# Programação Concorrente e Paralela

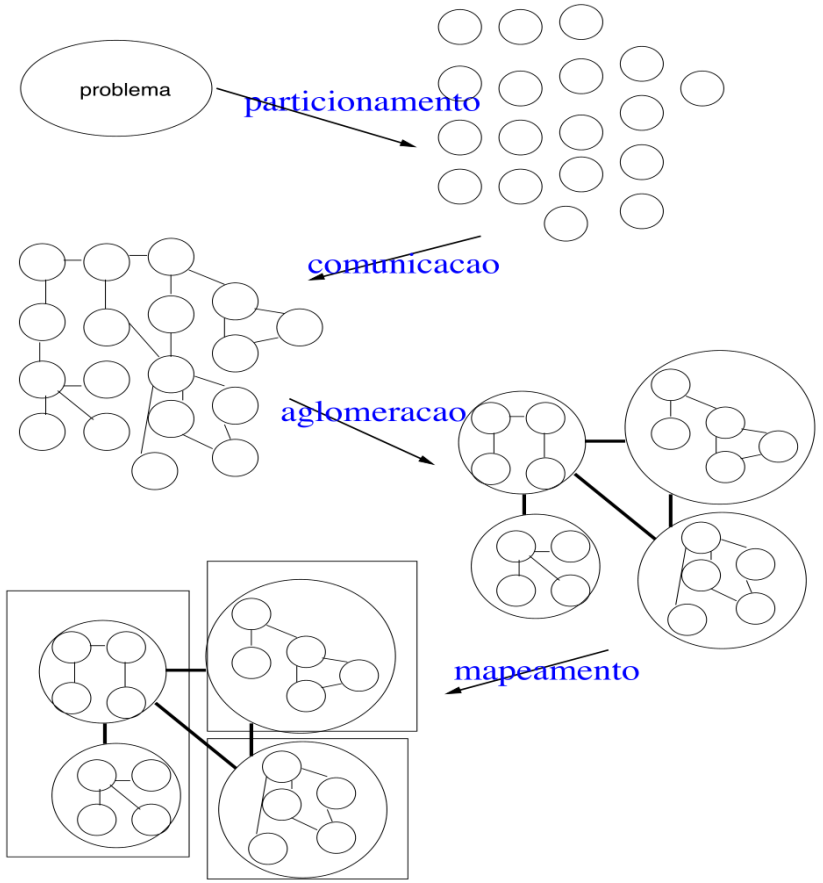
## projeto de programas paralelos

Noemi Rodriguez

2016

# PCAM

M. Quinn




- desenhada para memória distribuída mas muitas idéias em comum

# Particionamento

- idéia é expor oportunidades de paralelismo
  - foco em determinar uma grande quantidade de pequenas tarefas
- decomposição funcional e decomposição por domínio
- taxonomia de Flynn (transposta para programas)
  - spsd
  - spmd
  - mpsd
  - mpmd

