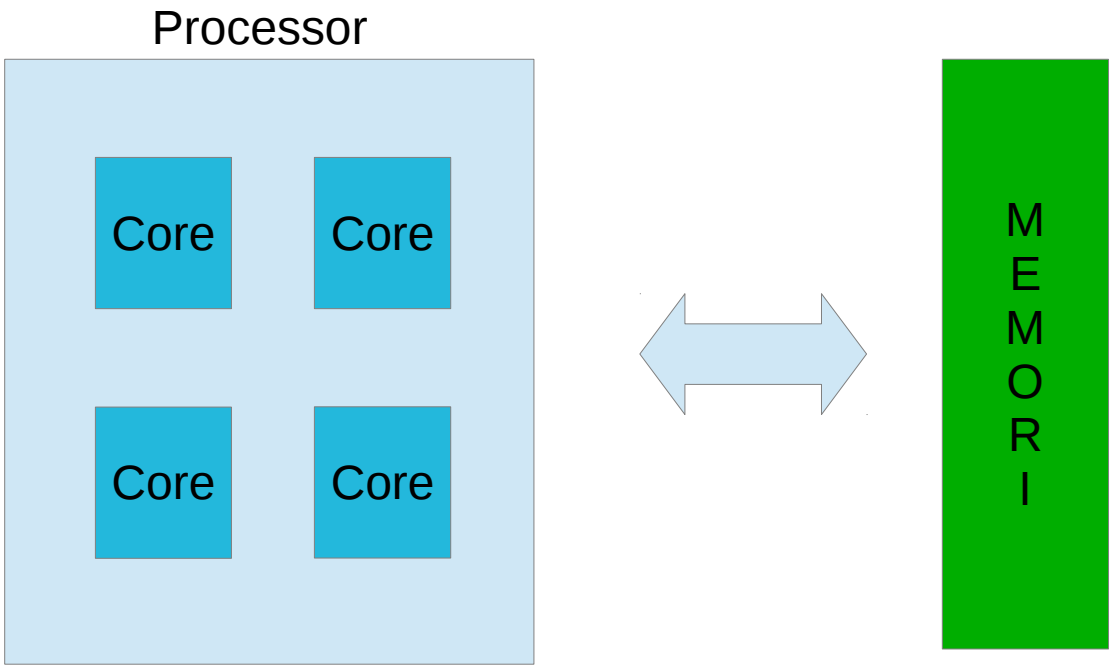
Komputasi Paralel

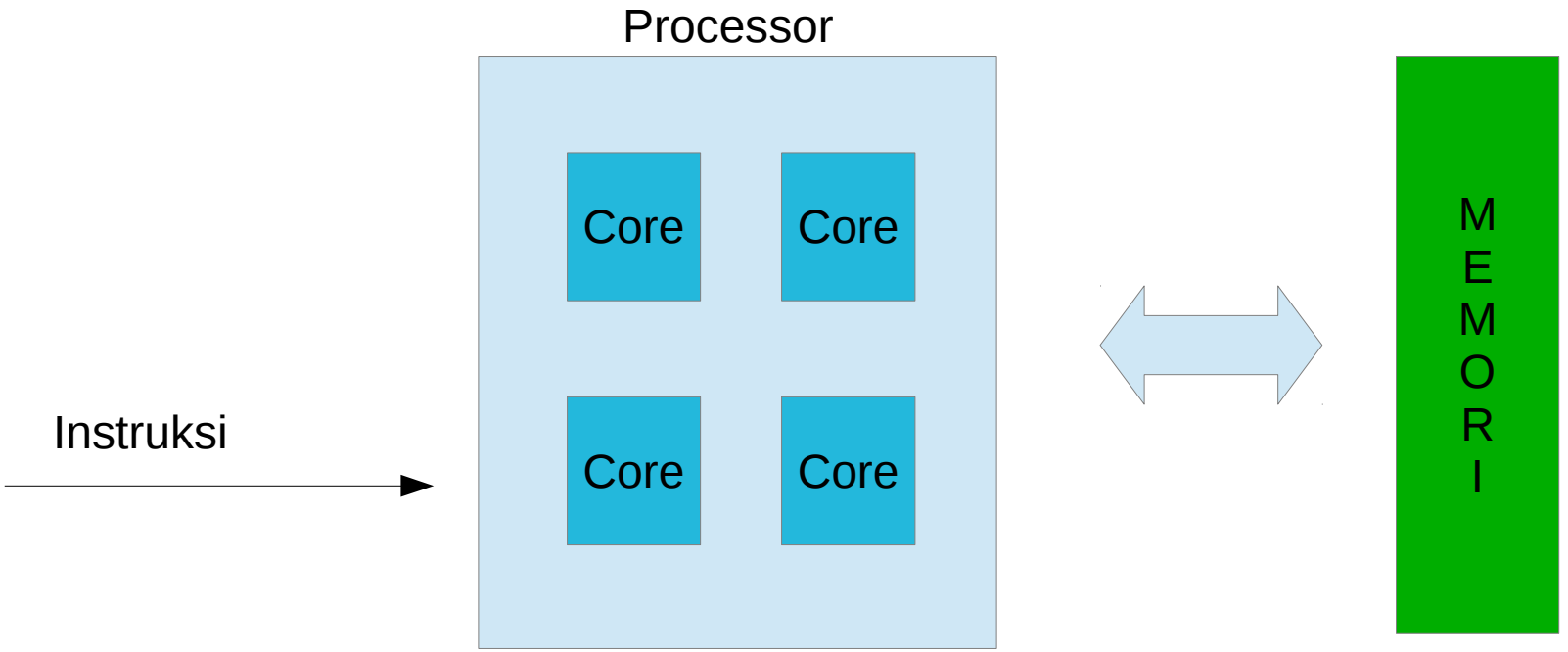
Ahmad Rio Adriansyah
arasy@nurulfikri.ac.id

