

Advanced Topics in Data Mining and Logic Programming

Paralelismo em Programação Lógica

Ricardo Rocha
DCC-FCUP, University of Porto
ricroc@dcc.fc.up.pt

Concorrência ou Paralelismo Potencial

- Concorrência ou paralelismo potencial diz-se quando um programa possui **tarefas** (partes contíguas do programa) que podem ser executadas em qualquer ordem sem alterar o resultado final.

| | | | |
|-----------|--------------|----------------|------------|
| começar() | uma_tarefa() | outra_tarefa() | terminar() |
|-----------|--------------|----------------|------------|

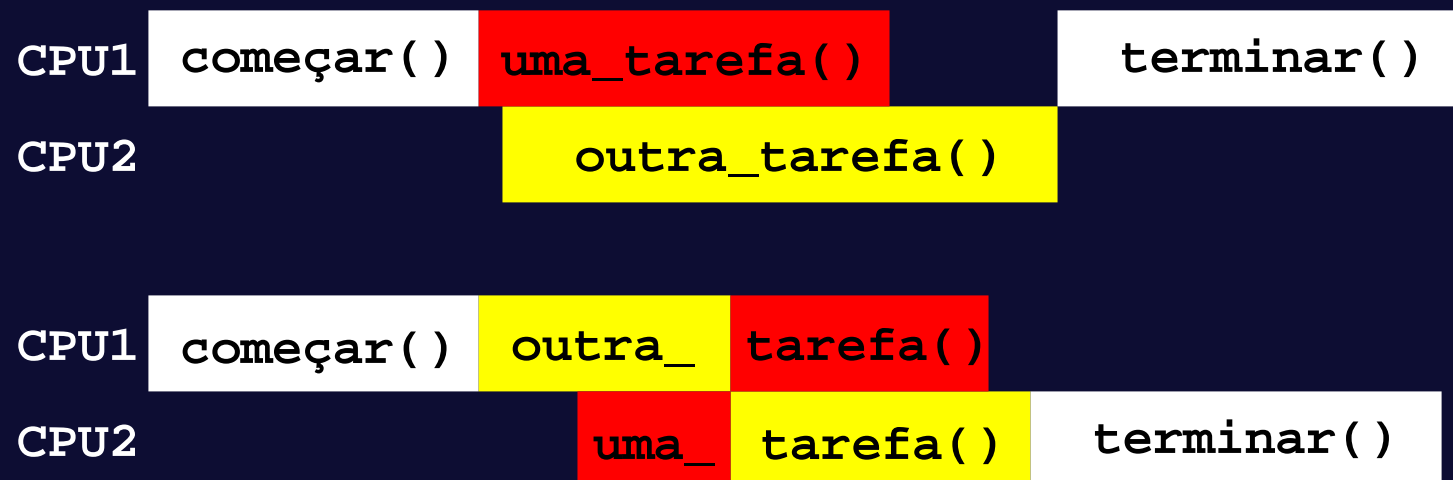
| | | | |
|-----------|----------------|--------------|------------|
| começar() | outra_tarefa() | uma_tarefa() | terminar() |
|-----------|----------------|--------------|------------|

| | | | | | |
|-----------|------|--------|----------|----------|------------|
| começar() | uma_ | outra_ | tarefa() | tarefa() | terminar() |
|-----------|------|--------|----------|----------|------------|

- Uma razão óbvia para explorar concorrência é conseguir reduzir o tempo de execução dos programas.

Paralelismo

- Paralelismo diz-se quando as tarefas concorrentes de um programa são executadas em simultâneo numa máquina com mais do que um processador.



Paralelismo Explícito

O paralelismo diz-se explícito quando cabe ao **programador**:

- Anotar as tarefas para execução em paralelo.
- Atribuir (possivelmente) as tarefas aos processadores.
- Controlar a execução, indicando nomeadamente os pontos de sincronização.
- Conhecer a arquitectura da máquina de forma a conseguir melhor performance (aumentar localidade, diminuir comunicação, etc.)

Paralelismo Explícito

O paralelismo diz-se explícito quando cabe ao **programador**:

- Anotar as tarefas para execução em paralelo.
- Atribuir (possivelmente) as tarefas aos processadores.
- Controlar a execução, indicando nomeadamente os pontos de sincronização.
- Conhecer a arquitectura da máquina de forma a conseguir melhor performance (aumentar localidade, diminuir comunicação, etc.)

Vantagens e inconvenientes:

- **(+)** Programadores experientes produzem soluções muito eficientes para problemas específicos.
- **(-)** O programador é o responsável por todos os detalhes da execução (*debugging* pode ser deveras penoso).
- **(-)** Pouco portátil entre diferentes arquitecturas.

Paralelismo Implícito

O paralelismo diz-se implícito quando cabe ao **compilador** e ao próprio **sistema de execução**:

- Detectar o paralelismo potencial do programa.
- Atribuir as tarefas para execução em paralelo.
- Controlar e sincronizar toda a execução.

Paralelismo Implícito

O paralelismo diz-se implícito quando cabe ao **compilador** e ao próprio **sistema de execução**:

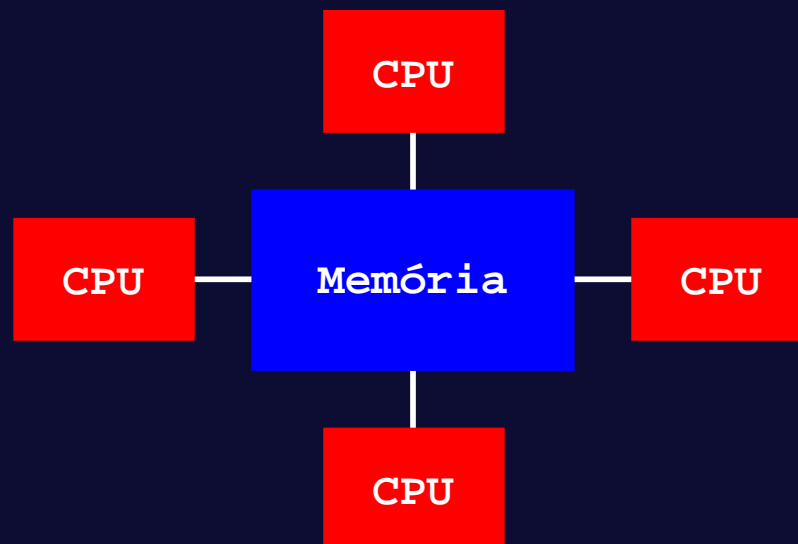
- Detectar o paralelismo potencial do programa.
- Atribuir as tarefas para execução em paralelo.
- Controlar e sincronizar toda a execução.

Vantagens e inconvenientes:

- **(+)** Liberta o programador dos detalhes da execução paralela.
- **(+)** Solução mais geral e mais flexível.
- **(-)** Difícil conseguir-se uma solução eficiente para todos os casos.

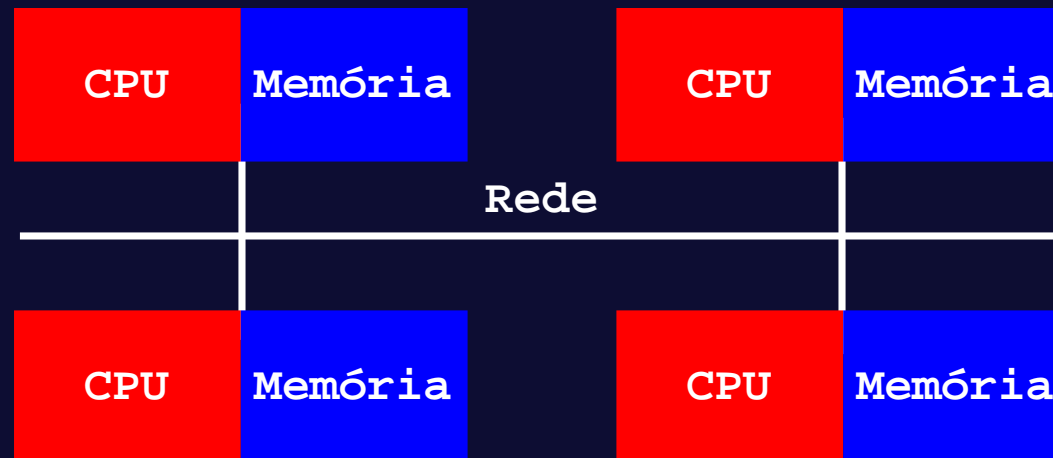
Computações Paralelas

- Uma computação diz-se paralela quando um programa é executado sobre uma máquina multiprocessador em que todos os processadores partilham o acesso à memória física disponível.
- ◆ Os processadores executam de forma independente mas o espaço de endereçamento global é partilhado.
- ◆ Qualquer alteração sobre uma posição de memória realizada por um determinado processador é igualmente visível por todos os restantes processadores.



Computações Distribuídas

- Uma computação diz-se distribuída quando um programa é executado sobre um conjunto de computadores ligados por rede em que cada computador tem acesso exclusivo à sua memória física.
- ◆ O espaço de endereçamento de cada computador não é partilhado pelos restantes computadores, ou seja, não existe o conceito de espaço de endereçamento global.
- ◆ As alterações sobre uma posição de memória realizada por um determinado processador não são visíveis pelos processadores dos restantes computadores.



Porquê Paralelismo em Prolog?

Porquê Paralelismo em Prolog?

➤ Eficiência das implementações sequenciais

- ◆ Os compiladores de Prolog baseados no modelo de execução sequencial WAM provaram ser altamente eficientes o que tornou o Prolog bastante popular.

Porquê Paralelismo em Prolog?

➤ Eficiência das implementações sequenciais

- ◆ Os compiladores de Prolog baseados no modelo de execução sequencial WAM provaram ser altamente eficientes o que tornou o Prolog bastante popular.

➤ Poder declarativo da linguagem

- ◆ Um programa é simplesmente um conjunto de cláusulas de predicados lógicos que podem ser vistos como especificações executáveis. As cláusulas são executadas pela ordem que são escritas e os objectivos de cada cláusula são executados da esquerda para a direita.

Porquê Paralelismo em Prolog?

➤ Eficiência das implementações sequenciais

- ◆ Os compiladores de Prolog baseados no modelo de execução sequencial WAM provaram ser altamente eficientes o que tornou o Prolog bastante popular.

➤ Poder declarativo da linguagem

- ◆ Um programa é simplesmente um conjunto de cláusulas de predicados lógicos que podem ser vistos como especificações executáveis. As cláusulas são executadas pela ordem que são escritas e os objectivos de cada cláusula são executados da esquerda para a direita.

➤ Potencial para a exploração implícita de paralelismo

- ◆ A execução paralela ocorre de modo automático por acção do próprio sistema de execução que detecta o paralelismo potencial do programa.
- ◆ Solução mais geral e mais flexível que liberta o programador dos detalhes da execução paralela e que torna a programação paralela tão simples como a sequencial.

Principais Formas de Paralelismo

➤ Paralelismo na Unificação

- ◆ Corresponde à unificação simultânea dos vários termos e/ou sub-termos de um objectivo.
- ◆ Aparece durante o processo de unificar os argumentos de um objectivo com os argumentos da cabeça de uma cláusula com o mesmo nome e aridade.
- ◆ Este tipo de paralelismo é de granularidade muito fina e não tem sido o principal foco de investigação em paralelismo em programação lógica.

Principais Formas de Paralelismo

➤ Paralelismo na Unificação

- ◆ Corresponde à unificação simultânea dos vários termos e/ou sub-termos de um objectivo.
- ◆ Aparece durante o processo de unificar os argumentos de um objectivo com os argumentos da cabeça de uma cláusula com o mesmo nome e aridade.
- ◆ Este tipo de paralelismo é de granularidade muito fina e não tem sido o principal foco de investigação em paralelismo em programação lógica.

➤ Paralelismo-Ou

- ◆ Corresponde à execução simultânea do corpo de diferentes cláusulas.
- ◆ Aparece quando mais do que uma cláusula define um predicado e uma chamada ao predicado unifica com a cabeça de mais do que uma cláusula.

```
a(X,Y) :- b(X), c(Y).  
a(X,Y) :- d(X,Y), e(Y).  
a(X,Y) :- f(X,Z), g(Z,Y).
```

Principais Formas de Paralelismo

➤ Paralelismo-E

- ◆ Corresponde à execução simultânea dos vários objectivos do corpo de uma cláusula.
- ◆ Aparece quando existe mais do que um objectivo presente no corpo de uma cláusula.

$a(X,Y) \text{ :- } b(X), c(Y).$

Principais Formas de Paralelismo

➤ Paralelismo-E

- ◆ Corresponde à execução simultânea dos vários objectivos do corpo de uma cláusula.
- ◆ Aparece quando existe mais do que um objectivo presente no corpo de uma cláusula.

$a(X,Y) :- b(X), c(Y).$

➤ Paralelismo-E Independente

- ◆ Ocorre quando dois ou mais objectivos do corpo duma cláusula não partilham variáveis por instanciar.
- ◆ A potencial instanciação das variáveis de um objectivo é sempre compatível com o resultado dos outros objectivos.

$a(X,Y) : - b(X), c(Y).$

Principais Formas de Paralelismo

➤ Paralelismo-E Dependente

- ◆ Ocorre quando dois ou mais objectivos no corpo de uma cláusula têm variáveis comuns por instanciar.
- ◆ A execução simultânea desses objectivos pode levar a instâncias incompatíveis das variáveis comuns.

$a(X,Y) : - d(X,Y), e(Y).$

$a(X,Y) : - f(X,Z), g(Z,Y).$

Principais Formas de Paralelismo

➤ Paralelismo-E Dependente

- ◆ Ocorre quando dois ou mais objectivos no corpo de uma cláusula têm variáveis comuns por instanciar.
- ◆ A execução simultânea desses objectivos pode levar a instâncias incompatíveis das variáveis comuns.

$a(X,Y) : - d(X,Y), e(Y).$

$a(X,Y) : - f(X,Z), g(Z,Y).$

- ◆ Uma solução é executar os objectivos independentemente até que ambos terminem. A compatibilidade das atribuições a variáveis comuns é verificada apenas no final (**back unification**). Esta solução é de certa forma semelhante ao Paralelismo-E Independente.

Principais Formas de Paralelismo

➤ Paralelismo-E Dependente

- ◆ Ocorre quando dois ou mais objectivos no corpo de uma cláusula têm variáveis comuns por instanciar.
- ◆ A execução simultânea desses objectivos pode levar a instâncias incompatíveis das variáveis comuns.

