Outline for Discussion

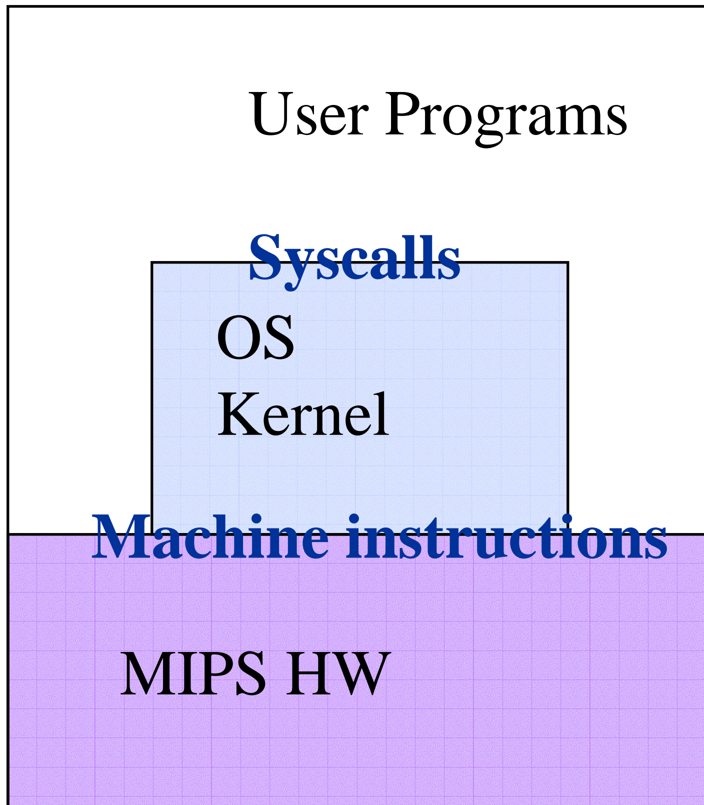
- Objective: To provide a basic understanding of the structure of Nachos and how to get started on Programming Assignment #1

- Slides: Dr. Ellis and adapted from Dr. Stark's "A Quick Tour of **NACHOS**"

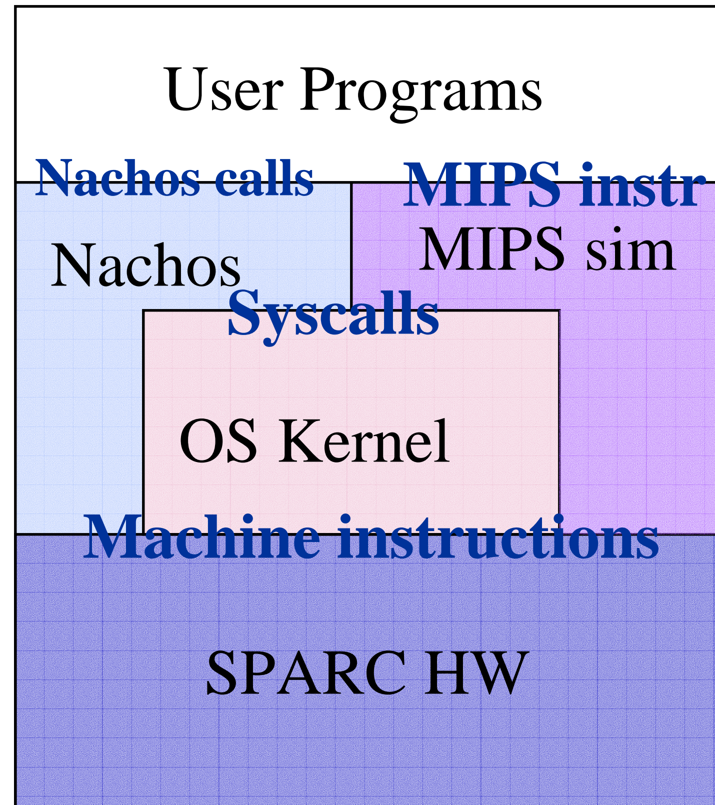
Nachos

- Real (instructional) operating system
- Runs partially on real hardware(kernel), and partially on simulated hardware(user programs)
- Kernel written in C++
- user programs written in ANSI C, compiled for MIPS R2/3000.

