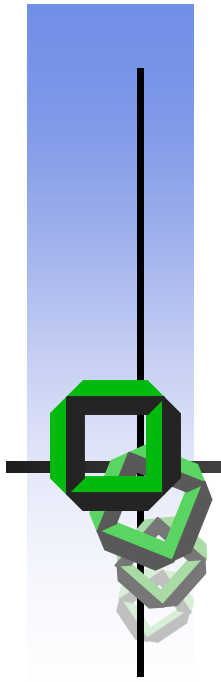


Systems Design and Implementation

I.1 – Introduction

System Architecture Group, SS 2009

University of Karlsruhe

20 April 2009

Jan Stoess

University of Karlsruhe

Tuesdays 17:30-19:00 SR-134, 50.41 (AVG)

Thursdays 15:45-17:15 SR-134, 50.41 (AVG)

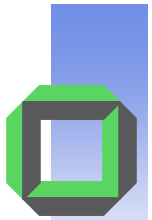
Goal

I hear and I forget.

I see and I remember.

I do and I understand.

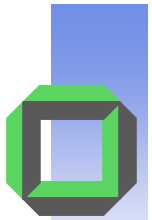
- Chinese Proverb



Goal

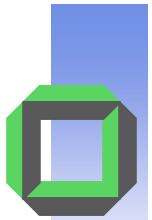
Provide students with a deeper understanding of operating systems through practical experience

Approach: Participate in the design and implementation of a simple operating system



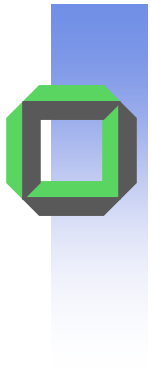
Aims

- Provide experience in OS design and development, including:
  - Microkernels
  - Multi-server systems
  - Alternative OS designs
  - Resource management
  - Device drivers, File systems, ...
- Demonstrate the importance of design
- Provide experience of being a team member in a software project



Aims

- Expose students to a mostly realistic OS development environment
  - Similar to professional OS or embedded systems developer
- Give an understanding of what's involved in constructing an entire OS
  - Understanding
  - Design
  - Implementation
- Encourage you to undertake a thesis, or do research within the System Architecture Group



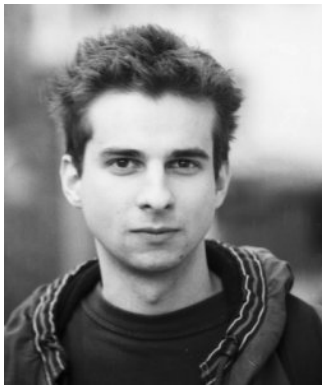
Prerequisites

- Students are expected to be competent programmers, with C (or C++) experience
- Students are expected to be familiar with
  - basic computer architecture concepts
  - basic system architecture concepts
- Familiarity with Intel x86 assembly language would be advantageous
- Familiarity with the “standard PC” architecture would also be advantageous

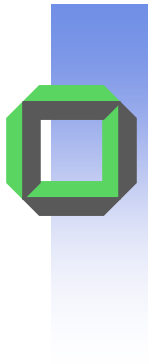
Lecturers and Tutors

- Lecturer
  - Jan Stoess
    stoess@ira.uka.de
  - Philipp Kupferschmied
    (fall-back)
    pkupfer@ira.uka.de

- Tutors
  - Marcel Noe

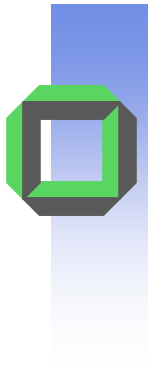


  - Consultation Time:
    Monday 4pm-6pm



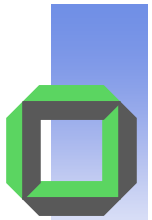
Administration – Diplom Students

- Lecture can count 2 hours towards an oral exam in VF System Architecture
- Lab course can count for a Praktikumsschein
- The following combinations are permitted
  - 2 hour lecture
  - 2 hour Praktikumschein
  - 2 hour lecture + 2 hour Praktikumschein
- Or just for fun



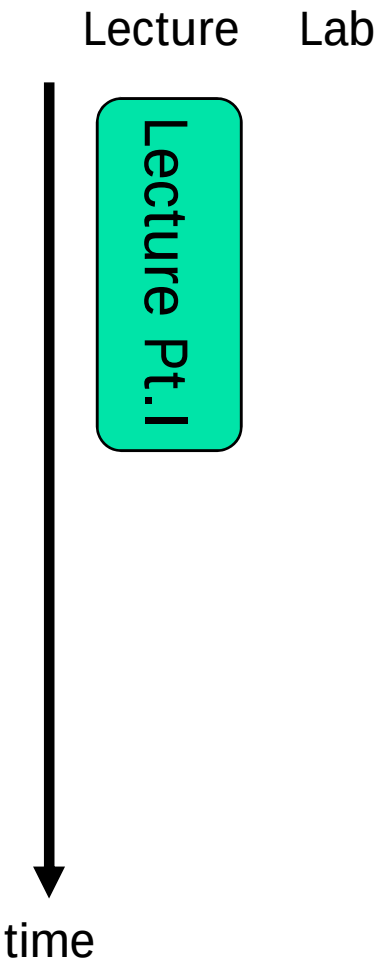
Administration – Master Students

- Modul “Systementwurf und Implementierung” - 3 LP
  - VF 4 Betriebssysteme
  - Lecture counts 3 LP
  - „Die Erfolgskontrolle erfolgt in Form einer mündlichen Prüfung im Umfang von i.d.R. 15 Minuten“
- Modul “Multi-Server Systeme” - 6 LP
  - VF 4 Betriebssysteme
  - Lecture counts 3 LP
  - Lab counts 3 LP
  - „Die Erfolgskontrolle erfolgt durch Beurteilung der Design-Beschreibung und den Programmquellen eines kleinen Entwicklungsprojektes sowie durch die Beurteilung der Präsentation des Ergebnisses als Erfolgskontrolle anderer Art“



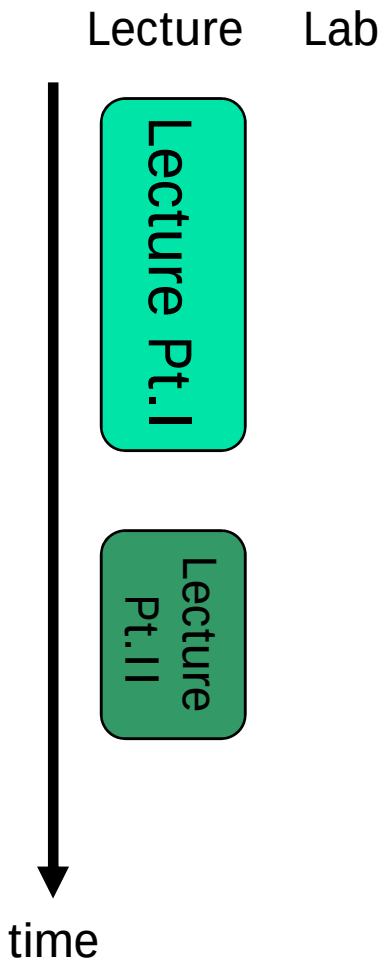
Course Structure

- Lecture Part I
  - Aim: teach (some) foundations of systems design
  - “Building blocks” of a modern operating system
    - Communication
    - OS Interfaces
    - Naming
    - Threads and Scheduling
    - Memory Management
    - File Systems
    - Device Drivers
  - Provides some theory
  - Presents case studies
    - Monolithic Systems
    - Multiserver Systems
    - Advanced Operating System Concepts
    - Virtualization



