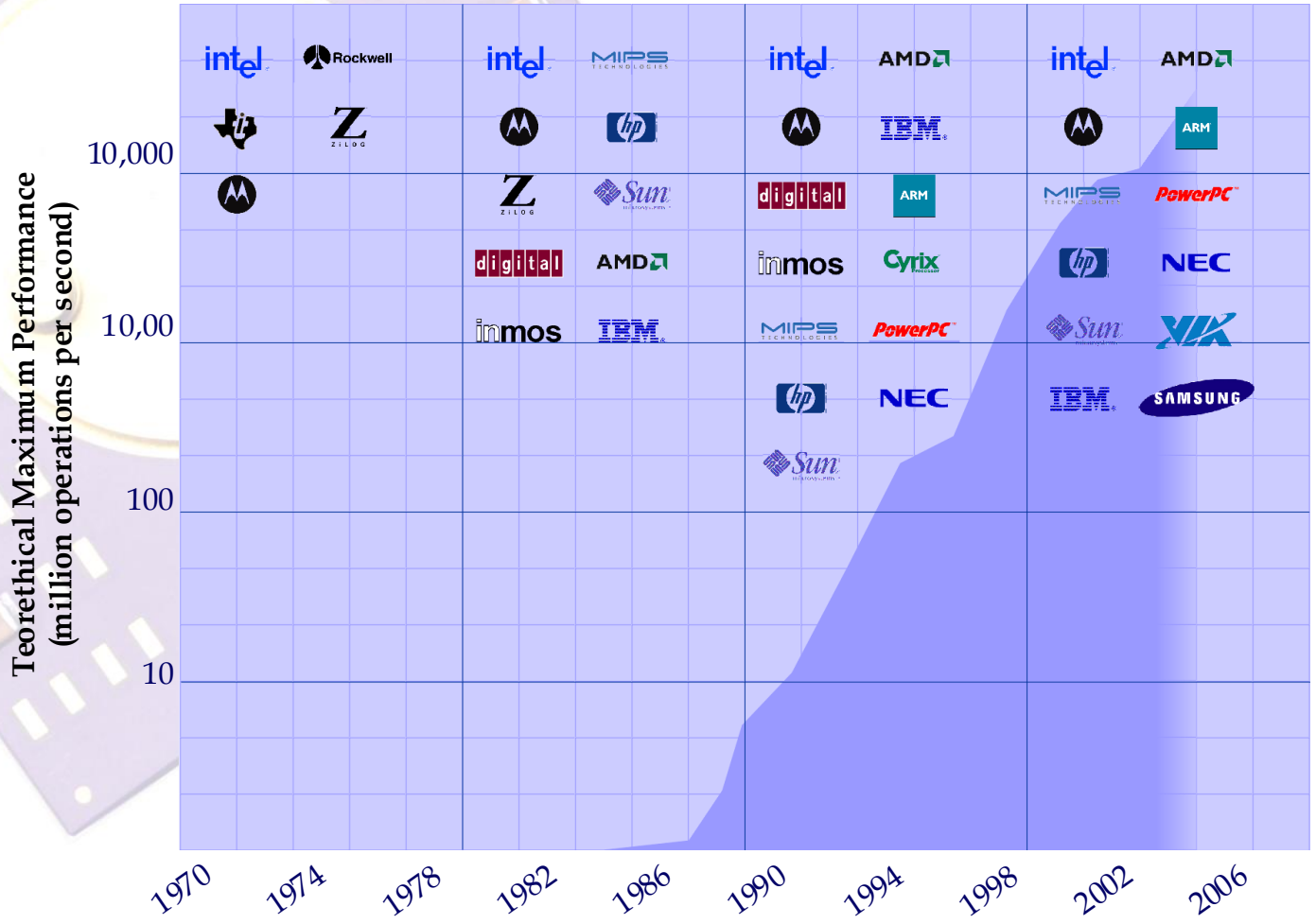
June 2006

Departament d'Enginyeria
[DEIM]
Informàtica i Matemàtiques
Universitat Rovira i Virgili

What's in this document

- Previous concepts and goals
- Specifications
- Design and development
- User's Guide and Simulation Examples
- Conclusions
- References

Computer Architecture Evolution and Revolution



Computer Architecture Simulators

- Highly competitive market
  - increasing demand of performance and reliability
  - need to have suitable tools to be the first
- An architectural simulator is a tool that reproduces the behavior of a computing device
  - faster and more flexible development cycle
  - permits more design space exploration
  - facilitates validation before H/W becomes available

SimpleScalar Tool Set

- Computer architecture research test bed
- Freely available with source and docs from UW-Madison
- First public release in July ’96
- Primary Advantages
  - extensible
  - portable
  - detailed
  - performance

