

# FDPS 講習会 C++編

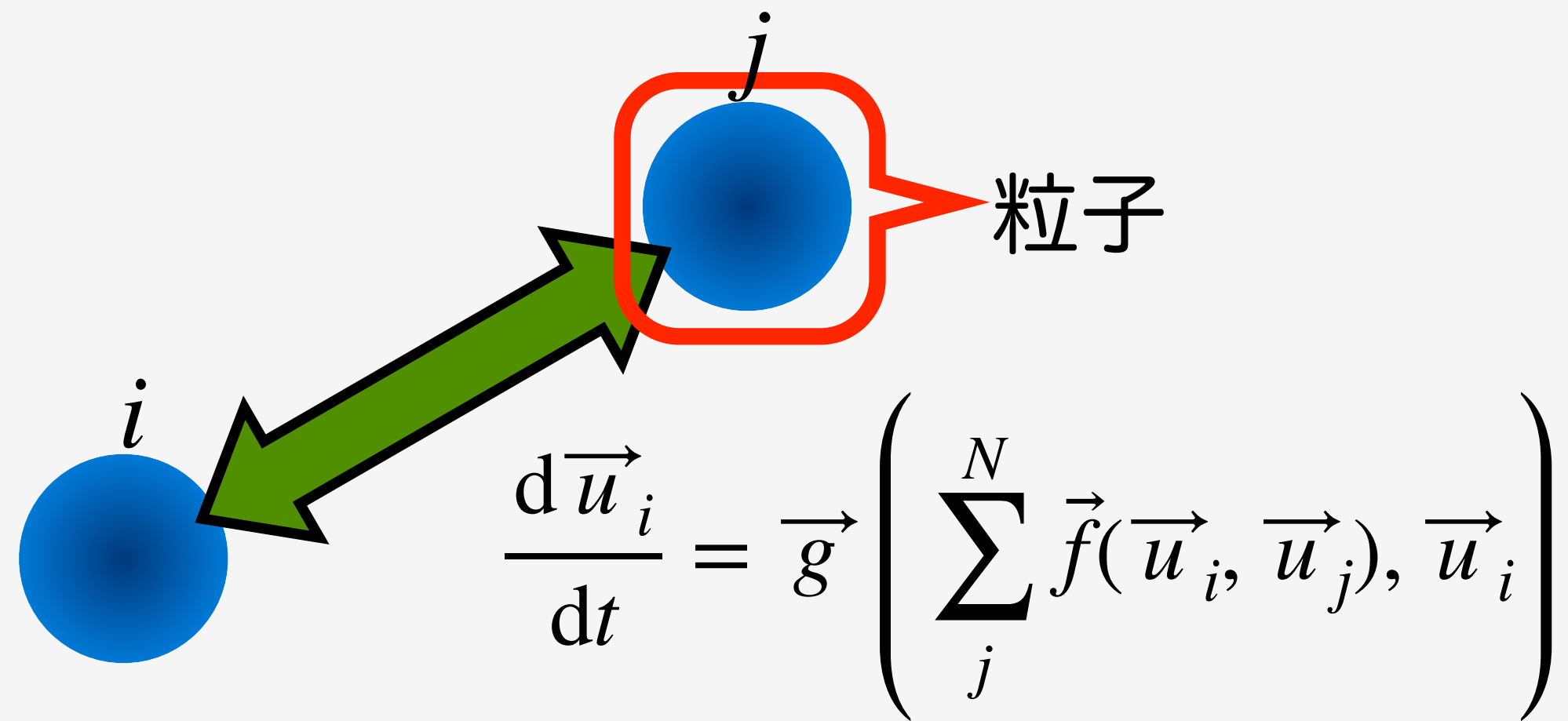
細野 七月 (神戸大学/RIKEN CCS)

2020/08/19 FDPS講習会

```
//=====\\
||
||  . . . . .  . . . . .  . . . . .  . . . . .
||  . .        . .      :  . .      :  . .
||  . . . . .  . .      :  . . . . .  '  . . . . .
||  . .        . . . . . '  . .      '  . . . . .
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```

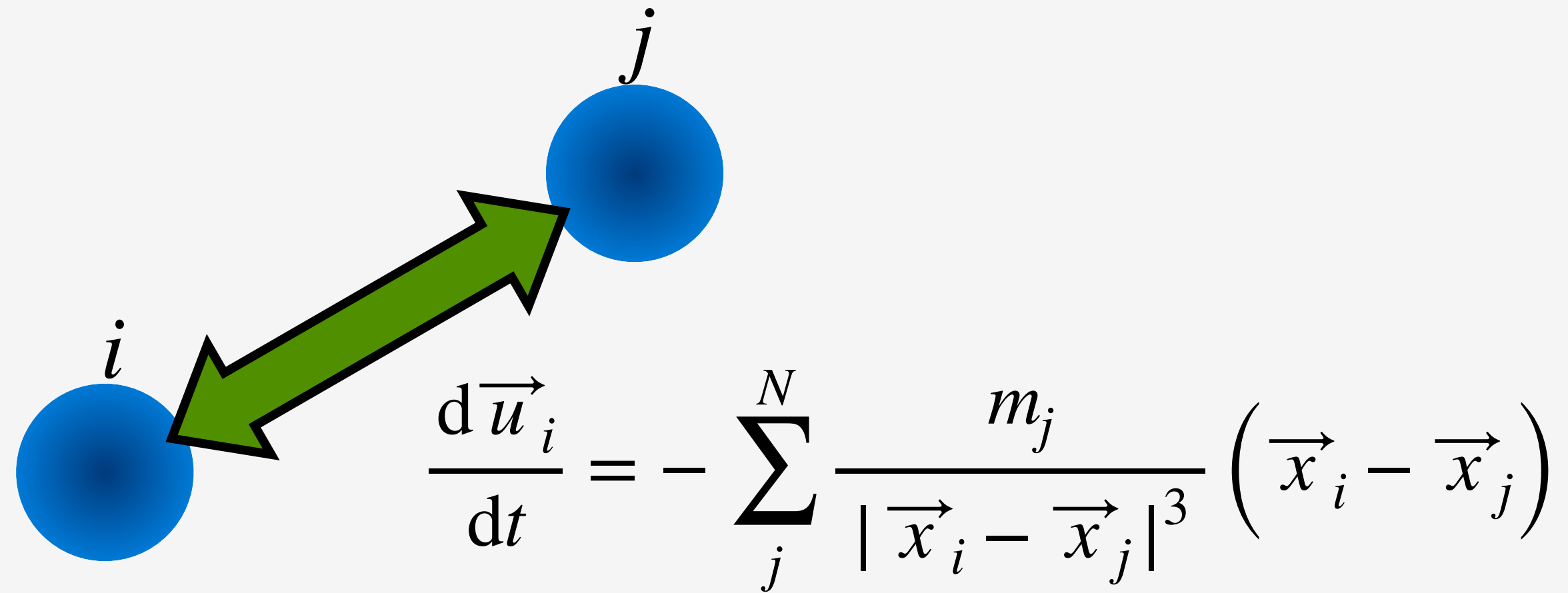
# FDPSが扱うもの

◆個々の要素(粒子)に作用する力が、**粒子間相互作用の重ね合わせ**で記述できるもの



# FDPSが扱うもの

◆個々の要素(粒子)に作用する力が、**粒子間相互作用の重ね合わせ**で記述できるもの



## 例 ニュートン重力

2020/08/19 FDPS講習会

```
//=====\\
||
||
||  . . . . .  . . . . .  . . . . .  . . . . .
||  .:          :  :  :  :  :  :
||  . . . . .  :  :  :  :  :  :  :  :  :
||  .:          :  :  :  :  :  :  :  :  :
||
||  Framework for Developing
||  Particle Simulator
||
||=====\\
```

# 習得しておきたいC++の機能

## ◆FDPS用いるにおいて、知っておきたいC++の機能

- 名前空間
  - `PS::F64` など
- クラス
  - メンバ変数とメンバ関数
- テンプレート
  - `template<typename T>` など
- 標準ライブラリ
  - `std::vector` `std::cout` など
  - ただし今回のサンプルコードではI/O用の`std::cout`などが使われている程度なので省略

これ以上の事は基本的に必要ない。

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          :::::::::::  ::          \  \  .....
||
||          Framework for Developing
||          Particle Simulator
||
||=====\\
```

# C++の機能：名前空間とは

- ◆グローバルな変数、関数、型名などの名前被りを防ぐ機能。  
FDPSでは、F64vecといった型が存在している(後述)が、PSという名前空間に含まれているので、ユーザーがF64vecという型を定義しても問題ない。
- ◆名前空間へのアクセスは::演算子を用いる。

```
PS::F64vec v_FDPS;  
F64vec v_user;
```

2020/08/19 FDPS講習会

```
//=====\\
||
||
||  . . . . .  . . . . .  . . . . .  . . . . .
||  .:          :  :  :  :  :  :
||  . . . . .  :  :  :  :  :  :  :  :  :
||  .:          :  :  :  :  :  :  :  :  :
||
||  Framework for Developing
||  Particle Simulator
||
||=====\\
```



