2015 Summer School (East Ontario) on High Performance Computing

Queen's University, July 27-31

Combination of MPI and OpenMP

Gang Liu and Hartmut Schmider

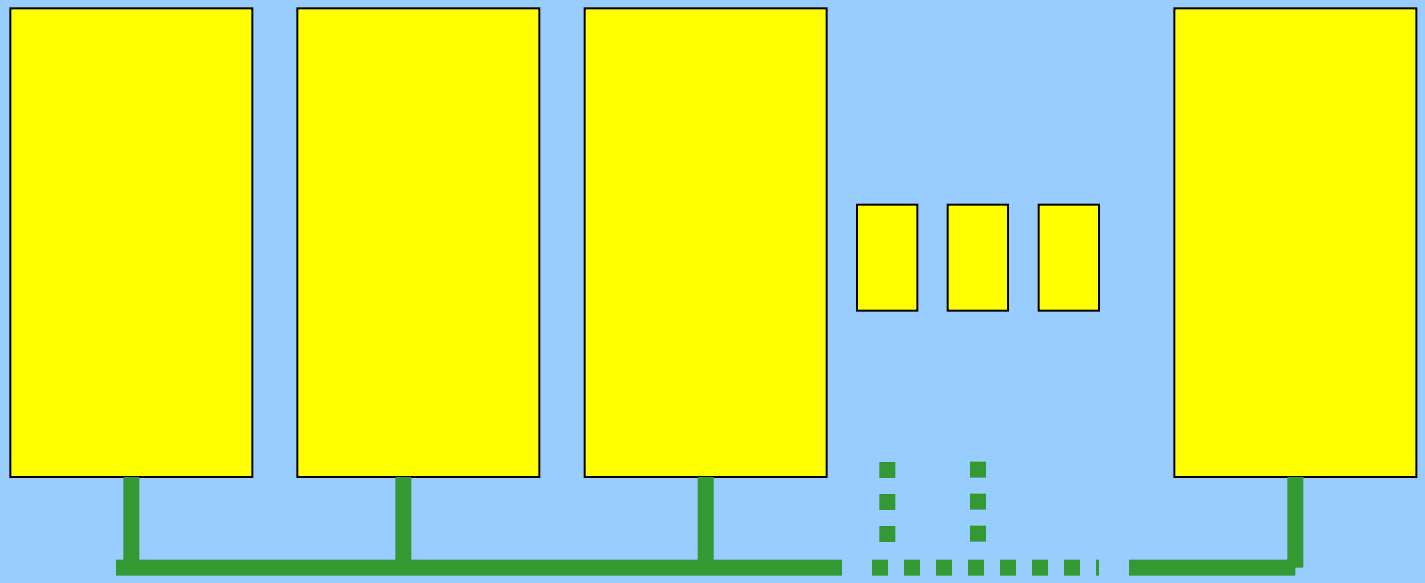
{gang.liu, hartmut.schmider}@queensu.ca

MPI and OpenMP are completely different parallelisms. MPI is for distributed memory, while OpenMP is for shared memory system.

However, they can be combined.

A typical cluster

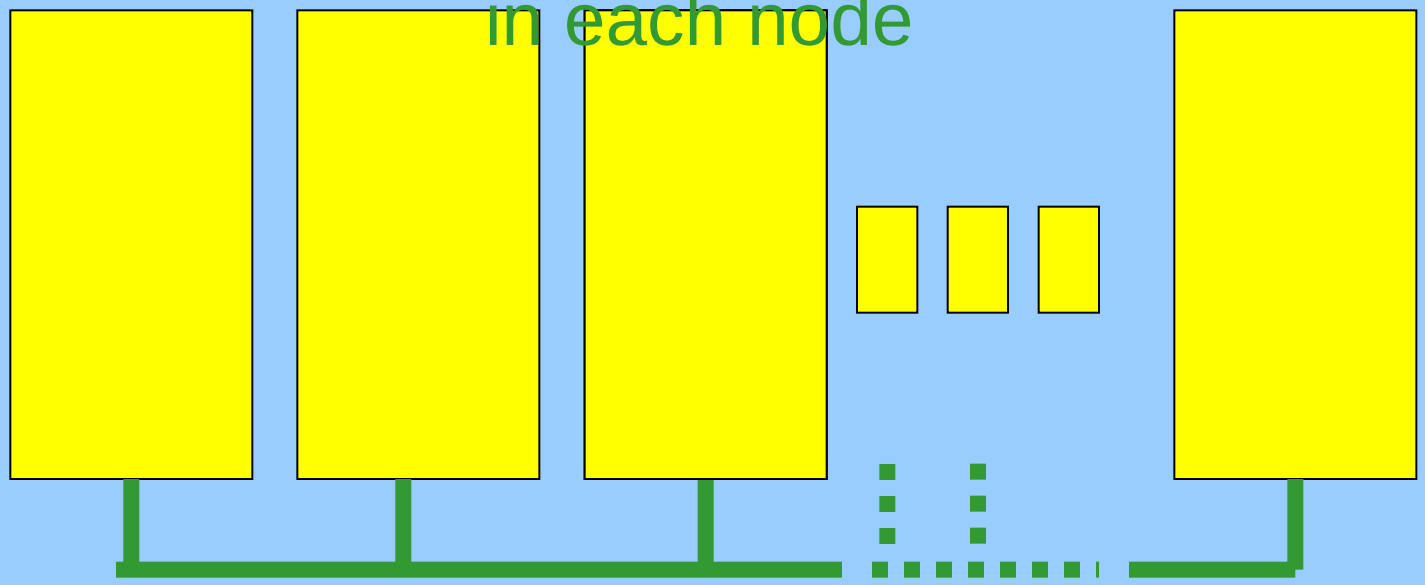
is made of many independent nodes. CPUs of any node can not access memory of any other nodes.



A typical cluster

is made of many independent nodes. CPUs of any node can not access memory of any other nodes.

Many cores/CPUs
with shared memory
in each node



Nowa

is ma odes.
CPUs ess
memo

Many cores/CPUs
with shared memory
in each node

This feature of the physical system can be very well described by a mixture of MPI and OpenMP parallelisms. MPI can perform the parallelism across nodes/processes, while OpenMP does those inside each node/process.

A framework of mixture of MPI and OpenMP

Parallelize the whole code with MPI as an outer layer parallelism.

Then regard each MPI process as a regular serial code/run, and parallelize it with OpenMP as an inner layer parallelism.

A Challenge

Can MPI communications be done in OpenMP parallel region ?

If yes, how to do them safely, since all threads of a fixed process share the same MPI rank number ?

The Challenge

is an MPI thread-safe problem.
Some MPI implementations claim so, while most not.

Let us consider no thread-safe only.
Then we suggest:

A basic rule

MPI is only allowed between different processes, with only one thread per process involved at a given time (placed into an OpenMP critical region)

A simple example of mixture of MPI and OpenMP

The square-root example with two processes

```
for(i=0;i<=M;i++)
DO I=0,M,1
```

$$S = \sum_{i=0}^{M} \sqrt{i}$$

The square-root example with two processes

```
for(i=myid;i<=M;i=i+2)
DO I=MYID,M,2
```

$$S = \sum_{i=0}^{M} \sqrt{i}$$

The square-root example with two processes

```
for(i=myid;i<=M;i=i+2)
DO I=MYID,M,2
```

$$S = \sum_{i=0}^{M} \sqrt{i}$$

Process 0

$$\mathrm{MYS} = \sqrt{0} + \sqrt{2} + \sqrt{4} + \cdots;$$

Process 1

$$\mathrm{MYS} = \sqrt{1} + \sqrt{3} + \sqrt{5} + \cdots$$

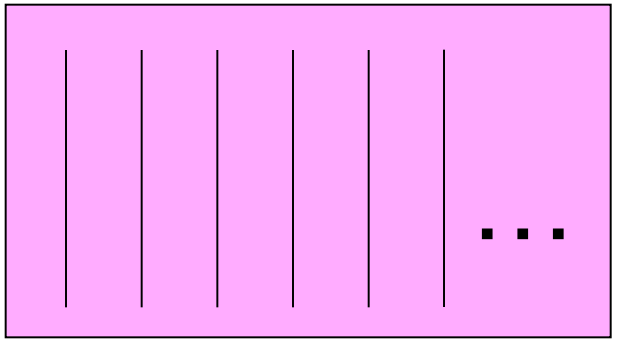
The square-root example with two processes

```
for(i=myid;i<=M;i=i+2)
DO I=MYID,M,2
```

$$S = \sum_{i=0}^{M} \sqrt{i}$$

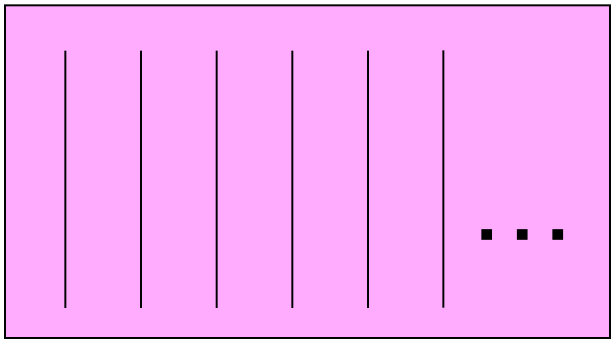
Process 0

$$\mathrm{MYS} = \sqrt{0} + \sqrt{2} + \sqrt{4} + \cdots;$$



Process 1

$$\mathrm{MYS} = \sqrt{1} + \sqrt{3} + \sqrt{5} + \cdots$$



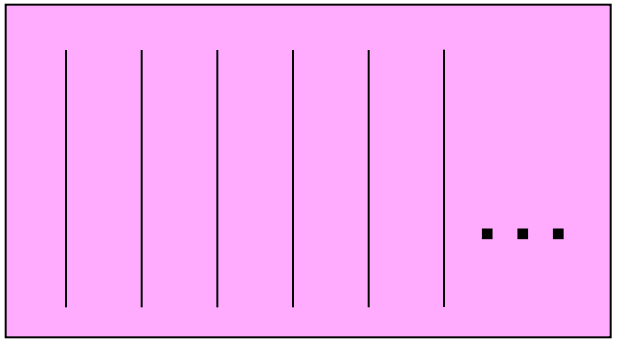
The square-root example with two processes

```
for(i=myid;i<=M;i=i+2)
DO I=MYID,M,2
```

$$S = \sum_{i=0}^{M} \sqrt{i}$$

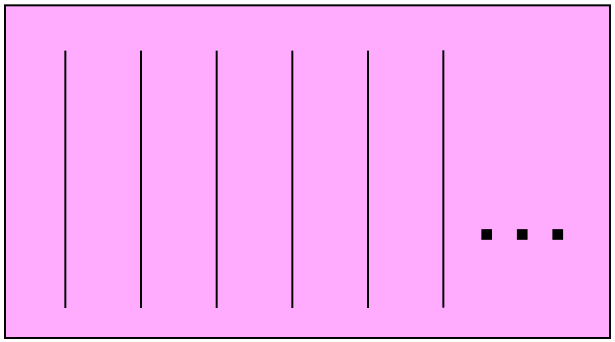
Process 0

$$MYS = \sqrt{0} + \sqrt{2} + \sqrt{4} + \cdots;$$



Process 1

$$MYS = \sqrt{1} + \sqrt{3} + \sqrt{5} + \cdots$$



$$MPI\_Reduce: S = MYS(process\,0) + MYS(process\,1)$$

Click here for mixed example in C

Click here for mixed example in F90

The square-root example with two processes

```
for(i=myid;i<=M;i=i+2)
DO I=MYID,M,2
```

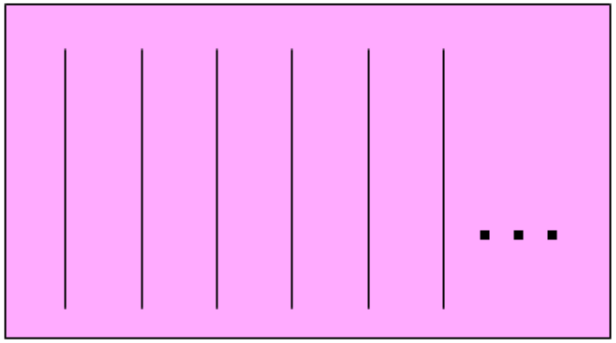
$$S = \sum_{i=0}^{M} \sqrt{i}$$

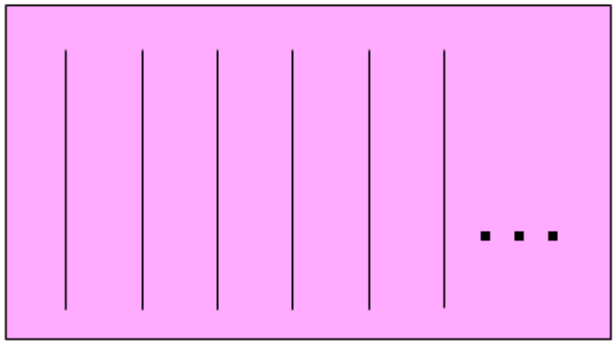
Process 0

$$\mathrm{MYS} = \sqrt{0} + \sqrt{2} + \sqrt{4} + \cdots;$$

Process 1

$$\mathrm{MYS} = \sqrt{1} + \sqrt{3} + \sqrt{5} + \cdots$$





$$\mathrm{MPI\_Reduce}: \mathrm{S} = \mathrm{MYS(process\ 0)} + \mathrm{MYS(process\ 1)}$$

Click here for mixed example in C

in F90

Lab Work I: Mixed/C(F90)/mixed

C compiling:

```
mpiicc –o mixed.exe –O3 –fopenmp  mixed.c
```

F90 compiling:

```
mpiifort –o mixed.exe –O3 –fopenmp  mixed.f90
```

Run:

```
cat mixed.in
OMP_NUM_THREADS=2 timex mpirun –np 2 ./miexed.exe < mixed.in
```

From now no, we will focus on

Double-layer Master-Slave Model

- A good example of MPI and OpenMP mixed
- Already converted into a library with open source code
- Distributes independent jobs dynamically, then useful for many researchers

