

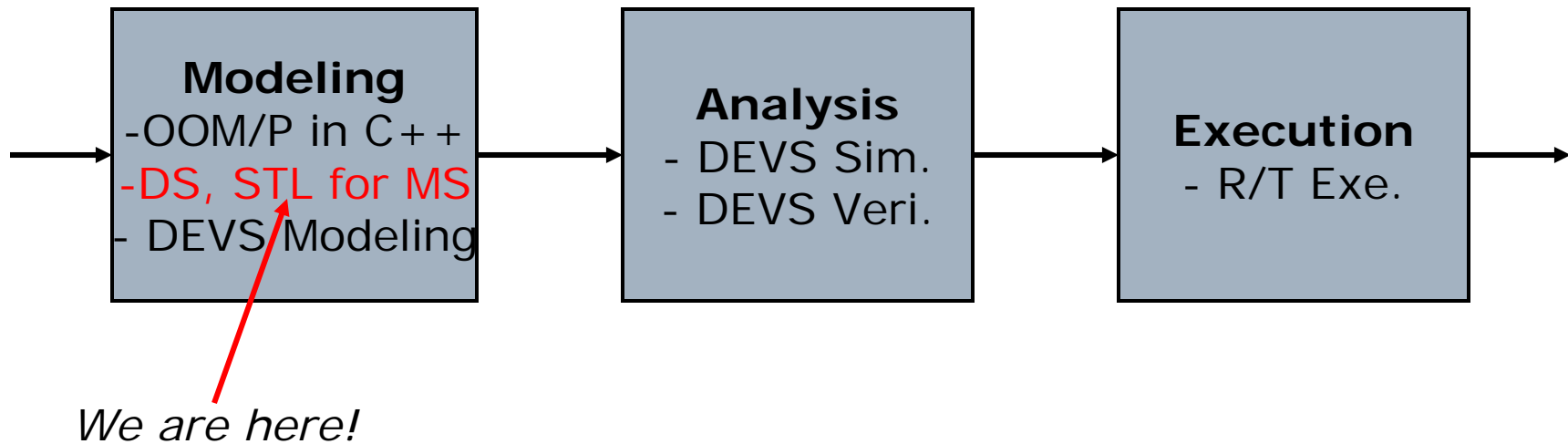
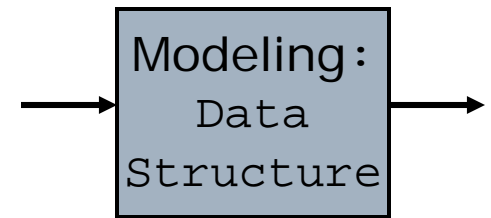
ECE575/ Chapter 3. Data Structure in C++ for Modeling and Simulation of Discrete Event Systems

Part1: System Modeling

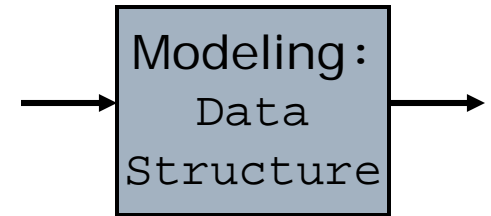
Moon Ho Hwang
mhhwang@ece.arizona.edu

2006 Fall
ECE Department,
University of Arizona

Where are we now?

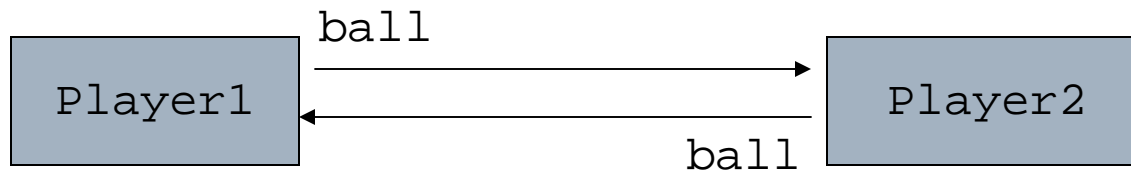


Ex: Ping-Pong Game

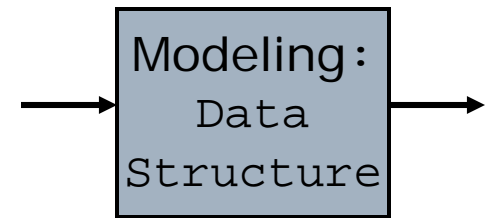


□ Similar Examples:

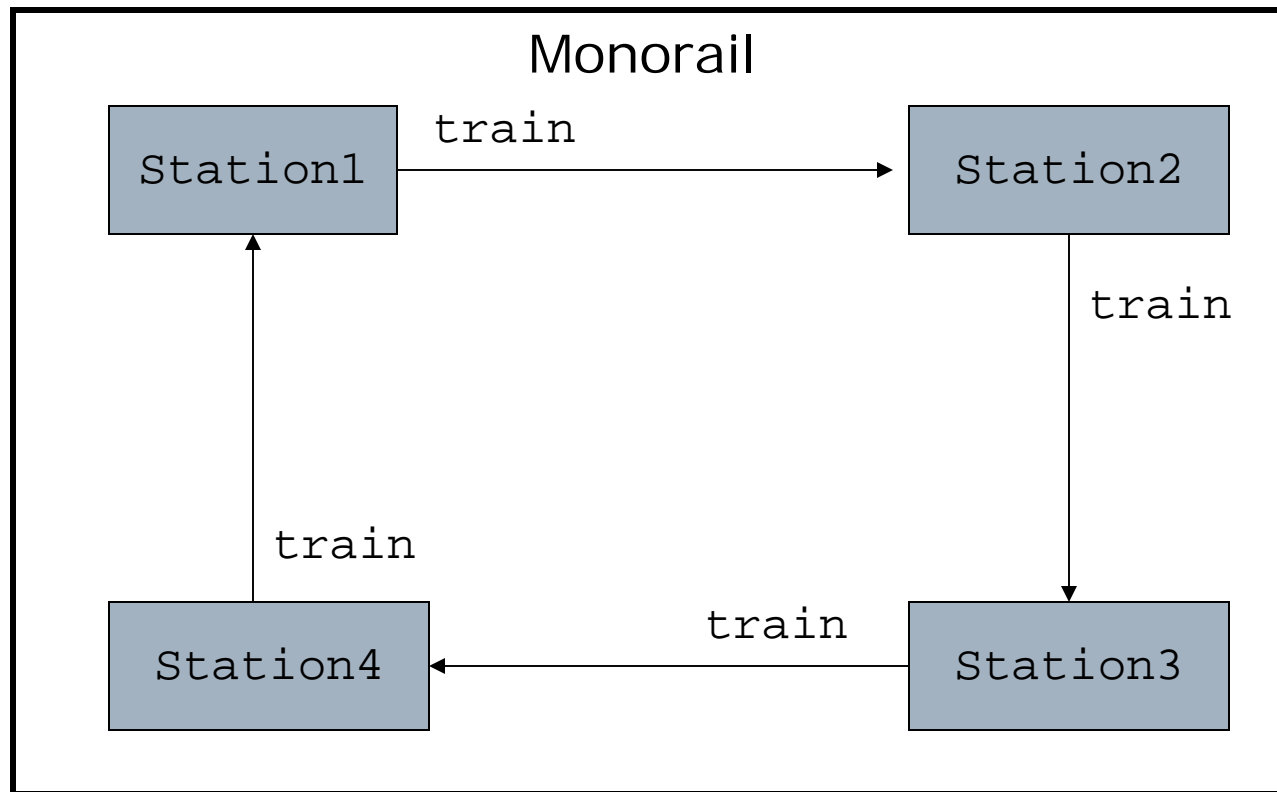
- tennis, communication protocol



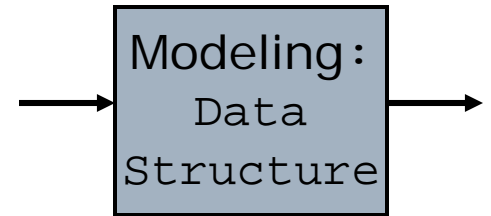
Ex: Monorail



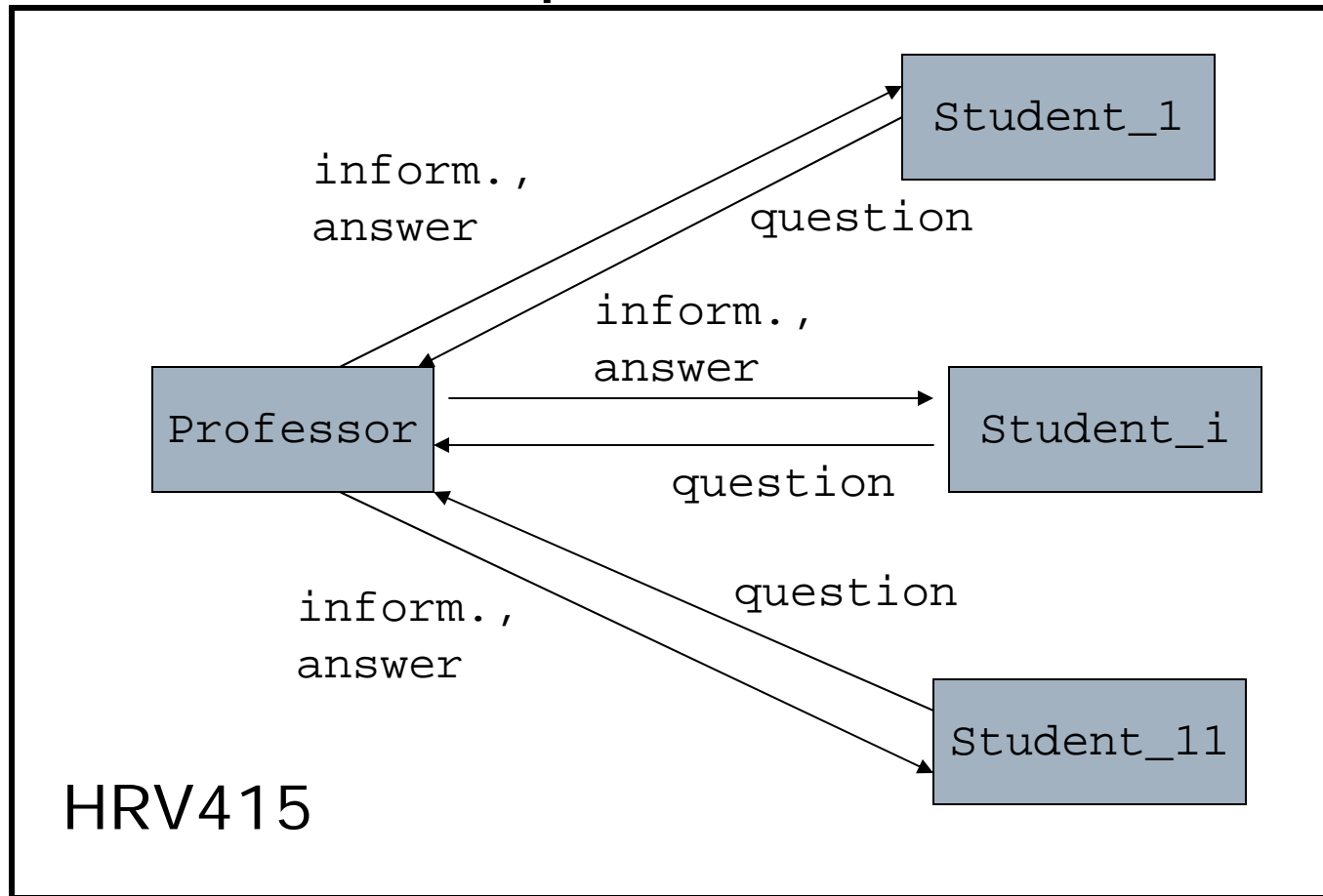
- Similar Examples:
 - Token ring communication



