Introduction to operating system labs

- Introduction to the course
- Unix shell
- Support from hardwares
- What is an OS?

- Introduction to the course

An introduction

- 余子濠　zihaoyu.x@gmail.com
- course main page
  - http://cslab.nju.edu.cn/ics/index.php/os:2012
- office hour
  - 周五晚上7:00~9:00
  - 基础实验楼乙223

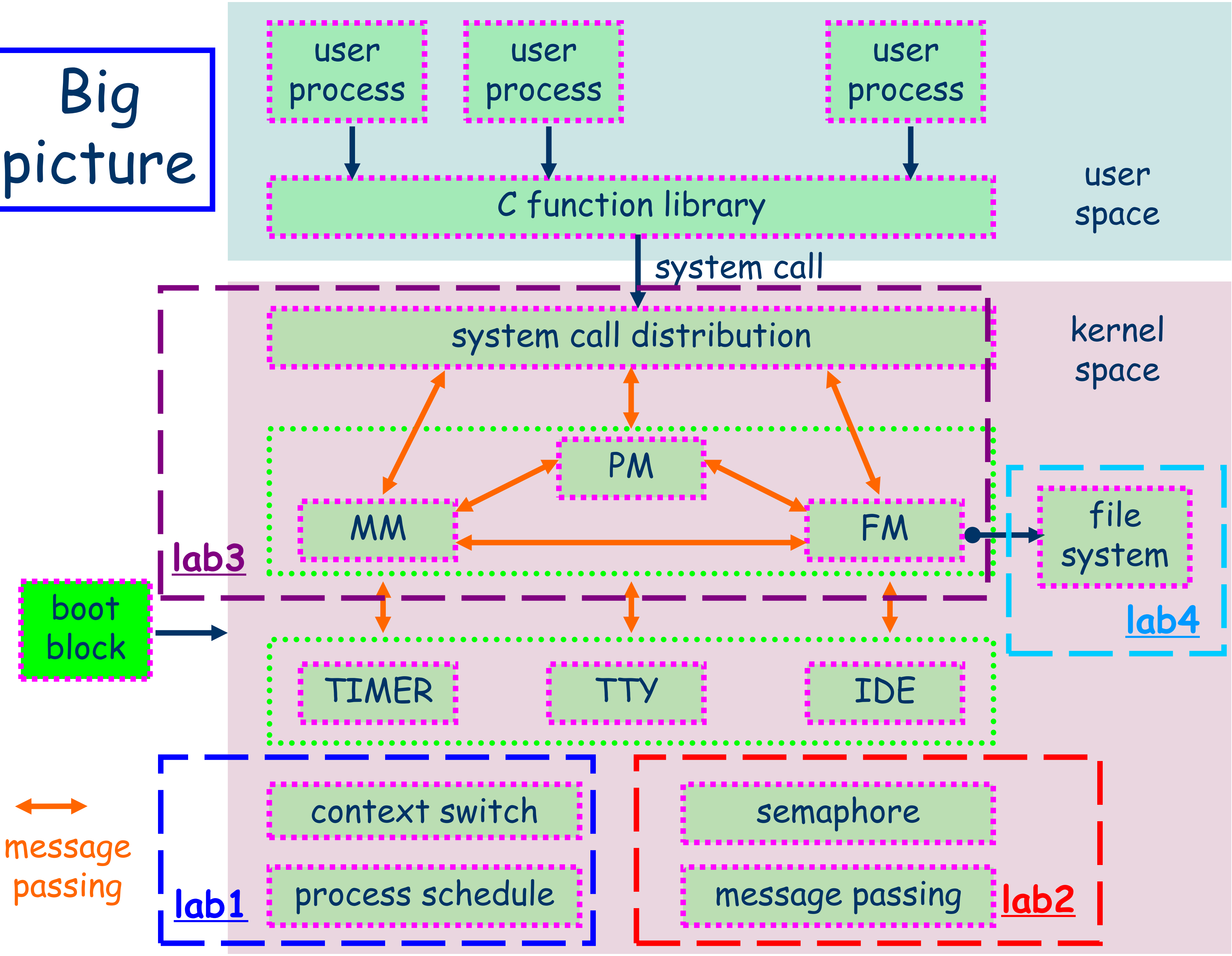
Declaration

- The main content of this course is inherited from “NJU OS labs, 2012”, designed by jyy.
- The skeleton code we use is based on “NJU OS labs, 2012”, provided by jyy.
- We have obtained agreement from jyy.
- Some new features are added by us.
- The ppts are designed by us.

What

- implement a tiny, but with necessary functions, operating system
  - called Nanos, named by jyy
  - with a kenel, 3 drivers, 3 servers, 18 system calls
- from the view of bottom-up
- in a “terrible” development environment

Big picture


user process
user process
user process
C function library
user space
system call
system call distribution
kernel space
PM
MM
FM
file system
lab3
lab4
boot block
TIMER
TTY
IDE
context switch
process schedule
lab1
semaphore
message passing
lab2
message passing

5 Labs

- Lab0 – a game without support of OS
- Lab1 – context switch
- Lab2 - IPC
- Lab3 – user process & system call
- Lab4 - file system

This is “the hardest” lab course

- pure C language
- Linux CLI
- about 5000+ lines of code
- no libaray functions is avaliable
- materials are not easy to understand
- debugging is “inconvenient”

- You will fail, fail, fail, fail, …, then secceed.

Why

- Implementation has its own troubles.
  - more practical
  - but seldom mentioned in theoretical course
- Experiments helps you
  - understand theoretical knowledge better
  - find your weak points

Why (cont.)

```
int a[1000];
int main() {
    a[100] = 1;
    a[1000] = 1;
    a[10000] = 1;
}
```

- System design helps you justify some phenomena from a systematic view.
- Training
  - programming, debugging, searching material…
- http://cslab.nju.edu.cn/opsystem/#Why

How

- Don’t be afraid.
  - It is a good chance for training.
- Make yourself clear about the basic concept.
- Get rid of “拖延症”.
  - NEVER start your tasks around the deadline.
- Make good use of reference materials and Internet.

Troubles

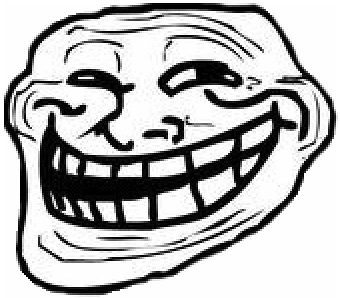
- Try to solve them by yourself
  - textbook
  - reference materials
  - our ppts
  - Linux manual pager
  - www.google.com
  - en.wikipedia.org
  - stackoverflow.com

But NOT

- www.baidu.com
- baike.baidu.com
- zhidao.baidu.com

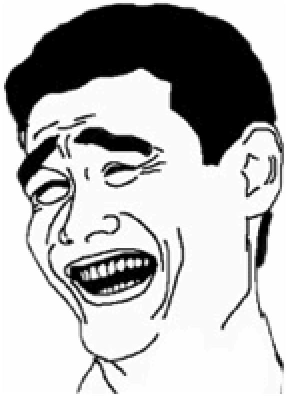
Honor code

- Every line of your code must be coded by yourself.
- discount factor – f
  - f = 1
  - every time you cheat
    - f = f * 2
    - effect score of that lab = 0
  - final score = sum of effect score / f
- This rule applys for two parties.

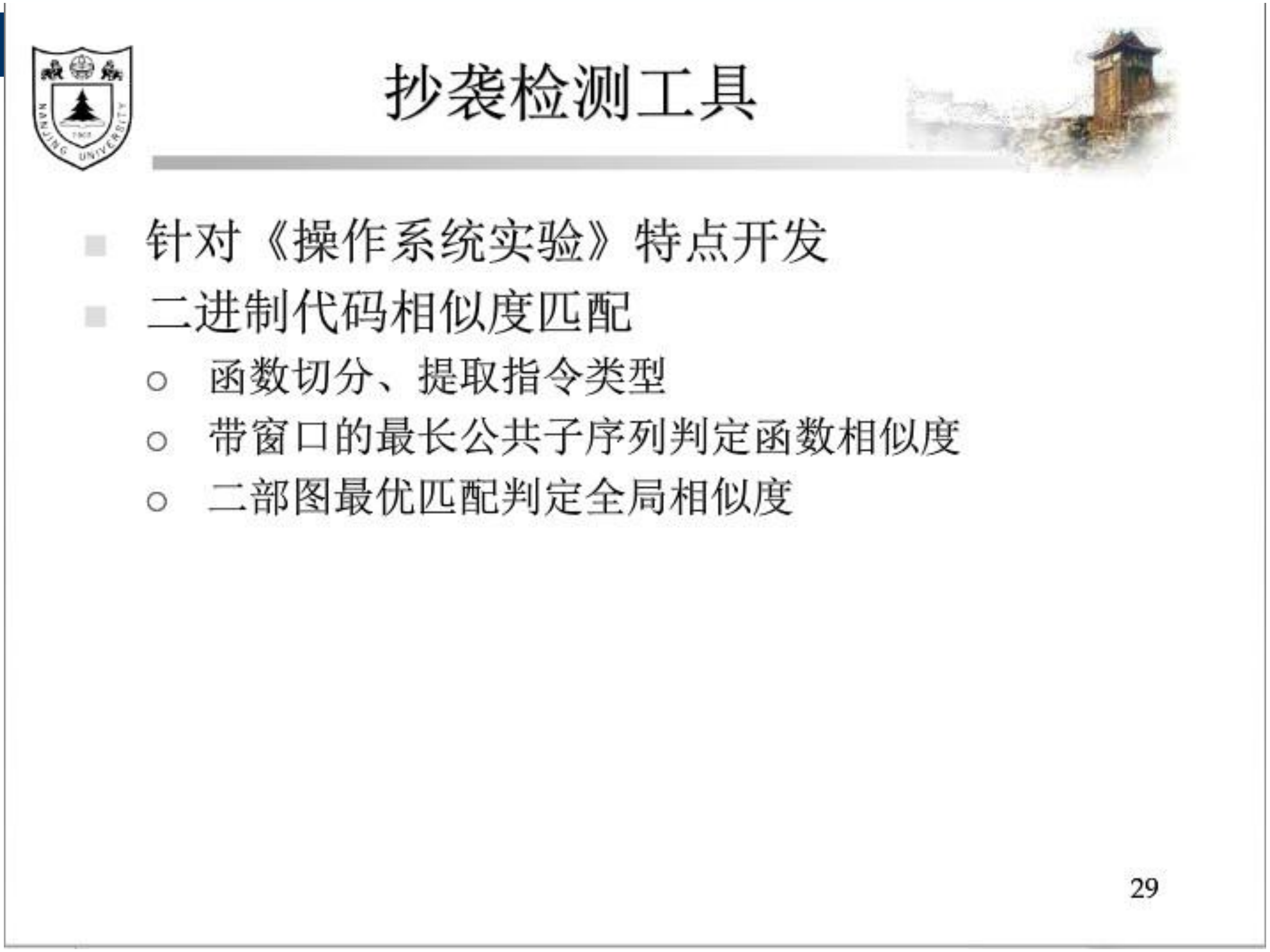


Honor code (cont.)