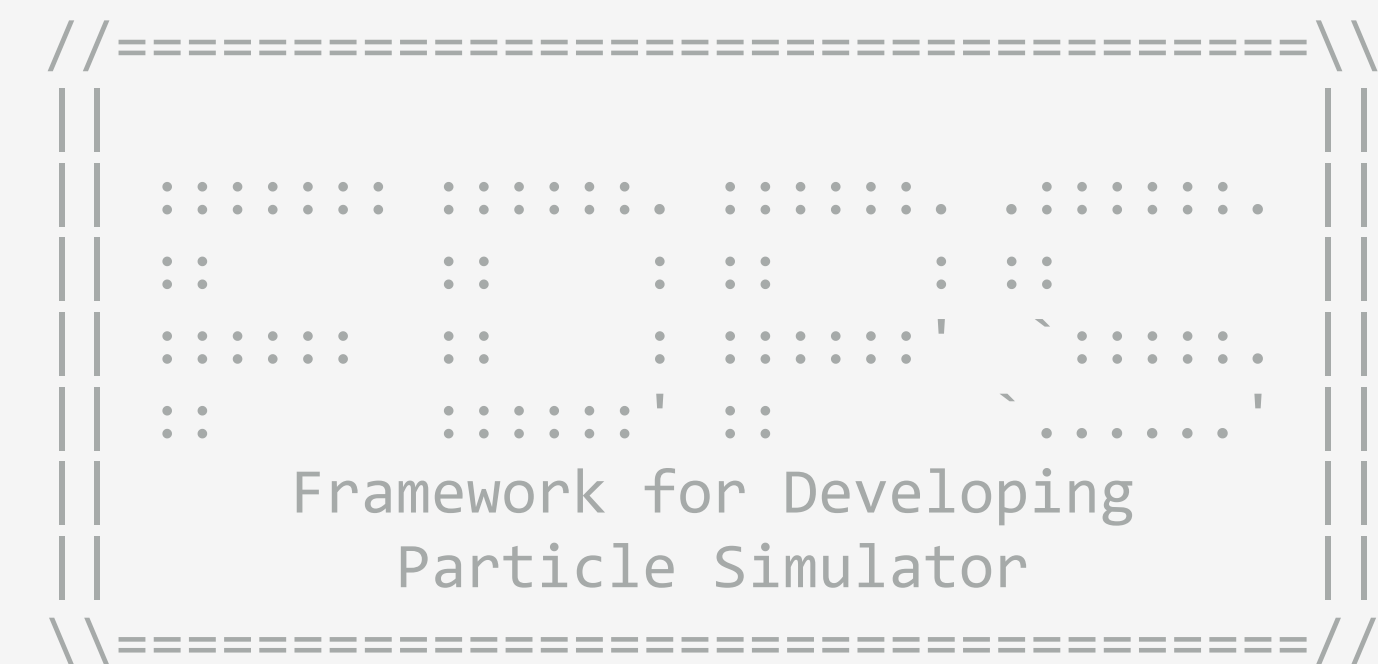
# C++の機能：クラスとは

◆複数のデータをまとめたもの  
Cで言うところの構造体

◆**クラス**を使うと

- 演算子を定義出来る。
- サブルーチンに値を送るときに楽。
- 後からデータ付け足すのも楽。

◆FDPS中では、三次元ベクトルをやりとりするのに便利な**ベクトルクラス**が用意されている。  
また、ユーザーは粒子データをやりとりするために必要な**粒子クラス**を記述する必要がある。



# C++の機能：クラス例(ベクトルクラス)

- ◆これは数学演算子を定義している例  
三次元ベクトル演算を実装してあるFDPSの組み込みクラス  
(二次元にすることも可能)

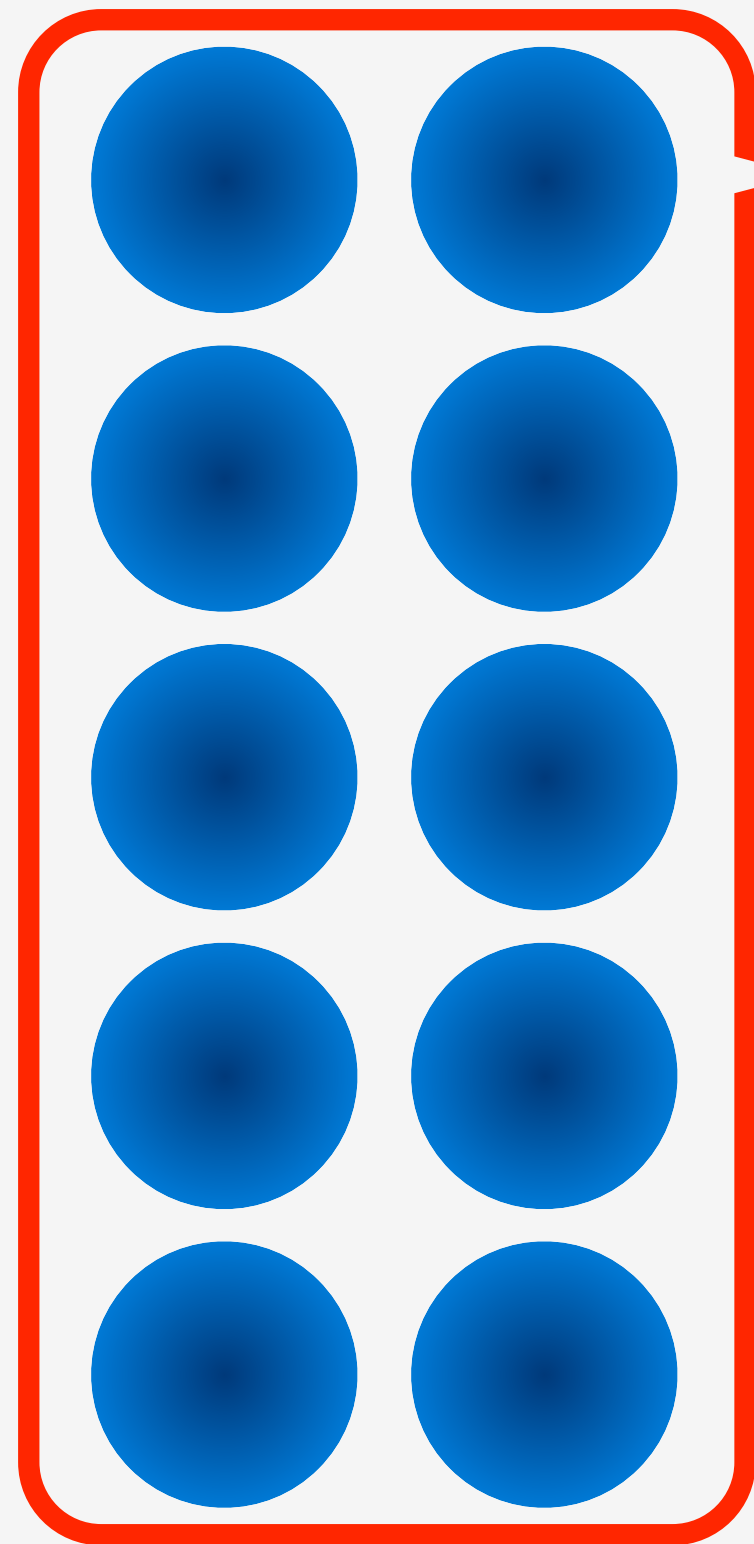
```
PS::F64vec v1 = PS::F64vec( 1.0,  2.5, -3.0);
PS::F64vec v2 = PS::F64vec(-2.0, -1.5,  2.0);

PS::F64vec v_add = v1 + v2;
PS::F64vec v_sub = v1 - v2;
PS::F64     inner = v1 * v2;
PS::F64vec outer = v1 ^ v2;

std::cout << v_add.x << std::endl;
std::cout << v_sub    << std::endl;
```

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          ::::::::::: '  ::          \  ..... '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```