# Flynn

SISD — SINGLE INSTRUCTION SINGLE DATA — maq sequencial

SIMD — MULTIPLE DATA — arqs vetoriais

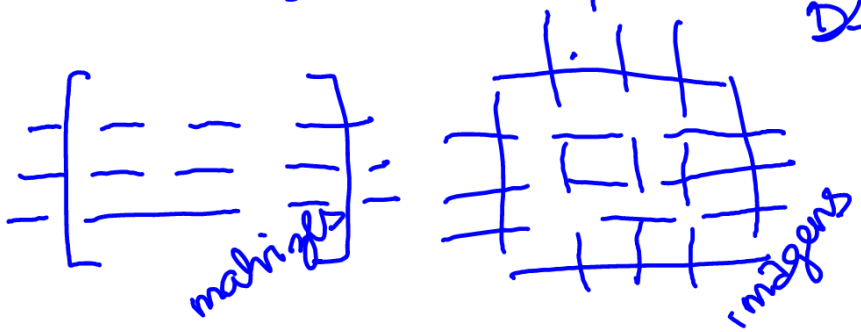
MISD — MULTIPLE INSTRUCTION SINGLE DATA

MIMD — MULTIPLE INST, MULTIPLE DATA

para paralelização de programas

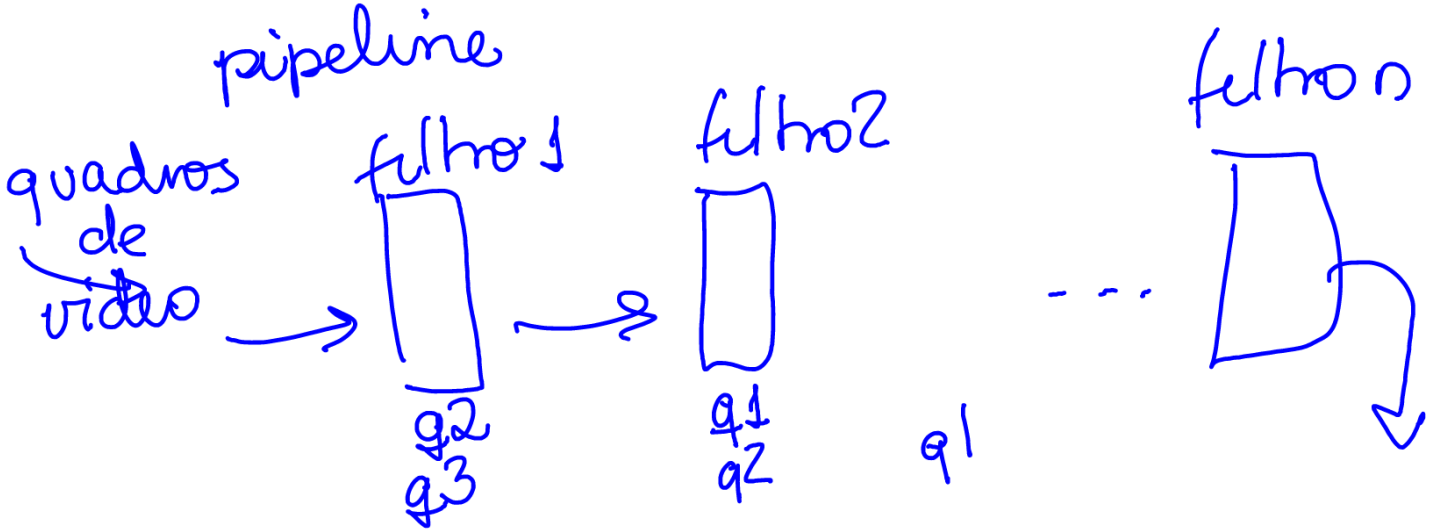
SPMD — SINGLE PROGRAM, M. DATA

DECOMPOSIÇÃO DE DOMÍNIO DADOS



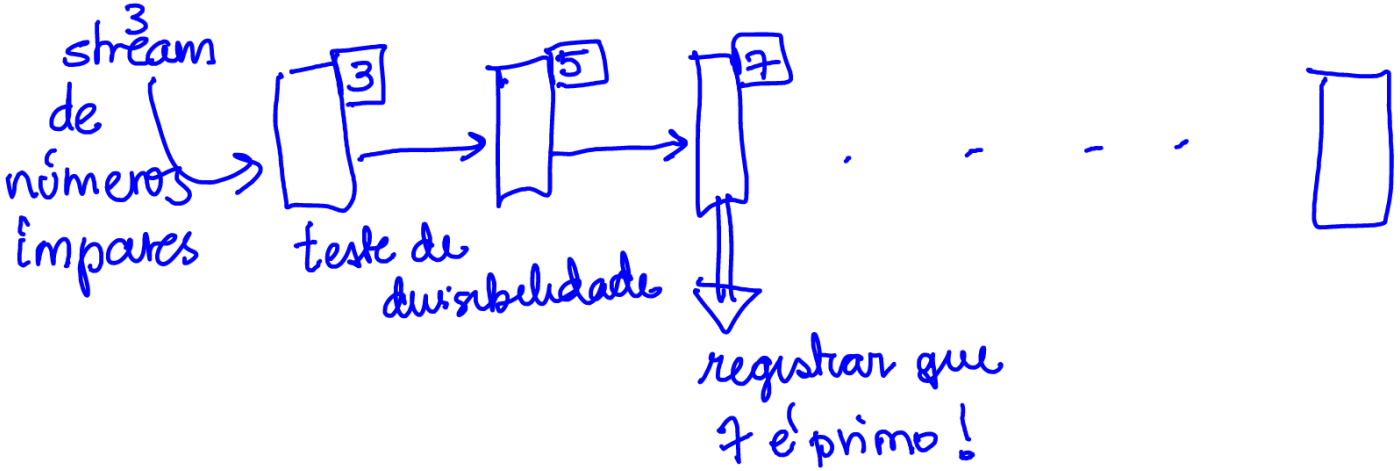
# Multiple Program Single Data

pipeline




---

pipeline com crivo de Eratóstenes



MPMD

multiple program, multiple data

DECOMPOSIÇÃO FUNCIONAL

ex: servidores com

threads atendendo requisições

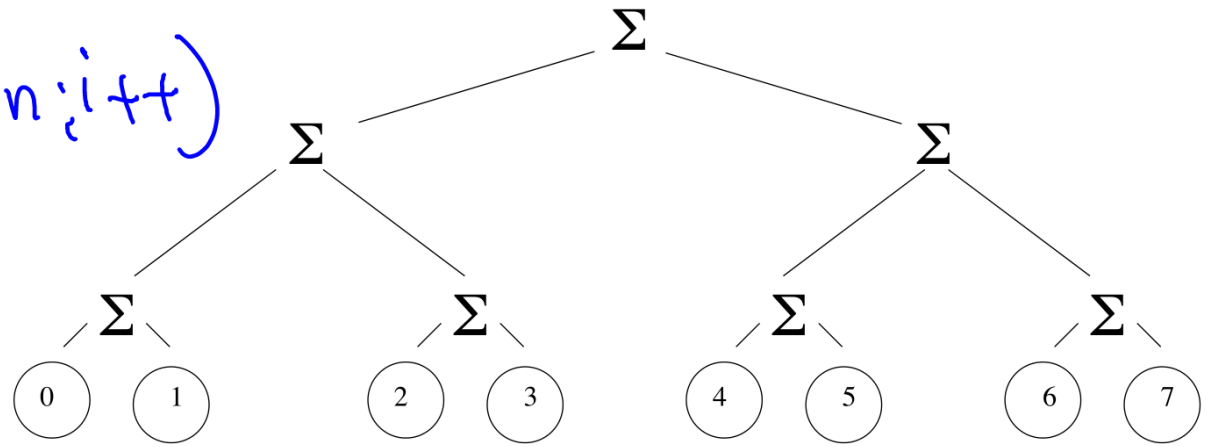
threads realizando compactação

threads monitorando estado do servidor

# Oportunidades de Paralelização

- solução paralela parte de versão sequencial?
- versão sequencial pode ou não ser a que oferece maiores oportunidades
  - exemplo soma de N números

```
s=0
for (i=0; i<n; i++)
  s += i;
```




