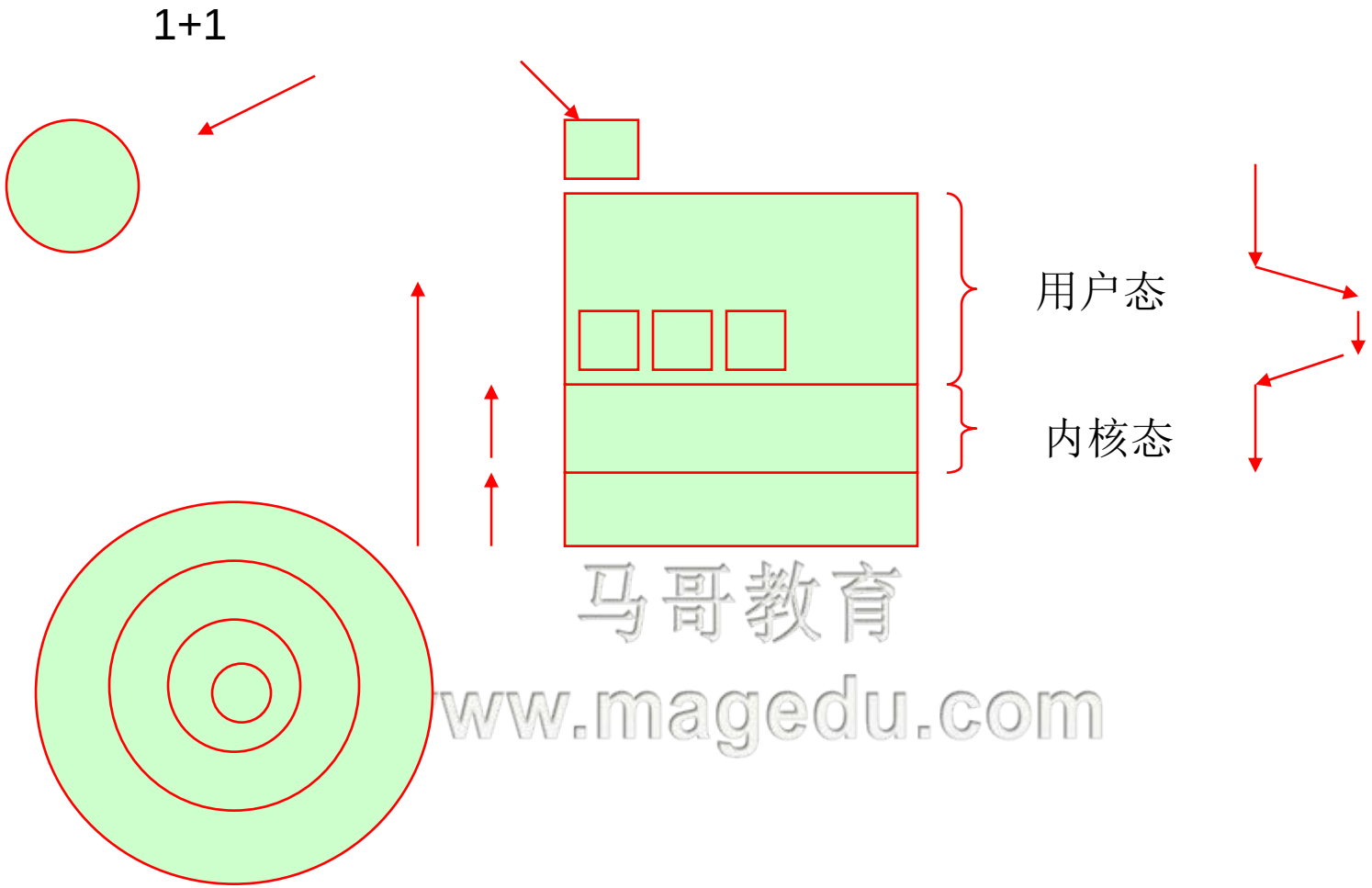
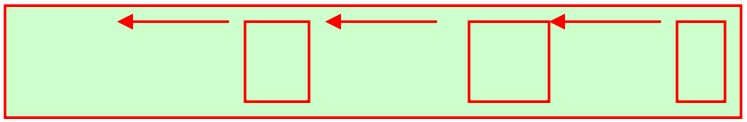
马哥教育

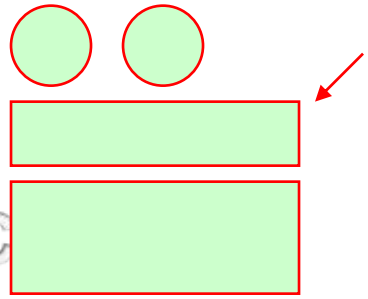
Understanding the Linux operating system

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

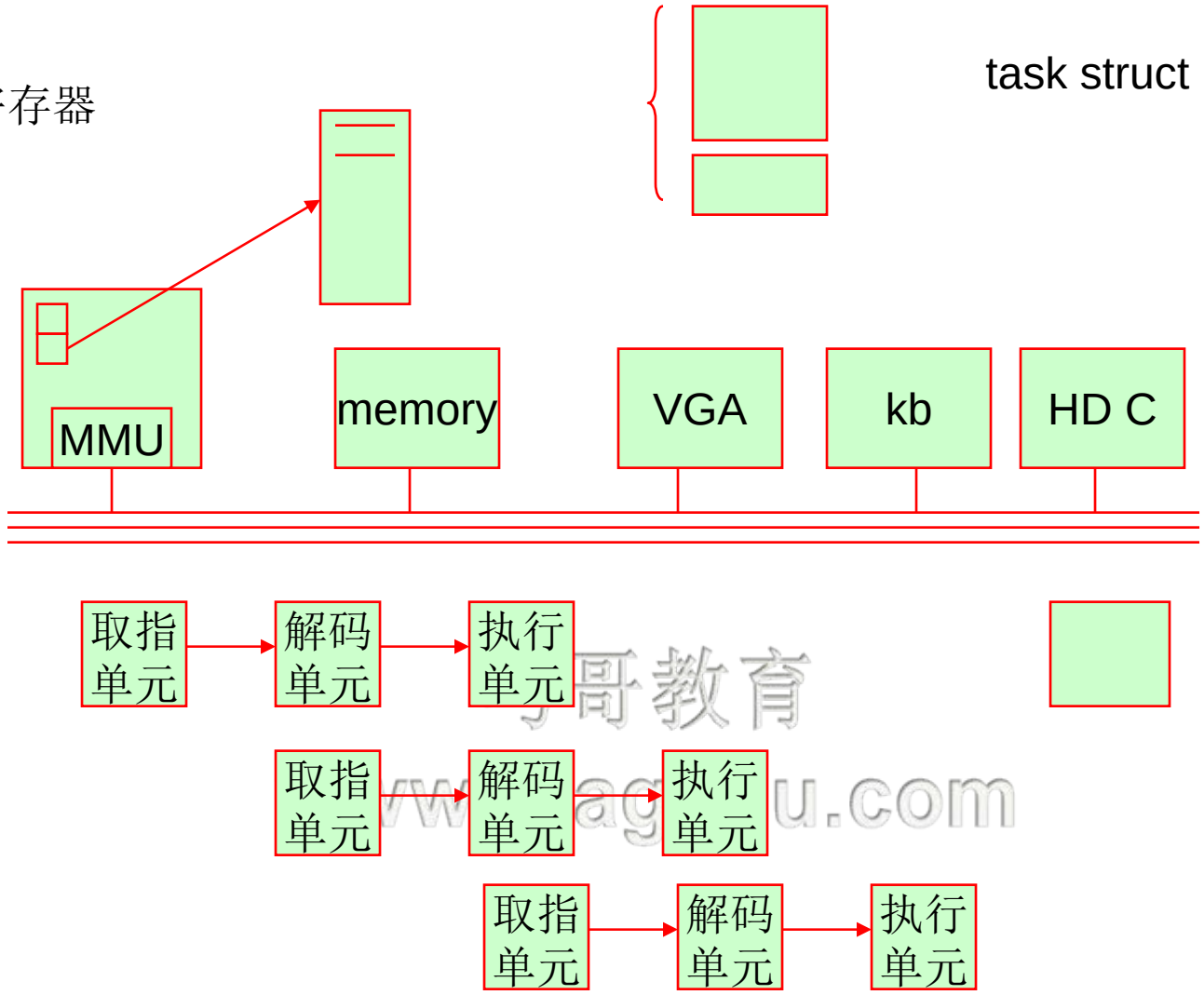


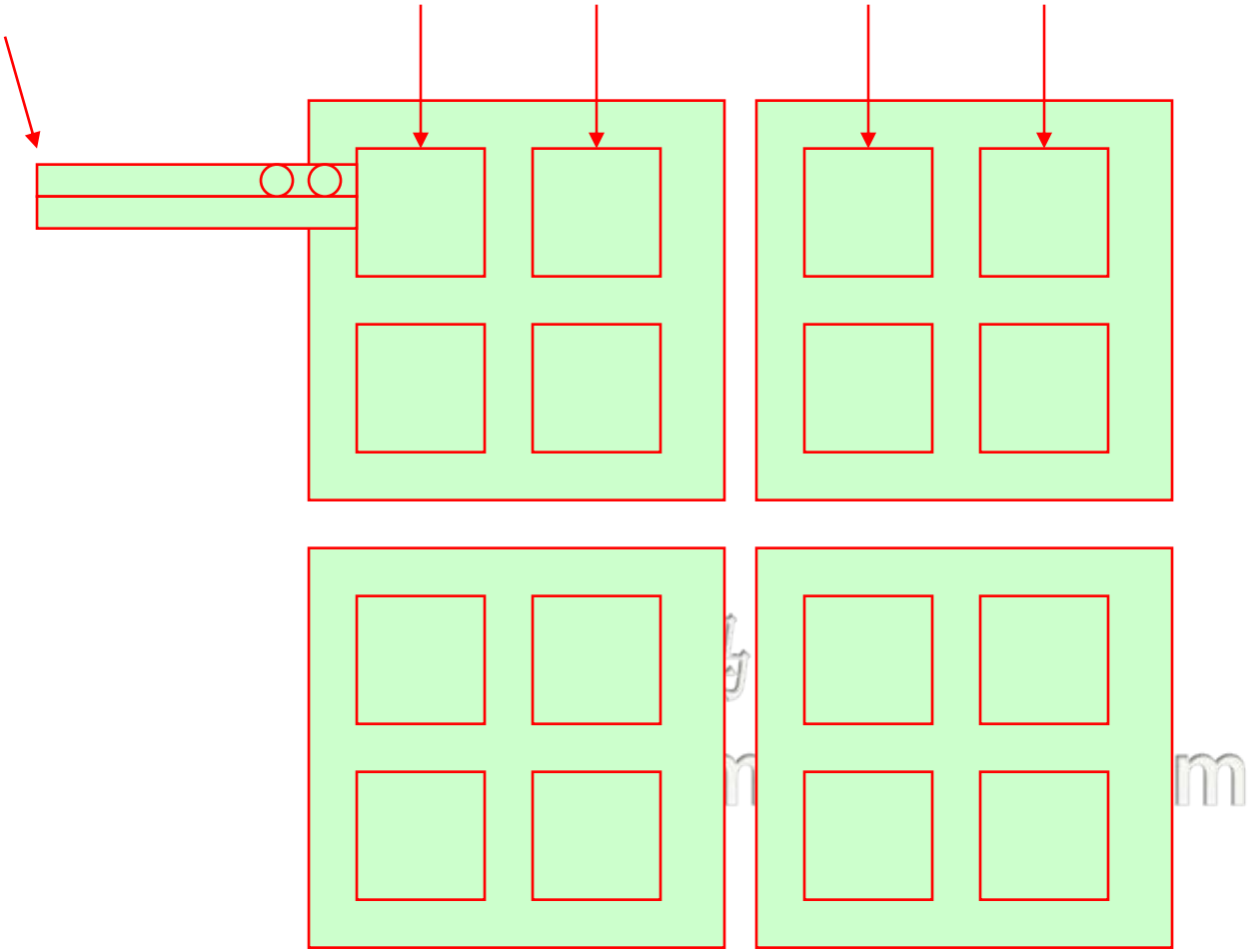
- 第一代：真空管，穿孔卡片
- 第二代：晶体管，批处理系统
  - Mainframe
  - Fortran
- 第三代：集成电路，多道程序设计
  - 分时
- 第四代：PC
  - LSI，CP/M