From now no, we will focus on

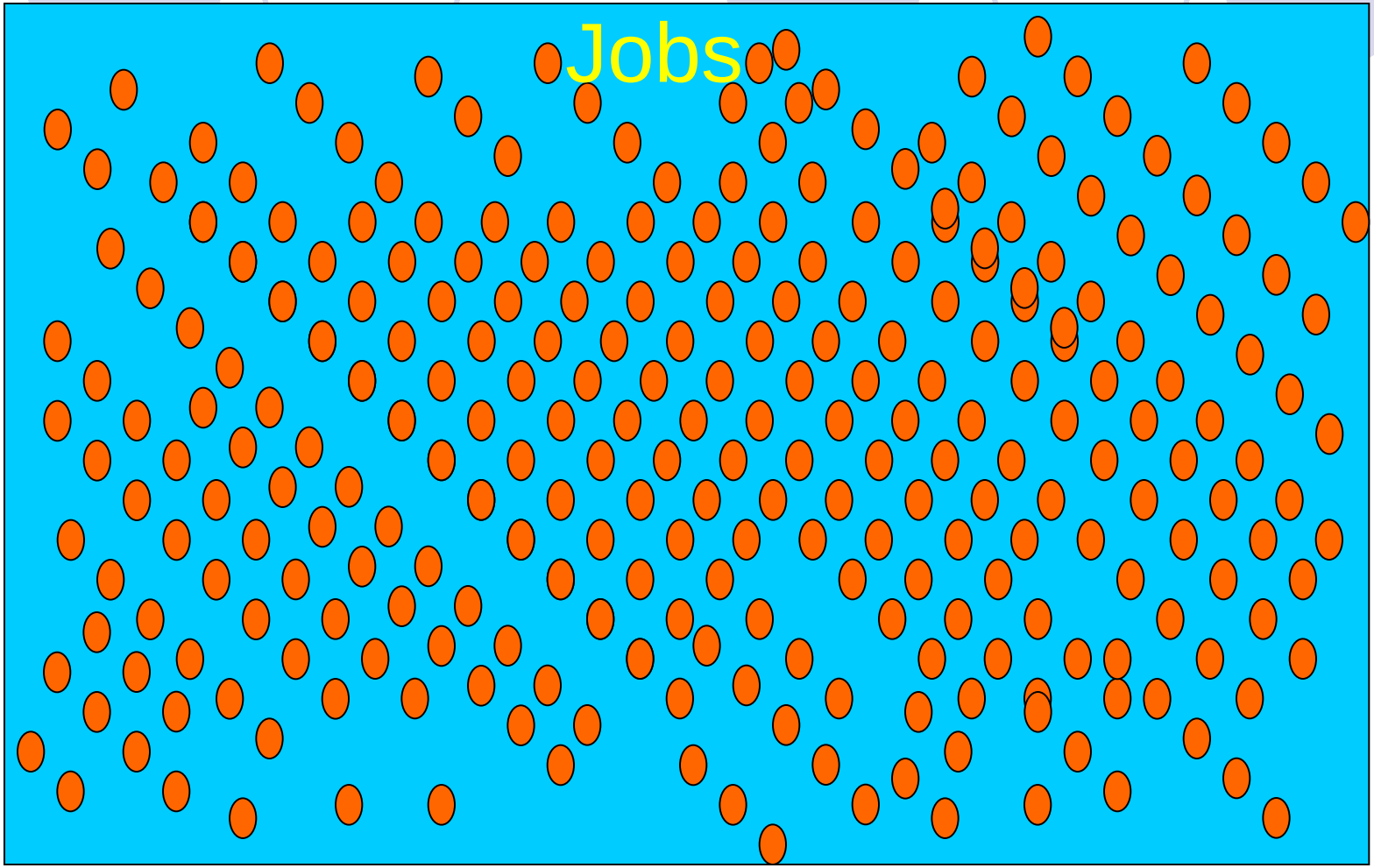
Double-layer Master-Slave Model

A combination of

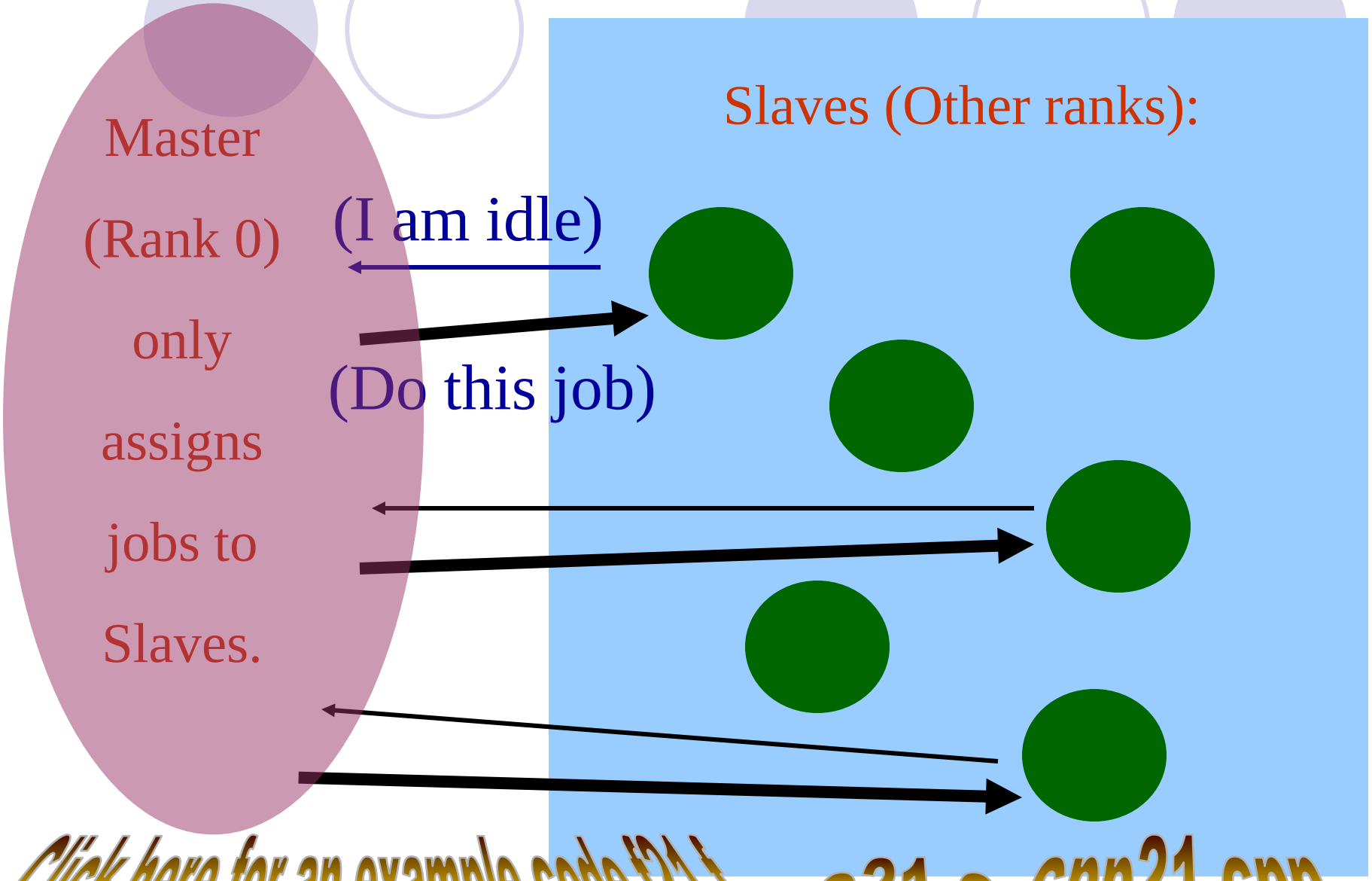
MPI master-slave model and

OpenMP all-slave model

MPI Master-slave parallel model

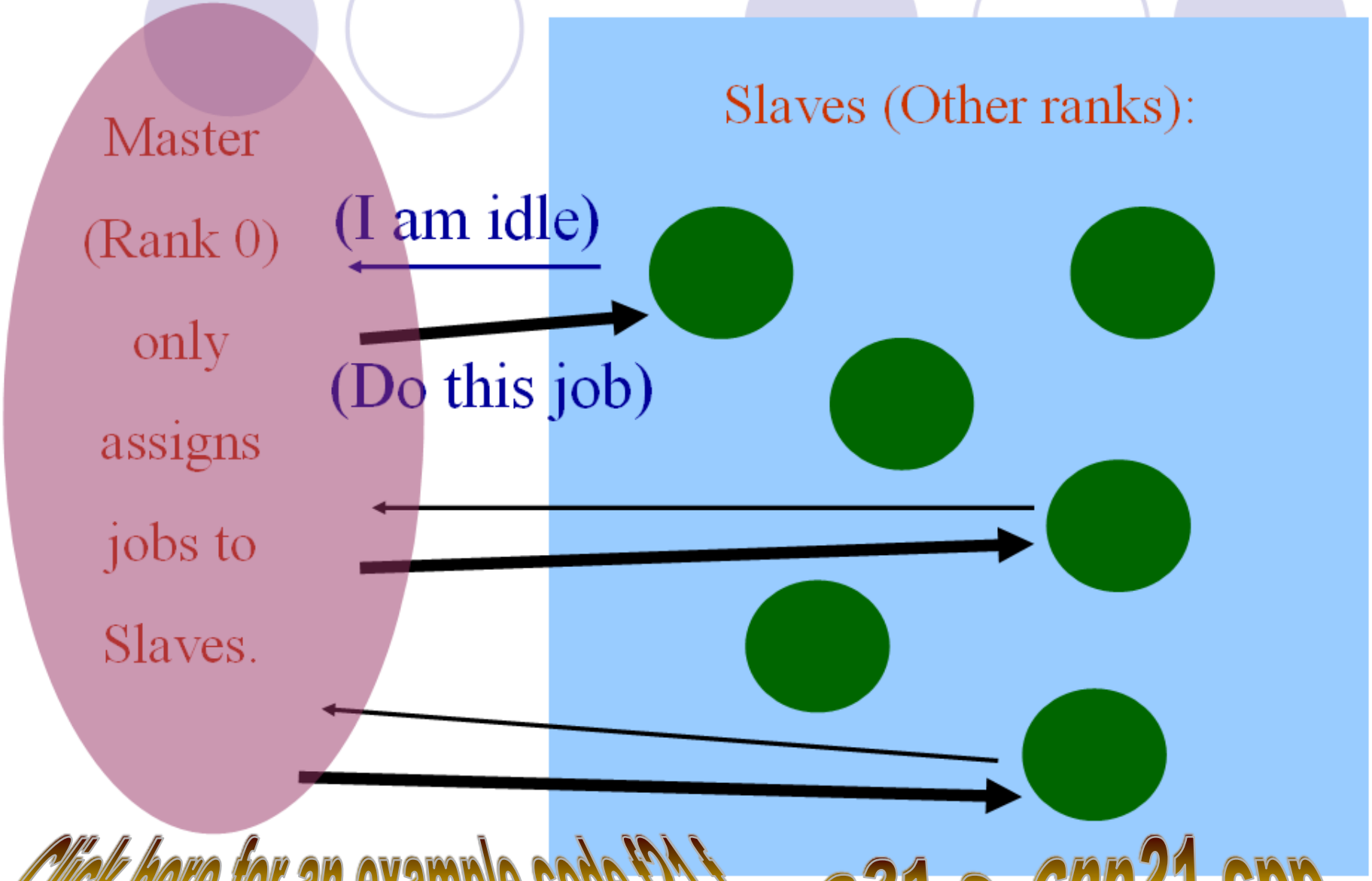


Master-slave parallel model



Click here for an example code f21.f c21.c cpp21.cpp

Master-slave parallel model



Click here for an example code f21.f c21.c cpp21.cpp

As you see,

in this MPI master-slave model, there must be a process to maintain the job number/counter. Since it is in the process local (distributed) memory, the process has to maintain it dedicatedly and send it to any other process who needs it, then the process becomes the master.

In OpenMP,

if the job number/counter is defined as a shared variable, then every thread can access it, and no master is needed.

OpenMP All-slave Model

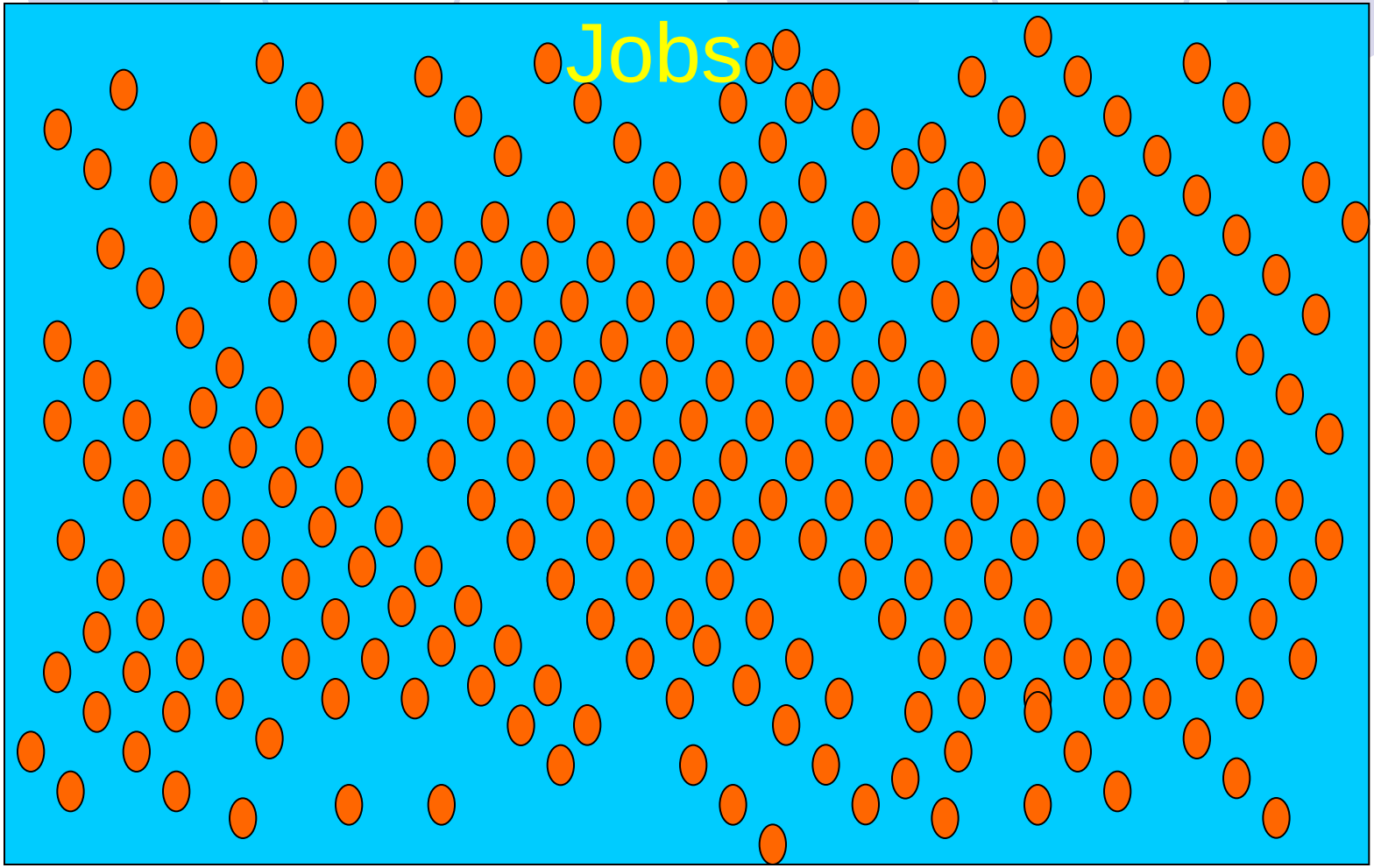
where whenever any thread becomes idle, he checks the shared job number/counter (of course, in a critical region). If any job left, he updates job number/counter, takes and completes the next job.

