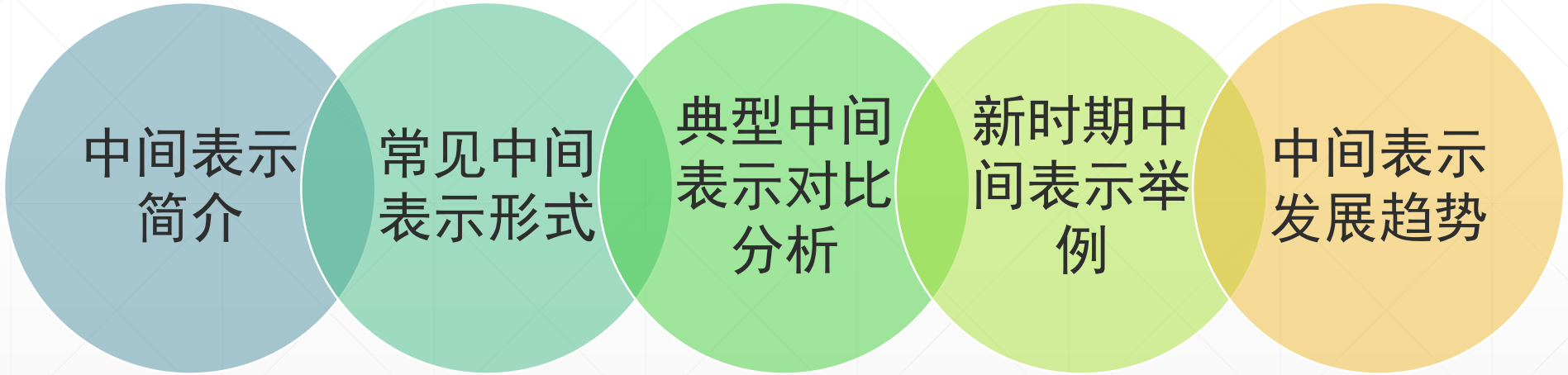
编译器中间表示的形式、特点及发展趋势

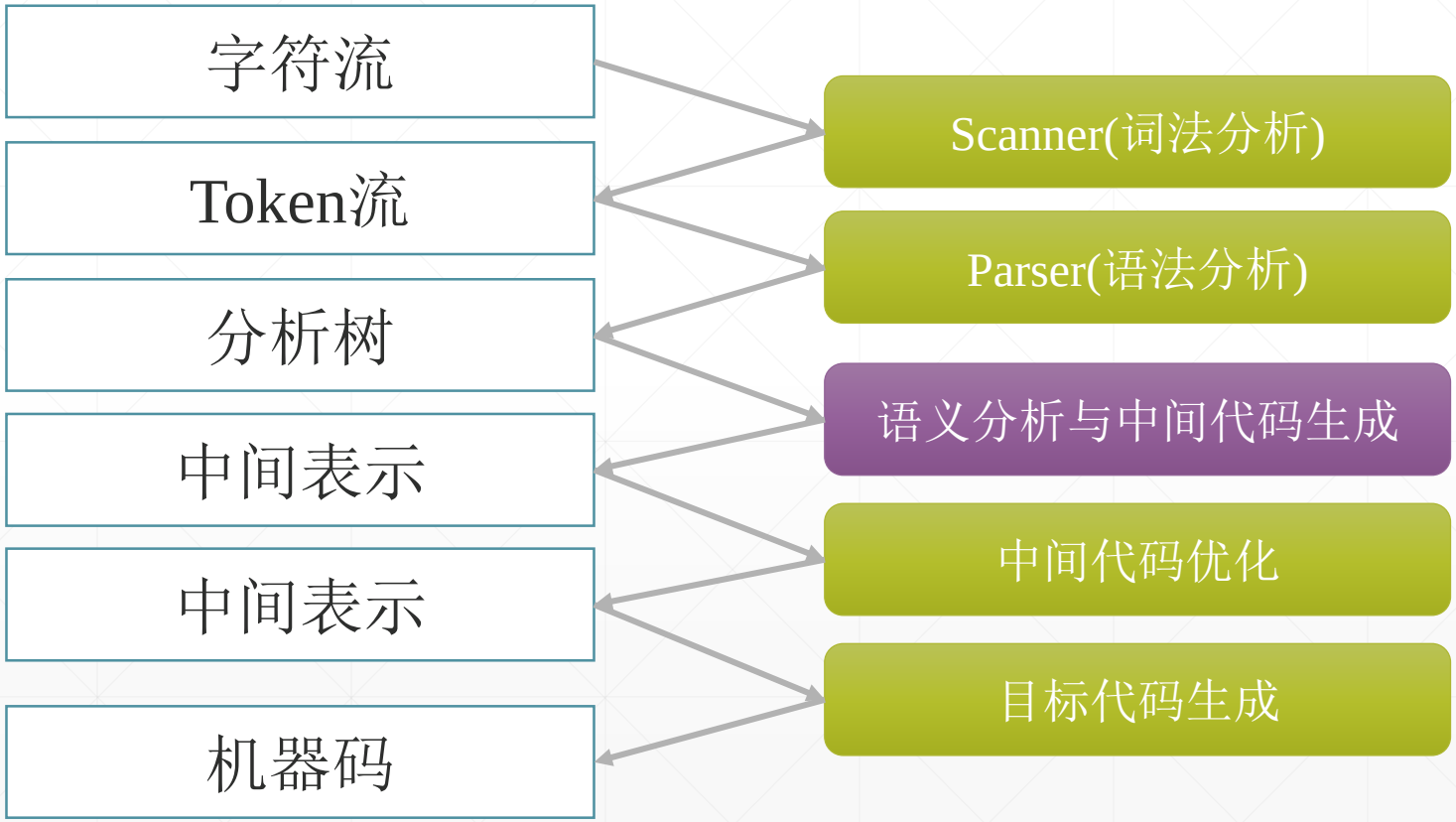
计卫星

北京理工大学 计算机学院

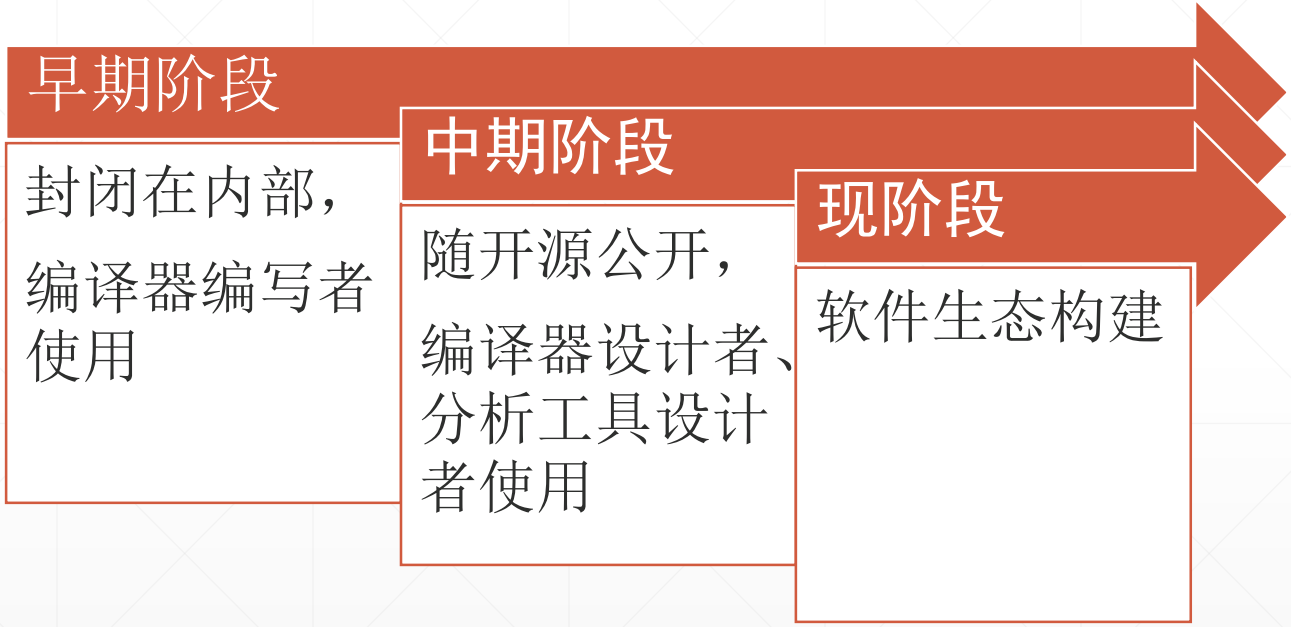
内 容 提 纲



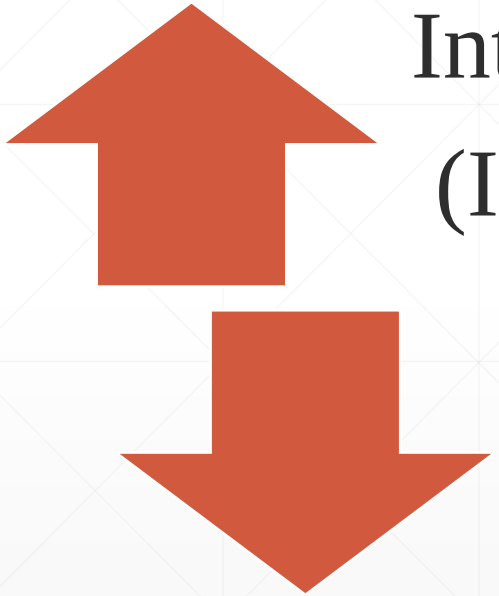
中间表示简介


字符流
Token流
分析树
中间表示
中间表示
机器码
Scanner(词法分析)
Parser(语法分析)
语义分析与中间代码生成
中间代码优化
目标代码生成

中间表示简介


早期阶段
封闭在内部，
编译器编写者使用
中期阶段
随开源公开，
编译器设计者、分析工具设计者使用
现阶段
软件生态构建

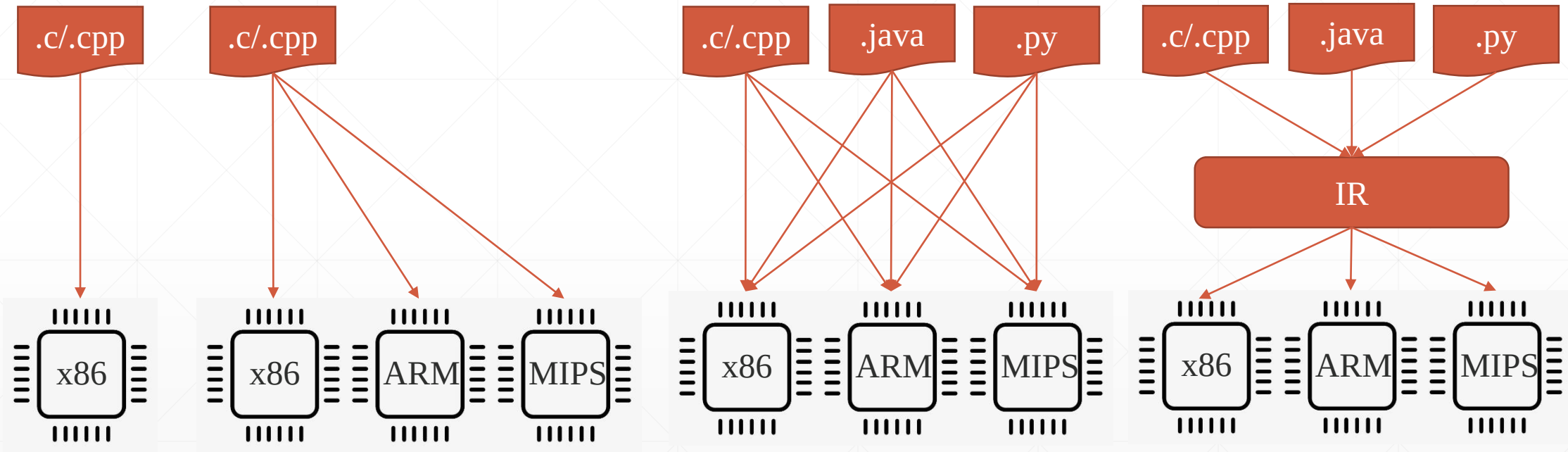
中间表示简介



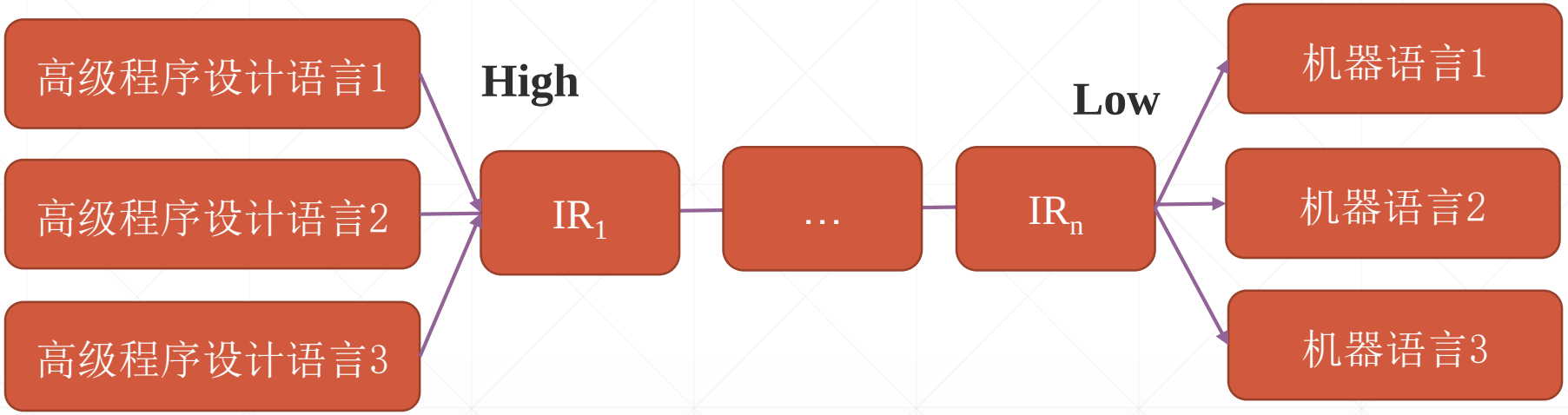
Intermediate Representation (IR)

