



中山大學
SUN YAT-SEN UNIVERSITY

计算机学院（软件学院）

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle 编译原理

第26讲： Summary & Advanced

张献伟

xianweiz.github.io

DCS290, 6/20/2023



中山大學
SUN YAT-SEN UNIVERSITY

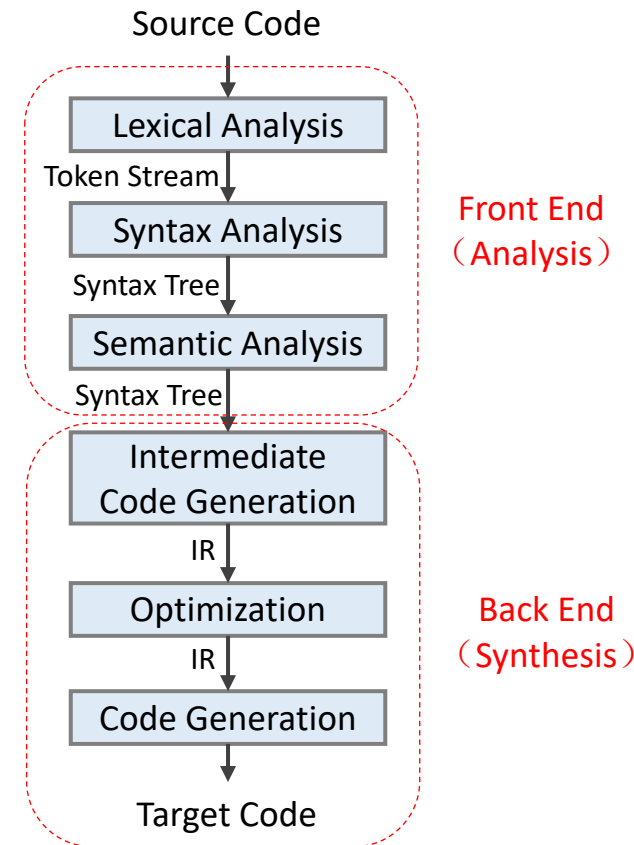


Final Exam

- 考试时间：
 - 6.27/周二，14:30 – 16:30
- 关于试卷
 - 中文（专业术语标注英文）
 - A、B卷，学院指定
- 成绩计算
 - 期末：60%
 - 平时：40%
 - 课堂：15%
 - 作业：25%
- 题型及分值
 - 一、判断题（10分）
 - 10小题，每小题1分
 - 二、填空题（10分）
 - 几个小题，10个空，每空1分
 - 三、简答题（20 - 25分）
 - 3小题，每小题5 - 10分
 - 四、应用题（55 - 60分）
 - 3小题，每小题10 - 25分
- 主要内容
 - 词法分析
 - 语法分析
 - 语义分析
 - 代码生成及优化

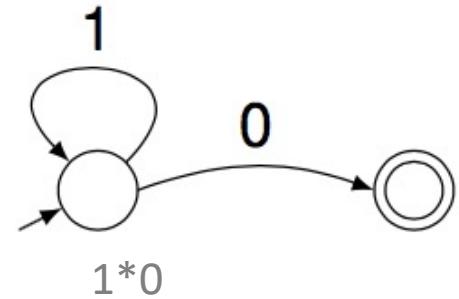
Compilation Phases[编译阶段]

- **Lexical:** source code \rightarrow tokens
 - RE, NFA, DFA, ...
- **Syntax:** tokens \rightarrow AST or parse tree
 - CFG, LL(1), LALR(1), ...
- **Semantic:** AST \rightarrow AST +symbol table
 - SDD, SDT, typing, scoping, ...
- **Int. Code Generation:** AST \rightarrow TAC
 - IR, offset, CodeGen, ...
- **Optimization:** TAC \rightarrow (optimized) TAC
 - BB, CFG, DAG, ...
- **Code generation:** TAC \rightarrow Instructions
 - Instruction, register, stack, ...

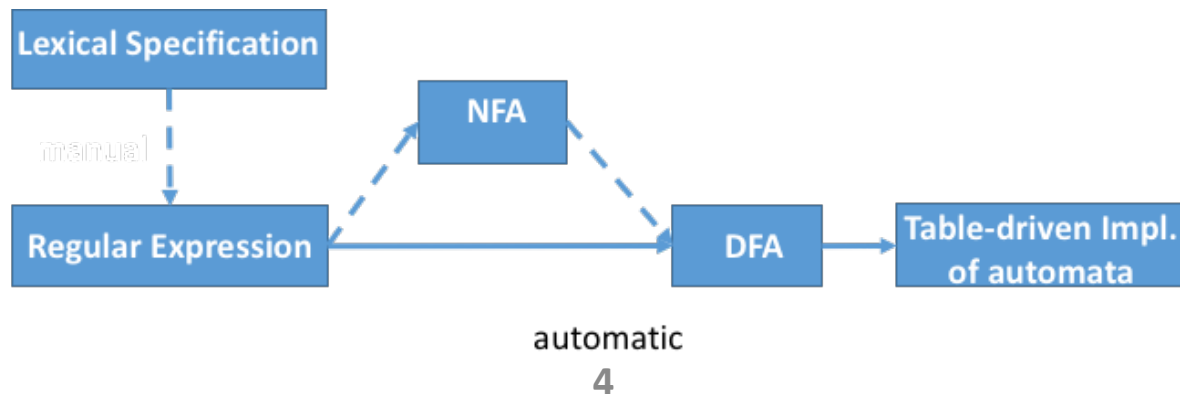


Lexical Analysis[词法分析]

- Characters --> tokens
 - 二元组: <class, lexeme>
- How to specify tokens?
 - Regular expression
 - Atomic, compound
- How to recognize tokens?
 - Transition diagram[转换图]
 - NFA, DFA, table



Any number of '1's followed by a single '0'



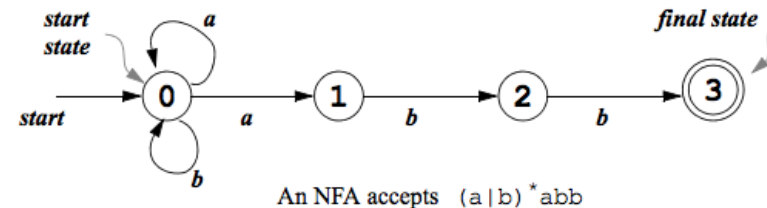
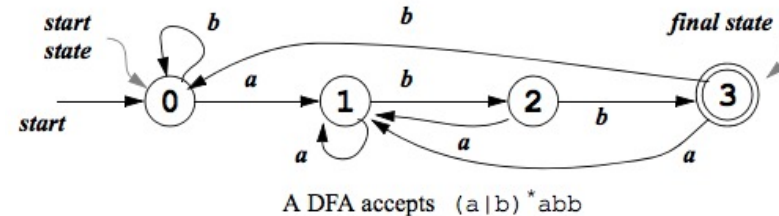
Lexical Analysis (cont.)