Click here for an example in FORTRAN

in C

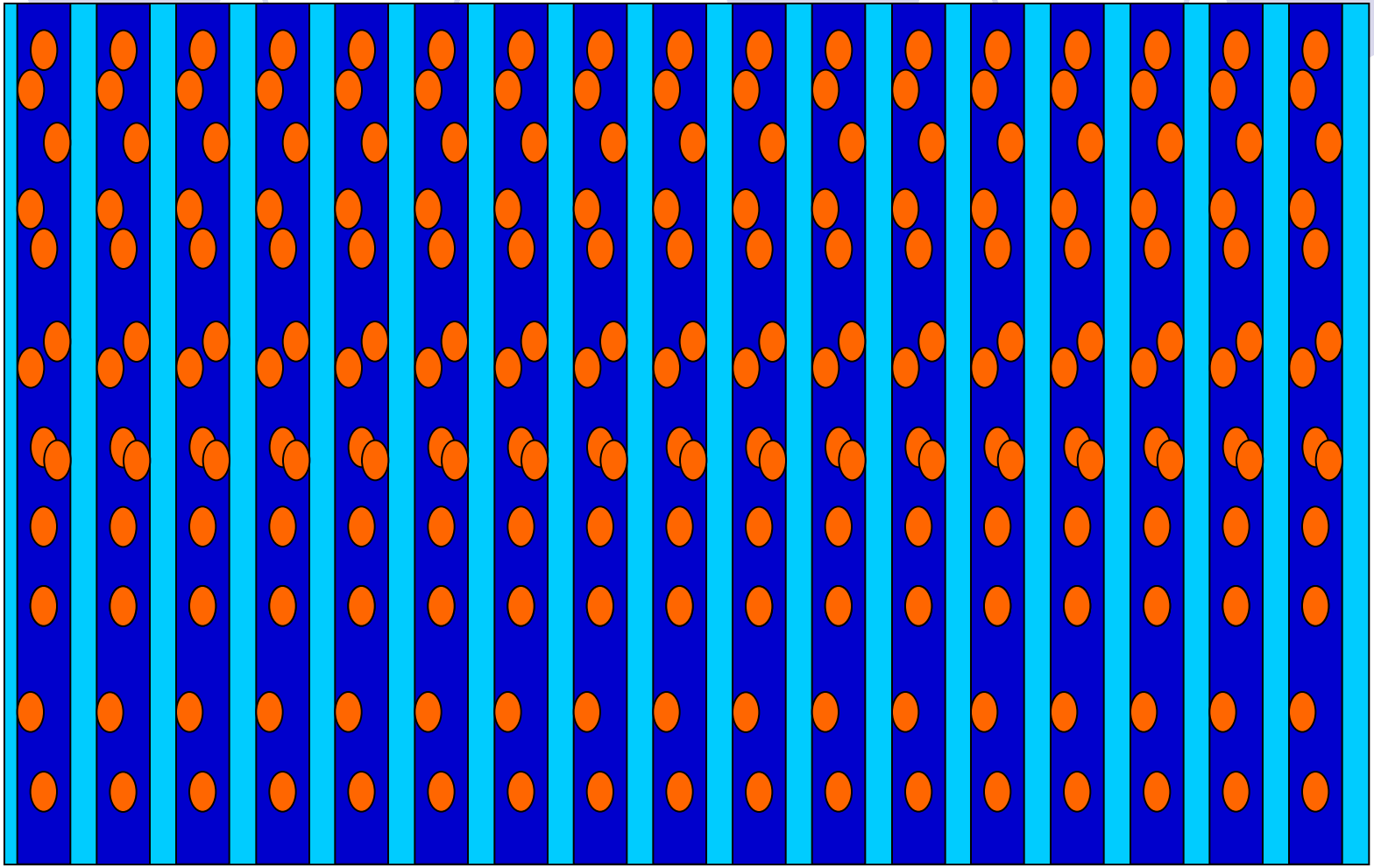
Double-layer Master-Slave Model

Double-layer Master-Slave Model



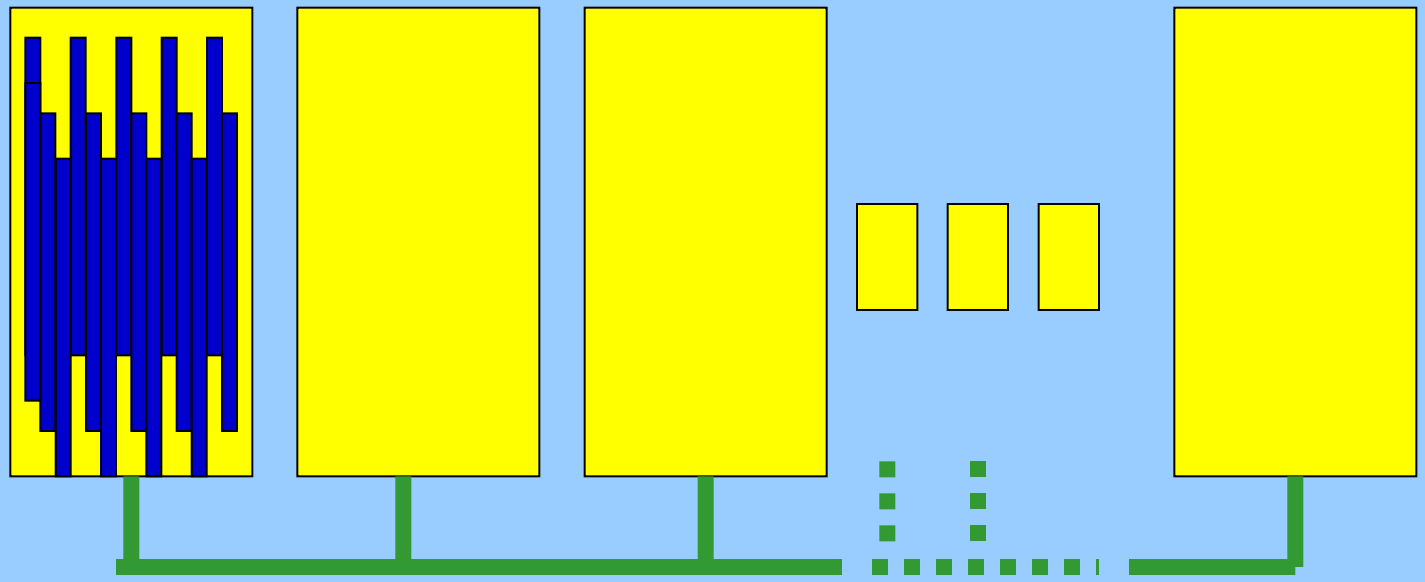
Double-layer Master-Slave Model

Jobs grouped



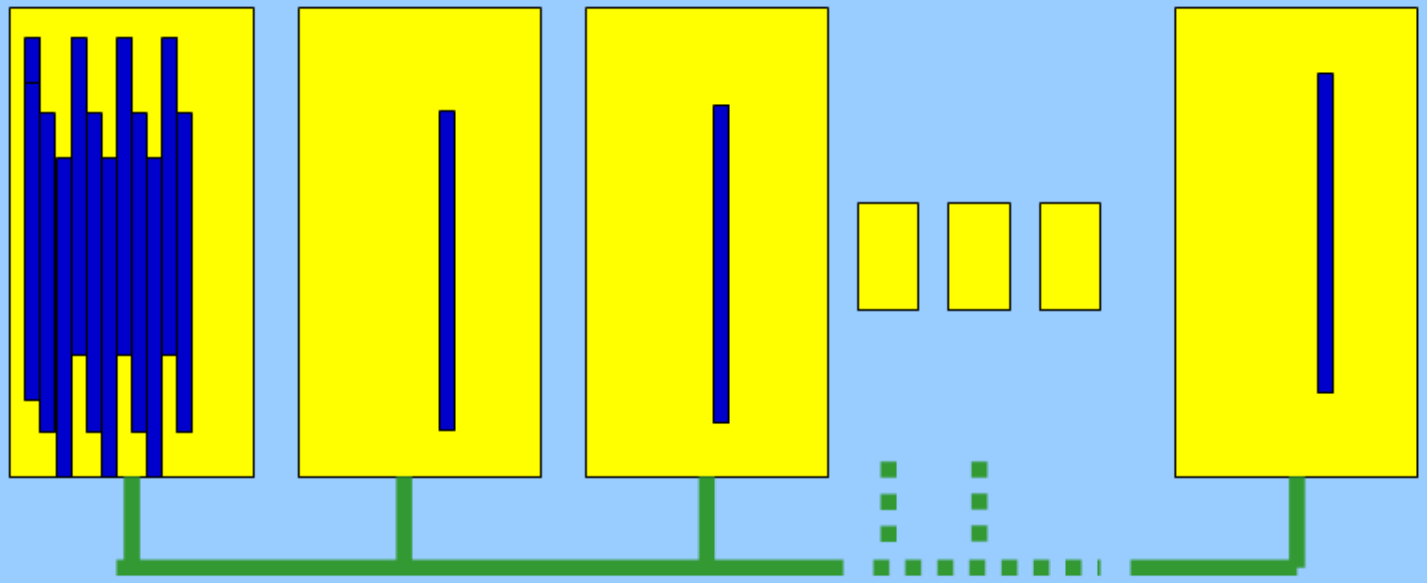
Double-layer Master-Slave Model

Job groups sent to nodes via
MPI master-slave model



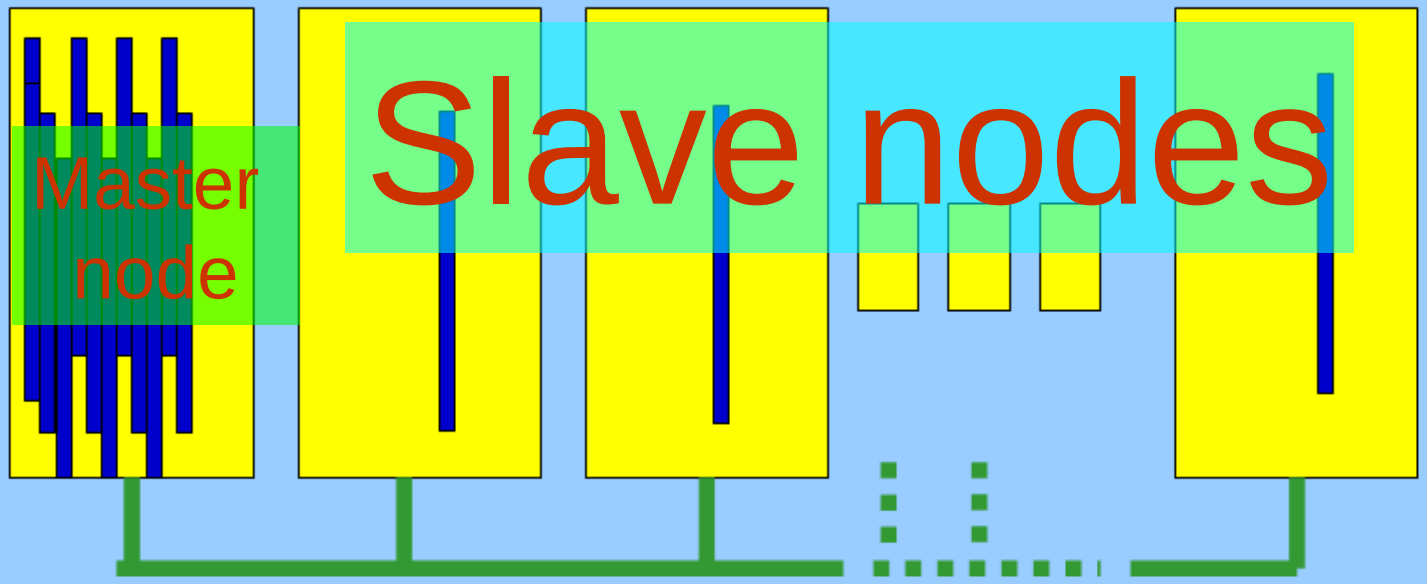
Double-layer Master-Slave Model

Job groups sent to nodes via
MPI master-slave model



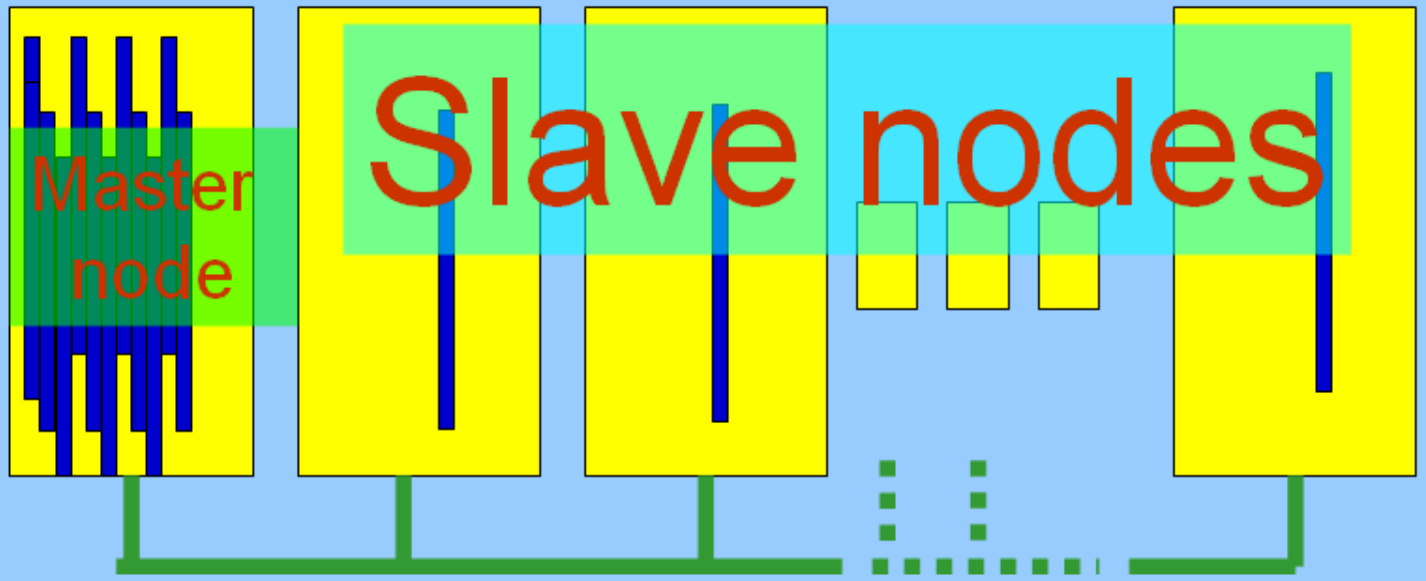
Double-layer Master-Slave Model

Job groups sent to nodes via
MPI master-slave model



Double-layer Master-Slave Model

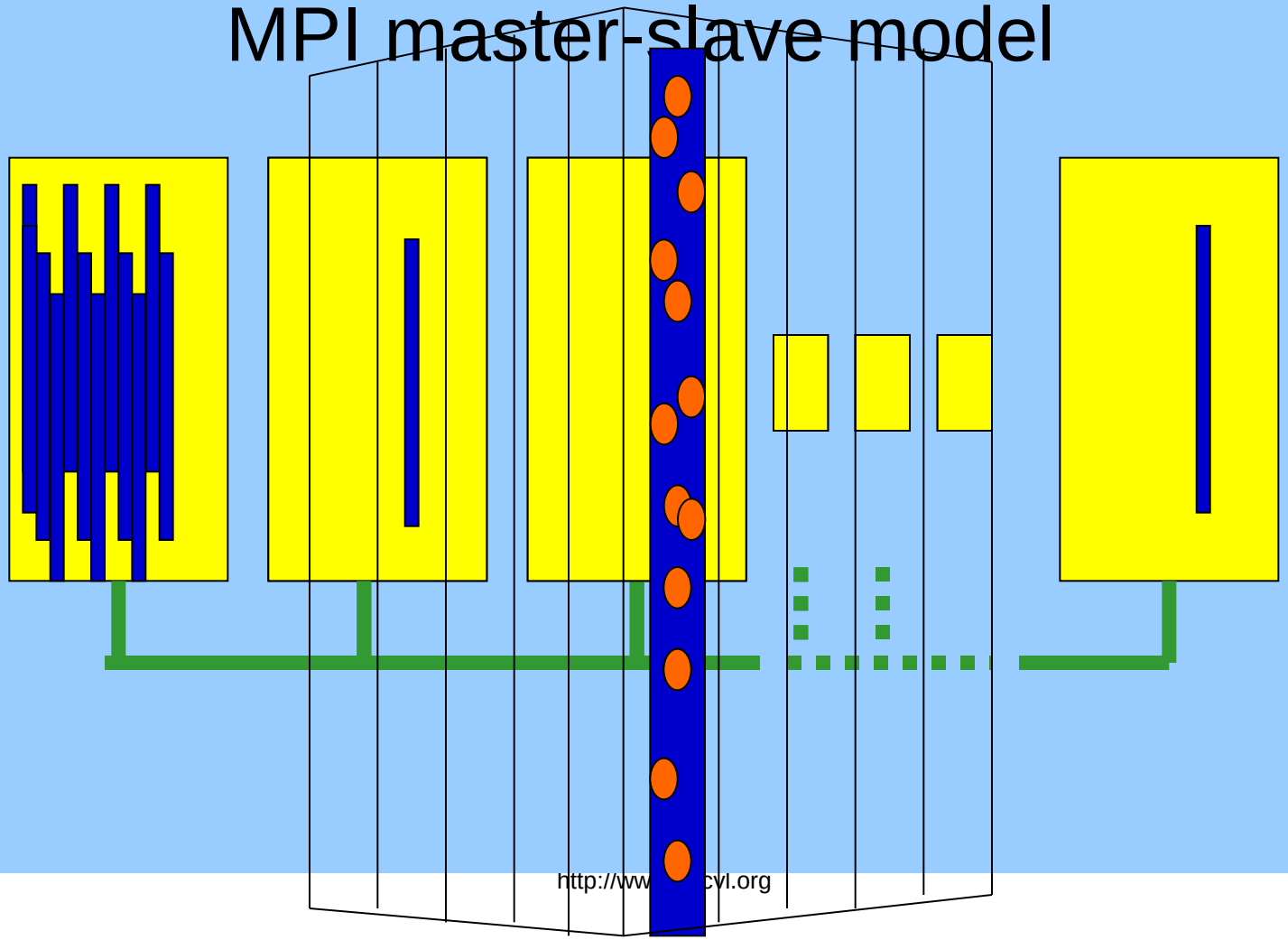
Job groups sent to nodes via
MPI master-slave model



Double-layer Master-Slave Model

Jobs in a group executed in the node by threads via an OpenMP all-slave model

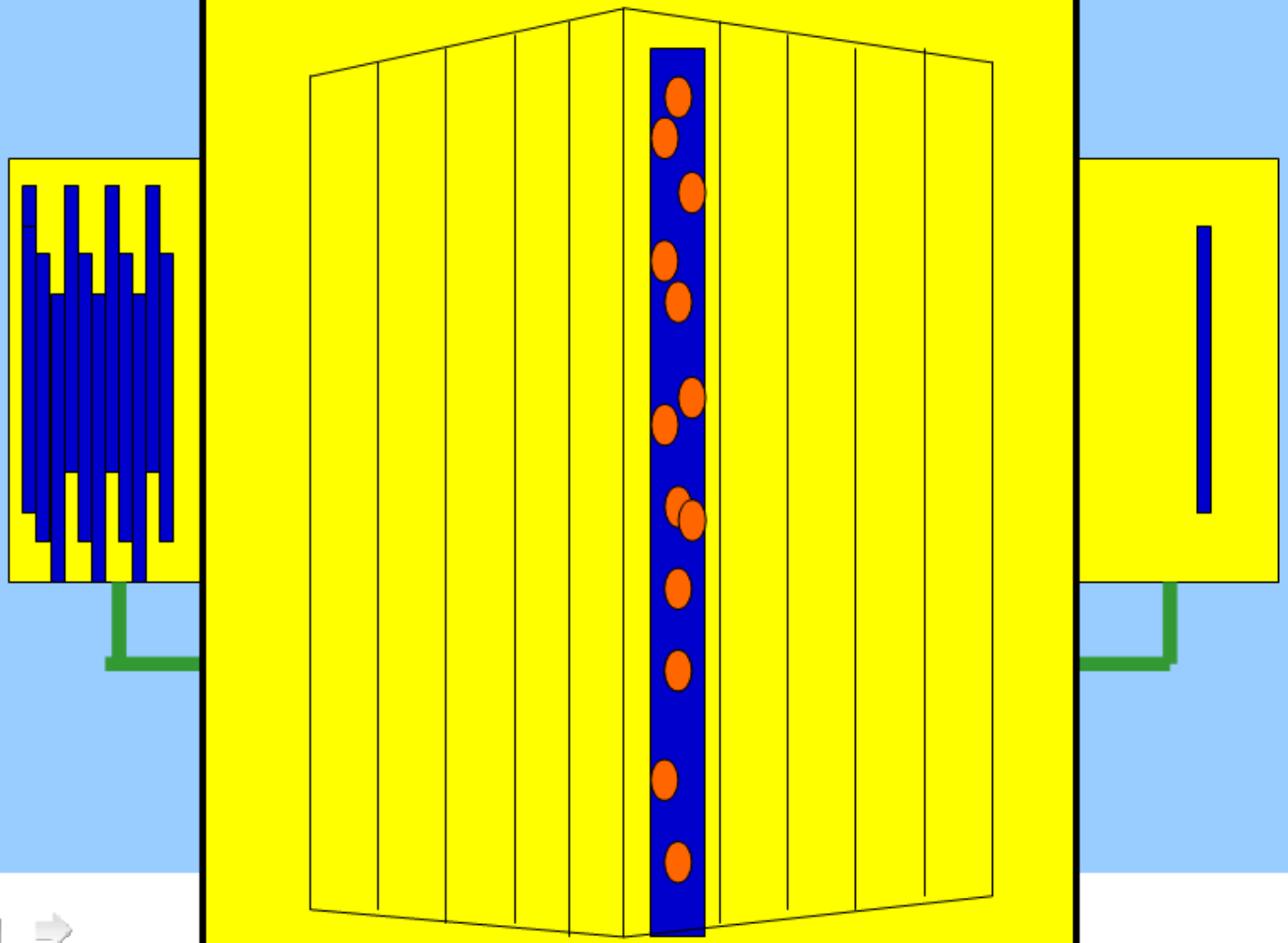
Job groups sent to nodes via MPI master-slave model



Doubl Model

Jo a

Jobs in a group executed in the node by threads via an OpenMP all-slave model



Double-layer Master-Slave Model

HPCVL supplies the DMSM library with source code for free.

A basic rule

MPI is only allowed between different processes, with only one thread per process involved at a given time (placed into an OpenMP critical region)

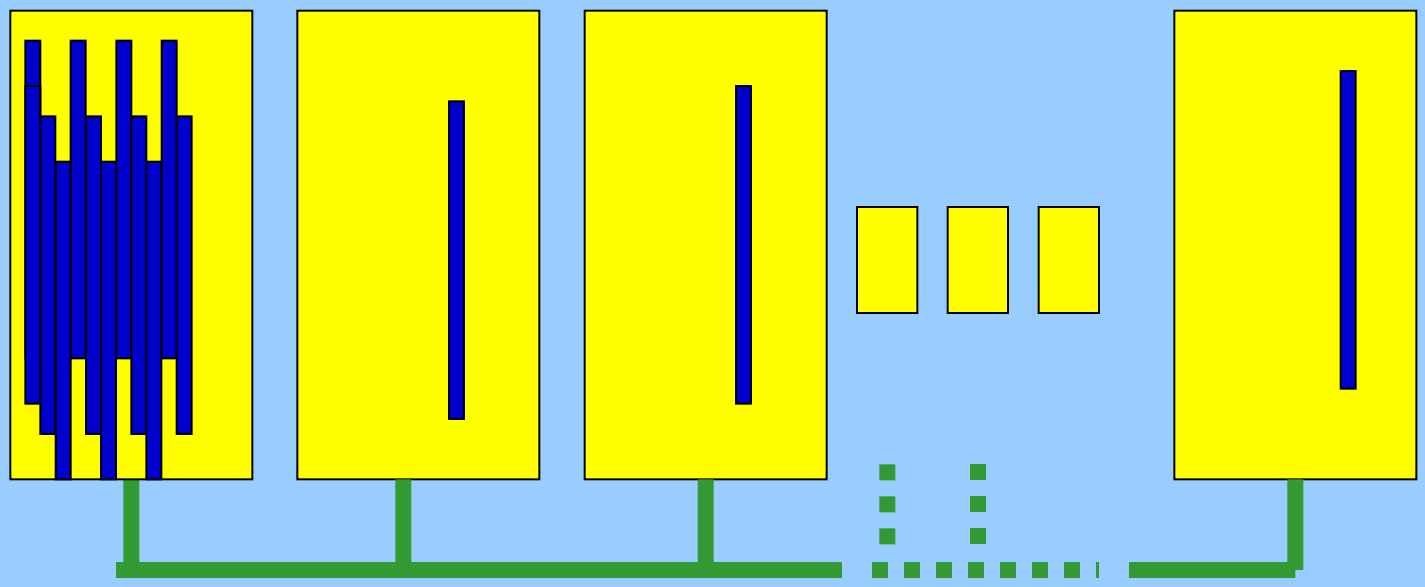
Double-layer Master-Slave Model

For levels of reliability and efficiency, any combination of the following three modes of the master node and three modes of the slave nodes are supplied in the library, although all of them have been tested in the HPCVL clusters with a success.

