CS 5264/4224; ECE 5414/4414
(Advanced) Linux Kernel Programming
Lecture 2

Building and Exploring Linux Kernel

January 23, 2025

Huaicheng Li

Acknowledgement: Dr. Changwoo Min for the original LKP slides.

Linux is a Monolithic Kernel

- A traditional design: all of the OS runs in kernel, privileged mode
  - share the same address space
- Kernel interface ~= system call interface
- Good: easy for subsystems to cooperate
  - one cache shared by file system and virtual memory
- Bad: interactions are complex leads to bugs, no isolation within kernel

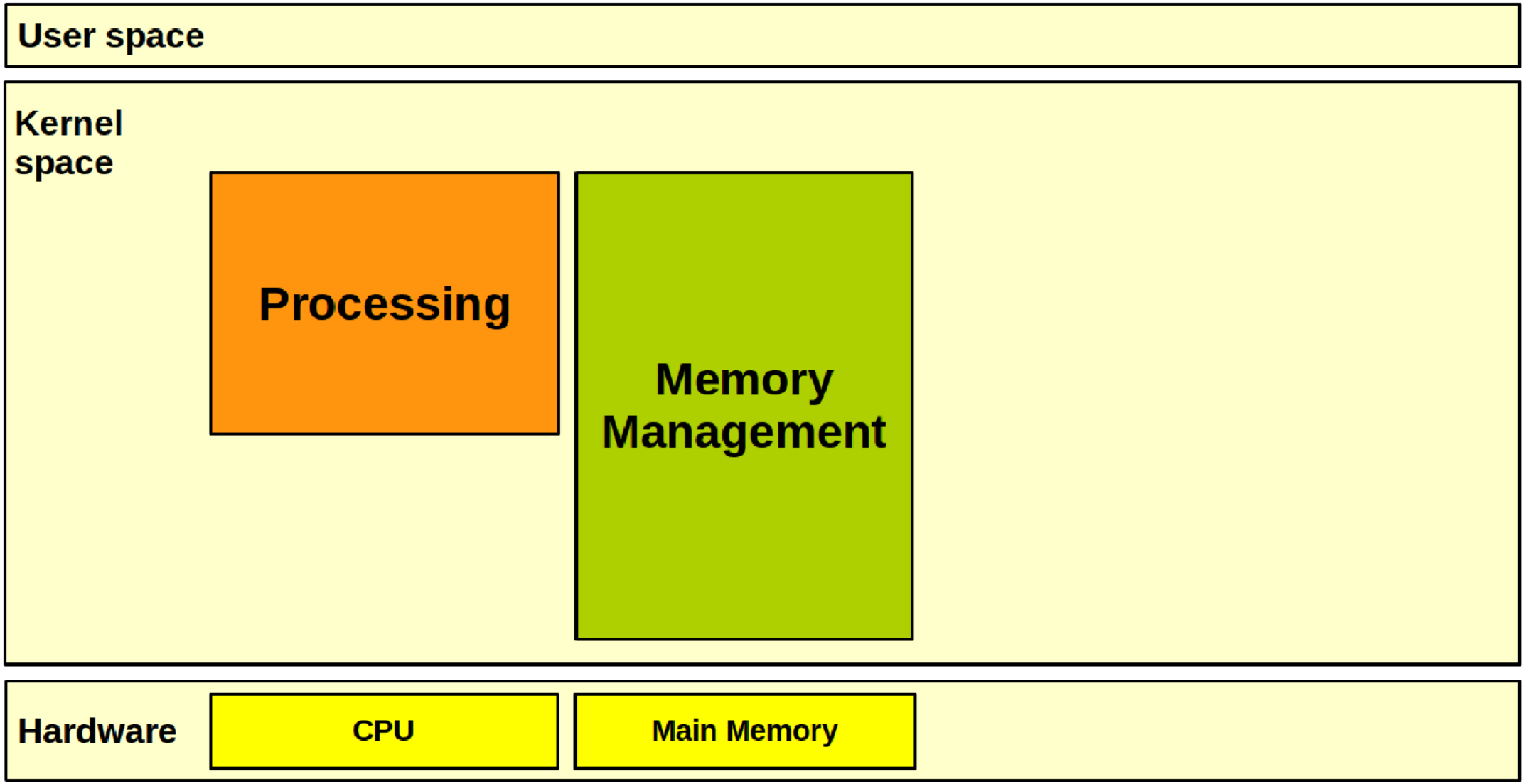
Alternative: Microkernel Design

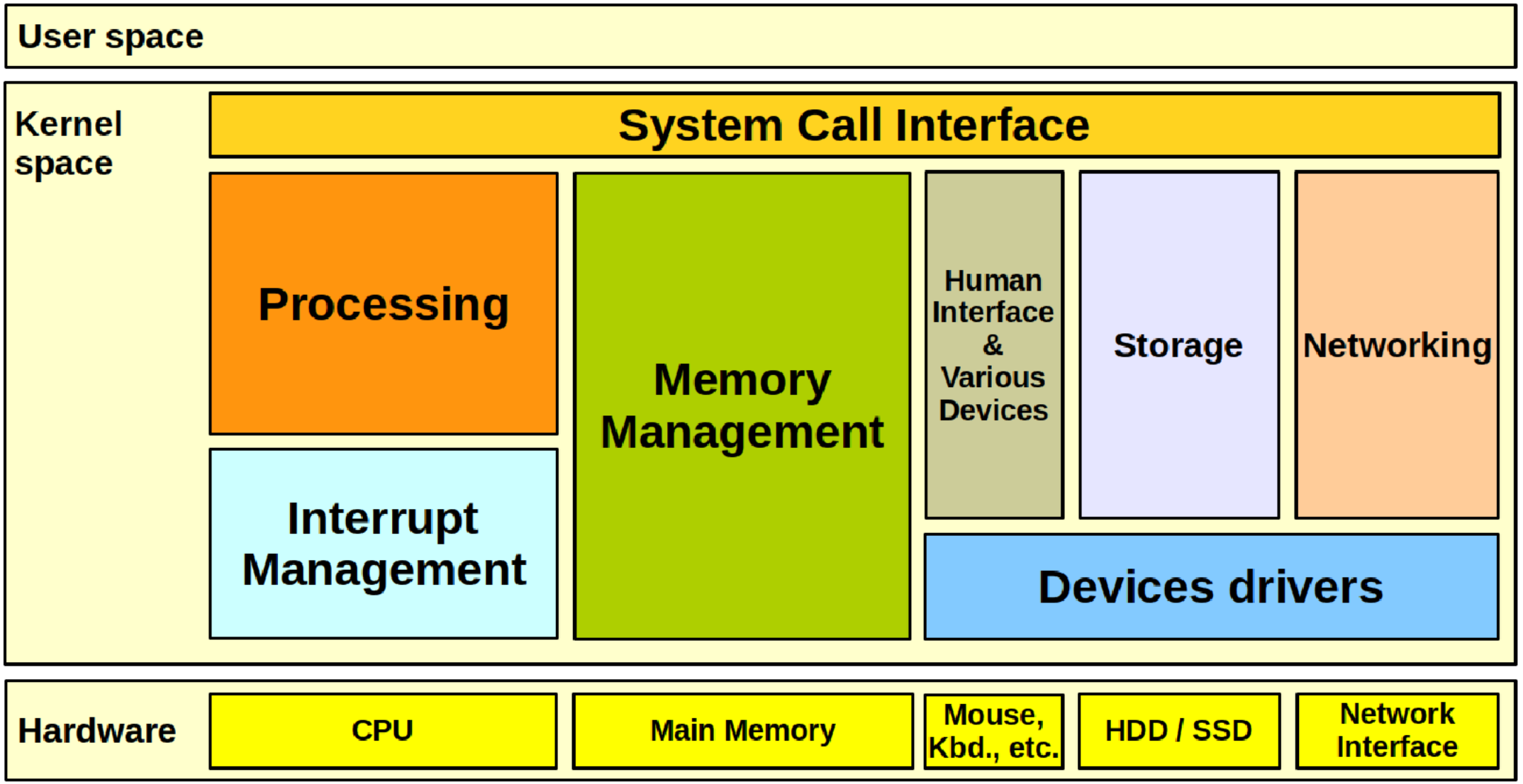
- Many OS services run as ordinary user programs
  - e.g., file system in a file server
- Kernel implements minimal mechanism to run services in user space
  - IPC, virtual memory, threads
- Kernel interface != system call interface
  - applications talk to servers via IPCs
- Good: more isolation
- Bad: IPCs may be slow

