Compilers 101

Intro

Course overview

1. Intro (this lecture)
2. Compiler frontend
3. Static analysis
4. Intermediate representations
5. Optimizations
6. Backend
7. Debuggers
8. Heterogeneous compilers

Contacts

Lectures: Оболенский Арсений
- E-mail: me@gooddoog.ru
- Via portal.unn.ru

Practice: Лычков Михаил
- E-mail: mikelitch@gmail.com

Practice: Кузнецов Артём
- E-mail: artyomka6192@gmail.com

GitHub organization: https://github.com/NN-complr-tech
Lecture materials repository: https://github.com/NN-complr-tech/Complr-course-lectures

What is a compiler?

First compilers

A-0, 1952

- Grace Hopper at Remington Rand
- Not a real compiler, functioned more as a loader and a linker
- Photo: https://en.wikipedia.org/wiki/Grace_Hopper

Fortran

1954-1957

- John Backus at IBM
- Fortran is still used and actively developed by ISO. The last release is Fortran 2018.
- Photo: https://en.wikipedia.org/wiki/John_Backus

Lisp

1958

- John McCarthy at MIT
- Second-oldest high-level programming language
- Photo: https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)

C

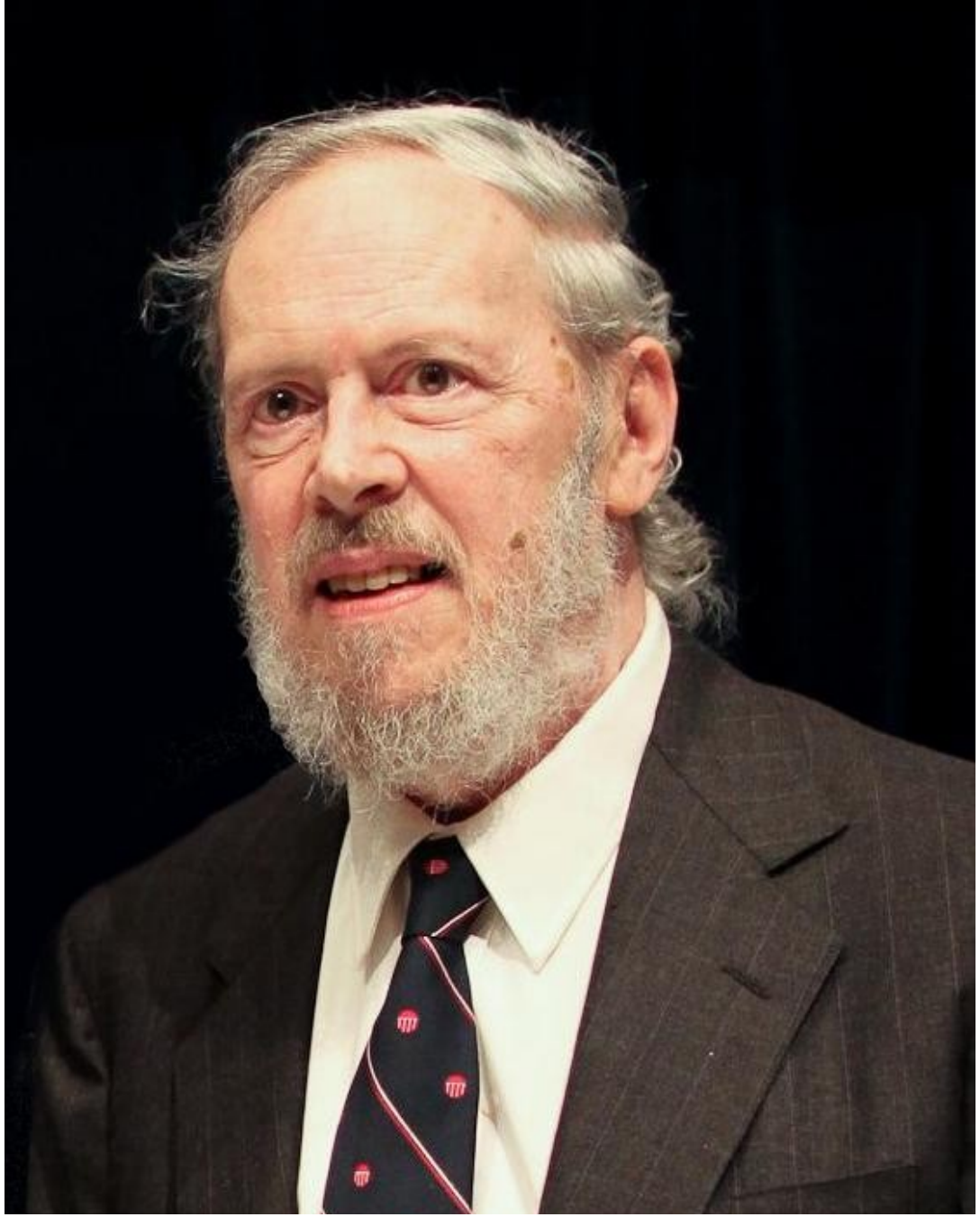
1972

Dennis Ritchie at Bell Labs

Developed the first C compiler (derived from the B language)

Became the backbone of the Unix operating system

- Photo: https://en.wikipedia.org/wiki/Dennis_Ritchie



C++

1979

Created by Bjarne Stroustrup at Bell Labs

Evolved from C to add object-oriented programming features

First commercial implementation released in 1985

- Photo: https://www.stroustrup.com/

Beyond software

Hardware description languages

- Verilog – 1984
- VHDL – 1987

Heterogenous calculation languages

- OpenCL – 2009
- SYCL – 2014

- Photo: https://www.flickr.com/photos/intelfreepress/8267616249/in/photostream/lightbox/

OpenCL

- Purpose: Designed for creating programs that leverage the parallel computing capabilities of heterogeneous platforms.
- Cross-Platform: Supports a wide range of devices from various manufacturers, including GPUs, CPUs, DSPs, and FPGAs.
- Parallel Programming Model: Enables efficient execution of high-performance computing, graphical, and real-time applications.

```
__kernel void vector_add(__global const float* A, __global const float* B, __global float* C) {
    // Get the index of the current element
    int i = get_global_id(0);

    // Perform the addition
    C[i] = A[i] + B[i];
}
```

SYCL

SYCL (pronounced "sickle") is a high-level programming model built on top of OpenCL that enables developers to write code for heterogeneous devices (such as CPUs, GPUs, DSPs, and FPGAs) using standard C++.

It abstracts away much of the boilerplate code associated with OpenCL, allowing for easier development of parallel computing applications.

```cpp
#include <CL/sycl.hpp>

int main() {
    using namespace cl::sycl;
    queue q;

    // Define the size of the vectors
    const int size = 1024;
    // Create two input vectors and one output vector
    std::vector<float> A(size, 1.0f), B(size, 2.0f), C(size);

    // Create buffers from the vectors
    buffer<float> bufA(A.data(), range<1>(size));
    buffer<float> bufB(B.data(), range<1>(size));
    buffer<float> bufC(C.data(), range<1>(size));

    // Submit a command group to the queue to perform the computation
    q.submit([&](handler& h) {
        // Create accessors for the buffers
        auto a = bufA.get_access<access::mode::read>(h);
        auto b = bufB.get_access<access::mode::read>(h);
        auto c = bufC.get_access<access::mode::write>(h);

        // Define the kernel
        h.parallel_for(range<1>(size), [=](id<1> idx) {
            c[idx] = a[idx] + b[idx];
        });
    });

    // The destructor of the buffer c will copy the data back to C
    return 0;
}
```

Why study compilers?

- Write better software
  - Compiler is a tool, mastering this tool will help you write fast and reliable software
- Compilers are not only about code generation
  - Static analysis, code-to-code conversion
  - Example: Intel® DPC++ Compatibility Tool (aka SYCLomatic), https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compatibility-tool.html#gs.p25a6d
- Compiler writers get paid!

Course emphasis

- Practice oriented
  - Study modern compiler tooling and infrastructure
  - Lab assignments
- Provide overview of basic algorithms and theory
- Non-goal: how to build a complete compiler optimizer
  - There are lots of links on the slides, make sure to check them out if you want to know more

The rest of this lecture

- Hardware architecture refresh
  - CPUs, GPUs, FPGAs
  - Just enough to understand the problem
- Quick compiler architecture introduction
- Overview of practical applications