- Your cheating may be exposed by “anti-cheating” system designed by jyy
  - welcome to provide test sets
  - query jyy for more details



“Anti-cheating” system
- from jyy’s report



抄袭检测工具

- 针对《操作系统实验》特点开发
- 二进制代码相似度匹配
  - 函数切分、提取指令类型
  - 带窗口的最长公共子序列判定函数相似度
  - 二部图最优匹配判定全局相似度

Small test about C

- What should be filled ?

A. int **a

B. int *a [20]

C. int (*a) [20]

D. int (*a) (int [20])

E. int (*a) (int)

F. int *a [10]

G. int (*a) [10]

H. int (*a) (int [10])

I. None of above is right

```
void fun( ______ );
int main() {
    int a[10][20];
    fun(a);
    …
}
```

- If you feel confused, review is necessary
  - http://docs.huihoo.com/c/linux-c-programming/

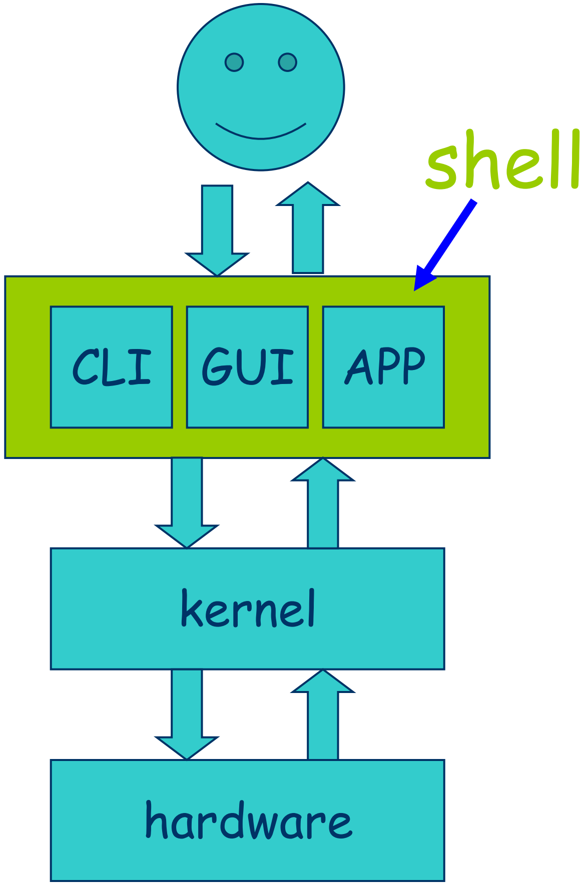
Axioms of debugging

- **Axiom 1** The machine is always correct.
  - **Corollary** If the program does not produce the desired output, it is the programmer's fault.
- **Axiom 2** Every line of untested code is always wrong.
  - **Corollary** Mistakes are likely to appear in the "must-be-correct" code.
- They are proposed as facts by jyy.

- Unix shell

Where is shell?

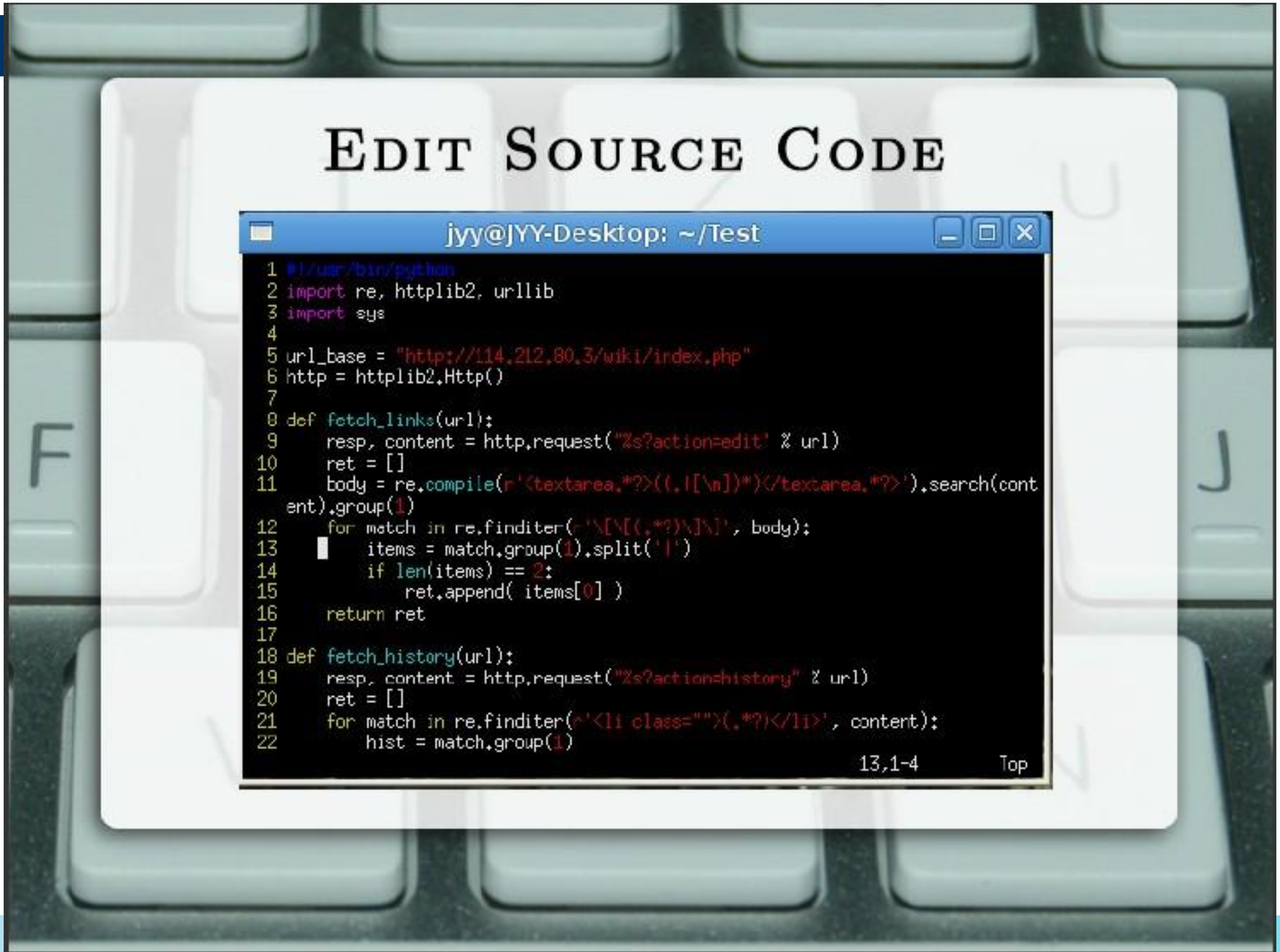
- hardware
  - the entity of computer
  - CPU, memory, disk, adapter…
- kernel
  - the kernel of operating system
  - control the hardware “directly”
- shell
  - the shell of kernel
  - receive commands from user
  - communicate with kernel
- user
  - issue commands to shell



Is GUI necessary?

- Why do you use computer?
  - edit documents
  - programming
  - surf the Internet
  - QQ
  - movie
  - game
  - …

