Sistem Terdistribusi

TIK-604

MPI

Kuliah 08: 15 s.d 17 April 2019

Husni

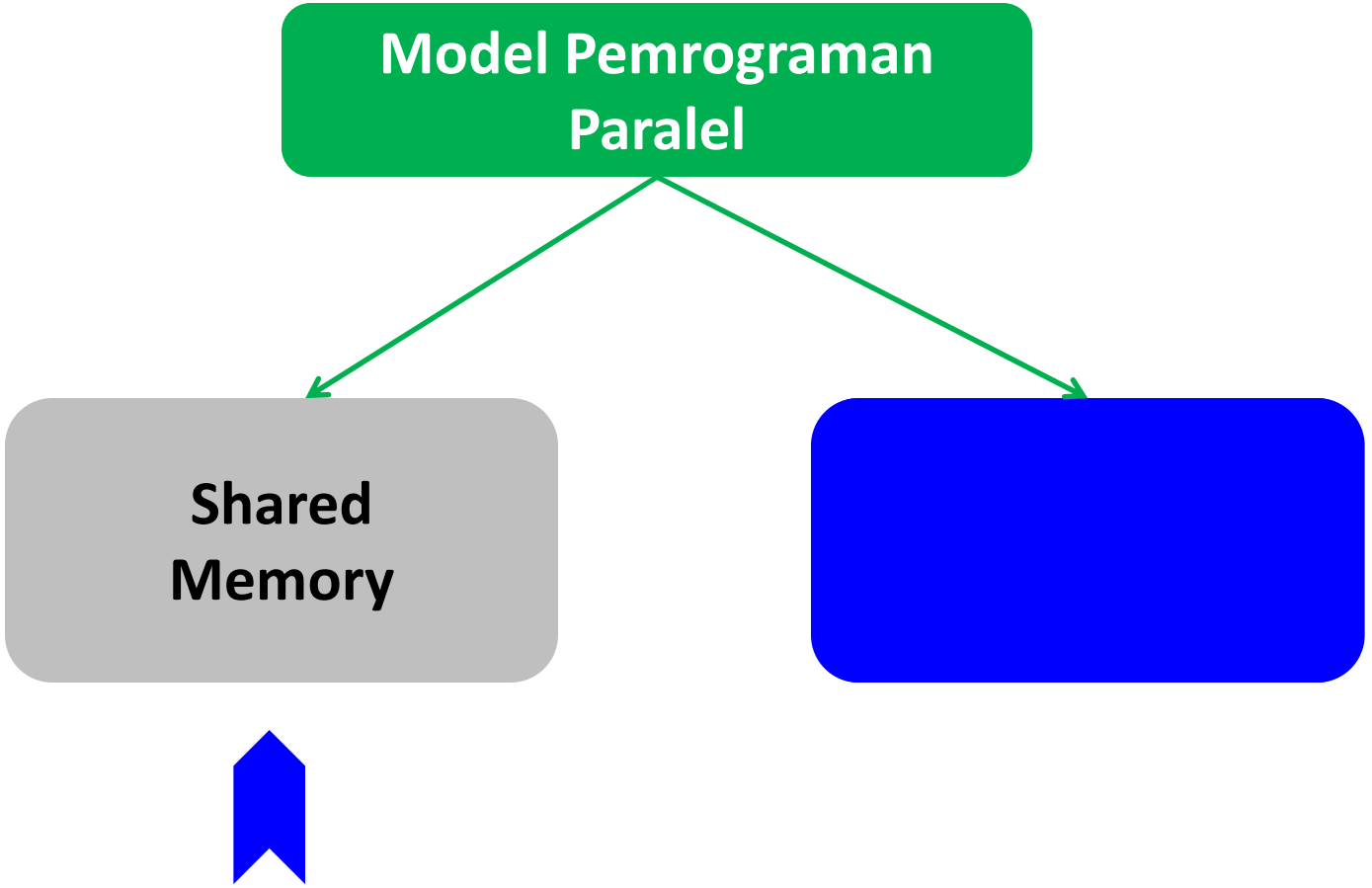
Hari ini

- Pertemuan Terakhir:
  - Eksklusi Mutual Terdistribusi
  - Algoritma Pemilihan
- Pembahasan hari ini:
  - Model-model Pemrograman: MPI
- Pengumuman:
  - ….

Model Pemrograman Paralel

- Apa itu model pemrograman Paralel?
  - Adalah suatu *abstraksi* yang disediakan oleh system bagi programmer sehingga mereka dapat menggunakannya untuk mengimplementasikan algoritma-algoritmanya
  - Model ini menentukan bagaimana programmer *dengan mudah* dapat menerjemahkan algoritma mereka ke dalam unit-unit parallel dari komputasi (yaitu task-task)
  - Model menentukan bagaimana *secara efektif* task-task paralel dapat dieksekusi pada sistem tersebut

Model Pemrograman Paralel Tradisional



Model Shared Memory

- Dalam model pemrograman shared memory, abstraksi yang disediakan tersebut mengimplikasikan bahwa tugas-tugas parallel dapat mengakses suatu lokasi dari memorynya

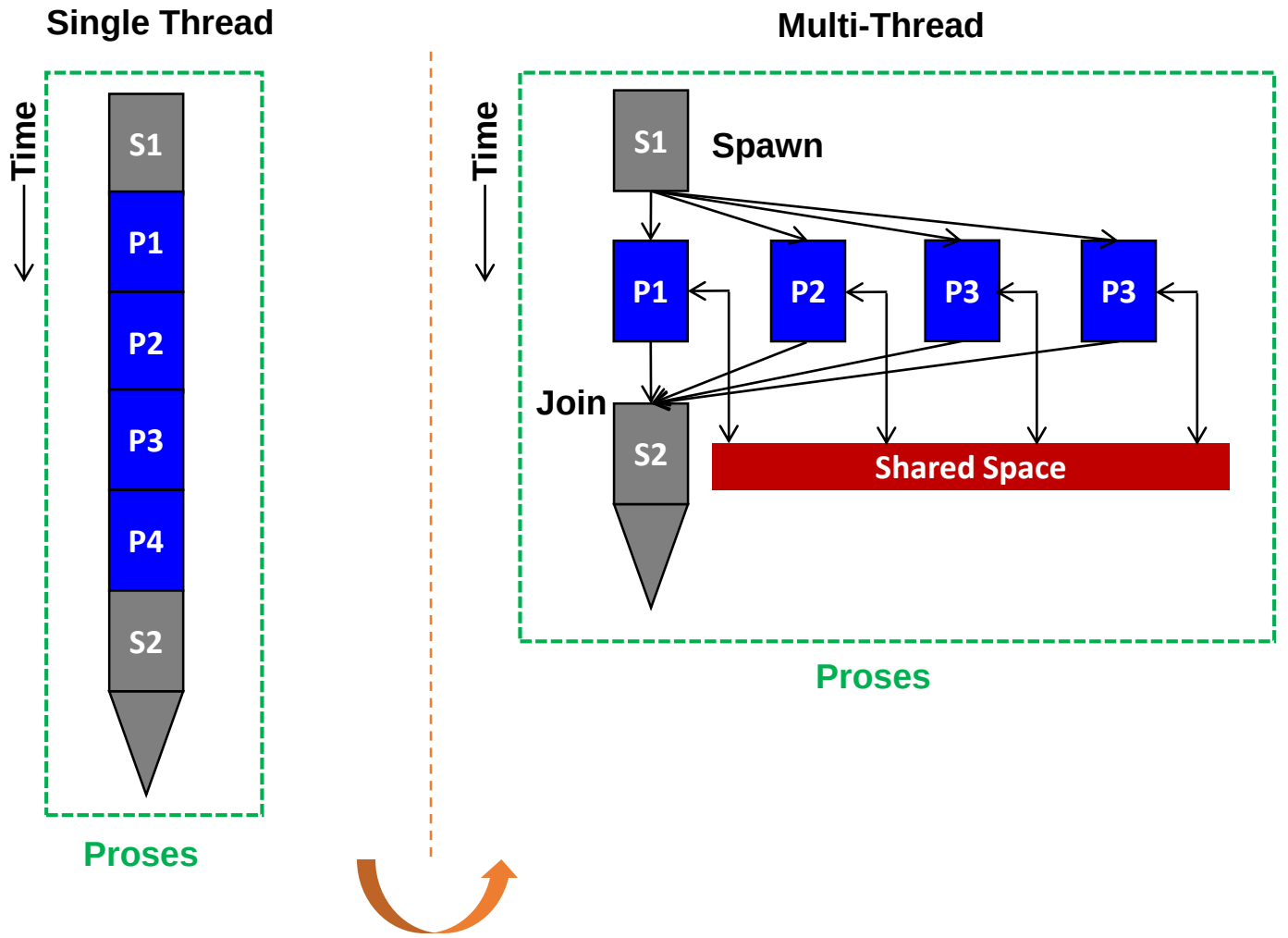
- Sesuai dengan itu, tugas-tugas paralel dapat berkomunikasi melalui pembacaan (reading) dan penulisan (writing) lokasi memory bersama (umum)

