codeplay®

Enabling AI & HPC to be open, safe and accessible to all

Building the Foundations for the Next Generation of High-Performance Software

February 2023

Andrew Richards, CEO, Codeplay
Multicore World, Wellington, February 2023

Codeplay

- A compiler company, founded in 2002
- Based in Edinburgh, Scotland
- Specialise in compilers for *accelerator processors* (e.g. GPUs)
- Acquired by Intel in 2022 and continuing as a subsidiary

Who am I?

- I was a videogame developer
- In the early days of videogames (8&16-bit), we were using processors from much older business systems
- We had years to squeeze the most out of older processors

What changed?

- In the 90s, people discovered you could write a game on a PC, doing new things never before possible, which would run super-slow and be unplayable
- But, by the time your game was released, there were new processors that would run it fast enough
- People could plug in new devices like GPUs to their PC
- By the end of the 90s, videogames and PCs were the driving force of innovation of the tech industry

Why did I start Codeplay?

I decided:

- I wanted to be part of driving new ideas in tech
- Writing software on today's hardware and running it on tomorrow's hardware was the way to do this
- This requires *compilers*

What is a compiler?

- What does it do?
  - A compiler takes computer source code (written by a programmer) and turns it into instructions that a processor can execute
- What is a compiler for?
  - A compiler enables programmers to write code that does what the user wants

➢ What a compiler does is the *opposite* of what it's for

Enabling people to build software: the *"ecosystem"*

- Much larger teams of people with variety of skills
- Showing people how to write source code: training
- Working out how to target a variety of different machines
- Integrating with other software
- Standardizing the language to enable people to write once & use many compilers
- Working out the intellectual property rules with lawyers
- *If we don't do this work, then our compiler doesn't serve its purpose*

