

PROGRAMAÇÃO PARALELA ARQUITETURAS PARALELAS

Marco A. Zanata Alves



A close-up portrait of Gordon Moore, an elderly man with white hair and glasses, smiling broadly. He is wearing a dark suit and a light blue shirt.

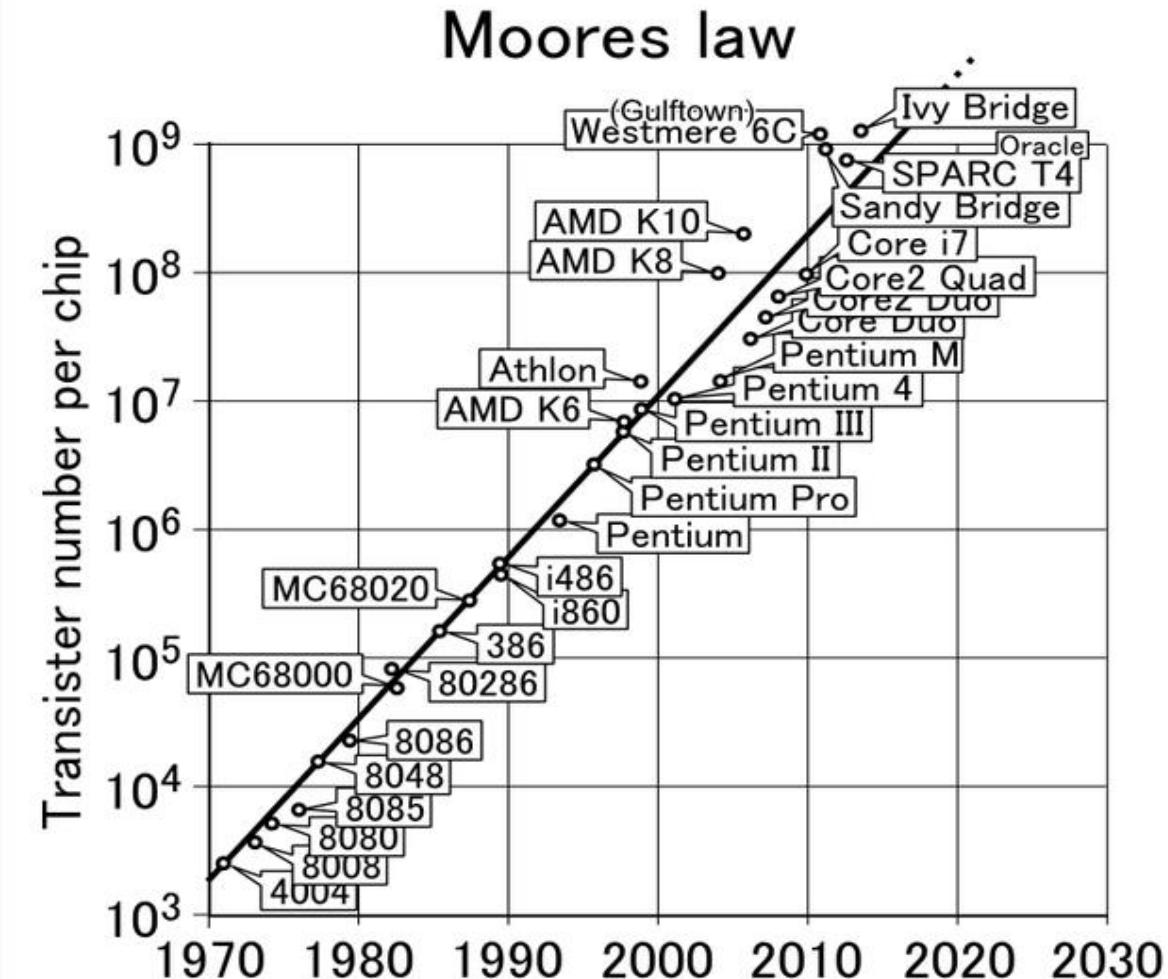
Gordon M

(★192
co-fundador

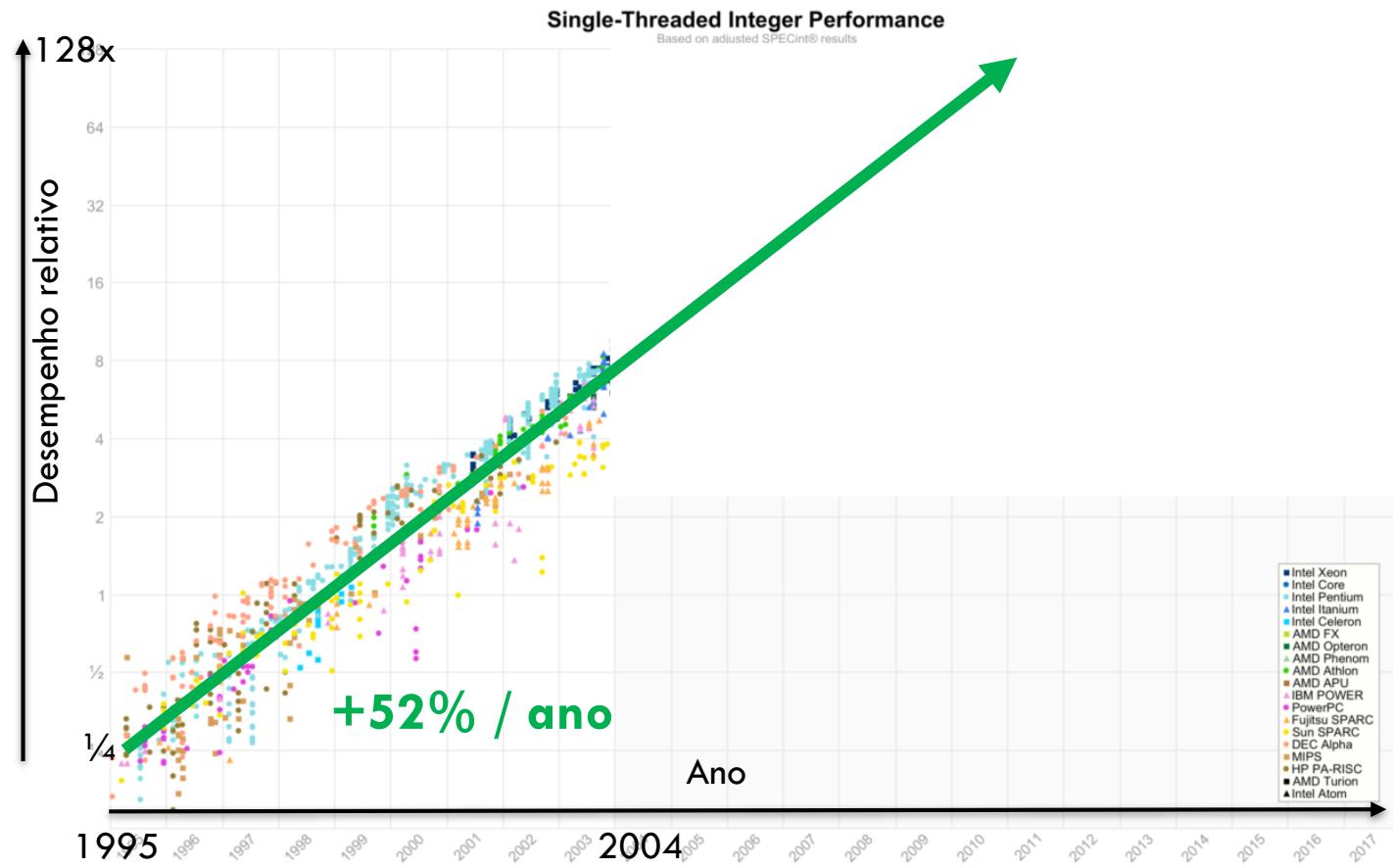
- Lei de Moore prevista em 1965
- A densidade dos semicondutores deve dobrar a cada 18 meses

Fontes: <http://projetodraft.com/selecao-draft-para-a-melhor-tecnologia-de-semicondutor>
<http://www.cringely.com/2013/10/15/breakthroughs-in-semiconductor-technology>

Fontes: <http://projetodraft.com/selecao-draft-lei-de-moore-2/>
<http://www.cringely.com/2013/10/15/breaking-moores-law/>



DESEMPENHO AO LONGO DOS ANOS



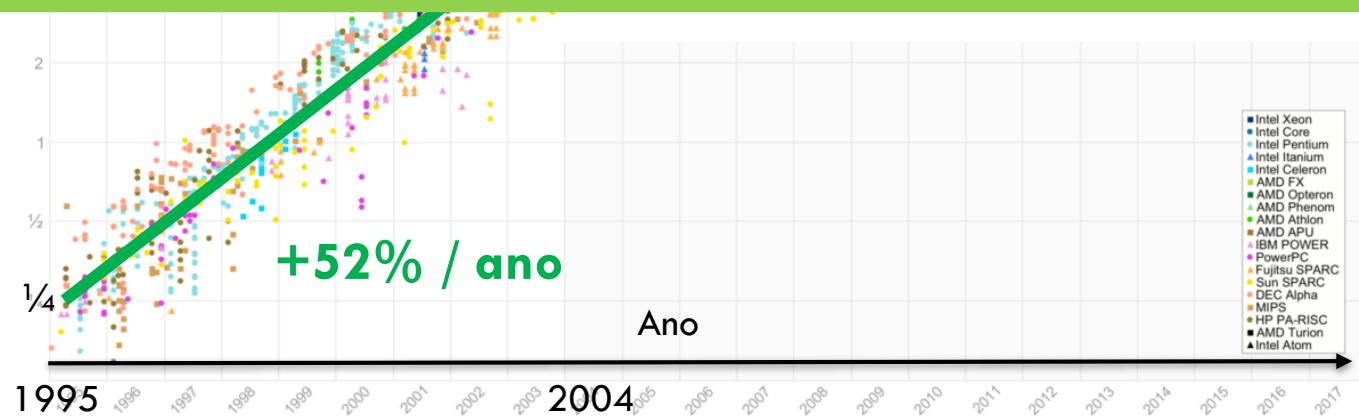
DESEMPENHO AO LONGO DOS ANOS

↑
128x

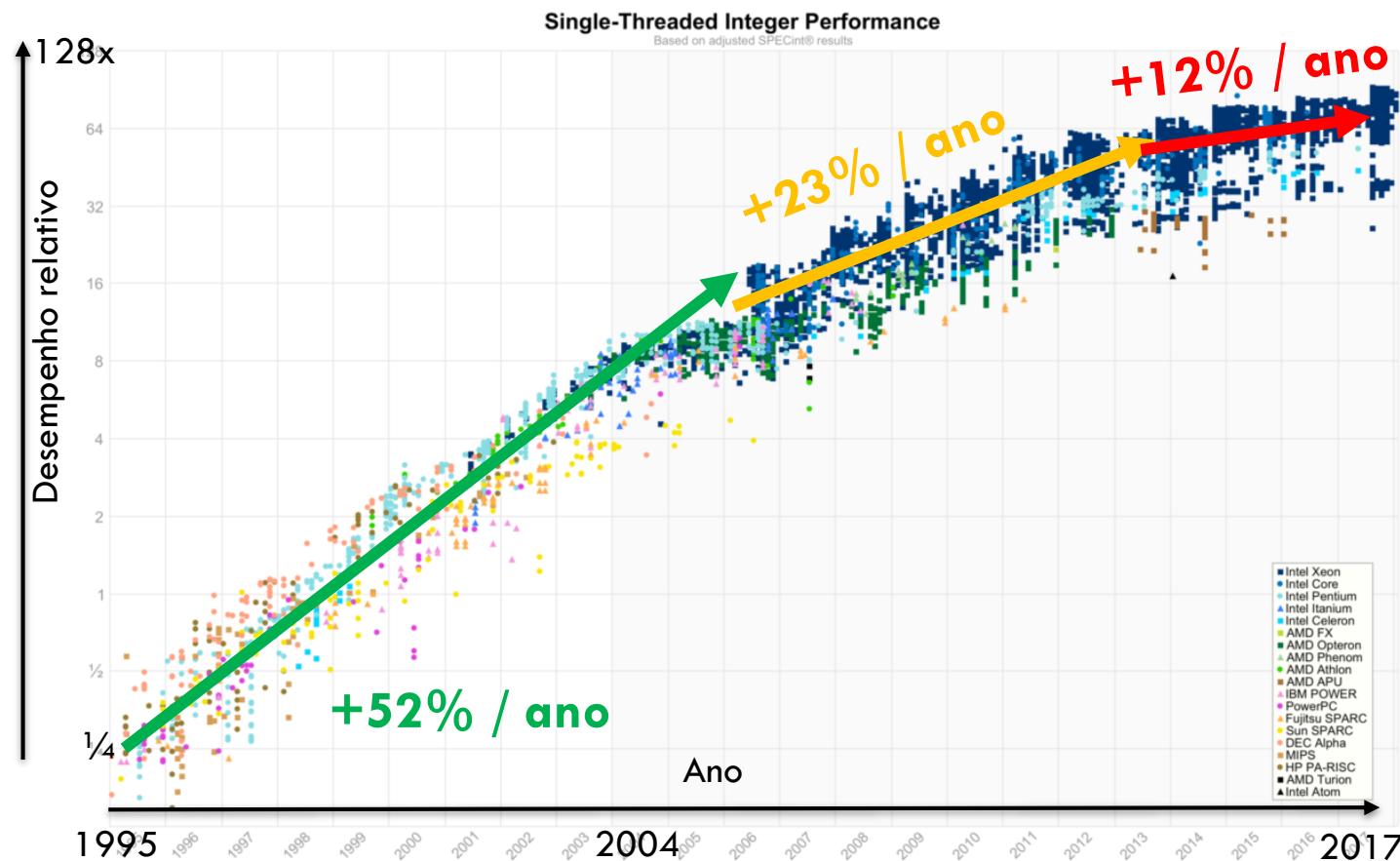
Single-Threaded Integer Performance

Based on adjusted SPECint® results

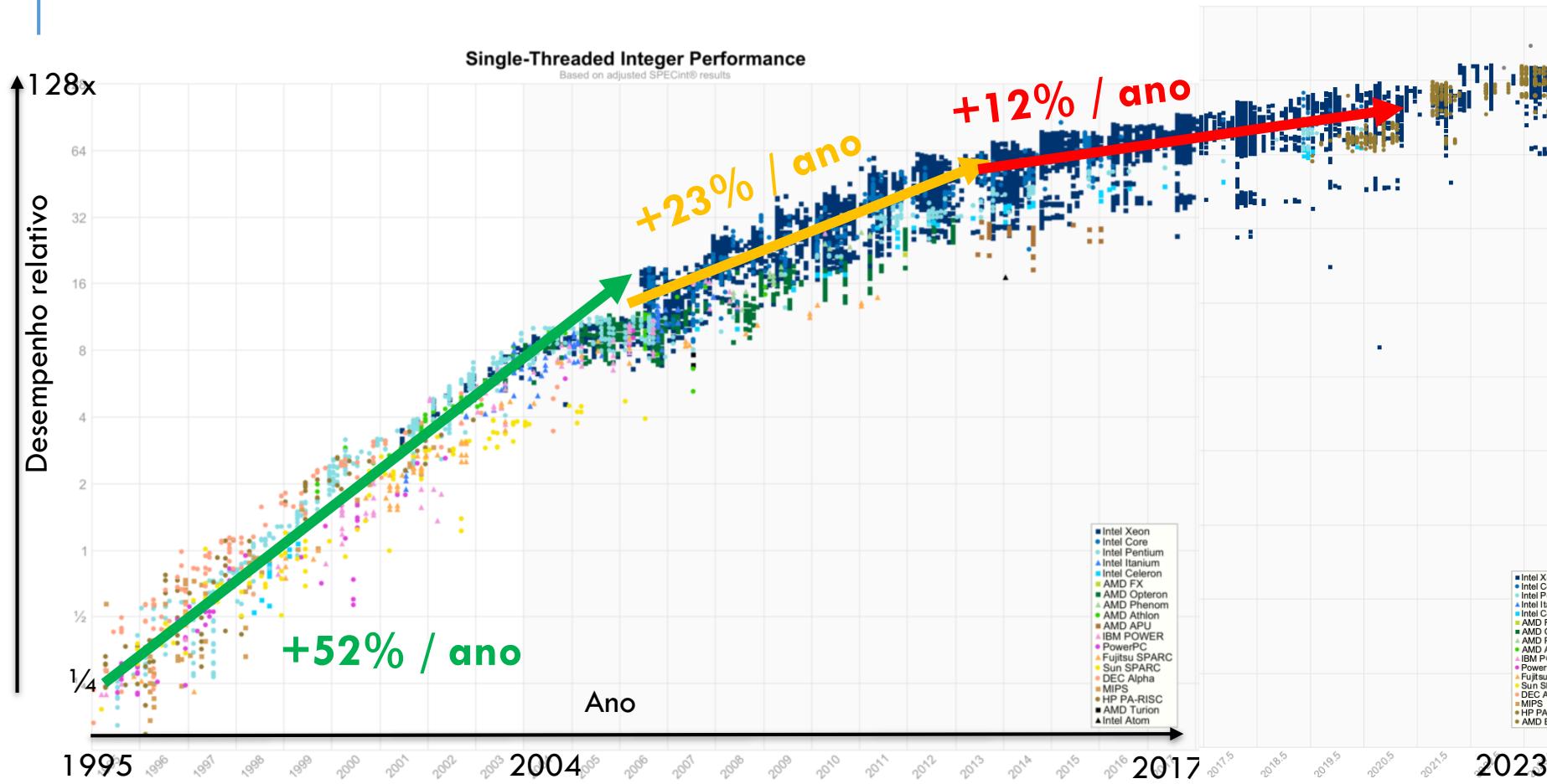
Escreva seu programa como desejar e os **Gênios-do-Hardware** vamos cuidar do desempenho de forma “mágica”.



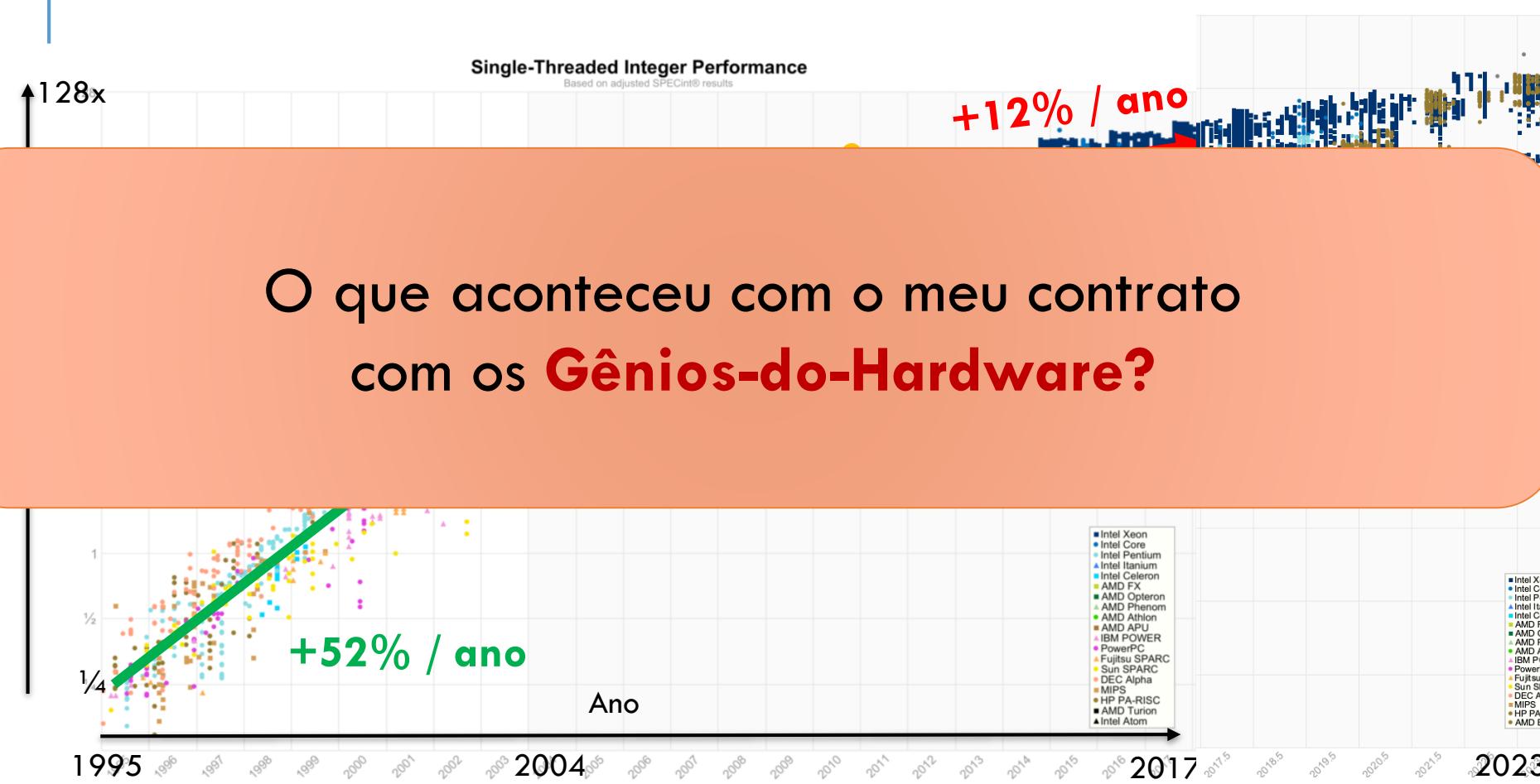
DESEMPENHO AO LONGO DOS ANOS

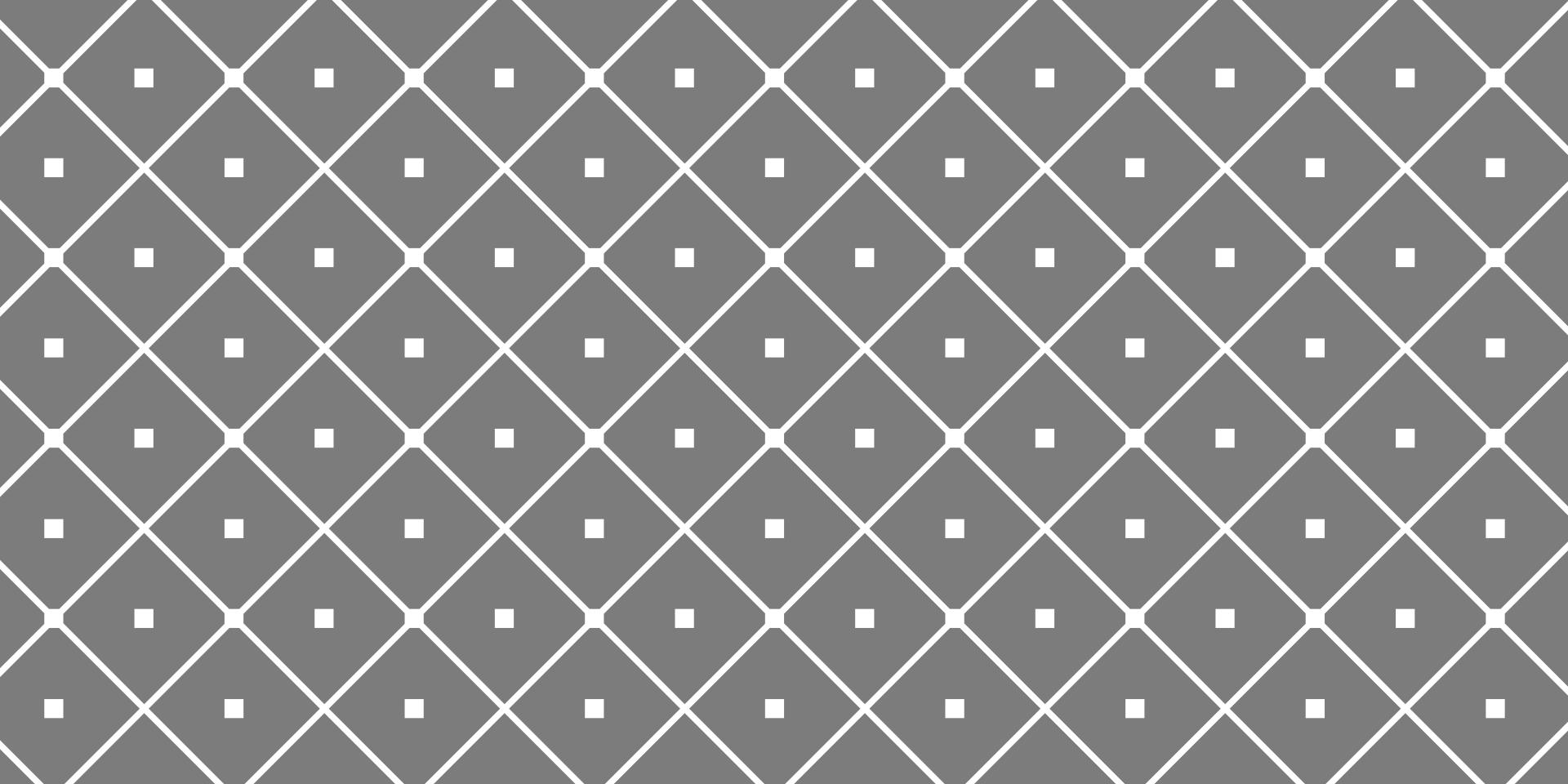


DESEMPENHO AO LONGO DOS ANOS



DESEMPENHO AO LONGO DOS ANOS





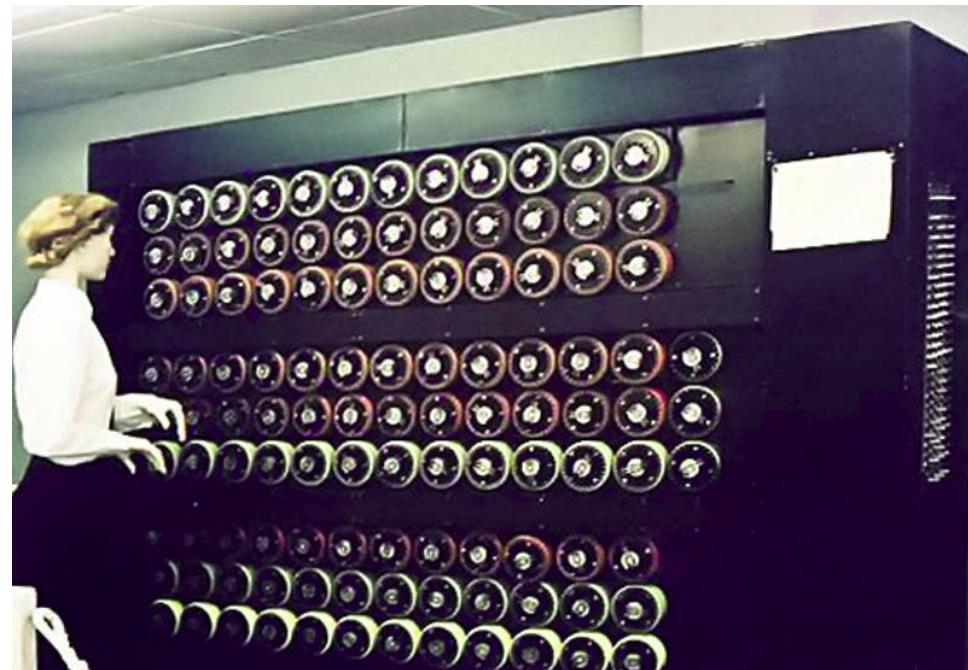
MICROPROCESSADORES: PASSADO, PRESENTE E FUTURO



Computer
History
Museum

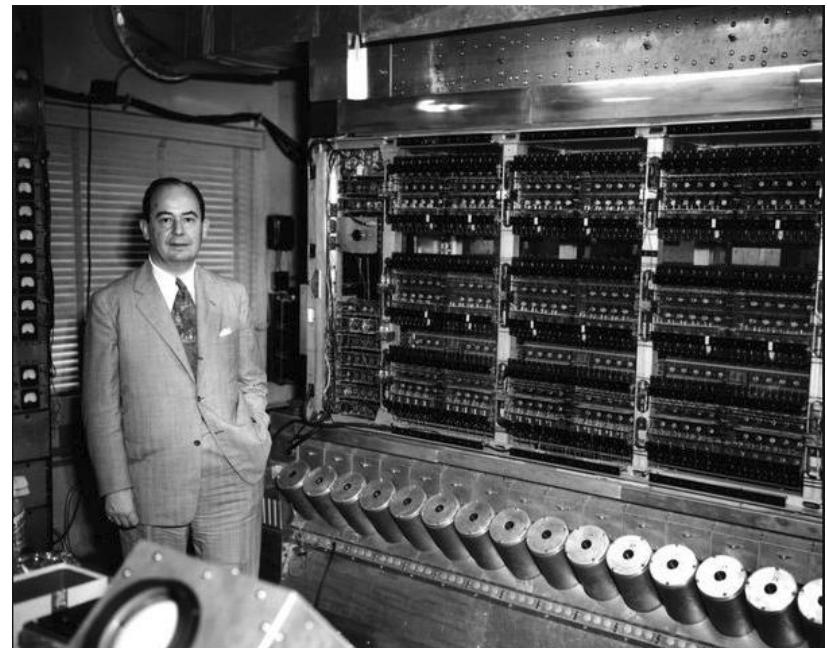
BREVE HISTÓRIA DOS COMPUTADORES

1941 – Alan Turing construiu uma máquina eletromecânica para apenas decriptografar as comunicações militares (Bombe) baseadas no ENIGMA



BREVE HISTÓRIA DOS COMPUTADORES

1945 - John von Neumann
descreve a arquitetura de um
programa de computador
armazenado (**propósito-**
geral)



BREVE HISTÓRIA DOS COMPUTADORES

1941 - Konrad Zuse
desenvolveu e construiu o
**primeiro computador digital
binário do mundo**, o Z1.

**1945 - Ele finaliza a primeira
linguagem de programação
algorítmica** (Plankalkül - Plan
Calculus), com o objetivo de
criar soluções para problemas
gerais.



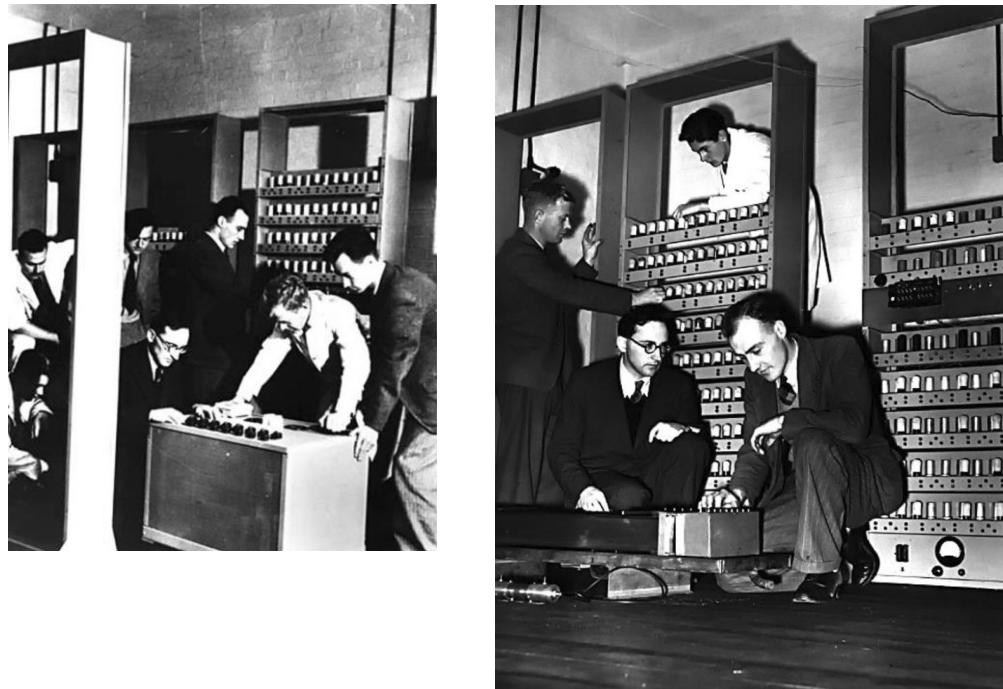
BREVE HISTÓRIA DOS COMPUTADORES

1946 – MIT lança um computador construído **usando partes eletrônica invés de eletromecânicas** (ENIAC).



BREVE HISTÓRIA DOS COMPUTADORES

1949 – Cambridge constrói um computador usando **tubos a vácuo** e linha de atraso de memória em mercúrio (EDSAC)



BREVE HISTÓRIA DOS COMPUTADORES

1956 - MIT constrói o primeiro computador de propósito-geral **usando transistores** (TX-0).



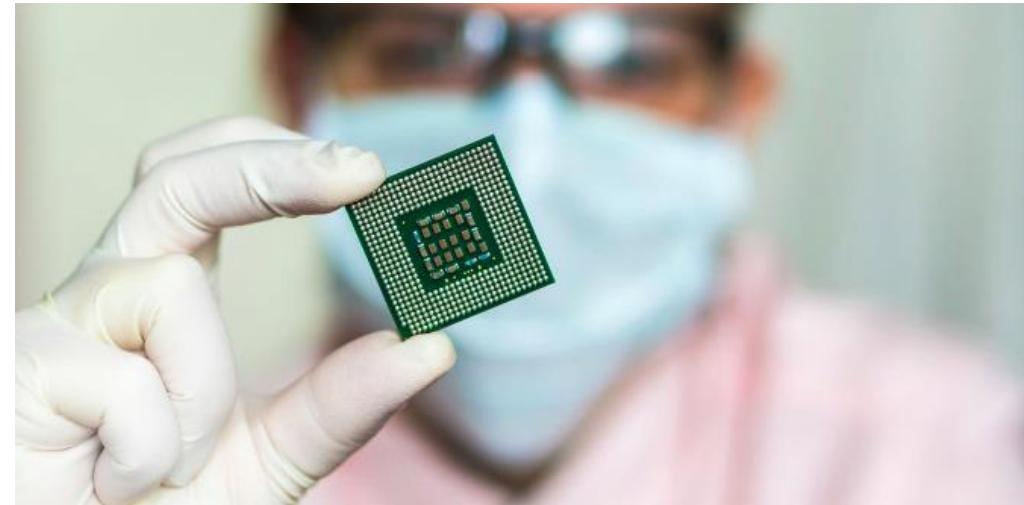
BREVE HISTÓRIA DOS COMPUTADORES

1970 – Intel introduz o
primeiro microprocessador com
2,250 transistores (Intel 4004)



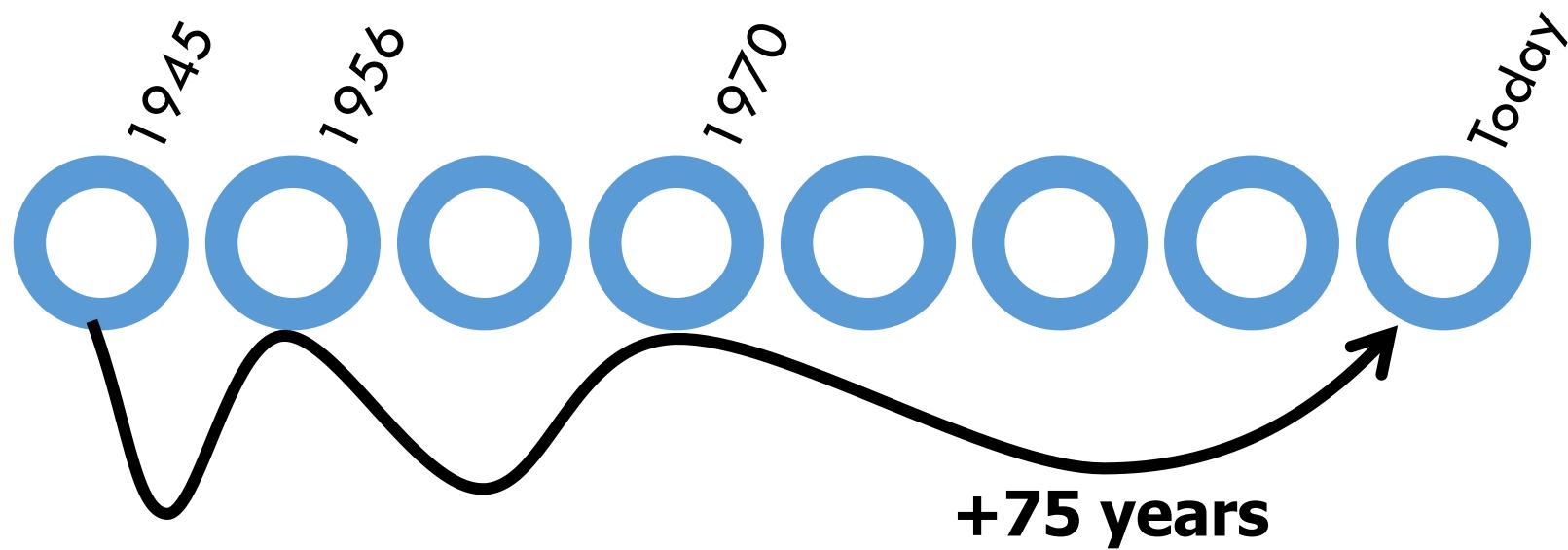
Engenheiro Ted Hoff apresenta o microprocessador 4004 em 1971

BREVE HISTÓRIA DOS COMPUTADORES

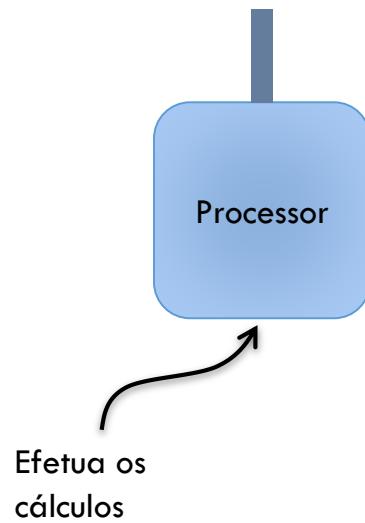


Atualmente – Intel/AMD/NVidia vendem processadores com +8 bilhões de transistores

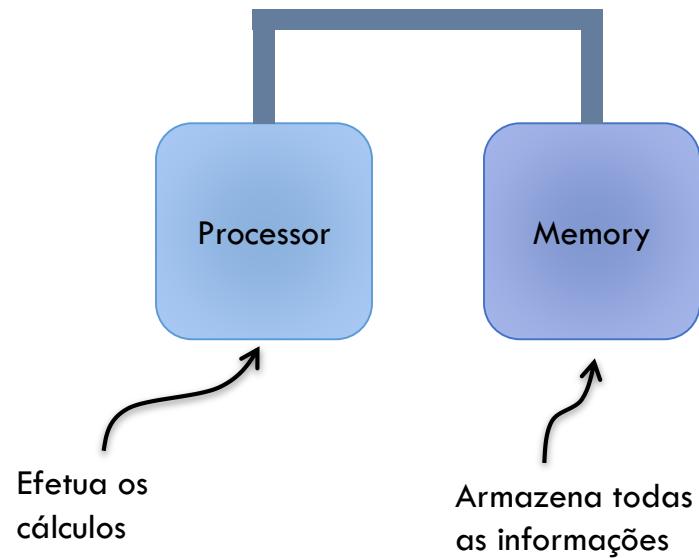
DESDE 1945 SEGUIMOS O CONCEITO DE
VON NEUMANN



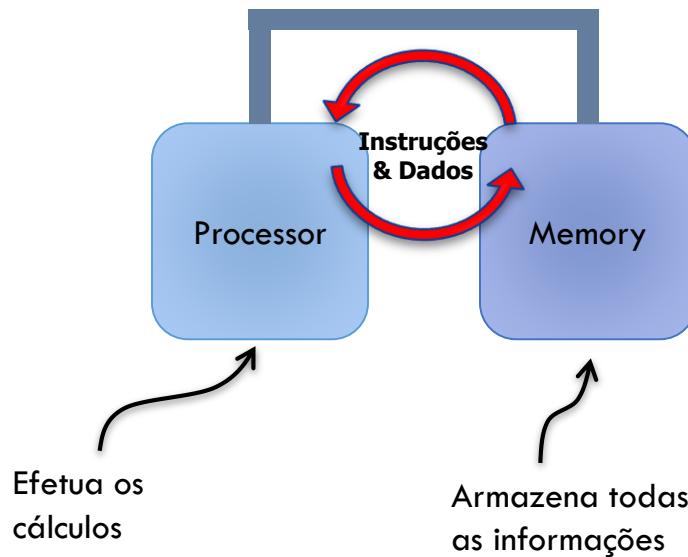
ARQUITETURA DE VON NEUMANN



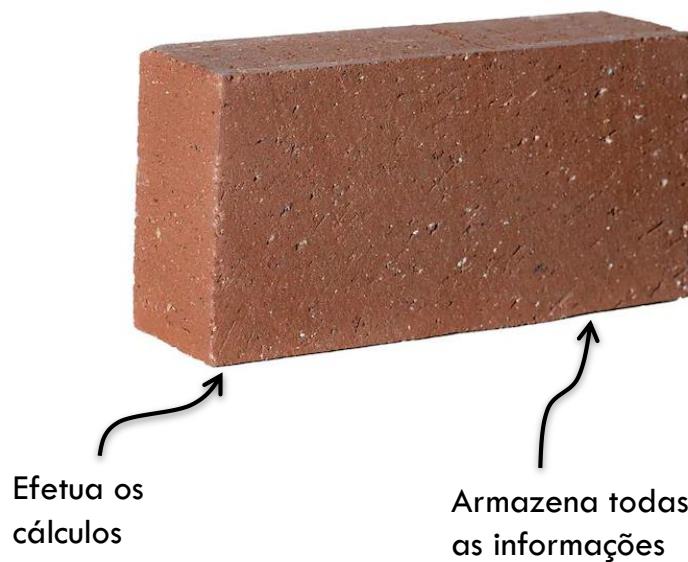
ARQUITETURA DE VON NEUMANN



ARQUITETURA DE VON NEUMANN



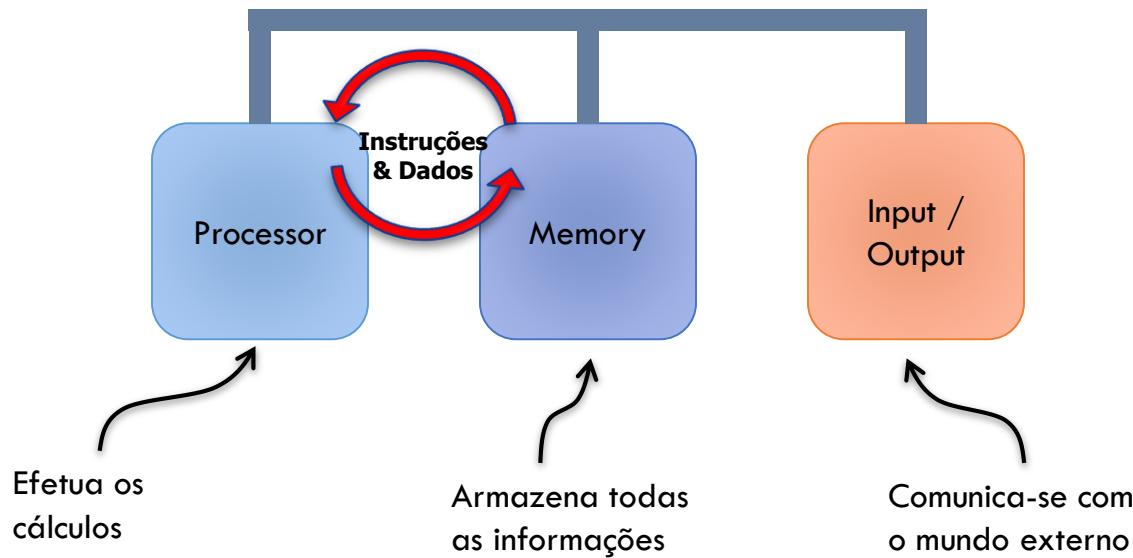
O CONCEITO DE UM TIJOLO

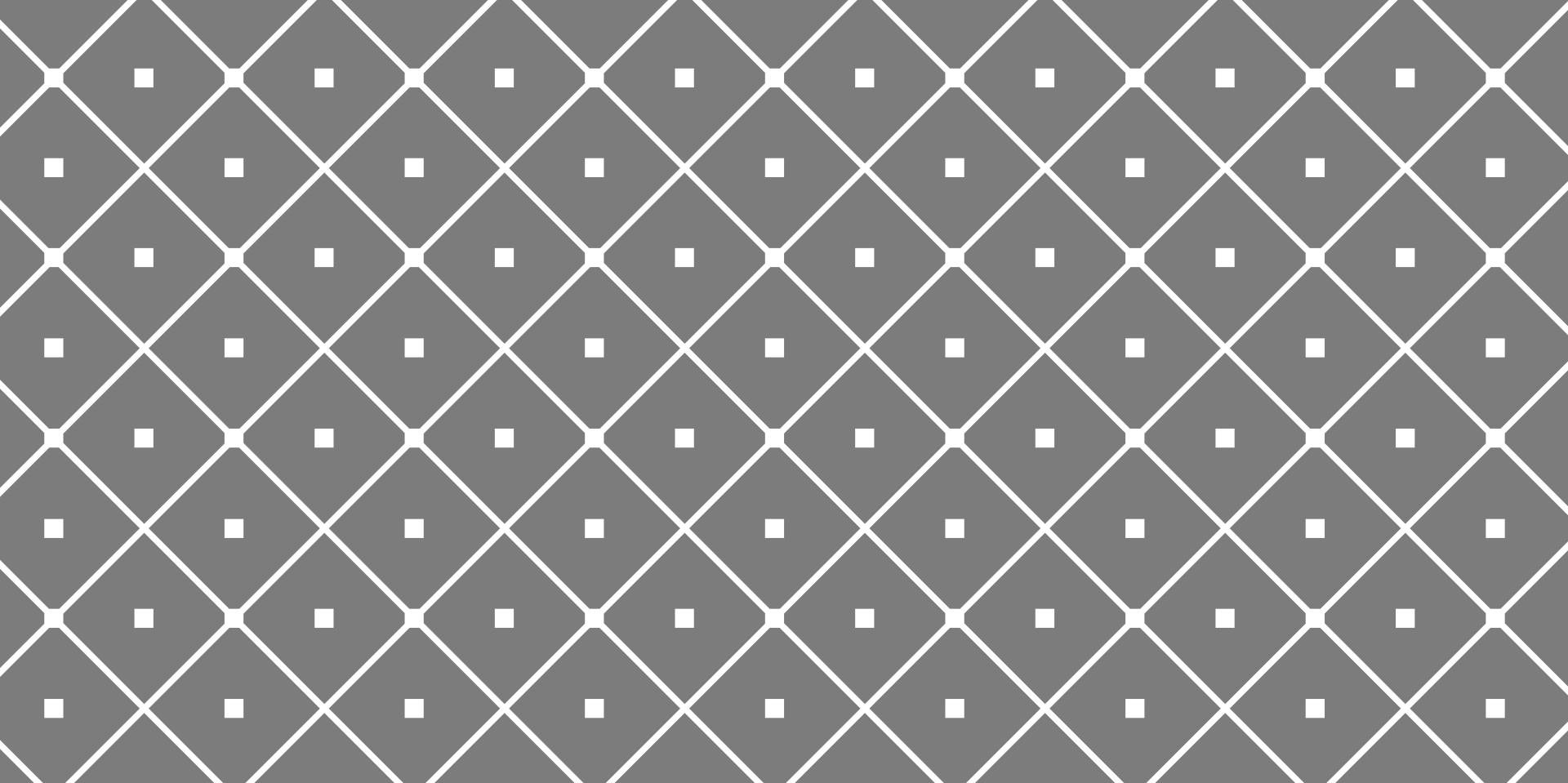


Efetua os
cálculos

Armazena todas
as informações

ARQUITETURA DE VON NEUMANN





CLASSIFICANDO OS COMPUTADORES

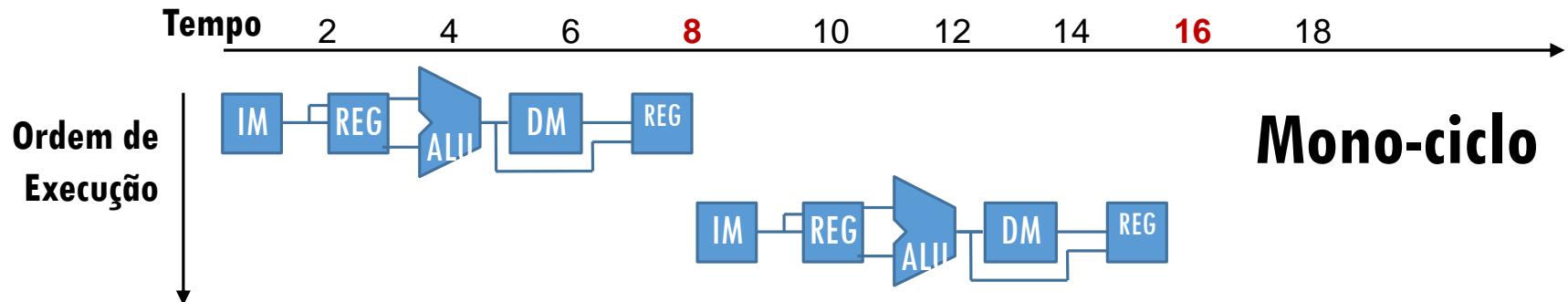
TAXONOMIA DE FLYNN

*[Flynn, 1972]

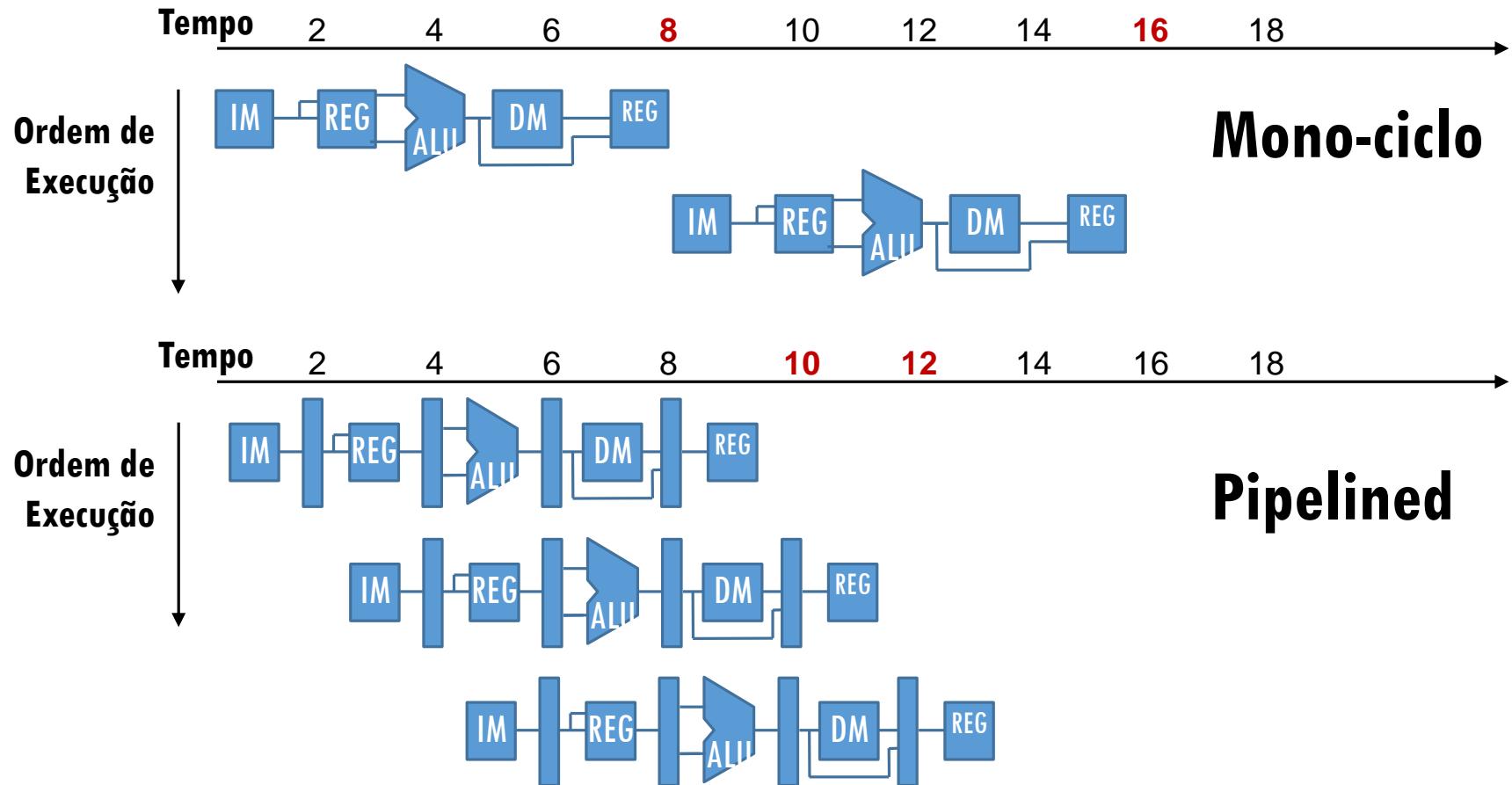
**[De Rose,
2003]

*	SD (Single Data)	MD (Multiple Data)
SI (Single Instruction)	<ul style="list-style-type: none">** <p>SISD (Máquinas von Neumann)</p>	<ul style="list-style-type: none">** <p>MISD (Sem representante / Arquiteturas Sistólicas)</p>
MI (Multiple Instruction)	<ul style="list-style-type: none">** <p>SIMD (Máquinas Vetoriais)</p>	<ul style="list-style-type: none">** <p>MIMD (Multiprocessadores e Multicomputadores)</p>

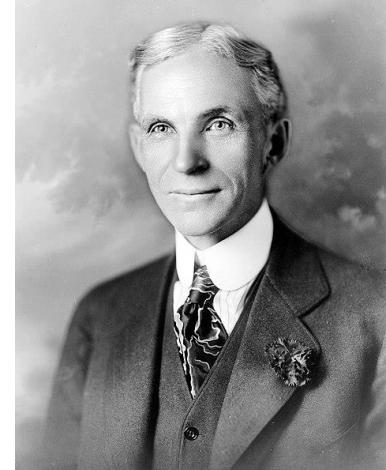
MONOCICLO VS. PIPELINE



MONOCICLO VS. PIPELINE



TRANSFORMANDO O PROCESSADOR EM UMA LINHA DE MONTAGEM



Fordismo, termo criado por Henry Ford, em 1914, refere-se aos sistemas de produção em massa (linha de produção) e gestão idealizados em 1913 pelo empresário americano Henry Ford (1863-1947), fundador da Ford Motor Company.

