



# FUNDAMENTOS DA COMPUTAÇÃO

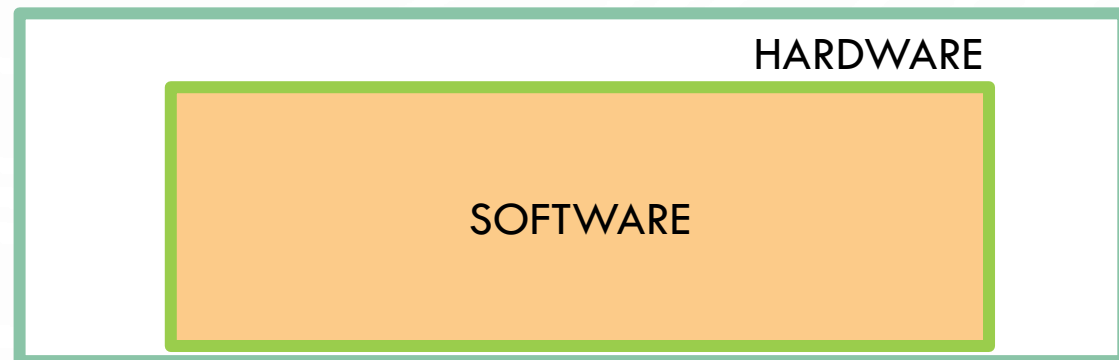
PROF. JOSENALDE OLIVEIRA

[josenalde.oliveira@ufrn.br](mailto:josenalde.oliveira@ufrn.br)

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

# SOFTWARE

**Software:** em oposição ao que denominamos *hardware*, o software (popularmente chamado *programa de computador*) compõe o que torna a máquina funcional, “útil”, para os mais variados **públicos** (e **aplicações**). O hardware sem software pode ser compreendido como um amontoado de circuitos, que precisam ser **programados**, adequadamente instruídos por **algoritmos**, codificados em **linguagens de programação** e disponibilizados como produtos. É portanto, artefato intelectual, sujeito à direito de propriedade (registro).



# SOFTWARE: ALGORITMO IMPLEMENTADO->TRADUZIDO

**Exemplo de algoritmo:** exibir números pares de 1 a 10

**Programa** par;

**Variáveis:**

**n** : inteiro;

**Início:**

n=1; // inicializa contador;

Enquanto (n<=10) faça

    se resto(n,2) = 0 então

        imprima(n + 'é par');

    senão

        imprima(n + 'é impar');

    fim\_se

    n = n + 1;

Fim\_Enquanto

**Fim**

# SOFTWARE: ALGORITMO IMPLEMENTADO->TRADUZIDO

## Implementação (Octave)

```
function par

n = 1;
while (n<=10)
    if (mod(n,2) == 0)
        disp(n);disp('é par');
    else
        disp(n);disp('é impar');
    end
    n = n + 1;
end

end
```

## Implementação (C)

```
void par() {
    n = 1;
    while (n<=10) {
        if (n%2 == 0)
            printf("%d par\n", n);
        else
            printf("%d par\n", n);

        n++;
    }
}
```

# SOFTWARE: ALGORITMO IMPLEMENTADO->TRADUZIDO

## Implementação (python)

```
def par():
    n = 0
    while (n < 10):
        n += 1
        if n % 2 == 0:
            print (n, ' par')
        else:
            print (n, 'impar')
```

## Implementação (javascript)

```
<script>
    var n = 10;
    while (n < 10) {
        if (n%2 == 10) {
            document.write("par "+num+"<br>");
        } else {
            document.write("impar"+num+"<br>");
        }
        n++;
    }
</script>
```

# SOFTWARE

**Observação:** sabemos que a CPU reconhece e opera internamente com o sistema binário (0, 1), logo, o programador necessitaria escrever o código de seu programa no sistema binário ?

A programação de computadores evoluiu bastante e são utilizadas as chamadas **LINGUAGENS DE ALTO NÍVEL**, muito próximas do idioma natural e outros softwares (categorizados como básicos), chamados **COMPILADORES** geram código de **BAIXO NÍVEL** (mais próximo ao entendido pela máquina e circuitos).

**Linguagens de alto nível:** C, C++, Pascal, Basic, Java, Php, Javascript, Python etc.)

**Linguagem de baixo nível:** Assembler (Ex.: programação de microcontroladores)

# EXEMPLO: SISTEMA EMBARCADO

## Piscar um LED na plataforma Arduino (ATMEGA328P)

```
void setup() {  
    pinMode(8,OUTPUT);  
    Serial.begin(115200);  
}  
  
void loop() {  
    digitalWrite(8,HIGH);  
    delay(1000);  
    digitalWrite(8,LOW);  
    delay(1000);  
}
```

