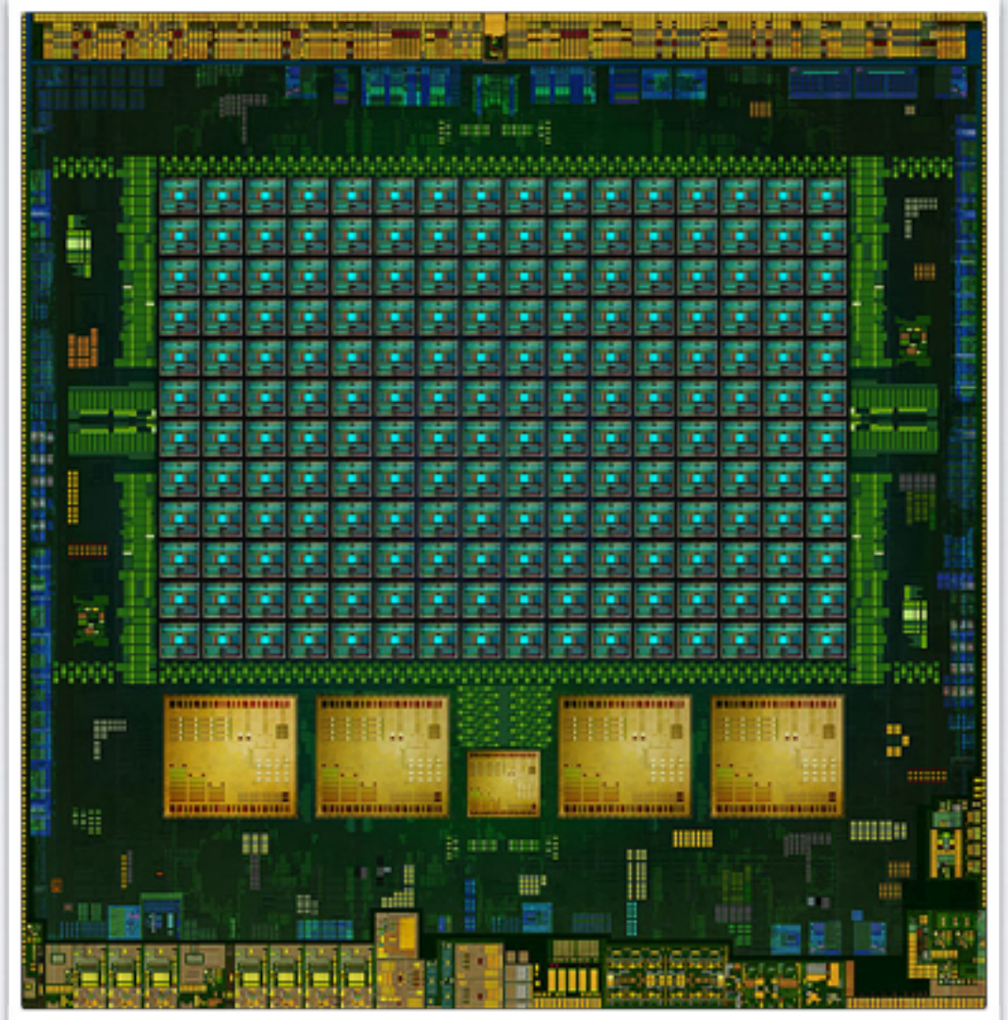
# ORDENAÇÃO DA SUBMISSÃO DE KERNELS CONCORRENTES PARA MAXIMIZAR A UTILIZAÇÃO DOS RECURSOS DA GPU

Aluno:
BERNARDO BREDER
Orientador:
LÚCIA M. A. DRUMMOND
Membro da Banca:
ESTEBAN CLUA

# AGENDA

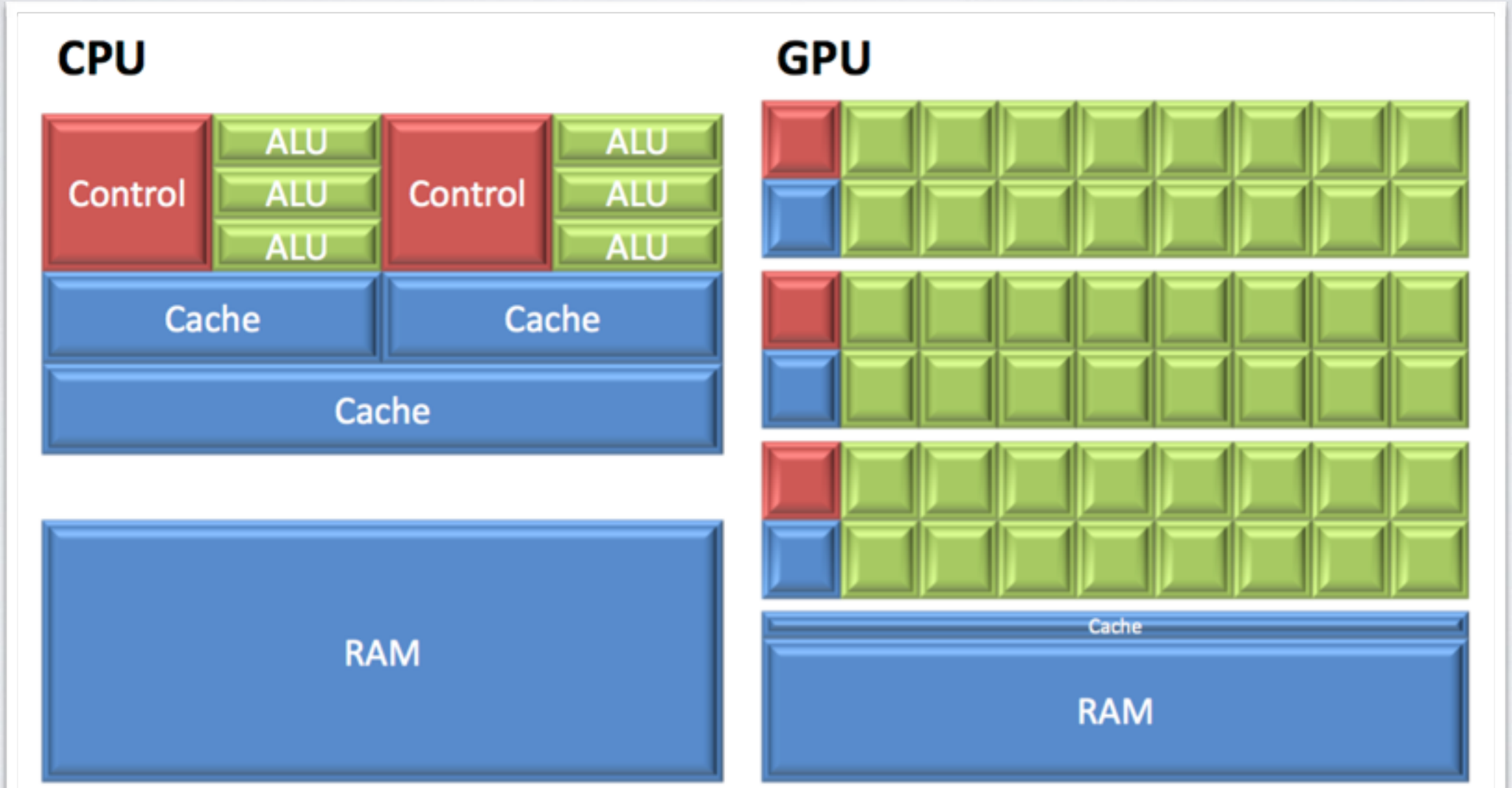
- Introdução
- Trabalhos Relacionados
- Processo de Ordenação
- Problema da Mochila
- Adaptação para GPU
- Algoritmo de ordenação
- Resultados



# INTRODUÇÃO

## Introdução da arquitetura e do trabalho

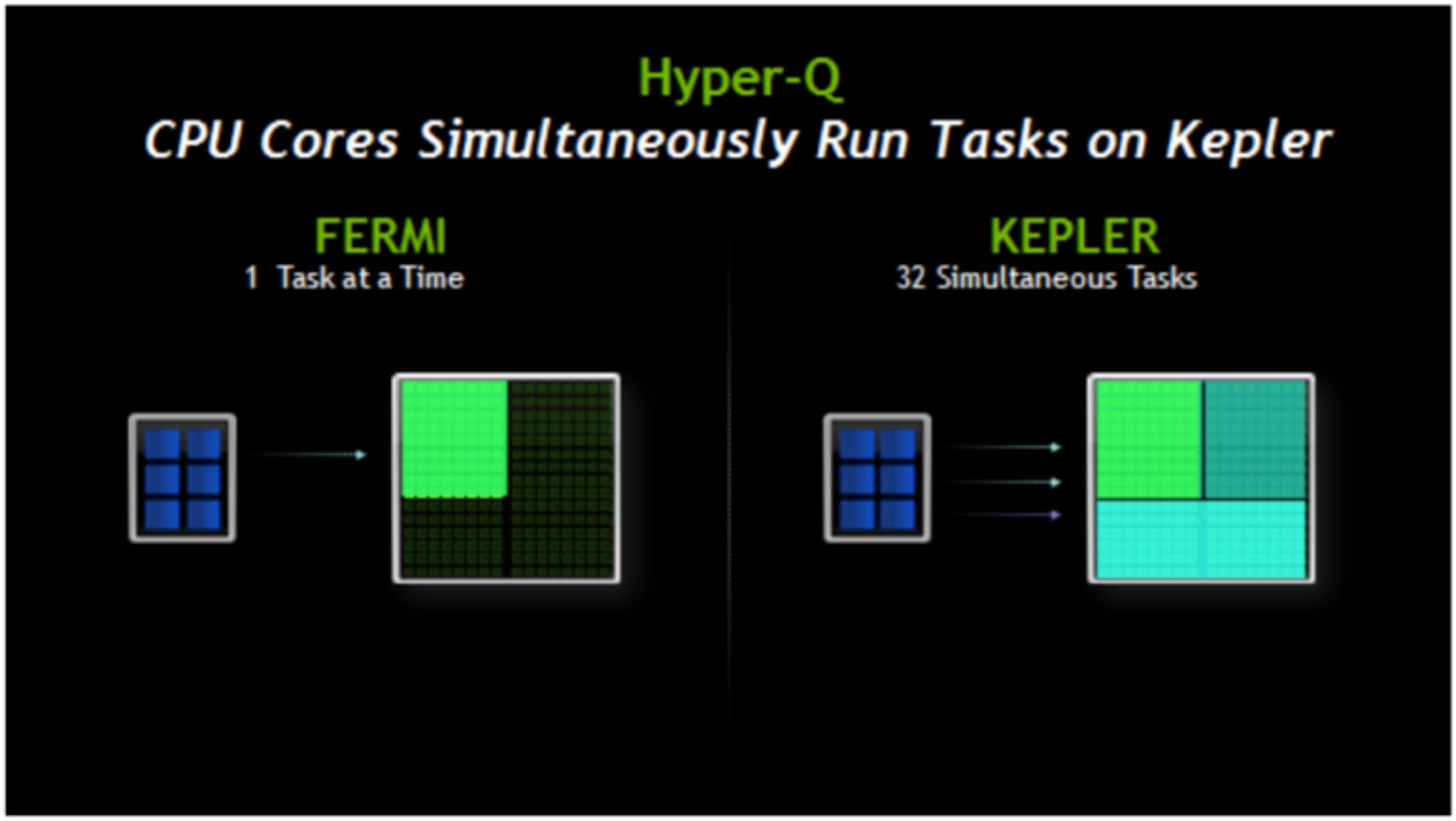
# INTRODUÇÃO


CPU
Control
ALU
ALU
ALU
Control
ALU
ALU
ALU
Cache
Cache
Cache
RAM
GPU
Cache
RAM

# INTRODUÇÃO

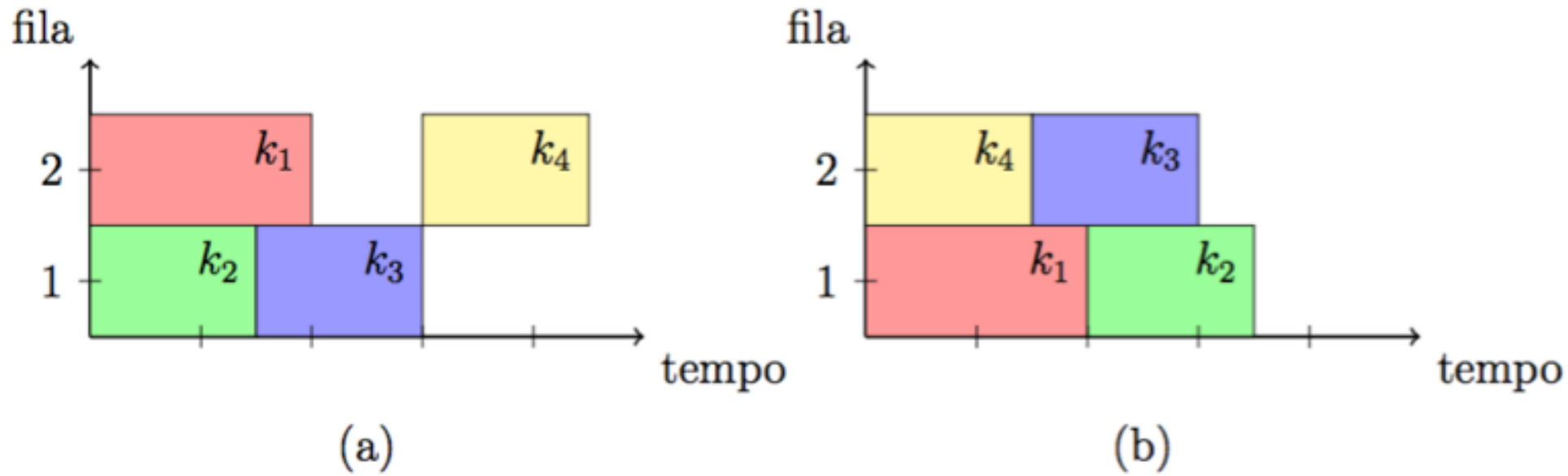
| | **TitanX** | **K40** |
|---|---|---|
| Número de Cores | 3,072 | 2,880 |
| Core Clock | 1000 MHz | 745 MHz |
| RAM | 24GB | 12GB |
| Memory Bandwidth | 336.5 GB/s | 288 GB/s |
| Capability | 5.2 | 3.5 |
| Número de SMs | 24 | 15 |
| Shared Memory por SM | 96KB | 48KB |
| Numero of Registers por SM | 64K | 64K |
| Max número de threads por SM | 2048 | 2048 |
| Arquitetura | Maxwell | Kepler |
| Pico de desempenho | 6.6 TFLOPs | 4.2 TFLOPS |

# INTRODUÇÃO


Hyper-Q
CPU Cores Simultaneously Run Tasks on Kepler
FERMI
1 Task at a Time
KEPLER
32 Simultaneous Tasks

# INTRODUÇÃO

| Exemplo Didático | Kernels | | | |
|---|---|---|---|---|
| | $k_1$ | $k_2$ | $k_3$ | $k_4$ |
| Recursos ($w_i$) | 30% | 30% | 50% | 60% |
| Tempo ($t_i$) | $20ms$ | $15ms$ | $15ms$ | $15ms$ |



# TRABALHOS RELACIONADOS

Estudos anteriores na área de execução de kernels concorrentes

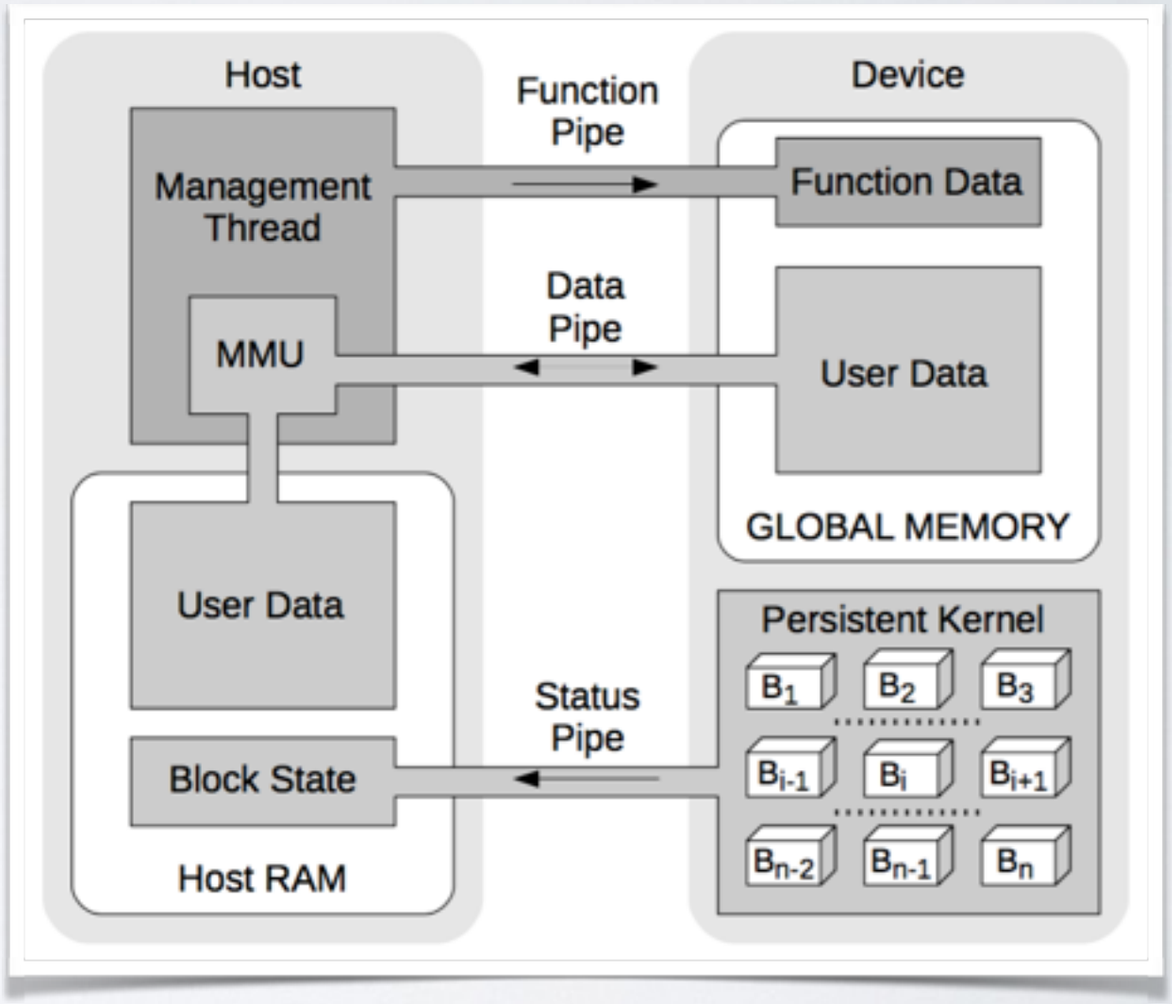
# TRABALHOS RELACIONADOS

## UNIÃO

União de kernels em 1 para simular concorrência

# TRABALHOS RELACIONADOS TIPO - UNIÃO

- Peters et al. [37]
- Cada bloco de threads observa um espaço de memória em um laço infinito
- Ao ler argumentos e a funções, os algoritmo é executado