- Regular expression

- 自然语言描述 \leftrightarrow 正则表达式
- 正则表达式 \rightarrow NFA \rightarrow DFA
 - M-Y-T算法 (McNaughton-Yamada-Thompson)
 - 子集构建法 (Subset construction)
- 局限性: RE vs. CFG
 - $L = \{a^n b^n \mid n \geq 1\}$ vs. $L = \{a^n b^n \mid 10 \geq n \geq 1\}$

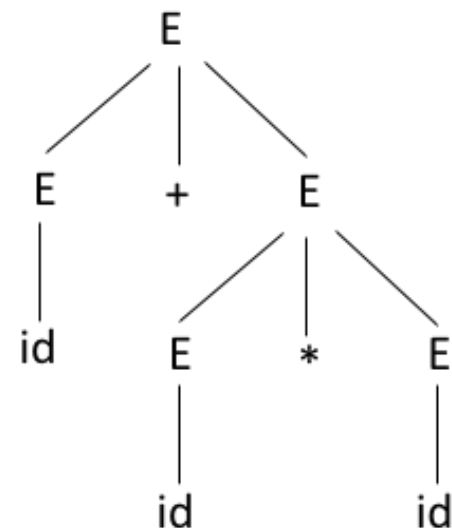
- NFA, DFA

- 状态和边的含义 (ϵ -move)
 - 初始状态、终结状态
- 形式上的区别
- 意义上等价
 - NFA \rightarrow DFA: ϵ -闭包
 - 状态最小化



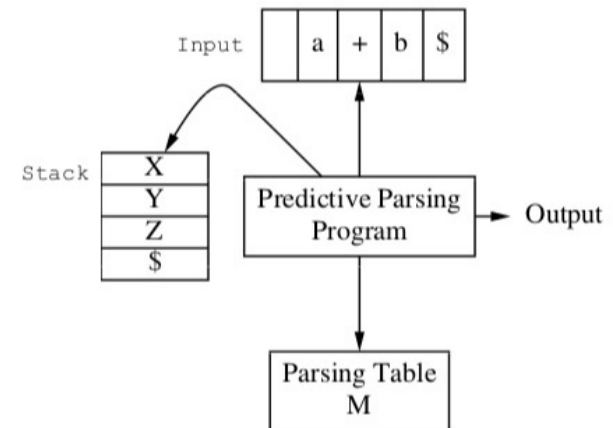
Syntax Analysis[语法分析]

- Tokens --> parse tree
- Context-free grammar
 - 四元组: T, N, s, σ
 - 但通常只写 σ
 - Production rule: LHS --> RHS
 - CFG → 所定义语言的自然描述
- Derivation[推导]
 - Leftmost, rightmost
 - Parse tree: 推导的图形化表示
 - Sentential form[句型]、Sentence[句子]
 - Ambiguity[二义性]
 - 证明及消除 (优先级、结合性)



Syntax Analysis (cont.)

- Parser
 - Top-down: leftmost derivation
 - Bottom-up: reverse order of the rightmost derivation
- Top-down
 - Recursive descent, Predictive/LL(k)
 - Left recursion[左递归]: rewriting
 - Common prefix[共同前缀]: left factoring
- LL(1)
 - Build the parse table
 - FIRST, FOLLOW
 - Use the parse table: expand or match
 - 给定输入串的分析过程
 - Determine if G is LL(1)



Syntax Analysis (cont.)

- Bottom-up

- Shift-reduce

- Handle[句柄]、Viable Prefix[活前缀]、Phrase[短语]、Simple Phrase[直接短语]、Leftmost Simple Phrase[最左直接短语]

- 活前缀不能越过句柄：分析栈存放的都是活前缀，在等句柄出现；一旦出现就规约这个句柄
 - 句柄是一个直接短语

- LR

- More powerful than LL

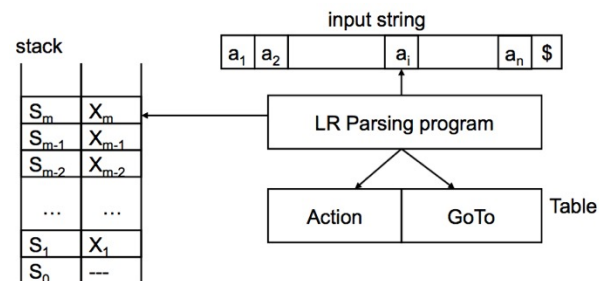
- Parse table:

- Action table: shift/reduce/accept/error
 - Goto table

- LR分析器的工作过程实际上就是逐步产生规范句型的活前缀

- 构造识别所有活前缀的DFA == 构造基于项目集的DFA

- 给定parse table，对输入串进行分析



Syntax Analysis (cont.)

- LR(0): build parse table, construct a FA
 - Item/configuration: initial, reduce, accept
 - State/configuration set: closure()
 - Augmented grammar
 - DFA \rightarrow parse table
 - Conflicts: shift-reduce, reduce-reduce
- Other LR's
 - SLR
 - Improve LR(0): FOLLOW
 - LR(1)
 - LR(1) item: LR(0) item + lookahead symbols
 - Configuration set: closure()
 - LALR(1)
 - YACC/Bison

Semantic Analysis[语义分析]

- Semantic attributes
 - Synthesized, inherited
- Semantic rules or actions
- SDD vs SDT
 - Syntax directed definitions: attributes + semantic rules
 - S-attributed, L-attributed
 - Syntax directed translation scheme: attributes + semantic actions
 - An executable specification of the SDD
- Performed in parsing
 - Top-down: recursive descent, predictive
 - Bottom-up: marker, backpatching
- Annotated parse tree
 - With actions

Code Generation, Optimization[后端]

- Intermediate representation
 - Three-address code
 - CodeGen: variable, array, control, ...
- Runtime/Target code
 - Stack, AR, calling conventions
 - Memory: code, data (global/static, stack, heap)
 - Instruction selection, register allocation, instruction ordering
- Code optimizations
 - Concepts: basic block, flow graph, DAG
 - Optimization: metrics, techniques
 - Peephole, local, loop, global
 - Dead code elimination, common subexpression elimination
 - Strength reduction, constant folding, ...

