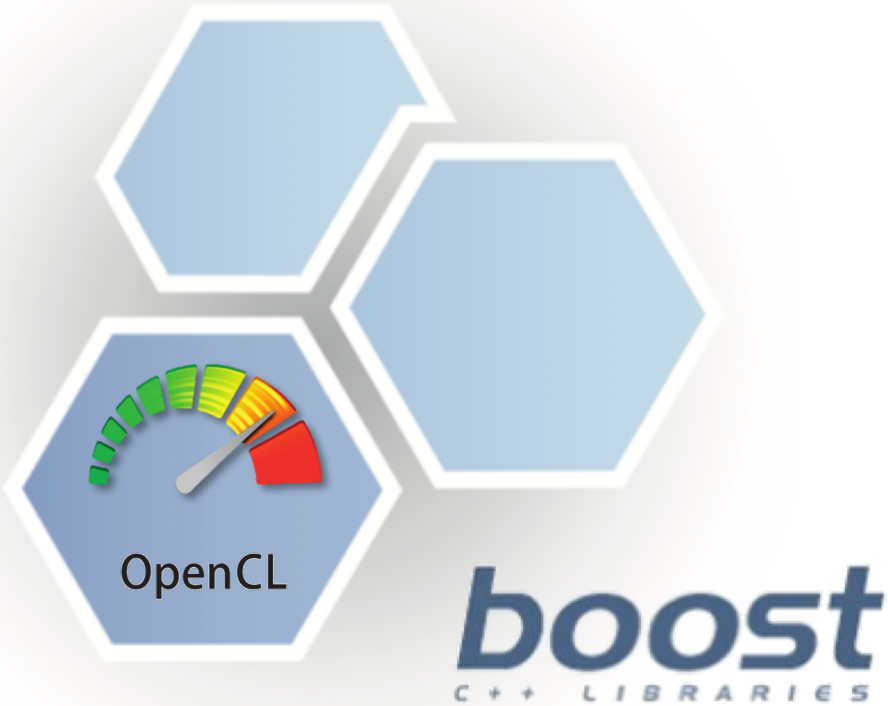
Harnessing the Power of Heterogeneous Computing using Boost.Compute + OpenCL

Armin Sobhani
asobhani@sharcnet.ca

SHARCNET

University of Ontario Institute of Technology (UOIT)

August 15, 2018



SHARCNET™

Outline

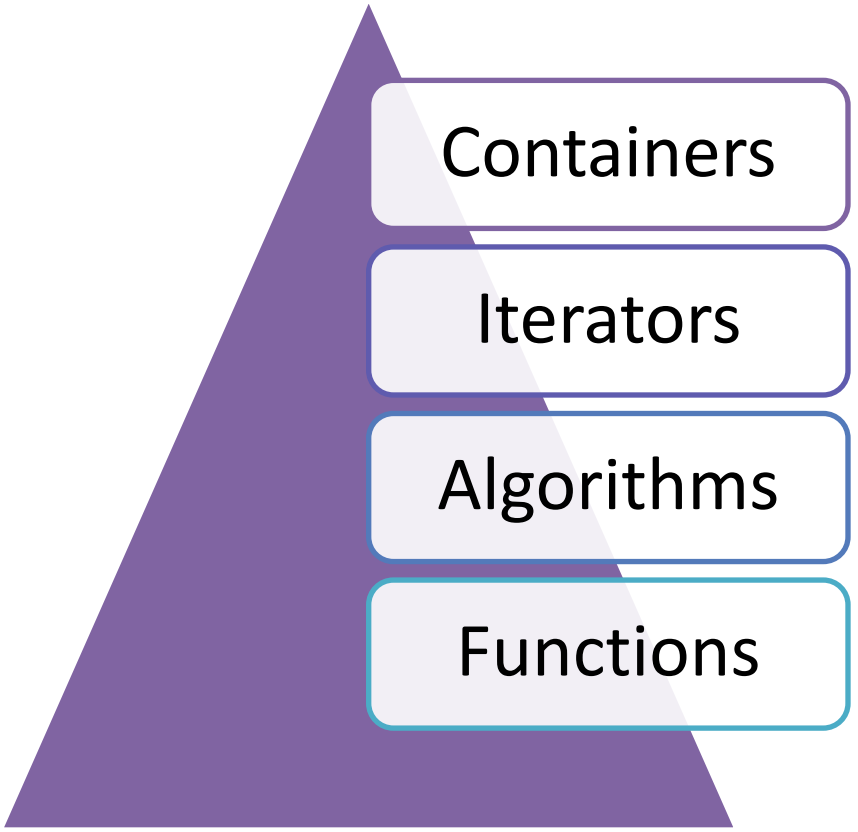
- A Quick introduction to Boost.Compute and OpenCL

