بسم الله الرحمن الرحیم

سیستم عامل

جلسه اول - معرفی درس

Original slides by: Jonathan Walpole & Hadi Foroughmand Arabi

معرفی خودم

- Alireza Tofighi
- Ph.D Student at Sharif, CTO at Balad.ir
- Email: alirtofighim+os1403@gmail.com
- https://alireza.dev/teaching/os1403/
- **مشاوره و رفع اشکال؟**
  - *به من ایمیل بزنید، ممکن است در اتاق ۱۱۴ دانشکده یا لابی پیدایم کنید!*

Class Goals

- Understand the basic concepts of operating systems
  - *designing & building operating systems, not using them!*
- Gain some practical experience so that it is not just words!

قوانین درس

تمرین‌های برنامه‌نویسی

- ۵ سری تمرین عملی
- با زبان KPL، باید مستندات زبان را هم بخوانید که یاد بگیرید و استفاده کنید. (زبانِ سختی نیست :دی)
- شروع تمرین اول از امروز! و مهلت تا پایان ۲۰ مهر
- قوانین تاخیر؟
  - *در مجموع ۱۰ روز تاخیر مجاز، ولی هر تمرین حداکثر ۳ روز تاخیر مجاز*
  - *محاسبه‌ی تاخیر به شکل ساعتی*
  - *تاخیر بیش از زمان => نمره‌ای کسب نمی‌شود.*

Programming Projects

- You will read, understand and write operating system code!
- We will use the BLITZ system, written by Harry Porter
  - *CPU emulator, assembler, high-level language, operating system, and debugging environment*
  - *- Simple enough to understand in detail how everything works!*
  - *- Realistic enough to understand in detail how everything works!*
  - *- Runs on the most modern machines*

آزمونک!

- تلاش برای دنبال کردن بخش تئوری درس
- ۵ سری
- به شکل حضوری – کتاب بسته و در پایان کلاس‌ها
- زمان آن از قبل اعلام می‌شود.

اصل مطلب – بارم‌بندی

- **۵ سری تمرین عملی و برنامه‌نویسی** - مجموعاً ۶ نمره
- **۵ سری آزمونک** - مجموعاً ۳ نمره
- **آزمون میان‌ترم** - ۵ نمره
- **آزمون پایان‌ترم** - ۷ نمره

جنگ اول به از صلح آخر – تقلب/مشابهت

- انتظار داریم تمام بخش‌های نمره‌دار درس تلاش مستقیم خودتان باشد.
- مثلا در تمرین‌های عملی، این موارد مجاز نیست: (موارد مشابه هم مجاز نیست)
  - *استفاده از کد هم‌کلاسی‌هایتان یا هر فرد دیگر*
  - *استفاده از ابزارهای تولید کد همچون GPTها*
  - *استفاده از کدهای موجود در اینترنت*
  - *دیدن کد هم‌کلاسی یا شنیدن آن*
- چرا سخت‌گیری؟
- وجود مشابهت => **احتمال صفر گرفتن آن تمرین یا کل تمرینات درس یا افتادن درس**! به تشخیص تیم درس 👿

تیم درس

- علیرضا توفیقی محمدی - استاد درس
alirtofighim+os1403@gmail.com
- الهه توحیدی - سردستیار درس
elahetohidi2017@gmail.com
- امیرسالار صفایی قادری - دستیار درس
- سید ستار لطفی فاطمی - دستیار درس
- سوگل زمانیان - دستیار درس
- امیر طایفه - دستیار درس
- رابعه پرهیزکاری - دستیار درس
- وحید انصاری - دستیار درس
- عرفان موسویان - دستیار درس

اطلاعات درس

https://alireza.dev/teaching/os1403/

تقویم درس در سایت درس

alireza.dev/teaching/os1403/

این تقویم در طول ترم بروز خواهد شد.

جلسه	تاریخ	موضوع جلسه	آزمونک	تمرین
۱	۰۳/۰۷/۰۲	معرفی درس - شروع		تمرین ۱ آشنایی با ساختار پروژه
۲	۰۳/۰۷/۰۷	معماری کامپیوتر		
۳	۰۳/۰۷/۰۹	پردازه		
۴	۰۳/۰۷/۱۴	پردازه		
۵	۰۳/۰۷/۱۶	ریسمان	آزمونک ۱ معماری کامپیوتر - پردازه	
۶	۰۳/۰۷/۲۱	همروندی		تمرین ۲ همروندی
۷	۰۳/۰۷/۲۳	همروندی		
۸	۰۳/۰۷/۲۸	همروندی		
۹	۰۳/۰۷/۳۰	بن‌بست	آزمونک ۲ همروندی	
۱۰	۰۳/۰۸/۰۵	بن‌بست		
۱۱	۰۳/۰۸/۰۷	بن‌بست		
۱۲	۰۳/۰۸/۱۲	زمان‌بندی		تمرین ۳ همروندی - ناحیه بحرانی
۱۳	۰۳/۰۸/۱۴	مدیریت حافظه		
۱۴	۰۳/۰۸/۱۹	مدیریت حافظه		

