CSE 421/521 - Operating Systems
Fall 2013

Lecture - XXIII

Project Discussion - II

Tevfik Koşar

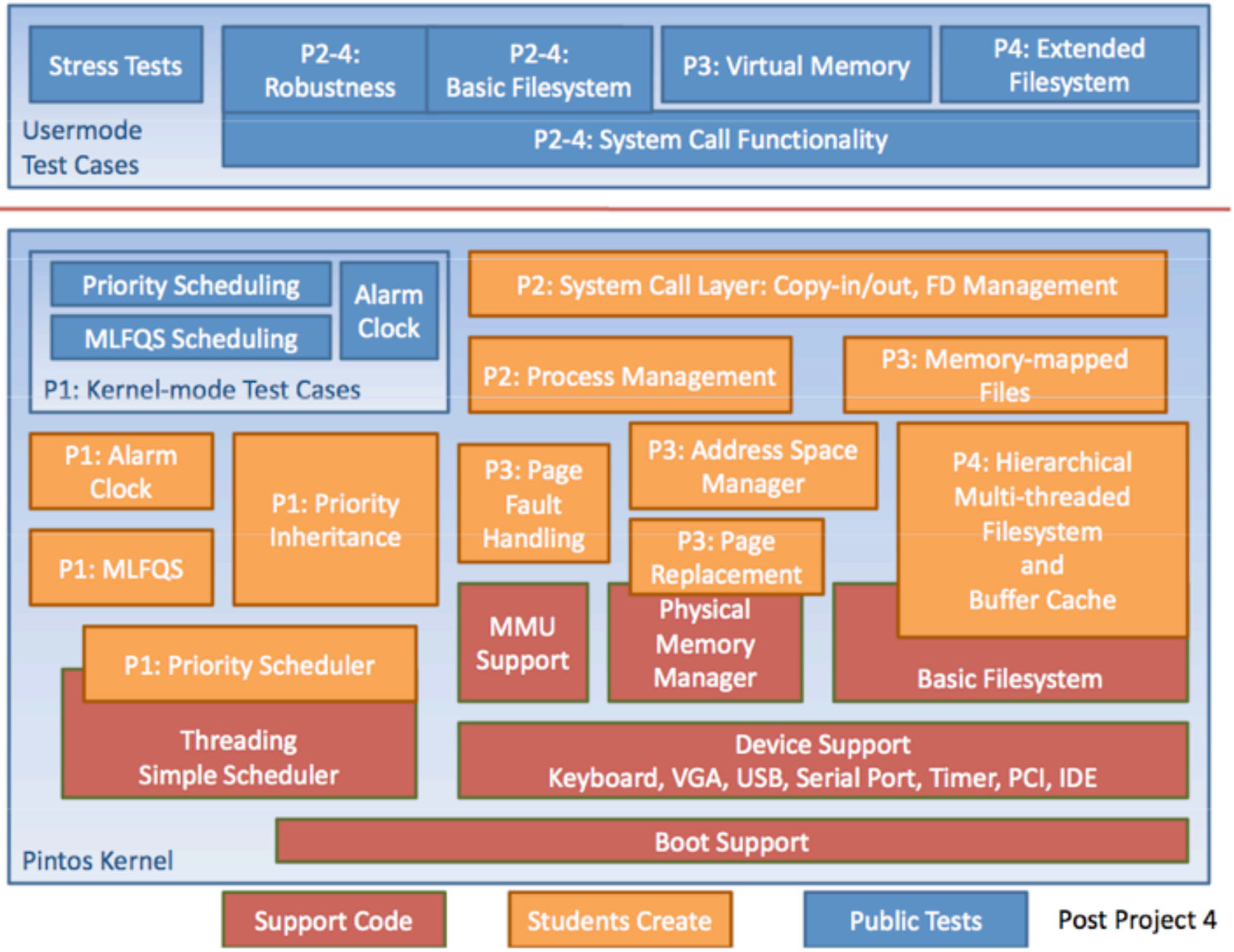
University at Buffalo

November 21st, 2013

