A Proxy class is generated for each chare class
- The runtime needs to unpack data and figure out where the chare class is
- Functions exist to communicate and handle chare class data
- chareProxy.entryMethod() is a function that broadcasts data. Without a subscript it broadcasts all the data
- contribute()
  - This is a reduction function
  - It can have no arguments or have these arguments: contribute(int bytes, const void *data, CkReduction::reducerType type)
- The output for reduction goes into a callback object
  - CkCallback* cb = new
    CkCallback(CkIndex_myType::myReductionFunction(NULL), thisProxy);
  - contribute(bytes, data, reducerType, cb);
- The reduction data is processed by the reduction function
  - void myType::myReductionFunction(CkReductionMsg *msg) {
    int size = msg->getSize() / sizeof(type);
    type *output = (type *) msg->getData();
    ...
  }
- Load Imbalance
  - This is when work is unequally distributed across processes
  - Calculated with: max load / mean load
- Load Balancing is the process of correcting this
- You have to decide when load balancing is really appropriate because it also has an overhead
- Static Load Balancing: Managing the initial load distribution
- Dynamic Load Balancing: Managing load distribution over time
- Centralized Load Balancing: All the data is collected into one process with a global view and then the work is distributed
- **Distributed Load Balancing**: Each process knows and manages the load of n of its neighbors.

- **Hybrid/Heirarchical Load Balancing**: Combines both the strategies.

- **Computional Load, Communication Load, and the Communication Graph is used in load balancing.**

- **Load Balancing Goals:**
  - Bring the process with the maximum load close to the average load
  - Minimize data migration

- **Greedy Strategy for Load Balancing:**
  - Sort the processes by load and then take load from the heaviest process and assign it to the lightest

- **Work Stealing**: A process takes load from nearby processes when it has completed its task.