- Tutorial for developing Boost.Compute applications on SHARCNet clusters:

  https://git.sharcnet.ca/asobhani/bc-tutorial

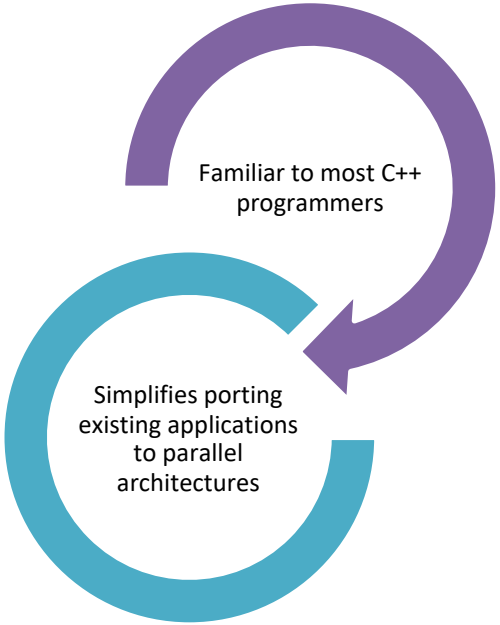
Standard Template Library (STL)

- Software library for the C++
- Influenced many parts of the C++ Standard Library
- Consisting of 4 components:



Parallel STL

Why?



