

# Computação Paralela e Distribuída

Ano lectivo 2023-24

Rascunho - Será reorganizado e actualizado

Tema #4: Programação em Sistemas de Memória Distribuída  
Título: Metodologia de Foster

**João José da Costa**  
joao.costa@isptec.co.ao

**Coordenação de Engenharia Informática**  
Departamento de Engenharias e Tecnologias  
Instituto Superior Politécnico de Tecnologias e Ciências

# Metodologia de Foster

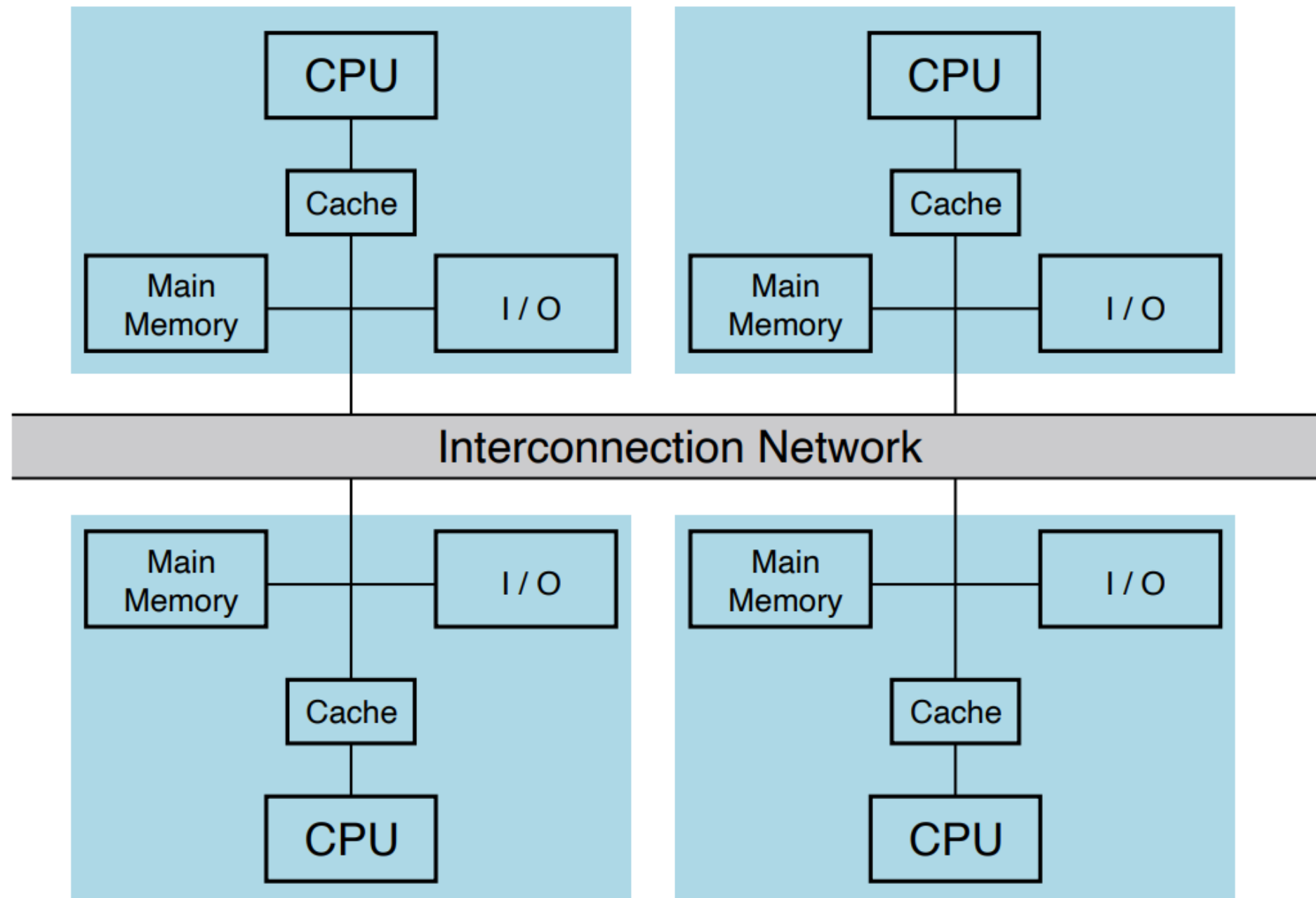
Projecto de programa paralelo para arquitectura de  
memória distribuída.

# Tópicos

- Programação de sistema de memória distribuída
- Metodologia de projecto de Foster
  - Particionamento
  - Comunicação
  - Aglomeração
  - Mapeamento
- Exemplos de aplicação

# Arquitetura

## Sistemas de Memória Distribuída



# Memória distribuída

## Sistemas de Memória Distribuída

- Programação paralela de sistemas de memória distribuída é significativamente diferente dos sistemas de memória partilhada essencialmente devido ao grande *overhead* em termos de:
  - Comunicação
  - Inicialização/término de tarefa (processo)
- Paralelização eficiente requer que esses efeitos seja tidos em conta desde o princípio!
- Cada processador recebe uma tarefa (grande)
  - **escalonamento estático**: todas as tarefas começam no início da computação
  - **escalonamento dinâmico**: as tarefas começam conforme necessário
- A aplicação é tipicamente um único programa
  - O número de identificação de cada tarefa indica qual é o seu trabalho

# Memória distribuída

## Modelo Tarefa/Canal

- Programação paralela para sistemas de passagem de mensagem utiliza um modelo Tarefa/Canal
- Computação paralela é representado como um conjunto de tarefas que podem se comunicar pelo envio de mensagens pelo canal
  - **Tarefa**: programa + memória local + portas de I/O
  - **Canal**: filas de mensagens que conecta a porta de saída de uma tarefa à porta de entrada de outra tarefa
- Todas as tarefas iniciam simultaneamente e o tempo de término é determinado pelo momento em que a última tarefa interrompe sua execução.

# Memória distribuída

## Modelo Tarefa/Canal

- A ordem dos dados no canal é mantida
- Receber blocos de tarefas até que um valor esteja disponível no receptor
- O remetente nunca bloqueia, independentemente de mensagens anteriores ainda não serem entregues
- No modelo Tarefa/Canal
  - A recepção é uma operação síncrona
  - O envio é uma operação assíncrona

# Metodologia de Foster

## Metodologia de Programação

- Um dos métodos mais conhecidos para projectar algoritmos paralelos é a metodologia de Ian Foster (1996).
- Esta metodologia permite que o programador se concentre inicialmente nos aspectos **não-dependentes da arquitectura**, como sejam a **concorrência** e a **escalabilidade**, e só depois considere os aspectos **dependentes da arquitectura**, como sejam **aumentar a localidade** e **diminuir a comunicação da computação**.
- Ou seja, permite o desenvolvimento de algoritmos paralelos escaláveis atrasando decisões dependentes de máquina para estágios posteriores.