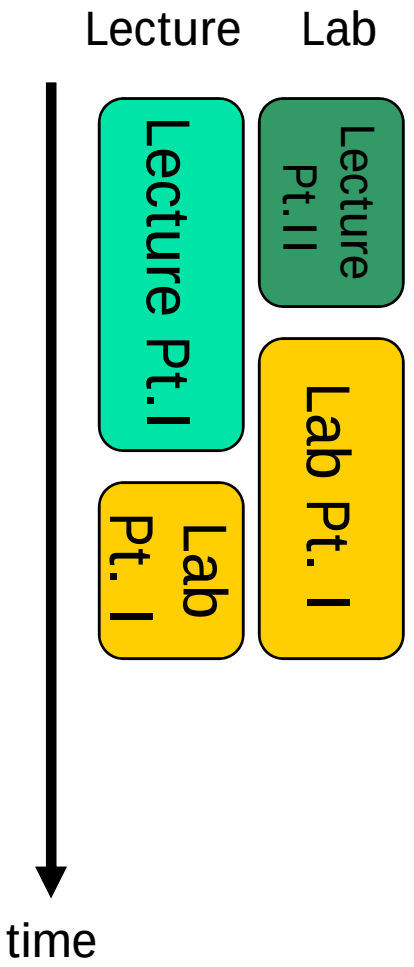
Course Structure

- Lecture Part II
  - Aim: teach foundations of systems design on L4
  - L4 API crash course
  - Basic concepts
  - System calls and their usage
  - Debugging facilities
  - IDL4 compiler for stub generation
- Problem:
  - Lab course depends on Lecture Pt. II
- Solution:
  - Lectures Part I & II will be given in parallel



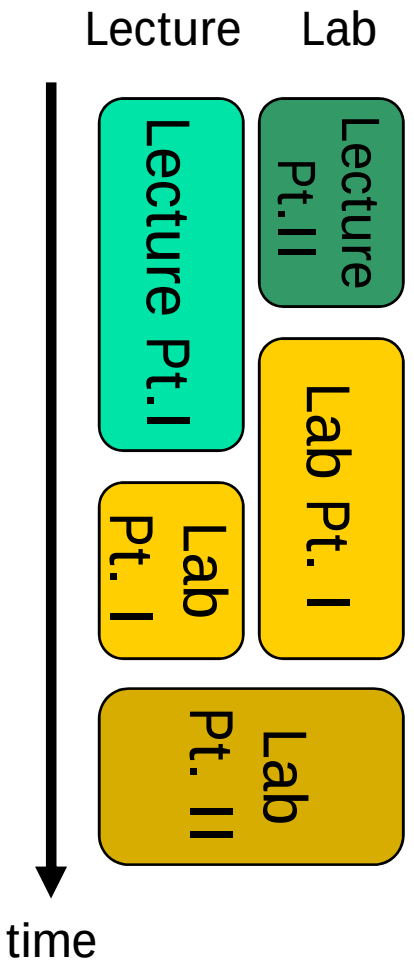
Course Structure

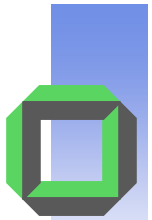
- Lab Part I
  - Aim: design an operating system
  1. Get together as SDI group
  2. Choose a *design topic*
  3. Consider lecture material
  4. Discuss particular design
  5. Present proposed designs



Course Structure

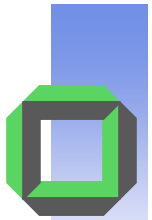
- Lab Part II
  - Aim: implement an operating system (at least partly)
  1. Get together as SDI group
  2. Consider presented designs
  3. Implement the components





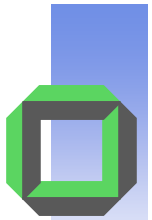
Preliminary Lecture Schedule
Lecture Part I

21.4.	Introduction
28.4.	Communication
5.5.	OS Interfaces
12.5.	Naming
19.5.	J. Stoess – Project Kittyhawk
26.5.	File Systems
2.6.	Threads, Scheduling
9.6.	Memory Management
16.6.	Drivers
23.6.	Device Service Design (2)
30.6.	Lab
7.7.	Lab
14.7.	Lab
21.7.	Lab
23.4.	L4 API Crash Course (I)
30.4.	L4 API Crash Course (II)
7.5.	IDL4, Debugging on L4
14.5.	Debugging on L4 (Lab)
21.5.	*- Christi Himmelfahrt -*
28.5.	Name Service Design (3)
4.6.	File Service Design (2)
11.6.	*- Fronleichnam -*
18.6.	Task Service Design (2)
25.6.	MM Service Design (2)
2.7.	Lab
9.7.	Lab
16.7.	Lab
23.7.	Lab Demos + Conclusion



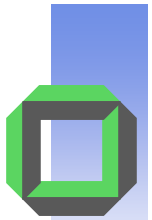
Preliminary Lecture Schedule
Lecture Part II

21.4.	Introduction	23.4.	L4 API Crash Course (I)
28.4.	Communication	30.4.	L4 API Crash Course (II)
5.5.	OS Interfaces	7.5.	IDL4, Debugging on L4
12.5.	Naming	14.5.	Debugging on L4 (Lab)
19.5.	J. Stoess – Project Kittyhawk	21.5.	*- Christi Himmelfahrt -*
26.5.	File Systems	28.5.	Name Service Design (3)
2.6.	Threads, Scheduling	4.6.	File Service Design (2)
9.6.	Memory Management	11.6.	*- Fronleichnam -*
16.6.	Drivers	18.6.	Task Service Design (2)
23.6.	Device Service Design (2)	25.6.	MM Service Design (2)
30.6.	Lab	2.7.	Lab
7.7.	Lab	9.7.	Lab
14.7.	Lab	16.7.	Lab
21.7.	Lab	23.7.	Lab Demos + Conclusion



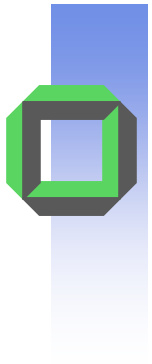
Preliminary Lecture Schedule
Lab Part I

21.4.	Introduction
28.4.	Communication
5.5.	OS Interfaces
12.5.	Naming
19.5.	J. Stoess – Project Kittyhawk
26.5.	File Systems
2.6.	Threads, Scheduling
9.6.	Memory Management
16.6.	Drivers
23.6.	Device Service Design (2)
30.6.	Lab
7.7.	Lab
14.7.	Lab
21.7.	Lab
23.4.	L4 API Crash Course (I)
30.4.	L4 API Crash Course (II)
7.5.	IDL4, Debugging on L4
14.5.	Debugging on L4 (Lab)
21.5.	*- Christi Himmelfahrt -*
28.5.	Name Service Design (3)
4.6.	File Service Design (2)
11.6.	*- Fronleichnam -*
18.6.	Task Service Design (2)
25.6.	MM Service Design (2)
2.7.	Lab
9.7.	Lab
16.7.	Lab
23.7.	Lab Demos + Conclusion



