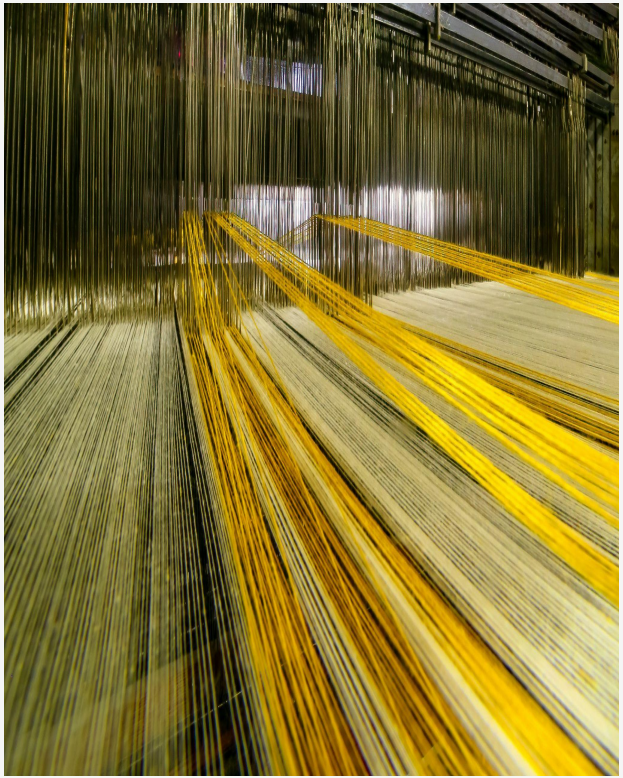


Photo by iggii on Unsplash

Programação concorrente e distribuída

Aula 2 - Conceitos básicos

Prof. João Robson

Plano de ensino

- Datas;
- Forma de avaliação;
- Bibliografia.

Ferramentas

- Estudo:
  - AVA (aulas/slides, links);
  - Biblioteca;
  - Google/IAs.
- Prática:
  - Inglês;
  - Java:
    - Curso da Universidade de Helsinki’s
    - Curso da Baeldung
  - Git/GitHub;
  - IDE (Eclipse).

Motivação

- Como realizar uma tarefa mais rápido?
  - Realizar a tarefa numa velocidade maior;
  - Mudar a forma de realizar a tarefa (fazer de forma mais inteligente, mudar as ferramentas usadas, etc.);
  - Pedir ajuda para alguém.
- Exemplo: pintura de uma casa
  - Aumentar velocidade da passagem de tinta;
  - Utilizar um rolo ou spray no lugar de um pincel;
  - Chamar o cunhado para ajudar.

Motivação

- Como realizar uma tarefa **computacional** mais rápido?
  - Realizar a tarefa numa velocidade maior:
    - Utilizar uma CPU mais rápida (*clock* maior)
  - Mudar a forma de realizar a tarefa (fazer de forma mais inteligente, mudar as ferramentas usadas, etc.):
    - Mudar o algoritmo utilizado
  - Pedir ajuda para alguém:
    - **Processamento concorrente, paralelo ou distribuído**

Computação sequencial vs paralela

Programação sequencial

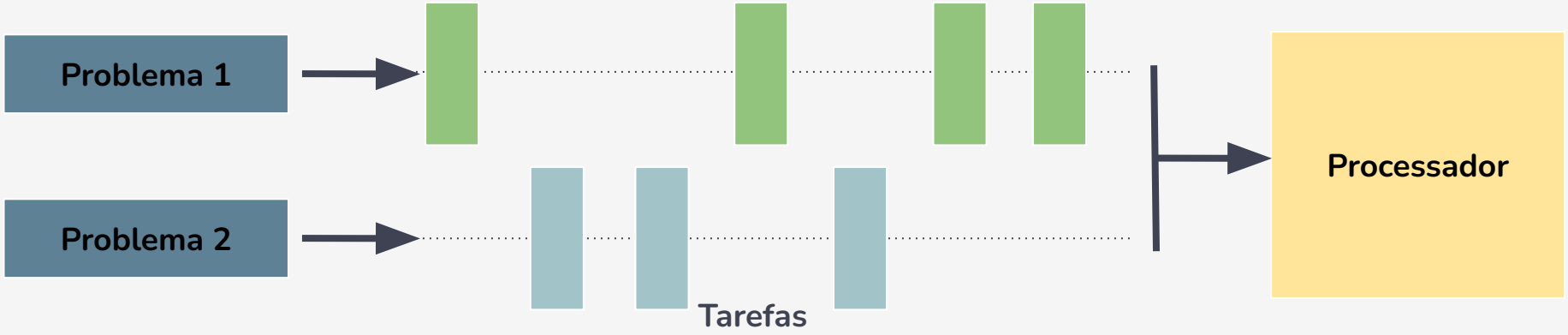
- Estratégia em que uma série de tarefas ou instruções são executadas de forma linear;
- **Cada instrução só é executada após o término da anterior.**

Programação sequencial

- Exemplo: cozinhar arroz
  - Tarefas:
    - Lavar o arroz (T1);
    - Picar cebola (T2);
    - Amassar o alho (T3);
    - Refogar alho e cebola no óleo (T4);
    - ...

