Ninf-Gプログラムの開発と実行
- 中上級者編 -

産業技術総合研究所
グリッド研究センター
基盤ソフトチーム

武宮　博

Grid Technology Research Center AIST

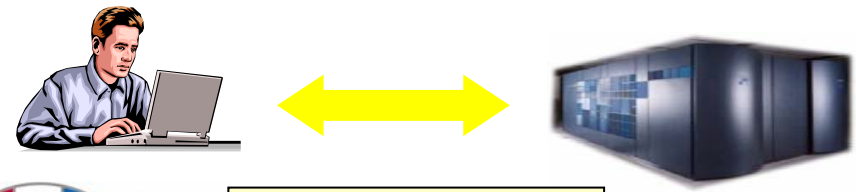
AIST

内容

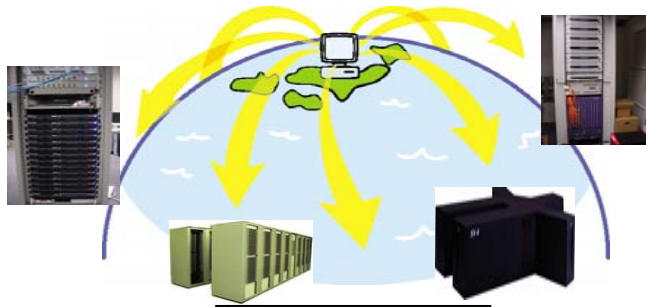
- **Ninf-G at a glance**
  - GridRPCとは何か？
  - Ninf-G2プログラムの開発と実行
- **Fortran Programの実行**
  - クライアントプログラムの作成
  - 遠隔実行プログラムの作成
- **MPI Programの実行**
  - クライアントプログラムの作成
  - 遠隔実行プログラムの作成
- **プログラムの柔軟性，頑健性，効率性の実現**
  - 効率化：関数ハンドルarray，dynamic scheduling
  - 柔軟化：time-out mechanism
  - 頑健化：fault detection mechanism

GridRPCとは何か？

- **GridRPC**
  - グリッドプログラミングモデルの一つ
    - グリッド環境上における遠隔手続き呼び出し(RPC)を支援
  - GGFにおいてGridRPC WGで標準仕様策定中
- **利用シナリオ**
  - 遠隔ライブラリ呼び出し
    - 遠隔に設置された高性能計算機資源での大規模タスクの実行
      - スパコン/クラスタ上での行列計算
      - 専用計算機を用いた重力計算
  - 大規模タスク並列処理
    - 分散計算機資源を用いた大規模タスク並列処理の実行
      - 分子立体配座探索の実行



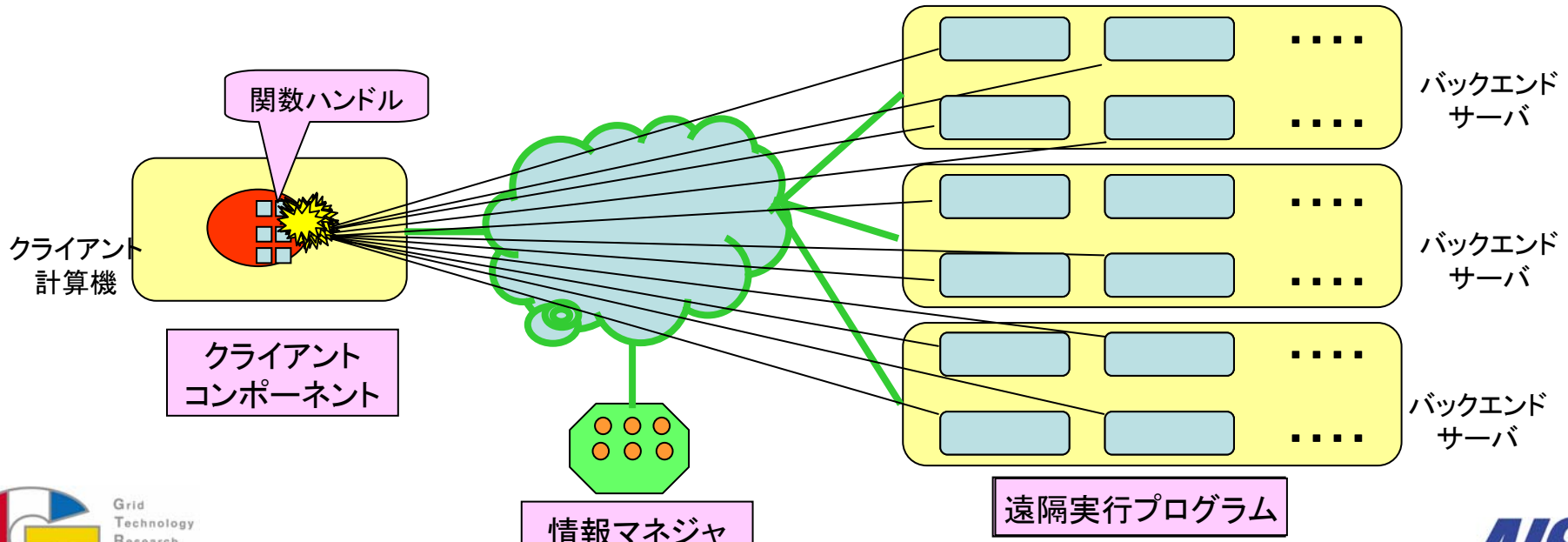
遠隔ライブラリ呼び出し



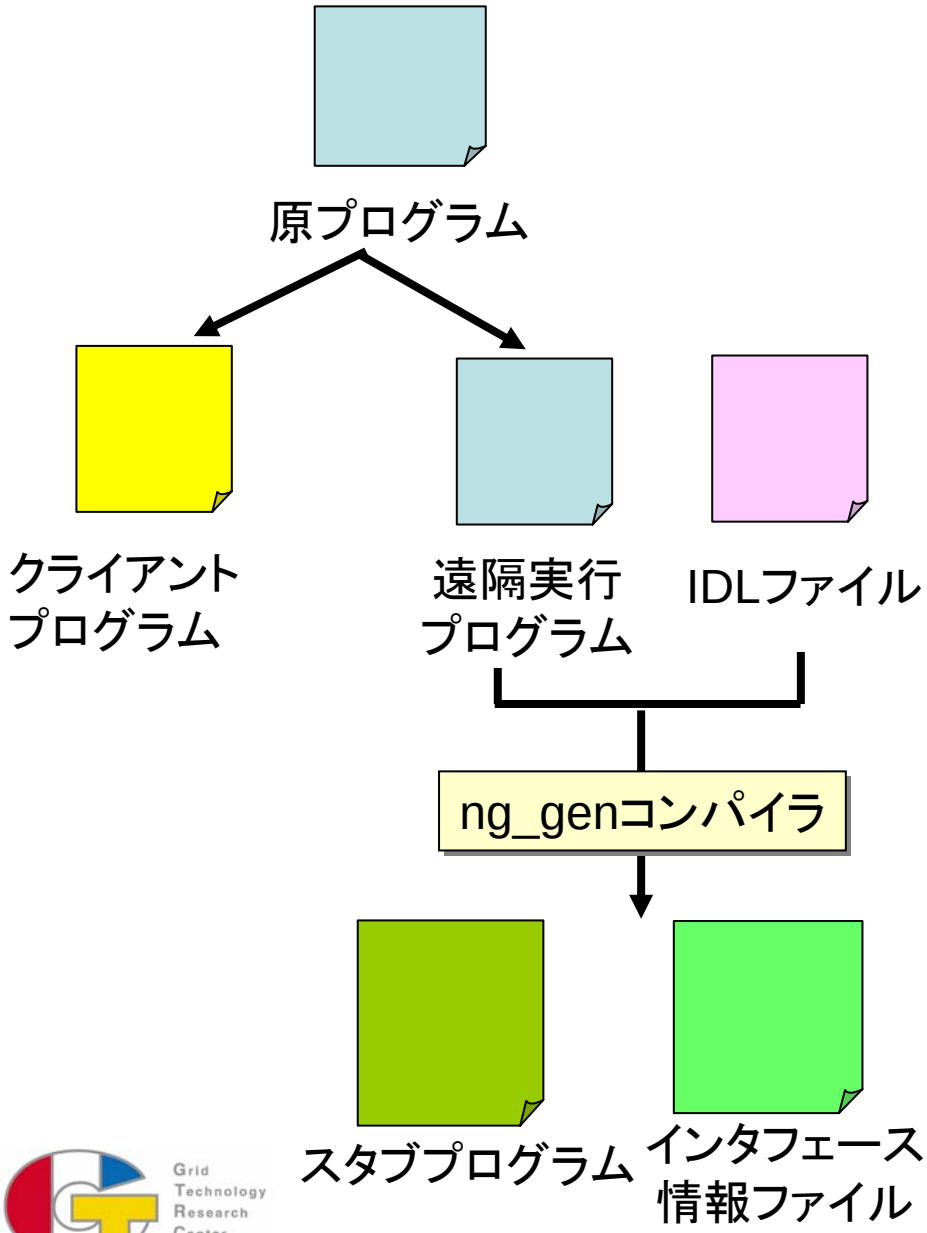
タスク並列処理

GridRPCモデル

- **クライアントコンポーネント**
  - GridRPCのcaller.
  - クライアント計算機上で関数ハンドルを用いてリモート実行プログラムを管理
- **遠隔実行プログラム**
  - GridRPCのcallee. バックエンド計算機上に動的に生成
- **情報マネジャ**
  - リモート実行プログラムのインタフェース情報を管理



Ninf-Gを用いたGrid化



クライアントプログラム

```
main {int argc, char** argv){
 grpc_initialize();
 grpc_function_handle_init
        (handle, server, "Param_survey")
 grpc_call(handle, in1, in2, size, result)
   :                                          }
```

遠隔実行プログラム

```
Param_survey
  (double in1, int in2, int size, double* result){
・・・
パラメータサーベイ処理
・・・
}
```

IDLファイル

```
Module Survey_prog;

Define Param_survey
  (IN double in1, IN int in2, IN int size,
                    OUT double result[size]);
Required "param_survey.o"
Calls "C" Param_survey(in1, in2, size, result);
```

Ninf-Gプログラムの実行

- **Configuration fileにより，プログラム実行に必要な情報をサーバ毎に規定**
  - XML-likeに属性名，値の組で記述
    - クライアント計算機情報：<client> ~ </client>
    - バックエンドサーバ情報: <server> ~ </server>
    - 情報マネジャ情報：<mds_server> ~ </mds_server>，<LOCAL_LDIF> ~ </LOCAL_LDIF>
  - grpc_initialize関数の引数としてファイル名を指定

```
<SERVER>
  hostname                  aaa.ninf.go.jp          host名
  jobmanager                jobmanager-pbs          gram jobamanger名
  job_startTimeout          30                      秒数
  job_stopTimeout           30                      秒数
  heartbeat                 60                      秒数
  heartbeat_timeoutCount    5                       回数
  protocol                  binary                  データ転送プロトコル
  compress                  zlib                    圧縮ルーチン名
  compress_threashold       64Kbyte                 圧縮対象サイズ
  argument_transfer         nowait                  データ転送モード
</SERVER>
<LOCAL_LDIF>
  filename                  aaa.ninf.go.jp.ngdef    ngdefファイル名
</LOCAL_LDIF>
```

Grid化における留意点

- **関数ハンドル vs. オブジェクトハンドル**
  - 関数ハンドル:
    - 単一呼び出しインタフェースをサポート
  - オブジェクトハンドル:
    - 複数呼び出しインタフェースをサポート
- **同期呼び出し vs. 非同期呼び出し**
  - 同期呼び出し:
    - クライアントの動作は遠隔実行プログラム実行終了までブロック
  - 非同期呼び出し:
    - クライアントの動作は遠隔実行プログラム実行終了を待たない