Nachos



Abstract View



Reality

Nachos: simulated peripherals

- Console terminal device
- Disk device
- Timer device
- *Network interface*

Nachos

- start with very limited functionality
  - Only one user program at a time
  - Contiguous loading at low end of physical memory
  - no system calls
- Improve nachos by designing and implementing missing and inadequate portions of the system

Nachos Assignments

- Lab 1: The Trouble with Concurrent Programming
- Lab 2: Threads and Synchronization
- Lab 3: Programming with Threads
- Lab 4: Multiprogrammed Kernel
- Lab 5: I/O
- Lab 6: Virtual Memory

Lab 1: The Trouble with Concurrent Programming

- become familiar with Nachos and the behavior of a working (but incomplete) thread system.
- use what is supplied to experience the joys of concurrent programming.
- all use of the Nachos thread primitives will be internal to your Nachos operating system kernel
- For now, you are using these internal Nachos primitives to create simple concurrent programs as applications under Unix (e.g., Solaris).

Learning about nachos

- You cannot learn all about software systems from textbooks.
- Instead, read the source code for systems that other people have written.
- As soon as possible, begin reading over the **NACHOS** source code,
- try to understand where the various pieces of the system live, and how they fit together.
- It will take a while to develop an understanding. Don't worrry!.

Learning about nachos

- CVS
  - This directory contains control information for the CVS source code management system. Similar subdirectories are present at each level of the tree. You should not change anything in these directories.
- Makefile
  - This file controls what happens when you type gmake in the code directory. It describes how to completely compile all the **NACHOS** code and user application programs.

Learning about nachos

- Makefile.common
  - This file contains the lists of source and object files that go into making each version of **NACHOS**. Each time you add a new source or object file of your own, you will have to edit Makefile.common and add this file to the appropriate lists.
- Makefile.dep
  - This file contains some system-dependent parameters that control the building of **NACHOS**. You probably don't need to change this.

Learning about nachos

- threads
    - This directory contains source code to support threads (lightweight processes) and synchronization. The threads support is fully functional, though some of the synchronization primitives have not been implemented. Implementation and testing of these synchronization primitives will be part of your job for Homework #2.

