Compiler Construction: Practical Introduction

System Course for SRR Engineers

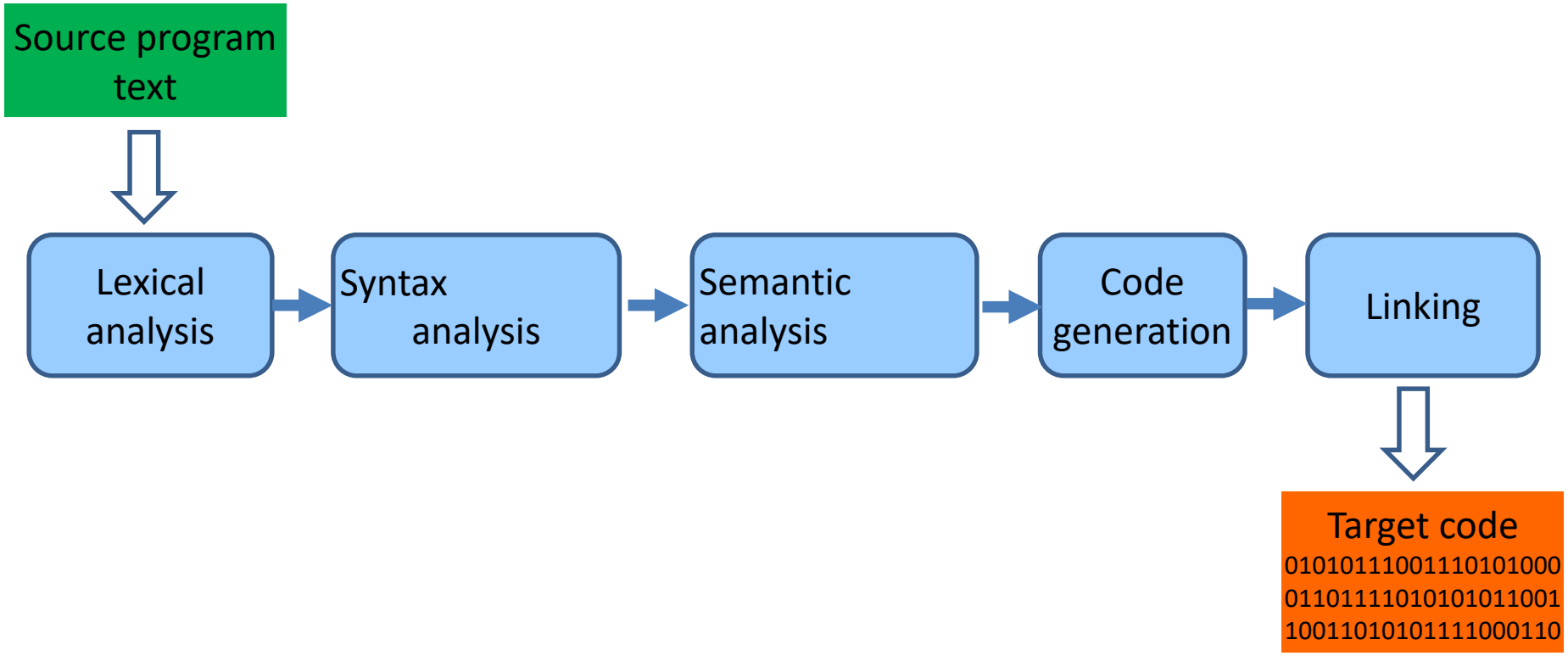
Samsung Research Russia
Moscow 2019

Lecture 9
Modern Compiler Architecture

- Conventional monolitic compiler
- LLVM compiler infrastructure
- V8 JavaScript Engine
- OpenCL framework

