Brief Intro to LLVM Backend

Shi Ningning/史宁宁

Contents

About Me

snsn1984

博客专家

原创	粉丝	喜欢	评论
171	937	506	460

等级：博客 7　　访问：96万+

积分：1万+　　排名：1893

勋章：

LLVM每日谈

阅读量：205568　　38 篇

LLVM零基础学习

阅读量：86769　　8 篇

深入研究Clang

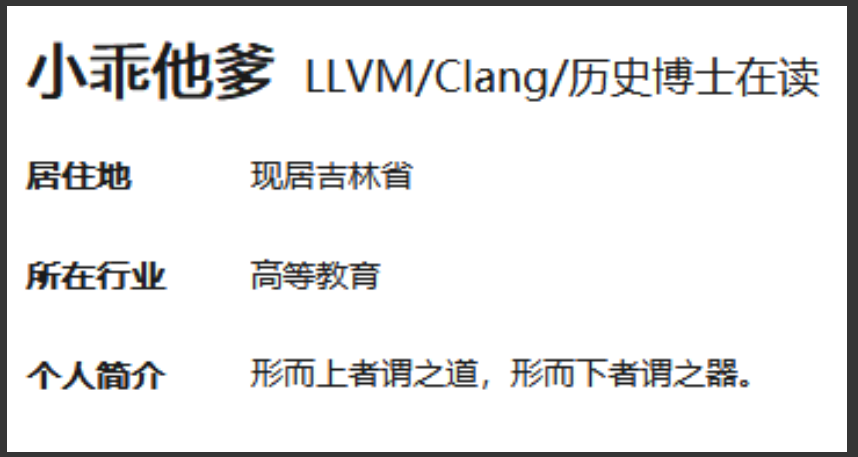