Scheduling Processes

- Modern general-purpose computers have an operating system to run other programs
  - Multitasking operating systems share common processing resources for multiple tasks
    - cooperative multitasking
    - preemptive multitasking
  - The scheduler is the component of the kernel that selects which process to run next
    - multiprogramming systems
    - time-sharing systems
    - real-time systems

Project Aims

- Multitasking operating systems produce lost of performance
  - Need to store and restore the CPU context to memory on context switch
  - Increment of cache accesses and misses
  - Decrement of branch predictor hits
- Our primary aim is to hack Sim·OutOrder to simulate multiple processes running concurrently using a configurable scheduling strategy

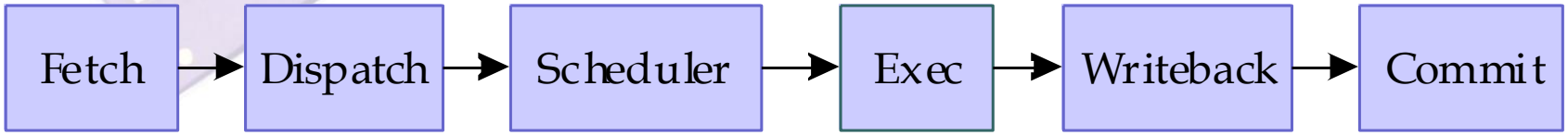
Sim-OutOrder

- Sim-OutOrder is a very detailed performance simulator
  - generates timing statistics for a detailed out-of-order issue processor core with two-level cache memory hierarchy and main memory
- Source code design philosophy
  - infrastructure facilitates “rolling your own”
  - performance and flexibility before clarity

Main code

```
ruu_init();
for(;;)
  {
     ruu_commit();
     ruu_writeback();
     lsq_refresh();
     ruu_issue();
     ruu_dispatch();
     ruu_fetch();
  }
```

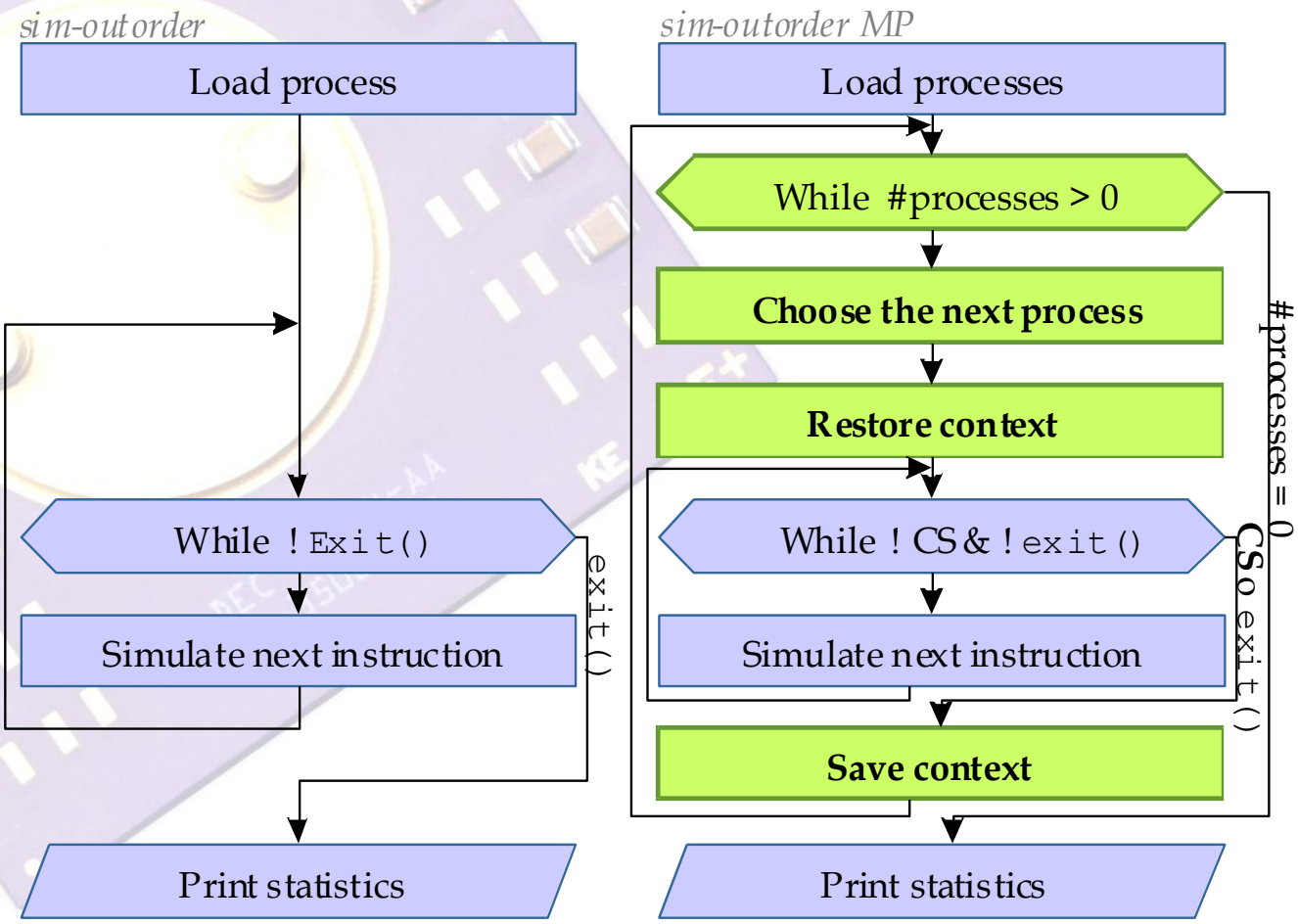
- in-order retirement
- wait for outputs
- execution
- wait for inputs and functional unit
- fetch and decode instr.