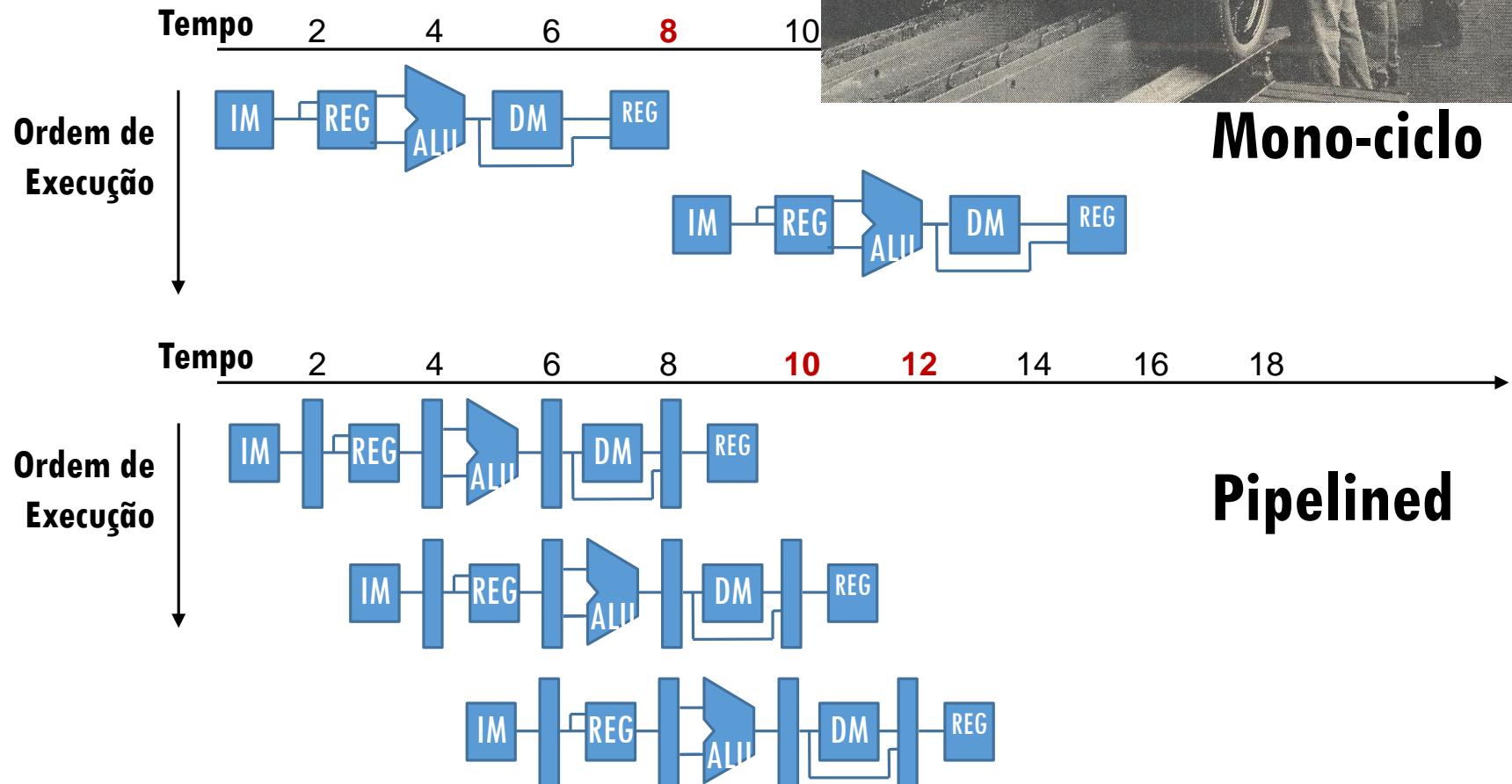
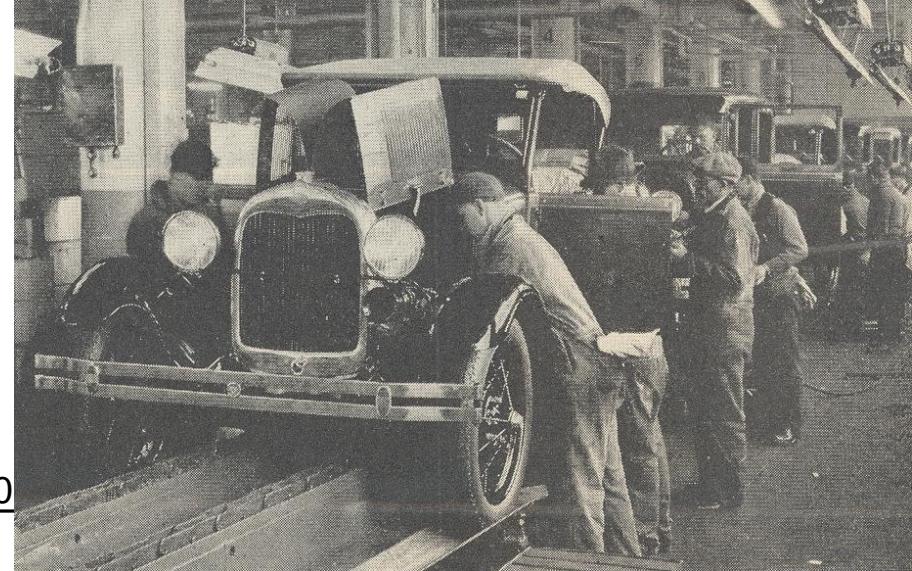
Esse modelo revolucionou a indústria automobilística a partir de 1914, quando Ford introduziu a primeira linha de montagem automatizada.

O aperfeiçoamento da linha de montagem se baseava em:

1. Os veículos eram montados em esteiras rolantes, que se movimentavam enquanto o operário ficava praticamente parado.
2. Cada operário realizava apenas uma operação simples ou uma pequena etapa da produção.
3. Desta forma não era necessária quase nenhuma qualificação dos trabalhadores.

O método de produção fordista permitiu que a Ford produzisse mais de 2 milhões de carros por ano, durante a década de 1920.

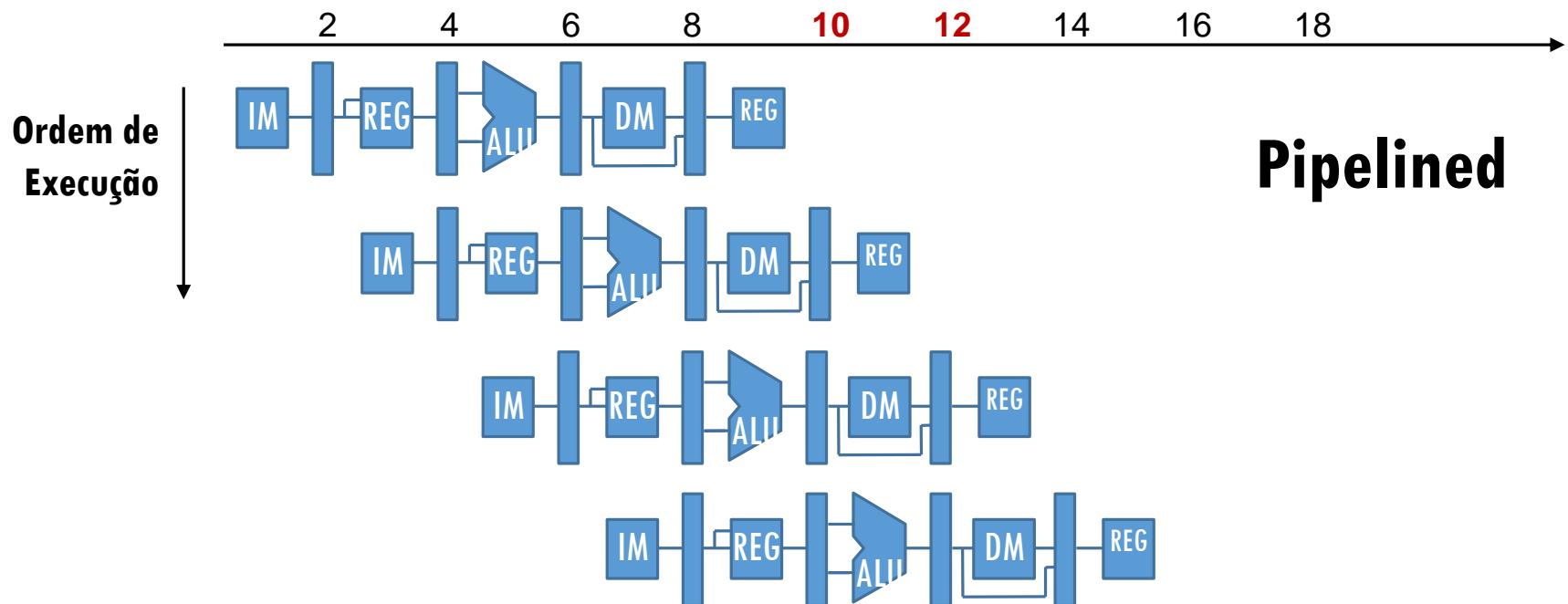
MONOCICLO VS. PIPELINE



SINGLE CORE - PIPELINING

Pipelining **não reduz a latência** de uma instrução única.

Mas **aumenta o throughput (vazão)** de todo workload



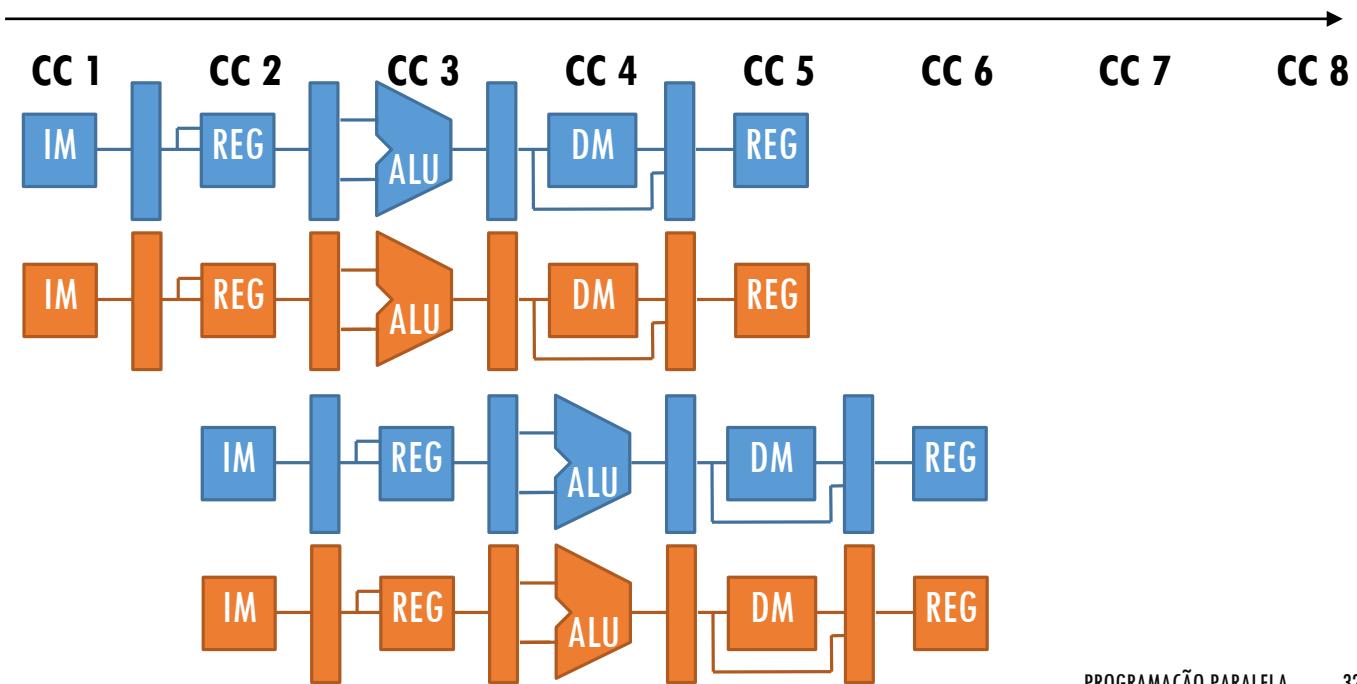
SINGLE CORE - PIPELINE SUPERSCALAR

Pipelining continua ativo

IPC ideal >1

ILP Paralelismo de instruções.

Tempo

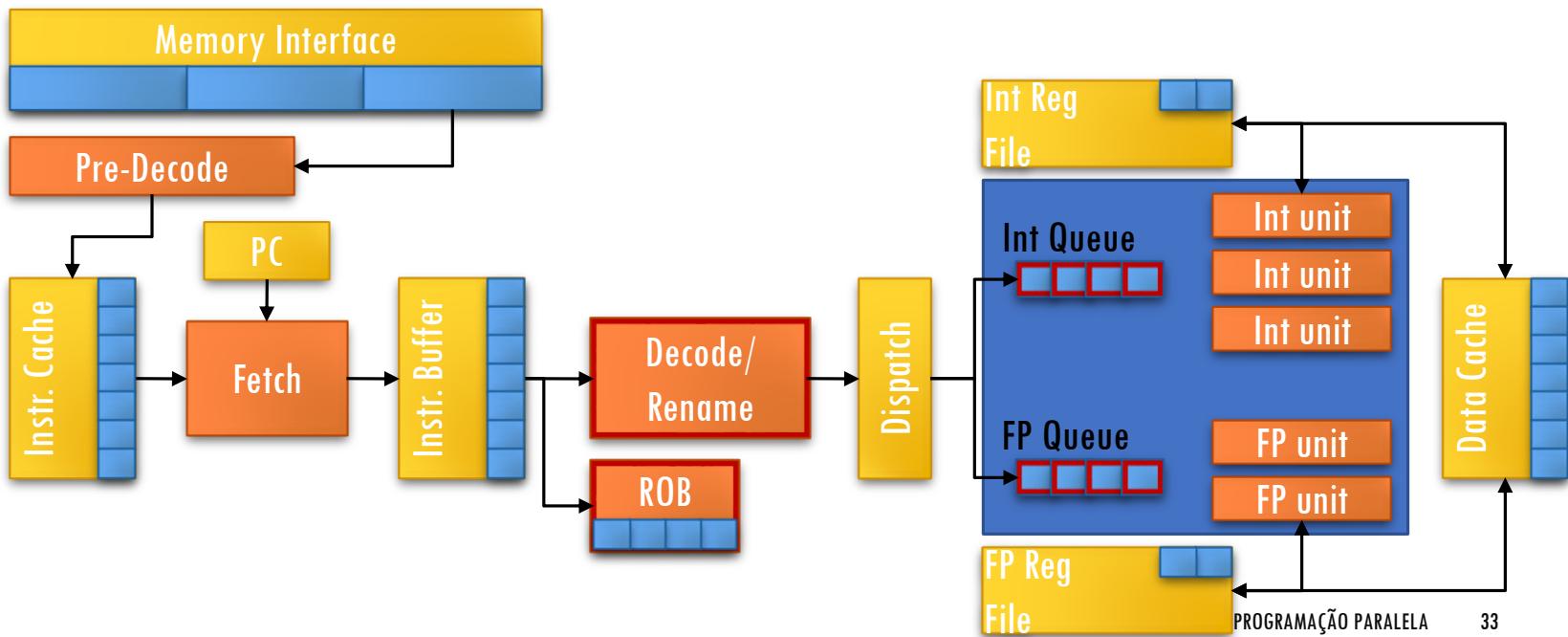


SINGLE CORE - PIPELINE SUPERSCALAR O³ (OUT-OF-ORDER)

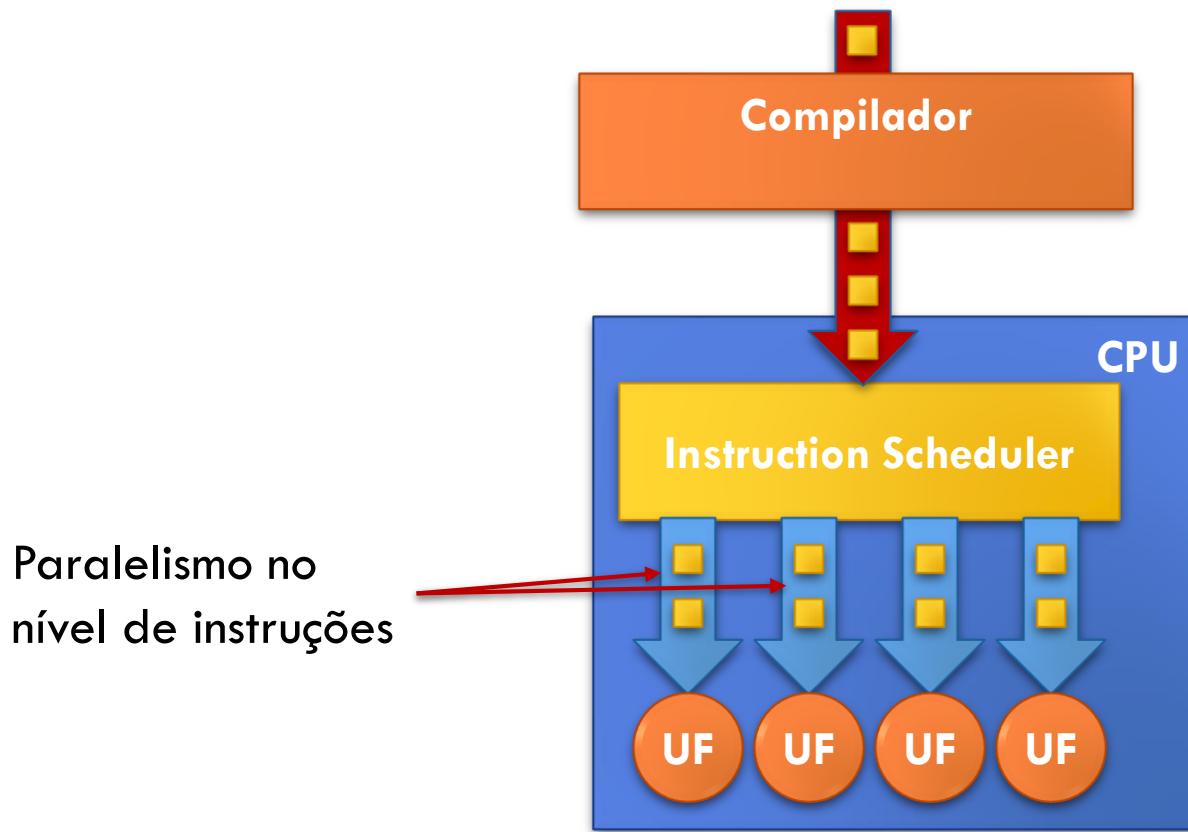
Início do Paralelismo (**ILP – Instruction Level Parallelism**)

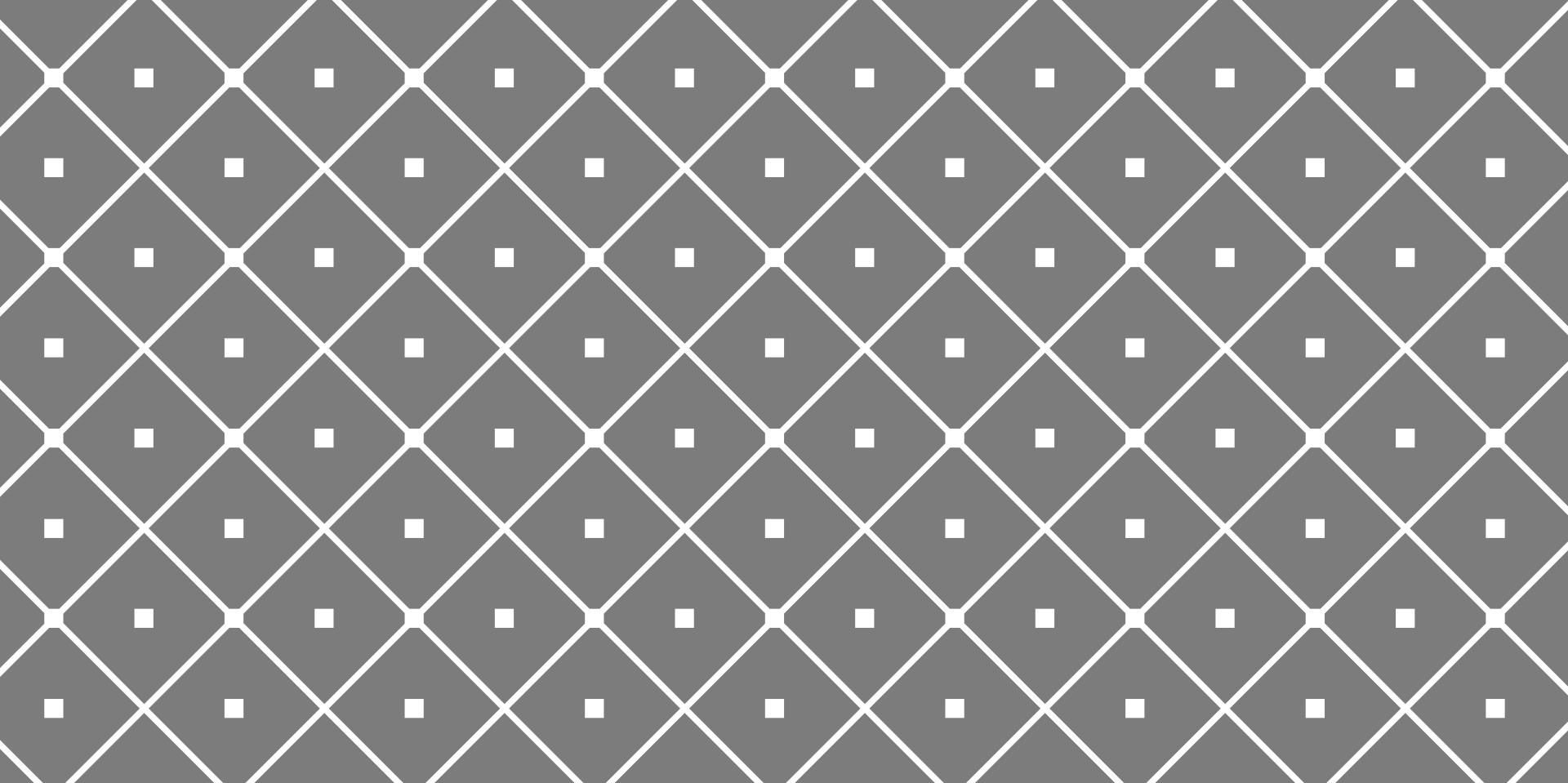
Busca, decodificação e execução de mais de uma instrução por ciclo.

Paralelismo agressivo com despacho e execução fora-de-ordem (OOO).



SINGLE CORE - PIPELINE SUPERSCALAR O³ (OUT-OF-ORDER)

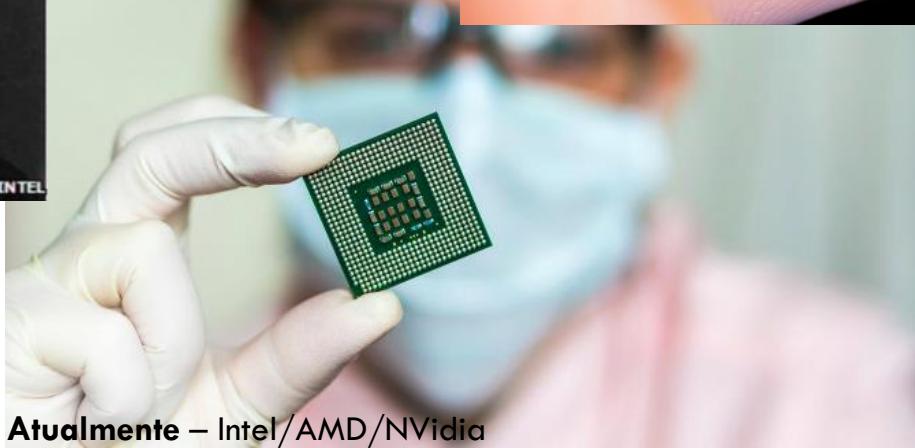




REVOLUÇÃO DOS TRANSISTORES

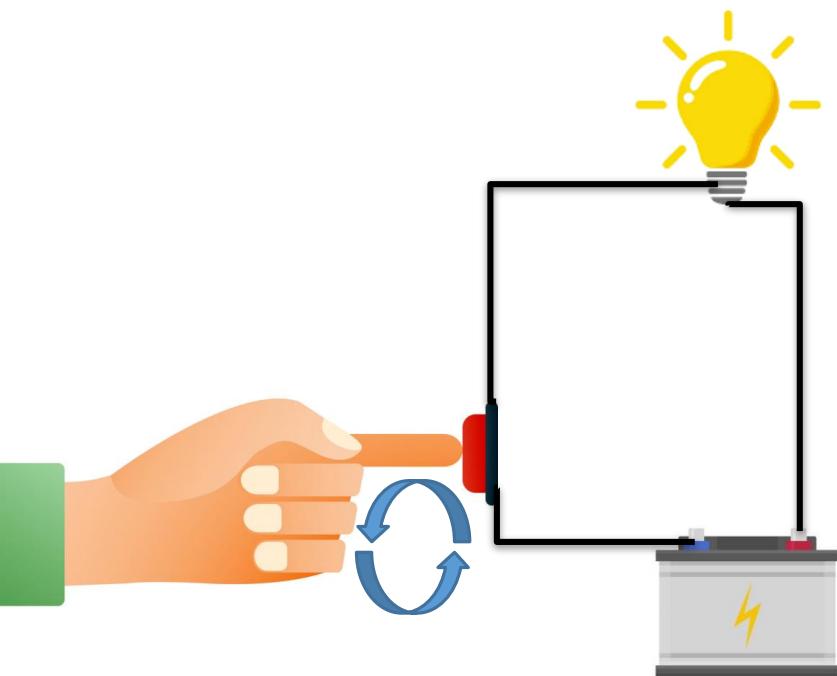
AUMENTO NO NÚMERO DE TRANSISTORES

1971 – Intel introduz o primeiro microprocessador com **2,250 transistores** (Intel 4004)

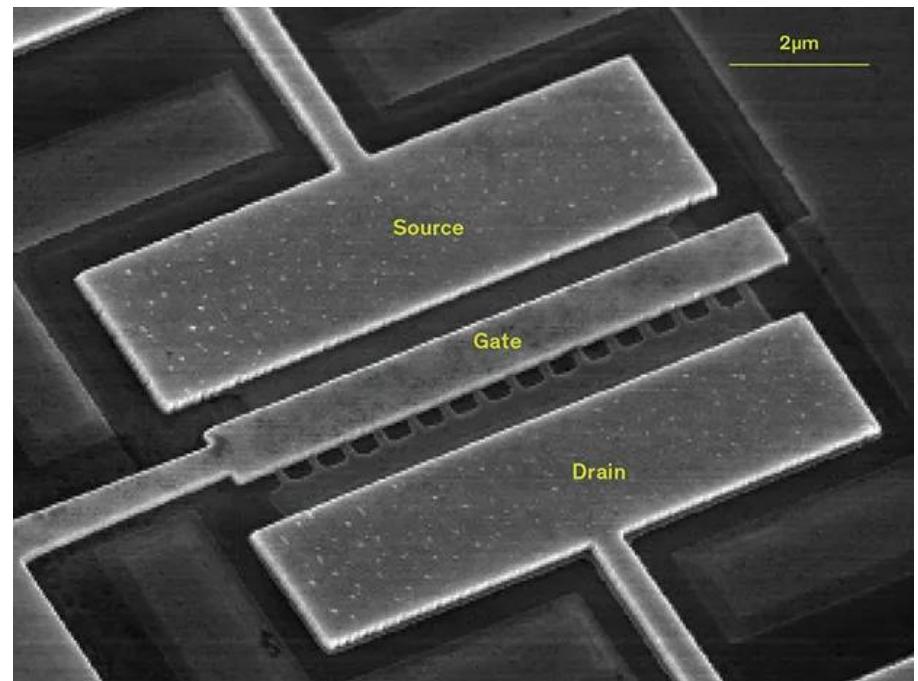
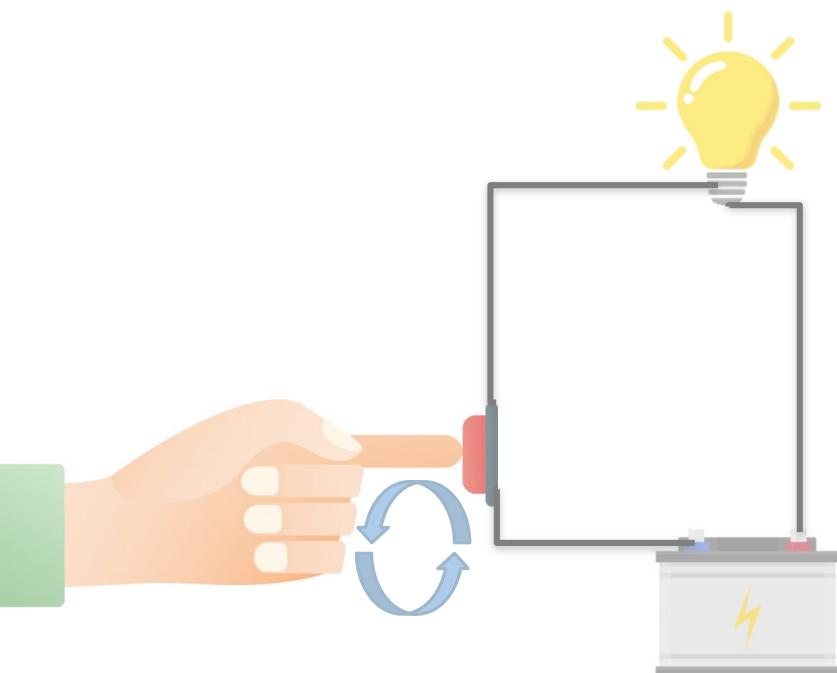


Atualmente – Intel/AMD/NVidia vendem processadores com **+8 bilhões de transistores**

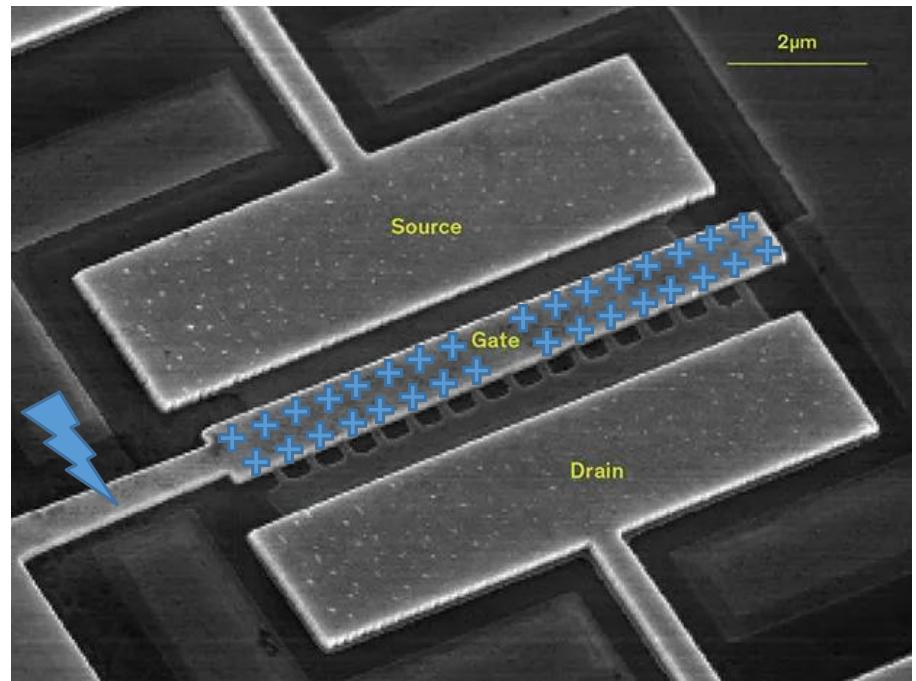
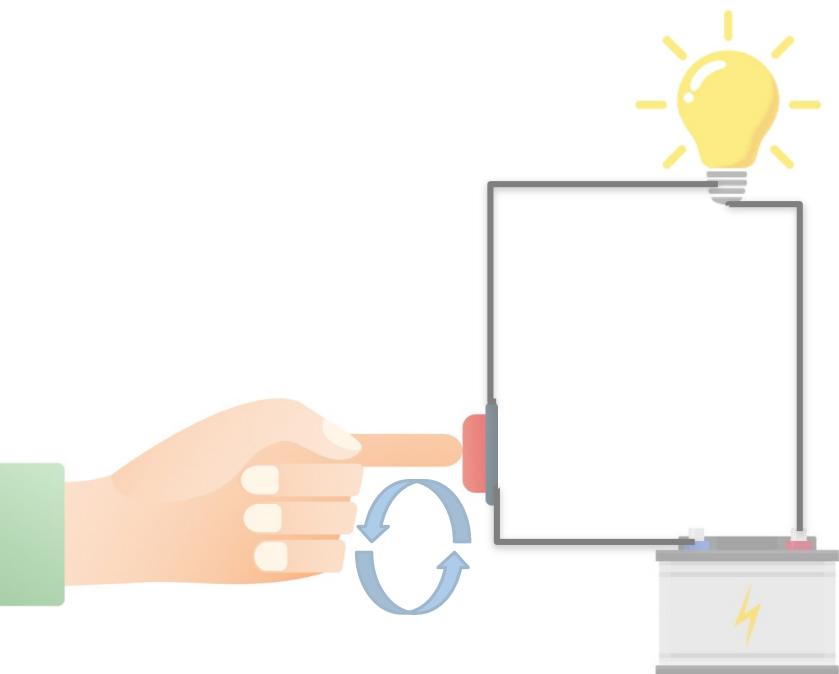
O QUE É UM TRANSISTOR?



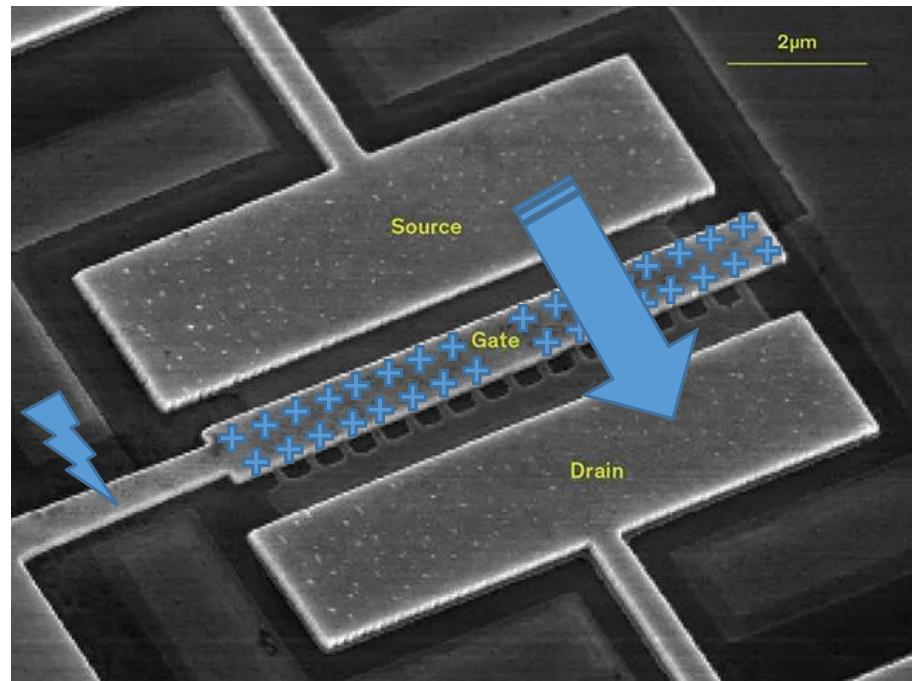
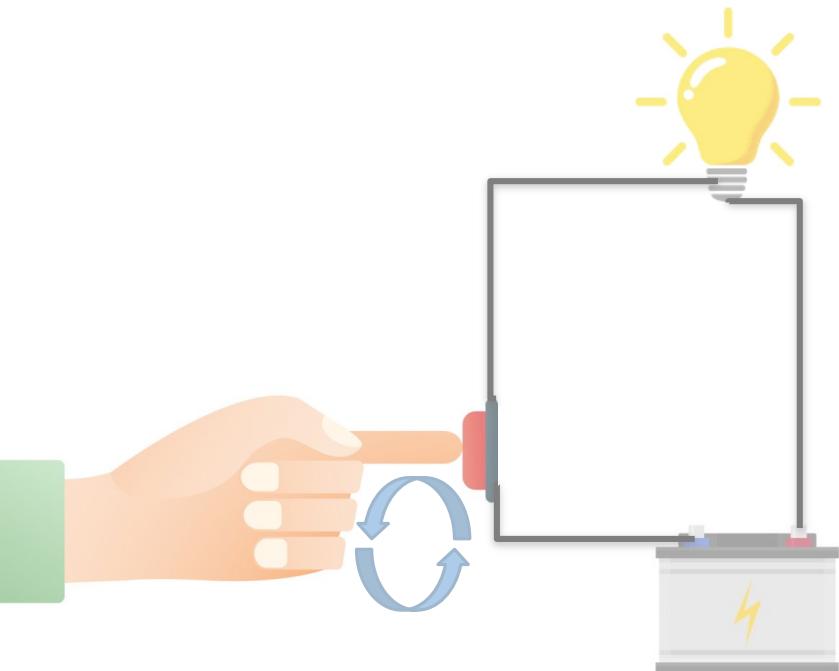
O QUE É UM TRANSISTOR?



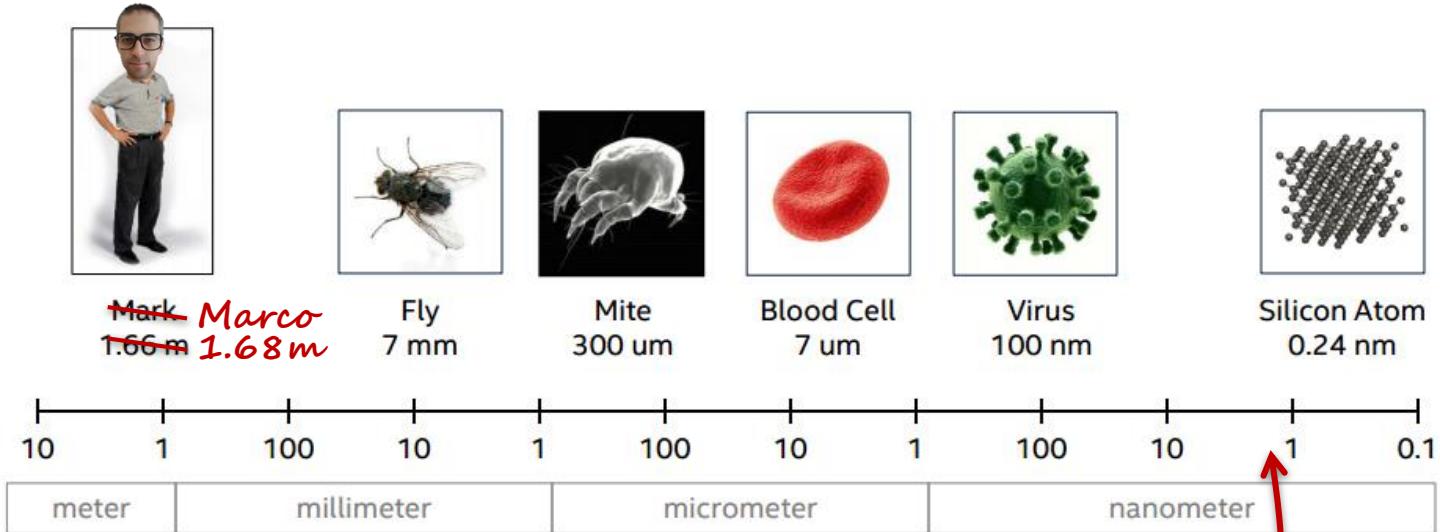
O QUE É UM TRANSISTOR?



O QUE É UM TRANSISTOR?

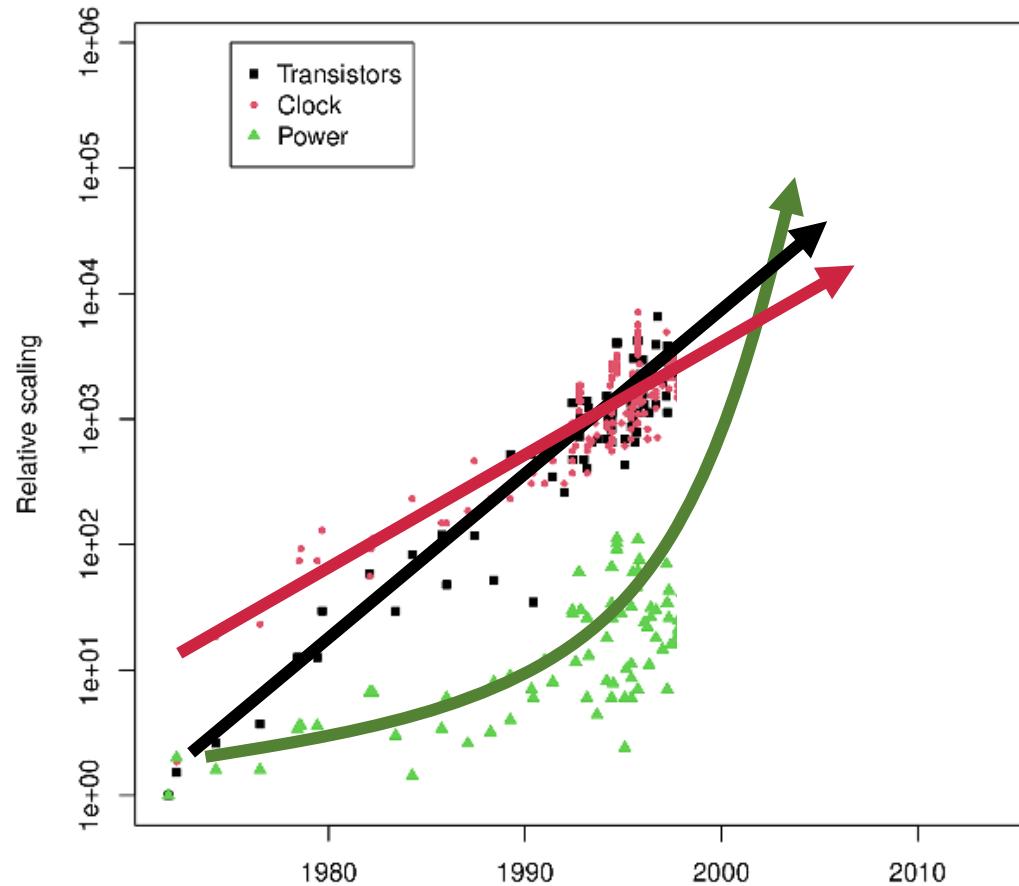


QUÃO PEQUENO É 14NM? [IDF'14]