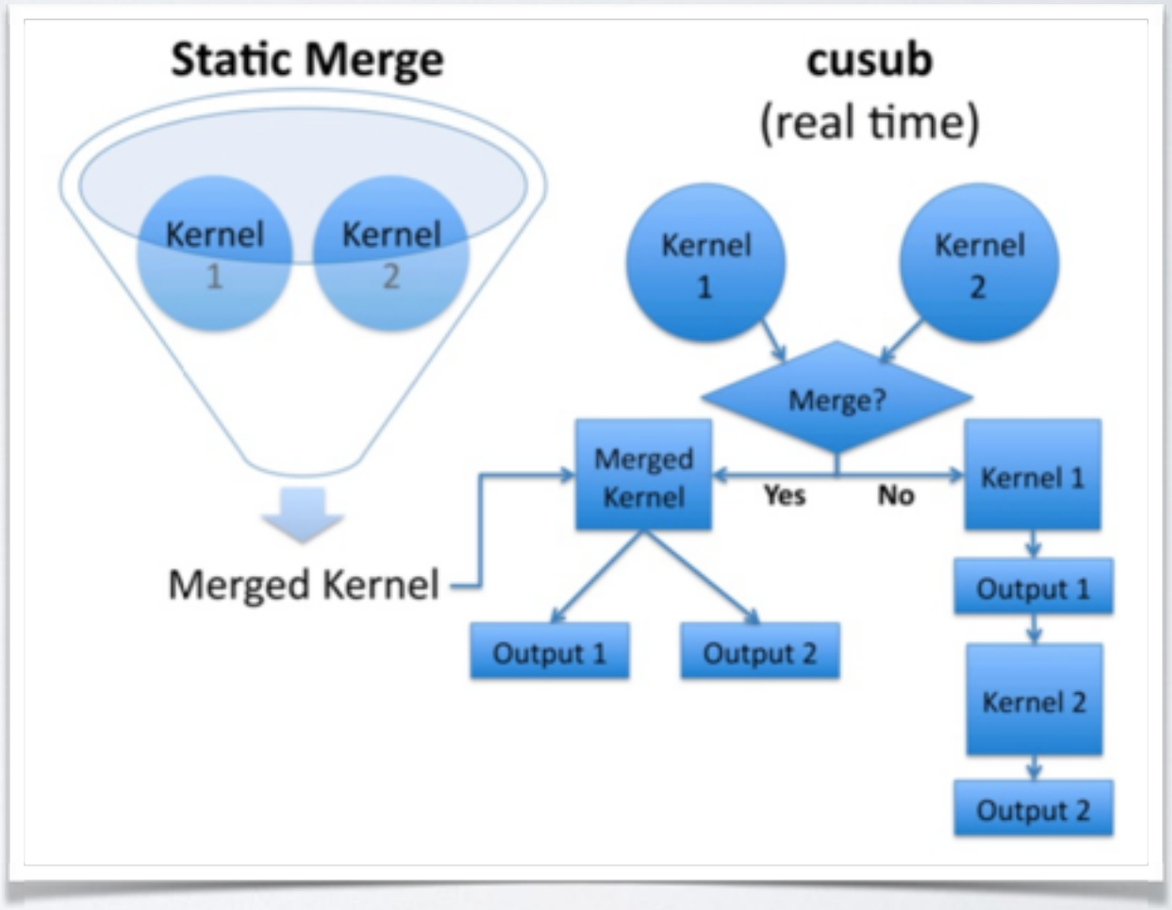
# TRABALHOS RELACIONADOS TIPO - UNIÃO

- Guevara et al. [17]

- Merging de kernels para simular concorrência

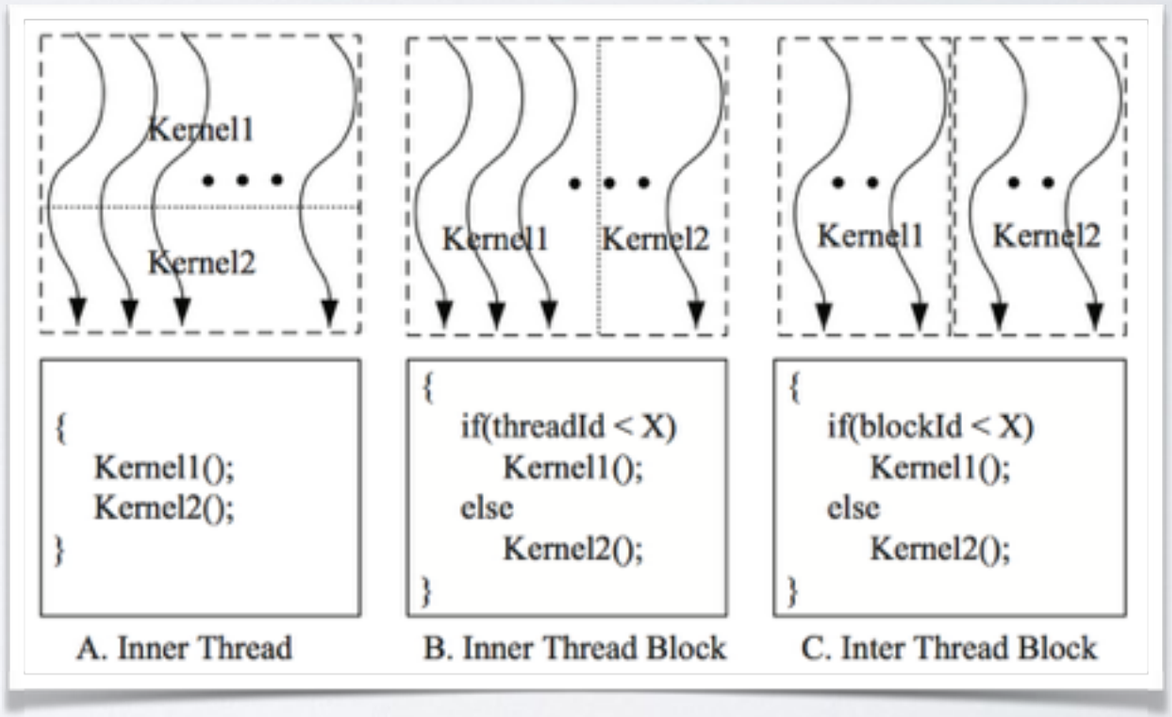
- Processo de compilação que une 2 kernels em 1

- Somente kernels com recursos concorrentes são unidos



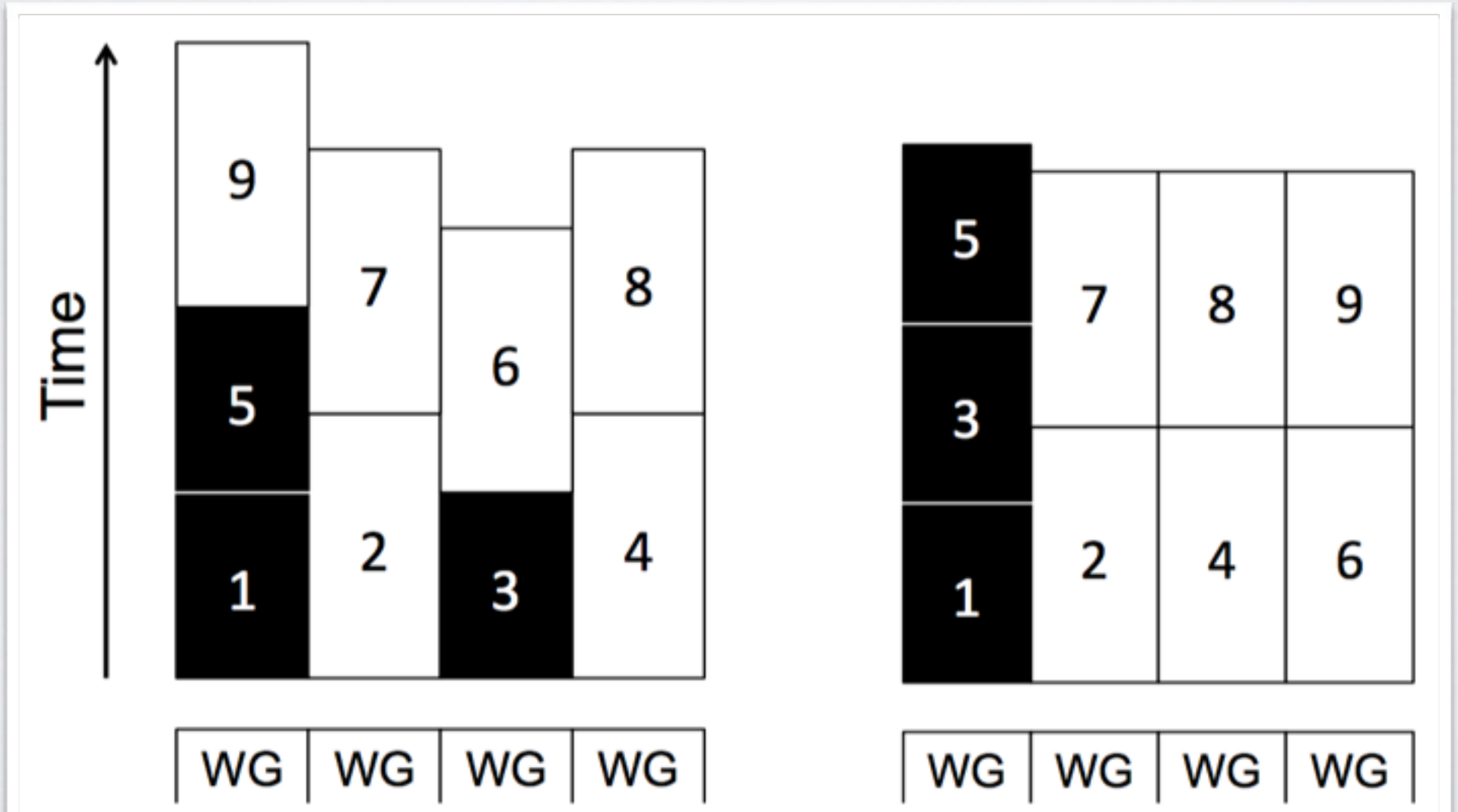
# TRABALHOS RELACIONADOS TIPO - UNIÃO

- G. Wang et al. [45]
- Une códigos com objetivo de reduzir consumo de energia
- Possui métodos de computar consumo de energia



# TRABALHOS RELACIONADOS TIPO - UNIÃO

Gregg et al. [16]



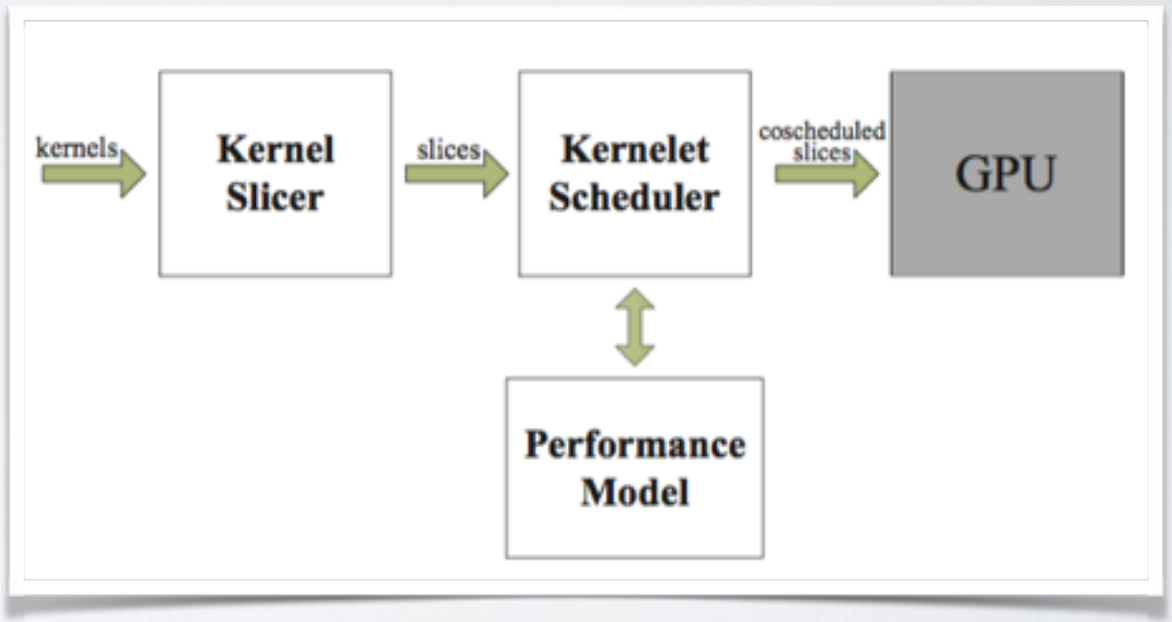
# TRABALHOS RELACIONADOS

## DIVISÃO

Dividir o kernel em unidades menores para melhorar a taxa de utilização da GPU

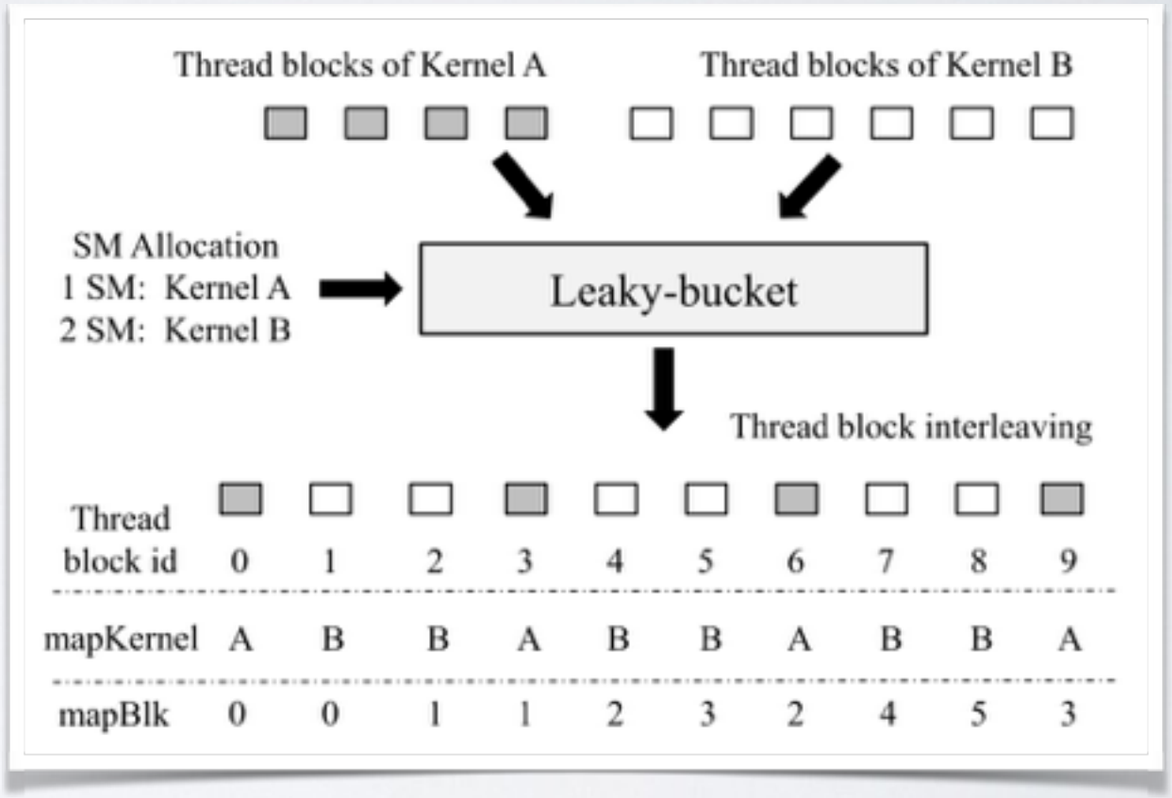
# TRABALHOS RELACIONADOS TIPO - DIVISÃO

- G. Wang et al. [45]
- Propõem "kernelet" que divide os kernels em kernels menores
- A partir do modelo de performance, os "kernelet" são escalonados



# TRABALHOS RELACIONADOS TIPO - DIVISÃO

- Liang et al. [22]
- Quantifica a memoria utilizada para cada kernel
- Particiona os blocos de threads dos kernels nos SMs
- Decide aonde os kernels irão executar simulando concorrência



