UNIX INTERNALS

Ms. Radha Senthilkumar, Lecturer
Department of IT
MIT, Chromepet
Anna University, Chennai.

MEMORY MANAGEMENT POLICIES

Table of Contents

- SWAPPING
- DEMAND PAGING

SWAPPING

- UNIX systems transferred entire processes between primary memory and the swap device
- Easier to implement
- Less system overhead
- **Three parts to the description of swapping algorithm**
  - **Managing space on the swap device**
  - **Swapping process out of main memory**
  - **Swapping process into main memory**

Allocation of Swap Space

	Swap device	kernel
Allocate For file	Group of Contiguous blocks	One block at a time
Data Structure Catalog Free space	Map(in core table)	Linked List of free blocks

- **MAP is array where each consists of an address of an allocable Resource and the number of resource units.**
- **Initially a map contains one entry that indicates the address and the total number of space.**

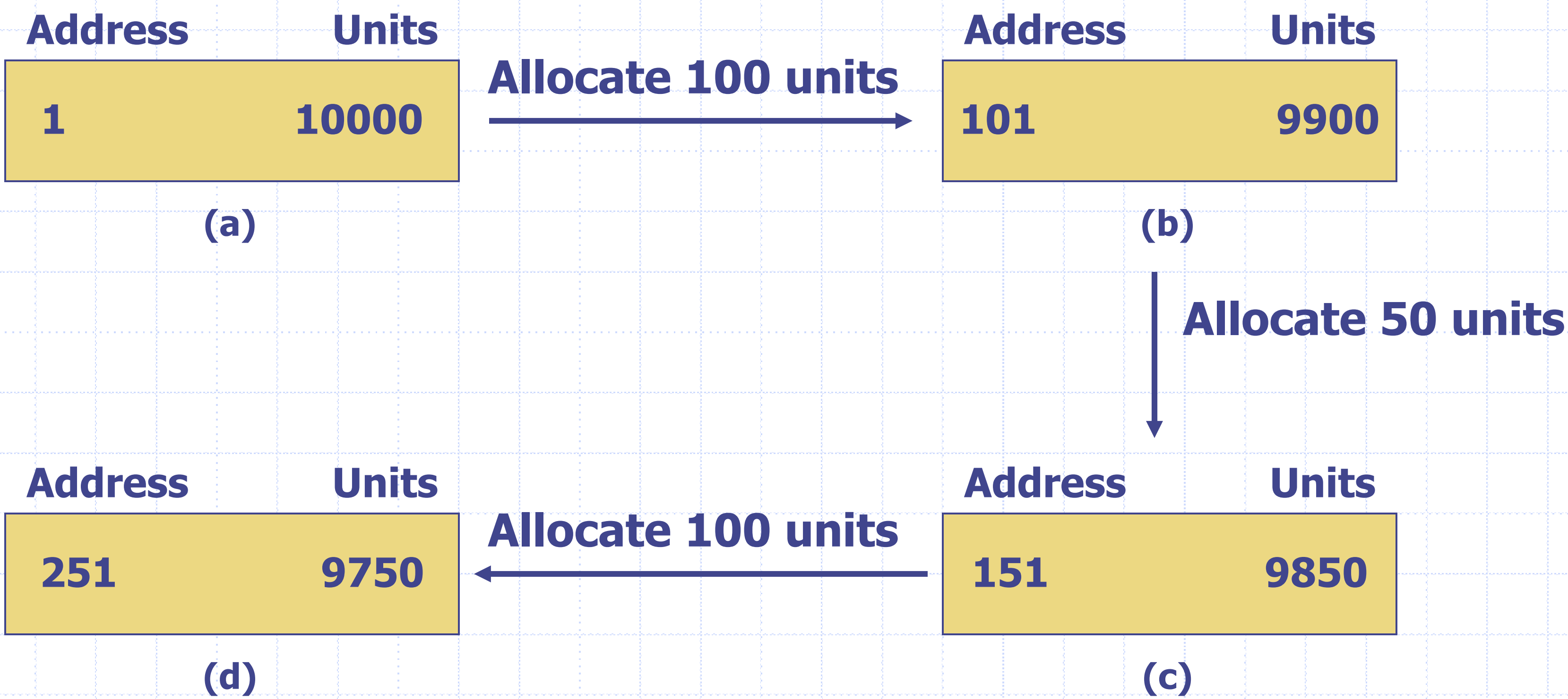
Address	Units
1	10000

Initial Swap Map

Algorithm for Allocating Space from Maps

```
algorithm malloc /*algorithm to allocate map space */
input: (1) map address    /indicates which map to use */
       (2)requested number 0f units
output: address , if successful 0, otherwise
{
    for(every map entry)
    {
        if(current map entry can fit requested units)
        {
            if(requested units == number of units in entry)
                delete entry from map;
            else
                adjust start address of entry;
            return (original address of entry);
        }
    }
    return 0;
}
```

Allocating Swap Space



How to find proper position when freeing resource

- **Completely fill a hole in the map:**
  - **Combines the newly freed resources and the existing (two) entries into in the map.**
- **Partially fill a hole in the map:**
  - **If precede or contiguous with the map entry(but not both) that adjusts the address and its fields.**
- **Partially fill a hole but are not contiguous to any resources in the map:**
  - **Creates a new entry for the map and inserts it in the proper position.**

Freeing Swap Space

Address	Units
251	9750

(a)

50 units free at 101 →

Address	Units
101	50
251	9750

(b)

100 units free at 1 →

Address	Units
1	150
251	9750

(c)

Allocating Swap Space from second Entry in the Map

Address	Units
1	150
251	9750

(a)

free 200 Units →

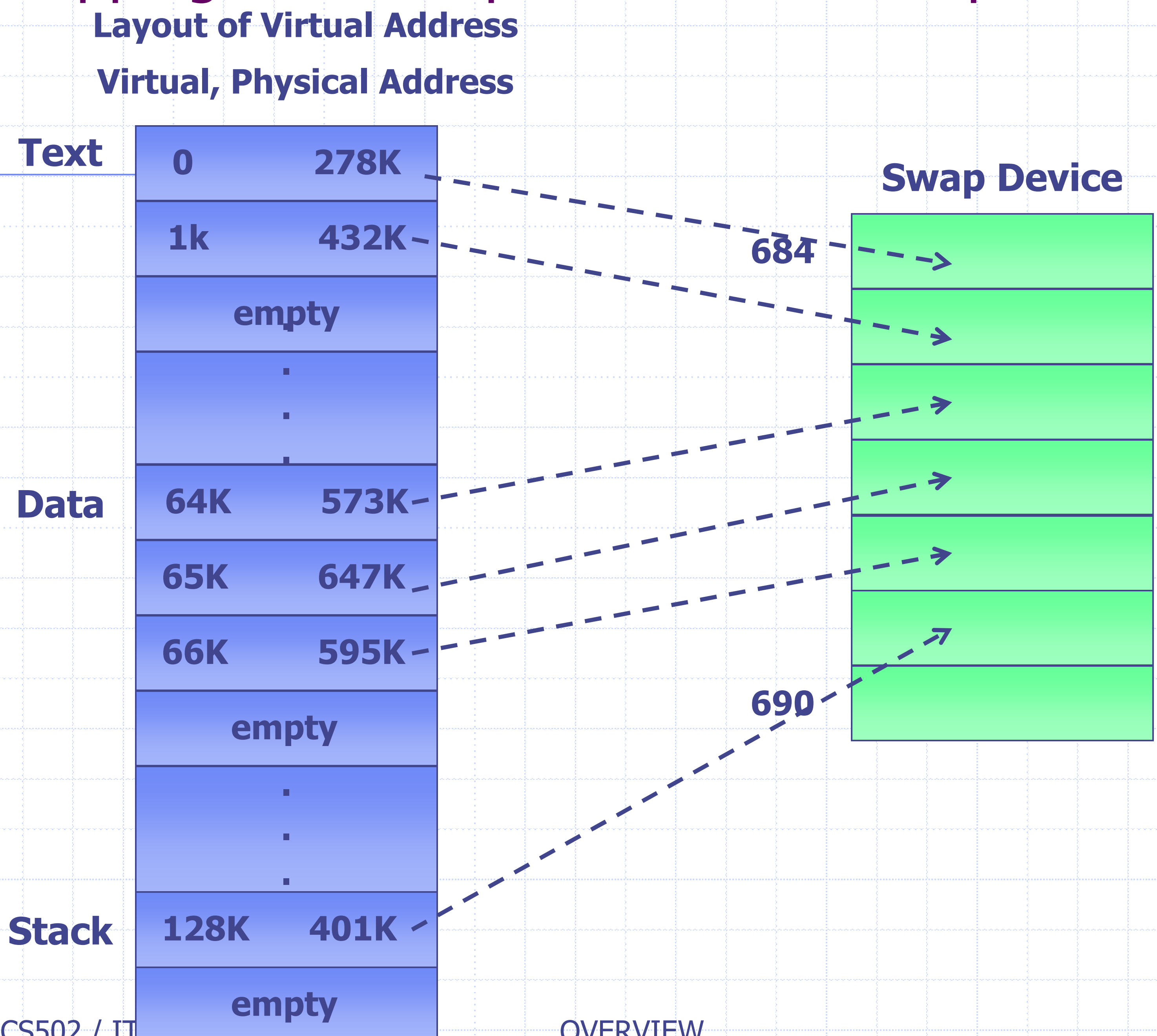
Address	Units
101	150
451	9550

(b)

Swapping process Out

- The kernel swaps a process out if it needs space in memory, which may result from any of the following:
  - The fork system call must be allocate space for a child process,
  - The brk system call increases the size of a process,
  - A Process becomes larger by the natural growth of its stack,
  - The kernel wants to free space in memory for process it hand previously swapped out should now swap in.
- process is eligible for swapping from memory, decrements the reference count of each region in the process and swaps the region out if its reference 0. And lock process in memory.

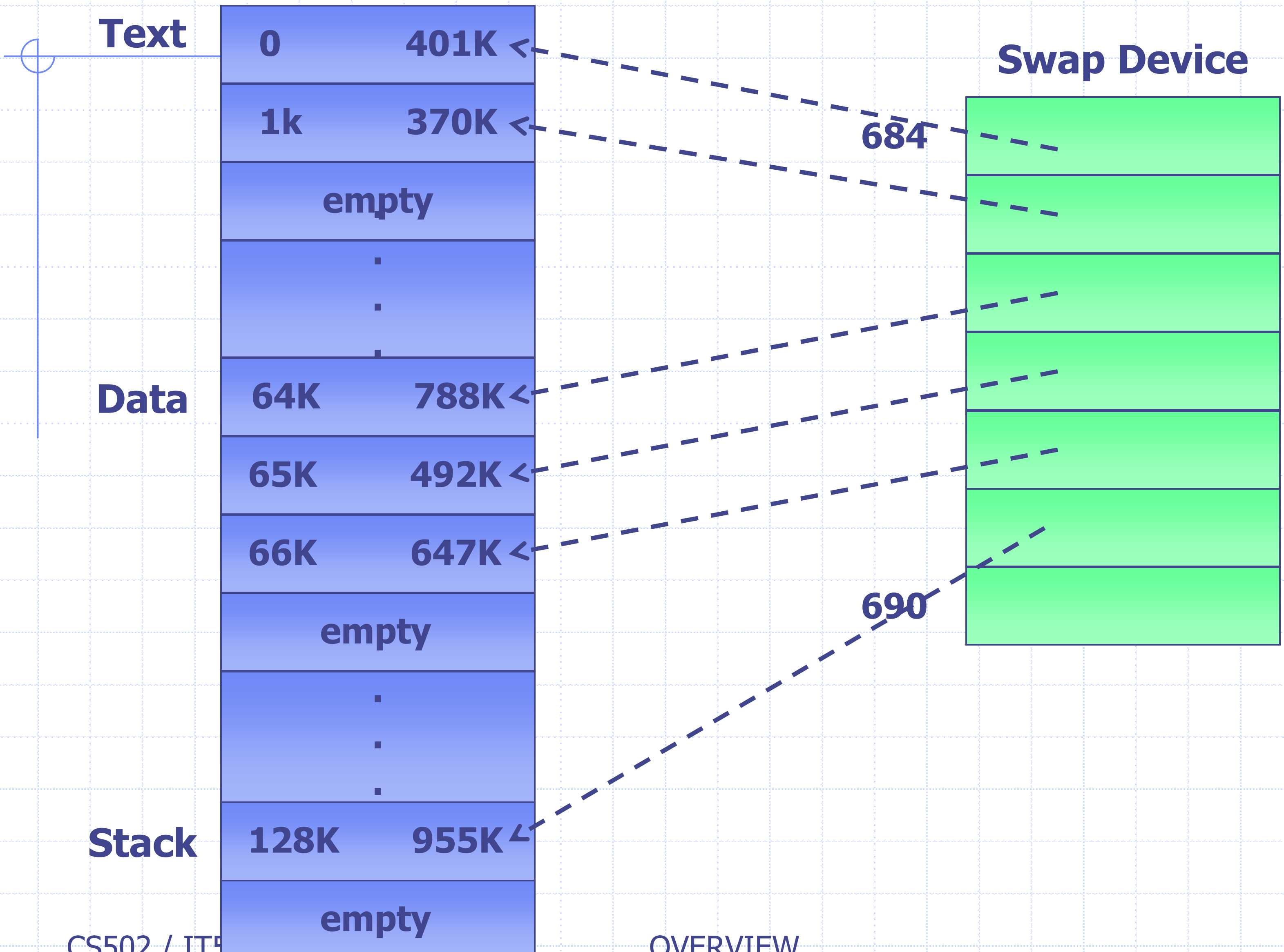
Mapping Process Space onto the Swap Device



