# Sistemas Distribuídos

Prof. André Nasserala

andre.nasserala@ufac.br

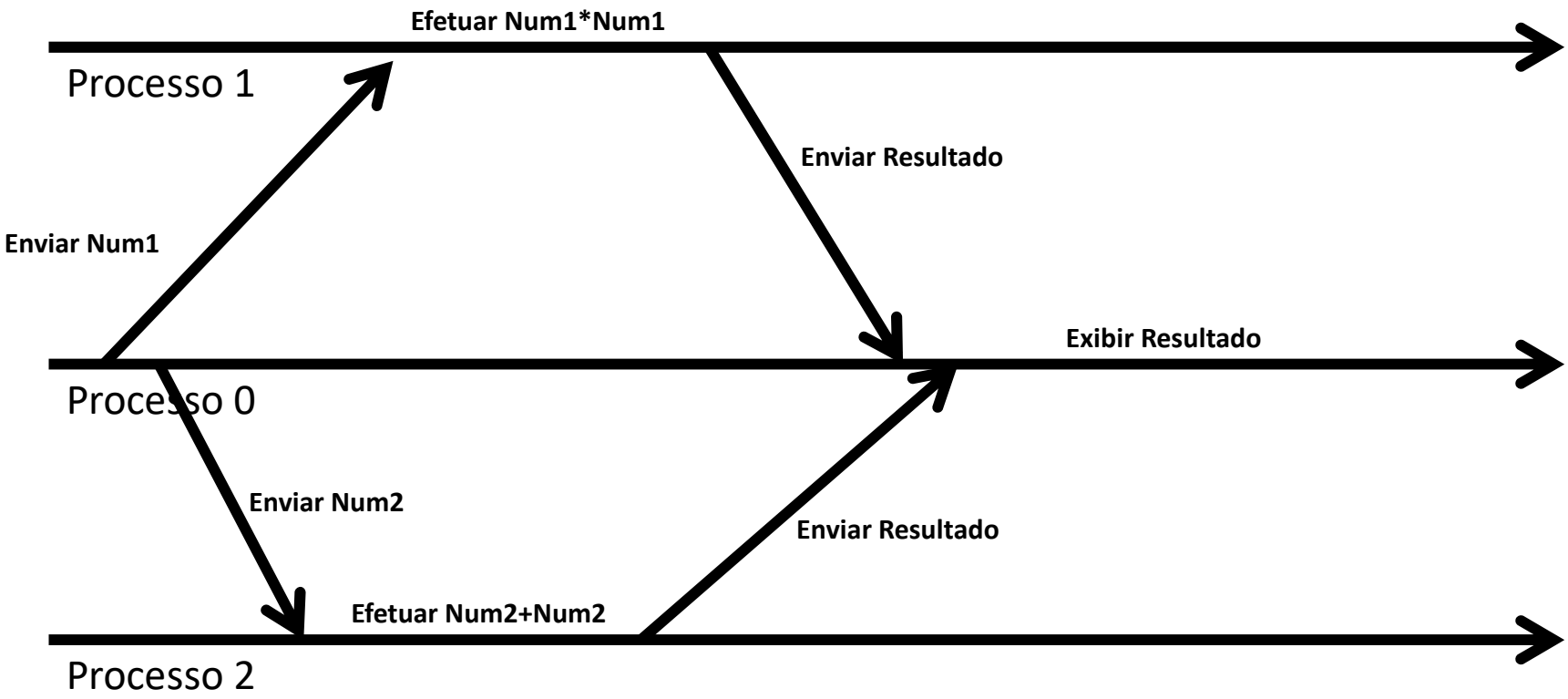
## Será visto nessa parte:

- Problemas Clássicos da Programação Distribuída.

# Exemplo1

- Fazer um programa em C com MPI que utilizando 3 processos, faça o processo 0 enviar dois números, um para o processo 1 e outro para o processo 2. O processo 1 deve receber o número do processo 0, fazer a operação Num1*Num1, e então mandar o resultado para o processo 0. O processo 2 deve receber o numero do processo 0, fazer a operação Num2+Num2, e então devolver o resultado para o processo 0. Finalmente o processo 0 deve exibir os resultados recebidos dos processos 1 e 2.

