NachOS

Sebastian Biemüller
(temporarily not available),
Daniel Kirchner

Course Team

- Personal Meetings (R154)
  - Sebastian Biemüller biemueller@ira.uka.de
  - Daniel Kirchner kirchner@ira.uka.de
- Consultation Time:

  Monday 14:30-15:30

Overview

- Motivation
- NachOS Architecture
- NachOS Assignments

Closer Look at the Code:

- NachOS CPU Emulation
- NachOS Syscall

Organizational Issues

Motivation

- You can not sleep anyway
- You learned a lot and want to use it
- You want some bonus points
- You want to be a witty octopus juggling daily **new balls** of different size on the back of a **jumping dolphin** at the **shore of Waikiki** and take care of **sharks** and other bad guys around you.

What Is NachOS? (1)

- NachOS:
  - Not Another Completely Heuristic Operating System
- An educational OS written by Tom Anderson and his students at UC Berkeley in C++

  http://www.cs.washington.edu/homes/tom/nachos/

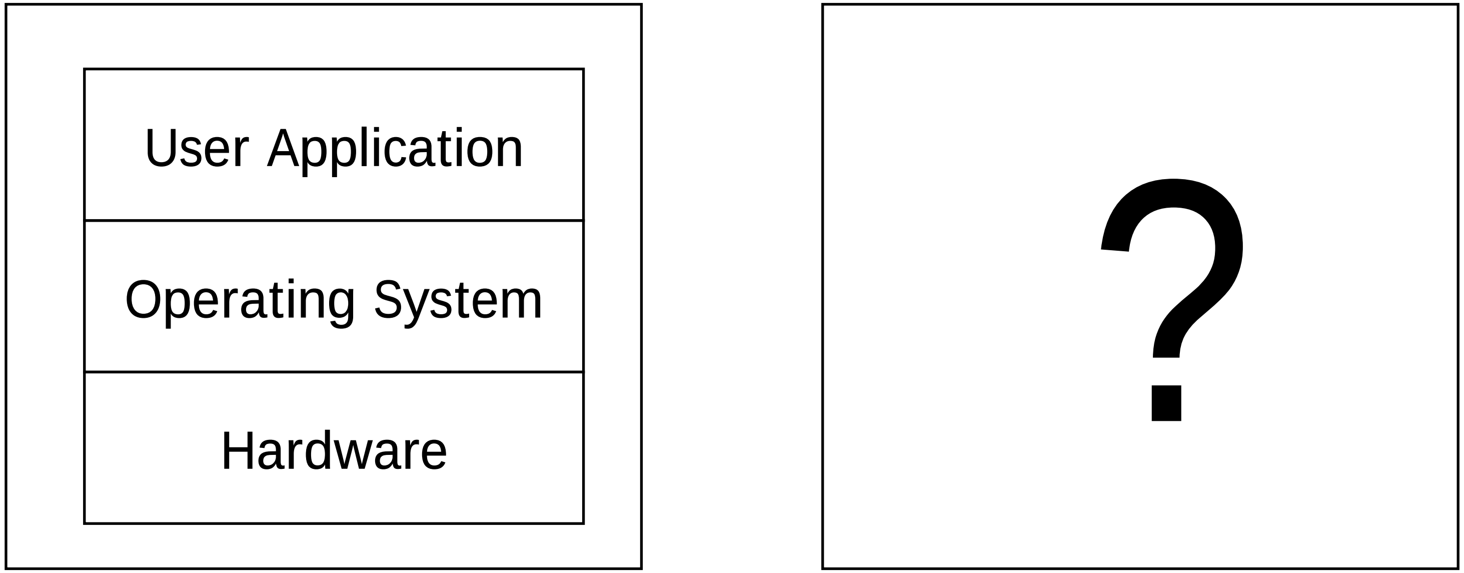
What Is NachOS? (2)

- An educational OS used to:
  - Teach monolithic kernel design and implementation
  - Do experiments
- Fact:
  - Real hardware is difficult to handle.
  - May break if handled wrong.
- Approach:
  - Use a virtual MIPS machine
  - Provide some basic OS elements

