

Lab 4d: implementação de broadcast (my_Bcast)
e comparação com o MPI_Bcast

Incluído nas páginas seguintes figuras com
dicas de COMO pode ser implementado o my_Bcast

OBS1.: vários trabalhos já entregues estão com medições diferentes de latência e vazão. Note que se for medido com poucas mensagens existe muita variação nos resultados.

Como meio de evitar resultados errados devido ao problema acima podemos fazer assim:

medir com número de mensagens suficiente para que cada experiencia dure de 3 a 4 segundos aproximadamente.
Isso seria suficiente para que a medição seja mais precisa, sem gastar muito tempo

Indicar quantas mensagens foram transmitidas para CADA tamanho de mensagem.

Por exemplo: para mensagens de 8 bytes
o numero de mensagens para que
o experimento dure 3 segundos é
BEM MAIOR que o numero de mensagens
para tamanho 16000.

(continua...)

OBS2: Na sua implementação você pode usar `MPI_Isend`, ou `MPI_Send`. Porém, alguns alunos apontaram um aumento de latência quando transmitimos MUITAS mensagens. Isso pode ter sido causado pelo uso de `MPI_Isend`. SE isso estiver acontecendo, troque sua implementação para usar `MPI_Send`.

OBS3:
O Prof. disponibilizou uma função

`verifica_my_Bcast(...)`

Essa função deve ser chamada AO FINAL do seu programa, DEPOIS que ele apresentar suas medições, E antes de desalocar seu buffer, e também ANTES de finalizar o MPI. Essa função vai verificar se sua função `my_Bcast` está funcionando adequadamente.

OBS4:
Seu programa 4c DEVE funcionar adequadamente para qualquer número de processos MPI, e qualquer raiz de broadcast especificada na linha de comando com o parametro `-r`

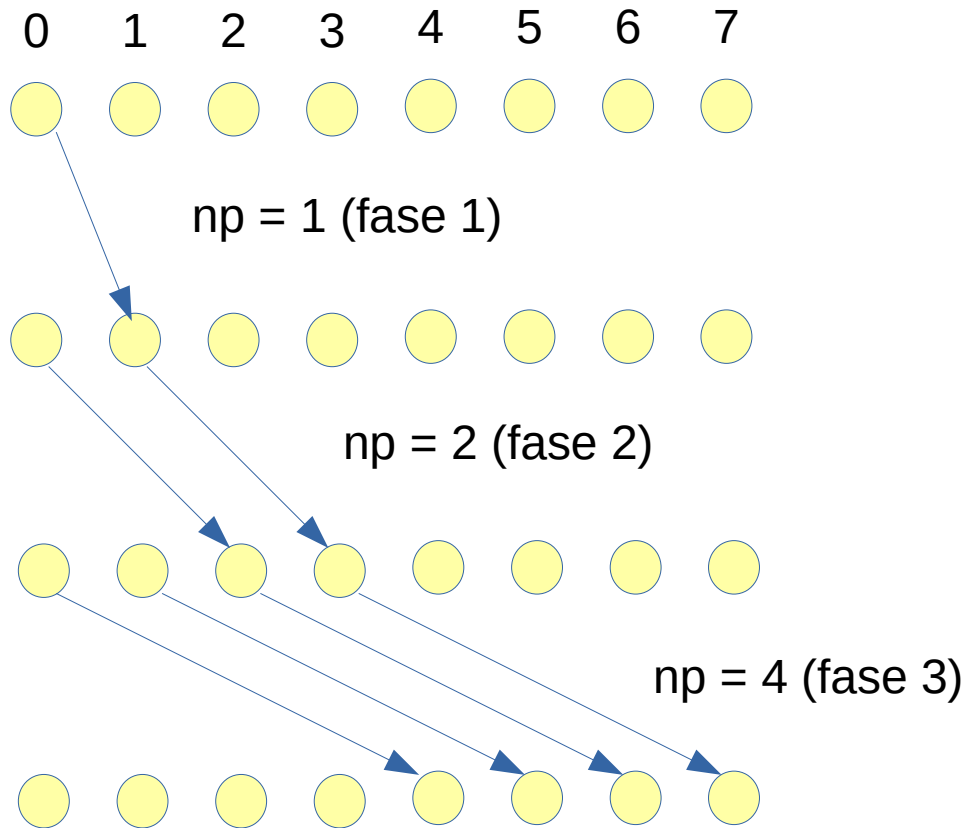
OBS5:
Sua função `my_Bcast` DEVE ter a mesma interface (usar os parâmetros e tipos) na função `MPI_Bcast`.
Ou seja, deve ter a mesma funcionalidade e aceitar o parametro que indica que nodo será a raiz do broadcast.