Double-layer Master-Slave Model

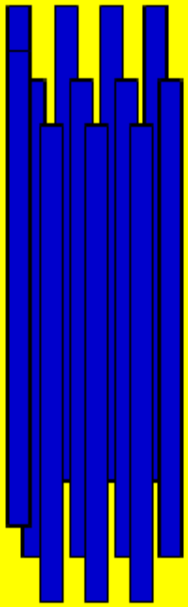
Master node
no OpenMP.

Master Mode 1

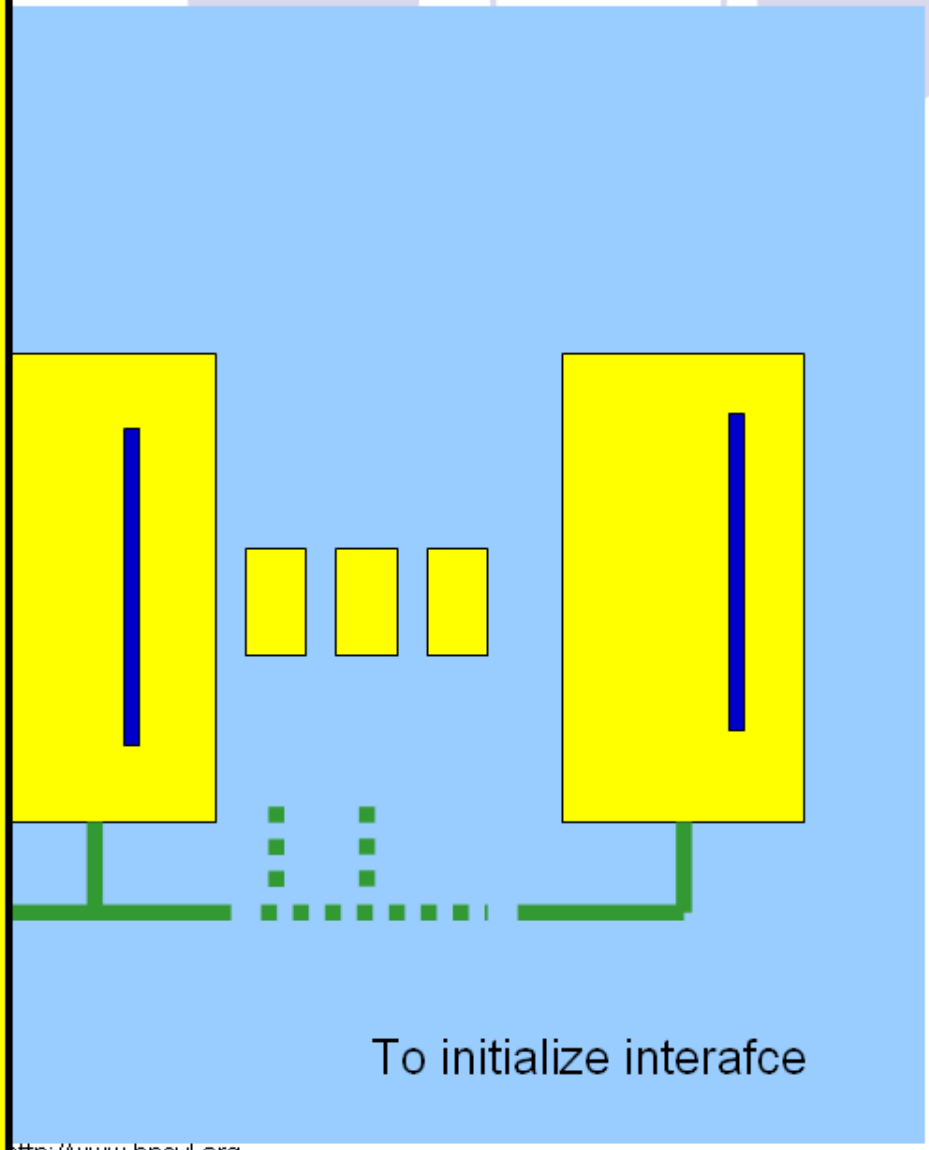


To initialize interafce

Master node
no OpenMP.



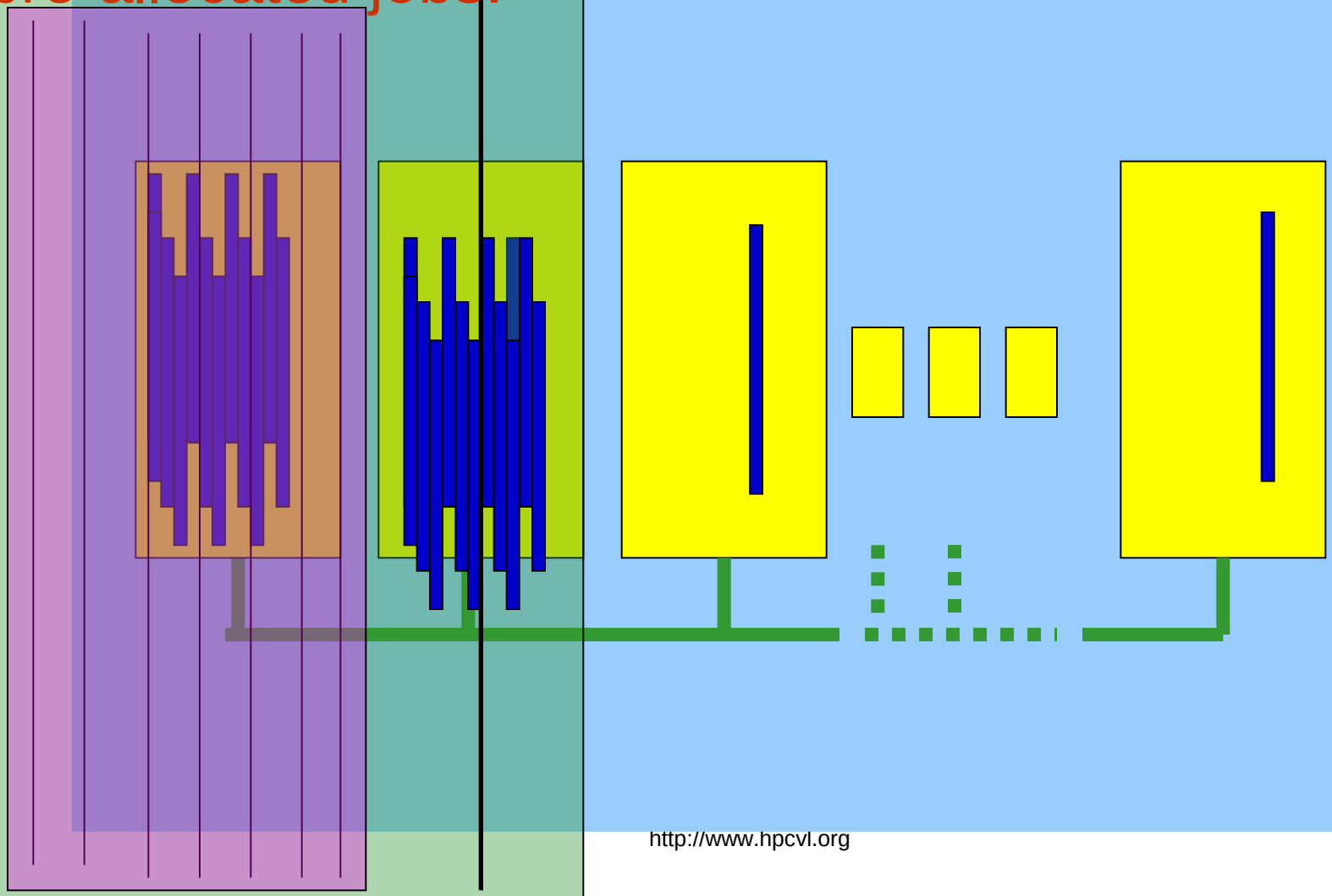
laster-Slave Model



Double-layer Master-Slave Model

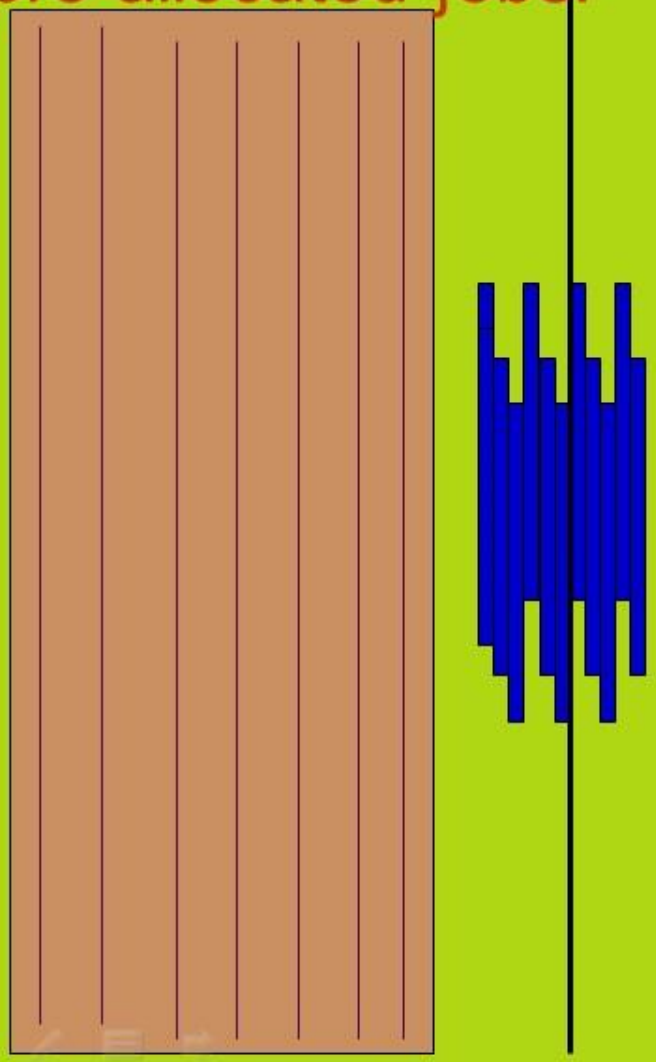
Master Mode 2: always in OpenMP

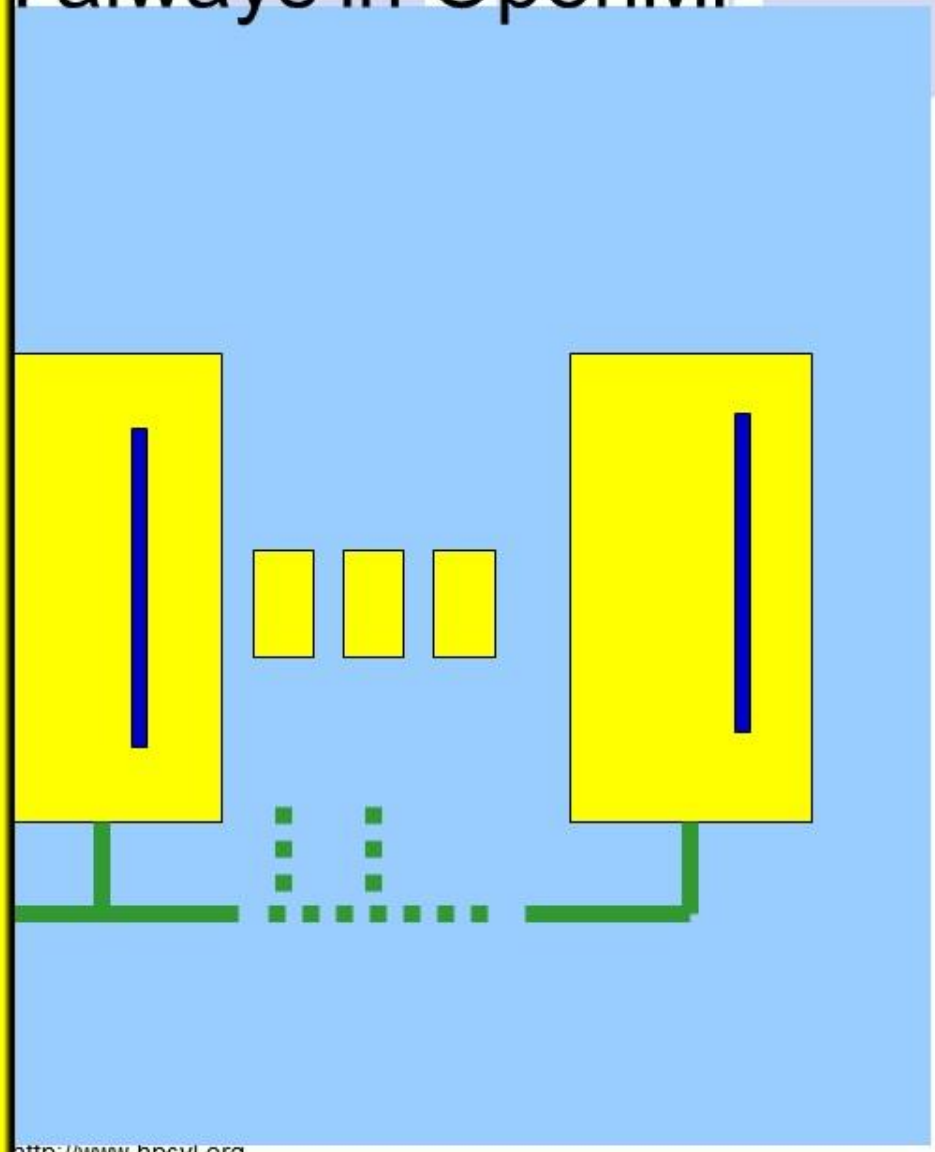
In the master node, main thread is the master, others do pre-allocated jobs.



laster-Slave Model
: always in OpenMP

In the master node, main thread is the master, others do pre-allocated jobs.