Swapping a Process into memory

Layout of Virtual Address

Virtual, Physical Address



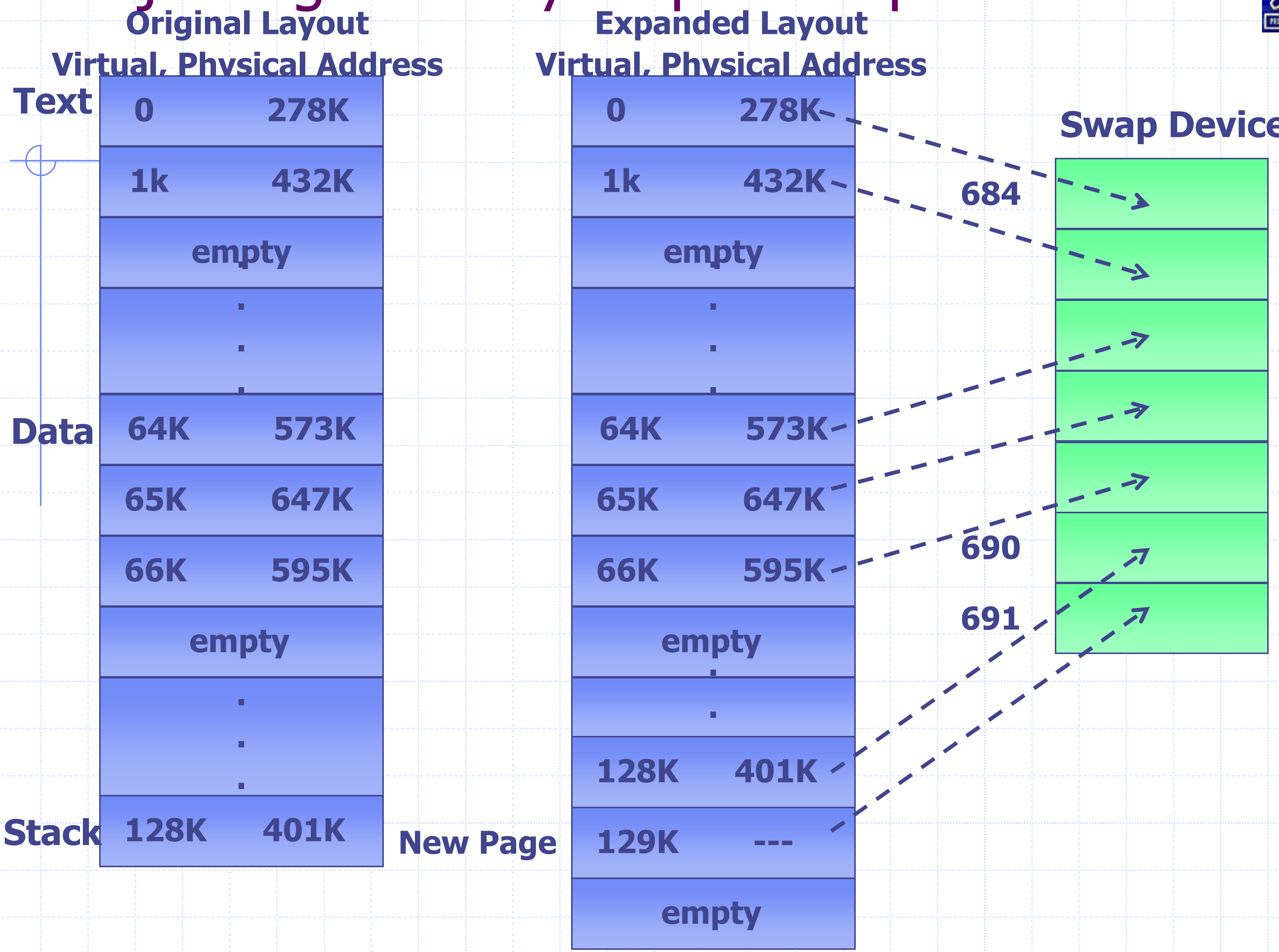
Fork Swap

- There is not enough memory when fork() called.
- Swaps the process out without freeing the memory occupied by the in-core copy.
- When the swap is complete, the child process exists on the swap device.
- parent places the child in the "ready-to-run" state and returns to user mode.
- Swap in when kernel schedule it

Expansion Swap

- When requires more physical memory than is currently allocated it.(stack growth or brk system call)
- Reserve enough space on swap device.
- Adjust the address translation mapping but, Not allocate.
- Finally swap the process out in normal swapping operation, zeroing out the newly allocated space on the swap device.

Adjusting Memory Map for Expansion



Swapping Process in

- Process 0, the swapper is the only process that swaps process into memory from swap device.
- At the conclusion of system initialization, swapper goes into an infinite loop.
- Swapper swaps process in or out ,unable to work sleeps.
- Kernel periodically wakes swapper up.

Algorithm for the swapper

```
algorithm swapper    /*swap in swapped out process, * swap out other process to make room */
input: none
output: none
{
   loop:
        for(all swapped out process that are ready to run)
                  pick process swapped out longest;
        if (no such process)
        {
                sleep(event must swap in);
                goto loop;
        }
         if (enough room in main memory for process)
        {
                swap process in;
                goto loop;
        }
   /* loop2: here in revised algorithm (see page 285) */
        for (all process loaded in main memory, not zombie and not locked in memory)
        {
                if (there is a sleeping process)
                         choose process such that priority + residence time is numerically highest;
                else /*no sleeping processes */
                         choose process such that residence time + nice is numerically highest;
        }
        if (chosen process not sleeping or residency requirements not satisfied)
               sleep (event must swap process in);
       else
               swap out process;
      goto loop;                                  /* goto loop2 in revised algorithm */
}
```

Sequence of swapping Operations

Time	Proc A	B	C	D	E
0	0 runs	0	swap out 0	swap out 0	swap out 0
1	1	1 runs	1	1	1
2	2 swa out	2 swap out 0	2 swap out 0 runs	2 swap out 0	2
3	1	1	1	1 runs	3
4	2 swap in 0	2	2 swap out 0	2 swap out 0	4 swap out 0
5	1 runs	3	1	1	1
6	2 swap out 0	4 swap in 0 runs	2 swap out 0	2	2 swap out 0

Thrashing due to Swapping

Time	Proc A	B	C	D	E
0	0 runs	0	swap out 0	nice 25 swap out 0	swap out 0
1	1	1 runs	1	1	1
2	2 swa out 0	2 swap out 0	2 swap in 0 runs	2 swap in 0	2
3	1	1	1	1 swap out 0	3 swap out o runs
4	2 swap in 0	2	2 swap out 0	1	1
5	1	3 swap in 0 runs	1	1	2 swap out 0
6	2 swap out 0	1	2	3 swap in 0 runs	1

DEMAND PAGING

- Free from size limitation available physical memory, but depend on virtual memory.
- Load its portion dynamically and able to execute it .
- Locality and working set
- LRU(least recently used) algorithm
- Pure working set is impractical Why ?
- Two parts of implement Paging subsystem
  - Swapping rarely used pages to a swapping device
  - Handling page faults

Working Set of a Process

Sequence of Page References	Working Sets Window sizes 2	3	4	5
24	24	24	24	24
15	15 24	15 24	15 24	15 24
18	1815	18 15 24	18 15 24	18 15 24
23	23 18	23 18 15	23 18 15 24	23 18 15 24
24	24 23	24 23 18	⋮	⋮
17	17 24	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17	17 17 24	⋮	⋮
24	24 18	⋮		
18	18 24			
17	17 18			
17	17			
15	15 17	15 17 18	15 17 18 24	
24	24 15	24 15 17	⋮	
17	17 24	⋮		
24	24 17			
18	18 24	18 24 17		

Data Structures for Demand Paging

- 4 major data structure to support low-level memory manage and demand paging
  - **Page table entry**
  - **Disk block descriptor**
  - **Page frame data table(pfdata)**
  - **Swap-use table**
- The pfdata table describes each page of physical memory and is indexed by page number. The fields of an entry are
  - **The page state**
  - **The number of process that reference the page**
  - **Logical device and block number**
  - **Pointers to other pfdata table entries**