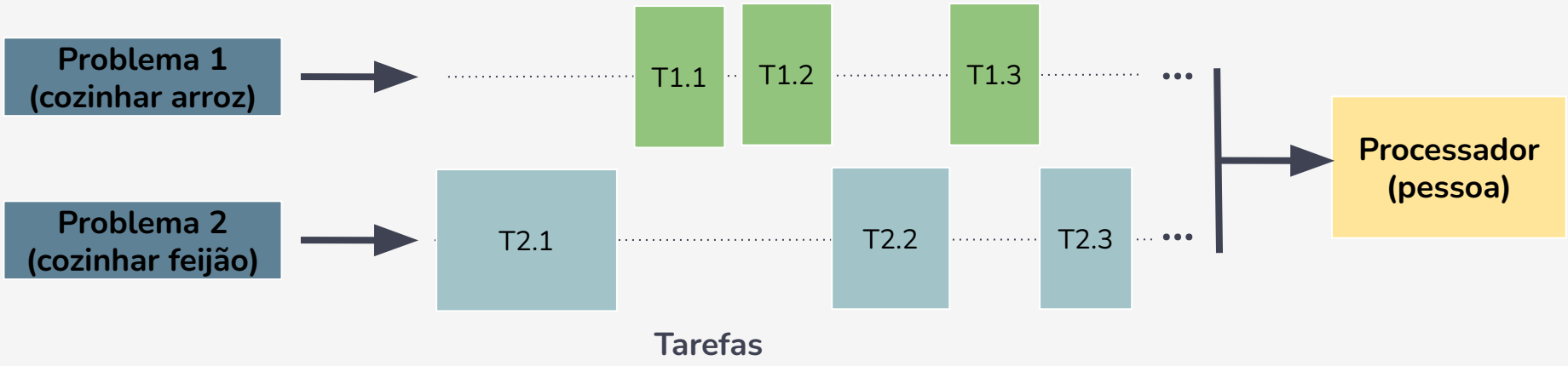
Programação concorrente

- Permite que dois ou mais problemas possam ser executados ou obtenham progresso **de forma aparentemente simultânea (“pseudoparalelismo”)**;
- Tarefas dentro de cada problema podem começar, serem executadas e concluídas em **períodos de tempo intercalados**;
- Tarefas não precisam ter uma ordem pré-definida, ou seja, **o resultado não é impactado pela ordem em que as instruções são executadas.**



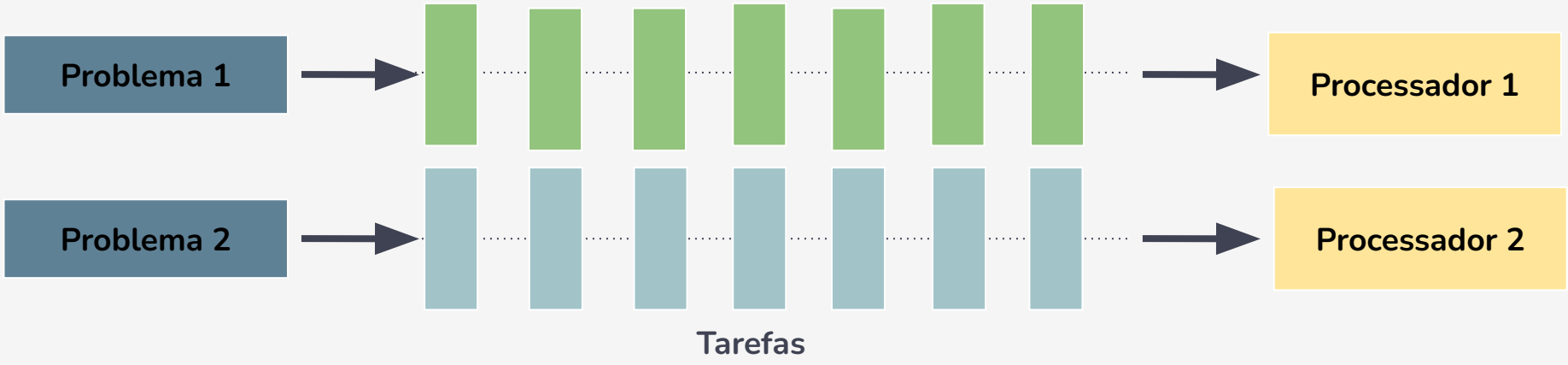
Programação concorrente

- Exemplo: cozinhar arroz (P1) + cozinhar feijão (P2)
  - P1 -> lavar o arroz (T1.1) + picar cebola (T1.2) + amassar o alho (T1.3) + refogar alho e cebola no óleo (T1.4) + ...
  - P2 -> deixar feijão de molho na água (T2.1) + cozinhar feijão na panela de pressão (T2.2) + refogar feijão (T2.3) + ...