- Ini mirip dengan thread dalam suatu proses tunggal (dalam OS tradisional), yang berbagi-pakai (share) suatu ruang alamat tunggal

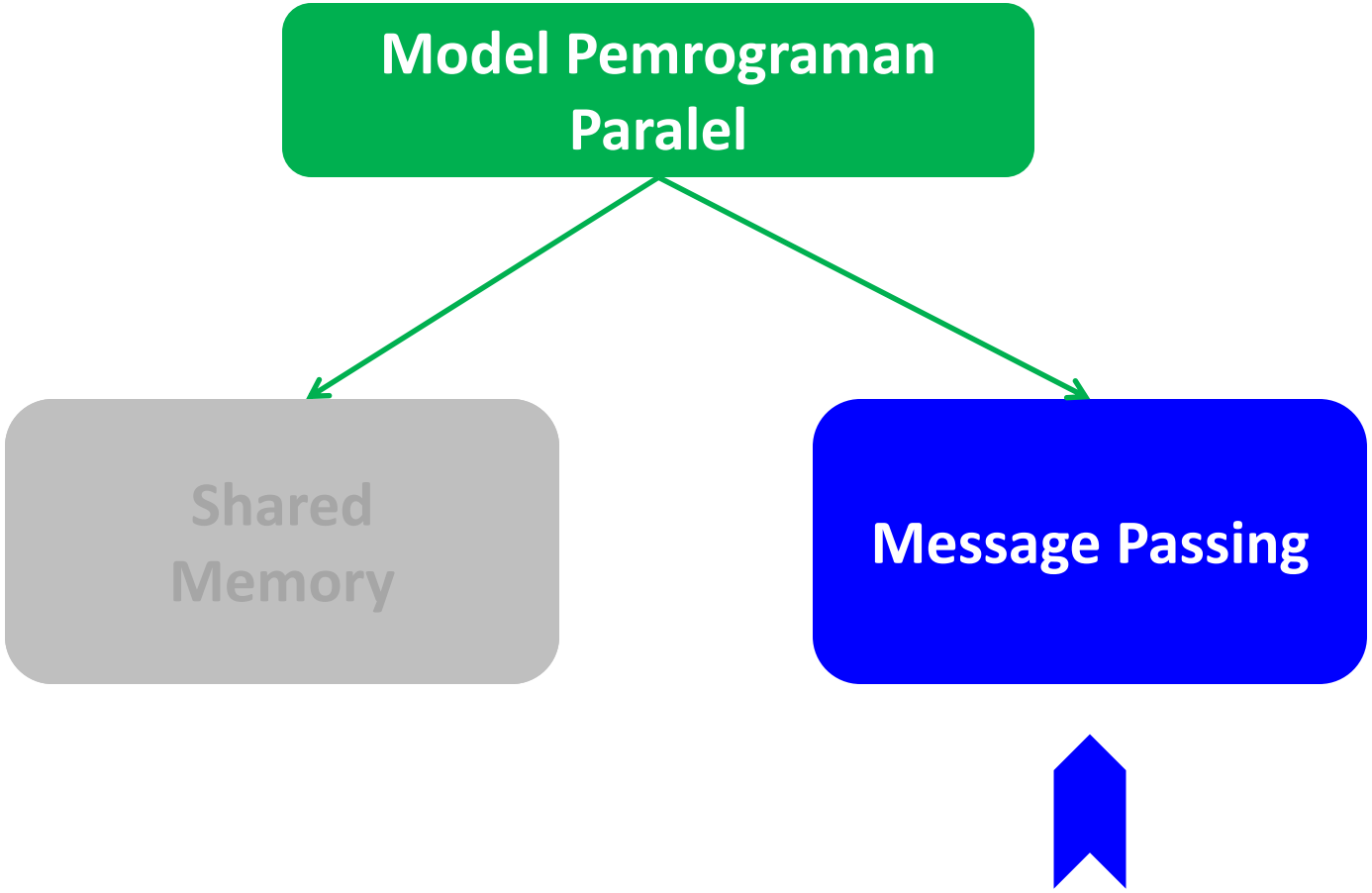
- Program berthread-banyak (multi-threaded, seperti program OpenMP) menggunakan model pemrograman shared memory ini.