Descriptors

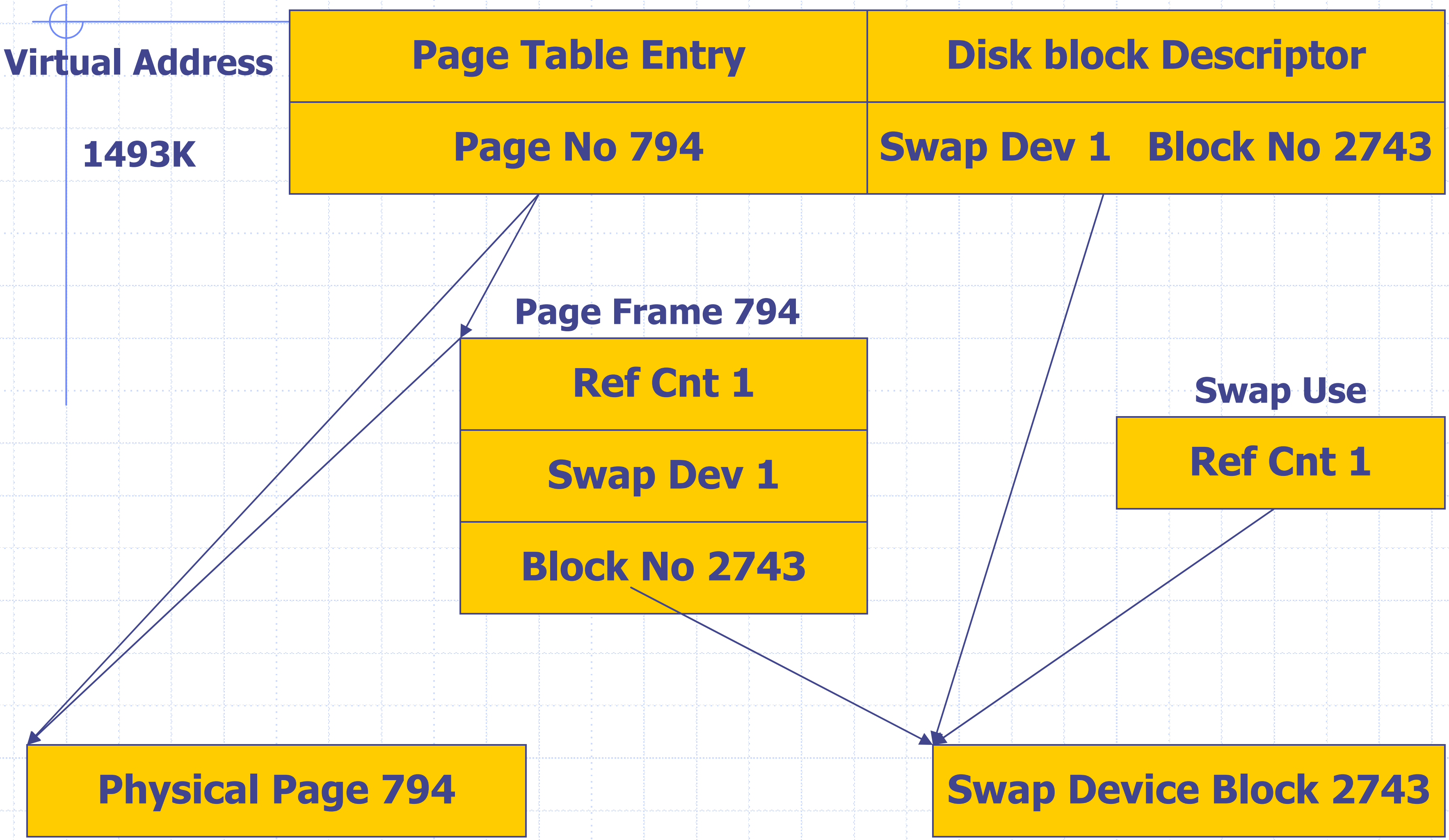
Page Table Entry

Page (Physical) Address	Age	Cp/Wrt	Mod	Ref	Val	Prot

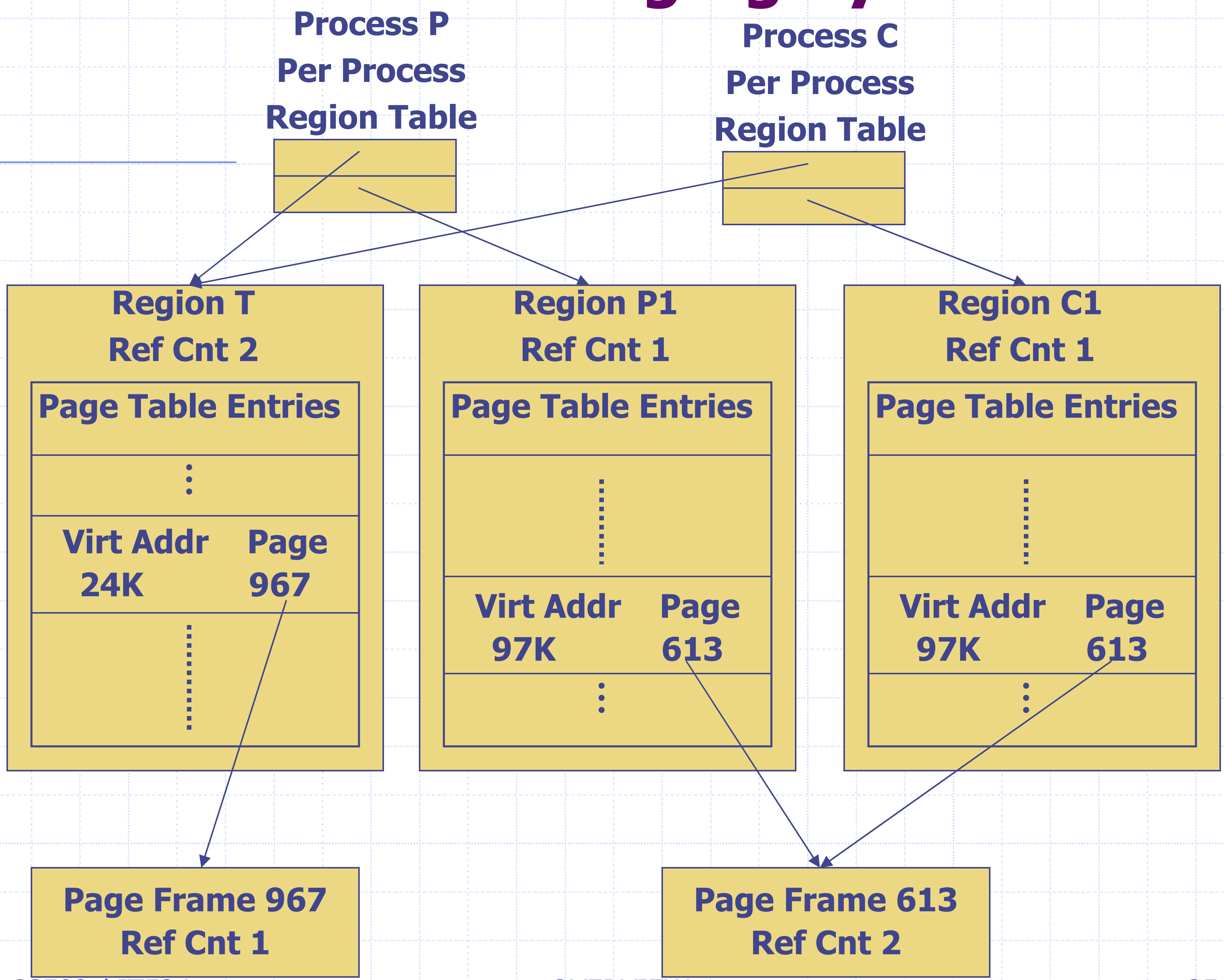
Disk Block Descriptor

Swap Dev	Block Num	type (swap, file, fill 0, demand fill)

Relationship of Data Structures for Demand Paging



Fork in a Paging System



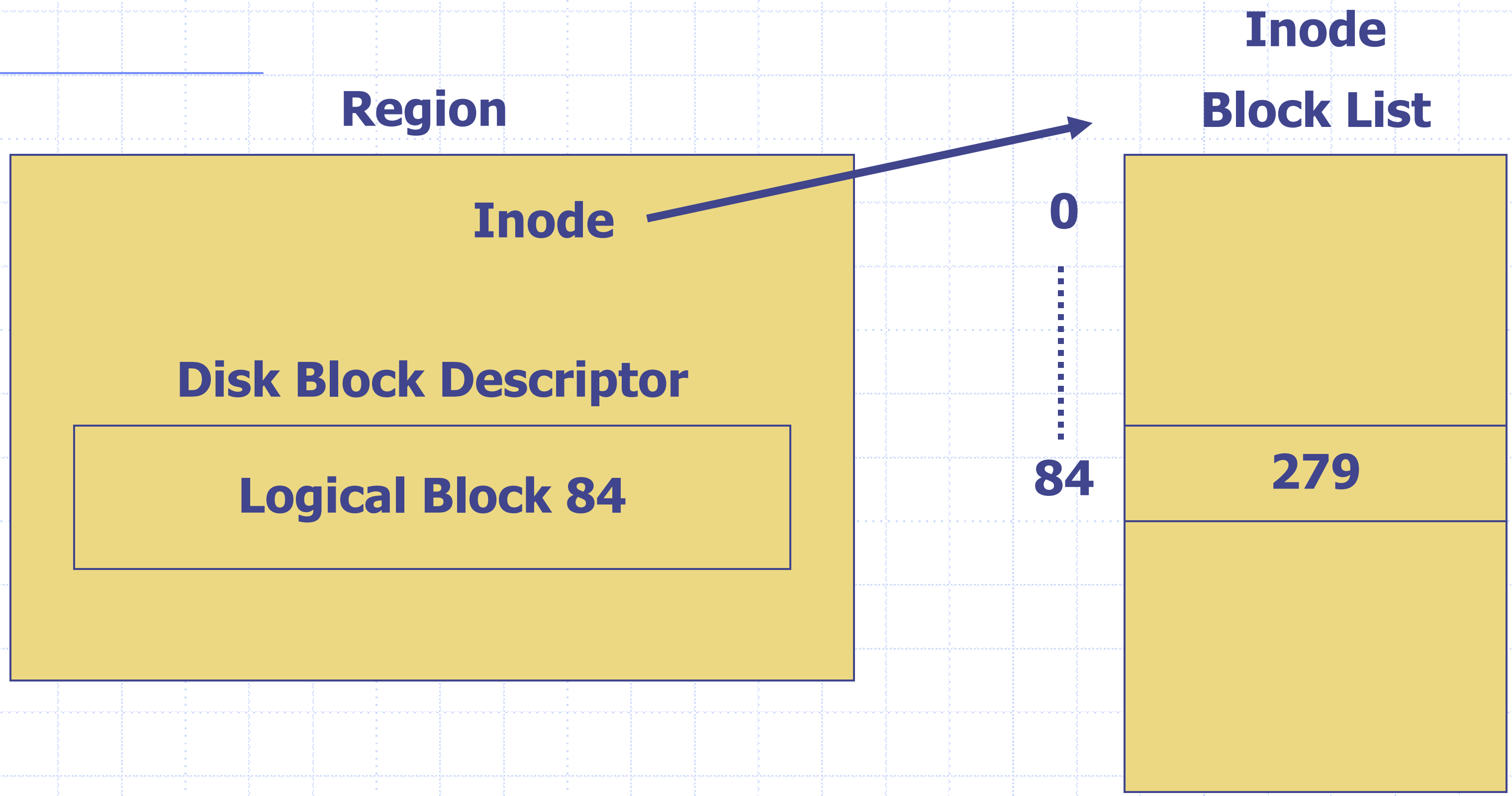
Vfork and Corruption of Process Memory

```
Int glolal;
Main()
{
          int local;

         local = 1;
         if (vfork() == 0)
         {
                  /* child */
                  global = 2;
                  local = 3;
                   _exit();
         }

}
```

Exec in a Paging system

- Kernel reads the executable file into memory from the file system.
- Executable file may be too large available memory
- not preassign memory but "faults " it in , assigning memory as needed.
- To directly from an executable file, Kernel finds all the disk block numbers of executable file and attaches list to the file inode.

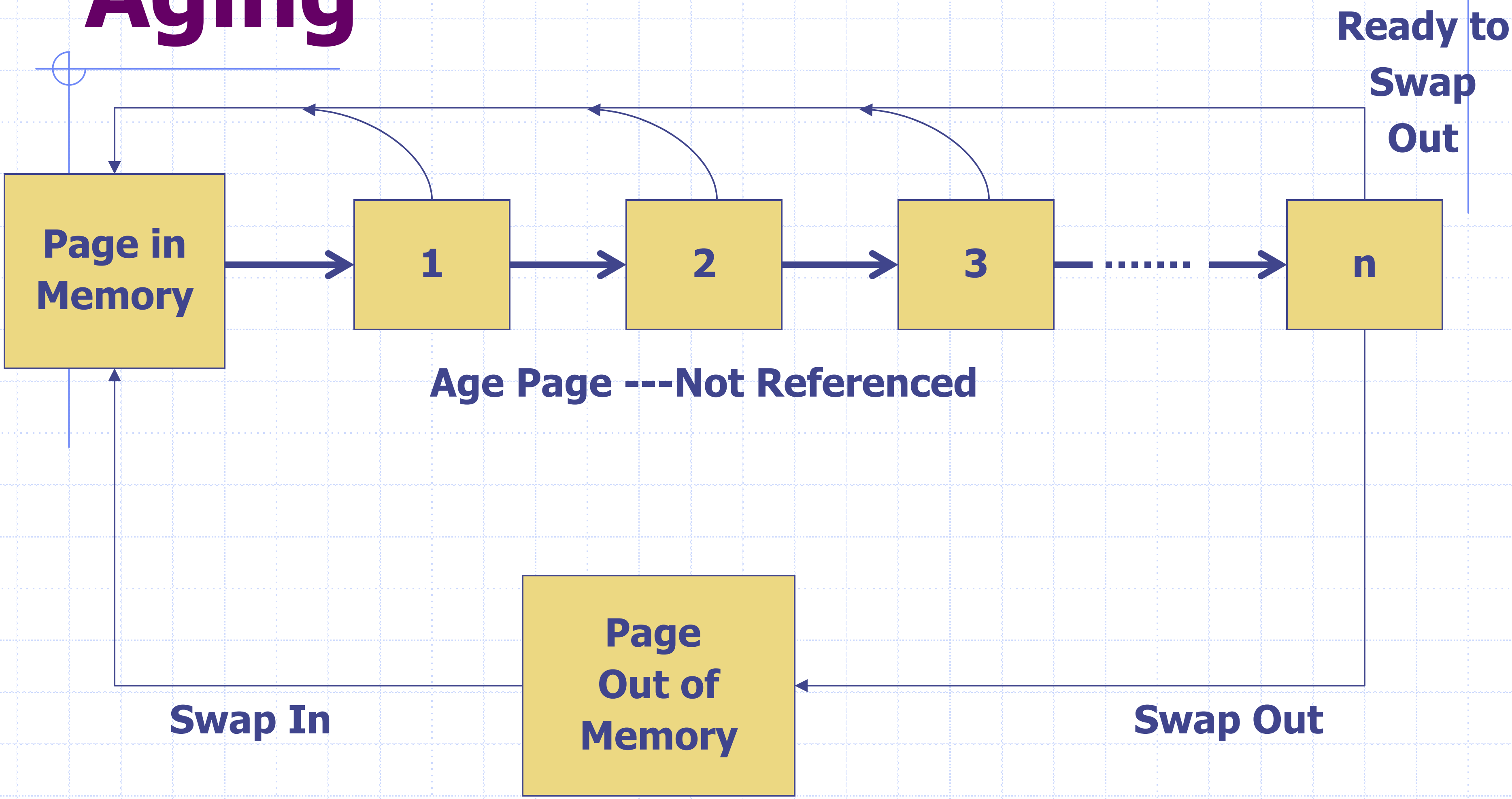
Mapping A File into a Region



The Page-stealer Process

- Kernel process that swaps out memory pages that are no longer part of the working set.
- Two paging state in memory
  - Page is aging but, not eligible swapping
  - Eligible swapping and available reassignment
- Wake up when free memory is below low-water mark.
- Swapping out until exceeds a high water mark .