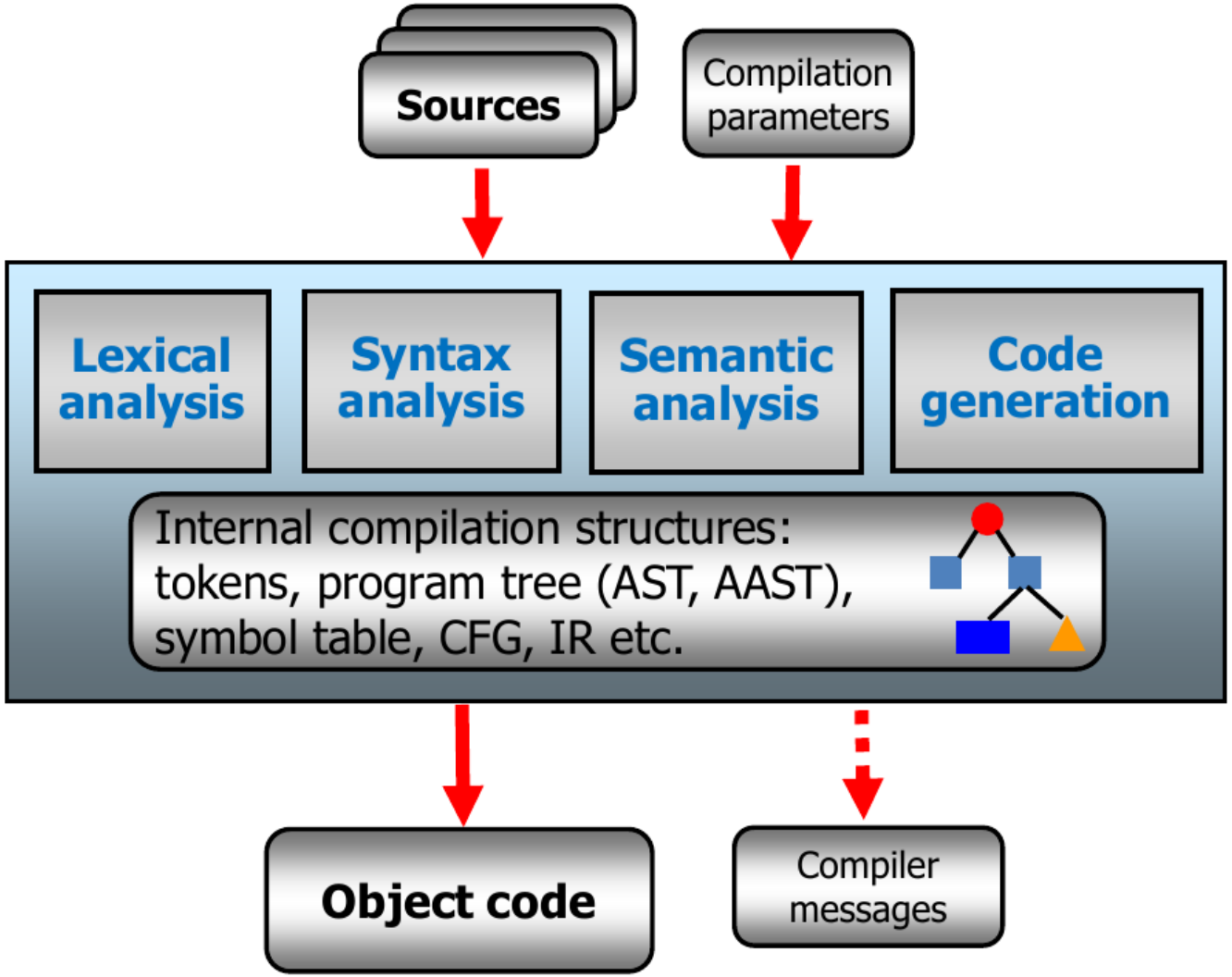
Compilation: An Ideal Picture

A program written by a human (or by another program)



A program binary image suitable for immediate execution by a machine

Conventional Compiler



Toward Multi-Platform Compiler

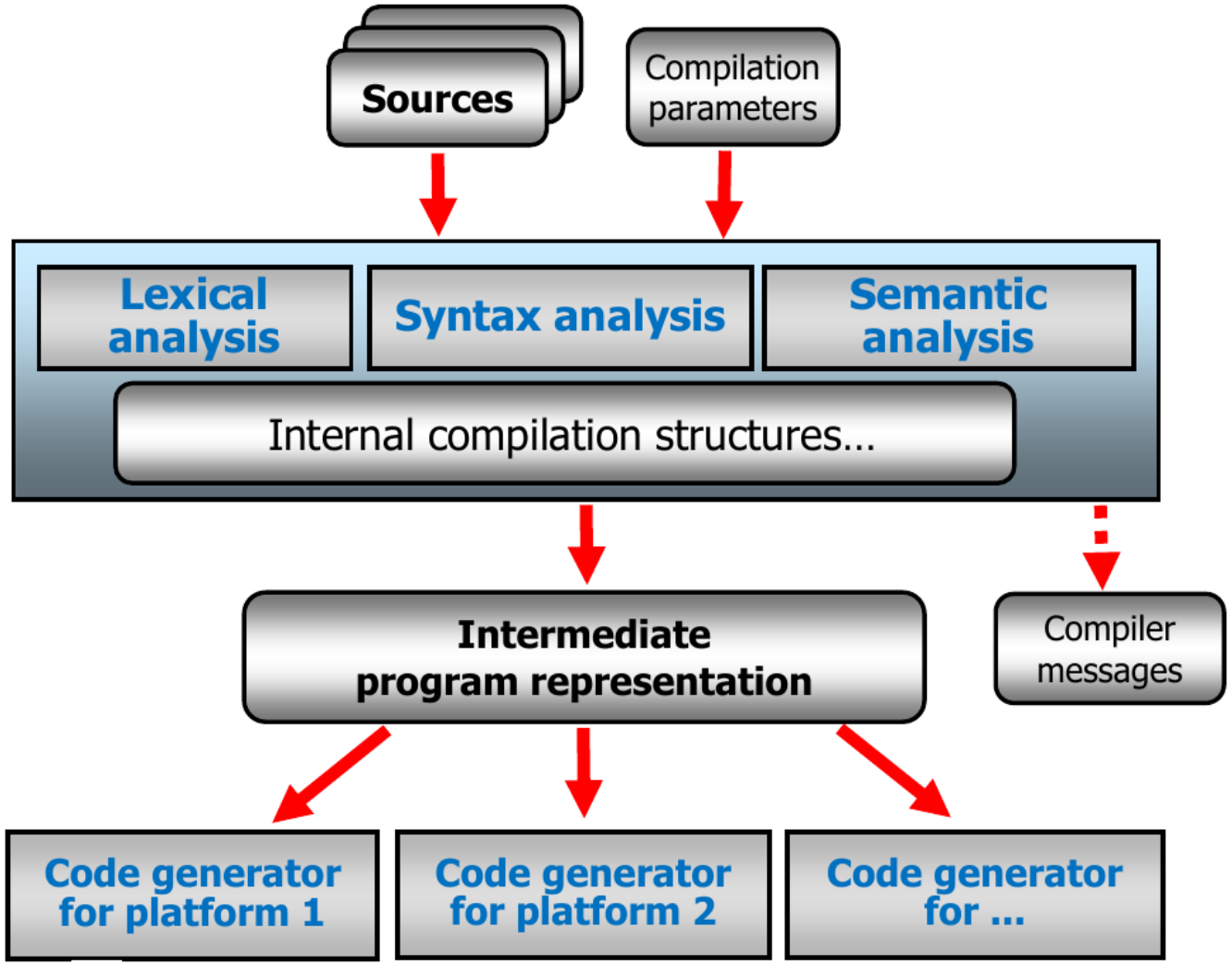
Issues with monolitic compiler

- One needs to rewrite compiler from scratch for every new programming language
- One needs to rewrite compiler from scratch for every new processor architecture
- Compiler optimizations are not re-usable.

