2.向量

（a）接口与实现

邓俊辉

deng@tsinghua.edu.cn

Abstract Data Type vs. Data Structure

❖ **抽象数据类型** = 数据模型 + 定义在该模型上的一组操作

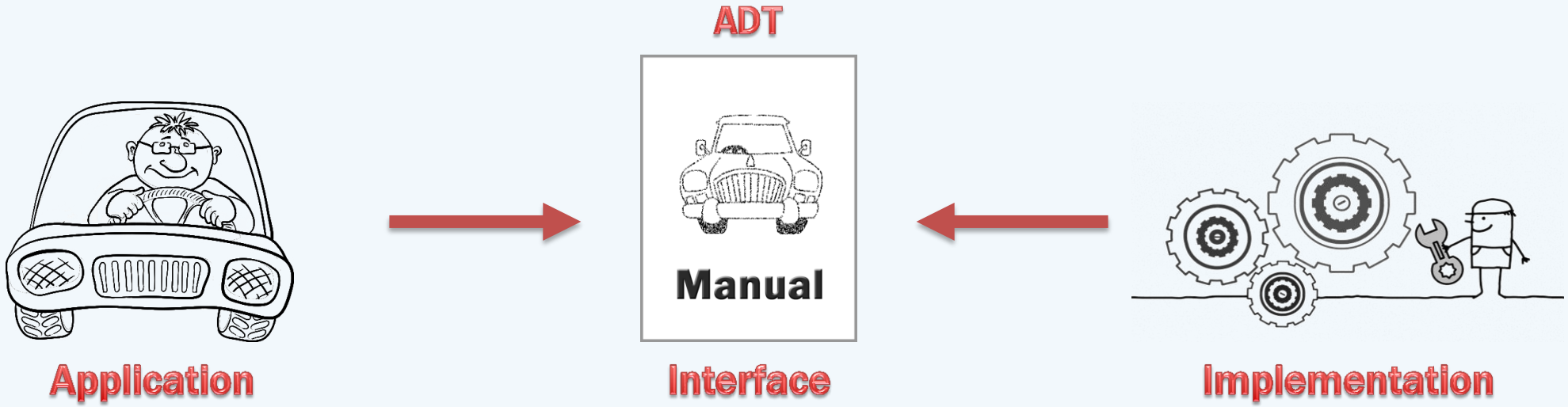
抽象定义 外部的逻辑特性 操作&语义

一种定义 不考虑时间复杂度 不涉及数据的存储方式

数据结构 = 基于某种特定语言，实现ADT的一整套算法

具体实现 内部的表示与实现 完整的算法

多种实现 与复杂度密切相关 要考虑数据的具体存储机制



Application = Interface x Implementation

- 在数据结构的具体实现与实际应用之间，ADT就分工与接口制定了统一的规范

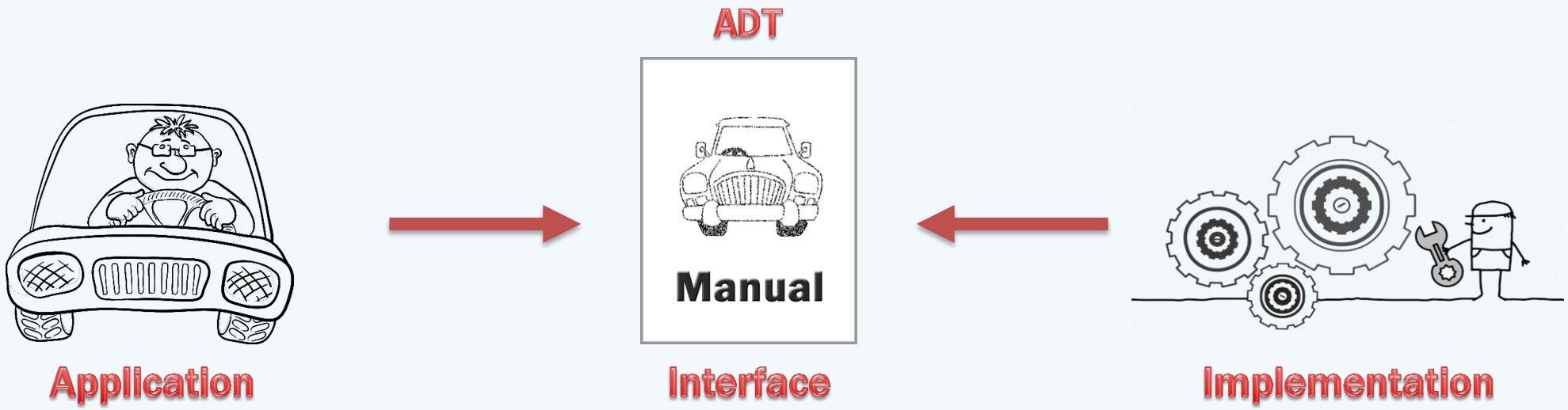
  实现：高效率地兑现数据结构的ADT接口操作 //做冰箱、造汽车

  应用：便捷地通过操作接口使用数据结构 //用冰箱、开汽车

- 按照ADT规范：高层算法设计者与底层数据结构实现者可高效地分工协作

  不同的算法与数据结构可以任意组合，便于确定最优配置

  每种操作接口只需统一地实现一次，代码篇幅缩短，软件复用度提高



从数组到向量

❖ C/C++语言中，数组A[]中的元素与[0, n)内的编号一一对应