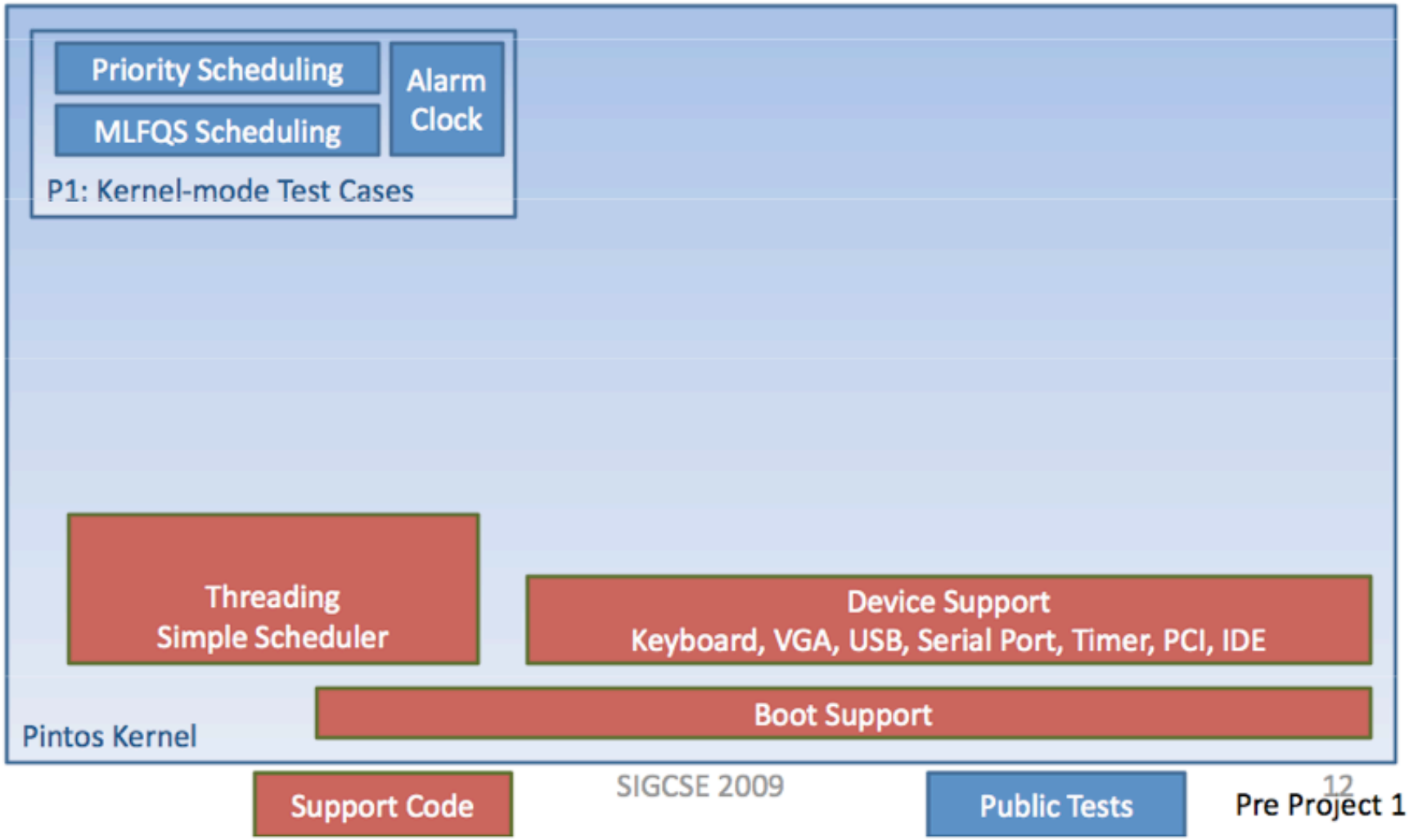
Pintos Projects

1. Threads
2. User Programs <-- Need also this (provided by us!)
3. Virtual Memory <-- CSE 421/521 Project 2
4. File Systems