# C++の機能：クラス例(粒子クラス)



## Nbody個の粒子

## 必要なデータは…

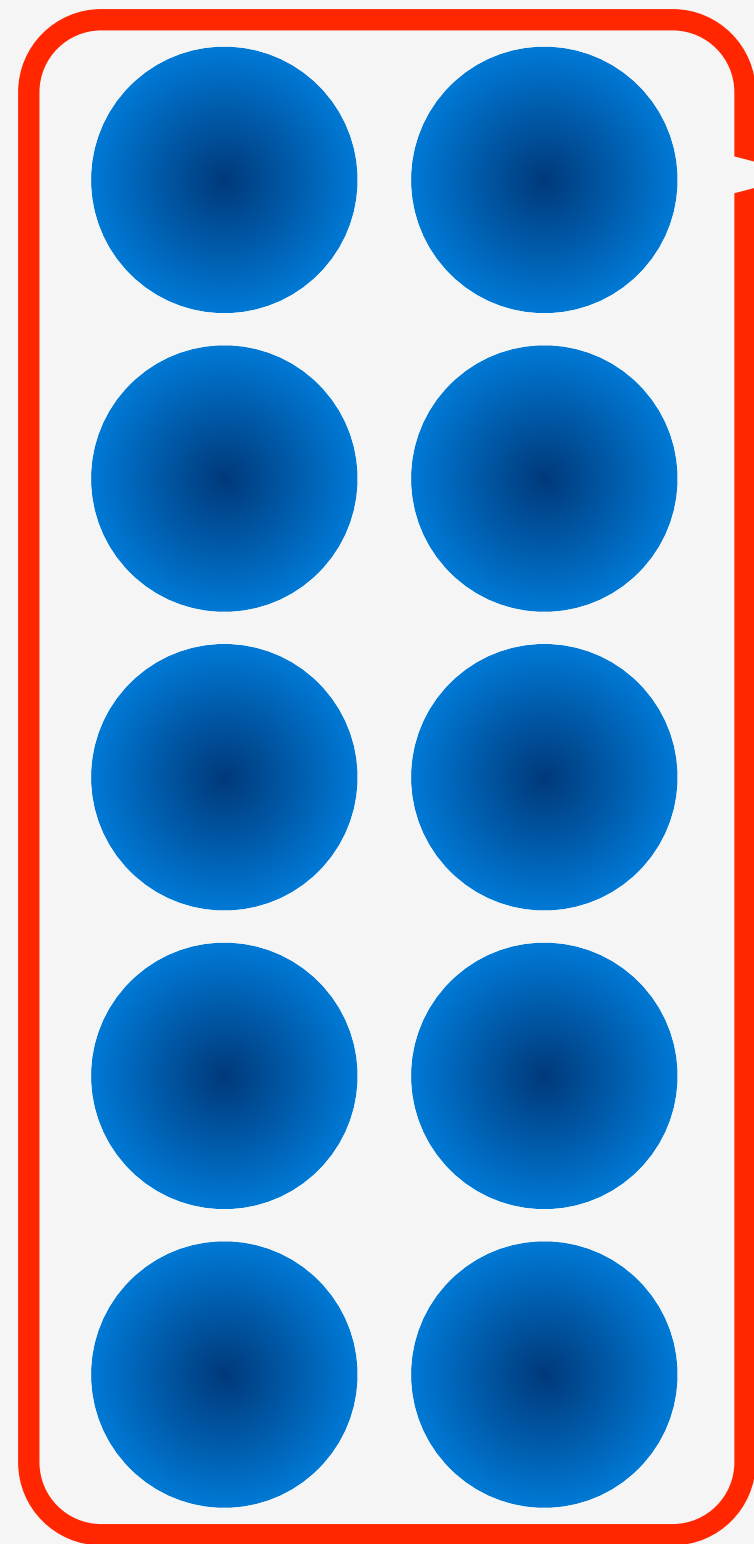
- mass
- position(x, y, z)
- velocity(x, y, z)
- acceleration(x, y, z)

2020/08/19 FDPS講習会

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          ::::::::::: '  ::          \  ..... '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```



# C++の機能：クラス例(粒子クラス)

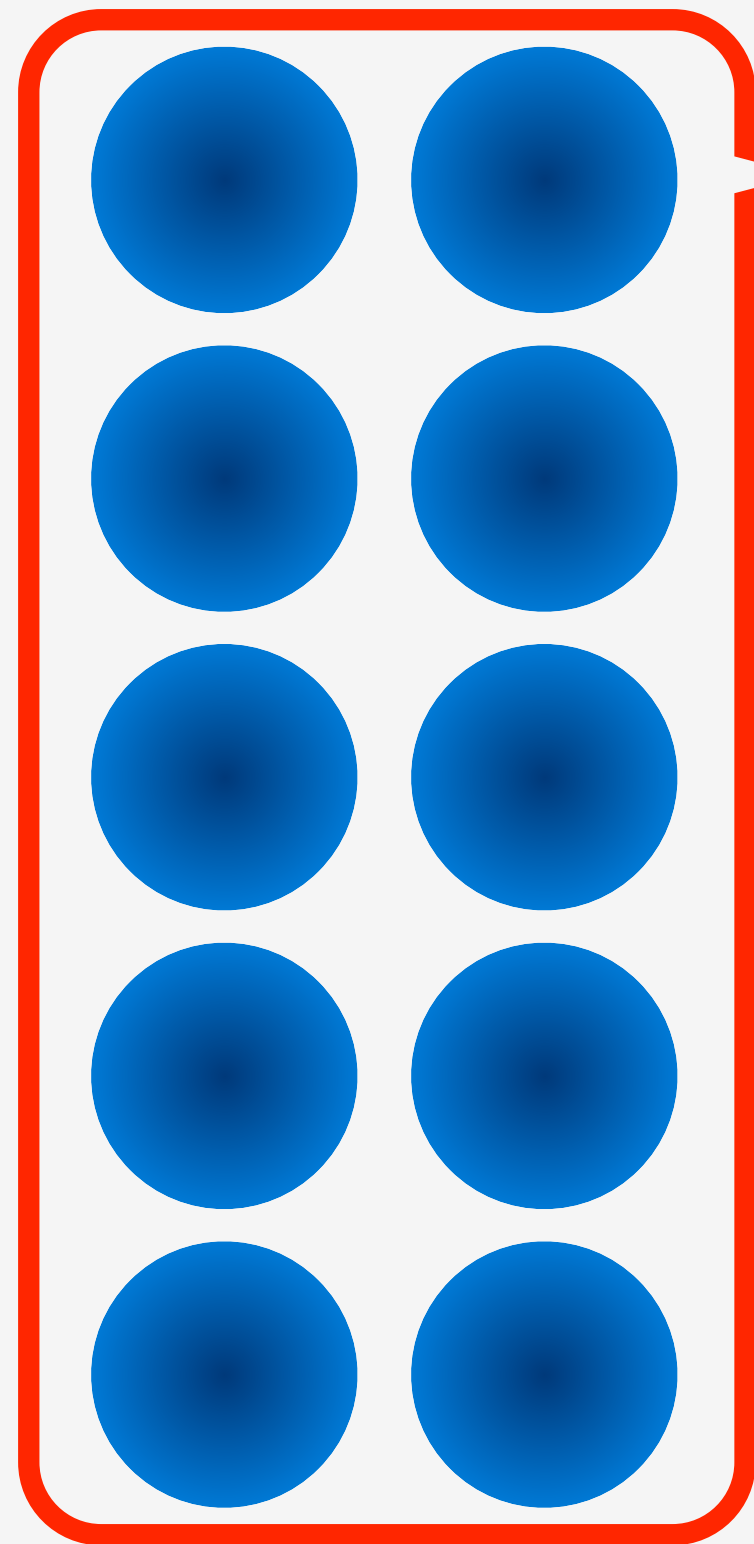


## Nbody個の粒子

```
double mass[Nbody];
double position[Nbody][3];
double velocity[Nbody][3];
double acceleration[Nbody][3];
```

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          ::::::::::: '  ::          \  ..... '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```

# C++の機能：クラス例(粒子クラス)

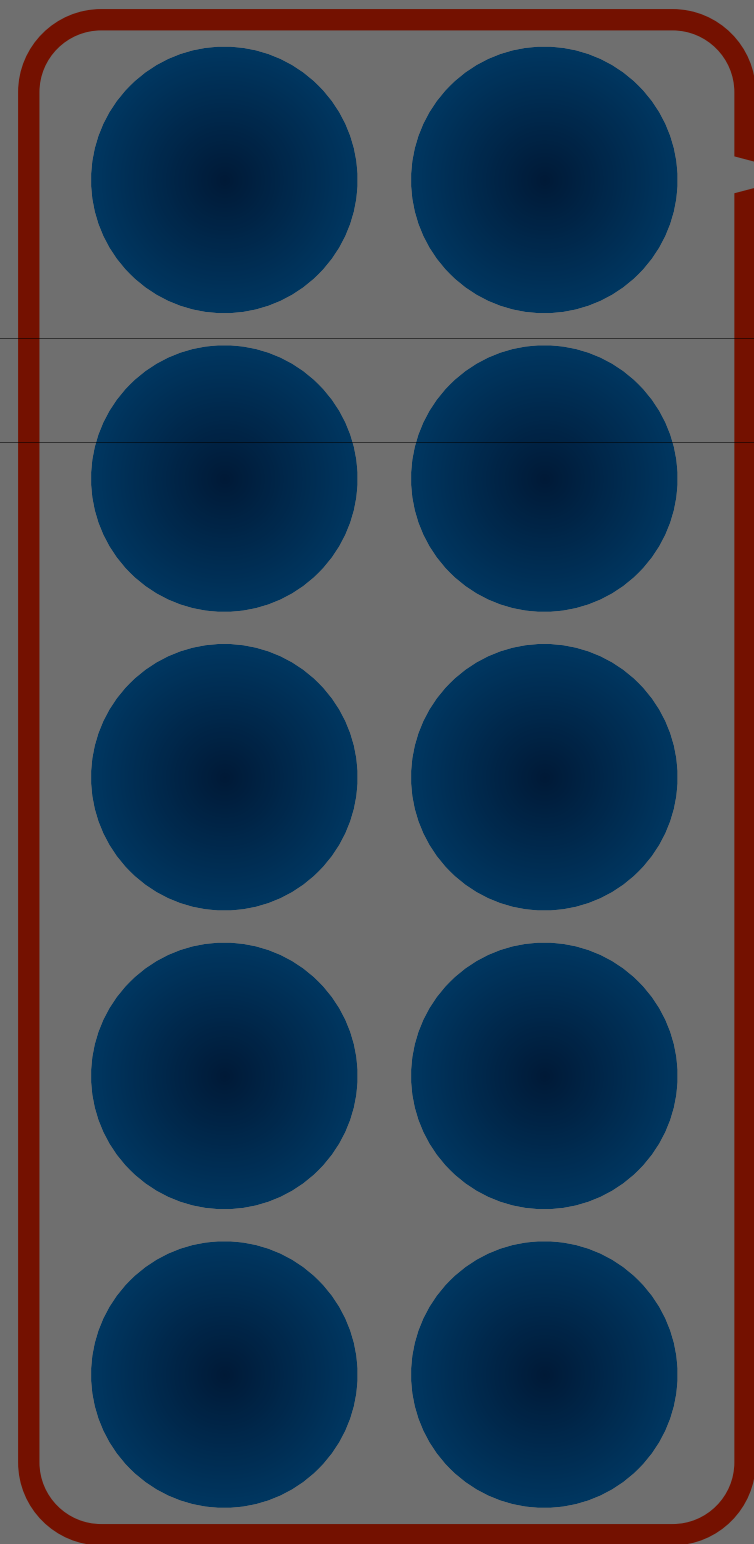


## Nbody個の粒子

```
class FPGrav{
    double mass;
    double position[3];
    double velocity[3];
    double acceleration[3];
}body[Nbody];
```