OBS6:
Os GRAFICOS do relatório e medidas apresentadas pelo seu programas devem ter as seguintes UNIDADES:
Vazao de broadcasts: em MB/s (megabytes por segundo)
Onde mega = 10^6 (ou seja, potencia de DEZ)

Latências de broadcasts: em us (microsegundos)

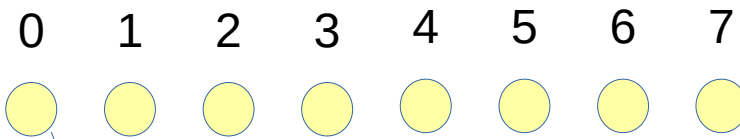
(figuras nas páginas abaixo...)

np = número de participantes a cada fase



... note que np dobra a cada fase

np = número de participantes a cada fase



np = 1 (fase 1)



np = 2 (fase 2)



np = 4 (fase 3)



Que nodos fazem MPI_Recv ?

Regra simples
Em um dado broadcast
TODOS recebem,
menos o raiz

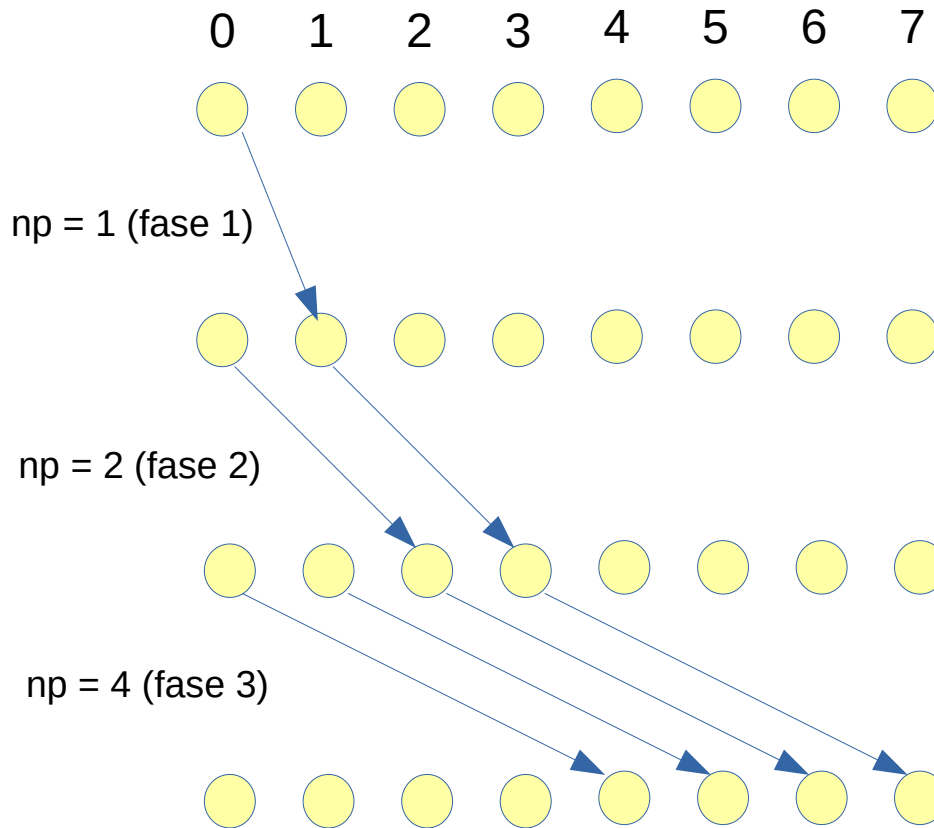
Qual é a origem de
CADA MPI_Recv ?

Regra simples
Para evitar ter de especificar
a origem, podemos
simplesmente especificar
MPI_ANY_SOURCE

np = número de participantes a cada fase

Que nodos fazem MPI_Send ?

