How to build a Grid
How to use a Grid
- the second half
GridRPC and Ninf-G

Yoshio Tanaka, Hidemoto Nakada
Grid Technology Research Center, AIST

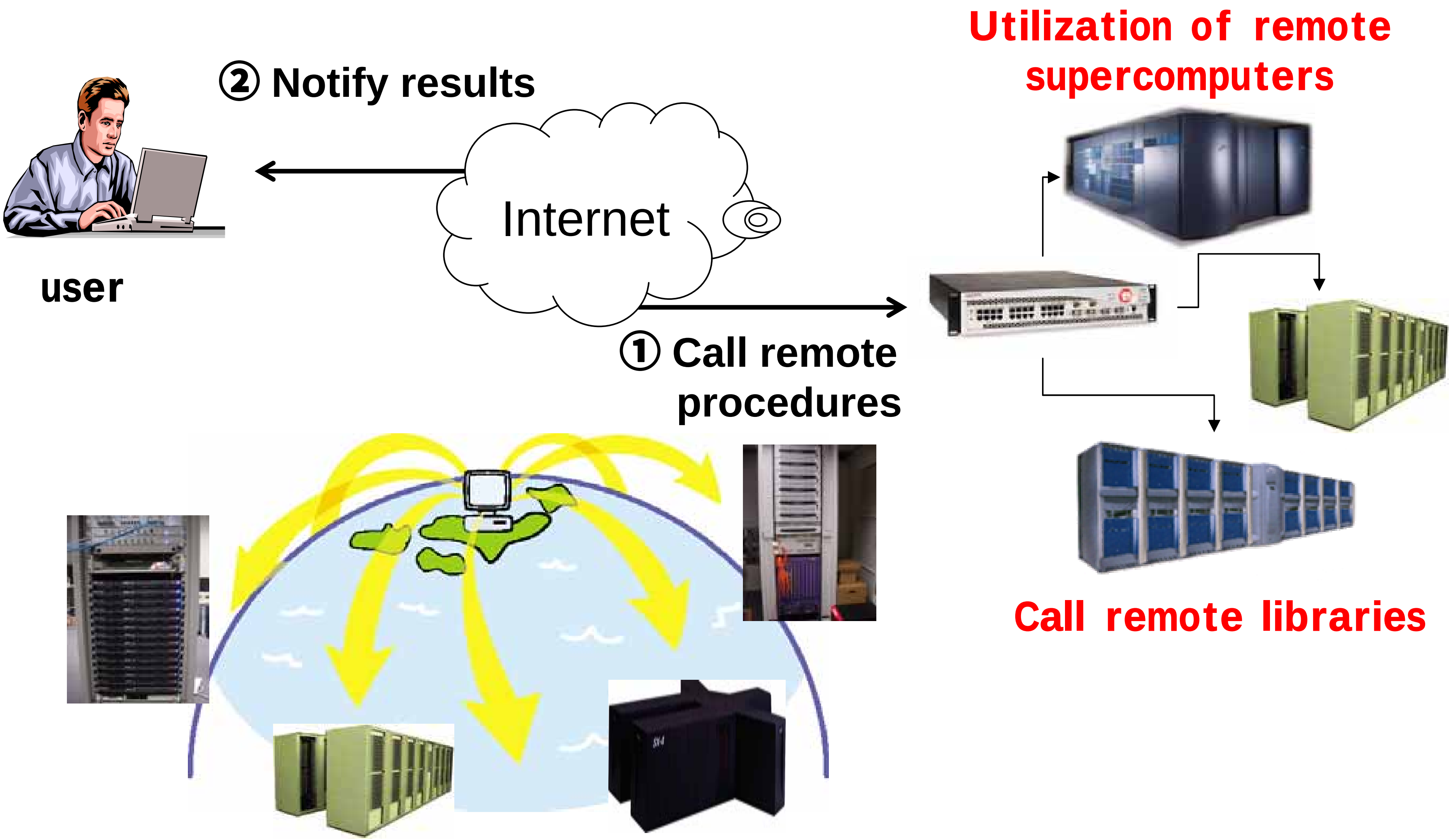
Outline

- **Overview and architecture of the GT2**
  (by Yoshio Tanaka, 60min)
  - Common software infrastructure of both building and using Grids
  - Introduce the GT2 from a point of view of users
- **How to build a Grid**
  (by Yoshio Tanaka, 60min)
  - general issues
  - use the ApGrid Testbed as an example
- **How to use a Grid**
  (by Hidemoto Nakada, 90min)
  - Programming on the Grid using GridRPC
  - use Ninf-G as a sample GridRPC system

Outline of the second half

- **GridRPC**
  - Programming model
  - Detailed API
  - Program Example
- **Ninf-G**
    - An implementation of the GridRPC
  - Implementation
  - How to use Ninf-G
    - Installation
    - Usage example

GridRPC: RPC based Programming model



Large scale computing utilizing multiple supercomputers on the Grid

The GridRPC API

- **Provide standardized, portable, and simple programming interface for Remote Procedure Call**
- **Attempt to unify client access to existing grid computing systems (such as NetSolve and Ninf-G)**
- **Working towards standardization through GGF WG**
  - Initially standardize API; later deal with protocol
  - Standardize only minimal set of features; higher-level features can be built on top
  - Provide several reference implementations
    - Not attempting to dictate any implementation details

Features

- Medium to coarse-grained calls (due to communication overhead)
- Asynchronous task-parallel programming
- Dynamic resource discovery and scheduling
- Manage IDLs on server side only

The GridRPC API - Fundamentals

- **Function handle – *grpc_function_handle_t***
  - Represents a mapping from a function name to an instance of that function on a particular server
  - Once created, calls using a function handle always go to that server
- **Session ID – *int***
  - Identifier representing a previously issued non-blocking call
  - Allows checking status, canceling, waiting for, or getting the error code of a non-blocking call

Initializing and Finalizing

- **int grpc_initialize(char *config_file_name)**
  - Reads *config_file_name* and initializes the system
  - Initialization is system dependent
  - Must be called before any other GridRPC calls
  - Return value:
    - GRPC_OK if successful
    - GRPC_ERROR if not successful
- **int grpc_finalize()**
  - Releases any resources being used by GridRPC
  - Return value:
    - GRPC_OK if successful
    - GRPC_ERROR if not successful

Function Handle Management

- **int grpc_function_handle_default(**
  grpc_function_handle_t *handle,
  char *func_name)
  - Get a function handle for function *func_name* using the default server
  - Server selection implementation-dependent
- **int grpc_function_handle_init(**
  grpc_function_handle_t *handle,
  char *host_name,
  int port, char *func_name)
  - Allows explicitly specifying the server in *host_name* and *port*
- **Both functions return GRPC_OK on success and GRPC_ERROR on failure**

Function Handle Management (cont.)

- **int grpc_function_handle_destruct(grpc_function_handle_t *handle)**
  - Release the memory allocated for *handle*
  - Returns GRPC_OK on success and GRPC_ERROR on failure
- **grpc_function_handle_t * grpc_get_handle(int sessionId)**
  - Returns the function handle corresponding to *sessionId*

GridRPC Call Functions

- **int grpc_call(grpc_function_handle_t *handle, ...)**
  - Blocking remote procedure call
  - Returns GRPC_OK on success, GRPC_ERROR on failure
- **int grpc_call_async(grpc_function_handle_t *handle, ...)**
  - Non-blocking remote procedure call
  - Returns session ID (positive integer) on success, GRPC_ERROR on failure
  - Session ID can be checked for completion later

GridRPC Call Functions Using ArgStack

- **int grpc_call_argstack(**
  grpc_function_handle_t *handle,
  grpc_arg_stack *stack)
  - Blocking call using argument stack
  - Returns GRPC_OK on success, GRPC_ERROR on failure
- **int grpc_call_argstack_async(**
  grpc_function_handle_t *handle,
  grpc_arg_stack *stack)
  - Non-blocking call using argument stack
  - Returns session ID (positive integer) on success, GRPC_ERROR on failure
  - Session ID can be checked for completion later

Asynchronous Session Control Functions

- **int grpc_probe(int sessionID)**
  - Checks whether call specified by *sessionID* has completed
  - Returns 1 if complete, 0 if not
- **int grpc_cancel(int sessionID)**
  - Cancels a previous call specified by *sessionID*
  - Returns GRPC_OK on success, GRPC_ERROR on failure

Asynchronous Wait Functions

- **int grpc_wait(int sessionID)**
  - Wait for call specified by *sessionID* to complete
  - Returns GRPC_OK on success, GRPC_ERROR on failure
- **int grpc_wait_and(int *idArray, int length)**
  - Wait for all calls specified in *idArray* to complete
  - *length* is the number of elements in *idArray*
  - Returns GRPC_OK if all the GridRPC calls succeeded
  - Returns GRPC_ERROR if any of the calls failed
  - Use grpc_get_error() to get the error value for a given session ID

Asynchronous Wait Functions (cont.)

- **int grpc_wait_or(int *idArray, int length, int *idPtr)**
  - Wait for any call specified in *idArray* to complete
  - *length* is the number of elements in *idArray*
  - On return, *idPtr* contains the session ID of the call that completed
  - Returns GRPC_OK if the completed call succeeded otherwise returns GRPC_ERROR
- **int grpc_wait_all()**
  - Wait for all calls to complete
  - Returns GRPC_OK if all calls succeeded
  - Returns GRPC_ERROR if any of the calls failed
  - Use grpc_get_error() to get the error value for a given session ID

Asynchronous Wait Functions (cont.)