阅读量：89540　　12 篇

About Me

CSDN ID:snsn1984

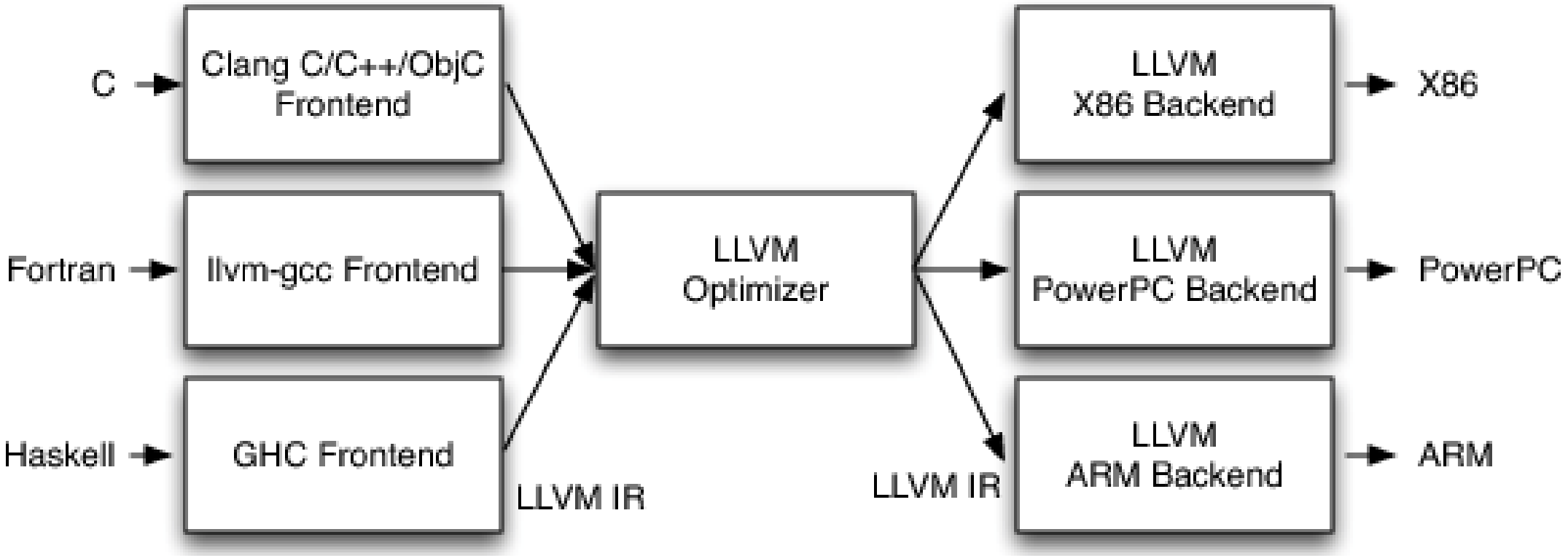
About Me

- Zhihu ID：小乖他爹



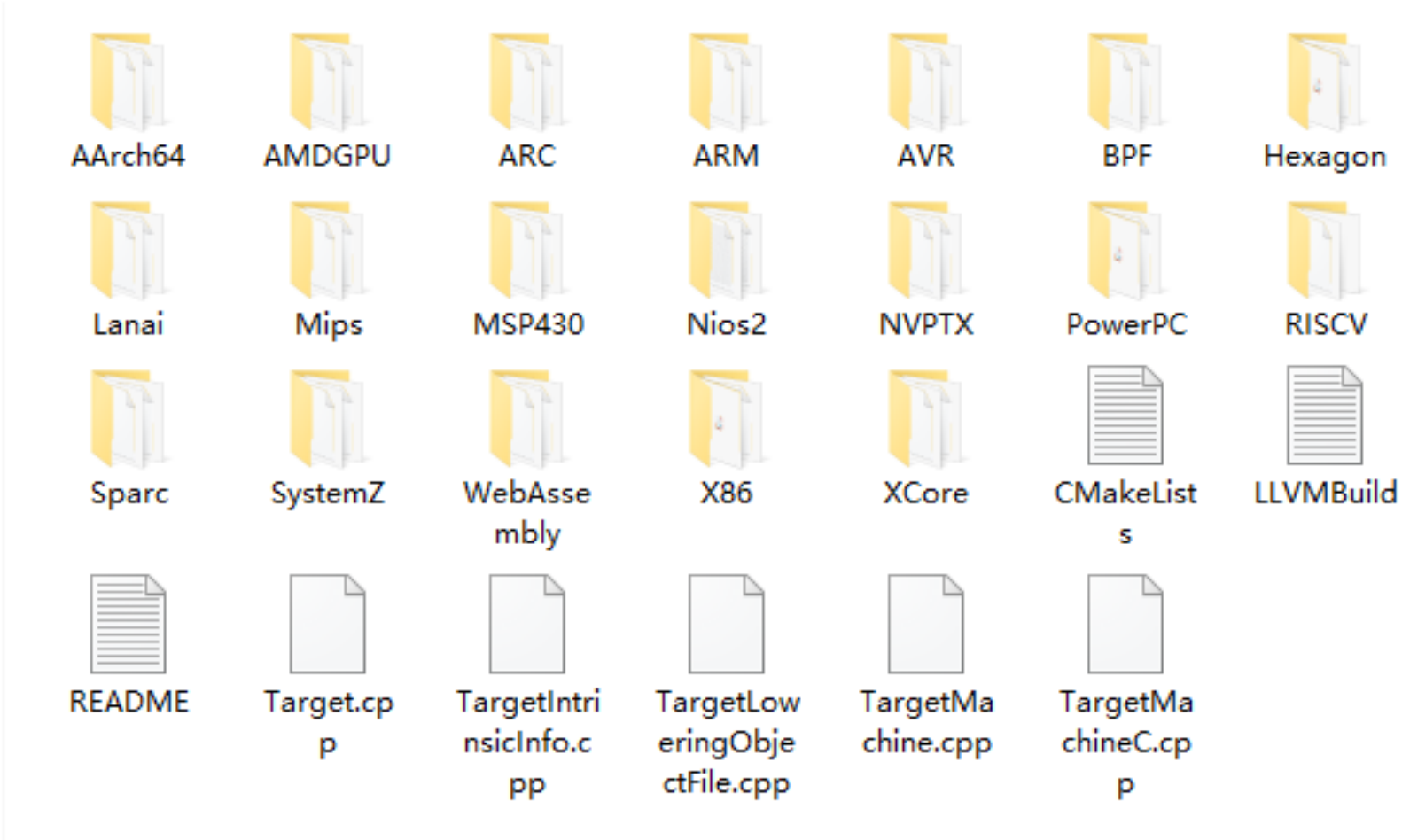
Overview of LLVM Backend

LLVM's Implementation of the Three-Phase Design



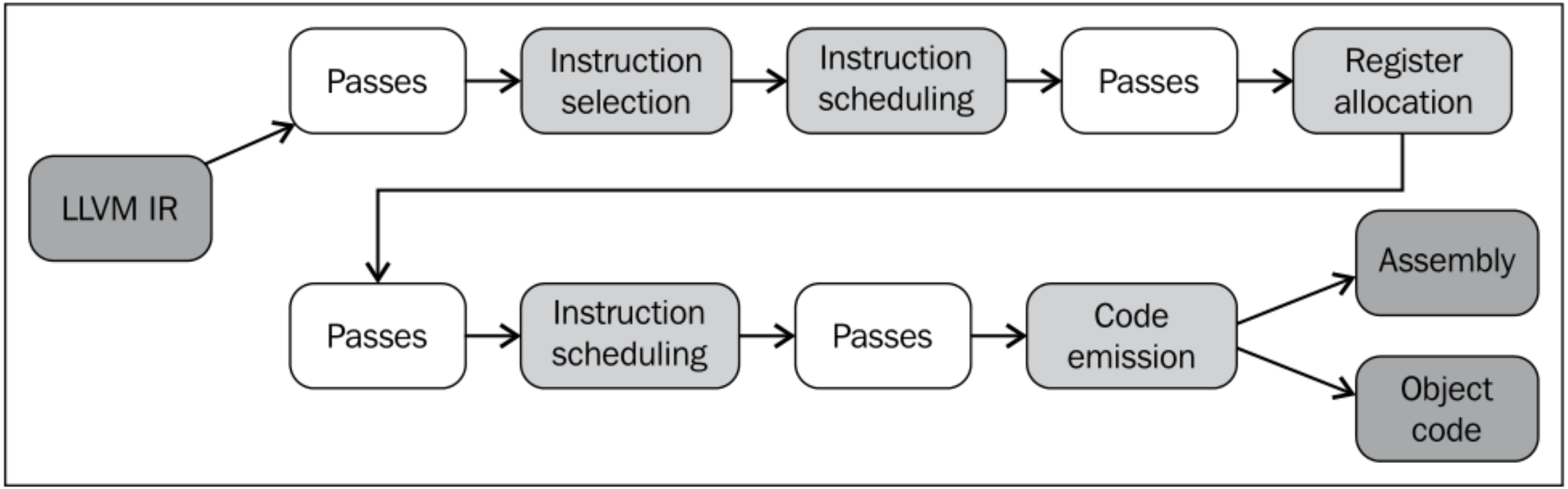