Hacking Strategy

- Load all processes
- Add a configurable scheduler
  - Quantum
  - Flush of structures
  - Penalizations
  - Blocking
- Add new individual statistics for each process
- Do simulations to validate our modifications

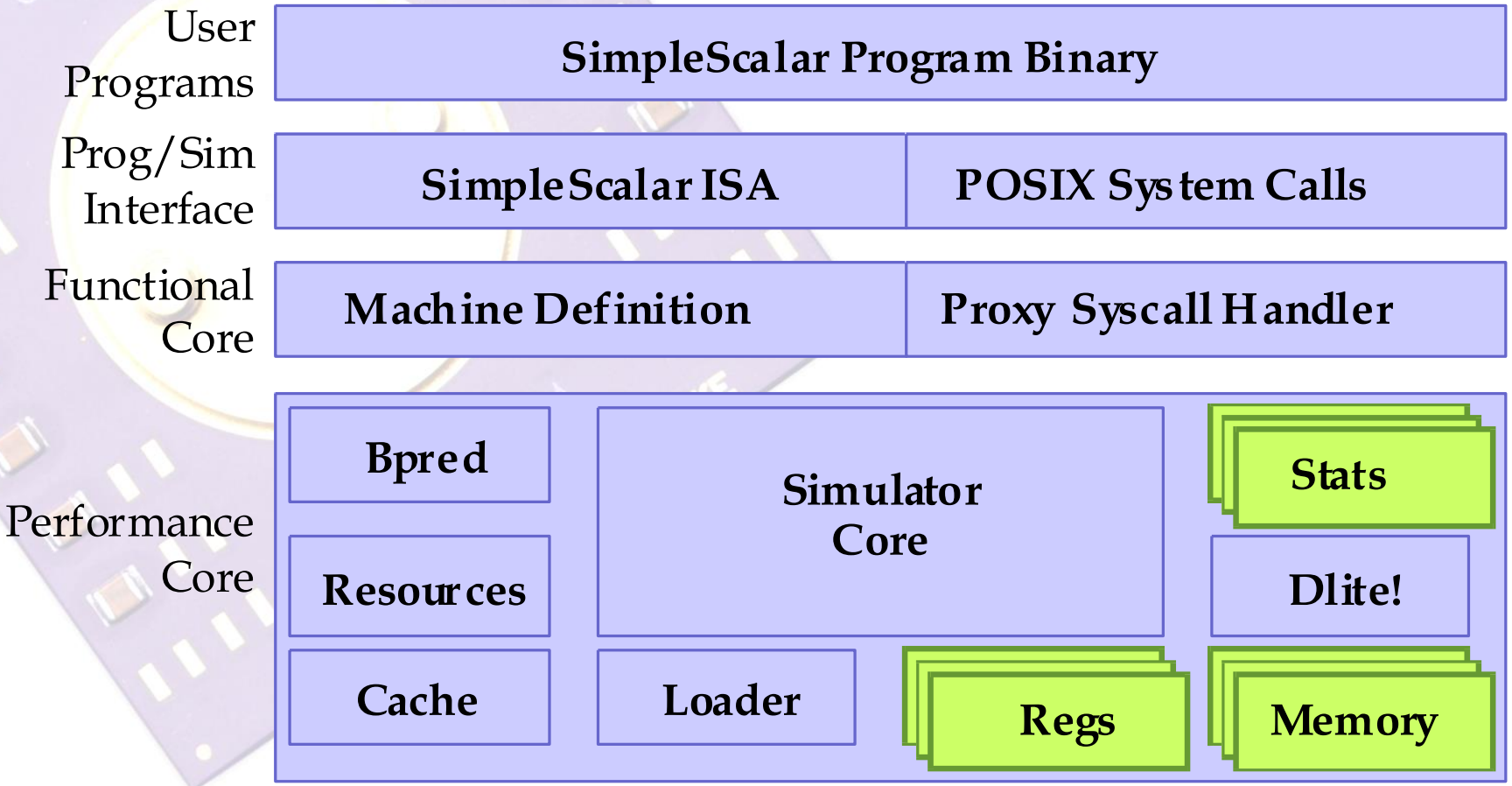
Execution Scheme



Process Concept

- A process is a sequential program in execution
  - unique ID
  - memory space
    - object program to be executed
    - data on which the program will execute
  - virtual CPU
    - registers & PC
- Implemented in **`schd.h`**