- **int grpc_wait_any(int *idPtr)**
  - Wait for any call to complete
  - On return, *idPtr* contains the session ID of the call that completed
  - Returns GRPC_OK if the call (returned in *idPtr*) succeeded, otherwise returns GRPC_ERROR
  - Use grpc_get_error() to get the error value for a given session ID

Error Reporting Functions

- **void grpc_perror(char *str)**
  - Prints the error string of the last call, prefixed by *str*
- **char * grpc_error_string(int error_code)**
  - Gets the error string given a numeric error code
  - For *error_code* we typically pass in the global error value *grpc_errno*
- **int grpc_get_error(int sessionID)**
  - Get the error code for the non-blocking call specified by *sessionID*
- **int grpc_get_last_error()**
  - Get the error code for the last call

Argument Stack Functions

- **grpc_arg_stack * grpc_arg_stack_new(int max)**
  - Creates a new argument stack with at most *max* entries
  - Returns the new argument stack or NULL if there was an error
- **int grpc_arg_stack_destruct**
  - Frees resources associated with the argument stack

Argument Stack Functions (cont.)

- **int grpc_arg_stack_push_arg(**
  grpc_arg_stack *stack, void *arg)
  - Pushes *arg* onto *stack*
  - Returns 0 on success, -1 on failure
- **void * grpc_arg_stack_pop_arg(grpc_arg_stack *stack)**
  - Returns the top element of *stack* or NULL if the stack is empty
- **Arguments are passed in the order they were pushed onto the stack. For example, for the call F(a,b,c), the order would be:**
  - Push(a);
  - Push(b);
  - Push(c);

Example Program Segment

```
int n=5, incx=1, incy=1, status;
double ns_result = 0.0;
double dx[] = {10.0, 20.0, 30.0, 40.0, 50.0};
double dy[] = {60.0, 70.0, 80.0, 90.0, 100.0};
grpc_function_handle_t handle;

grpc_initialize(NULL);

grpc_function_handle_default(&handle, “ddot”);

status = grpc_call(&handle, &n, dx, &incx,
                   dy, &incy, &ns_result);

...
```

Example With Assigned Server

```
int n=5, incx=1, incy=1, status;
double ns_result = 0.0;
double dx[] = {10.0, 20.0, 30.0, 40.0, 50.0};
double dy[] = {60.0, 70.0, 80.0, 90.0, 100.0};
grpc_function_handle_t handle;

grpc_initialize(NULL);

grpc_function_handle_init(&handle,
  “cypher01.sinrg.utk.edu”, 9999, “ddot”);

status = grpc_call(&handle, &n, dx, &incx,
             dy, &incy, &ns_result);

...
```

Asynchronous Example

```
grpc_initialize(NULL);

grpc_function_handle_default(&handle, “ddot”);

sessionID = grpc_call_async(&handle, &n, dx,
     &incx, dy, &incy, &ns_result);

/* do some other work... */

status = grpc_wait(sessionID);
```

Example Using Argument Stack

```
grpc_initialize(NULL);

stack = grpc_arg_stack_new(10);
grpc_arg_stack_push_arg(stack, &n);
grpc_arg_stack_push_arg(stack, dx);
grpc_arg_stack_push_arg(stack, &incx);
grpc_arg_stack_push_arg(stack, dy);
grpc_arg_stack_push_arg(stack, &incy);
grpc_arg_stack_push_arg(stack, &ns_result);

grpc_function_handle_default(&handle, “ddot”);

status = grpc_call_arg_stack(&handle, stack);
...
```

Session Control Example

```
int len = 10, sessionID, c = 5;
grpc_function_handle_default(&handle,“sleeptest”);
sessionID = grpc_call_async(&handle, &len);
for(;;) {
  if(grpc_probe(sessionID)) {
    grpc_wait(sessionID);
    break;
  }
  sleep(1);
  if(c-- <= 0) {  /* if we’re tired of waiting */
    grpc_cancel(sessionID);
    break;
  }
}
```

Example Using grpc_wait_all

```
int i, status, len[] = {15,8,5,10,20},
  sessionID[NUM_REQ];
grpc_initialize(NULL);

grpc_function_handle_default(&handle,"sleeptest");

for(i=0;i<NUM_REQ;i++)
  sessionID[i] =
    grpc_call_async(&handle, &len[i]);

/* Wait for all calls to complete */
status = grpc_wait_all();
```

Example Using grpc_wait_any

```
int i, status, len[] = {15,8,5,10,20},
   sessionID[NUM_REQ];

grpc_function_handle_default(&handle,"sleeptest");

for(i=0;i<NUM_REQ;i++)
  sessionID[i] =
    grpc_call_async(&handle, &len[i]);

for(i=0;i<NUM_REQ;i++) {
  int id;
  status = grpc_wait_any(&id);
}
```

Example Using grpc_wait_and

```
int i, status, len[] = {15,8,5,10,20},
  sessionID[NUM_REQ],
  ss1 = NUM_REQ/2, ss2 = NUM_REQ-(NUM_REQ/2);

grpc_function_handle_default(&handle,"sleeptest");

for(i=0;i<NUM_REQ;i++)
  sessionID[i] =
      grpc_call_async(&handle, &len[i]);

status = grpc_wait_and(sessionID,ss1);

status = grpc_wait_and(sessionID+ss1,ss2);
```

Example Using grpc_wait_or

```
int i, status, len[] = {15,8,5,10,20},
  sessionID[NUM_REQ], id,
  ss1 = NUM_REQ/2, ss2 = NUM_REQ-(NUM_REQ/2);

grpc_function_handle_default(&handle,"sleeptest");

for(i=0;i<ss1;i++)
  status = grpc_wait_or(sessionID, ss1, &id);

for(i=0;i<ss2;i++)
  status = grpc_wait_or(sessionID+ss1, ss2, &id);
```

Error Handling Example

```
status = grpc_call(&handle, &n, dx, &incx, dy,
  &incy, &ns_result);

if(status != GRPC_OK) {
  printf("GRPC error status  = %d¥n", status);

  grpc_perror("grpc_call");

     /* ----- OR ----- */

  printf("grpc_call: %s¥n“,
     grpc_error_string(grpc_get_last_error());

}
```

Handling Errors from Asynchronous Calls

```
status = grpc_wait_and(sessionID,ss1);

if(status == GRPC_ERROR) {
  for(i=0;i<ss1;i++)
    if(grpc_get_error(sessionID[i]) != GRPC_OK)
      printf("Warning: call %d failed.¥n", i);
}
```

Reference Implementations

- **Currently two complete reference implementations exist**
  - NetSolve (UTK)
  - Ninf-G (AIST)
- **Systems are largely similar in operation except that:**
  - Ninf-G is built on top of Globus
  - NetSolve performs its own scheduling, load balancing, etc, but can also interact with other systems
  - Therefore, the protocols and IDLs differ

Issues/Challenges

- **Service Name Collisions**
  - Different systems may have different calling sequences for the same function
  - So, in some sense a GridRPC program may be bound to a particular implementation
- **Function handle binding**
  - Function handle represents a persistent function-to-server mapping
  - Binding a function handle a long time before the actual calls may result in less satisfactory resource selection because of changing server workloads and network conditions

Issues/Challenges (cont.)

- **Implementability**
  - Does the specification of the API preclude implementation on any systems?
- **Persistent Data**
- **Security**
- **Thread Safety**
  - At least not thread “hostile”

Conclusions

- **Attempting to provide:**
  - Simple API upon which higher-level services could be implemented
  - Low burden on programmer attempting to transition code to the Grid
- **Future work:**
  - Charter for GridRPC Working Group still under development
  - Consider the many challenges from previous slides as we standardize the API

- **GridRPC-WG was officially approved and will have a meeting at GGF-8 (at Seattle)**
  - GGF GridRPC WG home page
    - http://www.ggf.org/7_APM/GRPC.htm
- **Please subscribe the ML**
  - Send e-mail Majordomo@gridforum.org

    subscribe gridrpc-wg

Ninf-G

Tutorial Goal

Understand "how to develop" Grid applications using Ninf-G

Difference between Ninf-G and Netsolve

Ninf-G

- **Is Implemented on top of Globus toolkit**
  - Good: Remote library can access to the globus-enabled resources – ex. GridFTP servers
  - Bad: you have to install the Globus toolkit in advance
- **Has Simple IDL**
  - ANSI-C like IDL syntax
  - Do not have GUI
- **Does not provide automatic servers selection capability**

Outline

Ninf-G

- Overview and architecture
- How to install Ninf-G
- How to build remote libraries (server side ops)
- How to call remote libraries (client side ops)
- Java API
- Ongoing work and future plan

