MultifileCallgraph 调研

张艺潇 王宇飞 赵宏祝

MultifileCallgraph 调研

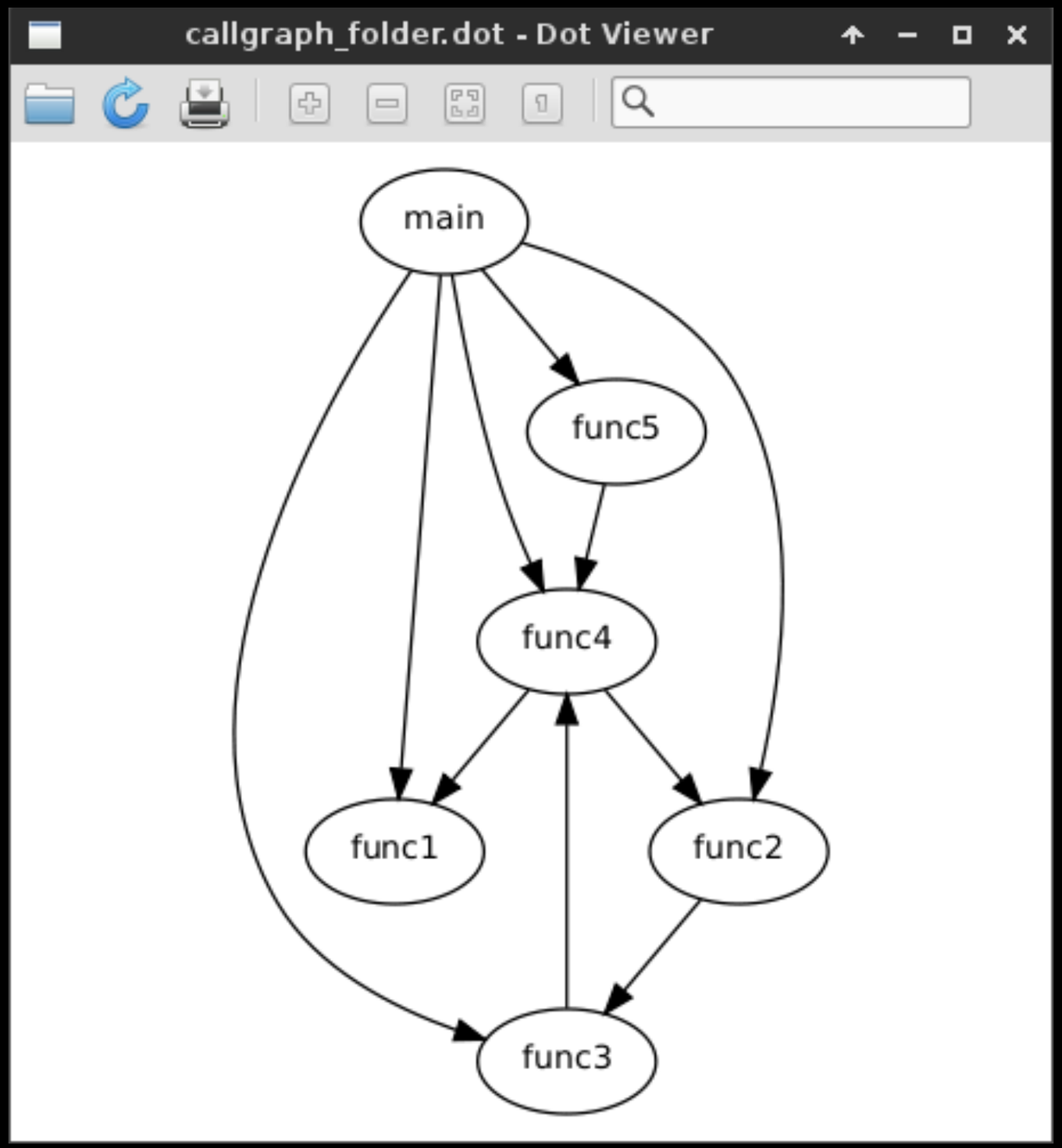
函数调用图理论

相关LLVM库

CallGraph 程序

我们的工作

函数调用图理论


callgraph_folder.dot - Dot Viewer
main
func5
func4
func1
func2