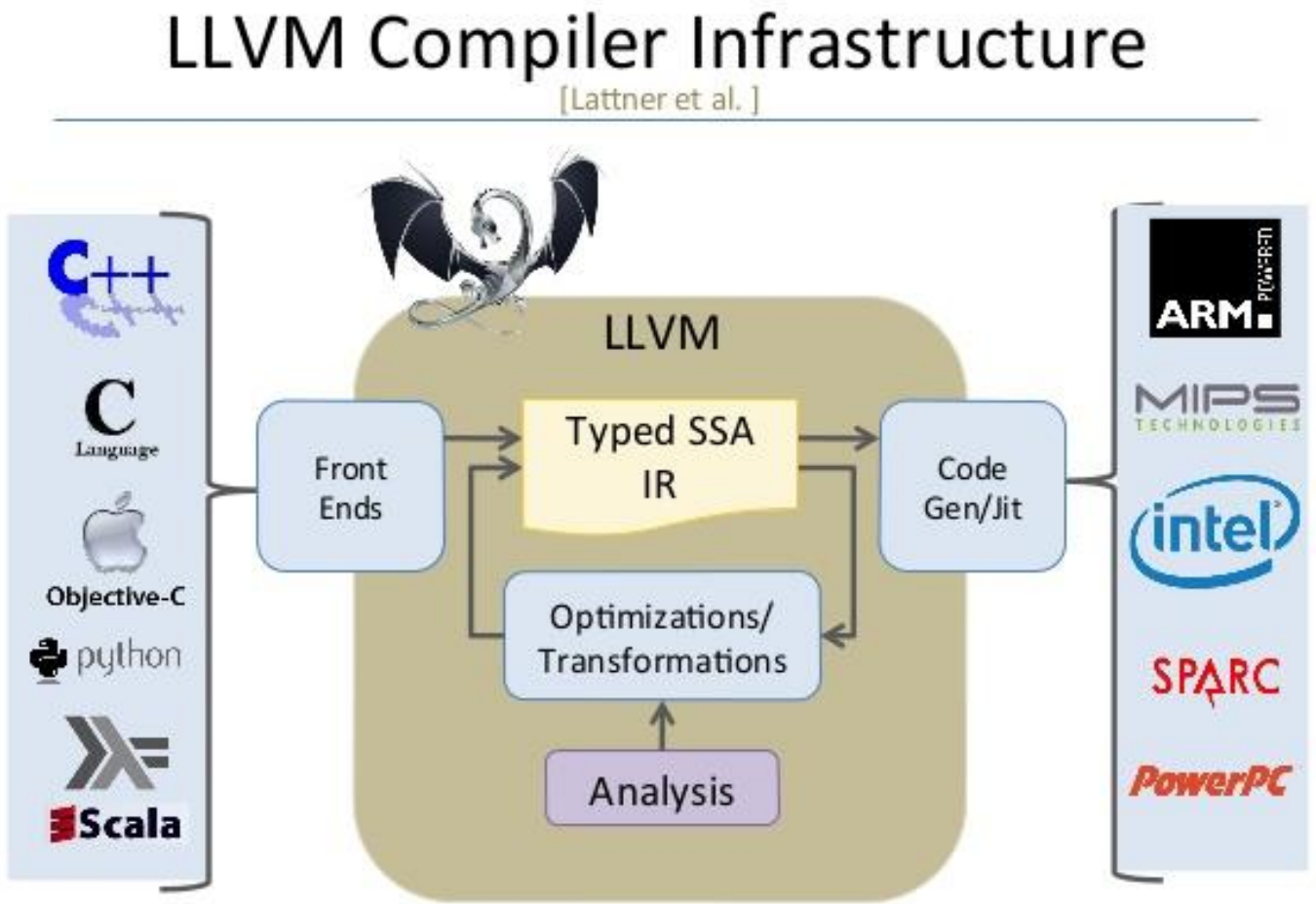
Multi-language multi-platform compiler

- Design common Intermediate Representation
- All source programming languages compile into the same IR
- Generate code to all processors from the same IR

Multi-platform Architecture



LLVM Infrastructure



LLVM Primary Components

- **The LLVM Virtual Instructions Set:**
  - The common language- and target-independent IR.
  - Internal (IR) and external (persistence) representations.
- **The collection of well-integrated libraries:**
  - Analysis, optimizations, code generators, JIT compiler, garbage collection support, profiling, …
- **The collection of tools built from the libraries:**
  - Assemblers, automatic debugger, linker, code generators, compiler driver, modular optimizer, …

LLVM IR: Goals

- **Easy to produce, understand, and define.**
- **Language- and Target-Independent:**
  - AST-level IR is not very feasible
    - Every analysis/xform must known about 'all' languages.
- **One IR for analysis and optimizations:**
  - IR must be able to support aggressive IPO, loop opts, scalar opts, … high- and low-level optimizations.
- **Optimize as much as early as possible:**
  - Can't postpone everything until link- or runtime.
  - No lowering in the IR.