Intermediate Language (IL)

中间表示简介


.c/.cpp
.c/.cpp
.c/.cpp
.java
.py
.c/.cpp
.java
.py
IR
x86
x86
ARM
MIPS
x86
ARM
MIPS
x86
ARM
MIPS

中间表示简介

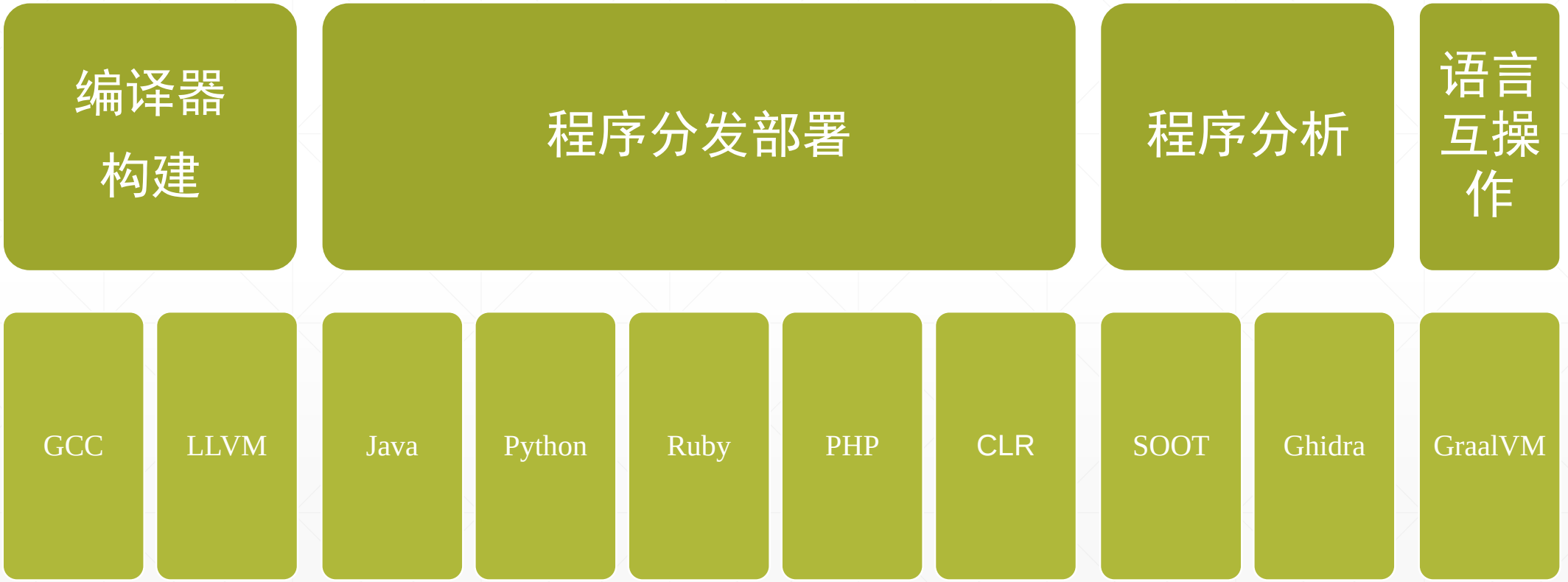


- 多种层次化控制结构
- 比较短的代码序列
- 完整的程序信息

- 较少的控制结构
- 较长的代码序列
- 较少的程序信息
- 层次化嵌套/Flat

- 多种机器指令
- 很长的代码序列
- 非常少的程序信息
- Flat结构

中间表示简介：应用


编译器构建
程序分发部署
程序分析
语言互操作
GCC
LLVM
Java
Python
Ruby
PHP
CLR
SOOT
Ghidra
GraalVM

中间表示简介：存在形式

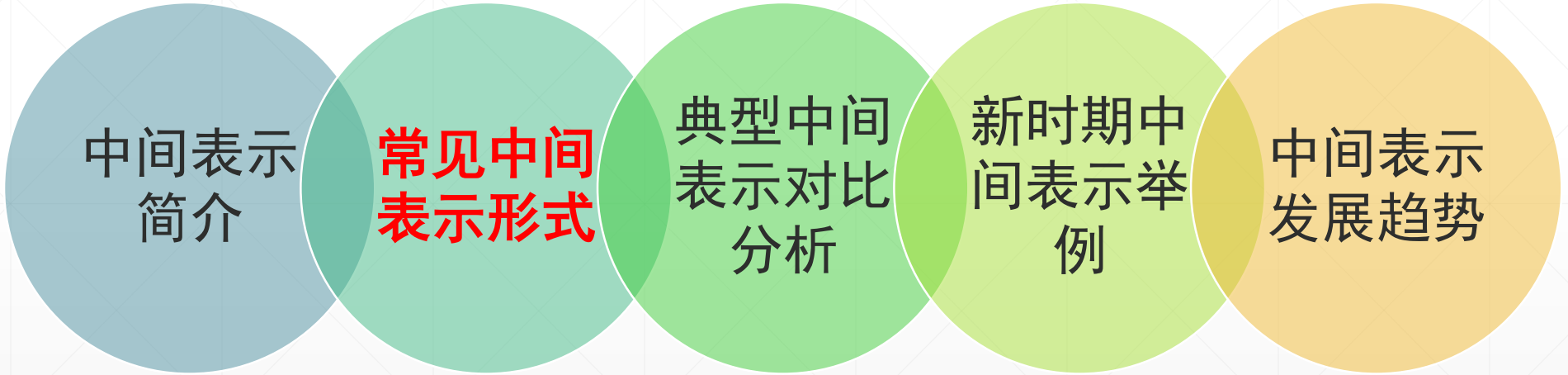
内存表示

文本表示

```
IL_0000:  ldc.i4.0    //
IL_0001:  stloc.0     // sum = 0
IL_0002:  ldarg.0     // load a on the stack
IL_0003:  stloc.1     // store a in first var
(i=a)
IL_0004:  br IL_0011  // --+
IL_0009:  ldloc.0     //   |   <--+
IL_000a:  ldloc.1     //   |      |
IL_000b:  add         //   |      |
IL_000c:  stloc.0     //   |      |
IL_000d:  ldloc.1     //   |      |
IL_000e:  ldc.i4.1    //   |      |
IL_000f:  add         //   |      .
IL_0010:  stloc.1     //   |      .
IL_0011:  ldloc.1     // <-+      .
IL_0012:  ldarg.1     // load b   |
IL_0013:  ble IL_0009 // i<=b   -+
IL_0018:  ldloc.0
IL_0019:  ret
```

二进制表示

内 容 提 纲



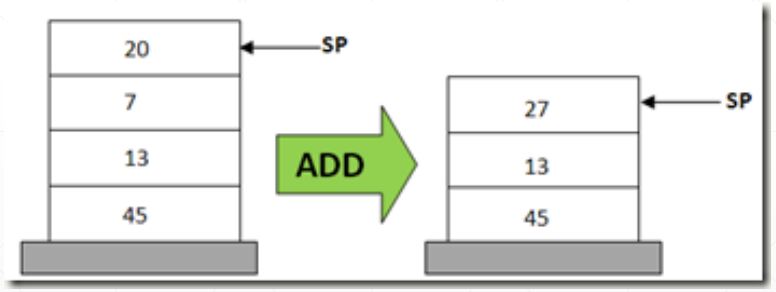
常见中间表示形式

- **机器模型**: 基于栈的和基于寄存器的
- **表示结构：** Hierarchical/Graphical和Flat/Linear
- **相关性**: 无、 控制、 数据、混合
- …

常见中间表示形式

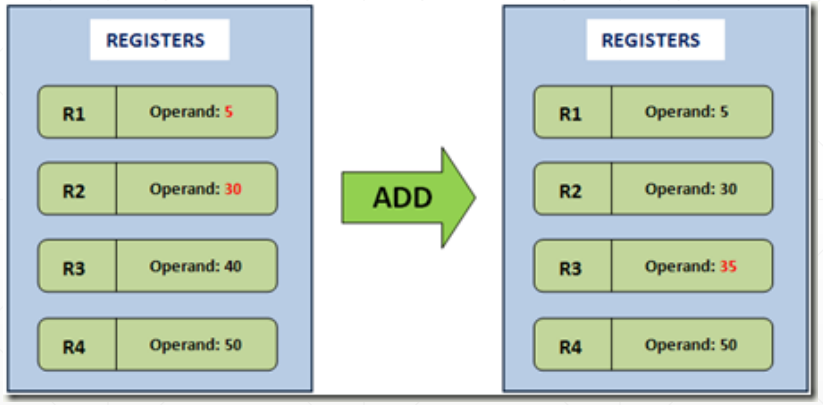
- **机器模型：基于栈的和基于寄存器的**

Java VM、.Net CLR、Python VM



ADD

LLVM IR、Dalvik VM、Lua VM



ADD R1, R2, R3 # R3=R1+R2

常见中间表示形式

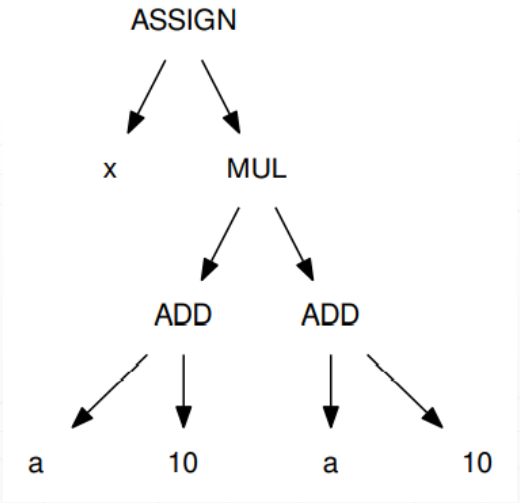
- 机器模型：基于栈的和基于寄存器的

~43%	~47%	~25%	~30%
静态指令数	动态指令数	代码大小	运行效率

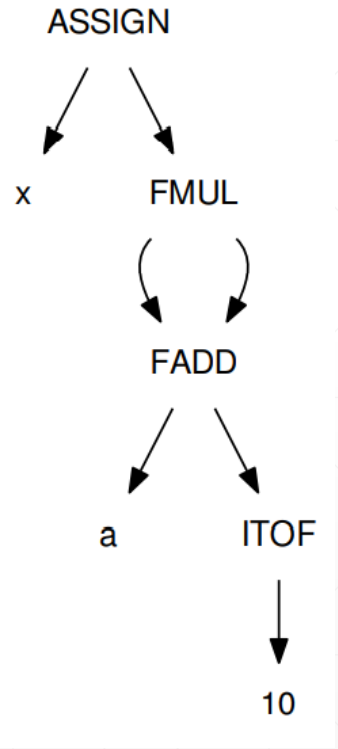
https://www.usenix.org/legacy/events/vee05/full_papers/p153-yunhe.pdf

常见中间表示形式

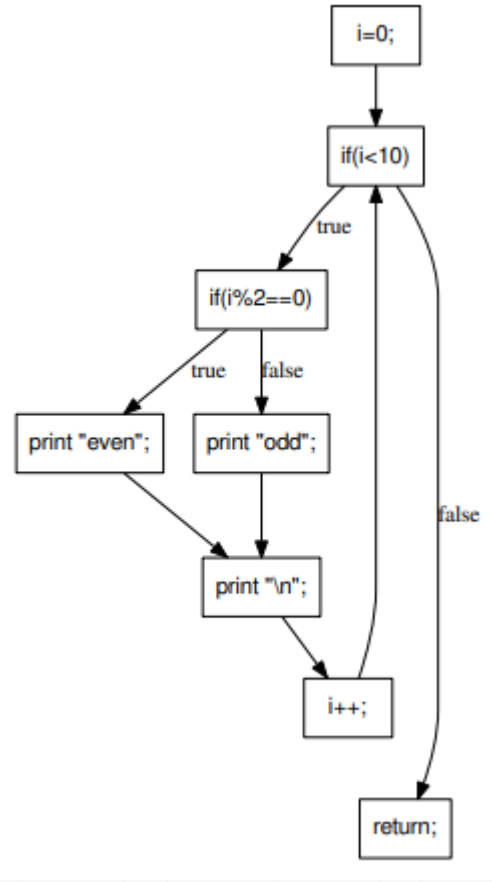
- 表示结构：hierarchical



AST



DAG



CFG

常见中间表示形式

- **表示结构：flat**

```
x = 1
a = x
b = a + 10
x = 20 * b
x = x + 30
```

TAC/3AC

```
x_1 = 1
a_1 = x_1
b_1 = a_1 + 10
x_2 = 20 * b_1
x_3 = x_2 + 30
```

SSA

```
PUSH a
PUSH 10
ITOF
FADD
COPY
FMUL
POP x
```

基于栈的IR

常见中间表示形式

- **表示结构：SSA**
  - 更容易推断变量的生命周期以及某个点活跃的变量
  - 更容易实现指令排序相关的优化

```
if(y < 10)
{
    x = a;
} else {
    x = b;
}
```

```
if(y_1 < 10) {
    x_2 = a;
} else {
    x_3 = b;
}
x_4 = phi(x_2, x_3)
```