CPUs

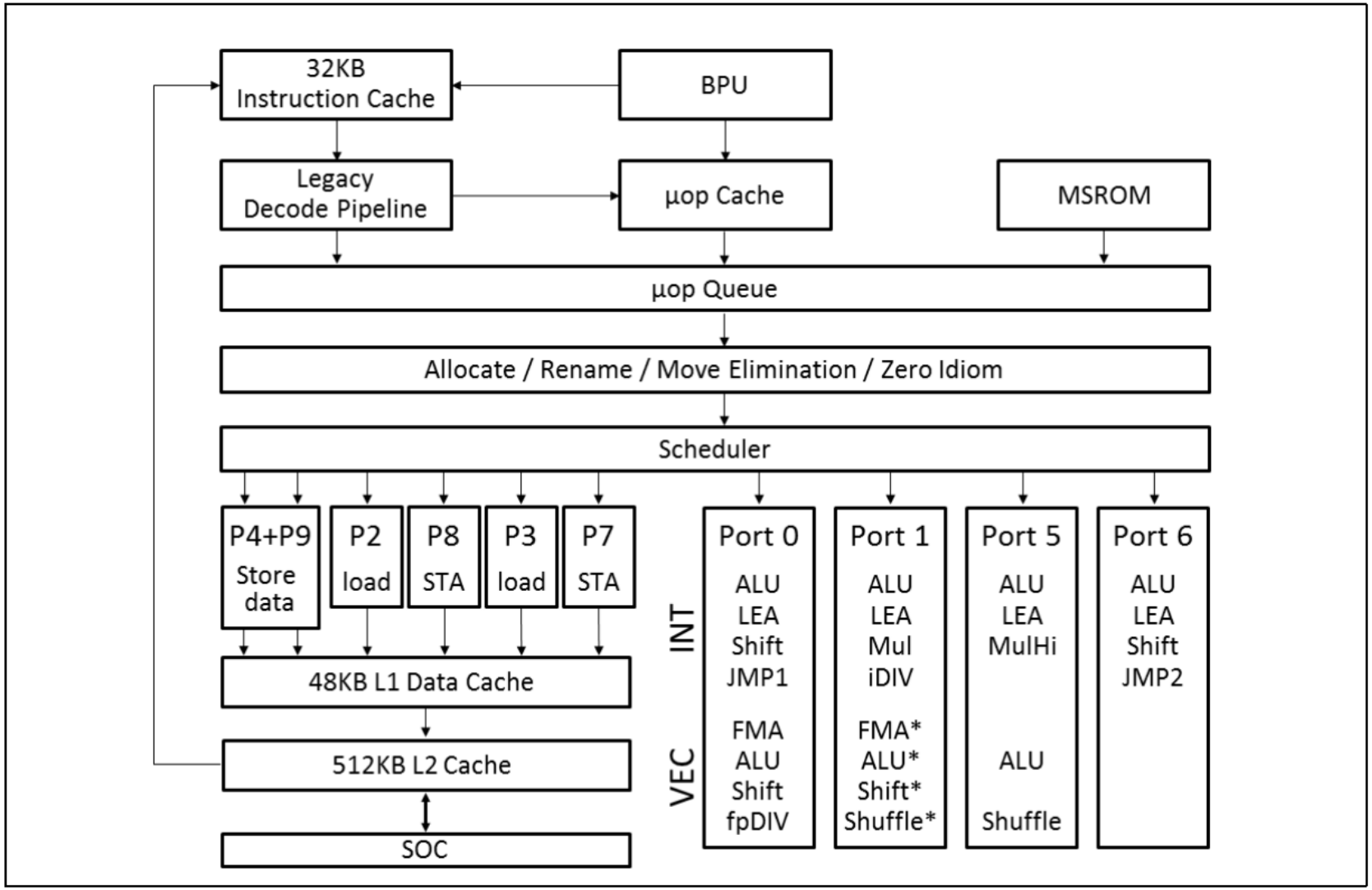


Figure 2-1. Processor Core Pipeline Functionality of the Ice Lake Client Microarchitecture[1]

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

CPU pipeline

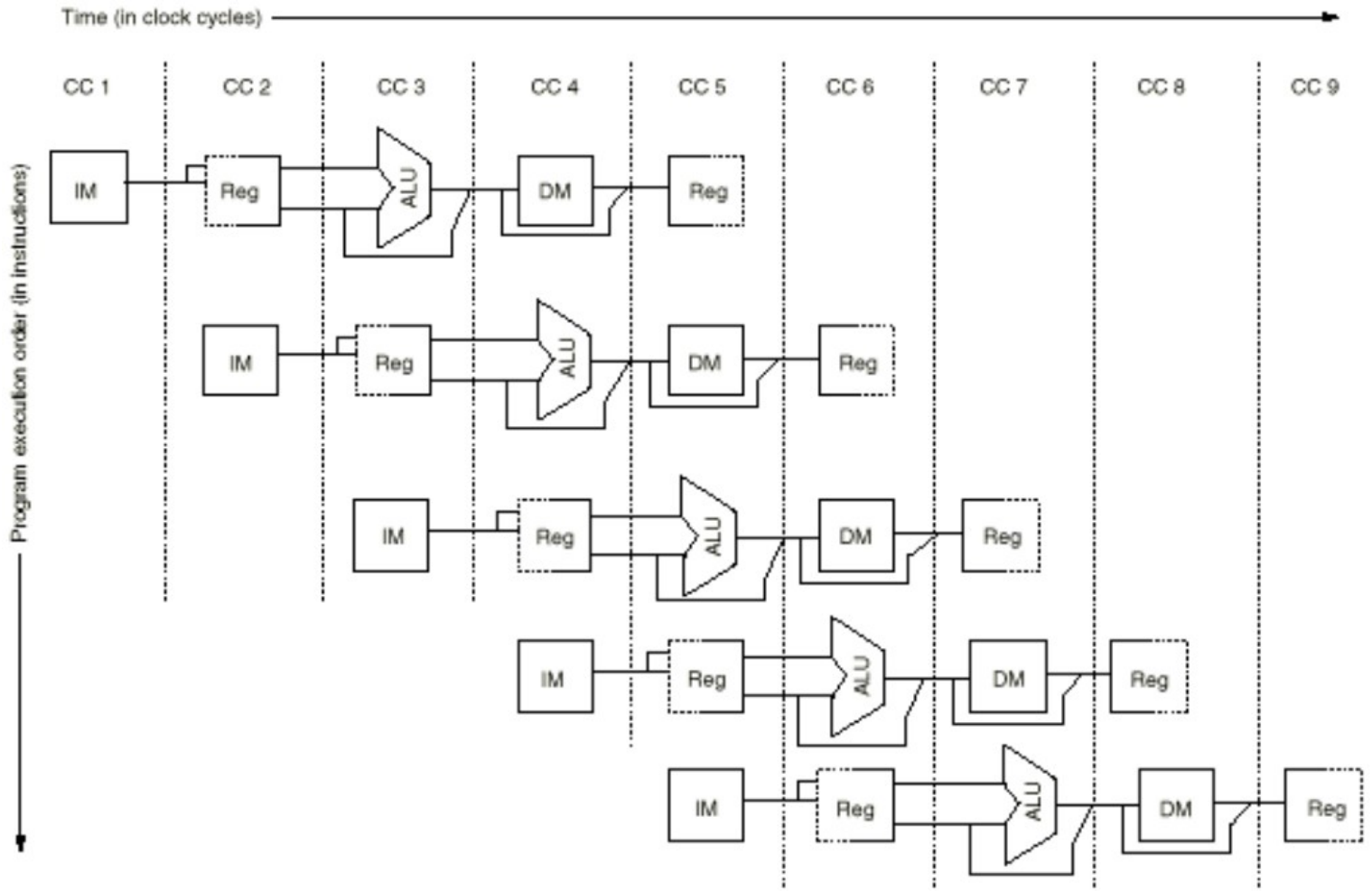


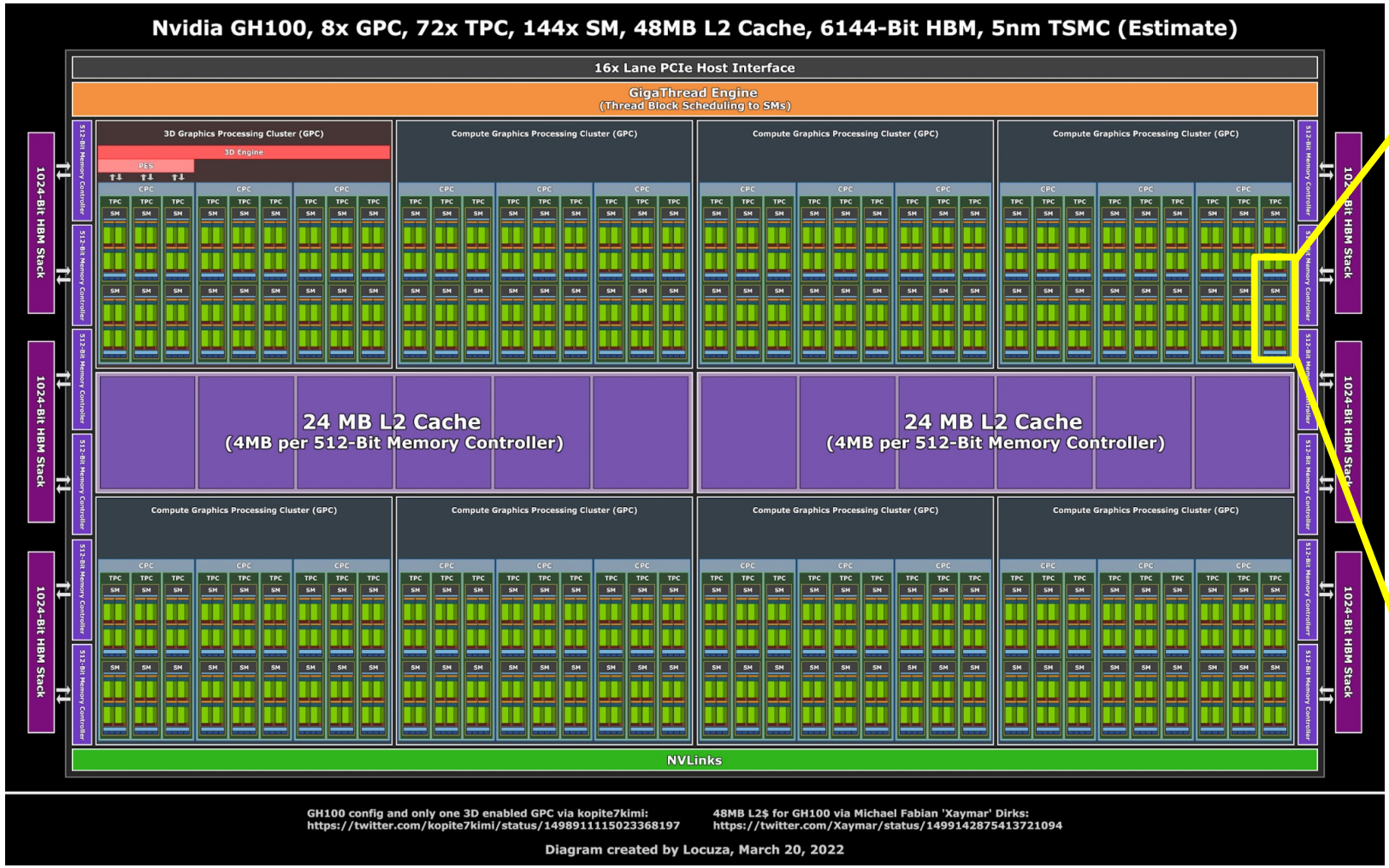
FIGURE 3.3 The pipeline can be thought of as a series of datapaths shifted in time.

