

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

Building and Exploring Linux Kernel

**January 23, 2025**

**Huaicheng Li**

# Linux is a Monolithic Kernel

- A traditional design: all of the OS runs in kernel, privileged mode
  - share the same address space
- Kernel interface  $\sim$  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

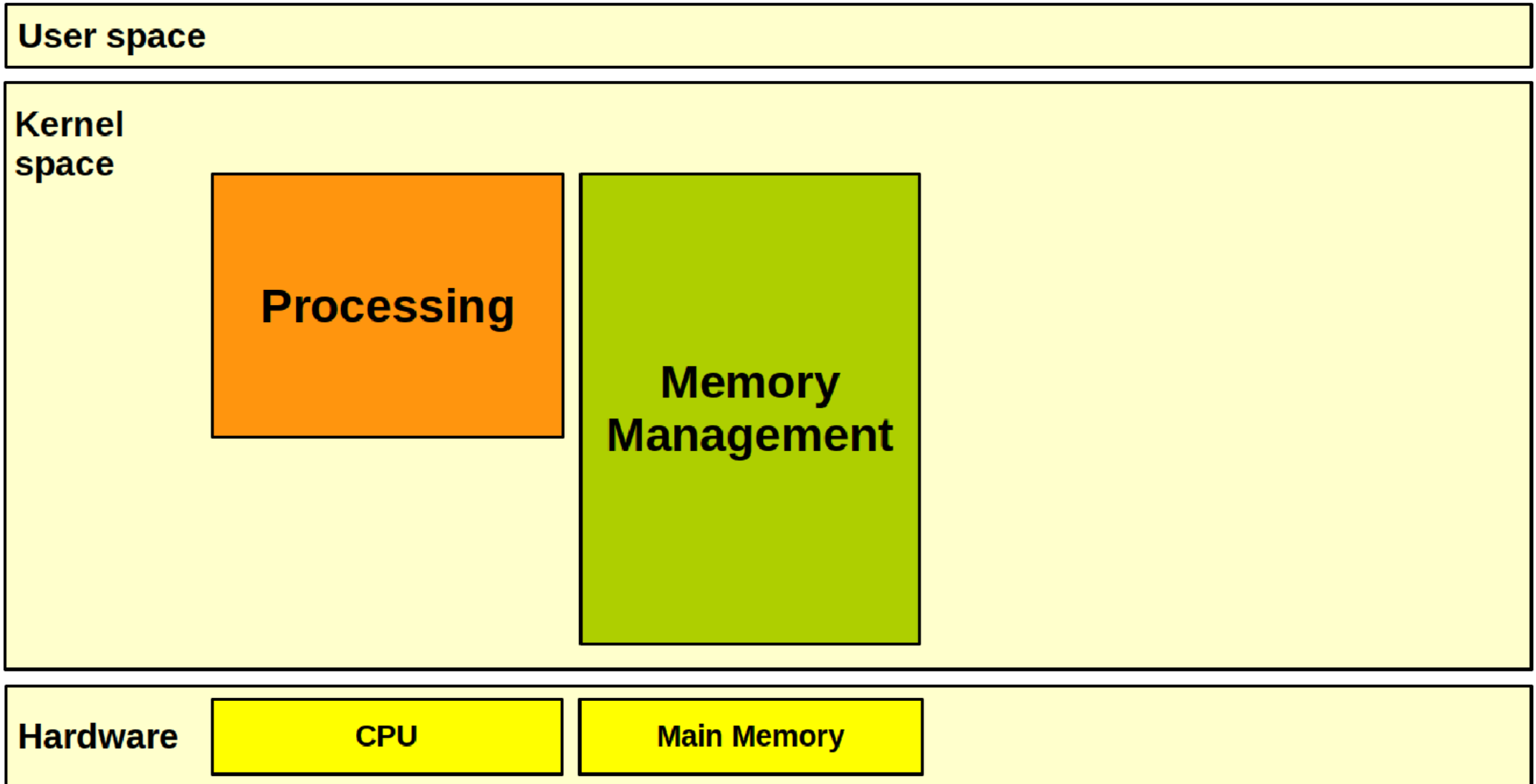
# Alternative: Microkernel Design

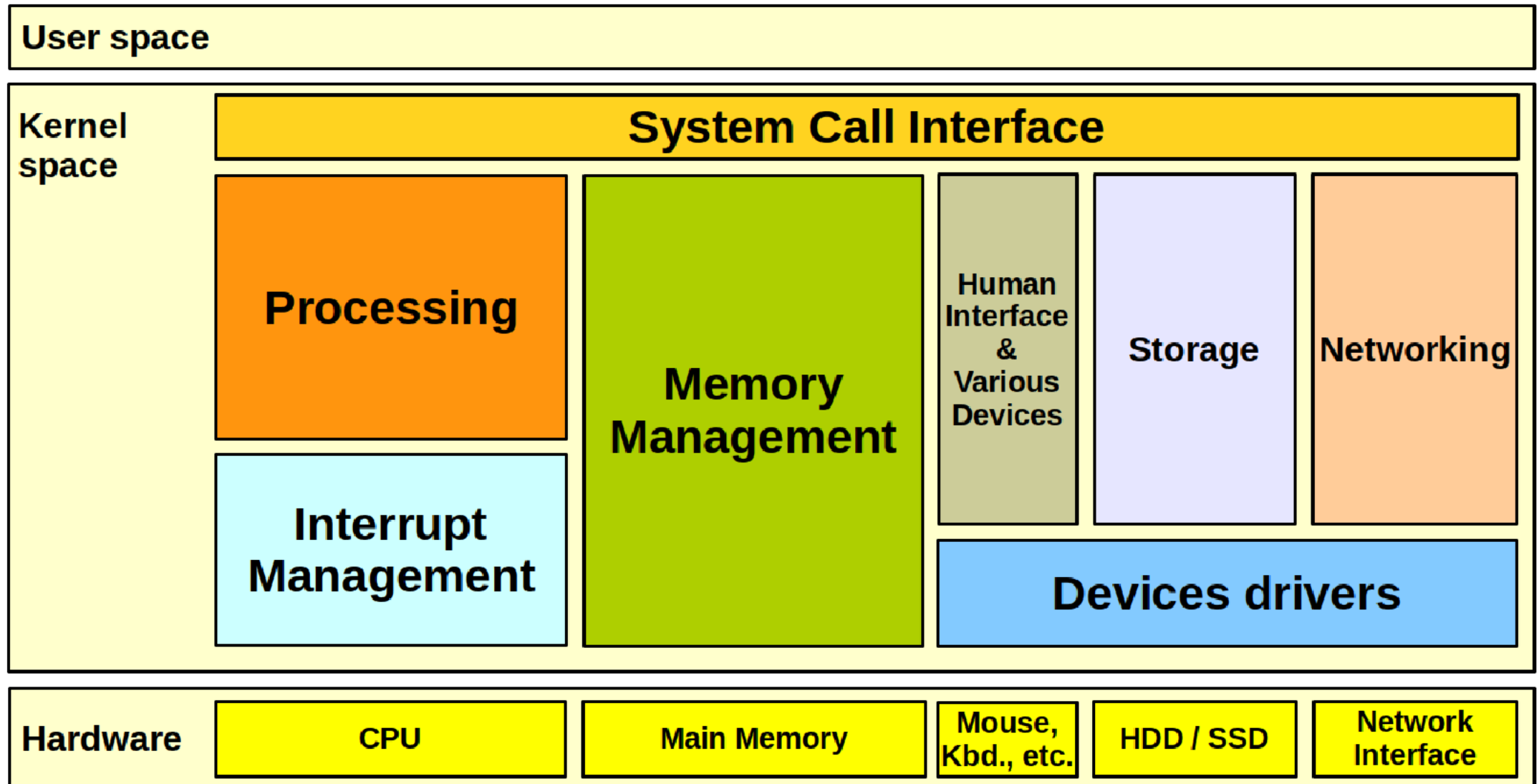
- Many OS services run as ordinary user programs
  - e.g., file system in a file server
- Kernel implements minimal mechanism to run services in user space
  - IPC, virtual memory, threads
- Kernel interface  $\neq$  system call interface
  - applications talk to servers via IPCs
- Good: more isolation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