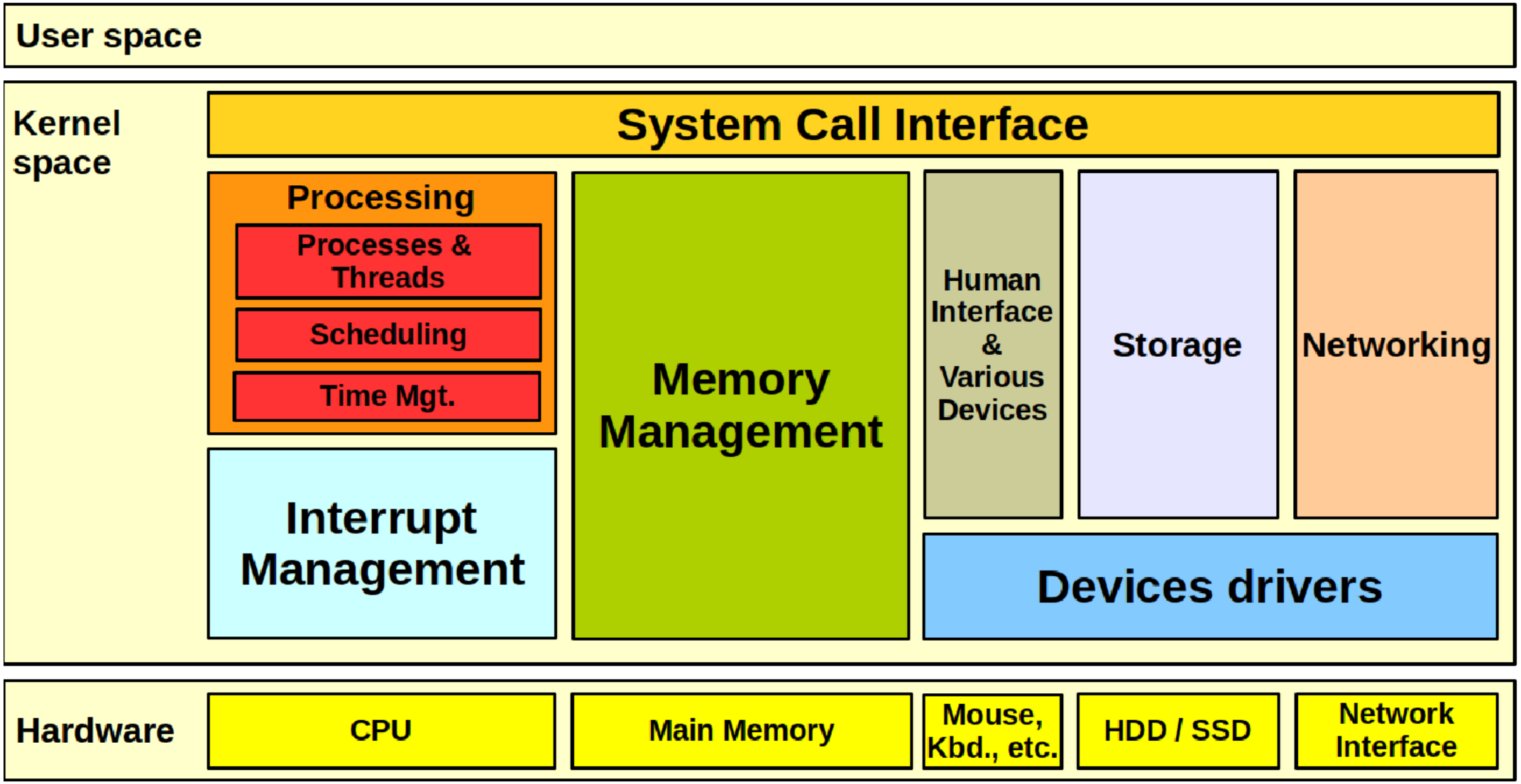
Debate

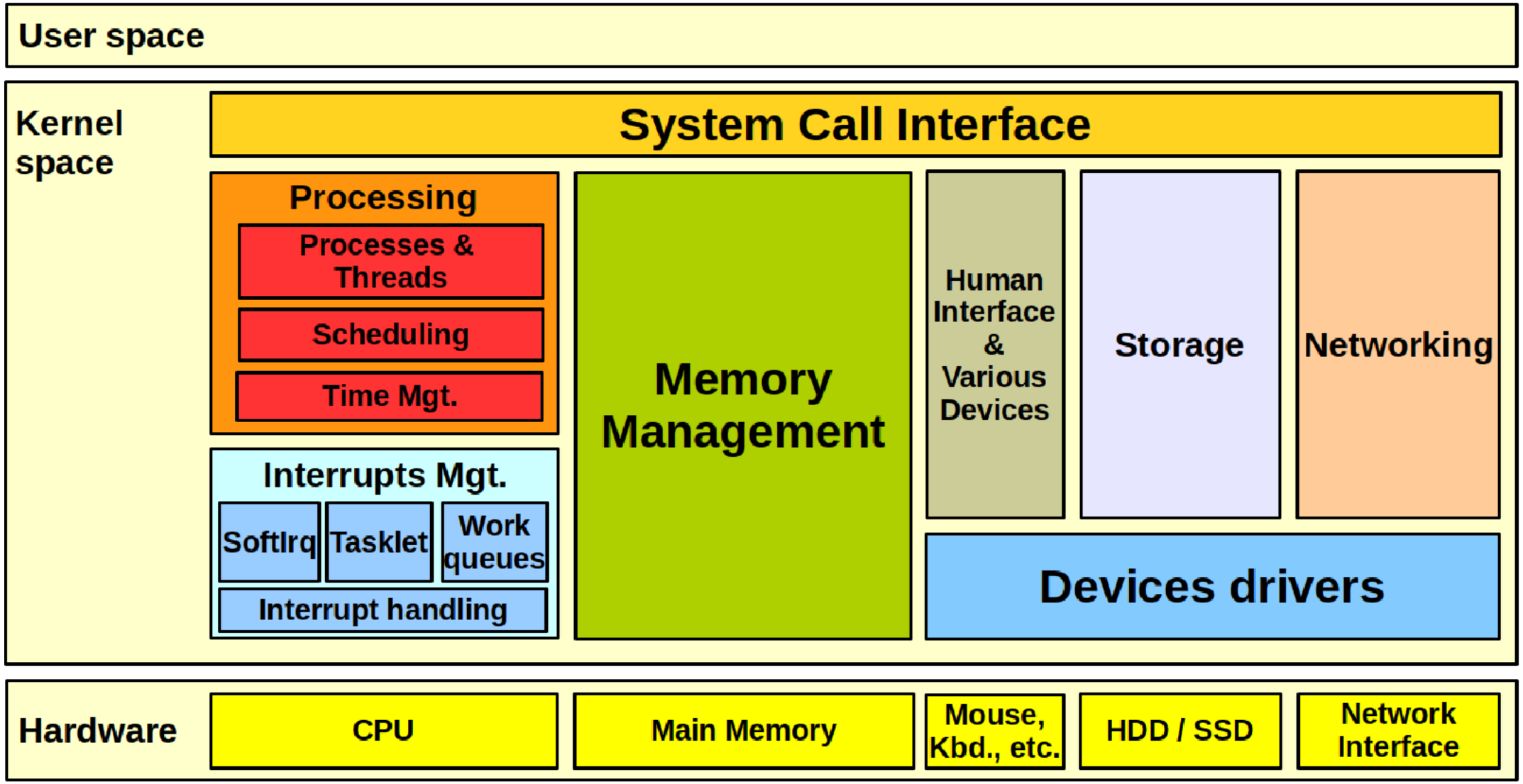
- Tanenbaum-Torvalds debate
- Most real-world kernels are mixed: Linux, OS X, Windows
  - e.g., X Window Systems

