

CSE 40677 Spring 2015  
Open Source Software Development

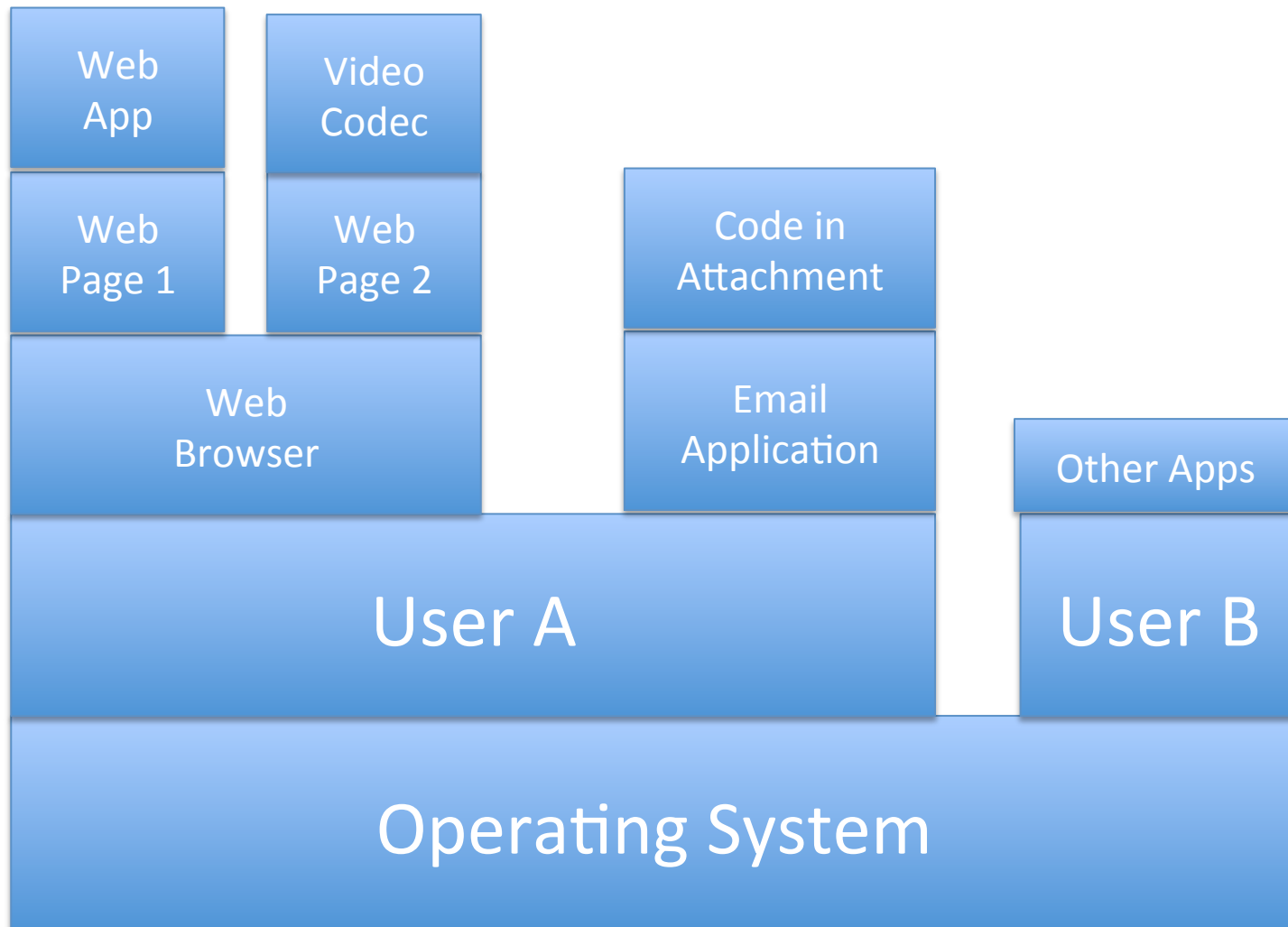
# Project Vision

Prof. Douglas Thain  
University of Notre Dame

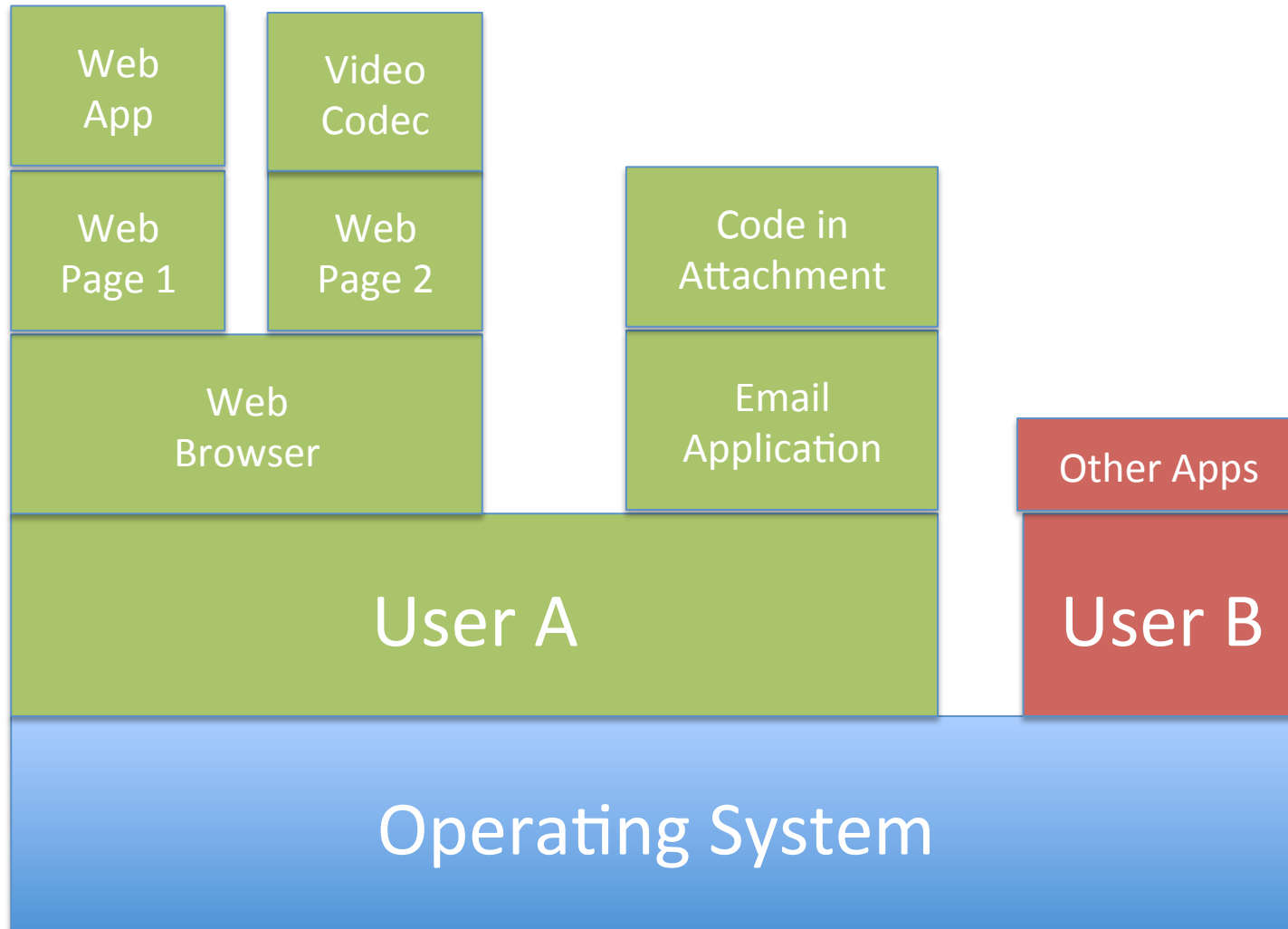
## Semester Goals:

- 1 - Build a prototype operating system in which hierarchical containment is the organizing principle.**
- 2 - Make it a sustainable open-source project so that others can easily contribute.

# Modern systems have many layers of mutually untrusting software.



# But the OS only separates users.



# The Traditional Unix Model:

- Only the super-user can create new protection domains (i.e. users) in order to separate untrusted software.
- Ordinary users cannot create new protection domains, and so they cannot protect themselves from untrusted software!

# This results in several big problems:

- Security:
  - An ordinary program has no simple way of protecting itself from a sub-program.
- Configuration:
  - Users cannot set up the software configuration that they want to use, without getting the sysadmin to “install” the software for everyone.
- Contamination:
  - Every software package expects to have access to every other software package simultaneously.

# Containment is **possible** today, but expensive and complicated:

- Mandatory Access Control (SELinux): Someone must configuration the interactions of everything on the system.
- Virtual Machines – Much too heavyweight to run individual components.
- Containers – Getting better, but still too heavyweight just to run a single webapp, video codec, or data analysis task.
- Runtimes like JVM/CLR/JS – provide containment, but only if you program in that language!