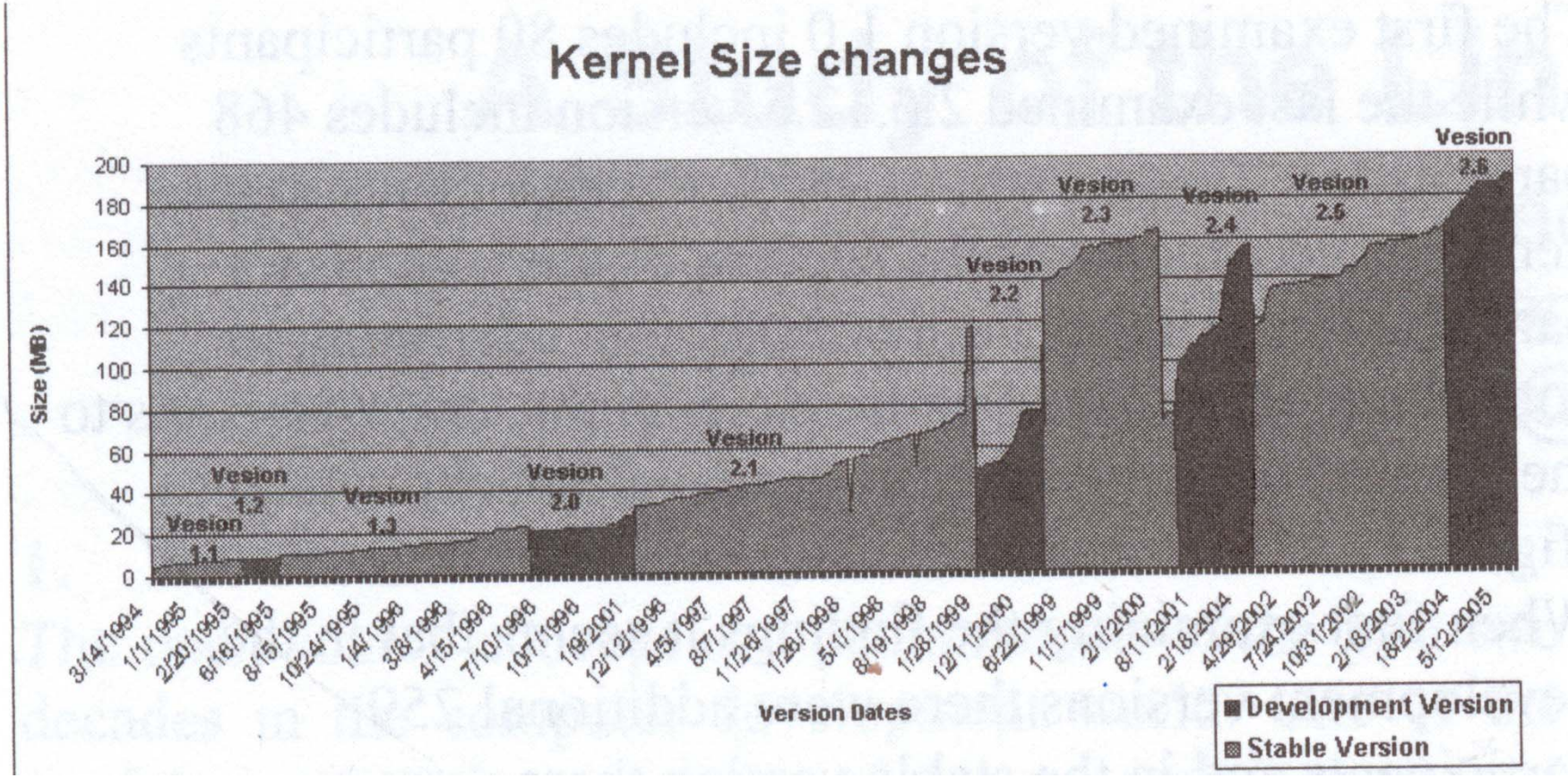
Preliminary Lecture Schedule
Lab Part II

21.4.	Introduction	23.4.	L4 API Crash Course (I)
28.4.	Communication	30.4.	L4 API Crash Course (II)
5.5.	OS Interfaces	7.5.	IDL4, Debugging on L4
12.5.	Naming	14.5.	Debugging on L4 (Lab)
19.5.	J. Stoess – Project Kittyhawk	21.5.	*- Christi Himmelfahrt -*
26.5.	File Systems	28.5.	Name Service Design (3)
2.6.	Threads, Scheduling	4.6.	File Service Design (2)
9.6.	Memory Management	11.6.	*- Fronleichnam -*
16.6.	Drivers	18.6.	Task Service Design (2)
23.6.	Device Service Design (2)	25.6.	MM Service Design (2)
30.6.	Lab	2.7.	Lab
7.7.	Lab	9.7.	Lab
14.7.	Lab	16.7.	Lab
21.7.	Lab	23.7.	Lab Demos + Conclusion



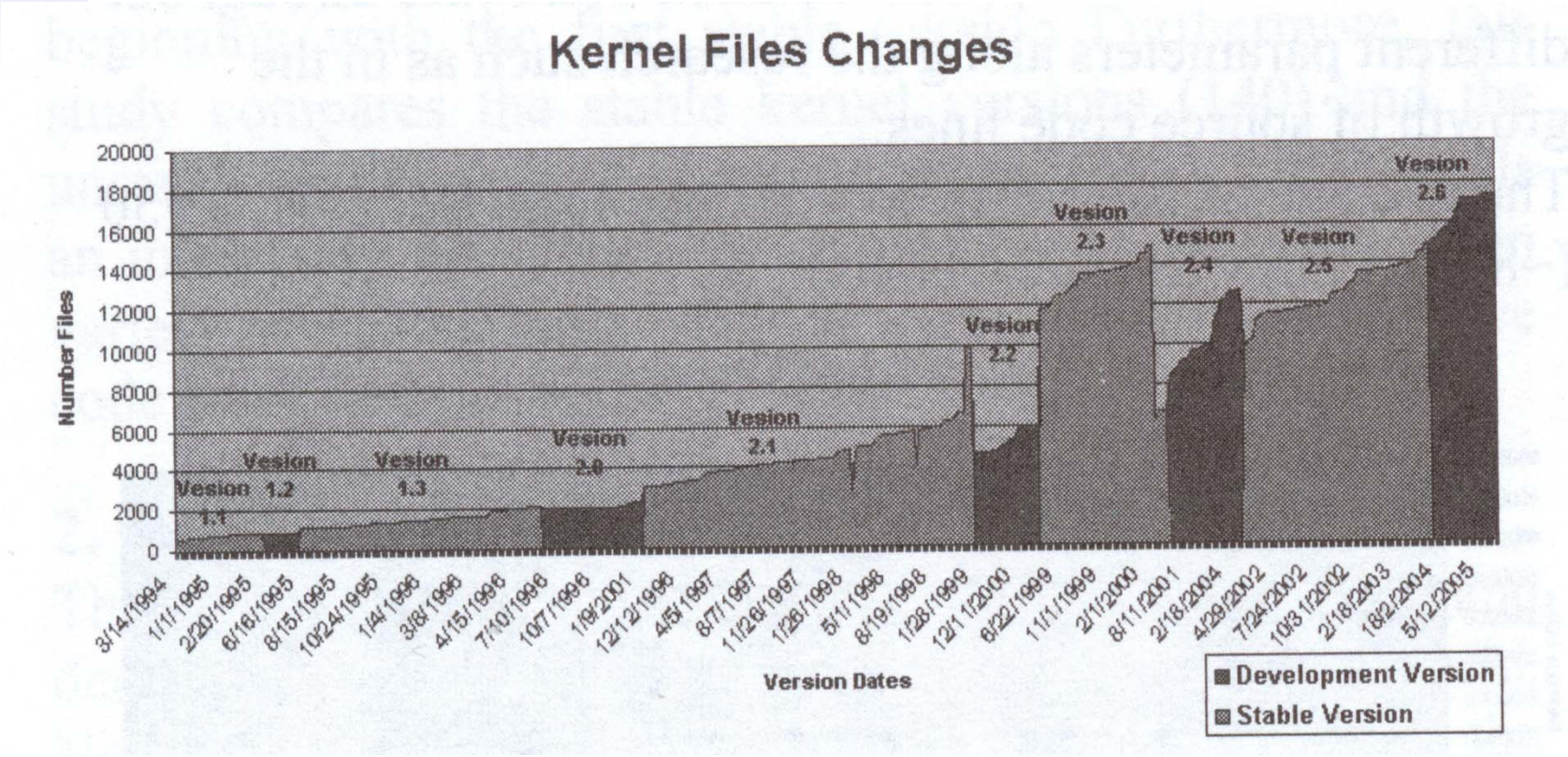
Introduction to the envisaged operating system…

Linux Kernel Evolution



Source: Oded Koren: **A study of the Linux Kernel.** ACM SIGOPS Operating System Review 40-6, 2006

