codeplay®
THE HETEROGENEOUS SYSTEMS EXPERTS

OpenCL™ and SYCL™ Open Standards for ADAS Vision Processing and Machine Learning

Charles Macfarlane, VP Product Marketing

How do we deliver embedded intelligence?

How do we get from here…

Level 0
- Warnings

Level 1
- Adaptive
- Assist

Level 2
- Execute
- Automated manoeuvres

Level 3
- Limited overall control

Level 4
- Deep self control
- All conditions

Level 5
- Autonomous
- Stages from very local to extensive journeys

… to here?

These are the *SAE levels* for autonomous vehicles. Similar challenges apply in other embedded intelligence industries

We have a mountain to climb

- How do we get to the top?
- When we don't know what the top looks like...
- ... and we want to get there in safe, manageable, affordable steps...
- ... without getting lost on our own...
- ... or climbing the wrong mountain

This presentation will focus on:

- The targets hardware and software platforms that will be able to deliver the results

- The open standards that will enable solutions to interoperate

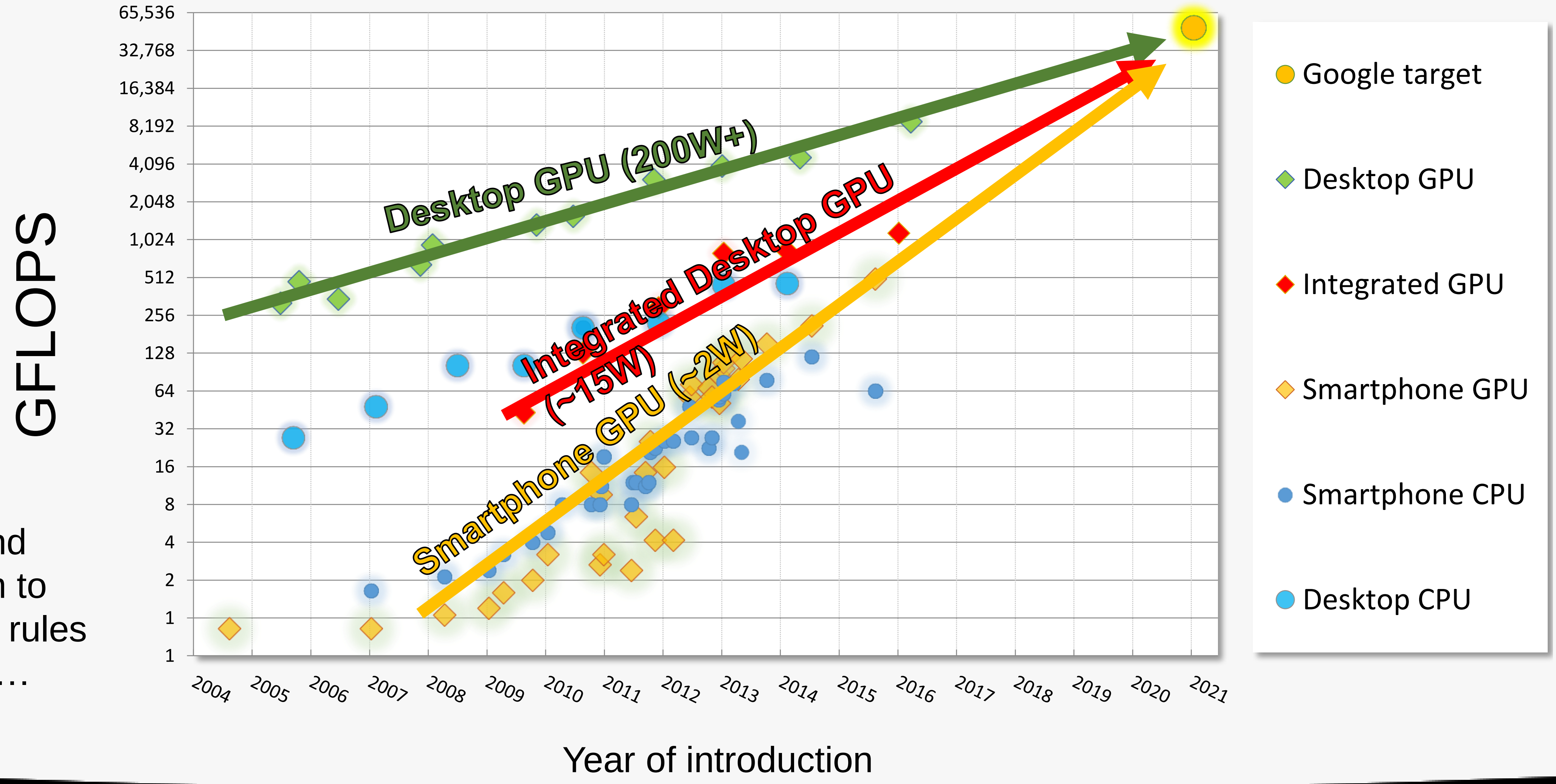
- How Codeplay can help deliver embedded intelligence