Programação paralela

- Permite que dois ou mais problemas possam ser executados ou obtenham progresso **de forma simultânea**;
- Tarefas são executadas literalmente ao mesmo tempo;



Programação paralela

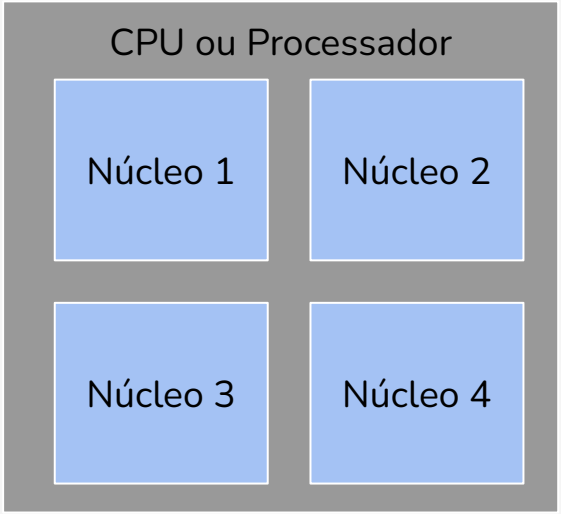
- Exemplo: cozinhar arroz (P1) + cozinhar feijão (P2)
  - P1 -> lavar o arroz (T1.1) + picar cebola (T1.2) + amassar o alho (T1.3) + refogar alho e cebola no óleo (T1.4) + ...
  - P2 -> deixar feijão de molho na água (T2.1) + cozinhar feijão na panela de pressão (T2.2) + refogar feijão (T2.3) + ...

Noções de sistemas operacionais

- Para compreender paralelismo e concorrência é necessário entender como programas são executados pelo computador;
- A execução das tarefas e programas é controlada pelo sistema operacional e depende da arquitetura do computador também;
- Em geral, computadores executam processos (“programas”) em seus **processadores**.

CPU vs Processador vs Núcleos

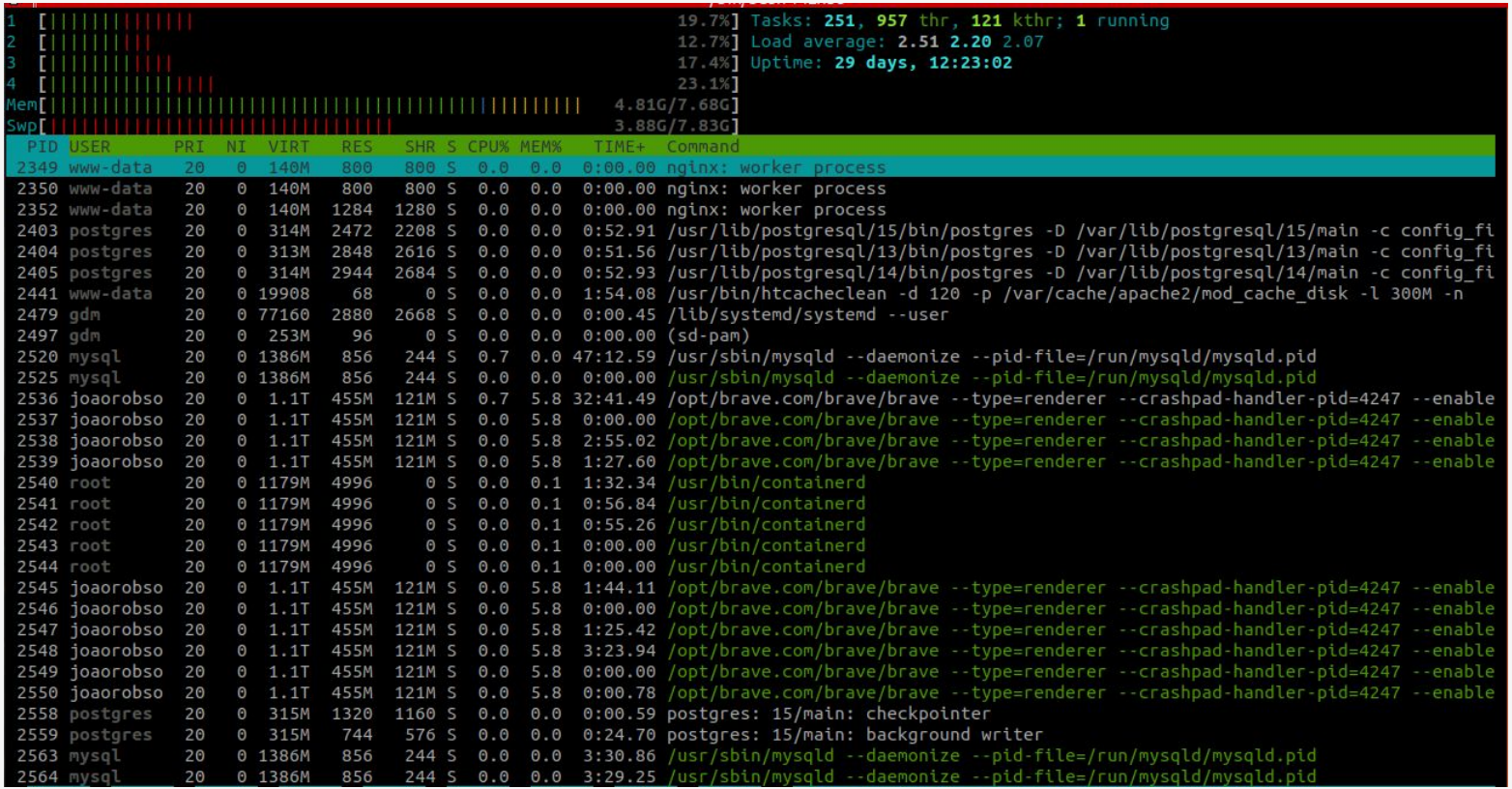
- A **Unidade Central de Processamento** (**CPU**, na sigla em inglês) é o componente principal do computador, responsável por executar as instruções e processar dados;
- Muitas vezes, usa-se o termo **processador** para designar de forma genérica qualquer forma de CPU;
- Um **núcleo** é uma unidade de execução dentro de uma CPU capaz de executar **um único processo por vez**;