func3

函数调用图理论

MultifileCallgraph 调研

函数调用图理论

相关LLVM库

CallGraph 程序

我们的工作

相关LLVM库

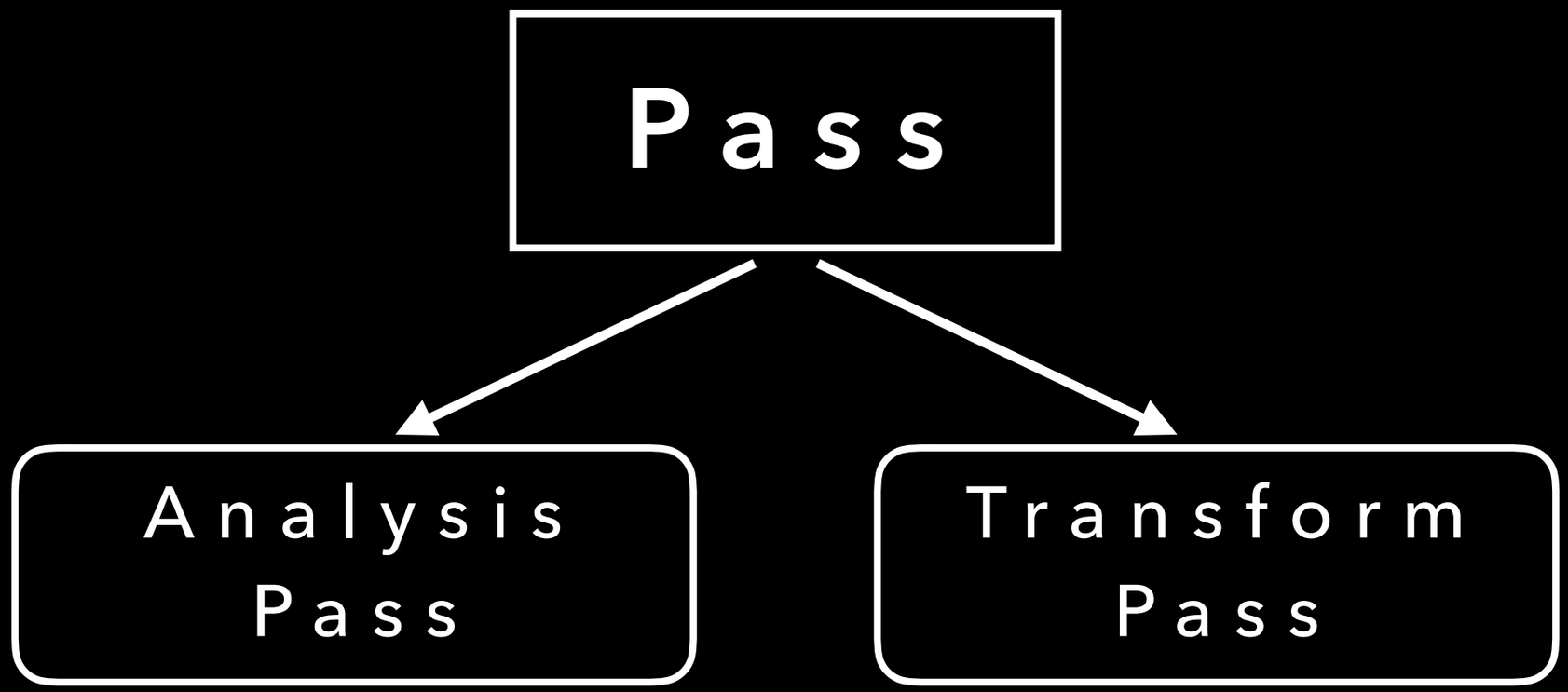
相关LLVM库

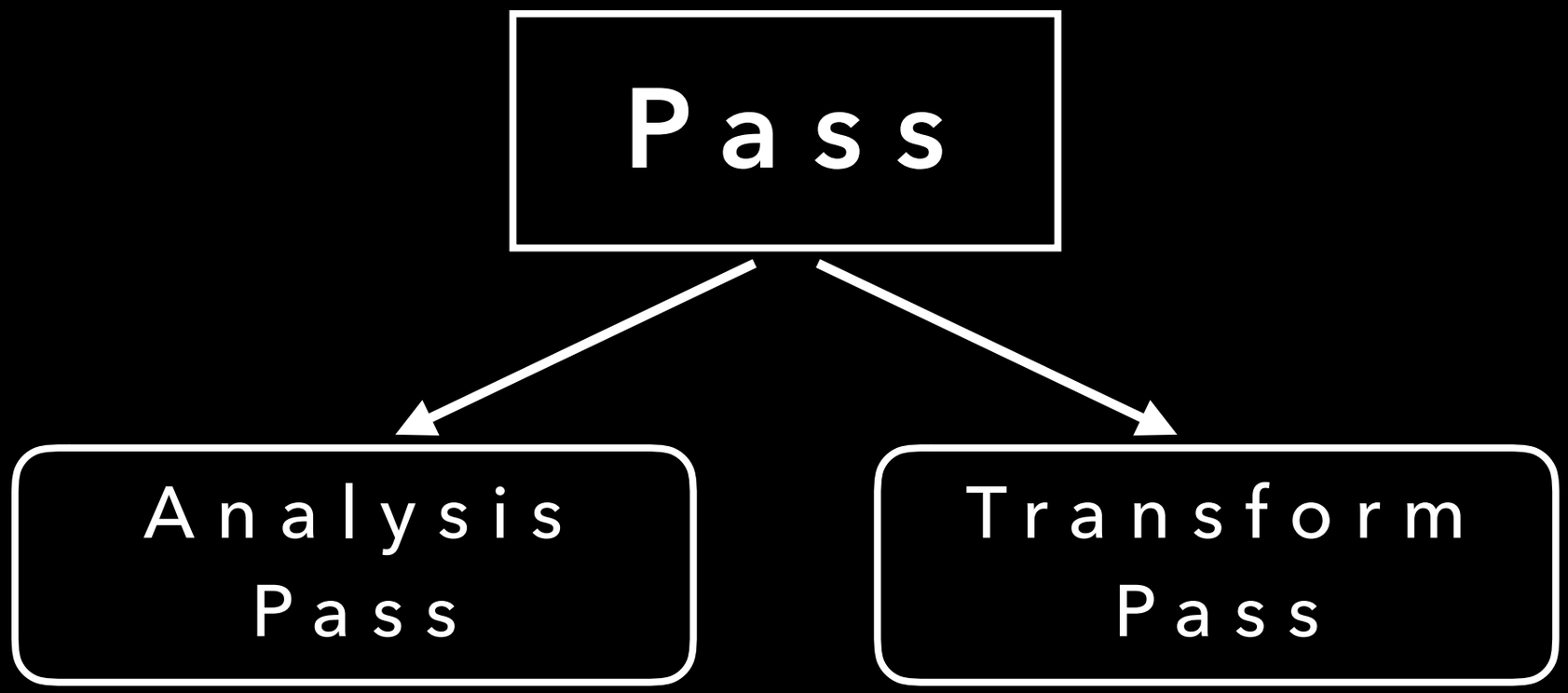
- Pass
- Data Structure Analysis (DSA)