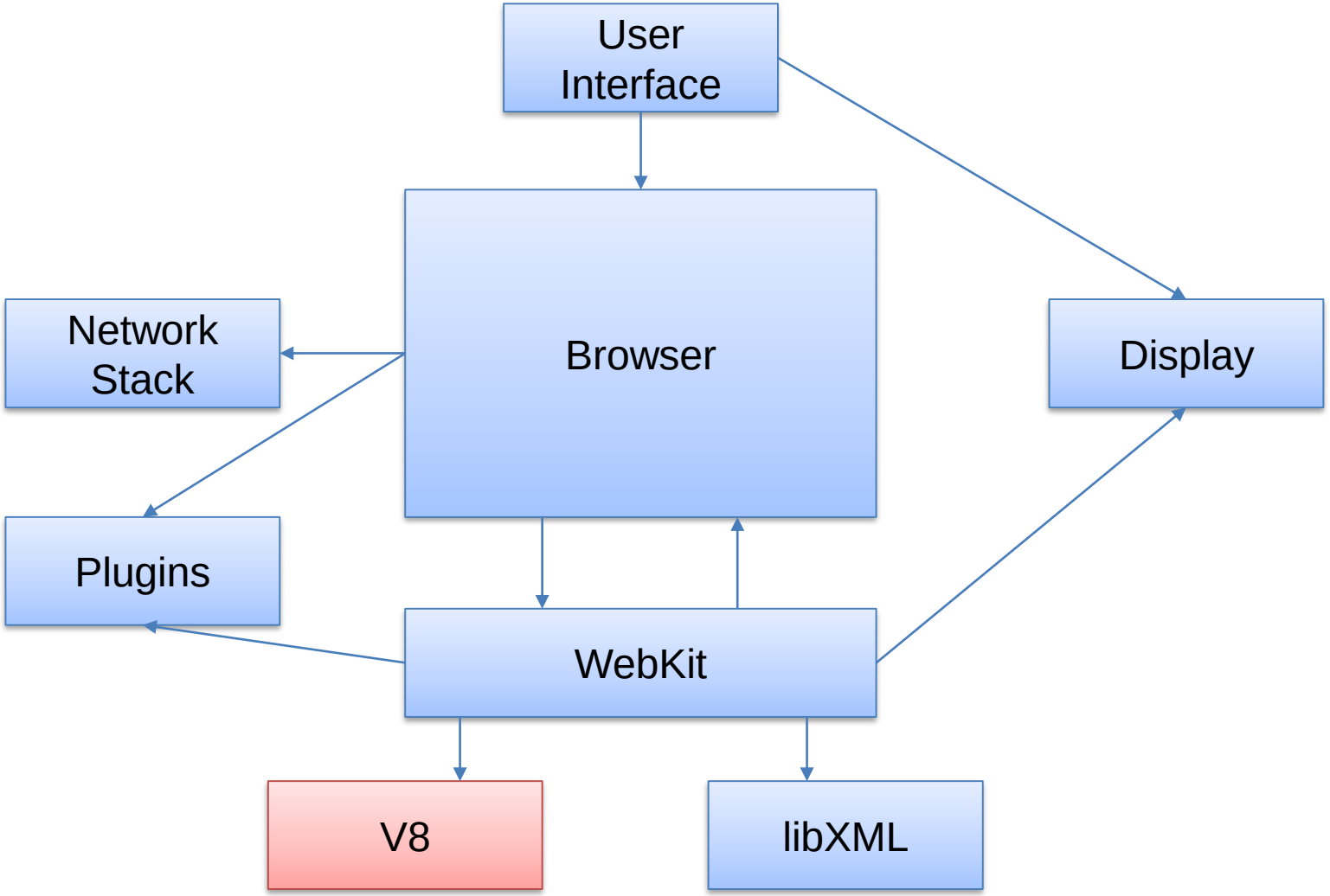
Five Points of LLVM IR

- **Extremely simple IR to learn and use:**
  - 1-to-1 correspondence between `.ll`, `.bc`, and C++ IR.
- **Powerful and modular optimizer:**
  - Easy to extend, or just use what is already there.
- **Clean and modular code generator:**
  - Easy to retarget, easy to replace/tweak components.
- **Many "productivity tools" (bugpoint, verifier).**
- **Active development community, good documentation.**