Notes：From LLVM DOC：《Intro to LLVM:Book chapter providing a compiler hacker's introduction to LLVM》

The Backends in LLVM6.0.0



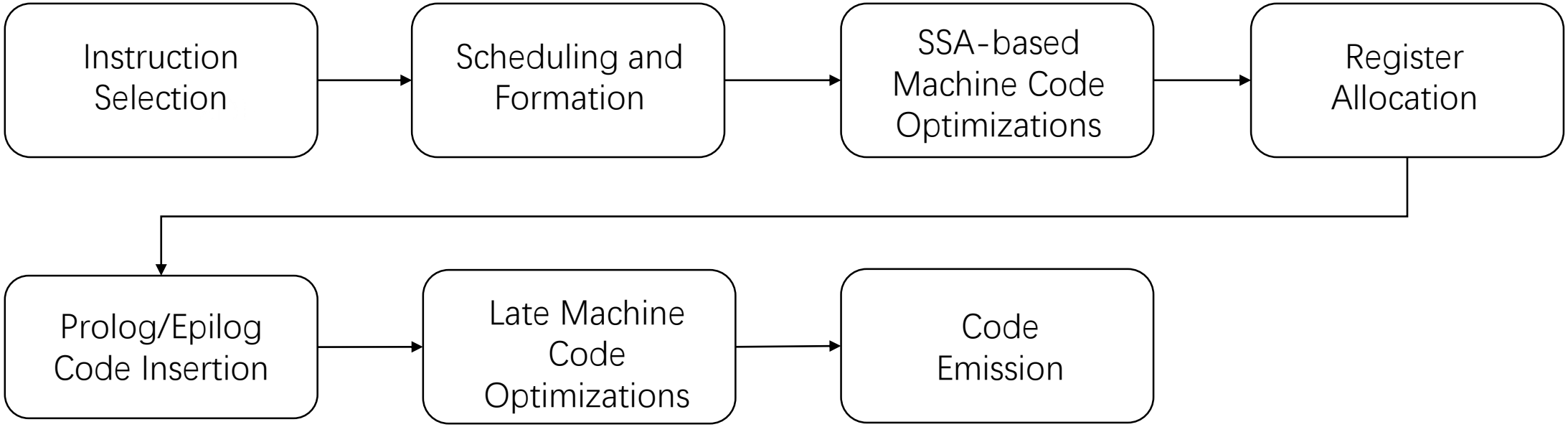
Notes：This is the source code dir of LLVM6.0.0，its location is LLVM/lib/Target/.