Where do we need to go?

“On a 100 millimetre-squared chip, Google needs something like 50 teraflops of performance”

- Daniel Rosenband (Google’s self-driving car project) at HotChips 2016

Performance trends



These trend lines seem to violate the rules of physics…

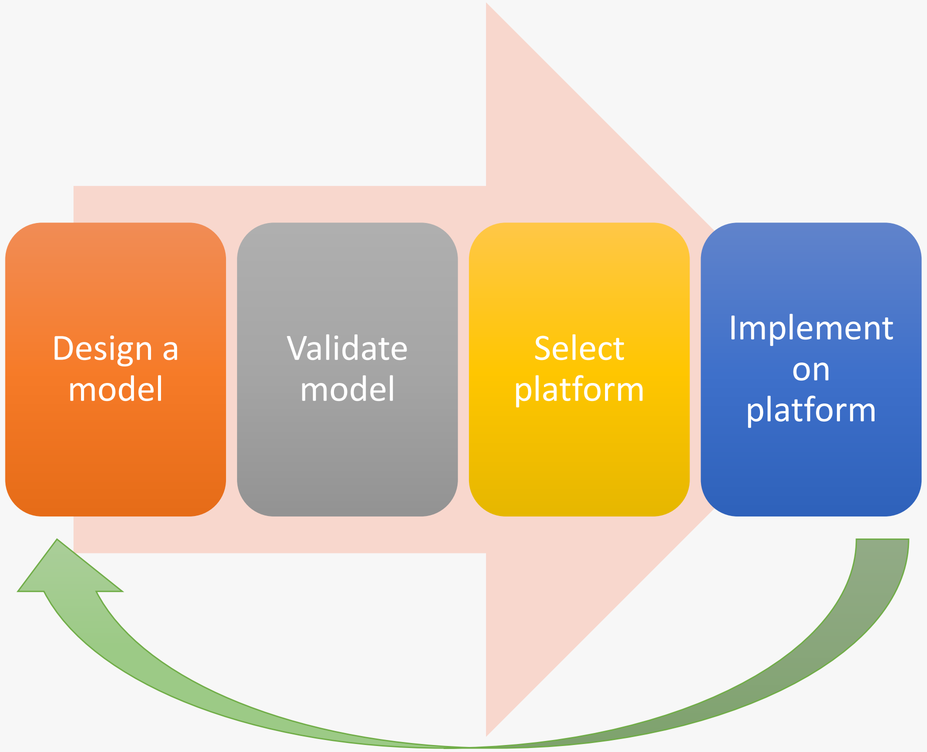
How do we get there from here?