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          :::::::::::  '  ::          \  .....  '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```

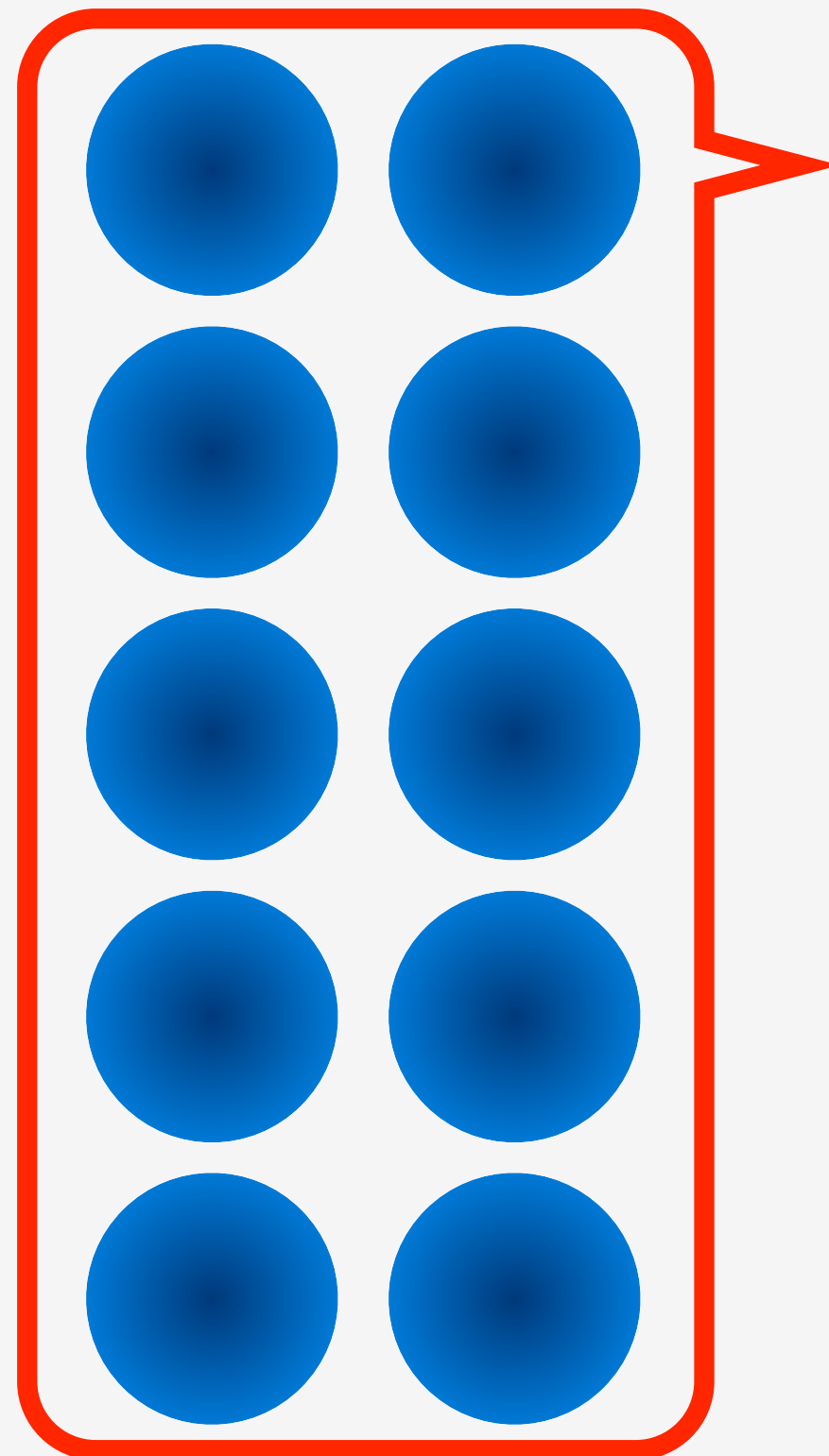
# C++の機能：クラス例(粒子クラス)



# Nbody個の粒子

```
class FPGrav{
    double mass;
    double position[3];
    double velocity[3];
    double acceleration[3];
}body[Nbody];
```

# C++の機能：クラス例(粒子クラス)



## Nbody個の粒子

```
class FPGrav{
    PS::F64      mass;
    PS::F64vec   position[3];
    PS::F64vec   velocity[3];
    PS::F64vec   acceleration[3];
}body[Nbody];
```

## クラス内クラスも可能

2020/08/19 FDPS講習会

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          ::::::::::: '  ::          \  ..... '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```



# C++の機能：クラスのメリット

## ◆クラスを使うとサブルーチンに値を送るときに楽。

◆後からデータ付け足すのも楽。

```
void doSomething(double mass[],
                 double position[][3],
                 double velocity[][3],
                 double acceleration[][3]){
    //do something here
}

void doSomething(FPGrav particle[]){
    //do something here
}
```

[illegible]

# C++の機能：メンバ関数

- ◆クラス内の変数(メンバ変数)達に対して、何らかの処理を加えたい時に記述するもの。
- ◆FDPSの場合、  
FDPSとユーザーコード間でデータのやりとりをするためのメンバ関数を書く必要がある。

```
class FPGrav{
    PS::F64      mass;
    PS::F64vec   position[3];
    PS::F64vec   velocity[3];
    PS::F64vec   acceleration[3];
    PS::F64vec   getPos(){
        return position;
    }
}body[Nbody];

PS::F64vec position = body[0].getPos();
```

```
//=====\\
||
||
||  ::::::::::  ::::::::::  ::::::::::  .::::::::::::
||  ::          ::          :  ::          :  ::
||  ::::::::::  ::          :  ::::::::::  '  \  ::::::::::
||  ::          ::::::::::'  ::          \  .....'
```

Framework for Developing  
Particle Simulator

```
\\=====//
```

# C++の機能：メンバ関数

- ◆ **クラス**内の変数(メンバ変数)達に対して、何らかの処理を加えたい時に記述するもの。
- ◆ FDPSの場合、  
FDPSとユーザーコード間でデータのやりとりをするための **メンバ関数**を書く必要がある。

```
class FPGrav{
    PS::F64      mass;
    PS::F64vec   position[3];
    PS::F64vec   velocity[3];
    PS::F64vec   acceleration[3];
    PS::F64vec   getPos(){
        return position;
    }
}body[Nbody];

PS::F64vec position = body[0].getPos();
```



# C++の機能：メンバ関数

- ◆クラス内の変数(メンバ変数)達に対して、何らかの処理を加えたい時に記述するもの。
- ◆FDPSの場合、  
FDPSとユーザーコード間でデータのやりとりをするためのメンバ関数を書く必要がある。

```
class FPGrav{
    PS::F64      mass;
    PS::F64vec   position[3];
    PS::F64vec   velocity[3];
    PS::F64vec   acceleration[3];
    PS::F64vec   getPos(){
        return position;
    }
}body[Nbody];

PS::F64vec position = body[0].getPos();
```

