MODULES AND LIBRARIES

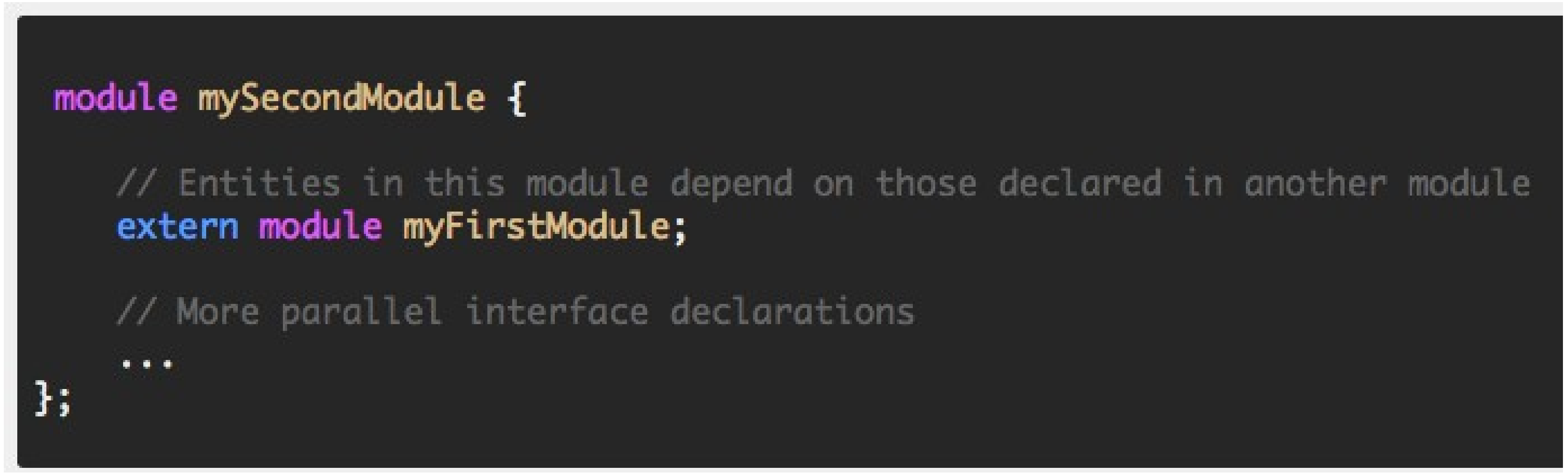
Modules and libraries

- So far, our programs had one .ci file, and one module in it

- Modular programming requires that we should be able to factor the program into multiple modules
- The final program is a composition of multiple modules
- In case of libraries, these are modules someone else wrote earlier that we want to use in our application
  - It may even be available as a (say, proprietary) binary
  - Of course, with the necessary header files

Multiple modules

- A program with multiple modules
  - Can be in a single .ci file, but typically, each module has its own .ci file
- You have to include in your .ci file, as extern, the module you plan to use



```
module mySecondModule {

   // Entities in this module depend on those declared in another module
   extern module myFirstModule;

   // More parallel interface declarations
   ...
};
```

Reachability

- Every module that you want to use must:
  - Either : be included via a chain of “extern module” commands starting from the mainmodule
  - Or: be listed in a “–module modulename” phrase as a link-time option.

Separately compiled libraries

- The decl.h file of the imported (library) module must be provided by the library writer to the application (i.e. importing module)
  - Along with a .h file, as usual in sequential programs
- But the .C file doesn't need to be provided.