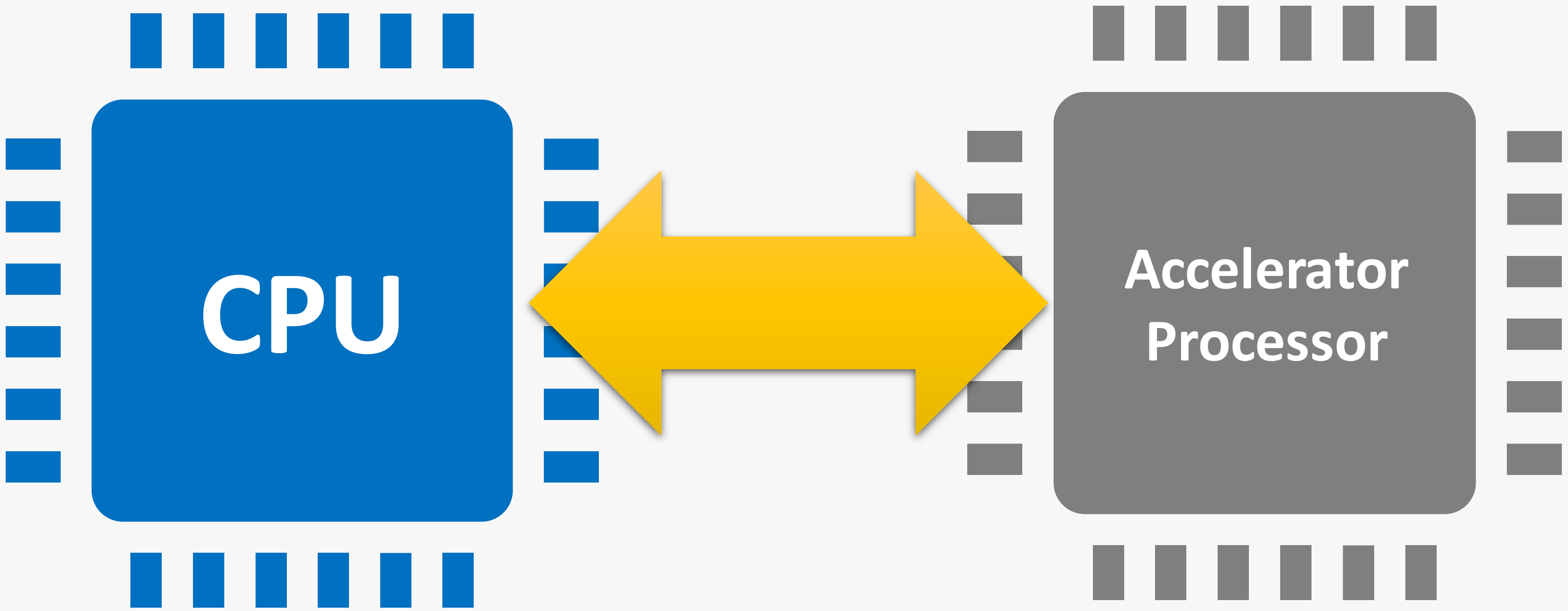
Building compilers

- Small team of specialist compiler developers
- Convert source code to machine code

What is an accelerator processor?

- Early computers did everything with a CPU
  - My ZX81 had to share time between running my program and driving the TV
- Then computers added special hardware to do very specific things like graphics and sound
  - Early videogames had “Sprites” and “Scrolling backgrounds”
  - If you wanted to do anything else you had to use the CPU: this was limiting
- But by the early 2000s, the graphics processors (GPUs) had become fully-programmable
  - This enable videogames designers to do incredible new things in graphics