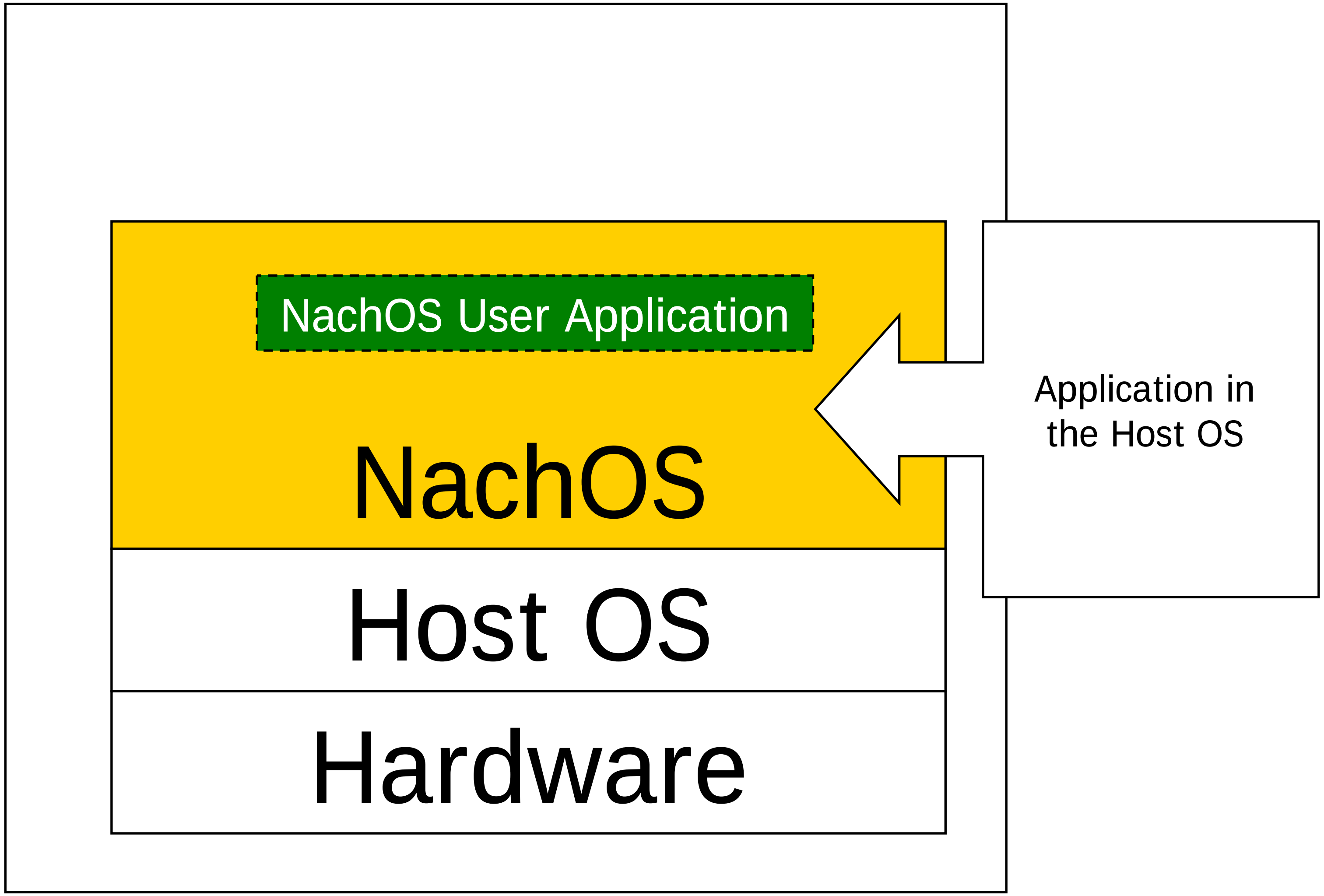
NachOS Architecture: Environments

Common System

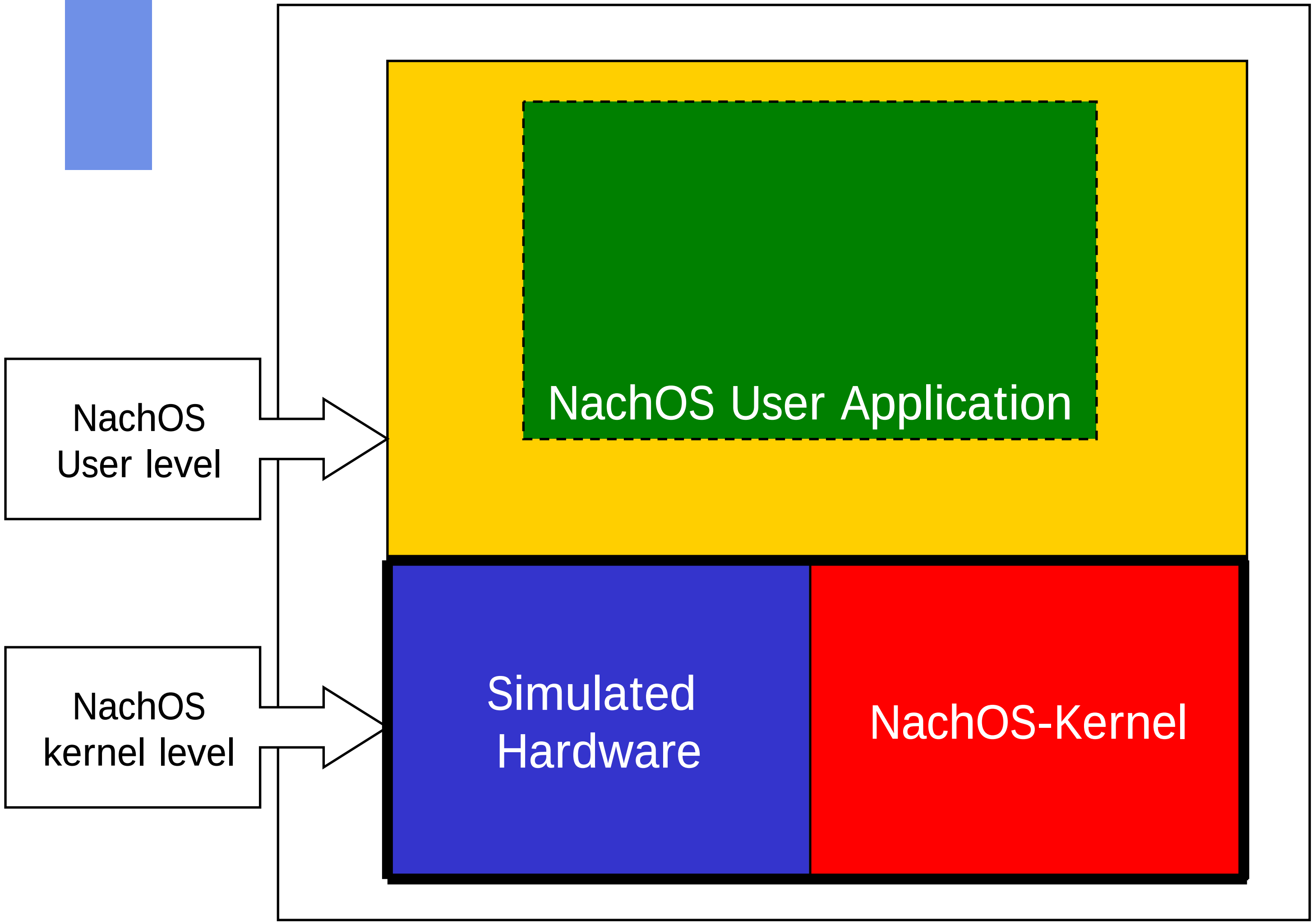
NachOS



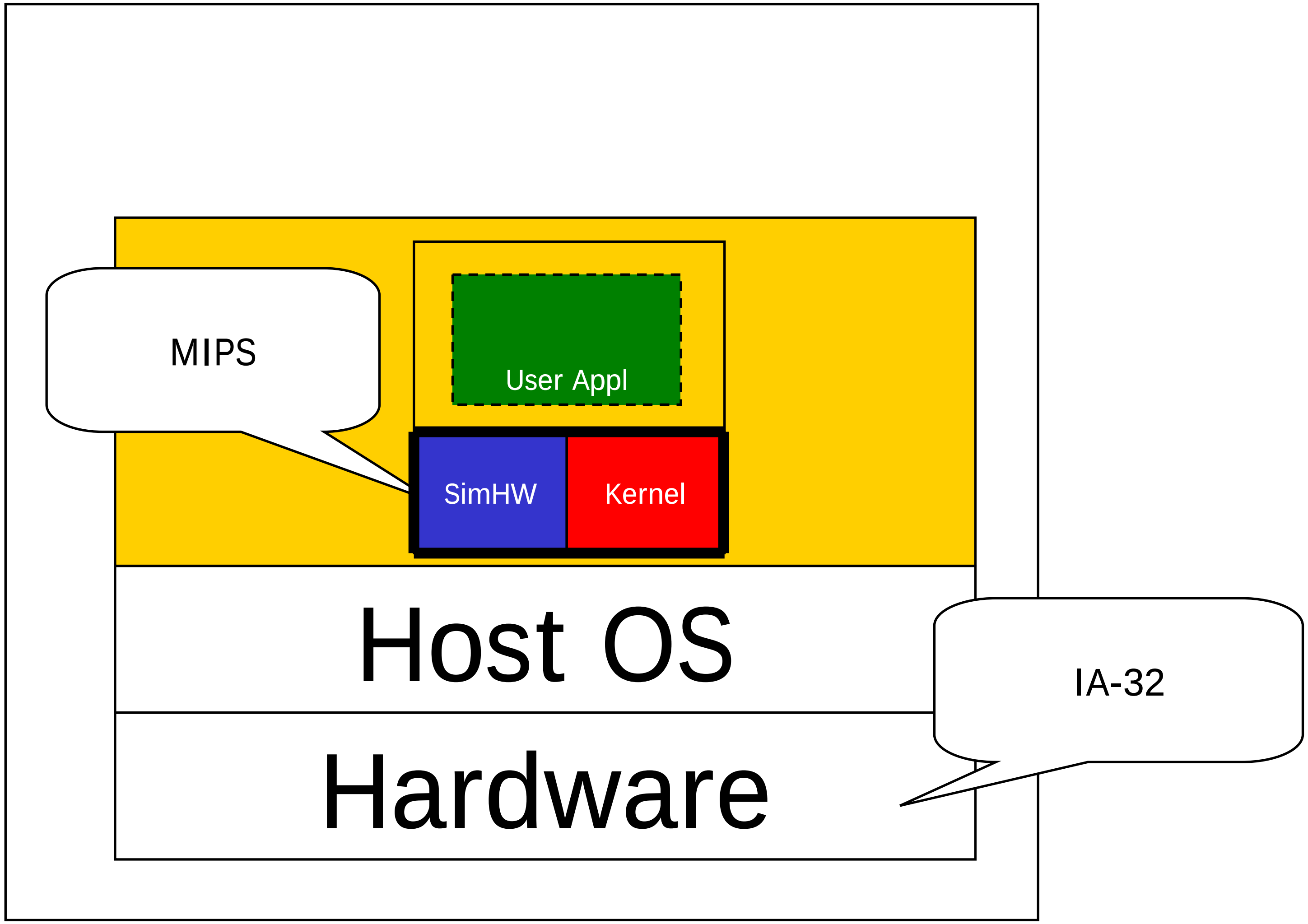
NachOS Environment



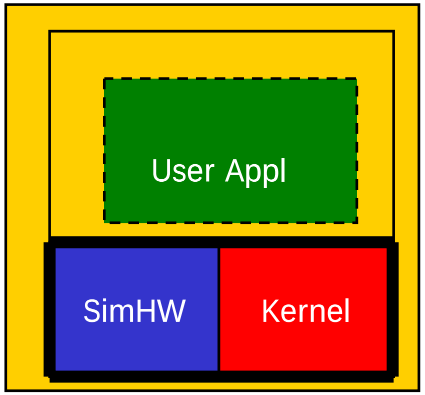
NachOS Arch – Inside Application



NachOS Environment



Code Inspection

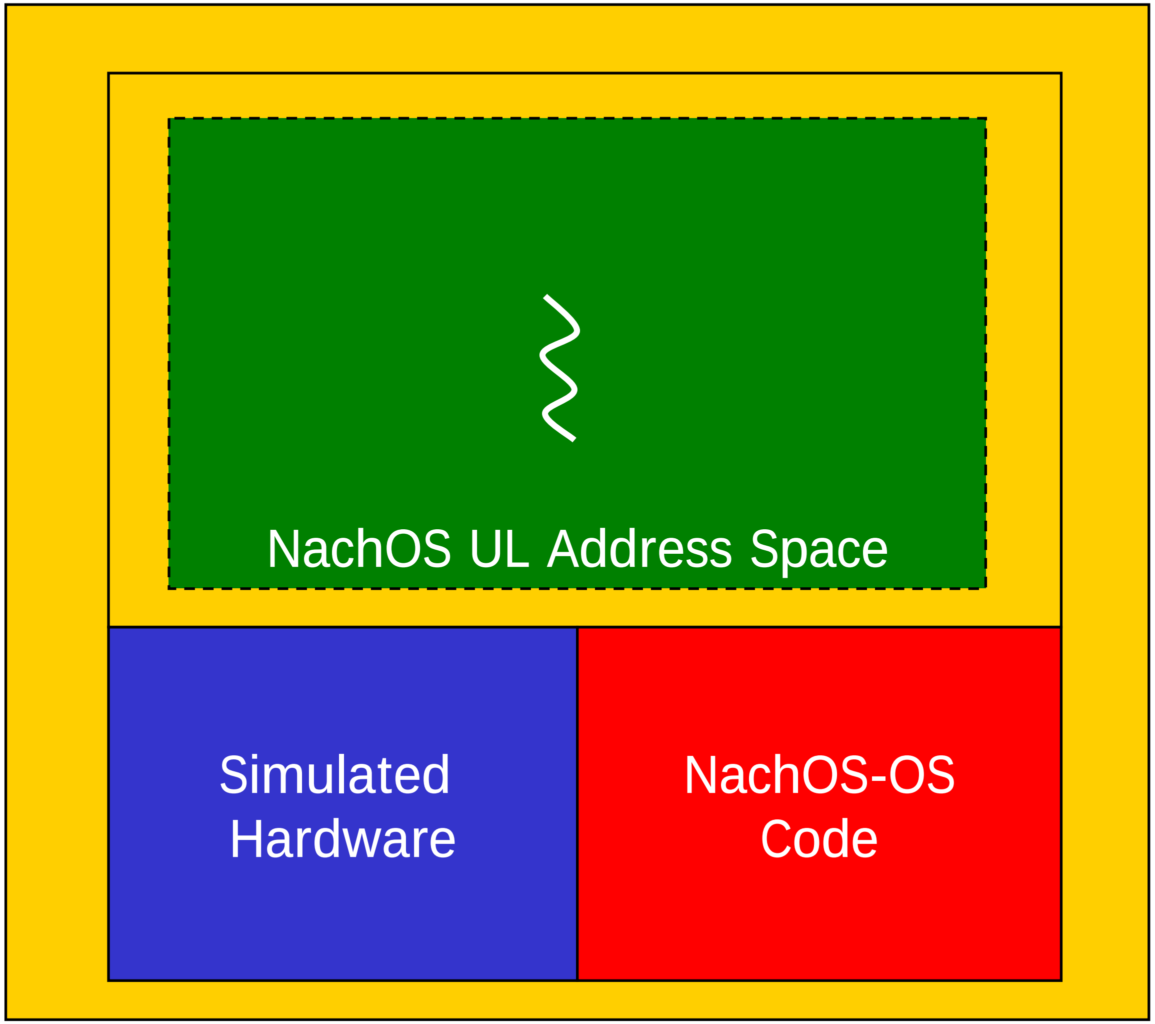


Dir	Target	HW/SW	Level	Content
/threads	IA-32	SW	NKL	KLT Management
/machine	IA-32	HW	n.def	HW Simulation
/userprog	IA-32	SW	NKL	UL Representation Struct.
/filesys	IA-32	SW	NKL	NachOS-Kernel FS
/disk	IA-32	HW	n.def	Simulated Hard Disk
/test	MIPS	SW	NUL	User Applications