CISC vs RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

https://csarassignment.files.wordpress.com/2013/05/risc-vs-cisc.png

GPUs

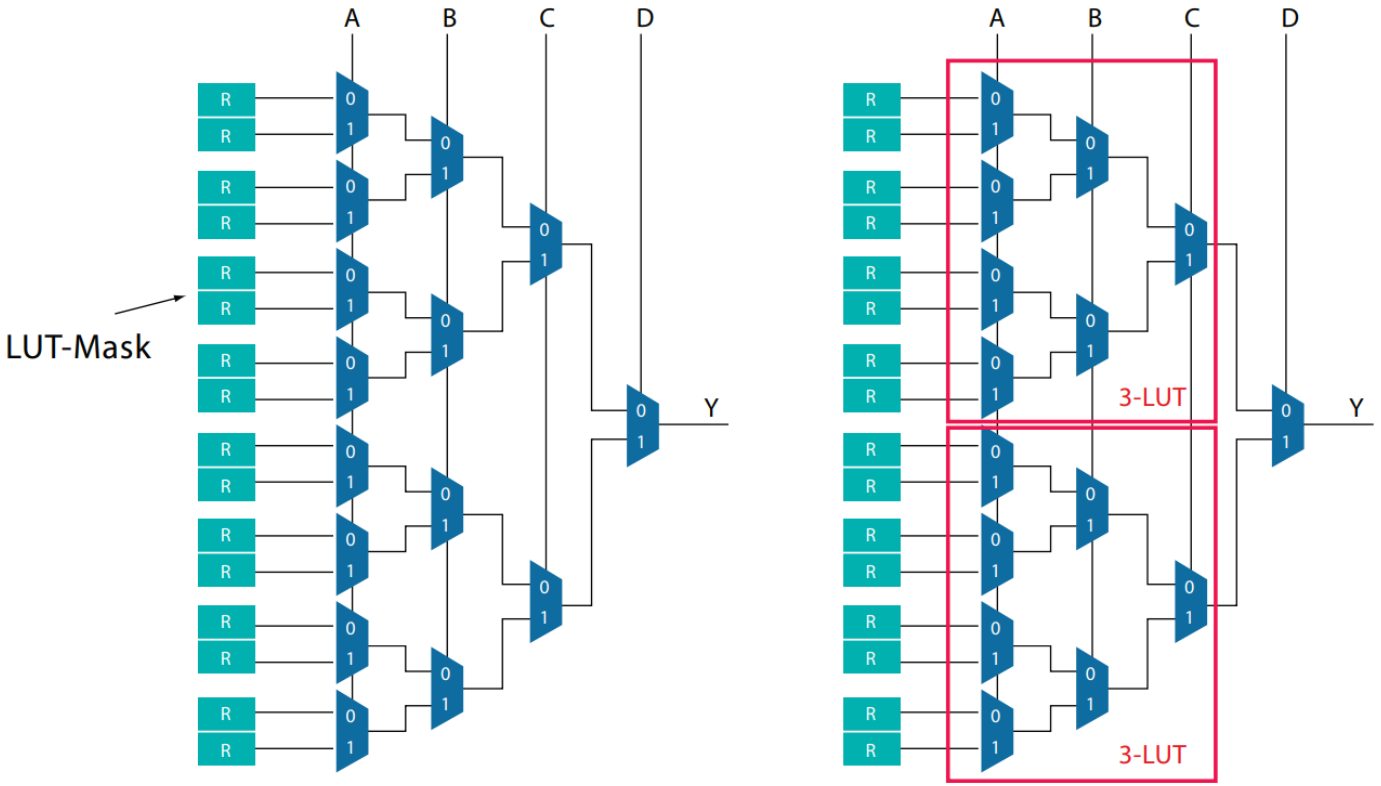


https://videocardz.com/newz/nvidia-gh100-hopper-gpu-comes-with-48mb-of-l2-cache-only-one-gpc-has-graphics-enabled