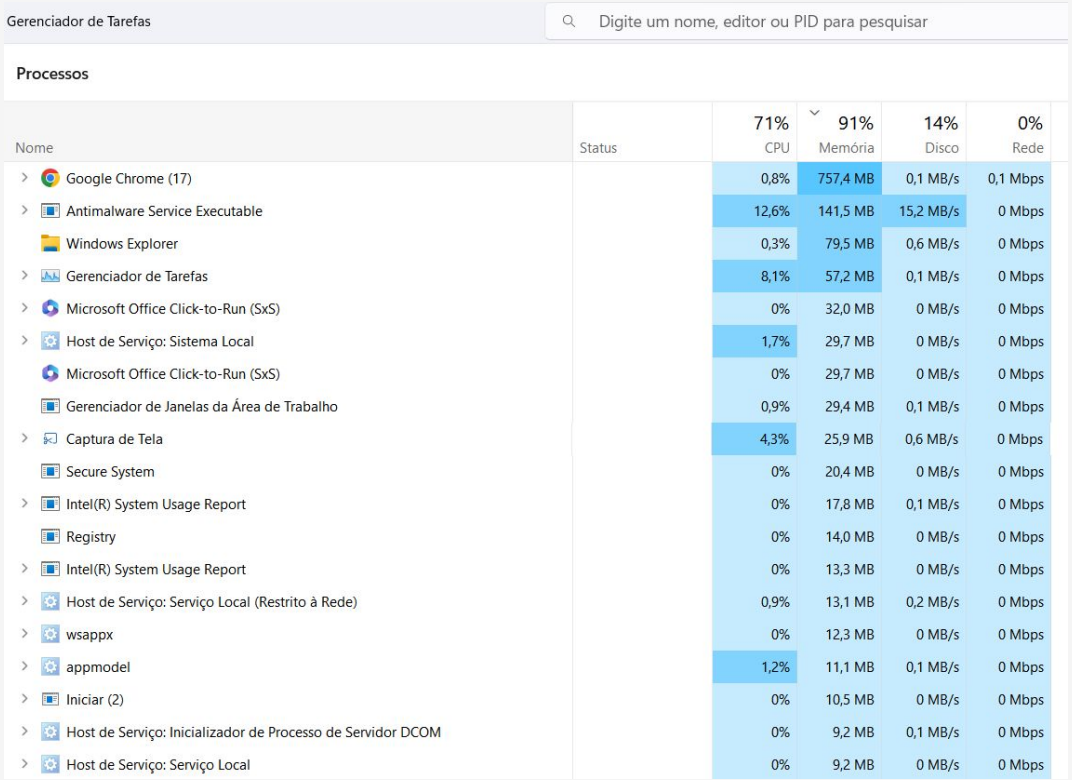
Processos

- Uma instância de um programa em execução por um computador é chamado de **processo;**
- Cada processo tem um número identificador (**PID**) e espaços de endereçamento independentes;
- O computador armazena uma tabela, acessada pelo PID, com informações sobre cada processo:
  - Contador de programa;
  - Ponteiro de pilha;
  - Alocação de memória;
  - Estado dos arquivos abertos;
  - Dados sobre **escalonamento**, permitindo reiniciar processo no futuro.