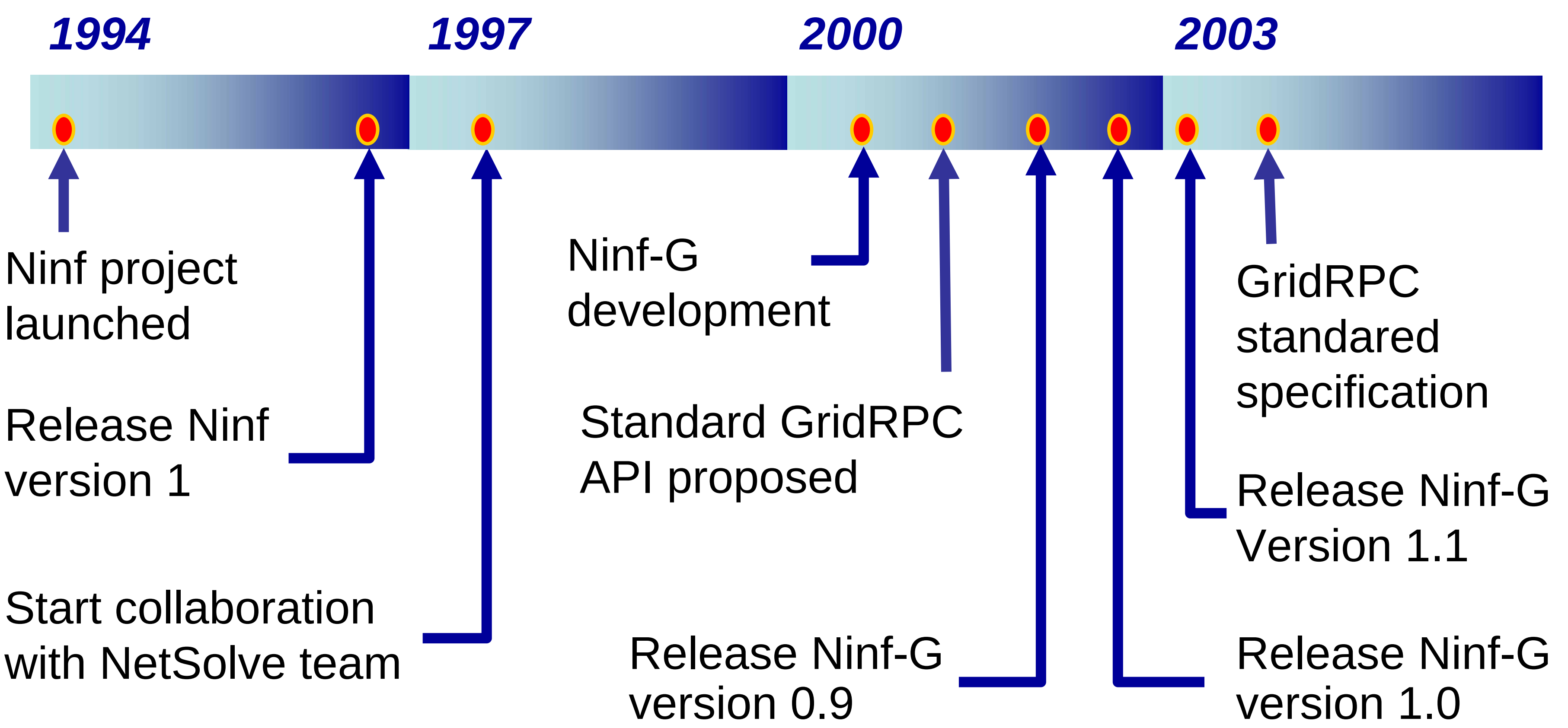
Ninf-G

Overview and Architecture

Ninf Project

- **Started in 1994**
- **Collaborators from various organizations**
  - AIST
    - Satoshi Sekiguchi, Umpei Nagashima, Hidemoto Nakada, Hiromitsu Takagi, Osamu Tatebe, Yoshio Tanaka,
      Kazuyuki Shudo
  - University of Tsukuba
    - Mitsuhisa Sato, Taisuke Boku
  - Tokyo Institute of Technology
    - Satoshi Matsuoka, Kento Aida, Hirotaka Ogawa
  - Tokyo Electronic University
    - Katsuki Fujisawa
  - Ochanomizu University
    - Atsuko Takefusa
  - Kyoto University

Brief History of Ninf/Ninf-G



What is Ninf-G?

- **A software package which supports programming and execution of Grid applications using GridRPC.**
- **Ninf-G includes**
  - C/C++, Java APIs, libraries for software development
  - IDL compiler for stub generation
  - Shell scripts to
    - compile client program
    - build and publish remote libraries
  - sample programs
  - manual documents

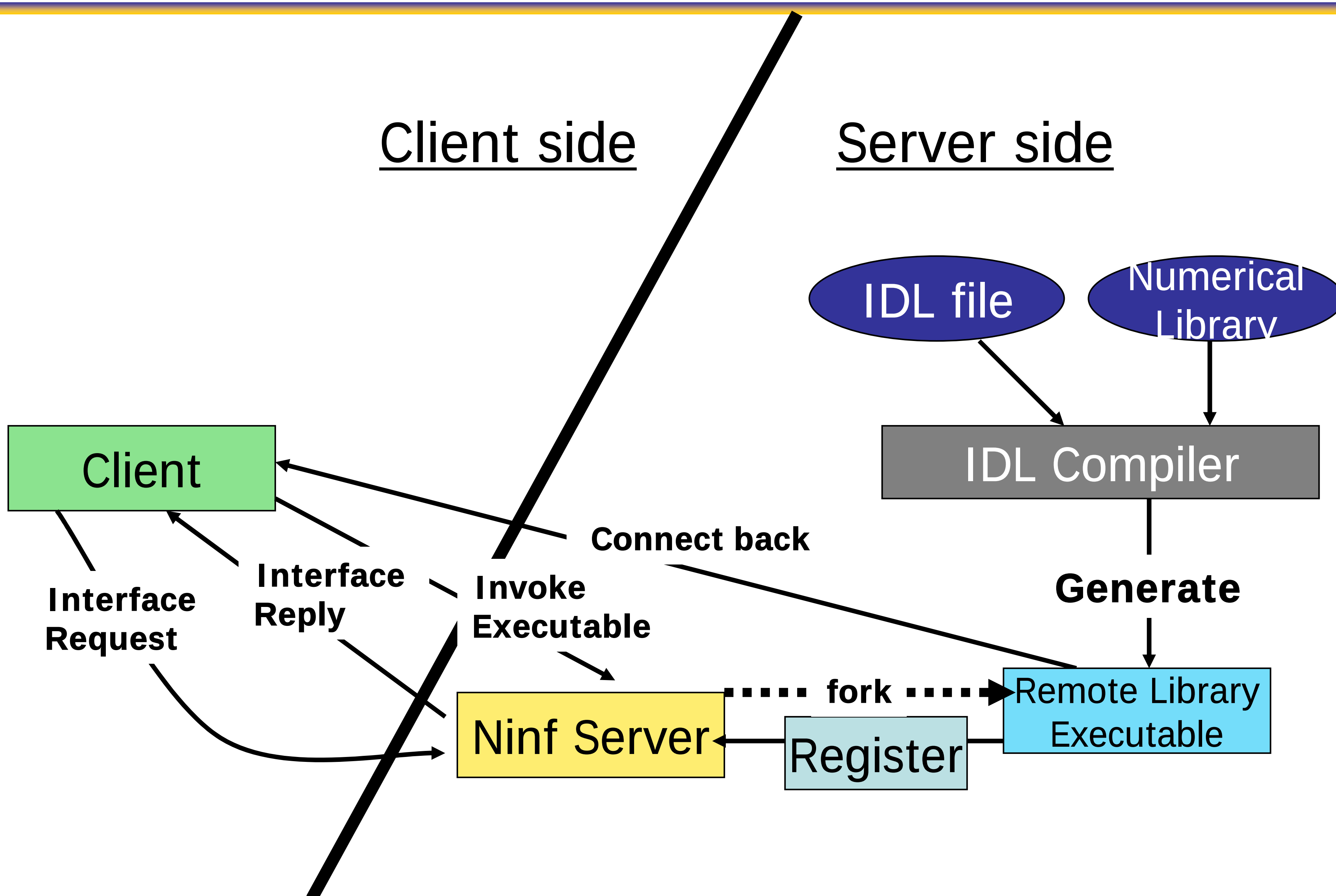
Ninf-G: Features At-a-Glance

- **Ease-of-use, client-server, Numerical-oriented RPC system**
- **No stub information at the client side**
- **User's view: ordinary software library**
  - Asymmetric client vs. server
- **Built on top of the Globus Toolkit**
  - Uses GSI, GRAM, MDS, GASS, and Globus-IO
- **Supports various platforms**
  - Ninf-G is available on Globus-enabled platforms
- **Client APIs: C/C++, Java**