$a(X,Y) : - d(X,Y), e(Y).$

$a(X,Y) : - f(X,Z), g(Z,Y).$

- ◆ Uma solução é executar os objectivos independentemente até que ambos terminem. A compatibilidade das atribuições a variáveis comuns é verificada apenas no final (**back unification**). Esta solução é de certa forma semelhante ao Paralelismo-E Independente.
- ◆ Outra solução é executar os objectivos independentemente até que uma variável comum seja instanciada por um dos objectivos (designado por **produtor**). Os restantes objectivos (designados por **consumidores**) lêem a atribuição como sendo um argumento de entrada para a variável. A partir desse momento, sempre que o produtor encontrar uma nova atribuição para a variável comum, uma nova sequência pode ser iniciada por parte dos consumidores.

Sistemas Mais Conhecidos

➤ Paralelismo-Ou

- ◆ Aurora
- ◆ Muse

➤ Paralelismo-E Independente

- ◆ &-Prolog
- ◆ &ACE

➤ Paralelismo-E Dependente

- ◆ DASWAM
- ◆ ACE

Princípio da Ortogonalidade

- Intuitivamente, como cada uma destas formas de paralelismo explora diferentes pontos de não-determinismo da semântica operacional da resolução SLD, deveria ser possível explorar todas elas simultaneamente num único sistema.
- Um tal sistema deverá verificar o **princípio da ortogonalidade**, ou seja, cada forma de paralelismo deverá ser explorada sem afectar a exploração de outra.

```
function sld_resolution(Query) {  
    select_literal(Query, B);           // Paralelismo-E  
    do {  
        select_clause(Program, (Head :- Body)); // Paralelismo-Ou  
        if (mgu = most_general_unifier(B, Head)) { // Paralelismo na unificação  
            Query = assign(mgu, Query - B + Body);  
            if (Query) sld_resolution(Query);  
            else SUCCESS;  
        }  
    } until (no_clauses_left);  
    FAIL;  
}
```

Princípio da Ortogonalidade

- Todavia, a experiência mostra que o princípio da ortogonalidade é de difícil concretização numa implementação real já que, até à data, nenhum sistema eficiente foi ainda construído que explore simultaneamente todas as formas de paralelismo. A aproximação com maior sucesso foi o sistema **Andorra-I** para a exploração simultânea de Paralelismo-Ou e Paralelismo-E Dependente.
- Um sistema que explore o máximo de paralelismo dos programas lógicos enquanto mantém a melhor performance possível, é o objectivo fundamental dos investigadores em paralelismo em programação lógica.

Paralelismo-Ou: Motivação

Das formas de paralelismo, o Paralelismo-Ou é aquele que tem maior sucesso. A razão fundamental é que este parece ser mais fácil e mais produtivo de explorar.

Paralelismo-Ou: Motivação

Das formas de paralelismo, o Paralelismo-Ou é aquele que tem maior sucesso. A razão fundamental é que este parece ser mais fácil e mais produtivo de explorar.

- **Simplicidade:** como as cláusulas de um mesmo predicado são independentes, é possível explorar Paralelismo-Ou sem solicitar qualquer anotação extra ao programador e sem necessitar de análises complexas em tempo de compilação.

Paralelismo-Ou: Motivação

Das formas de paralelismo, o Paralelismo-Ou é aquele que tem maior sucesso. A razão fundamental é que este parece ser mais fácil e mais produtivo de explorar.

- **Simplicidade:** como as cláusulas de um mesmo predicado são independentes, é possível explorar Paralelismo-Ou sem solicitar qualquer anotação extra ao programador e sem necessitar de análises complexas em tempo de compilação.
- **Proximidade do modelo sequencial:** é possível explorar Paralelismo-Ou num modelo de execução bastante próximo do modelo sequencial. Isto significa que se torna mais fácil manter a semântica da linguagem, e que podemos tirar total vantagem da tecnologia existente de implementação sequencial para diminuir o custo do modelo paralelo sobre o modelo sequencial.

Paralelismo-Ou: Motivação

Das formas de paralelismo, o Paralelismo-Ou é aquele que tem maior sucesso. A razão fundamental é que este parece ser mais fácil e mais produtivo de explorar.

- **Simplicidade:** como as cláusulas de um mesmo predicado são independentes, é possível explorar Paralelismo-Ou sem solicitar qualquer anotação extra ao programador e sem necessitar de análises complexas em tempo de compilação.
- **Proximidade do modelo sequencial:** é possível explorar Paralelismo-Ou num modelo de execução bastante próximo do modelo sequencial. Isto significa que se torna mais fácil manter a semântica da linguagem, e que podemos tirar total vantagem da tecnologia existente de implementação sequencial para diminuir o custo do modelo paralelo sobre o modelo sequencial.
- **Granularidade:** para uma ampla classe de programas, o Paralelismo-Ou apresenta um potencial de paralelismo de granularidade grossa. A granularidade de uma computação paralela diz respeito ao tamanho das tarefas que podem ser executadas sem interagir com as restantes tarefas processadas em simultâneo.

Paralelismo-Ou: Modelo Básico de Execução

- Inicialmente, todos os agentes encontram-se no estado de **suspensos**. Após a invocação de uma nova questão, um dos agentes inicia a execução.

Paralelismo-Ou: Modelo Básico de Execução

- Inicialmente, todos os agentes encontram-se no estado de **suspensos**. Após a invocação de uma nova questão, um dos agentes inicia a execução.
- Sempre que um agente executa uma chamada que unifica com mais do que uma cláusula, ele cria um ponto de escolha **privado**. Pontos de escolha privados indicam a presença de trabalho disponível para ser executado em paralelo.

Paralelismo-Ou: Modelo Básico de Execução

- Inicialmente, todos os agentes encontram-se no estado de **suspensos**. Após a invocação de uma nova questão, um dos agentes inicia a execução.
- Sempre que um agente executa uma chamada que unifica com mais do que uma cláusula, ele cria um ponto de escolha **privado**. Pontos de escolha privados indicam a presença de trabalho disponível para ser executado em paralelo.
- Logo que um agente suspenso detecta que existe trabalho disponível, ele solicita directamente a um dos agentes **ocupados** a partilha desse trabalho por forma a cooperar na sua exploração.

Paralelismo-Ou: Modelo Básico de Execução

- Inicialmente, todos os agentes encontram-se no estado de **suspensos**. Após a invocação de uma nova questão, um dos agentes inicia a execução.
- Sempre que um agente executa uma chamada que unifica com mais do que uma cláusula, ele cria um ponto de escolha **privado**. Pontos de escolha privados indicam a presença de trabalho disponível para ser executado em paralelo.
- Logo que um agente suspenso detecta que existe trabalho disponível, ele solicita directamente a um dos agentes **ocupados** a partilha desse trabalho por forma a cooperar na sua exploração.
- Se um agente ocupado decide partilhar parte do seu trabalho, então ele torna **públicos** os pontos de escolha que pretende partilhar antes de sincronizar o seu estado com o agente suspenso.

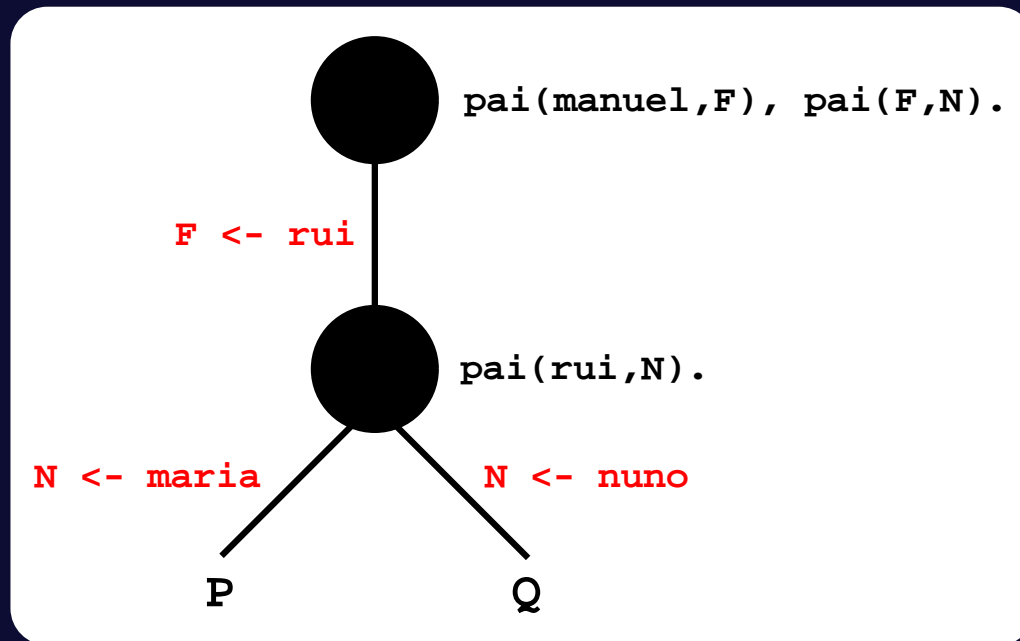
Paralelismo-Ou: Modelo Básico de Execução

- Inicialmente, todos os agentes encontram-se no estado de **suspensos**. Após a invocação de uma nova questão, um dos agentes inicia a execução.
- Sempre que um agente executa uma chamada que unifica com mais do que uma cláusula, ele cria um ponto de escolha **privado**. Pontos de escolha privados indiciam a presença de trabalho disponível para ser executado em paralelo.
- Logo que um agente suspenso detecta que existe trabalho disponível, ele solicita directamente a um dos agentes **ocupados** a partilha desse trabalho por forma a cooperar na sua exploração.
- Se um agente ocupado decide partilhar parte do seu trabalho, então ele torna **públicos** os pontos de escolha que pretende partilhar antes de sincronizar o seu estado com o agente suspenso.
- Assim que um agente explore todo o seu trabalho, este regressa ao estado de suspenso e reinicia a procura por novo trabalho disponível. O processo termina quando todo o trabalho tiver sido explorado e todos os agentes se encontrarem suspensos.

Paralelismo-Ou: Principais Problemas a Resolver

➤ Múltiplos Ambientes

- ◆ **Problema:** como representar e aceder eficientemente aos múltiplos ambientes que coexistem simultaneamente?

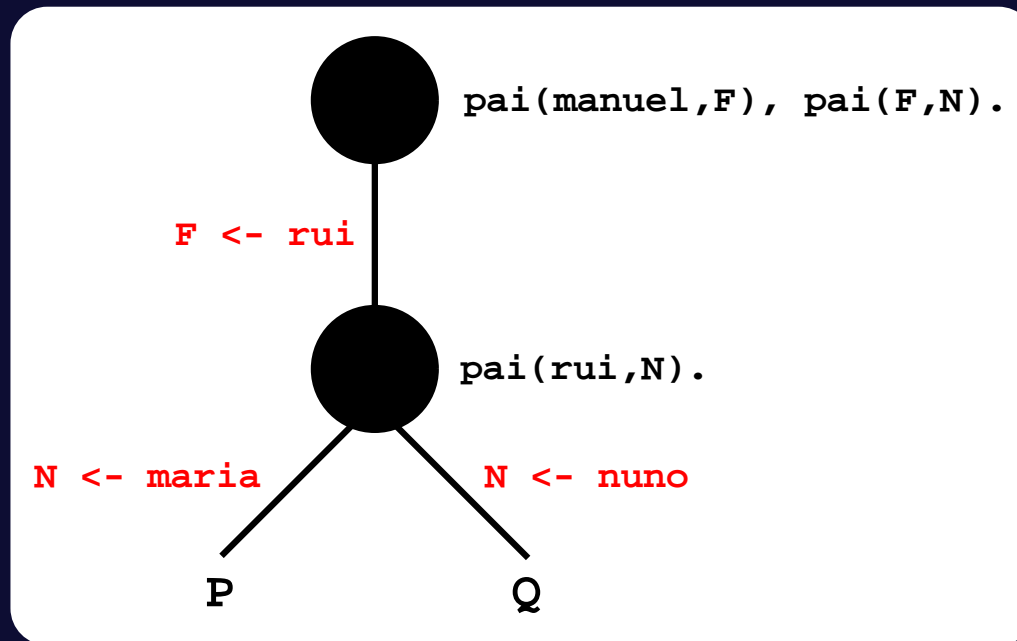


```
pai(manuel, rui).  
pai(pedro, joaquim).  
pai(rui, maria).  
pai(rui, nuno).  
pai(maria, alberto).
```

Paralelismo-Ou: Principais Problemas a Resolver

➤ Múltiplos Ambientes

- ◆ **Problema:** como representar e aceder eficientemente aos múltiplos ambientes que coexistem simultaneamente?



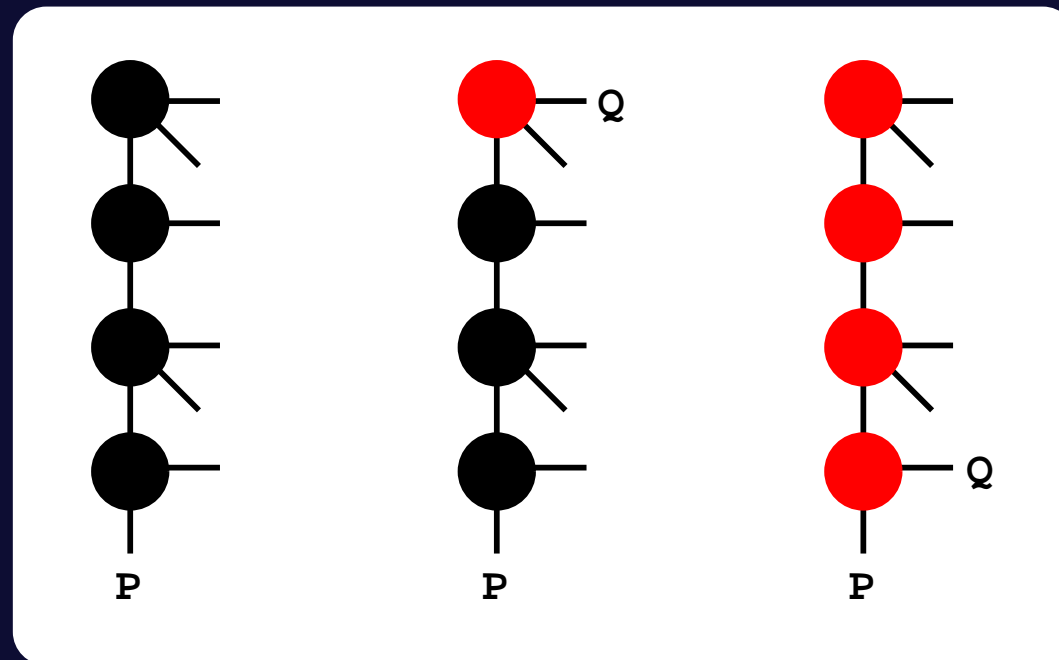
```
pai(manuel, rui).  
pai(pedro, joaquim).  
pai(rui, maria).  
pai(rui, nuno).  
pai(maria, alberto).
```

- ◆ **Solução:** utilizar mecanismos onde cada ambiente tem uma área privada onde guarda as suas atribuições **condicionais**. Uma atribuição a uma variável é condicional se a variável foi criada antes do ponto de escolha corrente.