Simulator Structure



Structure Implementation

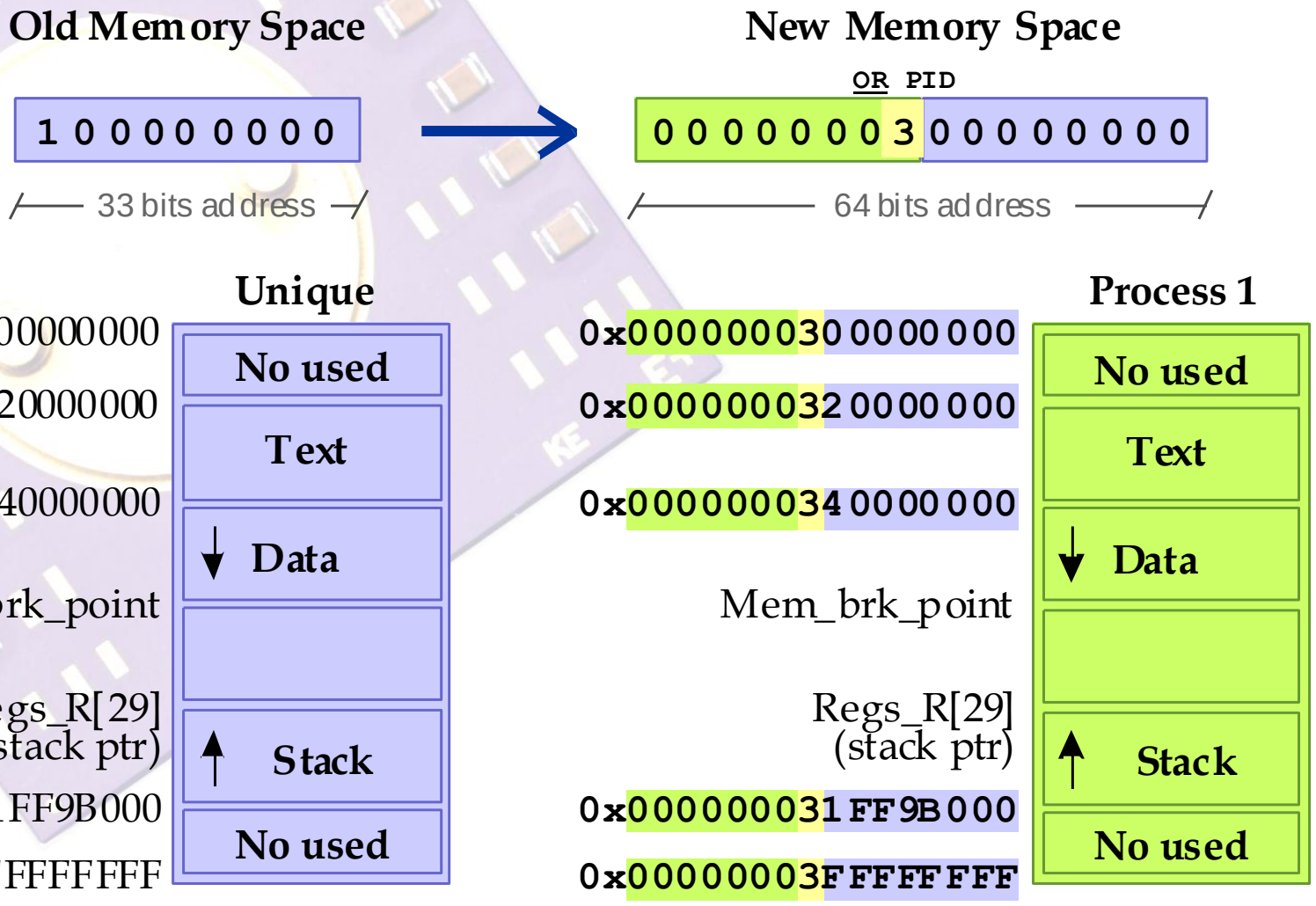
- Structure duplications

```
static struct regs_t mp_regs[LD_MAX_PROCS];
static struct mem_t *mp_mem[LD_MAX_PROCS];
mp_{nom_estadística} [LD_MAX_PROCS]
```