Note que a cada fase podemos especificar o rank de quem recebe em função de np, assim:



rank < np
→ 0 < 1

rank < np
→ 0 < 2
→ 1 < 2

rank < np
→ 0 < 4
→ 1 < 4
→ 2 < 4
→ 3 < 4

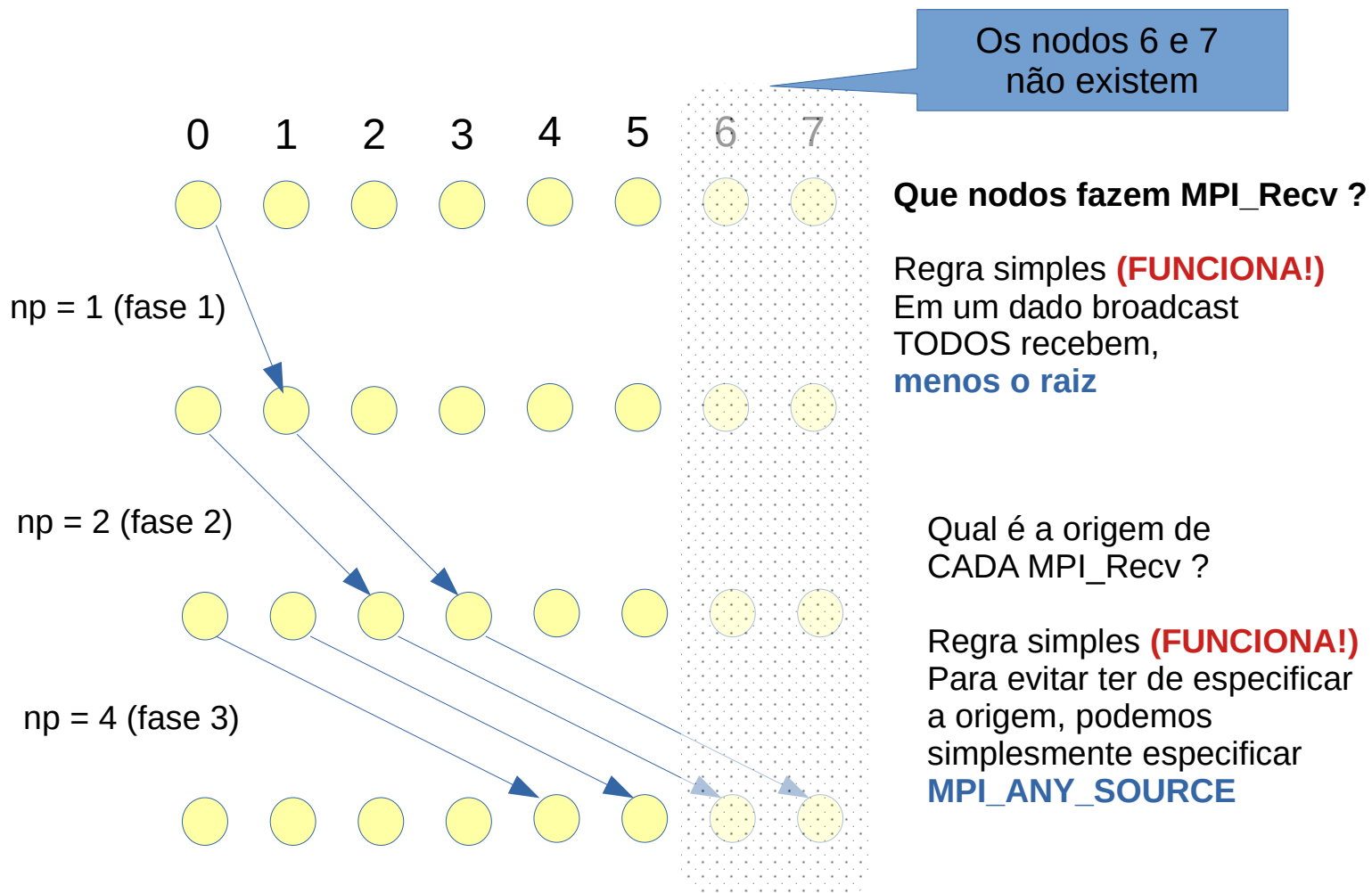
...

Que acontece SE número de nodos NÃO é potencia de 2 ?

Para a etapa de recepção nada precisa ser feito,
Porque o rank do nodo não existirá,
Então simplesmente a recepção não ocorre para nodos inexistentes.