State Diagram for Page Aging



Example of Aging a Page

Page state	Time(Last Reference)	
In Memory	0	
	1	
	2	Page Referenced
	0	
	1	Page Referenced
	0	
	1	
	2	
	3	Page Referenced
Out of memory		

Three possibility a copy of the page is on a swap device

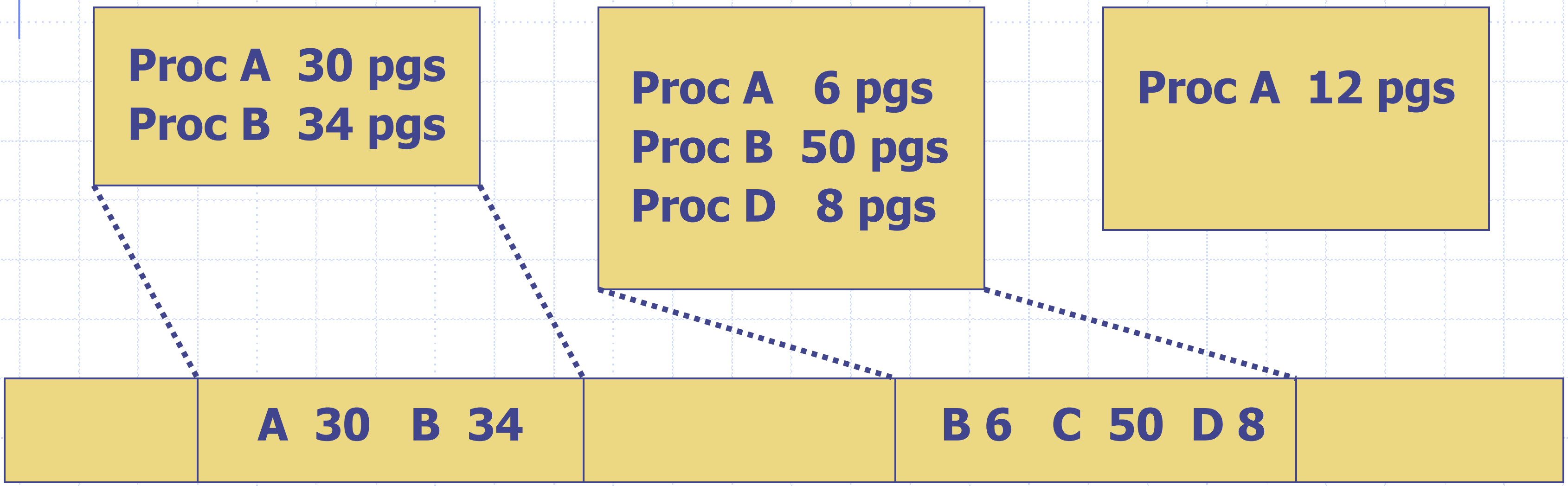
- No copy of the page is on a swap device
  - Place the page on a list of pages to be swapped out, continue
- Copt of the page is on swap and same its in-core contents
  - Cleat valid bit, decrements reference count and put the entry on free list
- Copy of the page is on swap but different its in-core contents
  - Schedule the page for swapping, free the space it currently occupies on the swap device.

Allocation of Swap Space in Paging Scheme

Two phases to swapping a page from memory

- Finds page eligible for swapping and places page number on a list of pages to be swapped
- Copy page to swap device, turn off valid bit in page table entry , decrement pfdata table entry reference count, and places pfdata table entry at end of the free list if its reference count 0.

Groups of 64 pages to Swap



Validity fault handler [1]

- If process attempts to access a page that valid bit 0, it incurs a validity fault
- Find the page table entry and disk block descriptor for the page
- "Segmentation violation" when there is no record on disk block descriptor

Validity Fault Handler [2]

- The page that caused the fault is in one of five states.
  - On a swap device and not memory
  - On the free page list in memory
  - In an executable file
  - Marked “demand zero”
  - Marked “demand fill.”

Algorithm for Validity Fault Handler (1)

```
Algorithm vfault   /*handler for validity faults  */
Input: address where process faulted
Output: none
{
    find region,page table entry, disk block descriptor
                    corresponding to faulted address, lock region;
    if (address outside virtual address space)
    {
            send signal (SIGSEGV : segmentation violation ) to process;
            goto out;
    }
    if (address now valid )     /* process may have slept above */
            goto out;
    if (page in cache)
    {
            remove page from cache;
            adjust page table entry;
            while (page contents not valid);
                    sleep (event contents become valid);     /* another proc faulted first  */
    }
```

Algorithm for Validity Fault Handler (2)

```
    else        /* page  not in cache  */
    {
            assign new page to region;

            put new page in cache, update pfdata entry;
            if (page not previously loaded and page "demand zero")
                    clear assigned page to 0;
            else
            {
                    read virtual page from swap dev or exec file;
                    sleep (event I/O done);
             }
            awaken processes (event page contents valid);
     }
     set page valid bit;
     clear page modify bit, page age;
    recalculate process priority;
  out: unlock region;

}
```

Occurrence of a Validity Fault

Virt Addr	Page Table Entries: Phys Page	Page Table Entries: State			Disk block Descriptors: State	Disk block Descriptors: Block	
0							
1K	1648	Inv			File	3	Case 3
2K							
3K	None	Inv			DF	5	Case 5
4K							
64K	1917	Inv			Disk	1206	Case 2
65K	None	Inv			DZ		Case 4
66K	1036	Inv			Disk	847	Case 1
67K							

Page Frames

Page	Disk Block	Count
1036	387	0
1648	1618	1
1861	1206	0

After Swapping Page into Memory

Virt addr	Page Table Entries: Phys Page	State			Disk block Descriptors: State	Block
66K	1776	Val			Disk	847

Page	Disk Block	Count
1776	847	1

Double Fault on a Page

Process A	Process B
Incurs Page Fault Legal Page Sleep until Page Read	
	Incurs Page Fault Legal Page Sleep until Page Read
Wake up – Page in Memory Mark Page Valid Wake up other Sleeping Process Resume Execution	
	Wakes up
	Resume Execution

Protection Fault Handler

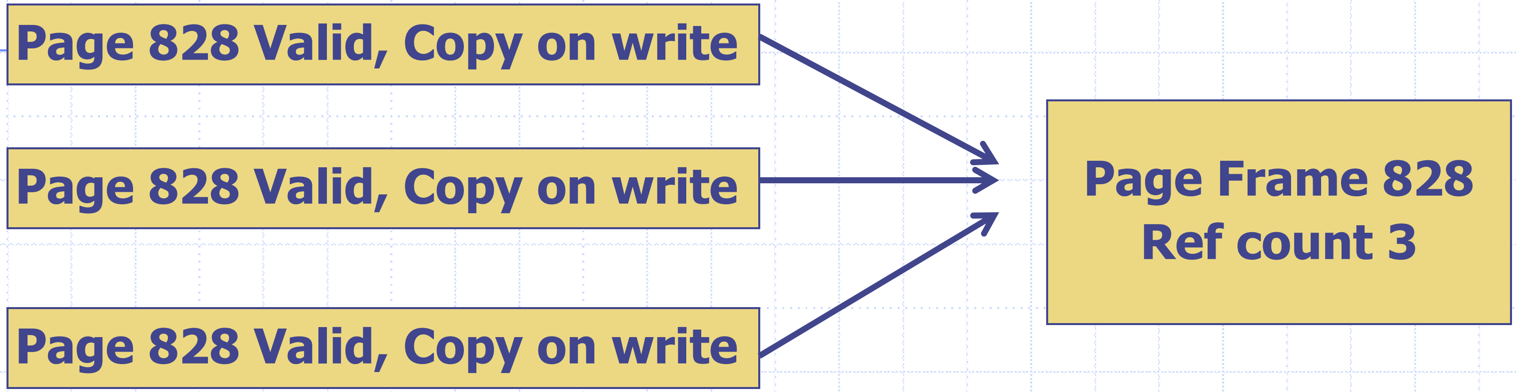
- Process accessed a valid page but page did not permit access
- When attempts to write the page that "copy on write" is set
- Find another region and page table entry then locks the region
- After finish, it set modify and protection bit but clears the copy on write bit

Algorithm for Protection Fault Handler

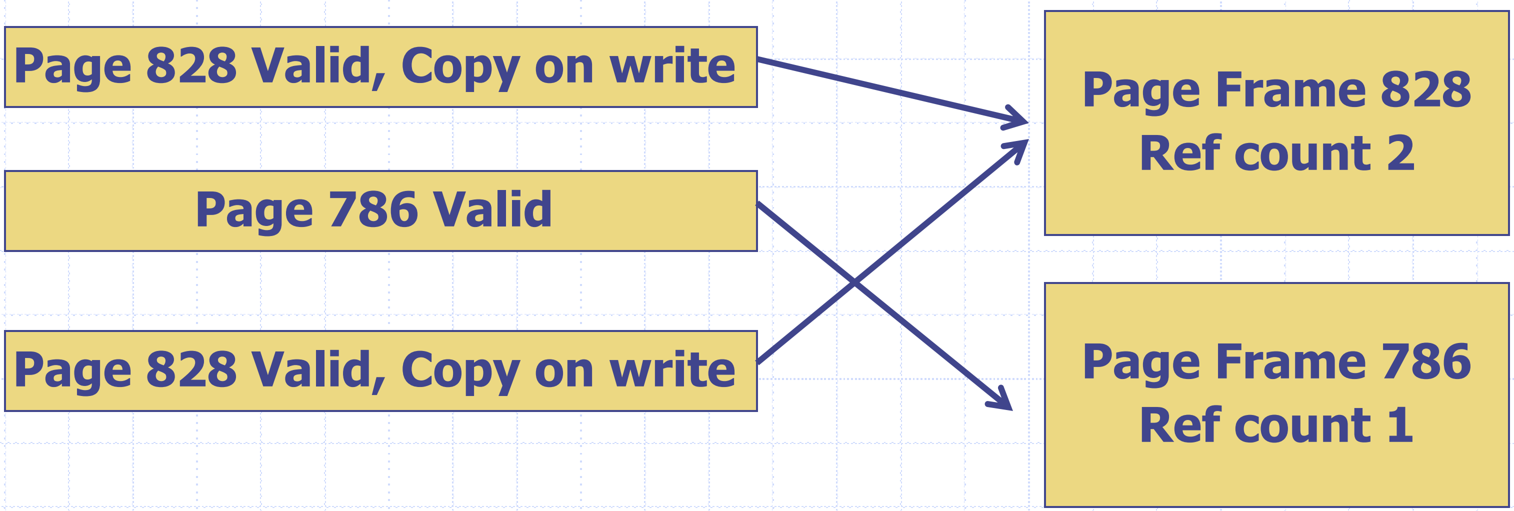
```
Algorithm pfault    /*protection fault handler  */
Input: address where process faulted
Output: none
 {
      find region, page table entry, disk block descriptor, page frame for address, lock region;
      if (page not valid in memory)
             goto out;
      if (page on write bit not set)
             goto out;
      if (page frame reference count > 1)
      {
             allocate a new physical page;
             copy contents of page to new page;
             decrement old page frame reference count;
             update page table to point to new physical page;
      }
      else          /* "steal" page, since nobody else is using it  */
      {
             if (copy of page exist on swap device)
                     free space on swap device, break page association;
             if (page is on page hash queue)
                     remove from hash queue;
      }
      set modify bit, clear copy on write bit in page table entry;
      recalculate process priority;
      check for signals;
  out: unlock region;
 }
```

Protection Fault with Copy on Write Set

(a)Before Proc B Incurs Protection Fault



(b)After Protection Fault Handler Runs for Proc B



Demand Paging on Less –Sophisticated Hardware

- It is possible to implement the paging algorithms when only have valid and protection bits using software bit
- Mimicking Hardware Modify bit in Software

Hardware Valid	Software Valid	Software Valid
Off	On	Off

(a) Before modifying Page

Hardware Valid	Software Valid	Software Valid
On	On	On

(b) After modifying Page

A HYBRID SYSTEM WITH SWAPPING AND DEMAND PAGING

- Thrashing
- The swapper swaps out entire processes until available memory exceeds the high-water mark
- Slow down the system fault rate and reduces thrashing

The I/O Subsystem

1. Driver Interfaces
2. Disk Drivers
3. Terminal Drivers
4. Streams

I/O subsystem?

- allows a process to communicate with peripheral devices
  - device driver: kernel modules that control devices
  - device drivers : device types = 1: 1
    - One terminal driver control all terminals
  - Software devices
    - Have no associated physical device

      Ex)The system treats physical memory
      as a device
      to allow a process access to physical memory
      outside its address space

I/O subsystem?