کوئرای درس

رمز کلاس: os1403

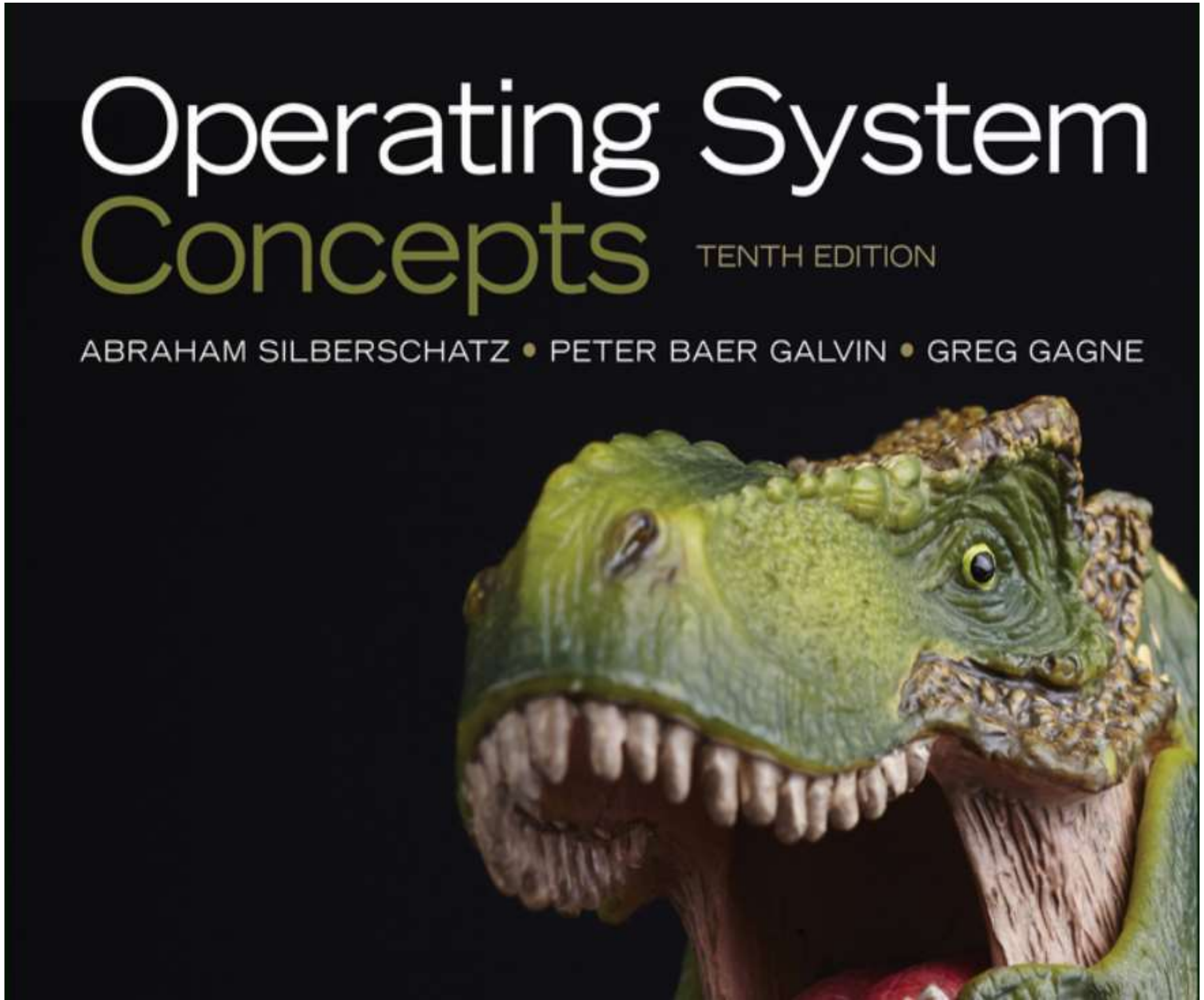
گروه تلگرامی

تیم درس رو می‌تونید در تلگرام هم بیابید.

لطفا ولی به من در PV تلگرام برای کارهای درس پیام ندید :D

QUESTIONS?

معرفی منبع درس



PART ONE ■ OVERVIEW

Chapter 1 Introduction

1.1 What Operating Systems Do 4
1.2 Computer-System Organization 7
1.3 Computer-System Architecture 15
1.4 Operating-System Operations 21
1.5 Resource Management 27
1.6 Security and Protection 33
1.7 Virtualization 34
1.8 Distributed Systems 35
1.9 Kernel Data Structures 36
1.10 Computing Environments 40
1.11 Free and Open-Source Operating Systems 46
Practice Exercises 53
Further Reading 54

Chapter 2 Operating-System Structures

2.1 Operating-System Services 55
2.2 User and Operating-System Interface 58
2.3 System Calls 62
2.4 System Services 74
2.5 Linkers and Loaders 75
2.6 Why Applications Are Operating-System Specific 77
2.7 Operating-System Design and Implementation 79
2.8 Operating-System Structure 81
2.9 Building and Booting an Operating System 92
2.10 Operating-System Debugging 95
2.11 Summary 100
Practice Exercises 101
Further Reading 101

منبع درس

منبع درس

PART TWO ■ PROCESS MANAGEMENT

Chapter 3 Processes

3.1 Process Concept 106
3.2 Process Scheduling 110
3.3 Operations on Processes 116
3.4 Interprocess Communication 123
3.5 IPC in Shared-Memory Systems 125
3.6 IPC in Message-Passing Systems 127
3.7 Examples of IPC Systems 132
3.8 Communication in Client–Server Systems 145
3.9 Summary 153
Practice Exercises 154
Further Reading 156

xxi

Contents

Chapter 4 Threads & Concurrency

4.1 Overview 160
4.2 Multicore Programming 162
4.3 Multithreading Models 166
4.4 Thread Libraries 168
4.5 Implicit Threading 176
4.6 Threading Issues 188
4.7 Operating-System Examples 194
4.8 Summary 196
Practice Exercises 197
Further Reading 198

منبع درس

Chapter 5 CPU Scheduling

5.1 Basic Concepts 200
5.2 Scheduling Criteria 204
5.3 Scheduling Algorithms 205
5.4 Thread Scheduling 217
5.5 Multi-Processor Scheduling 220
5.6 Real-Time CPU Scheduling 227
5.7 Operating-System Examples 234
5.8 Algorithm Evaluation 244
5.9 Summary 250
Practice Exercises 251
Further Reading 254

منبع درس

PART THREE ■ PROCESS SYNCHRONIZATION

Chapter 6 Synchronization Tools

6.1 Background 257
6.2 The Critical-Section Problem 260
6.3 Peterson's Solution 262
6.4 Hardware Support for Synchronization 265
6.5 Mutex Locks 270
6.6 Semaphores 272
6.7 Monitors 276
6.8 Liveness 283
6.9 Evaluation 284
6.10 Summary 286
Practice Exercises 287
Further Reading 288

Chapter 7 Synchronization Examples

7.1 Classic Problems of Synchronization 289
7.2 Synchronization within the Kernel 295
7.3 POSIX Synchronization 299
7.4 Synchronization in Java 303
7.5 Alternative Approaches 311
7.6 Summary 314
Practice Exercises 314
Further Reading 315

Chapter 8 Deadlocks

8.1 System Model 318
8.2 Deadlock in Multithreaded Applications 319
8.3 Deadlock Characterization 321
8.4 Methods for Handling Deadlocks 326
8.5 Deadlock Prevention 327
8.6 Deadlock Avoidance 330
8.7 Deadlock Detection 337
8.8 Recovery from Deadlock 341
8.9 Summary 343
Practice Exercises 344
Further Reading 346

PART FOUR ■ MEMORY MANAGEMENT

منبع درس

Chapter 9 Main Memory

9.1 Background 349
9.2 Contiguous Memory Allocation 356
9.3 Paging 360
9.4 Structure of the Page Table 371
9.5 Swapping 376
9.6 Example: Intel 32- and 64-bit Architectures 379
9.7 Example: ARMv8 Architecture 383
9.8 Summary 384
Practice Exercises 385
Further Reading 387

Chapter 10 Virtual Memory

10.1 Background 389
10.2 Demand Paging 392
10.3 Copy-on-Write 399
10.4 Page Replacement 401
10.5 Allocation of Frames 413
10.6 Thrashing 419
10.7 Memory Compression 425
10.8 Allocating Kernel Memory 426
10.9 Other Considerations 430
10.10 Operating-System Examples 436
10.11 Summary 440
Practice Exercises 441
Further Reading 444

PART FIVE ■ STORAGE MANAGEMENT

منبع درس

Chapter 11 Mass-Storage Structure

11.1 Overview of Mass-Storage Structure 449
11.2 HDD Scheduling 457
11.3 NVM Scheduling 461
11.4 Error Detection and Correction 462
11.5 Storage Device Management 463
11.6 Swap-Space Management 467
11.7 Storage Attachment 469
11.8 RAID Structure 473
11.9 Summary 485
Practice Exercises 486
Further Reading 487

Chapter 12 I/O Systems

12.1 Overview 489
12.2 I/O Hardware 490
12.3 Application I/O Interface 500
12.4 Kernel I/O Subsystem 508
12.5 Transforming I/O Requests to Hardware Operations 516
12.6 STREAMS 519
12.7 Performance 521
12.8 Summary 524
Practice Exercises 525
Further Reading 526

منبع درس

PART SIX ■ FILE SYSTEM

Chapter 13 File-System Interface

13.1 File Concept 529
13.2 Access Methods 539
13.3 Directory Structure 541
13.4 Protection 550
13.5 Memory-Mapped Files 555
13.6 Summary 560
Practice Exercises 560
Further Reading 561

