# Padrões de programação paralela aplicados às arquiteturas híbridas

Conceitos, ferramentas e estudo de casos

**Marcos Barreto**

LaSiD / DCC / UFBA
marcoseb@dcc.ufba.br

**Centro Universitário La Salle (UNILASALLE)**
Porto Alegre, 22 de março de 2013

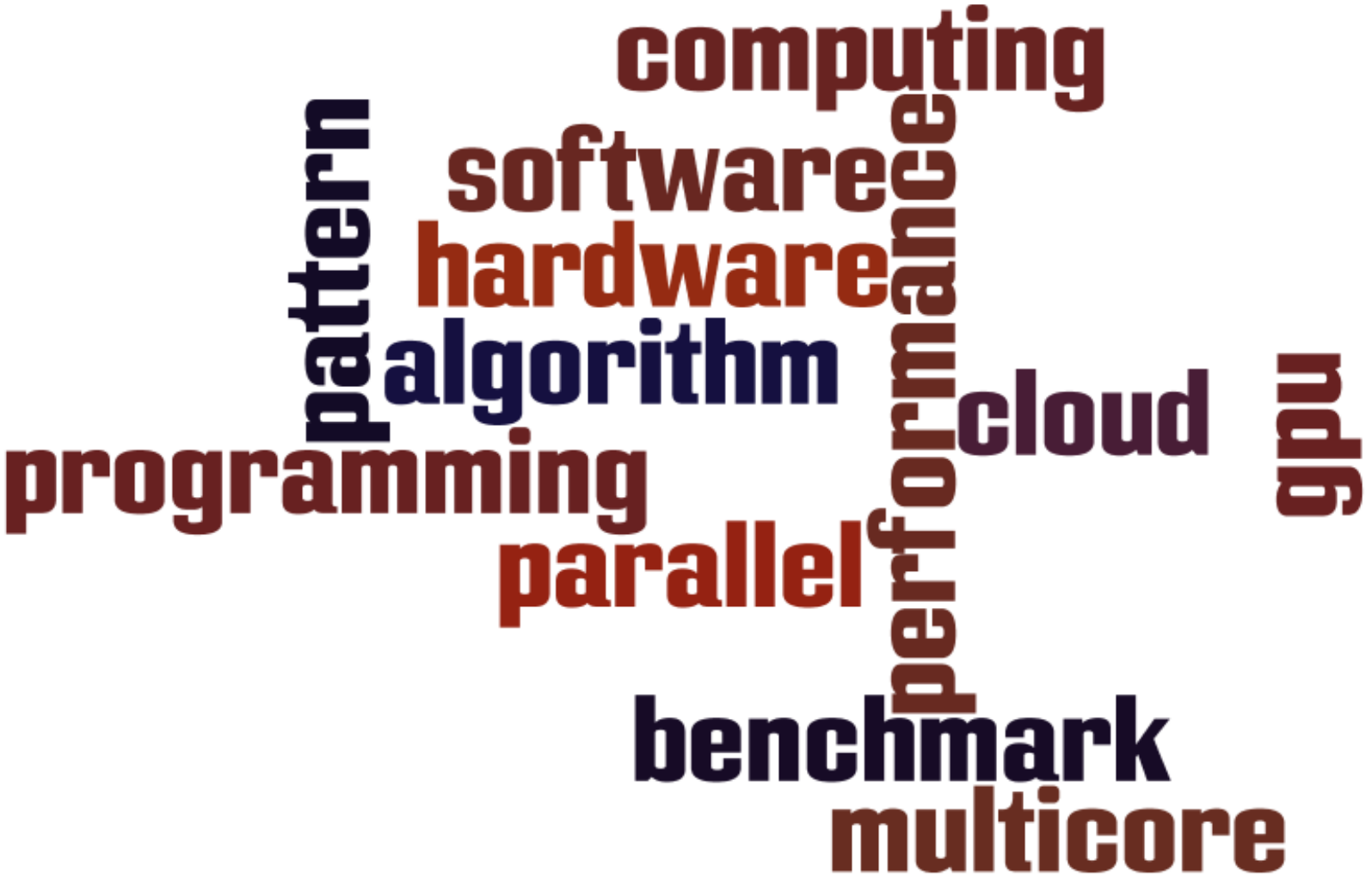
# Parceiros e patrocinadores

- **Análise de modelos de desempenho para arquiteturas híbridas**
  - UFBA, UNEB, UNIVASF, UFPE, Univ. Poli. de Valência, Univ. de Múrcia e Univ. Autônoma de Barcelona
  - Fomento: UFBA (Permanecer 2011/2012), Rede CoCADA, TIN2012-38342-C04-03, PROMETEO/2009/2013 e CAPAP-H
- **Minha cloud científica ($mc^2$)**
  - LNCC (Antônio Tadeu), UFCG (Francisco Brasileiro), UFBA, PUC-RJ, CESUP/UFRGS, CENAPAD-CE, FIOCRUZ
  - Fomento: MCT/RNP – Grupos de Trabalho 2011 – 2013
- **Work. Group on Ontologies for Robotics and Automation**
  - NIST (Craig Schlenoff), UFRGS (Edson Prestes), UFBA, UNIGE, POLIMI (Itália), Univ. Lisboa (Portugal), A.U. of Cairo (Egito), LISSI, CEA LIST (França), Monash (Malásia), Univ. of New Brunswick (Canadá), Óbuda Univ. (Hungria).
  - Fomento: IEEE RAS/SC – Project P1872 (2011 – 2015)

# Roteiro

1. **Contextualização**
   - Arquiteturas híbridas e programação paralela
2. **Padrões para programação paralela**
   - Ferramentas
   - Estudo de casos
3. **Computação em nuvem e padrões**
   - Padrões de oferta e de desenvolvimento
   - Estudo de casos
4. **Síntese e discussão**

# Contextualização

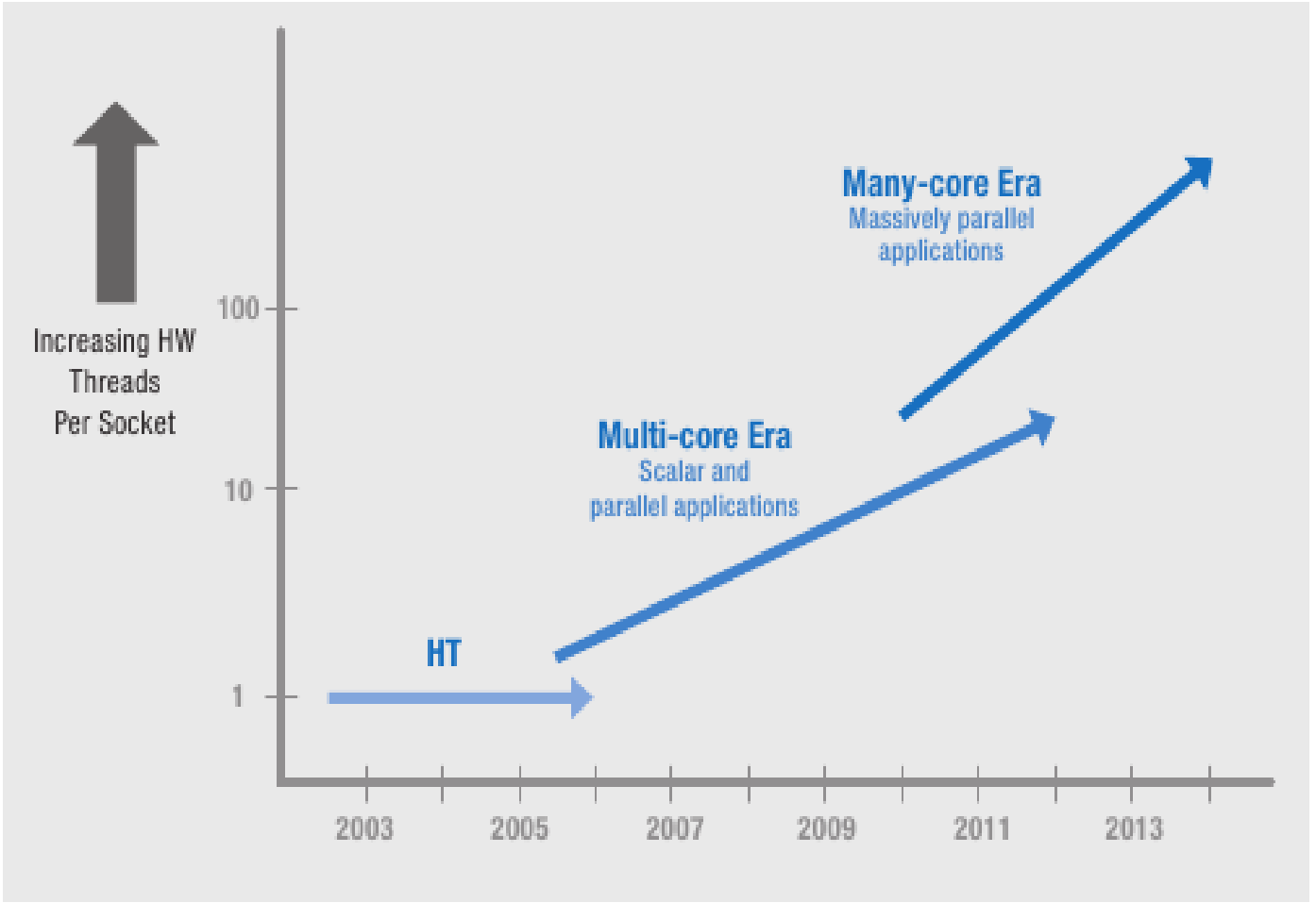


- “Hardware sempre foi paralelo...”
  - ...mas, em geral, programado de forma sequencial.

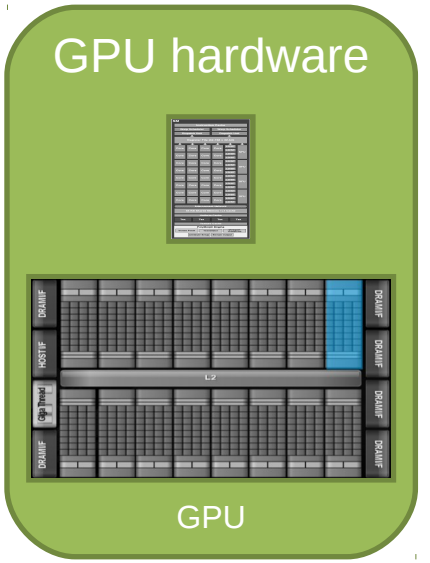
- Esforço intenso dos projetistas em prover arquiteturas (processadores) capazes de transformar algoritmos sequenciais em algoritmos paralelos.
  - Pipelining, processadores superescalares, compiladores “paralelizadores”
  - Execução especulativa, reordenação, *prefetch*
  - Foco na exploração do paralelismo em nível de instrução (ILP) e de threads (TLP)

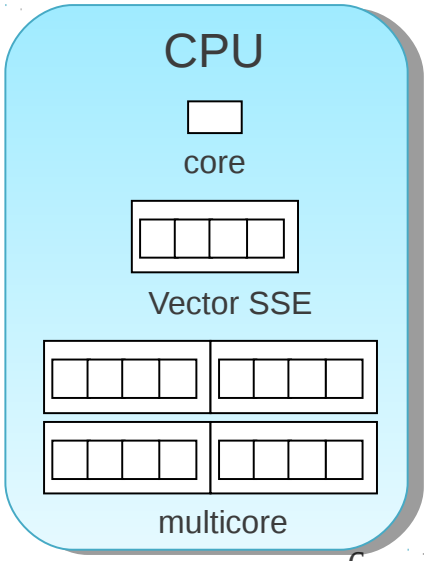
É possível estabelecer, de forma automática, um algoritmo mais adequado à execução paralela a partir de um algoritmo sequencial?

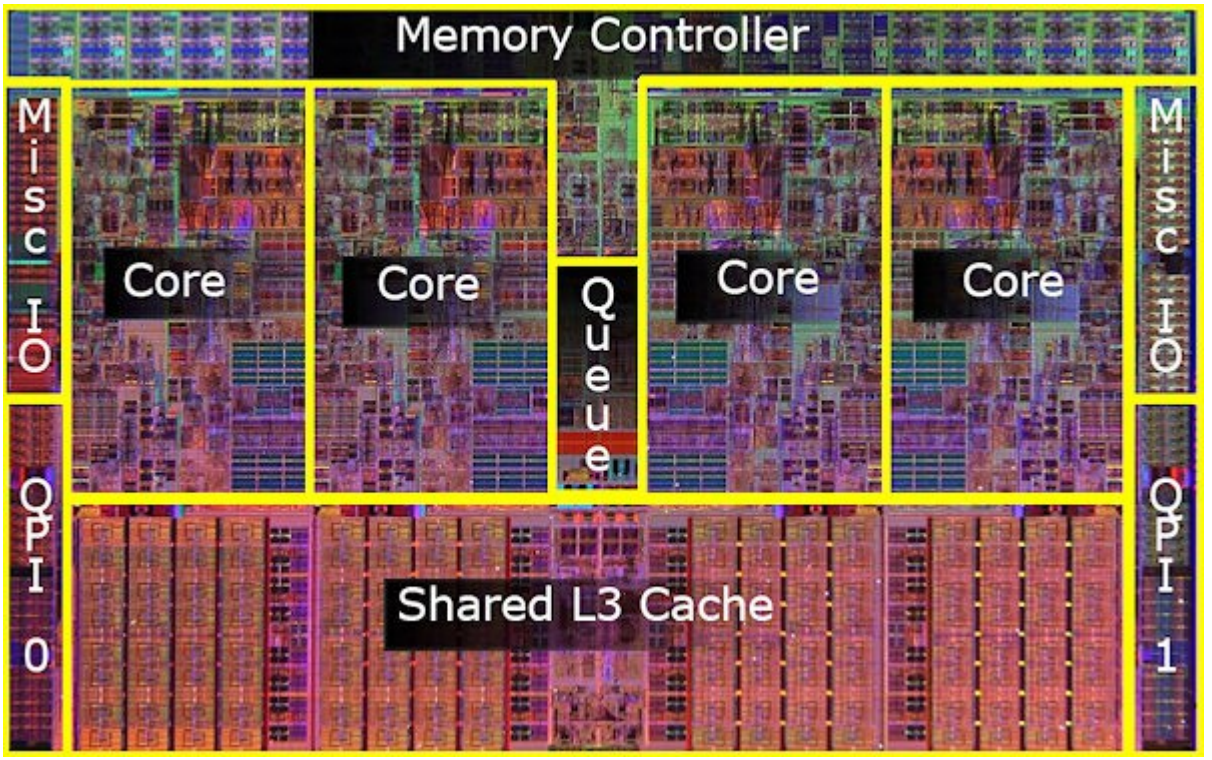
- Sistemas atuais são arquiteturas híbridas:
  - Múltiplos processadores, coprocessadores paralelos, multicore, SIMD: processadores vetoriais; extensões MMX, SSE e AVX; unidades gráficas (GPUs)

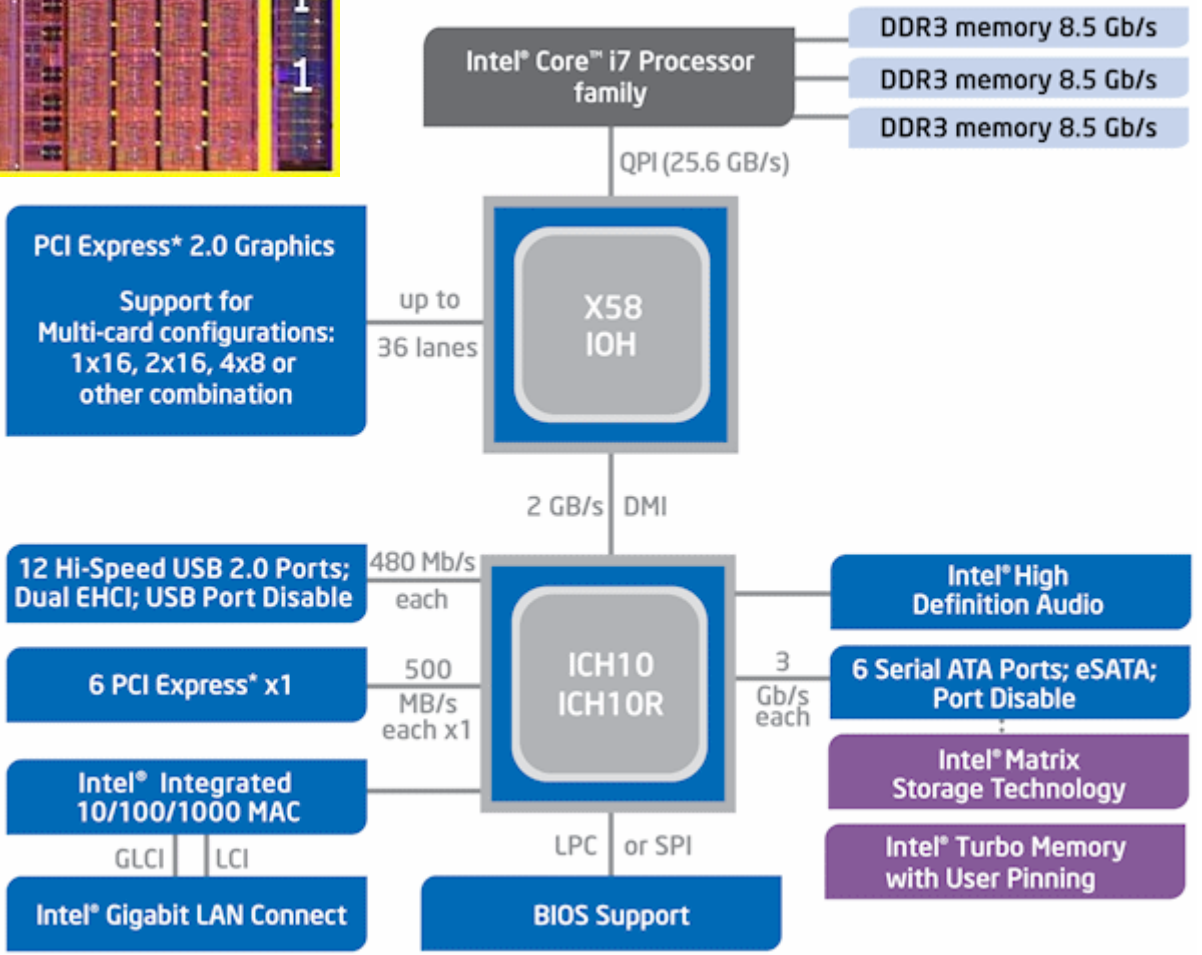



**Current and expected eras of Intel processor architecture.**
(BORKAR, S. Platform 2015: Intel processor and platform evolution for the next decade. 2005)

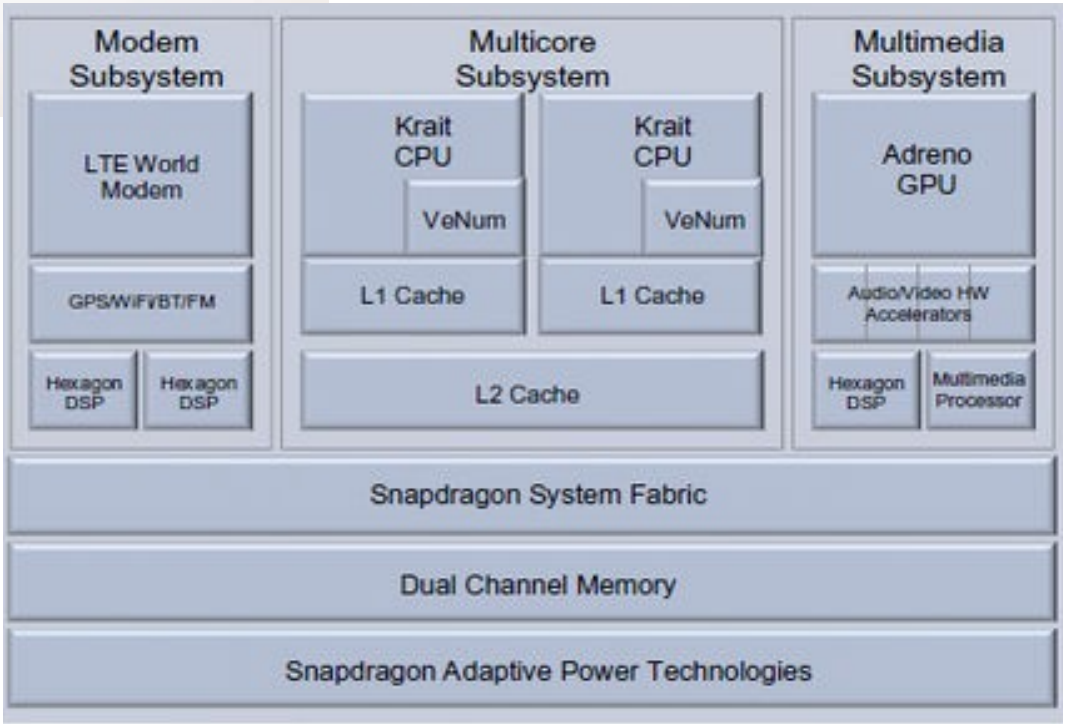




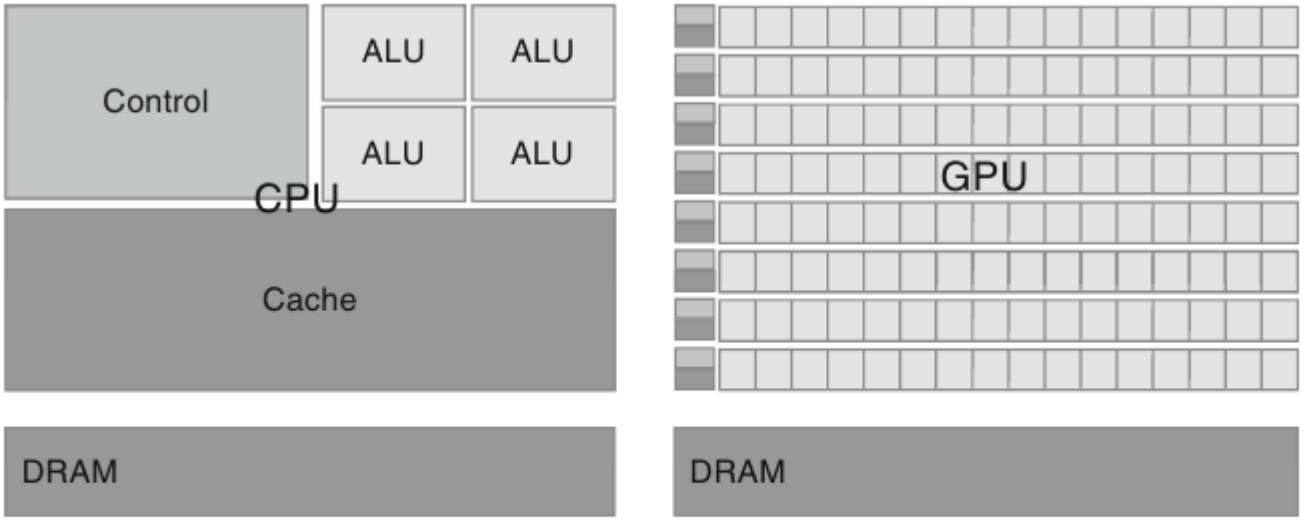




**http://techreport.com/review/15818/intel-core-i7-processors**

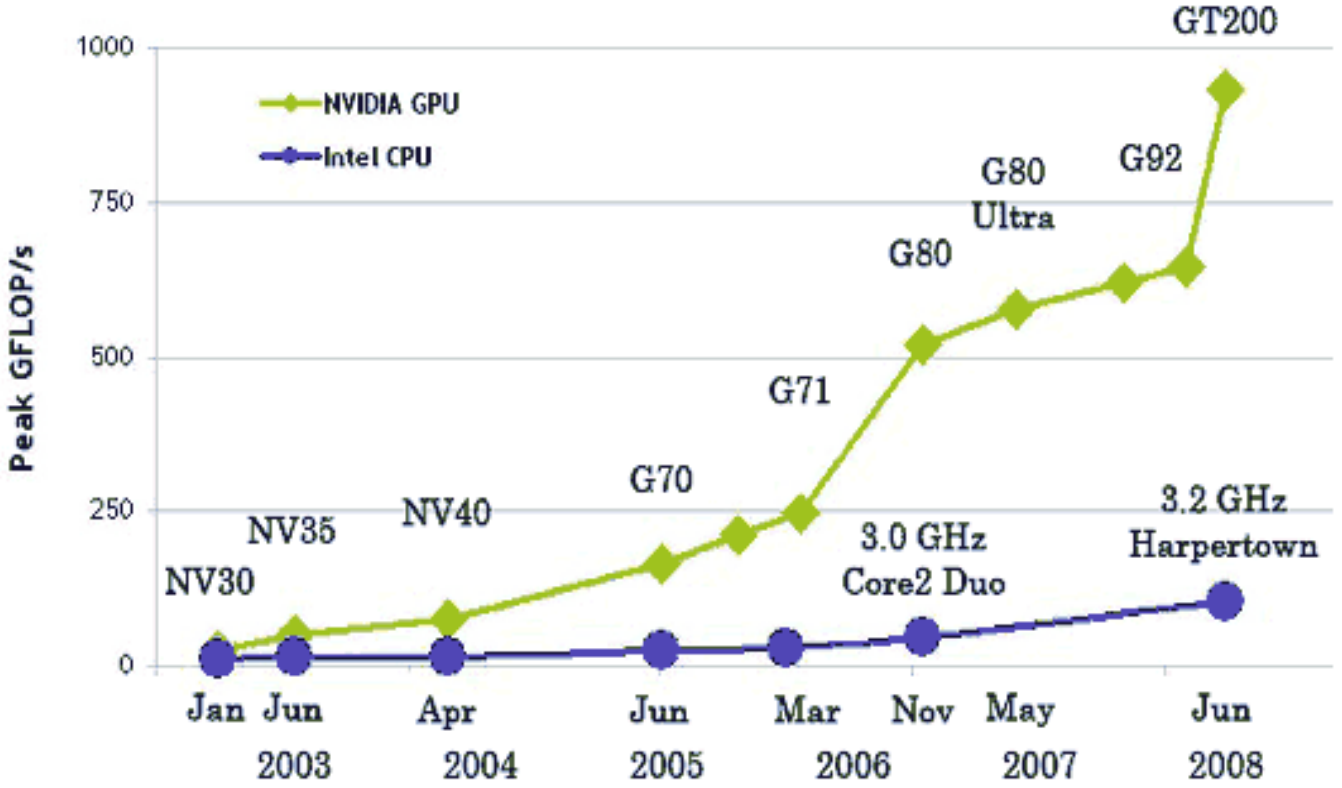



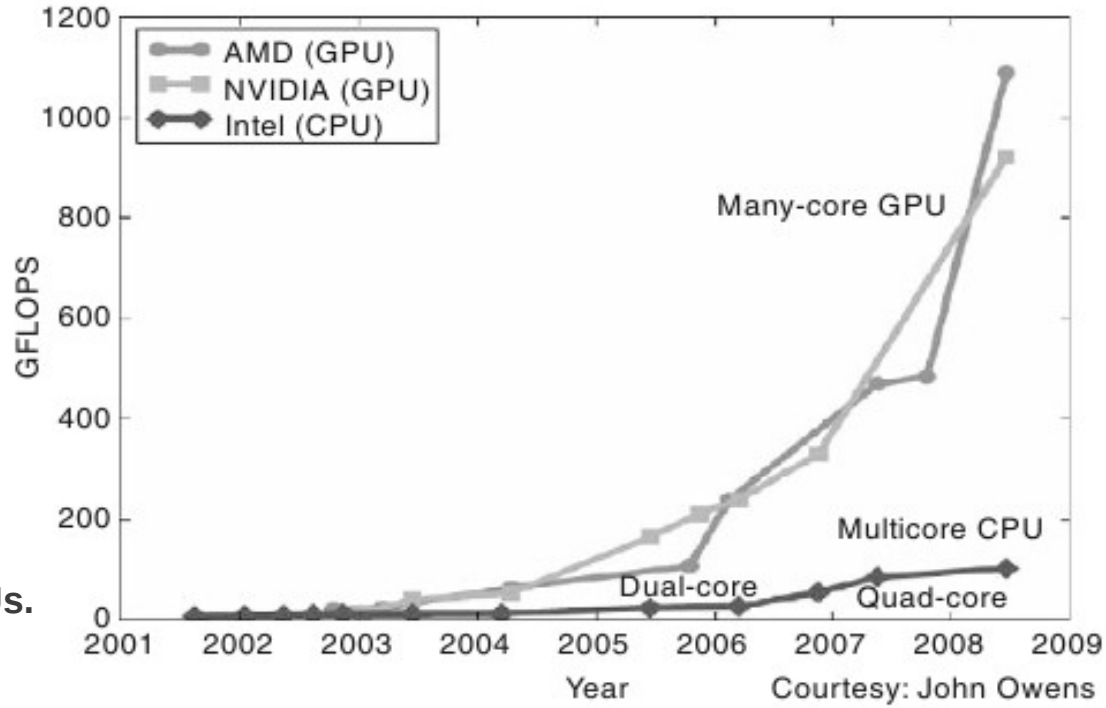

**http://www.qualcomm.com/snapdragon**

- **Arquitetura híbrida**
  - Múltiplos núcleos + uma (ou mais) GPU(s)
  - Projetos com focos diferentes:
    - CPU: tarefas irregulares, otimizada para desempenho sequencial, grande capacidade de cache, lógica de controle sofisticada.
    - GPU: computação intensiva, otimizada para operações de ponto flutuante, grande quantidade de threads, cache pequena para controle.



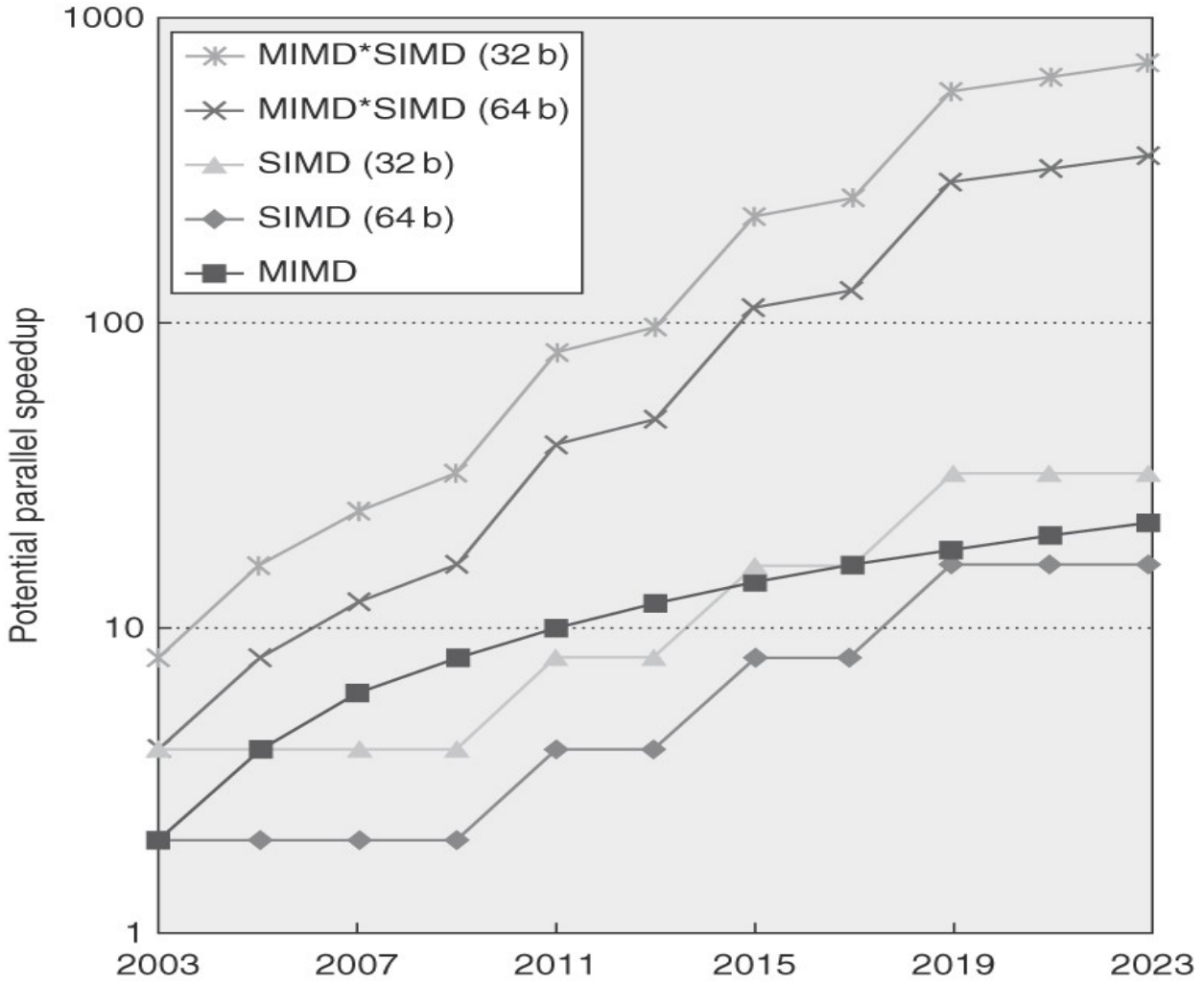

**CPU and GPU design philosophies.**
(KIRK, D. Programming massively parallel processor. NVIDIA, 2010)



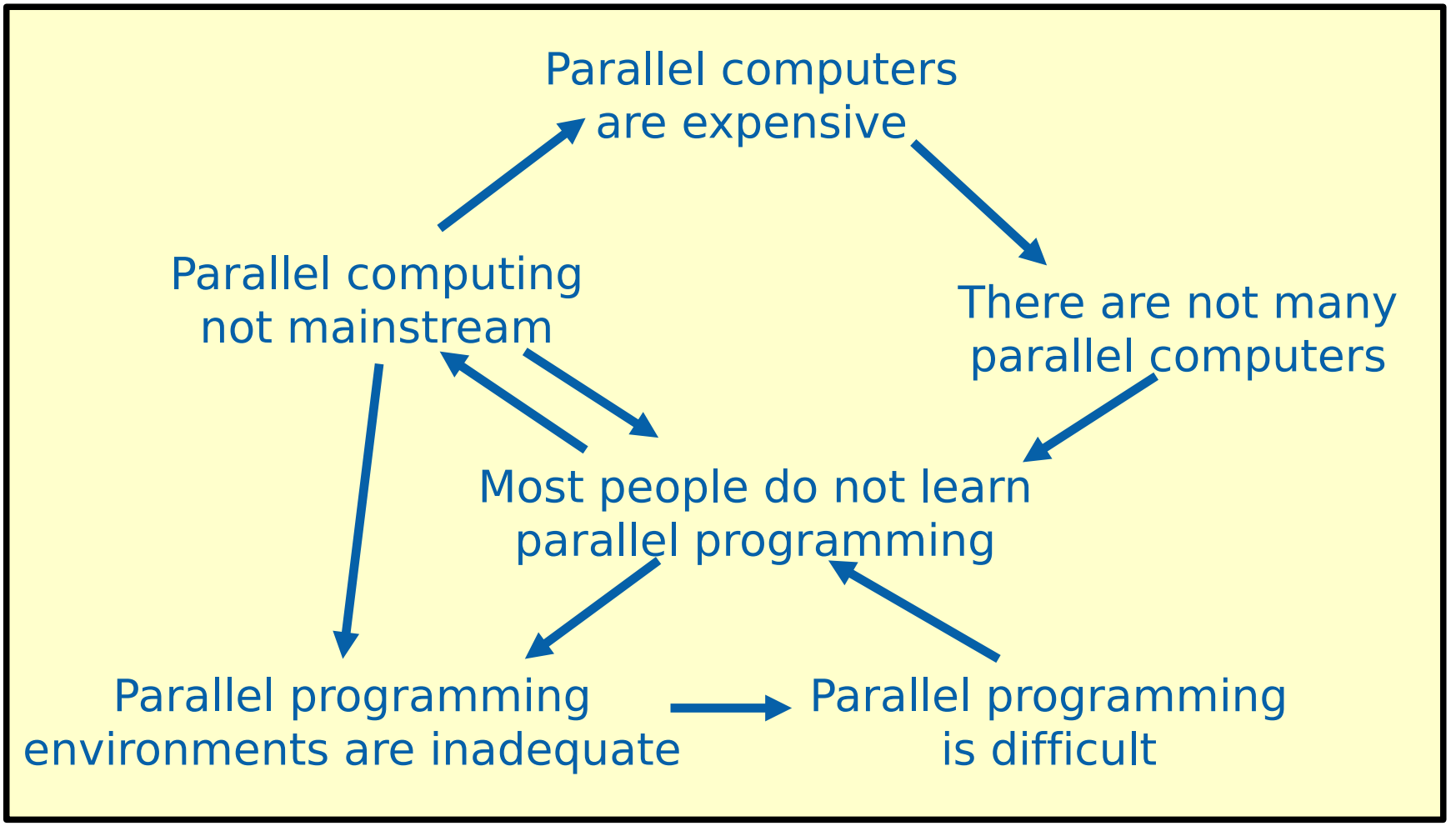

**Enlarging performance gap between GPUs and CPUs.**
(KIRK, D. Programming massively parallel processors. NVIDIA, 2010)

- Relação SIMD x MIMD
  - ✔ “Abordagens SIMD são mais fáceis de se programar do que arquiteturas puramente MIMD.”