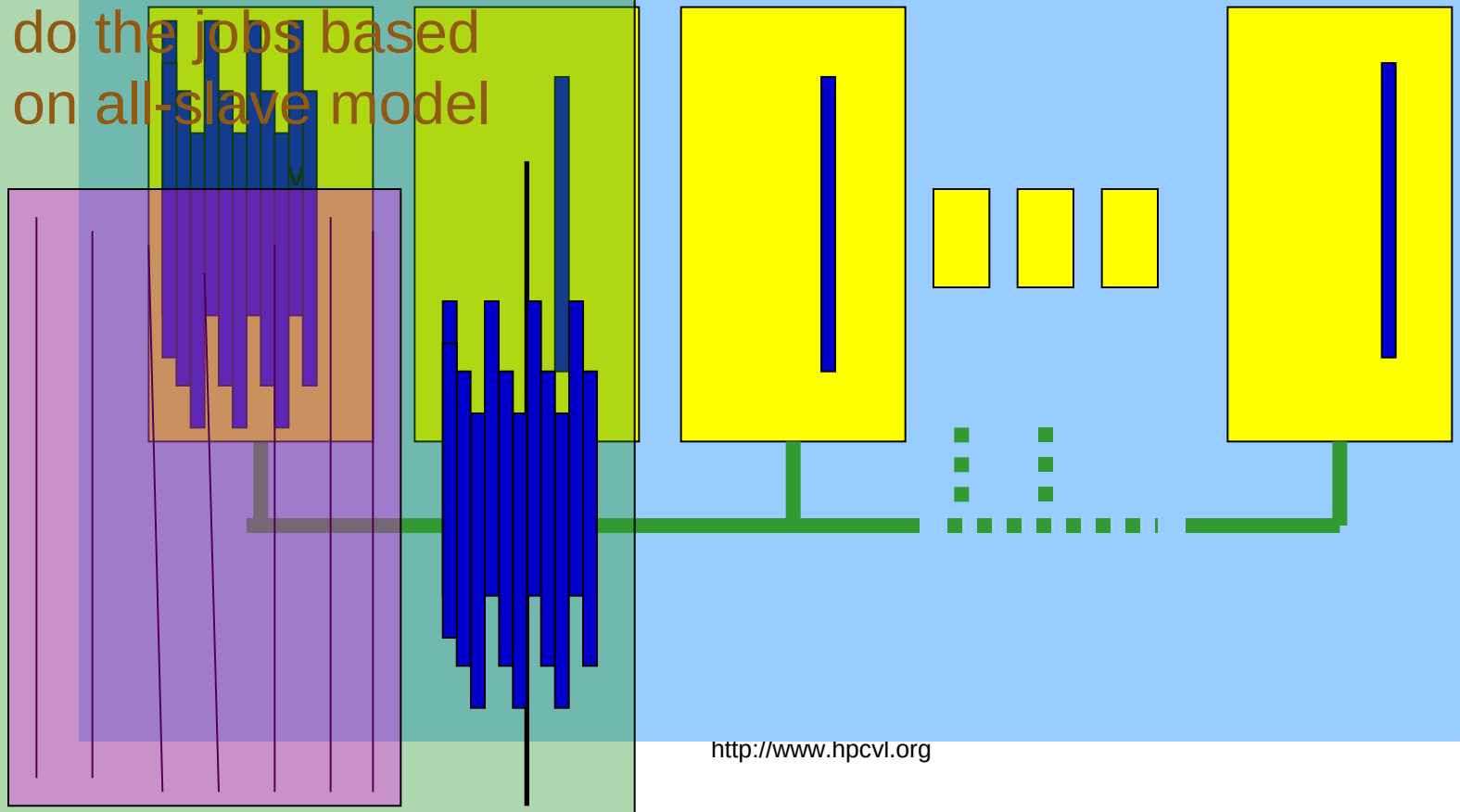




Double-layer Master-Slave Model

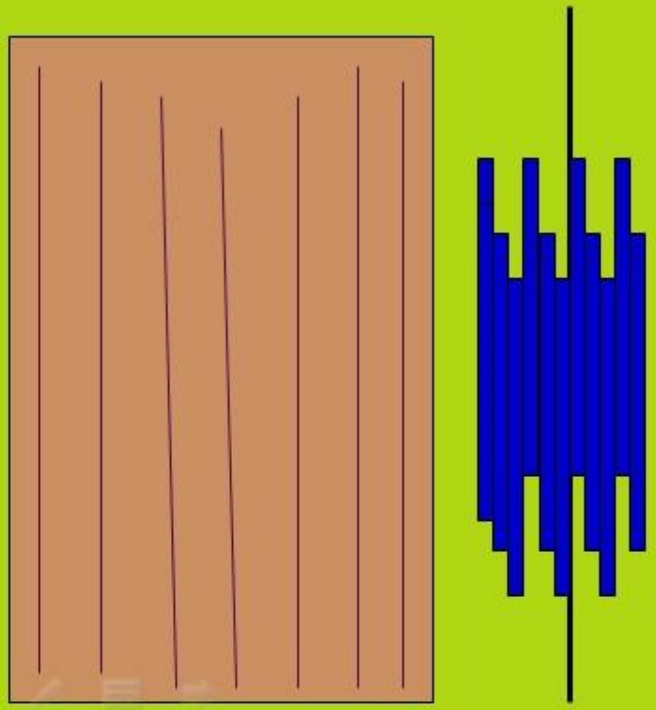
Master Mode 3: always in OpenMP

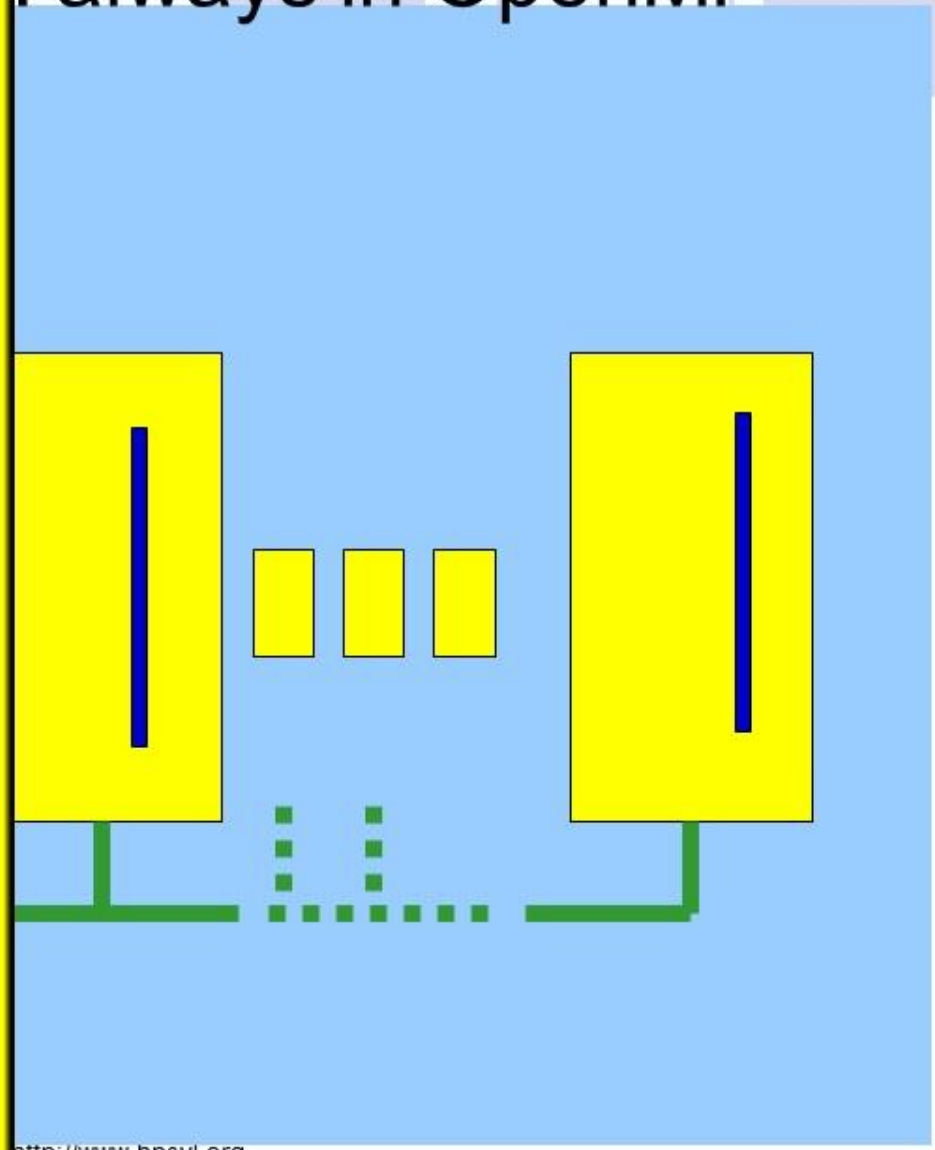
In the master node, main thread is the master, any other will get job groups inside the node in a critical region and do the jobs based on all-slave model



laster-Slave Model
: always in OpenMP

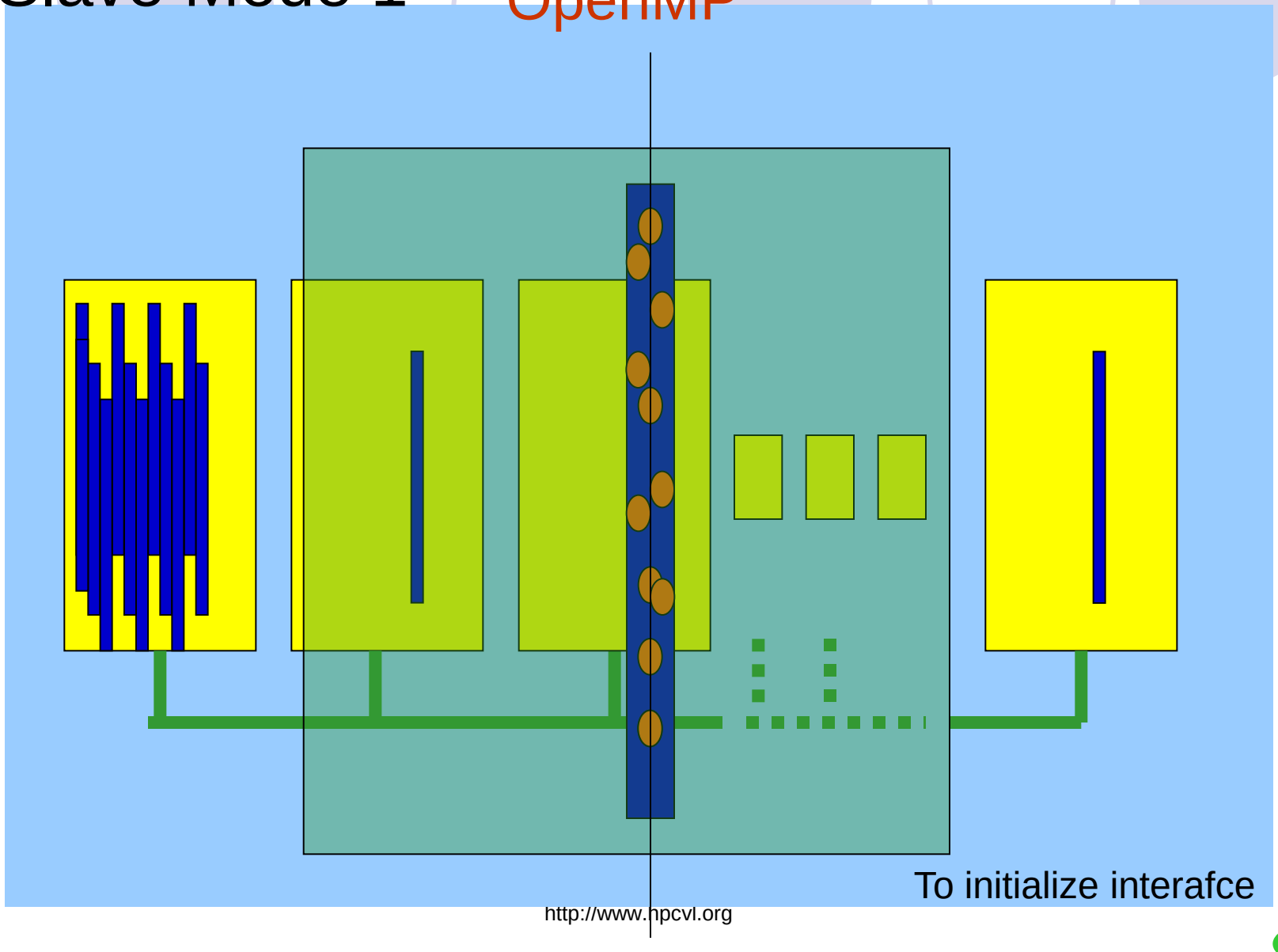
In the master node, main thread is the master, any other will get job groups inside the node in a critical region and do the jobs based on all-slave model



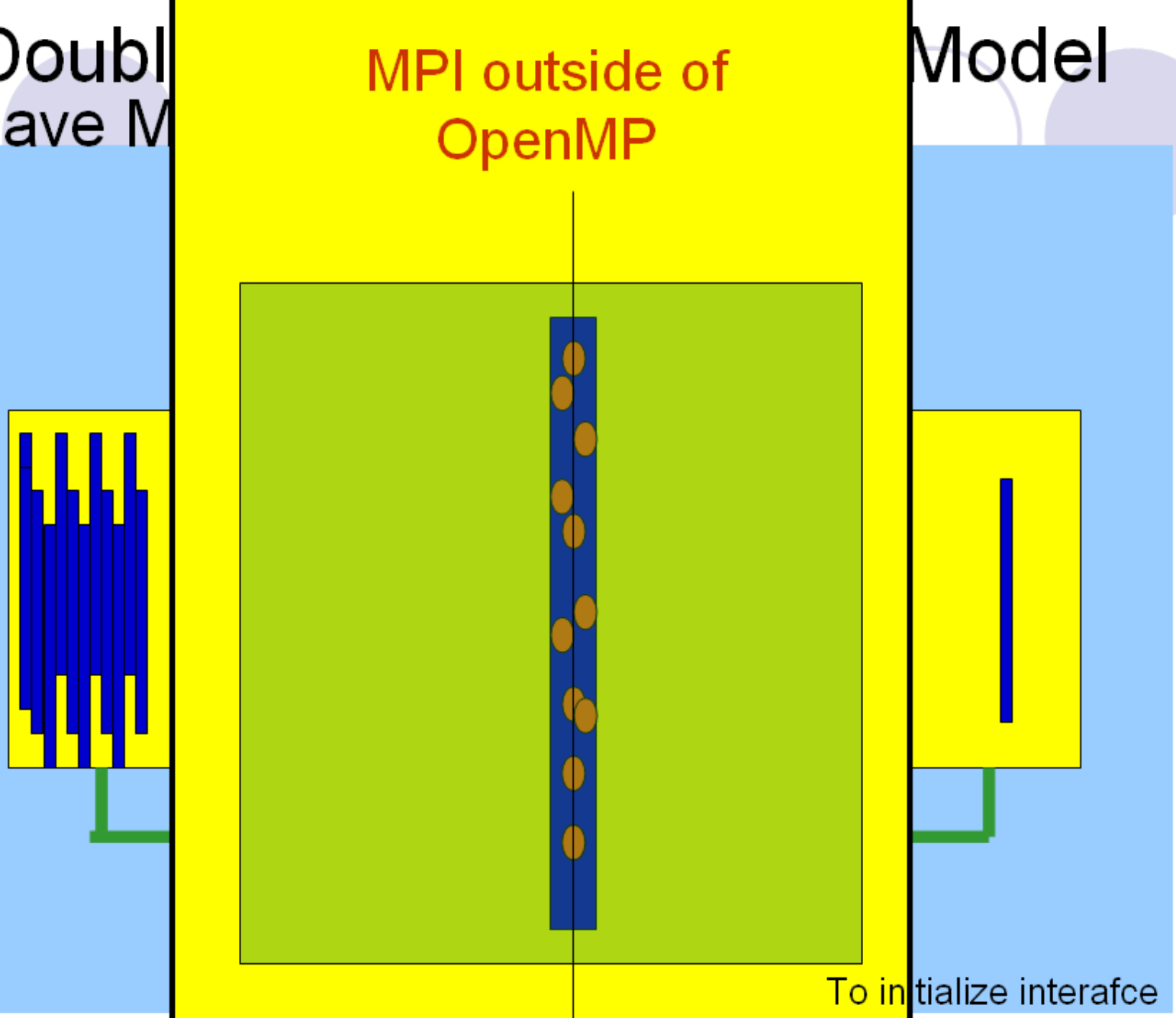


Double-layer Master-Slave Model

Slave Mode 1



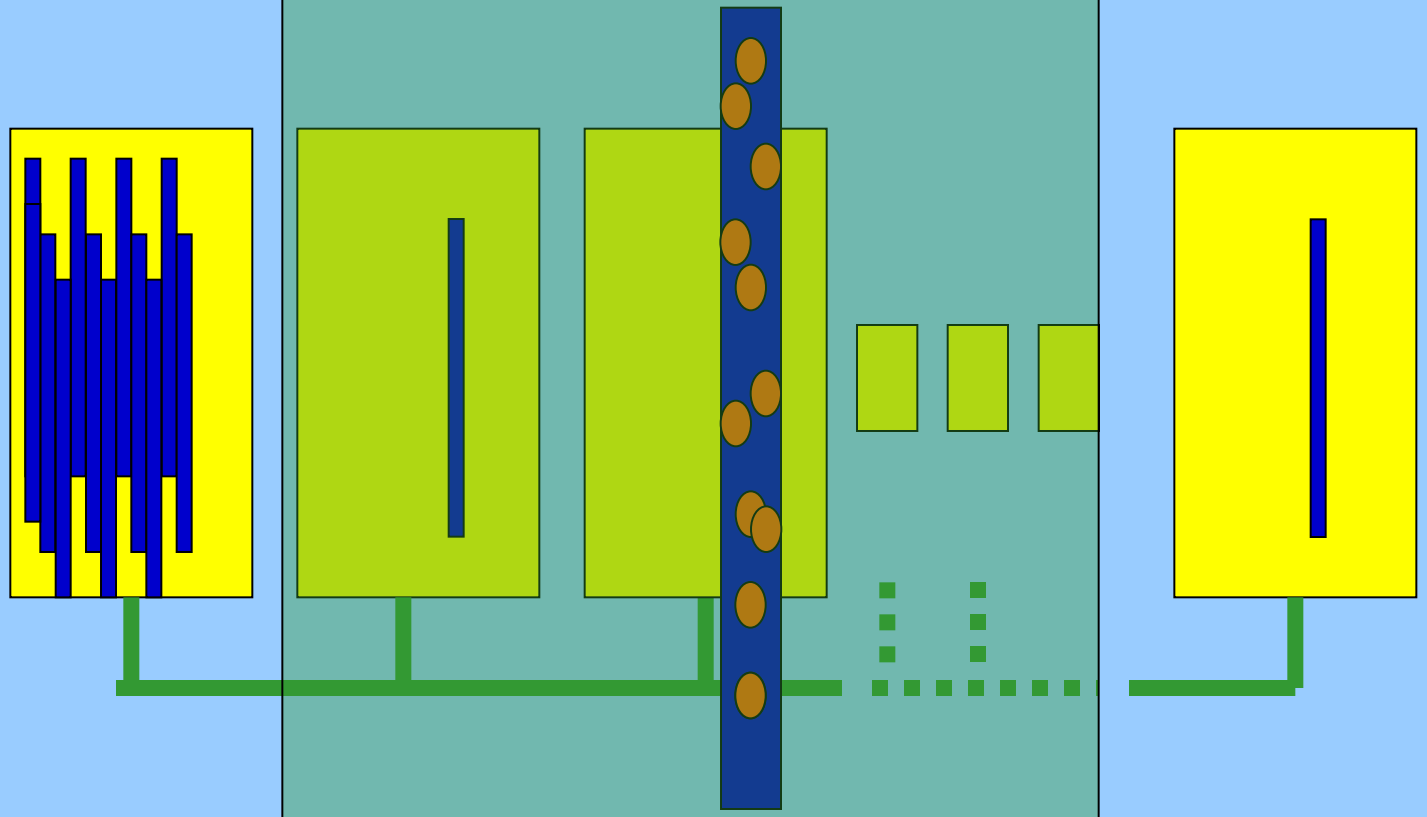
Doubl Model
Slave M



Double-layer Master-Slave Model

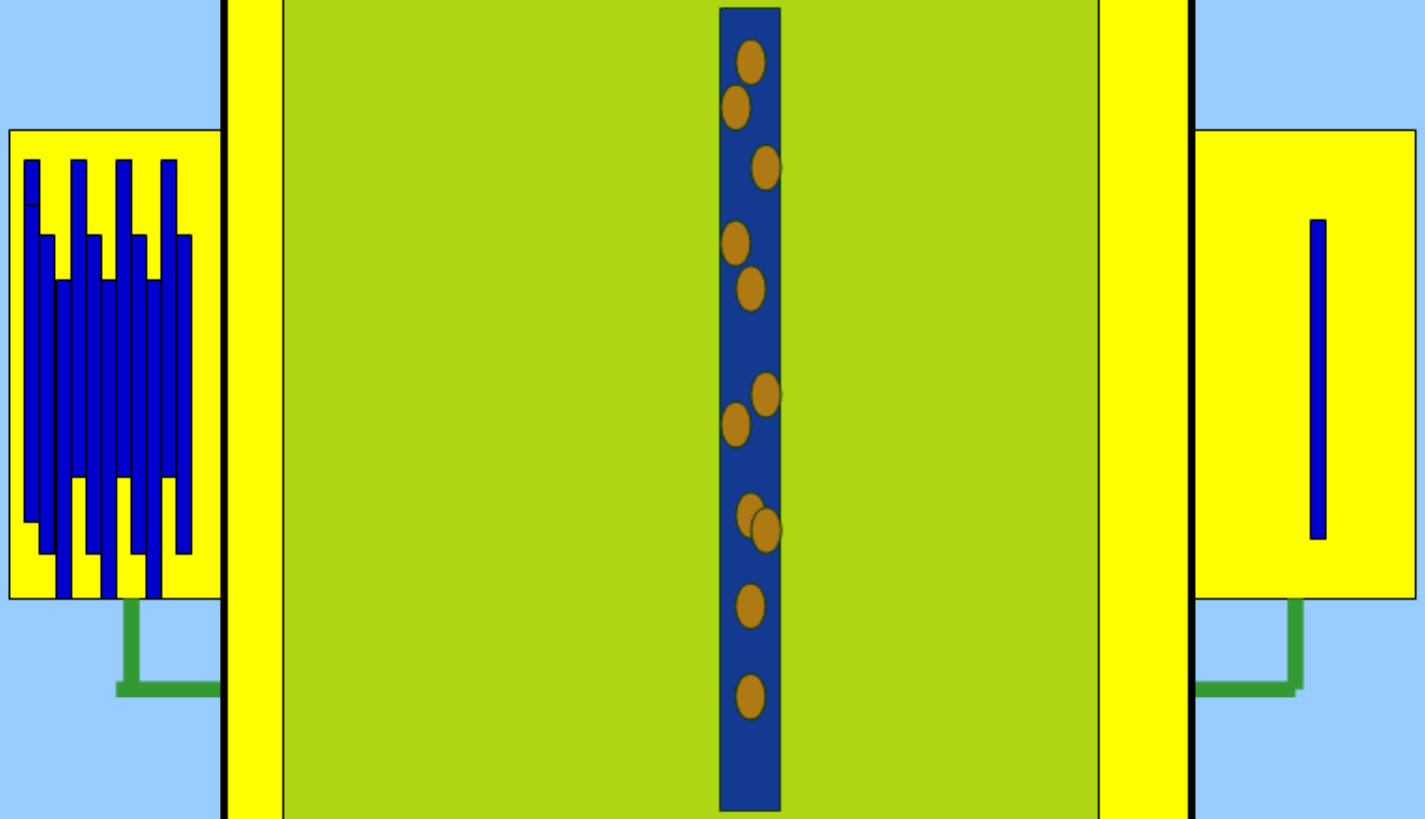
Slave Mode 2: anywhere is in OpenMP

Always in OpenMP, only main thread asks groups from the master.