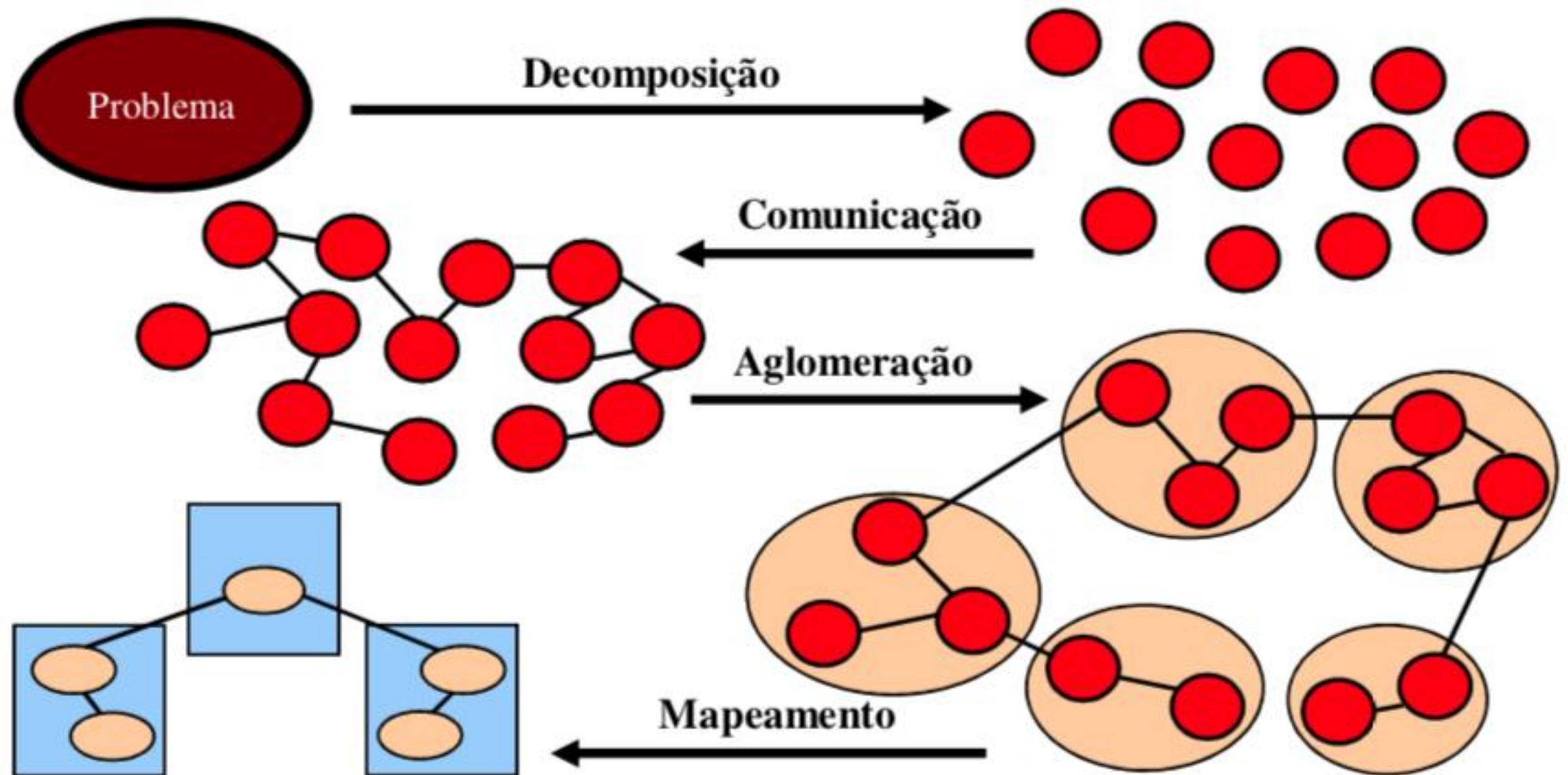
# Metodologia de Foster

## Metodologia de Programação

- A metodologia de programação de Foster divide-se em 4 etapas:
  - Decomposição
  - Comunicação
  - Aglomeração
  - Mapeamento

# Metodologia de Foster

## Metodologia de Programação



# Metodologia de Foster

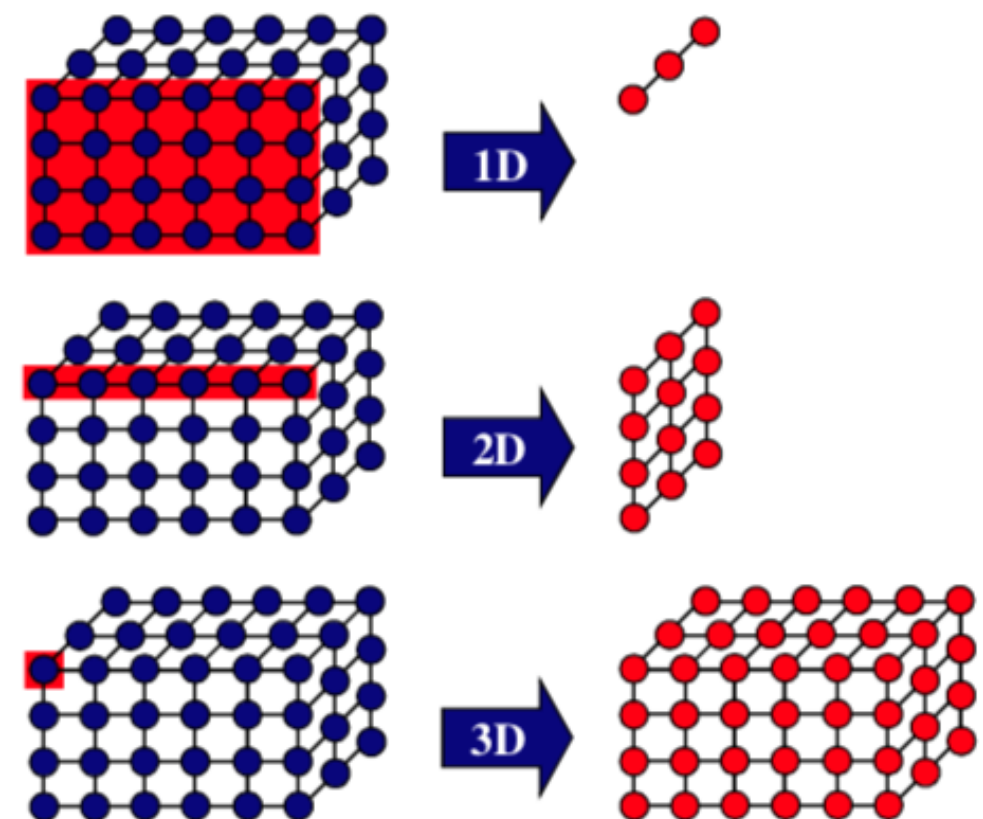
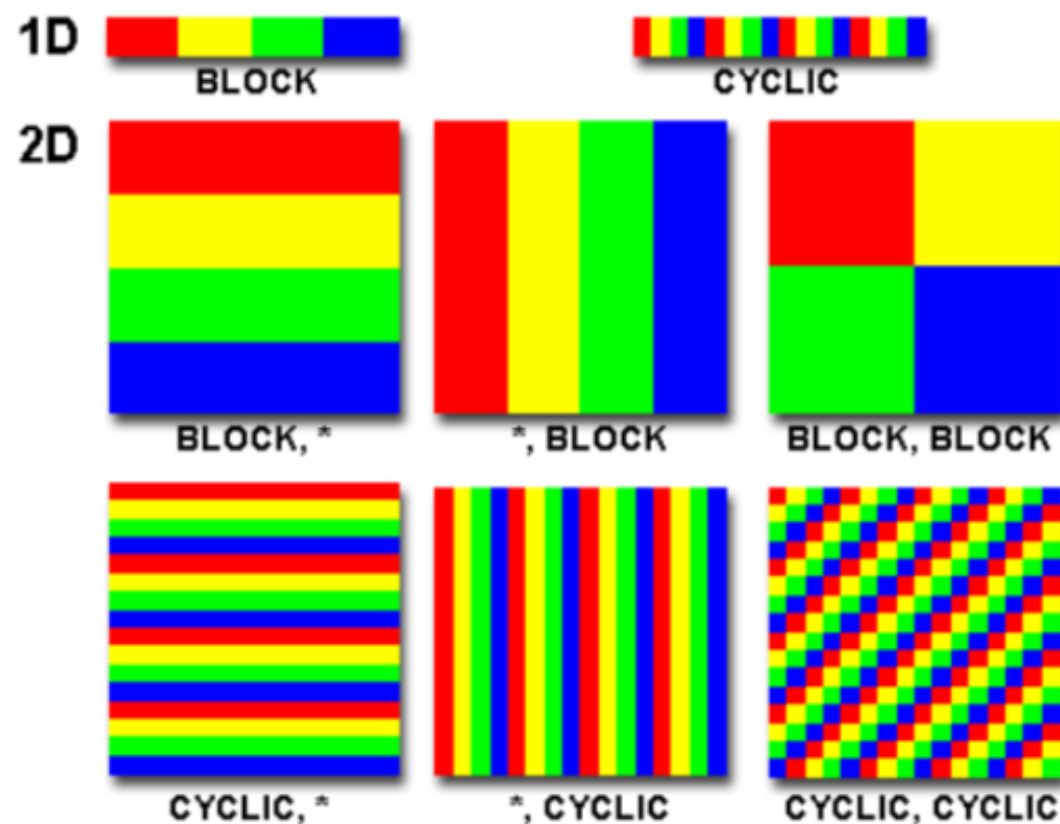
## Decomposição

- Uma forma de **diminuir a complexidade** de um problema é conseguir **dividi-lo em tarefas mais pequenas** de modo a **aumentar a concorrência** e a **localidade de referência** de cada tarefa.
- **Particionamento** – é o processo de dividir a computação e os dados em muitas pequenas tarefas primitivas.
- Existem duas estratégias principais de decompor um problema:
  - ✓ **Decomposição do Domínio**: decompor o problema em função dos dados.
  - ✓ **Decomposição Funcional**: decompor o problema em função da computação.
- Um boa decomposição tanto divide os dados como a computação em múltiplas tarefas mais pequenas.

# Metodologia de Foster

## Decomposição do Domínio

- Tipo de decomposição em que **primeiro se dividem os dados em partições** e só depois se determina o processo de associar a computação com as partições.
- Todas as tarefas executam as mesmas operações.