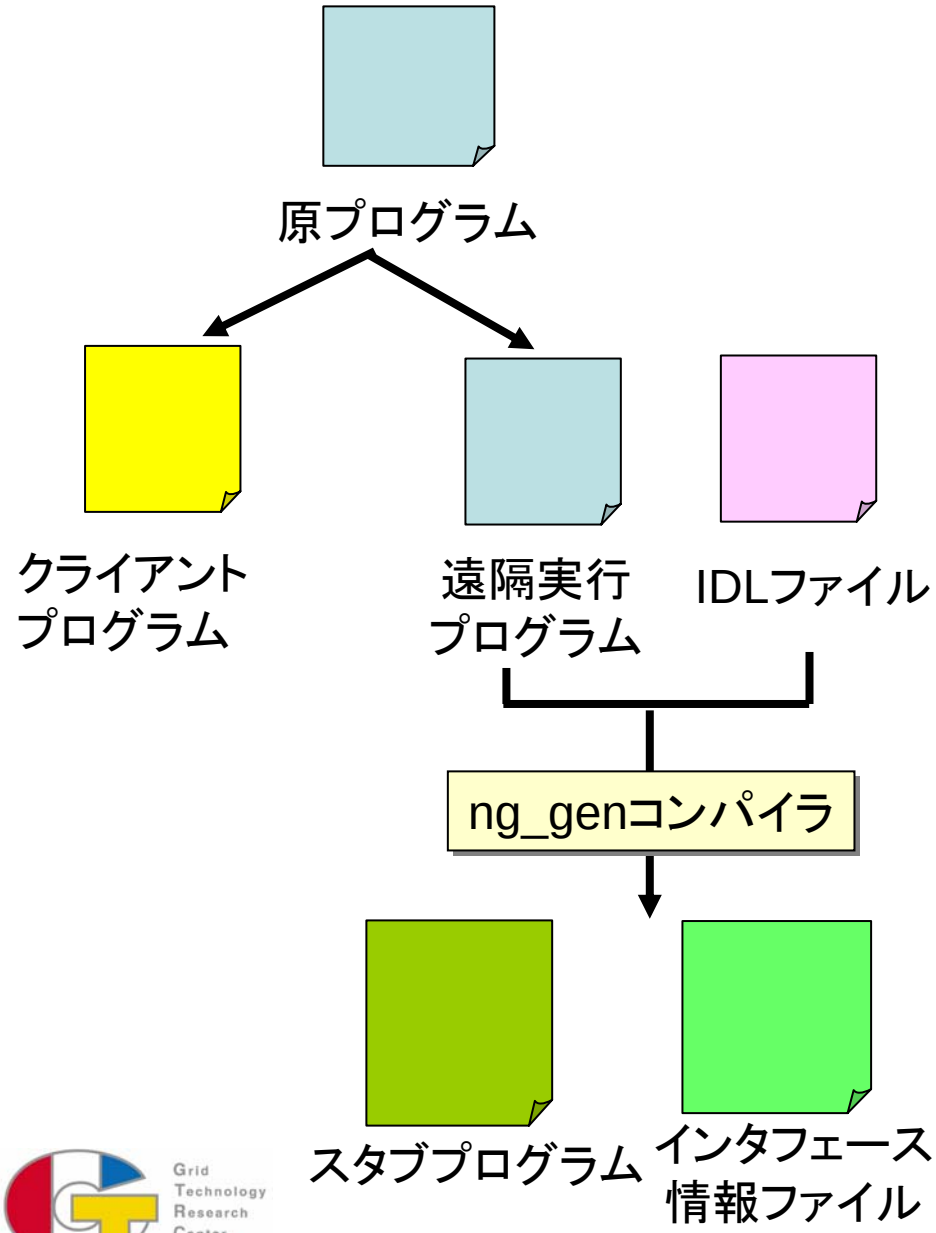
利用 IDL ファイル

	関数ハンドル	オブジェクトハンドル
	Module survey; Define Param_survey(IN double in1.. Required “survey.o” Calls “C” survey(in1, ….);	Module survey; DefClass survey_class Required “survey.o” { DefMethod meth1(IN double in1.. Calls “C” meth1(in1, ….); DefMethod meth2(IN int in2, …) Calls “C” meth1(in2, …);}

Advanced Topic

- **FortranプログラムのNinf化**
  - Server側作業
    - IDL fileの作成
  - Client側作業
    - Ninf-G関数のWrapper作成
- **MPIプログラムのNinf化**
  - Client/serverの切り分け
  - IDL fileの作成
- **Ninf化プログラムの効率化，柔軟化，頑健化**
  - 効率化：関数ハンドルarray，dynamic scheduling
  - 柔軟化：time-out mechanism
  - 頑健化：fault detection mechanism

Ninf-Gを用いたGrid化(Fortran版)



クライアントプログラム

```
program main
 my_grpc_initialize();
 my_grpc_function_handle_init
        (handle, server, 'Param_survey')
 my_grpc_call(han                    result)
    :
```

Wrapperの作成

遠隔実行プログラム

```
subroutine Param_survey
  (double in1, int in2, int size, double result)
・・・
パラメータサーベイ処理
・・・
```

IDLファイル

```
Module Survey_prog;
Define Param_survey
  (IN double in1, IN int in2, IN int size,
                  OUT double result[size]);
Required "param_survey.o"
Calls "Fortran"
Param_survey(i
FortranFormat "_%s"
```

Fortran用keywordの追加

Grid Technology Research Center AIST

Wrapperの作成

- **Ninf-GはC用のGridRPCインタフェースのみを提供**
  - FortranプログラムからGridRPCインタフェースを利用するためのwrapperが必要
- **Tips on wrapper creation**
  - クライアントプログラムは関数としてwrapper関数を呼ぶ
  - Naming conventionに注意
  - Fortranは全てアドレス渡し
  - characterはnull terminateされていない．サイズ情報が陰に渡される

```
Program main
integer ret, my_grpc_call
Integer in1, out1
      :
ret = my_grpc_call(in1, out1)
      :
```

クライアントプログラム(F)

```
grpc_function_handle_t* handle;

int my_grpc_call_(int* in1, int* out1)
{
   int ret;
   ret = grpc_call(handle, *in1, out1);
   return ret;
}
```

Wrapper(C)

Ninf-G
GridRPC
インタフェース

IDLファイルへのFortran用Keywordの挿入

2種類のKey wordを挿入

- Calls “Fortran” <サブルーチンインタフェース情報>
  - Ninf-Gがcallするfortran subroutineのインタフェース情報を記述する
- FortranFormat “name convention”
  - 利用compilerにおけるname convention ruleを記述する
  - Ex.
    - “_%s” : gcc, g77利用時

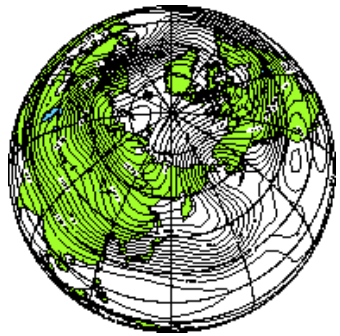
```
Module Survey_prog;
Define Param_survey
  (IN double in1, IN int in2, IN int size,
                    OUT double result[size]);
Required “param_survey.o”
Calls “Fortran”   Param_survey(in1, in2, size, result);
FortranFormat “_%s”
```

⇐ Fortran subroutine “Param_survey”を呼ぶことを宣言

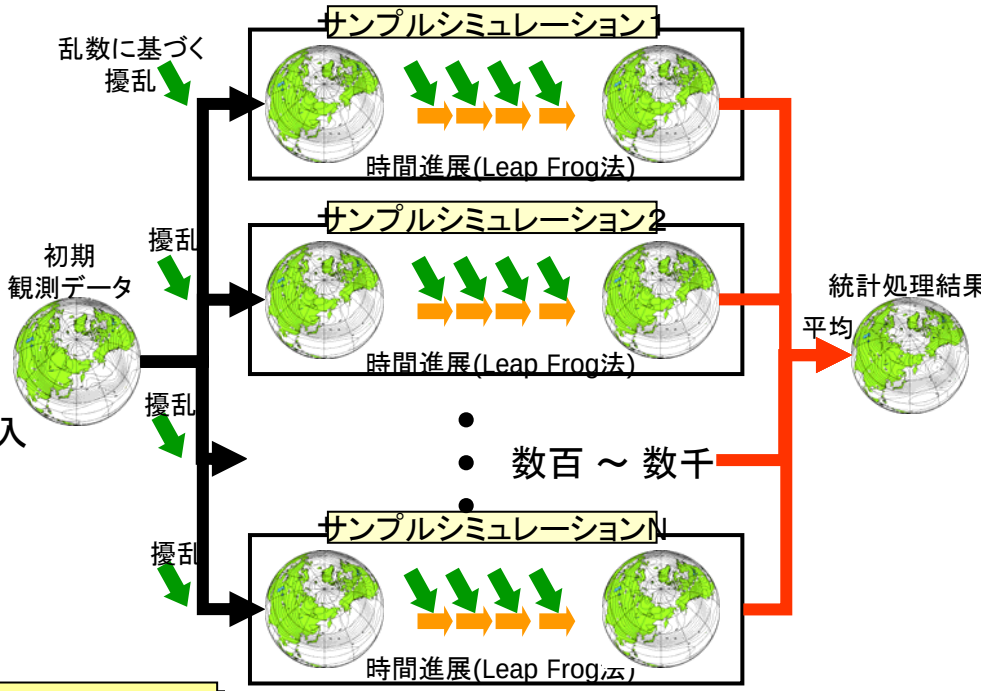
⇐ 関数名の前に’_’がつく事を宣言

アプリケーション事例 - 気象予報シミュレーションプログラム -

- 中長期間の気象変動を予測
- 順圧S-モデルに基づく
  - 筑波大学　田中助教授の考案
  - FORTRAN プログラム
- 簡易かつ精密
  - 大気の鉛直平均（順圧成分）の予測
    - カオス性の抑制に効果
    - 100日予測を150 秒程度で実行
  - 特徴的現象を精度良く再現
    - ジェットストリームの分布
    - 高気圧のブロッキング
- 長期予報への対応
  - アンサンブル予報
    - 各シミュレーションに乱数に基づく擾乱を導入
    - 数百~数千のサンプルシミュレーション実施
    - 典型的タスク並列プログラム



1989/1/30-2/12



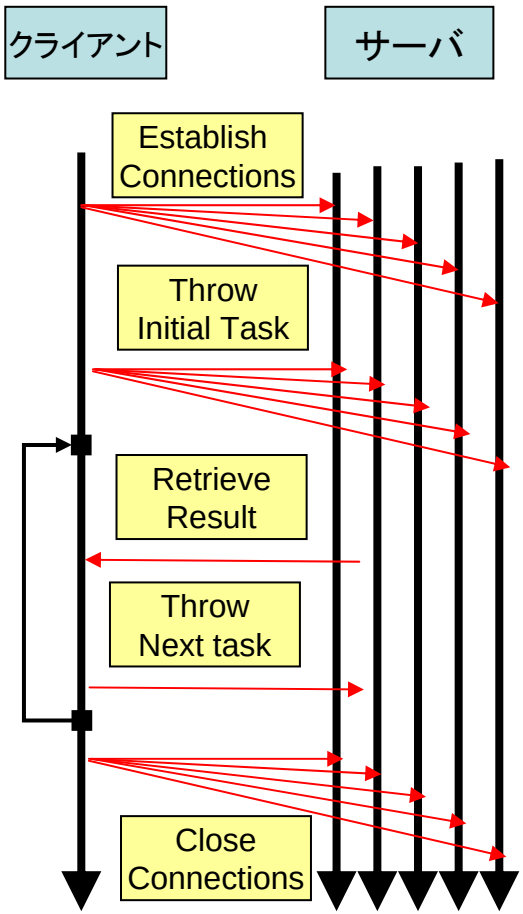
Grid化により高性能処理が期待

気象予報プログラムのGrid化（Grid環境上）

- **Ninf-G関数の挿入**
  - 非同期呼び出し(grpc_call_async)によるタスク並列処理を実現
- **スケジューリングルーチンの作成**
  - Pure self schedulingの採用
- **遠隔呼び出し関数用スタブの作成**
  - 下図IDLファイルより自動生成
- **LDIFファイルのMDSへの登録**
- **FORTRANプログラム用ラッパの作成**