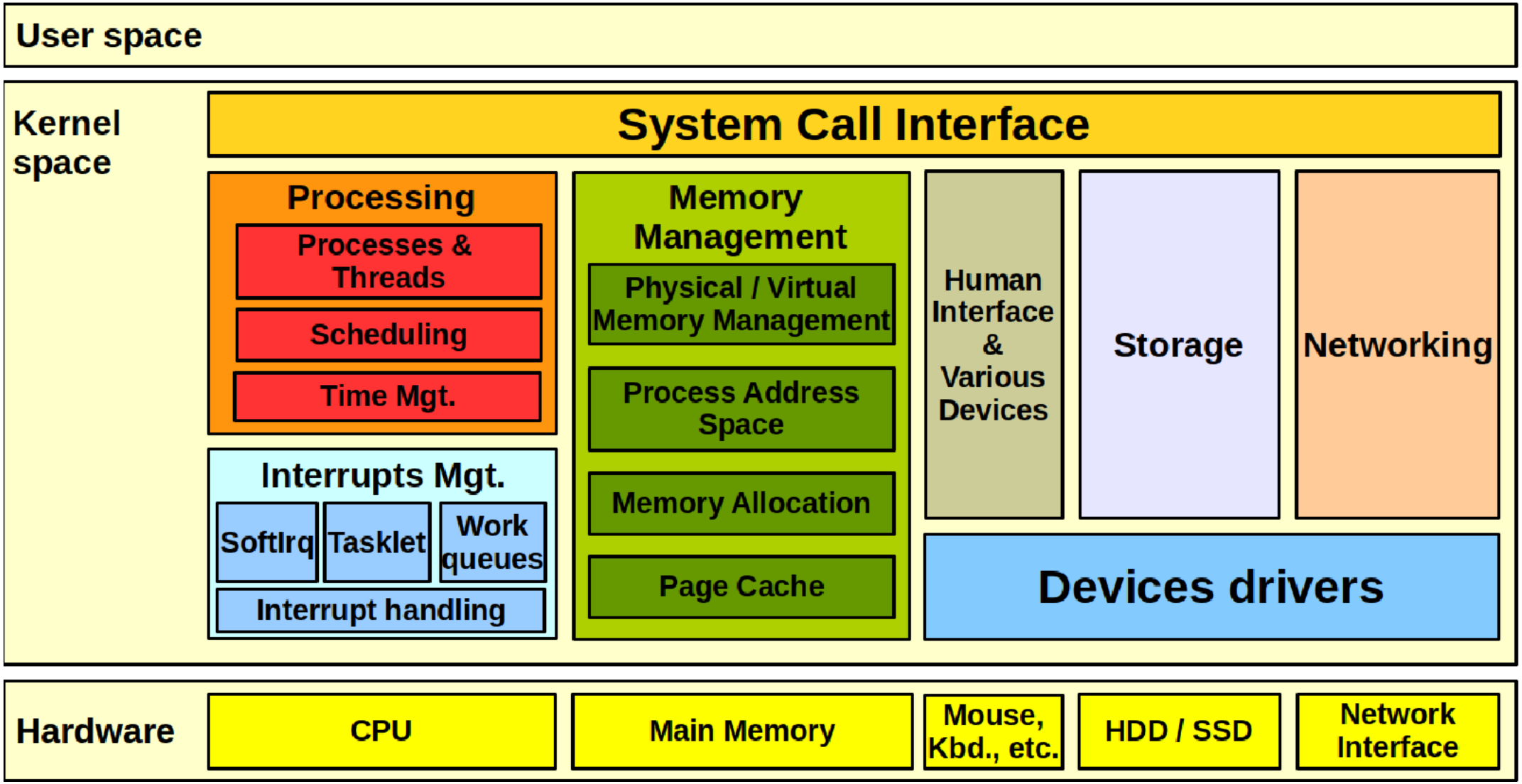
Kernel and Course Map

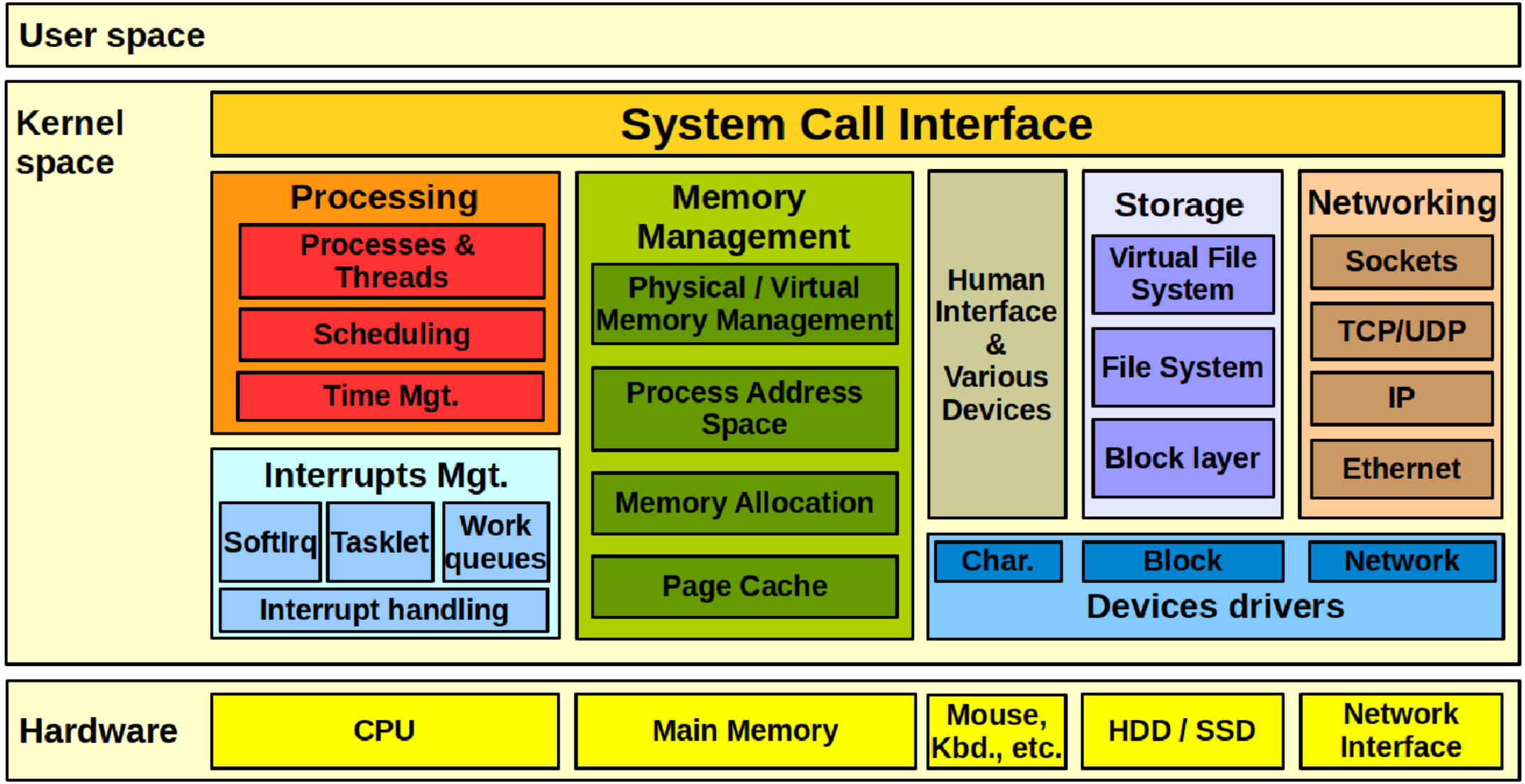


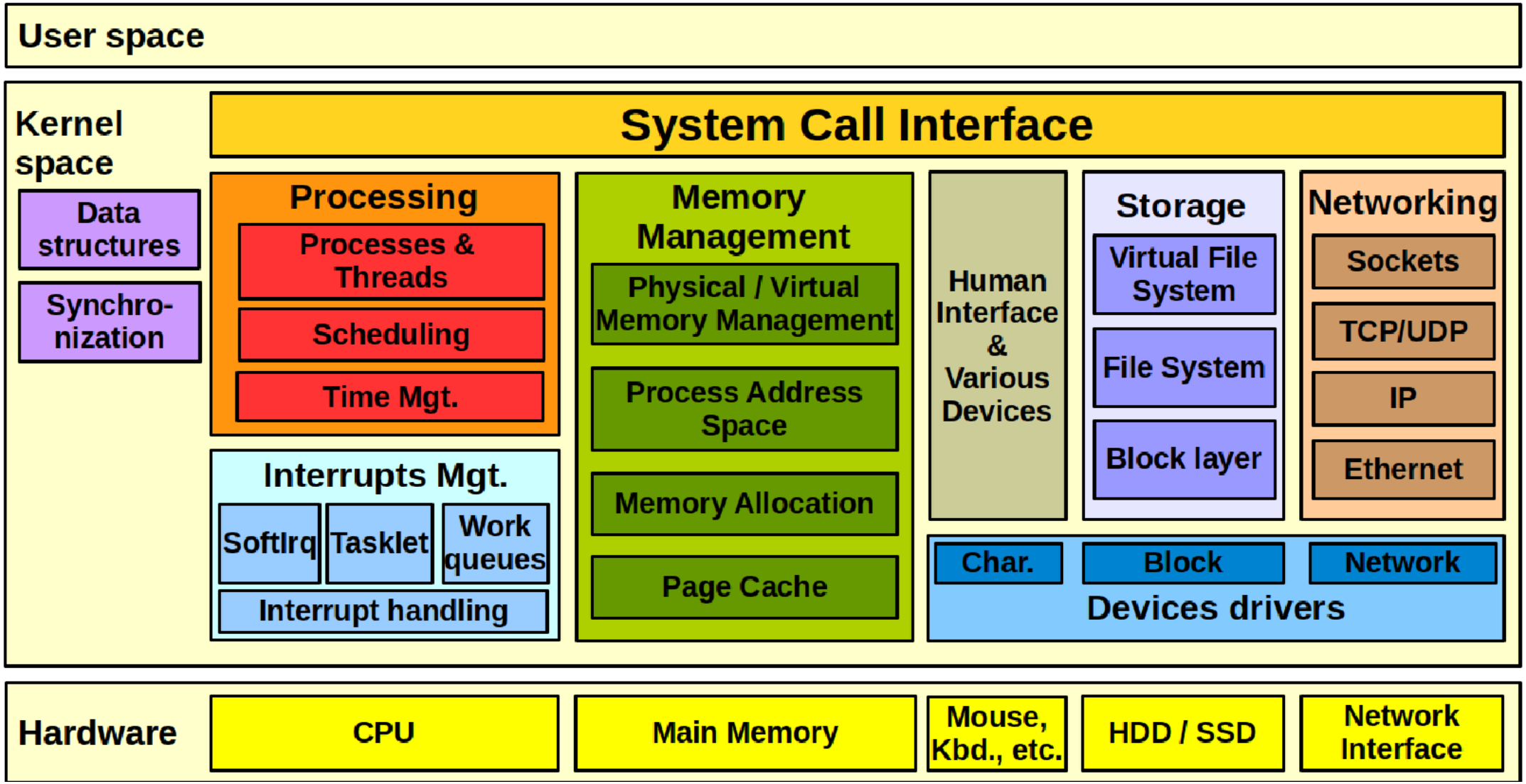

User space
Kernel space
System Call Interface
Processing
Interrupt Management
Memory Management
Human Interface & Various Devices
Storage
Networking