NachOS

Assignments

NachOS Architecture – Currently

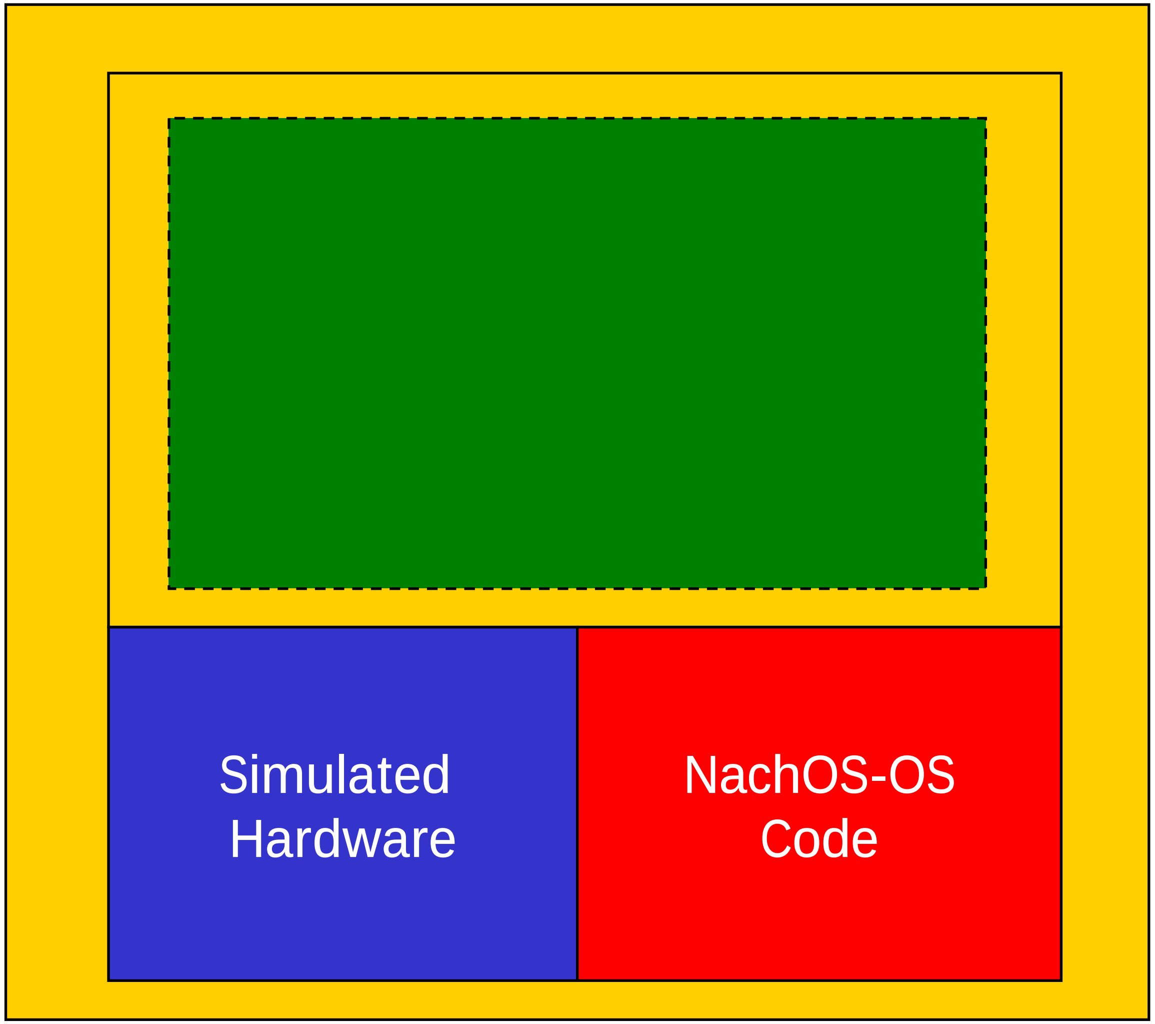


NachOS UL Address Space

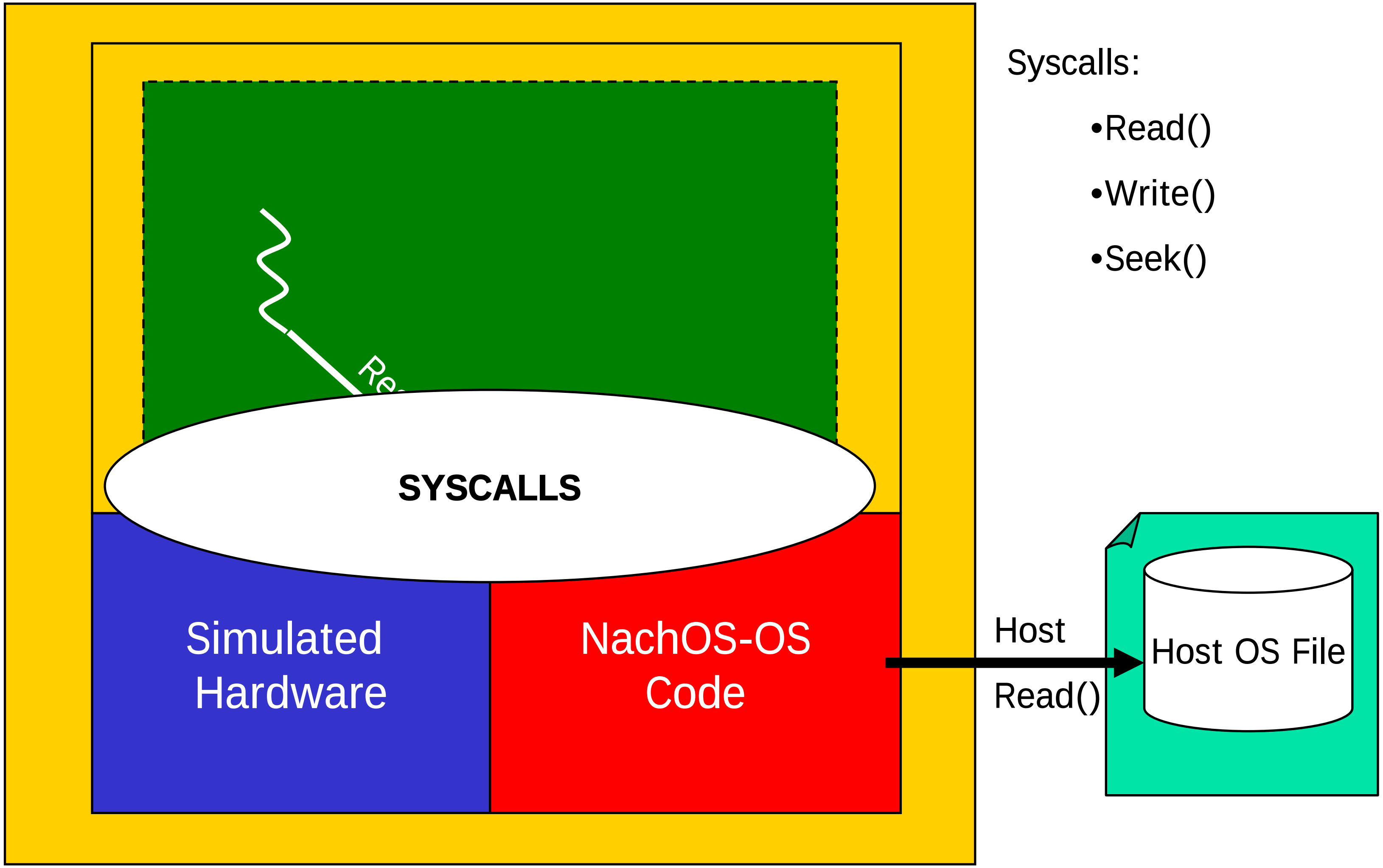
Simulated Hardware

NachOS-OS Code

NachOS Architecture – Currently



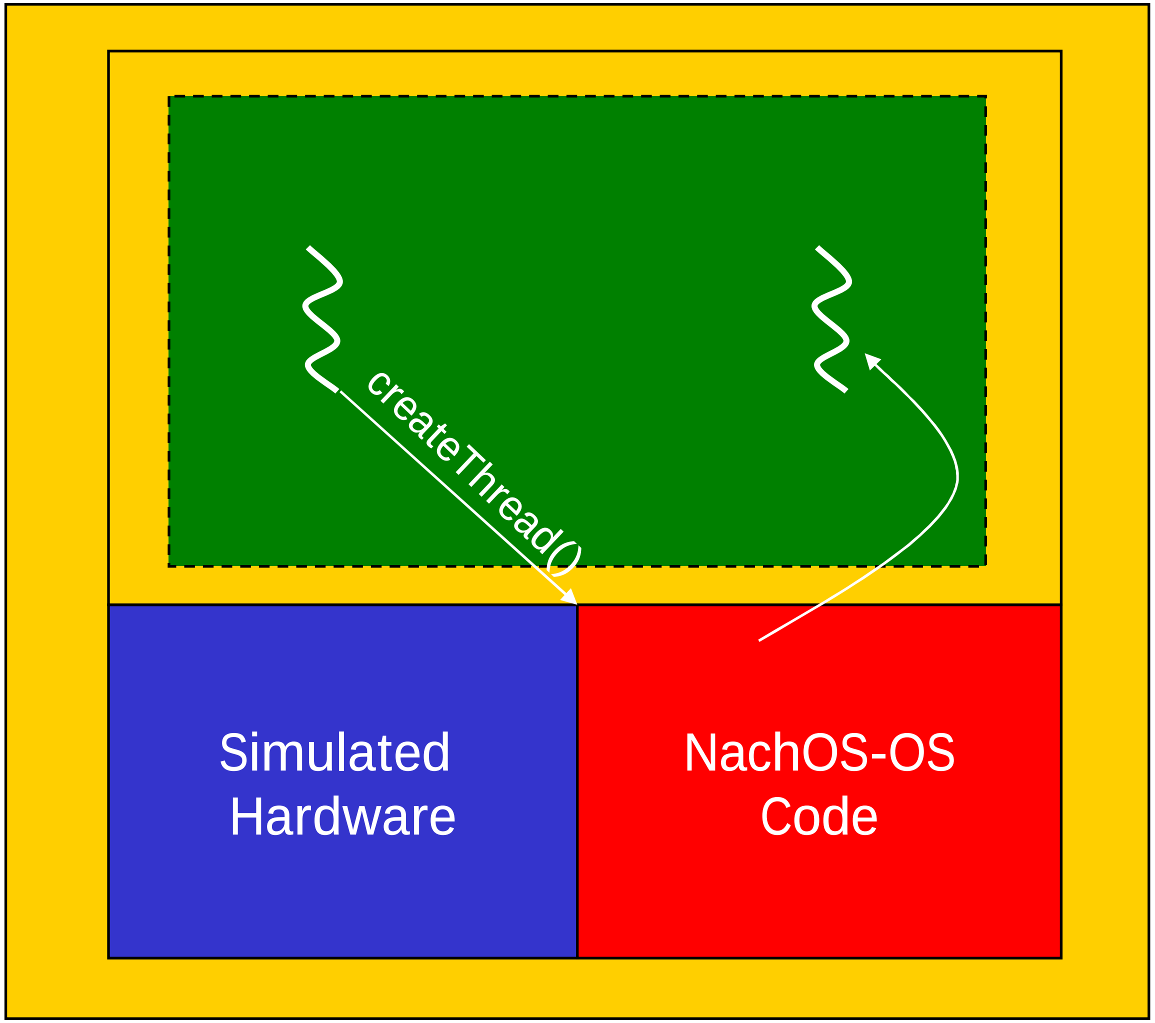
