TDDD56
Lab Lesson 2
Lab 3: Skeleton Programming with SkePU

Sehrish Qummar

sehrish.qummar@liu.se

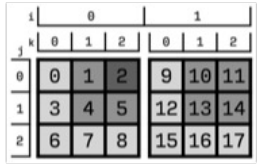
Lab Schedule

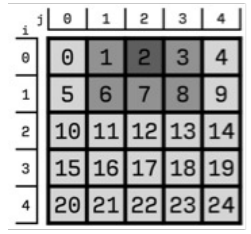
	WebReg	Week	
CPU	Lab 1	v45	Load Balancing
	Lab 2	v46	Non-Blocking Data Structures
	Lab 3	v47	High level parallel programming
GPU	Lab 4	v48	CUDA 1
	Lab 5	v49	CUDA 2
	Lab 6	v50	OpenCL

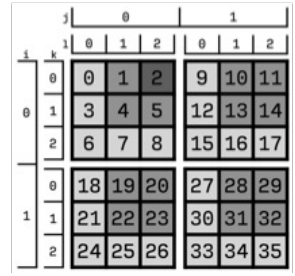
Lesson 2

SkePU

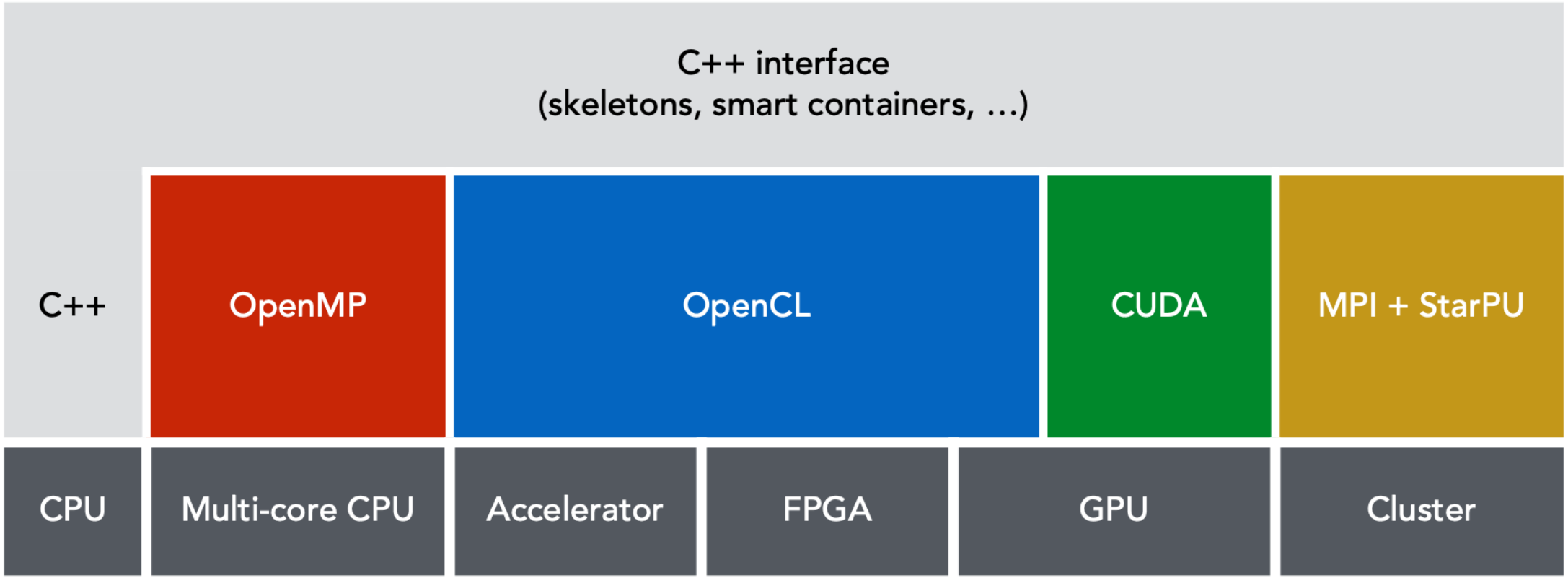
- Skeleton programming framework
  - C++11 **library** with skeleton and data container classes
  - A source-to-source pre-compiler
- Smart containers:
  Vector<T>, Matrix<T>, Tensor3<T>, Tensor4<T>
- For **heterogeneous multicore** systems and clusters
  - Multiple backends with dynamic backend selection
- Active research tool (A good topic for your thesis)