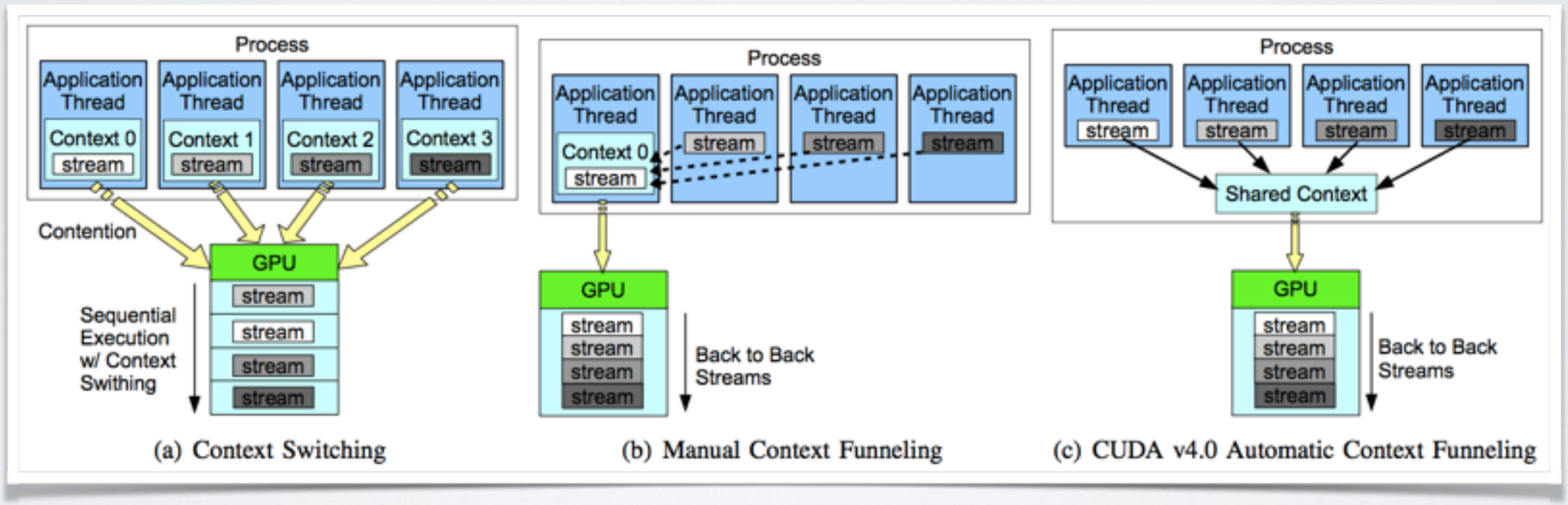
# TRABALHOS RELACIONADOS

## CONTEXTO

Compartilhamento de contexto para diferentes aplicações

# TRABALHOS RELACIONADOS
# TIPO - CONTEXTO

## L. Wang et al. [46]

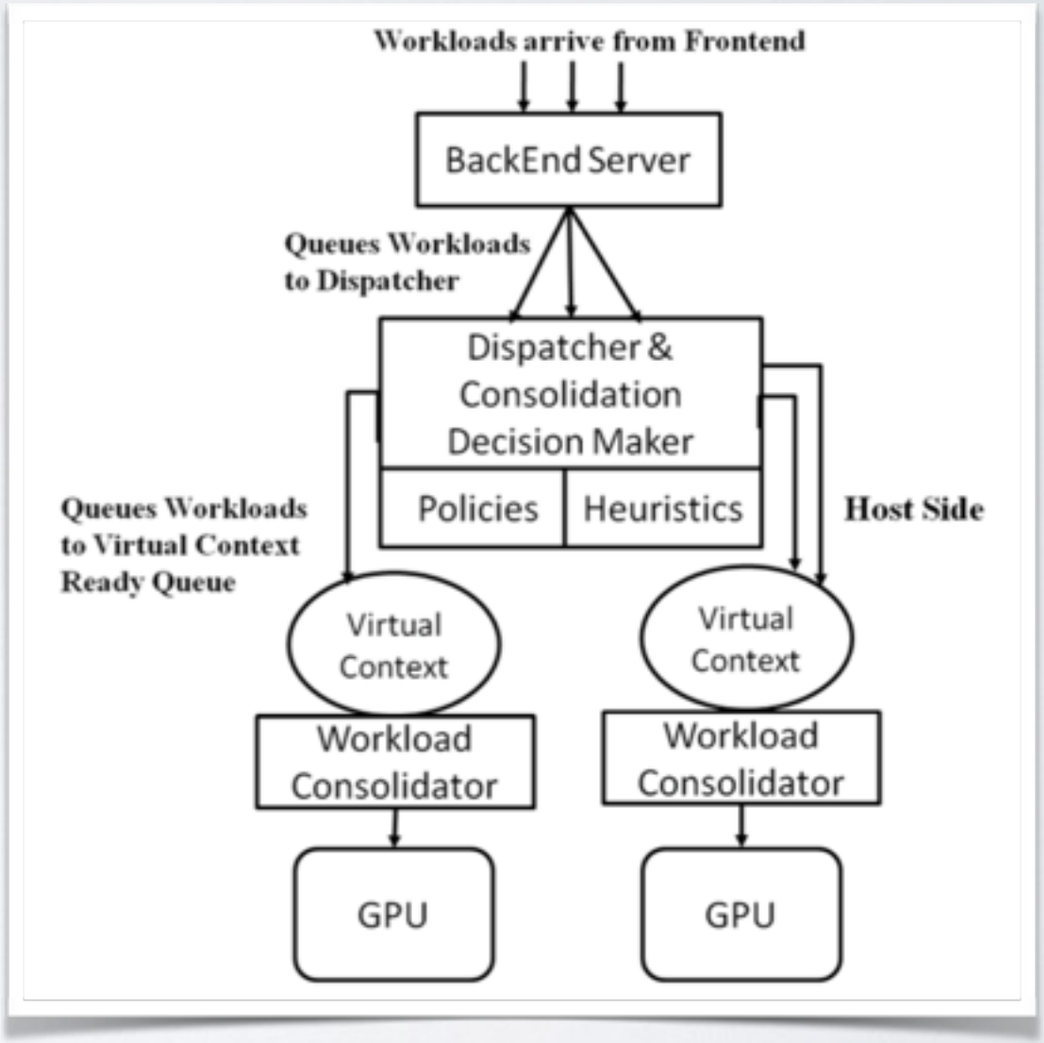



(a) Context Switching (b) Manual Context Funneling (c) CUDA v4.0 Automatic Context Funneling

# TRABALHOS RELACIONADOS TIPO - CONTEXTO

- Ravi et al. [38]

- Framework de aplicações de contextos diferentes executando numa máquina virtual de forma transparente



# TRABALHOS RELACIONADOS

## REORDENAÇÃO

Reordenação dos kernels que são submetidos para GPU para melhorar a taxa de ocupação

# TRABALHOS RELACIONADOS TIPO - REORDENAÇÃO

- Wende et al. [48]

- Na arquitetura Fermi foi proposto uma reordenação seguindo o Round-Robin

- Intercala diferentes filas de execução

- Hardware mais moderno possui o Hyper-Q

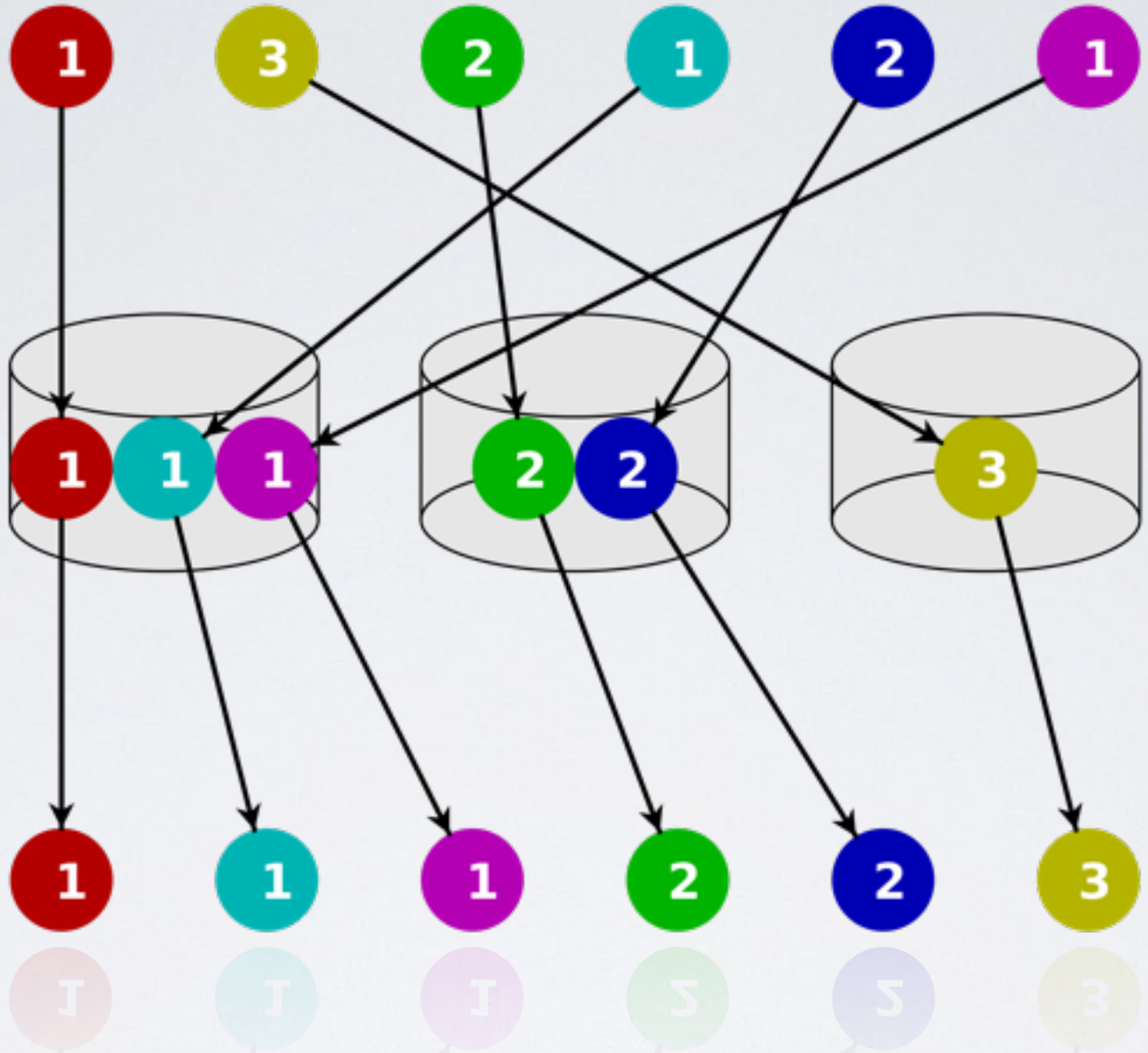
# TRABALHOS RELACIONADOS TIPO - REORDENAÇÃO

- Li et al. [21]

- Utiliza do Hyper-Q para ordernar os kernels concorrentes

- Utiliza de técnicas de Guloso para escalonar as tarefas nas filas de execução e outras variáveis

- A técnica de Guloso não garante a solução ótima.

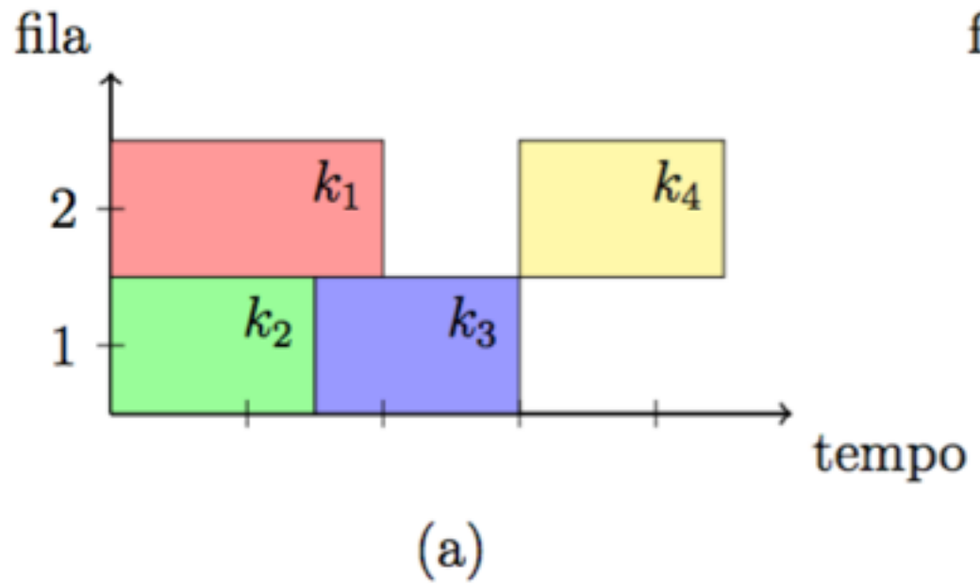


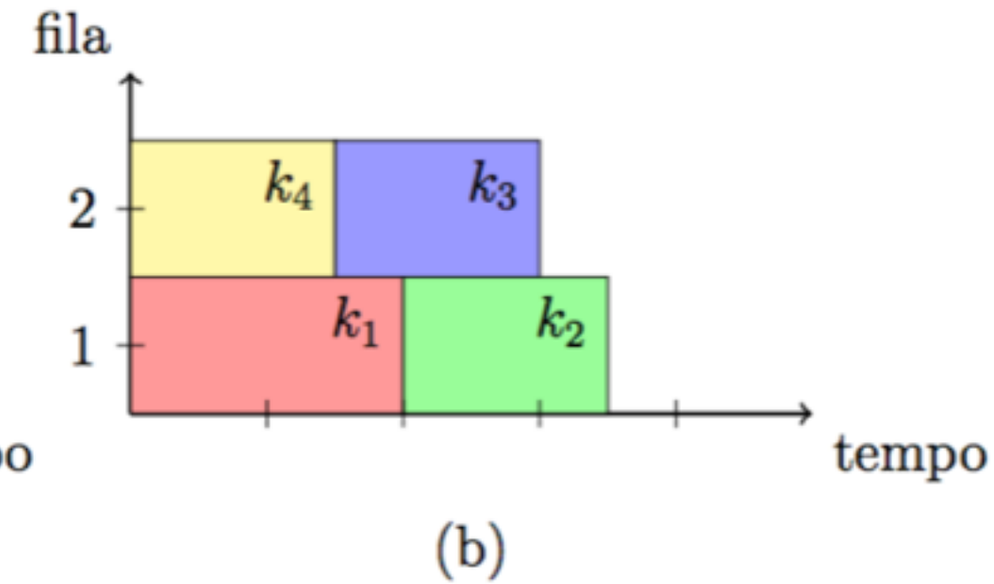

# PROCESSO DE ORDENAÇÃO

Estudo de ordenação mostrando por fim a entrada e a saída de uma ordenação de kernels

# PROCESSO DE ORDENAÇÃO

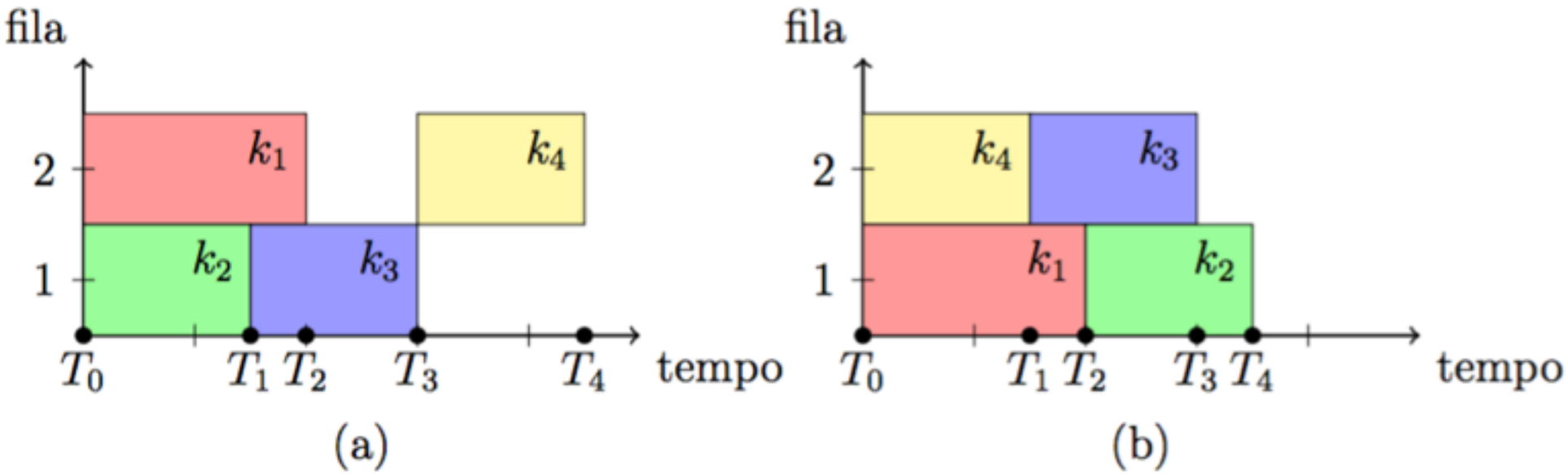
| Exemplo Didático | Kernels | | | |
|---|---|---|---|---|
| | $k_1$ | $k_2$ | $k_3$ | $k_4$ |
| Recursos $(w_i)$ | 30% | 30% | 50% | 60% |
| Tempo $(t_i)$ | $20ms$ | $15ms$ | $15ms$ | $15ms$ |