Chapter 14 File-System Implementation

14.1 File-System Structure 564
14.2 File-System Operations 566
14.3 Directory Implementation 568
14.4 Allocation Methods 570
14.5 Free-Space Management 578
14.6 Efficiency and Performance 582
14.7 Recovery 586
14.8 Example: The WAFL File System 589
14.9 Summary 593
Practice Exercises 594
Further Reading 594

Chapter 15 File-System Internals

15.1 File Systems 597
15.2 File-System Mounting 598
15.3 Partitions and Mounting 601
15.4 File Sharing 602
15.5 Virtual File Systems 603
15.6 Remote File Systems 605
15.7 Consistency Semantics 608
15.8 NFS 610
15.9 Summary 615
Practice Exercises 616
Further Reading 617

PART SEVEN ■ SECURITY AND PROTECTION

منبع درس

Chapter 16 Security

16.1 The Security Problem 621
16.2 Program Threats 625
16.3 System and Network Threats 634
16.4 Cryptography as a Security Tool 637
16.5 User Authentication 648
16.6 Implementing Security Defenses 653
16.7 An Example: Windows 10 662
16.8 Summary 664
Further Reading 665

Chapter 17 Protection

17.1 Goals of Protection 667
17.2 Principles of Protection 668
17.3 Protection Rings 669
17.4 Domain of Protection 671
17.5 Access Matrix 675
17.6 Implementation of the Access Matrix 679
17.7 Revocation of Access Rights 682
17.8 Role-Based Access Control 683
17.9 Mandatory Access Control (MAC) 684
17.10 Capability-Based Systems 685
17.11 Other Protection Improvement Methods 687
17.12 Language-Based Protection 690
17.13 Summary 696
Further Reading 697

PART EIGHT ■ ADVANCED TOPICS

منبع درس

Chapter 18 Virtual Machines

18.1 Overview 701
18.2 History 703
18.3 Benefits and Features 704
18.4 Building Blocks 707
18.5 Types of VMs and Their Implementations 713
18.6 Virtualization and Operating-System Components 719
18.7 Examples 726
18.8 Virtualization Research 728
18.9 Summary 729
Further Reading 730

Chapter 19 Networks and Distributed Systems

19.1 Advantages of Distributed Systems 733
19.2 Network Structure 735
19.3 Communication Structure 738
19.4 Network and Distributed Operating Systems 749
19.5 Design Issues in Distributed Systems 753
19.6 Distributed File Systems 757
19.7 DFS Naming and Transparency 761
19.8 Remote File Access 764
19.9 Final Thoughts on Distributed File Systems 767
19.10 Summary 768
Practice Exercises 769
Further Reading 770

INTRODUCTION TO OPERATING SYSTEMS

1. What is an Operating System?
2. Review of OS-Related Hardware

What is an Operating System?

- “A *program* ... that controls the execution of application programs and implements an interface between the user of a computer and the computer hardware”
- Runs on PCs, workstations, servers, smart phones, routers, embedded systems, etc

Operating System Roles

- Abstract Machine
  - *Hides complex details of the underlying hardware*
  - *Provides common API to applications and services*
  - *Simplifies application writing*

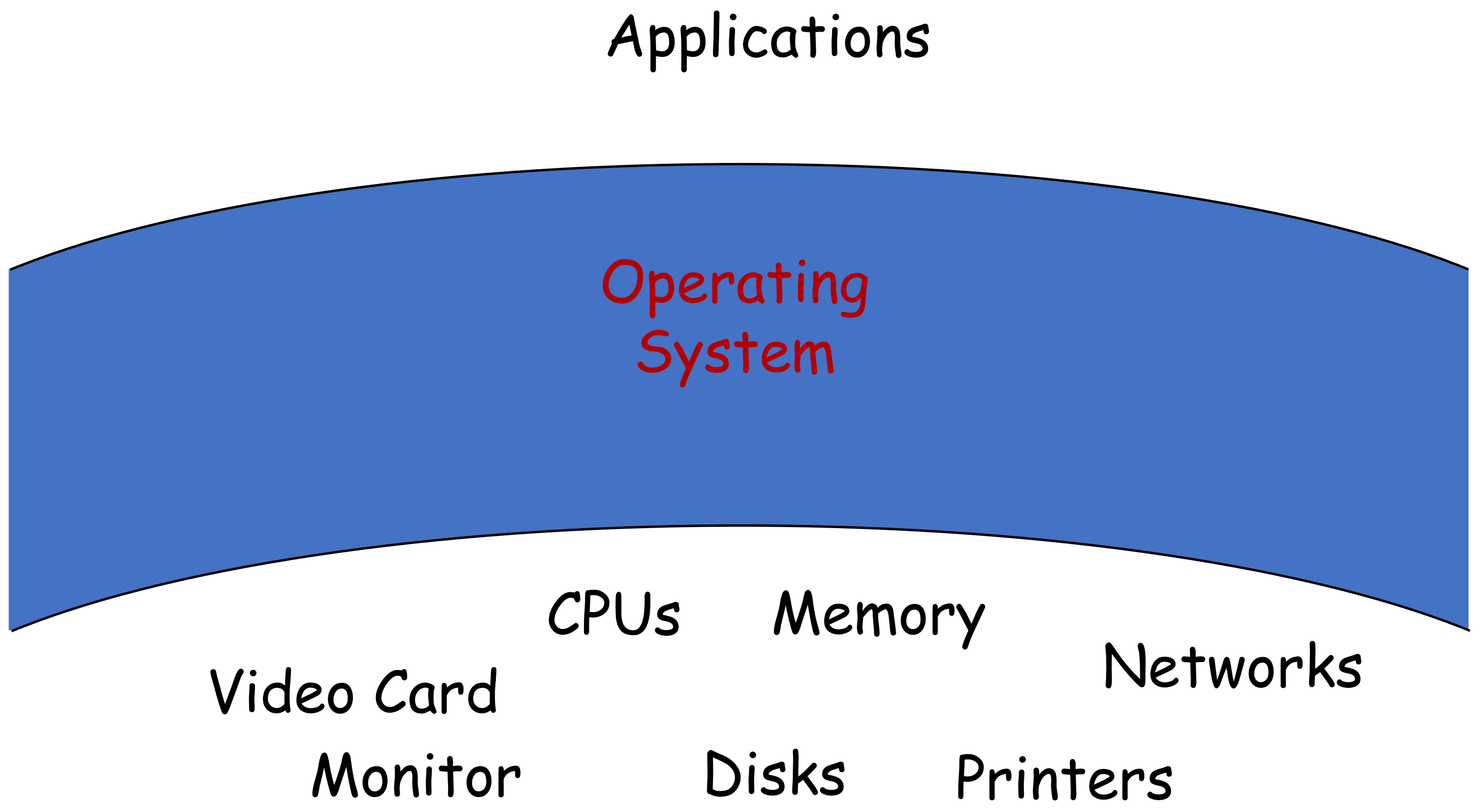
- Resource Manager
  - *Controls accesses to shared resources*
    - CPU, memory, disks, network, ...
  - *Allows for global policies to be implemented*

