Advanced C programming

Agenda

- VFIO
- OOM Killer
- JTAG Debugger
- DPDK
- GIT

VFIO Driver

What is VFIO?

- A new user level driver framework for Linux
- **V**irtual **F**unction **I**/**O**[*]
- Originally developed by Tom Lyon (Cisco)
- IOMMU-based DMA and interrupt isolation
- Full devices access (MMIO, I/O port, PCI config)
- Efficient interrupt mechanisms
- Modular IOMMU and device backends

VFIO Driver

- VFIO is a device driver
  - supports modular device driver backends
  - vfio-pci binds to non-bridge PCI devices
  - pci-stub available as “no access” driver
    - Allows admins to restrict access within a group
  - Users cannot attempt to use in-service host devices
  - Devices in use by users cannot be simultaneously claimed by other host drivers

VFIO Driver

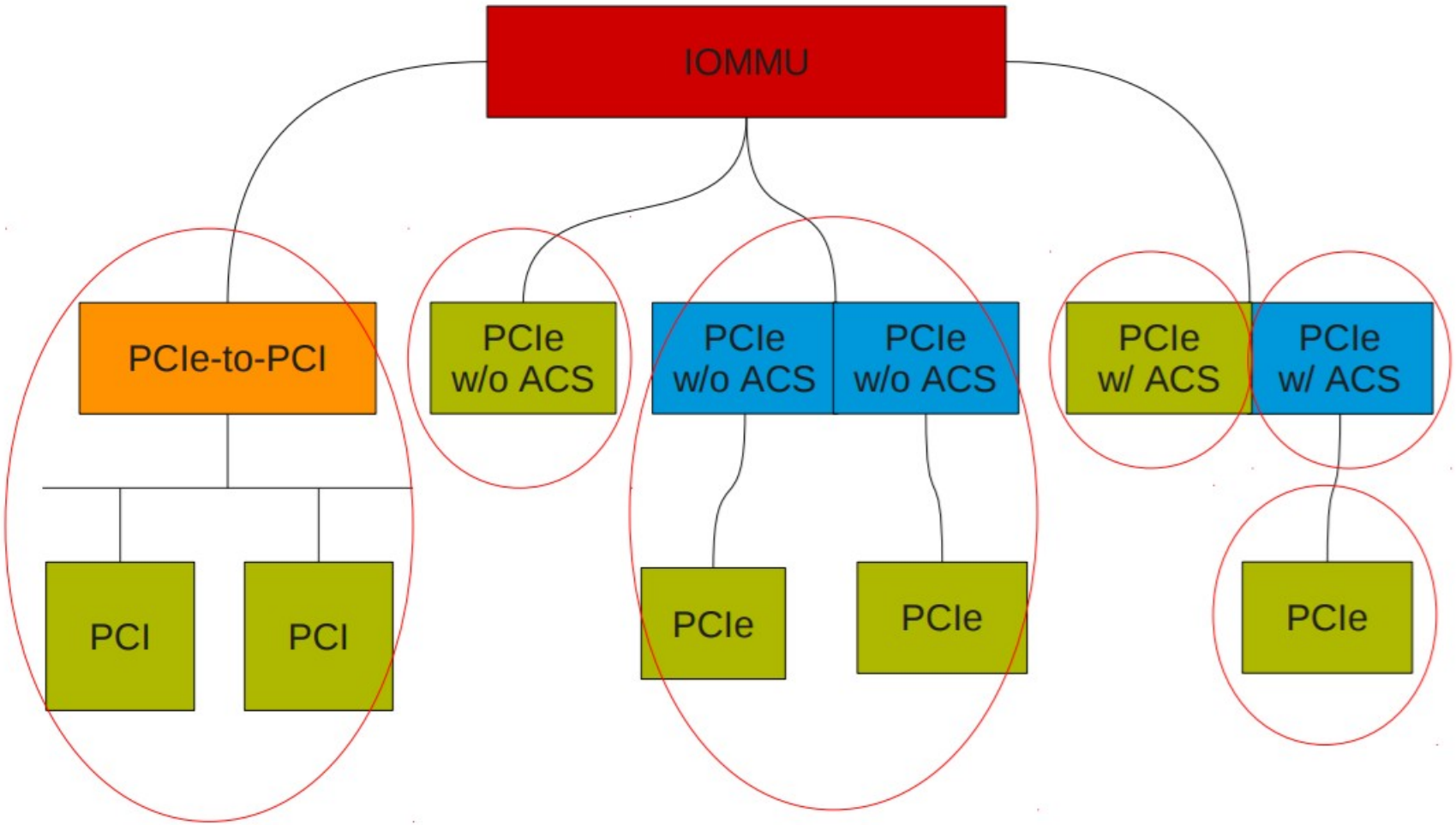
- VFIO device backends provide secure resource access
  - No device access without IOMMU isolation
  - Integral to the interface
    - Not outsourced to pci-sysfs
  - Virtualized access to PCI config space

VFIO Driver

- VFIO uses IOMMU groups
  - Allows the IOMMU driver to define both visibility and containment
  - Solves devices hidden by bridges
    - IOMMU cannot differentiate devices behind PCI bridge
  - Solves peer-to-peer back channels
    - All transactions required to reach IOMMU for translation
    - For PCIe, ACS (Access Control Services) indicates support
  - Result is better **security**

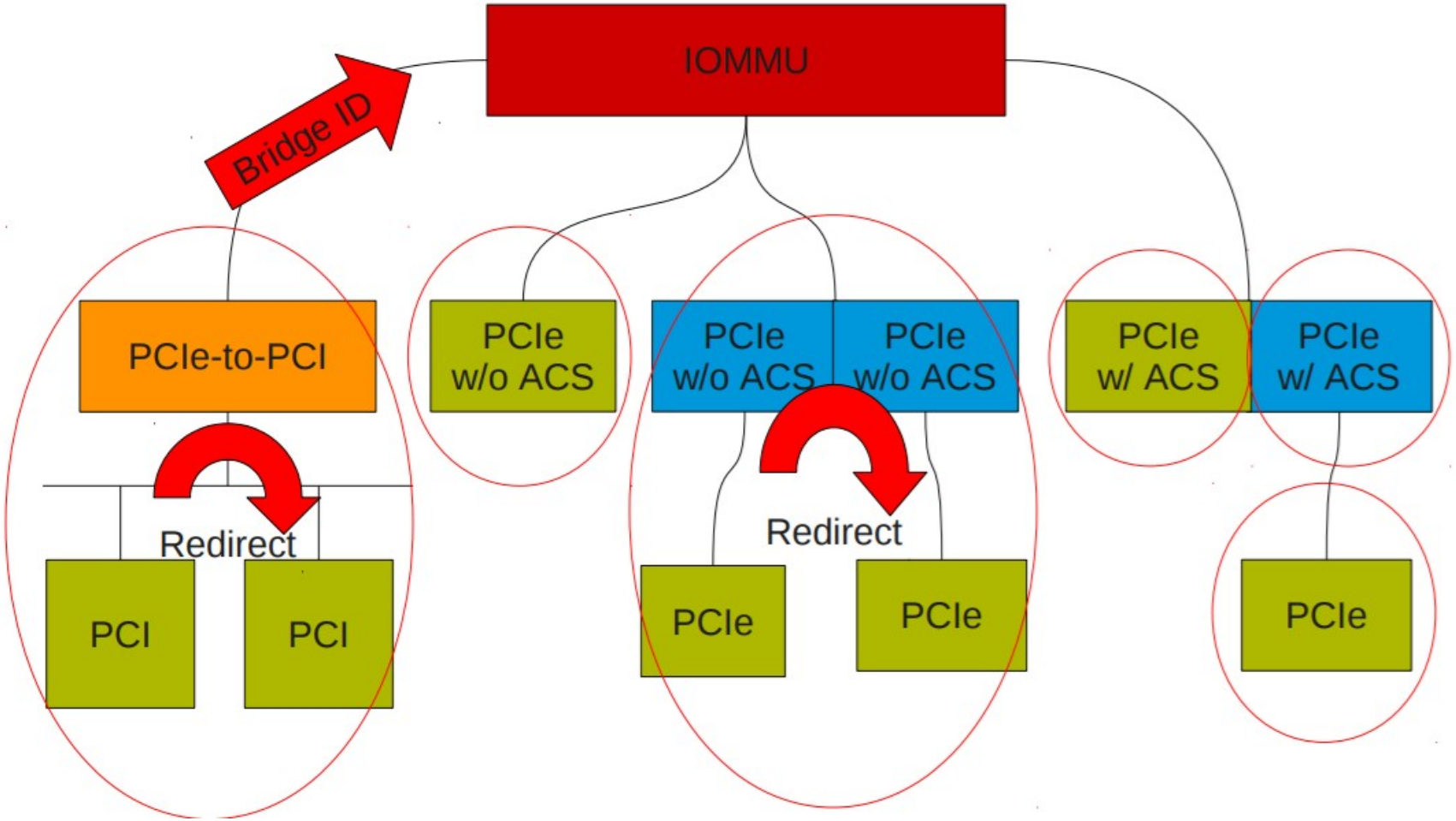
VFIO Driver

IOMMU Group examples



VFIO Driver

IOMMU Group examples



VFIO Driver

- VFIO moves ownership to the group level
  - Access to device file grants ownership
  - Ownership extends to all devices within the group
  - All accesses through VFIO
- VFIO supports a modular IOMMU interface
  - IOMMU API (type1) implemented
  - POWER (SPAPR) under development
- VFIO supports a modular device interface
  - PCI (vfio-pci) implemented
- VFIO has no KVM dependencies
  - Used only for acceleration
  - Non-x86 guests on x86 host work today
    - ppc g3beige – Big Endian driver test platform!
    - Any guest platform with PCI support

VFIO Driver

- VFIO moves ownership to the group level
  - Access to device file grants ownership
  - Ownership extends to all devices within the group
  - All accesses through VFIO
- VFIO supports a modular IOMMU interface
  - IOMMU API (type1) implemented
  - POWER (SPAPR) under development
- VFIO supports a modular device interface
  - PCI (vfio-pci) implemented
- VFIO has no KVM dependencies
  - Used only for acceleration
  - Non-x86 guests on x86 host work today
    - ppc g3beige – Big Endian driver test platform!
    - Any guest platform with PCI support