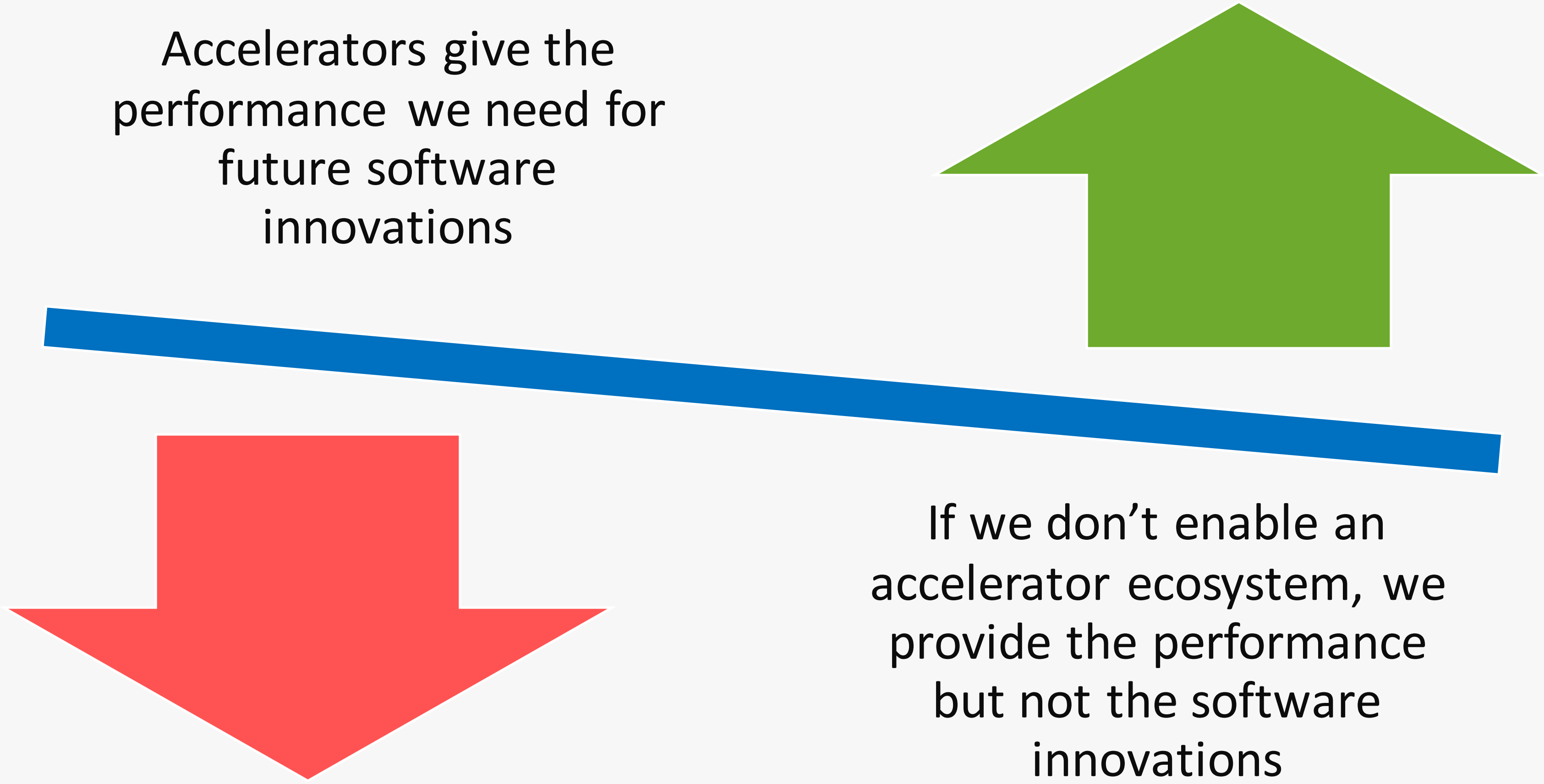
CPUs and Accelerators



CPUs to GPUs to more general accelerators

CPUs	GPUs	Other Processors
• Great at latency-sensitive tasks (reacting quickly) • Huge, well-developed ecosystem	• Great at graphics • Great at processing lots of floating-point data • Huge, well-developed ecosystem for graphics • Single-vendor ecosystem for non-graphics data-intensive applications	• Great at specific tasks • Much longer time-to-market, • No open ecosystem for software • Struggled to keep up with new innovations (e.g. AI)

The Opportunity vs the Challenge



Accelerators give the performance we need for future software innovations

If we don’t enable an accelerator ecosystem, we provide the performance but not the software innovations

The challenge today: Closed systems

- Block innovations in AI by making it very hard to design new operations
- Very hard to integrate with more complex software
- Low customer adoption

Closed AI graph compilers

- Block innovations in accelerator hardware
- Only works for software that maps well to specific hardware
- Hard to move software between processors to get best performance

Single-device programming models

- Blocks progress to accelerator-based systems
- Lack of performance means lack of software progress

CPU-only programming models

The solutions

- We know the technology answers
  - There isn’t one solution for everyone, so we need to integrate the best-in-class technology solutions into one ecosystem-ready integrated solution
- We know the organizational answers
  - We know that industry standards and open-source projects can deliver ecosystem-friendly, innovator-friendly platforms for innovation