内 容 提 纲

中间表示简介

常见中间表示形式

典型中间表示对比分析

新时期中间表示举例

中间表示发展趋势

典型中间表示对比分析

- IR指令分类

典型中间表示对比分析

- GCC IR

C

```
int main() {
    int a = 0;
    int b = 1;
    int c = a + b;
    return c;
}
```

GENERIC

```
{
  int a = 0;
  int b = 1;
  int c = a + b;

    int a = 0;
    int b = 1;
    int c = a + b;
  return c;
}
```

GIMPLE

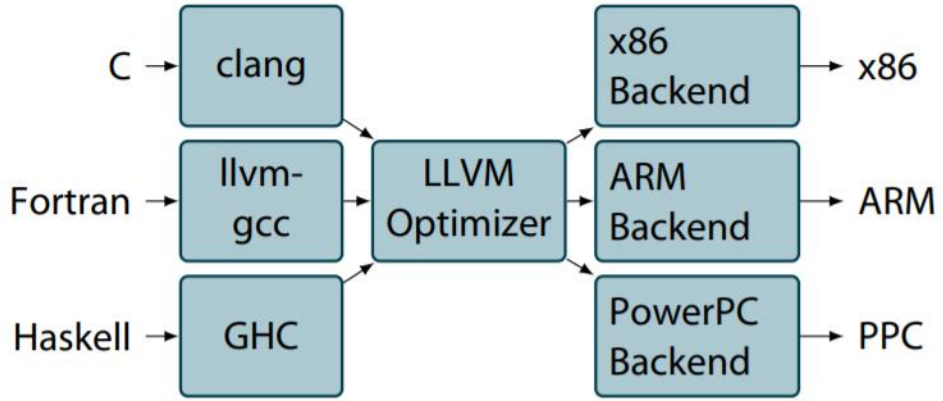
```
main ()
gimple_bind <
  int D.1727;
  int a;
  int b;
  int c;

  gimple_assign <integer_cst, a, 0,
NULL, NULL>
  gimple_assign <integer_cst, b, 1,
NULL, NULL>
  gimple_assign <plus_expr, c, a, b,
NULL>
  gimple_assign <var_decl, D.1727,
c, NULL, NULL>
  gimple_return <D.1727>
>
```

RTL

```
(note# 0 0 NOTE_INSN_DELETED)
(note# 0 0 [bb 2]
NOTE_INSN_BASIC_BLOCK)
(insn/f# 0 0 2 (set (mem:DI (pre_dec:DI
(reg/f:DI 7 sp)) [  S8 A8])
        (reg/f:DI 6 bp)) test.c:1#
{*pushdi2_rex64}
     (nil))
(insn/f# 0 0 2 (set (reg/f:DI 6 bp)
        (reg/f:DI 7 sp)) test.c:1#
{*movdi_internal_rex64}
     (nil))
(insn# 0 0 2 (set (mem/v:BLK (scratch:DI)
[  A8])
```

典型中间表示对比分析

- LLVM IR




```
float f( int a, int b, float x ) {
    float y = a*x*x + b*x + 100;
    return y;
}
```

```
define float @f(i32 %a, i32 %b, float %x) #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca float, align 4
  %y = alloca float, align 4
  store i32 %a, i32* %1, align 4
  store i32 %b, i32* %2, align 4
  store float %x, float* %3, align 4
  %4 = load i32* %1, align 4
  %5 = sitofp i32 %4 to float
  %6 = load float* %3, align 4
  %7 = fmul float %5, %6
  %8 = load float* %3, align 4
  %9 = fmul float %7, %8
  %10 = load i32* %2, align 4
  %11 = sitofp i32 %10 to float
  %12 = load float* %3, align 4
  %13 = fmul float %11, %12
  %14 = fadd float %9, %13
  %15 = fadd float %14, 1.000000e+02
  store float %15, float* %y, align 4
  %16 = load float* %y, align 4
  ret float %16
}
```

典型中间表示对比分析

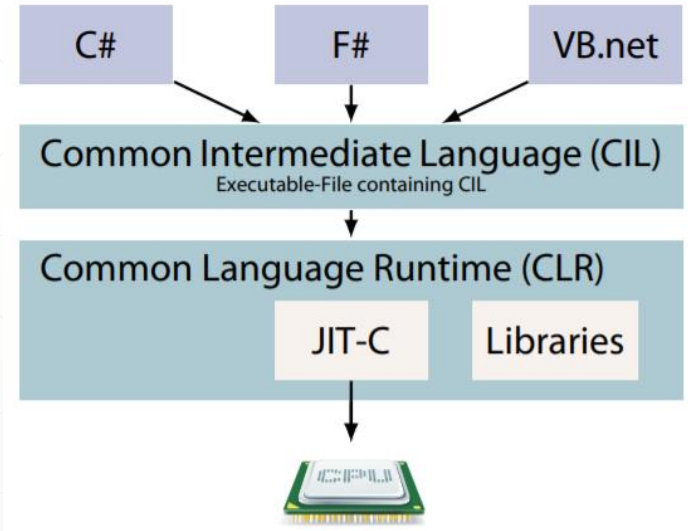
- JVM IR

```
int fact(int n){
  if( n!=1 )
      return n * fact(n -1);
  else
      return 1;
```

```
int fact(int);
   Code:
      0: iload_1
      1: iconst_1
      2: if_icmpeq     15
      5: iload_1
      6: aload_0
      7: iload_1
      8: iconst_1
      9: isub
     10: invokevirtual #5 // Method fact:(I)I
     13: imul
     14: ireturn
     15: iconst_1
     16: ireturn
```

典型中间表示对比分析

- Microsoft CIL(Common Intermediate Language)




```
.method public static hidebysig
default int32 sum (int32 a, int32 b)  cil
managed {
.maxstack 2
.locals init (int32 V_0, int32  V_1)
IL_0000:  ldc.i4.0    //
...
}
```

```
IL_0000:  ldc.i4.0    //
IL_0001:  stloc.0     // sum = 0
IL_0002:  ldarg.0     // load a on the stack
IL_0003:  stloc.1     // store a in first var
(i=a)
IL_0004:  br IL_0011  // --+
IL_0009:  ldloc.0     //   |    <--+
IL_000a:  ldloc.1     //   |       |
IL_000b:  add         //   |
IL_000c:  stloc.0     //   |
IL_000d:  ldloc.1     //   |
IL_000e:  ldc.i4.1    //   |
IL_000f:  add         //   |       .
IL_0010:  stloc.1     //   |       .
IL_0011:  ldloc.1     // <-+       .
IL_0012:  ldarg.1     // load b    |
IL_0013:  ble IL_0009 // i<=b    -+
IL_0018:  ldloc.0
IL_0019:  ret
```

典型中间表示对比分析

- SOOT IR

Java/Bytecode

```
public class Hello
    public int A(){
        int a = 1;
        int b = 2;
        return a + b;
    }
}
public class Hello {
    public int A() {
        byte var1 = 1;
        byte var2 = 2;
        return var1 + var2;
    }
}
```

Baf

```
public int A()
    {
        word r0;
        r0 := @this: Hello;
        push 1;
        push 2;
        add.i;
        return.i;
    }
```

Jimple

```
public int A()
    {
        int $i2;

        Hello r0;
        r0 := @this: Hello;
        $i2 = 1 + 2;
        return $i2;
    }
```

Shimple

```
public int A()
    {
        byte b0, b1;
        int $i2;
        Hello r0;
        r0 := @this: Hello;
        b0 = 1;
        b1 = 2;
        $i2 = b0 + b1;
        return $i2;
    }
```

Grimp

```
public int A()
    {
        Hello r0;

        r0 := @this;
        return 1 + 2;
    }
```

典型中间表示对比分析

- SOOT Jimple

算术运算类

- AddExpr
- SubExpr
- DivExpr

关系运算类

- JNeExpr
- EqExpr
- GtExpr

位运算类

- JShlExpr
- JUShrExpr
- JShrExpr

面向对象相关

- InstanceOfExpr
- NewExpr
- CastExpr

逻辑运算类

- JAndExpr
- JNegExpr
- XnorExpr

函数相关

- JStaticInvokeExpr
- JSpecialInvokeExpr
- JReturn

其它语句

- JIdentityStmt
- JInvokeStmt
- JLookupSwitchStmt

分支跳转类

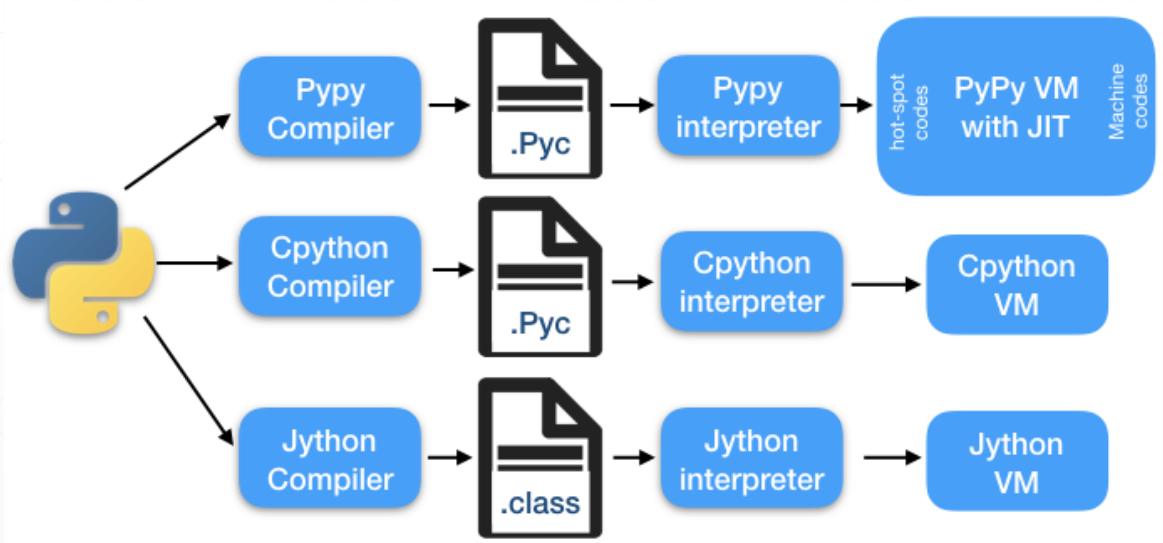
- JGotoStmt
- JIfStmt

其它类

- JLengthExpr
- JNewArrayExpr
- JNewMultiArrayExpr

典型中间表示对比分析

- Python VM IR



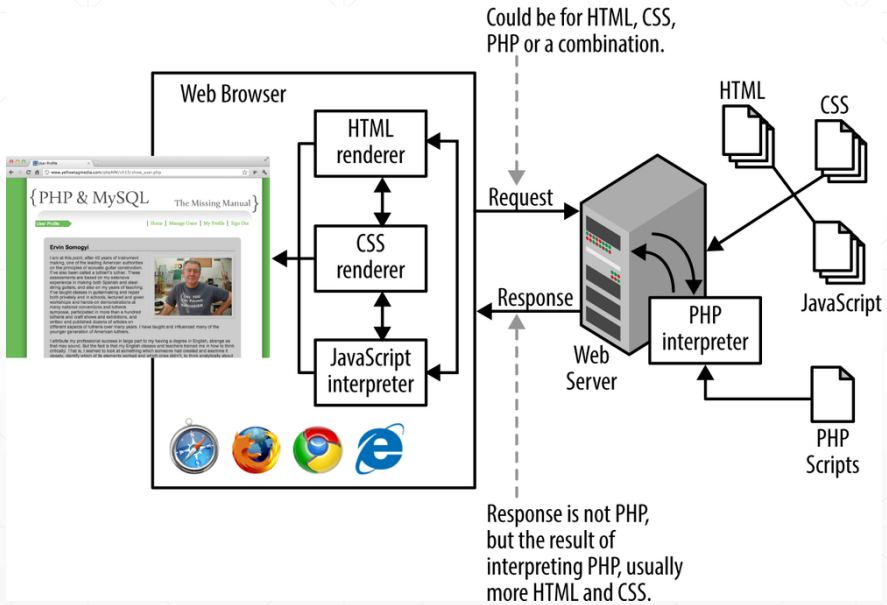

```
def add(val0,val1):
...     val2 = val0 + val1
...     return val2
```

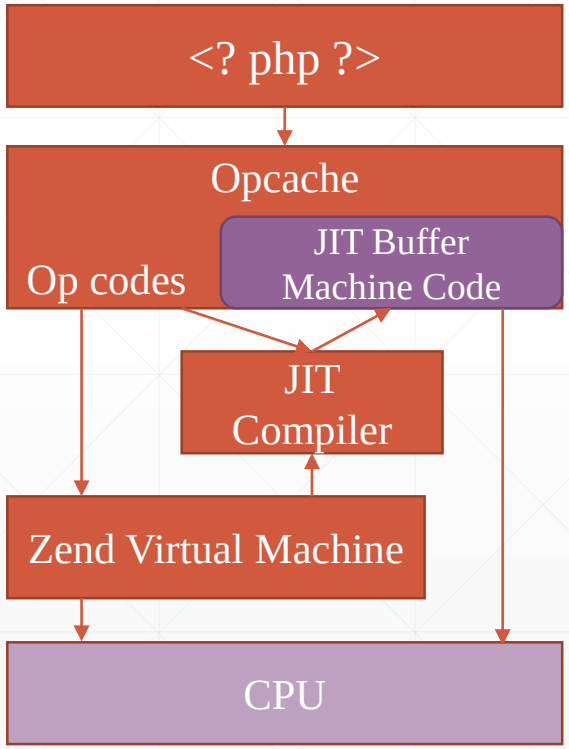
```
2           0 LOAD_FAST                0 (val0)
            2 LOAD_FAST                1 (val1)
            4 BINARY_ADD
            6 STORE_FAST               2 (val2)

3           8 LOAD_FAST                2 (val2)
           10 RETURN_VALUE
```

https://towardsdatascience.com/why-numba-sometime-way-slower-than-numpy-15d077390287