A[0], A[1], A[2], ..., A[n-1]

❖ 反之，每个元素均由（非负）编号唯一指代，并可直接访问

A[i]的物理地址 = A + i×s，s为单个元素占用的空间量

故亦称作线性数组（linear array）

❖ 向量是数组的抽象与泛化，由一组元素按线性次序封装而成

各元素与[0, n)内的秩（rank）一一对应　　//循秩访问（call-by-rank）

操作、管理维护更加简化、统一与安全

元素类型可灵活选取，便于定制复杂数据结构　　//Vector<PFCTree*> pfcForest;

向量ADT接口

操作	功能	适用对象
size()	报告向量当前的规模（元素总数）	向量
get(r)	获取秩为r的元素	向量
put(r, e)	用e替换秩为r元素的数值	向量
insert(r, e)	e作为秩为r元素插入，原后继依次后移	向量
remove(r)	删除秩为r的元素，返回该元素原值	向量
disordered()	判断所有元素是否已按非降序排列	向量
sort()	调整各元素的位置，使之按非降序排列	向量
find(e)	查找目标元素e	向量
search(e)	查找e，返回不大于e且秩最大的元素	有序向量
deduplicate(), uniquify()	剔除重复元素	向量/有序向量
traverse()	遍历向量并统一处理所有元素	向量