Model Shared Memory

S_i = *Serial*
P_j = *Paralel*



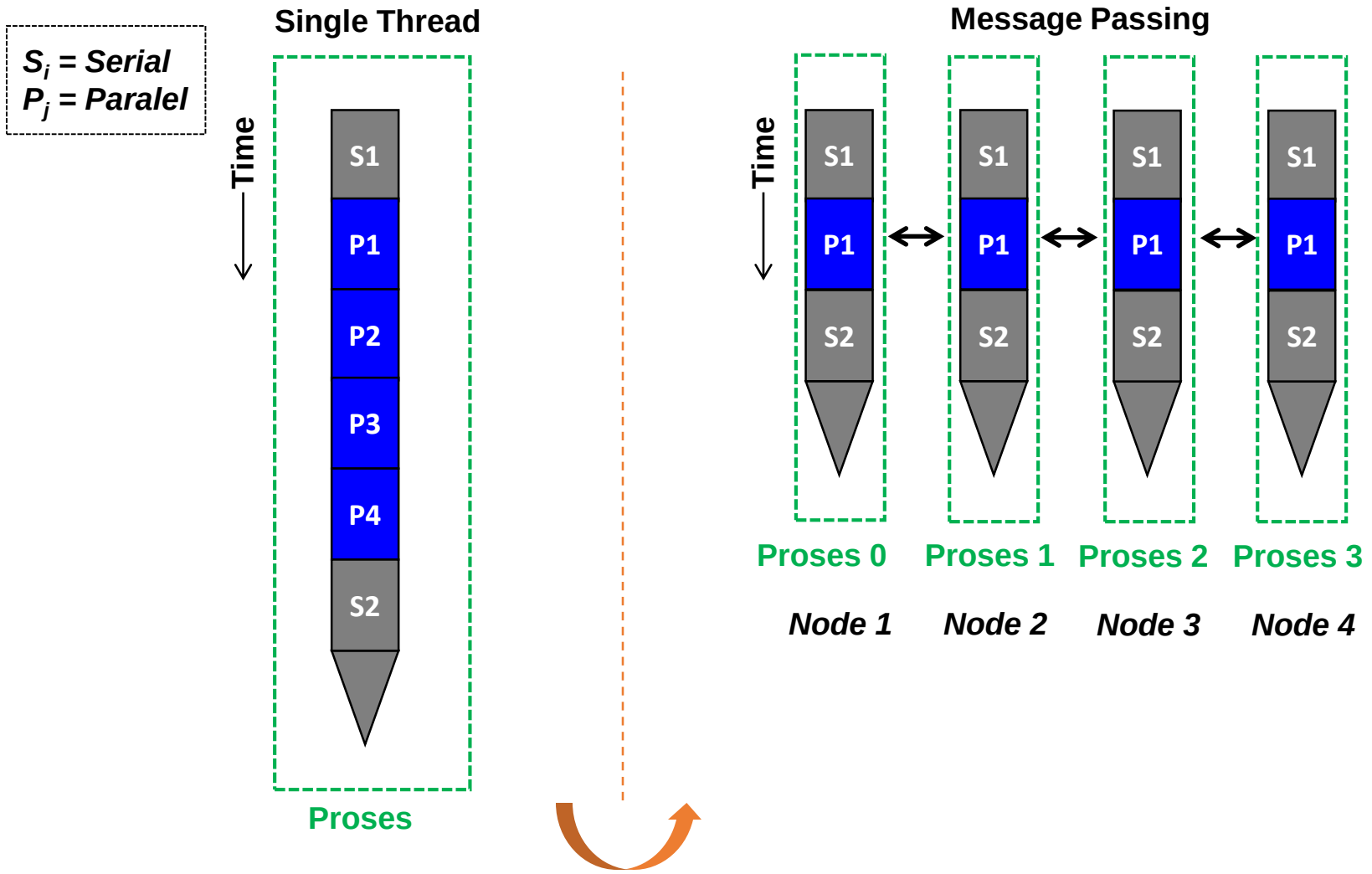
Model Pemrograman Paralel Tradisional



Model Message Passing

- Dalam message passing, setiap tugas parallel mempunyai memory lokalnya sendiri
- Satu task tidak dapat mengakses memory task lainnya
- Karena itu, task-task harus bersandar pada message passing eksplisit untuk berkomunikasi
- Ini serupa dengan abstraksi dari proses dalam suatu SO tradisional, yang tidak berbagi (share) suatu ruang alamat (address space)
- Contoh: Message Passing Interface (MPI)

Model Message Passing



Shared Memory vs. Message Passing

- Perbandingan antara model pemrograman shared memory dan message passing dilhat dari beberapa aspek:

Aspek	Shared Memory	Message Passing
Komunikasi	Implicit (via loads/stores)	Explicit Messages
Sinkronisasi	Explicit	Implicit (Via Messages)
Dukungan Hardware	Typically Required	None
Upaya Pengembangan	Lower	Higher
Upaya Tuning	Higher	Lower

Message Passing Interface: MPI

- Kita akan focus pada MPI:
  - Definisi
  - Komunikasi Point-to-point (P2P)
  - Komunikasi kolekstif.

Message Passing Interface

- Kita akan fokus pada MPI:
  - Definisi
  - Komunikasi Point-to-point (P2P)
  - Komunikasi koleksyif.

Apa itu MPI?

- MPI merupakan suatu model message passing standard untuk pengembangan program message passing.
- Tujuan dari MPI adalah untuk mendirikan suatu pustaka yang *portable*, *efficient*, dan *flexible* bagi message passing
- Dengan sendirinya, MPI BUKANlah suatu library (pustaka), tetapi suatu spesifikasi yang harus dipenuhi oleh suatu pustaka MPI
- MPI bukan pula suatu standard IEEE atau ISO, tetapi secara fakta, menjadi *industry standard* untuk penulisan program message passing pada platform HPC.

Alasan Penggunaan MPI

Alasan	Deskripsi
Standarisasi	MPI hanya pustaka message passing yang dapat dianggap suatu standard. Didukung secara virtual pada semua platform HPC
Portability	Tidak ada kebutuhan untuk memodifikasi source code saat memporting aplikasi ke platform berbeda yang mendukung standard MPI
Peluang Kinerja	Implementasi vendor harus mampu mengeksploitasi fitur-fitur hardware natif untuk mengoptimalkan kinerja
Functionality (Kemampuan)	Lebih dari 115 rutin telah didefinisikan
Availability (Ketersediaan)	Variasi implementasi tersedia, baik domain vendor maupun publik.

Communicators dan Groups

- MPI menggunakan obyek bernama *communicators/groups* untuk mendefinisikan koleksi proses mana yang dapat saling berkomuniasi utuk menyelesaikan suatu masalah tertentu

- Sebagian besar rutin MPI mengharuskan kita menetapkan suatu communicator sebagai argumennya

- Communicator **MPI_COMM_WORLD** sering digunakan dalam pemanggilan sub-rutin komunikasi

- MPI_COMM_WORLD merupakan communicator pra-definisi yang menyertakan semua proses MPI

Ranks

- Di dalam suatu communicator, setiap proses mempunyai ID uniknya sendiri yang dirujuk sebagai *rank*, diberikan oleh system saat proses tersebut diinisiasi
- Rank kadang kala disebut *task ID*– rank-rank bersebelahan dan dimulai dari nol (*zero*)
- Rank digunakan olenh programmer untuk menentukan source dan destination dari messages
- Rank sering digunakan secara kondisional oleh program untuk mengontrol eksekusi (missal *if rank=0 do this / if rank=1 do that*)

Banyak Communicators

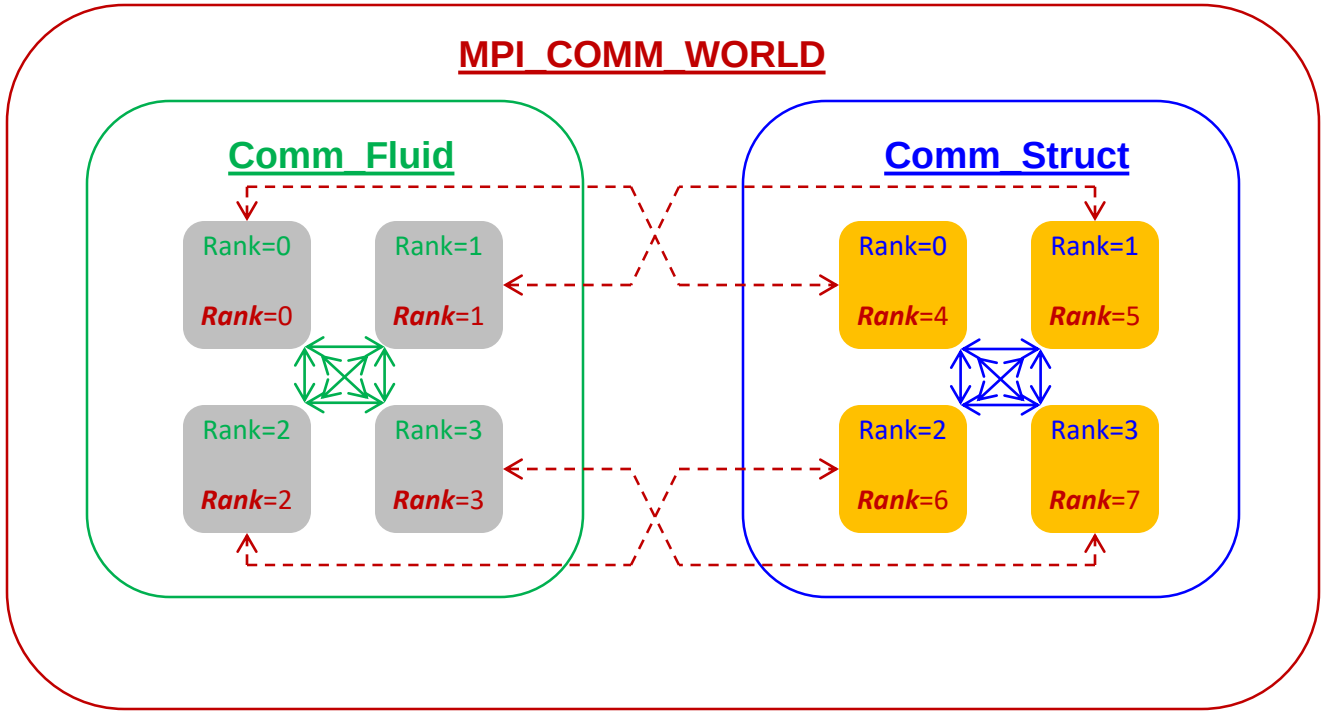
- Suatu masalah dapat terdiri dari beberapa sub-masalah dimana masing-masingnya dapat dipecahkan secara independen

