Programming on the Grid using GridRPC

SEAIP 2008 & 4th PRAGMA Inst.

Dec 4, 2008

Yoshio Tanaka

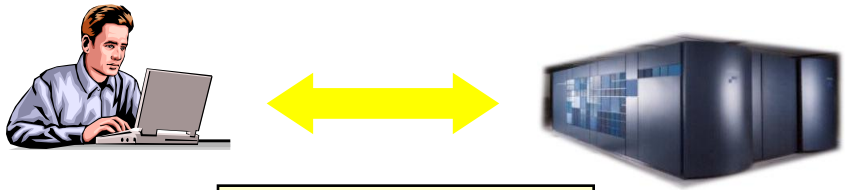
AIST, Japan

Menu

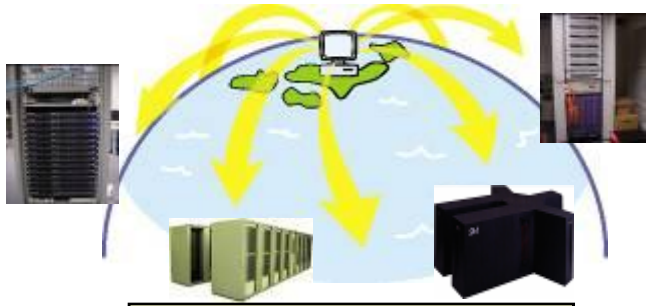
- Introduction of GridRPC and Ninf-G (～30 minutes)
- Practical (60～ minutes)

GridRPC: A programming model based on RPC

- **What is GridRPC?**
  - Utilize remote procedure call (RPC) on the Grid
  - Based on client-server model
  - The GridRPC API is published as a proposed recommendation (GFD-R.P 52) at the GGF
- **Usage scenarios**
  - Remote library call
    - Executing compute-intensive tasks on a remote high performance computing resource
  - Task parallel processing
    - Executing large numbers of independent tasks on distributed computing resources



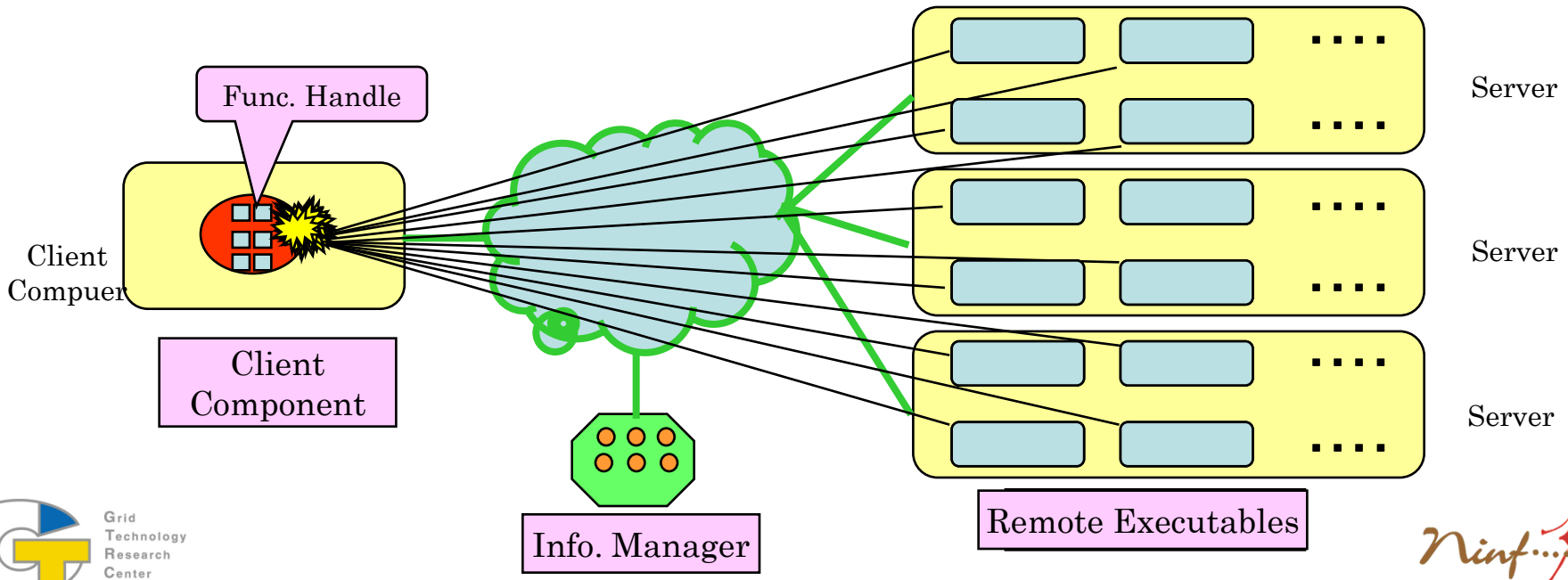
Remote Library Call



Task parallel processing

GridRPC Model

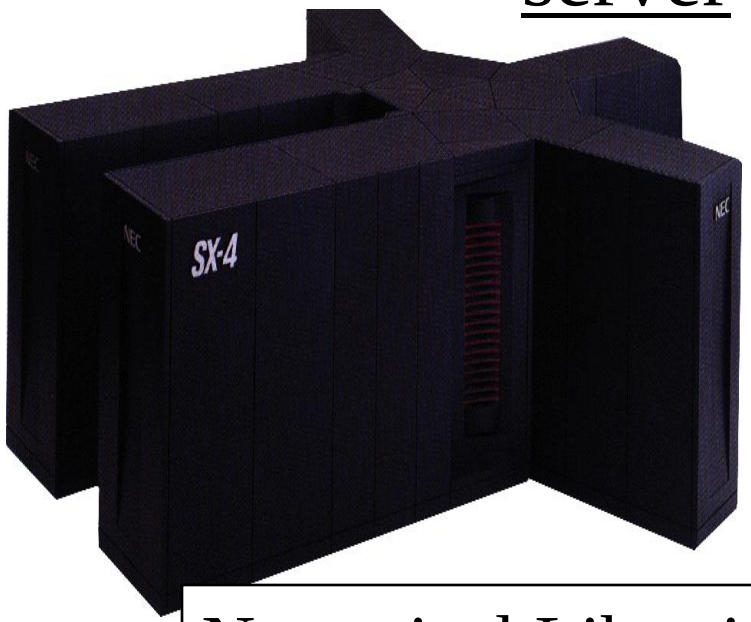
- **Client Component**
  - Caller of GridRPC.
  - Manages remote executables via function handles
- **Remote Executables**
  - Callee of GridRPC.
  - Dynamically generated on remote servers.
- **Information Manager**
  - Manages and provides interface information for remote executables.



Typical scenario 1: desktop supercomputing

- **Utilize remote supercomputers from your desktop computer**
- **Reduce cost for maintenance of libraries**
- **ASP-like approach**