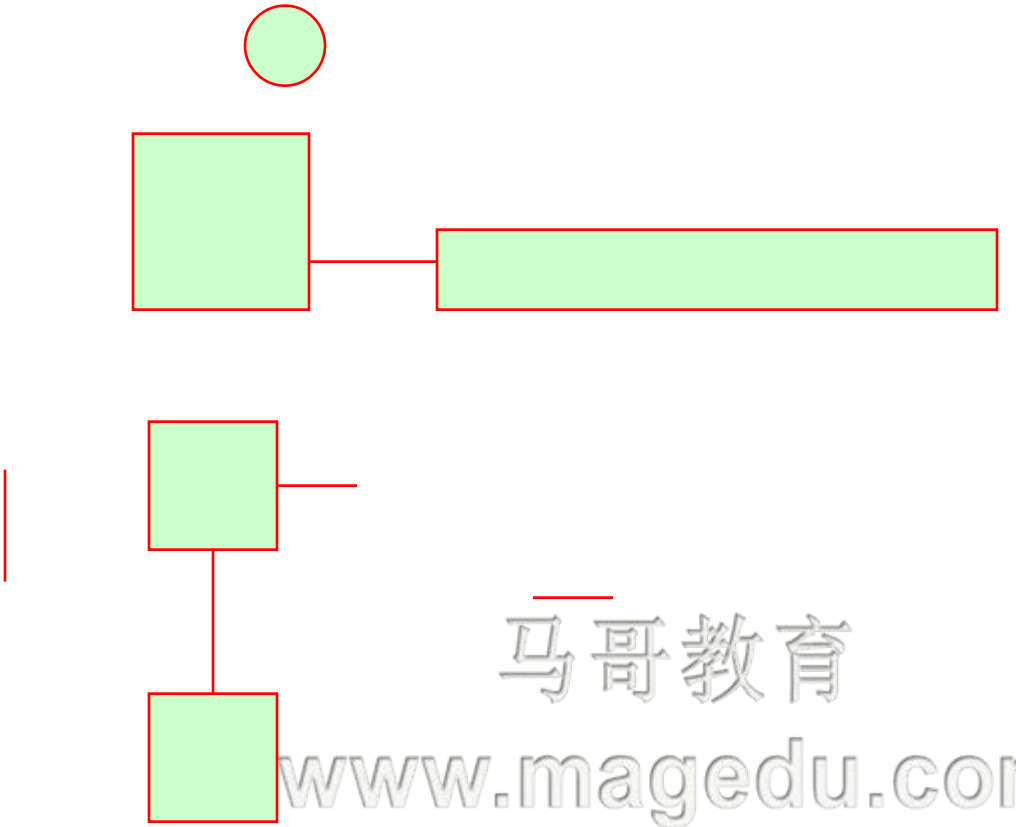


分页：memory page

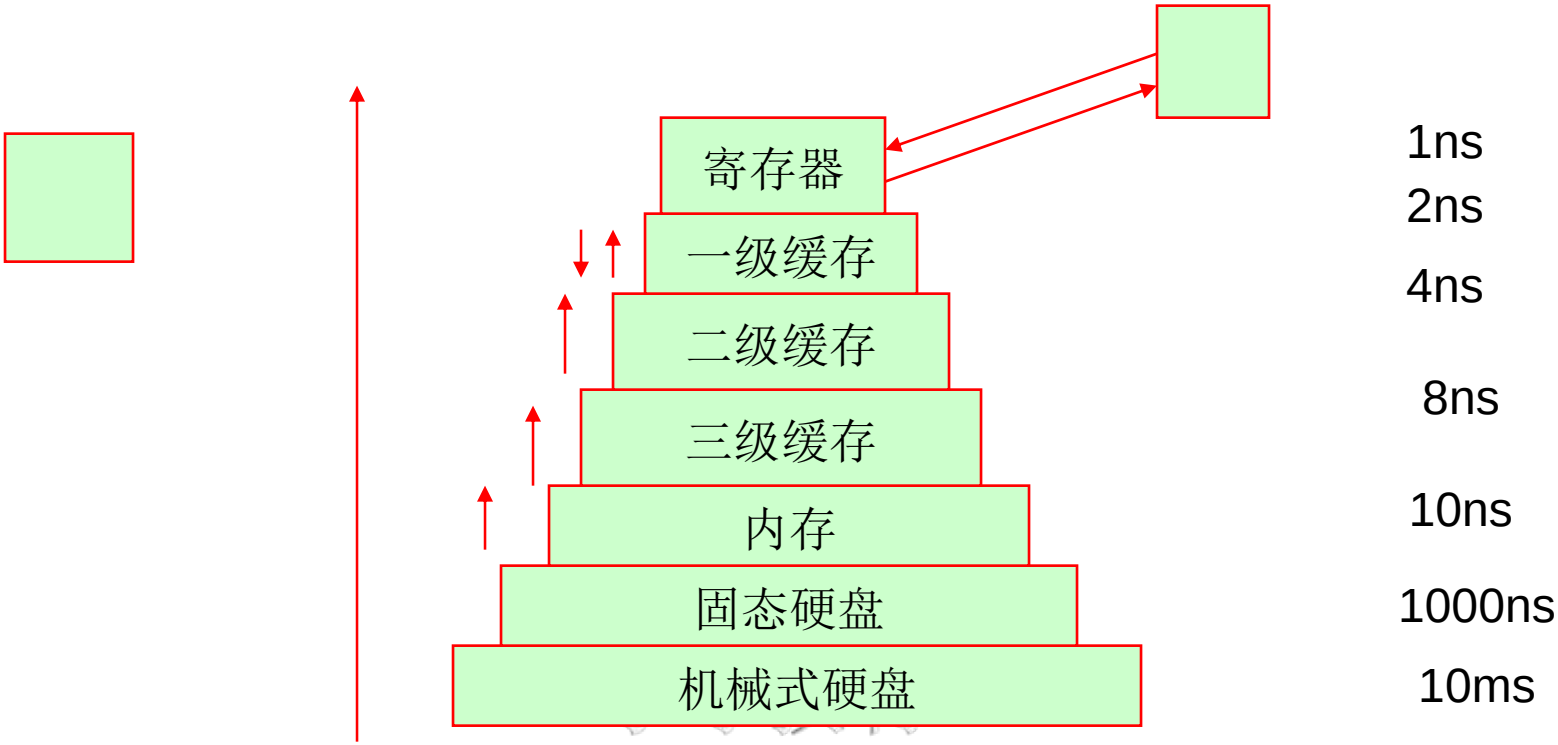


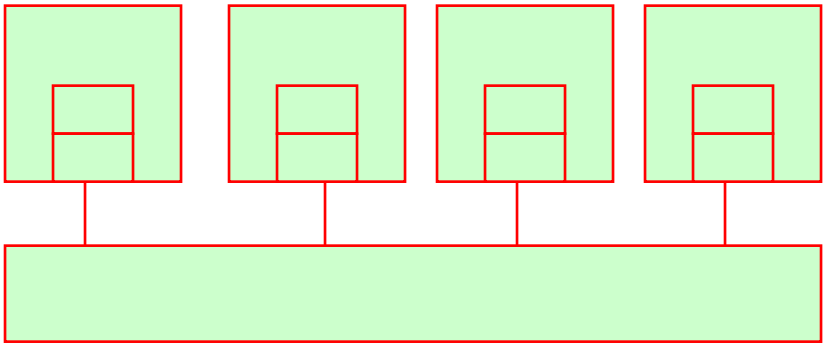


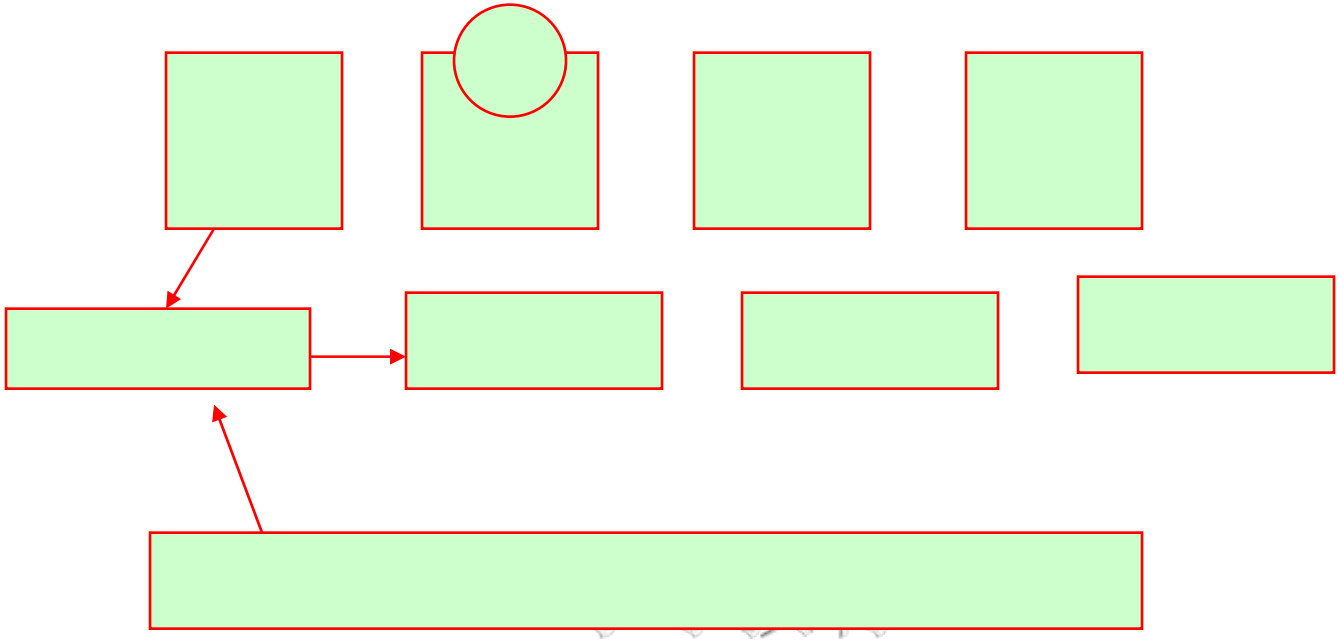
CPU Socket



n-way associate

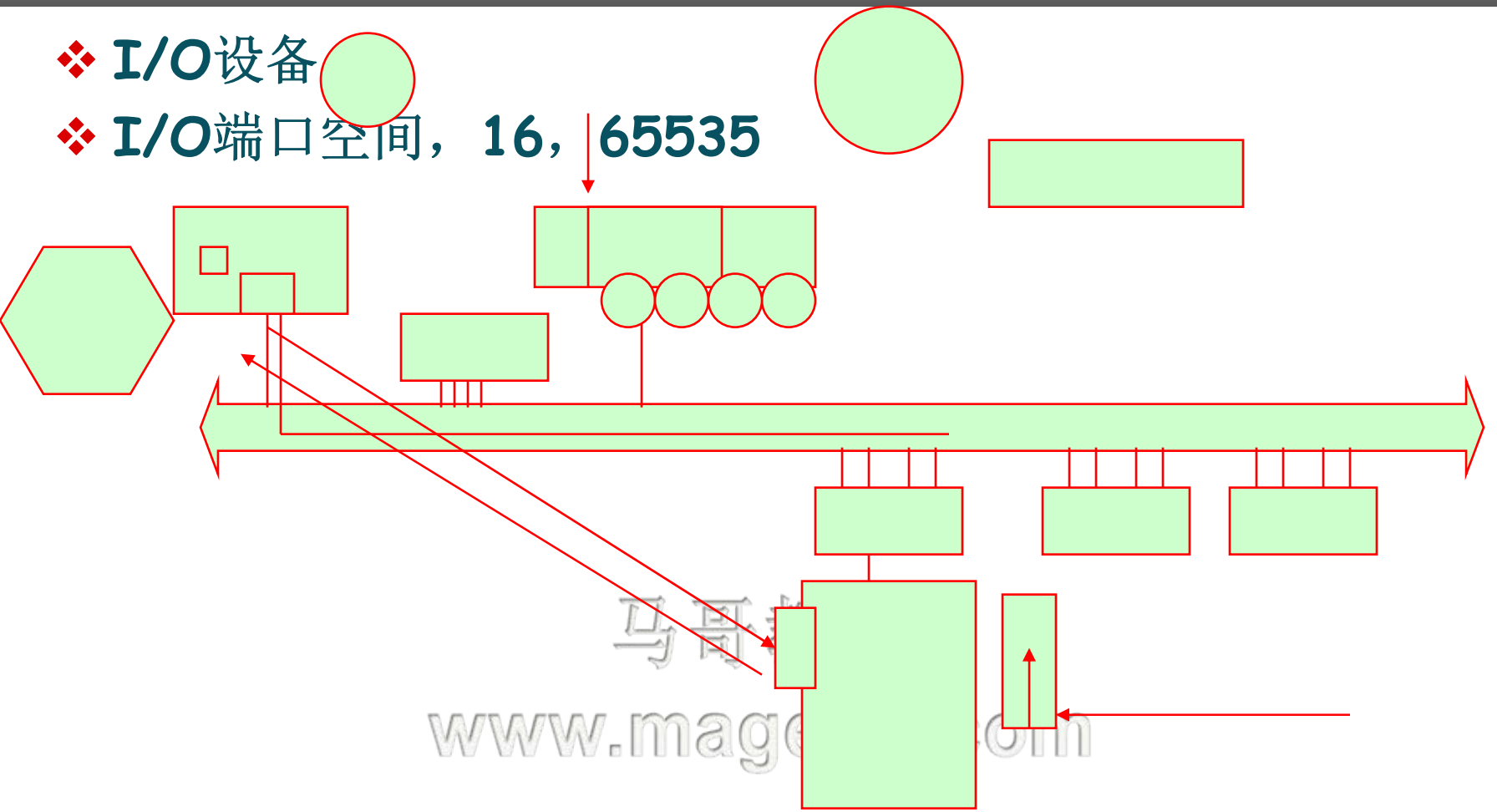


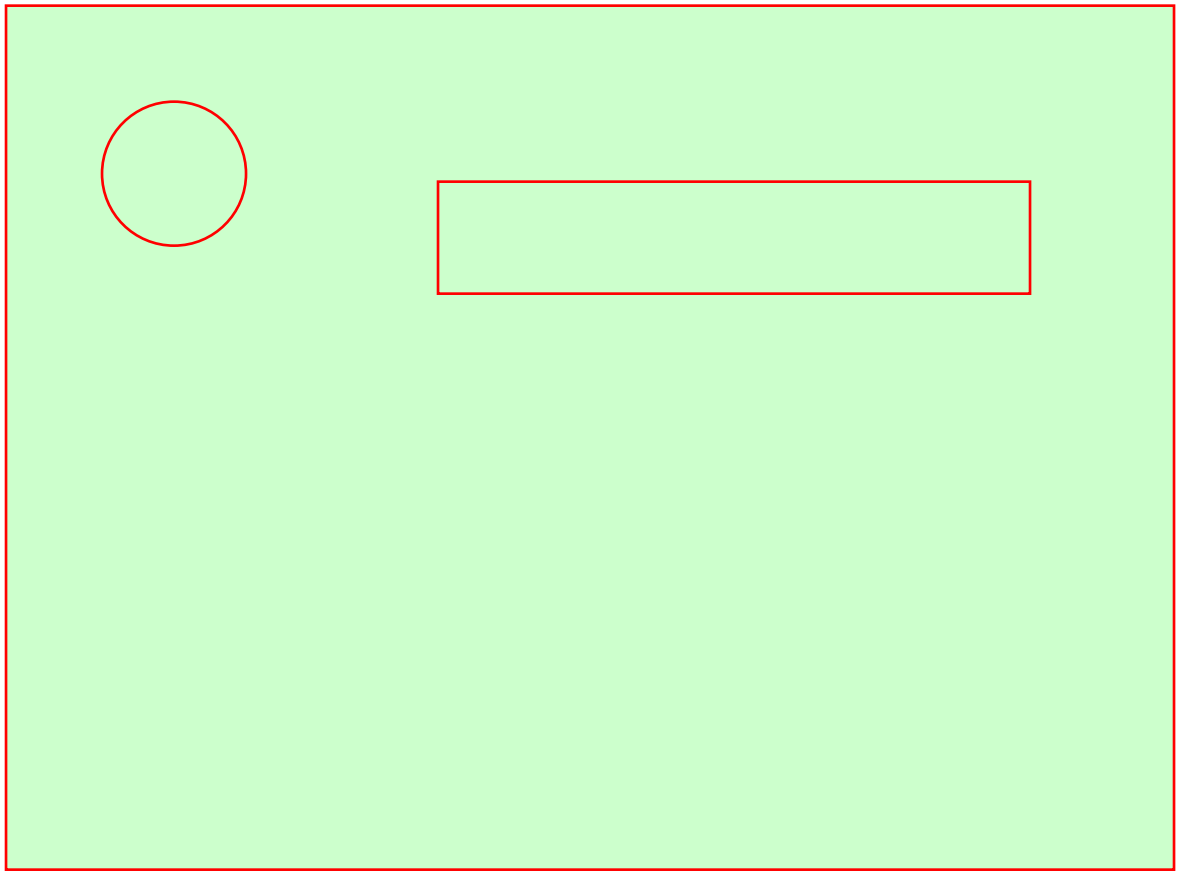


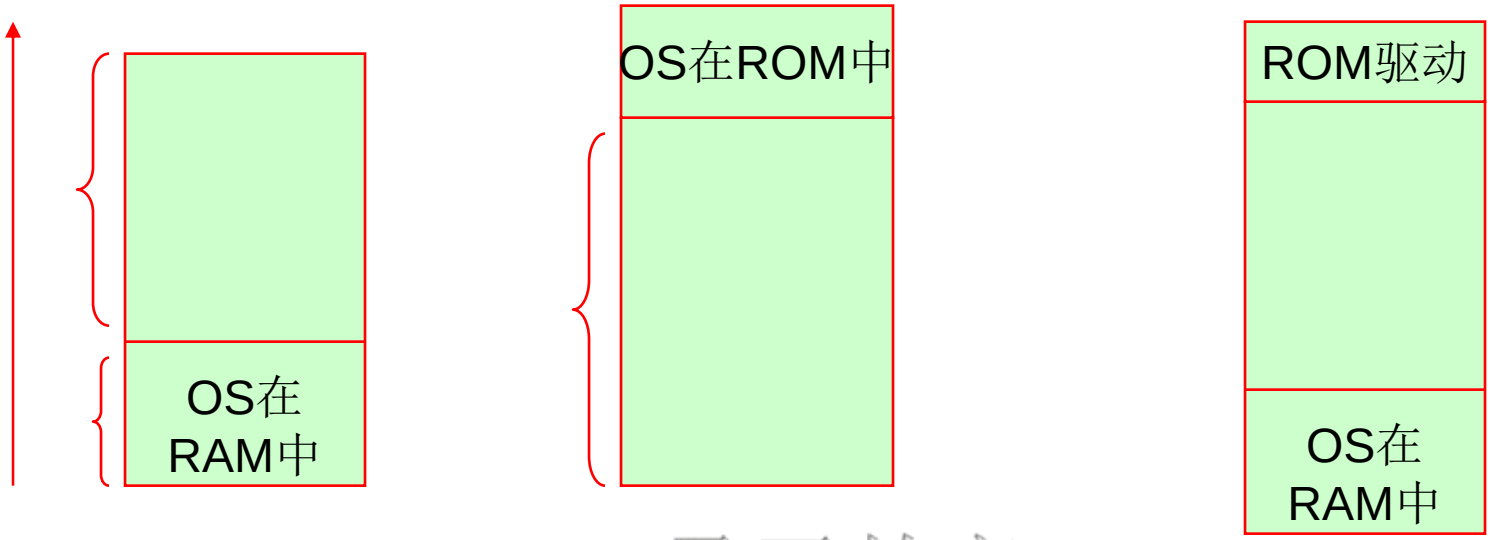


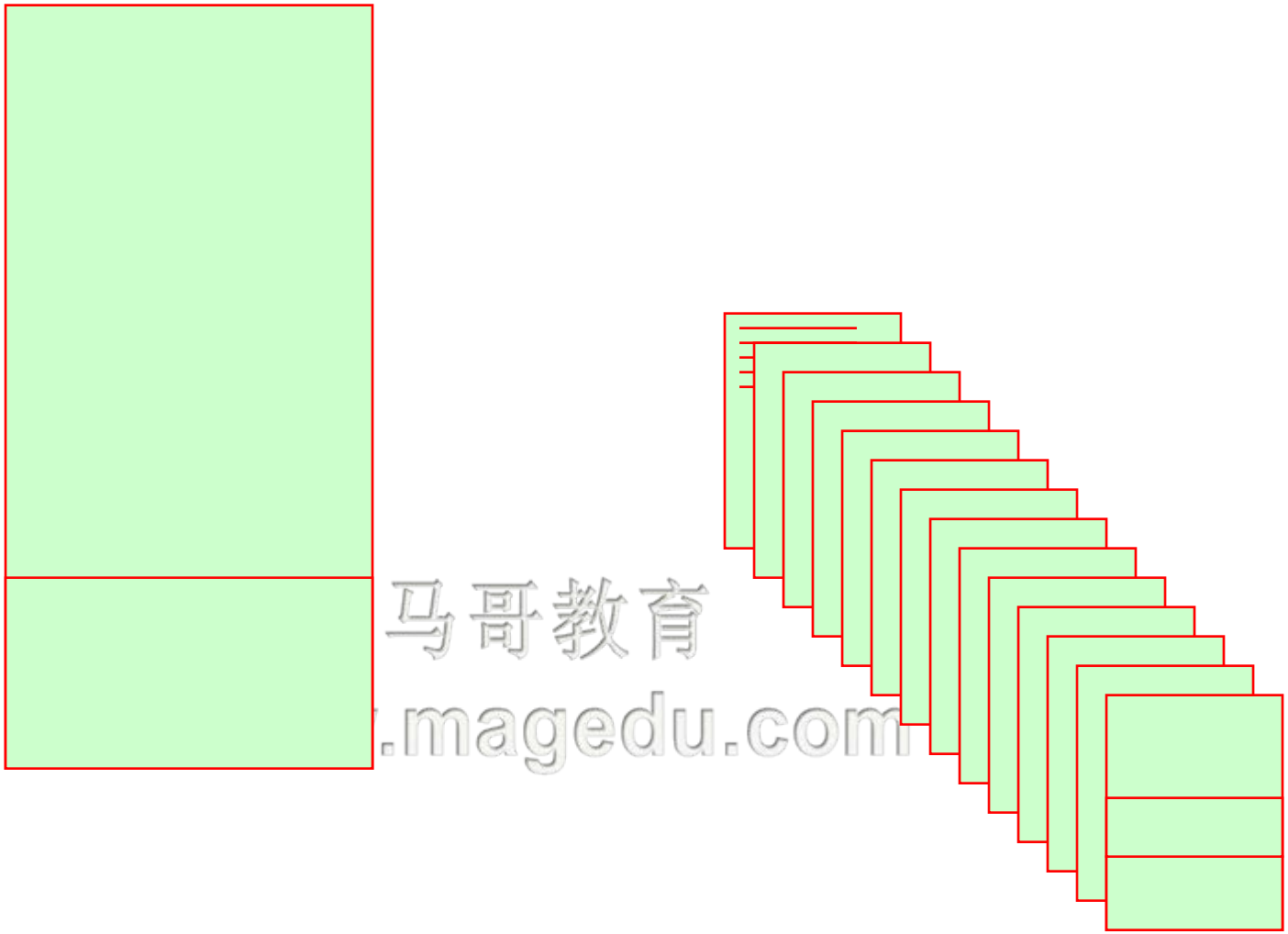
I/O设备

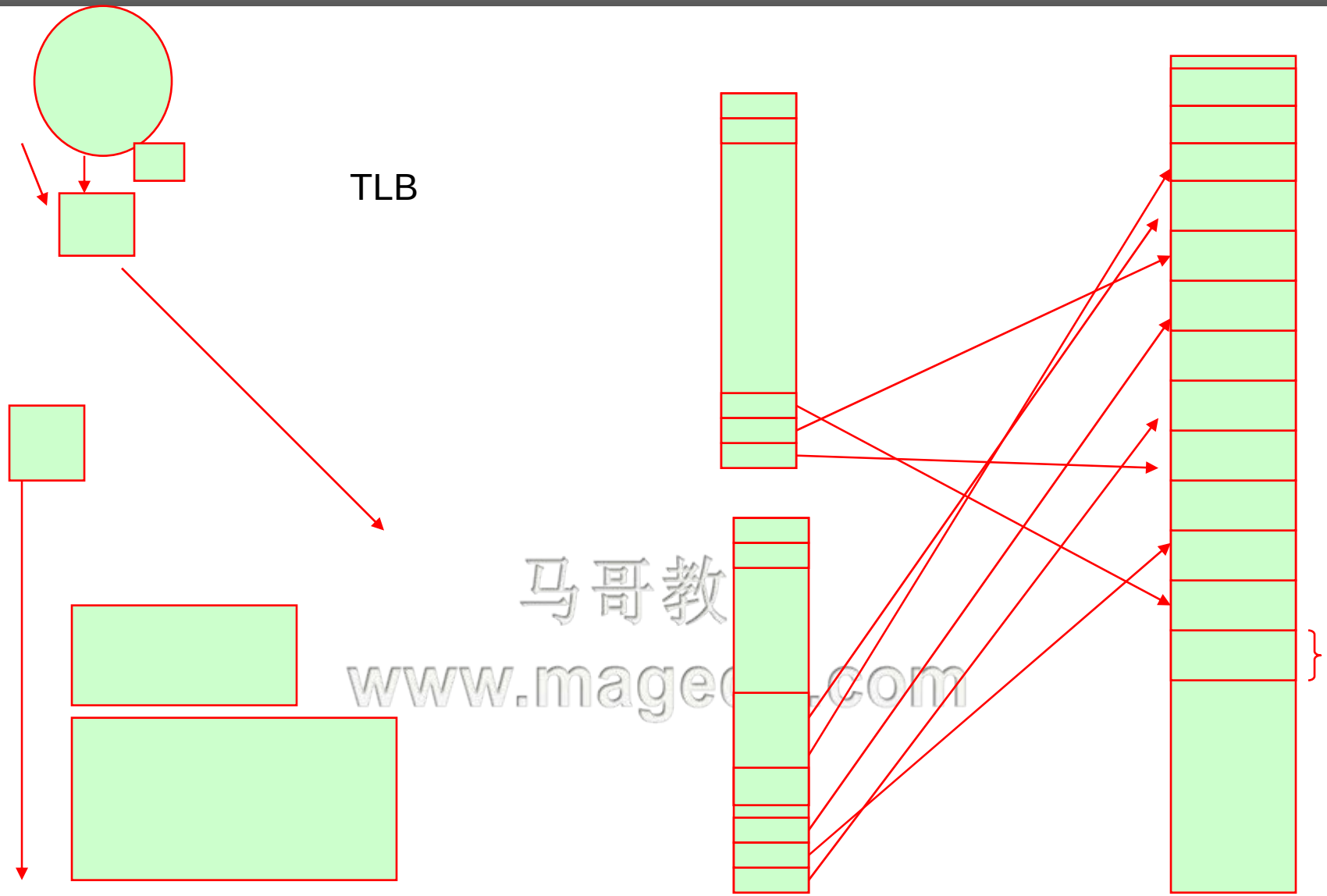
- I/O设备
- I/O端口空间，16，65535

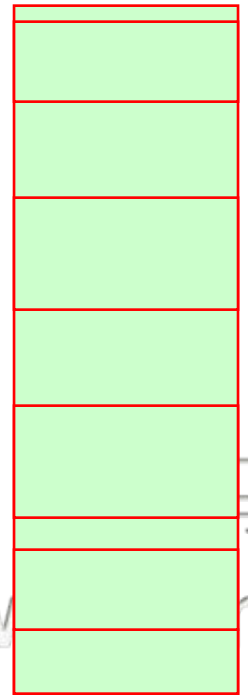




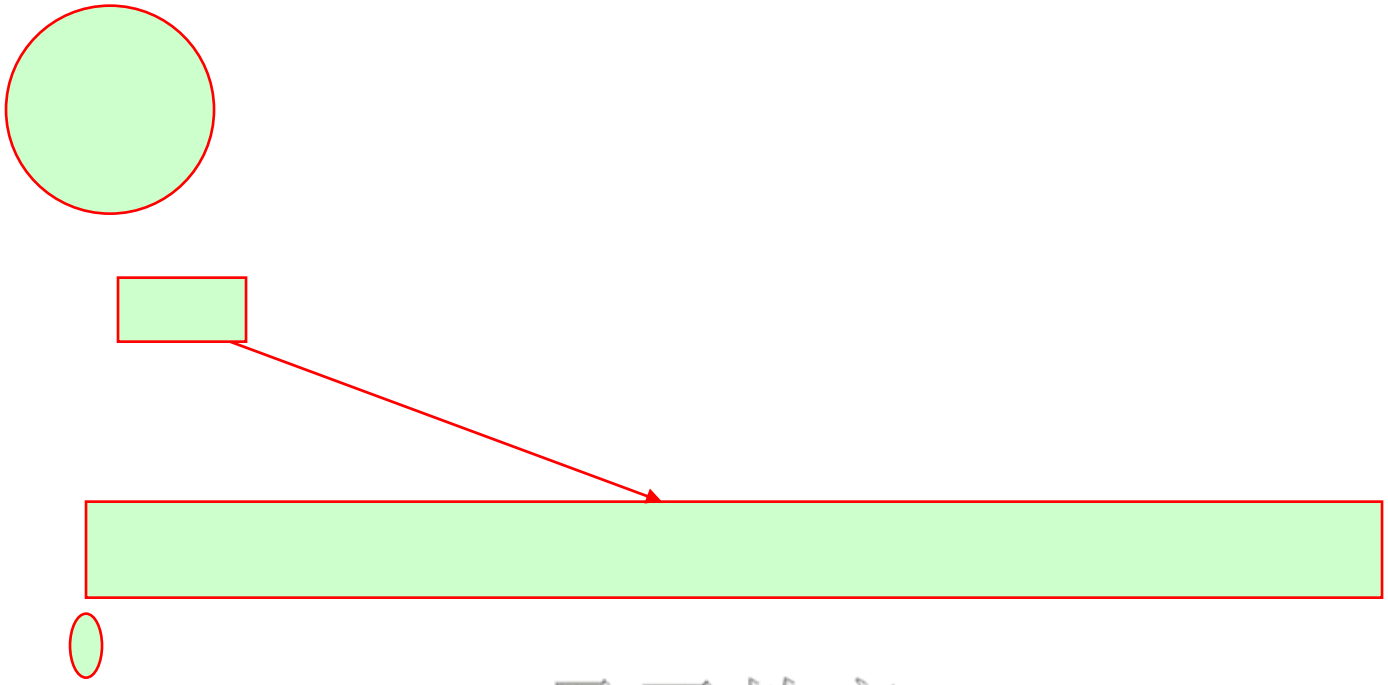




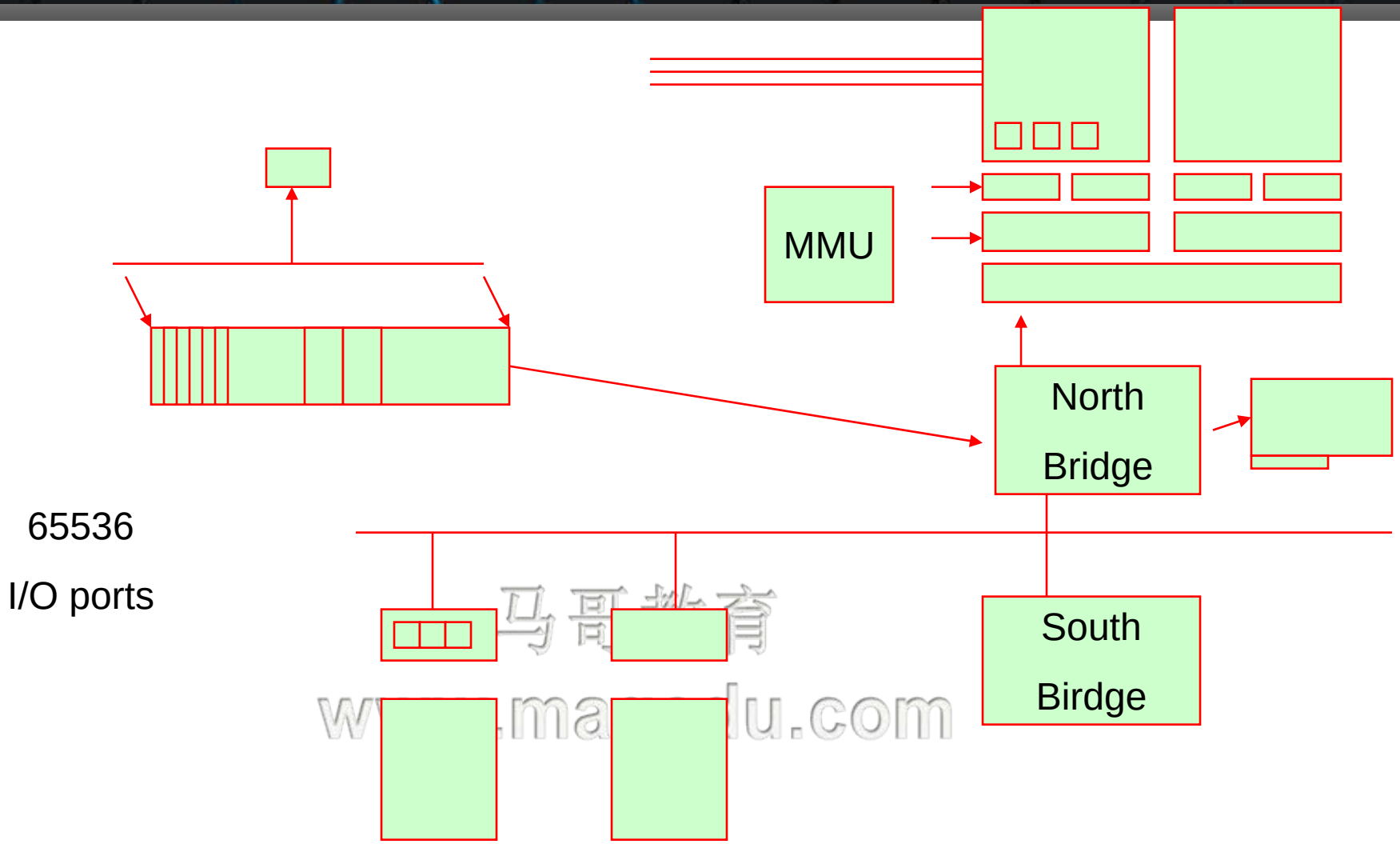


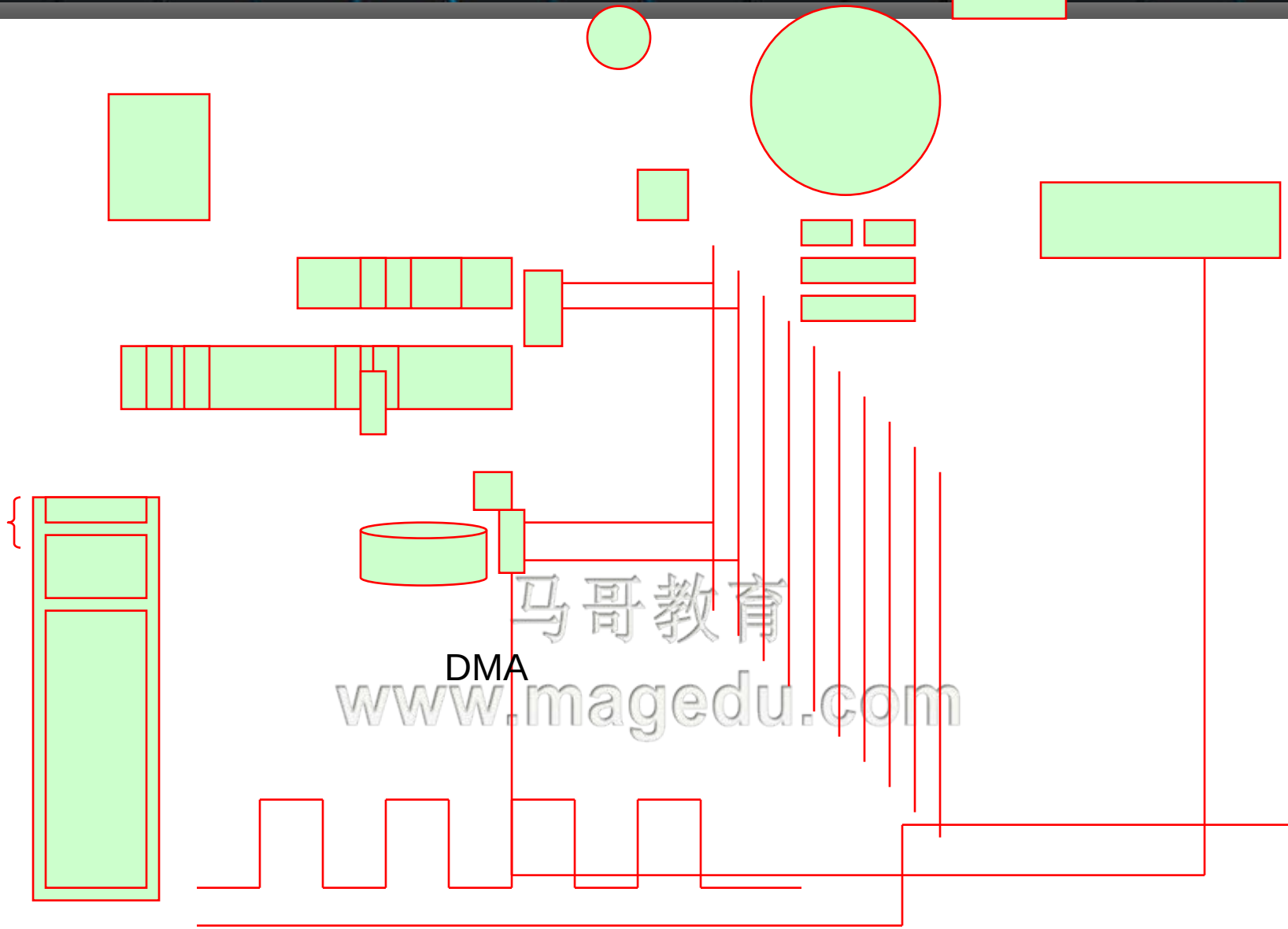


CPU 字长，32bit

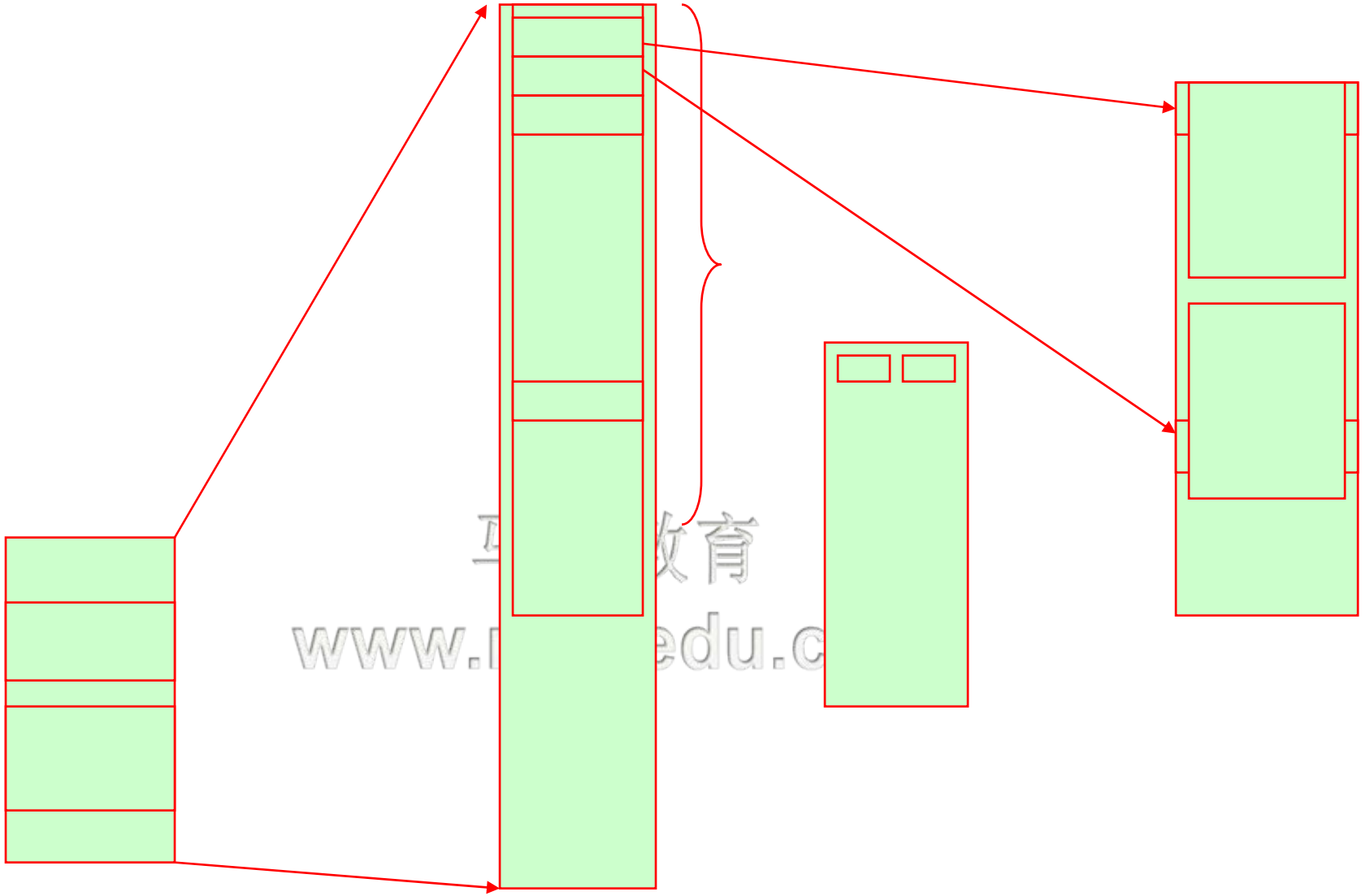


程序=指令+数据(32bit,4GBytes)

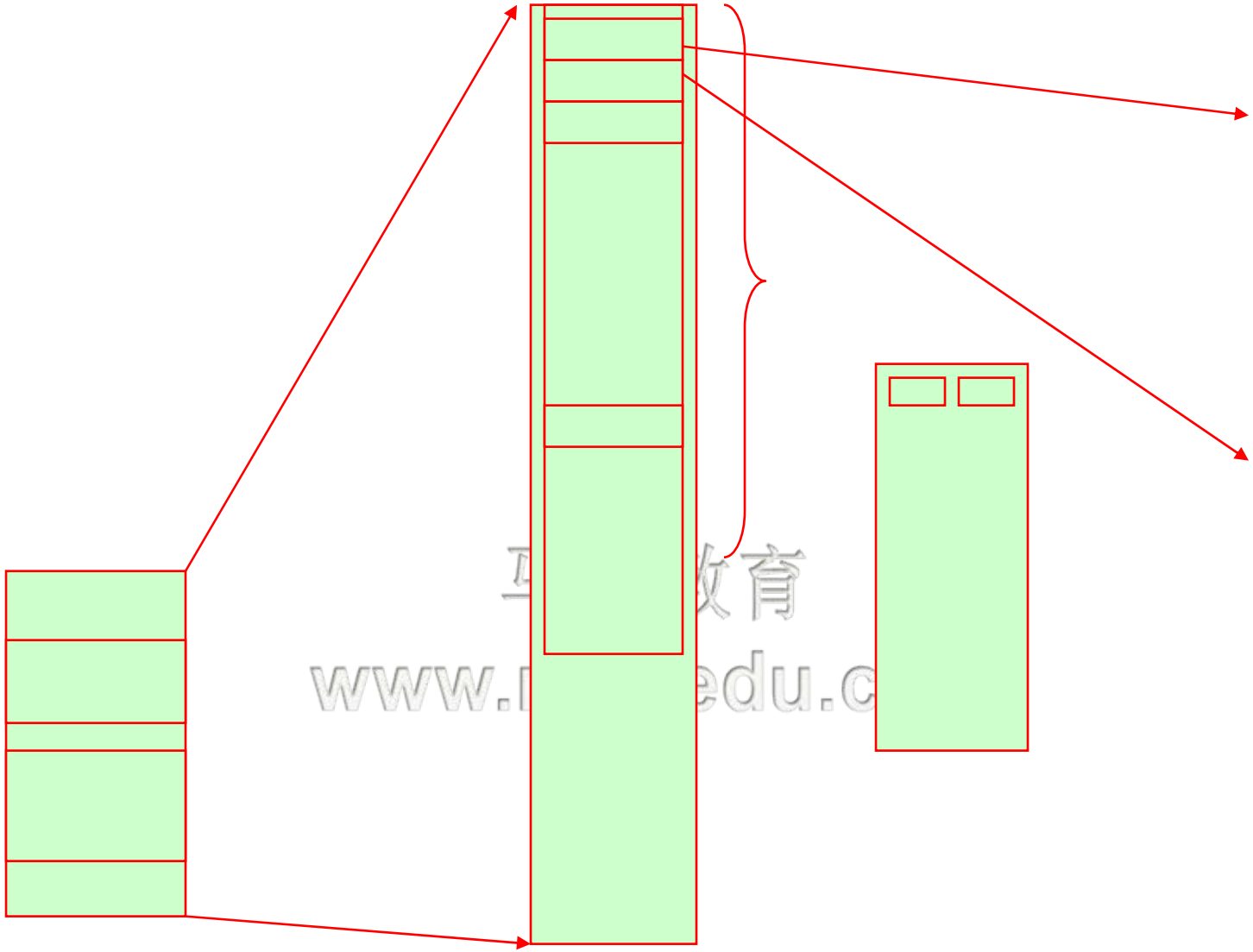


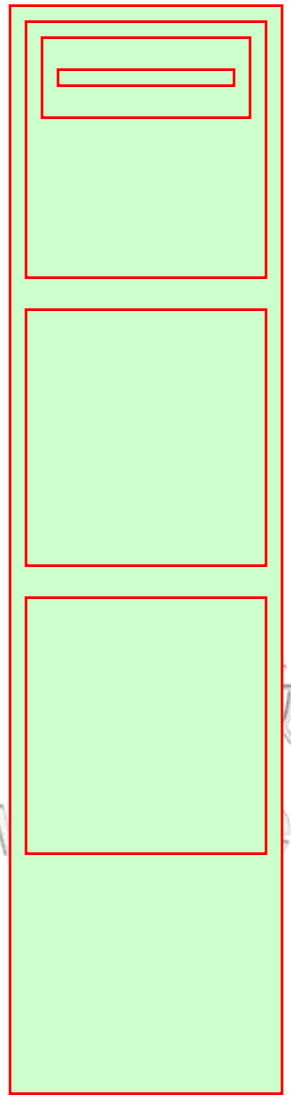


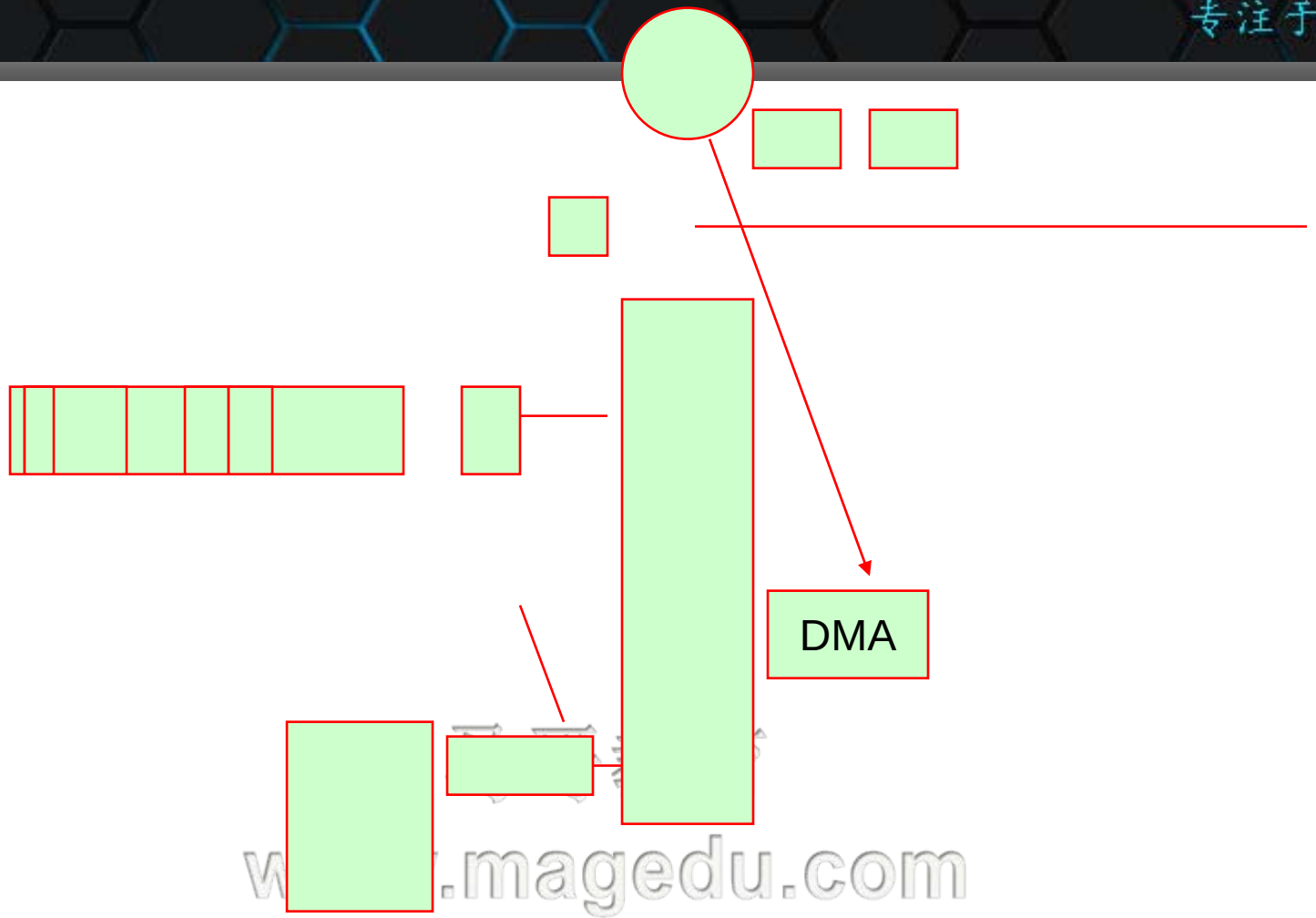
线性地址，task_struct

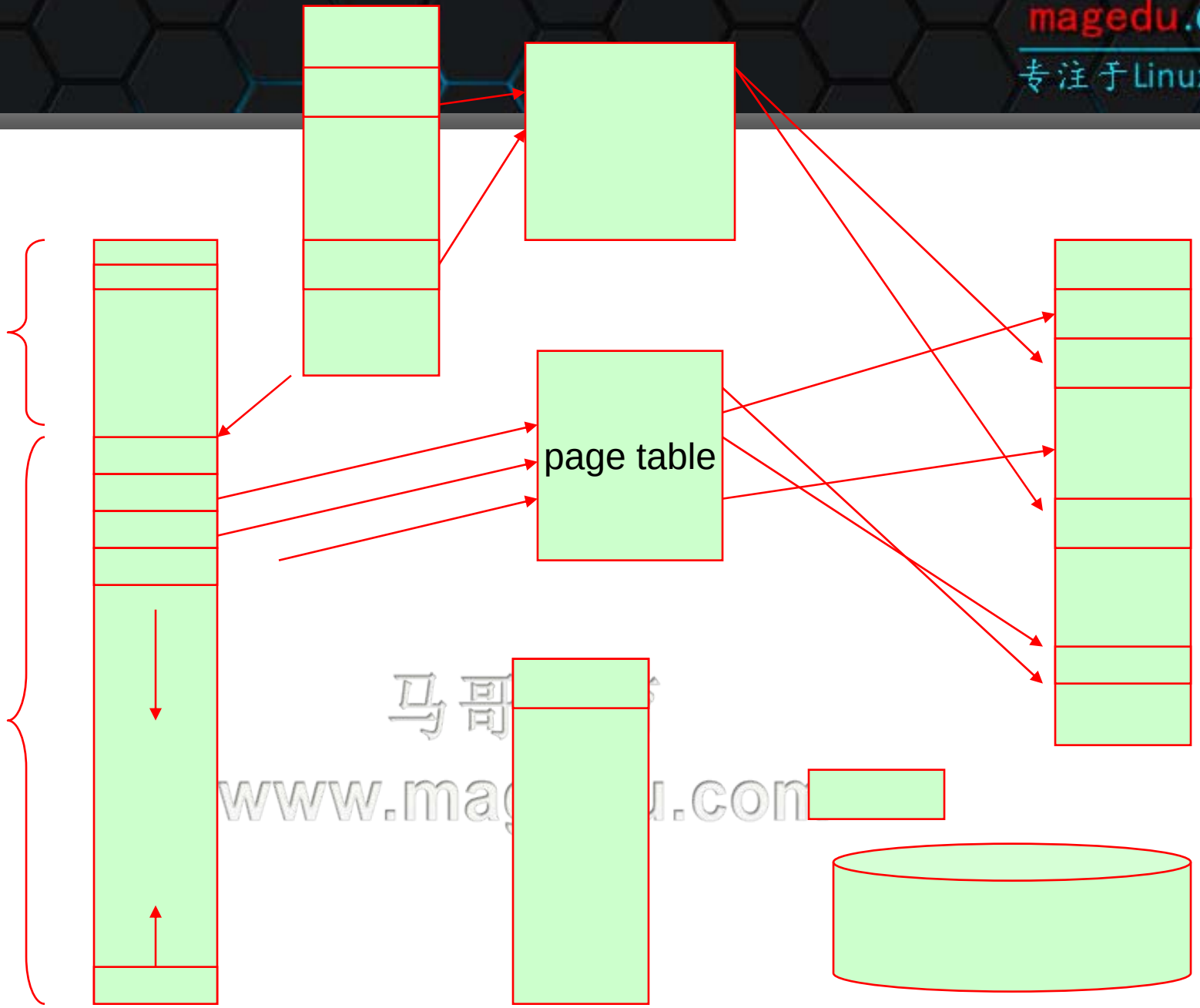


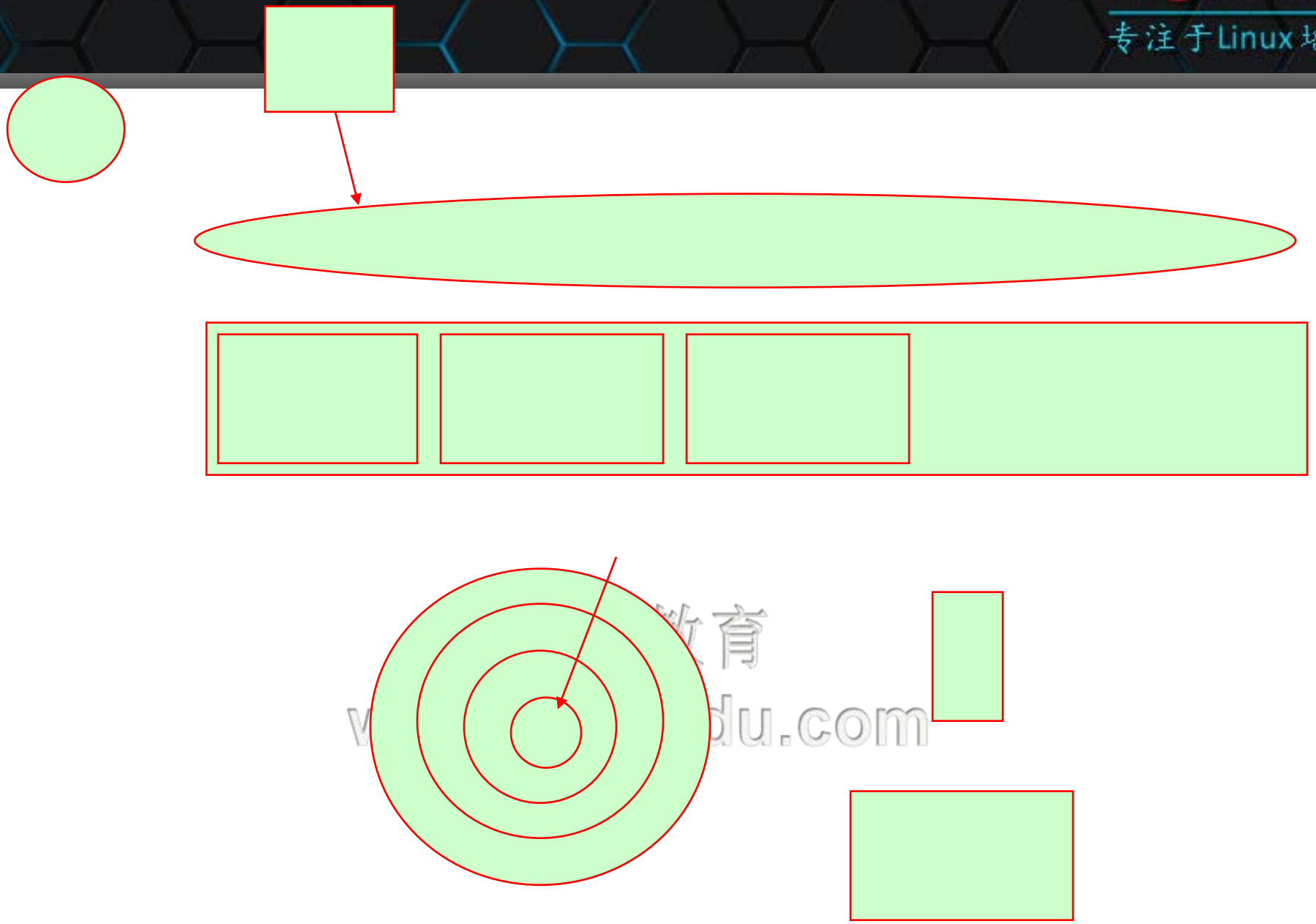
线性地址，task_struct

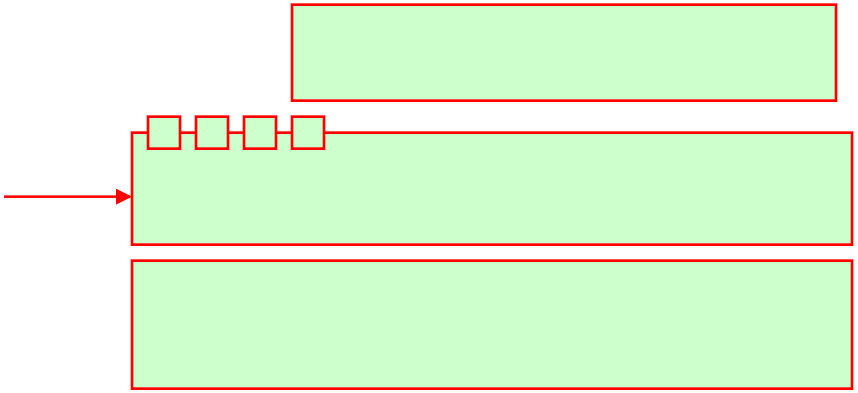


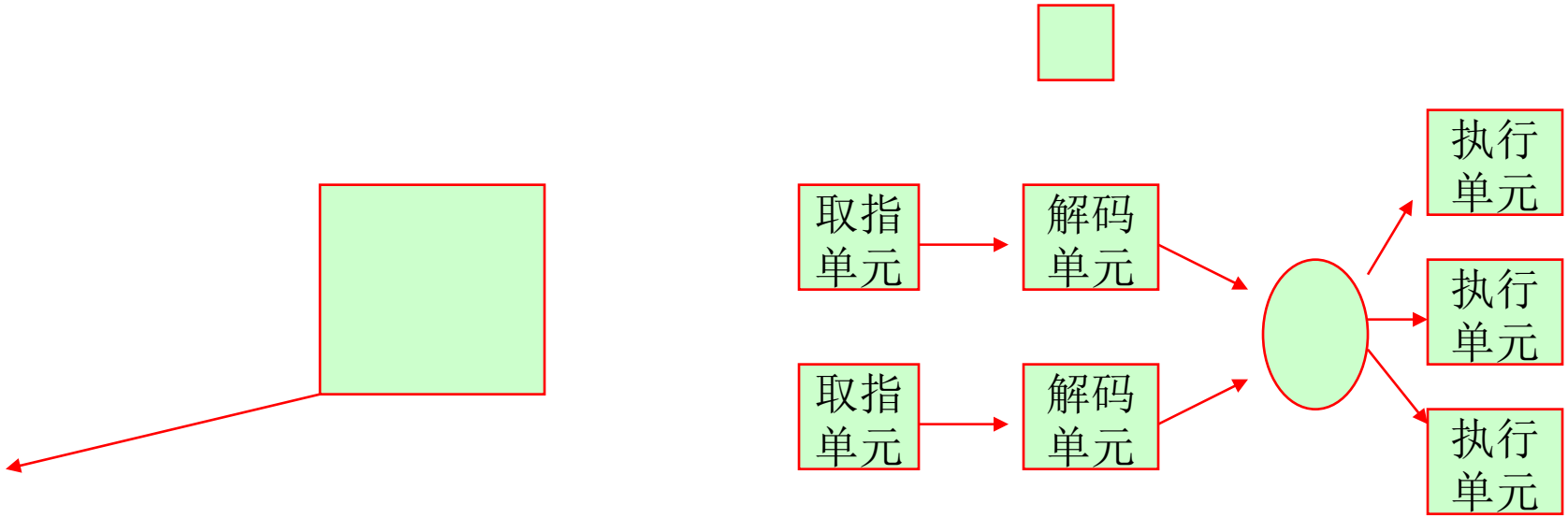












register

poll

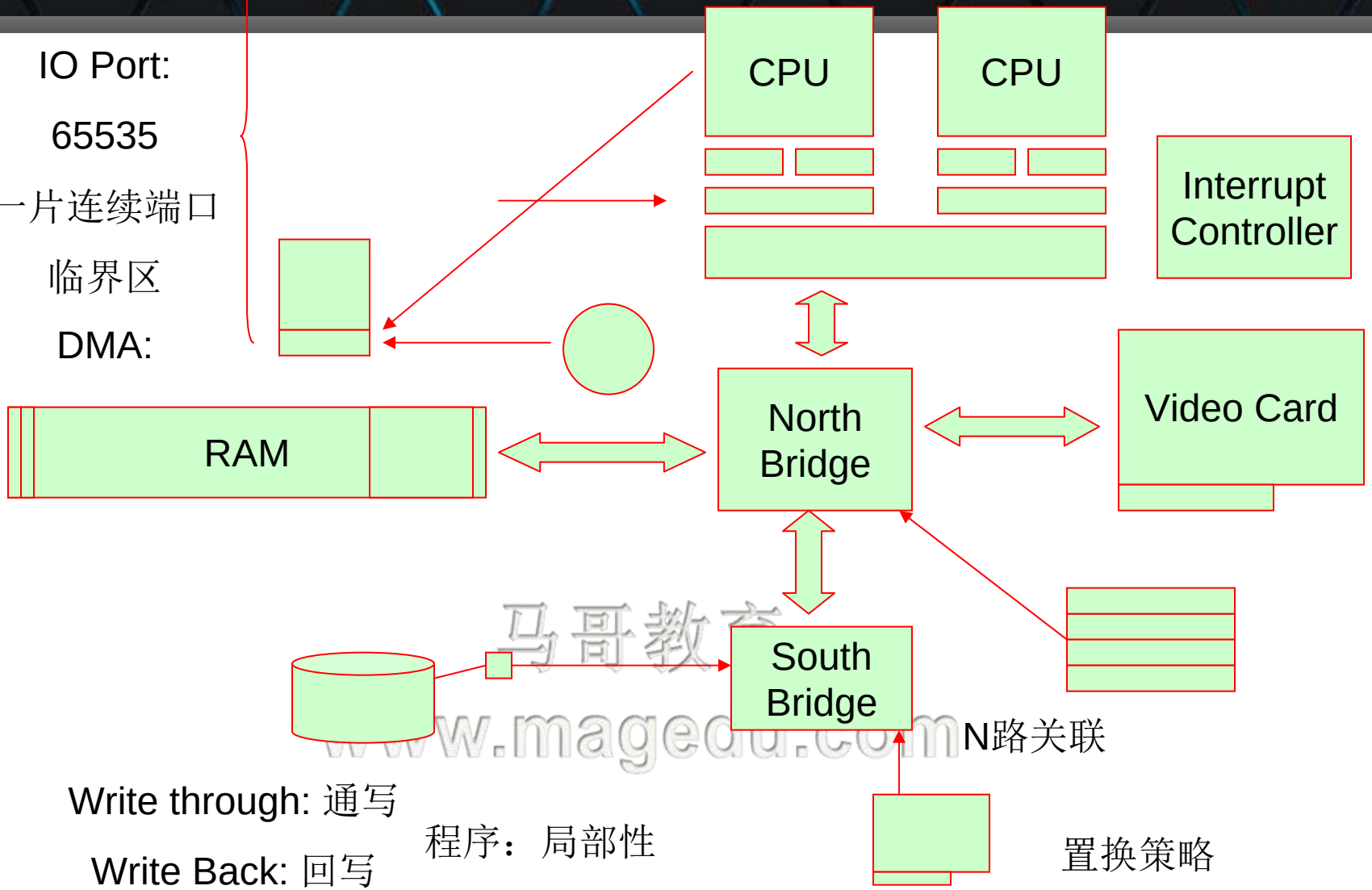
IO Port:

65535

一片连续端口

临界区

DMA:



N路关联

Write through: 通写

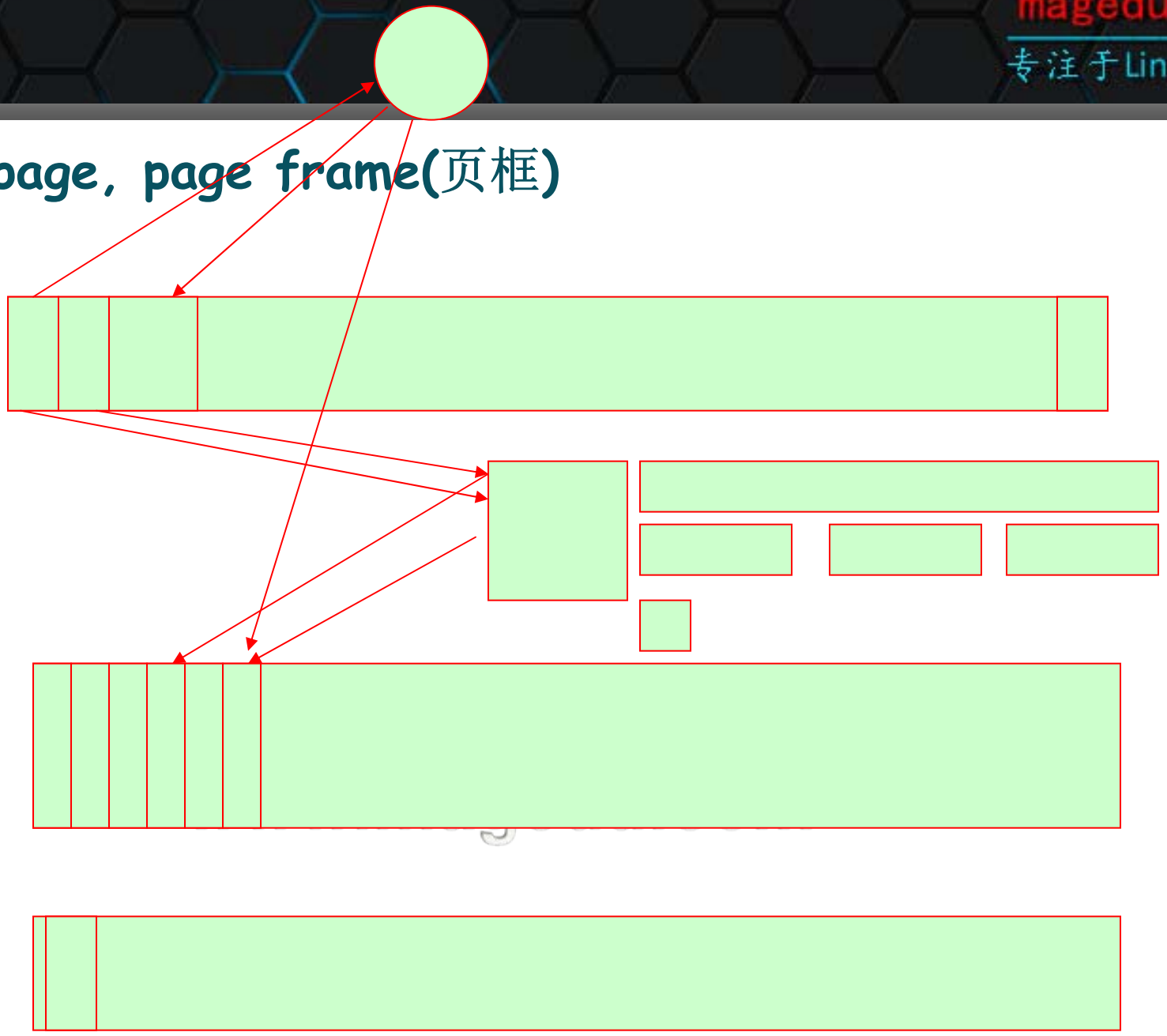
Write Back: 回写

程序：局部性

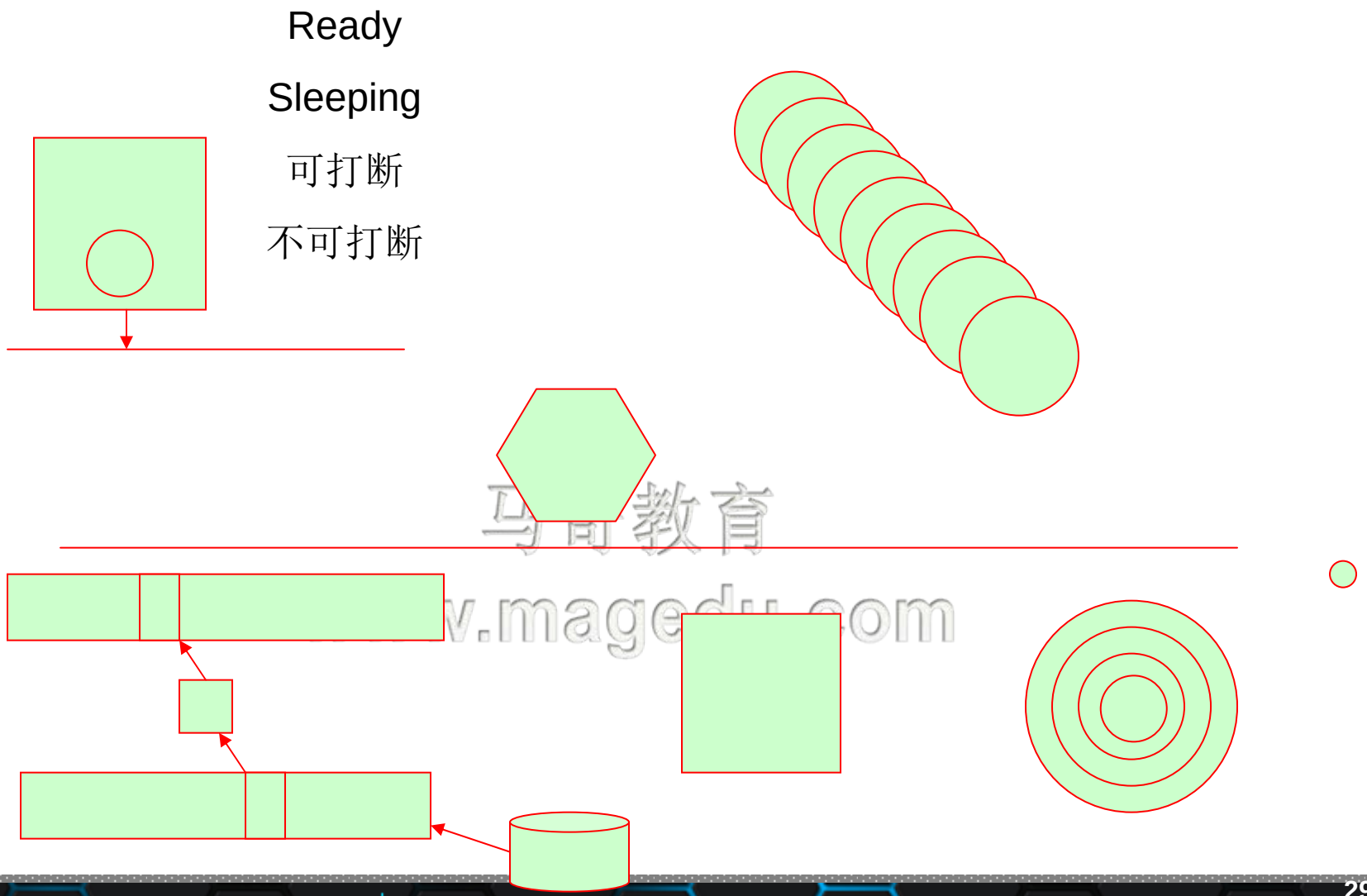
空间局部性、时间局部性

置换策略

- 4k，page，page frame(页框)