- Kita dapat membuat suatu communicator baru bagi setiap sub-problem sebagai subset dari communicator yang telah ada (existing)

- MPI mengijinkan kita mewujudkan itu dengan menggunakan **MPI_COMM_SPLIT**

Contoh Banyak Communicators

- Perhatikan suatu masalah dengan suatu bagian dinamika fluida dan analisis struktural, dimana setiap bagian dapat dikomputasi secara paralel



- ✓ Rank-rank di dalam MPI_COMM_WORLD dicetak merah
- ✓ Rank-rank di dalam Comm_Fluid dicetak hijau
- ✓ Rank-rank di dalam Comm_Struct dicetak biru

Message Passing Interface:MPI

- Kita akan fokus pada MPI:
  - Definisi
  - Komunikasi Point-to-point (P2P)
  - Komunikasi kolektif.

Rutin Komunikasi Point-To-Point MPI

Rutin	*Signature*
Blocking send	int *MPI_Send*(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
Non-blocking send	int *MPI_Isend*(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
Blocking receive	int *MPI_Recv*(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
Non-blocking receive	int *MPI_Irecv*(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Message Passing Interface: MPI

- Kita akan fokus pada MPI:
  - Definisi
  - Komunikasi Point-to-point (P2P)
  - Komunikasi kolektif

Komunikasi Kolektif

- Komunikasi kolektif memungkinkan pertukaran data antar suatu kelompok proses
- Harus mencakup ***semua*** proses dalam lingkup suatu communicator
- Argumen communicator dalam suatu rutin komunikasi kolektif harus menentukan proses-proses mana yang dilibatkan dalam komunikasi
- Karena itu, adalah merupakan tanggungjawab programmer untuk memastikan bahwa semua proses di dalam suatu communicator berpartisipasi dalam suatu operasi kolektif.

Pola Komunikasi Kolektif

- Ada beberapa pola dari komunikasi kolektif:

1. *Broadcast*
2. *Scatter*
3. *Gather*
4. *Allgather*
5. *Alltoall*
6. *Reduce*
7. *Allreduce*
8. *Scan*
9. *Reducescatter*

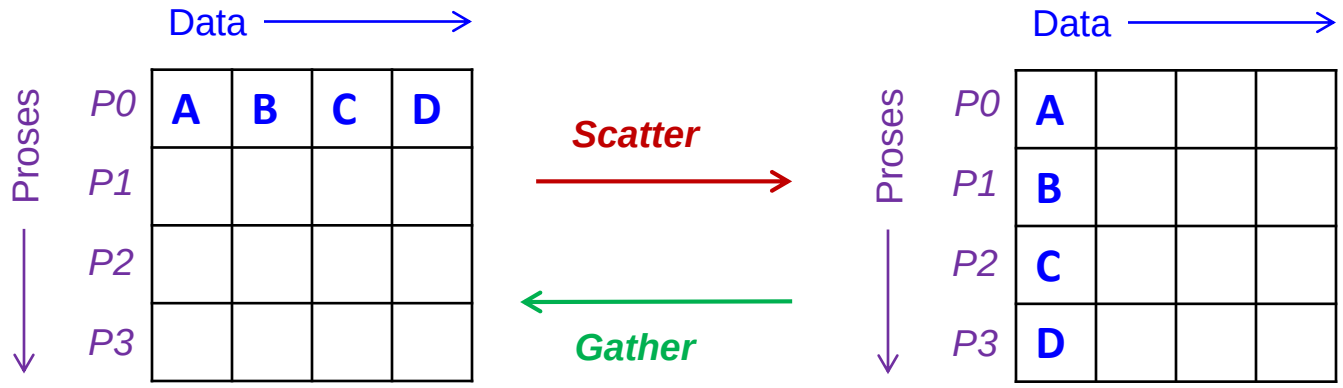
1. Broadcast

- *Broadcast* mengirimkan suatu message dari proses dengan rank *root* ke semua proses lain di dalam grup

int **MPI_Bcast** (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

2-3. Scatter dan Gather

- *Scatter* mendistribusikan message-message berbeda dari suatu source task tunggal ke setiap task di dalam grup
- *Gather* mengumpulkan message-message berbeda dari setiap task di dalam grup ke suatu destination task tunggal



int MPI_Scatter (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)

int MPI_Gather (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

4. All Gather

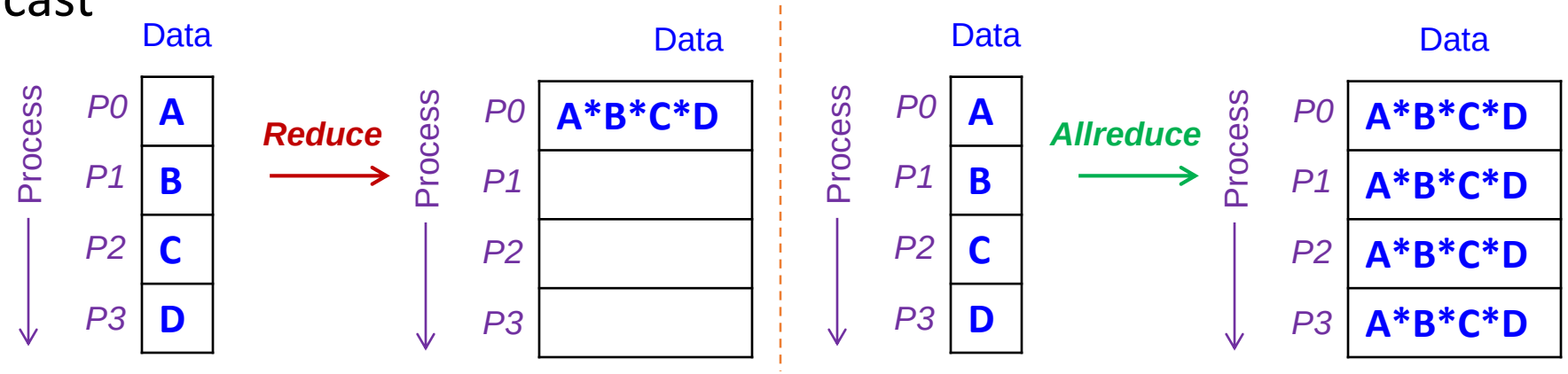
- *Allgather* mengumpulkan data dari semua tasks dan mendistribusikannya ke semua task
  - Setiap task di dalam grup, efeknya, mengerjakan suatu operasi broadcasting one-to-all di dalam grup tersebut.


```
int MPI_Allgather ( void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf,
                    int recvcount, MPI_Datatype recvtype, MPI_Comm comm )
```