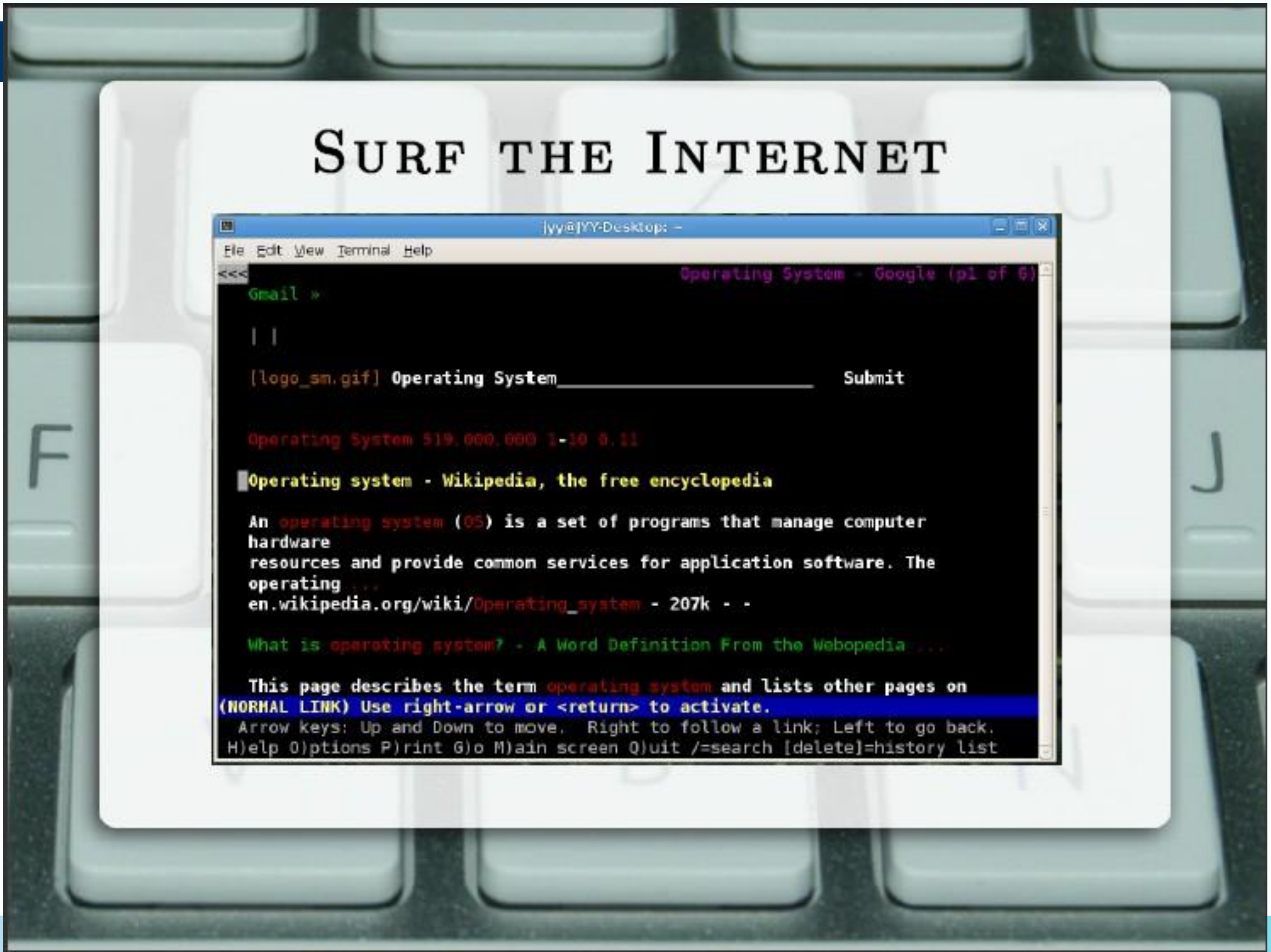
Examples from jyy



EDIT SOURCE CODE

```
#!/usr/bin/python
import re, httplib2, urllib
import sys

url_base = "http://114.212.80.3/wiki/index.php"
http = httplib2.Http()

def fetch_links(url):
    resp, content = http.request("%s?action=edit' % url)
    ret = []
    body = re.compile(r'<textarea.*?>((.|[\n])*)</textarea.*?>').search(cont
ent).group(1)
    for match in re.finditer(r'\[\[(.*?)\]\]', body):
        items = match.group(1).split('|')
        if len(items) == 2:
            ret.append( items[0] )
    return ret

def fetch_history(url):
    resp, content = http.request("%s?action=history" % url)
    ret = []
    for match in re.finditer(r'<li class="">(.*?)</li>', content):
        hist = match.group(1)
                                                          13,1-4        Top
```

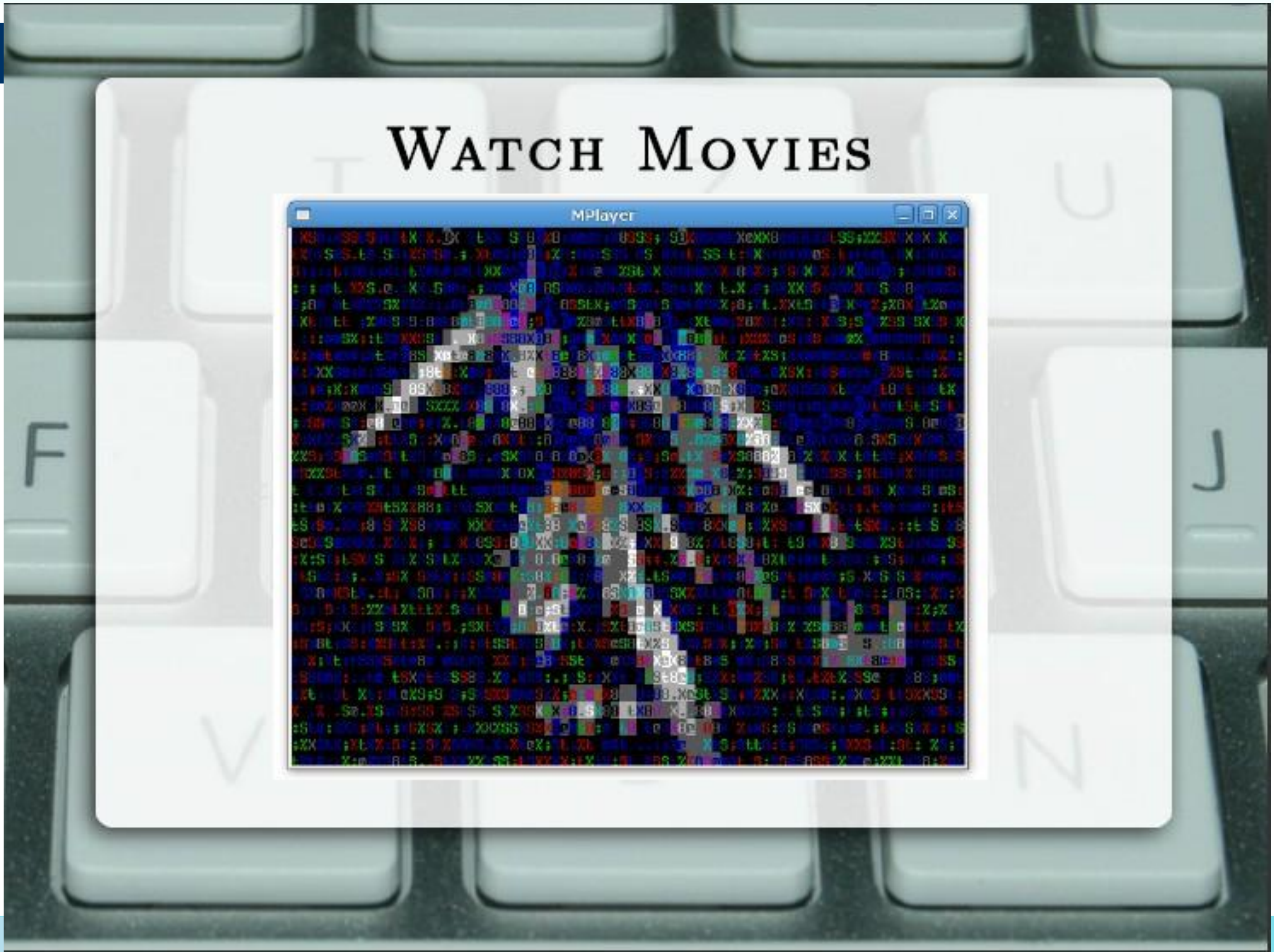
Examples from jyy (cont.)



Surf the Internet

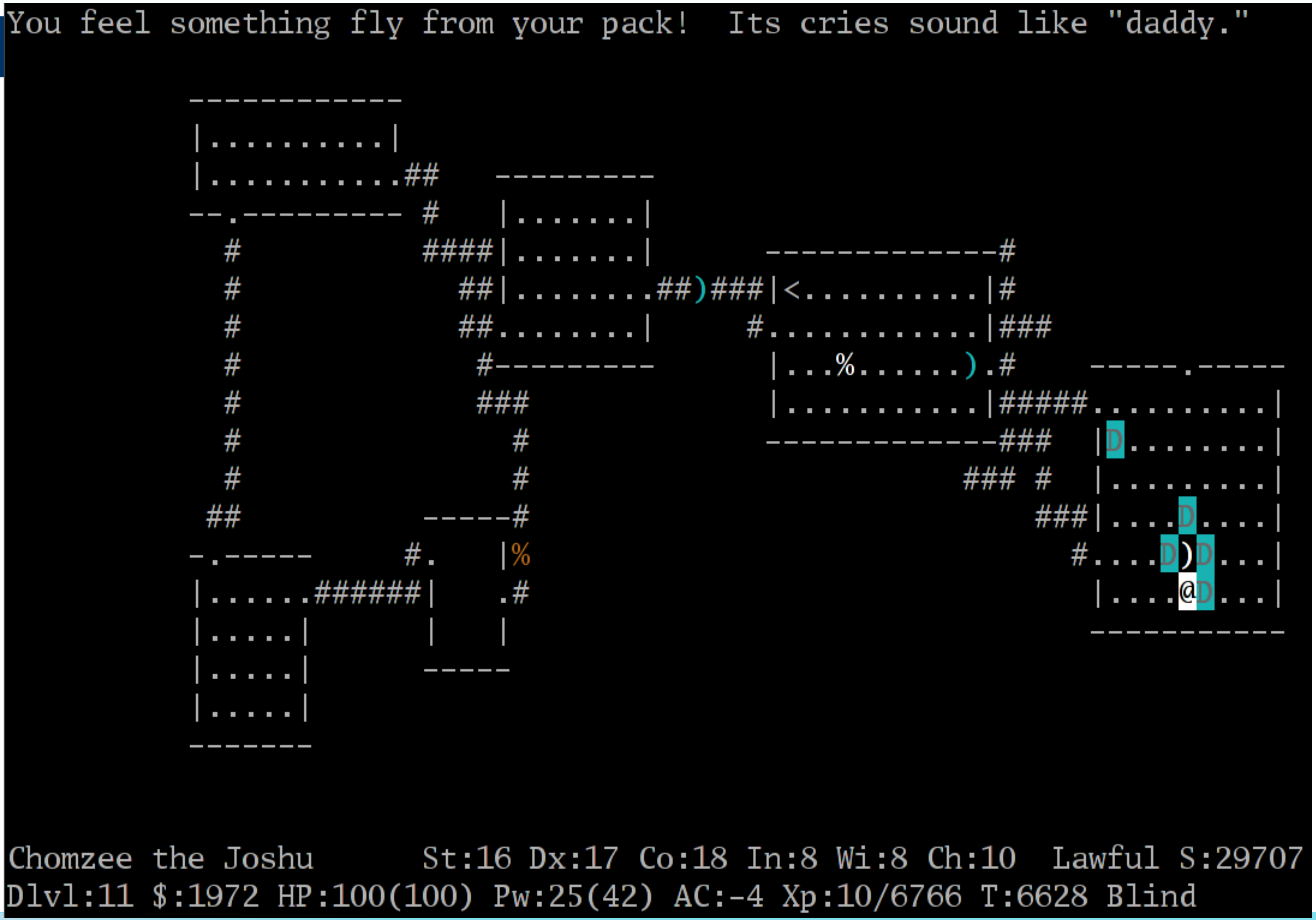
```
<<<                                     Operating System - Google (p1 of 6)
   Gmail »

   | |

   [logo_sm.gif] Operating System__________________________    Submit


   Operating System 519,000,000 1-10 0.11

  Operating system - Wikipedia, the free encyclopedia

   An operating system (OS) is a set of programs that manage computer
   hardware
   resources and provide common services for application software. The
   operating ...
   en.wikipedia.org/wiki/Operating_system - 207k - -

   What is operating system? - A Word Definition From the Webopedia ...

   This page describes the term operating system and lists other pages on
(NORMAL LINK) Use right-arrow or <return> to activate.
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Examples from jyy (cont.)



Watch Movies

Game - Nethack



```
You feel something fly from your pack!  Its cries sound like "daddy."