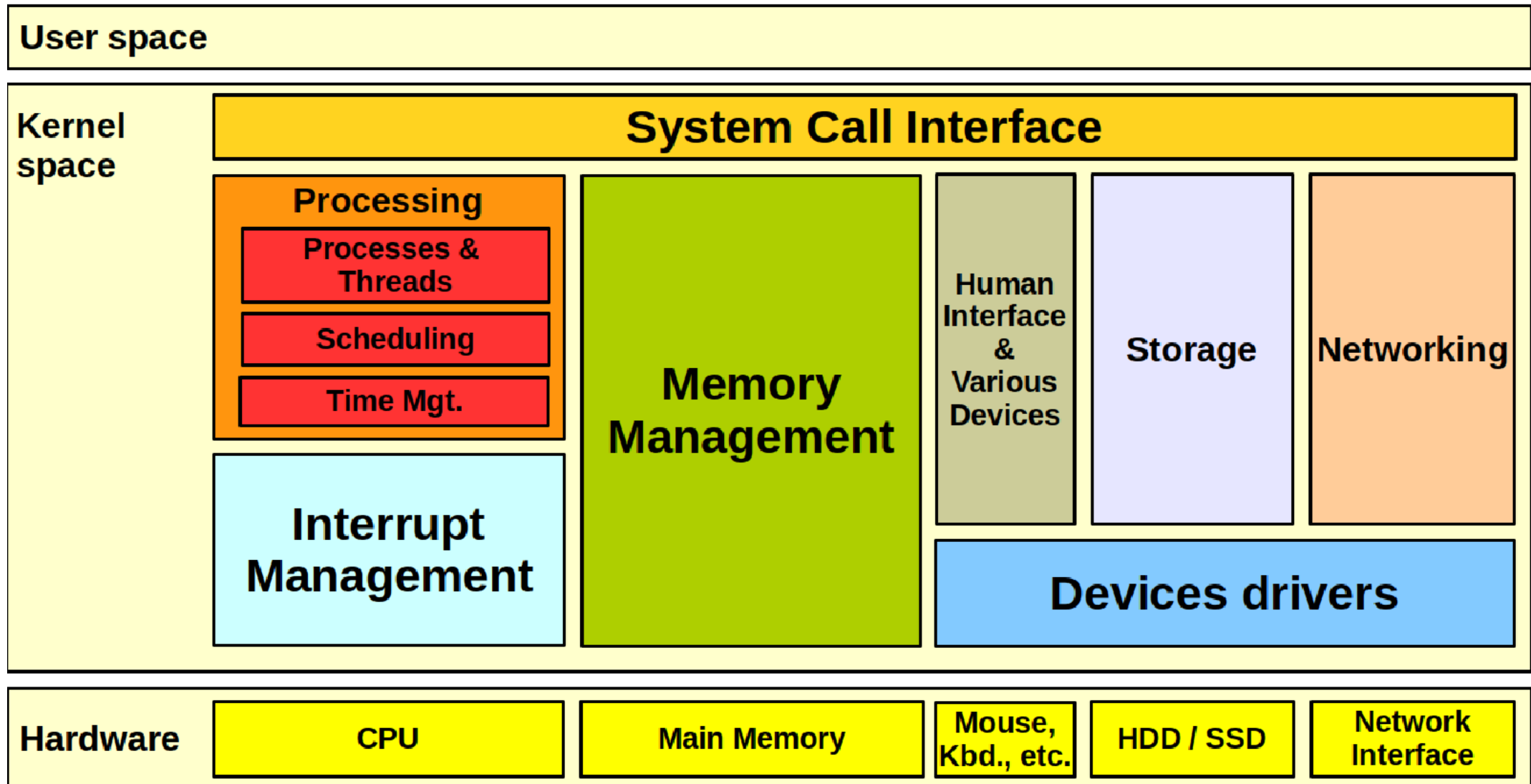
# Debate

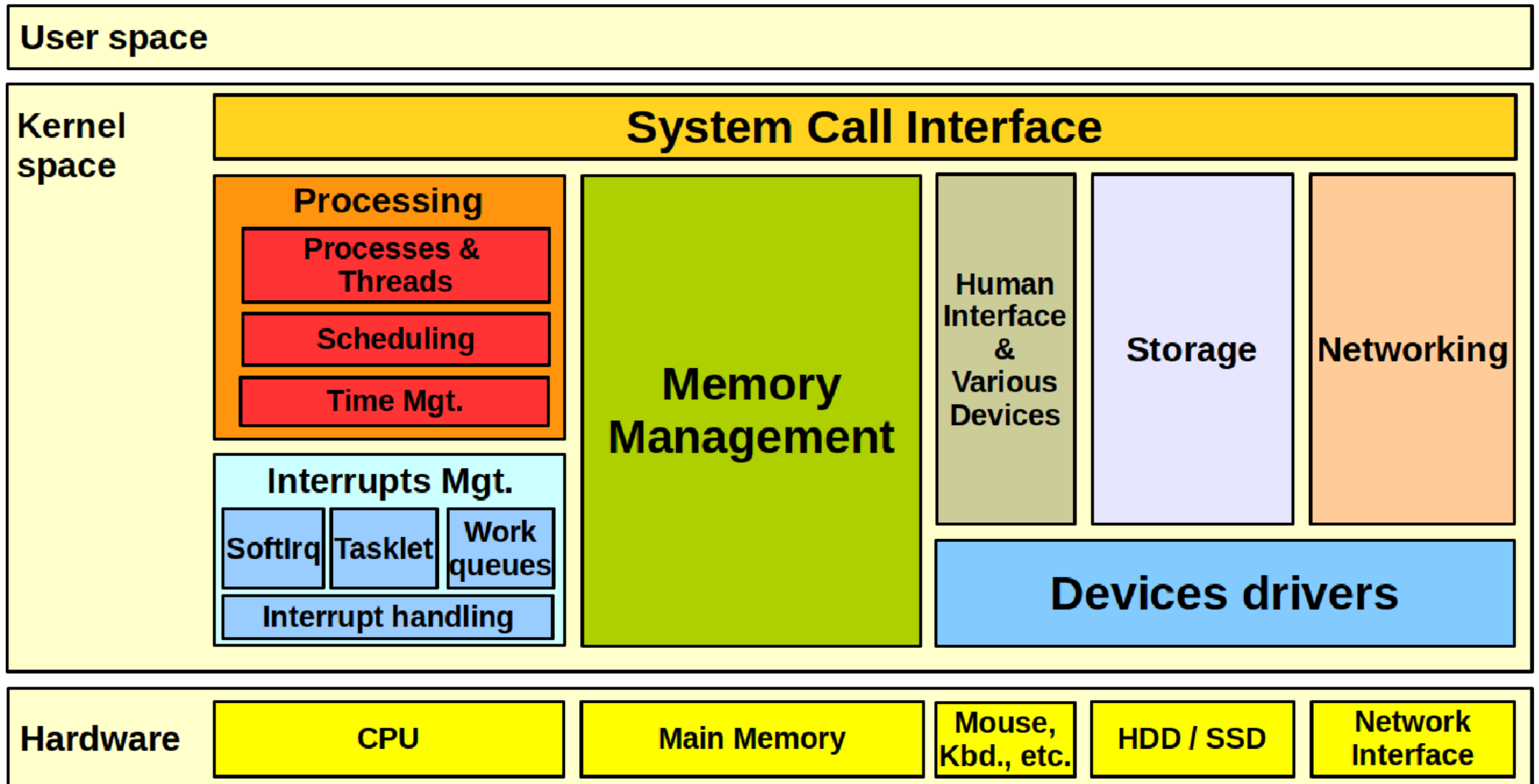
- Tanenbaum-Torvalds debate
- Most real-world kernels are mixed: Linux, OS X, Windows
  - e.g., X Window Systems