Learning about nachos

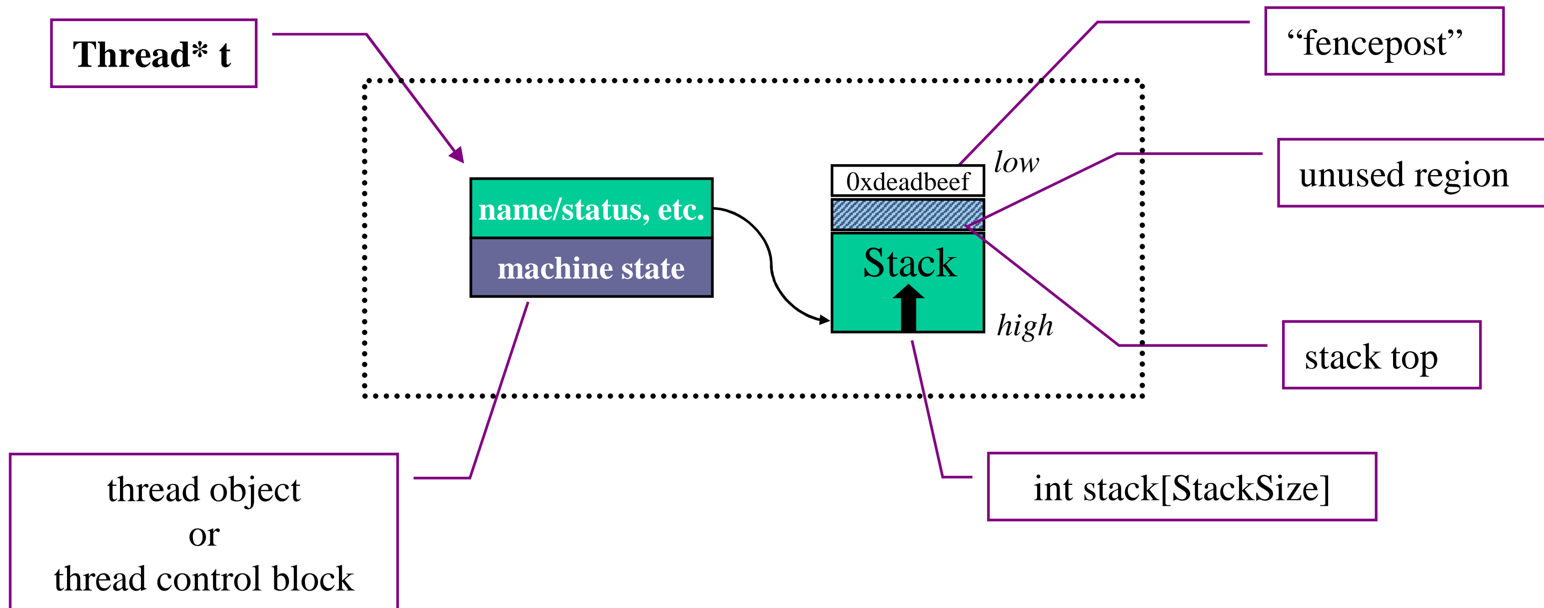
- userprog
  - This directory contains source code to support the loading and execution of user application programs.
- vm
  - This directory will contain the source code for the virtual memory subsystem, when you implement it for Homework #6.
- filesys
  - This directory contains source code for a ``stub'' implementation of the **NACHOS** filesystem. This implementation is very limited and incomplete. Your job in Homework #5 will be to rewrite and improve it.

Learning about nachos

- test