Paralelismo-Ou: Principais Problemas a Resolver

➤ Distribuição de Trabalho (Scheduling)

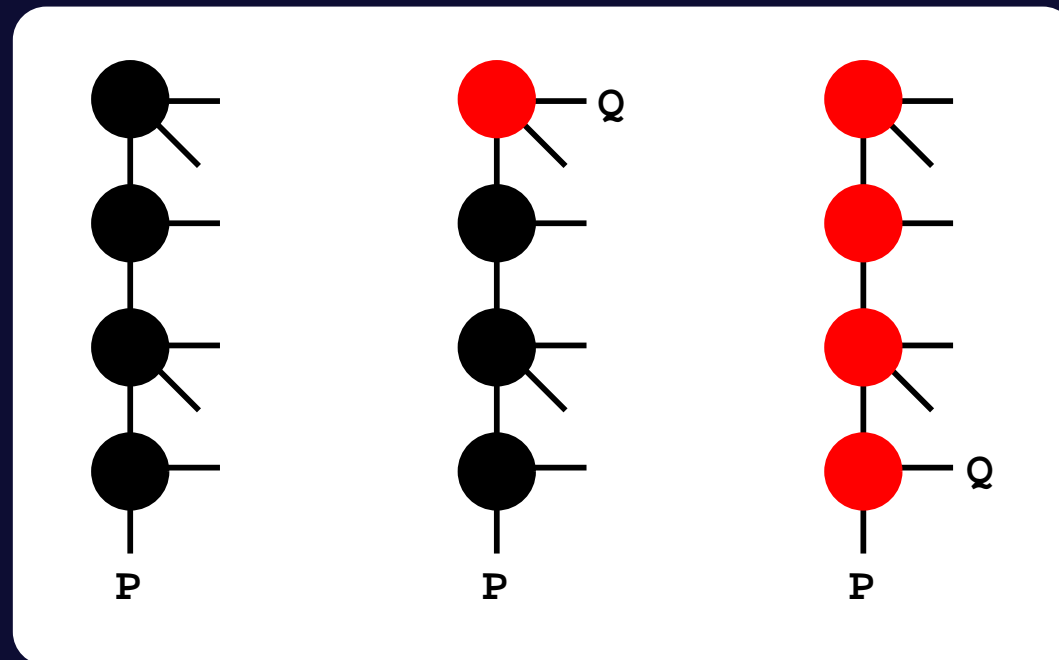
- ◆ **Problema:** como distribuir eficientemente o trabalho existente pelos agentes disponíveis de modo a minimizar o tempo total de execução?



Paralelismo-Ou: Principais Problemas a Resolver

➤ Distribuição de Trabalho (Scheduling)

- ◆ **Problema:** como distribuir eficientemente o trabalho existente pelos agentes disponíveis de modo a minimizar o tempo total de execução?



- ◆ **Solução:** existem duas políticas principais: distribuir trabalho o mais acima possível e distribuir trabalho o mais abaixo possível.

Paralelismo-Ou: Principais Modelos de Execução

Memória Partilhada:

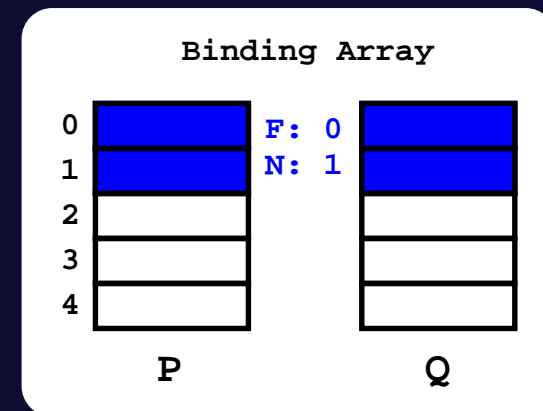
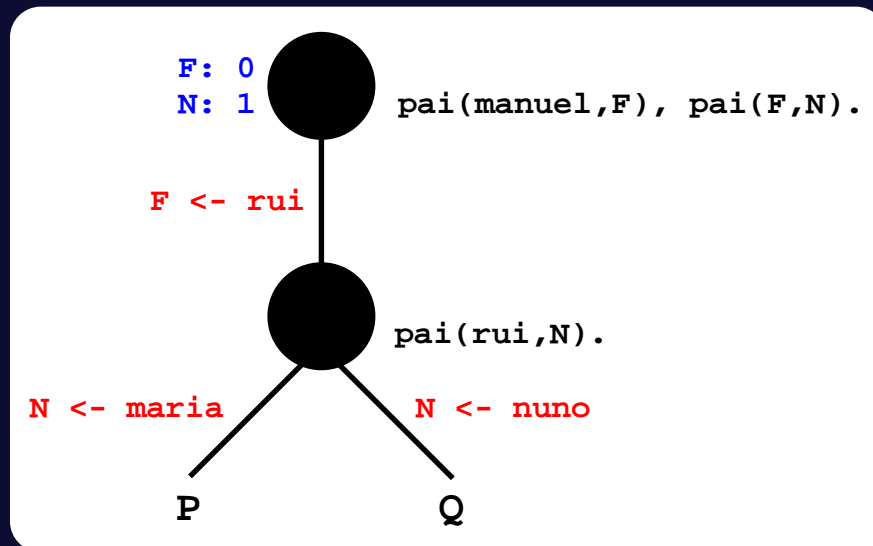
- **Binding Arrays**
- **Cópia de Ambientes**

Memória Distribuída:

- **Stack Splitting**

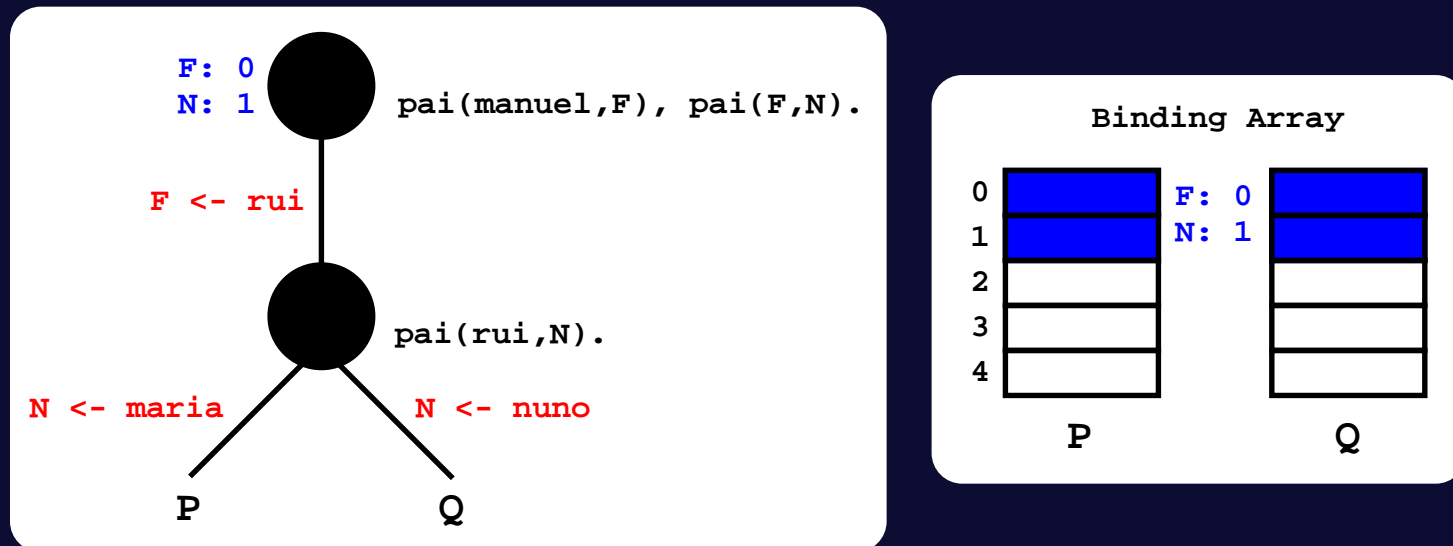
Binding Arrays

- Neste modelo, cada agente possui uma estrutura de dados auxiliar designada por **binding array**.



Binding Arrays

- Neste modelo, cada agente possui uma estrutura de dados auxiliar designada por **binding array**.



- Sempre que uma nova variável é criada esta é etiquetada com um **valor único** que identifica a sua posição no binding array. Isto é conseguido através de um **contador**. As variáveis são etiquetadas com o valor corrente do contador e em seguida este é incrementado. A numeração das variáveis ao longo de uma qualquer ramificação forma uma sequência estritamente crescente.

Binding Arrays

- Por forma a permitir que, sempre que uma nova alternativa for tomada num ponto de escolha, a sequência possa continuar, o valor do contador é guardado nos pontos de escolha.

Binding Arrays

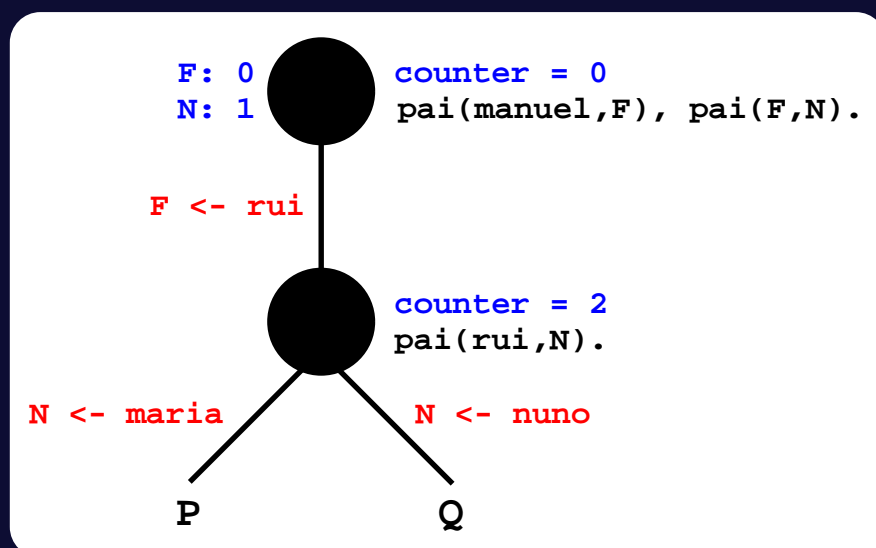
- Por forma a permitir que, sempre que uma nova alternativa for tomada num ponto de escolha, a sequência possa continuar, o valor do contador é guardado nos pontos de escolha.
- Uma atribuição a uma variável é **condicional** se o valor da sua etiqueta é menor que o valor do contador guardado no ponto de escolha corrente.

Binding Arrays

- Por forma a permitir que, sempre que uma nova alternativa for tomada num ponto de escolha, a sequência possa continuar, o valor do contador é guardado nos pontos de escolha.
- Uma atribuição a uma variável é **condicional** se o valor da sua etiqueta é menor que o valor do contador guardado no ponto de escolha corrente.
- O problema da coexistência de múltiplos ambientes é resolvido guardando as atribuições condicionais no binding array do próprio agente.

Binding Arrays

- Por forma a permitir que, sempre que uma nova alternativa for tomada num ponto de escolha, a sequência possa continuar, o valor do contador é guardado nos pontos de escolha.
- Uma atribuição a uma variável é **condicional** se o valor da sua etiqueta é menor que o valor do contador guardado no ponto de escolha corrente.
- O problema da coexistência de múltiplos ambientes é resolvido guardando as atribuições condicionais no binding array do próprio agente.

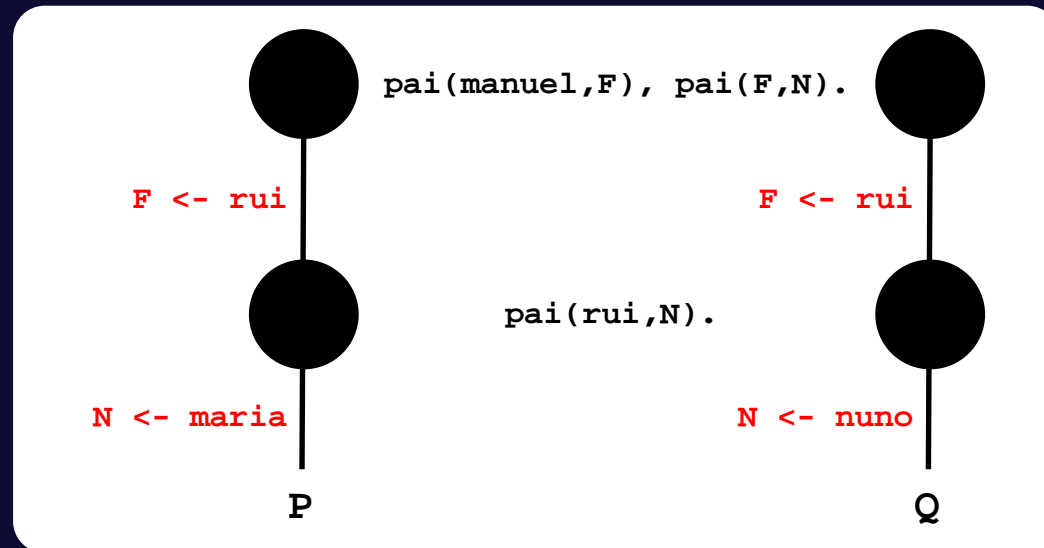


Binding Array

| | | |
|---|-------|------|
| 0 | - | - |
| 1 | maria | nuno |
| 2 | | |
| 3 | | |
| 4 | | |
| | P | Q |

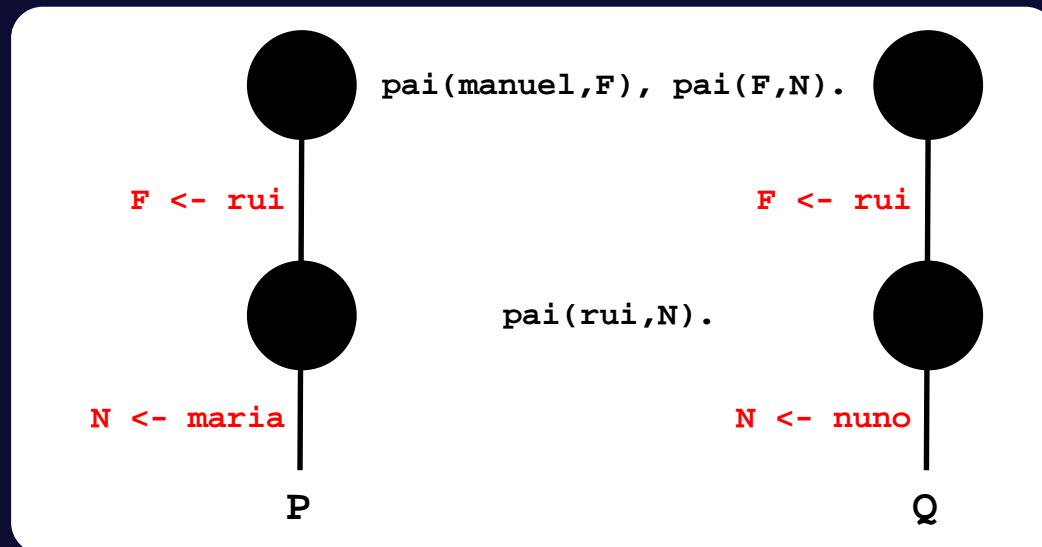
Cópia de Ambientes

- Neste modelo, cada agente mantém uma cópia separada do seu ambiente, no qual pode escrever sem causar conflito de atribuições.