NachOS Architecture – Assignment 2



Syscalls:

- Read()
- Write()
- Seek()

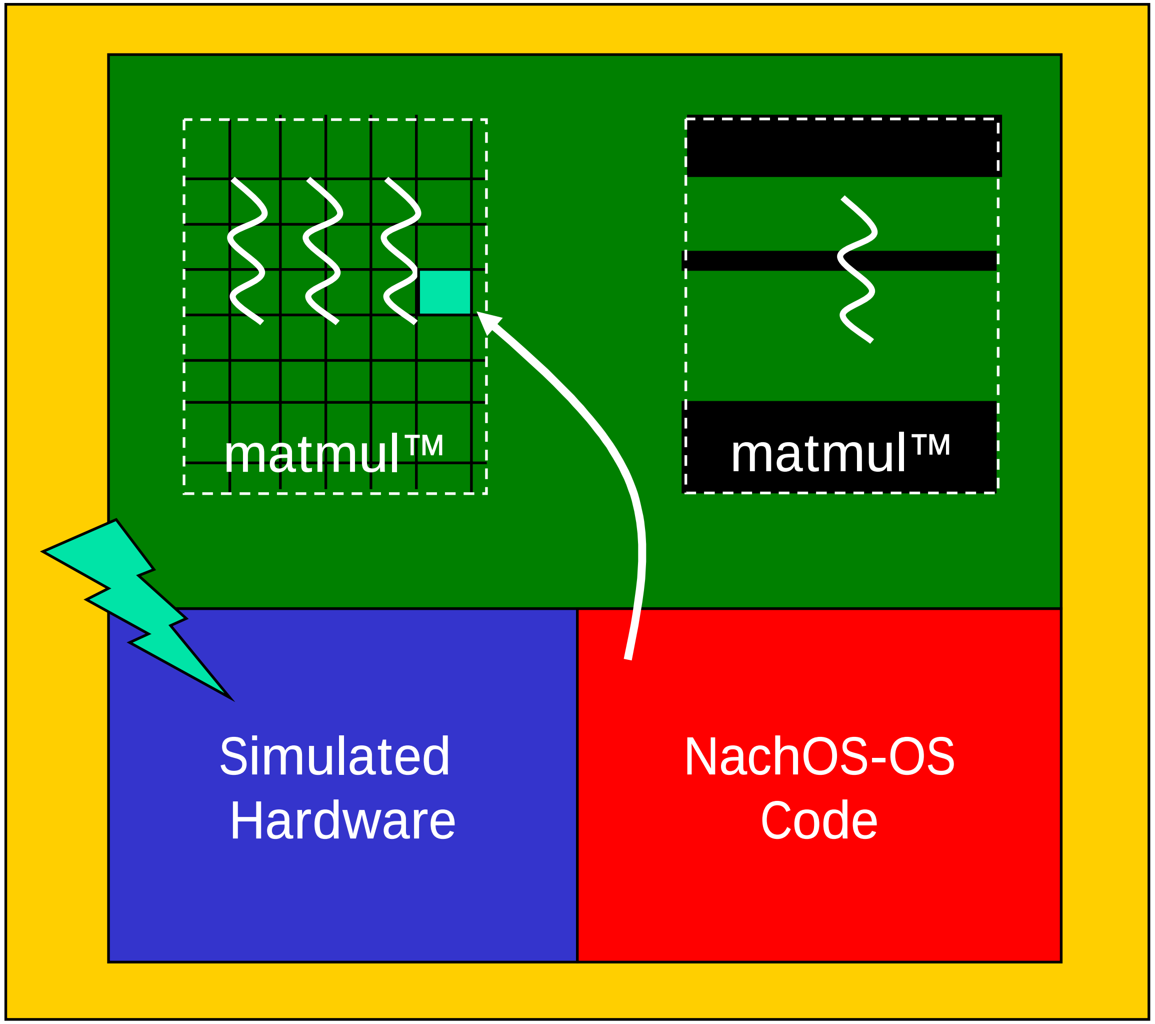
NachOS Architecture – Assignment 2



Syscalls:

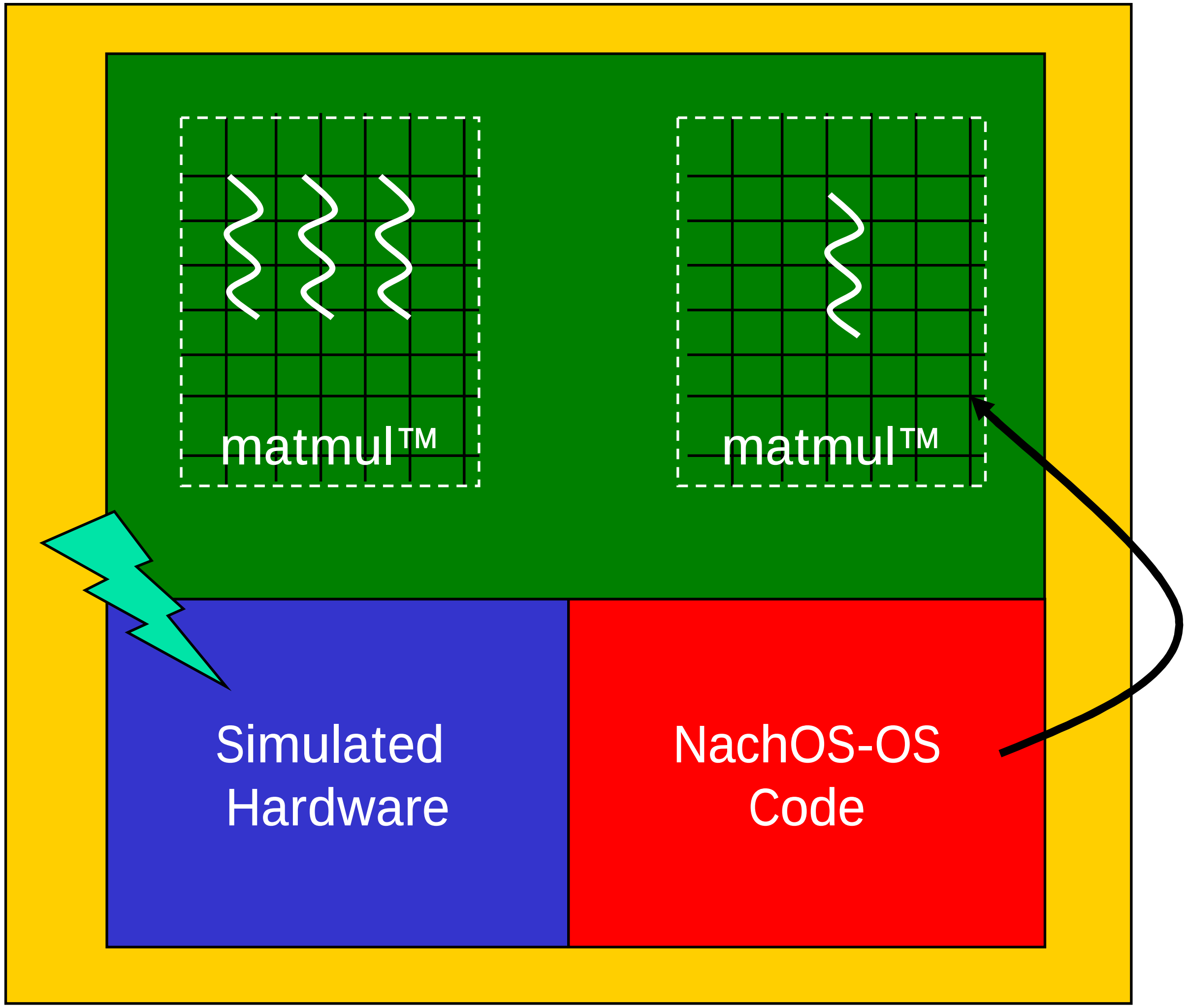
- ThreadCreate()
- ThreadJoin()
- ThreadYield()