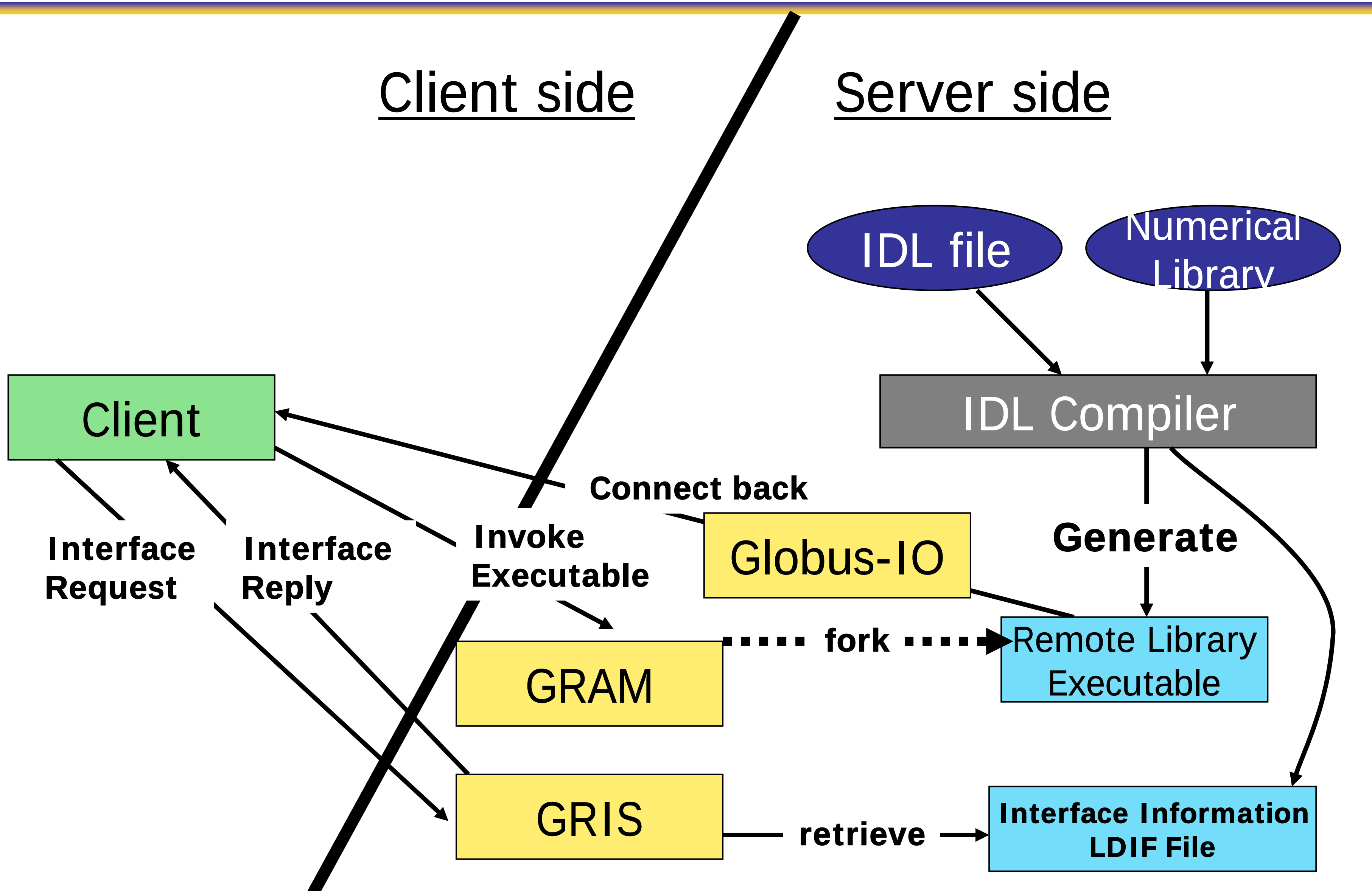
Sample Architecture Review

- **Client API**
  - Provides users easy to use API
- **Remote Library Executable**
  - Execute numerical operation
- **Information Server**
  - Provides library interface info.
- **Invocation Server**
  - Invokes remote library executable
- **IDL compiler**
  - Compiles Interface description
  - Generates 'stub main' for remote library executable
  - Helps to link the executable
- **Register driver**
  - Registers remote library executable into the Information Server

Architecture of Ninf



Architecture of Ninf-G



Ninf-G

How to install Ninf-G

Getting Started

- **Globus Toolkit (ver.1 or 2) must be installed prior to the Ninf-G installation**
  - from source bundles
  - all bundles (resource, data, info - server/client/SDK) (recommended)
  - same flavor
- **Create a user ‘ninf’ (recommended)**
- **Download Ninf-G**
  - http://ninf.apgrid.org/packages/
  - Ninf-G is provided as a single gzipped tarball.
- ***Before starting installation, be sure that Globus related env. variable is correctly defined (e.g. GLOBUS_LOCATION, GLOBUS_INSTALL_PATH)***

Installation Steps (1/4)

- **Uncompress and untar the tarball.**

  % gunzip –c ng-latest.tgz | tar xvf –

  This creates a directory "*ng-1.0*" which makes up the source tree along with its subdirectories.

Installation Steps (2/4)

- **Change directory to the “*ng-1.0*” directory and run configure script.**

  % cd ng-1.0
  % ./configure

- **Example:**

  % ./configure --prefix=/usr/local/ng

  - Notes for GT2.2 or later
    - --without-gpt option is required if GPT_LOCATION is not defined.
  - Run configure with --help option for detailed options of the configure script.

Installation Steps (3/4)

- **Compile all components and install Ninf-G**

  % make
  % make install

- **Register the host information by running the following commands (required at the server side)**

  % cd ng-1.0/bin
  % ./server_install

  This command will create
  ${GLOBUS_LOCATION}/var/gridrpc/
  and copies a LDIF file for host information

  - Notes: these commands should be done as *root* (for GT2) or user *globus* (for GT1)

Installation Steps (4/4)

- **Add the information provider to GRIS (required at the server side)**
  - For GT1:
    - Add the following line to ${GLOBUS_DEPLOY_PATH}/etc/grid-info-resource.conf

      0 cat ${GLOBUS_DEPLOY_PATH}/var/gridrpc/*.ldif
  - For GT2:
    - Add the following line to ${GLOBUS_LOCATION}/etc/grid-info-slapd.conf

      include ${GLOBUS_LOCATION}/etc/grpc.schema

      The line should be put just below the following line:

      include ${GLOBUS_LOCATION}/etc/grid-info-resource.schema
    - Restart GRIS.

      % ${GLOBUS_LOCATION}/sbin/SXXGris stop
      % ${GLOBUS_LOCATION}/sbin/SXXGris start

Ninf-G

How to Build Remote Libraries
- server side operations -

Ninf-G remote libraries

- **Ninf-G remote libraries are implemented as executable programs (Ninf-G executables) which**
  - contains stub main routine and numerical library
  - will be spawned off by GRAM
- **The stub routine handles**
  - communication with clients and Ninf-G system itself
  - argument marshalling
- **Underlying executable (numerical library) can be written in C, C++, Fortran, etc.**

Prerequisite

- **GRAM (gatekeeper) and GRIS are correctly configured and running**
- **Following env. variables are appropriately defined:**
  - GLOBUS_LOCATION / GLOBUS_INSTALL_PATH
  - NS_DIR
- **Add ${NS_DIR}/bin to $PATH (recommended)**
- **Notes for dynamic linkage of the Globus shared libraries: Globus dynamic libraries (shared libraries) must be linked with the Ninf-G stub executables. For example on Linux, this is enabled by adding ${GLOBUS_LOCATION}/lib in /etc/ld.so.conf and run ldconfig command.**

How to build Ninf-G remote libraries (1/3)

- **Write an interface information using Ninf-G Interface Description Language (Ninf-G IDL).**
  Example:

```
Module mmul;
Define dmmul (IN int n,
              IN double A[n][n],
              IN double B[n][n],
              OUT double C[n][n])
Require "libmmul.o"
Calls "C" dmmul(n, A, B, C);
```

- **Compile the Ninf-G IDL with Ninf-G IDL compiler**

  % ns_gen <IDL_FILE>

  ns_gen generates stub source files and a makefile (<module_name>.mak)

How to build Ninf-G remote libraries (2/3)

- **Compile stub source files and generate Ninf-G executables and LDIF files (used to register Ninf-G remote libs information to GRIS).**