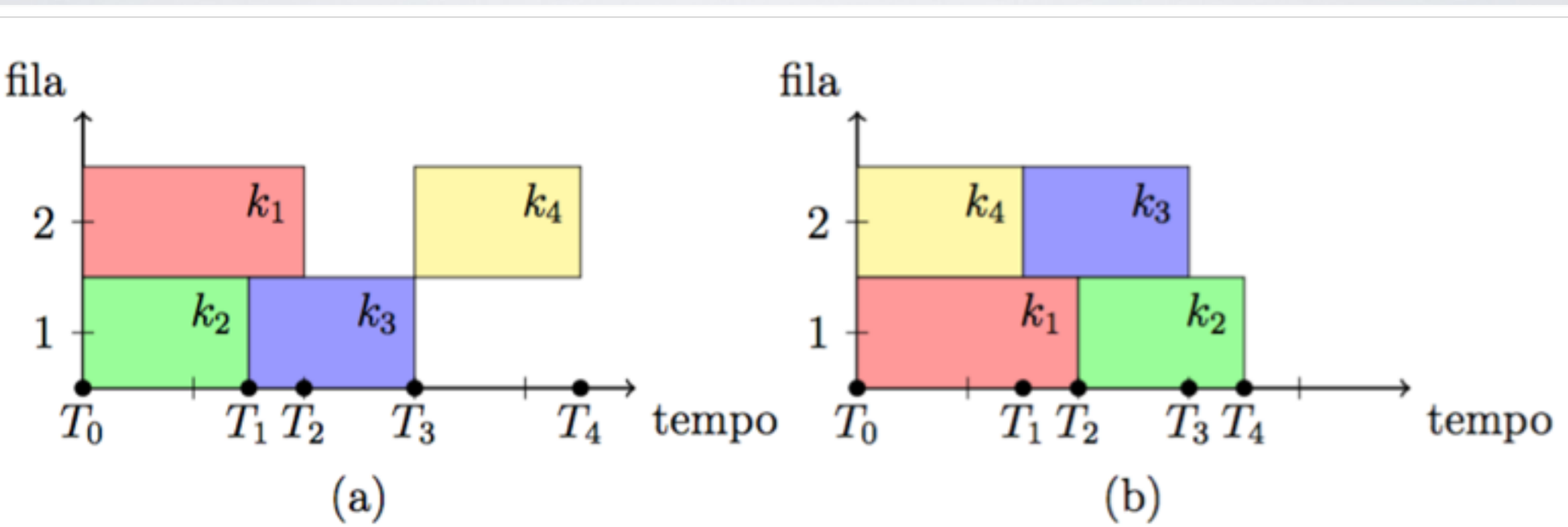



# PROCESSO DE ORDENAÇÃO

| Exemplo Didático | Kernels | | | |
|---|---|---|---|---|
| | $k_1$ | $k_2$ | $k_3$ | $k_4$ |
| Recursos ($w_i$) | 30% | 30% | 50% | 60% |
| Tempo ($t_i$) | $20ms$ | $15ms$ | $15ms$ | $15ms$ |



# PROCESSO DE ORDENAÇÃO



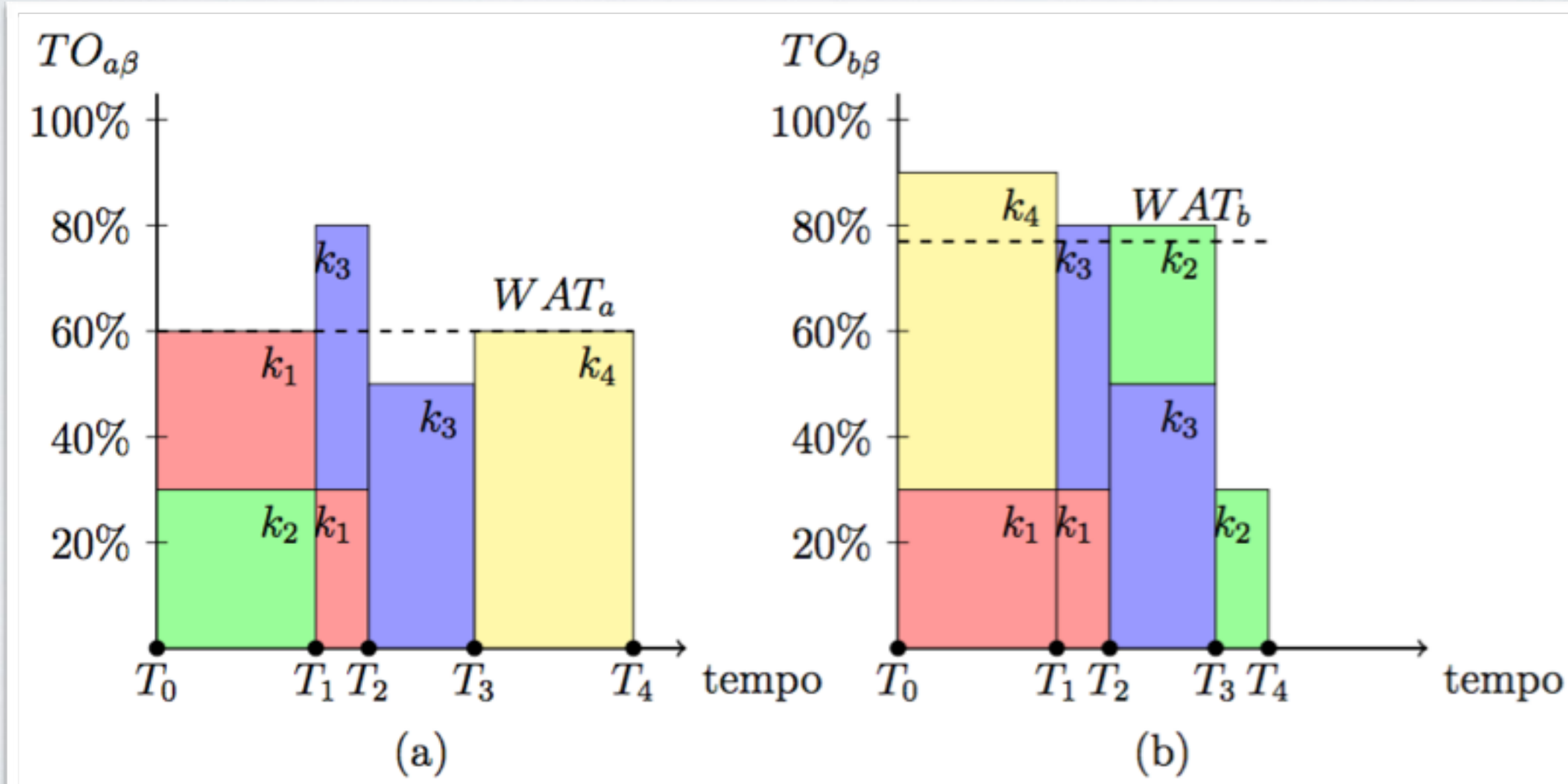

| $TO_{\alpha\beta}$ | $T_0$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $WAT_\alpha$ |
|---|---|---|---|---|---|---|
| $Ord_a$ | $60\% \times 15ms$ | $80\% \times 5ms$ | $50\% \times 10ms$ | $60\% \times 15ms$ | $0\% \times 0ms$ | 60% |
| $Ord_b$ | $90\% \times 15ms$ | $80\% \times 5ms$ | $80\% \times 10ms$ | $30\% \times 5ms$ | $0\% \times 0ms$ | 77% |

Tabela 3.2: Taxa de ocupação para cada instante $T_i$

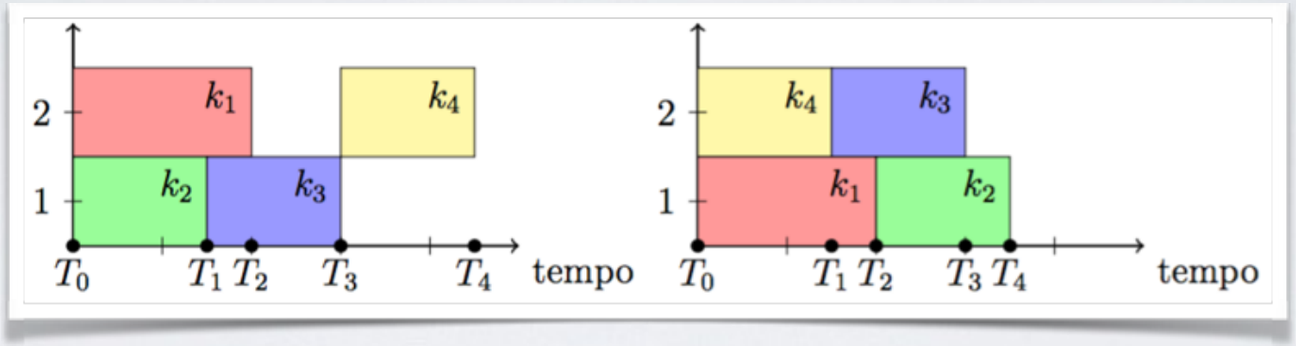
# PROCESSO DE ORDENAÇÃO

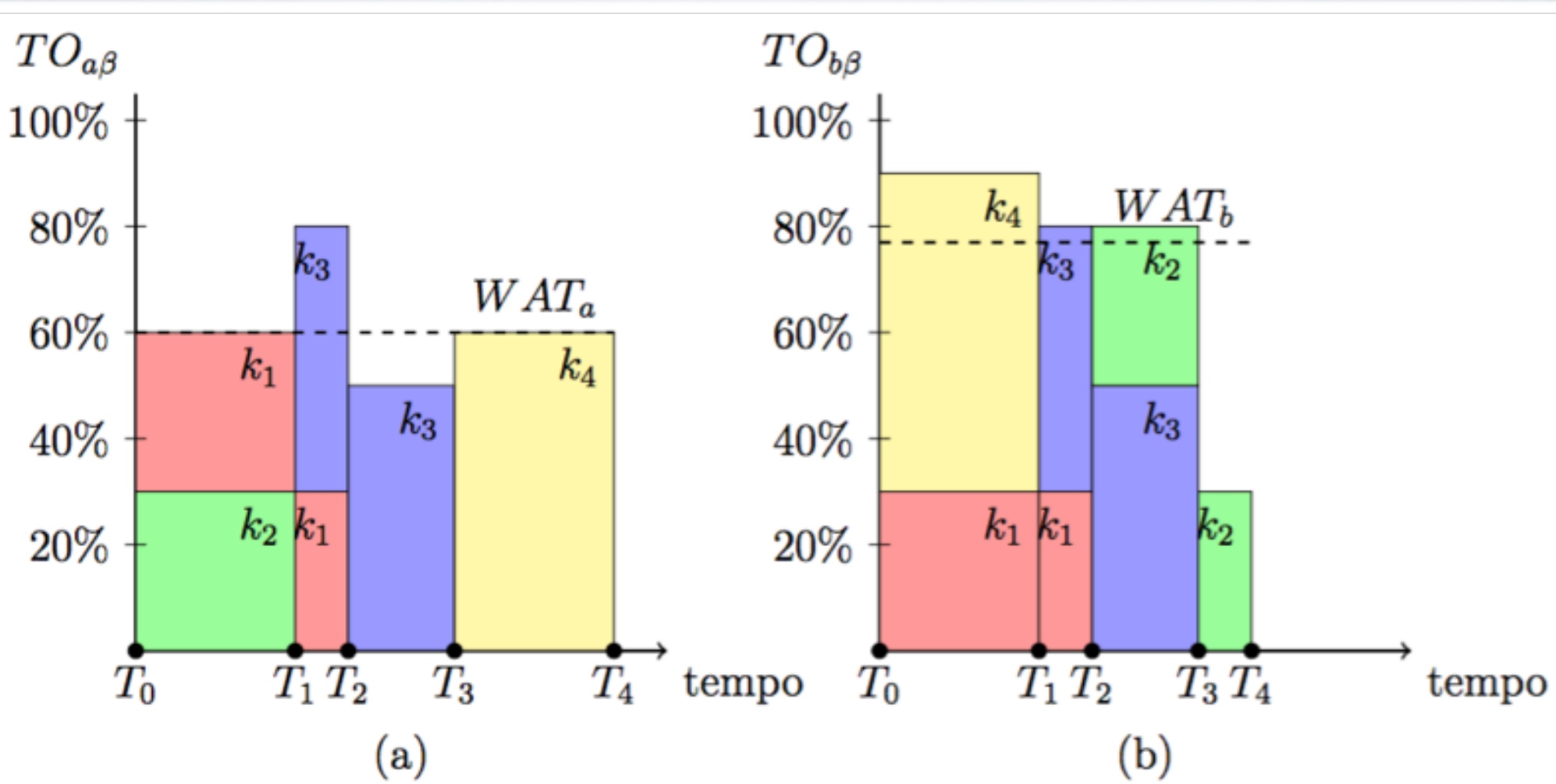
| $TO_{\alpha\beta}$ | $T_0$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $WAT_\alpha$ |
|---|---|---|---|---|---|---|
| $Ord_a$ | $60\% \times 15ms$ | $80\% \times 5ms$ | $50\% \times 10ms$ | $60\% \times 15ms$ | $0\% \times 0ms$ | 60% |
| $Ord_b$ | $90\% \times 15ms$ | $80\% \times 5ms$ | $80\% \times 10ms$ | $30\% \times 5ms$ | $0\% \times 0ms$ | 77% |

Tabela 3.2: Taxa de ocupação para cada instante $T_i$

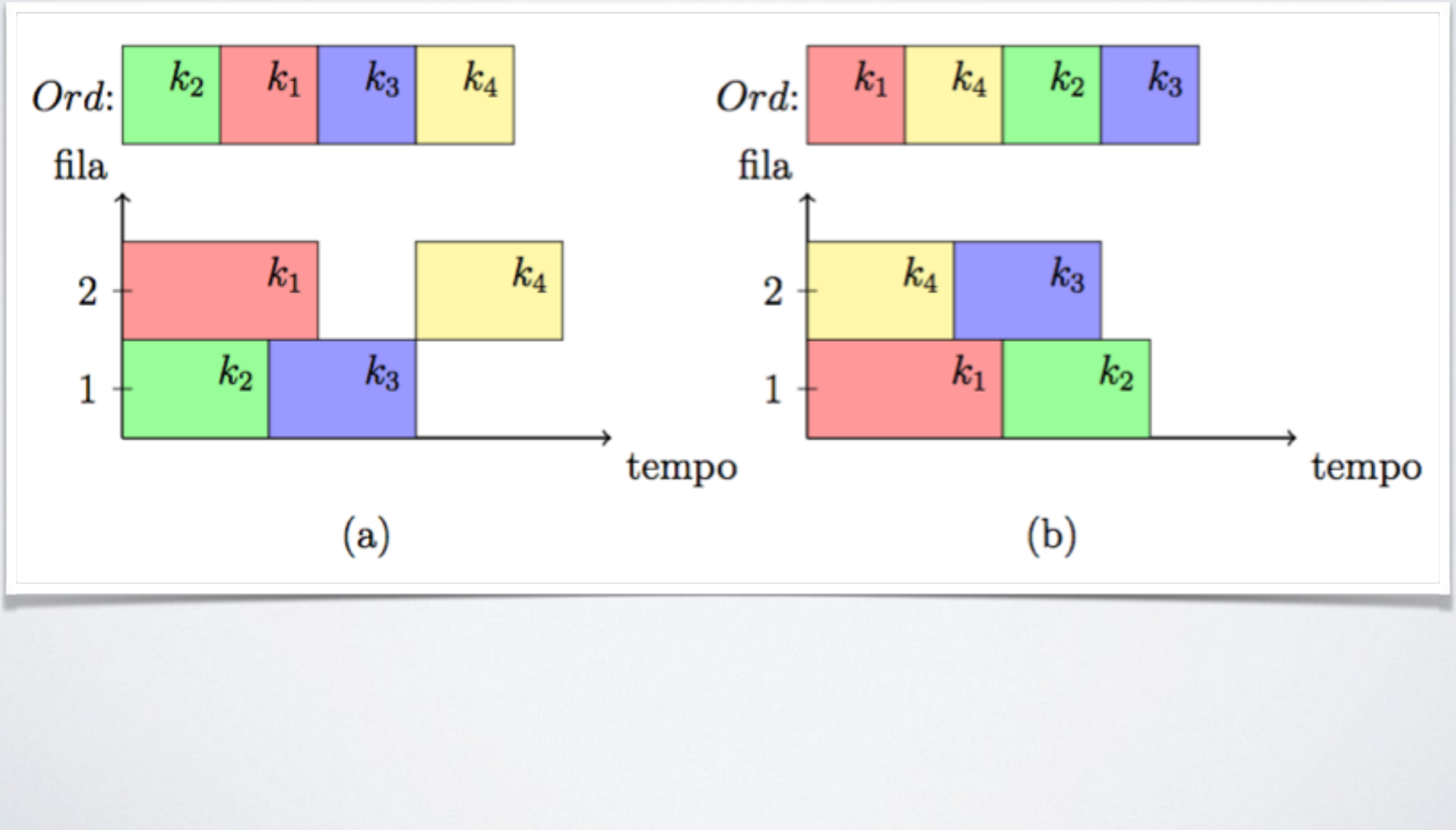


# PROCESSO DE ORDENAÇÃO



# PROCESSO DE ORDENAÇÃO



# PROCESSO DE ORDENAÇÃO

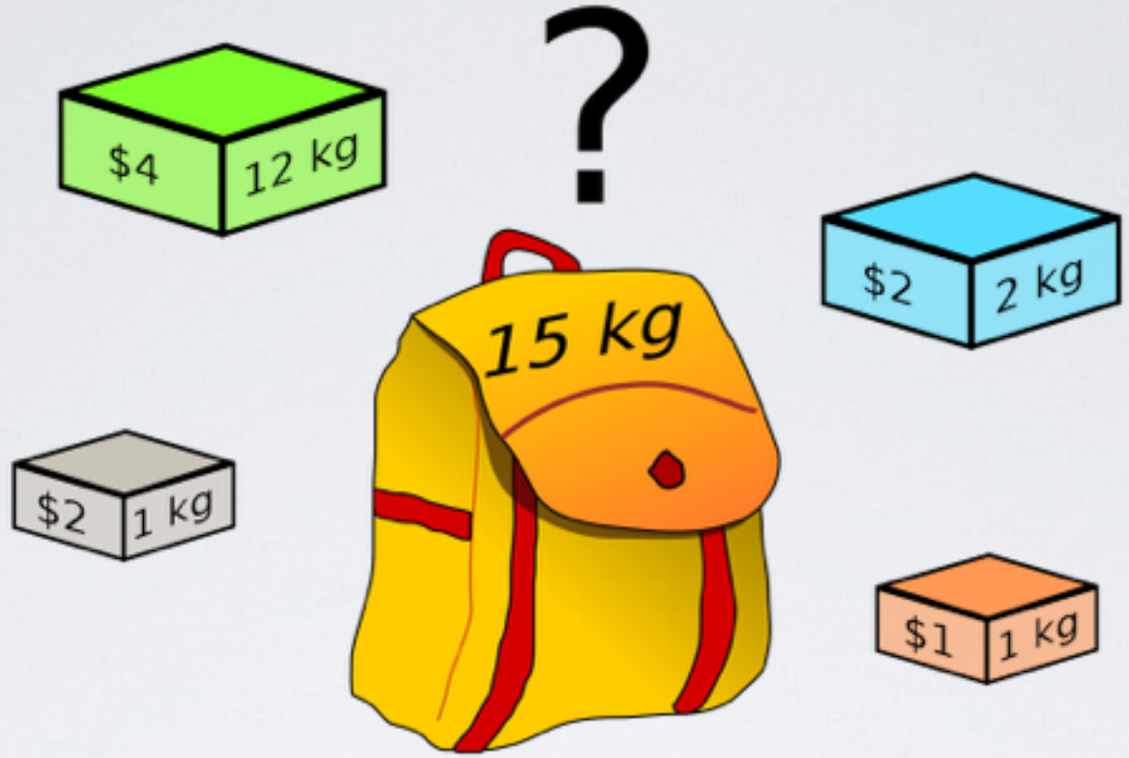
| Exemplo Didático | Kernels | | | |
|---|---|---|---|---|
| | $k_1$ | $k_2$ | $k_3$ | $k_4$ |
| Recursos $(w_i)$ | 30% | 30% | 50% | 60% |
| Tempo $(t_i)$ | $20ms$ | $15ms$ | $15ms$ | $15ms$ |

| $TO_{\alpha\beta}$ | $T_0$ | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $WAT_\alpha$ |
|---|---|---|---|---|---|---|
| $Ord_a$ | $60\% \times 15ms$ | $80\% \times 5ms$ | $50\% \times 10ms$ | $60\% \times 15ms$ | $0\% \times 0ms$ | 60% |
| $Ord_b$ | $90\% \times 15ms$ | $80\% \times 5ms$ | $80\% \times 10ms$ | $30\% \times 5ms$ | $0\% \times 0ms$ | 77% |

Tabela 3.2: Taxa de ocupação para cada instante $T_i$

$Ord$: $k_2$ $k_1$ $k_3$ $k_4$

$Ord$: $k_1$ $k_4$ $k_2$ $k_3$





# PROBLEMA DA MOCHILA

Busca pela solução de ordenação para otimizar a taxa de ocupação

# PROBLEMA DA MOCHILA

Ex: { 3, 4 } has value 40.

