UMBC Guest Lecture

The Composite Component-Based Operating System

Gabriel Parmer
Computer Science Dept
The George Washington University

aka. **Gabe**

Discussion encouraged...

- Please stop me at any moment
- Let me know if you haven't yet learned something or don't know a term
- *Questions, questions, questions!*

Today

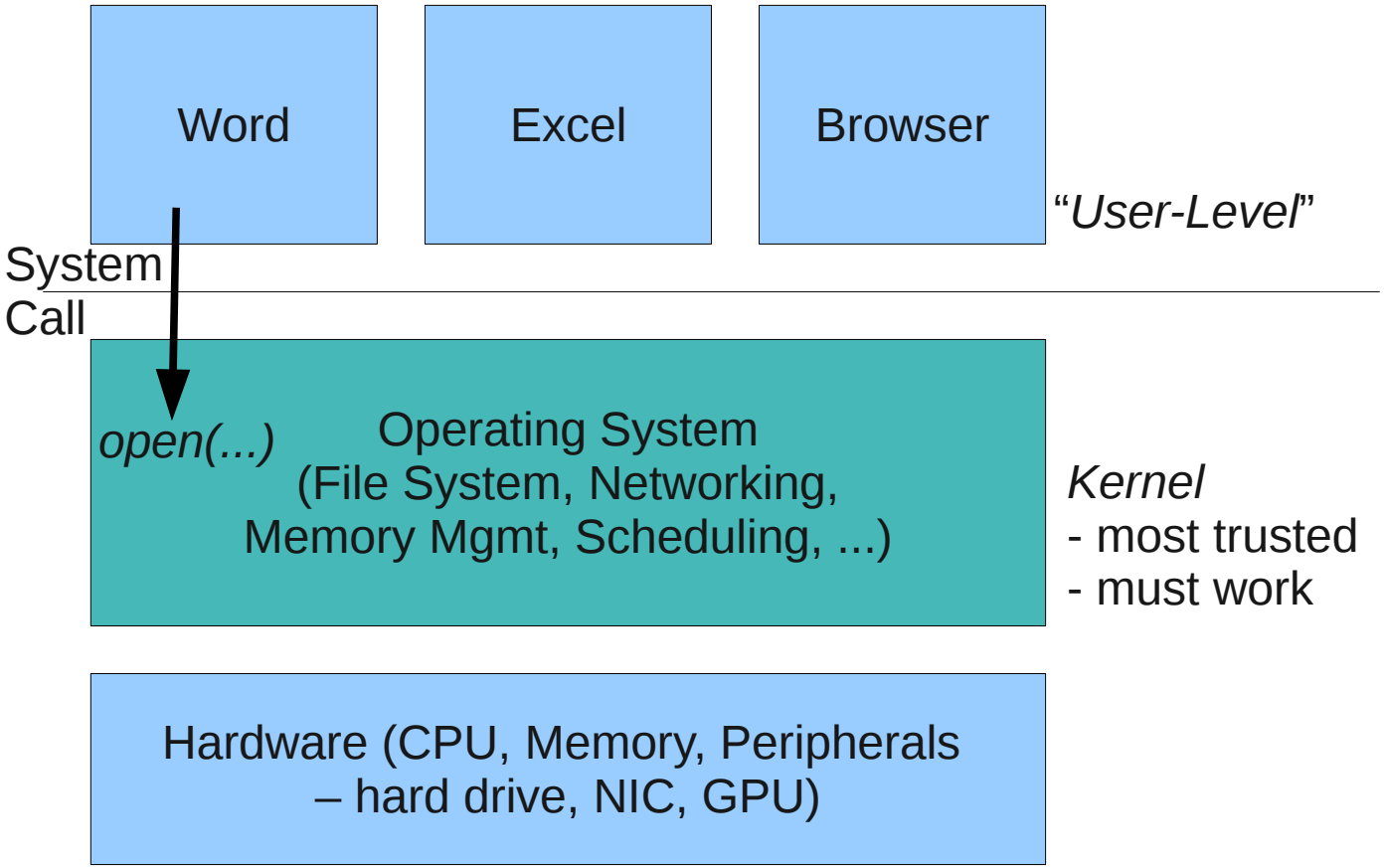
- What is a component-based operating system?
- A design study of one of the most important mechanisms