ABSTRACT MACHINE

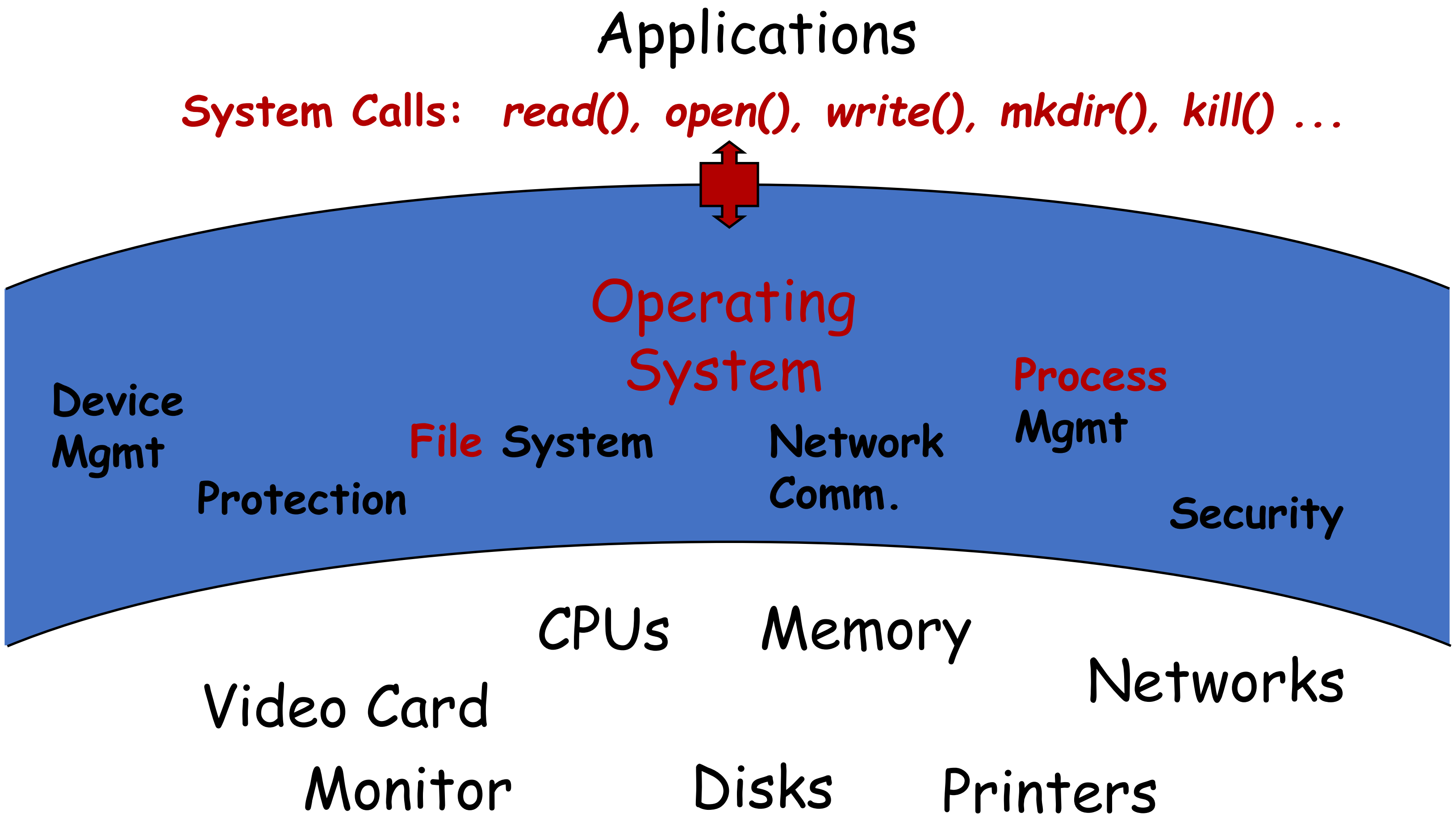
The Abstract Machine Role

- Without operating systems, application writers would have to program all device access directly:
  - *Load device command codes into device registers*
  - *Handle initialization, recalibration, sensing, timing*
  - *Understand physical characteristics and data layout*
  - *Control motors*
  - *Interpret return codes*
  - *...*
- Application programming would be complicated
- Applications would be difficult to maintain, upgrade and port
  - *This OS code could be written just once and then shared!*

Providing abstraction via system calls



Providing abstraction via system calls



RESOURCE MANAGER

OS as a resource manager

- Allocating resources to applications
  - *time sharing resources (scheduling)*
  - *space sharing resources (allocation)*
- Making efficient use of limited resources
  - *improving utilization*
  - *minimizing overhead*
  - *improving throughput/good put*
- Protecting applications from each other
  - *enforcement of boundaries*

Problems Solved by OS

- Time sharing the CPU among applications
- Space sharing the memory among applications
- Space sharing the disk among users
- Time sharing access to the network

More Problems Solved by OS

- Protection of applications from each other, of user data from other users and of I/O devices
- Protection of the OS itself!
- Prevention of direct access to hardware, where this would cause problems

QUESTIONS?

OS NEEDS FROM HARDWARE

OS Needs Help from Hardware

- The OS is just a program!
- When it is not running, it can't do anything!
- Its goal is to run applications, not itself!

- The OS needs help from the hardware in order to detect and prevent certain activities, and to maintain control

Brief Review of Hardware

- Instruction sets define all that a CPU can do, and differ among CPU architectures
- All have load and store instructions to move data between memory and registers
- Many instructions for comparing and combining values in registers

Basic Anatomy of a CPU

- Program Counter (PC)
  - *Holds the memory address of the next instruction*
- Instruction Register (IR)
  - *holds the instruction currently being executed*
- General Registers (Reg. 1..n)
  - *hold variables and temporary results*
- Arithmetic and Logic Unit (ALU)
  - *performs arithmetic functions and logic operations*