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface

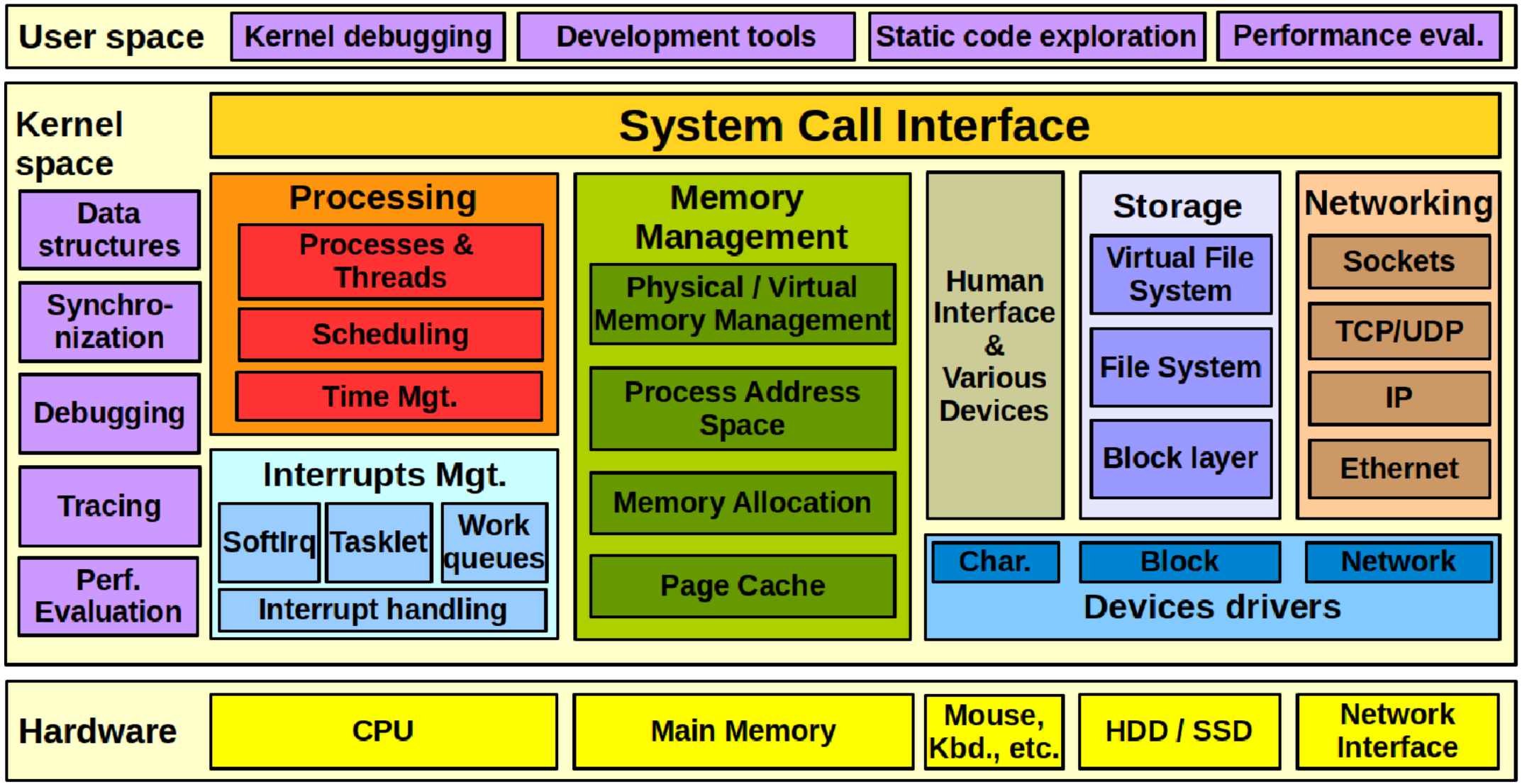

User space
Kernel space
System Call Interface
Processing
Processes & Threads
Scheduling
Time Mgt.
Interrupt Management
Memory Management
Human Interface & Various Devices
Storage
Networking
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface


