运行时刻环境
Part1：存储组织

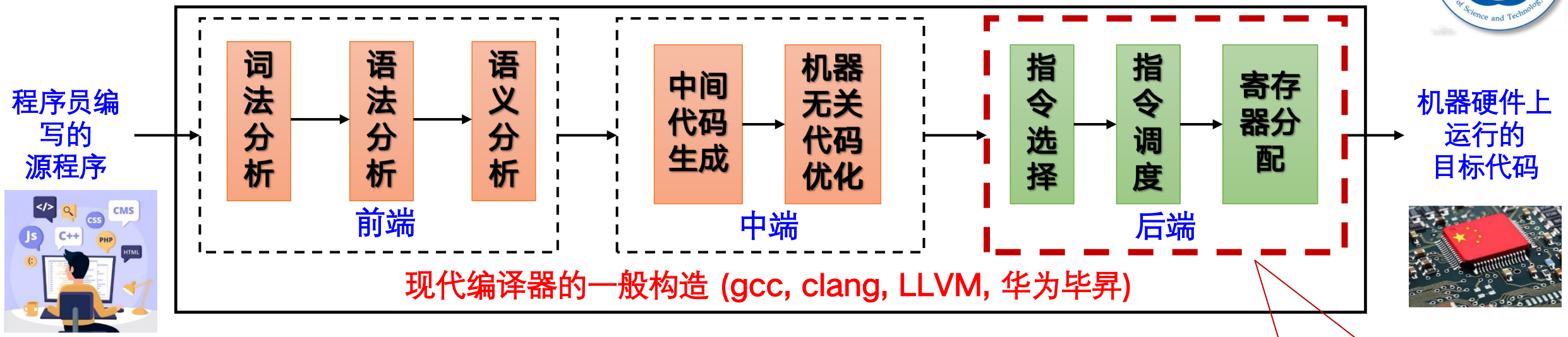
徐 伟

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年4月10日

本节提纲



- 运行时环境概述
- 存储空间的组织与分配

目标程序需要一个运行环境!

运行时环境

- **编译器为目标程序创建并管理一个运行时环境，目标程序运行在该环境中。**
  - 对象存储位置和空间的分配
  - 访问变量的机制
  - 过程间的连接
  - 参数传递机制

运行时涉及的主要内容

- 运行时存储空间组织管理概述
- 活动树与栈式空间分配
- 调用序列与返回序列
- 非局部数据的访问

由编译器、操作系统、目标机器共同完成
编译器视角：目标程序运行在逻辑地址空间
操作系统视角：将逻辑地址转换为物理地址
目标机器视角：真正执行指令，访问数据，同时限制了存储空间的组织和数据的访问