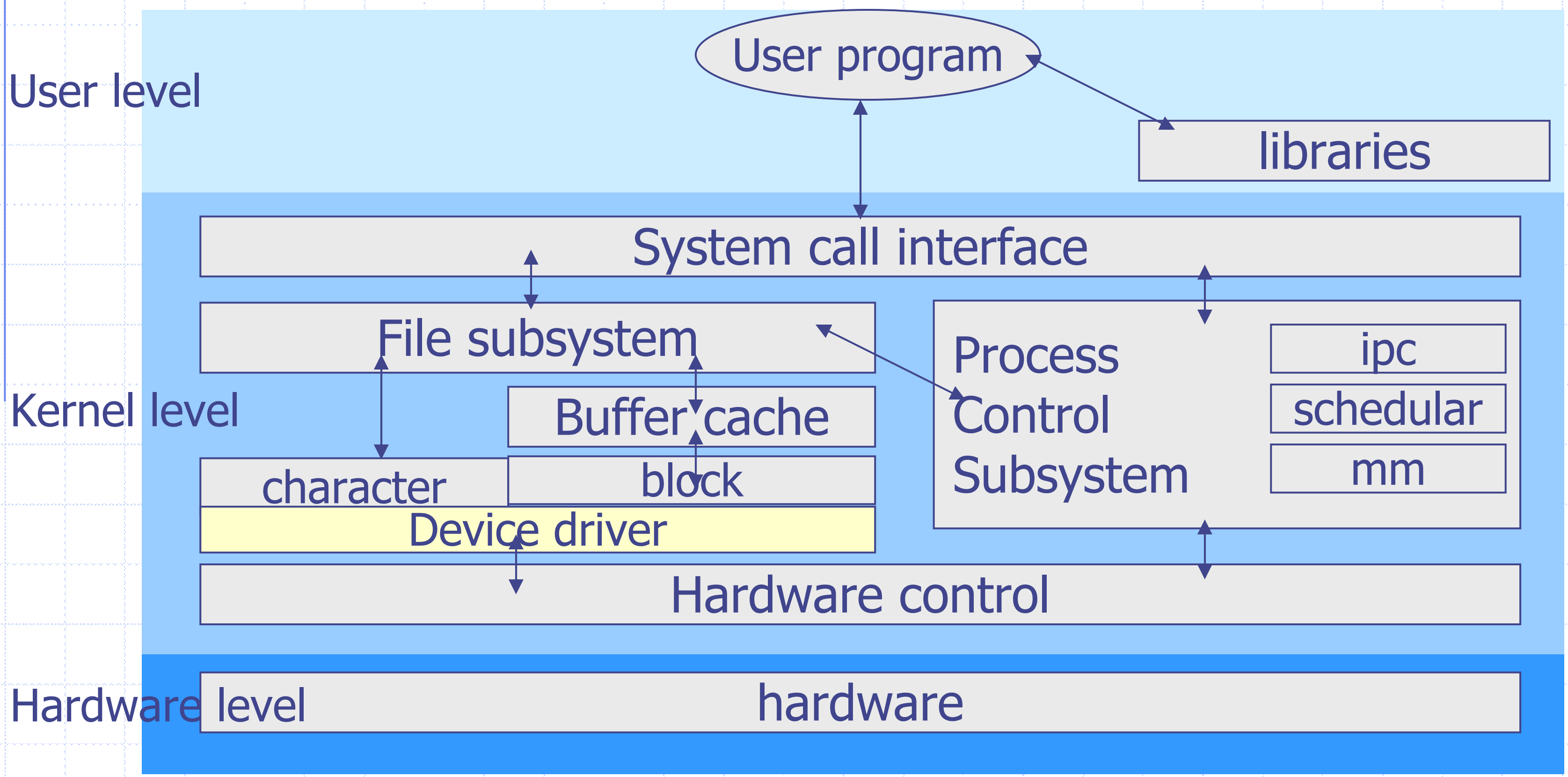


Figure 2.1 block diagram of the System Kernel

Driver Interfaces

- Unix System contains two types of devices
  - Block devices
    - Disk, tapes
  - Raw or character devices
    - Teminal, newtork media
  - User interface to devices goes through the file system
    - Looks like a file
    - But inode file type is
      - Block or
      - Character special
  - System calls for regular file have an appropriate meaning for devices

System Configuration

- System Configuration?
  - procedure by which administrators specify parameters that are installation dependent
    - Some parameter: size of kenel tables
      - inode table, file table, no of bufffer
    - Other parameter: specify device configuration
      - which devices are included in the installation and their address

System Configuration

- 3 stages of device configuration
  - 1st: hard-code configuration data into files
    - that are compiled and linked when building the kernel code
  - 2nd: supply configuration information after the system is already runing
  - 3rd: self-identifying devices permit the kernel to recognize which devices ar installed

System Configuration

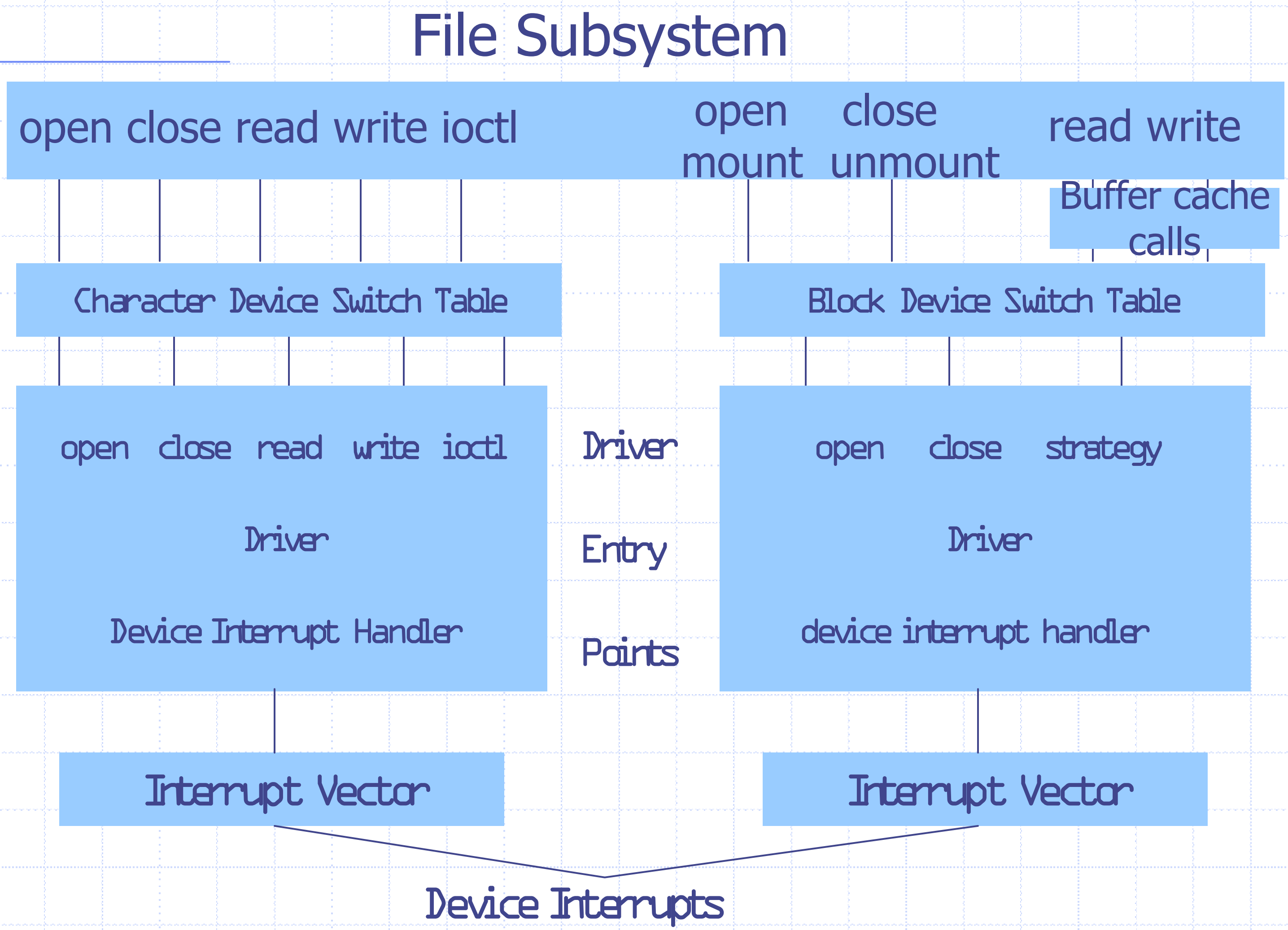


Figure 10.1. Driver Entry Points

Device special file

- Create the device file
  - mknod /dev/tty13 c 2 13
    - /dev/tty13: file name of the device
    - c: character special file (b: block special file)
    - 2: major number
      - Indicate a device type that correspoinds to the appropriate entry in the block or charcter devcie switch tables,
    - 13: minor number:
      - A unit of the device

System Call and the Driver Interface

Block device switch table			
entry	open	close	strategy
0	gdopen	gdclose	gdsstrategy
1	gtopen	gtclose	gtstrategy

Character device switch table					
entry	open	close	read	write	Ioctl
0	conopen	conclose	conread	conwrite	conioctl
1	dzbopen	dzbclose	dzbread	dzbwrite	dzbioctl
2	synopen	nulldev	syread	sywrite	syioctl
3	nulldev	nulldev	m mread	m mwrite	nodev
4	gdopen	gdclose	gdread	gdwrite	nodev
5	gtopen	gtclose	gtread	gtwrite	nodev

Figure 10.2. Sample Block and Character Device Switch Table

Open

- CALL the System call Open()
  1. Check file type from Inode table
  2. If character?
     see character device switch table
     using major number as index
  3. If block?
     see block device switch table
     using major number as index
  4. Driver function call
     with minor number parameter

Open

```
Alogrithm open /* for device drivers*/
Input: pathname,  openmode  Output: file descriptor
{
        convert pathname to inode, increment inode reference count,
        allocate entry in file table, user file descriptor,
                as, in open of regular file;
        get major, minor number from inode;

        save context (algorithm setjmp) in case of long jump from driver;

        if (block device)
        {
                use major number as index to block device switch table;
                call driver open procedure for index:
                        pass minor number, open modes;
        }
        else
        {
                use major number as index to character device switch table;
                call driver open procedure for index:
                        pass minor number, open modes;
        }

        if ( open fails in driver)
                decrement file table, inode counts;
}
```

Close

- Kernel incokes the device specific close procedure only for the last close of the device
  - Only if no other process have the device open
  - The device close procedure terminates hardware connection
- Consideration
  - 1. if Inode reference cout == 0 then close?
    - No
  - 2. Close even if mounted?
    - No

Close

```
Alogrithm close /* for devices */
Input: file descriptor   Output: none
{
        do regular close algorithm ( chap 5xxx);
        if (file table reference count not 0)
                go finish;
        if ( there is another open file and its major, minor numbers
                        are same as device being closed)
                go finish;
        if( character device)
        {
                use major number to index into character device switch table;
                call driver close routine:   parameter minor number;
        }
        if ( block device)
        {
                if (device mounted)
                        go finish;
                write device blocks in buffer cache to device;
                use major number to index into block device switch table;
                call driver close routine:   parameter minor number;
                invalidate device blocks still in buffer cache;
        }
   finish:
        release inode;

}
```

Read and Write

- Similar to those for a regular file
- Some cases?
  - kernel transmits data directly between the user address space and the device
    - ➔device drivers may buffer data internally
- The precise method in which a driver communicates with a device depends on hardware
  - Memory mapped I/O
  - Programmed I/O
  - (raw I/O)

Memory Mapped I/O

- Certain addresses in th59e kernel address space are not locations in physical memory but are special registers that control particular devices
- VAX-II computer
  - CSR: control status register
  - TDB: transmit data buffer register
  - RDB: receive data buffer register
- Ex) To write a character to terminal "/dev/tty09"
  - CSR = 1, TDB = 'character'

Memory Mapped I/O

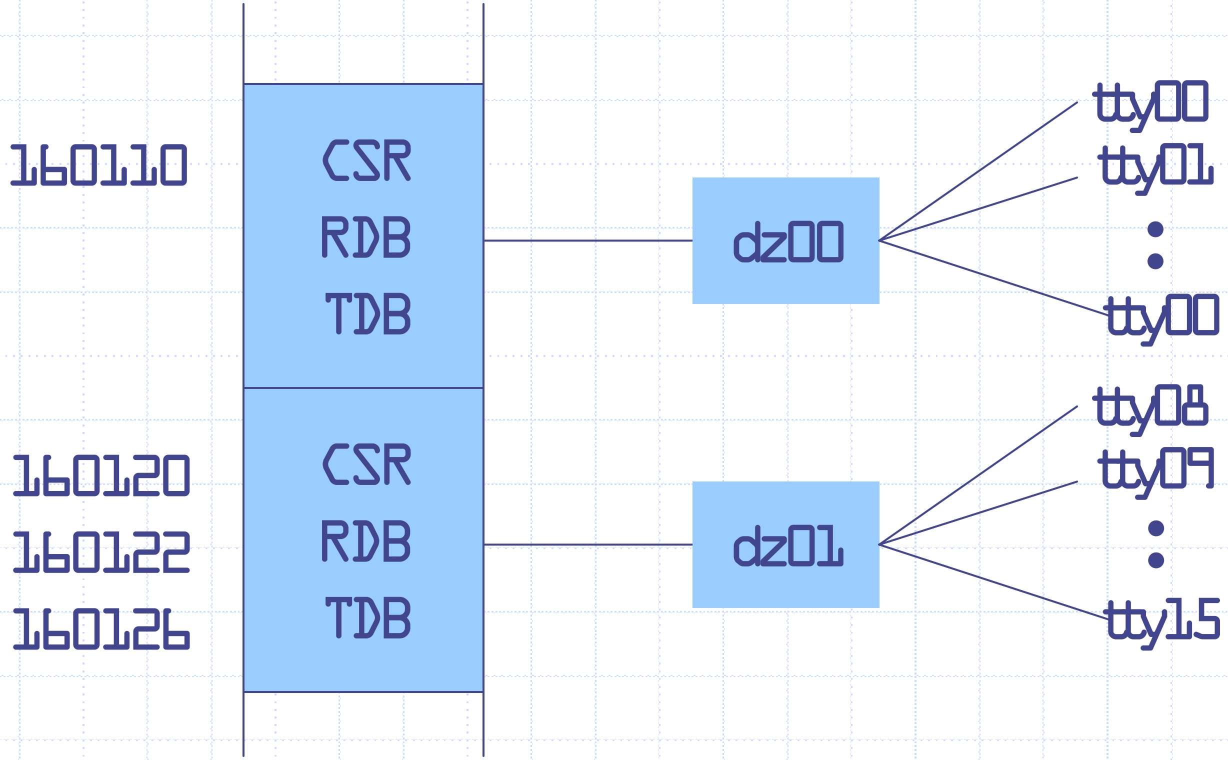


Figure 10.5. Memory Mapped I/O with the VAX DZ11 Controller

Programmed I/O

- Machine contains instructions to control devices
- Drivers control devices by executing the appopriate instruction
- Both Memory-Mapped and programmed I/O
  - A driver can issue control sequences to device to set up DMA