Available Implementations

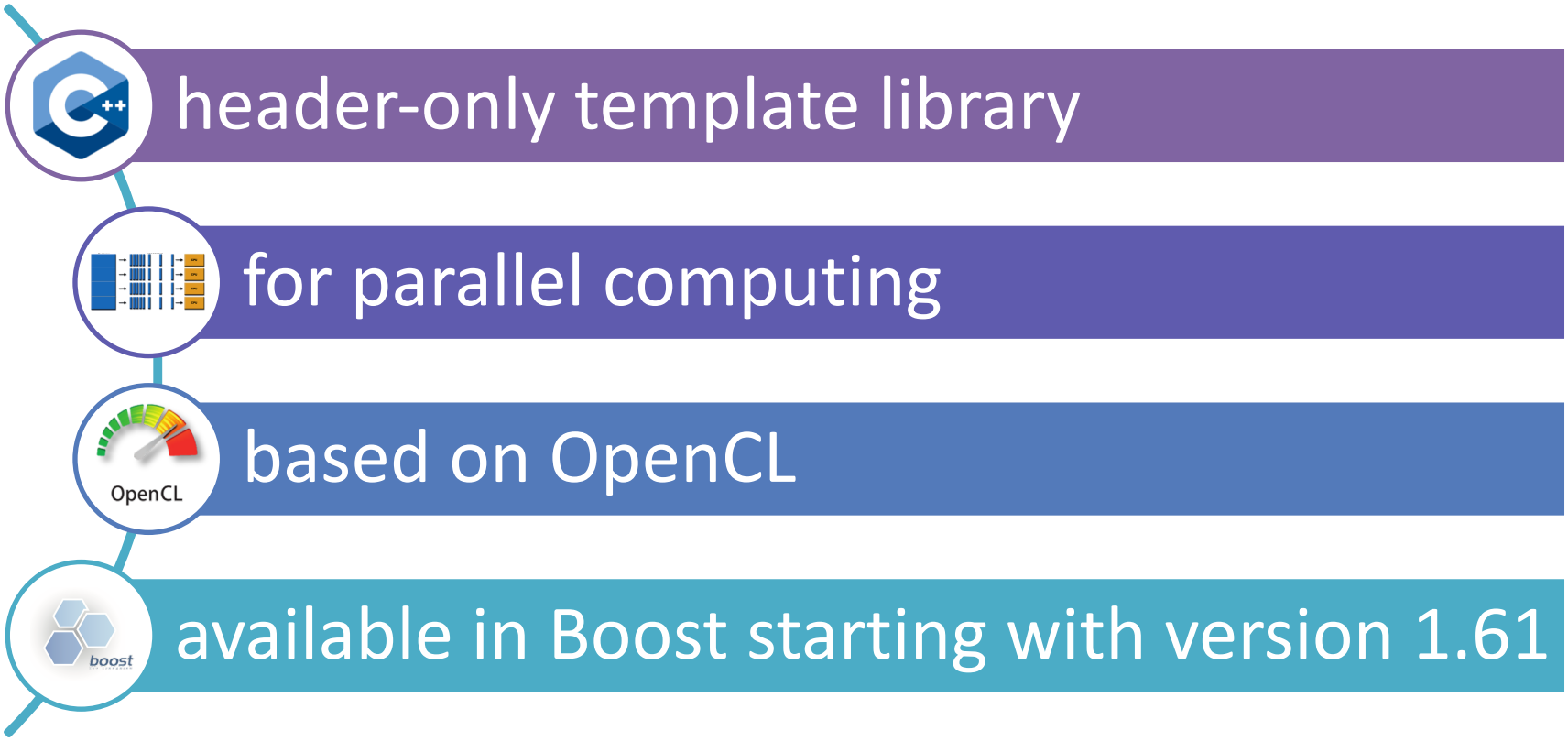
C++17 Parallel Algorithms

- Intel's open source Parallel STL
- KhronosGroup's SYCL Parallel STL

Third-Party C++ Libraries

- Boost.Compute
- Nvidia's Thrust
- AMD's Bolt

Boost.Compute

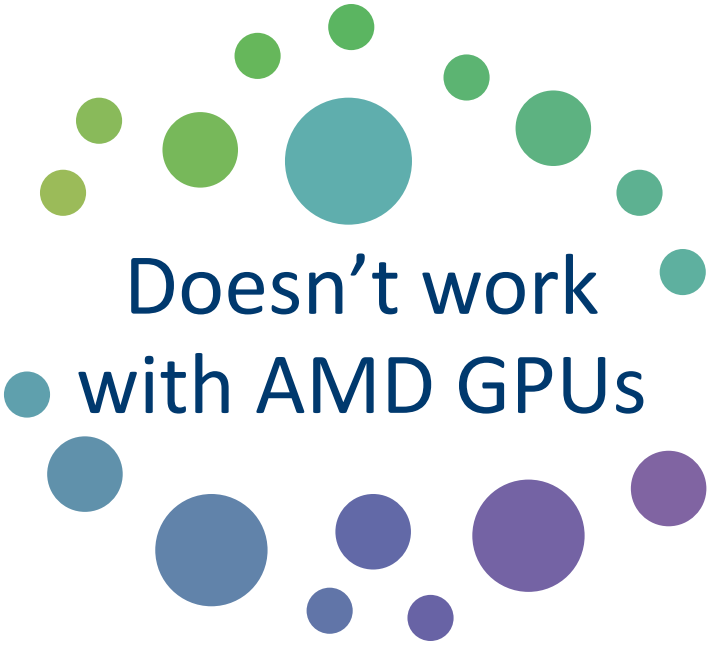


OpenCL

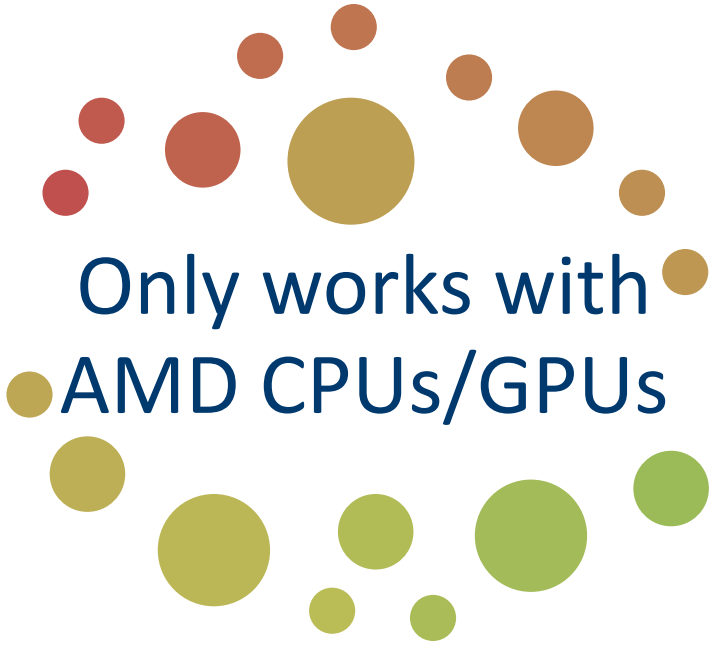
- Open Computing Language
- Open standard for parallel programming of heterogenous systems
- Latest version is 2.2
- Views a computing system as consisting of a number of *compute devices*
- Functions executed on an OpenCL device are called *kernels*
- Works by compiling C99 code at run-time to generate kernel objects