W = 11

| Item | Value | Weight | V/W |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 6 | 2 | 3 |
| 3 | 18 | 5 | 3.60 |
| 4 | 22 | 6 | 3.66 |
| 5 | 28 | 7 | 4 |

Greedy: repeatedly add item with maximum ratio $v_i / w_i$.

Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

# PROBLEMA DA MOCHILA

OPT(i,w) = valor máximo com i elementos e w de capacidade

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

# PROBLEMA DA MOCHILA

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

```
Input: n, w1,…,wN, v1,…,vN

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 1 to W
      if (wi > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi ]}

return M[n, W]
```

# PROBLEMA DA MOCHILA

W + 1 →

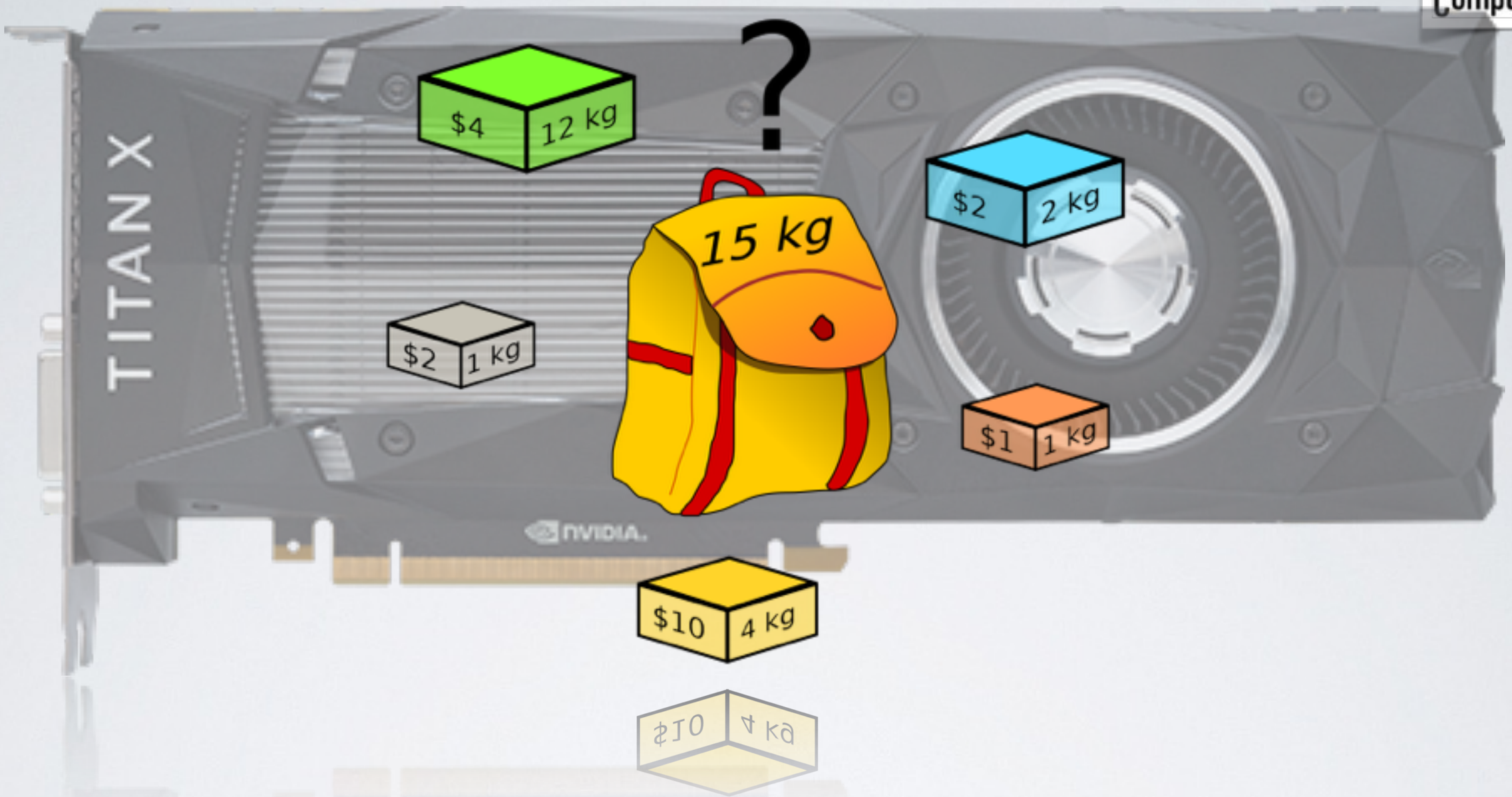
n + 1 ↓

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\phi$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| {1} | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| {1, 2} | 0 | 1 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| {1, 2, 3} | 0 | 1 | 6 | 7 | 7 | 18 | 19 | 24 | 25 | 25 | 25 | 25 |
| {1, 2, 3, 4} | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 24 | 28 | 29 | 29 | 40 |
| {1, 2, 3, 4, 5} | 0 | 1 | 6 | 7 | 7 | 18 | 22 | 28 | 29 | 34 | 34 | 40 |

OPT: { 4, 3 }
value = 22 + 18 = 40

W = 11

| Item | Value | Weight |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 6 | 2 |
| 3 | 18 | 5 |
| 4 | 22 | 6 |
| 5 | 28 | 7 |

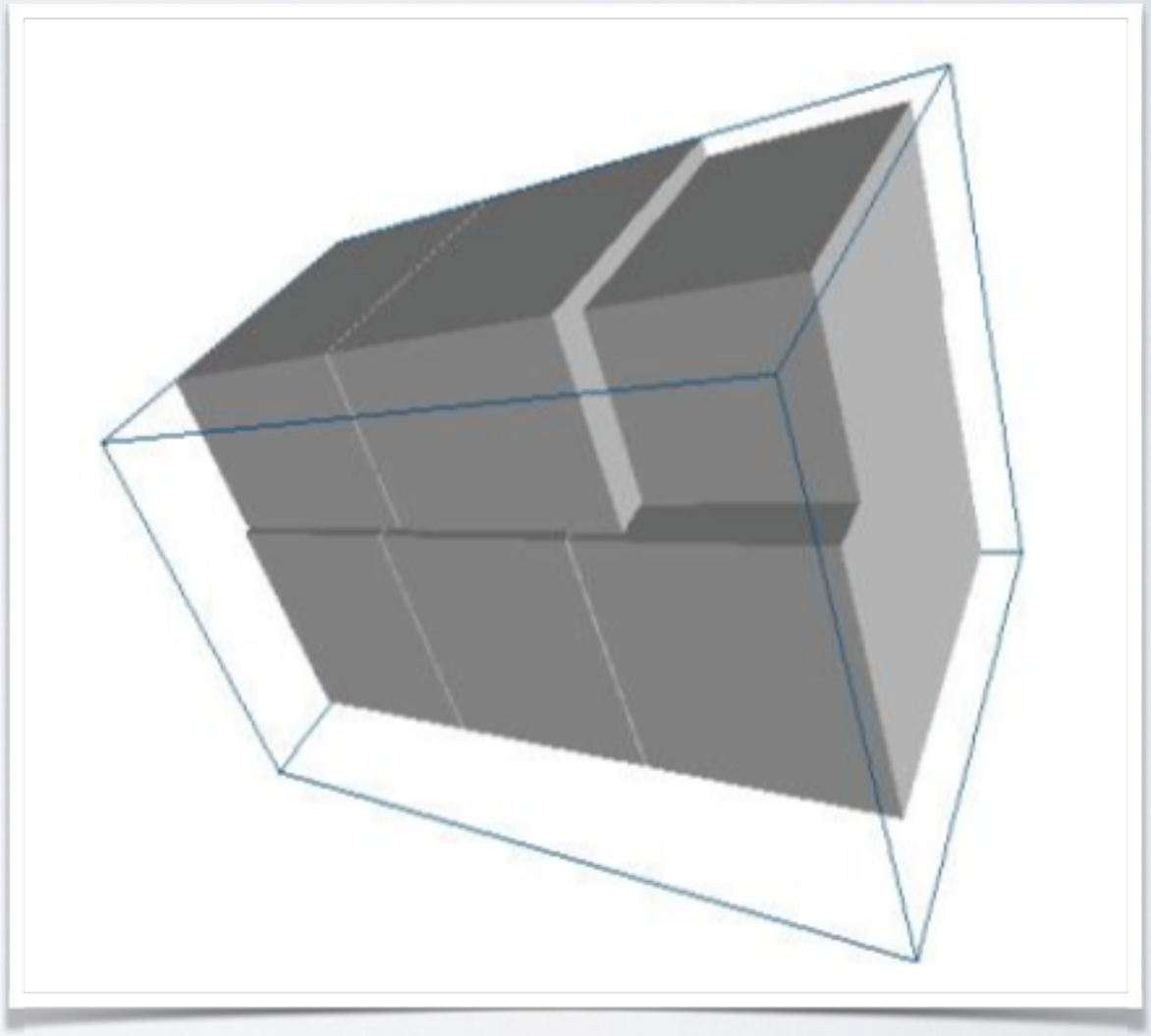



# ADAPTAÇÃO PARA GPU

## Adaptando o problema da mochila para GPU

# ADAPTAÇÃO PARA GPU

- Cada tarefa tem 3 recursos
  - Registrador
  - Memória Compartilhada
  - Threads
- Idealmente deveria usar o problema da mochila multi-dimensional. Os recursos foram linearizados como otimização do algoritmo
- O calculo do valor leva em consideração o tempo estimado



# ADAPTAÇÃO PARA GPU

| $k_i$ | $\overrightarrow{w_i}$ | $w_i$ | $v_i$ |
|---|---|---|---|
| $k_1$ | $(1, 1, 1)$ | 26 | 20 |
| $k_2$ | $(1, 2, 3)$ | 33 | 40 |
| $k_3$ | $(1, 0, 2)$ | 22 | 30 |
| $k_4$ | $(1, 3, 1)$ | 36 | 60 |
| $k_5$ | $(2, 1, 1)$ | 51 | 70 |

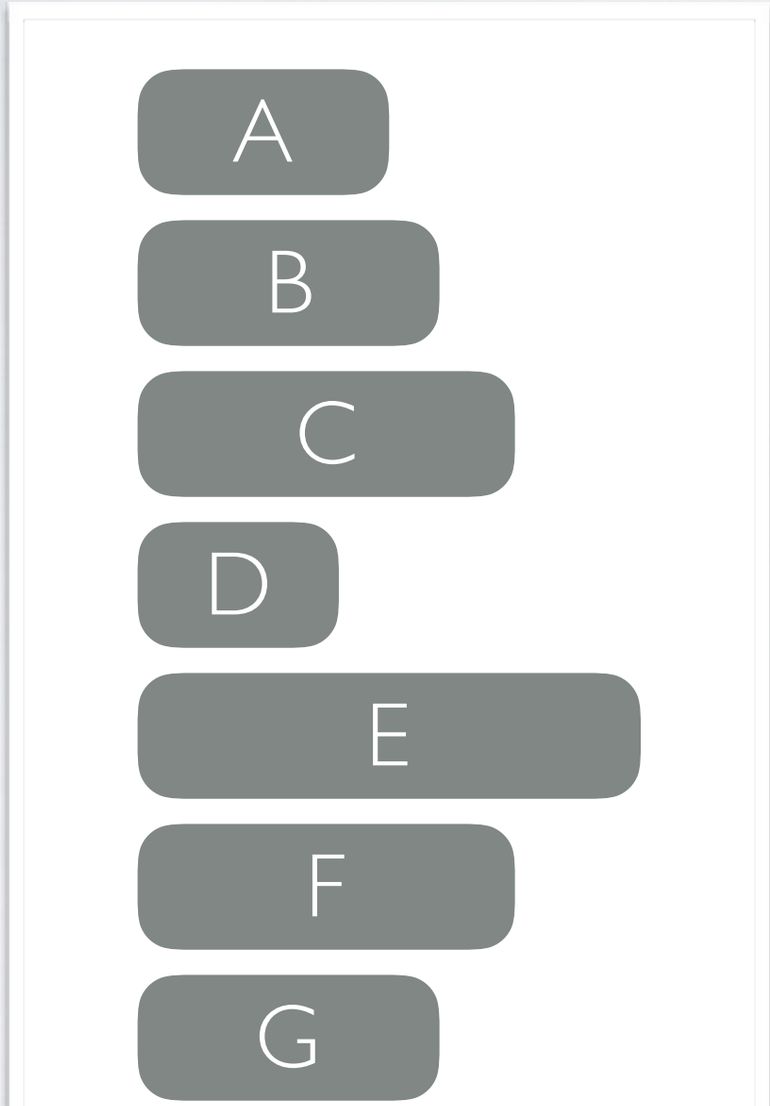
$$w_i = (w_{i1} \times (W_2 + 1) \times (W_3 + 1)) + (w_{i2} \times (W_2 + 1)) + w_{i3}$$

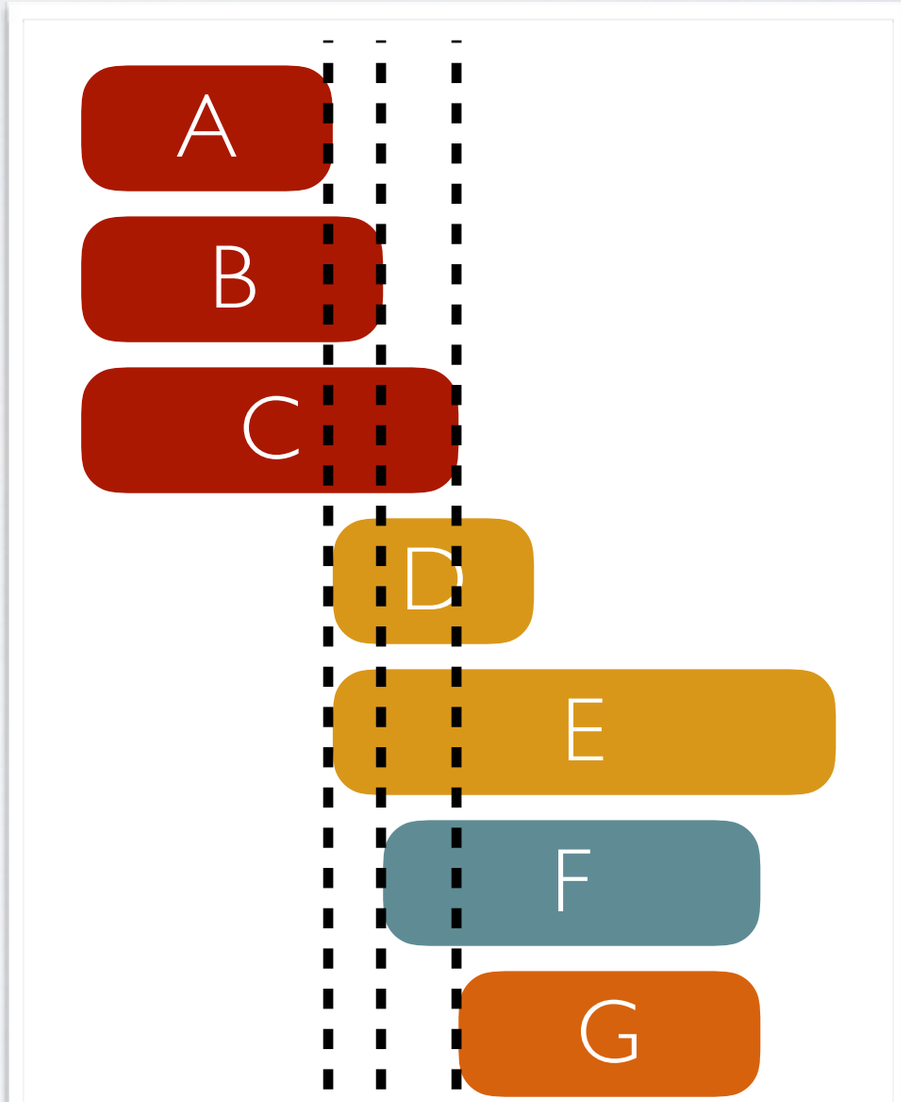
$$v_i = \frac{\frac{w_{i1}}{W_1} + \frac{w_{i2}}{W_2} + \frac{w_{i3}}{W_3}}{3 \times t_i^{est}}$$