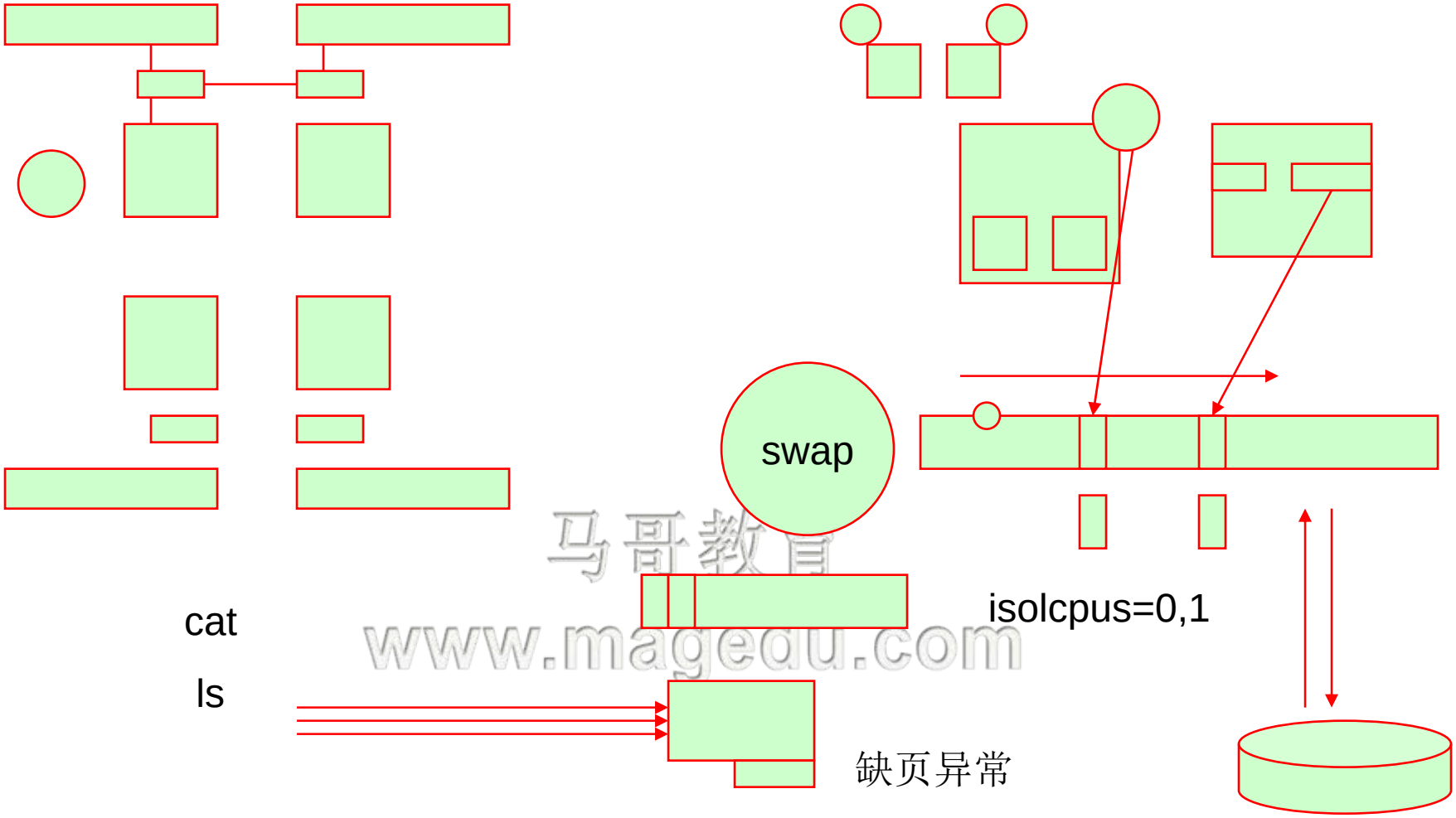
❖ 仿真

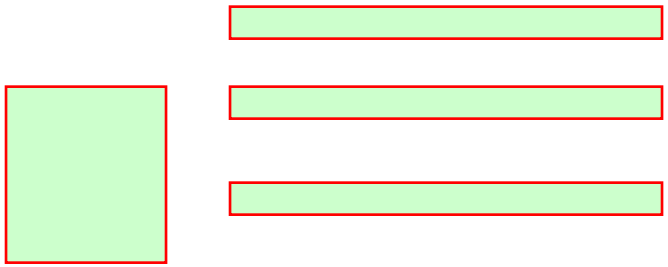


马哥教育

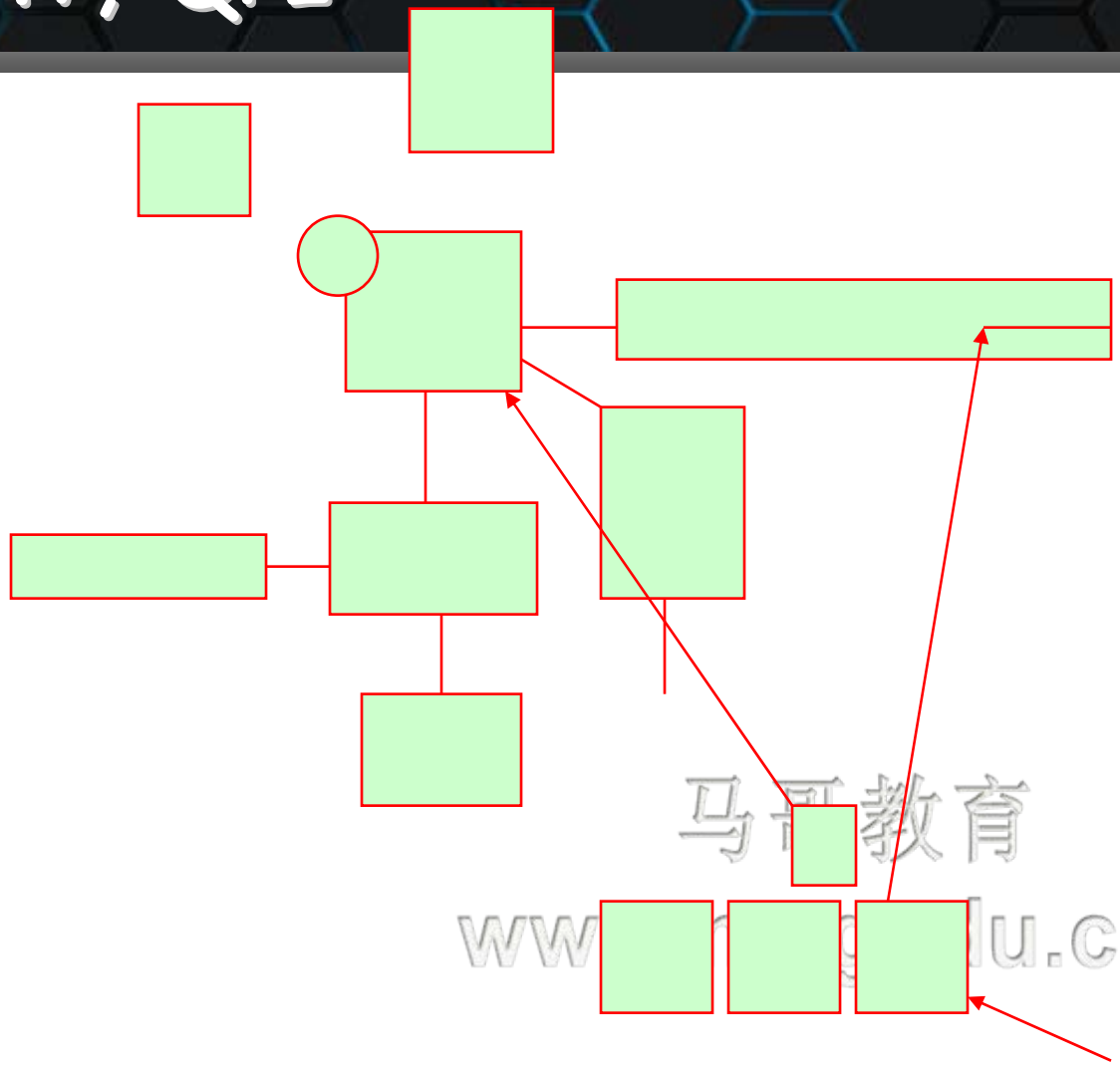
Linux System Performance Tuning

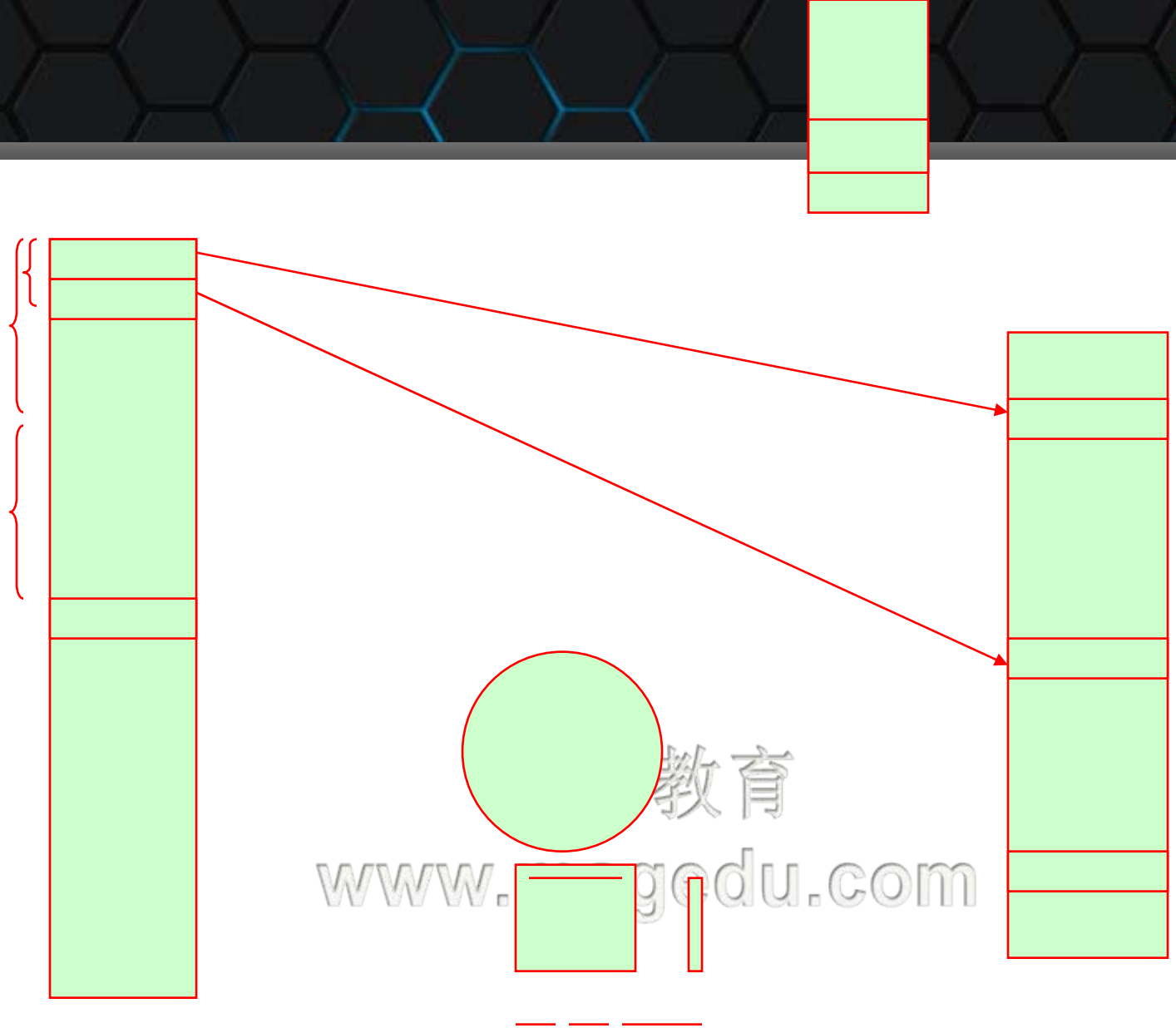
主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

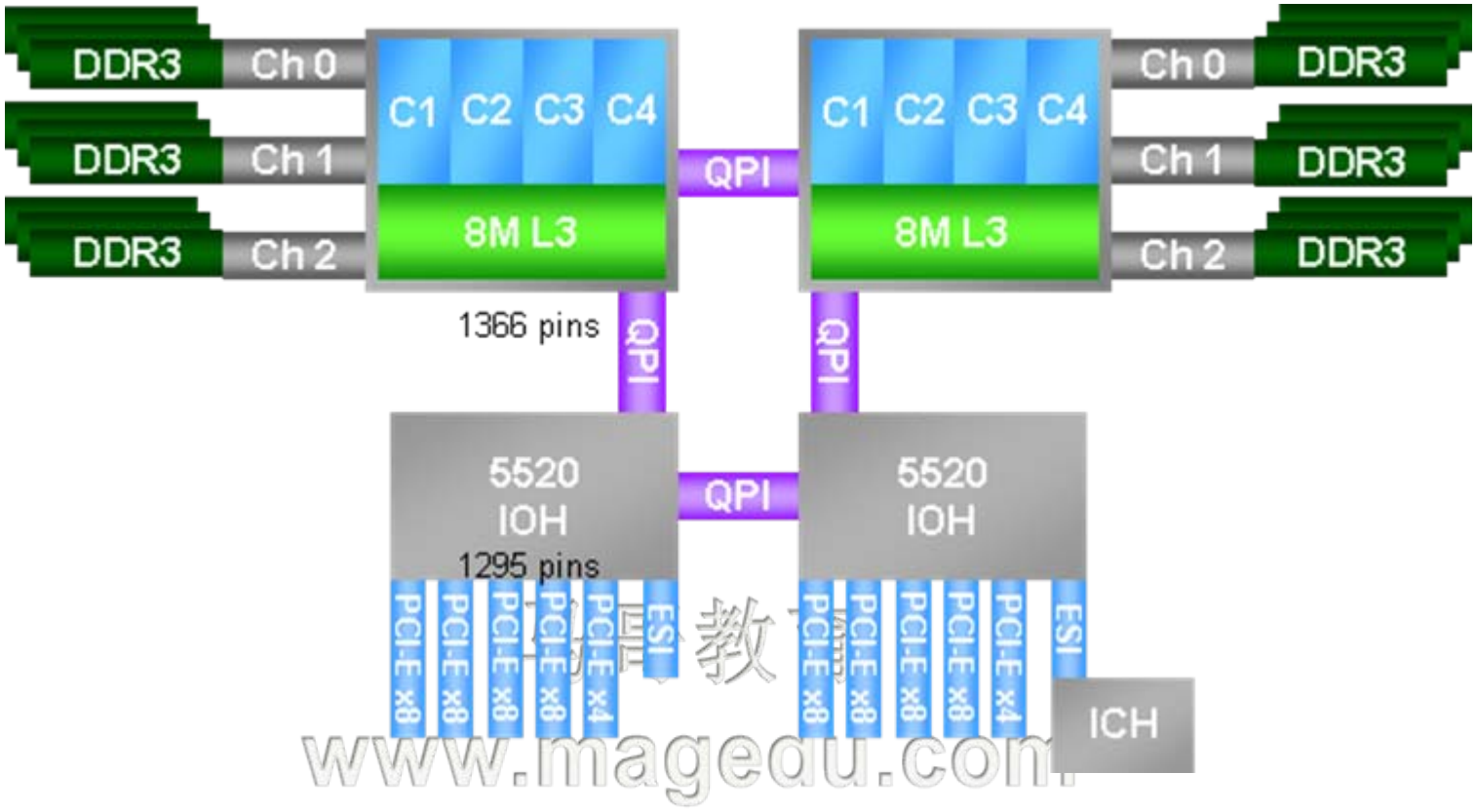




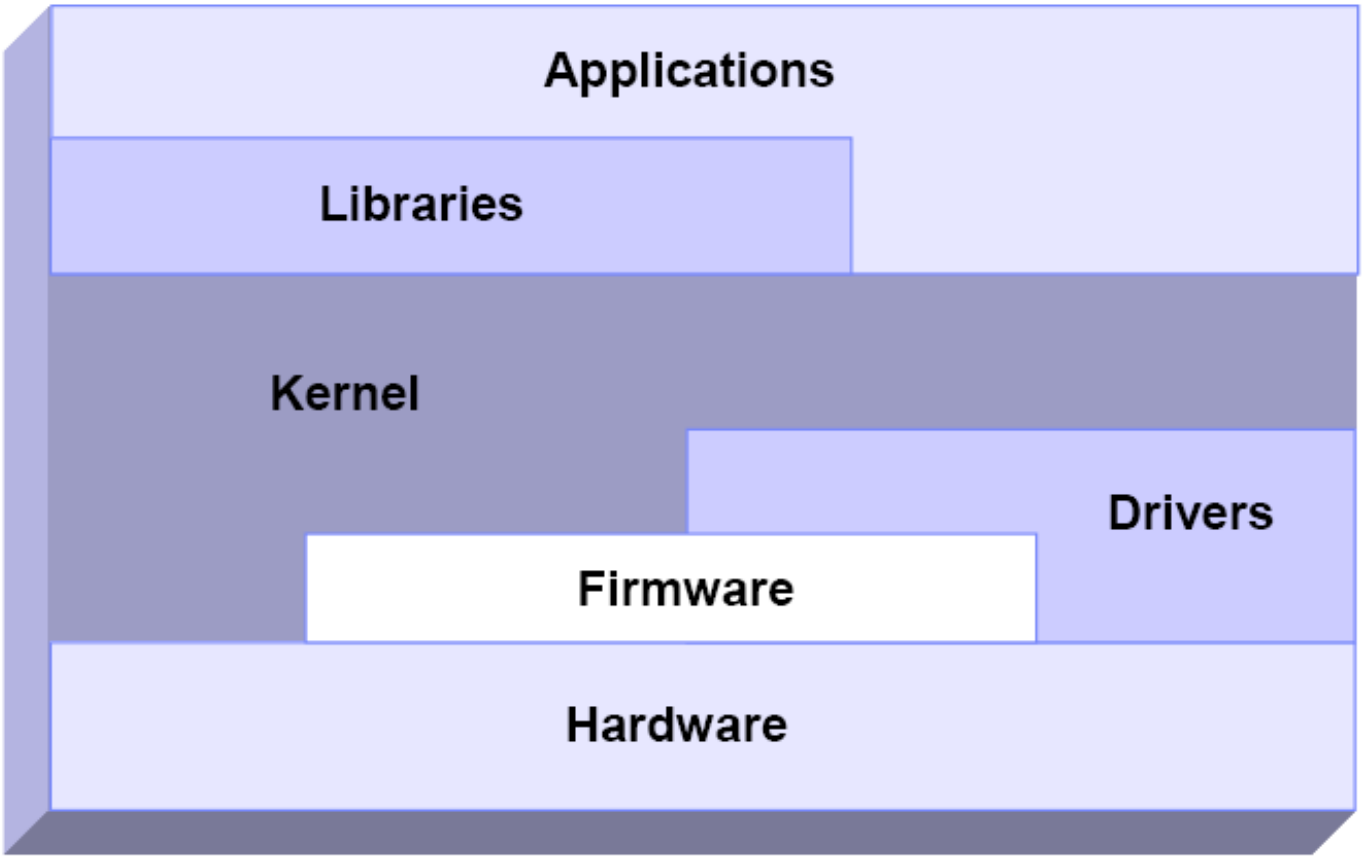
HT, QPI



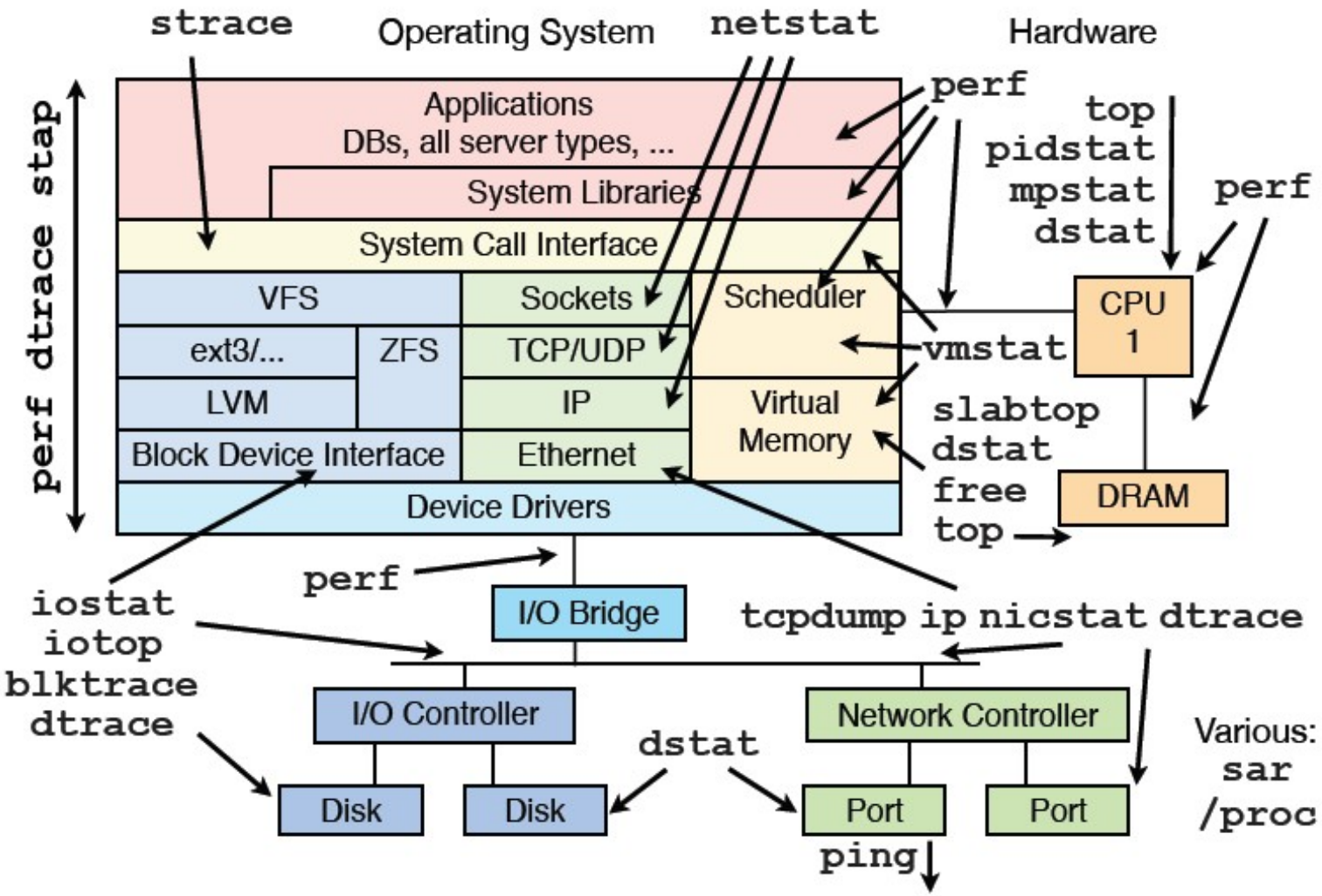


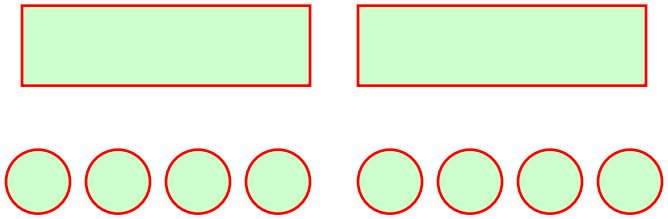


Schematic interaction of different performance components



Linux常用性能调优工具索引





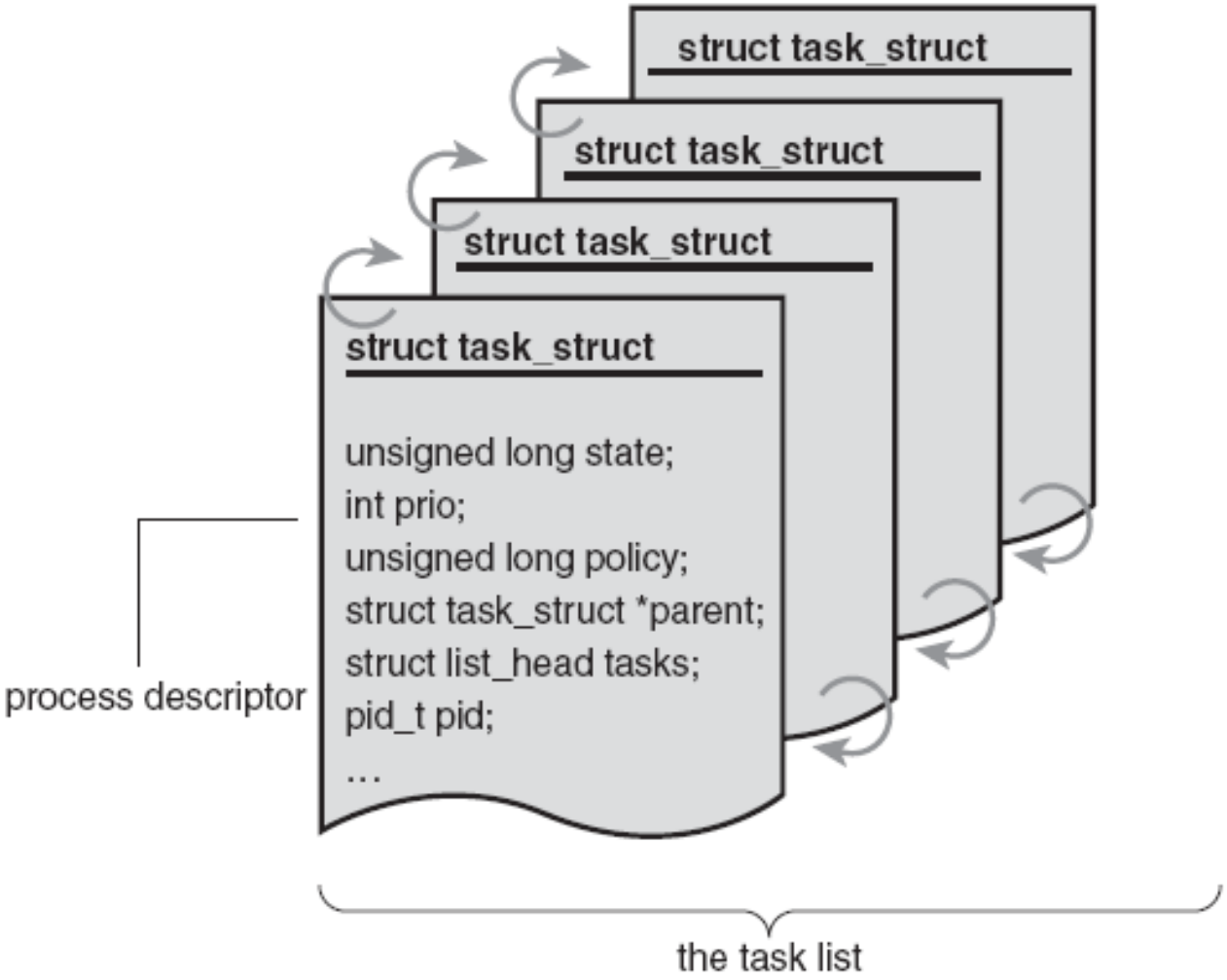
马哥教育

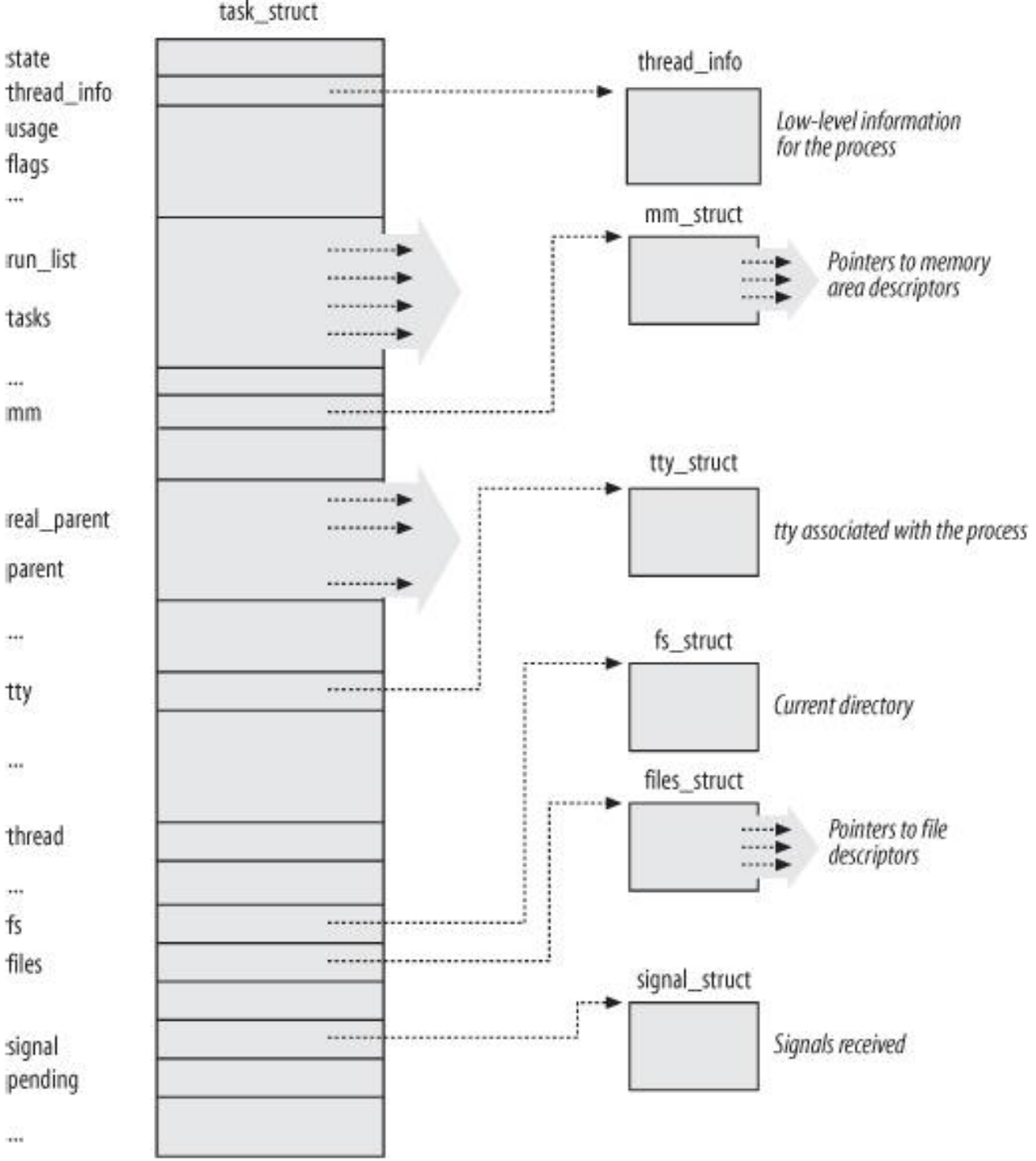
Linux process management

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

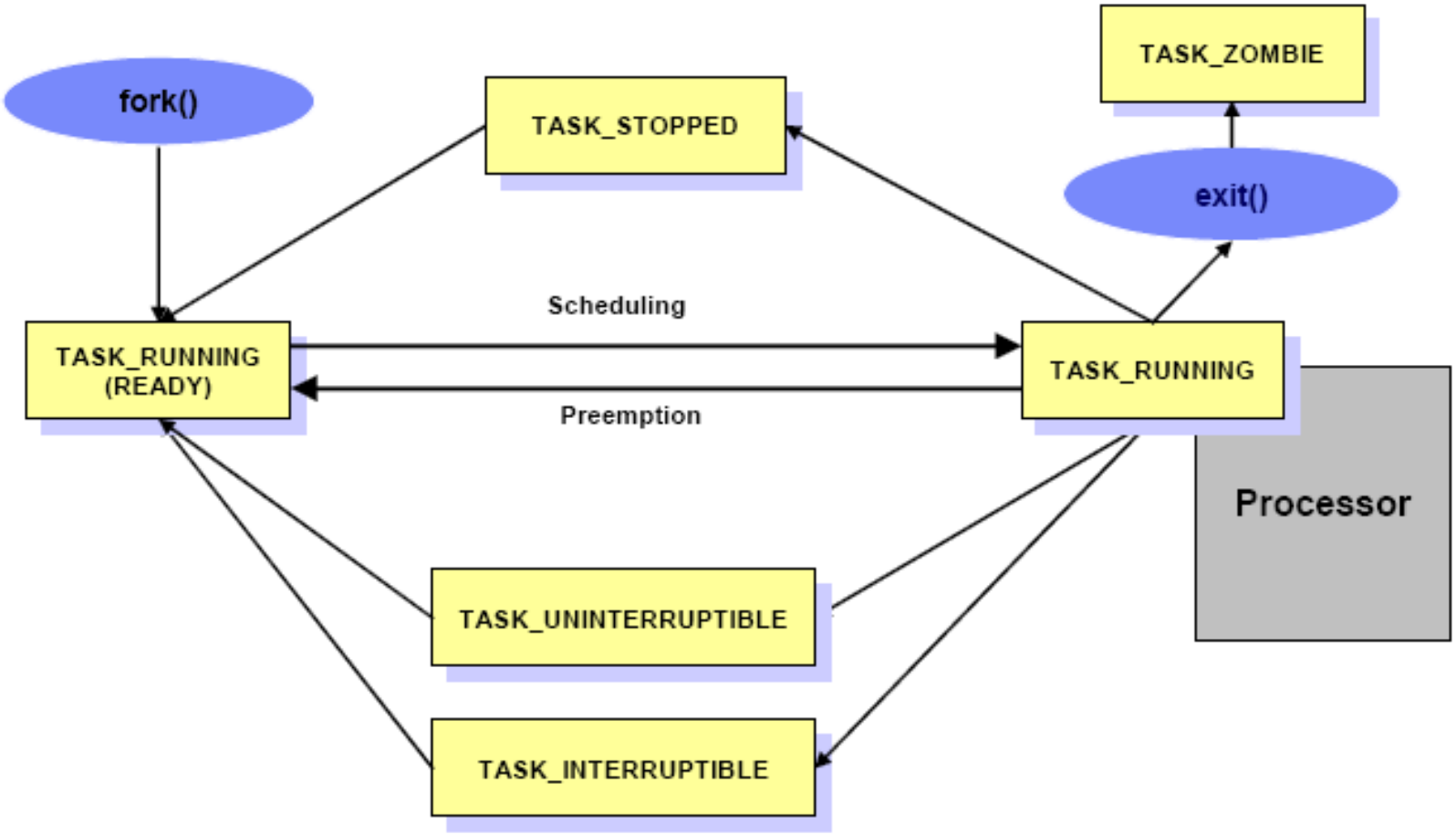
The process descriptor and task list

- List
  - Tasl



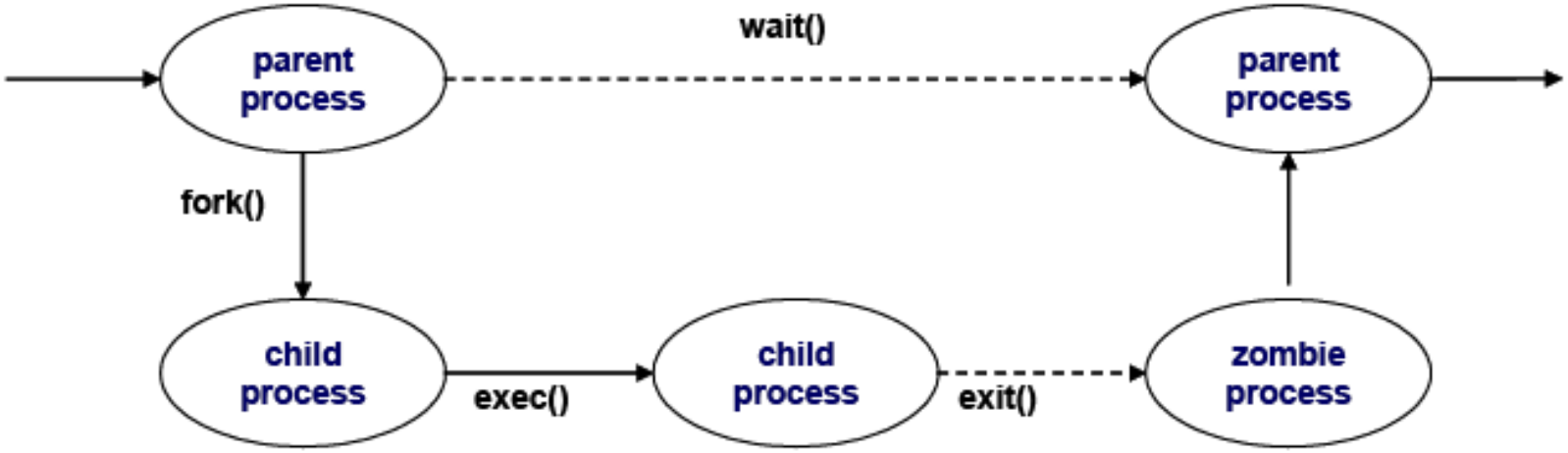

task_struct
state
thread_info
usage
flags
...
run_list
tasks
...
mm
real_parent
parent
...
tty
...
thread
...
fs
files
signal
pending
...
thread_info
Low-level information for the process
mm_struct
Pointers to memory area descriptors
tty_struct
tty associated with the process
fs_struct
Current directory
files_struct
Pointers to file descriptors
signal_struct
Signals received

Process State



Life cycle of a process

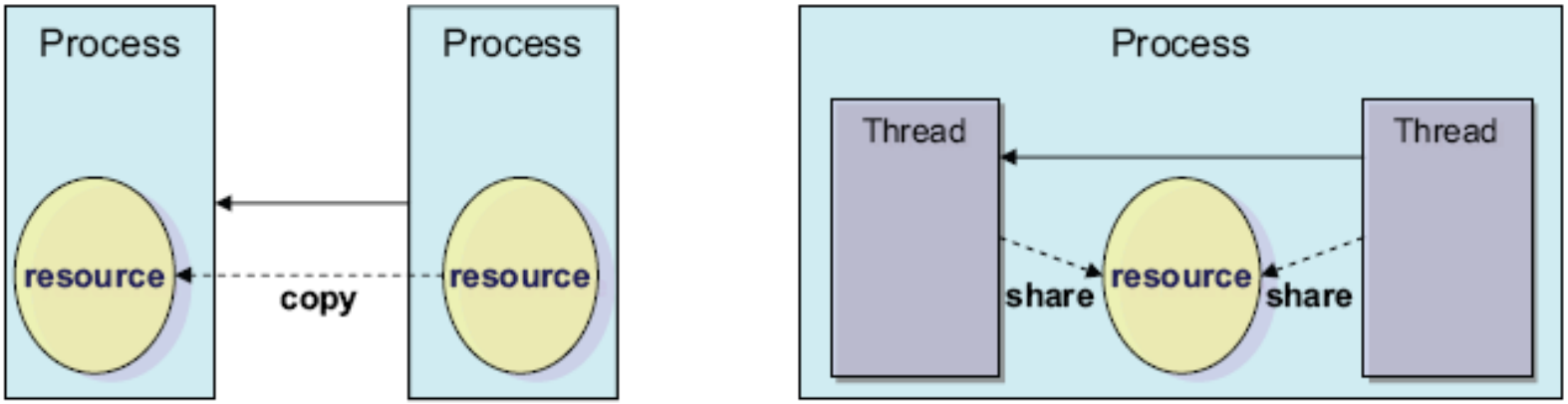
- Every process has its own life cycle such as creation, execution, termination, and removal



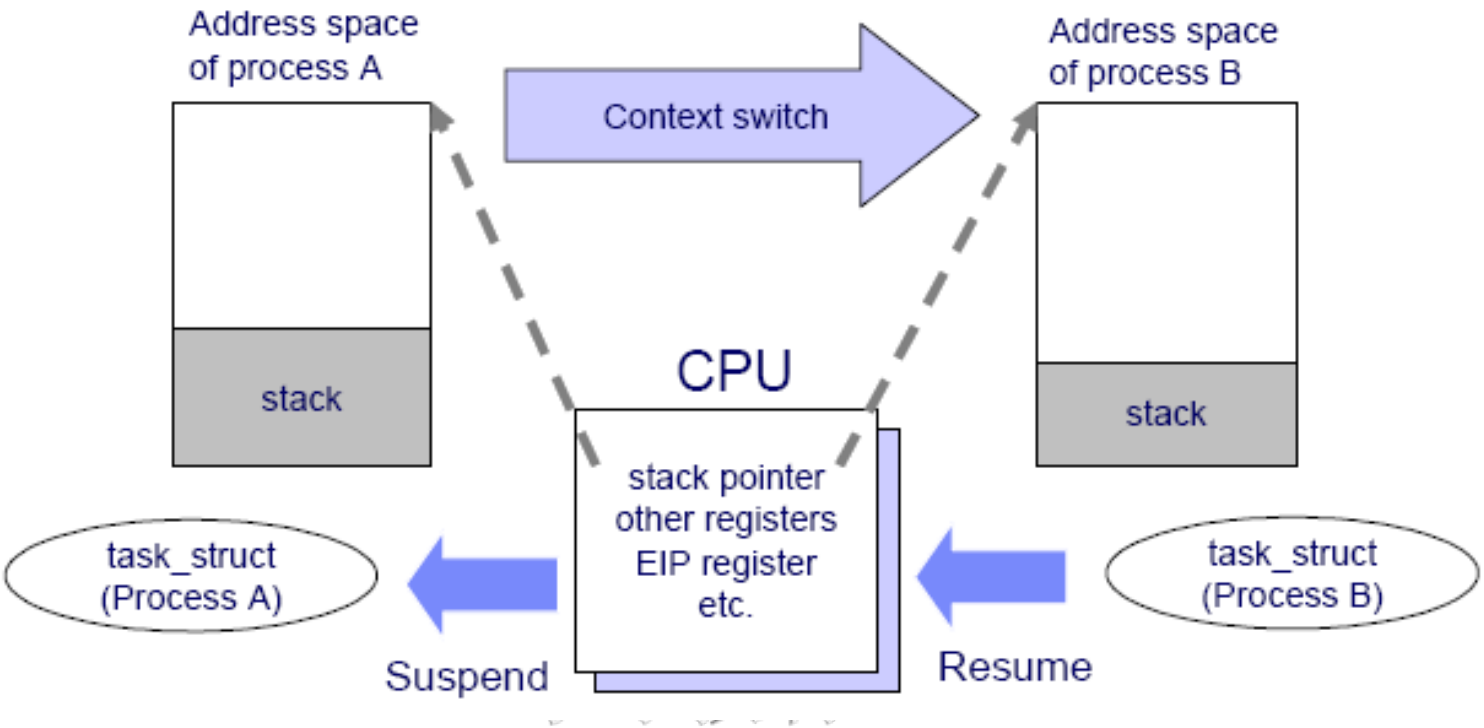
Thread

- A thread is an execution unit generated in a single process
  - It runs parallel with other threads in the same process
  - They can share the same resources such as memory, address space, open files, and so on
  - They can access the same set of application data
  - A thread is also called *Light Weight Process* (LWP)
  - Because they share resources, each thread should not change their shared resources at the same time
    - The implementation of mutual exclusion, locking, serialization, and so on, are the user application's responsibility

- From the performance perspective, thread creation is less expensive than process creation because a thread does not need to copy resources on creation



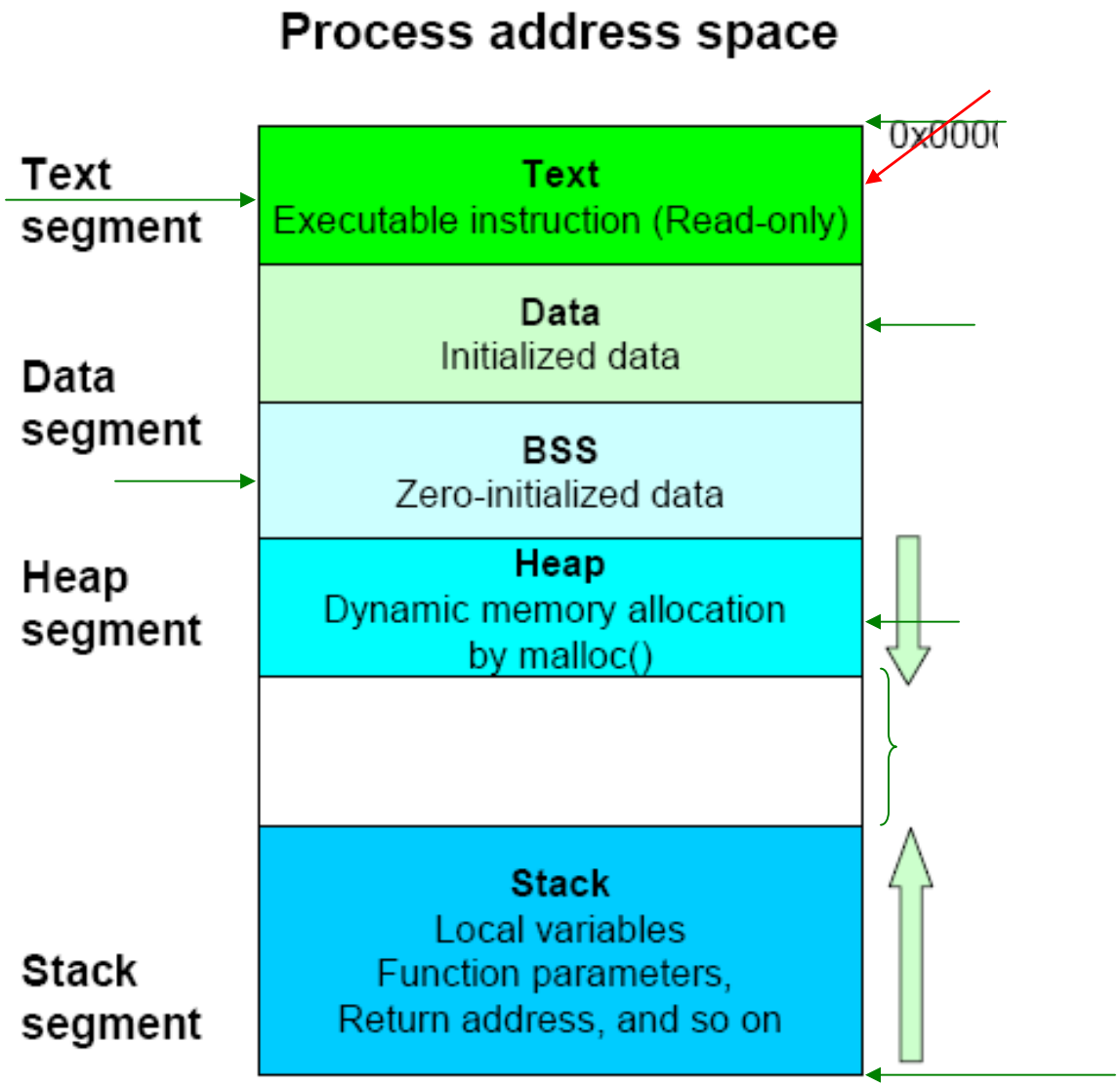
Context switching



Process memory segments

- A process uses its own memory area to perform work
- The work varies depending on the situation and process usage
  - A process can have different workload characteristics and different data size requirements
  - The process has to handle a of variety of data sizes
- To satisfy this requirement, the Linux kernel uses a dynamic memory allocation mechanism for each process

Process address space



Classes of processes

- Interactive processes
  - These interact constantly with their users, and therefore spend a lot of time waiting for keypresses and mouse operations
- Batch processes
  - These do not need user interaction, and hence they often run in the background
- Real-time processes
  - These have very strong scheduling requirements
  - Such processes should never be blocked by lower-priority processes, they should have a short response time and, most important, such response time should have a minimum variance
  - Typical real-time programs are video and sound applications, robot controllers, and programs that collect data from physical sensors