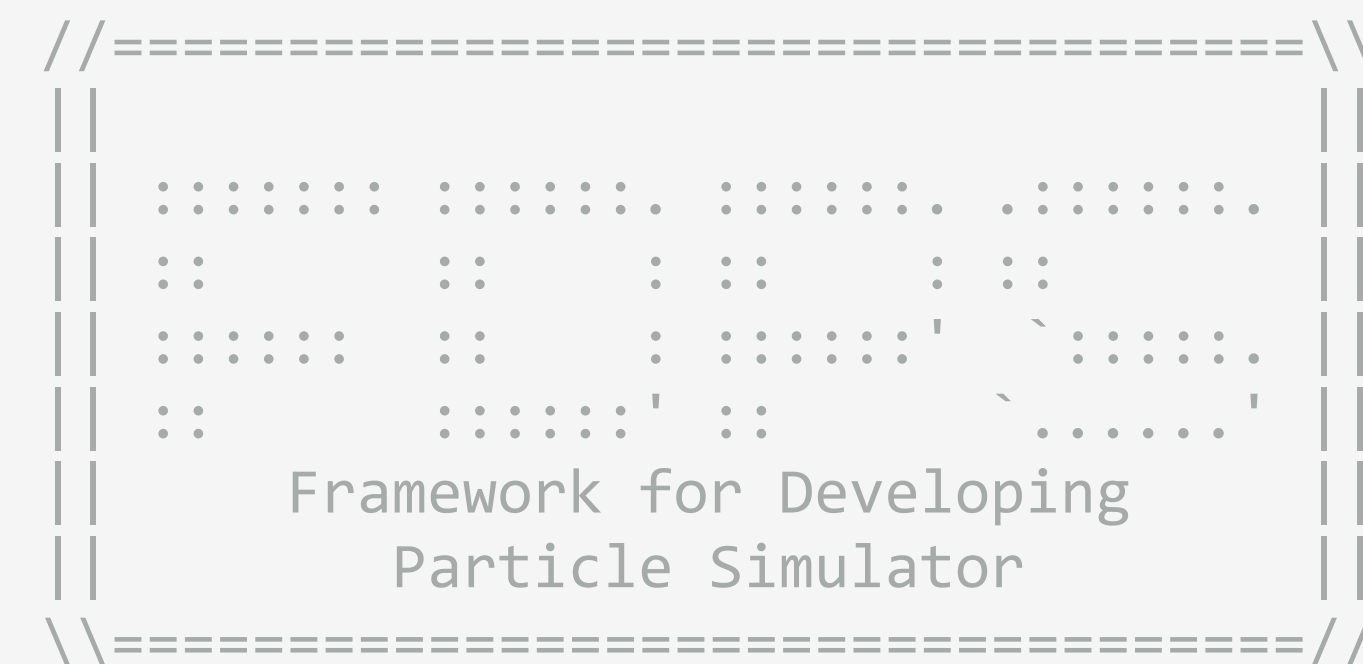


# C++の機能：テンプレート

◆同じ処理を違う型に対して行いたい時に書くもの。

クラスをテンプレート化したクラステンプレートと、  
関数をテンプレート化した関数テンプレートが存在する。

◆まずクラステンプレートについて解説し、その後関数テンプレートに関して解説する。



# C++の機能：クラステンプレート

- ◆FDPSでは、粒子群クラステンプレートの<>の中に  
粒子クラスを入れる事で、粒子群クラスの実体が作られる。  
同様に、相互作用ツリークラスの実体作られる。

```
//粒子群クラス
PS::ParticleSystem<FPGrav> system_grav;
//相互作用ツリークラス
PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>Monopole tree_grav;
```

# C++の機能：クラステンプレート

- ◆FDPSでは、粒子群クラステンプレートの<>の中に粒子クラスを入れる事で、粒子群クラスの実体が作られる。同様に、相互作用ツリークラスの実体が作られる。

```
//粒子群クラス
```

```
PS::ParticleSystem<FPGrav> system_grav;
```

```
//相互作用ツリークラス
```

```
PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>Monopole tree_grav;
```

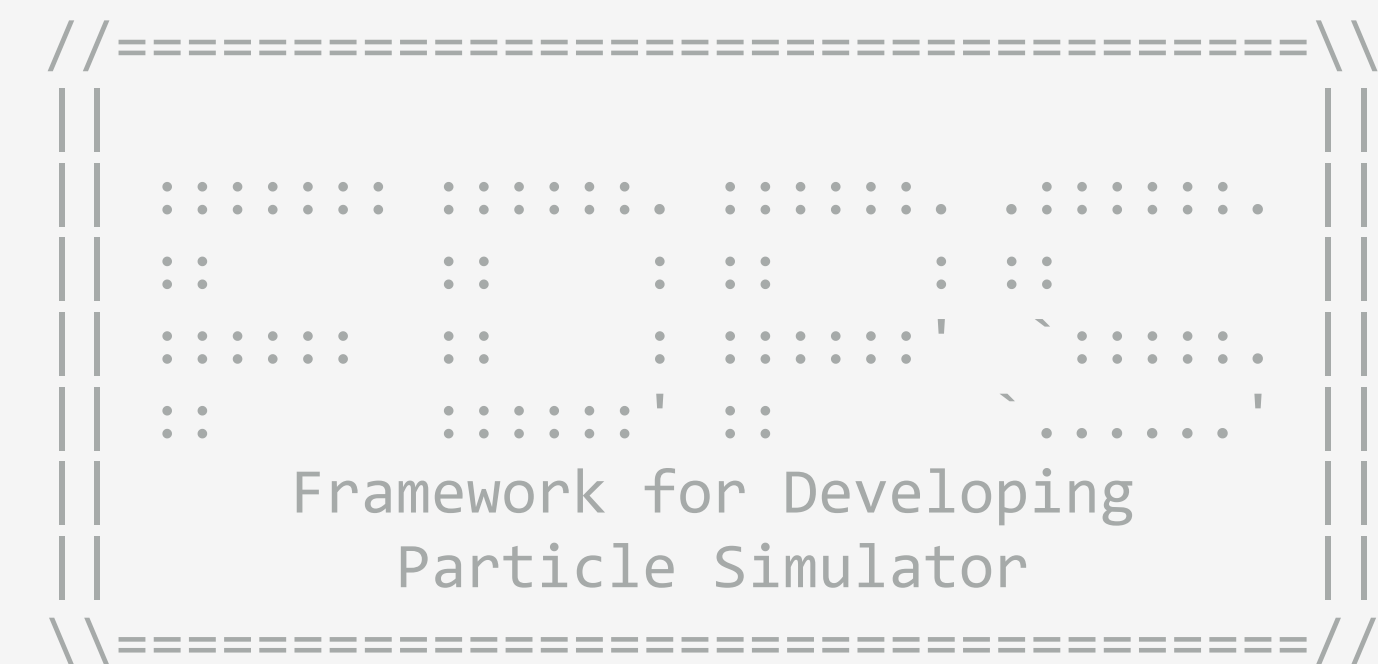
```
class FPGrav{
    PS::F64    mass;
    PS::F64vec position[3];
    PS::F64vec velocity[3];
    PS::F64vec acceleration[3];
    PS::F64vec getPos(){
        return position;
    }
}

body[Nbody];
```



# C++の機能：関数テンプレート

- ◆FDPSでは相互作用関数は関数テンプレートを用いて記述出来る。
- ◆関数テンプレートを用いずに記述する事も可能だが、相互作用が長距離力の場合、計算内容的には全く同じものにも関わらず、近傍の粒子からの相互作用とツリーノードからの相互作用2つを書かなければならない。
- ◆関数テンプレートを用いると記述するのは1つだけで済むので、今回は関数テンプレートを用いたサンプルコードを紹介する。





# C++の機能：関数テンプレート(続き)

- ◆FDPSでは、ツリーノードクラスは既に用意されている(PS::SPJMonopole)ため、それを使う。
- ◆相互作用関数はSPJMonopoleも使うので、getPos();などを用いて書く必要がある。

```
template <class Tparticle>
void CalcGravity(const FPGrav * ep_i,
                const PS::S32 n_ip,
                const TParticle * ep_j,
                const PS::S32 n_jp,
                FPGrav * force){
    for(PS::S32 i = 0 ; i < n_ip ; i ++){
        PS::F64vec xi = ep_i[i].getPos();
        for(PS::S32 j = 0 ; j < n_jp ; j ++){
            PS::F64vec rij = xi - ep_j[j].getPos();
            //calc gravity
        }
    }
}
```

```
//=====\\
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::      :  ::      :  ::
||  ::::::::::  ::      :  ::::::::::'  \  ::::::::::
||  ::          ::::::::::'  ::      \  .....'
```

Framework for Developing  
Particle Simulator

```
//=====\\
```



# サンプルコード

## ◆今回のサンプルコードの内容

- 粒子間相互作用は、重力
  - Phantom-GRAPe (Tanikawa+, 2011; 2012) ありとなし両方
- 時間積分法はleap-frog法
- 初期条件はその場生成
  - ファイル読み込みではない

## ◆ファイル構成

- user-defined.hpp
- nbody.cpp

他にも色々入っているが必要なものはすべてこの2つの中にある

```
//=====\\
||
||
||  . . . . .  . . . . .  . . . . .  . . . . .
||  .:         :  :         :  :
||  . . . . .  :  :         :  :         :  :
||  .:         . . . . .  :  :         :  :
||
||  Framework for Developing
||  Particle Simulator
||
||=====\\
```

# サンプルコード used\_defined.hpp

```
#pragma once
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        fscanf(fp, "%lf\n", &time);
        fscanf(fp, "%lld\n", &n_body);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n", n_body);
    }
};
```

```
class FPGrav{
public:
    PS::S64      id;
    PS::F64      mass;
    PS::F64vec   pos;
    PS::F64vec   vel;
```

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          ::::::::::: '  ::          \  ..... '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```



# サンプルコード used\_defined.hpp

```
#pragma once
class FileHeader{ クラス
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        fscanf(fp, "%lf\n", &time);
        fscanf(fp, "%lld\n", &n_body);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n", n_body);
    }
};
```

```
class FPGrav{
public:
    PS::S64 id;
    PS::F64 mass;
    PS::F64vec pos;
    PS::F64vec vel;
```