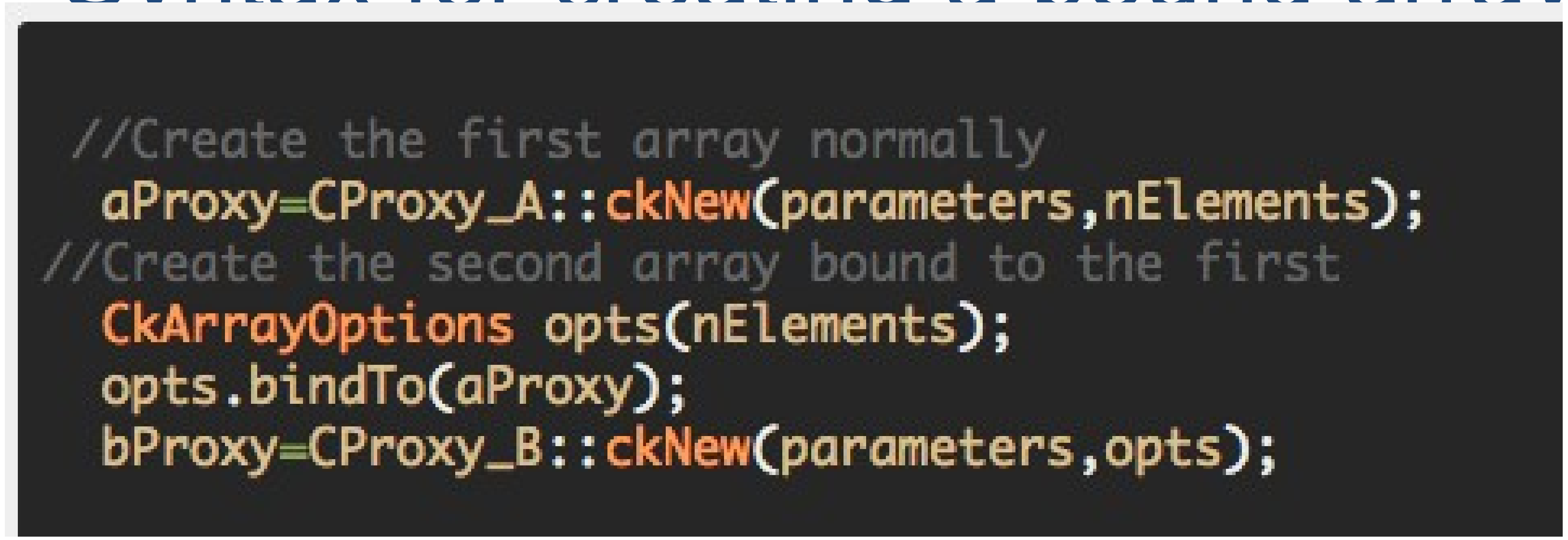
Matching size of the library array

- Suppose you want to use a library for sorting elements in a chare array
  - The client (your application) has a 1D chare array App of size N
  - You want the library to also have a 1D array of the same size
  - You also want the correspondng elements to be on the same processor
  - So: you can hand over your elements to the corresponding element of the library array locally, and get back the sorted result from it

Bound arrays

- You can bind one chare array to another
  - That means: the corresponding elements of the 2 arrays live on the same processor.
  - If the “parent” array elements migrates, any array element bound to it also migrates with it
  - You can make regular (without proxy) sequential method calls between the corresponding elements (via ckLocal() call)

Syntax for creating a bound array



```
 //Create the first array normally
  aProxy=CProxy_A::ckNew(parameters,nElements);
//Create the second array bound to the first
  CkArrayOptions opts(nElements);
  opts.bindTo(aProxy);
  bProxy=CProxy_B::ckNew(parameters,opts);
```

- We use the “options” we learned before
- The bound array (sometimes called the shadow array) is created *after* the parent array is created.
- The chare types of the two arrays can be different
  - Typically, they are different

Using bound arrays in libraries

- To use a parallel "sort" library,
  - you create your 1D chare array,
  - Pass its proxy (in your main chare) to the initialization call of the sort library
  - Which will create its array bound to your array

ckLocal()

- If a chare is on the same processor as you,
  - you can get a (regular, C++) pointer to it, and
  - invoke methods on it directly
  - (or even access its public data members)
- How?
  - x = A[i].ckLocal()
  - A is a proxy to a 1D chare array
  - x gets pointer to a C++ object
  - What if the A[i] is not on your processor?
    - This call returns NULL