Compilation

Front-end

Back-end

期末考试分值分布

科研/工作相关性





中山大學
SUN YAT-SEN UNIVERSITY

计算机学院 (软件学院)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

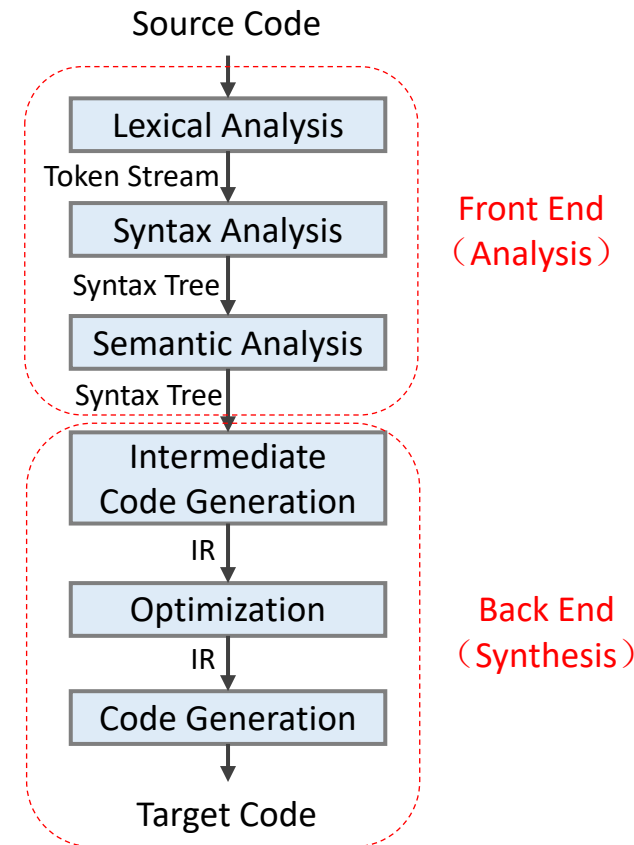
Compilers

in an era of Hardware/Software co-design

DCS290, 06/20/2023

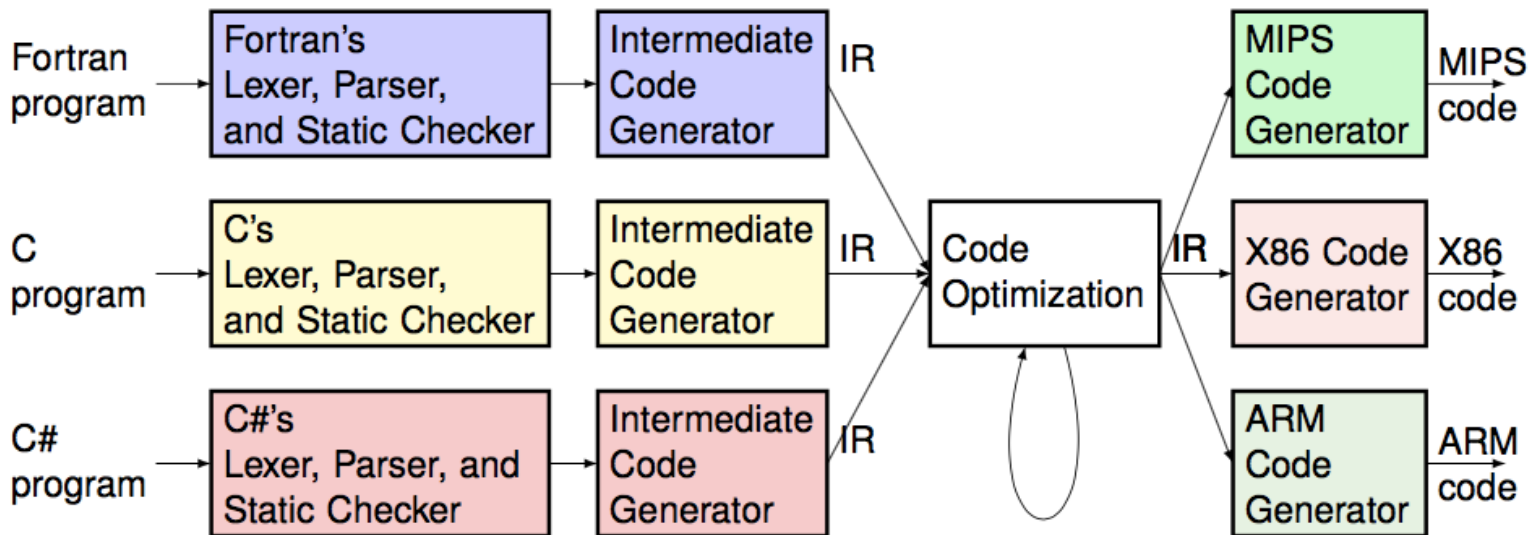
Compilation Phases[编译阶段]

- **Lexical:** source code \rightarrow tokens
 - RE, NFA, DFA, ...
- **Syntax:** tokens \rightarrow AST or parse tree
 - CFG, LL(1), LALR(1), ...
- **Semantic:** AST \rightarrow AST +symbol table
 - SDD, SDT, typing, scoping, ...
- **Int. Code Generation:** AST \rightarrow TAC
 - IR, offset, CodeGen, ...
- **Optimization:** TAC \rightarrow (optimized) TAC
 - BB, CFG, DAG, ...
- **Code generation:** TAC \rightarrow Instructions
 - Instruction, register, stack, ...



Modern Compilers[现代编译器]

- Compilation flow[编译流程]
 - First, translate the source program to some form of intermediate representation (IR, 中间表示)
 - Then convert from there into machine code
- IR provides advantages[IR的优势]
 - Increased abstraction, cleaner separation, and retargeting, etc



Opportunities for Compiler?



Compiler???



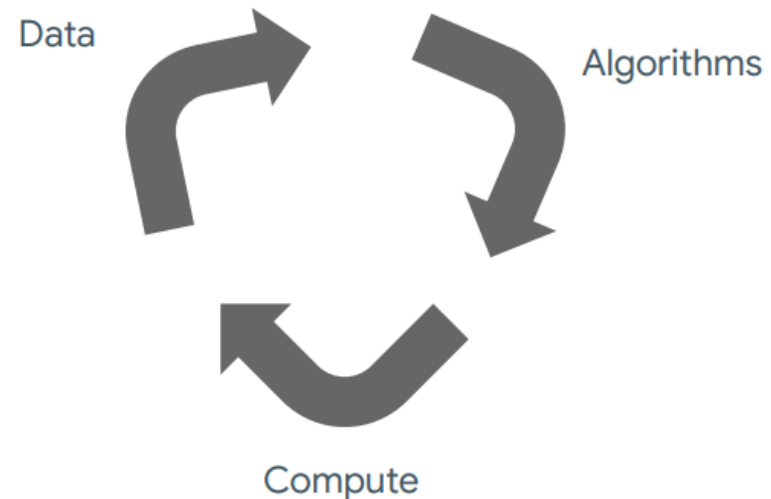
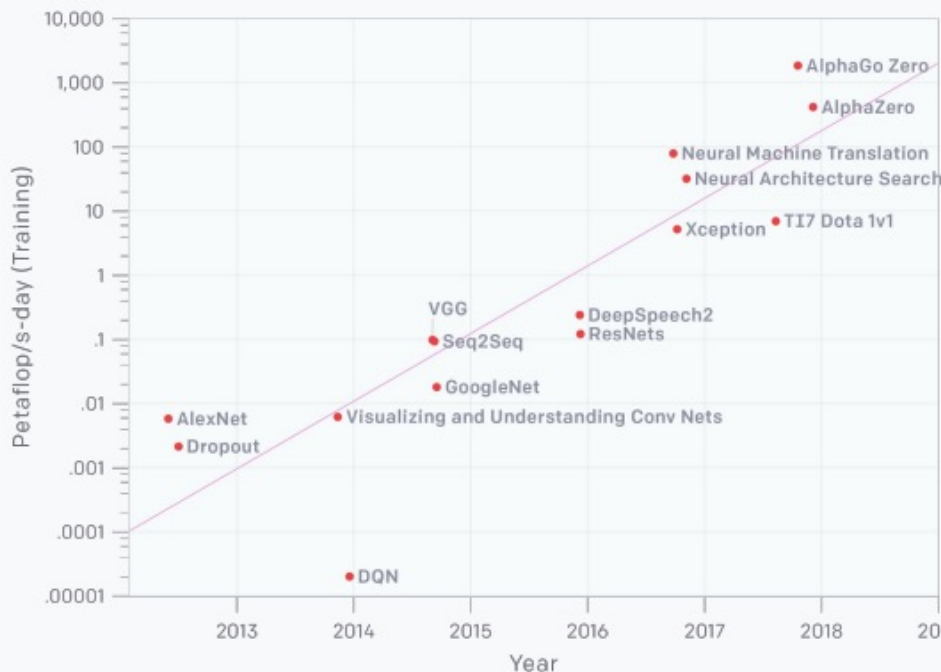
Outline

- The Trends[需求趋势]
 - Application, software, hardware
- The Issues[传统编译的问题所在]
 - Limitations of classical compilers
- The Solutions[潜在编译技术方案]
 - MOJO, TVM, MLIR
- Summary

ML Applications[机器学习应用]

- Models are growing and getting more complex
 - Model Size: larger models require more multiply accumulate operations
 - Model Complexity: as model complexity increases it becomes harder to fully utilize hardware
 - Much faster than Moore's law

AlexNet to AlphaGo Zero: A 300,000x Increase in Compute



[1] [IR Design for Heterogeneity: Challenges and Opportunities](#)

ML Software Explosion[机器学习框架]

- Many frameworks
- Many different graph implementations
- Each framework is trying to gain a usability and performance edge over each other



 PyTorch

 mxnet



ONNX



PaddlePaddle

 [M]^s

MindSpore

High Performance Computing[高性能计算]

- Larger scale applications
 - Climate change, new drug discovery
 - Data analytics, modeling and simulation
- Various parallel programming models
 - MPI, OpenMP, OpenACC, SYCL/DPC++