User space
Kernel space
System Call Interface
Processing
Processes & Threads
Scheduling
Time Mgt.
Interrupts Mgt.
SoftIrq
Tasklet
Work queues
Interrupt handling
Memory Management
Human Interface & Various Devices
Storage
Networking
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface


User space
Kernel space
System Call Interface
Processing
Processes & Threads
Scheduling
Time Mgt.
Interrupts Mgt.
SoftIrq
Tasklet
Work queues
Interrupt handling
Memory Management
Physical / Virtual Memory Management
Process Address Space
Memory Allocation
Page Cache
Human Interface & Various Devices
Storage
Networking
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface


User space
Kernel space
System Call Interface
Processing
Processes & Threads
Scheduling
Time Mgt.
Interrupts Mgt.
SoftIrq
Tasklet
Work queues
Interrupt handling
Memory Management
Physical / Virtual Memory Management
Process Address Space
Memory Allocation
Page Cache
Human Interface & Various Devices
Storage
Virtual File System
File System
Block layer
Networking
Sockets
TCP/UDP
IP
Ethernet
Char.
Block
Network
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface


User space
Kernel space
System Call Interface
Data structures
Synchro-nization
Processing
Processes & Threads
Scheduling
Time Mgt.
Interrupts Mgt.
SoftIrq
Tasklet
Work queues
Interrupt handling
Memory Management
Physical / Virtual Memory Management
Process Address Space
Memory Allocation
Page Cache
Human Interface & Various Devices
Storage
Virtual File System
File System
Block layer
Networking
Sockets
TCP/UDP
IP
Ethernet
Char.
Block
Network
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface


User space
Kernel debugging
Development tools
Static code exploration
Performance eval.
Kernel space
System Call Interface
Data structures
Synchro-nization
Debugging
Tracing
Perf. Evaluation
Processing
Processes & Threads
Scheduling
Time Mgt.
Interrupts Mgt.
SoftIrq
Tasklet
Work queues
Interrupt handling
Memory Management
Physical / Virtual Memory Management
Process Address Space
Memory Allocation
Page Cache
Human Interface & Various Devices
Storage
Virtual File System
File System
Block layer
Networking
Sockets
TCP/UDP
IP
Ethernet
Char.
Block
Network
Devices drivers
Hardware
CPU
Main Memory
Mouse, Kbd., etc.
HDD / SSD
Network Interface

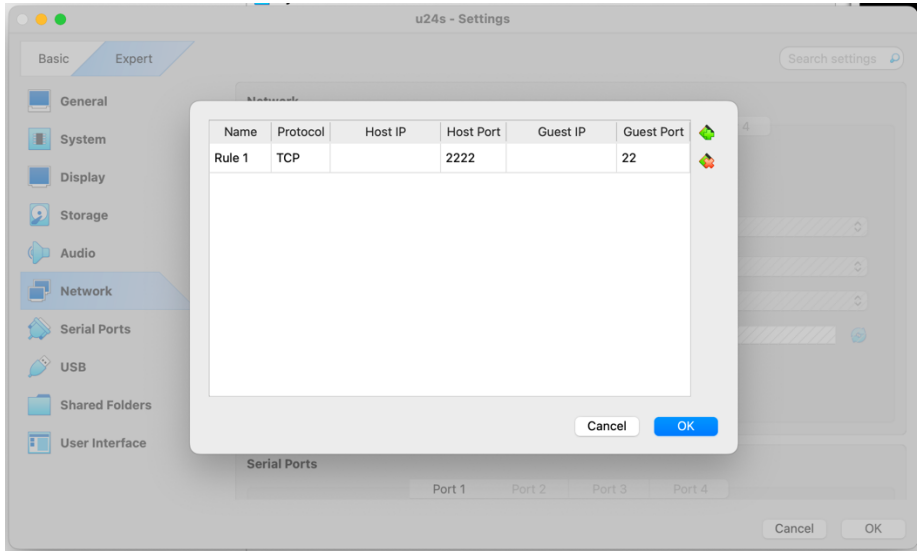
Today's Focus

- Tools
  - Version control: git, tig
  - Configure, build, and install the kernel: make
  - Explore the code: cscope, ctags
  - Editor: vim, emacs
  - Screen multiplexing: tmux
- Kernel vs. user programming

Option 1: Installing Linux on VirtualBox

- Support Windows / MacOS / Linux hosts
- Add a port forwarding rule on VirtualBox
  - map the guest-OS's port to the host OS
- Login from a terminal on the host
  - ssh -p 2222 $username@localhost
- File transmissions between host and VM

  (do the following from the host)
  - Host → VM: scp -P 2222 src_file $username@localhost:/path/to/file/in/the/VM
  - VM → Host: scp -P 2222 $username@localhost:/path/to/file/in/the/VM $target-path



Optional 2: Installing Linux on QEMU

- QEMU is a powerful machine emulator
- Can be used for debugging Linux kernel (similar to using gdb to debug user program)
- Use FEMU/QEMU if you already run Linux on your host
  - QEMU: https://www.qemu.org/
  - FEMU: https://github.com/MoatLab/FEMU (A QEMU variant, *recommended*)
    - » Detailed instructions, check github repo
    - » Directly usable VM image
    - » The usage is exactly the same as QEMU, and it is functionally equivalent to VirtualBox
    - » It should only take you 10-20min to set it up and get it running ...

Obtaining Linux Kernel Source Code

- Tarball
  - https://www.kernel.org
- Git repo
  - Linus's git: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/
  - Github mirror: https://github.com/torvalds/linux

Version Control: git

- Git is a version control software
  - tracking changes in files in an organized way
- Developed by Linus Torvalds
  - Extensively used by many other software
  - Github (https://github.com) is a git service provider, gitlab / bitbucket are similar
- Distributed revision control system
  - Every git repo is a full-fledged repository with complete editing history