  - This directory contains source code for some simple **NACHOS** user application programs. It also contains Makefile for compiling these programs and converting them from COFF to NOFF.
- machine
  - This directory contains source code for the machine emulator. It might be instructive to look at some of the header files in this directory, but **you shouldn't have to modify** anything here.

A Nachos Thread

```
t = new Thread(name);
t->Fork(MyFunc, arg);
currentThread->Sleep();
currentThread->Yield();
```



Thread Operations

new thread - inits a thread control block

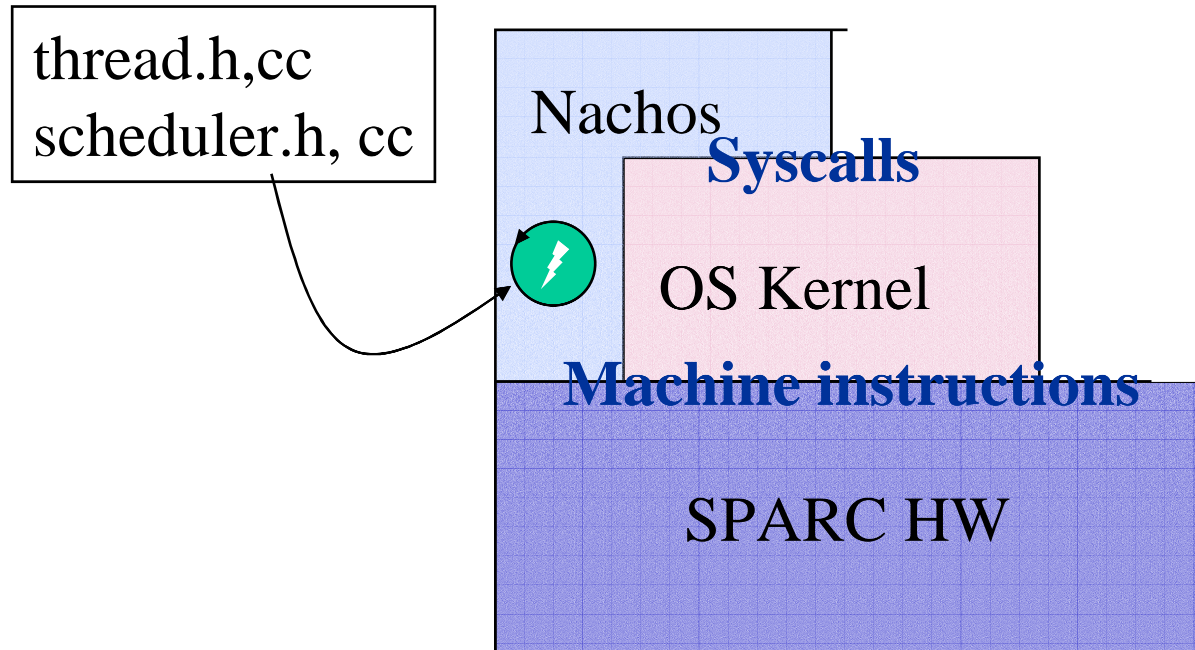
Thread::Fork - runs a specified procedure in the newly created thread (allocates stack and makes ready to run)