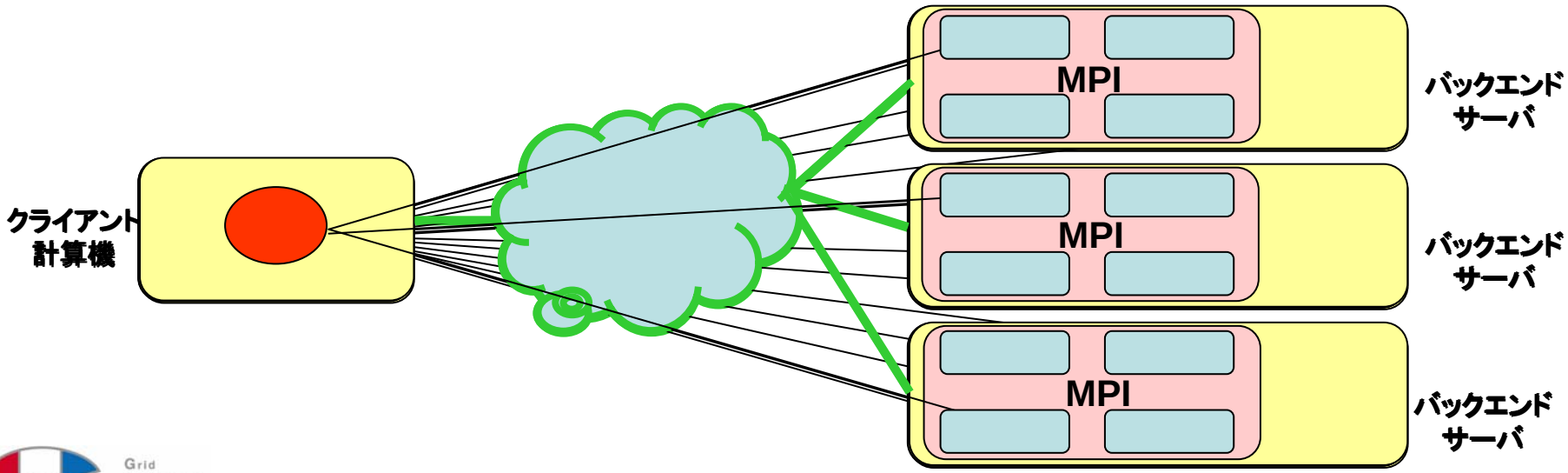
```
Module S-model;
Define servmain(IN int N1, IN int IBUF[N1],
                IN int N2, IN double DBUF[N2],
                IN int N3, IN double WW[N3],
                IN int N4, OUT double TARRAY[N4],
                IN int N5, OUT double WTOT[N5])
Required "S-model_serv.o"
Calls "F                                    BUF,
N3, WW
FortranFormat "%s_";
```

10個の引数
IN:8個　OUT:2個



MPIプログラムのNinf化

- 遠隔実行プログラムとしてMPIプログラムを利用
- 利用シナリオ
  - 長時間並列ジョブの実行
    - 長時間を要する並列ジョブを動的に計算機を切替えながら実行する
    - 動的にMPIプログラムを起動することにより計算機を使い分け
  - 大規模パラメータサーベイの実行
    - 並列実行される処理のパラメータサーベイ
    - 並列処理の並列実行



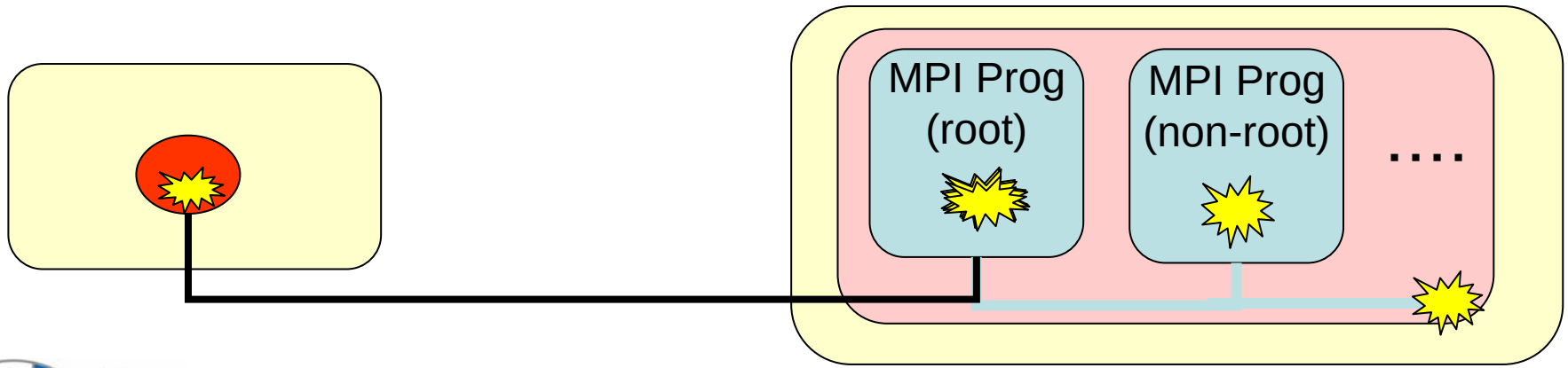
Ninf-Gを用いたMPIプログラムの実行の流れ

- 実行の流れ
  - MPIプログラムの起動，初期化
  - Root processとの通信路確立
  - 引数データの送信
  - 引数データの配布
  - 並列実行
  - 結果の収集
  - 計算結果の送信
  - MPIプログラムの終了

Ninf-Gが支援

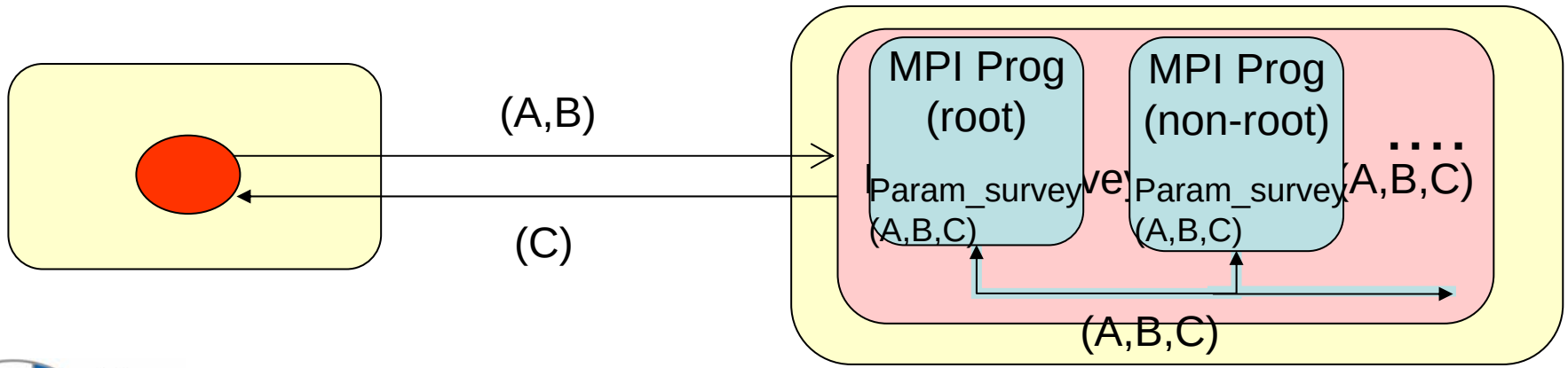
ユーザが記述

Ninf-Gが支援



MPIプログラムのNinf化指針

- 以下の3ステップでNinf化を行う
  - (1) 遠隔実行プログラムを逐次プログラムと考え，RPCのインタフェースを検討する
  - (2) 遠隔実行プログラムをMPIプログラムとして捉えなおしroot processと非root process間の入力データの配布，結果の収集処理を検討する
  - (3) 実際に遠隔実行プログラム内で行う処理を関数化する



既存プログラムの変形

- **既存MPIプログラムを3つの部分に分類，変形**
  - MPIプログラムの初期化，終了処理 ⇨ Ninf-Gが代行
  - 引数データの配布＋結果の収集処理 ⇨ Interfaceで実施
  - 実際の処理 ⇨ 実際の処理ルーチン内で実施

```
Int main(int argc, char** argv)
{
  MPI_initialize();
  if my_rank == 0 {
    Read_Input();}
  broadcast_vars();

  Param_survay();

  collect_result();
  MPI_finalize();
}
```

MPIプログラムの初期

データ入力処理

入力データ

実際の処理

結果データの収集

MPIプログラムの終了

```
Int main(int argc, char** argv){
  MPI_initialize();
   if my_rank == 0 {
    Read_Input();}
  Param_survey_interface();
  MPI_finalize();}
```

MPIプログラムの初期化

データ入力処理

MPIプログラムの終了

```
Param_survey_interface(){
  broadcast_vars();

  Param_survay();

  collect_result(); }
```

入力データの配布

実際の処理

結果データの収集

Grid Technology Research Center AIST

AIST

遠隔実行プログラムの作成

- 変形MPIプログラムからインタフェース関数を抽出
- 配布される入力データの記憶領域確保(非root process)

```
Param_survey_interface(int in1, int* int2, int* in3, int* out1){
  if (my_rank != 0){
    allocate_memory(in2, in3, out1);
  }
  broadcast_vars();

  Param_survay();

  collect_result(out1); }
```

記憶領域確保

入力データの配布

実際の処理

結果データの収集

IDLファイルの作成

MPIプログラム用Key wordを挿入

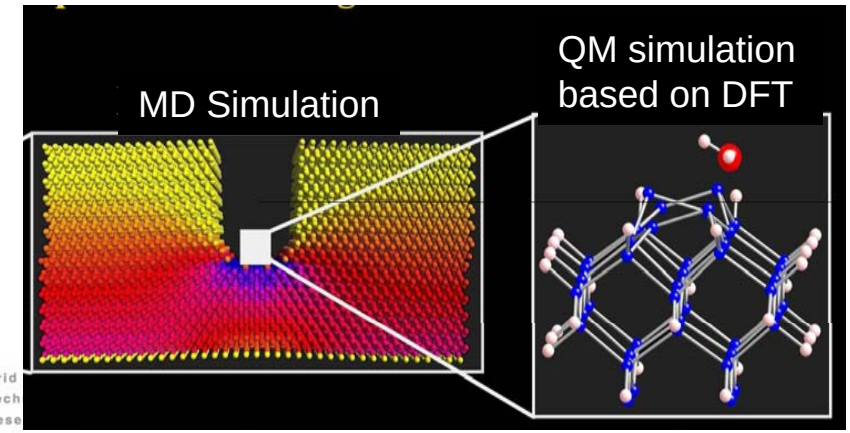
- Backend mpi
  - 遠隔実行プログラムがMPIであることを宣言
- Allocate, broadcase修飾子の利用
  - Allocate: 非root processにおける変数の記憶域を自動確保
    - Ex. IN allocate int in2[in1]
  - Broadcast:非root processにおける記憶域自動確保＋変数の自動配布
    - Ex. IN broadcase int in3[in1]

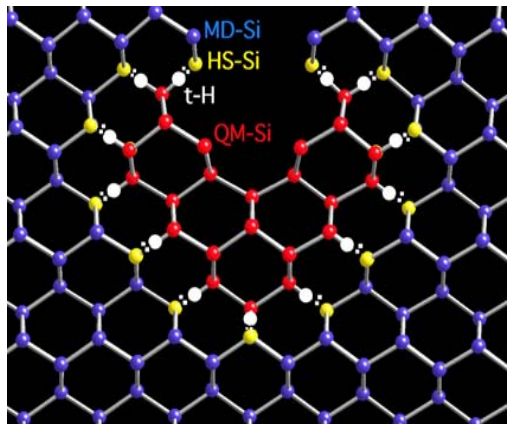
```
Module Survey_prog;
Backend mpi
Define Param_survey_interface
  (IN int in1, IN int in2[in1], IN int int3[in1],
                    OUT double result[in1]);
Required “param_survey.o”
Calls “Fortran”   Param_survey(in1, in2, size, result);
FortranFormat “_%s”
```

⇐ 遠隔実行プログラムがMPIであることを宣言

MPIプログラムNinf化事例 -Hybrid QM/MD Simulation-

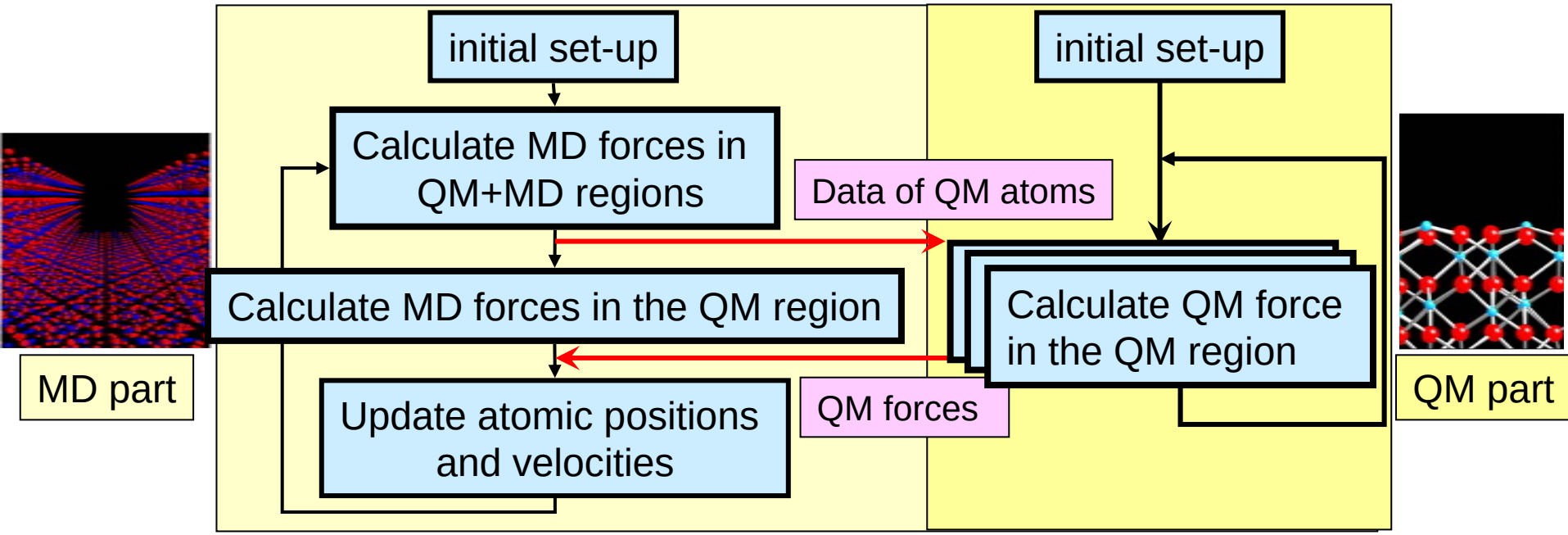
- **大規模なAtomistic simulationの高精度実行を可能に**
  - MD SimulationとQM simulationを連携
    - MD simulation
      - 全領域の原子の振舞いを計算
      - 経験的原子間ポテンシャルを用いた古典MDシミュレーション
    - QM simulation
      - 興味のある領域のみを対象に実行，MDの結果を修正
      - Density Functional Theory (DFT)に基づくQMシミュレーション
      - 複数のQM領域計算を並列に実行