Processos - htop (Linux)



Processos - Gerenciador de tarefas (Win.)

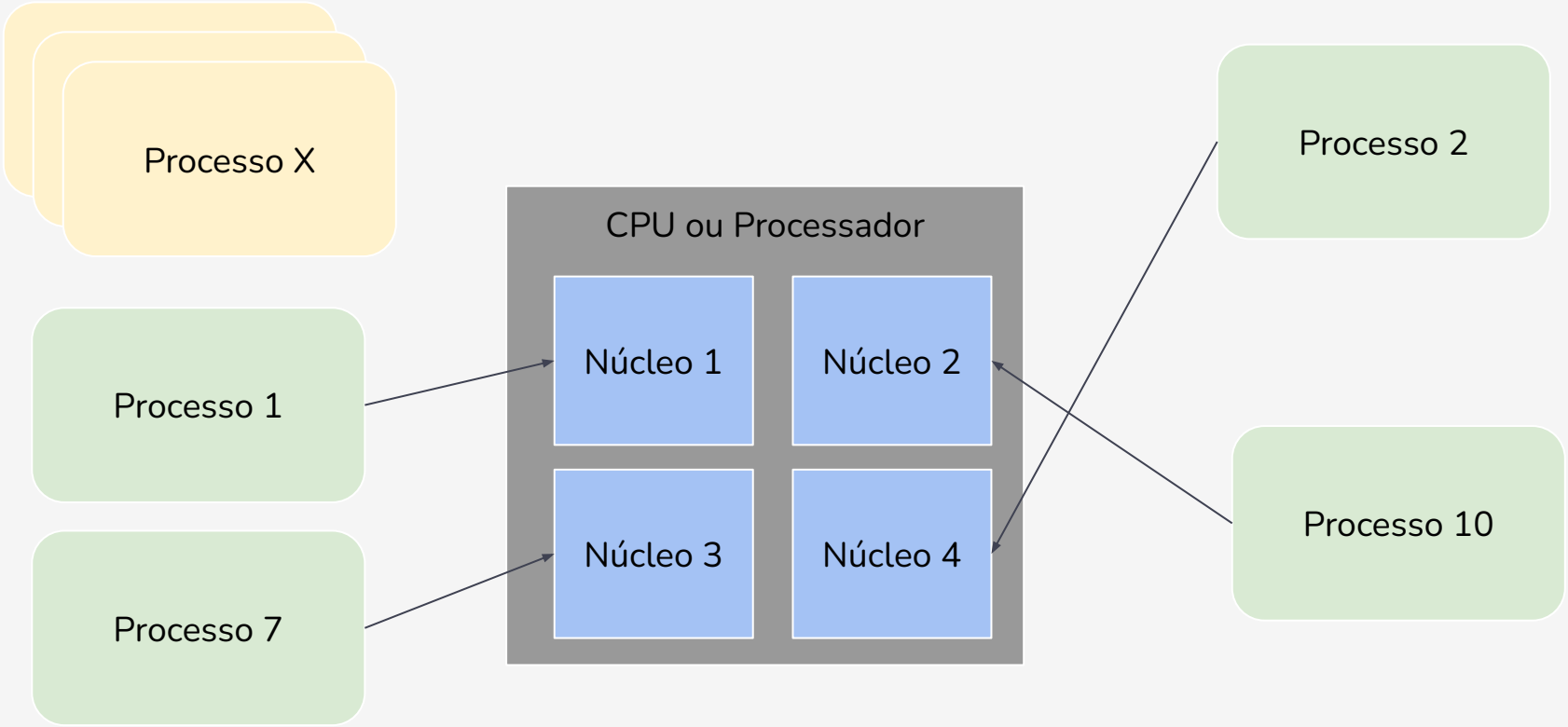


Nome	Status	71% CPU	91% Memória	14% Disco	0% Rede
Google Chrome (17)		0,8%	757,4 MB	0,1 MB/s	0,1 Mbps
Antimalware Service Executable		12,6%	141,5 MB	15,2 MB/s	0 Mbps
Windows Explorer		0,3%	79,5 MB	0,6 MB/s	0 Mbps
Gerenciador de Tarefas		8,1%	57,2 MB	0,1 MB/s	0 Mbps
Microsoft Office Click-to-Run (SxS)		0%	32,0 MB	0 MB/s	0 Mbps
Host de Serviço: Sistema Local		1,7%	29,7 MB	0 MB/s	0 Mbps
Microsoft Office Click-to-Run (SxS)		0%	29,7 MB	0 MB/s	0 Mbps
Gerenciador de Janelas da Área de Trabalho		0,9%	29,4 MB	0,1 MB/s	0 Mbps
Captura de Tela		4,3%	25,9 MB	0,6 MB/s	0 Mbps
Secure System		0%	20,4 MB	0 MB/s	0 Mbps
Intel(R) System Usage Report		0%	17,8 MB	0,1 MB/s	0 Mbps
Registry		0%	14,0 MB	0 MB/s	0 Mbps
Intel(R) System Usage Report		0%	13,3 MB	0 MB/s	0 Mbps
Host de Serviço: Serviço Local (Restrito à Rede)		0,9%	13,1 MB	0,2 MB/s	0 Mbps
wsappx		0%	12,3 MB	0 MB/s	0 Mbps
appmodel		1,2%	11,1 MB	0,1 MB/s	0 Mbps
Iniciar (2)		0%	10,5 MB	0 MB/s	0 Mbps
Host de Serviço: Inicializador de Processo de Servidor DCOM		0%	9,2 MB	0,1 MB/s	0 Mbps
Host de Serviço: Serviço Local		0%	9,2 MB	0 MB/s	0 Mbps