Raw I/O

- High-speed devices
  - Transfer data directly between the device and the user's address space, without intervention of a kernel buffer

  → result in higher speed
    - Reason 1: ther is one less copy operation in kernel
    - Reason 2: the amount of data transmitted per transfer operation is not bounded by the size of kernel buffers

Strategy Interface

- Kenel uses the strategy interface
  - Transmit data between buffer cache and devcie
  - Read, wirte → sometimes use their strategy interface
  - Ex> block devcie
    - Use buffer cache algorithm with major and minor number

Ioctl

- Generalization of the terminal-specific stty and gtty system call
- What type of file they are dealing with
  - The are device specific
- Ioctl(fd, cmd, arg)
  - Fd: file descriptor
  - Cmd: request of the driver to do a particular action
  - Arg: parameter for command

Other File system Related Calls

- Stat, chmod: do for regular file
- Lseek: just updata file table offset
- Read, write: use u-are file table offset

Interrupt Handler

- Intrrupt vector
  - If share → resolve which decie caused the interrupt
- Interrupt handelr identify
  - Major number: identifies a hardware type
  - Minor number: hardware unit

Interrupt Handler

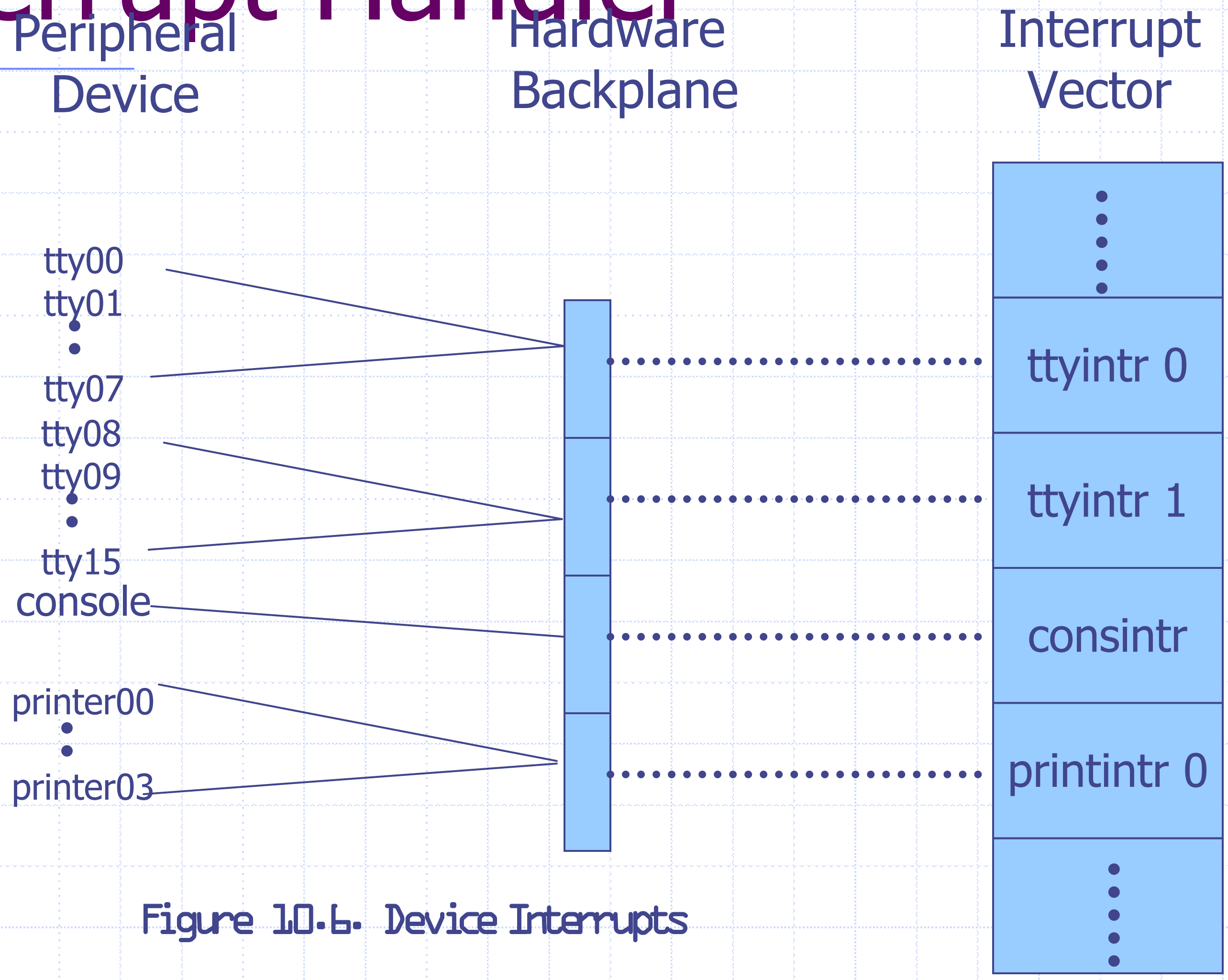


Figure 10.6. Device Interrupts

Disk Drivers

- The [disk] pack to be broken up into more managable piece
  - A disk can contains 4 file system
  - Distinguish them with minor number
- Section
  - Section may overlap on disk
  - The overlap of sections does not matter, provided the file system contained in sections are configured such that they do not overlap
  - ls –l /dev/dsk15 /dev/rdisk15
    - br-------- 2 root root 0, 21 Feb 12 15:40 /dev/dsk15
    - crw-rw---- 2 root root 7, 21 Mar 7 09:29 /dev/rdisk15

Disk Drivers

Section Start Block Length in Blocks

Size of block = 512 bytes

Section	Start Block	Length in Blocks
1	64000	944000
2	168000	840000
3	336000	672000
4	504000	504000
5	672000	336000
6	840000	168000
7	0	1008000

Figure 10.7. Disk Sections for RP07 Disk

Disk Drivers

- No raw interface
- Raw interface

- Raw interface → more faster

Disk Drivers

```
#include "fcntl.h"
main()
{   char buf1[4096], buf2[4096];
    int fd1, fd2, i;

    if ( ( ( fd1 = open("/dev/dsk5", O_RDONLY) ) == -1) ||
            ( ( fd2 = open("/dev/rdsk5", O_RDONLY) ) == -1 ){
            printf("failure on open\n");
            exit();
    }

    lseek(fd1, 8192L, 0);
    lseek(fd2, 8192L, 0);

    if ( ( read(fd1, buf1, sizeof(buf1) ) == -1) || ( read (fd2, buf2,
    sizeof(buf2) ) == -1) ){
            printf("failure on read\n");
            exit();
    }

    for( i = 0; i < sizeof(buf1); i++)
            if (buf1[I] !=buf2[I]){
                    printf("different at offset %d\n", i);
                    exit();
```

Terminal Drivers

- Terminal driver have the same function as other drivers
  - To control the transmissionof data to and from terminals
  - However, terminals are special, because they are the user’s interface to the system
- Two mode
  - Canonical mode
    - The line discipline converts raw data sequences typed at the keyboard to a canonical form before sending the data to a receiving process
  - Raw mode
    - The line discipline passes data between processes and the terminal without such conversion

Function of Line Discipline

- Parse input string into lines
- To process erase characters
- To process a “kill” character that invalidates all characters typed so far on the current line
- To echo received characters to the terminal
- To expand output such as tab characters to a sequence of blank spaces
- To generate signals to processes for terminal hang-ups, line breaks, or in response to a user hitting the delete key
- To allow a raw mode that does not interpret special characters such as erase, kill or carriage return

Terminal Drivers

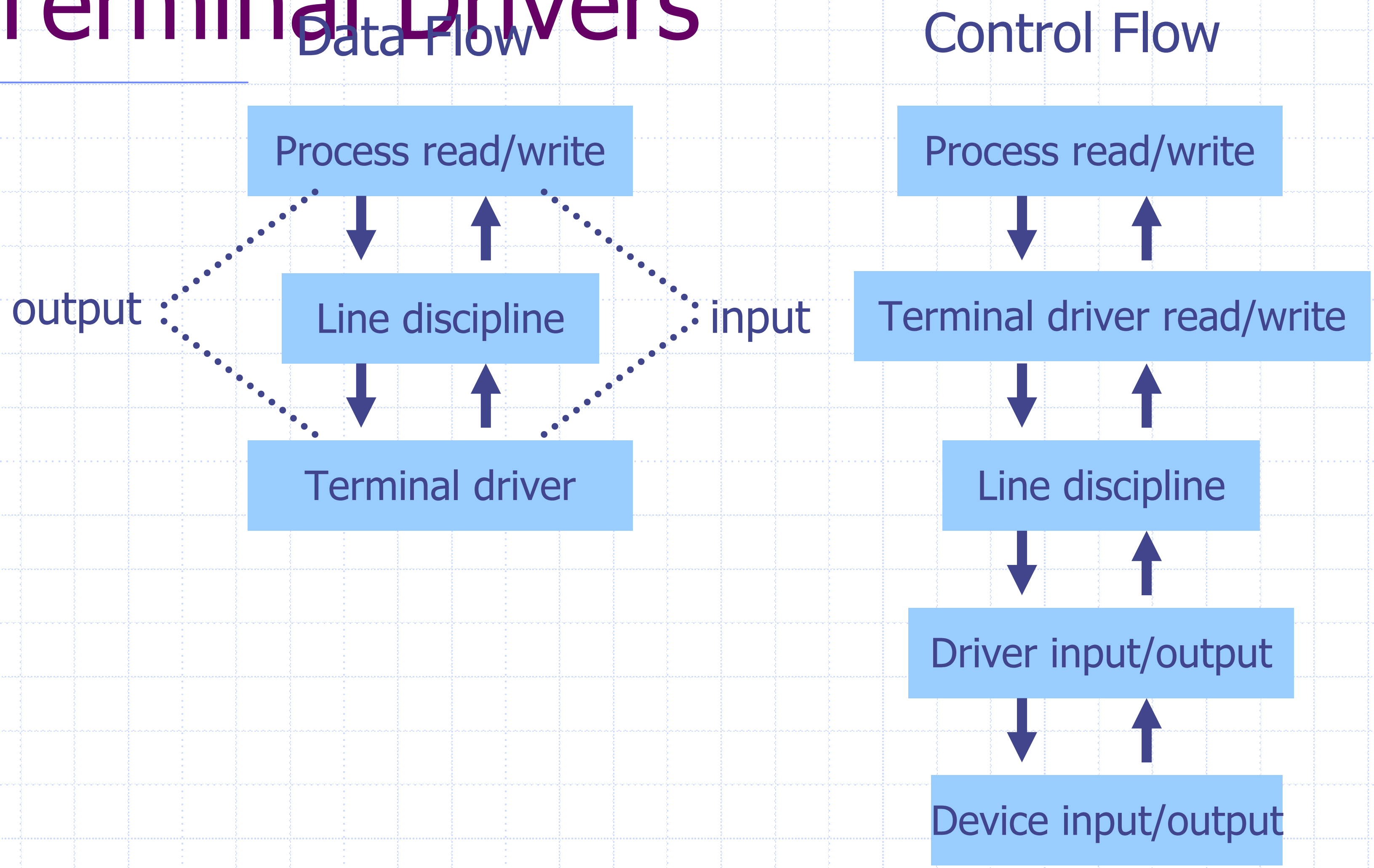


Figure 10.9. Call sequence and Data flow through line discipline

Clists

- Character list, variable length liked list of cblocks with a count of number of characters on the list
- Kernel maintains a linked list of free cblocks and has six operation on clist and cblocks