# ALGORITMO DE ORDENAÇÃO

## Breve descrição do algoritmo de ordenação

# ALGORITMO DE ORDENAÇÃO



# ALGORITMO DE ORDENAÇÃO

**Algorithm 1** Algoritmo de ordenação dos Kernels

1: **function** KERNELREORDER($KernelList, W^{av}$)
2: **while** $KernelList$ not empty **do**
3: $NextSubmitList \leftarrow$ GETKERNELSTOSUBMIT($KernelList, W^{av}$)
4: **for each** $k_i \in NextSubmitList \wedge i \leq NQ$ **do**
5: $q_j \leftarrow$ FINDSHORTQUEUE($Q$)
6: $q_j \leftarrow q_j + k_i$
7: $W^{av} = W^{av} - w_i$
8: $KernelList = KernelList - k_i$
9: $q_j \leftarrow$ FINDSHORTQUEUE($Q$)
10: $k_e \leftarrow q_j.last$
11: $W^{av} = W^{av} + w_e$
12: **return** ROUNDROBINGCROSS($Q$)

# RESULTADOS

Resultados obtidos com o escalonamento

# METODOLOGIA

- Existem diversos benchmarks para CUDA na literatura, tais como Rodinia [9], Parboil [40] e CUDA SDK [28]

- Foi utilizado um benchmark sintético para realizar os testes com o objetivo de ter um maior controle sobre as execuções

# BENCHMARK SINTÉTICO

- Foi construído um gerador de kernels sintético
- O gerador constrói kernels com característica de número de registradores, memória compartilhada, número de threads e tempo estimado de execução
- Cada kernel gerado possui códigos para alocar os recursos e executar no tempo estimado

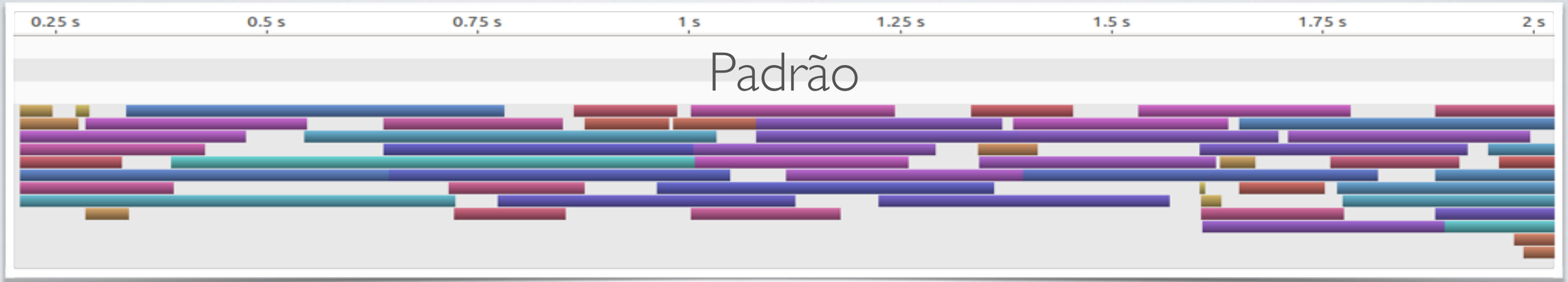
# BENCHMARK SINTÉTICO

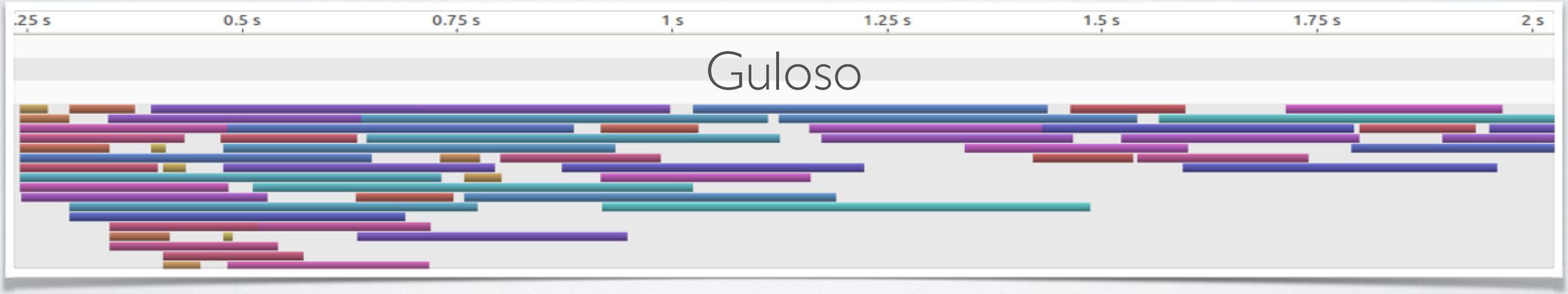
- Foram criados 256 kernels sintéticos através do gerador
- O número aleatório para os recursos seguiram o intervalo:
  - Registradores: [1-64K]
  - Memória compartilhada: [0-48KB]
  - Threads: [1-16K]
  - Tempo: [1-1000ms]

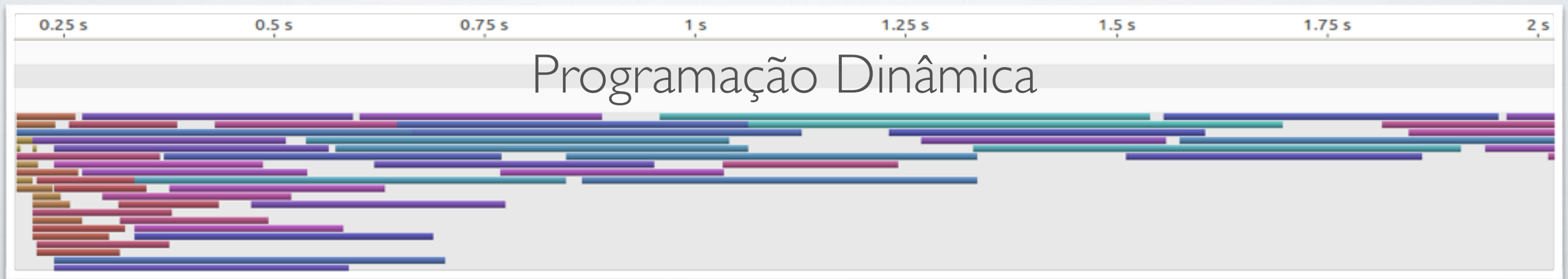
# TITAN X

| | TitanX | K40 |
|---|---|---|
| Número de Cores | 3,072 | 2,880 |
| Core Clock | 1000 MHz | 745 MHz |
| RAM | 24GB | 12GB |
| Memory Bandwidth | 336.5 GB/s | 288 GB/s |
| Capability | 5.2 | 3.5 |
| Número de SMs | 24 | 15 |
| Shared Memory por SM | 96KB | 48KB |
| Numero of Registers por SM | 64K | 64K |
| Max número de threads por SM | 2048 | 2048 |
| Arquitetura | Maxwell | Kepler |
| Pico de desempenho | 6.6 TFLOPs | 4.2 TFLOPS |

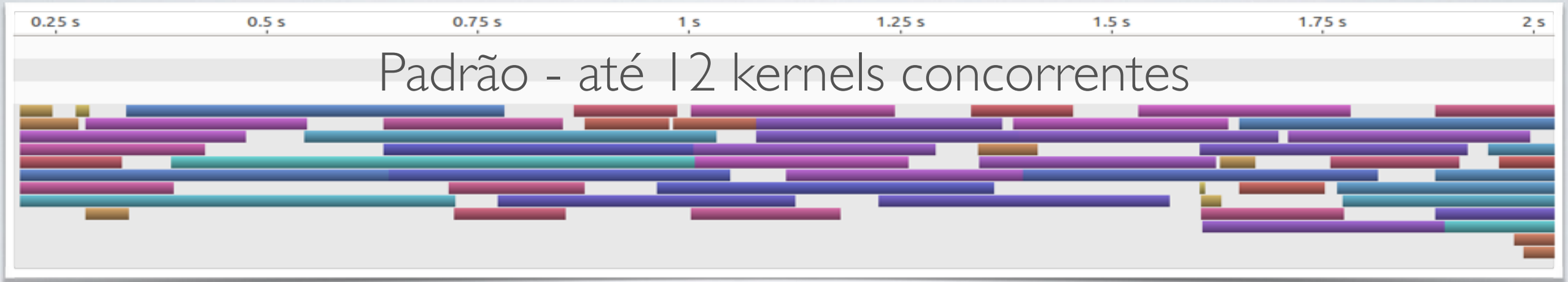
# TITAN X - PROFILE

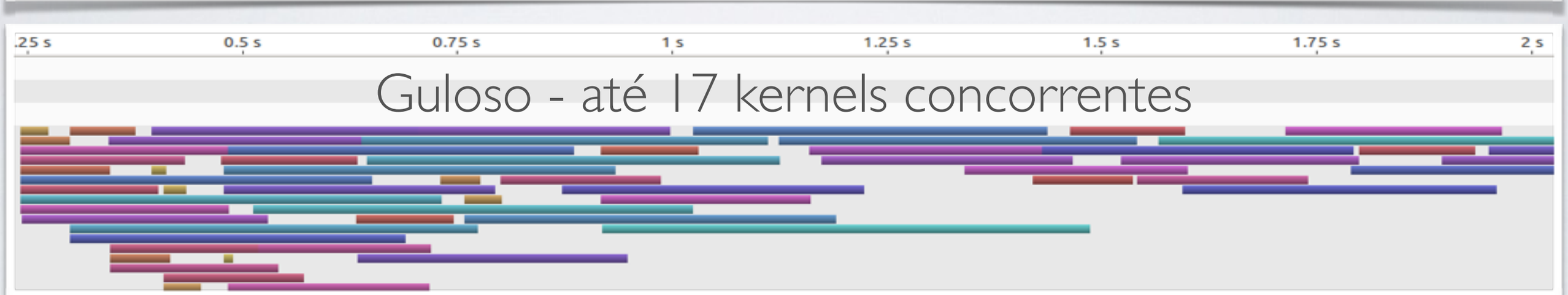


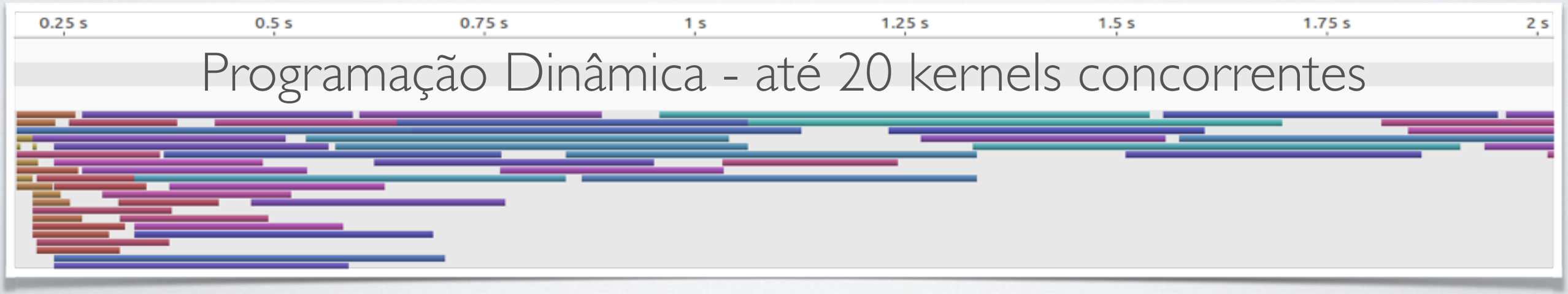



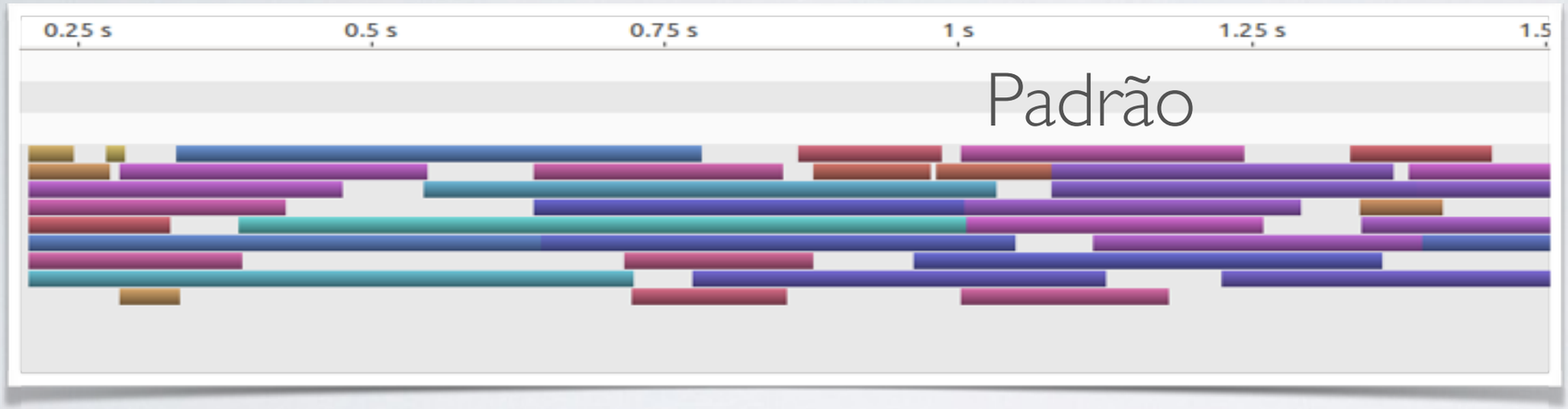
# TITAN X - PROFILE



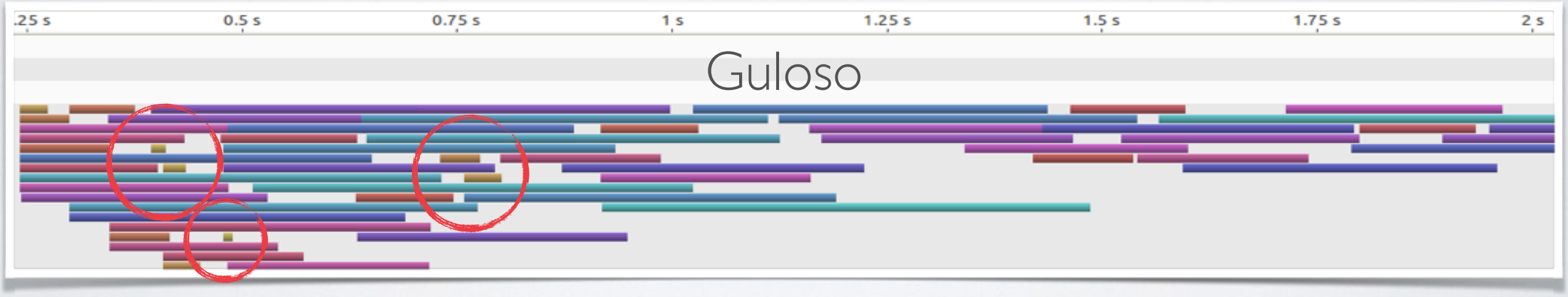



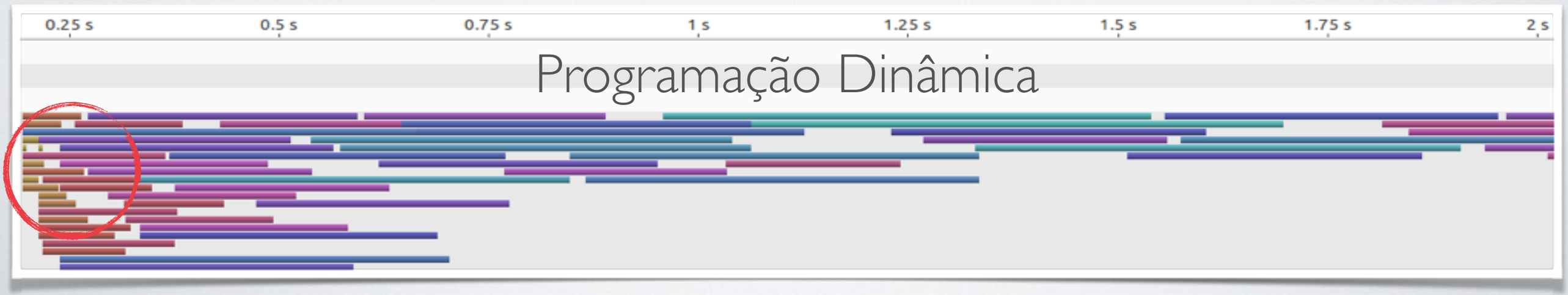



# TITAN X - PROFILE

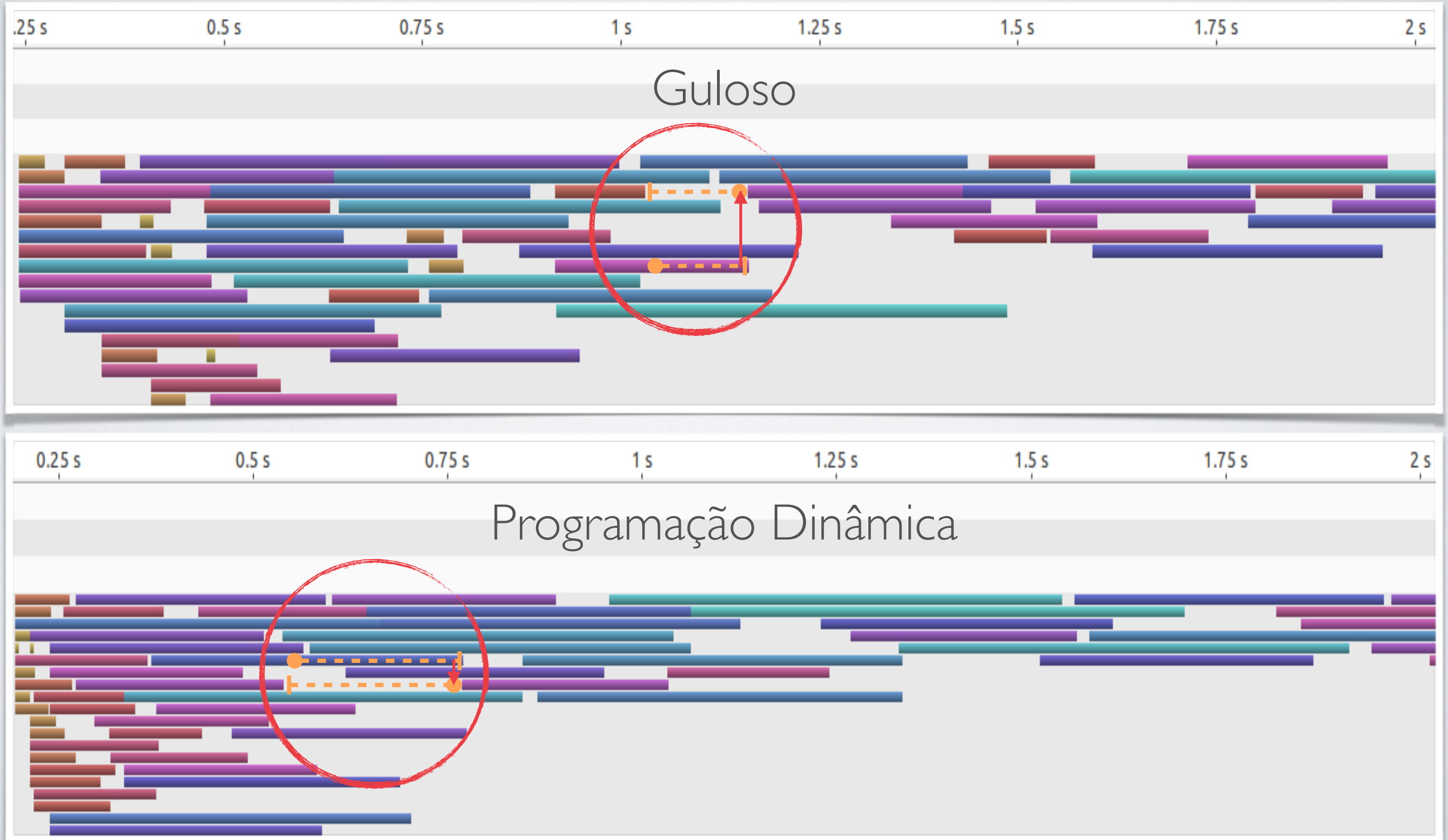



$$v_i = \frac{\frac{w_{i1}}{W_1} + \frac{w_{i2}}{W_2} + \frac{w_{i3}}{W_3}}{3 \times t_i^{est}}$$