Boost.Compute vs. Thrust vs. Bolt

Nvidia's Thrust



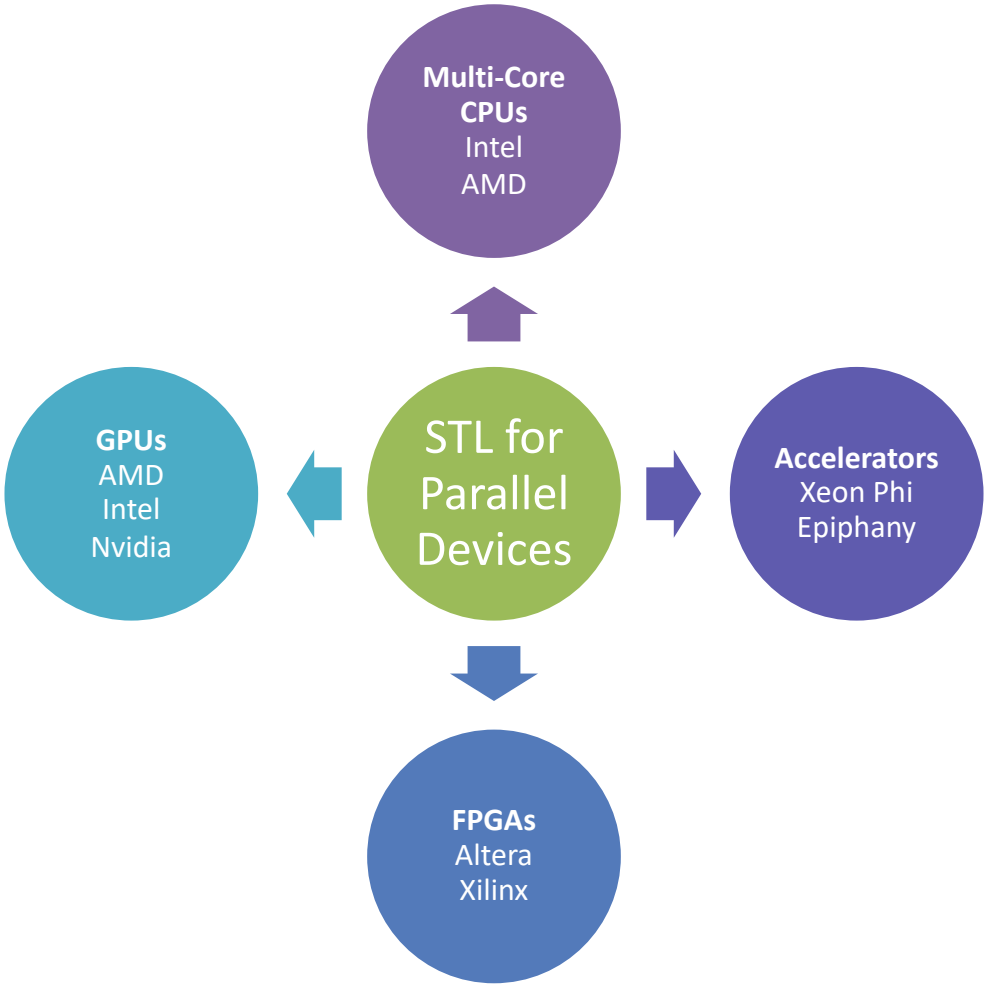
AMD's Bolt



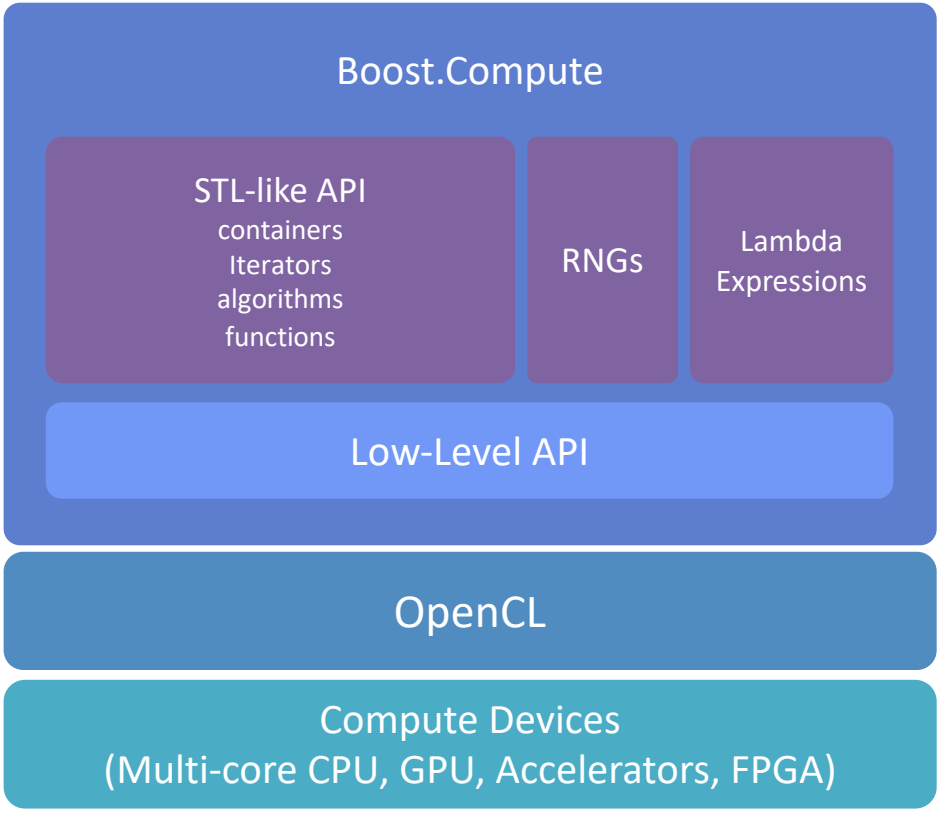
Boost.Compute 101

SHARCNET™

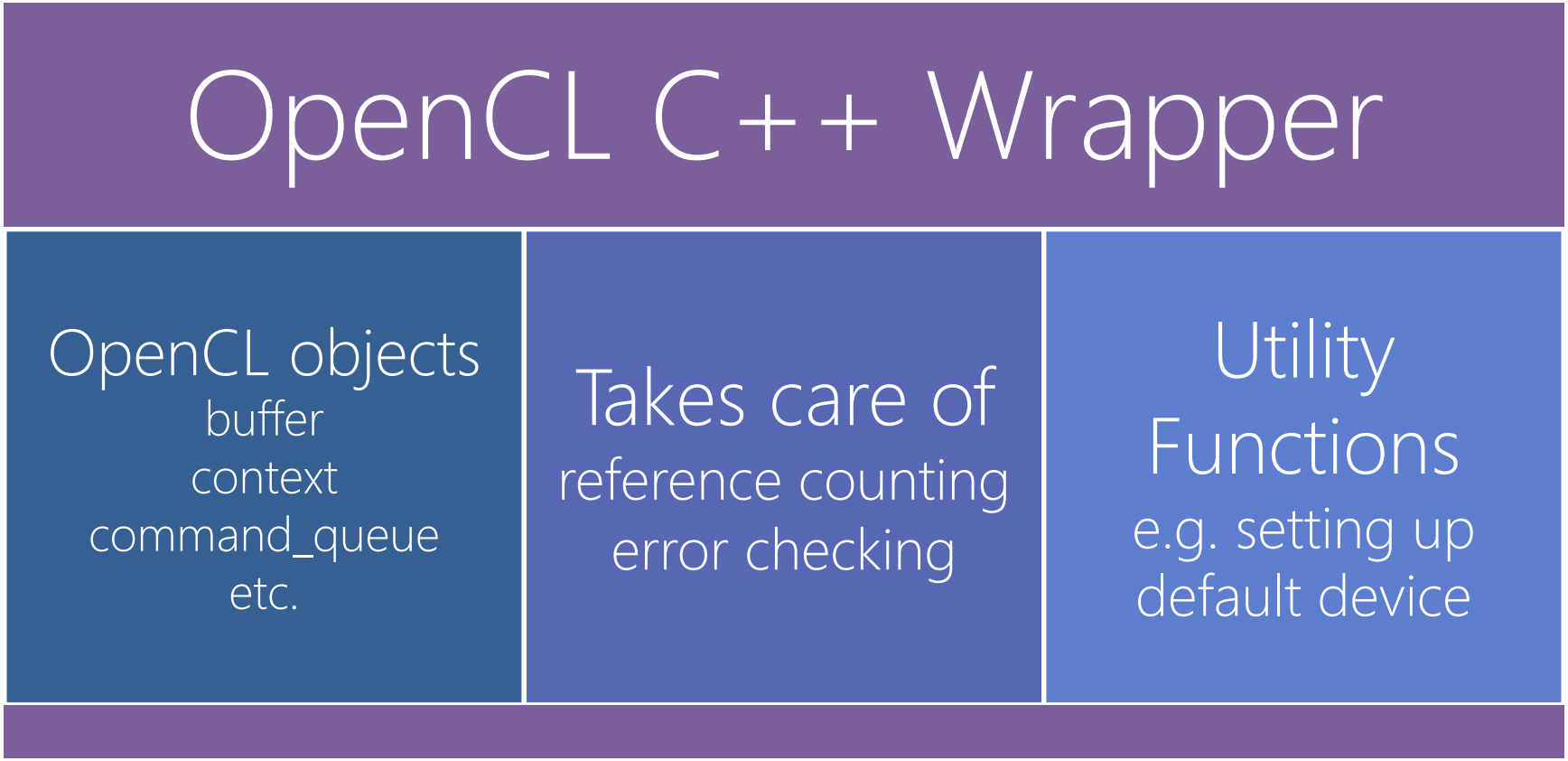
Boost.Compute



Library Architecture



Low-Level API



Low-Level API

```
for (auto const& device : boost::compute::system::devices())
    std::cout << "device = " << device.name() << std::endl;
```

```
// query number of opencl platforms
cl_uint num_platforms = 0;
cl_int ret = clGetPlatformIDs(0, NULL, &num_platforms);
if(ret != CL_SUCCESS){
    std::cerr << "failed to query platforms: " << ret << std::endl;
    return -1;
}

// check that at least one platform was found
if(num_platforms == 0){
    std::cerr << "found 0 platforms" << std::endl;
    return 0;
}

// get platform ids
cl_platform_id *platforms = new cl_platform_id[num_platforms];
clGetPlatformIDs(num_platforms, platforms, NULL);

// iterate through each platform and query its devices
for(cl_uint i = 0; i < num_platforms; i++){
    cl_platform_id platform = platforms[i];