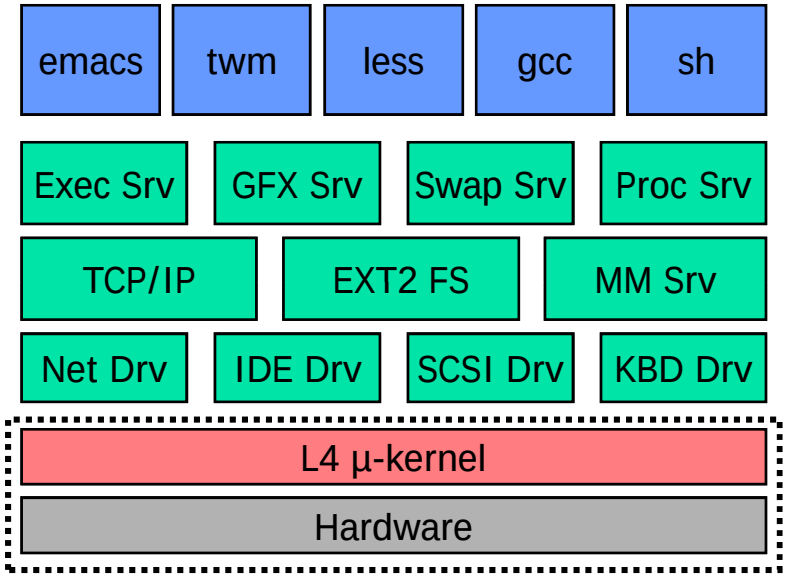
Linux Kernel Evolution



Source: Oded Koren: **A study of the Linux Kernel.** ACM SIGOPS Operating System Review 40-6, 2006

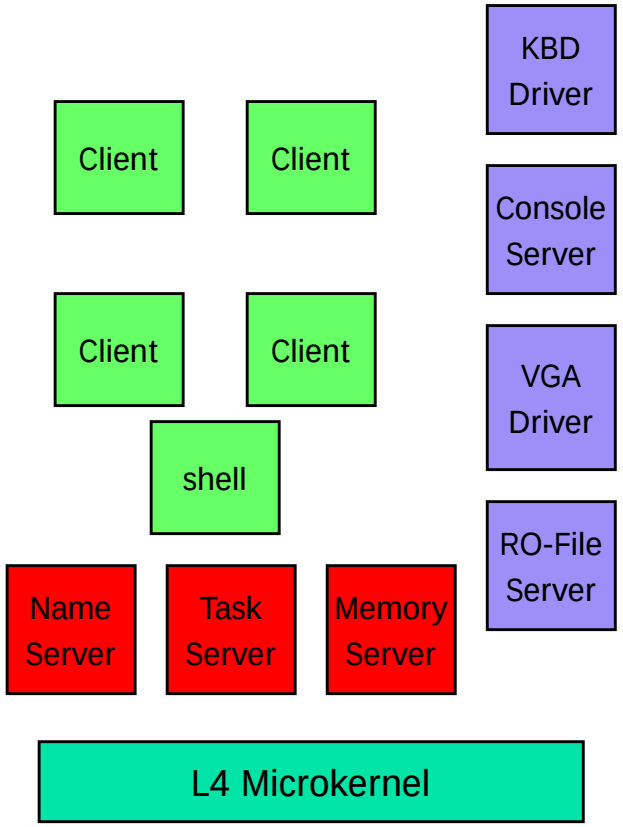
Multiserver Operating Systems

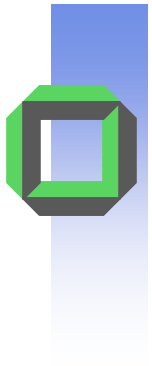
- A set of components running as servers on the microkernel
  - + Modularity
  - + Flexibility
  - + Robustness
  - + Security



Envisaged System

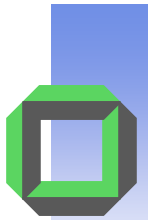
- Multi-server OS built on the L4 microkernel L4Ka::Pistachio
  - A “simple” base to build upon
  - Hides some of hardware complexity
  - Already implements some OS functionality
  - The mechanisms provided are flexible enough to still tackle OS issues at a low level





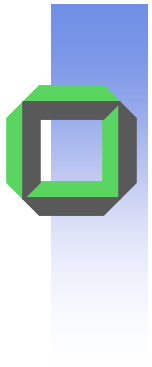
Is this too much work?

- A real OS is beyond 14 weeks work
  - We must limit the scope of the project to be achievable
- Single-user system
  - Limited protection, e.g.:
    - enforce address space boundaries
    - enforce read-only access to read-only files
    - no “user” identifiers, all tasks *potentially* have all access rights
  - No sophisticated security
    - No authentication, authorization etc.



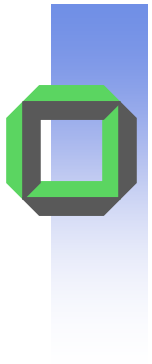
Is this too much work?

- Aim for designs that are
  - Thoroughly thought through
  - Not conceptually limiting
  - “Good”
- Aim for implementations with known limits to ease implementation
  - Implement only needed functionality (but trap unimplemented functionality)
  - General functionality can be limited
  - e.g.: Use limited static arrays, rather than general tree based structures



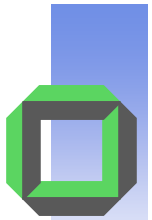
SDI Lecture

- Aims:
  - Teach foundations of systems design
  - Provide a broader view on how to construct different OS personalities
  - Teach a frame of reference for reasoning on OS design issues
- Method:
  - Different OS concepts presented in detail
  - Examples and case studies from existing OSes



Lesson to be learned

- Well designed components with poor implementations can be easily replaced or improved in isolation.
- Badly designed components with the best implementations
  - Still perform poorly
  - Require system redesign to improve them
  - May have to rewrite the complete system from scratch

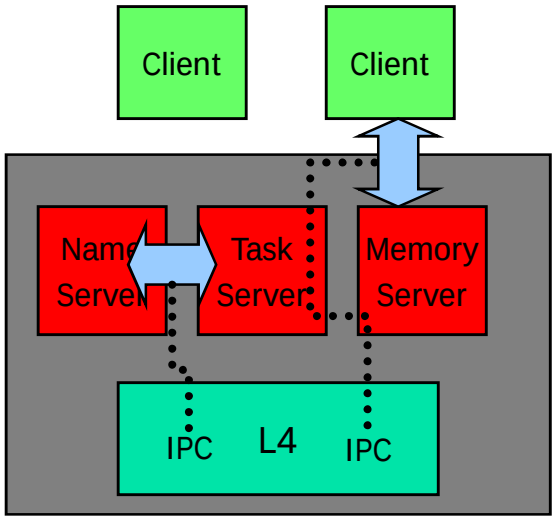


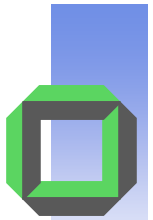
Issues to tackle: Communication

Lecture Part I

Lecture Pt.II

- Communication
  - Why communicate?
    - Data exchange
    - Synchronization
    - Control transfer
  - Who needs to communicate?
    - Applications
    - OS components
    - Servers
  - How do entities communicate
    - ... in monolithic systems?
    - ... in multi-server systems?
    - ... in virtualized systems?
  - How communicate in SDI OS?
    - L4 IPC