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

FPGAs

Figure 2. Building a LUT

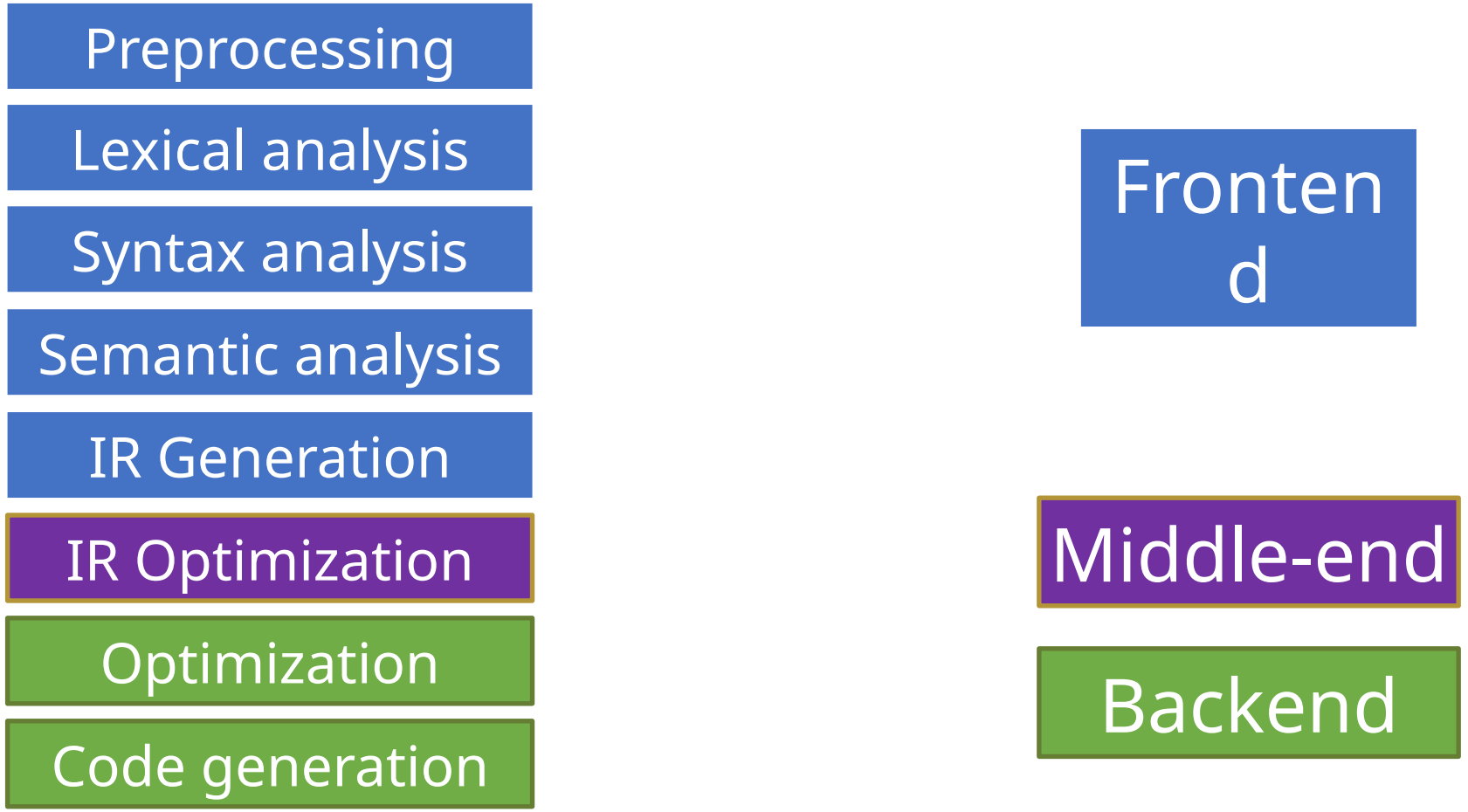


a'b'c'd' + abcd + abc'd' = 1000 0000 0000 1001 = 0x8009

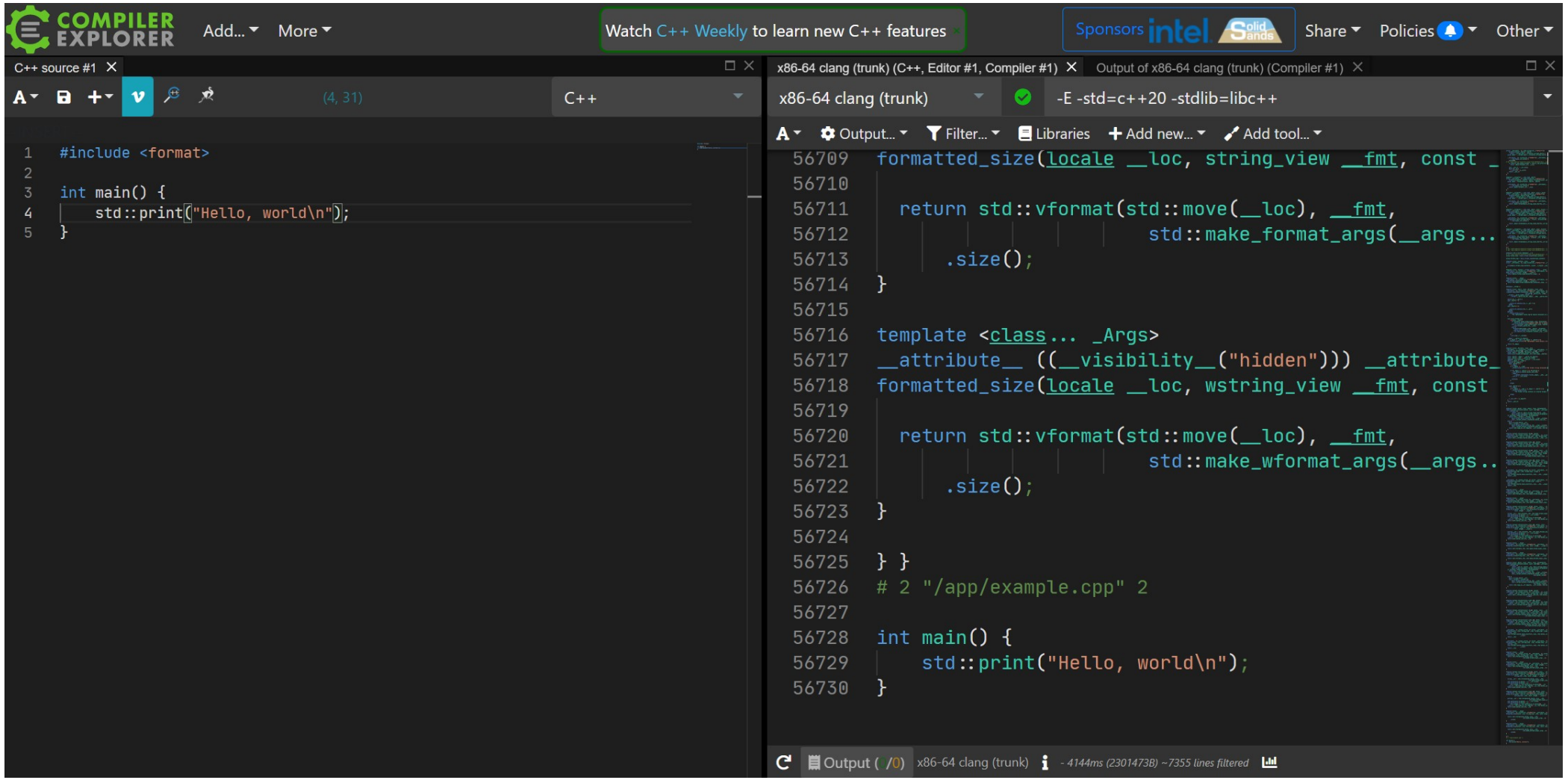
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf