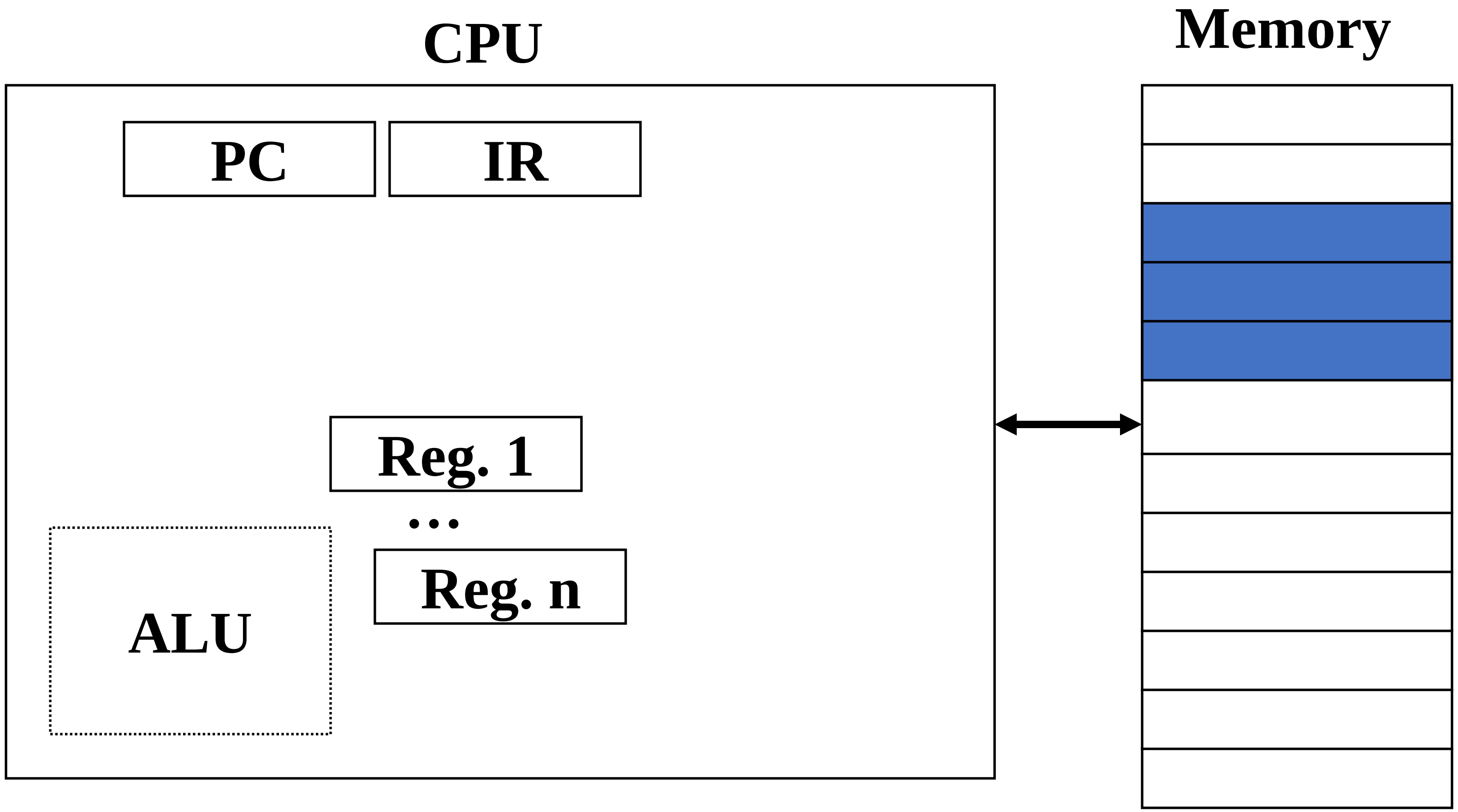
Basic Anatomy of a CPU

- Stack Pointer (SP)
  - *holds memory address of a stack with a frame for each active procedure's parameters & local variables*
- Processor Status Word (PSW)
  - *contains various control bits including the mode bit which determines whether privileged instructions can be executed at this time*

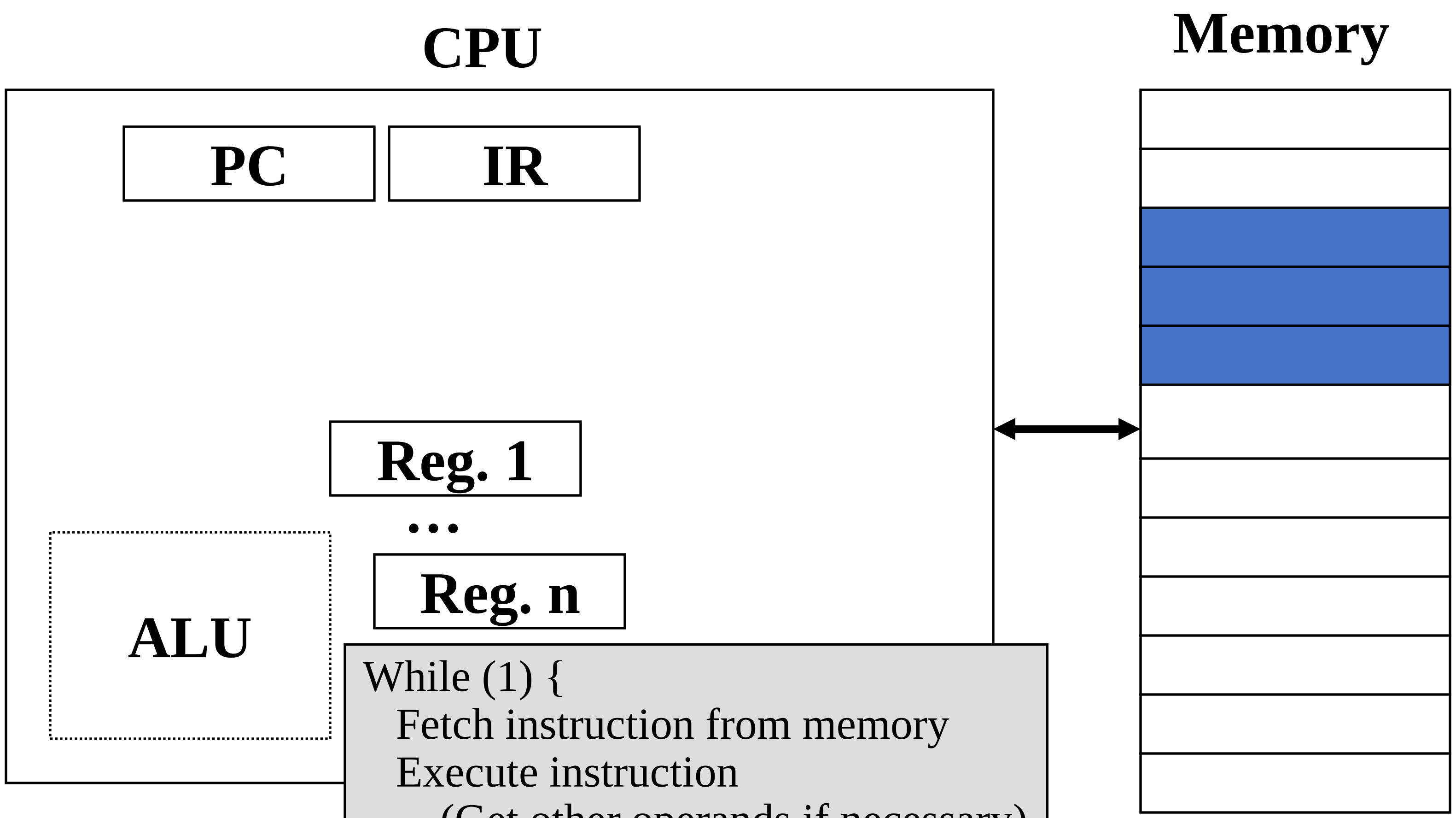
Program Execution

- The Fetch/Decode/Execute cycle
  - *fetch next instruction pointed to by PC*
  - *decode it to find its type and operands*
  - *execute it*
  - *repeat*
- At a fundamental level, this is all a CPU does, regardless of which program it is executing