Komputer Modern



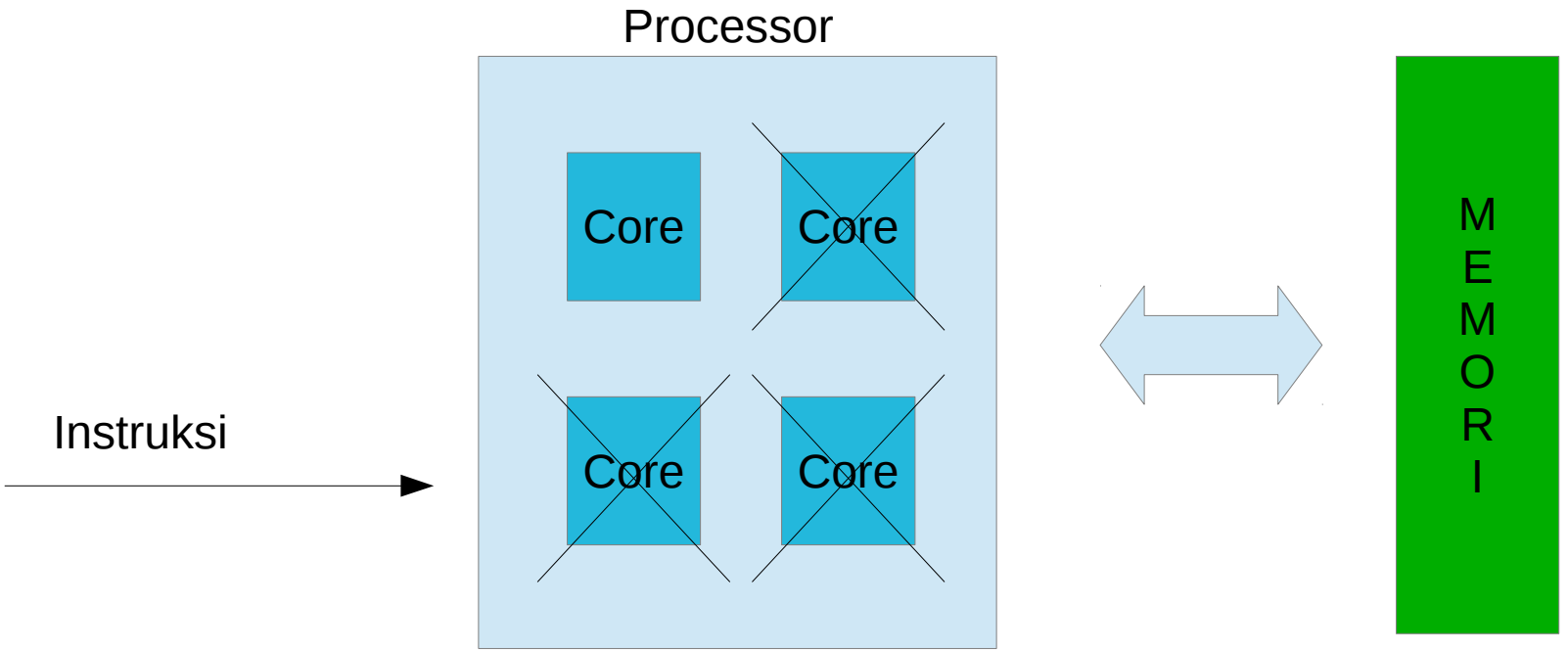
Program Sekuensial

*Program dijalankan



Program Sekuensial

*Yang bekerja hanya 1 core, yang lainnya idle



Open MP

Standard API untuk aplikasi paralel dalam C, C++, Fortran pada *shared memory computer*

API OpenMP mengandung :

- Compiler directives
- Runtime subroutines/functions
- Environment variables

Open MP

Kepanjangannya :

Singkat = Open Multi-Processing

Panjang = Open specification for Multi-Processing via collaborative work between interested parties from the hardware and software industry, government, and academia

Keunggulan Open MP

Open MP Menyediakan thread programming model pada high level language

User tidak perlu memikirkan detail pembuatan dan pemberian tugas ke thread

Simple : User membuat keputusan stategis, compiler yang akan mengatur detailnya

Kode serial dan paralel dalam satu source code

Perubahan pada bagian serial dari kode tidak terlalu banyak

Tujuan Open MP

1. Standardisasi : mendukung standard diantara banyak platform/arsitektur shared memory
2. Bermakna : directive dibuat sederhana dan terbatas. Paralelisme yang signifikan bisa diraih dengan hanya beberapa directive saja
3. Kemudahan : mendukung paralelisasi bertahap, juga implementasi coarse grained dan fine grained
4. Portabilitas : Sudah otomatis terdapat pada beberapa platform besar (Unix/Linux, Windows)

Open MP tidak ...

1. ...ditujukan untuk sistem paralel distributed memory (jika sendirian)
2. ...perlu diimplementasikan secara sama oleh semua vendor
3. ...menjamin bahwa penggunaan shared memorinya paling efisien
4. ...menyangkut paralelisasi otomatis (compiler generated) , tapi ada compiler directive
5. ...menjamin bahwa input dan output sinkron dengan sendirinya

User/programmer yang bertanggung jawab terhadap hal-hal tersebut

Sejarah

- Awal 90-an, para vendor mesin shared memory menyediakan ekstensi untuk pemprograman fortran yang mirip dan berbasis pada perintah (directive based)
  - Pengguna menambahkan program fortrannya dengan perintah yang spesifik mengatur loop mana yang diparalelisasikan
  - Compiler bertanggung jawab memparalelisasikan loop tersebut dalam mesin shared memory

Sejarah

- Implementasinya memiliki fungsi serupa, tapi lama kelamaan divergen
- Standard pertama yang diusulkan adalah draf ANSI X3H5 tahun 1994. Tidak pernah dijalankan karena distributed memory semakin populer. (Ketertarikan vendor memudar karena terbagi antara shared memory dan distributed memory)

Sejarah

- Tak berapa lama, shared memory menjadi umum dan para vendor tertarik kembali
- Spesifikasi standard OpenMP dimulai kembali tahun 1997, melanjutkan ANSI X3H5
- Standardisasinya dipimpin oleh OpenMP Architecture Review Board (ARB)

Member OpenMP ARB

Permanent Member :

AMD

Convey Computer

Cray

Fujitsu

HP

IBM

Intel

NEC

NVIDIA

Oracle Corporation

Red Hat

ST Microelectronics

Texas Instrument

Versi Rilis

Tahun	Versi
Oct 1997	Fortran 1.0
Oct 1998	C/C++ 1.0
Nov 1999	Fortran 1.1
Nov 2000	Fortran 2.0
Mar 2002	C/C++ 2.0
May 2005	OpenMP 2.5
May 2008	OpenMP 3.0
Jul 2011	OpenMP 3.1
Jul 2013	OpenMP 4.0
Nov 2015	OpenMP 4.5

Komponen API Open MP

Compiler Directive (44)