# Kernel and Course Map

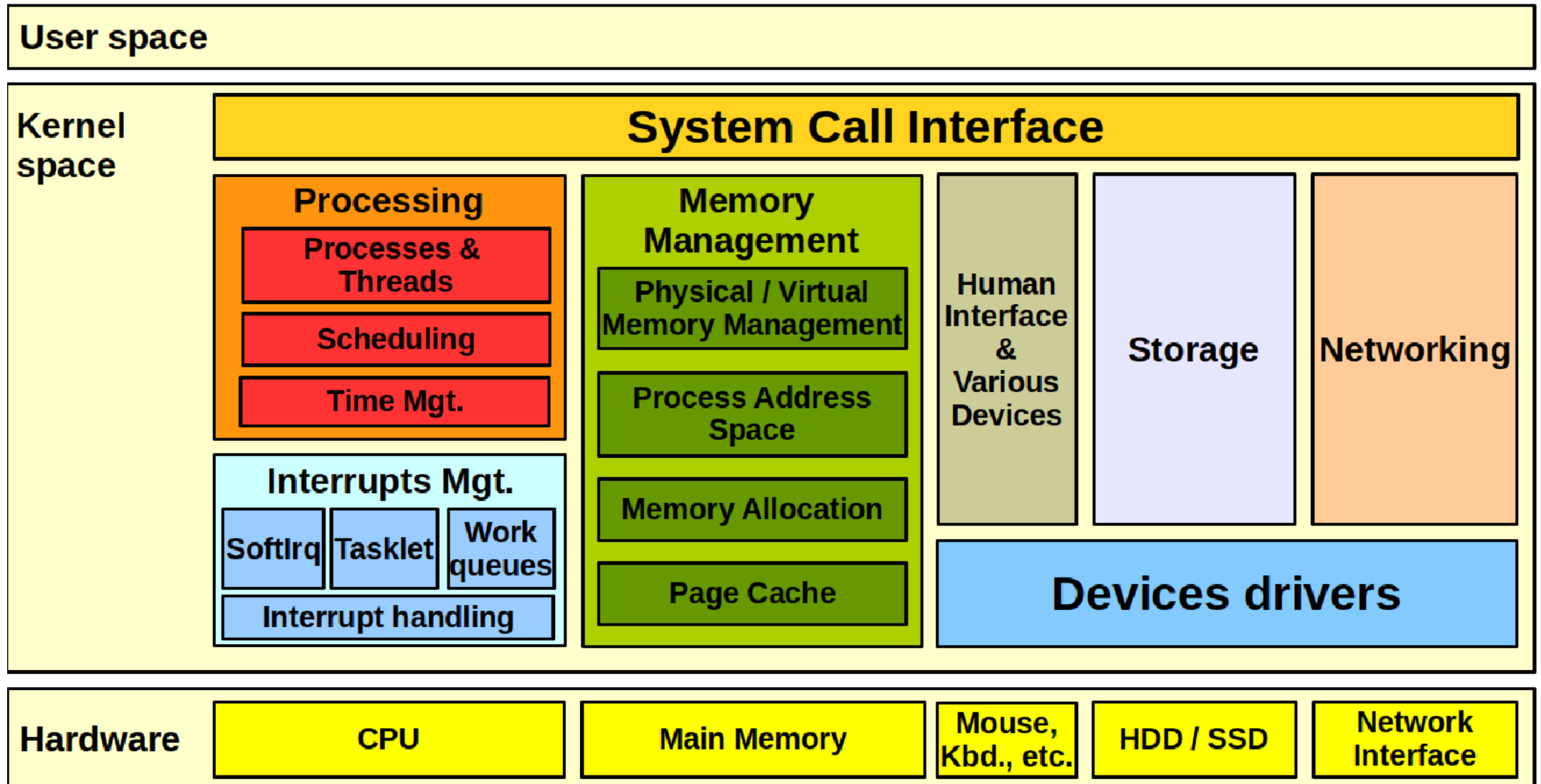


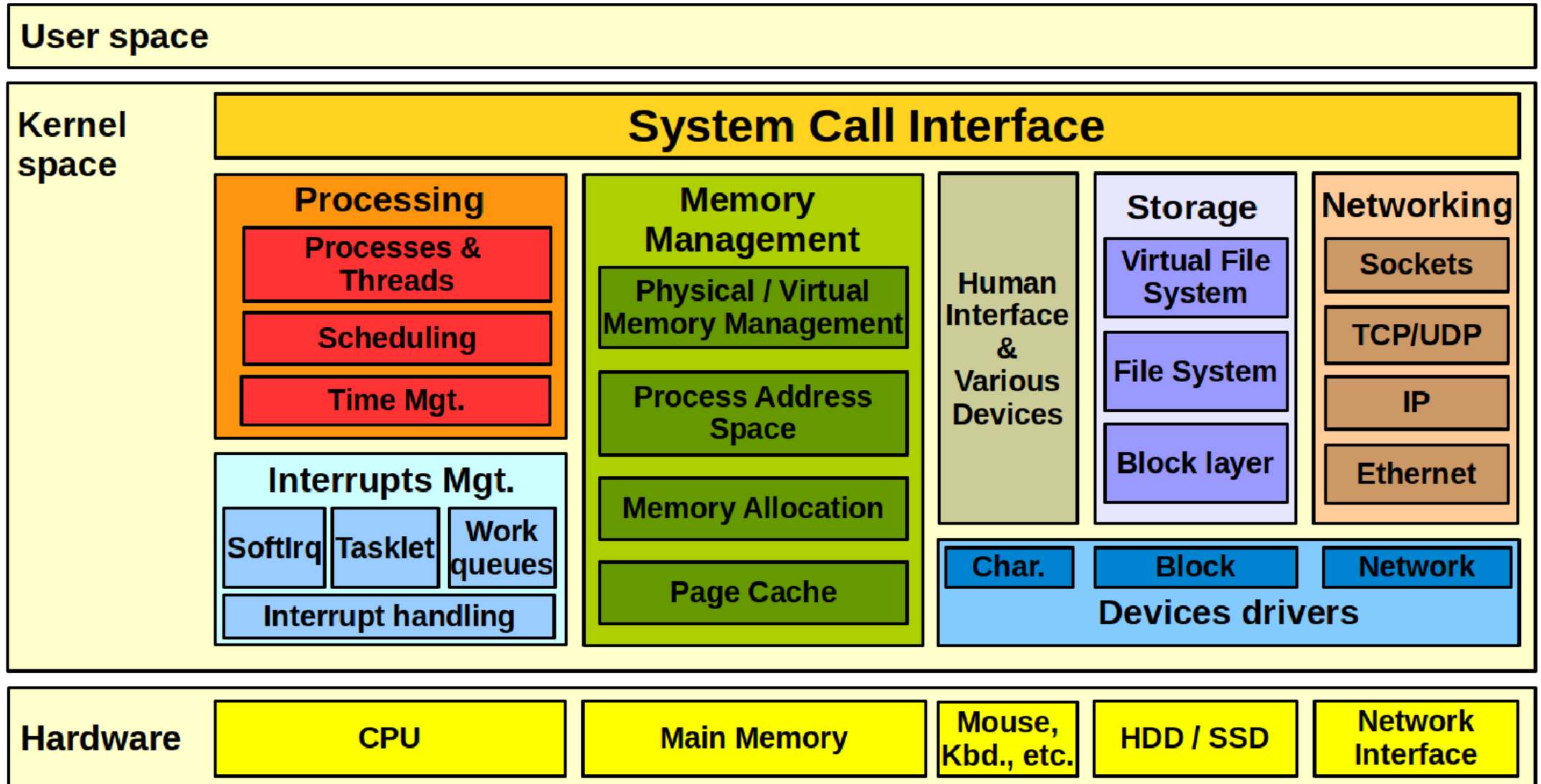


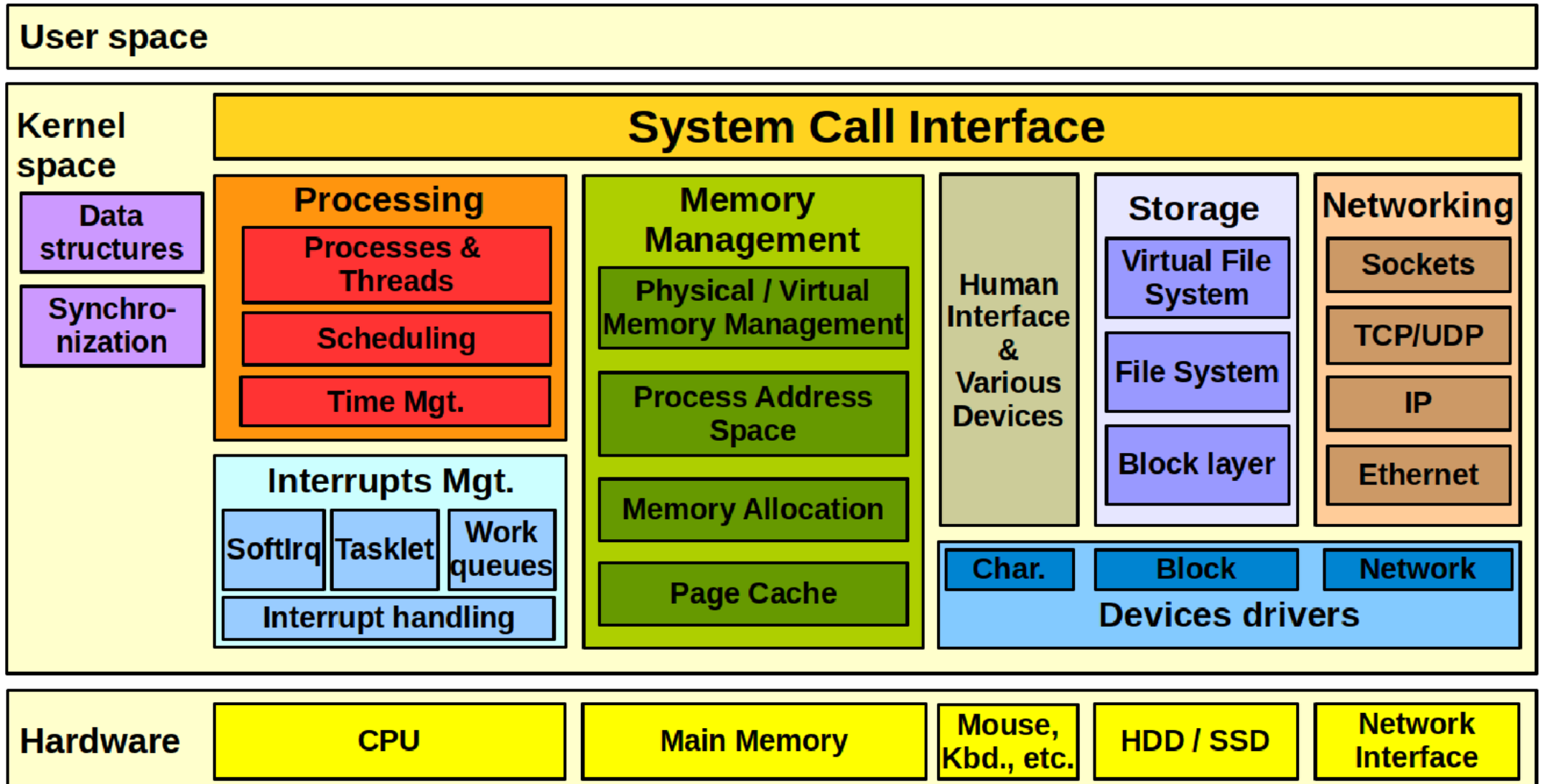


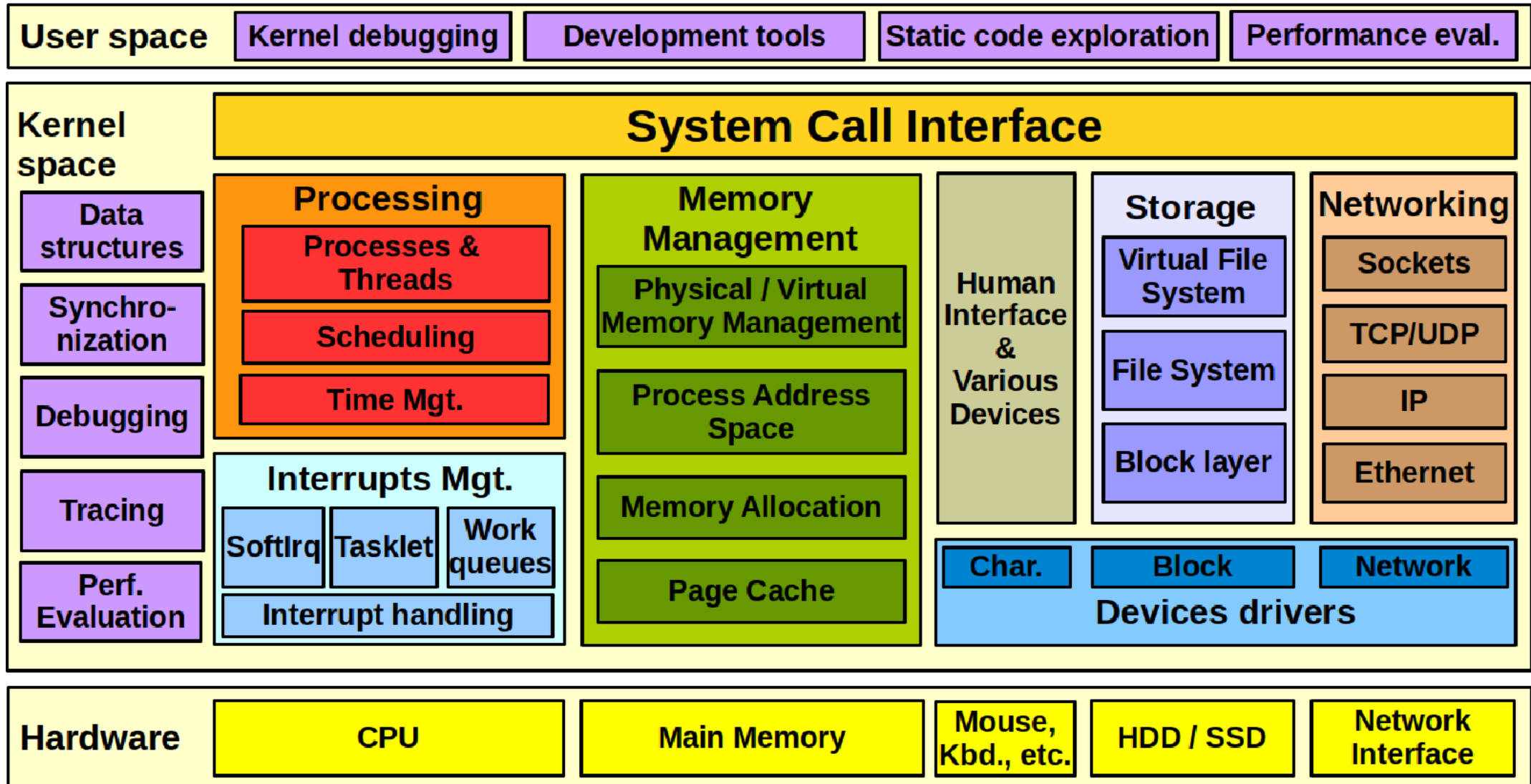