典型中间表示对比分析

- PHP Op Codes






```php
<?php
    $name = "jdj";
    echo "name is ".$name;
?>
```

line	op	return	operands	
2	ASSIGN		!0	jdj'
3	CONCAT	~2	name+is+'	!0
	ECHO		~2	
5	RETURN		1	

https://www.oreilly.com/library/view/php-mysql/9781449355517/ch01.html

典型中间表示对比分析

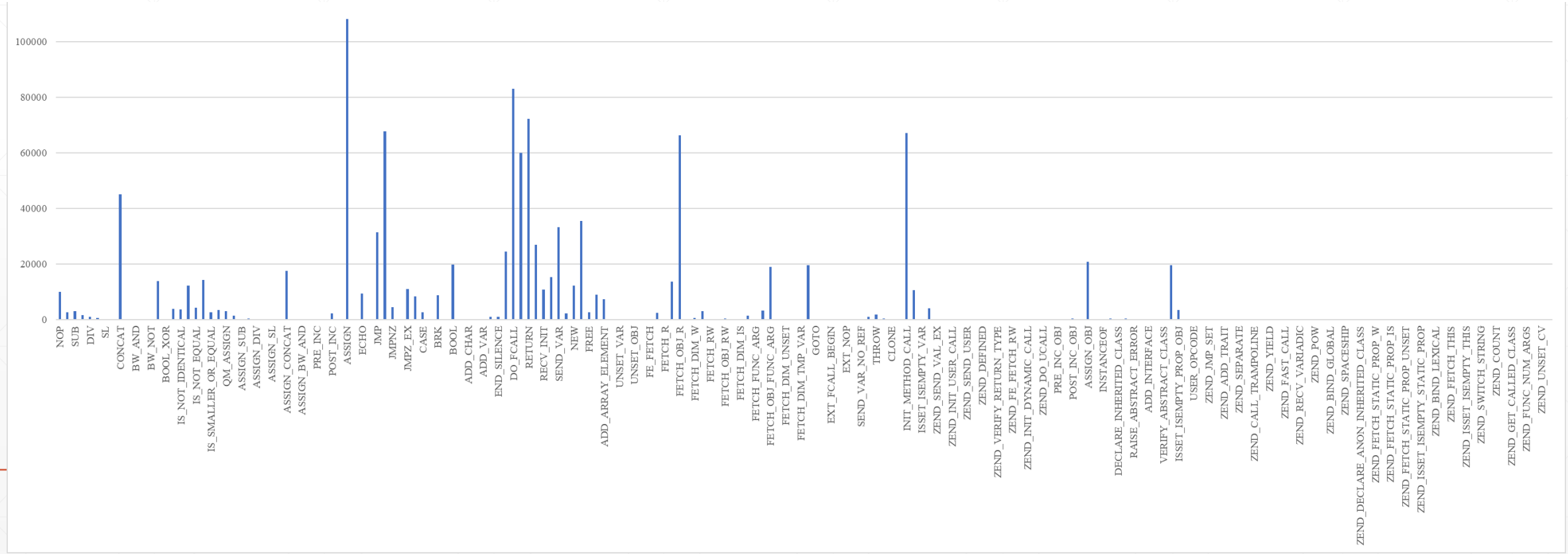
- PHP Op Codes

算术运算类	逻辑运算类	关系运算类	位运算类	分支跳转类
ADD	BOOL_XOR	IS_IDENTICAL	BW_OR	JMP
SL	BOOL_NOT	IS_EQUAL	BW_AND	JMPZ
CONCAT		IS_SMALLER	BW_XOR	JMPNZ

存储访问类	面向对象相关	语句类	调用类	其他类
FETCH_R	NEW	ASSIGN	DO_FCALL	INIT_ARRAY
FETCH_DIM_R	INSTANCEOF	BRK	SEND_VAR	BEGIN_SILIENCE
FETCH_CONSTANT	FETCH_OBJ_R	EXIT	RETURN	ECHO

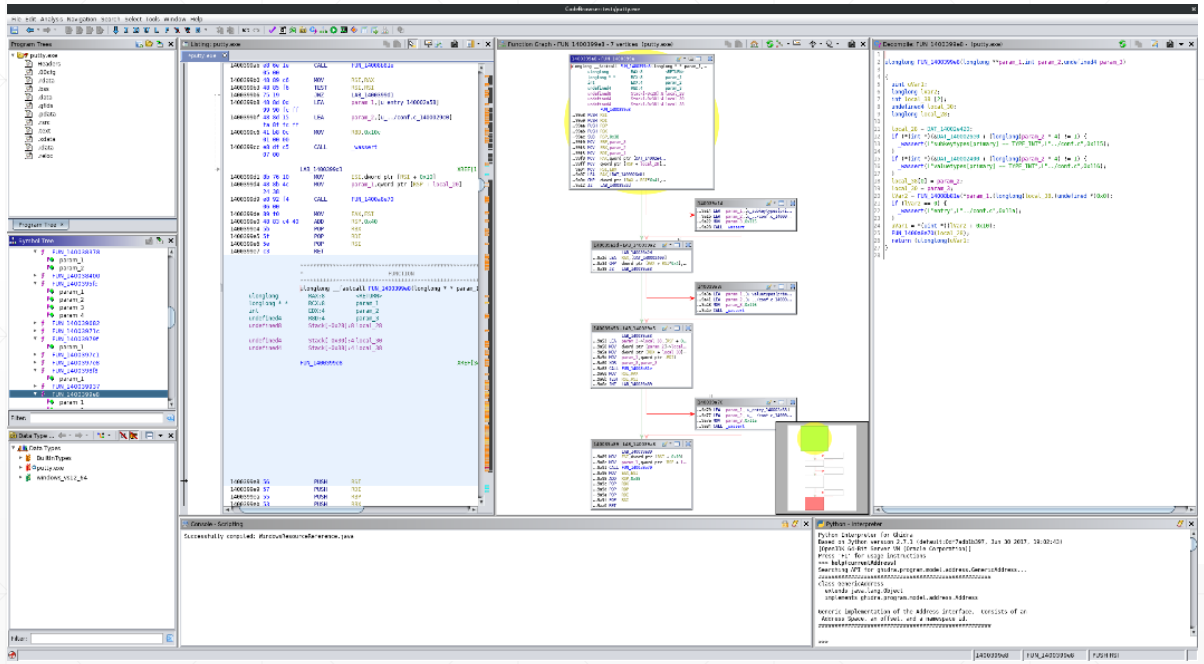
典型中间表示对比分析

- Moodle插件：399个PHP文件
  - ASSIGN_SUB、 ASSIGN_MOD、 GOTO、 ZEND_*、…



典型中间表示对比分析

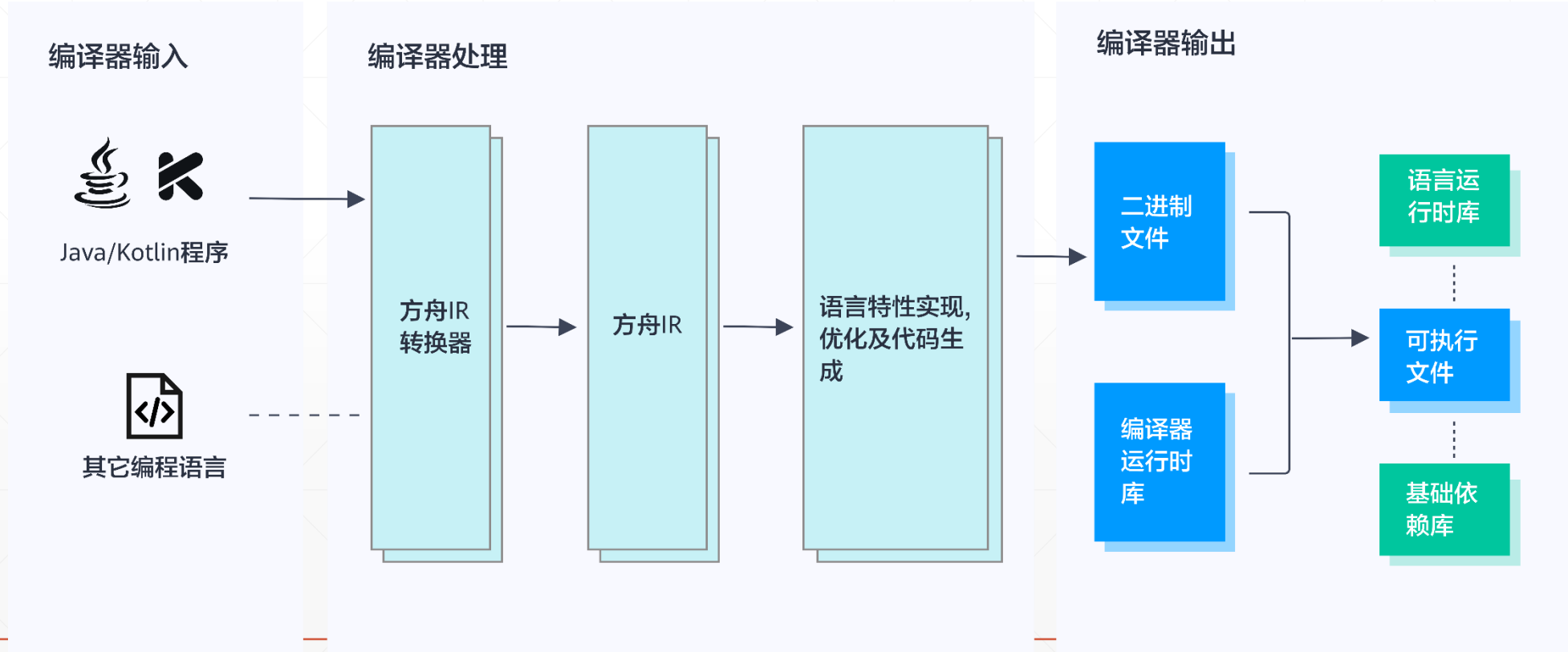
- Ghidra p-code



```
                        LAB_00004342                                    XREF[2]:     00001450(*), 00004346(j)
00004342 04 4f          ld.bu       d15,[a4+]
                                    (unique, 0x1fe0, 4) = COPY (register, 0xff90, 4)
                                    (register, 0xff90, 4) = INT_ADD (register, 0xff90, 4), ...
                                    (unique, 0x6cb0, 1) = LOAD (const, 0x1a1, 8), (unique, ...
                                    (register, 0xff3c, 4) = INT_ZEXT (unique, 0x6cb0, 1)
00004344 24 5f          st.b        [a5+]=>DAT_d0000a80,d15
                                    (unique, 0x1f70, 4) = COPY (register, 0xff94, 4)
                                    (register, 0xff94, 4) = INT_ADD (register, 0xff94, 4), ...
                                    STORE (const, 0x1a1, 8), (unique, 0x1f70, 4), (register...
00004346 fc 7e          loop        a7,LAB_00004342
                                    (unique, 0x73a0, 4) = COPY (register, 0xff9c, 4)
                                    (register, 0xff9c, 4) = INT_SUB (register, 0xff9c, 4), ...
                                    (unique, 0x73c0, 1) = INT_NOTEQUAL (unique, 0x73a0, 4),...
                                    CBRANCH (ram, 0x4342, 4), (unique, 0x73c0, 1)
```

典型中间表示对比分析

- MAPLE IR



典型中间表示对比分析

- MAPLE IR
  - 尽可能保留源码信息
  - 高层次树状层次化结构
  - 低层次flat结构
  - 可扩展：支持新的语言和控制结构
  - 原始类型
    - 无类型:void
    - 有符号整型:i8,i16,i32,i64
    - 无符号整型:u8,u16,u32,u64
    - 布尔类型:u1
    - 地址类型:ptr, ref, a32, a64
    - 浮点数:f32,f64
    - 复数:c64,c128

```
opcode fields (opnd0, opnd1, opnd2)

a=b
dassign $a (dread i32 $b)

a=b+c
dassign $a (
      add i32(dread i32 $b, dread i32 $c))

a=b+c-d
dassign $a (
  sub i32(
    add i32(dread i32 $b, dread i32 $c),
    dread i32 $d))
```

典型中间表示对比分析

- MAPLE IR

```
int foo(int i,int j){
    return (i + j) * -998;
}
```

```
func &foo (var %i i32, var %j i32) i32 {
return (
  mul i32 (
    add i32 (dread i32 %i, dread i32 %j),
    constval i32 -998))}
```

典型中间表示对比分析

- MAPLE IR

a[i]=i

```
iassign<*i32>(
   array a32 <* [10] i32> (addrof a32 $a, dread i32 $i),
   dread i32 $i)
```