server



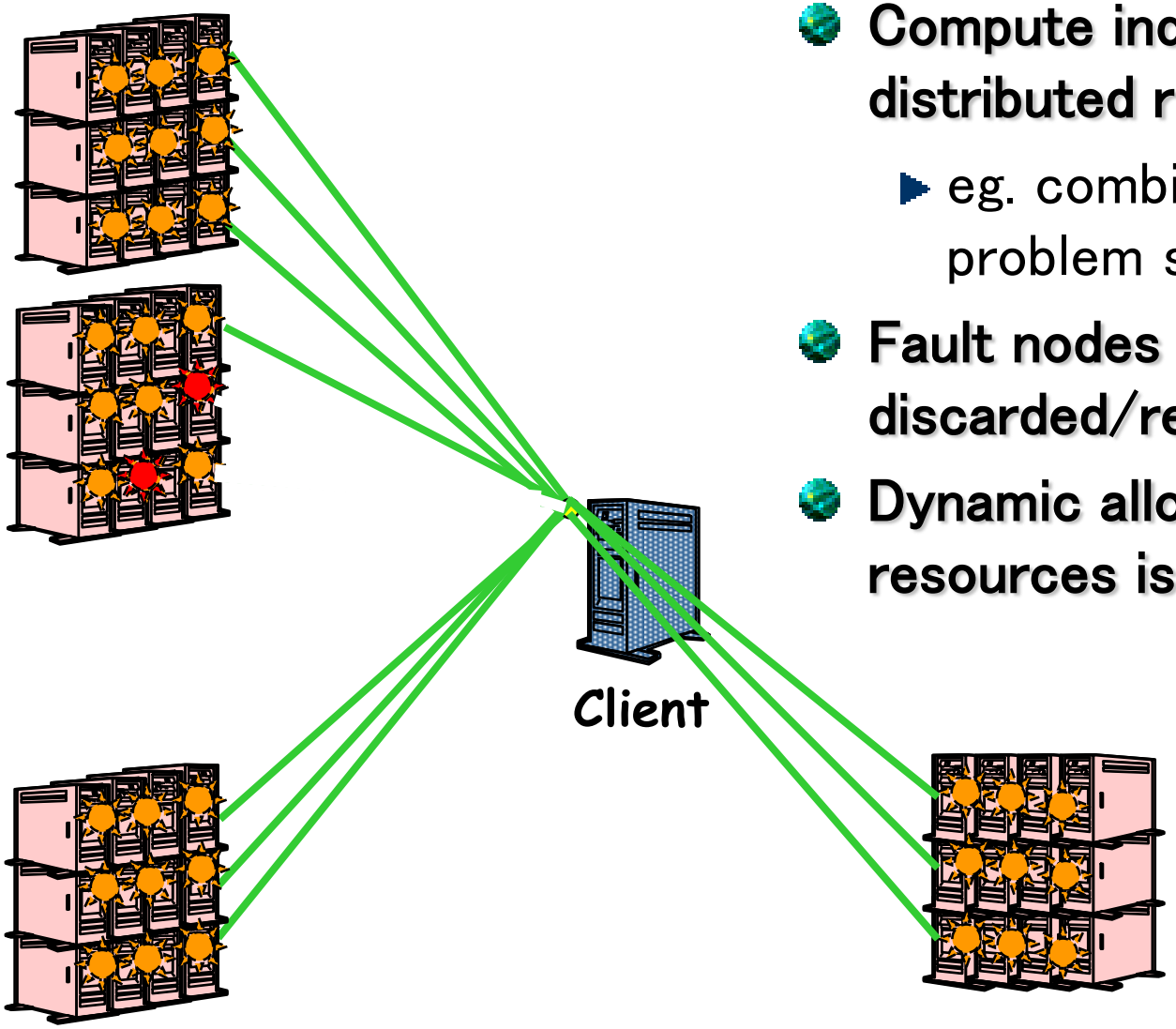
client

arguments

results

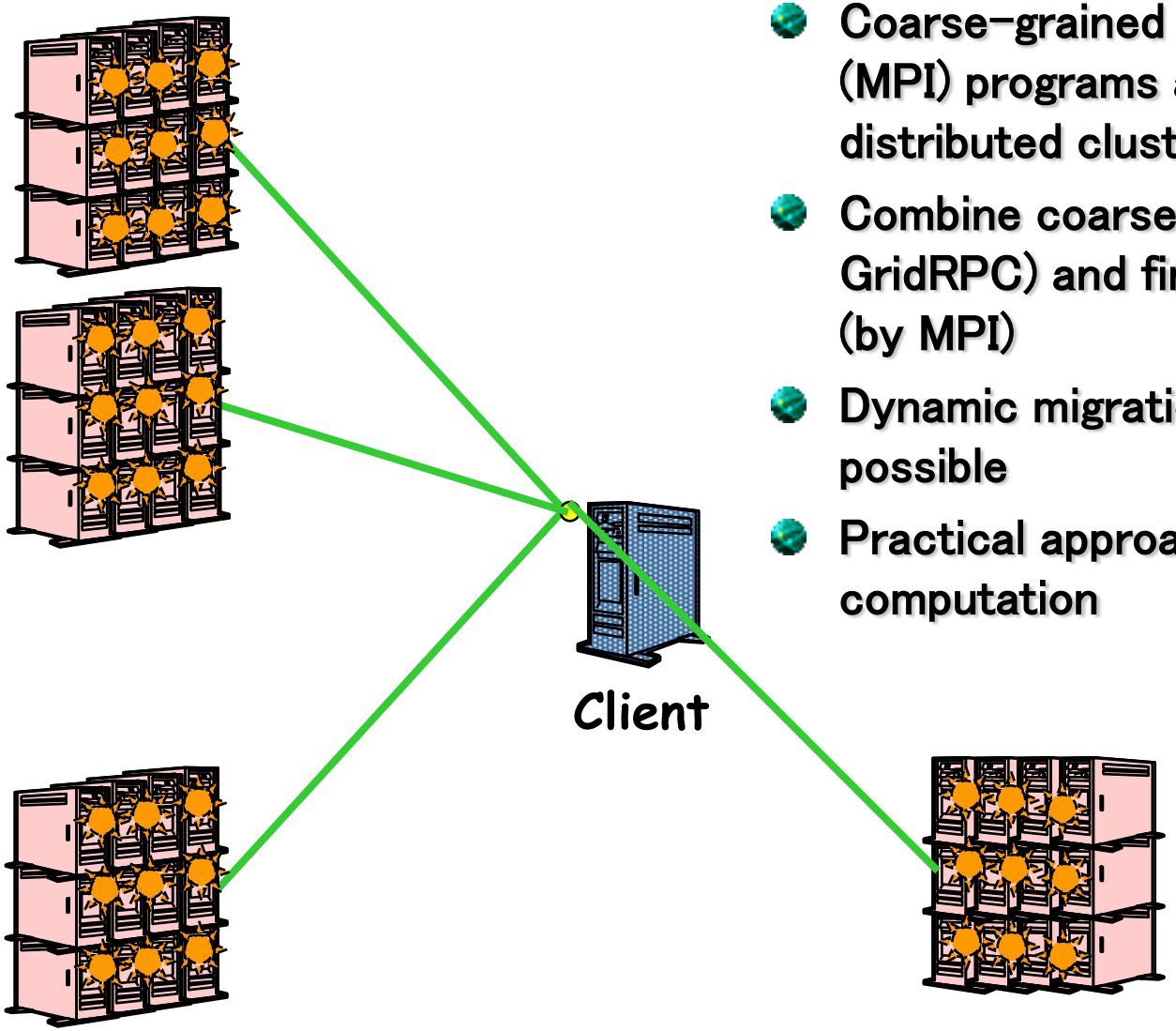
Numerical Libraries
Applications

Typical scenario 2: parameter surevey



- **Compute independent tasks on distributed resources**
  - eg. combinatorial optimization problem solvers
- **Fault nodes can be discarded/retried**
- **Dynamic allocation / release of resources is possible**

Typical scenario 3: GridRPC + MPI



- Coarse-grained independent parallel (MPI) programs are executed on distributed clusters
- Combine coarse-grained parallelism (by GridRPC) and fine-grained parallelism (by MPI)
- Dynamic migration of MPI jobs is possible
- Practical approach for large-scale computation

GridRPC v.s. MPI

	GridRPC	MPI
parallelism	task parallel	data parallel
model	client/server	SPMD
API	GridRPC API	MPI
co-allocation	dispensable	indispensable
fault tolerance	good	poor (fatal)
private IP nodes	available	unavailable
resources	can be dynamic	static *
others	easy to gridify existing apps.	well known seamlessly move to Grid

* May be dynamic using process spawning

Features of GridRPC

- **GridRPC enables us to develop flexible, fault tolerant, and efficient applications on Grid and Cloud.**
  - Flexibility: allowing dynamic resource allocation/migration
  - Fault tolerance: detecting errors and recovering from faults automatically for a long run
  - Efficiency: managing thousands of CPUs

- **The GridRPC API is the first final recommendation, an OGF standard (GFD-R.52).**

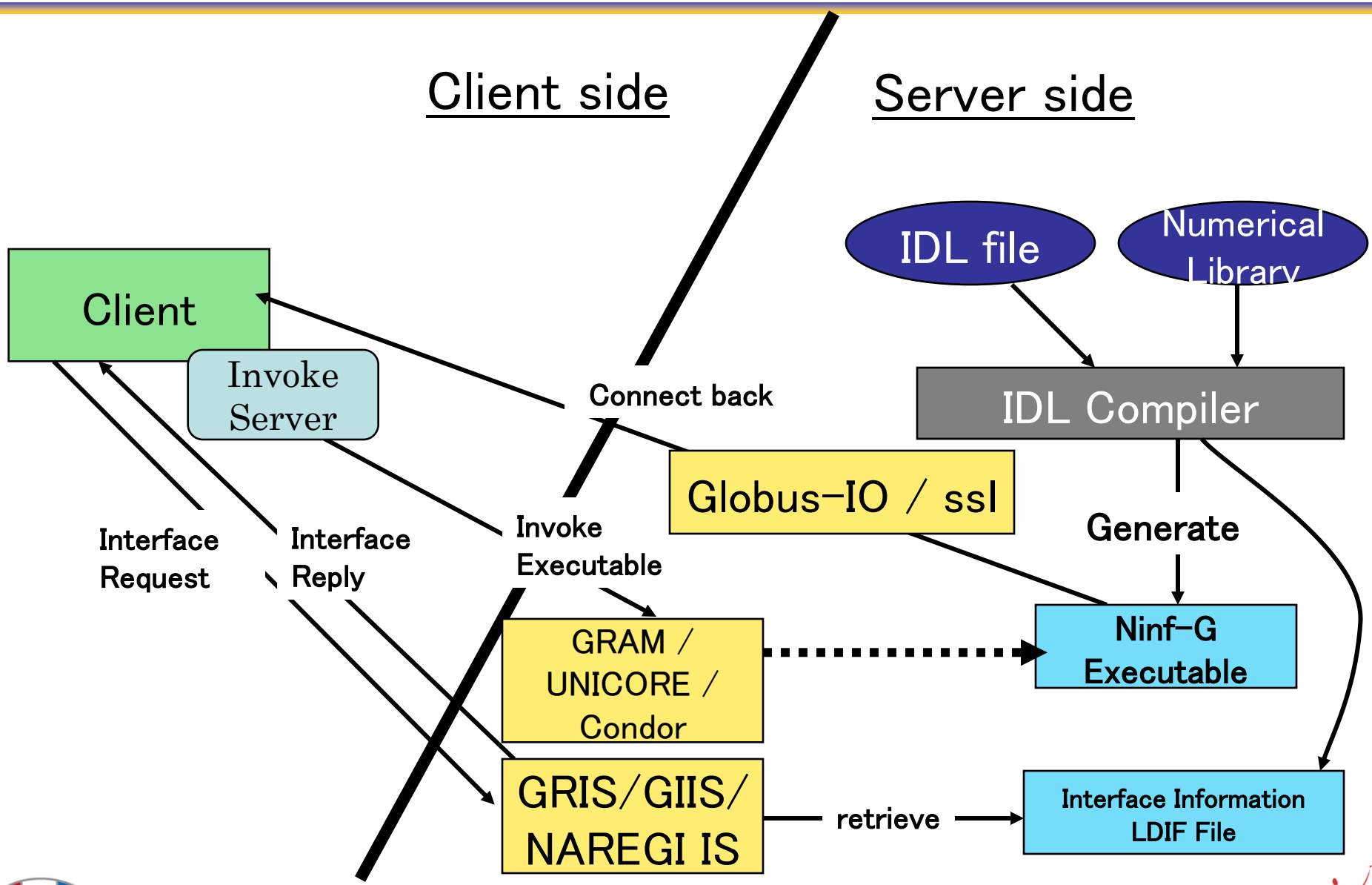
Ninf Project

- **Started in 1994**
- **Collaborators from various organizations**
  - AIST
    - Satoshi Sekiguchi
    - Hidemoto Nakada
    - Yoshio Tanaka
    - Atsuko Takefusa
    - Yusuke Tanimura
    - Hiroshi Takemiya
  - University of Tsukuba
    - Mitsuhisa Sato
    - Taisuke Boku
    - Osamu Tatebe
  - Tokyo Institute of Technology
    - Satoshi Matsuoka
    - Kento Aida
  - Tokyo Denki University
    - Katsuki Fujisawa

Ninf-G: A Reference Implementation of the GridRPC API

- **Two major versions**
  - Version 4 (Ninf-G4)
    - Enables RPC via various middleware including
      - GT2 GRAM, GT4 GRAM, Condor, UNICORE, NAREGISS, ssh
    - Any other middleware can be used for RPC by implementing interface modules
    - Latest version is 4.2.4
  - Version 5 (Ninf-G5)
    - Works in non-Globus environments.
      - Easy installation and light weight communication protocols
      - Ninf-G5 is appropriate for a single system as well as for Grid
    - Latest version is 5.0.0

Architecture of Ninf-G



How to use Ninf-G

- **Build remote libraries on server machines**
  - Write IDL files
  - Compile the IDL files
  - Build and install remote executables
- **Develop a client program**
  - Programming using GridRPC API
  - Compile
- **Run**
  - Create a client configuration file
  - Generate a proxy certificate
  - Run

Sample Program

- **Parameter Survey**
  - No. of surveys: n
  - Survey function: survey(in1, in2, result)
  - Input Parameters: double in1, int in2
  - Output Value: double result[]