  - exemplo paralelização de computação com dependências

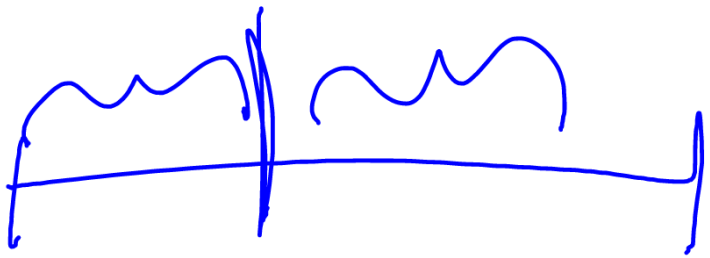
# Oportunidades de Paralelização

## ordenação

- que algoritmos se prestam a paralelização?

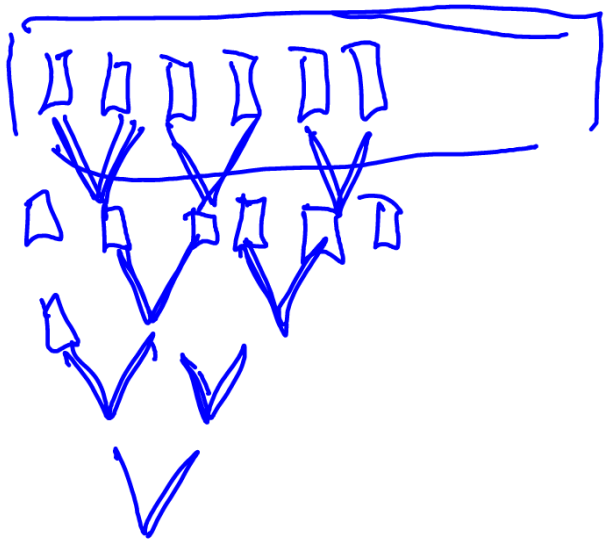
# Ordenação: bubble sort

```
for (list_length = n; list_length >= 2; list_length--)
  for (i=0; i<list_length; i++)
    if (a[i] > a[i+1]) {
      tmp = a[i];
      a[i] = a[i+1];
      a[i+1] = tmp;
    }
```



pouca oportunidade de paralelização!

# Ordenação: *odd-even transposition*



```
for (phase = 0; phase < n; phase++)
  if (phase%2)  /* impar */
    for (i = 1; i < n-1; i += 2)
      if (a[i] > a[i+1]) Swap(&a[i], &a[i+1]);
  else          /* par */
    for (i = 1; i < n; i += 2)
      if (a[i-1] > a[i]) Swap (&a[i-1], &a[i]);
```

Parecido porém com mais oportunidades

# *odd-even transposition* – versão paralela em openmp

```
for (phase = 0; phase < n; phase++)
  if (phase%2)  /* impar */
    for (i = 1; i < n-1; i += 2)
      if (a[i] > a[i+1]) Swap(&a[i], &a[i+1]);
  else          /* par */
    for (i = 1; i < n; i += 2)
      if (a[i-1] > a[i]) Swap (&a[i-1], &a[i]);
```

não podemos simplesmente colocar o loop externo em paralelo!

# *odd-even transposition* – versão paralela em openmp

```
   for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
#        pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
         for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
               tmp = a[i-1]; a[i-1] = a[i]; a[i] = tmp;
            }
         }
      else
#        pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
         for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
               tmp = a[i+1]; a[i+1] = a[i]; a[i] = tmp;
            }
         }
```

- muita criação e destruição de threads !

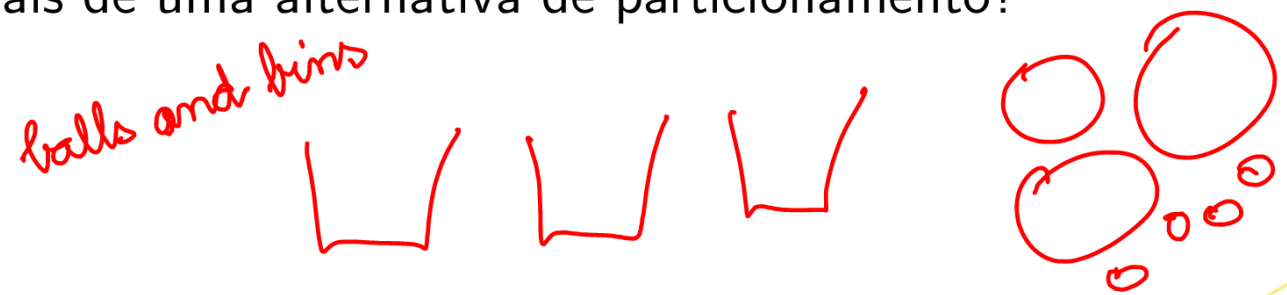
# *odd-even transposition* – versão 2 em openmp

```
#  pragma omp parallel num_threads(thread_count) \
      default(none) shared(a, n) private(i, tmp, phase)
   for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
#        pragma omp for
         for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
               tmp = a[i-1]; a[i-1] = a[i]; a[i] = tmp;
            }
         }
      else
#        pragma omp for
         for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
               tmp = a[i+1]; a[i+1] = a[i]; a[i] = tmp;
            }
         }
   }
```