# What we want:

- Containment:
  - Every process is limited to a subset of resources (display, disk, memory etc) and cannot venture outside of those limits.
- **Hierarchical Containment:**
  - Every process is capable of starting sub-processes whose limits fall within that of the parent process.
- As a result:
  - Every process is effectively the superuser with respect to all of its sub-processes. No need for sudo/setuid!

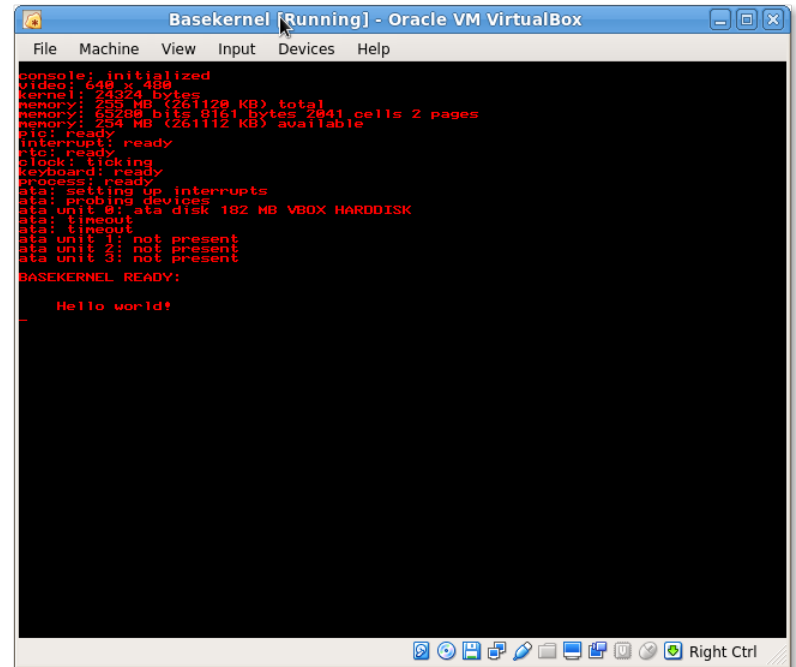


# A **prototype** operating system:

- A kernel that can manage the most essential resources (display, disk, memory) and run some example programs so as to clearly demonstrate hierarchical containment.
- Focus on the design of the basic kernel architecture and system call API to enable hierarchical containment.
- (It must have some working drivers, but that isn't the interesting point.)