Processos

- Uma instância de um programa em execução é chamado de **processo;**
- Cada processo tem um identificador (**PID**) e espaços de endereçamento independentes;
- O computador armazena uma tabela, acessada pelo PID, com informações de cada processo:
  - Contador de programa;
  - Ponteiro de pilha;
  - Alocação de memória;
  - Estado dos arquivos abertos;
  - Dados sobre **escalonamento**, permitindo reiniciar processo no futuro.’

Processador - Múltiplos processos

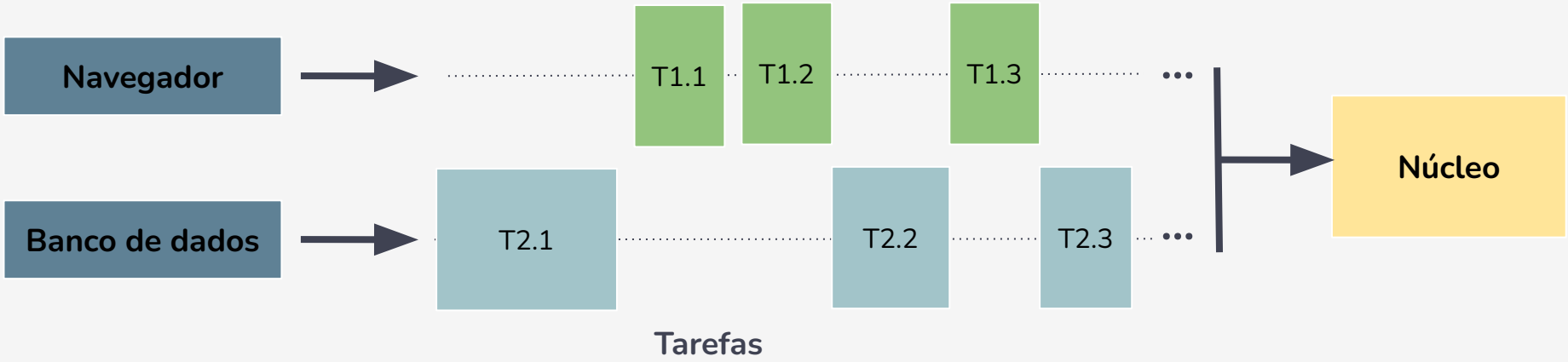


Processos - Escalonamento

- Cada núcleo do processador (“CPU”) só consegue executar um processo de cada vez;
- Um SO multitarefa permite que processos se alternem no uso do processador por meio de um **escalonamento**:
  - Qual processo deve ser executado agora?
  - **Algoritmo de escalonamento:** Estabelece sequência de execução de processos (fila, prioridades, prazos, entre outros critérios).
- Os SOs adotam políticas para garantir que todos os processos sejam atendidos de forma justa e equitativa.
- Processos do SO e aplicações críticas têm prioridade maior;
- Processos podem se comunicar por meio de mecanismos de comunicação.