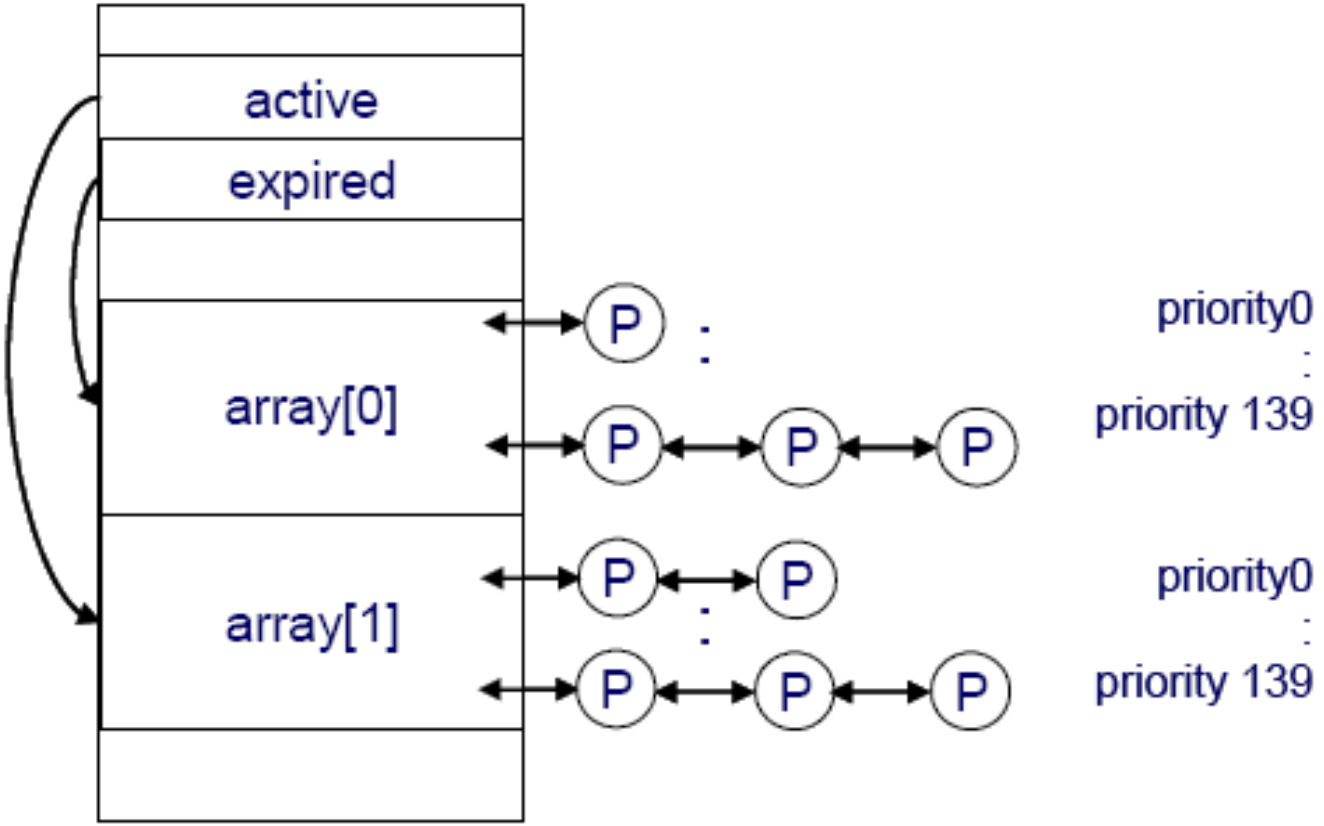
Process Scheduling

- Multitasking
  - Multitasking operating systems come in two flavors: *cooperative multitasking* and *preemptive multitasking*
  - Linux, like all Unix variants and most modern operating systems, implements preemptive multitasking
  - In preemptive multitasking, the scheduler, one of the kernel subsystem, decides when a process is to cease running and a new process is to begin running
    - The act of involuntarily suspending a running process is called *preemption*

Linux's Process Scheduler

- O(1) scheduler
  - It scales well with the number of runnable processes, because it selects the process to run in constant time, independently of the number of runnable processes
  - It also scales well with the number of processors because each CPU has its own queue of runnable processes
  - The new algorithm does a better job of distinguishing interactive processes and batch processes
  - But, O(1) scheduler had several pathological failures related to scheduling latency-sensitive applications
    - Thus, although the O(1) scheduler was ideal for large server workloads—which lack interactive processes—it performed below par on desktop systems, where interactive applications are the *raison d'être*

Linux kernel 2.6 O(1) scheduler



- Linux scheduling is based on the *time sharing* technique
  - Several processes run in "time multiplexing" because the CPU time is divided into slices, one for each runnable process
  - Time sharing relies on timer interrupts and is thus transparent to processes
- The scheduling policy is also based on ranking processes according to their priority

Process Preemption

- If a process enters the TASK_RUNNING state, the kernel checks whether its dynamic priority is greater than the priority of the currently running process
  - If it is, the execution of current is interrupted and the scheduler is invoked to select another process to run
  - a process may also be preempted when its time quantum expires
    - when this occurs, the need_resched field of the current process is set, so the scheduler is invoked when the timer interrupt handler terminates

Preempting the current process

- Standard preemption rules
  - CPU receives a hard interrupt
  - Process requests IO
  - Process voluntarily surrenders the CPU via sched_yield
  - Scheduler algorithm determines that process should be preempted
- Viewing scheduler policy and priority

  chrt -p pid

  ps axo pid,comm,rtprio,policy

  top
- The init process starts with SCHED_OHTER
  - Each process inherits parent's scheduling policy and priority at creatation time

Scheduling Policy

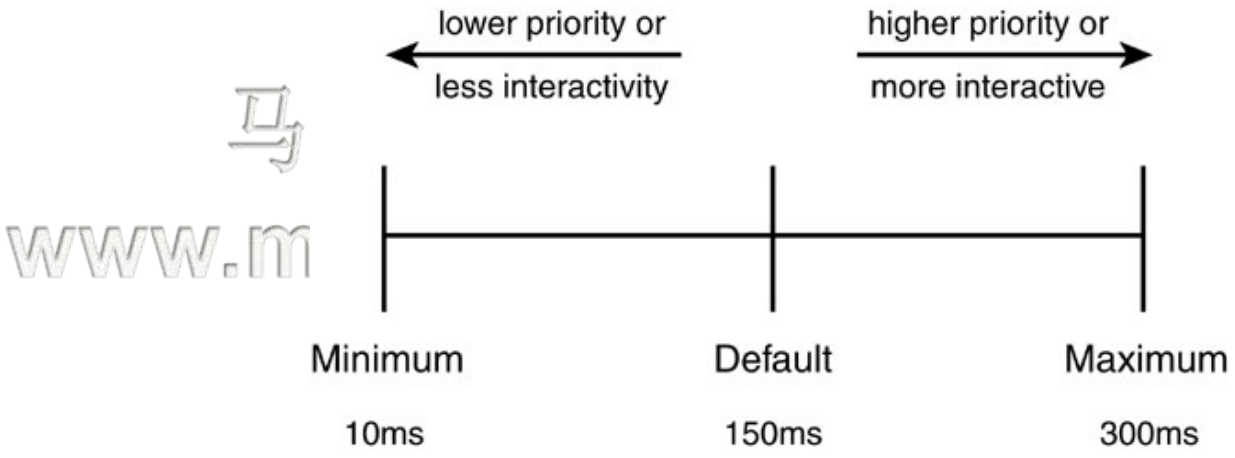
- Policy is the behavior of the scheduler that determines what runs when
  - A scheduler's policy often determines the overall feel of a system and is responsible for optimally utilizing processor time
- The scheduling algorithm of traditional Unix operating systems must fulfill several conflicting objectives:
  - fast process response time
  - good throughput for background jobs
  - avoidance of process starvation
  - reconciliation of the needs of low
  - high-priority processes
  - and so on
- The set of rules used to determine when and how to select a new process to run is called **scheduling policy**

Characterizing processes

- When speaking about scheduling, processes are traditionally classified as I/O-bound or CPU-bound
  - The former is characterized as a process that spends much of its time submitting and waiting on I/O requests
  - Conversely, processor-bound processes spend much of their time executing code
    - A scheduler policy for processor-bound processes tends to run such processes less frequently but for longer durations
- Linux, aiming to provide good interactive response and desktop performance, optimizes for process response (low latency), thus favoring I/O-bound processes over processor-bound processors

Process Priority

- In Linux, process priority is dynamic
  - The scheduler keeps track of what processes are doing and adjusts their priorities periodically
    - Processes that have been denied the use of a CPU for a long time interval are boosted by dynamically increasing their priority
    - Processes running for a long time are penalized by decreasing their priority



Static priority

- Every conventional process has its own **static priority**, which is a value used by the scheduler to rate the process with respect to the other conventional processes in the system
- The kernel represents the static priority of a conventional process with a number ranging from 100 (highest priority) to 139 (lowest priority)
- New process always inherits the static priority of its parent
  - However, a user can change the static priority of the processes that he owns by passing some "nice values"

real-time priority

- 0-99
  - higher real-time priority values correspond to a greater priority
- All real-time processes are at a higher priority than normal processes; that is, the real-time priority and nice value are in disjoint value spaces
  - ps -eo state,uid,pid,ppid,rtprio,time,comm

Timeslice

- The timeslice is the numeric value that represents how long a task can run until it is preempted
  - The scheduler policy must dictate a default timeslice, which is not a trivial exercise

The Scheduling Algorithm

- Scheduler Classes
  - The Linux scheduler is modular, enabling different algorithms to schedule different types of processes
    - This modularity is called *scheduler classes*
    - Scheduler classes enable different, pluggable algorithms to coexist, scheduling their own types of processes
    - Each scheduler class has a priority. The highest priority scheduler class that has a runnable process wins, selecting who runs next

Scheduling classes

- Every Linux process is always scheduled according to one of the following scheduling classes :
  - SCHED_FIFO [1-99]
    - A First-In, First-Out real-time process
  - SCHED_RR
    - A Round Robin real-time process
  - SCHED_NORMAL (100-139)
    - A conventional, time-shared process
    - Also named by SCHED_OTHER
    - For normal processes

SCHED_OTHER

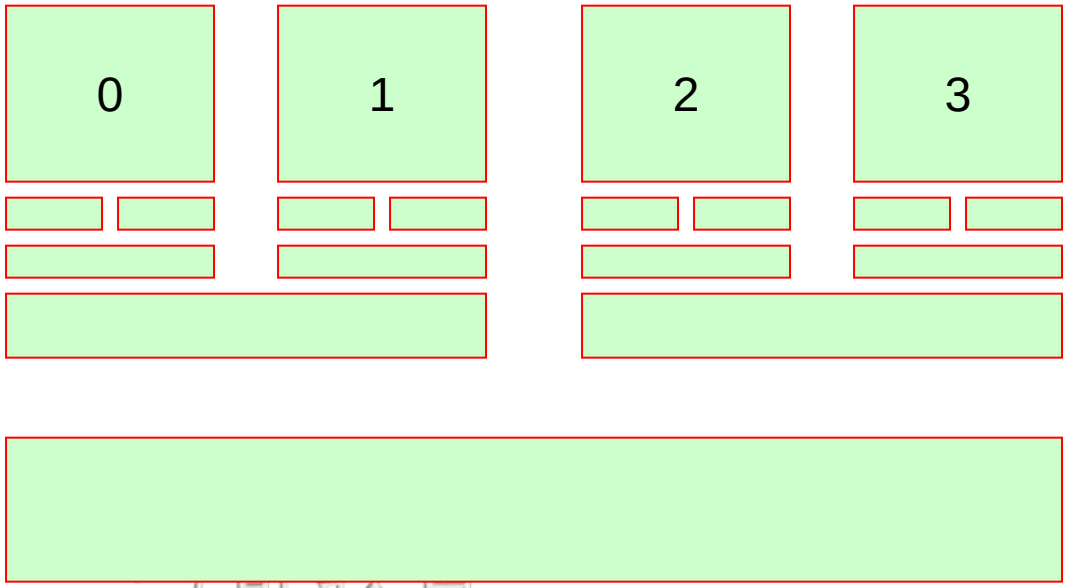
- Priority may vary
  - Processes with equal priority can preempt current process every 20ms to prevent CPU starvation
  - CPU bound processes receive a +5 priority penalty after preemption
- Interactive tasks spend time waiting for IO
  - Scheduler tracks time spent waiting for IO for each process and calculates a sleep average
  - High sleep average indicates interactive process
    - Interactive processes may be re-inserted into the active queue
    - If not, receive a -5 priority boost and move to expired queue

Tuning scheduler policy

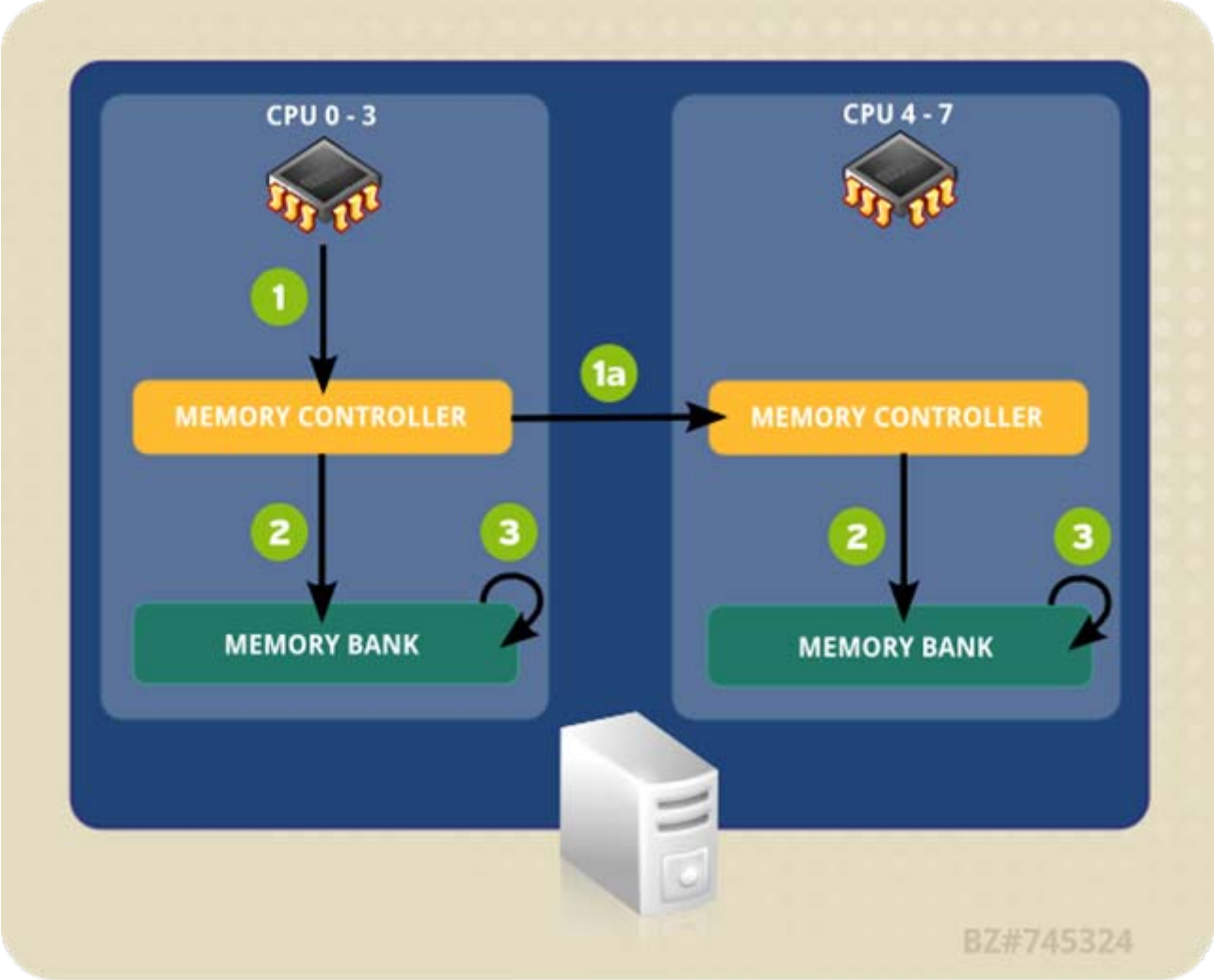
- SCHED_FIFO
  - chrt -f [1-99] /path/to/program arguments
- SCHED_RR
  - chrt -r [1-99] /path/to/program arguments
- SCHED_OTHER (SCHED_NORMAL)
  - nice
  - renice

- 调度类别：
  - RT
    - SCHED_FIFO
    - SCHED_RR
  - 100-139
    - SCHED_Other
  - SCHED_BATCH
  - SCHED_IDLE
- 抢占
  - tick：时钟中断
    - 100Hz
    - 1000Hz
- RHEL 6.4
  - tick less
  - interrupt-driven
    - 硬中断
    - 软中断
    - 深度睡眠

- I1，D1



Local and Remote Memory Access in NUMA Topolog



Viewing CPU performance data

- Load average: average length of run queues
  - Considers only tasks in TASK_RUNNABLE and TASK_UNINTERRUPTABLE

    sar -q
    htop
    w
    uptime
    vmstat 1 5
- CPU utilization

    mpstat 1 2
    sar -P ALL 1 2
    iostat -c 1 2
    /proc/stat
    dstat -c

Equalizing CPU visit count

- Process moves to the expired queue when preempted
  - Imposes a built-in affinity for the CPU
  - Can lead to unbalanced run queues
- Scheduler rebalances run queues
  - Every 100ms if all processors are busy
  - Every 1ms if a CPU is idle
  - View a specific process

    watch -n.5 'ps axo comm,pid,psr| grep *program_name*'
- Consequences
  - Lower visit count leads to higher throughput
  - Moving a task to another CPU guarantees a cache miss

Tuning process affinity with taskset

- Use the taskset command to pin a task to a CPU

  taskset [opts] [mask|list] [pid|command [arg]...]

  # taskset -c -p *cpulist* PID
  - mask
    - 0x00000001 CPU #0
    - 0x00000002 CPU #1
    - 0x00000003 CPU #0 and CPU #1
- Consequences
  - Improve cache hits(lower service time) for applications with unit cache stride
  - Unbalanced run queues can cause log wait times
  - For NUMA, avoid non-local memory accesses

Tuning run queue length with taskset

- Restrict length of a CPU run queue
  - Isolate a CPU from automatic scheduling in /etc/grub.conf

    isolcpus=*cpu number,…,cpu number*
  - Pin tasks to that CPU with taskset
  - Consider moving IRQs off the CPU

Scheduler domains

- Group processors into cpusets
  - Each cpuset represents a scheduler domain
  - Supports both multi-core and NUMA architectures
  - Simple management interface through the cpuset virtual file system
- The root cpuset contains all system resources
- Child cpusets
  - Each cpuset must contain at least one CPU and one memory zone
  - Child cpusets can be nested
  - Dynamically attach tasks to a cpuset
- Consequences
  - Control latency due to queue length, cache, and NUMA zones
  - Assign processes with different CPU characteristics to different cpusets
  - Scalable for complex performance scenarios

Configuring the root cpuset

- Create a mount point at /cpusets
- Add an entry to /etc/fstab

  cpuset /cpusets cpuset defaults 0 0
- Mount the filesystem to automatically create the cpuset

  /cpusets/cpus

  /cpusets/mems

  /cpusets/tasks
  - All CPUs and memory zones belong to the root cpuset
  - All existing PIDs are assigned to the root cpuset

Configuring a child cpuset

- Create a subdirectory of the root cpuset

  mkdir /cpusets/magedu

- Assign resources as a range or comma-separated list

  echo 0 > /cpusets/magedu/cpus

  echo 0 > /cpusets/magedu/mems

- Attach one task at a time

  echo `pidof *process*` > /cpusets/magedu/tasks

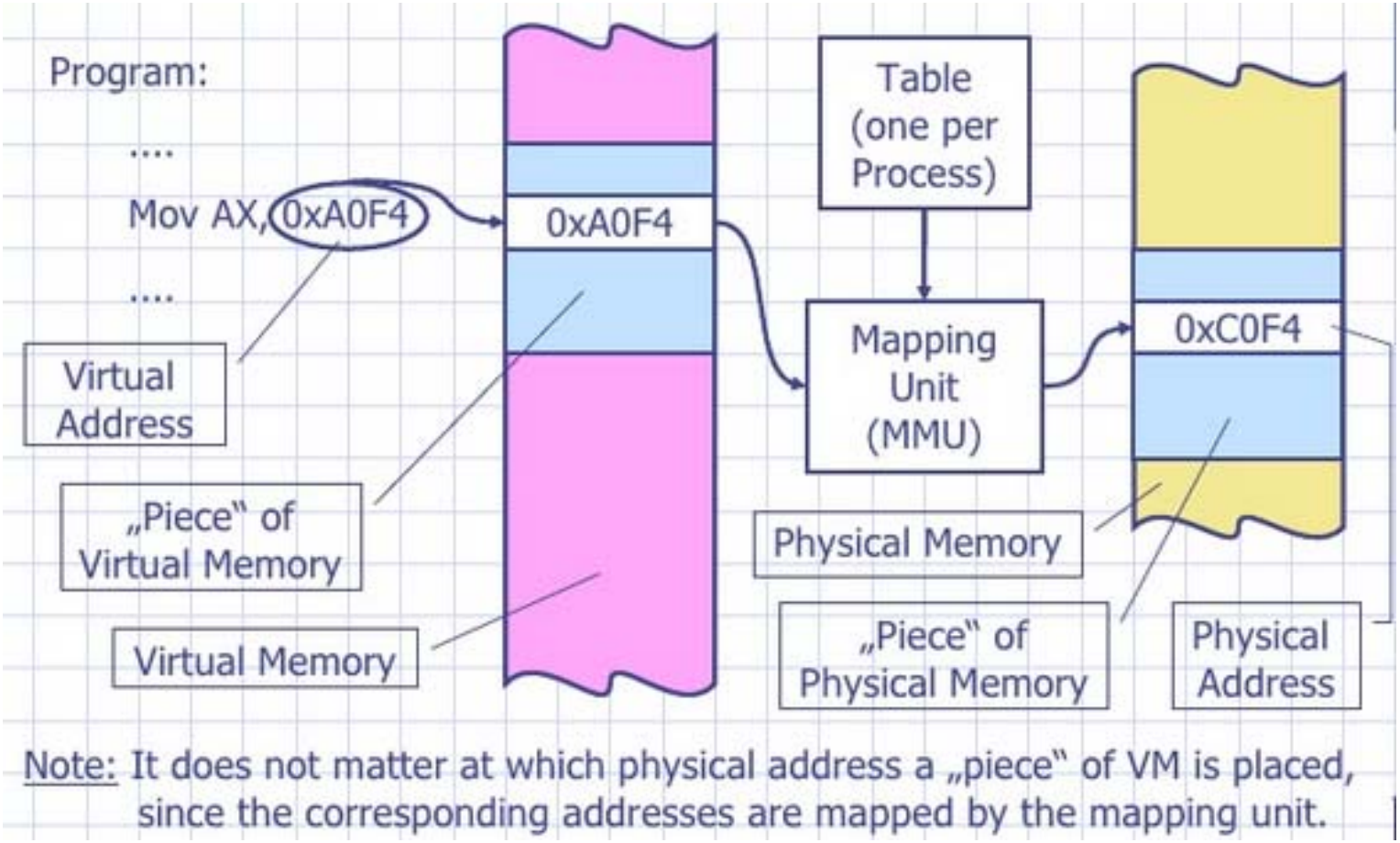
- Persist in /etc/rc.local

马哥教育

Linux memory architecture

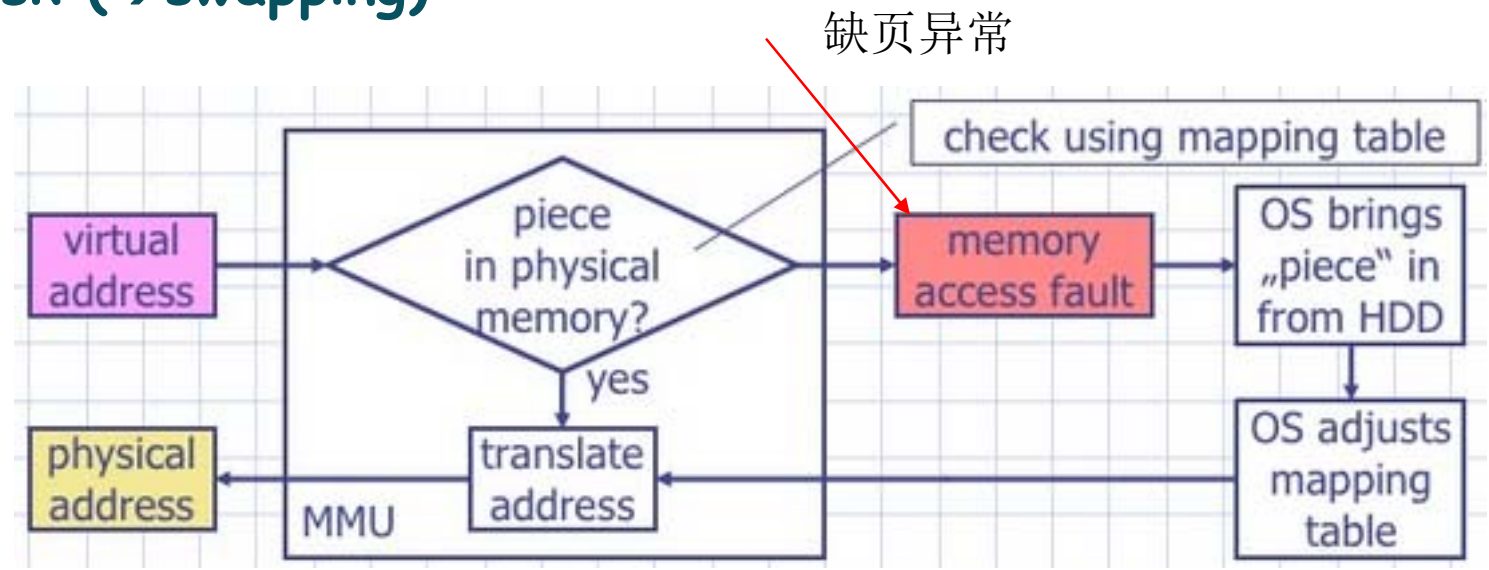
主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

What is VM?

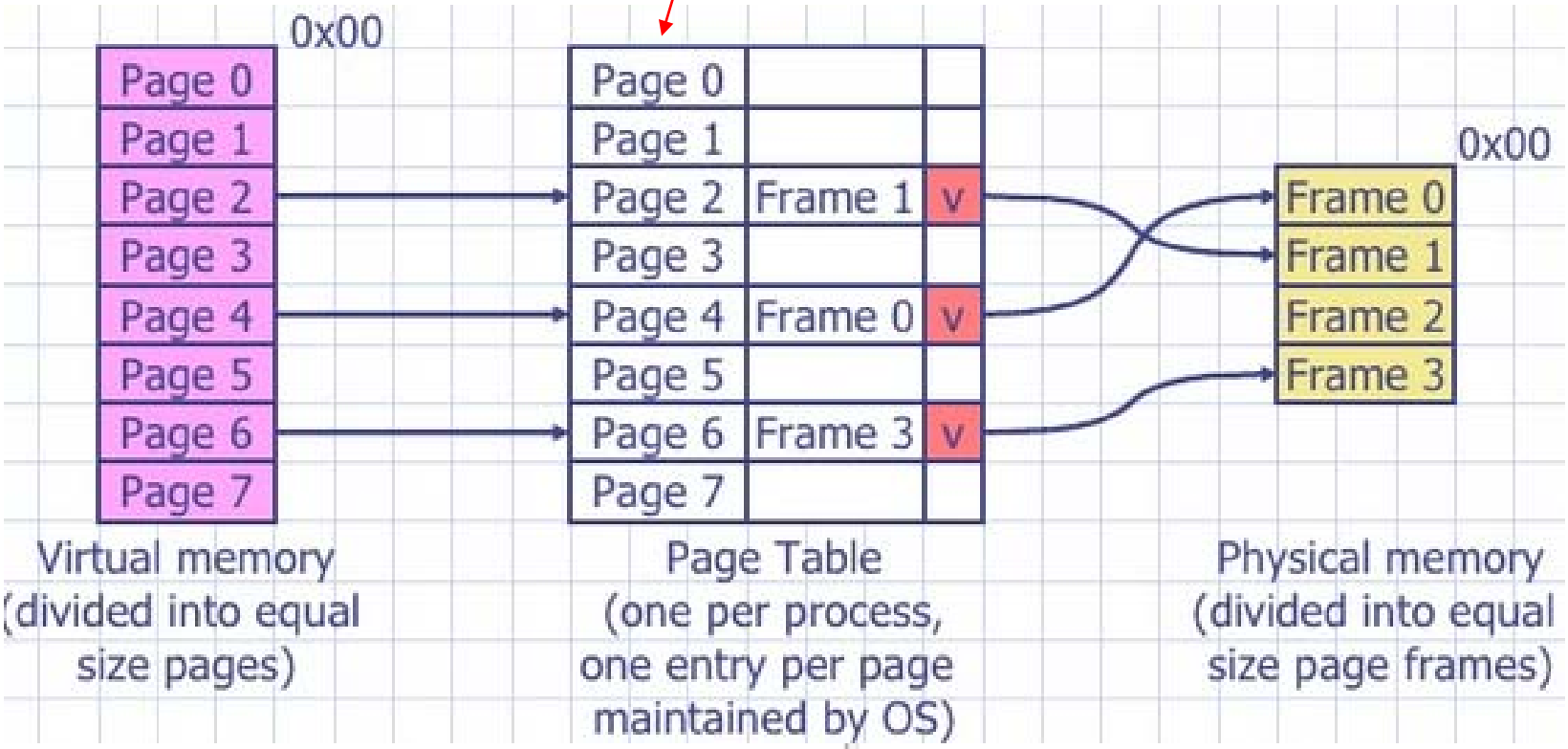

Program:
....
Mov AX, 0xA0F4
....
Virtual Address
„Piece" of Virtual Memory
Virtual Memory
0xA0F4
Table (one per Process)
Mapping Unit (MMU)
0xC0F4
Physical Memory
„Piece" of Physical Memory
Physical Address
Note: It does not matter at which physical address a „piece" of VM is placed, since the corresponding addresses are mapped by the mapping unit.

The mapping process

- Usually every process hash its own mapping table
  - Own virtual address space (assumed from now on)
- Not every "piece" of VM has to be present in PM
  - "Pieces" may be loaded from HDD as they are referenced
  - Rarely used "pieces" may be discarded or written out to disk (→swapping)



What is paging?



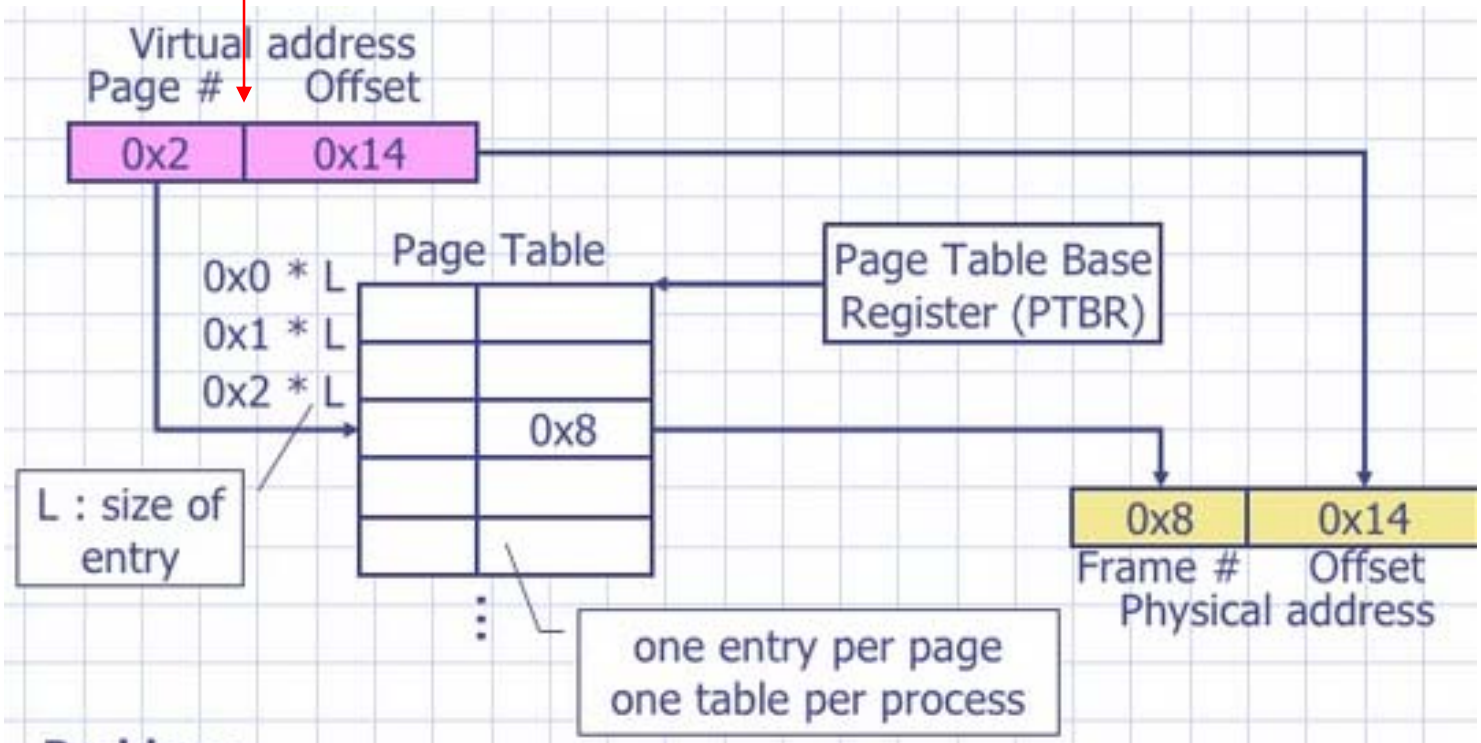
Paging Implementation

Typical Page Table Entry

r	w	x	v	re	m	s	c	su	pid	g	gd	other	Page Frame #

r	read	s	shared
w	write	c	caching disabled
x	execute	su	super-page
v	valid	pid	process id
re	referenced	g	(extended) guard
m	modified	gd	guard data

Single level page tables

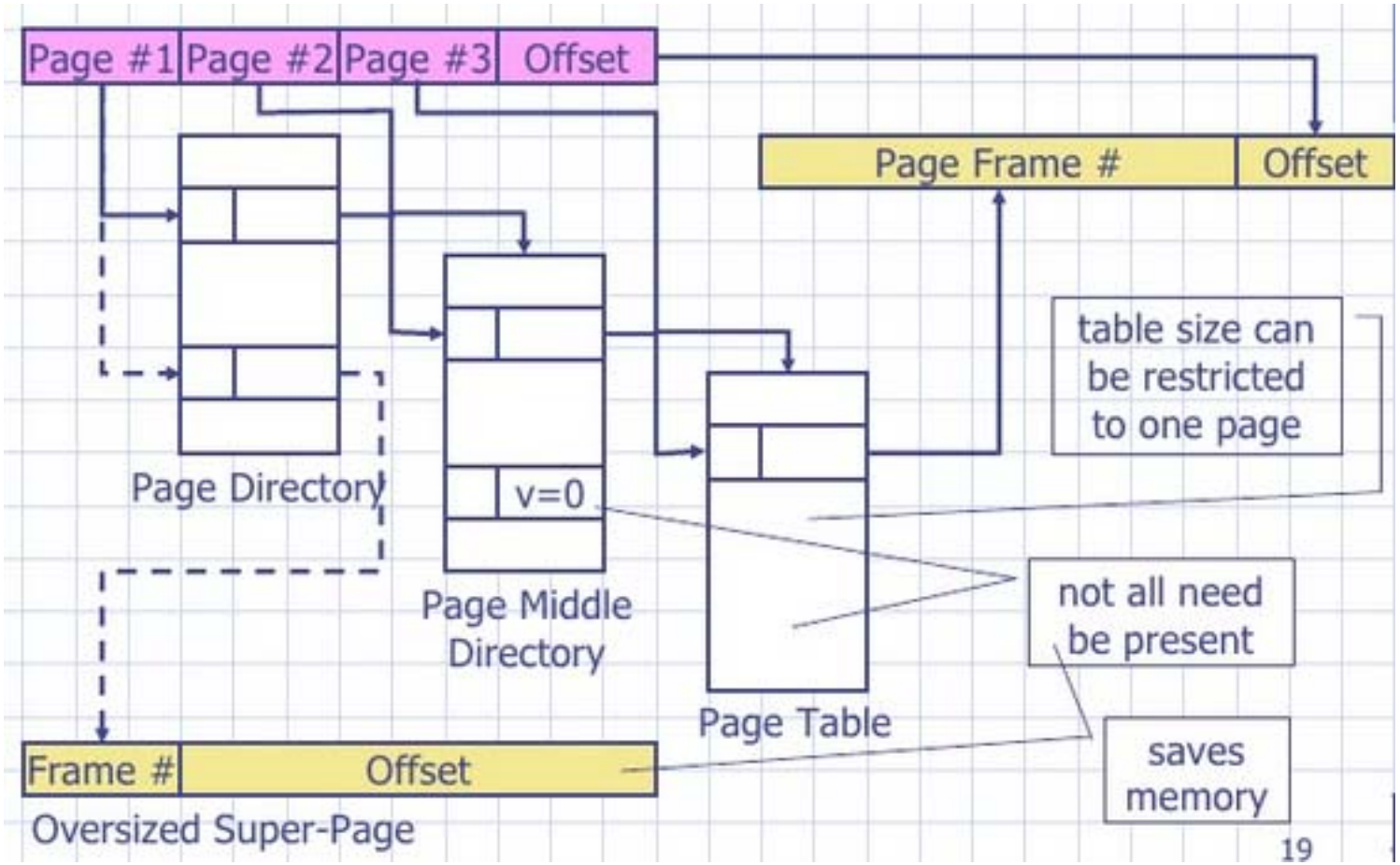


Problem:

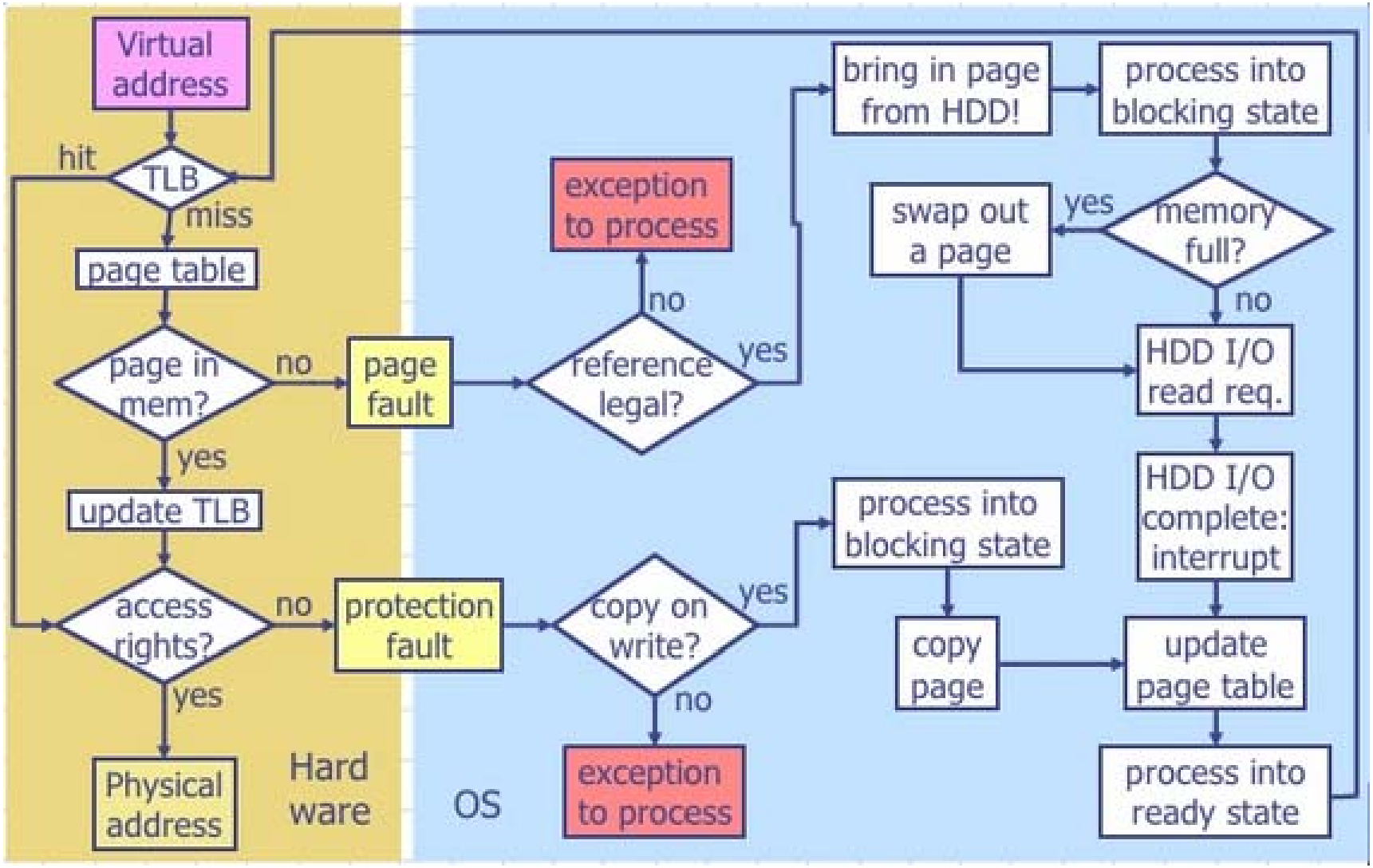
Page tables can get **very large**, e.g. 32 bit address space, 4KB pages
→ 2^20 entries per process → 4MB at 4B per entry
64 bit → 16777216 GB page table!!!!