OpenACC
More Science, Less Programming

SYCL™

NVIDIA.
CUDA®

OpenMP®

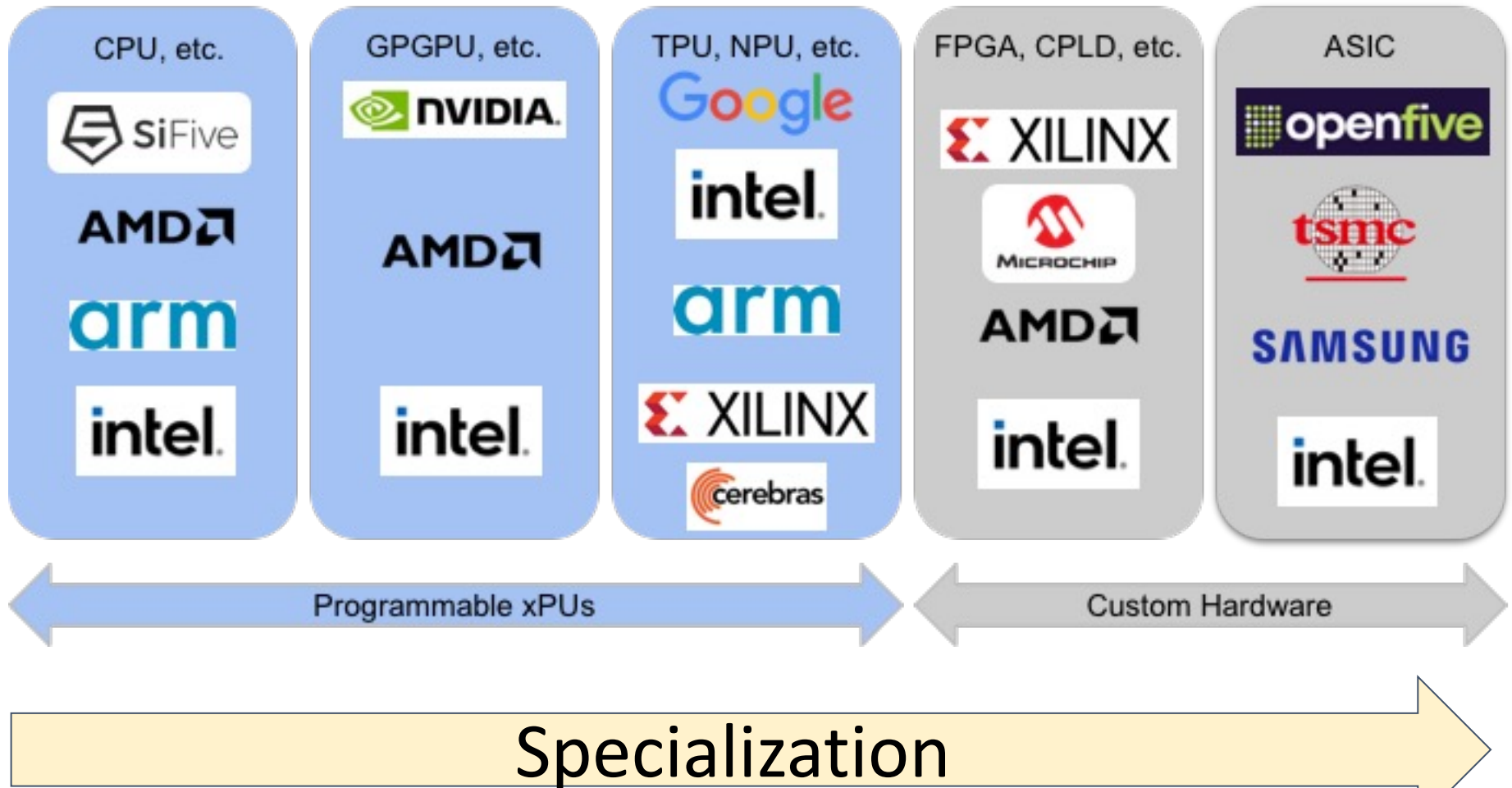
RAJA

AMD
ROCm

1
oneAPI

More Hardware... More Complexity...

- Lots of players! (an incomplete list!)



Domain Specific Arch.[领域专用架构]

- Achieve higher efficiency by tailoring the architecture to characteristics of the domain[体系结构适配领域特性]
 - Not one application, but a domain of applications
 - Different from strict ASIC
 - Requires more domain-specific knowledge than general purpose processors need
- Examples:
 - Neural network processors for machine learning
 - GPUs for graphics, virtual reality
 - Programmable network switches and interfaces

Domain Specific Languages[领域专用语言]

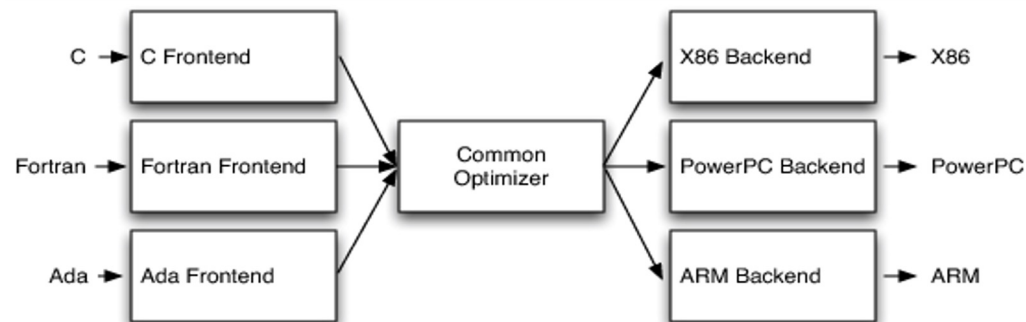
- Invent a new language for the specialized HW
 - Better exploit application knowledge: directly connecting users to HW, bypassing the ISA
- OpenCL is such an example
 - HW vendors can regularly change GPU ISAs without affecting user code
 - OpenCL is the new contract
- It can be a bitter experience for programmers
 - Rewrite code for new HW
 - Not all new languages will survive in the long term

Outline

- The Trends[需求趋势]
 - Application, software, domain specific
- The Issues[问题所在]
 - Limitations of classical compilers
- The Solutions[潜在编译技术方案]
 - MOJO, TVM, MLIR
- Summary

The History[过去的成功经验]

- For more than 50 years, we have enjoyed exponentially increasing compute power[算力急剧增长]
- The growth is based on a fundamental contract between HW and SW[得益于软硬件之间的协议]
 - HW may change radically “under the hood”
 - Old SW can still on new HW (even faster)
 - HW looks the same to SW, always speaking the same language
 - The ISA, allows the decoupling of SW development from HW dev
- Three-phase compiler design (e.g., LLVM)[三段式编译器]
 - One frontend for many backends, one backend for many frontends