System Structure

- *System Structure:* How different software parts
    1) Are separated from each other (*Why?)*
    2) Communicate
- How does a system separate software using
    - dual mode
    - v*irtual address spaces*
- Implications on
    - Security/Reliability
- *What are some common system structures?*

Monolithic System Structure

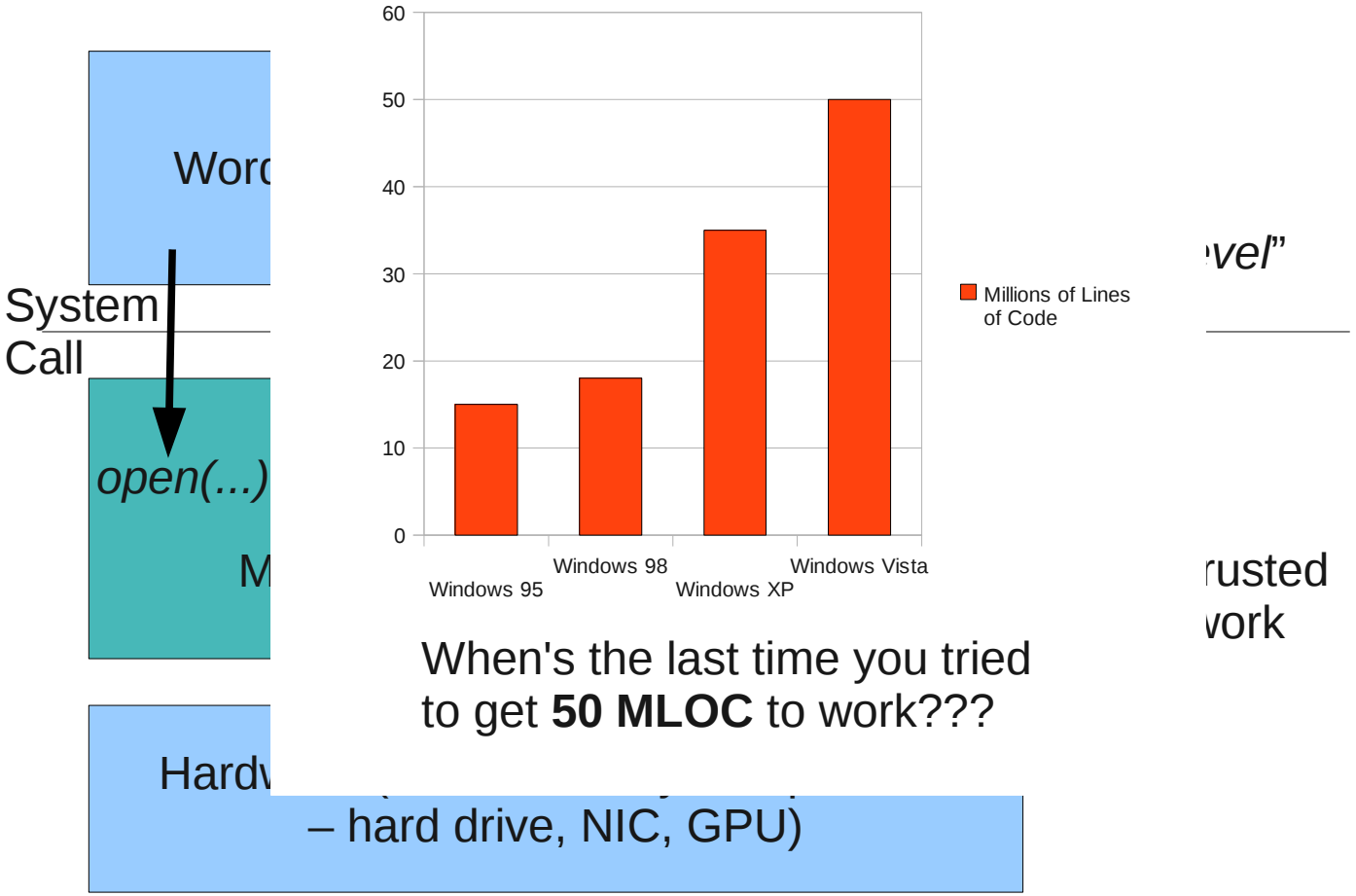
- Includes Unix/Windows/OSX



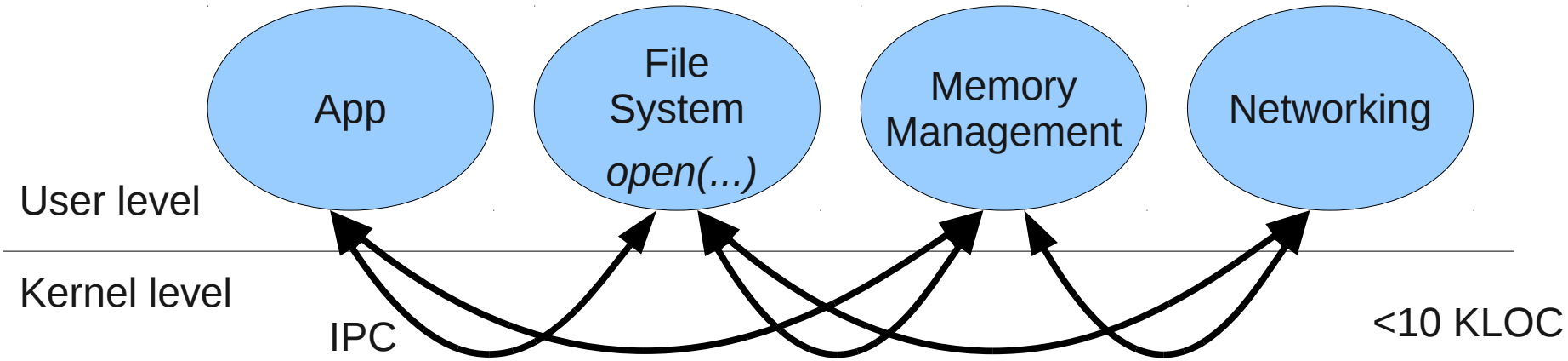
Dual mode protection? Virtual address spaces?