# Today's Focus

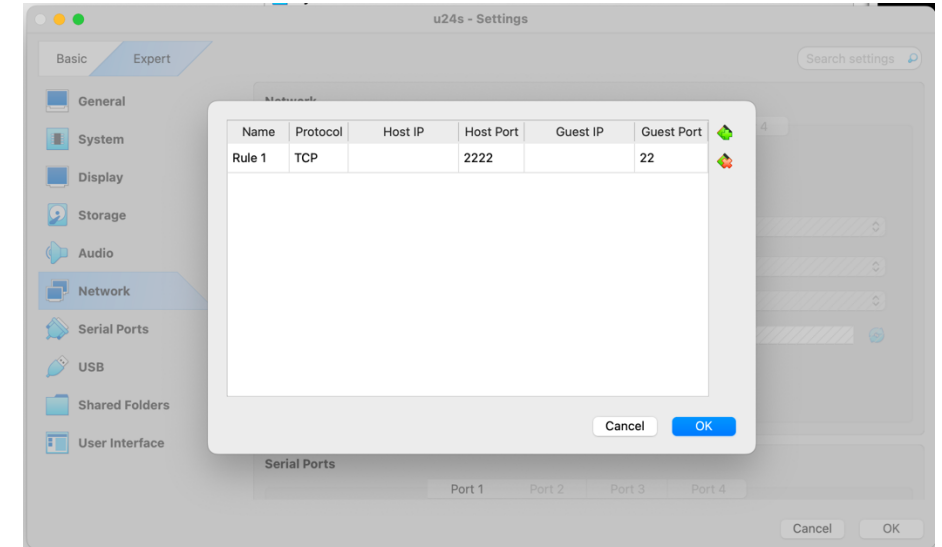
- Tools
  - Version control: git, tig
  - Configure, build, and install the kernel: make
  - Explore the code: cscope, ctags
  - Editor: vim, emacs