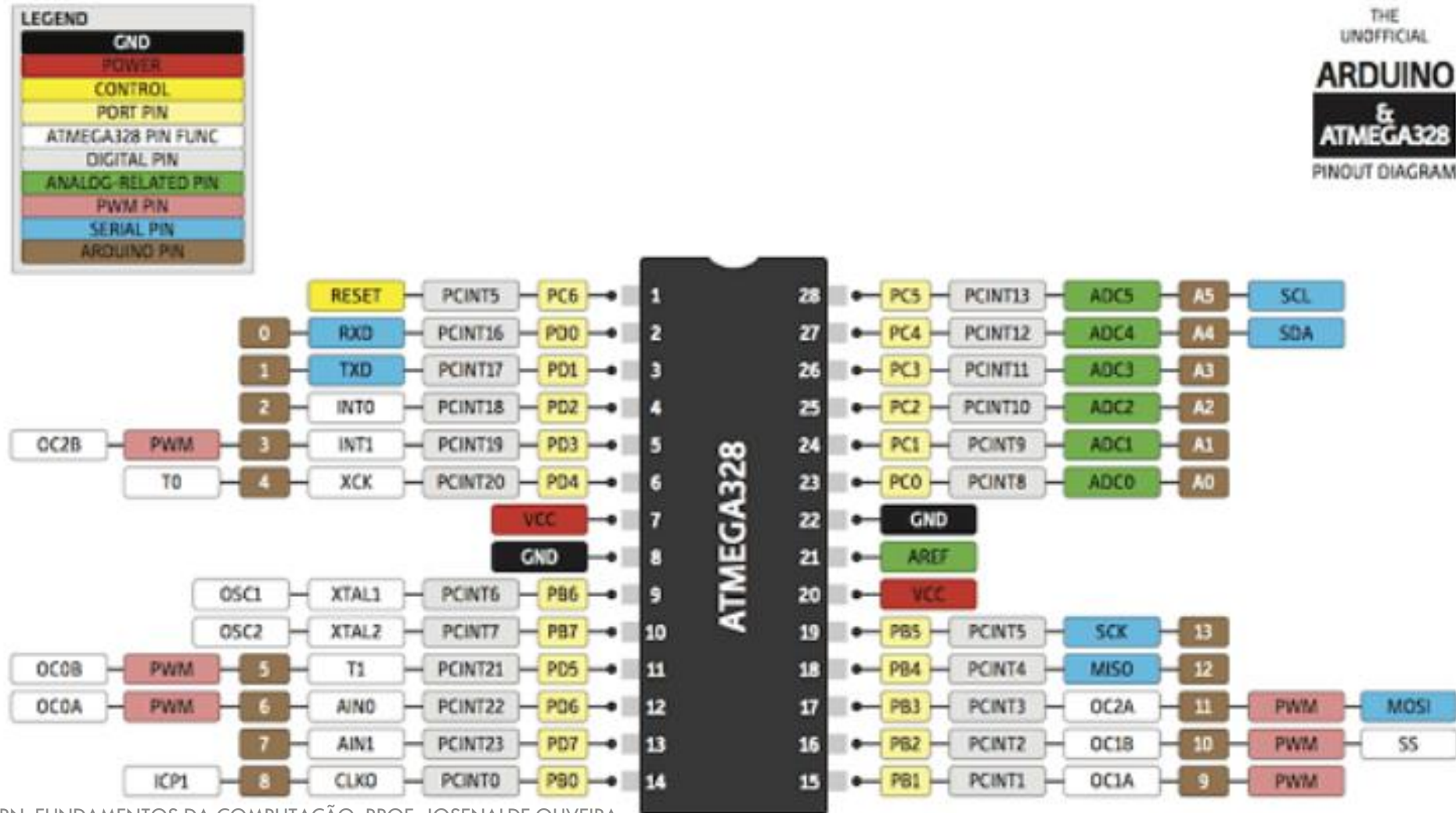
Alto nível de abstração  
API (conjunto de bibliotecas, diretivas etc.)

```
void setup() {  
    DDRB = 0xFF; // 255  
}  
  
void loop() {  
    PINB.F0 = 1;  
    delay_ms(1000);  
}
```