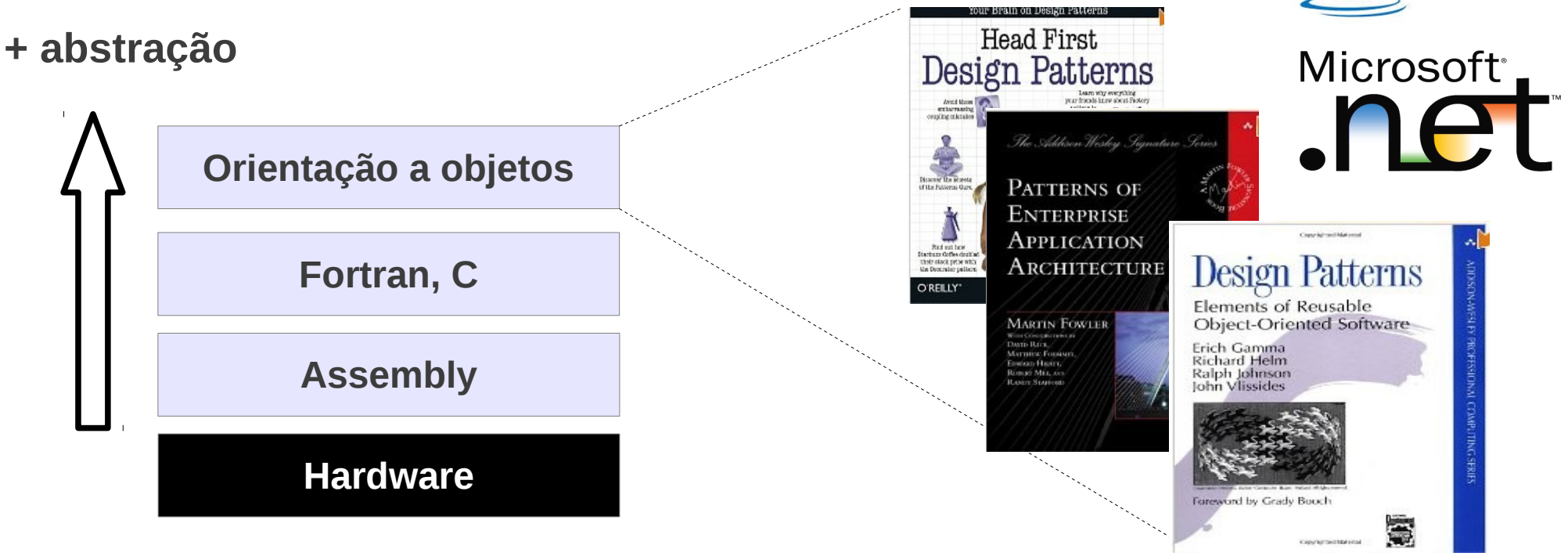

**Potential speedup via parallelism from MIMD, SIMD, and both MIMD and SIMD over time for x86 computers.** Two cores per chip for MIMD will be added every two years and the number of operations for SIMD will double every four years. (HENNESSY, PATTERSON. Computer Architecture. 5 ed, 2011)

- Para uso efetivo de arquiteturas híbridas e aumento escalável de desempenho, devemos especificar explicitamente o paralelismo de nossos algoritmos!!!
  - ✔ **Programadores atuais devem ser "programadores paralelos"!!!**




**Why parallel? Why now?** Introduction to parallel programming – part I. Intel

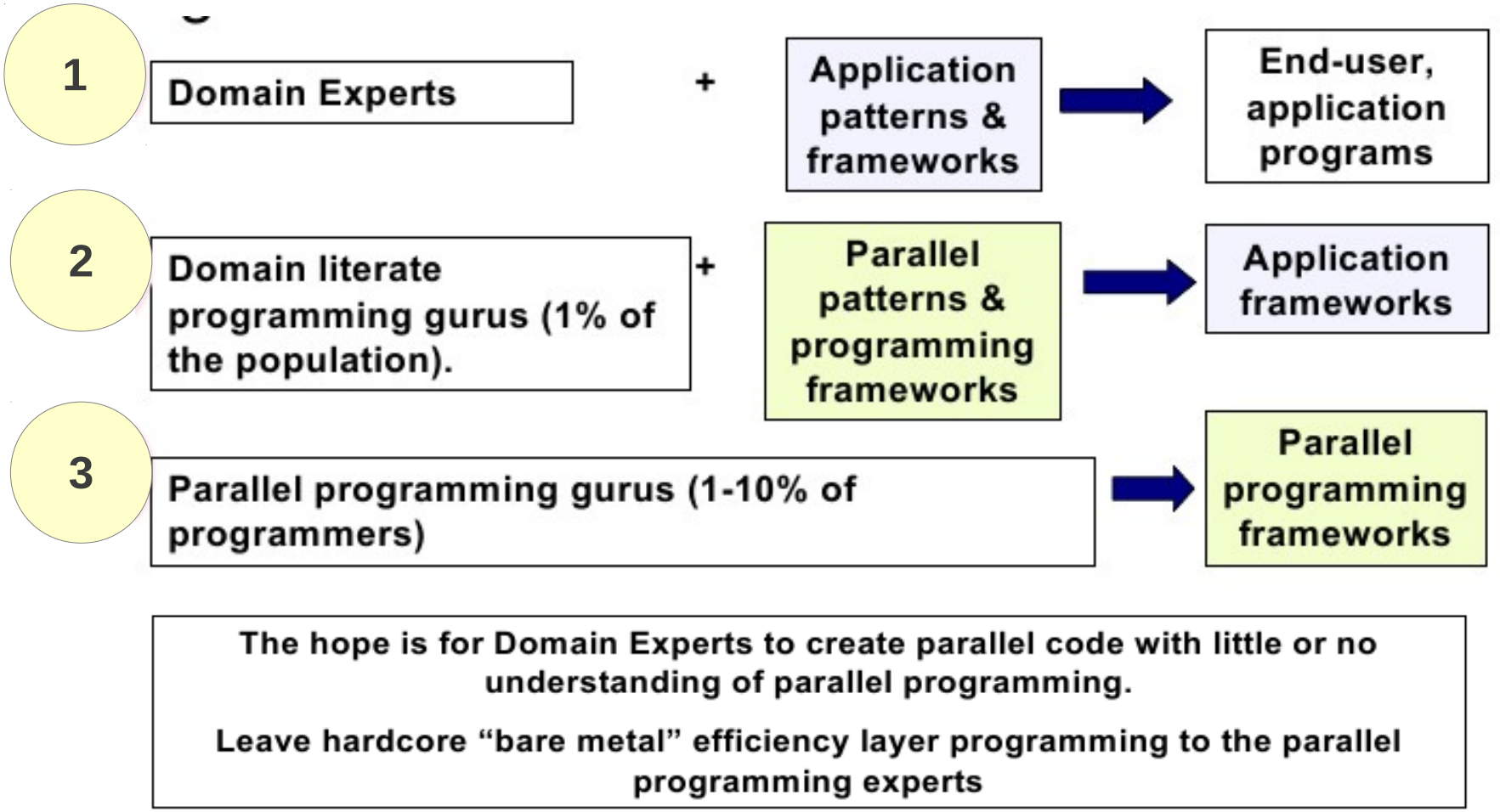
- E o software?
- E a “contribuição” da Engenharia de Software?




“Concorrência é a próxima grande revolução na forma como desenvolvemos software.”
(SUTTER, H. The free lunch is over. Dr. Dobb's Journal, 30(3), March 2005)

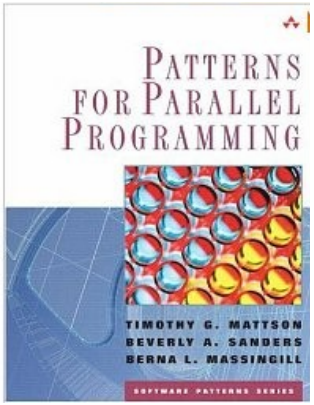
“Embora impulsionada pelos avanços do hardware, a 'revolução paralela' é primeiramente uma 'revolução do software'.”
(MILLER, A. A guide to parallel programming. Microsoft patterns & practices)

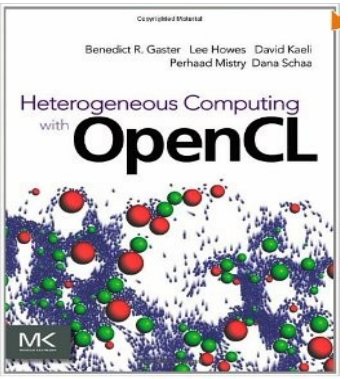
- Como fazer?
  - “Apesar do grande número de ferramentas disponíveis, o número de programadores (paralelos) ainda é pequeno. É preciso entender como projetar software paralelo.”

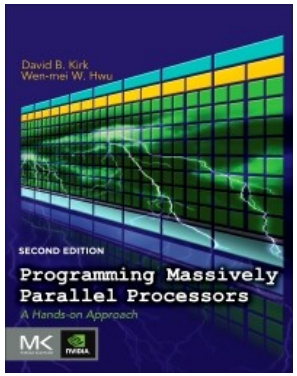


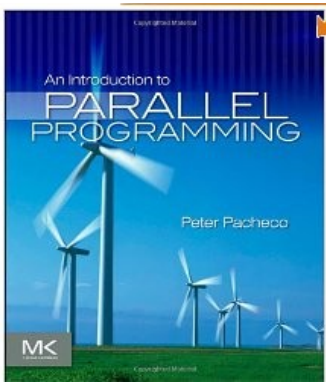

(KEUTZER, K.; MATTSON, T. A design pattern language for engineering (parallel) software)

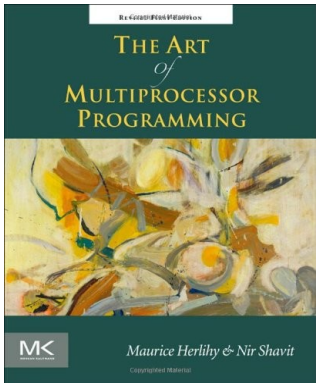
- Como fazer? **“Think parallel”**
  - Evitar a semântica sequencial.
  - Programar com base em padrões paralelos, os quais representam as “melhores práticas”.
  - Programar com ferramentas que forneçam implementações eficientes destes padrões.

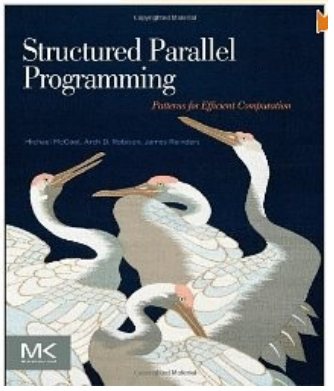


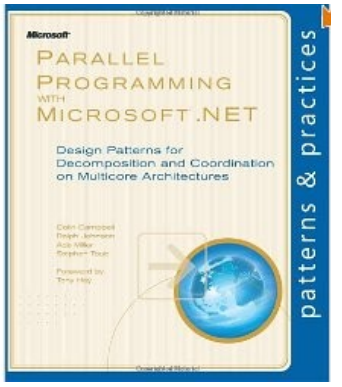



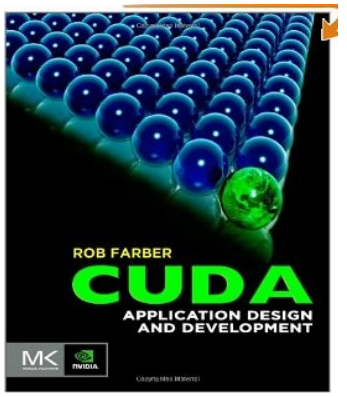



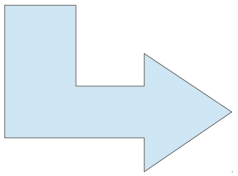



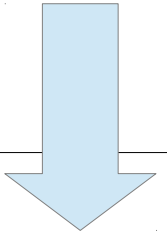
- Evitar a semântica sequencial.

```
for (i=0; i < numero_web_sites; i++)
  busca(palavra, websites[i]);
```



```
parallel_for (i=0; i < numero_web_sites; i++)
  busca(palavra, websites[i]);
```

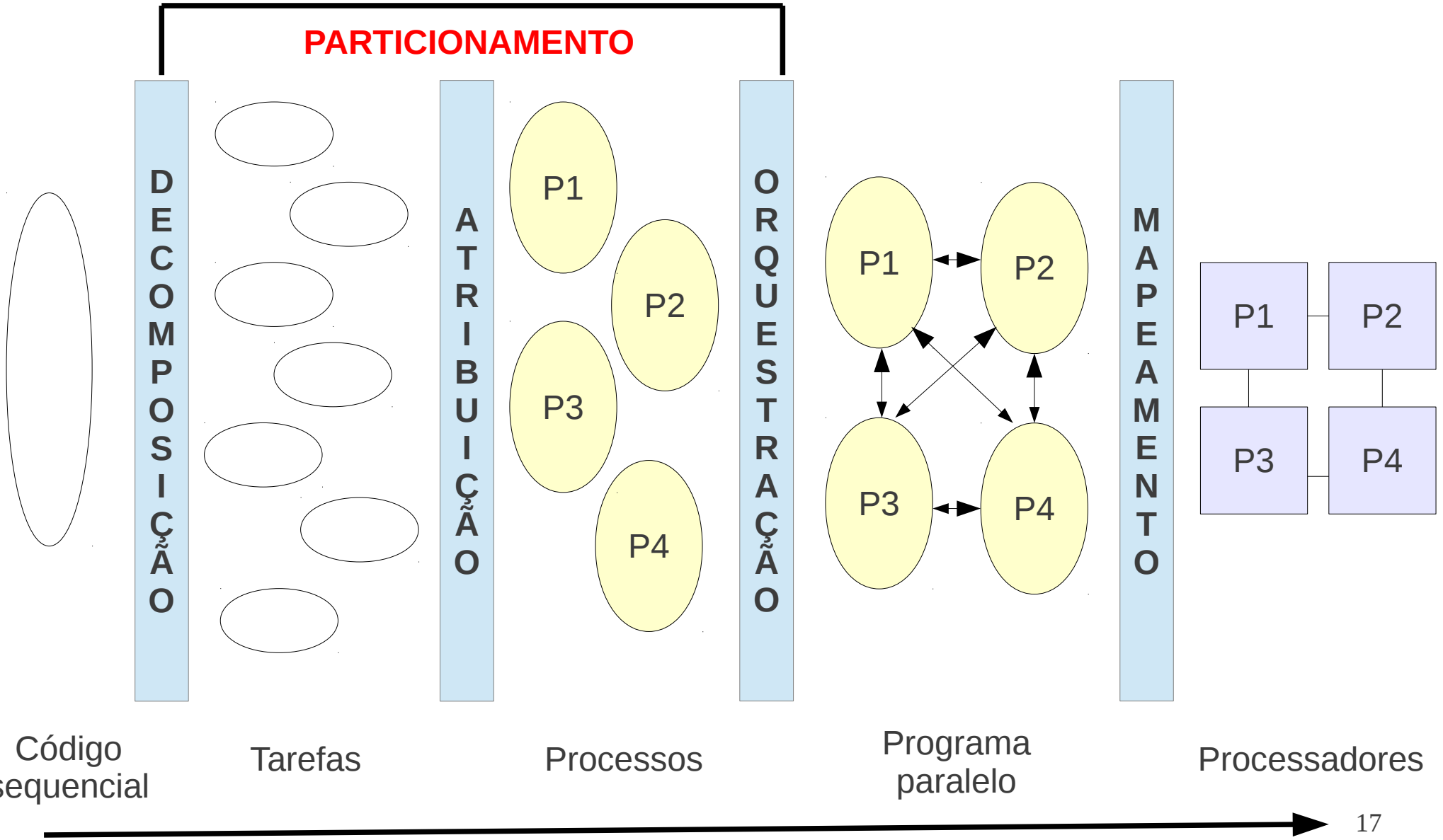
```
void somaVetores(int n, double a[n], double b[n], double c[n])
{
  for (int i=0; i < n; i++)
    a[i] = b[i] + c[i];
}
```



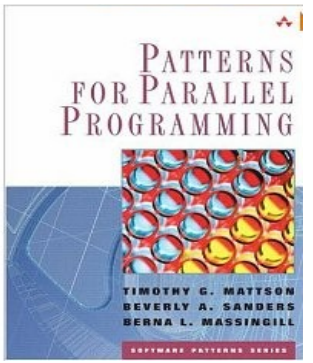
```
void somaVetores(int n, double a[n], double b[n], double c[n])
{
  a[:] = b[:] + c[:];
}
```

Soma de dois vetores usando a notação de array no Cilk Plus

- Evitar a semântica sequencial
  - Etapas para criar um programa paralelo.



- Escopos de desenvolvimento




Projeto do algoritmo

- Identificar a concorrência
  - Definir tarefas concorrentes
- Estrutura do algoritmo
  - Mapear tarefas para processos para explorar a arquitetura paralela

Desenvolvimento do software

- Estruturas de suporte
  - Padrões para codificação
- Mecanismos de implementação
  - Recursos usados para escrever programas paralelos

- **Decomposição**
  - Identificar a concorrência e decidir como explorá-la.
    - Paralelismo de tarefas (decomposição funcional)
    - Paralelismo de dados (decomposição de domínio)
    - Pipeline

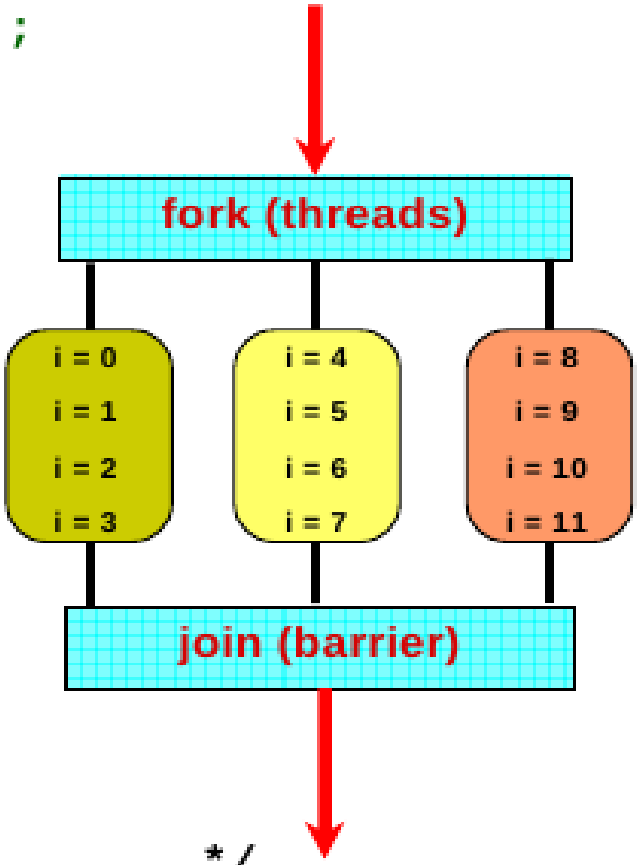
- **Paralelismo de dados**
- Uma mesma função operando em paralelo sobre dados diferentes.
- Aumenta de acordo com a quantidade de dados.
- Favorece o ganho de desempenho (*speedup*) e o paralelismo de grão fino.
- Questões importantes
  - Tamanho e formato do pedaço (*chunk*) do dado
  - Dependências entre dados

Bloco Filhos de Gandhy – 16.000 sócios.

- **Paralelismo de dados**

```
for (i = 0; i < 12; i++)
      C[i] = A[i] + B[i];
```

```
int A[12] = {...}; int B[12] = {...}; int C[12];

void add_arrays(int start)
{
   int i;
   for (i = start; i < start + 4; i++)
     C[i] = A[i] + B[i];
}

int main (int argc, char *argv[])
{
   pthread_t threads_ids[3];
   int rc, t;
   for(t = 0; t < 4; t++) {
     rc = pthread_create(&thread_ids[t],
                         NULL        /* attributes       */,
                         add_arrays /* function         */,
                         t * 4       /* args to function */);
   }
   pthread_exit(NULL);
}
```

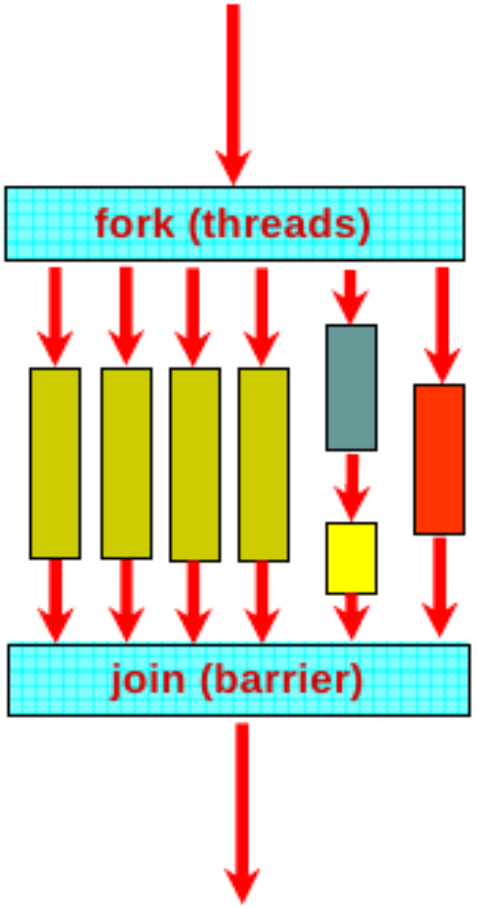



(RABAH, R. Design patterns for parallel programming.MIT 2007)

- **Paralelismo de tarefas**
  - Diferentes funções simultâneas executando de forma independente.
  - Questões:
    - Quantidade de tarefas
    - Carga de trabalho por tarefa
    - Dependências entre tarefas

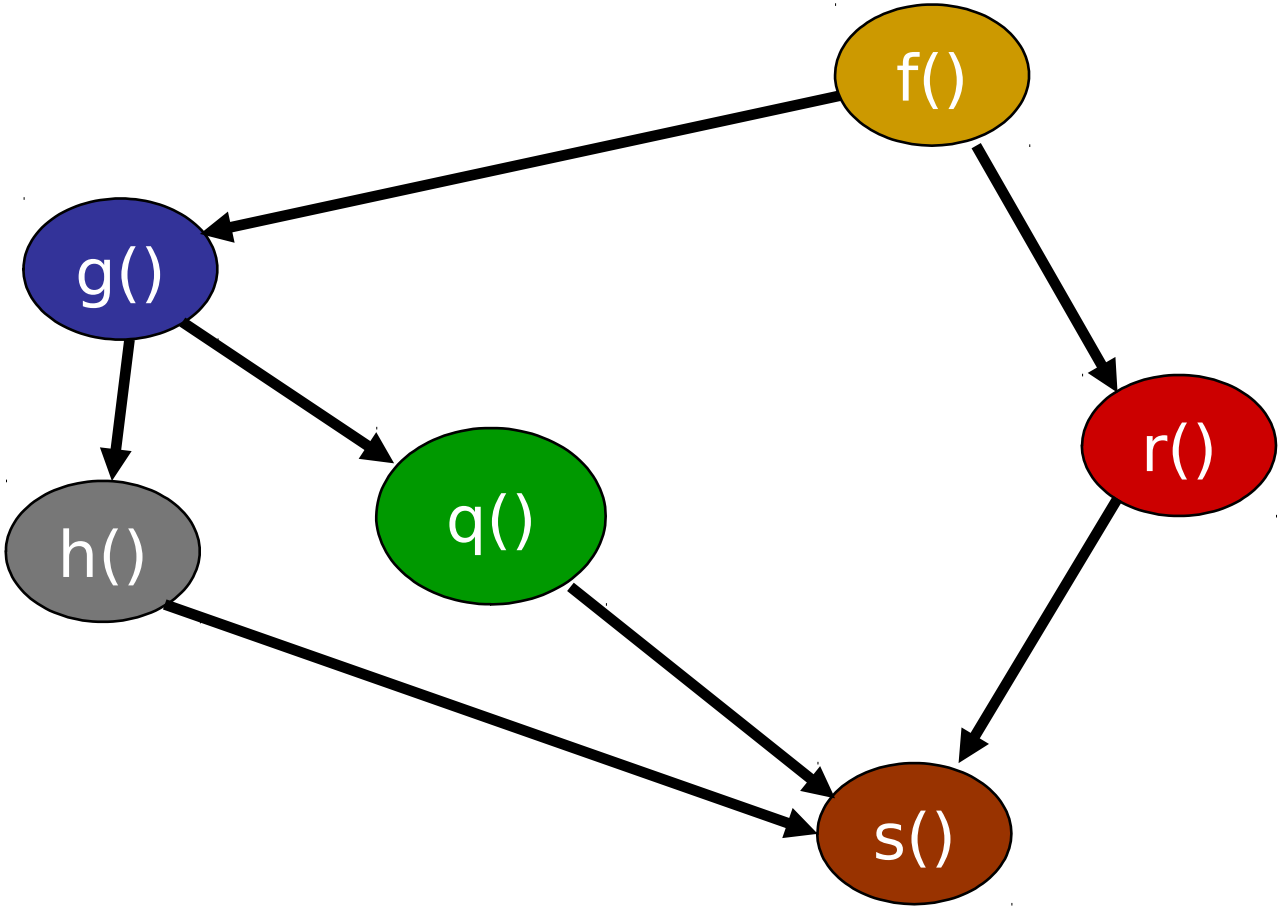
Orixás

```
pthread_create(/* thread id          */,
               /* attributes        */,
               /* any function      */,
               /* args to function */);
```

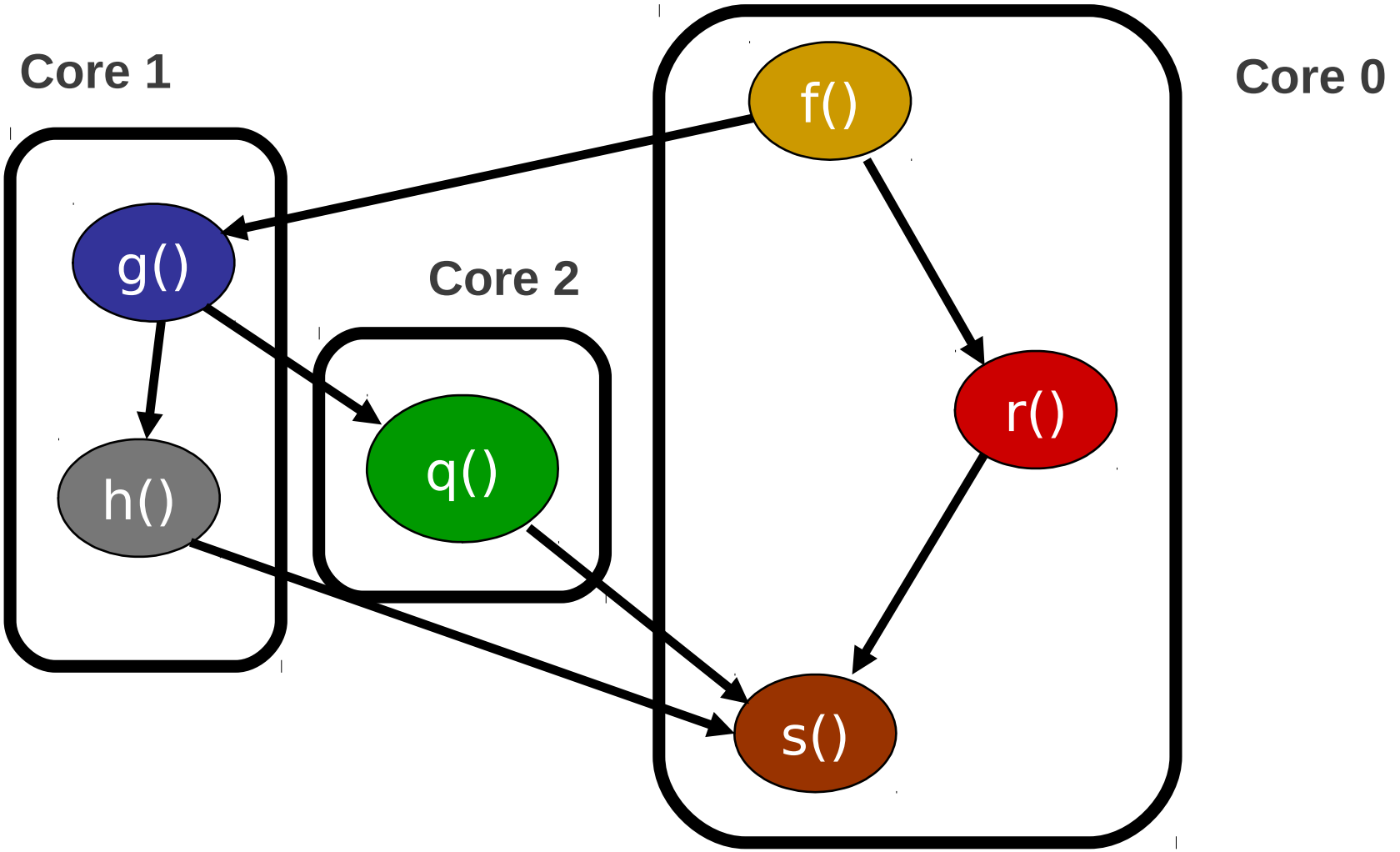



(RABAH, R. Design patterns for parallel programming, MIT 2007)