# Metodologia de Foster

## Decomposição do Domínio

- O particionamento de dados apropriado é fundamental para o desempenho paralelo.
- Etapas
  - Identificar os dados nos quais as computações são realizadas
  - Particionar os dados em várias tarefas
- A decomposição pode ser baseada em
  - Dados de entrada
  - Dados de saída
  - Dados de entrada+saída
  - Dados intermédio

# Metodologia de Foster

## Decomposição do Domínio – dado de entrada

- Aplicável se cada saída for computada como uma função de entrada
- Pode ser a única decomposição natural se a saída for desconhecida
  - Problema de encontrar o mínimo em um conjunto ou outras reduções
- Associar uma tarefa a cada partição de dados de entrada
  - A tarefa realiza computação em sua parte dos dados
  - O processamento subsequente combina resultados parciais de tarefas anteriores



# Metodologia de Foster

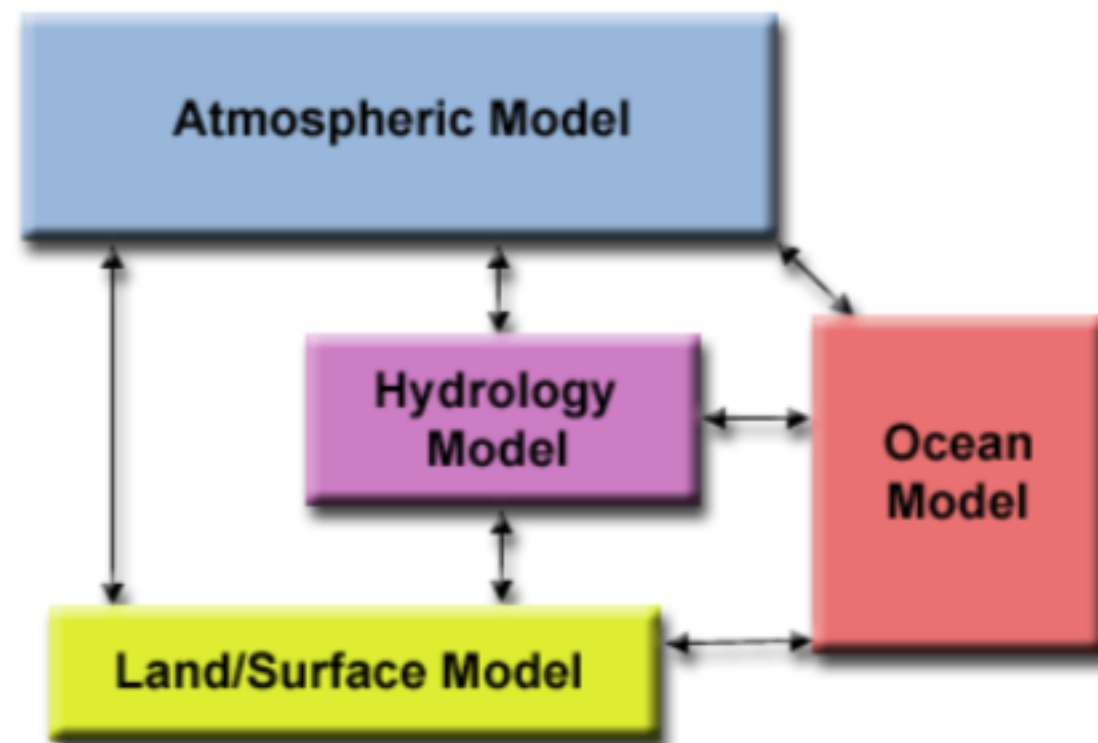
## Decomposição do Domínio – dado de saída

- Aplicável se cada elemento da saída puder ser computado de forma independente
  - O algoritmo é baseado em funções um para um ou muitos para um
- Particionar os dados de saída entre as tarefas
- Garantir que cada tarefa execute a computação para suas saídas
- Exemplo: multiplicação de vectore-matriz.

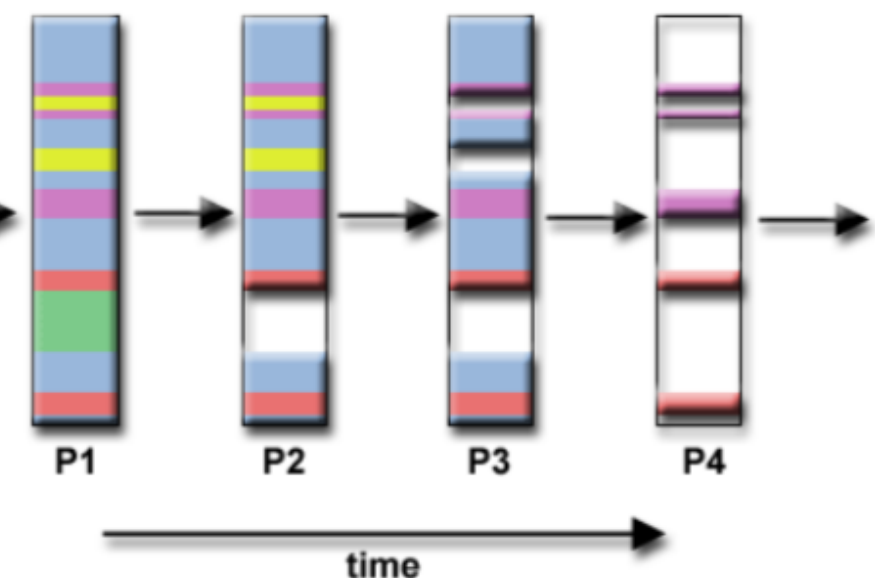
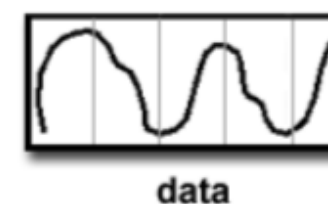
# Metodologia de Foster

## Decomposição Funcional

- Tipo de decomposição em que **primeiro se divide a computação em partições** e só depois se determina o processo de associar os dados com cada partição.
- Diferentes tarefas executam diferentes operações.



Modelação climática



Processamento de sinal



# Metodologia de Foster

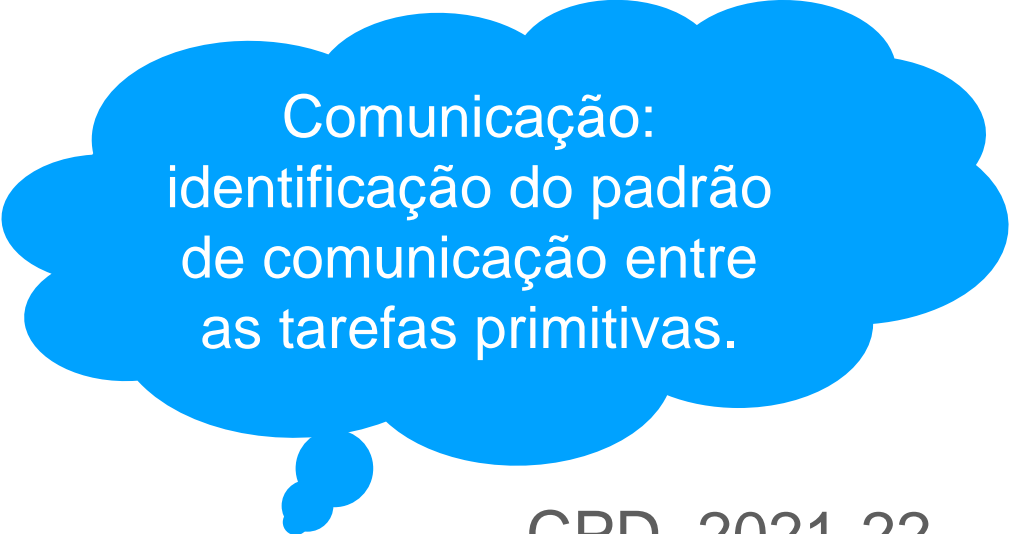
## Decomposição Recursiva – pouco usada

- Adequada para problemas solucionáveis utilizando divide-e-conquista
- Etapas
  - Decompor um problema em um conjunto de subproblemas
  - Decompor recursivamente cada subproblema
  - Parar a decomposição quando a granularidade mínima desejada for atingida

# Metodologia de Foster

## Comunicação

- A **natureza do problema** e o **tipo de decomposição** **determinam o padrão de comunicação** entre as diferentes tarefas. A execução de uma tarefa pode envolver a sincronização/acesso a dados pertencentes/calculados por outras tarefas.
- Para haver **cooperação entre as tarefas** é necessário definir **algoritmos** e **estruturas de dados** que permitam uma **eficiente troca de informação**. Alguns dos principais factores que limitam essa eficiência são:
  - ✓ **Custo da Comunicação**
  - ✓ **Necessidade de Sincronização**
  - ✓ **Latência e Largura de Banda**



Comunicação:  
identificação do padrão  
de comunicação entre  
as tarefas primitivas.

# Metodologia de Foster

## Comunicação

- Alguns dos principais factores que limitam essa eficiência são:
  - ✓ **Custo da Comunicação**: existe sempre um custo associado à troca de informação e enquanto as tarefas processam essa informação não contribuem para a computação.
  - ✓ **Necessidade de Sincronização**: enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.
  - ✓ **Latência** (tempo mínimo de comunicação entre dois pontos) e **Largura de Banda** (quantidade de informação comunicada por unidade de tempo): é boa prática enviar poucas mensagens grandes do que muitas mensagens pequenas.