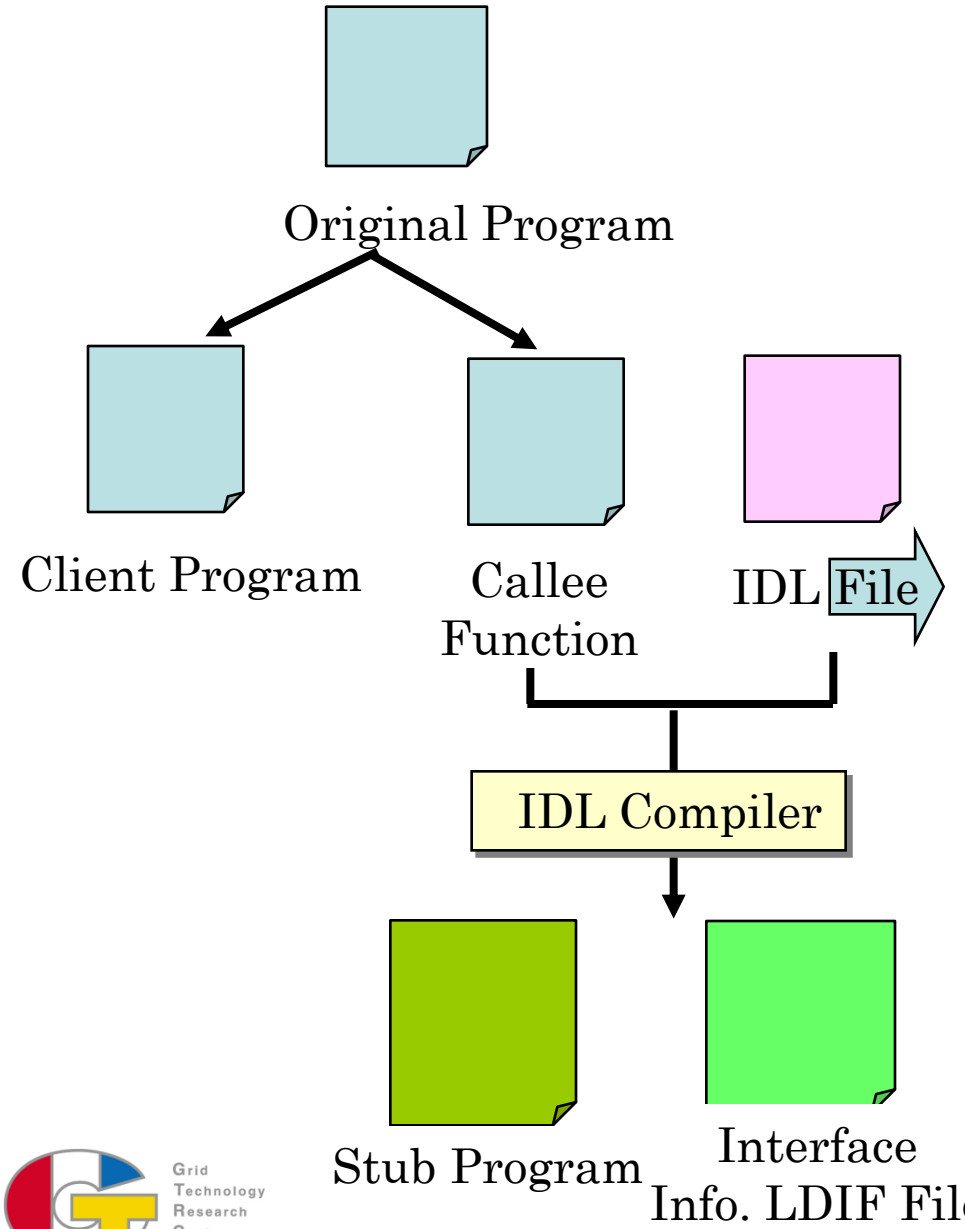
Main Program

```
Int main(int argc, char** argv)
{
int i, n, in2;
double in1, result[100][100];

Pre_processing();

For(I = 0; I < n, i++){
    survey(in1, in2, resul+100*n)
}

Post_processing();
```

Survey Function

```
survey(double in1, int in2, double* result)
{
…
  Do Survey
…

}
```

Build remote library (server-side operation)



Callee Function

```
survey
  (double in1, int in2, int size, double* result)
...
  Do Survey
...
}
```

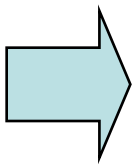
Specify size of argument

IDL File

```
Module Survey_prog;

Define survey
  (IN double in1, IN int in2, IN int size,
                  OUT double* result);

Required "survey.o"
Calls "C" survey(in1, in2, size, result);
```

Ninfy the original code (client-side)

```
Int main(int argc, char** argv)
{
int i, n, in2;
double in1, result[100][100];

Pre_processing();

For(I = 0; I < n, i++){
    survey(in1, in2, resul+100*n)
}

Post_processing();
```



```
Int main(int argc, char** argv){
int i, n, in2;
double in1, result[100][100];
grpc_function_handle_t handle [100];

Pre_processing();

grpc_initialize();
for(I = 0; I < n; i++) {
  handle[i] = grpc_function_handle_init();
}

For(I = 0; I < n, i++){
    grpc_call_async
        (handles, in1,in2,100, result+100*n)
}
grpc_wait_all();

for(I = 0; i<n; i++){
  grpc_function_handle_destruct();
}
grpc_finalize();

Post_processing();
```

Declare func. handles

Init func. handles

Async. RPC

Retrieve results

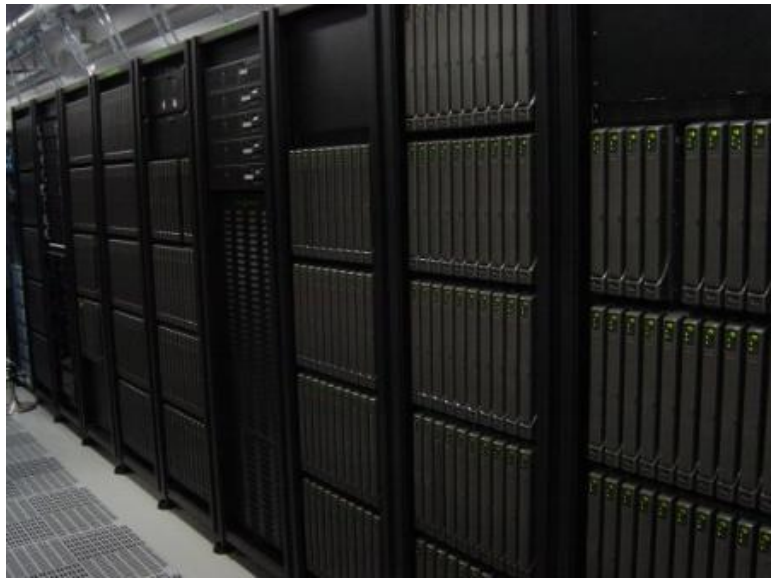
Destruct handles

Practical

Step 1: Login to machines and test Globus

Environment of this practical (1/4)

- **AIST Super Cluster F32**
  - 260 nodes dual Intel Xeon 3.06GHz
  - Giga-bit Ethernet
  - Software
    - RedHat Linux 8.0
    - Globus Toolkit 4.0.3
      - Default jobmanager
        - Pre-WS GRAM: SGE
        - WS GRAM: fork
    - Sun Grid Engine 6.0
    - Ninf-G Version 4.2.1
    - GridMPI 0.11



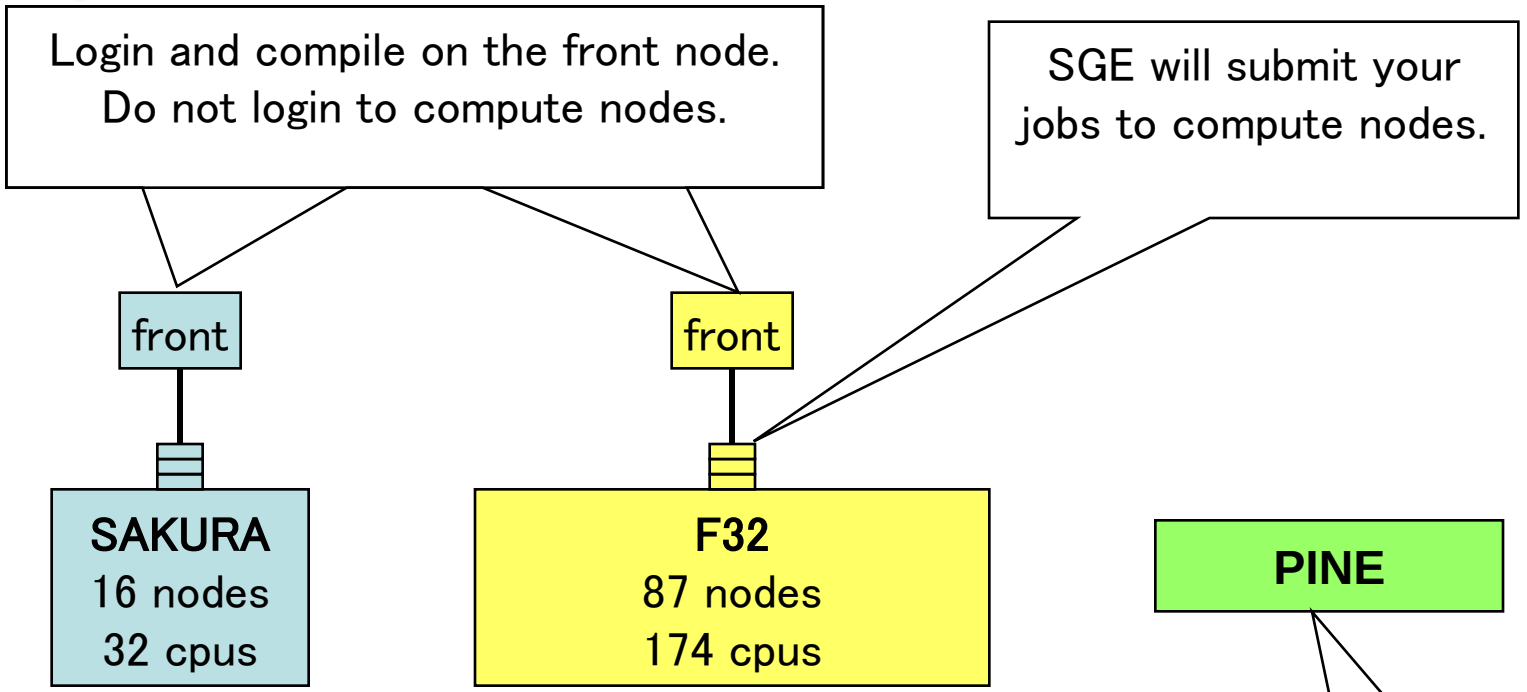
  - A part of F32 will be used in this hands-on.
    - FP3: 174CPU/87nodes (dedicated for this seminar)
  - Use as a server

Environment of this practical (2/4)

- **SAKURA Cluster**
  - 16 nodes dual AMD Opteron 1.8GHz (Sun Microsystems V20z)
  - Giga-bit Ethernet
  - Software
    - CentOS 5
    - Globus Toolkit 4.0.4
      - Default jobmanager
        - Pre-WS GRAM: SGE
        - WS GRAM: fork
    - Sun Grid Engine 6.0
    - Ninf-G Version 4.2.1
    - GridMPI 2.0rc1
  - Use as a server
- **Workstation PINE**
  - Sun Fire V240
  - Software
    - Sun Solaris 9
    - Globus Toolkit 4.0.3
    - Ninf-G Version 4.2.1
  - Use as a client machine

Environment of this practical (3/4)

- **Configuration of the three machines**



- **Front (login) node**
  - **SAKURA: sakura.hpcc.jp**
  - **F32: fsvc002.asc.hpcc.jp**
  - **PINE: pine.hpcc.jp**

Environment of this practical (4/4)

- **Your accounts are ready on PINE, F32, and SAKURA. (ng??)**
  - Please use an account from the table on the paper sheet.
  - Your account is marked by a color pen.
  - Necessary environment variables will be set when you login to the system.
  - These accounts will be deactivated after this seminar.
- **Default jobmanager of the Globus Toolkit**
  - jobmanager-sge for Pre-WS GRAM
    - Your submitted job via Globus GRAM will be invoked on compute nodes through SGE.
    - Your job may be queued if the system is busy
  - jobmanager-fork for WS GRAM
    - Your submitted job via Globus GRAM will be invoked on the front node through fork/exec.
    - You must explicitly specify jobmanager-sge to use SGE for process invocation.
- **Your Globus certificate is available on PINE.**
  - Globus password is same with your login password.
  - Your certificate is valid for 7 days.

Step1-1/3：Login to PINE, F32, and SAKURA

- **Login to PINE (pine.hpcc.jp) using ssh (You don't need to logout until the end of this session)**

  % ssh ng00@pine.hpcc.jp

  ↑

  - Please replace by your account (marked by color)
  - Password is on the paper sheet.