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::      ::      :  :  :  :  :  :  :  :  :  |
|  :::::::::: :  :  :  :  :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  :  |
|  Framework for Developing |
|  Particle Simulator       |
\\=====\\
```

# サンプルコード used\_defined.hpp

```
#pragma once
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        fscanf(fp, "%lf\n", &time);
        fscanf(fp, "%lld\n", &n_body);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n", n_body);
    }
};
```

ファイルヘッダークラス

```
class FPGrav{
public:
    PS::S64 id;
    PS::F64 mass;
    PS::F64vec pos;
    PS::F64vec vel;
```

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::      ::      :  :  :  :  :  :  :  :  :  |
|  :::::::::: :  :  :  :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  |
|  Framework for Developing |
|  Particle Simulator       |
\\=====\\
```

# サンプルコード used\_defined.hpp

```
#pragma once
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        fscanf(fp, "%lf\n", &time);
        fscanf(fp, "%lld\n", &n_body);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%lld\n", n_body);
    }
};
```

## ファイルヘッダークラス

```
class FPGrav{
public:
    PS::S64      id;
    PS::F64      mass;
    PS::F64vec   pos;
    PS::F64vec   vel;
```

```
//=====\\
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::      :  ::          :  ::
||  ::::::::::  ::      :  ::::::::::  '  \  ::::::::::
||  ::          ::::::::::'  ::          \  .....'
```

Framework for Developing  
Particle Simulator

```
\\=====\\
```

# サンプルコード used\_defined.hpp

```
class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec vel;
    PS::F64vec acc;
    PS::F64    pot;
    static PS::F64 eps;
    PS::F64vec getPos() const {
        return pos;
    }
    PS::F64 getCharge() const {
        return mass;
    }
    void copyFromFP(const FPGrav & fp){
        mass = fp.mass;
        pos  = fp.pos;
    }
    void copyFromForce(const FPGrav & force) {
        acc = force.acc;
        pot = force.pot;
    }
};
```

# 粒子クラス

```

//=====\\
||
||
||  :::::  :::::  :::::  :::::
||  ::      ::      :  ::      :  ::
||  :::::  ::      :  :::::  '  \  :::::
||  ::      :::::  '  ::      \  .....  '
||
||      Framework for Developing
||      Particle Simulator
||
\\=====\\

```



# サンプルコード used\_defined.hpp

```
class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec vel;
    PS::F64vec acc;
    PS::F64    pot;
    static PS::F64 eps;
    PS::F64vec getPos() const {
        return pos;
    }
    PS::F64 getCharge() const {
        return mass;
    }
    void copyFromFP(const FPGrav & fp){
        mass = fp.mass;
        pos  = fp.pos;
    }
    void copyFromForce(const FPGrav & force) {
        acc = force.acc;
        pot = force.pot;
    }
};
```

# 粒子クラス

```

//=====\\
||
||
|| :::::  :::::  :::::  .:::
|| ::      ::      :  ::      :
|| :::::  ::      :  :::::  '  \  :::::
|| ::      :::::  '  ::      \  ..... '
||
||           Framework for Developing
||           Particle Simulator
||
\\=====\\

```

# サンプルコード used\_defined.hpp

```
PS::F64vec getPos() const {
    return pos;
}
PS::F64 getCharge() const {
    return mass;
}
void copyFromFP(const FPGrav & fp){
    mass = fp.mass;
    pos  = fp.pos;
}
void copyFromForce(const FPGrav & force) {
    acc = force.acc;
    pot = force.pot;
}
void clear() {
    acc = 0.0;
    pot = 0.0;
}
```

## メンバー関数

```
void writeAscii(FILE* fp) const {
    fprintf(fp, "%lld\t%g\t%g\t%g\t%g\t%g\t%g\n",
            this->id, this->mass,
            this->pos.x, this->pos.y, this->pos.z,
            this->vel.x, this->vel.y, this->vel.z);
}
```

```
//=====\\
||
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::      :  ::          :  ::
||  ::::::::::  ::      :  ::::::::::'  \  ::::::::::
||  ::          ::::::::::'  ::          \  .....'
```

Framework for Developing  
Particle Simulator

```
\\=====\\
```

# サンプルコード used\_defined.hpp

## I/O用メンバー関数

```
void clear() {
    acc = 0.0;
    pot = 0.0;
}

void writeAscii(FILE* fp) const {
    fprintf(fp, "%lld\t%g\t%g\t%g\t%g\t%g\t%g\t%g\n",
            this->id, this->mass,
            this->pos.x, this->pos.y, this->pos.z,
            this->vel.x, this->vel.y, this->vel.z);
}

void readAscii(FILE* fp) {
    fscanf(fp, "%lld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
           &this->id, &this->mass,
           &this->pos.x, &this->pos.y, &this->pos.z,
           &this->vel.x, &this->vel.y, &this->vel.z);
}

};
```

```
#ifdef ENABLE_PHANTOM_GRAPE_X86
```

```
template <class TParticle>
void CalcGravity(const FPGrav * iptcl,
                const PS::S32 ni,
```

[illegible]



# サンプルコード used\_defined.hpp

```
}  
};
```

```
#ifdef ENABLE_PHANTOM_GRAPE_X86
```

関数テンプレート

Phantom GRAPEあり

```
template <class TParticle>  
void CalcGravity(const FPGrav * iptcl,  
                const PS::S32 ni,  
                const TParticleJ * jptcl,  
                const PS::S32 nj,  
                FPGrav * force) {  
    const PS::S32 nipipe = ni;  
    const PS::S32 njpipe = nj;  
    PS::F64 (*xi)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);  
    PS::F64 (*ai)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);  
    PS::F64 *pi = (PS::F64 *)malloc(sizeof(PS::F64) * nipipe);  
    PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * njpipe * PS::DIMENSION);  
    PS::F64 *mj = (PS::F64 *)malloc(sizeof(PS::F64) * njpipe);  
    for(PS::S32 i = 0; i < ni; i++) {  
        xi[i][0] = iptcl[i].getPos()[0];  
        xi[i][1] = iptcl[i].getPos()[1];  
        xi[i][2] = iptcl[i].getPos()[2];  
        ai[i][0] = 0.0;  
    }  
}
```

```
.....  
::      ::      ::      ::  
.....  
::      ::      ::      ::  
.....  
Framework for Developing  
Particle Simulator  
\\=====\\
```



# サンプルコード used\_defined.hpp

```
        force[i].acc[0] += ai[i][0];
        force[i].acc[1] += ai[i][1];
        force[i].acc[2] += ai[i][2];
        force[i].pot    -= pi[i];
    }
    free(xi);
    free(ai);
    free(pi);
    free(xj);
    free(mj);
}

#else

template <class TParticleJ>
void CalcGravity(const FPGrav * ep_i,
                const PS::S32 n_ip,
                const TParticleJ * ep_j,
                const PS::S32 n_jp,
                FPGrav * force) {
    PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
    for(PS::S32 i = 0; i < n_ip; i++){
        PS::F64 vec_xi = ep_i[i].getPos().
```

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::      ::      :  :  :  :  :  :  :  :  :  |
|  :::::::::: :  :  :  :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  |
|  Framework for Developing |
|  Particle Simulator       |
\\=====//
```

# サンプルコード used\_defined.hpp

`template <class TParticle>` } 相互作用関数テンプレート

```
void CalcGravity(const FPGrav * ep_i,  
                const PS::S32 n_ip,  
                const TParticleJ * ep_j,  
                const PS::S32 n_jp,  
                FPGrav * force) {
```

```
PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
```

```
for(PS::S32 i = 0; i < n ip; i++){
```

```
PS::F64vec xi = ep_i[i].getPos();
```

```
PS::F64vec ai = 0.0;
```

```
PS::F64 poti = 0.0;
```

```
for(PS::S32 j = 0; j < n_jp; j++){
```

```
PS::F64vec rij    = xi - ep_j[j].getPos();
```

```
PS::F64    r3_inv = rij * rij + eps2;
```

```
PS::F64    r_inv    = 1.0/sqrt(r3_inv);
```

```
r3_inv = r_inv * r_inv;
```

```
r_inv *= ep_j[j].getCharge();
```

```
r3 inv *= r inv;
```

```
ai -= r3_inv * rij;
```

```
poti -= r inv;
```

2

```
force[i].acc += ai;
```

## メンバ関数呼び出し

```
//=====\\
||
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::          :  ::          :  ::
||  ::::::::::  ::          :  ::::::::::  '  \  ::::::::::
||  ::          ::::::::::'  ::          \  .....'
```

Framework for Developing  
Particle Simulator

```
\\=====\\
```

# サンプルコード used\_defined.hpp

```
        PS::F64    r3_inv = rij * rij + eps2;
        PS::F64    r_inv  = 1.0/sqrt(r3_inv);
        r3_inv     = r_inv * r_inv;
        r_inv      *= ep_j[j].getCharge();
        r3_inv     *= r_inv;
        ai         -= r3_inv * rij;
        poti       -= r_inv;
    }
    force[i].acc += ai;
    force[i].pot += poti;
}

#endif
```

# サンプルコード user\_defined.hpp

◆user\_defined.hppはこれだけ。  
おおよそ150行。

2020/08/19 FDPS講習会