Runtime Library Routine (35)

Environment Variable (13)

Compiler Directive

- Muncul sebagai komentar pada source code kecuali jika menggunakan compiler flag tertentu
- Diantaranya digunakan untuk :
  - Memunculkan region paralel
  - Membagi blok kode antar thread
  - Mendistribusikan iterasi pada loop antar thread
  - Menserialkan bagian kode
  - Sinkronisasi antar thread

Runtime Library Routine

- Dapat digunakan dengan menambahkan header <omp.h>
- Diantaranya digunakan untuk :
  - Mengeset dan mengkueri informasi tentang thread dan team thread
  - Mengkueri region paralel dan levelnya
  - Mengeset, memulai, dan mengakhiri lock
  - Mengkueri waktu (wall clock time) dan resolusi

Environment Variable

- Untuk mengontrol eksekusi kode paralel pada saat runtime
- Diantaranya digunakan untuk :
    - Mengeset banyaknya thread
    - Membinding thread ke prosesor
    - Mengeset level nested paralelism
    - Mengeset stacksize, wait policy dari thread

Test – Hello Parallel World

Compiler mendukung Open MP?

```
#include <omp.h>
#include <stdio.h>
int main()
{
#pragma omp parallel
printf("hello from thread %d, nthread =
   %d\n",omp_get_thread_num(),omp_get_num_threads() );
}
```

Periksa Prosesor

Linux :

cat /proc/cpuinfo

lscpu

top -H

Windows :

msinfo32

wmic

Compiler

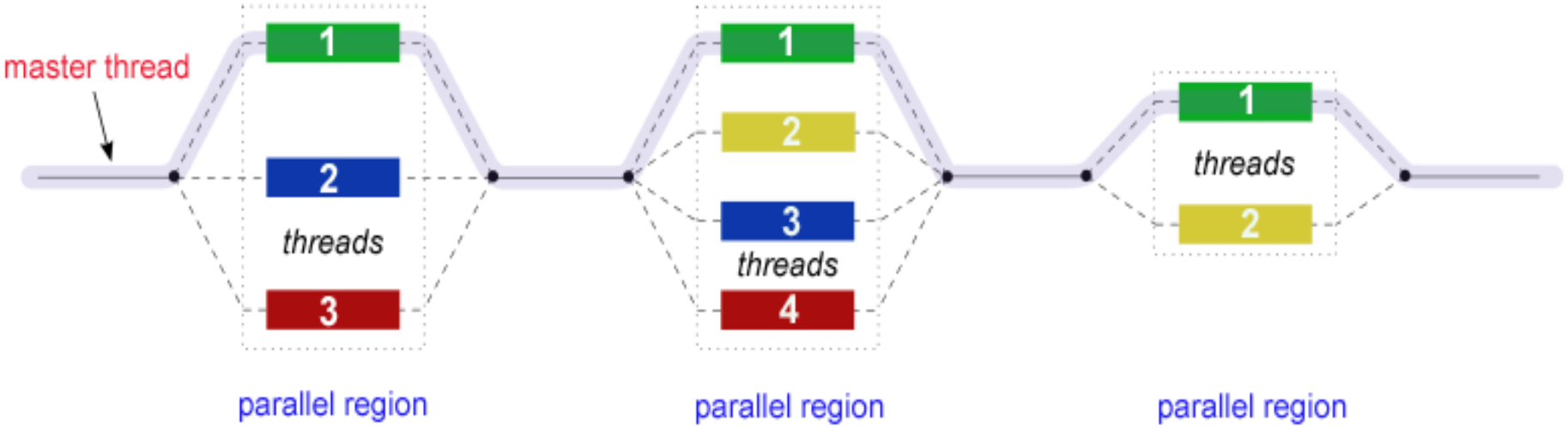
Vendor	Compiler	Flag
GNU	gcc	-fopenmp
Oracle	C/C++/Fortran	-xopenmp
Intel	C/C++/Fortran (10.1)	-Qopenmp (Windows) -openmp (Linux/MacOS)
IBM	bgcc_r , bgxlc_r , dll	-qsmp=omp
Portland Group Compiler and Tools	C/C++/Fortran	-mp
NAG	nagfor	-openmp
LLVM	clang	-fopenmp

http://openmp.org/wp/openmp-compilers/

Open MP pada IDE Code::Block / DevCPP

- Add **gomp** pada linker settings
  - Compiler settings > Linker settings > add **gomp**
- Add **-fopenmp** pada option compiler settings
  - Compiler settings > Other options > add **-fopenmp**

Fork-Join Model



Fork-Join Model

1. di awal program berjalan dengan sebuah thread (master thread/thread 0)
2. di permulaan blok paralel, master thread membuat tim berisi beberapa pekerja paralel, thread (FORK)
3. pernyataan di dalam blok paralel dikerjakan oleh tim thread secara bersamaan
4. di akhir blok paralel, semua thread melakukan singkronisasi dan bergabung kembali dengan master thread (JOIN)

Fork-Join Model

Task bisa dikerjakan oleh thread yang baru dibuat atau oleh thread yang sudah ada pada pool of threads (untuk mengurangi overhead)

Directive Untuk Blok Paralel

```
#pragma omp parallel [klausa]
{
}
```

Fungsi : Menyabangkan thread dan menggabungkannya kembali

Semua directive pada Open MP mengikuti format tersebut (#pragma omp [directive] [klausa])

Mengatur Jumlah Thread

```
omp_set_num_threads(int n);
```

Mengatur Jumlah Thread

omp_set_num_threads(int n);