Processos - Escalonamento

- Exemplo: abrir um navegador + executar um banco de dados

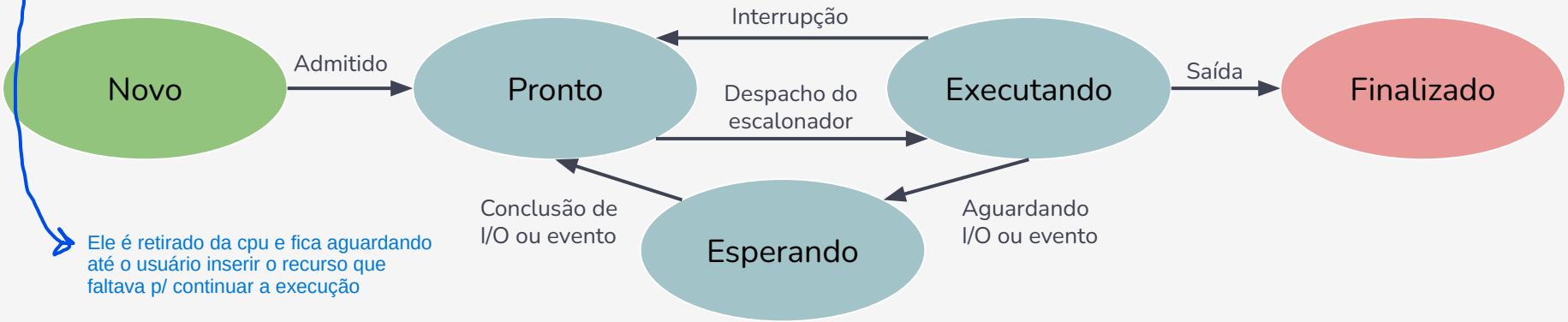


Processos - Estados

- Processos em geral são autossuficientes, mas podem necessitar acessar recursos externos às vezes (dados no disco, terminais, etc.) ou se comunicar com outros processos;
- Processo ocioso/aguardando evento -> processo esperando ou bloqueado;

Processos - Estados

- Novo: o processo está na fase de criação.
- Pronto: o processo possui os recursos disponíveis necessários para ser executado, mas a CPU não está atualmente processando suas instruções.
- Executando: a CPU está processando as instruções deste processo.
- Esperando: o processo não pode ser executado no momento, pois está aguardando que algum recurso esteja disponível ou que algum evento ocorra
- Finalizado: o processo foi concluído.



Threads

- “Fluxos” ou “fios”, em inglês;
  - Fluxos de execução dentro de um processo;
  - Processos “leves” dentro de processos “reais” ou “pesados”;
- Permite a execução de um mesmo trecho de código “simultaneamente”;
- Um processo pode conter 1 ou mais *threads*;
- *Threads* dentro de um processo compartilham recursos, memória e espaços de endereçamento;
  - Compartilham o código executável, as variáveis globais e estáticas, a pilha e o heap;
  - Possuem contador de programa, *stack*/pilha e conjunto de registradores próprios;
  - Podem trocar dados entre si e coordenar suas atividades.

Todo processo - mín. 1 thread

ex.: lista com as notas da turma. iremos encontrar a maior nota:
- com 1 fluxo de execução, ou;
- podemos dividir a lista e cada partição atribuir a uma thread.

Conclusão>: Temos um grande problema, repartimos ele e atribuímos cada parte a uma thread

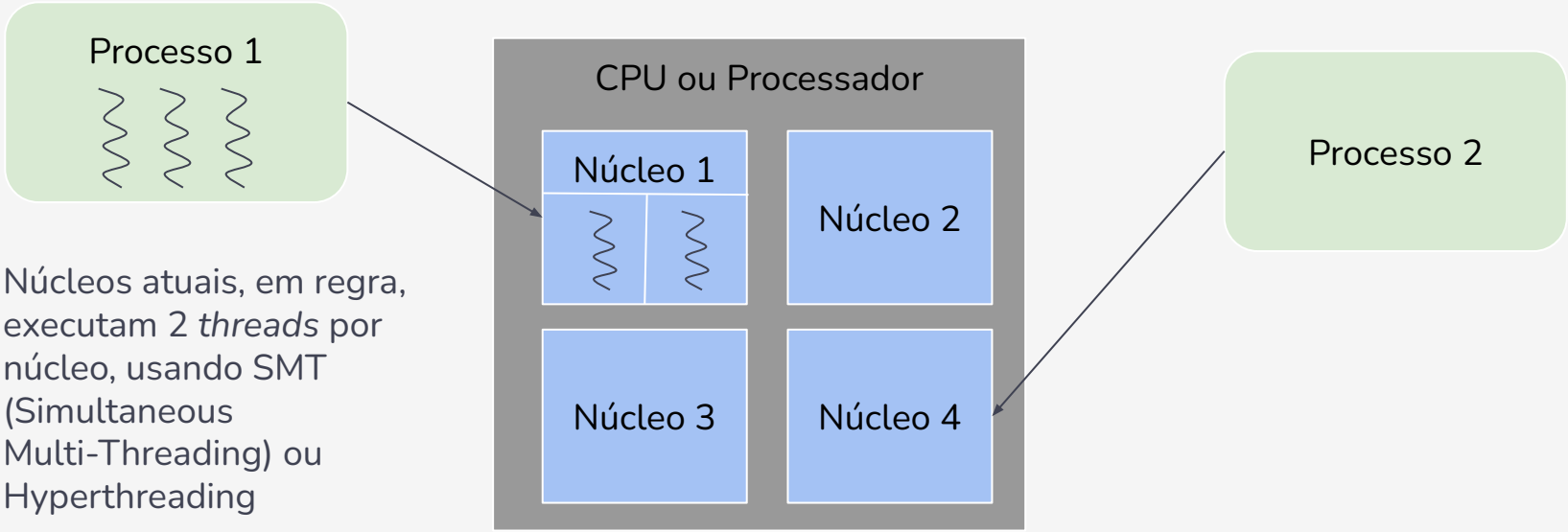
Threads - Contextos

processos são completamente independentes, já threads de um mesmo processo não são independentes... elas compartilham...