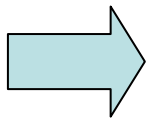


Hybrid QM/MD Simulation Algorithm

- シミュレーションアルゴリズム
  - Dr. Nakano (USC), Dr. Ogata (Nitech), et.alにより開発



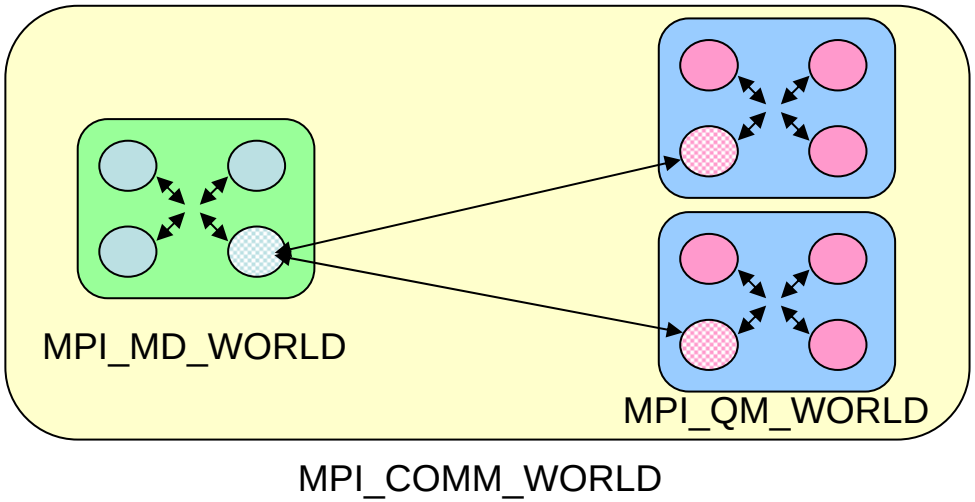
- 現プログラムはMPIを用いて実装
  - MPICH-G2や他のGrid aware MPIを用いればGrid上で無修正で実行可能
  - 幾つかの問題が存在
    - Co-allocation problem
    - Static configuration
    - Weak fault tolerance



Ninf-Gを用いた再実装

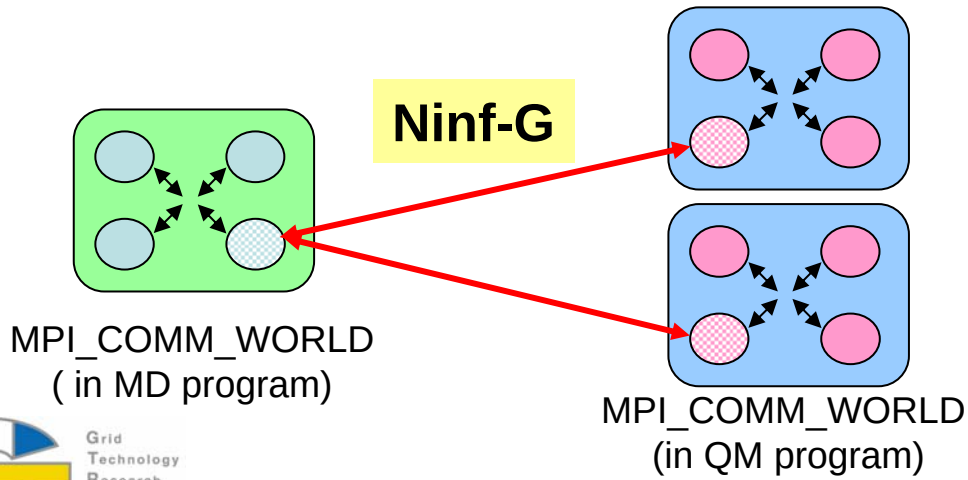
Ninf-Gを用いた原プログラムの修正：修正方針

原プログラム (MPI)



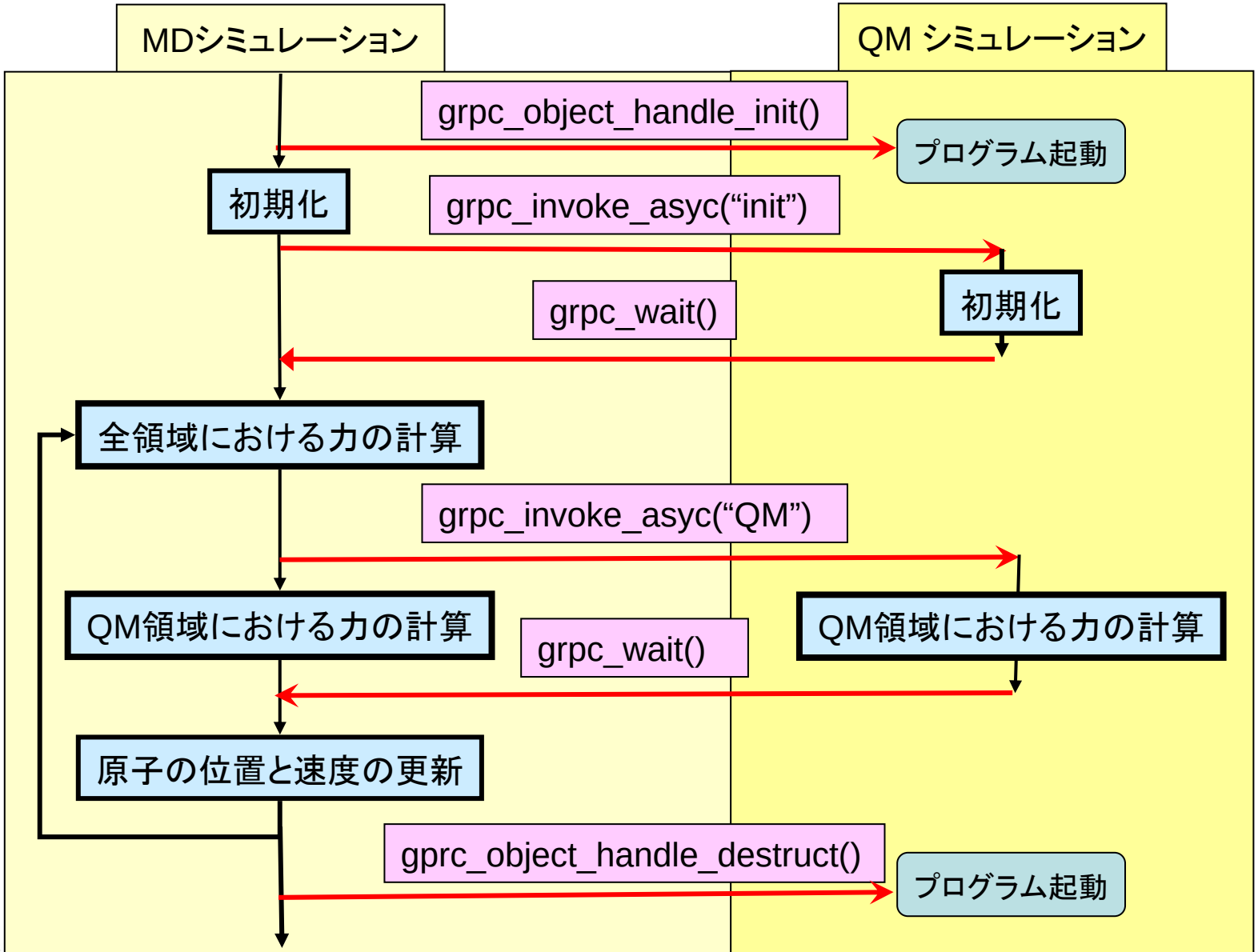
■単一プログラム
■シミュレーション内通信：Local communicatorを用いたMPI通信
■シミュレーション間通信：Global communicatorを用いたMPI通信

修正プログラム (GridRPC + MPI)



■2プログラム(MD + QM)
■シミュレーション内通信：Global communicatorを用いたMPI通信
■シミュレーション間通信：Ninf-Gを利用

QM/MDシミュレーションのNinf化



IDLファイルの記述(1)

- ヘッダ部，初期化手続き記述部，シミュレーション手続き記述部から構成

- ヘッダ部

```
Module QMMD;
Globals {int handle_id, nodes_qm_global;}

DefClass QM
Required "QM_serv.o QM_module.o fdlda_qm.o
ba_qm.o eigen_qm.o schmidt_qm.o fermi_qm.o
vxc_qm.o chgdns_qm.o poisson_qm.o funcs_qm.o
kbpp_qm.o vpp_qm.o force_qm.o cluster_qm.o
sbalin_qm.o datadump_qm.o mulliken_qm.o
mpi_comm_qm.o"
Backend "MPI"
```

リモートオブジェクトであることを宣言

遠隔実行プログラムがMPIであることを宣言

IDLファイルの記述(2)

初期化手続き記述部

```
{
DefMethod qm_init(IN int izeroQMCL, IN int iDFT, IN int npxi, IN int
npyi, IN int npzi, IN int cleanup_flag)
   {
   char buf[1024];
   int my_rank;
  int range[1][3];

// broadcasting variables from a root process to others
   MPI_Bcast(&izeroQMCL, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&iDFT, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&npxi, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&npyi, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&npzi, 1, MPI_INT, 0, MPI_COMM_WORLD);

//   execute fortran routines
   initialize_qm_(&izeroQMCL, &iDFT, &npxi, &npyi, &npzi);
```

初期化手続きの宣言

引数データの配送
(全てintなので記憶域確保の必要なし)

実際の初期化手続き(Fortran)の呼び出し

全ての引数はIN属性なので，結果を収集する必要なし

IDLファイルの記述(3)

シミュレーション手続き記述部

```
 DefMethod correctQMCL(IN double h[18], IN int Nqm1, IN int Nqm2, IN double Xqm1[3*Nqm1],
INOUT double Xqm2[3*Nqm2], IN int is_qm1_md[Nqm1], IN int is_qm2_md[Nq
ibondQM[5*Nqm2], OUT double* epotQM, OUT double dForceQM[3*(Nqm1+Nqm2)], OUT
filename logfname, OUT filename snapshot)
  {
     int my_rank;
     char buf[1024];

     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
// broadcasting some fundamental parameters first
       MPI_Bcast(&Nqm1, 1, MPI_INT, 0, mpi_qm_world);
       MPI_Bcast(&Nqm2, 1, MPI_INT, 0, mpi_qm_world);

// may need to allocate memory if my rank is not 0
       if(my_rank != 0){
         h = (double*)malloc(sizeof(double)*18);
         Xqm1 = (double*)malloc(sizeof(double)*3*Nqm1);
         Xqm2 = (double*)malloc(sizeof(double)*3*Nqm2);
         is_qm1_md = (int*)malloc(sizeof(int)*Nqm1);
         is_qm2_md = (int*)malloc(sizeof(int)*Nqm2);
         ibondQM = (int*)malloc(sizeof(int)*5*Nqm2);
         epotQM = (double*)malloc(sizeof(double));
         dForceQM =        (double*)malloc(sizeof(double)*3*(Nqm1+Nqm2));
       }
```

シミュレーション手続きの宣言

配列データのサイズ情報配送
（配列データの記憶域確保のため）

非root processにおける配列データの記憶域確保

IDLファイルの記述(4)