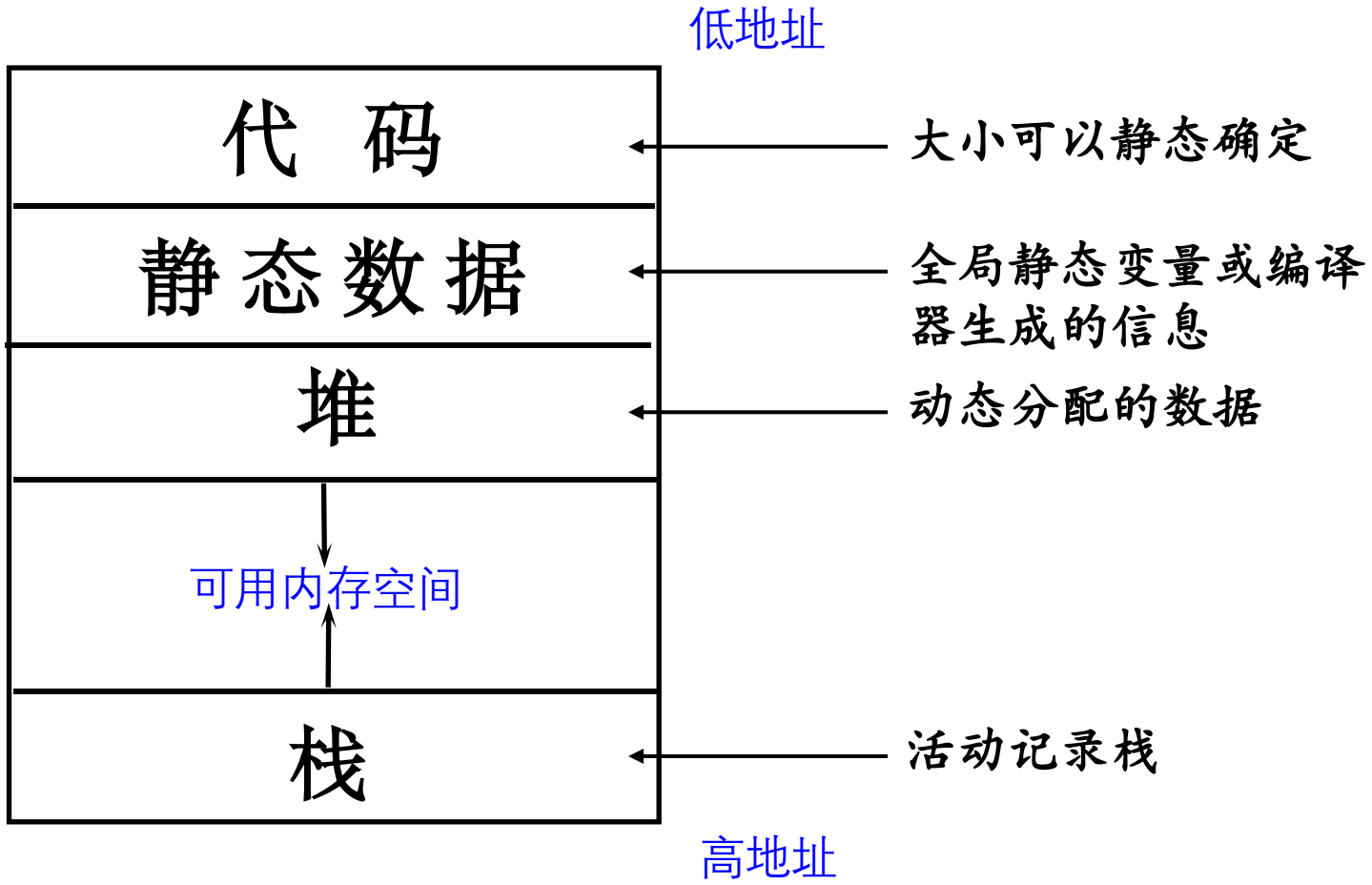
存储分配的策略

- 编译器必须为源程序中出现的一些数据对象分配运行时的存储空间
  - 静态存储分配
  - 动态存储分配
- 对于那些在编译时刻就可以确定大小的数据对象，可以在编译时刻就为它们分配存储空间，这样的策略成为静态存储分配
  - 比较简单

存储分配的策略

- 如果不能在编译时刻完全确定数据对象的大小，就要采用动态存储分配的策略。即，在编译时刻仅产生各种必要的信息，而在运行时刻，再动态地分配存储空间。
  - 栈式存储分配
  - 堆式存储分配
- 静态和动态这两个概念分别对应编译时刻和运行时刻

程序的存储分配


低地址
代 码
静 态 数 据
堆
可用内存空间
栈
高地址
大小可以静态确定
全局静态变量或编译器生成的信息
动态分配的数据
活动记录栈

影响存储分配策略的语言特征

- 过程能否递归
- 当控制从过程的活动返回时, 局部变量的值是否要保留
- 过程能否访问非局部变量
- 过程调用的参数传递方式
- 过程能否作为参数被传递
- 过程能否作为结果值传递
- 存储块能否在程序控制下被动态地分配
- 存储块是否必须被显式地释放

过程的存储组织与分配

- **过程**
  - FORTRAN的子例程(subroutine)
  - PASCAL的过程/函数(procedure/function)
  - C的函数
- **过程的激活（调用）与终止（返回）**
- **过程的执行需要：**
  - 代码段＋活动记录（过程运行所需的额外信息，如参数，局部数据，返回地址等）

