拥抱方舟开源编译器：Maple IR 分析及 Toy Runtime 介绍

史宁宁

中科院软件所智能软件研究中心程序语言与编译技术实验室

软件绿色联盟
Software Green Alliance

目 录

1 方舟编译器概况

2 MAPLE IR的设计与实现

3 MIR与其他IR的横向对比

4 Phase体系的设计与实现

5 Toy Runtime简介

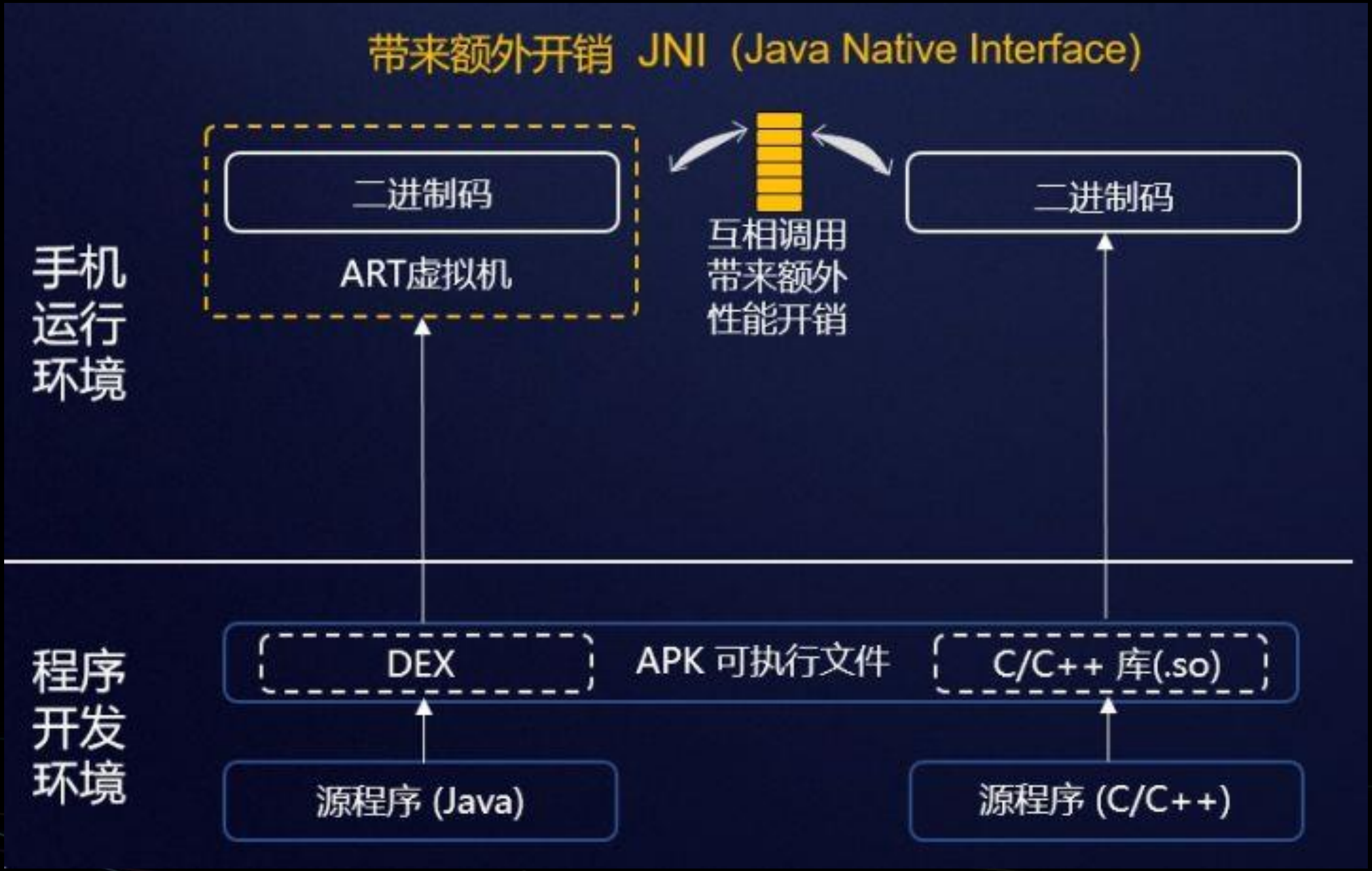
方舟编译器概况

方舟编译器概述

方舟编译器是为支持多种编程语言、多种芯片平台的联合编译、运行而设计的统一编程平台，包含编译器、工具链、运行时等关键部件。 方舟编译器还在持续演进中，陆续将上述能力实现和开源。