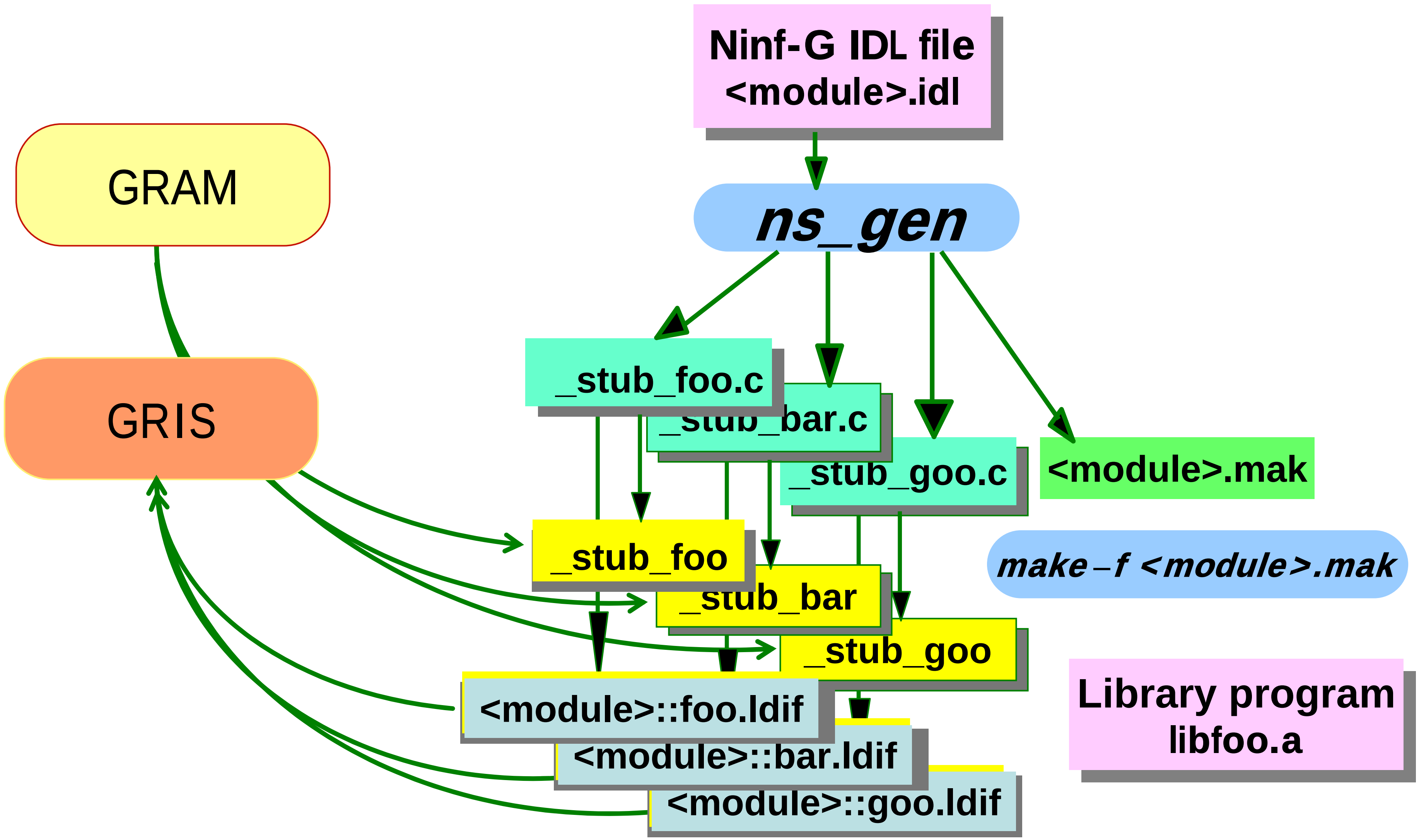
  % make –f <module_name>.mak

- **Publish the Ninf-G remote libraries**

  % make –f <module_name>.mak install

  This copies the LDIF files to ${GLOBUS_LOCATION}/var/gridrpc

How to build Ninf-G remote libraries (3/3)



Ninf-G IDL Statements (1/2)

- **Module** ***module_name***
  - specifies the module name.
- **CompileOptions** ***"options"***
  - specifies compile options which should be used in the resulting makefile
- **Library** ***"object files and libraries"***
  - specifies object files and libraries
- **FortranFormat** ***"format"***
  - provides translation format from C to Fortran.
  - Following two specifiers can be used:
    - %s: original function name
    - %l: capitalized original function name
  - Example:

    FortranFormat "_%l_";
    Calls "Fortran" fft(n, x, y);

    will generate function call

    FFT(n, x, y);

    in C.

Ninf-G IDL Statements (2/2)

- **Globals** ***{ … C descriptions }***
  - declares global variables shared by all functions
- **Define** ***routine_name*** **(*parameters*…)**
  ["*description*"]
  [Required "*object files or libraries*"]
  [Backend "MPI"|"BLACS"]
  [Shrink "yes"|"no"]
  {*{C descriptions}* |
  Calls "C"|"Fortran" *calling sequence*}
  - declares function interface, required libraries and the main routine.
  - Syntax of parameter description:
    [mode-spec] [type-spec] formal_parameter [[dimension [:range]]+]+

Syntax of parameter description (detailed)

- **mode-spec: one of the following**
  - IN: parameter will be transferred from client to server
  - OUT: parameter will be transferred from server to client
  - INOUT: at the beginning of RPC, parameter will be transferred from client to server. at the end of RPC, parameter will be transferred from server to client
  - WORK: no transfers will be occurred. Specified memory will be allocated at the server side.
- **type-spec should be either *char, short, int, float, long, longlong, double, complex, or filename*.**
- **For arrays, you can specify the size of the array. The size can be specified using scalar IN parameters.**
  - Example:
    IN int n, IN double a[n]

Sample Ninf-G IDL (1/2)

Matrix Multiply

```
Module matrix;

Define dmmul (IN int n,
              IN double A[n][n],
              IN double B[n][n],
              OUT double C[n][n])
"Matrix multiply: C = A x B"
Required "libmmul.o"
Calls "C" dmmul(n, A, B, C);
```

Sample Ninf-G IDL (2/2)

ScaLAPACK (pdgesv)

```
Module SCALAPACK;

CompileOptions "NS_COMPILER = cc";
CompileOptions "NS_LINKER = f77";
CompileOptions "CFLAGS = -DAdd_ -O2 –64 –mips4 –r10000";
CompileOptions "FFLAGS = -O2 -64 –mips4 –r10000";
Library "scalapack.a pblas.a redist.a tools.a libmpiblacs.a –lblas –lmpi –lm";

Define pdgesv (IN int n, IN int nrhs, INOUT double global_a[n][lda:n], IN int lda,
                    INOUT double global_b[nrhs][ldb:n], IN int ldb, OUT int info[1])
Backend "BLACS"
Shrink "yes"
Required "procmap.o pdgesv_ninf.o ninf_make_grid.of Cnumroc.o descinit.o"
Calls "C" ninf_pdgesv(n, nrhs, global_a, lda, global_b, ldb, info);
```

Ninf-G

How to call Remote Libraries
- client side APIs and operations -

Grid Technology Research Center AIST

Prerequisite

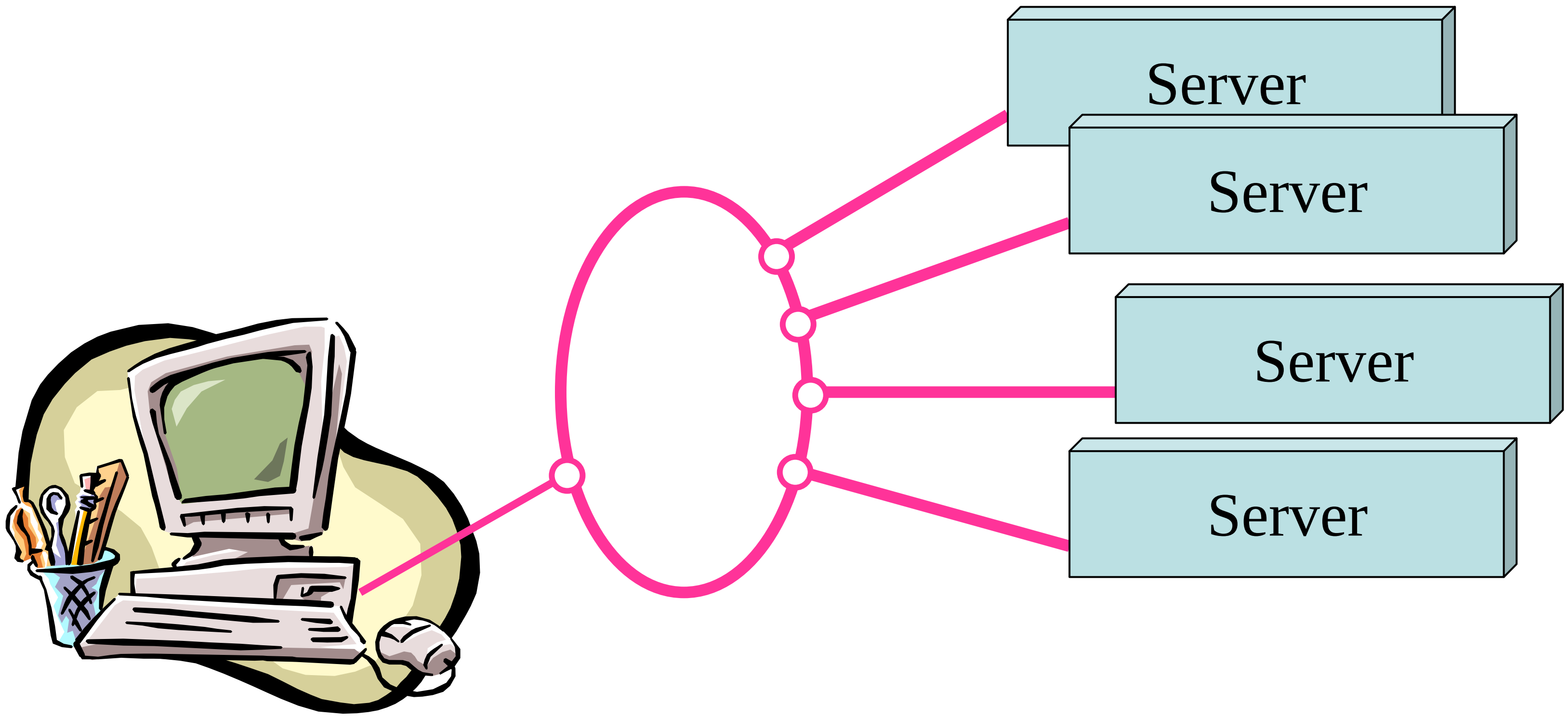
- **should be able to submit jobs to the Globus Gatekeeper on servers**
  - has a user certificate
  - User's dn appears on grid-mapfile
  - Run *grid-proxy-init* command before executing Ninf-G apps
- **Following env. variables are appropriately defined:**
  - GLOBUS_LOCATION / GLOBUS_INSTALL_PATH
  - NS_DIR
- **Add ${NS_DIR}/bin to $PATH (recommended)**

(Client) User's Scenario

- Write client programs using GridRPC API
- Compile and link with the supplied Ninf-G client compile driver (*ns_client_gen*)
- Write a configuration file in which runtime environments can be described
- Run *grid-proxy-init* command
- Run the program