Step1-2/3：Test Globus on PINE

- **Check your ~/.globus directory**

  % cd .globus

  % ls –al *.pem

  - r w – r - - r - - usercert.pem ... Your public key certificate

  - r - - - - - - - - - userkey.pem ... Your private key

  % grid-cert-info –file usercert.pem

- **Check environment variable**
  % env | grep GLOBUS

- **Create your proxy certificate**
  % grid-proxy-init
  - **Enter Globus password (same with the login password)**

- **Check your proxy**
  % grid-proxy-info

Step1-3/3: Test Globus on PINE and login to F32 and SAKURA

- **Test Globus authentication**
  % globusrun -a -r fsvc003.asc.hpcc.jp
  % globusrun -a -r sakura.hpcc.jp

- **Run Globus**
  % globus-job-run fsvc003.asc.hpcc.jp /bin/hostname
  % globusrun-ws -F fsvc003.asc.hpcc.jp –Ft SGE –s -submit -c /bin/hostname
  % globus-job-run sakura.hpcc.jp /bin/hostname
  % globusrun-ws -F sakura.hpcc.jp -Ft SGE -s -submit -c /bin/hostname

- **Login to F32 (fsvc003.asc.hpcc.jp) and SAKURA (sakura.hpcc.jp) using ssh (You don't need to logout until the end of this session)**
  % ssh ng00@fsvc003.asc.hpcc.jp
  % ssh ng00@sakura.hpcc.jp

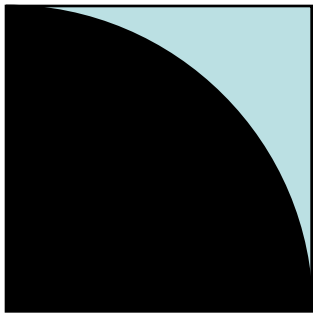
  - Please replace by your account (marked by color)
  - Password is on the paper sheet.

Step 2: Target Application

Calculate PI using Mote-Carlo Simulation

- **A dot is randomly placed within a square whose side length is 1.**
  - S: The number of steps
  - p: The number of dots whose distance from the origin is shorter than 1
  - PI = (p / S) x 4

```
long pi_trial(int seed, long times)
{
    long l, counter = 0;

    srandom(seed);

    for (l = 0; l < times; l++) {
        double x = (double)random() / RANDOM_MAX;
        double y = (double)random() / RANDOM_MAX;
        if (x * x + y * y < 1.0) {
            counter++; } }
    return counter;
}
```



Grid Technology Research Center AIST

Step2-1/1：Run sequential version of PI

- **Run on PINE (pine.hpcc.jp)**
- **change samples/serial directory**

  pine$ cd samples/serial
- **View the source**

  pine$ more pi_serial.c

  pine$ more pi_trial.c
- **Compile & Run**

  pine$ make

  pine$ ./pi_serial 1000000
  - Argument is the number of points

Step 3: 1 client – 1 server

– Ninf-G Client running on PINE makes RPC on F32 –

Makes RPC via Pre-WS GRAM

Notes

We will not use MDS as an Information Service module.

Retrieve information from local file.

Step3-1/8: Prepare server

- **Execute on F32 (fsvc003.asc.hpcc.jp)**

- **change to samples/1server directory**

  fsvc003$ cd samples/1server

- **view sources**

  fsvc003$ more pi.idl

  fsvc003$ more pi_trial.c

  ▶…

Step3-2/8：Server-side program

pi.idl

```
Module pi;

Define pi_trial (
  IN int seed,
  IN long times,
  OUT long * count)
"monte carlo pi computation"
Required "pi_trial.o"
{
  long counter;
  counter = pi_trial(seed, times);
  *count = counter;
}
```

pi_trial.c

```
long pi_trial (int seed, long times) {
  long l, counter = 0;

  srandom(seed);
  for (l = 0; l < times; l++) {
    double x =
      (double)random() / RAND_MAX;
    double y =
      (double)random() / RAND_MAX;

    if (x * x + y * y < 1.0)
      counter++;
  }
  return counter;
}
```

Actual codes include error checking codes and debug prints

Step3-3/8 : Generate Stub sources

Compile IDL file

- Use ng_gen to compile IDL

  fsvc003% ng_gen pi.idl

- Generated files
  - _stub_pi_trial.c
    - Includes stub functions generated by the IDL compiler
    - A wrapper function of the actual callee function
    - Handle arguments/results transfers between a Ninf-G Client
  - pi.mak
    - Makefile for generating Ninf-G Executable and LDIF file.

Step3-4/8 : Generate Ninf-G Executable

Generate Ninf-G Executable

- Use makefile generated by ng_gen command

  fsvc003% make -f pi.mak

- Generated files
  - _stub_pi_trial
    - Ninf-G Executable
  - _stub_pi_trial.inf
    - Information file containing arguments information, etc.
  - pi::pi_trial.ldif
    - LDIF file which is used to register function information to MDS.
  - pi.fsvc003.asc.hpcc.jp.ngdef
    - A file containing function information which is readable as a local file

Step3-5/8：Prepare client

- **Execute on PINE（pine.hpcc.jp）**

- **Change to samples/1server directory**

  pine$ cd samples/1server

- **View source**

  pine$ more pi_client_1server.c

Step3-6/8 : Client Program

```
#include "grpc.h"

char *func_name = "pi/pi_trial";
char *config_file = "client.conf";
int port = 0;

int
main(int argc, char *argv[])
{
    grpc_function_handle_t handle;
    grpc_error_t result = GRPC_NO_ERROR;
    char *host = NULL;
    long times, answer;

    if (argc != 3){
        fprintf(stderr,
          "USAGE: %s TIMES HOSTNAME¥n", a
        exit(2);
    }
    times = atol(argv[1]);
    host = argv[2];

    /* Initialize */
    result = grpc_initialize(config_file);
```

```
    /* Initialize Function handle */
    result = grpc_function_handle_init(
       &handle, host, func_name);


    /* Synchronous call */
    result = grpc_call(
       &handle, 0, times, &answer);

    /* Destruct Function handles */
    result = grpc_function_handle_destruct
       (&handle);

    /* Compute and display pi. */
    printf("PI = %f¥n",
        4.0 * ((double)answer / times));

    /* Finalize */
    result = grpc_finalize();

    return 0;
}
```

Step3-7/8：Generate Ninf-G Client

- **Compile Ninf-G Client program**

  pine$ make pi_client_1server

Step3-8/8 : Run

- **Copy Local LDIF file from F32**

  pine$ scp fsvc003.asc.hpcc.jp:samples/1server/pi.fsvc003.asc.hpcc.jp.ngdef .

- **View Client configuration file**

  % vi client.conf

- **Run**
  - Since your proxy has been created, you don't need to run grid-proxy-init again.
  - Run

    pine$ ./pi_client_1server 1000000 fsvc003.asc.hpcc.jp

Step 4: 1 client – 2 servers

– Ninf-G Client running on PINE makes RPC on F32 and SAKURA –

Use asynchronous RPC

Use Pre-WS GRAM for F32 and WS-GRAM for SAKURA

Step4-1/4

- **Prepare server**
  - Execute on F32
- **Change to samples/2servers directory**

  fsvc003$ cd samples/2servers
- **Generate remote library**
  - Compile IDL

    fsvc003$ ng_gen pi.idl
  - Generate Ninf-G Executable

    fsvc003$ make -f pi.mak

Step4-2/4

- **Prepare server**
  - Execute on SAKURA
- **Change to samples/2servers directory**

  sakura$ cd samples/2servers
- **Generate remote library**
  - Compile IDL

    sakura$ ng_gen pi.idl
  - Generate Ninf-G Executable

    sakura$ make -f pi.mak

Step4-3/4

- **Generate Ninf-G Client**
  - Execute on PINE
- **Change to samples/2servers directory**

  pine$ cd samples/2servers
- **Compile client program**

  pine$ make pi_client_2servers
- **Copy Local LDIF files.**

  pine$ scp fsvc003.asc.hpcc.jp:samples/2servers/pi.fsvc003.asc.hpcc.jp.ngdef .

  pine$ scp sakura.hpcc.jp:samples/2servers/pi.sakura.hpcc.jp.ngdef .

Step4-4/4

- **View client configuration file**

  pine$ more client.conf

- **Check <SERVER> section for sakura.hpcc.jp.**

```
<SERVER>
    hostname sakura.hpcc.jp
    invoke_server   GT4py
    jobmanager jobmanager-sge
</SERVER>
```

Use GT4py for SAKURA to make RPC via WS GRAM

Specify jobmanager –sge since the default jobmanager of WS GRAM is fork.

- **Run**

  pine$ ./pi_client_2servers 1000000 fsvc003.asc.hpcc.jp sakura.hpcc.jp

What are the difference?

- **Make RPC to two servers**
- **One is through Pre-WS GRAM and the other one is through WS GRAM**
- **Used asynchronous RPC**
  - grpc_call_async()
- **Wait results (sync)**
  - grpc_wait_all()
- **Important issue**
  - Attributes (e.g. available GRAM and jobmanager) for each host is described in the Client Configuration file. You don't need to change application.

Step 5: Use array of function handles

– How to manage many function handles? –

Use array functions to create multiple function handles by a single function

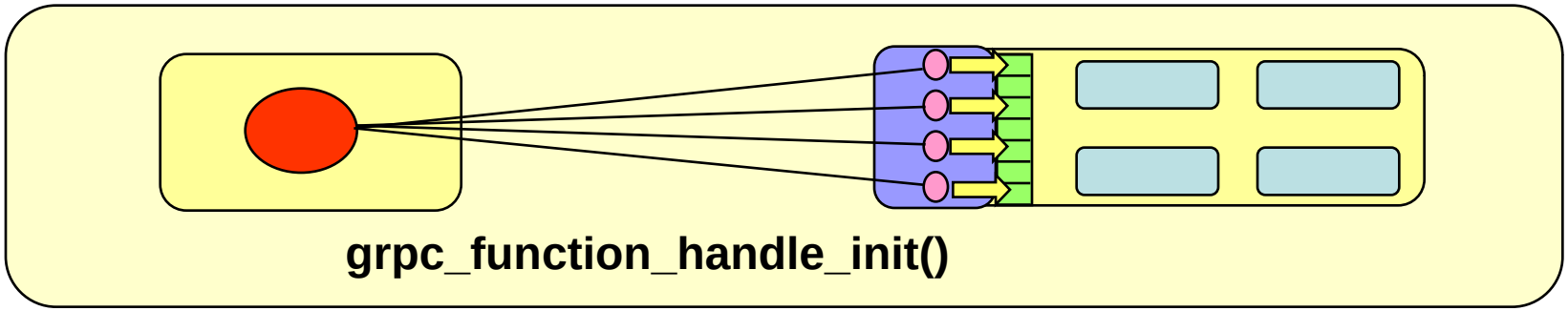
Use asynchronous RPCs for parallel processing

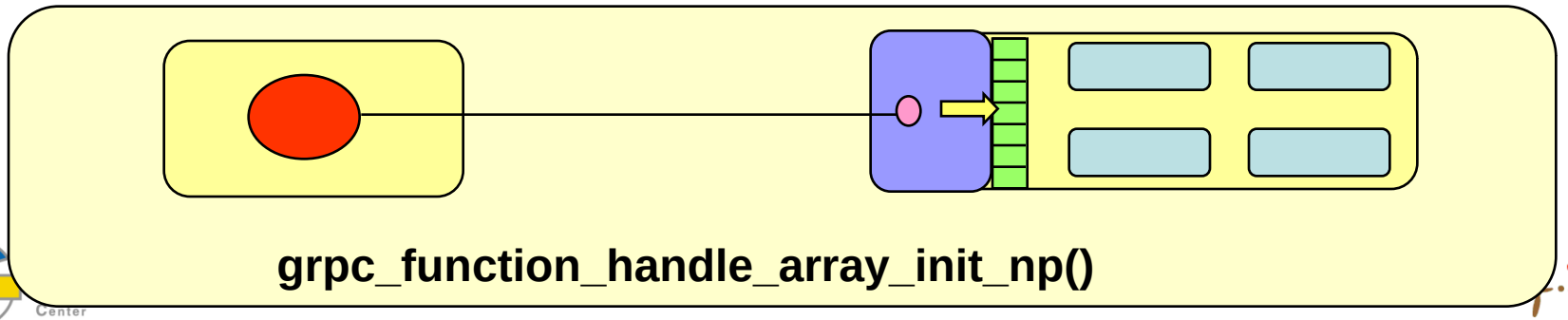
A function to create multiple function handles at a time

- **grpc_function_handle_init/default()**
  - Invoke Ninf-G Executable
  - If GRAM is used,
    - AuthN + AuthZ by GSI
    - Invoke a jobmanager process
    - Invoke an Executable via batch scheduler
  - It is practically impossible to invoke several tens of Ninf-G Executables on a single cluster
- **Ninf-G provides a function which create an array of function handles**
  - Create multiple function handles by a single GRAM call.