VFIO Driver

- AMD-Vi or Intel VT-d capable hardware
- Linux 3.6+ host
  - CONFIG_VFIO_IOMMU_TYPE1=m
  - CONFIG_VFIO=m
  - CONFIG_VFIO_PCI=m
  - modprobe vfio-pci

VFIO Driver

- Device to assign:

```
01:10.0 Ethernet controller: Intel Corporation 82576
Virtual Function (rev 01)
```

- Find the group:

```
$ readlink /sys/bus/pci/devices/0000:01:10.0/iommu_group
../../../../kernel/iommu_groups/15
```

- IOMMU Group = 15
- Check the devices in the group:

```
$ ls /sys/bus/pci/devices/0000:01:10.0/iommu_group/devices/
0000:01:10.0
```

VFIO Driver

- Unbind from device driver

```
$ echo 0000:01:10.0 | sudo tee \
/sys/bus/pci/devices/0000:01:10.0/driver/unbind
```

- Find vendor & device ID

```
$ lspci -n -s 01:10.0
01:10.0 0200: 8086:10ca (rev 01)
```

- Bind to vfio-pci

```
$ echo 8086 10ca | sudo tee \
/sys/bus/pci/drivers/vfio-pci/new_id
```

- Check

```
$ ls /dev/vfio
15  vfio
```

VFIO Driver

- Unbind from device driver

```
$ echo 0000:01:10.0 | sudo tee \
/sys/bus/pci/devices/0000:01:10.0/driver/unbind
```

- Find vendor & device ID

```
$ lspci -n -s 01:10.0
01:10.0 0200: 8086:10ca (rev 01)
```

- Bind to vfio-pci

```
$ echo 8086 10ca | sudo tee \
/sys/bus/pci/drivers/vfio-pci/new_id
```

- Check

```
$ ls /dev/vfio
15  vfio
```

VFIO Driver

- The primary difference between uio and vfio is that vfio is capable of programming the platform's IOMMU, which is a critical piece of hardware for ensuring memory safety in user space drivers.
- In order for SPDK to take control of a device, it must first instruct the operating system to relinquish control. This is often referred to as unbinding the kernel driver from the device and on Linux is done by writing to a file in sysfs. SPDK then rebinds the driver to one of two special device drivers that come bundled with Linux - uio or vfio. These two drivers are "dummy" drivers in the sense that they mostly indicate to the operating system that the device has a driver bound to it so it won't automatically try to re-bind the default driver. They don't actually initialize the hardware in any way, nor do they even understand what type of device it is.

OOM Killer

- The Linux kernel allocates memory upon the demand of the applications running on the system. Because many applications allocate their memory up front and often don't utilize the memory allocated, the kernel was designed with the ability to over-commit memory to make memory usage more efficient. This over-commit model allows the kernel to allocate more memory than it actually has physically available. If a process actually utilizes the memory it was allocated, the kernel then provides these resources to the application. When too many applications start utilizing the memory they were allocated, the over-commit model sometimes becomes problematic and the kernel must start killing processes in order to stay operational. The mechanism the kernel uses to recover memory on the system is referred to as the out-of-memory killer or OOM killer for short.

OOM Killer

- When troubleshooting an issue where an application has been killed by the OOM killer, there are several clues that might shed light on how and why the process was killed. In the following example, we are going to take a look at our syslog to see whether we can locate the source of our problem. The oracle process was killed by the OOM killer because of an out-of-memory condition. The capital K in Killed indicates that the process was killed with a -9 signal, and this is usually a good sign that the OOM killer might be the culprit.

```
grep -i kill /var/log/messages*
host kernel: Out of Memory: Killed process 2592 (oracle).
```

OOM Killer

- We can also examine the status of low and high memory usage on a system. It's important to note that these values are real time and change depending on the system workload; therefore, these should be watched frequently before memory pressure occurs. Looking at these values after a process was killed won't be very insightful and, thus, can't really help in investigating OOM issues.

```
[root@test-sys1 ~]# free -lm
             total       used       free     shared    buffers     cached
Mem:           498         93        405          0         15         32
Low:           498         93        405
High:            0          0          0
-/+ buffers/cache:         44        453
Swap:         1023          0       1023
```

OOM Killer

- we are using the vmstat command to look at our resources every 1 seconds 100 times. The -S switch shows our data in a table and the -M switch shows the output in megabytes to make it easier to read.

```
test@test-VirtualBox:/media/sf_shared/git/AdvancedC/OOM$ vmstat -SM 1 100
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0    827    444      4    128    0    0    53   160   92  169  1  0 99  0  0
 0  0    827    444      4    128    0    0    32     0  123  273  7  1 91  1  0
 0  0    827    444      4    128    0    0     0     0  129  279 12  1 87  0  0
 1  1    827    444      4    128    0    0     8     0  119  250 10  0 90  0  0
 0  0    827    444      4    128    0    0   244     0  401 1208 20  3 72  4  0
 2  0    867     82      1     92    0   53  7824 54852  849 1924 20 78  2  0  0
 3  0   1253     82      1     82    0  386  2368 395320 1658 2377  7 93  0  0  0
 0  0   1490    169      1     77    0  237   612 243056  970 1385  9 58 32  1  0
 0  0   1490    166      1     80    0    0  2716     0  231  528 13  8 75  4  0
 0  0   1490    165      1     81    0    0  1060     0  170  350 13  0 82  4  0
 0  0   1489    165      1     81    0    0    88     0  130  295 10  0 88  2  0
 0  0   1489    165      1     81    0    0   120     0  160  403 11  1 86  2  0
```

OOM Killer

- VMSTAT Meaning

Procs

r: The number of processes waiting for run time.

b: The number of processes in uninterruptible sleep.

Memory

swpd: the amount of virtual memory used.

free: the amount of idle memory.

buff: the amount of memory used as buffers.

cache: the amount of memory used as cache.

inact: the amount of inactive memory. (-a option)

active: the amount of active memory. (-a option)

Swap

si: Amount of memory swapped in from disk (/s).

so: Amount of memory swapped to disk (/s).

OOM Killer

IO

bi: Blocks received from a block device (blocks/s).

bo: Blocks sent to a block device (blocks/s).

System

in: The number of interrupts per second, including the clock.

cs: The number of context switches per second.

CPU

These are percentages of total CPU time.

us: Time spent running non-kernel code. (user time, including nice time)

sy: Time spent running kernel code. (system time)

id: Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.

wa: Time spent waiting for IO. Prior to Linux 2.5.41, included in idle.

st: Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown.

OOM Killer

IO

bi: Blocks received from a block device (blocks/s).

bo: Blocks sent to a block device (blocks/s).

System

in: The number of interrupts per second, including the clock.

cs: The number of context switches per second.

CPU

These are percentages of total CPU time.

us: Time spent running non-kernel code. (user time, including nice time)

sy: Time spent running kernel code. (system time)

id: Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.

wa: Time spent waiting for IO. Prior to Linux 2.5.41, included in idle.

st: Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown.

OOM Killer

- In some environments, when a system runs a single critical task, rebooting when a system runs into an OOM condition might be a viable option to return the system back to operational status quickly without administrator intervention. While not an optimal approach, the logic behind this is that if our application is unable to operate due to being killed by the OOM killer, then a reboot of the system will restore the application if it starts with the system at boot time. If the application is manually started by an administrator, this option is not beneficial.

- The following settings will cause the system to panic and reboot in an out-of-memory condition. The sysctl commands will set this in real time, and appending the settings to sysctl.conf will allow these settings to survive reboots. The X for kernel.panic is the number of seconds before the system should be rebooted. This setting should be adjusted to meet the needs of your environment.

```
sysctl vm.panic_on_oom=1
sysctl kernel.panic=X
echo "vm.panic_on_oom=1" >> /etc/sysctl.conf
echo "kernel.panic=X" >> /etc/sysctl.conf
```

OOM Killer

- We can also tune the way that the OOM killer handles OOM conditions with certain processes. Take, for example, our oracle process 2592 that was killed earlier. If we want to make our oracle process less likely to be killed by the OOM killer, we can do the following.

  echo -15 > /proc/2592/oom_adj

- We can make the OOM killer more likely to kill our oracle process by doing the following.

  echo 10 > /proc/2592/oom_adj

- If we want to exclude our oracle process from the OOM killer, we can do the following, which will exclude it completely from the OOM killer. It is important to note that this might cause unexpected behavior depending on the resources and configuration of the system. If the kernel is unable to kill a process using a large amount of memory, it will move onto other available processes. Some of those processes might be important operating system processes that ultimately might cause the system to go down.

  echo -17 > /proc/2592/oom_adj

OOM Killer

- We can set valid ranges for oom_adj from -16 to +15, and a setting of -17 exempts a process entirely from the OOM killer. The higher the number, the more likely our process will be selected for termination if the system encounters an OOM condition. The contents of /proc/2592/oom_score can also be viewed to determine how likely a process is to be killed by the OOM killer. A score of 0 is an indication that our process is exempt from the OOM killer. The higher the OOM score, the more likely a process will be killed in an OOM condition.

- The OOM killer can be completely disabled with the following command. This is not recommended for production environments, because if an out-of-memory condition does present itself, there could be unexpected behavior depending on the available system resources and configuration. This unexpected behavior could be anything from a kernel panic to a hang depending on the resources available to the kernel at the time of the OOM condition.

  sysctl vm.overcommit_memory=2

  echo "vm.overcommit_memory=2" >> /etc/sysctl.conf

JTAG Debugger

Porting Linux

Bringing Linux up on a new board will require some knowledge of assembly language for your processor.

There are several transitions from assembly to “C” and back if we’re using zImages

Debugging at this level will require the use of JTAGs, or other hardware assistance

JTAG Debugger

Hardware Debugging Tools

The traditional hardware debug tool was the In-Circuit Emulator (ICE).