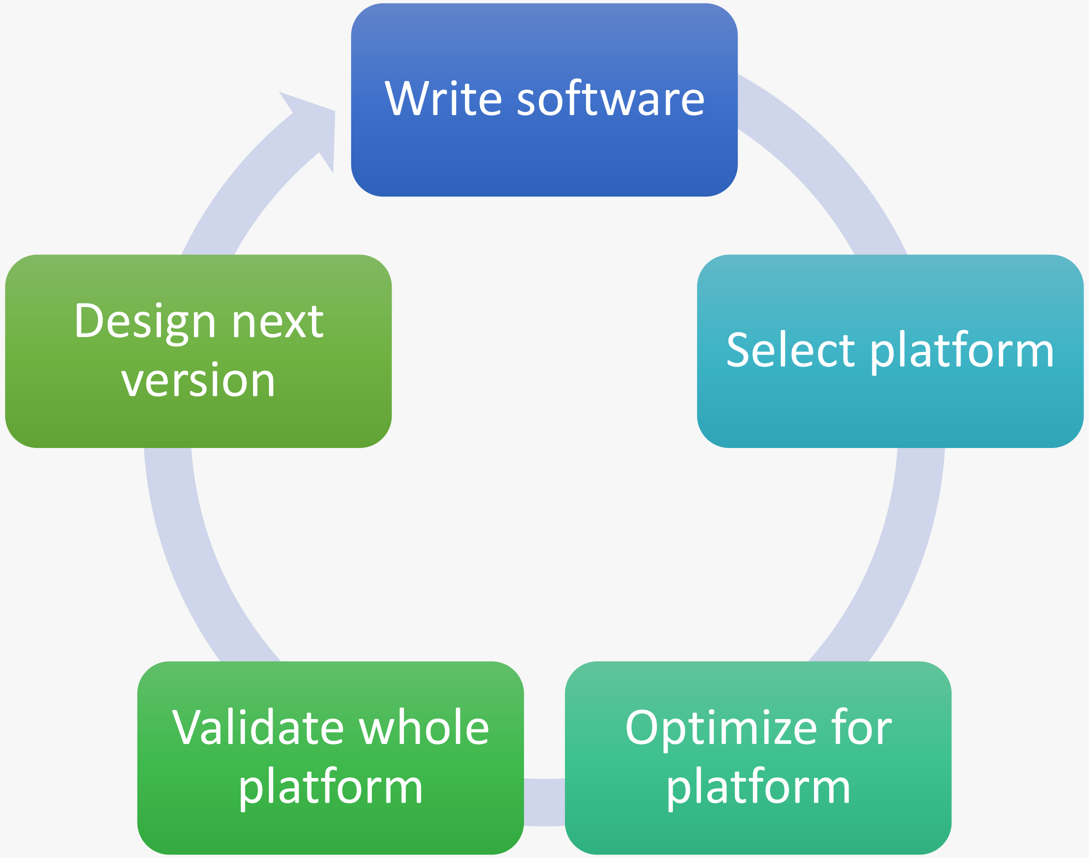
We need to write software today for platforms that cannot be built yet