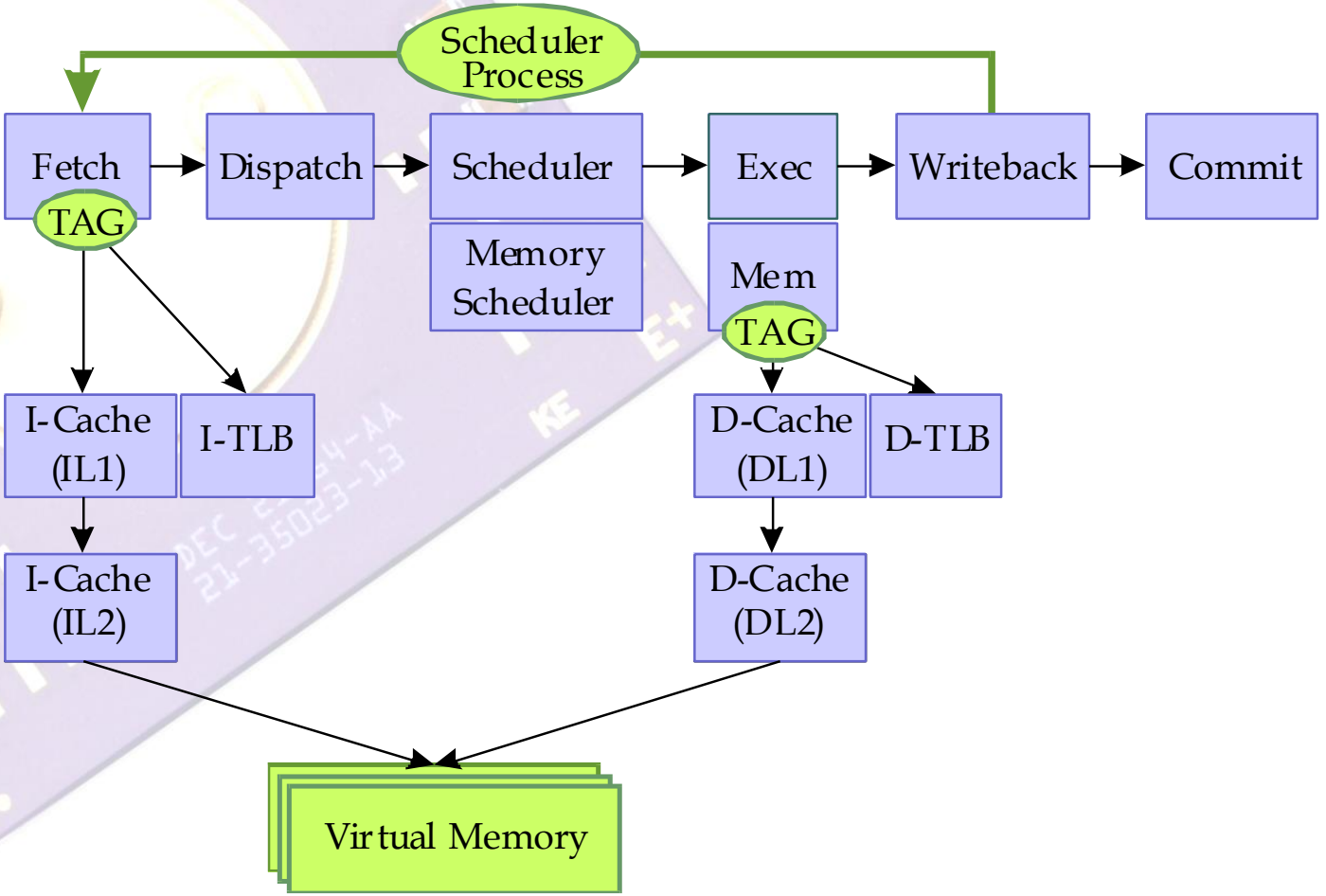
- Memory identification
  - SimpleScalar uses 2^{33}bits of address
  - With higher bits can identify the process' memory space

```
addr = addr | (ld_proc_id<<34);
```