  - Screen multiplexing: tmux
- Kernel vs. user programming

# 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
```

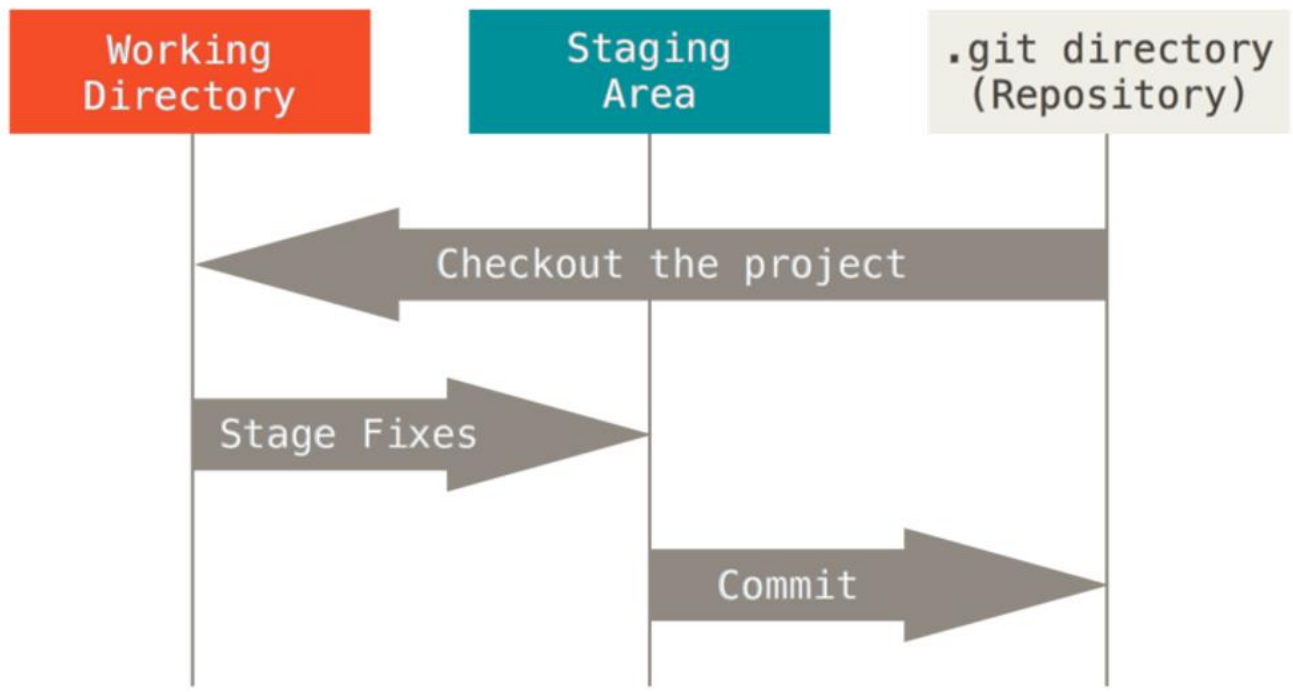
## # 6. publish and update

```