From：https://www.openarkcompiler.cn/home

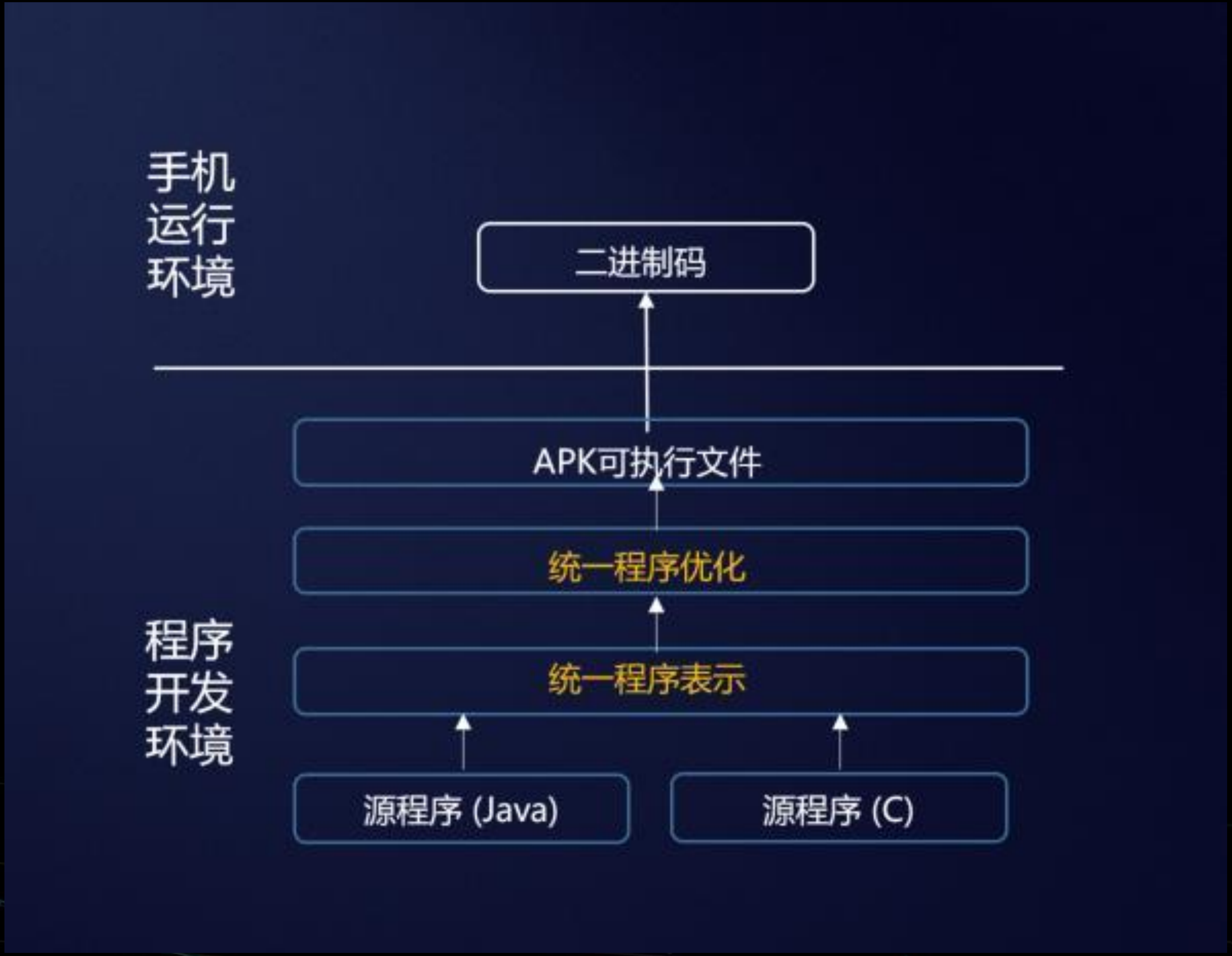
APP开发及运行过程



带来额外开销 JNI（Java Native Interface）

From: https://zhuanlan.zhihu.com/p/77108133

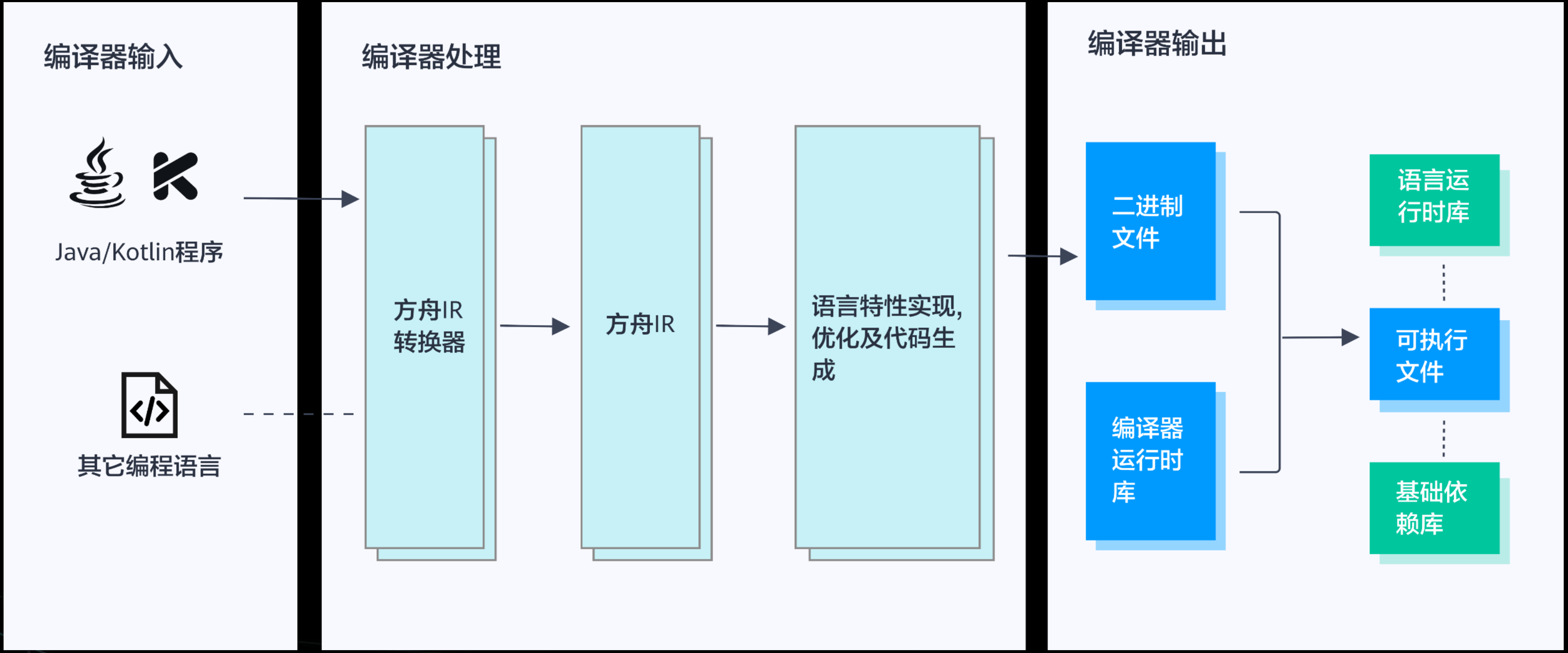
APP开发及运行过程



From: https://zhuanlan.zhihu.com/p/77108133

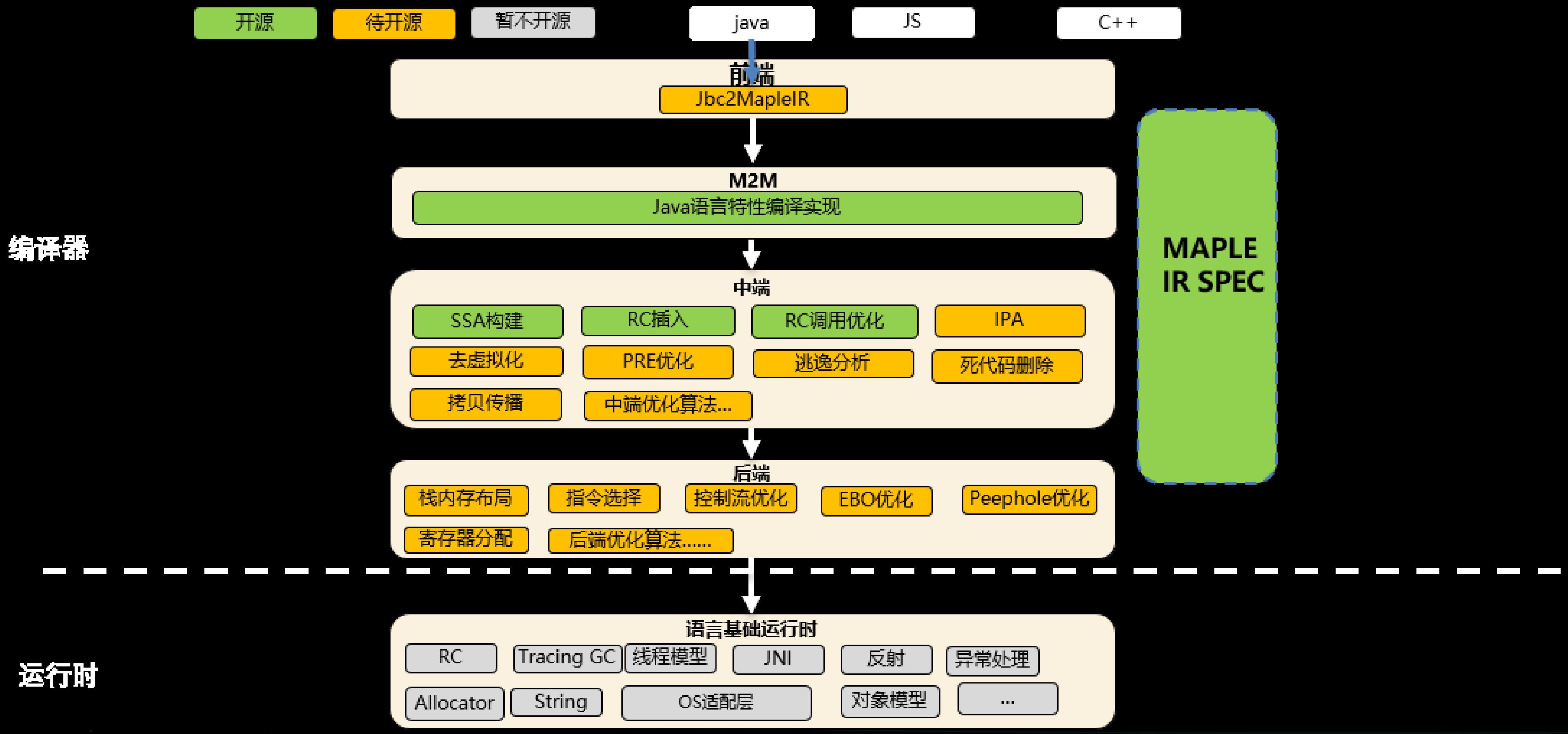
方舟编译器架构示意图

软件绿色联盟
Software Green Alliance



From: https://www.openarkcompiler.cn/document/frameworkDesgin

方舟编译器8月开源状况



🟩 开源
🟨 待开源
⬜ 暂不开源

java
JS
C++

编译器

前端
Jbc2MapleIR

M2M
Java语言特性编译实现

中端
SSA构建
RC插入
RC调用优化
IPA
去虚拟化
PRE优化
逃逸分析
死代码删除
拷贝传播
中端优化算法...

后端
栈内存布局
指令选择
控制流优化
EBO优化
Peephole优化
寄存器分配
后端优化算法......