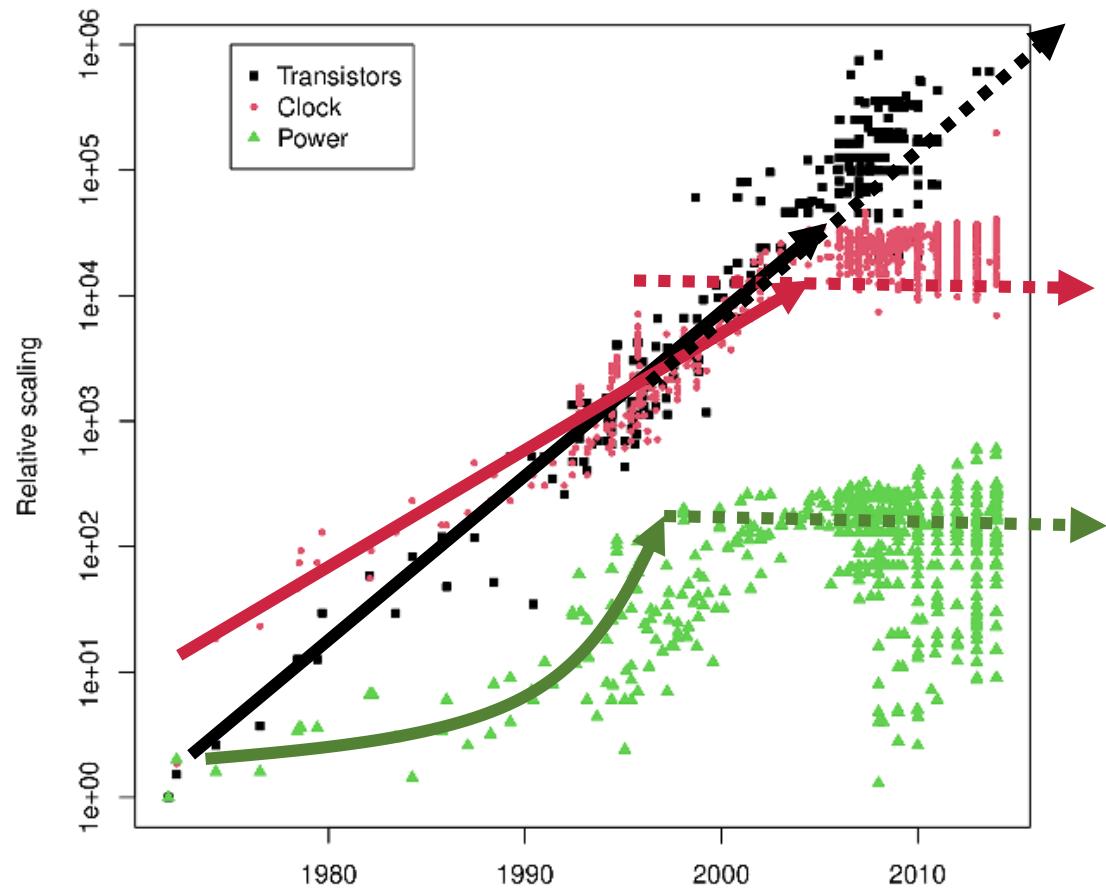


Transistor
gate width

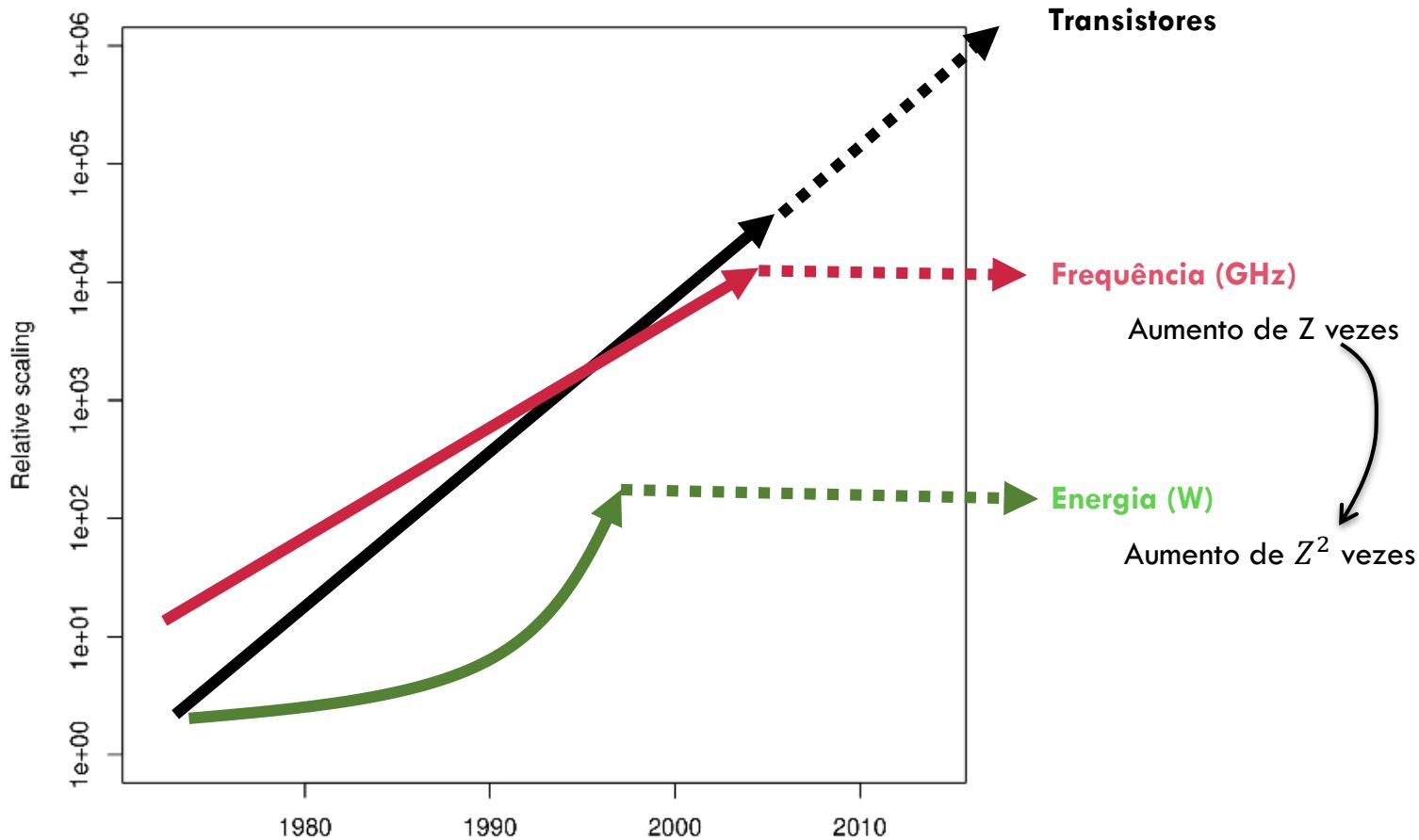
TENDÊNCIA DE TRANSITORES E FREQUÊNCIA



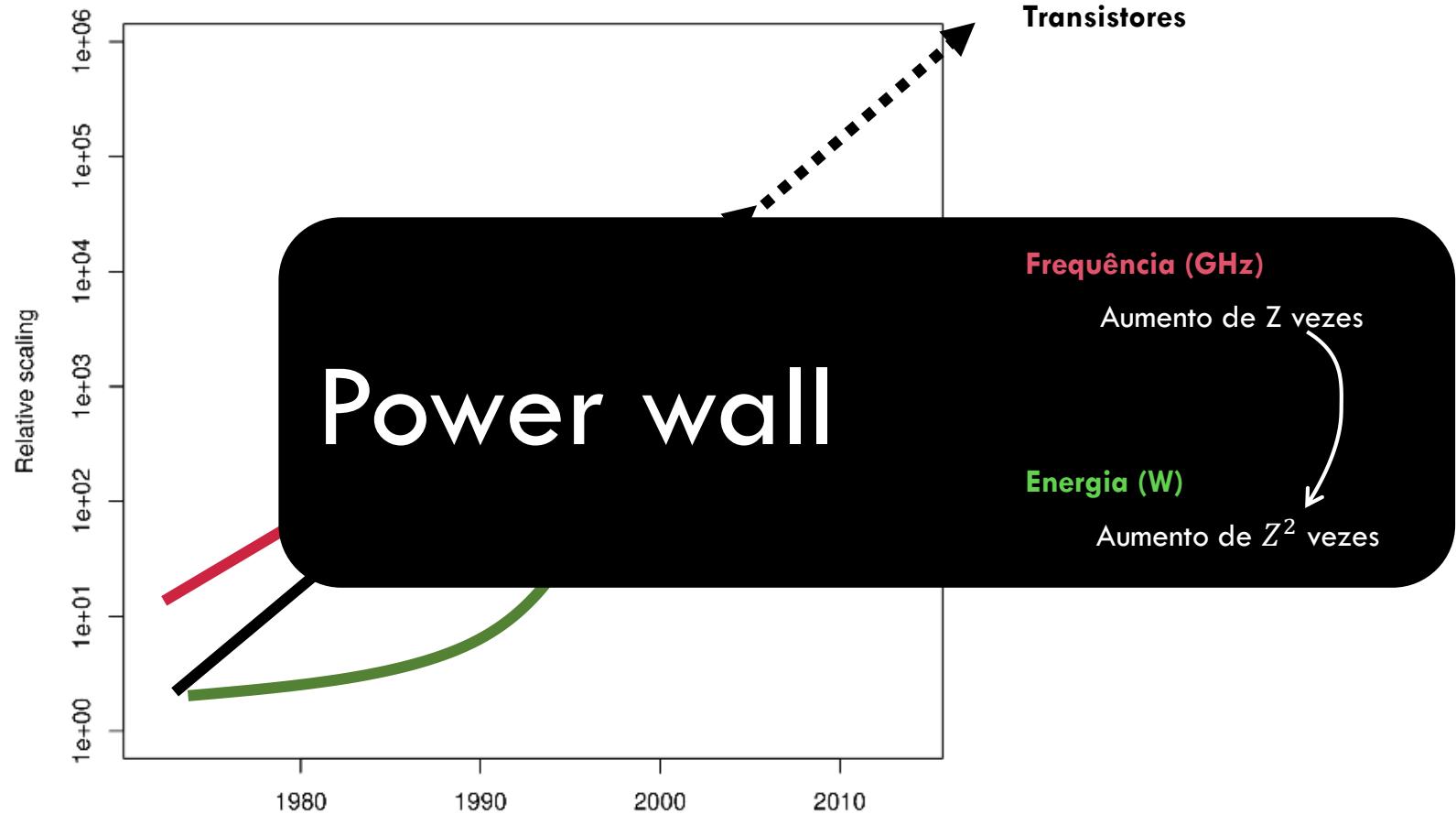
TENDÊNCIA DE TRANSITORES E FREQUÊNCIA



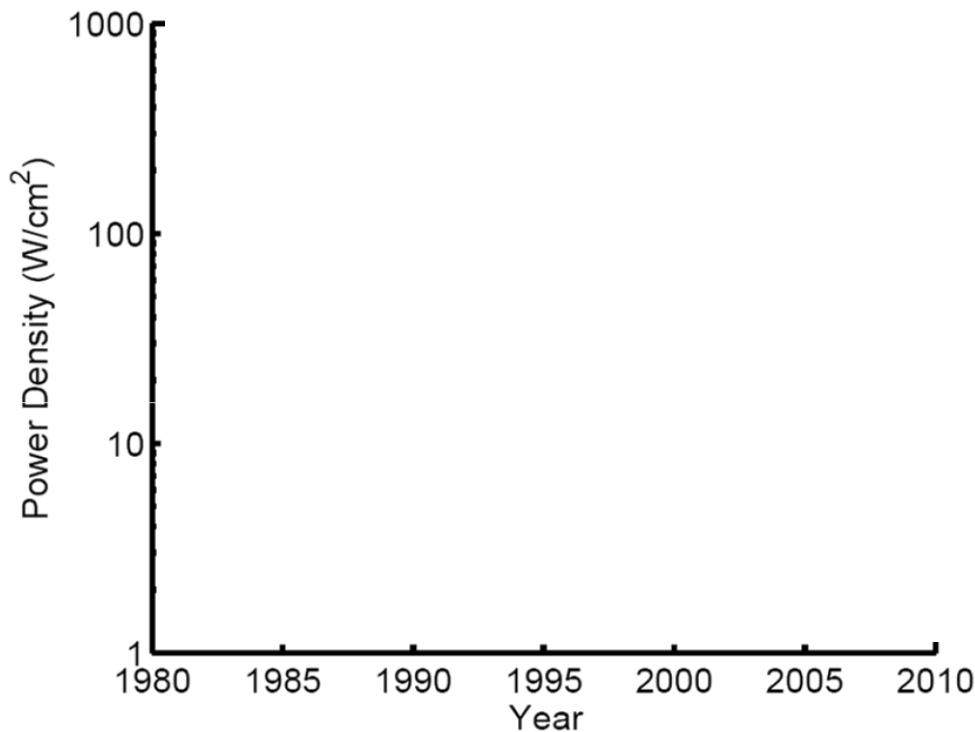
TENDÊNCIA DE TRANSITORES E FREQUÊNCIA



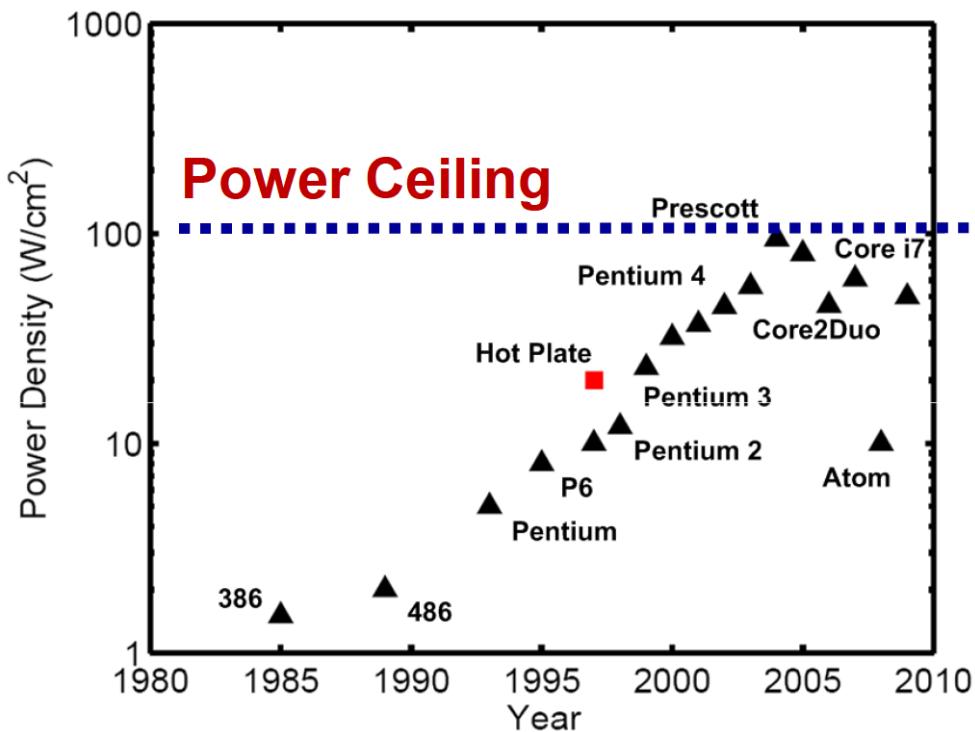
TENDÊNCIA DE TRANSITORES E FREQUÊNCIA



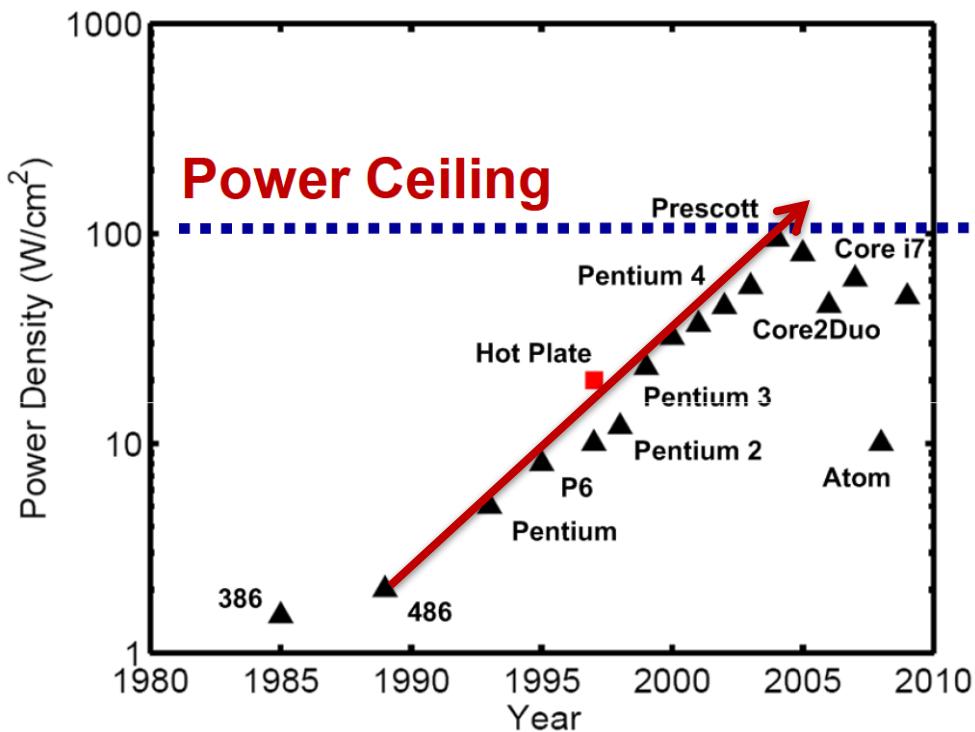
CONSEQUÊNCIAS DO AUMENTO DA FREQUÊNCIA DE OPERAÇÃO



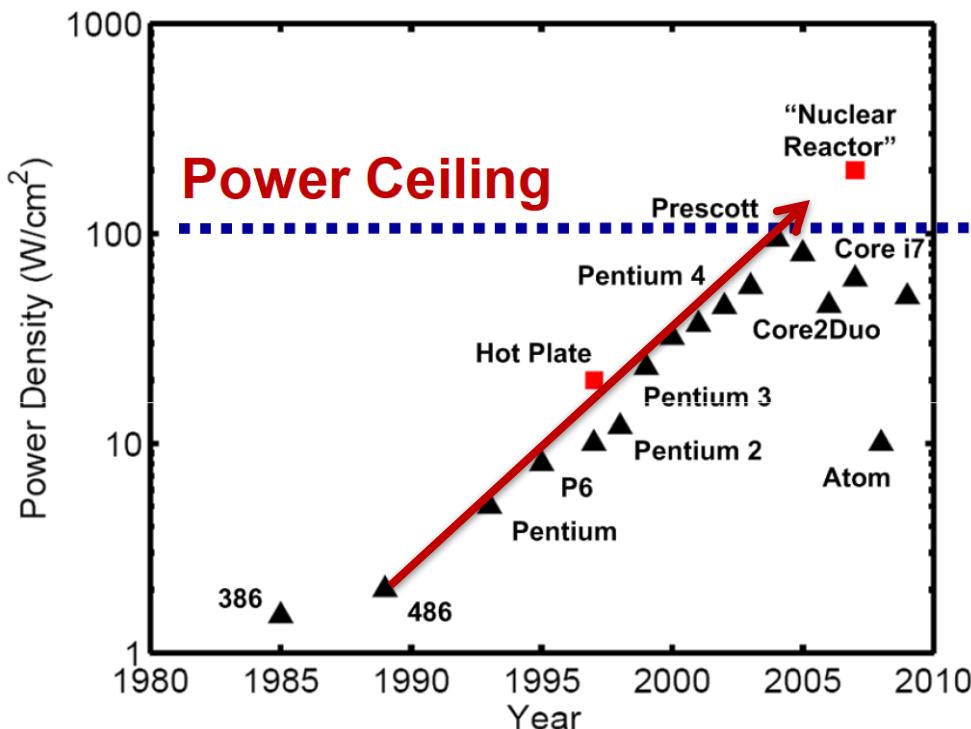
CONSEQUÊNCIAS DO AUMENTO DA FREQUÊNCIA DE OPERAÇÃO

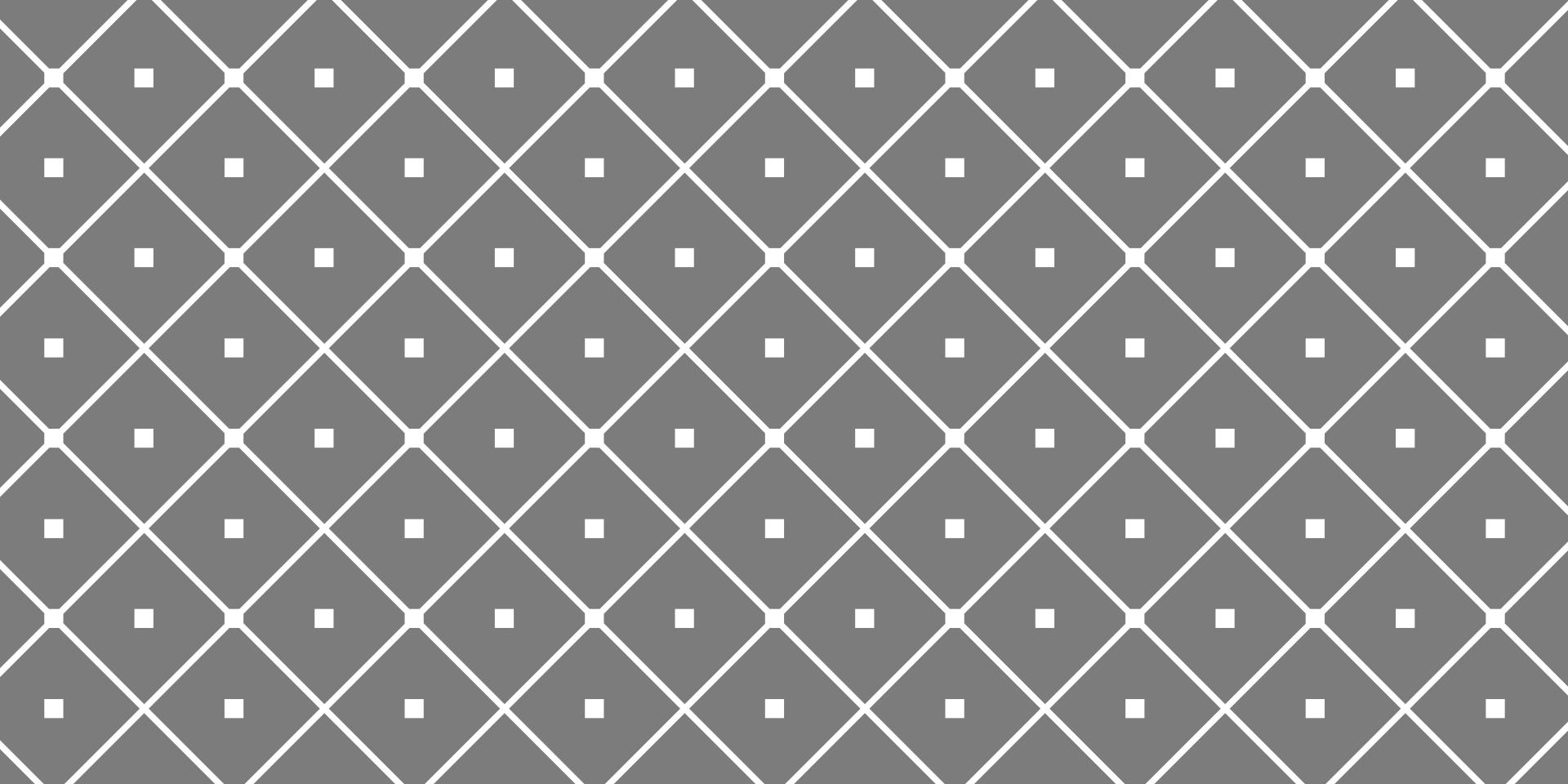


CONSEQUÊNCIAS DO AUMENTO DA FREQUÊNCIA DE OPERAÇÃO



CONSEQUÊNCIAS DO AUMENTO DA FREQUÊNCIA DE OPERAÇÃO





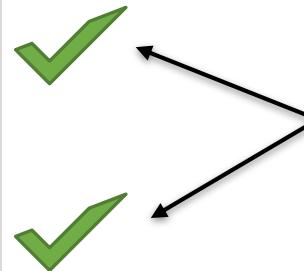
MUDANÇA DE CONTEXTO

CONTEXTO DA MUDANÇA SINGLE-CORE → MULTI-CORE

Instruction Parallelism wall

- Programas tem um limite para extração de ILP;

```
Add r3=r1+r2
Sub r5=r3-r4
And r8=r6&r7
Mul r9=r3*r3
```



Paralelismo em nível de instrução (ILP)

*[Agarwal, 2000]

**[Borkar, 1999]

CONTEXTO DA MUDANÇA SINGLE-CORE → MULTI-CORE

Instruction Parallelism wall

- Programas tem um limite para extração de ILP;

Clock wall

- Difícil aumentar a profundidade do Pipeline; *
- Wire-Delay – O clock muda de sinal antes de se propagar/estabilizar;

Os fios têm um atraso de propagação aproximado de 1 ns para cada 6 polegadas (15 cm) de comprimento

Balch, Mark (2003). Complete Digital Design A Comprehensive Guide To Digital Electronics And Computer System Architecture. McGraw-Hill Professional. p. 430. ISBN 978-0-07-140927-8.

*[Agarwal, 2000]

**[Borkar, 1999]

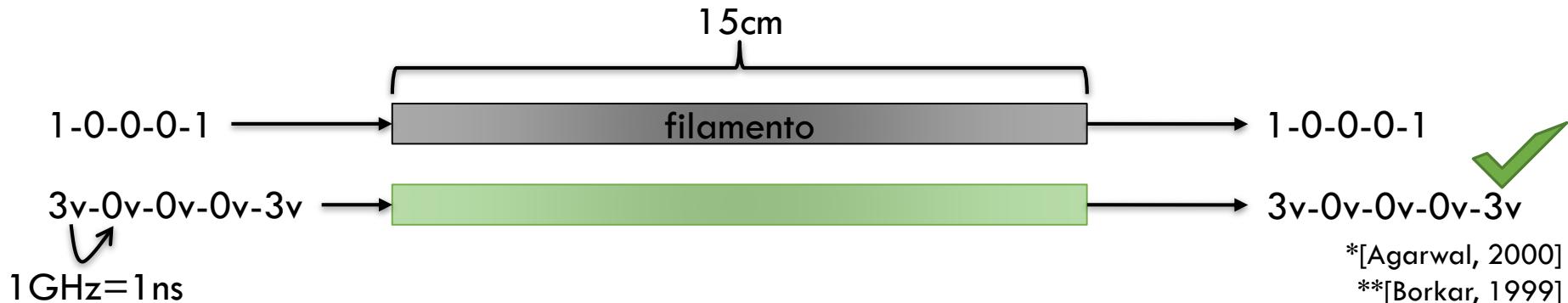
CONTEXTO DA MUDANÇA SINGLE-CORE → MULTI-CORE

Instruction Parallelism wall

- Programas tem um limite para extração de ILP;

Clock wall

- Difícil aumentar a profundidade do Pipeline; *
- Wire-Delay – O clock muda de sinal antes de se propagar/estabilizar;



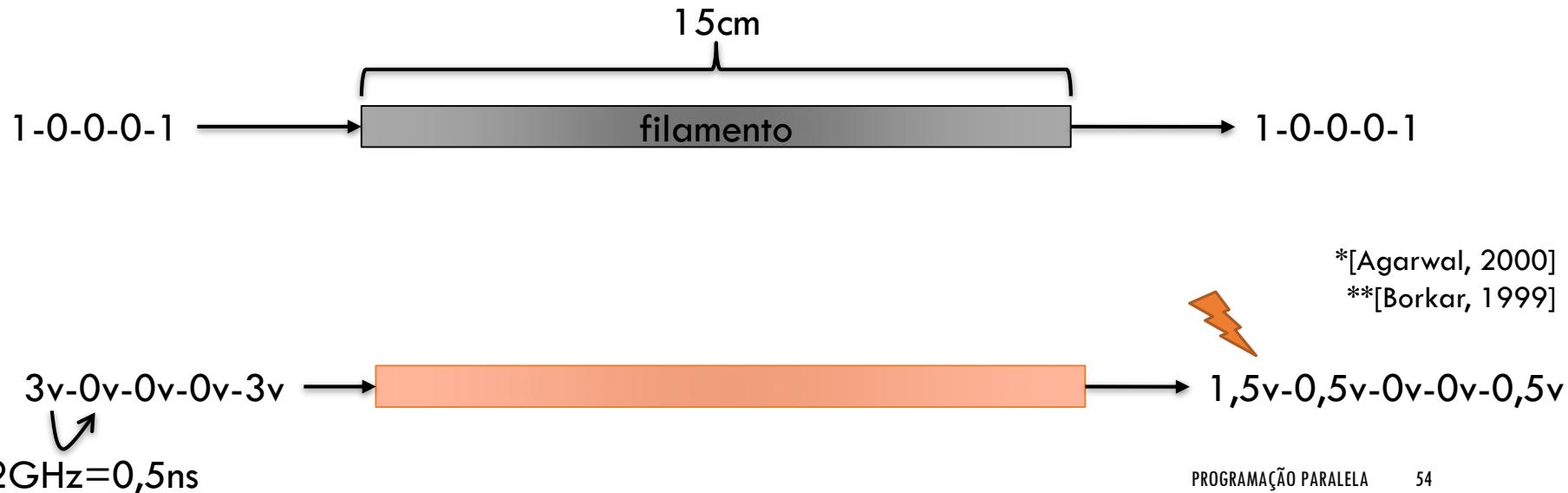
CONTEXTO DA MUDANÇA SINGLE-CORE → MULTI-CORE

Instruction Parallelism wall

- Programas tem um limite para extração de ILP;

Clock wall

- Difícil aumentar a profundidade do Pipeline; *
- Wire-Delay – O clock muda de sinal antes de se propagar/estabilizar;



CONTEXTO DA MUDANÇA SINGLE-CORE → MULTI-CORE

Instruction Parallelism **wall**

- Programas tem um limite para extração de ILP;

Clock **wall**

- Difícil aumentar a profundidade do Pipeline; *
- Wire-Delay – O clock muda de sinal antes de se propagar/estabilizar;

Power **wall**

- Leakage aumenta 7.5x a cada nova geração;**
- Bloco de Controle muito complexo;

$$P = CV^2F$$

*[Agarwal, 2000]

**[Borkar, 1999]

CONSUMO DE ENERGIA EM SEMICONDUTORES

$$P = CV^2 F$$

Potência:
Taxa de consumo de energia

Voltagem: Influenciada pela capacidade e frequência

Capacitância:
Capacidade de armazenar energia

Frequência de operação
(chaveamentos por segundo)

The diagram illustrates the formula for power consumption in semiconductors, $P = CV^2 F$. Four blue arrows point from explanatory text to the corresponding variables in the equation:

- An arrow points from "Potência: Taxa de consumo de energia" to the variable P .
- An arrow points from "Voltagem: Influenciada pela capacidade e frequência" to the variable V^2 .
- An arrow points from "Capacitância: Capacidade de armazenar energia" to the variable C .
- An arrow points from "Frequência de operação (chaveamentos por segundo)" to the variable F .

CONSUMO DE ENERGIA EM SEMICONDUTORES

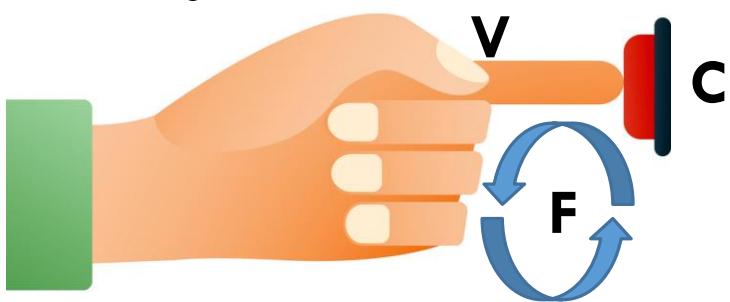
$$P = CV^2 F$$

Potência:
Taxa de consumo de energia

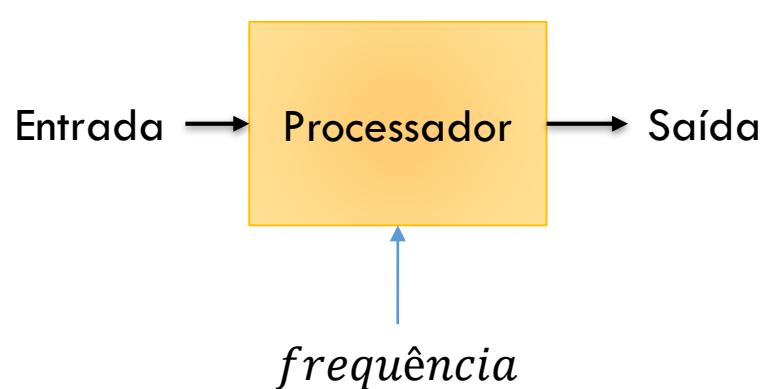
Voltagem: Influenciada pela capacidade e frequência

Capacitância: Capacidade de armazenar energia

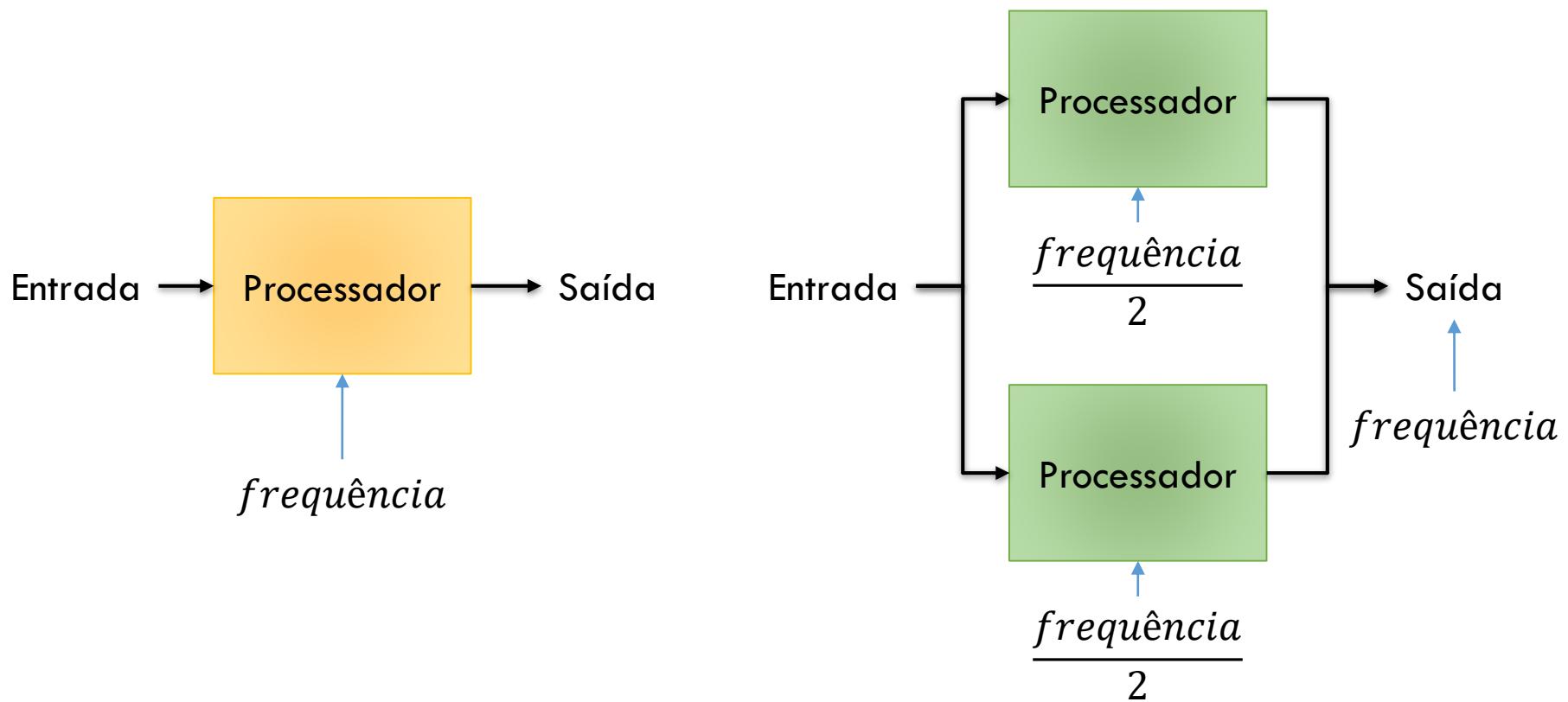
Frequência de operação (chaveamentos por segundo)



COMO O PARALELISMO NOS AJUDA?



COMO O PARALELISMO NOS AJUDA?



COMO O PARALELISMO NOS AJUDA?

Single Core

Capacitância = C

Voltagem = V

Frequência = F

$$P = CV^2F$$

COMO O PARALELISMO NOS AJUDA?

Single Core

$$Capacitância = C$$

$$Voltagem = V$$

$$Frequência = F$$

$$P = CV^2F$$

Dual Core

$$Capacitância = 2,2 \cdot C$$

$$Voltagem = 0,6 \cdot V$$

$$Frequência = 0,5 \cdot F$$

Aumenta pois precisamos de mais fios

Escala junto com a frequência, porém temos que considerar leakage, etc.

COMO O PARALELISMO NOS AJUDA?

Single Core

$$Capacitância = C$$

$$Voltagem = V$$

$$Frequência = F$$

$$P = CV^2F$$

Dual Core

$$Capacitância = 2,2 \cdot C$$

$$Voltagem = 0,6 \cdot V$$

$$Frequência = 0,5 \cdot F$$

$$P = 0,396CV^2F$$

-60%
consumo
energia

Aumenta pois
precisamos de
mais fios

Escala junto com a frequência,
porém temos que considerar
leakage, etc.

CONTEXTO DA MUDANÇA SINGLE-CORE → MULTI-CORE

Instruction Parallelism (ILP) **wall**

- Programas tem um limite para extração de ILP;

Clock **wall**

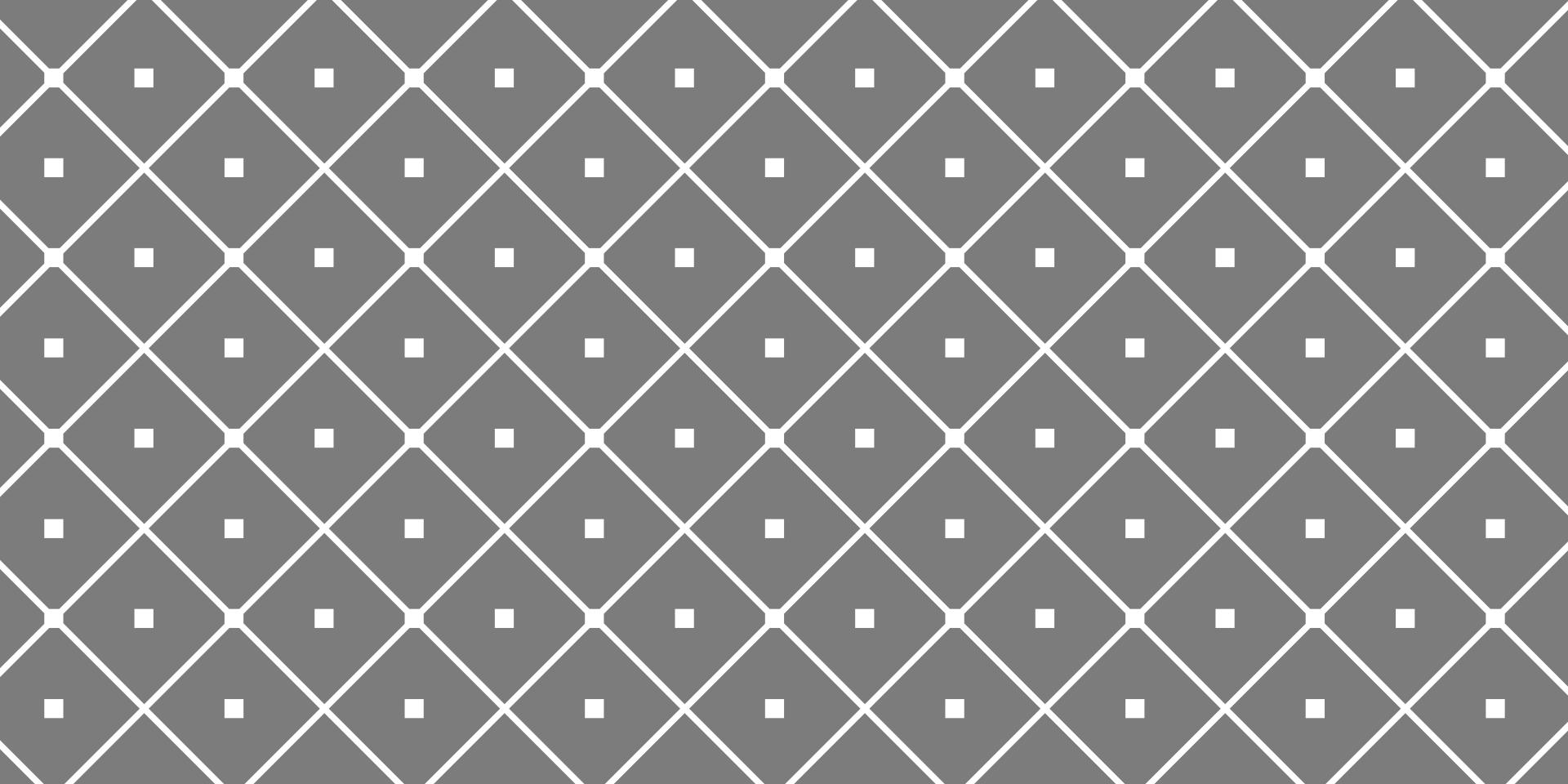
- Difícil aumentar a profundidade do Pipeline; *
- Wire-Delay – O clock muda de sinal antes de se propagar/estabilizar;

Power **wall**

- Leakage aumenta 7.5x a cada nova geração;**
- Bloco de Controle muito complexo;

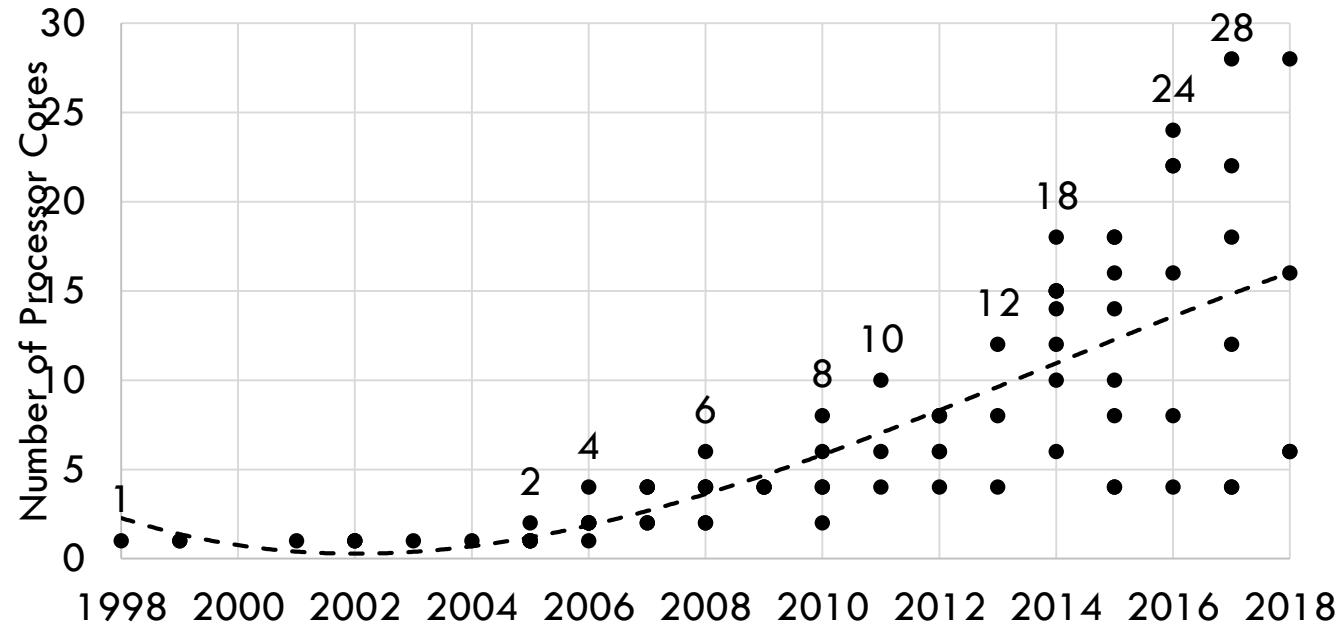
*[Agarwal, 2000]

**[Borkar, 1999]



EFEITO DESSE LIMITE

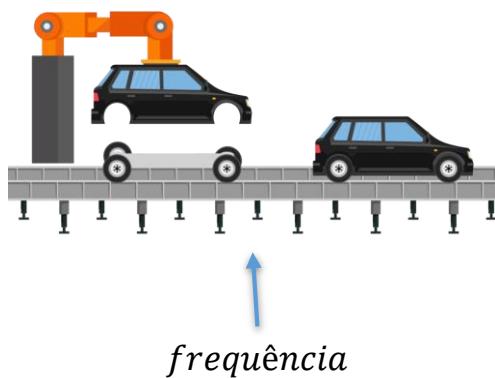
EVOLUÇÃO DO INTEL XEON - # NÚCLEOS



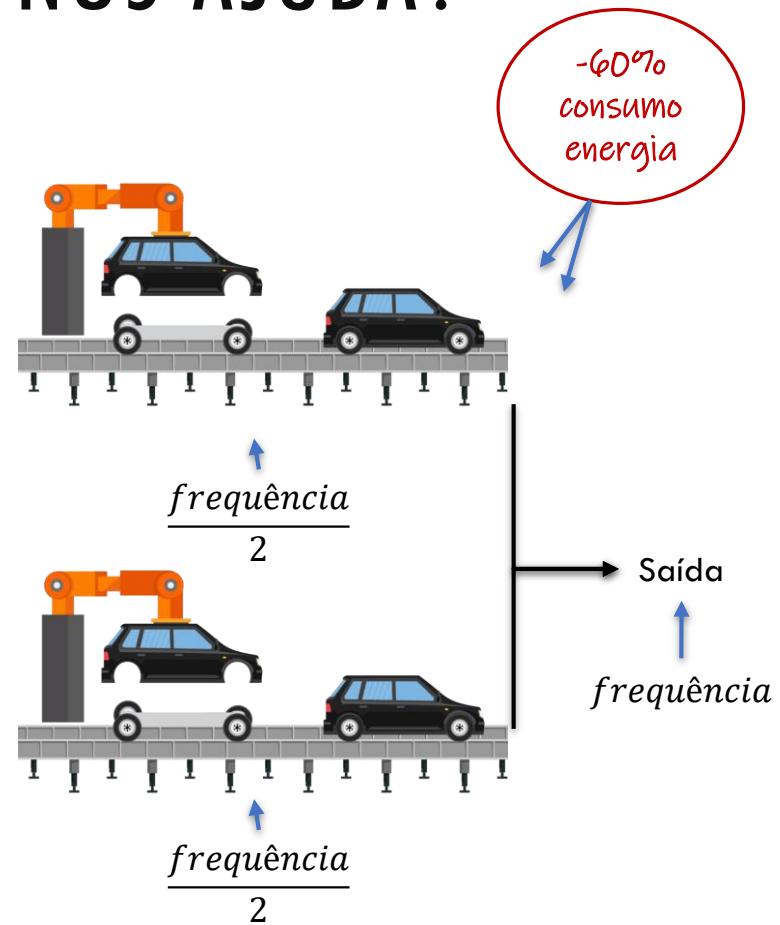
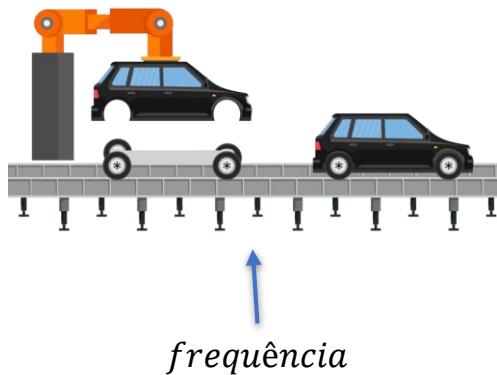
EVOLUÇÃO DO INTEL XEON - # NÚCLEOS



COMO O PARALELISMO NOS AJUDA?

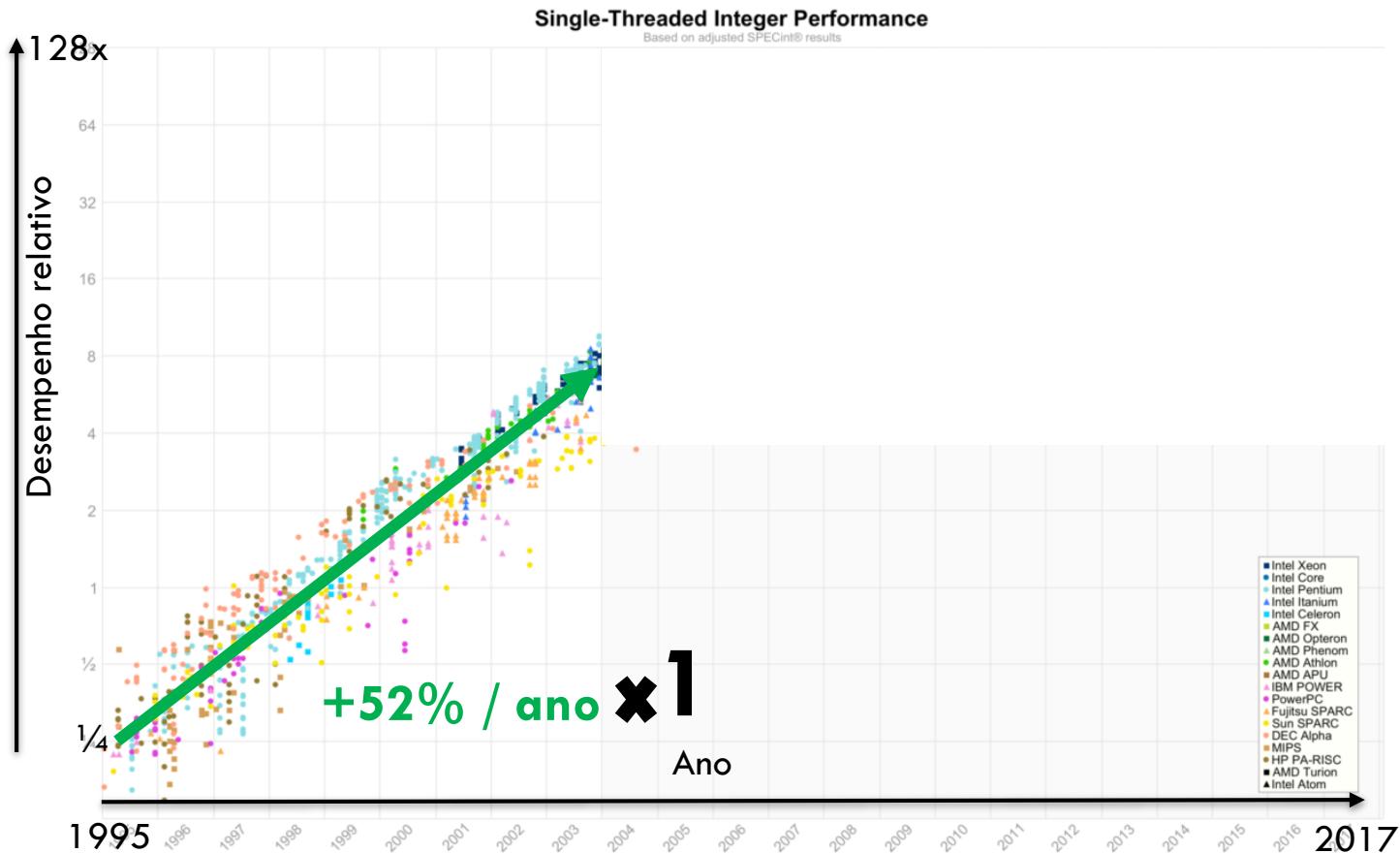


COMO O PARALELISMO NOS AJUDA?

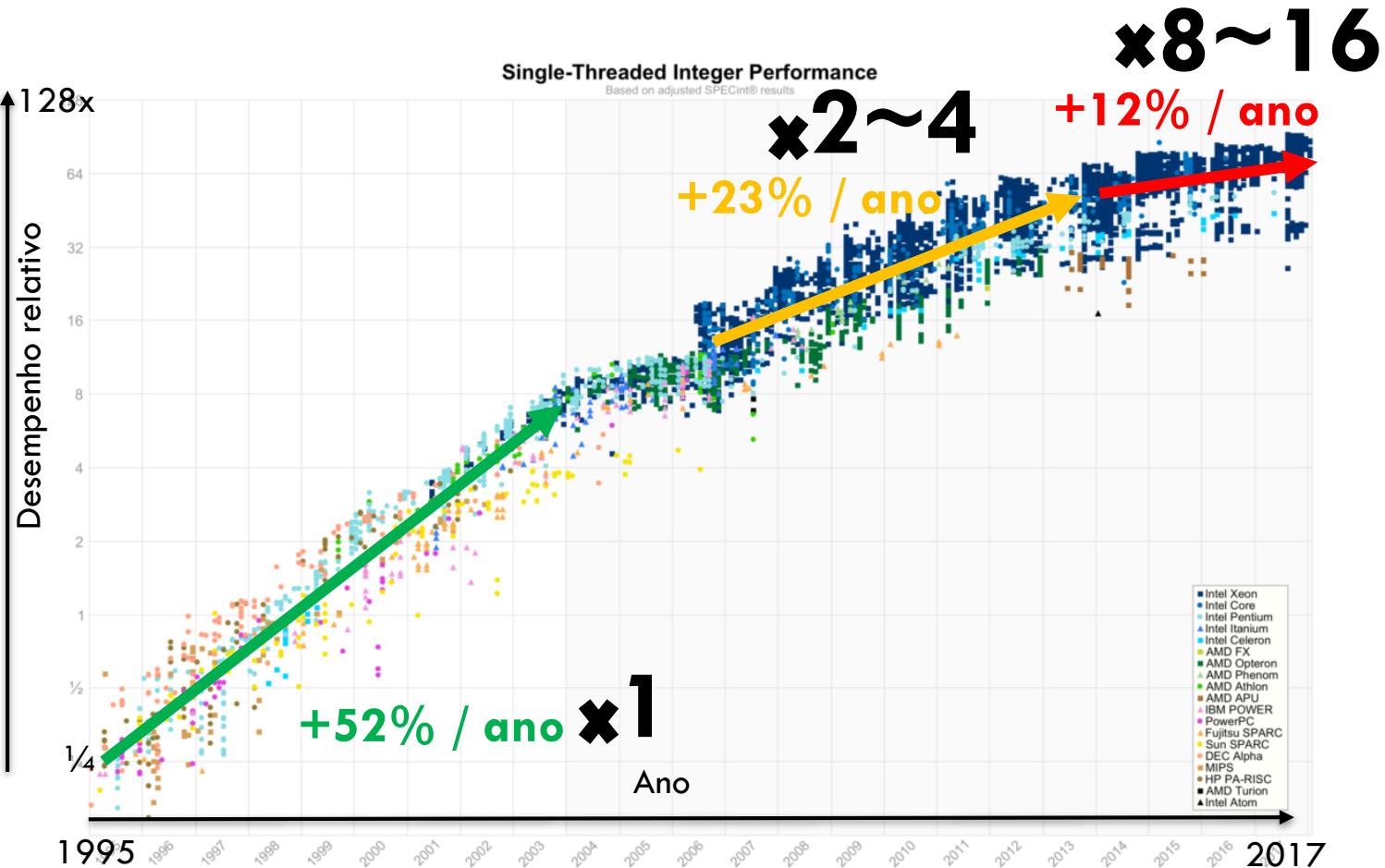


DESEMPENHO AO LONGO DOS ANOS

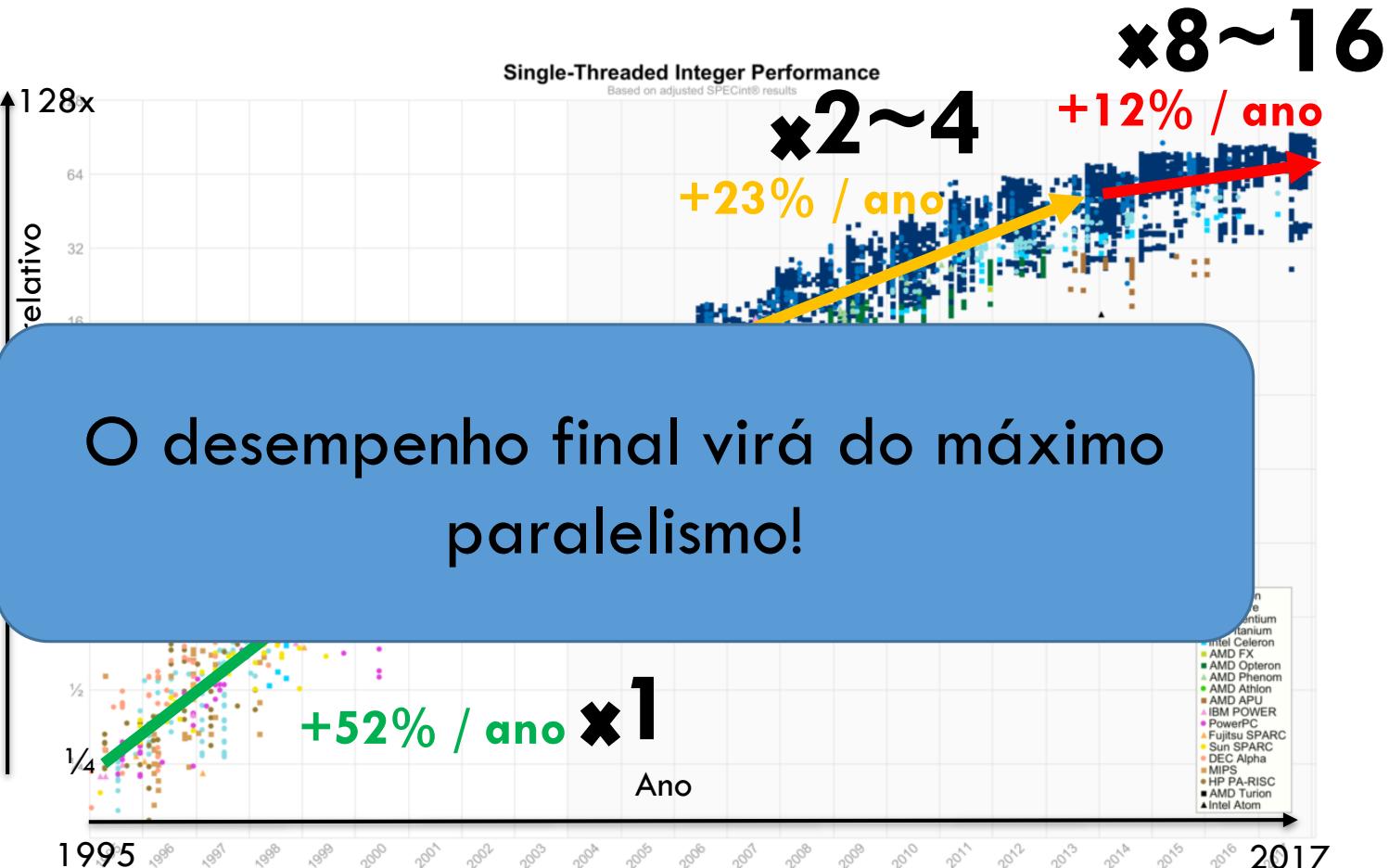
Fonte: <http://top500.org/2010/01/a-look-back-at-single-threaded-performance/>