Conhecer REGISTRADORES DA CPU



# EXEMPLO: SISTEMA EMBARCADO: PINAGEM CI



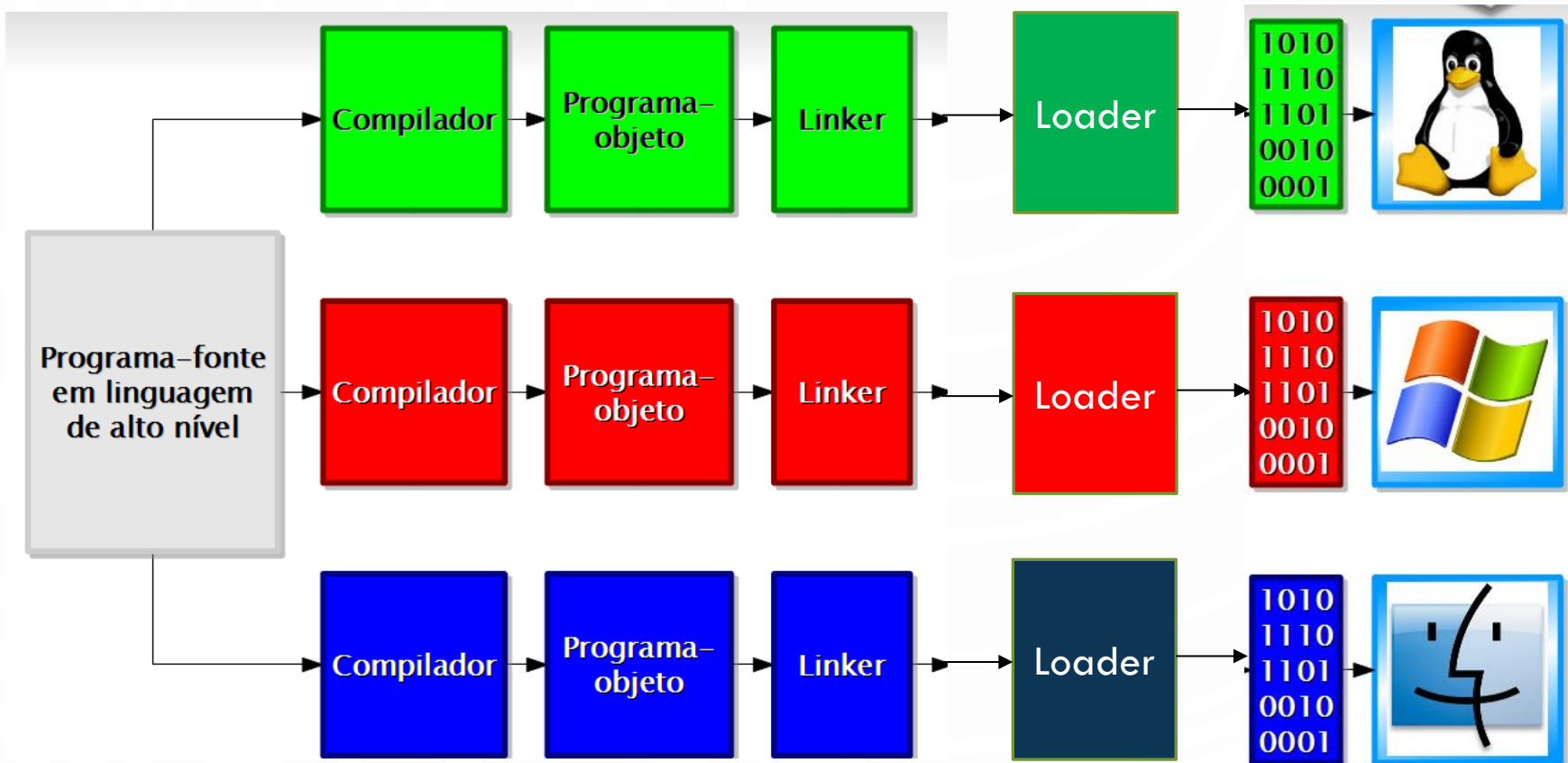


# LINGUAGENS INTERPRETADAS X COMPILADAS

**Compilação** (Exemplos C, C++): conjunto de etapas: pré-processamento (código fonte é expandido, pela abertura das diretivas #), verificação sintática, para cada arquivo fonte (exemplo .c) é criado um arquivo objeto, com instruções de linguagem de máquina que correspondem ao arquivo fonte compilado; por último, a link-edição une todos os arquivos objeto num único arquivo executável (fonte + objetos das bibliotecas usadas) – compilação estática

