CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 39: Memória Compartilhada em MPI

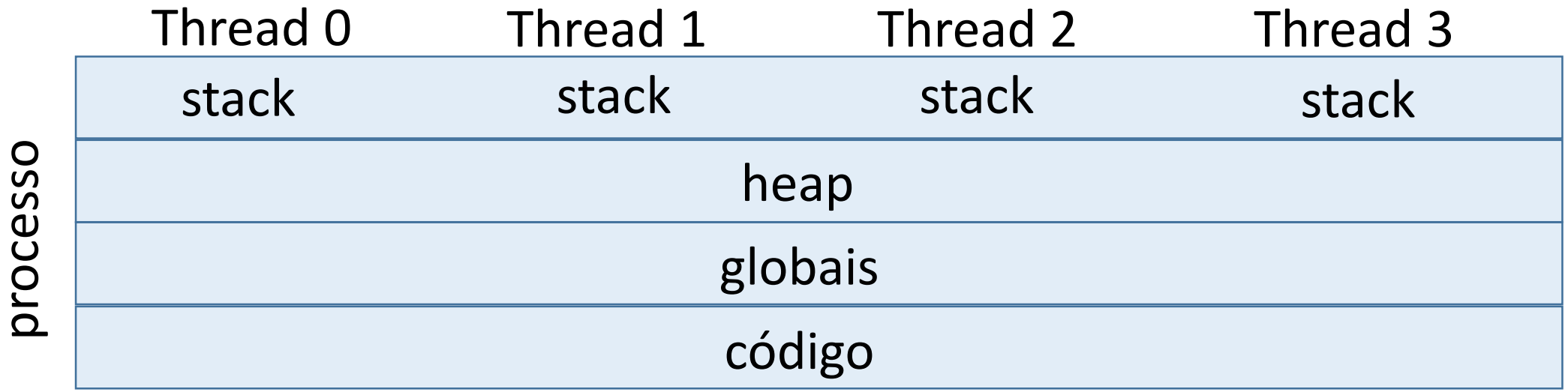
Celso L. Mendes, Stephan Stephany

LAC / INPE

Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

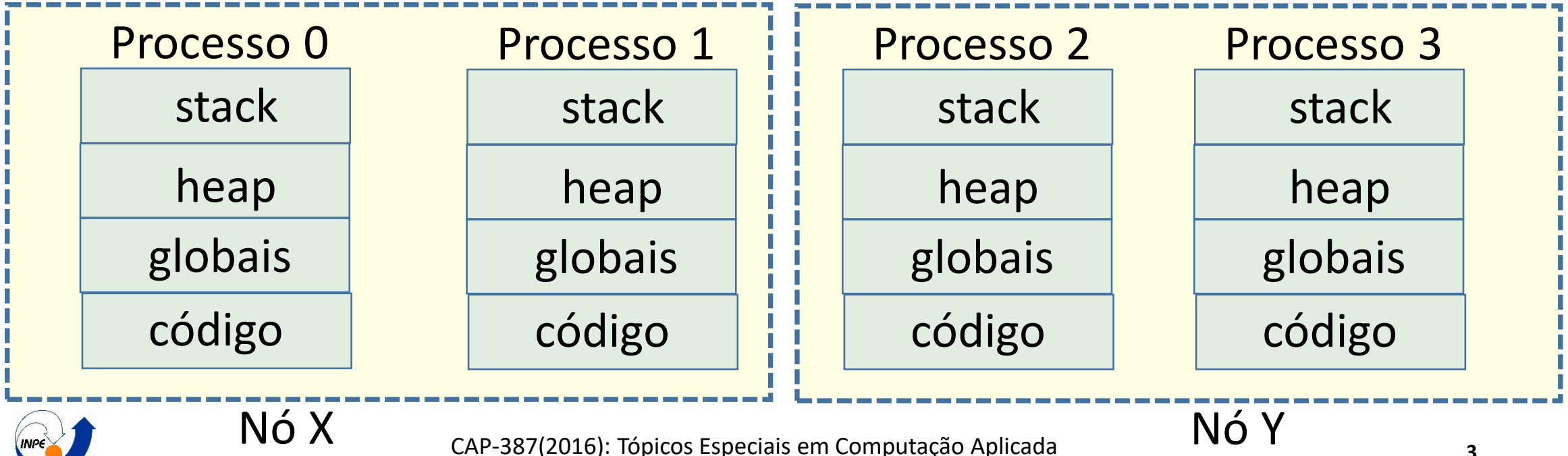
Modelo Compartilhado

- Programação em memória compartilhada
  - Compartilhamento total de memória entre CPUs