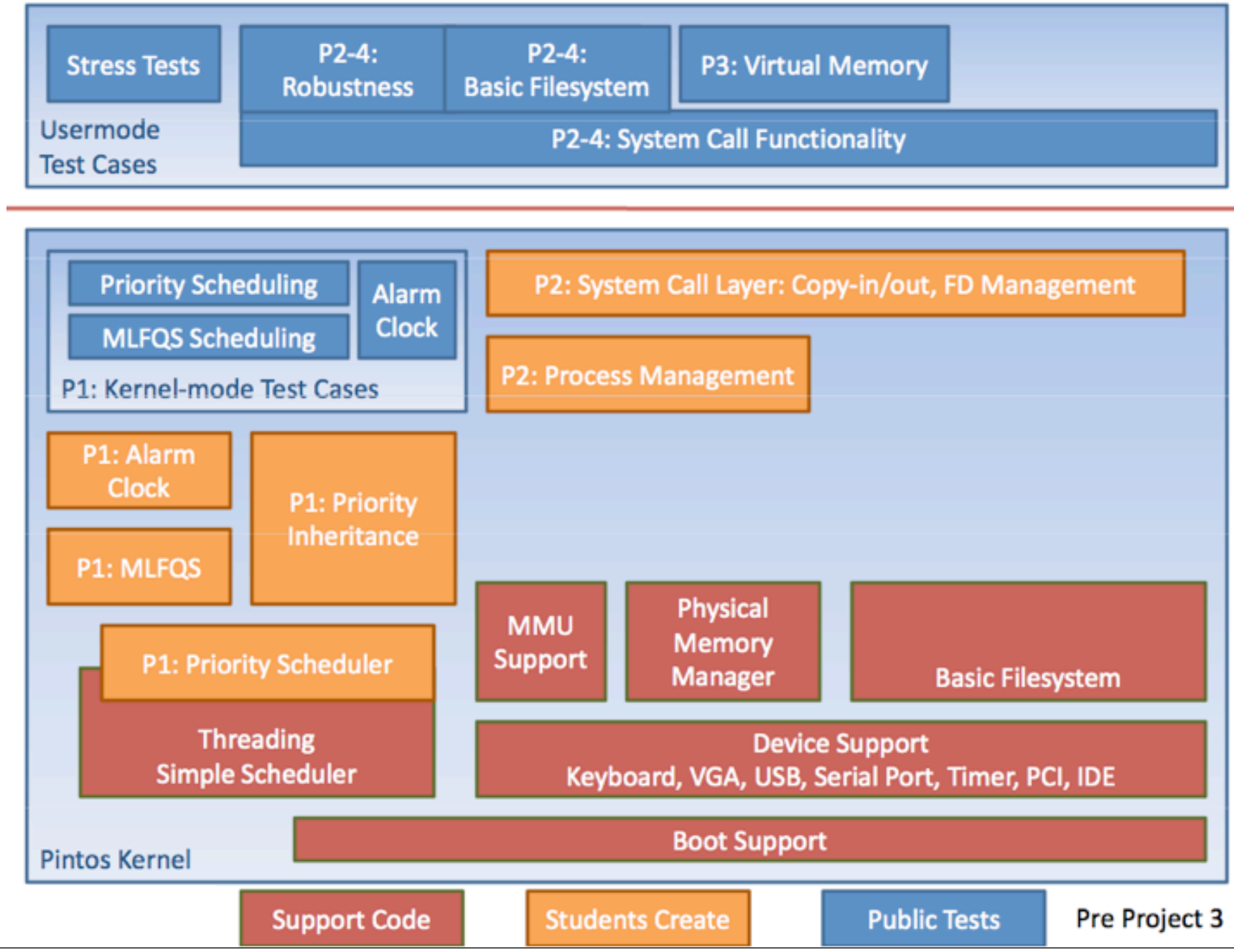
Pintos after full implementation (post prj-4)