Double Model
Slave MenMP

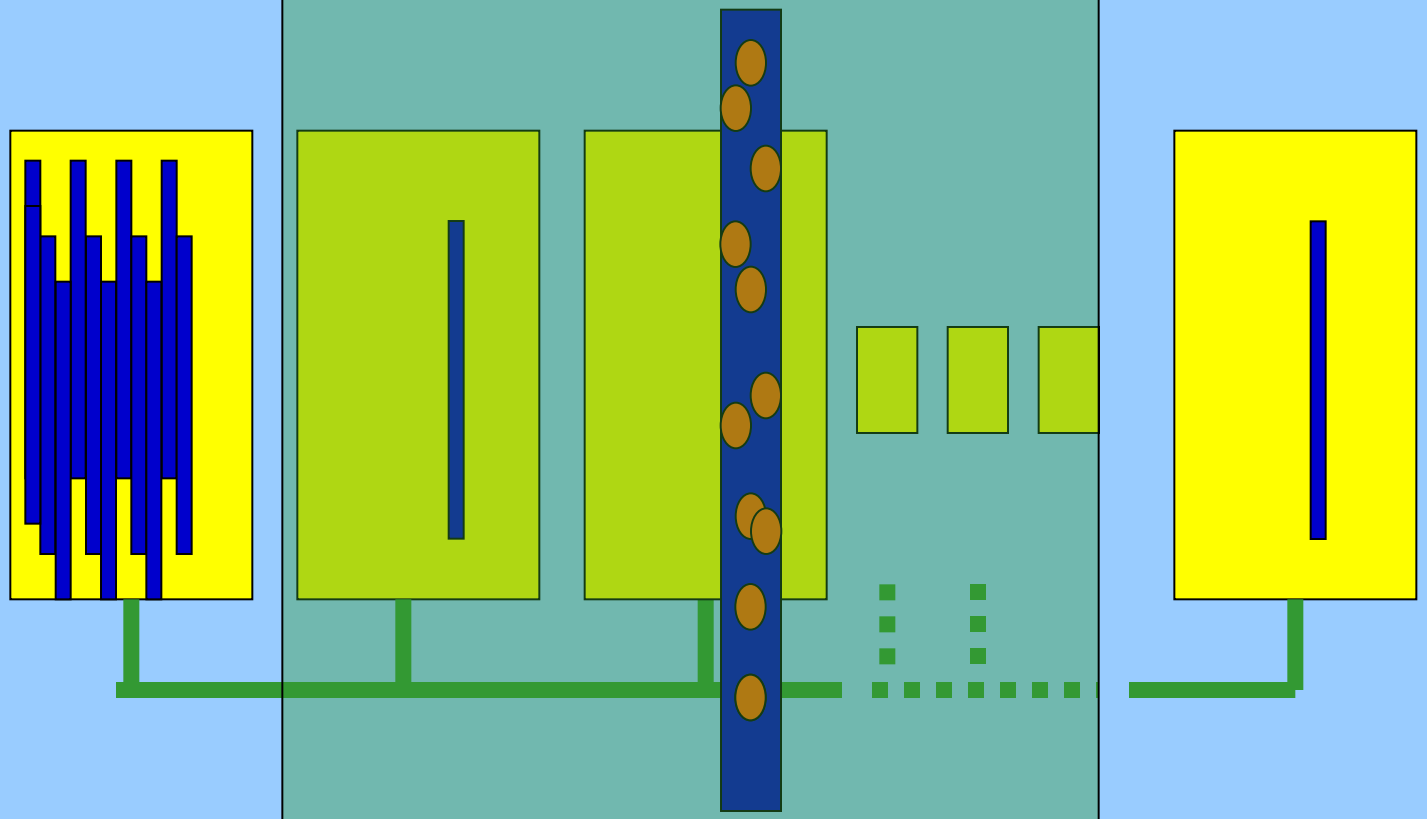
Always in OpenMP, only main thread asks groups from the master.



Double-layer Master-Slave Model

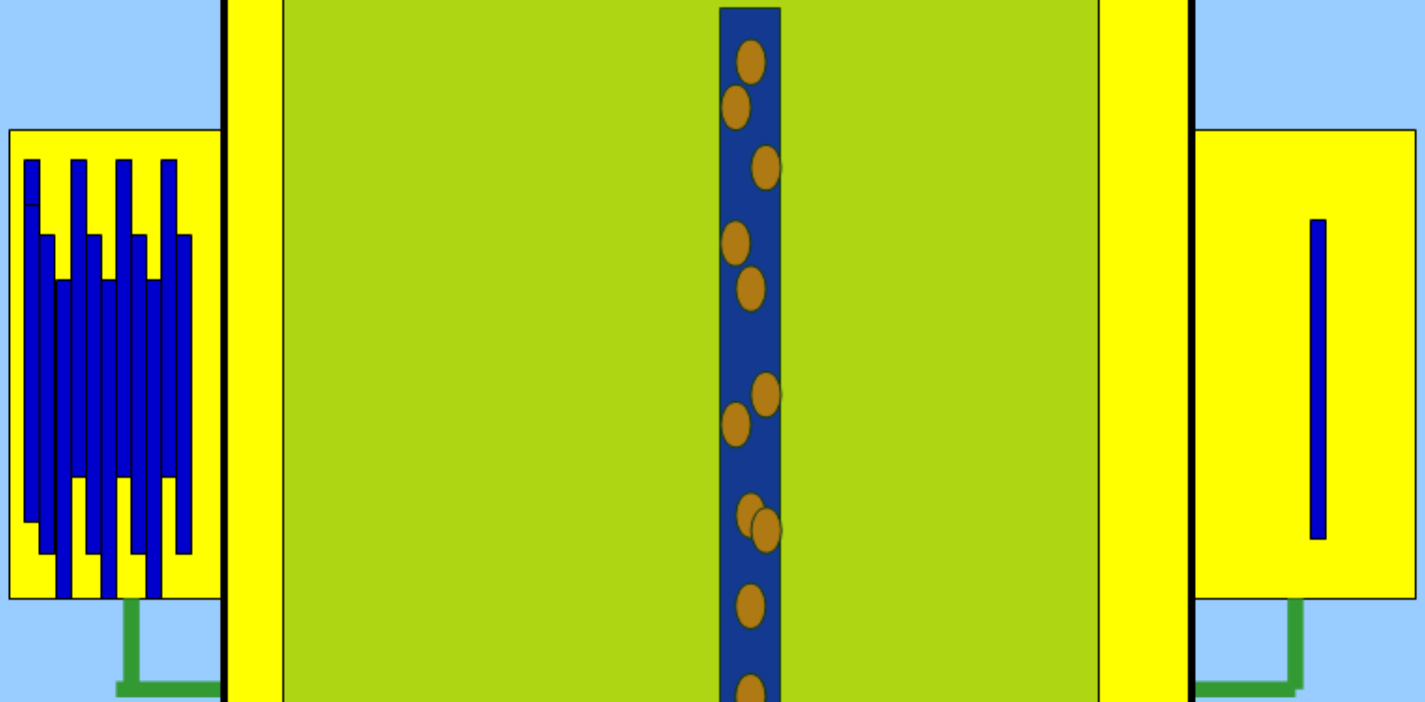
Slave Mode 3: anywhere is in OpenMP

Any thread asks groups from the master.



Double Model
Slave M enMP

Any thread asks groups from the master.



Double-layer Master-Slave Model

Normally, mode 3 for both Master and Slave nodes are suggested, for efficiency.

Double-layer Master-Slave Model

To use the DMSM library, users only need to code their own routines for data initialization, computing a job, calling this library, and managing the calculated results (including outputs).

Double-layer Master-Slave Model

Job (grouped) distributions and all kinds of communications associated with them are done by the library. The library also takes down how each job was done including CPU-time, checks and reports whether these operations were done normally at the end.

Double-layer Master-Slave Model

The DMSM library adopts a naming convention:

- inside the library, all names of constants, variables, routines, and files generated by the library begin with DMSM_ prefix.

Double-layer Master-Slave Model

The DMSM library files:

C/C++:
dmsm.c and dmsm.h

FORTRAN 90:
dmsm.f90 and dmsm.ctimer.c

Double-layer Master-Slave Model

The interface of a simple usage of DMSM library in Fortran 90

```
USE    DMSM_MODULE
```

then call the following routines in sequence:

```
CALL  DMSM_INITIALIZE(…)
CALL  DMSM_WORKING(…)
CALL  DMSM_JOB_DISTRIBUTION_CHECKUP()
CALL  DMSM_FINALIZE()
```

where the last two take no arguments.

The four integer arguments of DMSM_INITIALIZE in Fortran 90

```
THREADS_PER_PROCESS,    &
JOB_DISTRIBUTION_PLAN,  &
TOTAL_JOBS,             &
NUM_OF_JOBS_PER_GROUP
```

where the JOB_DISTRIBUTION_PLAN is master_mode*10+slave_mode.

The four arguments of DMSM_WORKING in Fortran 90

```
DO_THE_JOB(),                    &
JOB_GROUP_PREPARATION(),         &
RESULT_COLLECTION(),             &
RESULT_COLLECTION_ENABLED
```

where the first three are user supplied subroutines and the last is a logic variable.

The first argument of DMSM_WORKING in Fortran 90

DO_THE_JOB()
is, mandatory, to do user's specific jobs.
It takes JOB_SEQUENCIAL_NUMBER (from 1) as the only one argument.

The second argument of DMSM_WORKING in Fortran 90

JOB_GROUP_PREPARATION()
is, optional, to prepare the initial data for a job group, when the job group is assigned from the master to anybody.
It takes from_job, to_job, my_MPI_rank, and the_receiver_MPI_rank as the only four integer arguments.
Race condition addressed later.

The third argument of DMSM_WORKING in Fortran 90

RESULT_COLLECTION()

is, optional, to collect available calculated results to the master from anybody when they communicate. The fourth logic RESULT_COLLECTION_ENABLED must follow. If it is .TRUE., the collection happens during computation, otherwise only when all computation is done, inside our lib.

The third argument of DMSM_WORKING in Fortran 90

RESULT_COLLECTION()
takes MY_MPI_RANK, and
MPI_RANK_OF_RESULT_FROM
as the only two integer arguments.

Race condition addressed later.

DMSM_WORKING in Fortran 90

is overloaded.

Double-layer Master-Slave Model

The F90 interface of DMSM library

A wrapper for

```
CALL  DMSM_INITIALIZE()
CALL  DMSM_WORKING()
CALL  DMSM_JOB_DISTRIBUTION_CHECKUP()
CALL  DMSM_FINALIZE()
```

in sequence is also supplied as

```
SUBROUTINE DMSM_ALL()
```

with the combined arguments of

DMSM_INITIALIZE ()

and

DMSM_WORKING() in order.

Interface of DMSM_ALL() in F90

```
SUBROUTINE DMSM_ALL(
    THREADS_PER_PROCESS,          &
    JOB_DISTRIBUTION_PLAN,        &
    TOTAL_JOBS,                   &
    NUM_OF_JOBS_PER_GROUP,        &
    DO_THE_JOB(),                 &
    JOB_GROUP_PREPARATION(),      &
    RESULT_COLLECTION(),          &
    RESULT_COLLECTION_ENABLED     )
```

which is also overloaded.

Double-layer Master-Slave Model

The C interface of a simple usage of the DMSM library

```
#include "dmsm.h"
void DMSM_Initialize()
void DMSM_Working()
void DMSM_Job_Distribution_Checkup()
void DMSM_Finalize()
```

and the corresponding wrapper

```
void DMSM_All()
```

Double-layer Master-Slave Model

The C interface of DMSM library

```
void DMSM_Initialize(
        int Total_Number_Of_Threads_Per_Process,
        int Job_Distribution_Plan,
        int Total_Num_Of_Jobs,
        int Num_Of_Jobs_Per_Group
         );
```

Double-layer Master-Slave Model

The C interface of DMSM library

```
void DMSM_Working(
        void (*Do_The_Job) (int),
        void (*Job_Group_Preparation) (int,int,int,int),
        void (*Result_Collection) (int,int),
        int  Result_Collection_Enabled
         );
```

If no Job_Group_Preparation or Result_Collection, pass NULL, as C does not allow overloading.

Double-layer Master-Slave Model

The C interface of DMSM library

```
void DMSM_All(
        int Total_Number_Of_Threads_Per_Process,
        int Job_Distribution_Plan,
        int Total_Num_Of_Jobs,
        int Num_Of_Jobs_Per_Group
        void (*Do_The_Job) (int),
        void (*Job_Group_Preparation) (int,int,int,int),
        void (*Result_Collection) (int,int),
        int  Result_Collection_Enabled
         );
```

Double-layer Master-Slave Model

The C interface of DMSM library

```
void DMSM_Working or DMSM_ALL(
        …
        void (*Do_The_Job) (int),
        void (*Job_Group_Preparation) (int,int,int,int),
        …
         );
```

Job (sequential) numbers as arguments start from 0.

Double-layer Master-Slave Model

In all the following examples, the computational tasks/jobs are the same:

$$\sum_{i=0}^{M_1} \sqrt{i}, \quad \sum_{i=0}^{M_2} \sqrt{i}, \quad \sum_{i=0}^{M_3} \sqrt{i} \quad \ldots$$

Although the summation with the maximum upper limit includes all the rest, we regard all of the summation jobs as independent ones for demonstration purpose.

Double-layer Master-Slave Model

Example 1:

As all initial data is made available to any thread in any node beforehand, and all the calculated results is collected afterwards by the user, only the routine DO_THE_JOB() is supplied when the lib is called.

Click here for example 1 in C in F90

Lab Work II: Mixed/C(F90)/dmsm

Check example source code: dmsm.example1.c or dmsm.example1.f90

C compiling: mpiicc –O3 –fopenmp dmsm.c dmsm.example1.c

F90 compiling: mpiicc –c –O3 –fopenmp dmsm.ctimer.c

mpiifort –cpp -O3 -fopenmp dmsm.f90 dmsm.example1.f90 dmsm.ctimer.o

Run: cat in.dat

cat integers.dat

echo 2 TotalNumberOfOpenMPThreadsPerProcess

OMP_NUM_THREADS=2 mpirun –np 4 ./a.out

more DMSM_journal.txt

hh1 1 1

more DMSM_journal.txt

Double-layer Master-Slave Model

Example 2:

Suppose the user wants to prepare initial data for any job group only when the job group is assigned to a node dynamically, for any reason. Then a user's routine

JOB_GROUP_PREPARATION(…)

to do so, should also be passed into the library.

Double-layer Master-Slave Model

Example 2:

However there is an OpenMP race condition here. When one thread received a new job group and is trying to update the initial data for it, another thread of the same node may need the initial data for a job of the old job group. To avoid this problem, the DMSM library supplies two more routines, which should be called in the routines JOB_GROUP_PREPARATION() and DO_THE_JOB() respectively.

Double-layer Master-Slave Model

The F90 interface of DMSM library

SUBROUTINE DMSM_WAIT_FOR_INITIAL_LOCKS()

(should be called in the user's routine

JOB_GROUP_PREPARATION()

before updating any initial data of a job group)

SUBROUTINE DMSM_UNSET_AN_INITIAL_LOCK()

(should be called in the user's routine

DO_THE_JOB()

as soon as the initial data of a job is no longer needed, e.g. saved somewhere else.)

Double-layer Master-Slave Model

The C interface of DMSM library

void DMSM_Wait_For_Initial_Locks();

(should be called in the user's routine

JOB_GROUP_PREPARATION()

before updating any initial data of a job group)

void DMSM_Unset_An_Initial_Lock();

(should be called in the user's routine

DO_THE_JOB()

as soon as the initial data of a job is no longer needed, e.g. saved somewhere else.)

Double-layer Master-Slave Model

In fighting against this

OpenMP race condition

with OpenMP locks or any other mechanism employed, the DMSM library does all the rest jobs, including initialization, setting, and finalization.