18

Multi Level Page Tables

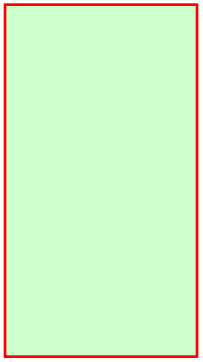


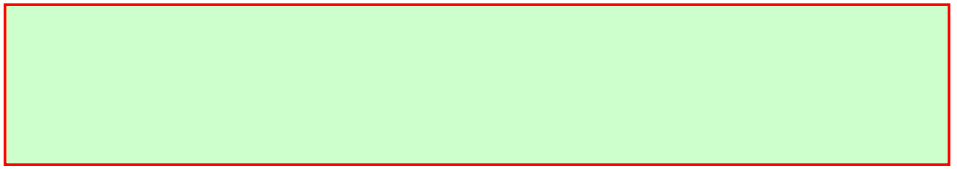
Conversation of Virtual Address



OOM: Out Of Memory

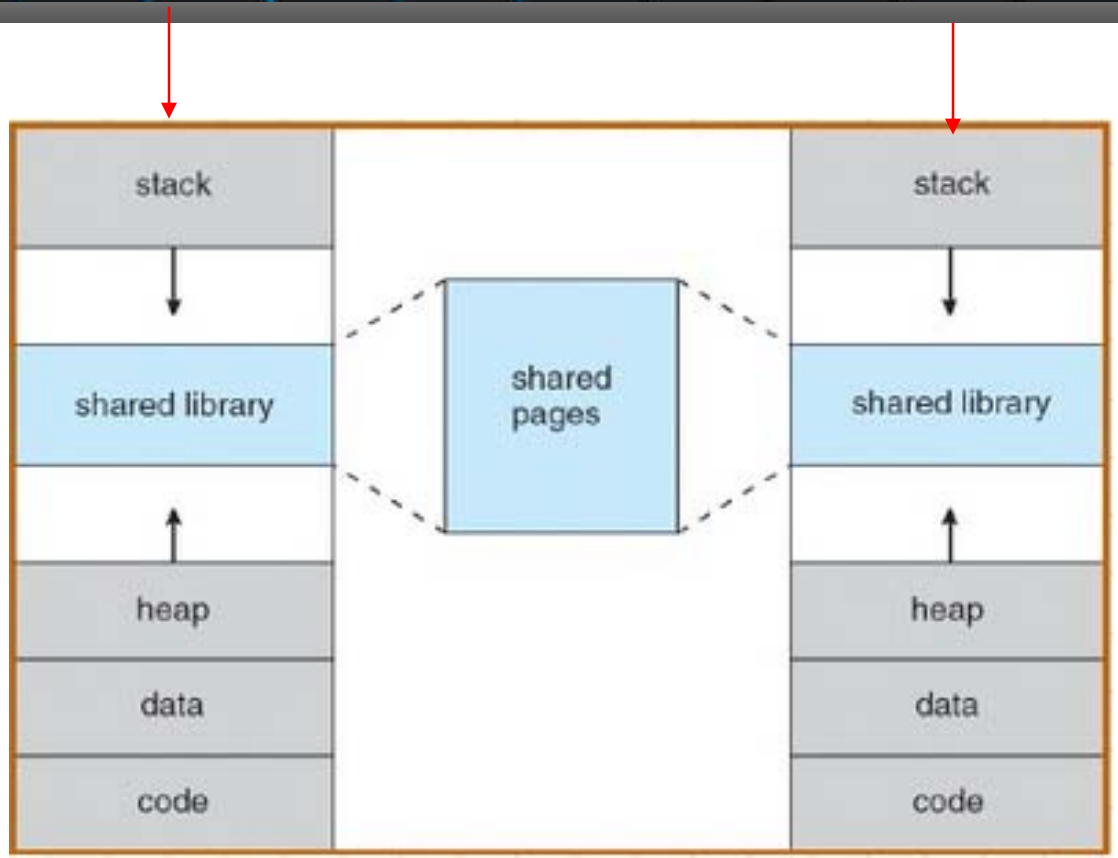
- oom kill



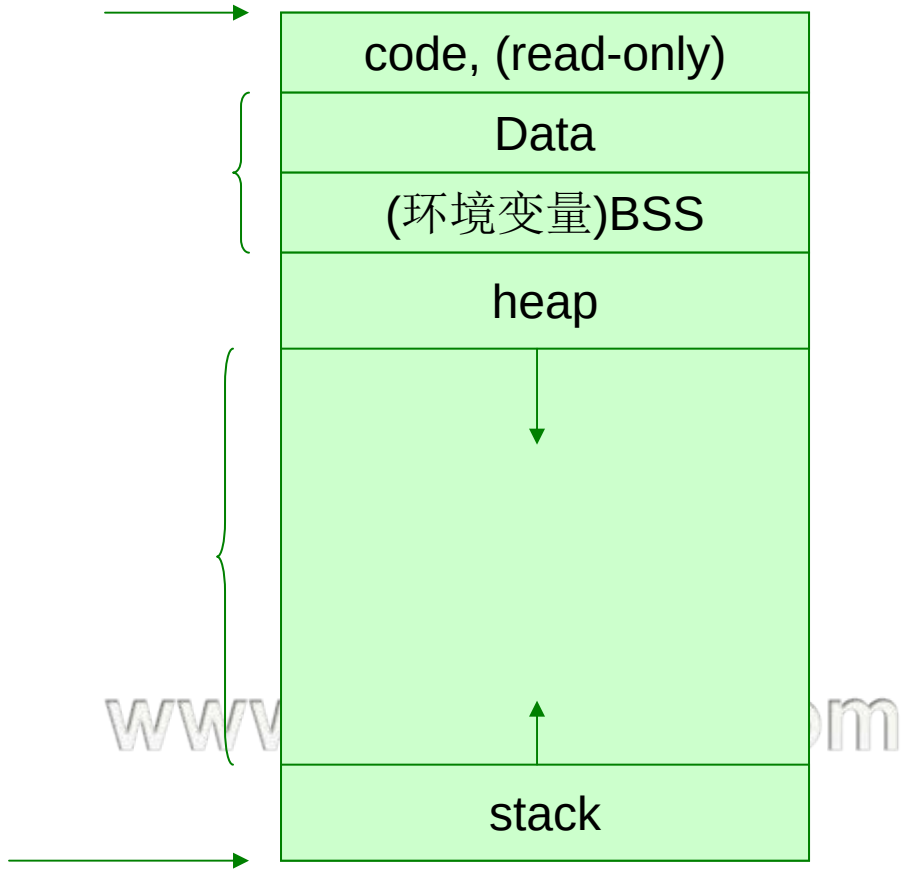


- 2G, 4G
- swap + memory*overcommit_ratio

Shared library is VA



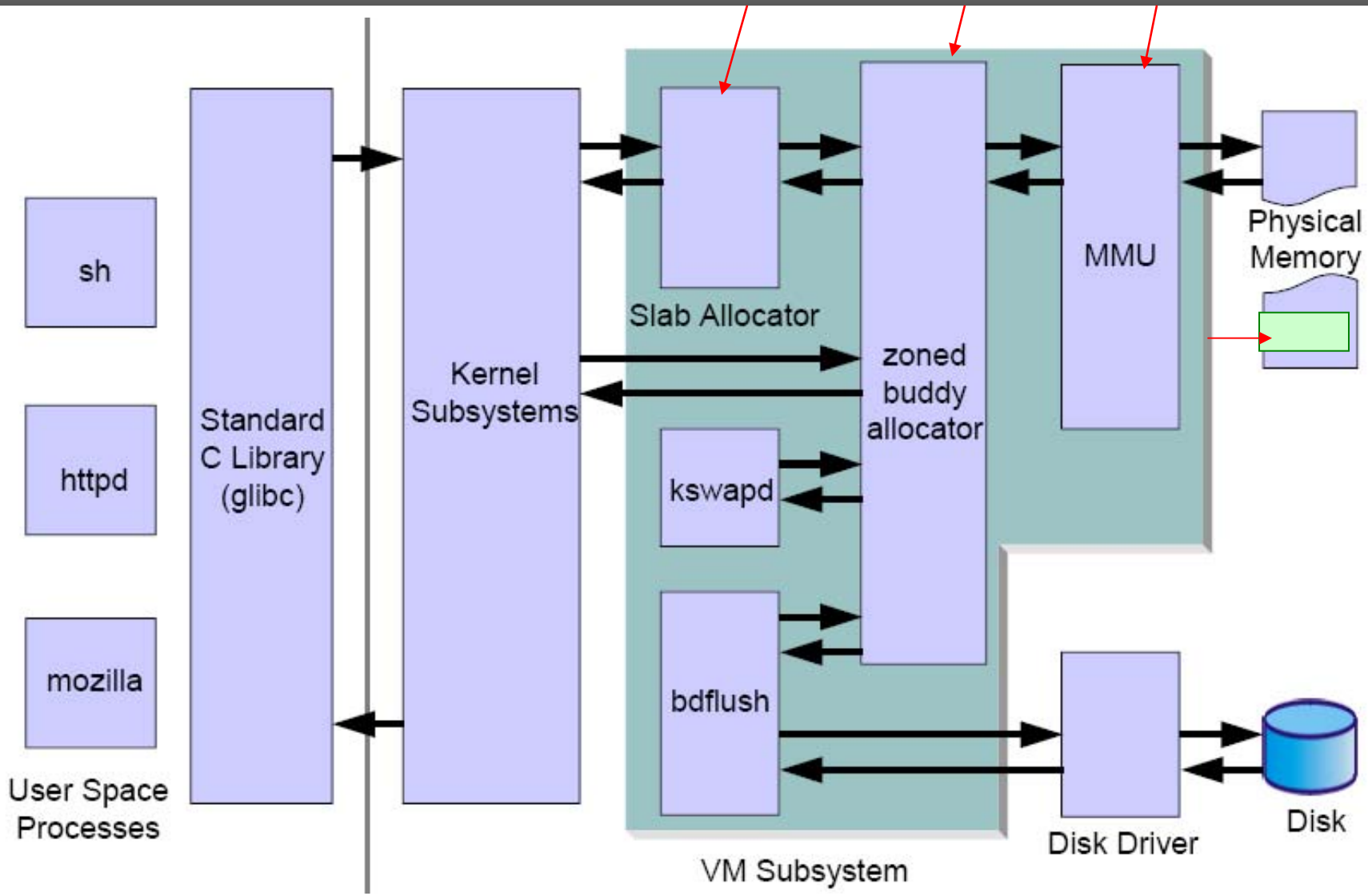
task_struct



Memory Addresses

- Logical address (segment)
  - Included in the machine language instructions to specify the address of an operand or of an instruction
  - This type of address embodies the well-known 80 x 86 segmented architecture
- Linear address (also known as virtual address)
  - A single 32-bit unsigned integer that can be used to address up to 4 GB that is, up to 4,294,967,296 memory cells
  - Linear addresses are usually represented in hexadecimal notation; their values range from 0x00000000 to 0xffffffff
- Physical address
  - Used to address memory cells in memory chips

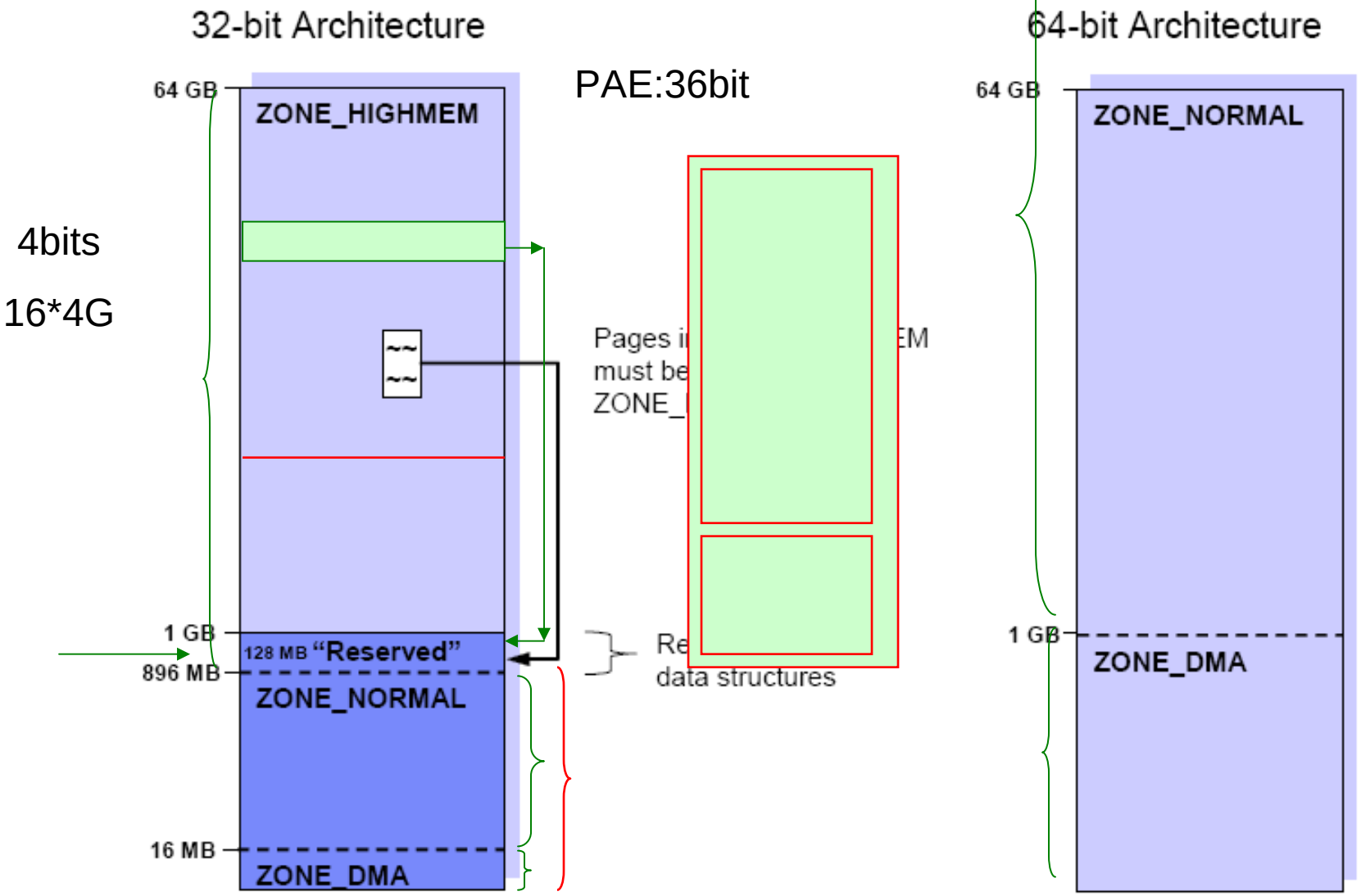
Virtual memory manager



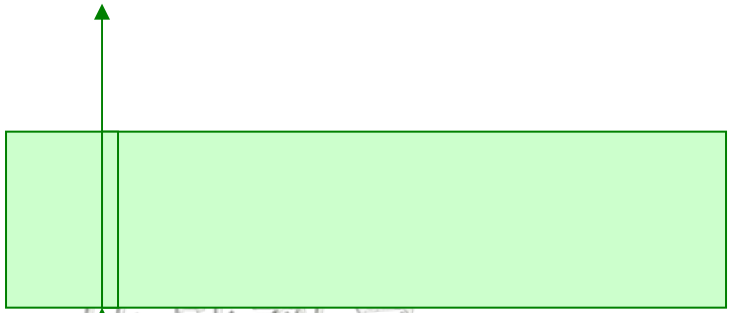
Paging in Hardware

- For the sake of efficiency, linear addresses are grouped in fixed-length intervals called **pages**
- The paging unit translates linear addresses into physical ones
- The paging unit thinks of all RAM as partitioned into fixed-length **page frames**
  - Each page frame contains a page that is, the length of a page frame coincides with that of a page
  - A page frame is a constituent of main memory, and hence it is a storage area
- The data structures that map linear to physical addresses are called **page tables**
  - They are stored in main memory and must be properly initialized by the kernel before enabling the paging unit

The Linux Memory Architecture



- SRAM
  - Static RAM
- DRAM
- DDR
  - Dynamic RAM
  - SDRAM
  - DDR
  - Rambus DRAM



Zones

- Because of hardware limitations, the kernel cannot treat all pages as identical
  - Some pages, because of their physical address in memory, cannot be used for certain tasks
  - Because of this limitation, the kernel divides pages into different zones
  - The kernel uses the zones to group pages of similar properties

- ZONE_DMA
  - This zone contains pages that can undergo DMA
- ZONE_DMA32
  - Like ZOME_DMA, this zone contains pages that can undergo DMA
  - Unlike ZONE_DMA, these pages are accessible only by 32-bit devices
  - On some architectures, this zone is a larger subset of memory
- ZONE_NORMAL
  - This zone contains normal, regularly mapped, pages
- ZONE_HIGHMEM
  - This zone contains "high memory," which are pages not permanently mapped into the kernel's address space

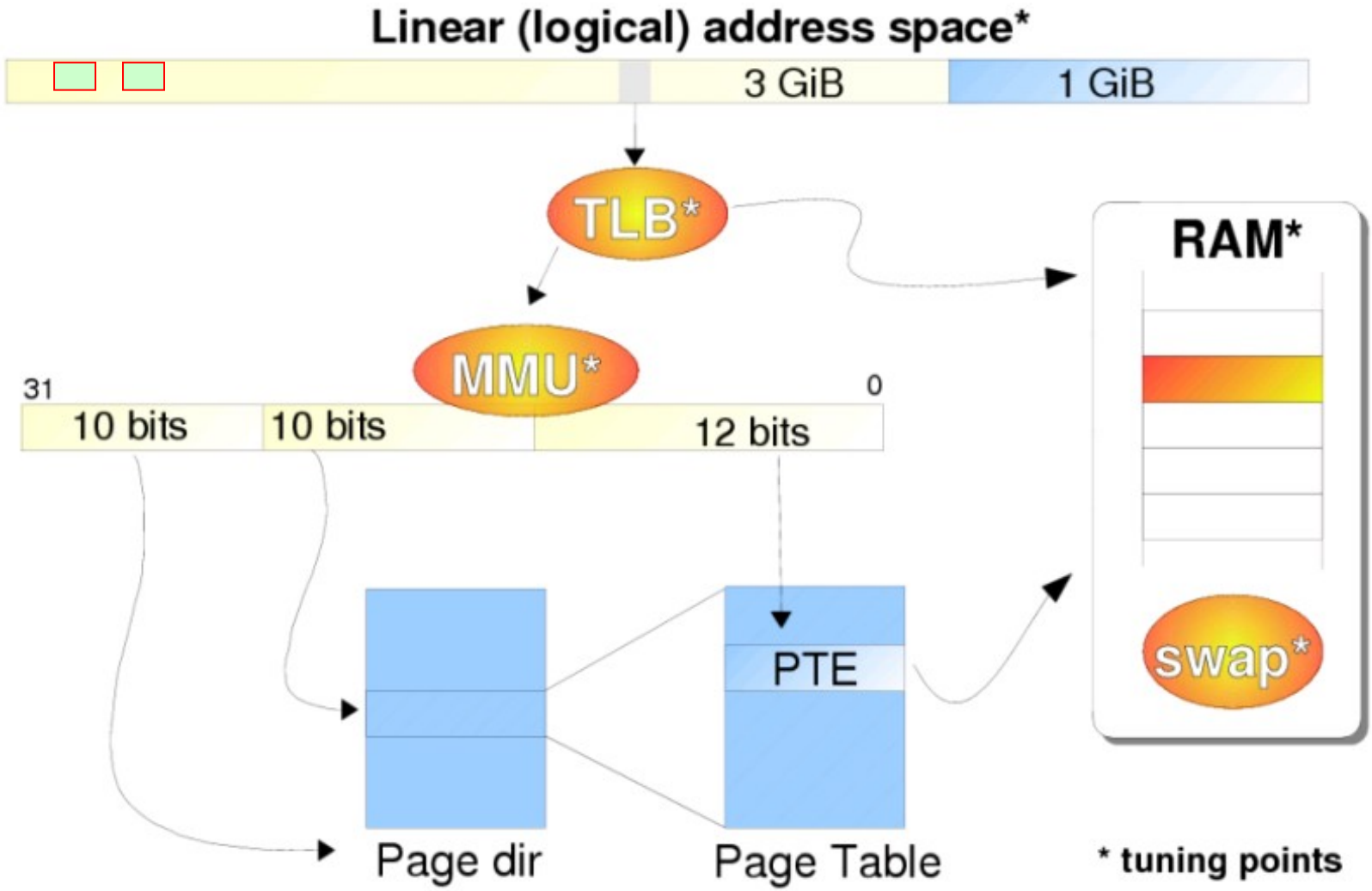
Virtual memory addressing layout

- On 32-bit architectures, the maximum address space that single process can access is 4GB
  - The Linux kernel can directly address only the first gigabyte of physical memory (896 MB when considering the reserved range)
- In a standard implementation, the virtual address space is divided into a 3 GB user space and a 1 GB kernel space

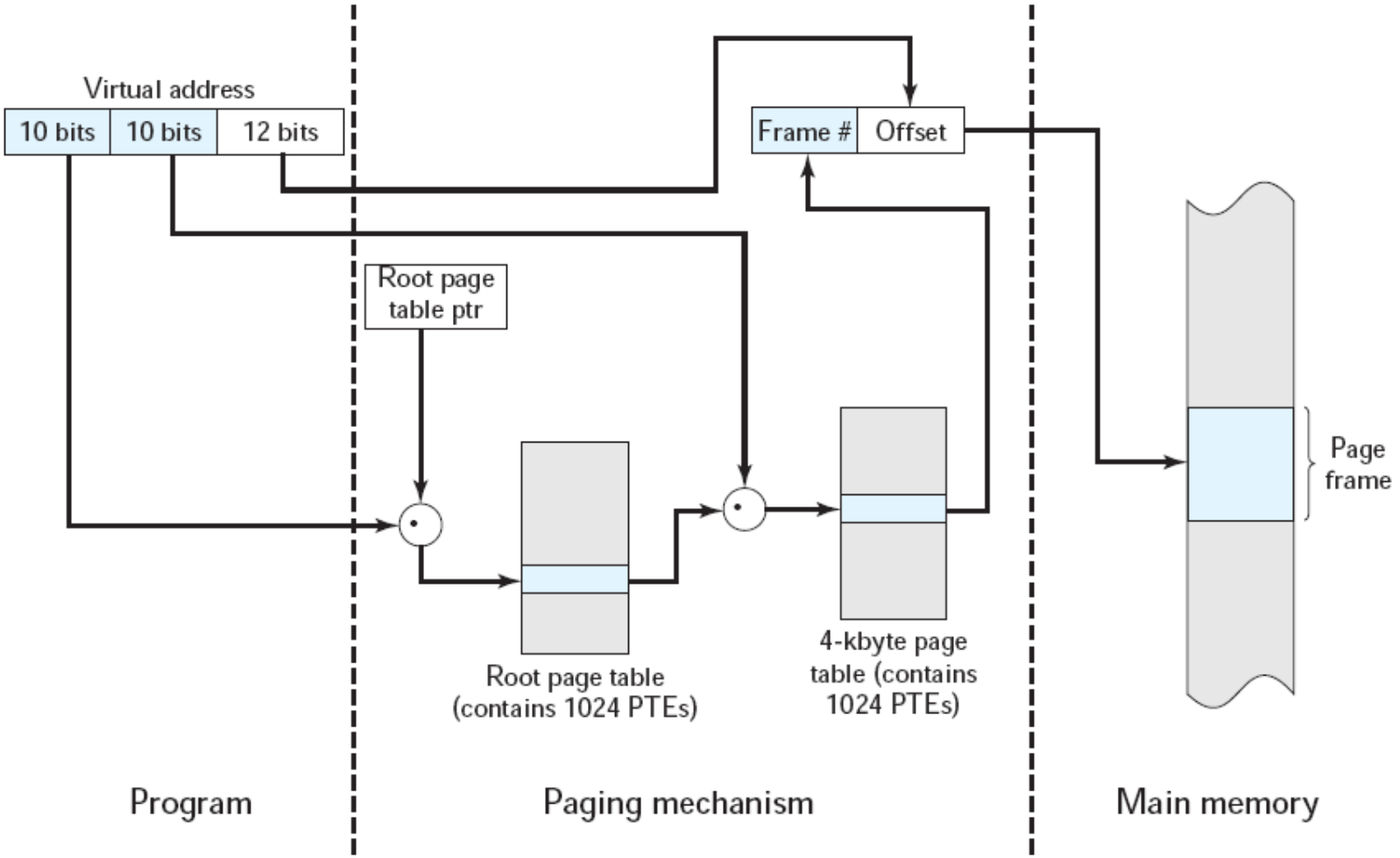
Physical address space

- Kernel uses page table entries to translate virtual addresses tinto physical pages
  - PTEs are cached in translation lookaside buffers(TLB)
  - Hardware TLB determines data and instruction page sizes
    - x86info -c
    - dmesg
- Kernel memory is non-pageable(non-swappable)
- Keeping track of in-use pages
  - A page frame refers to a page of RAM
  - The kernel uses page tables to keep track of page frames
  - Each process has its own page table structures
  - The page table maps virtual addresses to physical addresses

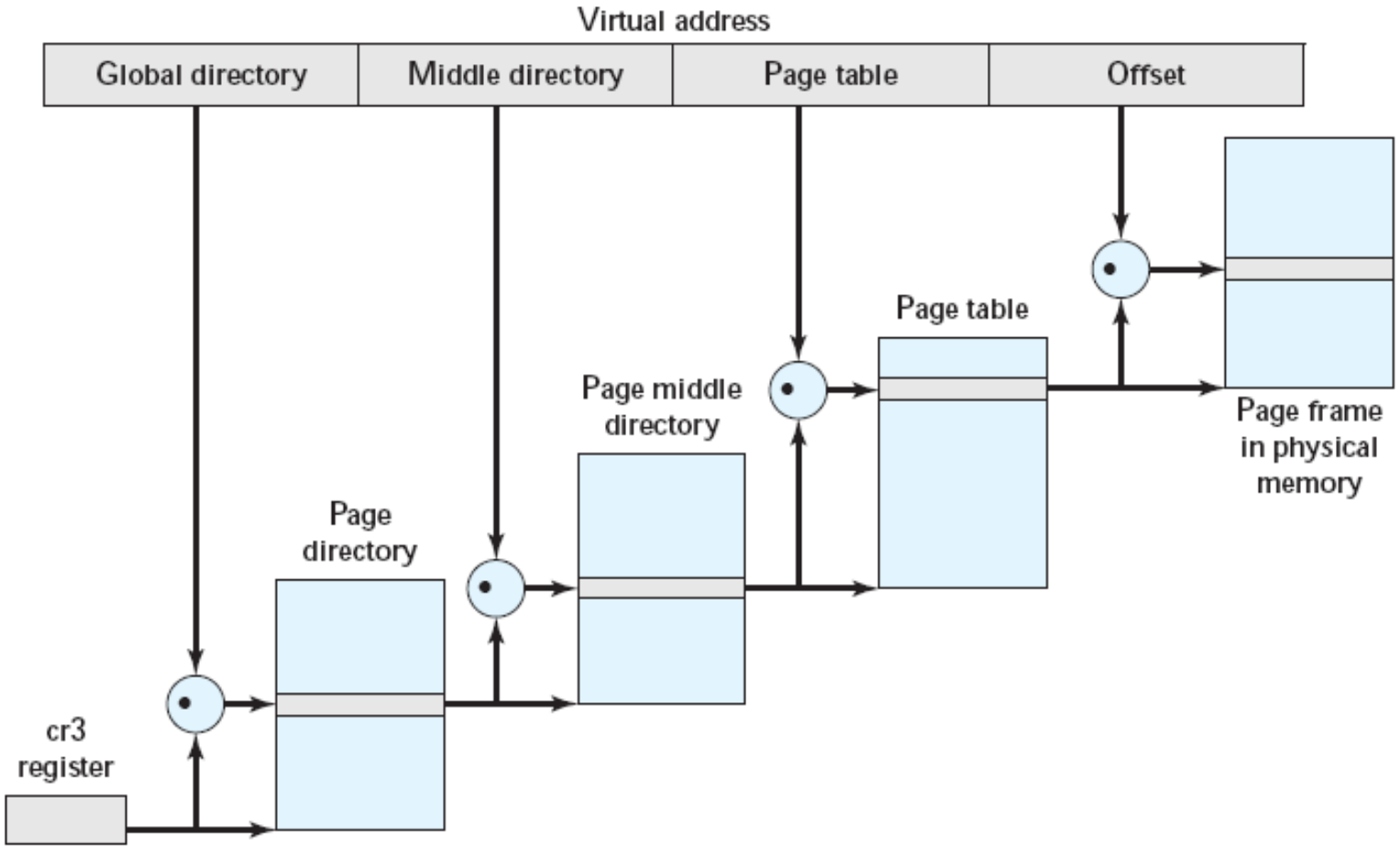
Mapping virtual addresses(x86)



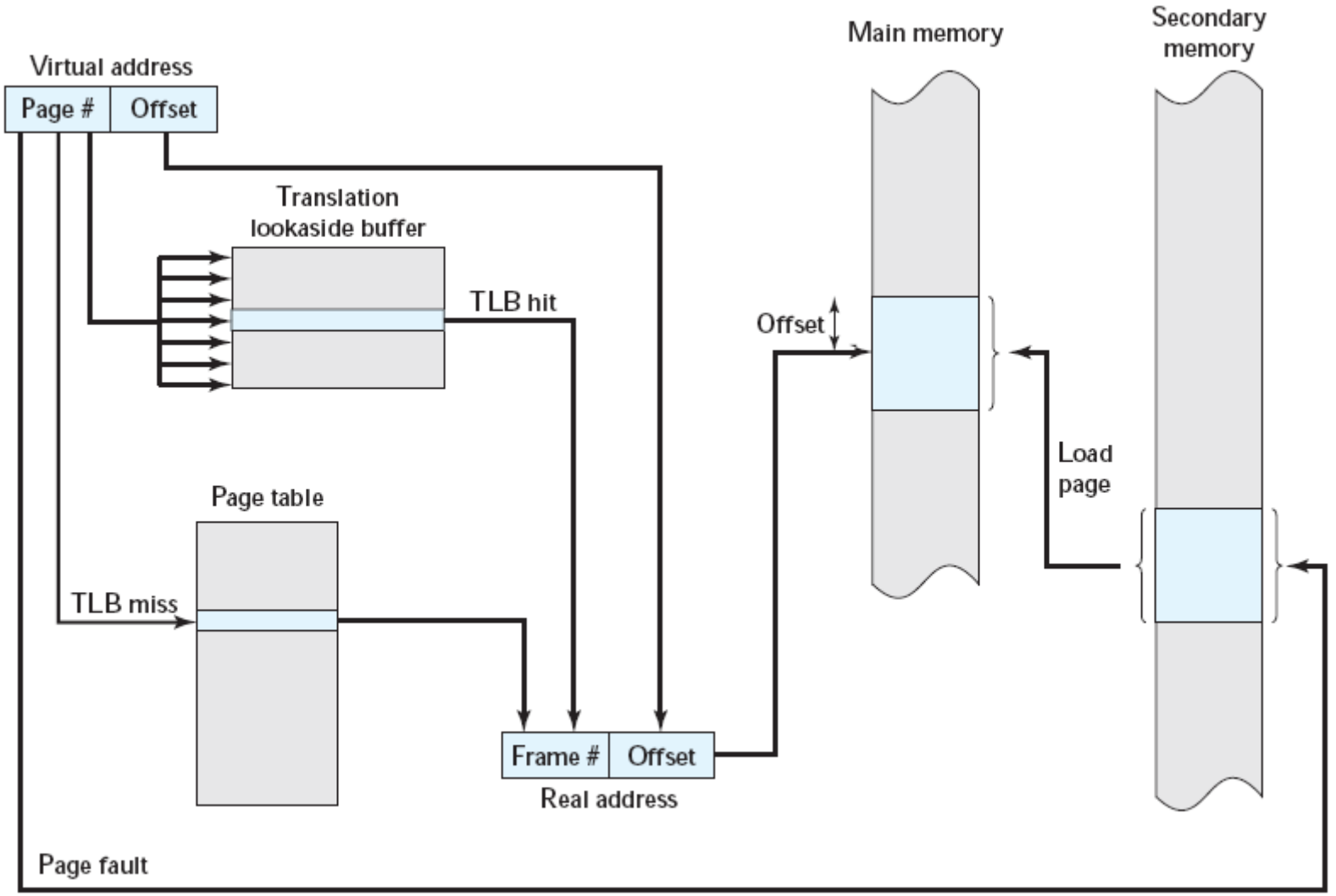
Address Translation



Address Translation in Linux Virtual Memory Scheme



Use of a Translation Lookaside Buffer



Viewing process address space

- Proc files
  - /proc/PID/status
  - /proc/PID/statm
- Commands
  - gnome-system-monitor
  - pmap
  - memusage
    - yum install glibc-utils
  - dstat -m
  - top

/proc/PID/statm

- Provides information about memory status in pages
- The columns are:
  - size total program size
  - resident resident set size
  - share shared pages
  - text text (code)
  - lib library
  - data data/stack
  - dt dirty pages (unused in Linux 2.6)

Overview of memory allocation

- Process forks or execs child process
  - Child process uses parent process's page frame until a write access is made
  - This is referred to as copy on write
- New process requests memory
  - kernel commits additional virtual address space to process
- New process uses memory
  - Triggers a page fault exception
    - Minor: Kernel allocates a new page frame with help form MMU
    - Major: Kernel blocks process while page is retrieved from disk
- Process frees memory
  - Kernel reclaims pages
  - ps axo minflt,majflt
  - ps -o minflt,majflt PID

Page frame allocation

- A page is usually 4 K bytes in size
- When a process requests a certain amount of pages, if there are available pages the Linux kernel can allocate them to the process immediately
- Otherwise pages have to be taken from some other process or page cache
- The kernel knows how many memory pages are available and where they are located

Improving RAM performance

- Static RAM (SRAM)
  - Keeps data without updating
  - more expensive
- Dynamic RAM(DRAM)
  - Must be refreshed
  - Synchronous DRAM (SDRAM) is synchronized to CPU clock
  - Double data rate DRAM (DDR) reads on both sides of clock signal
  - Rambus DRAM (RDRAM) uses a high-speed, narrow bus architecture

Hugetable page

- Hugetlbfs support is built on top of multiple page size support that is provided by most modern architectures
- Users can use the huge page support in Linux kernel by either using the mmap system call or standard Sysv shared memory system calls (shmget, shmat)

  cat /proc/meminfo | grep HugePage

Improving TLB performance

- Kernel must usually flush TLB entries upon a context switch
- Use free, contiguous physical pages
  - Automatically via the buddy allocator
    /proc/buddyinfo
  - Manually via hugepages (not pageable)
    - Linux supports large sized pages through the hugepages mechanism
    - Sometimes known as bigpages, largepages or the hugetlbfs filesystem
- Consequences
  - TLB cache hit more likely
  - Reduces PTE visit count

Tuning TLB performance

- Check size of hugepages
  - x86info -a | grep "Data TLB"
  - dmesg
  - cat /proc/meminfo
- Enable hugepages
  - In /etc/sysctl.conf
    vm.nr_hugepages = *n*
  - Kernel parameter
    hugepages=*n*
- Configure hugetlbfs if needed by application
  - mmap system call requires that hugetlbfs is mounted

```
mkdir  /hugepages
mount -t  hugetlbfs  none  /hugepages
```

  - shmat and shmget system calls do not require hugetlbfs

Viewing system calls

- Trace every system call made by a program

  strace -o /tmp/strace.out -p PID

  grep mmap /tmp/strace.out

- Summarize system calls

  strace -c -p PID or

  strace -c COMMAND

- Other uses
  - Investigate lock contentions
  - Identify problems caused by improper file permissions
  - Pinpoint IO problems

Strategies for using memory

- Reduce overhead for tiny memory objects
  - Slab cache
- Reduce or defer service time for slower subsystems
  - Filesystem metadata: buffer cache (slab cache)
  - Disk IO: page cache
  - Interprocess communications: shared memory
  - Network IO: buffer cache, arp cache, connection tracking
- Considerations when tuning memory
  - How should pages be reclaimed to avoid pressure?
  - Larger writes are usually more efficient due to re-sorting

A closer look at demand paging

- Page frames are not allocated until the process has something to store
  - Resident pages are located in RAM
  - Non-resident pages are on disk or have not been used
- Allows the kernel to overcommit memory system-wide
  - Advantageous for scientific applications
  - Usually works well due to locality of reference for executable images and memory
  - Memory (RAM+swap) may not be available when the process hash something to store

Tuning page allocation

- Set using

  vm.min_free_kbytes

  - Tuning vm.min_free_kbytes only be necessary when an application regularly needs to allocate a large block of memory, then frees that same memory
  - It may well be the case that the system has too little disk bandwidth, too little CPU power, or too little memory to handle its load

- Consequences
  - Reduces service time for demand paging
  - Memory is not available for other useage
  - Can cause pressure on ZONE_NORMAL

Tuning overcommit

- Set using

  vm.overcommit_memory

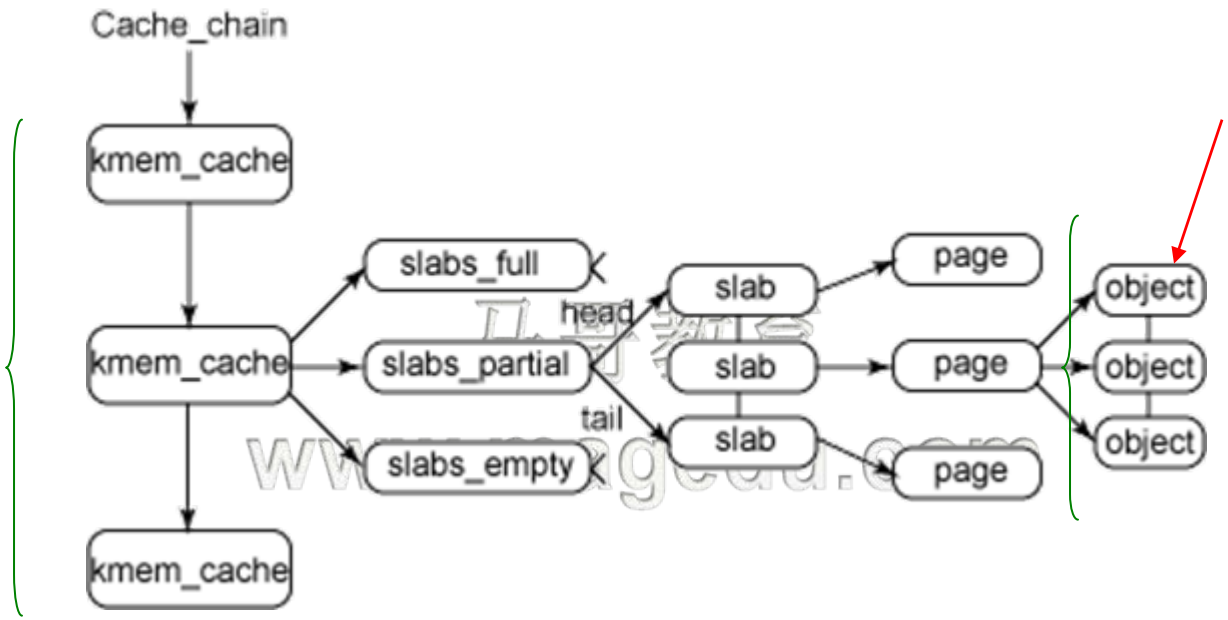
  - 0 = heuristic overcommit
  - 1 = always overcommit
  - 2 = commit all RAM plus a percentage of swap (may be > 100)

    vm.overcommit_ratio

    - Specified the percentage of physical memory allowed to be overcommited when the vm.overcommit_memory is set to 2

- View Committed_AS in /proc/meminfo
  - An estimate of how much RAM is required to avoid an out of memory (OOM) condition for the current workload on a system

- The slab memory cache contains pre-allocated memory pools that the kernel pulls memory from when it needs space to store various types of data structures



Slab cache

- Tiny kernel objects are stored in the slab
  - Extra overhead of tracking is better than using 1 page/object
  - Example: filesystem metadata (dentry and inode caches)
- Monitoring
  - /proc/slabinfo
  - slabtop
  - vmstat -m
- Tuning a particular slab cache
  - echo "cache_name limit batchcount shared" > /proc/slabinfo
    - limit the maximum number of objects that will be cached for each CPU
    - batchcount the maximum number of global cache objects that will be transferred to the per-CPU cache when it becomes empty
    - shared the sharing behavior for Symmetric MultiProcessing (SMP) systems

ARP cache

- ARP entries map hardware addresses to protocol addresses
  - Cached in /proc/net/arp
    - By default, the cache is limited to 512 entries as a soft limit and 1024 entries as a hard limit
  - Garbage collection removes stale or older entries
- Insufficient ARP cache leads to
  - Intermittent timeouts between hosts
  - ARP thrashing
- Too much ARP cache puts pressure on ZONE_NORMAL
- List entries

  ip neighbor list
- Flush cache

  ip neighbor flush dev ethX

Tuning ARP cache

- Adjust where the gc will leave arp table alone
  net.ipv4.neigh.default.gc_thresh1
  - default 128
- Soft upper limit
  net.ipv4.neigh.default.gc_thresh2
  - defalut 512
  - Becomes hard limit after 5 seconds
- Hard upper limit
  net.ipv4.neigh.default.gc_thresh3
- Garbage collection frequency in seconds
  net.ipv4.neigh.default.gc_interval

Page cache

- A large percentage of paging activity is due to I/O
  - File reads: each page of file read from disk into memory
  - These pages form the page cache
- Page cache is always checked for IO requests
  - Directory reads
  - Reading and writing regular files
  - Reading and writing via block device files, DISK IO
  - Accessing memory mapped files, mmap
  - Accessing swapped out pages
- Pages in the page cache are associated with file data

Tuning page cache

- View page cache allocation in /proc/meminfo
- Tune length/size of memory
  - vm.lowmem_reserve_ratio
  - vm.vfs_cache_pressure
- Tune arrival/completion rate
  - vm.page-cluster
  - vm.zone_reclaim_mode

- vm.lowmem_reserve_ratio
  - For some specialised workloads on highmem machines it is dangerous for the kernel to allow process memory to be allocated from the "lowmem" zone
  - Linux page allocator has a mechanism which prevents allocations which could use highmem from using too much lowmem
  - The `lowmem_reserve_ratio' tunable determines how aggressive the kernel is in defending these lower zones
  - If you have a machine which uses highmem or ISA DMA and your applications are using mlock(), or if you are running with no swap then you probably should change the lowmem_reserve_ratio setting

- vfs_cache_pressure
  - Controls the tendency of the kernel to reclaim the memory which is used for caching of directory and inode objects
  - At the default value of vfs_cache_pressure=100 the kernel will attempt to reclaim dentries and inodes at a "fair" rate with respect to pagecache and swapcache reclaim
  - Decreasing vfs_cache_pressure causes the kernel to prefer to retain dentry and inode caches
  - When vfs_cache_pressure=0, the kernel will never reclaim dentries and inodes due to memory pressure and this can easily lead to out-of-memory conditions
  - Increasing vfs_cache_pressure beyond 100 causes the kernel to prefer to reclaim dentries and inodes

- page-cluster
  - page-cluster controls the number of pages which are written to swap in a single attempt
  - It is a logarithmic value - setting it to zero means "1 page", setting it to 1 means "2 pages", setting it to 2 means "4 pages", etc
  - The default value is three (eight pages at a time)
  - There may be some small benefits in tuning this to a different value if your workload is swap-intensive

- zone_reclaim_mode
  - Zone_reclaim_mode allows someone to set more or less aggressive approaches to reclaim memory when a zone runs out of memory
  - If it is set to zero then no zone reclaim occurs
  - Allocations will be satisfied from other zones / nodes in the system
  - This is value ORed together of
    - 1 = Zone reclaim on
    - 2 = Zone reclaim writes dirty pages out
    - 4 = Zone reclaim swaps pages

Anonymous pages

- Anonymous pages can be another large consumer of data
- Are not associated with a file, but instead contain:
  - Program data – arrays, heap allocations, etc
  - Anonymous memory regions
  - Dirty memory mapped process private pages
  - IPC shared memory region pages
- View summary usage

  grep Anon /proc/meminfo

  cat /proc/PID/statm

  Anonymous pages = RSS - Shared
- Anonymous pages are eligible for swap

SysV IPC

- Potentially large consumer of memory
  - Semaphores
  - Message queues
  - Shared memory
- View SysV shared memory
  - Active usage

    ipcs
  - Limits

    ipcs -l
- Use POSIX shared memory filesystem tor fast storage
  - Using /dev/shm for temporary storage can be an effective technique to improve service time for critical applications

    # dd if=/dev/zero of=/dev/shm/test bs=1M count=50

Tuning SysV IPC

- Number of semaphores (flags)

  kernel.sem
- Size and number of messages (non-pageable)

  kernel.msgmni * kernel.msgmnb

  kernel.msgmax
- Size and number of shared segments

  kernel.shmmni * kernel.shmmax

  kernel.shmall

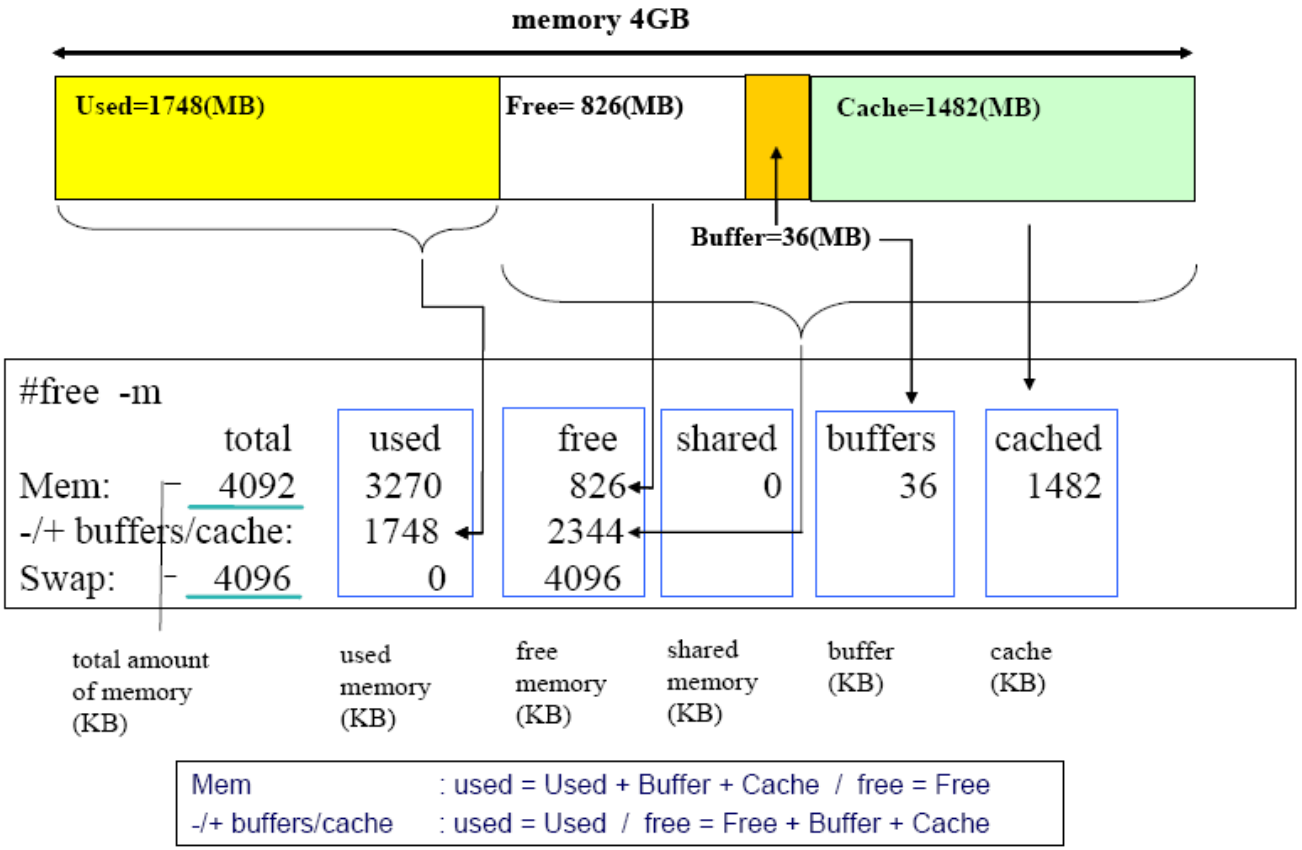
- kernel.sem
  - The maximum number of semaphores per semaphore array, default=250
  - The maximum number of semaphores allowed system-wide, default=32000
  - The maximum number of allowed operations per semaphore system call, default=32
  - The maximum number of semaphore arrays, default=128

- kernel.msgmnb
  - Specifies the maximum number of bytes in a single message queue, default = 16384
- kernel.msgmni
  - Specifies the maximum number of message queue identifiers, default = 16
- kernel.msgmax
  - Specifies the maximum size of a message that can be passed between processes
  - This memory cannot be swapped, default = 8192

- kernel.shmmni
  - Specifies the maximum number of shared memory segments system-wide, default = 4096
- kernel.shmall
  - Specifies the total amount of shared memory, in pages, that can be used at one time on the system, default = 2097152
  - This should be at least kernel.shmmax/PAGE_SIZE
- kernel.shmmax
  - Specifies the maximum size of a shared memory segment that can be created

Viewing memory with free

- Use free -m to view a summary of memory usage in MiB



Other commands to view memory usage

- Page tables

  /proc/vmstat

- System memory

  /proc/meminfo

  - Total physical memory
  - Memory being used by caches
  - Active (in use) vs. inactive