（注： <* [10] i32> 表示包含10个int类型元素的数组指针）

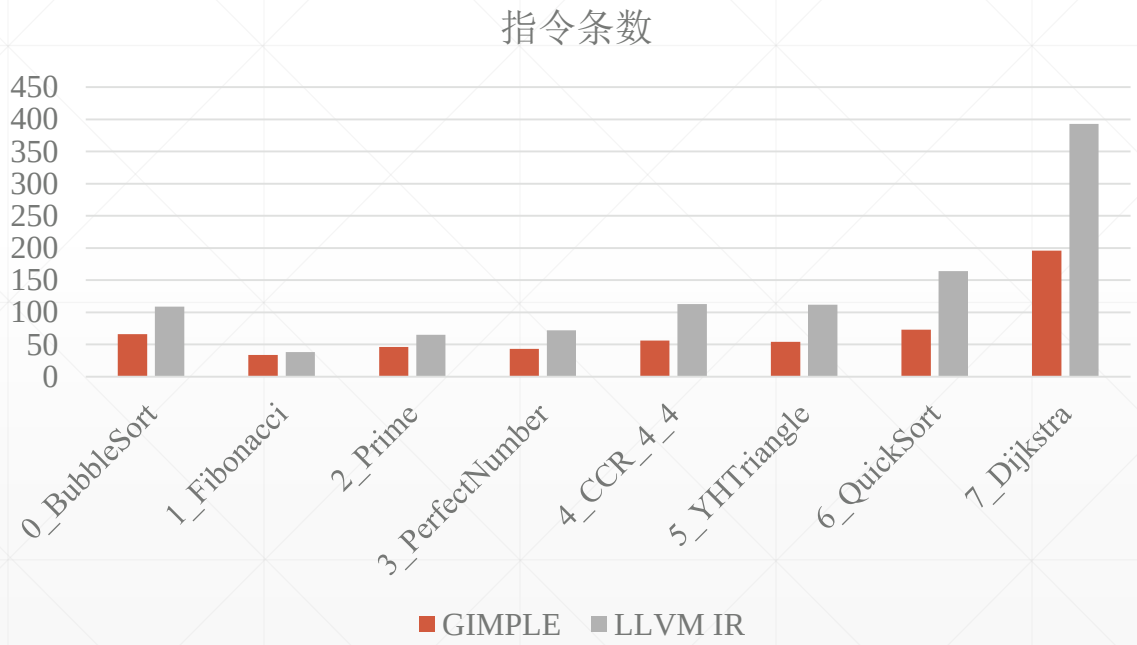
x=a[i, j]

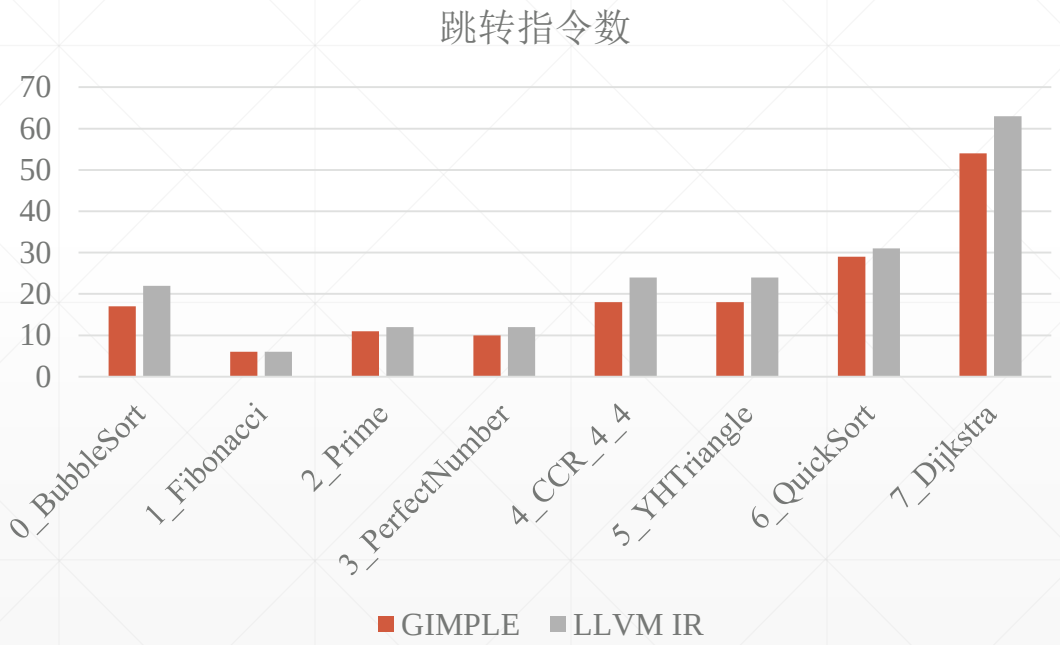
```
dassign $x (
  iread i32 <* i32>(
   array a32 <* [10] [10] i32> (addrof a32 a, dread i32 i,dread i32 $j)))
```