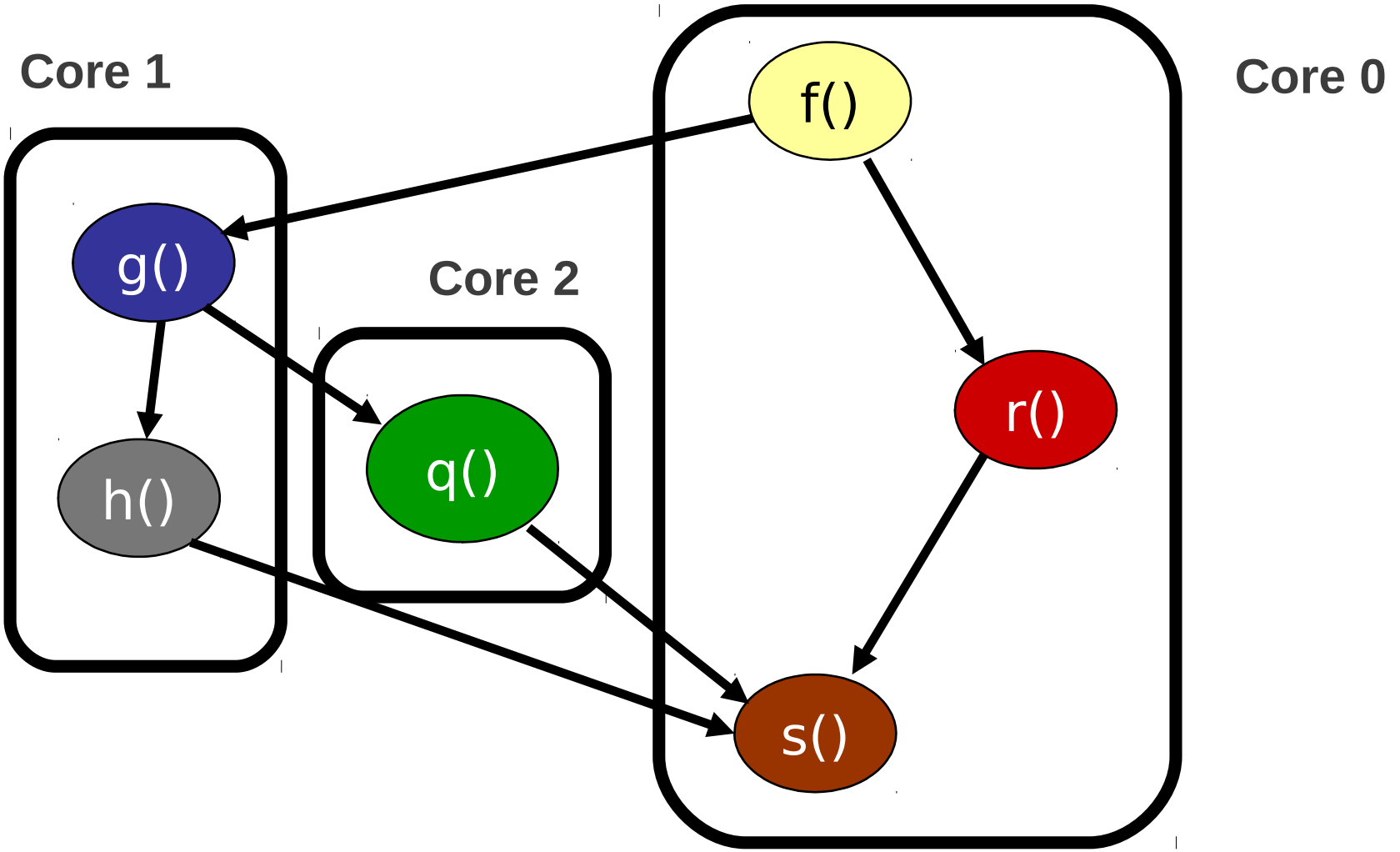
# Paralelismo de tarefas



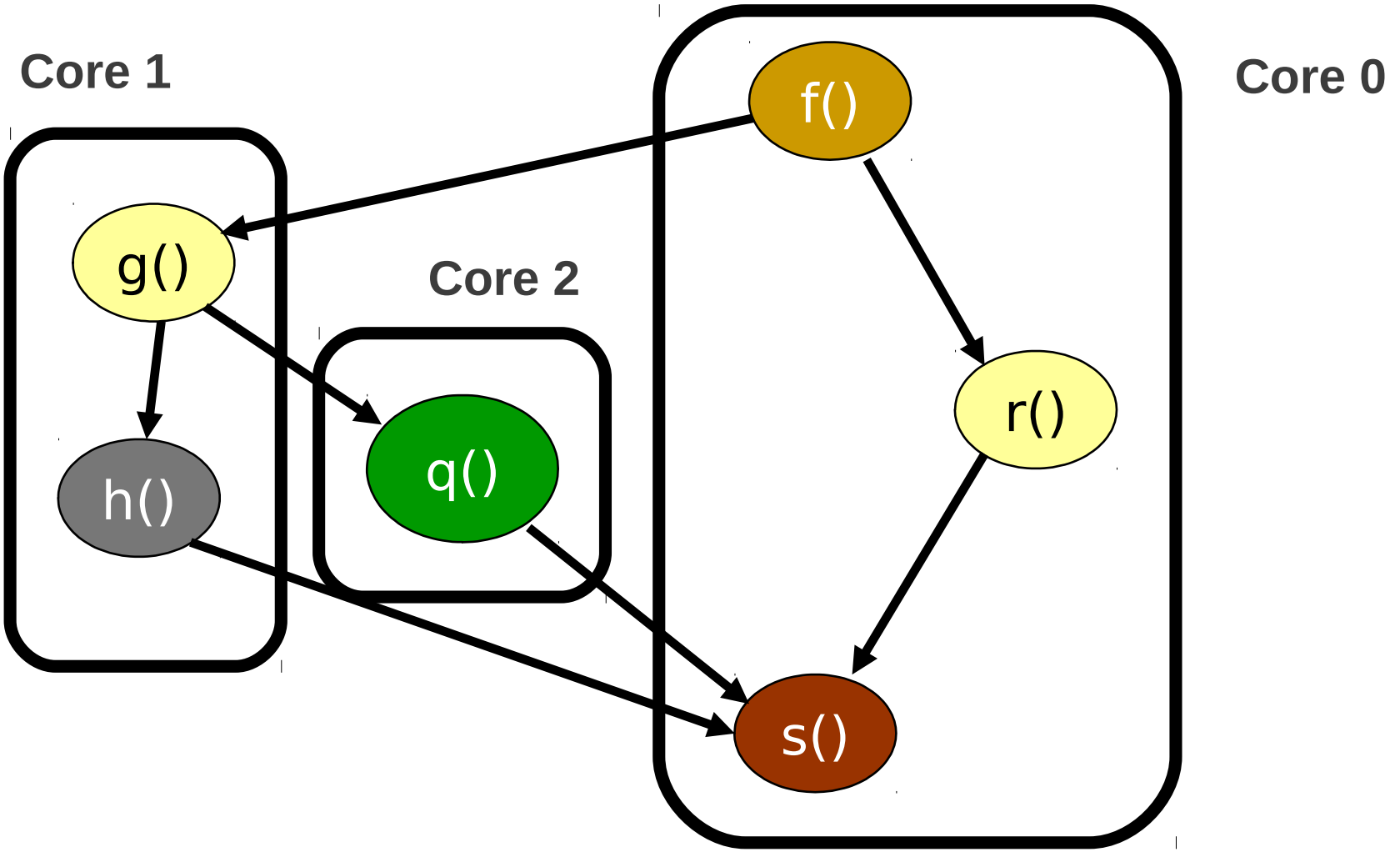
# Paralelismo de tarefas



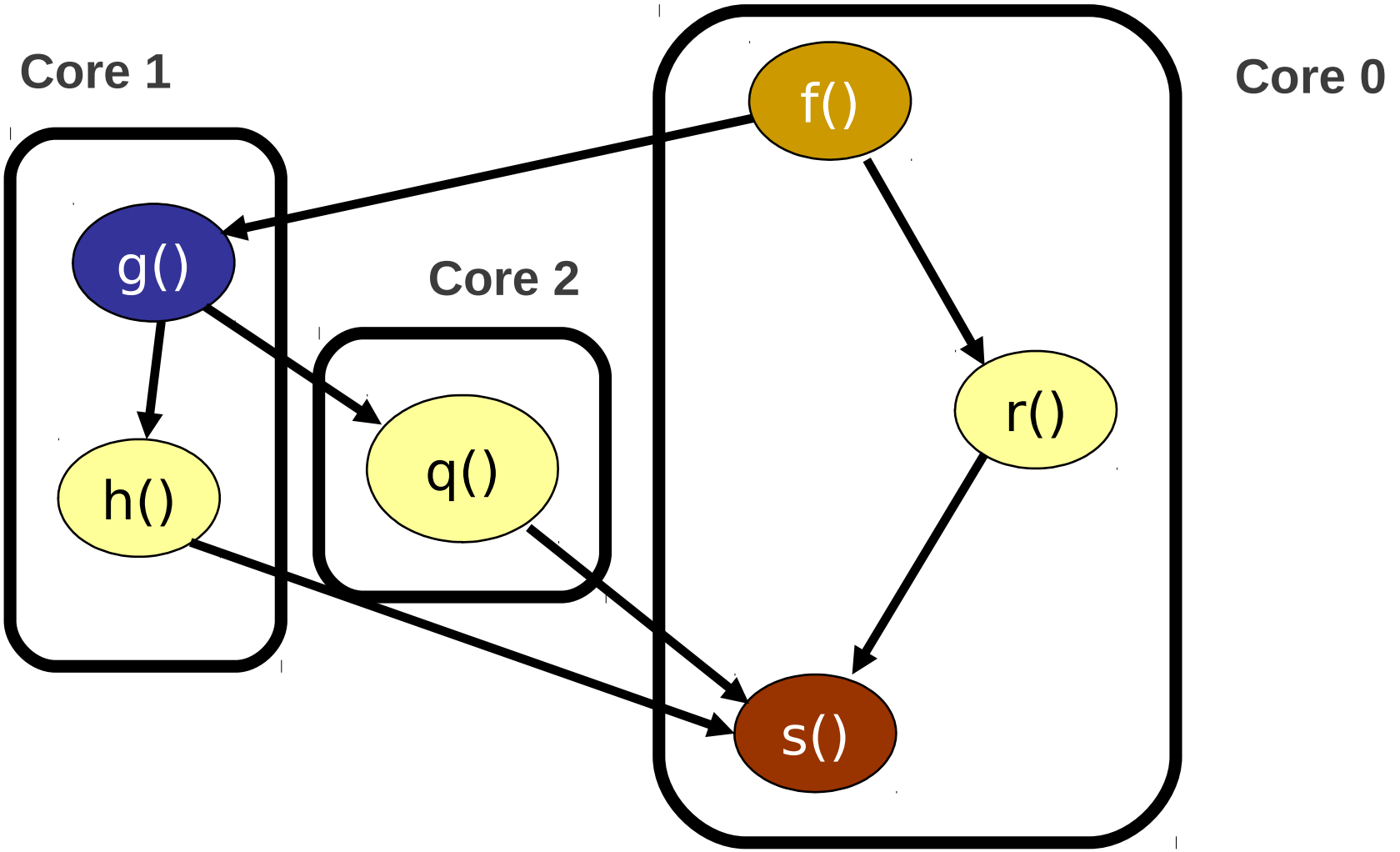
# Paralelismo de tarefas



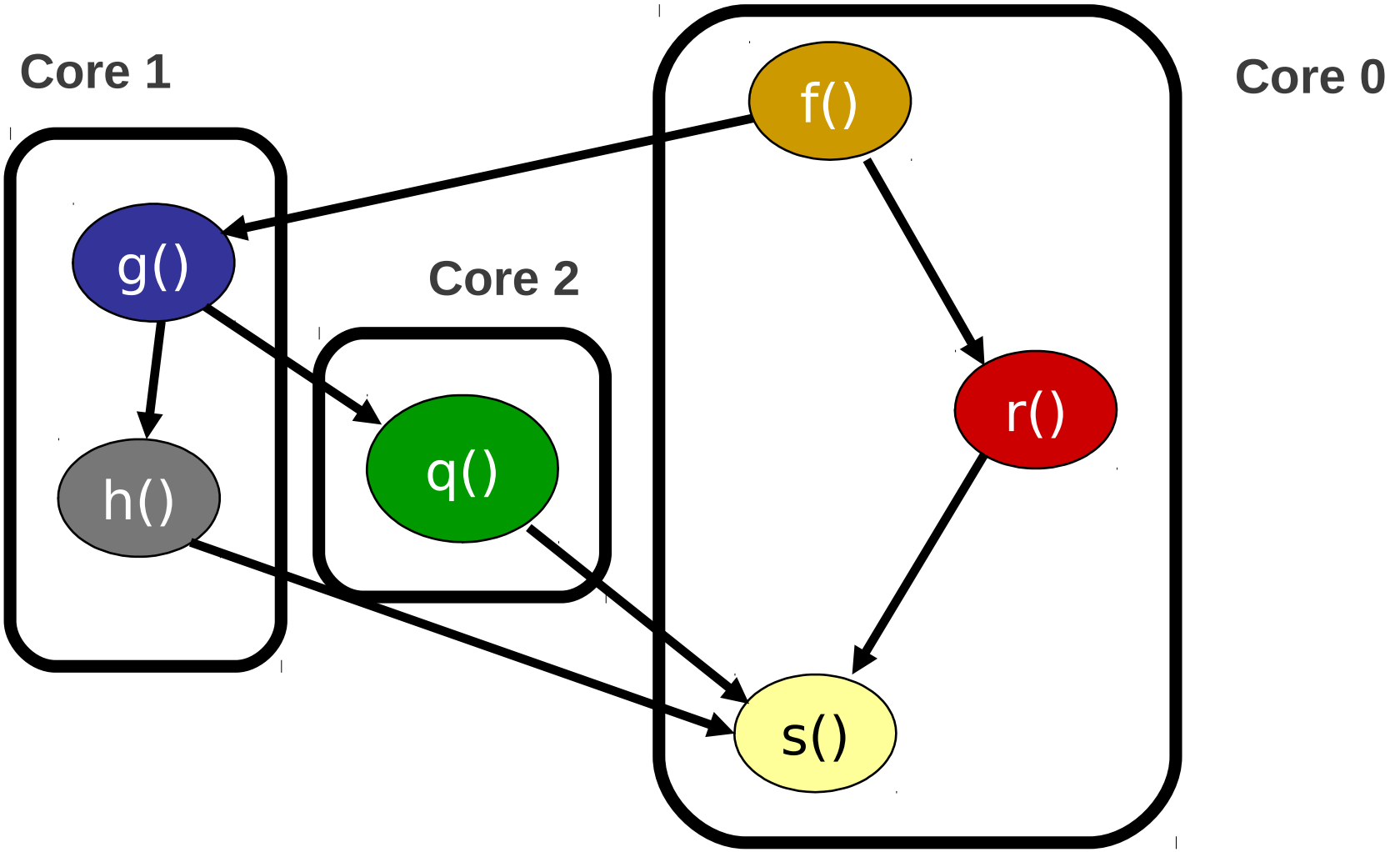
# Paralelismo de tarefas



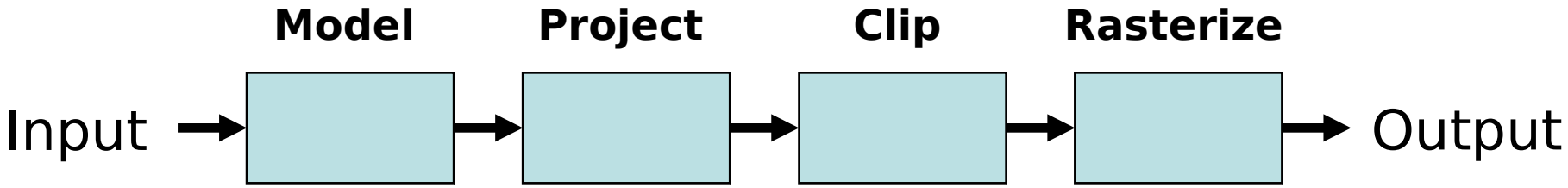
# Paralelismo de tarefas



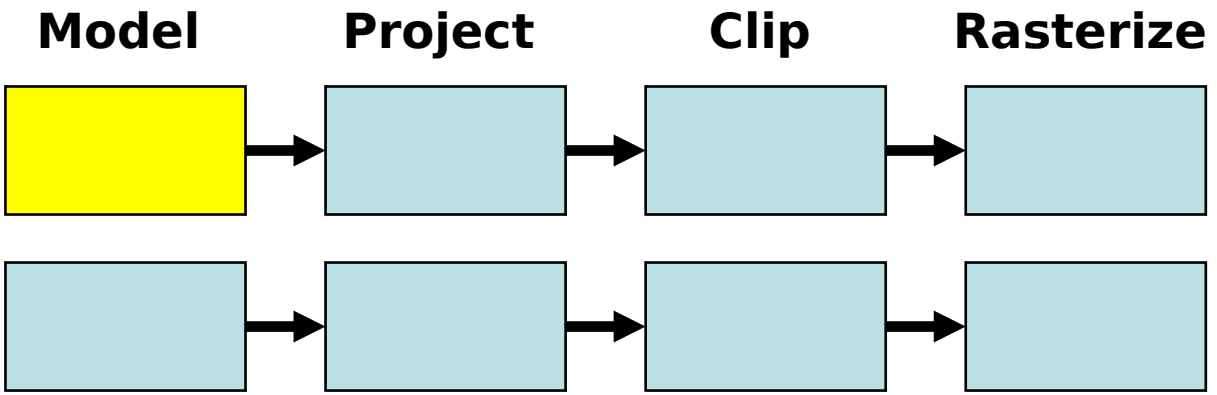
# Paralelismo de tarefas



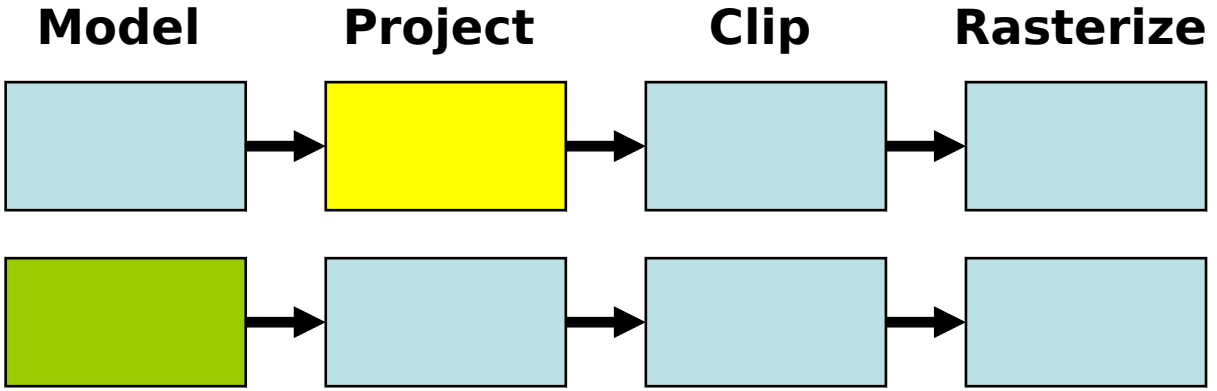
- **Pipeline**
- Caso especial de paralelismo de tarefa no qual existe uma dependência de dados entre as tarefas.
- Ex.: renderização 3D em computação gráfica