# Exemplo1


Efetuar Num1*Num1
Processo 1
Enviar Resultado
Enviar Num1
Exibir Resultado
Processo 0
Enviar Num2
Enviar Resultado
Efetuar Num2+Num2
Processo 2

# Exemplo1

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mpi.h"
main(int argc, char** argv)
{
    int meu_rank, tag=0;
    int Num1, Num2 , Res1, Res2;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &meu_rank);
    if (meu_rank == 0) {
        Num1 = 2;
        Num2 = 3;
        MPI_Send(&Num1, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);
        MPI_Send(&Num2, 1, MPI_INT, 2, tag, MPI_COMM_WORLD);
        MPI_Recv(&Res1, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &status);
        MPI_Recv(&Res2, 1, MPI_INT, 2, tag, MPI_COMM_WORLD, &status);
        printf("o Resultado do Processo 1  e %d\n", Res1);
        printf("o Resultado do Processo 2  e %d\n", Res2);
    }
```

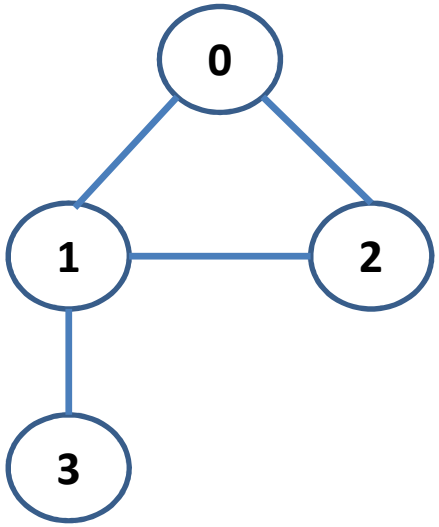
# Exemplo1

```
        else if (meu_rank == 1) {
            MPI_Recv(&Num1, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
            Res1 = Num1*Num1;
            MPI_Send(&Res1, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
        }
        else {
            MPI_Recv(&Num2, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
            Res2 = Num2+Num2;
            MPI_Send(&Res2, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
        }
        MPI_Finalize( );

}
```

# Exemplo2

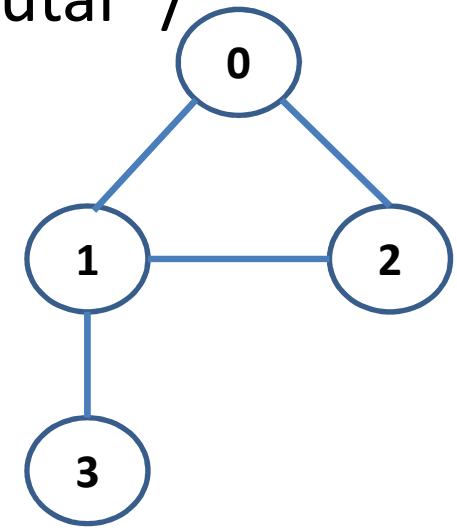
- Tem-se uma rede de computadores descrita conforma grafo abaixo:




- Faça um programa e C com MPI para que cada nó imprima quem são seus vizinhos.

# Exemplo2

- #include <stdio.h>
- #include <mpi.h>
- /*Definir o grafo da aplicação antes de executar*/
- int numeroDeTarefas = 4;
- int matrizVizinhanca[4][4] = {{0,1,1,0},
- {1,0,1,1},
- {1,1,0,0},
- {0,1,0,0}};
- /*1 se refere a ter ligação e 0 a não ter ligação*/



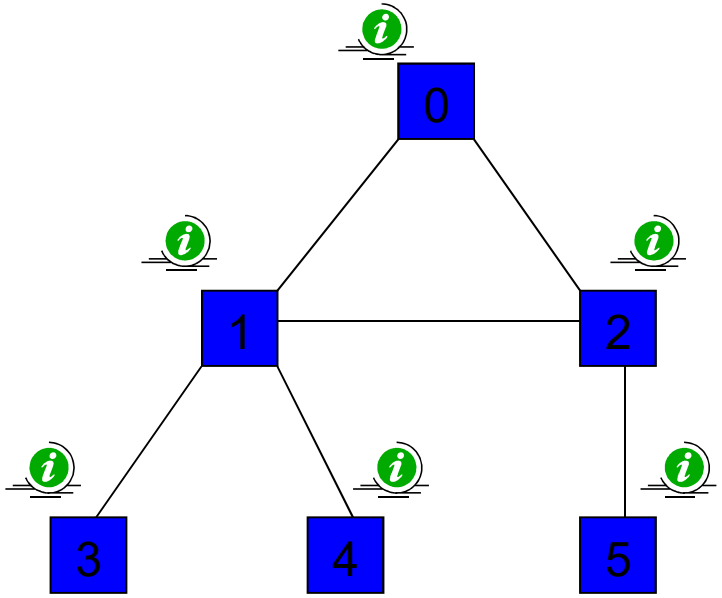
# Exemplo2