    // query number of opencl devices
    cl_uint num_devices = 0;
    ret = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 0, NULL, &num_devices);
    if(ret != CL_SUCCESS){
        std::cerr << "failed to lookup devices for platform " << i << std::endl;
        continue;
    }

    // print number of devices found
    std::cout << "platform " << i << " has " << num_devices << " devices:" << std::endl;

    // get device ids for the platform
    cl_device_id *devices = new cl_device_id[num_devices];
    ret = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, num_devices, devices, NULL);
    if(ret != CL_SUCCESS){
        std::cerr << "failed to query platform devices" << std::endl;
        delete[] devices;
        continue;
    }

    // iterate through each device on the platform and print its name
    for(cl_uint j = 0; j < num_devices; j++){
        cl_device_id device = devices[j];

        // get length of the device name string
        size_t name_length = 0;
        ret = clGetDeviceInfo(device, CL_DEVICE_NAME, 0, NULL, &name_length);
        if(ret != CL_SUCCESS){
            std::cerr << "failed to query device name length for device " << j << std::endl;
            continue;
        }

        // get the device name string
        char *name = new char[name_length];
        ret = clGetDeviceInfo(device, CL_DEVICE_NAME, name_length, name, NULL);
        if(ret != CL_SUCCESS){
            std::cerr << "failed to query device name string for device " << j << std::endl;
            delete[] name;
            continue;
        }

        // print out the device name
        std::cout << "  device: " << name << std::endl;

        delete[] name;
    }
    delete[] devices;
}
delete[] platforms;
```

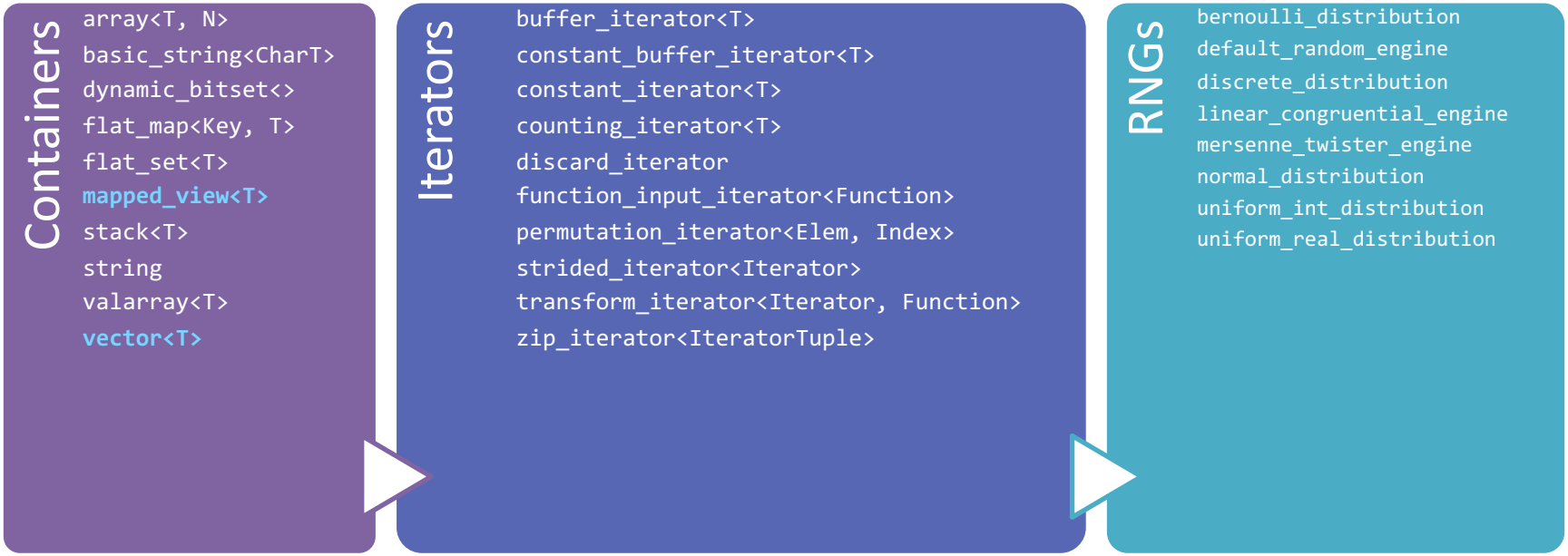
Low-Level API Example

```
#include <boost/compute/core.hpp>

// lookup default compute device
auto device = boost::compute::system::default_device();

// create OpenCL context for the device
auto ctx = boost::compute::context(device);

// get default command queue
auto queue = boost::compute::command_queue(ctx, device);

// print device name
std:cout << "device = " << device.name() << std::endl;
```