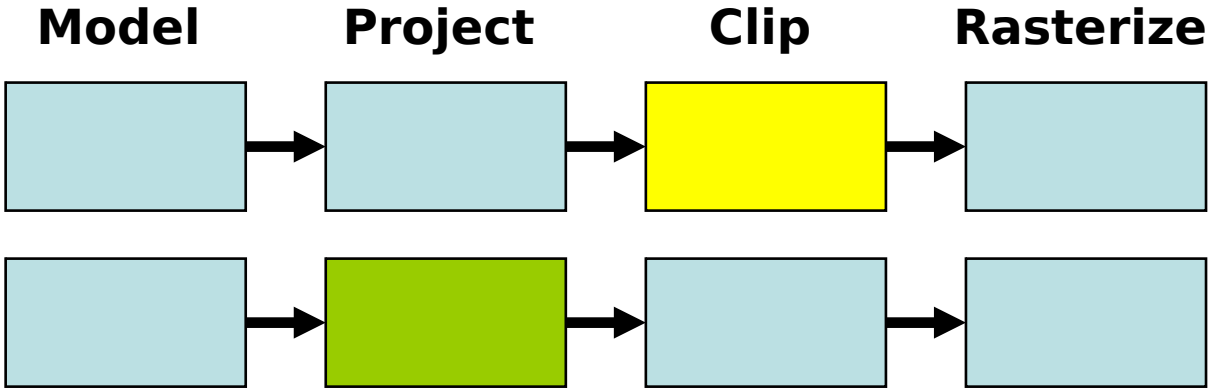
Passo 1



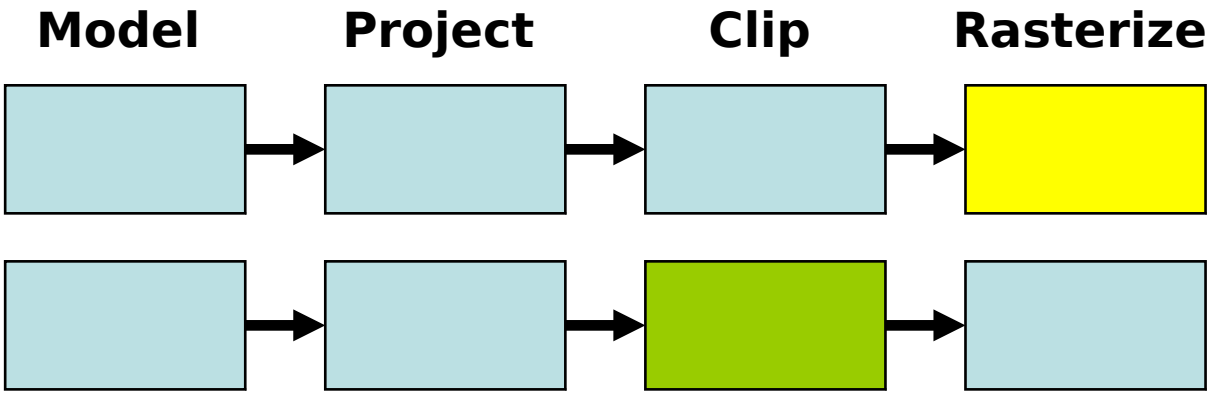

Passo 2



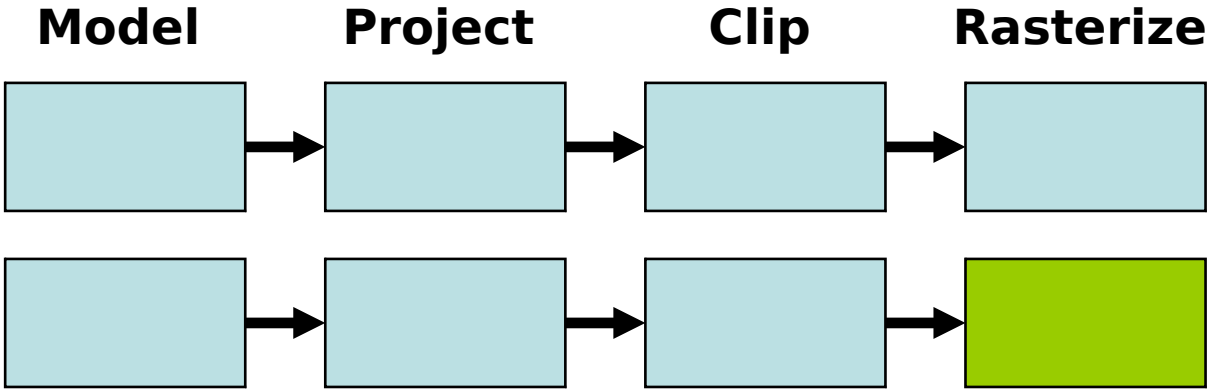

Passo 3



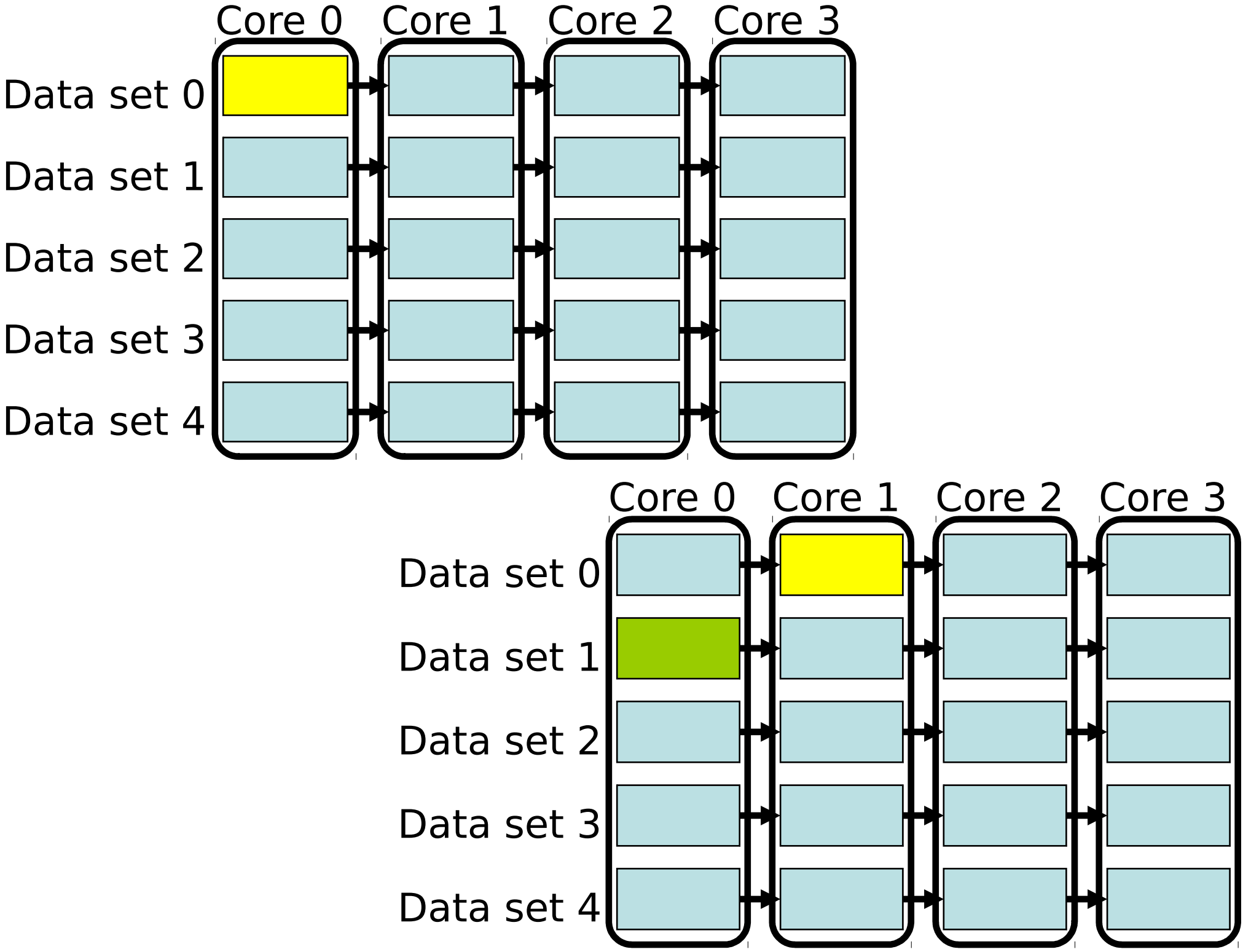
Passo 4

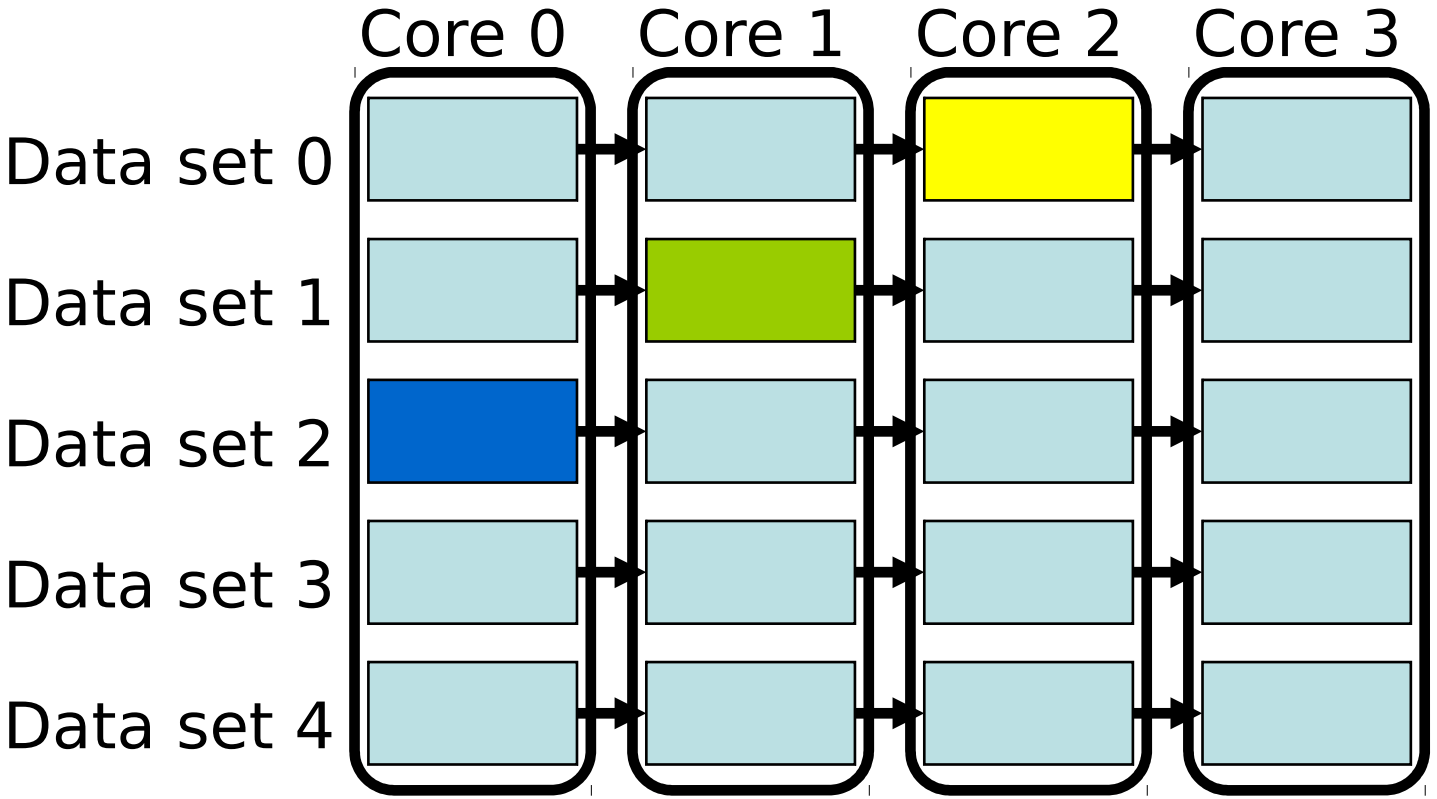


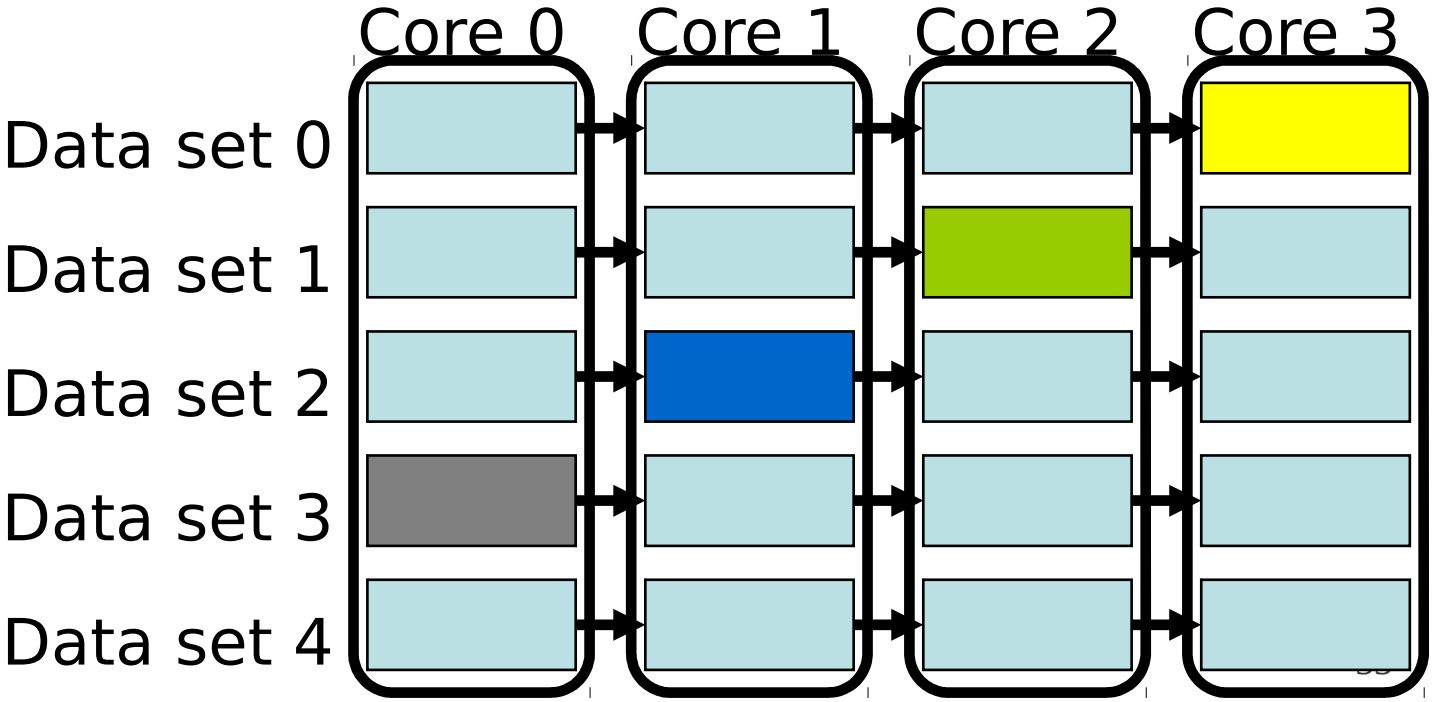

Passo 5

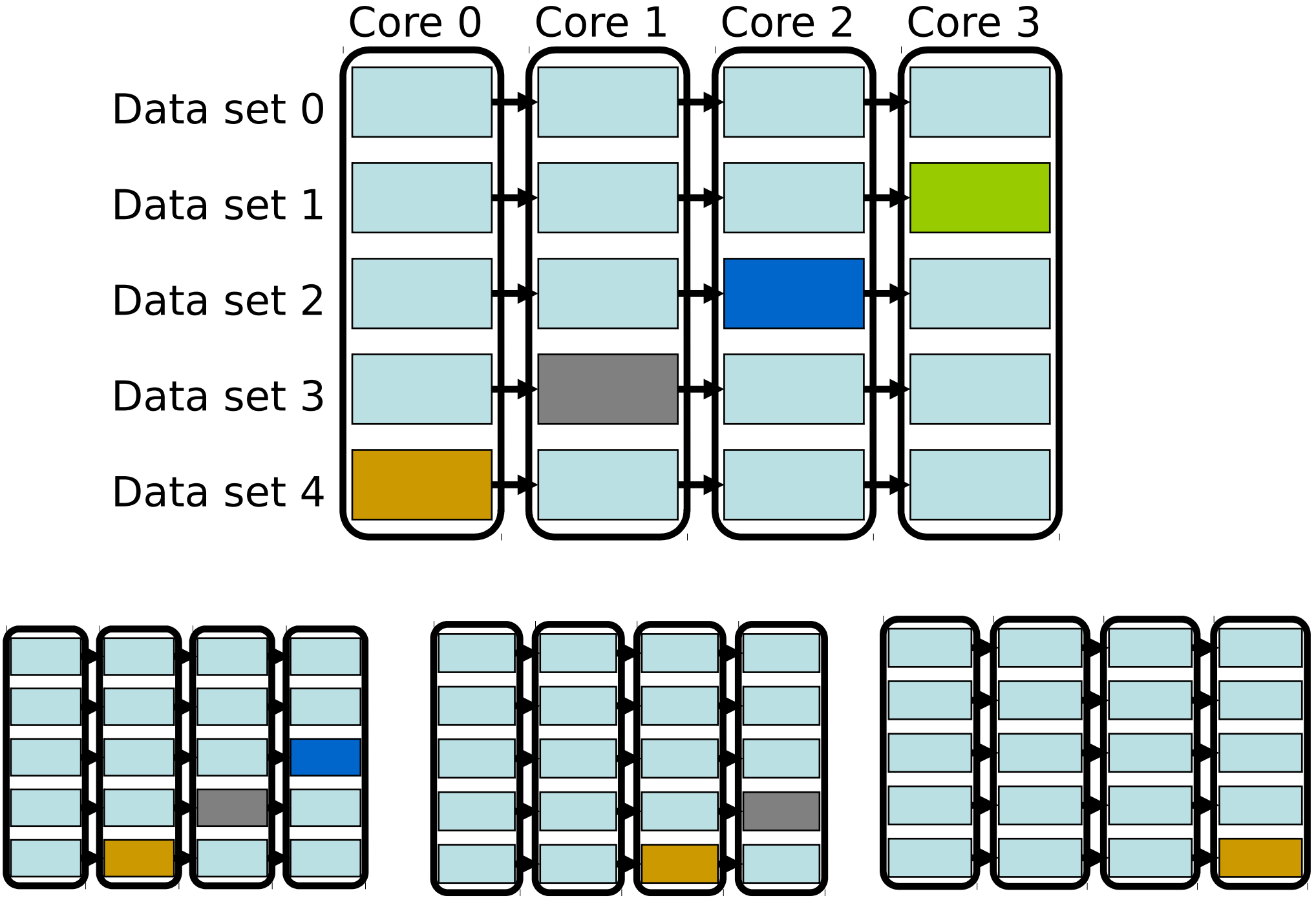



2 unidades de dados em 5 passos

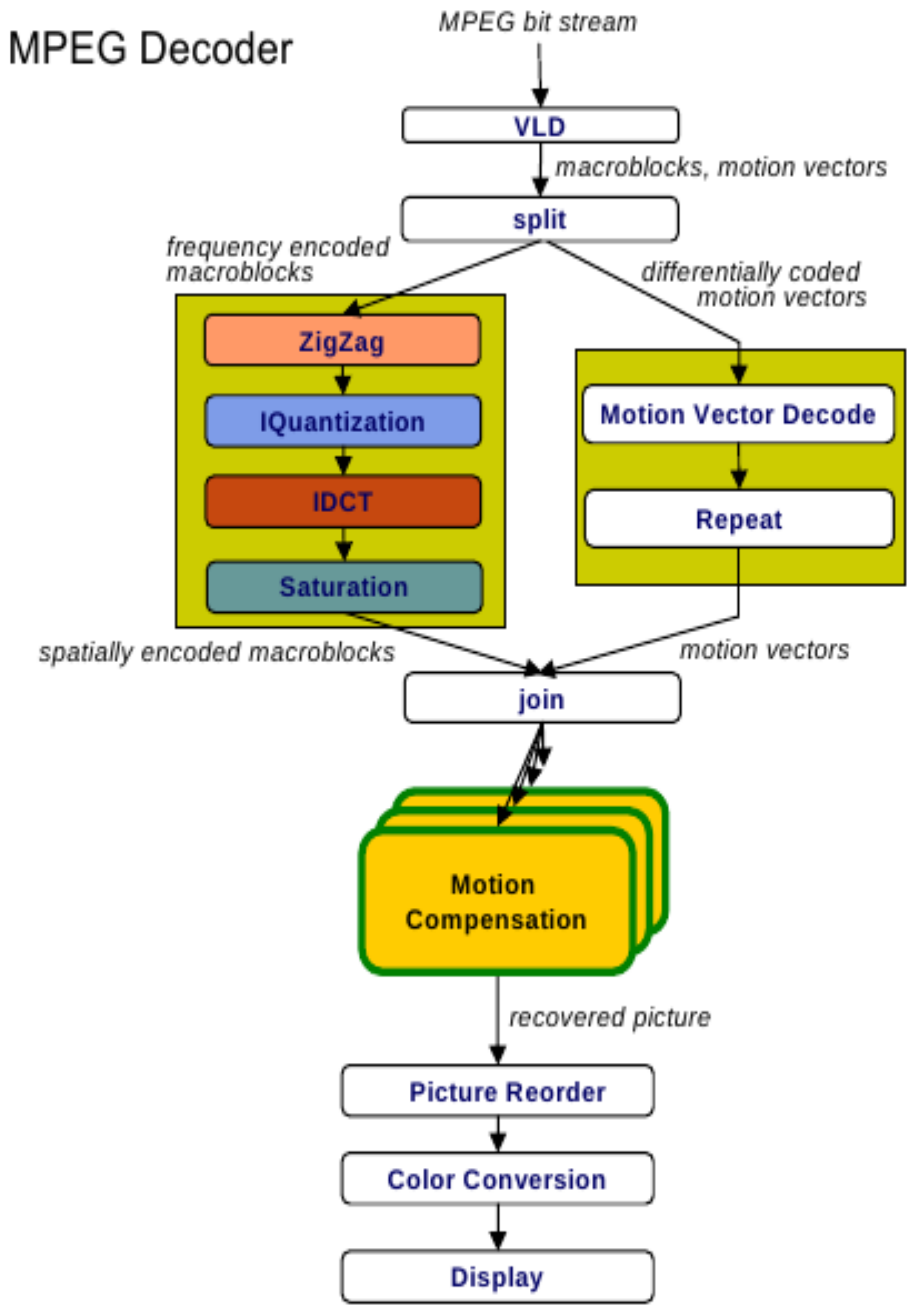

Core 0
Core 1
Core 2
Core 3
Data set 0
Data set 1
Data set 2
Data set 3
Data set 4
Core 0
Core 1
Core 2
Core 3
Data set 0
Data set 1
Data set 2
Data set 3
Data set 4


Core 0
Core 1
Core 2
Core 3
Data set 0
Data set 1
Data set 2
Data set 3
Data set 4

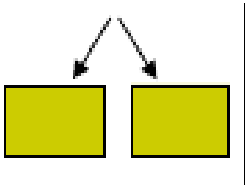

Core 0
Core 1
Core 2
Core 3
Data set 0
Data set 1
Data set 2
Data set 3
Data set 4


Core 0
Core 1
Core 2
Core 3
Data set 0
Data set 1
Data set 2
Data set 3
Data set 4

- Exemplo: decodificação MPEG

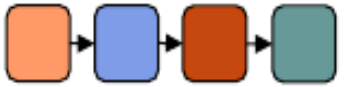



- Decomposição de tarefas
  - Paralelismo na aplicação



- Decomposição de dados
  - Mesma função em dados diferentes

- Pipeline
  - Sequências produtor-consumidor

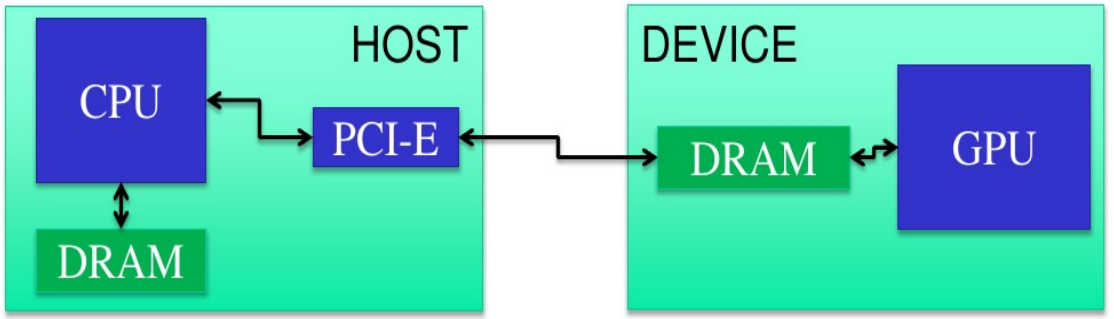


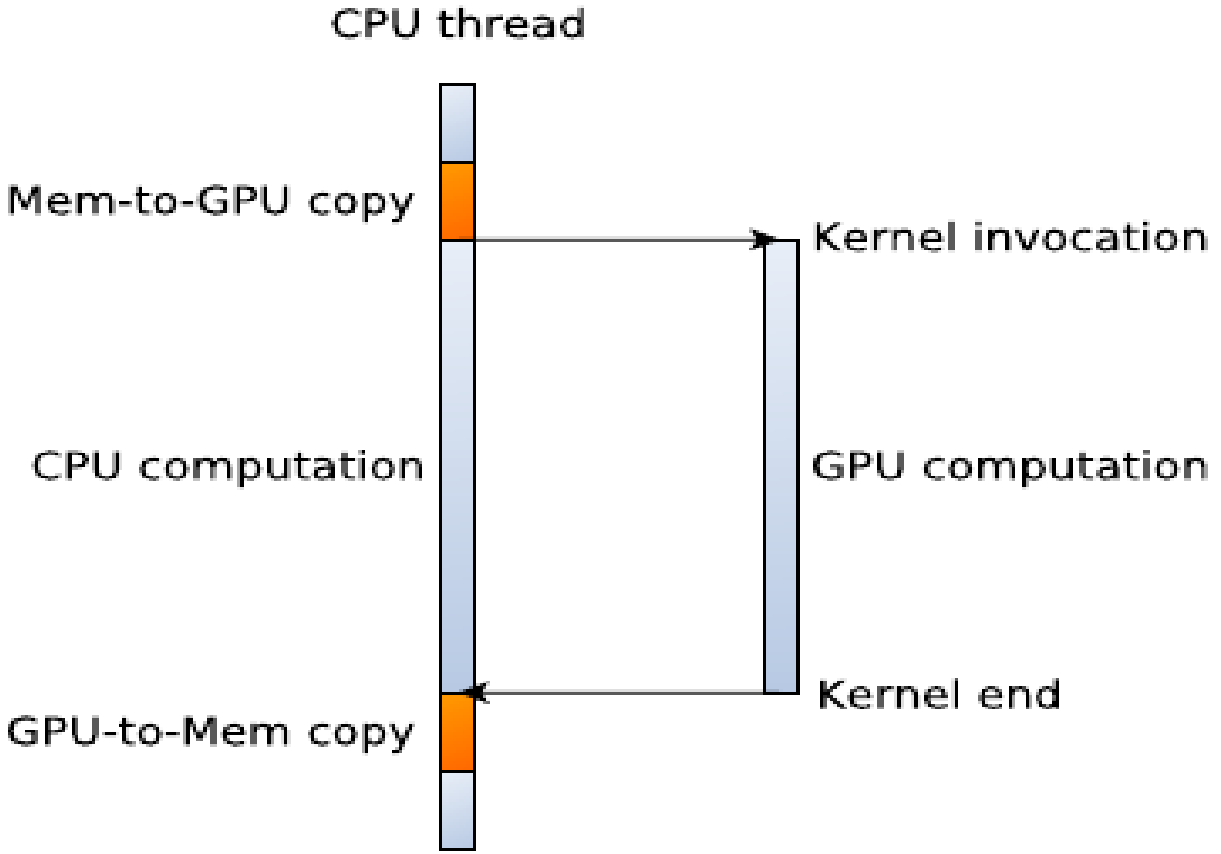
(RABAH, R. Design patterns for parallel programming.MIT 2007)

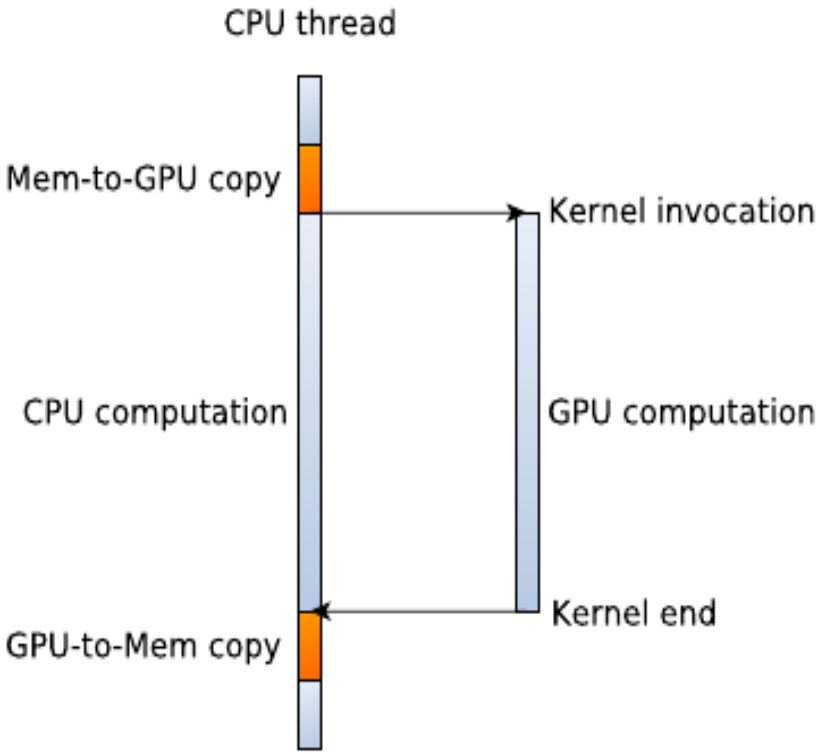
- **Atribuição**
  - Define a granularidade (tarefas x processos).
  - Agrupar as tarefas em processos, de modo a balancear a carga de trabalho e reduzir a comunicação.
  - Entendimento da aplicação, inspeção do código e uso de padrões conhecidos.

- **Orquestração e mapeamento**
  - Computação X comunicação.
  - Deve preservar a localidade de dados.
  - Deve priorizar o escalonamento de tarefas para atender as dependências de dados e de controle.

- **Questões importantes**
  - Gerência e movimentação de dados

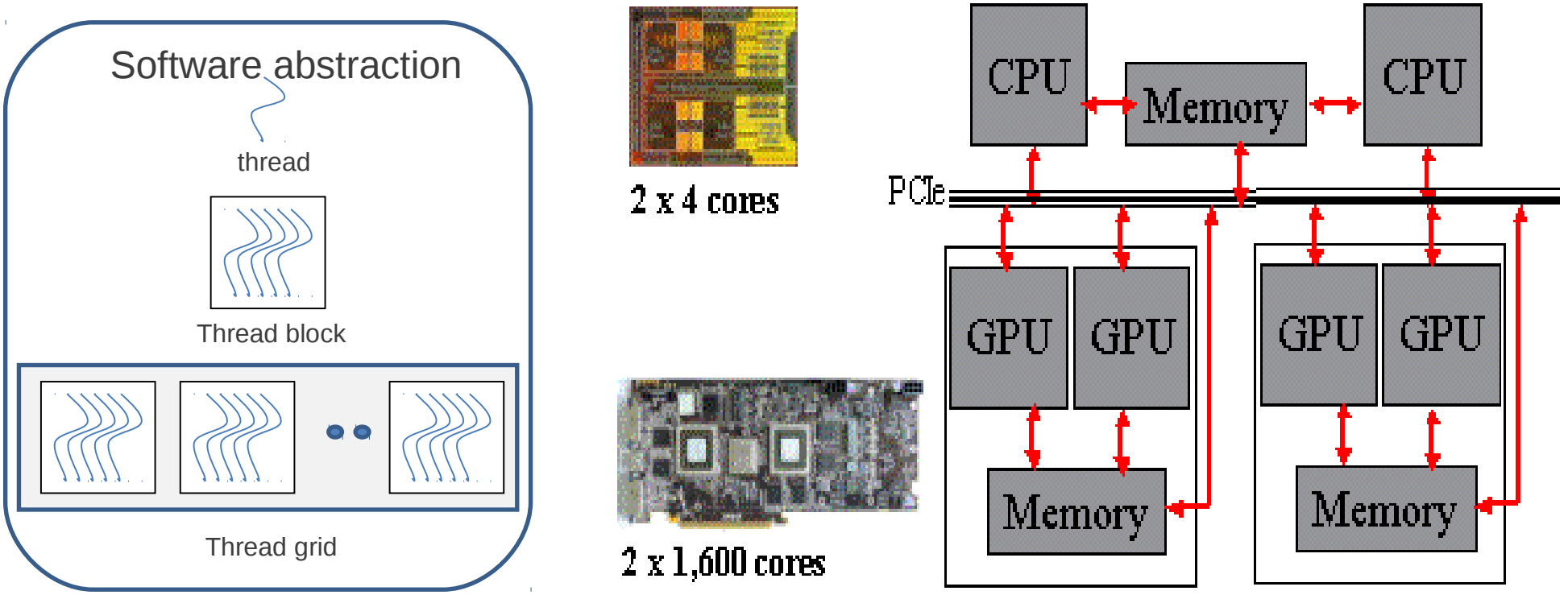





```
__global__ void add (int *a, int *b, int *c) {
    int tid = blockIdx.x;
    if (tid < N)
    c[tid] = a[tid] + b[tid];
}

int main(void) {
    int a[N], b[N], c[N], *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
    cudaMalloc((void**)&dev_a, N * sizeof(int)));
    // copy array 'a' to the GPU
    cudaMemcpy(dev_a, a, N * sizeof(int),  cudaMemcpyHostToDevice));
     add<<<N,1>>>( dev_a, dev_b, dev_c );
     // copy back from the GPU to the CPU
     cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost));
     // free the memory allocated on the GPU
     cudaFree(dev_a);
}
```

- **Questões importantes**
  - Programação eficiente da arquitetura híbrida
    1. Decidir se o paralelismo da aplicação pode ser eficientemente explorado na GPU?
    2. Se pode, de que forma usar a CPU para reduzir o tempo total de execução da aplicação?



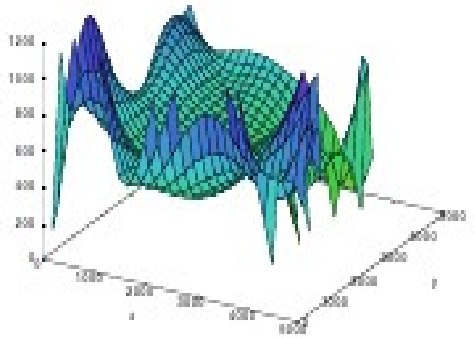

**Mapping GPU computing onto a CPU**
(FARBER, R. CUDA – application design and development. NVIDIA, 2011)

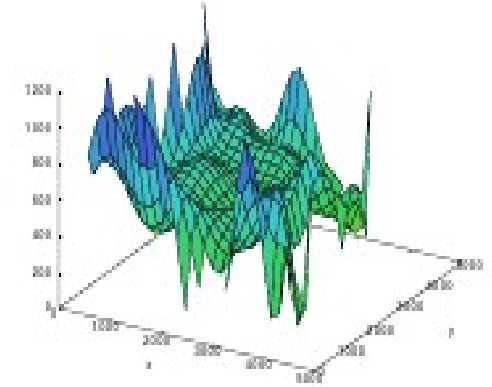
```
#include <iostream>   #include <vector>

int main()  {
  const int N=50000;

  // task 1: create the array
  vector<int> a(N);

  // task 2: fill the array
  for(int i=0; i < N; i++) a[i]=i;

  // task 3: calculate the sum of the array
  int sumA=0;
  for(int i=0; i < N; i++) sumA += a[i];

  // task 4: calculate the sum of 0 .. N-1
  int sumCheck=0;
  for(int i=0; i < N; i++) sumCheck += i;

  // task 5: check the results agree
  if(sumA == sumCheck) cout << "Test Succeeded!" << endl;
  else {cerr << "Test FAILED!" << endl; return(1);}
  return(0);
}
```

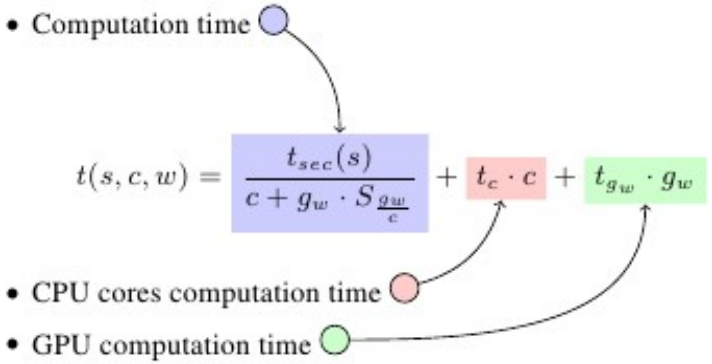
```
__global__ void fillKernel(int *a, int n) {
  int tid = blockIdx.x*blockDim.x + threadIdx.x;
  if (tid < n) a[tid] = tid;
}
void fill(int* d_a, int n) {
  int nThreadsPerBlock= 512;
  int nBlocks= n/nThreadsPerBlock + ((n%nThreadsPerBlock)?1:0);
  fillKernel <<< nBlocks, nThreadsPerBlock >>> (d_a, n);
}

int main() {
  const int N=50000;
  // task 1: create the array
  thrust::device_vector<int> a(N);
  // task 2: fill the array using the runtime
  fill(thrust::raw_pointer_cast(&a[0]),N);
  // task 3: calculate the sum of the array
  int sumA= thrust::reduce(a.begin(),a.end(), 0);
  // task 4: calculate the sum of 0 .. N-1
  int sumCheck=0;
  for(int i=0; i < N; i++) sumCheck += i;
  // task 5: check the results agree
  if(sumA == sumCheck) cout << "Test Succeeded!" << endl;
  else { cerr << "Test FAILED!" << endl; return(1);}
  return(0);
}
```

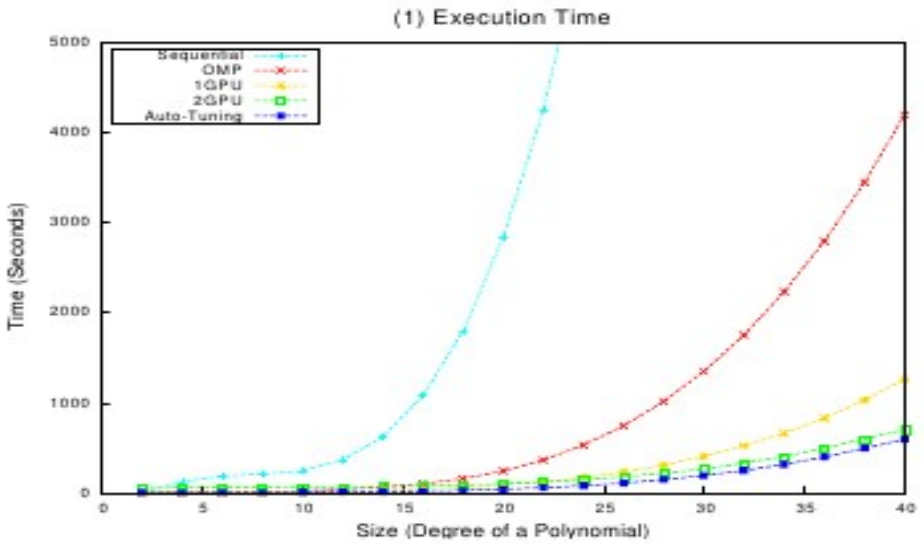
- Modelos de desempenho e frameworks
  - ✔ FastFlow, Muesli, SkePU, HMPP etc.
  - ✔ *Auto-tuning methodology to represent landform attributes on multicore and multi-GPU systems.*

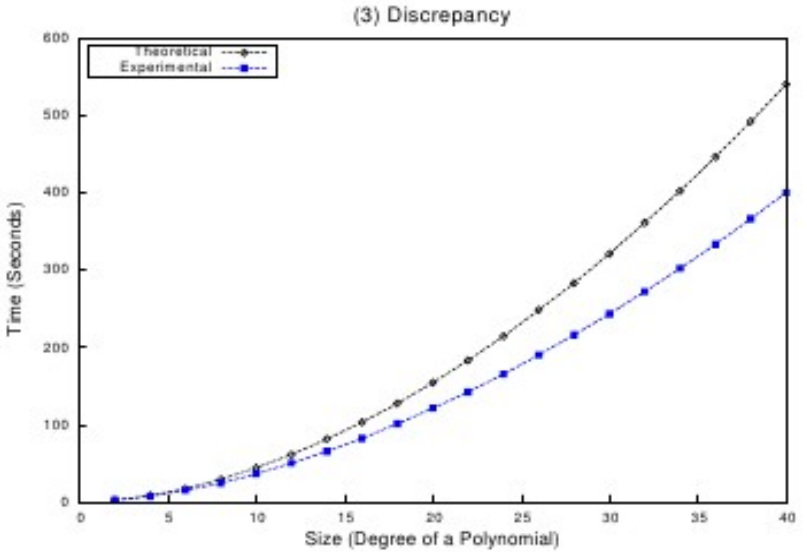
    (BORATTO, M.; BARRETO, M.; ALONSO, P. WoSiDA 2012 & PMAM 2013).



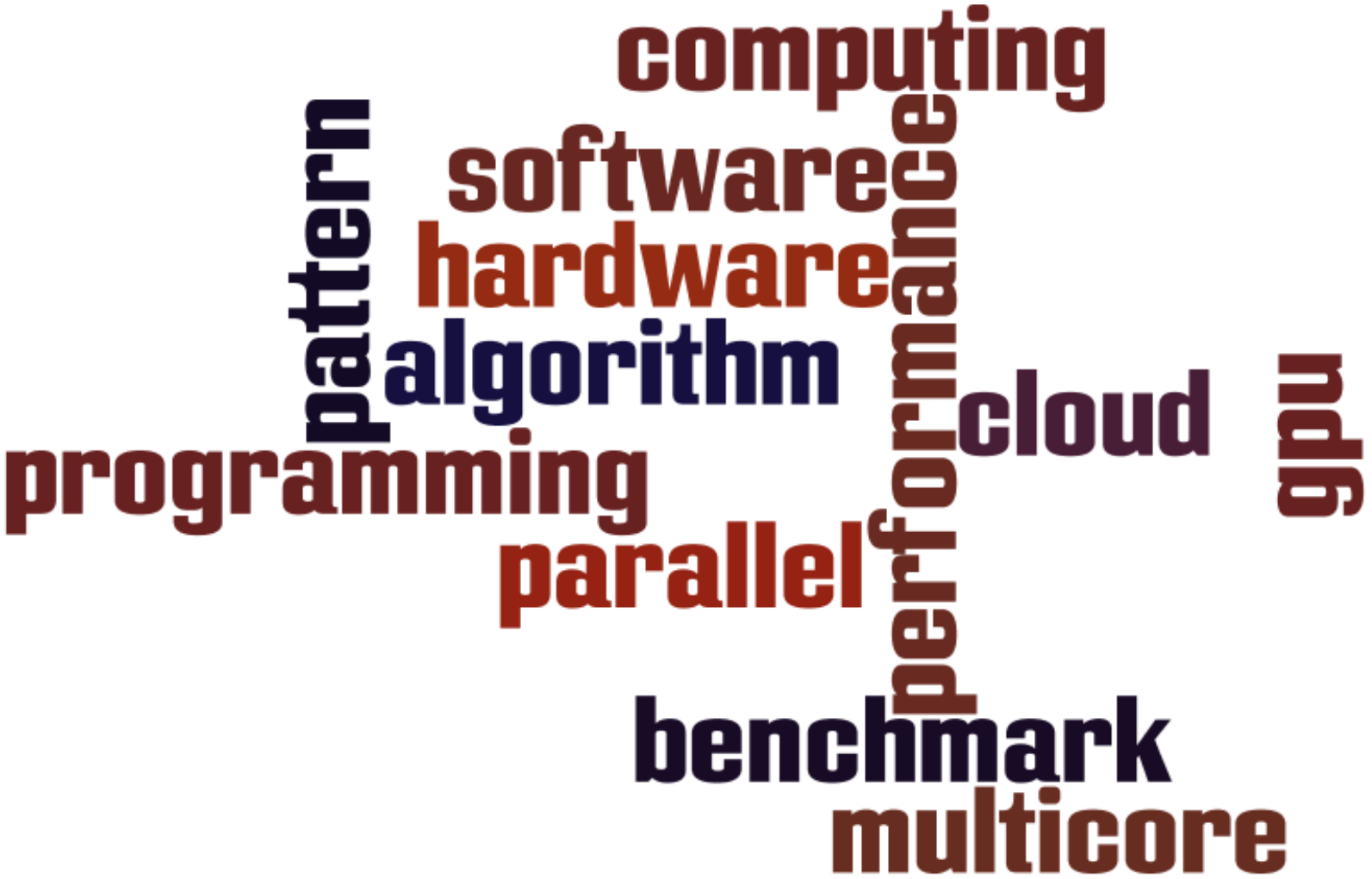




# Padrões paralelos

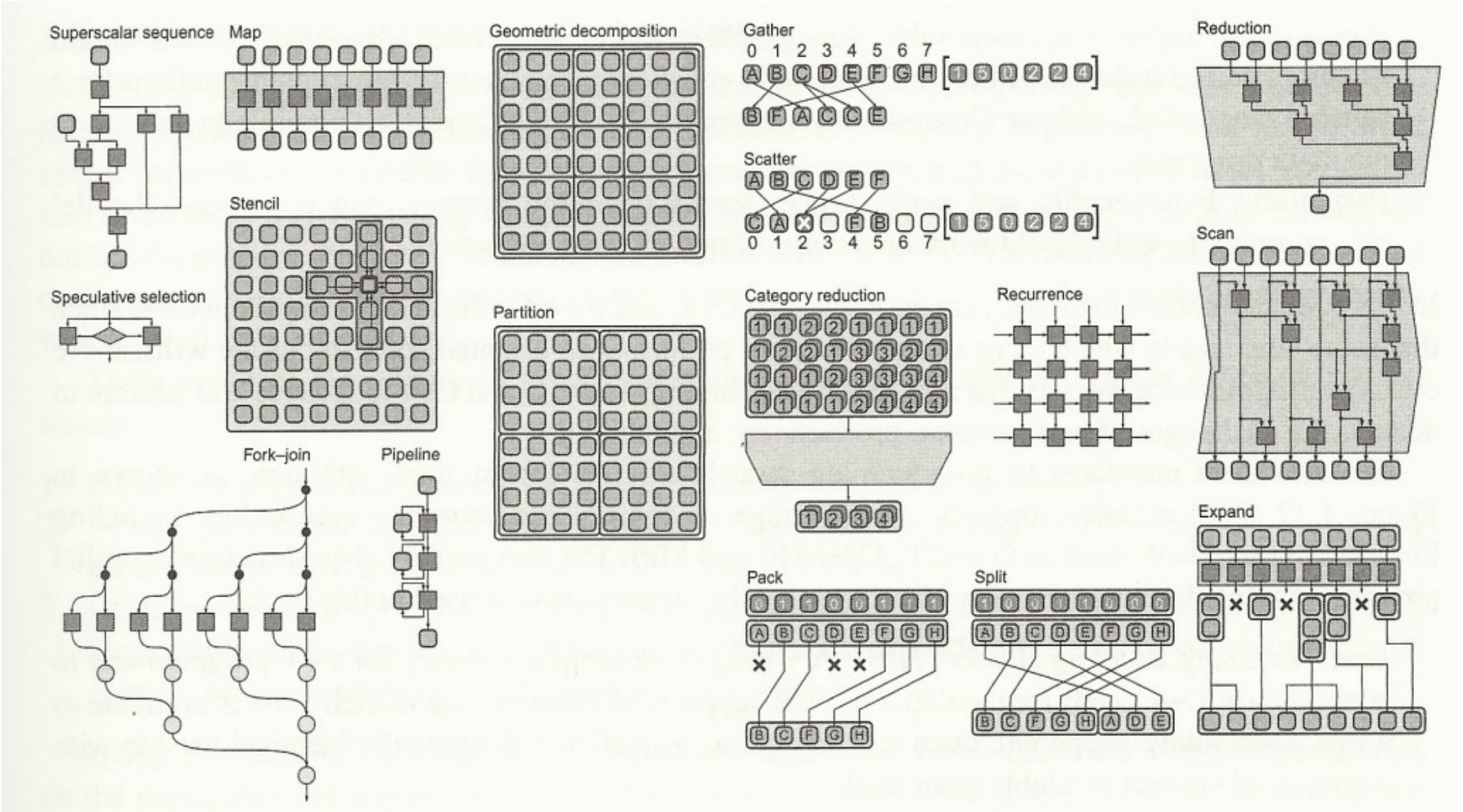


- **Padrões (projeto / programação) paralelos**
  - Estruturas algorítmicas comumente encontradas em programas paralelos, as quais representam um “esqueleto” para a organização do código paralelo.
  - Combinação recorrente de distribuição de tarefas e de acesso a dados para resolver um determinado problema de forma paralela.
  - Vantagens do emprego de padrões:
    - Provêem **eficiência** e **escalabilidade** para diferentes tipos de hardware, com diferentes capacidades computacionais.
    - A maioria dos padrões fornece **resultados determinísticos**.

