第 11 课

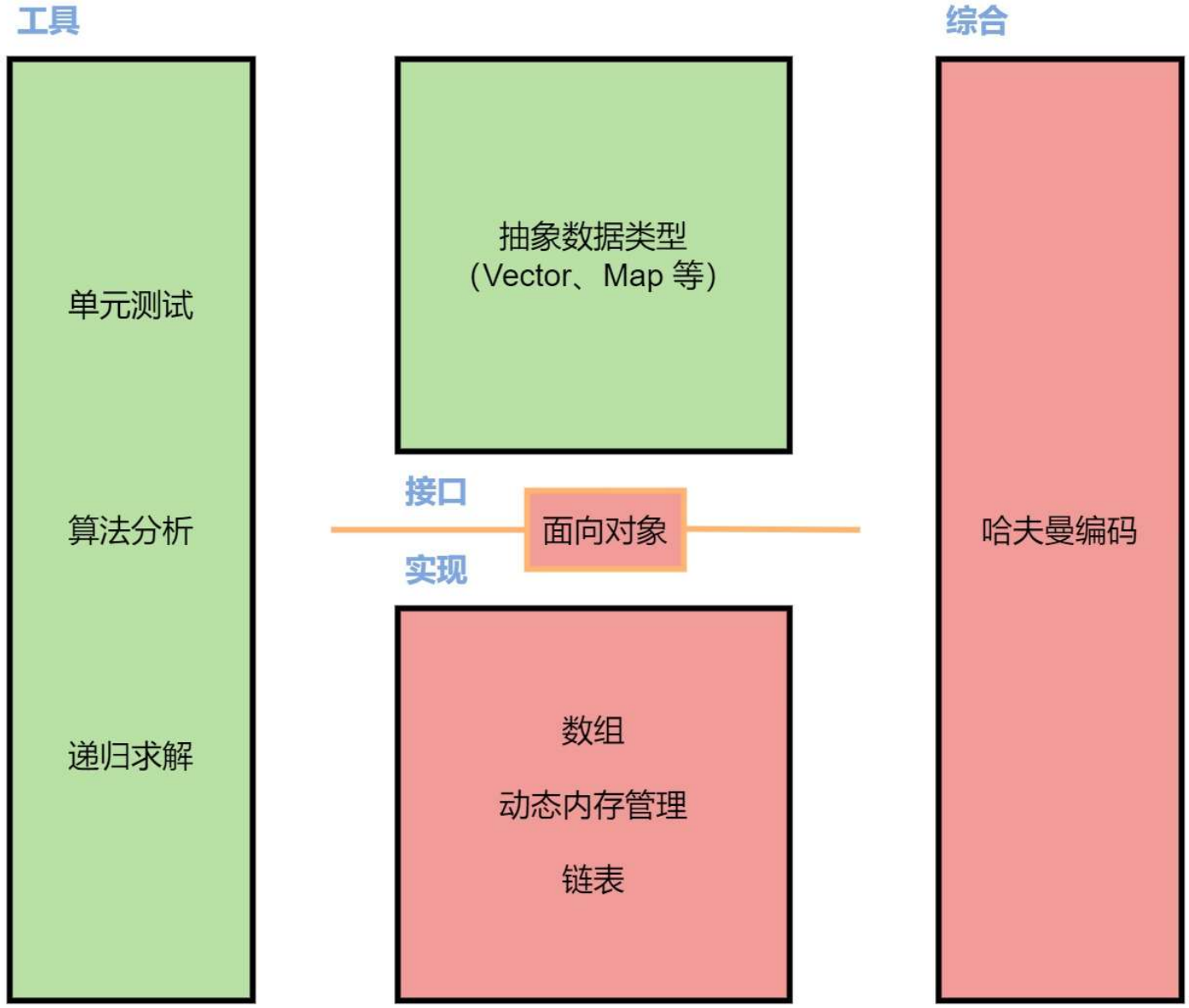
面向对象、动态内存

薛浩

xuehao0618@outlook.com

阅读

- Programming Abstraction in C++ *Chapter 6,11,12*


工具
综合
单元测试
算法分析
递归求解
抽象数据类型
(Vector、Map 等)
接口
面向对象
实现
数组
动态内存管理
链表
哈夫曼编码

今日话题

- 面向对象
- 动态内存

面向对象

再谈抽象

- 计算机建模：
  - 如何将问题转换为计算机可以表示的形式？
- 设计 ADT：
  - 如何提取事物的本质特性，创建自己的数据类型？

银行储户类

- 操作
  - 存钱
  - 取钱
  - 查询
  - 放贷
  - 发工资
- 数据
  - 姓名
  - 余额
  - 员工数
  - 总资产

银行储户类

- 操作
  - 存钱
  - 取钱
  - 查询
  - ~~放贷~~
  - ~~发工资~~
- 数据
  - 姓名
  - 余额
  - ~~员工数~~
  - ~~总资产~~