Essential git Command

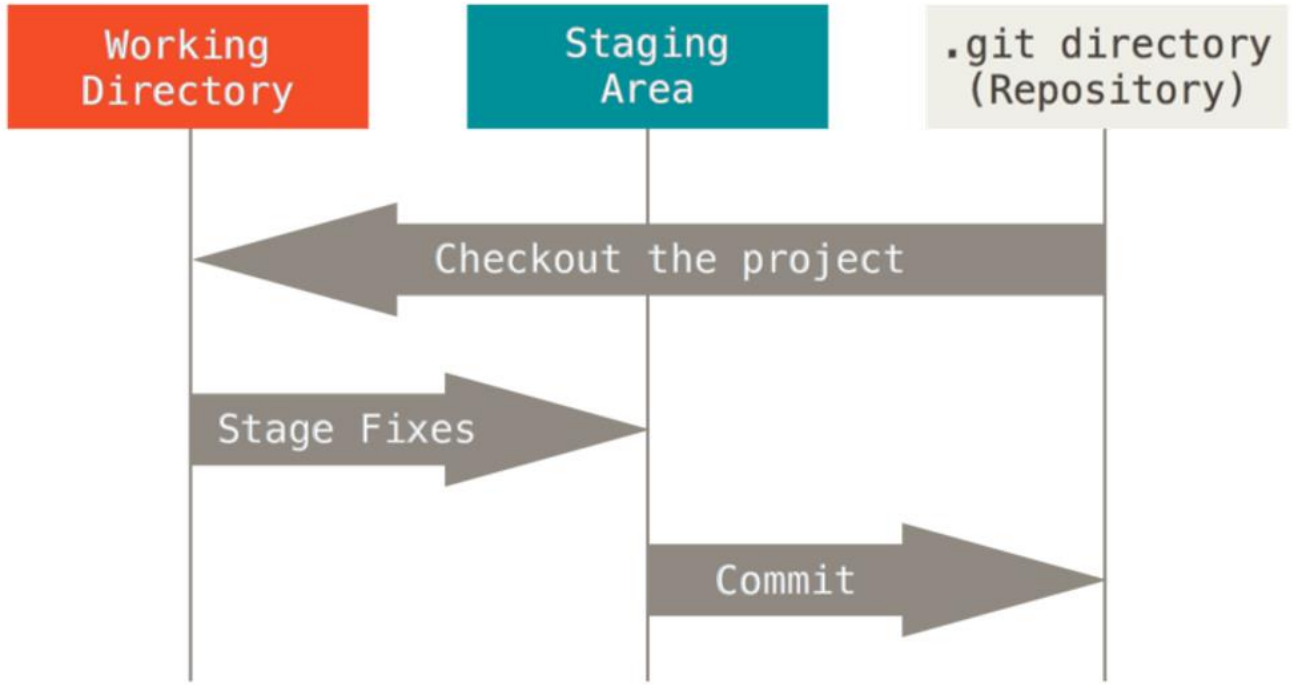
```
# 1. install and configure
$ sudo apt-get install git
$ git config --global user.name "John Doe"  # set your name and email for history
$ git config --global user.email johndoe@example.com

# 2. clone a repository
$ git clone https://github.com/torvalds/linux.git # clone an existing repo

# 3. tags
$ git tag             # list all existing tags
$ git checkout v6.12 # checkout the tagged version

# 4. commit history (or use tig for prettier output)
$ git log             # show all commit history
$ git log <file>      # show changes over time for a file
$ git blame <file>    # who changed what and when in <file>
```

 # 5. local changes
$ git status # show changed files
$ git diff # show changed lines
$ git add <file> # add <file> to the next commit
$ git commit # commit previously staged files to my local repo

 # 6. publish and update
$ git push # publish a committed local changes to a remote repo
$ git pull # update a local repo

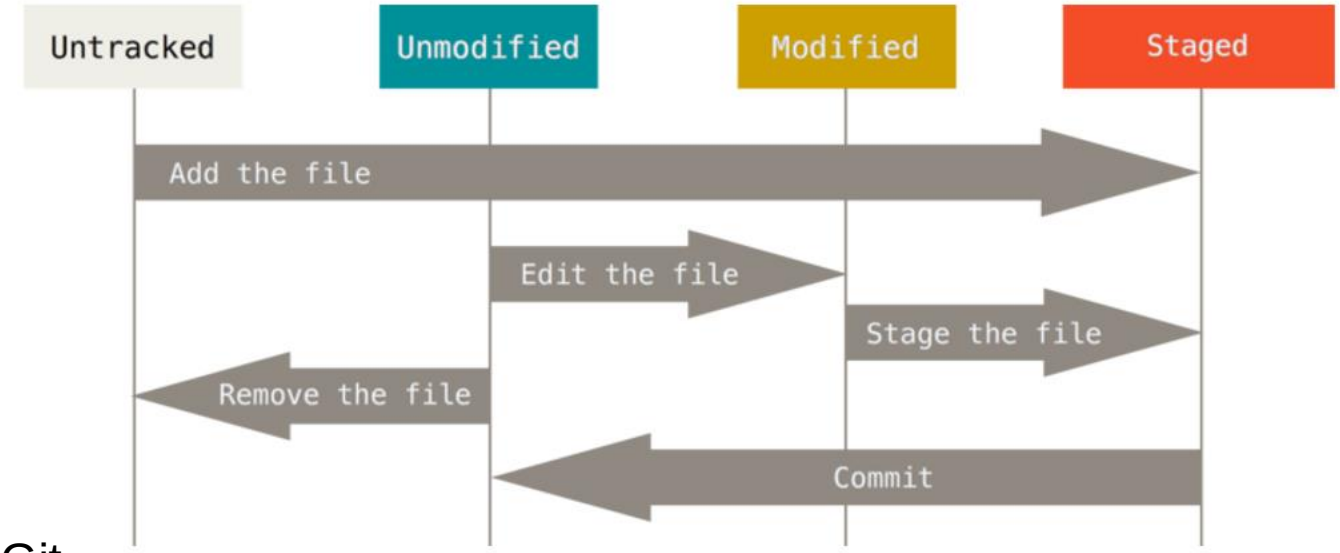
- **Useful git tutorials:**
  - Linux kernel (https://www.kernel.org/pub/software/scm/git/docs/gittutorial.html)
  - Pro Git: https://git-scm.com/book/en/v2

Git Workflow



Source: Pro Git

- **git add $a-new-file** *# Untracked -> Staged*
- **git add $an-existing-git-managed-file** *# Modified -> Staged*
- **git rm $an-existing-git-managed-file** *# Unmodified -> Untracked*
- **git commit** *# Staged -> Unmodified*



Source: Pro Git

Linux Kernel Source Tree

```
$ git clone https://github.com/torvalds/linux.git # clone the kernel repo
$ cd linux; git checkout v6.12   # checkout v6.12
$
$ tree -d -L 2      # list top two-level directories
├── arch                    # * architecture dependent code
│   ├── arm                 #   - ARM architecture
│   └── x86                 #   - Intel/AMD x86 architecture
├── block                   # * block layer: e.g., IO scheduler
├── Documentation           # * design documents
├── drivers                 # * device drivers
│   └── nvme                #   - NVMe SSD
├── fs                      # * virtual file system (VFS)
│   ├── ext4                #   - ext4 file system
│   └── xfs                 #   - XFS file system
├── include                 # * include files
│   ├── linux               #   - include files for kernel
│   └── uapi                #   - include files for user-space tools
├── init                    # * bootig: start_kernel() at main.c
├── ipc                     # * IPC: e.g., semaphore
│   ...
```

```
$ git clone https://github.com/torvalds/linux.git # clone the kernel repo
$ cd linux; git checkout v6.12   # checkout v6.12
$
$ tree -d -L 2      # list top two-level directories
│   ...
├── kernel            # * core features of the kernel
│   ├── locking       #   - locking: e.g., semaphore, mutex, spinlock
│   └── sched         #   - task scheduler
├── lib               # * common library: e.g., red-black tree
├── mm                # * memory management: e.g., memory allocation, paging
├── net               # * network stack
│   ├── ipv4          #   - TCP/IPv4
│   └── ipv6          #   - TCP/IPv6
├── security          # * security framework
│   └── selinux       #   - selinux
├── tools             # * user-space tools
│   └── perf          #   - perf: performance profiling tool
└── virt              # * virtualization
    └── kvm           #   - KVM type-2 hypervisor
```