（注： <* [10] [10] i32 表示10*10矩阵的指针）

典型中间表示对比分析

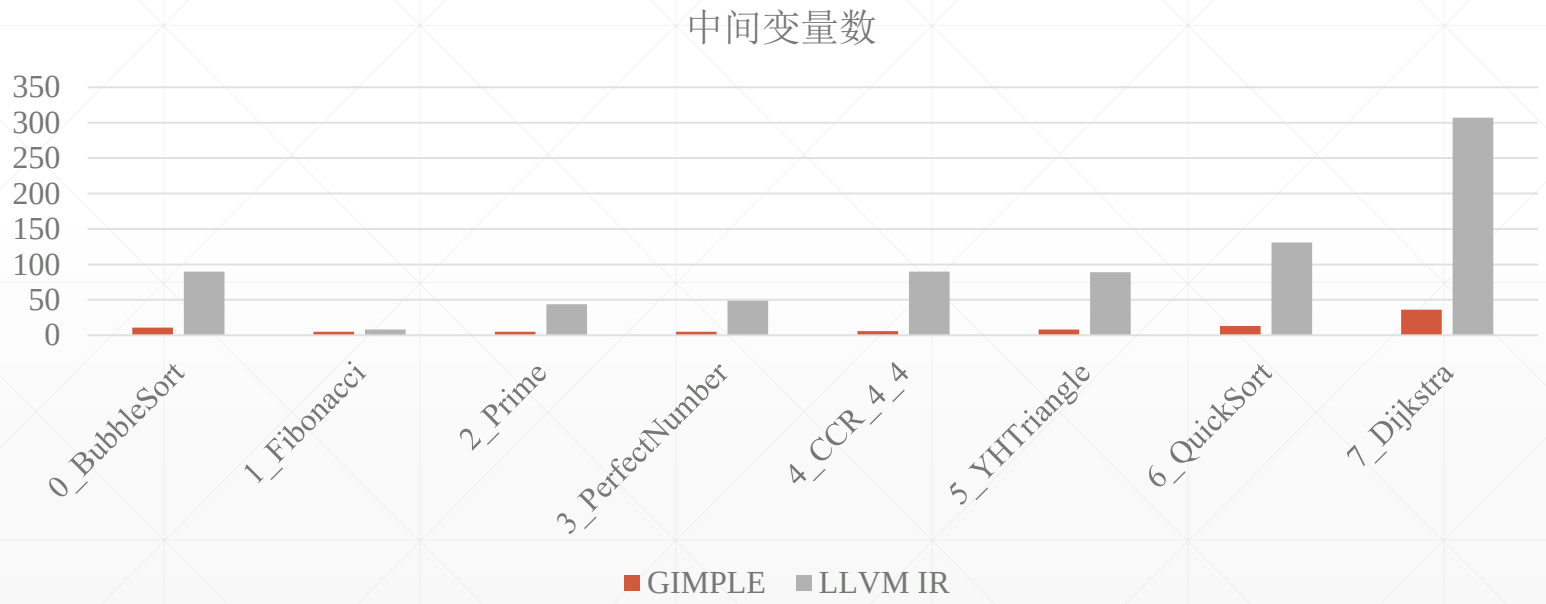
- GCC和LLVM简单对比



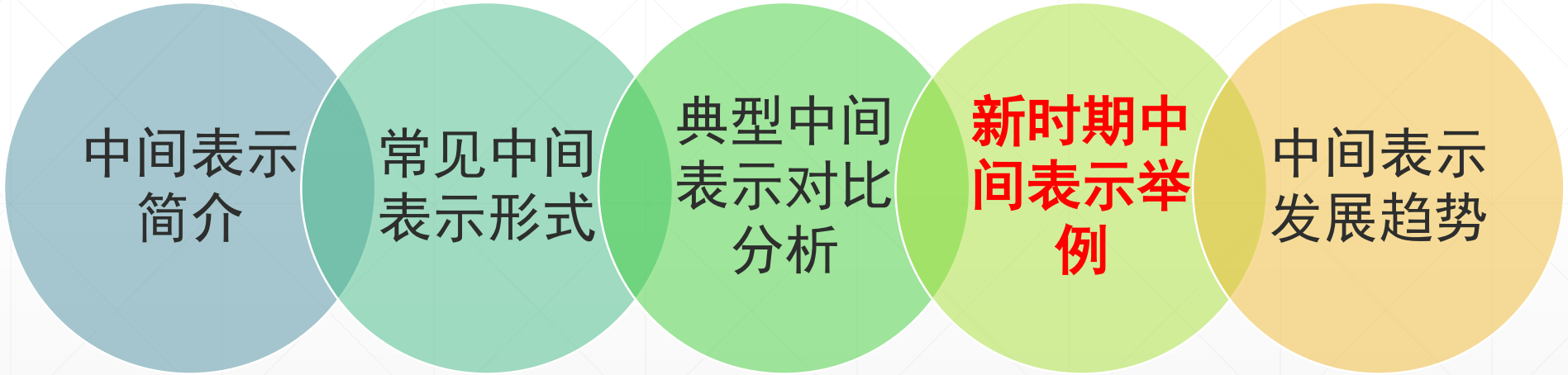


典型中间表示对比分析

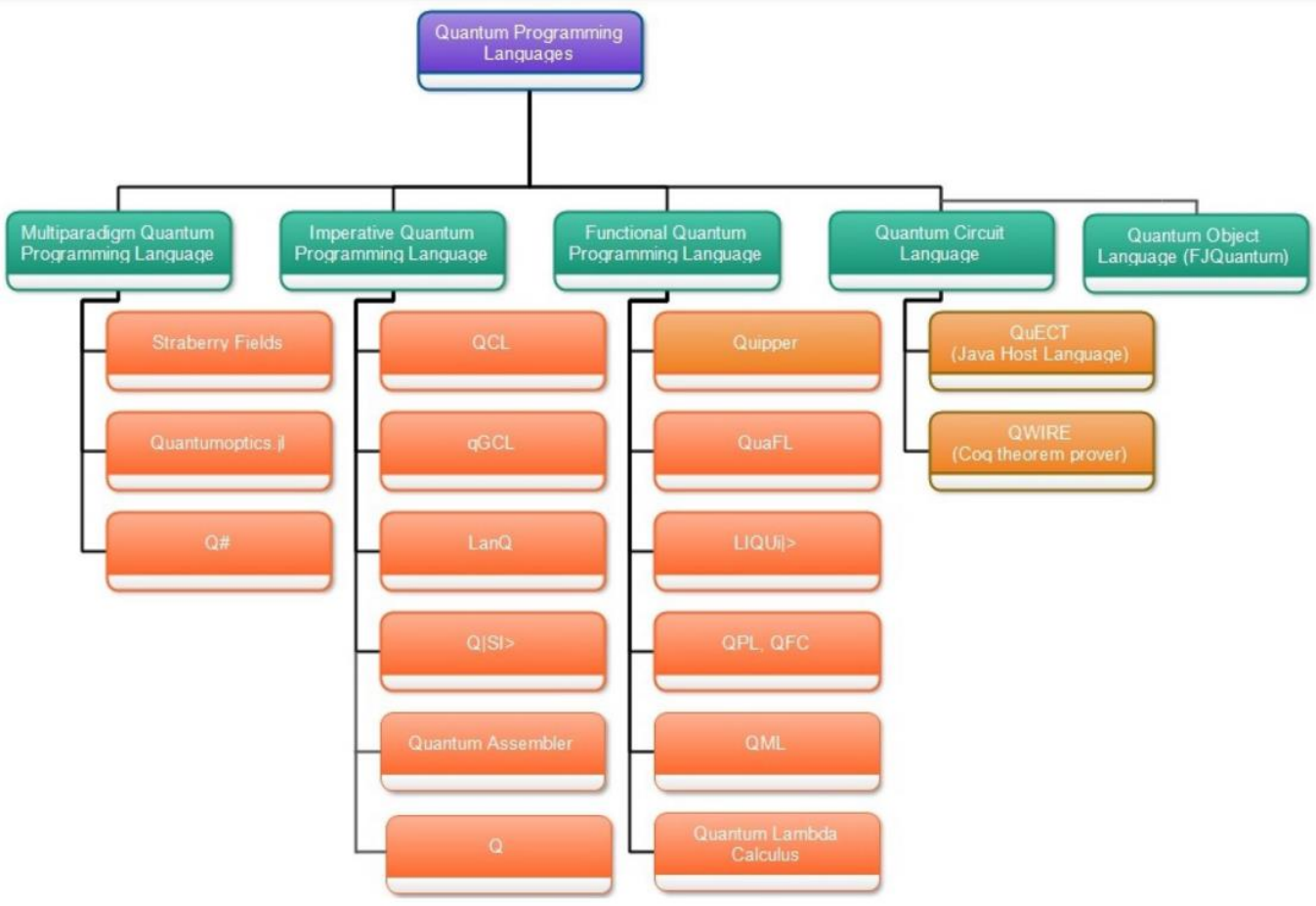
- GCC GIMPLE和LLVM IR简单对比



内 容 提 纲

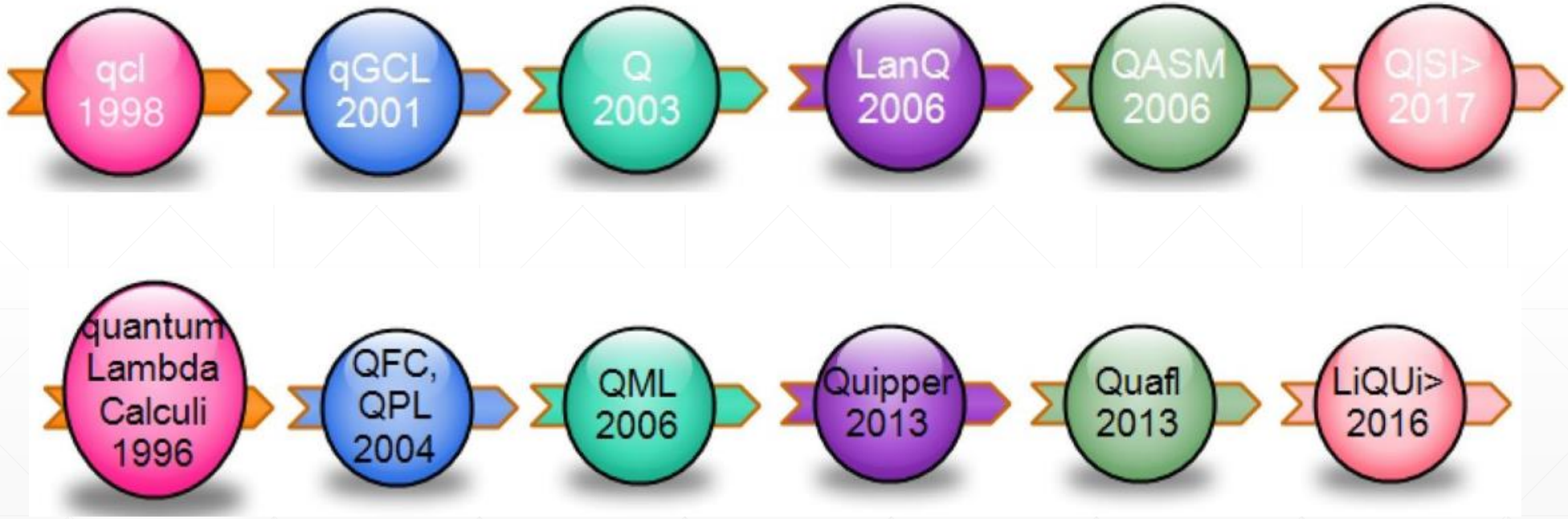


量子编程语言



Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages

量子编程语言

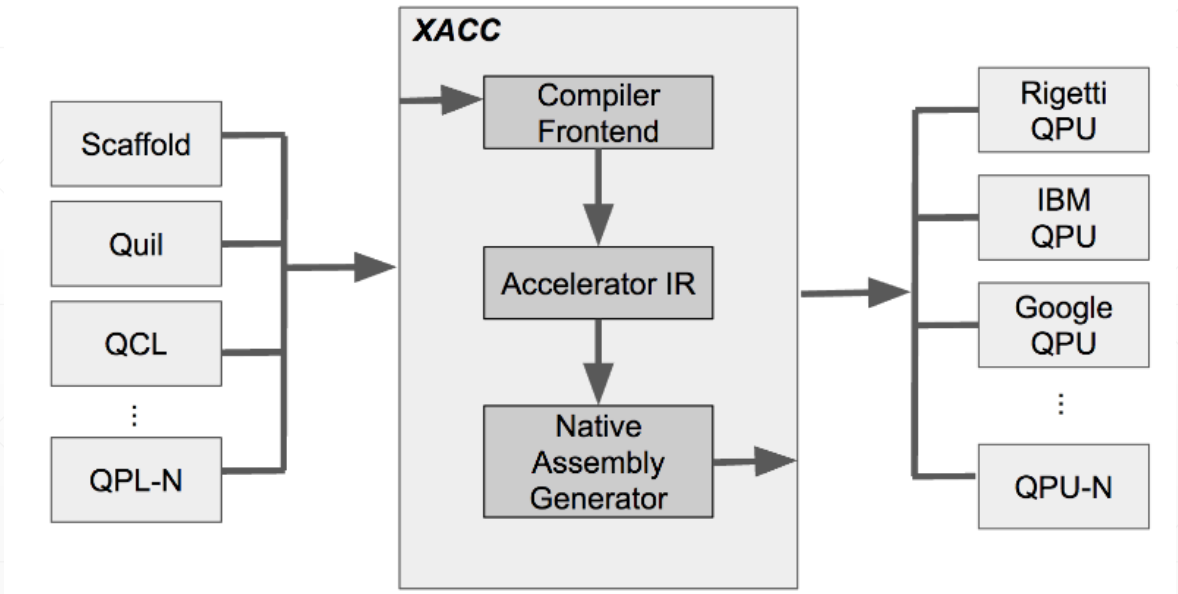

qcl
1998
qGCL
2001
Q
2003
LanQ
2006
QASM
2006
Q|SI>
2017
quantum
Lambda
Calculi
1996
QFC,
QPL
2004
QML
2006
Quipper
2013
Quafl
2013
LiQUi>
2016

量子中间表示：QIR

```
// Assumes that qb1 and qb2 are already in the |0> state
operation BellPair(qb1 : Qubit, qb2 : Qubit) : Unit
{
    H(qb1);
    CNOT(qb1, qb2);
}
```

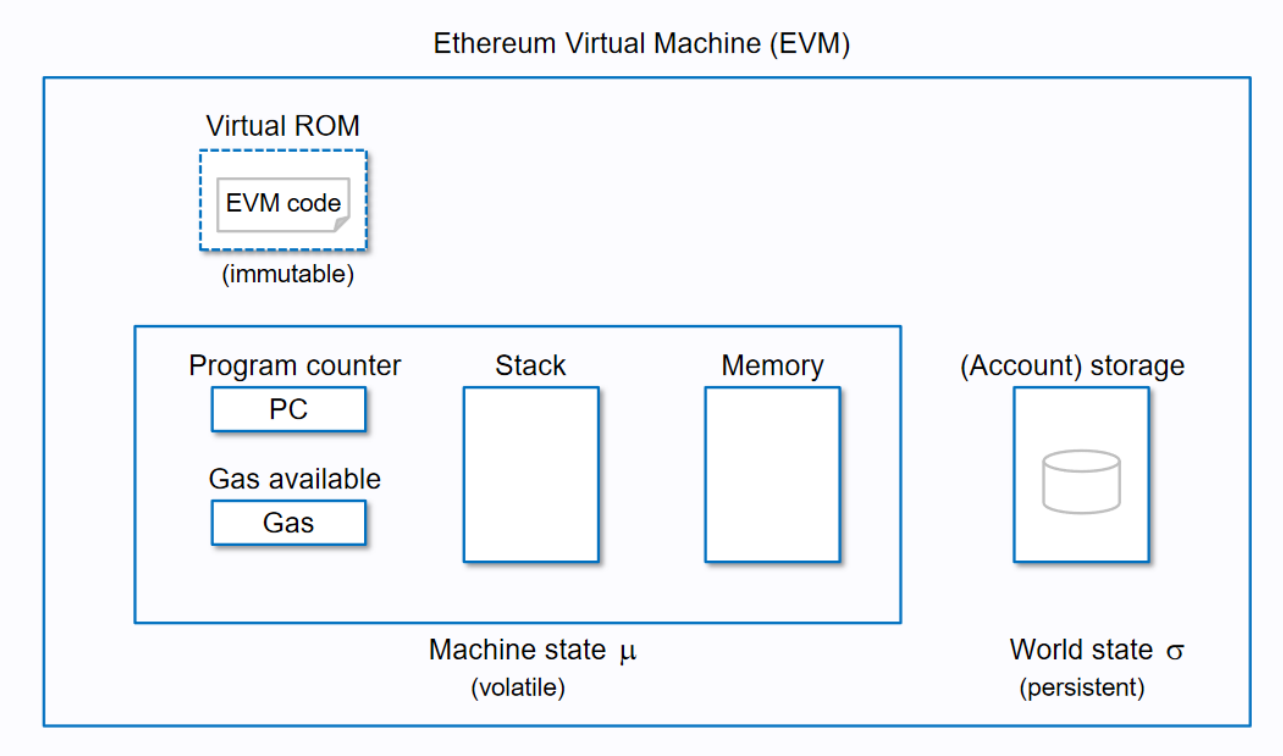
```
define void @BellPair__body(%Qubit* %qb1, %Qubit* %qb2) {
entry:
  call void @__quantum__qis__h(%Qubit* %qb1)
  call void @__quantum__qis__cnot(%Qubit* %qb1, %Qubit* %qb2)
  ret void
}
```

量子中间表示：XACC



- CNOT
- ConditionalFunction
- CPhase
- CZ
- Hadamard
- Identity
- Measure
- Rx
- Ry
- Rz
- Swap
- X
- Y
- Z

以太坊：智能合约




```solidity
pragma solidity ^0.5.0;

contract Coin {
    // The keyword "public" makes those variables
    // easily readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react to
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    constructor() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

https://ethereum.org/ja/developers/docs/evm/

以太坊：智能合约

```solidity
pragma solidity ^0.4.22;
contract Demo{
    uint public value1 = 0;
    uint public value2 = 0;

    function A(uint v) public returns(uint){
        value1 += v;
        return value1;
    }
    function B(uint v) public{
        value2 += A(v);
    }
}
```

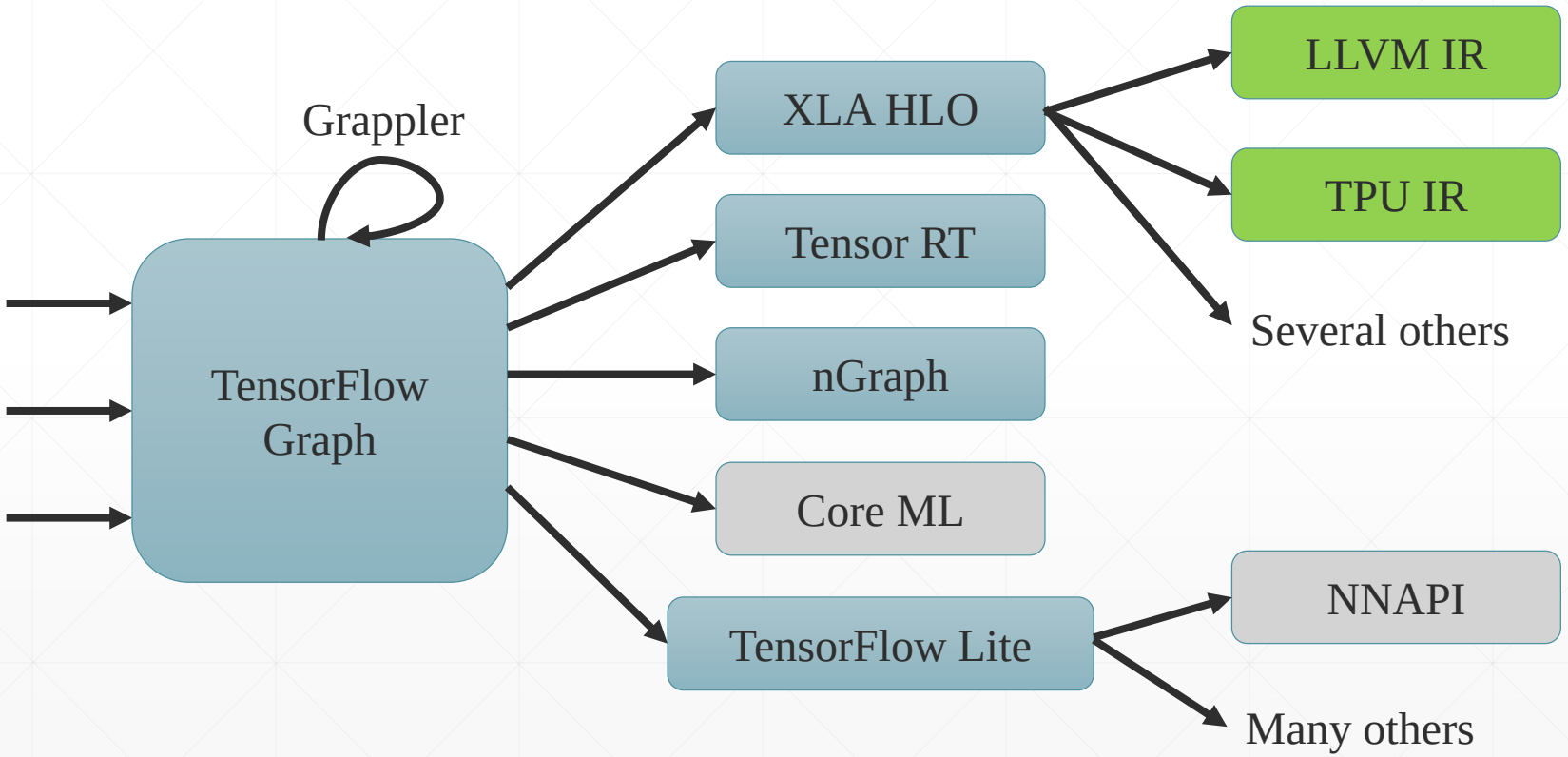
```
.code
  PUSH 80                    contract Demo{\n\tuint public ...
  PUSH 40                    contract Demo{\n\tuint public ...
  MSTORE                     contract Demo{\n\tuint public ...
  PUSH 0                     0
  DUP1                 uint public value1 = 0
  SSTORE                     uint public value1 = 0
  PUSH 0                     0
  PUSH 1                     uint public value2 = 0
  SSTORE                     uint public value2 = 0
  CALLVALUE                  contract Demo{\n\tuint public ...
  DUP1                 olidity ^
  ISZERO                     a
  PUSH [tag] 1               a
  JUMPI                      a
  PUSH 0                     a
  DUP1                 n
  REVERT                     .22;\ncontrac
tag 1                  a
  JUMPDEST                   a
  POP                  contract Demo{\n\tuint public ...
  PUSH #[$] 0000000000000000000000000000000000000000000000000000000000000000
  DUP1                 contract Demo{\n\tuint public ...
  PUSH [$] 0000000000000000000000000000000000000000000000000000000000000000
  PUSH 0                     contract Demo{\n\tuint public ...
  CODECOPY                   contract Demo{\n\tuint public ...
  PUSH 0                     contract Demo{\n\tuint public ...
  RETURN                     contract Demo{\n\tuint public ...
```

以太坊：智能合约