# Metodologia de Foster

## Padrões de Comunicação

- Comunicação Global:
  - ✓ Todas as tarefas podem comunicar entre si.
- Comunicação Local:
  - ✓ A comunicação é restrita a tarefas vizinhas
- Checklist:
  - ✓ Comunicação equilibrada entre as tarefas
  - ✓ Cada tarefa se comunica com um pequeno número de tarefas
  - ✓ As tarefas podem realizar sua comunicação simultaneamente
  - ✓ As tarefas podem realizar suas computações simultaneamente

# Metodologia de Foster

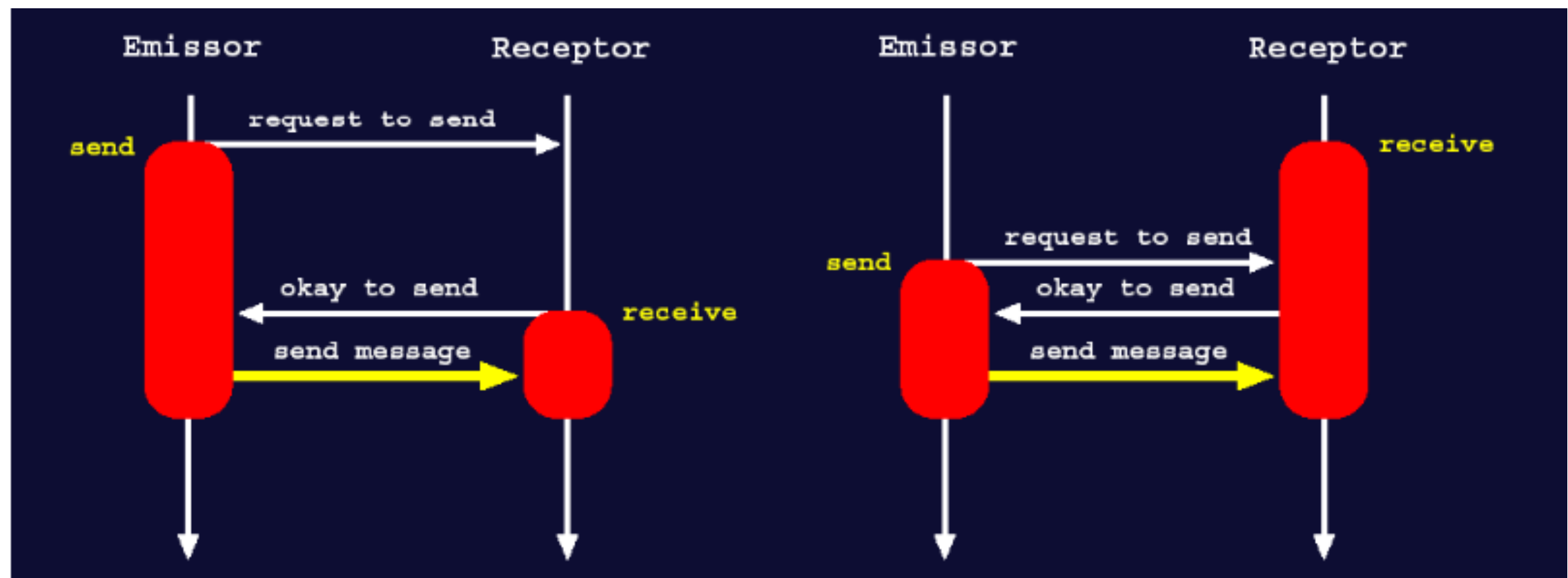
## Padrões de Comunicação

- Comunicação Estruturada:
  - ✓ Tarefas vizinhas constituem uma estrutura regular (e.g. árvore ou rede).
- Comunicação Não-Estruturada:
  - ✓ Comunicação entre tarefas constitui um grafo arbitrário.
- Comunicação Estática:
  - ✓ Os parceiros de comunicação não variam durante toda a execução.
- Comunicação Dinâmica:
  - ✓ A comunicação é determinada pela execução e pode ser muito variável.

# Metodologia de Foster

## Padrões de Comunicação

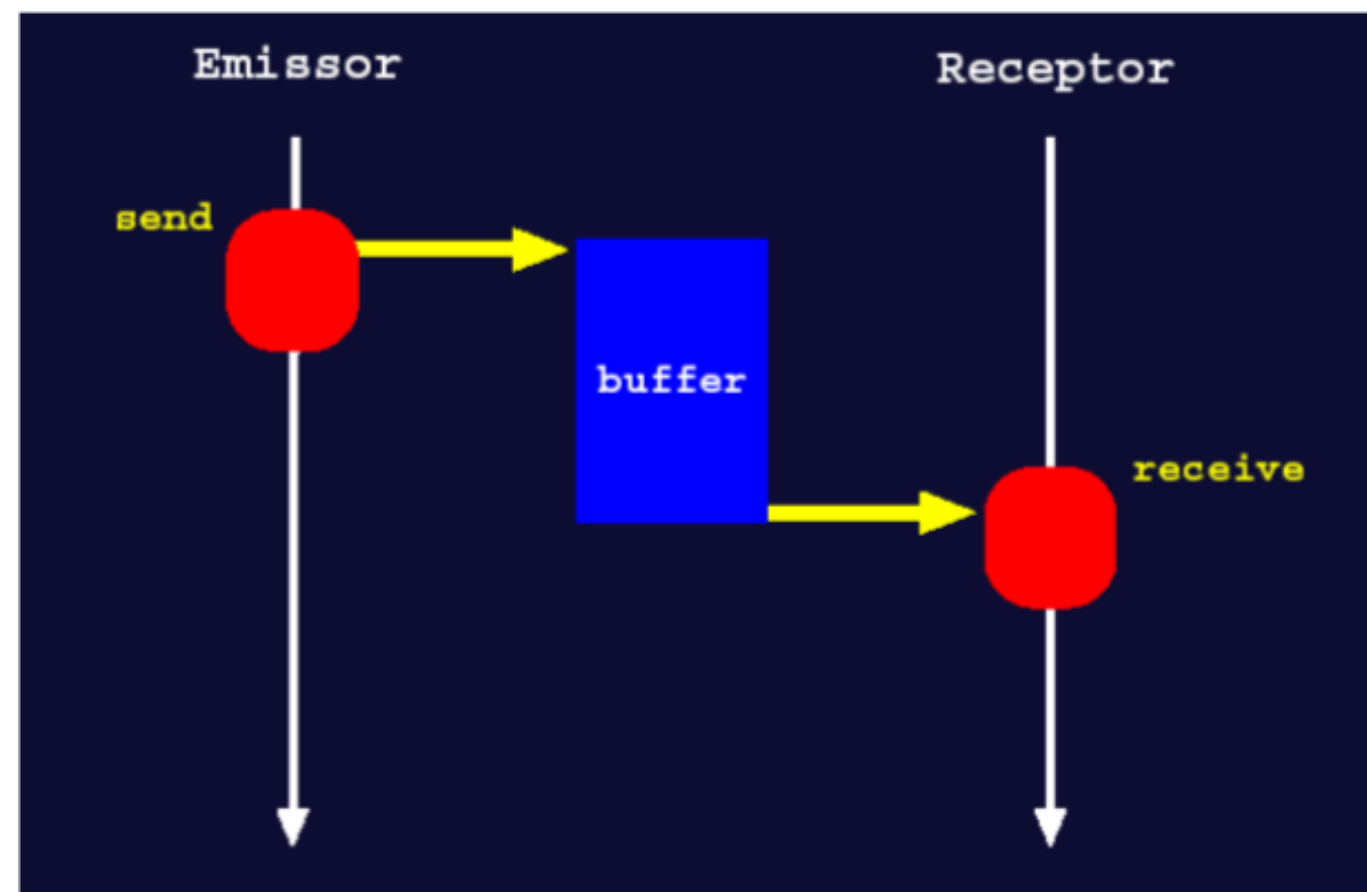
- Comunicação Síncrona:
  - ✓ As tarefas executam de forma coordenada e sincronizam na transferência de dados (e.g. protocolo das 3-fases ou rendez-vous: a comunicação apenas se concretiza quando as duas tarefas estão sincronizadas).



# Metodologia de Foster

## Padrões de Comunicação

- Comunicação Assíncrona:
  - ✓ As tarefas executam de forma independente não necessitando de sincronizar para transferir dados (e.g. buffering de mensagens: o envio de mensagens não interfere com a execução do emissor).