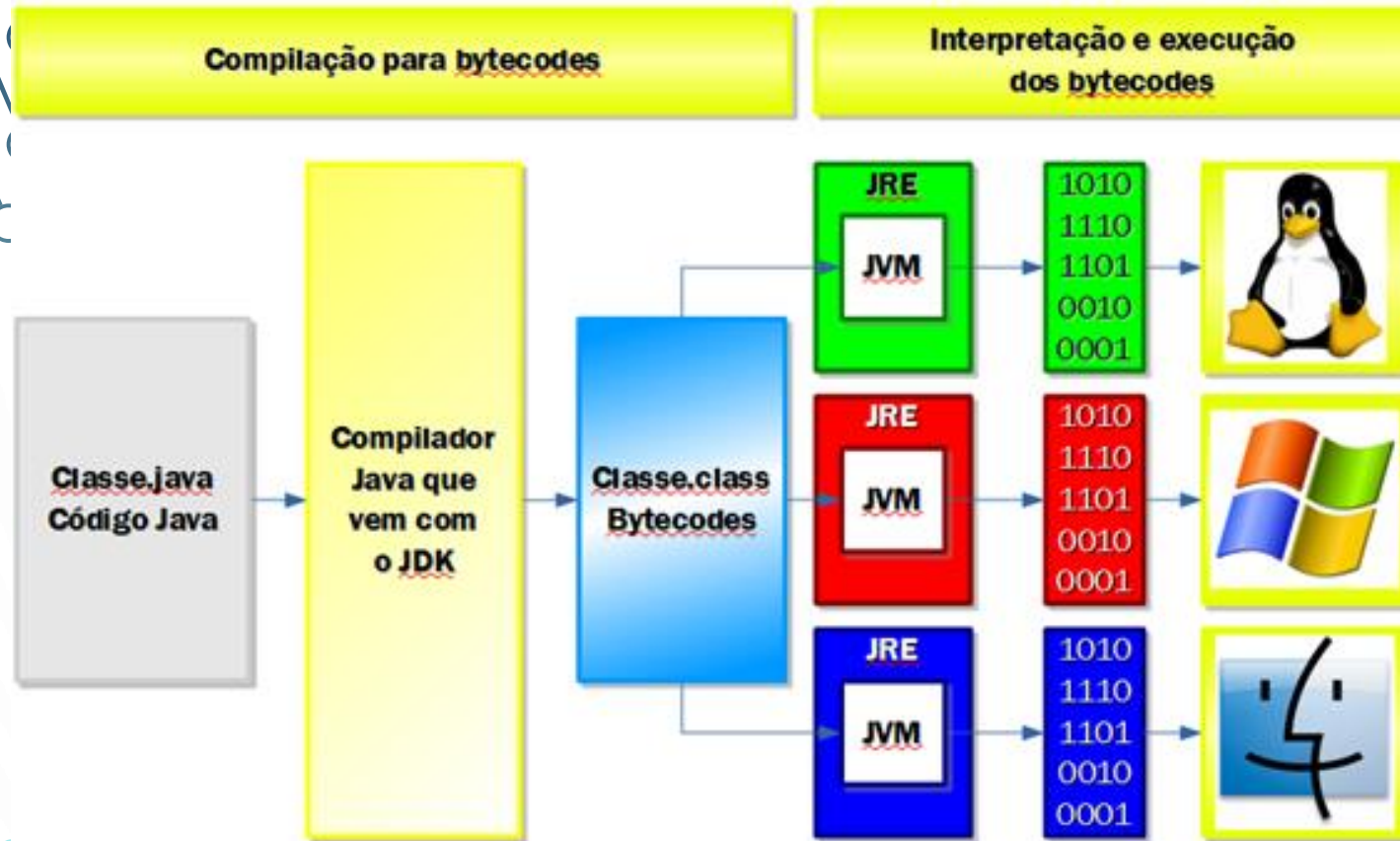
**Interpretação:** (Exemplos Javascript, Python, Matlab): tem seus códigos fonte transformados em linguagem intermediária que será interpretada pela máquina virtual da linguagem quando for executada. Este processo consiste na tradução da ling. intermediária para a linguagem da máquina virtual – “compilação dinâmica”

# LINGUAGENS INTERPRETADAS X COMPILADAS



<https://rogeraoaraujo.com.br/2013/01/06/java-compilacao-de-classes-java-parte-i/>

# LINGUAGENS INTERPRETADAS X COMPILADAS



JDK – compilador: .java->.class

JRE – ambiente de execução na plataforma alvo (JVM+API Java)

JVM – interpreta e executa o bytecode

A **máquina virtual Java** é a responsável pela **portabilidade!** Ela interpreta e executar o bytecode. É a provedora de formas e meios de o aplicativo conversar com o sistema operacional.

**API** (Java Application Programming Interface): É uma biblioteca de componentes que possui vários recursos úteis; e é utilizada para execução de aplicações Java.

<https://rogeraoaraujo.com.br/2013/01/21/java-compilacao-de-classes-java-parte-ii/>

# LINGUAGENS INTERPRETADAS X COMPILADAS

**FGV 2012 Senado Federal – Prova anulada – Análise de Sistemas – Questão 55]** Para permitir que um mesmo programa seja executado em vários sistemas operacionais, a plataforma java gera códigos genéricos \*.class e os traduz para o código da máquina local, \*.exe ou \*.bin, somente no momento da execução. Nesse contexto, os códigos específicos para a máquina virtual Java, e não para a máquina local, recebe o nome de:

- [A] microcode.
- [B] scriptcode.
- [C] framecode.
- [D] bytecode.
- [E] javacode.

<https://rogeraoaraujo.com.br/2013/01/21/java-compilacao-de-classes-java-parte-ii/>

# LINGUAGENS INTERPRETADAS X COMPILADAS

**[FCC 2010 TRT 9ª Região – Técnico Judiciário – Especialidade Tecnologia da Informação – Questão 36]** O JVM mais o núcleo de classes da plataforma Java e os arquivos de suporte formam o

- [A] o J2EE.
- [B] o JDK.
- [C] o JRE.
- [D] uma JSP.
- [E] uma API.