Platforms: long-term compatibility

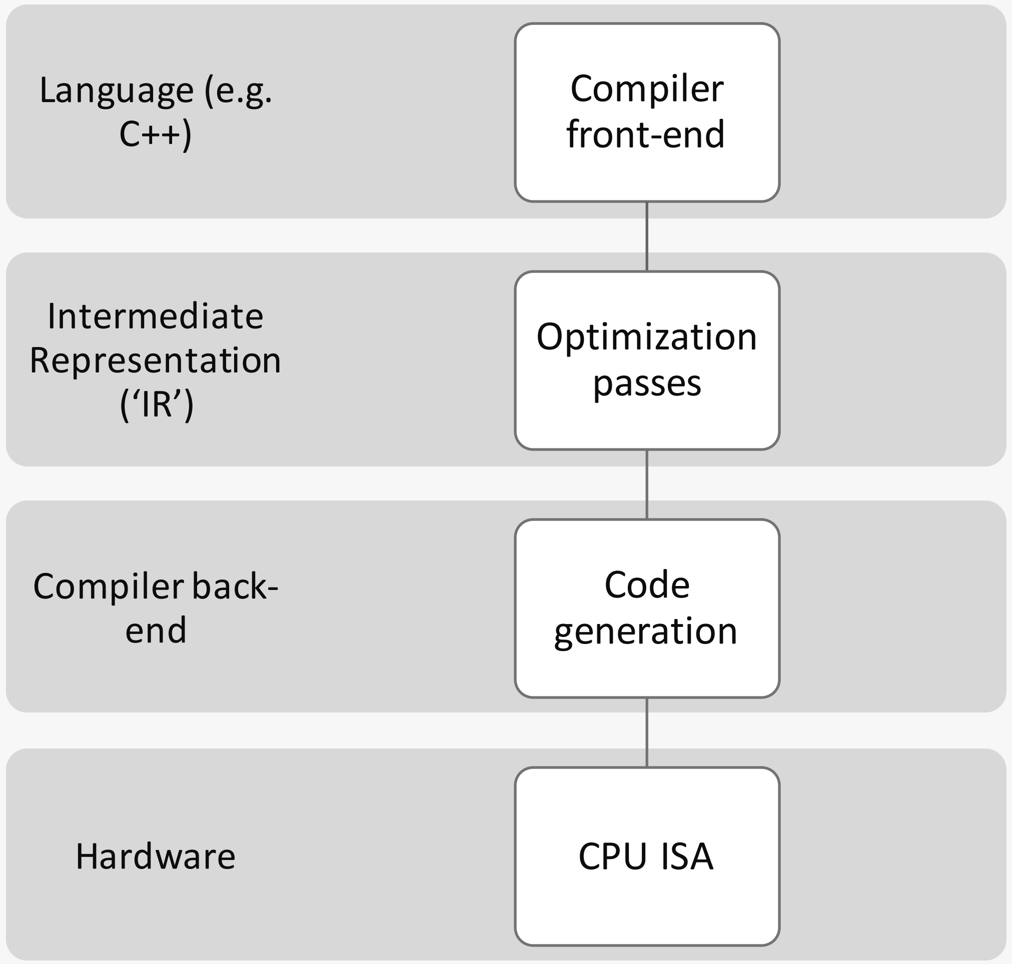
- My laptop stores its data on the C drive
  - The A and B drives disappeared multiple decades ago
- Success for hardware ecosystems comes over decades
  - Software takes a long time to develop
- oneAPI, SYCL, SPIR-V are designed for very long term support
  - Long term backwards compatibility