# Metodologia de Foster

## Aglomeração

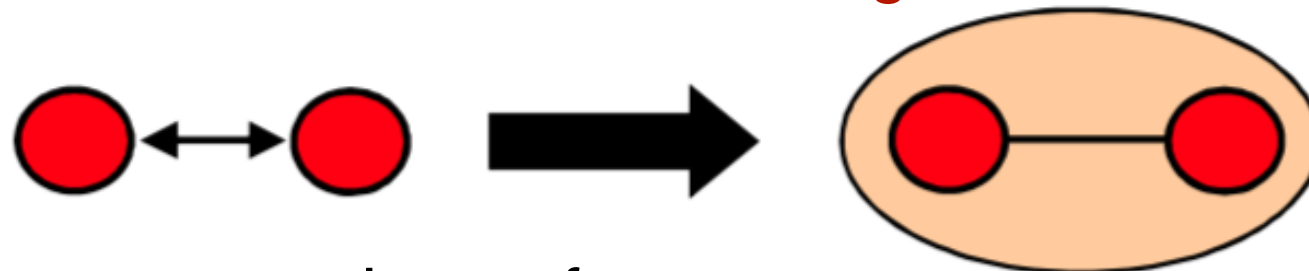
- Aglomeração é o processo de agrupar tarefas primitivas em tarefas maiores de modo a diminuir os custos de implementação do algoritmo paralelo e os custos de comunicação entre as tarefas.
- Custos de implementação do algoritmo paralelo:
  - ✓ O agrupamento em tarefas maiores permite uma maior reutilização do código do algoritmo serial na implementação do algoritmo paralelo.
  - ✓ No entanto, o agrupamento em tarefas maiores deve garantir a escalabilidade do algoritmo paralelo de modo a evitar posteriores alterações (p.e. optar por aglomerar as duas últimas dimensões duma matriz de dimensão  $8 \times 128 \times 256$  restringe a escalabilidade a um máximo de 8 processadores).



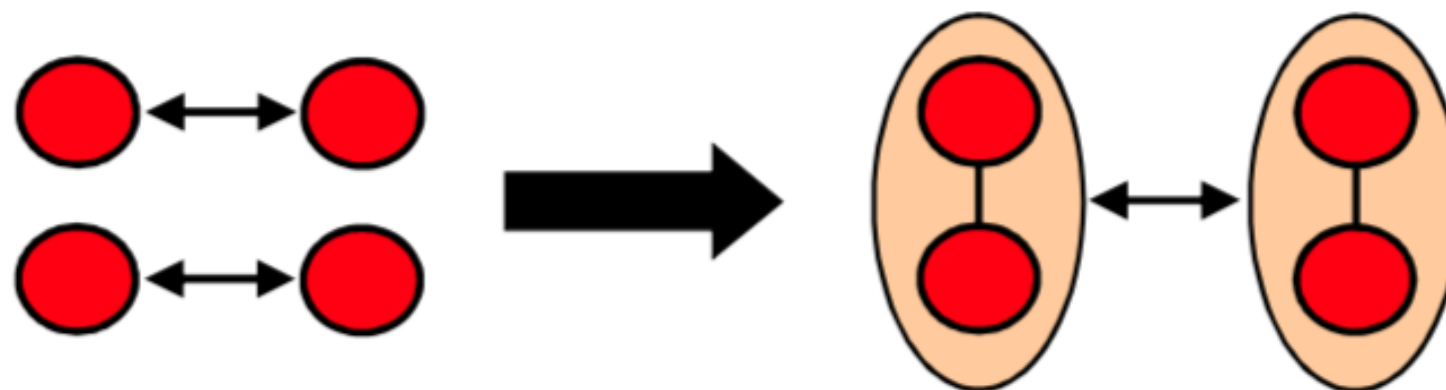
# Metodologia de Foster

## Aglomeração

- Custos de comunicação entre as tarefas:
  - ✓ O agrupamento de tarefas elimina os custos de comunicação entre essas tarefas e aumenta a **granularidade da computação**.



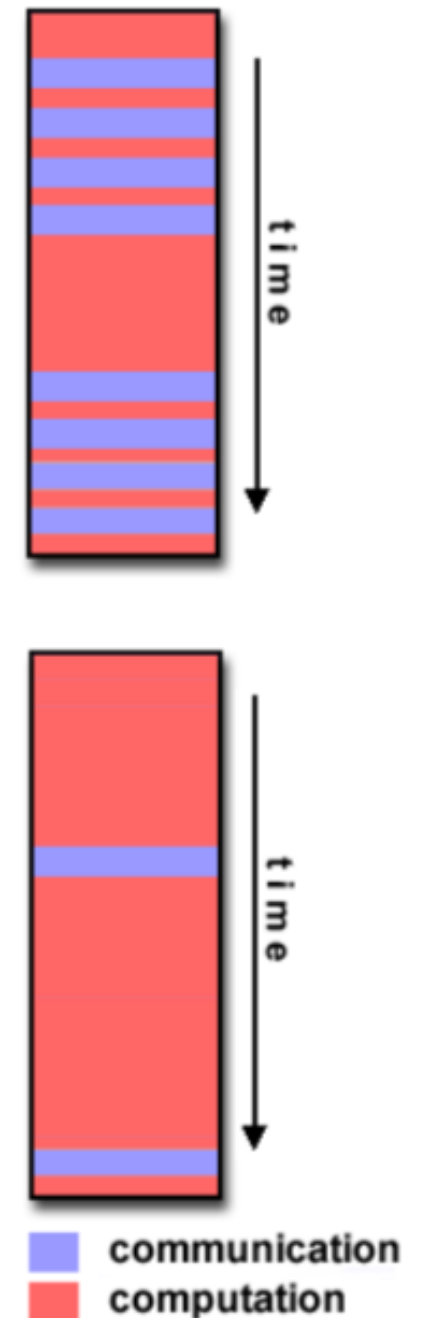
- ✓ O agrupamento de tarefas com pequenas comunicações individuais em tarefas com comunicações maiores permite aumentar a **granularidade das comunicações** e reduzir o número total de comunicações.



# Metodologia de Foster

## Aglomeración - Granularidade

- Períodos de computação são tipicamente separados por períodos de comunicação entre as tarefas. A granularidade é a medida qualitativa do rácio entre computação e comunicação.
- O número e o tamanho das tarefas em que a computação é agrupada determina a sua granularidade. A granularidade pode ser **fina**, **média** ou **grossa**.



**“Como agrupar a computação de modo a obter o máximo desempenho?”**

# Metodologia de Foster

## Aglomeración - Granularidade

- Granularidade Fina
  - ✓ A computação é agrupada num grande número de pequenas tarefas.
  - ✓ O rácio entre computação e comunicação é baixo.
  - ✓ Fácil de conseguir um balanceamento de carga eficiente.
  - ✓ O tempo de computação de uma tarefa nem sempre compensa os custos de criação, comunicação e sincronização.
  - ✓ Difícil de se conseguir melhorar o desempenho.

# Metodologia de Foster

## Aglomeración - Granularidade

- Granularidade Grossa
  - ✓ A computação é agrupada num pequeno número de grandes tarefas.
  - ✓ O rácio entre computação e comunicação é grande.
  - ✓ Difícil de conseguir um balanceamento de carga eficiente.
  - ✓ O tempo de computação compensa os custos de criação, comunicação e sincronização.
  - ✓ Oportunidades para se conseguir melhorar o desempenho.

# Metodologia de Foster

## Aglomeração

- Checklist
  - ✓ A localidade foi maximizada
  - ✓ As computações replicadas levam menos tempo do que as comunicações que substituem
  - ✓ A quantidade de dados replicados é pequeno o suficiente para permitir o dimensionamento do algoritmo
  - ✓ As tarefas são equilibradas em termos de computação e comunicação
  - ✓ O número de tarefas cresce naturalmente com o tamanho do problema
  - ✓ O número de tarefa é pequeno, mas pelo menos tão grande quanto  $P$
  - ✓ O custo de modificações no código sequencial é minimizado

# Metodologia de Foster

## Mapeamento

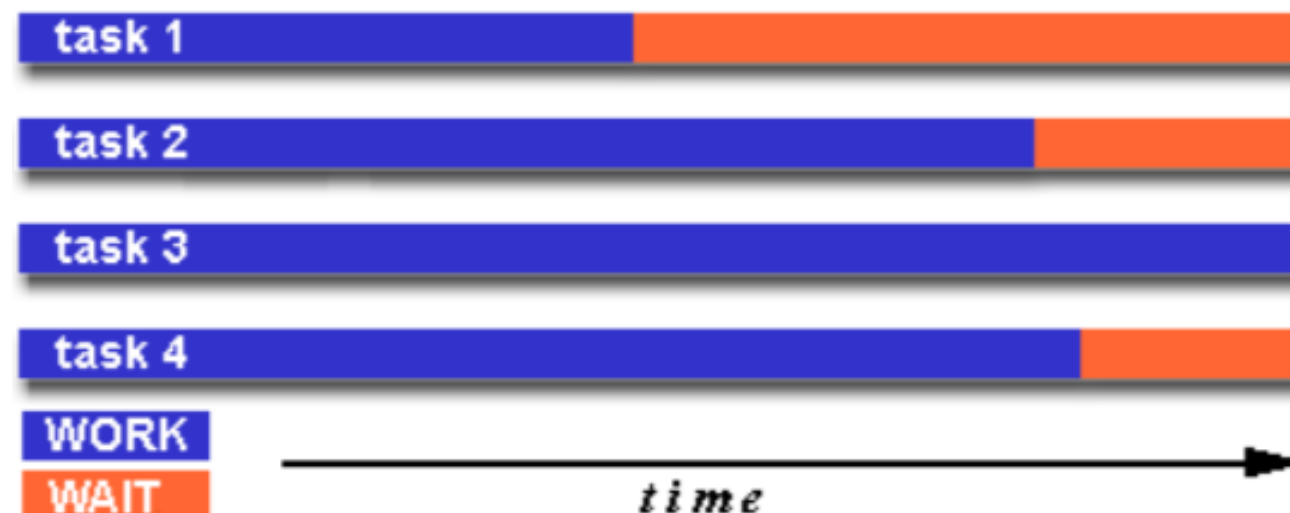
- Mapeamento é o processo de **atribuir tarefas** a processadores de modo a **maximizar a percentagem** de ocupação e **minimizar a comunicação** entre processadores.
  - ✓ A **percentagem de ocupação** é óptima quando a computação é **balanceada de forma igual pelos processadores**, permitindo que todos comecem e terminem as suas tarefas em simultâneo. A percentagem de ocupação decresce quando um ou mais processadores ficam suspensos enquanto os restantes continuam ocupados.
  - ✓ A **comunicação entre processadores é menor** quando **tarefas** que comunicam entre si são **atribuídas ao mesmo processador**. No entanto, este mapeamento nem sempre é compatível com o objectivo de maximizar a percentagem de ocupação.