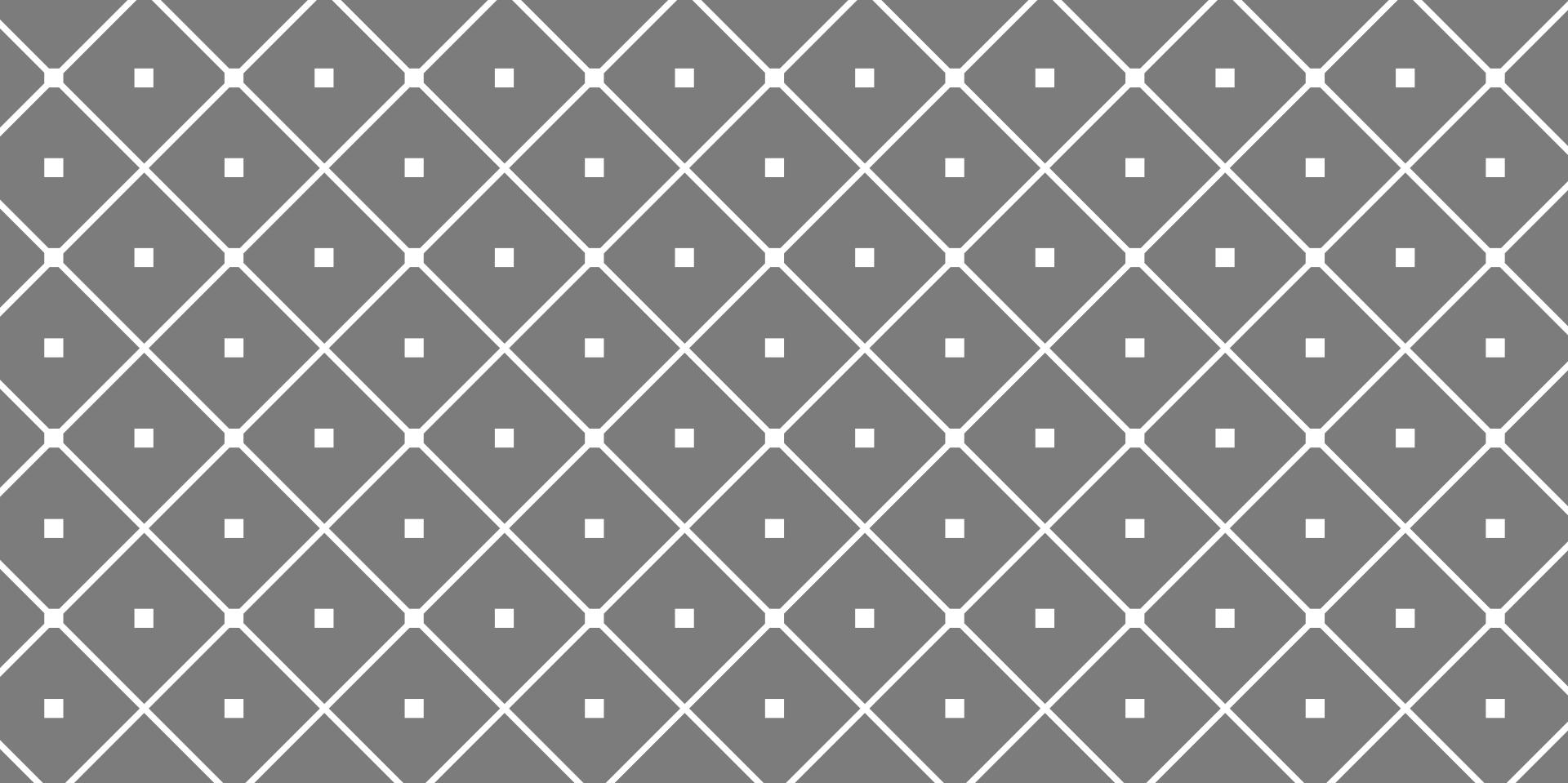


DESEMPENHO AO LONGO DOS ANOS



DESEMPENHO AO LONGO DOS ANOS

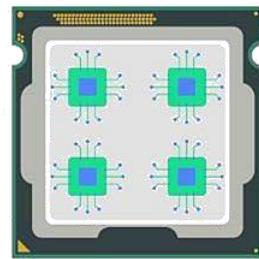




**BIG DATA =
BIG PROCESSING =
BIG PARALLELISM!**

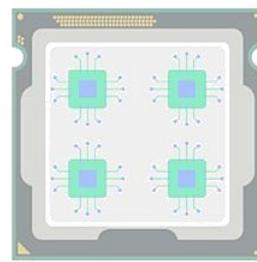
AUMENTANDO O PARALELISMO

Processadores multi-core

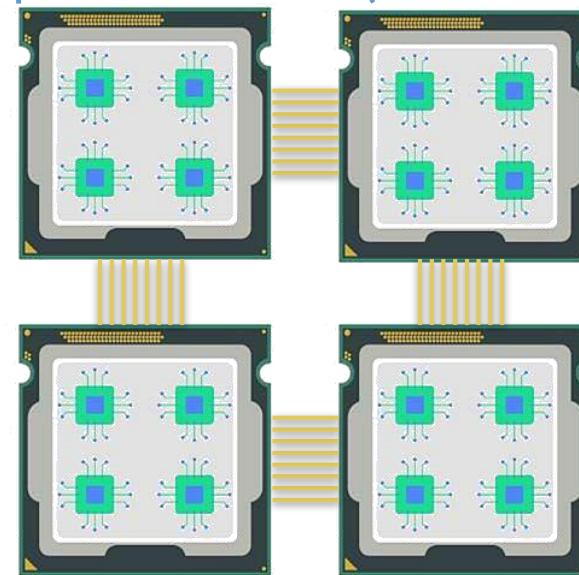


AUMENTANDO O PARALELISMO

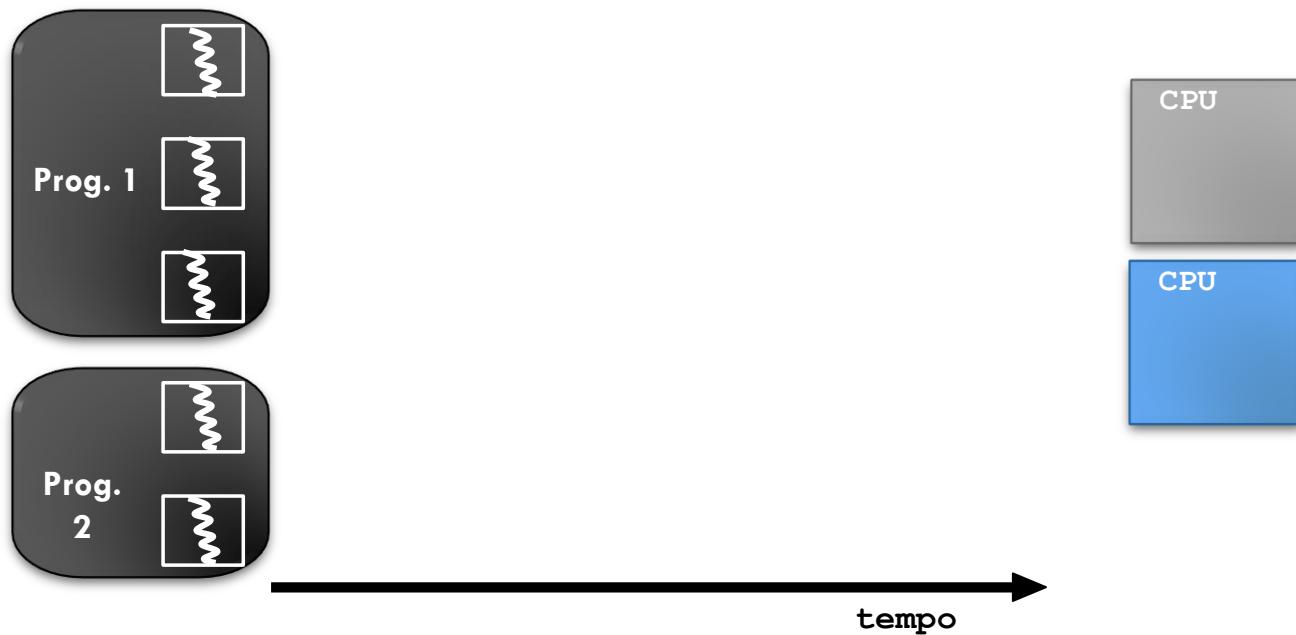
Processadores multi-core



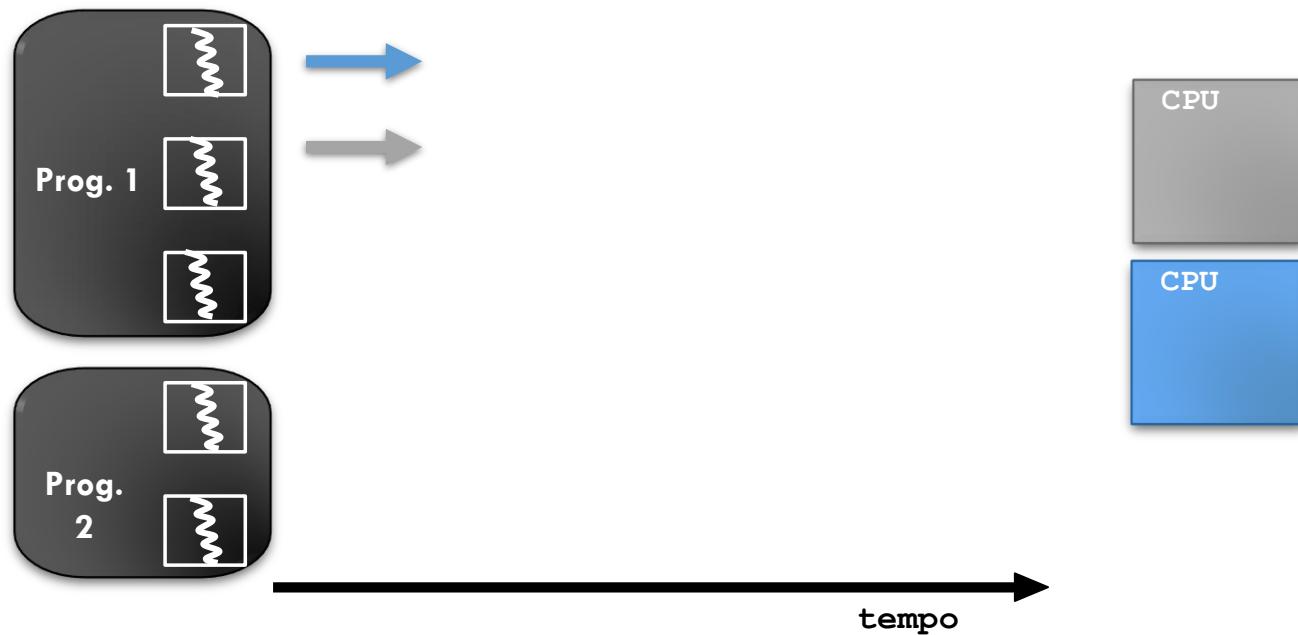
Máquinas com diversos processadores (i.e. NUMA)



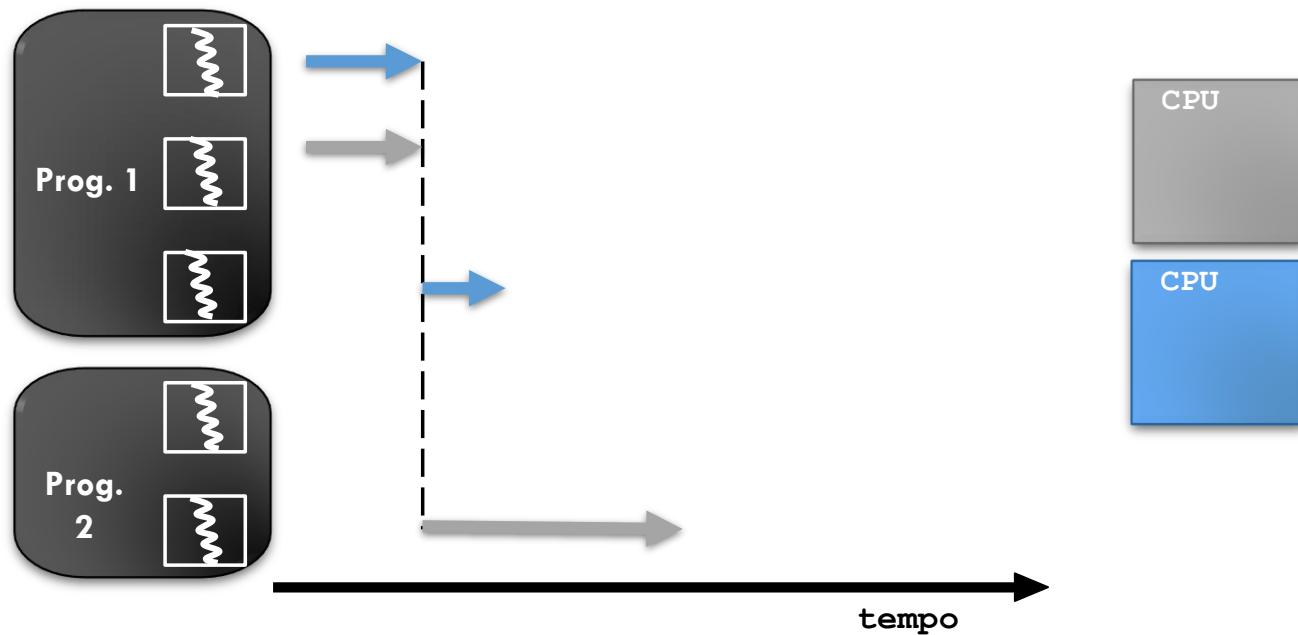
OPERAÇÃO BÁSICA DO SISTEMA OPERACIONAL



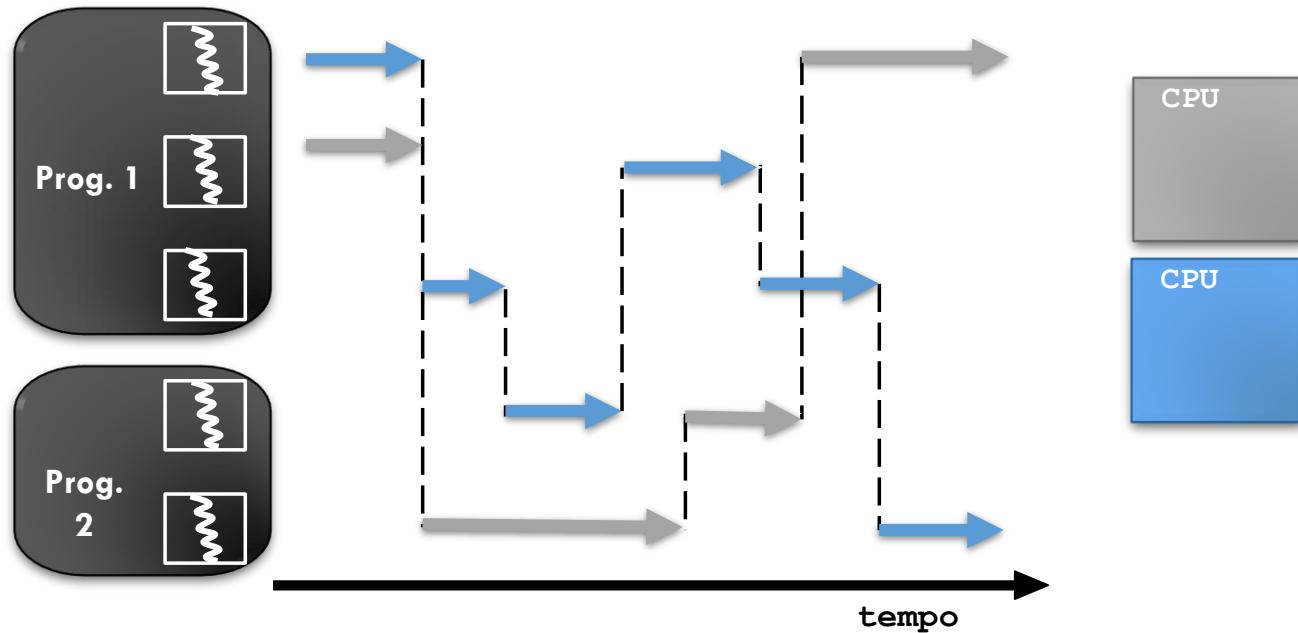
OPERAÇÃO BÁSICA DO SISTEMA OPERACIONAL



OPERAÇÃO BÁSICA DO SISTEMA OPERACIONAL



OPERAÇÃO BÁSICA DO SISTEMA OPERACIONAL



MÁQUINAS NUMA

Na arquitetura NUMA as partições dos Cores e as memórias são agrupadas em Nós

Uma máquina pode ter vários Nós NUMA, dependendo do modelo e quantidade de memória/processadores

A latência de acesso de um processador com a memória dentro de um Nós NUMA é muito baixa, pois o barramento torna eficiente o acesso dentro do mesmo Nós

MÁQUINAS NUMA

Na arquitetura NUMA as partições dos Cores e as memórias são agrupadas em Nós

Uma máquina pode ter vários Nós NUMA, dependendo do modelo e quantidade de memória/processadores

A latência de acesso de um processador com a memória dentro de um Nós NUMA é muito baixa, pois o barramento torna eficiente o acesso dentro do mesmo Nós

Entretanto o acesso do processador de um Nós X para a memória em um Nós Y é maior pois é necessário atravessar a interconexão que liga os diversos Nós

Pode parecer insignificante esta penalidade entretanto em acessos intensivos de memória a Nós remotos pode ter grande impacto no desempenho do sistema

A razão da diferença de latência entre o nodo local e um nodo remoto é chamado de **NUMA factor**.

MÁQUINAS NUMA MAPEAMENTO DE ENDEREÇOS

Diferentes tipos de mapeamento de endereço para os nodos NUMA podem ser adotados:

Entrelaçado			
0	1	2	3
4	5	6	7
8	9	10	11
...

Linear			
0	10	20	30
1	11	21	31
2	12	22	32
...

Bloco Entrelaçado				
0	5	10	15	
~	~	~	~	
4	9	14	19	
20				
...

MÁQUINAS NUMA

Exemplo de NUMA com capacidade total de 400 páginas de memória

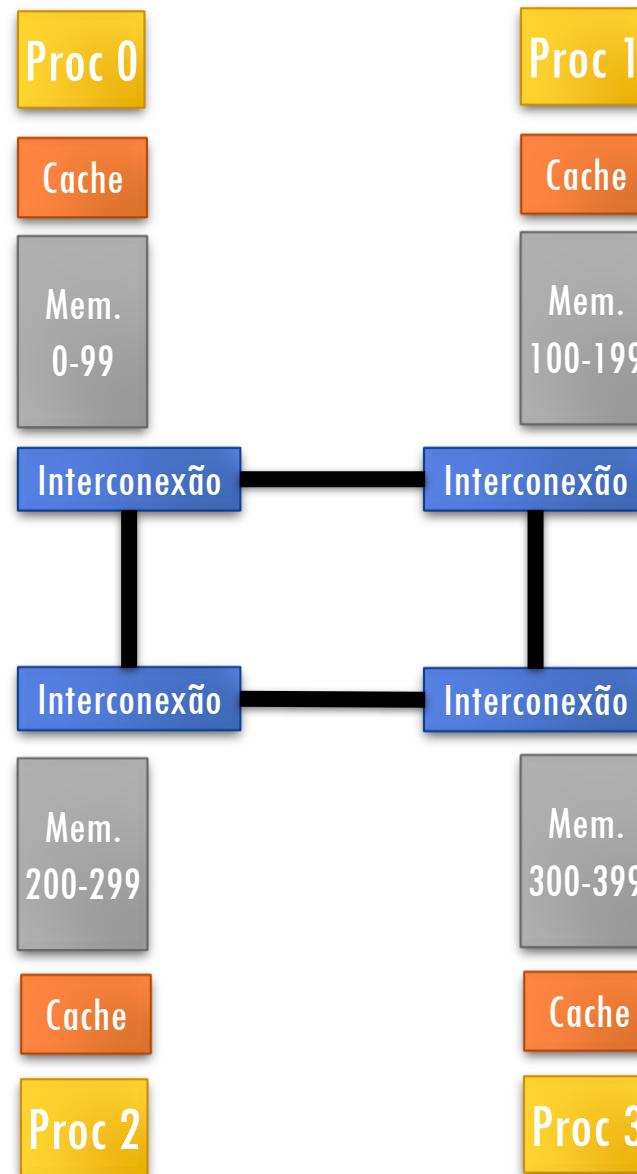
Cada nó contém um quadro de páginas contíguo

Considerando as latências:

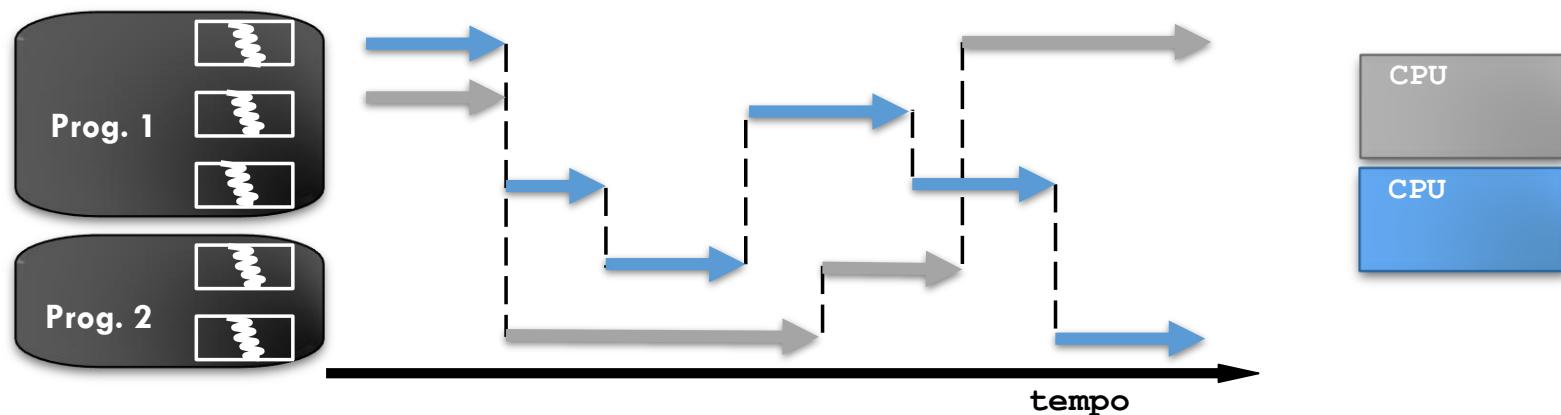
- Caches/Memória igual a **m ciclos**
- Interconexão/link igual a **n ciclos**

O custo de acesso do Proc 0 acessar:

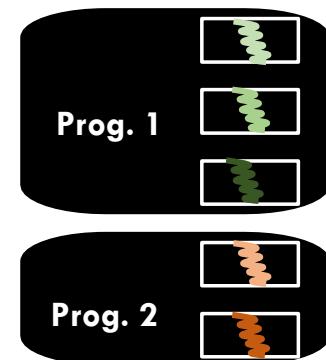
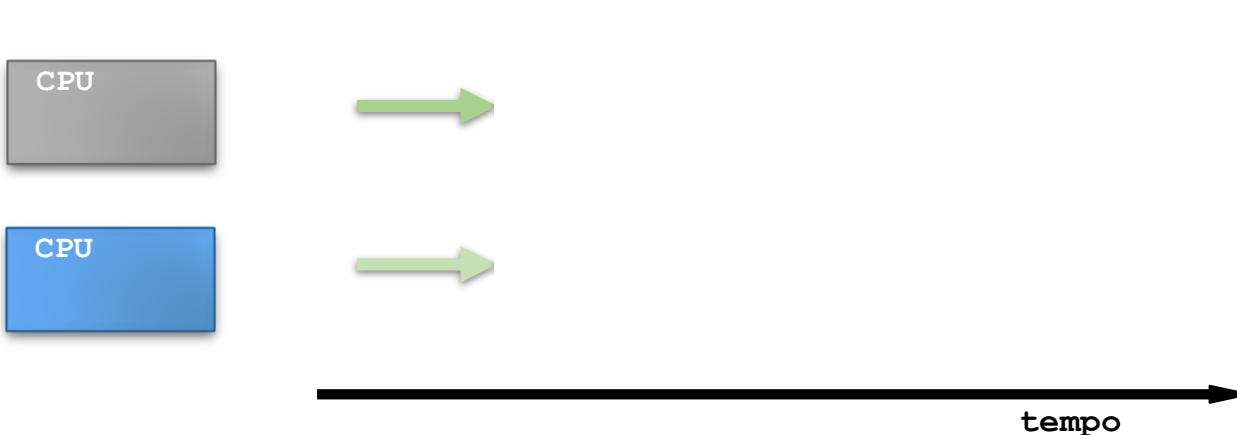
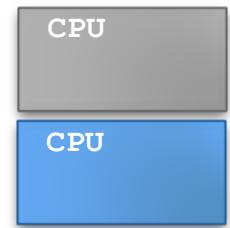
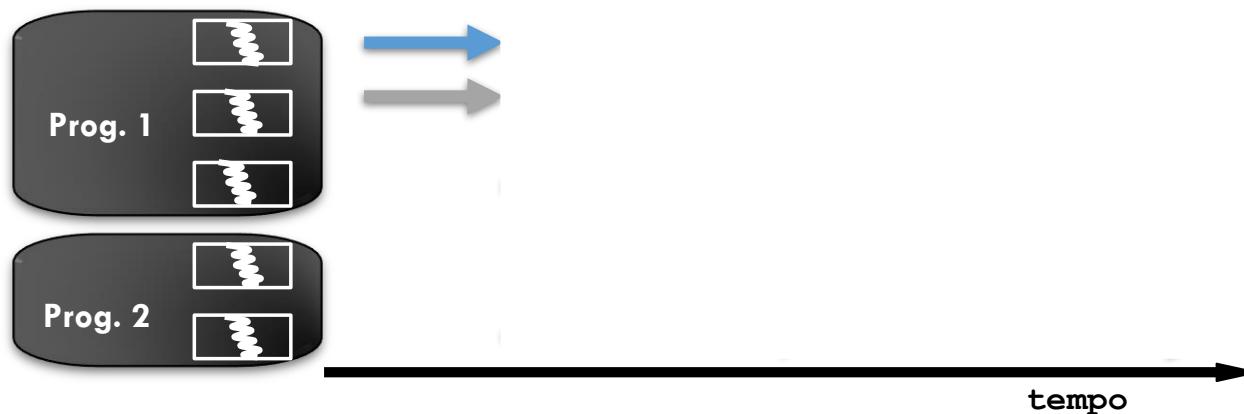
- Páginas(0~99) = **m ciclos**
- Páginas(100~299) = **n + m ciclos**
- Páginas(300~399) = **2n + m ciclos**



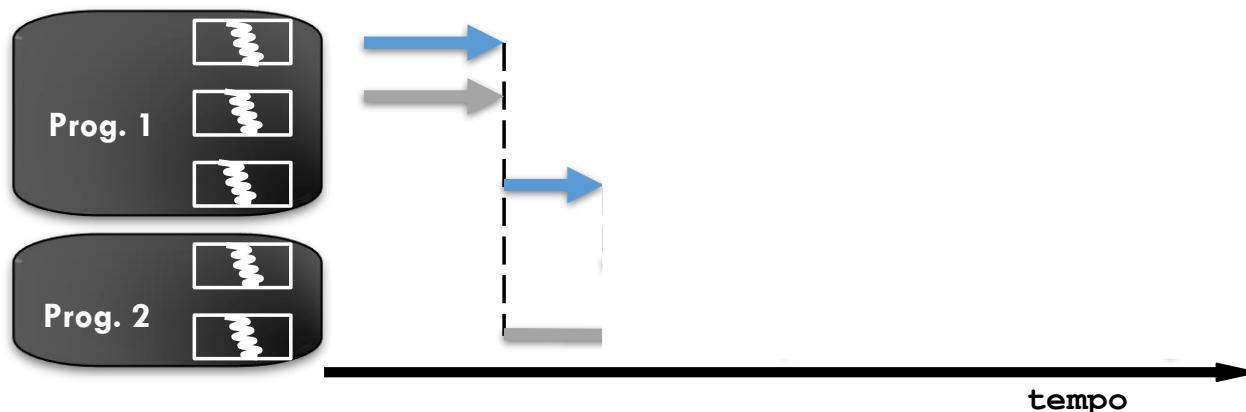
O QUE É TROCA DE CONTEXTO?



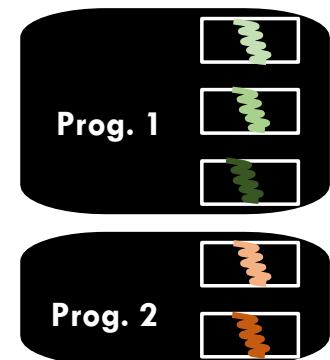
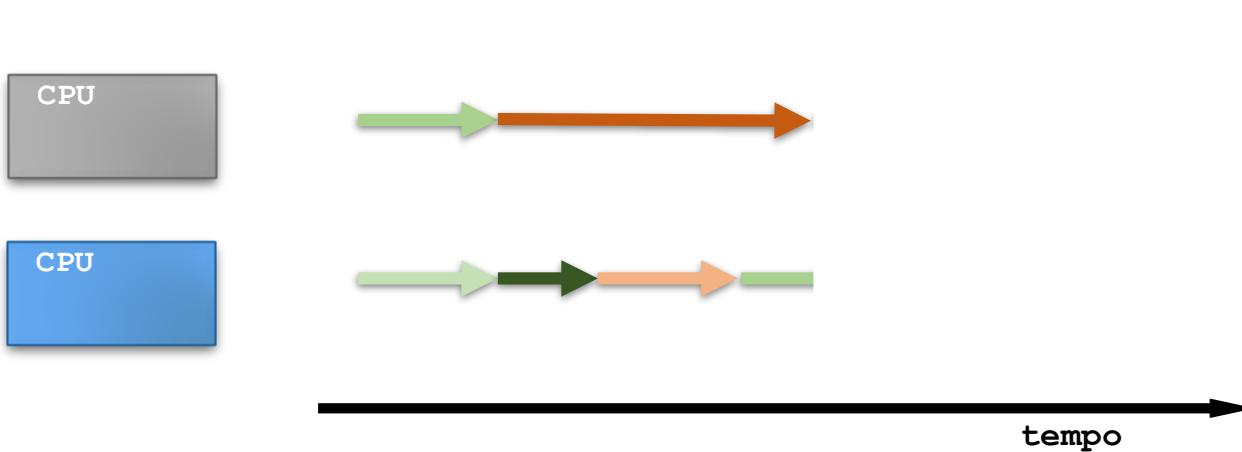
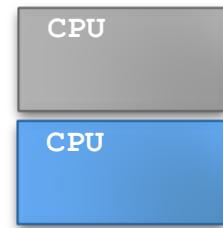
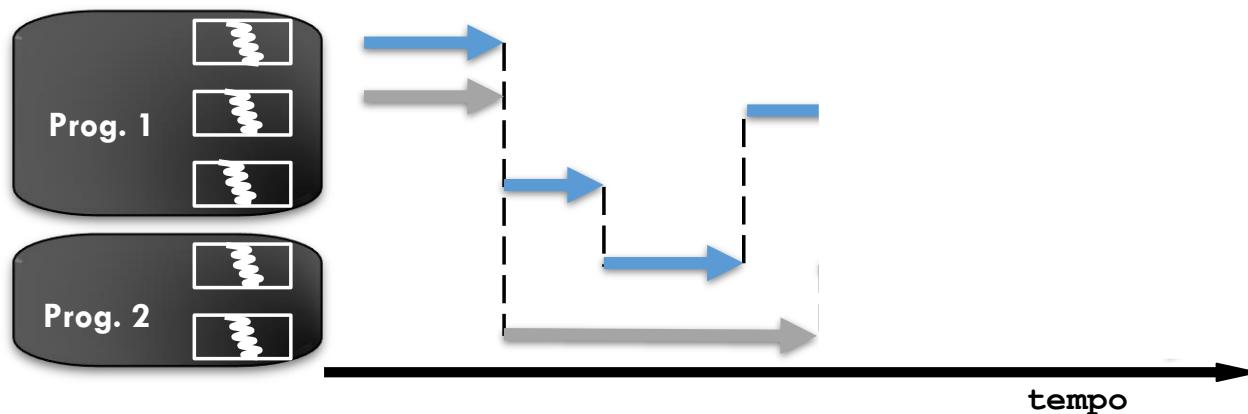
O QUE É TROCA DE CONTEXTO?



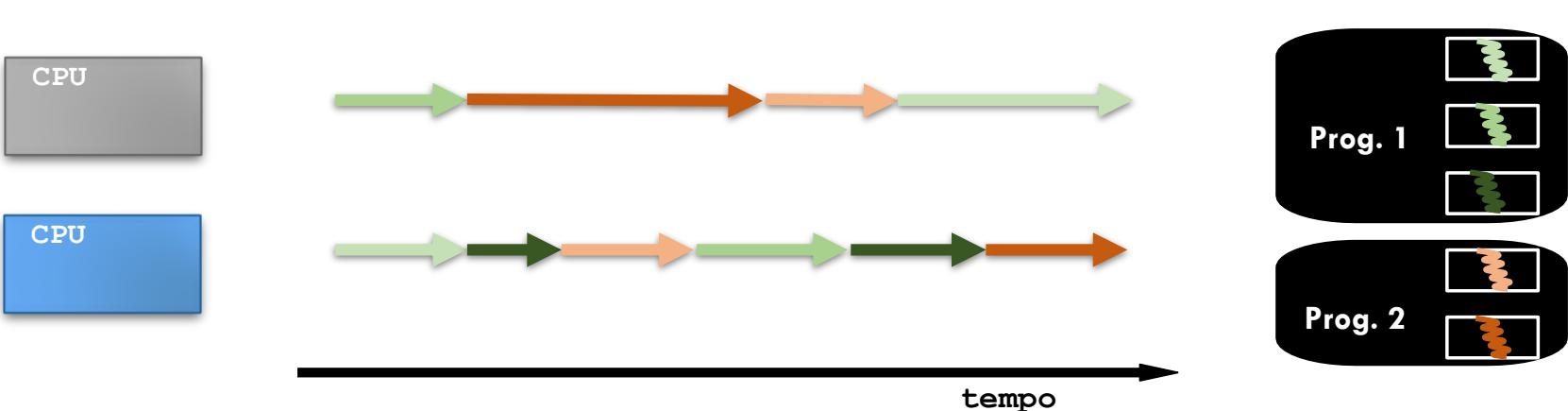
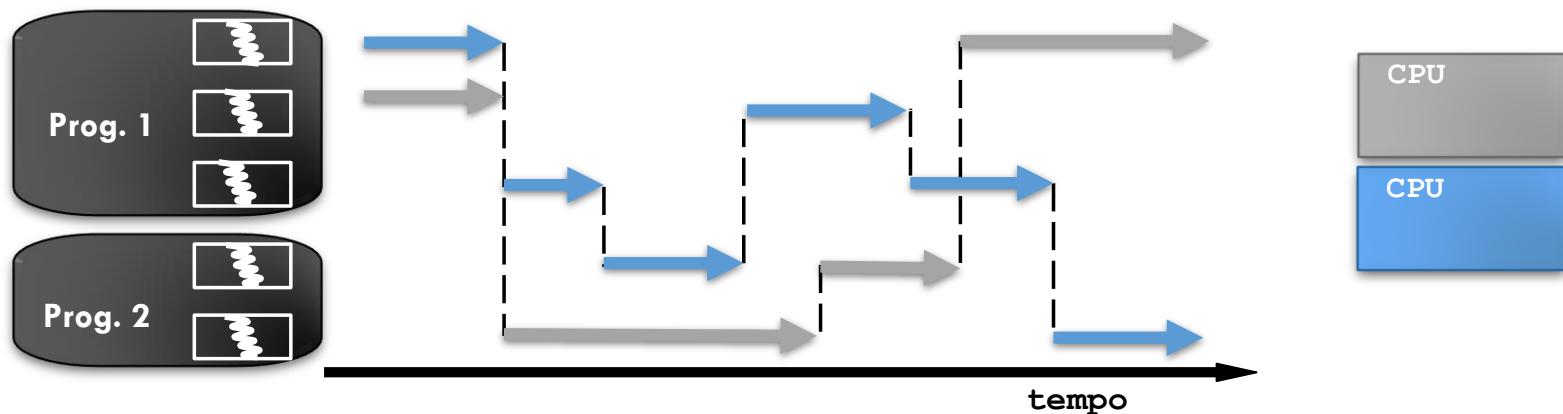
O QUE É TROCA DE CONTEXTO?



O QUE É TROCA DE CONTEXTO?



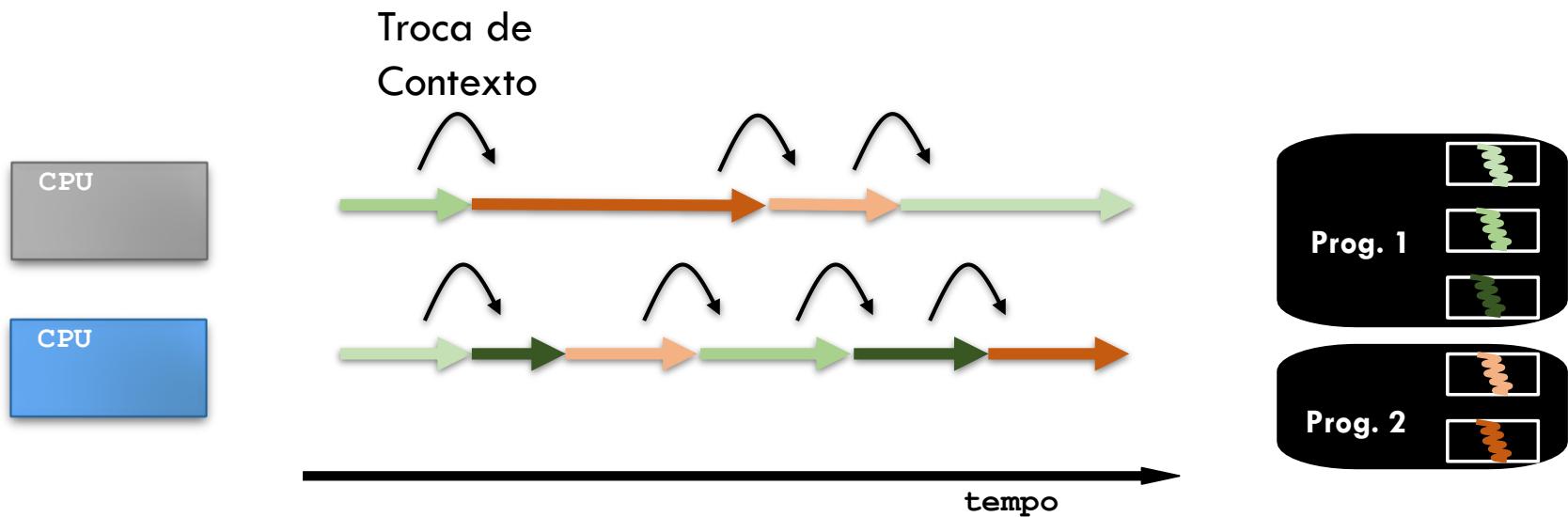
O QUE É TROCA DE CONTEXTO?



O QUE É TROCA DE CONTEXTO?

O procedimento de suspender e continuar processos

Ativado pela interrupção (ex. Interrupção de I/O), preempção de multi-tasks, ou parte do modelo de troca entre kernel/usuário.



MULTITHREADING

A troca de contexto tradicionalmente envolve

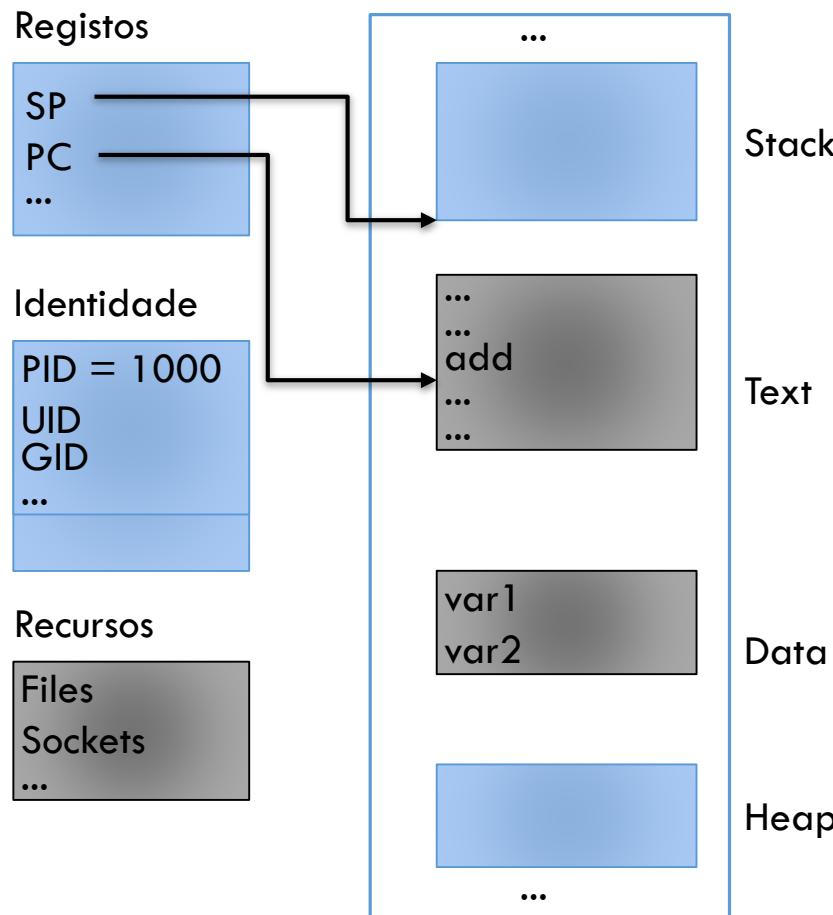
- O despejo do programa A, escrita na memória dos registradores do prog. A
- A carga do programa B, leitura da memória para carregar os registradores do prog. B

Uma das primeiras formas de paralelismo a ser implementada nos processadores comerciais foi o multithreading.

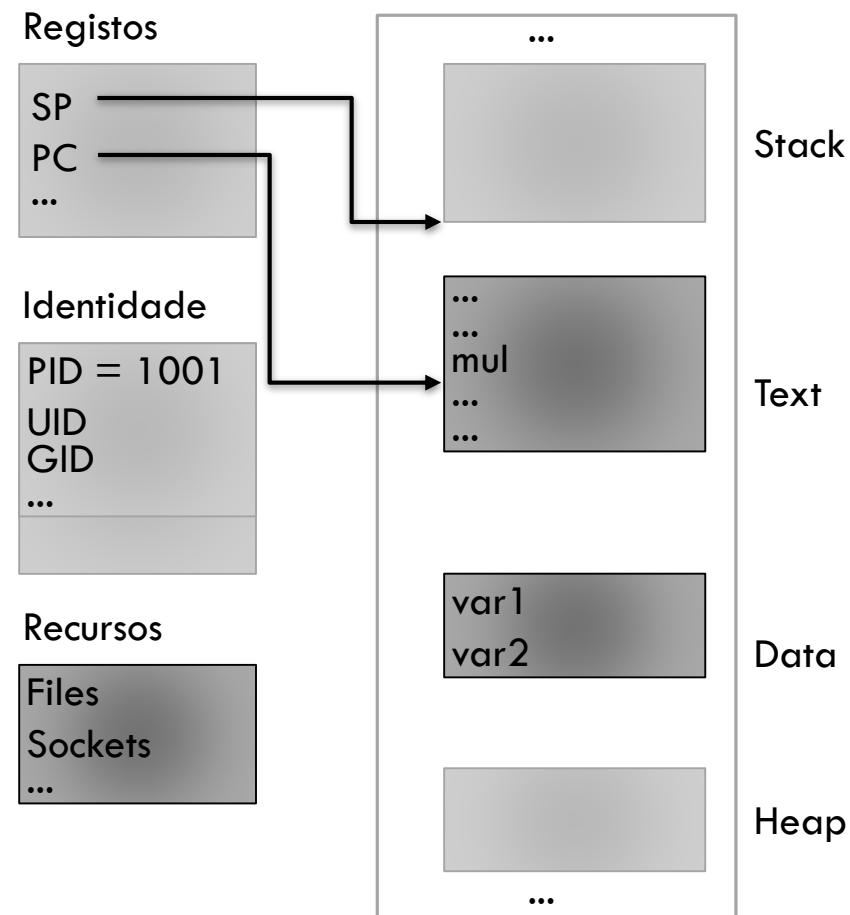
O objetivo foi modificar as arquitetura para dar **suporte a diversas threads ativas**, possibilitando a rápida troca de contexto.

ESTRUTURA DE PROCESSOS

Processo A

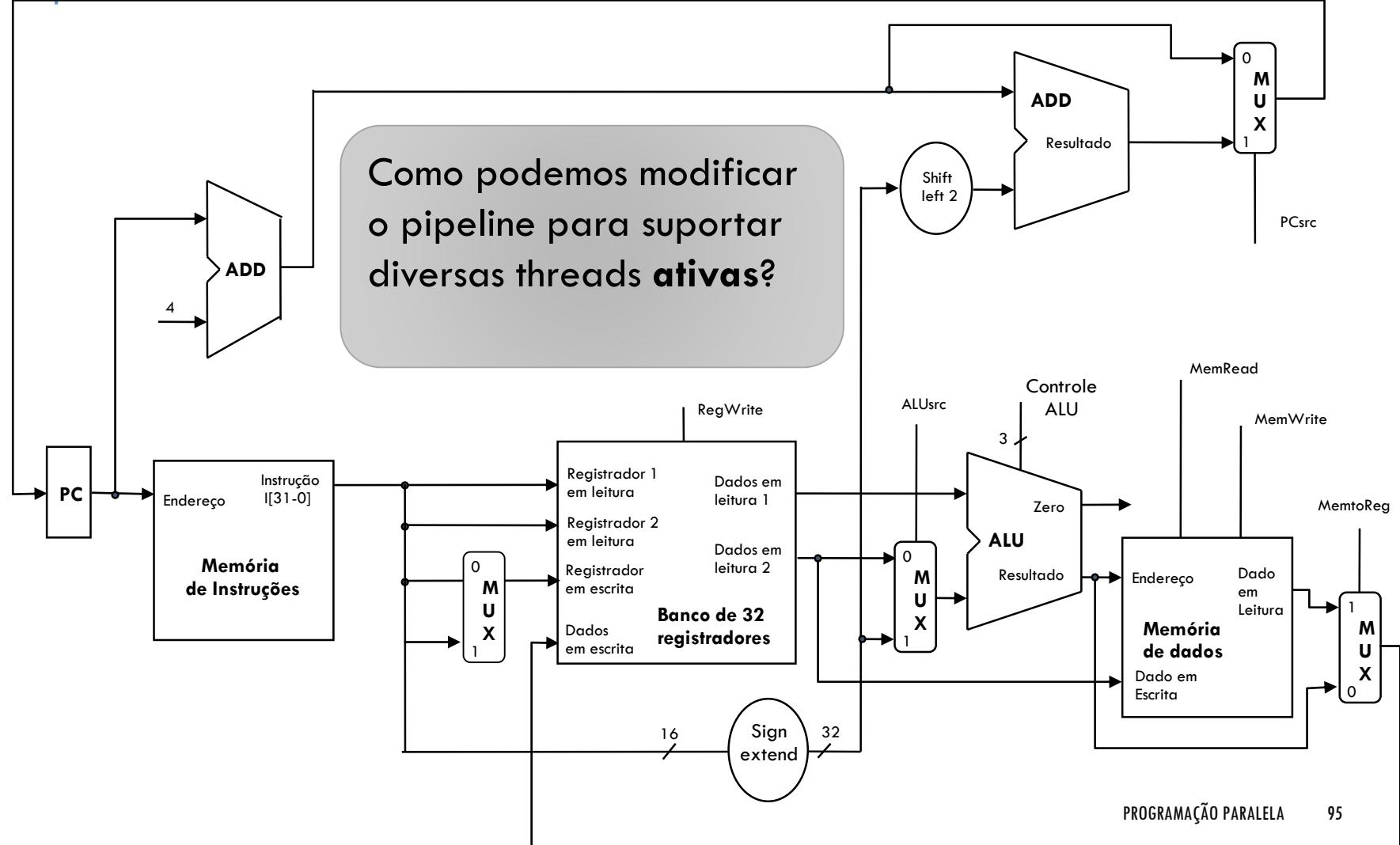


Processo B

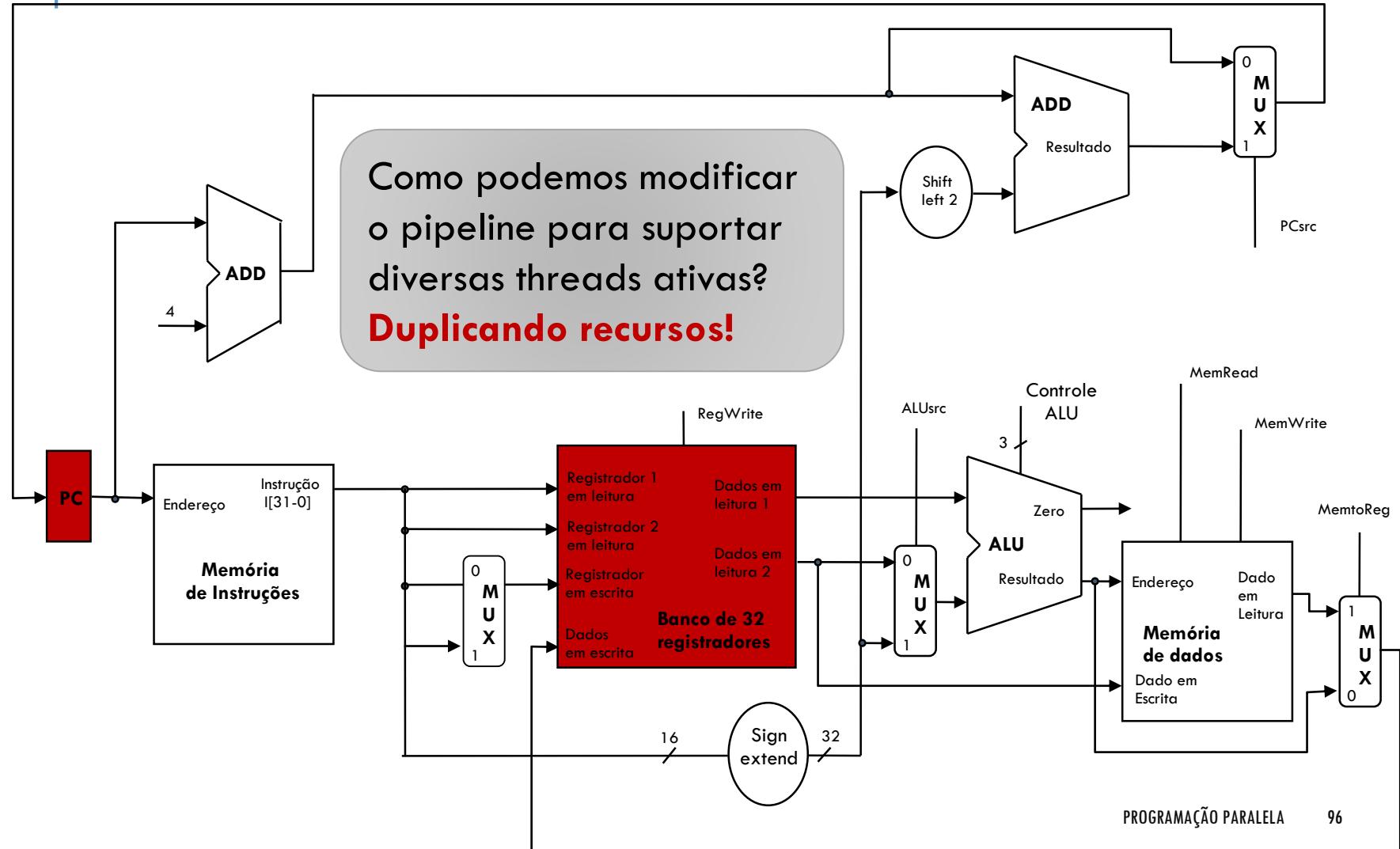


MULTITHREADING

Como podemos modificar o pipeline para suportar diversas threads ativas?



MULTITHREADING



MULTITHREADING

Não necessariamente teremos diversas threads em execução paralela

O processador deve suportar diversas **threads ativas**

Interleaved Multithreading – IMT

- Apenas **uma thread em execução** por ciclo
- Troca de thread em execução a cada ciclo

Blocked Multithreading – BMT

- Apenas **uma thread em execução** por ciclo
- Troca de thread em execução a cada evento de alta latência (ex. interrupção, acesso memória)

Simultaneous Multithreading – SMT

- **Diversas threads em execução** ao mesmo tempo
- Hyper-threading é o nome comercial usado pela Intel

MULTITHREADING

Uma única thread ativa (single-threading)

- O processador armazena apenas os dados de uma thread
- A troca de contexto entre as threads envolve salvar todo o contexto (registradores) na memória, e depois, carregar o contexto da outra thread
- Apenas uma thread pode estar em execução ao mesmo tempo.

MULTITHREADING

Uma única thread ativa (single-threading)

- O processador armazena apenas os dados de uma thread
- A troca de contexto entre as threads envolve salvar todo o contexto (registradores) na memória, e depois, carregar o contexto da outra thread
- Apenas uma thread pode estar em execução ao mesmo tempo.

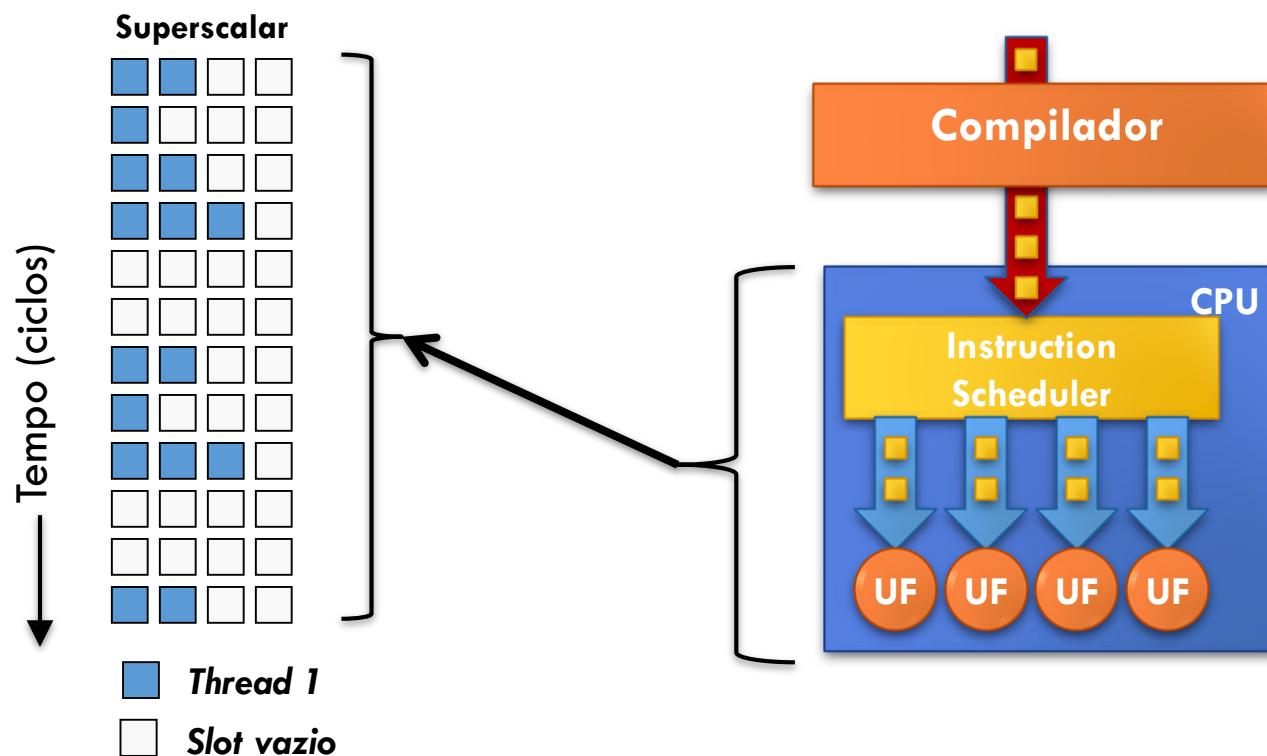
Diversas Threads Ativas

- O processador possui todas as informações sobre diversas threads
- A troca de contexto entre essas threads é feita rapidamente (temos um banco de registradores por thread ativa)
- As threads ativas **podem NÃO** estar em execução ao mesmo tempo.