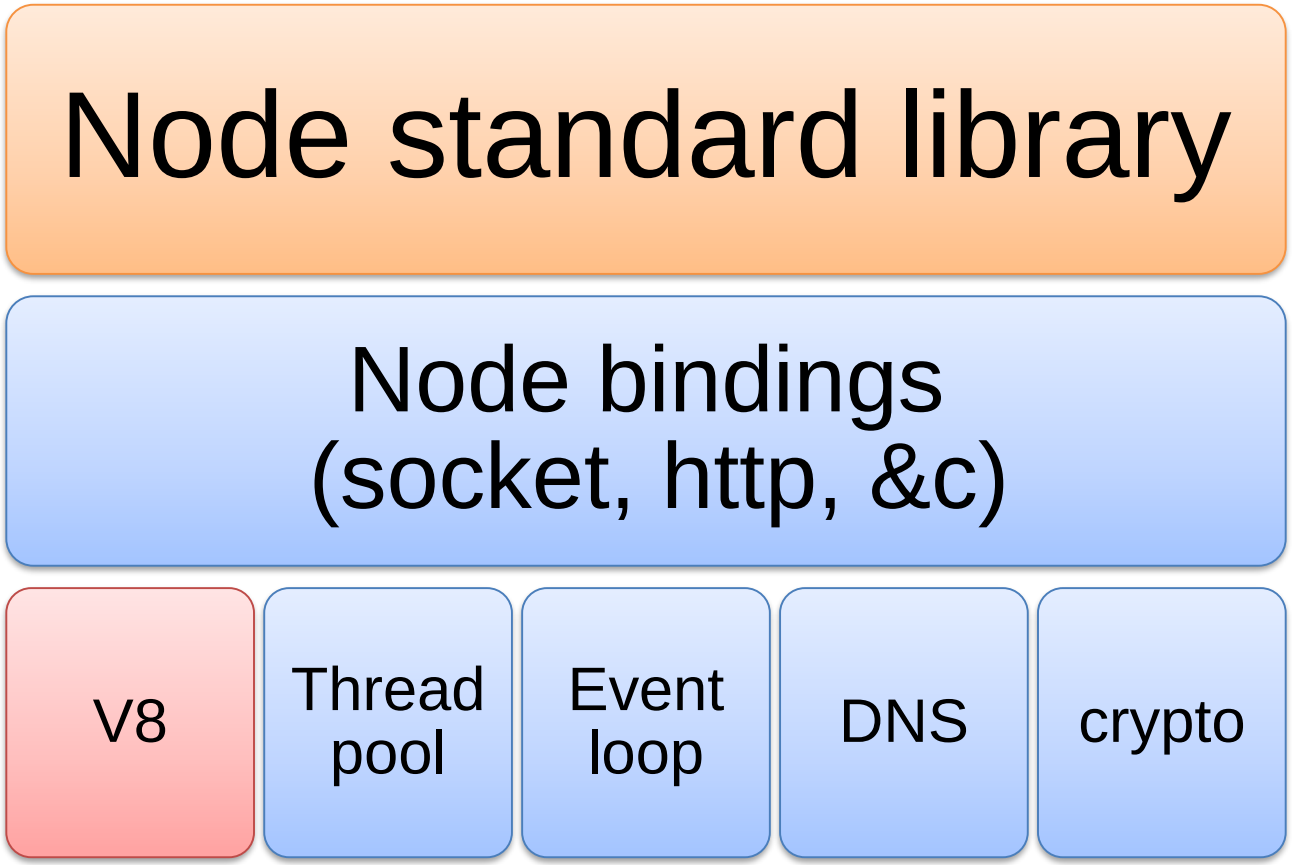
V8 JavaScript Engine

- JavaScript language is the only programming language for the Web (Webassembly is still not in production)
  - Invented by Brandan Eich in 1995, Netscape Navigator
  - Prototype based objects
  - Dynamic typing
  - Functions as a first class citizen
- Google V8 is the one of the fastest JavaScript Engines
  - Part of the Google Chrome web brwoser
  - Part of the Node.js server
  - Just-in-Time compilation

V8 in Chrome


User Interface
Network Stack
Browser
Display
Plugins
WebKit
V8
libXML

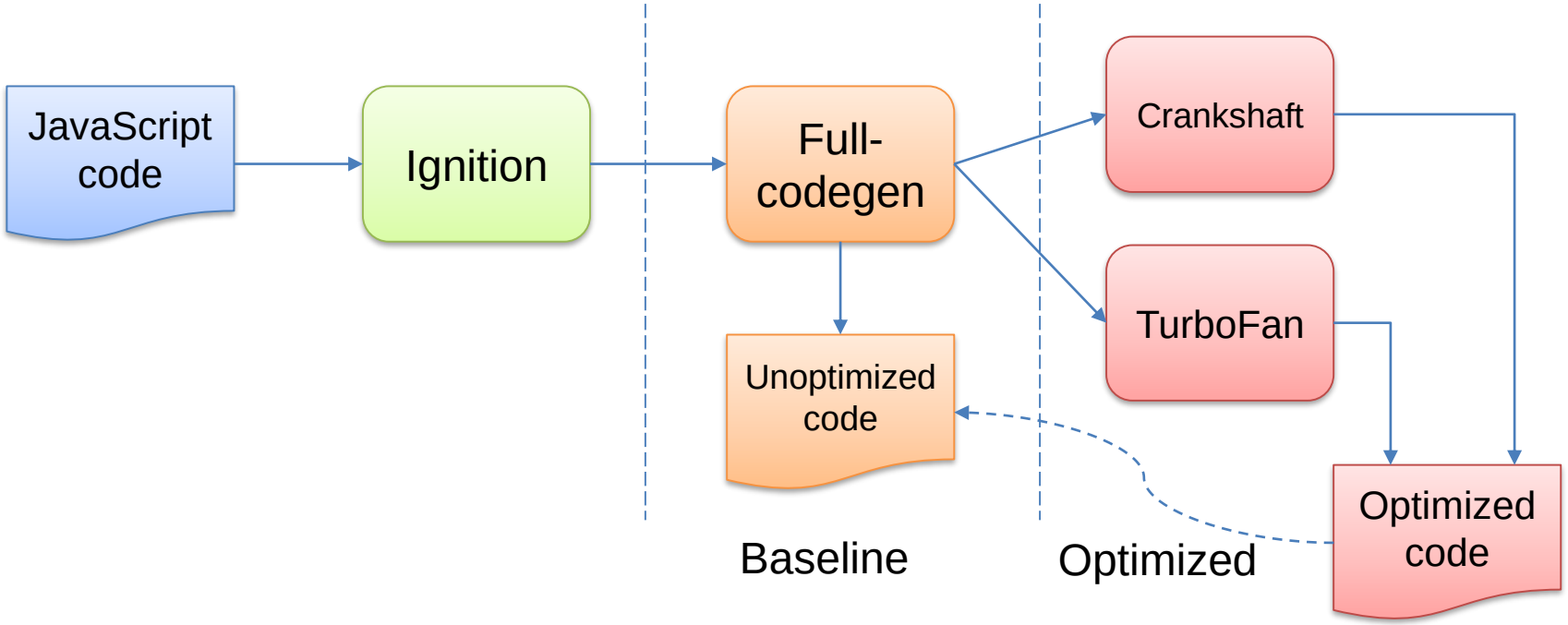
V8 in Node.js


Node standard library
Node bindings
(socket, http, &c)
V8
Thread pool
Event loop
DNS
crypto