          ------------
          |..........|
          |...........##   ---------
          --.--------- #   |.......|
            #          ####|.......|      -------------#
            #            ##|........##)###|<..........|#
            #            ##........|     #............|###
            #             #---------      |...%......).#    -----.-----
            #             ###             |...........|#####..........|
            #               #             -------------###  |D........|
            #               #                        ### #  |.........|
           ##          -----#                            ###|....D....|
          -.-----     #.   |%                              #....D)D...|
          |......######|   .#                               |....@D...|
          |.....|      |   |                                -----------
          |.....|      -----
          |.....|
          -------


Chomzee the Joshu      St:16 Dx:17 Co:18 In:8 Wi:8 Ch:10  Lawful S:29707
Dlvl:11 $:1972 HP:100(100) Pw:25(42) AC:-4 Xp:10/6766 T:6628 Blind
```

Terminal

- Most part of the work will be conducted under terminal.
- Why use it?
  - keyboard operation is faster than mouse
    - vim v.s. VS
  - provided by a variety of standard tools
  - can do things that GUI can not

Examples

- find how much you have coded

```
find . –name “*.[ch]” | xargs wc
```

- find the definitions of functions which match the pattern “init_…”

```
find . –name “*.c” | xargs grep –nE --color
                    “\binit_.*[^;]$”
```

  - regular expression, google it!

Examples (cont.)

- disassemble an executable file, and save the code into another file

```
objdump -d test > code.txt
```

- see hardware information

```
dmidecode | less
```

  - more details than “lu da shi”

More Examples

- See lab 负1
  - write a script to process a BMP file
- Break the complecated task into small parts;
- Handle each part with appropriate standard tools;
- Use pipe to let different tools work together;
- These are parts of Unix philosophy.

Unix philosophy

- http://en.wikipedia.org/wiki/Unix_philosophy

1. Small is beautiful.
2. Make each program do one thing well.
3. Build a prototype as soon as possible.
4. Choose portability over efficiency.
5. Store data in flat text files.