<table>
<tr><th>Next Ptr</th><th>Start Offset</th><th>End Offset</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th></th></tr>
<tr><td>↓</td><td>7</td><td>14</td><td>g</td><td>a</td><td>r</td><td>b</td><td>a</td><td>g</td><td>e</td><td>|</td><td></td><td>e</td><td>q</td><td>n</td><td></td><td>|</td><td></td><td>..</td></tr>
</table>

Figure 10.10. A Cblock

Clist example

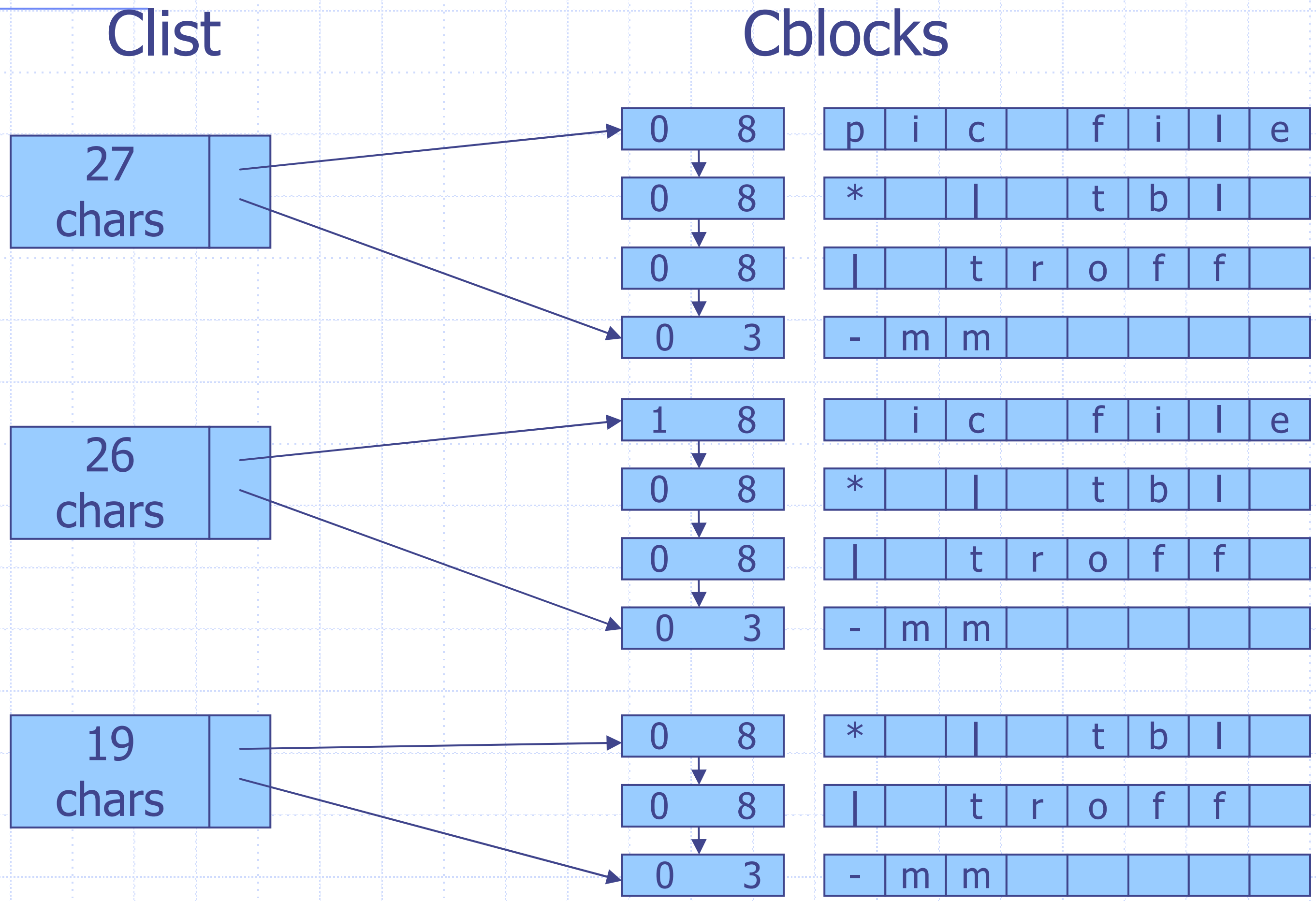


Figure 10.11. Removing Characters from a Clist

Clist example

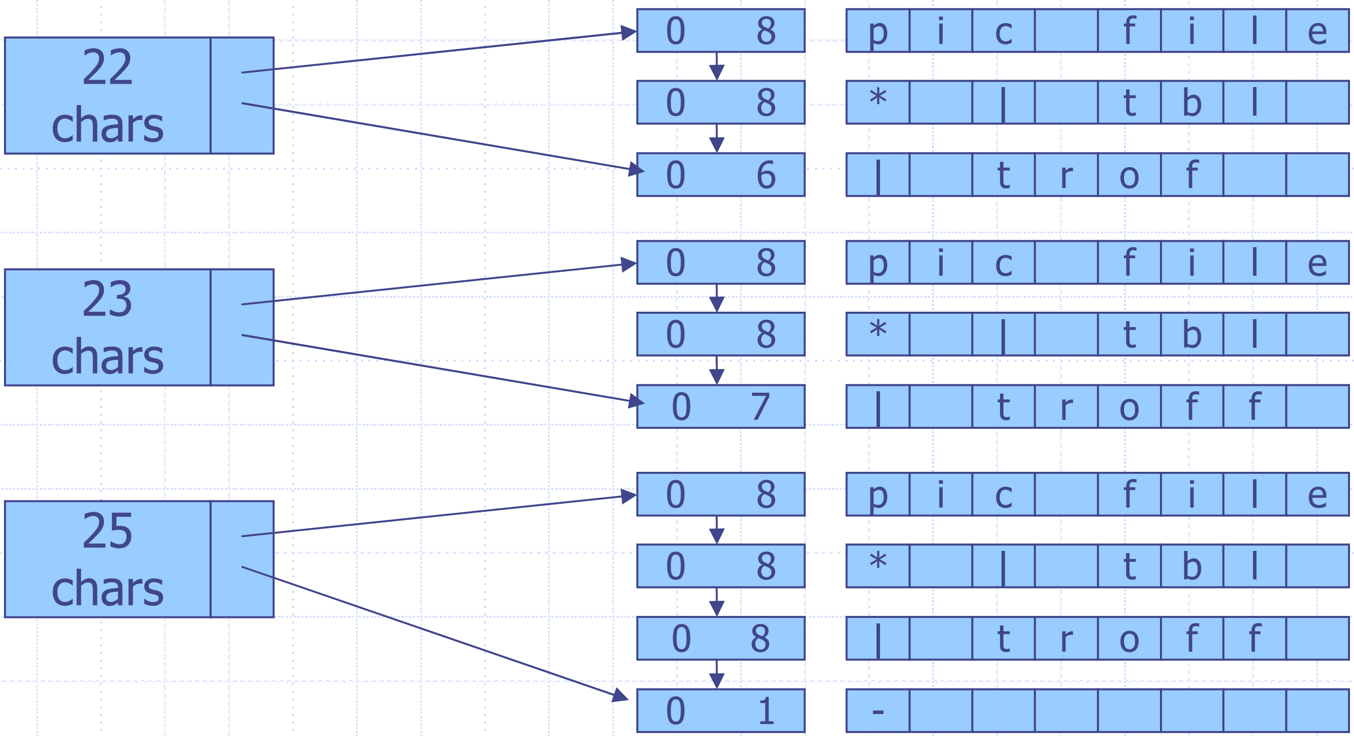


Figure 10.12. Placing a Character on a Clist

The Terminal Driver in Canonical Mode

```
algorithm terminal_write
{
        while (more data to be copied from user space)
        {
                if( tty flooded with output data)
                {
                        start write operation to hardware with data
                                                on output clist;
                        sleep( event: tty can accept more data);
                        continue;
                }
                copy cblock size of data from user space to output clists:
                        line discipline converts tab characters, etc;
        }
        start write operation to hardware with data on output clist;
}
```

Terminal Example

```
char form[] ="this is a sample output string from child ";
main(){
        char output[128];
        int i;

        for ( i = 0; i < 18 ; i++)   {
                switch (fork())
                {
                case -1: /* error - hit max procs*/
                        exit();
                default: /* parent process*/
                        break;
                case 0:  /* child process*/
                        /* format output string in variable output */
                        sprintf(output, "%s%d\n%s%d\n", form, i, form, i);
                        for (;;)   write(1, output, sizeof(output));
                }
        }
}
```

Terminal read

```
algorithm terminal_read{
        if ( no data on canonical clist){
                while( no data on raw clist){
                        if (tty opened with no delay option)
                                return;
                        if( tty in raw mode based on timer and timer not active)
                                arrange for timer wakeup(callout table);
                        sleep(event: data arrives from terminal);
                }
                /* there is data on raw clist*/
                if ( tty in raw mode)
                        copy all data from raw clist to canonical clist;
                else{   while ( characters on raw clist){
                                copy one character at a time from raw clist
                                        to canonical clist:
                                        do erase, kill processing;
                                if ( char is carriage return or end-of-file)
                                        break;
                        }
                }
        }
        while ( characters on canonical list and read count not  satisfied)
                copy from cblocks on canonical list to user address space;
}
```

Terminal input example

```
char input[256];
main()
{
        register int i;
        for ( i = 0; i < 18; i++) {
                switch( fork()){
                case –1: /* error*/
                        printf("error cannot fork\n");
                        exit();
                default: /* parent process */
                        break;
                case 0: /* child process */
                        for (;;){
                                read( 0, input, 256); /* read line*/
                                printf(" %d read %s\n", i, input);
                        }
                }
        }
}
```

The Terminal Driver in Raw Mode

```
#include <signal.h>
#include <termio.h>
Struct termio savetty;
main()
{
            extern sigcatch();
            struct termio newtty;
            int nrd;
            char buf[32];
            signal(SIGINT, sigcatch);
            if( ioctl(0, TCGETA, &savetty) == -1)
            {
                        printf("ioctl failed: not a tty\n");
                        exit();
            }
            newtty = savetty;
            newtty.c_lflge &= ~ICANON;
            newtty.c_lflge &= ~ECHO;
            newtty.c_cc[VMIN] = 5;
            newtty.c_cc[VTIME] = 100;
            if ( ioctl(0, TCSETAF, &newtty) == -1)
            {
                        printf("cannot put tty into raw mode\n");
                        exit();
            }
            for (;;)
            {
                        nrd = read(0, buf, sizeof(buf));
                        buf[nrd] = 0;
                        printf("read %d chars'%s'\n", nrd, buf);
            }
}
sigcatch()
{
            ioctl(0, TCSETAF, &savetty);
            exit();
}
```

Terminal Polling

- Select(nfds, rfds, wfds, efds, timeout)
  - Nfds: gives the number of file descriptors being selected
  - Rfds, wfds, efds: bit mask that select open file descriptor
  - Timeout: indicates how long select shoud sleep, waiting for data to arrive

Terminal Polling

```
#include <fcntl.h>
main()
{
        register int i, n;
        int fd;
        char buf[256];
        /*open terminal read-only with no-delay option*/
        if ( ( fd = open("/dev/tty", O_RDONLY|O_NDELAY) ) == -1)
                exit();
        n = 1;
        for(;;){
                for( i = 0; i < n; i ++) ;
                if( read (fd, buf, sizeof(buf)) < 0){
                        printf("read at n %d\n", n);
                        n --;
                } else  /* no data read; return due to no-delay*/
                        n ++;
        }
}
```

Logging In

```
algorithm login /* procedure for logging in */
{
        getty process executes;
        set process group( setpgrp system call);
        open tty line; /* sleep until opened*/
        if ( open successful)
        {
                exec login program;
                prompt for user name;
                turn off echo, prompt for password;
                if (successful)     /* matches password in /etc/passwd*/
                {
                        put tty in canonical mode( ioctl );
                        exec shell;
                }
                else
                        count login attempts, try again up to a point;
}
```

Streams

- Drawback of device drivers
  - The lack of comonality at the driver level percolates up to the user command level
    - Where several commands may accomplish common logical functions but over different media
  - Newtork protocols require a line discipline-like capability
    - Where each discipline implements on part of a protocol and the component parts can be combined in a flexible manner

Stream

- Streams are a scheme for improving the modularity of devie drivers and protocols
- A Stream is a full-duplex connection between a process and da device
- Streams modules are characterized by well-defined interfaces and by their flexibility for use in combination with other modules
- The flexibility they offer has strong benefits for network protocols and drivers