Pass

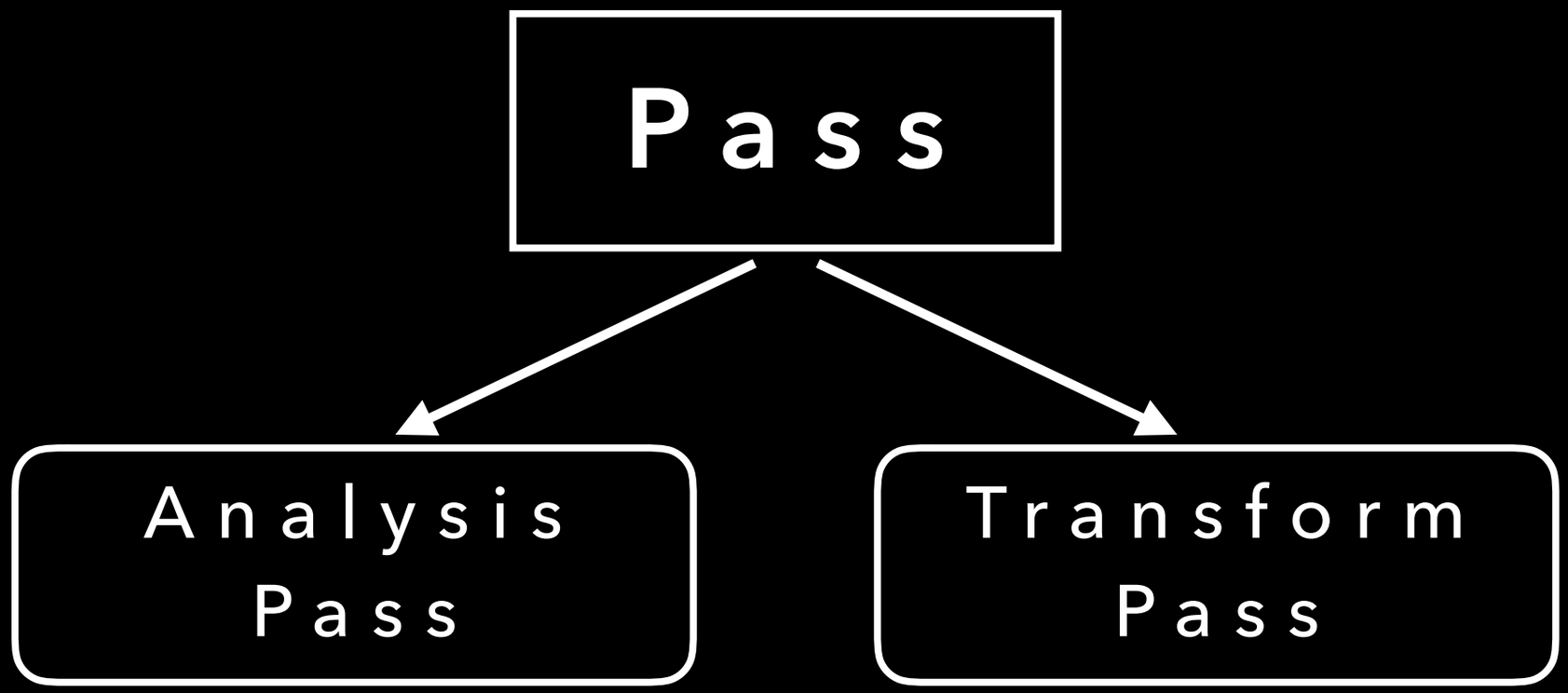
- 什么是 Pass
- 如何写一个自己的 Pass

Pass


Pass
Analysis Pass
Transform Pass



- 如何协调 Pass 间的关系?

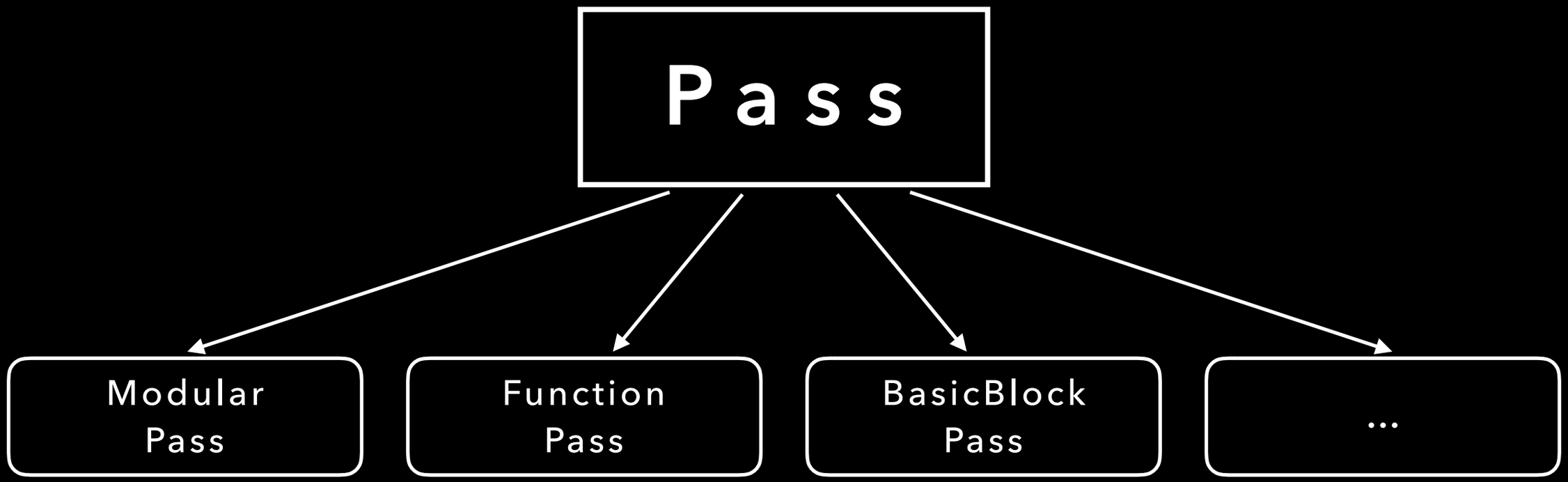


- 如何协调 Pass 间的关系?
  - Pass Manager!

Pass

- 什么是 Pass
- 如何写一个自己的 Pass

Pass


Pass
Modular Pass
Function Pass
BasicBlock Pass
...

如何写一个自己的 Pass

1. 选择合适的基类（如 Function Pass）

如何写一个自己的 Pass

1. 选择合适的基类（如 Function Pass）
2. 编写子类继承自该基类并定义 Pass 标志符 ID

如何写一个自己的 Pass

1. 选择合适的基类（如 Function Pass）
2. 编写子类继承自该基类并定义 Pass 标志符 ID

```
class MyPass : public FunctionPass {
public:
    static char ID;
    MyPass() : FunctionPass(ID) {};
    virtual bool runOnFunction(Function &F);
}

bool runOnFunction(){
        // here put the code where the pass does its work
        ...
        //return true if the pass modifies its object, false otherwise
}
```

如何写一个自己的 Pass

1. 选择合适的基类（如 Function Pass）
2. 编写子类继承自该基类并定义 Pass 标志符 ID
3. 重写 EntryPoint 方法（如 runOnFunction）

相关LLVM库

- Pass
- Data Structure Analysis (DSA)

Data Structure Analysis

- Alias Analysis
- Mod/Ref Information

Data Structure Analysis

- Alias Analysis

```
Query: alias(P1, S1, P2, S2)
Answer: Must Alias
        May Alias
        No Alias
```

Data Structure Analysis

- Alias Analysis

```
%1  = *P1
*P2 = %2
%3  = *P1
```

(P1 & P2 → No Alias)

Data Structure Analysis

- Alias Analysis

```
%1  = *P1                    %1  = *P1
*P2 = %2     ⟶              *P2 = %2
%3  = *P1                    %3  = %1
```

(P1 & P2 → No Alias)

Data Structure Analysis

- Alias Analysis
- Mod/Ref Information

```
 Query: modref(I, S, P)
Answer: No ModRef
        Mod
        Ref
        ModRef
```