Building the Kernel

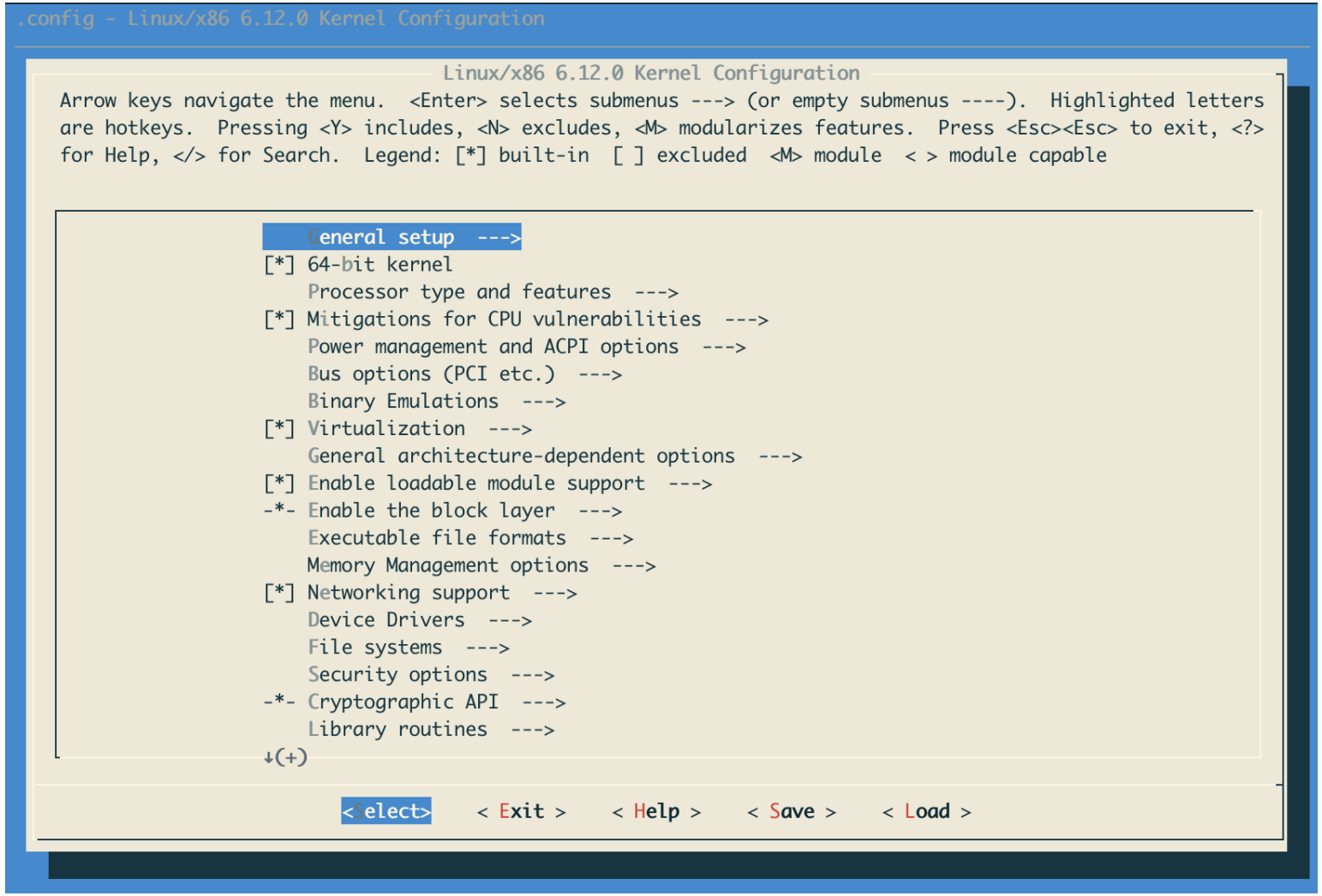
- **Configure the kernel**
  - Various compilation options (Check the ***Makefile***)
  - Configure file (.config)
- **Compiling the kernel**
  - compile and link the kernel source code (OS image and drivers)
- **Install the new kernel**
  - Add boot entry to the bootloader, e.g., grub
- **Use "make help" to see the detailed make options**
- **Reference: Documentation/admin-guide/README.rst**

Configure the Kernel

- Install dependencies: sudo apt install -y flex bison libncurses5 pkg-config libncurses5-dev libelf-dev libssl-dev
- make menuconfig
- make defconfig
  - Generate the default configuration of the running platform
  - linux/arch/x86/configs/x86_64_defconfig
- make oldconfig
  - Use the configuration file of running kernel
  - Will ask about new configurations
    - » If you're not sure, jut choose the default options
- make localmodconfig
  - Disable kernel features and drrivers for hardware or modules that are not currently in use on the system
  - Useful for creating a leaner and smaller kernel tailored to specific hardware environment

Kernel Configuration File

- ".config" file is at the root of the kernel source
  - preprocessor flags in the source code



Compile and Install the Kernel

- Compile the kernel: “make”
  - Compile all the source code according to kernel configuration file (.config)
  - Compiled kernel image: arch/x86/boot/bzImage
  - Use “-j XXX” for parallel “make”, specify the “XXX” as the number of CPUs to use
  - It could take a few minutes up to tens of minutes, depending on your .config
- Compile the kernel modules: “make modules”
- Installation
  - Install the modules: “sudo make modules_install”
  - Install the kernel image: “sudo make install”

Exploring the Code

- Code indexing tools (Web-based)
  - Elixir Cross Referencer: https://elixir.bootlin.com/linux/v6.12/source
  - Github: https://github.com/torvalds/linux/tree/v6.12
    - » search for identifiers (functions, variables, etc.)
    - » quickly lookup function declaration/definition
- cscope and ctags: for C code
  - installation: sudo apt install cscope exuberant-ctags
  - build cscope database for linux kernel
    - » cd linux; KBUILD_ABS_SRCTREE=1 make cscope # for all architectures
    - » cd linux; KBUILD_ABS_SRCTREE=1 ARCH=x86 make cscope # only for x86
    - » requires rebuilding the database after code changes
  - cscope is standalone, typically use "cscope -R" to build a database for any C project, "make cscope" is optimized for the kernel source code
  - vim + cscope, vim + ctags
- GenAI: only use it in good faith

cscope

- Search for:
  - C idnetifier occurrences (variable, function, struct, macros, etc.)
  - function/variable definition
  - functions called by/calling function
  - text string
- Terminating cscope: Ctrl-d

```
Cscope version 15.9



Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

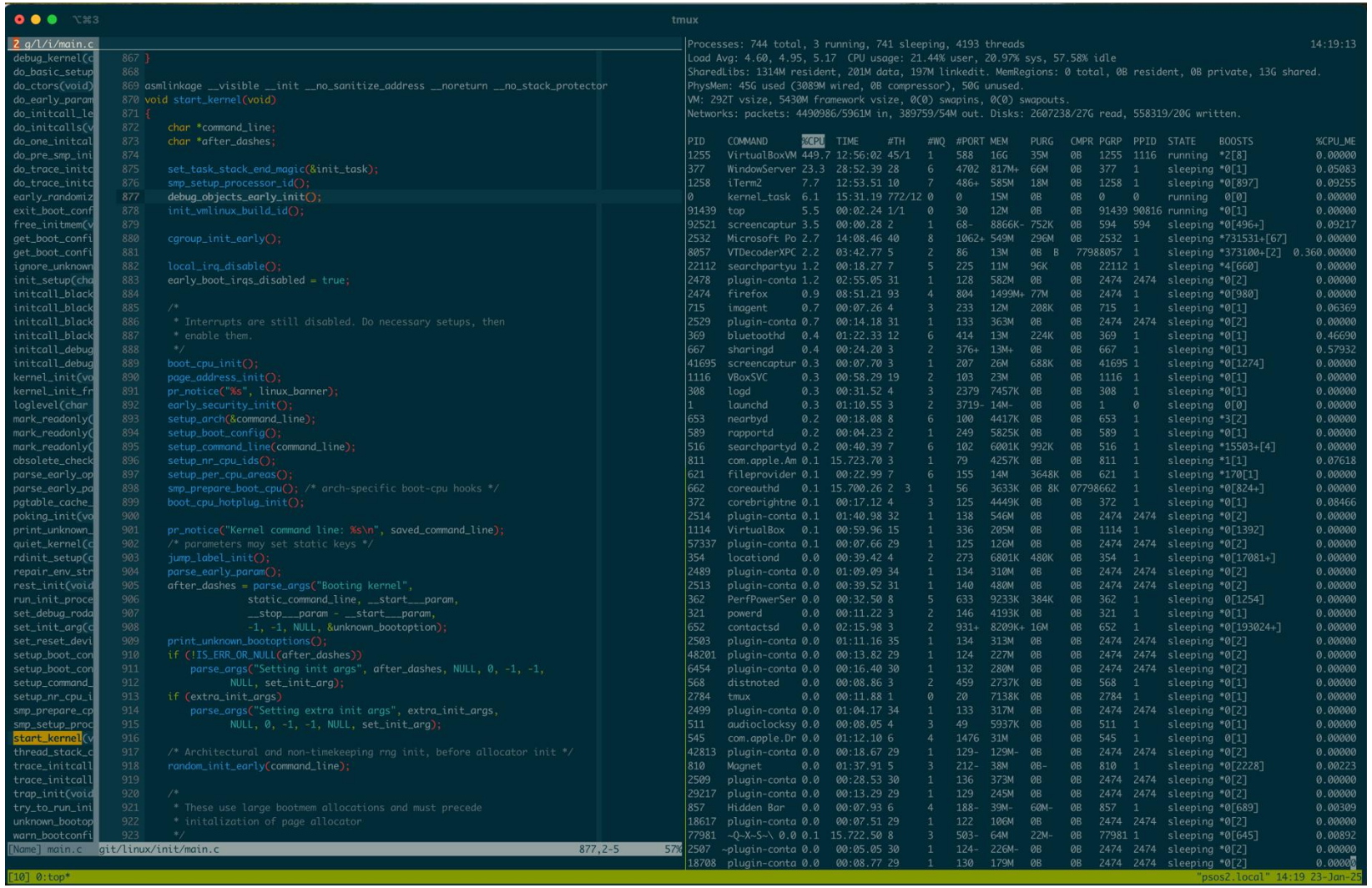
```
C symbol: spin_lock