```
int contaNumeroDeVizinhos(int meuRank) {
    int i;
    int contador = 0;
    for (i = 0; i < numeroDeTarefas; i++){
        if (matrizVizinhanca[meuRank][i] == 1) {
            contador++;
            printf("O no %d e vizinho de %d\n",meuRank,i);
        }
    }
    return contador;
}
```

# Exemplo2

- int main(int argc, char** argv) {
- int numeroDeVizinhos, meuRank;
- //inicialização do MPI
- MPI_Status status;
- MPI_Init(&argc, &argv);
- MPI_Comm_rank(MPI_COMM_WORLD, &meuRank);
- numeroDeVizinhos = contaNumeroDeVizinhos(meuRank);
- printf("o no %d tem %d vizinhos\n",meuRank, numeroDeVizinhos);
- MPI_Finalize();
- }

# Propagação de Informações

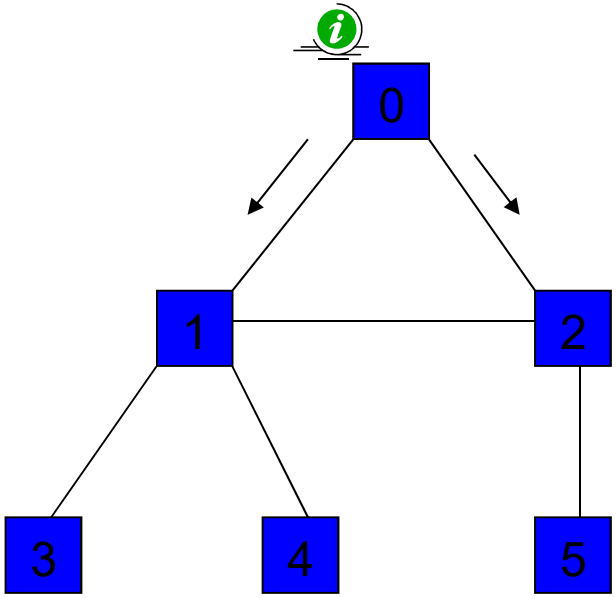



Nó 0 gera uma informação que tem de ser encaminhada a todos os demais.

# Propagação de Informações
# Algoritmo

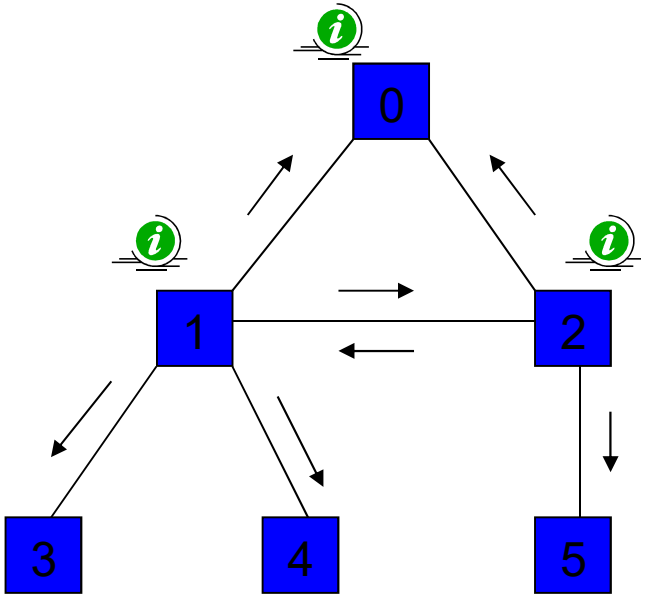
```
Variáveis:
    alcançado = falso
Ação para fonte inicial da informação
  (inf):
  alcançado := verdadeiro;
  envie inf a todos os vizinhos;
Ao receber inf
  se alcançado = falso
     alcançado := verdadeiro;
     envie inf a todos os vizinhos;
```

# Propagação de Informações
# Execução



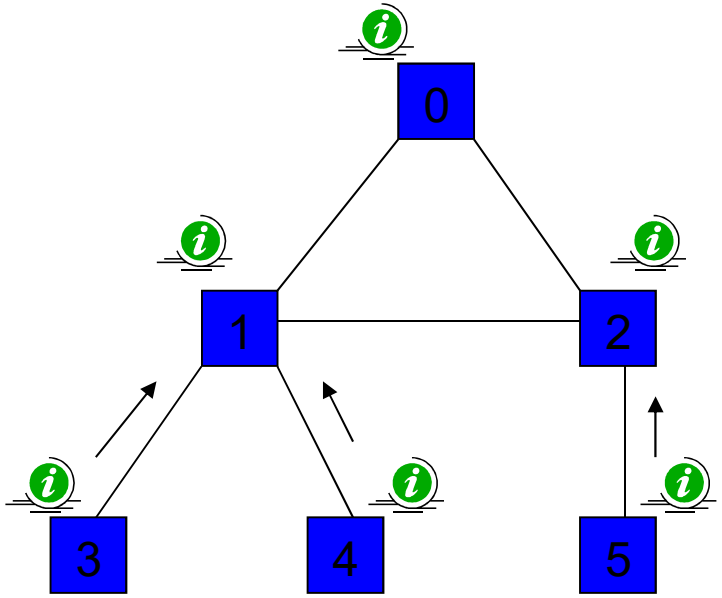

Nó 0 gera uma informação e a encaminha a todos os seus vizinhos

# Propagação de Informações
# Execução




Nós 1 e 2 recebem a informação e a encaminham a todos os seus vizinhos

# Propagação de Informações
# Execução




Nós 3, 4 e 5 recebem a informação e a encaminham a todos os seus vizinhos

# Exemplo3