Monolithic System Structure

- Includes Unix/Windows/OSX



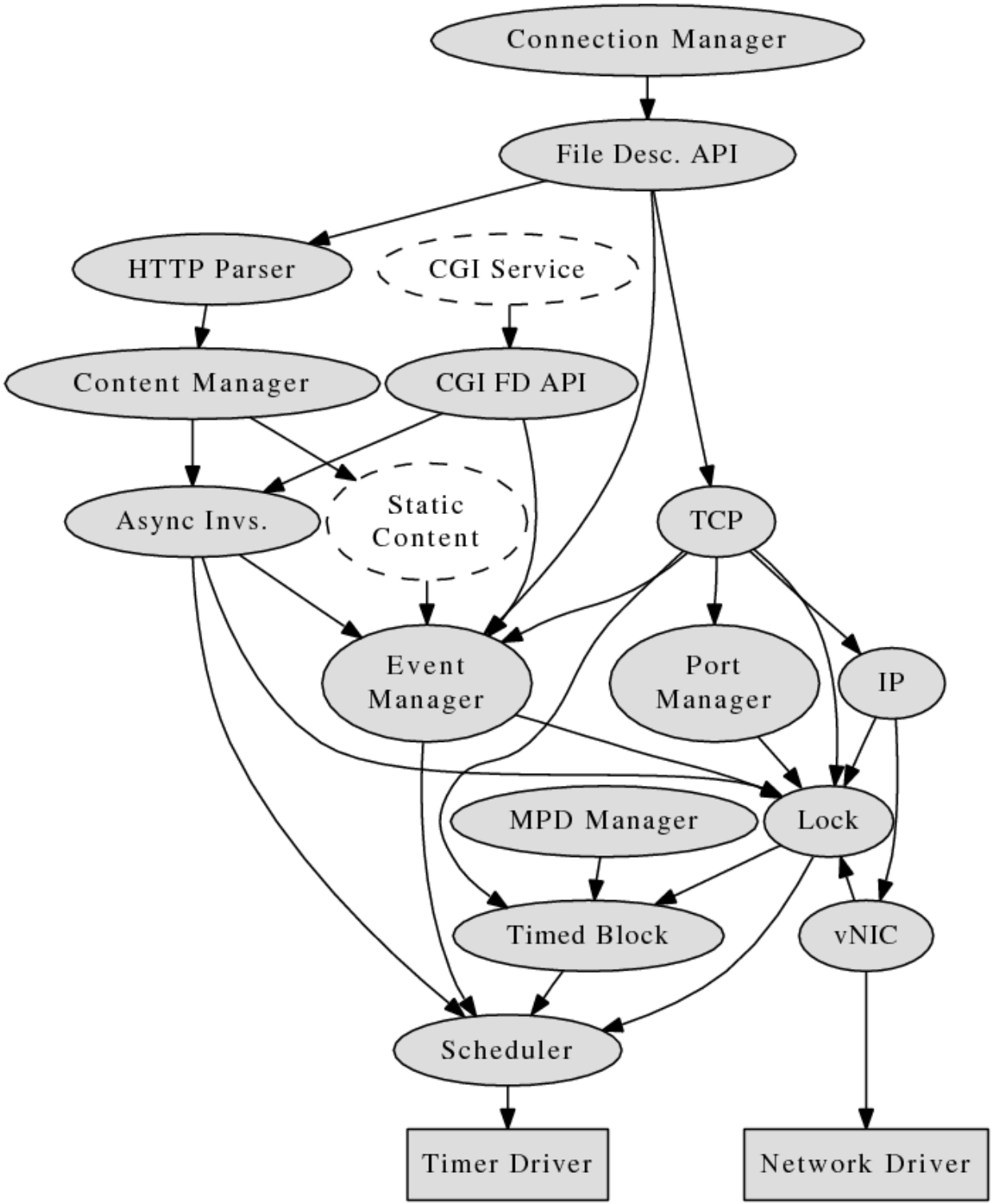
Microkernel System Structure



- Moves functionality from the kernel to "*user*" space
- Communication takes place between user *servers* using inter-process communication (IPC)
- Benefits:
  - Easier to add functionality
  - More reliable (*why?*)
  - More secure (*why?)*
- *Down-sides???*

Component-Based OS

- *Component*:
  - unit of functionality that exports an *interface*
  - uses other component's interfaces
  - User-level
  - separate virtual addr space
- *Interface:* Set of typed functions
- Even low-level functionality implemented in components
  - Scheduling, memory management, device drivers
- Kernel is minimal: not even scheduling!!!
- IPC for component communication


Connection Manager
File Desc. API
HTTP Parser
CGI Service
Content Manager
CGI FD API
Async Invs.
Static Content
TCP
Event Manager
Port Manager
IP
MPD Manager
Lock
Timed Block
vNIC
Scheduler
Timer Driver
Network Driver

vs. Microkernel?

- Microkernel:
  - Put subsystems at user-level
  - Networking, File system, etc...
  - *Focus*: Separate a normal system into servers
- Component-Based system
  - Break system into small chunks of functionality
  - Glue together specific components specific to the goals of the system: **customizability**