Exemplo:

- no desenho abaixo temos apenas 6 nodos (comm_size = 6)
- os nodos 6 e 7 não existem → então não farão MPI_Recv



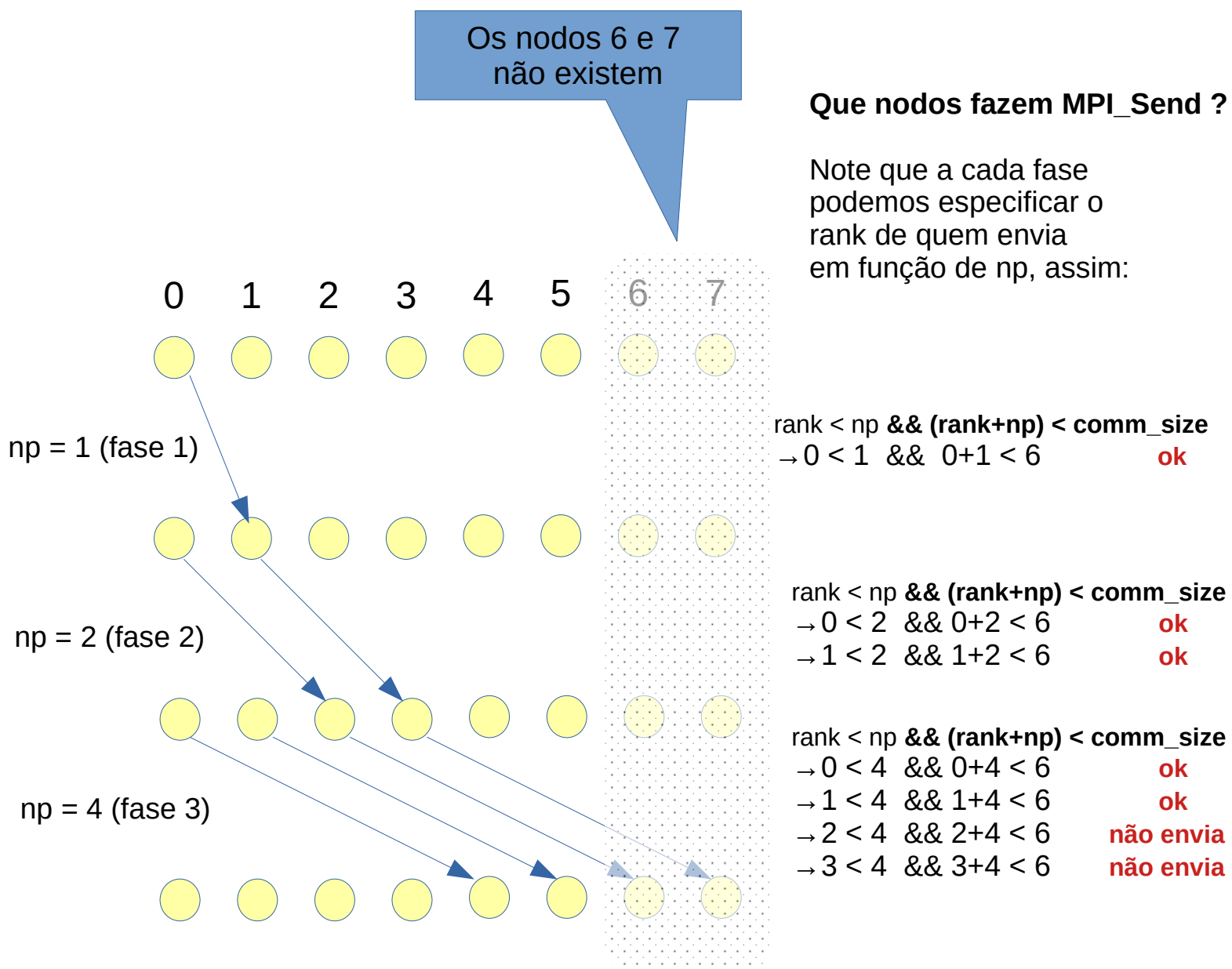
Que acontece SE número de nodos NÃO é potencia de 2 ?

Para a etapa de MPI_Send temos de cuidar para para NÃO enviar para nodo inexistente.
Como **o destino do envio é para o nodo rank + np**,
basta verificar TAMBÉM se:
 $\text{Rank} + \text{np} < \text{comm_size}$

Assim não envia SE o destino for nodo inexistente

Exemplo:

- no desenho abaixo temos apenas 6 nodos (comm_size = 6)
- os nodos 6 e 7 não existem → então os nodos x e x não enviarão



...

Que acontece SE o raiz NÃO é o nodo 0 ?