Array function

- **Reduce overhead to create multiple function handles**
  - grpc_function_handle_init
    - Invoke N jobmanager processes
    - N AuthN+Authz
    - N job submissions to batch system
  - grpc_function_handle_array_init_np()
    - Invoke 1 jobmanager process
    - 1 AuthN+AuthZ
    - 1 job submission to batch system





APIs for array functions

- **grpc_error_t grpc_function_handle_array_default_np (**
  grpc_function_handle_t *handle,
  size_t nhandles,
  char *func_name)
  - Creates multiple function handles via a single GRAM call
- **grpc_error_t grpc_function_handle_array_init_np (**
  grpc_function_handle_t *handle,
  size_t nhandles,
  char *host_port_str,
  char *func_name)
  - Specifies the server explicitly by the second argument.
- **grpc_error_t grpc_function_handle_array_destruct_np (**
  grpc_function_handle_t *handle,
  size_t nhandles)
  - Specifies the server explicitly by the second argument.

Step5-1/3

- **Prepare server**
  - Execute on F32 or SAKURA. F32 is recommended.
    - If you use SAKURA, you need to edit client.conf file.
- **Change to samples/array directory**

  fsvc003$ cd samples/array
- **Generate remote library**
  - Compile IDL

    fsvc003$ ng_gen pi.idl
  - Generate Ninf-G Executable

    fsvc003$ make -f pi.mak

Step5-2/3

- **Generate Ninf-G Client**
  - Execute on PINE

- **Change to samples/array directory**

  pine$ cd samples/array

- **Compile Ninf-G Client**

  pine$ make pi_client_array

- **Copy local LDIF file from F32**

  pine$ scp fsvc003.asc.hpcc.jp:samples/array/pi.fsvc003.asc.hpcc.jp.ngdef .

Step5-3/3

- **View Client Configuration File**

  pine$ more client.conf

- **Run**

  pine$ ./pi_client_array 1000000 fsvc003.asc.hpcc.jp 16

Step 6: Ninf-G Remote Object

- State-full Executable -

Define remote objects at the server-side

Client creates remote objects and calls methods

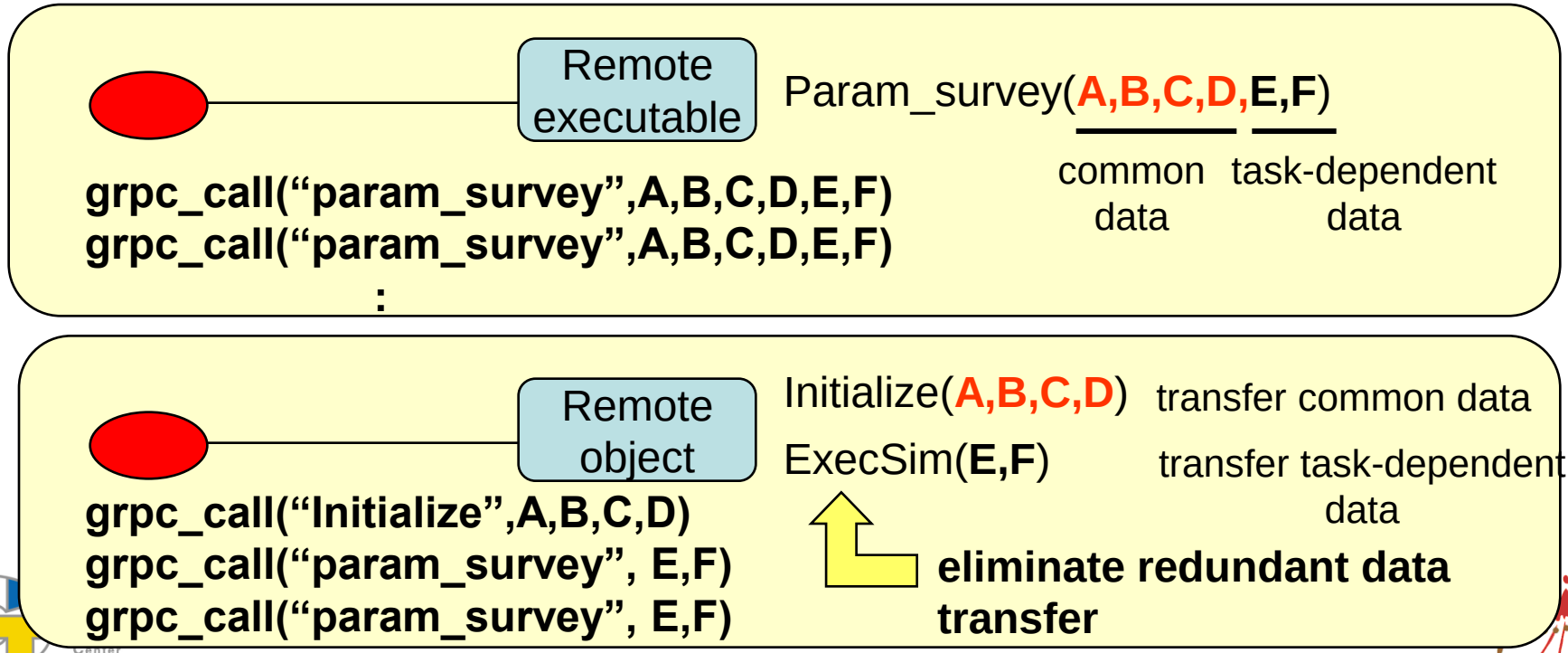
Asynchronous RPC for remote methods

What is a remote object?

- **A Ninf-G Function is state-less**
  - State of the last RPC is not kept by the Ninf-G Function
- **Some typical applications make RPC using the same large data with different parameters.**
- **Since Ninf-G Function is stateless, such large-data must be transferred for every RPC.**
- **Ninf-G Remote Object is state-full Executable.**
- **Introduce a concept of object oriented programming.**
  - Define class
    - methods and instance variables
  - Instantiate objects (object handles)
- **Provide IDL to define remote objects**
  - DefClass

Ninf-G Remote Objects

- **Reduce redundant data transfer**
  - Ninf-G Function
    - Has a single interface
    - Need to transfer data for every RPC
  - Ninf-G Remote Object
    - Has multiple interfaces
    - Able to send persistent data by a initialization method.
    - Use the other method for each task



Object handles

- **grpc_error_t grpc_object_handle_default_np (**
  grpc_object_handle_t_np *handle,
  char *class_name)
  - Creates an object handle to the default server
- **grpc_error_t grpc_object_handle_init_np (**
  grpc_function_object_t_np *handle,
  char *host_port_str,
  char *class_name)
  - Specifies the server explicitly by the second argument.
- **grpc_error_t grpc_function_object_destruct_np (**
  grpc_object_handle_t_np *handle)
  - Frees memory allocated to the function handle.

Object handles (cont'd)

- **grpc_error_t grpc_object_handle_array_default (**
  grpc_objct_handle_t_np *handle,
  size_t nhandles,
  char *class_name)
  - Creates multiple object handles via a single GRAM call.
- **grpc_error_t grpc_object_handle_array_init_np (**
  grpc_object_handle_t_np *handle,
  size_t nhandles,
  char *host_port_str,
  char *class_name)
  - Specifies the server explicitly by the second argument.
- **grpc_error_t grpc_object_handle_array_destruct_np (**
  grpc_object_handle_t_np *handle,
  size_t nhandles)
  - Frees memory allocated to the function handles.

Step6-1/3

- **Prepare server**
  - Execute on F32 or SAKURA (F32 is recommended)
    - If you use SAKURA, you need to edit client.conf file.
- **Change to samples/object directory**

  fsvc003$ cd samples/object
- **View sources**

  fsvc003$ more pi_object.idl
- **Generate Ninf-G Executable**
  - Compile IDL

    fsvc003$ ng_gen pi_object.idl
  - Generate Ninf-G Executable

    fsvc003$ make -f pi_object.mak

Step6-2/3

- **Generate Ninf-G Client**
  - Execute on PINE
- **Change to samples/object directory**

  pine$ cd samples/object
- **View source**

  pine$ more pi_client_object.c
- **Compile Ninf-G Client**

  pine$ make pi_client_object
- **Copy Local LDIF file from server**

pine$ scp fsvc003.asc.hpcc.jp:samples/object/pi_object.fsvc003.asc.hpcc.jp.ngdef .

Step7-3/3

- **View Client configuration file**

  pine$ more client.conf

- **Run**

  pine$ ./pi_client_object 1000000 fsvc003.asc.hpcc.jp

Step 8: Call MPI program as a server function

Execute a MPI program at the server-side

How do you manage hundreds to thousands of CPUs?

- **Simple client-server model has problems on scalability**
  - Client can be a bottleneck
- **Two possible options**
  - Use hierarchical RPCs.
  - Call MPI program at the server side
- **Call MPI program as a Ninf-G function**
  - Take advantages of both GridRPC and MPI
    - fine-grained parallelism by MPI
    - flexibility and robustness by GridRPC
- **Ninf-G IDL allows to specify "Backend MPI" to invoke MPI at the server.**

Step7-1/3

- **Prepare server**
  - Execute on F32 or SAKURA (F32 is recommended)
    - If you use SAKURA, you need to edit client.conf file.
- **Change to samples/mpi directory**

  fsvc003$ cd samples/mpi
- **View sources**

  fsvc003$ more pi_mpi.idl
- **Generate remote library**
  - Compile IDL

    fsvc003$ ng_gen pi_mpi.idl
  - Generate Ninf-G Executable

    fsvc003$ make -f pi_mpi.mak

Step7-2/3

- **Generate Ninf-G Client**
  - Execute on PINE.

- **Change to samples/mpi directory**

  pine$ cd samples/mpi

- **View sources**

  pine$ pi_client_mpi.c

- **Compile Ninf-G Client**

  pine$ make pi_client_mpi

- **Copy local LDIF file from server**

pine$ scp fsvc003.asc.hpcc.jp:samples/mpi/pi_mpi.fsvc003.asc.hpcc.jp.ngdef .