<u>We are building the software approach for decades to come</u>

Best-in-class components

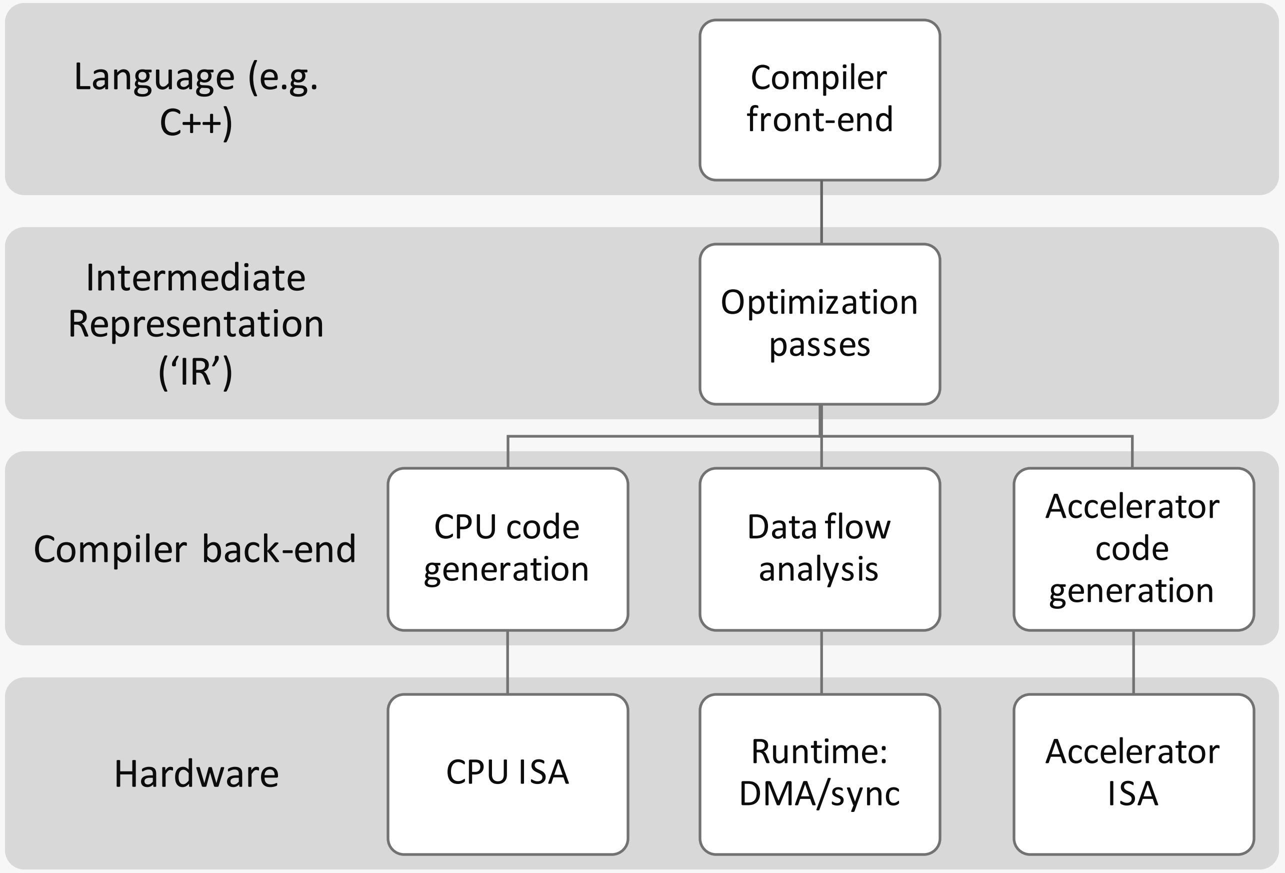
- For each component we need, we are striving to choose the best-in-class industry standard approaches