Fetch/Decode/Execute Cycle

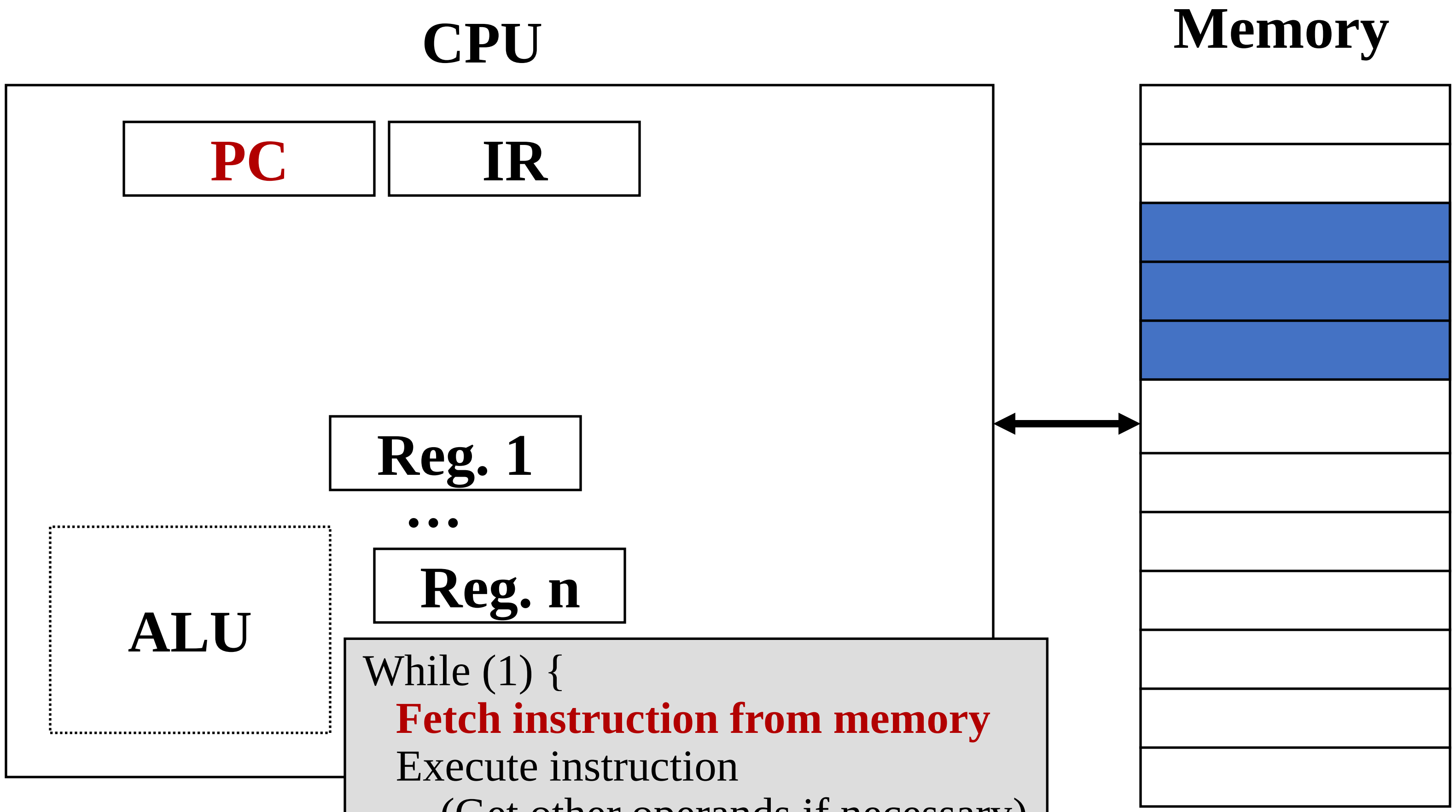


Fetch/Decode/Execute Cycle



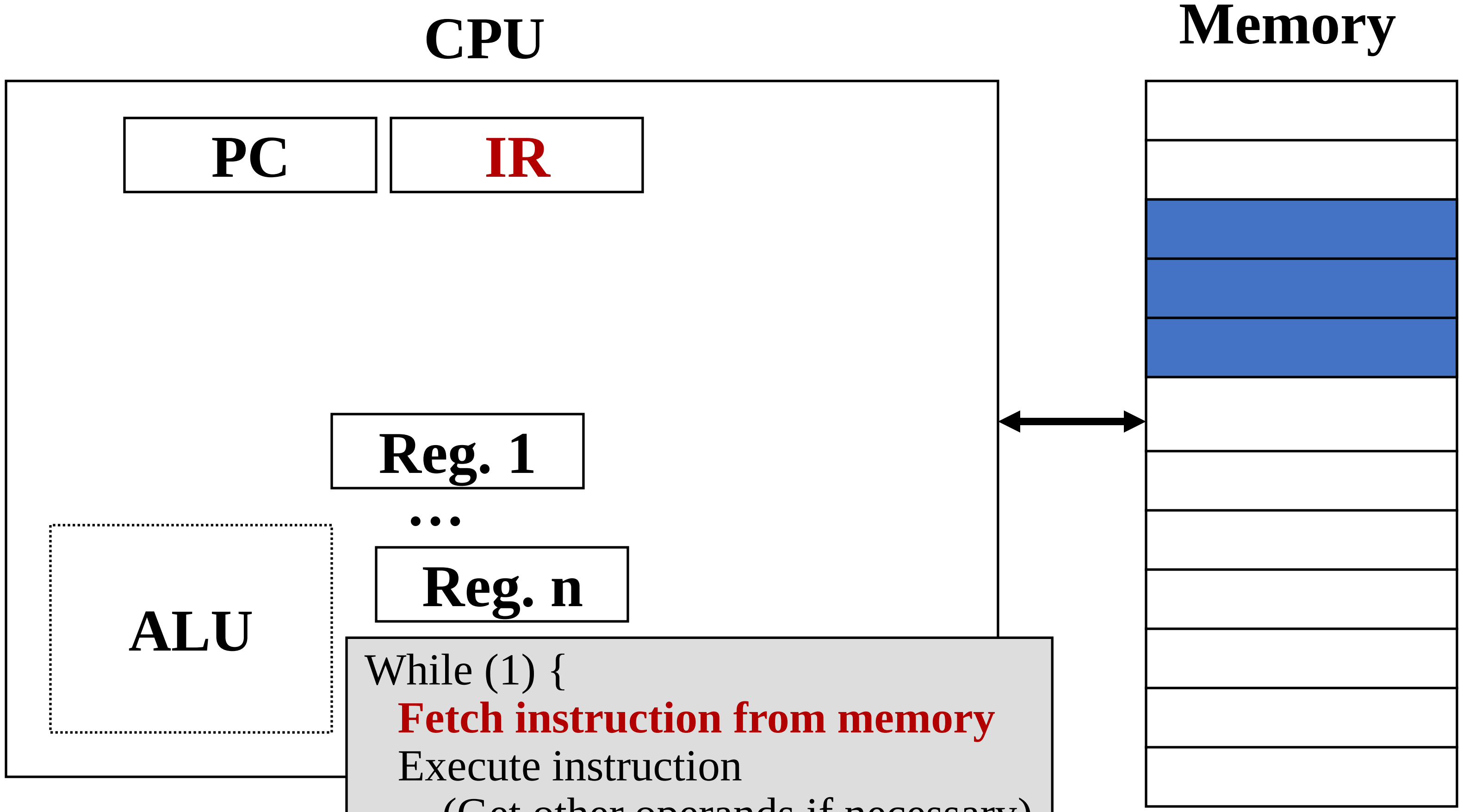

```
While (1) {
   Fetch instruction from memory
   Execute instruction
       (Get other operands if necessary)
   Store result
}
```