Bisa diset melebihi jumlah core/prosesor yang dimiliki

Bisa juga diset menggunakan environmental variabel

export OMP_NUM_THREADS=n

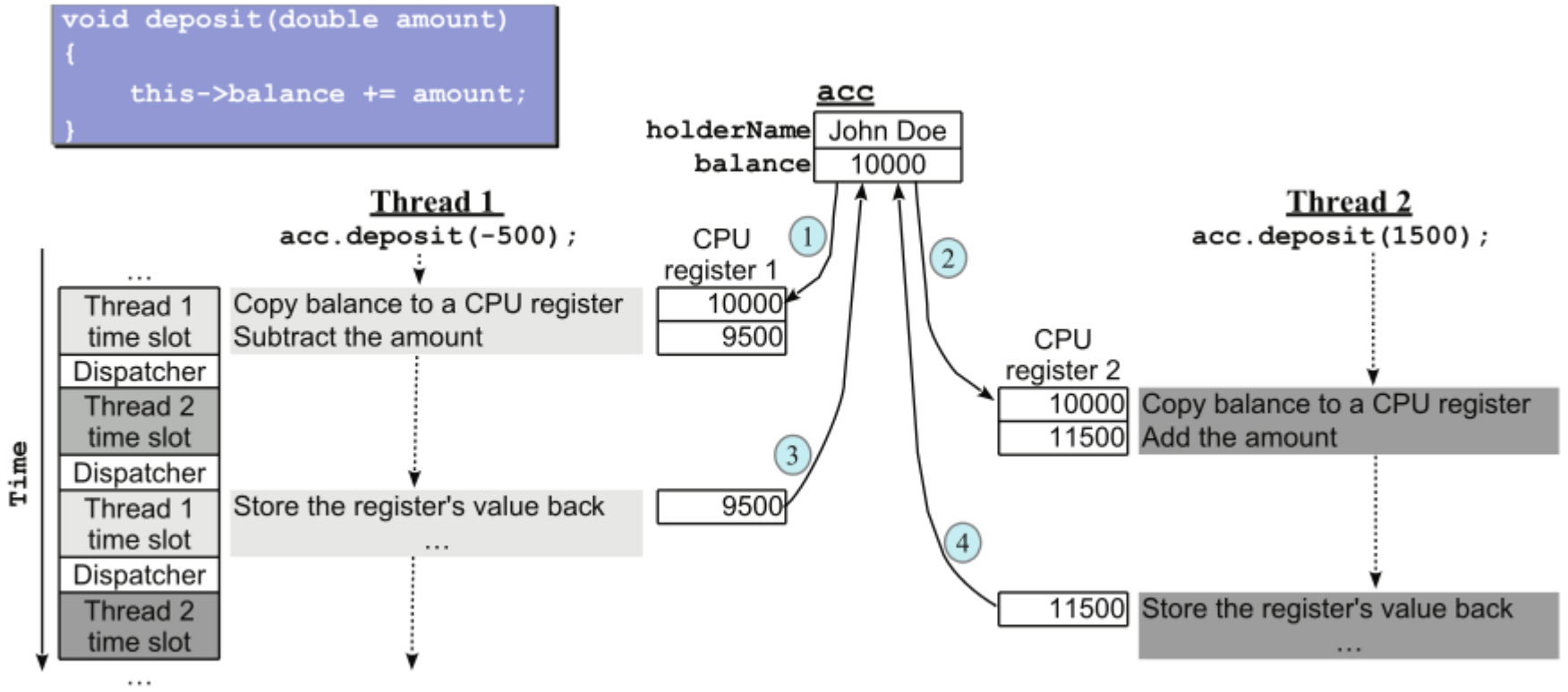
Race Condition

Kondisi pada komputer/elektronik dimana outputnya tergantung dengan timing atau urutan dari kejadian yang tidak dapat dikendalikan.

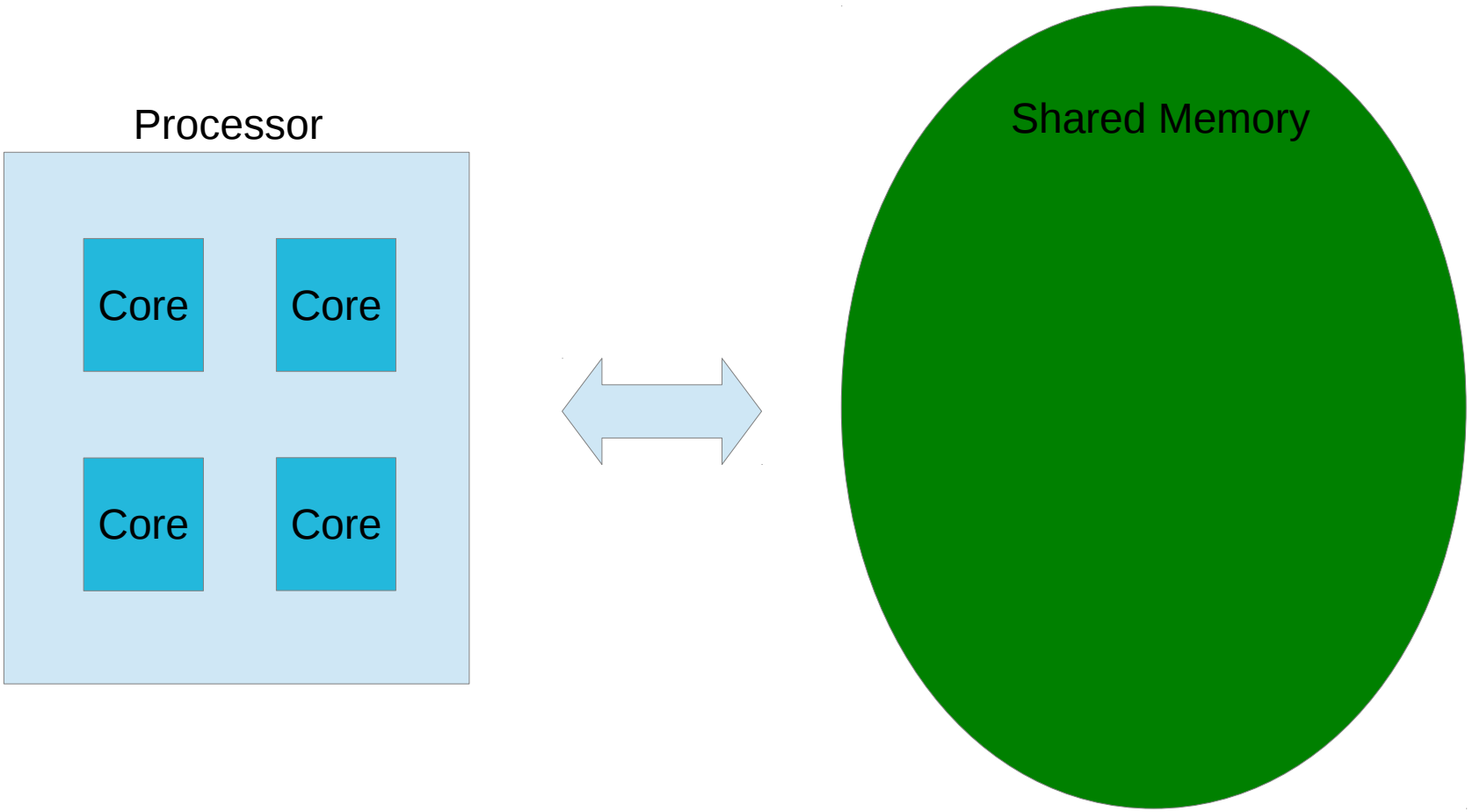
Urutan eksekusi thread saling bertumpukan (tidak beraturan)