# Starting Point: Basekernel

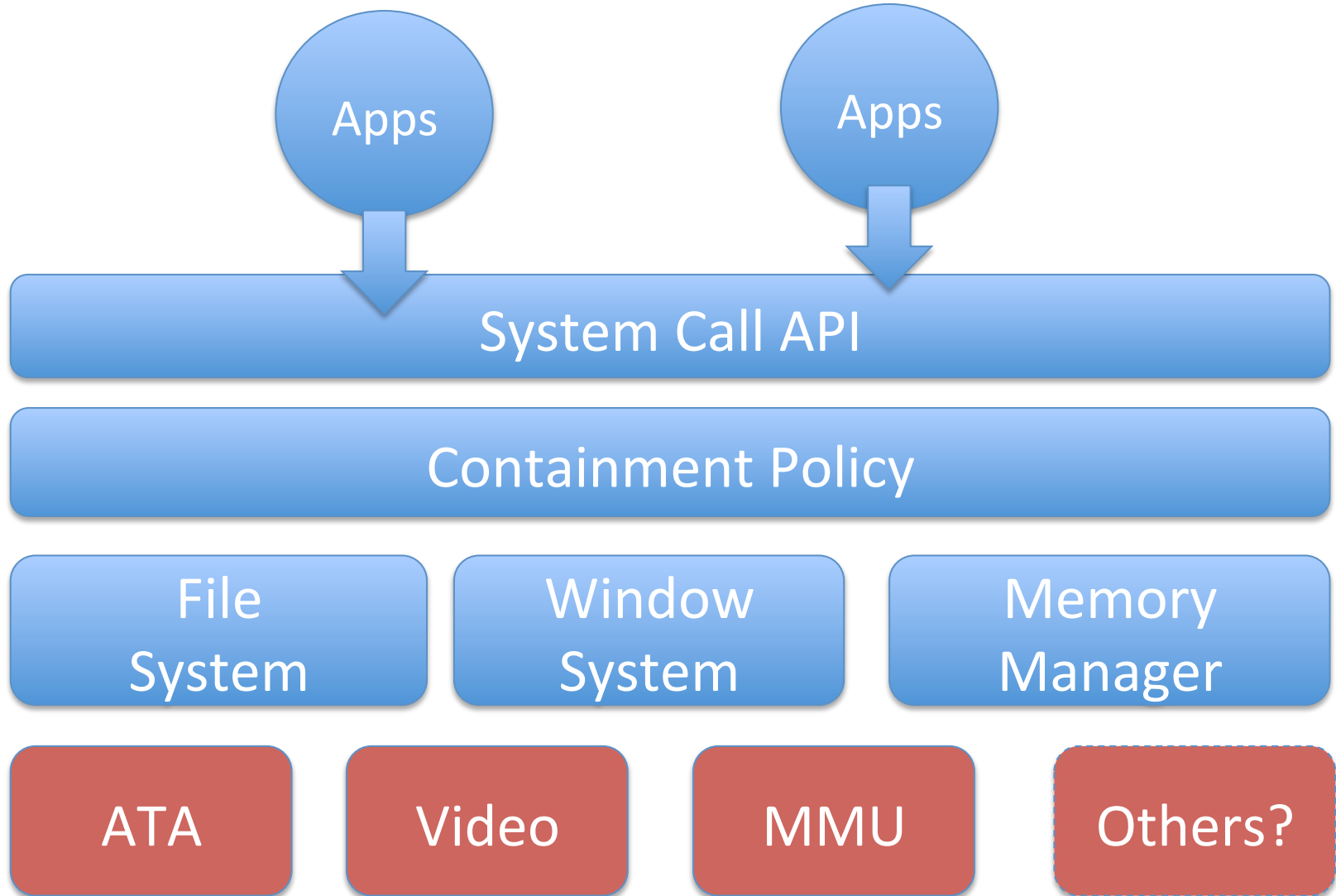
- A skeleton kernel that boots the system in 32 bit mode, and has minimal support for disks, memory, and video.



```
Basekernel [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
console: initialized
video: 640 x 480
kernel: 24324 bytes
memory: 234 MB (261120 KB) total
Memory: 65280 bits 8161 bytes 2041 cells 2 pages
memory: 234 MB (261112 KB) available
pic: ready
interrupt: ready
rtc: ready
clock: ticking
keyboard: ready
process: ready
ata: setting up interrupts
ata: probing devices
ata unit 0: ata disk 182 MB VBOX HARDISK
ata: timeout
ata unit 1: not present
ata unit 2: not present
ata unit 3: not present
BASEKERNEL READY:
Hello world!
```

<http://github.com/dthain/basekernel>

# Sketch of Complete System



## Semester Goals:

- 1 - Build a prototype operating system in which hierarchical containment is the organizing principle.
- 2 - Make it a sustainable open-source project so that others can easily contribute.**

# Project Requirements

- A public, high quality, and editable web presence which includes a compelling project vision, instructions for downloading and using the software, technical documentation and links regarding the details of the software, and a description of the membership and governance of the project.
- The project source code must be maintained in a public code repository, and changes accepted to the project through a standardized process. There must be a simple and well-documented process for building and using the source code.
- Public venues for reporting bugs, requesting help, and discussing project features. These may include issue trackers, forum software, or whatever is most appropriate to the project. Project development must be carried out using these tools.
- The final version of the software must meet the requirements of a Minimum Viable Product (MVP) which will be articulated by the team early in the semester.

# Meeting Structure

- Set up a schedule to work together on a regular basis each week, including some time to sync and prepare the weekly report.
- Present a weekly report to me with accomplishments of each member, overall status, and demo of the current code.
- Nothing is “real” until it is checked in and publically visible online!

# Getting Started

- Select open source tools for source code, web page, bug tracking, etc. (There is no perfect tool, so pick something and go forward...)
- Decide upon division of labor, at least to start. (It can change as you go forward.)
- Get the starter code, compile it, run it in a VM, and play around with it.
- Pick some *\*simple\** improvements that can be done quickly, to exercise your project process.

# Short Term Targets

- This Friday:
  - Team work schedule set up.
  - Infrastructure selected and initialized.
  - Demo of a few little hacks to the code.
- Next Friday:
  - Breakdown of big tasks into chunks.
  - Initial division of labor agreed on.
  - All infrastructure up with initial useful content.
  - First improvements written, debugged, accepted.