Fetch/Decode/Execute Cycle



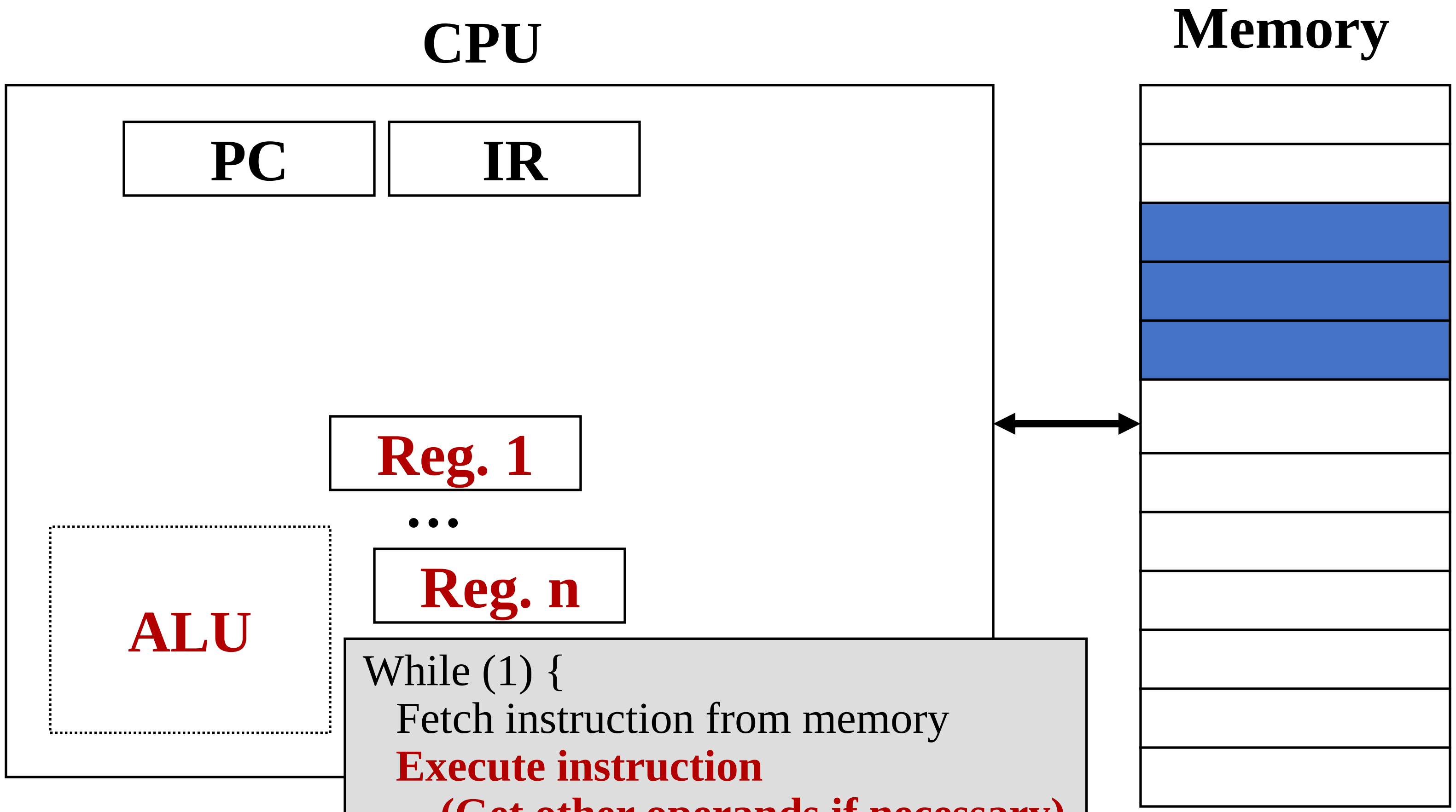

```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/Decode/Execute Cycle



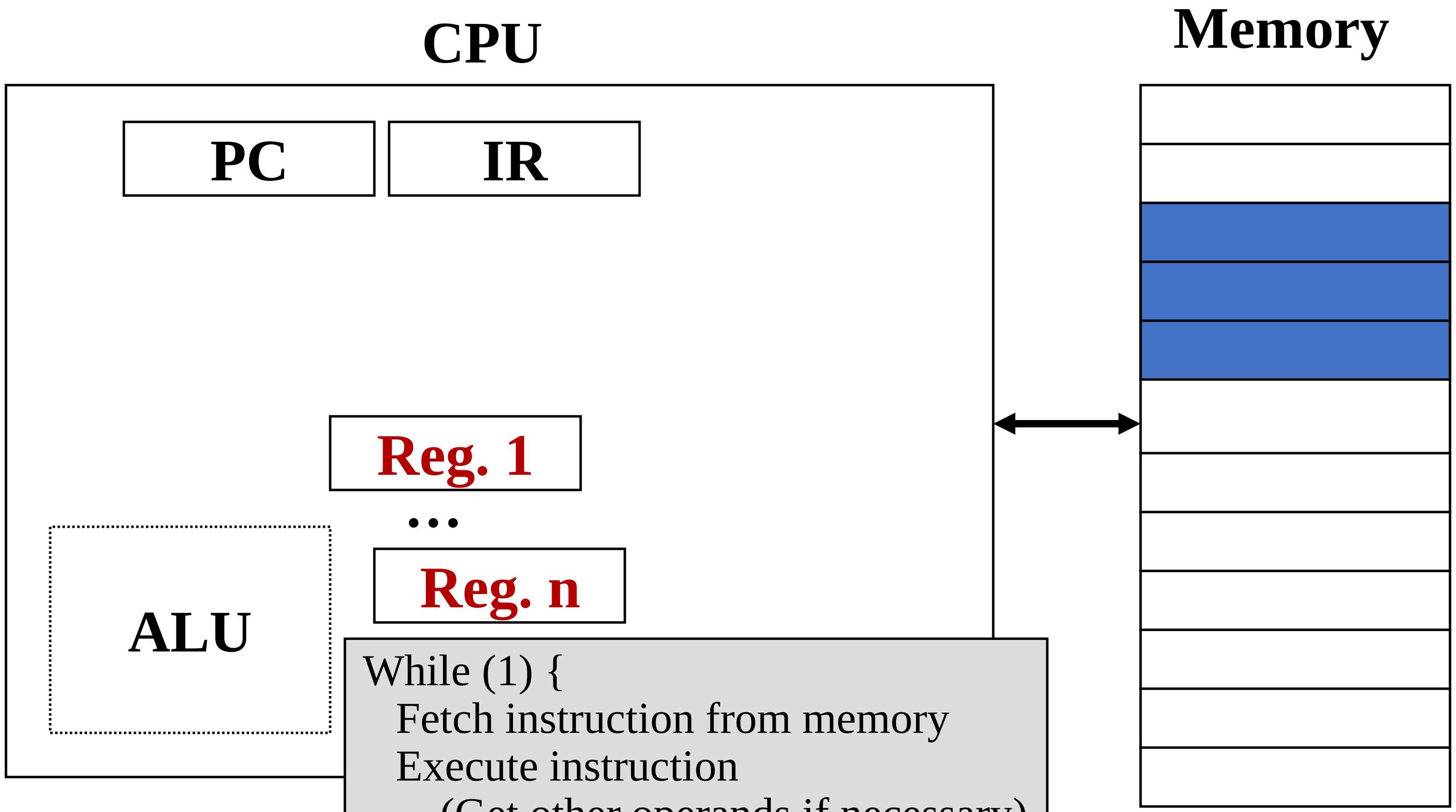