We need to start with simpler systems that are not fully autonomous

We need to validate the systems as safe

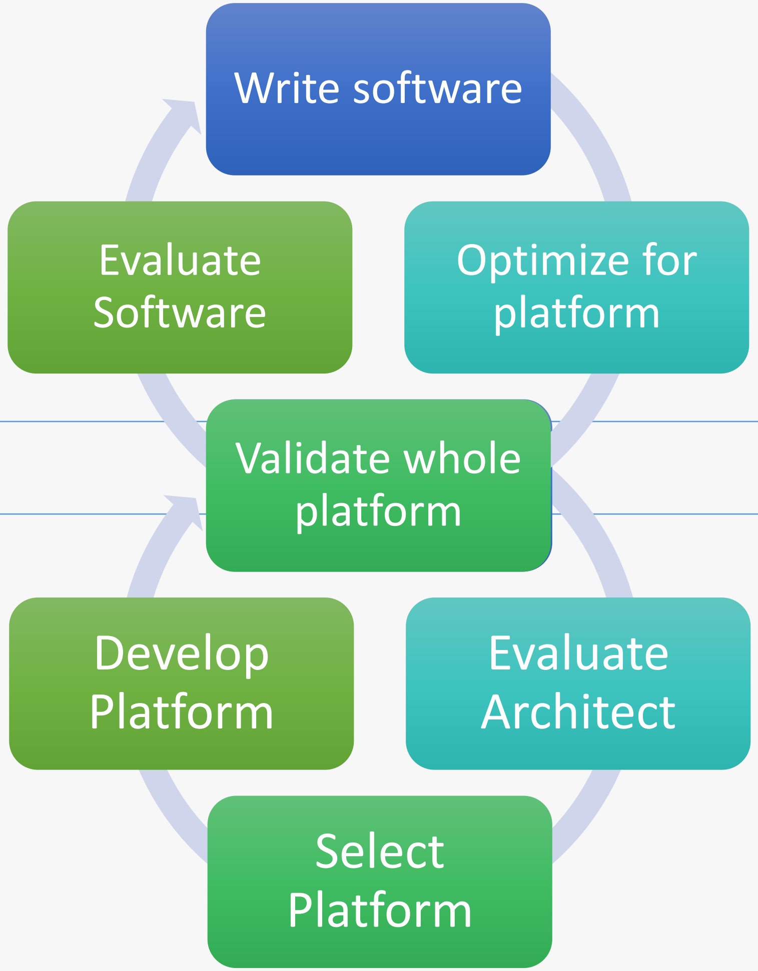
Two models of software development





Which method can get us all the way to full autonomy?

Desirable Solution Development

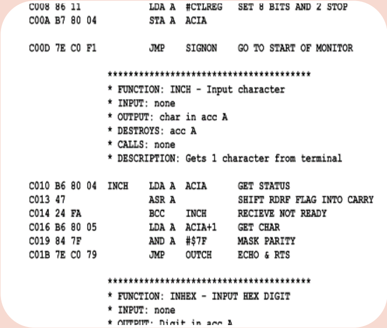


Software Application

Well Defined Middleware

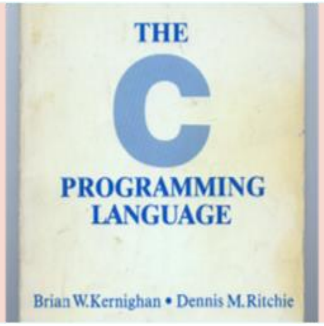
Hardware & Low-Level Software

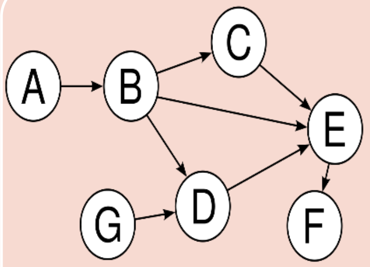
The different levels of programming model



```
C008 86 11          LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04       STA A  ACIA

C00D 7E C0 F1       JMP    SIGNON    GO TO START OF MONITOR

              ****************************************
              * FUNCTION: INCH - Input character
              * INPUT: none
              * OUTPUT: char in acc A
              * DESTROYS: acc A
              * CALLS: none
              * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04 INCH  LDA A  ACIA      GET STATUS
C013 47             ASR A            SHIFT RDRF FLAG INTO CARRY
C014 24 FA          BCC    INCH      RECIEVE NOT READY
C016 B6 80 05       LDA A  ACIA+1    GET CHAR
C019 84 7F          AND A  #$7F      MASK PARITY
C01B 7E C0 79       JMP    OUTCH     ECHO & RTS

              ****************************************
              * FUNCTION: INHEX - INPUT HEX DIGIT
              * INPUT: none
              * OUTPUT: Digit in acc A
```

SPIR





Device-specific programming

- Assembly language
- VHDL
- Device-specific C-like programming models

Higher-level language enabler

- NVIDIA PTX
- HSA
- OpenCL SPIR
- SPIR-V

C-level programming

- OpenCL C
- DSP C
- MCAPI/MTAPI

C++-level programming

- SYCL
- CUDA
- HCC
- C++ AMP

Graph programming

- OpenCV
- OpenVX
- Halide
- VisionCpp
- TensorFlow
- Caffe

Why graph programming?

When you scale the number of cores:

- You don't scale the number of memory ports
- Your compute performance increases
- But your off-chip memory bandwidth does not

Therefore:

- You need to reduce off-chip memory bandwidth by processing everything on-chip
- This is achieved by *tiling*

However, writing tiled image pipelines is hard

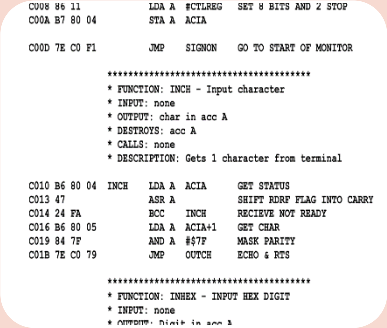
If we build up a graph of operations (e.g. convolutions) and then have a runtime system split into fused tiled operations across an entire system-on-chip, we get great performance