小试牛刀

分析一个智能家居系统

✨ 抽象

- 隐藏底层实现细节
- 仍然允许用户访问复杂的功能
- 目的是降低软件复杂度

✨ 类

- 用户自定义数据类型
- 对一个事物或数据的抽象

结构体 VS 类

```
struct Student {
    string name;
    int age;
};

Student s;
cout << "必须显式初始化" << endl;
cout << s.name << endl;
cout << s.age << endl;

cout << "没有检查机制" << endl;
s.name = "小明";
s.age = -5;
```

结构体 VS 类

```
class Student {
    string name;
    int age;



};

Student s;
s.name = "小明";
cout << s.name << endl;
```

结构体 VS 类

```
class Student {
    string name;
    int age;
public:
    void setName(string str) { name = str; }
    string getName() { return name; }
};

Student s;
s.setName("小明");
cout << s.getName() << endl;
```

结构体 vs 类

- 结构体的成员变量默认为 public 可被外界访问
- 类的成员变量默认为 private 仅在类内部访问

✨ 封装

把相关的一组数据和函数组合在一起，并定义数据的访问范围的过程

OOP IN CS106B

- Class
  - Construction, destruction
- Object
- Encapsulation
- Inheritance
- Abstraction
  - .h vs .cpp
  - Template classes
- Polymorphism
  - Operator overloading
  - Vitrial functions

OOP IN CS106B

- Class
  - Construction, destruction
- Object
- Encapsulation
- ~~Inheritance~~
- Abstraction
  - .h vs .cpp
  - ~~Template classes~~
- Polymorphism
  - Operator overloading
  - ~~Vitrial functions~~

OOP IN CS106B

- Class
  - Construction, destruction Basic
- Object
- Encapsulation
- ~~Inheritance~~
- Abstraction
  - .h vs .cpp
  - ~~Template classes~~
- Polymorphism
  - Operator overloading Basic
  - ~~Vitrial functions~~

OOP IN CS106B

- Class
  - Construction, destruction Basic
- Object
- Encapsulation
- ~~Inheritance~~
- Abstraction
  - .h vs .cpp
  - ~~Template classes~~
- Polymorphism
  - Operator overloading Basic
  - ~~Vitrial functions~~

OOP IN CS106B

- Class
  - Construction, destruction Basic
- Object
- Encapsulation
- ~~Inheritance~~
- Abstraction
  - .h vs .cpp
  - ~~Template classes~~
- Polymorphism
  - Operator overloading Basic
  - ~~Vitrial functions~~

类的设计

- 成员变量
  - 类封装的信息，一般不可以被外界直接访问
- 成员函数（方法）
  - 对象可以调用的操作逻辑，例如 vec.add(), vec.size() 等
- 构造器
  - 在定义类的变量，即对象时，都是通过构造器来创建的

银行账户类

- 成员变量：类封装的信息，一般不可以被外界直接访问
  - 账户名称 name
  - 账户余额 amount

```
class BankAccount {
    // ommitted
private:
    string name;
    double amount;
};
```

银行账户类

- 成员函数（方法）：对象可以调用的操作逻辑
  - 存钱 ba.deposit(100)
  - 取钱 ba.withdraw(100)
  - 查询 ba.getName()、ba.getAmount()

```
class BankAccount {
public:
    void deposit(double amount);
    void withdraw(double amount);

    double getAmount() const;
    string getName() const;

    // ommitted
};
```

银行账户类

- 构造器：定义类对象时，自动调用构造器来创建

```
class BankAccount {
public:
    BankAccount(string name, double amount);

    // ommitted
};
```

类的接口与实现

- 接口：定义了什么样的操作可以用于对象修改状态信息
- 实现：定义了这些操作具体的执行逻辑

接口：头文件

```
// BankAccount.h
#pragma once
#include <string>
using std::string;

class BankAccount {
public:
    BankAccount(string name, double amount);
    void deposit(double amount);
    void withdraw(double amount);
    void transfer(double amount, BankAccount &recipient);

    double getAmount() const;
    string getName() const;
```