- exemplo retirado de: Peter Pacheco. *An Introduction to Parallel Programming.*

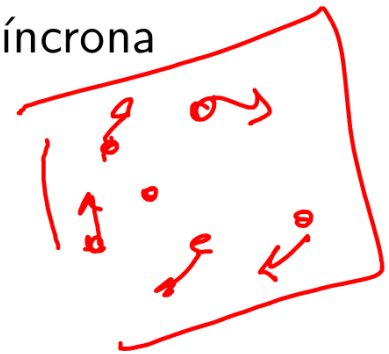
# Verificação do Particionamento

- partição gera número de tarefas com ordem de magnitude superior à do número de processadores disponíveis?
  - flexibilidade para estágios posteriores
- tarefas são de tamanho comparável?
- número de tarefas cresce junto com tamanho da entrada do problema?
- há mais de uma alternativa de particionamento?




# Comunicação

- como as tarefas definidas na fase 1 vão se comunicar?
  - definição de canais a serem usados
  - definição de mensagens
  - *para memória compartilhada:* custos de sincronização
- alguns critérios de classificação são úteis:
  - local x global, estruturada x não estruturada, estática x dinâmica
  - assíncrona x síncrona



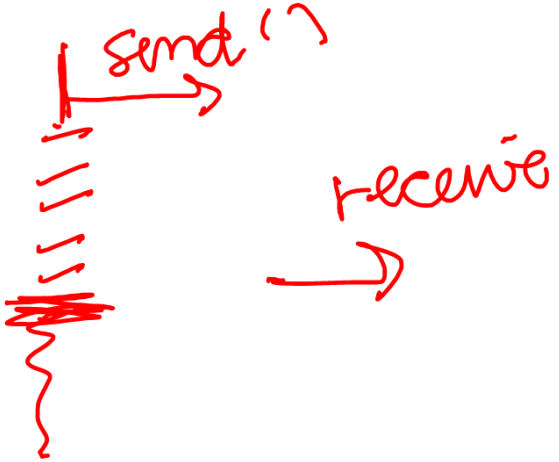
# Comunicação – padrões

- comunicações *globais* podem criar gargalos de comunicação
  - exemplos: bolsa de tarefas única, ...
- comunicações *estruturadas* podem facilitar a tarefa de aglomerar e mapear
- sincronismo: simplicidade x menos oportunidades de concorrência