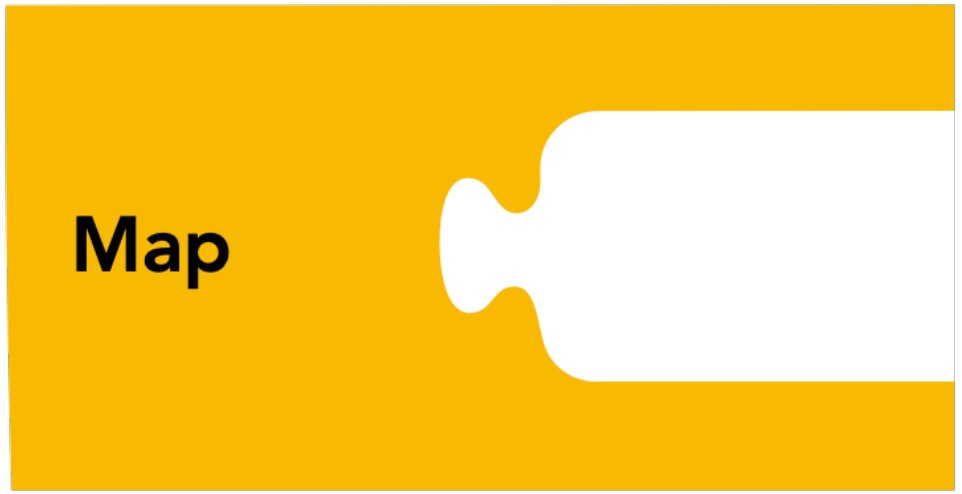


SkePU


C++ interface
(skeletons, smart containers, …)
C++
OpenMP
OpenCL
CUDA
MPI + StarPU
CPU
Multi-core CPU
Accelerator
FPGA
GPU
Cluster

SkePU Skeletons

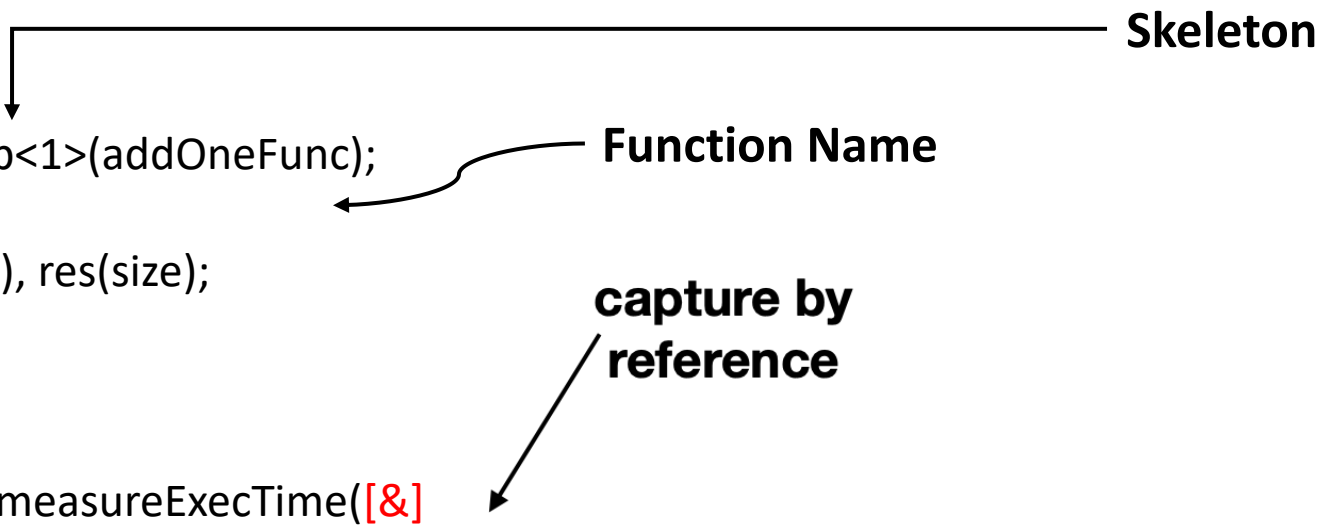
- Parametrizable higher-order functions implemented as C++ template classes
  - **Map**
  - **Reduce**
  - **MapReduce**
  - **MapOverlap**
  - MapPairs
  - MapPairsReduce
  - Scan