  - Modelos: Pthreads, OpenMP



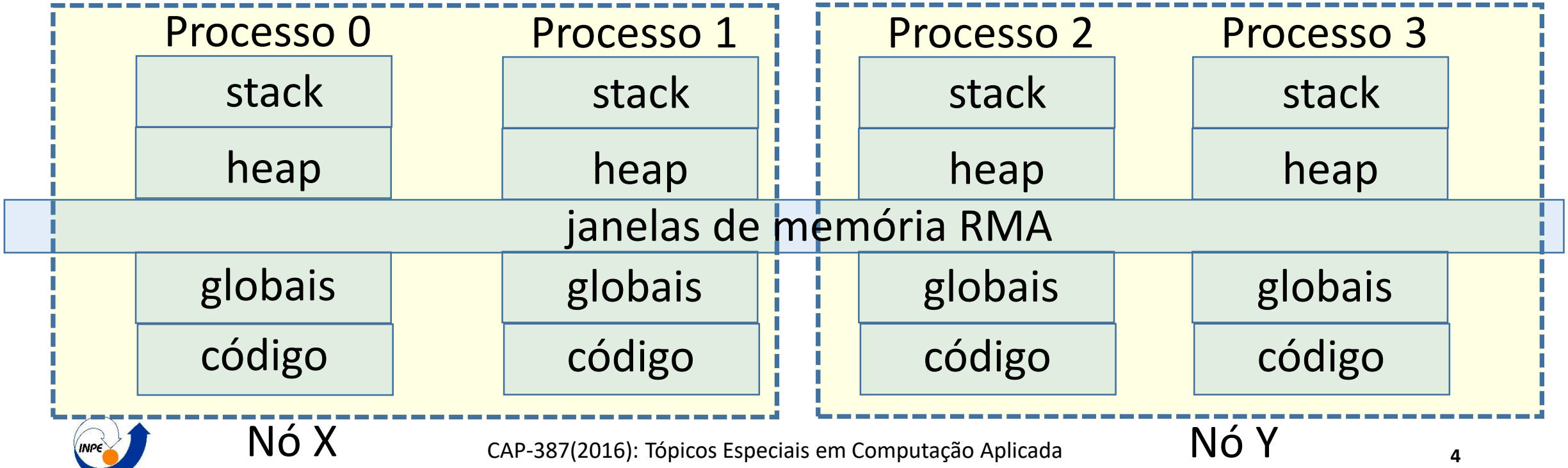
Modelo Distribuído

- Programação em memória distribuída
  - Nenhum compartilhamento de memória entre CPUs
  - Modelos: MPI, PVM, …



Compartilhamento com RMA

- Comunicação Unilateral em MPI
  - Janelas de memória expostas por cada processo MPI
  - Acesso remoto sempre via *MPI_put/MPI_get* (RMA)