シミュレーション手続き記述部(2)

```
// broadcasting variables from a root process to others
        MPI_Bcast(h, 18, MPI_DOUBLE, 0, mpi_qm_world);
        MPI_Bcast(Xqm1, 3*Nqm1, MPI_DOUBLE, 0, mpi_qm_world);
        MPI_Bcast(Xqm2, 3*Nqm2, MPI_DOUBLE, 0, mpi_qm_world);
        MPI_Bcast(is_qm1_md, Nqm1, MPI_INT, 0, mpi_qm_world);
        MPI_Bcast(is_qm2_md, Nqm2, MPI_INT, 0, mpi_qm_world);
        MPI_Bcast(ibondQM, 5*Nqm2, MPI_INT, 0, mpi_qm_world);

// execute fortran routines
        qmclcorrect_(h, &Nqm1, &Nqm2, Xqm1, Xqm2,
                     is_qm1_md, is_qm2_md,
                     ibondQM, epotQM, dForceQM,
                     &mpi_qm_world);

// free memory allocated for arguments if my_rank != 0
        if(my_rank != 0){
          free(h);
          free(Xqm1);
          free(Xqm2);
          free(is_qm1_md);
          free(is_qm2_md);
          free(ibondQM);
          free(epotQM);
          free(dForceQM);
        }
}
```

配列データの配送

実際のシミュレーション手続き(Fortran)の呼び出し
(結果の収集はFortranプログラム内で実施)

非root processにおける記憶域解放

Broadcast, allocate修飾子の利用

- 宣言が大幅に単純化
  - IN, INOUT変数にbroadcast修飾子をつける
  - OUT変数にallocate修飾子をつける

```
Module QMMD;
Globals {int handle_id, nodes_qm_global;}

DefClass QM
Required "QM_serv.o QM_module.o fdlda_qm.o ba_qm.o eigen_qm.o schmidt_qm.o
fermi_qm.o vxc_qm.o chgdns_qm.o poisson_qm.o funcs_qm.o kbpp_qm.o vpp_qm.o
force_qm.o cluster_qm.o sbalin_qm.o datadump_qm.o mulliken_qm.o mpi_comm_qm.o"
Backend "MPI“
{
DefMethod qm_init(IN broadcast int izeroQMCL, IN broadcast int iDFT, IN broadcast
int npxi, IN broadcast int npyi, IN broadcast int npzi, IN int broadcast cleanup_flag)

DefMethod correctQMCL(IN broadcast double h[18], IN broadcast int Nqm1, IN
broadcast int Nqm2, IN broadcast double Xqm1[3*Nqm1], INOUT broadcast double
Xqm2[3*Nqm2], IN broadcast int is_qm1_md[Nqm1], IN broadcast int
is_qm2_md[Nqm2], IN broadcast int ibondQM[5*Nqm2], OUT allocate double*
epotQM, OUT allocate double dForceQM[3*(Nqm1+Nqm2)])
```

Ninf化プログラムの考慮点

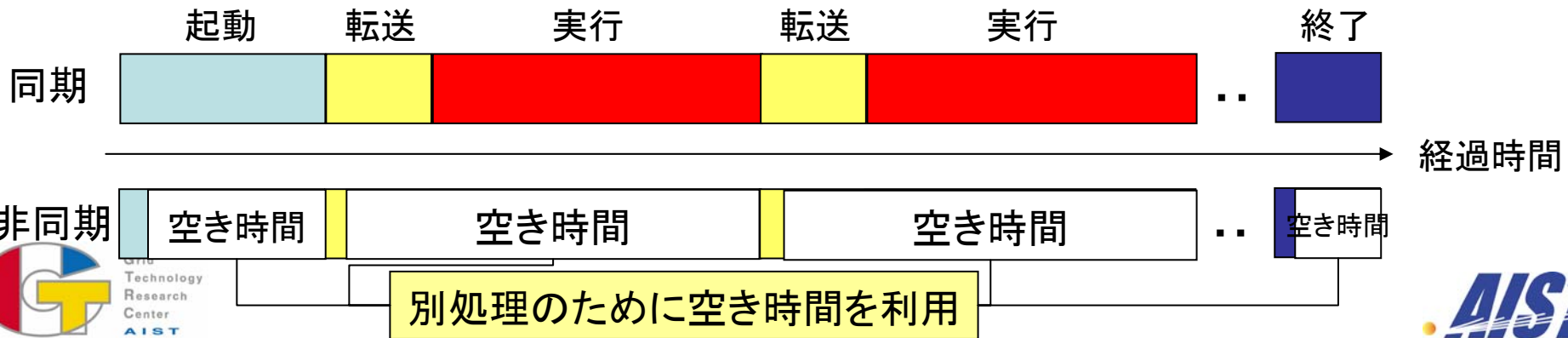
- **効率的な実行**
  - 遠隔実行プログラムを個々に起動 ⟹ 複数のプログラムを一括起動
    - 数秒～10数秒/遠隔実行プログラム => 数100～数1000秒/100遠隔実行プログラム
  - 関数ハンドルを固定された順番で利用 ⟹ 動的なスケジューリング
    - 非均質な計算資源では実行時間やデータ転送時間が異なる
    - 処理終了順に次のタスクを渡す
- **柔軟な実行**
  - 資源の共有可能性に対する考慮が必要 ⟹ timeout機能の利用
    - 計算機が他のユーザにより利用されていると実行開始に長時間待つ可能性
  - Resource awareであること ⟹ モニタリング機能(heart beat)の利用
    - ネットワーク,CPU負荷による通信，実行効率の低下
- **頑健な実行**
  - 単一計算機上での実行と比較してGrid上の実行は不安点 ⟹ Fault detection機能の利用
    - 長時間実行を想定した場合，RPCの実行失敗を想定したプログラミングが必要

Ninf-G2の実装機能

- **効率的な実行の支援**
  - 非同期起動/終了/通信/実行
  - リモートオブジェクト
  - 関数ハンドルの一括作成
  - バイナリデータ転送
  - MDSのバイパス
  - データ圧縮
  - 動的スケジューリング
- **柔軟な実行の支援**
  - 各種Time-out機能
  - ハートビートモニタリング
  - Ninf-G属性の動的変更
- **頑健な実行の支援**
  - Fault detection 機能

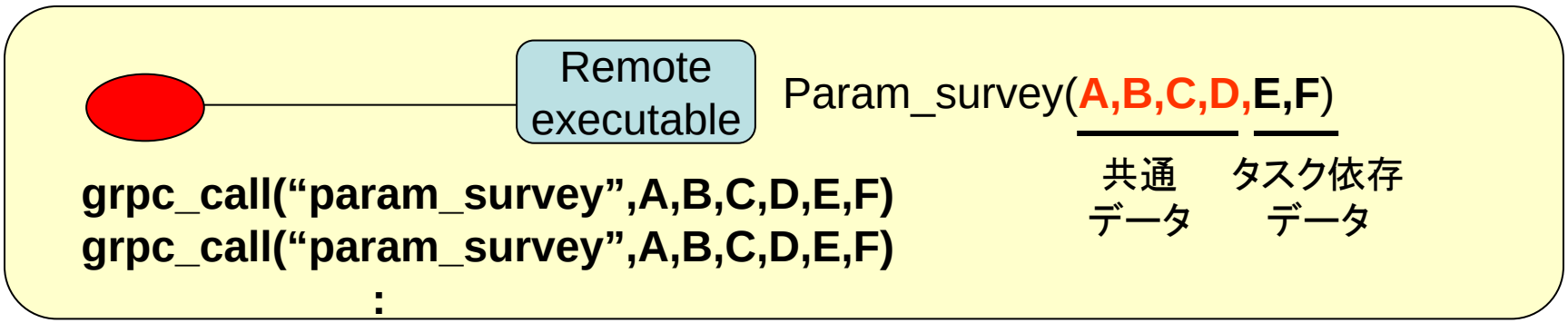
処理の効率化(1) –非同期mechanismの利用--

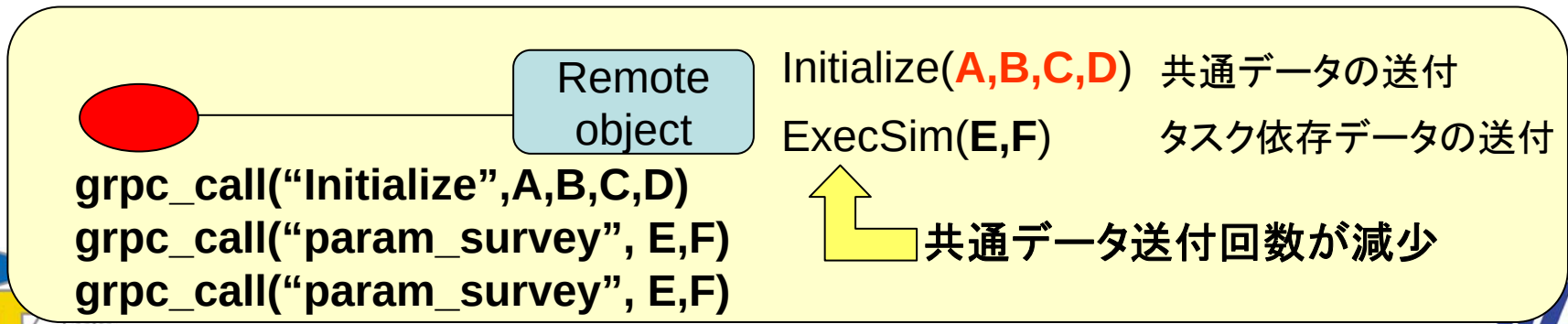
- 非同期mechanismの利用
  - 起動/終了処理
    - 遠隔実行プログラムが起動するまでに時間がかかる
    - 起動処理の非同期化はdefault action
  - 遠隔実行プログラム実行処理
    - 遠隔実行プログラムにおける処理に時間がかかる
    - Asynch callを利用
  - データ転送処理
    - クライアントプログラム←→遠隔実行プログラム間のデータ転送に時間がかかる
    - configuration fileにおいてargument_transfer属性をnowaitに設定



処理の効率化(2) – リモートオブジェクトの利用-

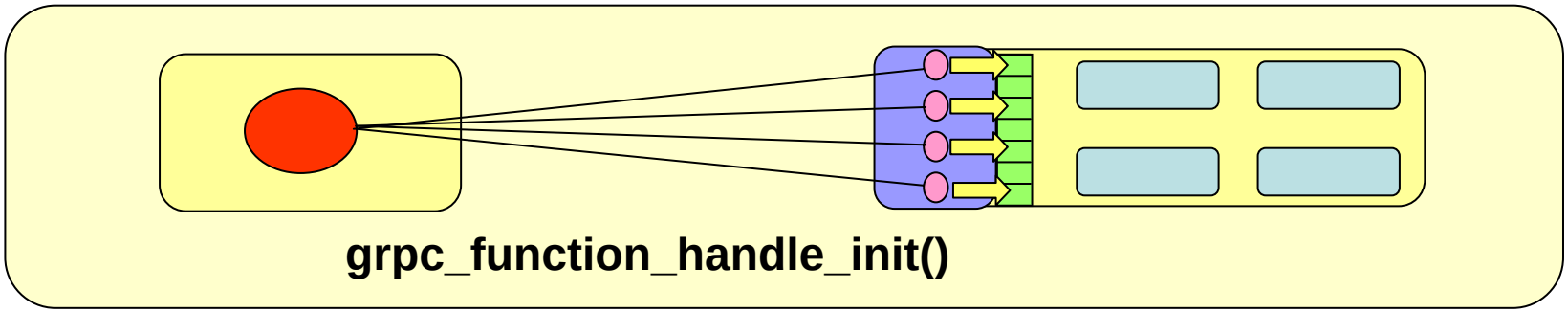
- **パラメータサーベイではタスクに共通なデータを送付することが多い**
- **パラメータサーベイにおける冗長データ転送を削減**
  - 遠隔実行プログラム
    - 単一インタフェースを持つ
    - タスクに共通なデータを毎回送付する必要あり
  - リモートオブジェクト
    - 複数インタフェースを持つ
    - 初期化用メソッドで共通データを送付
    - 別のメソッドでタスク依存データを送付