Thread::Finish - cleans up its state

Thread::Yield - gives up CPU - makes running thread ready to run and invokes scheduler to choose new running thread

Thread::Sleep - blocks thread (not on ready queue)

Nachos



Nachos Context Switches: Voluntary vs. Involuntary

On a uniprocessor, the set of possible execution schedules depends on *when context switches can occur*.

- ***Voluntary***: one thread explicitly yields the CPU to another.
  - A Nachos thread can suspend itself wth **Thread::Yield**.
  - It may also *block* to wait for some event with **Thread::Sleep**.
- ***Involuntary:*** the system *scheduler* suspends an active thread, and switches control to a different thread.
  - Thread scheduler tries to share CPU fairly by timeslicing.
  - Suspend/resume from a timer interrupt handler (e.g., **nachos -rs**)
  - This can happen "any time", so concurrency races can occur.

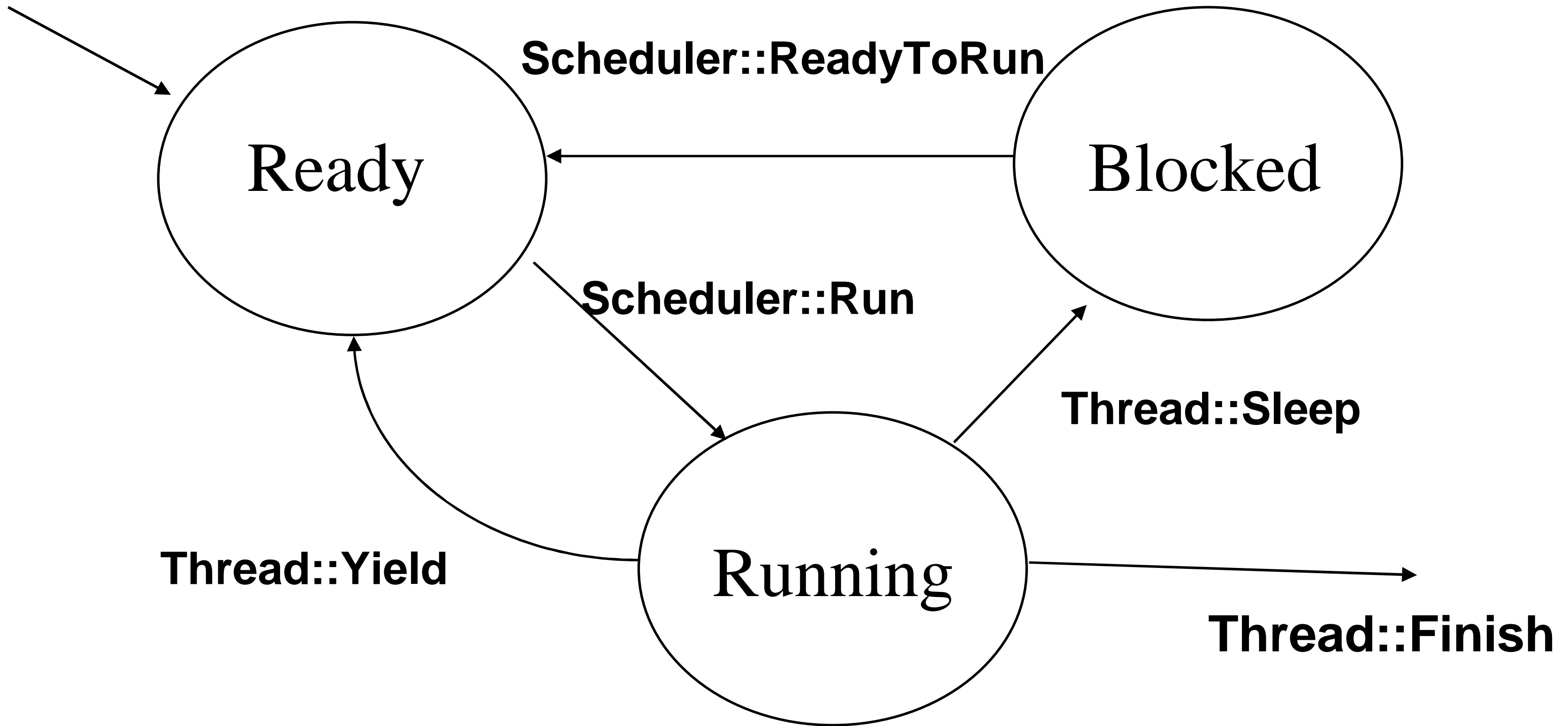
Blocking or Sleeping

- An executing thread may request some resource or action that causes it to *block* or *sleep* awaiting some event.
  - passage of a specific amount of time (a ***pause*** request)
  - completion of I/O to a slow device (e.g., keyboard or disk)
  - release of some needed resource (e.g., memory)
  - In Nachos, threads block by calling **Thread::Sleep*.***
- A sleeping thread cannot run until the event occurs.
- The blocked thread is awakened when the event occurs.
  - E.g., ***Wakeup*** or Nachos **Scheduler::ReadyToRun(Thread* t)**
- In an OS, processes may sleep while executing in the kernel to handle a system call or fault.