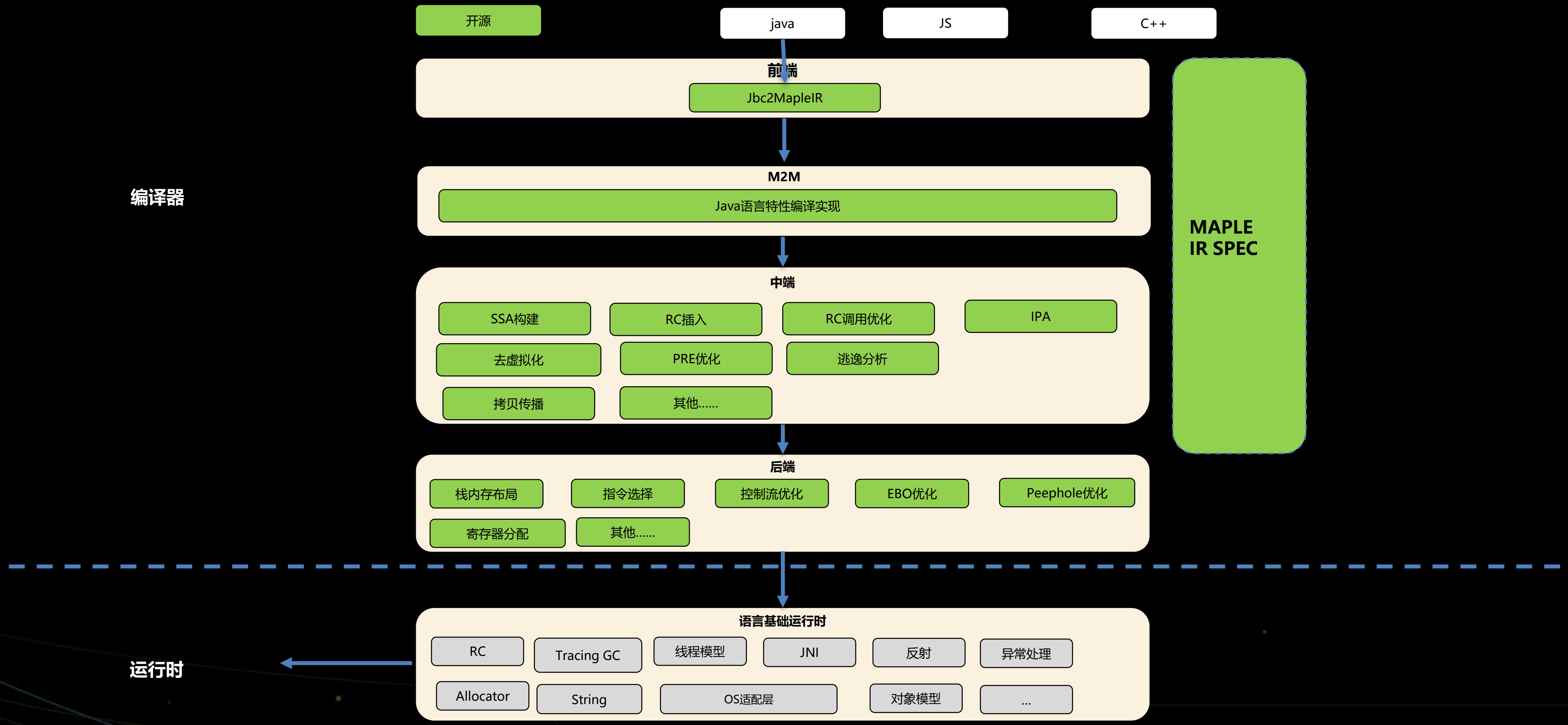
MAPLE IR SPEC

运行时

语言基础运行时
RC
Tracing GC
线程模型
JNI
反射
异常处理
Allocator
String
OS适配层
对象模型
...