$ 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

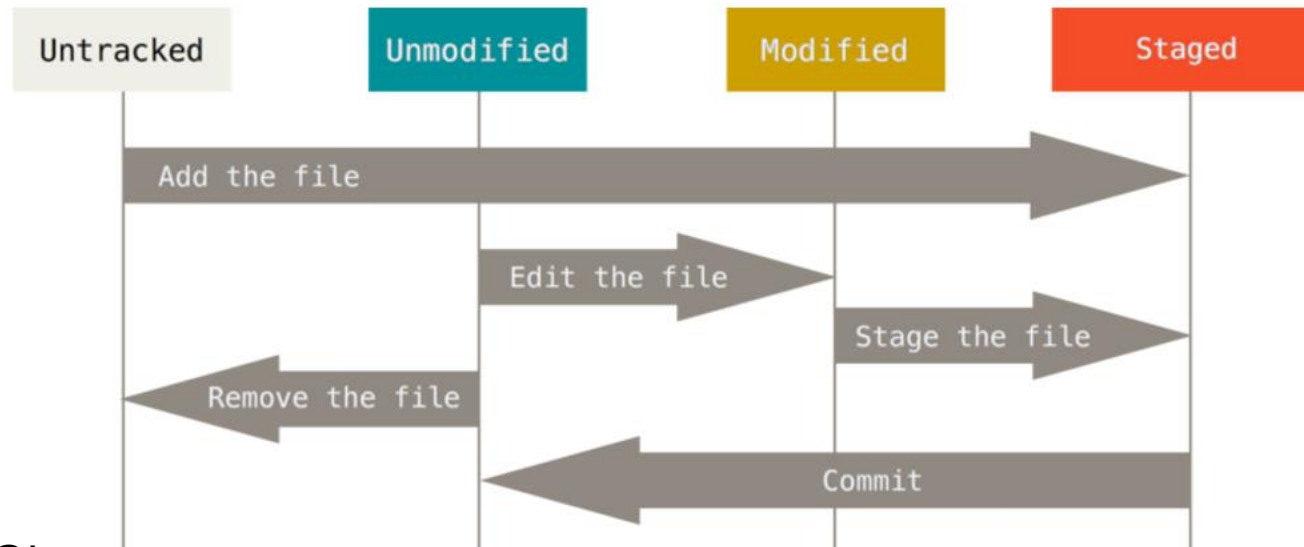
- `git add $a-new-file`
- `git add $an-existing-git-managed-file`
- `git rm $an-existing-git-managed-file`
- `git commit`