Click here for example 2 in C

in F90

simpler

Lab Work III: Mixed/C(F90)/dmsm

Check example source code: dmsm.example2.c or dmsm.example2.f90

C compiling: mpiicc –O3 –fopenmp dmsm.c dmsm.example2.c

F90 compiling: mpiicc –c –O3 –fopenmp dmsm.ctimer.c

mpiifort –cpp -O3 -fopenmp dmsm.f90 dmsm.example2.f90 dmsm.ctimer.o

Run: cat in.dat

cat integers.dat

echo 2 TotalNumberOfOpenMPThreadsPerProcess

OMP_NUM_THREADS=2 mpirun –np 4 ./a.out

more DMSM_journal.txt

hh2 1 1

more DMSM_journal.txt

Double-layer Master-Slave Model

Since this situation of OpenMP race condition is not very straightforward, where different threads update/access the same shared data structure but in different areas of the code, we removed all MPI stuff from example 2 and cut it into a much shorter race example and a corresponding norace example, for OpenMP users.

An OpenMP Race Condition Example

Many independent jobs will be performed group by group, and inside each group, the all-slave model is used to distribute the jobs. When one thread wants to update the initial data for a new group assigned, another thread may still need the initial data for the old job group, then the race condition.

Click here for race example in C in F90

An OpenMP Race Condition Example

We will use OpenMP locks to remove this race condition. When a job is assigned to a thread, a lock identified by the thread number is set. Later the lock is unset by the thread when he decides the initial data for his specific job is no longer needed (e.g. saved somewhere else). Then any thread can update the initial data for a whole new job group safely, only if all locks are confirmed unset.

Click here for norace example in C in F90

The four arguments of DMSM_WORKING in Fortran 90

```
DO_THE_JOB(),                    &
JOB_GROUP_PREPARATION(),         &
RESULT_COLLECTION(),             &
RESULT_COLLECTION_ENABLED
```

where the first three are user supplied subroutines and the last is a logic variable.

Double-layer Master-Slave Model

Suppose the user wants to further collect computed results when a new job group is assigned to a node dynamically, then another user's routine Collect_Results(…)
to do so, should also be passed into the library.

For this purpose, let us assume in each node computed results are placed into its own temporary data structure T_structure, and later all of them are collected into a shared data structure F_structure but only in the master node.

Double-layer Master-Slave Model

Since the library is coded in such a way where Collect_Results(…) is always called by the library from a critical region if meaningful, the race condition on the F_structure is avoided, if it is accessible through Collect_Results(…) only.

Then let us consider if race condition on T_structure.

Double-layer Master-Slave Model

Example 3:

If T_structure is THREADPRIVATE,
no race condition on T_structure either.

Click here for example 3 in C

in F90

simpler

Double-layer Master-Slave Model

However, if T_structure is SHARED, OpenMP race condition on it would happen. Reason 1, many threads may update it at the same time in the computation. Reason 2, when one thread, in the Collect_Results(…) routine, is sending it to the master and deleting it when the sending is finished, another thread may be trying to update it in computation. Again, many threads may update the same data structure at the same time but in different areas of the code.

Double-layer Master-Slave Model

In order to avoid the OpenMP race condition on the SHARED T_structure, the library declares, initializes, and finalizes an additional lock, and users are expected to
CALL DMSM_SET_NODE_RESULT_LOCK()
(void DMSM_Set_Node_Result_Lock(); in C)
before updating any T_structure and
CALL DMSM_UNSET_NODE_RESULT_LOCK()
(void DMSM_Unset_Node_Result_Lock(); inC)
after the updating.

Double-layer Master-Slave Model

Example 4:

CALL DMSM_SET_NODE_RESULT_LOCK()
(void DMSM_Set_Node_Result_Lock(); in C)
before updating any T_structure and
CALL DMSM_UNSET_NODE_RESULT_LOCK()
(void DMSM_Unset_Node_Result_Lock(); inC)
after the updating
to avoid race condition in result collection dynamically.

Click here for example 4 in C in F90

simpler

Double-layer Master-Slave Model

Commands to test in F90

For compiling

```
use studio12u3
mpif90 -dalign -fast -xopenmp -C \
   dmsm.f90 User_source_files.f90 -lm
```

To run

```
echo THREADS > \
   TotalNumberOfOpenMPThreadsPerProcess
mpirun -np PROCESSES x THREADS ./a.out
```

or

hi PROCESSES THREADS (for example i=1,2,3,4)

Double-layer Master-Slave Model

Commands to test in C

For compiling

```
use studio12u3
mpicc -dalign -fast -xopenmp \
   dmsm.c User_source_files.c –lm
```

To run

```
echo THREADS > \
   TotalNumberOfOpenMPThreadsPerProcess
mpirun -np PROCESSES x THREADS ./a.out
```

or

hi PROCESSES THREADS (for example i=1,2,3,4)

Lab Work IV: Mixed/C(F90)/dmsm

Check example source code: dmsm.exampleX.c or dmsm.exampleX.f90

C compiling: mpiicc –O3 –fopenmp dmsm.c dmsm.exampleX.c

F90 compiling: mpiicc –c –O3 –fopenmp dmsm.ctimer.c

mpiifort –cpp -O3 -fopenmp dmsm.f90 dmsm.exampleX.f90 dmsm.ctimer.o

Run: cat in.dat

cat integers.dat

echo 2 TotalNumberOfOpenMPThreadsPerProcess

OMP_NUM_THREADS=2 mpirun –np 4 ./a.out

more DMSM_journal.txt

hhX PROCESSES THREADS (for example X=1,2,3,4)

more DMSM_journal.txt

Double-layer Master-Slave Model

Three special cases in the DMSM lib

I, Total_processes=1 & Total_threads_per_node=1, then serial;

II, Total_processes=1 & Total_threads_per_node>1, then pure OpenMP all-slave model;

III, Total_processes>1 & Total_threads_per_node=1, then pure MPI Master-Slave model.

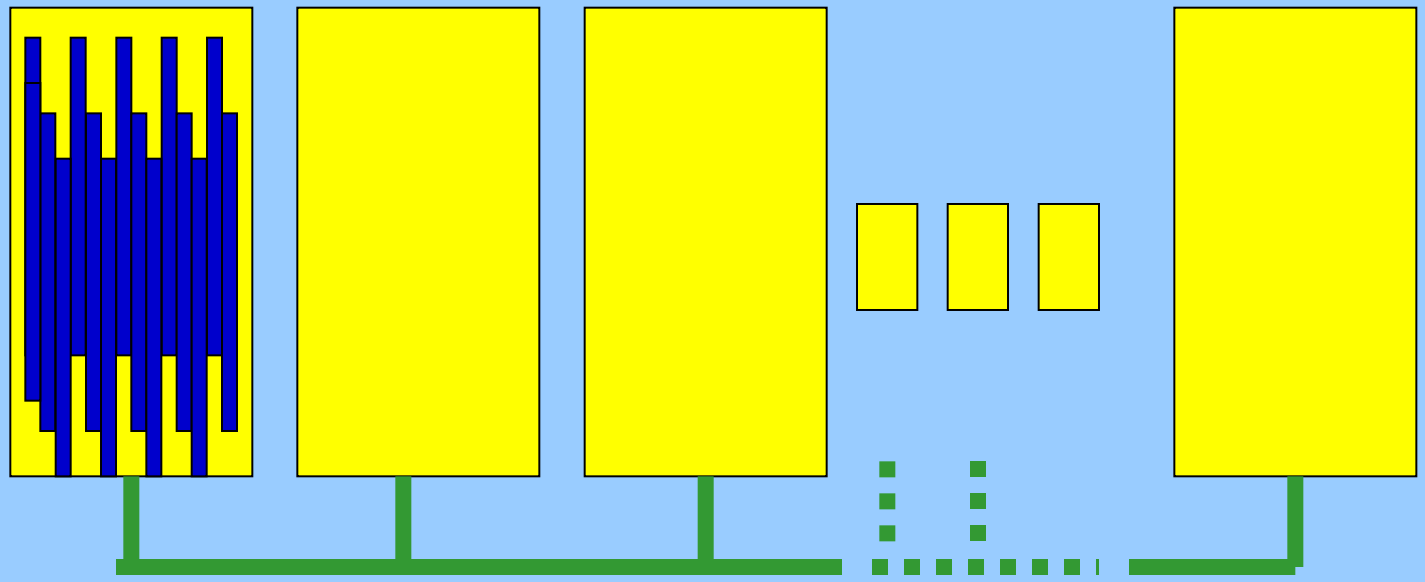
Double-layer Master-Slave Model

Extended usage of the DMSM lib based on the special case III

Total_processes>1 & Total_threads_per_node=1, then pure MPI Master-Slave model.

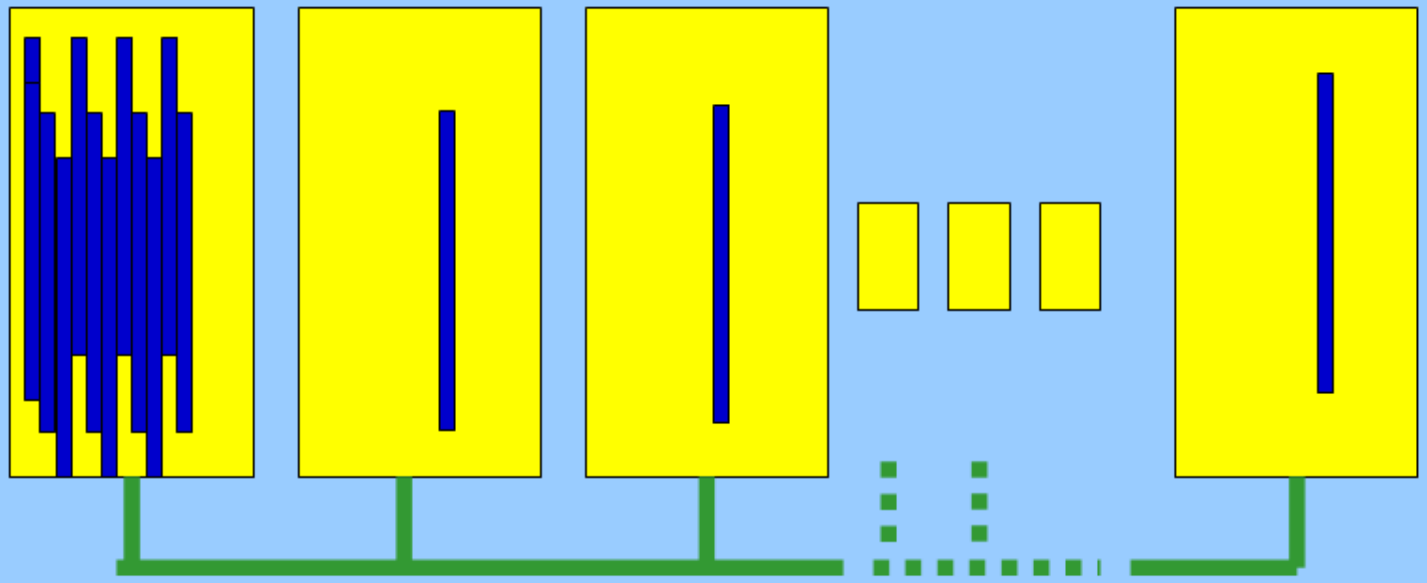
Double-layer Master-Slave Model

Job groups sent to nodes via
MPI master-slave model



Double-layer Master-Slave Model

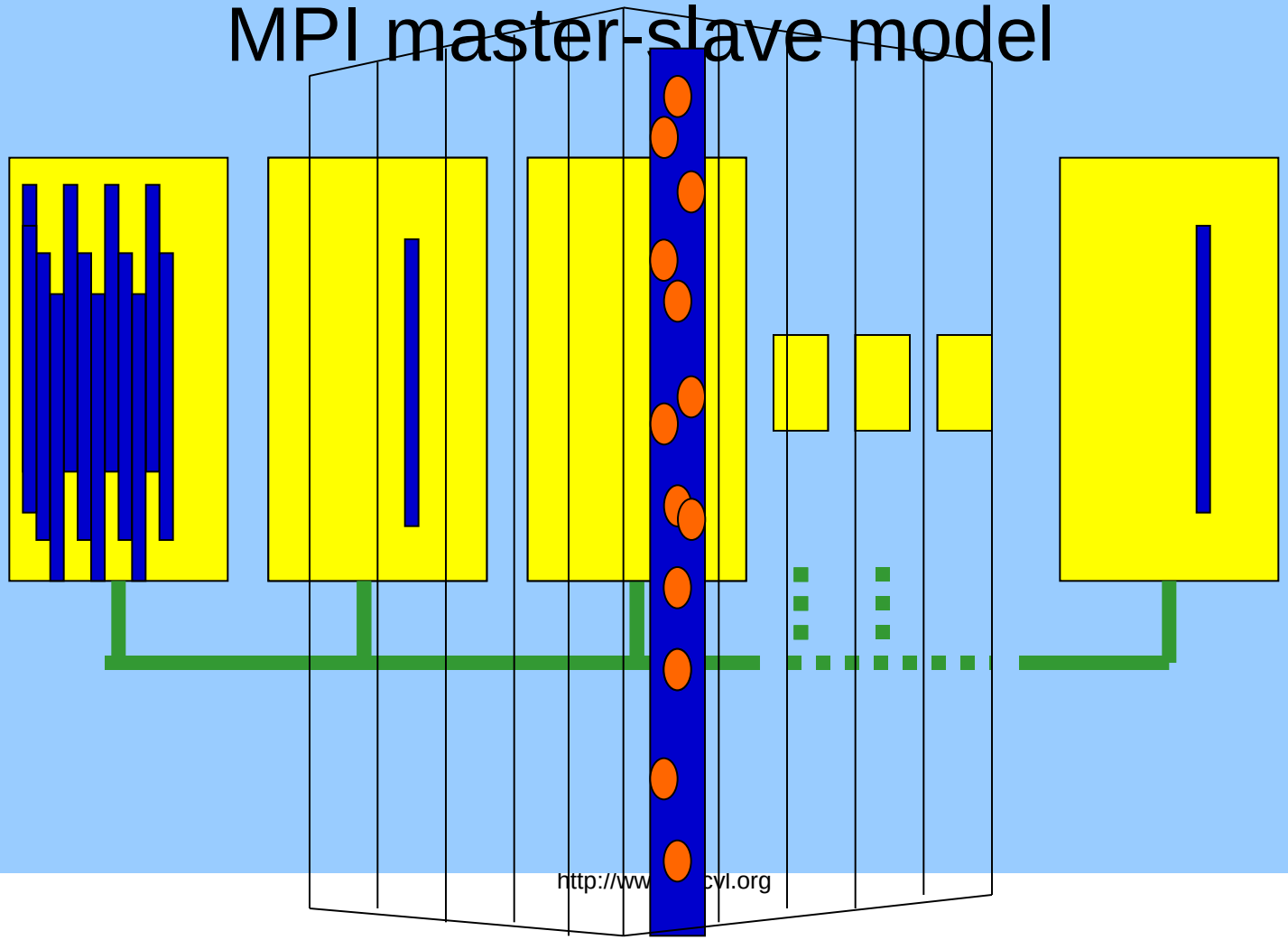
Job groups sent to nodes via
MPI master-slave model



Double-layer Master-Slave Model

Jobs in a group executed in the node by threads via an OpenMP all-slave model

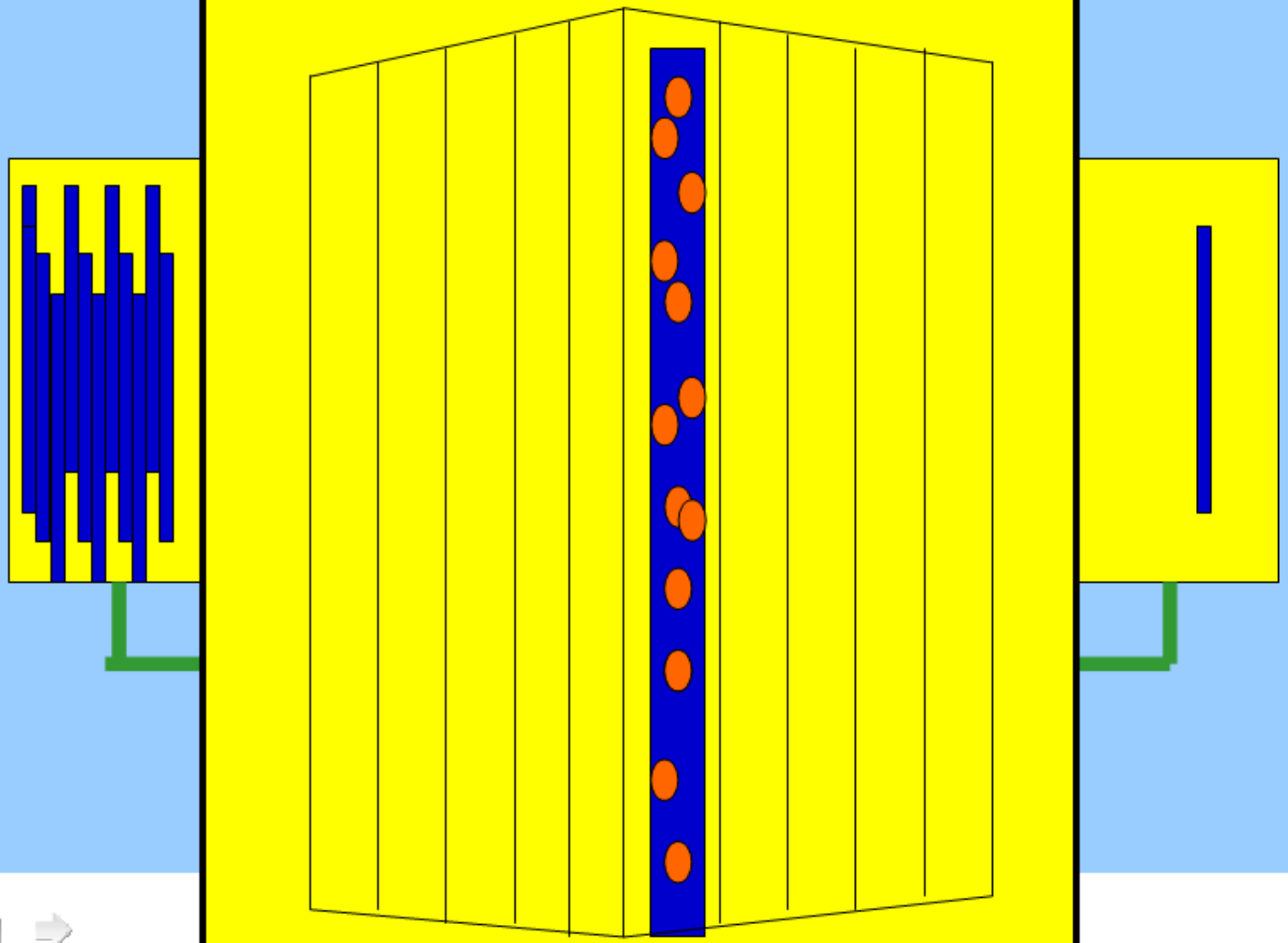
Job groups sent to nodes via MPI master-slave model