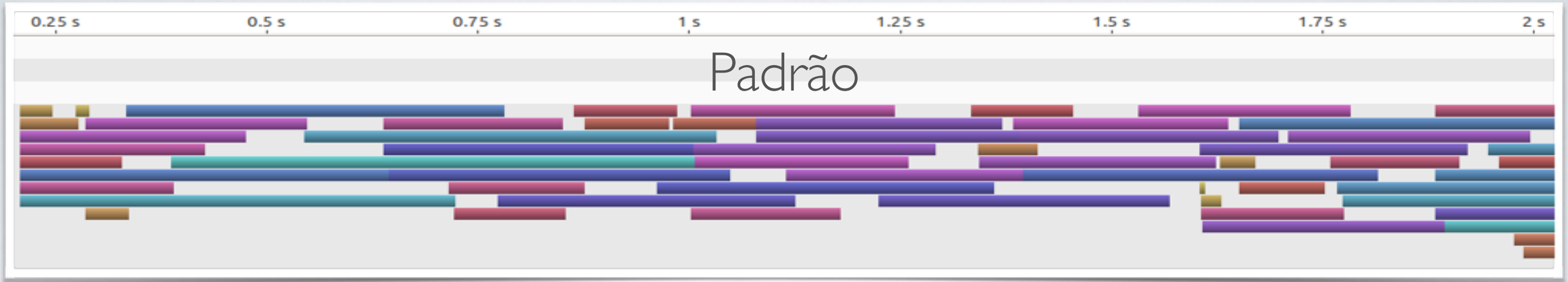


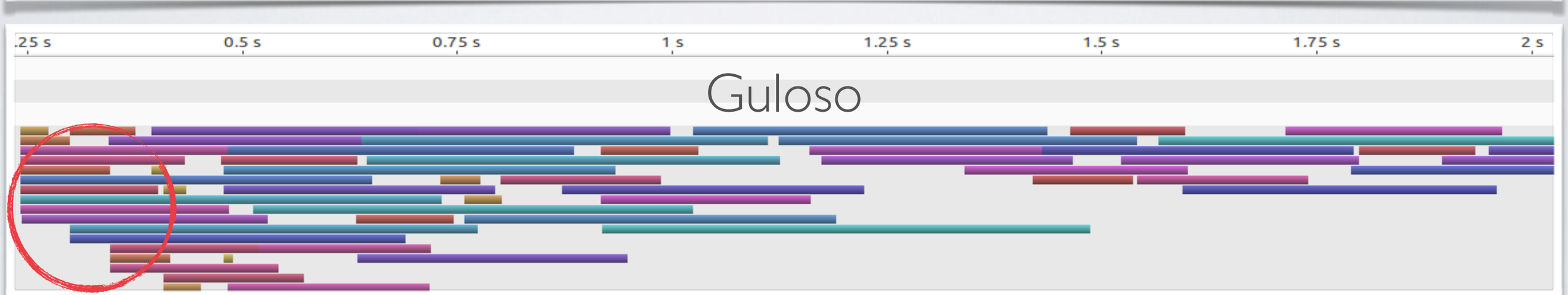



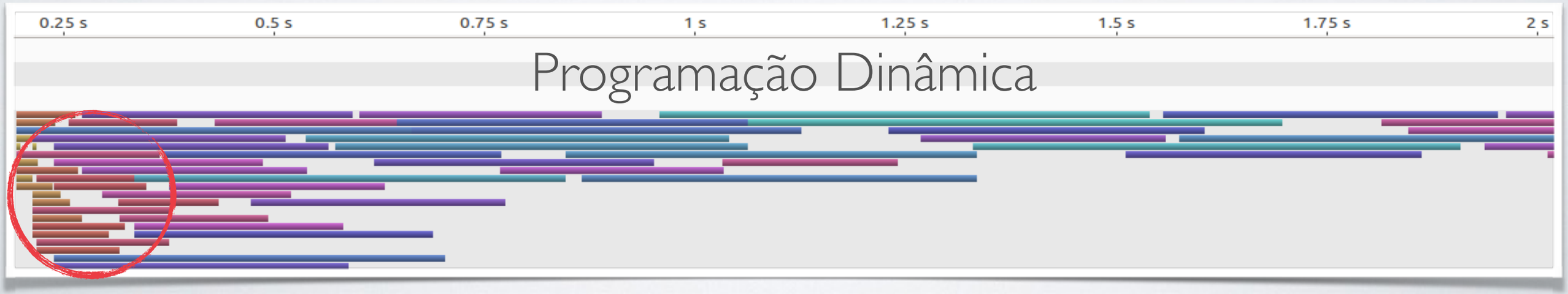
# TITAN X - PROFILE



# TITAN X - PROFILE



# ANTT

- Média normalizada da razão entre o tempo "turnaround time" concorrente com o não concorrente