**FCC 2007 TJ/PE – Analista Judiciário – Analista de Suporte – Questão 25]** O código Java compilado é gerado em arquivo com extensão

- [A] .ser
- [B] .jar
- [C] .java
- [D] .html
- [E] .class

<https://rogeraoaraujo.com.br/2013/01/21/java-compilacao-de-classes-java-parte-ii/>

# BYTECODES – EXEMPLO 1

```
int foo(int a) {  
    if (a == 0) {  
        return 0;  
    } else {  
        return 1;  
    }  
}
```

```
int foo(int a):  
    0: ILOAD_1  
    1: IFEQ 4  
    2: BIPUSH 1  
    3: IRETURN  
    4: BIPUSH 0  
    5: IRETURN
```

# BYTECODES – EXEMPLO 2

```
int foo() {  
    int result = 0;  
    while (result < 2) {  
        result++;  
    }  
    return result;  
}
```

Ver comando **javap** (disassembler)!

```
int foo():  
    0: BIPUSH 0  
    1: ISTORE_1  
    2: GOTO 4  
    3: IINC 1,1  
    4: ILOAD_1  
    5: BIPUSH 2  
    6: IF_ICMPGE 3  
    7: ILOAD_1  
    8: IRETURN
```



# LINGUAGENS: TIPAGENS

**Estaticamente** tipadas (Exemplo C/C++, Java, Rust):  
estruturadas, exigem definição de tipos de variáveis

**Dinamicamente** tipadas (Exemplo python, javascript, php...):  
não exigem definição de tipo – tipos numéricos normalmente *double*

**Fortemente/Fracamente** tipadas: flexibilidade na coerência das operações. Fracamente são aquelas que não se importam com o tipo de dados contido em uma variável. Permitem que o programador não tenha de fazer conversões de tipos (*cast*). Conversões dinâmica implícita para tradução das instruções, como no Java, não significa que ela é fracamente tipada.

```
int num1 = 10;  
string num2 = "5";  
int num3 = num1 * num2; // erro
```

Fortemente e estaticamente tipada: c/c++

1 var1 = 120	1 var1 = 120
2 var2 = "10"	2 var2 = "10"
3 var3 = var1 * var2	3 var2 = 10
4 # Erro	4 var3 = var1 * var2
	5 # resultado: 1200

Fortemente e dinamicamente tipada: python/ruby

```
a = 10;  
b = '20';  
c = a + b;  
d = a * b;  
print(c);  
print(d);
```

File "main.py", line 3, in <module>  
c = a + b; TypeError: unsupported  
operand type(s) for +: 'int' and 'str'

Fracamente e dinamicamente tipada: php/javascript

```
var a = 10;  
var b = '20';  
var c = a + b;  
var d = a * b;  
console.log(c); // 1020  
console.log(d); // 200
```



# SOFTWARE

## **Softwares aplicativos - orientados à tarefa (produtividade)**

- processadores de texto (Microsoft Word; LibreOffice Writer; GDocs etc.)
- editoração eletrônica (Pagemaker, CorelDraw, Gimp etc.)
- planilhas eletrônicas (Microsoft Excel; LibreOffice Calc)
- gerenciamento de banco de dados (Microsoft Access; LibreOffice Base)
- processamento e tratamento de imagens (Photoshop; Photopaint; Gimp)
- apresentações em slides (Microsoft Powerpoint; Impress; Prezi; Canva)
- comunicações (navegadores, correio eletrônico, social media etc.)

# SOFTWARE

**Softwares aplicativos - orientados à tarefa (produtividade): soluções WEB**



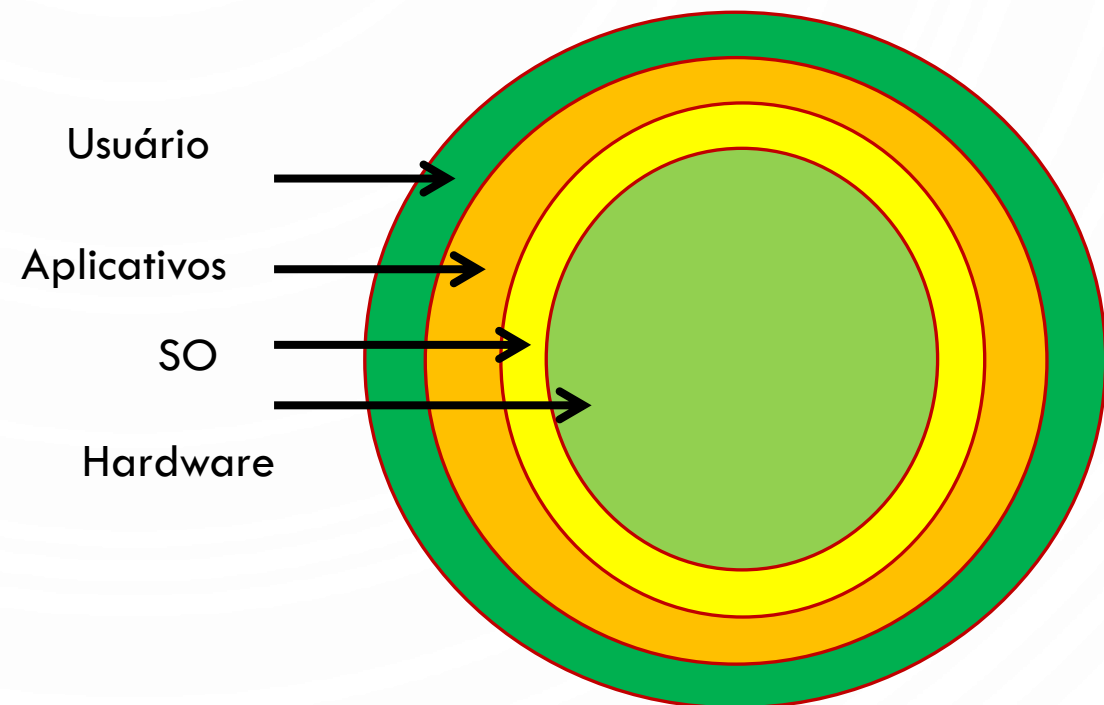
“Pesquisadores de Harvard descobriram: Prezi é mais envolvente, persuasivo, e eficaz do que PowerPoint.”