Doubl Model

Jo a

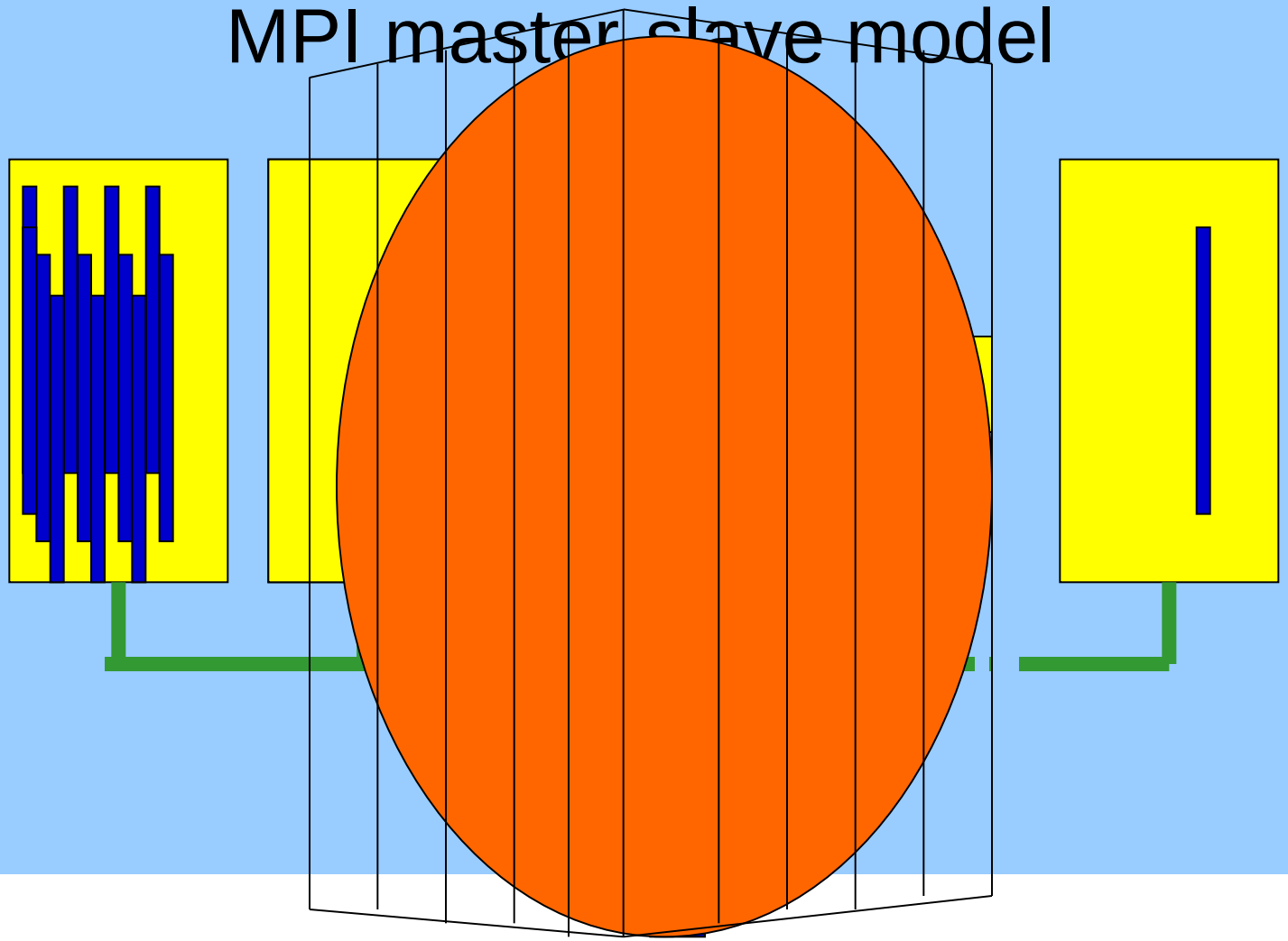
Jobs in a group executed in the node by threads via an OpenMP all-slave model



Double-layer Master-Slave Model

Now each job is executed by all threads of the node, in parallel.

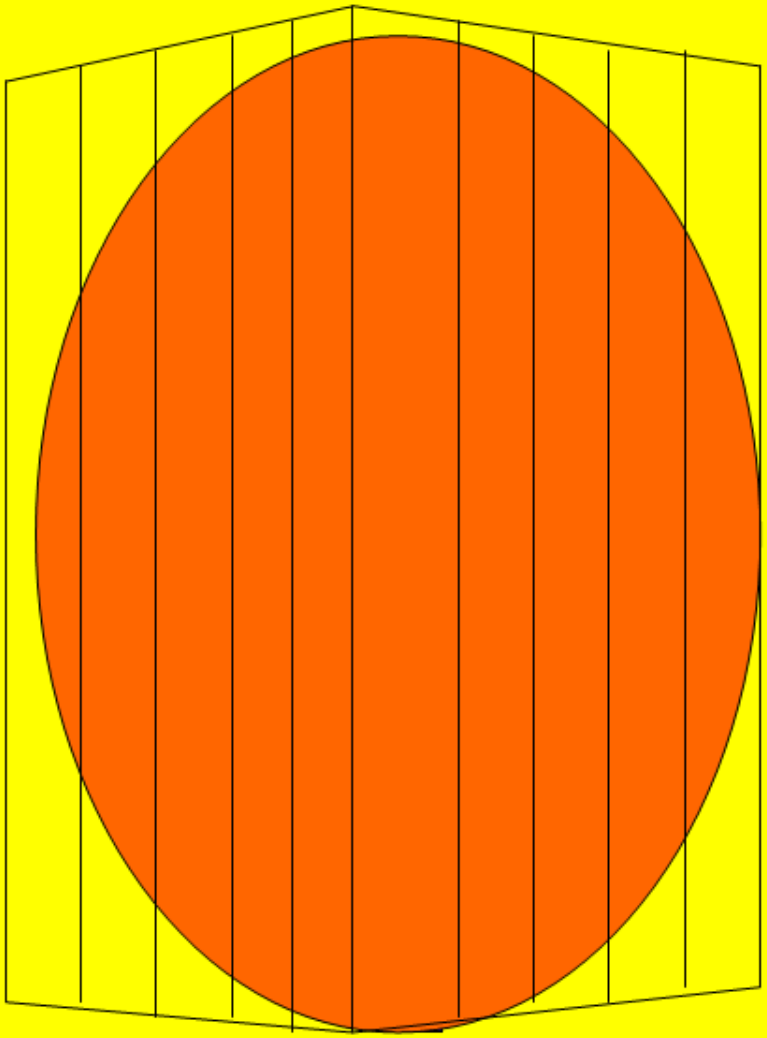
Job groups sent to nodes via MPI master slave model



Doubl Model

Jo a

Now each job is executed by all threads of the node, in parallel.



Double-layer Master-Slave Model

Key points for OpenMP parallelized jobs

1, Total_threads_per_node is set as 1, when the lib is called, although not true in reality;

2, All OpenMP parallelization is done inside the user's DO_THE_JOB() routine.

Double-layer Master-Slave Model

OpenMP job example:

still square root summation

Click here for the example in C in F90

Double-layer Master-Slave Model

Use script file to test it in F90 and C

hho PROCESSES THREADS

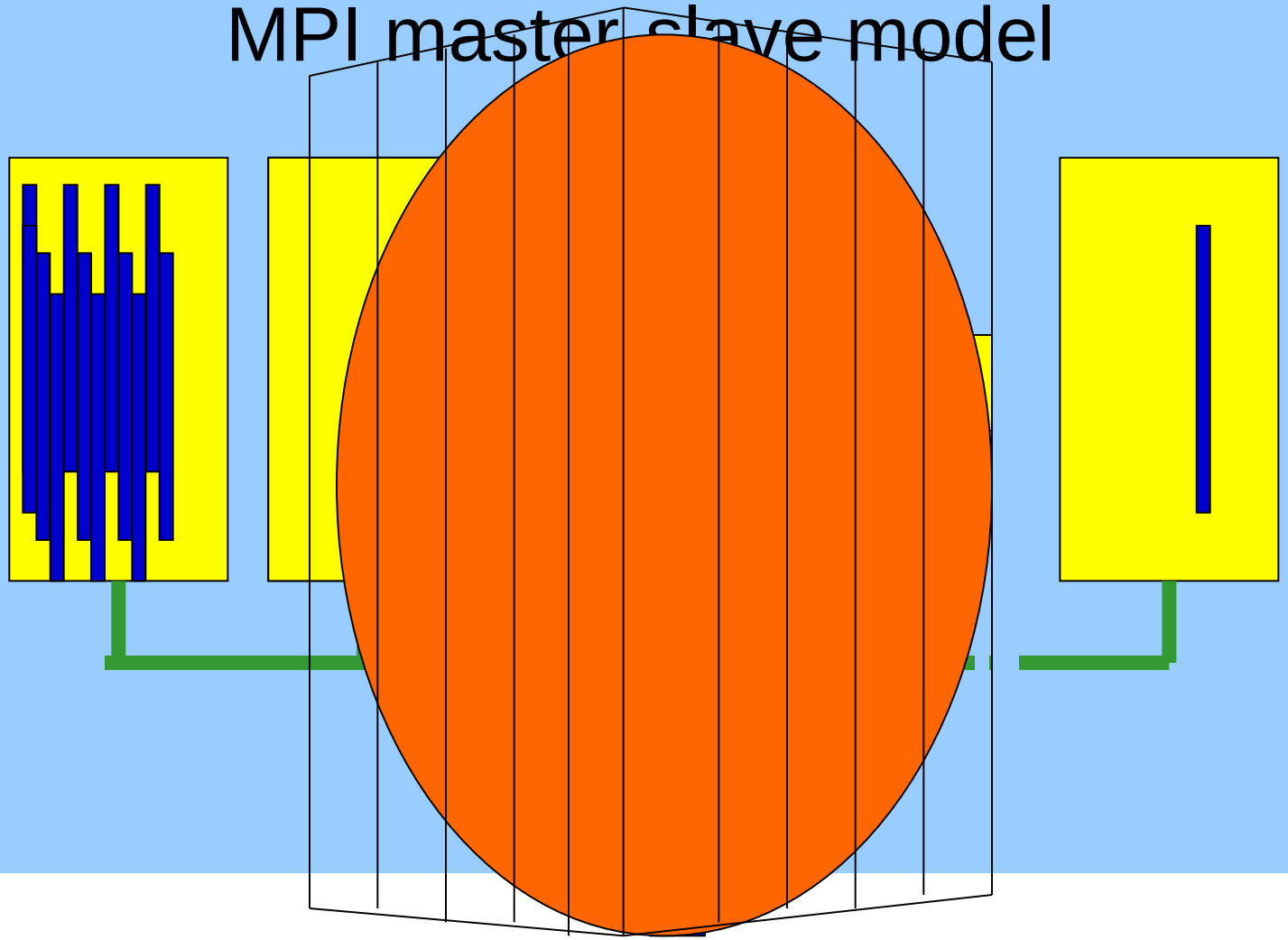
Double-layer Master-Slave Model

Then a normal existing OpenMP parallel code for a certain job can be converted into a user's DO_THE_JOB() routine, then call the DMSM lib of special case III to get an additional outer layer parallelism to submit many OpenMP parallel jobs together.

Double-layer Master-Slave Model

Now each job is executed by some MPI processes in parallel.

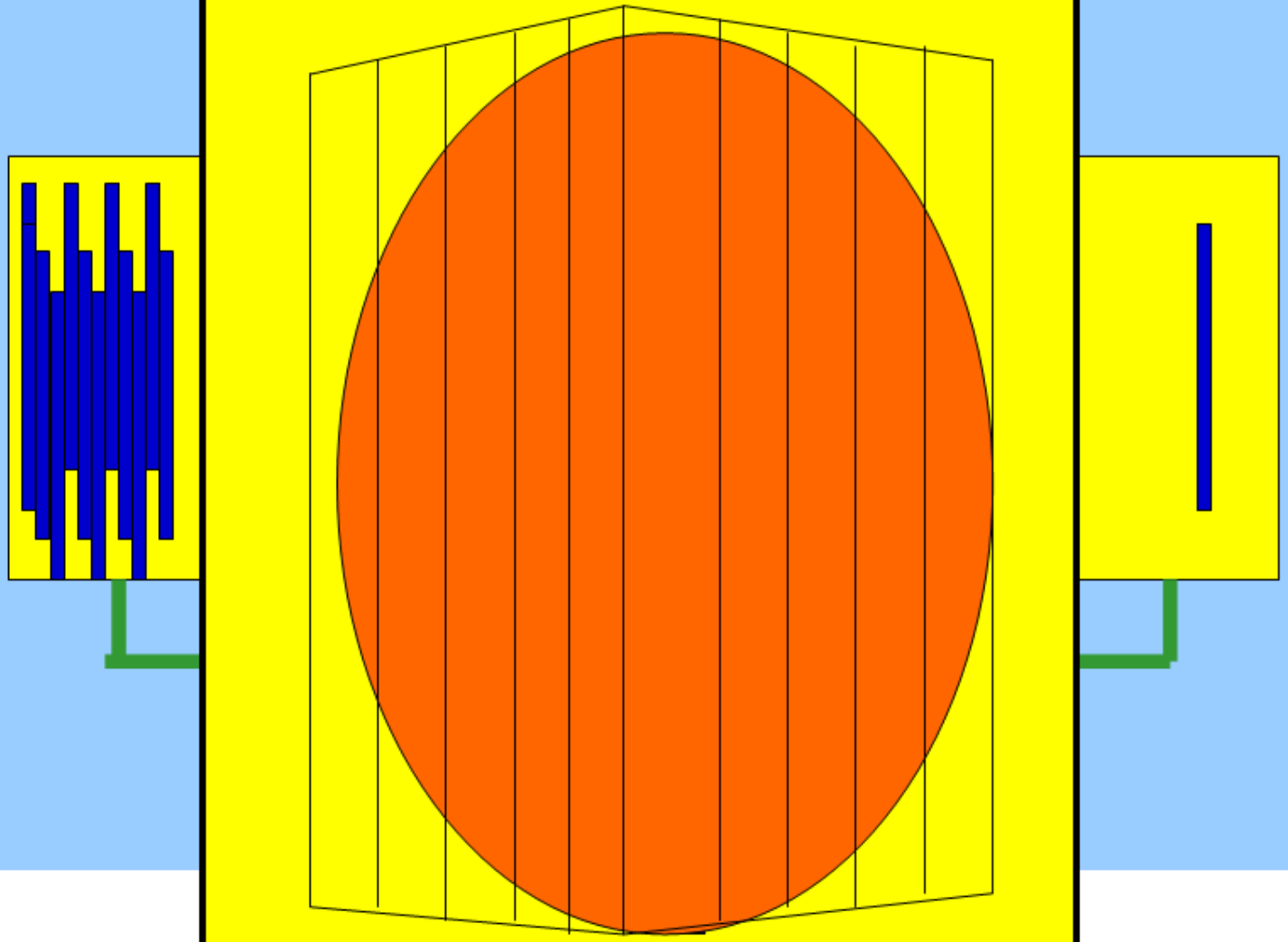
Job groups sent to nodes via MPI master slave model



Doubl Model

Jo a

Now each job is executed by some MPI processes in parallel.



Double-layer Master-Slave Model

An example for MPI parallelized jobs of DMSM lib

Total 81 processes:
1 process is the master to assign job groups to the rest processes;
every 10 of the rest 80 processes do a job together in MPI parallelism.

Double-layer Master-Slave Model

In such a situation, job communicators of processes performing every jobs in parallel are needed.

Double-layer Master-Slave Model

Additionally, a “dist communicator” of all processes with rank 0 from all job communicators for job (group) distribution is also necessary.

Double-layer Master-Slave Model

Extended interfaces for MPI parallelized jobs of DMSM lib in F90

```
CALL  DMSM_GEN_COMM_MPI_ALL(                          &
                                Job_processes_expected, &
                 All_comm,  All_rank,     All_processes, &
                Job_comm, Job_rank,  Job_processes, &
                Dist_comm, Dist_rank, Dist_processes   )

...
CALL DMSM_MPI_ALL(     Total_jobs,                    &
                       Num_of_jobs_per_group,  &
                       Do_my_job,                     &
                       Prepare_for_a_job_group, &
                 Collect_results, Enable_it_or_not)
```

Double-layer Master-Slave Model

Extended interfaces for MPI parallelized jobs of DMSM lib in C

```
void DMSM_Gen_Comm_MPI_All(        int Toal_Processes_For_a_Job,
        MPI_Comm    All_Comm, int    *All_Rank, int     *All_Processes,
        MPI_Comm *Job_Comm, int  *Job_Rank, int   *Job_Processes,
        MPI_Comm *Dist_Comm, int  *Dist_Rank, int *Dist_Processes);
```

…

```
void DMSM_MPI_All(                                  int  Total_Num_Of_Jobs,
                                                int  Num_Of_Jobs_Per_Group,
                                                void (*Do_The_Job) (int),
                                void (*Job_Group_Preparation) (int,int,int,int),
                                                void (*Result_Collection) (int,int),
                                                int  Result_Collection_Enabled);
```

Double-layer Master-Slave Model

MPI example:

still square root summation

Click here for the example in C in F90

Double-layer Master-Slave Model

Use script file to test it in F90 and C

hhm Total_processes processes_per_job

Double-layer Master-Slave Model

Then a normal existing MPI parallel code for a certain job can be converted into a user's DO_THE_JOB() routine, then call the DMSM lib of special case III to get an additional outer layer parallelism to submit many MPI parallel jobs together.

Double-layer Master-Slave Model

The examples and the library are made public available at:

http://www.hpcvl.org/misc/dmsm/dmsm.html

Thank you very much for your attention!

HAVE A NICE DAY!