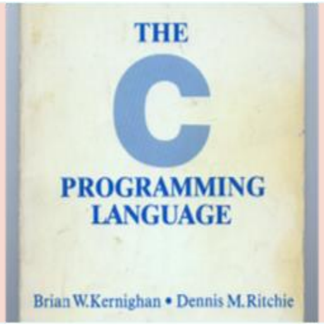
The route to full autonomy

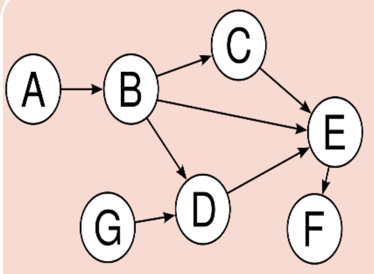
- Graph programming
  - This is the most widely-adopted approach to machine vision and machine learning

- Open standards
  - This lets you develop today for future architectures

Which model should we choose?







Device-specific programming

- Assembly language
- VHDL
- Device-specific C-like programming models

Higher-level language enabler

- NVIDIA PTX
- HSA
- OpenCL SPIR
- SPIR-V

C-level programming

- OpenCL C
- DSP C
- MCAPI/MTAPI

C++-level programming

- SYCL
- CUDA
- HCC
- C++ AMP

Graph programming

- OpenCV
- OpenVX
- Halide
- VisionCpp
- TensorFlow
- Caffe

They are not *alternatives*, they are *layers*

Graph programming

OpenCV	OpenVX	Halide	VisionCpp	TensorFlow	Caffe

C/C++-level programming

SYCL	CUDA	HCC	C++ AMP	OpenCL

Higher-level language enabler

NVIDIA PTX	HSA	OpenCL SPIR	SPIR-V

Device-specific programming

Assembly language	VHDL	Device-specific C-like programming models

Can specify, test and validate each layer

Graph programming	
Validate graph models	Validate the code using standard tools

C/C++-level programming			
OpenCL/SYCL specs	Clsmith testsuite	Conformance testsuites	Wide range of other testsuites

Higher-level language enabler	
SPIR/SPIR-V/HSAIL specs	Conformance testsuites

Device-specific programming	
Device-specific specification	Device-specific testing and validation

For Codeplay, these are our layer choices

We have chosen a layer of standards, based on current market adoption

- TensorFlow and OpenCV
- SYCL
- OpenCL (with SPIR)
- LLVM as the standard compiler back-end

Device-specific programming

- LLVM

Higher-level language enabler

- OpenCL SPIR

C/C++-level programming

- SYCL

Graph programming

- TensorFlow
- OpenCV

The actual choice of standards may change based on market dynamics, but by choosing widely adopted standards and a layering approach, it is easy to adapt