相关LLVM库

MultifileCallgraph 调研

函数调用图理论

相关LLVM库

CallGraph 程序

我们的工作

CallGraph 程序

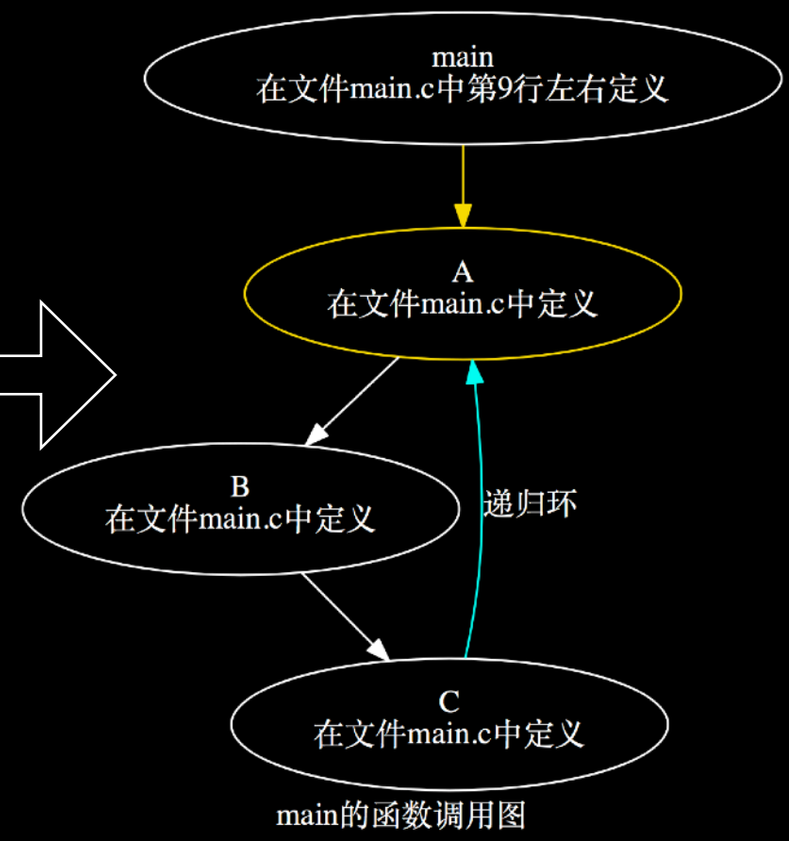
File 1

File 2

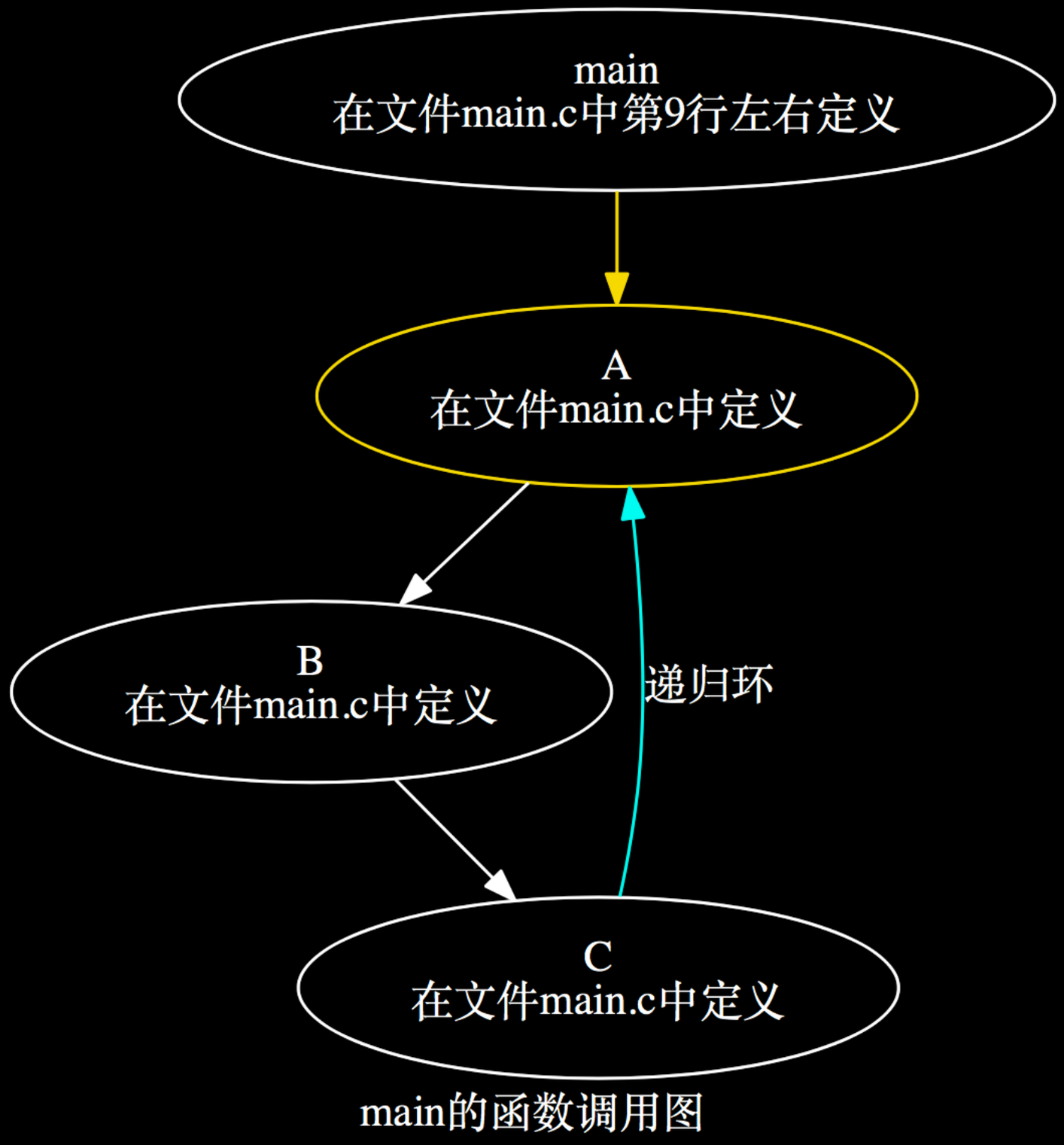
...