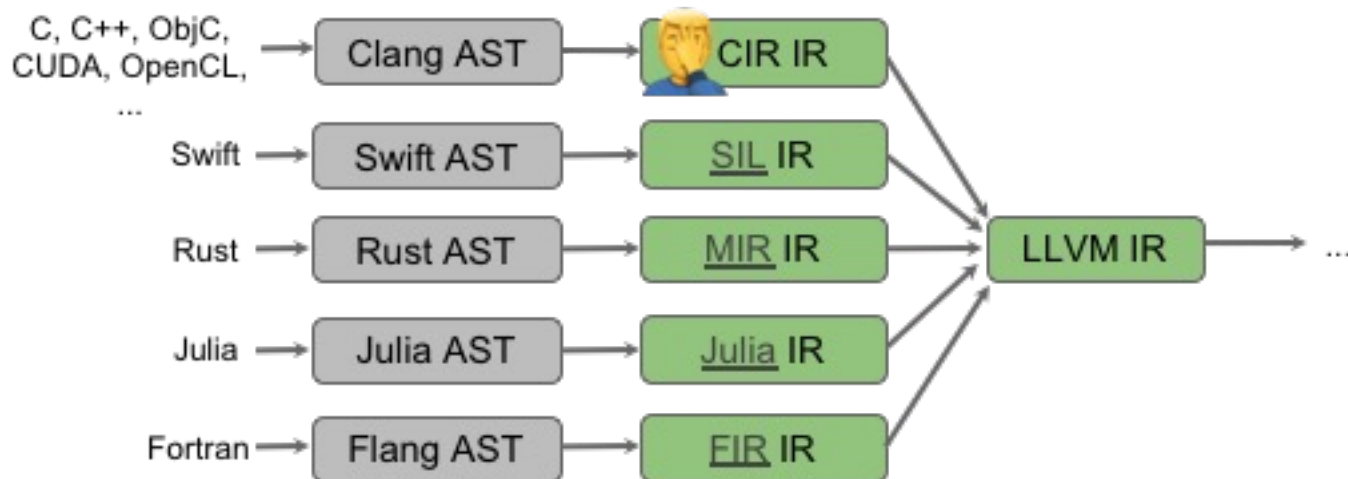


The Issue[当前面临的问题]

- The contract is breaking
 - The end of Moore’s Law forces new design approaches
 - Develop specialized HW to gain massive performance
 - Program and use the specialized and heterogeneous HWs
- Limitations of LLVM
 - “One size fits all” quickly turns into “one size fits none”
 - “fits all”: a single abstraction level to interface with the system
 - LLVM is: 👍 CPUs, “just ok” 👉 for SIMT, but 👎 for many accelerators
 - ... is not great for parallel programming models 🤩
- Many problems are better modeled at a higher- or lower-level abstraction
 - e.g. source-level analysis of C++ code is very difficult on LLVM IR

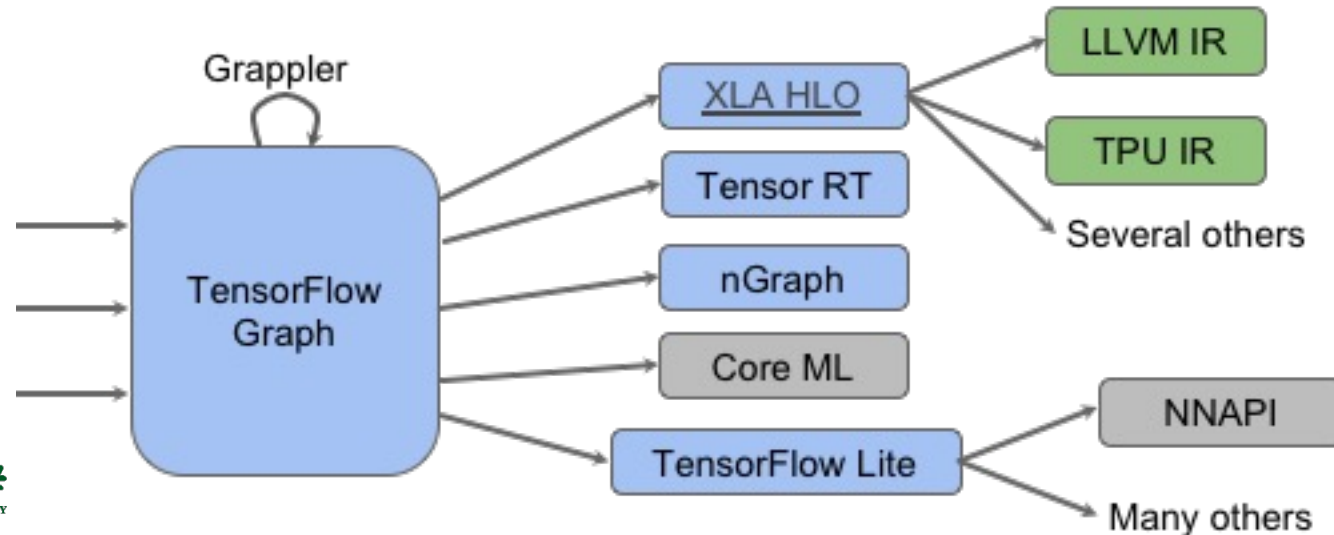
Issue: Modern Languages[编程语言]

- Modern languages pervasively invest in high level IRs[更高级、更抽象]
 - To solve domain-specific problems, like language/library-specific optimizations, flow-sensitive type checking (e.g., for linear types)
 - To improve the implementation of the lowering process
- Each compiler frontend is creating one or more high level IR in addition to their AST representations[上层IR]



Issue: ML Frameworks[机器学习框架]

- Compiler tech is widely deployed in others fields, including machine learning frameworks
- ML systems typically use “ML graphs” as a domain-specific abstraction
- TensorFlow is basically a huge compiler ecosystem
 - These boxes are all different domain-specific compiler systems:
 - Different limitations, challenges, owners, etc
 - No unifying theory and infrastructure to support this



Next-Gen Compilers & PL are Needed

- We need:
 - Hardware abstraction spanning diverse accelerators
 - Support for heterogeneous compute platforms
 - Domain specific languages and programming models
 - Quality, reliability, and scalability of infrastructure
- We see:
 - “No one size fits all” compiler
 - Shape of the problem is the same, but the accel details always vary
- This opportunity is beckoning a golden age in compiler and PL technology!

Outline

- The Trends[需求趋势]
 - Application, software, domain specific
- The Issues[问题所在]
 - Limitations of classical compilers
- The Solutions[潜在编译技术方案]
 - MOJO, TVM, MLIR
- Summary

- A binary instruction format for a stack-based VM
 - Designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.
- Features
 - “Byte code for the web”
 - Very rough idea: “JVM, without 3rd party plugins, directly in browser”
 - Lower-level than JavaScript
 - Fast
 - Compact binary format → quicker to parse
 - Instructions map closely to common hardware
 - Linear memory, no garbage collector
 - Safe: Typed, separated code and data, modules, ...
 - Portable: Support by all 4 major browsers, x86/ARM

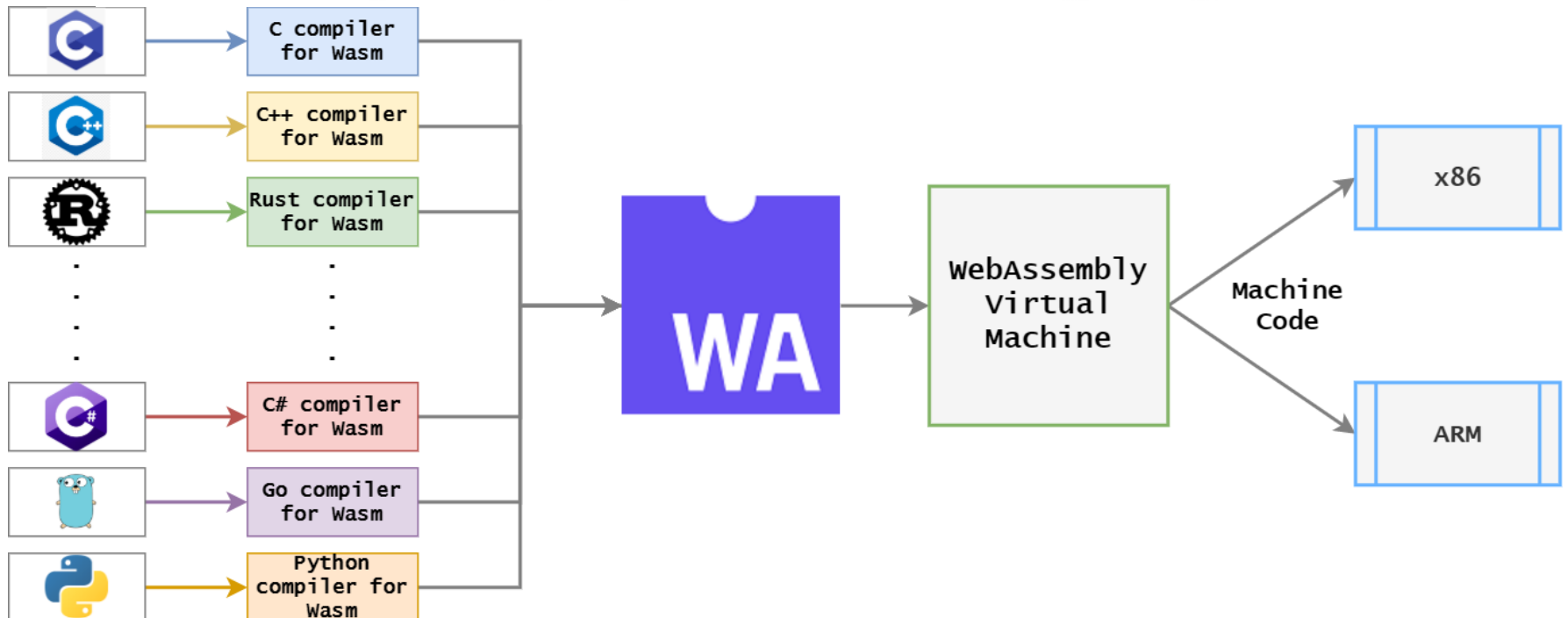
Bringing the Web up to Speed with WebAssembly