NachOS Architecture – Assignment 3a



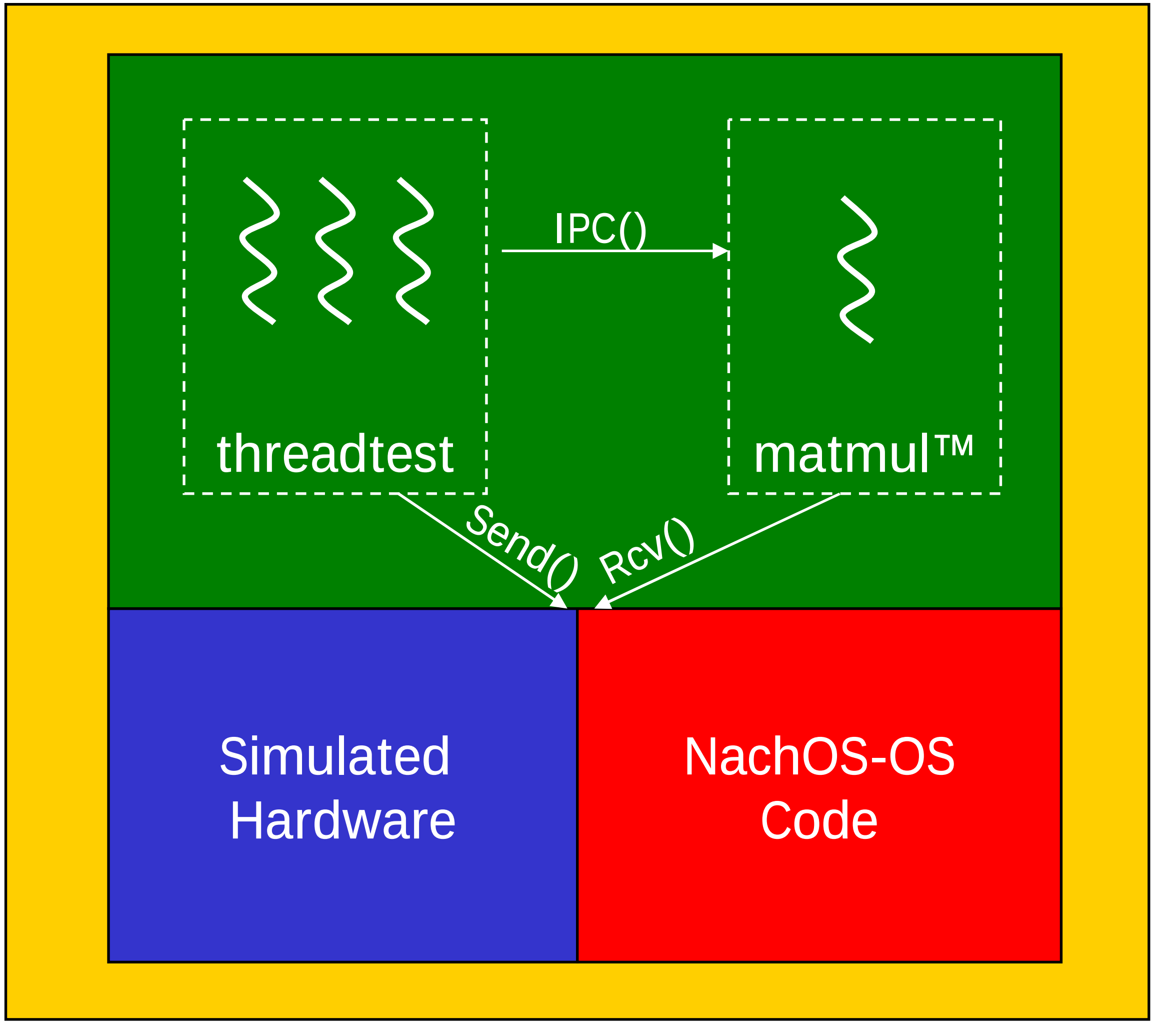
- Addr. Space Design
- Virtual Memory
- Paging
- Binary Loading

NachOS Architecture – Assignment 3a



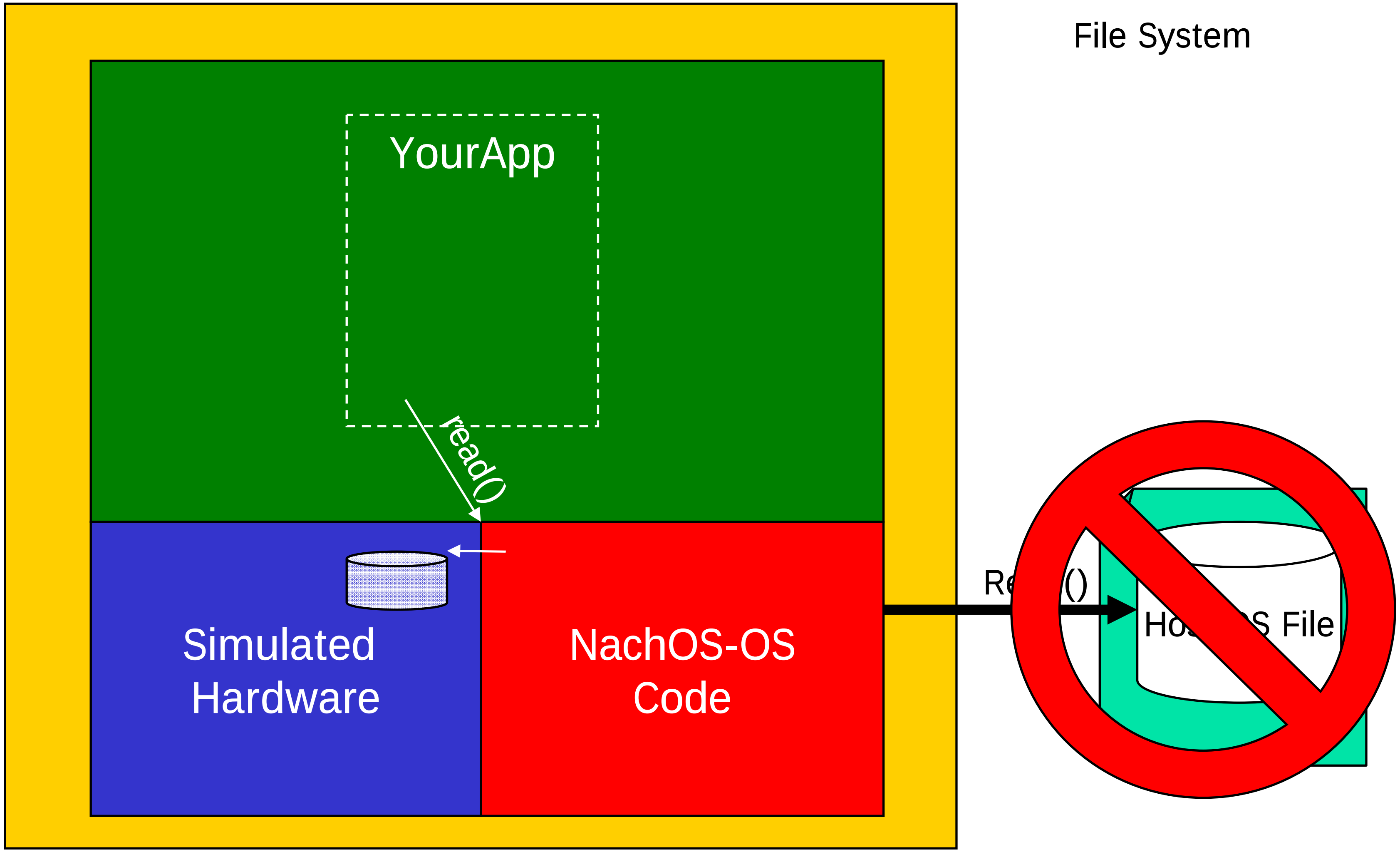
- Addr. Space Design
- Virtual Memory
- Paging
- Binary Loading