6. Use software leverage to your advantage.
7. Use shell scripts to increase leverage and portability.
8. Avoid captive user interfaces.
9. Make every program a filter.

Development tools

- use vim/emacs to edit source codes
- use gcc to compile them to executable files
- use qemu to virtualize your OS
- use Makefile to manage the dependency of source codes
- use gdb to debug your OS
- use git to manage the whole project

The MOST important command

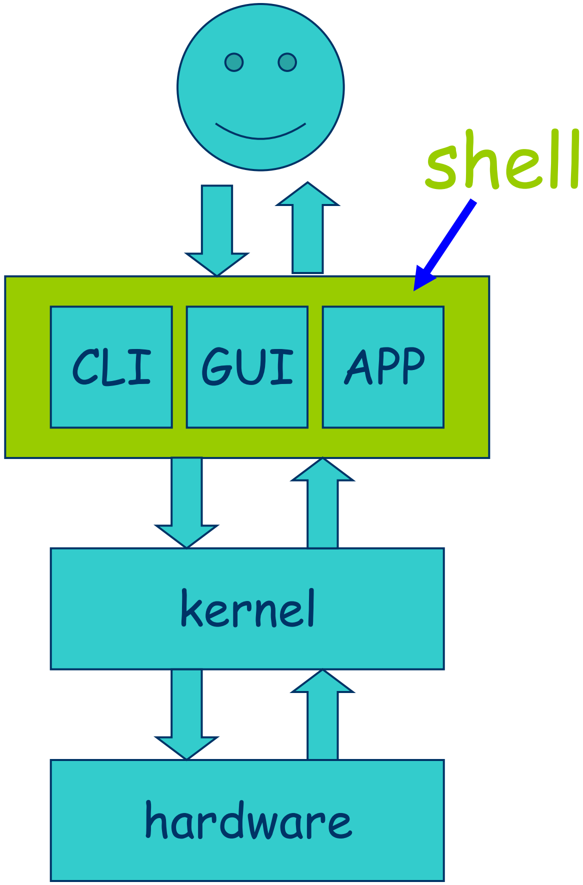
- man
  - Linux manual pager
  - Learn to use ‘man’, learn to use everything.

- 《鸟哥的Linux私房菜》

- Support from hardwards

Hardware

- OS is build over hardwares.
- Hardwares provide
  - registers
  - instruction sets
  - memory management
  - protection
  - interrupts & exceptions
  - peripherals
  - …

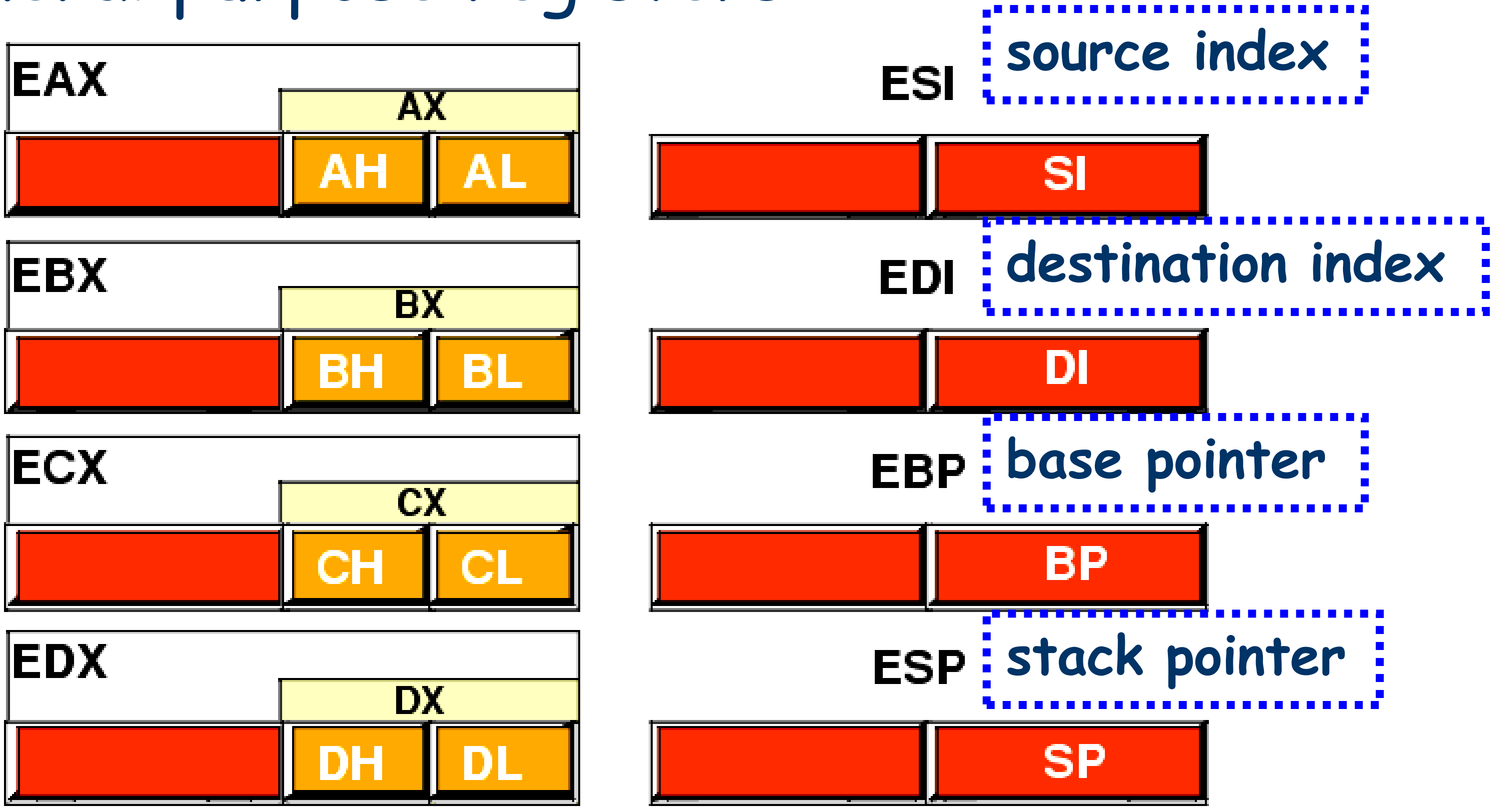


IA-32

- 32-bit Intel architecture
- first implemented on Intel 80386
- Nanos is built over IA-32

Registers - GPRs

- General purpose registers



EAX
AX
AH AL

EBX
BX
BH BL

ECX
CX
CH CL

EDX
DX
DH DL

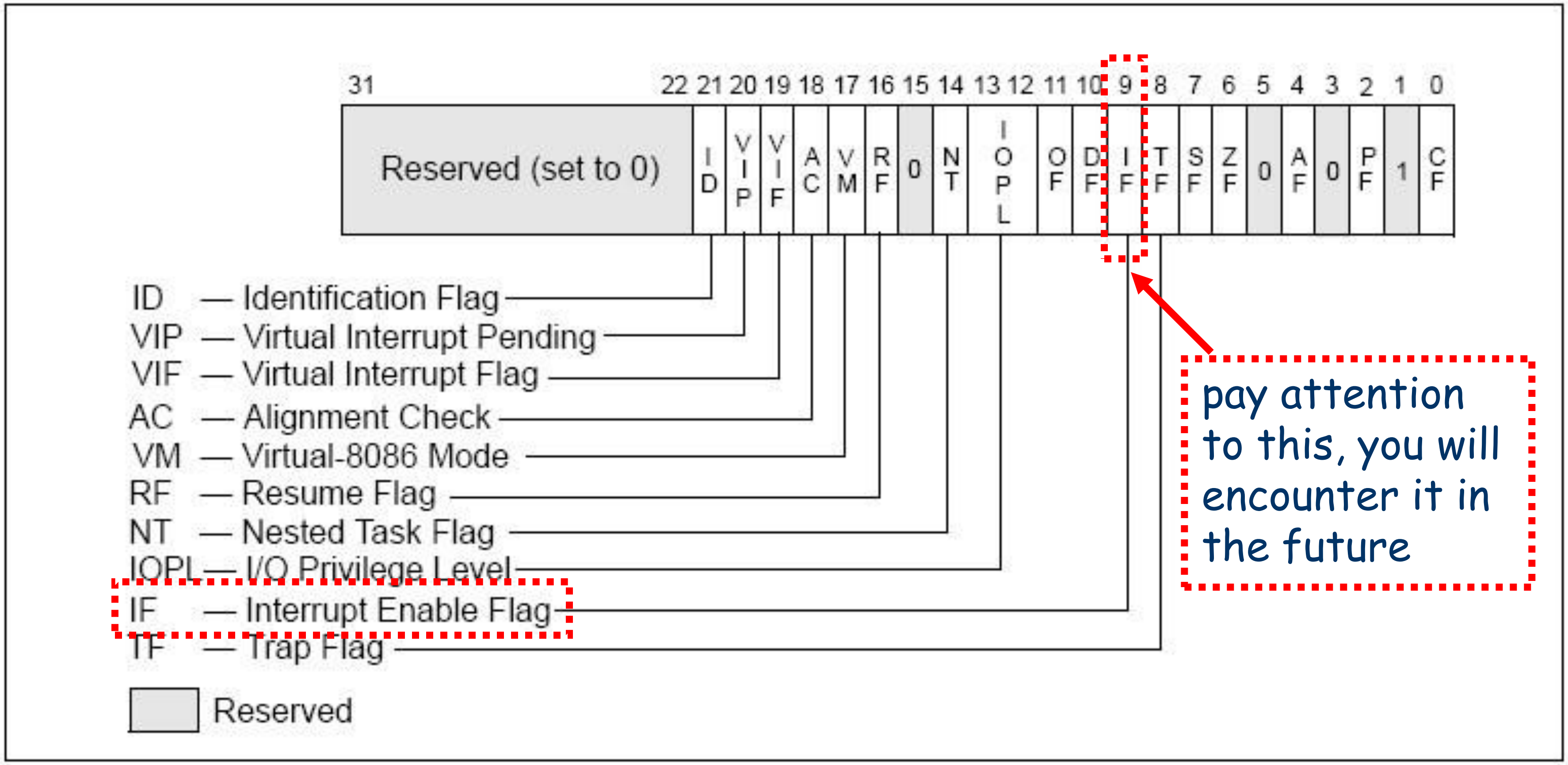
ESI source index
SI

EDI destination index
DI

EBP base pointer
BP

ESP stack pointer
SP

Registers - EFLAGS



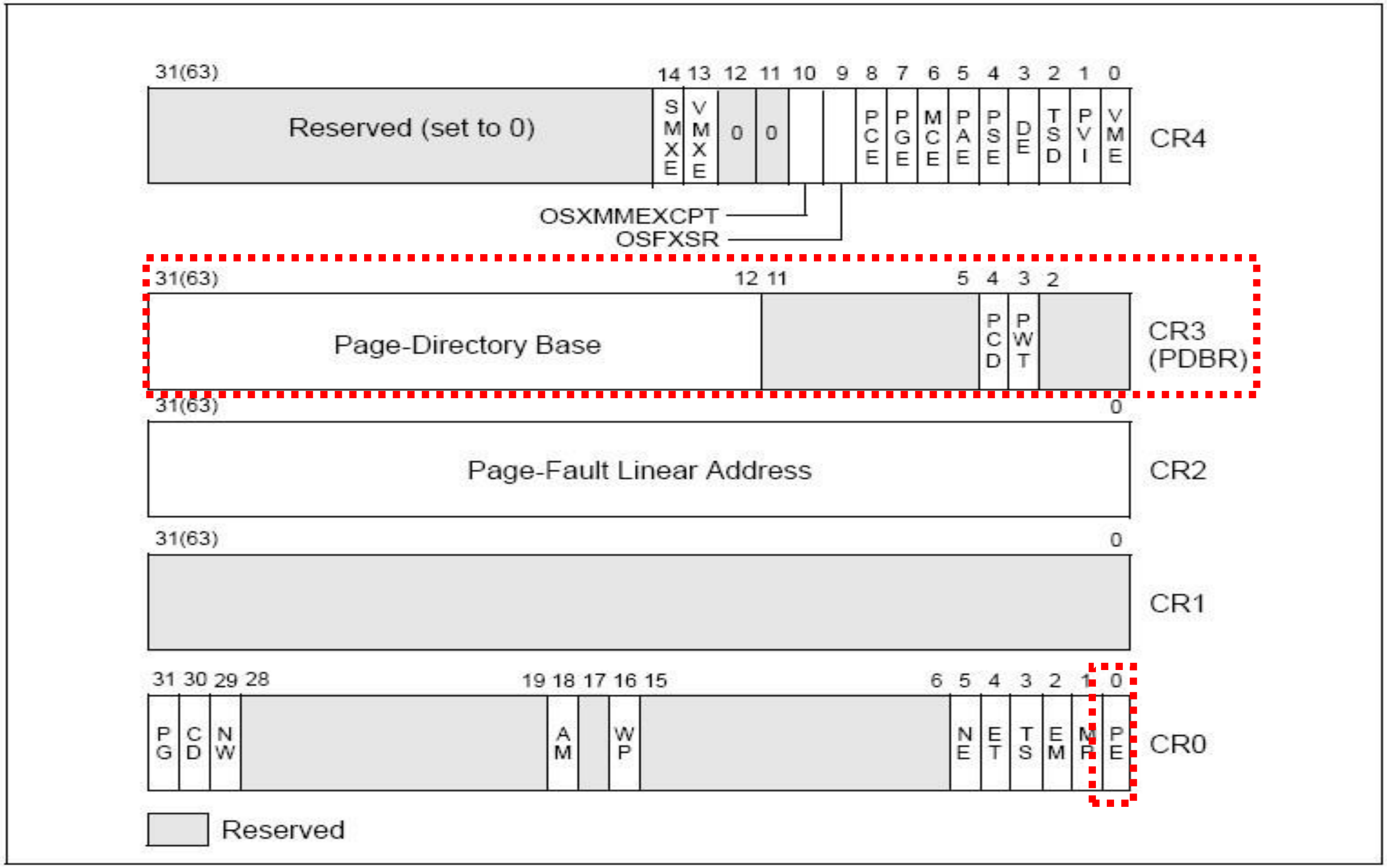
31	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved (set to 0)		ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

ID — Identification Flag
VIP — Virtual Interrupt Pending
VIF — Virtual Interrupt Flag
AC — Alignment Check
VM — Virtual-8086 Mode
RF — Resume Flag
NT — Nested Task Flag
IOPL — I/O Privilege Level
IF — Interrupt Enable Flag
TF — Trap Flag

▨ Reserved

pay attention to this, you will encounter it in the future

Registers - control registers

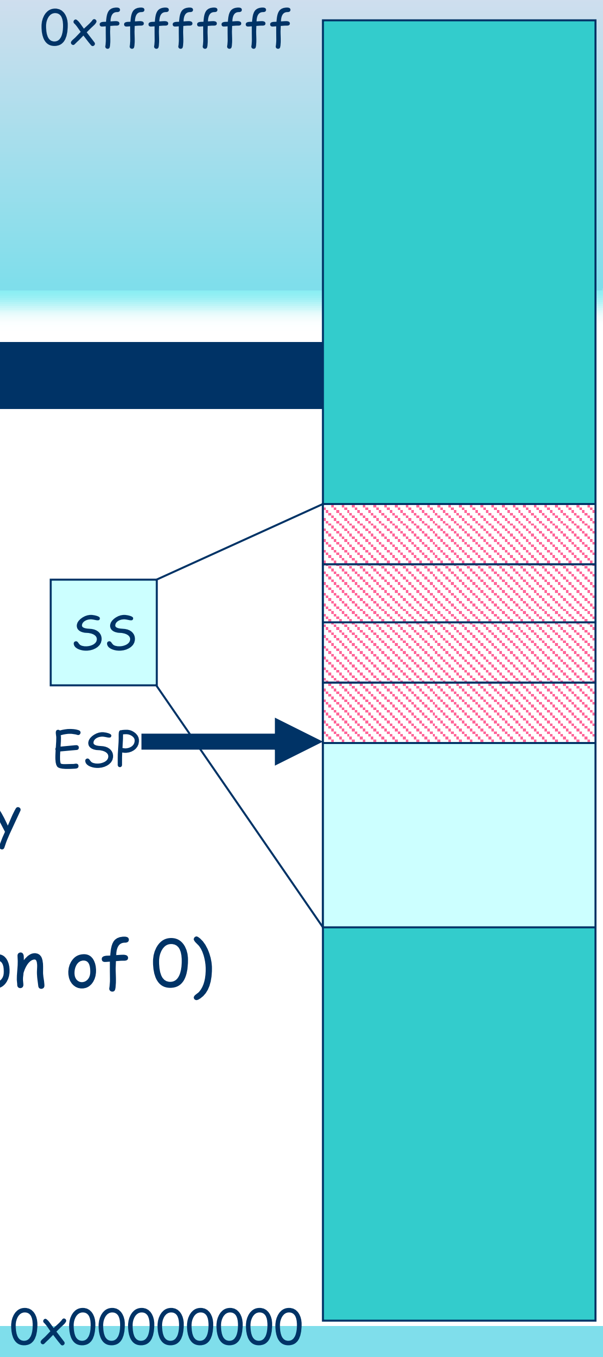


Registers - the rests

- instruction pointer
  - EIP
- segment registers
  - CS (code), DS (data), ES, FS, GS, SS (stack)
- memory-management registers
  - GDTR, IDTR, LDTR, TR
- other
  - DR0 – DR7, TR0, TR1 (we will not use them)
  - ...

Hardware stack

- in data structure
  - stack = array + top pointer
- in hardware
  - stack = SS + ESP
    - SS specifies a segment of memory
    - ESP indicates the top of stack
  - grow downward (to the direction of 0)
  - 4 bytes per elements



Hardware stack (cont.)

- operations – push, pop