*# Untracked -> Staged*

*# Modified -> Staged*

*# Unmodified -> Untracked*

*# 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/READMErst](#)**



# 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, just choose the default options
- `make localmodconfig`
  - Disable kernel features and drivers 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

```
.config - Linux/x86 6.12.0 Kernel Configuration

Linux/x86 6.12.0 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters
are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?>
for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

  General setup --->
[*] 64-bit kernel
    Processor type and features --->
[*] Mitigations for CPU vulnerabilities --->
    Power management and ACPI options --->
    Bus options (PCI etc.) --->
    Binary Emulations --->
[*] Virtualization --->
    General architecture-dependent options --->
[*] Enable loadable module support --->
-* Enable the block layer --->
    Executable file formats --->
    Memory Management options --->
[*] Networking support --->
    Device Drivers --->
    File systems --->
    Security options --->
-* Cryptographic API --->
    Library routines --->
↓(+)
```

```
< elect>  < Exit >  < Help >  < Save >  < Load >
```

# 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 identifier 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_illlocked    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

```

Z q/l/i/main.c
debug_kernel(c      867 }
do_basic_setup     868
doctors(void)      869
asmlinkage __visible __init __no_sanitize_address __noreturn __no_stack_protector
do_early_param     870 void start_kernel(void)
do_initcall_le     871 {
do_initcalls(v     872 char *command_line;
do_one_initcall   873 char *after_dashes;
do_pre_smp_init   874
do_trace_init     875 set_task_stack_end_magic(&init_task);
do_trace_init     876 smp_setup_processor_id();
early_randomiz    877 debug_objects_early_init();
exit_boot_conf    878 init_vmlinux_build_id();
free_initmem(v    879
get_boot_conf_i   880 cgroup_init_early();
get_boot_conf_i   881
ignore_unknown    882 local_irq_disable();
init_setup(char)  883 early_boot_irqs_disabled = true;
initcall_black    884
initcall_black    885 /*
initcall_black    886 * Interrupts are still disabled. Do necessary setups, then
initcall_black    887 * enable them.
initcall_debug    888 */
initcall_debug    889 boot_cpu_init();
kernel_init(vo    890 page_address_init();
kernel_init_fr    891 pr_notice("%s", linux_banner);
loglevel(char)   892 early_security_init();
mark_readonly()  893 setup_arch(&command_line);
mark_readonly()  894 setup_boot_config();
mark_readonly()  895 setup_command_line(command_line);
obsolete_check    896 setup_nr_cpu_ids();
parse_early_op    897 setup_per_cpu_areas();
parse_early_pa    898 smp_prepare_boot_cpu(); /* arch-specific boot-cpu hooks */
pgtable_cache_   899 boot_cpu_hotplug_init();
poking_init(vo   900
print_unknown_   901 pr_notice("Kernel command line: %s\n", saved_command_line);
quiet_kernel(c   902 /* parameters may set static keys */
rdinit_setup(c   903 jump_label_init();
repair_env_str    904 parse_early_param();
res_init(void)   905 after_dashes = parse_args("booting kernel",
run_init_proce   906 static_command_line, __start__param,
set_debug_rod    907 __stop__param - __start__param,
set_init_arg(c   908 -1, -1, NULL, &unknown_bootoption);
set_reset_devi   909 print_unknown_bootoptions();
setup_boot_con   910 if (!IS_ERR_OR_NULL(after_dashes))
setup_boot_con   911 parse_args("Setting init args", after_dashes, NULL, 0, -1, -1,
setup_command_   912 NULL, set_init_arg);
setup_nr_cpu_i   913 if (extra_init_args)
smp_prepare_cp   914 parse_args("Setting extra init args", extra_init_arg,
smp_setup_pro   915 NULL, 0, -1, -1, NULL, set_init_arg);
start_kernel(v   916
thread_stack_c   917 /* Architectural and non-timekeeping rng init, before allocator init */
trace_initcall  918 random_init_early(command_line);
trace_initcall  919
trap_init(voi    920 /*
try_to_run_ini   921 * These use large bootmem allocations and must precede
unknown_bootop  922 * initialization of page allocator
warn_bootconfi  923 */
[Name] main.c git/linux/init/main.c 877,2-5 57%
[10] 0:top*
Processes: 744 total, 3 running, 741 sleeping, 4193 threads
Load Avg: 4.60, 4.95, 5.17 CPU usage: 21.44% user, 20.97% sys, 57.58% idle
SharedLibs: 1314M resident, 201M data, 197M linkedit. MemRegions: 0 total, 0B resident, 0B private, 13G shared.
PhysMem: 45G used (3089M wired, 0B compressor), 50G unused.
VM: 292T vsz, 5430M framework vsz, 0(0) swpins, 0(0) swapouts.
Networks: packets: 4490986/5961M in, 389759/54M out. Disks: 2607238/27G read, 558319/20G written.