Memory Structure



Multiprocess Out-of-Order Issue Simulator

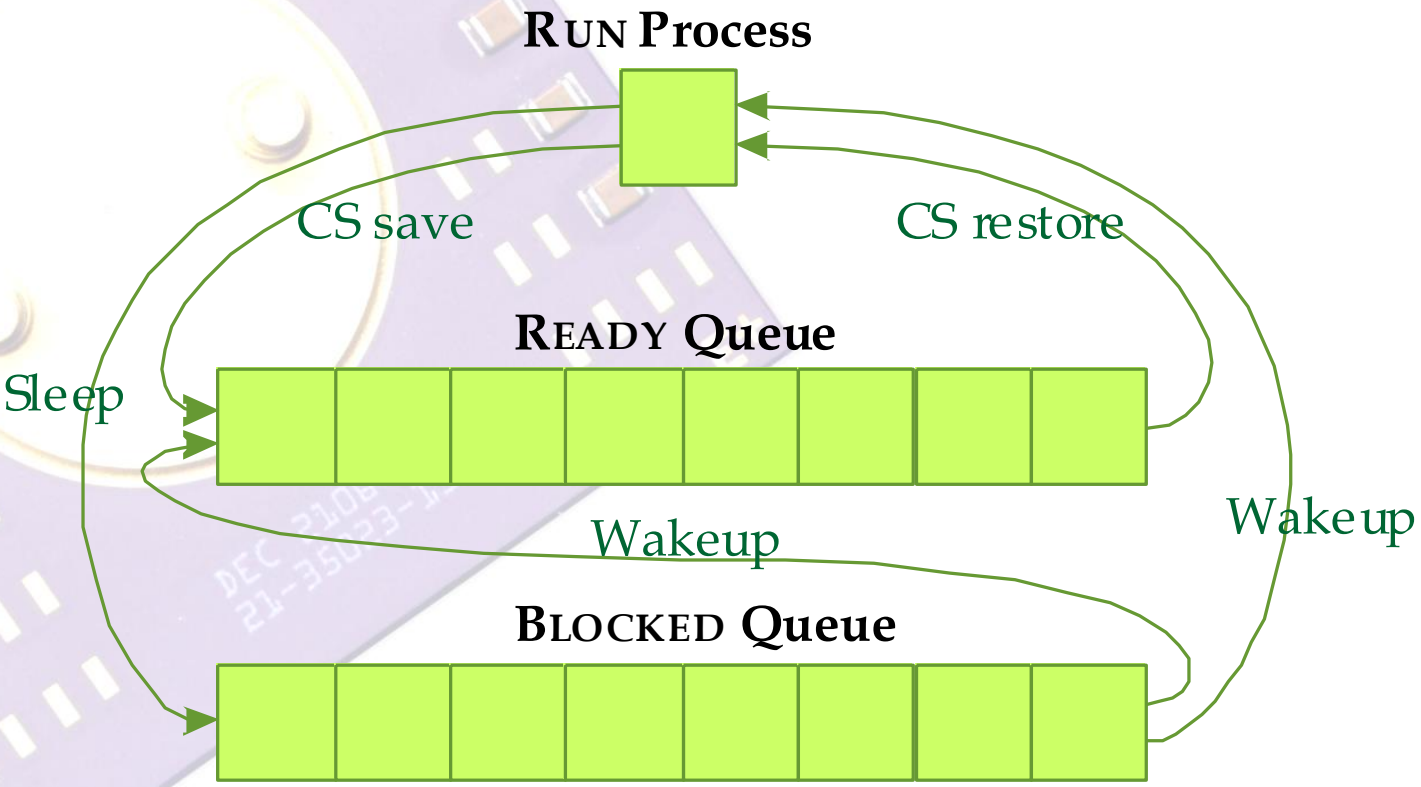


Scheduler & Context Switch

- Implemented in sim-oomp.c

```
sim_cycle++
wakeup_blocked_processes
if ( context_swich_required ) then
        context_save
        select_next_process
        context_restore
endif
```

Scheduler Queues



Scheduler Queues Implementation

- RUN points to process that is currently executing

```
unsigned int ld_proc_id;
```

- READY queue contains the processes that are waiting to run
  - queue ordered by priority (lower priority first)
- BLOCKED queue contains the processes that are waiting for a system call
  - queue ordered by wakeup cycle
- Macros for queues manipulation are implemented in **`schd.h`**

Statistics Implementation

- Preserve the global simulation statistics
- Add new individual statistics for each process

```
mp_{stat_name} [process_id]
```

User's Guide

- Installation

```
$ gunzip $HOME/SimpleScalar-3.0-MP.tar.gz
$ tar -vxf $HOME/SimpleScalar-3.0-MP.tar
$ cd $HOME/SimpleScalar-3.0-MP
$ make config-alpha
$ make sim-oomp
```