C++11

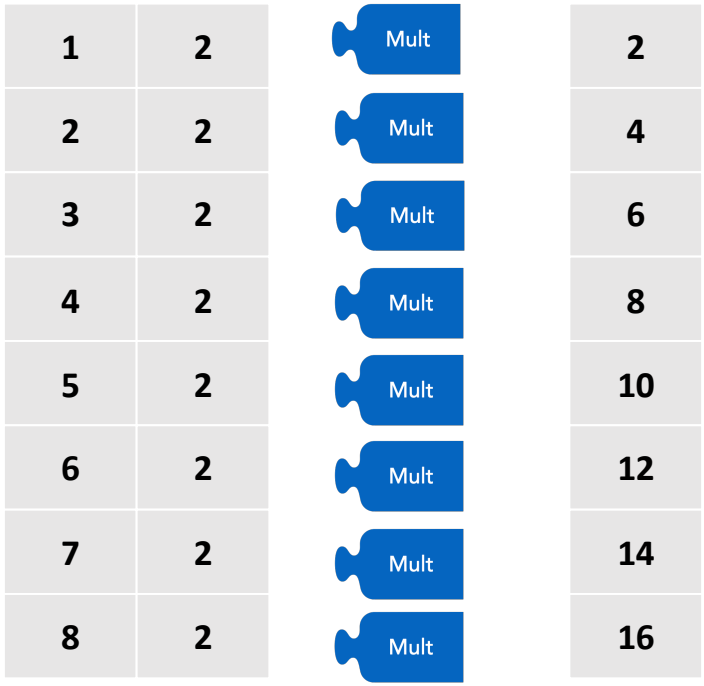
- Shift in the labs from C to C++11 (”modern” C++)

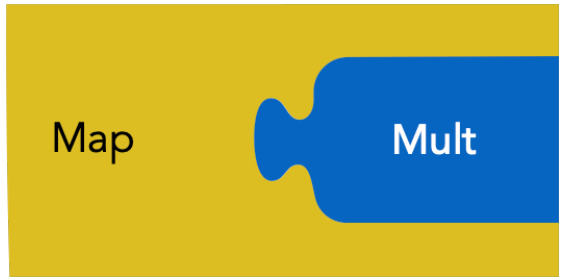
```
// ”auto” type specifier
auto addOneMap = skepu::Map<1>(addOneFunc);

skepu::Vector<float> input(size), res(size);
input.randomize(0, 9);

// Lambda expression
auto dur = skepu::benchmark::measureExecTime([&]
{
        addOneMap(res, input);
});
```