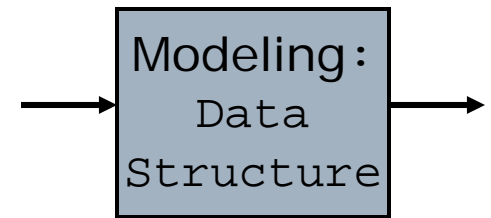
Ex: HARV415 of ECE575



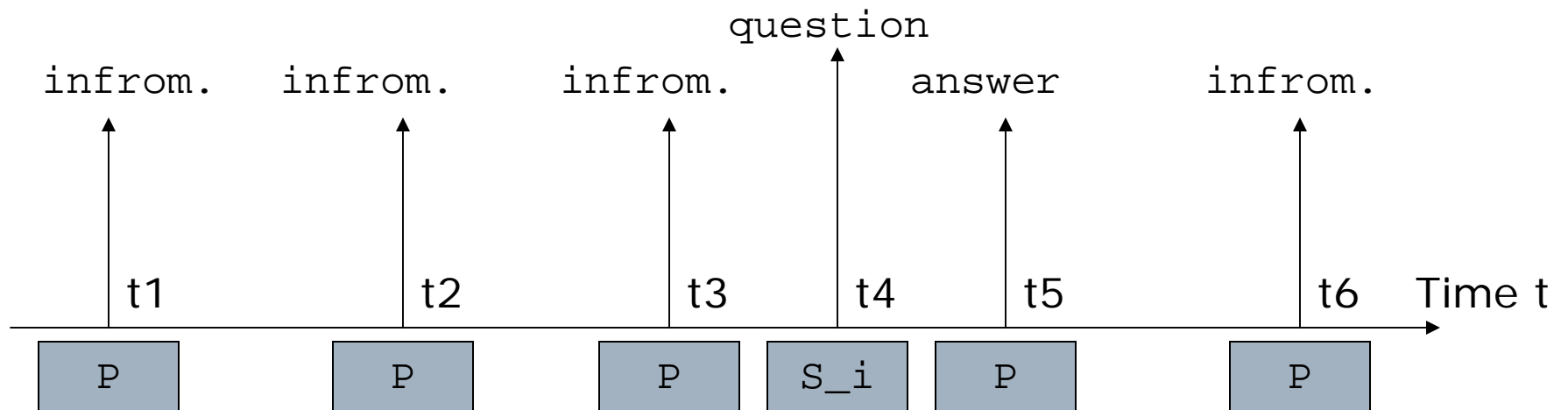
□ Similar Examples: Server and Client



Lecture Room of ECE575



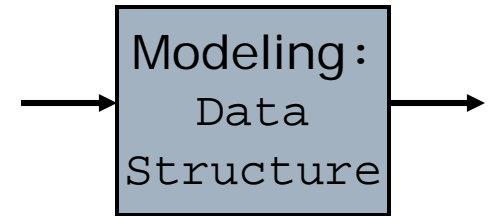
□ Dynamic Model



P: Professor

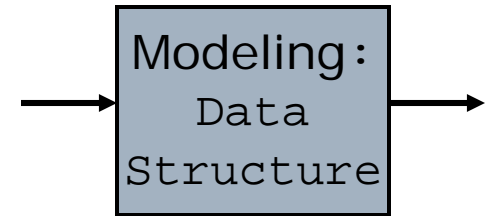
S_i: Student i

What we need to do

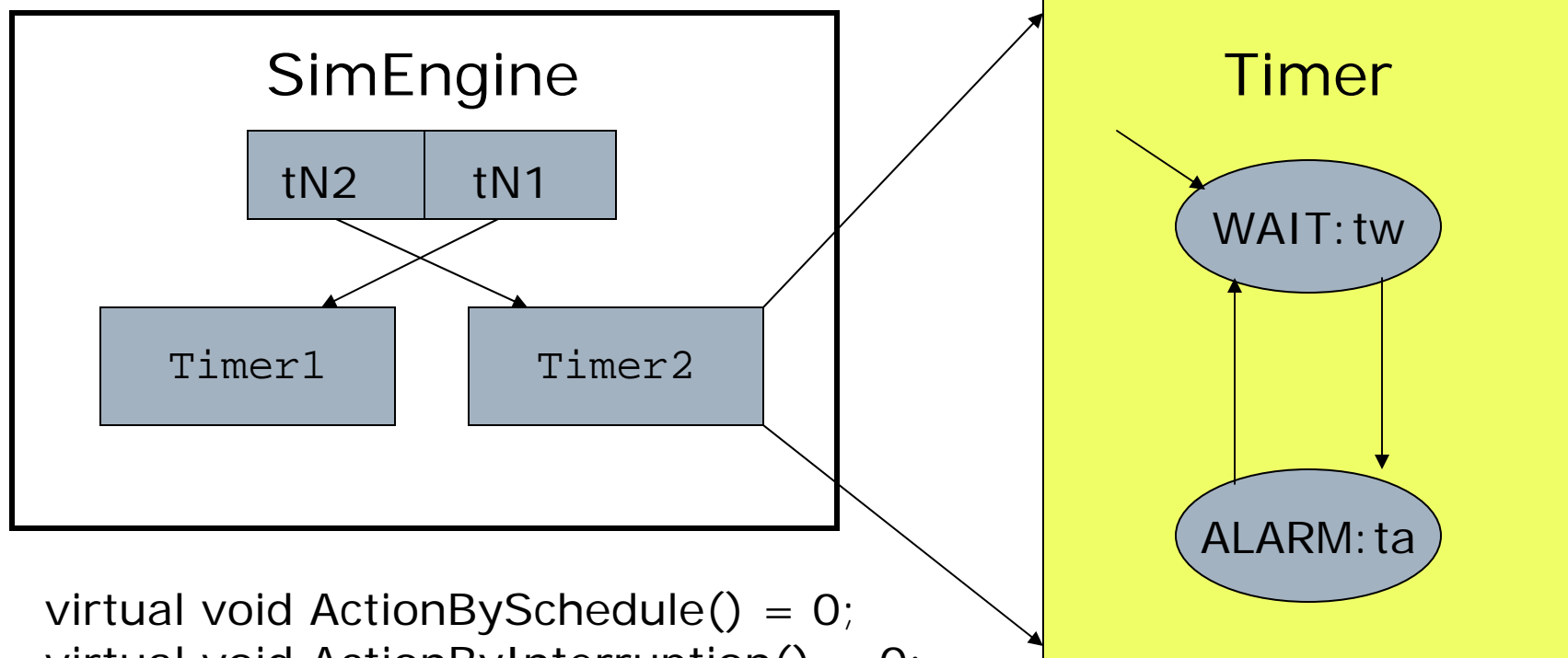


- Static Modeling
 - Define relations of sender/receiver
- Dynamic Modeling
 - Ordering Objects (or Events) by Schedule
 - Message Synchronization