Step7-3/3

- **View client configuration file**

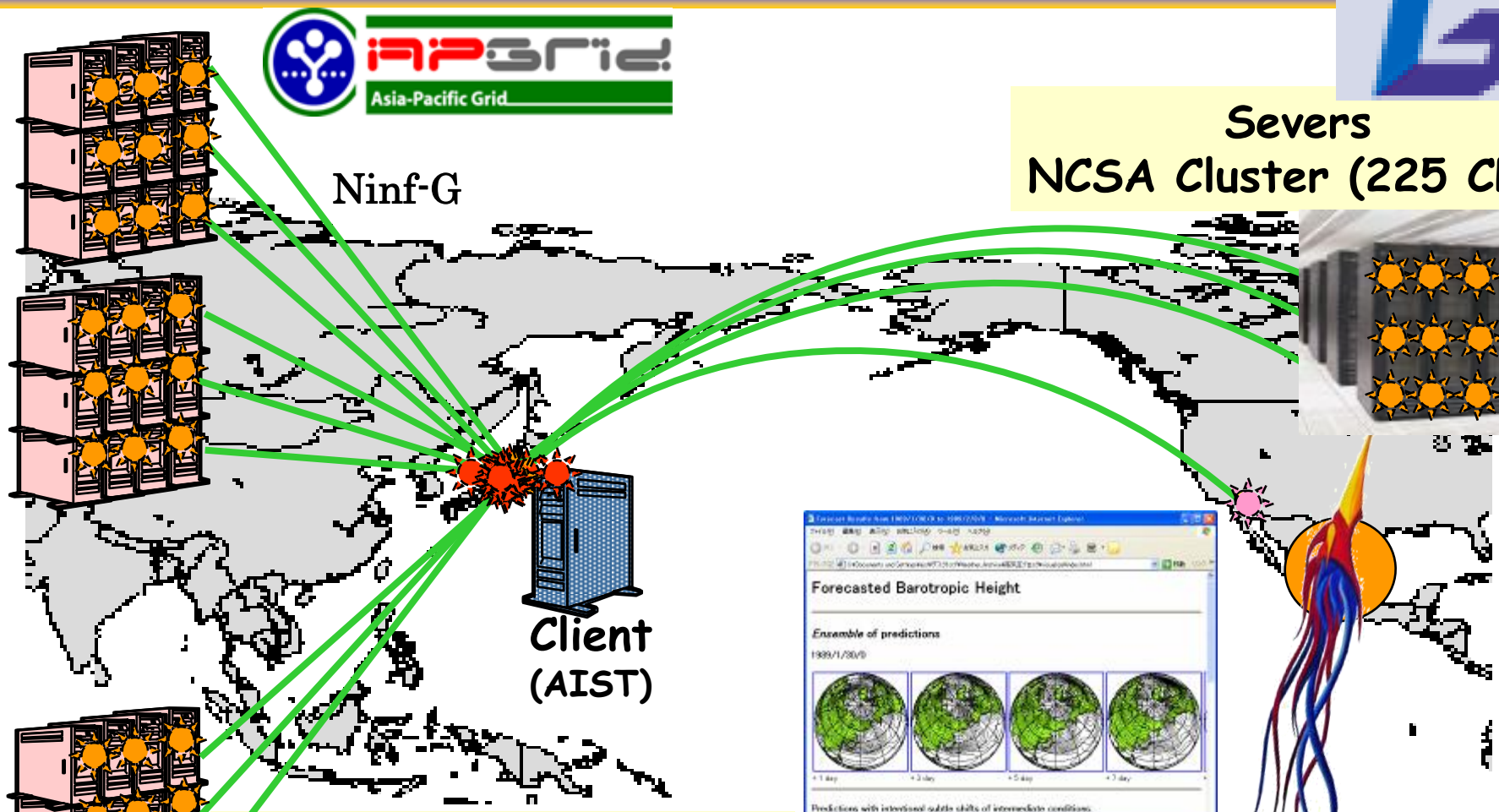
  pine$ more client.conf

- **Run**

  pine$ ./pi_client_mpi 1000000 fsvc003.asc.hpcc.jp

Advanced examples

Climate Simulation on ApGrid/TeraGrid at SC2003

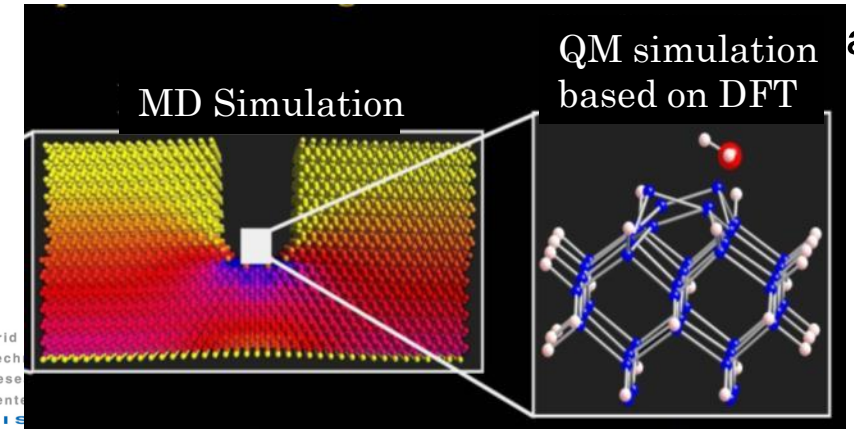


Grid Technology Research Center AIST

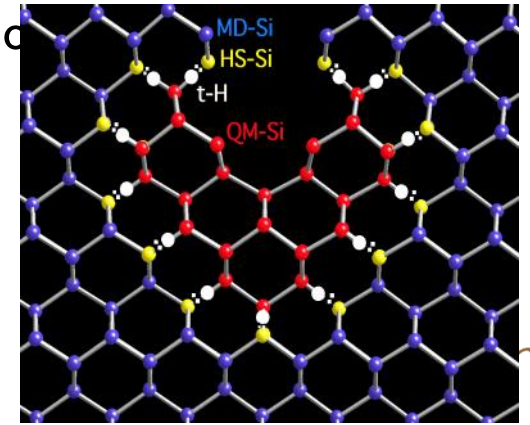
Ninf

Hybrid QM/CL Simulation (1)

- **Enabling large scale simulation with quantum accuracy**
  - Combining classical MD Simulation with quantum simulation
    - CL simulation
      - Simulating the behavior of atoms in the entire region
      - Based on the classical MD using an empirical inter-atomic potential
    - QM simulation
      - Modifying energy calculated by MD simulation only in the interesting regions

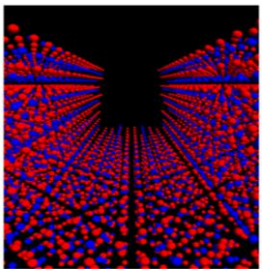


al theo

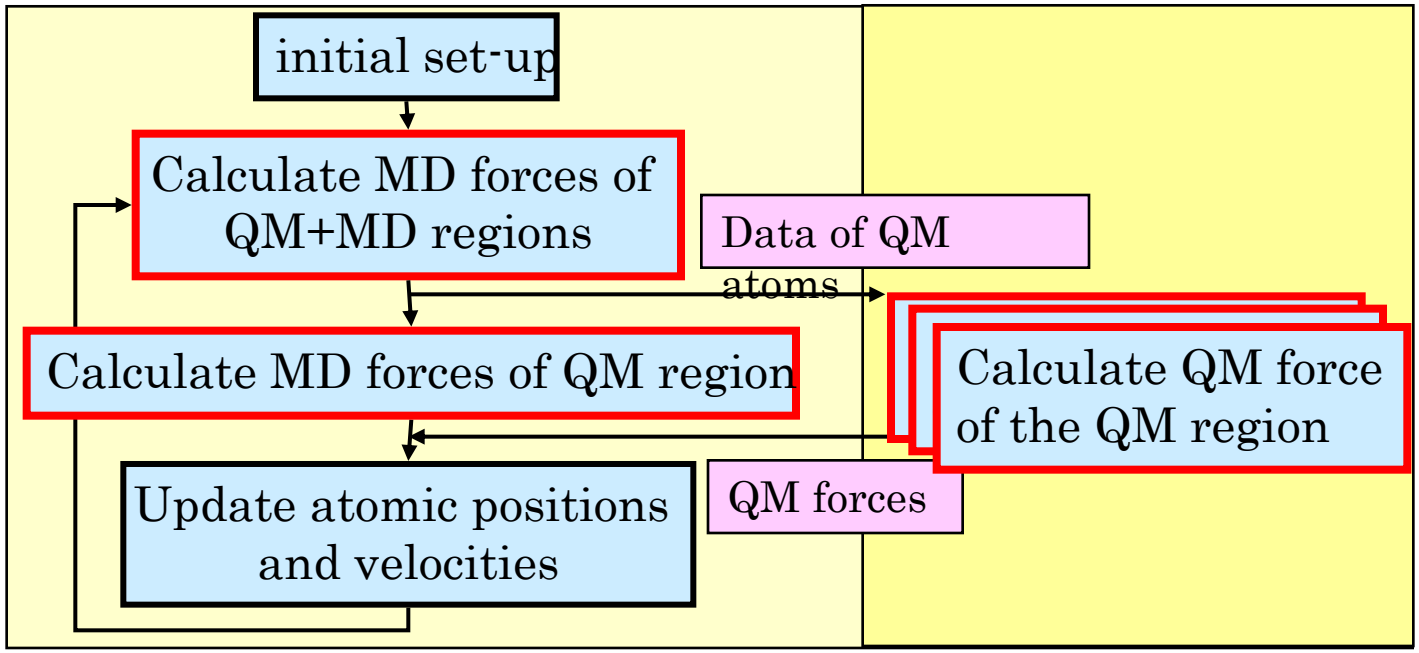


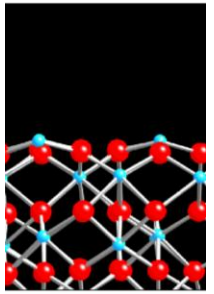
Hybrid QM/CL Simulation (2)

- **simulation algorithm**



MD part





QM part

- **Each QM computation is**
  - independent with each other
  - compute intensive
  - usually implemented as a MPI program

Approach to "gridify" applications

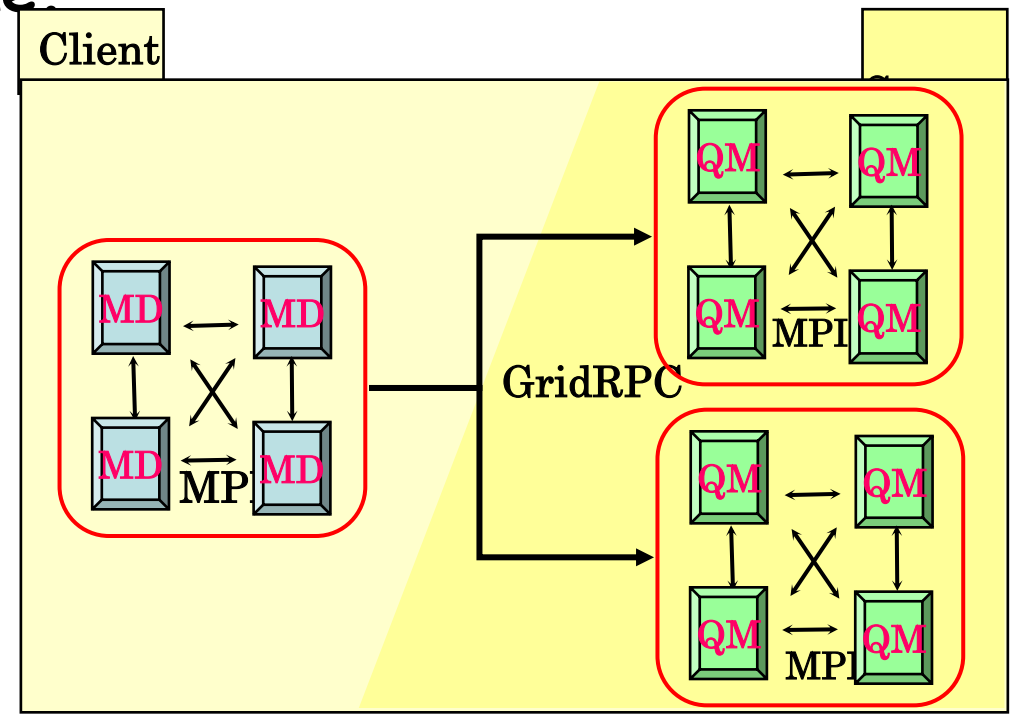
The new programming approach, combining GridRPC with MPI, takes advantages of both programming models complementarily to run large-scale applications on the Grid for a long time.