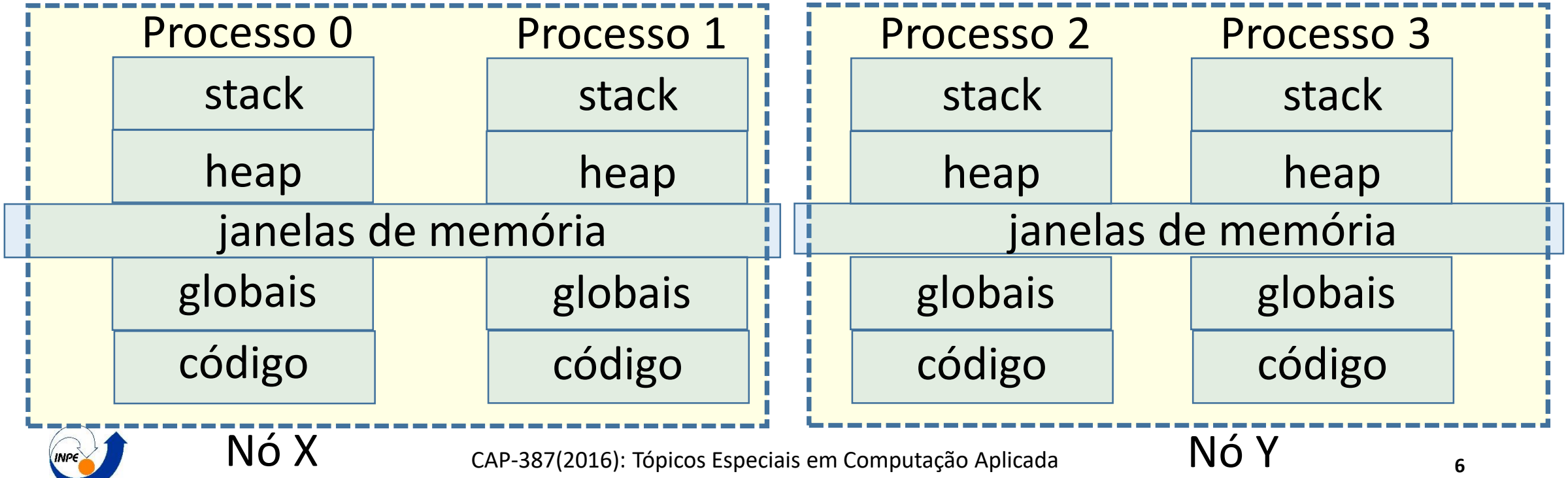


Compartilhamento de Memória

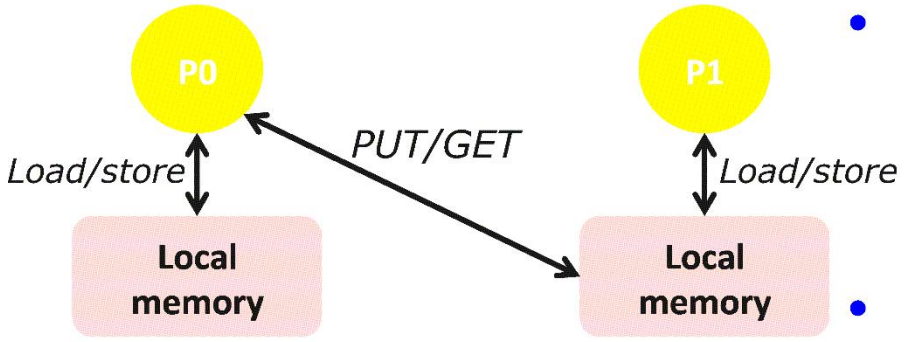
- Compartilhamento disciplinado de janelas
  - Vantagens:
    - Bugs têm muito menor potencial de ação (afetam apenas a área de memória na janela)
    - Programação e depuração são simplificadas
    - Fácil de integrar com programas MPI existentes
- MPI-3: Janelas em memória compartilhada
  - Mesmas propriedades de janelas MPI já vistas