A device that plugged into the CPU socket and emulated the CPU itself

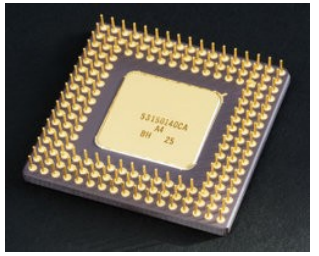
These were rather expensive $30K+ for the good ones

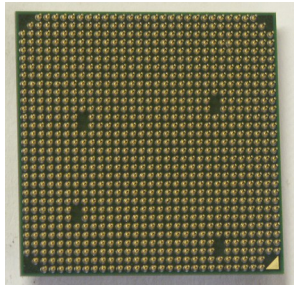
Today, most devices that call themselves an ICE are actually JTAGs

JTAG Debugger

Why the Traditional ICE has Faded Away

The biggest problem faced by the ICE concept was the increasing pin counts of processors E.g., 939 pins for the Athlon-64





Each pin required a wire to the ICE

Each wire started to become an antenna as frequencies increased

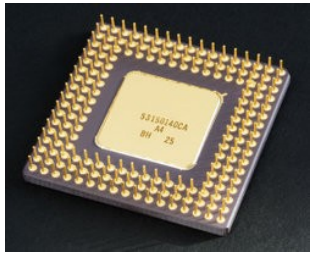
Processors also started to move to Ball Grid Array (BGA) packages

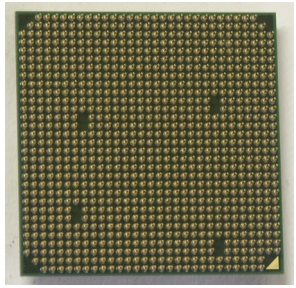
No way to get to the pins in the center of the part because the part is soldered to the motherboard

JTAG Debugger

Why the Traditional ICE has Faded Away

The biggest problem faced by the ICE concept was the increasing pin counts of processors E.g., 939 pins for the Athlon-64





Each pin required a wire to the ICE

Each wire started to become an antenna as frequencies increased

Processors also started to move to Ball Grid Array (BGA) packages

No way to get to the pins in the center of the part because the part is soldered to the motherboard

JTAG Debugger

The Joint Test Action Group (JTAG) is the name associated with the IEEE 1149.1 standard entitled *Standard Test Access Port and Boundary-Scan Architecture*.

Originally introduced in 1990 as a means to test printed circuit boards

An alternative to the *bed of nails*

JTAG Debugger

How JTAG works

JTAG is a boundary-scan device that allows the developer to sample the values of lines on the device

Allows you to change those values as well

JTAG is built to allow chaining of multiple devices Works for multi-core processors, too

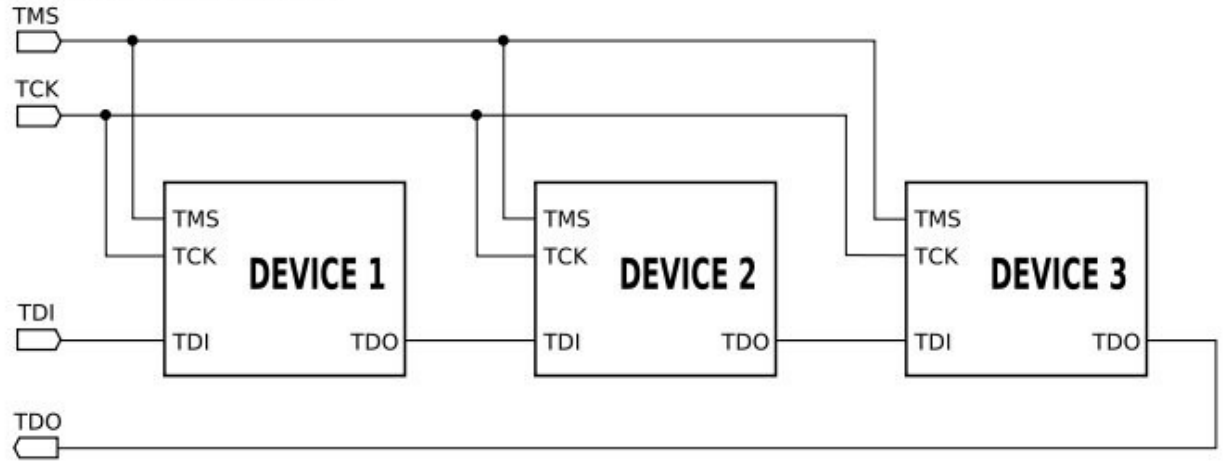
JTAG Debugger

JTAG Details

JTAG is a simple serial protocol

Configuration is done by manipulating the state machine of the device via the TMS line

1. TDI (Test Data In)
2. TDO (Test Data Out)
3. TCK (Test Clock)
4. TMS (Test Mode Select)
5. TRST (Test ReSeT) optional.



JTAG Debugger

JTAG-Aware Processors

Most embedded processors today support JTAG or one of its relatives like BDM

E.g., ARM/XScale, PPC, MIPS

Even the x86 has a JTAG port although it is rarely wired out

JTAG Debugger

JTAG Vendors

Several different vendors sell JTAG port interface hardware

JTAG is also referred to as On-Chip Debugging (OCD)

Here are a few of the vendors:

- Wind River Systems (http://www.windriver.com)
- Abatron AG (http://www.abatron.ch)
- American Arium (http://www.arium.com)
- Mentor Graphics (http://www.epitools.com)
- Lauterbach (https://www2.lauterbach.com/)

Some vendors do certain processors better than others

JTAG Debugger

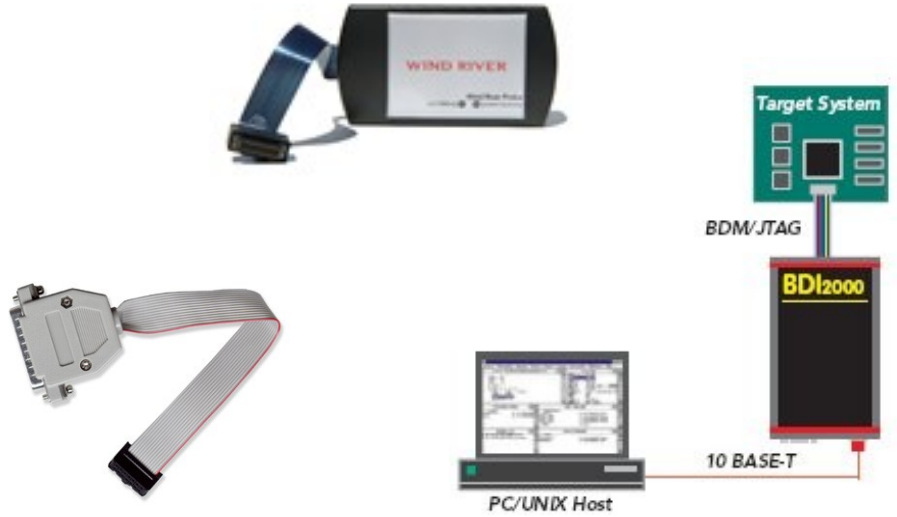
JTAG Connections

The maximum speed of JTAG is 100 MHz

- A ribbon cable is usually sufficient to connect to the target

Connection to the development host is accomplished via

- Parallel port
- USB
- Serial port
- Ethernet



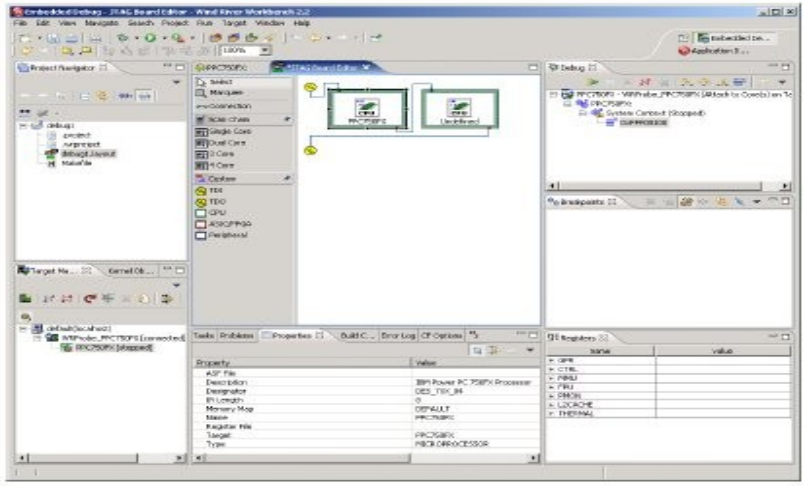
JTAG Debugger

Some JTAG interfaces use a GDB-style software interface

- Any GDB-aware front end will work

Others have Eclipse plug-ins to access the JTAG via an IDE

Some still use a command line interface



JTAG Debugger

What can you do with a JTAG?

Typical JTAG usage includes reflashing boot firmware

Even the really cheap JTAG units can do this

However, it is in the use as a debugging aid that JTAG comes into its ownYou can set hardware or software breakpoints and debug in source code

Sophisticated breakpoint strategies and multi-core debugging usually require the more expensive units

JTAG units can also be used to exercise the address bus and peripherals

- This is what JTAG was originally designed for

JTAG Debugger

Hardware Configuration Files

Most JTAG units require you to describe the hardware registers in a configuration file

This is also how you describe what processor architecture you are using

All of that information about register maps that you collected earlier now goes into the configuration file

Unfortunately, there is no standard format for these configuration files

Each JTAG vendor uses different syntax

JTAG Debugger

Example Configuration Files

Many JTAG units split the configuration files into a CPU register file and a board configuration file

```
;
; SDRAM Controller (SDRAMC)
;
sdramc_mr       MM      0xFFFFFF90      32      ;SDRAMC Mode Register
sdramc_tr       MM      0xFFFFFF94      32      ;SDRAMC Refresh Timer Register
sdramc_cr       MM      0xFFFFFF98      32      ;SDRAMC Configuration Register
sdramc_srr      MM      0xFFFFFF9C      32      ;SDRAMC Self Refresh Register
sdramc_lpr      MM      0xFFFFFFA0      32      ;SDRAMC Low Power Register
sdramc_ier      MM      0xFFFFFFA4      32      ;SDRAMC Interrupt Enable Register
```

```
; bdiGDB configuration file for AT91RM9200-DK
; ------------------------------------------
;
[INIT]
WREG    CPSR            0x000000D3  ;select supervisor mode
WM32    0xFFFFFF00      0x00000001  ;Cancel reset remapping
WM32    0xFFFFFC20      0x0000FF01  ;PMC_MOR : Enable main oscillator , OSCOUNT = 0xFF
;
;       Init Flash
WM32    0xFFFFFF10      0x00000000  ;MC_PUIA[0]
WM32    0xFFFFFF50      0x00000000  ;MC_PUP
WM32    0xFFFFFF54      0x00000000  ;MC_PUER: Memory controller protection unit disable
;WM32    0xFFFFFF04      0x00000000  ;MC_ASR
;WM32    0xFFFFFF08      0x00000000  ;MC_AASR
WM32    0xFFFFFF64      0x00000000  ;EBI_CFGR
WM32    0xFFFFFF70      0x00003284  ;SMC2_CSR[0]: 16bit, 2 TDF, 4 WS
;
;       Init Clocks
WM32    0xFFFFFC28      0x20263E04  ;PLLAR: 179,712000 MHz for PCK
DELAY   100
WM32    0xFFFFFC2C      0x10483E0E  ;PLLBR: 48,054857 MHz (divider by 2 for USB)
```

JTAG Debugger

Developing the Configuration File

The JTAG vendor will likely already have a register file for the processor

- ARM920, PPC8241, etc.