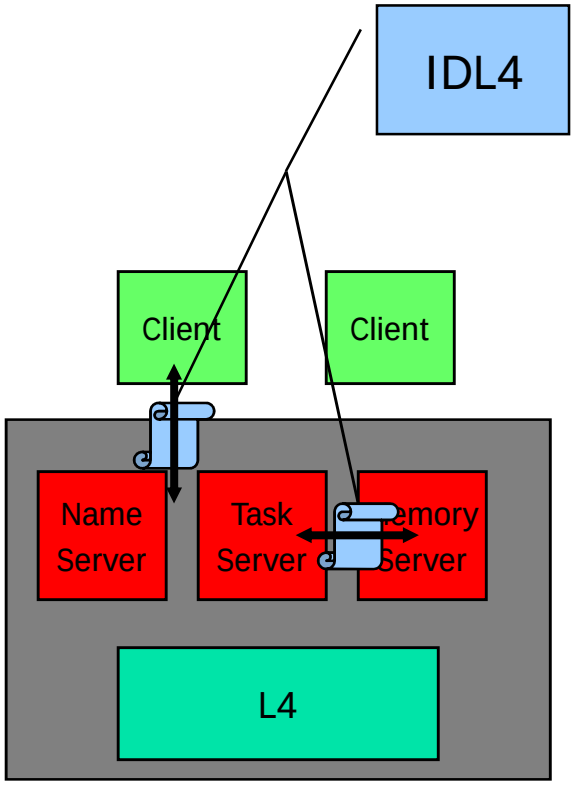
Issues to tackle: Interfaces

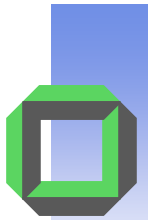
Lecture Part I

- Kernel Interfaces
  - Why interfaces?
    - Want logical separation of
      - Applications
      - OS subsystems
  - How to interface?
    - Need a structured way to interact
    - May want isolation
    - May want privilege separation
    - ...
  - Example OS interfaces
    - ... Linux modules
    - ... Windows WDM drivers
    - ... Multi-Server Systems
    - ...

Lecture Pt.II

- How to construct interfaces in SDI OS?
  - IDL4





Issues to tackle: Naming

Lecture Part I

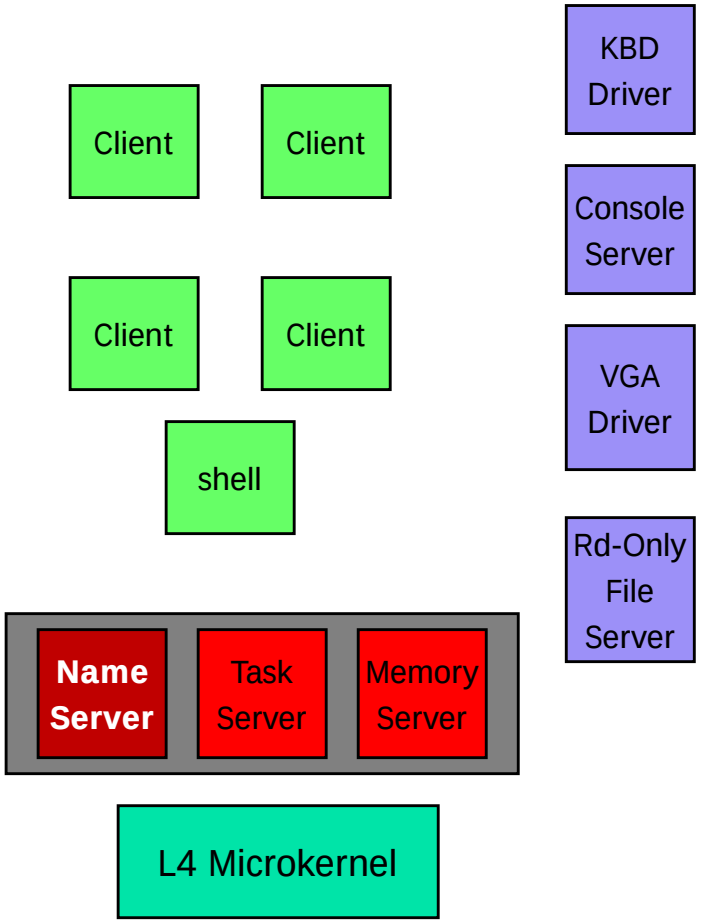
- What objects need names?
  - Names of components
  - Names of objects within components
- How can we navigate the system name spaces?
  - Protocols
  - Interfaces

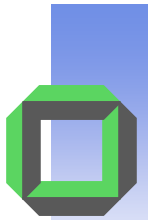
Lecture Pt. II

- SDI name service considerations

Lab Pt. I

- SDI name server design





Issues to tackle: Threads and Tasks

Lecture Part I

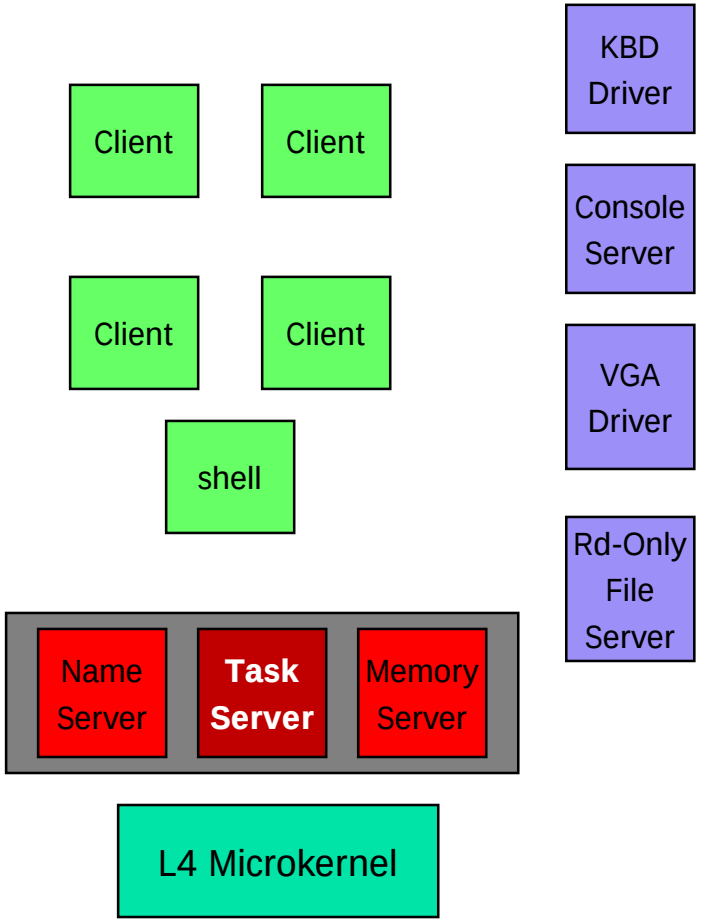
- Threads and Processes
  - Thread and Process Management
  - Program execution
- Thread Scheduling
  - Thread scheduling and accounting
  - Classic scheduling approaches
  - Scheduler activations et al.
- Multi-server systems
  - Scheduling issues
  - Case study: scheduling in K42

Lecture Pt.II

- Tasks in SDI
  - How do we create and destroy tasks?
  - What do we need to create a task?
  - How to find out when a task dies?

