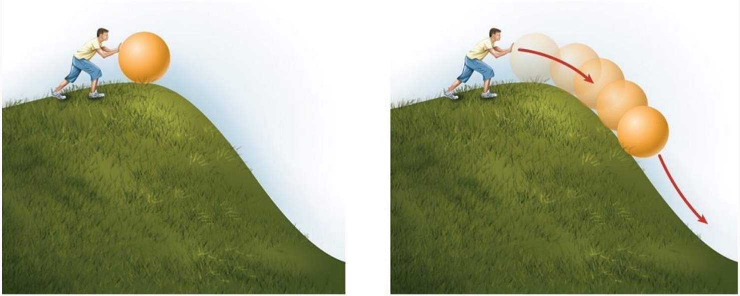
Overview of Computer Architecture

Operating Systems

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

December 6, 2019

http://carlhenrik.com

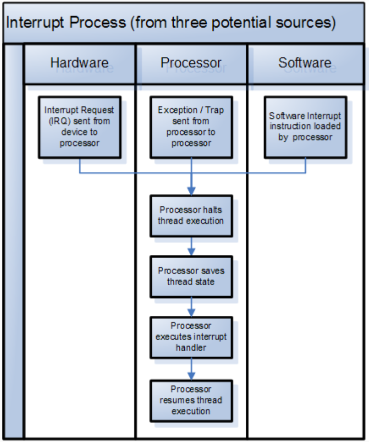


a) Potential energy

(b) Kinetic energy

Interrupts

- Software `instruction`
- Processor `TRAP`
- Hardware `IRQ`


Interrupt Process (from three potential sources)
Hardware
Processor
Software
Interrupt Request (IRQ) sent from device to processor
Exception / Trap sent from processor to processor
Software Interrupt instruction loaded by processor
Processor halts thread execution
Processor saves thread state
Processor executes interrupt handler
Processor resumes thread execution

Software Interrupts

Code

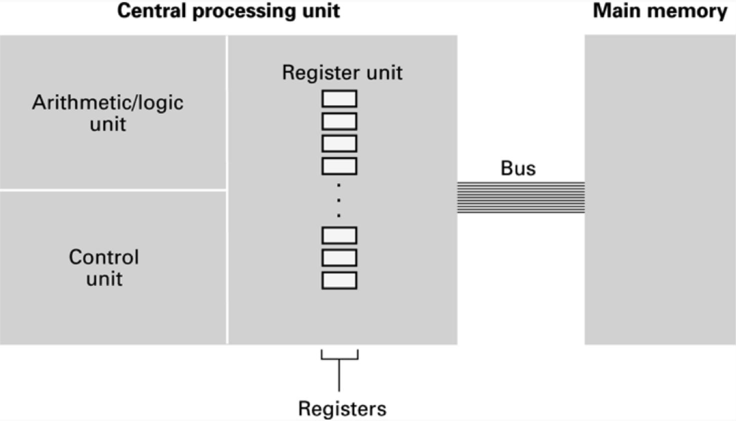
```
swi         0x11
```

TRAP

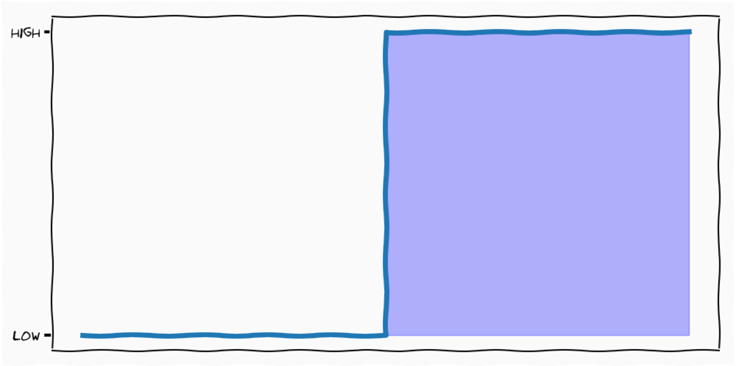
Code

```
mov        r0, #42
eor        r1, r1, r1
sdiv        r2, r1, r0
```

CPU


Central processing unit
Main memory
Arithmetic/logic unit
Register unit
Control unit
Bus
Registers