Task Parallel Application

- **Parallel RPCs using asynchronous call.**



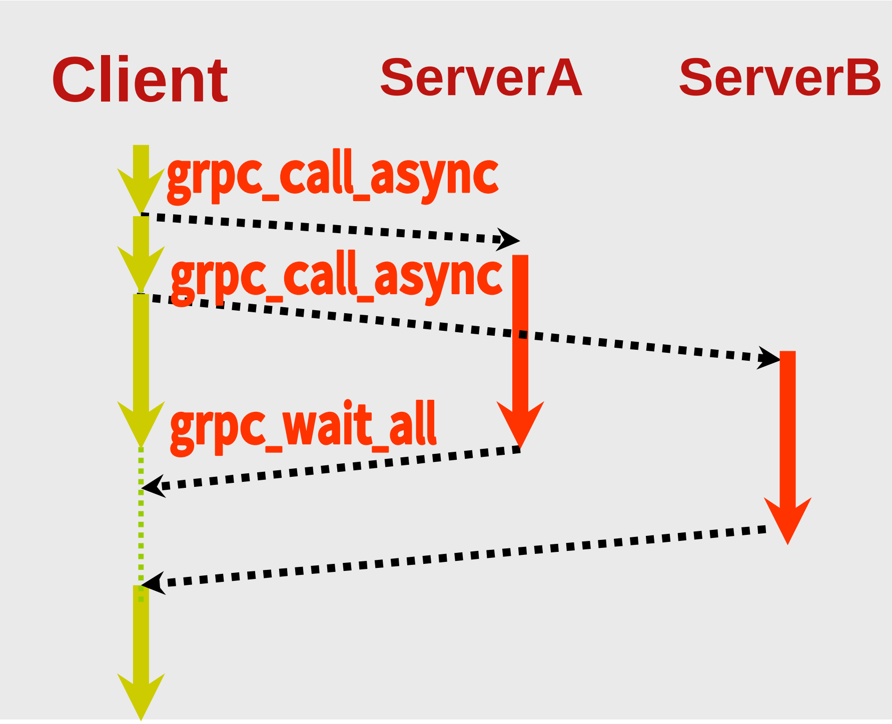
Task Parallel Application

Asynchronous Call

```
grpc_call_async(...);
```

Waiting for reply

```
grpc_wait(sessionID);
grpc_wait_all();
grpc_wait_any(idPtr);
grpc_wait_and(idArray, len);
grpc_wait_or(idArray, len, idPtr);
grpc_cancel(sessionID);
```



Various task parallel programs spanning clusters are easy to write

How the server can be specified?

- **Server is determined when the function handle is initialized.**
  - *grpc_function_handle_init();*
    - hostname is given as the second argument
  - *grpc_function_handle_default();*
    - hostname is specified in the client configuration file which must be passed as the first argument of the client program.

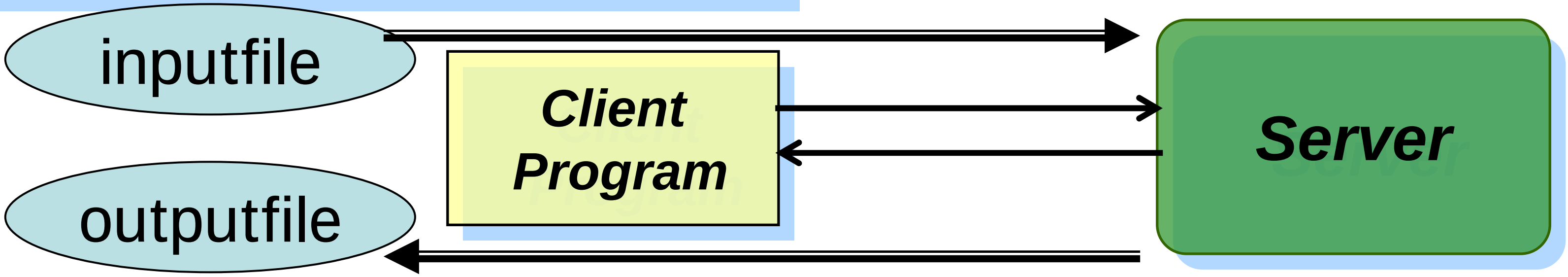
- **Notes: Ninf-G Ver.1 is designed to focus on GridRPC core mechanism and it does not support any brokering/scheduling functions.**

Automatic file staging

- **Ninf-G provides "Filename" type**
- **Local file is automatically shipped to the server**
- **Server side output file is forwarded to the client**

```
grpc_function_handle_default (
  &handle, "plot/plot");
grpc_call (&handle,
           "input.gp",
           "output.ps");
```

```
Module plot;
Define plot (IN filename src,
             OUT filename out)
{ char buf[1000];
  sprintf(buf, "gnuplot %s > %s,
          src, out);
  system(buf);}
```



Compile and run

- **Compile the program using *ns_client_gen* command.**

  % ns_client_gen –o myapp app.c

- **Before running the application, generate a proxy certificate.**

  % grid-proxy-init

- **When running the application, client configuration file must be passed somehow.**

  % ./myapp config.cl [args…]

Client Configuration File (1/2)

- **Specifies runtime environments.**
- **Available attributes:**
  - host
    - specifies client's hostname (callback contact)
  - port
    - specifies client's port number (callback contact)
  - serverhost
    - specifies default server's hostname
  - ldaphost
    - specifies hostname of GRIS/GIIS
  - ldapport
    - specifies port number of GRIS/GIIS (default: 2135)
  - vo_name
    - specifies Mds-Vo-Name for querying GIIS (default: local)
  - jobmanager
    - specifies jobmanager (default: jobmanager)

Client Configuration File (2/2)

- **Available attributes (cont'd):**
  - loglevel
    - specifies log leve (0-3, 3 is the most detail)
  - redirect_outerr
    - specifies whether stdout/stderr are redirect to the client side (yes or no, default: no)
  - forkgdb, debug_exe
    - enables debugging Ninf-G executables using gdb at server side (TRUE or FALSE, default: FALSE)
  - debug_display
    - specifies DISPLAY on which xterm will be opened.
  - debug_xterm
    - specifies absolute path of xterm command
  - debug_gdb
    - specifies absolute path of gdb command

Sample Configuration File

```
# call remote library on UME cluster
serverhost = ume.hpcc.jp

# grd jobmanager is used to launch jobs
jobmanager = jobmanager-grd

# query to ApGrid GIIS
ldaphost = mds.apgrid.org
ldapport = 2135
vo_name = ApGrid

# get detailed log
loglevel = 3
```

Ninf-G

Examples

Examples

- **Ninfy the existing library**
  - Matrix multiply
- **Ninfy task-parallel program**
  - Calculate PI using a simple Monte-Carlo Method

Matrix Multiply

- **Server side**
  - Write an IDL file
  - Generate stubs
  - Register stub information to GRIS
- **Client side**
  - Change local function call to remote library call
  - Compile by ns_client_gen
  - write a client configuration file
  - run the application

Matrix Multiply - Sample Code -

```
void  mmul(int n, double * a,
           double * b, double * c) {
  double t;
  int i, j, k;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      t = 0;
      for (k = 0; k < n; k++){
        t += a[i * n + k] * b[k * n + j]; }
      c[i*N+j] = t;
}}}
```

- **The matrix do not itself embody size as type info.**

Matrix Multiply- Server Side (1/3) -

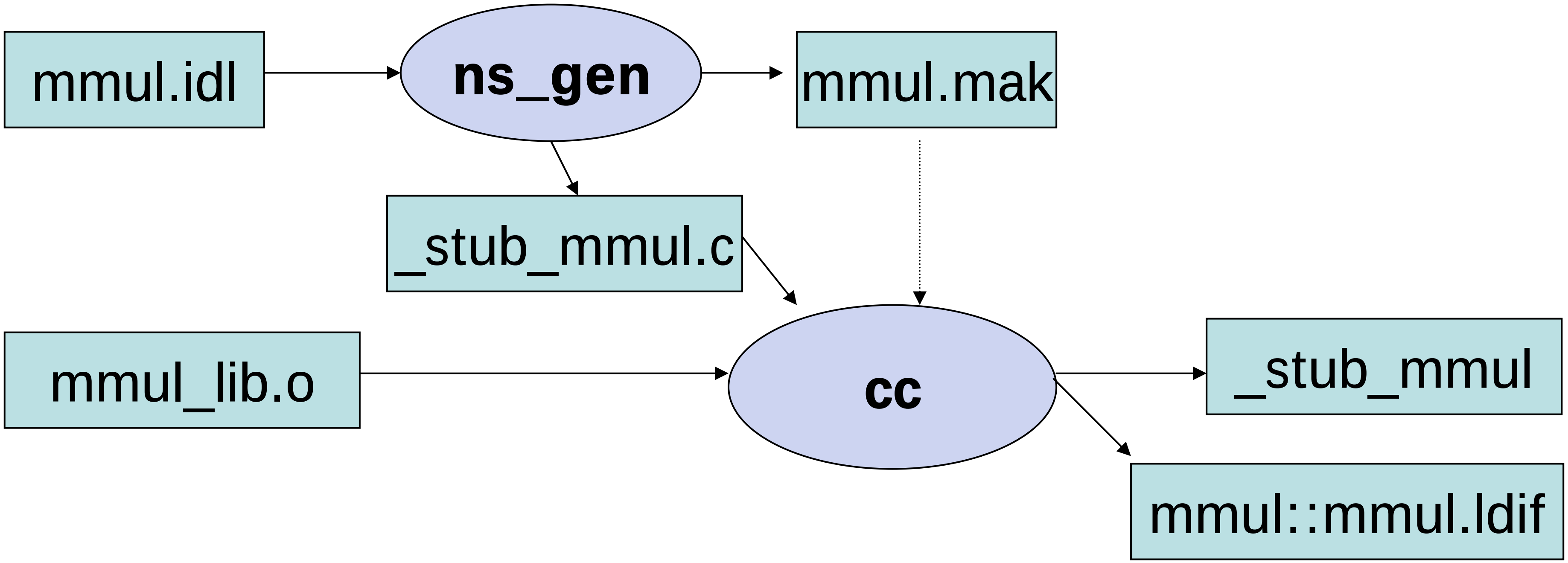
- **Write IDL file describing the interface information (mmul.idl)**

```
Module mmul;
Define mmul(IN int N,
            IN double A[N*N],
            IN double B[N*N],
            OUT double C[N*N])
“Matrix Multiply: C = A x B"
Required "mmul_lib.o"
Calls "C" mmul(N, A, B, C);
```