Cópia de Ambientes

- Neste modelo, cada agente mantém uma cópia separada do seu ambiente, no qual pode escrever sem causar conflito de atribuições.



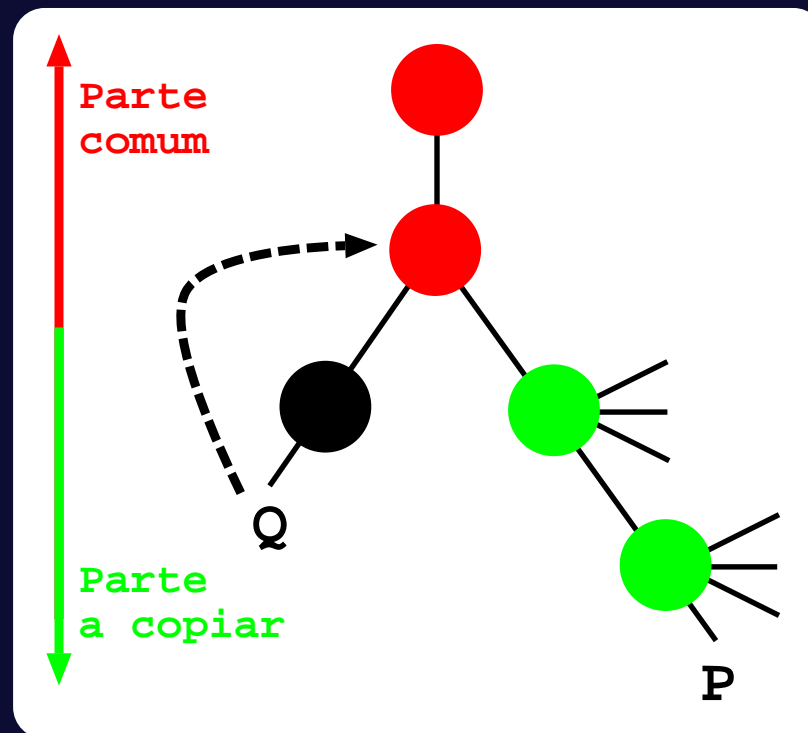
- A partilha de trabalho é feita por **cópia das pilhas de execução** (pilha local, pilha de termos e trilha) do agente ocupado para o agente suspenso.
- Para maximizar a quantidade de trabalho partilhado e evitar que os agentes partilhem trabalho frequentemente (diminuindo assim o potencial número de operações de cópia), o **trabalho é distribuído o mais abaixo possível**.

Cópia de Ambientes: Cópia Incremental

- **Problema:** a cópia de grandes quantidades de segmentos de memória pode degradar a performance do sistema.

Cópia de Ambientes: Cópia Incremental

- **Problema:** a cópia de grandes quantidades de segmentos de memória pode degradar a performance do sistema.
- **Solução:** utilizar **cópia incremental**. A ideia fundamental da cópia incremental é baseada no facto de que parte das pilhas de execução pode ser comum a ambos os agentes, e por conseguinte não necessitam de ser copiadas.

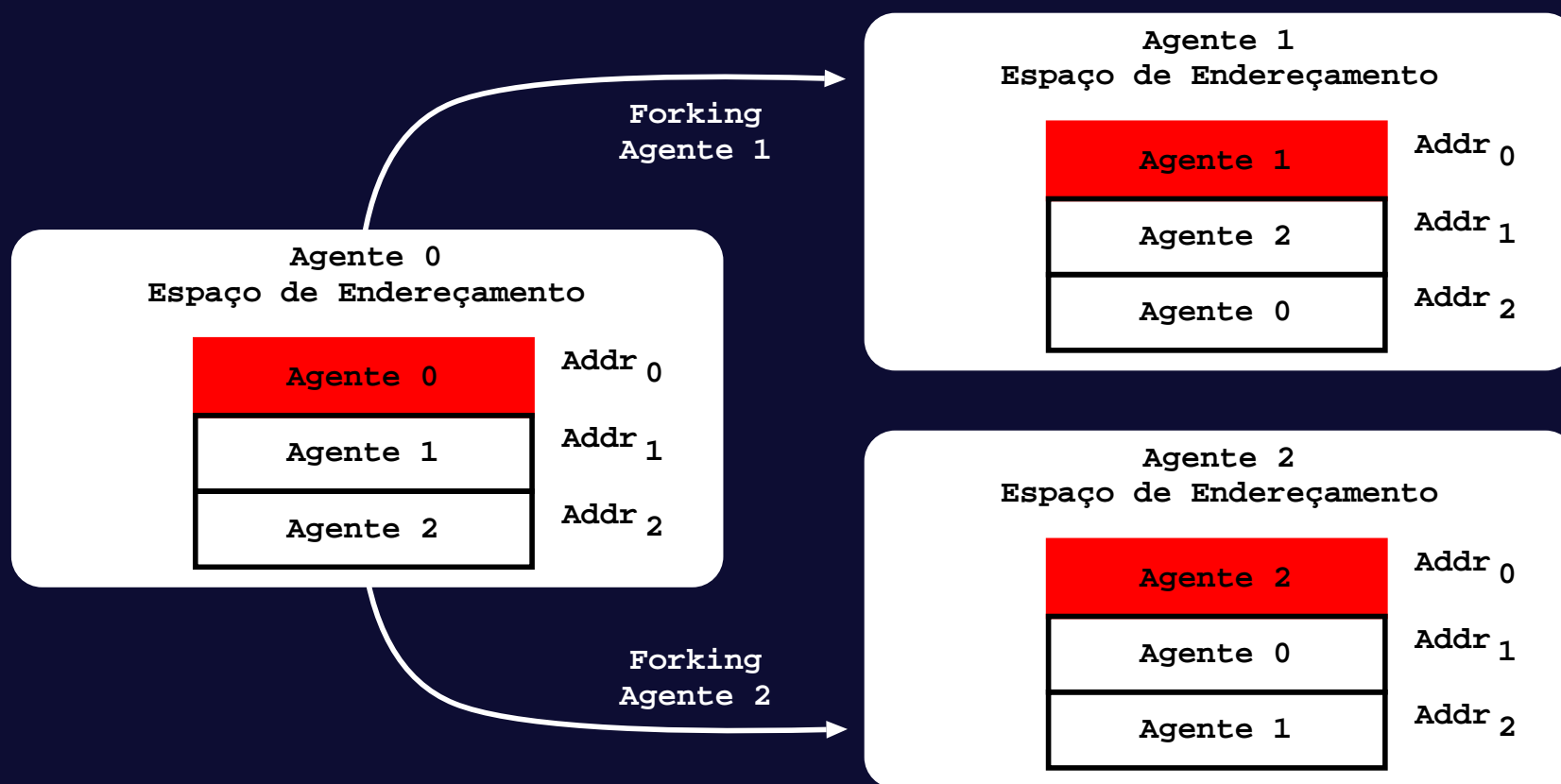


Cópia de Ambientes: Mapeamento de Memória

- **Problema:** a cópia de segmentos de memória entre agentes requer realocação de endereços de modo a fazerem sentido no novo espaço de endereçamento.

Cópia de Ambientes: Mapeamento de Memória

- **Problema:** a cópia de segmentos de memória entre agentes requer realocação de endereços de modo a fazerem sentido no novo espaço de endereçamento.
- **Solução:** mapear a memória de tal modo que todos os agentes vejam as suas próprias áreas no mesmo endereço, ou seja, o espaço de endereçamento de cada agente, dum ponto de vista individual, tem início no mesmo endereço.

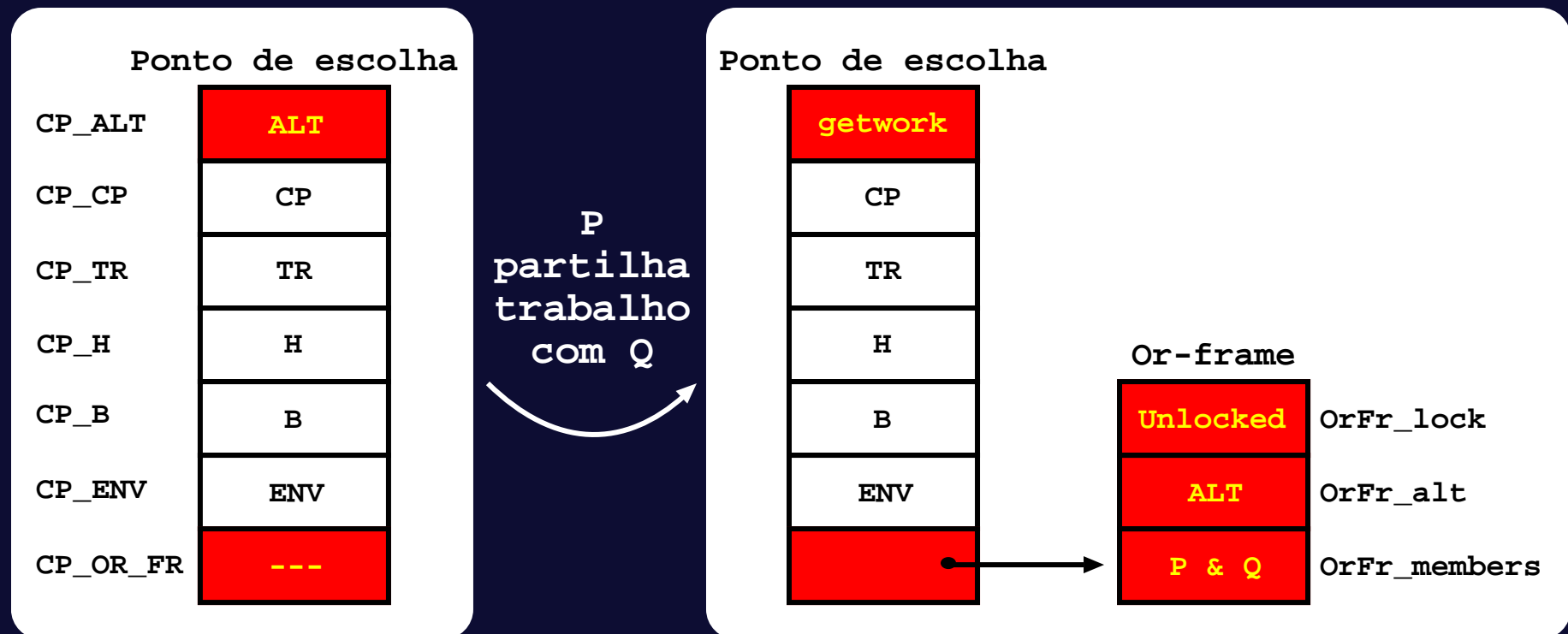


Cópia de Ambientes: Sincronização

- **Problema:** como sincronizar o acesso aos pontos de escolha públicos de modo a garantir que cada alternativa é tomada uma e uma só vez?

Cópia de Ambientes: Sincronização

- **Problema:** como sincronizar o acesso aos pontos de escolha públicos de modo a garantir que cada alternativa é tomada uma e uma só vez?
- **Solução:** guardar as alternativas numa estrutura partilhada (**or-frame**) e usar uma pseudo-instrução (**getwork**) para sincronizar o acesso à estrutura.

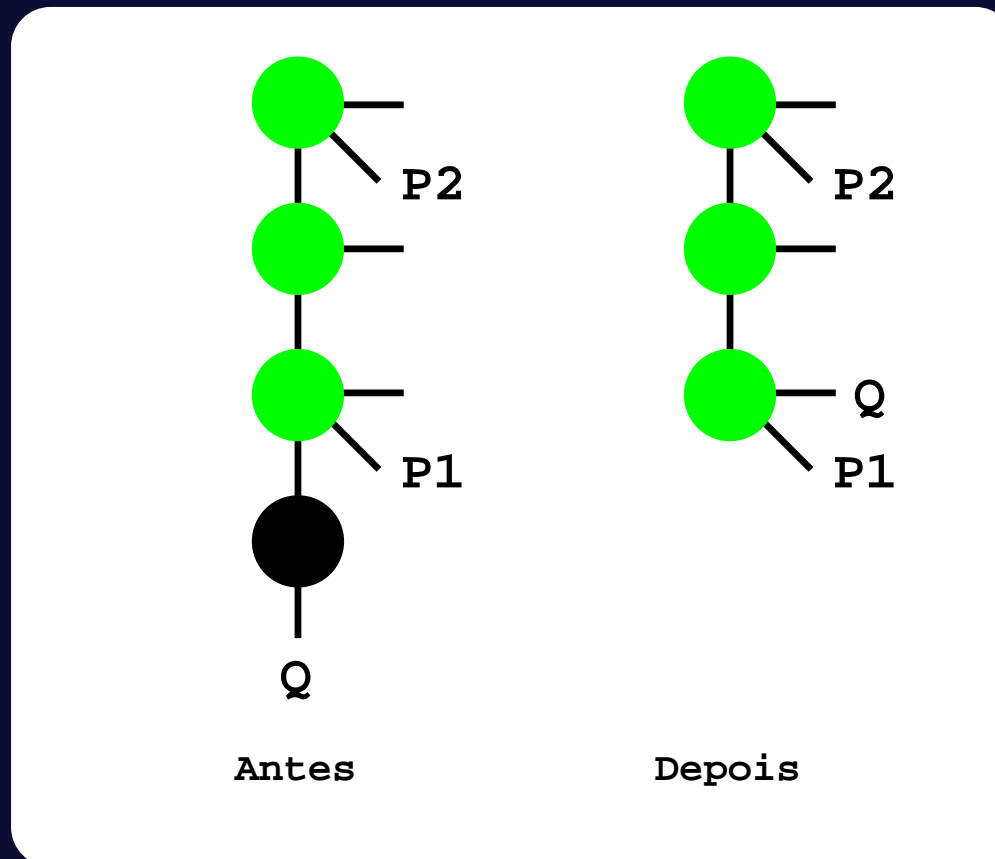


Cópia de Ambientes: Distribuição de Trabalho

- O principal objectivo do **distribuidor de trabalho** é minimizar o tempo total de execução mantendo correcta a semântica da linguagem.
- A intervenção do distribuidor de trabalho verifica-se sempre que um agente completa uma **tarefa**. Uma tarefa é uma porção contínua de trabalho executada na região privada da árvore de procura.
- A ideia básica do algoritmo do distribuidor de trabalho pode resumir-se ao seguinte:
 - ◆ Procurar trabalho na região partilhada.
 - ◆ Procurar trabalho em agentes ocupados.
 - ◆ Procurar um melhor posicionamento na árvore de procura.