局部存储分配

- 基本概念：作用域与生存期
- 活动记录的常见布局
  - 字节寻址、类型、次序、对齐
- 程序块：同名情况的处理

局部存储分配

- 名字的作用域
  - 一个声明起作用的程序部分称为该声明的作用域
  - 即使一个名字在程序中只声明一次，该名字在程序运行时也可能表示不同的数据对象

如下图代码中的n

```
int f(int n){
if (n<0) error("arg<0");
else if (n==0) return 1;
else return n*f(n-1);
}
```

局部存储分配

- **名字的作用域**

再下图代码中的

```
void exampleFunction()
{
    int localVar = 10; // 局部变量
    if (localVar > 5)
    {
        int innerVar = 20; // 在if语句块中定义的局部变量
        // innerVar在这里可以被访问
    }
    // innerVar在这里是不可访问的，因为它的作用域仅限于if语句块
}
```

局部存储分配

- 生存期
  - 从它被创建的时刻开始，到所在的代码块执行完毕时结束

```
void exampleFunction()
{
    int localVar = 10; // 局部变量← localVar生存期开始
    if (localVar > 5)
    {
        int innerVar = 20; // ← innerVar生存期开始
        // ← innerVar生存期结束
    }
    // ← localVar生存期结束
}
```

局部存储分配

- 生存期
  - 特例：如果局部变量被声明为静态，生存期会有所不同。
  - 静态局部变量在程序运行期间一直存在，它在程序的整个生命周期内都占用存储空间

```
void exampleFunction()
{
    static int staticLocalVar = 0; // 静态局部变量
    staticLocalVar++;
    printf("%d\n", staticLocalVar);
}
```

局部存储分配

- 局部存储分配的环境
  - 函数调用栈环境
    - 栈帧
      - 存储局部变量、函数参数、返回地址等信息的内存区域
    - 调用栈
      - 后进先出（LIFO）的数据结构，用于管理函数调用过程中的栈帧

假设有一个函数调用序列：main() → functionA() → functionB()

局部存储分配

- 局部存储分配的环境
  - 函数调用栈环境
  - 存储位置
    - 栈内存
    - 寄存器
  - 存储方式
    - 自动存储
    - 静态存储

局部存储分配

- 局部存储分配的状态
  - 分配状态
    - 已分配
    - 未分配

```
void exampleFunction()
{
    int localVar = 10; // 局部变量
    // localVar的存储空间在函数调用时被分配
}
```

当exampleFunction被调用时，localVar的存储空间被分配，处于已分配状态；
当exampleFunction返回时，localVar的存储空间被释放，处于未分配状态。

局部存储分配

- 局部存储分配的状态
  - 分配状态
  - 初始化状态
    - 已初始化
    - 未初始化

局部存储分配

- 局部存储分配的状态
  - 分配状态
  - 初始化状态
  - 访问状态
    - 可访问
    - 不可访问
  - 存储空间状态
    - 占用
    - 释放

局部存储分配

- 环境和状态
  - 环境把名字映射到左值，而状态把左值映射到右值（即名字到值有两步映射）
  - 赋值改变状态，但不改变环境
  - 过程调用改变环境
  - 如果环境将名字*x*映射到存储单元*s*，则说*x*被绑定到*s*