Hardware Interrupts



- ARM has 72 dedicated IRQ lines
- x86 has 16 dedicated IRQ lines

Interrupt is Triggered

1. Interrupt is triggered

Interrupt is Triggered

1. Interrupt is triggered
2. Disable Interrupts

Interrupt is Triggered

1. Interrupt is triggered
2. Disable Interrupts
3. Processor halts execution

Interrupt is Triggered

1. Interrupt is triggered
2. Disable Interrupts
3. Processor halts execution
4. Save state of current context

Interrupt is Triggered

1. Interrupt is triggered
2. Disable Interrupts
3. Processor halts execution
4. Save state of current context
5. Call Interrupt Handler

Interrupt is Triggered

1. Interrupt is triggered
2. Disable Interrupts
3. Processor halts execution
4. Save state of current context
5. Call Interrupt Handler
6. Restore state of previous context

Interrupt is Triggered

1. Interrupt is triggered
2. Disable Interrupts
3. Processor halts execution
4. Save state of current context
5. Call Interrupt Handler
6. Restore state of previous context
7. Enable Interrupts

Interrupt Vector Table

Interrupt ID	Address of Routine
IRQ 0	0x000fff00
IRQ 7	0x0ff0ff00

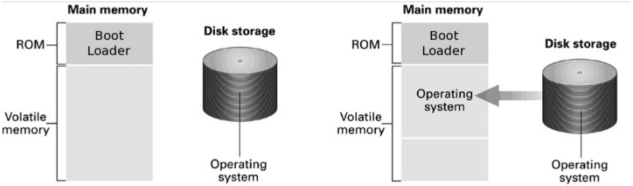
/proc/interrupts

Code

```
cat /proc/interrupts
```

Start

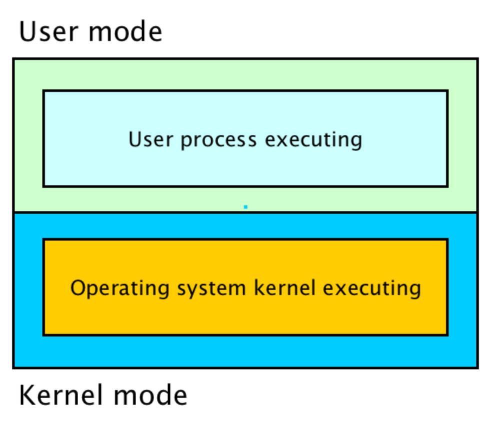
BIOS



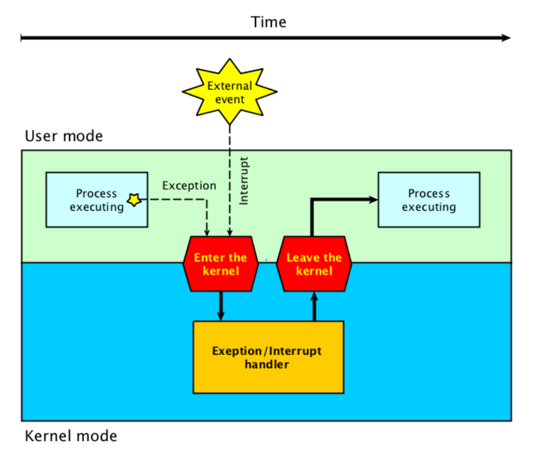
Step 1: Machine starts by executing the bootstrap program already in memory. Operating system is stored in mass storage.

Step 2: Boot loader program directs transfer of the operating system into main memory and then tranfers control to it.