No entanto elas tbm têm alguns recursos próprios.

- Durante a alternância entre duas threads de um mesmo processo no uso do processador, ocorre uma **troca de contexto parcial**:
  - O contador de programa, os registradores e a pilha são salvos.
  - Mais rápida do que uma troca de contexto entre processos.
- Uma troca de contexto completa é necessária quando uma thread de um processo, que não estava em execução, assume o processador.

Processador - Múltiplas *threads*



Núcleos atuais, em regra, executam 2 *threads* por núcleo, usando SMT (Simultaneous Multi-Threading) ou Hyperthreading

Threads - Prática (Java)

- Boa parte das linguagens de programação dão suporte à criação e execução de *threads* explicitamente;
- Na disciplina, usaremos **Java**;
- Java é executado em um sistema operacional via uma “máquina virtual” (JVM):
  - JVM executada como processo pelo SO;
  - Permite implementar *threads* internamente;
  - Executa método main como *thread* principal, possibilitando criar e executar novas *threads* a partir dela;
  - Threads de uma mesma JVM compartilham memória, portas de comunicação, arquivos e outros recursos.

Threads - Prática (Java)

- Java oferece suporte a dois tipos de *threads*:
  - *Green threads*:
    - *Threads* gerenciadas pela JVM, sem envolver o SO;
    - Escalonamento feito pela JVM;
  - ***Native threads*:**
    - *Threads* gerenciadas pelo SO;
    - Usadas em SOs com suporte a *threads* (Linux, Windows, etc.);
    - Possuem acesso aos recursos do sistema operacional e geralmente oferecem um melhor desempenho e escalabilidade em sistemas com múltiplos núcleos de processamento.

Threads - Prática (Java)

- Para declarar e utilizar *threads* em Java, usamos a classe Thread:
  - Podemos declarar uma classe representando uma *thread* herdando da classe nativa e implementando o método *run()*:

```
class Ola extends Thread {
    public void run() {
        System.out.println("Ola");
    }
}

public class ExemploThread {
    public static void main(String[] args) {
        Ola thread = new Ola();
        thread.start();
    }
}
```

Threads - Prática (Java)

Relembrar formas de interface

- Para declarar e utilizar *threads* em Java, usamos a classe Thread:
  - Podemos declarar uma classe representando uma *thread* implementando a interface *Runnable* e sobrescrevendo o método *run()*:

```
class Ola implements Runnable {
    public void run() {
        System.out.println("Ola!");
    }
}

public class ExemploRunnable {
    public static void main(String[] args) {
        Thread thread = new Thread(new Ola());
        thread.start();
    }
}
```

Threads - Prática (Java)

- Classe *Thread:*
  - Construtores: Thread(Runnable), Thread(Runnable, String);
  - Inicializar *thread*: start();
  - Obter nome da *thread*: getName();
  - Determinar nome da *thread*: setName();
  - Obter *thread* sendo executada: Thread.currentThread();
  - Esperar pelo fim da execução da *thread*: join();
  - Obter estados da *thread*: isAlive(), isInterrupted()
  - Liberar o processador: yield()
  - Destruir a thread: destroy()

Threads - Prática (Java)

- Prioridade da *thread:*
  - Valor de 1 a 10; 5 é o default padrão
  - A *thread* herda a prioridade da thread que a criou
  - Modificada com setPriority(int);
  - Obtida com getPriority();
  - Escalonamento: *Threads* com prioridades iguais são escalonadas em round-robin, com cada uma ativa durante um *quantum*.

Threads - Exercícios

- Usando *threads*:
  1. Calcule as 4 operações aritméticas básicas com 2 números;
  2. Identifique o maior valor de uma lista;
  3. Calcule a soma dos números primos dentro de um intervalo determinado.

Referências

- Tanenbaum, Andrew S. Sistemas operacionais modernos / Andrew S. Tanenbaum, Herbert Bos; tradução Jorge Ritter; revisão técnica Raphael Y. de Camargo. – 4. ed. – São Paulo: Pearson Education do Brasil, 2016.
- BELL, John T. Operating Systems - Course Notes. University of Illinois, Chicago. Disponível em: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/