PID COMMAND %CPU TIME #TH #WQ #PORT MEM PURG CMPR PGRP PPID STATE BOOSTS %CPU_ME
1255 VirtualBoxWM 449.7 12:56:02 45/1 1 588 16G 35M 0B 1255 1116 running *2[8] 0.00000
377 WindowServer 23.3 28:52.39 28 6 4702 817M+ 66M 0B 377 1 sleeping *0[1] 0.05083
1258 iTerm2 7.7 12:53.51 10 7 486+ 585M 18M 0B 1258 1 sleeping *0[897] 0.09255
0 kernel_task 6.1 15:31.19 772/12 0 0 15M 0B 0B 0 0 running 0[0] 0.00000
91439 top 5.5 00:02.24 1/1 0 30 12M 0B 0B 91439 90816 running *0[1] 0.00000
92521 screencaptur 3.5 00:00.28 2 1 68- 8866K- 752K 0B 594 594 sleeping *0[496+] 0.09217
2532 Microsoft Po 2.7 14:08.46 40 8 1062+ 549M 296M 0B 2532 1 sleeping *731531+[67] 0.00000
8057 VTDecoderXPC 2.2 03:42.77 5 2 86 13M 0B B 77988057 1 sleeping *373100+[2] 0.3600000
22112 searchpartyu 1.2 00:18.27 7 5 225 11M 96K 0B 22112 1 sleeping *4[660] 0.00000
2478 plugin-conta 1.2 02:55.05 31 1 128 582M 0B 0B 2474 2474 sleeping *0[2] 0.00000
2474 firefox 0.9 08:51.21 93 4 804 1499M+ 77M 0B 2474 1 sleeping *0[980] 0.00000
6135 imogent 0.7 00:07.26 4 3 233 12M 208K 0B 715 1 sleeping *0[1] 0.06369
2529 plugin-conta 0.7 00:14.18 31 1 133 363M 0B 0B 2474 2474 sleeping *0[2] 0.00000
369 bluetoothd 0.4 01:22.33 12 6 414 13M 224K 0B 369 1 sleeping *0[1] 0.46690
667 shardingd 0.4 00:24.20 3 2 376+ 13M+ 0B 0B 667 1 sleeping *0[1] 0.57932
41695 screencaptur 0.3 00:07.70 3 1 207 20M 688K 0B 41695 1 sleeping *0[1274] 0.00000
1116 VBoxSVC 0.3 00:58.29 19 2 103 23M 0B 0B 1116 1 sleeping *0[1] 0.00000
308 logd 0.3 00:31.52 4 3 2379 7457K 0B 0B 308 1 sleeping *0[1] 0.00000
1 launchd 0.3 01:10.55 3 2 3719- 14M- 0B 0B 1 0 sleeping 0[0] 0.00000
653 nearbyd 0.2 00:18.08 8 6 100 4417K 0B 0B 653 1 sleeping *3[2] 0.00000
589 rapportd 0.2 00:04.23 2 1 249 5825K 0B 0B 589 1 sleeping *0[1] 0.00000
516 searchpartyd 0.2 00:40.39 7 6 102 6001K 992K 0B 516 1 sleeping *15503+[4] 0.00000
811 com.apple.Am 0.1 15:723.70 3 1 79 4257K 0B 0B 811 1 sleeping *1[1] 0.07618
621 fileprovider 0.1 00:22.99 7 6 155 14M 3648K 0B 621 1 sleeping *170[1] 0.00000
662 coreauthd 0.1 15:700.26 2 3 1 56 3633K 0B 8K 07798662 1 sleeping *0[824+] 0.00000
372 corebrightne 0.1 00:17.12 4 3 125 4449K 0B 0B 372 1 sleeping *0[1] 0.08466
2514 plugin-conta 0.1 01:40.98 32 1 138 546M 0B 0B 2474 2474 sleeping *0[2] 0.00000
1114 VirtualBox 0.1 00:59.96 15 1 336 205M 0B 0B 1114 1 sleeping *0[1392] 0.00000
57337 plugin-conta 0.1 00:07.66 29 1 125 126M 0B 0B 2474 2474 sleeping *0[2] 0.00000
354 locationd 0.0 00:39.42 4 2 273 6801K 480K 0B 354 1 sleeping *0[17081+] 0.00000
2489 plugin-conta 0.0 01:09.09 34 1 134 310M 0B 0B 2474 2474 sleeping *0[2] 0.00000
2513 plugin-conta 0.0 00:39.52 31 1 140 480M 0B 0B 2474 2474 sleeping *0[2] 0.00000
362 PerfPowerSer 0.0 00:32.50 8 5 633 9233K 384K 0B 362 1 sleeping 0[1254] 0.00000
321 powerd 0.0 00:11.22 3 2 146 4193K 0B 0B 321 1 sleeping *0[1] 0.00000
652 contactsd 0.0 02:15.98 3 2 931+ 8209K+ 16M 0B 652 1 sleeping *0[193024+] 0.00000
2503 plugin-conta 0.0 01:11.16 35 1 134 313M 0B 0B 2474 2474 sleeping *0[2] 0.00000
48201 plugin-conta 0.0 00:13.82 29 1 124 227M 0B 0B 2474 2474 sleeping *0[2] 0.00000
6454 plugin-conta 0.0 00:16.40 30 1 132 280M 0B 0B 2474 2474 sleeping *0[2] 0.00000
568 distnoted 0.0 00:08.86 3 2 459 2737K 0B 0B 568 1 sleeping *0[1] 0.00000
2784 tmux 0.0 00:11.88 1 0 20 7138K 0B 0B 2784 1 sleeping *0[1] 0.00000
2499 plugin-conta 0.0 01:04.17 34 1 133 317M 0B 0B 2474 2474 sleeping *0[2] 0.00000
511 audioclocksy 0.0 00:08.05 4 3 49 5937K 0B 0B 511 1 sleeping *0[1] 0.00000
545 com.apple.Dr 0.0 01:12.10 6 4 1476 31M 0B 0B 545 1 sleeping 0[1] 0.00000
42813 plugin-conta 0.0 00:18.67 29 1 129- 129M- 0B 0B 2474 2474 sleeping *0[2] 0.00000
810 Magnet 0.0 01:37.91 5 3 212- 38M- 0B- 0B 810 1 sleeping *0[2228] 0.00223
2509 plugin-conta 0.0 00:28.53 30 1 136 373M 0B 0B 2474 2474 sleeping *0[2] 0.00000
29217 plugin-conta 0.0 00:13.29 29 1 129 245M 0B 0B 2474 2474 sleeping *0[2] 0.00000
857 Hidden Bar 0.0 00:07.93 6 4 188- 39M- 60M- 0B 857 1 sleeping *0[689] 0.00309
18617 plugin-conta 0.0 00:07.51 29 1 122 106M 0B 0B 2474 2474 sleeping *0[2] 0.00000
77981 -Q-X-S-\ 0.0 0 15:722.50 8 3 503- 64M 22M- 0B 77981 1 sleeping *0[645] 0.00892
2507 ~plugin-conta 0.0 00:05.05 30 1 124- 226M- 0B 0B 2474 2474 sleeping *0[2] 0.00000
18708 plugin-conta 0.0 00:08.77 29 1 130 179M 0B 0B 2474 2474 sleeping *0[2] 0.00000
[10] 0:top*
pos2_local" 14:19 23-Jan-25

```

# Basic tmux commands

- `tmux`: start a new tmux session
  - or “`tmux new-session -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 and 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 Kernel 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 kernel 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()`