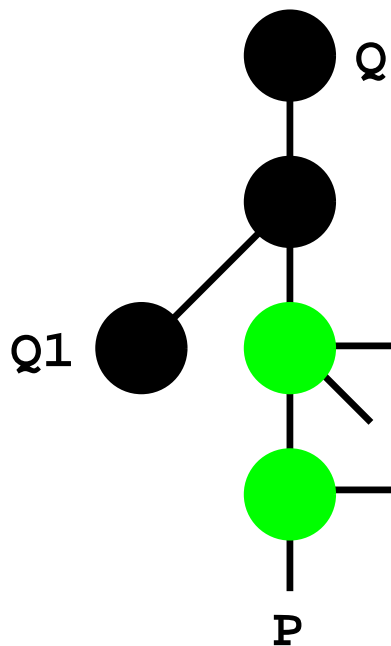
Cópia de Ambientes: Distribuição de Trabalho

- **Procurar trabalho na região partilhada:** tomar a alternativa por explorar mais próxima da posição actual do agente na árvore de procura.

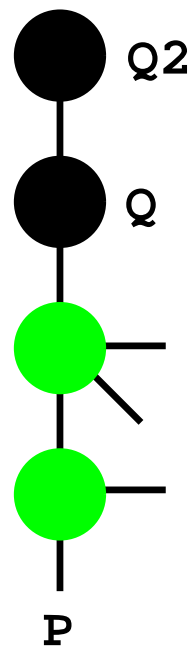


Cópia de Ambientes: Distribuição de Trabalho

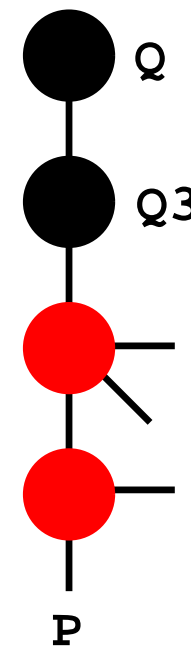
- **Procurar trabalho na subárvore do nó corrente:** solicitar trabalho ao agente ocupado que se encontre mais perto e que possua maior carga.



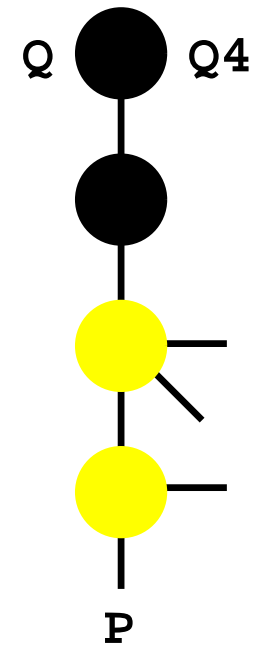
Q solicita
trabalho a P



Q solicita
trabalho a P



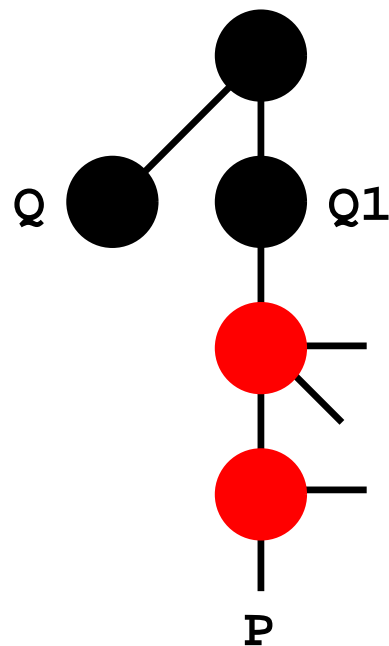
Q não solicita
trabalho a P



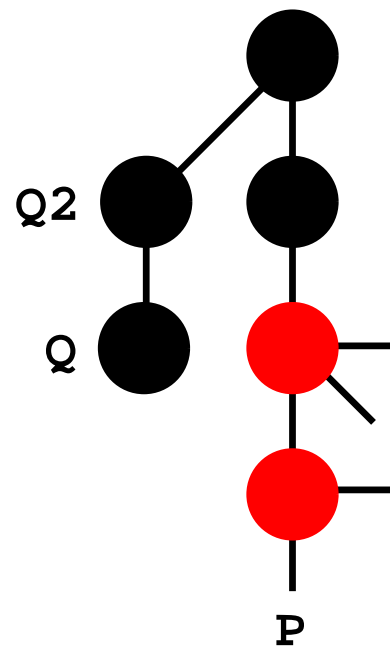
Q pode solicitar
trabalho a P

Cópia de Ambientes: Distribuição de Trabalho

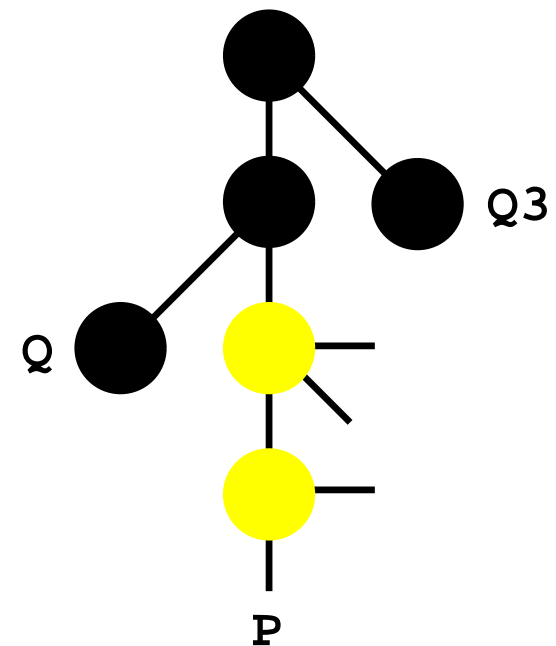
- **Procurar trabalho fora da subárvore do nó corrente:** solicitar trabalho ao agente ocupado que se encontre mais perto e que possua maior carga.



Q não solicita
trabalho a P



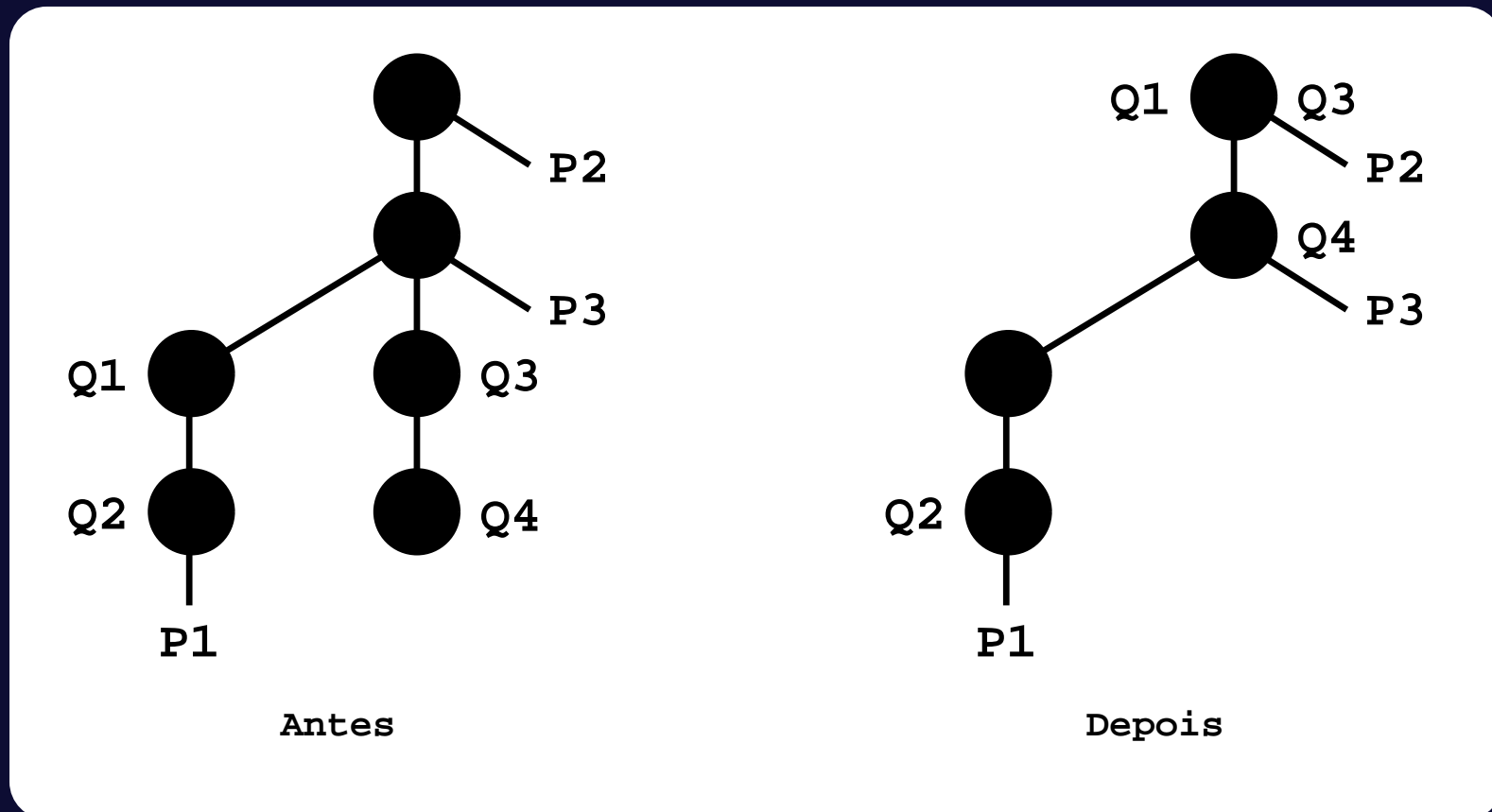
Q não solicita
trabalho a P



Q pode solicitar
trabalho a P

Cópia de Ambientes: Distribuição de Trabalho

- **Procurar um melhor posicionamento na árvore de procura:** retroceder para posições da árvore de procura que possuam agentes ocupados.



Stack Splitting

- **Problema:** utilizar mecanismos de exclusão mútua para sincronizar o acesso às or-frames em ambientes distribuídos pode levar a uma grande quantidade de troca de mensagens.

Stack Splitting

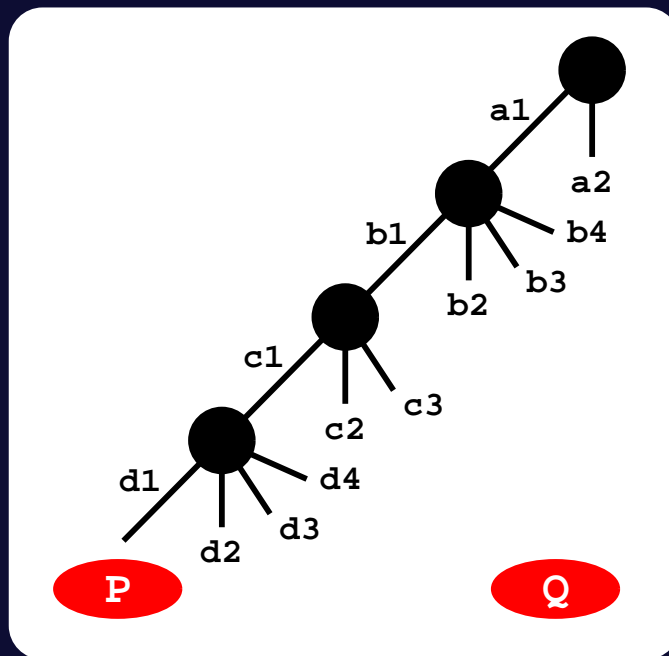
- **Problema:** utilizar mecanismos de exclusão mútua para sincronizar o acesso às or-frames em ambientes distribuídos pode levar a uma grande quantidade de troca de mensagens.
- **Solução:** dividir *à priori* as alternativas disponíveis pelos agentes envolvidos no processo de partilha de modo que cada agente fique desde logo com as suas alternativas. Esta técnica é designada por **stack splitting**.

Stack Splitting

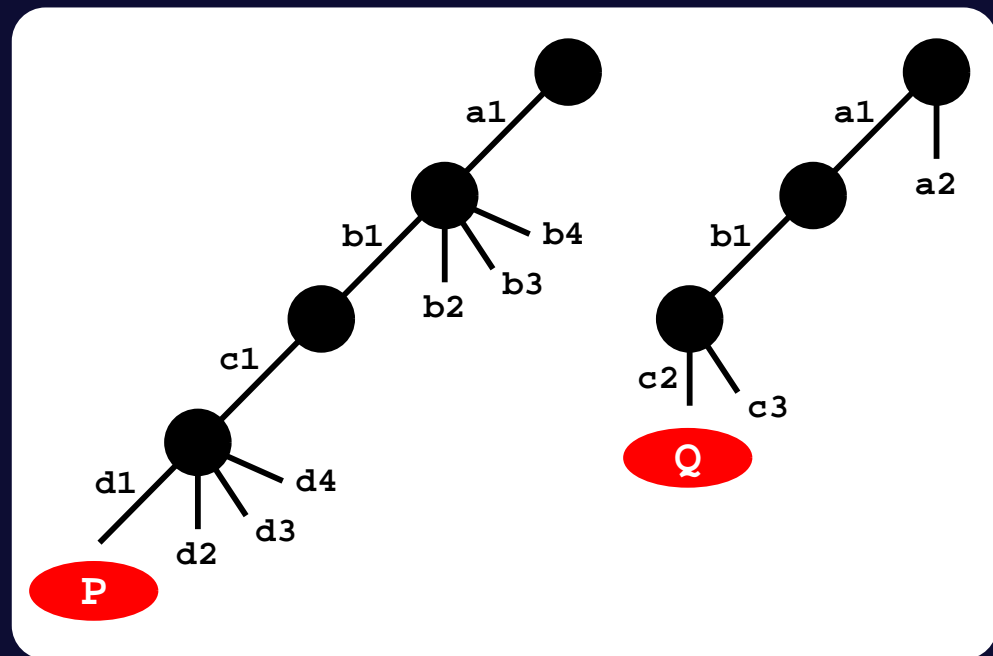
- **Problema:** utilizar mecanismos de exclusão mútua para sincronizar o acesso às or-frames em ambientes distribuídos pode levar a uma grande quantidade de troca de mensagens.
- **Solução:** dividir *à priori* as alternativas disponíveis pelos agentes envolvidos no processo de partilha de modo que cada agente fique desde logo com as suas alternativas. Esta técnica é designada por **stack splitting**.
- Neste modelo, a partilha de trabalho é igualmente feita por cópia das pilhas de execução acrescida duma fase posterior de **divisão de trabalho**.
- Existem três esquemas principais de divisão de trabalho:
 - ◆ Vertical Splitting
 - ◆ Horizontal Splitting
 - ◆ Diagonal Splitting

Stack Splitting: Vertical Splitting

- Os pontos de escolha são alternadamente divididos entre os dois agentes, começando pelo agente P que partilha trabalho.