6-7. Reduce dan All Reduce

- *Reduce* menerapkan suatu operasi reduksi pada semua task di dalam grup dan menempatkan hasilnya dalam satu task

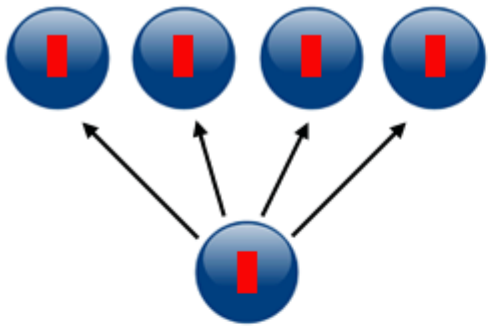
- *Allreduce* menerapkan operasi reduksi dan menempatkan hasilnya dalam semua task di dalam grup. Ini ekuivalen dengan MPI_Reduce diikuti oleh suatu MPI_Bcast



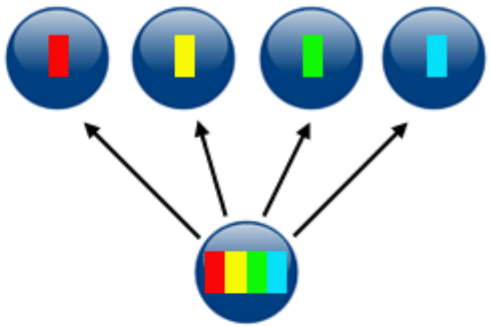
int MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

int MPI_Allreduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

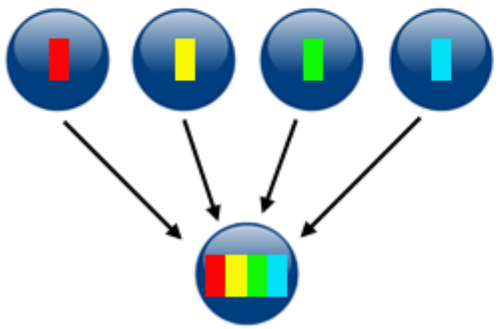
Recap



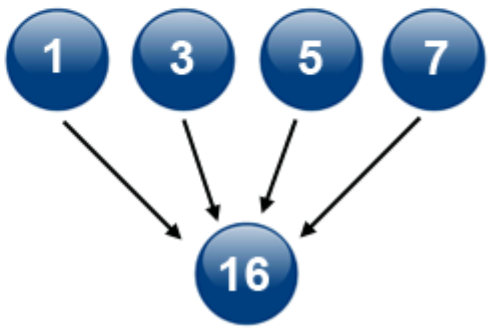
broadcast



scatter



gather



reduction

Contoh: PageRank

- PageRank adalah suatu fungsi yang memberikan suatu bilangan ril untuk setiap halaman di dalam Web

- Intuisi: semakin tinggi PageRank dari suatu page, makin “penting” page tersebut

- Simulasi dari *random surfers* memungkinkan memperkirakan dugaan intuitif dari “kepentingan” halaman web
  - Random surfers bermula pada halaman acak dan cenderung untuk berkerumun pada halaman yang penting (important pages)
    - Pages dengan jumlah surfers lebih besar adalah lebih “penting” daripada pages dengan jumlah surfers lebih kecil

Contoh: PageRank

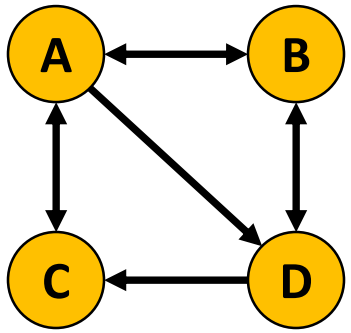
- Web dapat direpresentasikan sebagai suatu graf berarah, dengan halaman sebagai node dan link antar halaman sebagai edges

- Suatu random surfer bermula di A, selanjutkan akan berada di A, B, C, dan D dengan probabilitas masing-masing 0, 1/3, 1/3, dan 1/3

- Suatu random surfer bermula di B, selanjutnya akan berada di A, B, C dan D dengan probabilitas masing-masing ½, 0, 0, dan ½.

Contoh: PageRank

- Web dapat direpresentasikan sebagai suatu graf berarah, dengan halaman sebagai nodes dan links antar halaman sebagai edges



$$M = \begin{array}{c} \begin{matrix} A & B & C & D \end{matrix} \\ \begin{bmatrix} 0 & ½ & 1 & 0 \\ 1/3 & 0 & 0 & ½ \\ 1/3 & 0 & 0 & ½ \\ 1/3 & ½ & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

Suatu *transition matrix* dari Web, yang mendeskripsikan apa yang terjadi terhadap random surfers setelah *satu langkah*

Contoh: PageRank

- Distribusi peluang bagi lokasi dari suatu random surfer dapat dideskripsikan oleh suatu *column vector* (katakan, ***v***) yang mempunyai elemen ke-***j*** adalah peluang yang surfer tersebut pada halaman ***j***
  - Peluang ini adalah fungsi PageRank (yang diidealkan)

- Kita dapat memulai suatu random surfer pada semabarang dari 4 halaman dari graf Web kecil kita, degan probabilitas yang *sama*

$$\mathbf{v} = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \begin{matrix} A \\ B \\ C \\ D \end{matrix}$$

Contoh: PageRank

- Jika **M** adalah matriks transisi dari Web, setelah satu langkah, distribusi dari surfer akan menjadi **Mv**

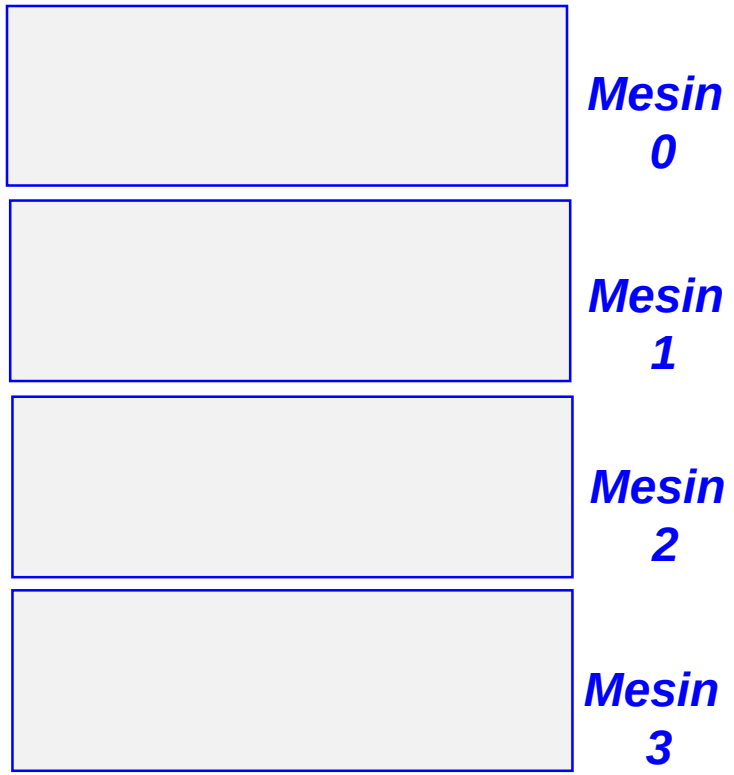
$$\begin{bmatrix} 0 & ½ & 1 & 0 \\ 1/3 & 0 & 0 & ½ \\ 1/3 & 0 & 0 & ½ \\ 1/3 & ½ & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

- Setelah dua langkah akan menjadi **M**(**Mv**) = $\mathbf{M^2v}$, dan seterusnya
  - Secara umum, mengalikan vektor awal **v** dengan **M** total ***i*** kali akan memberikan distribusi dari surfer setelah ***i*** langkah.