Lab Pt. I

- Task server design



Issues to tackle: Memory

Lecture Part I

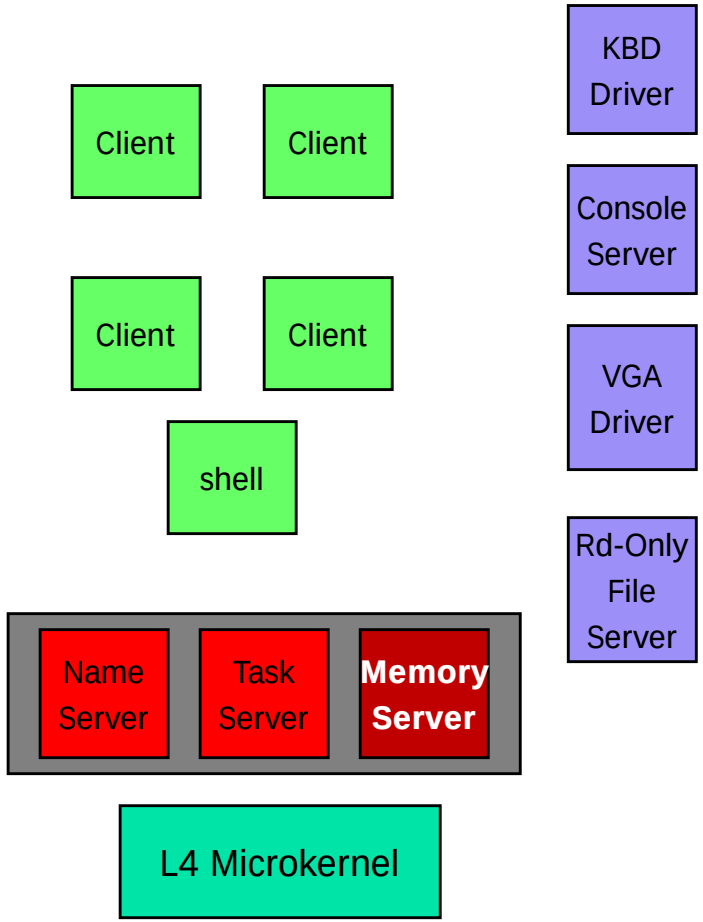
- Virtual memory management
  - Address space construction
  - Memory objects
- Case studies
  - VMM in 4.3 BSD
  - Dataspaces in SawMill
  - Double paging for virtual machines

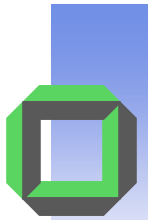
Lecture Pt.II

- Memory in SDI
  - How are page faults handled?
  - How do we construct address spaces
  - What kind of memory objects might we support?

Lab Pt. I

- Memory server design





Issues to tackle: File System

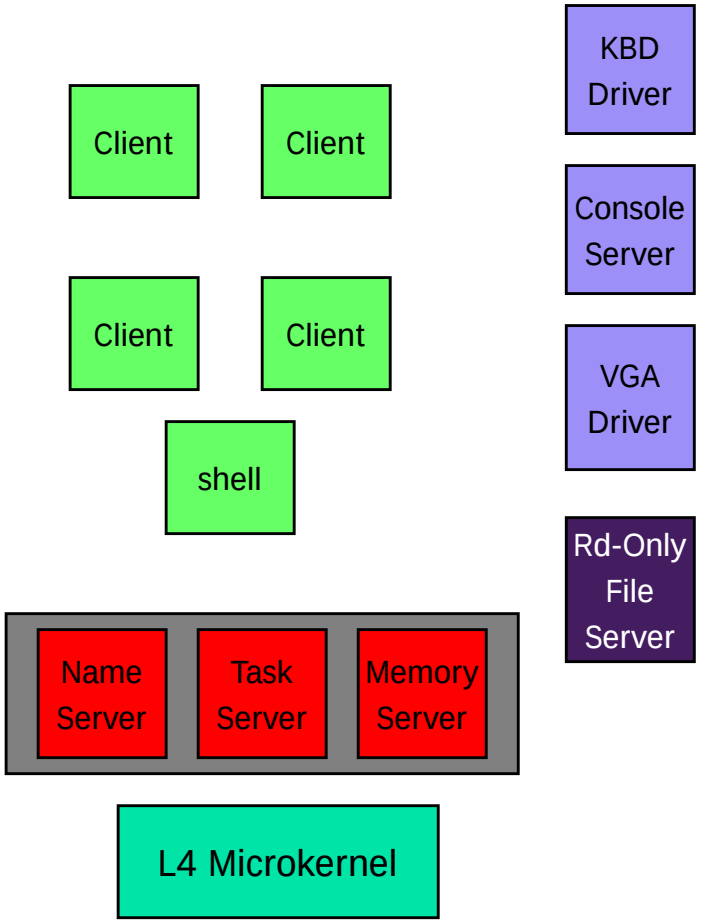
Lecture Part I

- File systems
  - File access techniques
  - ...
- Case studies
  - FAT
  - NFS
- File systems in SDI
  - We need something to provide initial files for testing.
  - Files might be part of initial boot image.
  - What operations are needed to access files?
  - Can we memory map files?

Lecture Pt. II

Lab Pt. I

- File server design
  - Simple read/write in-memory file system.
  - Maybe extended to use disk blocks (IDE driver).



Issues to tackle: Device Support

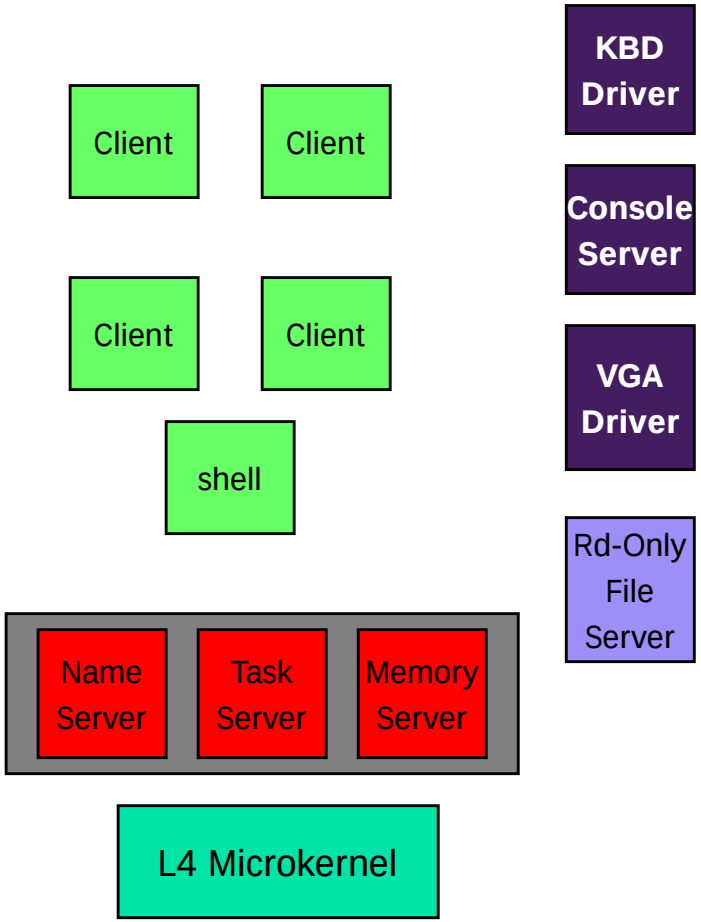
Lecture Part I

- Device drivers
  - Device I/O related concepts
  - Software issues and structure
- Case studies
  - The NS16550A UART
  - PC Screen and Keyboard
- Device drivers in SDI
  - KBD driver
  - Console server
    - Virtual consoles?
    - How to multiplex the hardware?
  - VGA driver