```
#include <stdio.h>
#include <mpi.h>
int numeroDeTarefas = 6;
int matrizVizinhanca[6][6] = {   {0,1,1,0,0,0},
                                 {1,0,1,1,1,0},
                                 {1,1,0,0,0,1},
                                 {0,1,0,0,0,0},
                                 {0,1,0,0,0,0},
                                 {0,0,1,0,0,0}};
int main(int argc, char** argv)
{
   int i, myRank, source, tag = 50, alcancado = 0;
   char message[100] = "Oi!";
   MPI_Status status;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
```

# Exemplo3

```
if (myRank == 0) {
      alcancado = 1;
      for (i = 0; i < numeroDeTarefas; i++){
        if (matrizVizinhanca[myRank][i] == 1) {
          printf("Enviando a informação para %d\n", i);
          MPI_Send(message, 100, MPI_CHAR, i, tag, MPI_COMM_WORLD);
        }
      }
   }
```

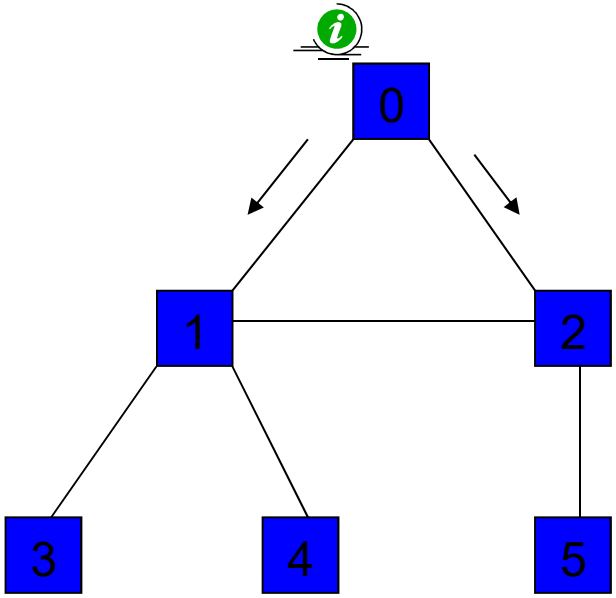
# Exemplo3