troca de msgs

síncrona

envio síncrono



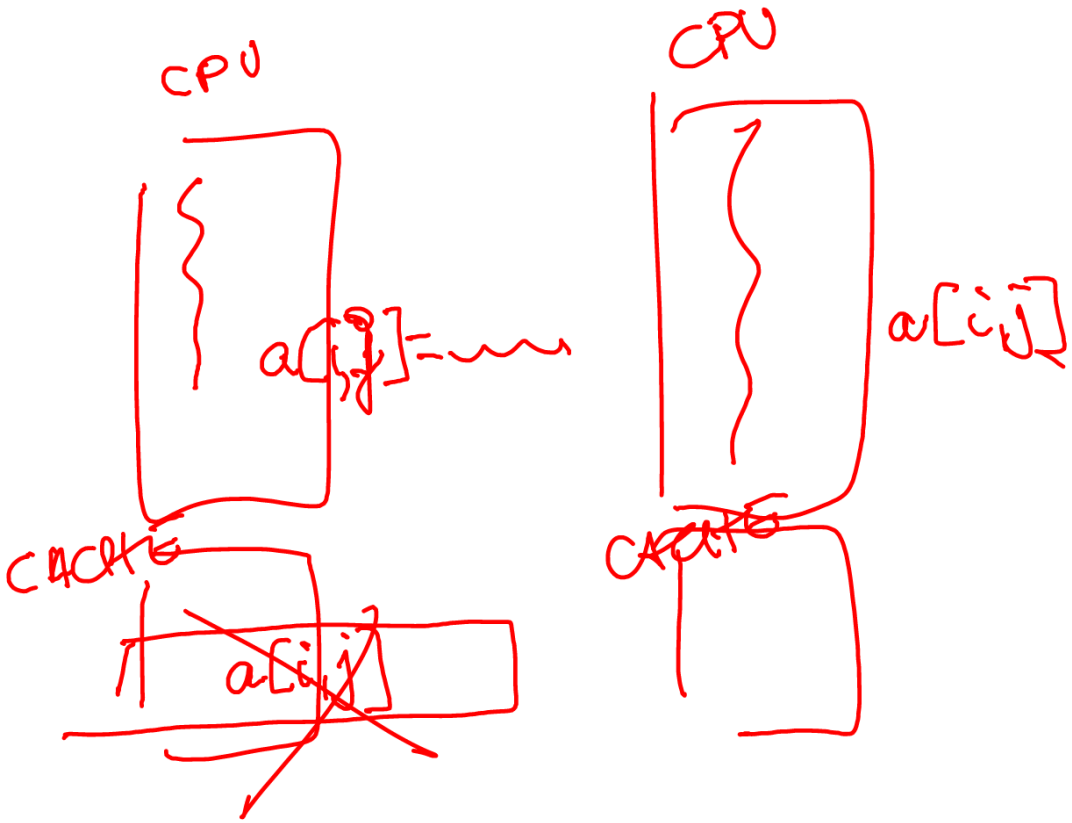

assíncrono

# Comunicação – verificação

- todas as tarefas têm aproximadamente a mesma carga de comunicação?
  - distribuição ou replicação de estruturas de dados
- várias operações de comunicação podem ocorrer concorrentemente?
- e a computação? pode ocorrer concorrentemente?

**em memória compartilhada**

- custos se referem principalmente à sincronização e a acessos a caches
  - problema do falso compartilhamento!


CPU
CPU
a[i,j]
a[i,j]
CACHE
CACHE
a[i,j]

# Aglomeração

- possibilidade de combinar tarefas
  - já pode levar em conta arquiteturas específicas
    - deixar o número de tarefas igual ao de processadores?
  - diminuição dos custos de comunicação e de troca de contexto
    - replicação de computação

# Aglomeração

- objetivos podem ser conflitantes
  - granularidade X custos de comunicação e criação de processos
  - n. tarefas $>$ processadores $\Rightarrow$ possibilidade de sobrepor comunicação e computação
  - maior número de tarefas facilita balanceamento de carga

# Aglomeração – verificação

- diminuímos a comunicação?
- houve ganhos com replicação de computação?
- se replicamos dados, isso não compromete a escalabilidade?
  - ... o número de tarefas ainda cresce com o tamanho do problema?
- ainda há concorrência suficiente?

# Mapeamento

1. *para sistemas de memória compartilhada:* não visível diretamente para o programador
2. colocar processos que podem executar concorrentemente em máquinas diferentes
3. colocar processos que se comunicam com frequência na mesma máquina
   - ... claramente pode haver conflitos

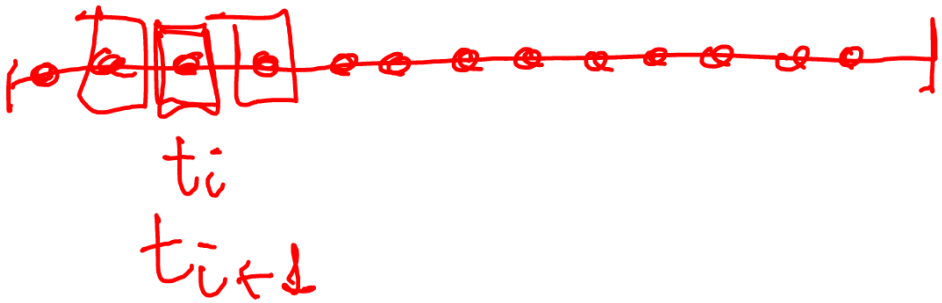
- problema geral de mapeamento é NP-completo
- na prática muitas vezes situações mais simples...

# Mapeamento – verificação

- analisamos diferentes estratégias de criação de processos?
  - SPMD x criação dinâmica de processos

# Exemplo: temperatura numa vara

- cálculo da evolução da temperatura ao longo do tempo

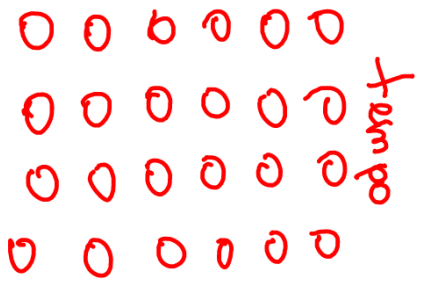



# Como paralelizar

1. particionamento
   - cada ponto da matriz tempo x posição é uma tarefa
2. comunicação
   - cálculo em cada ponto depende do resultado de outros três
3. aglomeração e mapeamento
   - não há como trabalhar em paralelo em diferentes pontos de uma coluna
   - aglomeração de algumas posições