JavaScript Compilation Issues

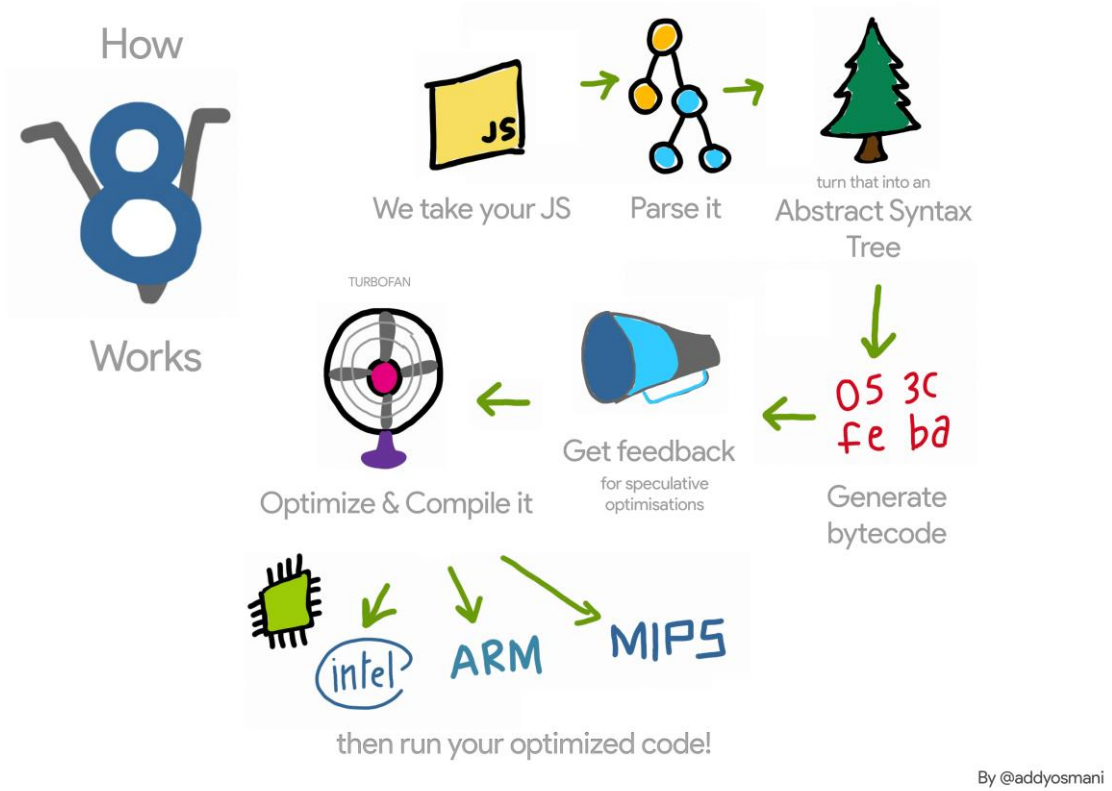
- Dynamic typing
  - Any variable can be assigned object of any type
  - The type of variable can be changed during program execution
- Prototype based objects
  - Object are created by cloning other objects
  - Every object has link to its prototype object
  - Prototype object can be changed during program execution
- eval() function: new code can be created at run time

V8 Architecture



- JavaScript source code compiled to internal represenation
- Full-codegen generates generic unoptimized machine code
- Hot regions are compiled to optimized code

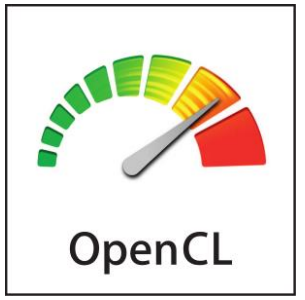
V8 Compilation Strategies



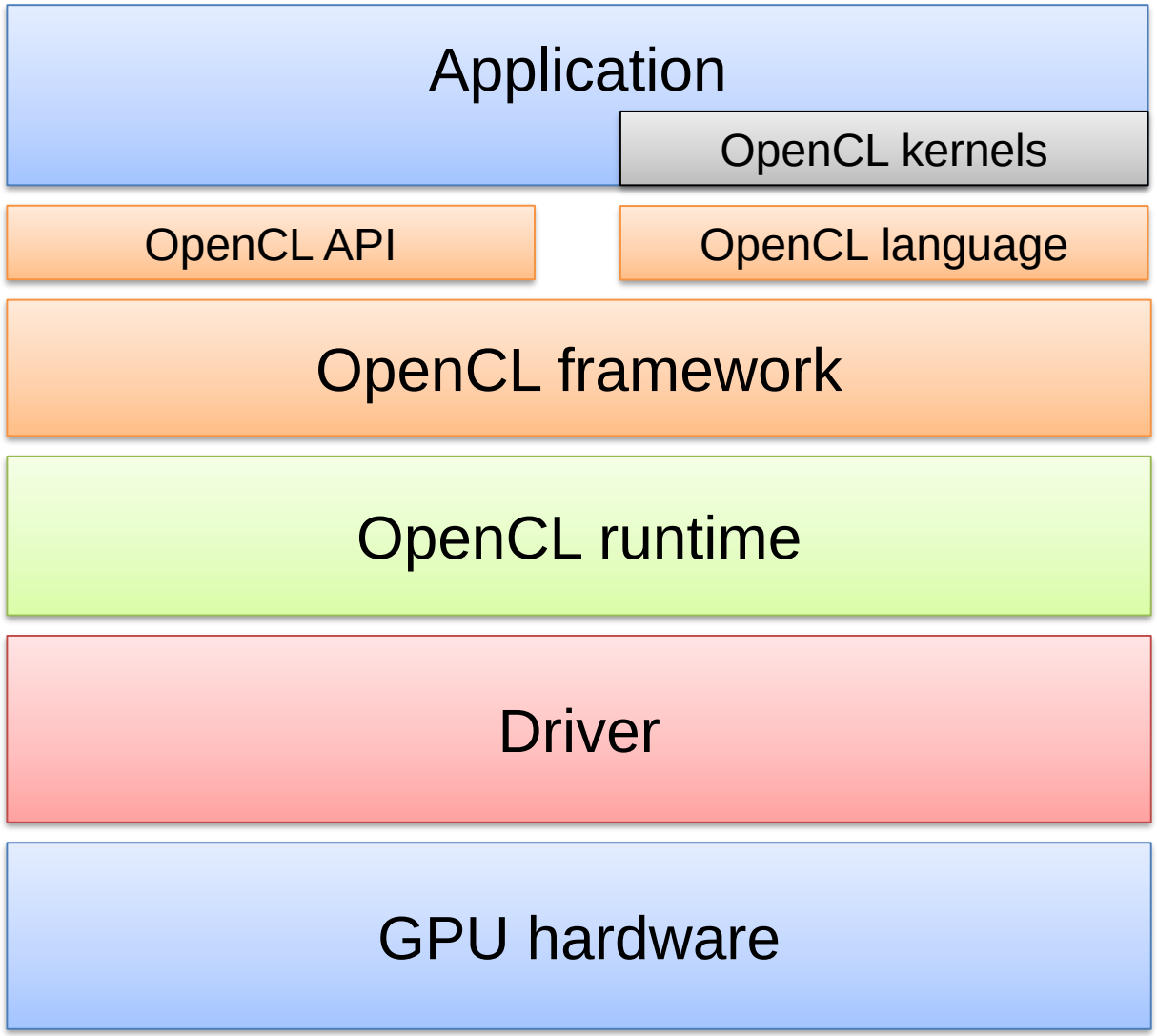
- Fast unoptimized code generation
- Inline Caching: virtual table at invocation site
- Hot region profiling: counters insertion
- Type-dependent optimization of hot-regions
- On-stack Replacement of recompiled code

