

TDDDD56

Lab 3: Skeleton programming with SkePU

August Ernstsson

august.ernstsson@liu.se

Labs schedule

| | WebReg | Week | | |
|-------------------------|--------|-------|-----|---|
| Responsible: August | CPU | Lab 1 | v46 | Load Balancing |
| | | Lab 2 | v47 | Non-blocking Data Structures |
| | | Lab 3 | v49 | High-level Parallel Programming Lesson 2 |
| Responsible: Ingemar | GPU | Lab 4 | v48 | CUDA 1 |
| | | Lab 5 | v50 | CUDA 2 |
| | | Lab 6 | v51 | OpenCL |

C++11

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

```
// "auto" type specifier
```

```
auto addOneMap = skepu2::Map<1>(addOneFunc);
```

```
skepu2::Vector<float> input(size), res(size);
```

```
input.randomize(0, 9);
```

```
// Lambda expression
```

```
auto dur = skepu2::benchmark::measureExecTime([&]
```

```
{
```

```
    addOneMap(res, input);
```

```
});
```

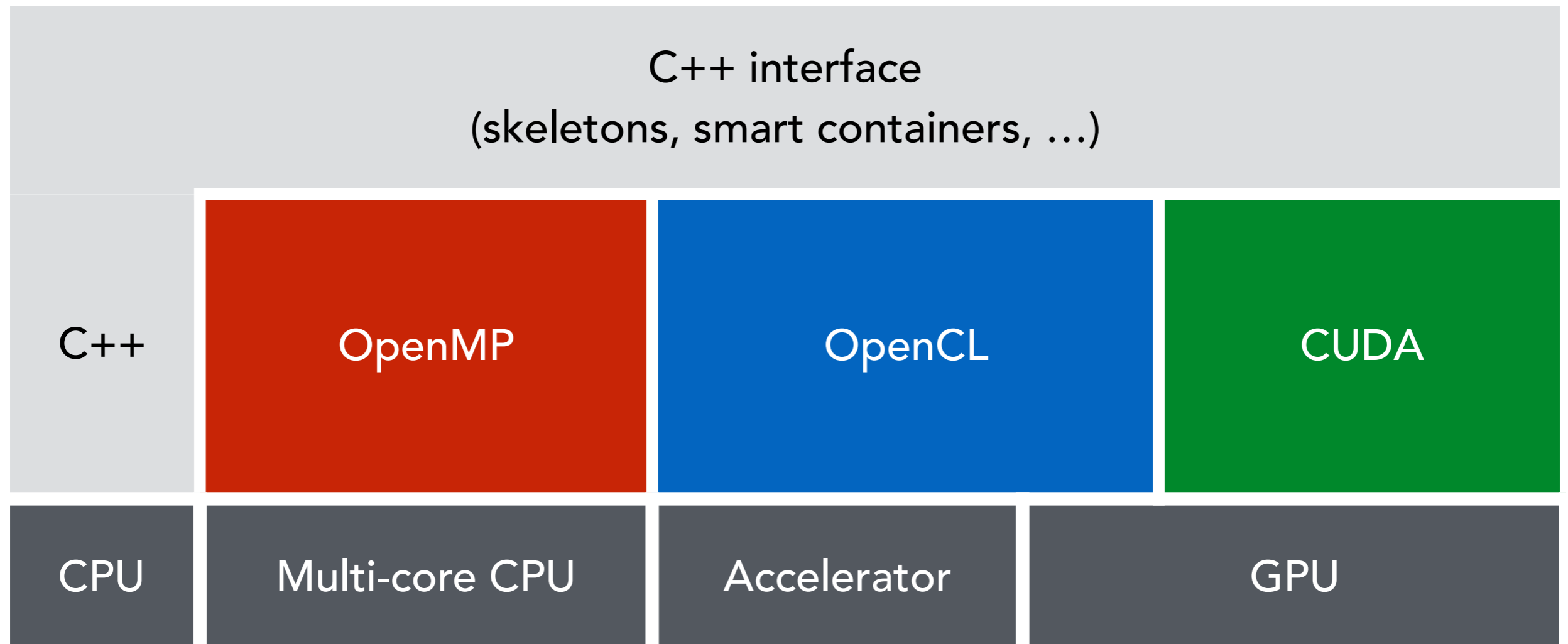
capture by
reference



SkePU

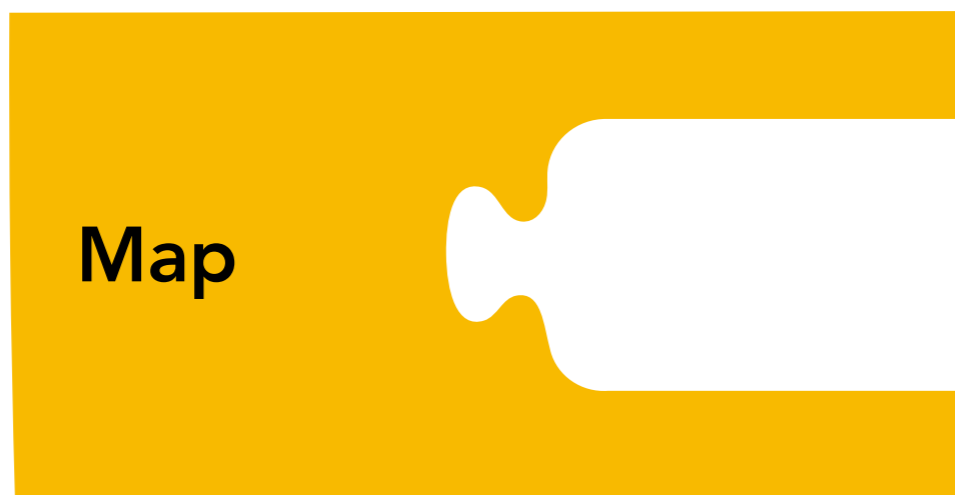
- Skeleton programming framework
 - C++11 **library** with skeleton and data container classes
 - A source-to-source **translator tool**
- Smart containers: `Vector<T>`, `Matrix<T>`
- For **heterogeneous multicore** systems
 - Multiple backends
- Active research tool (A good topic for your thesis?)