Janelas em Mem.Compartilhada

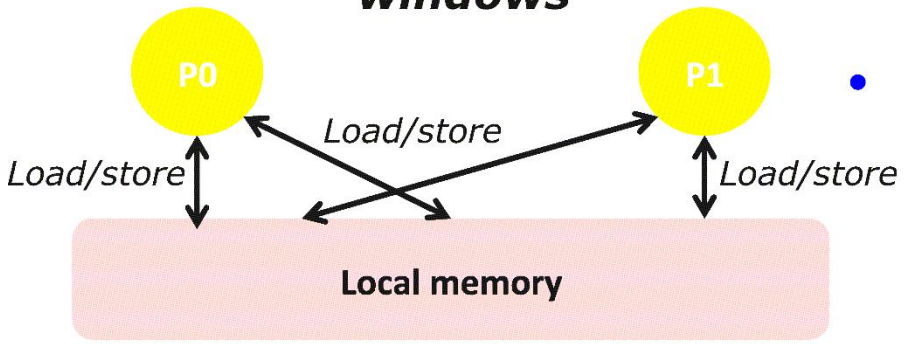
- Para processos MPI (*ranks*) num mesmo nó:
  - Janelas de memória *podem* ser acessadas via *load/store*!



Diferença de Janelas RMA



Traditional RMA windows



Shared memory windows

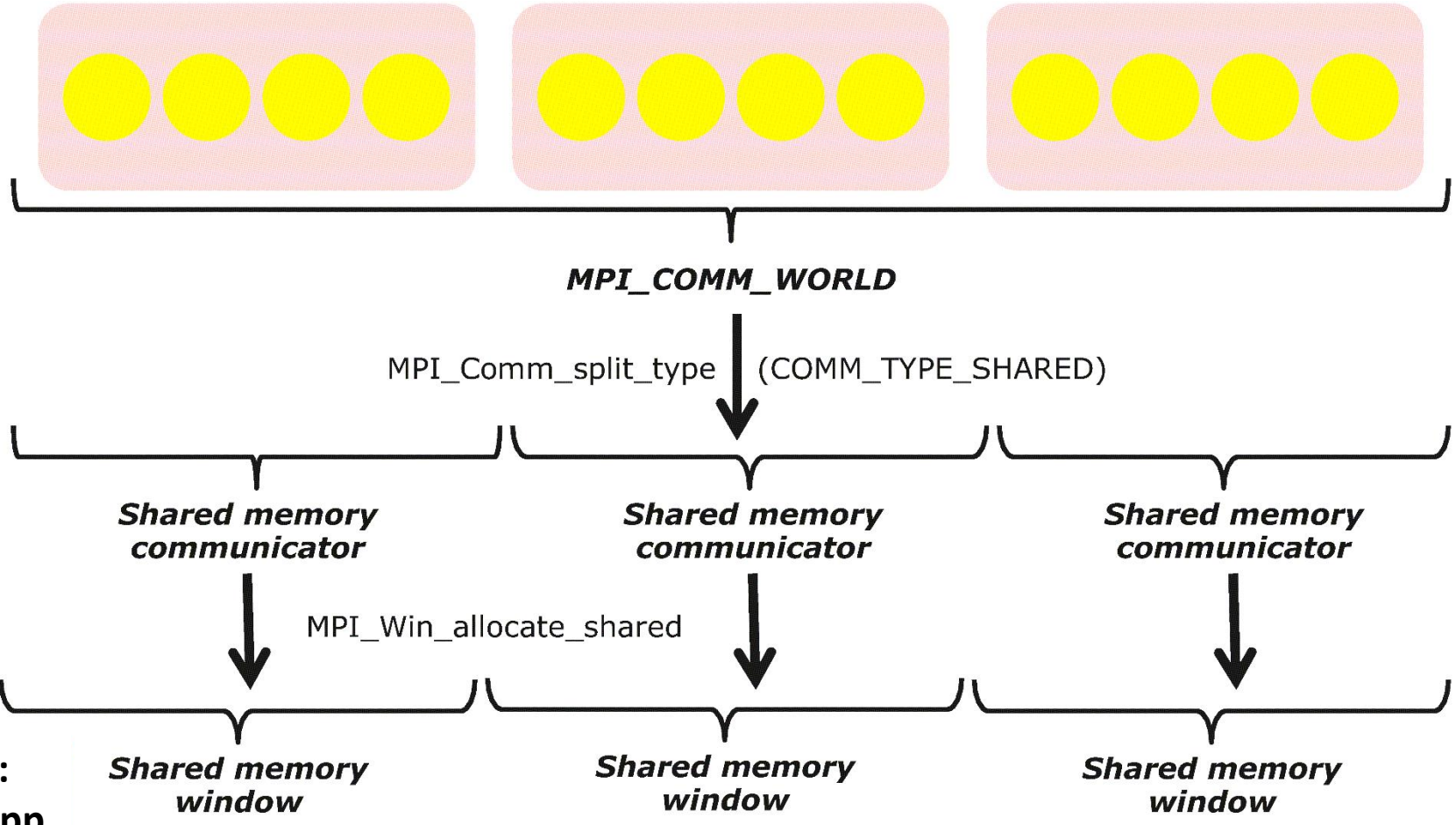
Fonte:
Bill Gropp

- Janelas RMA:
  - Acesso apenas via *put/get*
  - Podem estar no mesmo nó ou não
- Janelas Mem.Compartilhada:
  - Acesso via *put/get* ou via *load/store*
  - Propriedades de RMA ainda são válidas
  - Devem estar num certo nó

Janelas em Mem.Compartilhada

- Uso típico:
  - Alocação e armazenamento de estruturas de dados usadas por todos os processos MPI – apenas uma área por nó é utilizada, ao invés de uma por processo
    - Consumo de memória deixa de crescer com o número de núcleos do nó
- Caso mais simples: dados inicializados e só lidos
  - Basta colocar uma *fence* após a inicialização

Uso de Janelas de Mem. Compart.



Fonte:
Bill Gropp

Funções de Apoio: Sub-Comunicador

*int MPI_Comm_split_type(MPI_Comm comm, int split_type, int key, MPI_Info info, MPI_Comm *newcomm)*

- Cria um novo comunicador com ranks de mem. comum
- Comunicadores: *comm* (original), *newcomm* (novo)
- Se *split_type=MPI_COMM_TYPE_SHARED*, o novo comunicador incluirá o maior número possível de ranks que compartilham memória