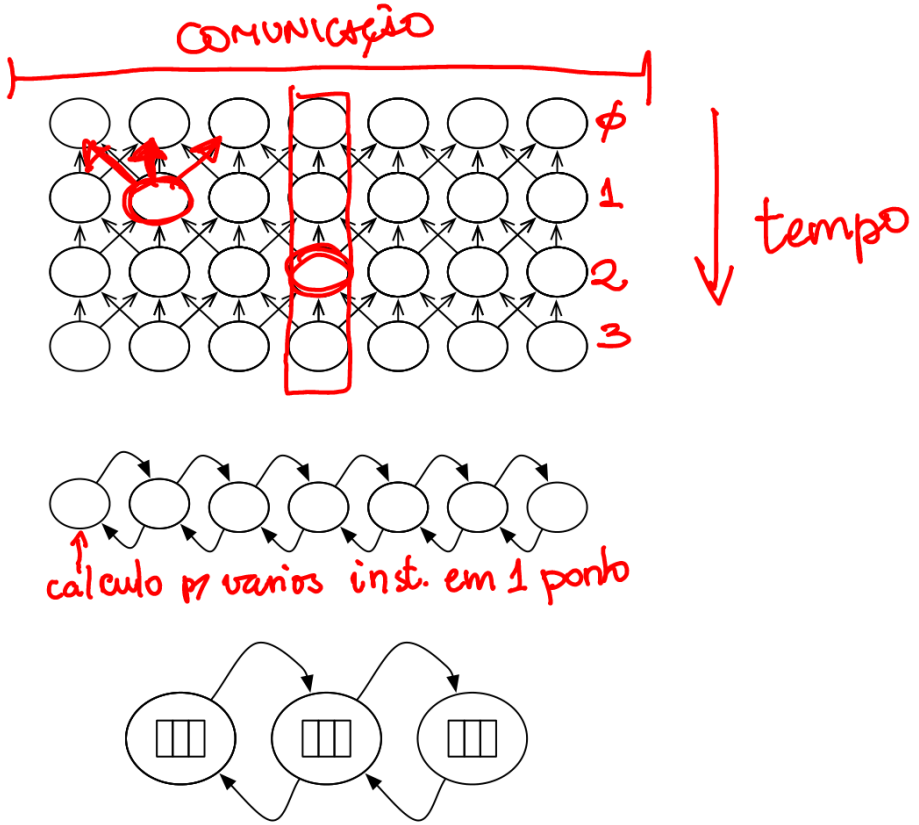
PARTICIONAMENTO

posição



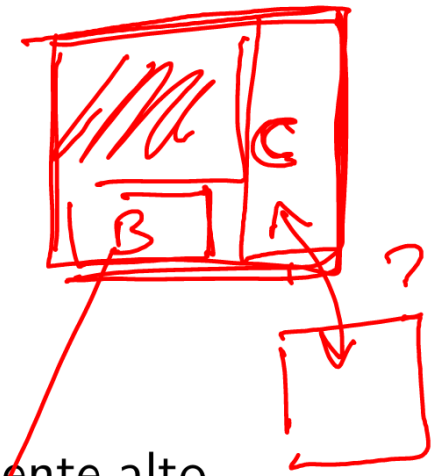

COMUNICAÇÃO

# Aglomeração



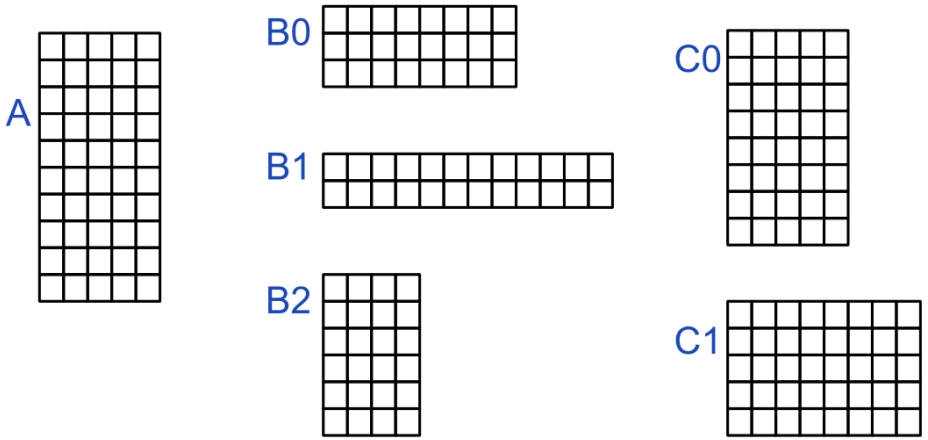
# Exemplo: projeto de componentes

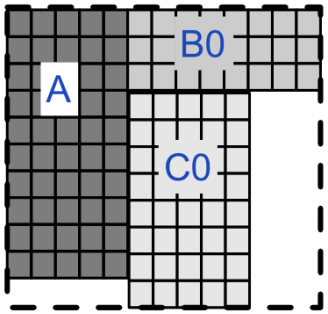
- n células têm que ser dispostas em uma placa
- existem restrições em relação às posições relativas
- exploração de todas as soluções tem custo proibitivamente alto
- uso de estratégia *branch and bound*