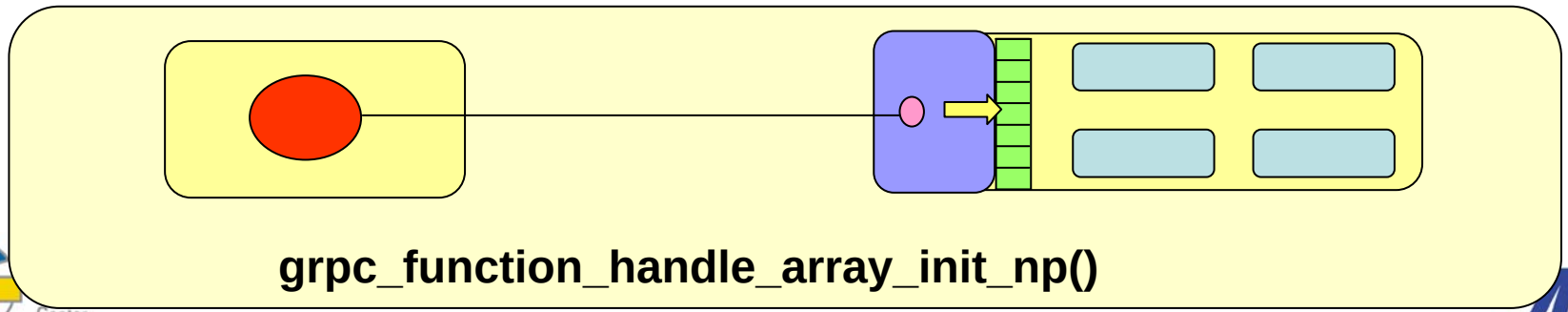




処理の効率化(3) – 関数ハンドル一括作成機能の利用-

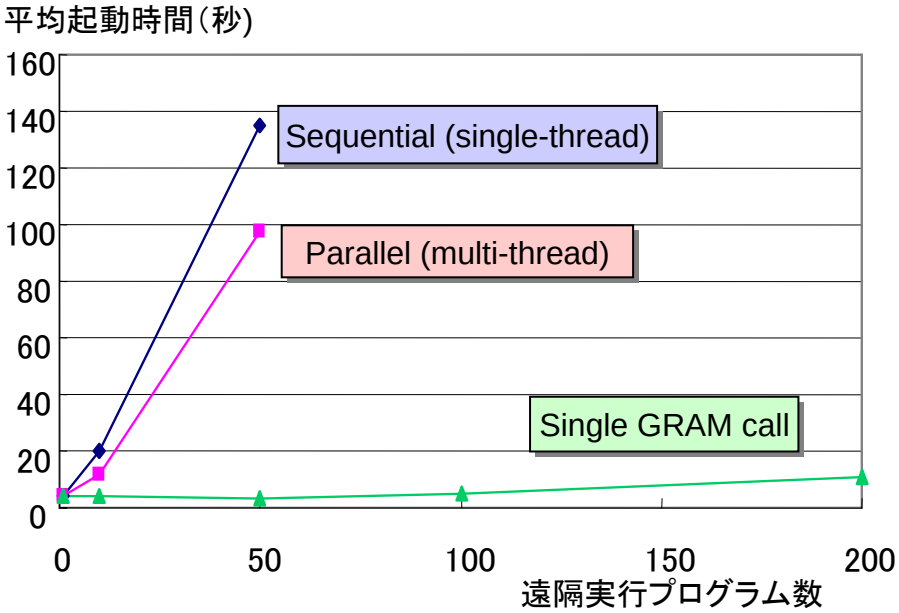
- **遠隔実行プログラム起動コストの削減**
  - grpc_function_handle_init
    - N個のjob managerが生成
    - N回の認証
    - N回の単一ジョブ実行リクエスト
  - grpc_function_handle_array_init_np()
    - 1個のjob managerが生成
    - 1回の認証
    - 1回のマルチジョブ実行リクエスト



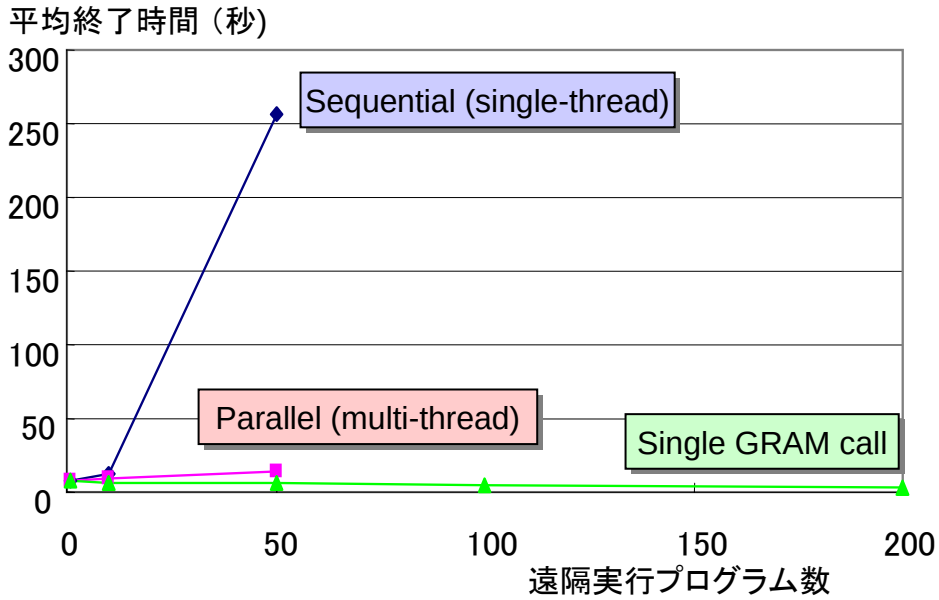


性能評価

- 起動・終了コストの評価
  - 利用クラスタ：TITECH
  - 3つのモードで計測
    - (1) 遠隔実行プログラムを逐次起動・終了 (single thread)
    - (2) 遠隔実行プログラムを並列起動・終了(multi threads)
    - (3) 遠隔実行プログラムを新規関数で起動・終了 (single GRAM call)
  - 新関数により起動・終了コストが大幅に削減
  - (1),(2)では100個以上の遠隔実行プログラムを安定に起動できず



起動コスト



終了コスト

処理の効率化(4) – 動的スケジューリング -

- 非均質，動的に変化する計算資源に対応
- Readyになった資源に対して動的にタスクを割付
  - grpc_wait_anyを用いる
- application logicに応じた種々のスケジューリングが可能

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

関数ハンドルを順に利用

全タスクの終了待ち

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

全ハンドルへのタスク割付

```
For(I = n_handles, I < n; i++){
  grpc_wait_any();
```

最速処理タスクの終了待ち

```
   grpc_call_async
        (handles, in1,in2,100, result+100*n)
}
```

そのハンドルへタスク再割付

```
grpc_wait_all();
```

最終的なタスク回収

処理の効率化(5) -- miscellaneous --

- バイナリデータ転送
  - Binary compatible machine間のデータ転送をraw dataで行う
  - Configuration fileにおいてprotocol属性をbinaryに設定
- MDSのバイパス
  - 遠隔実行プログラムのインタフェース情報をローカルファイルから取得
  - Configuration fileにおいてLOCAL_LDIF clause内のfilename属性にngdefファイル名を設定
  - Ngdefファイルはng_gen実行時に自動生成される
- データ圧縮
  - 転送データを圧縮し，データサイズを削減
  - Configuration fileにおいてcompress, compress_threshold属性を設定

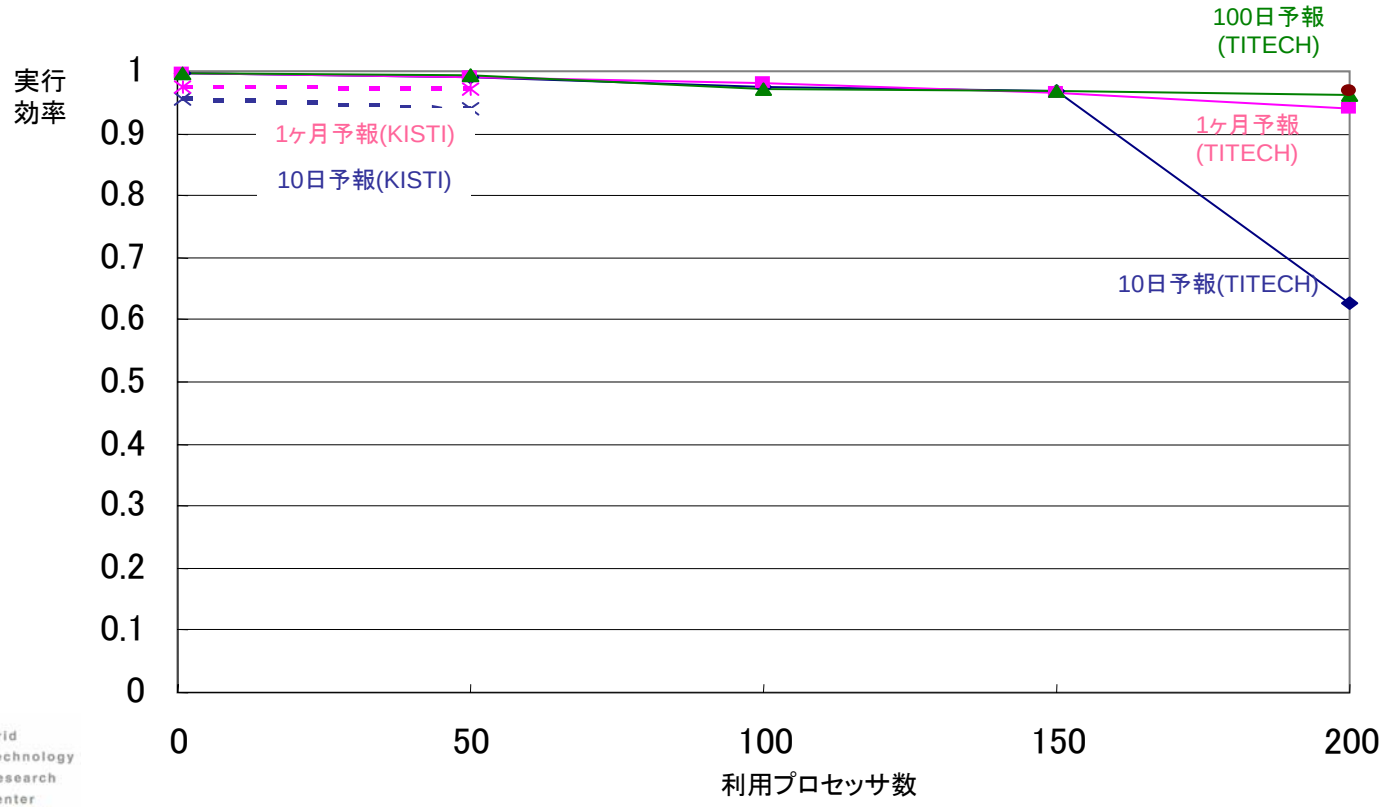
処理の効率化のためのConfiguration fileの設定

- 非同期データ転送 （argument_transfer属性）
- バイナリデータ転送 (protocol属性)
- MDSのバイパス (LOCAL_LDIF clause)
- データ圧縮 (compress, compress_threshold属性)

```
<SERVER>
  hostname                  aaa.ninf.go.jp          host名
  jobmanager                jobmanager-pbs          gram jobamanger名
  job_startTimeout          30                      秒数
  job_stopTimeout           30                      秒数
  heartbeat                 60                      秒数
  heartbeat_timeoutCount    5                       回数
  protocol                  binary                  データ転送プロトコル
  compress                  zlib                    圧縮ルーチン名
  compress_threashold       64Kbyte                 圧縮対象サイズ
  argument_transfer         nowait                  データ転送モード
</SERVER>
<LOCAL_LDIF>
  filename                  aaa.ninf.go.jp.ngdef    ngdefファイル名
</LOCAL_LDIF>
```

実行性能評価（単一クラスタ利用）

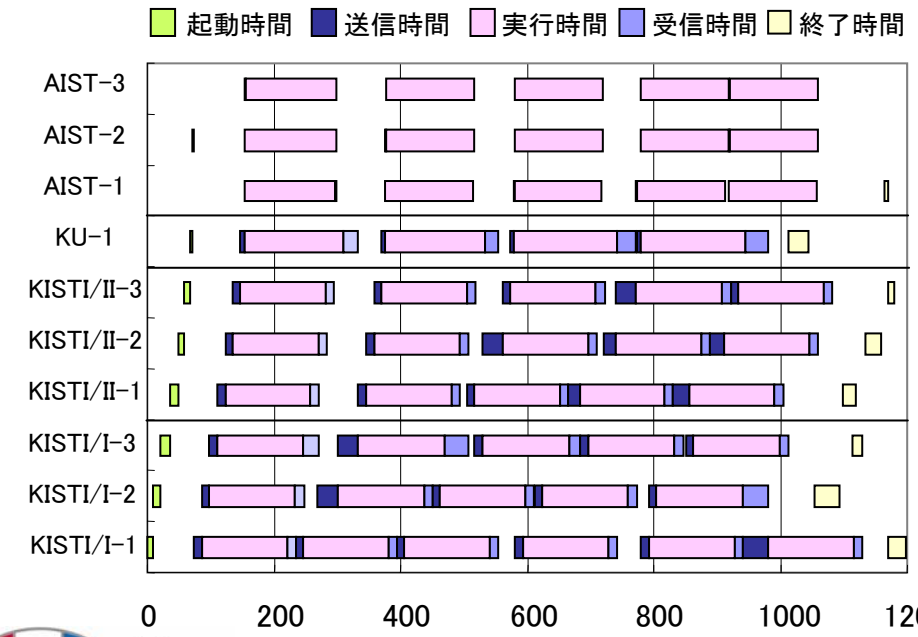
- **単一クラスタ上で気象予報シミュレーションを実行**
  - 利用クラスタ：TITECH, KISTI
  - 予報期間：10日，1ヶ月，100日
  - 計算時間：約12，35，120秒/(1サンプルシミュレーション)
  - 引数データサイズ：~3.4KB
  - 結果データサイズ：~140KB, 400KB, 1.35MB
  - サンプルシミュレーション数：利用ノード数×５