NachOS Architecture – Assignment 3b

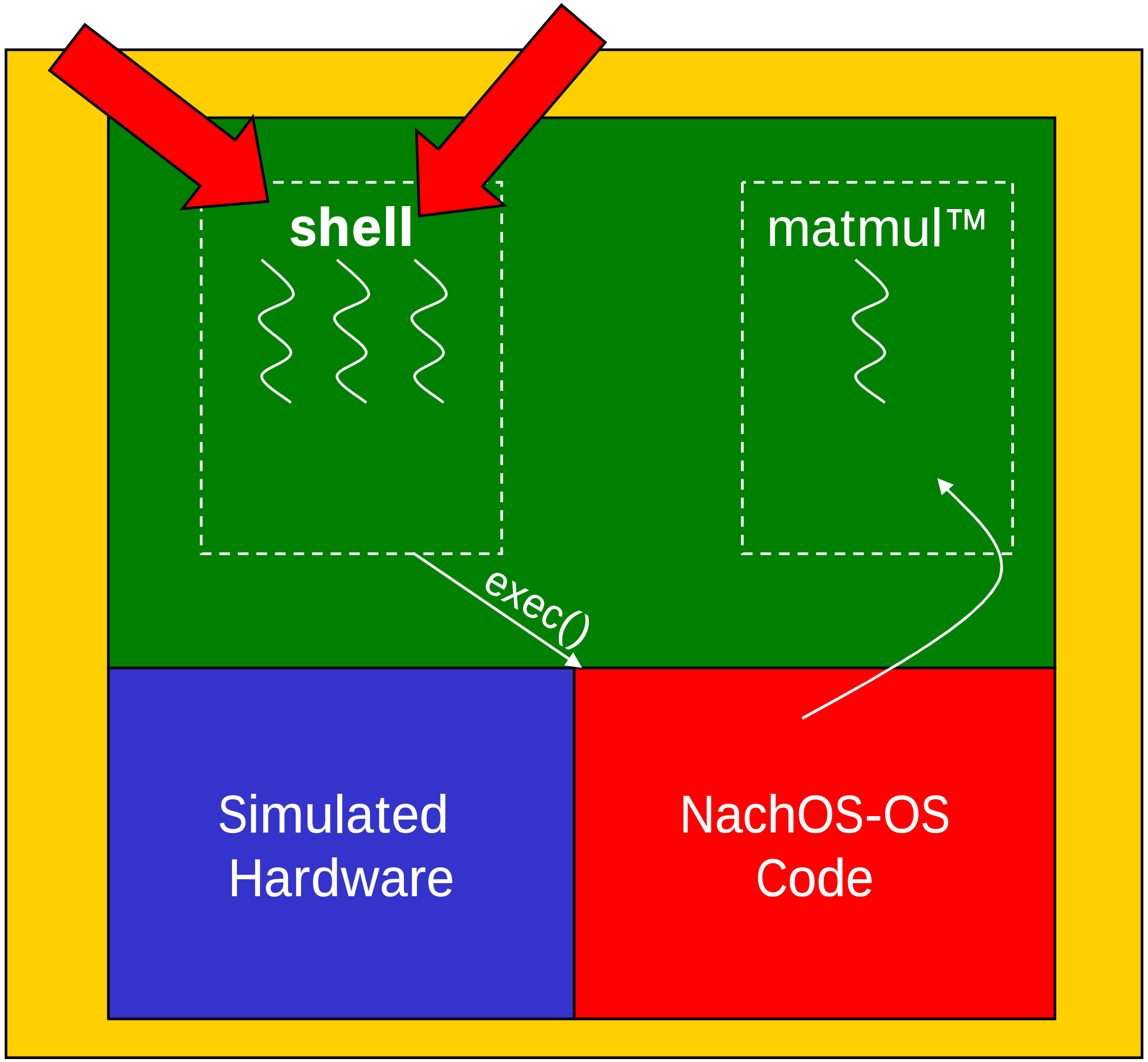


- IPC

NachOS Architecture – Assignment 4a



NachOS Architecture – Assignment 4b



Write a shell!

Part 2

A closer look at the NachOS code

Assignment 2

rn;

Ooops, too close...

What Do I have to do to run NachOS?

1.) Download CygWin or get access to a Linux machine (recommended)

2.) Download NachOS

3.) Download CrossCompiler

4.) Build NachOS

5.) Build coff2noff

6.) Build user test programms

7.) Have fun!

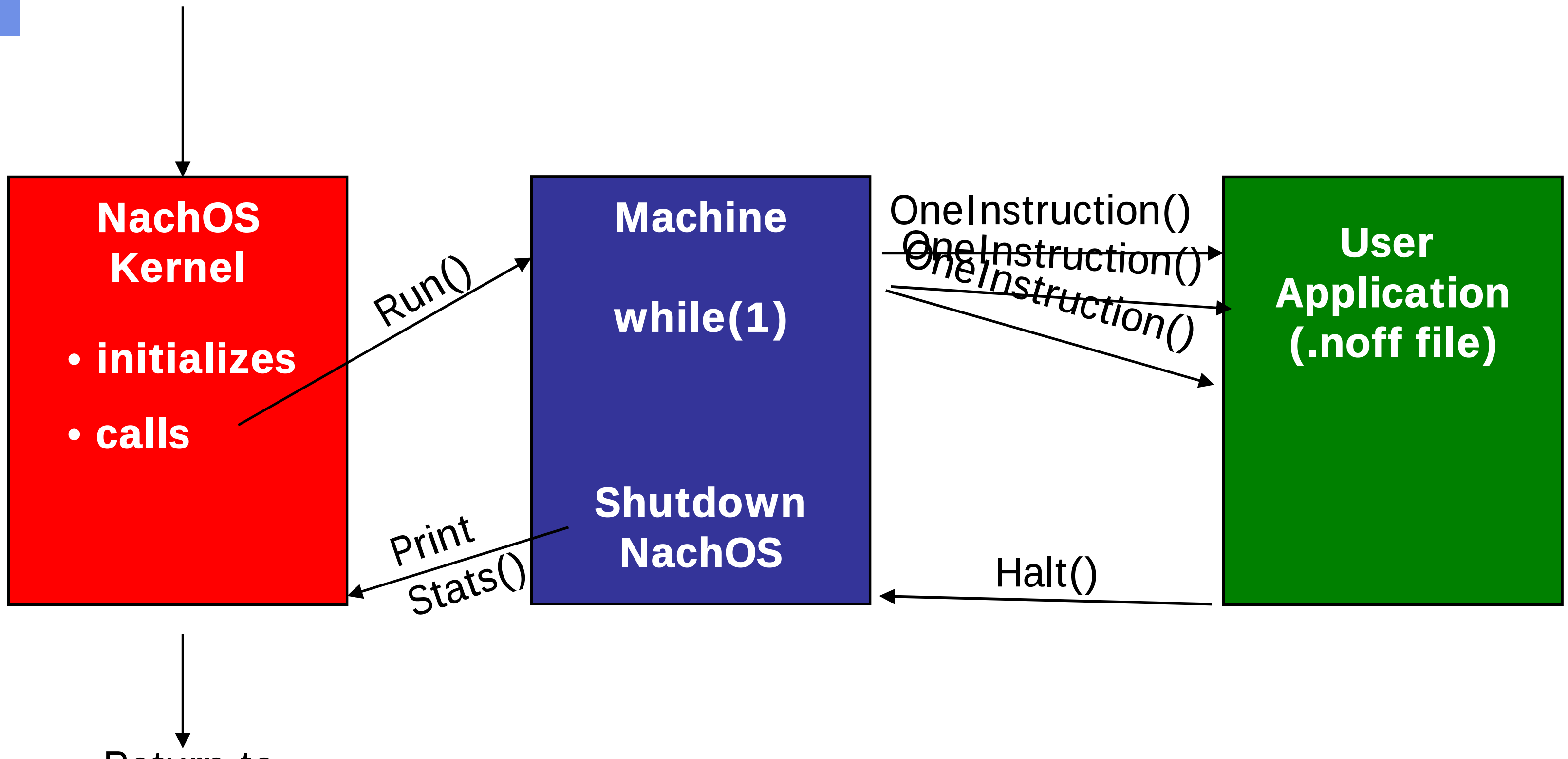
Where do we want to go today?

```
kirchner@yo_mama:~/nachos/$ ./nachos –x ../test/add.noff

tests summary: ok:0
Machine halting!

Ticks: total 28, idle 0, system 10, user 18
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
network I/O: packets received 0, sent 0

kirchner@yo_mama:~/nachos/$
```

How does NachOS work?

Start NachOS binary

(./nachos)



NachOS Kernel

- **initializes**
- **calls**

Run()

Machine

while(1)

OneInstruction()

OneInstruction()

OneInstruction()

User Application (.noff file)

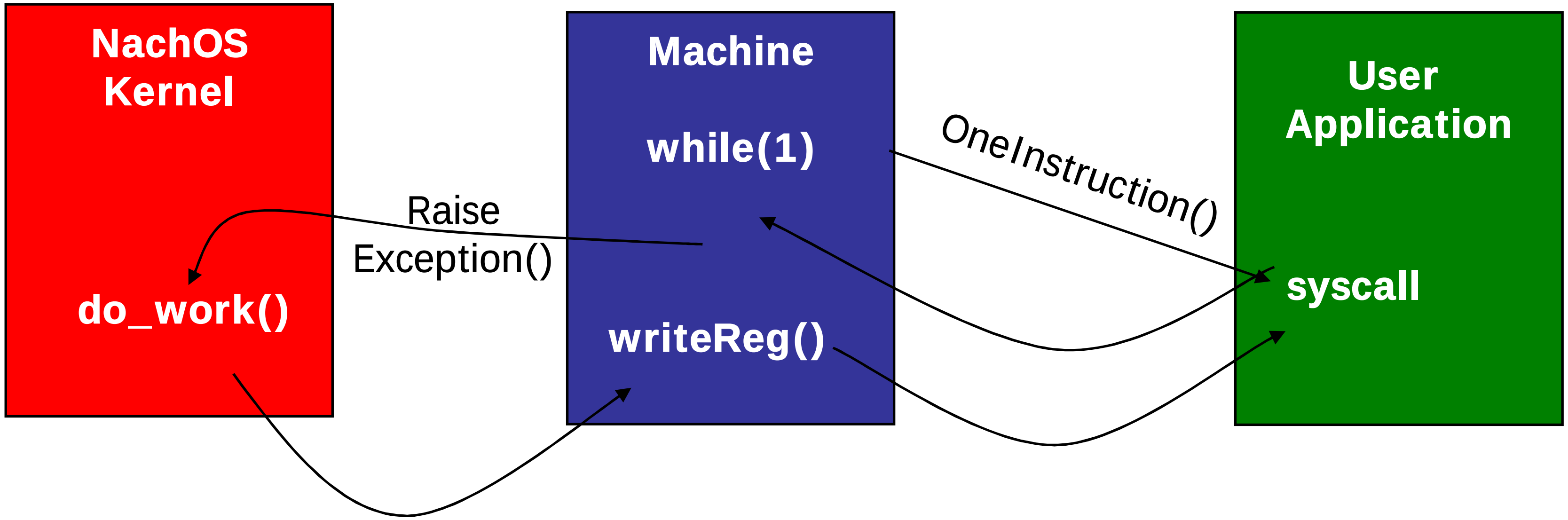
Halt()