Your task will be to develop the board configuration file

There may be a configuration file for the reference board that you can use as a starting point

The configuration file is essentially a script of commands to initialize the target board

You keep working on it until you can initialize memory

Once memory is on-line, you should then be able to write values into memory via the JTAG that can be read back

Then, enhance the configuration to initialize other peripherals

JTAG Debugger

Linux-Aware JTAGs

There are several rather tricky transitions during the Linux booting process

Transitioning from flash to RAM

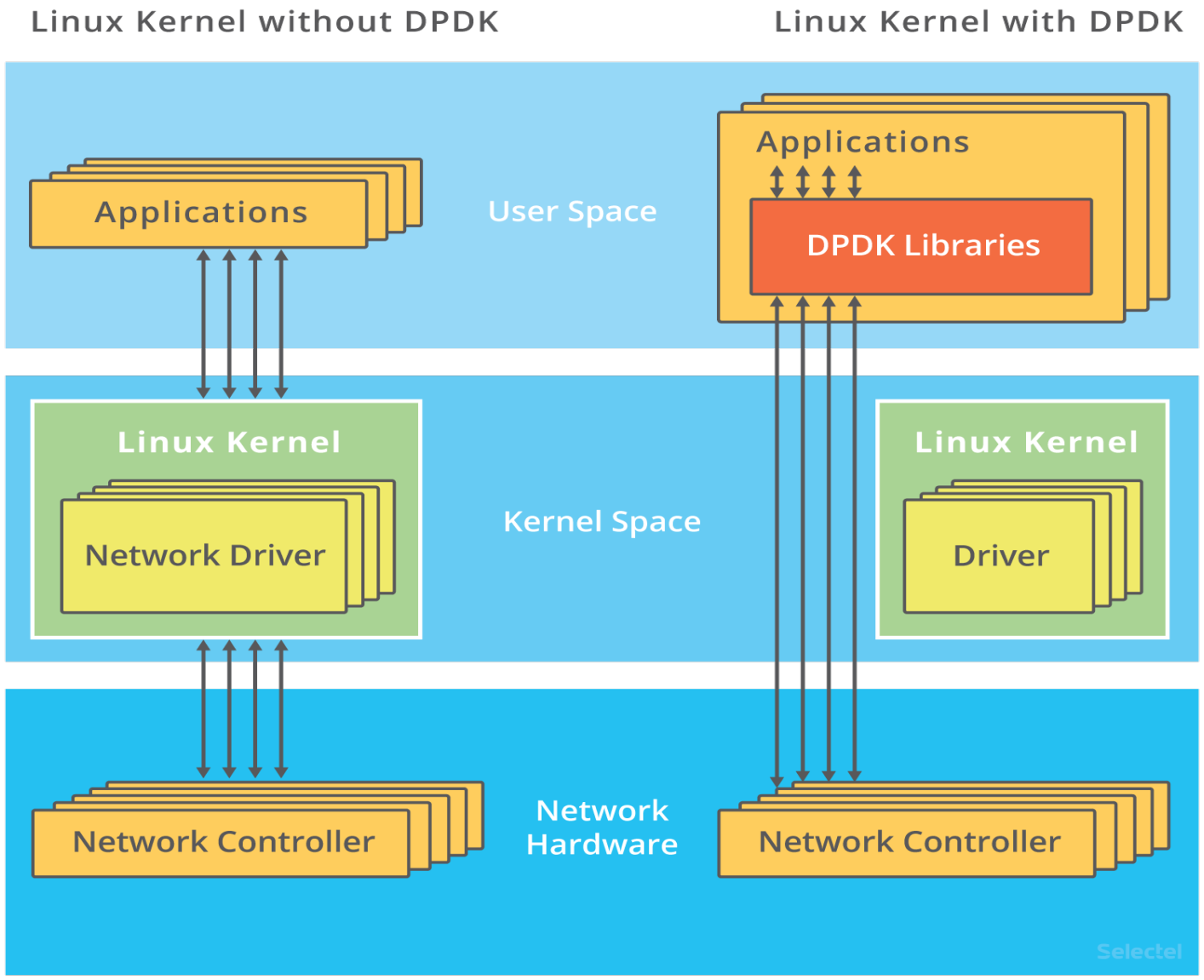
Transitioning from physical addresses to kernel virtual addresses

These transitions require the use of hardware breakpoints

Make sure that your JTAG is “Linux aware”

It must understand Linux’s use of the MMU to be of much use for driver debugging

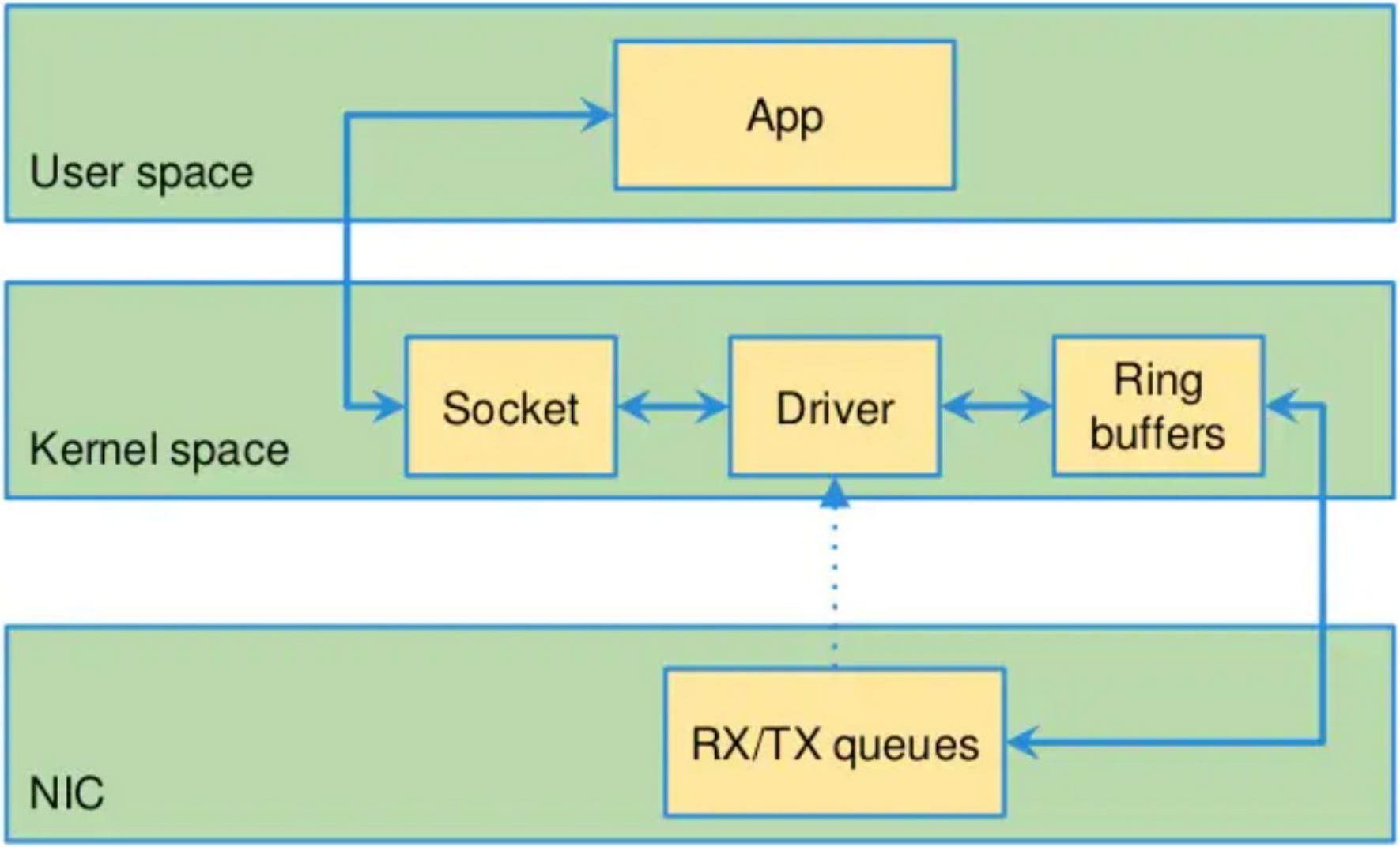
DPDK


Linux Kernel without DPDK
Linux Kernel with DPDK
Applications
User Space
Applications
DPDK Libraries
Linux Kernel
Network Driver
Kernel Space
Linux Kernel
Driver
Network Controller
Network Hardware
Network Controller
Selectel