# SOFTWARE

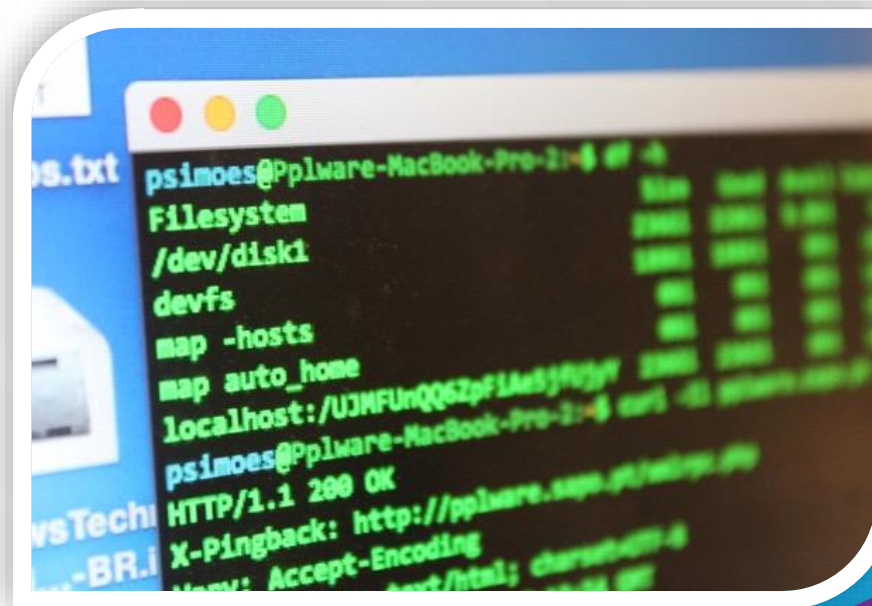
## Sistema operacional: software básico

OBS: *os softwares aplicativos não se comunicam diretamente com o hardware. É necessário um software intermediário (interlocutor), este é o **Sistema Operacional**.*

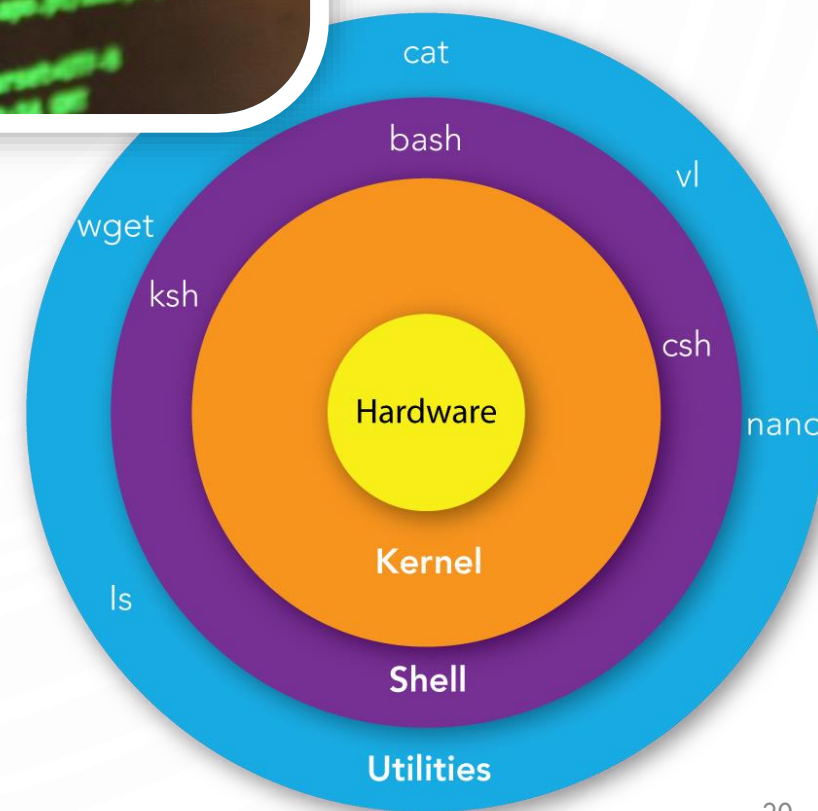


# SOFTWARE

## Shell e Sistema operacional



```
psimoes@Pplware-MacBook-Pro-2:~$ df -h
Filesystem
/dev/disk1
devfs
map -hosts
map auto_home
localhost:/UJMFUnQ06ZpFIAsjHjy?
psimoes@Pplware-MacBook-Pro-2:~$ curl -I http://pplware.sapo.pt/abcp.py
HTTP/1.1 200 OK
X-Pingback: http://pplware.sapo.pt/abcp.py
Content-Type: text/html; charset=utf-8
```