  File                Function                                 Line
0 binder.c            binder_transaction                      3050 spin_lock(&in_reply_to->lock);
1 binder.c            binder_transaction                      3195 spin_lock(&tmp->lock);
2 binder.c            binder_transaction                      3211 spin_lock(&tmp->lock);
3 binder.c            binder_thread_release                   5277 spin_lock(&t->lock);
4 binder.c            binder_thread_release                   5310 spin_lock(&t->lock);
5 binder.c            binder_node_release                     6148 spin_lock(&binder_dead_nodes_lock);
6 binder.c            print_binder_transaction_ilocked        6323 spin_lock(&t->lock);
7 binder.c            state_show                              6723 spin_lock(&binder_dead_nodes_lock);
8 binder.c            state_show                              6740 spin_lock(&binder_dead_nodes_lock);
9 binder_alloc.c      binder_alloc_prepare_to_free             172 spin_lock(&alloc->lock);
a binder_alloc.c      binder_alloc_new_buf                     600 spin_lock(&alloc->lock);
b binder_alloc.c      binder_alloc_free_buf                    796 spin_lock(&alloc->lock);
c binder_alloc.c      binder_alloc_deferred_release            896 spin_lock(&alloc->lock);
d binder_alloc.c      binder_alloc_print_allocated             967 spin_lock(&alloc->lock);

* Lines 250-264 of 10571, 10308 more - press the space bar to display more *
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

VIM + cscope; VIM + ctags

- vim can use the tag database of cscope, as well as ctags
- Search for function definition/variable declaration:
- ":tag start_kernel" or "cs find global start_kernel"
- ":help tag", or "help cs"
- Another way to find a function definition/variable declaration
  - Put the cursor on the symbol and press `Ctrl+]` or `Ctrl-t`
- To navigate back and forth between file:
  - ":bp" or ":bn"
- Cheatsheet:
  - "cs find s <symbol>" # Find the definition of a symbol
  - "cs find g <symbol>" # Find all references to a symbol
  - "cs find d <symbol>" # Find all functions that call a symbol (*direct callers*)
  - "cs find c <symbol>" # Find all functions called by a symbol
  - "cs find e <symbol>" # Find all symbols matching the symbol name (macros, variable)

Screen Multiplexing: tmux

- tmux is a tool to manage virtual consoles



Basic tmux commands

- tmux: start a new tmux session
  - or "tmux new-sesssion –s xxx"
- Ctrl-b % : split a pane vertically
- Ctrl-b " : split a pane horizontally
- Ctrl-b o: move to the next pane
- Ctrl-b z: zoom or restore a pane
- Ctrl-b c: create a new window
- Ctrl-b N: go to window (0-9)
- Ctrl-b d: detach from a session
- tmux a: attach to an existing session
  - "tmux ls"
  - "tmux a -t xxx"
- resize pane size horizontally and vertically

Kernel vs. User Programming

- No libc or standard headers
  - Instead, the kernel implements lots of libc-like functions
- Examples:
  - #include <string.h> → #include <linux/string.h>
  - printf("hello!\n") → printk(KERN_INFO "hello")
  - malloc(64) → kmalloc(64, GFP_KERNEL)
- Linux kernel code uses many GCC extensions
- Inline functions: "static inline void func()"
- Inline assembly: <2%
  - asm volatile("rdtsc" : "=a" (l), "=d" (h));
- Branch annotation: hint for better optimization
  - if (unlikely(error)) { ... }
  - if (likely(success)) { ... }

- No (easy) use of floating point
- Small, fixed-size stack: 8KB (2 pages) in x86
- No memory protection
  - "SIGSEGV" → kernel panic (oops)
- Example of kernel panic: here

- Synchronization and concurrency
  - preemptive multitasking → synchronization among tasks
    - » A task can be scheduled an re-scheduled at any time
  - Multi-core processor → synchronization among tasks
    - » A kernel code can execute on two more processors
  - Interrupt → synchronization with interrupt handlers
    - » can occur in the midst of execution (e.g., accessing resources)
    - » need to synchronize with interrupt handler

Linux Kenrel Coding Style

- Indentation: 1 tab → 8-character width (not 8 spaces)
- No CamelCase, use underscores: SpinLock → spin_lock
- Use C-style comments: /* use this style*/ not //
- Line length: 80 column
- Write code in a similar style with other kenrel code
  - Reference: Documentation/process/coding-style.rst

```
/*
 * a multi-lines comment
 * (no C++ '//' !)
 */

struct foo {
        int member1;
        double member2;
}; /* no typedef ! */

#ifdef CONFIG_COOL_OPTION
int cool_function(void) {
       return 42;
}
#else
int cool_function(void) { }
#endif /* CONFIG_COOL_OPTION */
```

```
void my_function(int the_param, char
*string, int a_long_parameter,
       int another_long_parameter)
{
       int x = the_param % 42;

       if (!the_param)
              do_stuff();

       switch (x % 3) {
       case 0:
              do_some_stuff();
              cool_function();
              break;
       case 1:
              /* Fall through */
       default:
              do_other_stuff();
              cool_function();
       }
}
```

Summary of Tools

- Version control: git, tig
- Configure the kernel: make oldconfig / menuconfig
- Build the kernel: make –j8; make modules –j8
- Install the kernel: make install; make modules_install
- Explore the code: make cscope tags –j8; cscope, ctags
- Editor: vim, emacs
- Screen: tmux
- The missing cs education: https://missing.csail.mit.edu/
  - watch the videos

Useful Online Resources

- The Linux Kernel Documentation: the extensive documents extracted from kernel source
- Linux Weekly News (LWN): easy explanation of recently added kernel features
- Linux Inside: textbook-style description on kernel subsystems
- Kernel newbies: useful information for new kernel developers
- Linux Kernel API Manual
- Kernel Recipes
- Kernel Planet

Next Lecture (on Thursday!)

- Isolation and system call
- Explore how the following three system calls are implemented in the kernel
  - open()
  - write()
  - fork()