Contoh: PageRank

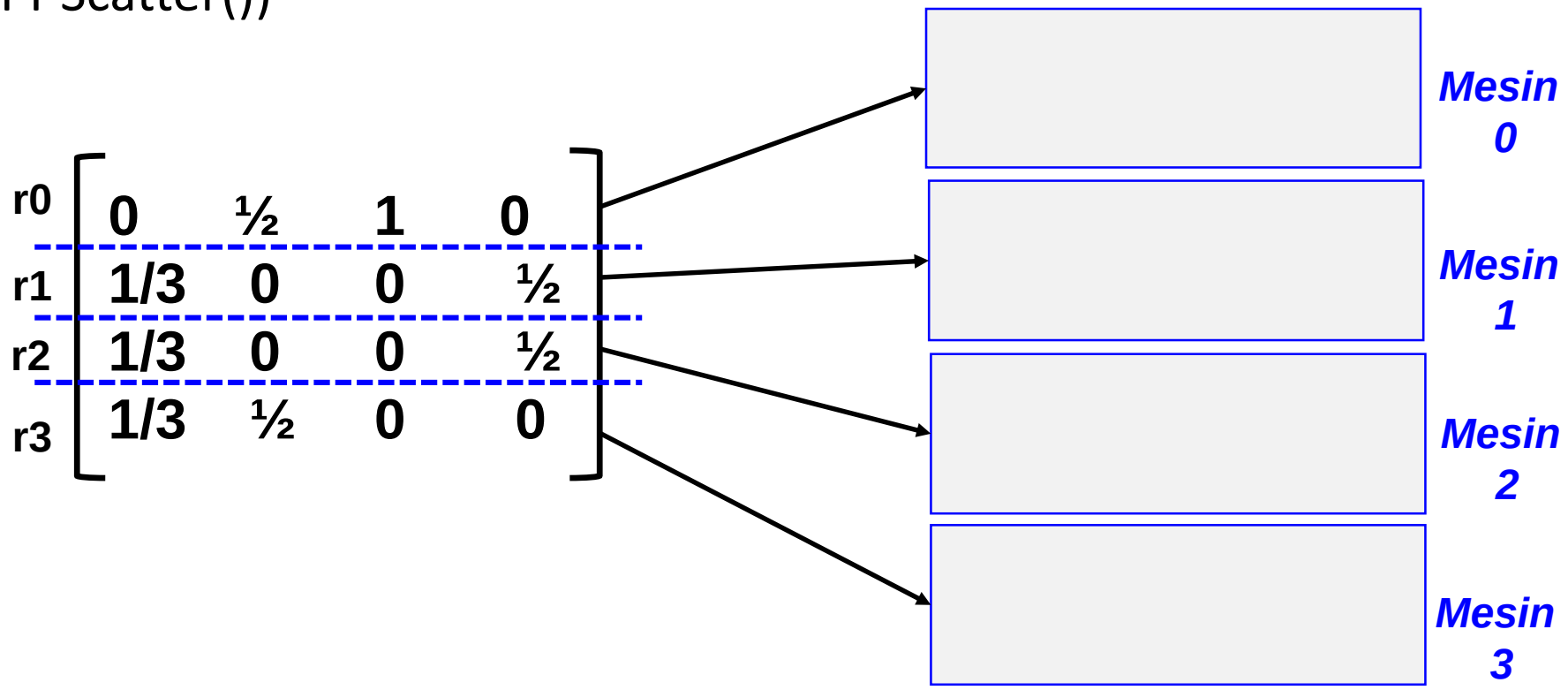
- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Pertama, partisi **M** pada master

r0	0	½	1	0
r1	1/3	0	0	½
r2	1/3	0	0	½
r3	1/3	½	0	0



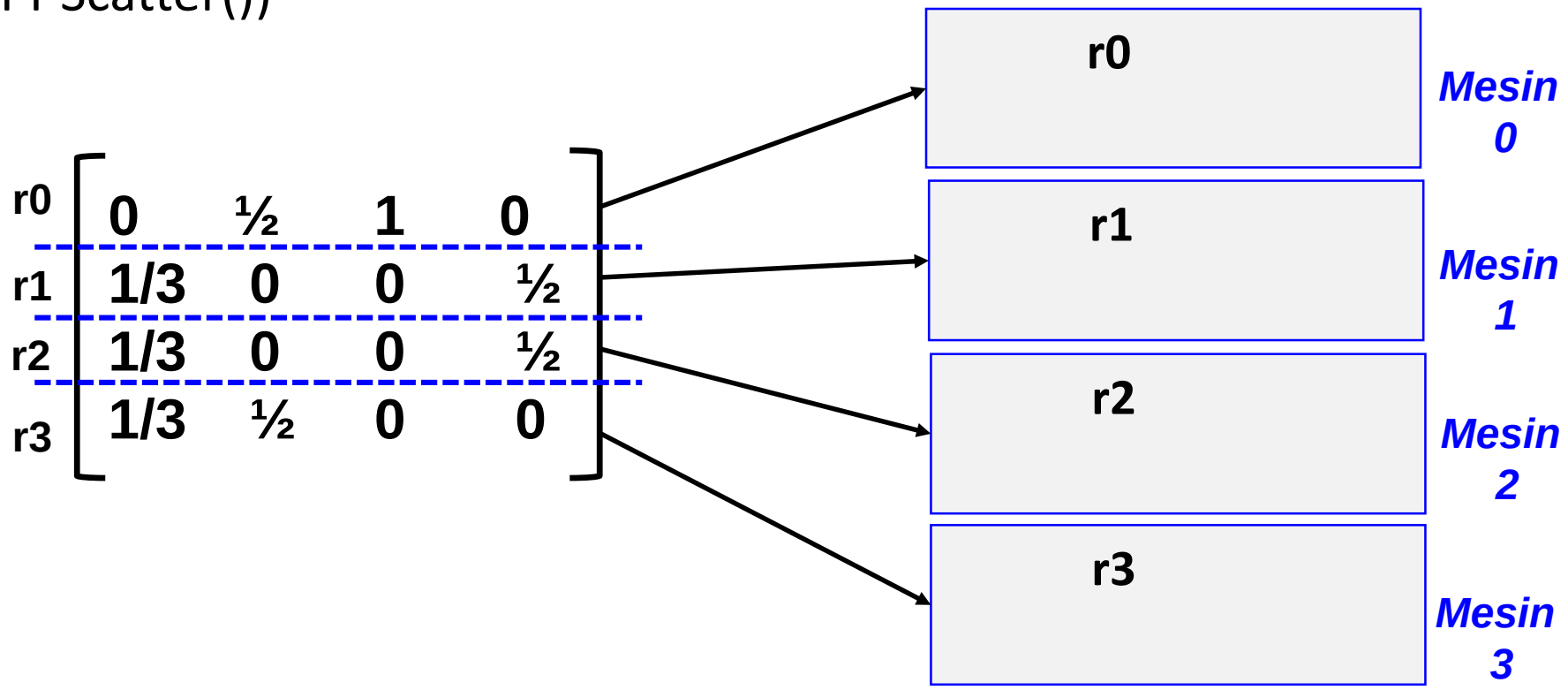
Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Kedua, distribusikan partisi-partisi **M** ke mesin-mesin (misal menggunakan MPI-Scatter())