Simple Ex: Two Timers

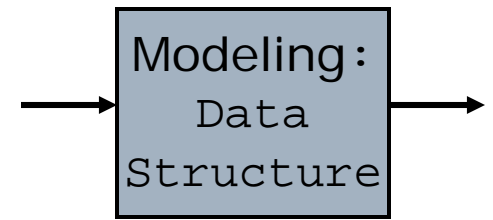


- See `TimerManager` Example Project

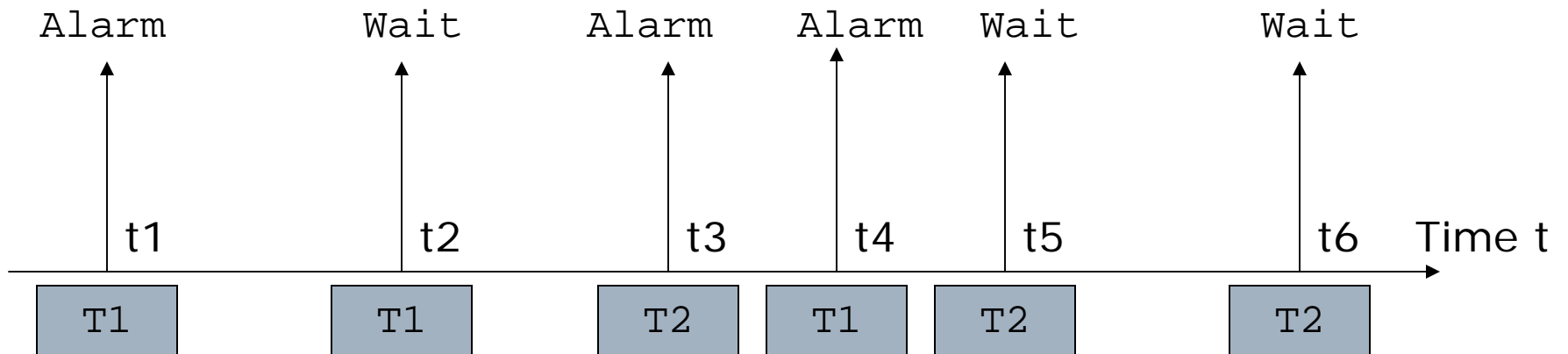


```
virtual void ActionBySchedule() = 0;  
virtual void ActionByInterruption() = 0;  
virtual double LifeTime() const = 0;
```

Two Timers



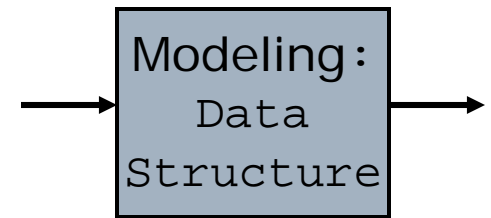
□ Dynamic Model



P: Professor

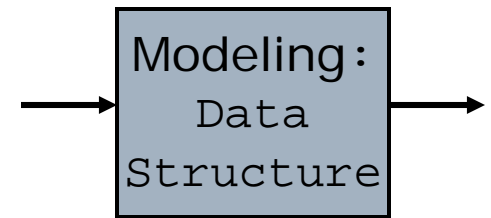
S_i: Student i

Data Structure and Algorithm



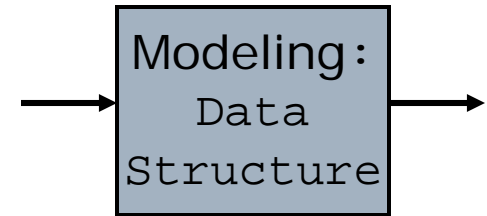
- What is Data Structure and Algorithm?
 - Data Structure: *A set of Container classes*
 - Algorithm: Their associated *efficient functions*
 - $O(\)$: *Big-Oh: worst case performance*
- What are the Containers?
 - The classes which are commonly used for *storing, retrieving and updating* data.

Well-known Containers



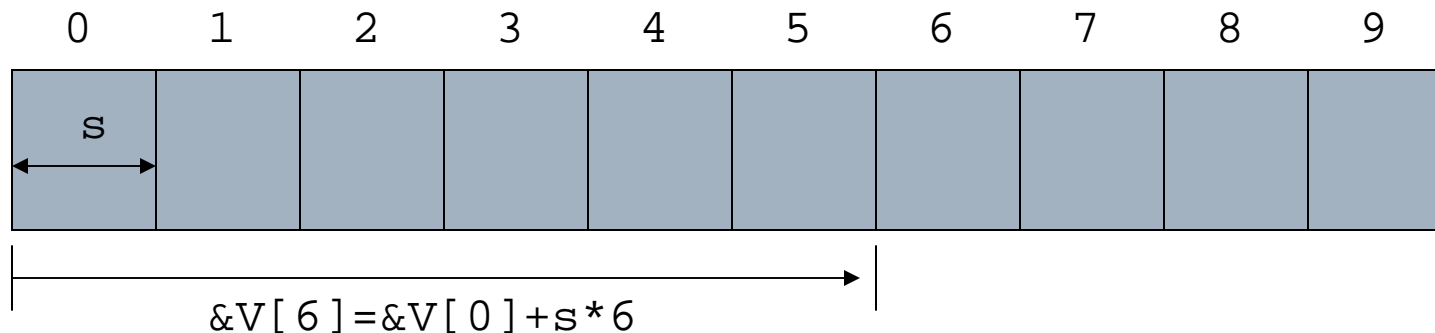
- ❑ Vector: *consequent* memory array
- ❑ Linked List: not-necessary consequent memory areas but they are *linked*
- ❑ Ordered Set: whose elements are ordered. (usually a *tree structure* are used for it)
- ❑ Map: A set of (*key, value*)
 - Ordered Map: Pairs are *ordered* by key
 - Hash Table: key is not ordered but *mapped* to index.