**“Como conseguir o melhor compromisso entre maximizar ocupação e minimizar comunicação?”**

# Metodologia de Foster

## Balanceamento de Carga

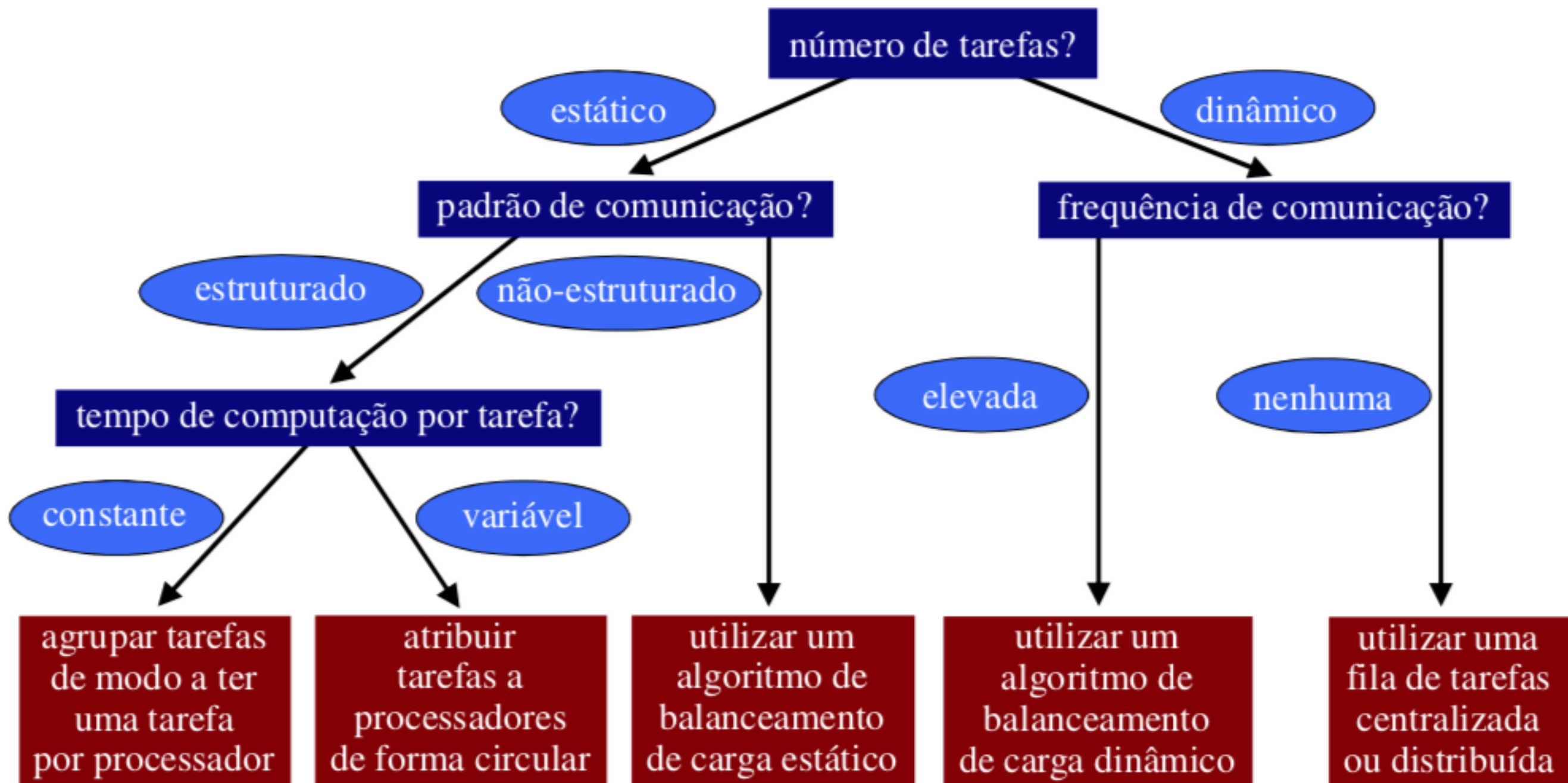
- O balanceamento de carga refere-se à capacidade de **distribuir tarefas** pelos processadores de modo a que **todos os processadores** estejam **ocupados todo o tempo**. O **balanceamento de carga** pode ser visto como uma **função de minimização do tempo** em que os processadores não estão ocupados.
- O balanceamento de carga pode ser estático (em tempo de compilação) ou dinâmico (em tempo de execução).





# Metodologia de Foster

## Balanceamento de Carga





# Metodologia de Foster

## Exemplo #1 – Soma dos elementos de um vector

- Problema: determinar a soma dos elementos de grande vector

# Lembre....

Revisão de conceitos fundamentais

# Limites de desempenho

## Factores que limitam o desempenho

- **Código Serial:** existem partes do código que são inerentemente seriais (p.e. iniciar/terminar a computação).
- **Concorrência:** o número de tarefas pode ser escasso e/ou de difícil definição. Comunicação: existe sempre um custo associado à troca de informação e enquanto as tarefas processam essa informação não contribuem para a computação.
- **Sincronização:** a partilha de dados entre as várias tarefas pode levar a problemas de contenção no acesso à memória e enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.

# Limites de desempenho

## Factores que limitam o desempenho

- **Granularidade**: o número e o tamanho das tarefas é importante porque o tempo que demoram a ser executadas tem de compensar os custos da execução em paralelo (e.g. custos de criação, comunicação e sincronização).
- **Balanceamento de Carga**: ter os processadores maioritariamente ocupados durante toda a execução é decisivo para o desempenho global do sistema.

# Tecnologias

## Modelo de Programação Paralela

### Modelo de programação paralelo

- uma arquitetura de computação e linguagem projectada para expressar paralelismo em sistemas de software e aplicações.
- **Abordagem padrão: deixe a tarefa de paralelização para o compilador!**
  - **Vantagens**
    - código existente facilmente portado
    - não há necessidade de aprender novas linguagens
    - os programadores podem se concentrar no desenvolvimento do algoritmo e não se distrair com a optimização do processo de paralelização

# Tecnologias

## Modelo de Programação Paralela

### Modelo de programação paralelo

- uma arquitetura de computação e linguagem projectada para expressar paralelismo em sistemas de software e aplicações.
- **Abordagem padrão: deixe a tarefa de paralelização para o compilador!**
  - **Desvantagens**
    - melhor algoritmo sequencial não mapeia necessariamente para uma boa implementação paralela
    - até mesmo o paralelismo inerente ao algoritmo pode ser ocultado pelo estilo de programação (ou seja, o compilador pode precisar redescobrir o paralelismo que o programador foi forçado a ocultar na linguagem de programação sequencial)
    - nenhum compilador eficiente foi desenvolvido

# Tecnologias

## Modelo de Programação Paralela

Abordagem alternative: criar uma nova linguagem de programação com construções de paralelismo explícito.

- Exemplos: Occam, HPF, ZPL, NESL e SISAL.
- **Vantagens**
  - o paralelismo inerente dos algoritmos pode ser descrito naturalmente
  - tarefa mais fácil para o compilador, portanto, executáveis mais eficientes são produzidos
- **Desvantagens**
  - novos compiladores precisam ser desenvolvidos, e estes levam algum tempo para amadurecer
  - deve suportar arquitecturas existentes e futuras
  - os programadores precisam aprender não apenas uma nova linguagem, mas um novo paradigma de programação

# Tecnologias

## Modelo de Programação Paralela

Outra abordagem: criar uma nova camada de programação, sendo a camada superior responsável pelas configurações paralelas (particionamento de dados e definição e criação de tarefas) e a camada inferior pela produção dos executáveis de cada tarefa.