Grid RPC enhances the flexibility and robustness by;

- dynamic allocation of server programs, and
- detection of network/cluster trouble.

MPI enhances the efficiency by;

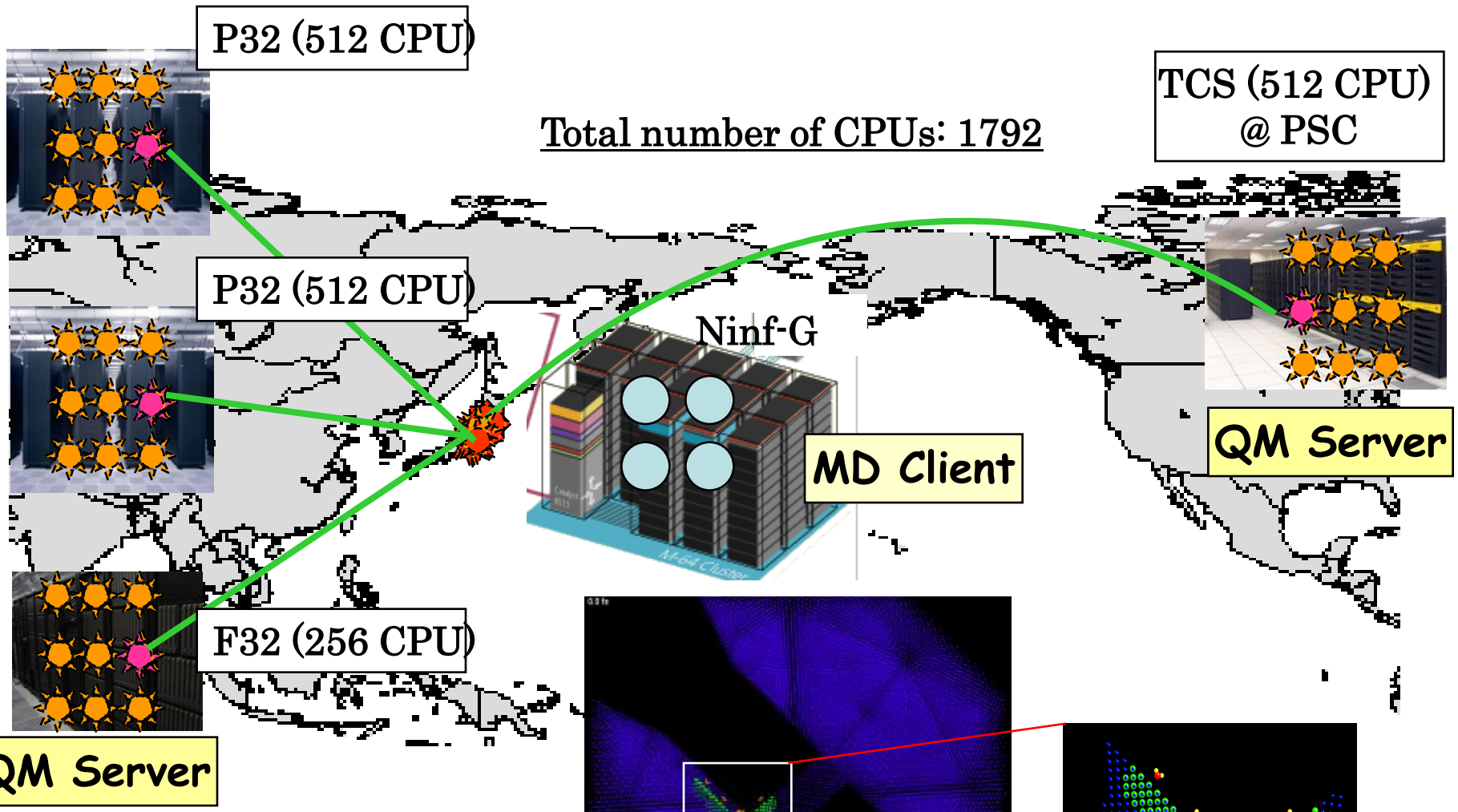
- highly parallel computing on a cluster for both client and server programs.



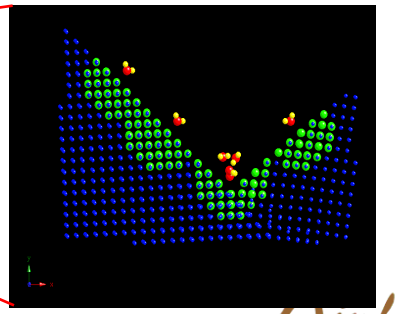
Does the implementation give solutions for the requirements?

- **Flexibility**
  - GridRPC enables dynamic join/leave of QM servers.
  - GridRPC enables dynamic expansion of a QM server.
- **Robustness**
  - GridRPC detects errors and application can implement a recovery code by itself.
- **Efficiency**
  - GridRPC can easily handle multiple clusters.
  - Local MPI provides high performance on a cluster by fine grain parallelism.