root
Functon

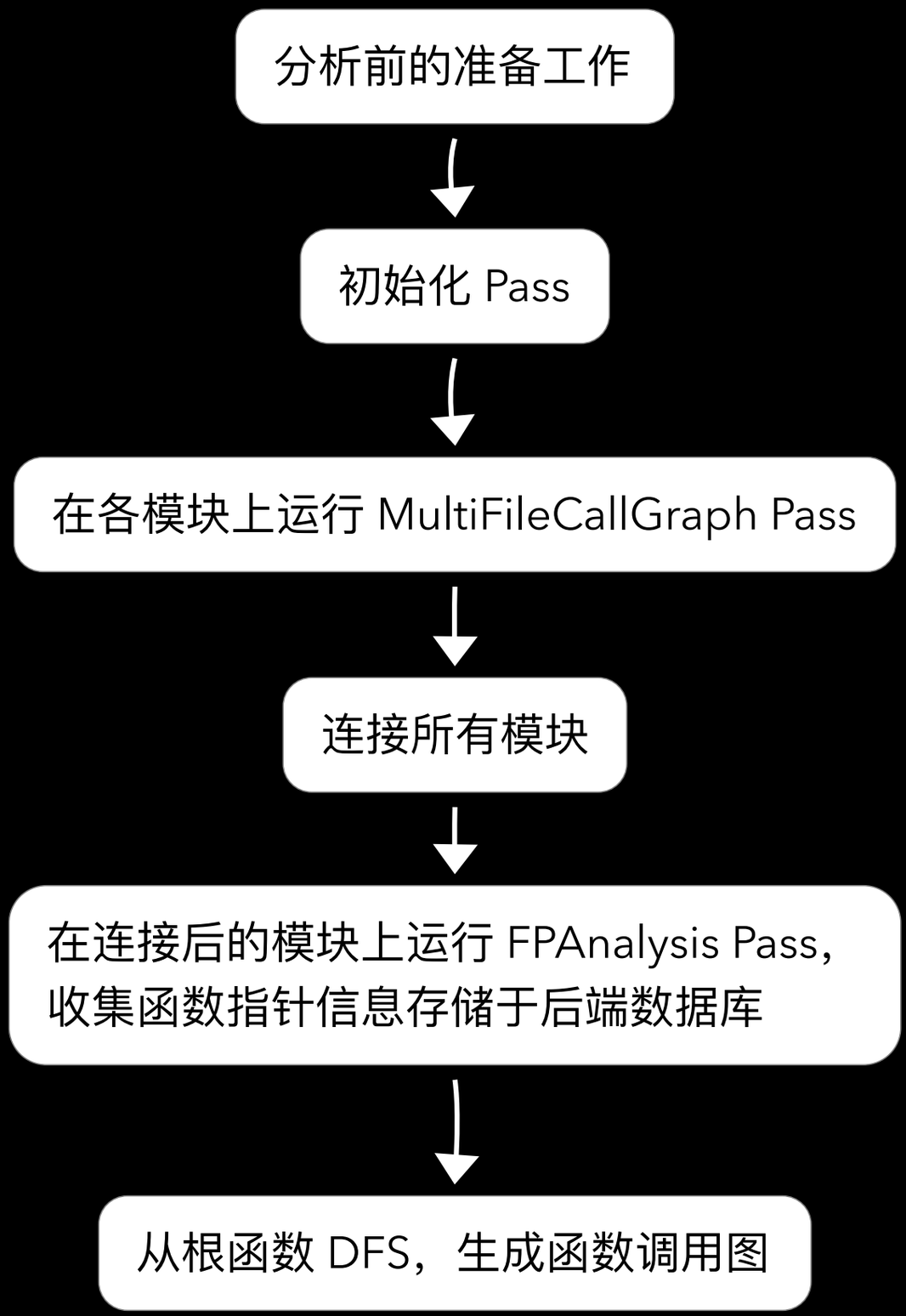
callgraph



main的函数调用图


main
在文件main.c中第9行左右定义
A
在文件main.c中定义
B
在文件main.c中定义
递归环
C
在文件main.c中定义

main的函数调用图


分析前的准备工作
初始化 Pass
在各模块上运行 MultiFileCallGraph Pass
连接所有模块
在连接后的模块上运行 FPAnalysis Pass，
收集函数指针信息存储于后端数据库
从根函数 DFS，生成函数调用图

构建数据库

构建数据库

Function List	CallInst List	Function Pointer Call

构建数据库

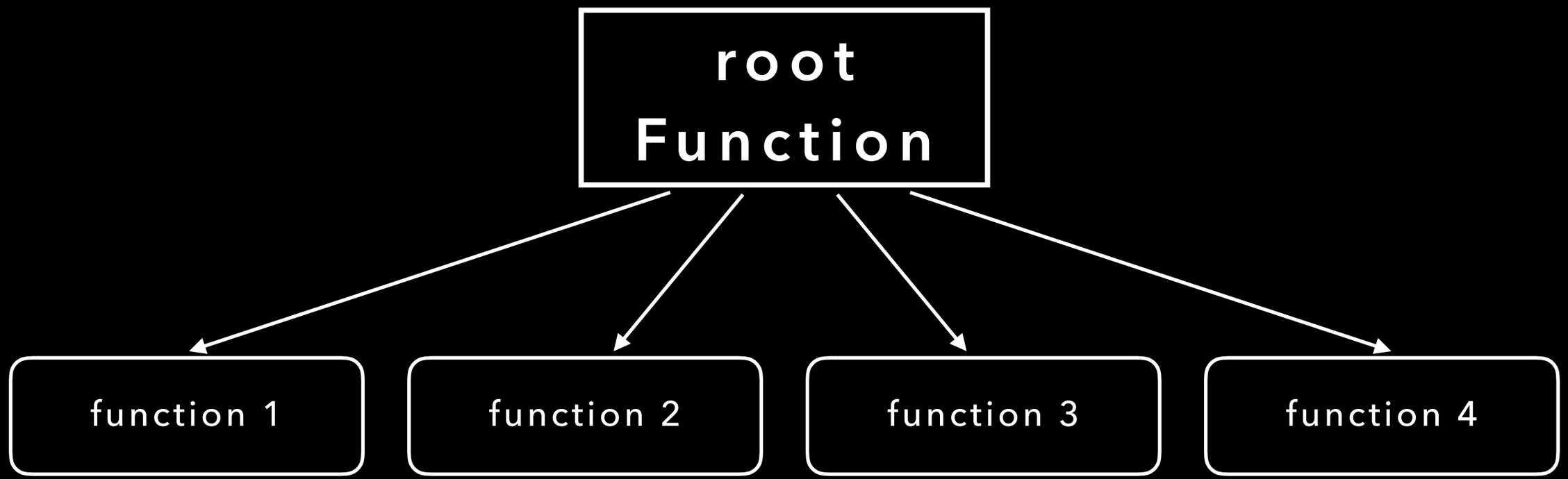
- MultiFlieCallGraph Pass

```
Search all the modules
in each module
search all the functions and add them to FunctionList
     in each function
     find all the call instructions and add them to CallInstList
```