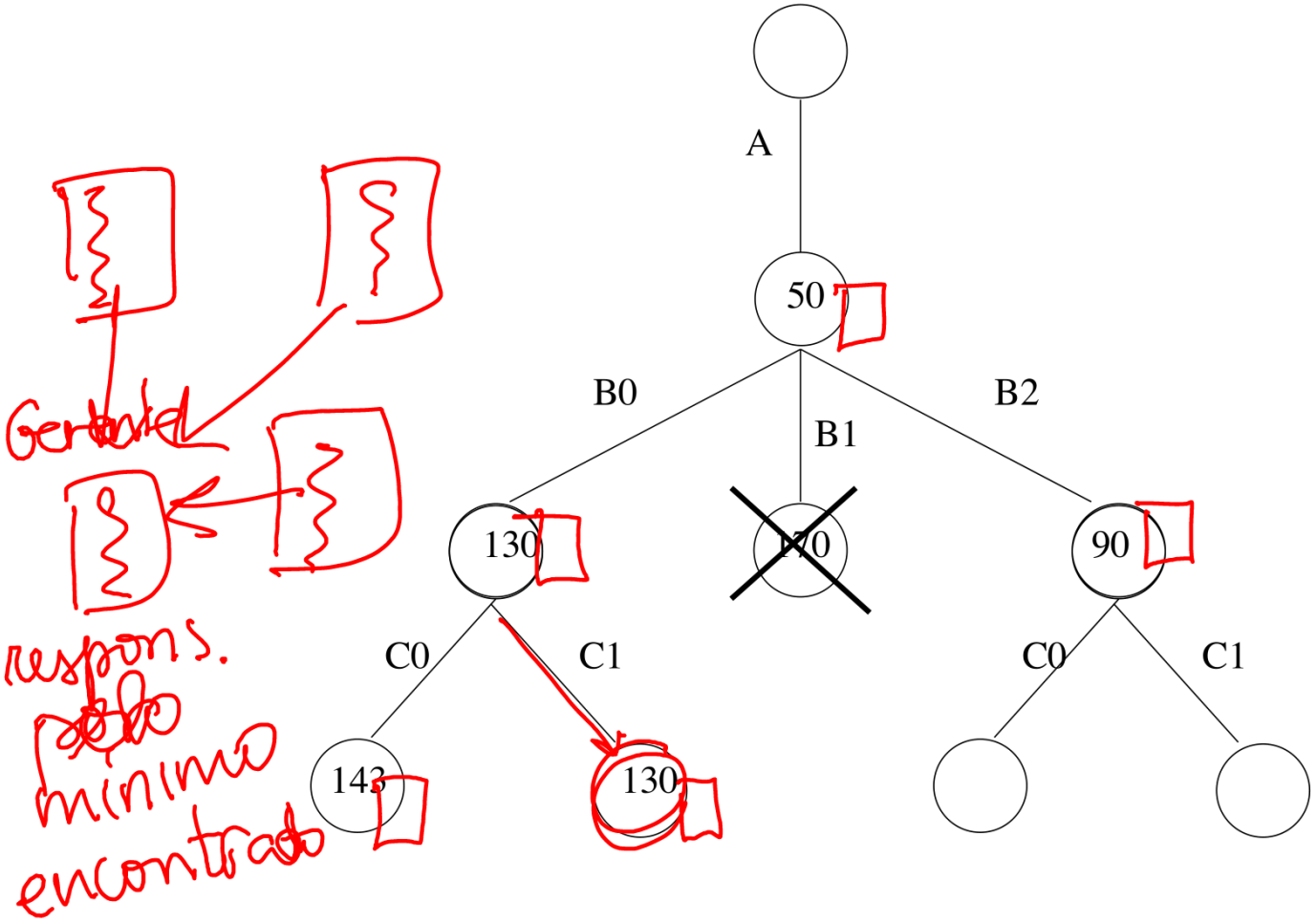

vêm componentes em formatos alternativos

# Componentes — alternativas




# Branch and Bound



# Branch and Bound – P

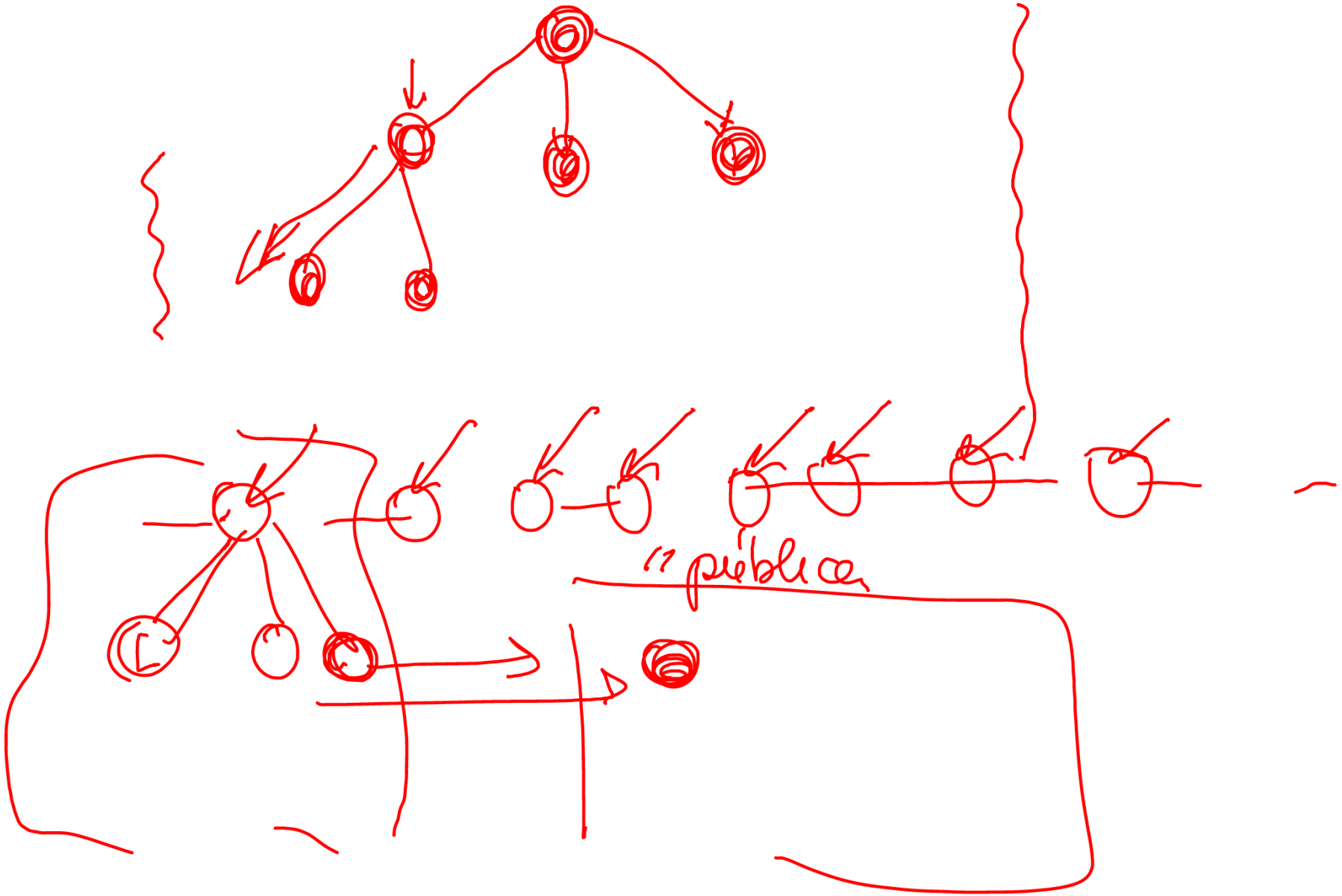
- particionamento: criação de processos para buscas paralelas
  - em primeira abordagem uma por nó...