Lecture Pt.II

Lab Pt. I

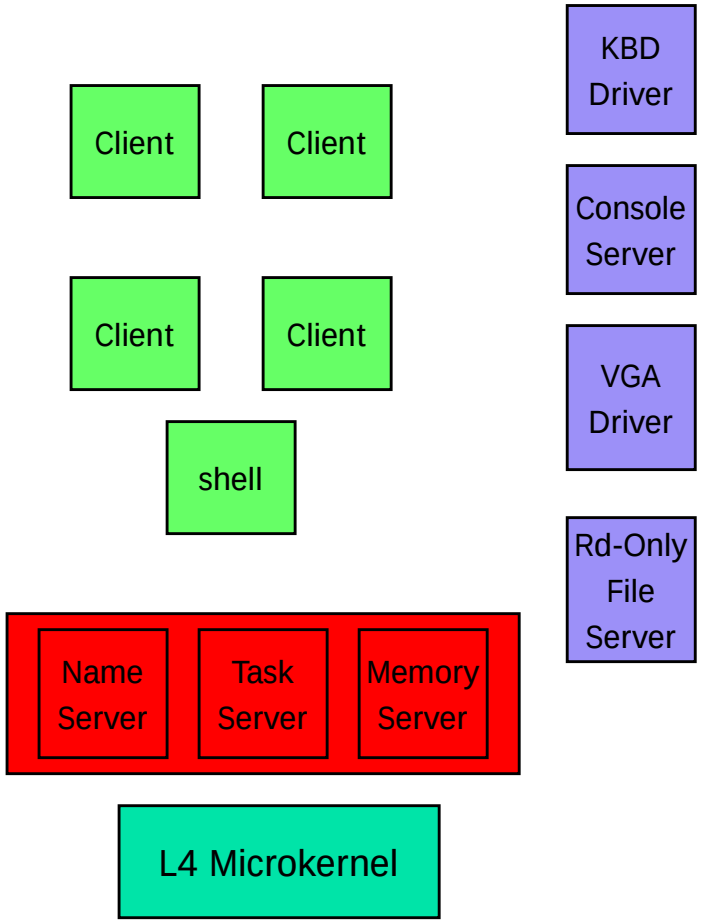
- Device driver design

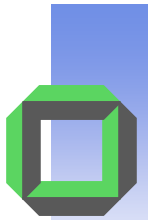


Issues to tackle: Resource Management

Lab

- How do we bootstrap the system?
- What resources need arbitrating?
  - Interrupts
  - Memory
  - Device memory
  - Other ?

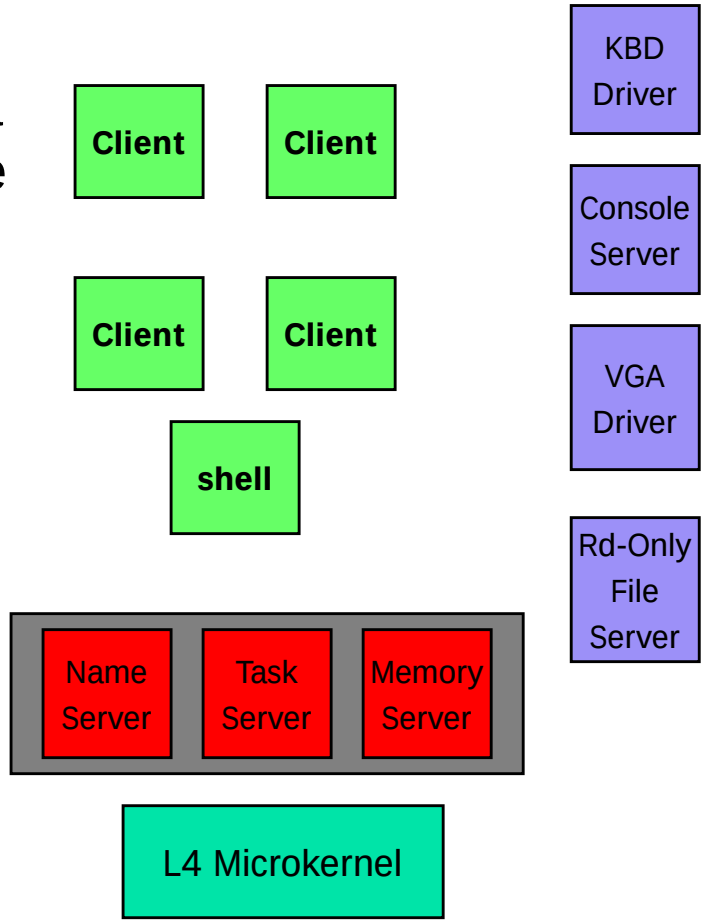


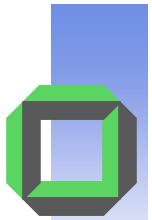


Issues to tackle: Application Support

Lab

- What libraries do we need?
- What do we need to build a simple “shell” to manipulate the system?

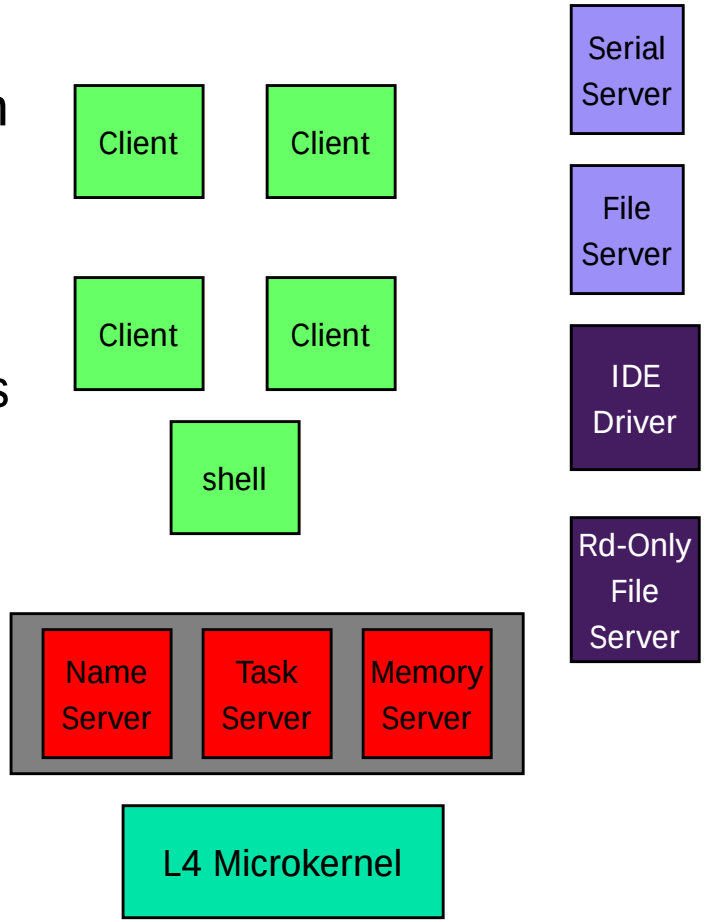


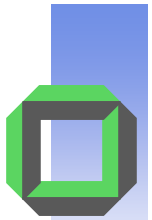


Issues to tackle: IDE

Lab (Optional)

- IDE device driver
  - How do we interact with the hardware?
  - How does the IDE hardware work?
  - What kind of interface is suitable for a block device?
  - How are drivers structured internally?