```
else {
    if ( alcancado == 0 ) {
     MPI_Recv(message, 100, MPI_CHAR, MPI_ANY_SOURCE, tag,MPI_COMM_WORLD, &status);
     printf("Processo %d recebendo %s\n",myRank, message);
     alcancado = 1;
      for (i = 0; i < numeroDeTarefas; i++) {
        if (matrizVizinhanca[myRank][i] == 1){
          printf("Processo %d enviando %s para %d\n", myRank, message, i);
          MPI_Send(message, 100, MPI_CHAR, i, tag,MPI_COMM_WORLD);
        }
      }
    }
  }
  MPI_Finalize();
}
```

# Propagação de Informações

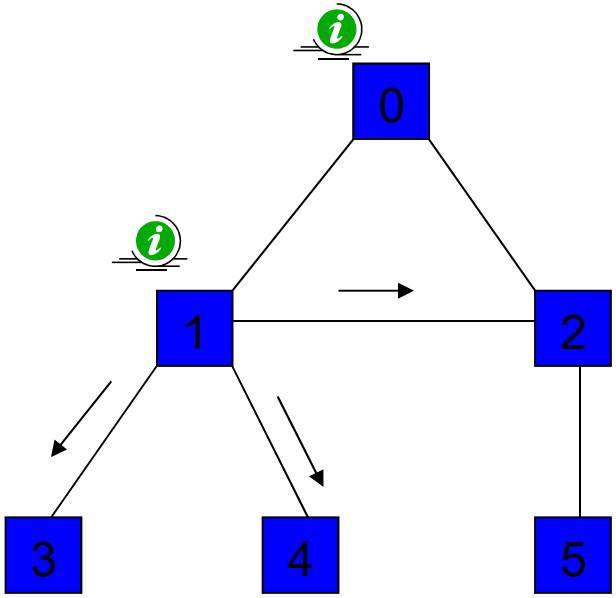
- Será possível aumentar a eficiência do algoritmo?
- Será necessária a propagação para TODOS os vizinhos?
- Inclusive o remetente da mensagem?
- Considere outra possível execução, representada nas próximas telas…

# Propagação de Informações
# Execução



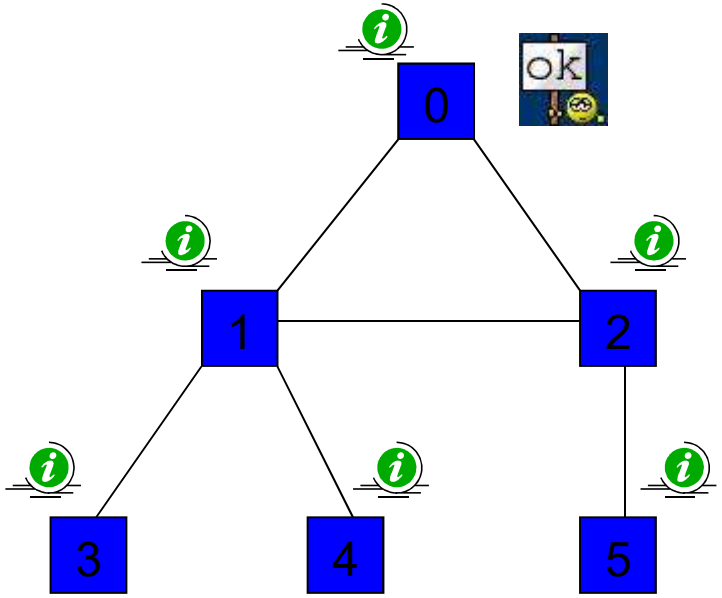

Nó 0 gera uma informação e a encaminha a todos os seus vizinhos