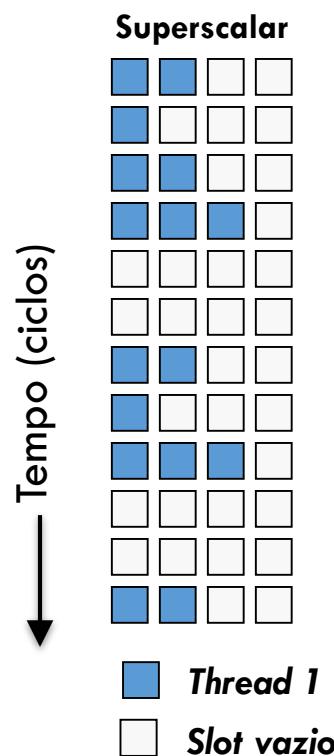
Diversas Threads em Execução

- O processador deve suportar diversas threads ativas
- Porém nesse caso, várias threads **podem** estar ao mesmo tempo no pipeline

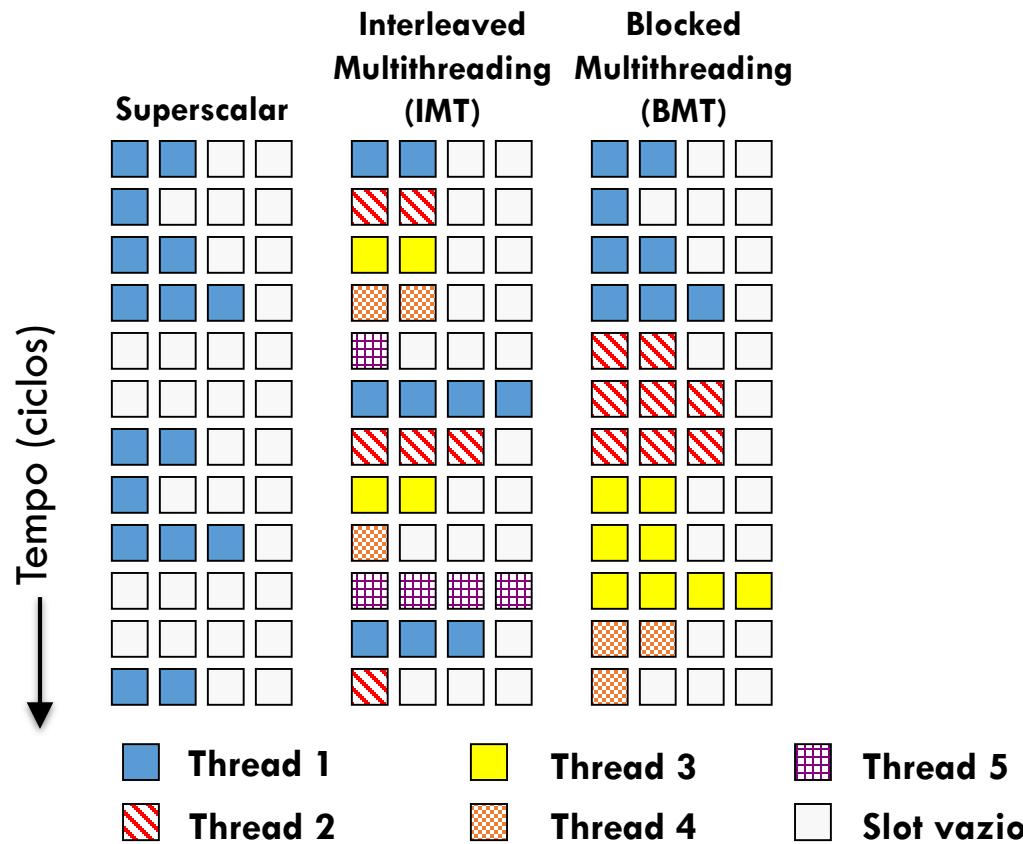
PROCESSADOR SUPERESCALAR



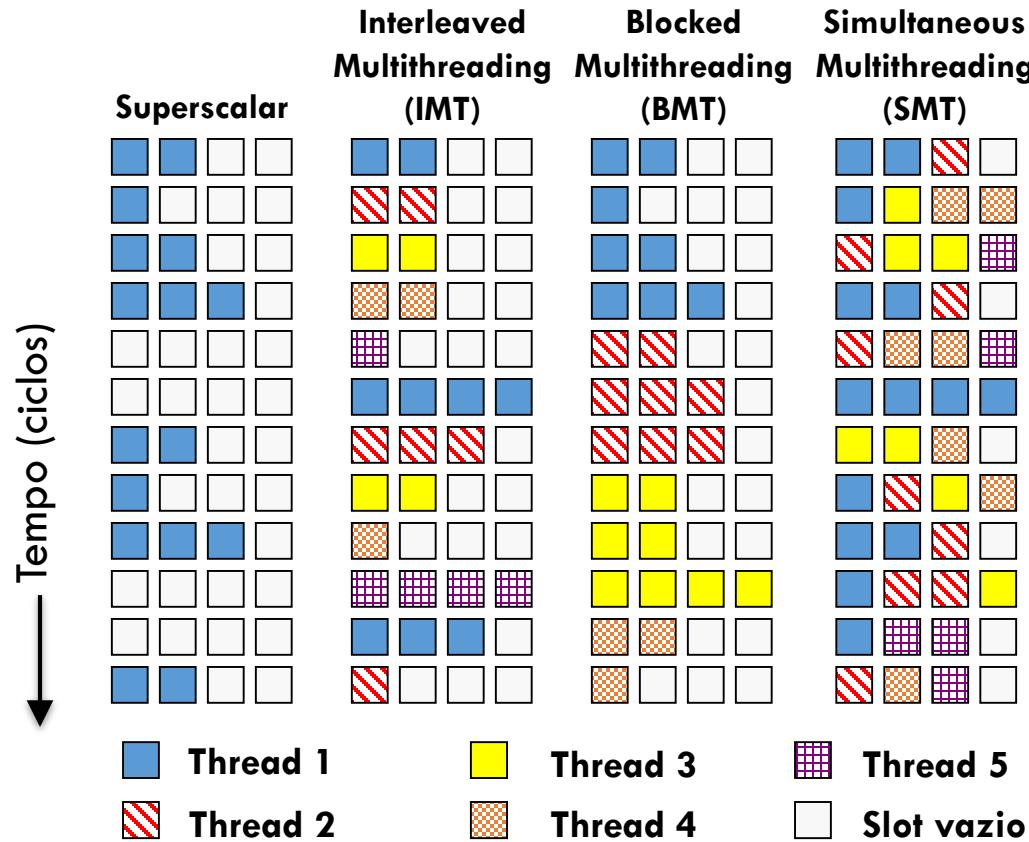
MULTITHREADING



MULTITHREADING



MULTITHREADING



CHIP MULTITHREADING

IMT, BMT E SMT

Deve suportar várias **threads ativas**.

Diversas threads ativas → (IMT/BMT/SMT).

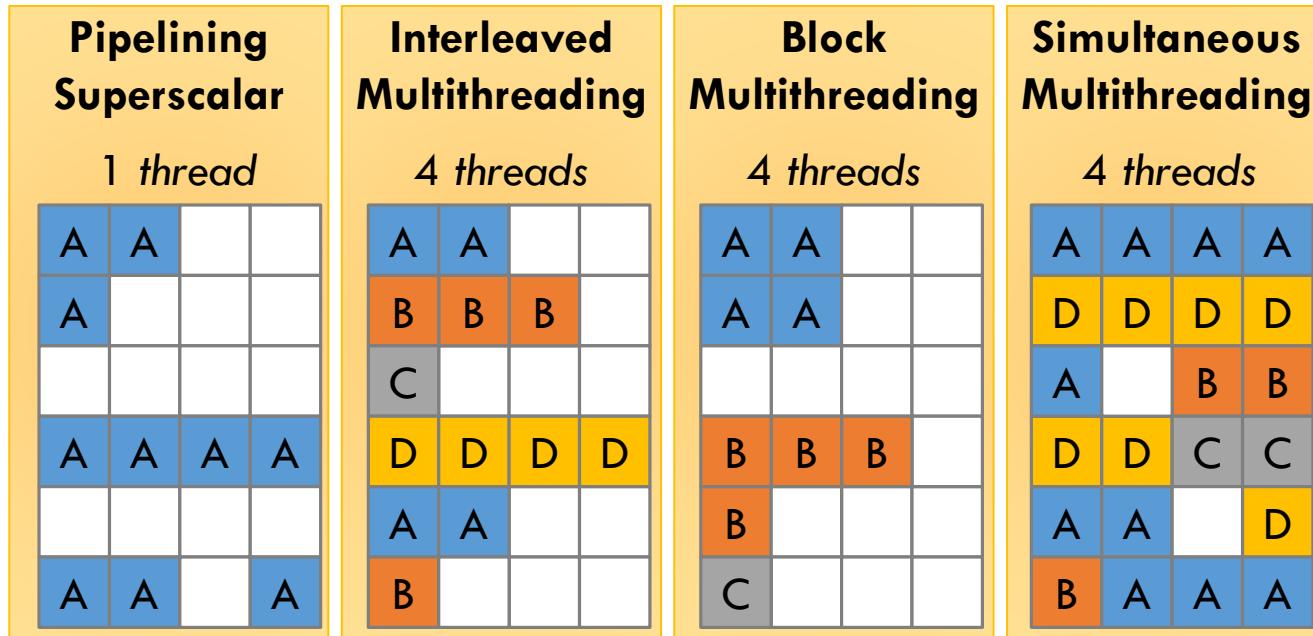
Diversas threads em execução paralela → (apenas o SMT).

Único processador físico.

Várias threads ativas (processadores virtuais).

Usa TLP e representa uma alternativa para ILP.

CHIP MULTITHREADING COM PIPELINE SUPERSCALAR



Latência de Memória	Não esconde	Esconde	Esconde	Esconde
Troca de Contextos	Lenta	Rápida	Rápida	Rápida
Utilização UFs	Baixa	Média	Média	Alta
Uso das Caches	Baixa taxa de faltas	Alta taxa de faltas	Alta taxa de faltas	Alta taxa de faltas

CHIP MULTITHREADING

IMT, BMT E SMT

O “esqueleto” do pipeline não é mudado

Recursos duplicados

- PC, Controle de mapeamento de registradores, etc.

Recursos compartilhados

- Banco de registradores (maior tamanho)
- Fila de instruções
- TLB e Caches (L1 e L2)
- Previsor de desvios

Exemplos de implementação

- Intel, a partir do Pentium 4 HT
- MIPS R10000 estendido
- Alpha estendido (21464)
- PowerPC 604

GANHOS E PERDAS COM MULTITHREADING

Quais aplicações apresentam **benefício?** Como?

Quais aplicações apresentam **perdas?** Como?

GANHOS E PERDAS COM MULTITHREADING

Quais aplicações apresentam **benefício?** Como?

- Aplicações com operações de I/O
- A troca de contexto é rápida o uso das FU's é otimizada
- Aplicações que usem FU's distintas e SMT
- Tentaremos usar todas FU's disponíveis ao mesmo tempo

Quais aplicações apresentam **perdas?** Como?

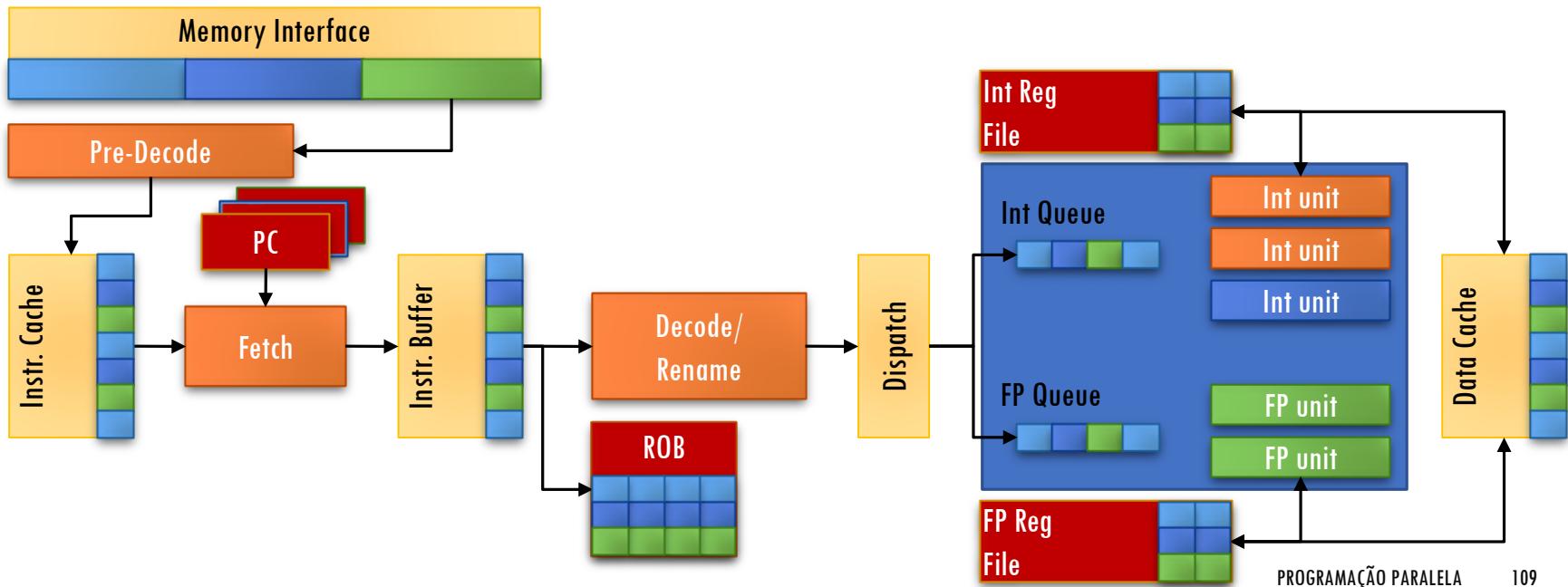
- Aplicações com muito acesso a dados
- Haverá um duelo para usar a cache, no final todos saem perdendo
- Aplicações com instruções de mesmo tipo (sem I/O)
- As aplicações ficaram esperando as FU's e ainda vão duelar pela cache

CHIP MULTITHREADING (IMT, BMT E SMT) COM EXECUÇÃO FORA DE ORDEM

Aumento de hardware em: PC, ROB, Reg File

Implicações: Mem. Principal, Cache Instr., Cache Dados compartilhados

Gargalos: Acesso à dados e instruções, Unidades funcionais



CHIP MULTI-CORE

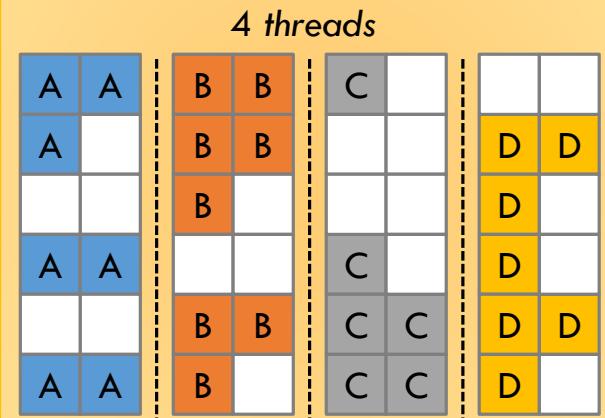
Também corresponde a um CMT. Várias threads ativas e várias em execução.

Ainda investe em ILP com pipelining superscalar

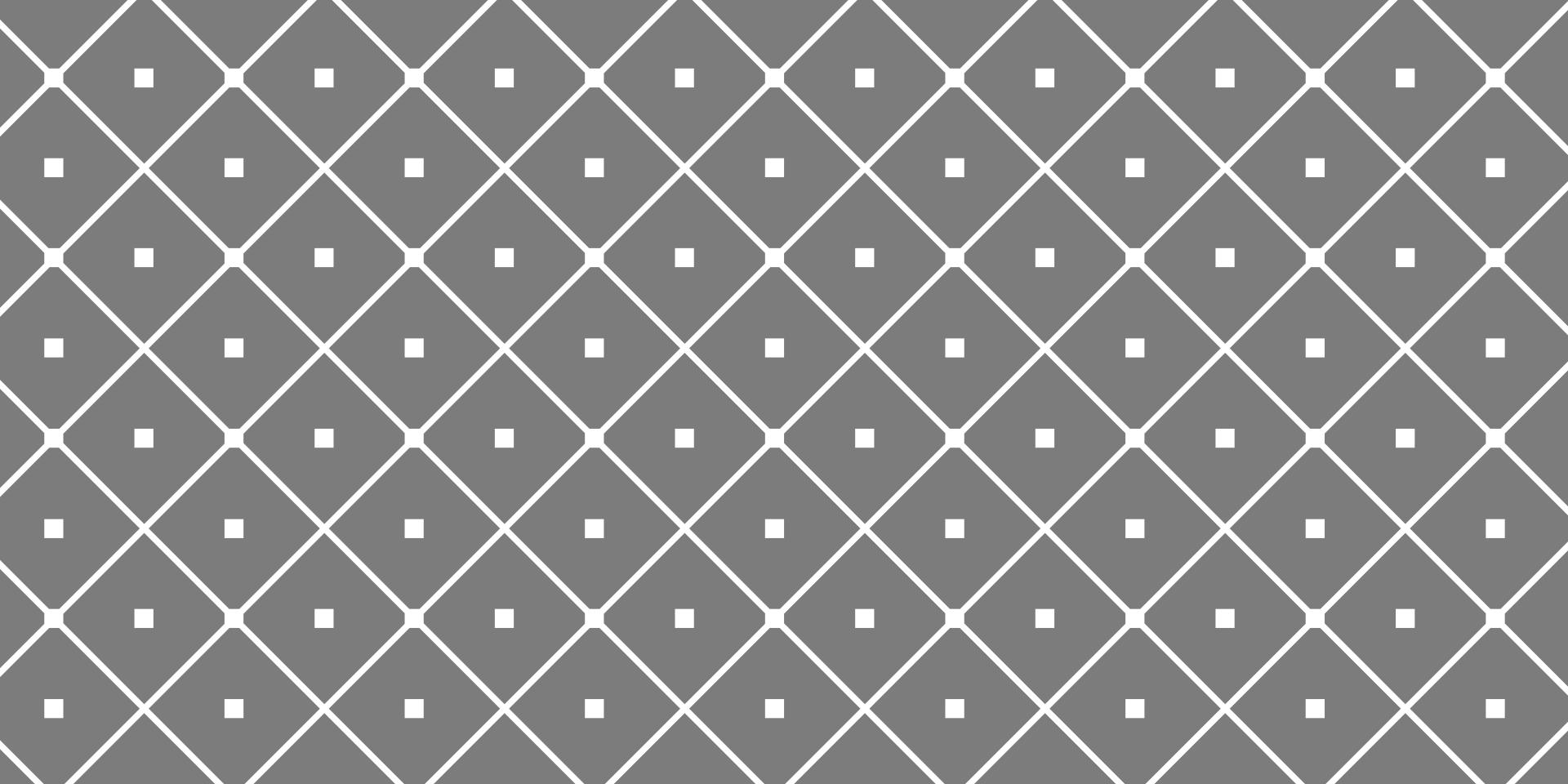
Maior ênfase na mudança de foco para TLP.

Também pode combinar técnicas IMT, BMT e SMT.

CMP – Chip Multiprocessor
(pipeline superscalar em cada núcleo)

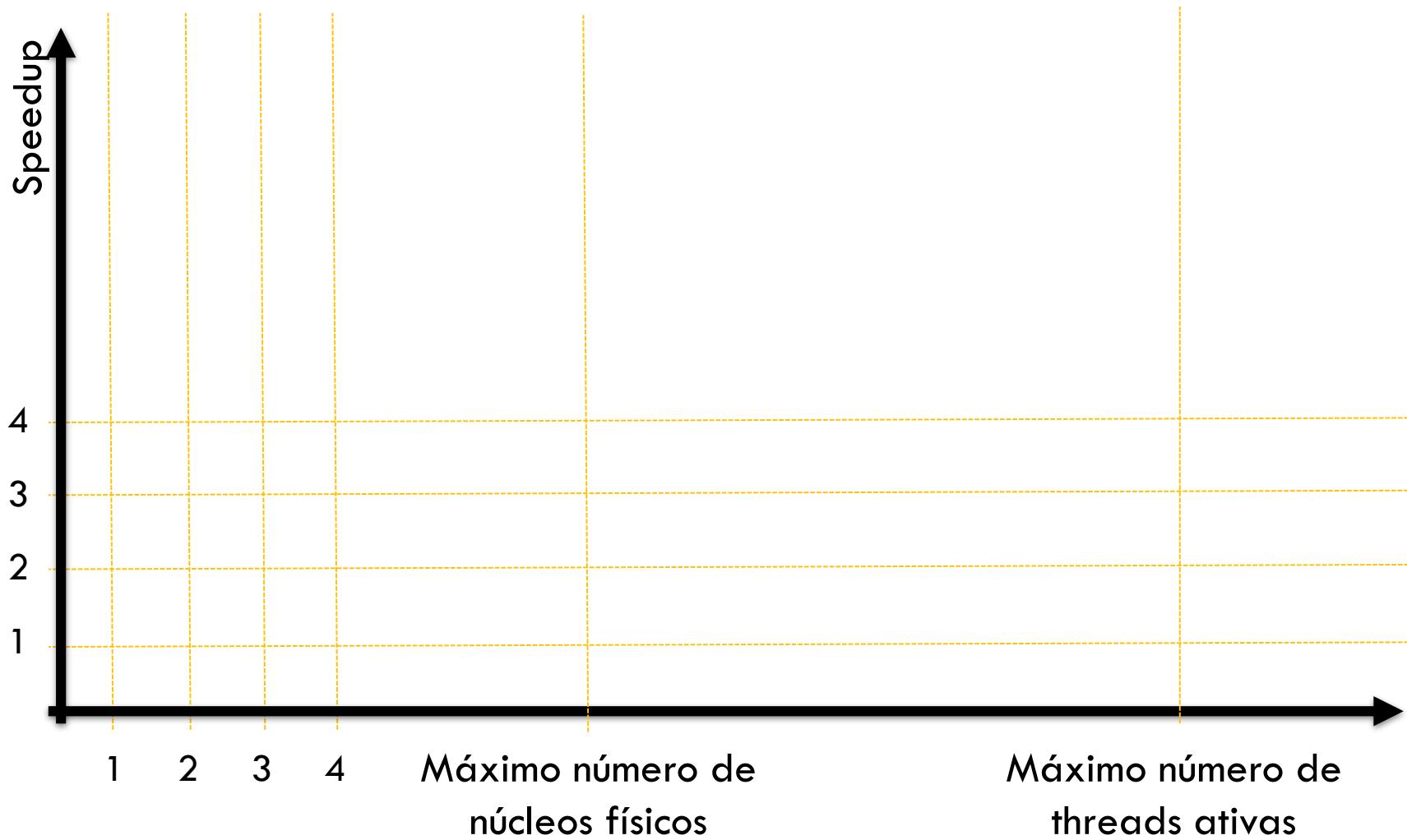


Latência de Memória	Não esconde
Troca de Contextos	Lenta
Utilização UF's	Baixa
Uso das Caches	Baixa taxa de misses

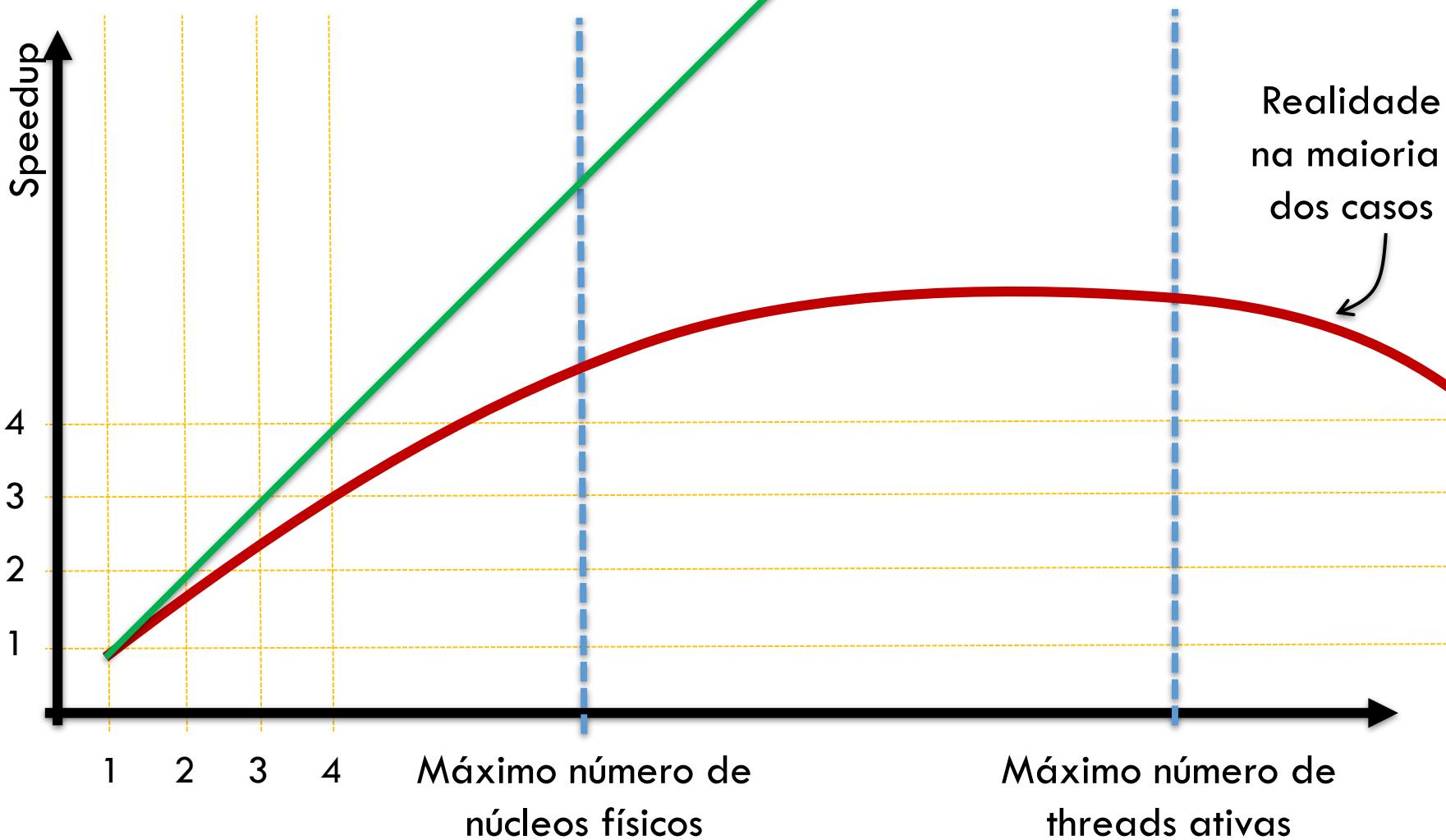


PREVENDO O DESEMPENHO

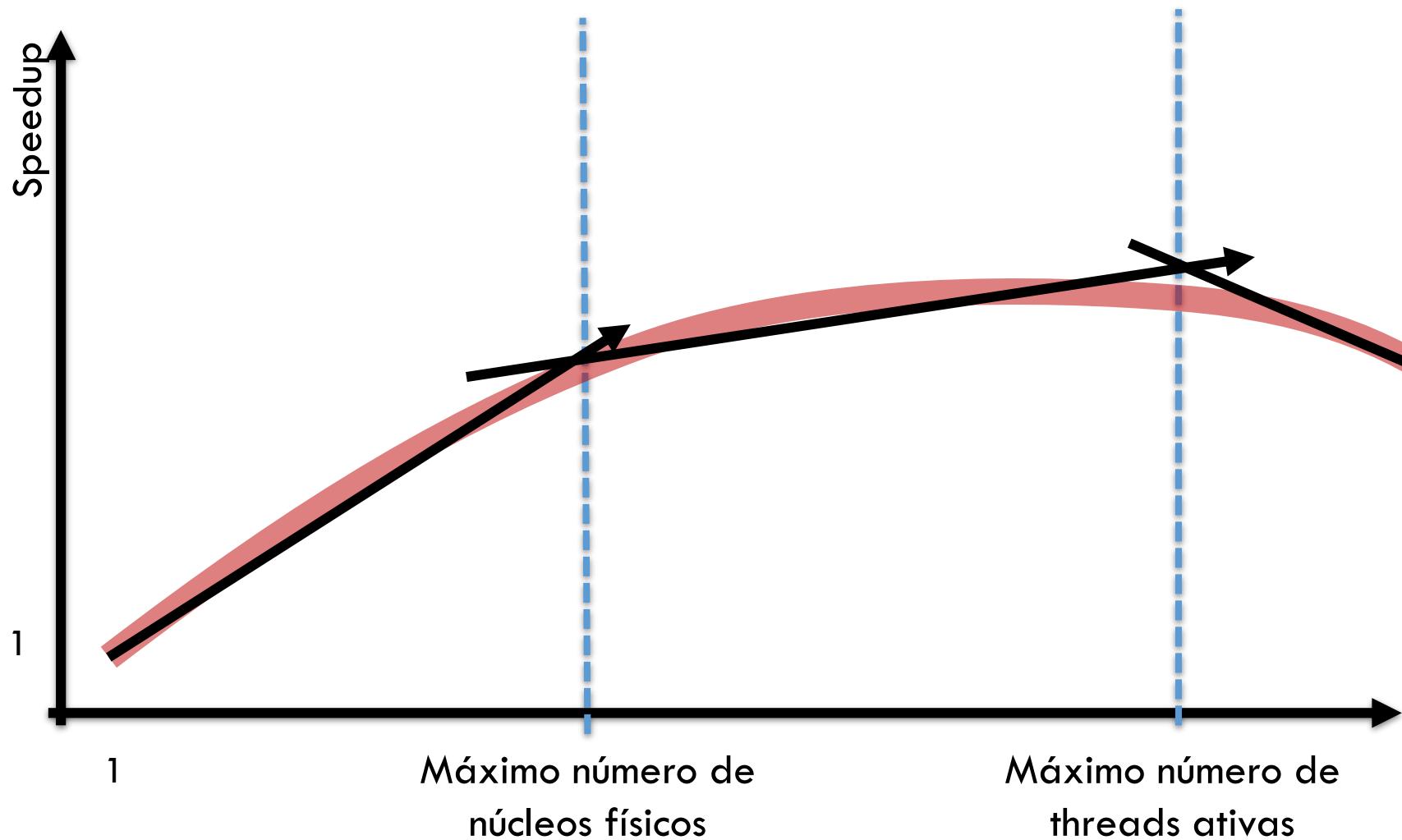
O QUE ESPERAR DO SPEEDUP?



O QUE ESPERAR DO SPEEDUP?



O QUE ESPERAR DO SPEEDUP?



SIMULTANEOUS MULTI-THREADING

HYPER-THREADING (INTEL)

Tecnologia que faz um processador suportar a execução simultânea de diversas threads

Uma das primeiras formas de paralelismo a ser implementada nos processadores comerciais

Um núcleo finge ser dois núcleos!!!

FORMAS PARA ALCANÇAR MAIOR PARALELISMO

Juntar PCs

- Cluster
- Grid
- Cloud

Vetorização

- Instruções
- GPUs

???

- ???
- ???

FORMAS PARA ALCANÇAR MAIOR PARALELISMO

Juntar PCs

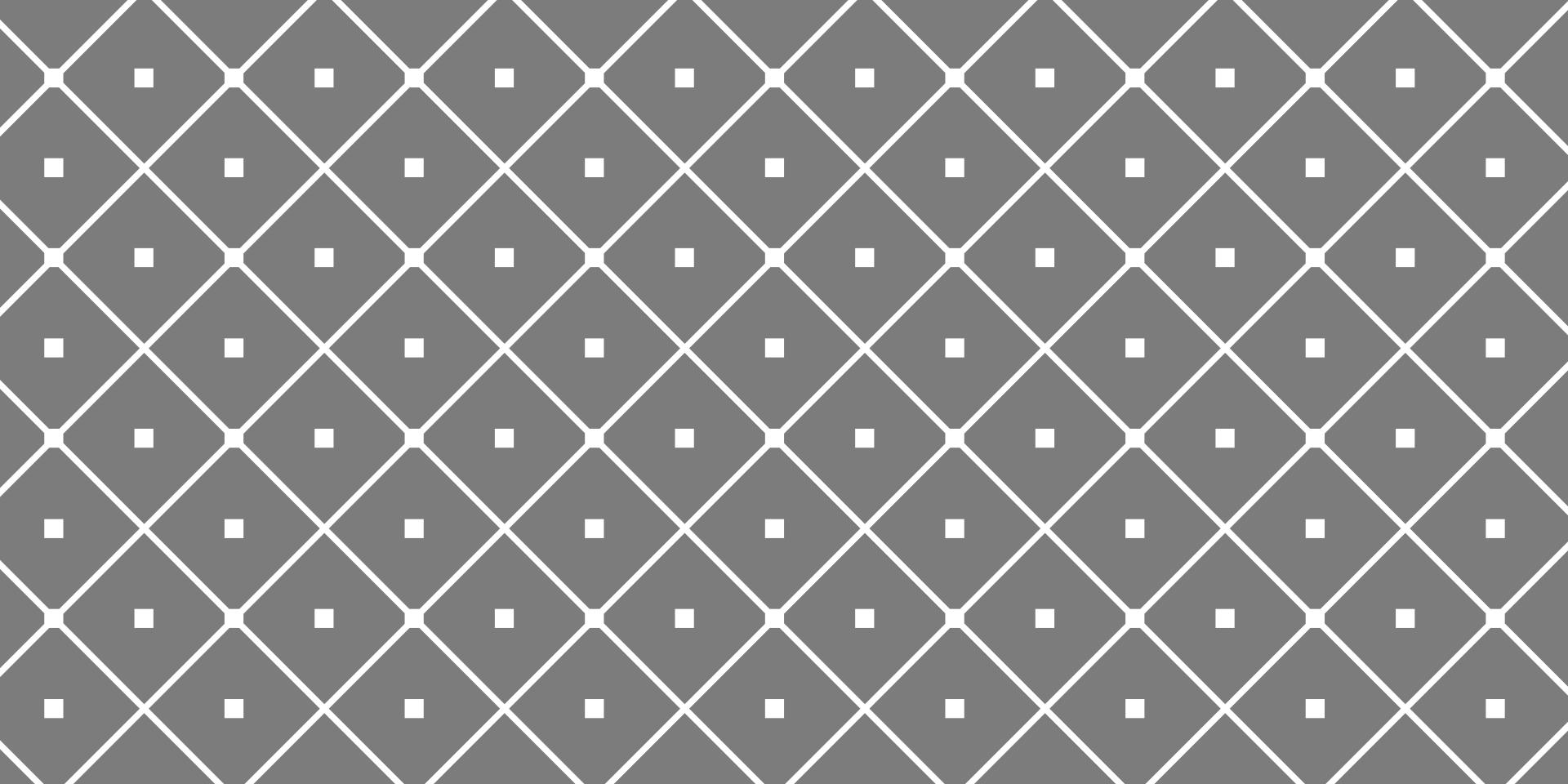
- Cluster
- Grid
- Cloud

Vetorização

- Instruções
- GPUs

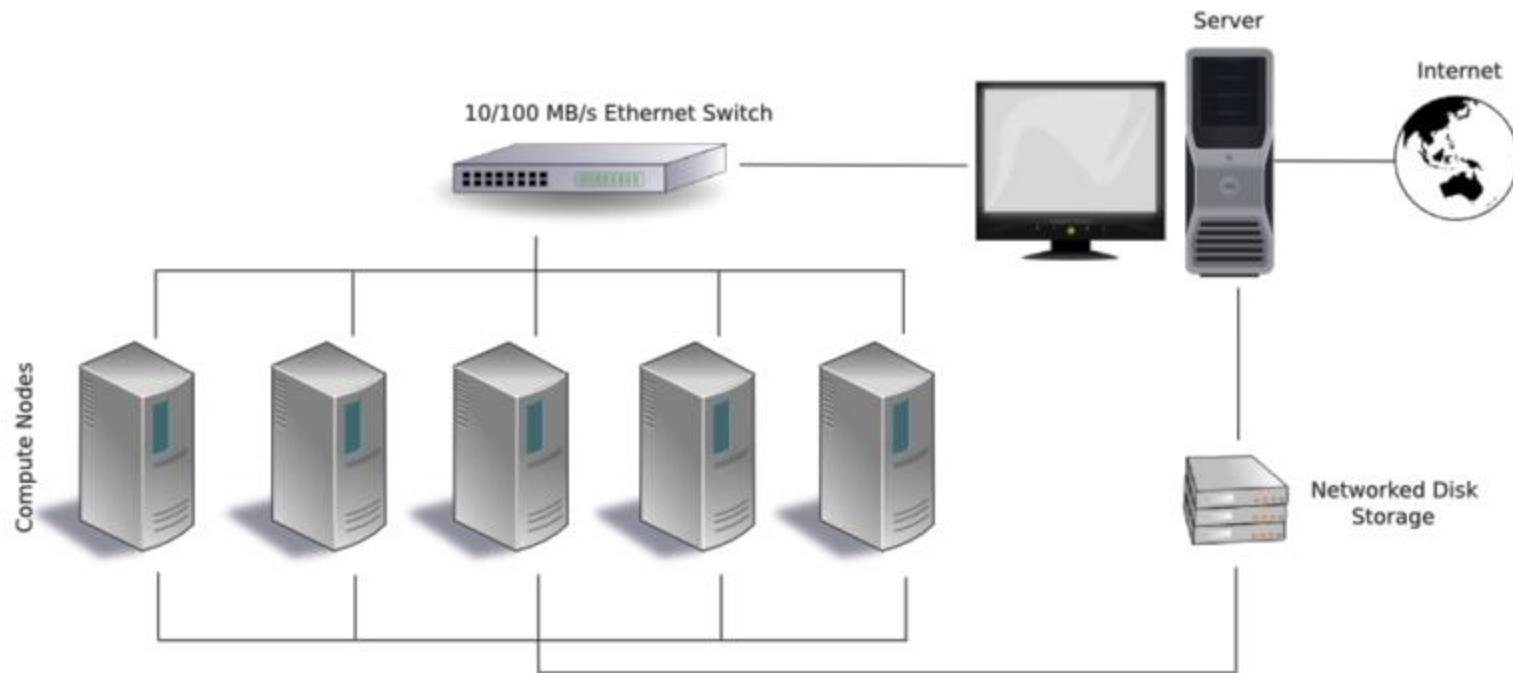
???

- ???
- ???



CLUSTER HPC

CLUSTER DE MÁQUINAS



CLUSTER DE MÁQUINAS

Paralelismo: Bastante claro em nível de processos.

Diversos espaços de endereçamento.

SO para cada máquina: Linux ou proprietário.

Tecnologias de Rede: Ethernet, InfiniBand.

Escalonador de Jobs: SLURM, LSF, Kubernets, PBS.

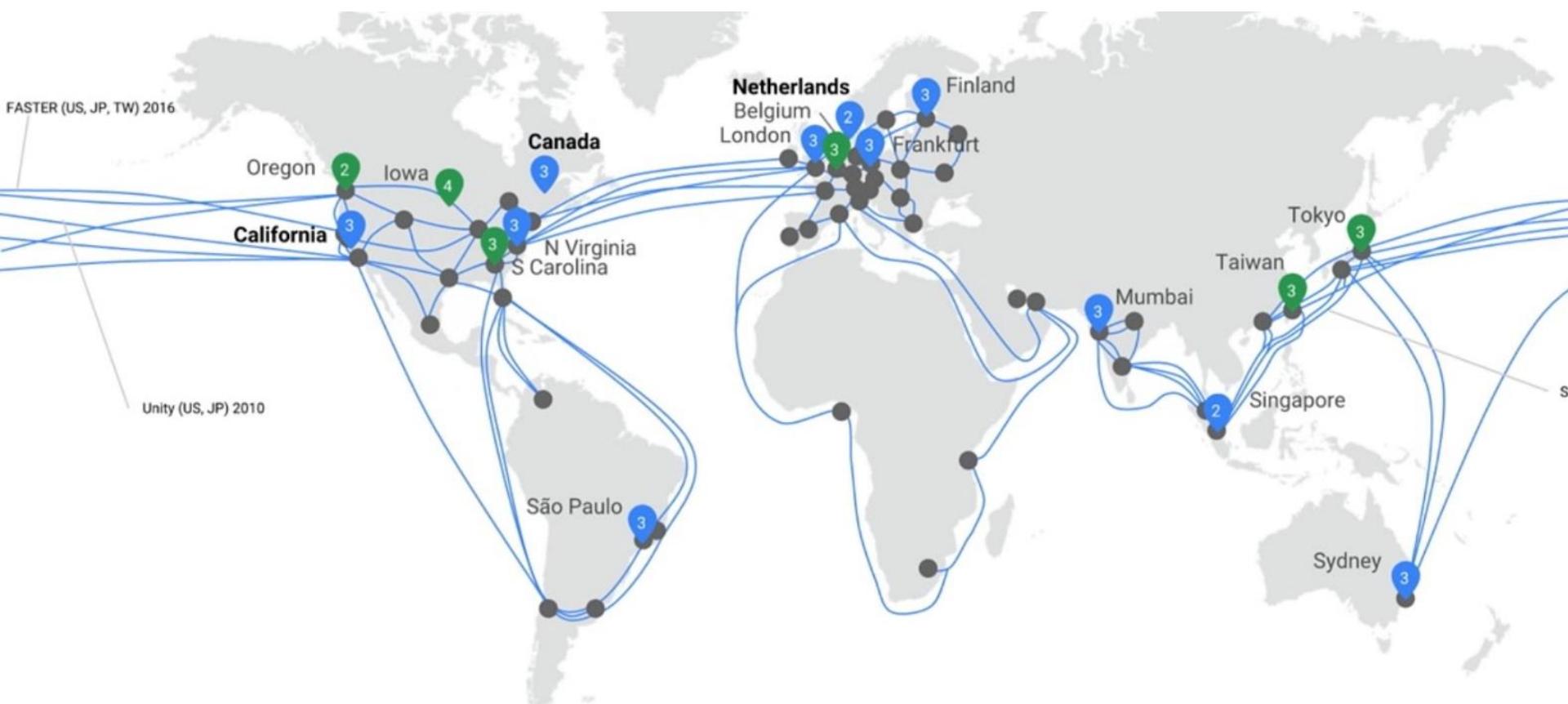
COMPUTER RACK



DATA CENTER



DATA CENTERS (GOOGLE CLOUD - 2017)



COMPUTAÇÃO EM NUVEM





**Não existe computação na nuvem...
...é apenas o computador de outra pessoa**

FORMAS PARA ALCANÇAR MAIOR PARALELISMO

Juntar PCs

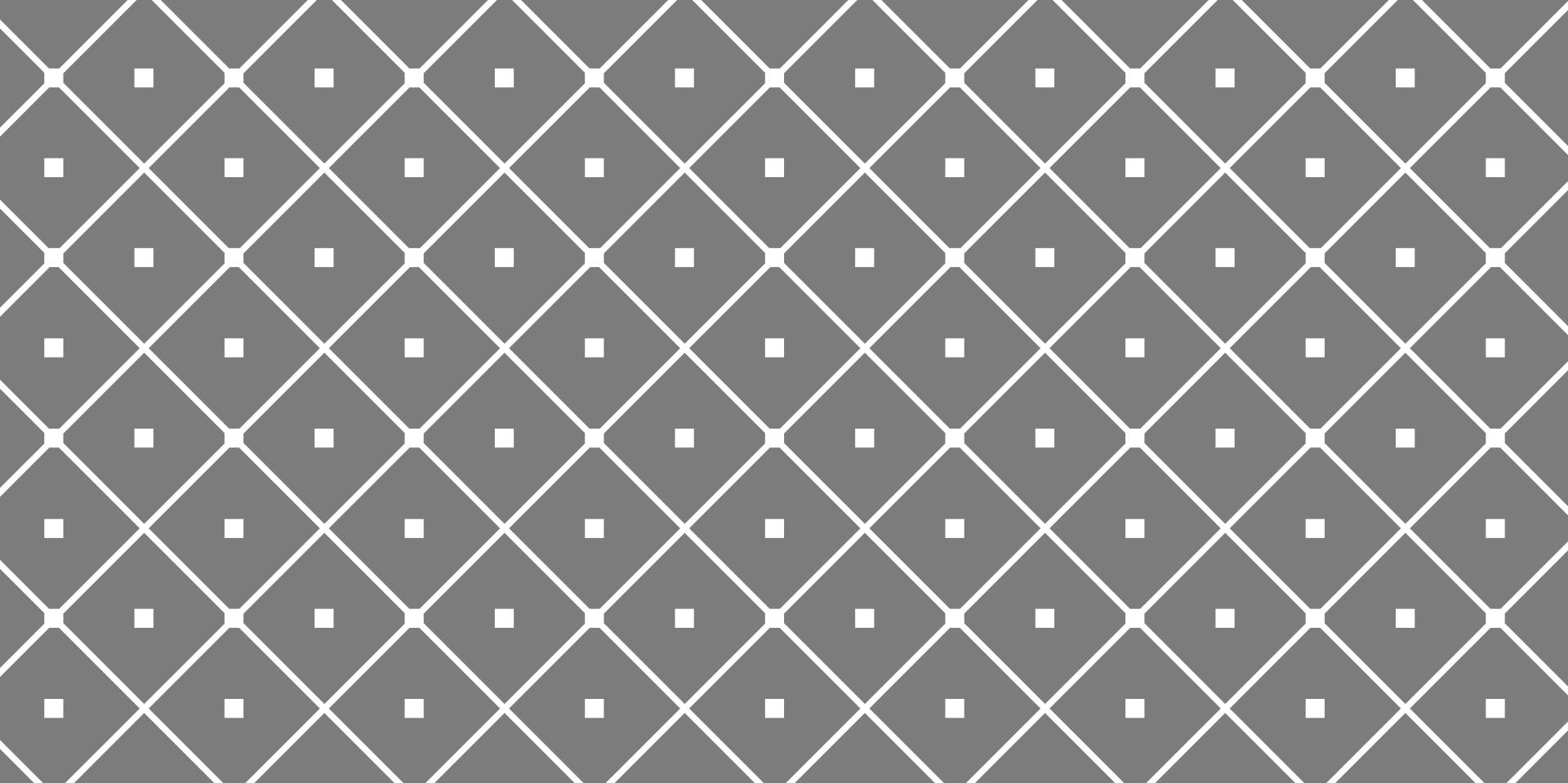
- Cluster
- Grid
- Cloud

Vetorização

- CPUs
- GPUs

???

- ???
- ???