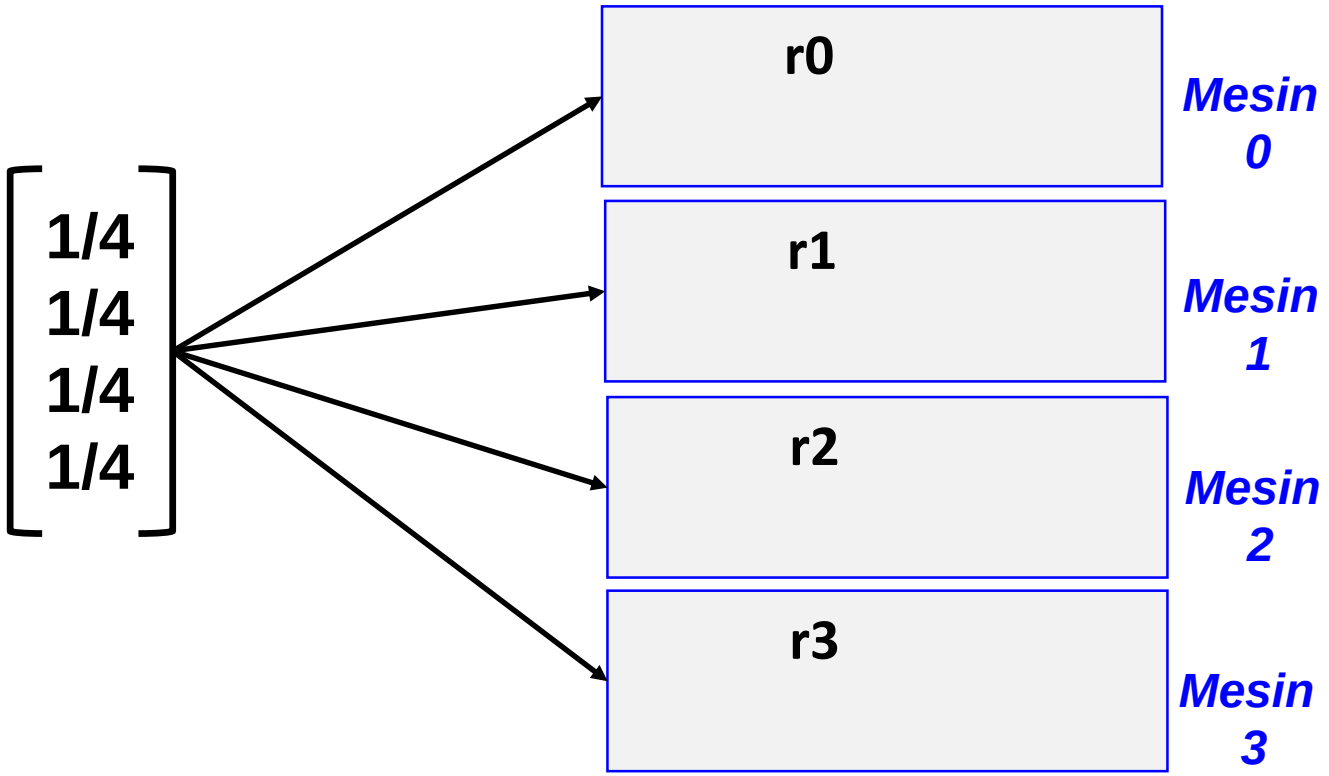
Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Kedua, distribusikan partisi-partisi **M** ke mesin-mesin (misal menggunakan MPI-Scatter())



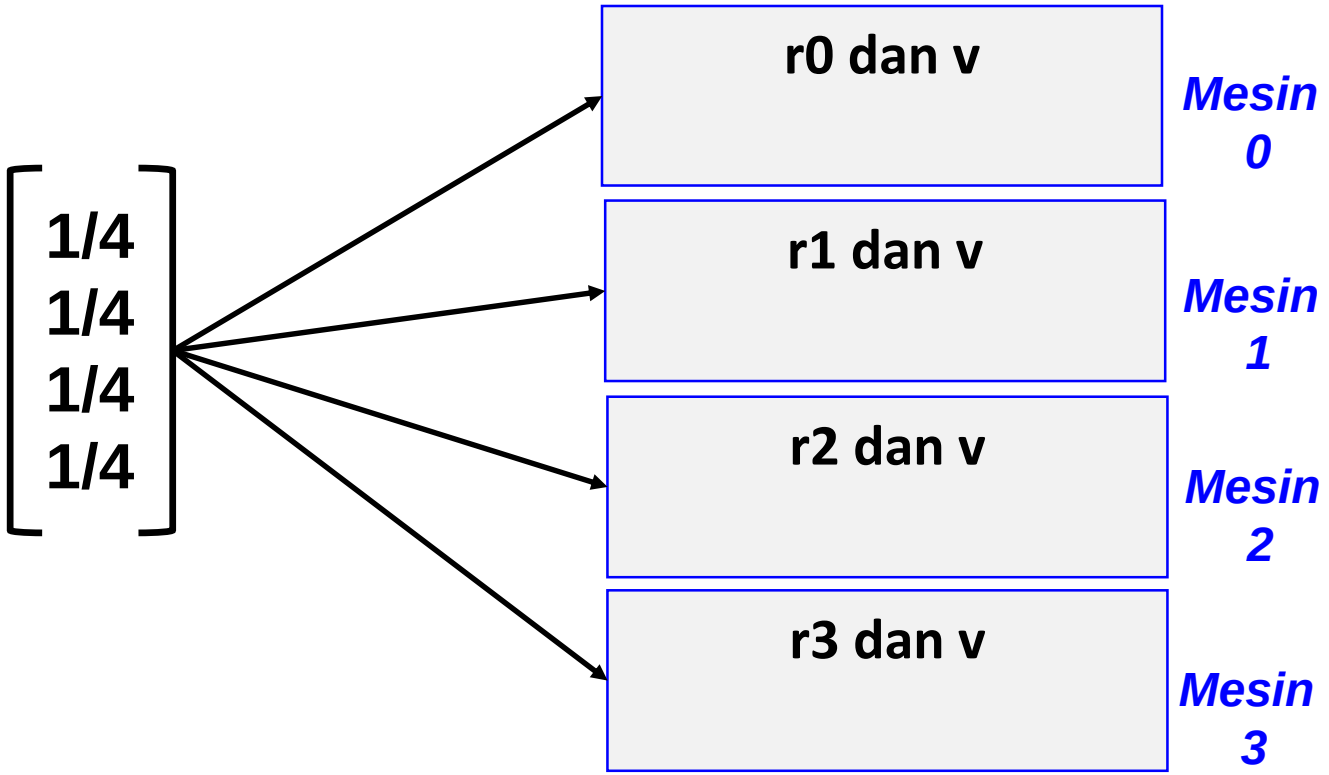
Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Ketiga, salin **v** ke setiap mesin (misal menggunakan MPI-Bcast())– *Iterasi 1 bermula di sini*