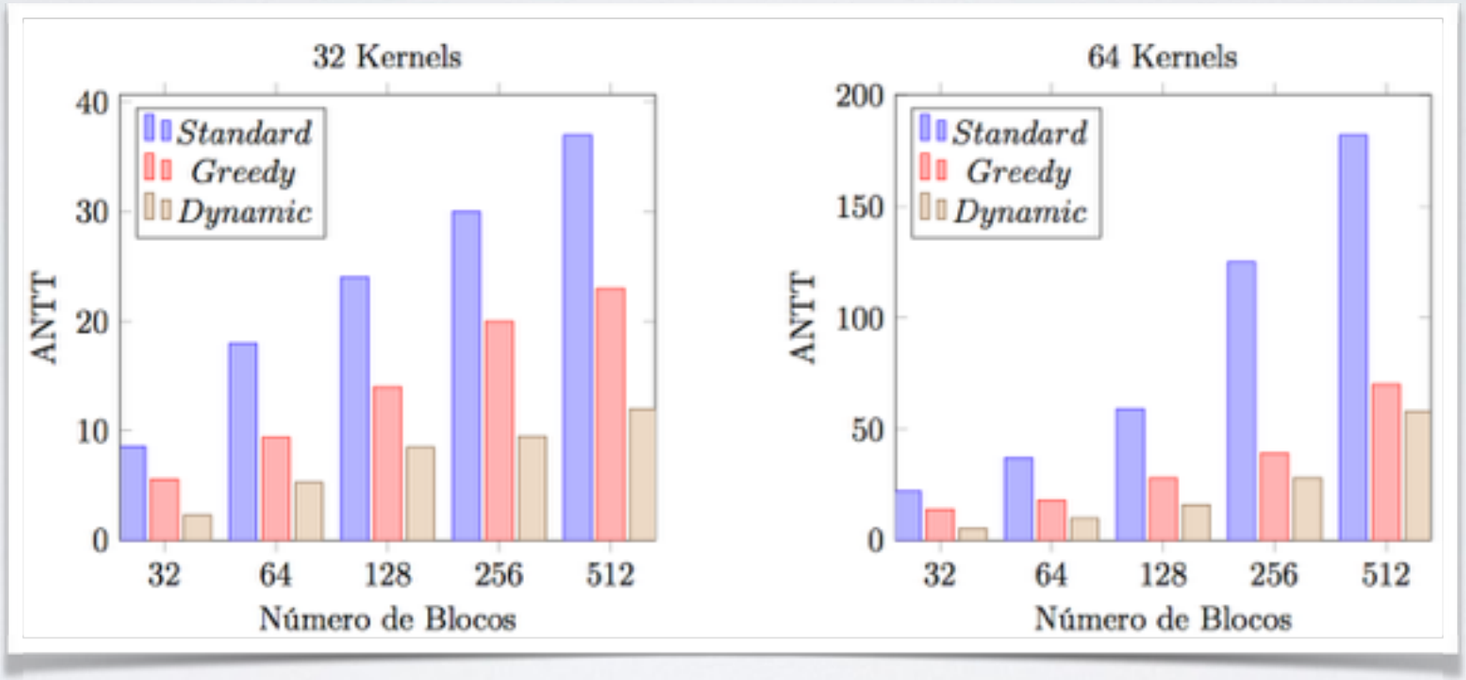
- Valor entre 1 e N

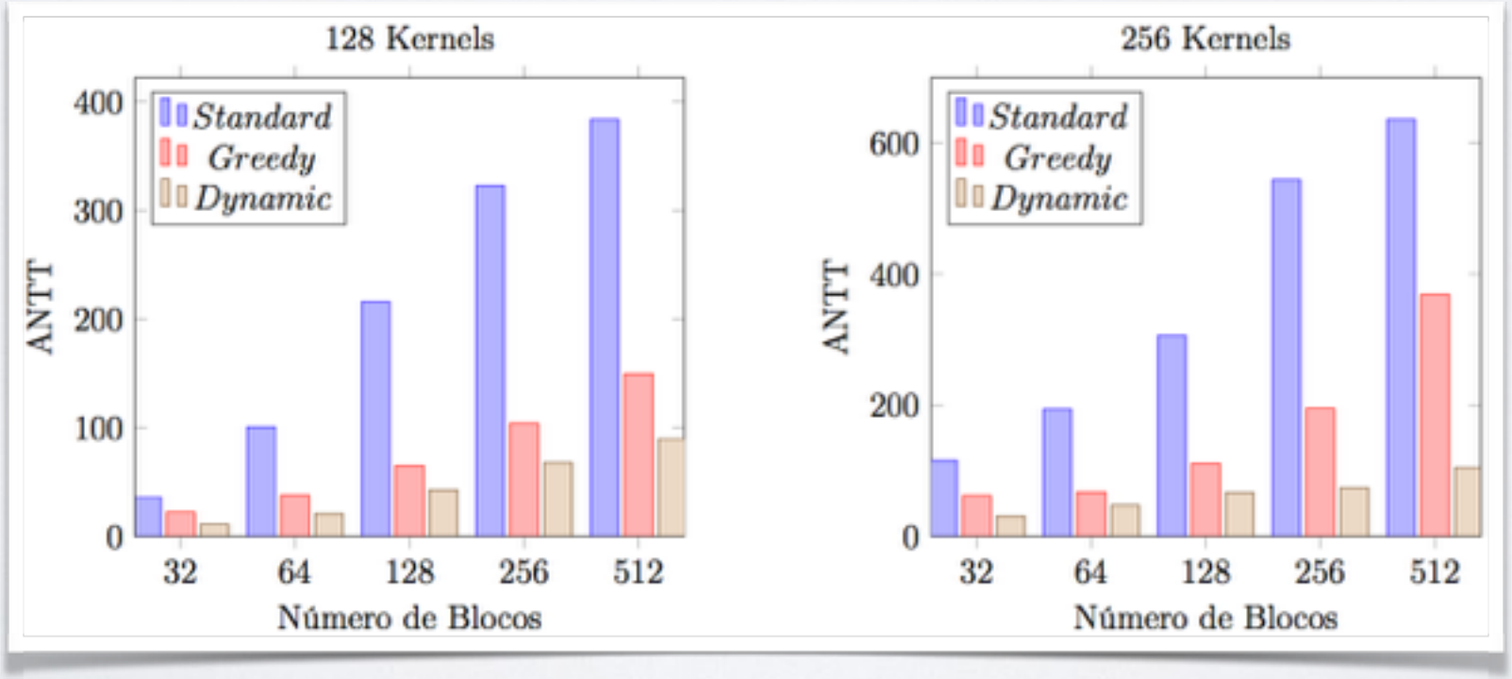
- Quanto menor, melhor

- Valor 1 quando a concorrência não interfere e N quando interfere muito

$$ANTT = \frac{1}{N} \sum_{i=1}^{N} \frac{TT_i}{TT_i^{Alone}}$$

# TITAN X - ANTT


32 Kernels
64 Kernels
Standard
Greedy
Dynamic
ANTT
Número de Blocos
0
10
20
30
40
50
100
150
200
32
64
128
256
512

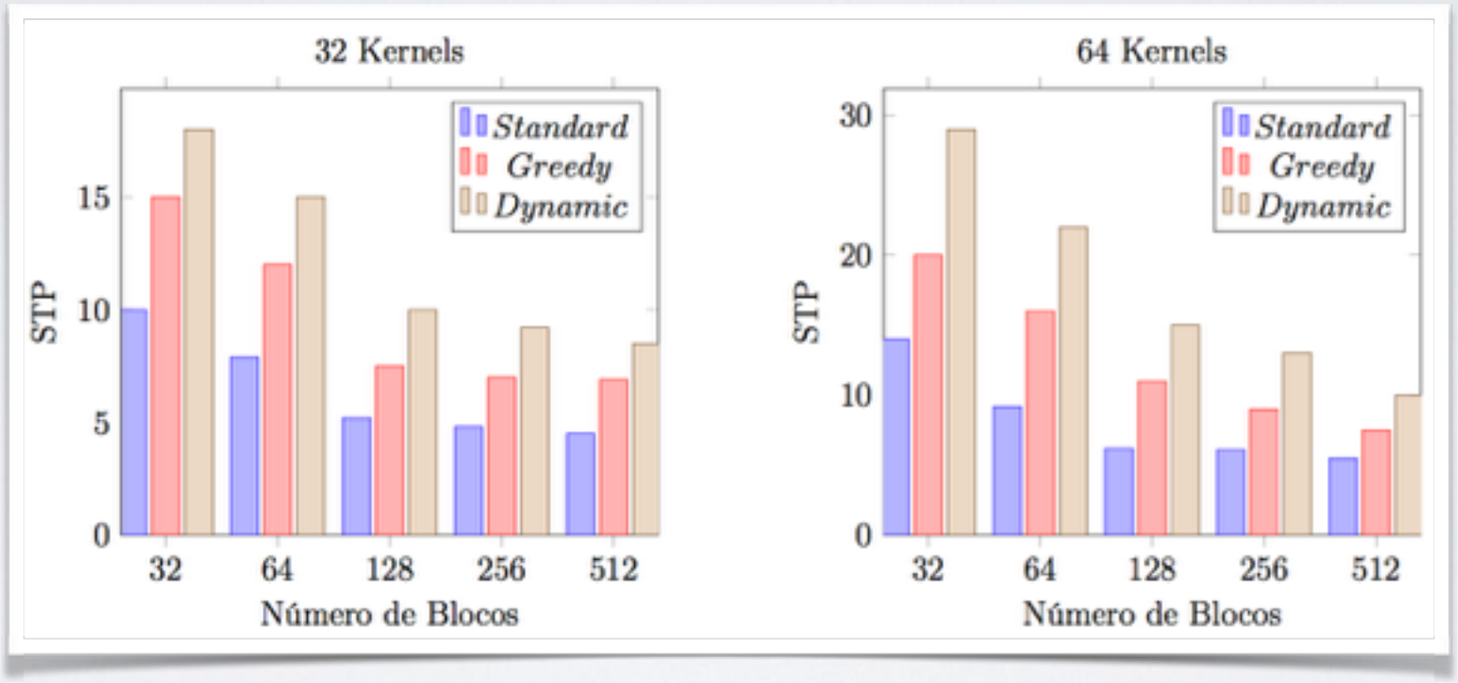

128 Kernels
256 Kernels
Standard
Greedy
Dynamic
ANTT
Número de Blocos
0
100
200
300
400
600
32
64
128
256
512

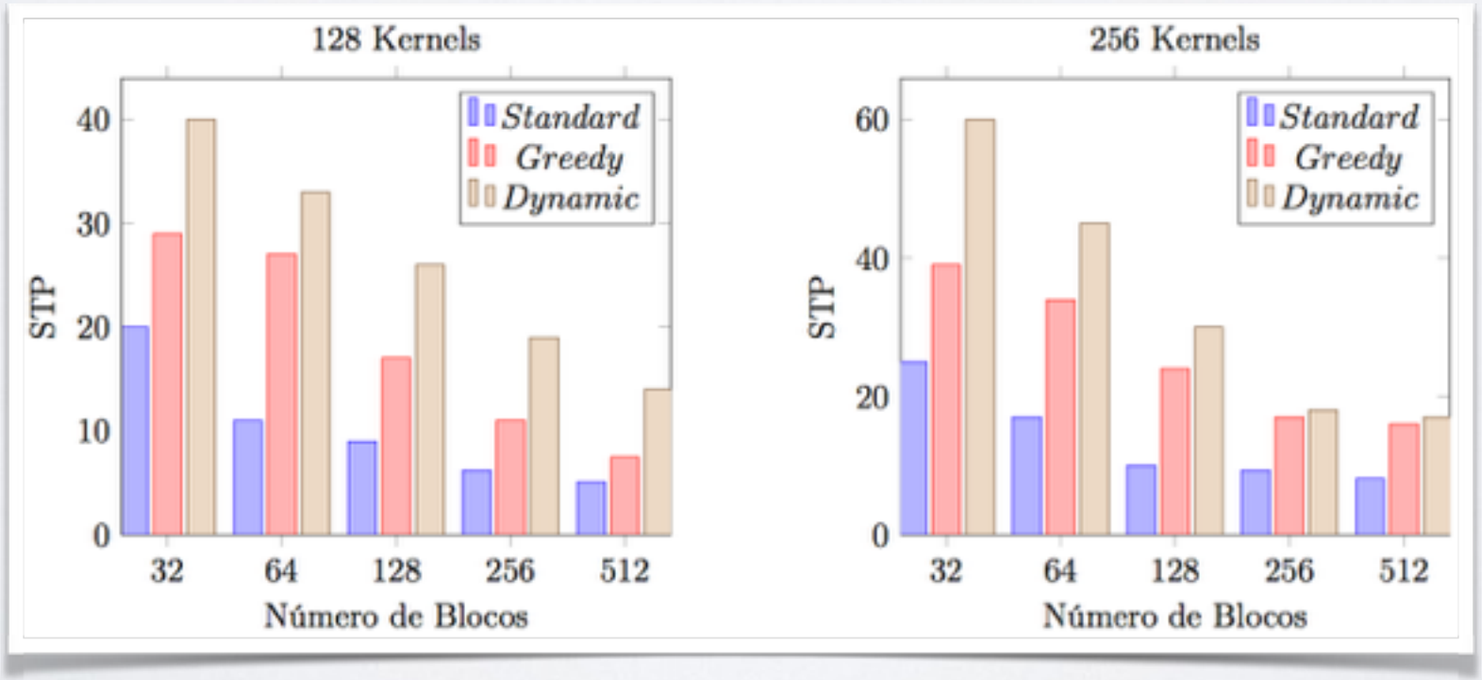
# STP

- Determina o número de tarefas completadas por unidade de tempo
- Valor entre 1 e N
- Quanto maior, melhor

$$STP = \sum_{i=1}^{N} \frac{TT_i^{Alone}}{TT_i}$$

# TITAN X - STP

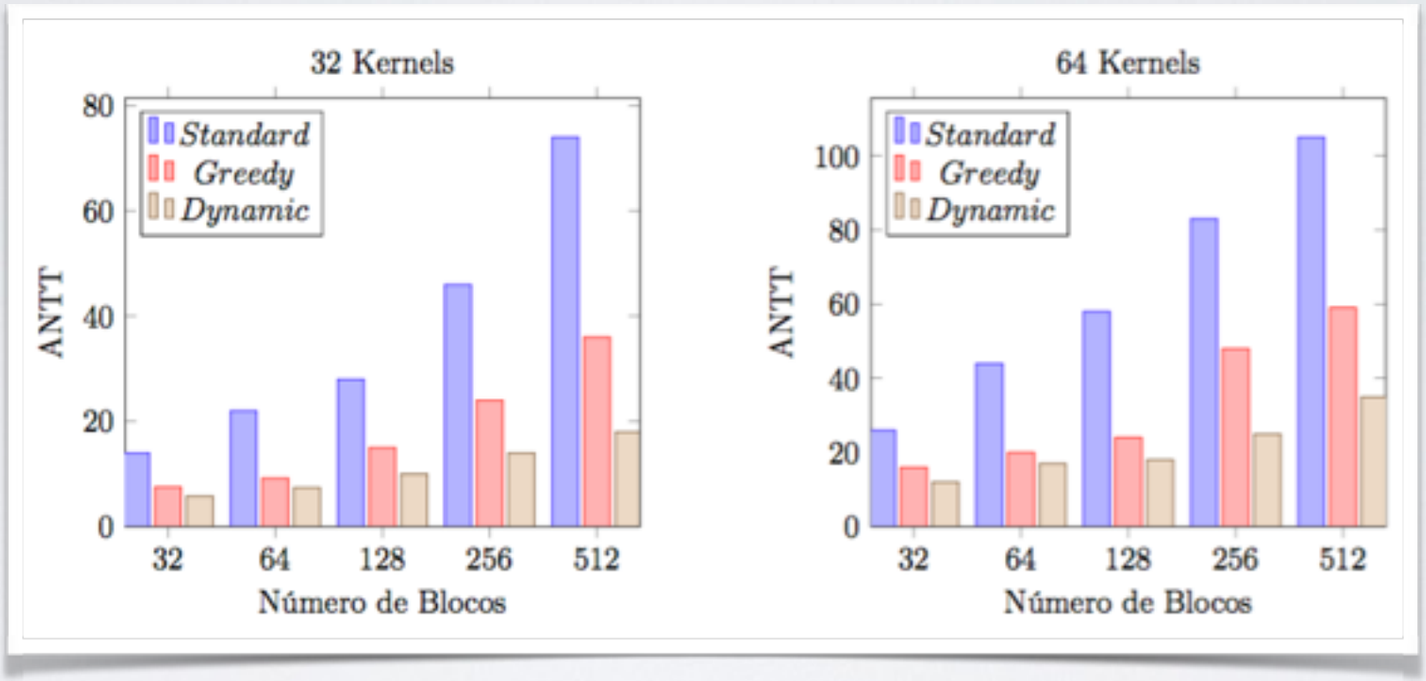

32 Kernels
64 Kernels
Standard
Greedy
Dynamic
STP
Número de Blocos
0
5
10
15
20
30
32
64
128
256
512

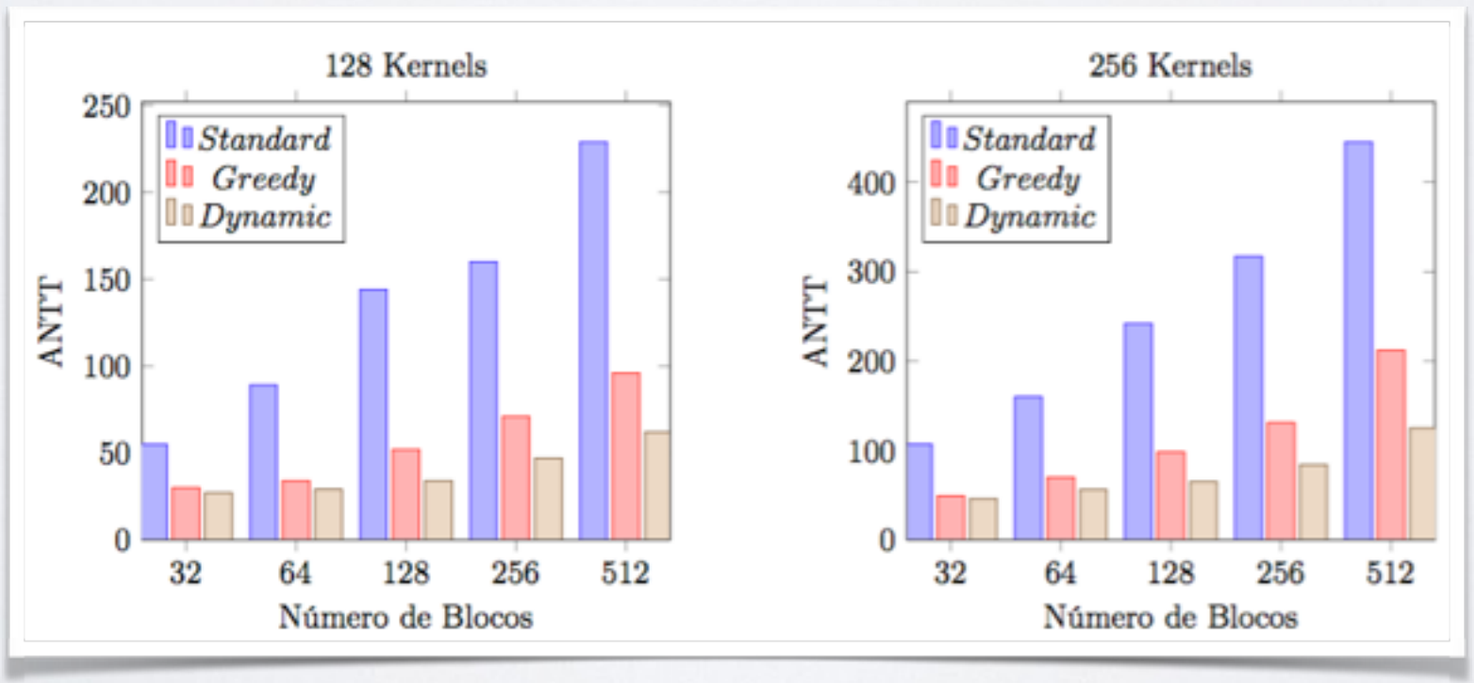

128 Kernels
256 Kernels
Standard
Greedy
Dynamic
STP
Número de Blocos
0
10
20
30
40
60
32
64
128
256
512

# K40

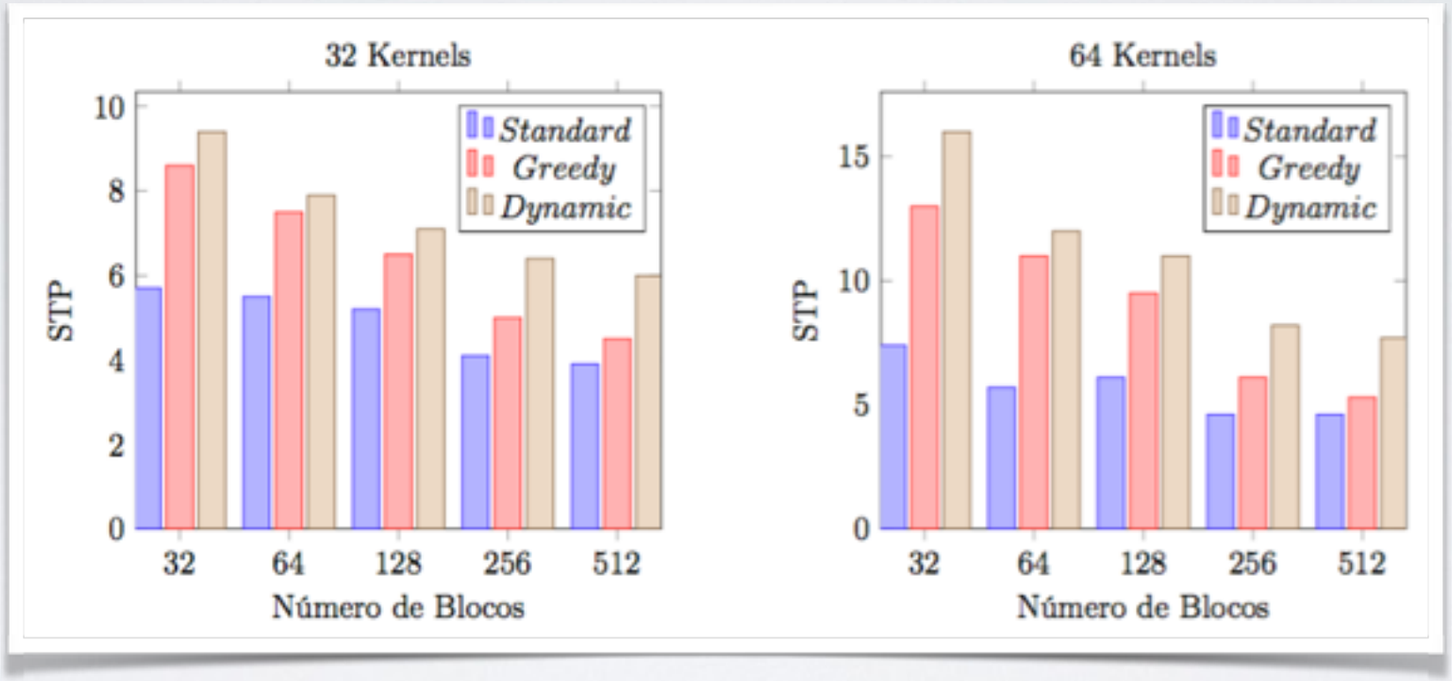
| | **TitanX** | **K40** |
|---|---|---|
| Número de Cores | 3,072 | 2,880 |
| Core Clock | 1000 MHz | 745 MHz |
| RAM | 24GB | 12GB |
| Memory Bandwidth | 336.5 GB/s | 288 GB/s |
| Capability | 5.2 | 3.5 |
| Número de SMs | 24 | 15 |
| Shared Memory por SM | 96KB | 48KB |
| Numero of Registers por SM | 64K | 64K |
| Max número de threads por SM | 2048 | 2048 |
| Arquitetura | Maxwell | Kepler |
| Pico de desempenho | 6.6 TFLOPs | 4.2 TFLOPS |

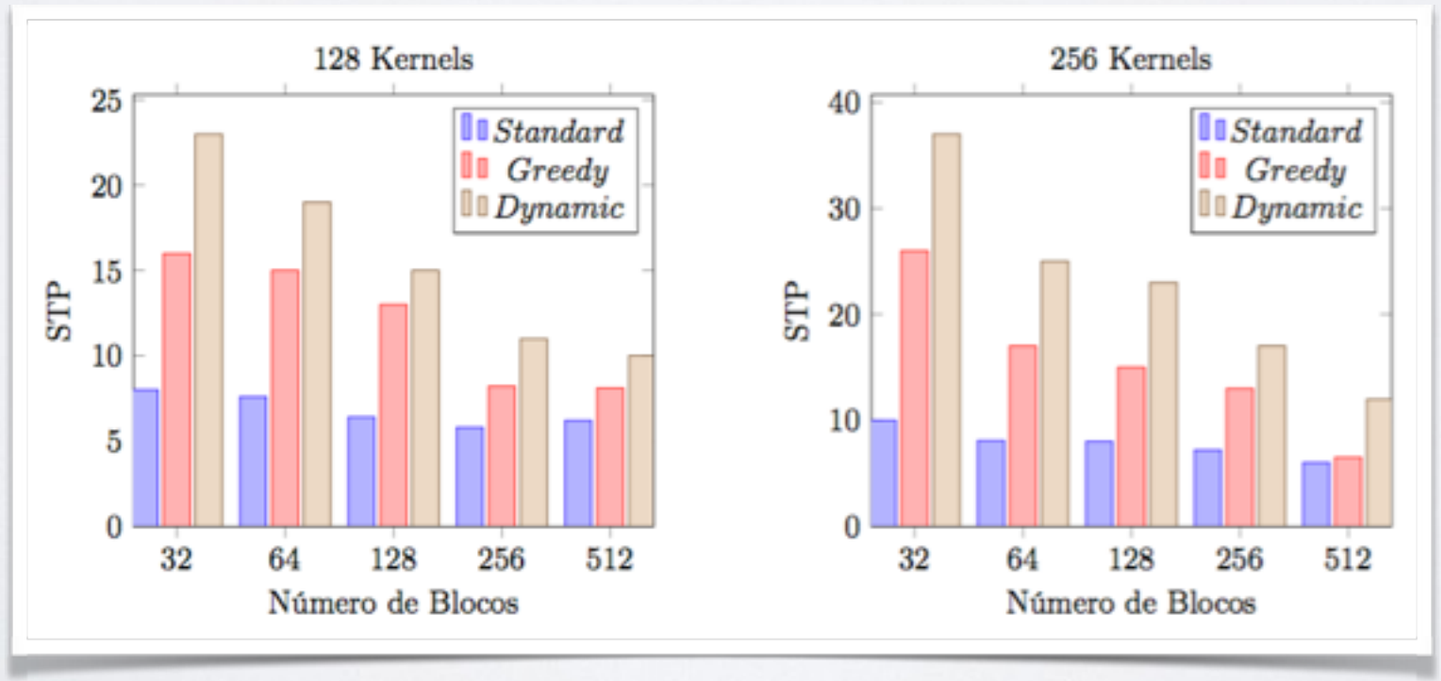
# K40 - ANTT


32 Kernels
64 Kernels
Standard
Greedy
Dynamic
ANTT
Número de Blocos
0
20
40
60
80
100
32
64
128
256
512


128 Kernels
256 Kernels
Standard
Greedy
Dynamic
ANTT
Número de Blocos
0
50
100
150
200
250
300
400
32
64
128
256
512

# K40 - STP


32 Kernels
STP
Número de Blocos
Standard
Greedy
Dynamic
0
2
4
6
8
10
32
64
128
256
512
64 Kernels
STP
Número de Blocos
Standard
Greedy
Dynamic
0
5
10
15
32
64
128
256
512


128 Kernels
STP
Número de Blocos
Standard
Greedy
Dynamic
0
5
10
15
20
25
32
64
128
256
512
256 Kernels
STP
Número de Blocos
Standard
Greedy
Dynamic
0
10
20
30
40
32
64
128
256
512


# POR FIM

Finalizando com algumas observações

# TRABALHOS FUTUROS

- Utilizar benchmarks não sintéticos para testar a ordenação
- A solução do problema da mochila através da programação dinâmica poderia usar multi-dimensional
- Estudar uma possibilidade, talvez em versões futuras da arquitetura da GPU, de determinar em quais SMs os blocos dos kernels serão executados
    - Através desse estudo concluído, o problema da mochila pode virar um problema de múltiplas mochila multi-dimensional
- Estudar uma forma de calcular o tempo estimado dos kernels para que possam ser executados no algoritmo de ordenação
- Estudar como comparar os resultados com o "estado da arte"

Obrigado

Bernardo Breder