How CPU compilers work



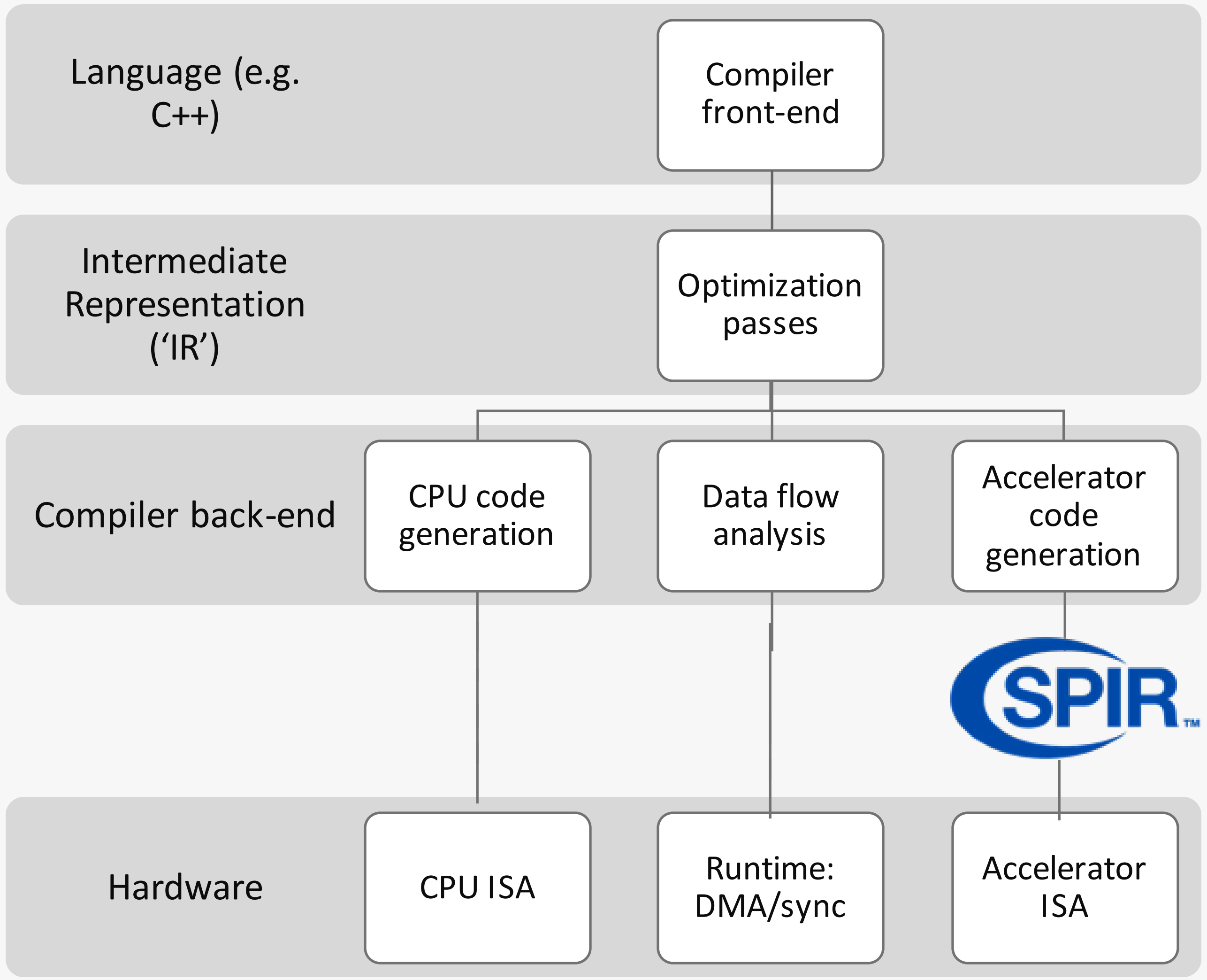
- We transform a language into an intermediate representation which contains a simplified representation of our code
- We do this because it’s much easier to *transform* an IR with *passes*

