MODULES AND LIBRARIES

Example of library: data-balancing with prefix

- Remember in our MP1 (from the course cs598lvk)
  - (data balancing using parallel prefix sum),
  - We used parallel prefix to calculate which data goes to which processor.
- Ideally, the prefix should have been a separate library
  - After all it has many other uses
- Instead, we just merged the application code and prefix algorithm in one module
- Now, let us do that properly

How to write prefix sum as a library

- What is the interface we want:
  - It gets initialized as a bound array to the application's array
    - Meant to be used with 1D chare array as clients (caller, application)
    - Regular function call.. Say: `prefixInit(clientArrayProxy)`
  - Each time we need to calculate prefix sum, the library should be called via a local call with the local value
    - We assume it can be called multiple times
    - Assuming a single-value prefix sum. We can generalize to vecs
  - The call won't return, but later on, another entry method of the application will called
    - (Why not a function call?)
    - How to make prefix call application's entry method?
      - Remember when prefix library is written, it doesn't know about any application that's using it
      - So, can't use any name from the application
      - Callback! That's flexible. Called can be threaded, or sdag, or…

Initialization and usage

- Initialization: From the main chare of the application,
  - after creating the application’s 1D array (that will be the caller), with proxy in A
  - Call function prefixInit(A)
    - This will create a bound chare array in prefixLib, and return its proxy, say *lib*
- Call: (this can be made multiple times)
  - libPtr = lib[thisIndex].ckLocal();
  - libPtr->computePrefix(myVal, callBack);
- I know ckLocal() won’t return NULL: its bound to me

What to do when you migrate?

- With a bound array, you can be sure that the “shadow” chare in prefix library will migrate with you.
- But the pointer will be different
- If you use it via ckLocal() as in previous slide, there is no problem
- But, if you have stored the pointer, you have to re-obtain it
  - That is where the migration constructor is useful
    - App::App(CkMigrateMessage *m)
      - { libPtr = lib[thisIndex].ckLocal() ; }
    - or you can use just after AtSync(), but this may do it unnecessarily, because you might not have migrated

Exercise

- Make the above design work for your MP1 prefix-based data balancing
- For better understanding, do this in 2 steps
  - First, write a program with 2 modules in the same folder
    - Make sure prefixLib doesn’t use any name from dataBalancer
  - Next, do it with both modules in separate folders, AND precompile the library..
    - What information about prefixLib do you have to provide to the dataBalancer application ? In what form? .h? .ci? prefixLib.decl.h? What information is minimal necessary? What documentation should you provide?