# Propagação de Informações
# Execução




- Nó 1 recebe a informação e a encaminha a todos os seus vizinhos, exceto o nó 0, mas canal de comunicação 0 - 2 está muito lento, e 2 recebe a informação de 1 antes de 0.
- Nó 0 poderá receber mensagem de 2 ou não, dependendo da ordem da chegada das mensagens!!

# Propagação de Informações com Realimentação

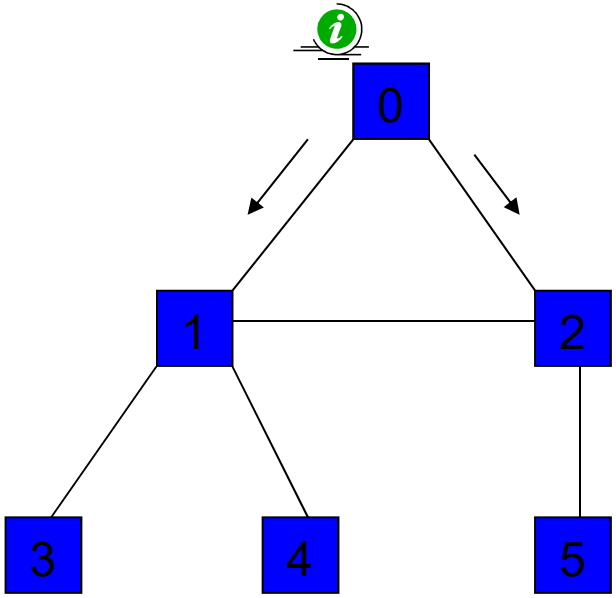



Nó 0 gera uma informação que tem de ser encaminhada a todos os demais e recebe uma confirmação quando a informação já estiver completamente disseminada.

# Propagação de Informações com Realimentação - Algoritmo

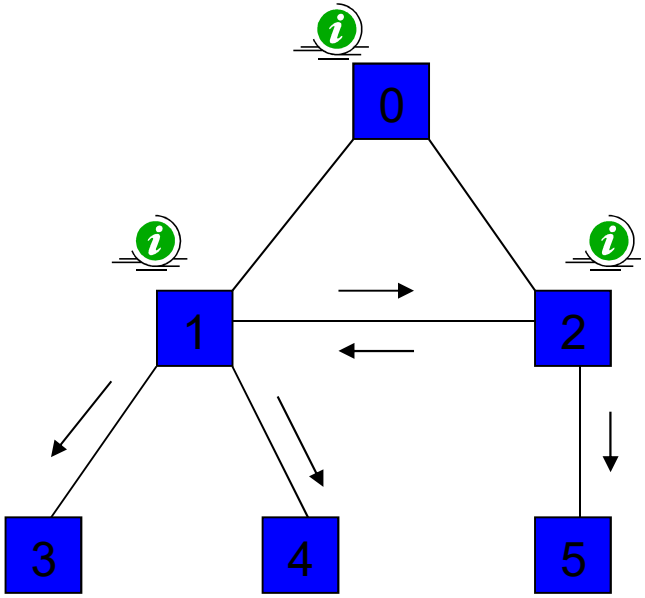
```
Variáveis
   pai := nil;
   count := 0;
   alcançado := falso;
Ação para fonte inicial da informação (inf):
   alcançado := verdadeiro;
   envie inf a todos os vizinhos;
Ao receber inf:
    count := count +1;
    se alcançado = falso
         alcançado := verdadeiro;
         pai := origem(inf);
         envie inf a todos os vizinhos exceto pai;
    se count = |vizinhos| e pai ≠ nil
         envie inf para pai;
```

# Propagação de Informações com Realimentação - Execução



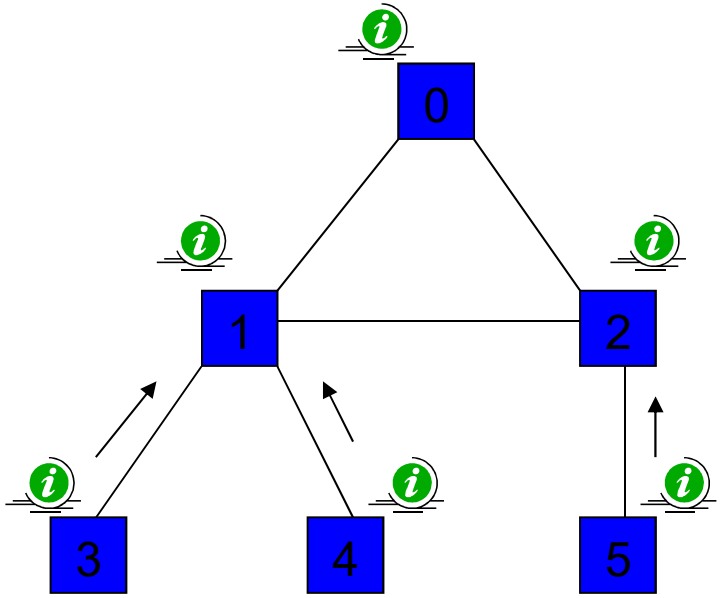