SkePU Skeletons

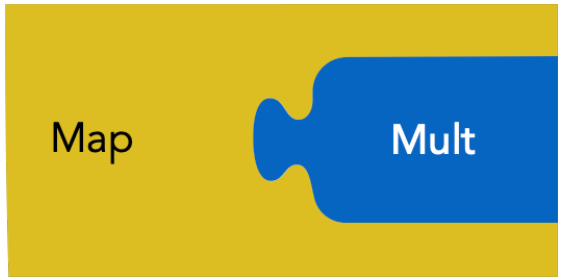
Sequential algorithm

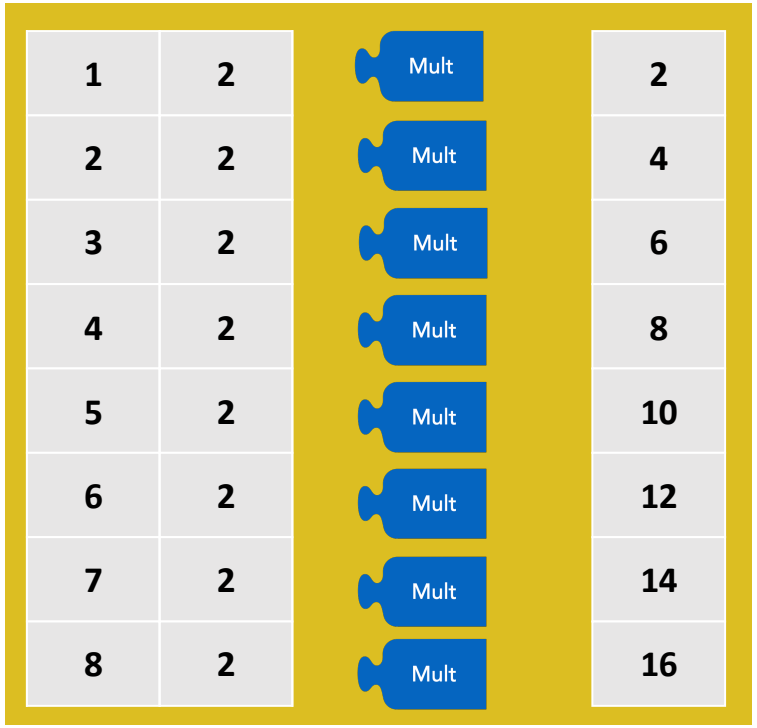




SkePU Skeletons

Parallel algorithm



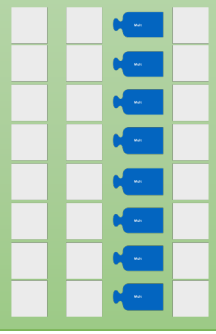


SkePU syntax

```
int add(int a, int b)
{
        return a + b;
}
```


```
auto vec_sum = Map<2>(add);
```


```
vec_sum(result, v1, v2);
```