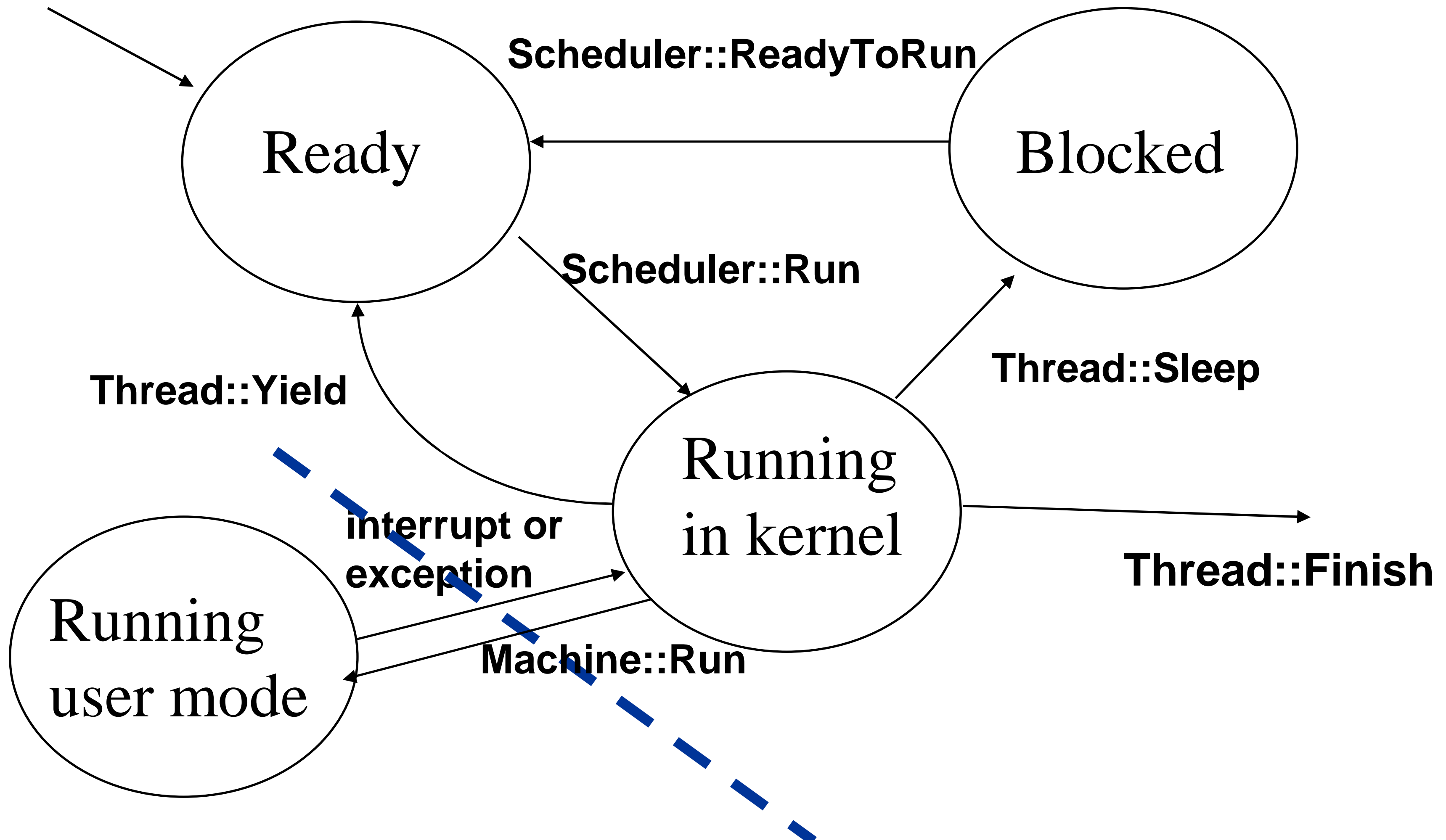
Nachos Thread State Transitions

t = new Thread(name);

t->Fork(MyFunc, arg);



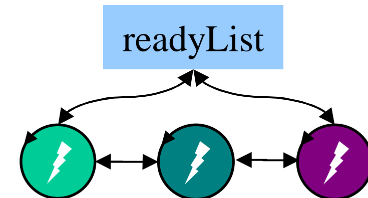
Nachos Thread State Transitions



The Nachos Scheduler

The core of Nachos is the *Scheduler* class:

- one global shared *scheduler* object
- pool of ready threads (the *ready list*)




```
new = scheduler->FindNextToRun();          /* get next ready thread */
scheduler->Run(t);                         /* run it */
```

Run calls ***SWITCH(currentThread, new)*** to suspend current thread and pass control to new thread.

A Nachos Context Switch

```
/*
 * Save context of the calling thread (old), restore registers of
 *  the next thread to run (new), and return in context of new.
 */
switch/MIPS (old, new) {
        old->stackTop = SP;
        save RA in old->MachineState[PC];
        save callee registers in old->MachineState

        restore callee registers from new->MachineState
        RA = new->MachineState[PC];
        SP = new->stackTop;

        return (to RA)
}
```

*Save current stack pointer and caller's return address in **old** thread object.*

Caller-saved registers (if needed) are already saved on the thread's stack.

Caller-saved regs restored automatically on return.

*Switch off of **old** stack and back to **new** stack.*

*Return to last procedure that called switch in **new**.*