How *heterogeneous* compilers work



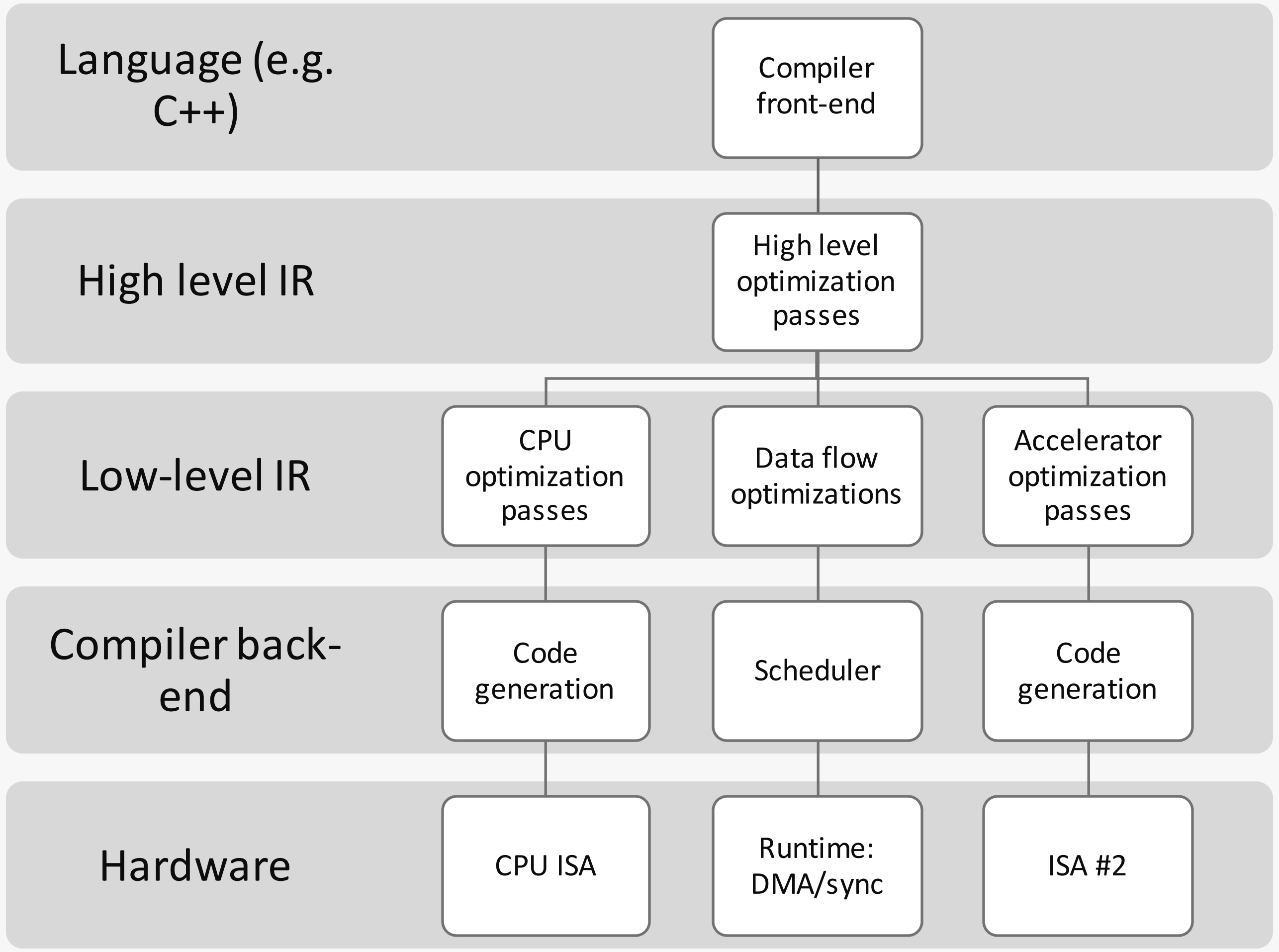
- We now need to create code for 2 (or more) processors
  - 2+ compiler back-ends
- And we also need to transfer data and synchronize
  - We have a *runtime*

We interface to hardware ISA via SPIR-V



- SPIR-V is the standard *intermediate representation* to interface the accelerator-specific ISA from the higher-level compiler stack

Multi-Level Intermediate Representation (MLIR)



- MLIR lets us do different optimizations at different levels
- Enables optimizations for different hardware
- We're adding MLIR support to enable a wider range of compiler optimizations in the stack

The C++ approach: SYCL

- SYCL is a royalty-free vendor-neutral industry standard C++ for parallel software and accelerator processors
- **SYCL takes proven C++ performance ideas & super-charges them for a heterogeneous processing world**
- Now we can:
  - Build our own C++ SYCL compilers for a variety of new processors
  - We can design our own optimizations
  - We can build C++ libraries that can adapt to the performance requirements of lots of different systems
  - We can integrate native compilation for different processors in one source file

How SYCL handles parallelism

```
cgh.parallel_for<class parallel_demo> (
                cl::sycl::range<1>(n),
                [=](cl::sycl::item<1> i)
{
    out [i] = f (in [i]);
});
```

For more complex parallelism where there are scheduling dependencies, there are a range of options: SYCL requires you to specify where your code *isn't parallel*