- **Padrões (projeto / programação) paralelos**
  - Níveis de abstração:
    - Padrões de projeto (*design patterns*)
      - Descrição mais generalista, de alto nível.
    - Estratégias para algoritmos paralelos
      - Abordagem focada na identificação de concorrência nos algoritmos e de que forma implementá-la no software.
    - Execução paralela
      - Como executar as estruturas concorrentes do software na arquitetura híbrida.

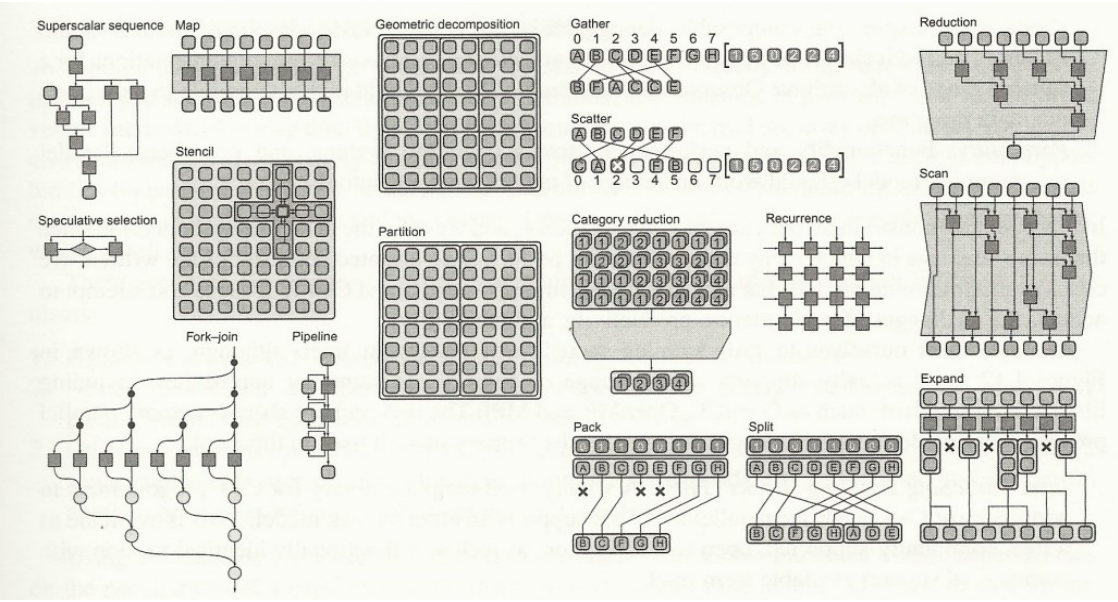
- Estratégias para algoritmos paralelos
  - Semântica
    - Corresponde a um arranjo de tarefas e dependências de dados entre elas.
    - Define de que forma o padrão é usado como um bloco básico para a construção do algoritmo.
    - Esconde detalhes de implementação específicos para uma linguagem ou arquitetura.
  - Implementação
    - Controle de granularidade.
    - Uso de recursos (ex. cache).
    - Diferentes implementações geram resultados diferentes, mas sempre com a mesma semântica!

- **Padrões (projeto / programação) paralelos**

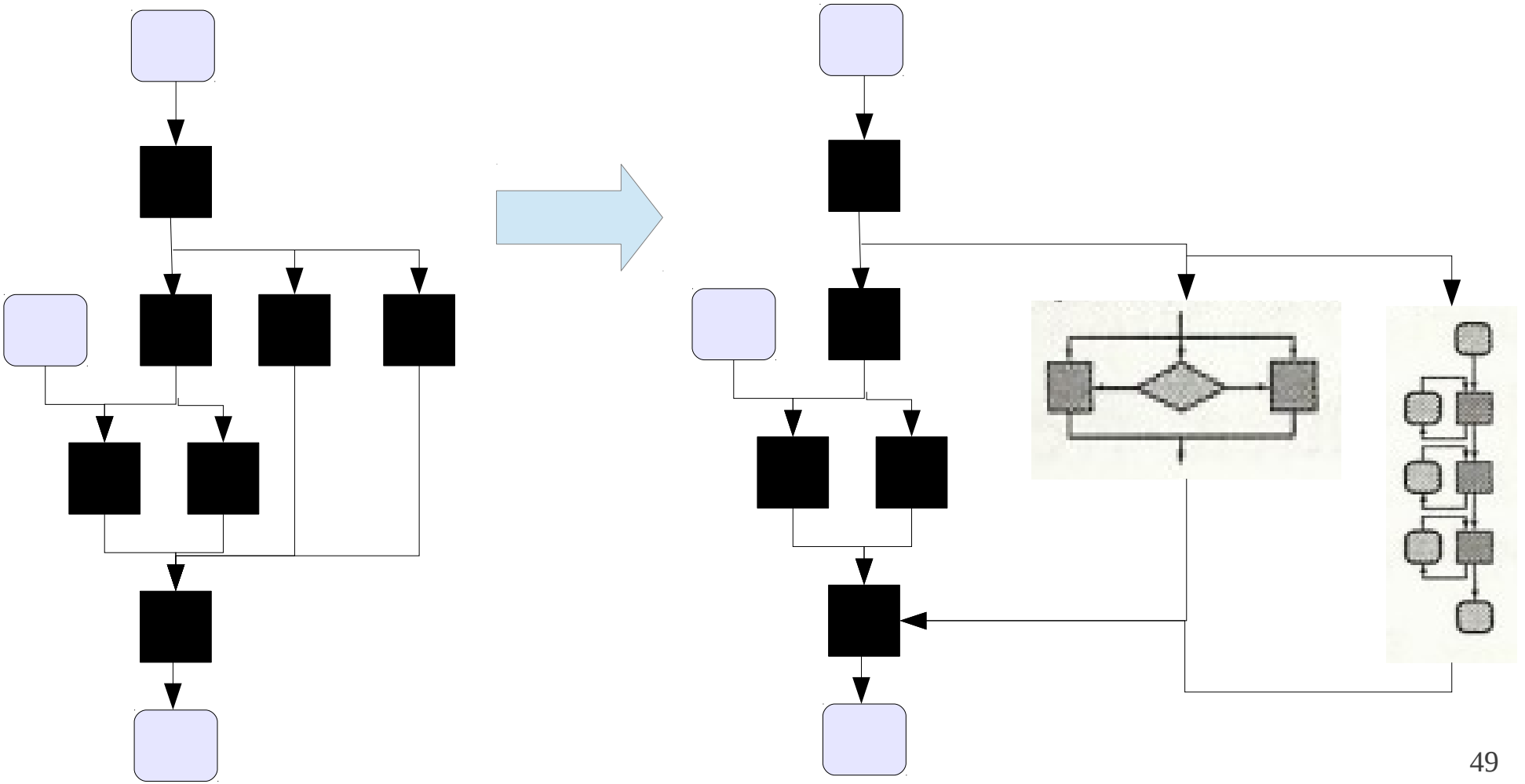



**Overview of parallel patterns.**
(McCOOL, M. et al. Structured parallel programming, Morgan Kaufmann, 2012 )

- **Classificação (controle de fluxo X dados)**
  - Aninhamento (composição)
  - Controle de fluxo sequencial
  - Controle paralelo
  - Gerência de dados sequenciais
  - Gerência de dados paralelos
  - Outros padrões
  - Padrões não-determinísticos



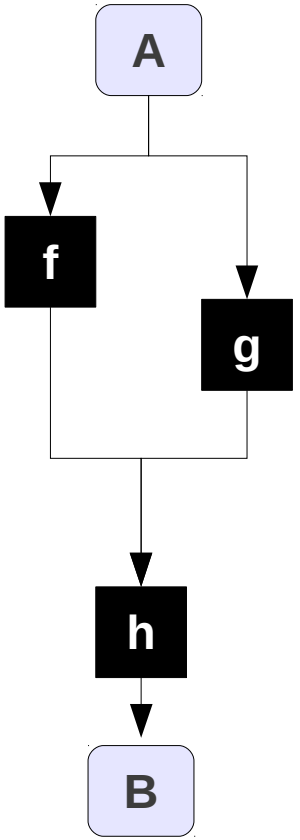
- **Aninhamento (*nesting*)**
  - Padrão de composição, que permite a um código paralelo usar diferentes padrões de forma hierárquica.
  - Qualquer bloco de tarefas pode ser substituído por outro padrão, preservando entradas, saídas e dependências.



- **Controle de fluxo sequencial**

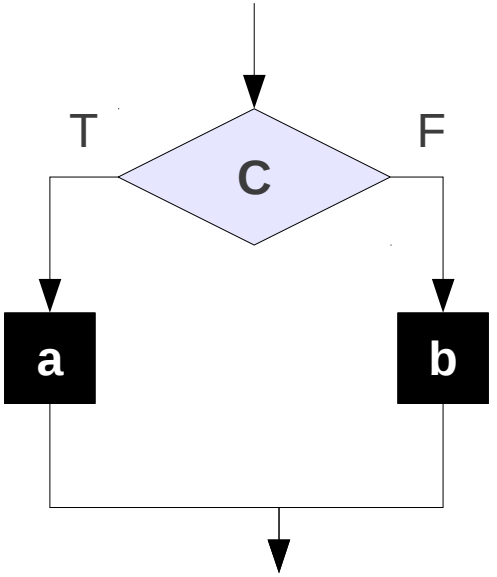
## Sequence

```
1. T = f(A);  1. T = f(A);
2. S = g(T);  2. S = g(A);
3. B = h(S);  3. B = h(S,T);
```


## Selection

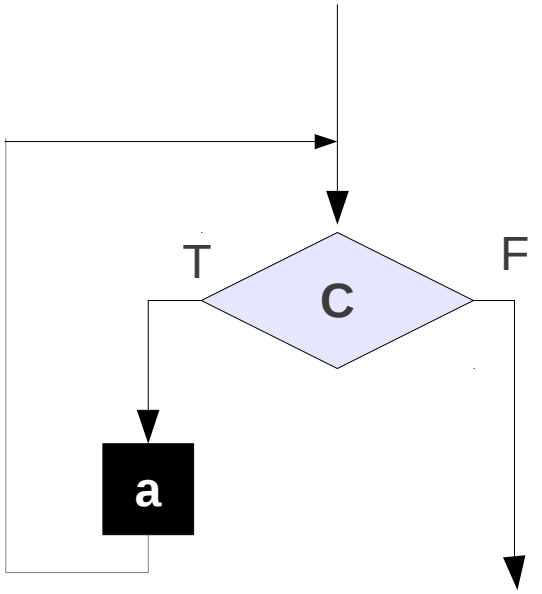
```
1. if (c) {
2.     a;
3. } else {
4.     b;
5. }
```




## Iteration

```
1. for(i=0; i<n; i++) {
2.     a;
3. }
```

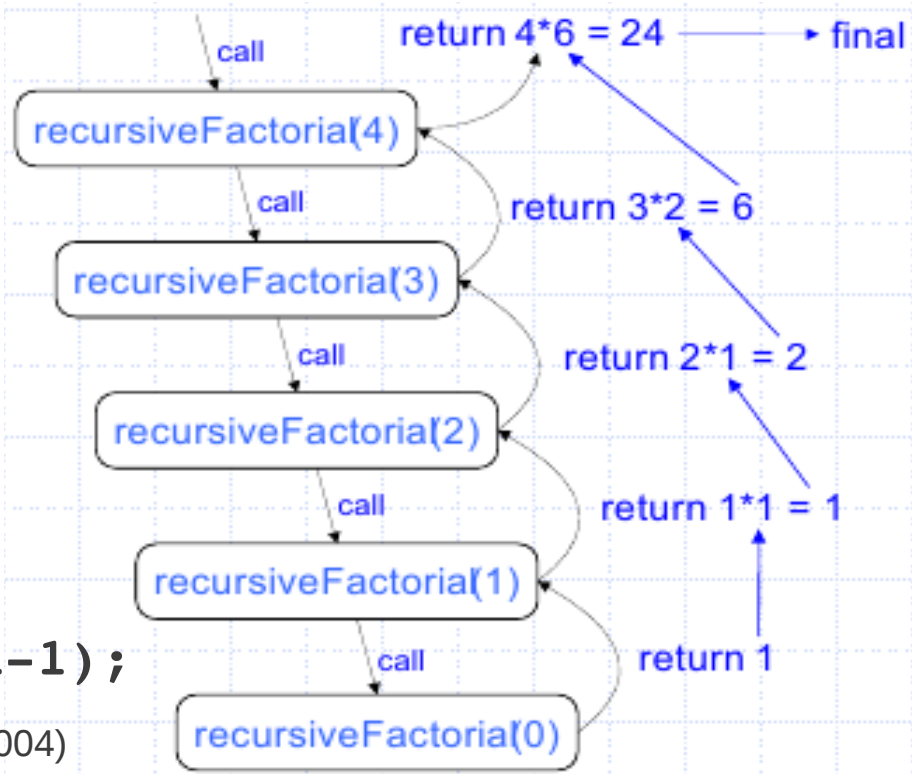
```
1. while (c)  {
2.     a;
3. }
```




dado  tarefa

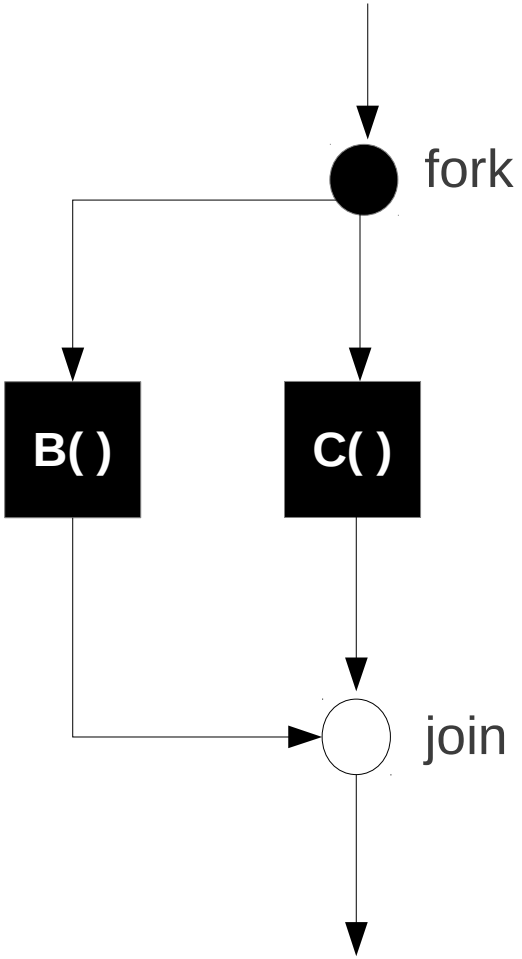
- **Controle de fluxo sequencial**
  - Recursion
    - Aninhamento dinâmico que permite a uma função chamar a si mesma, de forma direta ou indireta.
      - Tail recursion: pode ser convertida em interação, com a instância da função chamada retornando um valor à instância da função chamadora.

```
public static int
       recursiveFactorial(int n)
{
  if  (n == 0)  // basis case
    return 1;
  else          // recursive case
    return n * recursiveFactorial(n-1);
}
```




(GOODRICH, T. Using recursion. 2004)

- **Controle paralelo -** Fork-join
  - Um fluxo “pai” pode criar (*fork*) vários fluxos paralelos e então esperar (*join*) pelo término desses fluxos antes de prosseguir com sua execução.




**Cilk Plus**

```
cilk_spawn B( );
C( );
cilk_sync;
```

```
for (i=0; i<n; i++)
   if (a[i]!=0)
      cilk_spawn f(a[i]);
cilk_sync;
```

- **Controle paralelo -** Fork-join

**TBB (Threading Building Blocks)**

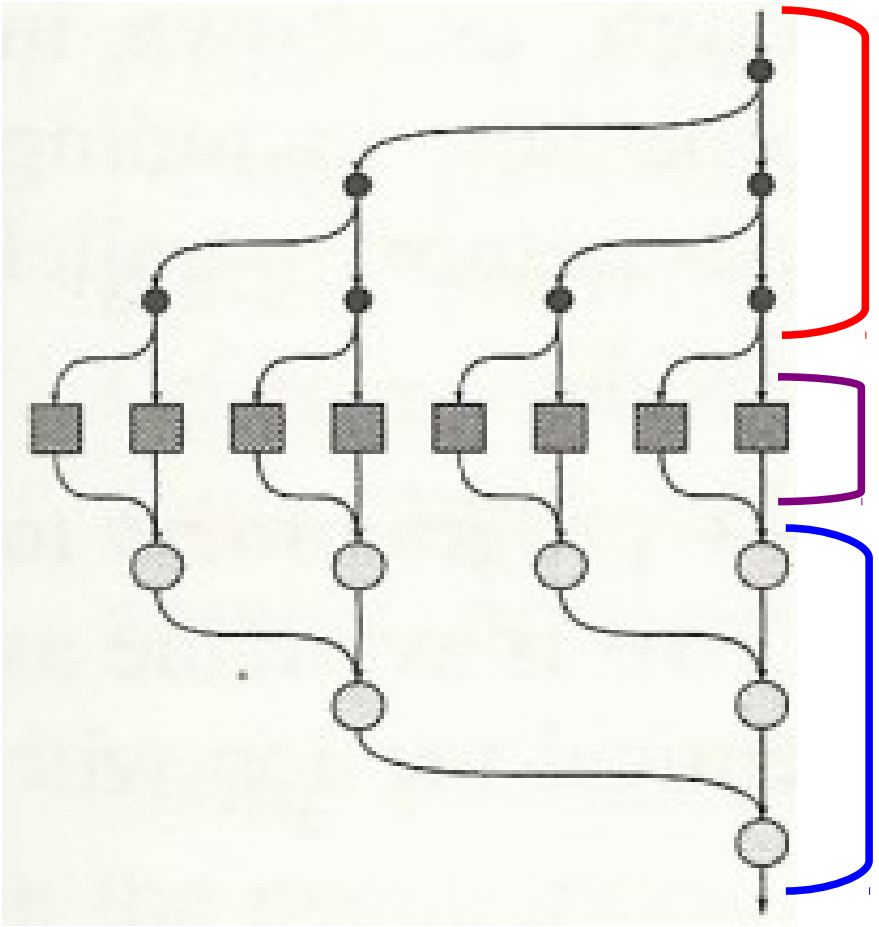
```
tbb::parallel_invoke(B, C);  // até 10 argumentos
```

```
task_group g;
for (i=0; i<n; i++)
  if (a[i]!=0)
     g.run( [=,&a]{f(a[i]);} );
g.wait;
```

**OpenMP**

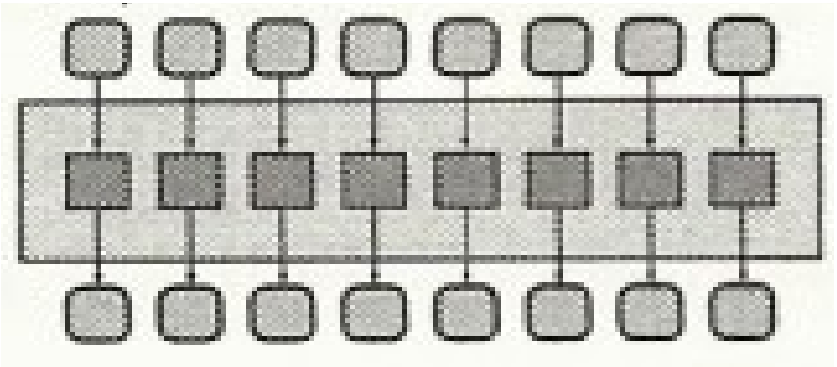
```
#pragma omp task
B( );
C( );
#pragma omp taskwait
```

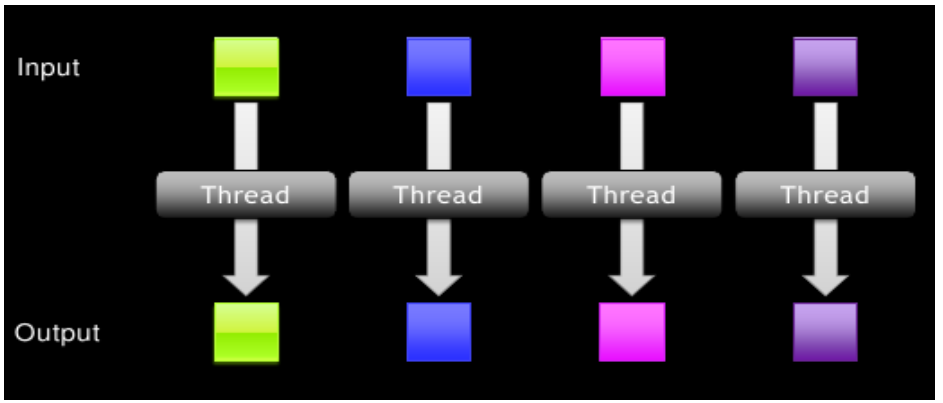
- **Controle paralelo -** Fork-join
  - Pode ser usado para implementar outros padrões: **map, reduce, recurrence** e **scan** e algoritmos de **divisão e conquista**.
    - Multiplicação de polinômios, ordenação (*quicksort*), algoritmos de gerência de localidade em cache etc.



```
void DivideAndConquer(Problem P)
{
  if (P is base case) {
    Solve P;
  } else {
    Divide P into K subproblems;
    Fork to conquer each K;
    Join;
    Combine subsolutions;
}
```

- **Controle paralelo -** Map
  - Uma função é aplicada a todos os elementos de uma coleção, podendo gerar uma nova coleção.
  - Exemplo  mais comum:
    - laços com iterações independentes e número de repetições conhecido, no qual o valor de cada computação depende somente do índice e do valor lido a partir desse índice.






(WANG, P. Data prallel programming with patterns, NVIDIA)

- **Controle paralelo -** Map
  - SAXPY (multiplicação escalar e soma de vetores)
    - ***y = ax + y***, sendo ***x*** e ***y*** vetores e ***a*** o valor escalar.

C

```
void saxpy_seq(
  int n,            // tamanho dos vetores
  float a,          // valor escalar
  const float x[], // vetor x
  float y[])        // vetor y
{
  for(i=0; i<n; ++i)
   y[i] = a * x[i] + y[i];
}
```

**TBB (Threading Building Blocks)**

```
tbb::parallel_for(
  tbb::blocked_range<int>(0,n),[&]tbb::blocked_range<int> r)
{
   for(i=r.begin(); i!= r.end(); ++i)
        y[i] = a * x[i] + y[i];
}
```

- **Controle paralelo -** Map
  - SAXPY (multiplicação escalar e soma de vetores)
    - ***y = ax + y***, sendo *x* e *y* vetores e *a* o valor escalar.

**Cilk Plus**

```
cilk_for(i=0; i<n; ++i)
   y[i] = a * x[i] + y[i];
}

// alternativa:
// notação de array
Y[0:n] = a * x[0:n] + y[0:n];
```

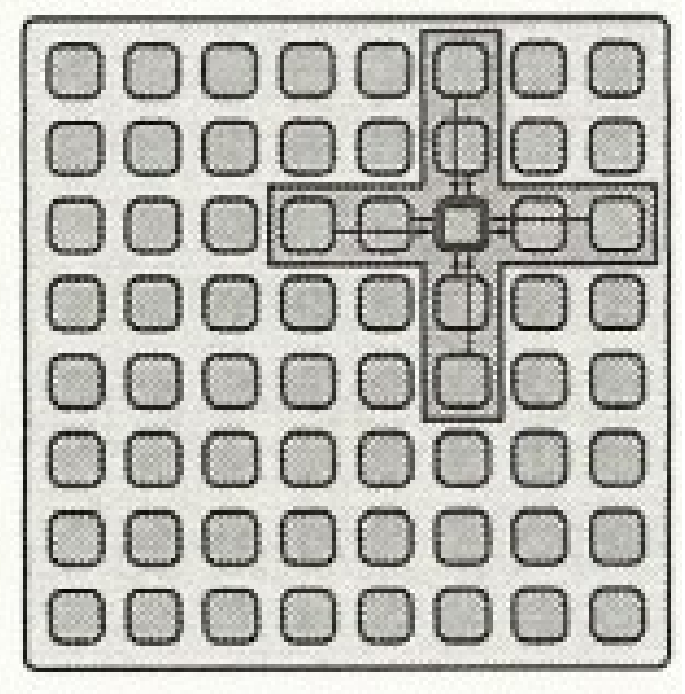
**OpenMP**

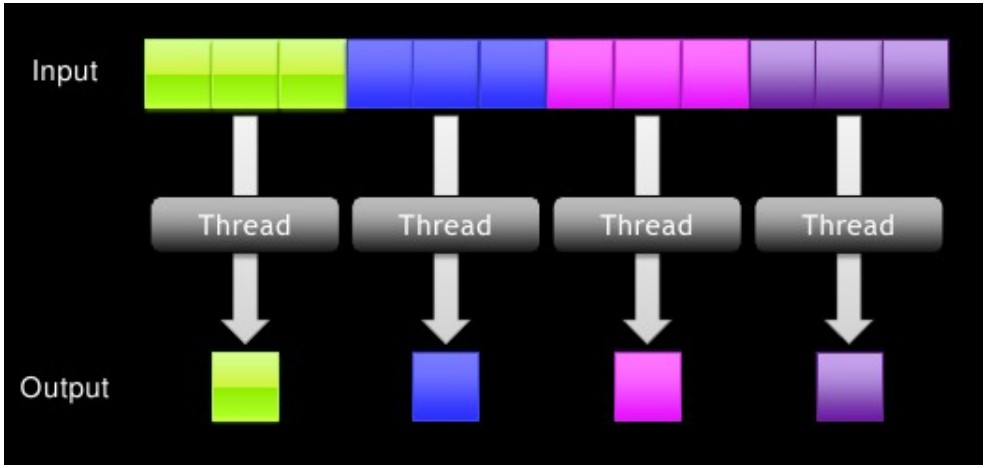
```
#pragma omp parallel for
 for(i=0; i<n; ++i)
   y[i] = a * x[i] + y[i];
}
```

**OpenCL**

```
_kernel void saxpy_opencl(
   _constant float a,
   _global float* x,
   _global float* y )
{
   int i = get_global_id(0);
   y[i] = a * x[i] + y[i];
}
```

- **Controle paralelo -** Stencil
  - Variação do padrão **map** no qual uma função elementar pode acessar um item de uma coleção e também os seus vizinhos.
  - Ex.: filtragem de imagens (convolução, mediana, redução de ruídos etc), simulação de fluídos, equações diferenciais, autômatos celulares, álgebra linear.
    - Jacobi, Gauss-Seidel, elementos finitos etc.

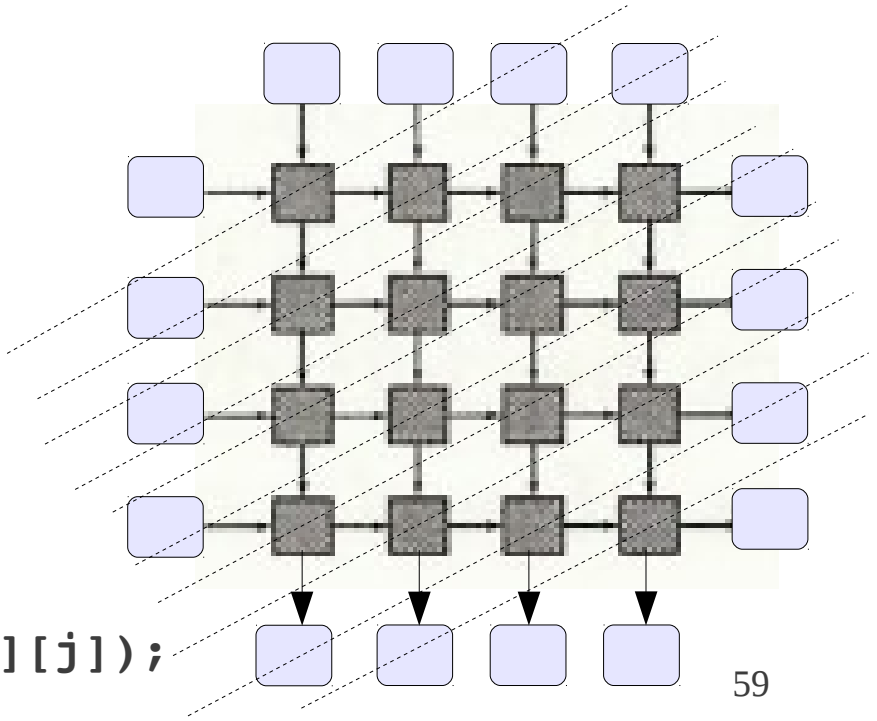




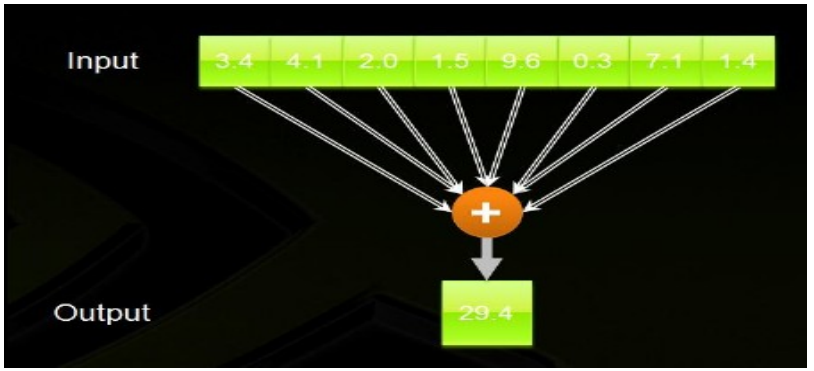

(WANG, P. Data prallel programming with patterns, NVIDIA)

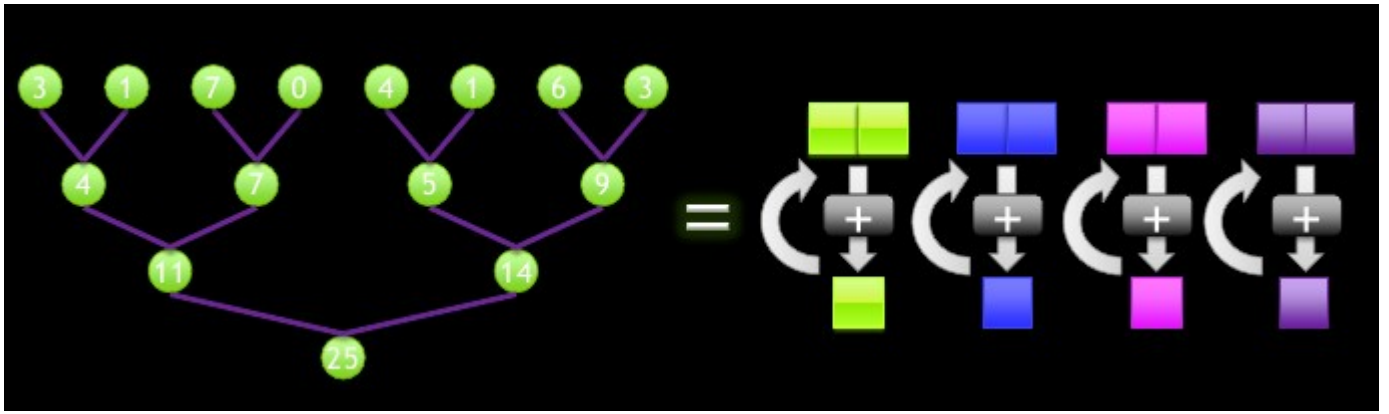
- **Controle paralelo -** Recurrence
  - Variação do padrão **map** para tratar casos mais complexos, nos quais as iterações do laço dependem umas das outras.
  - Se a dependência (*offset*) entre os elementos for constante, o padrão permite a execução paralela de tarefas (como no padrão stencil).
    - Exemplos mais comuns: fatoração de matrizes, processamento de imagens, equações diferenciais parciais (PDE) etc.

```
void my_recurrence(
  size_t v,
  size_t h,
  const float a[v][h],
  float b[v][h] )
{
  for (int i=1; i<v; ++i)
   for (int j=1; i<h; ++j)
    b[i][j] = f(b[i-1][j], b[i][j-1], a[i][j]);
```



- **Controle paralelo -** Reduction
  - Combina os elementos de uma coleção em um único elemento, através de uma função associativa.
    - Soma, multiplicação, mínimo, máximo, booleana etc.
  - Ex.: sistemas de equações lineares (gradiente conjugado), métodos de Monte Carlo, codificação de vídeo, multiplicação de matrizes.