Matrix Multiply - Server Side (2/3) -

- **Generate stub source and compile it**

```
> ns_gen mmul.idl
> make -f mmul.mak
```



Matrix Multiply - Server Side (3/3) -

Regisgter stub information to GRIS

```
dn: GridRPC-Funcname=mmul/mmul, Mds-Software-deployment=GridRPC-Ninf-G, __ROO
T_DN__
objectClass: GlobusSoftware
objectClass: MdsSoftware
objectClass: GridRPCEntry
Mds-Software-deployment: GridRPC-Ninf-G
GridRPC-Funcname: mmul/mmul
GridRPC-Module: mmul
GridRPC-Entry: mmul
GridRPC-Path: /usr/users/yoshio/work/Ninf-G/test/_stub_mmul
GridRPC-Stub:: PGZ1bmN0aW9uICB2ZXJzaW9uPSIyMjEuMDAwMDAwIiA+PGZ1bmN0aW9
 PSJtbXVsIiBlbnRyeT0ibW11bCIgLz4gPGFyZyBkYXRhX3R5cGU9ImludCIgbW9kZV90eXBl
 PSJpbiIgPgogPC9hcmc+CiA8YXJnIGRhdGFfdHlwZT0iZG91YmxlIiBtb2RlX3R5cGU9Imlu
 IiA+CiA8c3Vic2NyaXB0PjxzaXplPjxleHByZXNzaW9uPjxiaV9hcml0aG1ldGljIG5hbWU9
```

```
> make –f mmul.mak install
```

Matrix Multiply - Client Side (1/3) -

Modify source code

```
main(int argc, char ** argv){
  grpc_function_handle_t handle;
  …
  grpc_initialize(argv[1]);
  …
  grpc_function_handle_default(&handle, “mmul/mmul”);
  …
  if (grpc_call(&handle, n, A, B, C) == GRPC_ERROR) {
    …
  }
  …
  grpc_function_handle_destruct(&handle);
  grpc_finalize();
}
```

Matrix Multiply - Client Side (2/3) -

- **Compile the program by *ns_client_gen***

```
> ns_client_gen -o mmul_ninf mmul_ninf.c
```

- **Generate a proxy certificate**

```
> grid-proxy-init
```

- **Write a client configuration file**

```
serverhost = ume.hpcc.jp
ldaphost = ume.hpcc.jp
ldapport = 2135
jobmanager = jobmanager-grd
loglevel = 3
redirect_outerr = no
```

Matrix Multiply - Client Side (3/3) -

- **Generate a proxy certificate**

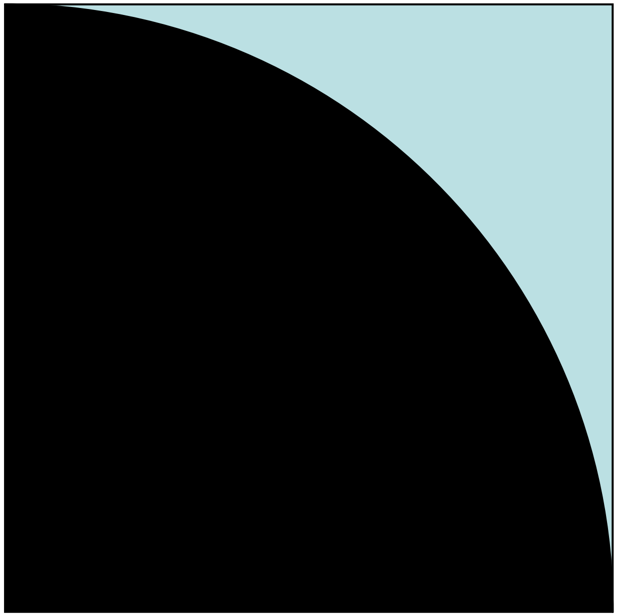
```
> grid-proxy-init
```

- **Run**

```
> ./mmul_ninf config.cl
```

Task Parallel Programs (Compute PI using Monte-Carlo Method)

- **Generate a large number of random points within the square region that exactly encloses a unit circle (1/4 of a circle)**
  - PI = 4 *p*



Compute PI - Server Side -

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;
}
```

Compute PI - Client Side-

```
#include "grpc.h"
#define NUM_HOSTS 8
char * hosts[] =
  {"host00", "host01", "host02", "host03",
   "host04", "host05", "host06", "host07"};
grpc_function_handle_t handles[NUM_HOSTS];

main(int argc, char ** argv){
  double pi;
  long times, count[NUM_HOSTS], sum;
  char * config_file;
  int i;
  if (argc < 3){
    fprintf(stderr,
    "USAGE: %s CONFIG_FILE TIMES ¥n",
    argv[0]);
    exit(2);
  }
  config_file = argv[1];
  times = atol(argv[2]) / NUM_HOSTS;

  /* Initialize */
  if (grpc_initialize(config_file)
      != GRPC_OK){
    grpc_perror("grpc_initialize");
    exit(2);
  }
```

```
  /* Initialize Function Handles */
  for (i = 0; i < NUM_HOSTS; i++)
    grpc_function_handle_init(&handles[i],
         hosts[i], port, "pi/pi_trial");

  for (i = 0; i < NUM_HOSTS; i++)
    /* Asynchronous RPC */
    if (grpc_call_async(&handles[i], i,
          times, &count[i]) == GRPC_ERROR){
      grpc_perror("pi_trial");
      exit(2);
    }
  /* Wait all outstanding RPCs */
  if (grpc_wait_all() == GRPC_ERROR){
    grpc_perror("wait_all");
    exit(2);
  }
  /* Display result */
  for (i = 0, sum = 0; i < NUM_HOSTS; i++)
    sum += count[i];
  pi = 4.0 *
    ( sum / ((double) times * NUM_HOSTS));
  printf("PI = %f¥n", pi);
  /* Finalize */
  grpc_finalize();
}
```

Ninf-G

Java API

Java language characteristics

- **Java language has**
  - Automatic garbage collection
    - Memory leak free
  - Integrated Multi-thread capability
    - Easy to handle several servers
  - Application portability
    - Easy to install softwares
    - No architecture dependent features
  - Affinity with Web-based systems
    - Easy to work with JSP or Servlet

Ninf-G Java API

- **Java API**
  - Client-side API
  - Implemented using Java CoG kit
  - Provides API similar to C based API
    - Simplified to conform with Object-oriented style
    - No-async and wait API taking advantage of Multi-thread feature
  - Property based configuration file
    - Uses Java standard property file format

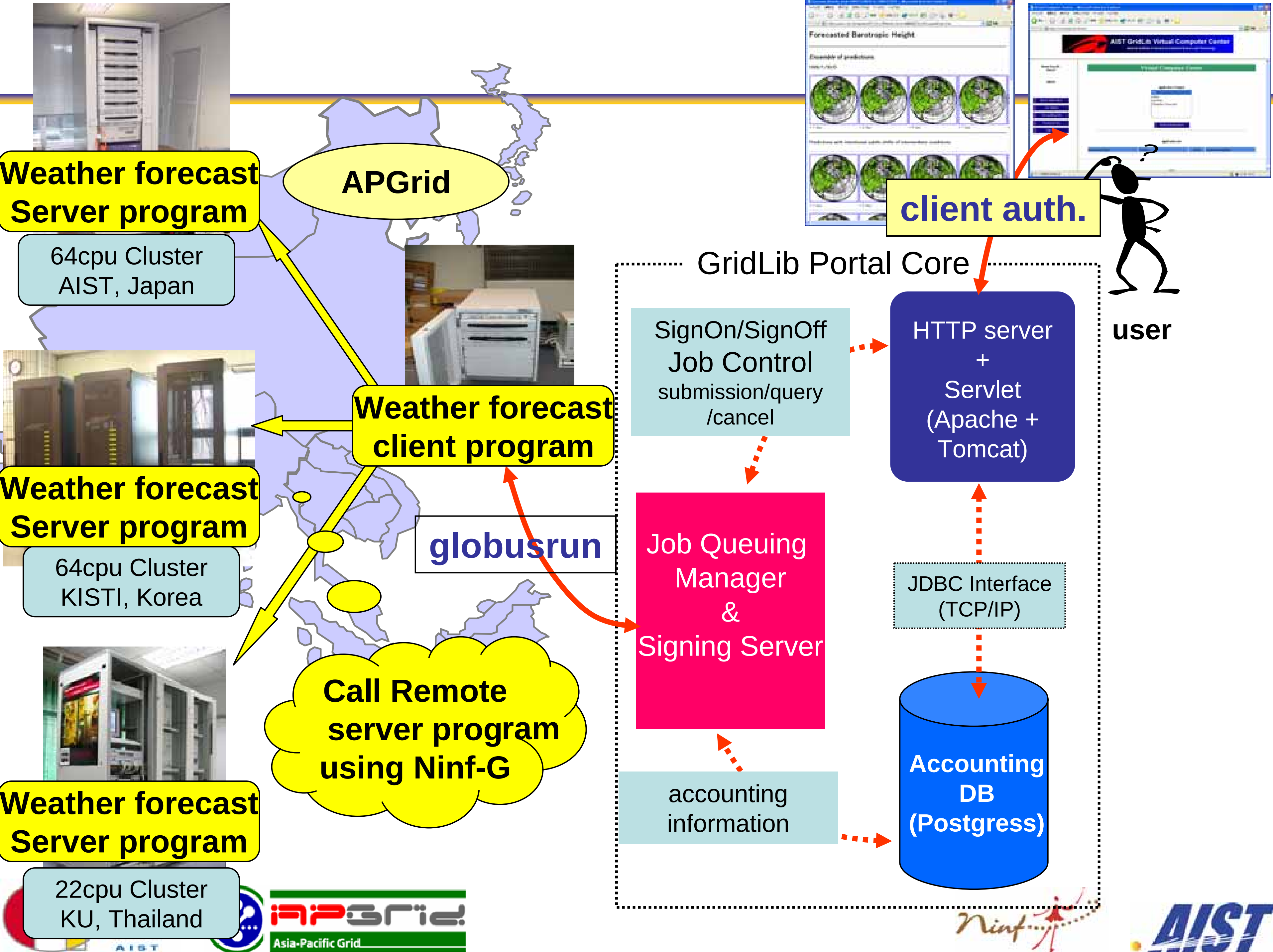
Java API Sample