The Steps in LLVM Backend



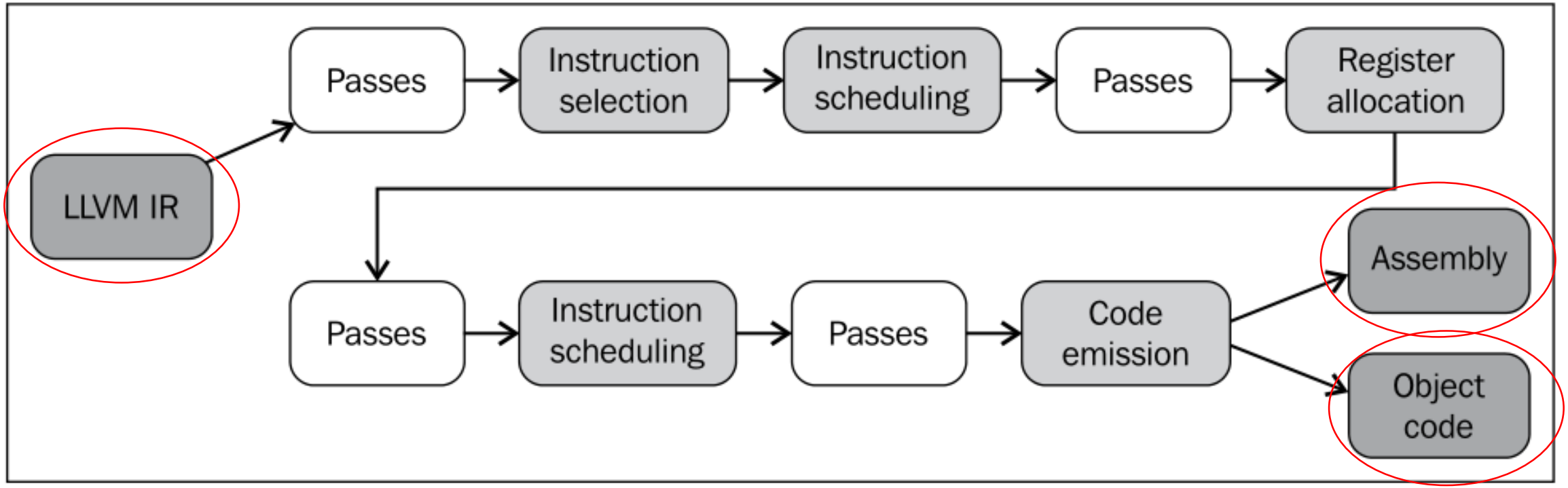
Notes: 《Getting Started with LLVM Core Libraries》P134.

The Steps in LLVM Backend（Another version）

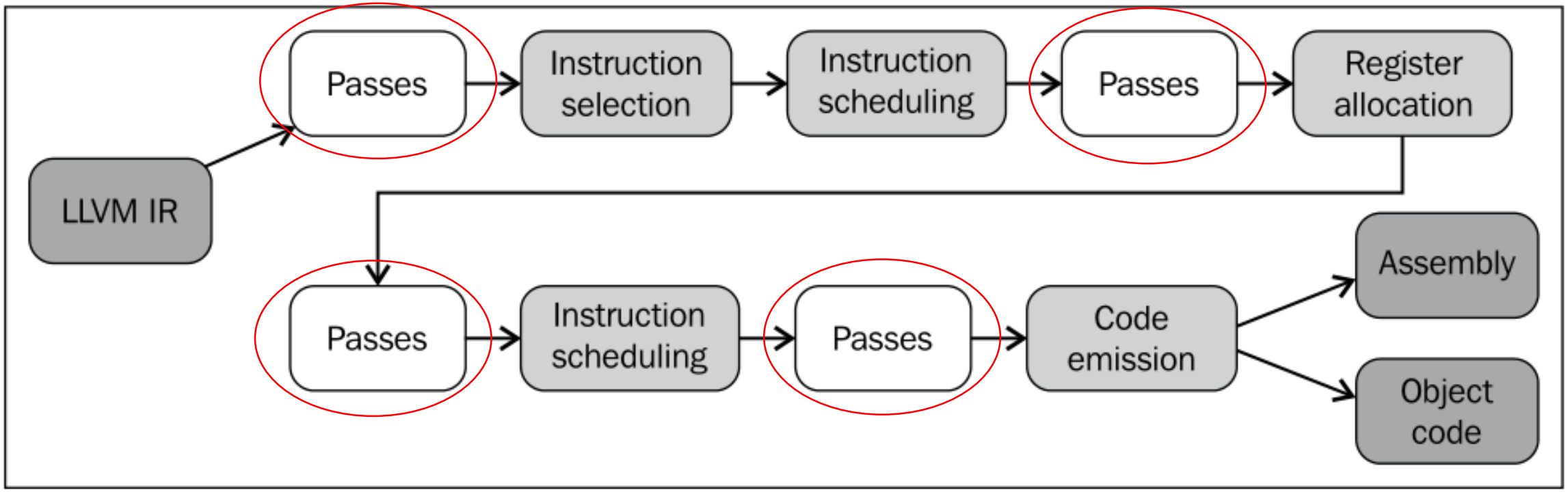


Notes：According the LLVM DOC《The LLVM Target-Independent Code Generator》.