From: 赵俊民《方舟基础架构与IR中间表示详细介绍》

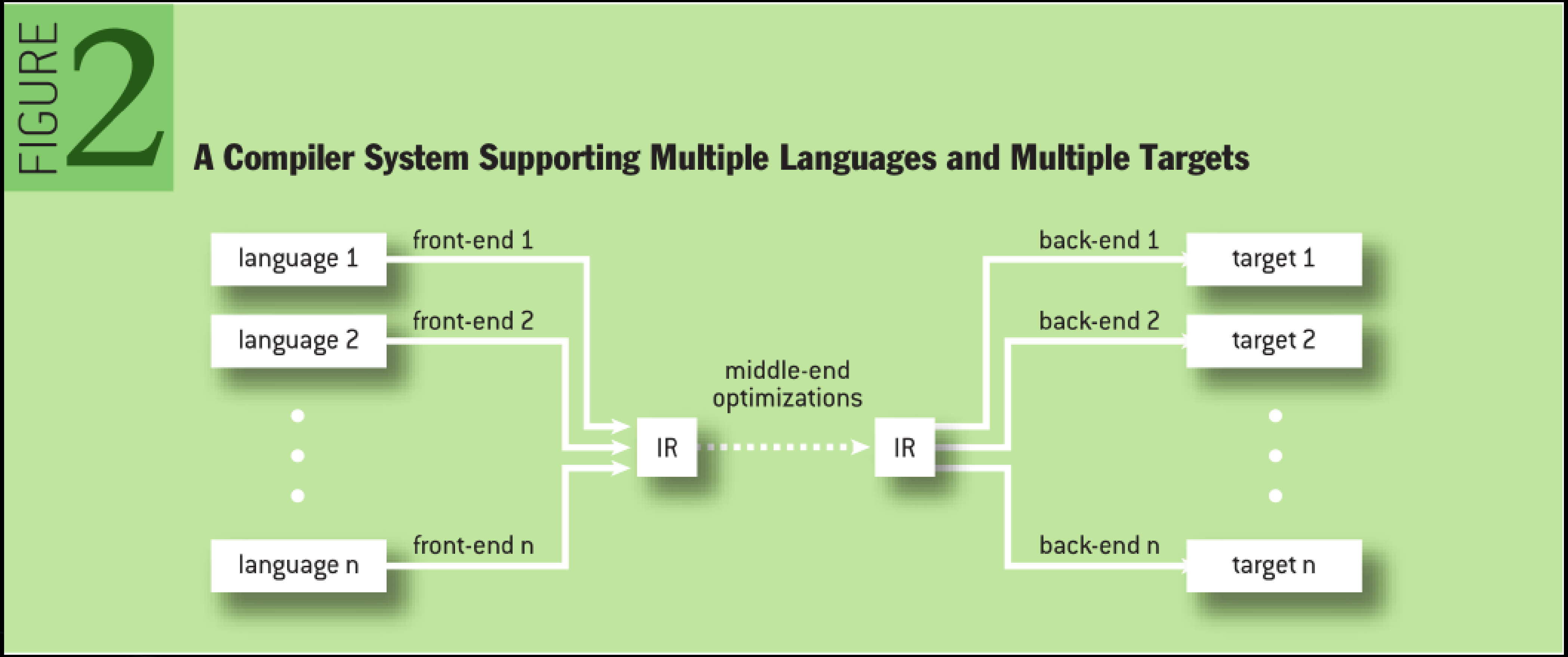
方舟编译器开源后续计划（2020）



From: 赵俊民《方舟基础架构与IR中间表示详细介绍》

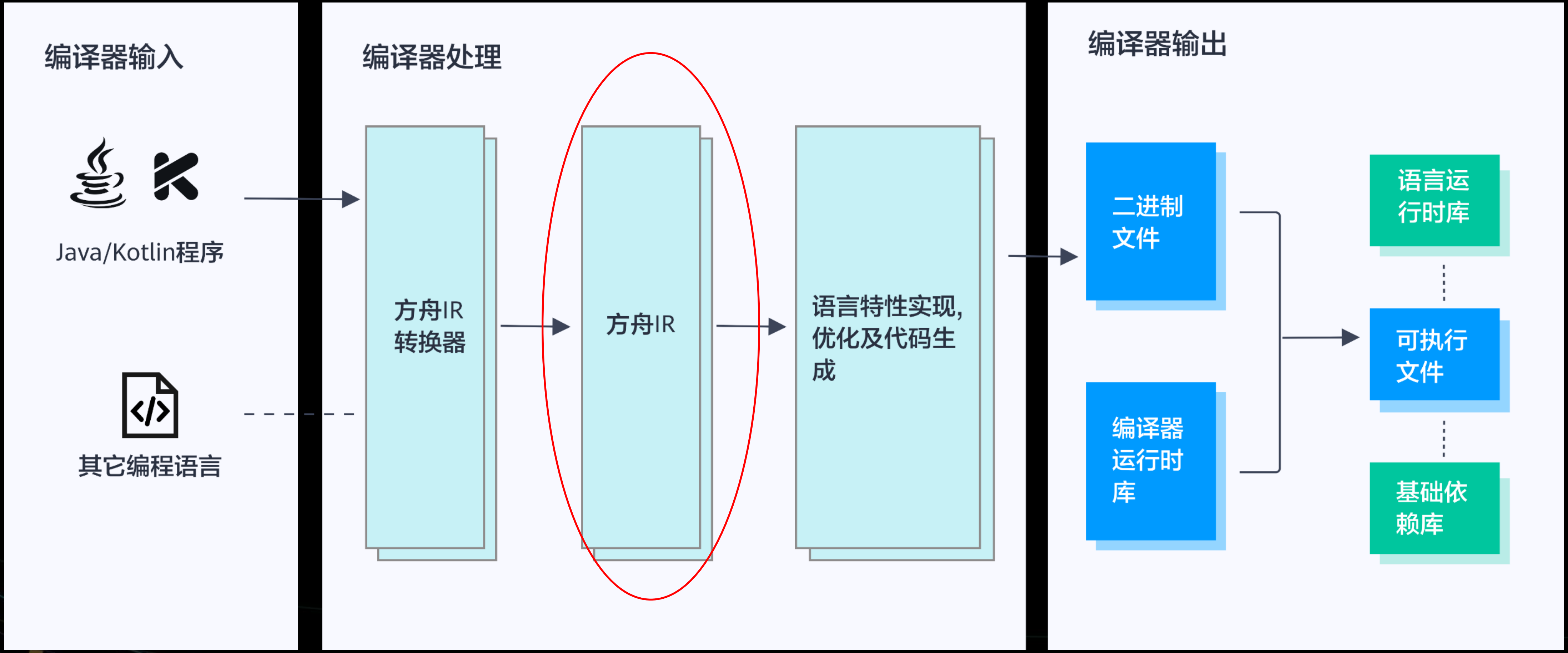
MAPLE IR的设计与实现

MAPLE IR的设计起源



From: Fred Chow 《The increasing significance of intermediate representations in compilers》

MAPLE IR在方舟编译器中的位置



From: https://www.openarkcompiler.cn/document/frameworkDesgin

方舟编译器的多层IR设计

1. MAPLE IR 's program representation at the highest level exhibits the following characteristics : many language constructs, short code sequences, constructs are hierarchical and no loss of program information.

2. At the lower levels, general purpose optimizations are performed. In particular, at the lowest level, MAPLE IR instructions map one-to-one to machine instructions most of the time, for the mainstream processor ISAs.

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/doc/en/MapleIRDesign.md

方舟编译器的多层IR设计

1. MAPLE IR represents program code intrinsically in the form of trees. At the highest level, it honors the hierarchical form of the program as it exists at the source level via the tree representation. It also honors the abstract operations defined by the language. As compilation proceeds, the abstract operations are lowered into general-purpose operations that require longer code sequences. The program structure also becomes more flat, as general-purpose processors work by executing lists of instructions sequentially.

2. Though MAPLE IR is target-independent at the highest level, the lowering process will make it become target-dependent.

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/doc/en/MapleIRDesign.md

方舟编译器的多层IR设计

1. 高层IR更接近于源程序，包含了更多的程序信息；
2. 底层IR更接近于目标平台的机器指令，甚至有的时候和和机器指令是一对一的关系；
3. 高层IR的保留了程序语言的层次结构，和目标机器平台无关；
4. 底层IR更加扁平化，依赖具体的目标平台。

多层IR设计的特点

优点：

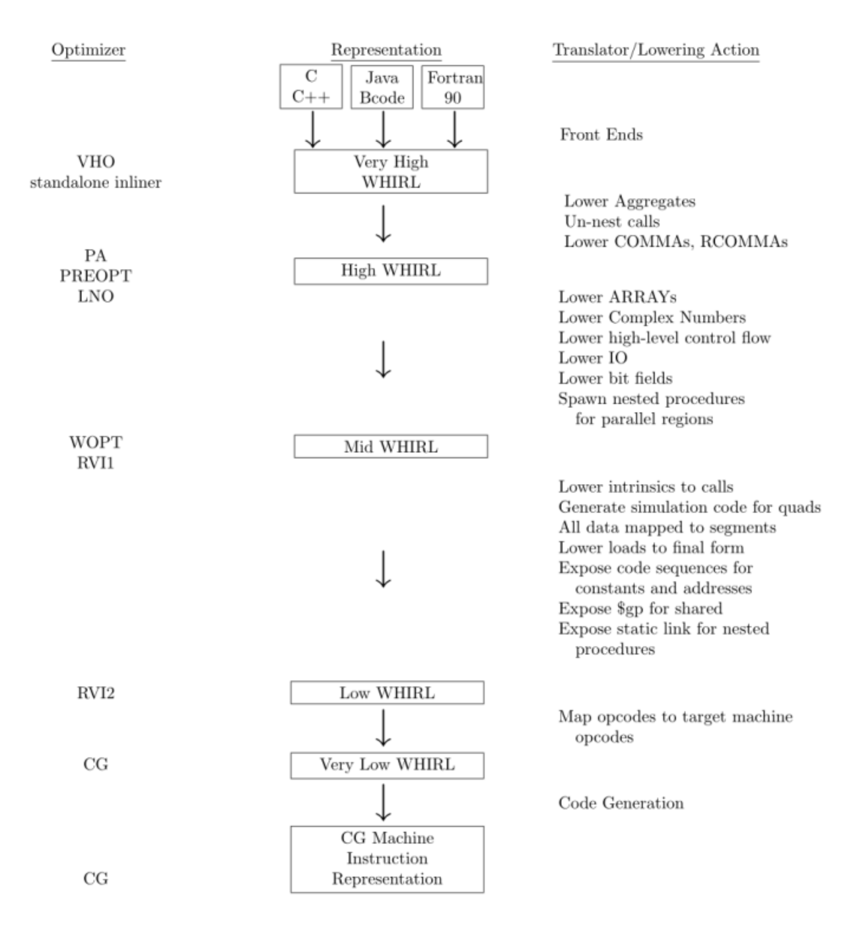
- 可以提供更多的源程序信息；
- IR表达上更加地灵活，更加方便优化；
- 使得优化算法更加地高效；
- 可以将优化算法的负面影响降到最低。

缺点：

- 底层IR的优化器将面临更多的可能，增加了特定语义的识别难度。

方舟编译器多层IR的分层

Open64多层IR的分层



Optimizer | Representation | Translator/Lowering Action

C C++ | Java Bcode | Fortran 90

Front Ends

VHO standalone inliner — Very High WHIRL

Lower Aggregates
Un-nest calls
Lower COMMAs, RCOMMAs

PA
PREOPT
LNO — High WHIRL

Lower ARRAYs
Lower Complex Numbers
Lower high-level control flow
Lower IO
Lower bit fields
Spawn nested procedures for parallel regions

WOPT
RVI1 — Mid WHIRL

Lower intrinsics to calls
Generate simulation code for quads
All data mapped to segments
Lower loads to final form
Expose code sequences for constants and addresses
Expose $gp for shared
Expose static link for nested procedures