实现: CPP 文件

```cpp
#include "BankAccount.h"
using std::string;

BankAccount::BankAccount(string name, double amount) {
    name = name;       // error
    amount = amount;   // error
}
```

命名空间限定符 : : 指明属于哪个类成员

THIS 关键字

避免命名冲突

```
BankAccount::BankAccount(string name, double amount) {
    this->name = name;       // fixed
    this->amount = amount;   // fixed
}
```

CONST 关键字

- 函数引用参数：不改变实际参数
- 类成员函数：不改变对象状态

```
double BankAccount::getAmount() const {
    return amount;
}
```

小试牛刀

实现银行账户类

今日话题

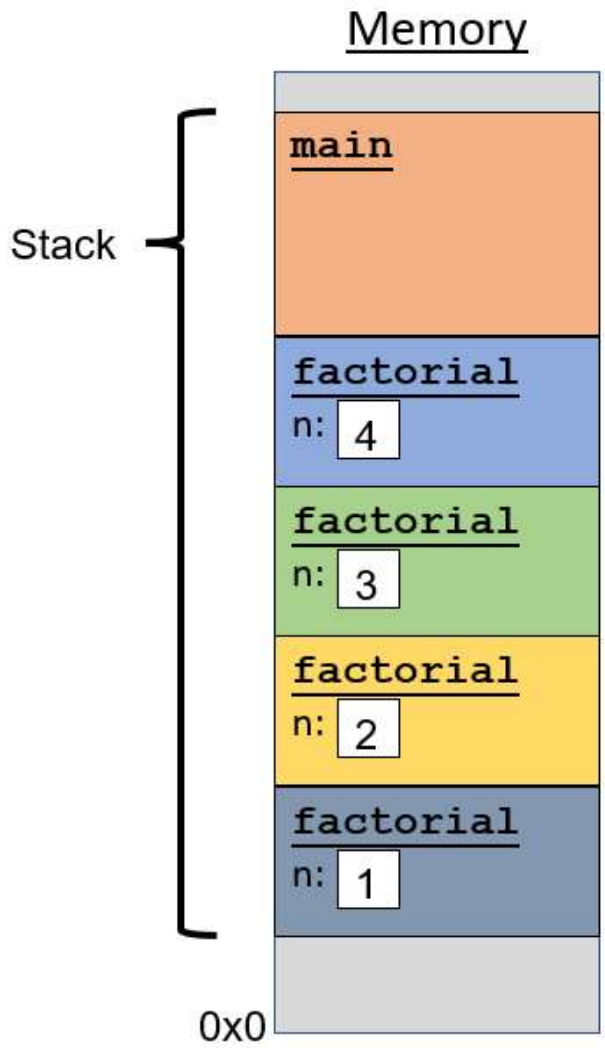
- ~~面向对象~~
- 动态内存

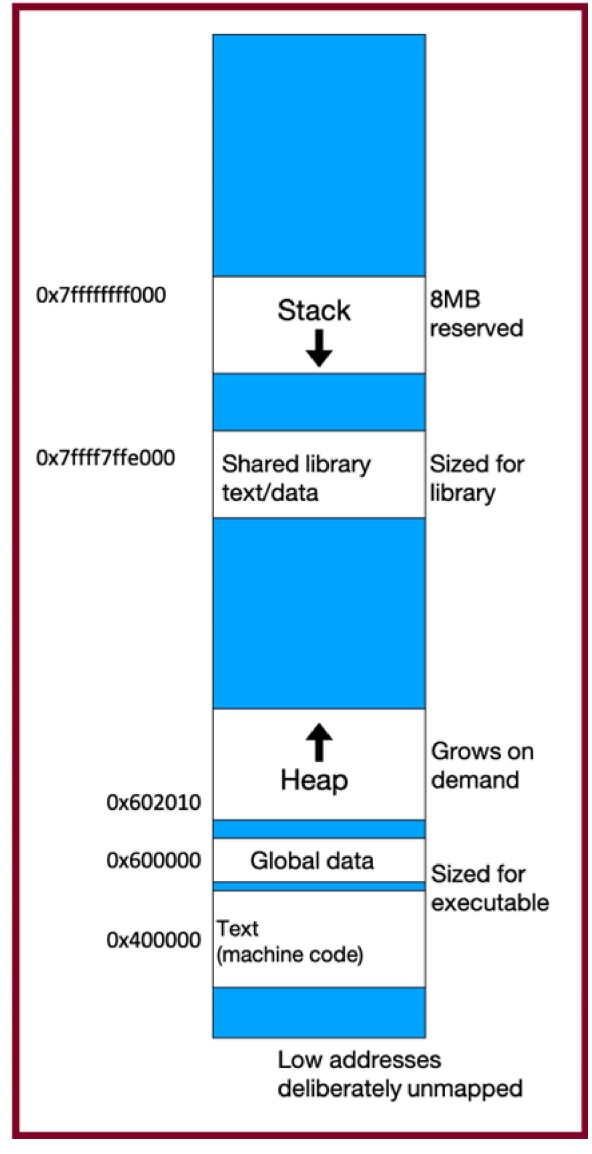
动态内存

❓ 问题

Vector、 Stack、 Queue 可以不断地添加元素，

这些元素如何存储?


Memory
Stack
main
factorial
n: 4
factorial
n: 3
factorial
n: 2