Vector

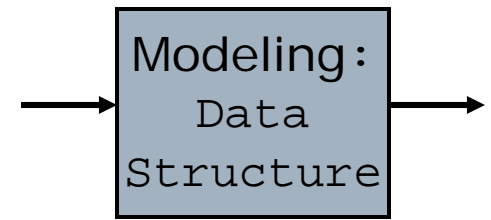


□ Features

- consecutive (resizable) memory area;
- **find** operation: $O(n)$
- **insert**, **remove**, **resize** operations: $O(n)$
- **indexing** operation: $O(1)$

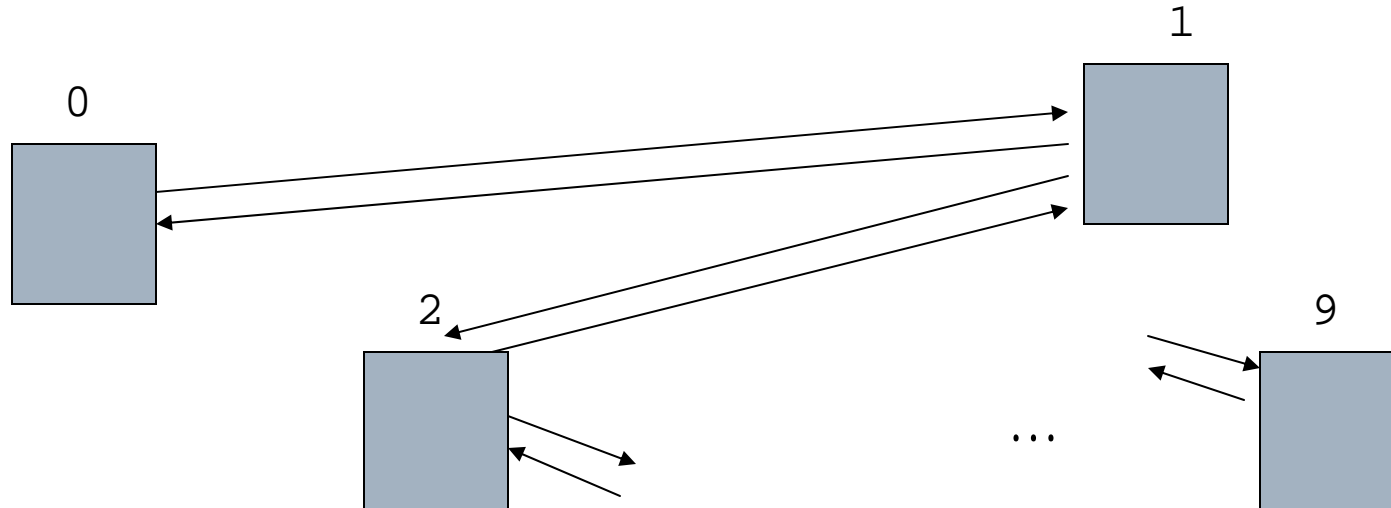


(Linked) List

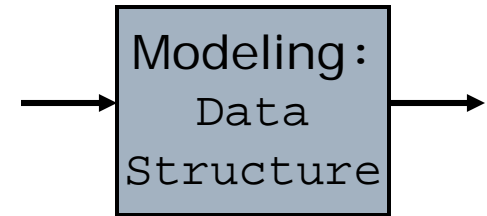


□ Features

- linked memory area;
- **find** operation: $O(n)$
- **insert**, **remove** operations: $O(1)$

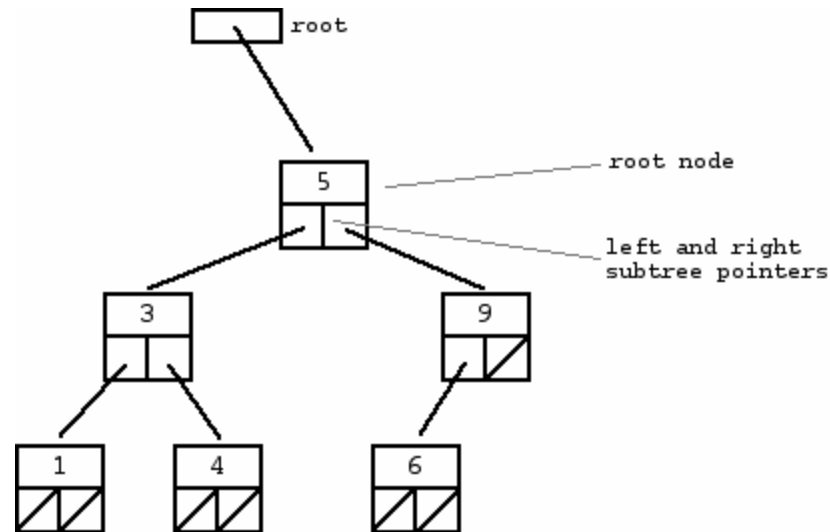


Ordered Set



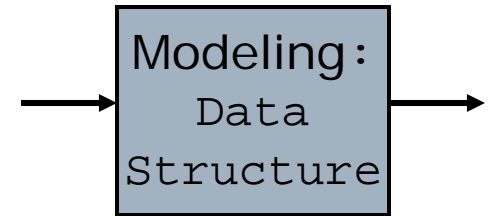
□ Features

- Tree for ordering;
- **find** operation: $O(\log(n))$
- **insert, remove** operations: $O(\log(n))$

