

espaços de tuplas



Sistemas de mensagens

- suporte para ponto a ponto e grupos
- desacoplamento
 - temporal
 - persistência das mensagens
 - espacial
 - canais e congêneres



MPI - message passing interface

- abstrações
 - endereços virtuais de processos
 - 0, 1, 2, ...
 - comunicação em grupo
 - comunicadores
- aplicações fortemente acopladas



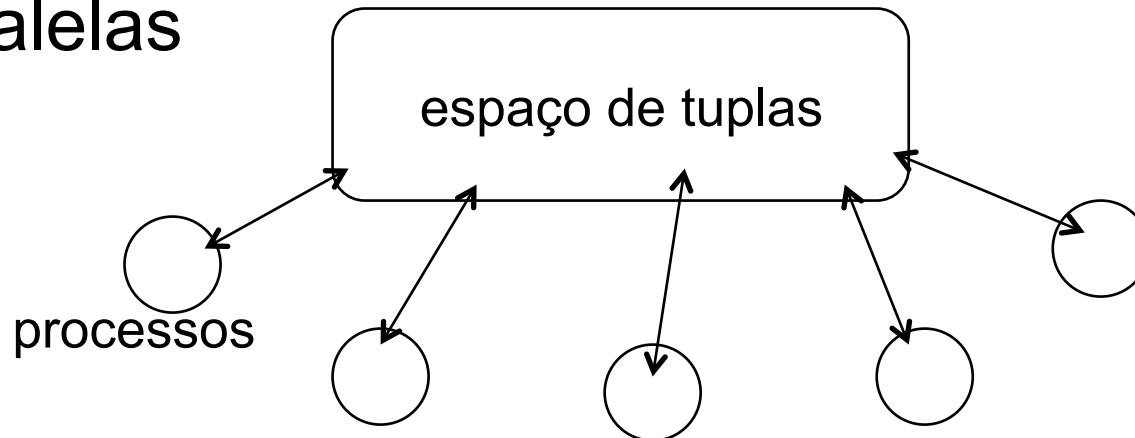
canais ou filas de mensagens

- desacoplamento espacial
 - destinatário é canal e não processo
- desacoplamento temporal
 - canal de destino pode sobreviver a execução do processo origem
- exemplo representativo: espaços de tuplas



Linda

- Carriero and Gelernter, 1986
 - Yale
- idéia de um espaço de tuplas compartilhado por diversos processos
- inicialmente voltado para aplicações paralelas



Linda - primitivas

- `out (tuple)`
 - coloca tupla `t` no espaço de tuplas
- `in (template)`
 - retorna retirando tupla que “case” com `template` do espaço de tuplas
 - bloqueante
- `rd (template)`
 - como `in` sem eliminar tupla do `et`
- `eval (t)`
 - dispara novo processo para calcular tupla e fazer `out(t)`
- primitivas para C e FORTRAN
- comunicação dentro de uma aplicação **fechada**



casamento de tuplas

- endereçamento por conteúdo (ou memória associativa)

– valores e tipos:

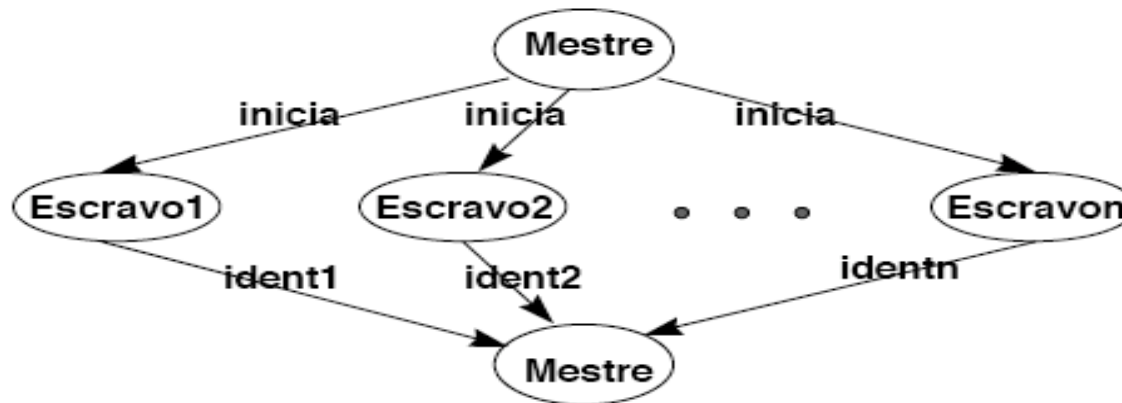
```
int x; float y;
```

```
in (3, ?x, "blablabla", ?y)
```

– dificuldade de implementação x utilidade?