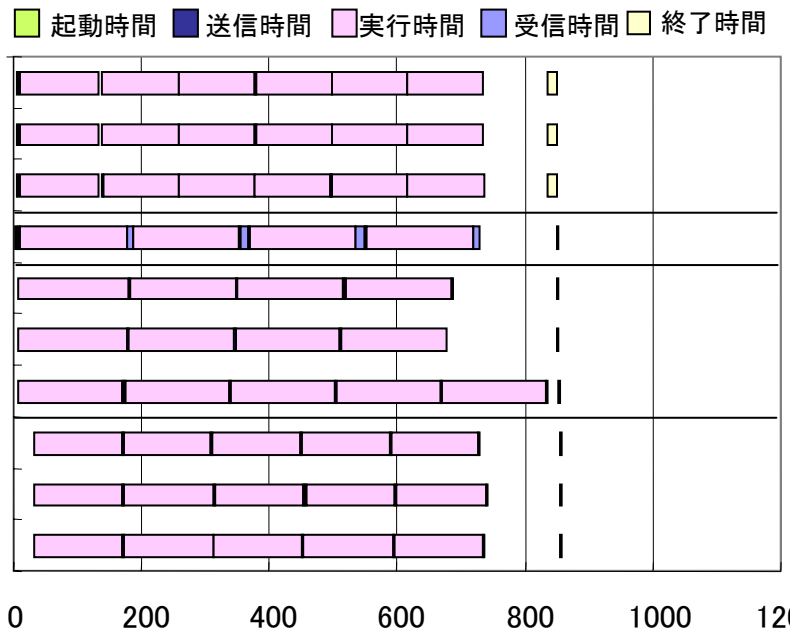
実行性能評価(複数クラスタ利用)

- **4クラスタ(3サイト)で気象シミュレーションを実行**
  - 利用クラスタ:AIST, KISTI×2, KU
  - 予報期間:100日間
  - サンプルシミュレーション数:50
- **実行時間が1200(秒)から850(秒)に減少**
  - 起動・終了コストの減少
  - 通信コストの減少
    - サーバのアイドル時間も減少

ハンドル一括作成機能の利用
Binary data転送機能の利用



Ninf-G



Ninf-G2

Grid Technology Research Center AIST

柔軟な実行の実現 –time out機能の利用 -

- **各種タイムアウト機能により長時間待ちを回避**
  - 起動・終了タイムアウト
    - job_startTimeout, job_stopTimeout
    - 遠隔実行プログラムの実行開始・終了にかかる時間を制限
      - 他のユーザのジョブ実行による遠隔実行プログラム実行待ち
    - 関数ハンドル作成・破棄要求後に一定時間経過するとジョブをキャンセルし，関数ハンドル作成，破棄失敗を通知
  - 実行タイムアウト
    - job_maxTime, job_maxCpuTime, job_maxWallTime, session_timeout
    - 遠隔実行プログラムの実行時間を制限
  - ハートビートタイムアウト
    - heartbeat, heatbeat_timeoutCount
    - 目的：遠隔実行プログラムの実行状態チェック
      - ネットワーク，計算資源の負荷増大により遠隔実行プログラムが終了しない
    - 遠隔実行プログラムから一定間隔でハートビートを送信
      - ハートビート間隔はコンフィグレーションファイルに設定
    - 指定時間以上ハートビートが検知されない場合，タスクの実行失敗を通知

柔軟な実行の実現　Ninf-G属性の動的変更

- **Configuration fileの再読み込み**
  - 実行途中におけるサーバの動的追加/削除
  - grpc_config_file_read_np()関数の利用
    - configuration file は通常grpc_initialize()実行時にのみ読み込まれる.
    - grpc_config_file_read_np()を用いることにより，動的に再読み込み可能
    - 読み込みタイミングに注意必要
      - 実行中のサーバを削除した場合の動作は保証されない.
- **Handle attributeの動的設定**
  - configuration fileに設定された属性の動的変更
    - Ex. MPIプログラムの並列実行プロセッサ数を動的に変更
  - grpc_handle_attr_set_np()関数の利用
    - gprc_handle生成時に実行

属性動的変更例

MPIプロセス数の動的変更

- Configuration fileに設定された属性をoverlay
  - Host名と関数名をキーとして属性値を動的に変更
  - 変更しない属性は設定する必要なし

```
Int CreateHandle(grpc_object_handle_t_np* handle){
 grpc_handle_attr_t_np attr;
 /* initialize handle attr */
   result = grpc_handle_attr_initialize_np(&attr);

   result = grpc_handle_attr_set_np
            (&attr, GRPC_HANDLE_ATTR_HOSTNAME, h_info->host_addr);

   result = grpc_handle_attr_set_np
            (&attr, GRPC_HANDLE_ATTR_FUNCNAME, t_info->func_name);

   result = grpc_handle_attr_set_np
            (&attr, GRPC_HANDLE_ATTR_MPI_NCPUS, &(h_info->n_cpu_cu

    /* Initialize Function handles */
   result = grpc_object_handle_init_with_attr_np(handle, &attr);
   return GRPC_NO_ERROR;}
```

ハンドル属性構造体の宣言

ハンドル属性構造体の初期化

ハンドル属性の設定 (hostname属性)

ハンドル属性の設定 (関数名属性)

ハンドル属性の設定 (関数名属性)

ハンドルの生成

Configuration fileにおけるタイムアウトの設定

- 起動/終了タイムアウト (job_startTimeout, job_stopTimeout属性)
- 実行時間タイムアウト
  (job_maxTime, job_maxCpuTime, job_maxWallTime, session_timeout属性)
- ハートビートタイムアウト (heartbeat, heartbeat_timeoutCount属性)

```
<SERVER>
  hostname                  aaa.ninf.go.jp          host名
  jobmanager                jobmanager-pbs          gram jobamanger名
  job_startTimeout          30                      秒数
  job_stopTimeout           30                      秒数
  job_maxTime               3000                    秒数
  job_maxCpuTime            3000                    秒数
  job_maxWallTime           5000                    秒数
  heartbeat                 60                      秒数
  heartbeat_timeoutCount    5                       回数
</SERVER>
<FUNCTION_INFO>
  hostname                  aaa.ninf.go.jp          host名
  funcname                  Param_survey            関数名
  path                      /home/aaa/bbb/ccc       path
  session_timeout           300                     秒数
</FUNCTION_INFO>
```

頑健な実行の実現 -fault detection機能-

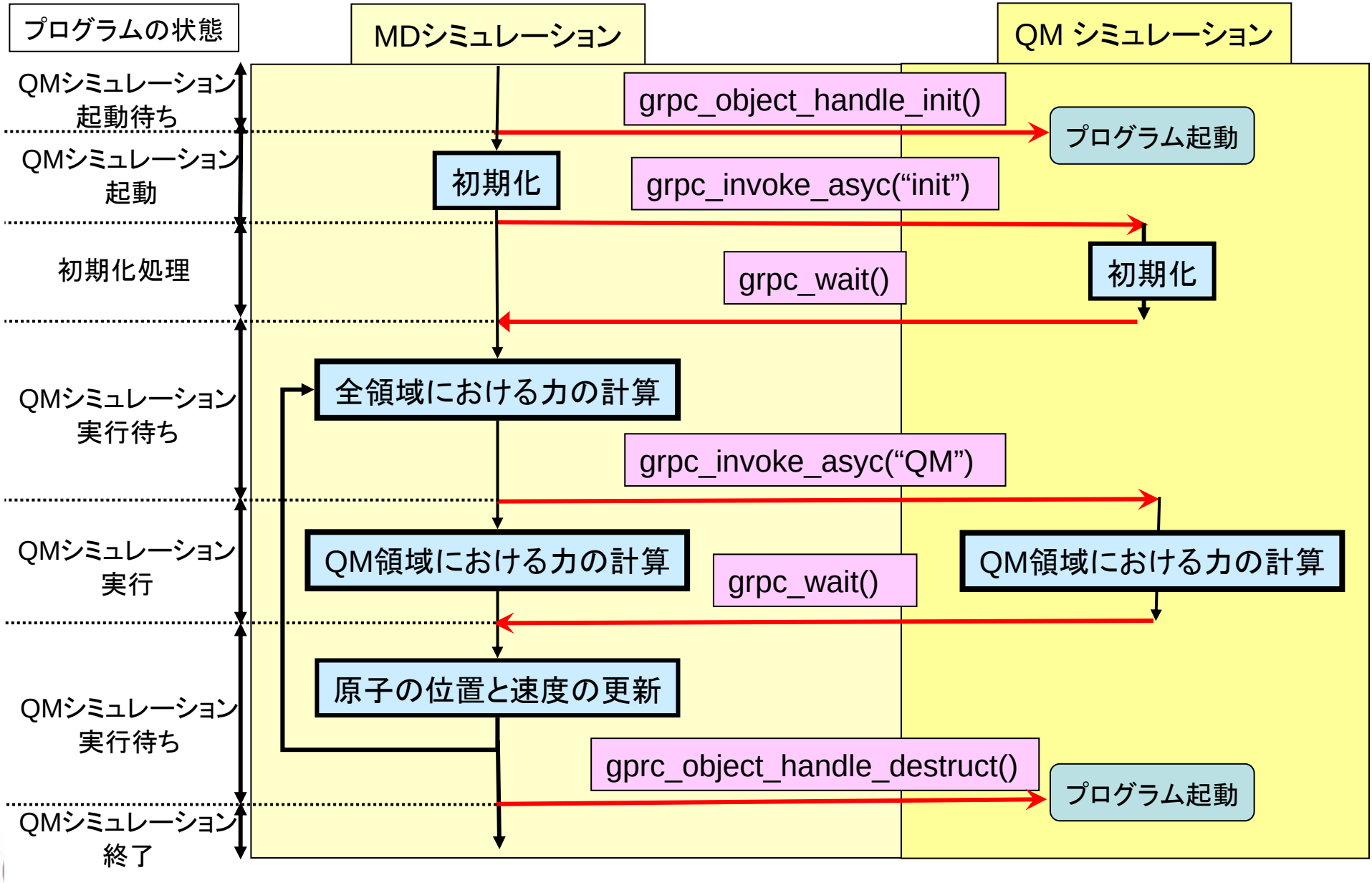
- GridRPCの各関数は実行の成功，失敗を返す
  - 成功時：GRPC_NOERROR
  - 失敗時：各種error値
- 失敗時にはretryすることにより，プログラムの頑健性を高められる
- 経験上，同一マシンにretryしても成功率は低い．別のマシンを利用したほうが成功率が高い

```
                :
ret = grpc_call_async():
If (ret != GRPC_NOERROR){
  decide_new_target_machine();
  grpc_call_async();
}
ret = grpc_wait();
If (ret != GRPC_NOERROR){
  retry_grpc_call();
}
                :
```