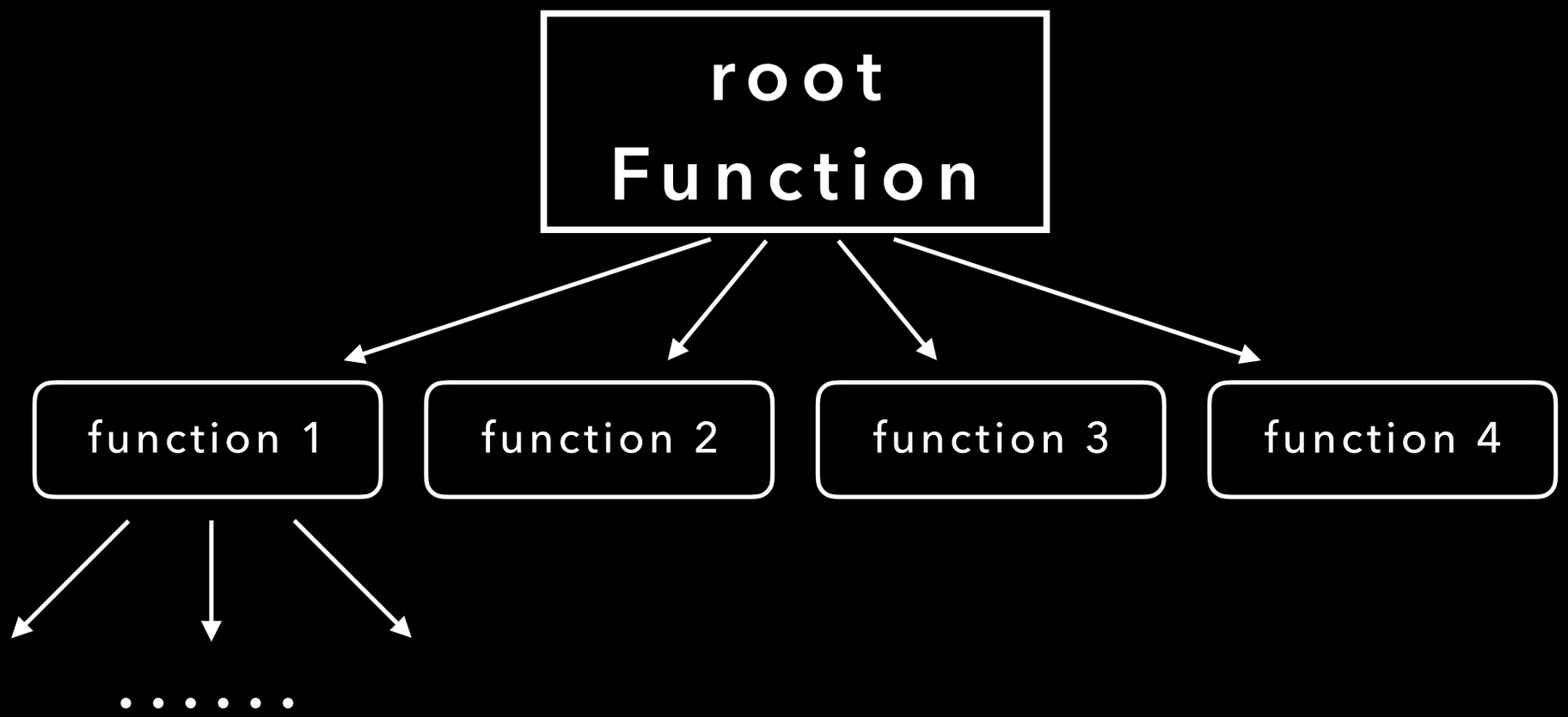
如何跨文件分析

root
Function

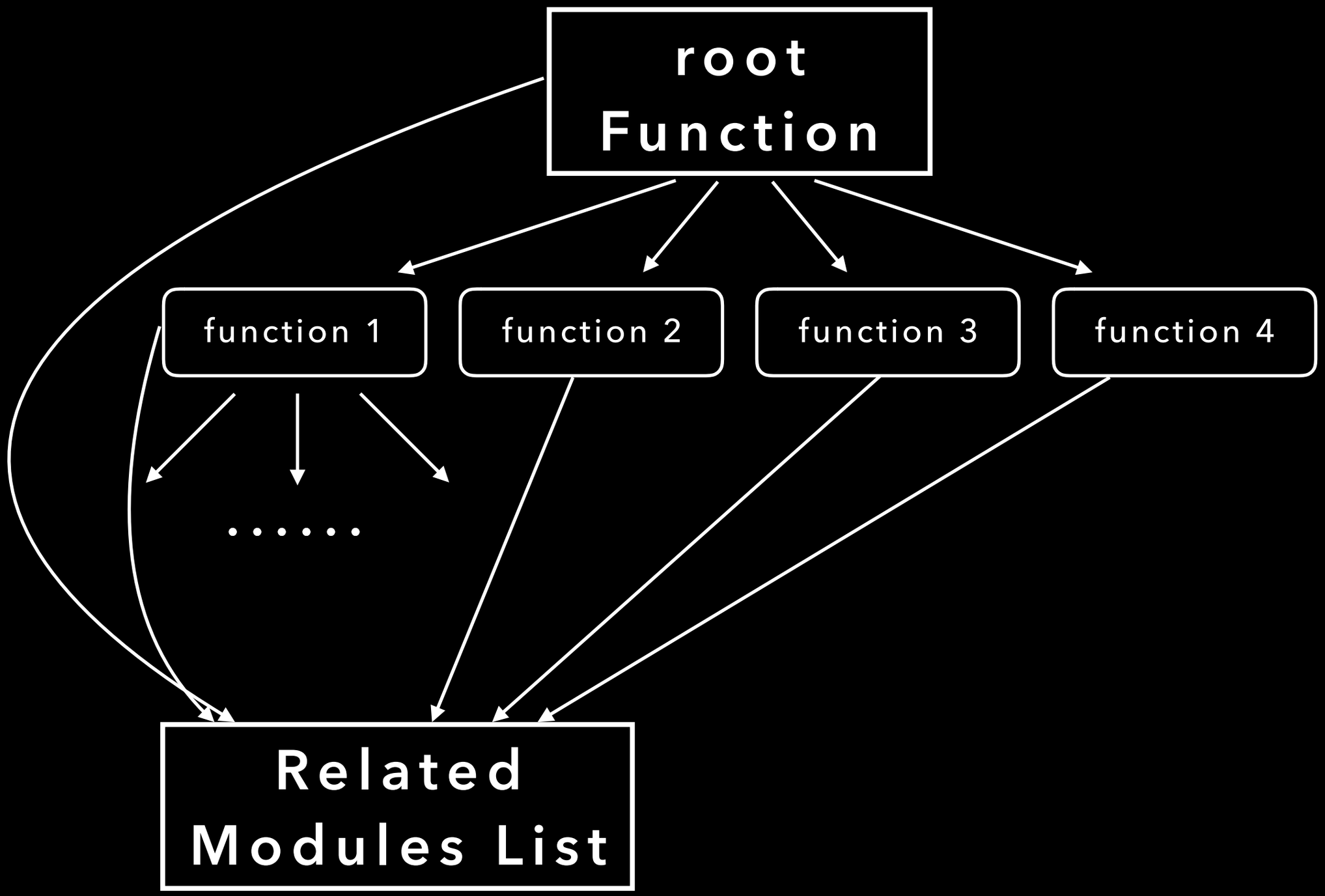
如何跨文件分析



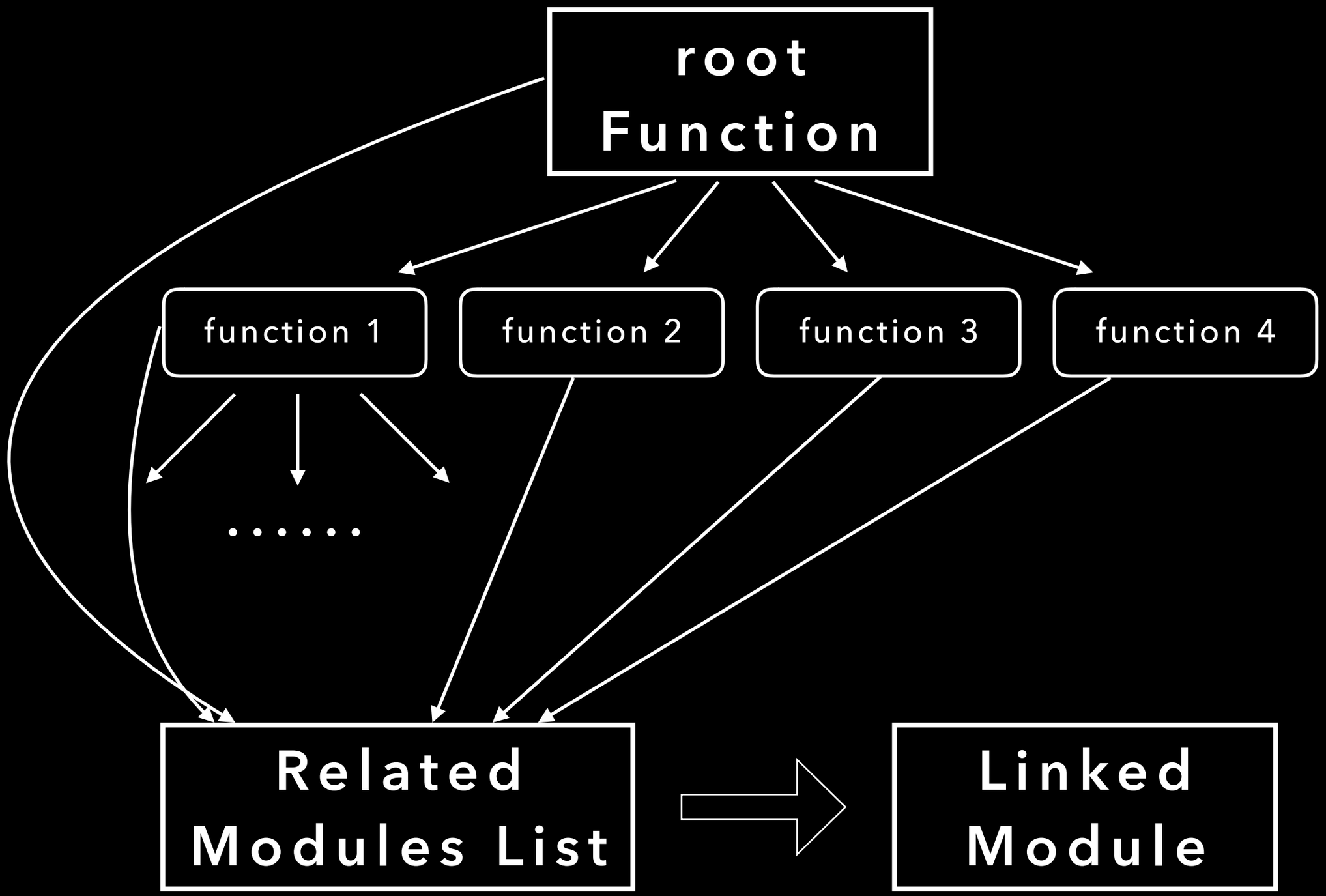
如何跨文件分析



如何跨文件分析


root
Function
function 1
function 2
function 3
function 4
.....
Related
Modules List

如何跨文件分析


root
Function
function 1
function 2
function 3
function 4
.....
Related
Modules List
Linked
Module

函数指针分析

- 在合并后的模块上调用 LLVM 提供的 DSA 分析模块，分析每个函数指针可能指向的位置
- 提取分析结果 →

Function Pointer Call

CallGraph 程序

MultifileCallgraph 调研

函数调用图理论

相关LLVM库

CallGraph 程序

我们的工作

我们的工作

我们的工作

- 支持 C++
- 支持函数调用次数统计

支持C++

- 难点：C++ 的面向对象机制
- 在 LLVM IR 层面上已经没有了面向对象的特征
- 如何将 IR 中的函数和源代码中的函数对应起来?