OpenCL

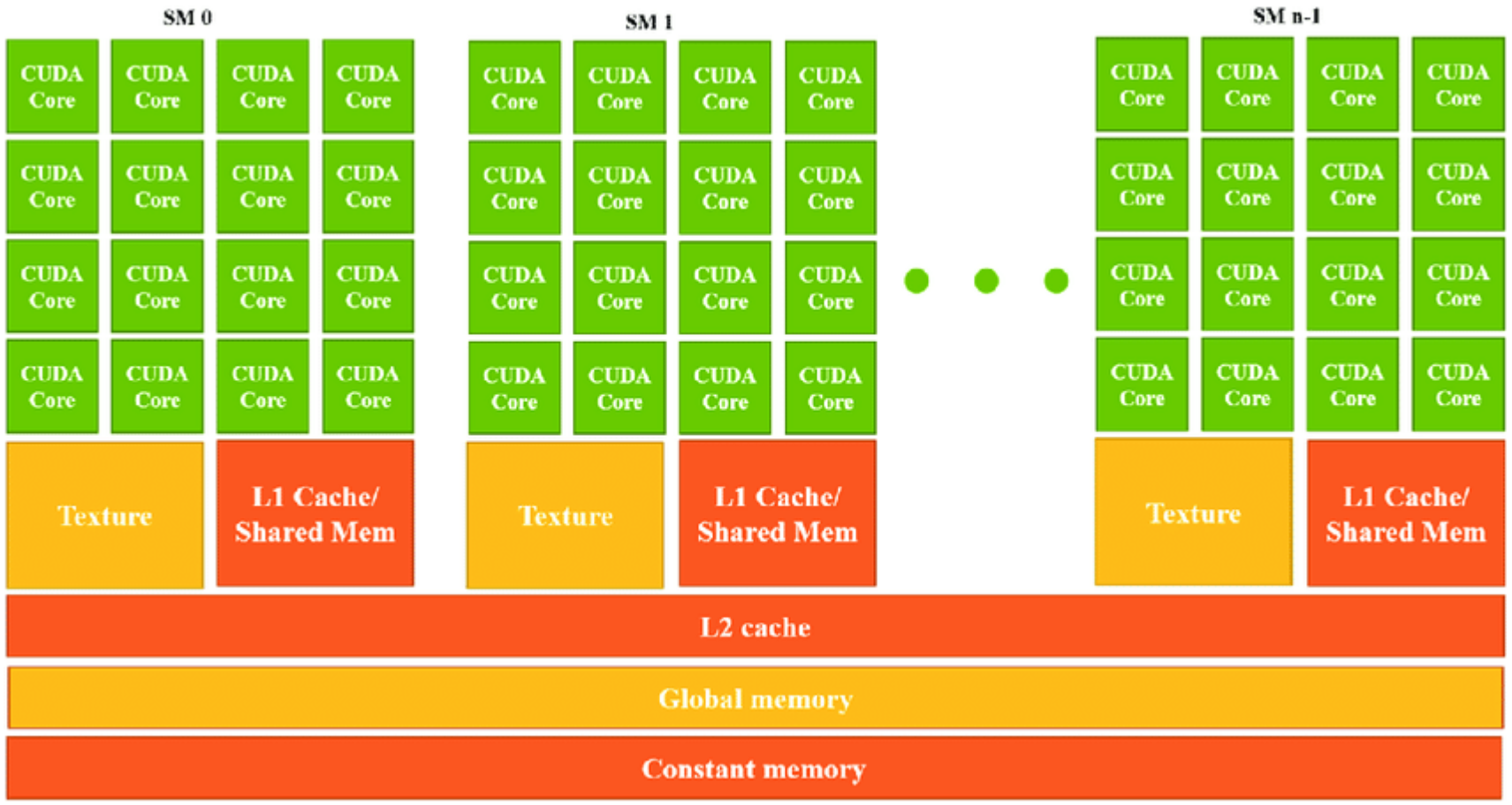
- OpenCL™ (Open Computing Language) is the open standard for cross-platform, parallel programming of diverse processors.
- OpenCL specifies:
  - A programming language based on C99 and C++11
  - Application programming interfaces (APIs) to control the platform and execute programs on the compute devices.
- OpenCL is constituted of:
  - OpenCL programming language compiler
  - OpenCL APIs library
  - Driver to control execution