⇐ 成功の確認

⇐ 成功の確認

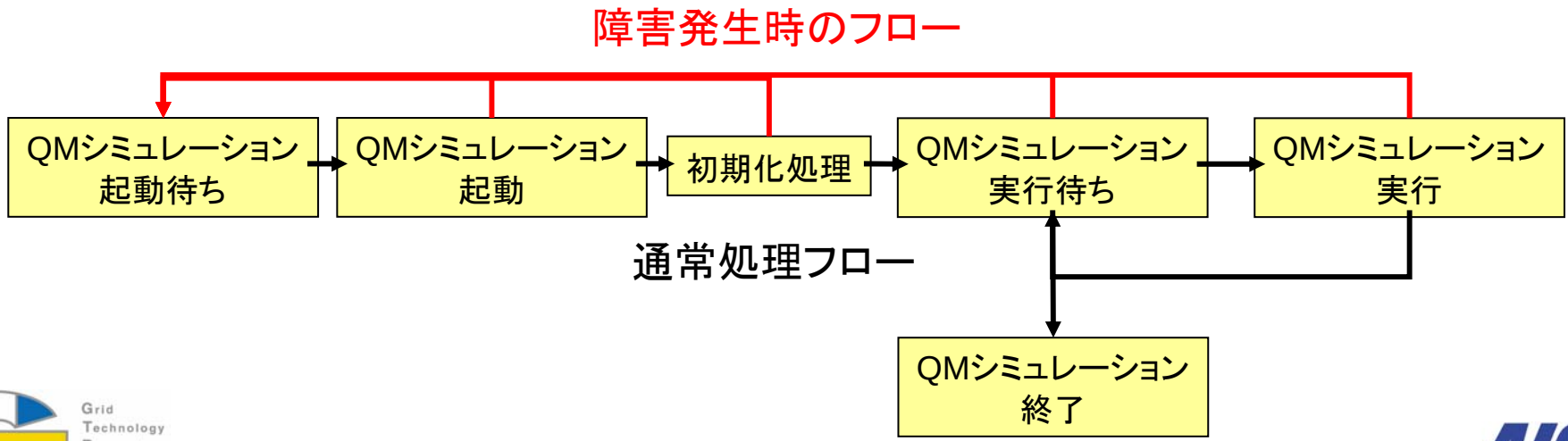
障害復旧機能実装例 – Hybrid QM/MD-



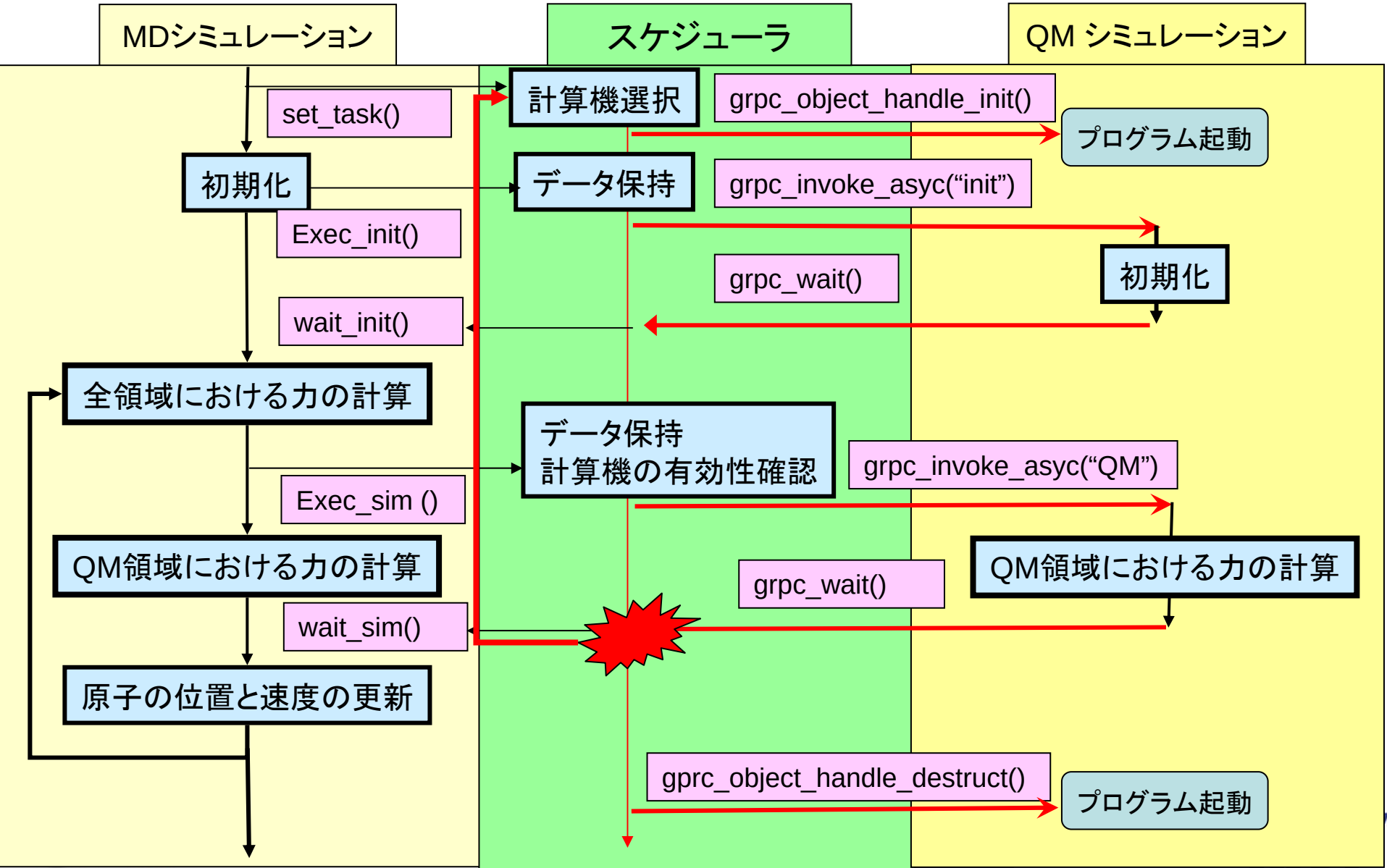
障害復旧機能実装例

障害復旧手順の実装

- 遠隔実行プログラムの状態を6種類に分類
- 各状態の遷移を規定
  - 障害発生時は常にQMシミュレーション起動待ち状態に遷移
- アプリケーションプログラムの変更はしない
  - アプリケーションレイヤとNinf-Gレイヤの間にスケジューリングレイヤを実装
  - スケジューリングレイヤに障害復旧手順を実装



スケジューリングレイヤの動作



計算機の選択

- 現在有効な計算機群の中から必要とされるノード数を持つものを選択
  - 計算機情報：ファイルにユーザが設定
  - ノード数：set_task()の引数で指定

```
<HOST>
 NAME F32-1                 :host名
 ID   110                   :hostID
 ADDR hpc.usc.edu           :host address
 FROM 2005/10/7/9/0/0       :有効期間（開始）
 TO   2005/11/8/23/30/0     :有効期間（終了）
 MAX_AVAIL 604800           :最大継続時間
 CPU_MAX 140                :最大利用可能CPU数
 CPU_INIT 128               :初期起動時CPU数
</HOST>
```

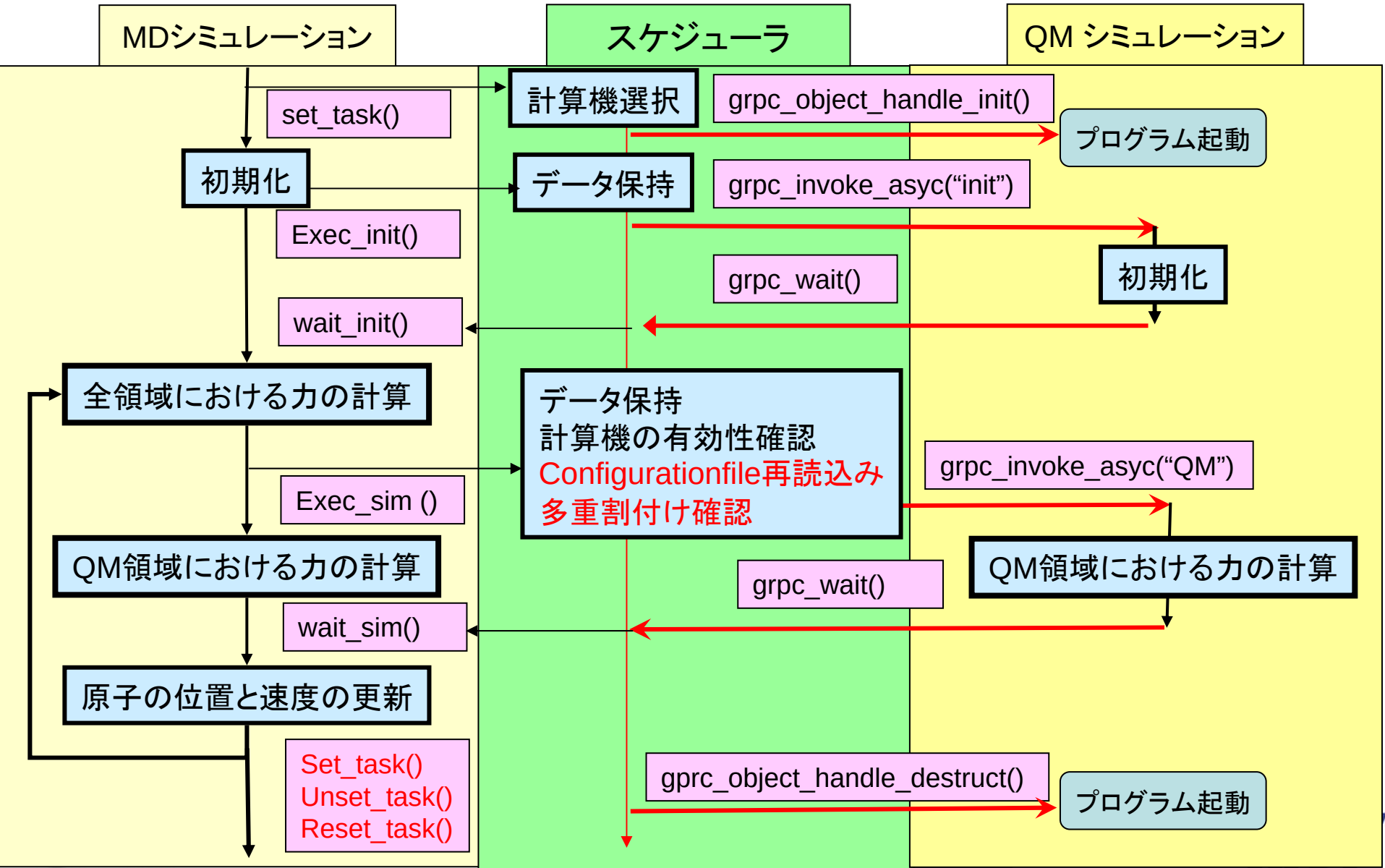
- 複数候補が存在する場合は，過去の実行履歴から失敗の少ないものを選択

柔軟性の実現例

- 各種タイムアウト機能を利用
- Hybrid QM/MDではQM原子数が動的に変化
  - 計算量が変化 ⇒ CPU数を調整する必要
  - Ninf属性変更機能を利用
- Hybrid QM/MDではQM領域数が動的に変化
  - Handleの動的な作成，破壊機能を利用
- 長期実行の際には，当初想定していなかった新たな計算機が利用可能になることもある
  - Configuration fileの動的読み込み機能の利用
- 利用可能計算機数がQM領域数以下になることもある
  - 単一クラスタに複数領域の計算を割り付け

最終的な処理の流れ

- アプリケーションはほとんど変更する必要なし



まとめ

- **Fortran Programの実行**
  - クライアントプログラムの作成 wrapperプログラムを作成
  - 遠隔実行プログラムの作成 Fortran用Key wordの利用
- **MPI Programの実行**
  - クライアントプログラムの作成 wrapperプログラムを作成
  - 遠隔実行プログラムの作成 MPI用Key word + データの集配
- **プログラムの柔軟性，頑健性，効率性の実現**
  - 効率的な実行
    - リモートオブジェクト
    - 関数ハンドルの一括作成
    - バイナリデータ転送
    - MDSのバイパス
    - データ圧縮
    - 動的スケジューリング
  - 柔軟な実行
    - 各種Time-out機能
    - ハートビートモニタリング
    - Ninf-G属性の動的変更
  - 頑健な実行の支援
    - Fault detection 機能

Ninf-Gの現状

- ウェブサイトにて最新版(Ninf-G2.4.0, Ninf-G4)を公開中
  - http://ninf.apgrid.org
- プラットフォーム
  - SPARC/Solaris
  - IA32/Linux (RedHat, Debian)
  - IA64/Linux (RedHat)
  - R10000/IRIX
- 前提ソフトウェア
  - Globus Toolkit Ver.2.2/Ver3 Pre-WSコンポーネント/Ver.4
- 詳細情報
  - ユーザーズマニュアル:
    - http://ninf.apgrid.org/document/ng2-manual/user-manual.html
  - 実装情報詳細:
    - 田中, 中田, 平野, 佐藤, 関口:"GlobusによるGridRPCシステムの実装と評価, HPC研究会, Vol.2001, No.77, pp.165-170
    - 武宮, 田中, 中田, 関口:"Ninf-G2:大規模Grid環境での利用に即した高機能, 高性能GridRPCシステムの実装と評価, SACSIS2004論文集
  - 実アプリケーション開発事例
    - 武宮, 首藤, 田中, 関口:"Grid環境上における気象予報シミュレーションシステムの構築", 情報処理学会論文誌, Vol.44, No. SIG 11