```solidity
pragma solidity ^0.4.22;
contract Demo{
    uint public value1 = 0;
    uint public value2 = 0;

    function A(uint v) public returns(uint){
        value1 += v;
        return value1;
    }
    function B(uint v) public{
        value2 += A(v);
    }
}
```

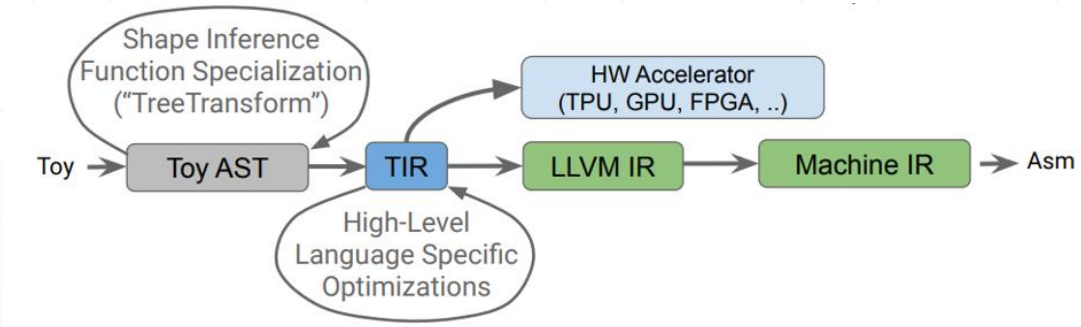
```
.data
  0:
    .code
      PUSH 80                   contract Demo{\n\tuint public ...
      PUSH 40                   contract Demo{\n\tuint public ...
      MSTORE                    contract Demo{\n\tuint public ...
      PUSH 4                    contract Demo{\n\tuint public ...
      CALLDATASIZE                      contract Demo{\n\tuint public ...
      LT                        contract Demo{\n\tuint public ...
      PUSH [tag] 1                      contract Demo{\n\tuint public ...
      JUMPI                     contract Demo{\n\tuint public ...
      PUSH 0                    contract Demo{\n\tuint public ...
      CALLDATALOAD                      contract Demo{\n\tuint public ...
      PUSH 100000000000000000000000000000000000000000000000000000000
      SWAP1                     contract Demo{\n\tuint public ...
      DIV                       contract Demo{\n\tuint public ...
      PUSH FFFFFFFF                     contract Demo{\n\tuint public ...
      AND                       contract Demo{\n\tuint public ...
      DUP1                      contract Demo{\n\tuint public ...
      PUSH 3033413B                     contract Demo{\n\tuint public ...
      EQ                        contract Demo{\n\tuint public ...
      PUSH [tag] 2                      contract Demo{\n\tuint public ...
      JUMPI                     contract Demo{\n\tuint public ...
      DUP1                      contract Demo{\n\tuint public ...
      PUSH 5D33A27F                     contract Demo{\n\tuint public ...
      EQ                        contract Demo{\n\tuint public ...
      PUSH [tag] 3                      contract Demo{\n\tuint public ...
      JUMPI                     contract Demo{\n\tuint public ...
      DUP1                      contract Demo{\n\tuint public ...
```

人工智能：MLIR

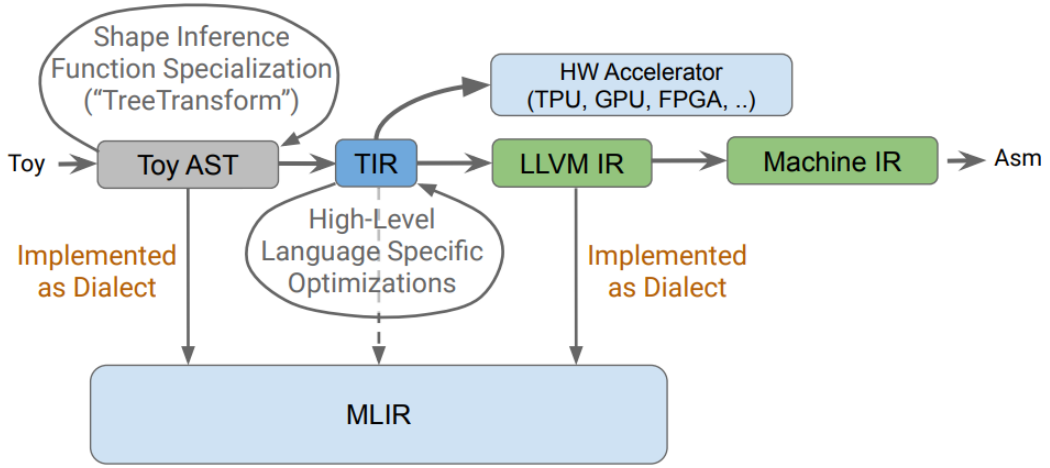

Grappler
TensorFlow Graph
XLA HLO
Tensor RT
nGraph
Core ML
TensorFlow Lite
LLVM IR
TPU IR
Several others
NNAPI
Many others

人工智能：MLIR

Compilers in a Heterogeneous World :



Use MLIR



MLIR_Tutorial.pdf (llvm.org)

人工智能：MLIR

- 类型系统
  - Scalars
    - F16, BF16, F32, …,
    - i1, i8, i16, i32, …, i3, i4, i7, i57
  - Vectors
    - vector<4 x f32>, vector<4x4xf16>,…
  - Tensors
    - tensor<4x4xf32>, tensor<4x?x?x17x?xf32>, …
  - Others

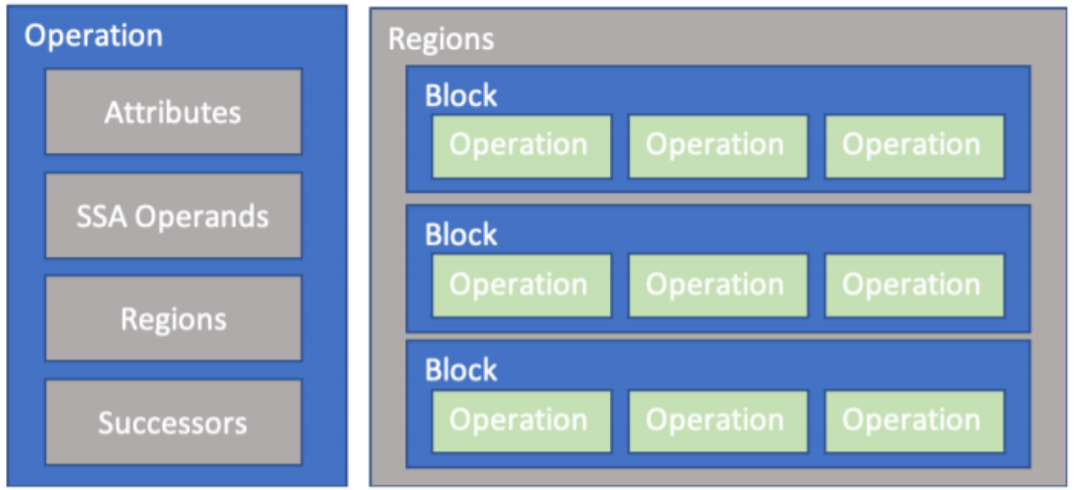


Fig. 1: Operations and Regions in MLIR.

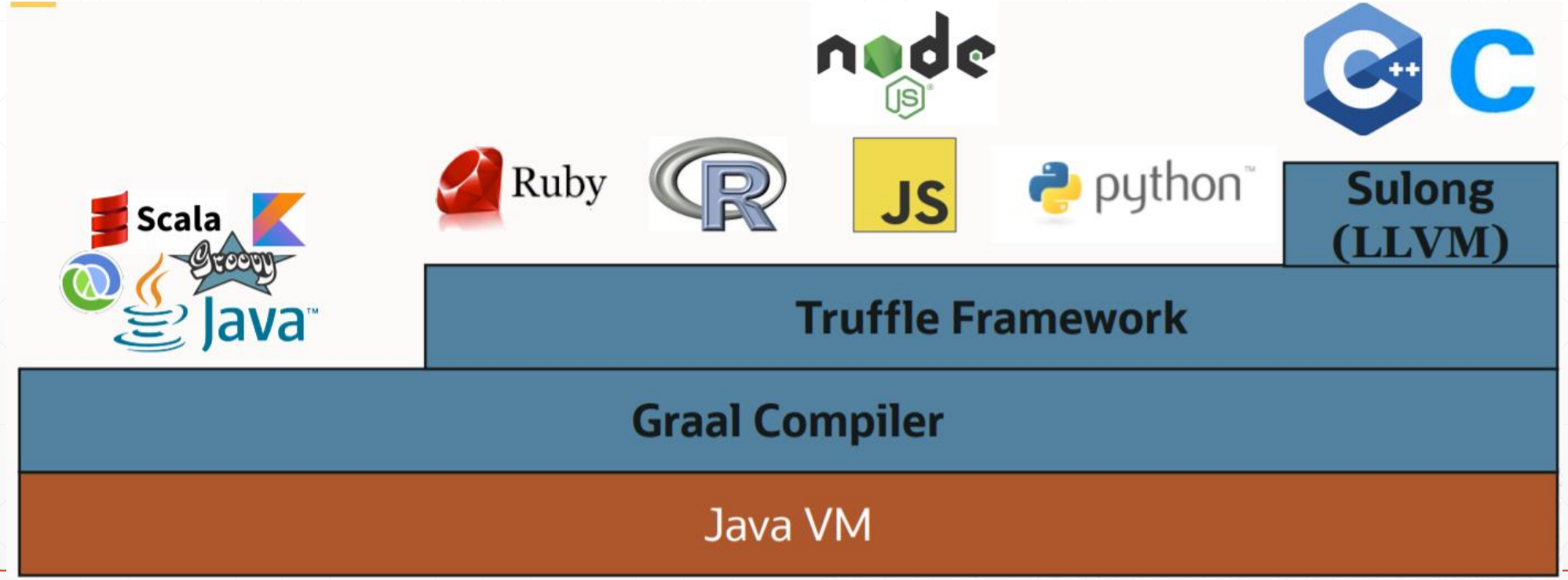
人工智能：MLIR

```
def main() {
  var a<2, 2> = [[1, 2], [3, 4]];
  var b<2, 2> = [1, 2, 3, 4];
  var c = multiply_transpose(a, b);
  print(c);
}
```