Modern compilers

Compiler stages

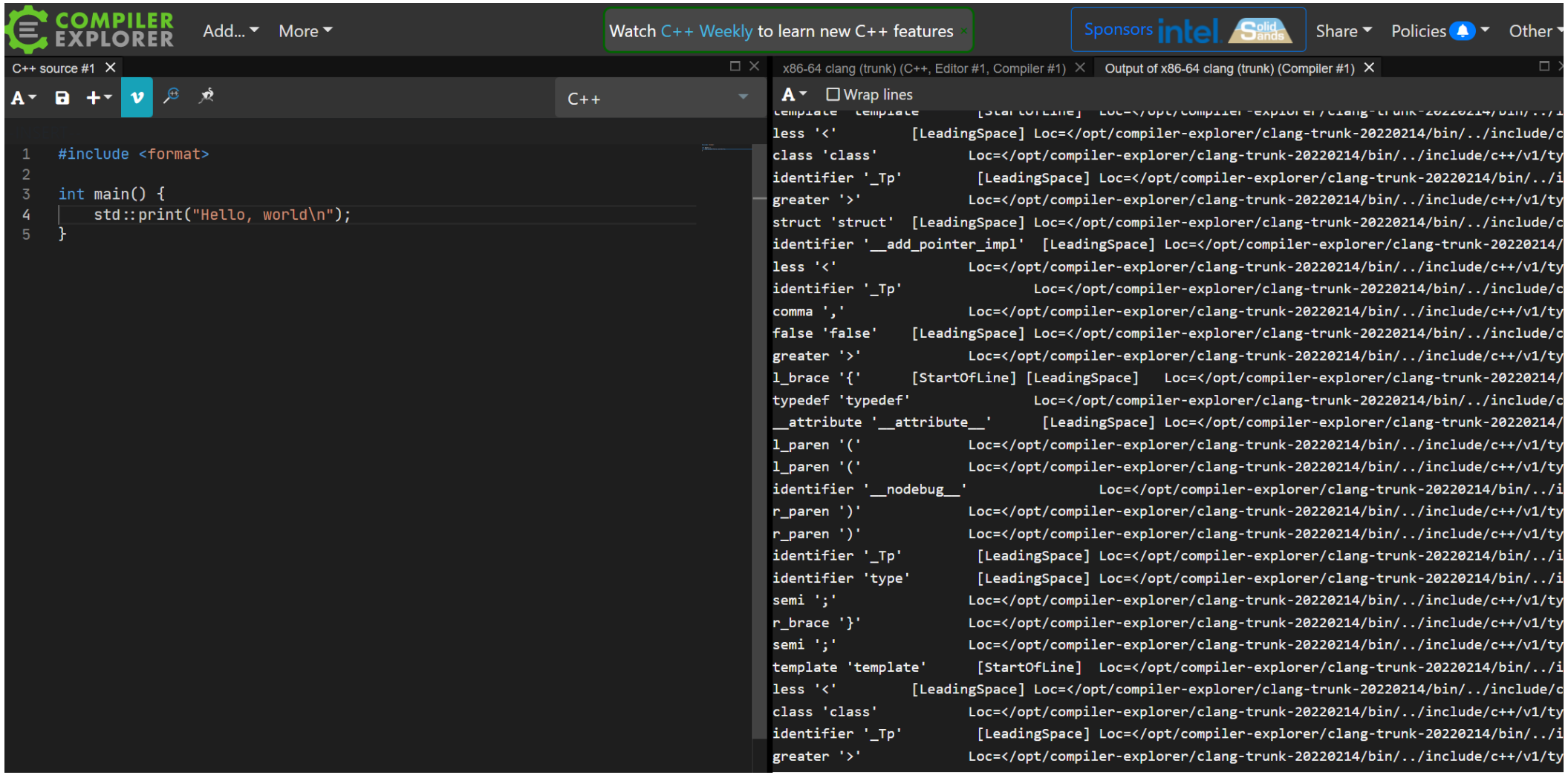


Preprocessing



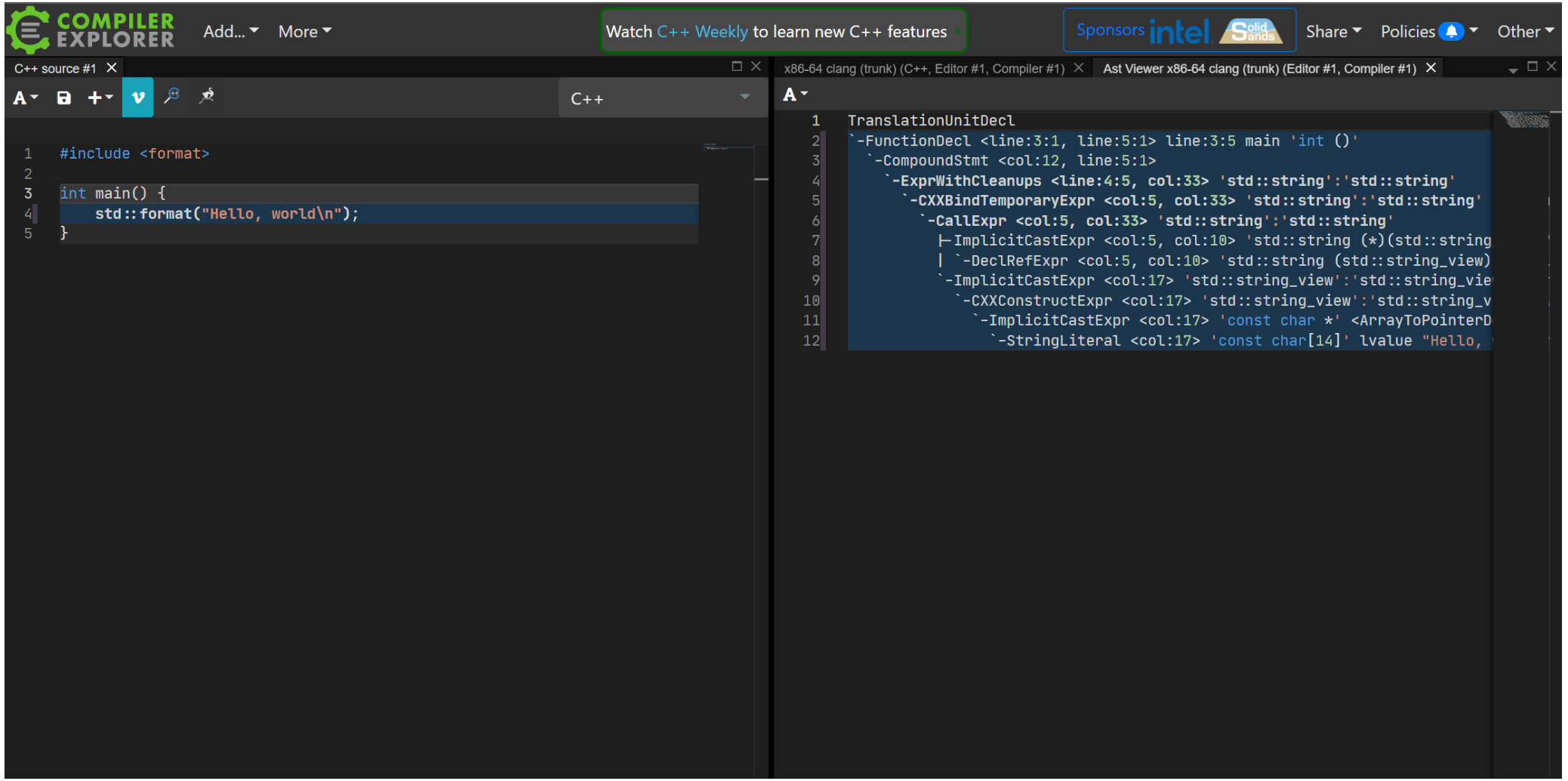
https://godbolt.org/z/o4K7TYr5E

Lexer



https://godbolt.org/z/T6ejdsEba

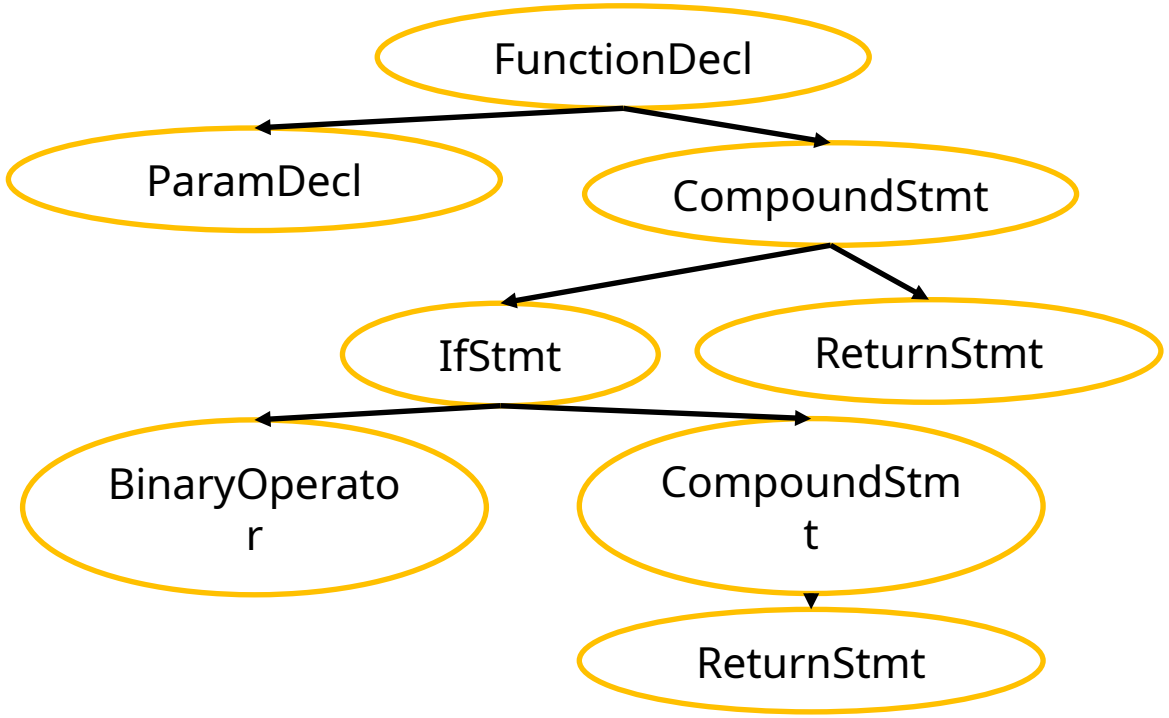
Syntax analysis



https://godbolt.org/z/cz3KjMcYb

Abstract Syntax Tree

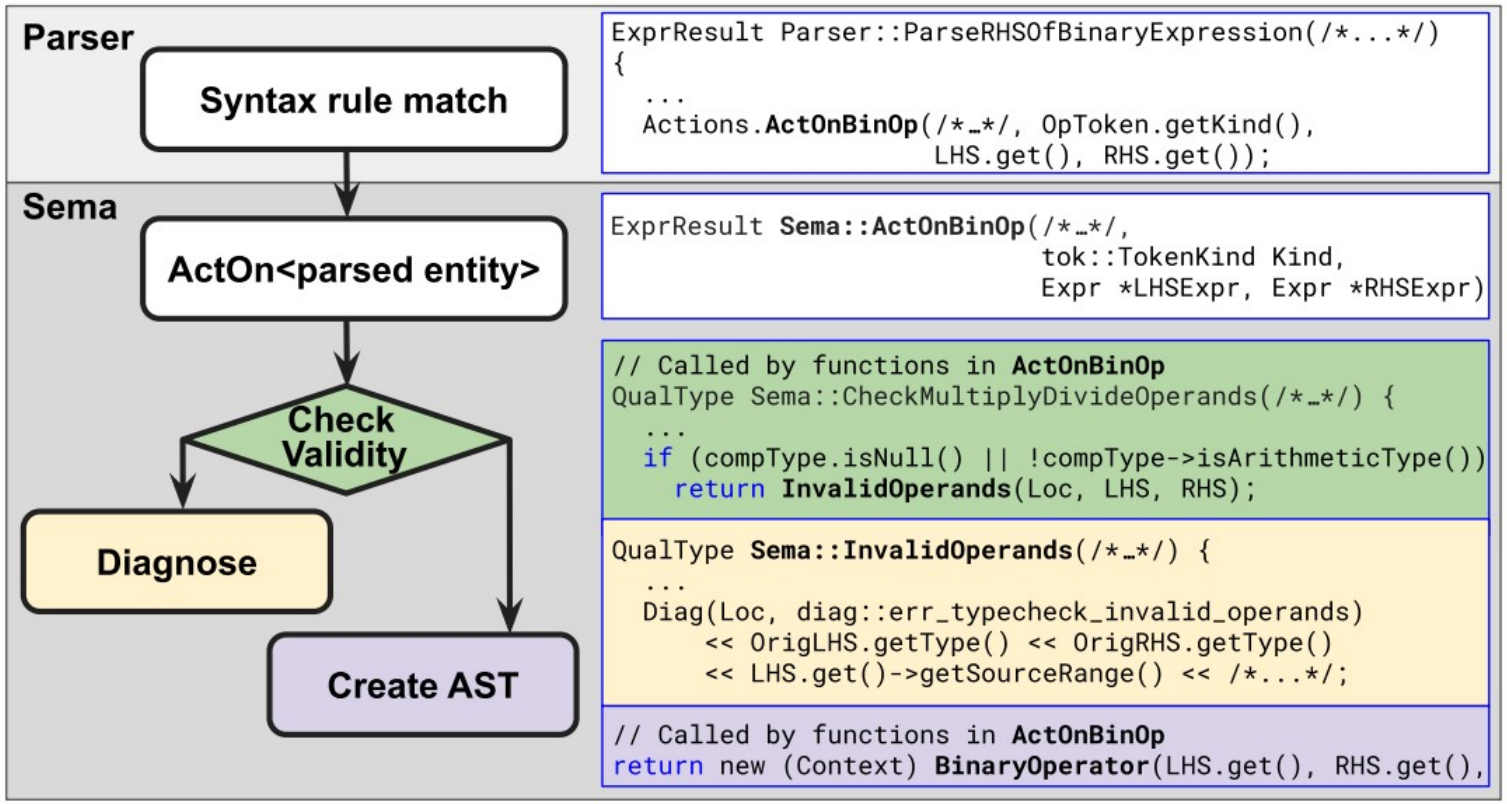
```
int foo(int i) {
    if (i < 10) {
        return i * 2;
    }
    return i;
}
```