```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/Decode/Execute Cycle



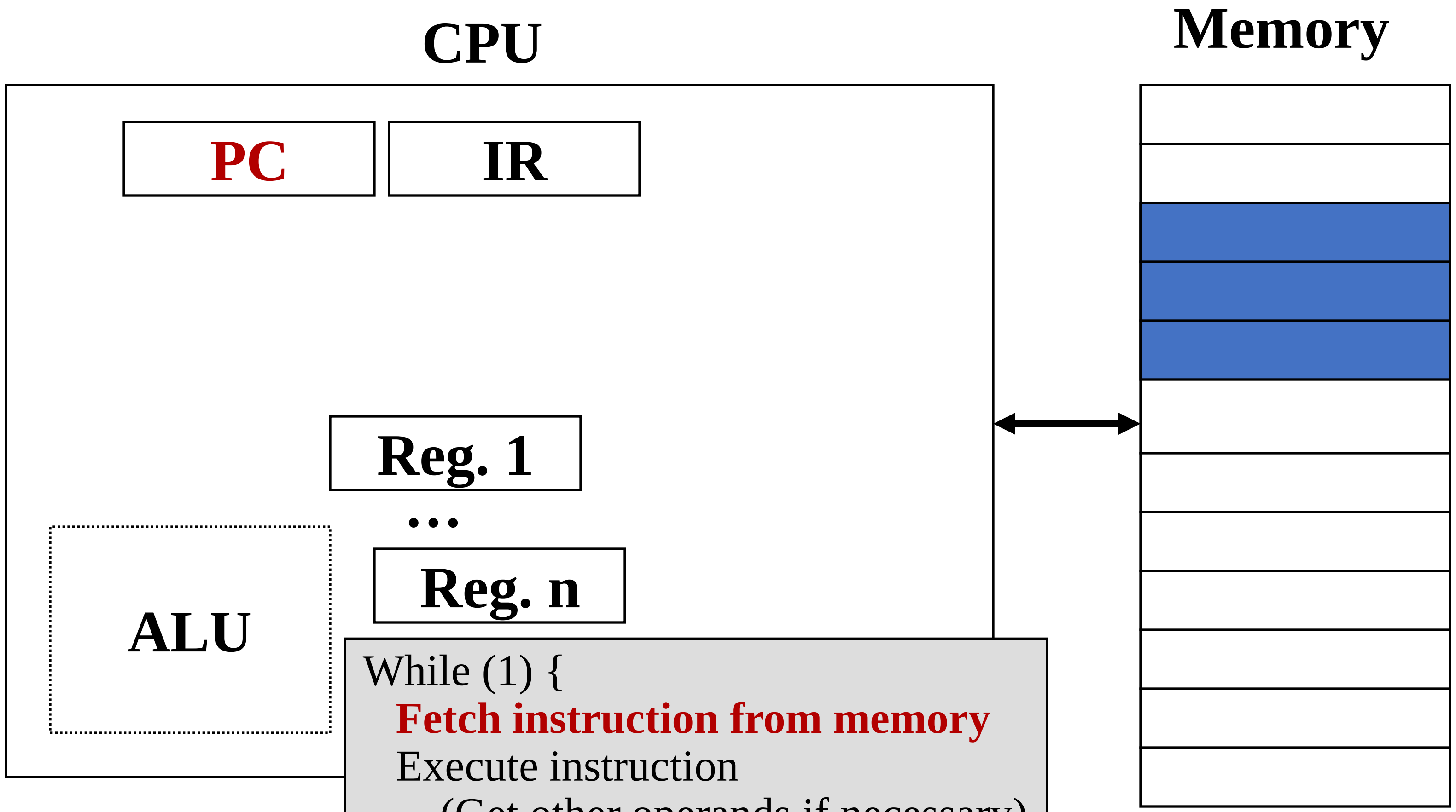

```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

Fetch/Decode/Execute Cycle




```
While (1) {
  Fetch instruction from memory
  Execute instruction
    (Get other operands if necessary)
  Store result
}
```

Fetch/Decode/Execute Cycle




```
While (1) {
   Fetch instruction from memory
   Execute instruction
      (Get other operands if necessary)
   Store result
}
```

The OS is Just a Program!

- The OS is just a sequence of instructions that the CPU will fetch/decode/execute
- How can the OS cause application programs to run?
- How can applications cause the OS to run?

How Can an OS Run Applications?

- The OS must load the address of the application’s starting instruction into the PC
- Example:
- computer boots and begins running the OS
- OS code must get into memory somehow
- fetch/decode/execute OS instructions
- OS requests user input to identify application program/file
- OS loads application (executable file) into memory
- OS loads the address of the app’s first instruction into the PC
- CPU fetches/decodes/executes the application’s instructions

The OS is Just a Program!

- How does the OS ever get to run again?
- How can the OS switch the CPU to run a new application (and later resume the first one)?
- How can the OS maintain control of what the application does when the OS is not running?
- In what ways can application try to seize control indefinitely (ie. cheat)?
- How can the OS prevent such cheating?

How Can the OS Regain Control?

- What if an application doesn’t call the OS and instead just hogs the CPU?
  - *OS needs interrupts from a timer device!*
  - *OS must register a future timer interrupt before handing control of the CPU over to an application*
  - *When the timer interrupt goes off the hardware starts running the OS at a pre-specified location called an interrupt handler*
  - *The interrupt handler is part of the OS program*
  - *The address of the interrupt handler’s first instruction is placed in the PC by the h/w*

Can the Application Cheat?

- Can the application disable the future timer interrupt so that the OS can not take control back from it?
- Disabling interrupts must be a privileged instruction that is not executable by applications
- The CPU knows whether or not to execute privileged instructions based on the value of the mode bit in the PSW!
- Privileged instructions can only be executed when the mode bit is set
- - eg. disabling interrupts, setting the mode bit!
- - attempted execution in non-privileged mode generally causes an interrupt (trap) to occur

Are There Other Ways to Cheat?

- What stops the running application from modifying the OS?
  - *- eg. modifying the timer interrupt handler to jump control back to the application?*

What Stops Applications From Modifying the OS?

- Memory protection!
- Memory protection instructions must be privileged
- - i.e., they can only be executed with the mode bit set …
- Why must the OS clear the mode bit before it hands control to an application?

How Can Applications Invoke the OS?

- Why not just set PC to an OS instruction address and transfer control that way?

How Can Applications Invoke the OS?

- Special trap instruction causes a kind of interrupt
  - *changes PC to point to a predetermined OS entry point instruction*
  - *simultaneously sets the mode bit*
  - *CPU is now running in privileged mode*
- Application calls a library procedure that includes the appropriate trap instruction
- fetch/decode/execute cycle begins at a pre-specified OS entry point called a system call handler

Are Traps Interrupts?

- Traps, like interrupts, are hardware events
- But traps are synchronous where as interrupts are asynchronous
- i.e. traps are caused by the executing program rather than a device external to the CPU

Switching to a New Application?

- To suspend execution of an application the OS must run!
- After that, simply
- - capture the application’s memory state and processor state
- - preserve all the memory values of this application
- - copy values of all CPU registers into a data structure which is saved in memory
- - restarting the application from the same point just requires reloading the register values

Recap

- Why do we need a timer device?
- Why do we need an interrupt mechanism?
- Why do we need privileged instructions?
- Why are system calls different to procedure calls?
- How are system calls different to interrupts?
- Why is memory protection necessary?
- How can the OS switch among applications?