(WANG, P. Data prallel programming with patterns, NVIDIA)

C++

```
float sprod(
  size_t n,
  const float a[],
  const float b[]
) {
  float res = 0.0f;
  for (size_t i=0; i<n; i++)
     res += a[i] * b[i];
}
```

- **Controle paralelo -**

Reduction

TBB (Threading Building Blocks)

```
float tbb_sprod(size_t n, const float *a, const float *b) {
 return tbb::parallel_reduce(tbb::blocked_range<size_t>(0,n),
 float(0),
 [=]( tbb::blocked_range<size_t>& r, float in) {
   return std::inner_product(a+r.begin(), a+r.end(),
                             b+r.begin(), in);
 },
 std::plus<float>()
 );  // lambda expression
}
```

- **Controle paralelo -** Reduction

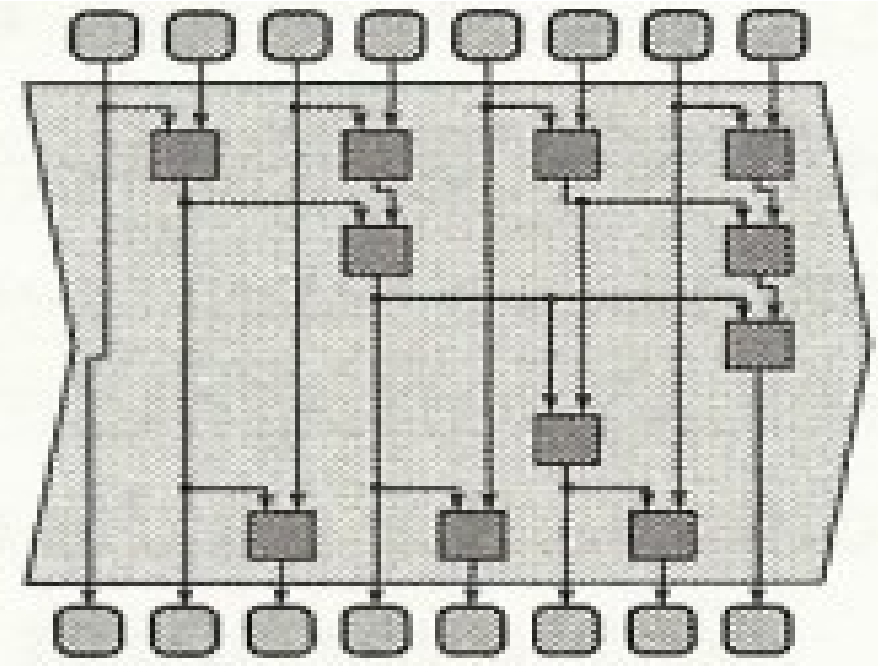
**Cilk Plus**

```
float cilkplus_sprod(
  size_t n,
  const float a[],
  const float b[] )
{
  return _sec_reduce_add(a[0:n] * b[0:n]);
}
```

**OpenMP**

```
float openmp_sprod(size_t n, const float *a,
                   const float *b )
{
  float res=0.0f;
  #pragma omp parallel for reduction(+:res)
    for(i=0; i<n; ++i)
       res += a[i] * b[i];
}
```

- **Controle paralelo -** Scan
  - Produz todas as reduções parciais de uma sequência de entrada, resultando em uma nova sequência.
  - Pode ser inclusivo (N elementos) ou exclusivo (N-1 elementos).
  - Fold: usa função sucessora para avançar de um estado anterior para um novo estado, dada uma nova entrada.



# Scan (Prefix Sums)

- Given array $A = [a_0, a_1, \ldots, a_{n-1}]$
  and a binary associative operator $\oplus$ with identity $I$,

  scan($A$) = $[I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})]$ (exclusive)

  scan($A$) = $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})]$ (inclusive)

- Example: if $\oplus$ is +, then

  Scan([3 1 7 0 4 1 6 3]) = [0 3 4 11 11 15 16 22] (exclusive)

  Scan([3 1 7 0 4 1 6 3]) = [3 4 11 11 15 16 22 25] (inclusive)

(SENGUPTA,S et al. Scan primitives for GPU computing. Graphics Hardware, 2007).)

C++

- **Controle paralelo -**

Scan

```
template<typename T, typename C>
void inclusive_scan(
  size_t n,      // número de elementos
  const T a[],   // dados de entrada
  T A[],         // dados de saída
  C combine,     // função de combinação
  T initial      // valor inicial
) {
  for (size_t i=0; i<n; ++i)
    initial = combine(initial, a[i]);
    A[i] = initial;
}
```

```
void parallel_scan(const Range& range, Body& body);
```

TBB

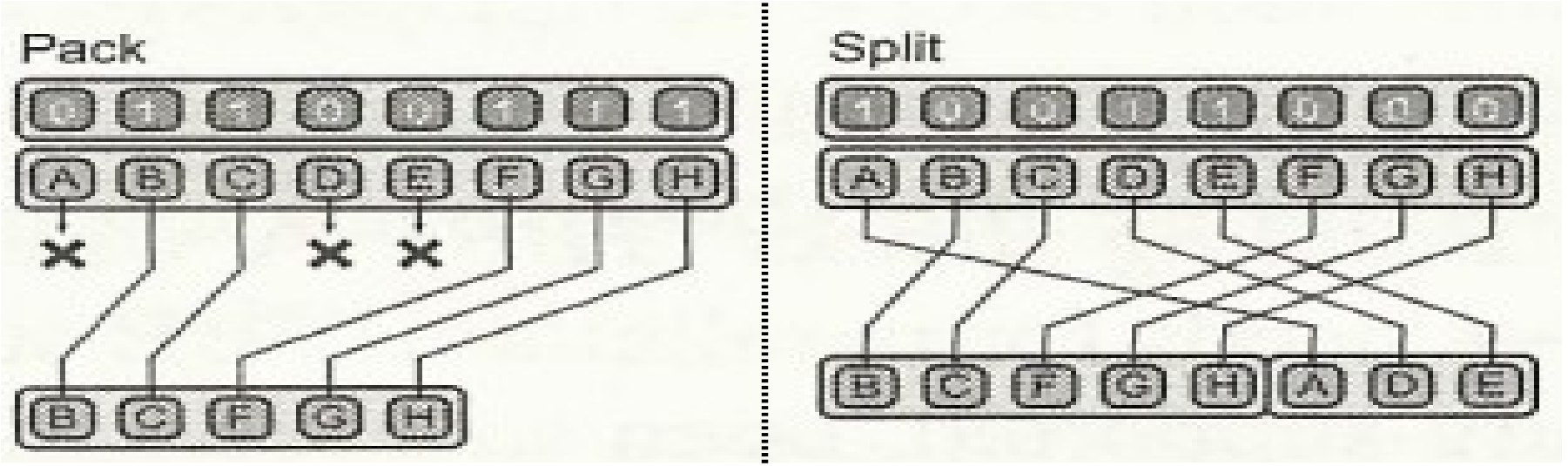
```
void cilk_scan(size_t n, T initial, size_t tilesize,
R reduce, C combine, S scan);
```

Cilk Plus

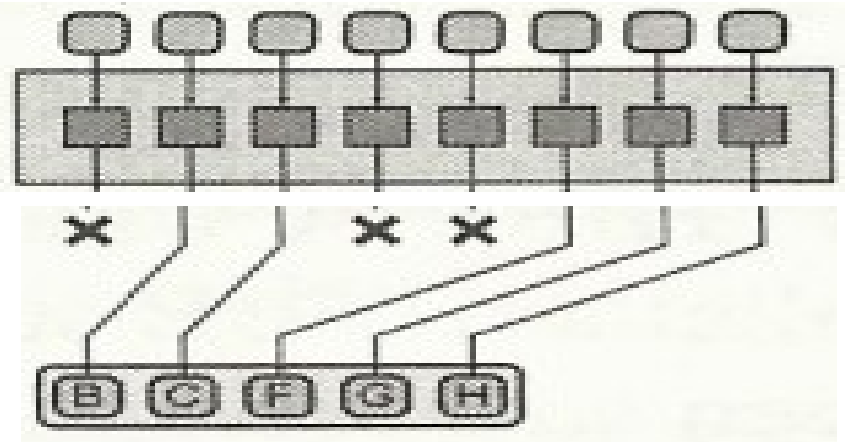
- **Gerência de dados sequenciais**
  - Operações de leitura, escrita e cópia.
  - Como os dados são armazenados e compartilhados?
  - Padrões:
    - Random read and write
      - Ponteiros => *aliasing*
    - Stack allocation
      - Alocação dinâmica, LIFO.
    - Heap allocation
    - Closures
      - Objetos que podem ser definidos e gerenciados como dados => uso de funções lambda.
    - Objects

- **Gerência de dados paralelos**
  - Organizar o acesso paralelo aos dados, para evitar *data races* => saber como e quando múltiplas threads podem modificar o mesmo dado.
  - Padrões:
    - Pack
    - Pipeline
    - Geometric decomposition
    - Gather
    - Scatter

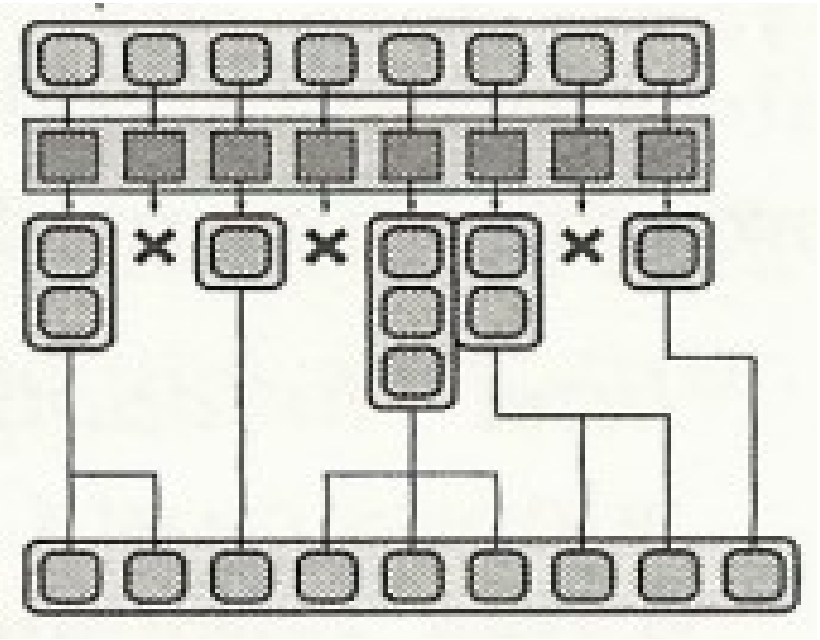
- **Gerência de dados paralelos –** pack (compact)
  - Usado para eliminar elementos não usados dentro de uma coleção, gerando uma nova sequência somente com os elementos válidos (ex.: saídas inválidas de outros padrões).
    - Gerência de memória, controle de fluxo SIMD.
  - Pack = scan + scatter condicional
  - Padrões relacionados: unpack, split (partition), unsplit.



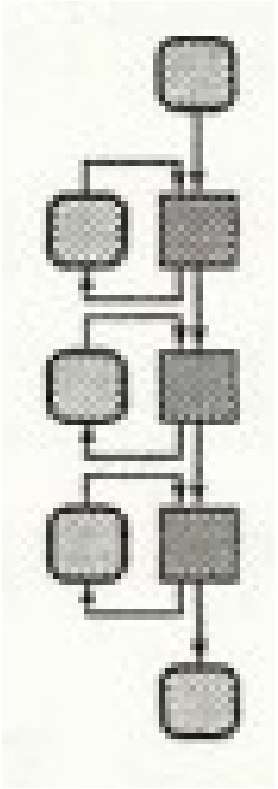
- **Gerência de dados paralelos**
  - Map + Pack
    - Ex.: detecção de colisões




  - Expand
    - Cada elemento pode produzir zero ou mais saídas, as quais são agrupadas em ordem.



- **Gerência de dados paralelos -** pipeline
  - Conecta tarefas em uma relação produtor-consumidor.
  - Todos os estágios são ativos e podem manter o estado do seu processamento.
  - Podem ter estágios sequenciais ou paralelos.




```
void serial_pipeline()
{
  while (t = F())
  {
     u = G(t);
     H(u);
  }
}
```

- **Gerência de dados paralelos - pipeline**

**TBB (Threading Building Blocks)**

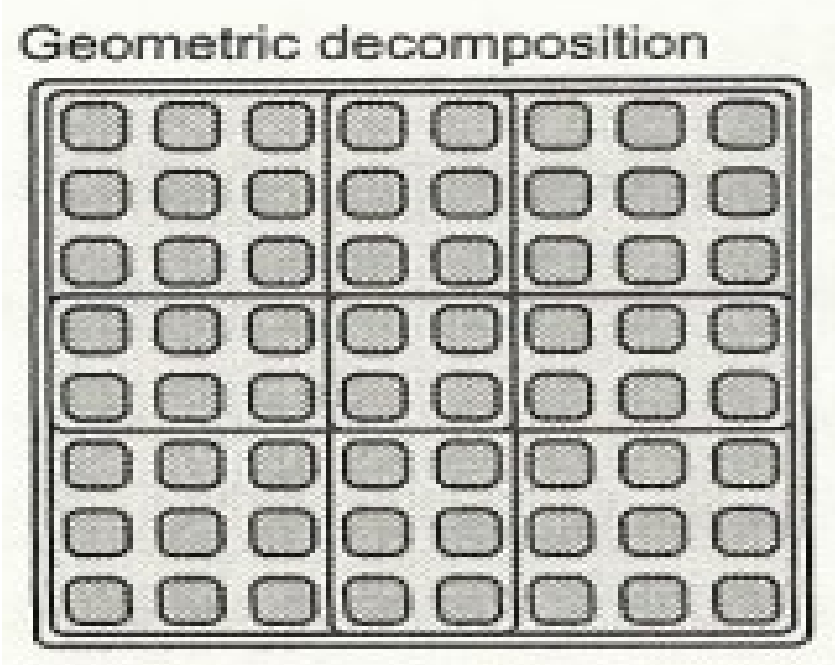
```
void tbb_sps_pipeline(size_t ntoken) {
  tbb::parallel_pipeline (
    ntoken,
    tbb::make_filter<void,T> (
      tbb::filter::serial_in_order,
      [&](tbb::flow_control& fc) -> T {
            T item = f();
            if (!item) fc.stop();
            return item;
      }
    ) &
    tbb::make_filter<T,U>( tbb::filter::parallel, g())
    &
    tbb::make_filter<U,void> (
      tbb::filter::serial_in_order, h() )
  );
}
```

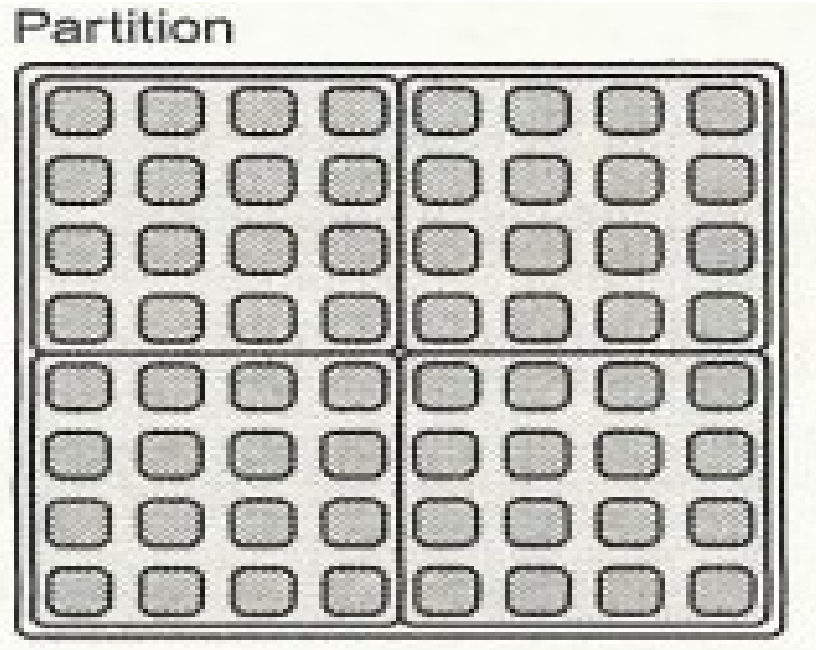
- **Gerência de dados paralelos - pipeline**

**Cilk Plus**

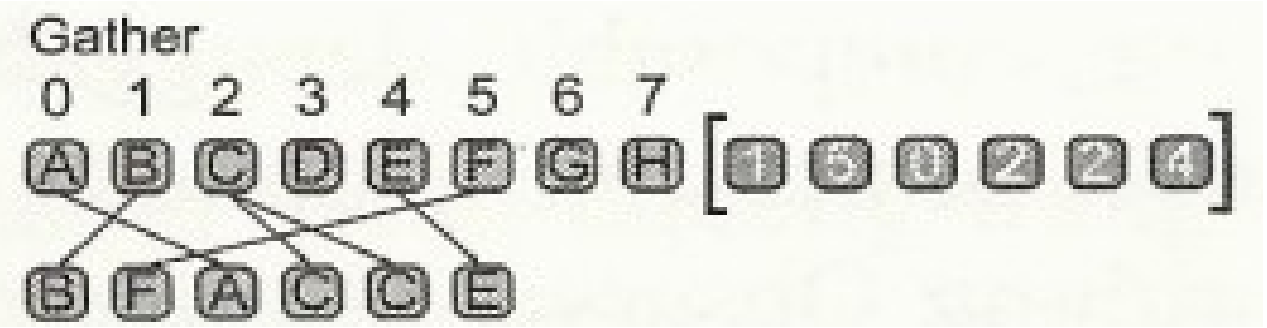
```
// function for third stage
extern void h_ (HState*, U u);
// mutable state for third stage
HState s;
// Reduce for third stage
reducer_consume<HState,U> sink (&s, h_ );

void stage2(T t) {
  U u = g(t);       // second stage
  sink.consume(u); // feed item to third stage
}

void cilk_sps_pipeline() {
  while(T t = f())              // first stage is serial
     cilk_spawn stage2(t);    // spawn second stage
  cilk_sync;
}
```

- **Gerência de dados paralelos –** geometric decomposition
  - Divide os dados em subconjuntos, processa cada conjunto individualmente e depois combina os resultados.
  - Divisão e conquista => recursão serial => fork-join.
  - Sem sobreposição => partition / segmentation.
  - Sobreposição parcial => stencil

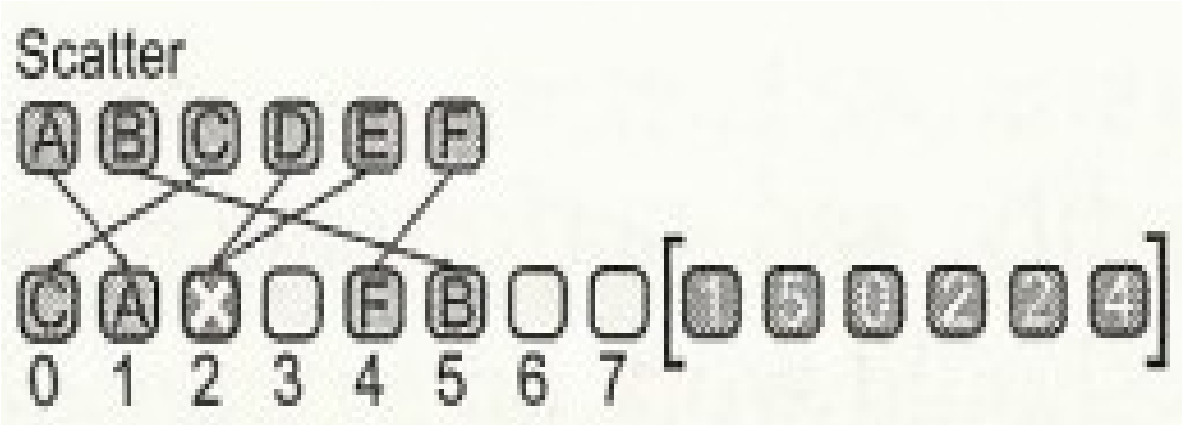


- **Gerência de dados paralelos –** gather
  - Seleciona um subconjunto de dados a partir de uma coleção de dados, com base nos índices selecionados.

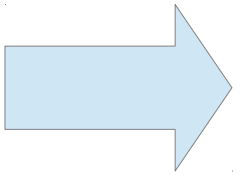


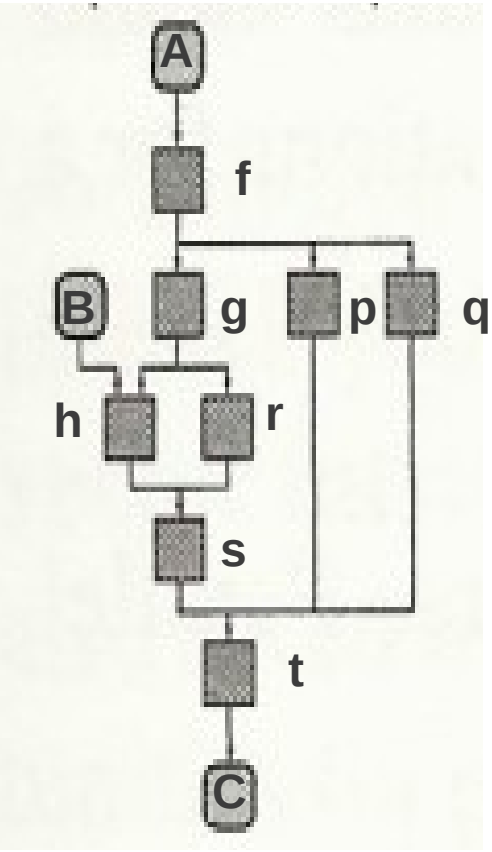

- **Gerência de dados paralelos –** scatter
  - Oposto do *gather* => um conjunto de dados de entrada e de índices é fornecido. Cada elemento da entrada é escrito num determinado local, com base no índice fornecido.



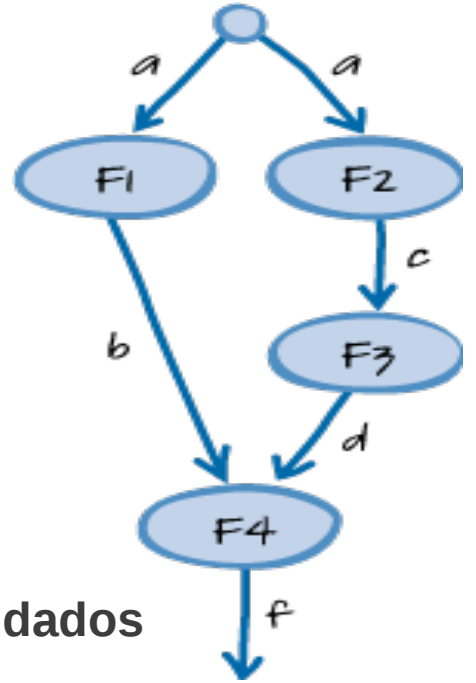
- **Outros padrões paralelos -** superscalar sequences
  - Tarefas são ordenadas de acordo com as dependências de dados.
  - Uma vez resolvidas as dependências, o sistema pode executar estas tarefas em paralelo ou em qualquer outra ordem diferente daquela especificada no código.

```
1. D = f(A);
2. E = g(D);
3. F = h(B,E);
4. G = r(E);
5. P = p(D);
6. Q = q(D);
7. H = s(F,G);
8. C = t(H,P,Q);
```



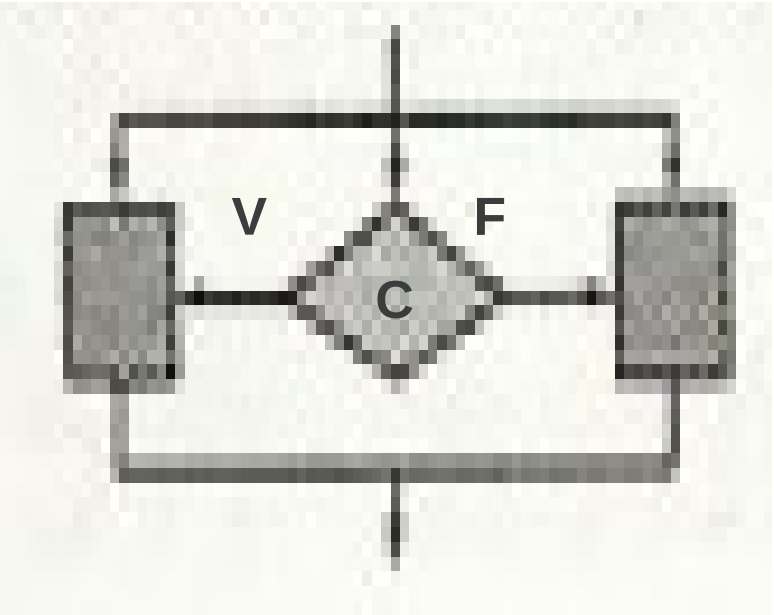


- **Outros padrões paralelos -** futures
  - Padrão semelhante ao fork-join, porém as tarefas não precisam estar aninhadas hierarquicamente.
  - Para cada tarefa criada, um objeto (*future*) é retornado, o qual é usado para gerenciar essa tarefa => esperar pelo seu término.
  - Usado para implementar grafos de tarefas mais complexos => *task cancellation, branch-and-bound, speculative selection.*

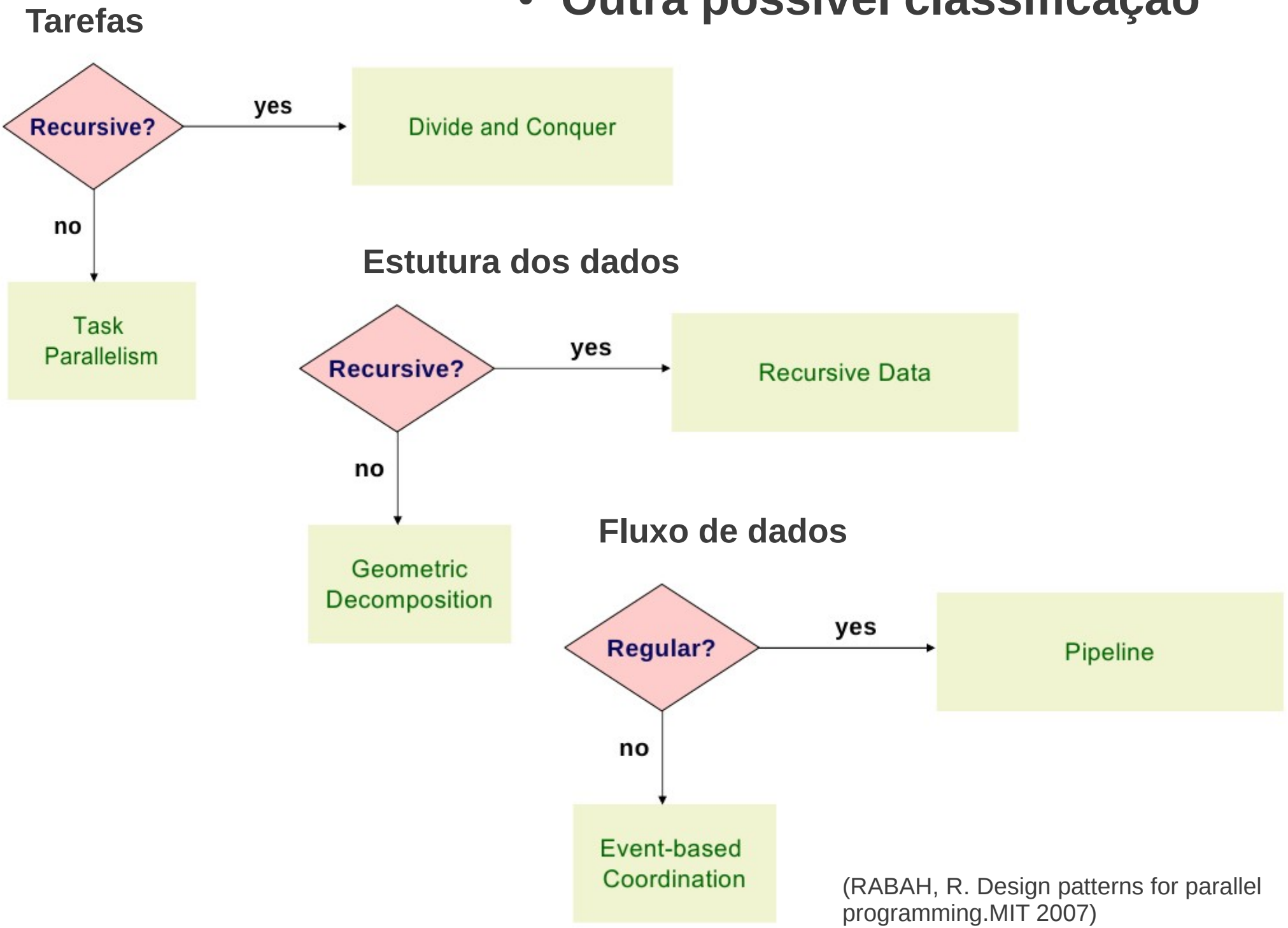



**Ordenação de tarefas de acordo com dependência de dados**
(MILLER, A. Patterns for parallel programming)

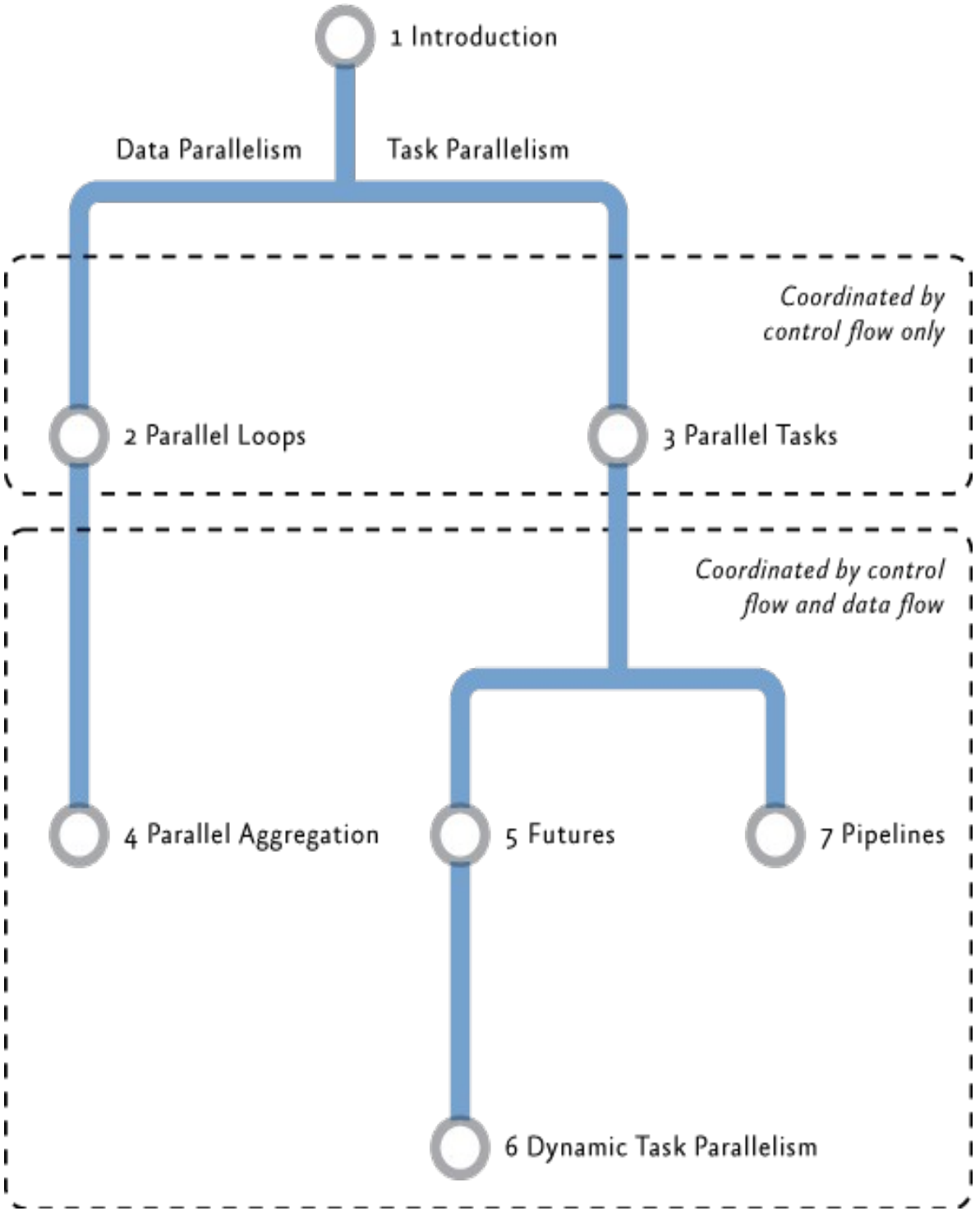
- **Outros padrões paralelos –** speculative selection
  - A cláusula condicional e ambos os fluxos (verdadeiro e falso) são executados em paralelo.
  - Quando a cláusula condicional for resolvida, o fluxo não desejado é cancelado. => pode aumentar a carga de trabalho.