RVI2 — Low WHIRL

Map opcodes to target machine opcodes

CG — Very Low WHIRL

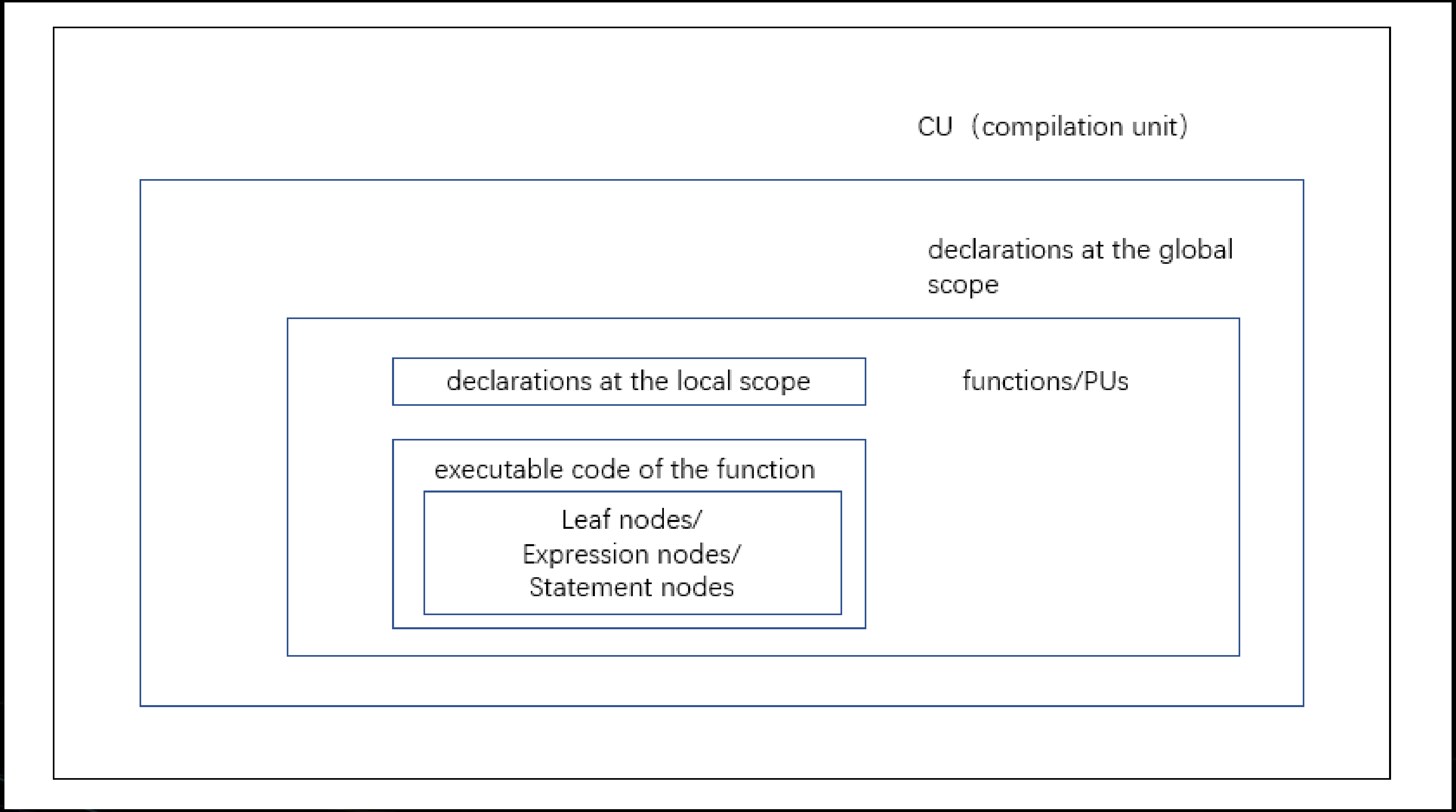
Code Generation

CG — CG Machine Instruction Representation

方舟编译器多层IR的分层

From: 《Open64 Compiler Whirl Intermediate Representation》

MAPLE IR的结构



CU（compilation unit）

declarations at the global scope

functions/PUs

declarations at the local scope

executable code of the function

Leaf nodes/
Expression nodes/
Statement nodes

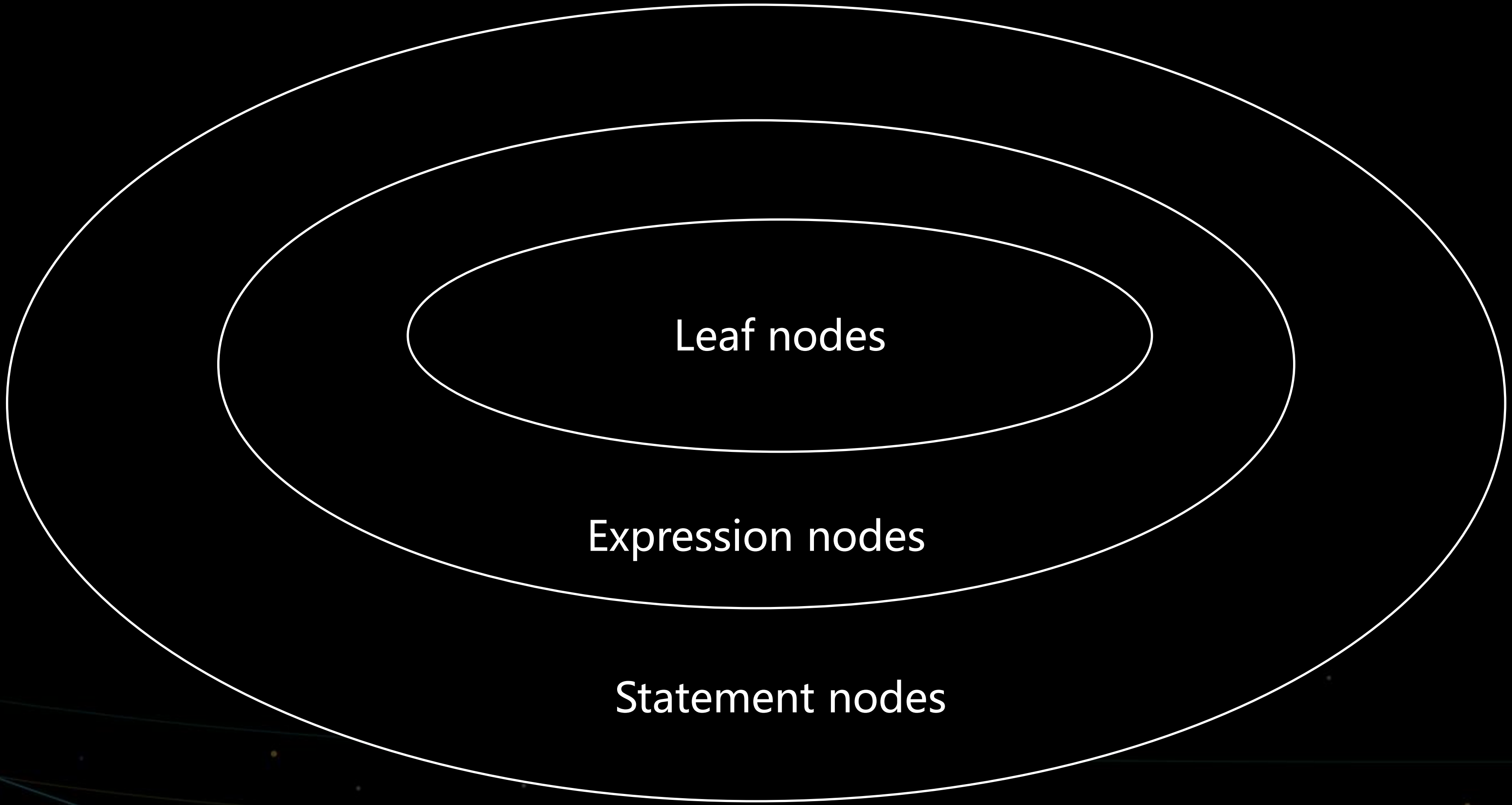
语句系统

There are three kinds of executable nodes in MAPLE IR:

- Leaf nodes - Also called terminal nodes, these nodes denote a value at execution time, which may be a constant or the value of a storage unit.
- Expression nodes - An expression node performs an operation on its operands to compute a result. Its result is a function of the values of its operands and nothing else. Each operand can be either a leaf node or another expression node. Expression nodes are the internal nodes of expression trees. The type field in the expression node gives the type associated with the result of the operation.
- Statement nodes - These represent the flow of control. Execution starts at the entry of the function and continues sequentially statement by statement until a control flow statement is executed. Apart from modifying control flow, statements can also modify data storage in the program. A statement nodes has operands that can be leaf, expression or statement.