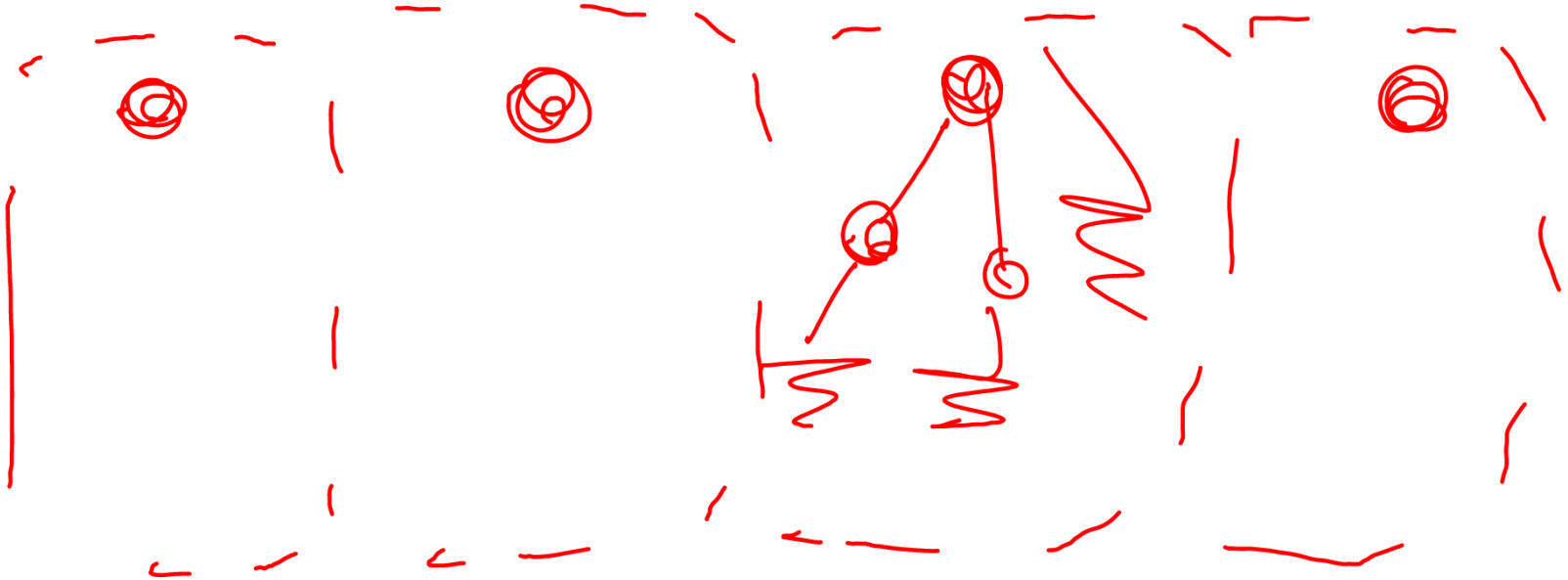
# Branch and Bound – C

- comunicação: divulgação de custos totais (mínimos parciais)
- balanço entre atualização do mínimo encontrado e economia de comunicação
  - gerente central pode manter soluções e mínimo
    - escalável?

# Branch and Bound – AM

- aglomeração: um único processo pode explorar sub-árvore
- mapeamento: alternativas
  - único processo raiz distribui tarefas para trabalhadores
  - diversos processos replicam trabalho inicial e assumem cada um uma sub-árvore a partir de algum ponto
    - padrão de replicação para evitar comunicação


"public



# Bibliografia


- I. Foster. Developing and Building Parallel Programs. (disponível em `www.mcs.anl.gov/~itf/dbpp/text/book.html`)
- Guy Steele. The Future Is Parallel: What's a Programmer to Do? Breaking Sequential Habits of Thought