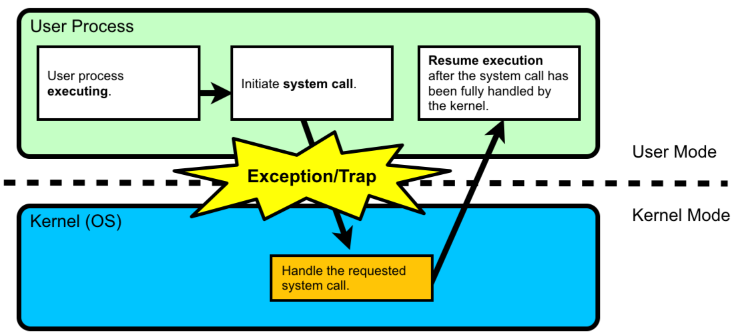
User/Kernel Mode



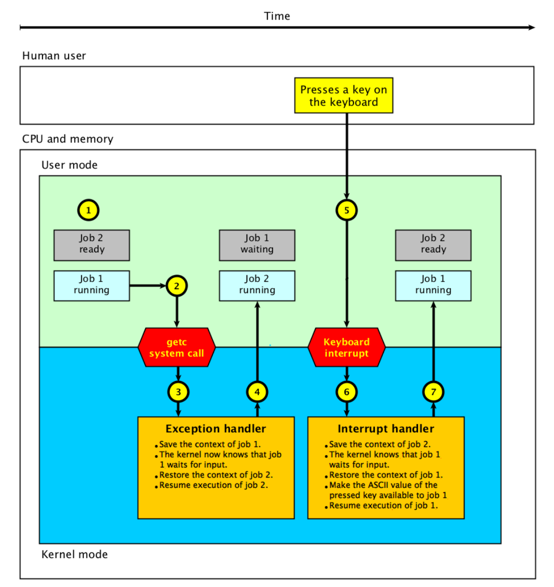
Interrupt


Time
External event
User mode
Process executing
Exception
Interrupt
Process executing
Enter the kernel
Leave the kernel
Exeption/Interrupt handler
Kernel mode

Interrupt


User Process
User process **executing**.
Initiate **system call**.
Resume execution after the system call has been fully handled by the kernel.
Exception/Trap
User Mode
Kernel Mode
Kernel (OS)
Handle the requested system call.

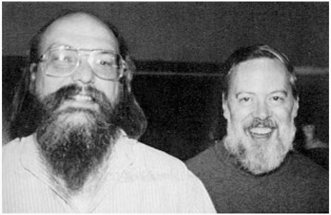
Multiple Processes


Time
Human user
Presses a key on the keyboard
CPU and memory
User mode
1
Job 2 ready
Job 1 running
2
Job 1 waiting
Job 2 running
5
Job 2 ready
Job 1 running
getc system call
Keyboard interrupt
3
4
6
7
Exception handler
• Save the context of job 1.
• The kernel now knows that job 1 waits for input.
• Restore the context of job 2.
• Resume execution of job 2.
Interrupt handler
• Save the context of job 2.
• The kernel knows that job 1 waits for input.
• Restore the context of job 1.
• Make the ASCII value of the pressed key available to job 1
• Resume execution of job 1.
Kernel mode

Operating Systems

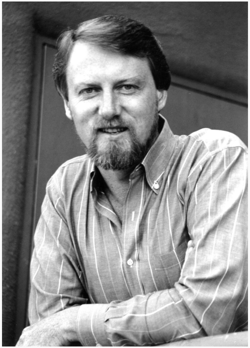
Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds



Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds



Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds

Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds

Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds

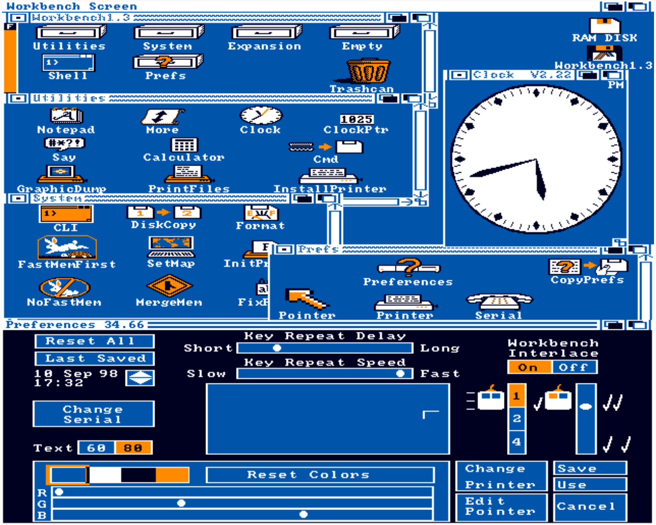
Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds

Operating Systems

- 1969 UNIX
  - Dennis Ritchie, Kenneth Thompson
- 1974 CP/M
  - Gary Kildal
- 1977 BSD
- 1980 (86/Q)-DOS
  - Seattle Computer Products
- 1987 MINIX
  - Andrew Tanenbaum
- 1991 Linux
  - Linus Torvalds

Operating Systems



UNIX

Henry Spencer Those who do not understand UNIX are condemned to reinvent it, poorly.

UNIX

Henry Spencer Those who do not understand UNIX are condemned to reinvent it, poorly.

Doug Gwyn UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things.

UNIX

Henry Spencer Those who do not understand UNIX are condemned to reinvent it, poorly.