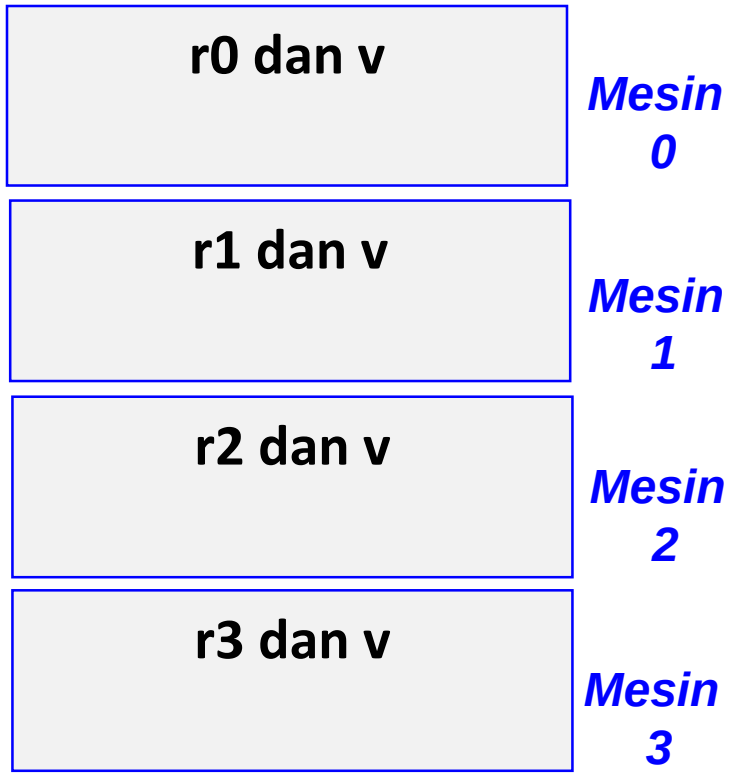
Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Ketiga, salin **v** ke setiap mesin (misal menggunakan MPI-Bcast())– *Iterasi 1 bermula di sini*



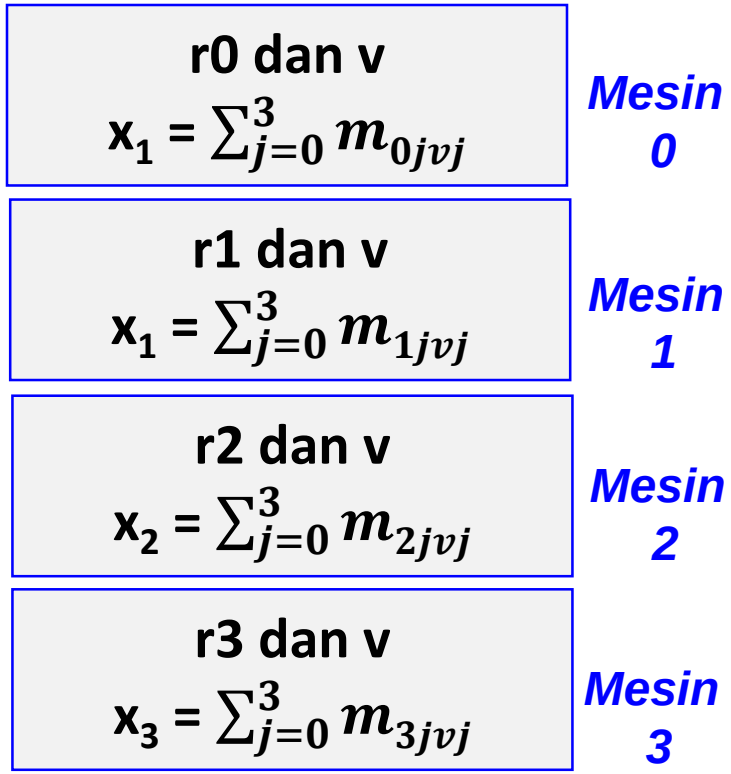
Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Keempat, terapkan logic perkalian **Mv** pada setiap mesin



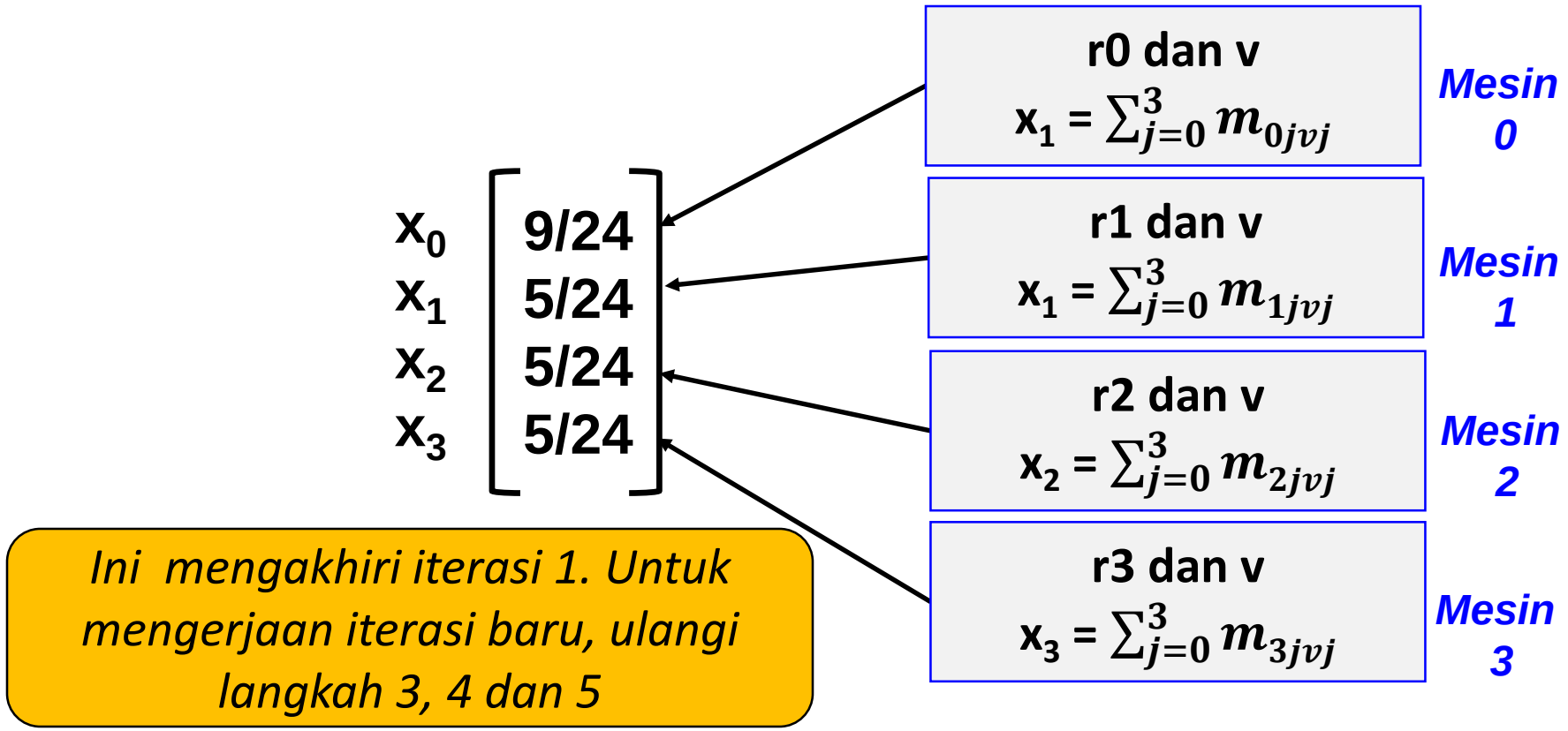
Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Keempat, terapkan logic perkalian **Mv** pada setiap mesin



Contoh: PageRank

- Bagaimana ini dapat diimplementasikan dalam MPI?
  - Kelima, setiap mesin mengirimkan balik elemennya ke master



Bahasa Selanjutnya...

- Caching (Replikasi sisi Client)