https://godbolt.org/z/6Wq38MbPj

Semantic analysis

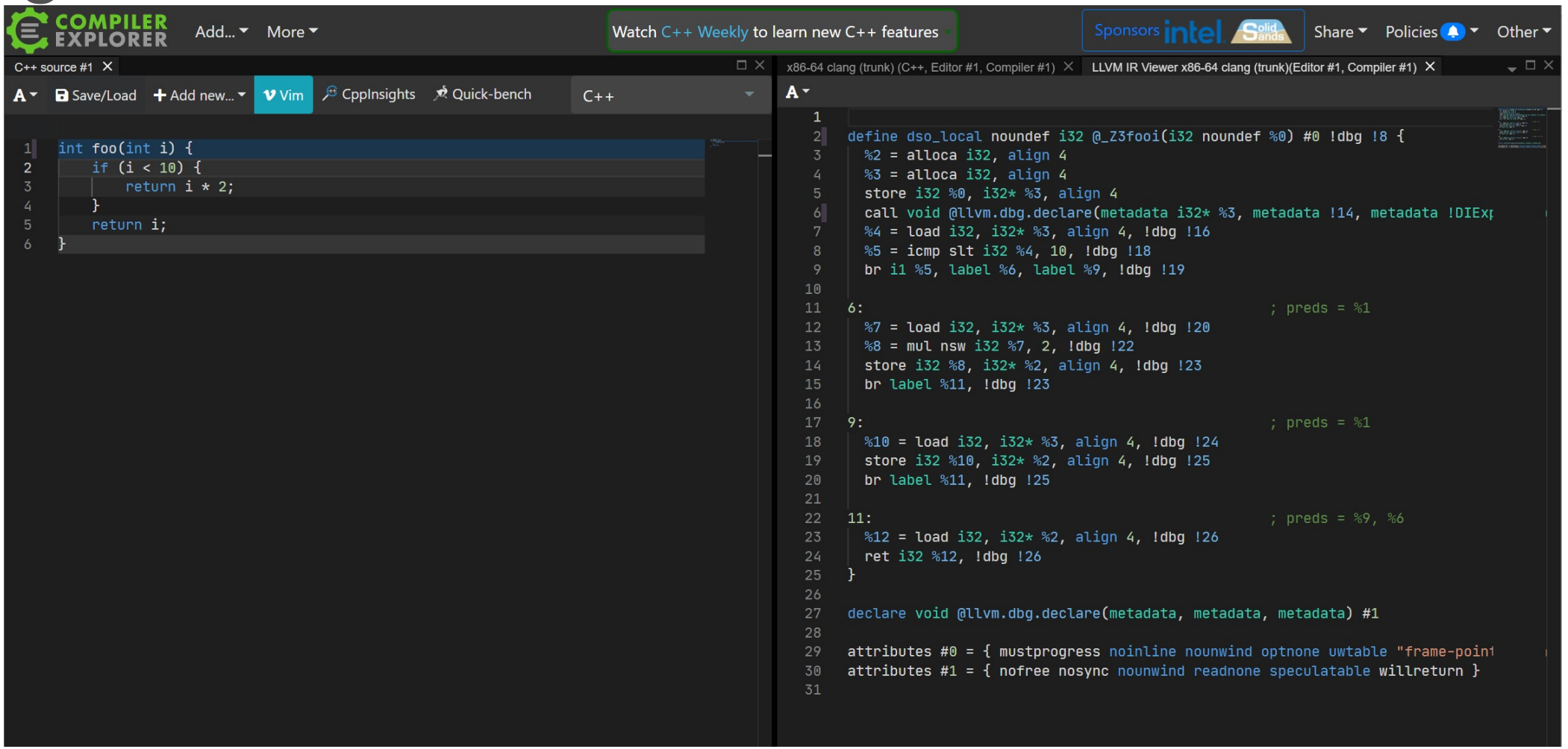
Sema Example



arm

https://llvm.org/devmtg/2019-10/slides/ClangTutorial-Stulova-vanHaastregt.pdf

IR generation



https://godbolt.org/z/or3z3oYnq

LLVM IR

LLVM IR is used for several key purposes within the LLVM compiler framework:

- Platform Independence
- Optimization
- Code Generation

C++

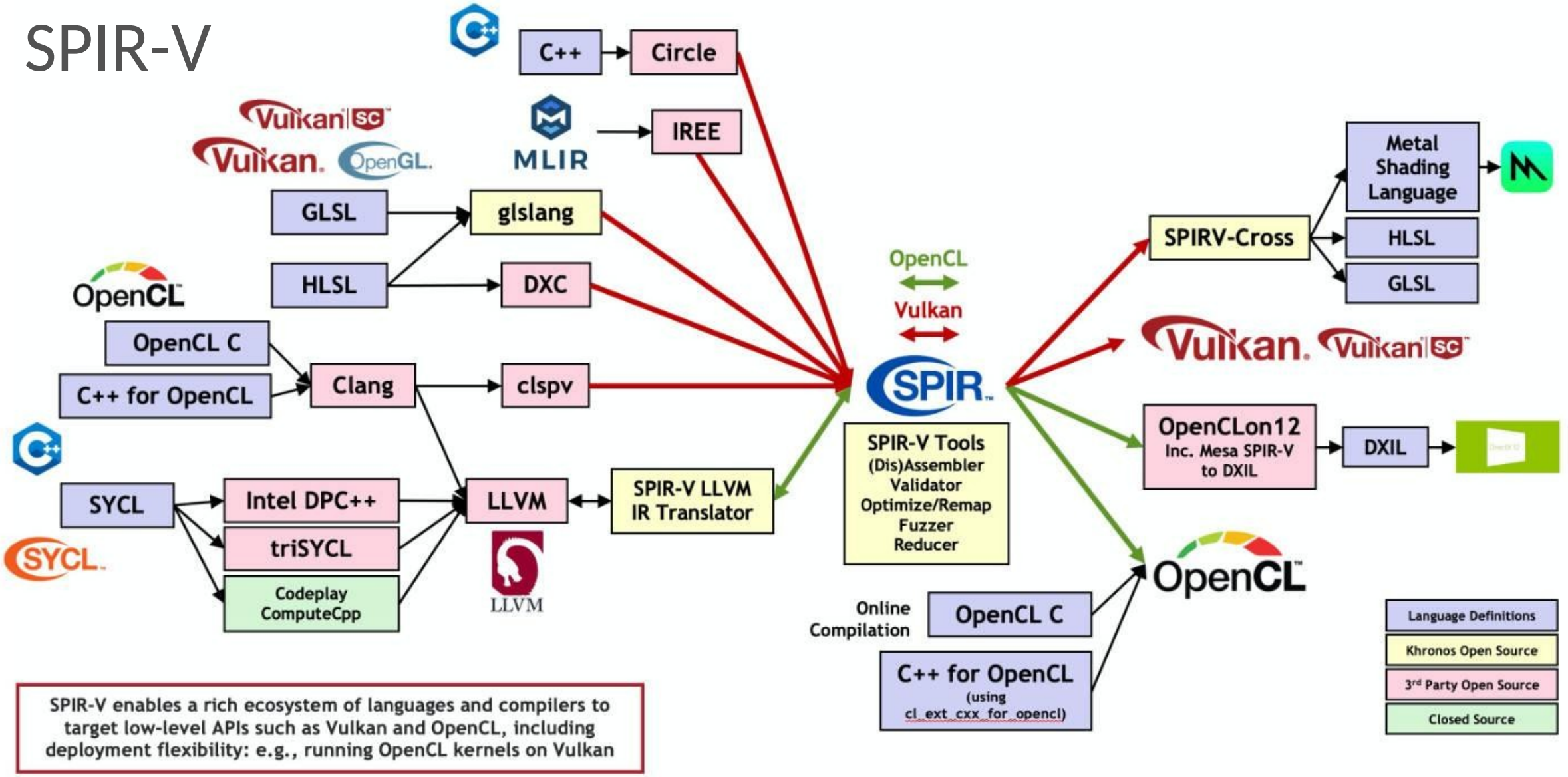
```cpp
int add(int a, int b) {
    return a + b;
}
```

The corresponding LLVM IR for this function might look something like this:

LLVM

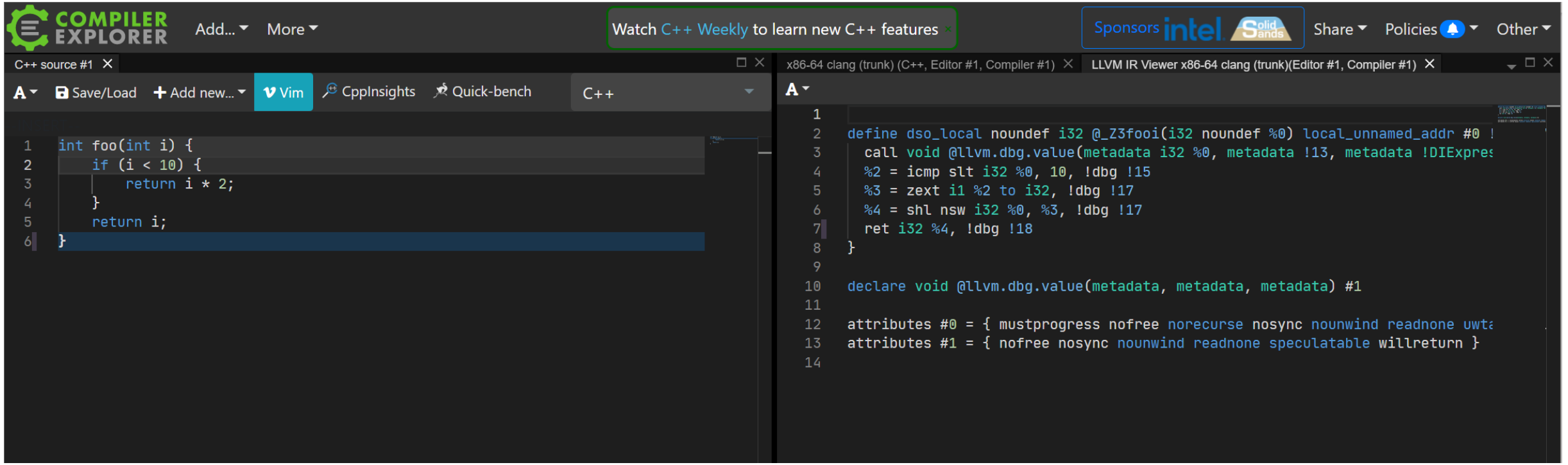
```
define i32 @add(i32 %a, i32 %b) {
entry:
  %0 = add i32 %a, %b
  ret i32 %0
}
```