```
// create an instance of the GrpcClient
GrpcClient remoteClient =
    GrpcClientFactory.getClient("org.apgrid.grpc.client.NgGrpcClient");

// initialize the GrpcClient specifying a properties file
remoteClient.activate(PROPERTY_FILENAME);

// create a handle for a remote function
GrpcHandle handle = client.getHandle("test/int_test");

// make a call using the handle
handle.callWith(new Integer(N), a, b);

//dispose the handle
 handle.dispose();

// deactivate the client instance
 remoteClient.deactivate();
```

Ninf-G

Demonstration

Grid
Technology
Research
Center
AIST



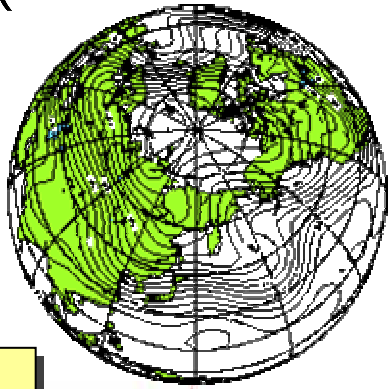
Weather forecast Server program
64cpu Cluster AIST, Japan
APGrid
Weather forecast client program
Weather forecast Server program
64cpu Cluster KISTI, Korea
globusrun
Weather forecast Server program
22cpu Cluster KU, Thailand
Call Remote server program using Ninf-G
client auth.
GridLib Portal Core
user
SignOn/SignOff
Job Control
submission/query /cancel
HTTP server + Servlet (Apache + Tomcat)
Job Queuing Manager & Signing Server
JDBC Interface (TCP/IP)
accounting information
Accounting DB (Postgress)
Forecasted Barotropic Height
AIST GridLib Virtual Computer Center
Asia-Pacific Grid
AIST

Weather Forecasting System

- **Goal**
  - Long term, global weather prediction
    - Winding of Jet-Stream
    - Blocking phenomenon of high atmospheric pressure
- **Barotropic S-Model**
  - Weather forecasting model proposed by Prof. Tanaka
  - Simple and precise

    Modeling complicated 3D turbulence as a horizontal one
    - 200 sec for 100-days prediction/ 1 simulation (Pentium III machine)

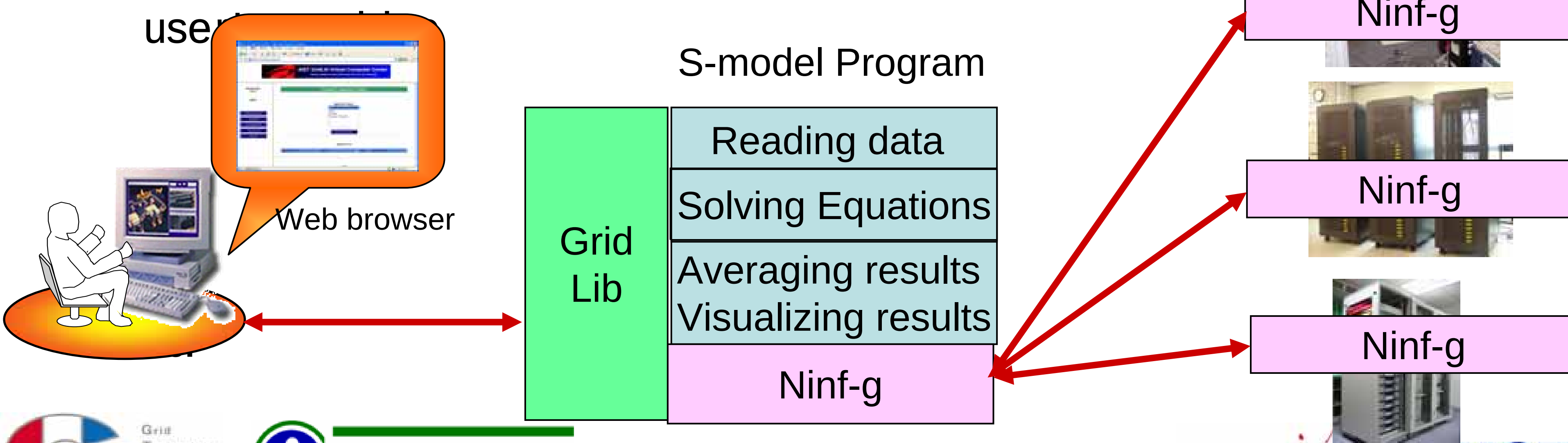
    Keep high precision over long periods
    - Taking a statistical ensemble mean
      - ~50 simulations
    - Introducing perturbation at every time step
    - Typical parameter survey



Gridifying the program enables quick response

Ninfy the Program

- **Dividing a program into two parts as a client-server system**
  - Client:
    - Pre-processing: reading input data
    - Post-processing: averaging results of ensembles, visualizing results
  - Server
    - Weather forecasting simulation
- **Integrating the system by using grid-middle ware**
  - Ninf-g library: communication between C/S
  - Grid lib portal: easy and secure access from a user



Ongoing and Future Work

Towards the release of Ninf-G Ver.2

Planned Additional Features

- **Get stub information without using LDAP**
  - To avoid unstableness of MDS
  - Enables to install Ninf-G without ‘globus’ privilege
- **Initialize multiple function handles with one GRAM call**
  - Reduce invocation cost
- **Callback from remote executable to the client**
  - Heatbeat monitoring, visualization, debugging
- **revise the structure of client configuration file**
  - multiple servers
  - jobmanager for each server
  - multiple ldapserver
  - …

Planned Additional Features (2)

Stateful Stubs

- Keep state on the server-side
  - Reduce communication
- Enable remote-object like operation

```
DefClass mvmul
Required "mvmul.o"
{
  DefState {
     double * tmpStorage;
  }
  Define init(IN int N, double  A[N][N]){
    tmpStorage = malloc(sizeof(double) * N * N);
    memcpy(tmpMat, A, sizeof(double) * N * N);
  }
  Define multiply(IN int N, IN double v_in[N],
                  OUT double vout[N])
  {
     mvmul(N, tmpMat, v_in, v_out);
  }
}
```

Planned Additional Features (3)

Stateful stubs ClientAPI

- Natural extension of the GridRPC API

```
double a[N][N];
double input_vectors[TIMES][N];
double output_vectors[TIMES][N];

grpc_handle_init_default(&handle,
                                "sample/mvmul");
grpc_invoke(&handle, "init", N, a);
for (int i = 0; i < TIMES; i++){
        grpc_invoke(&handle, "multiply", N,
                         input_vectors[i],
                         output_vectors[i]);
}
grpc_handle_finalize(&handle);
```

High-level API development

- **Ninf-G itself does not provide Scheduling, Fault Tolerance and Farming capability**
  - These Capability will be implemented on top of the primitive GridRPC
- **Scheduling:**
  - automatically choose suitable server
- **Fault Tolerance:**
  - detect error and re-submit the failed computation request
- **Farming:**
  - Support massive data-parallel applications

For More Info

- **Ninf home page**
  - http://ninf.apgrid.org
- **Contacts**
  - ninf@apgrid.org
- **Demonstration**
  - 5/14 Wednesday Afternoon
  - Grid Demo session
  - Weather forcast Demo

The End