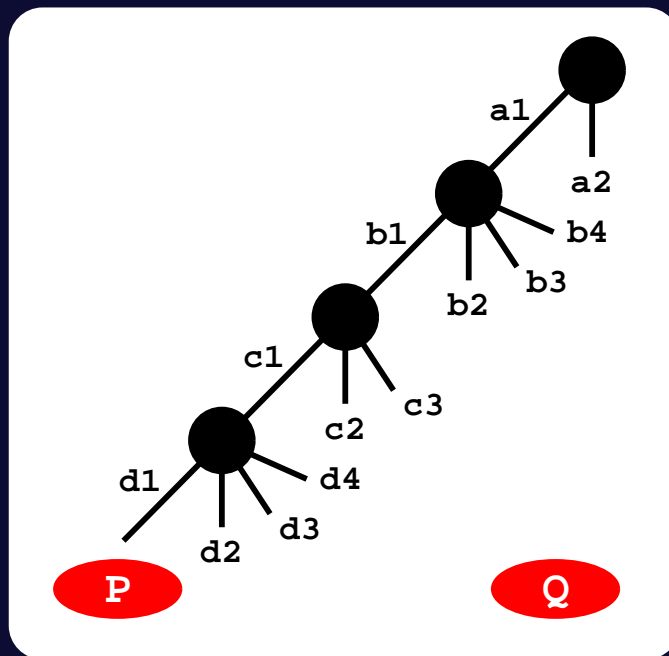
Antes



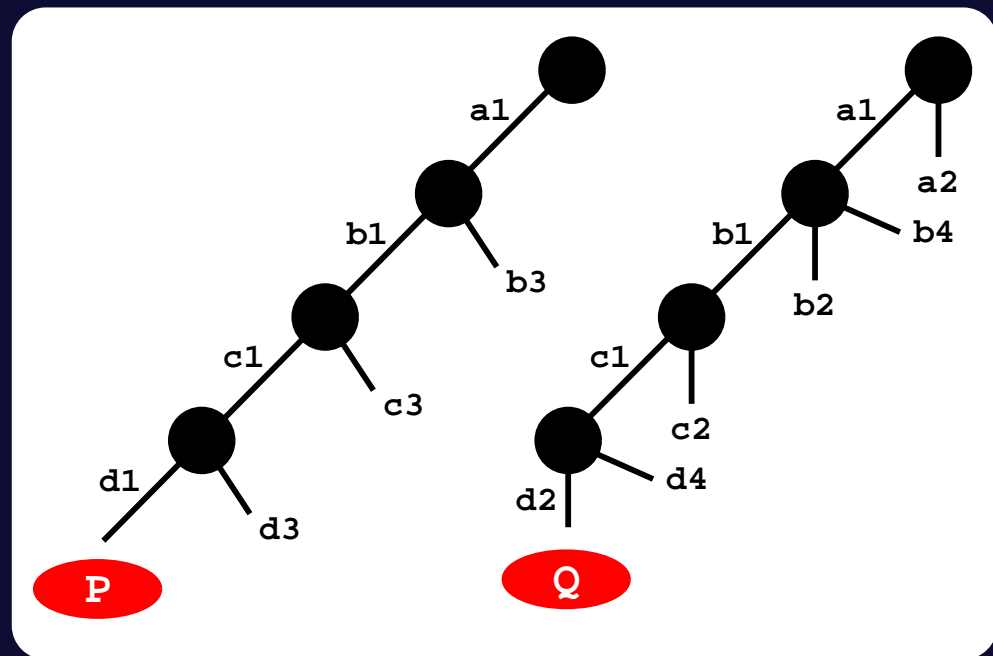
Depois

Stack Splitting: Horizontal Splitting

- As alternativas por explorar em cada ponto de escolha são alternadamente divididas entre o agente Q que pede trabalho e o agente P que partilha trabalho.



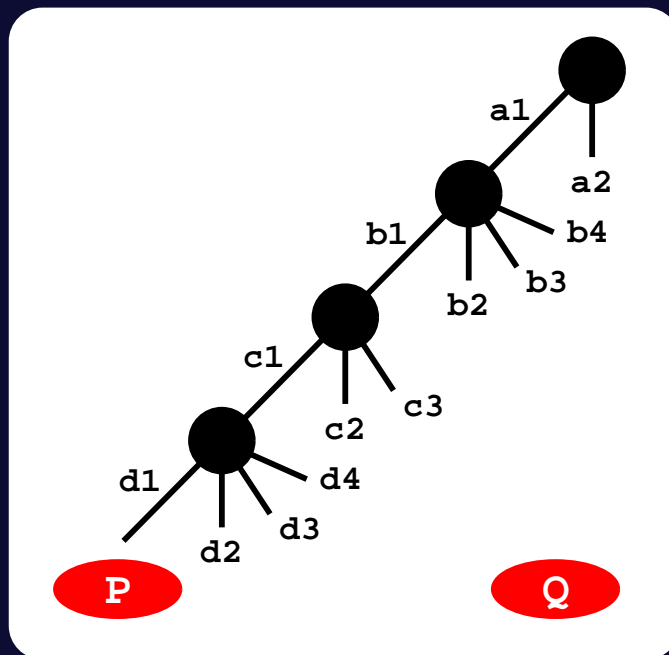
Antes



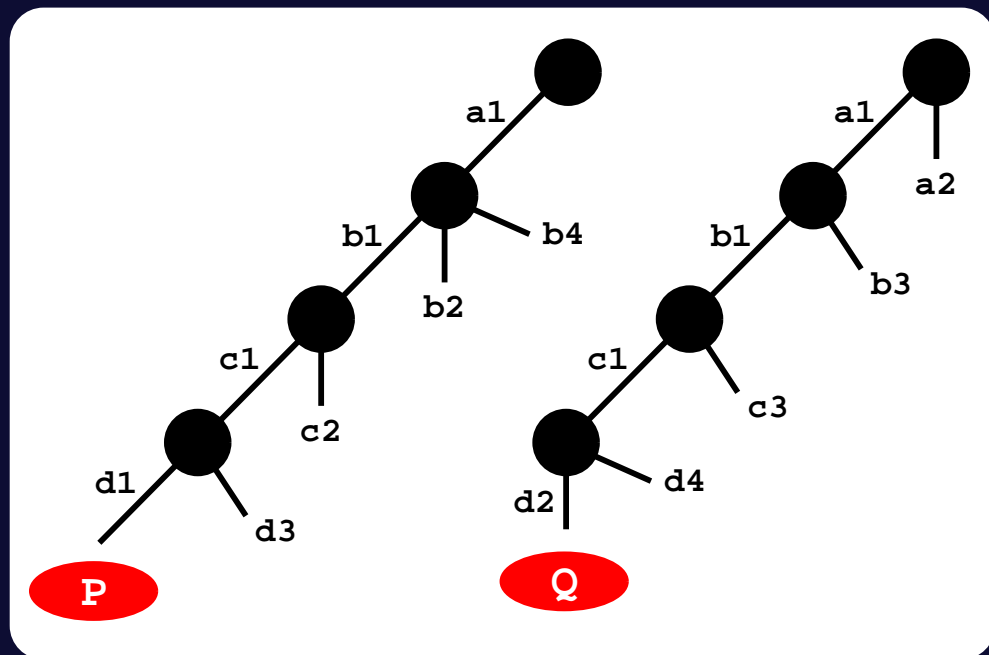
Depois

Stack Splitting: Diagonal Splitting

- O conjunto de todas as alternativas por explorar, independentemente dos pontos de escolha a que pertencem, é alternadamente dividido entre o agente Q que pede trabalho e o agente P que partilha trabalho.
- Este esquema é como que uma mistura dos dois anteriores e é o único que garante uma divisão equitativa das alternativas por explorar.



Antes



Depois

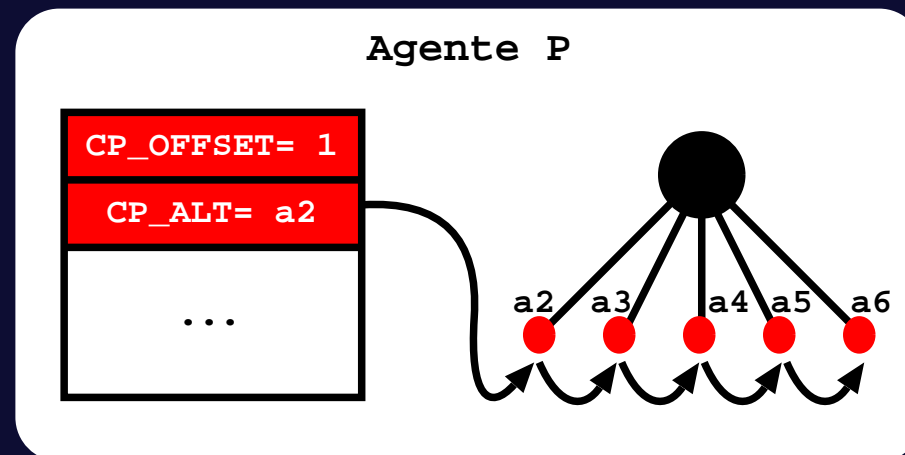
Stack Splitting: Como Dividir?

Stack Splitting: Como Dividir?

- Introduzir um novo campo nos pontos de escolha (**CP_OFFSET**) para indicar o deslocamento entre as alternativas que lhe pertencem.

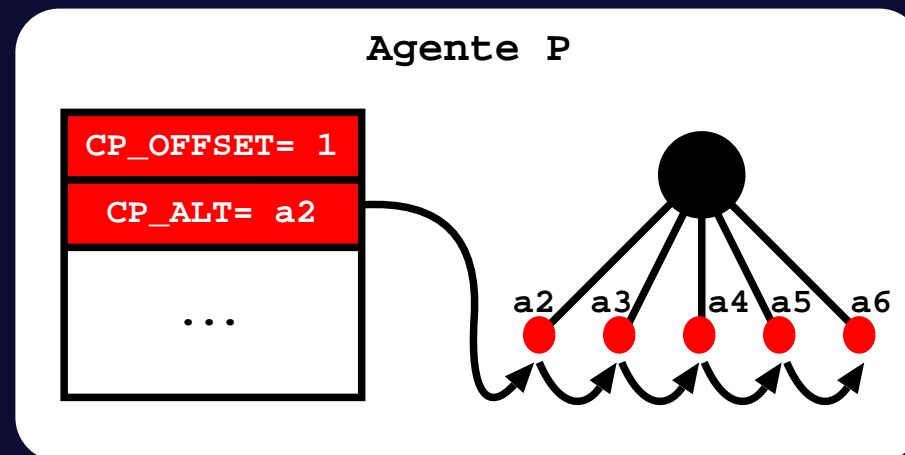
Stack Splitting: Como Dividir?

- Introduzir um novo campo nos pontos de escolha (**CP_OFFSET**) para indicar o deslocamento entre as alternativas que lhe pertencem.
- Para os pontos de escolha privados o valor do deslocamento é sempre 1.



Stack Splitting: Como Dividir?

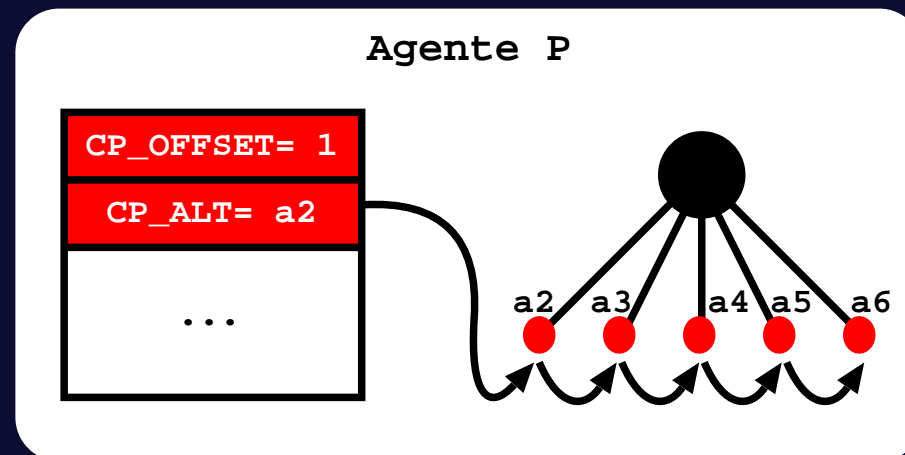
- Introduzir um novo campo nos pontos de escolha (**CP_OFFSET**) para indicar o deslocamento entre as alternativas que lhe pertencem.
- Para os pontos de escolha privados o valor do deslocamento é sempre 1.



- Quando se partilha um ponto de escolha duplica-se o valor do deslocamento.

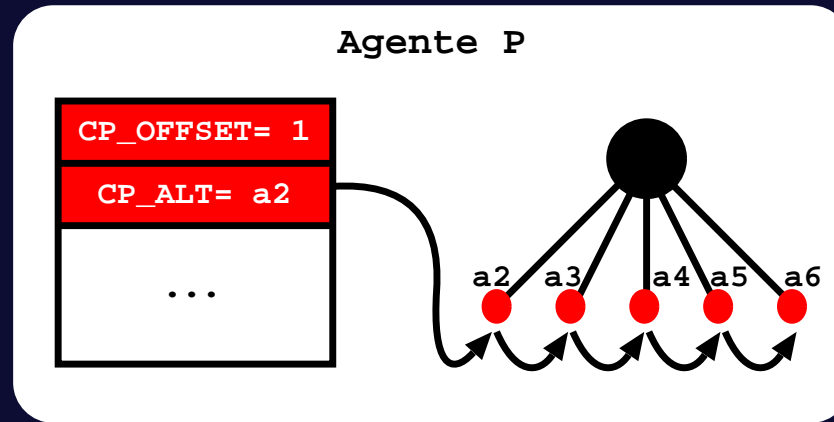
Stack Splitting: Como Dividir?

- Introduzir um novo campo nos pontos de escolha (**CP_OFFSET**) para indicar o deslocamento entre as alternativas que lhe pertencem.
- Para os pontos de escolha privados o valor do deslocamento é sempre 1.