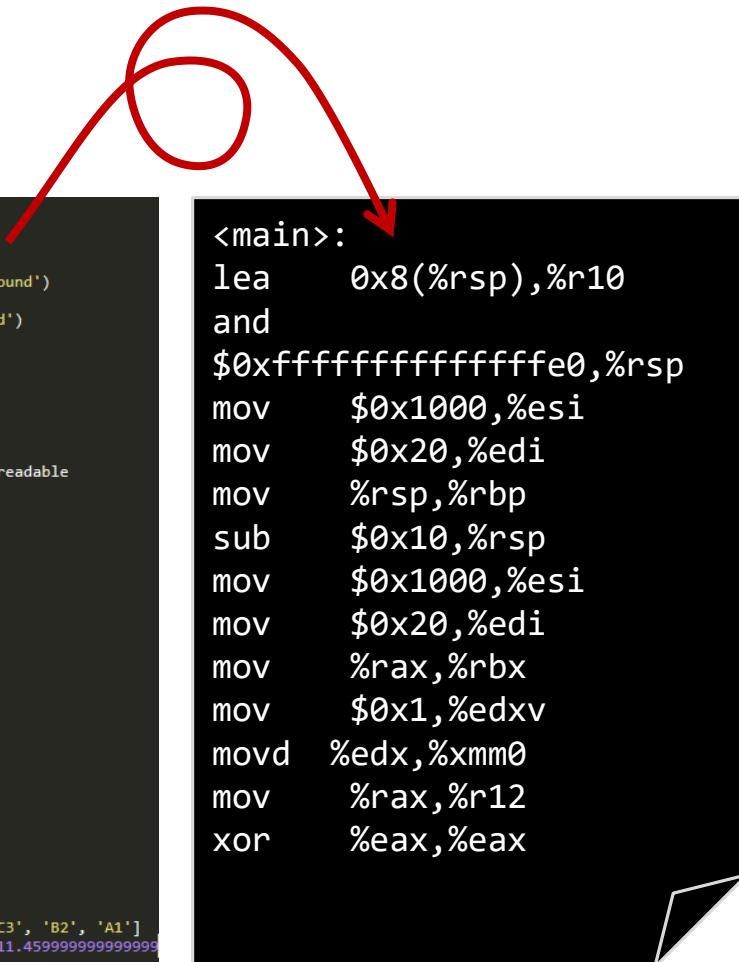


MÁQUINAS E INSTRUÇÕES VETORIAIS

SIMD – Single Instruction
Multiple Data

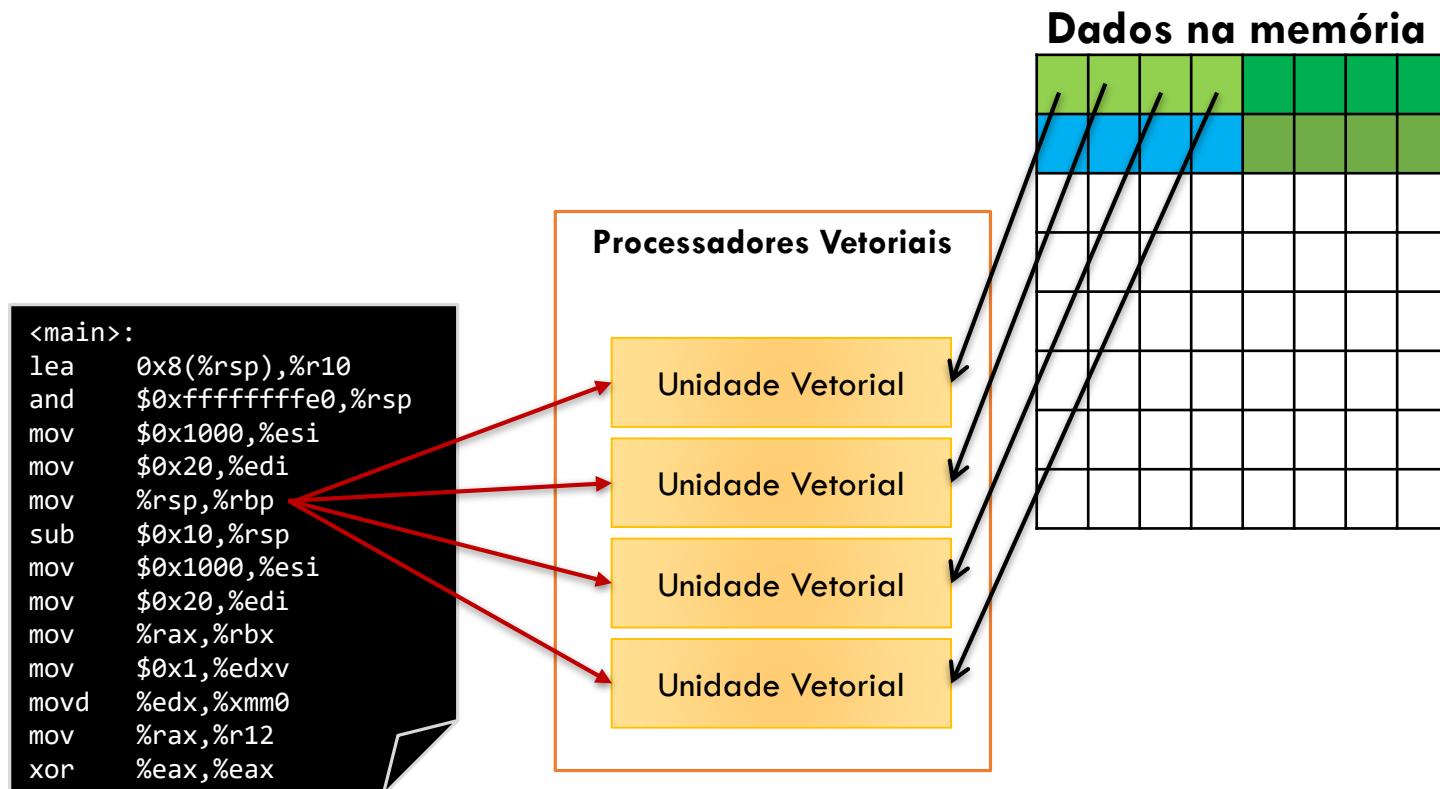
VAMOS CONSIDERAR UM CÓDIGO EXEMPLO

```
1 def dijkstra(graph,src,dest,visited=[],distances={},predecessors={}):
2     """ calculates a shortest path tree rooted in src
3
4     if src not in graph:
5         raise TypeError('The root of the shortest path tree cannot be found')
6     if dest not in graph:
7         raise TypeError('The target of the shortest path cannot be found')
8     if src == dest:
9         path=[]
10        pred=dest
11    while pred != None:
12        path.append(pred)
13        pred=predecessors.get(pred,None)
14        readable=path[0]
15        for index in range(1,len(path)): readable = path[index] + '-->' + readable
16        print('shortest path - array: '+str(path))
17        print("path: "+readable+", cost="+str(distances[dest]))
18    else :
19        if not visited:
20            distances[src]=0
21            for neighbor in graph[src] :
22                if neighbor not in visited:
23                    new_distance = distances[src] + graph[src][neighbor]
24                    if new_distance < distances.get(neighbor,float('inf')):
25                        distances[neighbor] = new_distance
26                        predecessors[neighbor] = src
27            visited.append(src)
28            unvisited={}
29            for k in graph:
30                if k not in visited:
31                    unvisited[k] = distances.get(k,float('inf'))
32            x=min(unvisited, key=unvisited.get)
33            dijkstra(graph,x,dest,visited,distances,predecessors)
34    if __name__ == "__main__":
35        graph = {'A1': { 'B1': 1, 'B2': 1.41, 'A2': 1},
36                 'A2': { 'B1': 1.41, 'B2': 1, 'B3': 1.41, 'A1': 1, 'A3': 1},
37                 .....
38        dijkstra(graph,'A1','H9')
39        shortest path - array: ['H9', 'G8', 'F8', 'E7', 'D6', 'D5', 'C4', 'C3', 'B2', 'A1']
40        path: A1-->B2-->C3-->C4-->D5-->D6-->E7-->F8-->G8-->H9, cost=11.459999999999999
```



```
<main>:
    lea    0x8(%rsp),%r10
    and
    $0xfffffffffffffe0,%rsp
    mov   $0x1000,%esi
    mov   $0x20,%edi
    mov   %rsp,%rbp
    sub   $0x10,%rsp
    mov   $0x1000,%esi
    mov   $0x20,%edi
    mov   %rax,%rbx
    mov   $0x1,%edxv
    movd  %edx,%xmm0
    mov   %rax,%r12
    xor   %eax,%eax
```

MÁQUINAS VETORIAIS (SIMD)



MÁQUINAS VETORIAIS (SIMD)

```
<main>:  
lea    0x8(%rsp),%r10  
and   $0xfffffffffe0,%rsp  
mov    $0x1000,%esi  
mov    $0x20,%edi  
mov    %rsp,%rbp  
sub   $0x10,%rsp  
mov    $0x1000,%esi  
mov    $0x20,%edi  
mov    %rax,%rbx  
mov    $0x1,%edxv  
movd   %edx,%xmm0  
mov    %rax,%r12  
xor    %eax,%eax
```

Processadores Vetoriais

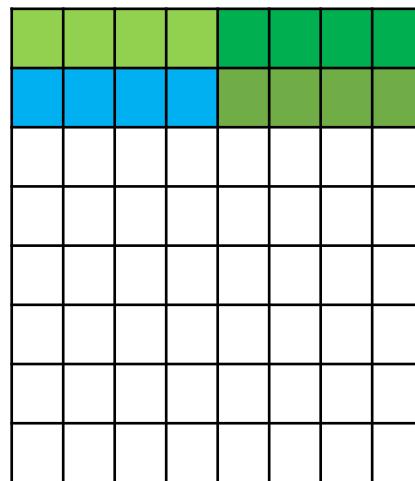
Unidade Vetorial

Unidade Vetorial

Unidade Vetorial

Unidade Vetorial

Dados na memória



MÁQUINAS VETORIAIS (SIMD)

```
<main>:  
lea    0x8(%rsp),%r10  
and   $0xfffffffffe0,%rsp  
mov    $0x1000,%esi  
mov    $0x20,%edi  
mov    %rsp,%rbp  
sub   $0x10,%rsp  
mov    $0x1000,%esi  
mov    $0x20,%edi  
mov    %rax,%rbx  
mov    $0x1,%edxv  
movd   %edx,%xmm0  
mov    %rax,%r12  
xor    %eax,%eax
```

Processadores Vetoriais

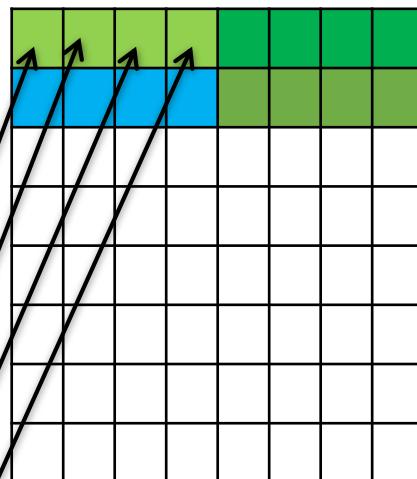
Unidade Vetorial

Unidade Vetorial

Unidade Vetorial

Unidade Vetorial

Dados na memória



EFICIÊNCIA DE ARQUITETURAS SIMD

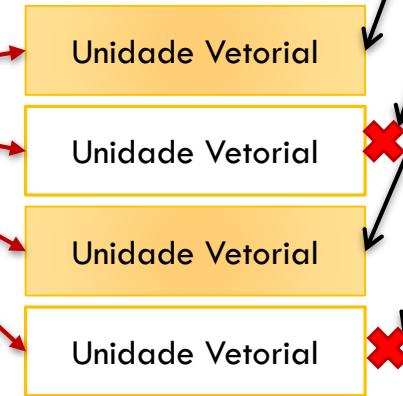
[+] SIMD são **eficientes** quando processam vetores/matrizes em laços “for”

[-] SIMD são **ineficientes** em aplicações do tipo “if/case”

- Cada unidade de execução executa operação diferente, dependendo do dado

MÁQUINAS SIMD

```
<main>:  
lea    0x8(%rsp),%r10  
and   $0xfffffffffe0,%rsp  
mov    $0x1000,%esi  
mov    $0x20,%edi  
mov    %rsp,%rbp  
sub   $0x10,%rsp  
mov    $0x1000,%esi  
mov    $0x20,%edi  
mov    %rax,%rbx  
mov    $0x1,%edxv  
movd  %edx,%xmm0  
mov    %rax,%r12  
xor    %eax,%eax
```



Dados na memória

PROCESSADORES VETORIAIS (1966)

Exemplo: Cray C-90

- Registrador vetorial tem 64×64 bits
- 2 pipelines funcionais vetoriais

Outras máquinas vetoriais

- Convex C3, DEC VAX 9000, Fujitsu VP2000, Hitachi S-810, IBM 390/VF

EARTH SIMULATOR

CONSTRUÍDO PELA NEC \$500MI

SUPERCOMPUTADOR TOP 1 ENTRE 2002~2004



EARTH SIMULATOR
CONSTRUÍDO PELA NEC \$500MI
SUPERCOMPUTADOR TOP 1 ENTRE
2002~2004



GPUS TAMBÉM SÃO VETORIAIS !!!

Exemplo: RTX A5000

16.384 CUDA Cores

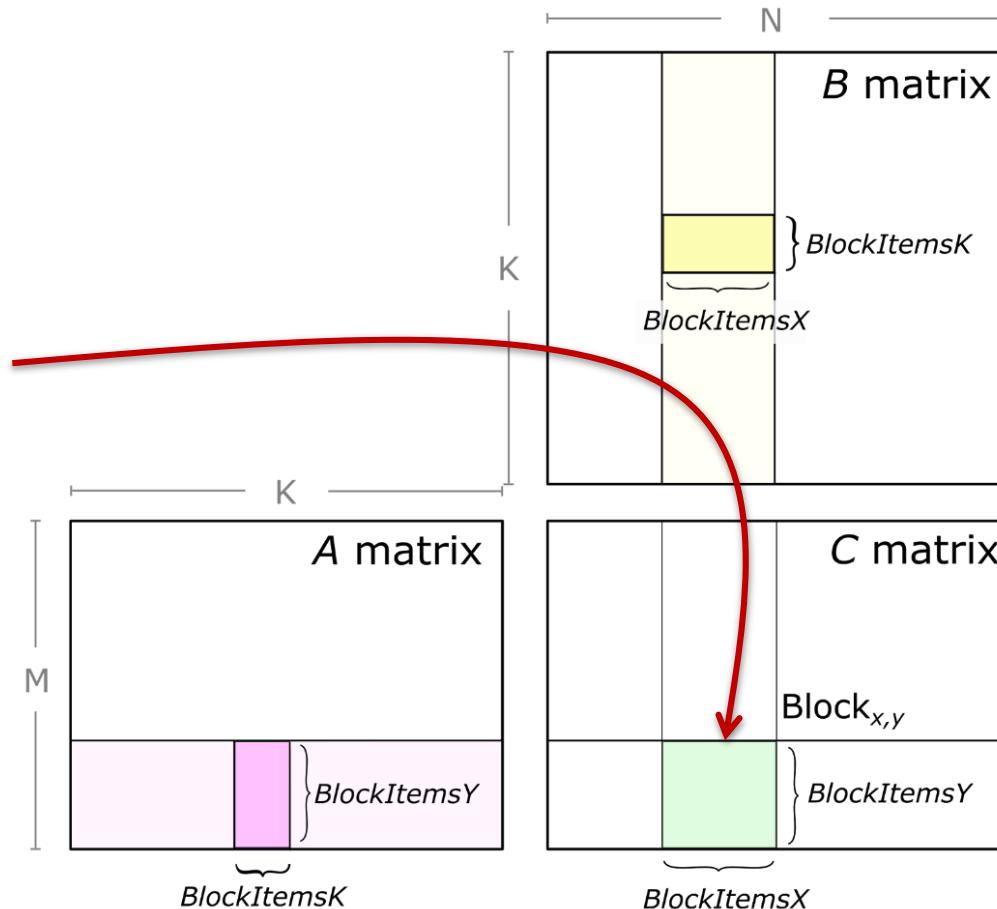
1.170MHz

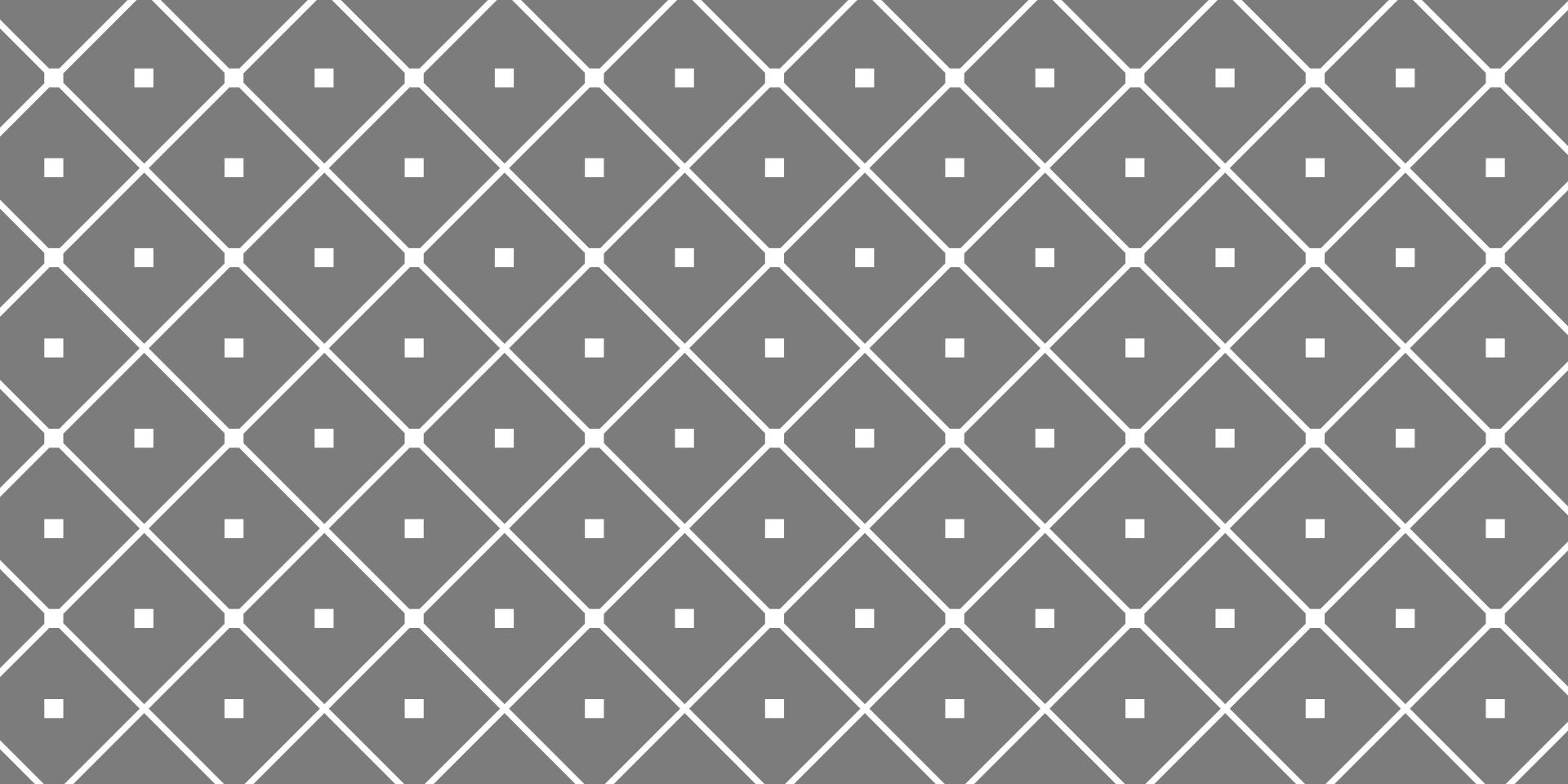
200 Watts



PROCESSAMENTO COM GPU

Cada “unidade vetorial”
calcula 1 bloco/elemento





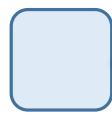
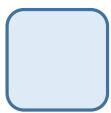
INSTRUÇÕES VETORIAIS (SIMD)

EXECUÇÃO ESCALAR (TRADICIONAL)



```
<main>:  
mov    $0x1024,%rax  
mov    $0x2048,%rbx  
add    %rax, %rbx, $0x3096  
mov    $0x1028,%rax  
mov    $0x2052,%rbx  
add    %rax, %rbx, $0x3100  
mov    $0x1032,%rax  
mov    $0x2056,%rbx  
add    %rax, %rbx, $0x3104  
mov    $0x1036,%rax  
mov    $0x2060,%rbx  
add    %rax, %rbx, $0x3108
```

EXECUÇÃO ESCALAR (TRADICIONAL)



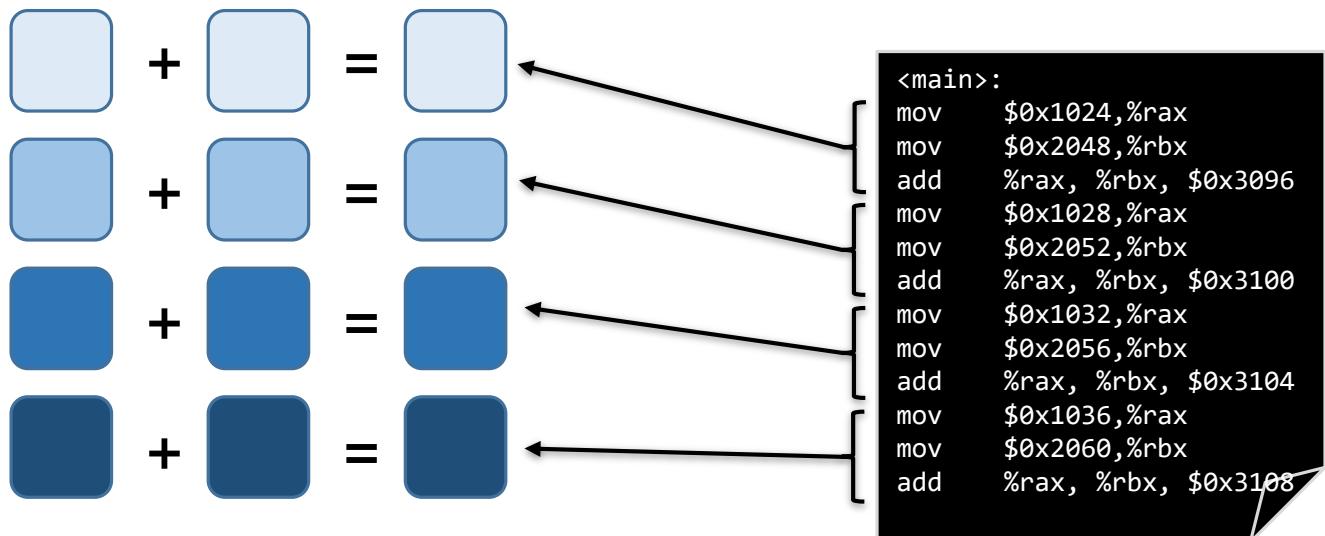
```
<main>:  
mov    $0x1024,%rax  
mov    $0x2048,%rbx  
add    %rax, %rbx, $0x3096  
mov    $0x1028,%rax  
mov    $0x2052,%rbx  
add    %rax, %rbx, $0x3100  
mov    $0x1032,%rax  
mov    $0x2056,%rbx  
add    %rax, %rbx, $0x3104  
mov    $0x1036,%rax  
mov    $0x2060,%rbx  
add    %rax, %rbx, $0x3108
```

EXECUÇÃO ESCALAR (TRADICIONAL)

$$\boxed{} + \boxed{} = \boxed{}$$

```
<main>:  
mov    $0x1024,%rax  
mov    $0x2048,%rbx  
add    %rax, %rbx, $0x3096  
mov    $0x1028,%rax  
mov    $0x2052,%rbx  
add    %rax, %rbx, $0x3100  
mov    $0x1032,%rax  
mov    $0x2056,%rbx  
add    %rax, %rbx, $0x3104  
mov    $0x1036,%rax  
mov    $0x2060,%rbx  
add    %rax, %rbx, $0x3108
```

EXECUÇÃO ESCALAR



EXECUÇÃO VETORIAL



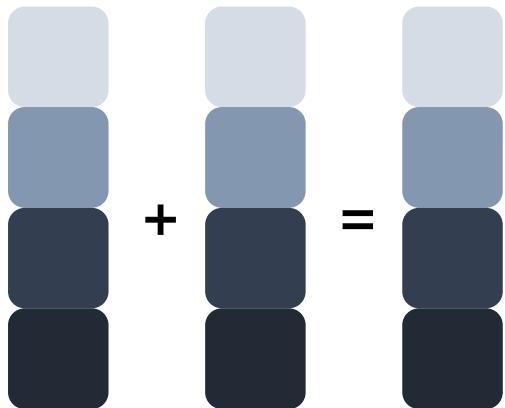
```
<main>:  
mov    $0x1024,%ymm0  
mov    $0x2048,%ymm1  
add    %ymm0,  
%ymm1,$0x3096
```

EXECUÇÃO VETORIAL



```
<main>:  
mov    $0x1024,%ymm0  
mov    $0x2048,%ymm1  
add    %ymm0,  
%ymm1,$0x3096
```

EXECUÇÃO VETORIAL



```
<main>:  
mov    $0x1024,%ymm0  
mov    $0x2048,%ymm1  
add    %ymm0,  
%ymm1,$0x3096
```

INTEL SIMD EVOLUTION

64 bit SIMD

MMX P3 1997

- (57 inst)

INTEL SIMD EVOLUTION

64 bit SIMD

MMX P3 1997

•(57 inst)

128 bit SIMD

SSE P3 1999

•(70inst)

SSE2 P4 2000

•(144inst)

SSE3 P4 2004

•(13inst)

SSE 3 Core 2006

•(32inst)

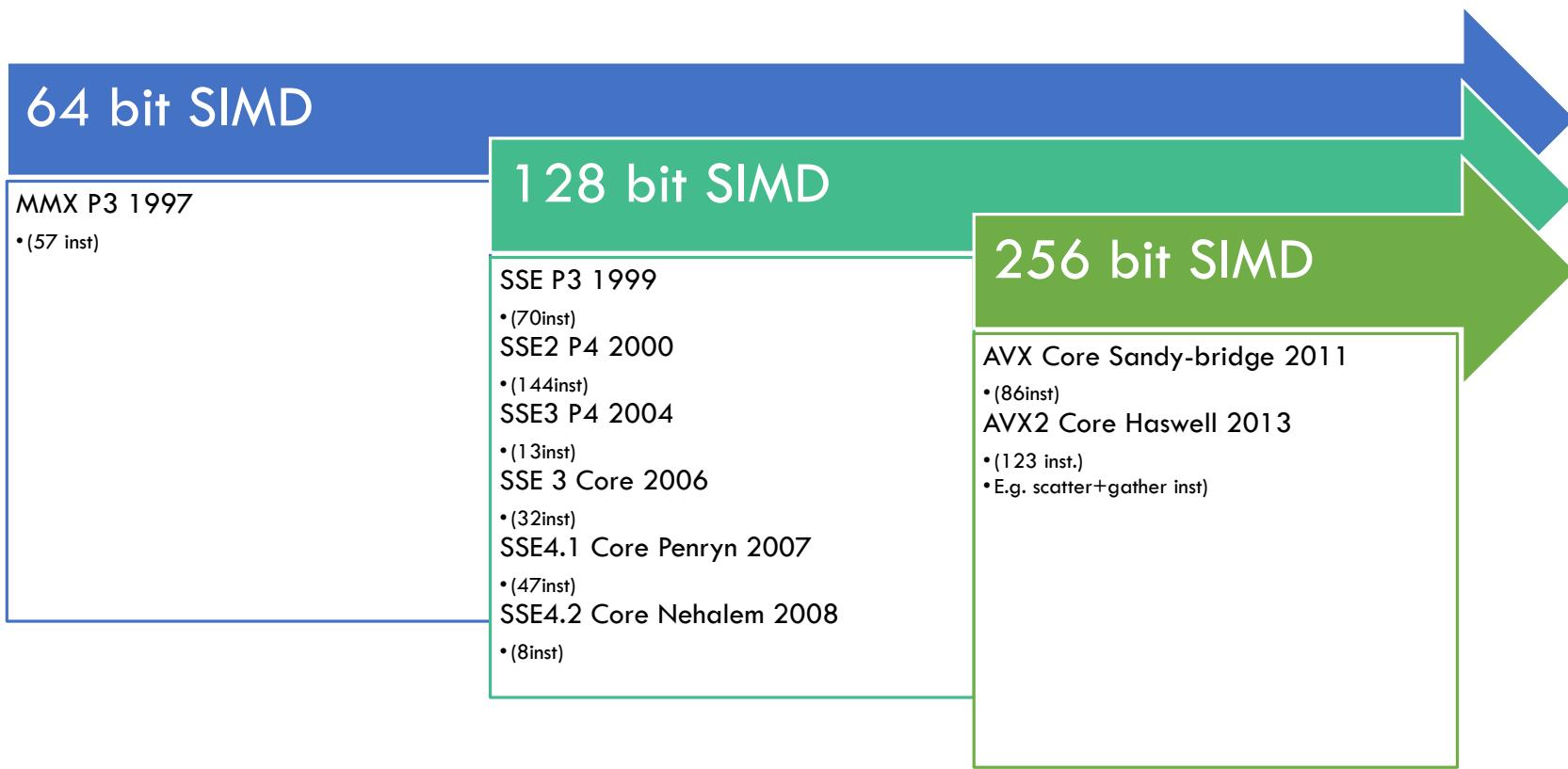
SSE4.1 Core Penryn 2007

•(47inst)

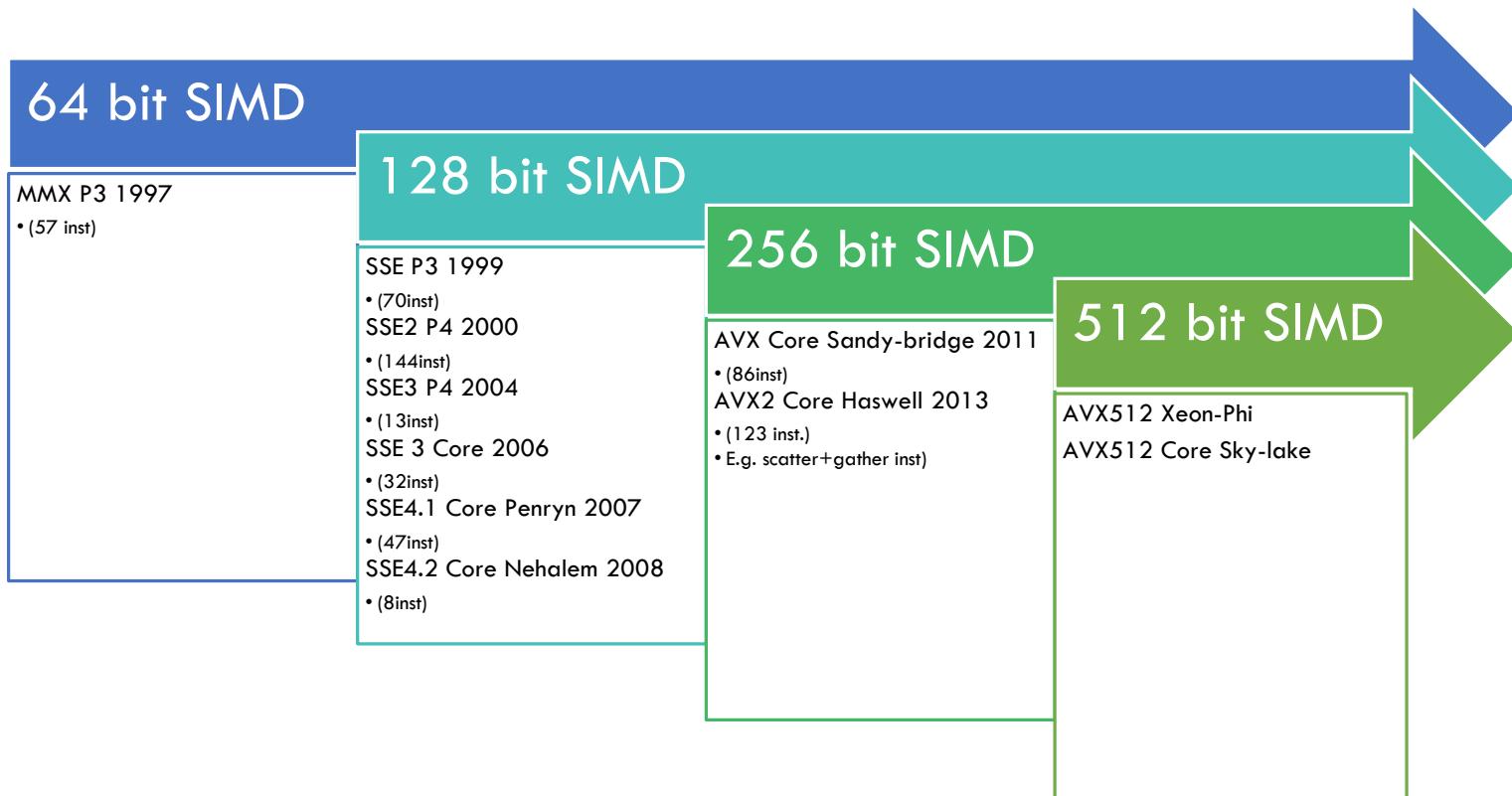
SSE4.2 Core Nehalem 2008

•(8inst)

INTEL SIMD EVOLUTION



INTEL SIMD EVOLUTION



VETORIZIZE SEU CÓDIGO

Multiplicação de matrizes ingênuas: **1264s**
(1000x1000 doubles)

```
def dot_naive(a,b): # 1.5 MFlops
    c = np.zeros((nrows, ncols), dtype='f8')
    for row in xrange(nrows):
        for col in xrange(ncols):
            for i in xrange(nrows):
                [row, col] += a[row,i] * b[i,col]
    return c
```

VETORIZIZE SEU CÓDIGO

Multiplicação de matrizes vetorizada: **20s**

(64x mais

```
def dot(a,b): # 100 Mflops
    rapido
    c = np.empty((nrows, ncols), dtype='f8')
    for row in xrange(nrows):
        for col in xrange(ncols):
            for i in xrange(nrows):
                [row, col] = np.sum(a[row] * b[:,col])
    return c
```

ALTERNATIVAS PARA ALCANÇAR MAIOR PARALELISMO

Juntar PCs

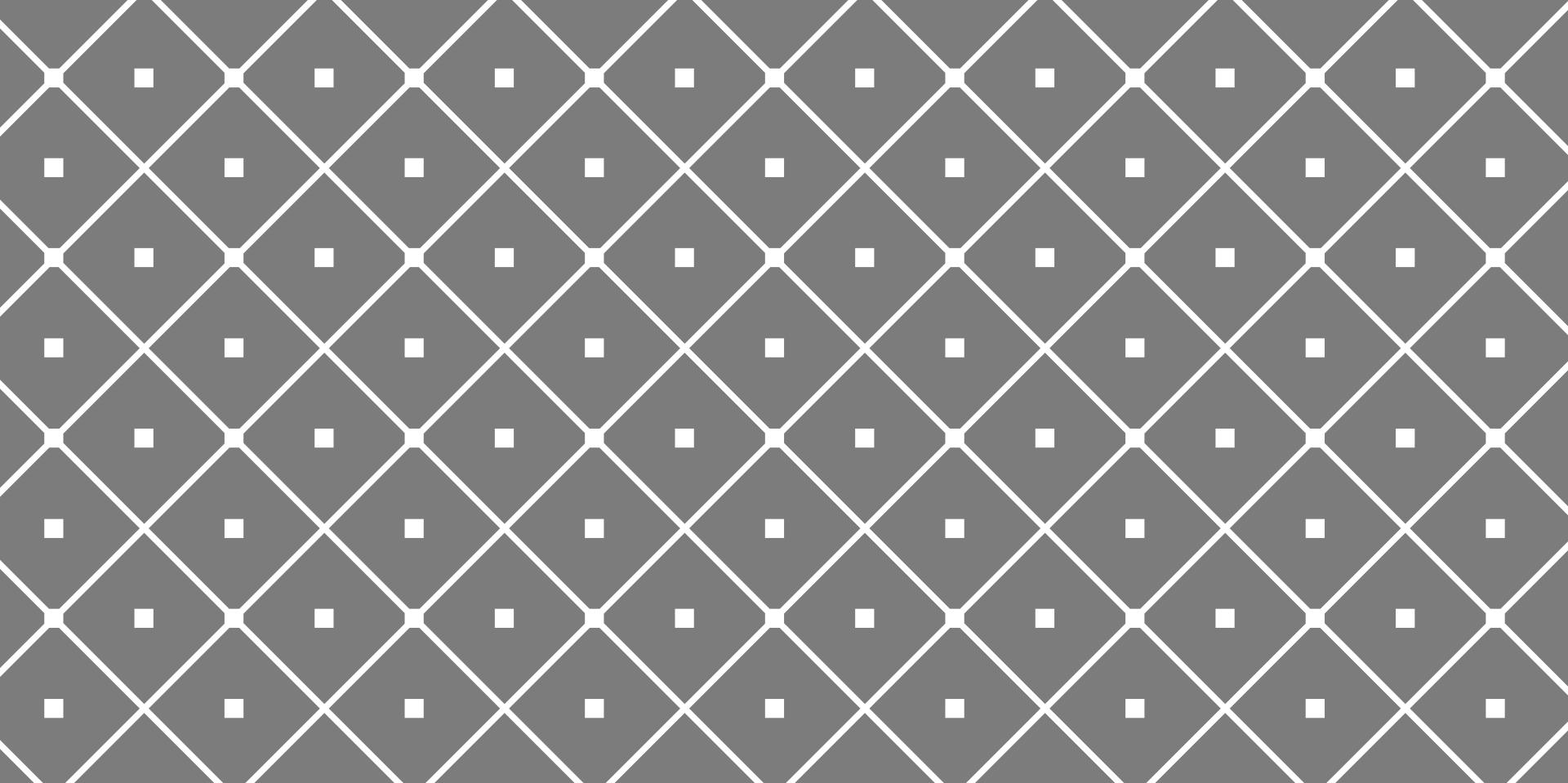
- Cluster
- Grid
- Cloud

Vetorização

- Instruções
- GPUs

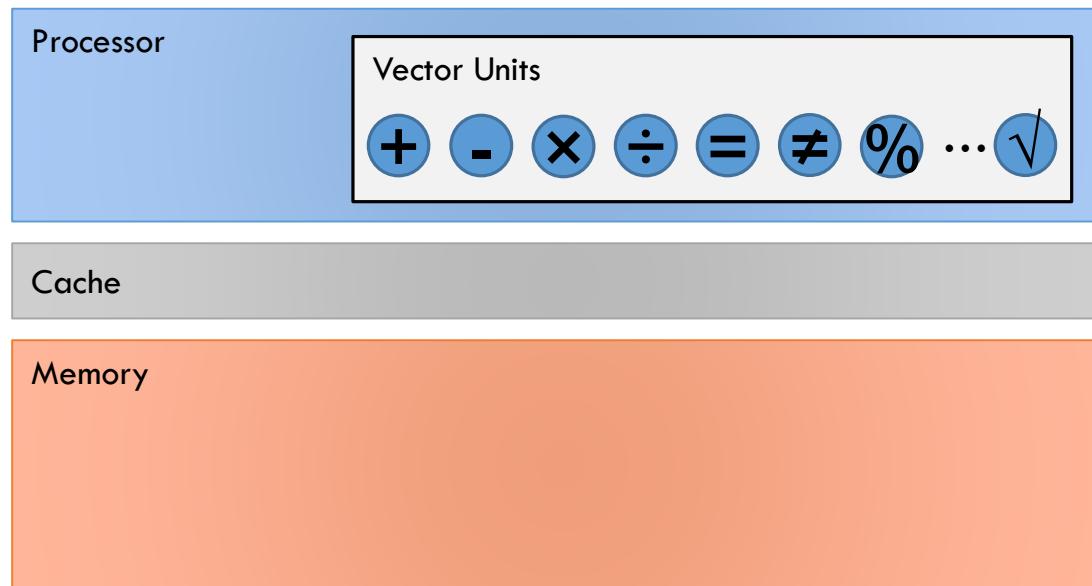
???

- Processamento na memória
- ???

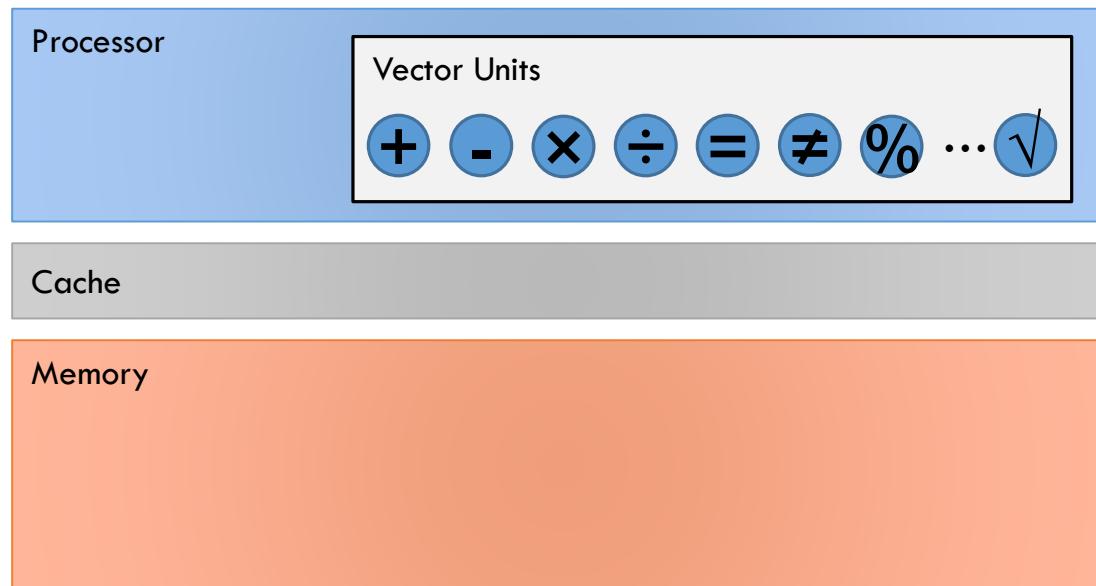


E SE PUDÉSSEMOS PROCESSAR
PRÓXIMO AOS DADOS?

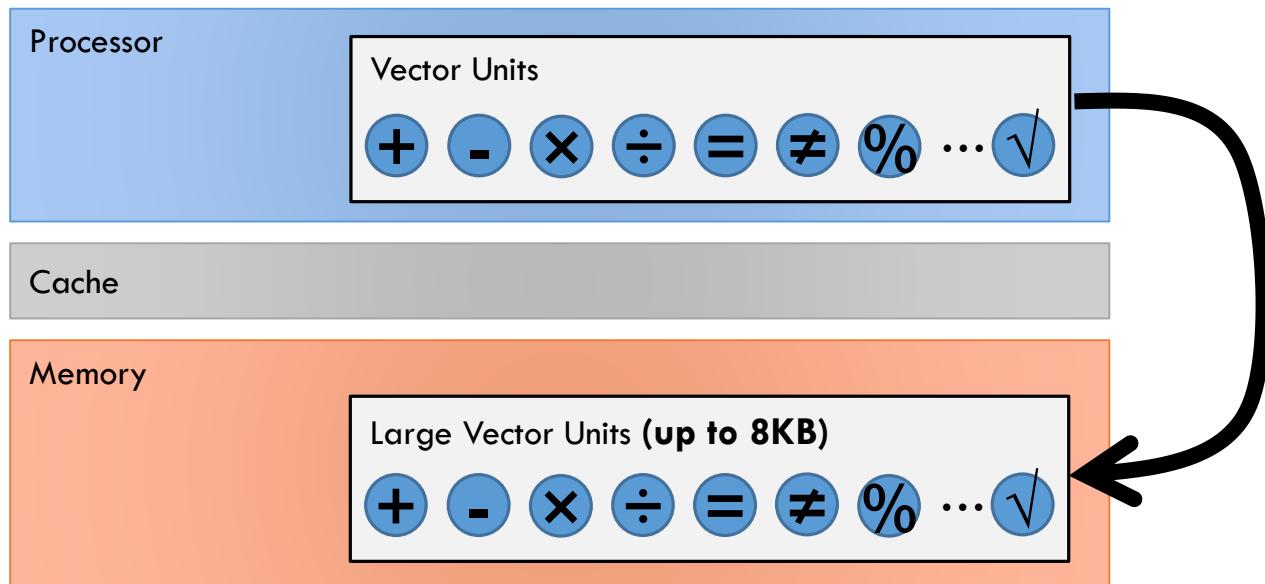
NOSSA OBSERVAÇÃO:



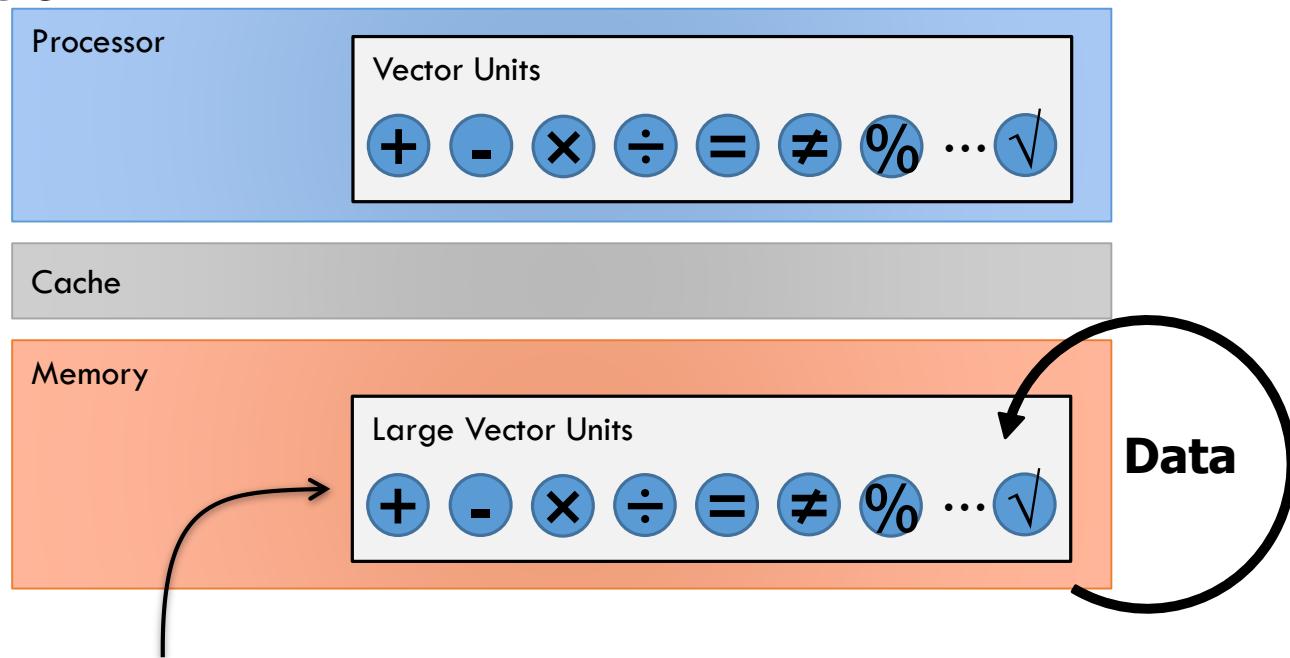
NOSSA PROPOSTA:



IMPLEMENTAR LÓGICA DE PROCESSAMENTO DENTRO DA MEMÓRIA



ASSIM, ECONOMIZANDO MOVIMENTOS DE DADOS PARA APLICAÇÕES INTENSIVAS EM DADOS



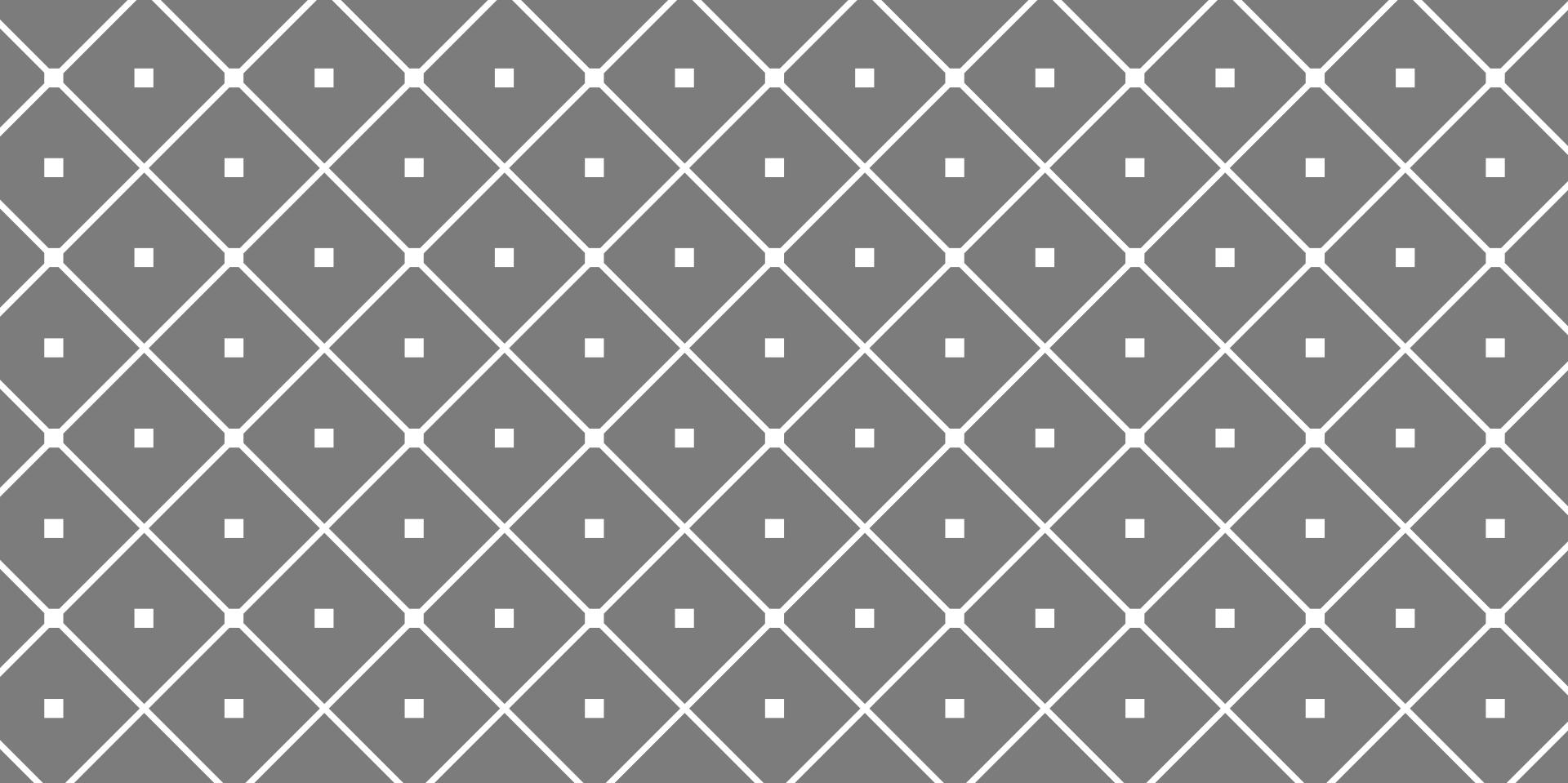
PIM – Processing in memory
(a.k.a. Smart memories)

GRANDES DESAFIOS

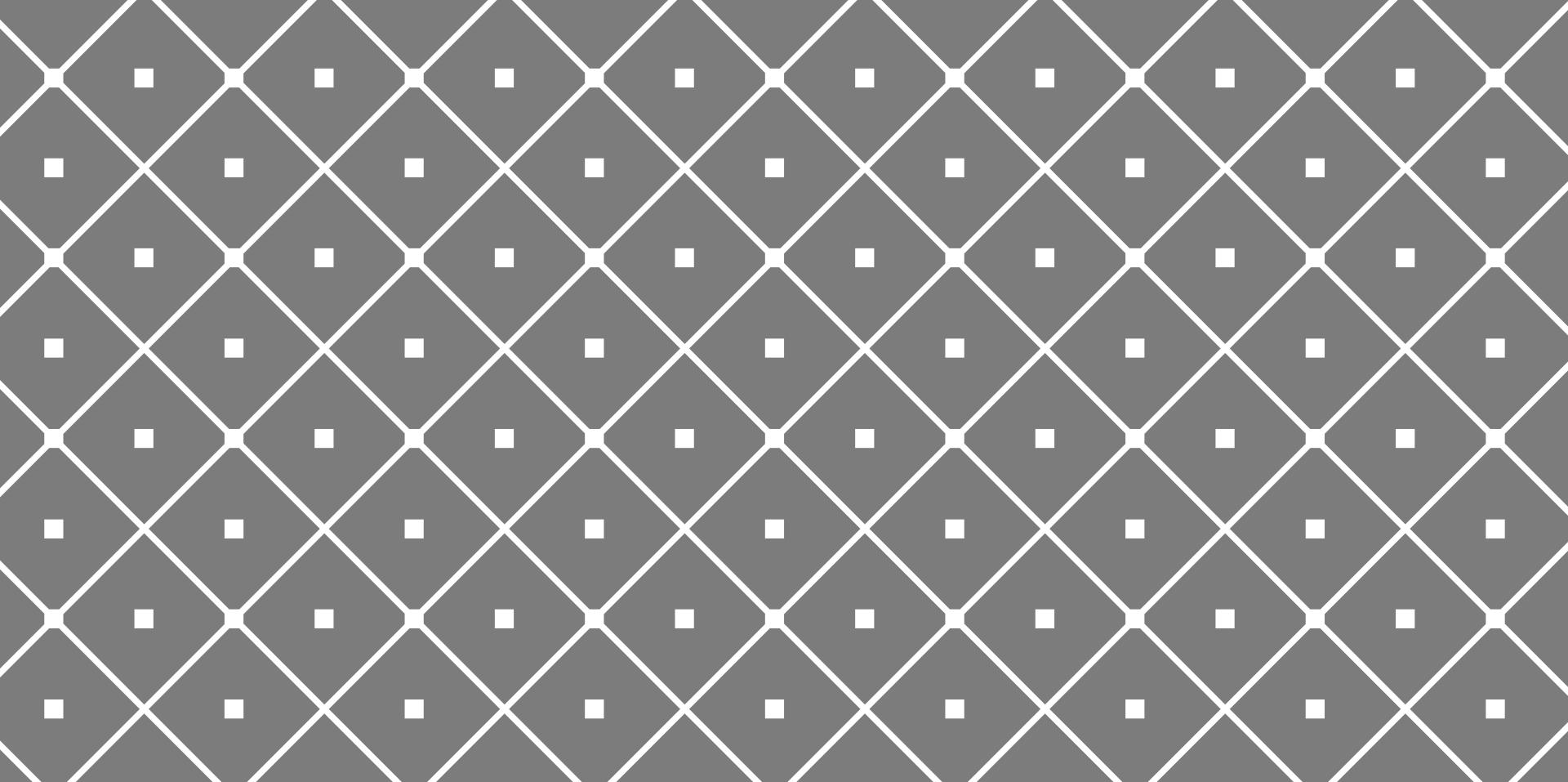
Novo modelo, diferente dos últimos 70 anos
(ou seja, estende a arquitetura von Neumann).

Requer novos designs de hardware e software

A DRAM está se tornando mais do que uma commodity (ou seja, menos dependente de custo)



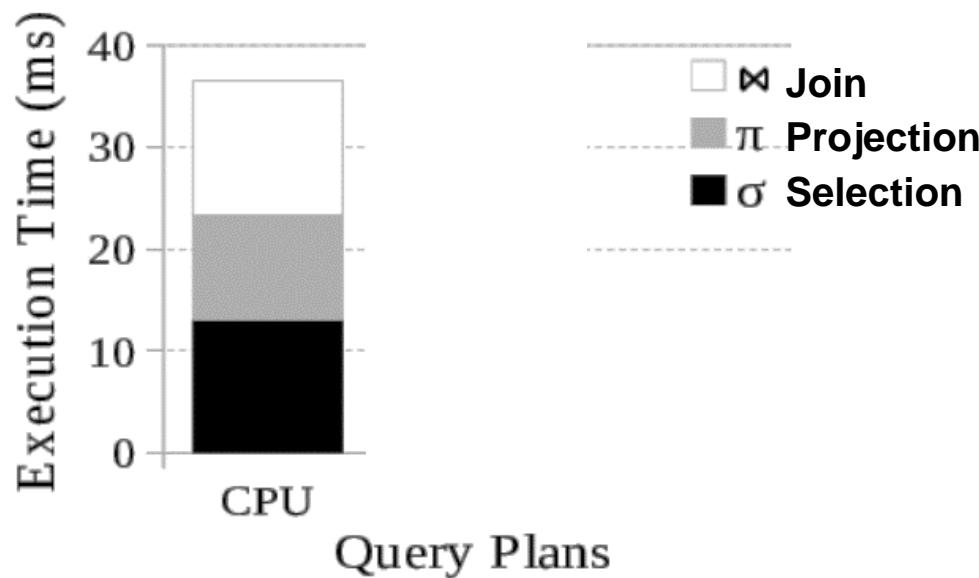
NOSSOS RESULTADOS



EX. |
DATABASE

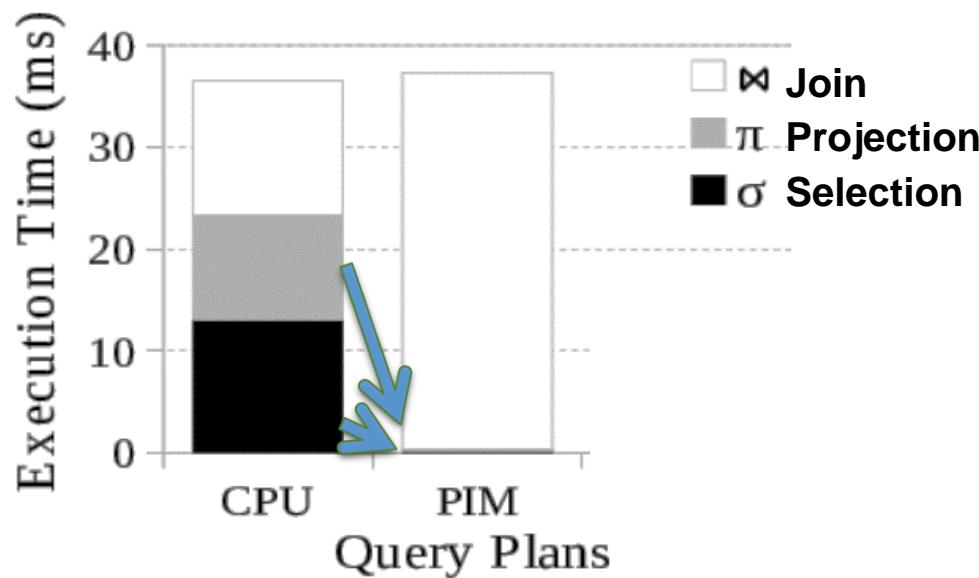
DATABASE MIGRATION TO NDP

We discovered that the most time-consuming operators are Selection, Projection and Join, which accounted for over 90%.