Basta usar **números lógicos** para os ranks nas condições de envio vistas anteriormente. Ou seja, em vez de usar ranks físicos de um nodo vamos ter o conceito de rank lógico, assim:

Por exemplo:


- no desenho abaixo temos o raiz com rank físico 3:
 - esse nodo raiz deve ter rank_lógico 0
 - O nodo seguinte tem rank físico 4, terá rank_lógico 1 ...
 - e assim por diante (fazendo a operação de módulo)
- Temos a seguinte fórmula para se obter o rank_logico de um nodo:

$$\text{rank_logico} = (\text{rank} + \text{comm_size} - \text{raiz}) \% \text{comm_size}$$

onde:

rank é o rank físico do nodo, obtido pela função MPI_Comm_rank(...)

Com a fórmula acima podemos ver os ranks lógicos obtidos para os físicos, para o exemplo com 8 nodos e **raiz do broadcast no nodo 3**:



Rank (físico)	0	1	2	3	4	5	6	7
Rank lógico	5	6	7	0	1	2	3	4

A fórmula acima funciona para qualquer comm_size.

Existe outra fórmula simples para convertes de rank_logico em físico, dada abaixo.

MAS pode ser que você NÃO precise disso, mesmo assim temos a fórmula, caso queira usá-la em alguma parte do seu código de broadcast:

- Temos a seguinte fórmula para se obter o rank (físico) a partir do rank_logico :

$$\text{rank} = (\text{rank_logico} + \text{raiz}) \% \text{comm_size}$$

onde:

rank é o rank físico do nodo

Como parar o loop interno da função my_Bcast ?

Basta usar **número de participantes (variável np)**.

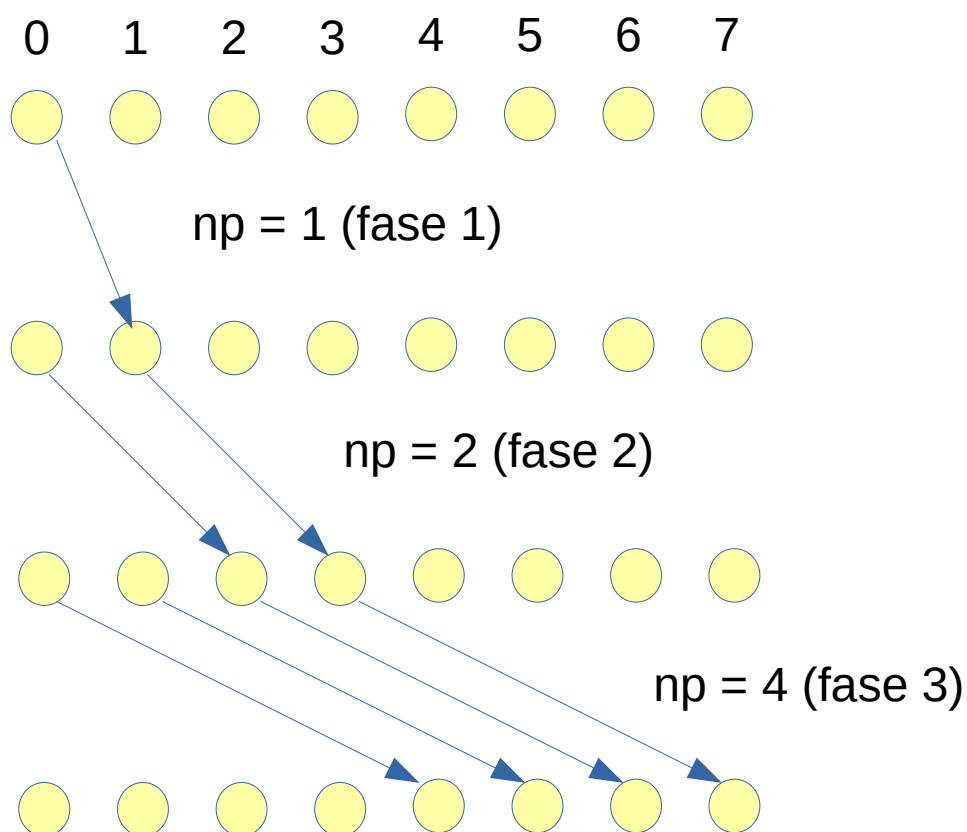
Uma iteração deve acontecer enquanto:

$$np < comm_size$$

Por exemplo, usando 8 nodos como o exemplo inicial, revisto abaixo:

- no desenho vemos que a última fase foi com $np = 4$.
- A próxima fase NÃO deve existir, ou seja, o loop deve parar pela condição acima pois dobrando o valor de np teríamos $np = 8$, e teríamos envios desnecessários

np = número de participantes a cada fase



... note que np dobra a cada fase