Andreas Haas Andreas Rossberg Derek L. Schuff* Ben L. Titzer
 Google GmbH, Germany / *Google Inc, USA
 {ahaas,rossberg,dschuff,titzer}@google.com

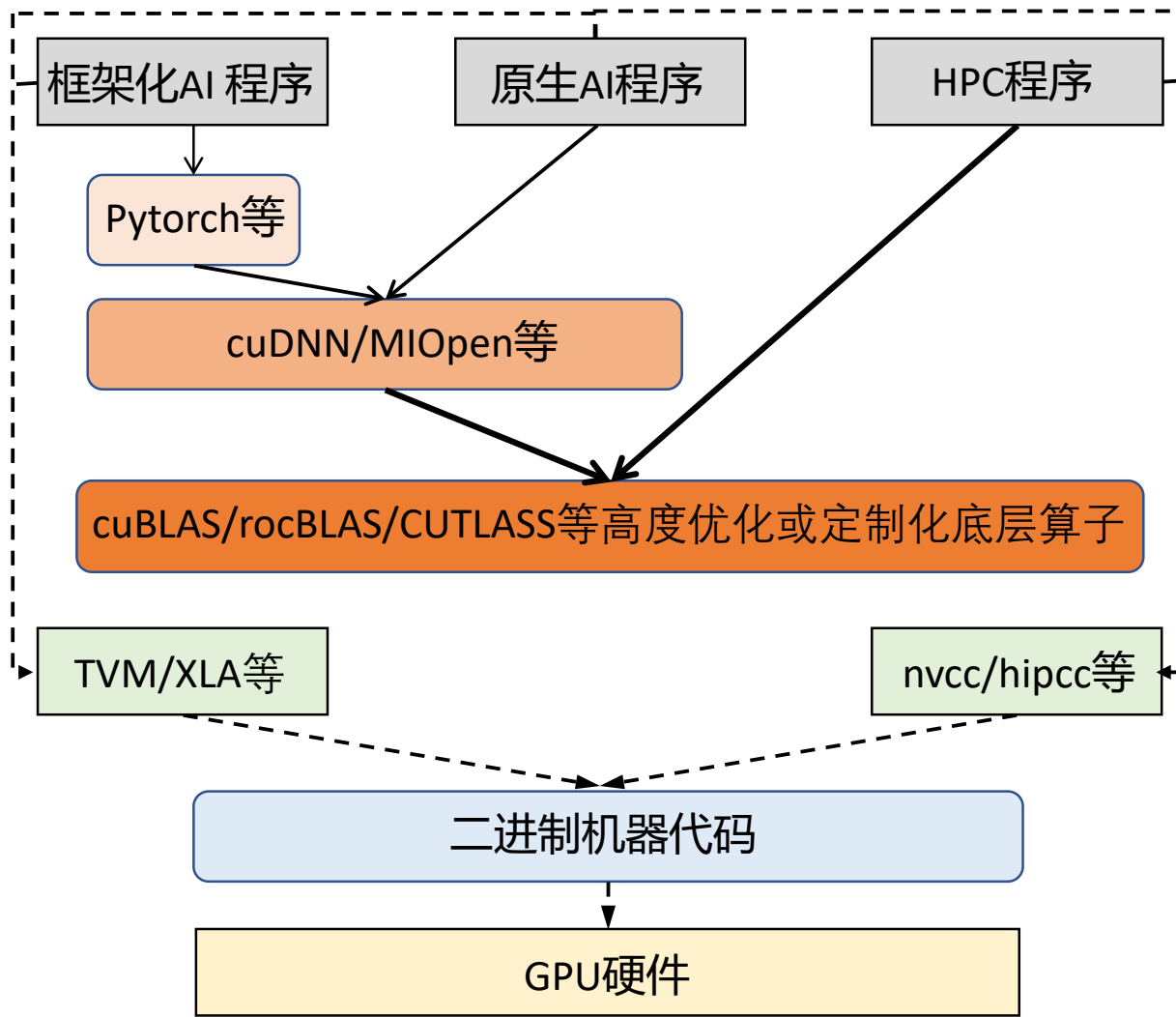
Michael Holman
 Microsoft Inc, USA
 michael.holman@microsoft.com

Dan Gohman Luke Wagner Alon Zakai
 Mozilla Inc, USA
 {sunfishcode,luke,azakai}@mozilla.com


JF Bastien
 Apple Inc, USA
 jfbastien@apple.com



ML Workflow





- A new programming language for all AI developers
 - Combines the usability of Python with the performance of C, unlocking unparalleled programmability of AI hardware and extensibility of AI models.
- Features
 - Progressive types: leverage types for better performance and error checking
 - Zero cost abstractions: take control of storage by inline-allocating values into structures
 - Portable Parametric Algorithms: leverage compile-time meta-programming to write hardware-agnostic algorithms and reduce boilerplate
 - Language Integrated Auto-tuning: automatically find the best values for your parameters to take advantage of target hardware
 - The full power of MLIR + Parallel heterogeneous runtime + Fast compile times

○○○ SOFTMAX. 