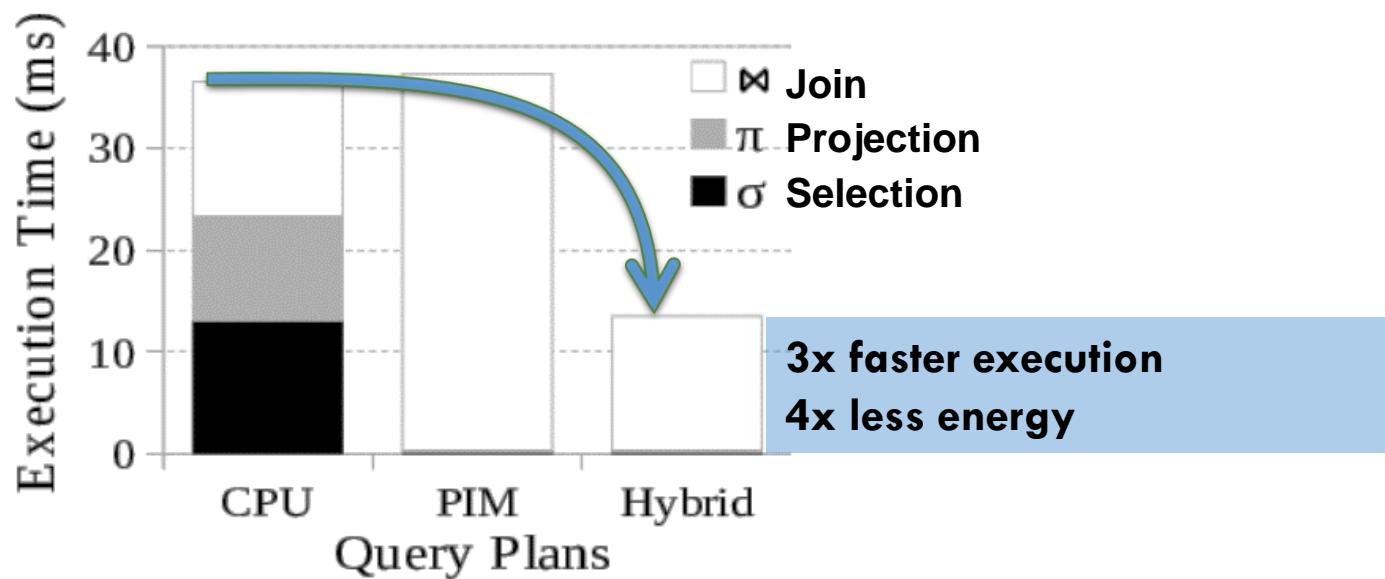
DATABASE MIGRATION TO NDP

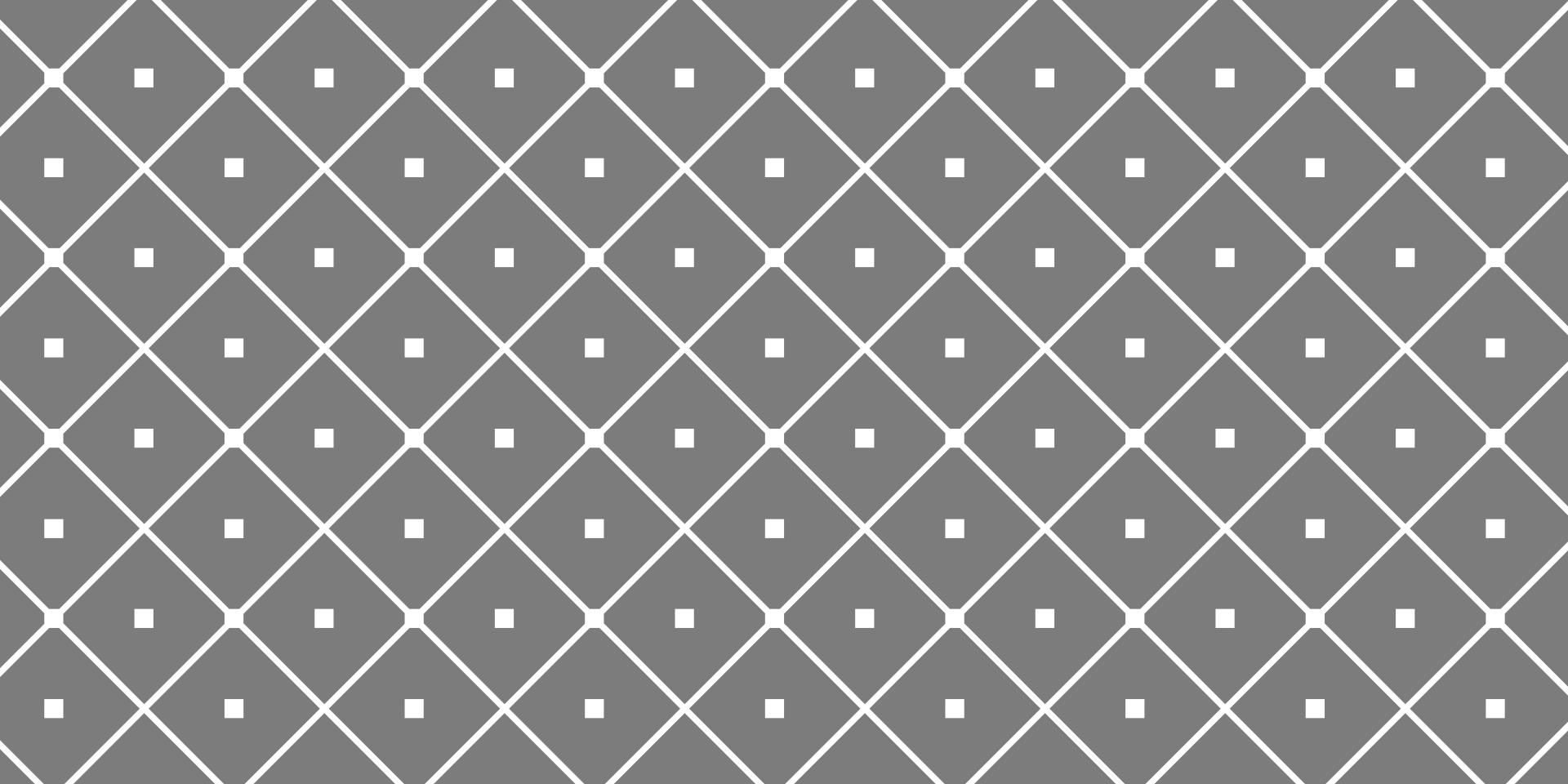
We discovered that the most time-consuming operators are Selection, Projection and Join, which accounted for over 90%.



DATABASE MIGRATION TO NDP

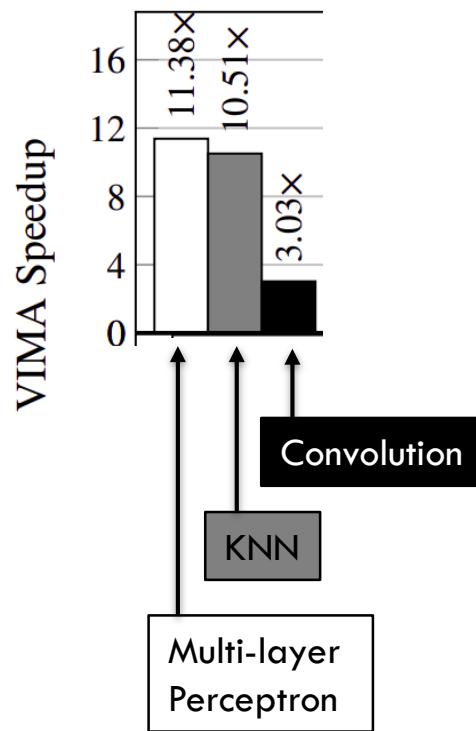
We discovered that the most time-consuming operators are Selection, Projection and Join, which accounted for over 90%.





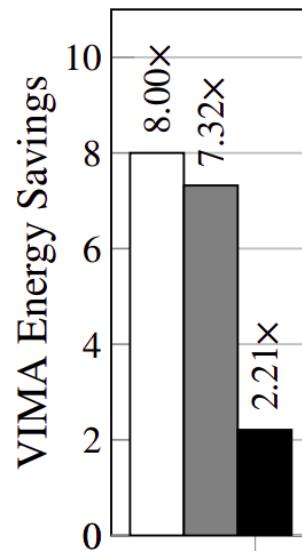
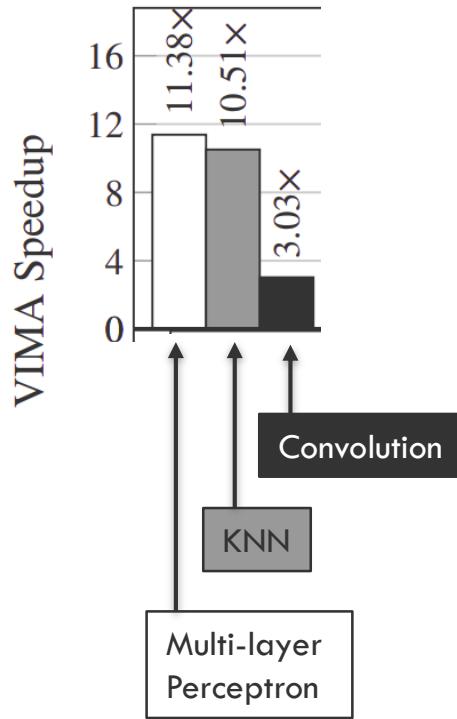
EX.
MACHINE LEARNING
~~TRAIN~~ + INFERENCE

MACHINE LEARNING INFERENCE

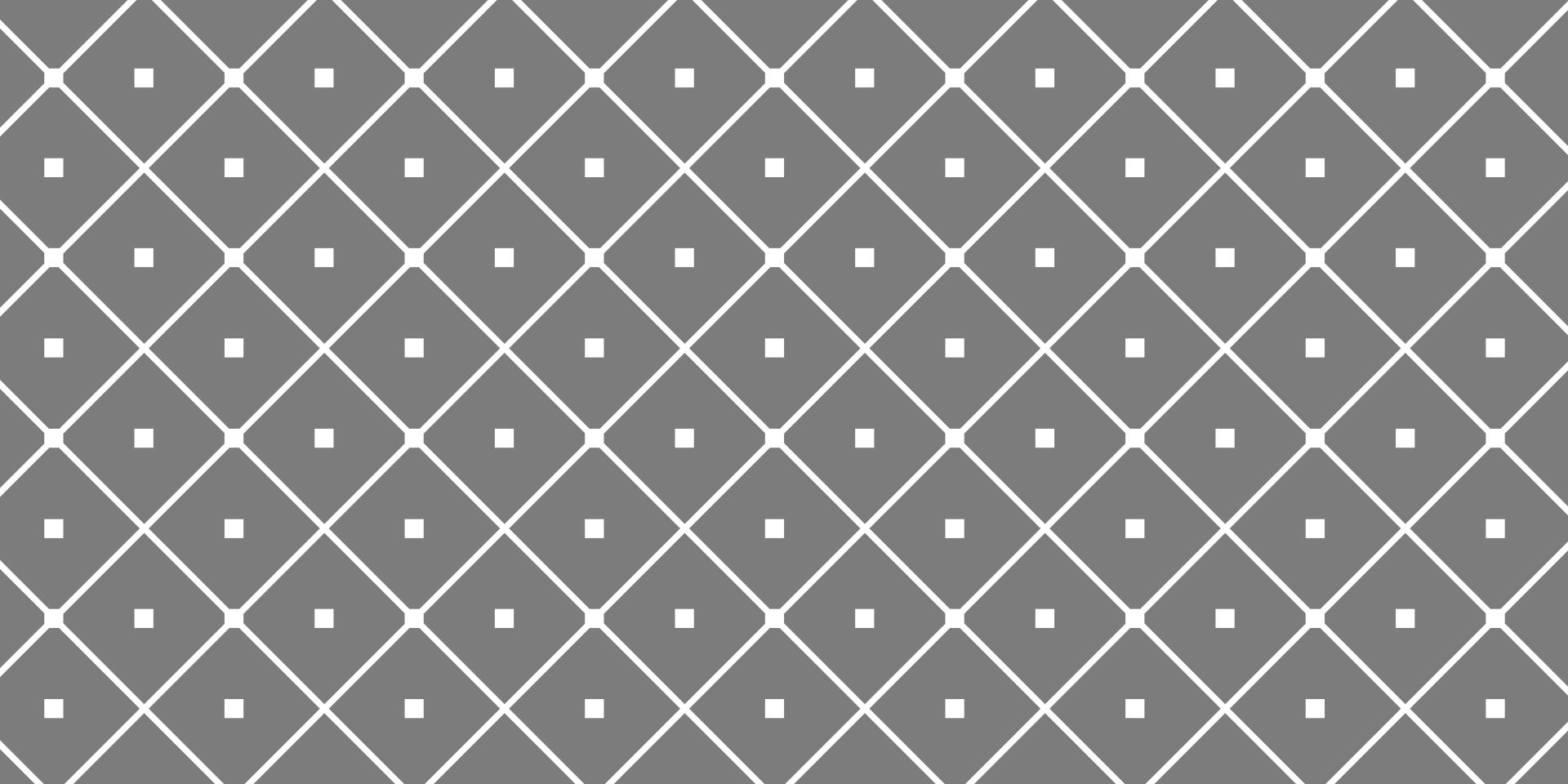


Cordeiro, Aline S., et al. Efficient Machine Learning execution with Near-Data Processing. **MICPRO**, 2022.

MACHINE LEARNING INFERENCE



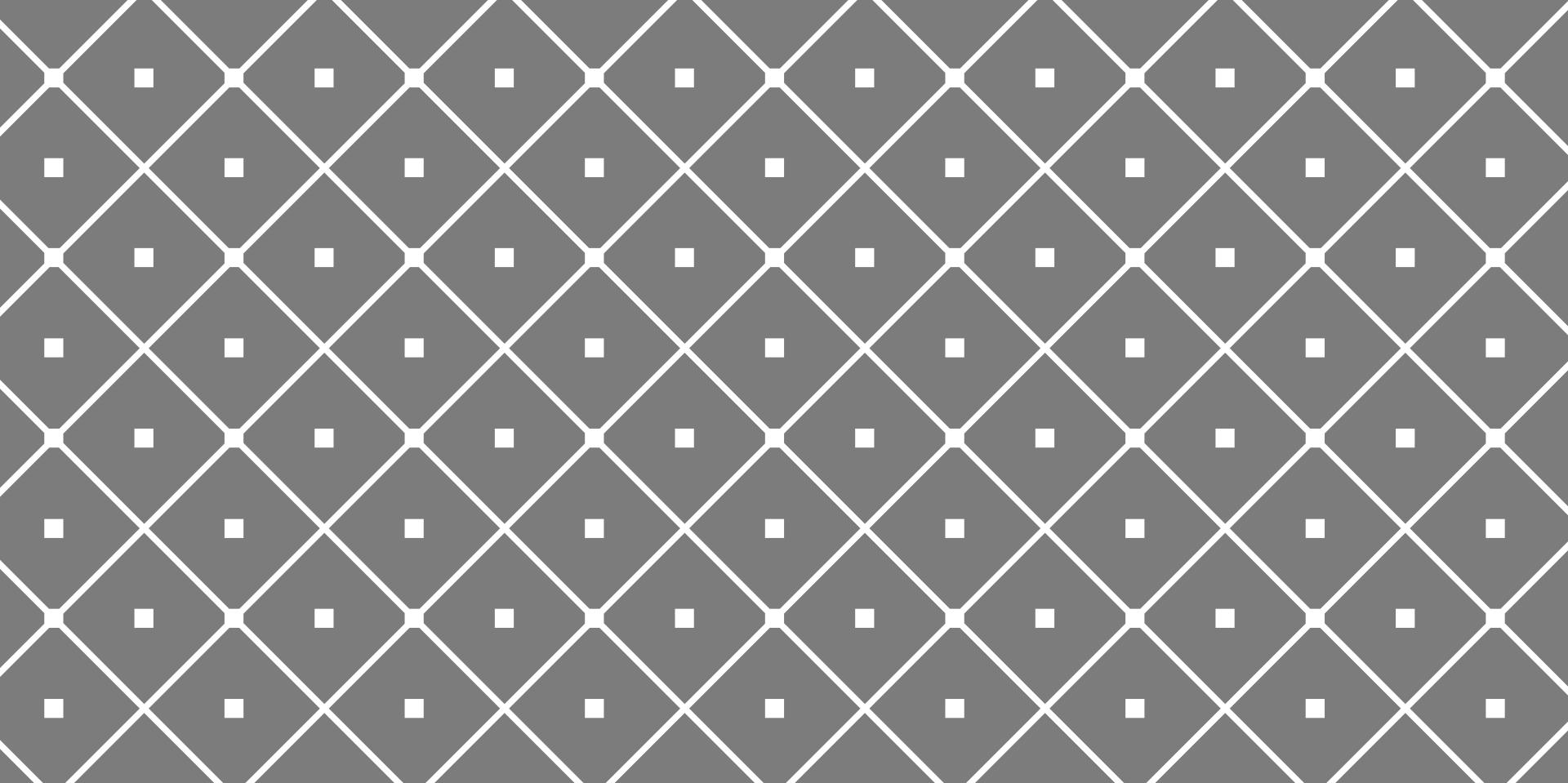
Cordeiro, Aline S., et al. Efficient Machine Learning execution with Near-Data Processing. **MICPRO**, 2022.



COMO USAR OS COMPUTADORES PARALELOS?

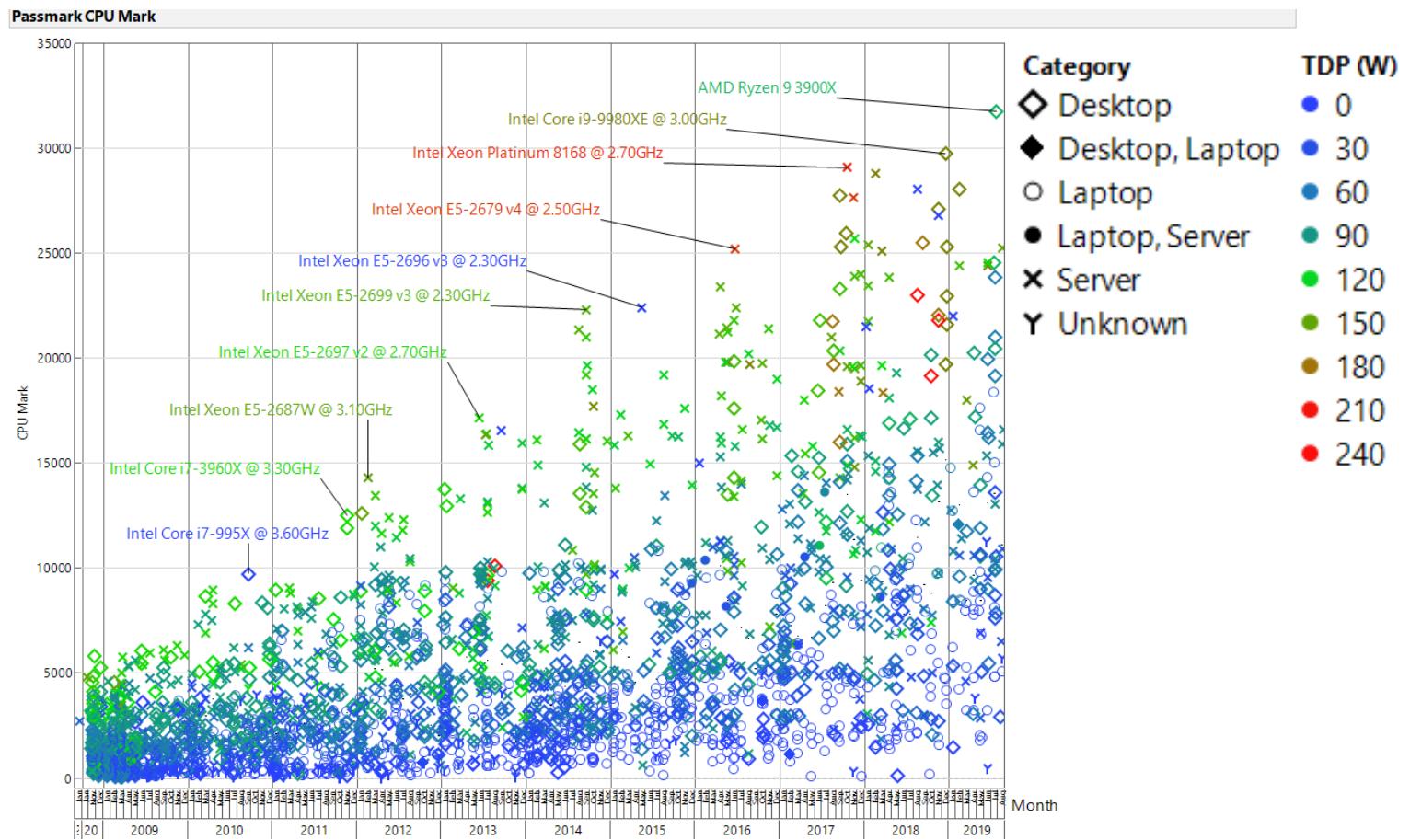
O QUE TEMOS DISPONÍVEL

Juntar PCs	Vetorização	PIM
<ul style="list-style-type: none">• Cluster• Grid• Cloud	<ul style="list-style-type: none">• Instruções• GPUs	<ul style="list-style-type: none">• UPMEM• SAMSUNG

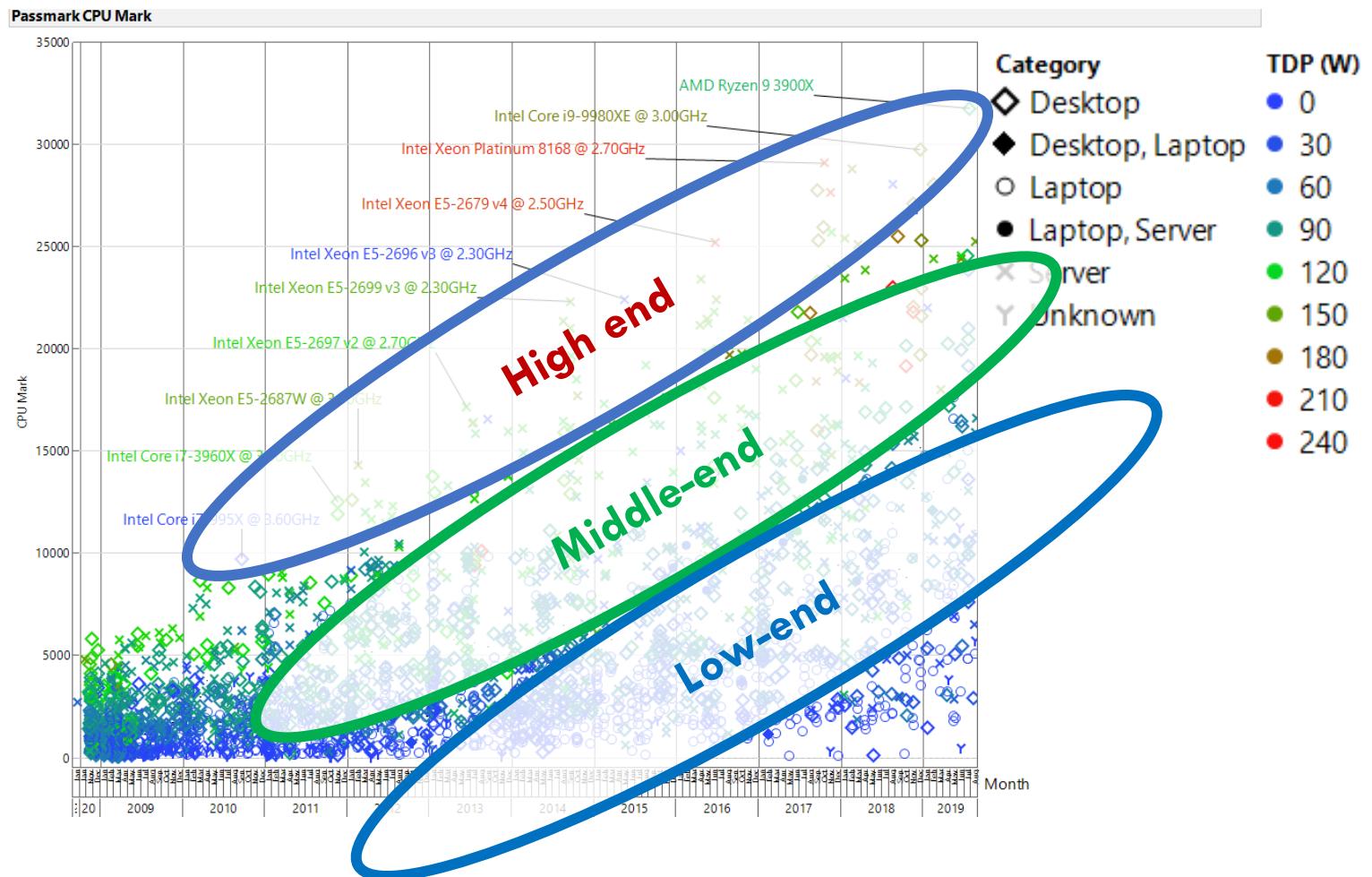


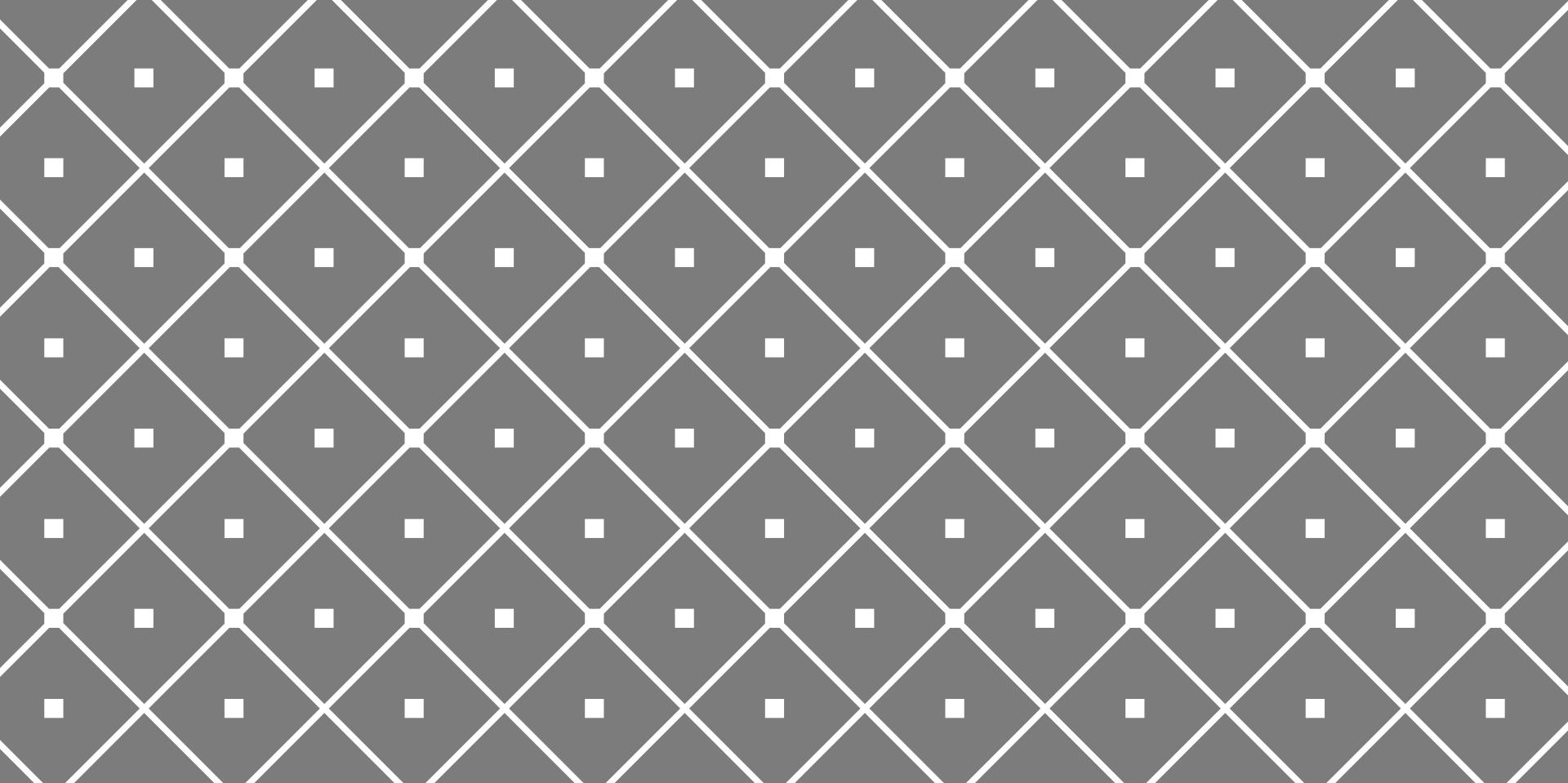
HIGH PERFORMANCE COMPUTING
=
GRANDES SERVIDORES?

ESPECTRO DOS PROCESSADORES



ESPECTRO DOS PROCESSADORES





O QUE ESPERAR DOS PROGRAMAS PARALELOS

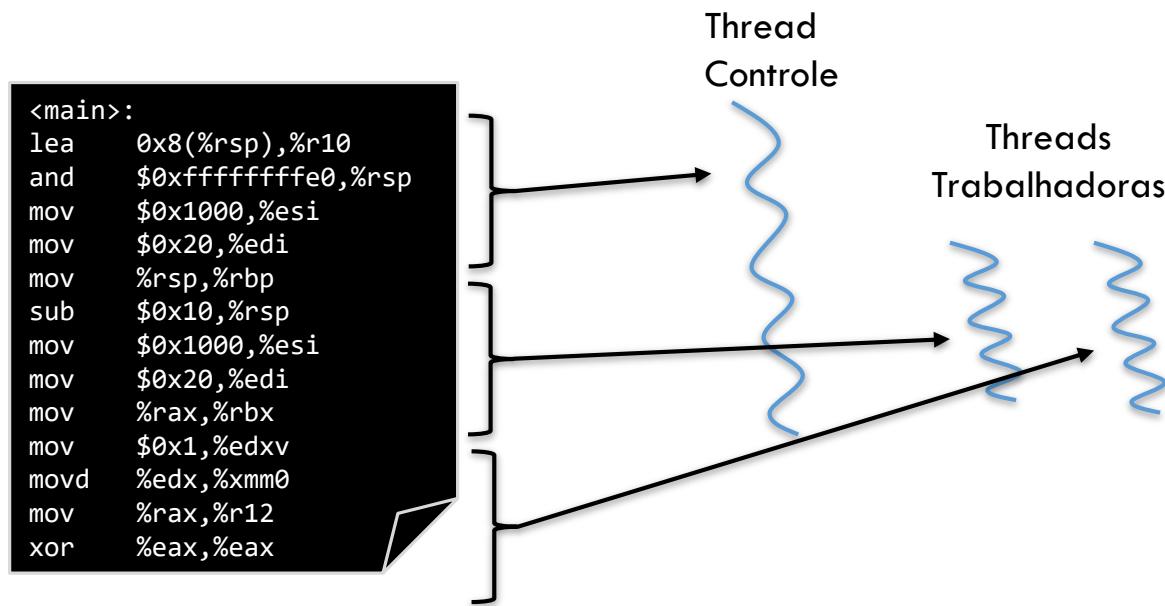
PROGRAMAS SEQUENCIAIS

```
<main>:  
    lea    0x8(%rsp),%r10  
    and   $0xfffffffffe0,%rsp  
    mov    $0x1000,%esi  
    mov    $0x20,%edi  
    mov    %rsp,%rbp  
    sub    $0x10,%rsp  
    mov    $0x1000,%esi  
    mov    $0x20,%edi  
    mov    %rax,%rbx  
    mov    $0x1,%edxv  
    movd   %edx,%xmm0  
    mov    %rax,%r12  
    xor    %eax,%eax
```

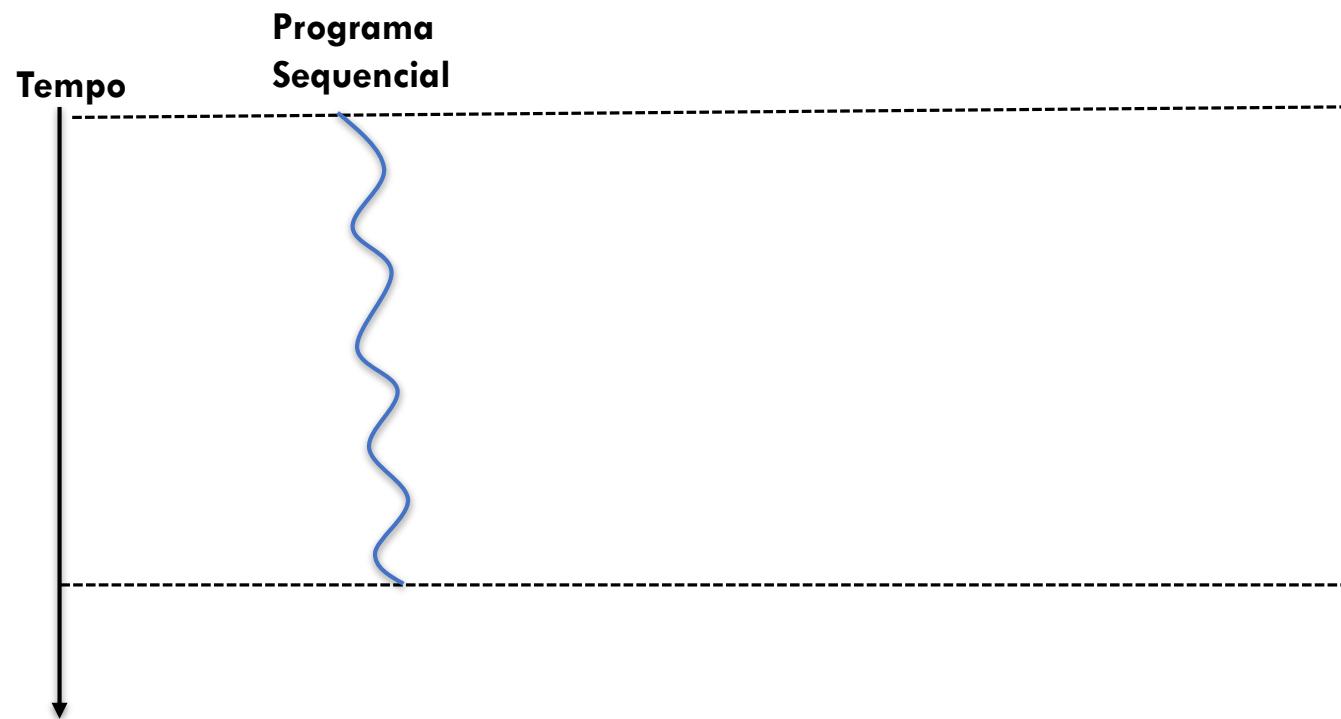
Thread sequencial



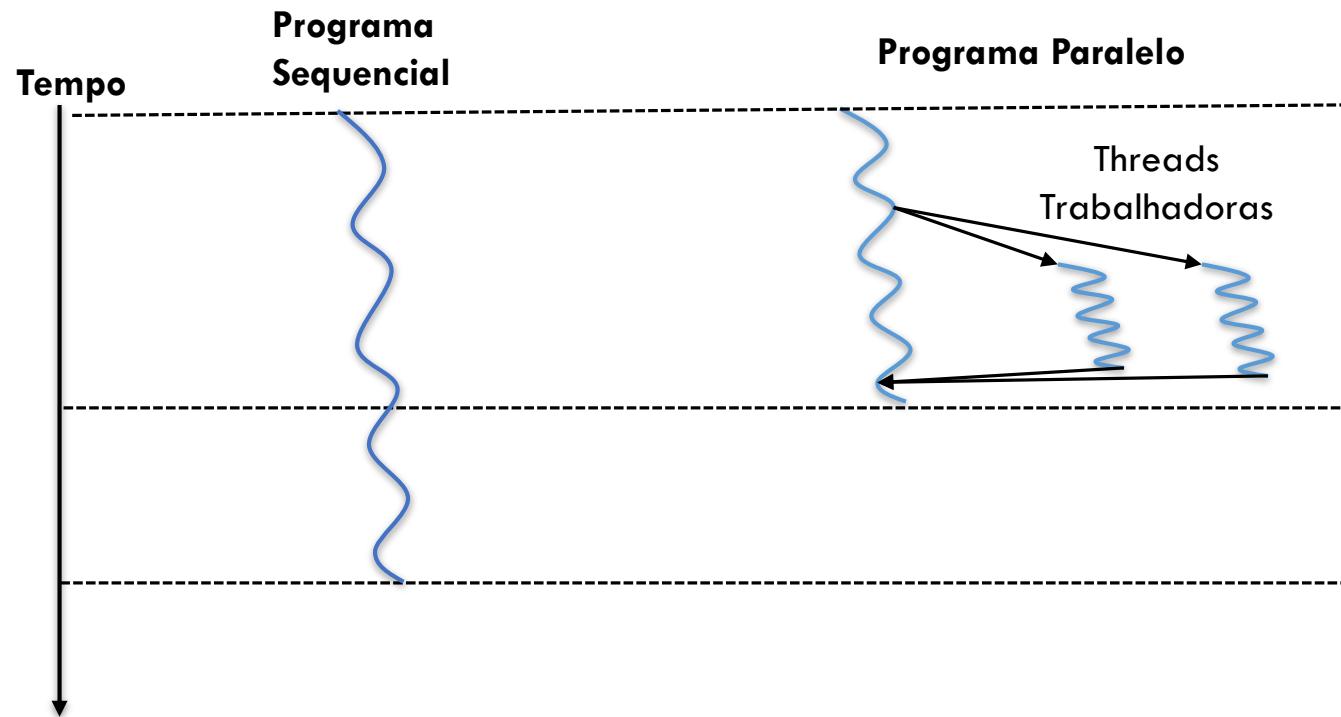
PROGRAMAS PARALELOS



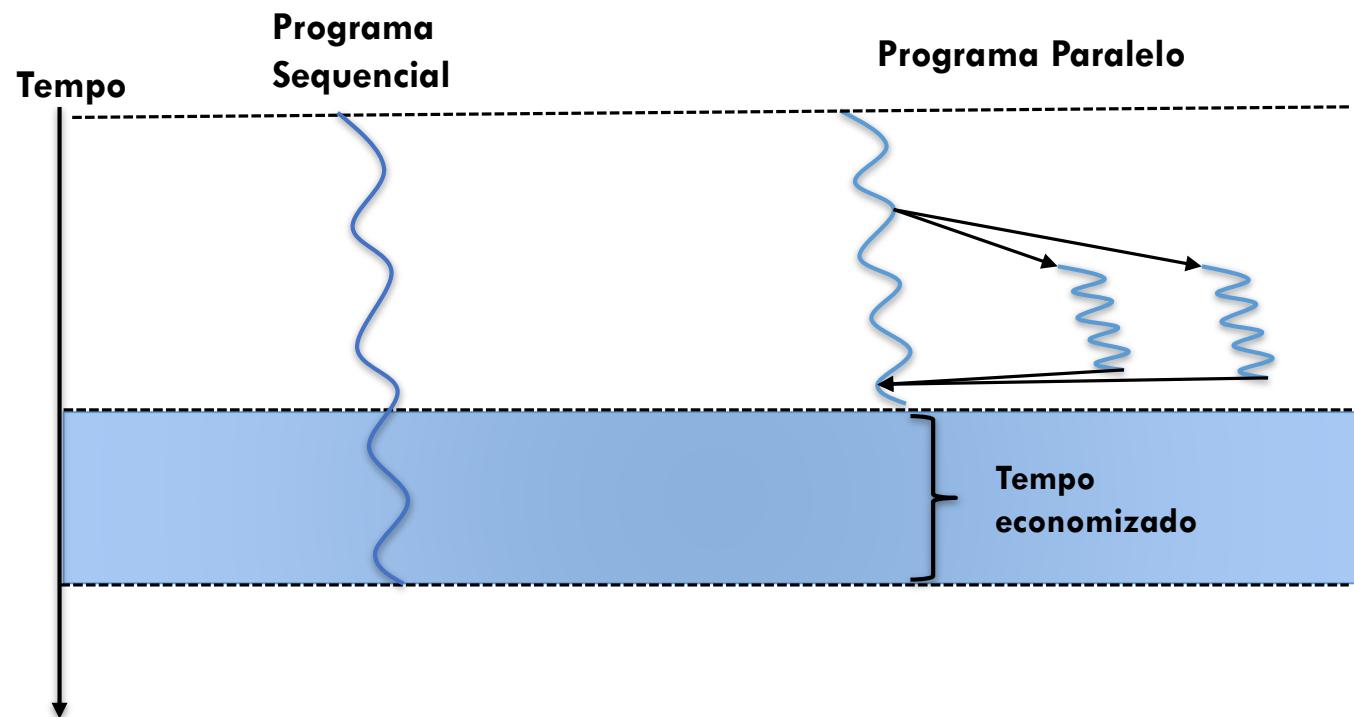
DESEMPENHO DE PROGRAMAS PARALELOS



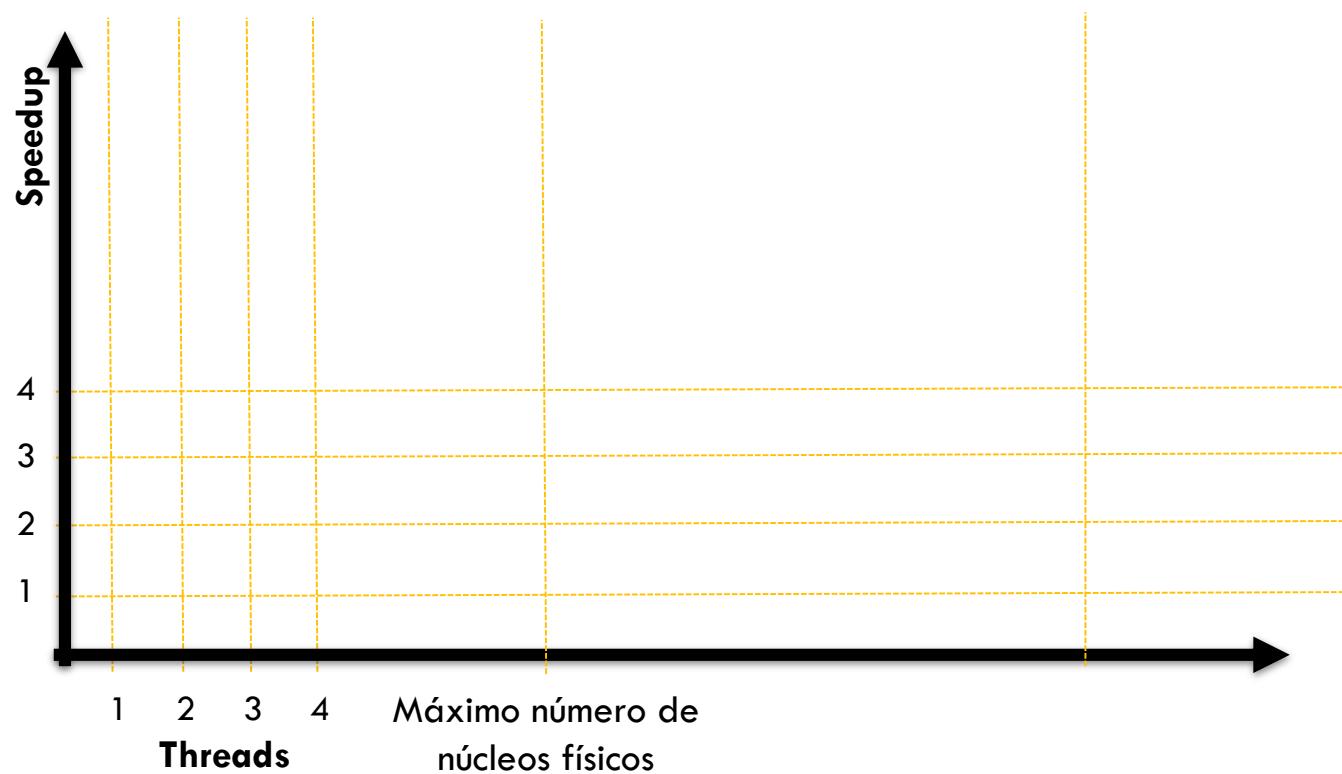
DESEMPENHO DE PROGRAMAS PARALELOS



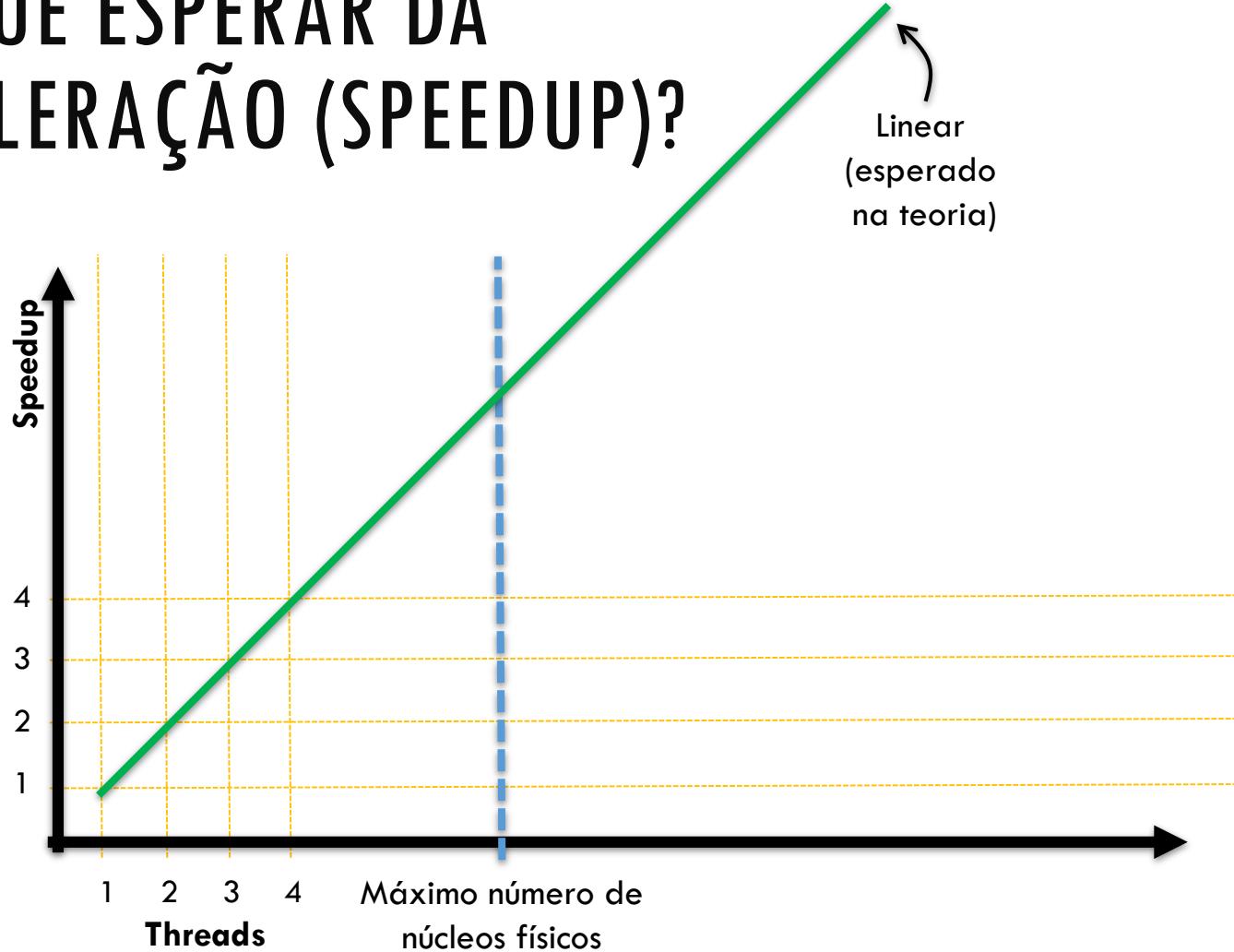
DESEMPENHO DE PROGRAMAS PARALELOS



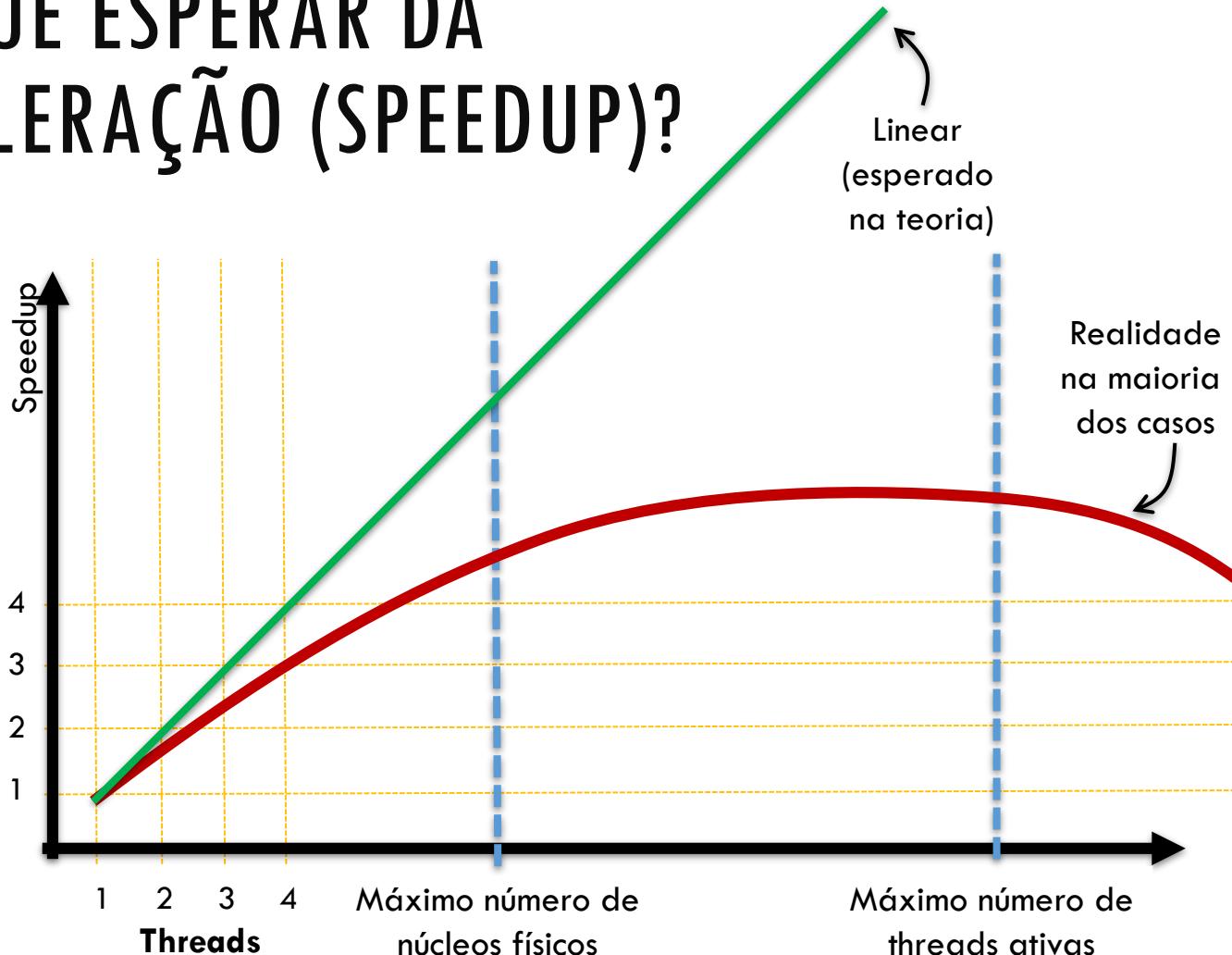
O QUE ESPERAR DA ACELERAÇÃO (SPEEDUP)?