DPDK


Application
Network
Driver
DPDK
Linux Kernel
Network
Controller
Selectel

DPDK


User space
App
Kernel space
Socket
Driver
Ring buffers
NIC
RX/TX queues

DPDK

System calls

Context switching on blocking I/O

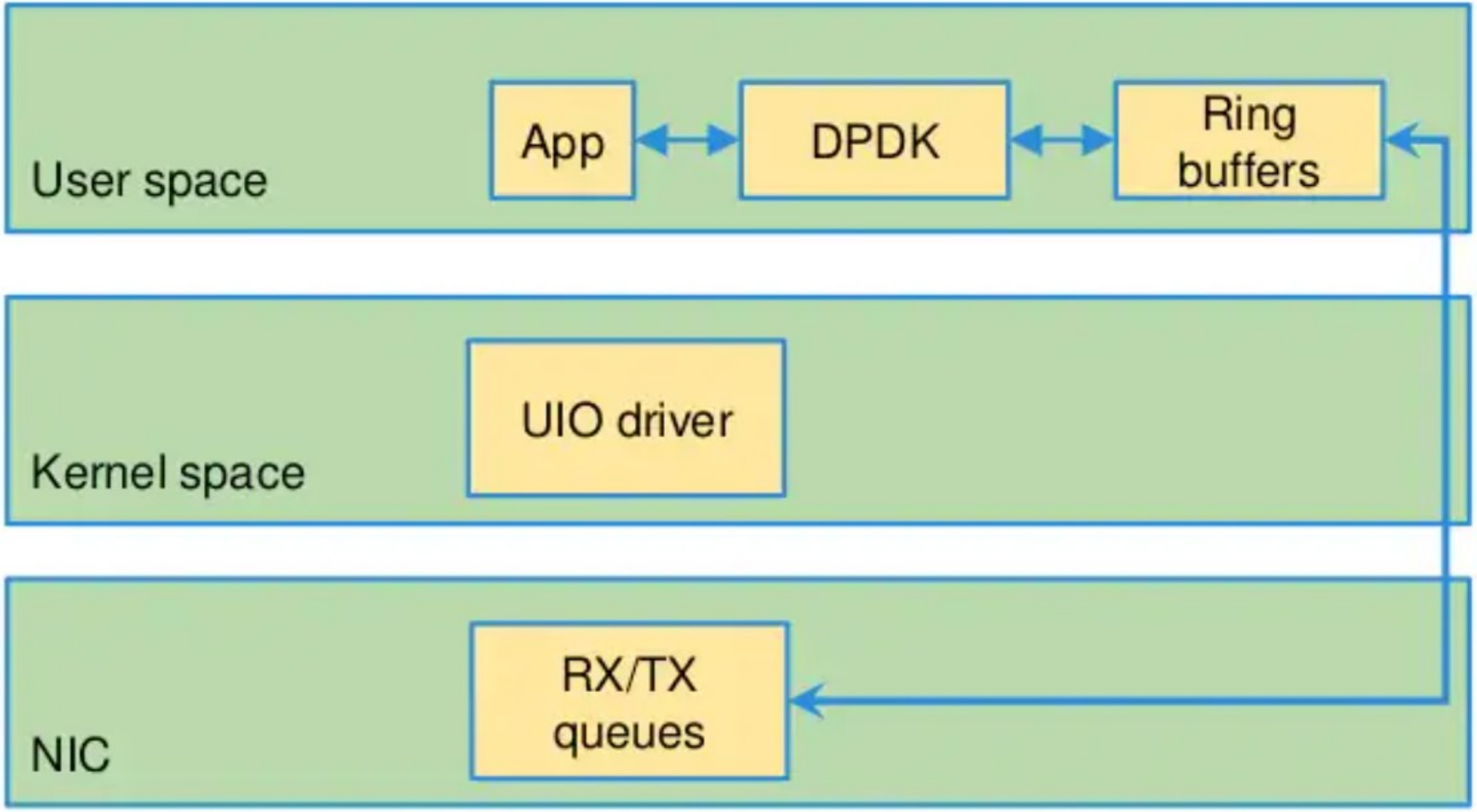
Data copying from kernel to user space

Interrupt handling in kernel

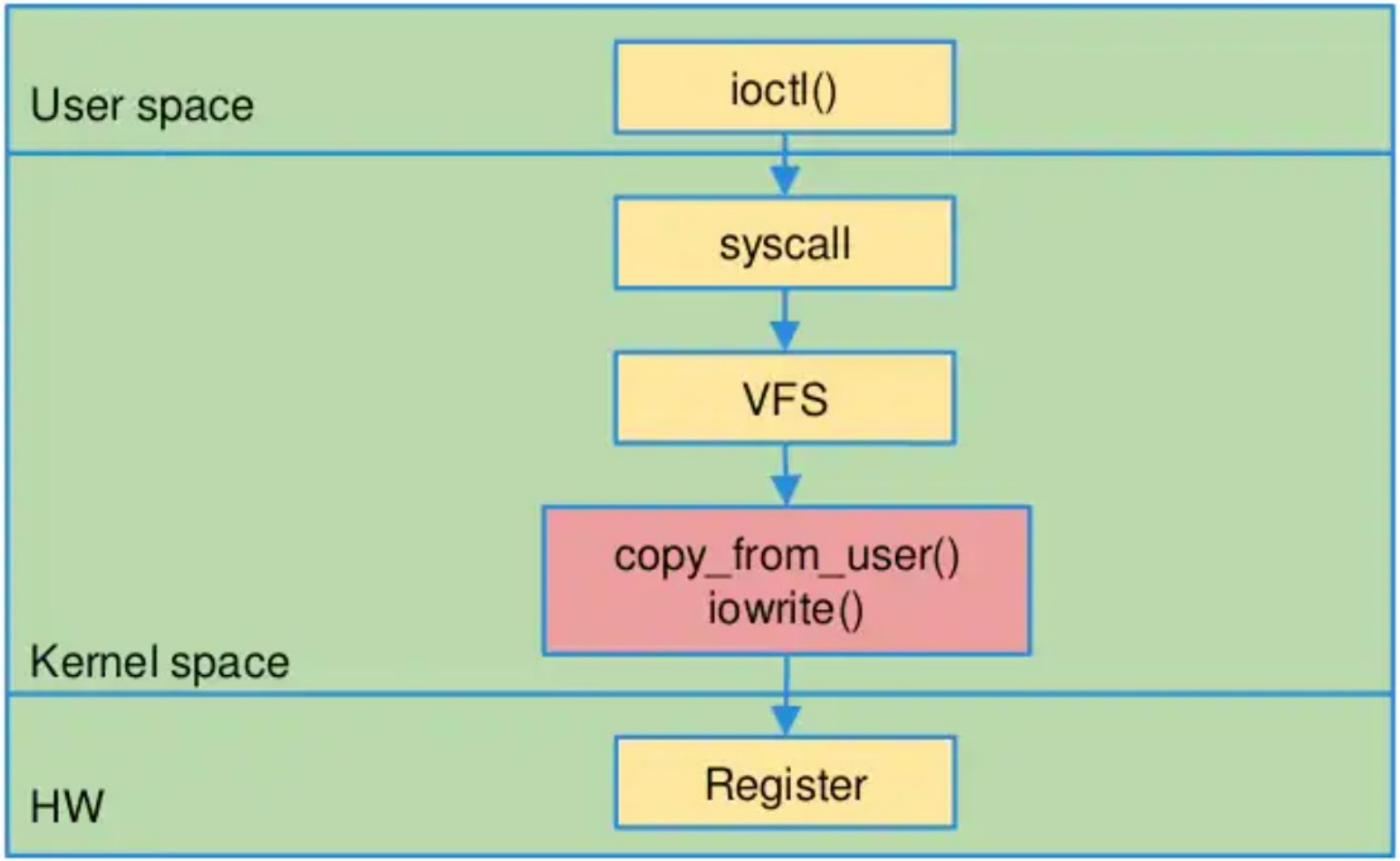
DPDK

Function	Activity	Time (ns)
sendto	system call	96
sosend_dgram	lock sock_buff, alloc mbuf, copy in	137
udp_output	UDP header setup	57
ip_output	route lookup, ip header setup	198
ether_otput	MAC lookup, MAC header setup	162
ixgbe_xmit	device programming	220
Total		**950**