- By default, a SYCL `parallel_for` can run entirely parallel
- We define a range to execute in parallel over
- We use a C++ lambda to define the loop body as that's standard now
- It is the job of the programmer to ensure 'f' is safe to run in parallel
- The loop is enqueued and run asynchronously to the CPU thread
- The parallel loop can execute on any SYCL supported core: CPU, GPU, FPGA, DSP, anything programmable

How SYCL handles data access

```
auto in = inp.get_access<cl::sycl::access::mode::read>(cgh);
auto out = outp.get_access<cl::sycl::access::mode::read_write>(cgh);
cgh.parallel_for<class parallel_demo> (
                cl::sycl::range<1>(n),
                [=](cl::sycl::item<1> i)
{
    out [i] = f (in [i]);
});
```

$\}$ *Access mode specified*

Performance on accelerators is more about data access than compute:

- GPUs have on-board HBM memory and a small amount of fast on-chip SRAM
- DSPs use DMA to transfer data rapidly to a larger amount of on-chip SRAM
- AI accelerators usually have a lot of fast on-chip SRAM

SYCL requires developer specify how to access data: which may enable maximum performance

How SYCL handles multiple, different, processors

```
gpu_queue.submit([&](cl::sycl::handler &cgh) {
    auto in = inp.get_access<cl::sycl::access::mode::read>(cgh);
    auto out = outp.get_access<cl::sycl::access::mode::read_write>(cgh);
    cgh.parallel_for<class parallel_demo> (
                cl::sycl::range<1>(n),
                [=](cl::sycl::item<1> i)
    {
        out [i] = f (in [i]);
    });
});
```

Compiled for CPU by any normal CPU C++ compiler & runs asynchronously on host CPU to enqueue kernels to accelerator

SYCL Device Compiler extracts this kernel and compiles it natively for accelerator processors

This kernel 'name' allows multiple C++ compilers to be stitched together

- Both host & device code are compiled via C++ native compilers
- When SYCL goes through OpenCL, it can (optionally) use SPIR-V as the compiler IR
    - ➢ But it's still C++ source compiled to native device ISA
- SYCL device compilers can have per-device extensions
- More than one device compiler can compile a single source file

Combines the benefits of chosen CPU compiler *and* chosen device compiler

How SYCL handles processor-specific optimizations

- Most vector instructions and memory models map to SYCL2020 today
- New instructions or memory systems can be mapped to SYCL extensions – there's a clear mechanism for this

- Then, these processor-specific performance features are *integrated into the template libraries* in an appropriate place
  - ➢ The aim is to enable processor-specific optimizations in the least disruptive way possible
  - ➢ Enables us to run the same software with high performance on lots of different processors

Building it

- SYCL started as a Codeplay project
  - We worked with collaborators to build out an ecosystem for C++ on accelerators
  - SYCL is now led by all the people who join
  - SYCL became the product of experts in their field coming together
- oneAPI started as an Intel project
  - We will work with collaborators to build out an even wider ecosystem: more languages, more libraries, more tools
  - oneAPI will now be led by all the people who join
  - oneAPI will now become an open-source project
  - We will bring together hardware and software innovators to build out a truly open industry-wide ecosystem for accelerator hardware and software

Building it together

- In 2023 we are moving oneAPI into a new open organization
- oneAPI is being structured as a constantly-evolving open-source project
- We enable a wide range of hardware and that will grow
- We have proven it on multiple GPUs as well as FPGAs
- We enable a wide range of software and languages: that will grow too
- We are enabling a rapidly-growing range of HPC & AI frameworks

Come join us

- You can join oneAPI and help shape our future
- You can checkout the oneAPI source code
- You can download pre-built binaries of oneAPI for Intel, NVIDIA and AMD GPUs
- You can bring your own hardware and software

- https://www.oneapi.io/community/

codeplay®

Enabling AI & HPC to be open, safe and accessible to all