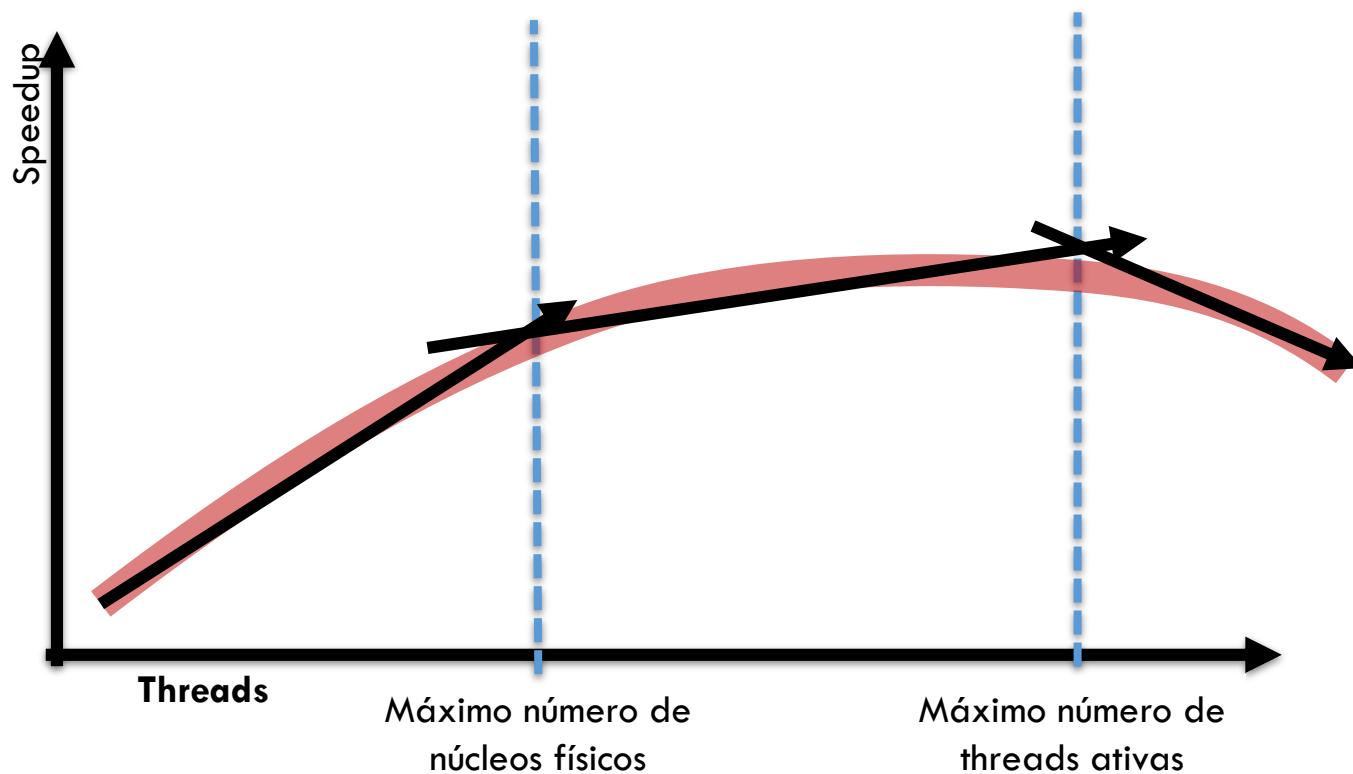
O QUE ESPERAR DA ACELERAÇÃO (SPEEDUP)?

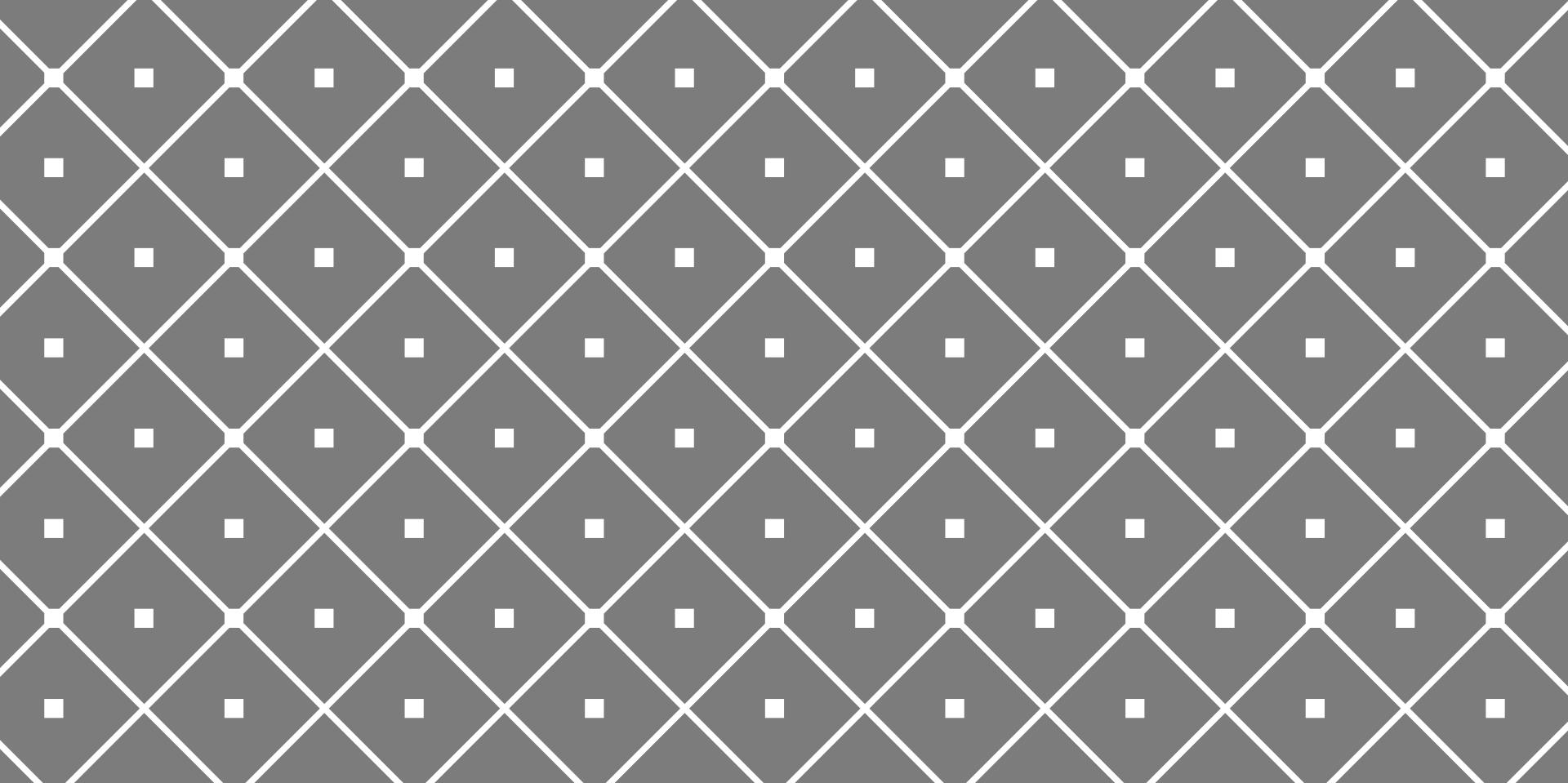


O QUE ESPERAR DA ACELERAÇÃO (SPEEDUP)?



O QUE ESPERAR DA ACELERAÇÃO (SPEEDUP)?





CONSUMO DE ENERGIA E POTÊNCIA

DECLARAÇÕES QUE SÃO VERDADEIRAS DESDE UMA DÉCADA OU MAIS

“Power is considered as the most important constraint in embedded systems.” [in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

“Power demands are increasing rapidly, yet battery capacity cannot keep up.” [in Ditzel et al.: Power-Aware Architecting for data-dominated applications, 2007, Springer]

PRINCIPAIS RAZÕES

O fornecimento de energia é caro

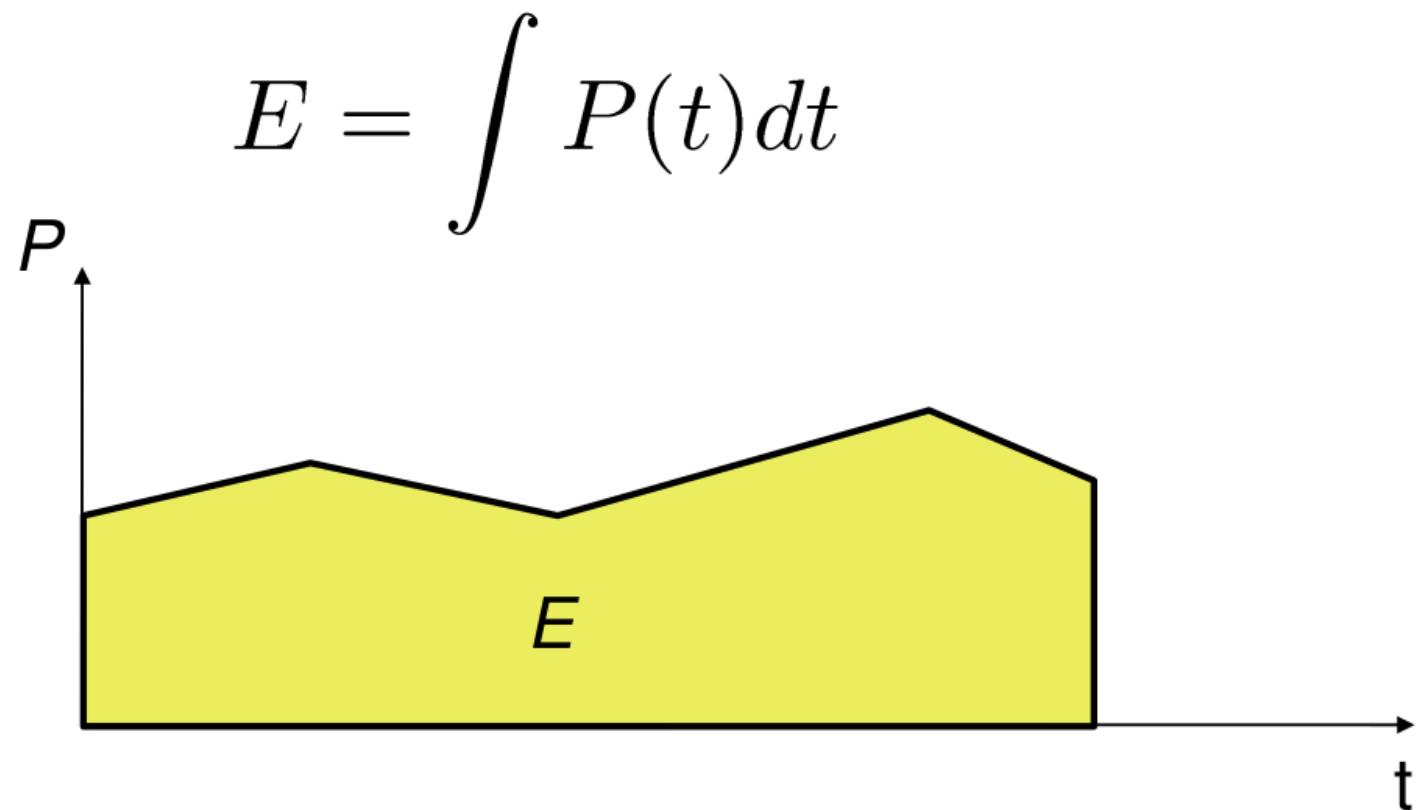
A capacidade da bateria está crescendo lentamente

Os dispositivos podem superaquecer

A colheita de energia (por exemplo, de células solares) é limitada
devido à densidade de energia disponível relativamente baixa

É preciso otimizar HW e SW!!!

ENERGIA V. POTÊNCIA



LOW POWER VS. LOW ENERGY

Minimizar o consumo de potência (tensão * corrente) é importante para

- O projeto da fonte de alimentação e reguladores de tensão
- O dimensionamento da interligação entre a fonte de alimentação e os componentes
- Resfriamento (resfriamento de curto prazo)
- Alto custo
- Espaço limitado

Minimizar o consumo de energia é importante devido à

- Disponibilidade restrita de energia (sistemas móveis)
- Capacidades limitadas da bateria (apenas melhorando lentamente)
- Custos de energia muito elevados (colheita de energia, painéis solares, manutenção/baterias)
- Longa vida útil, baixas temperaturas

GERENCIAMENTO DE POTÊNCIA

Dynamic Frequency Scaling (DFS): - O clock do processador é reduzido em alguns múltiplos do máximo, permitindo que o processador consuma menos energia às custas de desempenho reduzido

Clock Throttling: - Em contraste com o escalonamento de frequência onde a frequência do clock é realmente modificada, o clock throttling mantém o clock funcionando na frequência original, no entanto, o sinal do clock é desabilitado por algum número de ciclos em intervalos regulares.

Dynamic Voltage Scaling (DVS): - Reduz a energia consumida diminuindo sua tensão de operação.

*A escala de tensão é vantajosa porque a energia consumida por um processador é diretamente proporcional a V^2

CONSUMO DE ENERGIA EM SEMICONDUTORES

$$P = CV^2 F$$

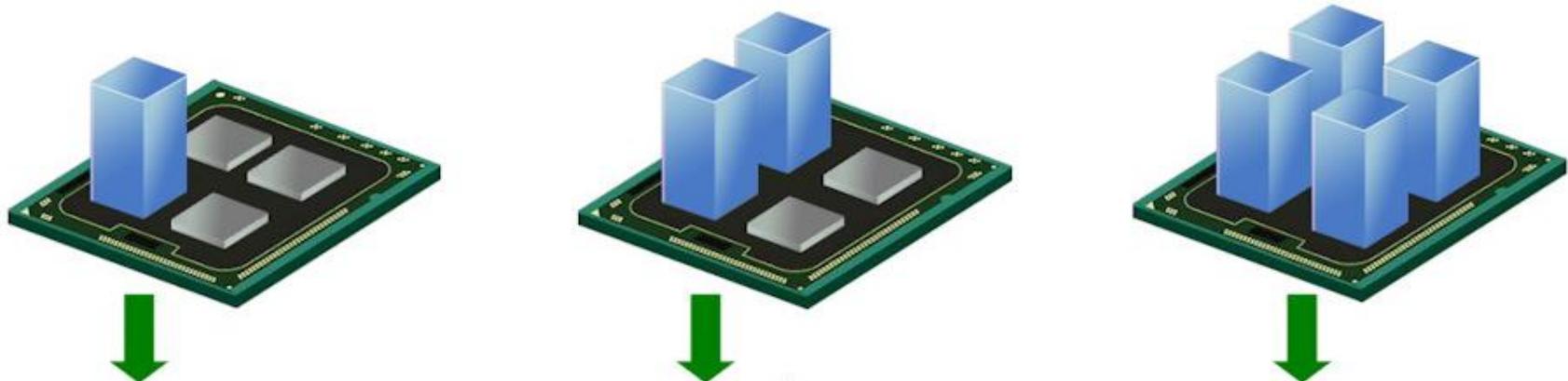
Potência:
Taxa de consumo de energia

Voltagem: Influenciada pela capacidade e frequência

Capacitância:
Capacidade de armazenar energia
(Fixo)

Frequência de operação
(chaveamentos por segundo)

DYNAMIC VOLTAGE AND FREQUENCY SCALING (ESCALA DINÂMICA DE TENSÃO E FREQUÊNCIA)



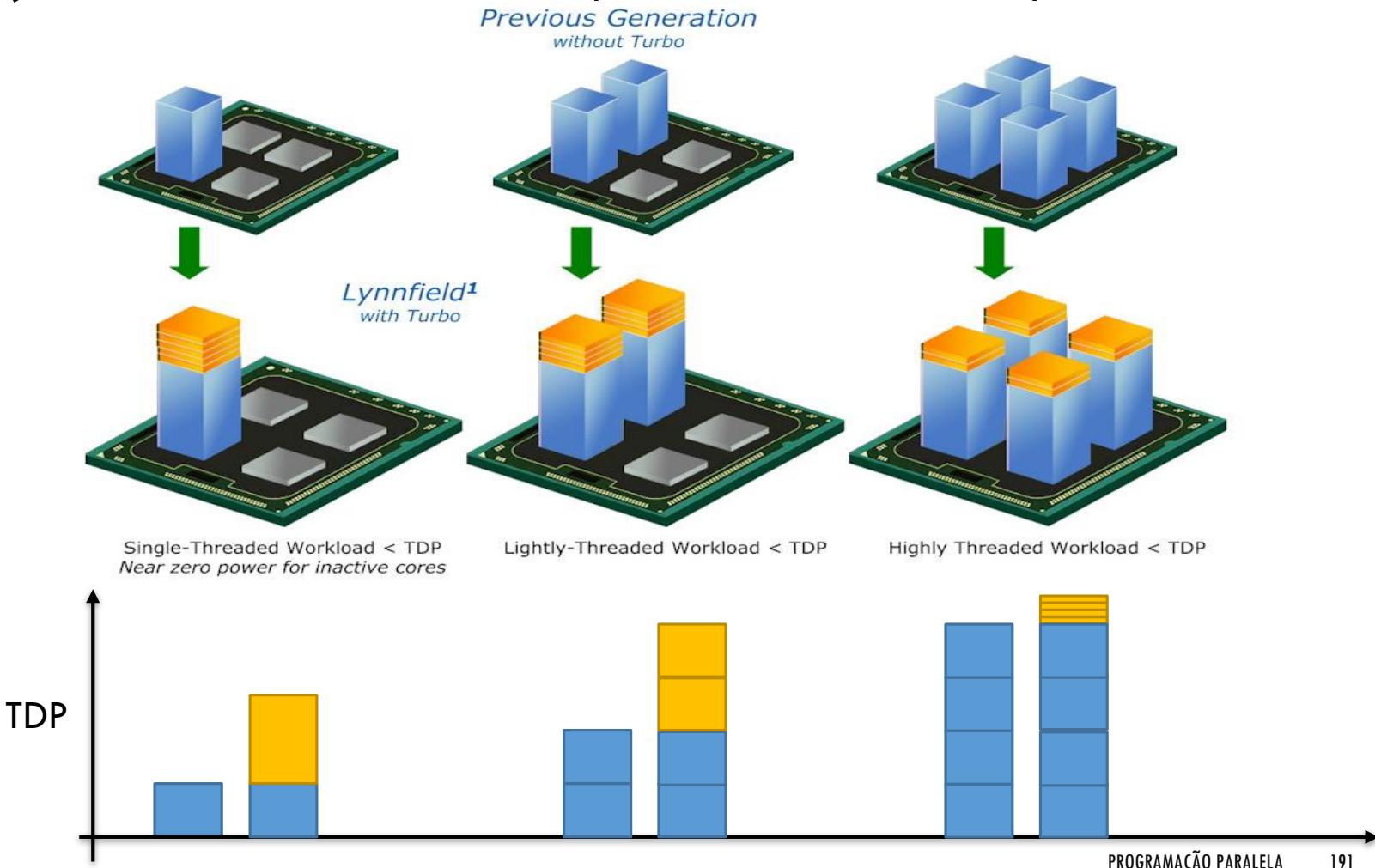
O processador pode assumir:

- Níveis de desempenho fixo: (low power, high performance)
- Modo dinâmico (idle, low power, high performance).

Os estados de desempenho do processador são definidos pela especificação Advanced Configuration and Power Interface (ACPI)

- Padrão aberto suportado por todos os principais sistemas operacionais;
- Nenhum software ou drivers adicionais são necessários para suportar a tecnologia.

OVERCLOCK DINÂMICO (TURBO BOOSTING / TURBO CORE)



TDP - THERMAL DESIGN POWER

Thermal Design Power: Basicamente nos diz, em Watts, quanto de calor um processador está gerando, servindo de parâmetro para os coolers que encontramos no mercado.

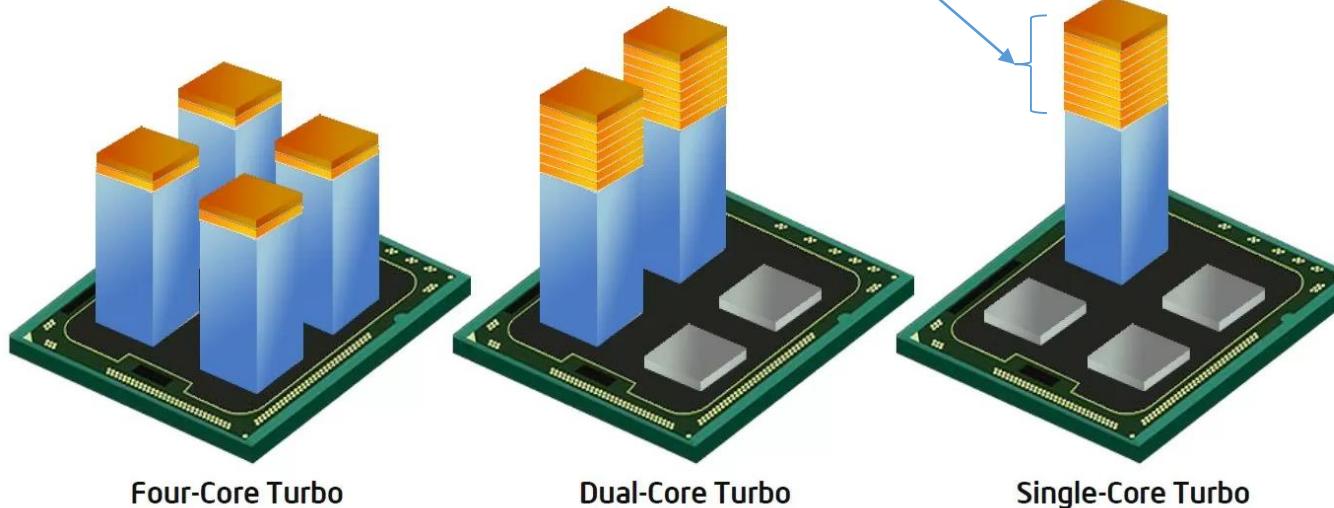
Via de regra, quanto maior o TDP do processador, mais calor ele gera.

Quanto mais alto o TDP de um cooler, mais calor ele será capaz de dissipar.

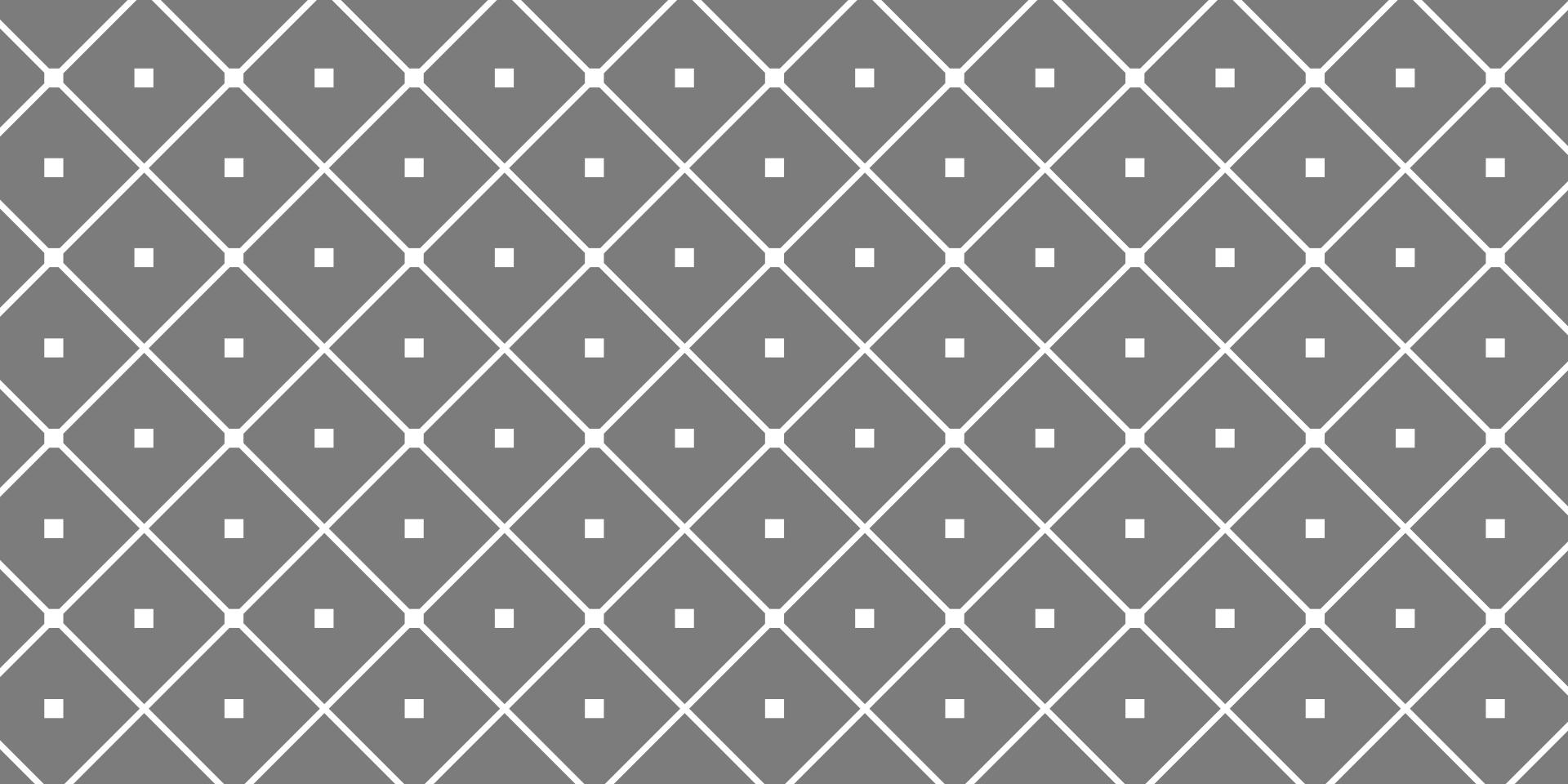
OVERCLOCK DINÂMICO (TURBO BOOSTING / TURBO CORE)

Em geral todos os núcleos ativos operam com a mesma frequência.

Não é raro encontrar processadores que aumentam em até 30% a frequência de operação

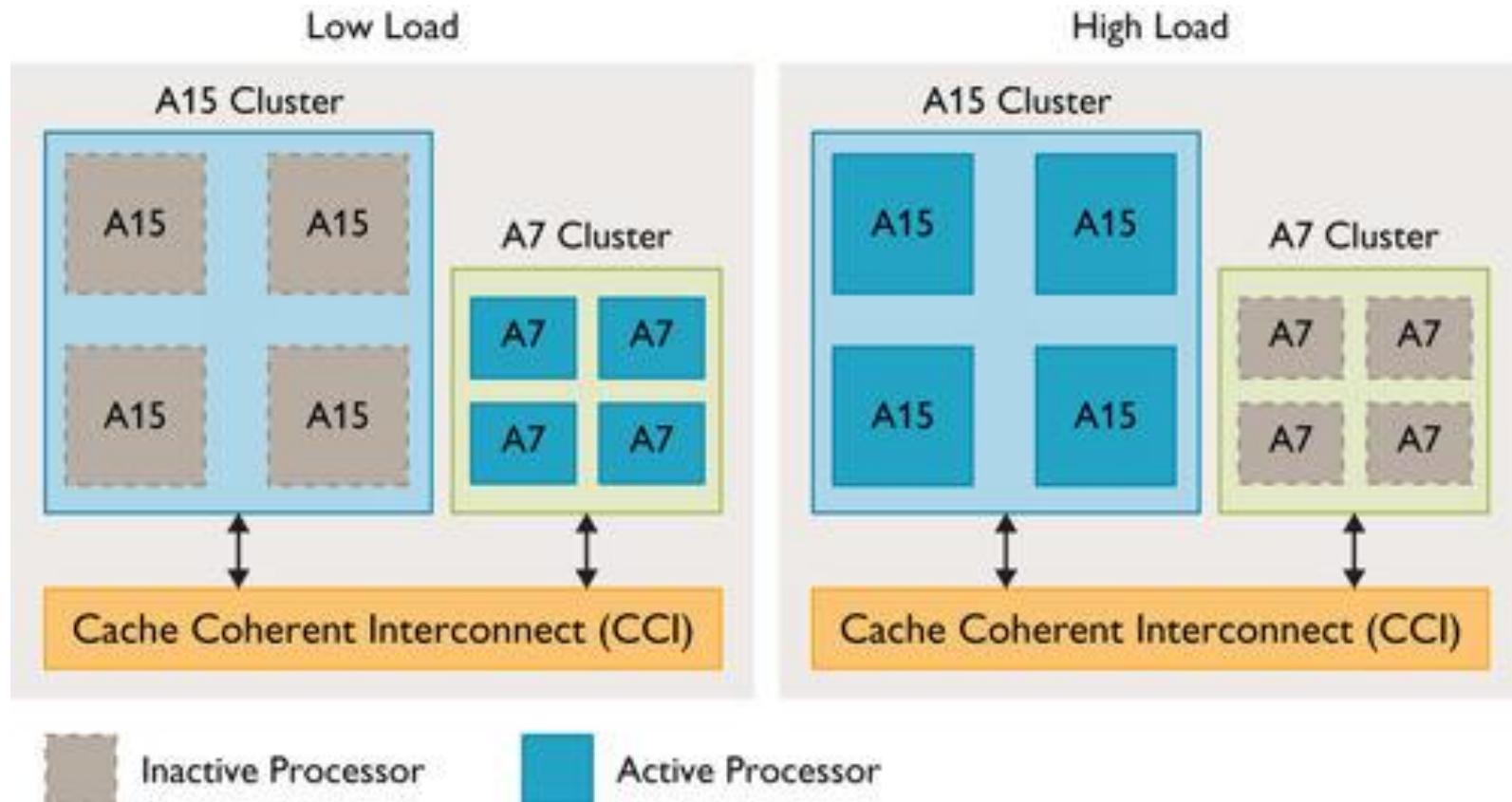


O rastreamento em tempo real da temperatura é capaz de elevar o clock além do limite superior de potência nominal por um curto período de tempo

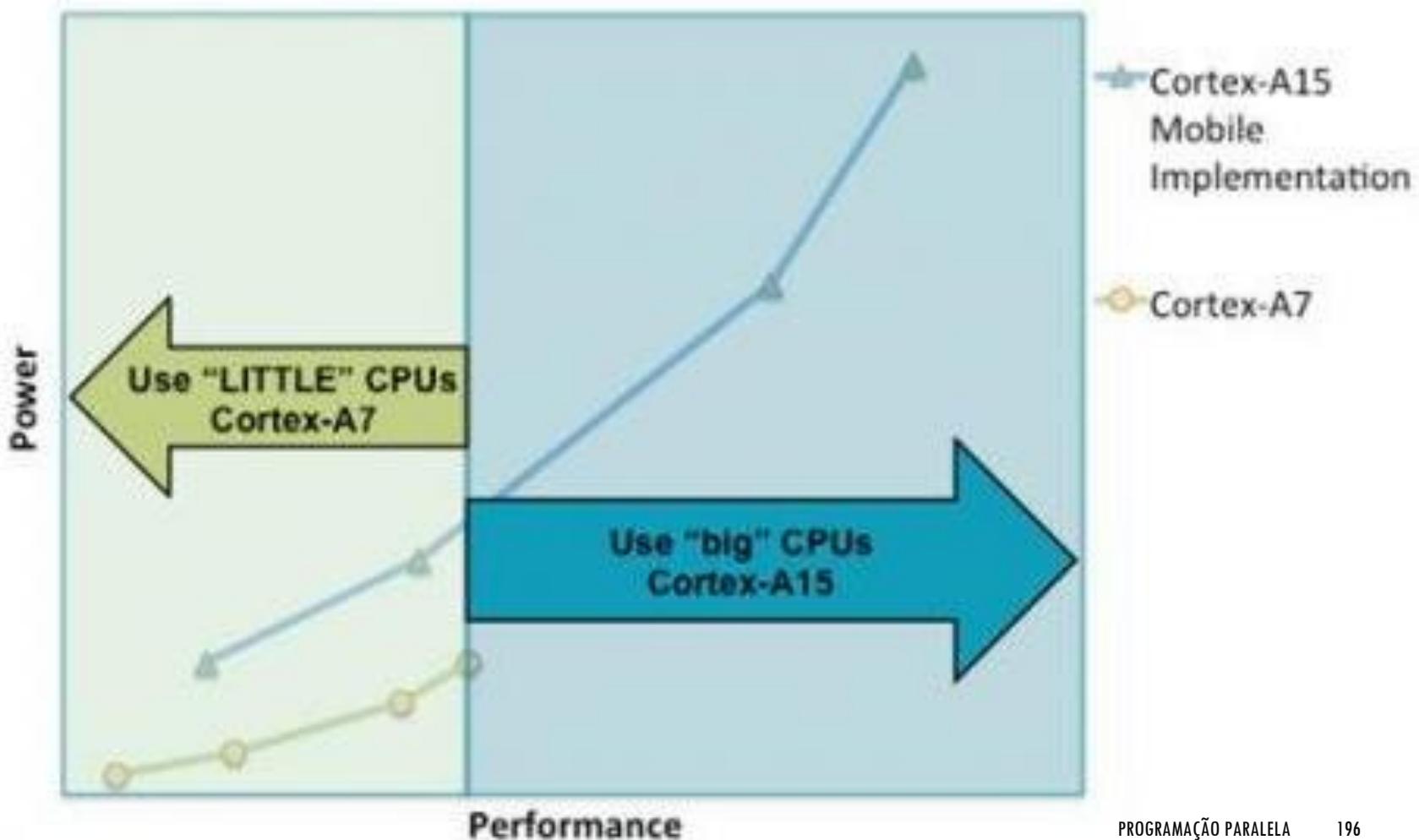


ARQUITETURAS HETEROGÊNEAS

ARM BIG.LITTLE



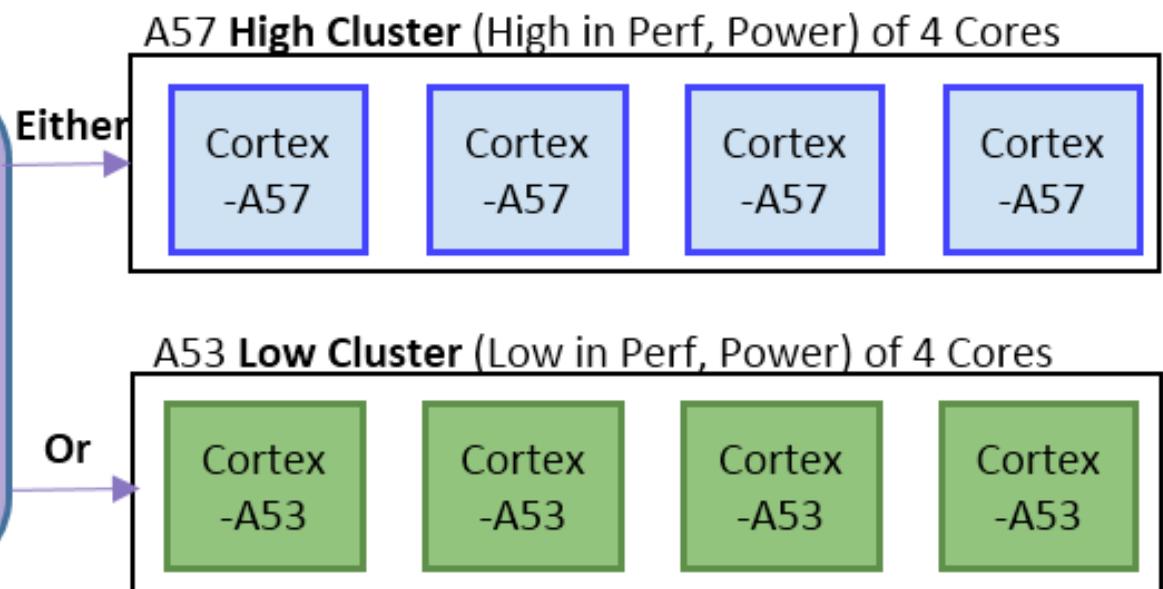
POWER CONSUMPTION



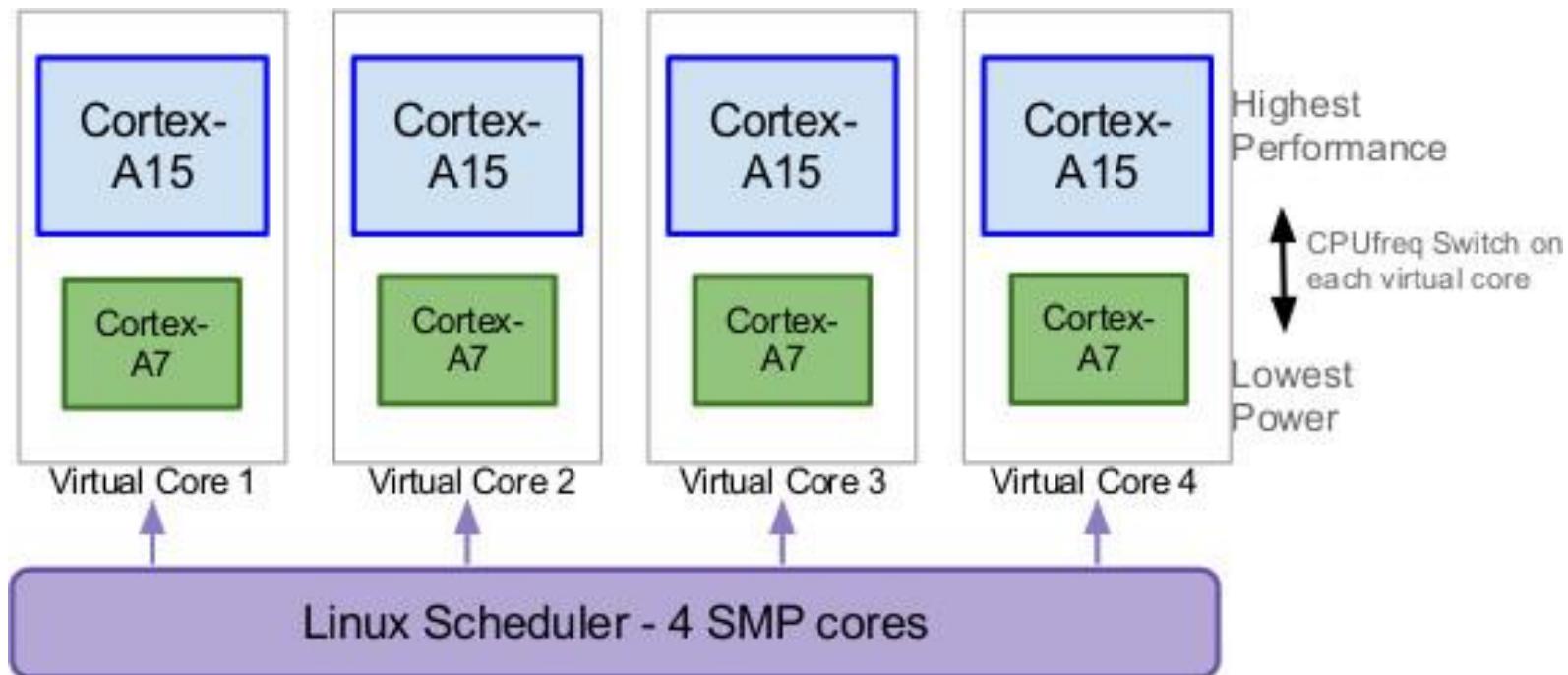
ESCALONAMENTO PELO SO

Linux Scheduler picks any **One Cluster** at a time,
But, no combinations

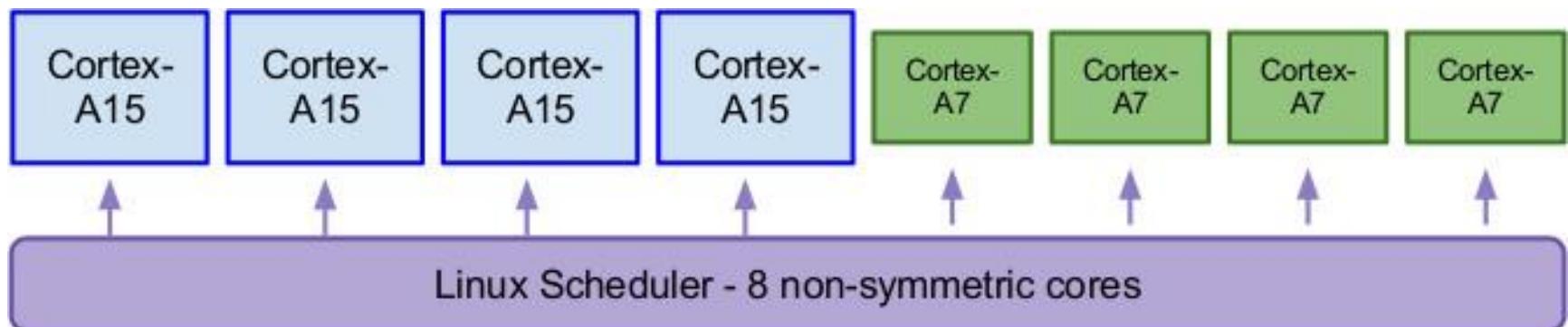
High Cluster is picked if at-least one High Core is needed, else **Low Cluster**

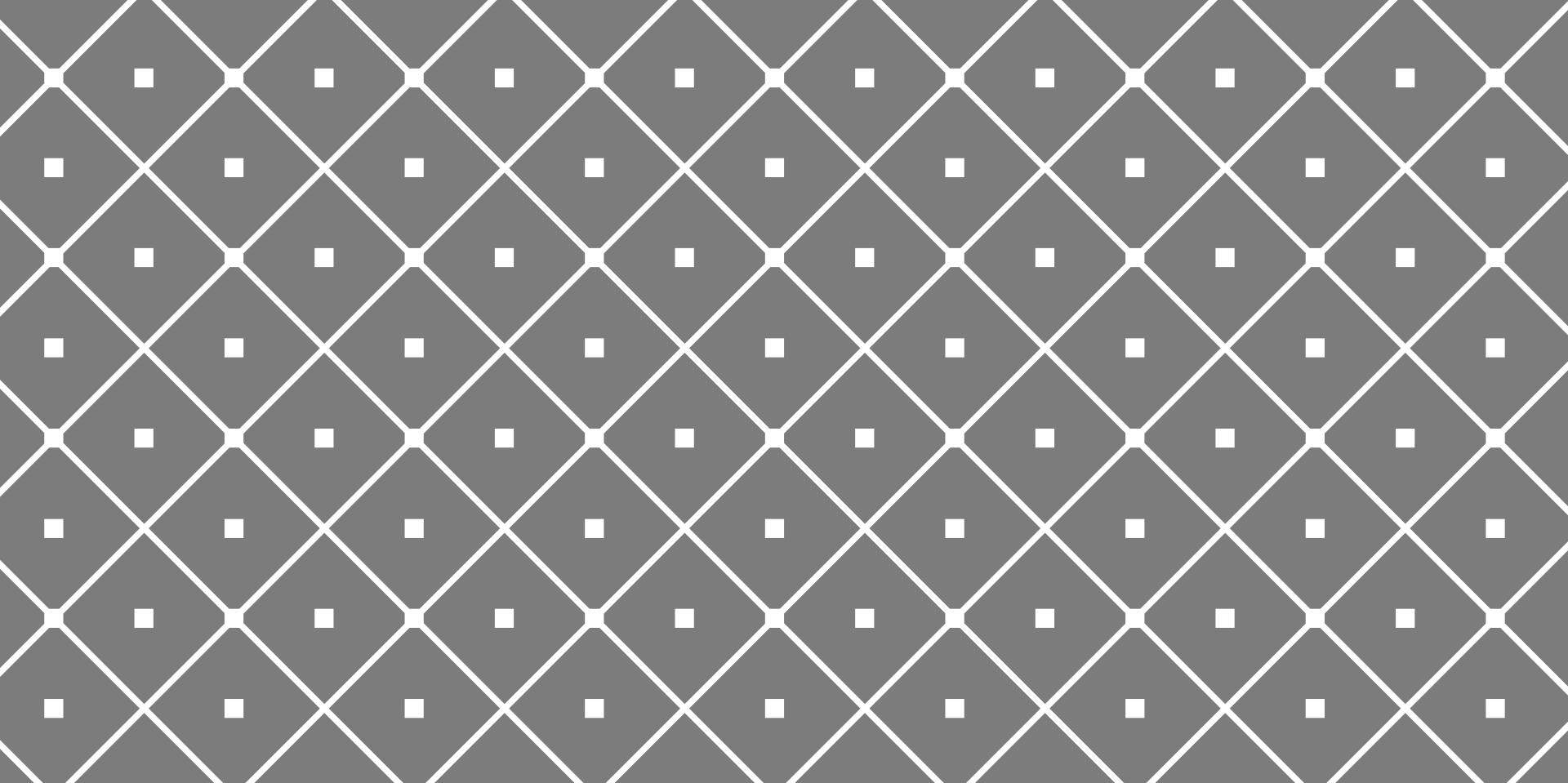


ESCALONAMENTO PELO SO



ESCALONAMENTO PELO SO





WRAP-UP

POR QUE PROGRAMAÇÃO PARALELA?

Os programas já não são rápidos o suficiente?

As máquinas já não são rápidas o suficiente?



- Requisitos estão sempre mudando
- Os problemas estão mais complexos

OS PROBLEMAS ESTÃO MAIS COMPLEXOS

Data volume (in zettabytes)

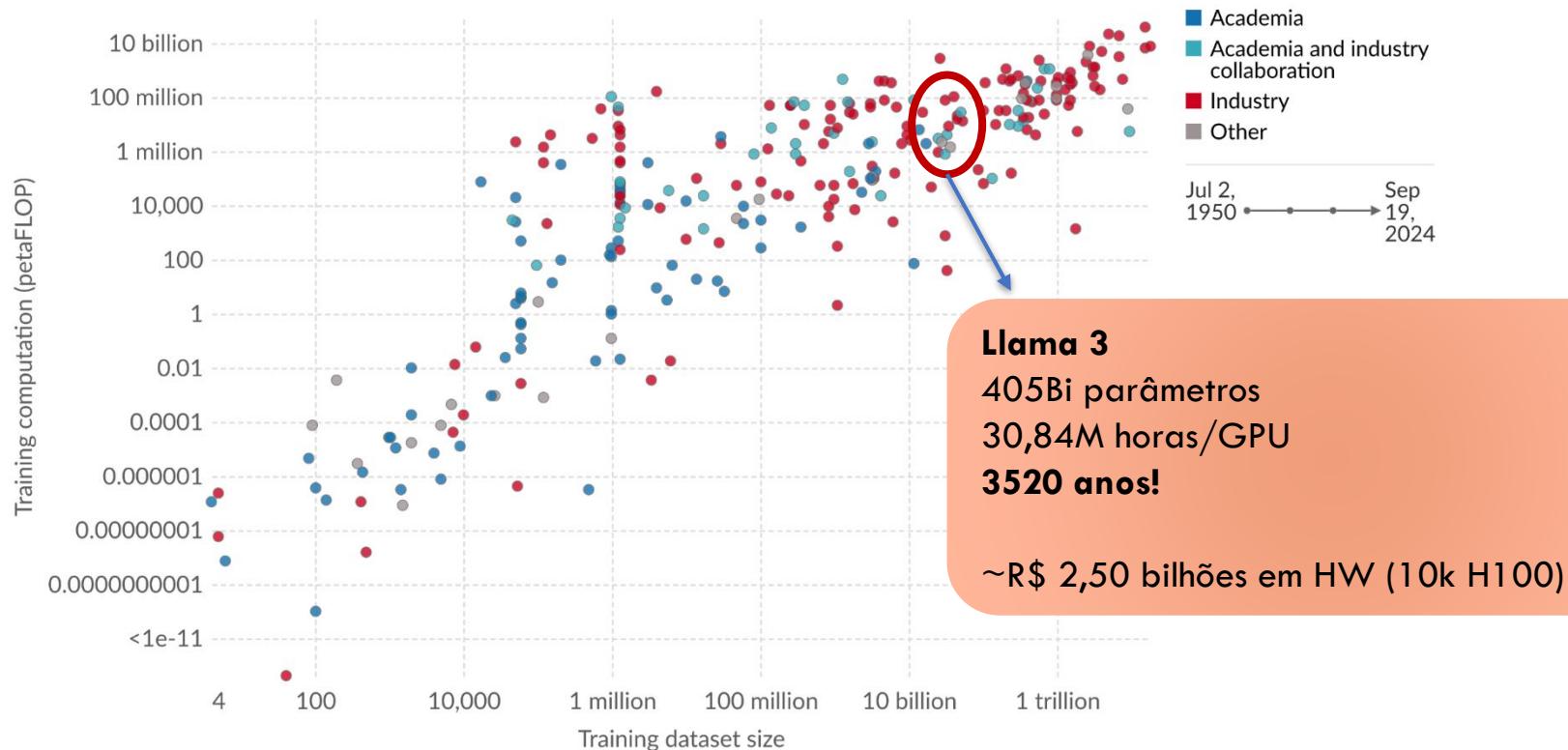


G2.com

Source: Statistica

Training computation vs. dataset size in notable AI systems, by researcher affiliation

Computation is measured in total petaFLOP, which is 10^{15} floating-point operations¹ estimated from AI literature, albeit with some uncertainty. Training dataset size refers to the volume of text that is employed to train a model effectively.



Data source: Epoch (2024)

OurWorldinData.org/artificial-intelligence | CC BY

1. Floating-point operation: A floating-point operation (FLOP) is a type of computer operation. One FLOP represents a single arithmetic operation involving floating-point numbers, such as addition, subtraction, multiplication, or division.

Embarcados com ML



PORQUÊ PROGRAMAÇÃO PARALELA?

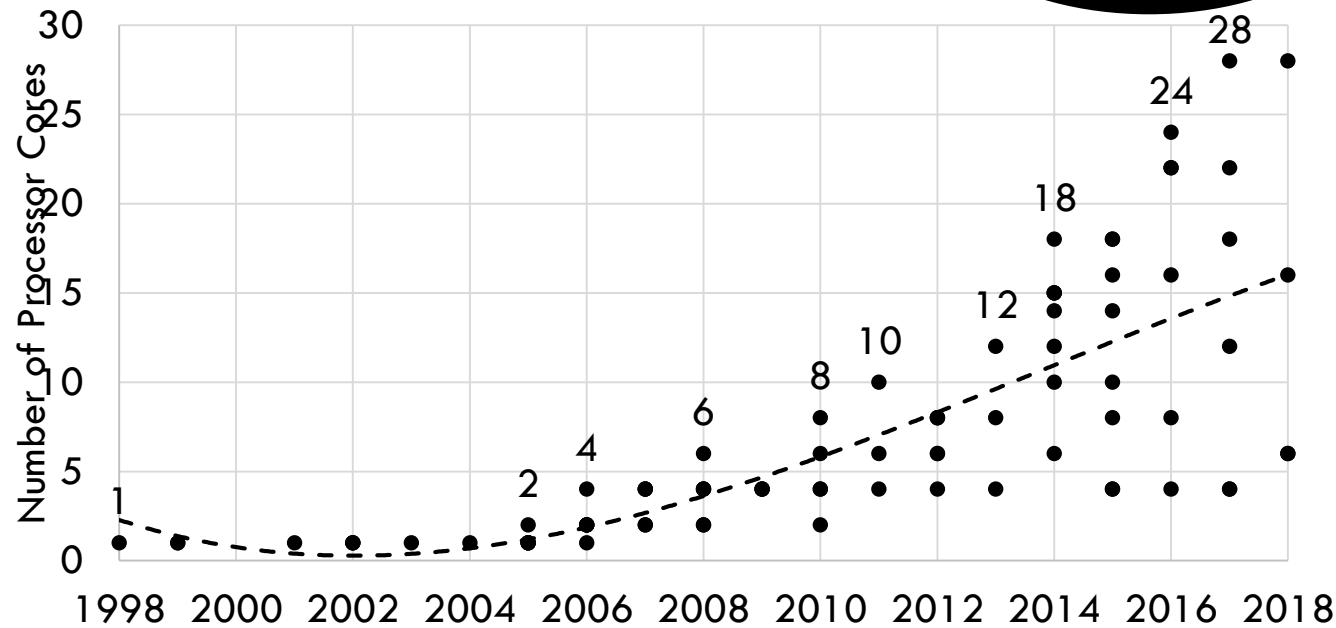
Dois dos principais motivos para utilizar programação paralela são:

- **Reducir o tempo** necessário para solucionar um problema.
- **Resolver problemas mais complexos** e de maior dimensão.

Outros motivos são: ???

EVOLUÇÃO DO INTEL XEON

Onde comprar um
processador single-
core?



PORQUÊ PROGRAMAÇÃO PARALELA?

Dois dos principais motivos para utilizar programação paralela são:

- Reduzir o tempo necessário para solucionar um problema.
- Resolver problemas mais complexos e de maior dimensão.

Outros motivos são:

- Utilizar recursos computacionais subaproveitados.
- Ultrapassar limitações de memória quando a memória disponível num único computador é insuficiente para a resolução do problema.
- Ultrapassar os limites físicos que atualmente começam a restringir a possibilidade de construção de computadores sequenciais cada vez mais rápidos.

ELEMENTOS PARA UM ECOSISTEMA HPC

Hardware

- Uso de servidores, clusters e supercomputadores

Software

- Software, ferramentas, componentes, armazenamento e serviços associados

Problemas a serem resolvidos

- Tarefas científicas, de engenharia ou analíticas

ELEMENTOS PARA UM ECOSISTEMA HPC

Hardware

- Uso de servidores, clusters e supercomputadores

Software

- Software, ferramentas, componentes, armazenamento e serviços associados

Problemas a serem resolvidos

- Tarefas científicas, de engenharia ou analíticas

Capital Humano (mais importante)

- Provedores de HPC - Planejar/instalar/gerenciar recursos de HPC
- Usuário de HPC - **Usar melhor o recurso de HPC**