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/doc/en/MapleIRDesign.md

语句系统



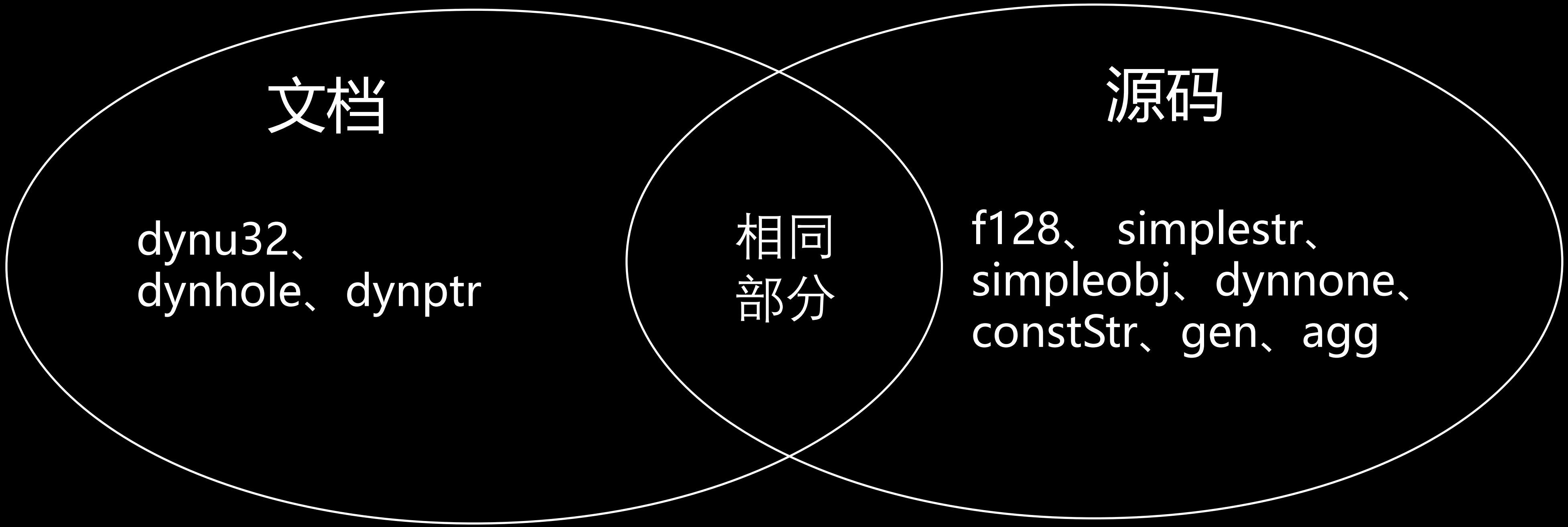
基本类型

文档中的基本类型：

- no type - void
- signed integers - i8, i16, i32, i64
- unsigned integers - u8, u16, u32, u64
- booleans- u1
- addresses - ptr, ref, a32, a64
- floating point numbers - f32, f64
- complex numbers - c64, c128
- javascript types - dynany、dynu32、dyni32、dynundef、dynnull、dynhole、dynbool、dynptr、dynf64、dynf32、dynstr、dynobj
- SIMD types - (to be defined)
- unknown

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/doc/en/MapleIRDesign.md

文档和源码中的基本类型对比



文档

dynu32、
dynhole、dynptr

相同
部分

源码

f128、 simplestr、
simpleobj、dynnone、
constStr、gen、agg

文档中的控制流语句1

Hierarchical control flow statements：

- doloop
- dowhile
- foreachelem
- if
- while

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/doc/en/MapleIRDesign.md

文档中的控制流语句2

Flat control flow statements：

- brfalse
- brtrue
- multiway
- return
- switch
- goto
- rangegoto
- indexgoto

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/doc/en/MapleIRDesign.md

源码中的控制流语句1

hierarchical control flow opcodes

```
 OPCODE(block, BlockNode, (OPCODEISSTMT | OPCODENOTMMPL), 0)
OPCODE(doloop, DoloopNode, (OPCODEISSTMT | OPCODENOTMMPL), 0)
OPCODE(dowhile, WhileStmtNode, (OPCODEISSTMT | OPCODENOTMMPL), 0)
OPCODE(if, IfStmtNode, (OPCODEISSTMT | OPCODENOTMMPL), 0)
OPCODE(while, WhileStmtNode, (OPCODEISSTMT | OPCODENOTMMPL), 0)
OPCODE(switch, SwitchNode, (OPCODEISSTMT | OPCODENOTMMPL), 8)
OPCODE(multiway, MultiwayNode, (OPCODEISSTMT | OPCODENOTMMPL), 8)
OPCODE(foreachelem, ForeachelemNode, (OPCODEISSTMT | OPCODENOTMMPL), 0)
```

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/src/maple_ir/include/opcodes.def

源码中的控制流语句2

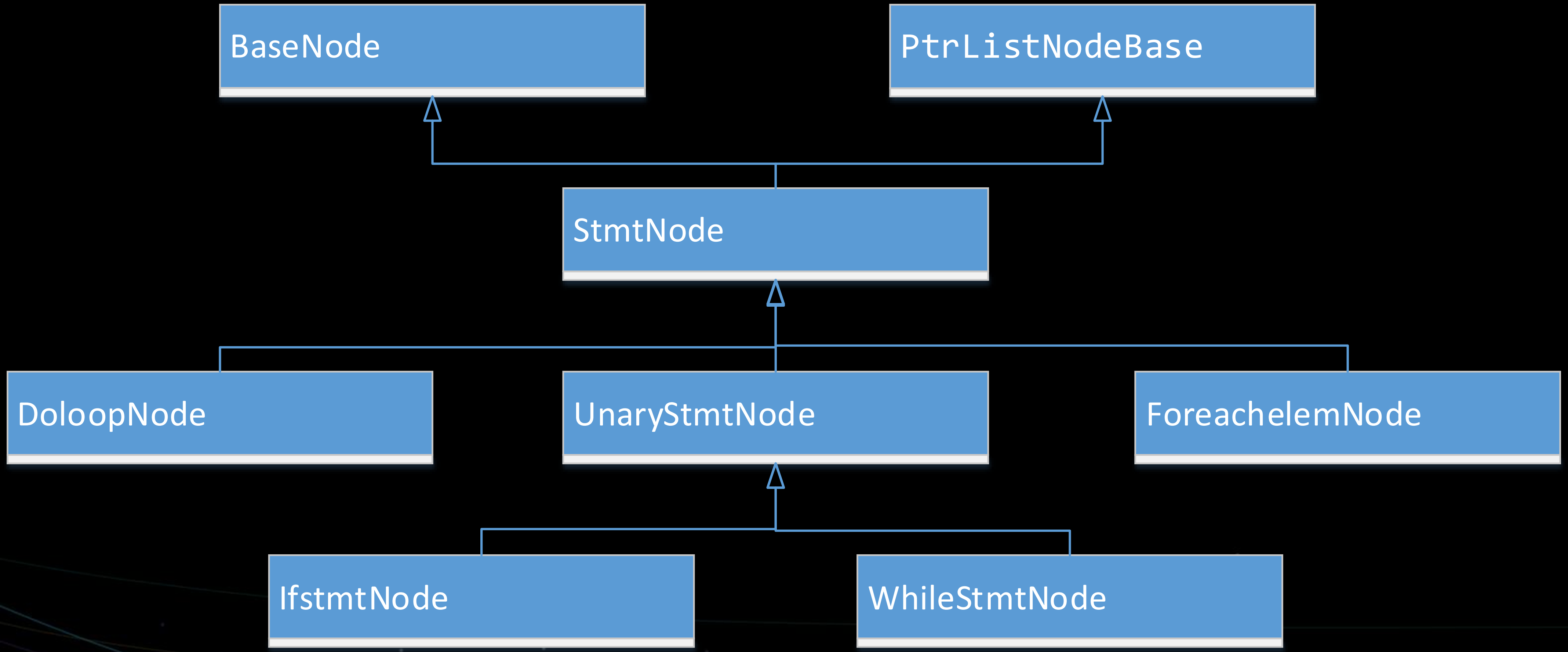
flat control flow opcodes

```
OPCODE(goto, GotoNode, OPCODEISSTMT, 8)
OPCODE(brfalse, CondGotoNode, OPCODEISSTMT, 8)
OPCODE(brtrue, CondGotoNode, OPCODEISSTMT, 8)
OPCODE(return, NaryStmtNode, (OPCODEISSTMT | OPCODEISVARSIZE |
    OPCODEHASSSAUSE), 0)
OPCODE(rangegoto, RangegotoNode, OPCODEISSTMT, 8)
```

From: https://gitee.com/harmonyos/OpenArkCompiler/blob/master/src/maple_ir/include/opcodes.def