# SOFTWARE BÁSICO: SISTEMA OPERACIONAL

Gerencia o acesso a todos os recursos de hardware e software

- Gerência de memória (alocação de recursos)
- **Gerência de processos** (cada programa em execução é chamado processo)
  - no windows: `tasklist`, `taskkill`; gerenciador de tarefas no terminal
    - `Ctrl + Alt + Del`
- Gerência de entrada/saída (dispositivos)
- Gerência de usuários e grupos
- Logo, podemos visualizar as funções básicas:
  - manter os recursos do computador, como a CPU, a memória, o disco rígido e demais dispositivos de E/S; estabelecer uma interface com o usuário; executar e oferecer recursos para softwares aplicativos

# SOFTWARE BÁSICO: SISTEMA OPERACIONAL

## Sistema Operacional (software em segundo plano - background)

o KERNEL (núcleo) é a parte mais importante do SO e gerencia todos os recursos do computador. Ele permanece na memória, logo é RESIDENTE.

Apenas quando necessário, carrega do disco de armazenamento para a memória outros programas do SO, chamados NÃO-RESIDENTES.

Ao ligar o computador, o kernel é carregado do disco rígido para a RAM, tornando-o disponível. Esse processo se chama *bootstrapping* ou *booting* (**boot**). Ao ligar o computador, um pequeno programa armazenado na ROM realiza alguns testes de componentes de hardware internos e carrega o *kernel* da unidade não volátil (POST – *Power On Self Test*).



# SOFTWARE BÁSICO: SISTEMA OPERACIONAL

**Interpretador de comandos: shell/terminal/prompt – comandos interpretados sequencialmente – interação com terminal em script**

```
#include <iostream>

int main(int argc, char *argv[]) {
    // argv[0] é o nome do arquivo executável e argc é no mínimo 1
    cout << "Shell parameters: << argc-1;
    cout << "Hello " << argv[1];
    int a = atoi(argv[2]); // converter texto - inteiro
    int b = atoi(argv[3]);
    return 0;
}
```

# SOFTWARE BÁSICO: SISTEMA OPERACIONAL

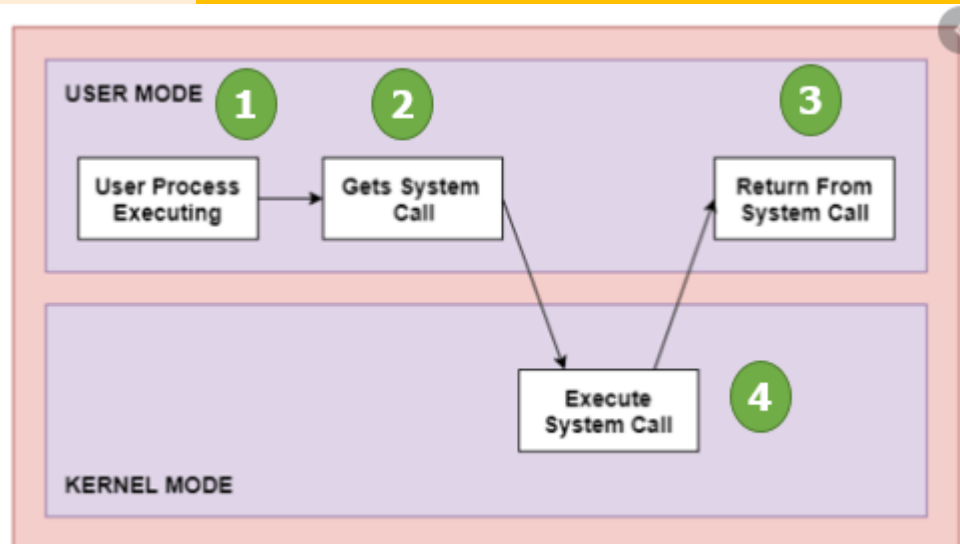
```
#include <time.h>
#include <stdio.h>

int main () {
    clock_t start_t, end_t, total_t;
    int i;
    start_t = clock();
    cout<< "Starting, start_t = " << start_t;
    for(i=0; i< 100000000; i++) {
    }
    end_t = clock();
```

```
total_t = (end_t - start_t);
cout<< "total cycles: " << total_t ;
```