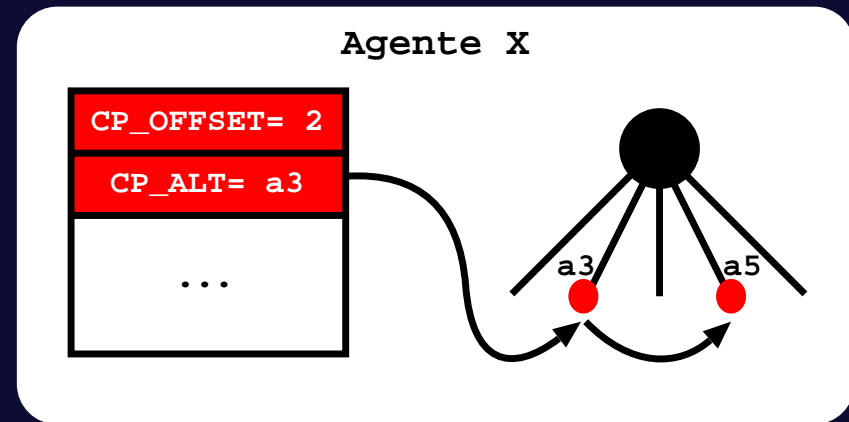
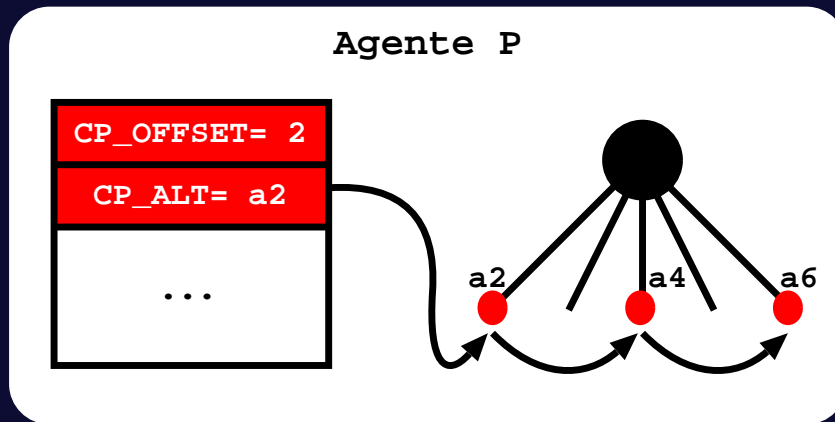


- Quando se partilha um ponto de escolha duplica-se o valor do deslocamento.
- O agente que não inicia o processo de divisão num dado ponto de escolha deve actualizar a referência para a próxima alternativa por explorar (campo **CP_ALT**).

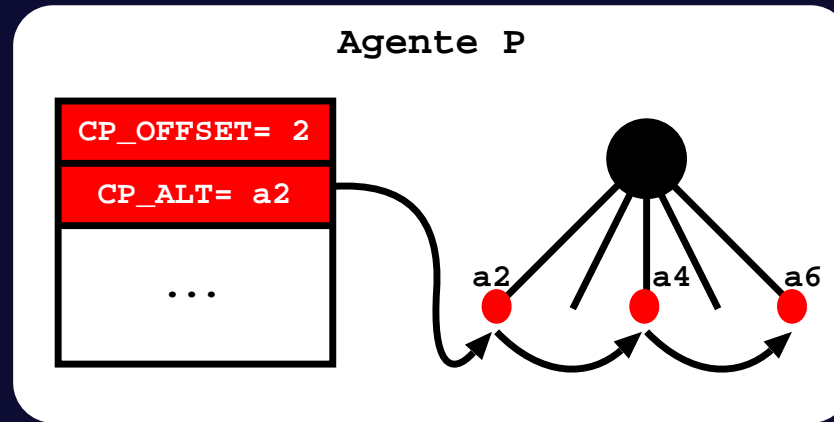
Stack Splitting: Como Dividir?



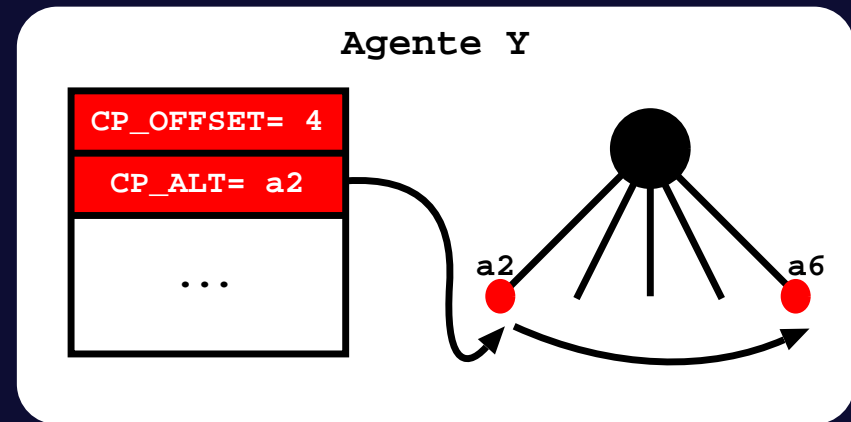
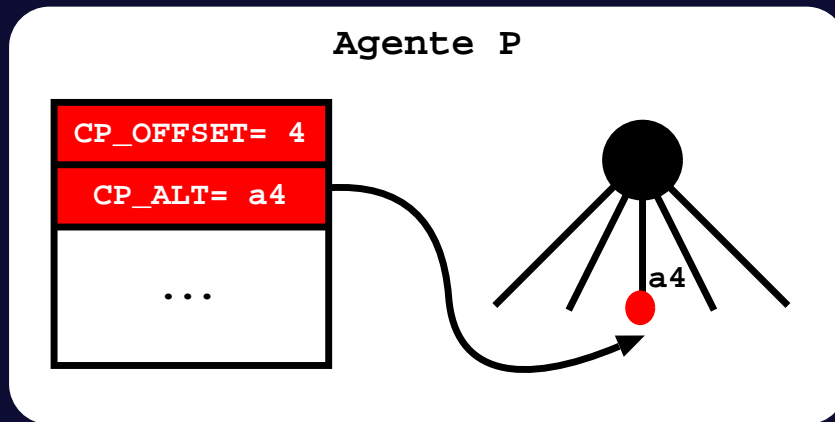
P partilha trabalho com X



Stack Splitting: Como Dividir?



P partilha trabalho com Y



Diagonal Splitting: Como Dividir?

- No esquema de Diagonal Splitting é necessário saber se o número de alternativas por explorar num ponto de escolha é par ou ímpar de modo a decidir qual o agente que inicia o processo de divisão no ponto de escolha seguinte.

Diagonal Splitting: Como Dividir?

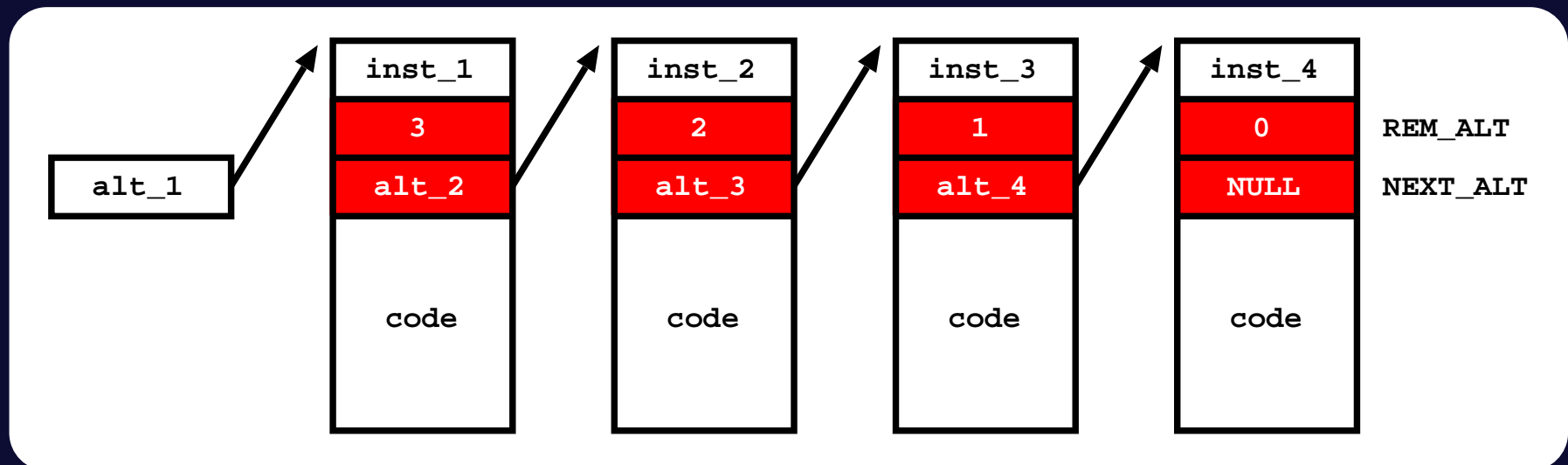
- No esquema de Diagonal Splitting é necessário saber se o número de alternativas por explorar num ponto de escolha é par ou ímpar de modo a decidir qual o agente que inicia o processo de divisão no ponto de escolha seguinte.
- Uma possibilidade é percorrer a lista de alternativas por explorar e contar o seu número.

Diagonal Splitting: Como Dividir?

- No esquema de Diagonal Splitting é necessário saber se o número de alternativas por explorar num ponto de escolha é par ou ímpar de modo a decidir qual o agente que inicia o processo de divisão no ponto de escolha seguinte.
- Uma possibilidade é percorrer a lista de alternativas por explorar e contar o seu número.
- Um esquema mais eficiente é incluir durante o processo de compilação informação relativa ao número de alternativas restantes a partir da alternativa corrente.

Diagonal Splitting: Como Dividir?

- No esquema de Diagonal Splitting é necessário saber se o número de alternativas por explorar num ponto de escolha é par ou ímpar de modo a decidir qual o agente que inicia o processo de divisão no ponto de escolha seguinte.
- Uma possibilidade é percorrer a lista de alternativas por explorar e contar o seu número.
- Um esquema mais eficiente é incluir durante o processo de compilação informação relativa ao número de alternativas restantes a partir da alternativa corrente.

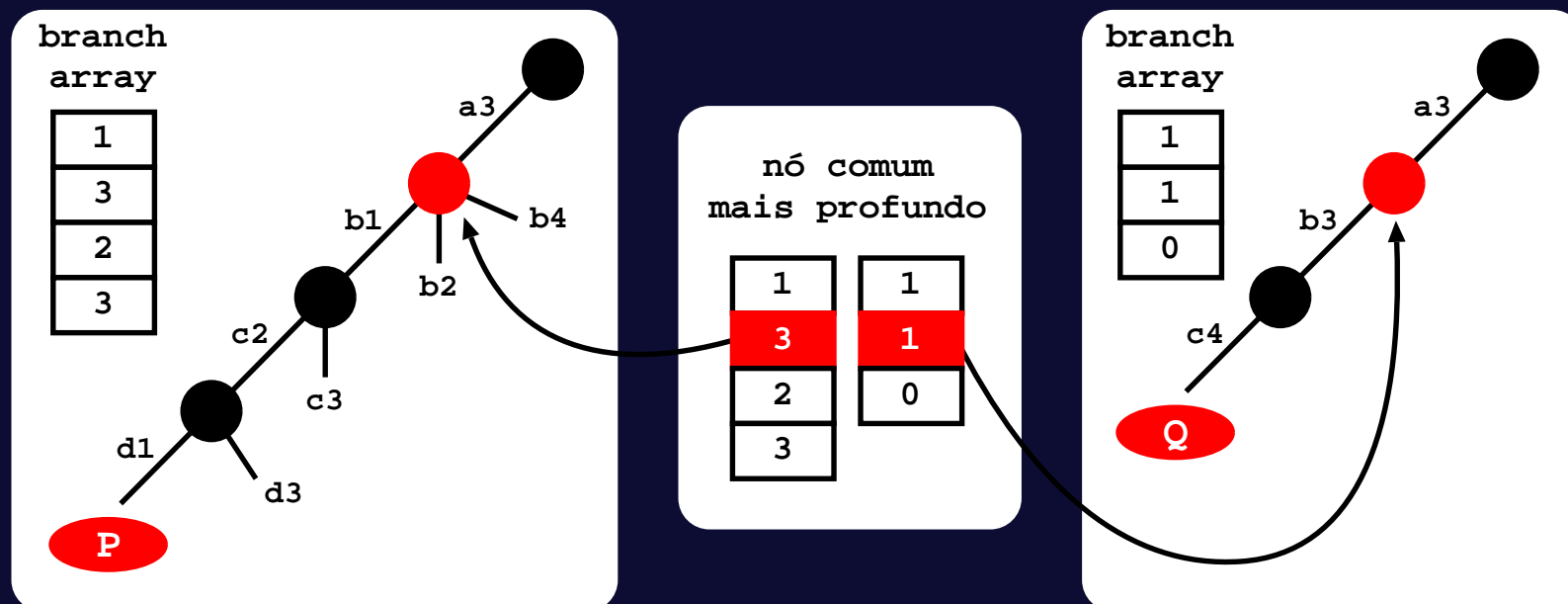


Diagonal Splitting: Branch Array

- **Problema:** para suportar cópia incremental é necessário um mecanismo que permita encontrar facilmente o **nó comum mais profundo** entre dois agentes.

Diagonal Splitting: Branch Array

- **Problema:** para suportar cópia incremental é necessário um mecanismo que permita encontrar facilmente o **nó comum mais profundo** entre dois agentes.
- **Solução:** para cada agente, utilizar um **branch array** que represente de modo único a posição do agente na árvore de procura. A profundidade de cada ponto de escolha identifica o deslocamento no branch array.



Diagonal Splitting: Partilhar Trabalho

- Q pede trabalho a P
 - ◆ Q envia uma mensagem a P que inclui o seu branch array
- P decide partilhar trabalho com Q
 - ◆ P calcula o nó comum mais profundo
 - ◆ P calcula os segmentos das pilhas de execução a copiar para Q
 - ◆ P empacota toda a informação numa mensagem e envia-a para Q
- Q recebe uma resposta positiva
 - ◆ Q copia os segmentos na mensagem para os locais apropriados das suas pilhas de execução
- P e Q dividem trabalho
 - ◆ P aplica o algoritmo de diagonal splitting
 - ◆ Q aplica o algoritmo de diagonal splitting

Os Nossos Protótipos

➤ YapOr

Extensão do Yap Prolog para a execução paralela de Prolog. Baseia-se no modelo de cópia de ambientes para explorar Paralelismo-Ou de forma implícita.

➤ YapDss

Extensão do Yap Prolog para a execução distribuída de Prolog. Baseia-se no modelo de diagonal stack splitting para explorar Paralelismo-Ou de forma implícita.