The default device is selected based on a set of heuristics and can be influenced using one of the following environment variables:

- **BOOST_COMPUTE_DEFAULT_DEVICE** – name of the compute device (e.g. "AMD Radeon")
- **BOOST_COMPUTE_DEFAULT_DEVICE_TYPE** – type of the compute device (e.g. "GPU" or "CPU")
- **BOOST_COMPUTE_DEFAULT_PLATFORM** – name of the platform (e.g. "NVIDIA CUDA")
- **BOOST_COMPUTE_DEFAULT_VENDOR** – name of the device vendor (e.g. "Intel")

High-Level API – Parallel STL



High-Level API – Parallel STL

Algorithms

- accumulate()
- adjacent_difference()
- adjacent_find()
- all_of()
- any_of()
- binary_search()
- copy()
- copy_if()
- copy_n()
- count()
- count_if()
- equal()
- equal_range()
- **exclusive_scan()**
- fill()
- fill_n()
- find()
- find_end()
- find_if()
- find_if_not()
- for_each()
- for_each_n()
- **gather()**
- generate()
- generate_n()
- includes()
- **inclusive_scan()**
- inner_product()
- inplace_merge()
- iota()
- is_partitioned()
- is_permutation()
- is_sorted()
- lower_bound()
- lexicographical_compare()
- max_element()
- merge()
- min_element()
- minmax_element()
- mismatch()
- next_permutation()
- none_of()
- nth_element()
- partial_sum()
- partition()
- partition_copy()
- partition_point()
- prev_permutation()
- random_shuffle()
- **reduce()**
- reduce_by_key()
- remove()
- remove_if()
- replace()
- replace_copy()
- reverse()
- reverse_copy()
- rotate()
- rotate_copy()
- scatter()
- search()
- search_n()
- set_difference()
- set_intersection()
- set_symmetric_difference()
- set_union()
- sort()
- **sort_by_key()**
- stable_partition()
- stable_sort()
- stable_sort_by_key()
- swap_ranges()
- transform()
- **transform_reduce()**
- unique()
- unique_copy()
- upper_bound()

Host Sort Example

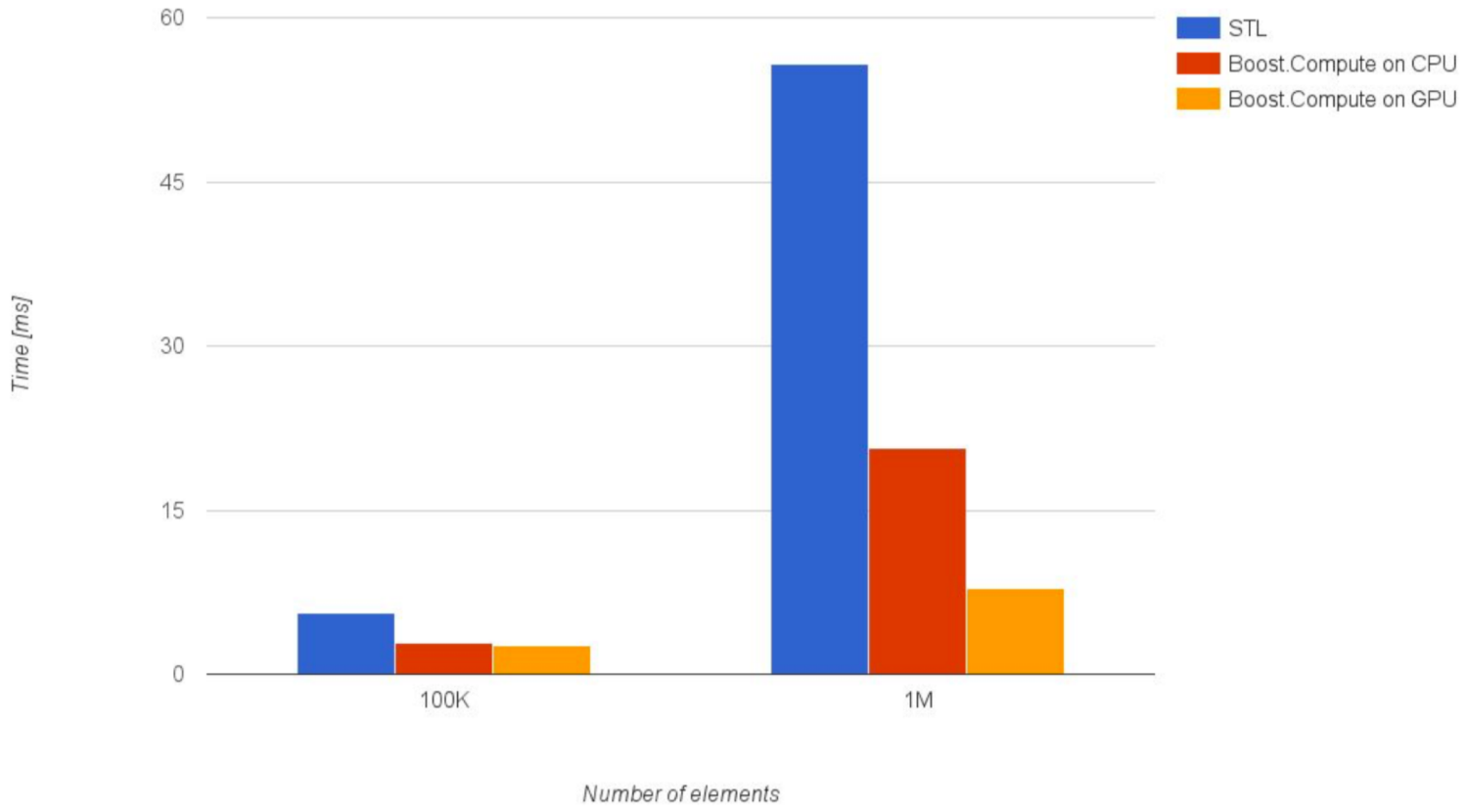
STL

```
#include <vector>
#include <algorithm>


std::vector<int> v{…};

// fill the vector with some data
std::sort(v.begin(), v.end());
```