factorial
n: 1
0x0


0x7fffffffff000
Stack
8MB reserved
0x7ffff7ffe000
Shared library text/data
Sized for library
Heap
Grows on demand
0x602010
0x600000
Global data
Sized for executable
0x400000
Text (machine code)
Low addresses deliberately unmapped

内存分配方式

- 静态分配
  - 由编译器为声明的变量分配
  - 程序生命周期内有效，如全局变量
- 自动分配
  - 程序运行时，函数在栈中分配
  - 函数返回后，空间收回
- 动态分配

动态内存分配

- 使用堆来管理内存的动态分配
- 在程序执行期间完成，可以更改内存大小
- 在不需要时，显式释放内存

数组

数组是计算机内存中连续的空间块，分成多个块，每个块可以包含一条信息

数组

- 连续意味着每个块都直接相邻，没有间隙
- 特定的类型，每个块只能存放指定的类型
- 每个块有一个索引，可以通过索引访问元素

数组

Index:	0	1	2	3	4	5	6

动态数组

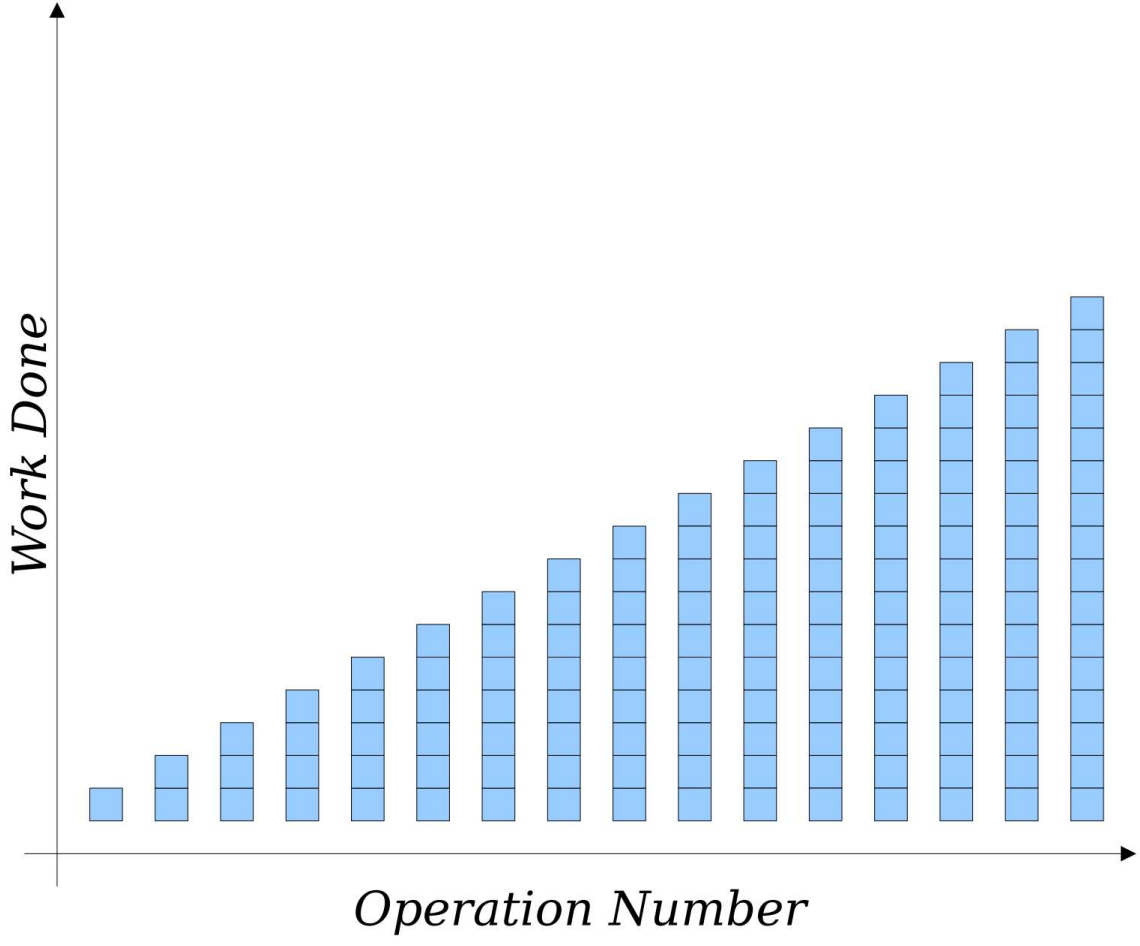
```
int* arr[10];                  // 自动分配（Stack）
int * arr = new int[10];       // 动态分配（Heap）
delete[] arr;                  // 收回动态内存
```