- Summary

  vmstat -s

  sar -r 1 10

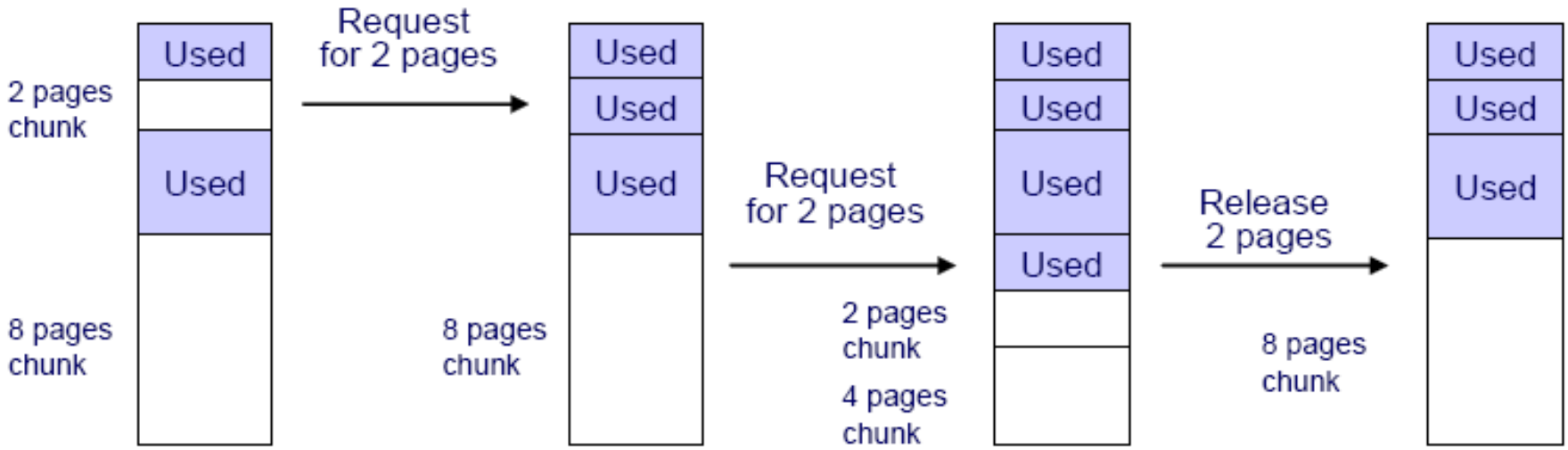
  - Report memory and swap space utilization statistics

Characterizing page status

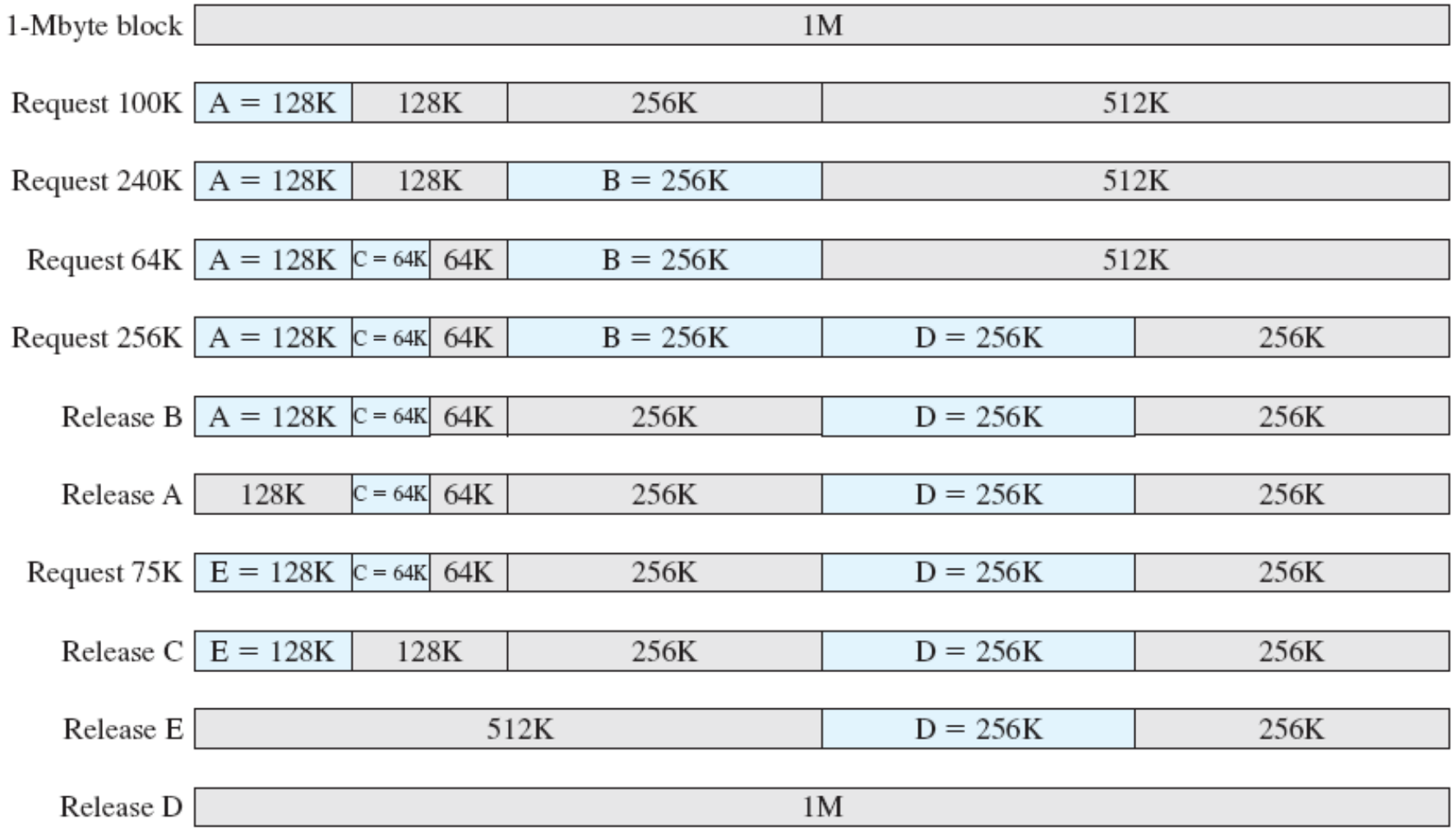
- Free
  - Page is available for immediate allocation
- Inactive Clean
  - Page content has been written to disk, or
  - Has not changed since being read from disk
  - Page is available for allocation
- Inactive Dirty
  - Page not in use, and
  - Page content has been modified but not written back to disk
- Active
  - Page is in use by a process

Buddy system

- The buddy system maintains free pages and tries to allocate pages for page allocation requests
  - It tries to keep the memory area contiguous



Example of Buddy System



Page frame reclaiming

- A kswapd kernel thread swaps out the least recently used pages to maintain a number of free pages
- *kswapd* kernel thread and try_to_free_page() kernel function are responsible for page reclaiming
  - While kswapd is usually sleeping in task interruptible state, it is called by the buddy system when free pages in a zone fall short of a threshold
  - It tries to find the candidate pages to be taken out of active pages based on the Least Recently Used (*LRU*) principle
  - Can take a look at how much memory is considered as active and inactive using the vmstat -a command

- SysV IPC
  - semaphores
    - kernel.sem
  - message queue
    - kernel.msgmni
    - kernel.msgmnb
    - kernel.msgmax
  - shared memory
    - kernel.shmmni
    - kernel.shmmax
    - kernel.shmall
- free -m

- kswapd also follows another principle
  - The pages are used mainly for two purposes: page cache and process address space
  - The page cache is pages mapped to a file on disk
  - The pages that belong to a process address space (called anonymous memory because it is not mapped to any files, and it has no name) are used for heap and stack
  - When kswapd reclaims pages, it would rather shrink the page cache than page out (or swap out) the pages owned by processes

Reclaiming dirty pages

- Using memory as cache creates a strong need to control how pages are reclaimed
  - Memory is volatile
  - Dirty pages need to be written to disk
  - Free up pages for use by other processes
- Handled by **pdflush** kernel threads
  - Default minimum of two threads
  - Additional threads created or destroyed according to IO activity
  - **vm.nr_pdflush_threads** show the current number of pdflush threads

Tuning pdflush

- Tune length/size of memory

  vm.dirty_background_ratio
  - Percentage (of total memory) of number of dirty pages at which pdflush starts writing

  vm.dirty_ratio
  - Percentage (of total memory) of number of dirty pages at which a process itself starts writing out dirty data
- Tune wait time

  vm.dirty_expire_centisecs
  - Interval between pdflush wakeups in 100ths of a second; set to zero to disable writeback
- Tune observation period (between pdflush wakeups)

  vm.dirty_writeback_centisecs
  - Defines when data is old enough, in 100ths of a second, to be eligible for writeout bye pdflush

Reclaiming clean pages

- 1. Flush all dirty buffers and pages
  - Sync command
  - fsync system call
  - Alt-SysRq-S magic system request
    echo s > /proc/sysrq-trigger
- 2. Reclaim clean pages
  echo 3 > /proc/sys/vm/drop_caches
    - 1 to free pagecache
    - 2 to free dentries and inodes
    - 3 to free pagecache, dentries and inodes
  - Eliminate bad data from cache
  - Reduce memory before laptop hibernate
  - Benchmark subsystem

Out-of-memory killer

- Kills processes if
  - All memory (incl.swap) is active
  - No pages are available in ZONE_NORMAL
  - No memory is available for page table mappings
- Interactive processes are preserved if possible
  - High sleep average indecates interactive
  - View level of immunity from oom-kill
    /proc/PID/oom_score
- Manually invoking oom-kill
  echo f > /proc/sysrq-trigger
  - Will call oom_kill to kill a memory hog process
  - Does not kill processes if memory is avaliable
  - Outputs verbose memory information in /var/log/messages

Tuning OOM policy

- Protect daemons from oom-kill

  echo n > /proc/PID/oom_adj

  - The process to be killed in an out-of-memory situation is selected based on its **badness score**
  - oom_score gets multiplied by 2^n
  - Caution: child processes inherit oom_adj from parent
- Disable oom-kill in /etc/sysctl.conf

  vm.panic_on_oom = 1
- oom-kill is not a fix for memory leaks

Detecting memory leaks

- Two types of memory leaks
  - Virtual: process requests but does not use virtual address space (vsize)
  - Real: process fails to free memory (rss)
- Use sar to observer system-wide memory change

  sar -R 1 120
  - Report memory statistics
- Use watch with ps or pmap

  watch -n1 'ps axo pid,comm,rss,vsize | grep httpd'
- Use valgrind
  - valgrind --tool=memcheck cat /proc/$$/maps

What is swap?

- The unmapping of page frames from an active process
  - Swap-out: page frames are unmapped and placed in page slots on a swap device
  - Swap-in: page frames are read in from page slots on a swap device and mapped into a process address space
- Which pages get swapped?
  - Inactvie pages
  - Anonymous pages
- Swap cache
  - Contains unmodified swapped-in pages
  - Avoids race conditions when multiple processes access a common page frame

Improving swap performance

- Defer swap until think time
  - Frequent, small swaps
  - Swap anonymous pages more readily
- Reduce visit count
  - Distribute swap areas across maximum of 32 LUNs
  - Assign an equal, high priority to multiple swap areas
  - Kernel uses highest priority first
  - Kernel uses round-robin forswap areas of equal priorty
- Reduce service time
  - Use partitions, not files
  - Place swap areas on lowest numbered partitions of fastest LUNs

Tuning swappiness

- Searching for inactive pages can consume the CPU
  - On large-memory boxes, finding and unmapping inactive pages consumes more disk and CPU resources than writing anonymous pages to disk
- Prefer anonymous pages (higher value)

  vm.swappiness
  - Linux prefer to swap anonymous pages when:

    % of memory mapped into page tables + vm.swappiness >= 100
- Consequences
  - Reduced CPU utilization
  - Reduced disk bandwidth

Tuning swap size

- Considerations
  - Kernel uses two bytes in ZONE_NORMAL to track each page of swap
  - Storage bandwidth cannot keep up with RAM and can lead to swaplock
  - If memory shorage is severe, kernel will kill user mode process
- general guidelines
  - Batch compute servers: up to 4 * RAM
  - Database server: <= 1 GiB
  - Application server: >= 0.5 * RAM
- Consequences
  - Avoid swaplock

Tuning swap for think time

- Swap smaller amounts

  vm.page_cluster
- Protect the current process from paging out

  vm.swap_token_timeout
  - Used to control how long a process is protected from paging when the system thrashing, measured in seconds
  - The value would be useful to tune thrashing behavior
- Consequences
  - Smoother swap behavior
  - Enable current process to clean up its pages

Tuning swap visit count

- Create up to 32 swap devices
- Make swap signatures

  mkswap -L *SWAP_LABEL* */path/to/device*
- Assign priority in /etc/fstab

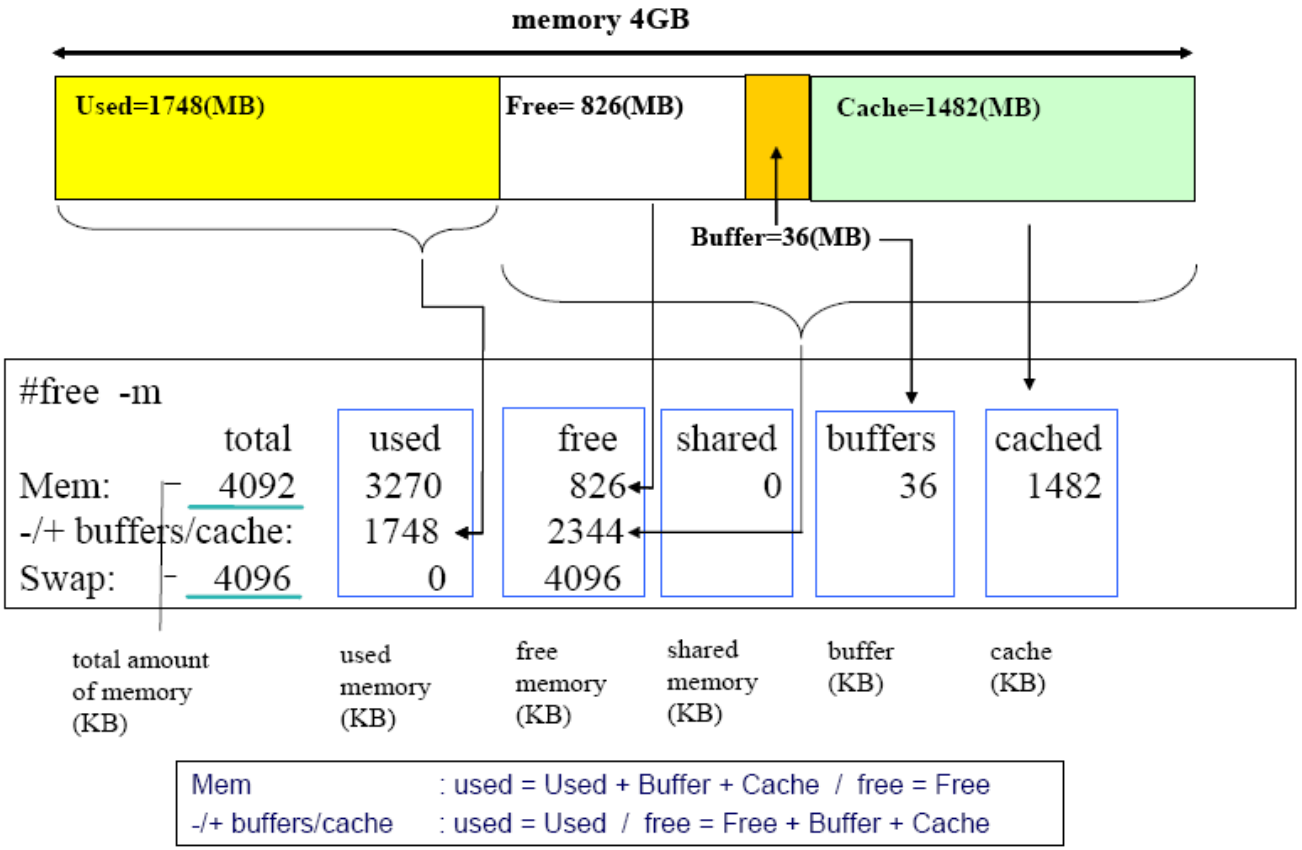
```
/dev/sda1              swap   swap   pri=5  0  0
/dev/sdb1              swap   swap   pri=5  0  0
LABEL=testswap         swap   swap   pri=5  0  0
/swaps/swapfile1       swap   swap   pri=1  0  0
```

- Active

  swapon -a
  - View active swap devices in /proc/swaps

Viewing memory with free

- Use free -m to view a summary of memory usage in MiB



- memory:
  - 32:
    - ZONE_DMA (16M)
    - ZONE_NORMAL (896M)
    - ZONE_RESERVED (1024M)
    - ZONE_HIGHMEM
  - 64:
    - ZONE_DMA (1G)
    - ZONE_DMA32 (4G)
    - ZONE_NORMAIL
- cat /proc/zoneinfo
- 4k
- 1G: 256*1024
- 4G: 2^20
- 64G:
- HugePage: 大页面
  - THP
  - 2M(), 1G(TB级)

Monitoring memory usage

- Memory activity

  vmstat -n [interval] [count]

  sar -r [interval] [count]

  - Report memory and swap space utilization statistics
- Rate of change in memory

  sar -R [interval] [count]

  - Report memory statistics
- Swap activity

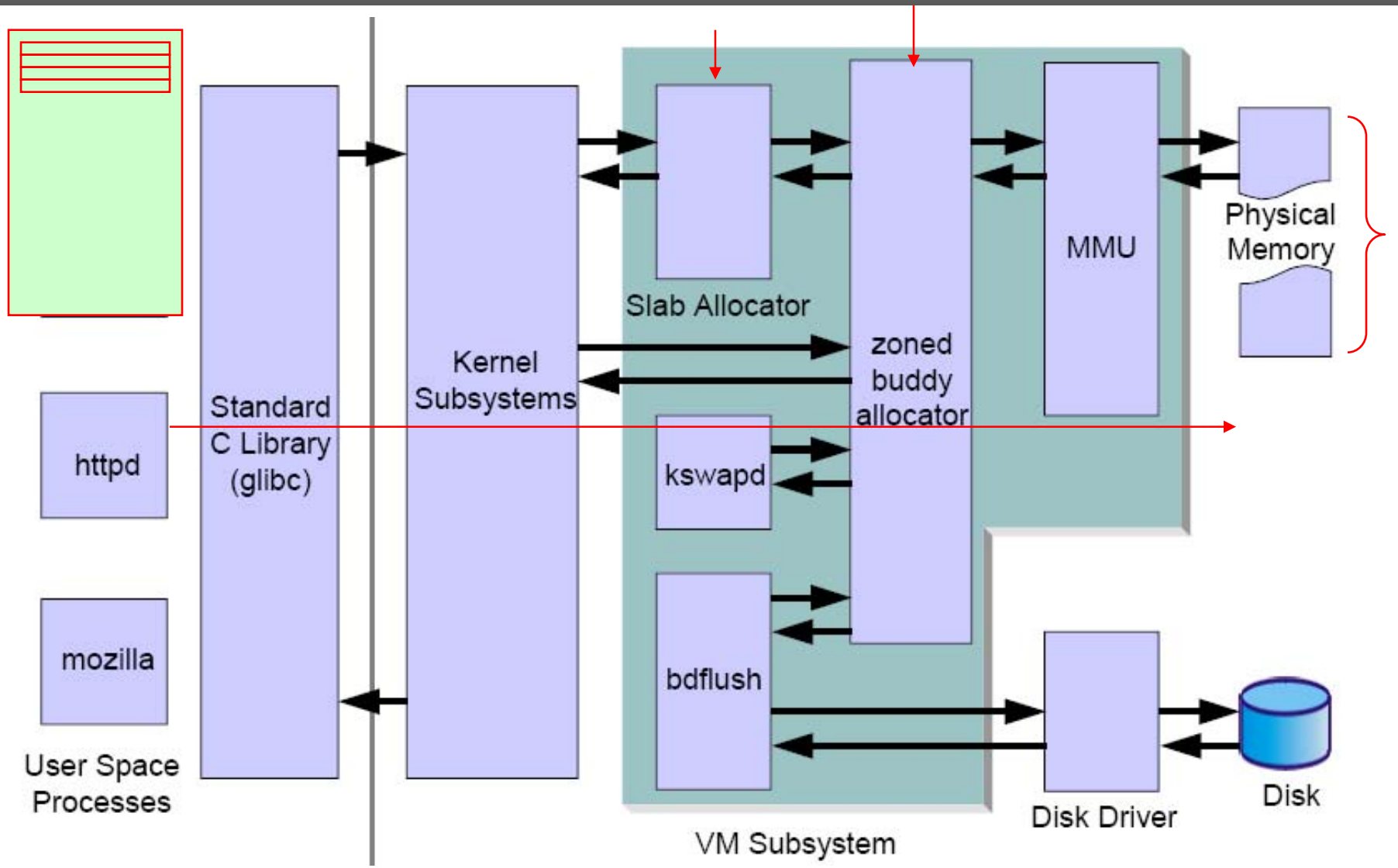
  sar -W [interval] [count]

  - Report swapping statistics
- All IO

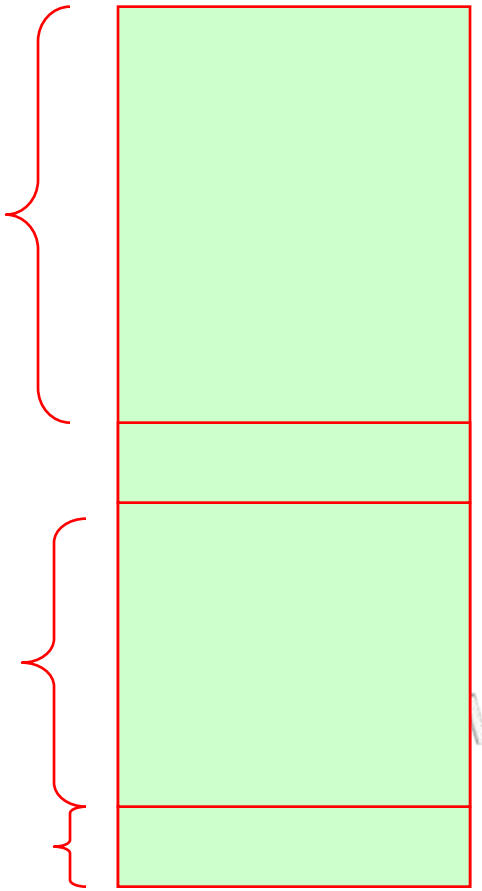
  sar -B [interval] [count]

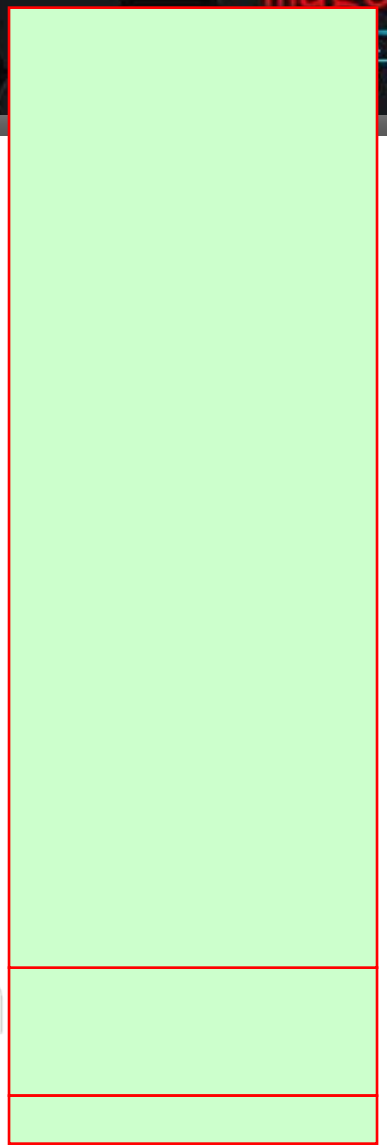
  - Report paging statistics

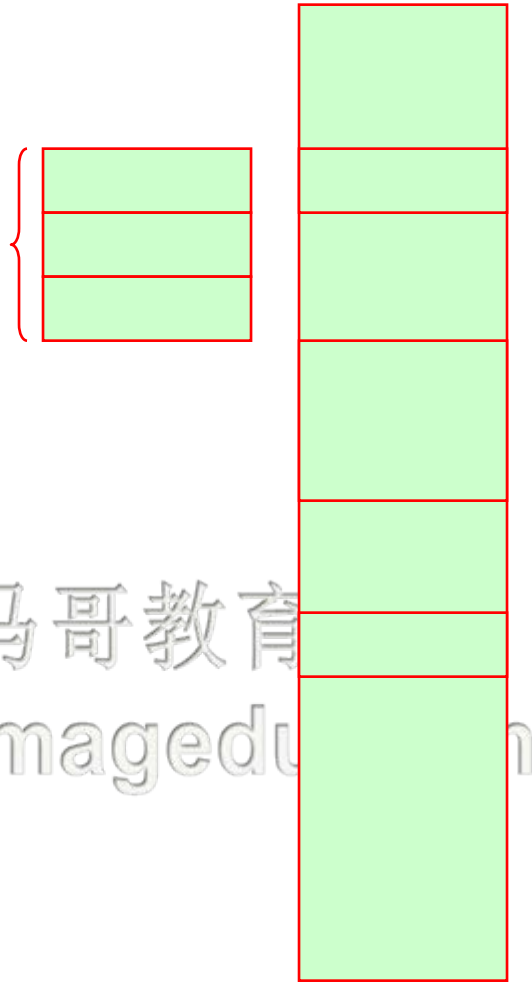
Virtual memory manager



64G, PAE (Physical Address Extension)







马哥教育

I/O Architecture and Device Drivers

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

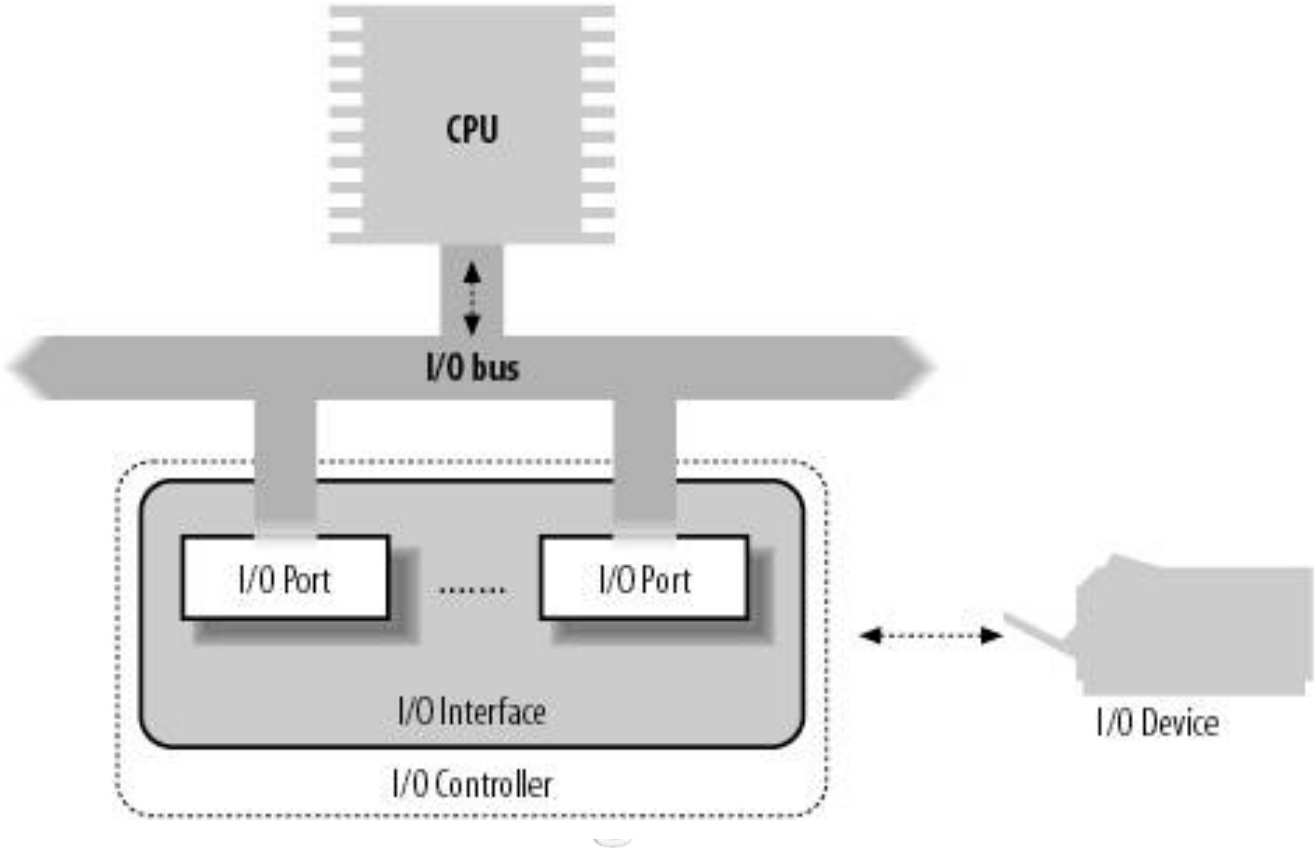
Overview

- The Virtual File System depends on lower-level functions to carry out each read, write, or other operation in a manner suited to each device
- To make a computer work properly, data paths must be provided that let information flow between CPU(s), RAM, and the score of I/O devices that can be connected to a personal computer
- These data paths, which are denoted as the **buses**, act as the primary communication channels inside the computer

- Any computer has a system bus that connects most of the internal hardware devices
  - A typical system bus is the PCI (Peripheral Component Interconnect) bus
  - Several other types of buses, such as ISA, EISA, MCA, SCSI, and USB, are currently in use
  - Typically, the same computer includes several buses of different types, linked together by hardware devices called bridges
  - Two high-speed buses are dedicated to the data transfers to and from the memory chips
    - the frontside bus connects the CPUs to the RAM controller
    - the backside bus connects the CPUs directly to the external hardware cache
  - The host bridge links together the system bus and the frontside bus

- The data path that connects a CPU to an I/O device is generically called an I/O bus
- The 80 x 86 microprocessors use 16 of their address pins to address I/O devices and 8, 16, or 32 of their data pins to transfer data
- The I/O bus, in turn, is connected to each I/O device by means of a hierarchy of hardware components including up to three elements: I/O ports , interfaces, and device controllers

PC's I/O architecture

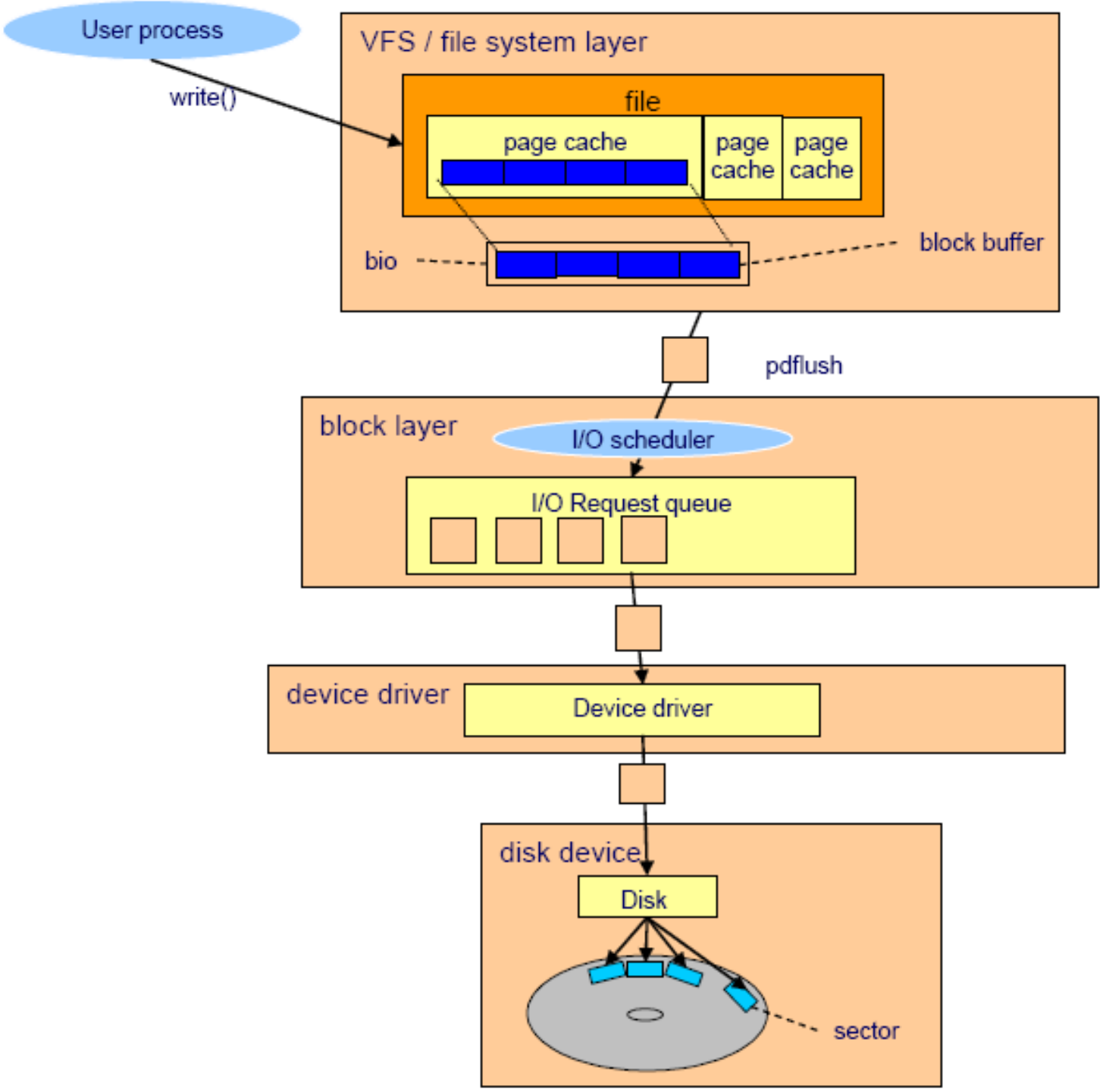


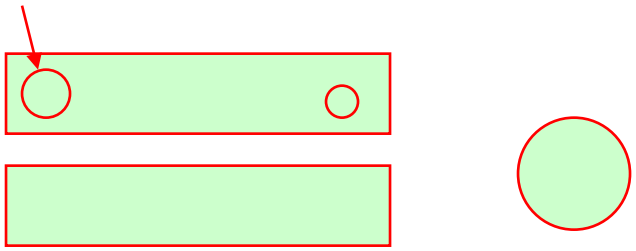
I/O Ports

- Each device connected to the I/O bus has its own set of I/O addresses, which are usually called I/O ports
- In the IBM PC architecture, the I/O address space provides up to 65,536 8-bit I/O ports
- I/O ports may also be mapped into addresses of the physical address space
  - The processor is then able to communicate with an I/O device by issuing assembly language instructions that operate directly on memory (for instance, mov, and, or, and so on)
  - Modern hardware devices are more suited to mapped I/O, because it is faster and can be combined with DMA

Access mode affects disk IO

- Character
  - Sequential IO
  - High overhead, low latency
    - The kernel will read or write one character to or from the device per device interrupt
  - Depending on size of transfers, IO may be buffered
  - Simple buffering provides little opportunity for tuning
- Block
  - Random IO, the unit of transfer files is a block
    - For each interrupt, a block's worth of data is delivered
    - Block size can range from 1KiB to PAGE_SIZE
  - High average access time but high throughput
  - The page is the basic unit used for IO requests
  - Complex buffering provides ample opportunity for tuning
    - The most general purpose filesystems an associated buffer is created in memory whenever the block device is mounted


User process
write()
VFS / file system layer
file
page cache
page cache
page cache
bio
block buffer
pdflush
block layer
I/O scheduler
I/O Request queue
device driver
Device driver
disk device
Disk
sector



Physical factors affect disk IO

- Electro-mechanical positioning
  - Rotational delay
  - Seek time
- Storage density: zoned constant angular velocity (ZCAV) drives
  - Outer tracks contain more linear storage area
  - For a given rotational speed, read more data per second on outer tracks
  - Lower partition numbers are on outside
- Remember the forced flow law!
  - The linux kernel handles more than 10Gib/s for a single stream
  - The bus interconnect may be a bottleneck

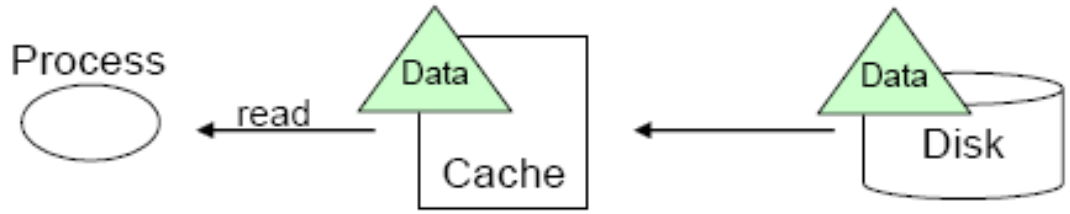
Flushing a dirty buffer

- When a process reads data from disk, the data is copied to memory
  - The process and other processes can retrieve the same data from the copy of the data cached in memory
  - When a process tries to change the data, the process changes the data in memory first
  - At this time, the data on disk and the data in memory is not identical and the data in memory is referred to as a *dirty buffer*
  - The dirty buffer should be synchronized to the data on disk as soon as possible, or the data in memory could be lost if a sudden crash occurs

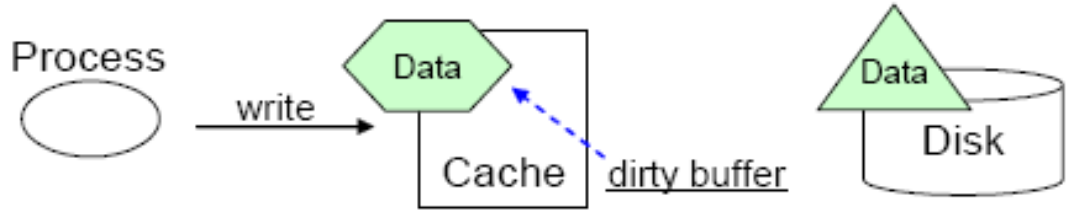
- The synchronization process for a dirty buffer is called flush
- In the Linux kernel 2.6 implementation, *pdflush* kernel thread is responsible for flushing data to the disk
  - The flush occurs on a regular basis (kupdate) and when the proportion of dirty buffers in memory exceeds a certain threshold (bdflush). The threshold is configurable in the **/proc/sys/vm/dirty_background_ratio** file

Flushing dirty buffers

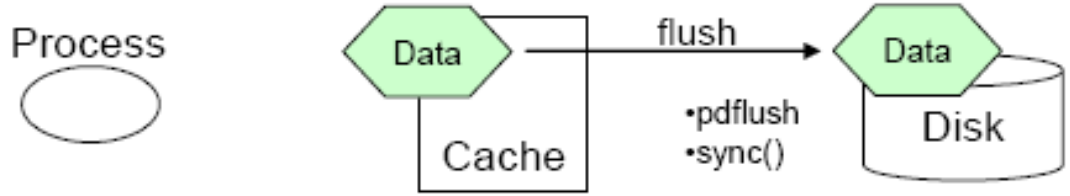
- Process read data from disk
  The data on memory and the data on disk are identical at this time.



- Process writes new data
  Only the data on memory has been changed, the data on disk and the data on memory is not identical.



- Flushing writes the data on memory to the disk.
  The data on disk is now identical to the data on memory.