```
pushl %eax
```

  - It is identical to

```
subl $4, %esp
movl %eax, (%esp)
```

  - ===

```
popl %ebx
```

  - It is identical to

```
movl (%esp), %ebx
addl $4, %esp
```

Recap

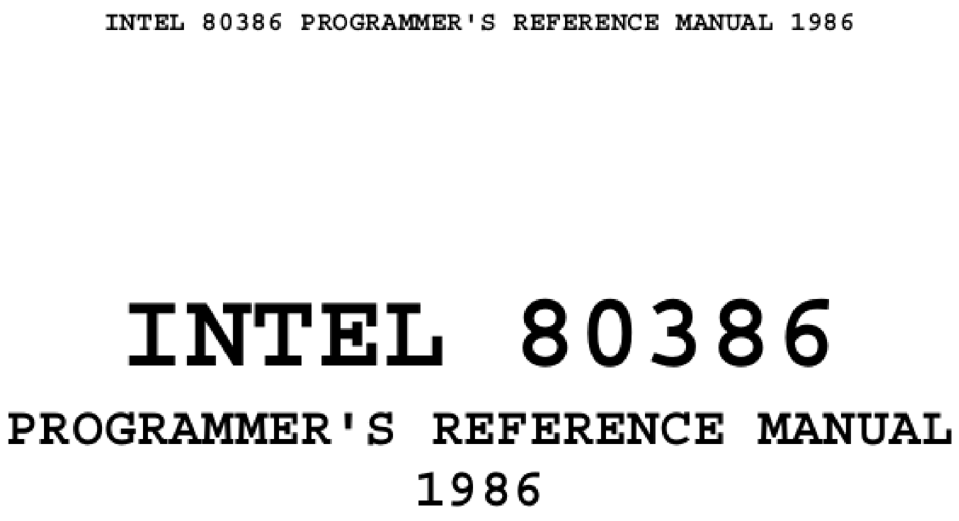
- assembly language
- calling convention
  - you are asked to implement printk() in Lab0
- IA-32 memory management
  - GDT
  - segment descriptor
  - segment selector
  - paging (although not occur in Lab0)

Prepare yourself

- You should know something about the following topics within IA-32
  - registers
  - instruction sets
  - memory management
  - protection
  - interrupts (the topic of next week)
- Some behavors of OS may be “mysterious” without enough understand of the topics above.

Reference book

- Make reference to INTEL 80386 PROGRAMMER'S REFERENCE MANUAL for ANY trouble about hardwares.



INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986

INTEL 80386

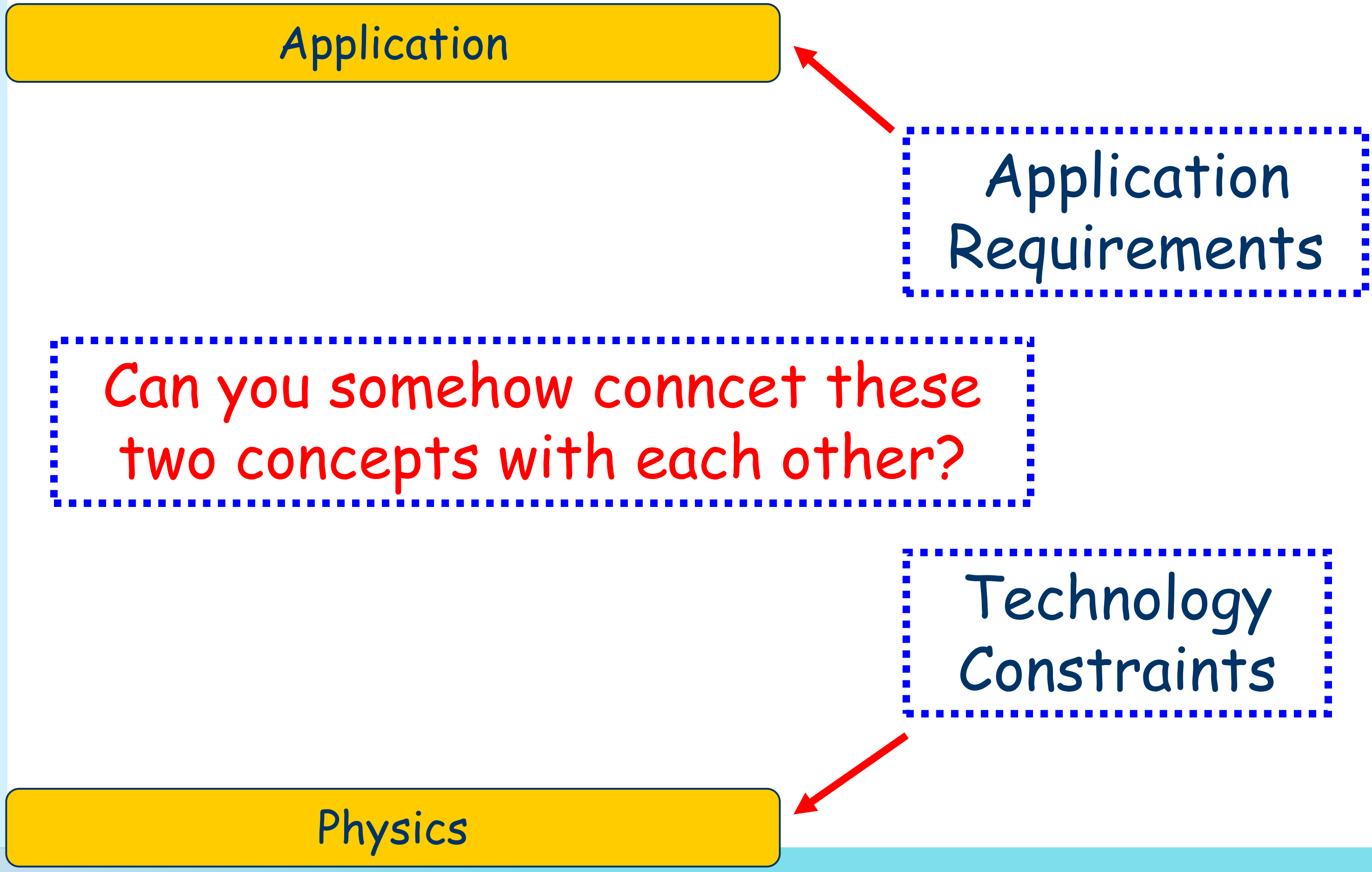
PROGRAMMER'S REFERENCE MANUAL

1986

- What is an OS?
  - from computer's view

Abstraction

- from Computer Architecture, Princeton University



Application

Application Requirements

Can you somehow conncet these two concepts with each other?

Technology Constraints

Physics

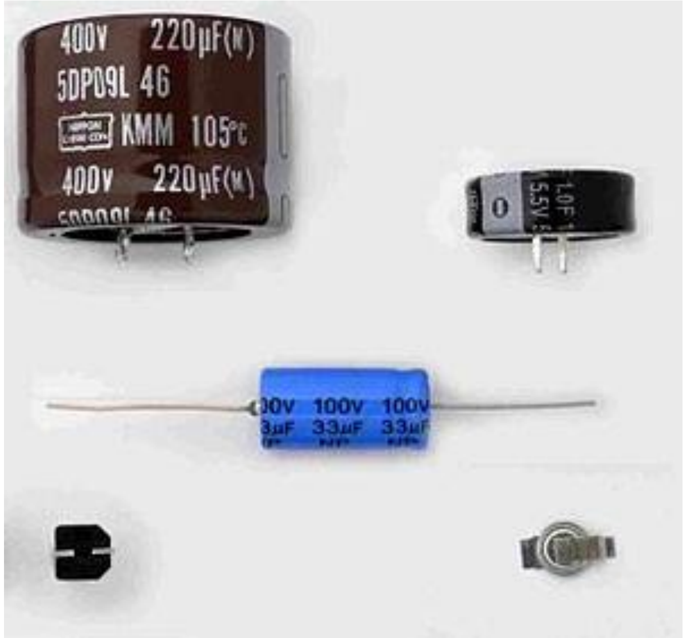
Abstraction
- from Computer Architecture, Princeton University

Application

Physics

Abstraction
- from Computer Architecture, Princeton University

Application

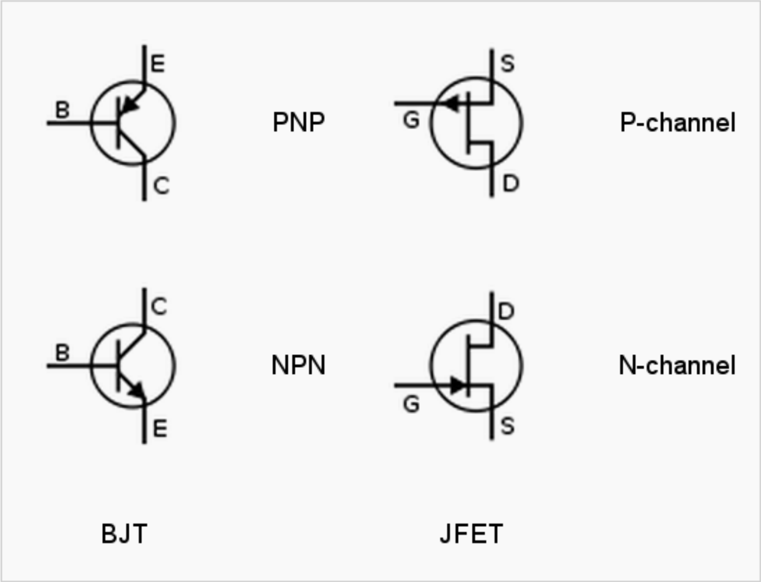


Devices

Physics

Abstraction
- from Computer Architecture, Princeton University

Application



Circuits

Devices

Physics

Abstraction

- from Computer Architecture, Princeton University

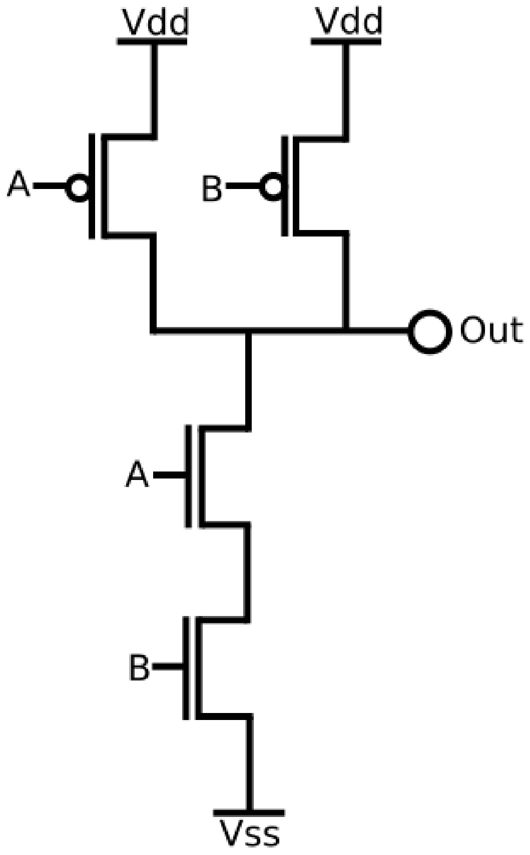
Application

Gates

Circuits

Devices

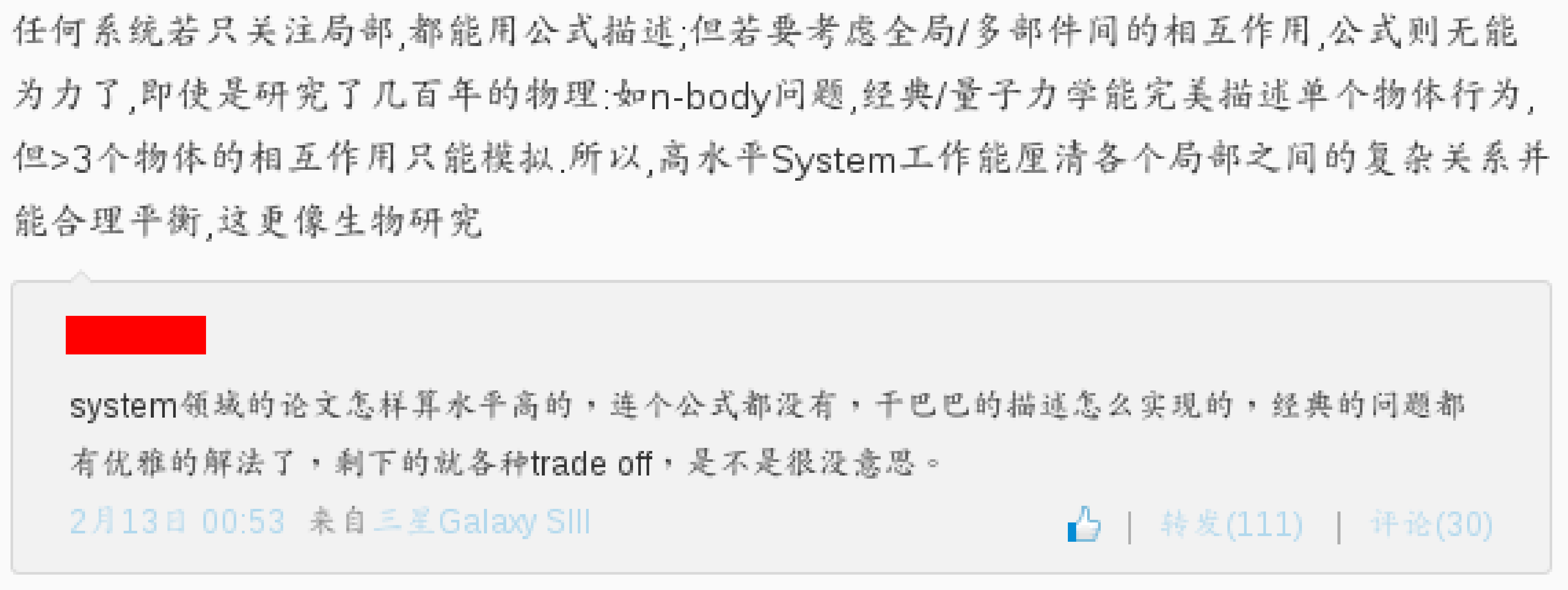
Physics



jyy's question

- Give you enough NAND gates and enough time, can you build a moderm computer which can function correctly?