局部存储分配

- 静态概念和动态概念的对应

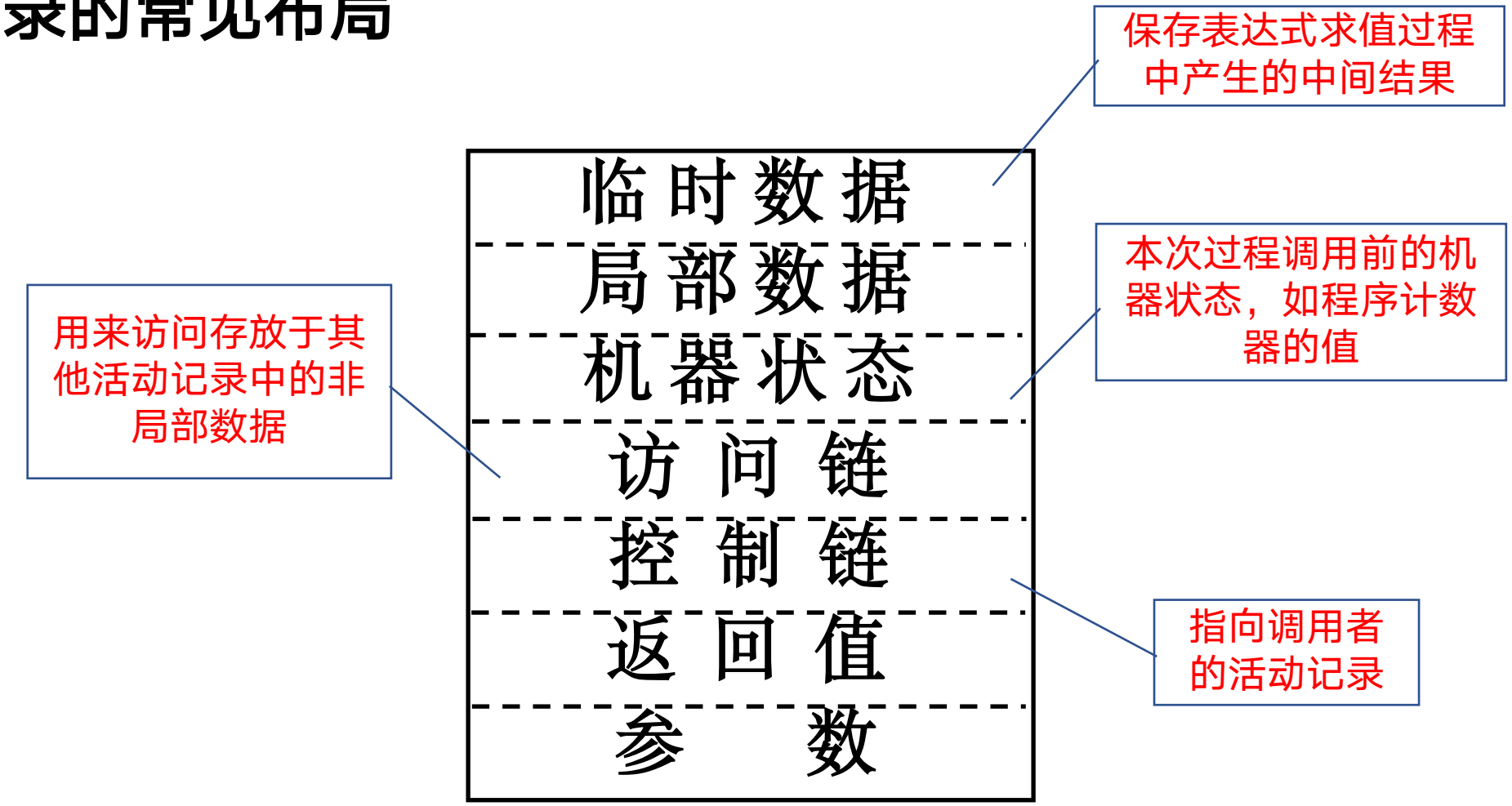
静 态 概 念	动 态 对 应
过程的定义	过程的活动
名字的声明	名字的绑定
声明的作用域	绑定的生存期

活动记录

- 使用过程(或函数、方法)作为用户自定义动作的单元的语言，其编译器通常以过程为单位分配存储空间
- 过程体的每次执行成为该过程的一个活动
- 编译器为每一个活动分配一块连续存储区域，用来管理此次执行所需的信息，这片区域称为活动记录(activation record)

活动记录

- 活动记录的常见布局



局部存储分配

- 局部数据的布局
  - 字节是可编址内存的最小单位
  - 变量所需的存储空间可以根据其类型而静态确定
  - 一个过程所声明的局部变量，按这些变量声明时出现的次序，在局部数据域中依次分配空间
  - 局部数据的地址可以用相对于活动记录中某个位置的地址来表示
  - 数据对象的存储布局还有一个对齐问题