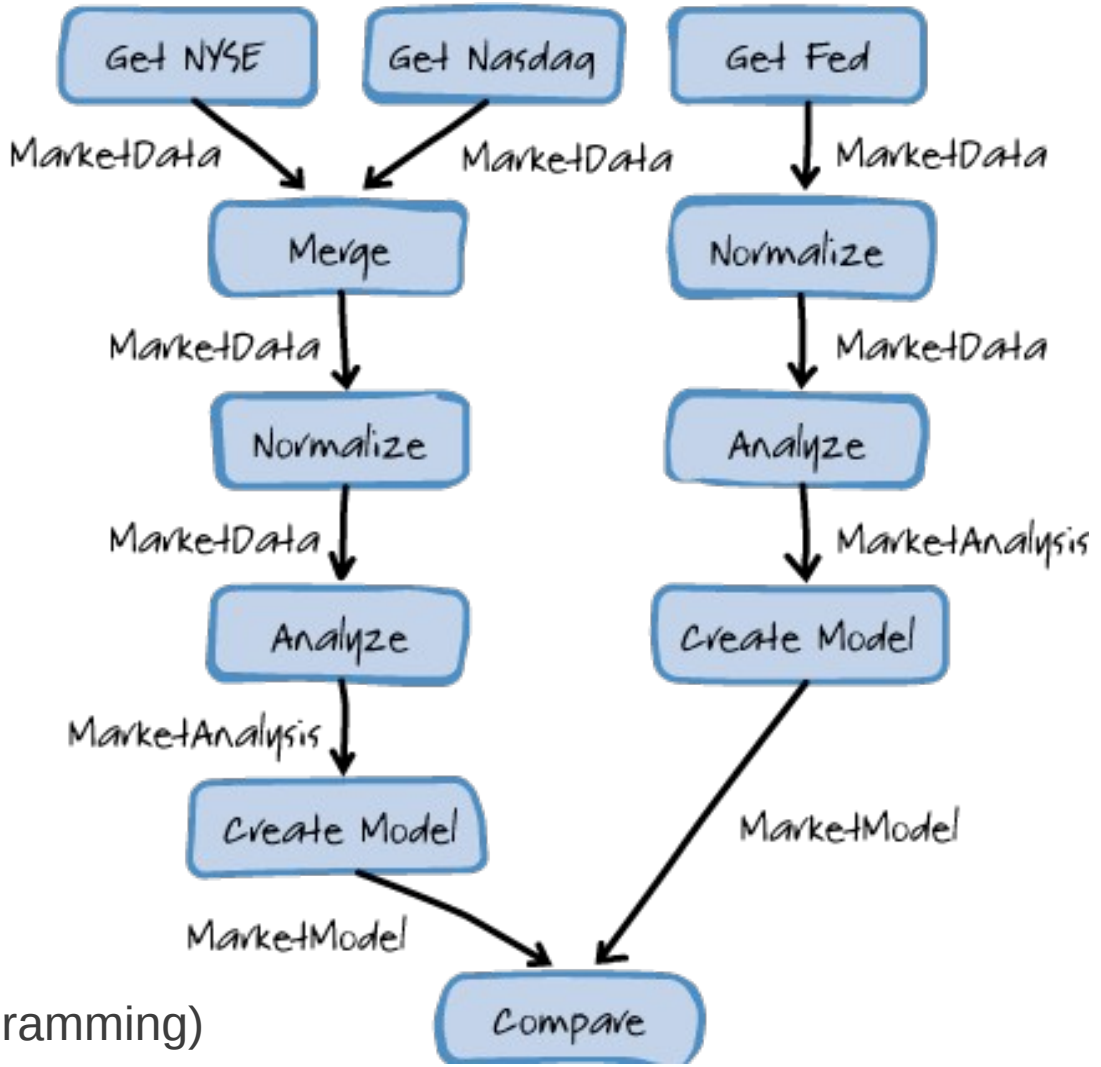
- **Outra possível classificação**



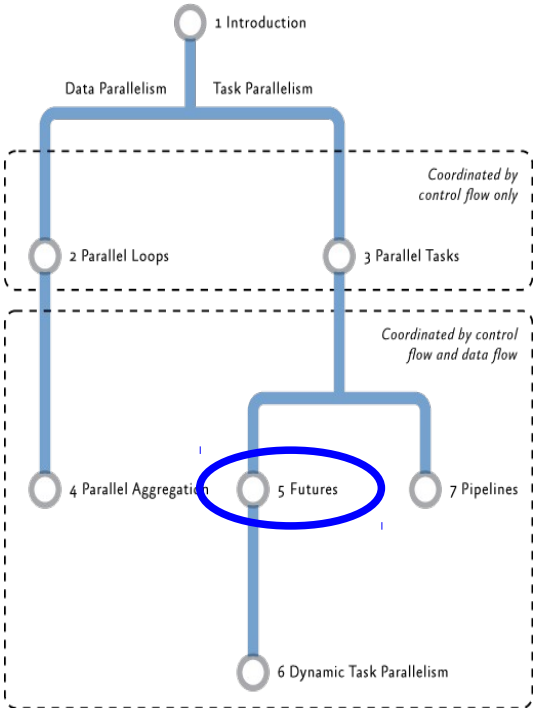

(RABAH, R. Design patterns for parallel programming.MIT 2007)

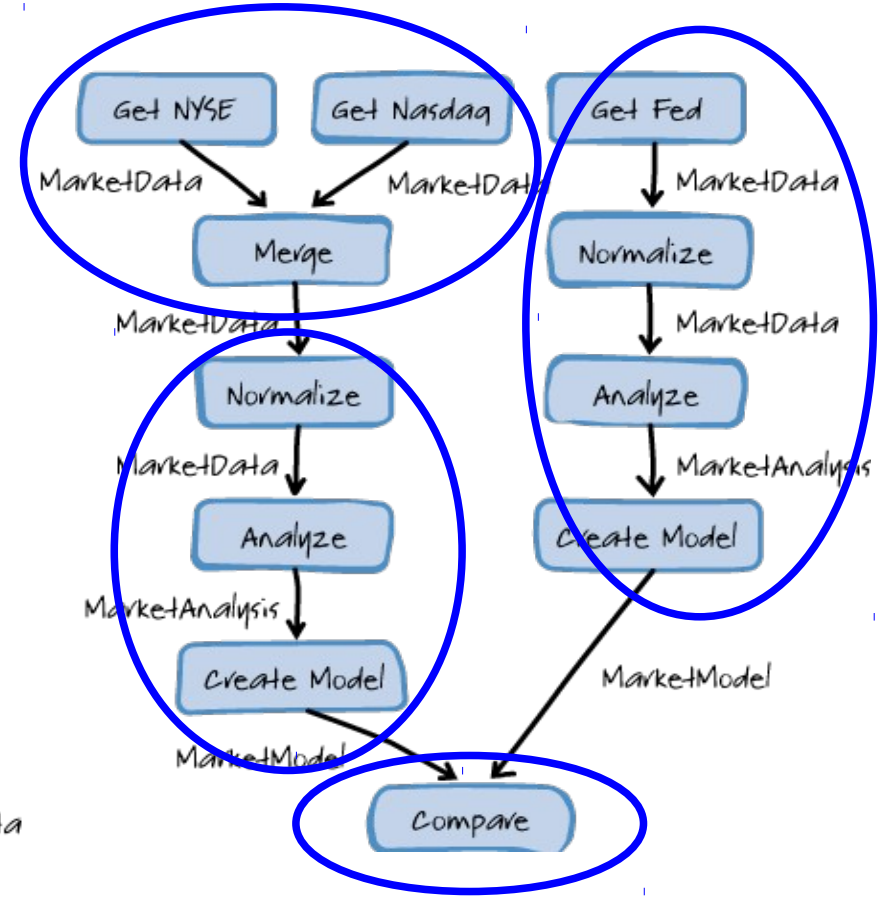

1 Introduction
Data Parallelism
Task Parallelism
Coordinated by control flow only
2 Parallel Loops
3 Parallel Tasks
Coordinated by control flow and data flow
4 Parallel Aggregation
5 Futures
7 Pipelines
6 Dynamic Task Parallelism

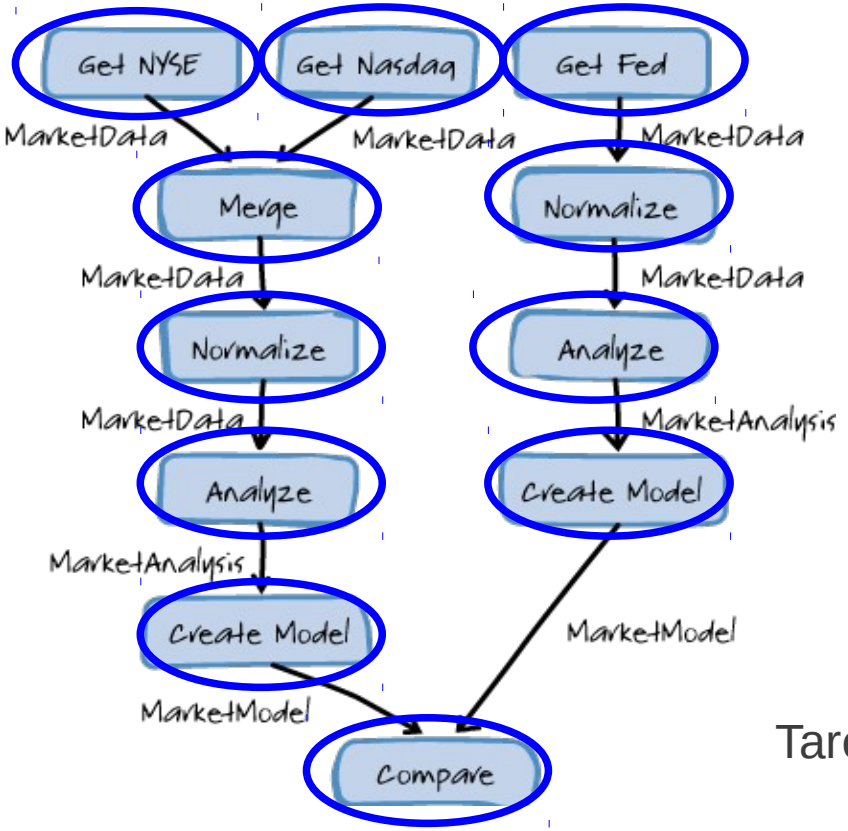
- **Estudo de caso: Adatum Dash**
  - Aplicação financeira para análise de risco (portfólio).
  - Dados recentes e históricos.
  - Compara modelos com as condições do mercado.




(MILLER, A. Patterns of parallel programming)

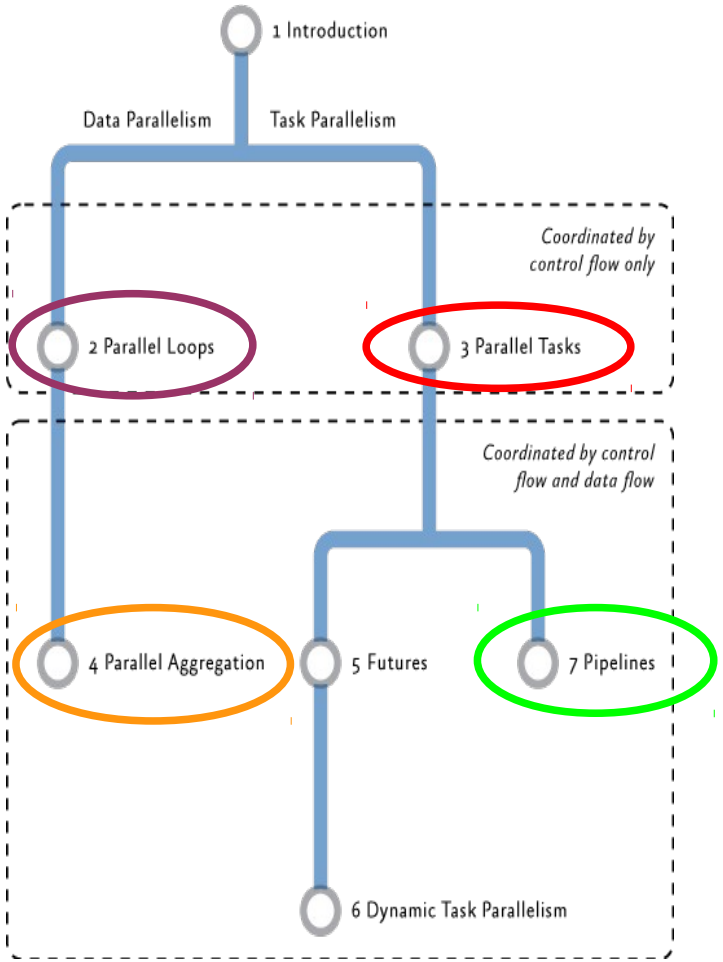


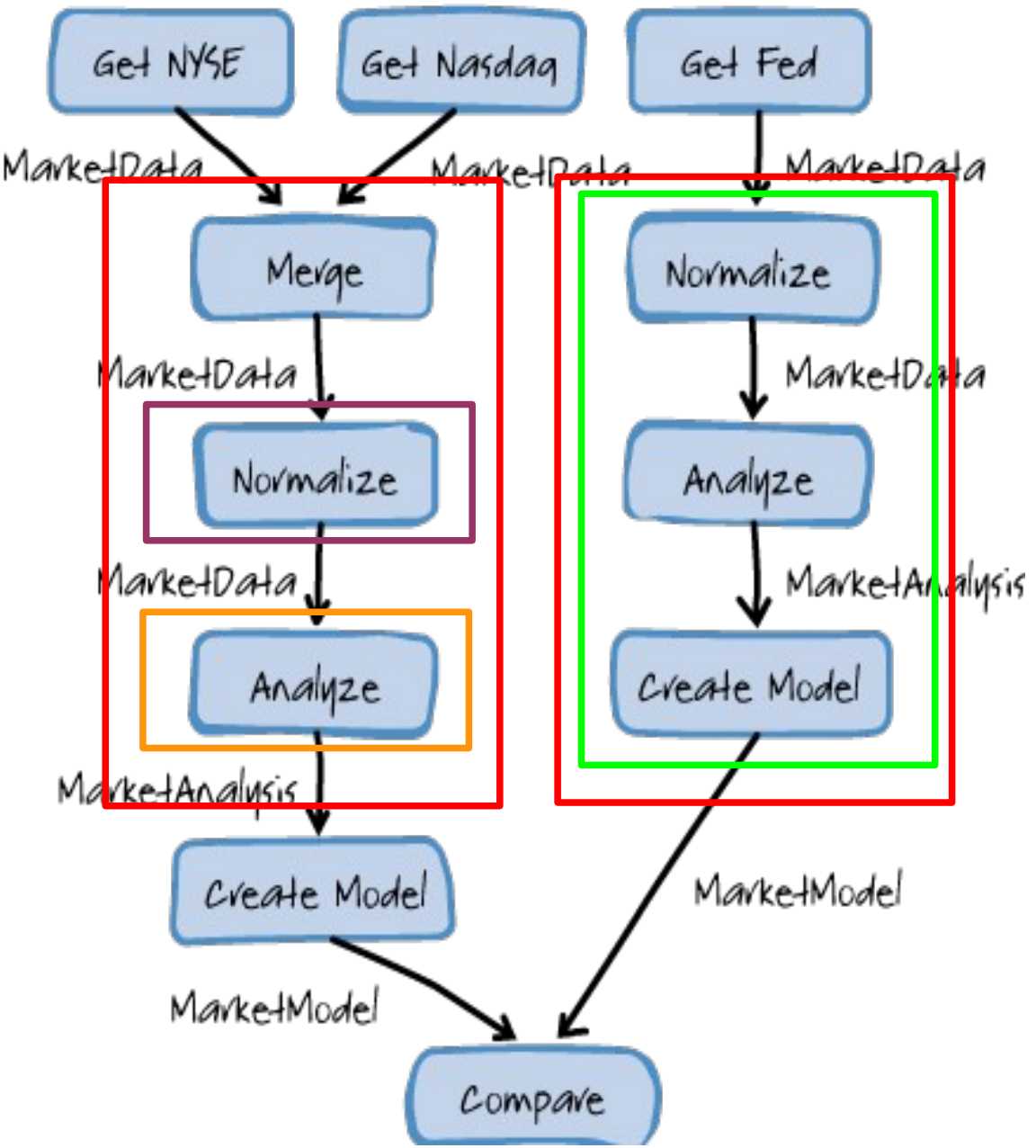




## Futures

Tarefas com dependências de dados.


1 Introduction
Data Parallelism
Task Parallelism
Coordinated by control flow only
2 Parallel Loops
3 Parallel Tasks
Coordinated by control flow and data flow
4 Parallel Aggregation
5 Futures
7 Pipelines
6 Dynamic Task Parallelism


Get NYSE
Get Nasdaq
Get Fed
MarketData
MarketData
MarketData
Merge
Normalize
MarketData
MarketData
Normalize
Analyze
MarketData
MarketAnalysis
Analyze
Create Model
MarketAnalysis
Create Model
MarketModel
MarketModel
Compare

- **Estudo de caso: modelagem de relevo**
  - BORATTO, M.; COELHO, L.; BARRETO, M. **Modelagem computacional da espacialização do relevo na região agrícola do Vale do Rio São Francisco.** (ICCSA 2012, ICCS 2012, Pesquisas Aplicadas em Modelagem Matemática. 1 ed.Ijuí : Unijuí, 2012, v.1, p. 10-20. ISBN 978-85-419-0037-9).

```
void matrizes(double *A, *x, *y, *z, int N, r){

 for(int l=0; l < N; ++l)
  for(int c=0; c < N; ++c)
      for(int i=0; i < n; ++i)
       A[l+c*N] += pow(x[i], (int)(l/(r+1)) +(int)(c/(r+1)))
                 *  pow(y[i],l%(r+1)+c%(r+1));

}
```

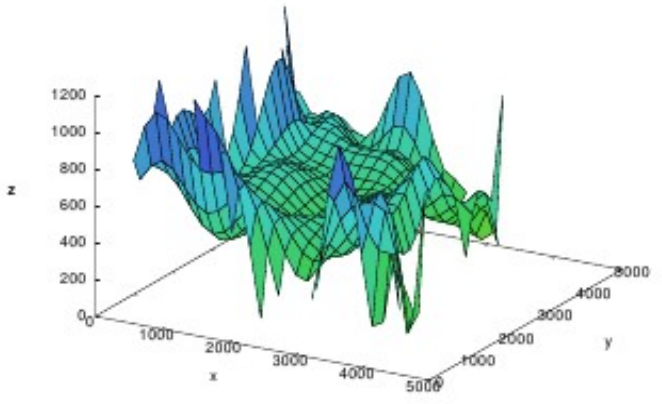
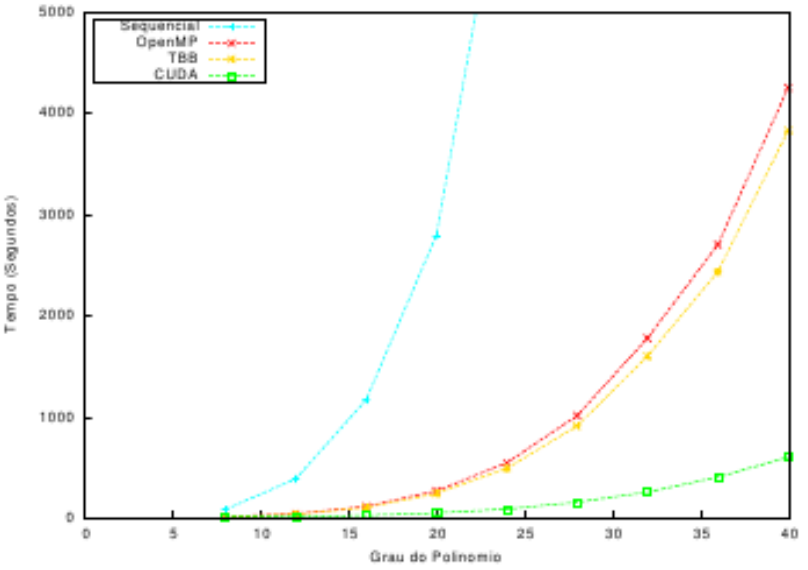



Tabela 2. Tempos de execução (em segundos) do algoritmo sequencial frente as ferramentas OpenMP, TBB e CUDA.

| Grau do Polinômio | Sequencial | OPENMP | TBB | CUDA |
|---|---|---|---|---|
| 8 | 84, 49 | 12, 32 | 11, 08 | 13, 61 |
| 12 | 386, 17 | 41, 85 | 37, 66 | 18, 04 |
| 16 | 1166, 88 | 114, 55 | 103, 09 | 25, 53 |
| 20 | 2842, 52 | 268, 32 | 241, 48 | 49, 29 |
| 24 | 5916, 06 | 544, 93 | 490, 47 | 88, 86 |
| 28 | 11064, 96 | 1011, 42 | 910, 27 | 156, 72 |
| 32 | 24397, 66 | 1777, 25 | 1599, 52 | 256, 62 |
| 36 | 30926, 82 | 2700, 00 | 2430, 00 | 404, 25 |
| 40 | 46812, 70 | 4252, 69 | 3827, 42 | 600, 00 |

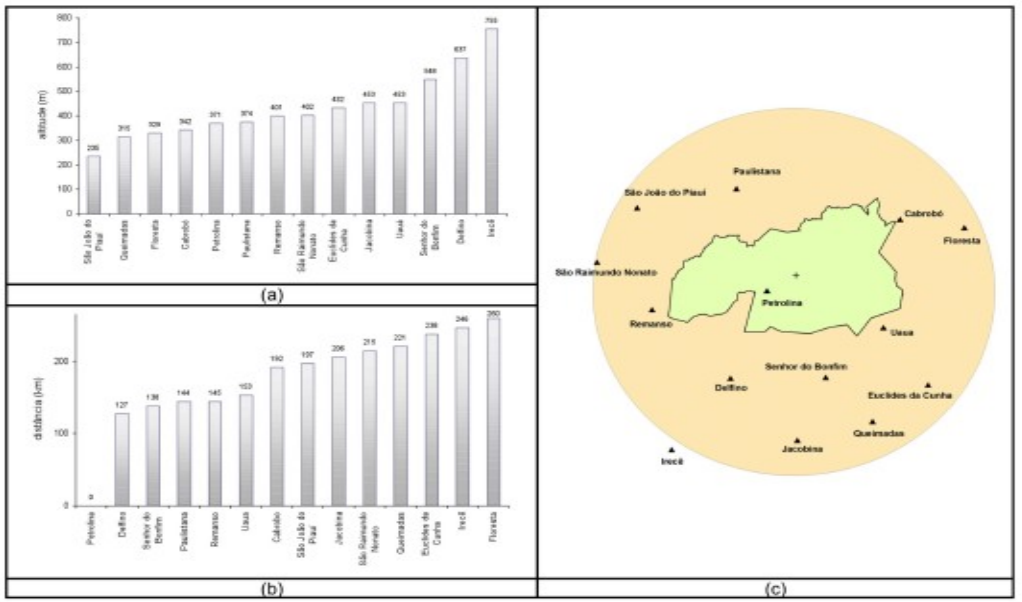


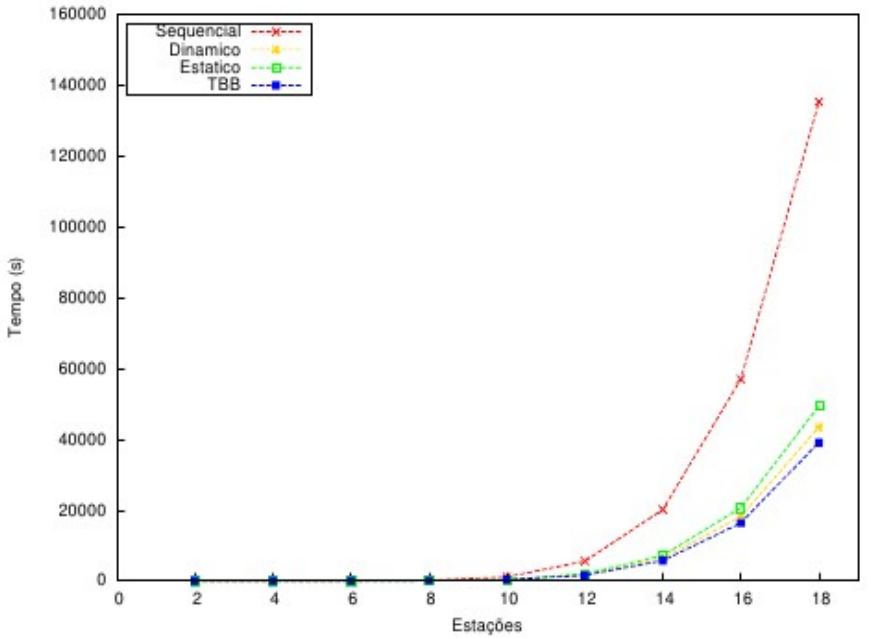
- **Estudo de caso: modelagem de clima**
  - BORATTO, M.; COELHO, L.; BARRETO, M. **Modelagem computacional da espacialização de variáveis meteorológicas na região agrícola do Vale do Rio São Francisco.** (Pesquisas Aplicadas em Modelagem Matemática. 1 ed.Ijuí : Unijuí, 2012, v.1, p. 30-40. ISBN 978-85-419-0037-9).

```
Flat profile:

  %   cumulative   self
 time   seconds   seconds  name
 30.46   2814.31  2814.31  clCombina::interpola()
 10.56   5510.12  2695.81  std::vector<clEstacao>
 12.23   7114.38  1604.26  sqlite3VdbeExec
  6.24   7932.07   817.69  std::vector<clEstacao>
  4.17   8478.42   546.34  sqlite3BtreeMovetoUnpacked
 ...
```

Tabela 6. Tempos de execução (em segundos) do algoritmo sequencial frente as ferramentas OpenMP com os escopos Estático e Dinâmico e TBB.

| Estações | Sequencial | OpenMP Estático | OpenMP Dinâmico | TBB |
|---|---|---|---|---|
| 2 | 0.73 | 0.53 | 0.53 | 0.47 |
| 4 | 4.93 | 2.67 | 2.53 | 2.27 |
| 6 | 31.10 | 14.40 | 13.30 | 11.97 |
| 8 | 188.97 | 78.03 | 69.83 | 62.83 |
| 10 | 1096.17 | 392.73 | 351.90 | 316.70 |
| 12 | 5538.90 | 1953.43 | 1737.87 | 1564.07 |
| 14 | 20135.70 | 7192.23 | 6357.47 | 5721.70 |
| 16 | 56987.70 | 20606.90 | 18154.90 | 16339.40 |
| 18 | 135352.70 | 49414.37 | 43436.27 | 39092.63 |

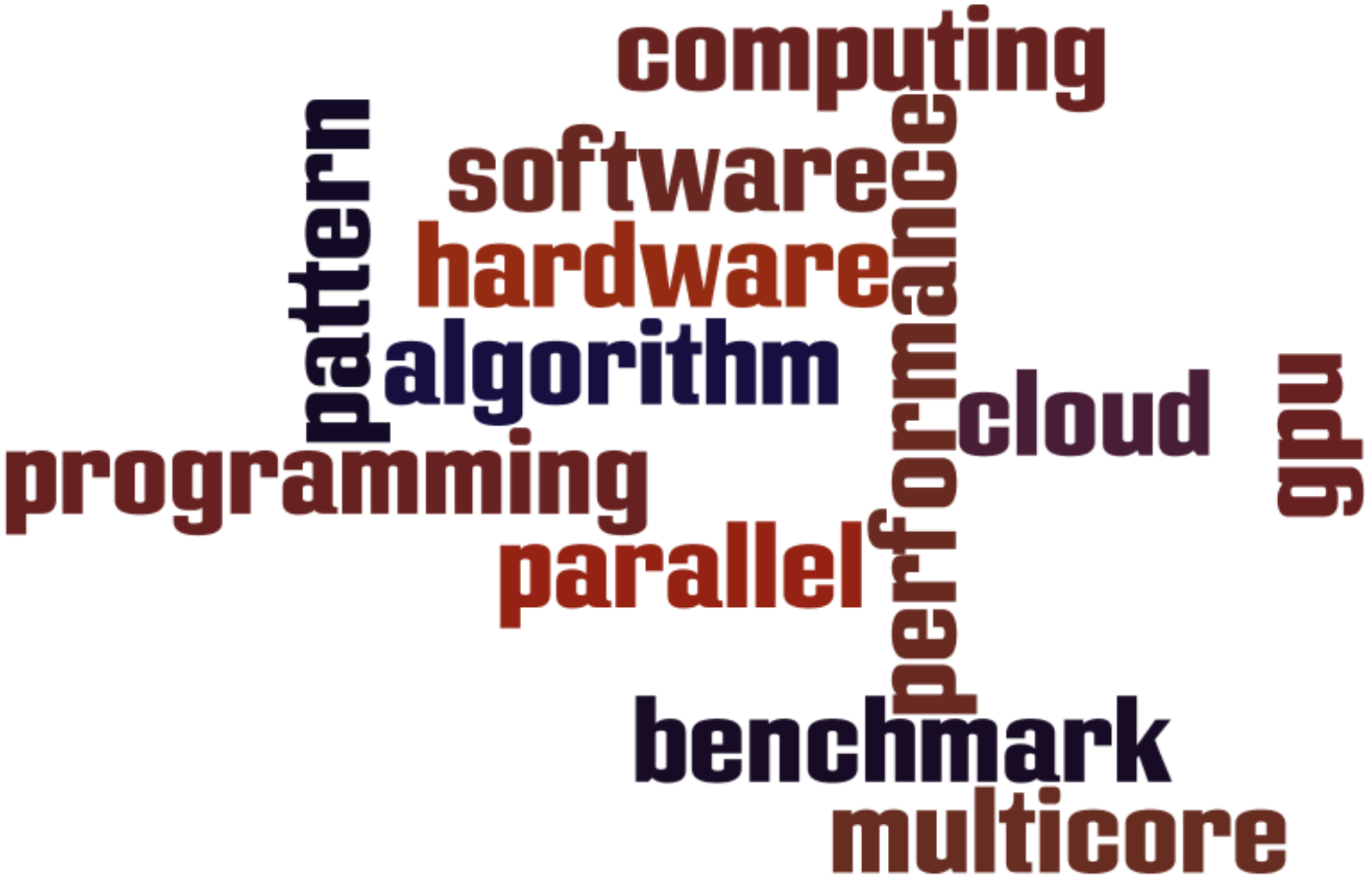


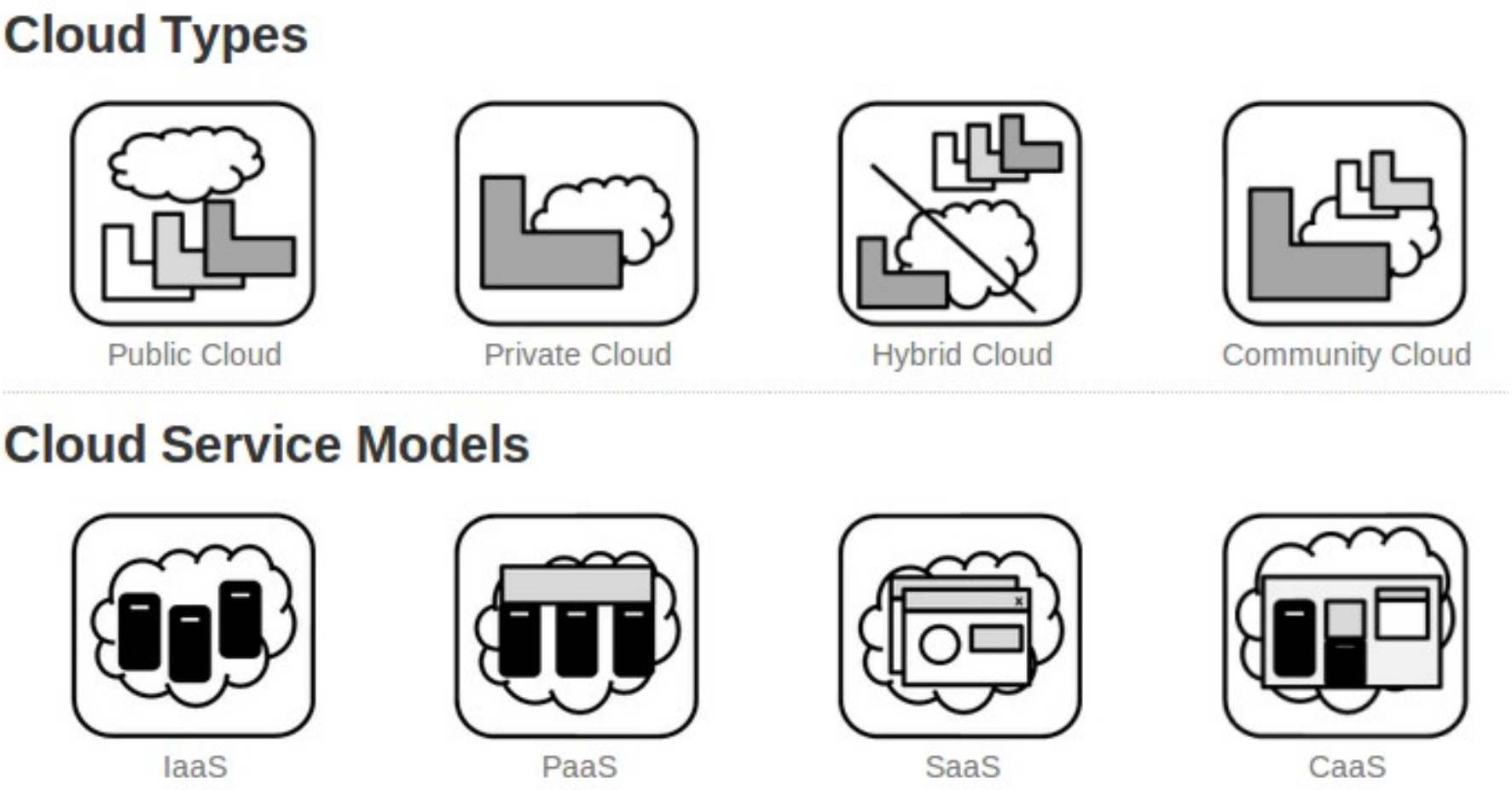
- **Padrões em GPGPU**
  - CUDPP (CUDA Data Primitives Parallel Library)
    - http://code.google.com/p/cudpp/
    - Algoritmos disponíveis
    - `cudppScan, cudppSegmentedScan, cudppReduce`
    - `cudppSort, cudppRand, cudppSparseMatrixVectorMultiply`
    - Algoritmos em desenvolvimento
      - Grafos, ordenação, hashing, autotuning
      -
  - Thrust (C++ template library for CUDA)
    - Containers
    - `thrust::host_vector<T>, thrust::device_vector<T>`
    - Algoritmos
    - `thrust::sort(), thrust::reduce(), thrust::inclusive_scan()`

```
__global__ void fillKernel(int *a, int n) {
  int tid = blockIdx.x*blockDim.x + threadIdx.x;
  if (tid < n) a[tid] = tid;
}
void fill(int* d_a, int n) {
  int nThreadsPerBlock= 512;
  int nBlocks= n/nThreadsPerBlock + ((n%nThreadsPerBlock)?1:0);
  fillKernel <<< nBlocks, nThreadsPerBlock >>> (d_a, n);
}

int main() {
  const int N=50000;
  // task 1: create the array
  thrust::device_vector<int> a(N);
  // task 2: fill the array using the runtime
  fill(thrust::raw_pointer_cast(&a[0]),N);
  // task 3: calculate the sum of the array
  int sumA= thrust::reduce(a.begin(),a.end(), 0);
  // task 4: calculate the sum of 0 .. N-1
  int sumCheck=0;
  for(int i=0; i < N; i++) sumCheck += i;
  // task 5: check the results agree
  if(sumA == sumCheck) cout << "Test Succeeded!" << endl;
  else { cerr << "Test FAILED!" << endl; return(1);}
  return(0);
}
```