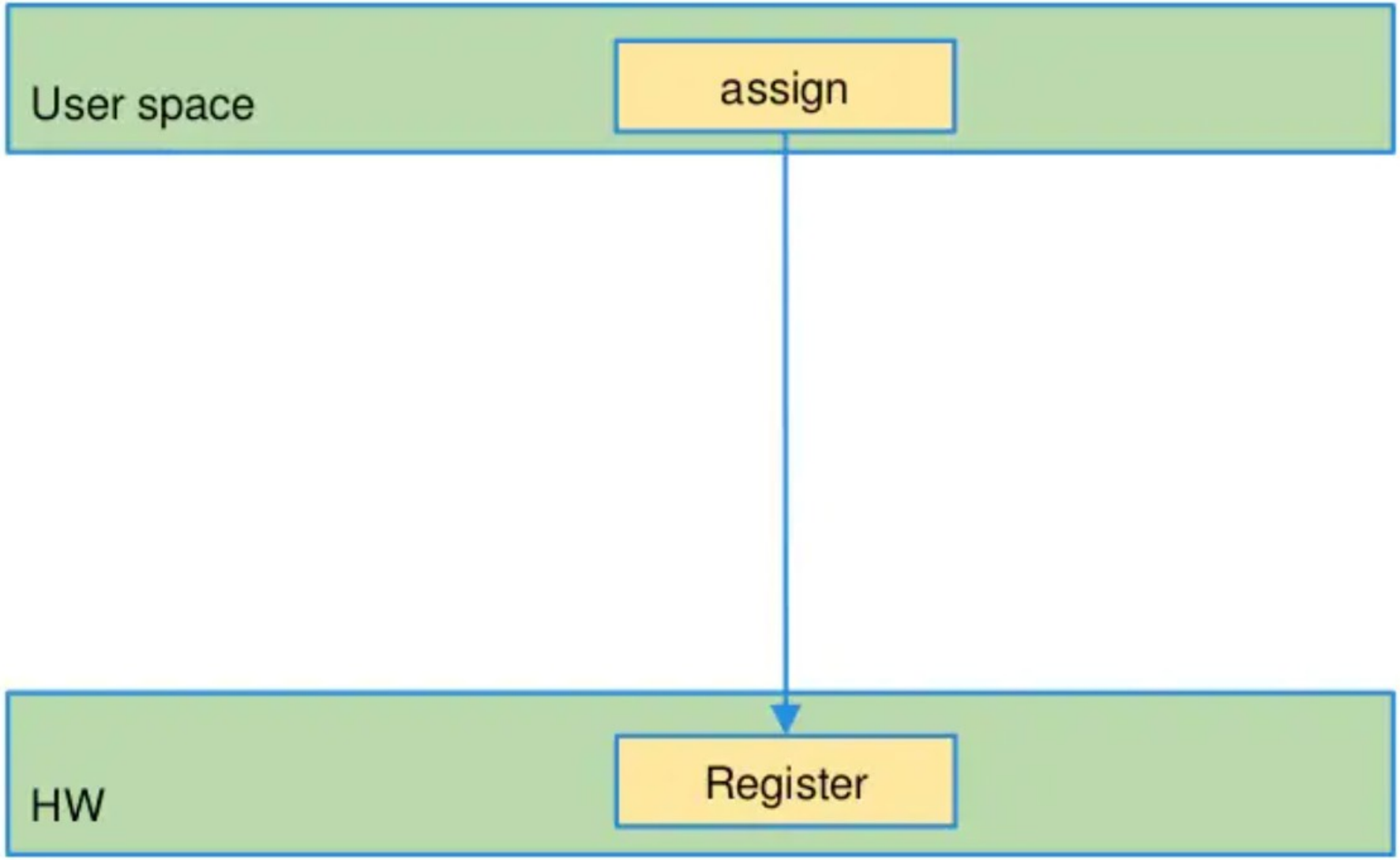
DPDK


User space
App
DPDK
Ring buffers
Kernel space
UIO driver
NIC
RX/TX queues

DPDK


User space
ioctl()
syscall
VFS
copy_from_user()
iowrite()
Kernel space
Register
HW

DPDK



DPDK

Uncompress DPDK and Browse Sources

First, uncompress the archive and move to the uncompressed DPDK source directory:

tar xJf dpdk-<version>.tar.xz
cd dpdk-<version>

The DPDK is composed of several directories:

lib: Source code of DPDK libraries
drivers: Source code of DPDK poll-mode drivers
app: Source code of DPDK applications (automatic tests)
examples: Source code of DPDK application examples
config, buildtools: Framework-related scripts and configuration

DPDK

Compiling and Installing DPDK System-wide

DPDK can be configured, built and installed on your system using the tools **meson** and **ninja**.

DPDK Configuration

To configure a DPDK build use:

meson <options> build

where “build” is the desired output build directory, and “<options>” can be empty or one of a number of meson or DPDK-specific build options, described later in this section. The configuration process will finish with a summary of what DPDK libraries and drivers are to be built and installed, and for each item disabled, a reason why that is the case. This information can be used, for example, to identify any missing required packages for a driver.

DPDK

Once configured, to build and then install DPDK system-wide use:

cd build

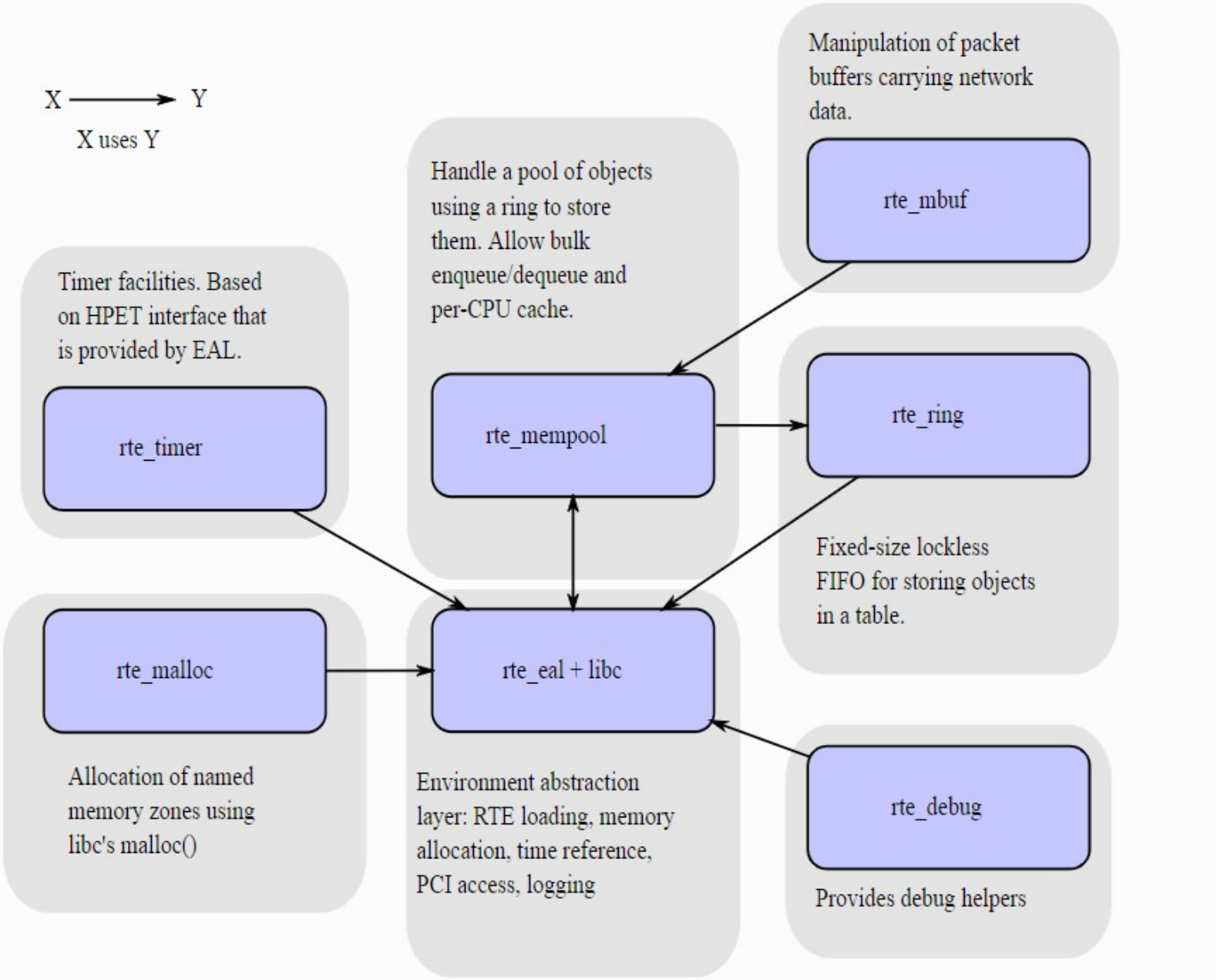
ninja

ninja install

ldconfig

The last two commands above generally need to be run as root, with the ninja install step copying the built objects to their final system-wide locations, and the last step causing the dynamic loader ld.so to update its cache to take account of the new objects.

DPDK


X → Y
X uses Y
Manipulation of packet buffers carrying network data.
rte_mbuf
Handle a pool of objects using a ring to store them. Allow bulk enqueue/dequeue and per-CPU cache.
Timer facilities. Based on HPET interface that is provided by EAL.
rte_timer
rte_mempool
rte_ring
Fixed-size lockless FIFO for storing objects in a table.
rte_malloc
rte_eal + libc
Allocation of named memory zones using libc's malloc()
Environment abstraction layer: RTE loading, memory allocation, time reference, PCI access, logging
rte_debug
Provides debug helpers

DPDK

Ring Manager (librte_ring)

The ring structure provides a lockless multi-producer, multi-consumer FIFO API in a finite size table. It has some advantages over lockless queues; easier to implement, adapted to bulk operations and faster. A ring is used by the Memory Pool Manager (librte_mempool) and may be used as a general communication mechanism between cores and/or execution blocks connected together on a logical core.

Memory Pool Manager (librte_mempool)

The Memory Pool Manager is responsible for allocating pools of objects in memory. A pool is identified by name and uses a ring to store free objects. It provides some other optional services, such as a per-core object cache and an alignment helper to ensure that objects are padded to spread them equally on all RAM channels.

DPDK

Network Packet Buffer Management (librte_mbuf)

The mbuf library provides the facility to create and destroy buffers that may be used by the DPDK application to store message buffers. The message buffers are created at startup time and stored in a mempool, using the DPDK mempool library.

This library provides an API to allocate/free mbufs, manipulate packet buffers which are used to carry network packets.

Timer Manager (librte_timer)

This library provides a timer service to DPDK execution units, providing the ability to execute a function asynchronously. It can be periodic function calls, or just a one-shot call. It uses the timer interface provided by the Environment Abstraction Layer (EAL) to get a precise time reference and can be initiated on a per-core basis as required.

DPDK

Ethernet* Poll Mode Driver Architecture

The DPDK includes Poll Mode Drivers (PMDs) for 1 GbE, 10 GbE and 40GbE, and para virtualized virtio Ethernet controllers which are designed to work without asynchronous, interrupt-based signaling mechanisms.

Packet Forwarding Algorithm Support

The DPDK includes Hash (librte_hash) and Longest Prefix Match (LPM,librte_lpm) libraries to support the corresponding packet forwarding algorithms.

librte_net

The librte_net library is a collection of IP protocol definitions and convenience macros. It is based on code from the FreeBSD* IP stack and contains protocol numbers (for use in IP headers), IP-related macros, IPv4/IPv6 header structures and TCP, UDP and SCTP header structures.

DPDK

Ethernet* Poll Mode Driver Architecture

The DPDK includes Poll Mode Drivers (PMDs) for 1 GbE, 10 GbE and 40GbE, and para virtualized virtio Ethernet controllers which are designed to work without asynchronous, interrupt-based signaling mechanisms.

Packet Forwarding Algorithm Support

The DPDK includes Hash (librte_hash) and Longest Prefix Match (LPM,librte_lpm) libraries to support the corresponding packet forwarding algorithms.

librte_net

The librte_net library is a collection of IP protocol definitions and convenience macros. It is based on code from the FreeBSD* IP stack and contains protocol numbers (for use in IP headers), IP-related macros, IPv4/IPv6 header structures and TCP, UDP and SCTP header structures.