Tantangannya : programmer harus bisa mengeluarkan hasil sesuai keinginan

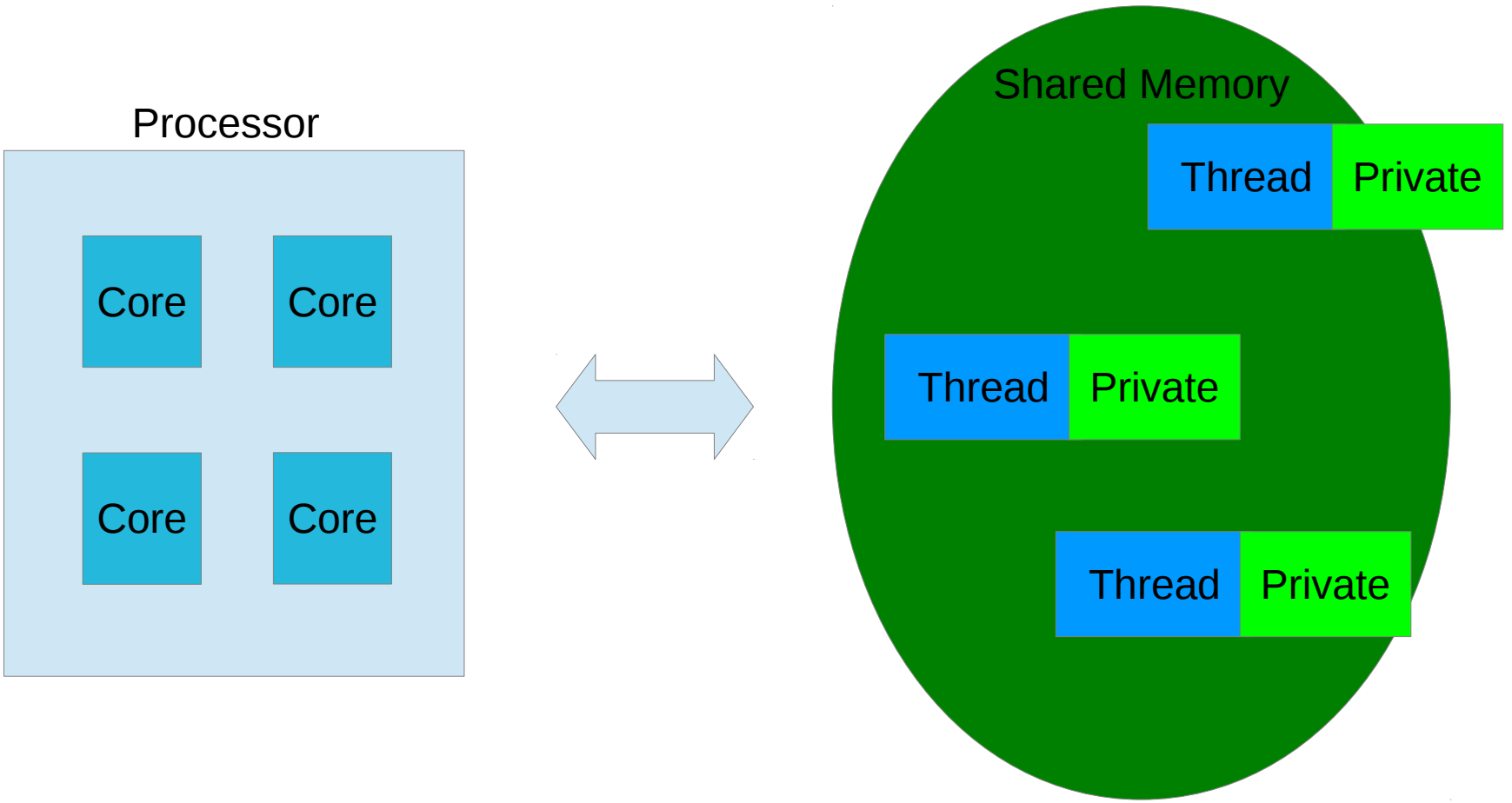
Race Condition



Thread



Thread


Processor
Core
Core
Core
Core
Shared Memory
Thread
Private
Thread
Private
Thread
Private

Variabel Shared dan Private

Tiap variabel yang dideklarasikan di luar blok #pragma dianggap shared

Variabel yang dideklarasikan di dalam blok #pragma adalah private

Defaultnya, hampir semua variabel pada OpenMP adalah shared dengan beberapa pengecualian seperti variabel indeks dan yang dideklarasikan di dalam blok paralel.

Bisa diatur dengan klausa pada blok paralelnya :

1. shared (list)

   semua variabel pada list dianggap shared, semua thread punya akses ke variabel tersebut

   #pragma omp parallel shared (a,b,c)

2. private (list)

   semua thread punya salinan yang bersifat private dari variabel pada list, tidak ada thread lain yang bisa mengakses variabel tersebut

   #pragma omp parallel private (a,b,c)