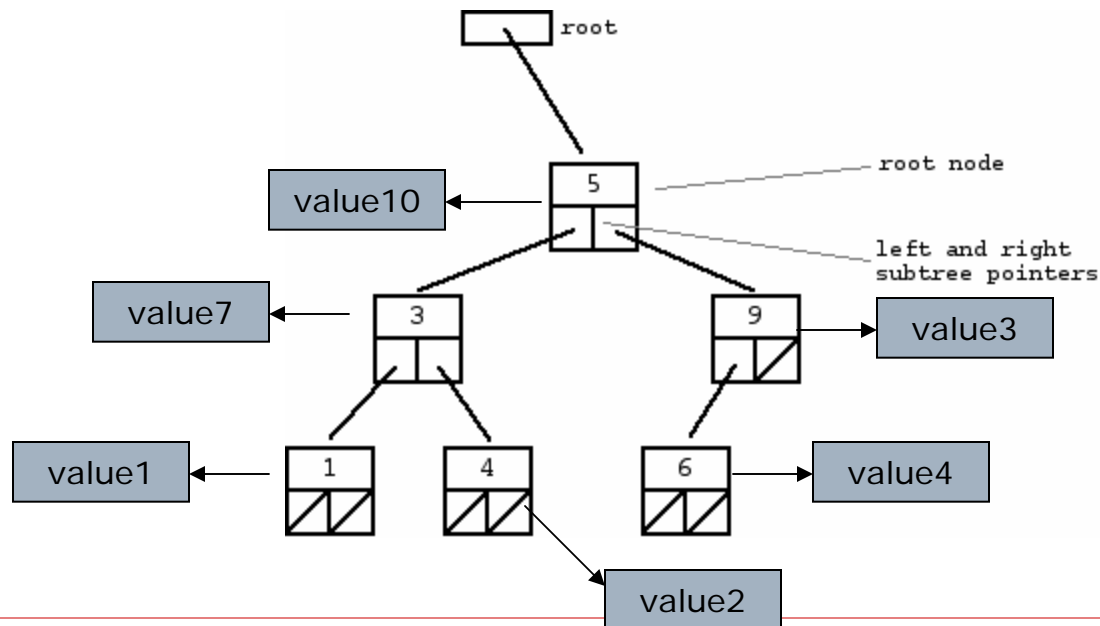


Copied from <http://cslibrary.stanford.edu/110/BinaryTrees.html>

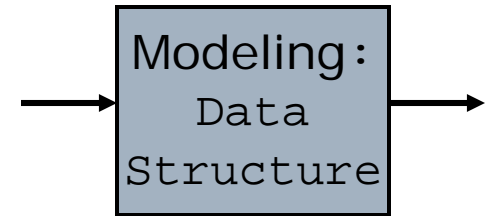
Map



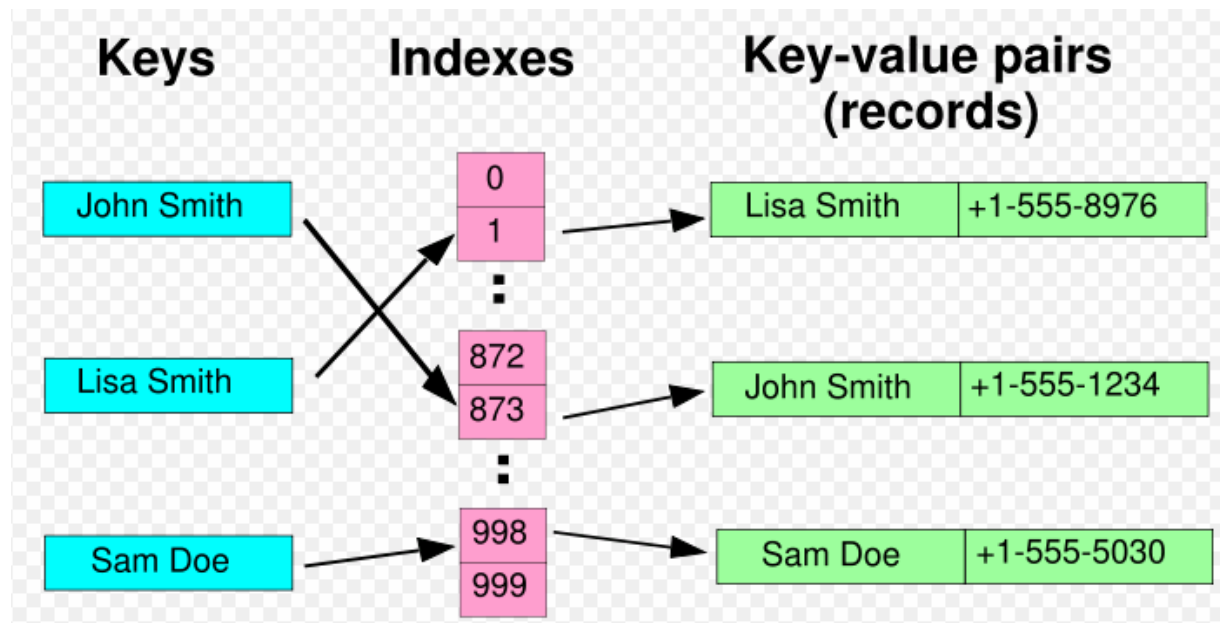
- Ordered Map: pairs of (key, value) are ordered by their **key**
 - Using tree like Ordered Set but ordering by key.



Map (cont.)



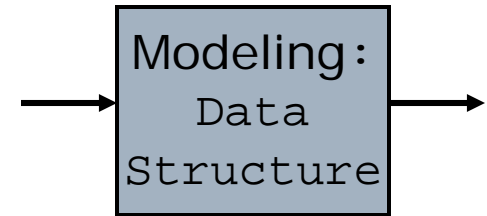
- Hash Map (Table): (key, value)
 - key mapped to the *finite* indexes pointing associated values.