MOJO  

```
def softmax(lst):  
    norm = np.exp(lst - np.max(lst))  
    return norm / norm.sum()  
  
struct NDArray:  
    def max(self) -> NDArray:  
        return self.pmap(SIMD.max)  
  
struct SIMD[type: DType, width: Int]:  
    def max(self, rhs: Self) -> Self:  
        return (self >= rhs).select(self, rhs)
```

LANGUAGES	TIME (s) *	SPEEDUP VS PYTHON
PYTHON 3.10.9	1027 s	1x
PYPY	46.1s	22x
SCALAR C++	0.20 s	5000x
 MOJO 	0.03 s	35000x

TVM

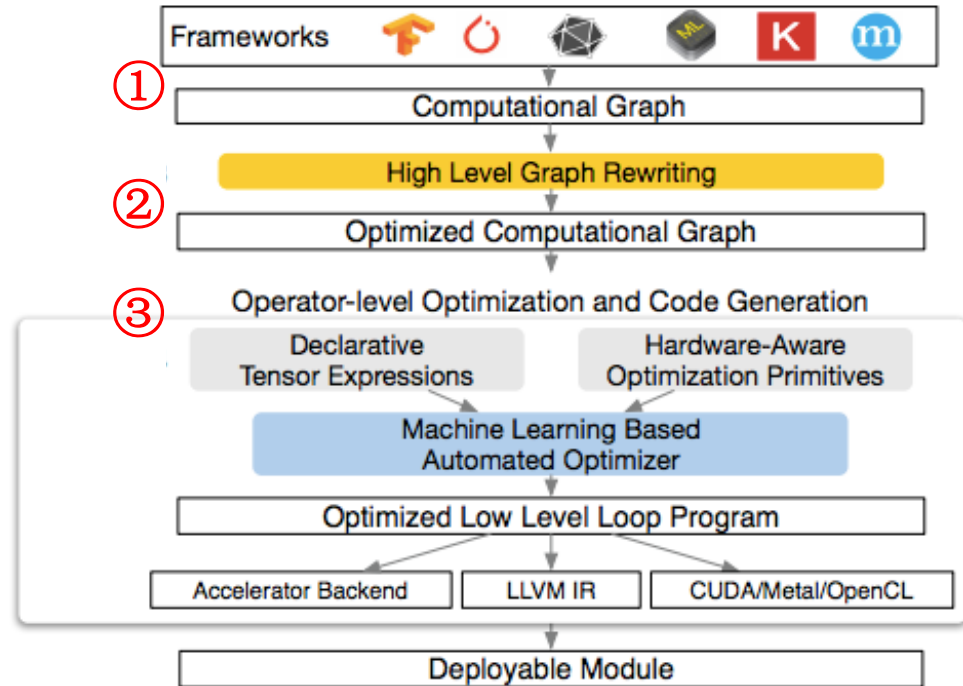
- Bring ML to a wide diversity of hardware devices
 - Current frameworks rely on vendor-specific operator libraries and optimize for a narrow range of server-class GPUs
 - Deploying workloads to new platforms – such as mobile phones, embedded devices, and accelerators (e.g., FPGAs, ASICs) – requires significant manual effort
- TVM: an end to end ML compiler framework for CPUs, GPUs and accelerators
 - Aims to enable machine learning engineers to optimize and run computations efficiently on any hardware backend

[1] [TVM: An Automated End-to-End Optimizing Compiler for Deep Learning](#), OSDI'2018

[2] [Apache TVM](#),

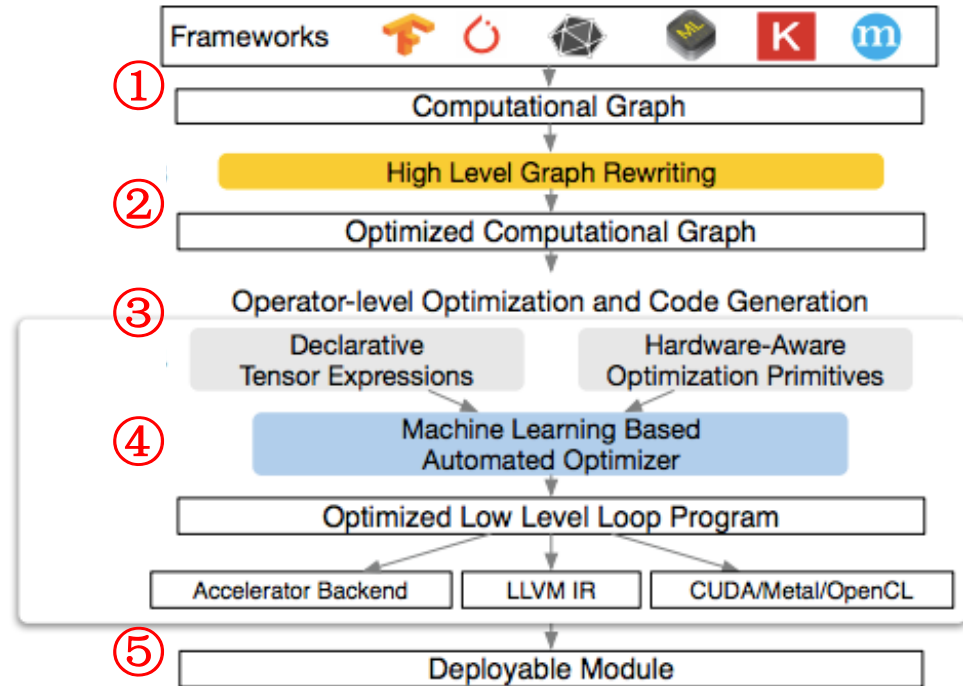
TVM (cont.)

- Execution steps in TVM
 - ① First takes as input a model from an existing framework and transforms it into a computational graph representation
 - ② Then performs high-level dataflow rewriting to generate an optimized graph
 - ③ The operator-level optimization module must generate efficient code for each fused operator in this graph



TVM (cont.)

- Execution steps in TVM
 - ④ TVM identifies a collection of possible code optimizations for a given hardware target's operators
 - Possible optimizations form a large space, so we use an ML-based cost model to find optimized operators
 - ⑤ Finally, the system packs the generated code into a deployable module



MLIR: Multi-Level Intermediate Representation

- MLIR: Compiler Infra at the End of Moore's Law
 - Joined LLVM, follows open library-based philosophy
 - **Modular**, extensible, general to many domains
 - Being used for CPU, GPU, TPU, FPGA, HW, quantum,
 - Easy to learn, great for research
 - MLIR + LLVM IR + RISC-V CodeGen = 🇹🇼🇹🇼



<https://mlir.llvm.org>

MLIR (cont.)

- MLIR is a novel approach to building reusable and extensible compiler infrastructure
 - Addresses software fragmentation, compilation for heterogeneous hardware
 - Significantly reducing the cost of building domain specific compilers, and connecting existing compilers together
- MLIR is intended to be a hybrid IR which can support multiple different requirements in a unified infrastructure
 - The ability to represent dataflow graphs (such as in TensorFlow)
 - Ability to host HPC-style loop optimizations across kernels, and to transform memory layouts of data
 - Ability to represent target-specific operations, e.g. accelerator-specific high-level operations.
 - Quantization and other graph transformations done on a Deep-Learning graph.

[1] [MLIR: Scaling Compiler Infrastructure for Domain Specific Computation](#), CGO'2021

[2] [Building a Compiler with MLIR](#), Google'2020

Compilers for ML \rightarrow ML for Compilers

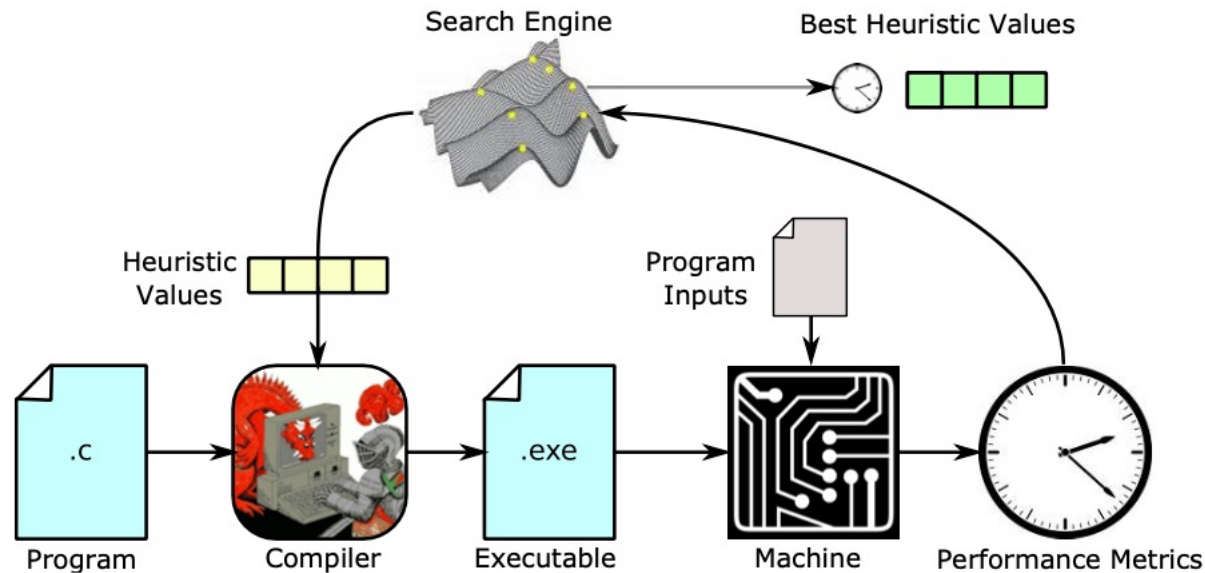


Fig. 1. Iterative Compilation: a search technique explores a space of compilation strategies, continually compiling, executing and profiling to find the best performing strategy.