- Exemplos: CODE, HeNCE.
- **Vantagens**
  - o paralelismo inerente dos algoritmos pode ser descrito naturalmente;
  - os compiladores existentes podem ser usados para gerar o executável
- **Desvantagens**
  - os programadores precisam aprender um novo sistema de programação
  - ferramentas maduras ainda indisponíveis



# Tecnologias

## Modelo de Programação Paralela

Abordagem mais popular: estender a linguagem existente com funções / directivas que permitem ao programador especificar as construções paralelas.

- Exemplos: OpenMP, PVM, MPI.
- **Vantagens**
  - pouca sobrecarga de aprendizado para um programador
  - reutilização de compiladores existentes, com pouca sobrecarga de desenvolvimento,
  - os programadores têm controlo total da implementação paralela
- **Desvantagens**
  - todo o trabalho de paralelização deixado para o programador
  - programação de baixo nível
  - nenhum suporte de compilador para verificação de erro

# Principais modelos

## Principais Modelos de Programação Paralela

- **Programação em Memória Partilhada**
  - ✓ Programação usando processos ou threads.
  - ✓ Decomposição do domínio ou funcional com granularidade fina, média ou grossa.
  - ✓ Comunicação através de memória partilhada.
  - ✓ Sincronização através de mecanismos de exclusão mútua.
- **Programação em Memória Distribuída**
  - ✓ Programação usando troca de mensagens.
  - ✓ Decomposição do domínio com granularidade grossa.
  - ✓ Comunicação e sincronização por troca de mensagens.

# Principais Paradigmas

## Principais Paradigmas de Programação Paralela

- Apesar da diversidade de problemas aos quais podemos aplicar a programação paralela, o desenvolvimento de algoritmos paralelos pode ser classificado num conjunto relativamente pequeno de diferentes paradigmas, em que cada paradigma representa uma classe de algoritmos que possuem o mesmo tipo de controle:
  - ✓ Master/Slave
  - ✓ Single Program Multiple Data (SPMD)
  - ✓ Data Pipelining
  - ✓ Divide and Conquer
  - ✓ Speculative Parallelism

# Principais Paradigmas

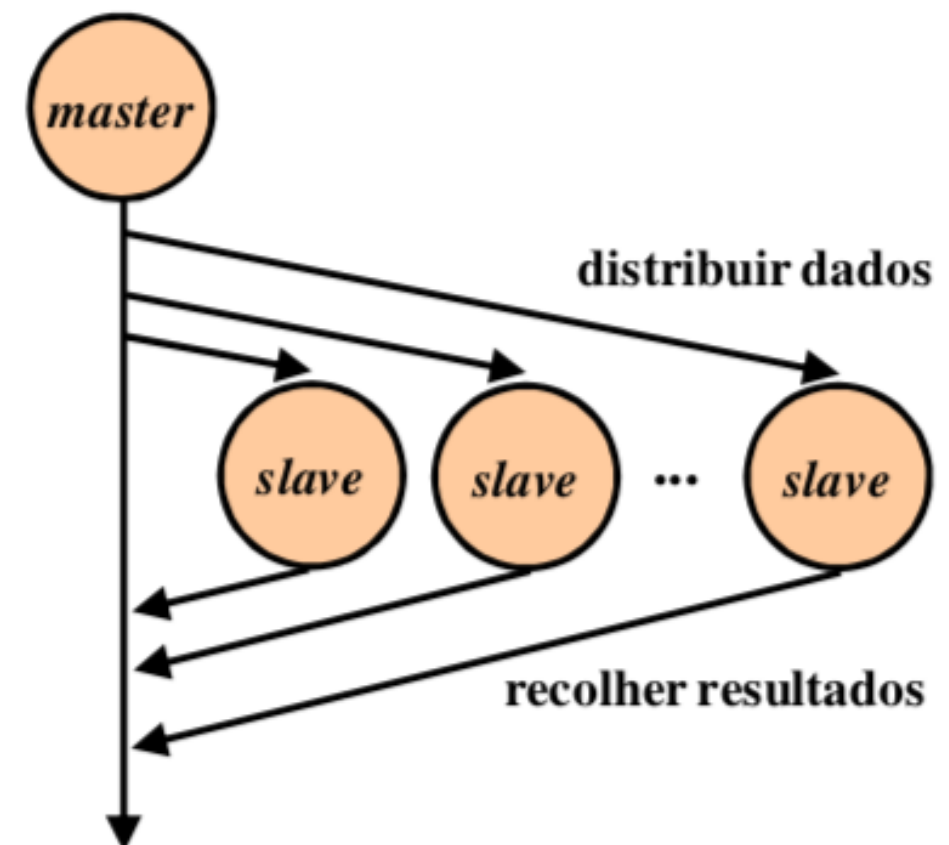
## Principais Paradigmas de Programação Paralela

- A escolha do paradigma a aplicar a um dado problema é determinado pelo:
  - ✓ Tipo de paralelismo inerente ao problema: decomposição do domínio ou funcional.
  - ✓ Tipo de recursos computacionais disponíveis: nível de granularidade que pode ser eficientemente suportada pelo sistema.

# Principais Paradigmas

## Master/Slave

- Este paradigma divide a computação em duas entidades distintas: o **processo master** e o **conjunto dos processos slaves**:
  - ✓ O master é o responsável por decompor o problema em tarefas, distribuir as tarefas pelos slaves e recolher os resultados parciais dos slaves de modo a calcular o resultado final.
  - ✓ O ciclo de execução dos slaves é muito simples: obter uma tarefa do master, processar a tarefa e enviar o resultado de volta para o master.



# Principais Paradigmas

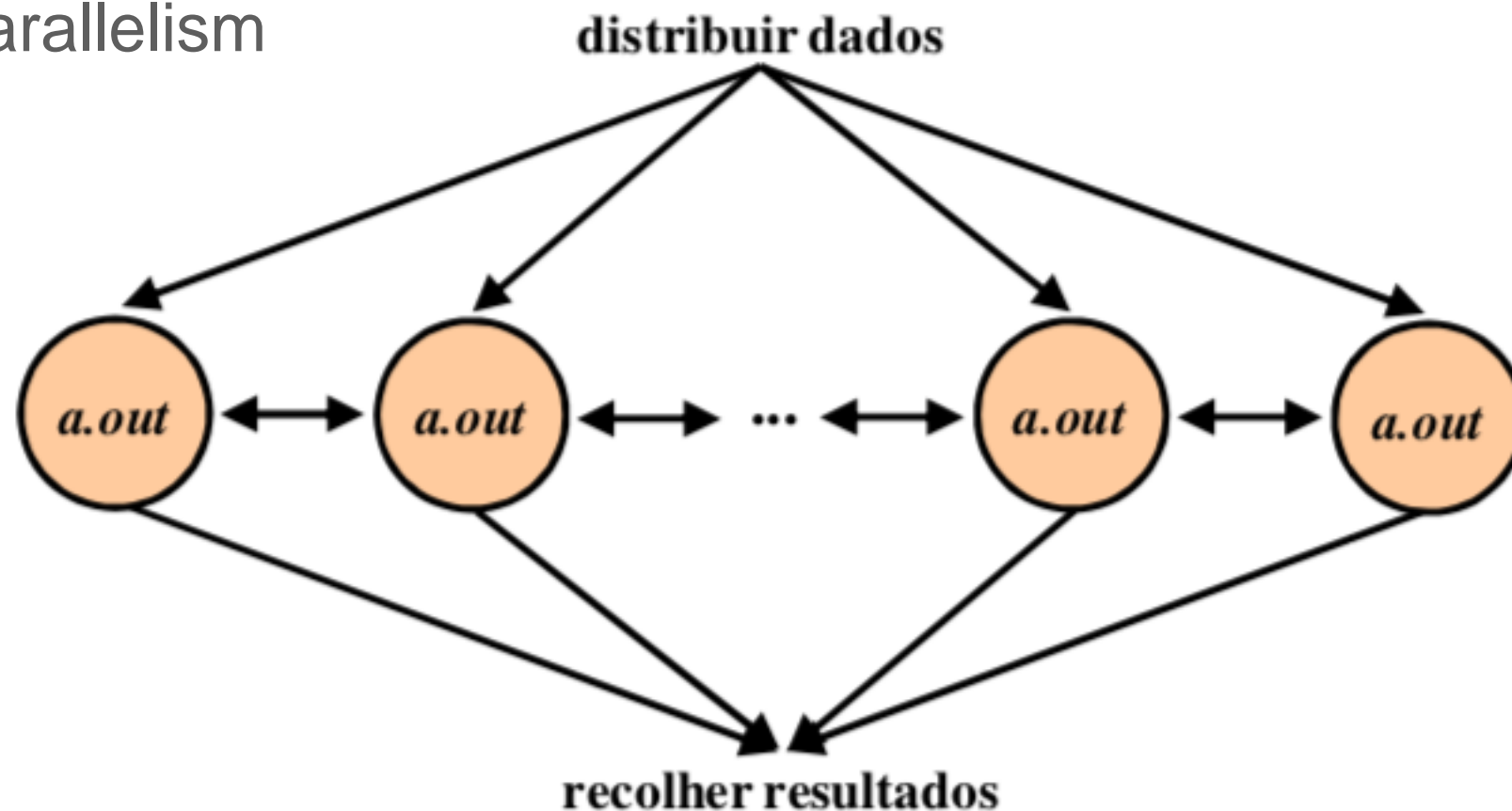
## Master/Slave

- O balanceamento de carga pode ser estático ou dinâmico:
  - ✓ É estático quando a divisão de tarefas é feita no início da computação. O balanceamento estático permite que o master também participe na computação.
  - ✓ É dinâmico quando o número de tarefas excede o número de processadores ou quando o número de tarefas ou o tempo de execução das tarefas é desconhecido no início da computação.
- Como só existe comunicação entre o master e os slaves, este paradigma consegue **bons desempenhos** e um **elevado grau de escalabilidade**.
  - ✓ No entanto, o controle centralizado no master pode ser um problema quando o número de slaves é elevado. Nesses casos é possível aumentar a escalabilidade do paradigma considerando vários masters em que cada um controla um grupo diferente de slaves.