Shutdown NachOS

Print Stats()

Return to HostOS shell

How does NachOS work? (2)



Example: creating the “Sub”-Syscall

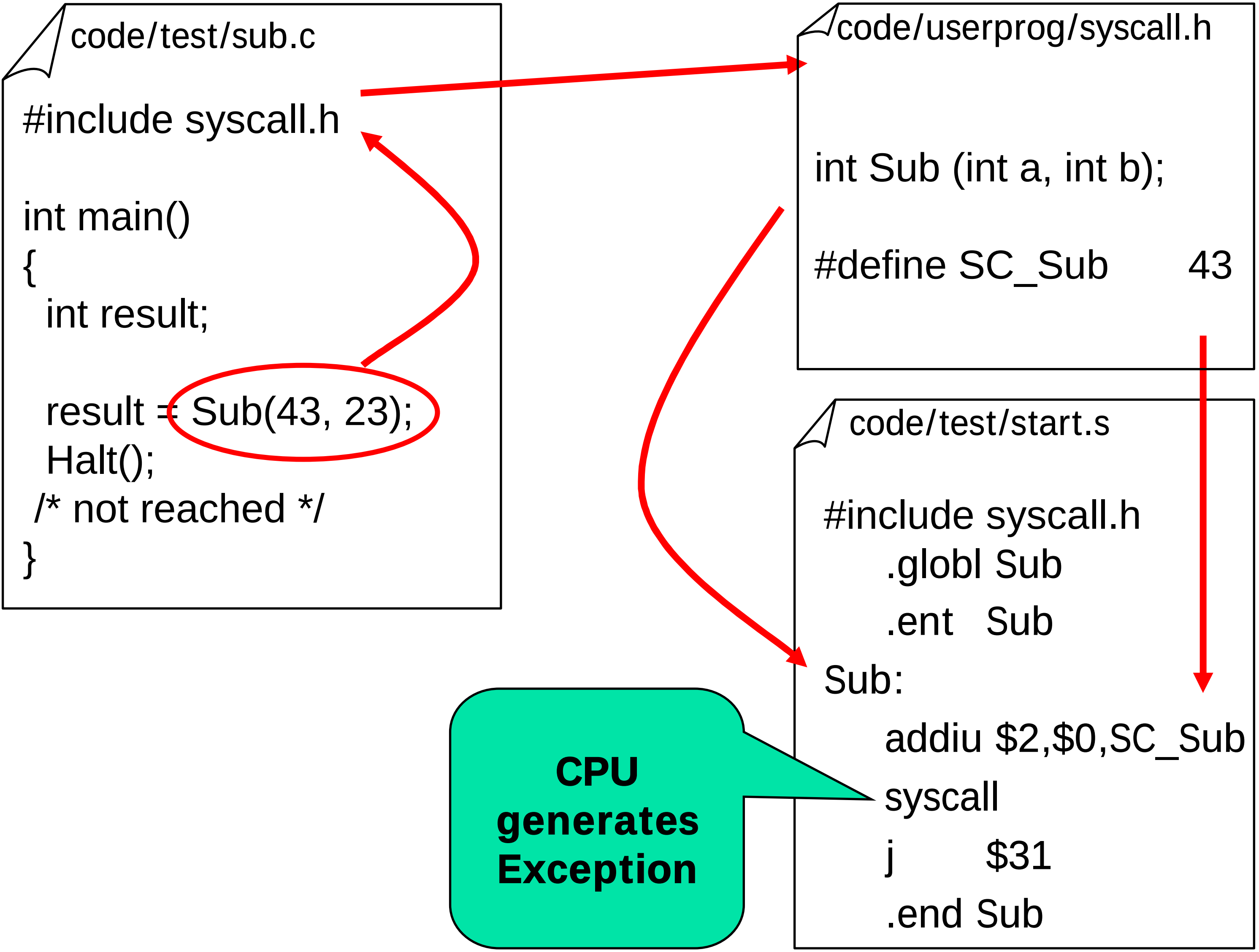
- User-Level Test Program
- Kernel Syscall Implementation

Approach (didactical):

1. Write User-Level Program
2. Define Syscall Number
3. Design Syscall Interface
4. Implement Syscall

User-Level Program

Syscall Binding



code/test/sub.c

```
#include syscall.h

int main()
{
  int result;

  result = Sub(43, 23);
  Halt();
 /* not reached */
}
```

code/userprog/syscall.h

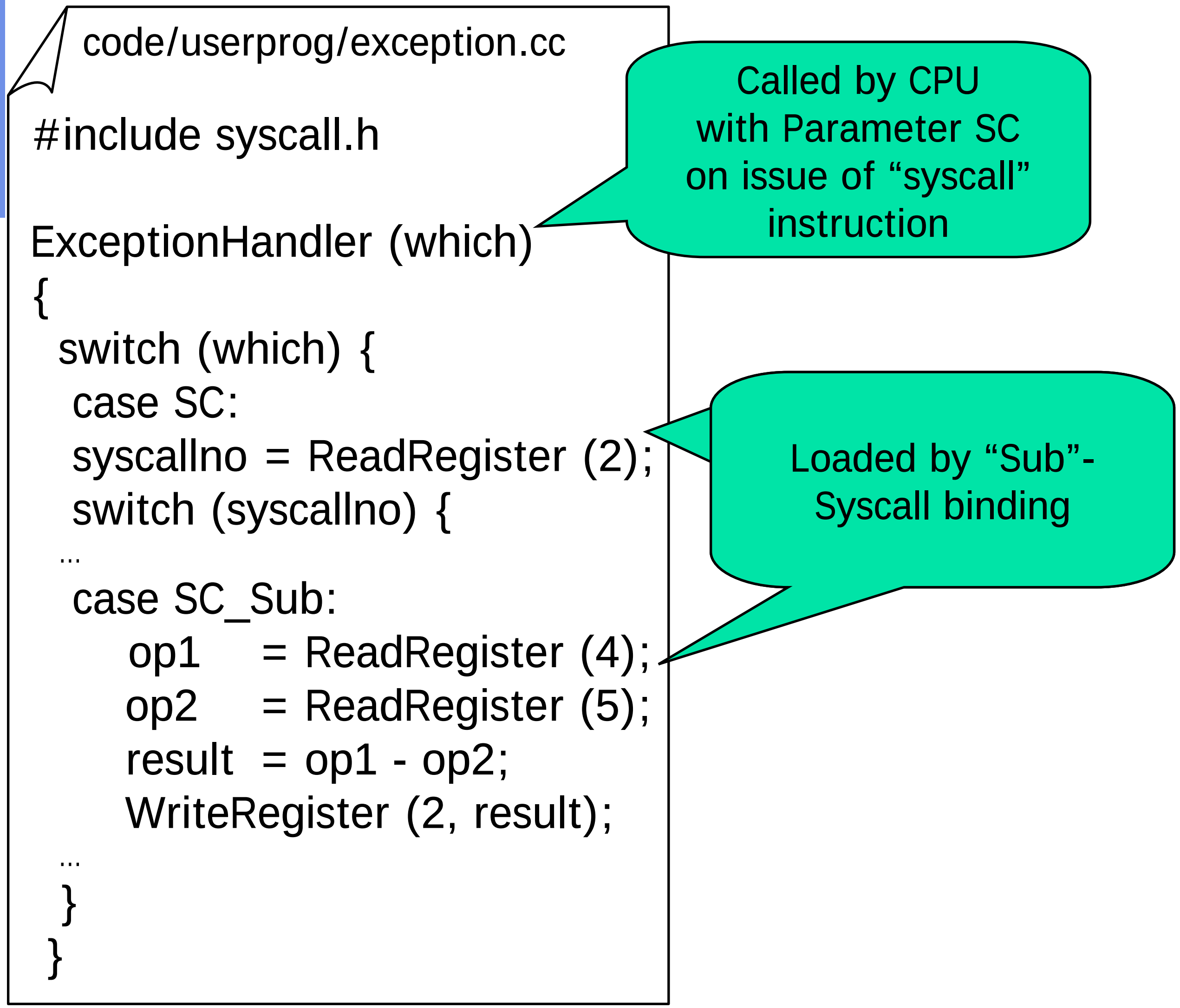
```
int Sub (int a, int b);

#define SC_Sub      43
```

code/test/start.s

```
#include syscall.h
      .globl Sub
      .ent   Sub
Sub:
      addiu $2,$0,SC_Sub
      syscall
      j       $31
      .end Sub
```