Phone Book

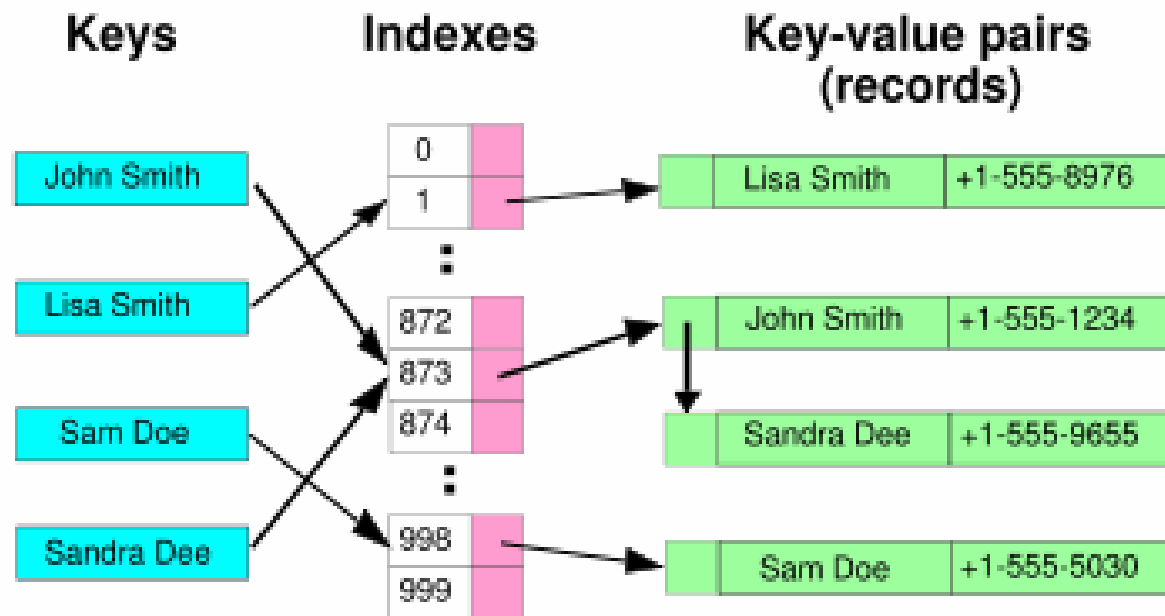
Copied from <http://www.wikipedia.com>

Hash Map (cont.)



Collision Resolution

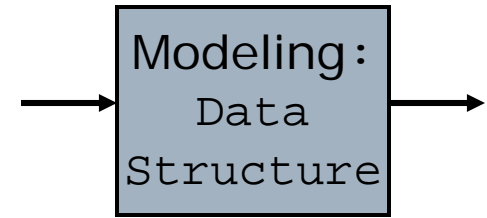
- **find**: normally $O(1)$; rare-worst case can be $O(n)$;
- **insert, remove**: the same as **find**.



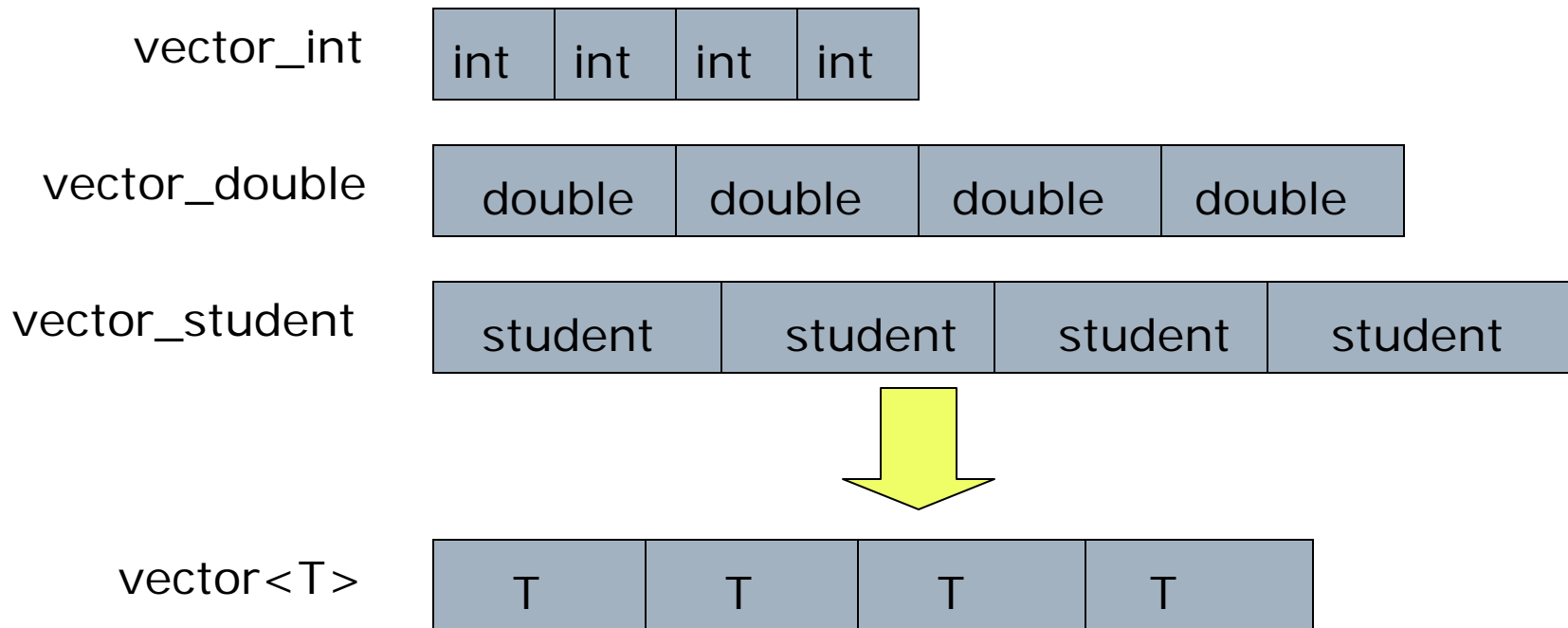
Collision Resolution by chaining

Copied from <http://www.wikipedia.com>

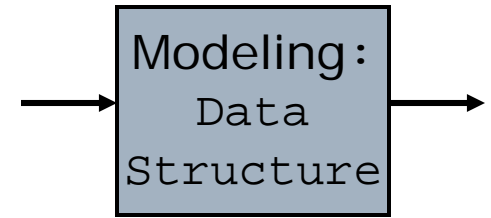
Template Class



- A class defined by *arguments*

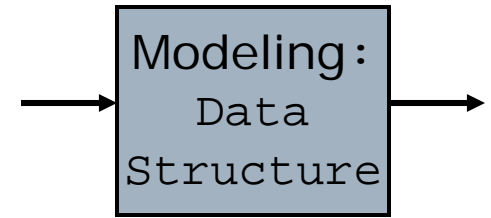


Template Class (cont.)