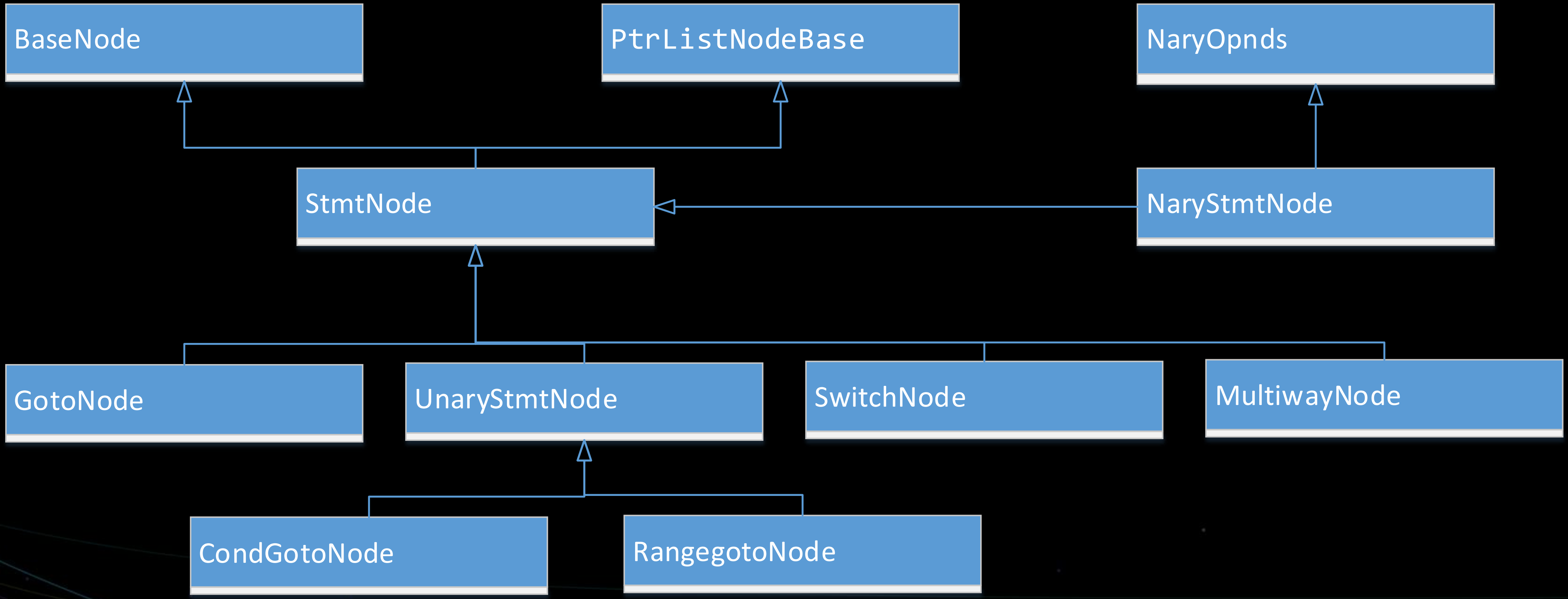
控制流语句对应的节点实现1

软件绿色联盟
Software Green Alliance



控制流语句对应的节点实现2

软件绿色联盟
Software Green Alliance



MIR与其它IR的横向对比

MIR与LLVM IR的基本类型对比

序号	方舟编译器类型类别	方舟编译器类型	LLVM类型类别	LLVM类型类别	LLVM类型类别	LLVM类型
1	no type	void				void
2	signed integers	i8	First Class Types	Single Value Types	Integer Type	iN
3		i16				
4		i32				
5		i64				
6	unsigned integers	u8				
7		u16				
8		u32				
9		u64				
10	booleans	u1				
11	addresses	ptr	First Class Types	Single Value Types	Pointer Type	<type> *
12		ref				
13		a32				
14		a64				

MIR与LLVM IR的基本类型对比（续）

序号	方舟编译器类型类别	方舟编译器类型	LLVM类型类别	LLVM类型类别	LLVM类型类别	LLVM类型
15	floating point numbers	f32	First Class Types	Single Value Types	Floating-Point Types	half
16		f64				float
17	complex numbers	c64				double
18		c128				fp128
19	javascript types	dynany				x86_fp80
20		dynu32				ppc_fp128
21		dyni32				
22		dynundef				
23		dynnull				
24		dynhole				
25		dynbool				
26		dynptr				
27		dynf64				
28		dynf32				
29		dynstr				
30		dynobj				
31	SIMD types	to be defined				
32		unknown				

MIR与LLVM IR的基本类型对比（续）

序号	方舟编译器类型类别	方舟编译器类型	LLVM类型类别	LLVM类型类别	LLVM类型类别	LLVM类型
33					Function Type	<returntype> (<parameter list>)
34			First Class Types	Single Value Types	X86_mmx Type	x86_mmx
35					Vector Type	< <# elements> x <elementtype> > ; Fixed-length vector < vscale x <# elements> x <elementtype> > ; Scalable vector
36				Label Type		label
37				Token Type		token
38				Metadata Type		metadata
39				Aggregate Types	Array Type	[<# elements> x <elementtype>]
40					Structure Type	%T1 = type { <type list> } ; Identified normal struct type %T2 = type <{ <type list> }> ; Identified packed struct type
41					Opaque Structure Types	%X = type opaque %52 = type opaque

MIR与LLVM IR的基本类型对比（续）

总结：

MAPLE IR和LLVM IR的基本类型设计思想不同，所以二者的基本类型采用的是不同的风格，基本没有相同的类型表示。

MIR与WHIRL IR的基本类型对比

序号	MAPLE IR基本类型类别	MAPLE IR基本类型	WHIRL IR 基本类型
1	no type	void	V
2	signed integers	i8	I1
3		i16	I2
4		i32	I4
5		i64	I8
6	unsigned integers	u8	U1
7		u16	U2
8		u32	U3
9		u64	U4
10	booleans	u1	B
11	addresses	ptr	
12		ref	
13		a32	A4
14		a64	A8

MIR与WHIRL IR的基本类型对比（续）

序号	MAPLE IR基本类型类别	MAPLE IR基本类型	WHIRL IR 基本类型
15	floating point numbers	f32	F4
16		f64	F8
17			F10
18			F16
19			FQ
20	complex numbers		C4
21		c64	C8
22		c128	CQ
23	javascript types	dynany	
24		dynu32	
25		dyni32	
26		dynundef	
27		dynnull	
28		dynhole	
29		dynbool	
30		dynptr	
31		dynf64	
32		dynf32	
33		dynstr	
34		dynobj	

MIR与WHIRL IR的基本类型对比（续）

序号	MAPLE IR基本类型类别	MAPLE IR基本类型	WHIRL IR 基本类型
35	SIMD types	to be defined	
36		unknown	
37			M
38			BS

总结：除去javascript的专用基本类型，二者有16/26种基本类型设计一致。

MIR与WHIRL IR的控制流语句对比

序号	MAPLE IR	WHIRL IR
1	doloop	DOLOOP
2	dowhile	DOWHILE
3	foreachelem	
4	if	IF
5	while	WHILEDO
6		FUNCENTRY
7		BLOCK
8		REGION