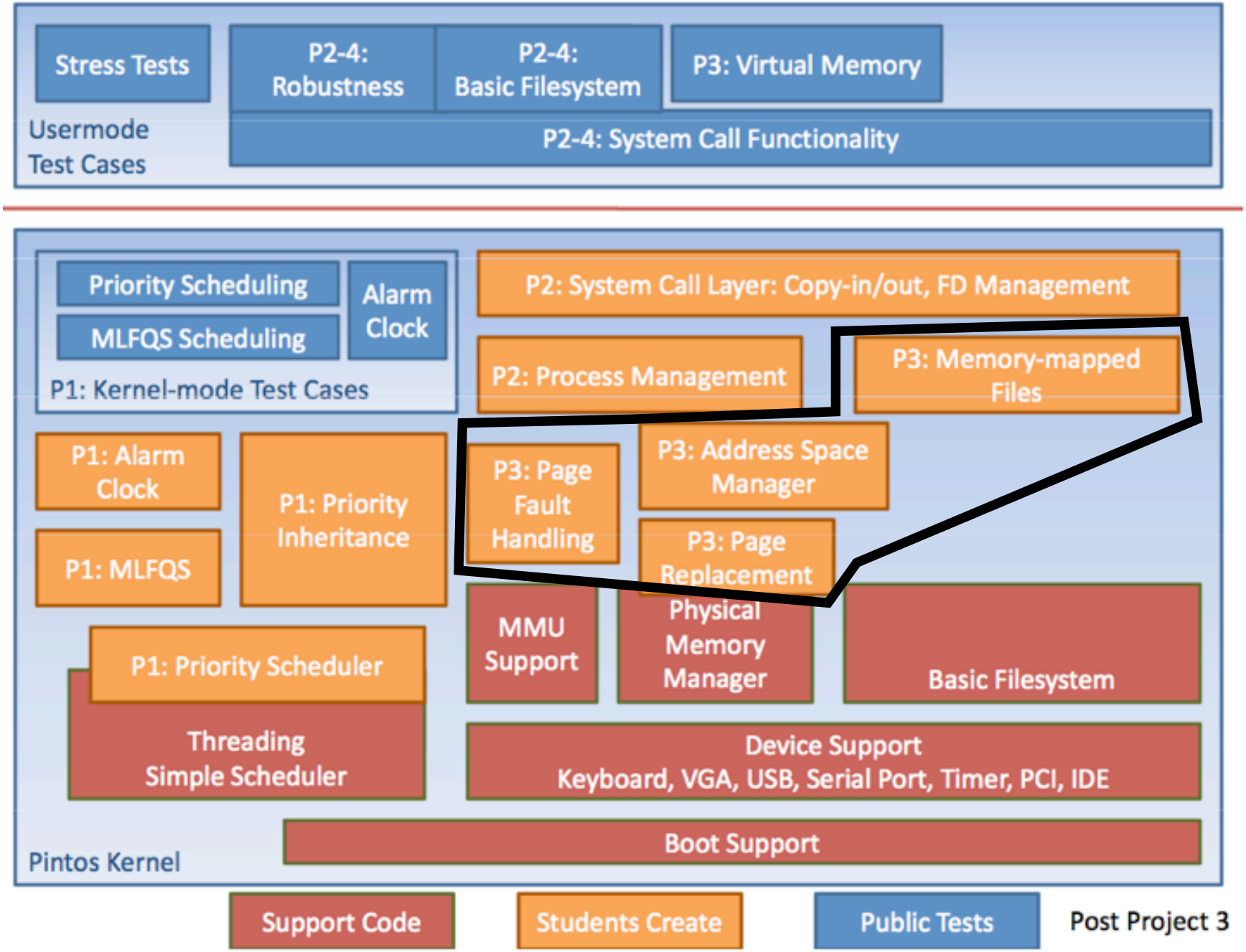
Pintos before any implementation (pre prj-1)