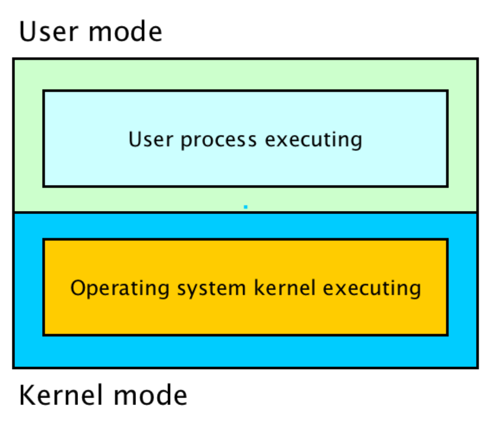
Doug Gwyn UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things.

Jeremy S. Anderson There are two major products that came out of Berkeley: LSD and UNIX. We don't believe this to be a coincidence.

Today

1. Processes
2. Process Scheduling
3. Memory Management

User/Kernel Mode



Process

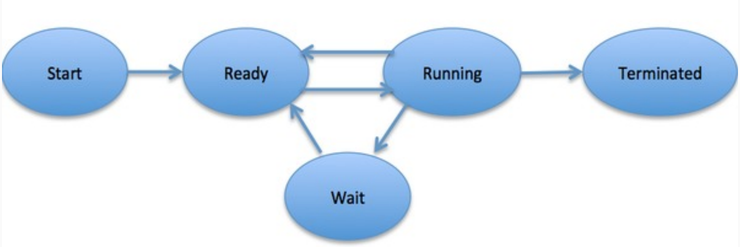
Code

```c
#include <stdio.h>


int main(void)
{
  int j = 0;
  for(int i=0;i<0xffff; i++)
    {
      j += i;
    }


  return j;
}
```

Process



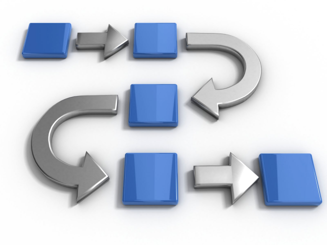
Process

Code

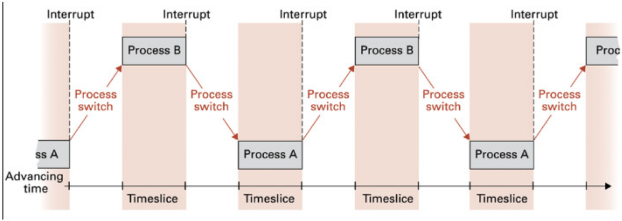
```
pstree
top
ps - e | grep xterm
sudo ls -l /proc/{PID}
```

Scheduler

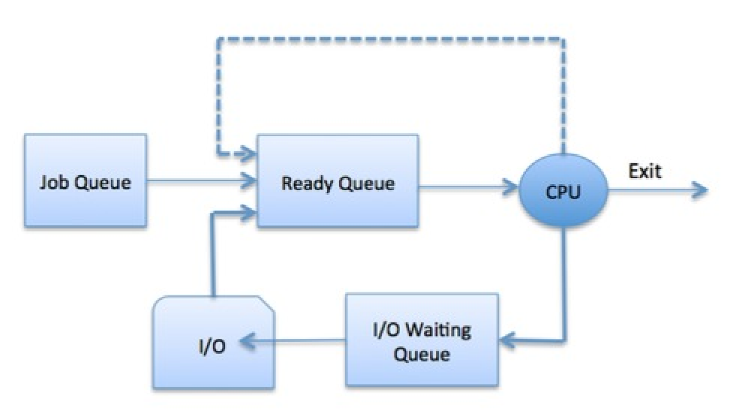
- Program is static
- Execution is not
- Process state
    - Program Counter
    - Registers
    - (Memory)
- Context switch



Scheduler


Interrupt
Interrupt
Interrupt
Interrupt
Interrupt
Process B
Process B
Proc
Process switch
Process switch
Process switch
Process switch
Process switch
ss A
Process A
Process A
Advancing time
Timeslice
Timeslice
Timeslice
Timeslice

Scheduler

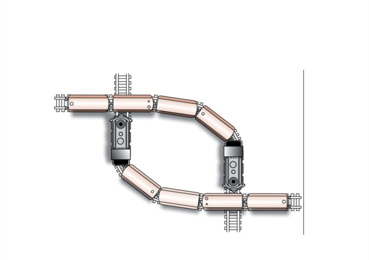


Short Term Scheduler

- How to structure execution?