软件绿色联盟
Software Green Alliance

MIR与WHIRL IR的控制流语句对比（续）

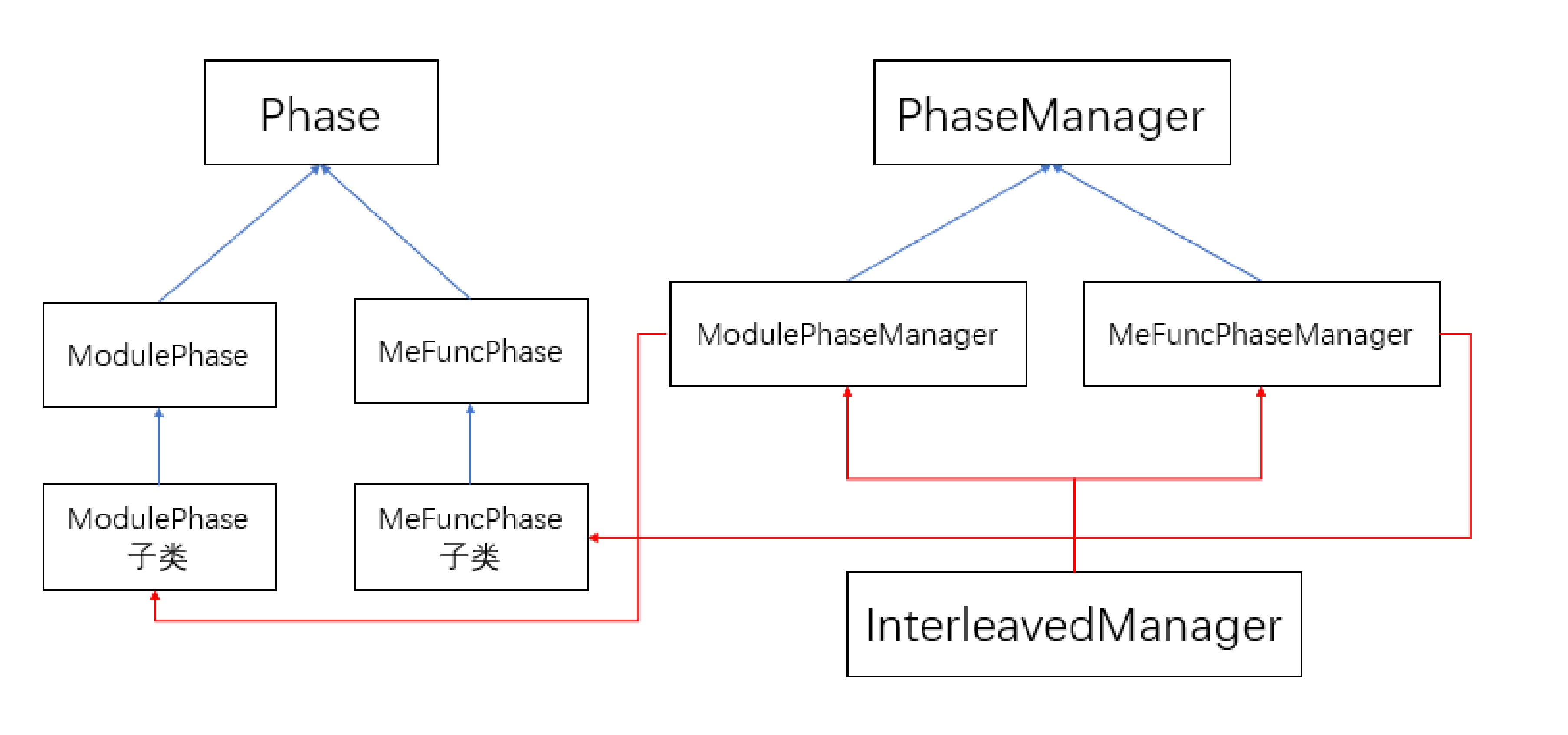
序号	MAPLE IR	WHIRL IR
9	brfalse	FALSEBR
10	brtrue	TRUEBR
11	multiway	
12	return	RETURN
13		RETURN_VAL
14	switch	SWITCH
15	goto	GOTO
16	rangegoto	
17	indexgoto	
18		GOTO_OUTER_BLOCK
19		CASEGOTO
20		COMPGOTO
21		XGOTO
22		AGOTO
23		REGION_EXIT
24		ALTENTRY
25		LABEL
26		LOOP_INFO

总结：除了个别控制流语句之外，MAPLE IR基本上是WHIRL IR的一个子集。

软件绿色联盟
Software Green Alliance

Phase体系的设计与实现

Phase体系的设计与实现



Phase列表（phases.def）

```
ADD_PHASE("classhierarchy", true)
ADD_PHASE("vtableanalysis", true)
ADD_PHASE("reflectionanalysis", true)
ADD_PHASE("gencheckcast", true)
ADD_PHASE("javaintrnlowering", true)
// mephase begin
ADD_PHASE("ssatab", true)
ADD_PHASE("aliasclass", true)
ADD_PHASE("ssa", true)
ADD_PHASE("analyzerc", true)
ADD_PHASE("rclowering", true)
ADD_PHASE("emit", true)
// mephase end
ADD_PHASE("GenNativeStubFunc", true)
ADD_PHASE("clinit", true)
ADD_PHASE("VtableImpl", true)
ADD_PHASE("javaehlower", true)
ADD_PHASE("MUIDReplacement", true)
```

From:
https://gitee.com/harmonyos/OpenArkCompiler/blob/master/src/maple_driver/defs/phases.def

ModulePhase类的phase

父类	子类	源码位置	phase名称
ModulePhase	DoCheckCastGeneration	src/mpl2mpl/include/gen_check_cast.h	gencheckcast
	DoClassInit	src/mpl2mpl/include/class_init.h	clinit
	DoGenericNativeStubFunc	src/mpl2mpl/include/native_stub_func.h	GenNativeStubFunc
	DoJavaIntrnLowering	src/mpl2mpl/include/java_intrn_lowering.h	javaintrnlowering
	DoKlassHierarchy	src/maple_ipa/include/module_phase_manager.h	classhierarchy
	DoMUIDReplacement	src/mpl2mpl/include/muid_replacement.h	MUIDReplacement
	DoReflectionAnalysis	src/mpl2mpl/include/reflection_analysis.h	reflectionanalysis
	DoVtableAnalysis	src/mpl2mpl/include/vtable_analysis.h	vtableanalysis
	DoVtableImpl	src/mpl2mpl/include/vtable_impl.h	VtableImpl
	JavaEHLowererPhase	src/maple_ir/include/java_eh_lower.h	javaehlower

MeFuncPhase类的phase

父类	子类	源码位置	phase名称
MeFuncPhase	MeDoAliasClass	src/maple_me/include/me_alias_class.h	aliasclass
	MeDoBBLayout	src/maple_me/include/me_bb_layout.h	bblayout
	MeDoDominance	src/maple_me/include/me_dominance.h	dominance
	MeDoEmission	src/maple_me/include/me_emit.h	emit
	MeDoIRMap	src/maple_me/include/me_irmap.h	irmap
	MeDoRCLowering	src/maple_me/include/me_rc_lowering.h	rclowering
	MeDoSSA	src/maple_me/include/me_ssa.h	ssa
	MeDoSSATab	src/maple_me/include/me_ssa_tab.h	ssaTab

Toy Runtime简介

Toy Runtime简介

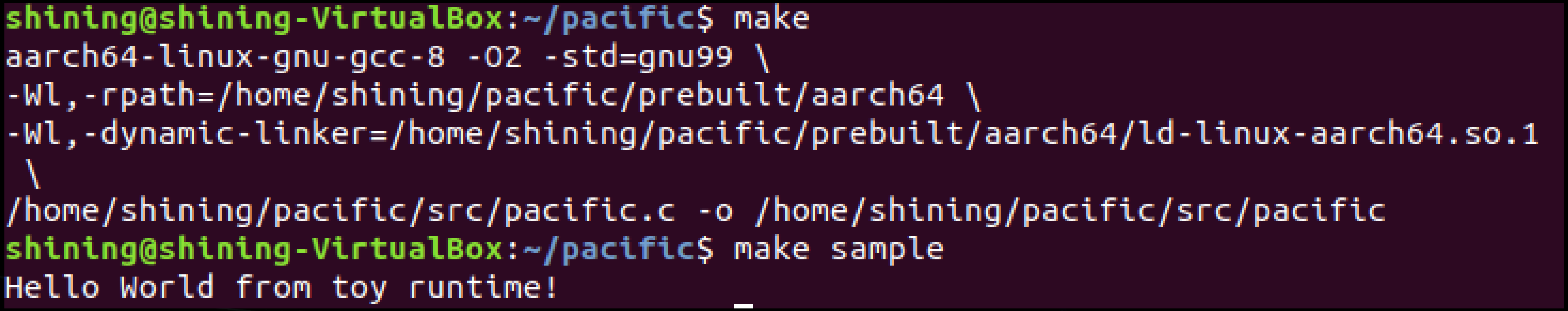
Toy Runtime是中科院软件所智能软件中心程序语言与编译技术实验室在开发的一个方舟编译器Runtime参考实现，这个项目是为了实现一个示例Runtime版本。

Toy Runtime开源地址：https://github.com/isrc-cas/pacific

Toy Runtime简介

目前Toy Runtime已经发布了V0.1版本。

在没有方舟运行时环境设计细节的前提下，我们进行了一定程度的hack和逆向。 采用QEMU来提供AArch64的架构支持，把方舟的Java的那一套巧妙地（硬生生）用GNU/Linux的方式「fake」 了一套可以跑「Hello World」的 Toy Runtime 。



```
shining@shining-VirtualBox:~/pacific$ make
aarch64-linux-gnu-gcc-8 -O2 -std=gnu99 \
-Wl,-rpath=/home/shining/pacific/prebuilt/aarch64 \
-Wl,-dynamic-linker=/home/shining/pacific/prebuilt/aarch64/ld-linux-aarch64.so.1
 \
/home/shining/pacific/src/pacific.c -o /home/shining/pacific/src/pacific
shining@shining-VirtualBox:~/pacific$ make sample
Hello World from toy runtime!
```

软件绿色联盟
Software Green Alliance

构生态·建未来

2019软件绿色联盟开发者大会