- *key* pode indicar a ordem dos ranks no novo comunicador (*key*=0: mesma ordem original)

Exemplo no Santos Dumont

Programa: hello.c

- Novo comunicador adicionado, com MPI_COMM_TYPE_SHARED

```
. . .
MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL,
&shmcomm);
MPI_Comm_size(shmcomm,&numprocs2);
MPI_Comm_rank(shmcomm,&myid2);
fprintf(stdout,"[%d] NewRank %d of %d on %s (hostname %s) \n",
        myid, myid2, numprocs2, processor_name, name);
```

- Cenário de execução:

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
```

Exemplo no Santos Dumont

Saída da execução:

```
Rank 1 of 16 on sdumont1407 (hostname sdumont1407)
Rank 9 of 16 on sdumont1408 (hostname sdumont1408)
Rank 2 of 16 on sdumont1407 (hostname sdumont1407)
Rank 3 of 16 on sdumont1407 (hostname sdumont1407)
Rank 4 of 16 on sdumont1407 (hostname sdumont1407)
Rank 10 of 16 on sdumont1408 (hostname sdumont1408)
Rank 5 of 16 on sdumont1407 (hostname sdumont1407)
Rank 11 of 16 on sdumont1408 (hostname sdumont1408)
Rank 6 of 16 on sdumont1407 (hostname sdumont1407)
Rank 14 of 16 on sdumont1408 (hostname sdumont1408)
Rank 7 of 16 on sdumont1407 (hostname sdumont1407)
Rank 15 of 16 on sdumont1408 (hostname sdumont1408)
Rank 0 of 16 on sdumont1407 (hostname sdumont1407)
Rank 8 of 16 on sdumont1408 (hostname sdumont1408)
Rank 12 of 16 on sdumont1408 (hostname sdumont1408)
Rank 13 of 16 on sdumont1408 (hostname sdumont1408)
```