SPIR-V



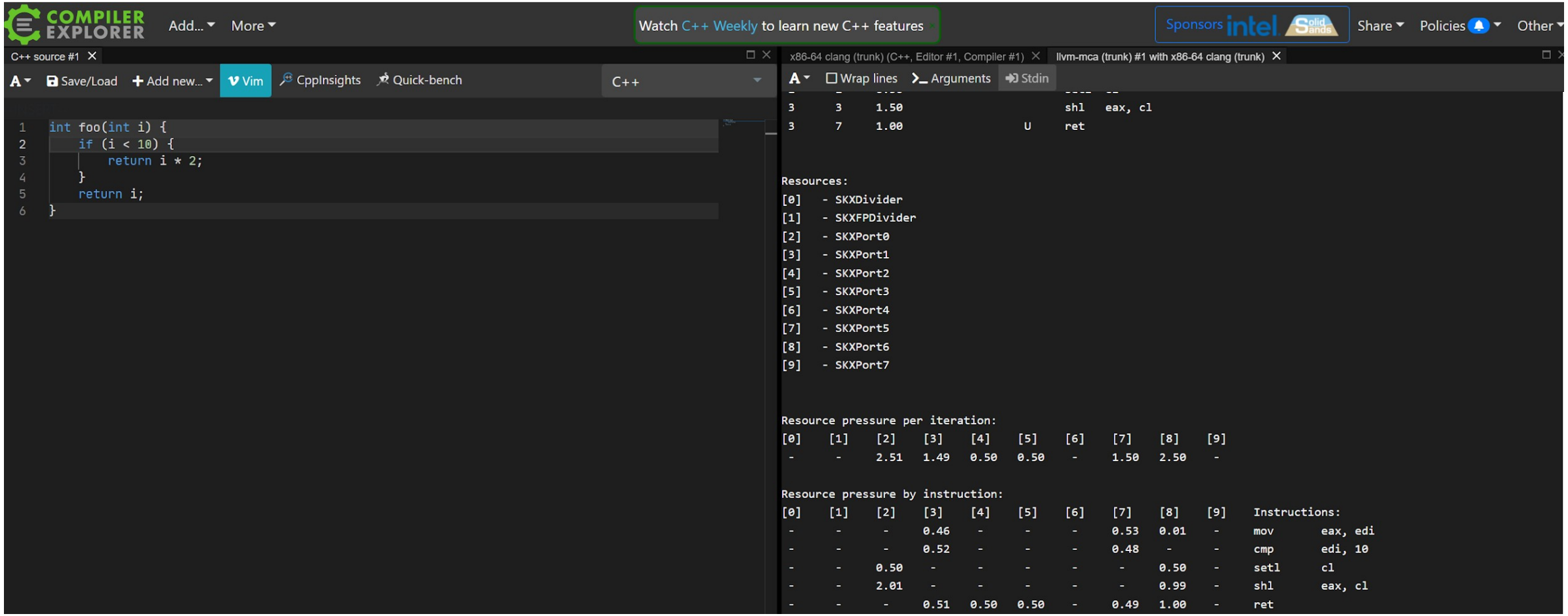
https://www.khronos.org/spir/

IR optimization



https://godbolt.org/z/YM6Wxno37

Code generation & optimization



https://godbolt.org/z/n3orv3Ydj

Linking

- Compiler operates on Translation Units
- Each translation unit produces an object file
- Linker assembles multiple object files and libraries into a program
- Not covered in this course
- A few pointers for those, who want to know more about linkers
  - How the C++ Linker Works – YouTube
  - 2017 LLVM Developers' Meeting: R. Ueyama "lld: A Fast, Simple, and Portable Linker" - YouTube

Debugging

- Compilers try to preserve source code <-> binary mapping
  - They’re not very successful when optimizations are involved
- Debug info is stored in a special format within a binary or in a separate file: lookup DWARF and PDB
- Debuggers get help from OS and hardware to enable breakpoints
- Will be covered in this course (stretch goal)
- More info:
  - 2016 EuroLLVM Developers' Meeting: D. Panickal & A. Warzynski "LLDB Tutorial: Adding debugger ..." – YouTube
  - 2013 LLVM Developers’ Meeting: “Adapting LLDB for your hardware: Remote Debugging the Hexagon DSP” - YouTube

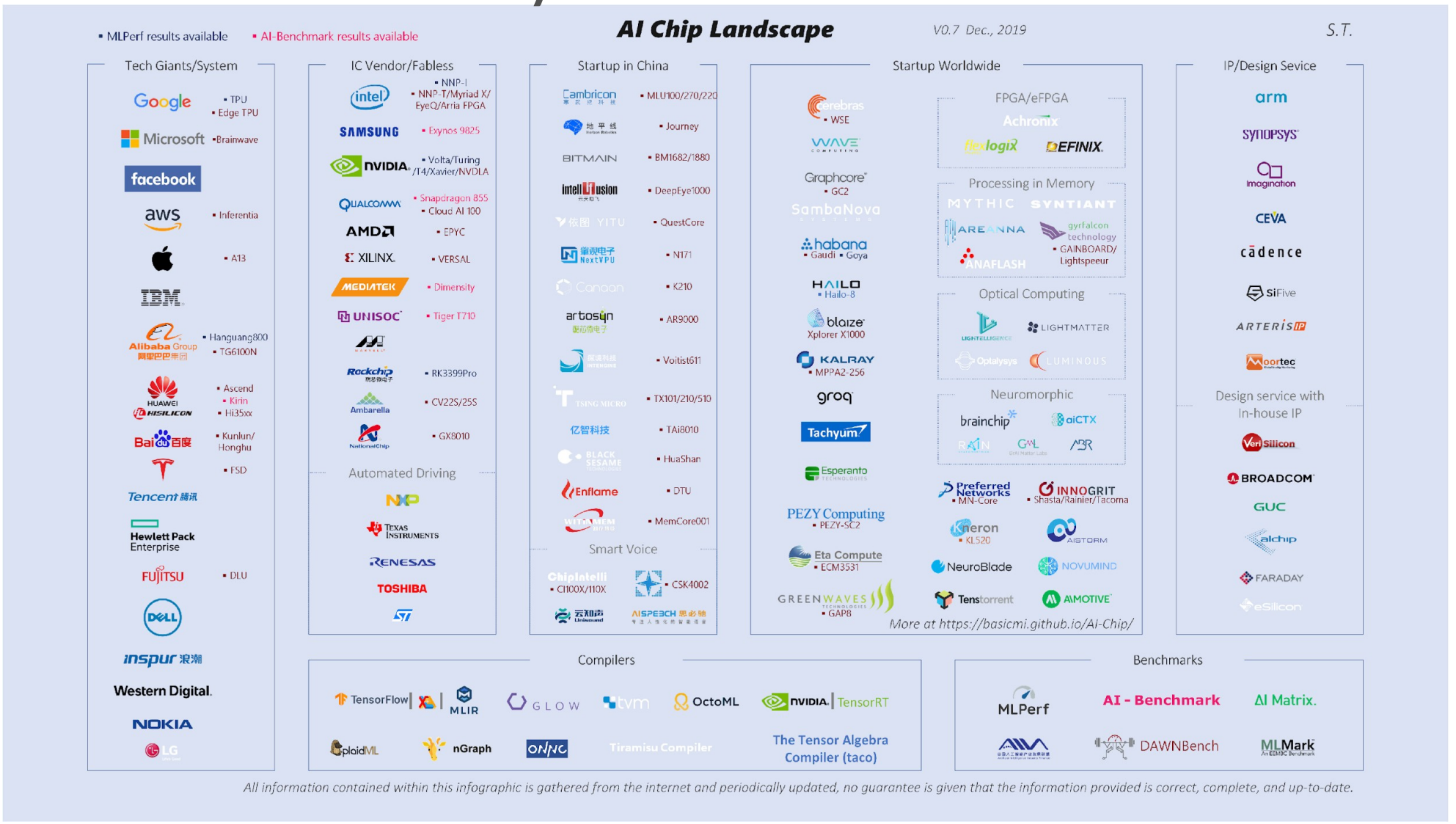
Static analysis

- Humans are not very good at computer programming
  - Lack of context, poor cognitive skills
- ASTs and IRs can be used to help developers find bugs
- Examples:
  - clang-tidy, PVS Studio, cppcheck, basically any IDE
  - More advanced: Rust borrow checker
- Will be covered in this course
- More info:
  - C++ Weekly - Ep 3 Intro to clang-tidy – YouTube
  - 2020 LLVM Developers' Meeting: "Using Clang-tidy for Customized Checkers and Large Scale Source..." - YouTube

LLVM

- Compiler framework
- Includes LLVM IR and backends for many platforms
- Started by Chris Lattner in 2001
- Multiple frontends under LLVM umbrella: clang, flang
- Photo: https://www.modular.ai/team

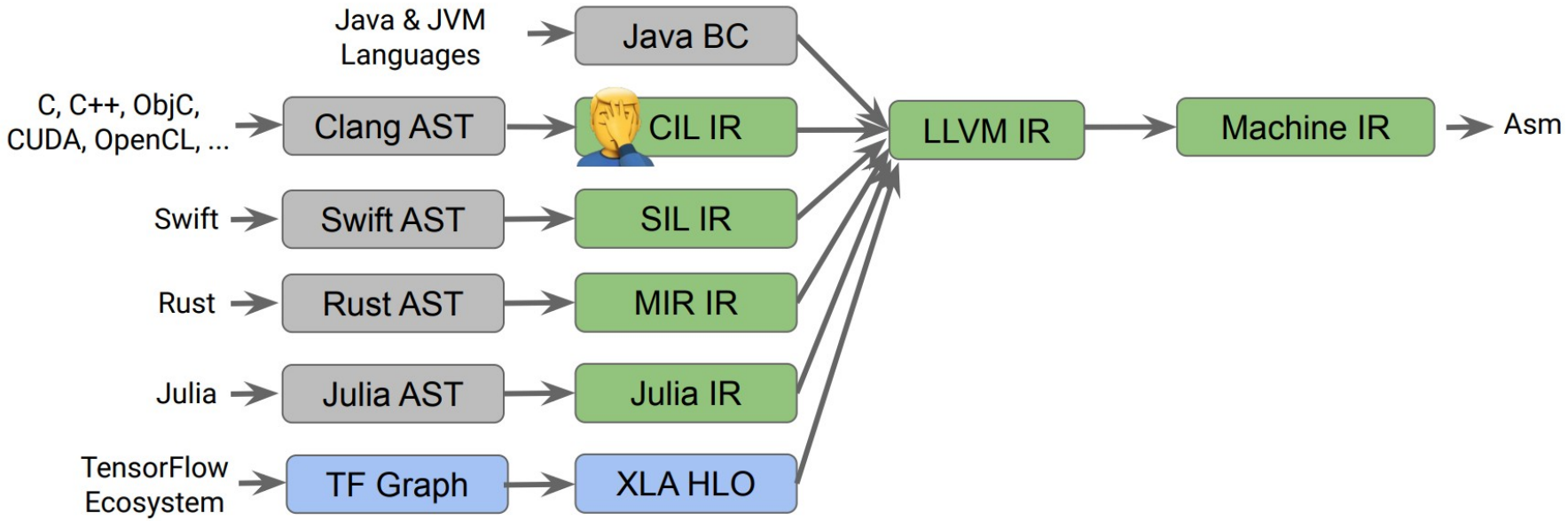
Hardware diversity



https://basicmi.github.io/AI-Chip/

Post-modern compilers

From Programming Languages to the TensorFlow Compiler



- Domain specific optimizations, progressive lowering
- Common LLVM platform for mid/low-level optimizing compilation in SSA form