- Execution

```
$ sim-oomp <configs> list_of_executables
```

List of Executables

- Highly configurable simulator

  <quan> quantum in cycles

  <penal> cycles of penalization at CS

  <blk%> probability of blocking during system calls

  <appr> preemptive scheduler

  <pri> priority of the process

  <wakeup> start cycle

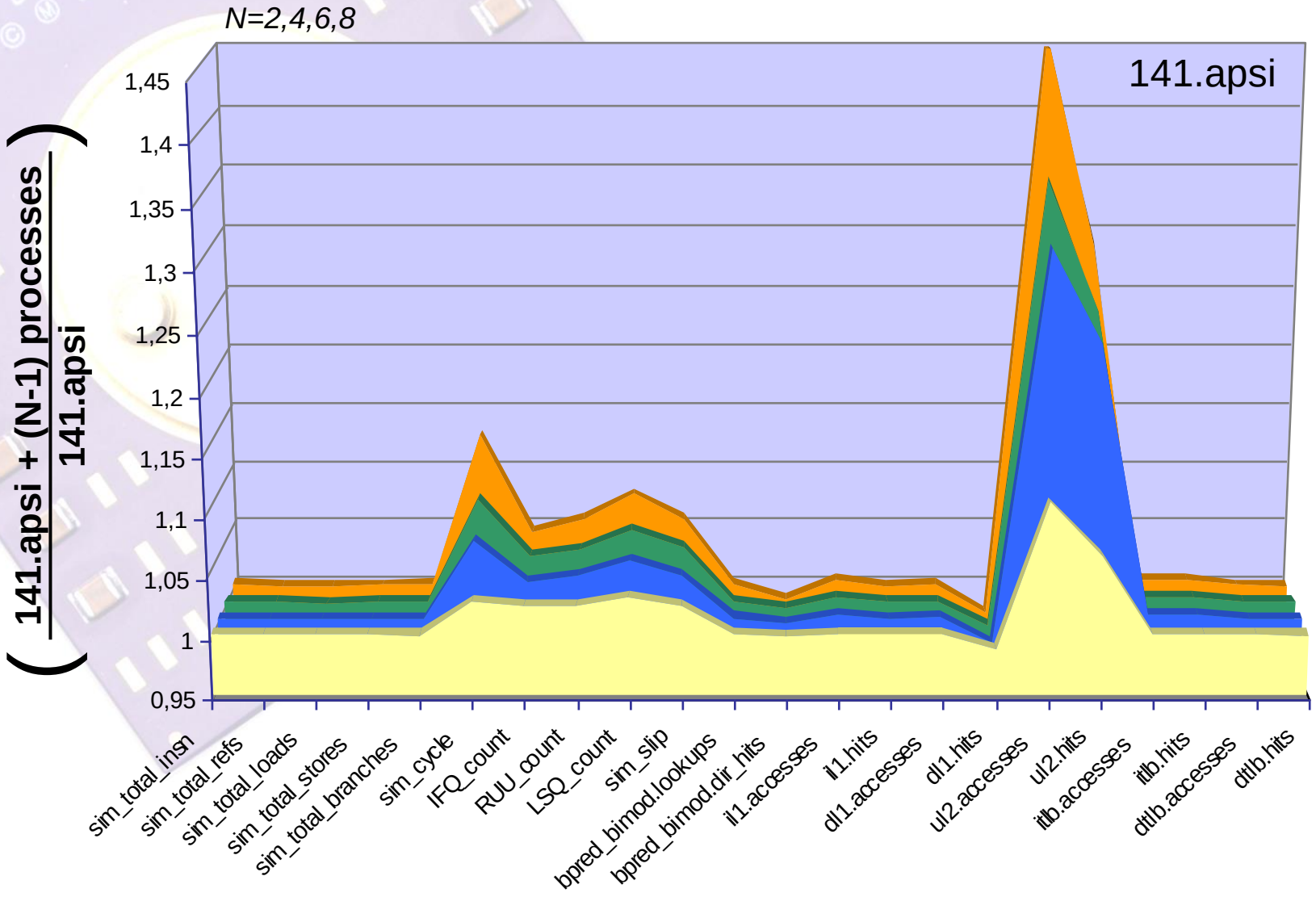
  <stdin> <exec> [arg] stdin file, executable and arguments

  <dtlb>, **<itlb>**, **<dl1>**, **<dl2>**, **<il1>**, **<il2>** flush structure at context switch