Hello Parallel World 2

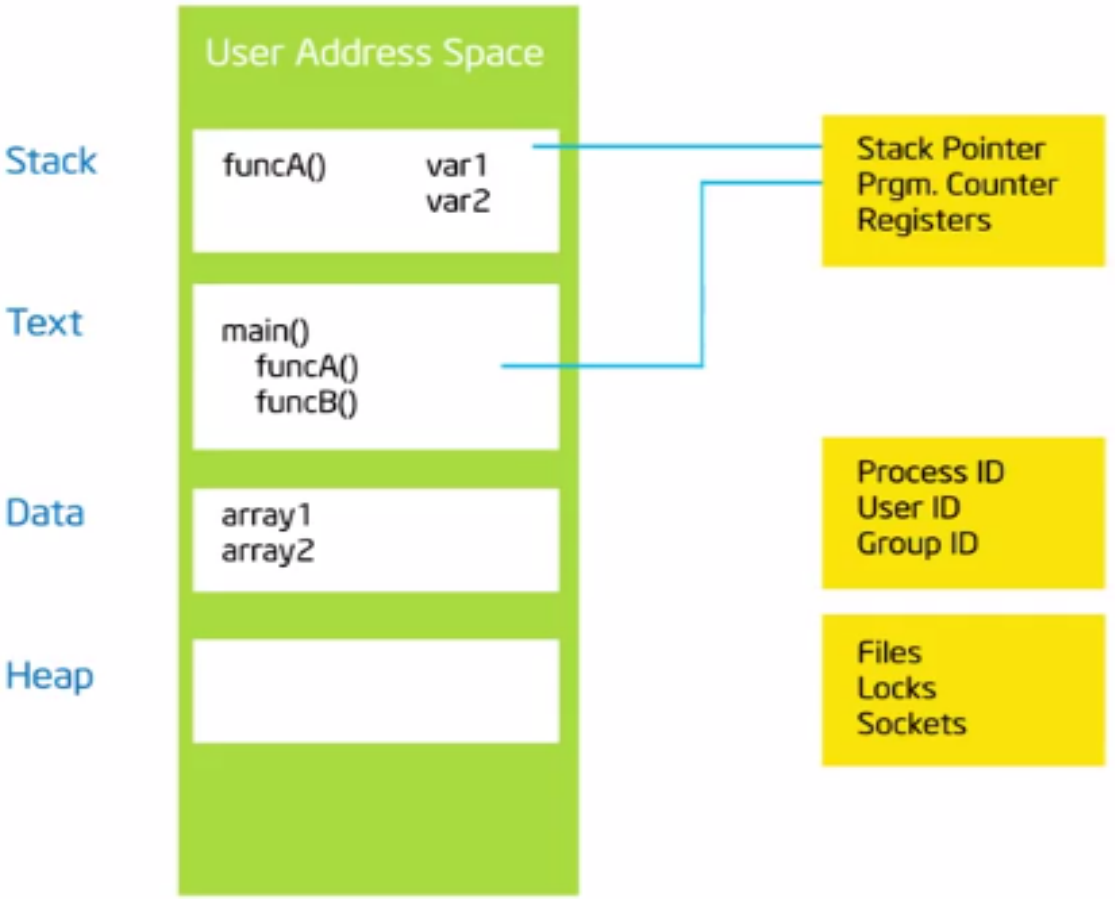
```
#include <omp.h>
#include <stdio.h>
int main()
{
int x = 0;
#pragma omp parallel
        {
        int id = omp_get_thread_num();
        int thread = omp_get_num_threads();
        x+=5;
        printf("hello from thread %d, nthread = %d : x is %d\n",id,thread,x );
        }
}
```

Hello Parallel World 2

tambahkan klausa shared(x) atau private(x) pada compiler directive parallel kode tersebut

perhatikan perubahannya

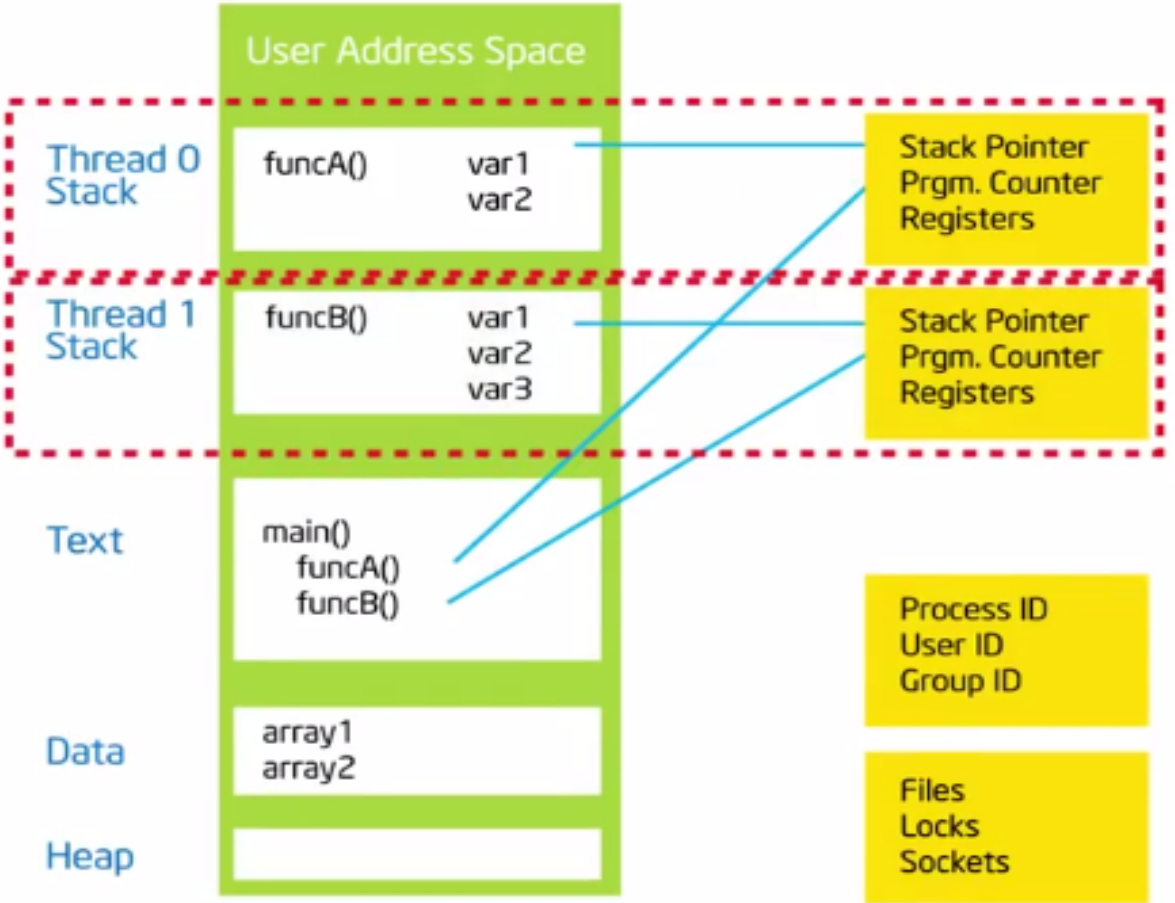
Proses (Serial)



Process:

* An instance of a program execution.
* The execution context of a running program... i.e. the resources associated with a program's execution.

Proses (Threading)



Threads:

- ★ Threads are "light weight processes"
- ★ Threads share Process state among multiple threads. This greatly reduces the cost of switching context.