https://llvm-hpc-2020-workshop.github.io/presentations/llvmhpc2020-amini.pdf

AI compilers

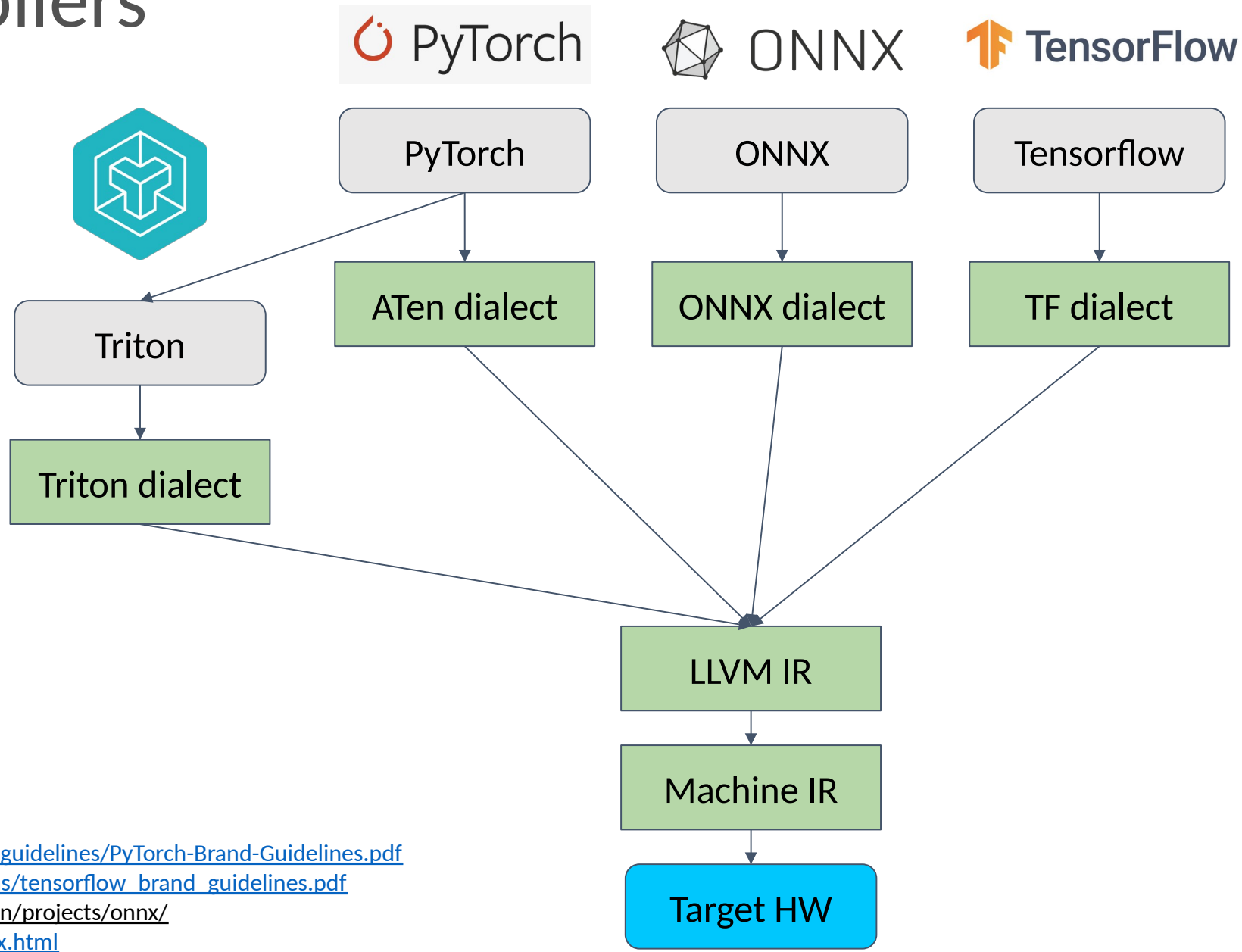
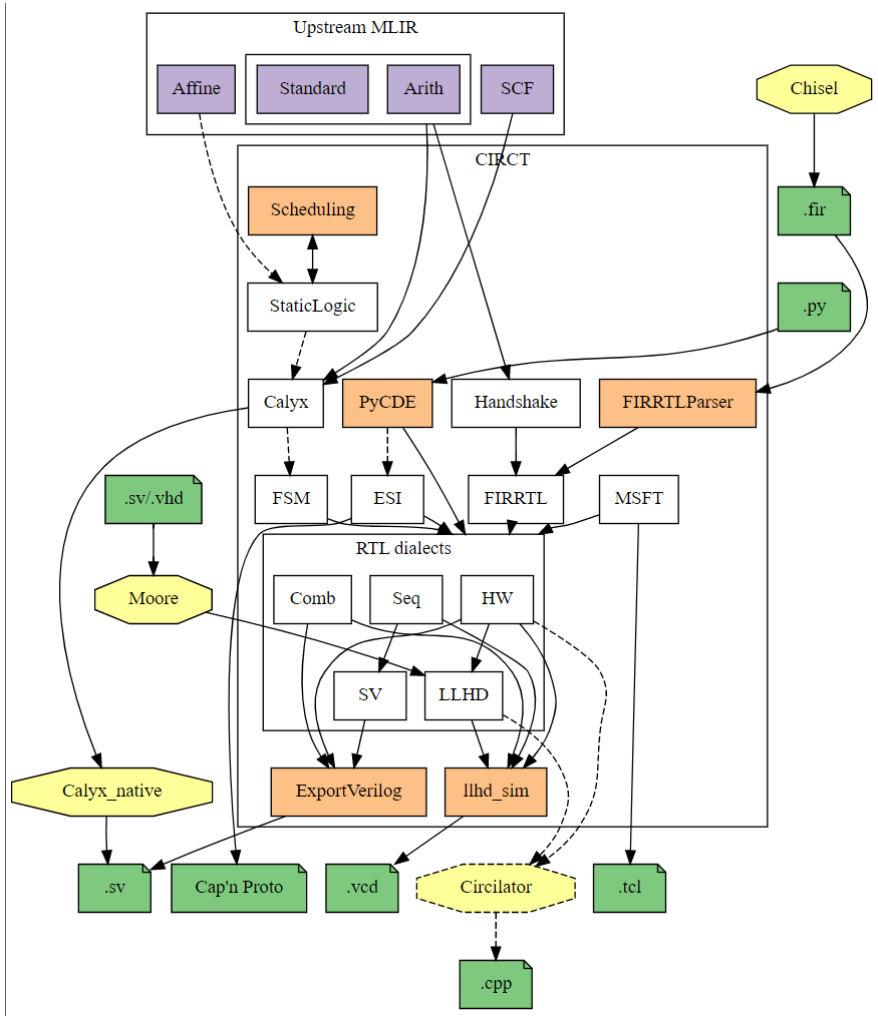


Photo:
https://pytorch.org/assets/brand-guidelines/PyTorch-Brand-Guidelines.pdf
https://www.tensorflow.org/extras/tensorflow_brand_guidelines.pdf
https://artwork.lfaidata.foundation/projects/onnx/
https://triton-lang.org/main/index.html

Customized hardware



- “CIRCT” / Circuit IR Compilers and Tools
- Compilers help design custom hardware
- Reduce cost of development and validation
- https://circt.llvm.org/

Career paths

- hardware vendors
- large software companies
- AI development

In the next episode…

- Compiler frontends
  - Different frontend stages: lexers, syntax, sema
  - Clang as an example

Extra materials

- Interview with Jim Keller
  - Part 1: https://www.youtube.com/watch?v=Nb2tebYAaOA
  - Part 2: https://www.youtube.com/watch?v=G4hL5Om4IJ4
- Interview with Chris Lattner
  - Part 1: https://www.youtube.com/watch?v=yCd3CzGSte8
  - Part 2: https://www.youtube.com/watch?v=nWTvXbQHwWs
- An overview of Clang, LLVM Dev 2019 - https://www.youtube.com/watch?v=5kkMpJpIGYU
- Compilers: Principles, Techniques, and Tools (aka the dragon book) - https://suif.stanford.edu/dragonbook/
- What Has My Compiler Done for Me Lately? (Matt Godbolt) - https://www.youtube.com/watch?v=bSkpMdDe4g4
- Engineering a Compiler (Keith D. Cooper & Linda Torczon) - https://github.com/lighthousand/books/blob/master/Engineering%20A%20Compiler%202nd%20Edition%20by%20Cooper%20and%20Torczon.pdf

- Chandler Carruth
  - Tuning C++: Benchmarks, and CPUs, and Compilers! Oh My! - https://www.youtube.com/watch?v=nXaxk27zwlk
  - High Performance Code 201: Hybrid Data Structures - https://www.youtube.com/watch?v=vElZc6zSIXM