Block layer

- The block layer handles all the activity related to block device operation
- The key data structure in the block layer is the *bio* structure
- The bio structure is an interface between the file system layer and the block layer
  - When a write is performed, the file system layer tries to write to the page cache which is made up of block buffers
  - It makes up a bio structure by putting the contiguous blocks together, then sends bio to the block layer

Block device IO transfers

- Two distinct steps to service an IO reques
  - 1. Place the IO request on a request queue
    - Each block device has its own request queue that combines reads and writes
    - Empty request queues are plugged before a new request is added
  - 2. Perform the transfer at some later point in time
- Direct IO bypasses the buffer layer
  - Used for self-caching applications
  - Avoids duplicate content in page frames
  - Read/Write system calls are slowed down by page cache and read-ahead

Improving buffer efficiency

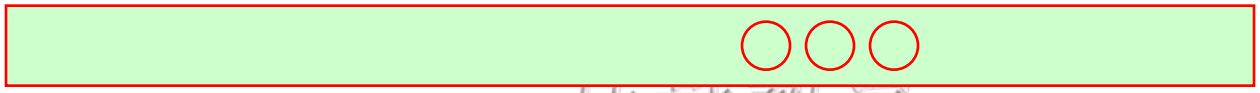
- Use larger request queue length
  - Allows reads to be sorted before wirtes
  - Longer queues allow more efficient re-sorting
  - Introduces more latency
- Kernel automatically reads ahead sequentially
  - Fewer seeks
  - Fewer operations issued to disk controller
  - Reduces wait time
  - Stopped when random file access is detected

Tuning buffers

- Queue length

  /sys/block/<dev>/queue/nr_requests
- Max read-ahead

  /sys/block/<dev>/queue/read_ahead_kb
  - Note: initial read-ahead window size is half the max
- Persist in /etc/rc.local



The IO scheduler

- The block layer handles the bio request and links these requests into a queue called the I/O request queue
- This linking operation is called I/O elevator
- Compensate for physical factors by re-sorting request queue
  - Referred to as merging
  - Measuring rrqm/s and wrqm/s

    iostat -x sda 1 10
- Conflicting goals
  - Keep disk head moving in one direction as long as possible,but
  - Prevent IO starvation (i.e., minimize wait time)

The Job of an I/O Scheduler

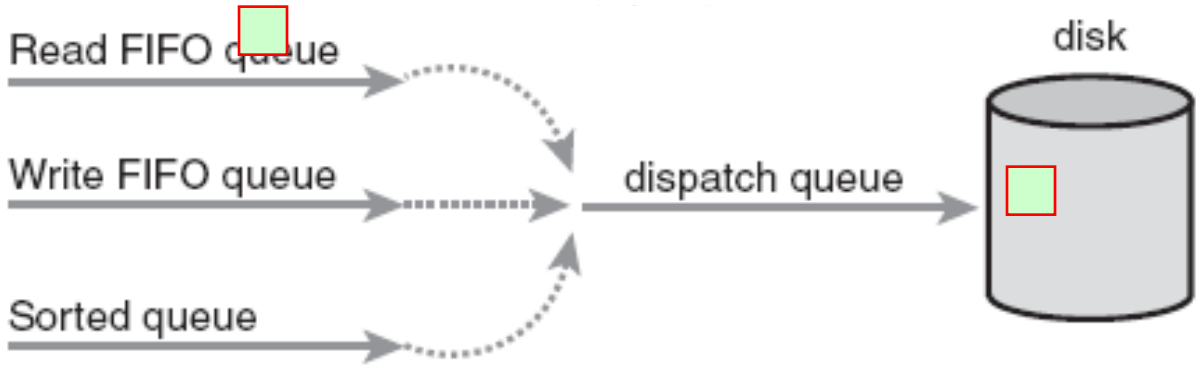
- An I/O scheduler works by managing a block device's request queue
  - It decides the order of requests in the queue and at what time each request is dispatched to the block device
- It manages the request queue with the goal of reducing seeks, which results in greater global throughput
- I/O schedulers perform two primary actions to minimize seeks: merging and sorting

Sorting the IO queues

- One elevator algorithm per LUN or whole block device
- CFQ(default)
  - Divides IO bandwidth equally among all processes
- Deadline
  - Optimize read and wirte queue times
  - Request must be serviced when expiration time is reached
  - When possible, multiple requests are served from new location
- Anticipatory
  - Optimize service time for dependent reads
  - After servicing an IO, wait a short time to give the process a chance to submit another IO
- NOOP
  - IO requests are placed in queue with minimal processing
  - Schedualing is left to the hardware
  - Use only when CPU clock cycles are too expencive

The Deadline I/O Scheduler

- In the Deadline I/O scheduler, each request is associated with an expiration time
  - By default, the expiration time is 500 milliseconds in the future for read requests and 5 seconds in the future for write requests
- It is a tough act to provide request fairness, yet maximize global throughput



Tuning the deadline scheduler

- Selection

  echo deadline > /sys/block/<dev>/queue/scheduler
- Tunables in /sys/block/<dev>/queue/iosched/
  - Max queue time

    read_expire (in ms)

    write_expire (in ms)
  - Number of requests to move from the scheduler list to the request queue

    fifo_batch
  - How many reads to allow before a write is dispatched

    wirtes_startved
  - Should we attempt to front-merge contiguous requests?
    - front_merges

- The Anticipatory I/O Scheduler
  - The anticipatory I/O elevator was created based on the assumption of a block device with only one physical seek head (for example a single SATA drive)
  - The Anticipatory I/O scheduler aims to continue to provide excellent read latency, but also provide excellent global throughput
  - The Anticipatory I/O scheduler attempts to minimize the seek storm that accompanies read requests issued during other disk I/O activity
    - When a read request is issued, it is handled as usual, within its usual expiration period
    - After the request is submitted, however, the Anticipatory I/O scheduler does not immediately seek back and return to handling other requests
    - Instead, it does absolutely nothing for a few milliseconds

Tuning the anticipatory scheduler

- Selection

  echo anticipatory > /sys/block/<dev>/queue/scheduler

- Tunables in /sys/block/<dev>/queue/iosched/
  - How long to wait for another, nearby read

    antic_expire
  - Max queue time

    read_expire

    write_expire
  - How long to batch-process reads or writes

    read_batch_expire

    write_batch_expire

- The Complete Fair Queuing I/O Scheduler
  - The CFQ I/O scheduler assigns incoming I/O requests to specific queues based on the process originating the I/O request
  - The CFQ I/O scheduler then services the queues round robin, plucking a configurable number of requests (by default, four) from each queue before continuing on to the next
  - This provides fairness at a per-process level, assuring that each process receives a fair slice of the disk's bandwidth

Tuning the cfq scheduler

- Selection

  echo cfq > /sys/block/<dev>/queue/scheduler

  - Uses 64 internal queues
  - Fills internal queues using round-robin
  - Requests are dispatched from non-empty queues
  - Sort occurs at dispatch queue

- Tunables in /sys/block/<dev>/queue/iosched/

  - Nubmer of requests dispatched to device per cycle

    quantum

    - slice_idle = 1
    - quantum = 64
    - group_idle = 1

- The Noop I/O Scheduler
  - The Noop I/O scheduler does not perform sorting or any other form of seek-prevention whatsoever
  - In turn, it has no need to implement anything akin to the slick algorithms to minimize request latency that you saw in the previous three I/O schedulers
  - The Noop I/O scheduler does perform merging, however, as its lone chore
  - It is intended for block devices that are truly random-access, such as flash memory cards

Tuning the noop scheduler

- Selection

  echo noop > /sys/block/<dev>/queue/scheduler

  - No tunable settings required
- Use when CPU clock cycles are "too expensive"
  - Host CPU cycles are usually cheaper than SAN CPU cycles
  - Sorting is still usefule for iSCSI
- Some controllers perform elevator functions
  - Tagged command queuing
  - Available on SCSI and some SATA drives

I/O Scheduler Selection

- Each of these I/O schedulers can be enabled and built into the kernel
- By default, block devices use the Complete Fair Queuing I/O scheduler
- This can be overridden via the boot-time option elevator= *ele_name*

Parameter	I/O Scheduler
as	Anticipatory
cfq	Complete Fair Queuing
deadline	Deadline
noop	Noop

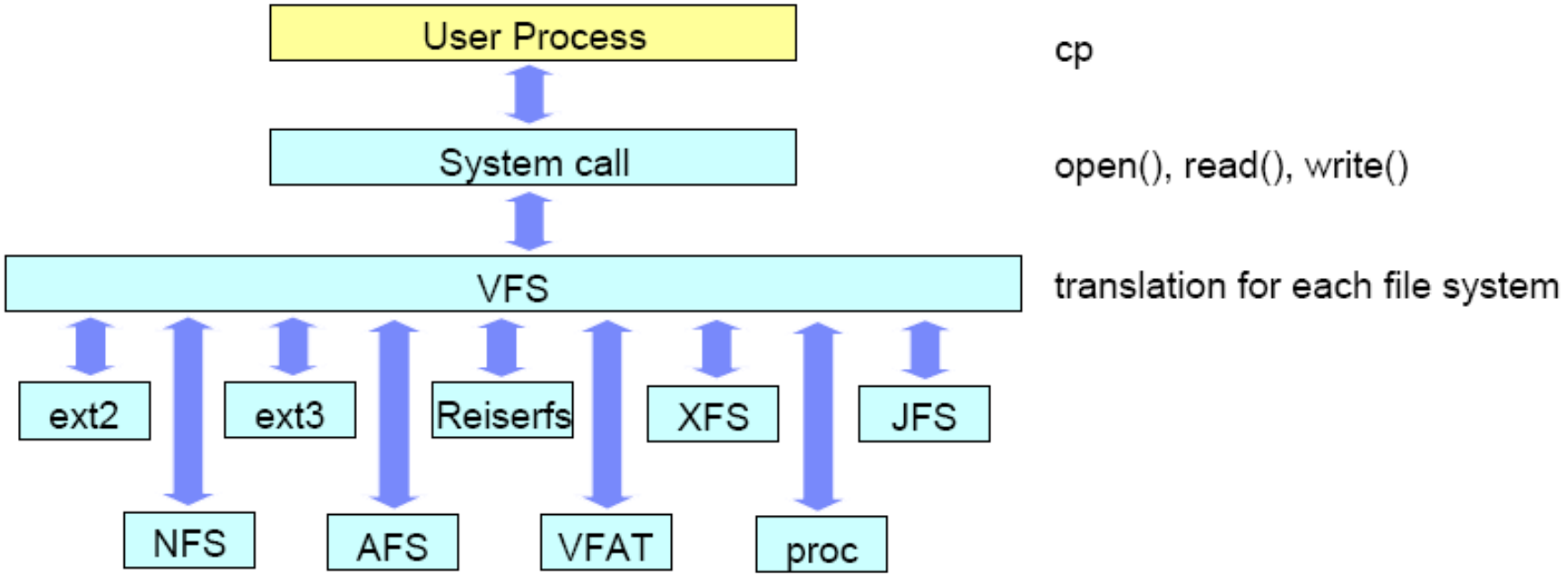
马哥教育

Linux File Systems

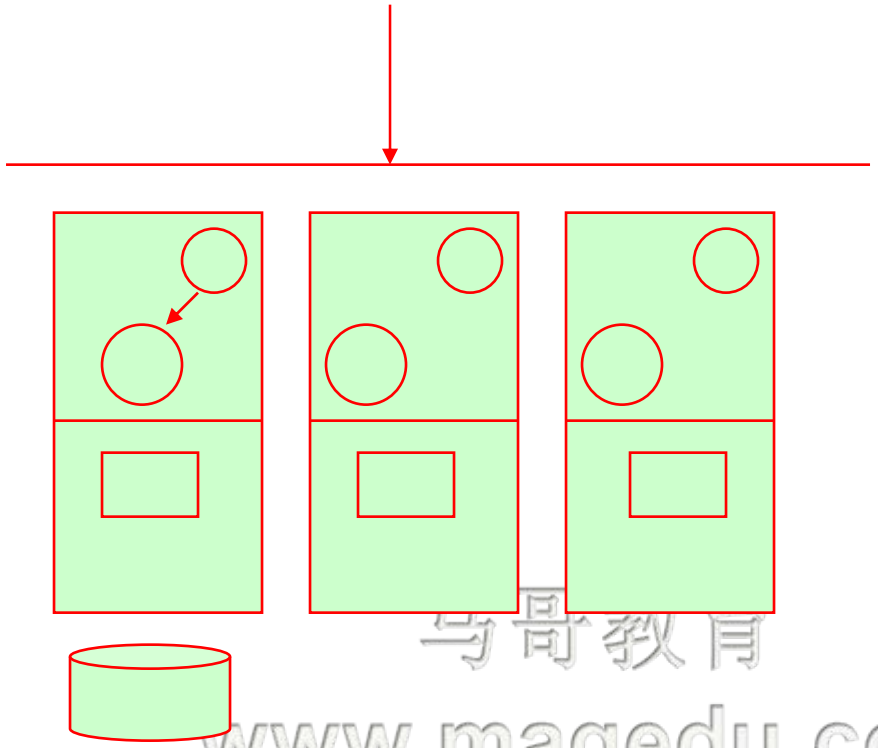
主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

Virtual file system

- Virtual Files System (VFS) is an abstraction interface layer that resides between the user process and various types of Linux file system implementations

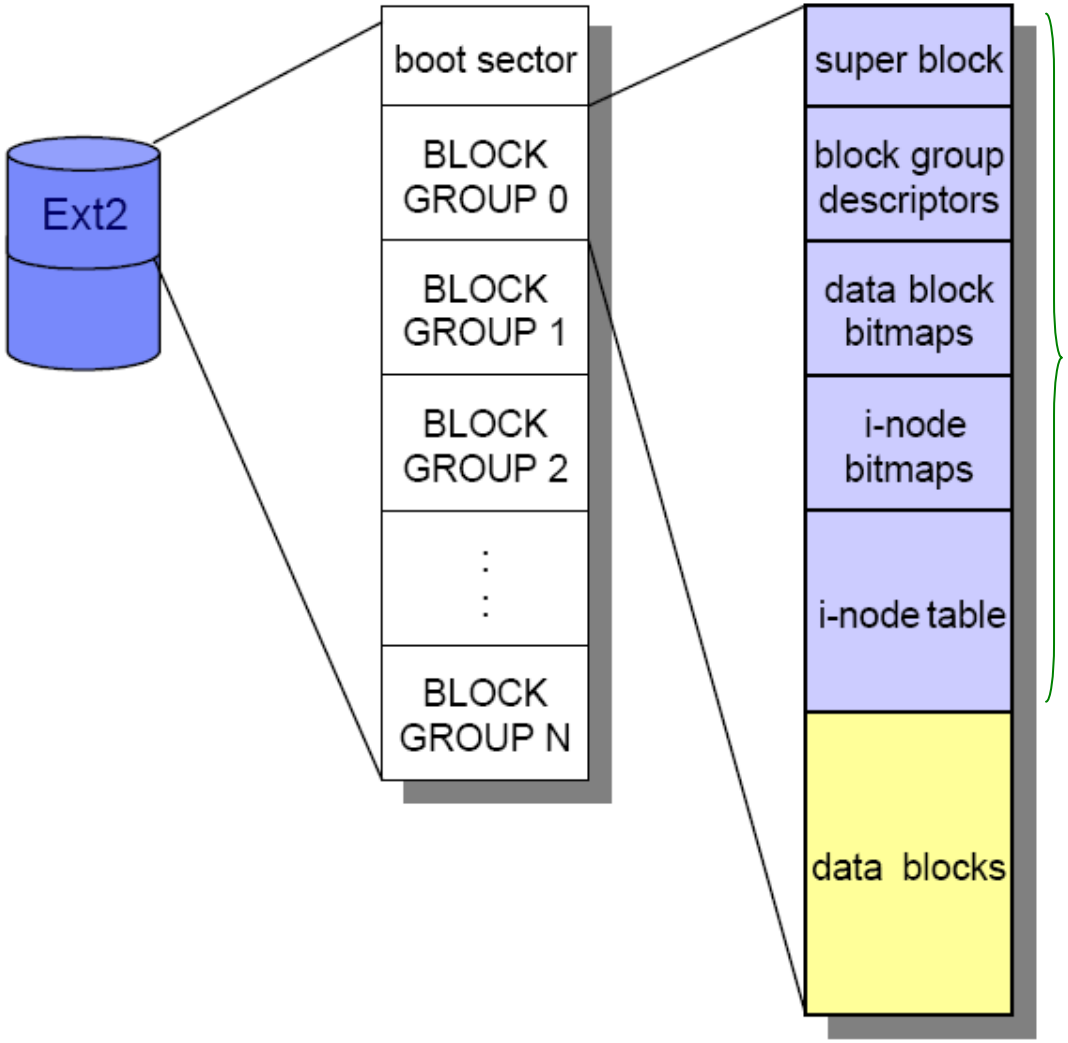


- mogilefs：块 (block)

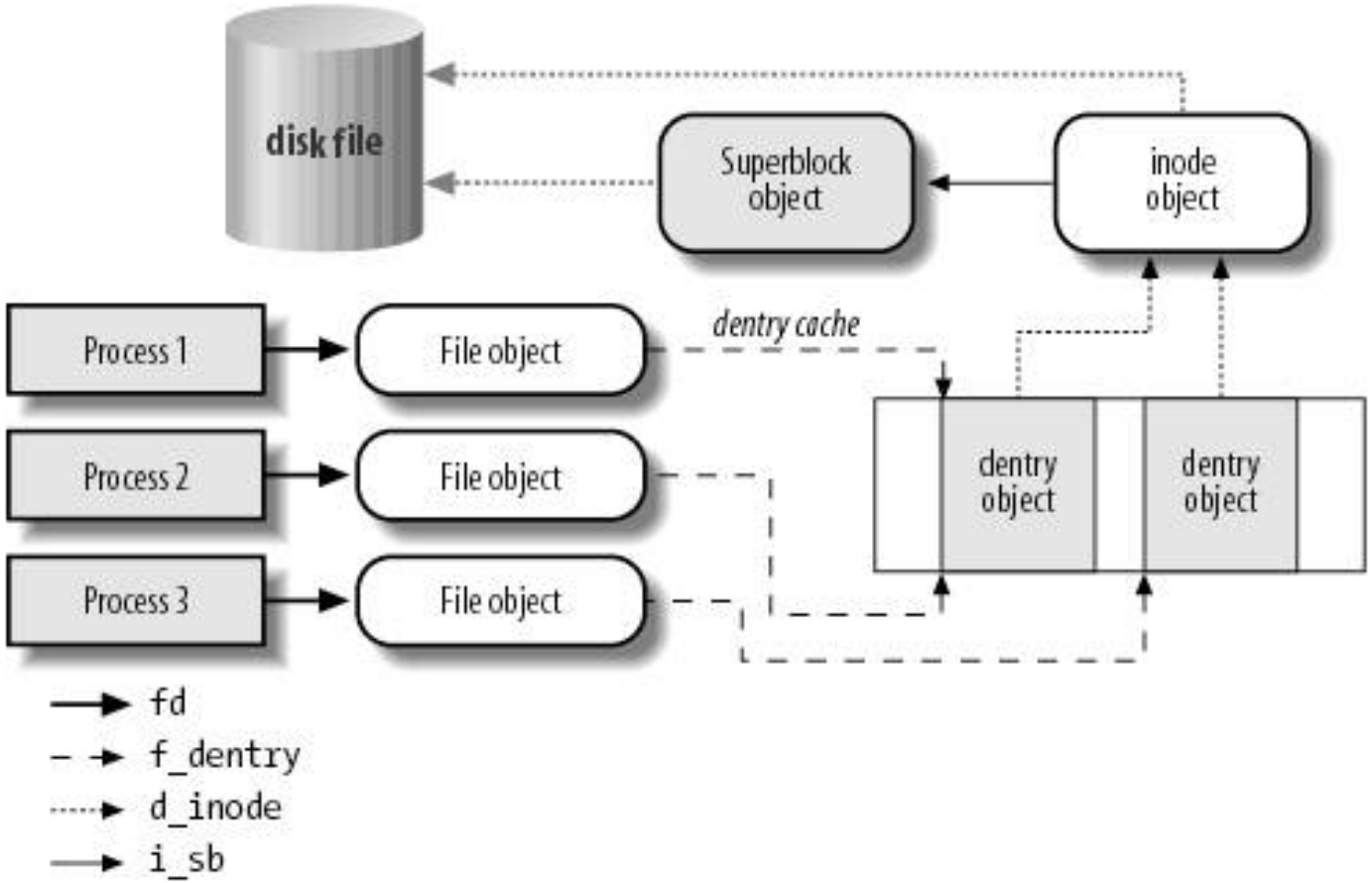


Ext2

- ext


Ext2
boot sector
BLOCK GROUP 0
BLOCK GROUP 1
BLOCK GROUP 2
BLOCK GROUP N
super block
block group descriptors
data block bitmaps
i-node bitmaps
i-node table
data blocks

Interaction between processes and VFS objects



Fragmentation

- Increates average time of sequential reads
- Ext2 and ext3 use reserved space to reduce fragmentation
  - Pre-allocated up to 8 blocks
  - Unused blocks are freed when file is closed
  - At file creation, inode is created in a block group different than directory (if possible)
  - When extending a file, uses blocks in same block group if possible
  - Problematic if partition is close to full

Viewing fragmentation

- For a specific file

  filefrag -v /path/to/file

- For a mounted filesystem

  dumpe2fs /dev/some_device_partition

- For an unmounted filesystem (boot-time)

  fsck /dev/some_device

- mount -o noatime
- nodiratime
- mount -o commit=15

Tuning fragmentation

- At filesystem creation

  mke2fs -m *reserved-percentage*

- After filesystem creation

  tune2fs -m *reserved-percentage*

  tune2fs -r *reserved-block-count*

- Considerations
  - Too low may degrade performance
  - Too high wastes capacity

Filesystem limits

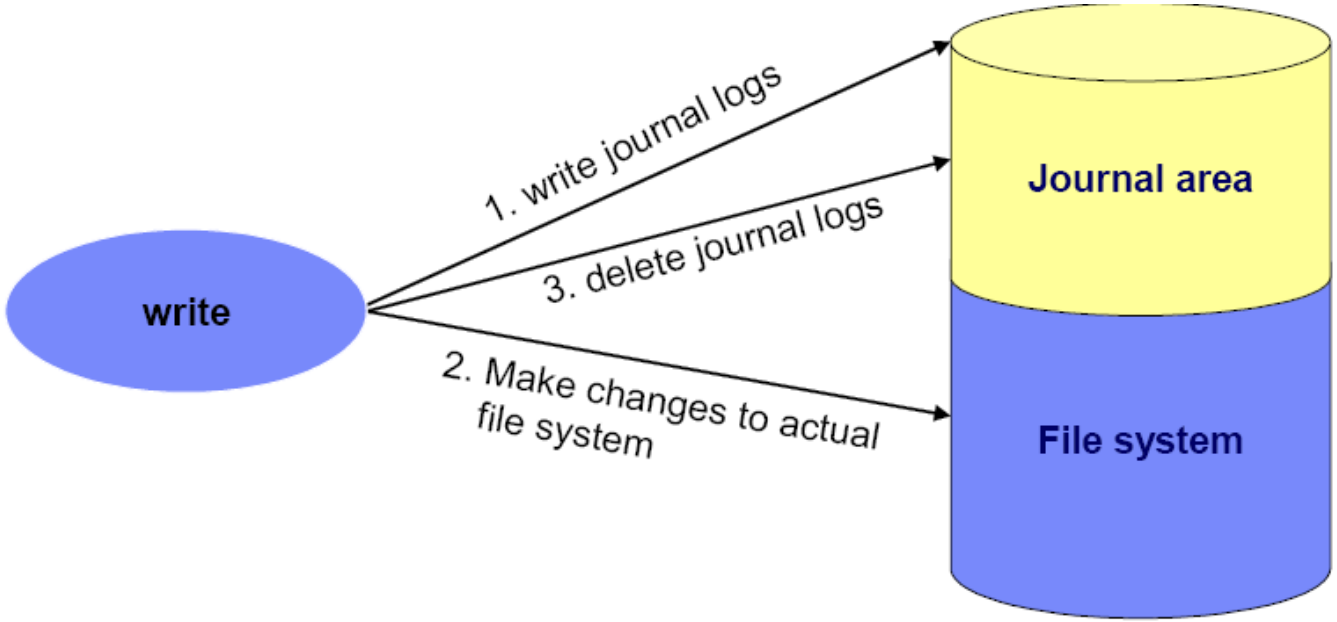
- Maximum file size
  - ext3: 2 TiB
  - GFS: 16 TiB (x86/32-bit) or 8 EiB (x86/64-bit)
- Maximum file system size
  - ext3: 16 TiB
  - GFS: 16 TiB (x86/32-bit) or 8 EiB (x86/64-bit)

Journaling

- In a non-journaling file system, when a write is performed to a file system the Linux kernel makes changes to the file system metadata first and then writes actual user data next
- A Journaling file system writes data to be changed to the area called the journal area before writing the data to the actual file system
  - The journal area can be placed both in the file system or out of the file system
  - The data written to the journal area is called the journal log
  - It includes the changes to file system metadata and the actual file data if supported

Journaling concept

- Journaling involves two writes to disk
  - Write modified blocks to journal
  - Write modified blocks to filesystem
  - Discard blocks in journal



Mode of journaling

- Ext3 supports three types of journaling modes
  - journal
    - Provides the highest form of data consistency by causing both file data and metadata to be journaled
    - It also has higher performance overhead
  - ordered
    - In this mode only metadata is written
    - However, file data is guaranteed to be written first
    - This is the default setting
  - writeback
    - Provides the fastest access to the data at the expense of data consistency

Journaling

- Which blocks?
  - Only the metadata (default)

    mount -o data=ordered
  - Data and metadata

    mount -o data=journal
  - Only the metadata, but no guarantee for order of commits

    mount -o data=writeback
- Consequence
  - Reduce time for consistency check after a system crash
  - data=journal avoids expensive seeks by resorting small, random writes

Improving journal performance

- By default, journal is located at an inode within the filesystem
  - Journal size can be from 1024 to 102400 blocks

    mke2fs -b 4096 -J size=32 /dev/sdb1 (Note: size in MiB)
- Place journal on a separate device
  - Data journaling is a mount-time option, so journal device and filesystem must use identical block size
  - Entire partition must be dedicated to journal device
  - Major:Minor device number of journal device is recorded in filesystem superblock
- Consequences
  - Reduce visit count
  - Reduce service time for both journal and filesystem

Tuning journal placement

- 1. Umount filesystem
- 2. Confirm filesystem block size and journal placement

  dumpe2fs /dev/sdb1 | less
- 3. Remove the internal journal from the existing filesystem

  tune2fs -O ^has_journal /dev/sdb1
- 4. Create an external journal device

  mke2fs -O journal_dev -b *block-size* /dev/sdc1
- 5. Update the filesystem superblock to use the external journal

  tune2fs -j -J device=/dev/sdc1 /dev/sdb1

Other mount-time options

- Consider disabling access time updates

  mount -o noatime

  - May have a significant impace on ext3 performance
- Mount ext3 with longer period between journal commits

  mount -o commit=15

  - Default is 5 seconds
  - Longer may improve performance but result in more lost data if system crashes

Finding lock contention

- Types of locks
  - Advisory (cooperative)
  - Mandatory (enforced by kernel)

    chmod g+s-x /path/to/file
- Heavily used by databases
- View current locks held by processes

  cat /proc/locks

Filesystems comparison using iozone

- iozone
  - IOzone is a filesystem benchmark tool
    - The benchmark generates and measures a variety of file operations
  - Iozone is useful for performing a broad filesystem analysis of a vendor's computer platform
  - The benchmark tests file I/O performance for the following operations:
    - *Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read, aio_write*

The options of iozone

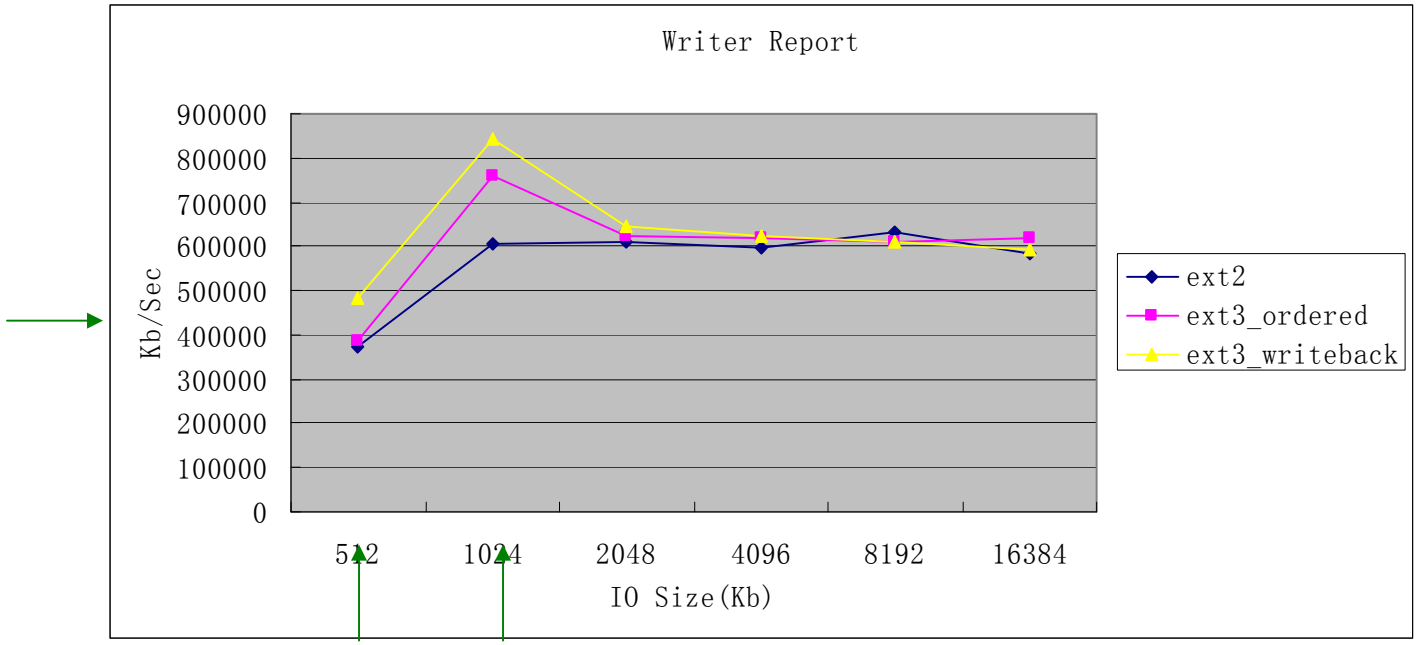
- -a
  - Used to select full automatic mode
  - Produces output that covers all tested file operations for record sizes of 4k to 16M for file sizes of 64k to 512M
- -b filename
  - Creates a binary file format file in Excel compatible output of results
- -R
  - Generate Excel report
  - Iozone will generate an Excel compatible report to standard out
- -z
  - Used in conjunction with -a to test all possible record sizes

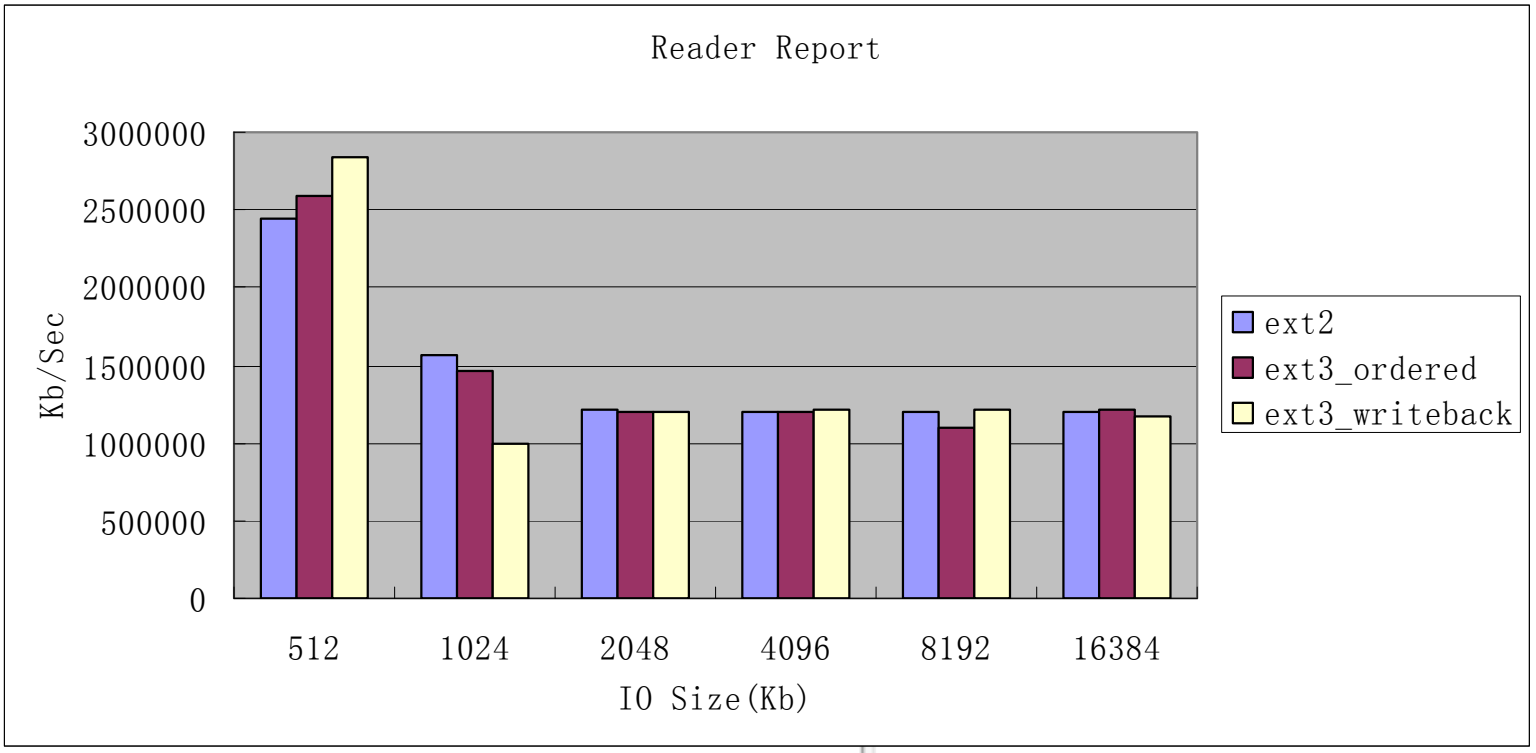
- -f filename
  - Used to specify the filename for the temporary file under test
- -g #
  - Set maximum file size (in Kbytes) for auto mode
- -n #
  - Set minimum file size (in Kbytes) for auto mode
- -s #
  - Used to specify the size, in Kbytes, of the file to test
  - One may also specify -s #k (size in Kbytes) or -s #m (size in Mbytes) or -s #g (size in Gbytes)

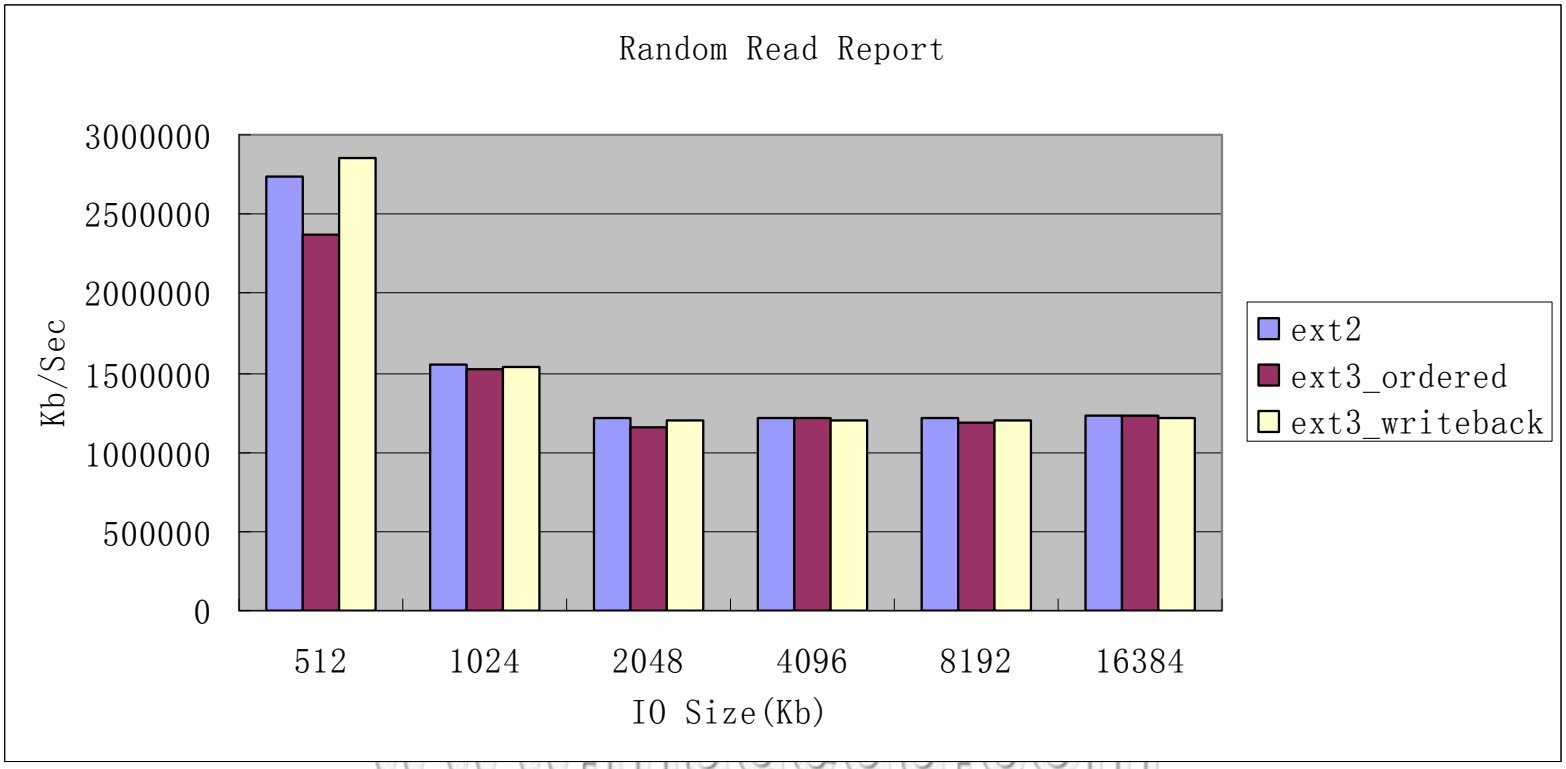
- -y #
  - Set minimum record size (in Kbytes) for auto mode
  - One may also specify -y #k (size in Kbytes) or -y #m (size in Mbytes) or -y #g (size in Gbytes)
- -q #
  - Set maximum record size (in Kbytes) for auto mode
- -r #
  - Used to specify the record size, in Kbytes, to test

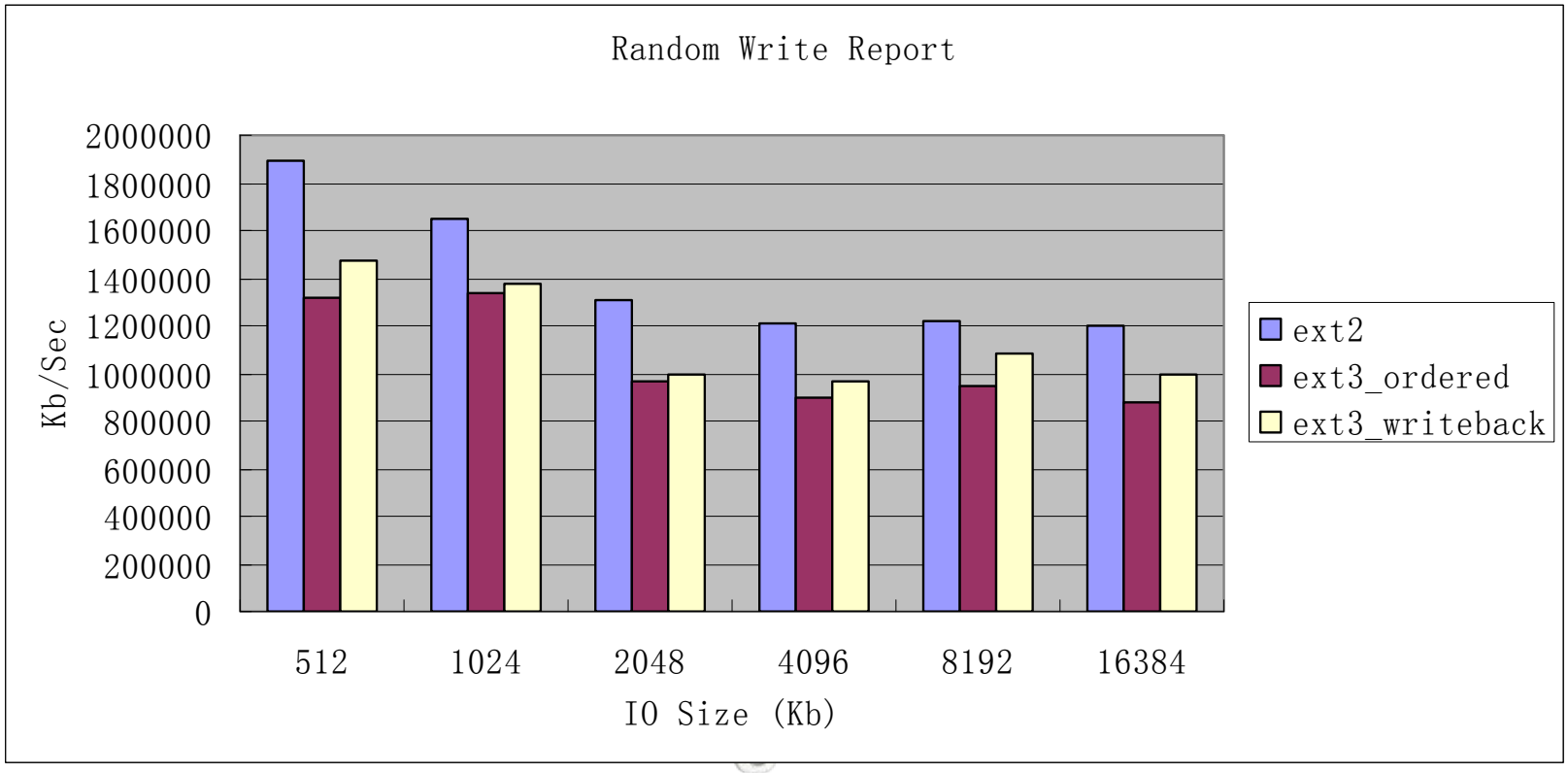
- -i #
  - Used to specify which tests to run
    - 0=write/rewrite
    - 1=read/re-read
    - 2=random-read/write
    - 3=Read-backwards
    - 4=Re-write-record
    - 5=stride-read
    - 6=fwrite/re-fwrite
    - 7=fread/Re-fread,
    - 8=random mix
    - 9=pwrite/Re-pwrite
    - 10=pread/Re-pread
    - 11=pwritev/Re-pwritev
    - 12=preadv/Repreadv
  - -i # -i # -i # is also supported so that one may select more than one test

- /test is the mount point
- iozone -a -s 128M -y 512 -q 16384 -i 0 -i 1 -i 2 -f /test/a.out -Rb /root/ext3_writeback.out






Random Read Report
3000000
2500000
2000000
1500000
1000000
500000
0
Kb/Sec
512
1024
2048
4096
8192
16384
IO Size(Kb)
ext2
ext3_ordered
ext3_writeback


Random Write Report
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Kb/Sec
512
1024
2048
4096
8192
16384
IO Size (Kb)
ext2
ext3_ordered
ext3_writeback

Most common special filesystems

Name	Mount point	Description
bdev	none	Block devices
binfmt_misc	any	Miscellaneous executable formats
devpts	*/dev/pts*	Pseudoterminal support (Open Group's Unix98 standard)
eventpollfs	none	Used by the efficient event polling mechanism
futexfs	none	Used by the futex (Fast Userspace Locking) mechanism
pipefs	none	Pipes
proc	*/proc*	General access point to kernel data structures
rootfs	none	Provides an empty root directory for the bootstrap phase
shm	none	IPC-shared memory regions
mqueue	any	Used to implement POSIX message queues
sockfs	none	Sockets
sysfs	*/sys*	General access point to system data
tmpfs	any	Temporary files (kept in RAM unless swapped)
usbfs	*/proc/bus/usb*	USB devices

马哥教育

Interrupts and Exceptions

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

Interrupts Overview

- An interrupt is usually defined as an event that alters the sequence of instructions executed by a processor
  - Interrupts enable hardware to signal to the processor
  - The processor receives the interrupt and signals the operating system to enable the operating system to respond to the new data
  - Events correspond to electrical signals generated by hardware circuits both inside and outside the CPU chip
- Interrupts are issued by interval timers and I/O devices; for instance, the arrival of a keystroke from a user sets off an interrupt

- Interrupts are often divided into synchronous and asynchronous interrupts
  - Synchronous interrupts are produced by the CPU control unit while executing instructions and are called synchronous because the control unit issues them only after terminating the execution of an instruction
  - Asynchronous interrupts are generated by other hardware devices at arbitrary times with respect to the CPU clock signals
- Intel microprocessor manuals designate synchronous and asynchronous interrupts as ***exceptions*** and ***interrupts***, respectively

- Exceptions, on the other hand, are caused either by *programming errors* or by *anomalous conditions* that must be handled by the kernel
  - In the first case, the kernel handles the exception by delivering to the current process one of the signals familiar to every Unix programmer
  - In the second case, the kernel performs all the steps needed to recover from the anomalous condition, such as a Page Fault or a requestvia an assembly language instruction such as int or sysenter for a kernel service

The Role of Interrupt Signals

- When an interrupt signal arrives, the CPU must stop what it's currently doing and switch to a new activity; it does this by saving the current value of the program counter (i.e., the content of the eip and cs registers) in the Kernel Mode stack and by placing an address related to the interrupt type into the program counter
- But there is a key difference between interrupt handling and process switching: the code executed by an interrupt or by an exception handler is not a process
  - It is a kernel control path that runs at the expense of the same process that was running when the interrupt occurred
  - A kernel control path, the interrupt handler is lighter than a process

- Interrupt handling is one of the most sensitive tasks performed by the kernel, because it must satisfy the following constraints:
  - Interrupts can come anytime, when the kernel may want to finish something else it was trying to do
    - The kernel's goal is therefore to get the interrupt out of the way as soon as possible and defer as much processing as it can
  - Because interrupts can come anytime, the kernel might be handling one of them while another one (of a different type) occurs
  - Although the kernel may accept a new interrupt signal while handling a previous one, some critical regions exist inside the kernel code where interrupts must be disabled

IRQs and Interrupts

- Each hardware device controller capable of issuing interrupt requests usually has a single output line designated as the Interrupt ReQuest (IRQ) line
- All existing IRQ lines are connected to the input pins of a hardware circuit called the *Programmable Interrupt Controller*
  - The IRQ lines are sequentially numbered starting from 0; therefore, the first IRQ line is usually denoted as IRQ 0
  - Each IRQ line can be selectively disabled

Interrupt Handlers

- The function the kernel runs in response to a specific interrupt is called an interrupt handler or interrupt service routine (ISR)
  - Each device that generates interrupts has an associated interrupt handler
  - They run in a special context called *interrupt context*
  - Because an interrupt can occur at any time, an interrupt handler can, in turn, be executed at any time

Top Halves Versus Bottom Halves

- These two goals—that an interrupt handler execute quickly and perform a large amount of work—clearly conflict with one another
- Because of these competing goals, the processing of interrupts is split into two parts, or halves
  - The interrupt handler is the *top half*
  - Work that can be performed later is deferred until the bottom half

/proc/interrupts

- A proc file populated with statistics related to interrupts on the system

```
[root@instructor ~]# cat /proc/interrupts
           CPU0
  0:        212   IO-APIC-edge      timer
  1:          2   IO-APIC-edge      i8042
  4:         17   IO-APIC-edge
  7:          0   IO-APIC-edge      parport0
  8:          1   IO-APIC-edge      rtc0
  9:          0   IO-APIC-fasteoi   acpi
 12:          7   IO-APIC-edge      i8042
 14:     330233   IO-APIC-edge      ide0
```

  - The first column is the interrupt line
  - The second column is a counter of the number of interrupts received
  - The third column is the interrupt controller handling this interrupt
  - The last column is the device associated with this interrupt

马哥教育

Network subsystem

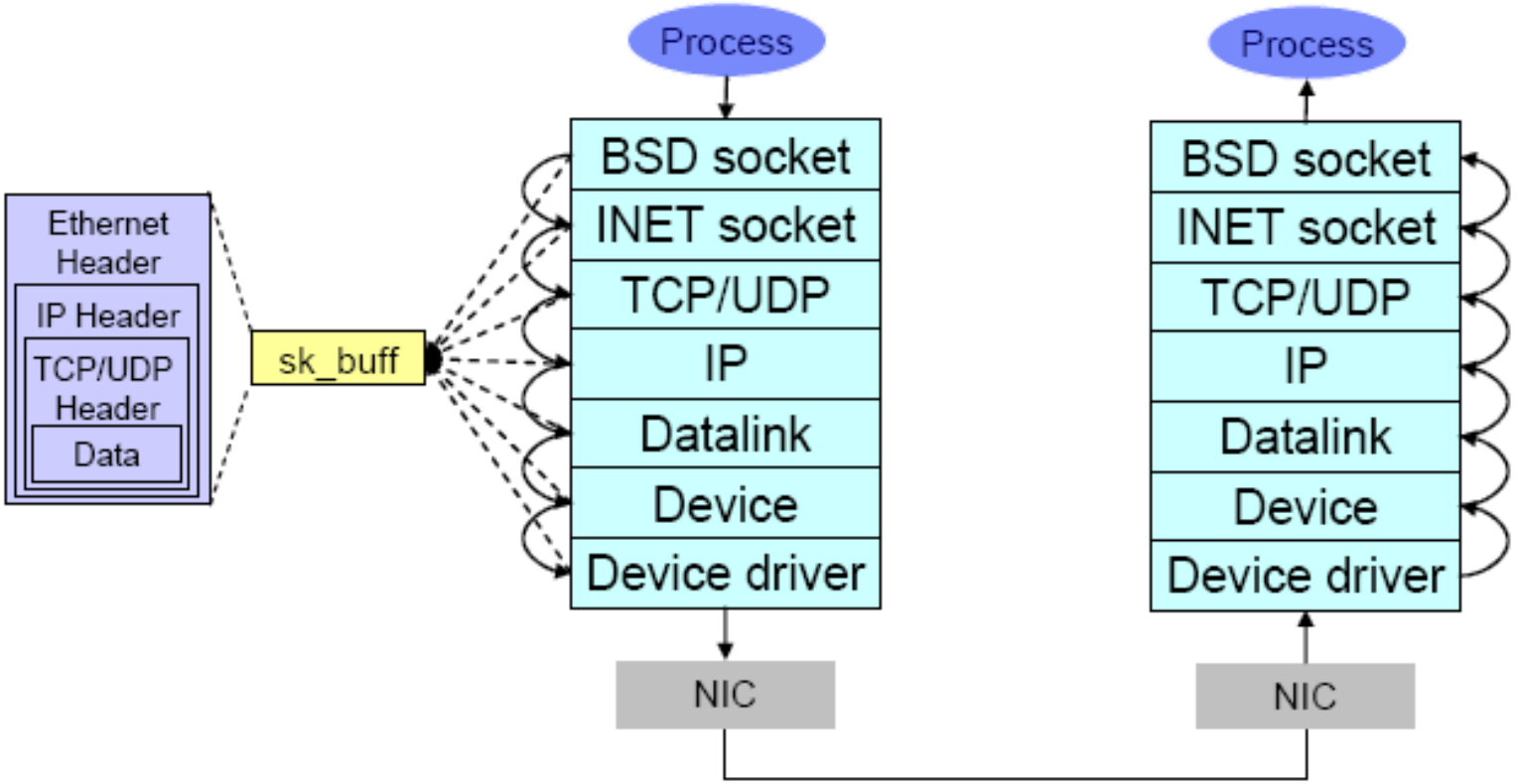
主讲：马永亮(马哥)
QQ：113228115,1661815153
http://www.magedu.com