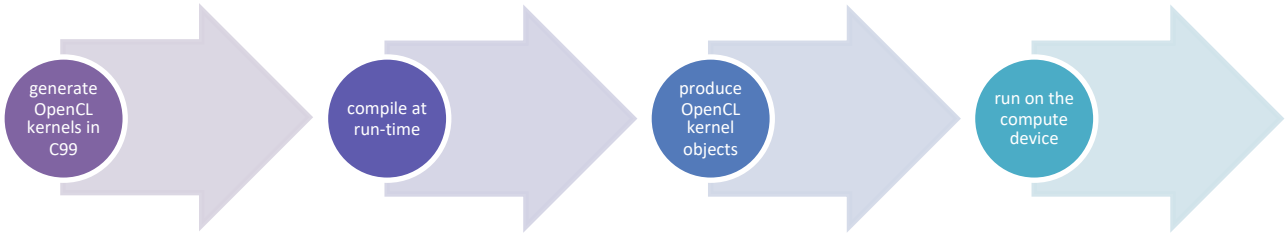
Boost.Compute

```
#include <boost/compute/algorithm.hpp>
//#include <boost/compute/algorithm/sort.hpp>
#include <vector>

std::vector<int> v{…};

// fill the vector with some data
boost::compute::sort(v.begin(), v.end(), queue);
```

Host Sort Example



Algorithm Internals



C++

```
boost::compute::reduce(
    data.begin(), data.end(), &sum, queue
);
```

OpenCL

```
__kernel void reduce(__global int* input,
                     const uint offset,
                     const uint count,
                     __global int* output,
                     const uint output_offset)
{
    const uint block_offset = get_group_id(0) * VPT * TPB;
    __global const int *block = input + offset + block_offset;
    const uint lid = get_local_id(0);
    __local int scratch[TPB];
    int sum = 0;
    for(uint i = 0; i < VPT; i++){
        if(block_offset + lid + i*TPB < count){
            sum = sum + block[lid+i*TPB];
        }
    }
    scratch[lid] = sum;
    for(int i = 1; i < TPB; i <<= 1){
       barrier(CLK_LOCAL_MEM_FENCE);
       uint mask = (i << 1) - 1;
       if((lid & mask) == 0){
           scratch[lid] += scratch[lid+i];
       }
    }
    if(lid == 0){
        output[output_offset + get_group_id(0)] = scratch[0];
    }
}
```

Custom Functions

```
BOOST_COMPUTE_FUNCTION(int, add_three, (int x),
{
      return x + 3;
});

boost::compute::vector<int> vector = { ... };

boost::compute::transform(
      vector.begin(),
      vector.end(),
      vector.begin(),
      add_three,
      queue
);
```

- `BOOST_COMPUTE_FUNCTION()` macro produces Boost.Compute function object
- Body of the function is checked during OpenCL compilation

Lambda Expressions

- Not the same as C++11 lambdas
- Easy way for specifying custom function for algorithms
- Fully type-checked by the C++ compiler

```
using boost::compute::_1;

boost::compute::vector<int> vec = { ... };
boost::compute::transform(vec.begin(), vec.end(), vec.begin(), _1 + 5, queue);
```

Boost.Compute Tutorial

SHARCNET™

Available on SHARCNet GitLab

https://git.sharcnet.ca/asobhani/bc-tutorial