QM/MD simulation over the Pacific at SC2004



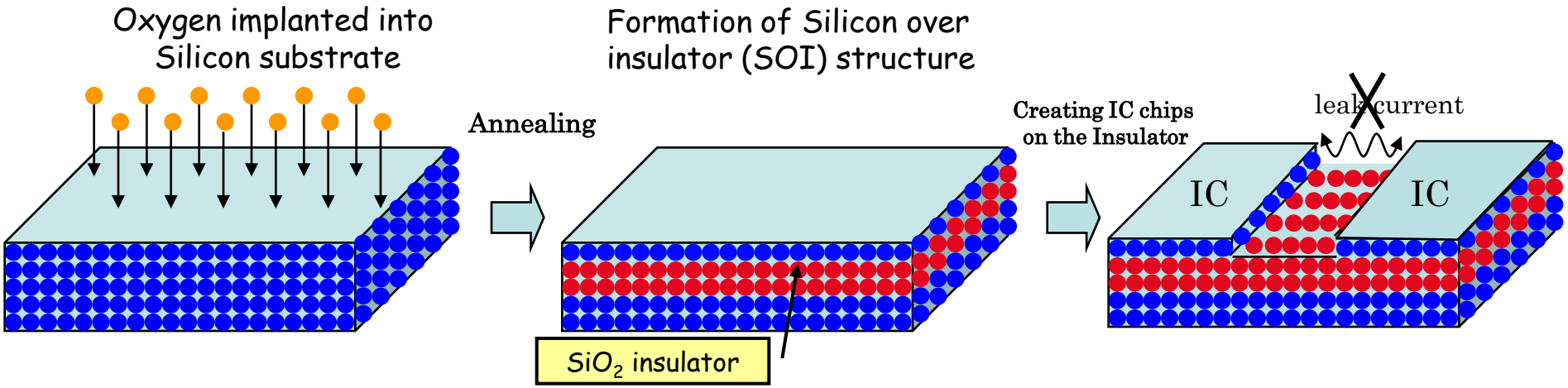
corrosion of Silicon under stress

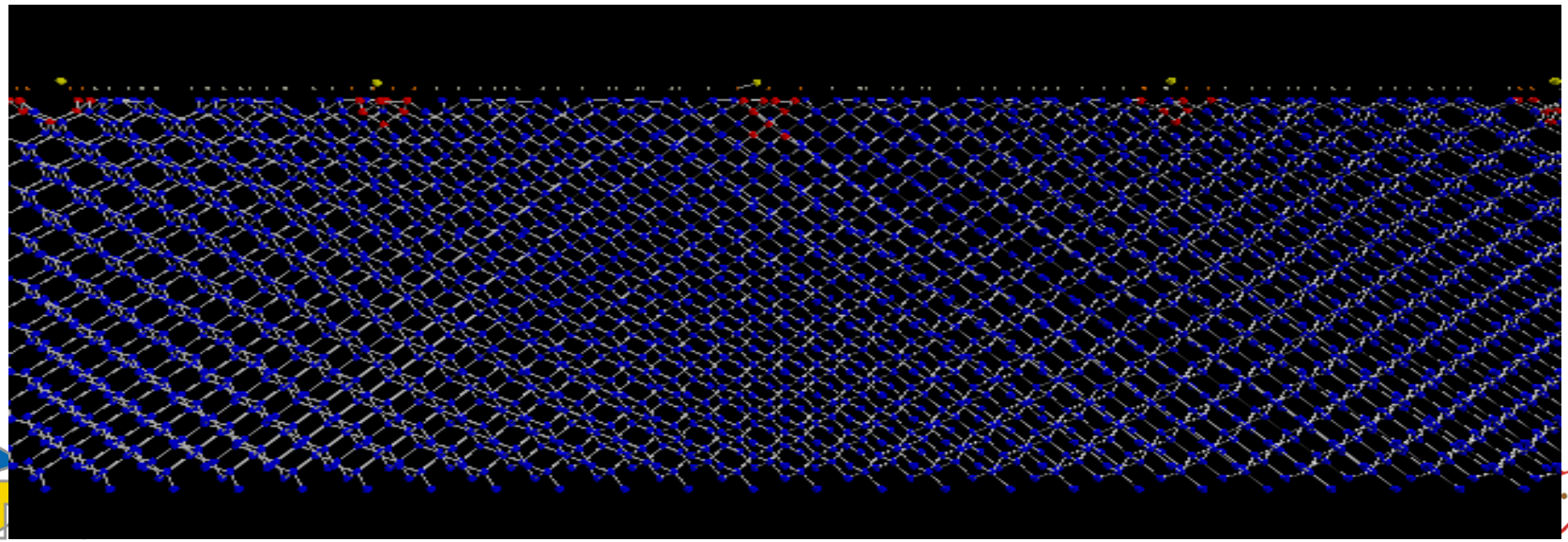


Close-up view

Grid-enabled SIMOX Simulation on Japan-US Grid Testbed at SC2005

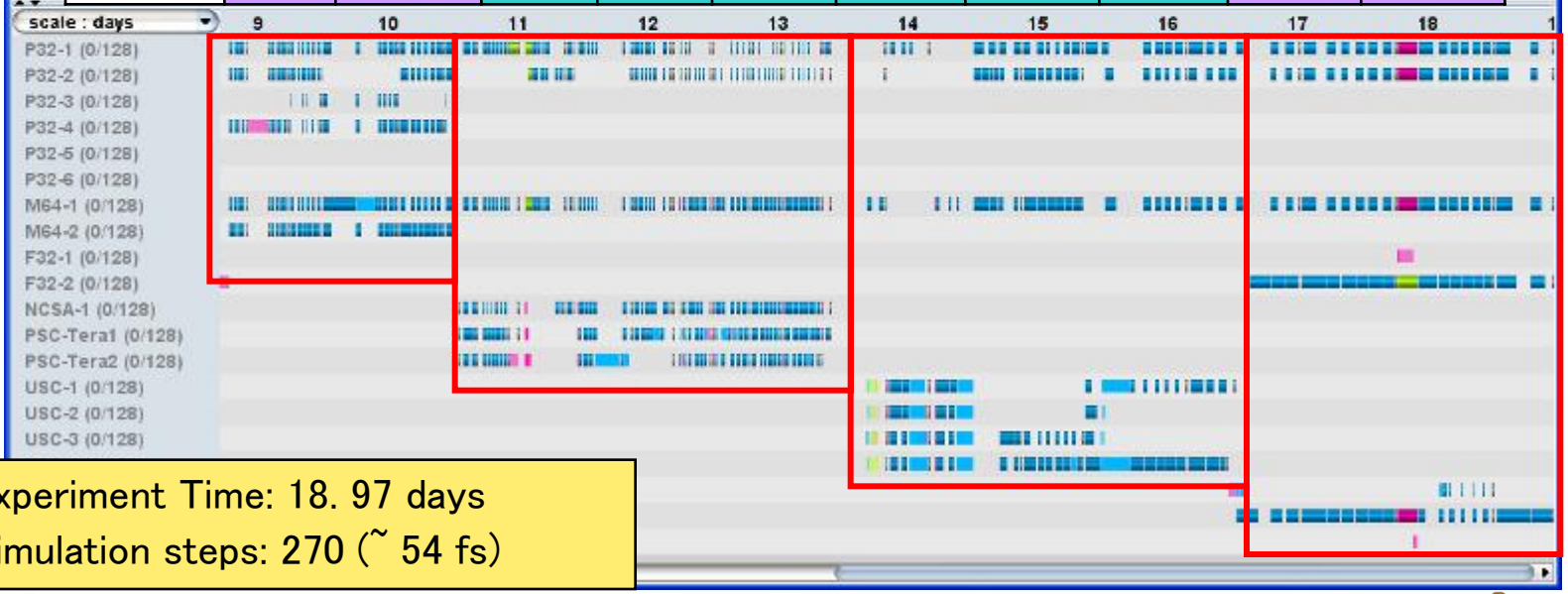
- A technique to fabricate a micro structure consisting of Si surface on the thin SiO_2 insulator
- Allows to create higher speed with lower power consumption device





Results of the experiment

QM1	P32	P32	P32	P32	P32	USC	USC	USC	ISTBS	ISTBS
QM2	P32	P32	NCSA	NCSA	NCSA	USC	USC	USC	Presto	Presto
QM3	M64	M64	M64	M64	M64	M64	M64	M64	M64	M64
QM4	P32	P32	TCS	TCS	TCS	USC	USC	USC	P32	P32
QM5	P32	P32	TCS	TCS	TCS	USC	USC	USC	P32	P32
Reserve	F32	F32	P32	P32	P32	P32	P32	P32	F32	F32



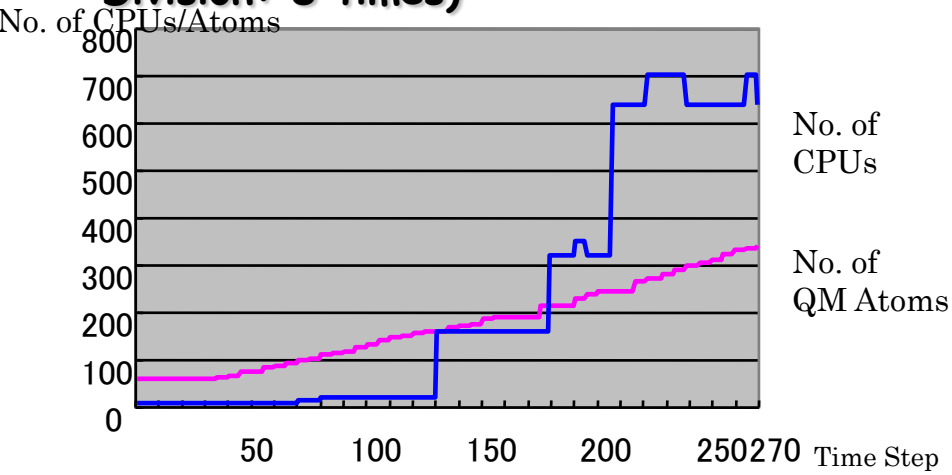
- Experiment Time: 18. 97 days
- Simulation steps: 270 (~ 54 fs)

Technology Research Center AIST

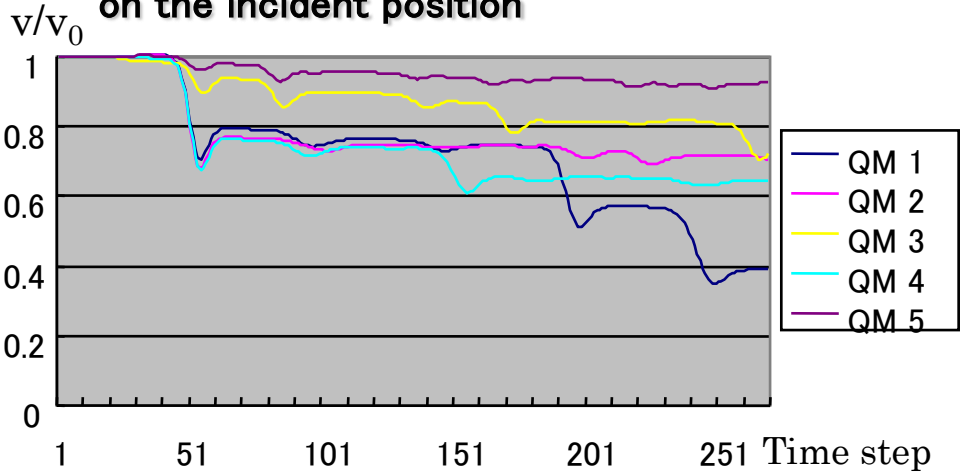
Phase 1 Phase 2 Phase 3 Phase 4

Results of the experiment (cont'd)

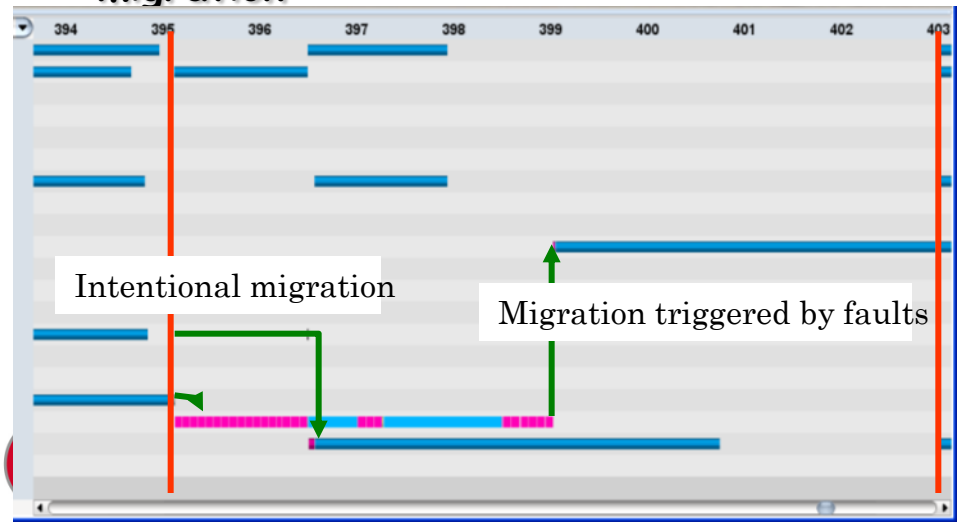
- **Expanding/Dividing QM regions at every 5 time steps(Expansion: 47 times, Division: 8 times)**

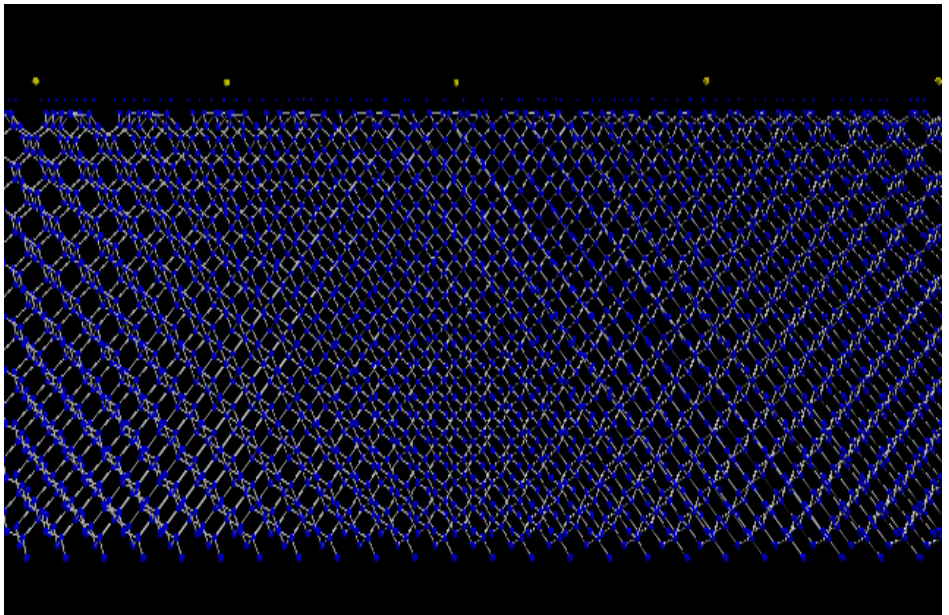


- **Behavior of oxygen atoms strongly depends on the incident position**



- **Succeeded in long-run by intentional/unintentional resource migration**





Summary

Remarkable Features of Ninf-G

- **Enabling RPC via any middleware**
  - Pre-WS GRAM, WS GRAM, UNICORE
  - Condor, SSH
  - Any others
- **Client configuration file that allows detailed description of execution environments**
  - Server, Client, Function attributes, etc.
- **Capability for fault detection**
  - Explicit faults
    - Server process died, network disconnection, etc.
  - Implicit faults
    - Timeout of Invocation, Execution, Heatbeat
- **High quality as the software**

Latest version: Ninf-G 5.0.0

- **Ninf-G Version 5.0.0 is available for download!**
  - http://ninf.apgrid.org/
- **What are differences with Ninf-G4?**
  - Lower prerequisites for installation
    - Ninf-G4 needs Globus Library since it uses Globus IO for client/server communications.
    - Ninf-G5 can be installed without Globus. i.e.,
- **Ninf-G5 can be installed according to the underlying software environments**
  - Three major components (remote process invocation, information retrieval, and client/server communication) can be pluggable.
  - e.g. without Globus, without TCP
  - Work efficiently from a single supercomputer to Grid
- **Other new features will be supported**
  - Connection less (client <-> server)
  - Client-side check pointing