```
//=====\\
||
||
||  :::::  :::::  :::::  :::::
||  ::      ::      :  ::      :  ::
||  :::::  ::      :  :::::  '  \  :::::
||  ::      :::::  '  ::      \  .....  '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```



# サンプルコード nbody.cpp

```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"

void makeColdUniformSphere(const PS::F64 mass_glb,
                           const PS::S64 n_glb,
                           const PS::S64 n_loc,
                           PS::F64 *& mass,
                           PS::F64vec *& pos,
                           PS::F64vec *& vel,
                           const PS::F64 eng = -0.25,
                           const PS::S32 seed = 0) {
    assert(eng < 0.0);
    {
        PS::MTTS mt;
        mt.init genrand(0);
    }
}
```

```
//=====\\
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::      :  ::      :  ::
||  ::::::::::  ::      :  ::::::::::'  \  ::::::::::
||  ::          ::::::::::'  ::      \  .....'
```

Framework for Developing  
Particle Simulator

```
//=====\\
```

# サンプルコード nbody.cpp

```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"
```

FDPSのヘッダー読み込み

```
void makeColdUniformSphere(const PS::F64 mass_glb,
                           const PS::S64 n_glb,
                           const PS::S64 n_loc,
                           PS::F64 *& mass,
                           PS::F64vec *& pos,
                           PS::F64vec *& vel,
                           const PS::F64 eng = -0.25,
                           const PS::S32 seed = 0) {
```

```
assert(eng < 0.0);
{
```

```
PS::MTTS mt;  
mt.init genrand(0);
```

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          ::::::::::: '  ::          \  ..... '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```

# サンプルコード nbody.cpp

```
}
```

```
template<class Tpsys>
void kick(Tpsys & system,
         const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].vel += system[i].acc * dt;
    }
}
```

粒子数が取得可能

```
template<class Tpsys>
void drift(Tpsys & system,
          const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].pos += system[i].vel * dt;
    }
}
```

```
template<class Tpsys>
void calcEnergy(const Tpsys & system,
               PS::F64 & etot,
               PS::F64 & ekin,
               PS::F64 & epot,
               const bool clear=true){
    if(clear){
```

```
//=====\\
||
||  :::::::::: :::::::::: :::::::::: ::::::::::
||  ::         ::         :         :         :
||  :::::::::: :         :         :         :
||  ::         :::::::::: :         :         :
||
||  Framework for Developing
||  Particle Simulator
||
||=====\\
```

# サンプルコード nbody.cpp

```
template<class Tpsys>
void kick(Tpsys & system,
         const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].vel += system[i].acc * dt;
    }
}
```

粒子群クラスに[i]をつけると粒子データにアクセス可能

```
template<class Tpsys>
void drift(Tpsys & system,
          const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].pos += system[i].vel * dt;
    }
}
```

```
template<class Tpsys>
void calcEnergy(const Tpsys & system,
               PS::F64 & etot,
               PS::F64 & ekin,
               PS::F64 & epot,
               const bool clear=true){
    if(clear){
```

```
//=====\\
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::      :  ::      :  ::
||  ::::::::::  ::      :  ::::::::::'  \  ::::::::::
||  ::          ::::::::::'  ::      \  .....'
```

Framework for Developing  
Particle Simulator

```
//=====\\
```



# サンプルコード nbody.cpp

`int main(int argc, char *argv[]) {` メイン関数開始

```
std::cout<<std::setprecision(15);  
std::cerr<<std::setprecision(15);
```

```
PS::Initialize(argc, argv);
```

```
PS::F32 theta = 0.5;
```

```
PS::S32 n_leaf_limit = 8;
```

```
PS::S32 n_group_limit = 64;
```

```
PS::F32 time_end = 10.0;
```

```
PS::F32 dt = 1.0 / 128.0;
```

```
PS::F32 dt_diag = 1.0 / 8.0;
```

```
PS::F32 dt_snap = 1.0;
```

```
char dir_name[1024];
```

```
PS::S64 n_tot = 1024;
```

```
PS::S32 c;
```

```
sprintf(dir_name, "./result");
```

```
opterr = 0;
```

```
while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
```

```
    switch(c){
```

```
        case 'o':
```

```
            sprintf(dir_name,optarg);
```

```
            break;
```

```
        case 't':
```

```
            theta = atof(optarg);
```

```
            std::cerr<<"theta =" << theta << std::endl;
```

```
            break;
```

```
//=====\\  
|  :::::::::: :::::::::: :::::::::: :::::::::: |  
|  ::      ::      :  :  :  :  :  :  :  :  :  |  
|  :::::::::: :  :  :  :  :  :  :  :  :  :  :  |  
|  ::      :::::::::: :  :  :  :  :  :  :  :  |  
|  ::      :::::::::: :  :  :  :  :  :  :  :  |  
|  Framework for Developing Particle Simulator  |  
|  ::::::::::::::::::::::::::::::::::::::::::  |  
\\=====//
```

# サンプルコード nbody.cpp

```
int main(int argc, char *argv[]) {
    std::cout<<std::setprecision(15);
    std::cerr<<std::setprecision(15);

    PS::Initialize(argc, argv);
    PS::F32 theta = 0.5;
    PS::S32 n_leaf_limit = 8;
    PS::S32 n_group_limit = 64;
    PS::F32 time_end = 10.0;
    PS::F32 dt = 1.0 / 128.0;
    PS::F32 dt_diag = 1.0 / 8.0;
    PS::F32 dt_snap = 1.0;
    char dir_name[1024];
    PS::S64 n_tot = 1024;
    PS::S32 c;
    sprintf(dir_name, "./result");
    opterr = 0;
    while((c=getopt(argc,argv,"i:o:d:D:t:T:l:n:N:hs:")) != -1){
        switch(c){
            case 'o':
                sprintf(dir_name,optarg);
                break;
            case 't':
                theta = atof(optarg);
                std::cerr<<"theta =" << theta << std::endl;
                break;
        }
    }
```

FDPSの初期化

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::         ::         :         :         |
|  :::::::::: :         :         :         |
|  ::         :::::::::: :         :         |
|  ::         :::::::::: :         :         |
|  Framework for Developing |
|  Particle Simulator       |
|=====\\
```

# サンプルコード nbody.cpp

```
fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
}
```

```
PS::ParticleSystem<FPGrav> system_grav;
system_grav.initialize();
```

粒子群クラスの生成と初期化

```
PS::S32 n_loc = 0;
PS::F32 time_sys = 0.0;
if(PS::Comm::getRank() == 0) {
    setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
} else {
    system_grav.setNumberOfParticleLocal(n_loc);
}
```

```
const PS::F32 coef_ema = 0.3;
PS::DomainInfo dinfo;
dinfo.initialize(coef_ema);
dinfo.decomposeDomainAll(system_grav);
system_grav.exchangeParticle(dinfo);
n_loc = system_grav.getNumberOfParticleLocal();
```

```
#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif
```

```
PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
```

```
#ifdef MULTI_WALK
```

```
//=====\\
| |  :::::::::: :::::::::: :::::::::: :::::::::: | |
| |  ::      ::      :  :  :  :  :  :  :  :  :  : | |
| |  :::::::::: :  :  :  :  :  :  :  :  :  :  : | |
| |  ::      :::::::::: :  :  :  :  :  :  :  :  : | |
| |      :::::::::: :  :  :  :  :  :  :  :  :  : | |
| |      Framework for Developing | |
| |      Particle Simulator | |
\\=====//
```



# サンプルコード nbody.cpp

```
+printf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
}
```

```
PS::ParticleSystem<FPGrav> system_grav;
system_grav.initialize();
PS::S32 n_loc = 0;
PS::F32 time_sys = 0.0;
if(PS::Comm::getRank() == 0) {
    setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
} else {
    system_grav.setNumberOfParticleLocal(n_loc);
}
```

```
const PS::F32 coef_ema = 0.3;
PS::DomainInfo dinfo;
dinfo.initialize(coef_ema);
dinfo.decomposeDomainAll(system_grav);
system_grav.exchangeParticle(dinfo);
n_loc = system_grav.getNumberOfParticleLocal();
```

## ドメイン情報クラスの生成と初期化、領域分割

```
#ifndef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif

    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
    tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
#endif
MULTI WALK
```

```
//=====\\
||
||
||  :::::::::::  :::::::::::  :::::::::::  .::::::::::::.
||  ::          ::          :  ::          :  ::
||  :::::::::::  ::          :  :::::::::::  '  \  :::::::::::
||  ::          :::::::::::  ::          \  \  .....  '
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```



# サンプルコード nbody.cpp

```
fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
}

PS::ParticleSystem<FPGrav> system_grav;
system_grav.initialize();
PS::S32 n_loc = 0;
PS::F32 time_sys = 0.0;
if(PS::Comm::getRank() == 0) {
    setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
} else {
    system_grav.setNumberOfParticleLocal(n_loc);
}

const PS::F32 coef_ema = 0.3;
PS::DomainInfo dinfo;
dinfo.initialize(coef_ema);
dinfo.decomposeDomainAll(system_grav);
system_grav.exchangeParticle(dinfo);
n_loc = system_grav.getNumberOfParticleLocal();

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif
PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
#ifdef MULTI_WALK
```

粒子交換

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::      ::      :  ::      :  ::      |
|  :::::::::: :  :::::::::: :  :::::::::: |
|  ::      :::::::::: :  ::      :  ::      |
|  Framework for Developing |
|  Particle Simulator       |
|=====\\
```

# サンプルコード nbody.cpp