问题

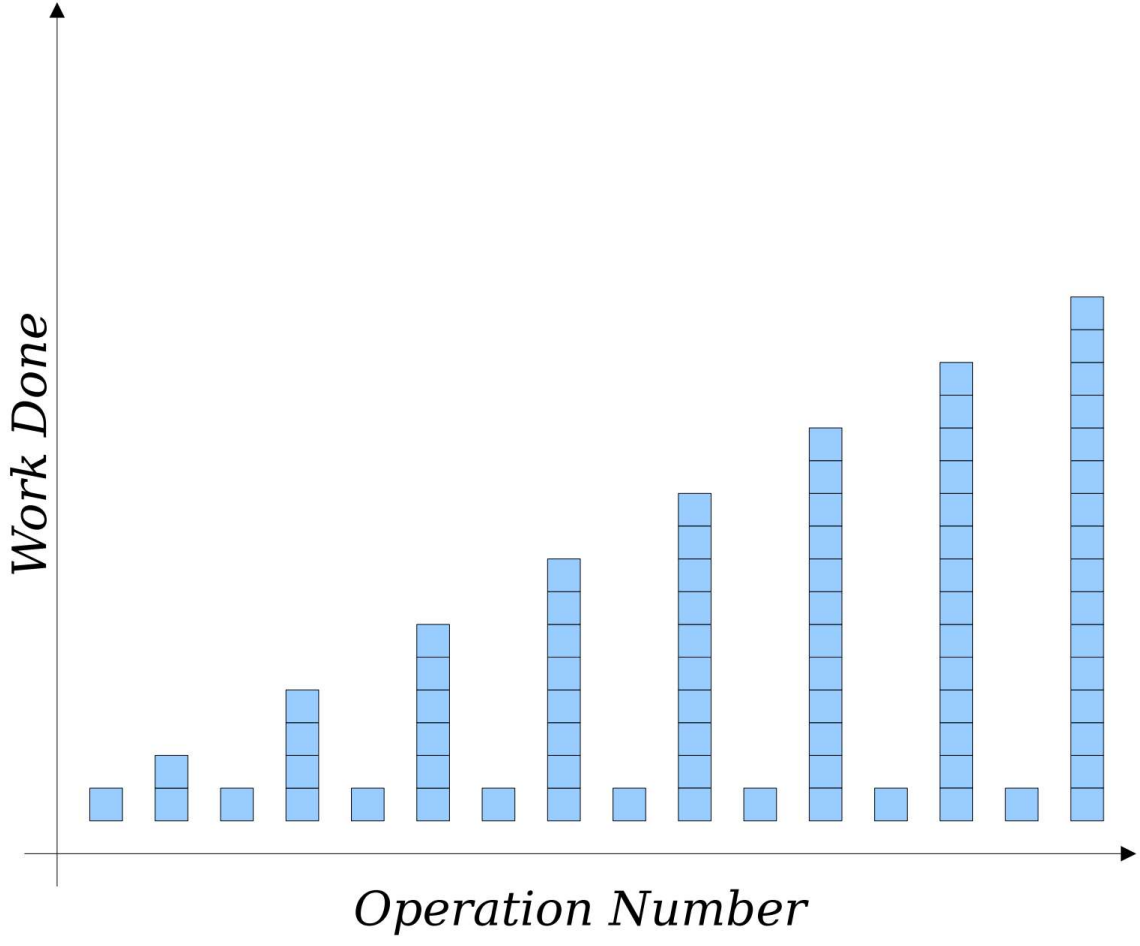
- 系统一旦分配，大小就固定，无法增加或缩小
- 数组没有边界检查

```
int * arr = new int[10];
arr[10]  // ???
```