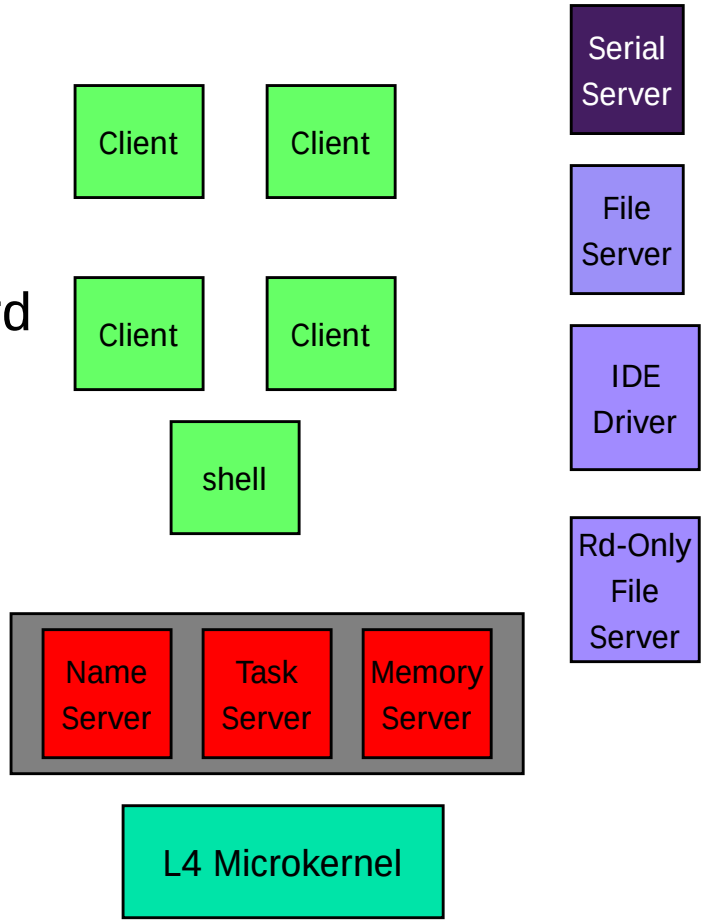




Issues to tackle

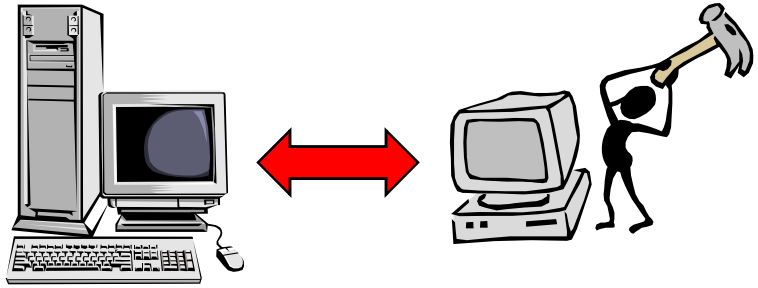
Lab (Optional)

- Serial Port Server
  - Provide a means for interaction with the system.
  - Learn about the standard PC UART chip.

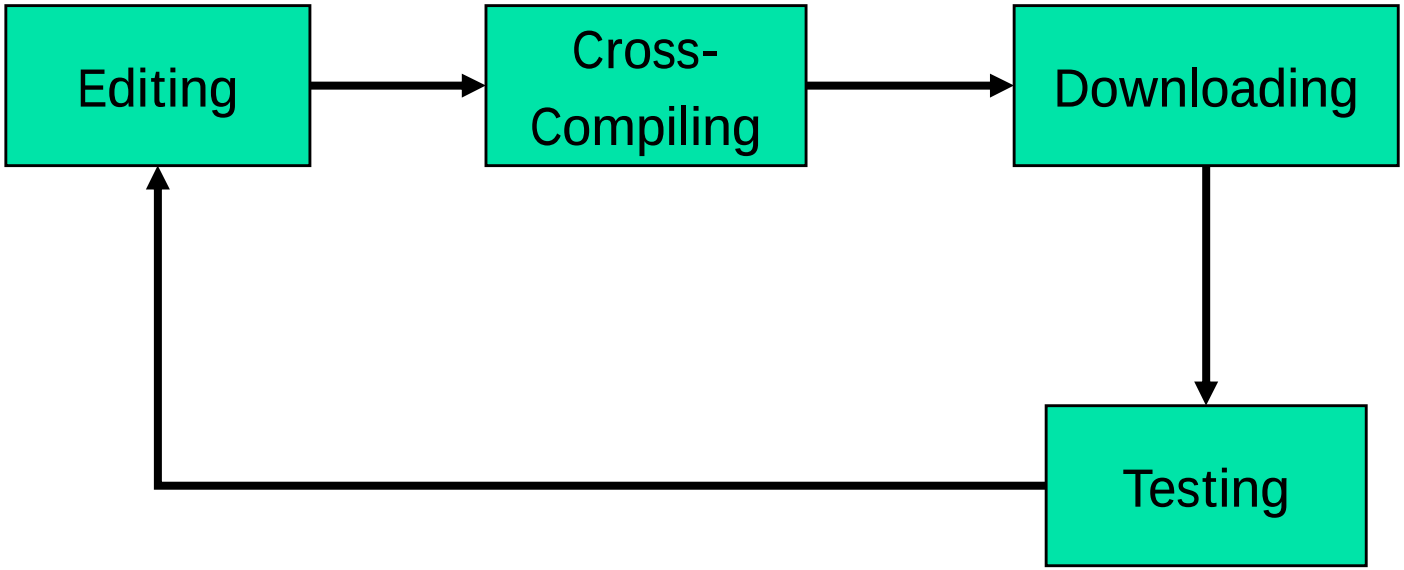


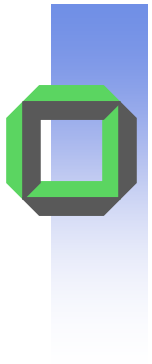
Development Environment

Development Host



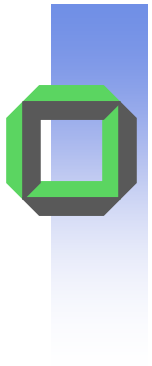
"Crash & Burn" Machine





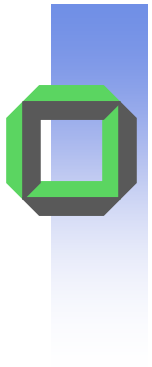
Development Environment

- Standard compiler – **`gcc`**
  - Generates object files for target system (IA-32 ELF32)
  - Same as native Linux compiler
- Include files and libraries
  - Specific for the target environment (L4)
  - Supplied libraries are severely limited (`printf`)
- Target specific linking
  - Custom start-up code **`crt0.o`**
  - Static linking to carefully chosen addresses
- Separate compilation – **`make`**
  - Determines which parts of program need recompiling
  - Issues commands to bring program up-to-date



Download: GRUB (GRand Unified Booter)

- A boot sector which subsequently:
  - Loads the rest of the booter
  - Loads a menu specification file
  - Loads the files from the chosen configuration from either
    - The disk used to load GRUB,
    - Or via the network using BOOTP and TFTP
  - Provides configuration info to the started program
- We will provide detailed info on how to set up GRUB

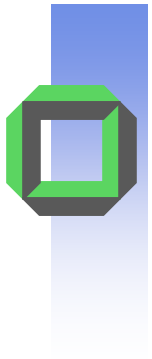


IDL4 Stub Code Generator

- Generates code for component communication
- Interface Definition Language – CORBA
- Pros:
  - Allows clear specification of interfaces
  - Automates remote procedure call (IPC) generation
    - Usually a tedious and error prone task
- Cons:
  - Extra layer to understand
  - Makes debugging more difficult

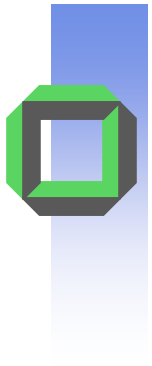
Testing & Debugging

- We use VMware as our Crash&Burn machines
  See http://www.vmware.com/
  - Virtual “standard” PC that runs on Windows NT and Linux
  - \+ Portable, cheap environment
  - \- May have unexpected “features”
- If necessary, we can also use real hardware



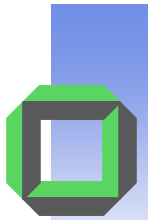
Getting Started

- Available in two weeks
- Development hosts in R.149
- We will provide you with example code and detailed instructions on getting started
  - Test your development environment
  - Provide examples of various features of the system
  - Base upon which to start the larger project



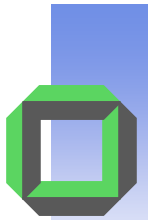
L4 User Manual

- Answers: How do I do ... on L4?
- Minor problem: It does not exist yet.
  - Well, only an outdated version exists.
  - ... and last year's SDI Wiki.
- You can help us write it
  - If you come across a problem, write it down!
  - If you solve a problem, document how!



Support

- Web sites
  - http://i30www.ira.uka.de/teaching/courses/sdi
    Official SDI site
  - http://l4ka.org/
  - http://l4hq.org/
- Mailing list
  - Proposal: Mass subscription next week
- Ask our tutor – R154



Thursday

- L4 API crash course – Part I

- TODO: form teams, first attempt
  - 3-4 students per team
  - No more than 12 teams
    - Use the registration web page
    - Employ a collision-free algorithm for allocating team numbers!!!
  - All team members should have similar goals