# Adding abstraction

## The OpenCL™ C++ bindings

# Goals

Lightweight, providing access to the low-level features of the original OpenCL™ C API

Compatible with standard C++ compilers (GCC 4.x and VS 2008)

C++ features that may be considered acceptable by all, e.g. exceptions, should not be required but may be supported by the use of polices that are not enabled by default

Should not require the use of the Standard Template Library

- Many organizations are wary of using the STL in their products

The bindings should be defined completely within a header, cl.hpp

- No linking to precompiled code should be necessary

# A simple example: Hello from OpenCL™ C++ bindings

```
#define __CL_ENABLE_EXCEPTIONS
#define __NO_STD_VECTOR
#define __NO_STD_STRING
#if defined(__APPLE__) || defined(__MACOSX)
#include <OpenCL/cl.hpp>
#else
#include <CL/cl.hpp>
#endif
#include <cstdio>
#include <cstdlib>
#include <iostream>

const char * helloStr = "__kernel void  hello(void) { }\n";
```

# A simple example: Hello from OpenCL™ C++ bindings (2)

```
int main(void) {
      try {
        cl::Context context(CL_DEVICE_TYPE_GPU, 0, NULL, NULL, &err);
        cl::vector<cl::Device> devices = context.getInfo<CL_CONTEXT_DEVICES>();
        cl::Program::Sources source(1, std::make_pair(helloStr,strlen(helloStr)));
        cl::Program program_ = cl::Program(context, source);
        program_.build(devices);
        cl::Kernel kernel(program_, "hello", &err);
        cl::CommandQueue queue(context, devices[0], 0, &err);
        cl::KernelFunctor func = kernel.bind(queue, cl::NDRange(4, 4),
    cl::NDRange(2, 2));
        func().wait();
      } catch (cl::Error err) {
        std::cerr << "ERROR: " << err.what() << "(" << err.err() << ")"  <<
    std::endl;
      }
       return EXIT_SUCCESS;
}
```