Marp

# Ownership in C++

Getting a handle on memory management in C++ (with Julia embedding)

# Ownership: a critical design pattern/concept in CS

Ownership is "Controling access to a managed resource".
Examples:

- Networks: proxy servers, ...
- C++: wrappers, pointers, proxies...


memory

# Example

```
struct S {
    int a = 0;
    double b = 0.;
}

my_struct* ptr1 = new S();
my_struct* ptr2(ptr1);
delete ptr1;
delete ptr2;  // double delete, UB
```

Who is responsible for releasing a shared resource?

# C++ and Julia need to "share" ownership of the soil matrix

The soil simulator is written in Julia. It evolves a 2D grid of cells representing "soil pixels".

The "Soil" needs to be shared between C++ and Julia e.g.:

# Example

```
Mesh soil_mesh{config};  // 3D mesh talos
Float* soil_array{soil_mesh};  // 2D array talos
talos.update() {
   timestep++;
   /** Julia Call **/
   soil_simulator.update(soil_array) {
       calculate forces;
       calculate angles;
       ...
       soil_array++;
   }
}
soil_mesh.update(soil_array)
talos.render(soil_mesh)
```

# Other considerations

Other considerations

- Representing the machine buckets
- Position of the buckets
- Parameters of the soil
- "Soil above the terrain"

# Julia garbage collector comes every day!

Here's the catch.

> By default, most memory allocated C++-side is incompatible with Julia.
> For example, a C++-side variable of type `char` is an 8-bit number, while a Julia-side `Char` has 32 bits.
> If we were to directly exchange memory between states, the value of the char would be corrupted.

# Example 1

dpxSoilJuliaInterface.hh::Line 328-329

```
/// @brief Julia shared data: Full terrain grid
public: jl_array_t *terrain_jl;
```

# Example 2

`dpxSoilJuliaInterface.cc::AllocTerrain(int terrain_size_x, int terrain_size_y)`

```
// Allocate the terrain
jl_value_t *two_array_type = jl_apply_array_type((jl_value_t *)jl_float64_type, 2);
this->terrain_jl = jl_alloc_array_2d(two_array_type, terrain_size_x, terrain_size_y);

double *terrain = (double *)jl_array_data(terrain_jl);
JuliaCatchException(__LINE__);

// Sharing terrain data between Julia and c++
return terrain;
```

# Smart pointers

`std::shared_ptr<T>`
`std::unique_ptr<T>`
`std::weak_ptr<T>`

implement the "Proxy" design pattern in C++ for sage management of resources.