  - *Focus*: Break system into small functionalities

IPC Implementation

- High frequency of “inter-process communication”
  - “inter-component communication”
  - Must be fast!!!
- *What are the minimal hardware operations required to get a message from C_0 to C_1?*
  - *user/kernel and virtual addr space switches?*
- *How many thread switches?*
  - *Assuming separate threads per component*

IPC Implementation II

- Asynchronous communication: UNIX Pipes
- C_0: write(p1, buf0, sz); r = read(p2, buf0, sz)
  C_1: read(p1, buf1, sz); r1 = write(p2, buf1, sz)

- *Hardware operations?*
- *Thread switches?*

IPC Implementation III

- Synchronous IPC – like function calls!
  C_0: int foo(){return bar();} C_1: int bar(){return 1;}
- *Hardware operations?*
- *Thread switches? Assumptions?*

IPC Implementation IV

- Synchronous IPC between threads
  C_0: call(C_1,buf,sz)
  C_1: recv(C_0,buf,sz); reply_recv(C_0,buf,sz)
- *Hardware operations?*
- *Thread switches? Assumptions?*

IPC Implementation V

- *What is a thread?*
- Synchronous IPC – thread migration
  C_0: foo() {return bar();}
  C_1: bar() {return 1;}
- No thread switches – same “schedulable entity”
- *Hardware operations?*

Composite CBOS

- See http://www.seas.gwu.edu/~gparmer/projects/composite/
- Github repository for source code
  - We're accepting outside contributions!
  - TODO list in doc/ – smallish tasks