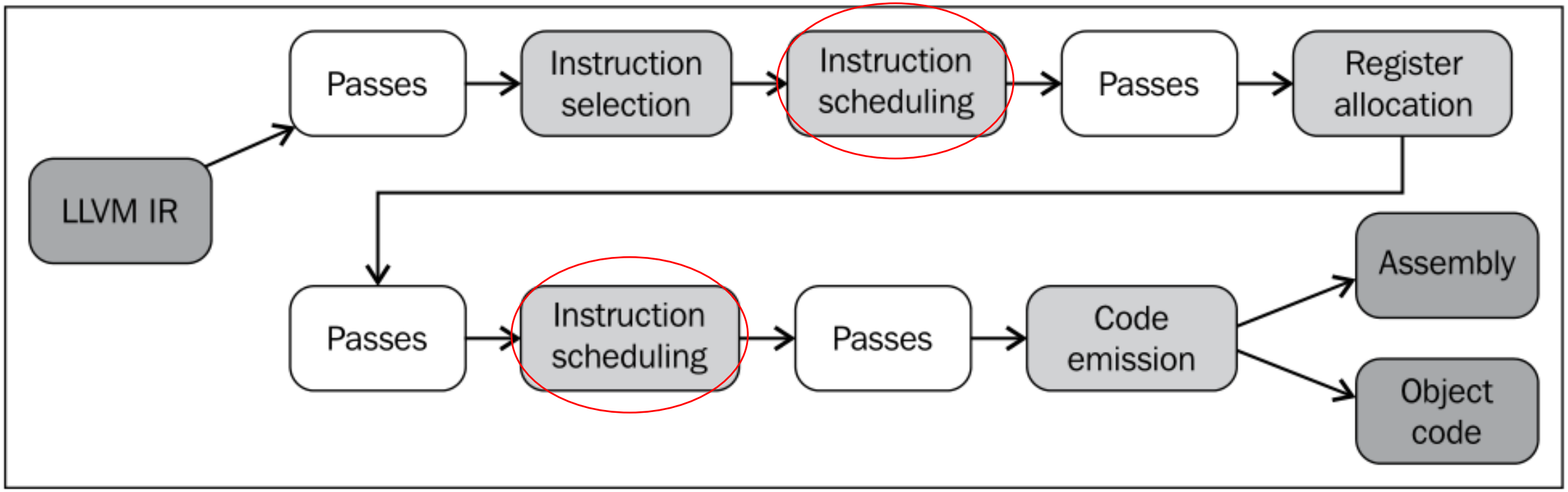
The Steps in LLVM Backend



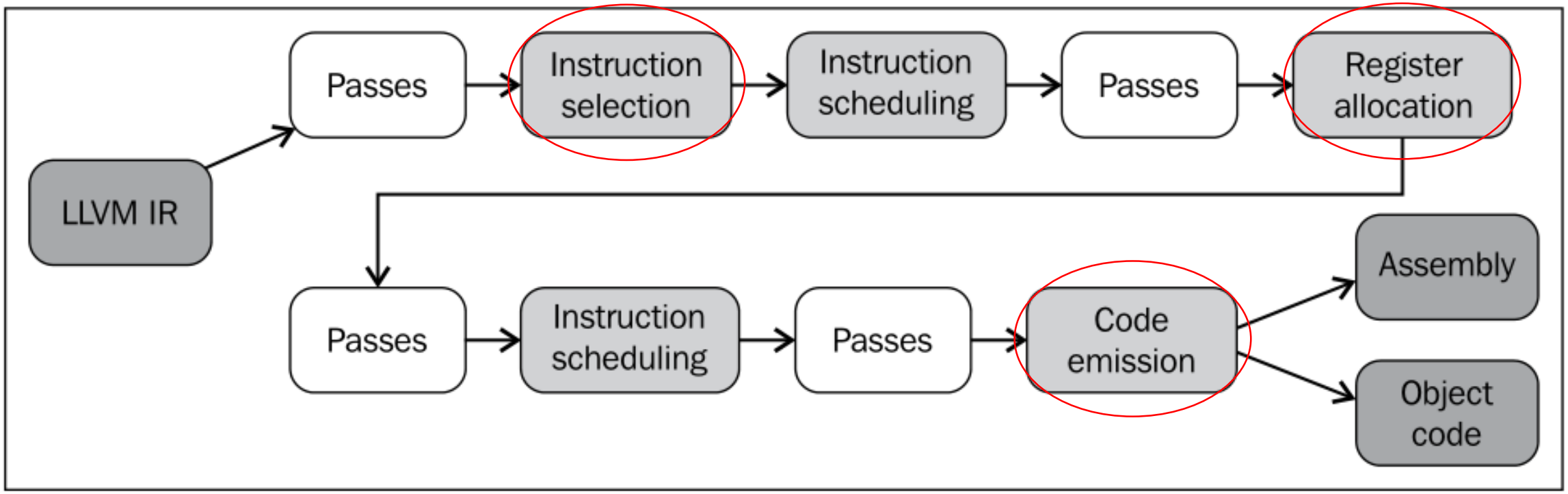
The Steps in LLVM Backend



The Steps in LLVM Backend

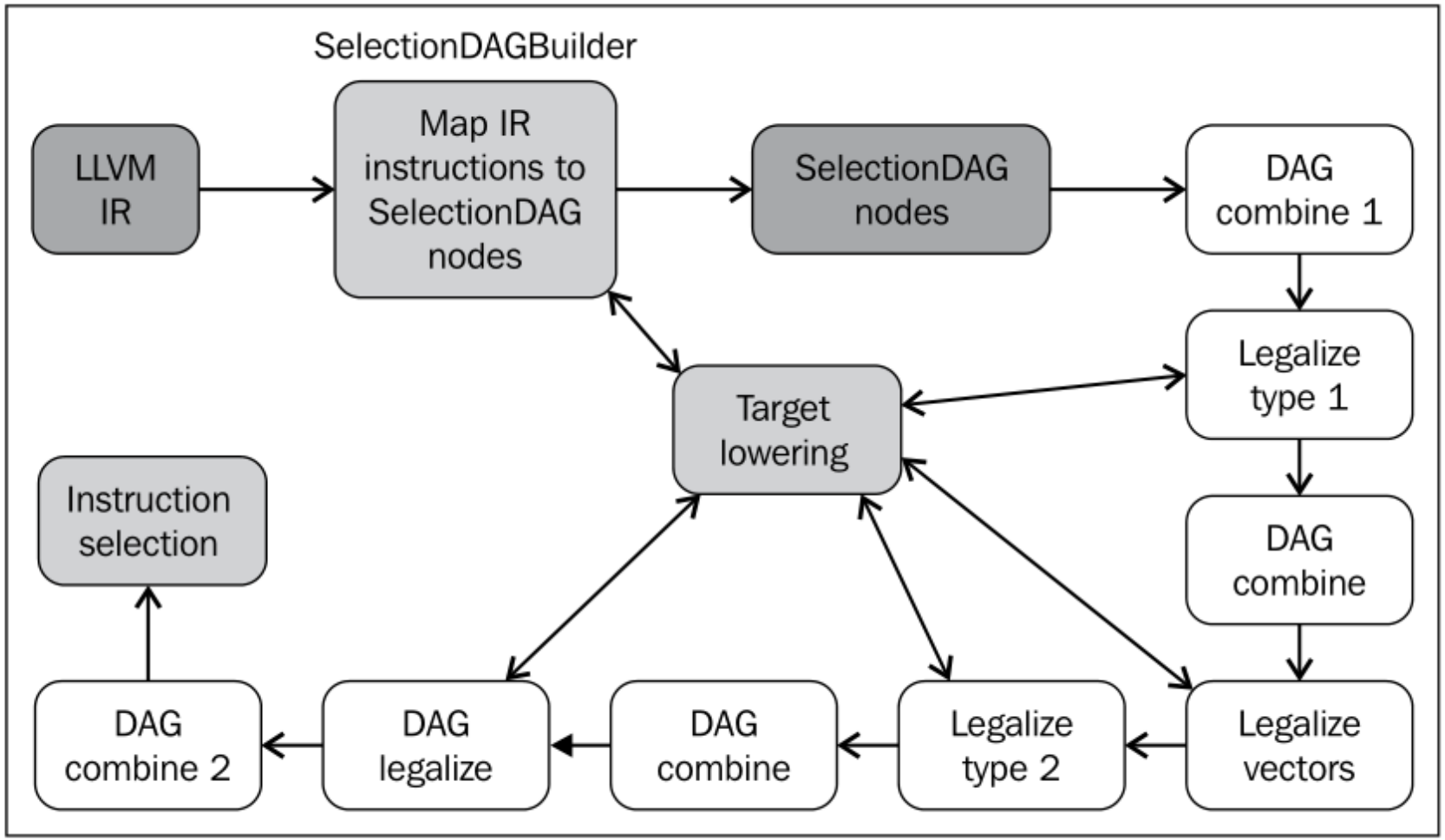


The Important Steps in LLVM Backend



Important Steps of Backend

Instruction selection



Notes: 《Getting Started with LLVM Core Libraries》 P150

Instruction selection

- The TargetLowering class is used by SelectionDAG based instruction selectors primarily to describe how LLVM code should be lowered to SelectionDAG operations. Among other things, this class indicate: which operations are natively supported by the target machine.
- The DAG combine pass optimizes suboptimal SelectionDAG constructions by matching a set of nodes and replacing them with a simpler construct whenever it is profitable.
- The type legalization pass guarantees that instruction selection only needs to deal with legal types. Legal types are the ones natively supported by the target.