```
func @main() {
  %0 = "toy.constant"() { value: dense<[[1., 2.], [3., 4.]]> : tensor<2x2xf64> }
                        : () -> tensor<2x2xf64>
  %1 = "toy.reshape"(%0) : (tensor<2x2xf64>) -> tensor<2x2xf64>
  %2 = "toy.constant"() { value: dense<tensor<4xf64>, [1., 2., 3., 4.]> }
                        : () -> tensor<4xf64>
  %3 = "toy.reshape"(%2) : (tensor<4xf64>) -> tensor<2x2xf64>
  %4 = "toy.generic_call"(%1, %3) {callee: @multiply_transpose}
                        : (tensor<2x2xf64>, tensor<2x2xf64>) -> tensor<*xf64>
  "toy.print"(%4) : (tensor<*xf64>) -> ()
  "toy.return"() : () -> ()
}
```

GraalVM及其中间表示

- GraalVM Architecture
  - Truffle: Language Implementation framework



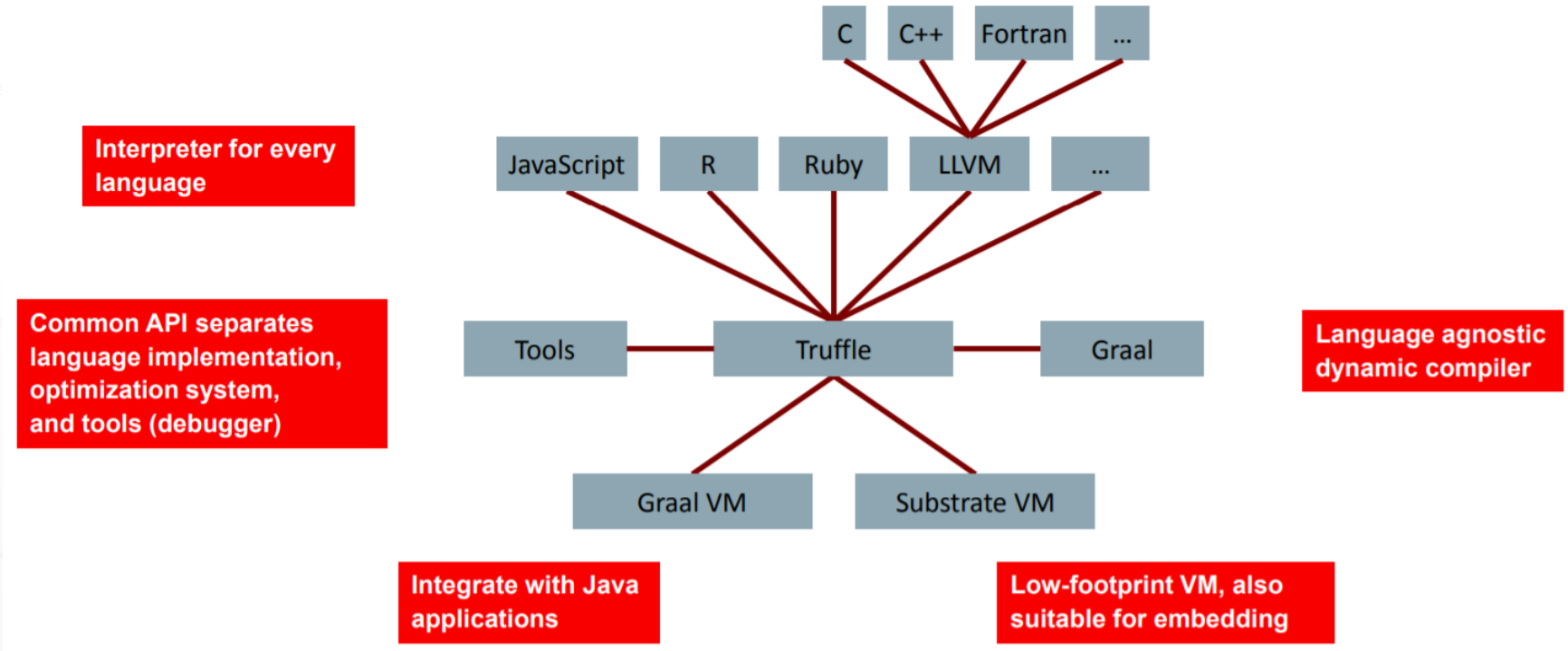
GraalVM及其中间表示

- Polyglot Embeddability

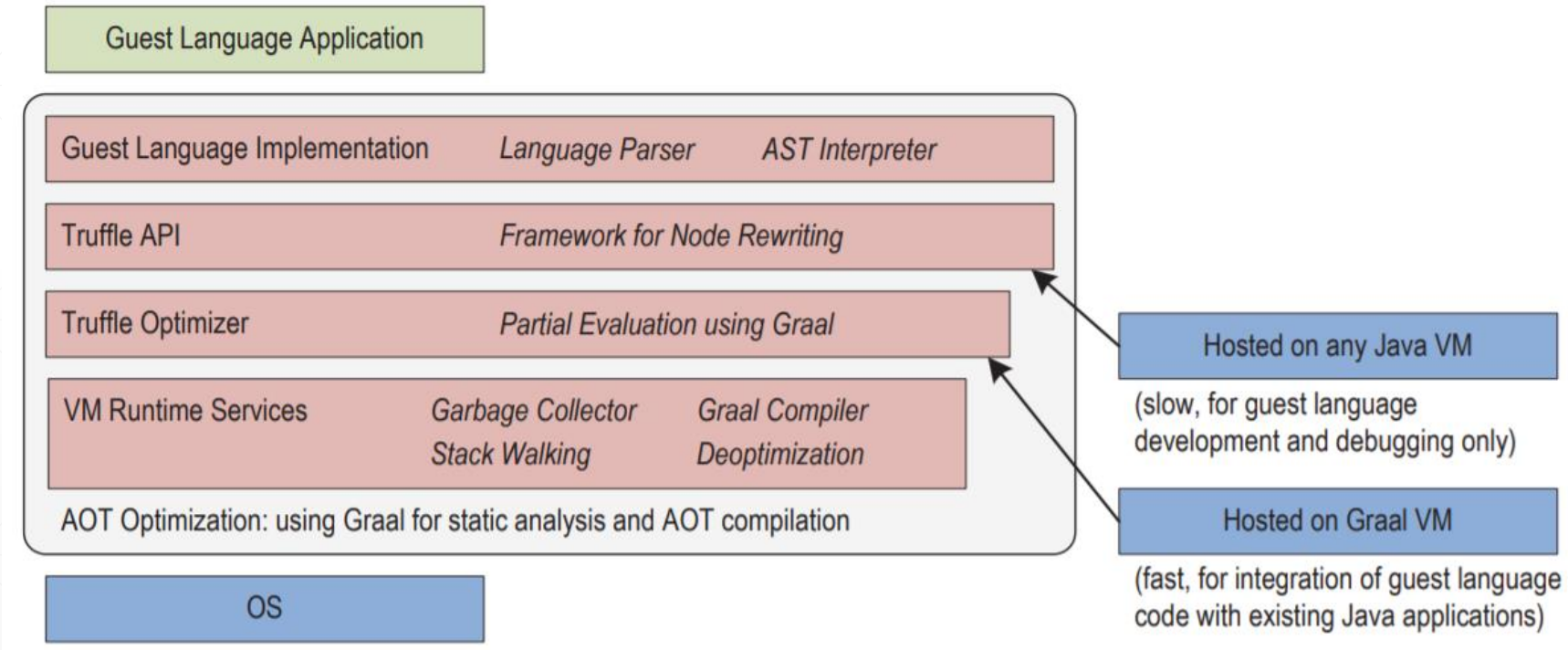
```
import org.graalvm.polyglot.*;


try (Context context = Context.create()) {
    context.eval("js",     "print('Hello JavaScript!');");
    context.eval("R",      "print('Hello R!');");
    context.eval("ruby",   "puts 'Hello Ruby!'");
    context.eval("python", "print('Hello Python!')");
}
```

https://www.graalvm.org/uploads/workshop-nov-2019/2019-11-25-GraalVMCommunityWorkshop-Truffle.pdf

GraalVM及其中间表示

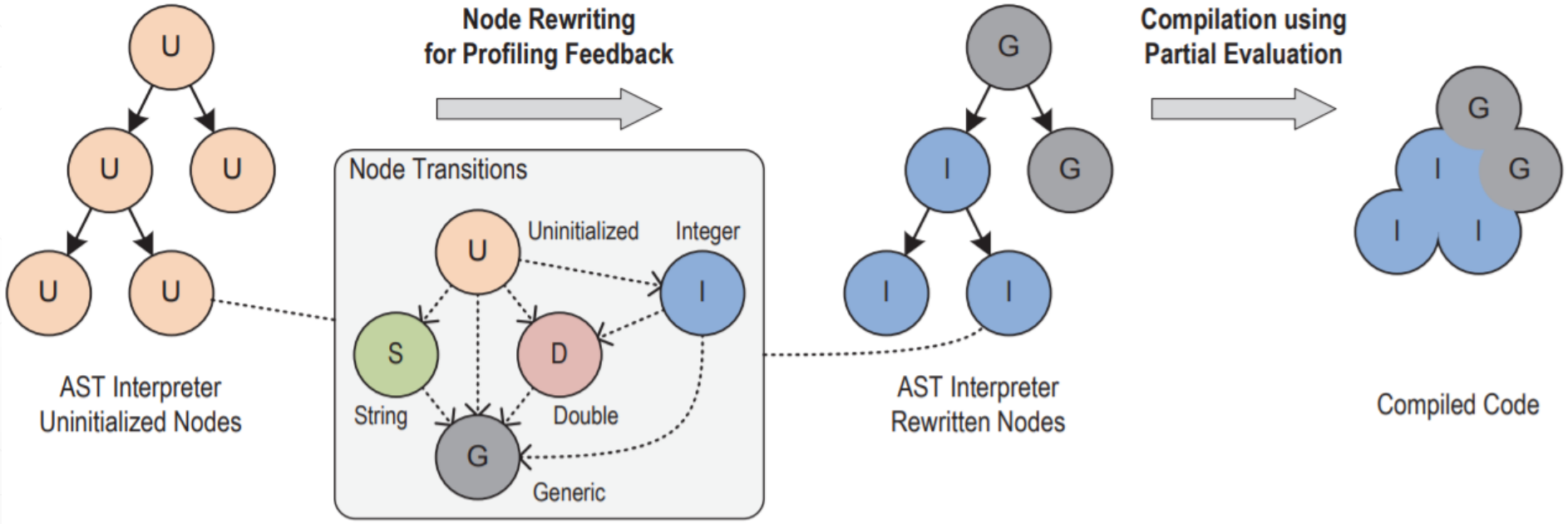


GraalVM及其中间表示



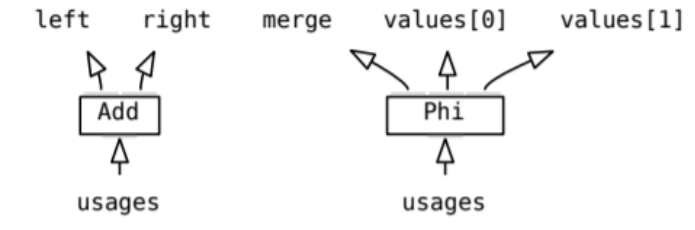
http://lafo.ssw.uni-linz.ac.at/papers/2013_Onward_OneVMToRuleThemAll.pdf

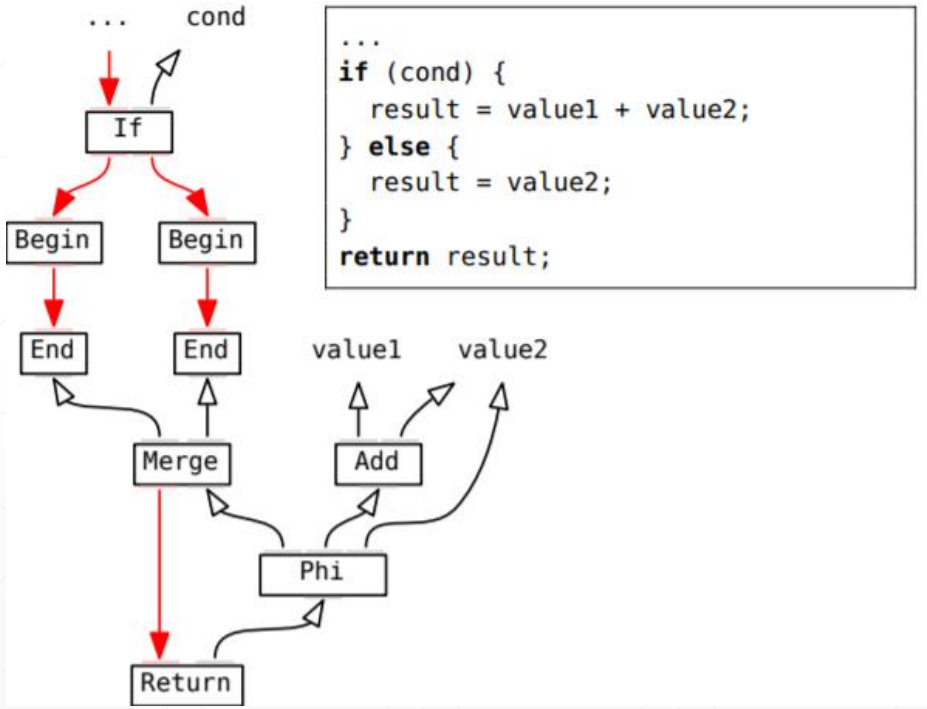
GraalVM及其中间表示



http://lafo.ssw.uni-linz.ac.at/papers/2013_Onward_OneVMToRuleThemAll.pdf

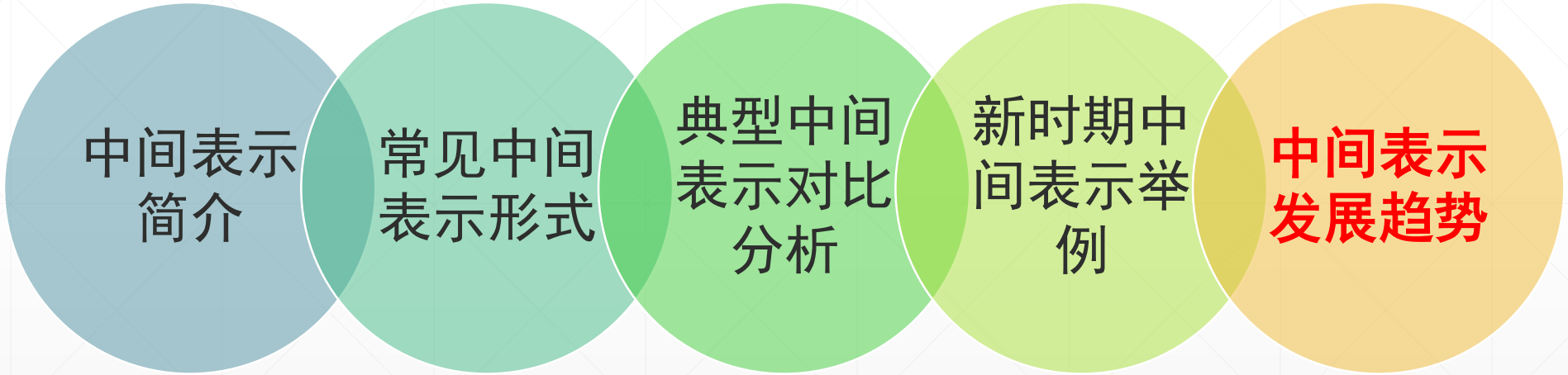
GraalVM及其中间表示





Graal IR: An Extensible Declarative Intermediate Representation

内 容 提 纲



发展趋势

- IR标准化与融合发展
  - 开源社区进一步推动通用标准化IR的发展
  - 统一中间表示体系的优势：多编译协同优化、软件分发部署、多语言互操作、软件生态构造、…
  - 困难和挑战：开源协议、多方协同、不同群体之间的竞争、…
  - 局部趋势已经凸显

发展趋势

- IR标准化与融合发展：两个不同的视角

运行

- computing industry
- emphasizes the virtual machine aspect
- requires less program information

编译

- compiler industry
- focuses on providing good support to the different aspects of compilation
- require greater content in the IR definition

- Completeness
- Semantic gap
- Hardware neutrality
- Manually programmable
- Extensibility

结 语

- 中间表示从封闭走向开放，发挥着更大的作用
- 多层次的中间表示能够覆盖不同的抽象级别和满足不同的任务需求
- 统一中间表示在局部已经凸显，是软件生态构建的重要组成部分
- 个体多样、局部统一、全局共存是短期内的发展现状