- Machine Learning in Compilers: Past, Present and Future, <https://ieeexplore.ieee.org/document/9232934>
- Profile Guided Optimization without Profiles: A Machine Learning Approach, <https://arxiv.org/abs/2112.14679>
- VESPA: static profiling for binary optimization, <https://dl.acm.org/doi/abs/10.1145/3485521>

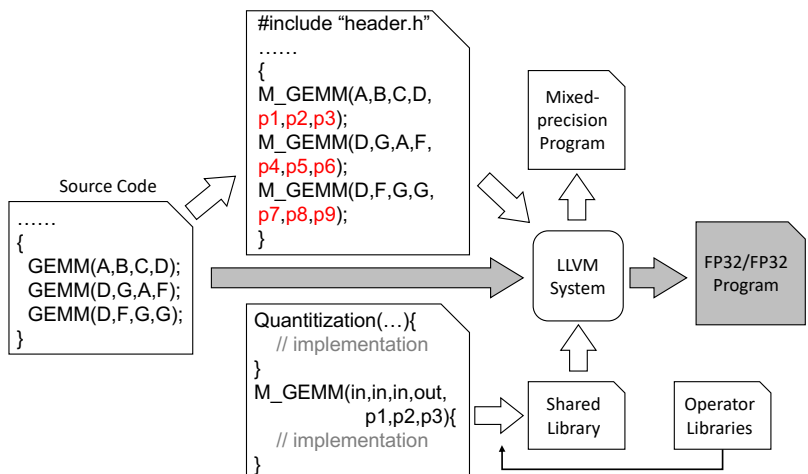
Outline

- The Trends[需求趨勢]
 - Application, software, domain specific
- The Issues[問題所在]
 - Limitations of classical compilers
- The Solutions[潛在編譯技術方案]
 - MOJO, TVM, MLIR
- Summary

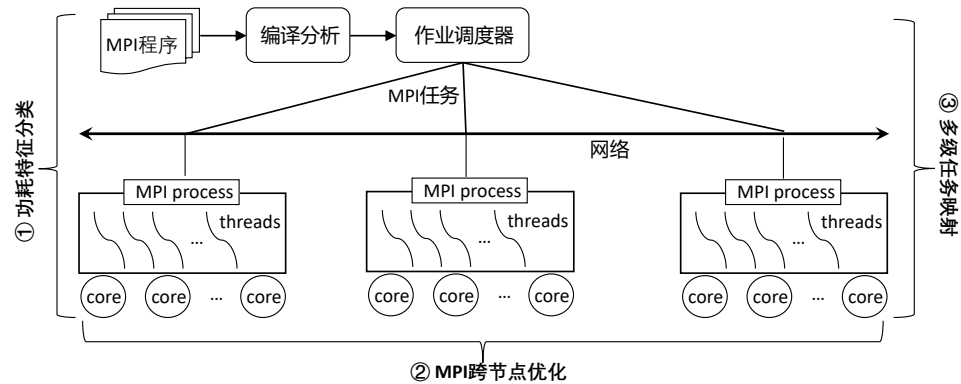
Summary

- Compiler/PL tech more important than ever!
 - The world is evolving fast at the “End of Moore’s Law”
 - Changing assumptions, expanding possibilities
- HW changes require new programming models and approaches:
 - Various models and frameworks
 - More high-level semantics
- We need compiler and PL experts to step up!

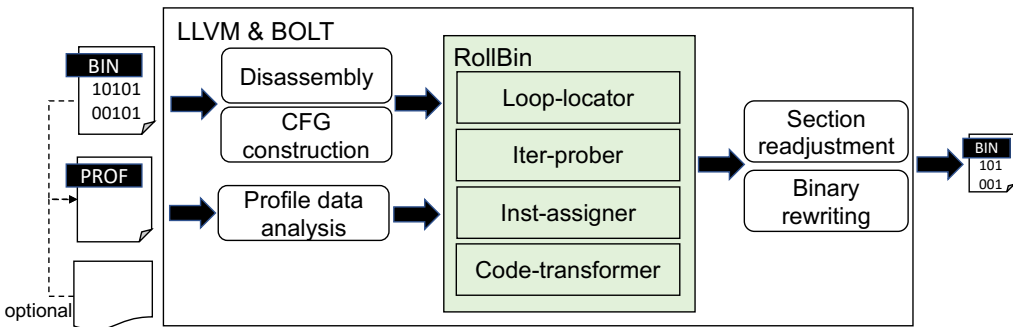
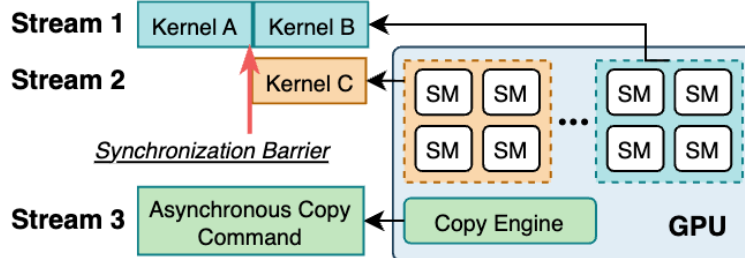
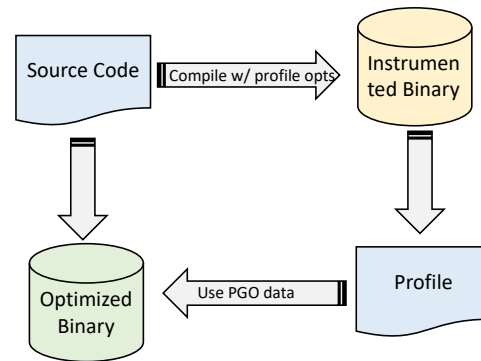
Our Work



moTuner: A Compiler-based Auto-tuning Approach for Mixed-precision Operators [CF'22]



② MPI跨节点优化 (MPI cross-node optimization)



RollBin: Reducing Code-size via Loop Rerolling at Binary Level [LCTES'22]

References

- [1] Chris Lattner, [The Golden Age of Compiler Design in an Era of HW/SW Co-design](#), Keynote @ International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Apr 2021.
- [2] Albert Cohen, [IR Design for Heterogeneity: Challenges and Opportunities](#), Keynote @ International Conference on Compiler Construction (CC), Feb 2020.
- [3] John Hennessy and David Patterson, [A New Golden Age for Computer Architecture](#), Turing Lecture @ The International Symposium on Computer Architecture (ISCA), June 2018.
- [4] Luis Ceze, Mark D. Hill and Thomas F. Wenisch, [Arch2030: A Vision of Computer Architecture Research over the Next 15 Years](#), Workshop @ The International Symposium on Computer Architecture (ISCA), June 2016.