- Why not?
  - unfamiliar with concepts
  - unfamiliar with implementations
- This is exactly what computer architecture does
  - performace will be considered

System - put everything together



任何系统若只关注局部,都能用公式描述;但若要考虑全局/多部件间的相互作用,公式则无能为力了,即使是研究了几百年的物理:如n-body问题,经典/量子力学能完美描述单个物体行为,但>3个物体的相互作用只能模拟.所以,高水平System工作能厘清各个局部之间的复杂关系并能合理平衡,这更像生物研究

[redacted]

system领域的论文怎样算水平高的，连个公式都没有，干巴巴的描述怎么实现的，经典的问题都有优雅的解法了，剩下的就各种trade off，是不是很没意思。

2月13日 00:53 来自三星Galaxy SIII

Abstraction

- from Computer Architecture, Princeton University

Application

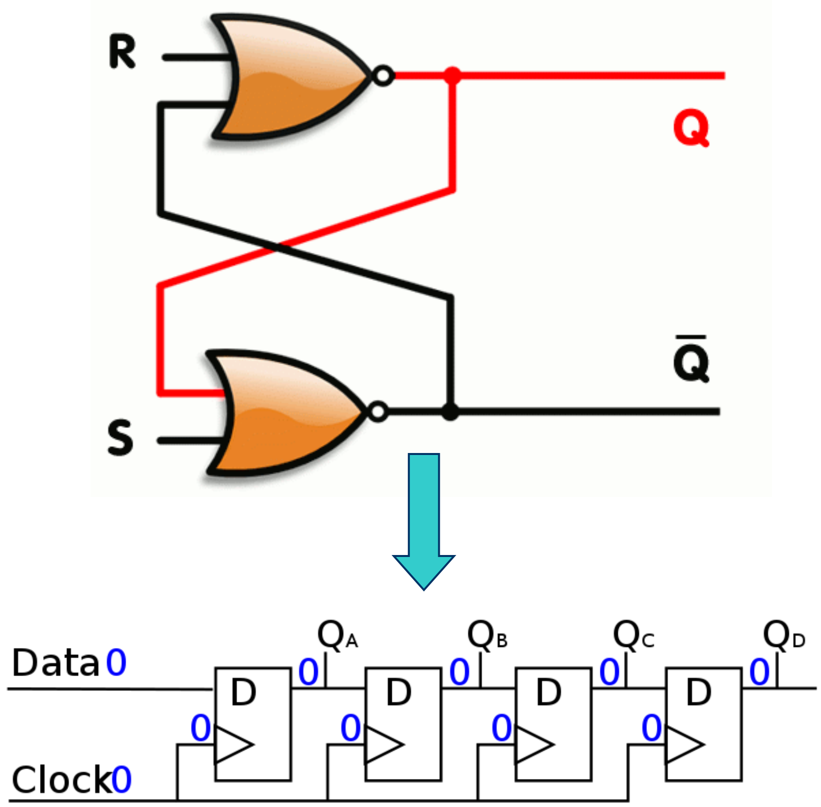
Register-Transfer Level

Gates

Circuits

Devices

Physics



Abstraction

- from Computer Architecture, Princeton University

Application

Micro-architecture

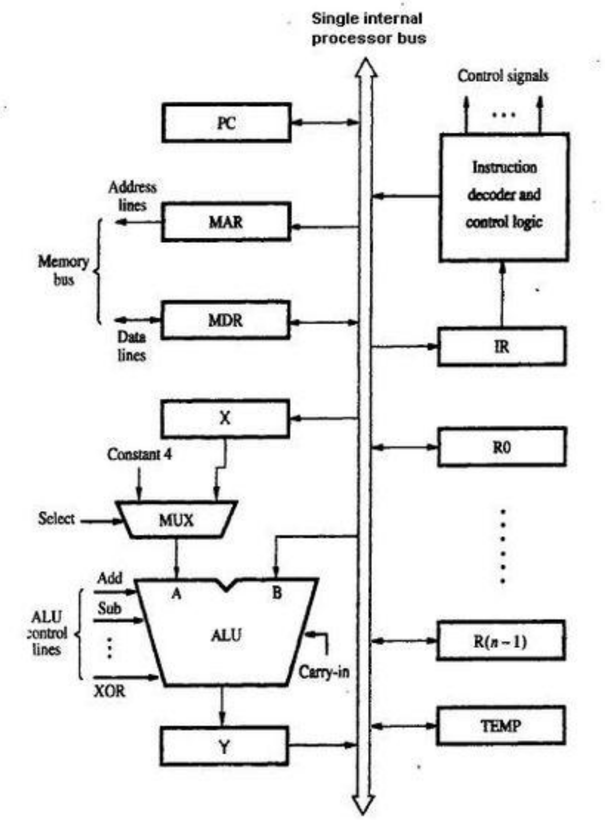
Register-Transfer Level

Gates

Circuits

Devices

Physics

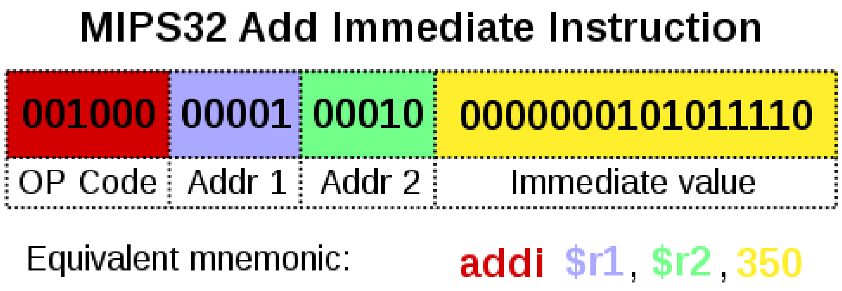


Single Bus Organization

Abstraction

- from Computer Architecture, Princeton University

- Application
- Instruction Set Architecture
- Micro-architecture
- Register-Transfer Level
- Gates
- Circuits
- Devices
- Physics



MIPS32 Add Immediate Instruction

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	Immediate value

Equivalent mnemonic: addi $r1, $r2, 350

Abstraction

- from Computer Architecture, Princeton University

Application

Operating System/Virtual Machines

Instruction Set Architecture

Micro-architecture

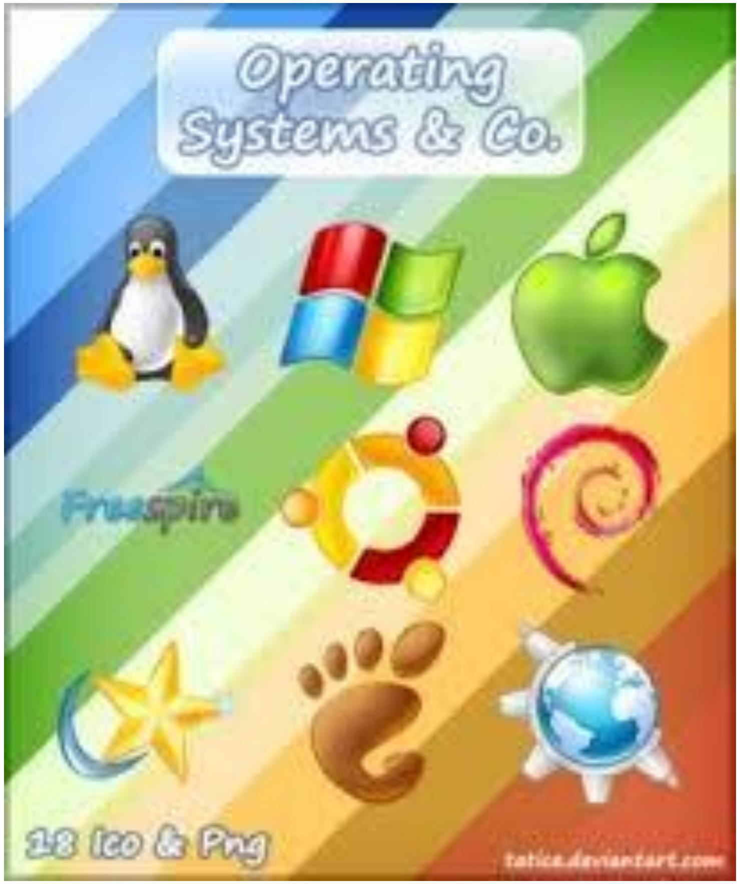
Register-Transfer Level

Gates

Circuits

Devices

Physics



Abstraction

- from Computer Architecture, Princeton University

Application

Programming Language

Operating System/Virtual Machines

Instruction Set Architecture

Micro-architecture

Register-Transfer Level

Gates

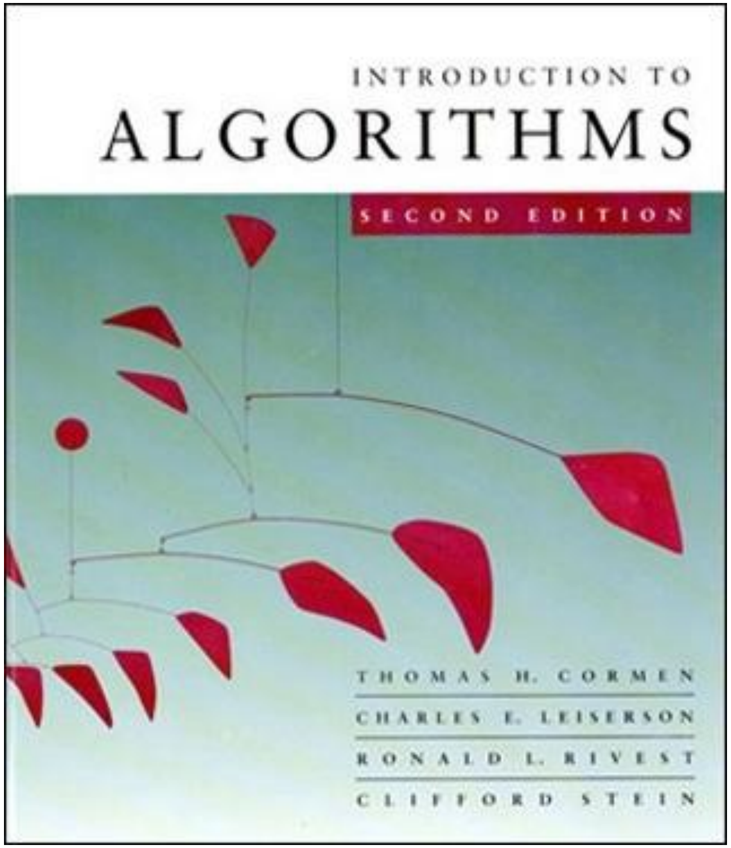
Circuits

Devices

Physics

Abstraction
- from Computer Architecture, Princeton University