# Principais Paradigmas

## Single Program Multiple Data (SPMD)

- Neste paradigma todos os processos executam o mesmo programa (executável) mas sobre diferentes partes dos dados. Este paradigma é também conhecido como:
  - ✓ Geometric Parallelism
  - ✓ Data Parallelism



# Principais Paradigmas

## Single Program Multiple Data (SPMD)

- Tipicamente, os dados são bem distribuídos (mesma quantidade e regularidade) e o padrão de comunicação é bem definido (estruturado, estático e local):
  - ✓ Os dados ou são lidos individualmente por cada processo ou um dos processos é o responsável por ler todos os dados e depois distribui-los pelos restantes processos.
  - ✓ Os processos comunicam quase sempre com processos vizinhos e apenas esporadicamente existem pontos de sincronização global.



# Principais Paradigmas

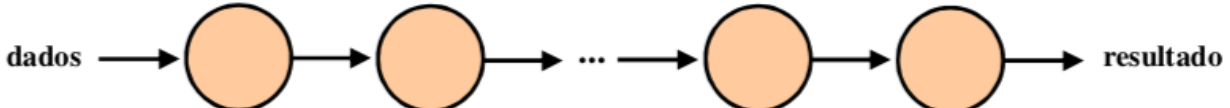
## Single Program Multiple Data (SPMD)

- Como os dados são bem distribuídos e o padrão de comunicação é bem definido, este paradigma consegue bons desempenhos e um elevado grau de escalabilidade.
  - ✓ No entanto, este paradigma é muito sensível a falhas. A perda de um processador causa necessariamente o encravamento da computação no próximo ponto de sincronização global.

# Principais Paradigmas

## Data Pipelining

- Este paradigma utiliza uma decomposição funcional do problema em que cada processo executa apenas uma parte do algoritmo total. Este paradigma é também conhecido como:

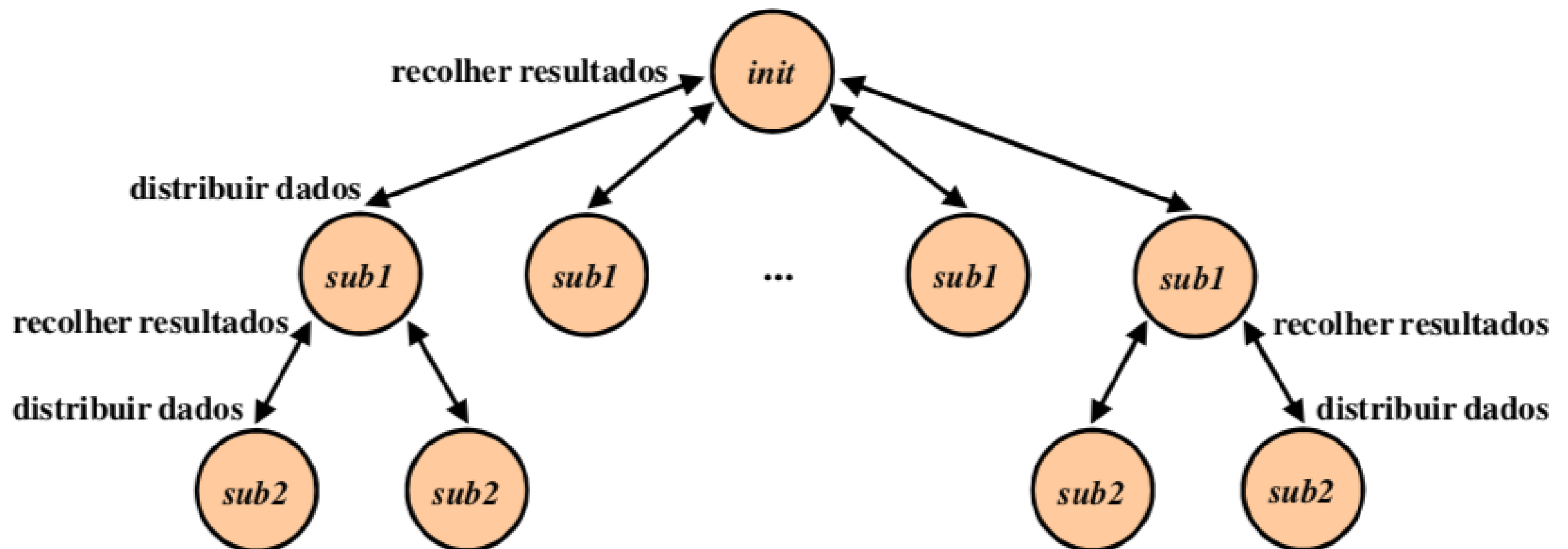
✓ Data Flow Parallelism 

- O padrão de comunicação é bem definido e bastante simples:
  - ✓ Os processos são organizados em sequência (*pipeline*) e cada processo só troca informação com o processo seguinte.
  - ✓ Toda a comunicação pode ser completamente assíncrona.
- O bom desempenho deste paradigma é directamente dependente da capacidade de balancear a carga entre as diferentes etapas da pipeline.

# Principais Paradigmas

## Divide and Conquer

- Este paradigma utiliza uma divisão recursiva do problema inicial em sub-problemas independentes (instâncias mais pequenas do problema inicial) cujos resultados são depois combinados para obter o resultado final.



# Principais Paradigmas

## Divide and Conquer

- A computação fica organizada numa espécie de árvore virtual:
  - ✓ Os processos nos nós folha processam as subtarefas.
  - ✓ Os restantes processos são responsáveis por criar as subtarefas e por agregar os seus resultados parciais.
- O padrão de comunicação é bem definido e bastante simples:
  - ✓ Como as subtarefas são totalmente independentes não é necessário qualquer tipo de comunicação durante o processamento das mesmas.
  - ✓ Apenas existe comunicação entre o processo que cria as subtarefas e os processos que as processam.

# Principais Paradigmas

## Divide and Conquer

- No entanto, o processo de divisão em tarefas e de agregação de resultados também pode ser realizado em paralelo, o que requer comunicação entre os processos:
  - ✓ As tarefas podem ser colocadas numa fila de tarefas única e centralizada ou podem ser distribuídas por diferentes filas de tarefas associadas à resolução de cada sub-problema.

# Principais Paradigmas

## Speculative Parallelism

- É utilizado quando as dependências entre os dados são tão complexas que tornam difícil explorar paralelismo usando os paradigmas anteriores.
- Este paradigma introduz paralelismo nos problemas através da execução de computações especulativas:
  - ✓ A ideia é antecipar a execução de computações relacionadas com a computação corrente na assumption otimista de que essas computações serão necessariamente realizadas posteriormente.
  - ✓ Quando isso não acontece, pode ser necessário repor partes do estado da computação de modo a não violar a consistência do problema em resolução.

# Principais Paradigmas

## Speculative Parallelism

- É utilizado quando as dependências entre os dados são tão complexas que tornam difícil explorar paralelismo usando os paradigmas anteriores.
- Este paradigma introduz paralelismo nos problemas através da execução de computações especulativas:
  - ✓ A ideia é antecipar a execução de computações relacionadas com a computação corrente na assumption otimista de que essas computações serão necessariamente realizadas posteriormente.
  - ✓ Quando isso não acontece, pode ser necessário repor partes do estado da computação de modo a não violar a consistência do problema em resolução.

# Principais Paradigmas

## Speculative Parallelism

- Uma outra aplicação deste paradigma é quando se utiliza simultaneamente diversos algoritmos para resolver um determinado problema e se escolhe aquele que primeiro obtiver uma solução.



# Principais Paradigmas

	Decomposição	Distribuição
<i>Master/Slave</i>	estática	dinâmica
<i>Single Program Multiple Data (SPMD)</i>	estática	estática/dinâmica
<i>Data Pipelining</i>	estática	estática
<i>Divide and Conquer</i>	dinâmica	dinâmica
<i>Speculative Parallelism</i>	dinâmica	dinâmica