Short Term Scheduler

- How to structure execution?
- Deadlock



Short Term Scheduler

- How to structure execution?
- Deadlock
- Starvation

Scheduling

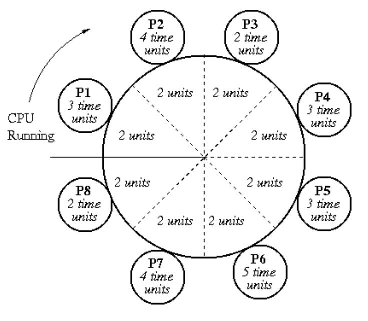
- Maximise Throughput
- Minimise Latency
- Minimise Overhead
- Responsiveness
- Real-time (deadline)

FIFO

- Naive
- Simple to Implement
- Low overhead

Round-Robin

- Slice time
- Each process in turn
- No starvation
- "Fair"
- Not sensible for real-time



Scheduling

- Each process associated with different priority
- Starvation
  - Ageing - increase priority over time
- Allows for real-time

Scheduling

- Multiple FIFO quest
- Different priority of queue
- Time-slicing per queue
- Lowest-level have round-robing

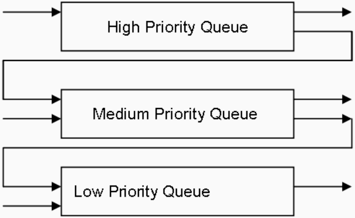


Figure – Multilevel Feedback Queue Scheduling

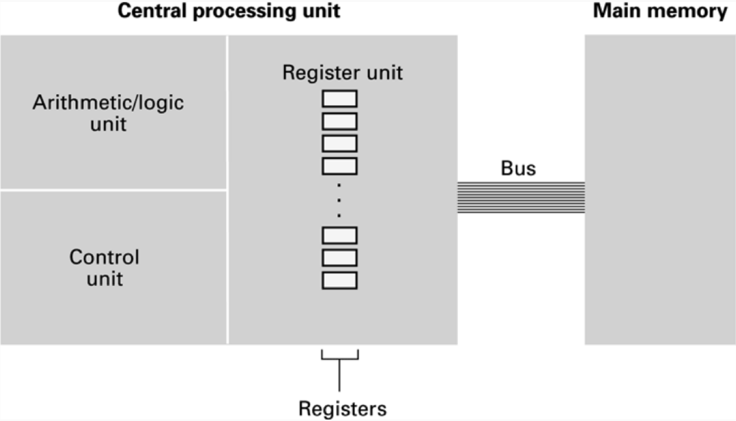
Linux Scheduling

- Interrupt ever 10^{-3} seconds
- Each process have priority N
    - A process runs uninterrupted for N slices
- *Goodness* measure of importance of process

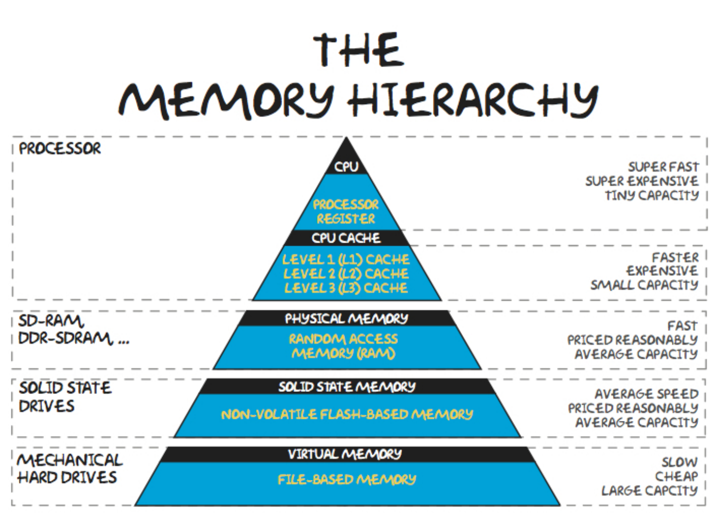
Summary

- Scheduling is very very hard
- Responsible for much of the look'n'feel of an operating system
- Combination of many different strategies common
- Remember *starvation* and *deadlock*

CPU


Central processing unit
Main memory
Arithmetic/logic unit
Register unit
Control unit
Bus
Registers