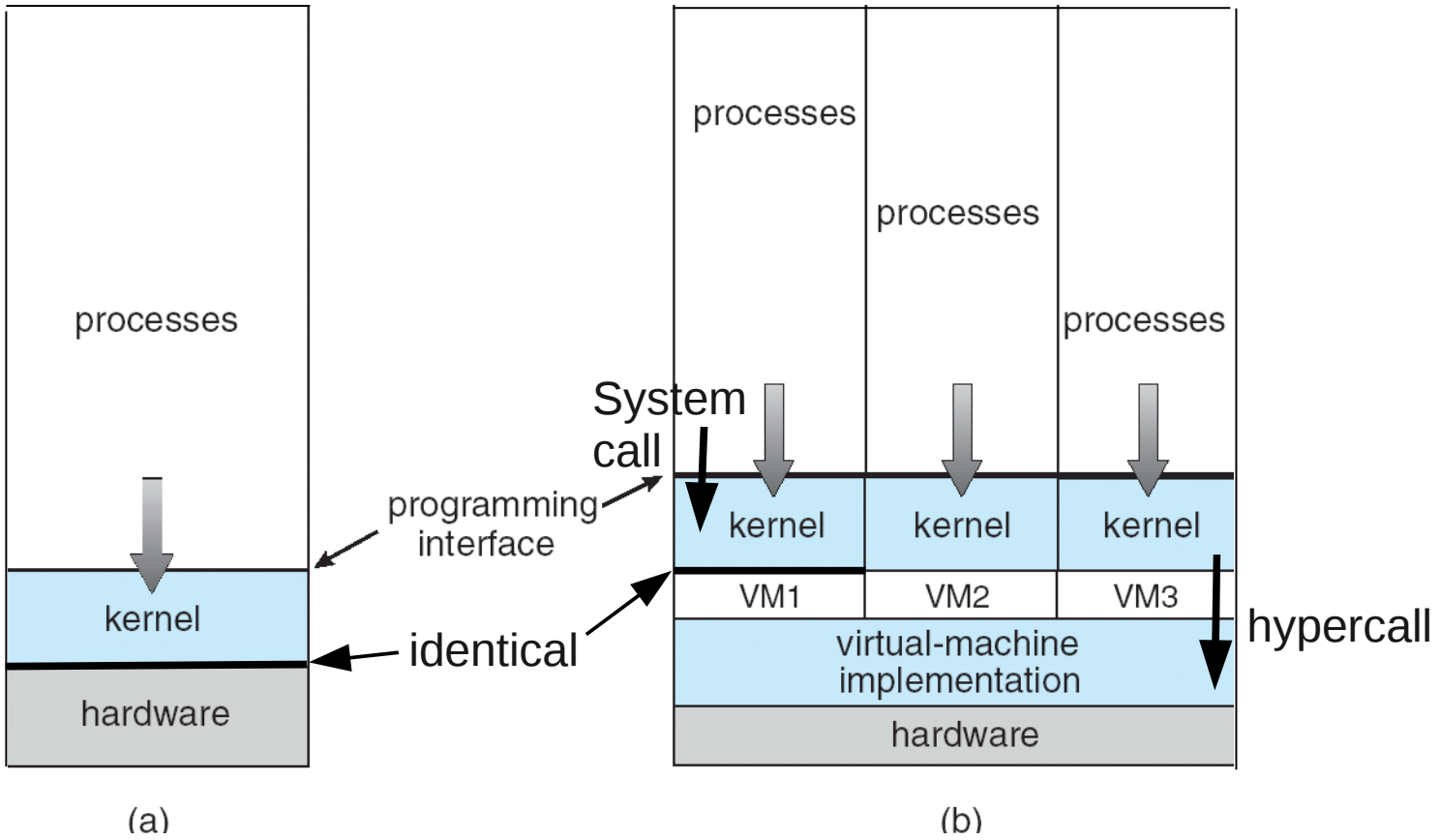
Virtual Machines I

- Do you know what these are?
- What is the structure of VMs?

Virtual Machines II

- A virtual machine *host* (the kernel) provides an interface *identical* to the underlying bare hardware
    - Other *guest* kernels execute in user-mode
    - The API for virtual machines is a copy of the machine!

Virtual Machines III



(a) non-virtual machine (b) virtual machine

Virtual Machine: Benefits

- Fundamentally, multiple operating systems share the same hardware
- Protected from each other
- Some sharing of files
- Communicate with each other via networking
- Useful for development, testing
- *Consolidation* of many low-resource use systems onto fewer busier systems