Yo will be provided with this (pre prj-3)

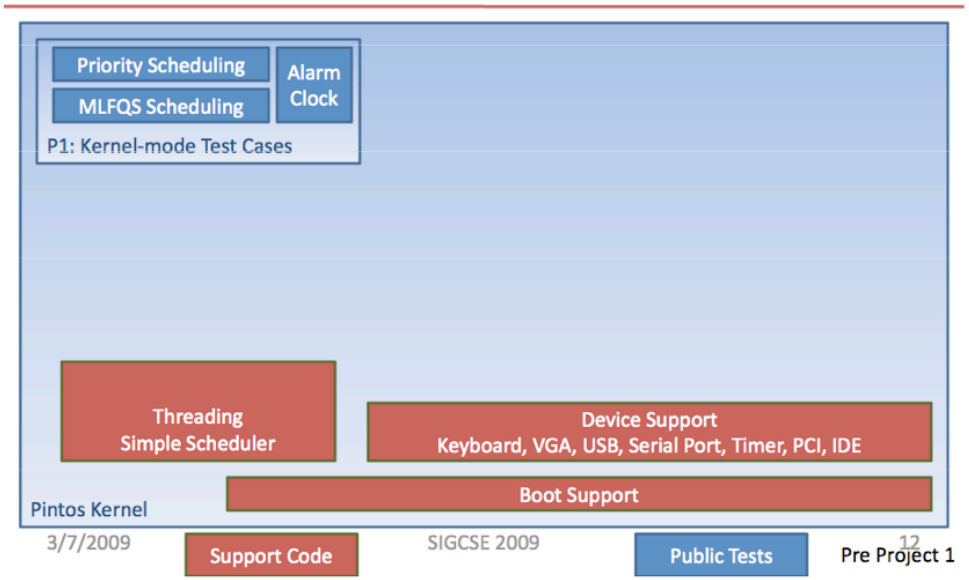


You will implement this (post prj-3)



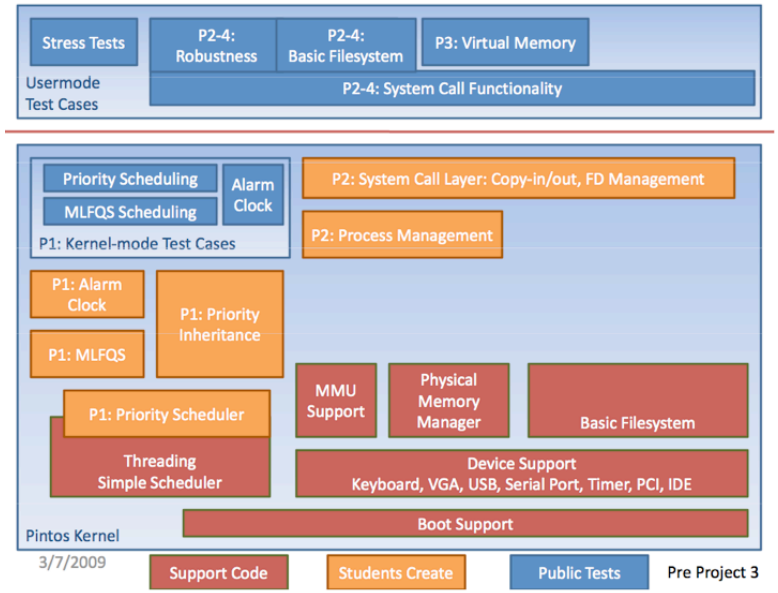
Initial src code provided to you

- Basic OS, with basic thread scheduler, no process management, no memory management, no file system etc.
- runs only 7 tests from “threads” successfully
- fails remaining (20) thread tests, as well as all userprog, vm, and filesys tests



New (patched) src code provided to you

- Comes with full “user program” support, system calls, basic file system, and basic memory management functionality
- pass all “userprog” tests, basic filesys tests
- would fail most vm tests --> this is what you need to implement