For Codeplay, these are our products

VisionCpp™

Graph programming

- TensorFlow
- OpenCV

ComputeCpp™

C/C++-level programming

- SYCL

ComputeAorta™

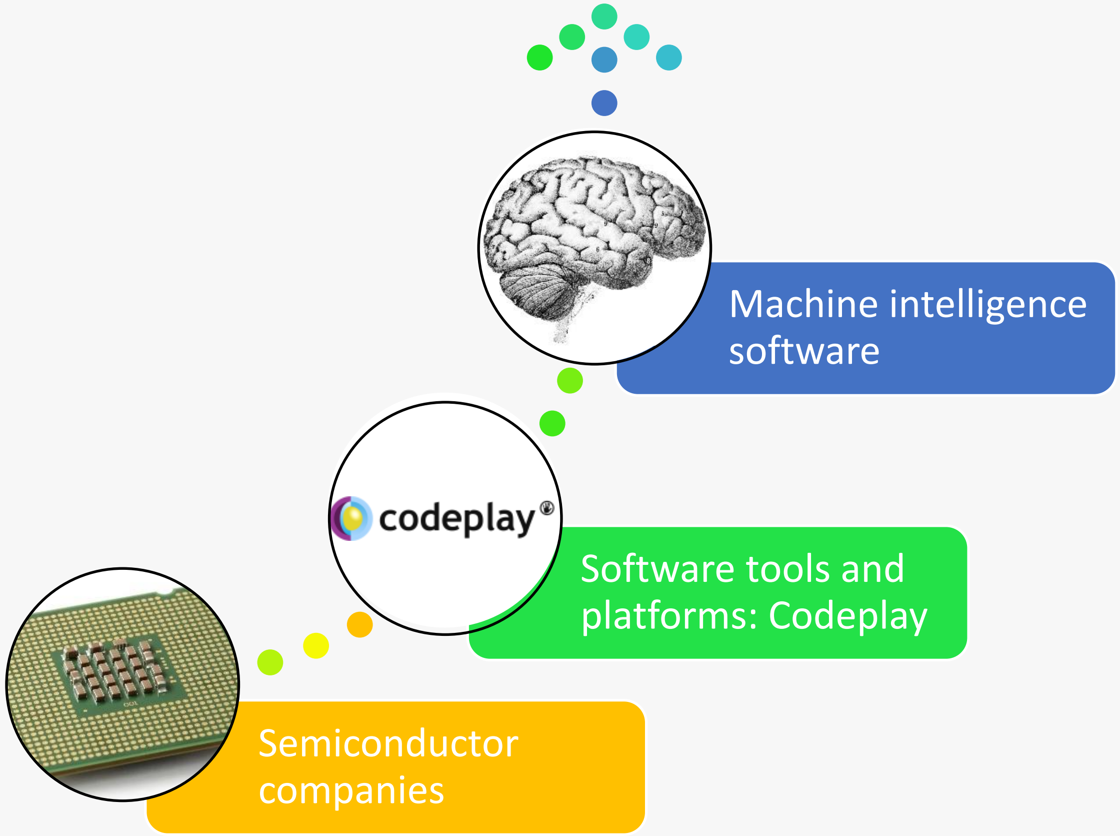
Higher-level language enabler

- OpenCL SPIR

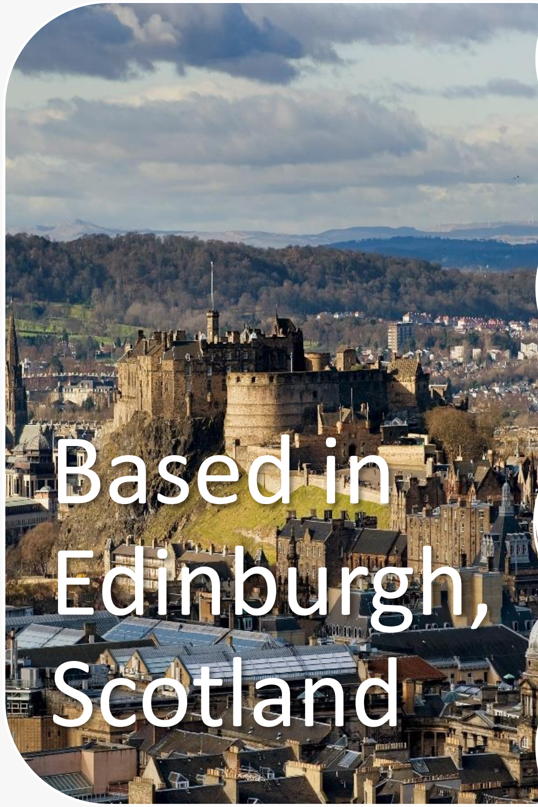
Device-specific programming

- LLVM

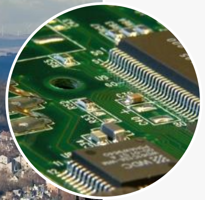
Where Codeplay fits in



Company




~60 staff, mostly engineering


License and customize technologies for semiconductor companies

ComputeAorta and ComputeCpp: implementations of OpenCL, Vulkan and SYCL


15+ years of experience in heterogeneous solutions

Further information

- OpenCL https://www.khronos.org/opencl/
- OpenVX https://www.khronos.org/openvx/
- HSA http://www.hsafoundation.com/
- SYCL http://sycl.tech
- OpenCV http://opencv.org/
- Halide http://halide-lang.org/
- VisionCpp https://github.com/codeplaysoftware/visioncpp

codeplay®

THE HETEROGENEOUS SYSTEMS EXPERTS

Charles.Macfarlane@Codeplay.com

VP Product Marketing

@codeplaysoft

/codeplaysoft

codeplay.com