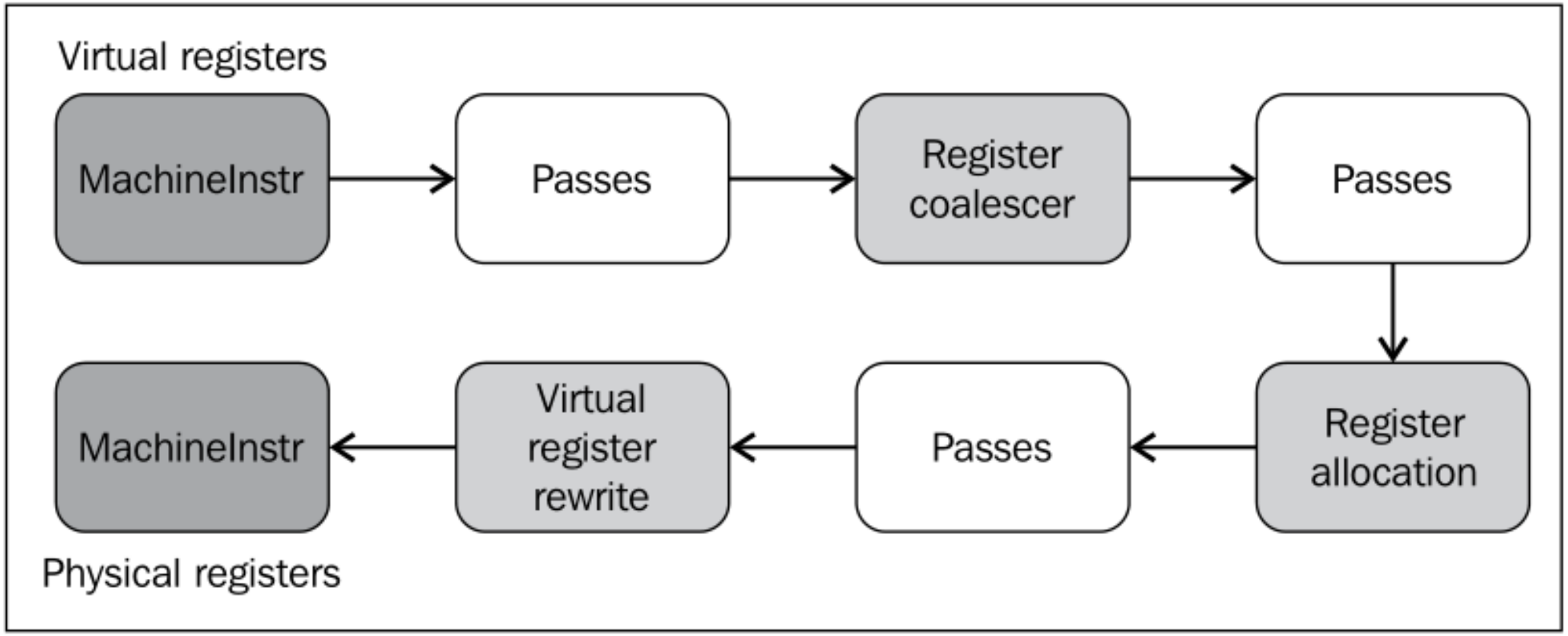
Instruction selection

- The SelectionDAG class employs a DAG to represent the computation of each basic block, and each SDNode corresponds to an instruction or operand.

- The primary payload of the SDNode is its operation code (Opcode) that indicates what operation the node performs and the operands to the operation. The various operation node types are described at the top of the *include/llvm/CodeGen/ISDOpcodes.h* file.

- The SelectionDAG optimization phase(DAG Combiner) is run multiple times for code generation, immediately after the DAG is built and once after each legalization. The first run of the pass allows the initial code to be cleaned up. Subsequent runs of the pass clean up the messy code generated by the Legalize passes, which allows Legalize to be very simple.

Register Allocation



Notes: 《Getting Started with LLVM Core Libraries》P162

Register Allocation

- The register coalesce removes redundant copy instructions(COPY) by joining intervals.