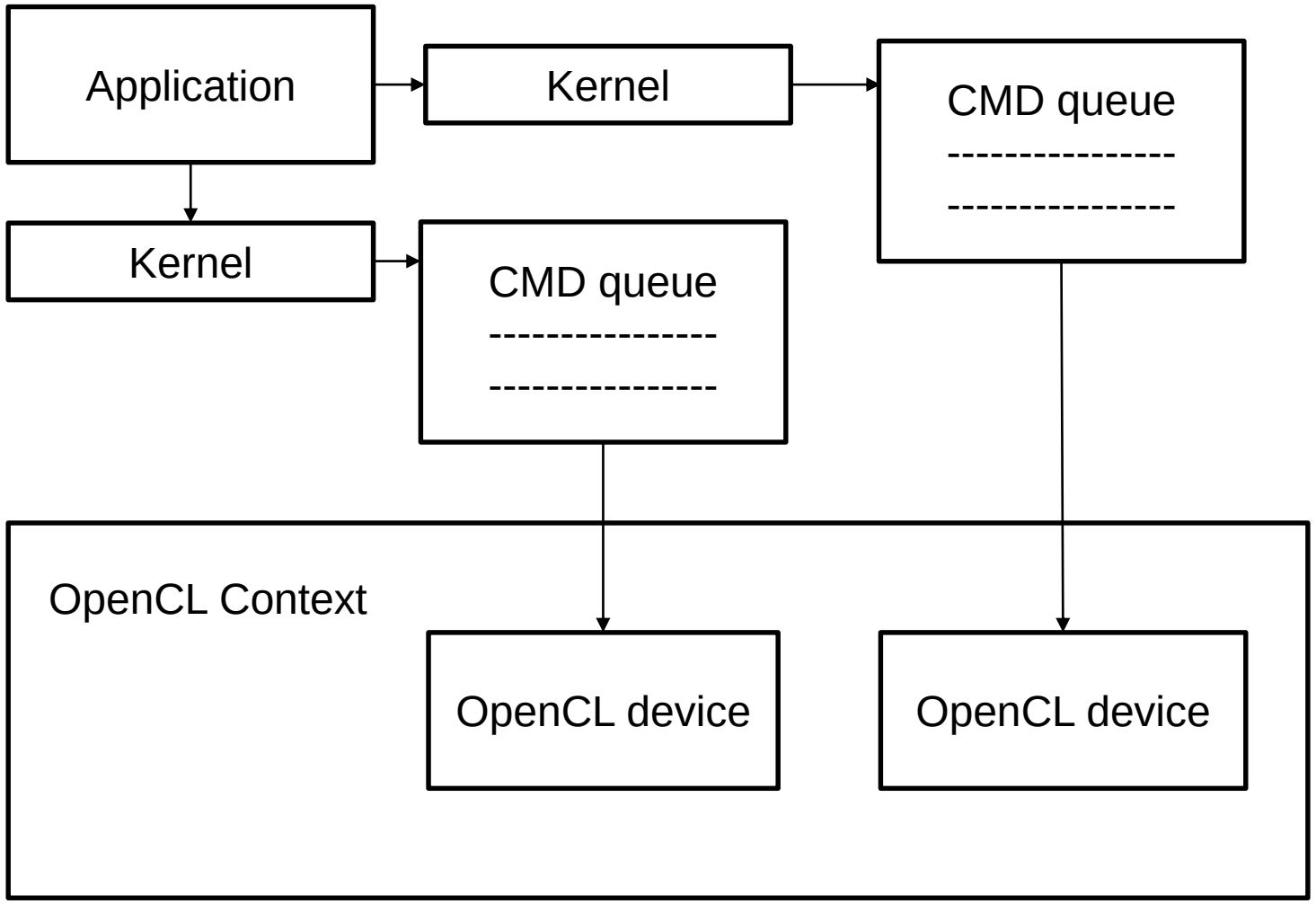
OpenCL Architecture



GPU Architecture



OpenCL Application



OpenCL Program

- An OpenCL «program» contains one or more «kernels» and any supporting routines that run on target device
- An OpenCL kernel is the basic unit of code that can be executed on a target device.

```
// Multiplies A*x, leaving the result in y.
// A is a row-major matrix, meaning the (i,j) element is at A[i*ncols+j].
__kernel void matvec(__global const float *A, __global const float *x,
                     uint ncols, __global float *y)
{
   size_t i = get_global_id(0);              // Global id, used as the row index
   __global float const *a = &A[i*ncols];    // Pointer to the i'th row
   float sum = 0.f;                          // Accumulator for dot product
   for (size_t j = 0; j < ncols; j++) {
      sum += a[j] * x[j];
   }
   y[i] = sum;
}
```

OpenCL Host Program

```
clGetDeviceIDs(NULL, CL_DEVICE_TYPE_GPU, num, devices, NULL);
cl_context context = clCreateContextFromType(NULL,
CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);
clGetDeviceIDs(NULL, CL_DEVICE_TYPE_DEFAULT, 1, devices,
NULL);
cl_command_queue queue = clCreateCommandQueue(context,
devices[0], 0, NULL);
cl_mem memobjs[] = { clCreateBuffer(context, CL_MEM_READ_ONLY
| CL_MEM_COPY_HOST_PTR, sizeof(float) * 2 * NUM_ENTRIES, NULL,
NULL),
                                        clCreateBuffer(context,
CL_MEM_READ_WRITE, sizeof(float) * 2 * NUM_ENTRIES, NULL, NULL) };
cl_program program = clCreateProgramWithSource(context, 1, (const
char **)& KernelSource, NULL, NULL);
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
cl_kernel kernel = clCreateKernel(program, "fft1D_1024", NULL);
clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&memobjs[0]);
clEnqueueNDRangeKernel(queue, kernel, 1, NULL, global_work_size,
local_work_size, 0, NULL, NULL);
```

OpenCL Compiler

- OpenCL compiler works a part of OpenCL framework
- OpenCL compiler is a library called by driver
- It compiles kernels and supporting routines to target specific code
- A driver either schedule execution of compiled kernels on device or save it as a binary
- OpenCL compiler compiles the body of a loop nest
- Loop boudaries are managed by the driver