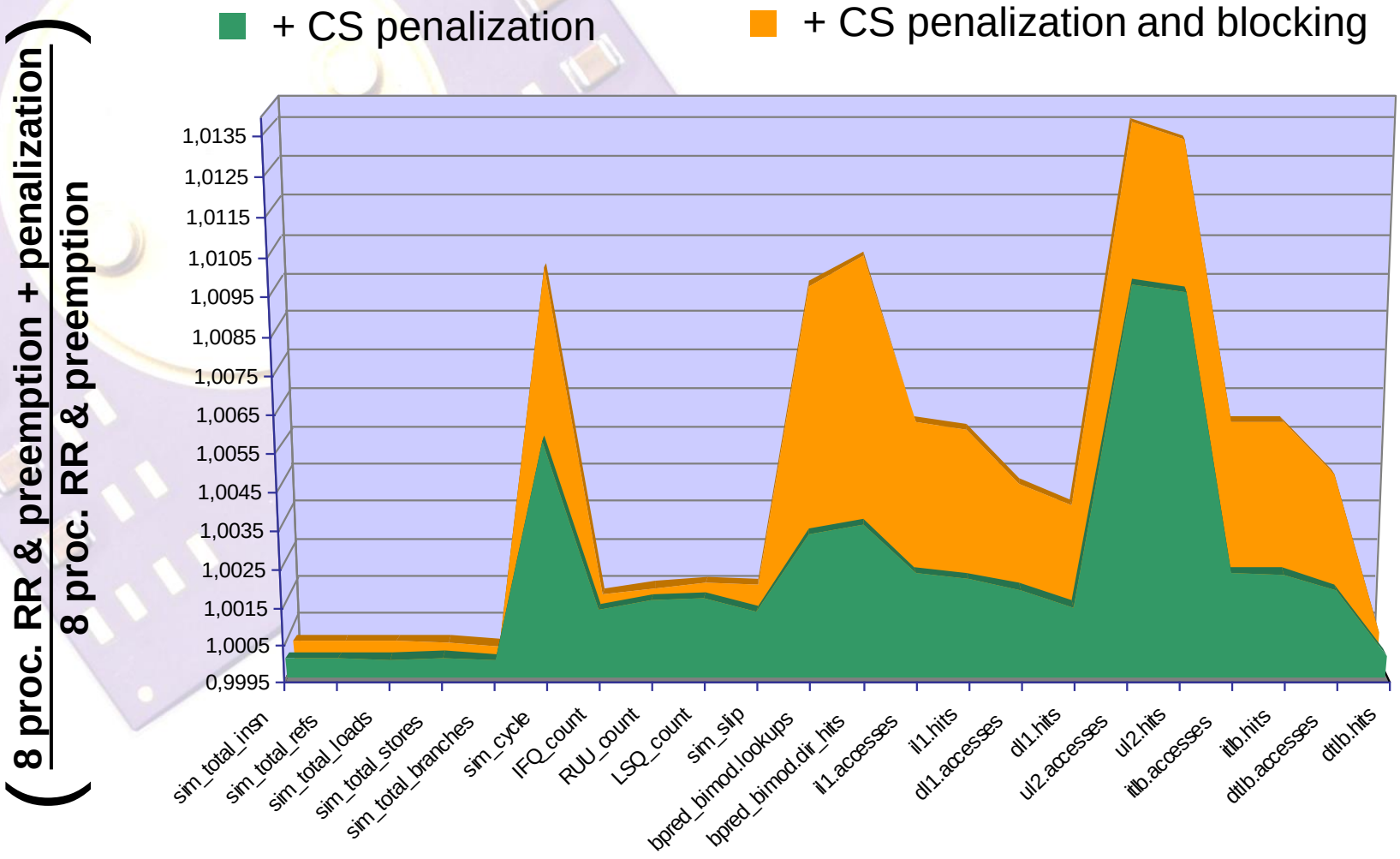
List of Executables Example

```
# -s <quan> <penal> <blk%> <bskip> <appr> <inter>
  <squan> <stdin> <sexec> [sargs]
  -s  15000  150  60.0  false  true  false  200
  NULL  $TESTS/test-printf

# -f <dtlb> <itlb> <dl1> <dl2> <il1> <il2>
  -f false  false  true  false true  false

# -e <pri> <exit> <wakeup> <fastfwd> <blk%> <maxins>
  <stdin> <exec> [arg]
  -e  2  s  6000  0  100.0  0  NULL  $TESTS/test-
  args  ARG1 ARG2
  -e  5  t  0  500  20.0  0  NULL  $TESTS/test-math
```

Simulation of *N* different processes



Simulation of 8 processes with penalizations



Interpretation of simulations

- The lost of performance in multitasking (more cycles) is reflected in
  - Higher L1 cache miss rate & more L2 cache accesses (higher latency)
  - Higher architecture queues (RUU, LSQ, IFQ) occupancy
  - More branch predictor accesses & higher TLB miss rate
- More concurrent processes produce less efficiency
- System overheads (scheduler and blocking) produce minimum lost of performance (previous simulation)

Conclusions

- Multitasking operating systems produce lost of performance
  - Context switch overhead
  - Cache and speculative misses
- Research Topics
  - hardware CS approach
  - cache replacement algorithms
  - branch prediction algorithms
  - cache memory structure/configuration

References

- T.M. Austin, "*A User's and Hacker's Guide to the SimpleScalar Architectural Research Tool Set (for tool set release 2.0)*", Intel MicroComputer Research Labs, gener 1997
- A. Silberschatz, P. Galvin, G. Gagne, *Sistemas Operativos (6th edition)*, Limusa Wiley, 2002
- J.S. Evans, R.H. Eckhouse, *Alpha™ RISC Architecture for programmers*, Prentice Hall, 1999
- *www.simplescalar.com*