Nó 0 gera uma informação e a encaminha a todos os seus vizinhos

# Propagação de Informações com Realimentação - Execução



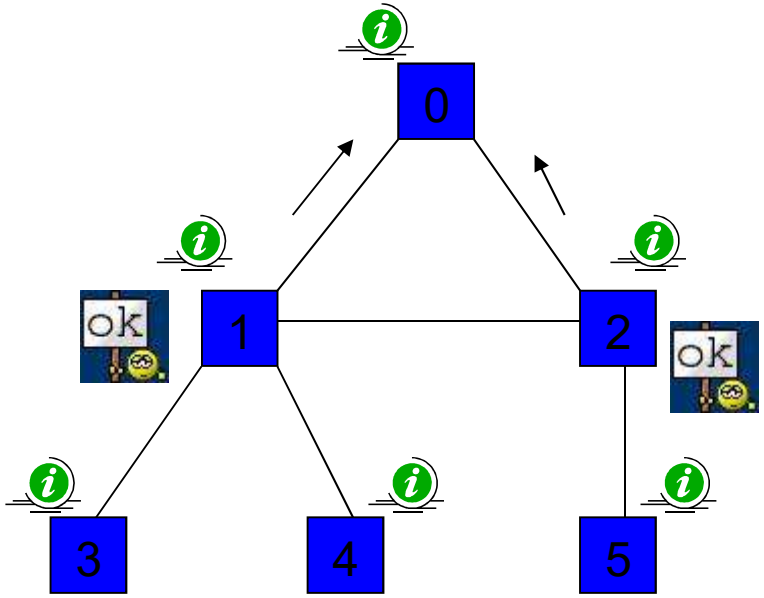

Nós 1 e 2 recebem a informação e a encaminham a todos os seus vizinhos, exceto pai

# Propagação de Informações com Realimentação - Execução



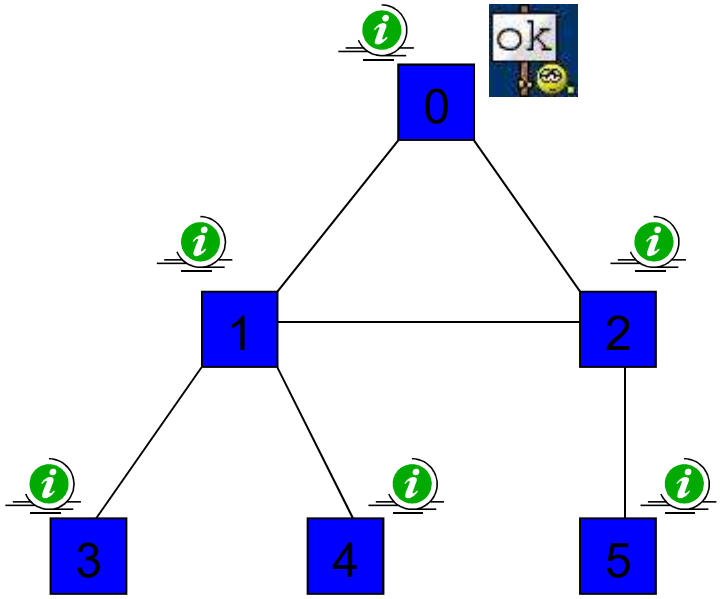

Nós 3, 4 e 5 recebem a informação e como são folhas, a retornam para seus pais.

# Propagação de Informações com Realimentação - Execução




Nós 1 e 2 receberam o retorno de seus filhos e podem informar ao seus pais

# Propagação de Informações com Realimentação - Execução




Nó 0 recebeu o retorno de todos os seus filhos e pode concluir que todos os nós do sistema já receberam a informação.

# Referências


- Valmir C. Barbosa, An Introduction to Distributed Algorithms, MIT Press, 1996
- http://www.ic.uff.br/~lucia/
- http://cs.ucsb.edu/~hnielsen/cs140/openmpi-install.html
- http://pt.wikipedia.org/wiki/Message_Passing_Interface