DPDK

helloworld.c

GIT Introduction

Git is a distributed revision control and source code management system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Git is a free software distributed under the terms of the GNU General Public License version 2.

GIT Advantages

Free and open source

Git is released under GPL's open source license. It is available freely over the internet. You can use Git to manage property projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.

Fast and small

As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.

GIT Advantages

Implicit backup

The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.

Security

Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that, it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.

GIT Advantages

No need of powerful hardware

In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don’t interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

Easier branching

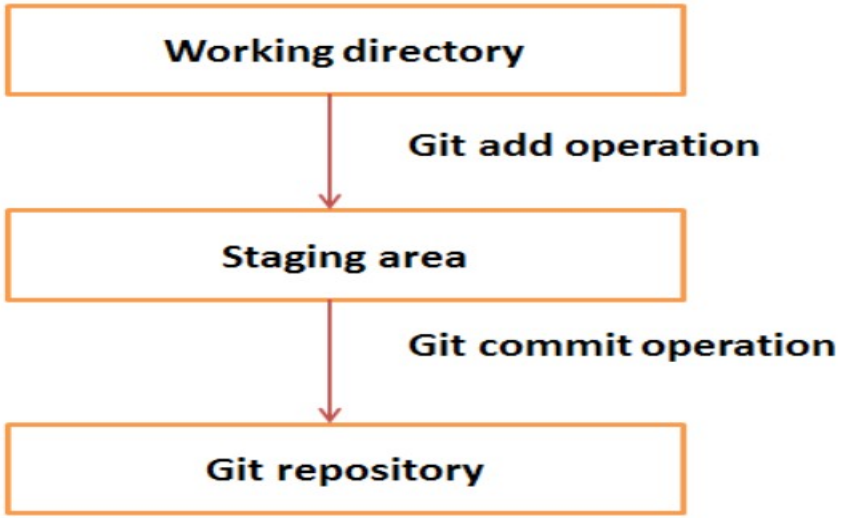
CVCS uses cheap copy mechanism, If we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with Git is very simple. It takes only a few seconds to create, delete, and merge branches.

GIT Working

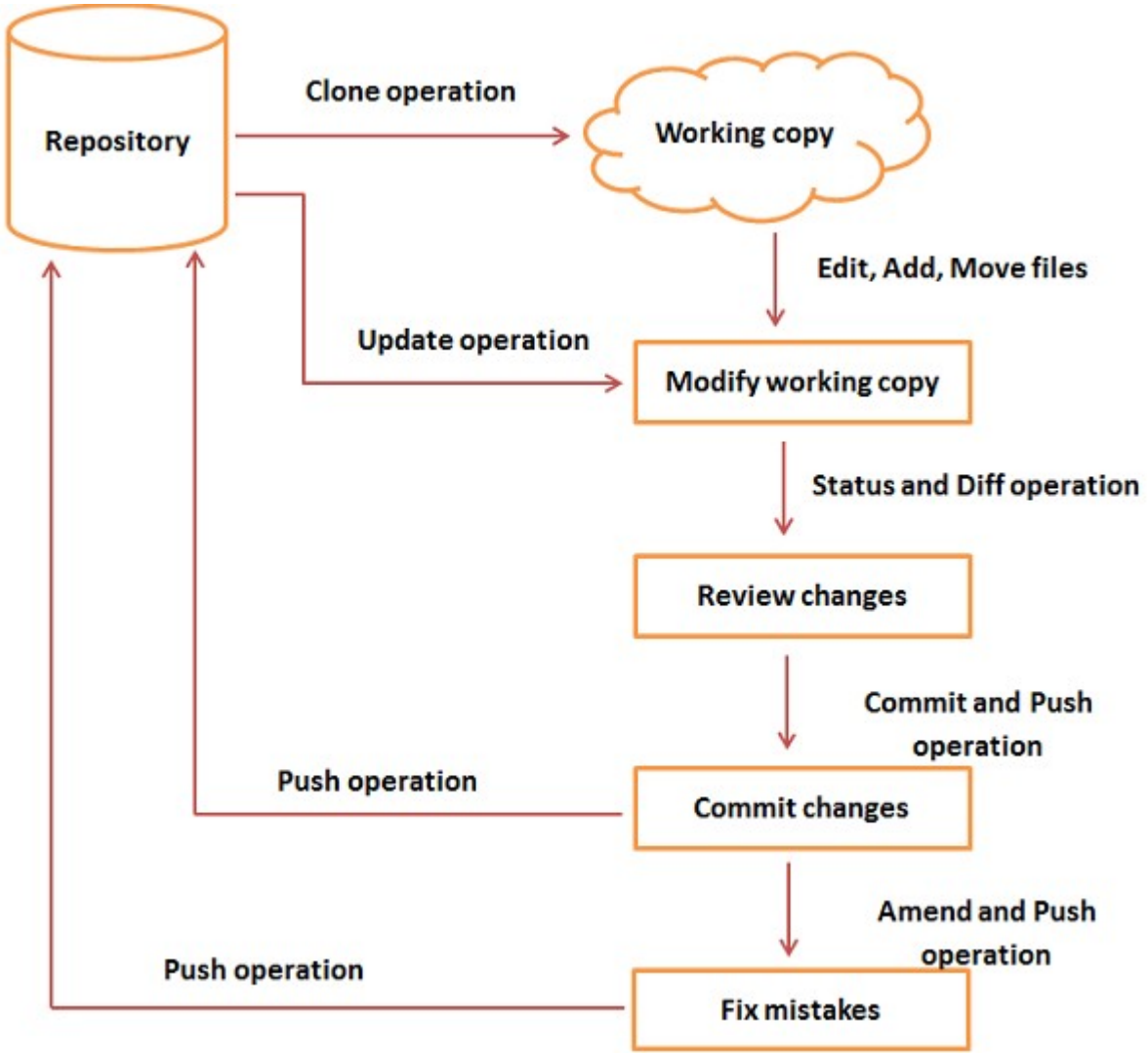
Step 1 − You modify a file from the working directory.

Step 2 − You add these files to the staging area.

Step 3 − You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.



GIT Working


Repository
Clone operation
Working copy
Edit, Add, Move files
Update operation
Modify working copy
Status and Diff operation
Review changes
Commit and Push operation
Push operation
Commit changes
Amend and Push operation
Push operation
Fix mistakes

GIT config

 # git config --list

 # git config --global user.name "Testing"

To set these configuration values as global, add the --global option, and if you omit --global option, then your configurations are specific for the current Git repository.

GIT add new repo

 # git init

 # echo 'Testing' > README

 # git status

 # git add .

 # git status

 # git commit -m 'Initial commit'

 # git log

GIT diff

 # git diff HEAD HEAD~ or git diff HEAD HEAD~2

GIT .gitignore

GIT undo changes

git reset --hard a1e8fb5

git revert HEAD

GIT tag

```
$ git tag -a v1.4 -m "my version 1.4"
$ git tag
$ git show v1.4

$ git tag v1.4-lw
$ git tag
$ git show v1.4-lw

$ git log --pretty=oneline
$ git tag -a v1.2 9fceb02
$ git tag
$ git show v1.2
```

GIT tag

```
$ git tag -d v1.4-lw
Deleted tag 'v1.4-lw' (was e7d5add)

$ git checkout v2.0.0
Note: switching to 'v2.0.0'.

$ git checkout -b version2 v2.0.0
Switched to a new branch 'version2'
```

GIT stash

 # git stash -a

 # git stash list

 # git stash pop stash@{0} or git stash pop or git stash apply

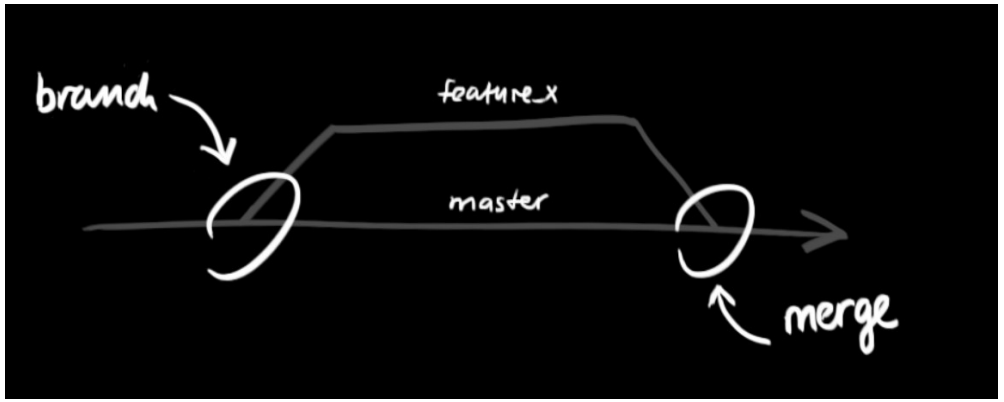
GIT stash

 # git stash -a

 # git stash list

 # git stash pop stash@{0} or git stash pop or git stash apply

GIT branch



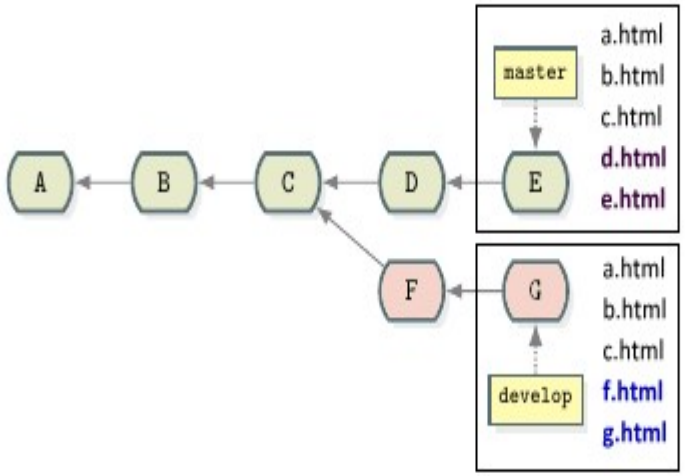
 # git checkout -b feature_x

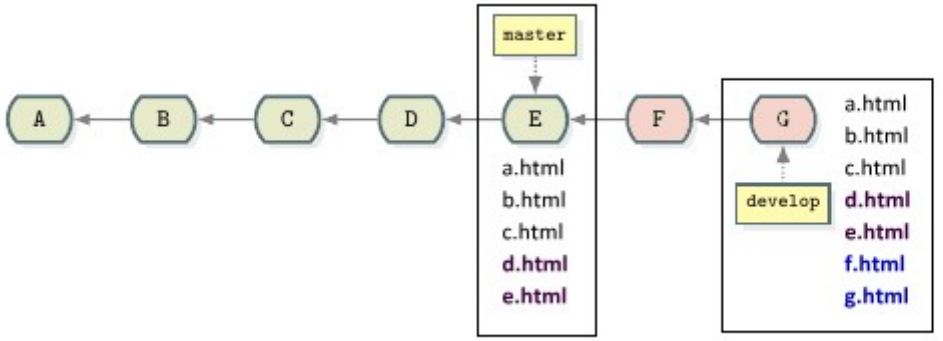
 # git checkout master

 # git merge <branch>

 # git branch -d feature_x

GIT rebase


A
B
C
D
E
F
G
master
develop
a.html
b.html
c.html
d.html
e.html
a.html
b.html
c.html
f.html
g.html