局部存储分配

- 例　在SPARC/Solaris工作站上下面两个结构体的size分别是24和16，为什么不一样？

```
typedef struct  _a{
        char   c1;
        long   i;
        char   c2;
        double f;
}a;
```

```
typedef struct  _b{
        char c1;
        char c2;
        long i;
        double f;
}b;
```

对齐：char : 1, long : 4, double : 8

局部存储分配

- 例　在SPARC/Solaris工作站上下面两个结构体的size分别是24和16，为什么不一样？

```
typedef struct  _a{
        char   c1;     0
        long   i;      4
        char   c2;     8
        double f;      16
}a;
```

```
typedef struct  _b{
        char c1;     0
        char c2;     1
        long i;      4
        double f;    8
}b;
```

对齐：char : 1, long : 4, double : 8

局部存储分配

• 例　在x86/Linux机器的结果和SPARC/Solaris工作站不一样，是20和16。

```
typedef struct  _a{                    typedef struct  _b{
        char   c1;     0                       char c1;     0
        long   i;      4                       char c2;     1
        char   c2;     8                       long i;      4
        double f;      12                      double f;    8
}a;                                    }b;
```

对齐：char : 1, long : 4, double : 4

局部存储分配

- 程序块
  - 本身含有局部变量声明的语句
  - 可以嵌套
  - 最接近的嵌套作用域规则
  - 并列程序块不会同时活跃
  - 并列程序块的变量可以重叠分配

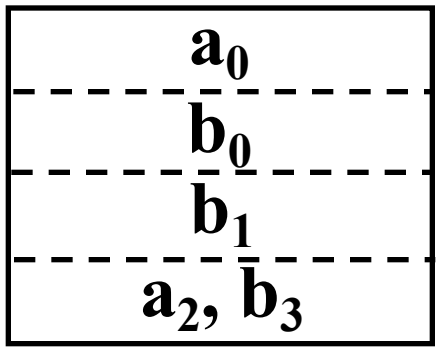
局部存储分配

```
main() / * 例 */
{                                   /* begin of B0 */
    int a = 0;
    int b = 0;
    {                                       /* begin of B1 */
        int b = 1;
        {                                           /* begin of B2 */
            int a = 2;
        }                                           /* end of B2 */
        {                                           /* begin of B3 */
            int b = 3;
        }                                           /* end of B3 */
    }                                       /* end of B1 */
}                                   /* end of B0 */
```

局部存储分配

```
main() / * 例 */
{ /* begin of B0 */
    int a = 0;
    int b = 0;
    { /* begin of B1 */
        int b = 1;
        {/* begin of B2 */
            int a = 2;
        }/* end of B2 */
        {/* begin of B3 */
            int b = 3;
        }/* end of B3 */
    }/* end of B1 */
}/* end of B0 */
```

声　明	作 用 域
int a = 0;	$B_0 - B_2$
int b = 0;	$B_0 - B_1$
int b = 1;	$B_1 - B_3$
int a = 2;	B_2
int b = 3;	B_3



重叠分配存储单元

静态分配

- 名字在程序被编译时绑定到存储单元，不需要运行时的任何支持
- 静态分配给语言带来限制
  - 递归过程不被允许
  - 数据对象的长度和它在内存中位置的限制，必须是在编译时可以知道的
  - 数据结构不能动态建立

静态分配

- 例　C程序的外部变量、静态局部变量以及程序中出现的常量都可以静态分配
- 声明在函数外面
  - 外部变量　　-- 静态分配
  - 静态外部变量　　-- 静态分配
- 声明在函数里面
  - 静态局部变量　　-- 也是静态分配
  - 自动变量　　-- 不能静态分配

动态分配

- 程序对数据区的需求在编译时是完全未知的，这是因为每个数据对象所需数据区的大小和数目在编译时是未知的。
- 主要有两种策略
  - 栈式存储：与过程调用返回有关，涉及过程的局部变量以及过程活动记录
  - 堆存储：关系到部分生存周期较长的数据

一起努力
打造国产基础软硬件体系！

徐 伟

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年4月10日