- Application
- Algorithm
- Programming Language
- Operating System/Virtual Machines
- Instruction Set Architecture
- Micro-architecture
- Register-Transfer Level
- Gates
- Circuits
- Devices
- Physics


INTRODUCTION TO

ALGORITHMS

SECOND EDITION

THOMAS H. CORMEN

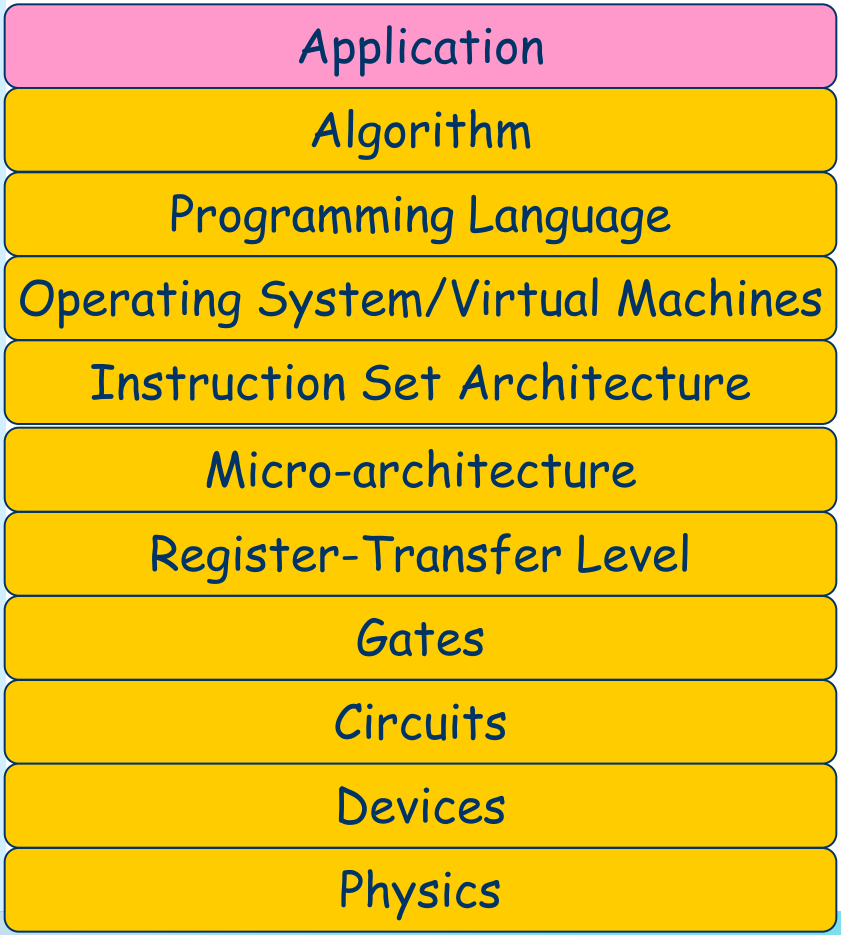
CHARLES E. LEISERSON

RONALD L. RIVEST

CLIFFORD STEIN

Abstraction

- from Computer Architecture, Princeton University



Application

Algorithm

Programming Language

Operating System/Virtual Machines

Instruction Set Architecture

Micro-architecture

Register-Transfer Level

Gates

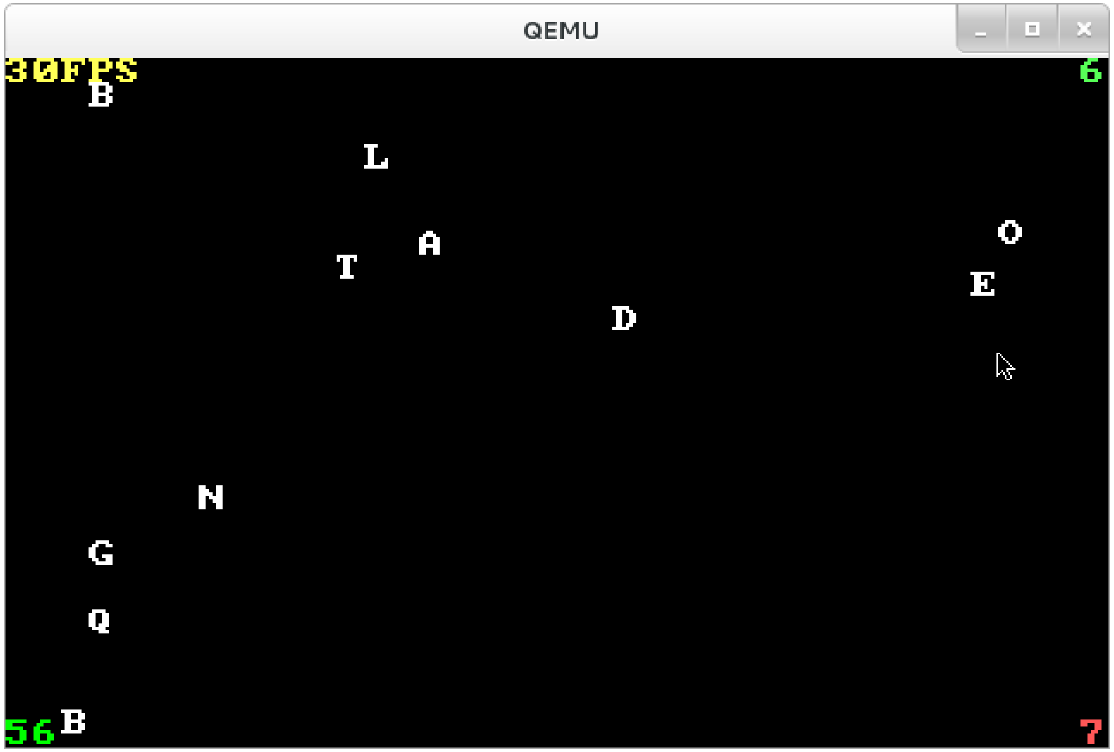
Circuits

Devices

Physics

That is the world of computer system!

- But is OS necessary?



What is an OS?

- Programs can still function without OS.
- But why we need OS?
- Without OS, what will the world of computer be? – Guide Question 1

What is an OS? (cont.)

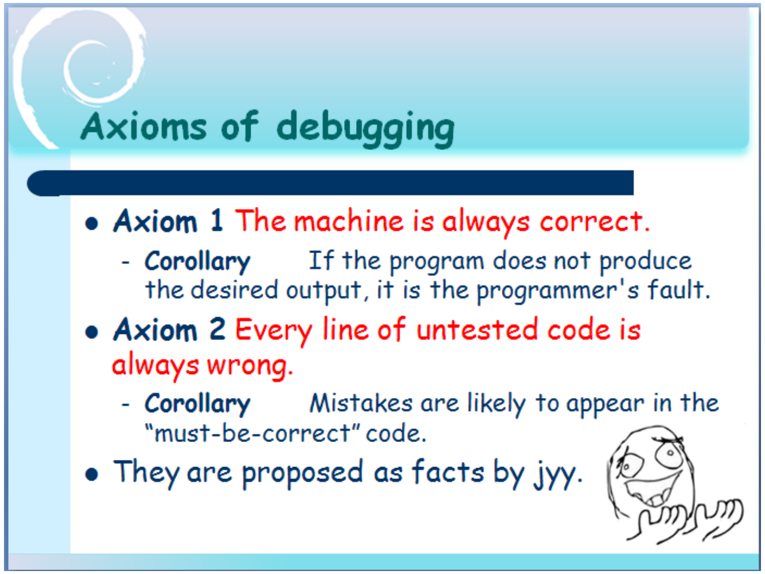
- When a program runs over OS, how do they connect with each other?
- After you issue command to run a "hello world" program, what happen to the OS exactly? – Guide Question 2
- Try to answer two guide questions
  - now
  - at the end of this course

More questions

- 旷日持久的计算机教学只为解答三个问题：
  - (theory, 理论计算机科学) 什么是计算？
  - (system, 计算机系统) 什么是计算机？
  - (application, 计算机应用) 我们能用计算机做什么？
- What is your future?
- What is the difference between you and a student graduating from “Lan Xiang”?

Anyway, enjoy the journey

- Lab0 is out!
- Now enjoy your journey to OS labs!
- But remember:



Axioms of debugging

- **Axiom 1** The machine is always correct.
  - **Corollary** If the program does not produce the desired output, it is the programmer's fault.
- **Axiom 2** Every line of untested code is always wrong.
  - **Corollary** Mistakes are likely to appear in the "must-be-correct" code.
- They are proposed as facts by jyy.