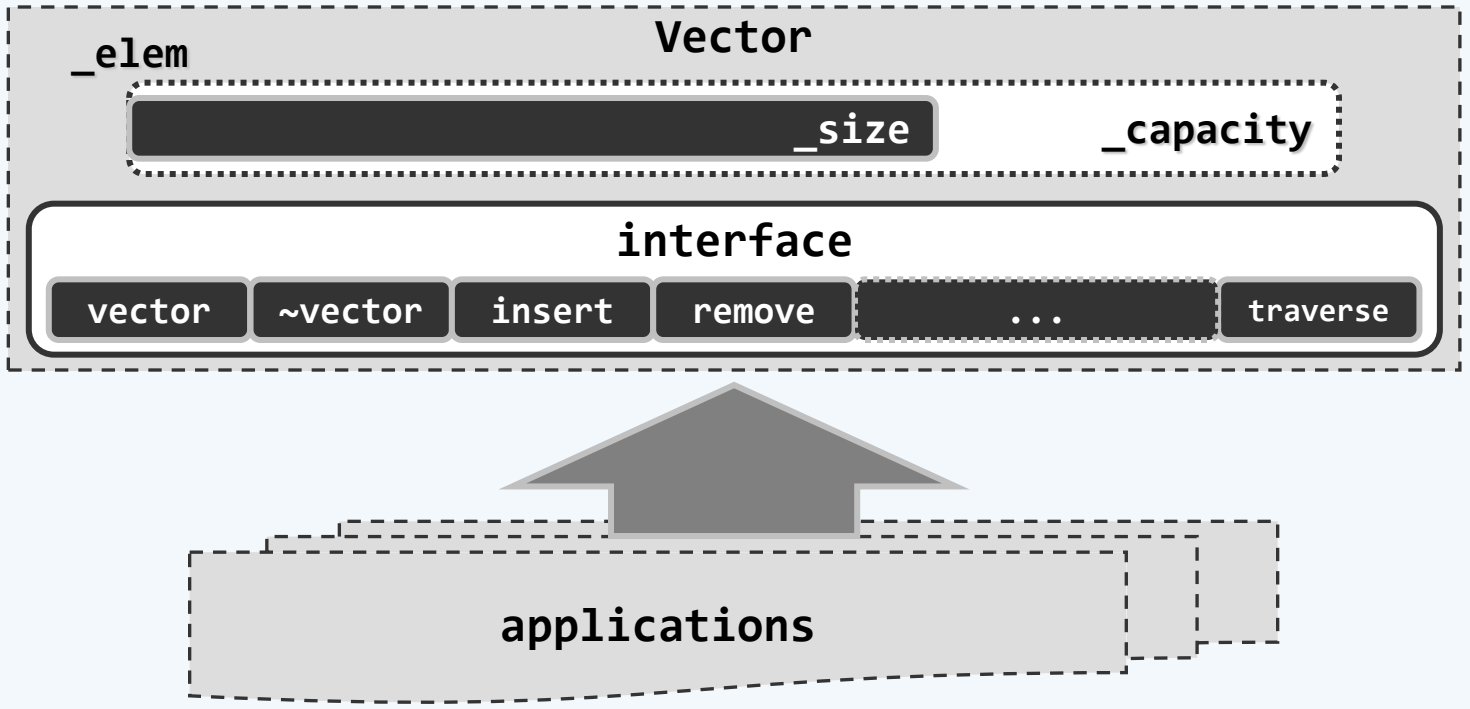
ADT操作实例

操作	输出	向量组成（自左向右）
初始化		
insert(0, 9)		9
insert(0, 4)		4 9
insert(1, 5)		4 5 9
put(1, 2)		4 2 9
get(2)	9	4 2 9
insert(3, 6)		4 2 9 6
insert(1, 7)		4 7 2 9 6
remove(2)	2	4 7 9 6
insert(1, 3)		4 3 7 9 6
insert(3, 4)		4 3 7 4 9 6
size()	6	4 3 7 4 9 6

操作	输出	向量组成（自左向右）
disordered()	3	4 3 7 4 9 6
find(9)	4	4 3 7 4 9 6
find(5)	-1	4 3 7 4 9 6
sort()		3 4 4 6 7 9
disordered()	0	3 4 4 6 7 9
search(1)	-1	3 4 4 6 7 9
search(4)	2	3 4 4 6 7 9
search(8)	4	3 4 4 6 7 9
search(9)	5	3 4 4 6 7 9
search(10)	5	3 4 4 6 7 9
uniquify()		3 4 6 7 9
search(9)	4	3 4 6 7 9

Vector模板类

```
typedef int Rank; //秩
#define DEFAULT_CAPACITY  3 //默认初始容量（实际应用中可设置为更大）
template <typename T> class Vector { //向量模板类
   private: Rank _size; int _capacity; T* _elem; //规模、容量、数据区
   protected:
      /* ... 内部函数 */
   public:
      /* ... 构造函数 */
      /* ... 析构函数 */
      /* ... 只读接口 */
      /* ... 可写接口 */
      /* ... 遍历接口 */
};
```



构造与析构

```
❖ Vector(int c = DEFAULT_CAPACITY)
     { _elem = new T[_capacity = c]; _size = 0; } //默认

❖ Vector(T const * A, Rank lo, Rank hi) //数组区间复制
     { copyFrom(A, lo, hi); }
  Vector(Vector<T> const& V, Rank lo, Rank hi)
     { copyFrom(V._elem, lo, hi); } //向量区间复制
  Vector(Vector<T> const& V)
     { copyFrom(V._elem, 0, V._size); } //向量整体复制

❖ ~Vector() { delete [] _elem; } //释放内部空间
```

基于复制的构造

```
❖ template <typename T> //T为基本类型，或已重载赋值操作符'='
  void Vector<T>::copyFrom(T const * A, Rank lo, Rank hi) {
     _elem = new T[_capacity = 2*(hi - lo)]; //分配空间
     _size = 0; //规模清零
     while (lo < hi) //A[lo, hi)内的元素逐一
        _elem[_size++] = A[lo++]; //复制至_elem[0, hi - lo)
  }  //O(hi - lo) = O(n)
```