Linda - exemplo de programa



exemplo em Linda

```
real_main (int argc, char *argv[ ]) {
    int procs, procid, hello( ), j;
    procs = atoi (argv[1]);
    for (j=0; j<procs; j++)
        eval ("worker", hello(j));
    for (j=0; j<procs; j++) {
        in ("worker", ?procid);
        fprintf (stdout, "fim do proc %d\n",procid);
    }
}
hello (int i){
    return i;
}
```



exemplo do wpp

```
lmain()
{
    int    i, ok;

    for(i = 2; i < LIMIT; ++i) {
        eval("primes", i, is_prime(i));
    }

    for(i = 2; i <= LIMIT; ++i) {
        rd("primes", i, ? ok);
        if (ok) printf("%d\n", i);
    }
}
```

```
is_prime(me)
    int    me;
{
    int    i, limit, ok;
    double    sqrt();

    limit = sqrt((double) me) + 1;

    for (i = 2; i < limit; ++i) {
        rd("primes", i, ? ok);
        if (ok && (me%i == 0)) return 0;
    }
    return 1;
}
```

N. Carriero e D. Gelernter. How to write parallel programs: a guide to the perplexed. ACM Computing Surveys, 21(3), set 1989.



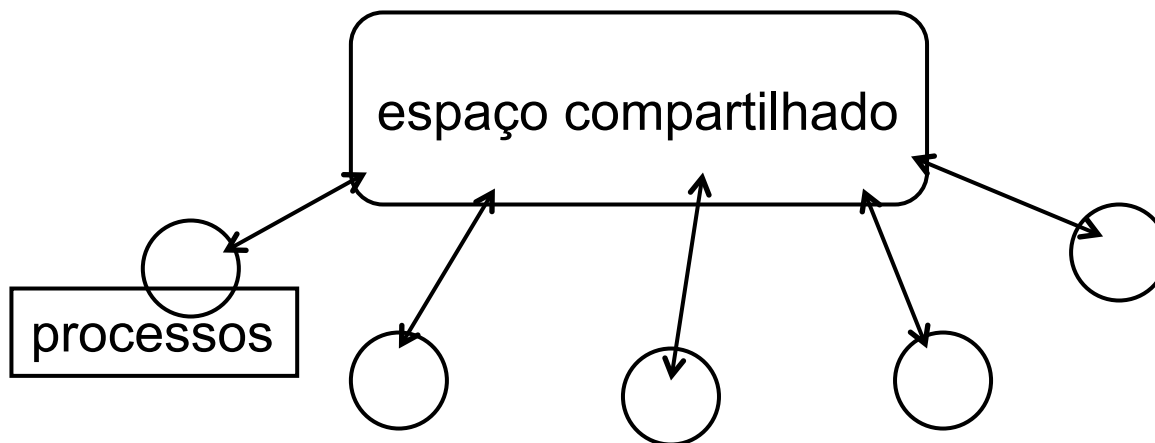
construção de outras abstrações

- canais tipados
- semáforos
- publish/subscribe



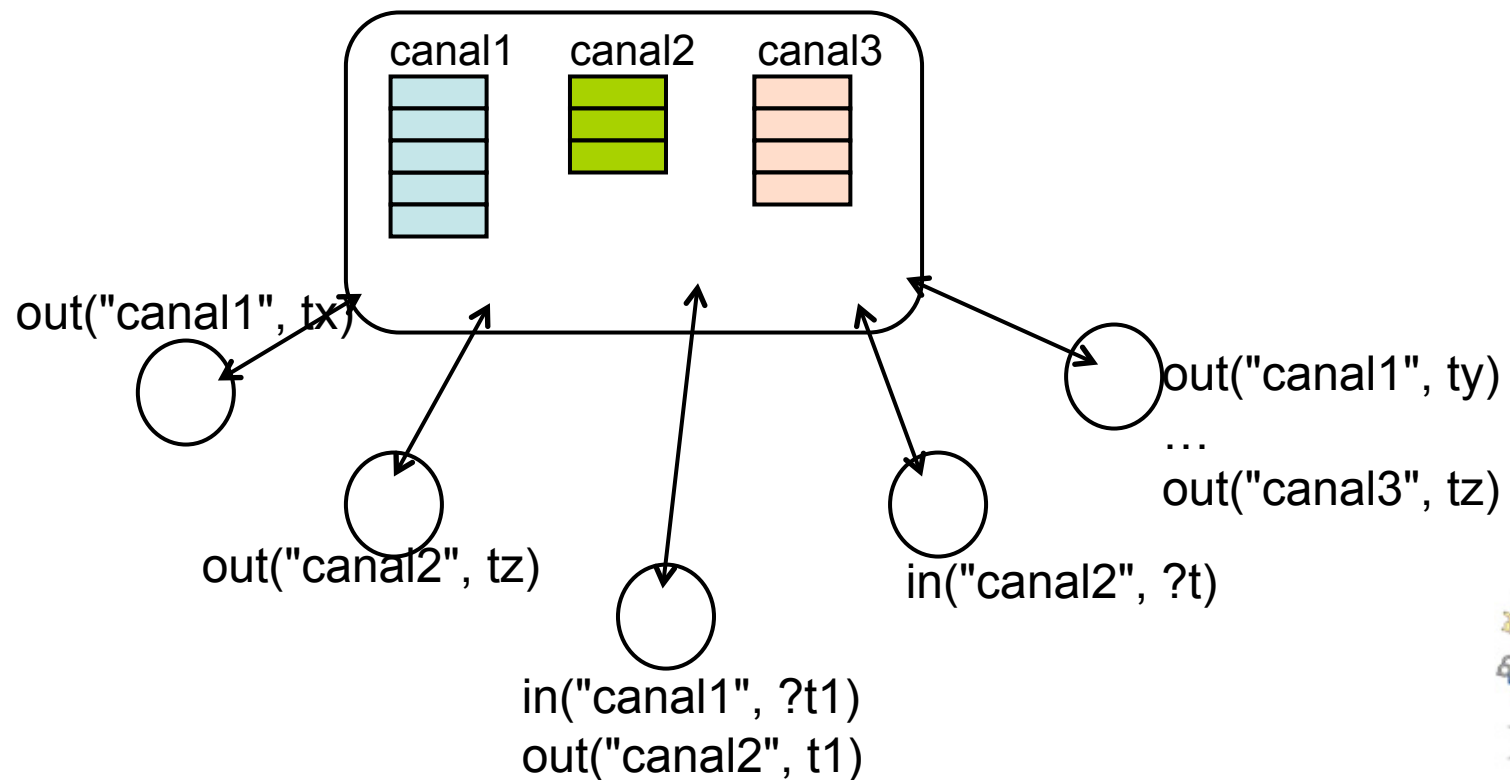
interpretações do espaço de tuplas

- muitas vezes chamado de espaço virtual compartilhado



interpretações do espaço de tuplas

- mas também podemos ver como um sistema de canais de comunicação



espaços de tuplas em Java

- Tspaces e *JavaSpaces*
 - `space.write (entry, ...)`
 - `space.read (template, ...)`
 - `space.take (template, ...)`

 - read e take ainda bloqueantes
 - eval não existe
- entradas e templates são instâncias de classes
 - um objeto encapsula a antiga tupla



JavaSpaces

- distanciamento de programação paralela e aproximação com sistemas distribuídos
 - espaços "ainda mais" persistentes
 - aplicação não precisa estar em execução para tupla ser retirada
 - desacoplamento maior
 - utilização simultânea de diferentes espaços
 - uso do registry RMI ou outros mecanismos de localização
 - integração com middleware Jini



implementações

- em geral centralizadas
- algumas implementações distribuídas
 - e muita discussão
 - replicação completa de espaço, rd e in locais, e out com broadcast
 - out local e broadcast de in e rd



Coordenação

- separação
 - processamento X comunicação e sincronização
 - Gelernter, D. and Carriero, N. 1992. Coordination languages and their significance. Commun. ACM 35, 2 (Feb. 1992), 97-107.