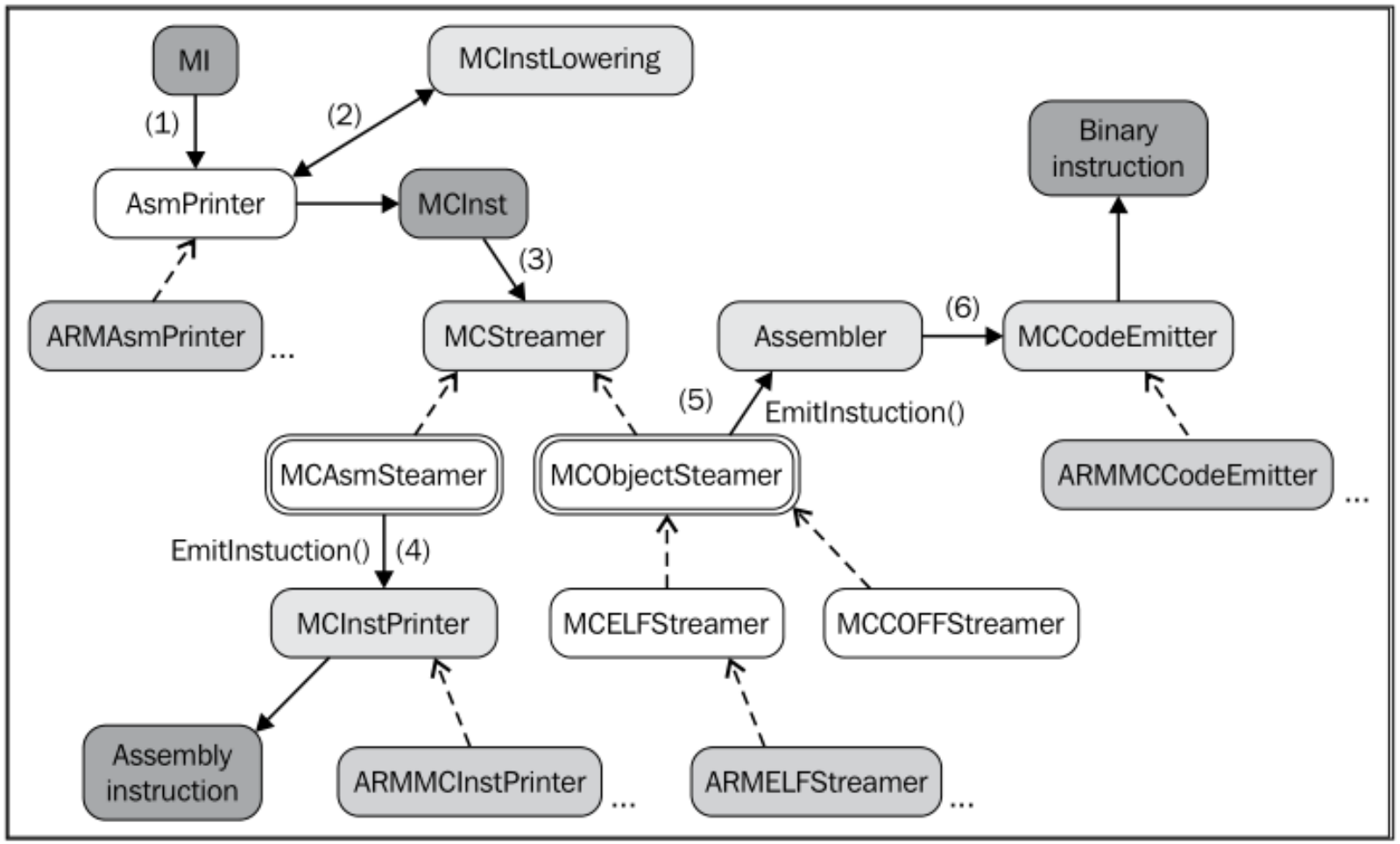
- The register allocation pass selects the physical registers to be used for each virtual one. VirRegMap holds the result from register allocation, containing a map from virtual to physical registers.

- The virtual register rewrite pass uses VirRegMap and replace virtual register references with physical ones.

Register Allocation

- In LLVM, physical registers are denoted by integer numbers that normally range from 1 to 1023.
- Some architectures contain registers that share the same physical location. These physical registers are marked as aliased in LLVM.
- Each virtual register can only be mapped to physical registers of a particular class.
- If the number of physical registers is not enough to accommodate all the virtual registers, some of them will have to be mapped into memory. These virtuals are called spilled virtuals.

Code Emission



Notes: 《Getting Started with LLVM Core Libraries》P170

Code Emission

- MachineInstr is an extremely abstract way of representing machine instructions. In particular, it only keeps track of an opcode number and a set of operands.
- MCInst is a simple class(much more so than MachineInstr) that holds a target-specific opcode and a vector of MCOperands.
- AsmPrinter is a machine function pass that first emits the function header and then iterates over all basic block, dispatching one MI instruction at a time to the EmitInstruction() method for further processing.

Code Emission

- The MCStreamer class processes a stream of MCInst instructions to emit them to the chosen output via two subclasses: MCAsmStreamer and MCObjectStreamer.

The Special Backends in History

The Special Backends in History

- CBackend (LLVM3.0)

CBackend/	142037	6 years	void	Creating release_30 branch

- CPPBackend (LLVM 3.8)

CppBackend/	257630	2 years	hans	Creating release_38 branch off revision 257626

The C backend does not require register allocation, instruction selection, or any of the other standard components provided by the system. As such, it only implements these two interfaces (TargetMachine and DataLayout), and does its own thing. Note that C backend was removed from the trunk since LLVM 3.1 release. ——《The LLVM Target-Independent Code Generator》

Notes：The screen shots from http://llvm.org/viewvc/llvm-project/.

The Special Backends in History

- LLVMTargetMachine is designed as a base class for targets implemented with the LLVM target-independent code generator. LLVMTargetMachine is defined as a subclass of a TargetMachine in *include/llvm/Target/TargetMachine.h*.

- The DataLayout class is the only required target description class. DataLayout specifies information about how the target lays out memory for structures, the alignment requirements for various data types, the size of pointers in the target, and whether the target is little-endian or big-endian.

Thanks!

2018-7-28