YapOr: Alguns Resultados

| Programa | 2 CPUs | 4 CPUs | 6 CPUs | 8 CPUs |
|------------------|--------------|--------------|--------------|--------------|
| puzzle | 4.835 (2.08) | 2.316 (4.34) | 1.550 (6.48) | 1.172 (8.57) |
| 9-queens | 2.047 (2.00) | 1.026 (3.98) | 0.690 (5.92) | 0.519 (7.87) |
| ham | 0.908 (1.98) | 0.474 (3.80) | 0.324 (5.56) | 0.245 (7.36) |
| 5cubes | 0.516 (1.99) | 0.260 (3.96) | 0.181 (5.69) | 0.145 (7.10) |
| 8-queens2 | 0.606 (1.75) | 0.288 (3.69) | 0.202 (5.26) | 0.149 (7.13) |
| 8-queens1 | 0.225 (2.00) | 0.118 (3.81) | 0.080 (5.63) | 0.067 (6.72) |
| nsort | 1.191 (1.75) | 0.609 (3.43) | 0.411 (5.08) | 0.315 (6.63) |
| sm*10 | 0.274 (1.92) | 0.158 (3.34) | 0.128 (4.12) | 0.115 (4.58) |
| db5*10 | 0.099 (1.69) | 0.065 (2.57) | 0.068 (2.46) | 0.061 (2.74) |
| db4*10 | 0.079 (1.68) | 0.056 (2.38) | 0.055 (2.42) | 0.060 (2.22) |
| Média | (1.88) | (3.53) | (4.86) | (6.09) |

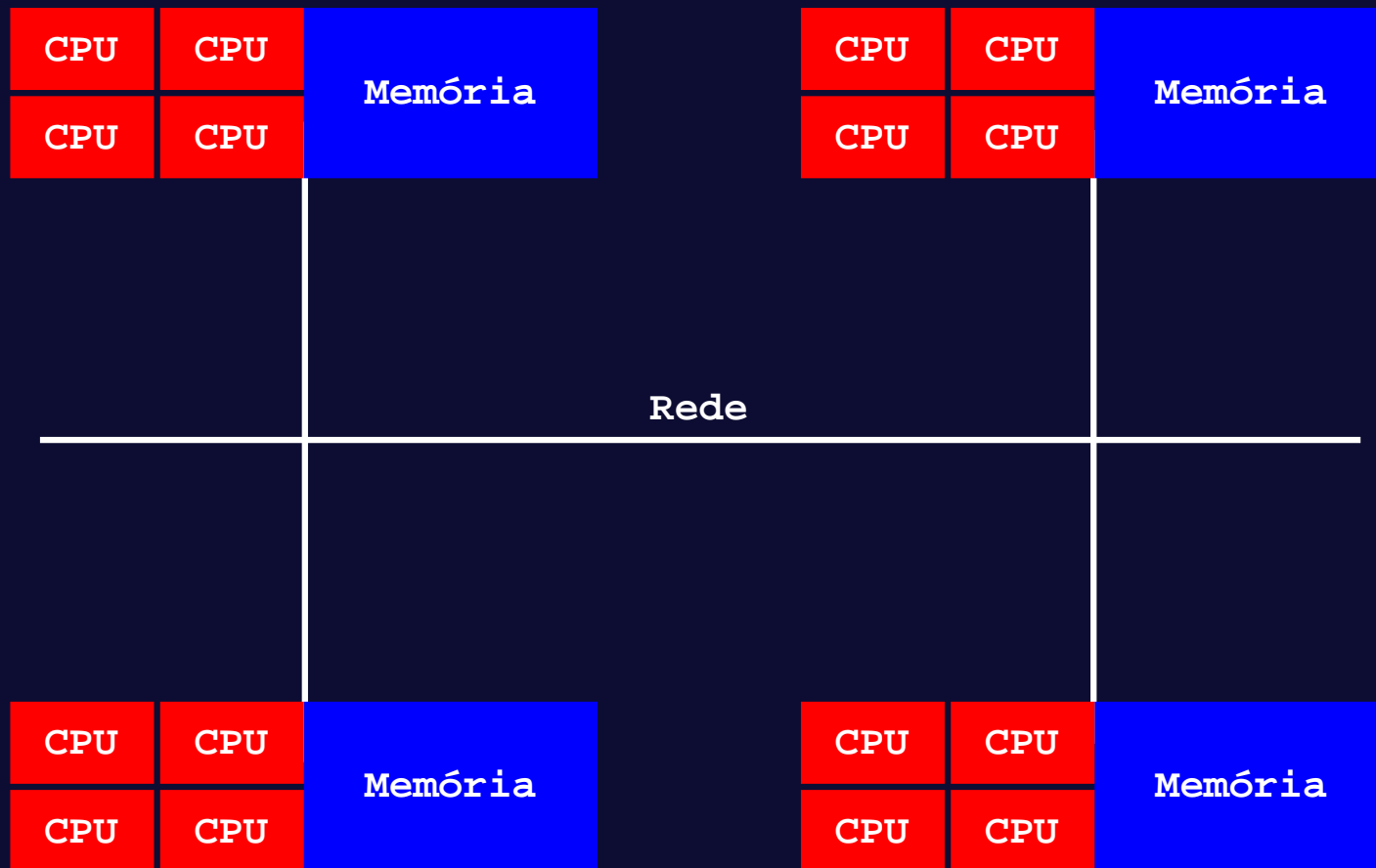
- Tempos de execução (em segundos) obtidos numa Sun SparcCenter 2000 com 8 processadores.
- Em média, o YapOr com um agente é cerca de 10% mais lento do que o Yap.

YapDss: Alguns Resultados

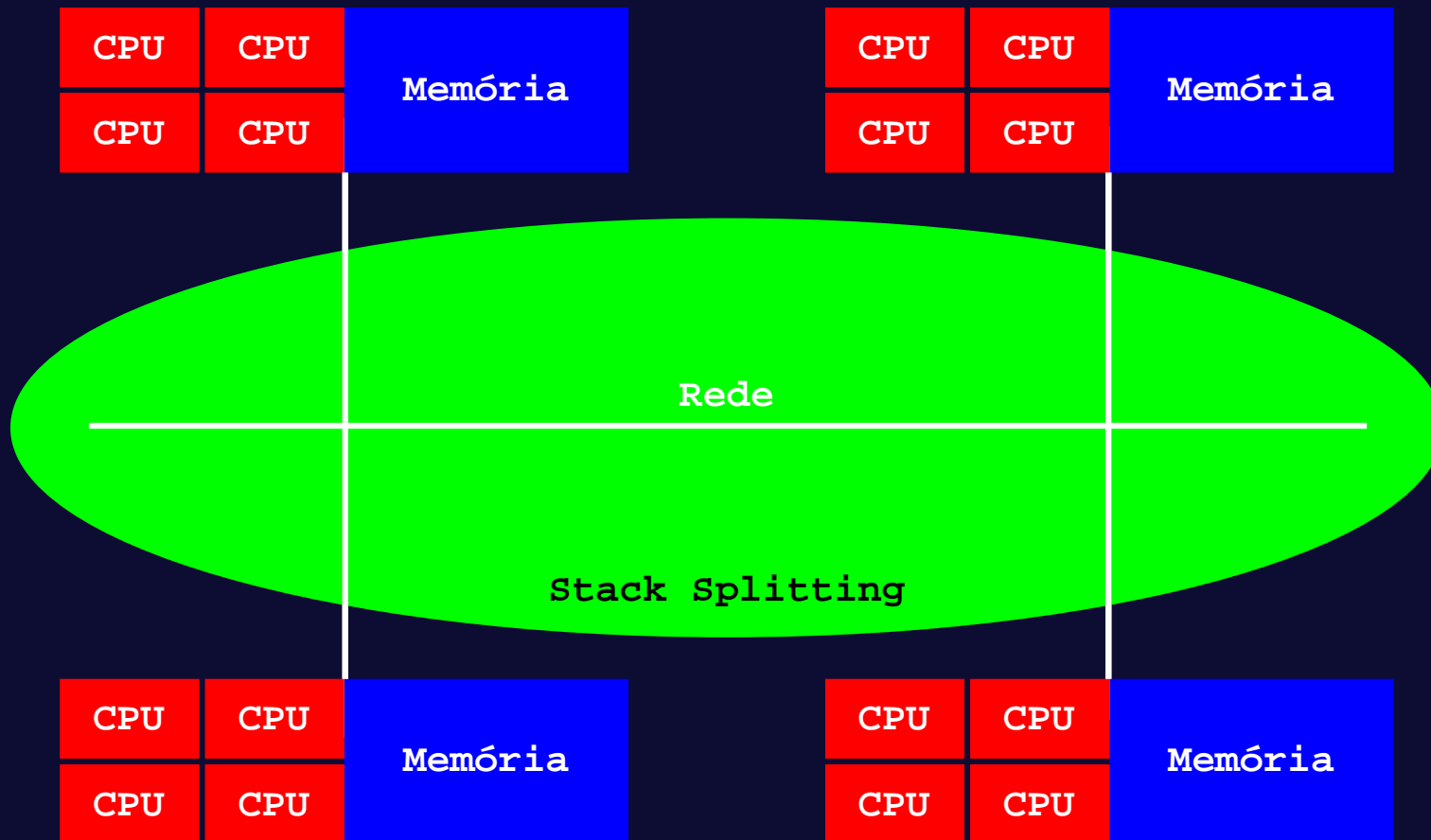
| Programa | 2 CPUs | 4 CPUs | 6 CPUs | 8 CPUs |
|------------------|---------------|--------------|--------------|--------------|
| 12-queens | 38.93 (1.99) | 19.63 (3.94) | 13.36 (5.80) | 10.12 (7.66) |
| nsort2 | 124.24 (1.98) | 63.14 (3.90) | 42.44 (5.80) | 33.06 (7.45) |
| puzzle4x4 | 34.00 (1.99) | 17.34 (3.91) | 11.83 (5.73) | 9.41 (7.20) |
| magic | 15.50 (1.99) | 7.88 (3.92) | 5.58 (5.53) | 4.38 (7.05) |
| 7cubes | 0.67 (1.96) | 0.40 (3.26) | 0.33 (3.90) | 0.23 (4.80) |
| ham | 0.17 (1.75) | 0.10 (2.81) | 0.09 (3.13) | 0.10 (2.95) |
| Média | (1.94) | (3.62) | (4.98) | (6.19) |

- Tempos de execução (em segundos) obtidos num cluster Beowulf com 4 nós dual Pentium II.
- Em média, o YapDss com um agente é cerca de 16% mais lento do que o Yap.

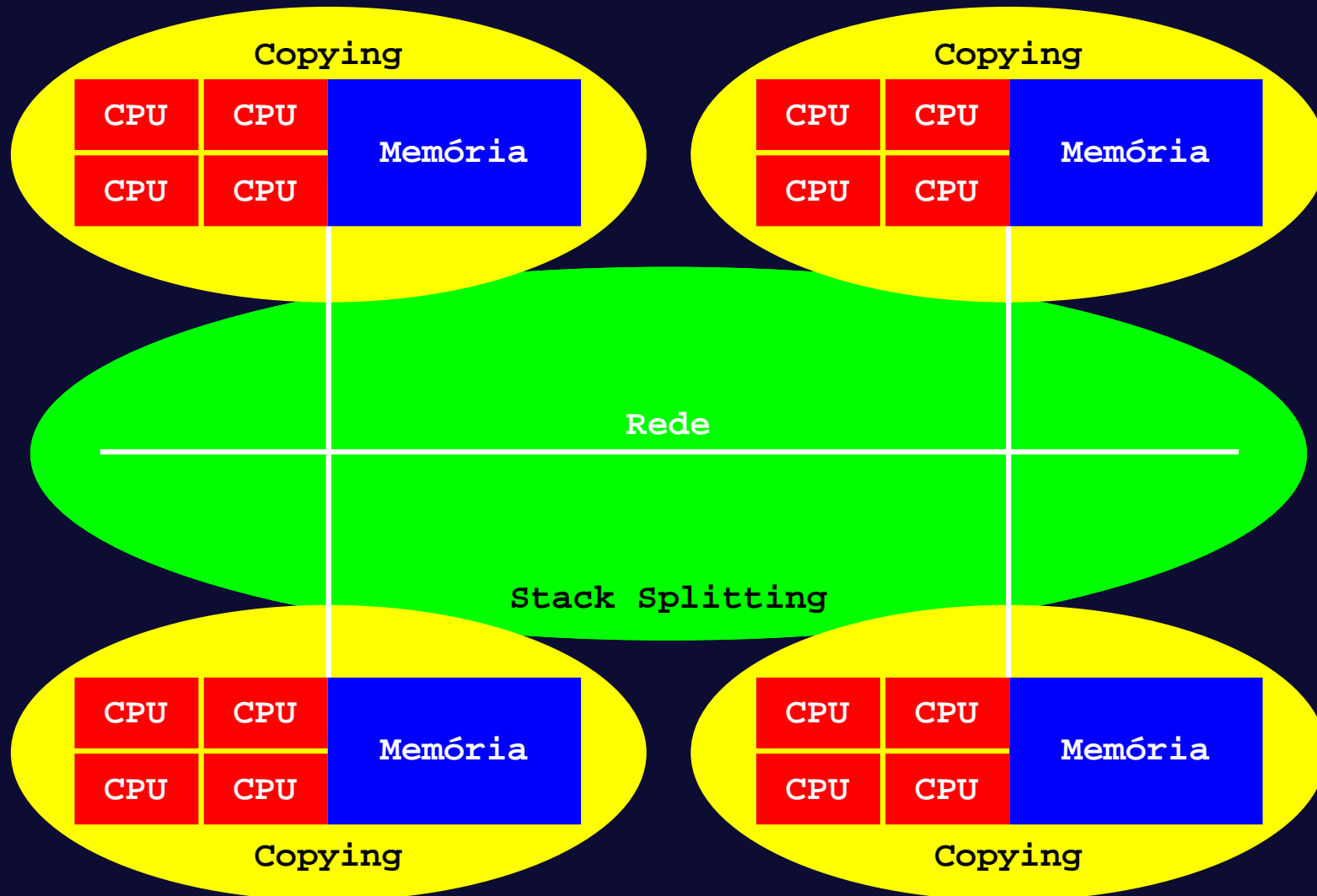
Trabalho Futuro



Trabalho Futuro



Trabalho Futuro



Bibliografia

- **YapOr: an Or-Parallel Prolog System Based on Environment Copying.** R. Rocha, F. Silva e V. Santos Costa. EPIA, Springer-Verlag LNAI 1695, páginas 178-192. 1999.
- **YapDss: an Or-Parallel Prolog System for Scalable Beowulf Clusters.** R. Rocha, F. Silva e R. Martins. EPIA, Springer-Verlag LNAI 2902, páginas 136-150. 2003.