Tip

1. OpenMP membuat data stack yang berbeda bagi setiap thread untuk menyimpan variabel private (master thread menggunakan stack reguler)
2. Ukuran stacknya tidak standard (Intel compiler 4MB, gcc 2MB)
3. Jika stack space terlampaui, program melempar segmentation fault
4. Untuk meningkatkan ukuran stack space, bisa digunakan environment variabel OMP_STACKSIZE
5. Untuk memastikan master thread memiliki stack space yang cukup bisa digunakan perintah ulimit -s (UNIX/LINUX)

Work Sharing (manual)

```
for (int i=0; i<N; i++)
        {
        kerjakan_tugas(i);
        }
```

Work Sharing (manual)

```
#pragma omp parallel private (id, num, s, f, n)
{
id = omp_get_thread_num();
num = omp_get_num_threads();
s = id*(N/num);
f = (id+1)*(N/num);
for (int i=s; i<f; i++)
        {
        kerjakan_tugas(i);
        }
}
```

Misal, ada 100 tugas (N=100) dibagi menjadi 4 thread (num=4). Tiap chunk berukuran N/num = 25 tugas.

Thread 0 mengerjakan tugas 0 s.d. 24

Thread 1 mengerjakan tugas 25 s.d. 49

Thread 2 mengerjakan tugas 50 s.d. 74

Thread 3 mengerjakan tugas 75 s.d. 99

- merepotkan
- mudah terjadi error
- bisa jadi ada pekerjaan yang terlupakan (tidak dikerjakan oleh thread

Membagi Loop (OpenMP)

Menggunakan directive for ;

```
#pragma omp parallel
{
        #pragma omp for
        for (int i=0; i<N; i++)
        {
                kerjakan_tugas(i);
        }
}
```

Directive for

Directive parallel dan for bisa digabungkan agar makin efisien baris source codenya

```
#pragma omp parallel for
        for (int i=0; i<N; i++)
        {
                kerjakan_tugas(i);
        }
```

Directive for

Directive for mengasumsikan bahwa sudah ada beberapa thread pada blok tersebut.

Jika blok paralel tidak diinisiasi sebelumnya, maka perintah di bawah directive for akan dijalankan pada sebuah prosesor secara serial.

Tugas 1

Ada array A berisi 1000 elemen double yang menyatakan jari-jari 1000 buah bola. Ingin dihitung volume keseribu bola tersebut dengan rumus :

V = 4/3 * pi * r^3

Klausa Schedule

Mengatur bagaimana iterasi dibagi kepada tiap anggota tim

Bentuknya

#pragma omp for schedule(tipe, chunk)

Tipe Schedule

1. Static : loop dibagi menjadi bongkahan dan diberikan secara statis kepada masing masing thread. Jika tidak ditentukan, iterasinya dibagi rata (kalau memungkinkan)
2. Dynamic : loop dibagi menjadi bongkahan dan diberikan secara dinamik kepada masing-masing thread. Saat thread menyelesaikan satu bongkah pekerjaan, secara dinamis pekerjaan lain diberikan lagi. Ukuran bongkahan default adalah 1.

Tipe Schedule

3. Guided : iterasi diberikan kepada thread secara dinamis dalam blok-blok. Mirip dengan dynamic, tapi ukuran blok berkurang tiap kali diberikan pada thread. Ukuran blok terkecil ditentukan oleh ukuran chunk.
4. Runtime : keputusannya ditunda hingga program berjalan.
5. Auto : keputusan schedulingnya diserahkan kepada compiler atau sistem

Klausa Lain Untuk For

Nowait : thread tidak melakukan singkronisasi di akhir blok paralel

Ordered : iterasi pada loop harus dilakukan seperti pada program serial

Collapse : jika terdapat nested loop, menentukan berapa banyak loop yang harus dirombak ke dalam iterasi