# Computação em nuvem



- **Padrões nos diversos níveis de oferta**
  - http://cloudcomputingpatterns.org/
  - http://www.cloudpatterns.org/

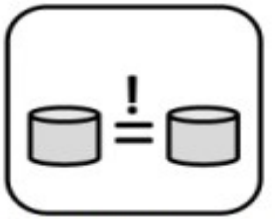


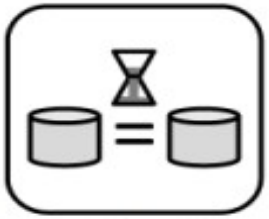
## Cloud Offerings

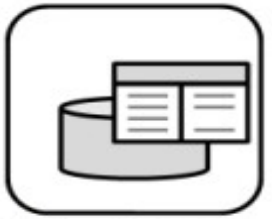

Low-available Compute Node

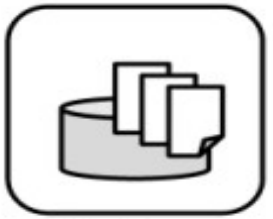

High-available Compute Node


Elastic Infrastructure

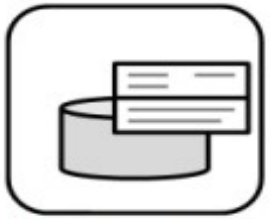

Strict Consistency

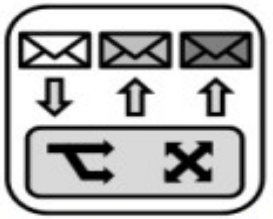

Eventual Consistency

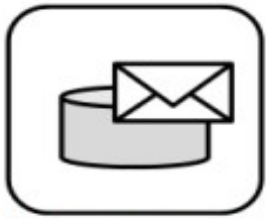

Relational Data Store


Blob Storage


Block Storage


NoSQL Storage

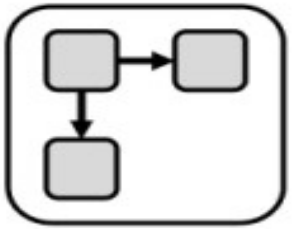

Message-oriented Middleware

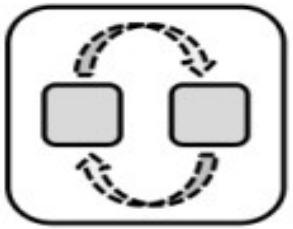

Reliable Messaging

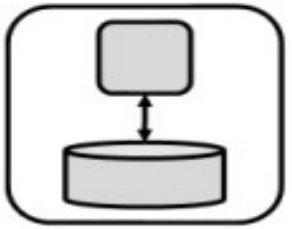

At-least-once


Exactly-once

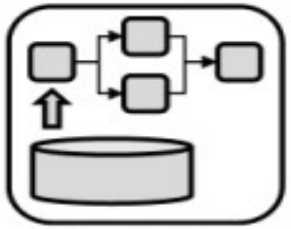
# Cloud Application Architectures

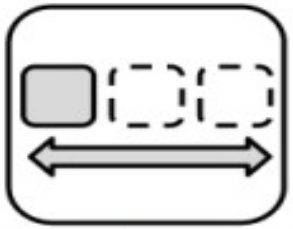

Composite Application

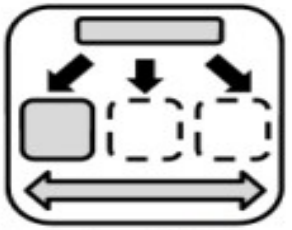

Loose Coupling

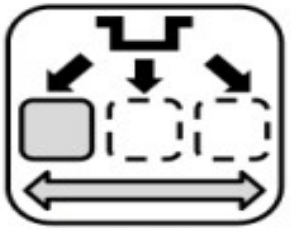

Stateless Component

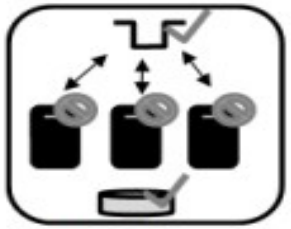

Idempotent Component

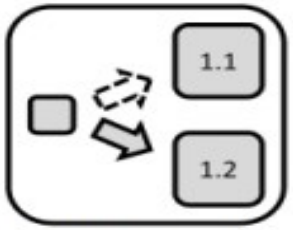

Map-Reduce


Elastic Component

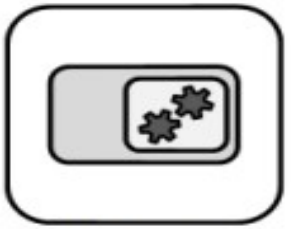

Elastic Load Balancer

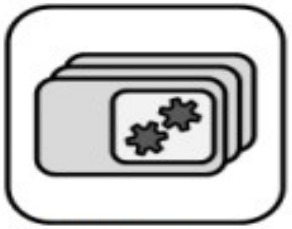

Elastic Queue

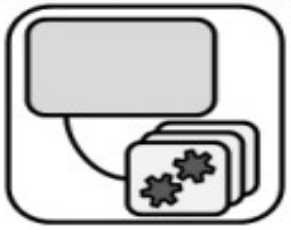

Watchdog



Update Transition

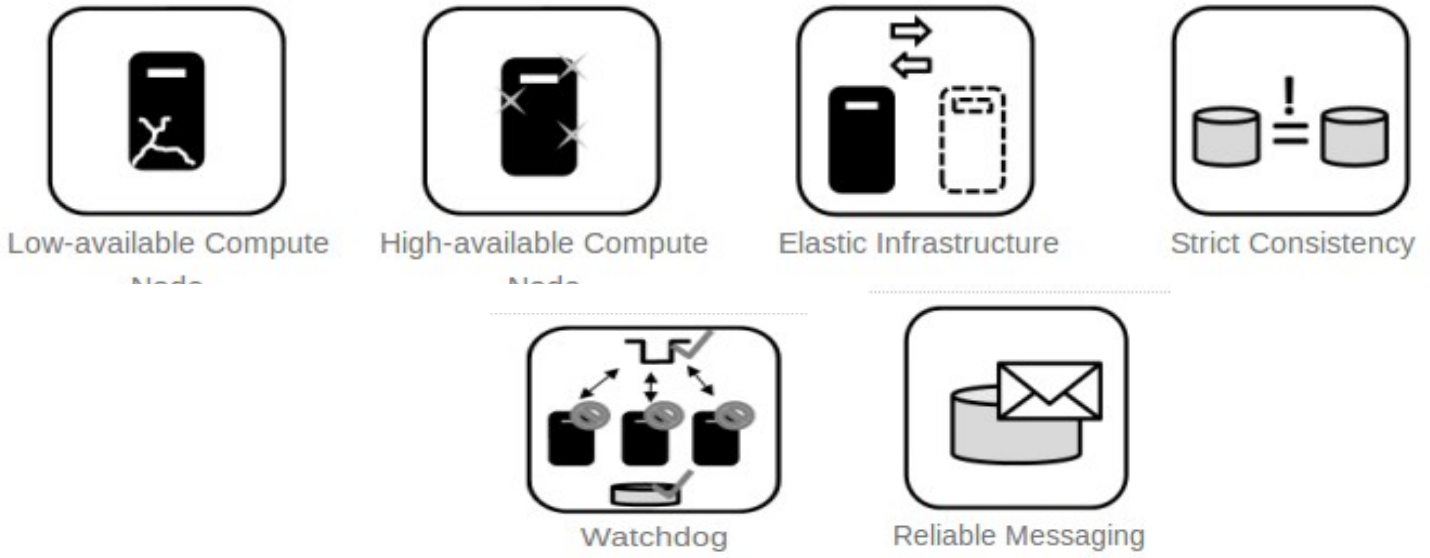

Single Instance Component


Multiple Instance Component

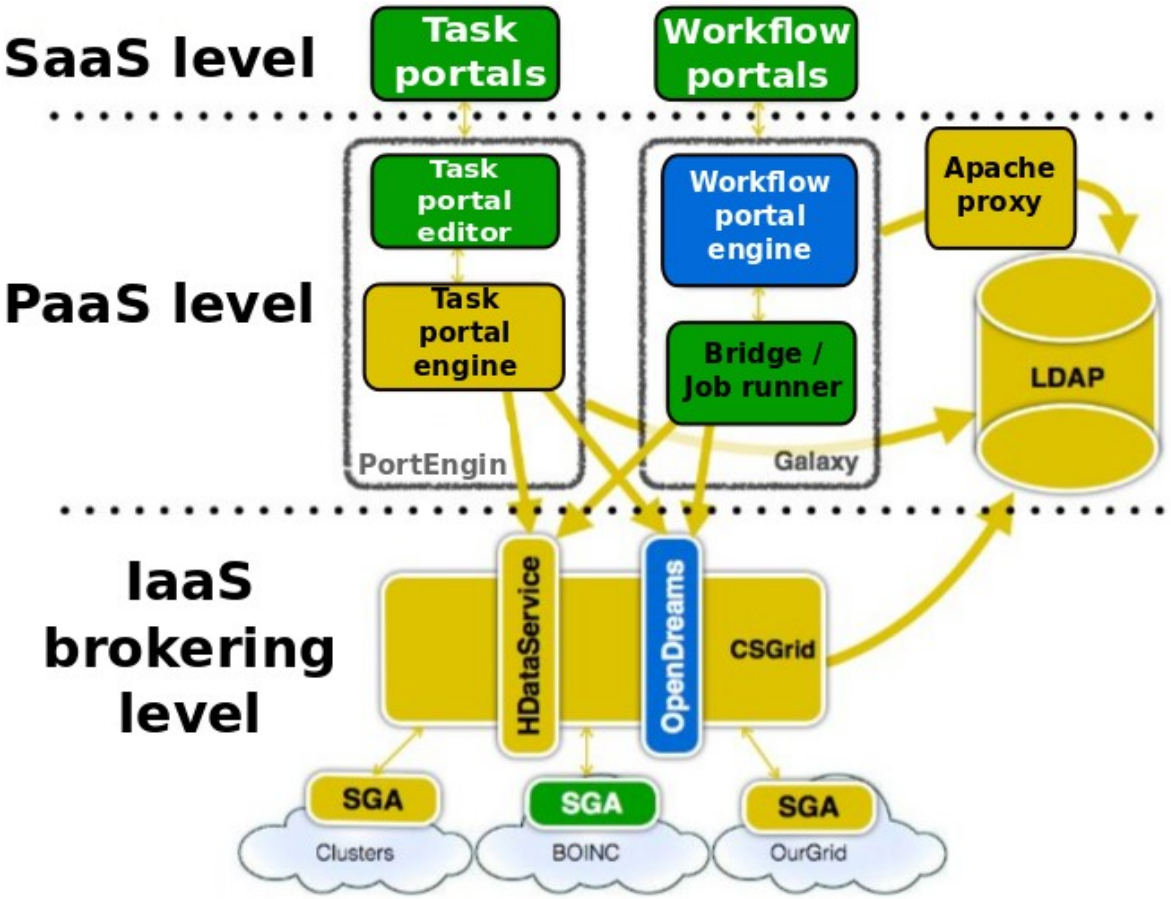

Single Configurable Instance Component

- **Estudo de caso: JiT Clouds**
  - JiT Clouds: uma proposta para ampliar a elasticidade de provedores de computação em nuvem baseada em federação de recursos computacionais amortizados.
  - Coordenação: Francisco Brasileiro (UFCG), Philippe Navaux (UFRGS)

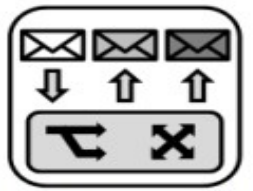

**Módulo de dependabilidade**
- Eucalyptus
- Remus (failover)
- DRDB (discos distribuídos)
- Corosync (comunicação em grupo)
- Pacemaker (gestão de clusters)

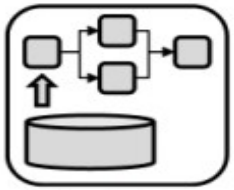


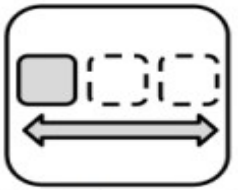
- **Estudo de caso: Minha cloud científica (GT-MCC)**
  - RNP – Grupos de Trabalho 2012 – 2013 (fase 2)
  - Coordenação: Antônio Tadeu (LNCC), Francisco Brasileiro (UFCG)

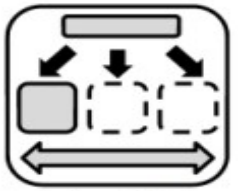


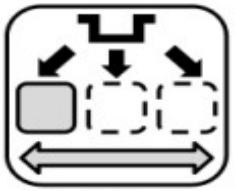




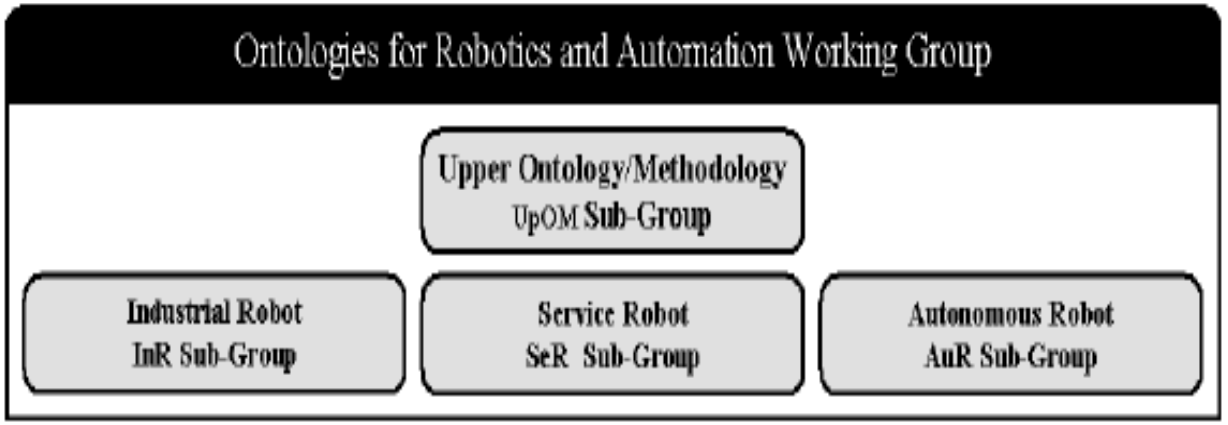



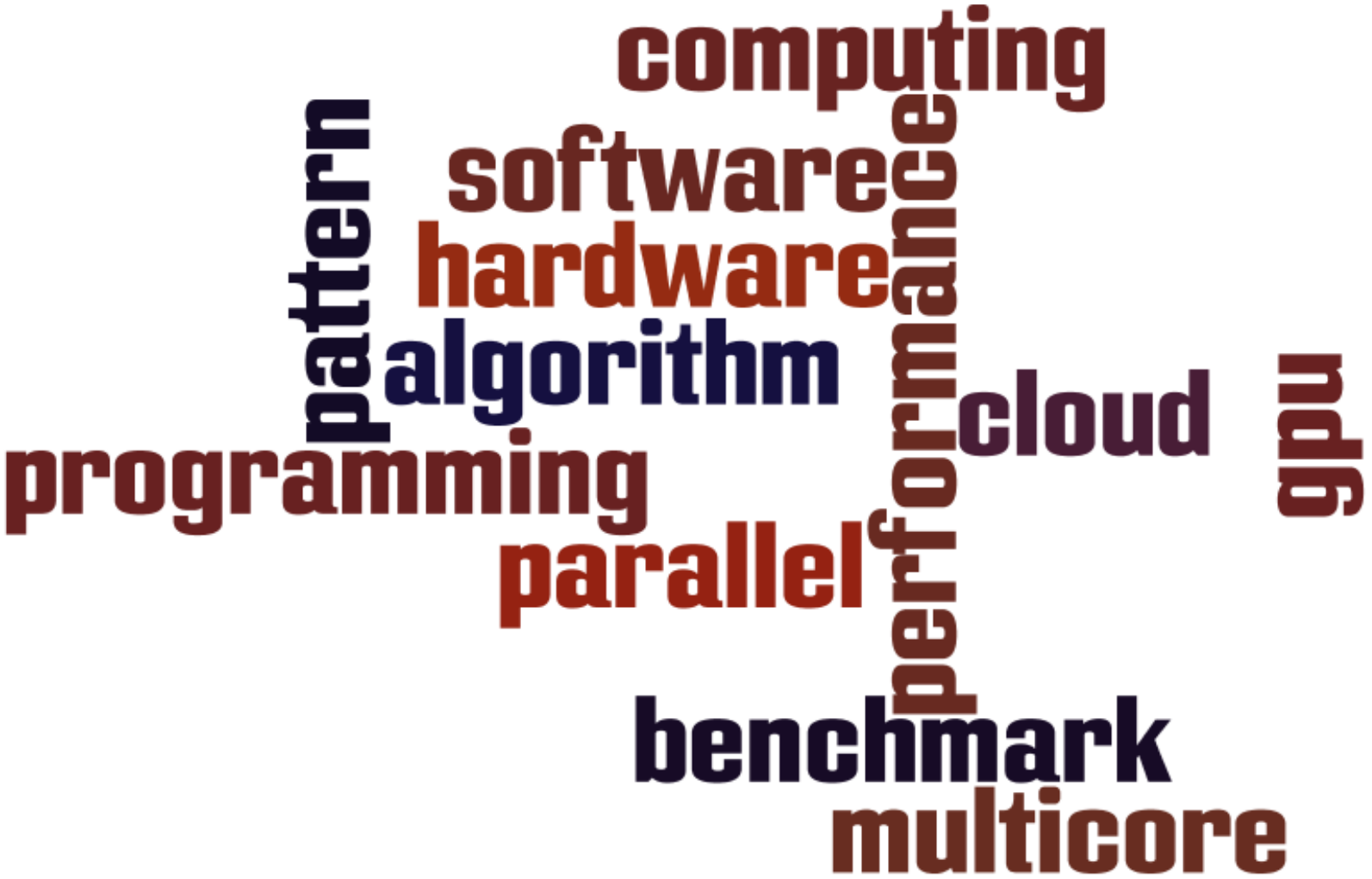



- **Estudo de caso: cloud robotics (IEEE ORA WG)**
  - Craig Schneloff (NIST), Edson Prestes (UFRGS).
  - Objetivos:
    - Ontologias e metodologias de avaliação para Robótica e Automação.
    - Serviços baseados em nuvem para:
      - Compartilhamento de conhecimento (algoritmos, configurações etc).
      - Processamento em tepo real.
      - Simulação de robôs.



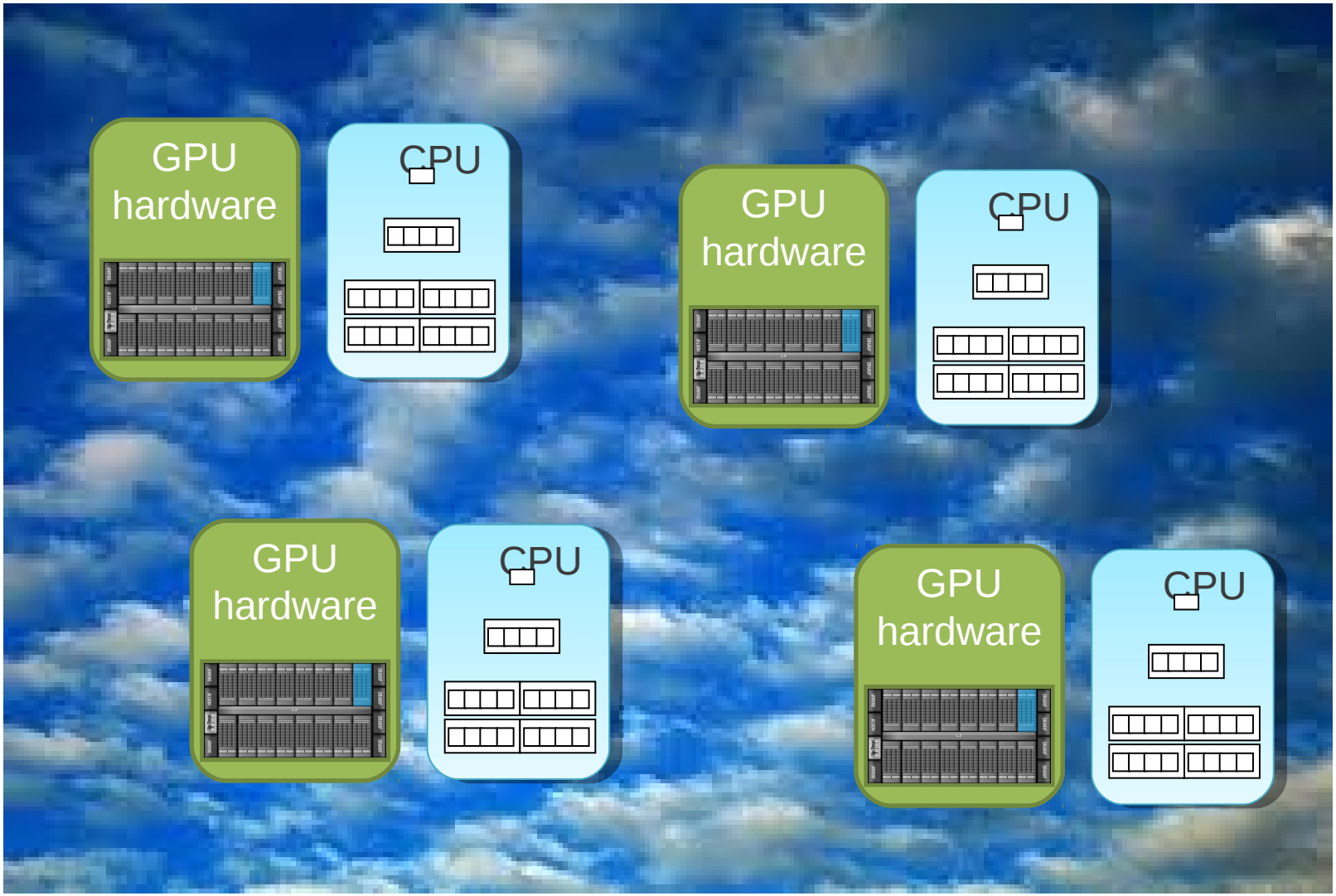

PRESTES, E.; SCHLENOFF, C. Towards an upper ontology and methodology for robotics and automation. (UBICOMP 2012).
HAIDEGGER, T.; BARRETO, M.; et al. Nesting the context for pervasive robotics. (UBICOMP 2012).
=> Robotics and Automation Systems.
http://www.journals.elsevier.com/robotics-and-autonomous-systems/

# Síntese e discussão



# Padrões de programação (~~paralela~~) aplicados às arquiteturas (~~híbr~~idas)

## Multicore + GPGPU + cloud



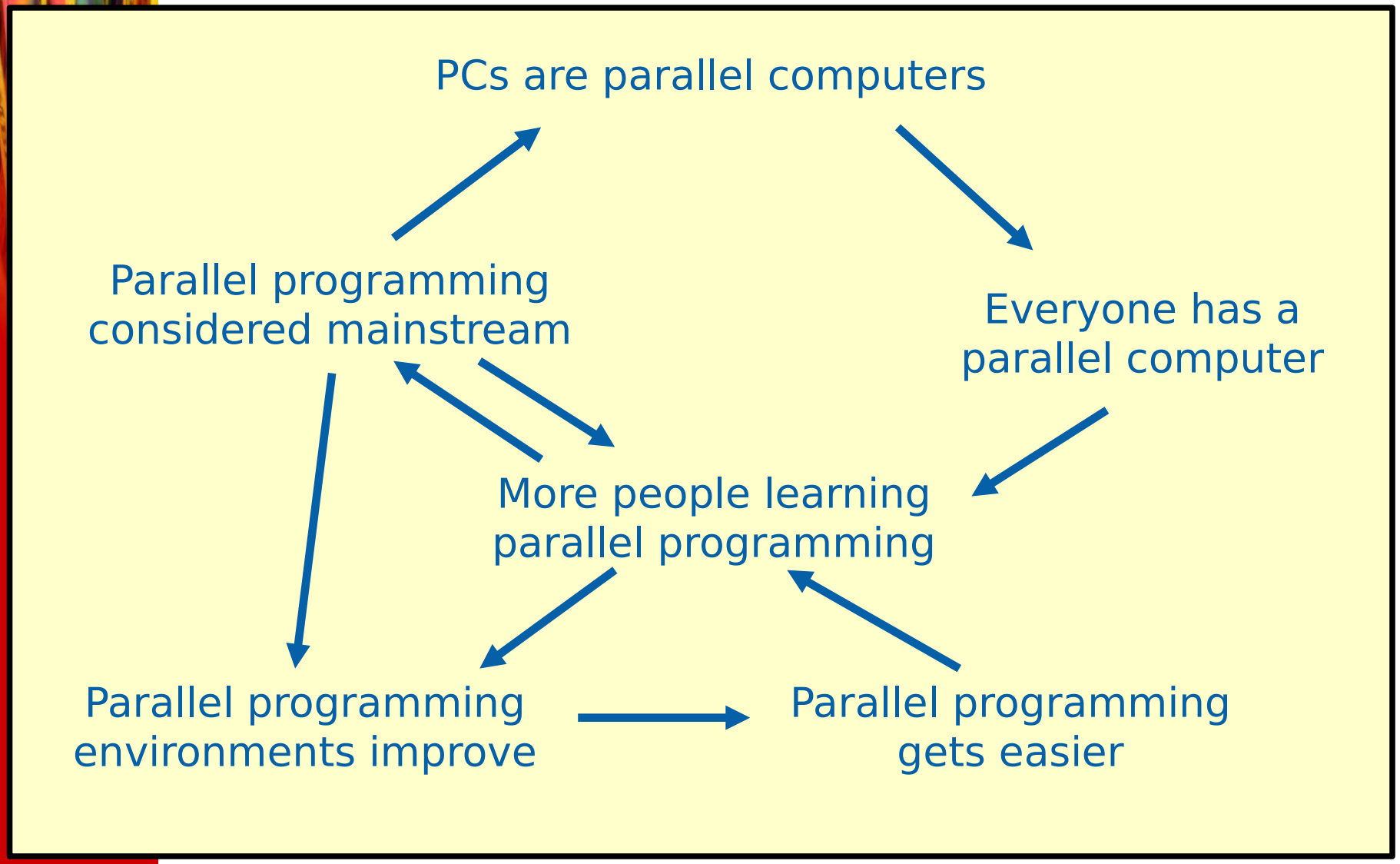
# Padrões de programação (~~paralela~~) aplicados às arquiteturas (~~híbr~~idas)

## Think parallel

Baiana de acarajé.

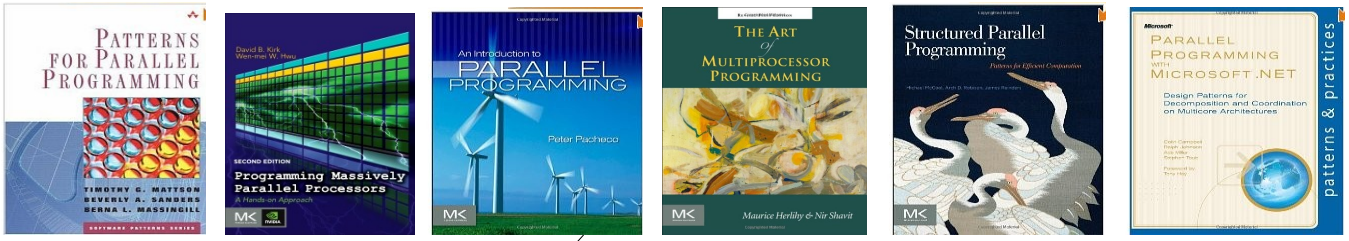
# Padrões de programação (~~paralela~~) aplicados às arquiteturas (~~híbr~~idas)

## Think parallel

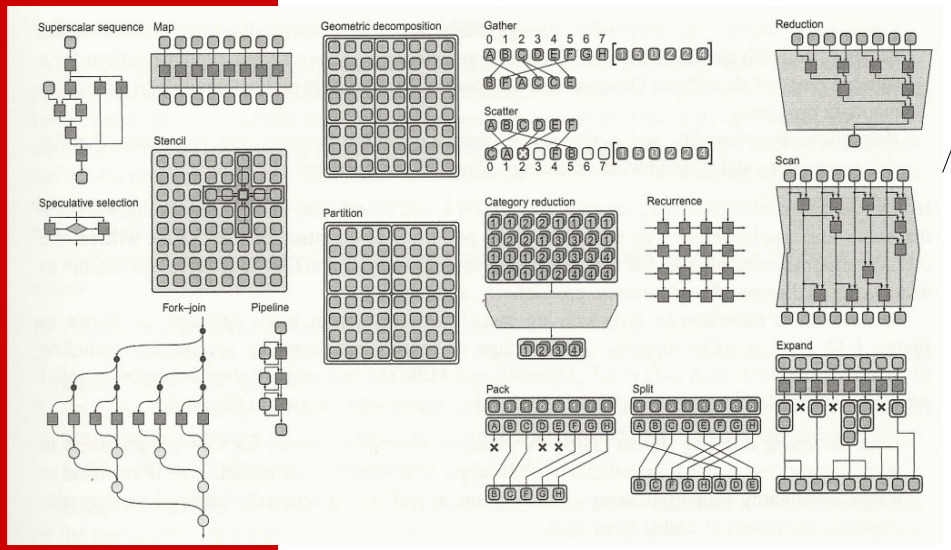


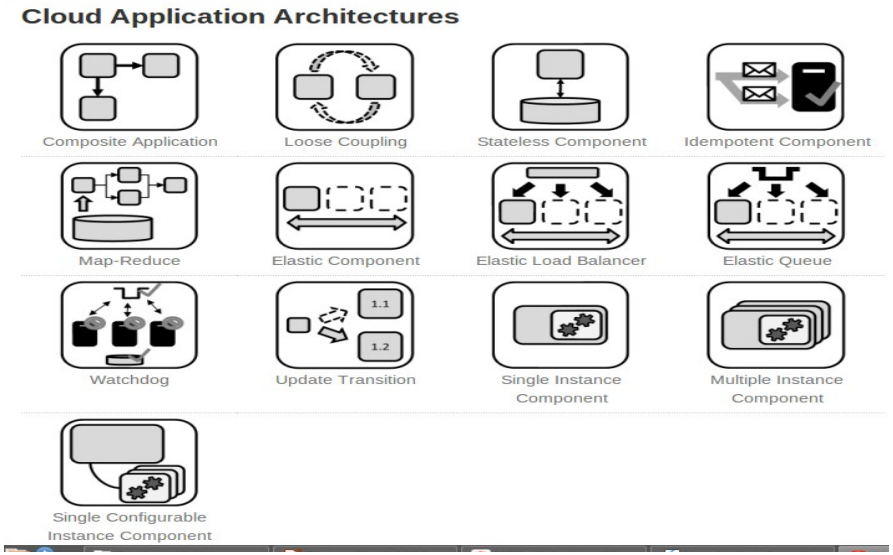
# Padrões de programação (~~paralela~~) aplicados às arquiteturas (~~híbr~~idas)

## Padrões


+



# Padrões de programação (~~paralela~~) aplicados às arquiteturas (~~híbr~~idas)

Intel - Recursos acadêmicos

http://software.intel.com/pt-br/academic

NVIDIA Education & Training

https://developer.nvidia.com/cuda-education-training

# Padrões de programação paralela aplicados às arquiteturas híbridas

Conceitos, ferramentas e estudo de casos

**Marcos Barreto**

LaSiD / DCC / UFBA
marcoseb@dcc.ufba.br

**OBRIGADO!**