Streams

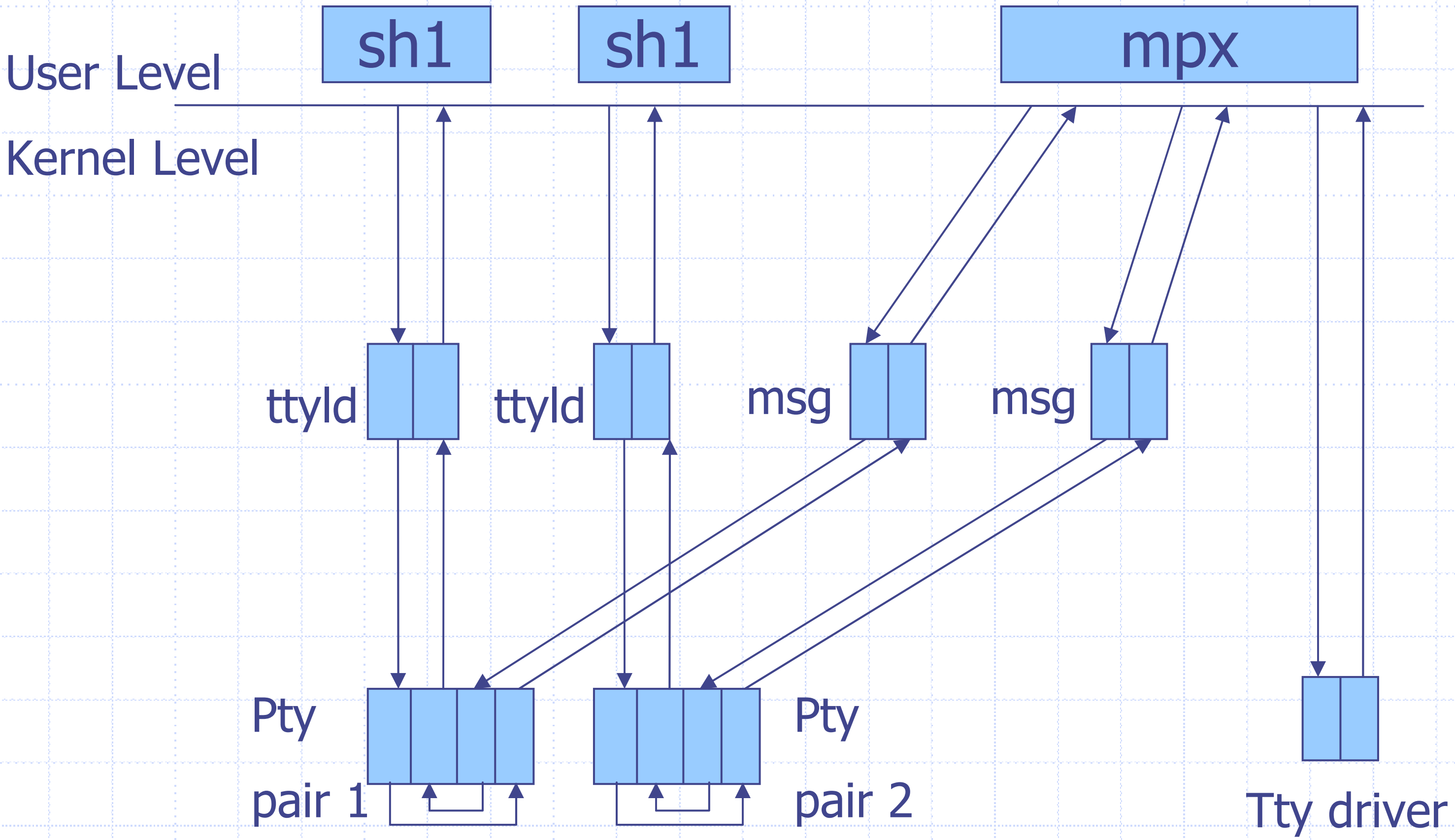


Figure 10.23. Windowing Virtual Terminals on a Physical Terminal

Streams

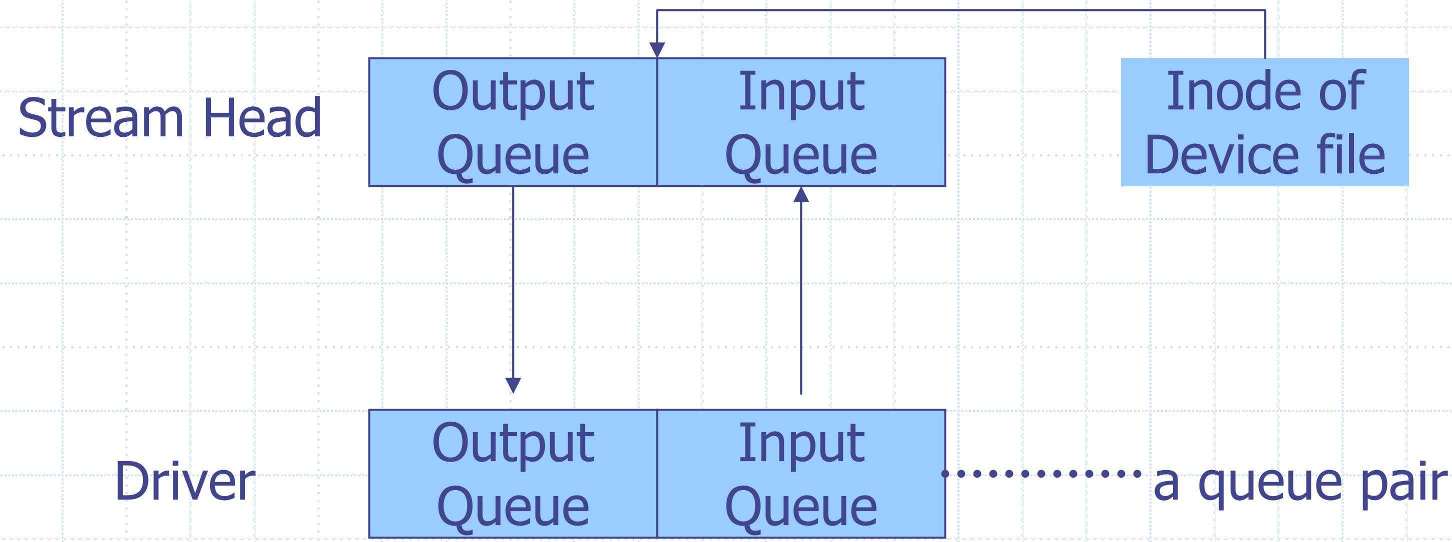


Figure 10.20. A Stream after open

Streams

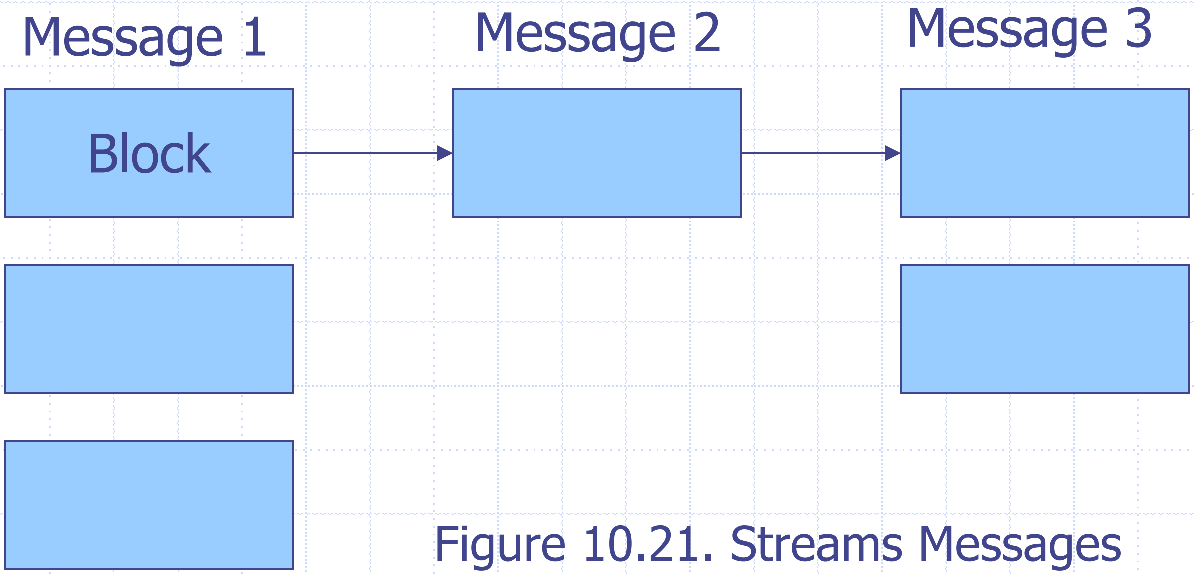


Figure 10.21. Streams Messages

A More Detailed Examples of Streams

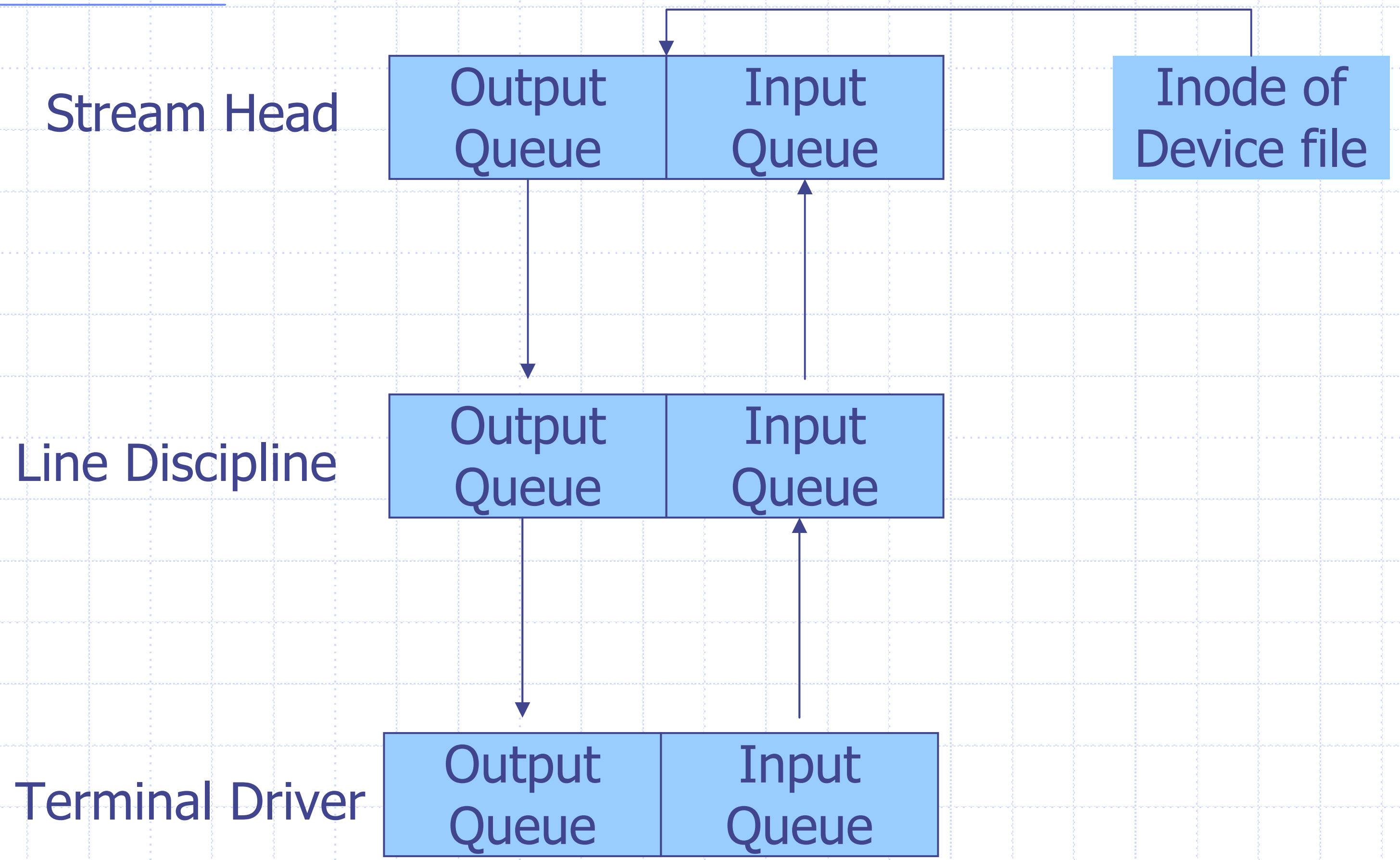


Figure 10.22. Pushing a Module onto a Stream

Reference

- LINUX KERNEL INTERNALS
  - Beck, Bohme, Dziadzka, Kunitz, Magnus, Verworner
- The Design of the Unix operating system
  - Maurice j.bach
- Understanding the LINUX KERNEL
  - Bovet, cesati