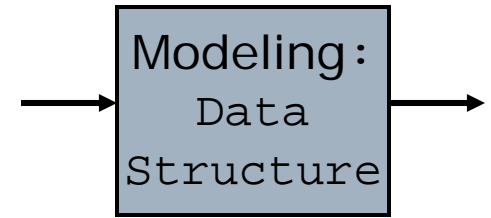
- For example, a standard template library provides `vector<T>`.
- The user can make the following instances:
 - `vector<int> a;`
 - `vector<double> b;`
 - `vector<Student> c;`
 - `vector<Student*> d;`

Standard Template Library (STL)



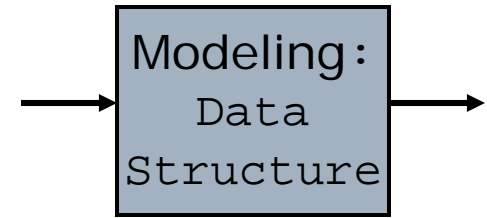
- A template container library which is a standard in C++.
- Consists of
 - Containers
 - Iterators
 - Algorithms
 - Functors (Function Objects)
- Reference: <http://www.sgi.com/tech/stl/>

STL Containers



- ❑ `vector<T>`
- ❑ `list<T>`
- ❑ `deque<T>`: double ended queue.
- ❑ `set<T, It>`: sorting elements by `It`.
- ❑ `multiset<T, It>`: same as a set but allows duplicate elements.
- ❑ `map<key, value, It>`: pair of (key, value) sorting key by `It`.
- ❑ `multimap<key, value, It>`: same as a map but allows duplicate keys.

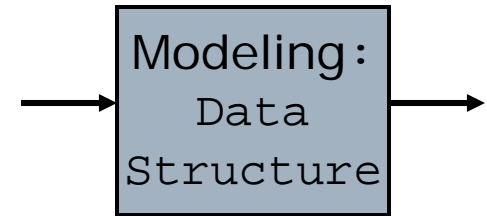
Iterators



- An iterator allows to enumerate every elements in a container.
- Example:

```
vector<Student*> ECE575S;  
for( vector<Stuent*>::iterator it = ECE575S.begin() ;  
      it != ECE575S.end(); it++ )  
{  
    (*it)->age++;  
}
```

Iterators (cont.)

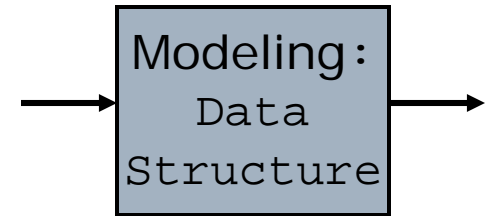


- ❑ In a **constant** function or for a **constant** container, we should use `const_iterator` rather than `iterator`.

- ❑ Example:

```
void print_out(const vector<Student*>& S)
{
    for( vector<Stuent*>::const_iterator
        it = S.begin(); != S.end(); it++ )
    {
        cout << (*it)->Name << endl;
    }
}
```

Iterators (cont.)



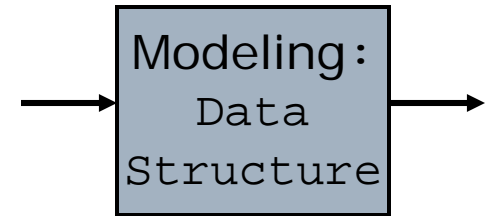
- All containers of STL have the elements in the range of iterators:

[begin(), end()).

- end() can be also used for checking if every container has an element or key:

```
vector<int> a;  
a.push_back(1);  
a.push_back(5);  
assert(a.find(2) == a.end());
```

Iterators (cont.)



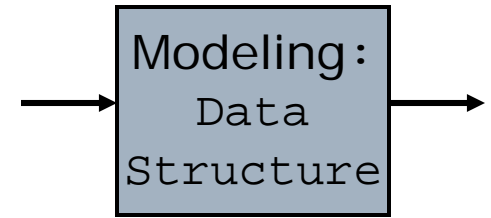
- ❑ Associate Containers such as `multiset` and `multimap` has `lower_bound` and `upper_bound` functions.

- ❑ For example,

```
typedef multimap<string,string> StrStrMMap;  
  
// create empty dictionary  
StrStrMMap dict;  
  
...  
// print all values for key "smart"  
string word("smart");  
cout << word << ": " << endl;  
for (pos = dict.lower_bound(word);  
     pos != dict.upper_bound(word); ++pos) {  
    cout << "    " << pos->second << endl;  
}
```

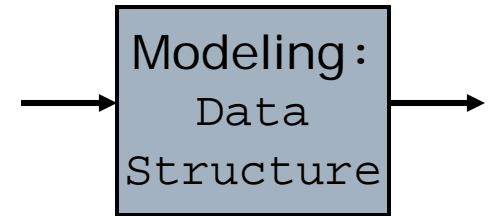
- ❑ See `multimap` Example.

Algorithms



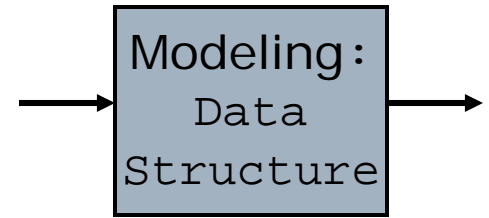
- Non-mutating:
 - `for_each`, `find`, `count`, ...
- Mutating:
 - `copy`, `swap`, ...
- Sorting:
 - **`sort`**, ...
 - See `sorting` and `TimerManager` Examples.

Applications of **sort** Algorithm



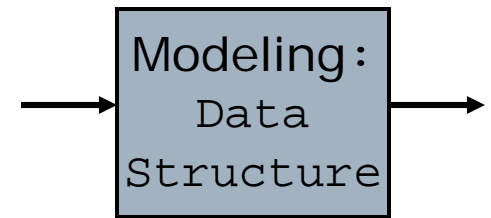
- Schedule List
 - Sorting by next events
- Service List of Clients
 - Sorting by arrival times
 - Sorting by remaining times
 - Sorting by client classes
- Performance: Binary Tree Sort:
 $O(n \log n)$

Function Objects (Functors)



- ❑ Functors provide a kind of variety with an algorithm.
- ❑ That is, even with an identical algorithm, different functors generate different results.
- ❑ See `Functors` and `TimerManager` Examples.

Summary



- Reviews of Containers and their associated algorithms.
- Template class and STL in terms of
 - Containers: `TimerManager` and `multimap`
 - Iterators: `TimerManager` and `multimap`
 - Algorithms: `TimerManager` and `sorting`
 - Functors: `TimerManager` and `Functors`
- We will see more examples when going through `ODEVS` and `VeriRTS` codes.