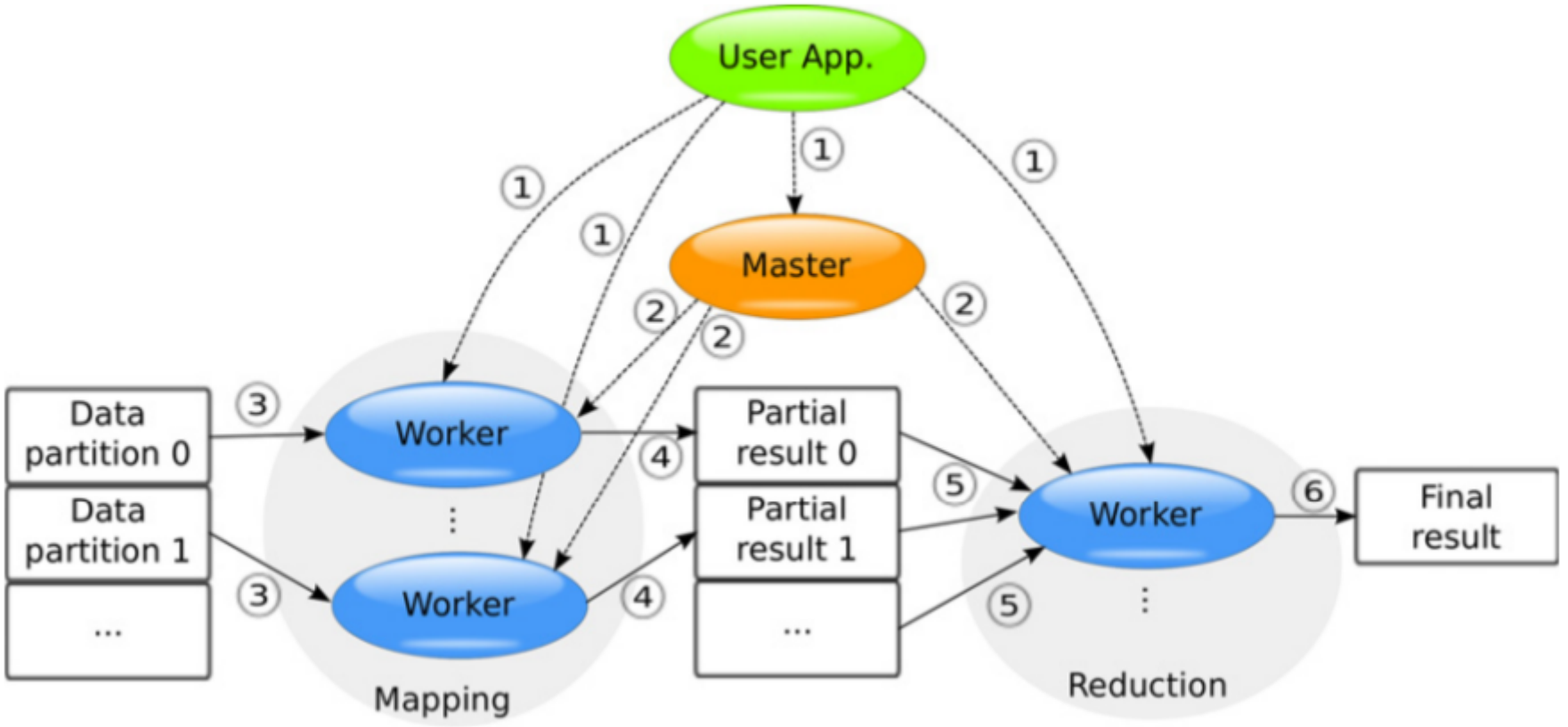
Map and Reduce Model

- Turunan dari pola Master-Worker
- Populer salah satunya karena diimplementasikan oleh google dalam mesin pencarinya.

Prosedur Map = membagi pekerjaan ke beberapa unit proses

Prosedur Reduce = menggabungkan lagi hasil pekerjaan parsial ke dalam satu hasil akhir

Map and Reduce Model



Map and Reduce Model

1. membuat master dan worker thread
2. task assignment oleh master thread
3. data input oleh worker yang terlibat dalam proses mapping
4. proses penyimpanan hasil parsial
5. membaca semua hasil parsial untuk proses reduce
6. menyimpan hasil akhir

Klausa Reduction

Banyak program yang menggunakan operasi seperti berikut :

```
for (int i=0; i<N; i++)
{
        a = a op expresi;
}
```

[a = a op expresi;] disebut sebagai operasi reduksi

Loop di atas mengandung ketergantungan (flow) antara satu i dengan i selanjutnya. Menghalangi paralelisasi

Klausa Reduction

Open MP memiliki klausa untuk operasi reduksi. Klausanya berupa reduction(op:list), dan dapat digunakan jika :

1. **a** adalah variabel skalar pada **list**
2. **expresi** adalah ekspresi skalar yang tidak berhubungan dengan **a**
3. operasi **op** tertentu (di slide selanjutnya)
4. untuk fortran, **op** bisa bersifat intrinsik seperti MAX, MIN, IOR
5. variabel pada list harus shared

Operator Reduksi

a = a op expr

a = expr op a (selain pengurangan)

a binop = expr

x++

++x

x--

--x

op bisa berupa +, *, -, /, &, ^, |, &&, ||

binop bisa berupa +, *, -, /, &, ^, |

Penjumlahan Elemen Array

```
#pragma omp parallel for reduction(+:sum)
for (int i=0;i<n;i++)
{
        sum = sum + A[i];
}
```

Tugas 2

Buat program paralel untuk menghitung hasil kali vektor (dot product), memanfaatkan fungsi reduksi pada openmp.

Tugas 3

Hitung

$$\int_0^1 4/(1+x^2)\,dx$$

Sections

Directive sections adalah konstruksi non-iteratif untuk pembagian pekerjaan

Directive ini memerintahkan kode-kode pada section untuk dibagi antara thread-thread pada tim

Tiap section yang independen ada di dalam directive sections, dijalankan satu kali oleh sebuah thread pada tim.

Tiap section bisa dijalankan oleh thread berbeda. Bisa juga satu thread menjalankan beberapa section jika cukup cepat dan diizinkan dalam implementasinya.

Section

```
#pragma omp parallel shared(a,b,c,d) private(i) {
  #pragma omp sections nowait
    {
    #pragma omp section
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
    #pragma omp section
    for (i=0; i < N; i++)
      d[i] = a[i] * b[i];
    }
  }
```

Quiz

Apa yang terjadi jika banyak section tidak sama dengan banyak thread?

1. section lebih banyak dari thread
2. thread lebih banyak dari section
3. thread mana yang akan menjalankan section yang mana?

Jawaban

1. ekstra section akan dikerjakan tergantung oleh implementasinya
2. ada thread yang bekerja, ada yang tidak
3. tergantung implementasinya. dalam satu eksekusi dan lainnya bisa saja berbeda

Single

Directive single memastikan bahwa yang terdapat di dalamnya hanya dikerjakan oleh satu thread dalam tim

Berguna untuk kode yang tidak boleh diakses oleh lebih dari 1 thread *(not thread safe code)*. Misalnya menuliskan file.

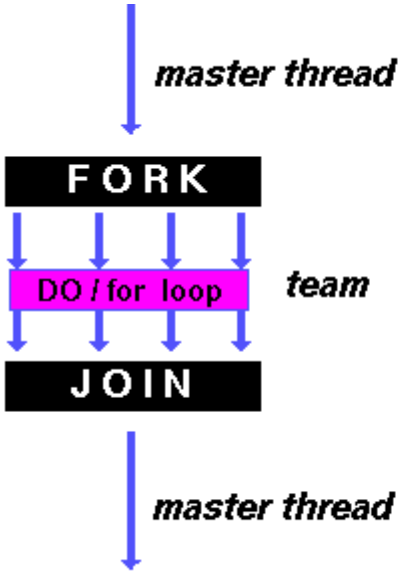
Single

Thread dalam tim tidak akan mengerjakan kode dalam blok single selain satu thread. Thread lainnya akan menunggu di akhir blok paralel kecuali disebutkan untuk nowait.

Ilustrasi

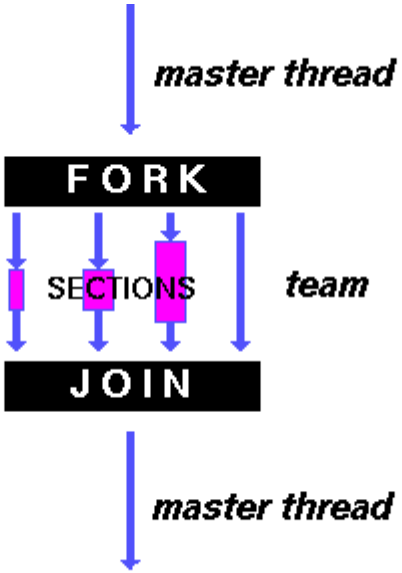
For

paralelisme data



Section

paralelisme fungsi



Single

menserialkan bagian kode