Overview

- The network subsystem is another important subsystem in the performance perspective
- Networking operations interact with many components other than Linux such as switches, routers, gateways, PC clients, and so on
- Though these components might be out of the control of Linux, they have a lot of influence on the overall performance

Networking implementation

- Network layered structure and overview of networking operation



Network sockets

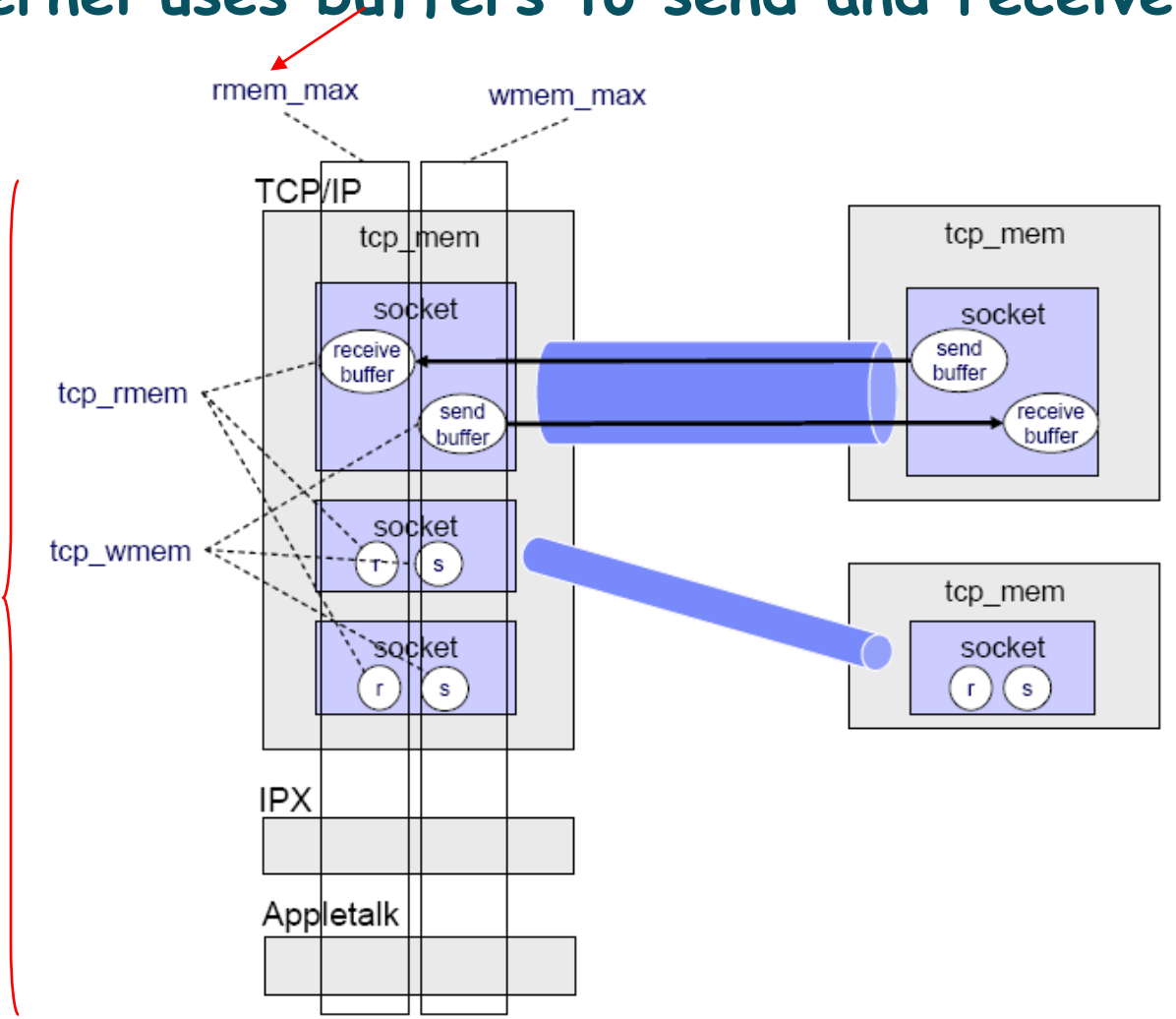
- Socket ties application to network stack for reading and writing
- Each socket (network connection) is treated like a virtal file
  - Write to the file to transmit data
  - Read from the file to receive data
  - Close the socket to destroy the file
- Read and write socket buffer store data for application
- TCP sockets require extra connection handling

- The sequence that outlines the fundamental operations that occur during network data transfer
  - When an application sends data to its peer host, the application creates its data
  - The application opens the socket and writes the data through the socket interface
  - The *socket buffer* is used to deal with the transferred data
    - The socket buffer has reference to the data and it goes down through the layers
  - In each layer, appropriate operations such as parsing the headers, adding and modifying the headers, check sums, routing operation, fragmentation, and so on are performed

- Finally, the data goes out to the wire from the network interface card
- The Ethernet frame arrives at the network interface of the peer host
- The frame is moved into the network interface card buffer if the MAC address matches the MAC address of the interface card
- The network interface card eventually moves the packet into a socket buffer and issues a hard interrupt at the CPU
- The CPU then processes the packet and moves it up through the layers until it arrives at (for example) a TCP port of an application such as Apache

Socket buffer

- The kernel uses buffers to send and receive data



Simplified transmit model

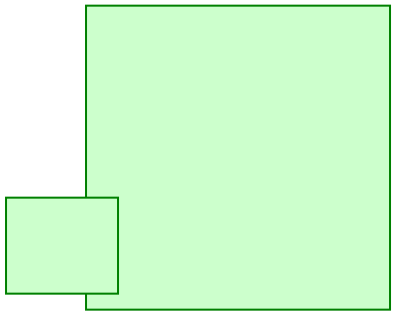
- Output/writer
  - * Write data to socket "file" (goes in transmit buffer)
  - Kernel encapsulates data into protocol data units
  - * PDUs are moved to per-device transmit queue
  - Driver copies PDU from head of queue to NIC
  - * NIC raises interrupt when PDU transmitted

* essential tuning points

Simplified receive model

- Input/reader
  - * NIC receives frame and uses DMA to copy into receive buffer
  - NIC raises CPU interrupt
  - Kernel handles interrupt and schedules a softirq
  - Softirq is handled to move packets up to the IP layer for routing decision
  - If local delivery
    - Decapsulate packet and place into socket's receive buffer
    - Wake the processes in the socket's wait queue
    - Process reads from socket (receive buffer)

* essential tuning points



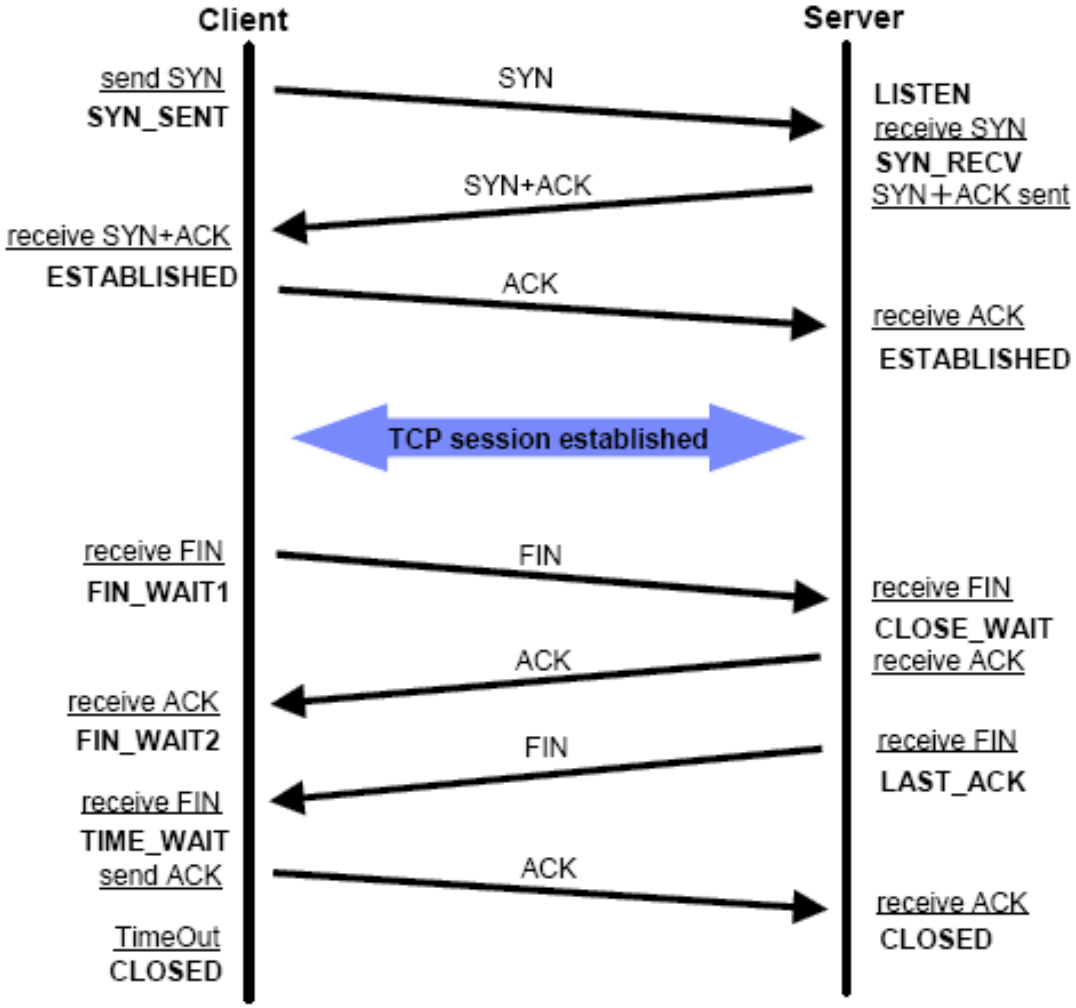
Network API (NAPI)

- In the traditional approach of handling network packets, the network interface card eventually moves the packet into a network buffer of the operating systems kernel and issues a hard interrupt at the CPU
  - Every time an Ethernet frame with a matching MAC address arrives at the interface, there will be a hard interrupt
- NAPI was introduced to counter the overhead associated with processing network traffic
  - For the first packet, NAPI works just like the traditional implementation as it issues an interrupt for the first packet
  - But after the first packet, the interface goes into a polling mode

TCP sockets

- Open with three-way hankshake
  - Client sends syn packet
  - Server replies with syn-ack packet
  - Client sends ack packet
- Systems terminate socket independently
  - One side sends fin packets
  - Remote side replies with fin-ack packet
  - Idle connections may be considered dead
  - Requires the use of keepalives
  - Half-closed connection terminated after default timeout if FIN-ACK not received

TCP state


Client
Server
send SYN
SYN_SENT
SYN
LISTEN
receive SYN
SYN_RECV
SYN+ACK sent
SYN+ACK
receive SYN+ACK
ESTABLISHED
ACK
receive ACK
ESTABLISHED
TCP session established
receive FIN
FIN_WAIT1
FIN
receive FIN
CLOSE_WAIT
ACK
receive ACK
receive ACK
FIN_WAIT2
FIN
receive FIN
LAST_ACK
receive FIN
TIME_WAIT
send ACK
ACK
receive ACK
CLOSED
TimeOut
CLOSED

Viewing network sockets

- Passive opens (listeners)

  netstat -tunlp

- Active sockets

  sar -n SOCK

  lsof -l

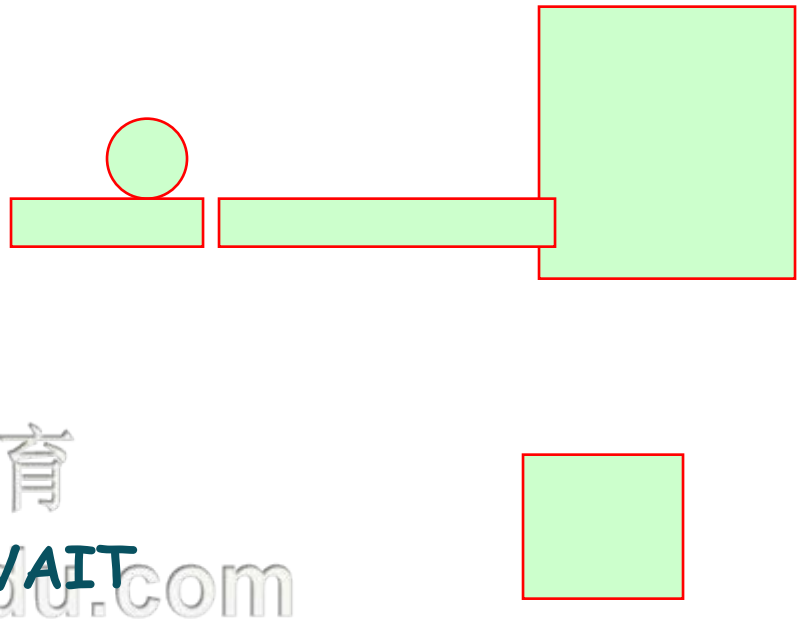
  netstat -tu

- All sockets

  netstat -taupe

- Half-closed connections

  netstat -tapn | grep TIME_WAIT



Tuning TCP socket creation

- Number of times initial SYNs for an active TCP connection attempt will be retransmitted

  net.ipv4.tcp_sync_retries
  - Should not be higher than 255
  - The default value is 5
- Listener queue length for unacknowledged connection attempts

  net.ipv4.tcp_max_syn_backlog
  - Maximum number of remembered connection requests, which are still did not receive an acknowledgment from connecting client. default 1024
- Reusing TCP connections

  net.ipv4.tcp_tw_recycle

Tuning TCP socket keepalive

- Wait time before sending first keepalive

  net.ipv4.tcp_keepalive_time
  - default: 2hours
- How often to send keepalive probes

  net.ipv4.tcp_keepalive_intvl
  - default: 75secs
- How many probes to send

  net.ipv4.tcp_keepalive_probes
  - default: 9
  - If no ACK response is received for nine consecutive times, the connection is marked as broken

Kernel socket buffers

- Kernel buffers
  - Core read and write buffers apply to non-TCP protocols
  - TCP window size
  - Fragmentation for receive/read
  - DMA for NIC receive
- Kernel adjusts size of buffers automatically
  - Buffers require non-pageable memory
  - Put pressure on ZONE_NORMAL
  - Receiver controls flow based on receive buffer size

Improving buffer sizes

- Transmit continuously to maximize throughput
- Bandwidth delay product (BDP)
  - In data communications, **bandwidth-delay product** refers to the product of a data link's capacity (in bits per second) and its end-to-end delay (in seconds)
    - The result, an amount of data measured in bits (or bytes), is equivalent to the maximum amount of data on the network circuit at any given time, i.e. data that has been transmitted but not yet received
    - Sometimes it is calculated as the data link's capacity times its round trip time (RTT)

Improving buffer sizes

- Computing the BDP
  - Total buffers = Bandwidth*Delay
  - Examples
    - Residential DSL: 2 Mb/s, 50 ms RTT
      - $B \times D = 2 \times 10^6$ b/s $\times$ 50×10^{-3} s = 100×10^3 b, or 100 kb / 12.5 kB
    - High-speed terrestrial network: 1 Gb/s, 1 ms RTT
      - $B \times D = 10^9$ b/s $\times$ 10^{-3} s = 10^6 b, or 1 Mb / 125 kB

Tuning core buffer size

- 1. Update /etc/sysctl.conf for BDP/#connec-tions
  - Input/reader in Bytes
    net.core.rmem_default
    net.core.rmem_max
  - Output/writer in Bytes
    net.core.wmem_default
    net.core.wmem_max
- 2. Reload /etc/sysctl.conf

  sysctl -p
- 3. Restart affected services

TCP window size

- TCP is a sliding window, not stop-n-wait protocol
  - TCP window scaling is enabled by default
  - Initial segment size is negotiated during socket creation
  - Must resend data not ACKed by receiver
- Must limit segment size based on
  - Latest rwnd sent by the receiver
  - Amount of unACKed data already transmitted

tcp_window_scaling=1

Window size

- In computer networking, RWIN (TCP Receive Window) is the amount of data that a computer can accept without acknowledging the sender
  - If the sender has not received acknowledgement for the first packet it sent, it will stop and wait and if this wait exceeds a certain limit, it may even retransmit
  - Even if there is no packet loss in the network, windowing can limit throughput
  - Throughput ≦ RWIN/RTT

Tuning TCP window size

- 1. Tune core buffers for BDP/#connections
- 2. Update /etc/sysctl.conf for BDP/#connec-tions
  - Overall TCP memory in pages

    net.ipv4.tcp_mem

    min defalut max
  - Input/reader in Bytes

    net.ipv4.tcp_rmem
  - Output/writer in Bytes

    net.ipv4.tcp_wmem
- 3. Reload /etc/sysctl.conf
- 4. Restart affected services

Tuning fragmentation buffers

- Maximum memory used to reassemble IP fragments
  net.ipv4.ipfrag_high_thresh
- Minimum memory used to reassemble IP fragments
  net.ipv4.ipfrag_low_thresh
- Time in seconds to keep an IP fragment in memory
  net.ipv4.ipfrag_time
  - Expiration time for fragments
  - The default value is 30 seconds

Viewing fragmentation statistics

- Summary statistics

  netstat -s

- View reassembly failures

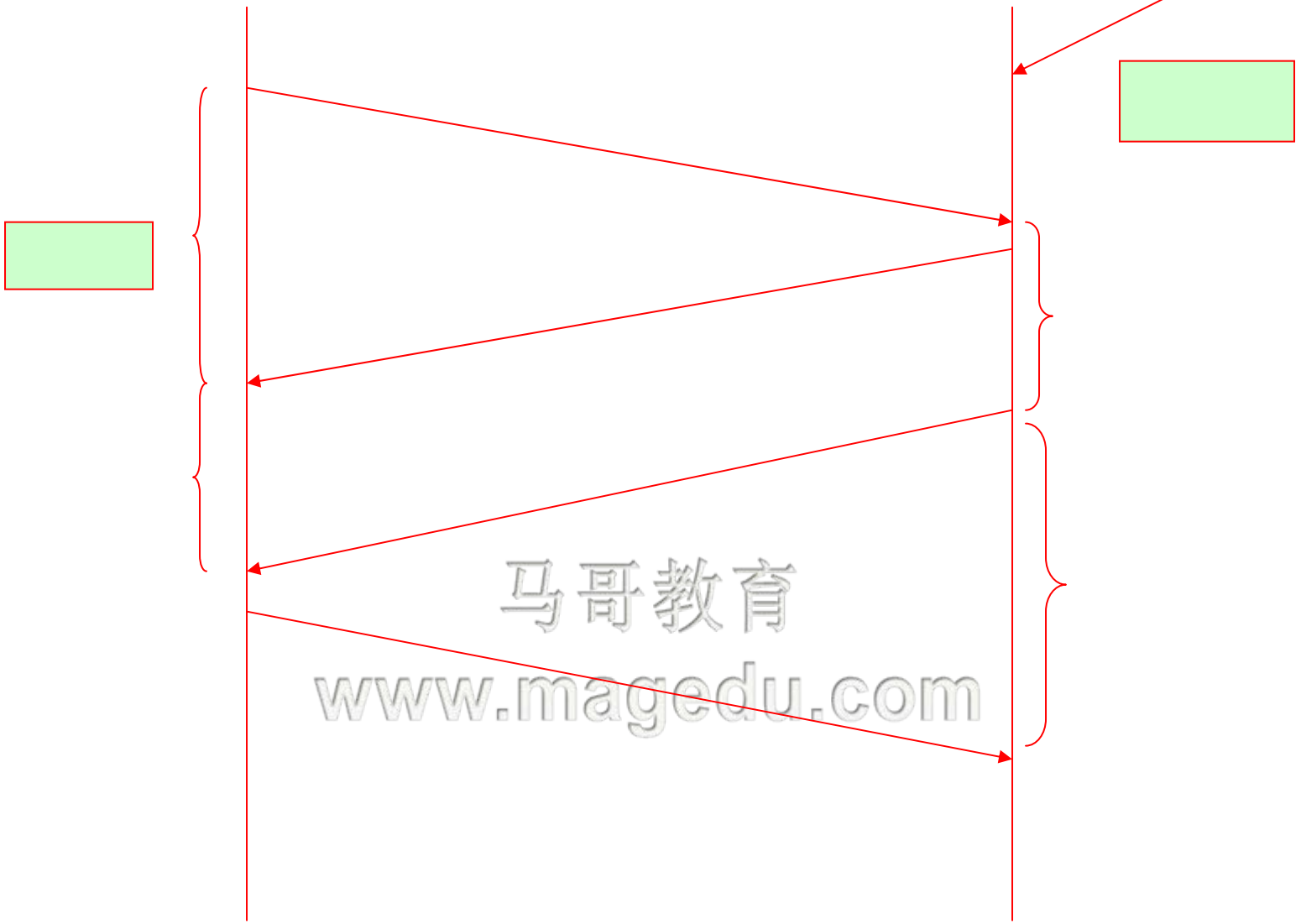
  cat /proc/net/snmp | grep '^Ip:' | cut -f17 -d' '

  - Reassembly failures indicate a need to tune buffer

- Causes of fragmentation
  - Denial of Service (DoS) attacks
  - NFS
  - Noisy networks
  - Failing network electronics

Other kernel parameters

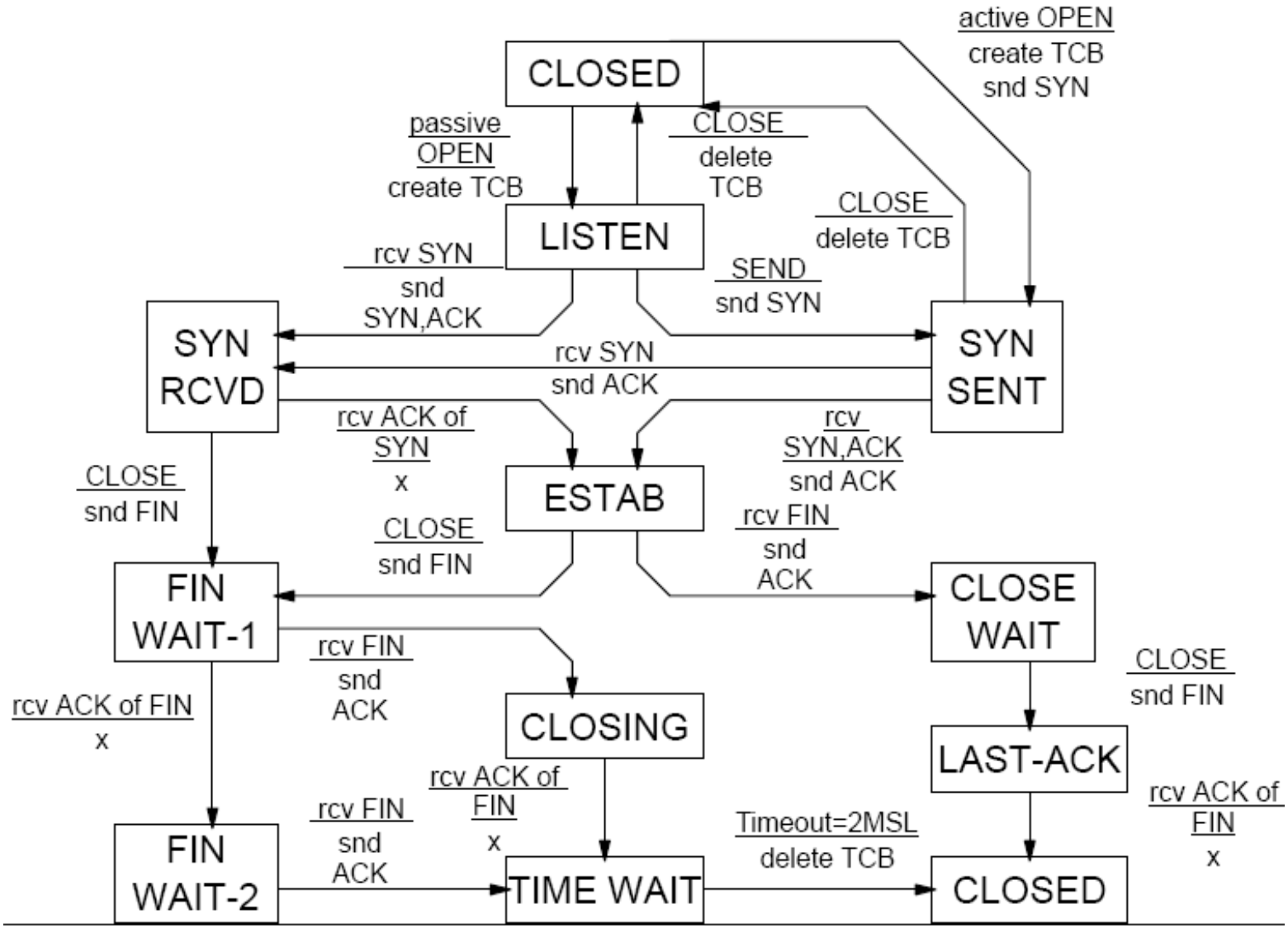
- net.ipv4.tcp_tw_reuse
  - Allow to reuse TIME-WAIT sockets for new connections when it is safe from protocol viewpoint
- tcp_syncookies
  - Send out syncookies when the syn backlog queue of a socket overflows
  - This is to prevent against the common "syn flood attack"
- tcp_max_orphans
  - Maximal number of TCP sockets not attached to any user file handle, held by system
  - This limit exists only to prevent simple DoS attacks
  - Must not rely on this or lower the limit artificially, but rather increase it



- tcp_fin_timeout
  - Time to hold socket in state FIN-WAIT-2, if it was closed by our side
  - Peer can be broken and never close its side, or even die unexpectedly
  - FIN-WAIT-2 sockets are less dangerous than FIN-WAIT-1, because they eat maximum 1.5 kilobytes of memory, but they tend to live longer
- tcp_max_tw_buckets
  - Maximal number of timewait sockets held by system simultaneously
  - This limit exists only to prevent simple DoS attacks, you must not lower the limit artificially, but rather increase it

- net.core.somaxconn
  - Limit of socket listen() backlog, known in userspace as SOMAXCONN
- netdev_max_backlog
  - Maximum number of packets, queued on the input side, when the interface receives packets faster than kernel can process them
  - Applies to non-NAPI devices only
  - The default value is 1000

TCP connection state diagram

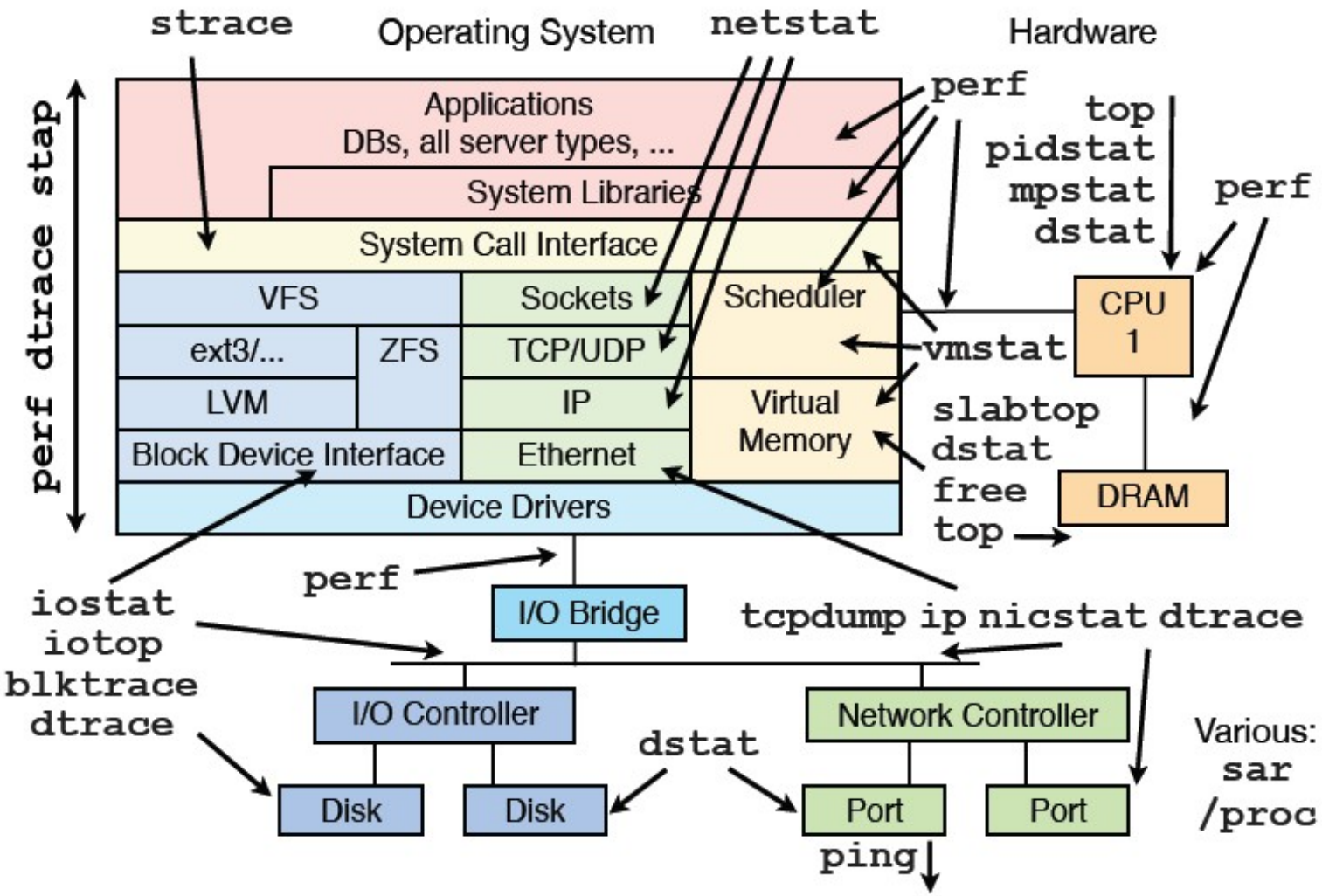


马哥教育

Understanding Linux performance metrics

主讲：马永亮(马哥)
QQ:113228115
客服QQ：2813150558，1661815153
http://www.magedu.com
http://mageedu.blog.51cto.com

Linux常用性能调优工具索引



Processor metrics

- CPU utilization
  - This is probably the most straightforward metric
  - It describes the overall utilization per processor
- User time
  - Depicts the CPU percentage spent on user processes, including nice time
  - High values in user time are generally desirable because, in this case, the system performs actual work
- System time
  - Depicts the CPU percentage spent on kernel operations including IRQ and softirq time
  - High and sustained system time values can point you to bottlenecks in the network and driver stack

- Waiting
  - Total amount of CPU time spent waiting for an I/O operation to occur
  - Like the *blocked* value, a system should not spend too much time waiting for I/O operations; otherwise youshould investigate the performance of the respective I/O subsystem
- Idle time
  - Depicts the CPU percentage the system was idle waiting for tasks
- Nice time
  - Depicts the CPU percentage spent on re-nicing processes that change the execution order and priority of processes

- Load average
  - The load average is not a percentage, but the rolling average of the sum of the following:
    - The number of processes in queue waiting to be processed
    - The number of processes waiting for uninterruptable task to be completed
  - That is, the average of the sum of TASK_RUNNING and TASK_UNINTERRUPTIBLE processes
- Runable processes
  - This value depicts the processes that are ready to be executed
  - This value should not exceed 10 times the amount of physical processors for a sustained period of time; otherwise a processor bottleneck is likely

- Blocked
  - Processes that cannot execute while they are waiting for an I/O operation to finish
  - Blocked processes can point you toward an I/O bottleneck
- Context switch
  - Amount of switches between threads that occur on the system
  - High numbers of context switches in connection with a large number of interrupts can signal driver or application issues
- Interrupts
  - Hard interrupts have a more adverse effect on system performance
  - High interrupt values are an indication of a software bottleneck, either in the kernel or a driver

Memory metrics

- Free memory
- Swap usage
  - This value depicts the amount of swap space used
  - Swap In/Out is a reliable means of identifying a memory bottleneck
  - Values above 200 to 300 pages per second for a sustained period of time express a likely memory bottleneck
- Buffer and cache
  - Cache allocated as file system and block device cache
- Slabs
  - Depicts the kernel usage of memory
- Active versus inactive memory
  - Inactive memory is a likely candidate to be swapped out to disk by the kswapd daemon

Network interface metrics

- Packets received and sent
- Bytes received and sent
- Collisions per second
  - This value provides an indication of the number of collisions that occur on the network that the respective interface is connected to
  - Sustained values of collisions often concern a bottleneck in the network infrastructure, not the server
  - On most properly configured networks, collisions are very rare unless the network infrastructure consists of hubs

- Packets dropped
  - This is a count of packets that have been dropped by the kernel, either due to a firewall configuration or due to a lack of network buffers
- Overruns
  - Represent the number of times that the network interface ran out of buffer space
  - This metric should be used in conjunction with the *packets dropped* value to identify a possible bottleneck in network buffers or the network queue length
- Errors
  - The number of frames marked as faulty
  - This is often caused by a network mismatch or a partially broken network cable
  - Partially broken network cables can be a significant performance issue for copper-based gigabit networks

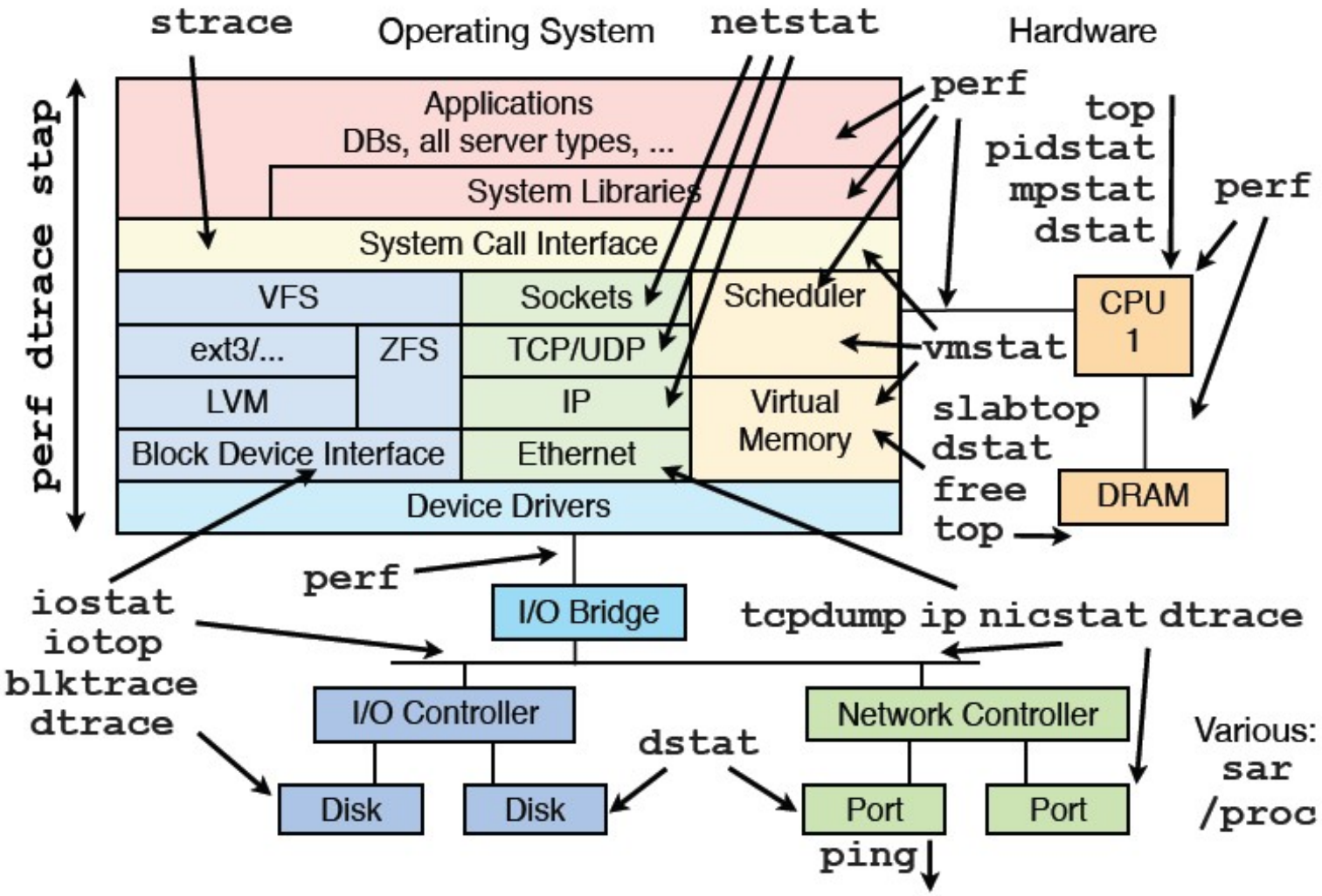
Block device metrics

- Iowait
  - Time the CPU spends waiting for an I/O operation to occur
  - High and sustained values most likely indicate an I/O bottleneck
- Average queue length
  - Amount of outstanding I/O requests
  - In general, a disk queue of 2 to 3 is optimal; higher values might point toward a disk I/O bottleneck
- Average wait
  - A measurement of the average time in ms it takes for an I/O request to be serviced
  - The wait time consists of the actual I/O operation and the time it waited in the I/O queue

- Transfers per second
  - Depicts how many I/O operations per second are performed (reads and writes)
  - The *transfers per second* metric in conjunction with the *kBytes per second* value helps you to identify the average transfer size of the system
  - The average transfer size generally should match with the stripe size used by your disk subsystem
- Blocks read/write per second
  - This metric depicts the reads and writes per second expressed in blocks of 1024 bytes as of kernel 2.6
  - Earlier kernels may report different block sizes, from 512 bytes to 4 KB

- Kilobytes per second read/write
  - Reads and writes from/to the block device in kilobytes represent the amount of actual data transferred to and from the block device

Linux常用性能调优工具索引



关于马哥教育

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：2813150558，1661815153，113228115
- QQ群：203585050，279599283

马哥教育

Thank You!

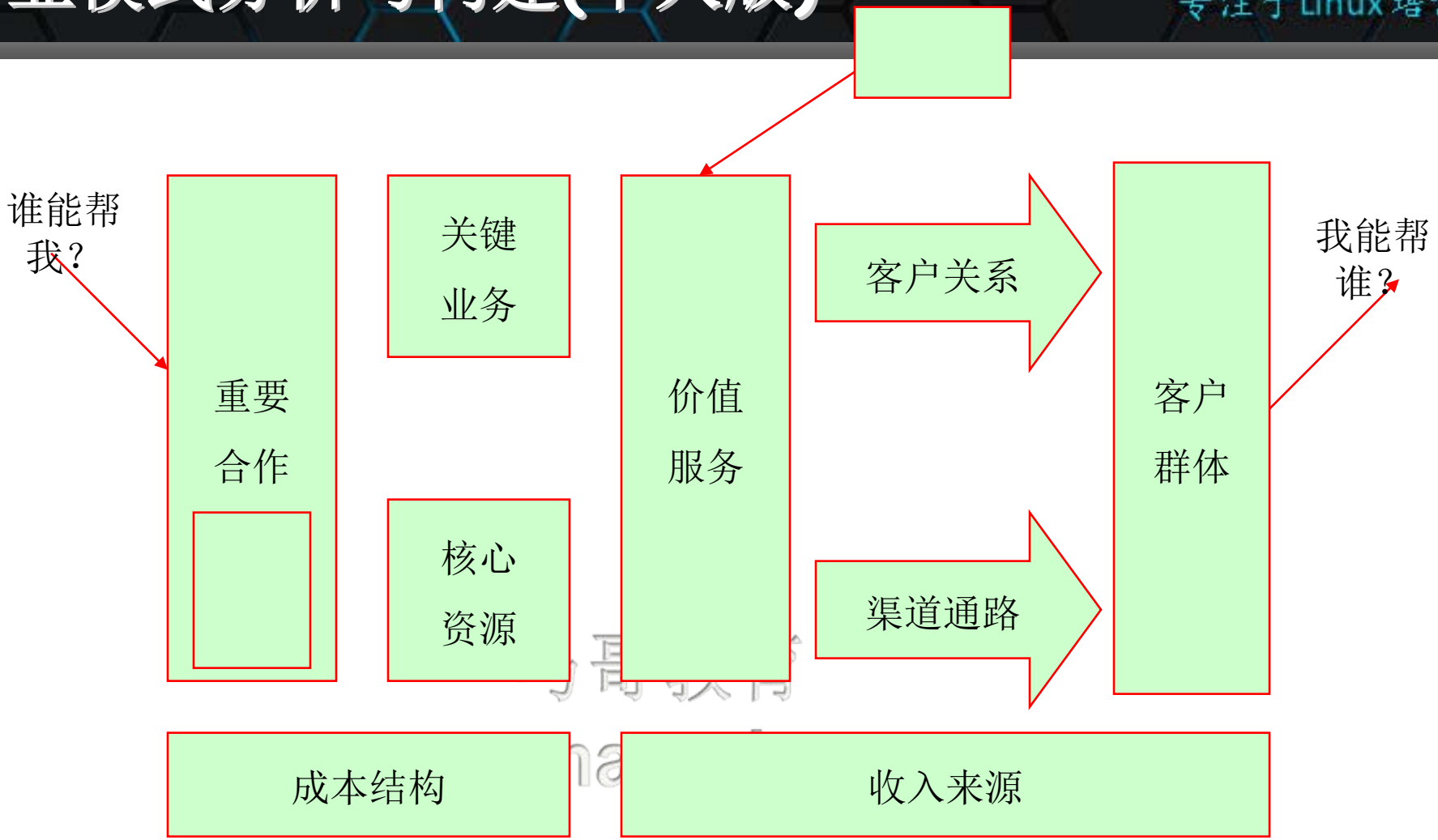
马哥教育

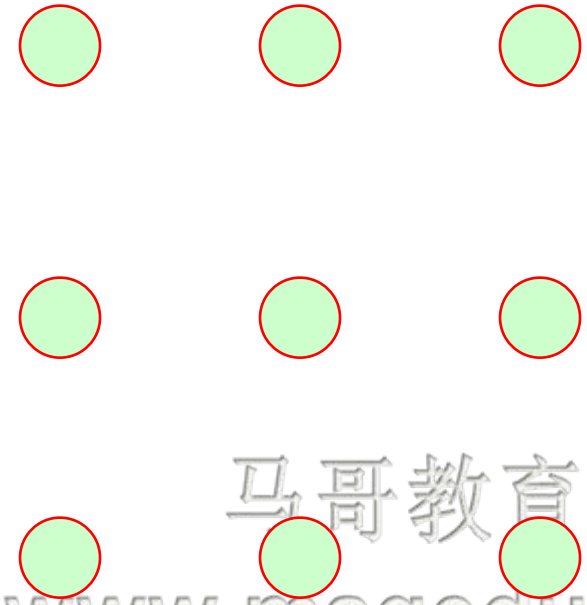
组织商业模式

组织机构向客户提供价值？

- 商业模式的构成要素
  - 客户群体(服务对象)
  - 价值服务(为客户解决的问题或满足的需求)
  - 渠道通路(沟通和交付价值的不同方式)
  - 客户关系(跟客户建立和维持的不同关系)
  - 收入来源(客户为获取到价值所支付的钱)
  - 核心资源(创建和交付前述的价值服务所依赖的资产)
  - 关键业务(创建和交付前述的价值服务所做的工作)
  - 重要合作(从组织外部获取的资源和支持)
  - 成本结构(获取核心资源、关键业务以及外资源时所支付的费用)

商业模式分析与构建(个人版)





- A，艺术型
- S，社交型
- I，学者型
- E，事业型
- R，现实型
- C，传统型