CPU generates Exception

Syscall – Kernel Implementation



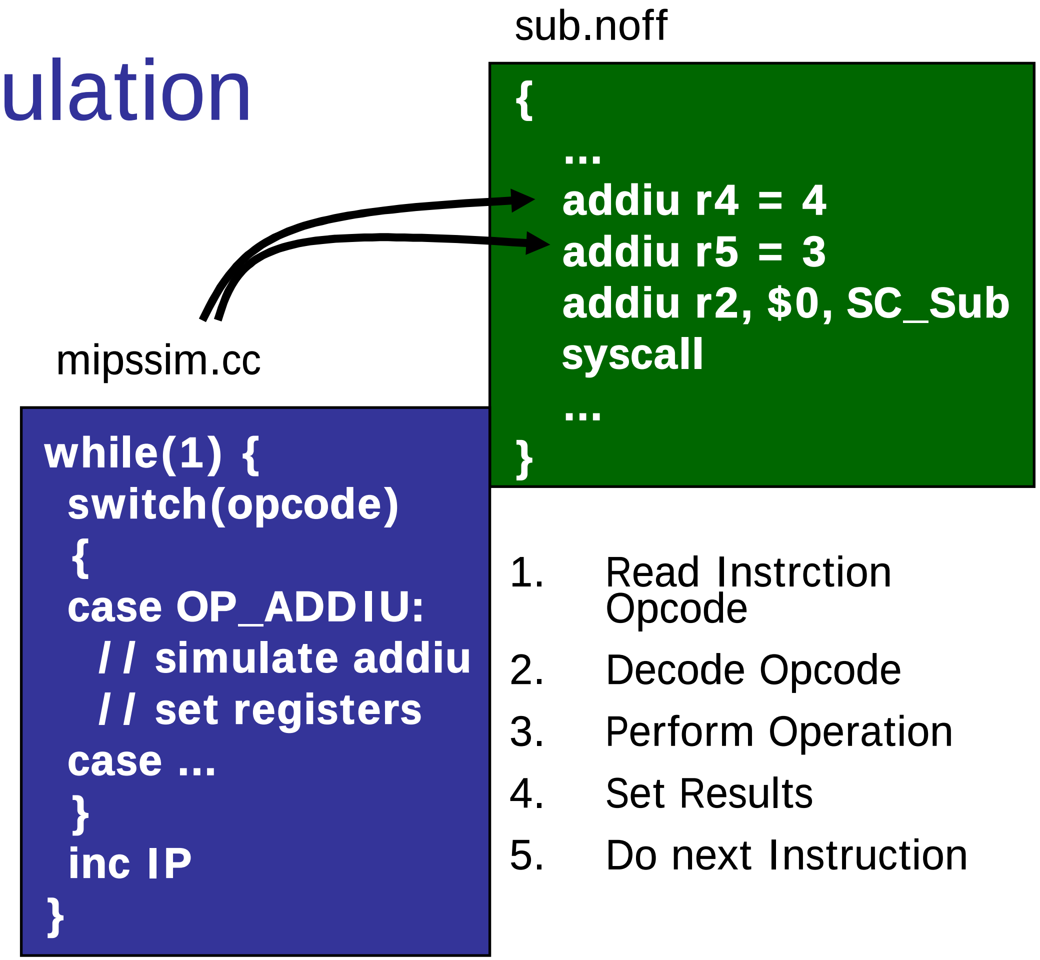
code/userprog/exception.cc

```
#include syscall.h

ExceptionHandler (which)
{
 switch (which) {
  case SC:
  syscallno = ReadRegister (2);
  switch (syscallno) {
 …
  case SC_Sub:
      op1    = ReadRegister (4);
      op2    = ReadRegister (5);
      result  = op1 - op2;
      WriteRegister (2, result);
 …
 }
}
```

Called by CPU with Parameter SC on issue of “syscall” instruction

Loaded by “Sub”-Syscall binding

NachOS CPU Emulation



sub.noff

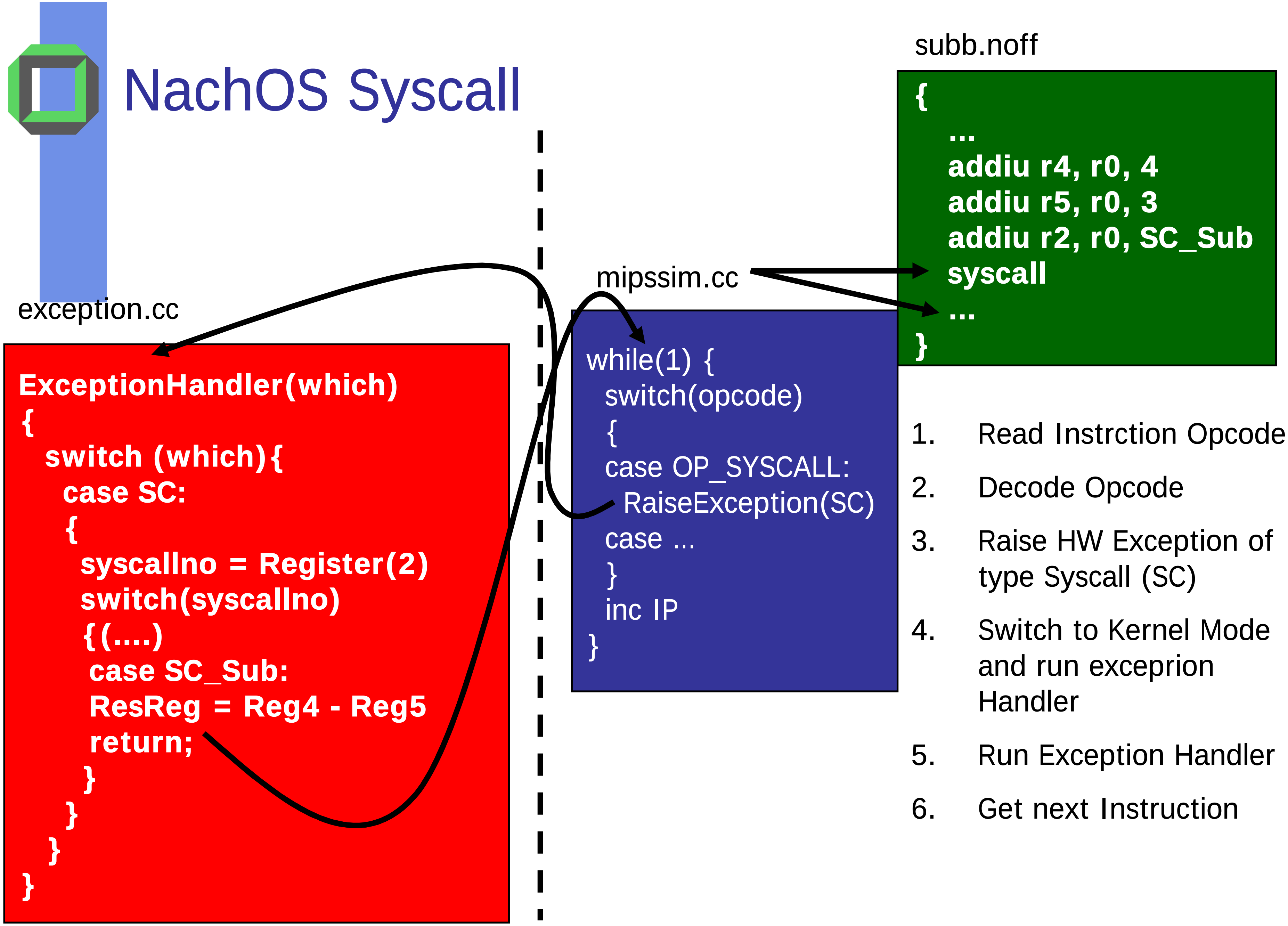
```
{
   ...
   addiu r4 = 4
   addiu r5 = 3
   addiu r2, $0, SC_Sub
   syscall
   ...
}
```

mipssim.cc

```
while(1) {
 switch(opcode)
 {
 case OP_ADDIU:
   // simulate addiu
   // set registers
 case ...
 }
 inc IP
}
```

1. Read Instrction Opcode
2. Decode Opcode
3. Perform Operation
4. Set Results
5. Do next Instruction

NachOS Syscall



subb.noff

```
{
   ...
   addiu r4, r0, 4
   addiu r5, r0, 3
   addiu r2, r0, SC_Sub
   syscall
   ...
}
```

mipssim.cc

```
while(1) {
  switch(opcode)
  {
  case OP_SYSCALL:
   RaiseException(SC)
  case ...
  }
  inc IP
}
```

exception.cc

```
ExceptionHandler(which)
{
  switch (which){
    case SC:
    {
      syscallno = Register(2)
      switch(syscallno)
      {(....)
       case SC_Sub:
       ResReg = Reg4 - Reg5
       return;
      }
    }
  }
}
```

1. Read Instrction Opcode
2. Decode Opcode
3. Raise HW Exception of type Syscall (SC)
4. Switch to Kernel Mode and run exceprion Handler
5. Run Exception Handler
6. Get next Instruction

Part 3

Useful things to know

Threads

- Each thread needs it own stack.
- Be aware of concurrency. Use the synchronization primitives provided in code/threads/synch.h.
- Threads in NachOS are kernel-level threads.
- Threads can only be forked on functions in the *kernel*.
- kernel->currentThread points to the thread currently running.

Address Spaces

- Currently addrspace.cc assumes to be alone in the whole system
- There are no Task structures in NachOS.

Programming

- Consider sanity checks.
- Using ASSERT() makes life a lot easier.
- Use the predefined error numbers.

Got a Problem but no Solution?

- Excellent Introduction to Syscalls on NachOS :
  - http://www-scf.usc.edu/~csci402/NachosP2NewDocumentation.htm

  (These are hints. Your not forced to do it exactly the same way.)
- Look at the NachOS Page
- Feel free to use the Forum

Organizational Issues (1)

- Register your Group!
- Max. 3 people per Group
- For Questions use the Forum!
- Deadline in Semester Holidays

Organizational Issues (2)

Code Review mandatory for each Group.

We Expect:

- Some working Code
- Deep Understanding of what you have done (Everyone).
- Presence of all members

Final Hints

Work as team, even if it's hard.

It pays off (at least on the global view).

- Balance Work
- Work Cooperatively

Think, Discuss, and Design a lot

before you start to code.

So you only have to implement it once... ;-)

Happy Hacking

"Thank, You.. We've been great."