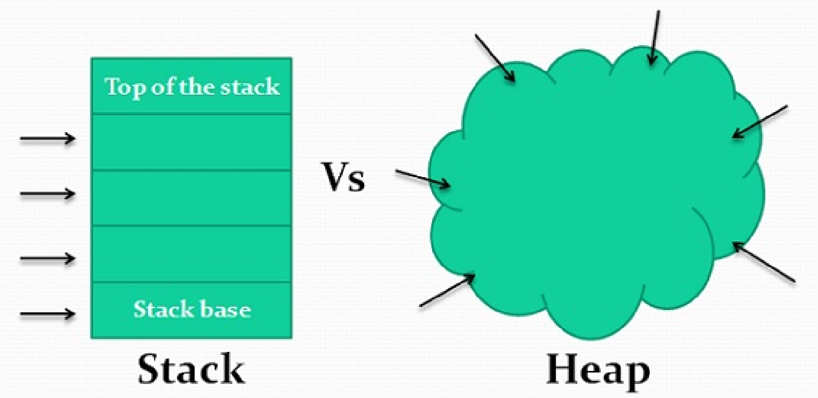
Memory Hierarchy



Memory

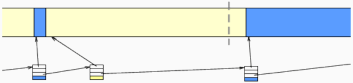
- Bus can address memory
- Tasks
  - Allocation (distribution)
  - Protection
- Fragmentation
- Overhead

Memory: Allocation

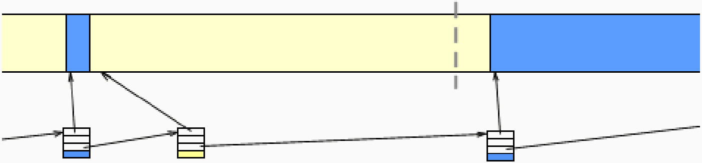


Memory: Allocation

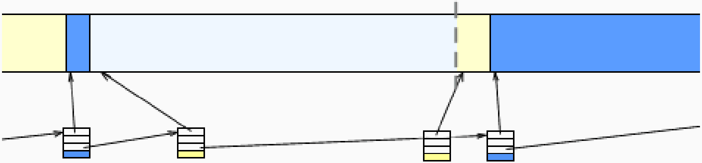
- Allocation
    - First fit
    - Best fit
    - Worst fit
- Sorting for quicker allocation



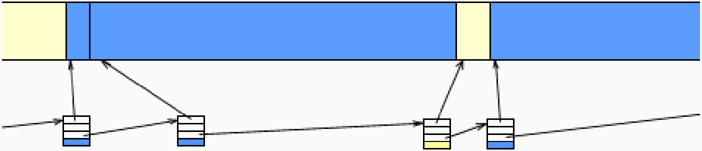
Memory: Allocation



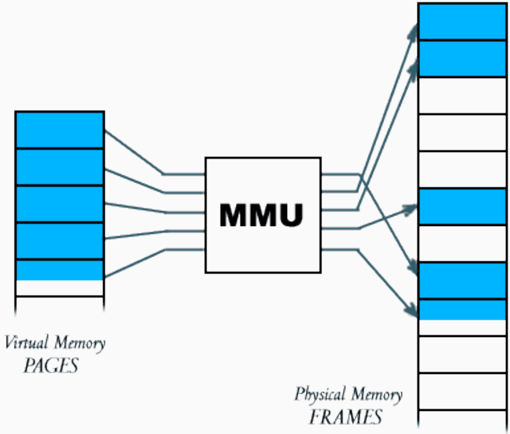
Memory: Allocation



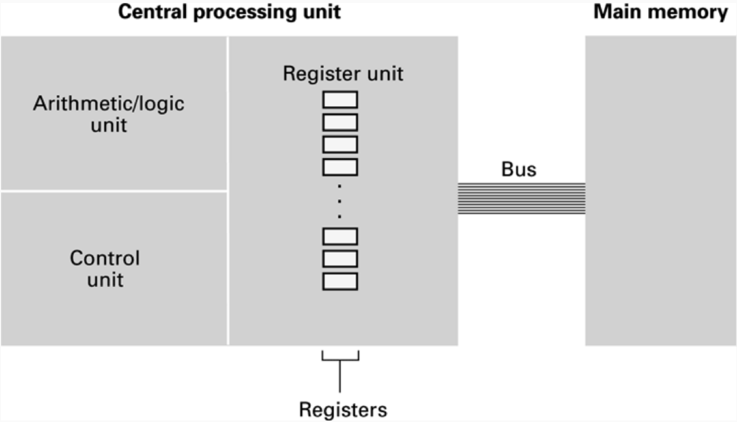
Memory: Allocation



Memory Management Unit



Memory Protection



Summary

- Memory management is very hard
- Trade-off between overhead and efficiency
- Allocation/Protection

Summary

Summary

- Booting
- Kernel/User space
- Executing management
- Memory management
- Its not important to know how the work, but to know why we have them