```
[0] NewRank 0 of 8 on sdumont1407 (hostname sdumont1407)
[1] NewRank 1 of 8 on sdumont1407 (hostname sdumont1407)
[2] NewRank 2 of 8 on sdumont1407 (hostname sdumont1407)
[8] NewRank 0 of 8 on sdumont1408 (hostname sdumont1408)
[3] NewRank 3 of 8 on sdumont1407 (hostname sdumont1407)
[9] NewRank 1 of 8 on sdumont1408 (hostname sdumont1408)
[4] NewRank 4 of 8 on sdumont1407 (hostname sdumont1407)
[10] NewRank 2 of 8 on sdumont1408 (hostname sdumont1408)
[5] NewRank 5 of 8 on sdumont1407 (hostname sdumont1407)
[11] NewRank 3 of 8 on sdumont1408 (hostname sdumont1408)
[6] NewRank 6 of 8 on sdumont1407 (hostname sdumont1407)
[12] NewRank 4 of 8 on sdumont1408 (hostname sdumont1408)
[7] NewRank 7 of 8 on sdumont1407 (hostname sdumont1407)
[13] NewRank 5 of 8 on sdumont1408 (hostname sdumont1408)
[14] NewRank 6 of 8 on sdumont1408 (hostname sdumont1408)
[15] NewRank 7 of 8 on sdumont1408 (hostname sdumont1408)
```

Funções de Apoio: Criar Janela

*int MPI_Win_allocate_shared(MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win)*

- Criação da janela com memória a ser compartilhada
- Mesmos parâmetros de janelas RMA normais
- Área alocada é retornada via ponteiro *baseptr*
- *baseptr* pode ser usado localmente: *baseptr[100]=1;*
- Contudo, ponteiro *baseptr* só é válido no processo que faz a criação da janela (não deve ser usado por outros processos)
- Comunicador *comm* deve conter apenas processos que compartilham memória

Funções de Apoio: Pesquisar Janelas

*int MPI_Win_shared_query(MPI_Win win, int rank, MPI_Aint *size, int *disp_unit, void *baseptr)*

- Recupera dados da janela/memória criada no processo *rank*
- *baseptr* é um ponteiro válido apenas no processo que chamou *MPI_Win_shared_query*
- *size* e *disp_unit* retornam propriedades da janela remota

Uma vez obtendo *baseptr*, pode-se acessar área remota diretamente:

baseptr[200]=0; x=baseptr[300]

Exemplo de uso: Stencil 2-D

```
MPI_Comm_split_type(comm, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &shmcomm);
MPI_Win_allocate_shared (size*sizeof(double), sizeof(double), info, shmcomm, &mem, &win);
MPI_Win_shared_query(win, north, &sz, &dispunit, &northptr);
MPI_Win_shared_query(win, south, &sz, &dispunit, &southptr);
MPI_Win_shared_query(win, east, &sz, &dispunit, &eastptr);
MPI_Win_shared_query(win, west, &sz, &dispunit, &westptr);
for (iter=0; iter<niters; ++iter) {
    MPI_Win_fence(0,win);
    if (north != MPI_PROC_NULL) for (i=0; i<bx; ++i) a2[…]=northptr[…];
    if (south != MPI_PROC_NULL) for (i=0; i<bx; ++i) a2[…]=southptr[…];
    if (east != MPI_PROC_NULL) for (i=0; i<by; ++i) a2[…]=eastptr[…];
    if (west != MPI_PROC_NULL) for (i=0; i<by; ++i) a2[…]=westptr[…];
    update_grid(&a1, &a2);
}
```

north,south,east,west: ranks vizinhos, devem estar no mesmo nó; se não estiverem, comunicação via *MPI_Send/MPI_Recv* será necessária