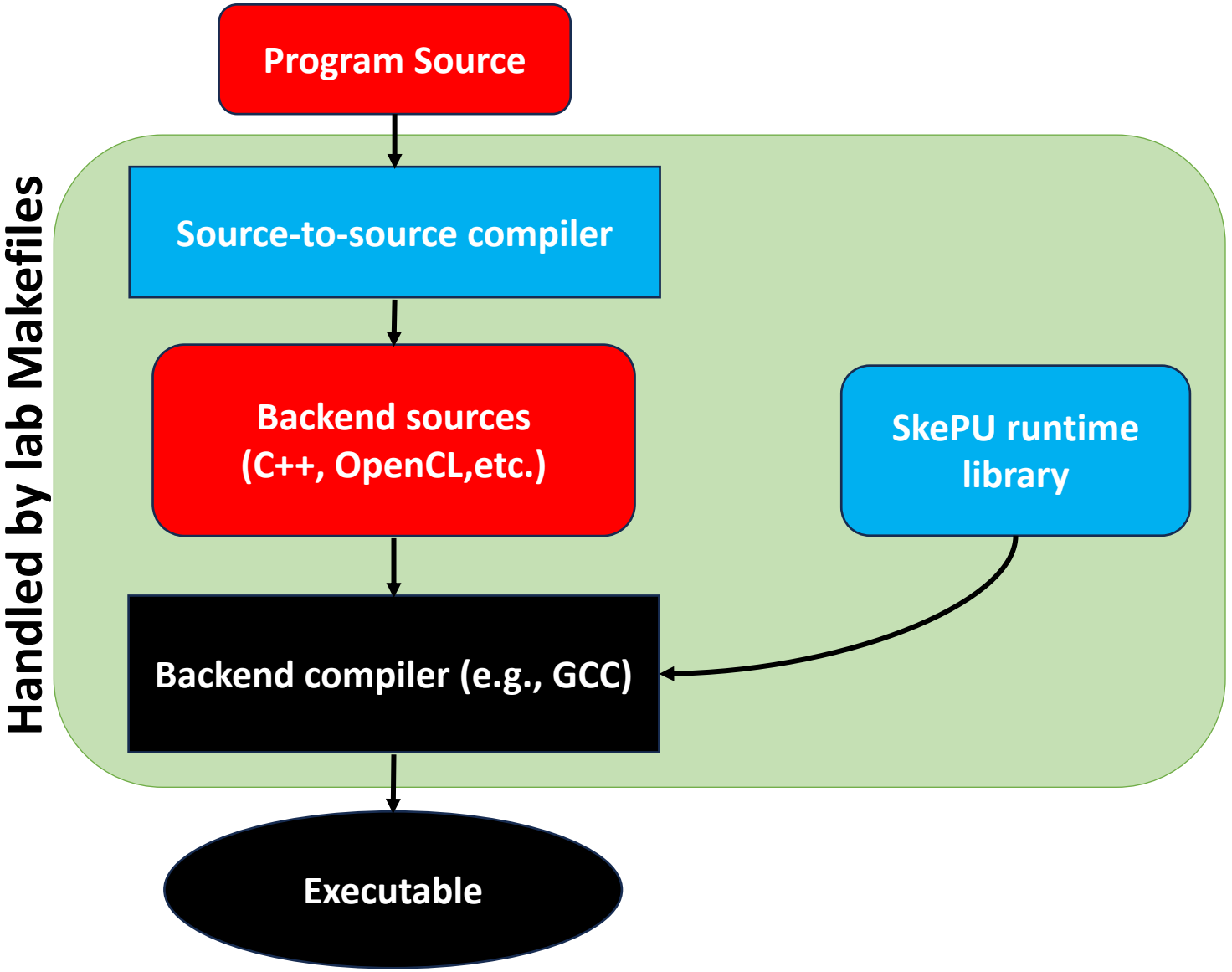
SkePU syntax, advanced

```
template<typename T>
T abs(T input)
{
  return input < 0 ? -input : input;
}

template<typename T>
T userfunc(Index1D row, const Mat<T> m, const Vec<T> v)
{
  T res = 0;
  for (size_t i = 0; i < v.size; ++i)
    res += m(row.i, i) * v(i);

  return abs(res);
}
```

SkePU containers

- **Smart** containers: Vector<T>, Matrix<T>, etc
- Manages data across CPU and GPU
- No data transfers unless necessary (lazy copying)
- Keeps track of most recent writes
  - Memory consistency through software

SkePU build process



Lab structure

- Three exercises:
  - Warm-up: dot product
  - Averaging image filter + gaussian filter
  - Median filter

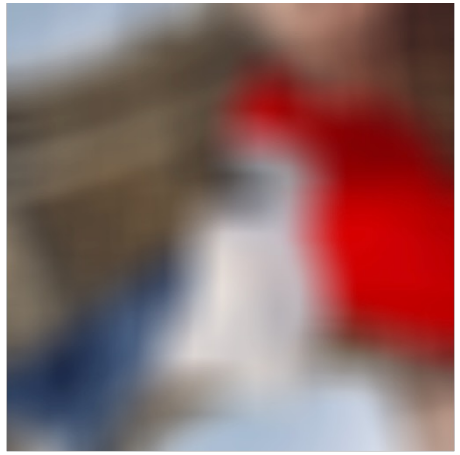
1. Dot product

- Implement two variants of dot product:
  - With **MapReduce** skeleton
  - With **Map** + **Reduce** skeletons
- Compare and contrast the variants
  - Why does SkePU have the MapReduce skeleton?
- Measure with different backends and problem sizes