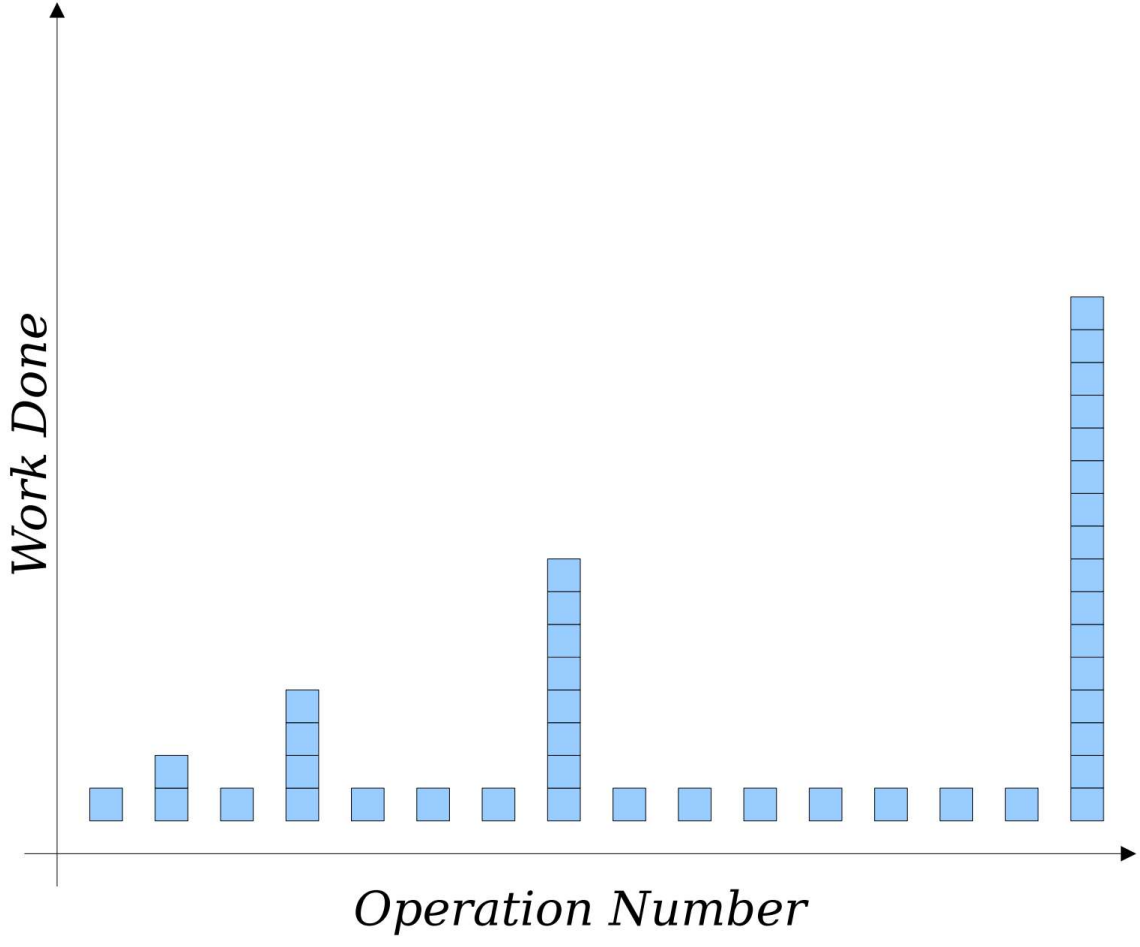
动态分配策略



动态分配策略



动态分配策略



小试牛刀

测试不同分配策略的性能

下一次课

- 效率和表示

THE END