Current test results

FAIL tests/vm/pt-grow-stack
FAIL tests/vm/pt-grow-pusha
pass tests/vm/pt-grow-bad
FAIL tests/vm/pt-big-stk-obj
pass tests/vm/pt-bad-addr
pass tests/vm/pt-bad-read
pass tests/vm/pt-write-code
FAIL tests/vm/pt-write-code2
FAIL tests/vm/pt-grow-stk-sc
FAIL tests/vm/page-linear
FAIL tests/vm/page-parallel
FAIL tests/vm/page-merge-seq
FAIL tests/vm/page-merge-par
FAIL tests/vm/page-merge-stk
FAIL tests/vm/page-merge-mm
pass tests/vm/page-shuffle

FAIL tests/vm/mmap-read
FAIL tests/vm/mmap-close
pass tests/vm/mmap-unmap
FAIL tests/vm/mmap-overlap
FAIL tests/vm/mmap-twice
FAIL tests/vm/mmap-write
FAIL tests/vm/mmap-exit
FAIL tests/vm/mmap-shuffle
pass tests/vm/mmap-bad-fd
FAIL tests/vm/mmap-clean
FAIL tests/vm/mmap-inherit
FAIL tests/vm/mmap-misalign
FAIL tests/vm/mmap-null
FAIL tests/vm/mmap-over-code
FAIL tests/vm/mmap-over-data
FAIL tests/vm/mmap-over-stk
FAIL tests/vm/mmap-remove
pass tests/vm/mmap-zero

- You need to make all of these tests pass!

Important

- Keep both versions of the code
- It is important for you to understand how different functionality is implemented in Pintos, and how the code evolves

How to access new (patched) code?

```
$ cd ${PINTOSDIR}
$ cp /web/faculty/tkosar/cse421-521/projects/project-2/pintos-patched.tar .
$ mv src src-unpatched
$ tar -xvf pintos-patched.tar
```

Where to start?

- We provide you with three additional source files under src/vm:
  - frame.c
  - page.c
  - swap.c
  - and the corresponding header files:
    - frame.h
    - page.h
    - swap.h

  - any additional source files would need to be defined in **"Makefile.build"**

Order of Implementation

- Design and implement your Frame table
  - (change `process.c' to use your frame table allocator)
- Design and implement your Page table
  - (supplemental page table)
- Implement a page fault handler
  - (change `process.c' to record the necessary information in the supplemental page table when loading an executable and setting up its stack.)
- Implement eviction (page replacement)
- Implement stack growth, mapped files

Adding new files to the code

To add a `.c' file, edit the top-level `**Makefile.build'**. Add the new file to variable `dir_SRC', where dir is the directory where you added the file.  A new `.h' file does not require editing the `Makefile's.

Loading and Running User Programs

Example:

```
pintos-mkdisk filesys.dsk --filesys-size=2
pintos -f -q
pintos -p ../../examples/echo -a echo -- -q
pintos -q run 'echo x'

pintos -q ls
pintos -q rm file
```

How do I compile new user programs?
Modify `src/examples/Makefile', then run make.

Testing

To completely test your submission, invoke $ **make check** from the project `build' directory. This will build and run each test and print a \pass" or \fail" message for each one. When a test fails, make check also prints some details of the reason for failure.

Make check will select the faster simulator by default, but you can override its choice by specifying `SIMULATOR=--bochs'.

i.e. $ **make check SIMULATOR=--bochs**

Testing (cont)

To run and grade a single test, make the `.result' file explicitly from the `build' directory, e.g. $ **make tests/threads/alarm-multiple.result**. If make says that the test result is up-to-date, but you want to re-run it anyway, either run make clean or delete the `.output' file by hand.

By default, each test provides feedback only at completion, not during its run. If you prefer, you can observe the progress of each test by specifying `VERBOSE=1' on the make command line, i.e.

$ **make tests/threads/alarm-multiple.result VERBOSE=1**