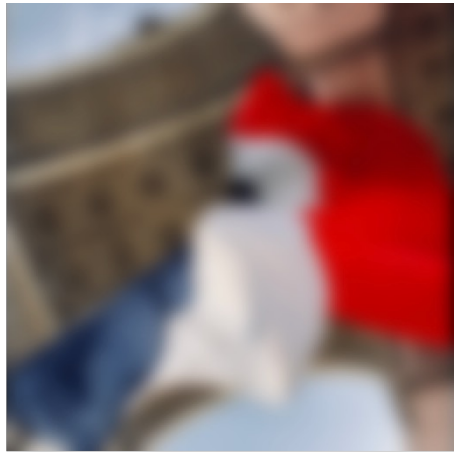
2. Averaging filters

- Averaging filter: find average color value in surrounding region
- Gaussian filter: averaging filter with **non-uniform** weights
- Use the MapOverlap skeleton

Original



Average



Gaussian

3. Median filter

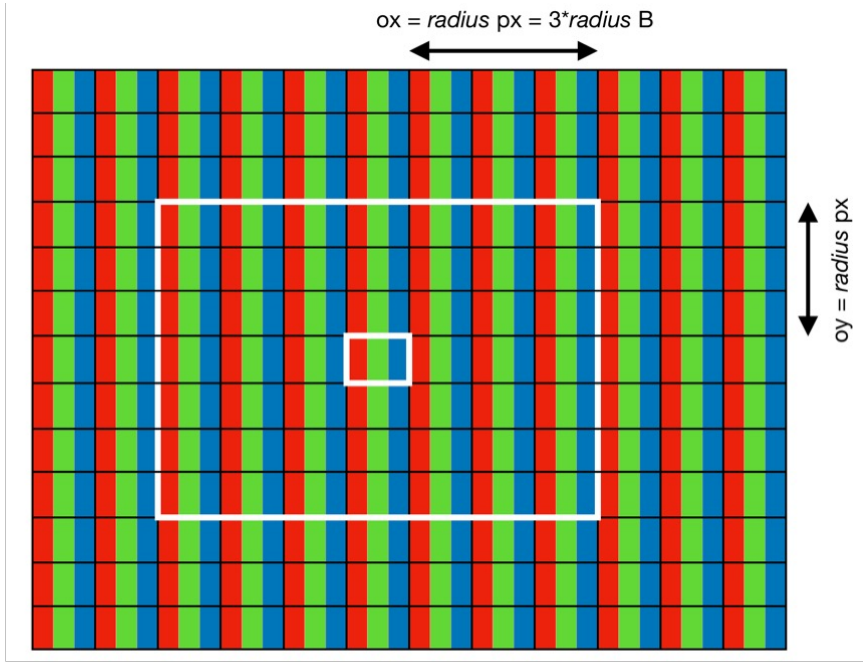
- Median filter: find **median** color value in surrounding region
- Requires sorting the pixel values in some way

Original

Median

Image filter

- Layout of image data in memory



1 pixel = 3 bytes!

Lab installation

- Get from course website as usual
- Slightly different from public SkePU distribution!
  - Pre-built binary
  - Runs on 64-bit Linux

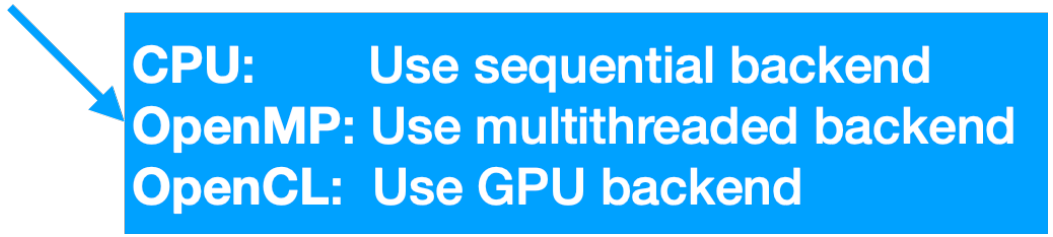
Lab build process

Build lab program:

```
> make bin/addone
```

Run lab program:

```
> bin/addone 100 CPU
```



A warning about warnings (and errors)

- SkePU is a C++ template library
- As such, gets very long and unreadable diagnostic messages if used incorrectly!
- Following the structure of the lab files should minimize errors
- Otherwise, be careful, and avoid using const!

Questions?