```
n_loc = system_grav.getNumberOfParticleLocal();

#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif
    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
    tree_grav.initialize(n_tot, theta, n_leaf_limit, n_group_limit);
#ifdef MULTI_WALK
    const PS::S32 n_walk_limit = 200;
    const PS::S32 tag_max = 1;
    tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP,
                                                RetrieveKernel,
                                                tag_max,
                                                system_grav,
                                                dinfo,
                                                n_walk_limit);
#else
    tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                       CalcGravity<PS::SPJMonopole>,
                                       system_grav,
                                       dinfo);
#endif
    PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
    calcEnergy(system_grav, Etot0, Ekin0, Epot0);
    PS::F64 time_diag = 0.0;
    PS::F64 time_snap = 0.0;
    PS::F64 time_1 = 0.0;
```

相互作用ツリークラスの  
生成と初期化

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::      ::      :      :      :      :      |
|  :::::::::: :      :      :      :      :      |
|  ::      :::::::::: :      :      :      :      |
|  ::      :::::::::: :      :      :      :      |
|  Framework for Developing |
|  Particle Simulator       |
|=====\\
```

# サンプルコード nbody.cpp

```

tag_max,
system_grav,
dinfo,
n_walk_limit);
#else
tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>,
                                   CalcGravity<PS::SPJMonopole>,
                                   system_grav,
                                   dinfo);
#endif
PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
calcEnergy(system_grav, Etot0, Ekin0, Epot0);
PS::F64 time_diag = 0.0;
PS::F64 time_snap = 0.0;
PS::S64 n_loop = 0;
PS::S32 id_snap = 0;
while(time_sys < time_end){
    if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
        char filename[256];
        sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
        FileHeader header;
        header.time = time_sys;
        header.n_body = system_grav.getNumberOfParticleGlobal();
        system_grav.writeParticleAscii(filename, header);
        time_snap += dt_snap;
    }
    calcEnergy(system_grav, Etot1, Ekin1, Epot1);
    if(PS::Comm::getRank() == 0){

```

相互作用関数を用いた計算

```

//=====
|
|  :::::::::: :::::::::: :::::::::: ..
|  ::      ::      ::      ::
|  :::::::::: ::      ::::::::::'
|  ::      ::::::::::' ::
|
|  Framework for Developing
|  Particle Simulator
|
\\=====

```

## 相互作用関数を用いた力の計算

```
//=====\\
||
||
||  ::::::::::  ::::::::::  ::::::::::  ::::::::::
||  ::          ::      :  ::          :  ::
||  ::::::::::  ::      :  ::::::::::  '  \  ::::::::::
||  ::          ::::::::::'  ::          \  .....'
```

# Framework for Developing Particle Simulator

```
\\=====\\
```



# サンプルコード nbody.cpp

```
PS::S32 id_snap = 0;
while(time_sys < time_end){
    if( (time_sys >= time_snap) || ( (time_sys + dt) - time_snap ) > (time_snap - time_sys) ){
        char filename[256];
        sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
        FileHeader header;
        header.time = time_sys;
        header.n_body = system_grav.getNumberOfParticleGlobal();
        system_grav.writeParticleAscii(filename, header);
        time_snap += dt_snap;
    }
    calcEnergy(system_grav, Etot1, Ekin1, Epot1);
    if(PS::Comm::getRank() == 0){
        if( (time_sys >= time_diag) || ( (time_sys + dt) - time_diag ) > (time_diag - time_sys) ){
            fout_eng << time_sys << " " << (Etot1 - Etot0) / Etot0 << std::endl;
            fprintf(stdout, "time: %10.7f energy error: %+e\n",
                time_sys, (Etot1 - Etot0) / Etot0);
            time_diag += dt_diag;
        }
    }
    kick(system_grav, dt * 0.5);
    time_sys += dt;
    drift(system_grav, dt);
    if(n_loop % 4 == 0){
        dinfo.decomposeDomainAll(system_grav);
        system_grav.exchangeParticle(dinfo);
    }
}
```

時間積分

```
//=====\\
|  :::::::::: :::::::::: :::::::::: :::::::::: |
|  ::      ::      :  :      :  :      :  :  |
|  :::::::::: :  :  :  :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  |
|  ::      :::::::::: :  :  :  :  :  :  :  |
|  Framework for Developing |
|  Particle Simulator      |
\\=====\\
```



# サンプルコード nbody.cpp

```
free_grav.CalcGravity<PS::SPJMonopole>(<CalcGravity<PS::SPJMonopole>,
                                        CalcGravity<PS::SPJMonopole>,
                                        system_grav,
                                        dinfo);

#endif
    kick(system_grav, dt * 0.5);
    n_loop++;
}
#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_close();
#endif
    PS::Finalize();
    return 0;
}
```

FDPS終了

# サンプルコード nbody.cpp

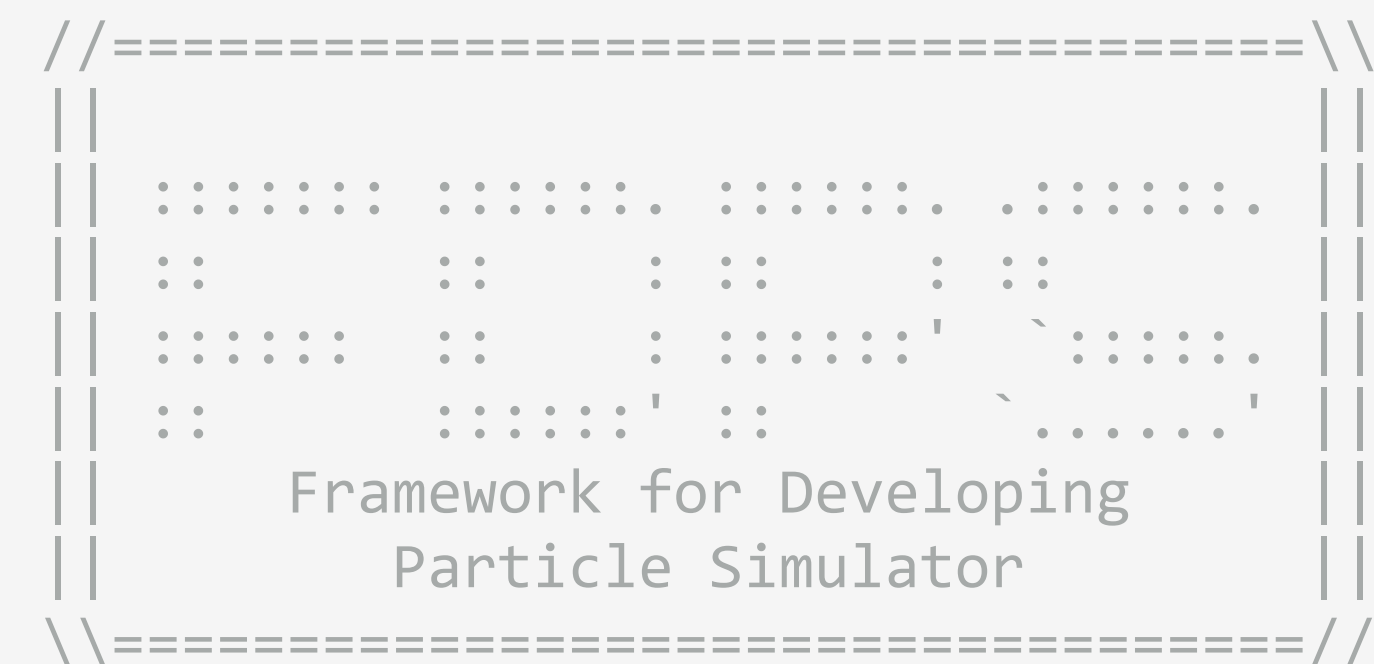
◆nbody.cppはこれだけ。  
おおよそ350行。

2020/08/19 FDPS講習会

```
//=====\\
||
||
||  .:.:.:.: .:.:.:.: .:.:.:.: .:.:.:.:
||  .:      .:      :  :      :  :
||  .:.:.:.: .:      : .:.:.:.' \ .:.:.:
||  .:      .:.:.:.' ::      \ .:.:.:.'
||
||      Framework for Developing
||      Particle Simulator
||
||=====\\
```

# 最後に

- ◆ユーザーが書かなくてはいけないのは、 $150+350 = 500$ 行 程度。  
この500行の中には、コマンドライン引数などの解析も含むため、  
そのようなものがなければ実際には更に少なくて済む。
- ◆コード内に、並列化を意識しなくてはならないような場所は無かった。  
つまり、コンパイルの方法だけでOpenMPやMPIを切り替えられる。



# 実習の流れ

◆実習用のFOCUSスパコンにログイン後、サンプルコードを

(1)並列化なし (2)OpenMP (3)OpenMP + MPI

の3パターンに関してコンパイルし、実行。

サンプルコードは以下の2つ。

- 重力 cold collapse
- 流体 (Smoothed Particle Hydrodynamics法) adiabatic sphere collapse

実行が終わったら、結果の解析

詳しくは、以下の講習会資料を参照

<https://github.com/FDPS/fdps-devel/blob/master/schools/text/resume.pdf>