A
B
C
D
E
F
G
master
develop
a.html
b.html
c.html
d.html
e.html
a.html
b.html
c.html
d.html
e.html
f.html
g.html

GIT hooks

Hook Name	Invoked By	Description	Parameters (Number and Description)
applypatch-msg	git am	Can edit the commit message file and is often used to verify or actively format a patch's message to a project's standards. A non-zero exit status aborts the commit.	(1) name of the file containing the proposed commit message
pre-applypatch	git am	This is actually called *after* the patch is applied, but *before* the changes are committed. Exiting with a non-zero status will leave the changes in an uncommitted state. Can be used to check the state of the tree before actually committing the changes.	(none)
post-applypatch	git am	This hook is run after the patch is applied and committed. Because of this, it cannot abort the process, and is mainly used for creating notifications.	(none)
pre-commit	git commit	This hook is called before obtaining the proposed commit message. Exiting with anything other than zero will abort the commit. It is used to check the commit itself (rather than the message).	(none)

GIT hooks

Hook Name	Invoked By	Description	Parameters (Number and Description)
prepare-commit-msg	git commit	Called after receiving the default commit message, just prior to firing up the commit message editor. A non-zero exit aborts the commit. This is used to edit the message in a way that cannot be suppressed.	(1 to 3) Name of the file with the commit message, the source of the commit message (message, template, merge, squash, or commit), and the commit SHA-1 (when operating on an existing commit).
commit-msg	git commit	Can be used to adjust the message after it has been edited in order to ensure conformity to a standard or to reject based on any criteria. It can abort the commit if it exits with a non-zero value.	(1) The file that holds the proposed message.
post-commit	git commit	Called after the actual commit is made. Because of this, it cannot disrupt the commit. It is mainly used to allow notifications.	(none)
pre-rebase	git rebase	Called when rebasing a branch. Mainly used to halt the rebase if it is not desirable.	(1 or 2) The upstream from where it was forked, the branch being rebased (not set when rebasing current)

GIT hooks

Hook Name	Invoked By	Description	Parameters (Number and Description)
post-checkout	git checkout and git clone	Run when a checkout is called after updating the worktree or after git clone. It is mainly used to verify conditions, display differences, and configure the environment if necessary.	(3) Ref of the previous HEAD, ref of the new HEAD, flag indicating whether it was a branch checkout (1) or a file checkout (0)
post-merge	git merge or git pull	Called after a merge. Because of this, it cannot abort a merge. Can be used to save or apply permissions or other kinds of data that git does not handle.	(1) Flag indicating whether the merge was a squash.
pre-push	git push	Called prior to a push to a remote. In addition to the parameters, additional information, separated by a space is passed in through stdin in the form of “<local ref> <local sha1> <remote ref> <remote sha1>”. Parsing the input can get you additional information that you can use to check. For instance, if the local sha1 is 40 zeros long, the push is a delete and if the remote sha1 is 40 zeros, it is a new branch. This can be used to do many comparisons of the pushed ref to what is currently there. A non-zero exit status aborts the push.	(2) Name of the destination remote, location of the destination remote

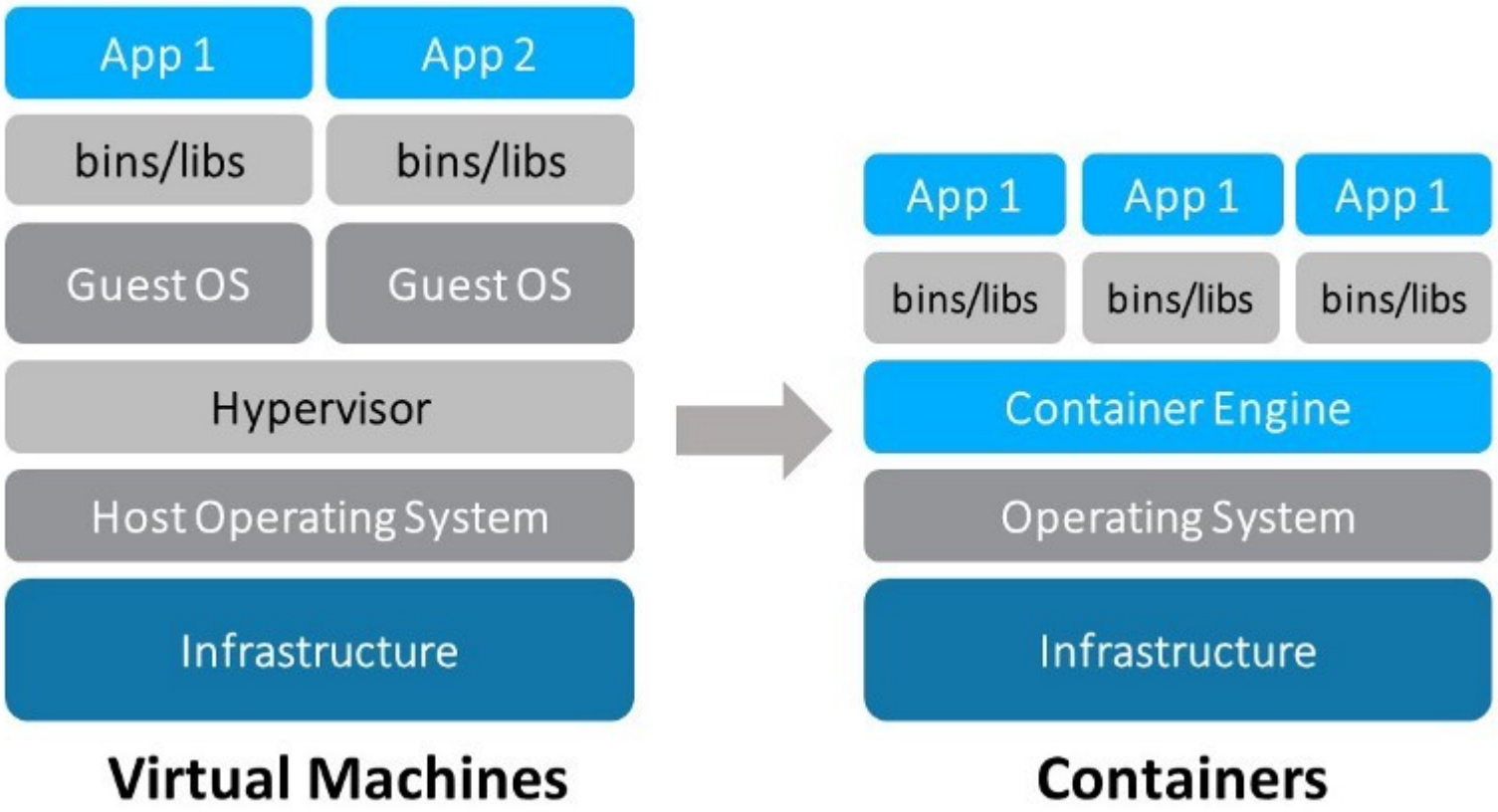
GIT hooks

Hook Name	Invoked By	Description	Parameters (Number and Description)
pre-receive	git-receive-pack on the remote repo	This is called on the remote repo just before updating the pushed refs. A non-zero status will abort the process. Although it receives no parameters, it is passed a string through stdin in the form of “<old-value> <new-value> <ref-name>” for each ref.	(none)
update	git-receive-pack on the remote repo	This is run on the remote repo once for each ref being pushed instead of once for each push. A non-zero status will abort the process. This can be used to make sure all commits are only fast-forward, for instance.	(3) The name of the ref being updated, the old object name, the new object name
post-receive	git-receive-pack on the remote repo	This is run on the remote when pushing after the all refs have been updated. It does not take parameters, but receives info through stdin in the form of “<old-value> <new-value> <ref-name>”. Because it is called after the updates, it cannot abort the process.	(none)

GIT hooks

Hook Name	Invoked By	Description	Parameters (Number and Description)
post-update	git-receive-pack on the remote repo	This is run only once after all of the refs have been pushed. It is similar to the post-receive hook in that regard, but does not receive the old or new values. It is used mostly to implement notifications for the pushed refs.	(?) A parameter for each of the pushed refs containing its name
pre-auto-gc	git gc --auto	Is used to do some checks before automatically cleaning repos.	(none)
post-rewrite	git commit --amend, git-rebase	This is called when git commands are rewriting already committed data. In addition to the parameters, it receives strings in stdin in the form of “<old-sha1> <new-sha1>”.	(1) Name of the command that invoked it (amend or rebase)

VM and Container



VM and Container

Container Orchestration Technologies

When there are handful of containerized applications, it is not that difficult to manage the deployment, running and maintenance of containers. But, if there are thousands of containers and services, then it become difficult to manage. Orchestration tools and technology automates the deployment, management, scaling, networking and availability of container-based applications across cluster of physical or virtual machines.

- Kubernetes
- Docker Swarm

Thank you