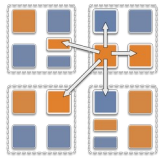
Asynchronous Methods

- Entry methods are invoked by performing a C++ method call on a chare's proxy

```
CProxy_MyChare proxy =
    CProxy_MyChare::ckNew(…constructor arguments…);

proxy.foo();
proxy.bar(5);
```

- The `foo` and `bar` methods will then be executed with the arguments, wherever the created chare, `MyChare`, happens to live
- The policy is ***one-at-a-time scheduling*** (that is, one entry method on one chare executes on a processor at a time)



PPL UIUC

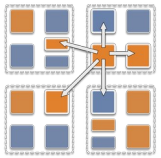
Asynchronous Methods

- Method invocation is not ordered (between chares, entry methods on one chare, etc.)!
- For example, if a chare executes this code:

```
CProxy_MyChare proxy = CProxy_MyChare::ckNew();
proxy.foo(); proxy.bar(5);
```

- These prints may occur in **any** order

```
MyChare::foo() {
   ckout << “foo executes” << endl;
}
MyChare::bar(int param) {
   ckout << “bar executes with ” << param << endl;
}
```



PPL UIUC

Asynchronous Methods

- If a chare invokes the same entry method twice

```
proxy.bar(7);
proxy.bar(5);
```

- These may be delivered in **any** order

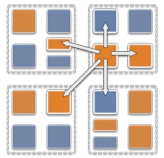
```
MyChare::bar(int param) {
   ckout << “bar executes with ” << param << endl;
}
```

- Output

```
bar executes with 5
bar executes with 7
```

or

```
bar executes with 7
bar executes with 5
```

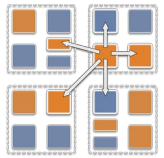


PPL UIUC

Asynchronous Example: .ci file

```
mainmodule MyModule {
    mainchare Main {
        entry Main(CkArgMsg *m);
    };

    chare Simple {
        entry Simple(double y);
        entry void findArea(int radius, bool done);
    };
};
```

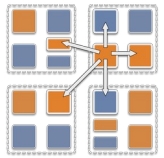


PPL UIUC

Does this program execute correctly?

```
struct Main : public CBase_Main {
   Main(CkArgMsg* m) {
      CProxy_Simple sim =
CProxy_Simple::ckNew(3.1415);
      for (int i = 1; i < 10; i++) sim.findArea(i,
false);
      sim.findArea(10, true); } };

struct Simple : public CBase_Simple {
   double y;
   Simple(double pi) { y = pi; }
   void findArea(int r, bool done) {
      ckout << “radius: ” << r << “Area:” << y*r*r <<
endl;
      if (done) CkExit(); } };
```



PPL UIUC

No! Methods are Asynchronous

- If a chare invokes several entry methods

```
sim.findArea(1, false);
...
sim.findArea(10, true);
```

- These may be delivered in ***any*** order

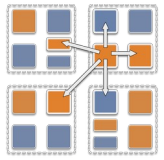
```
Simple::findArea(int r, bool done){
    ckout << "radius: " << r <<
    if (count++ = 10) CkExit(); }
    };
    if (done) CkExit(); } };
```

- Output:

```
radius: 254.34
radius: 200.96
radius: 28.26
radius: 3.14
radius: 12.56
radius: 153.86
radius: 50.24
radius: 78.50
radius: 314.00
```

or

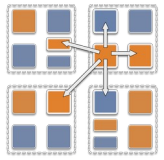
```
radius: 28.26
radius: 78.50
radius: 3.14
radius: 113.04
radius: 314.00
```



PPL UIUC

Readonly Variables

- The only global variables officially allowed in a Charm++ program are `readonly` variables
  - `readonly`s can be modified only in the constructor of the main chare
    - And from any functions called from them (not entry methods)
  - After the constructor finishes, before any other chares execute any methods, the `readonly` variables are available on all processors
- Declare a `readonly` in .ci file and also in the .cpp file
  - In .ci: `readonly int grainSize;`
  - In .cpp, just a normal declaration



PPL UIUC

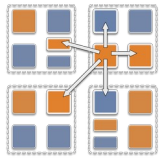
Data Types and Entry Methods

- You can pass basic C++ types to entry methods (int, char, bool)
- C++ STL data structures can be passed by including `pup_stl.h`
- Arrays of basic data types can also be passed like this:
- .ci file:

```
entry void foobar(int length, int data[length]);
```

- .cpp file

```
MyChare::foobar(int length, int* data) {
   // ... foobar code ...
}
```



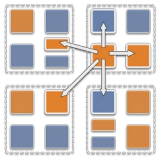
PPL UIUC

Exercise 1. Fibonacci Sequence

- Files under **exercises/ex1**

- **Goal**: Learn how to compile and run a Charm++ program
  - A simple, recursive program that computes a Fibonacci number
  - Detailed instructions provided in README

- Please let us know if you run into any problems!



PPL UIUC