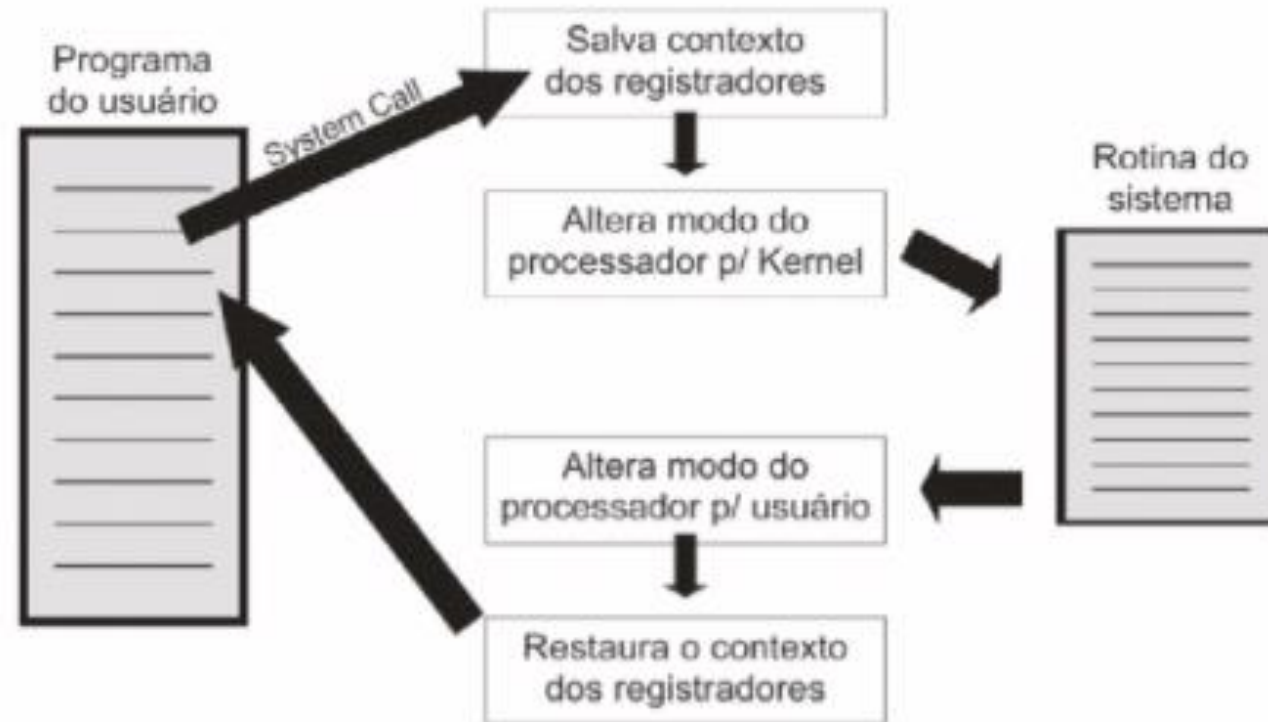
```
return(0);
```

**System calls (Chamadas de sistema para acessar Hardware/Driver)**





# SOFTWARE BÁSICO: SISTEMA OPERACIONAL



# SOFTWARE BÁSICO: SISTEMA OPERACIONAL

## Tipos de Sistemas Operacionais

- computadores pessoais (desktop e portáteis): Windows, MacOS, Linux etc.
  - Terminais bancários, terminais de supermercado (Linux)
- computadores de rede (servidores de serviços): Windows Server, Linux versões Server etc.
- smartphones, tablets (iOS, Android etc.), smart watch...
- Nuvem: Chrome OS...
- Sistemas embarcados/tempo real (FreeRTOS, uClinux...)
- **Robótica (Robotic OS – ROS)**

# SOFTWARE BÁSICO: SISTEMA OPERACIONAL

## ROS

### What is ROS?

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

[Read More](#)



# TIPOS DE SOFTWARES

## Tipos de Softwares (propriedade)

- proprietário (privado). Ex: Microsoft Windows, Microsoft Office
  - necessário adquirir licença de uso (existem vários tipos)
  - código fonte não disponível (em alguns casos tem sido aberto: text2speech Windows etc.)
  - correções apenas por parte da equipe de desenvolvedores
  - custo
  - acesso à suporte técnico
- livre (público)
  - licença de uso público
  - código fonte disponível
  - pode ser usado e distribuído, mas caso haja alteração, disponibilizar a mesma
  - em geral gratuito

# TIPOS DE SOFTWARES

- Freeware (gratuito, mas não necessariamente com código aberto – open source)
- Adware (software de uso normalmente gratuito, mas que aparecem banners com propagandas)
- Demo software (demonstração, com recursos reduzidos)
- Trial Version (versão de teste, possui prazo para expirar)
- Firmware (software embutido nos chips, como ROMs, microcontroladores etc.)
- Malware (softwares com conteúdo malicioso – *vírus, trojans, worms, keyloggers*)

# PARA IR ALÉM

- Pesquise sobre os conceitos de Just In Time Compiling (JIT) e Ahead of Time Compiling (AOT)
- Pesquise sobre a licença de software livre GNU
- Procedimentos para registro de software (AGIR-UFRN)
- Monetização de APPs nas 'lojas' e como disponibilizar (modelos)