C++ name mangling 机制

- mangling:

  `A::PrintA() -> _ZN1A6PrintAEv`

- demangling:

  `_ZN1A6PrintAEv -> A::PrintA()`

支持C++

- 原来的分析流程：用户输入函数名 → 从文件中获取函数名，并对所有的函数名执行 demangling 操作，存入列表 → 与用户输入的函数名进行比较，选定根节点 → 开始分析
- 存在的问题：
  - C++ 的函数名 demangling 之后不能作为 graphviz 的节点名称
  - C++ 中，不同的函数 demangling 之后可能具有相同的名称

支持C++

- 解决方案-修改代码逻辑：
  - 生成节点所用名称：IR 中的函数名
  - 比较函数所用名称：IR 中的函数名
  - 节点标签信息：demangling 之后的函数名
  - 要先将用户输入的函数名进行 mangling操作

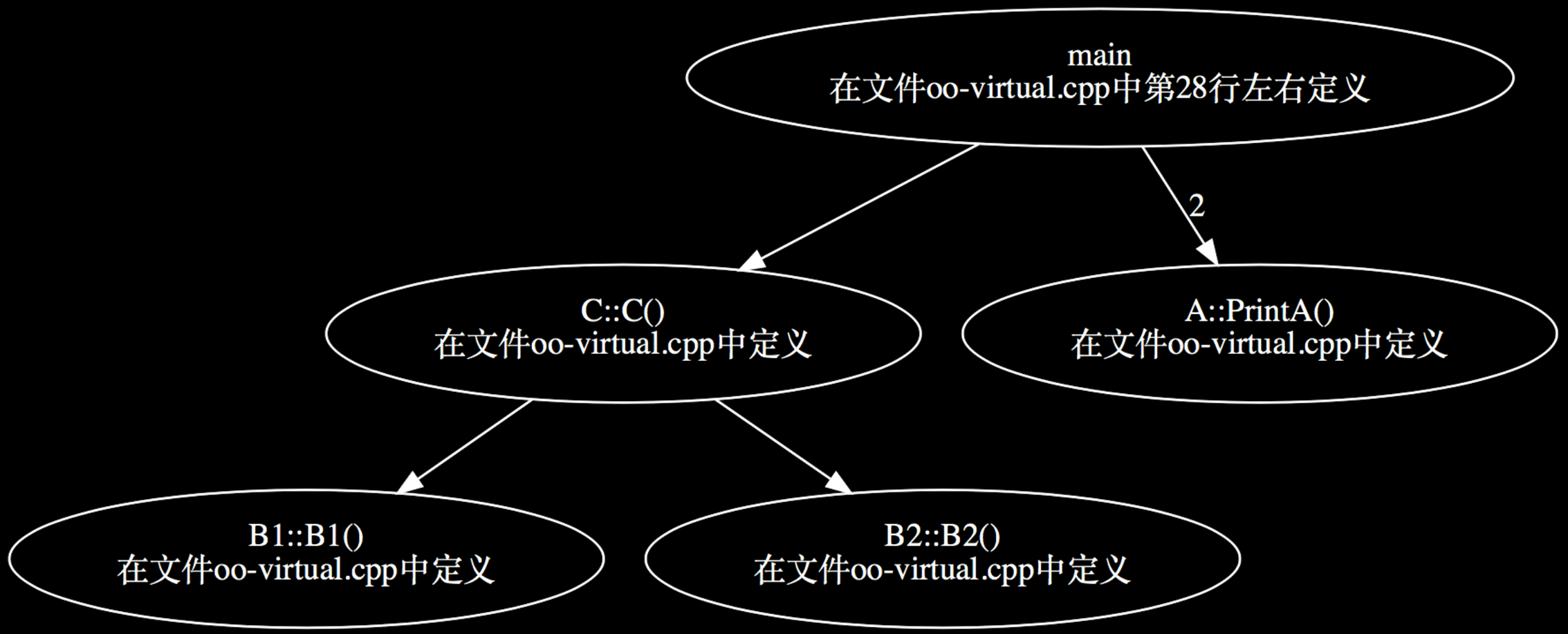
我们的工作

- 支持 C++
- 支持函数调用次数统计

支持函数调用次数统计

- 如何统计被调用函数的次数?
  - 在函数列表中添加一个属性
  - 分析时每次遇到相应的函数节点则将变量加一
- 存在的问题：静态分析的时候不能统计递归调用的次数

```
#include <iostream>

using namespace std;

class A {
    public:
        int a;
        int PrintA ( void )
        {
            return a;
        }
};

class B1: virtual public A {

};

class B2: virtual public A {

};

class C: public B1, public B2 {

};

int main ( void )
{
    C c;

    c.B1::a = 1;
    c.B2::a = 2;
    c.B1::PrintA();
    c.B2::PrintA();

    return 0;
}
```


main
在文件oo-virtual.cpp中第28行左右定义
2
C::C()
在文件oo-virtual.cpp中定义
A::PrintA()
在文件oo-virtual.cpp中定义
B1::B1()
在文件oo-virtual.cpp中定义
B2::B2()
在文件oo-virtual.cpp中定义

main的函数调用图

可能的改进

- 函数调用图剪枝：只保留需要分析的函数所在的路径
- 处理外部函数：建立外部函数表来跟踪访问，并生成标签信息
- 优化 DSA 分析效率：可调的精度
- 优化程序实现：建立一个从函数名到 Function * 的映射，利用哈希散列的方法来减少函数查找的比较次数
- 将程序前端从主函数中分离开

我们的工作

MultifileCallgraph 调研

函数调用图理论

相关LLVM库

CallGraph 程序

我们的工作

总结

参考文献

[1] Lattner C A. Macroscopic Data Structure Analysis and Optimization[J]. 2005.

[2] Lattner C, Lenharth A, Adve V. Making context-sensitive points-to analysis with heap cloning practical for the real world[J]. Acm Sigplan Notices, 2007, 42(6):278-289.

[3] Hammer C, Snelting G. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs[J]. International Journal of Information Security, 2009, 8(6):399-422.

[4] Lopes B C, Auler R. Getting Started with LLVM Core Libraries[M]. Packt Publishing, 2014.

[5] 宁宇, 张昱. LLVM 3.9 介绍[R]. 合肥:中国科学技术大学, 2016.

[6] LLVM Project. LLVM Alias Analysis Infrastructure[EB/OL]. llvm.org/docs/AliasAnalysis.html, 2016.

[7] LLVM Project. LLVM's Analysis and Transform Passes[EB/OL]. llvm.org/docs/Passes.html, 2016.

[8] LLVM Project. Writing an LLVM Pass[EB/OL]. http://llvm.org/docs/WritingAnLLVMPass.html, 2016.

[9] Wikipedia. Name mangling[EB/OL]. https://en.wikipedia.org/wiki/Name_mangling, 2016.

[10] Wikipedia. Call graph[EB/OL]. https://en.wikipedia.org/wiki/Call_graph, 2016.