SkePU architecture



SkePU skeletons

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



SkePU skeletons

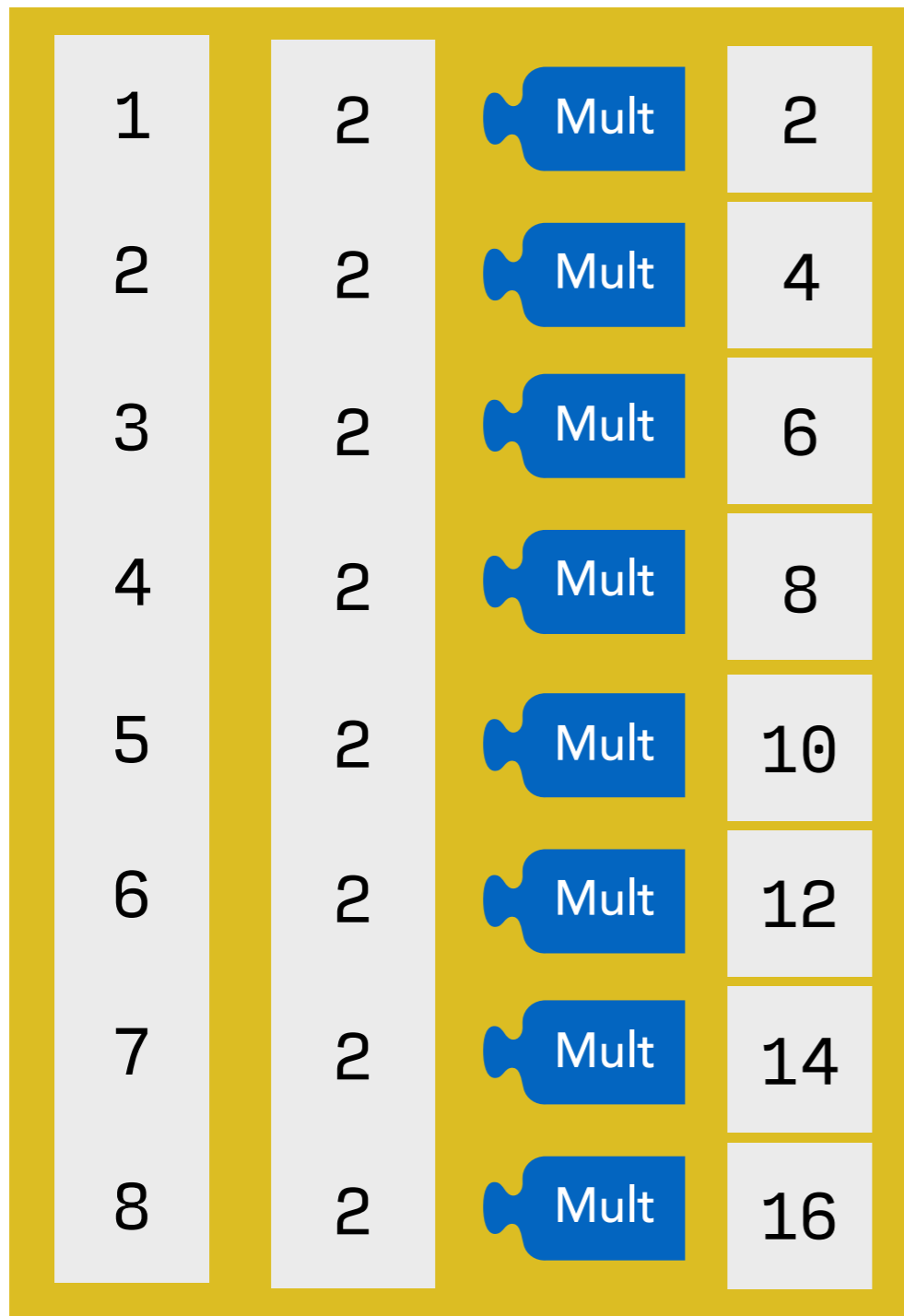
Sequential algorithm

| | | | |
|---|---|------|----|
| 1 | 2 | Mult | 2 |
| 2 | 2 | Mult | 4 |
| 3 | 2 | Mult | 6 |
| 4 | 2 | Mult | 8 |
| 5 | 2 | Mult | 10 |
| 6 | 2 | Mult | 12 |
| 7 | 2 | Mult | 14 |
| 8 | 2 | Mult | 16 |



SkePU skeletons

Parallel algorithm

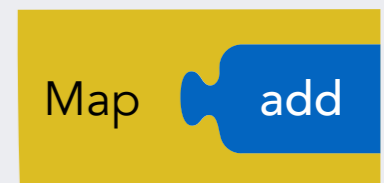


SkePU syntax

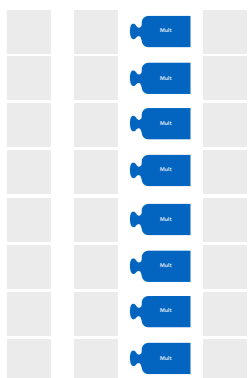
```
int add(int a, int b, int m)
{
    return (a + b) % m;
}
```



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



```
vec_sum(result, v1, v2, 5);
```



SkePU syntax, advanced

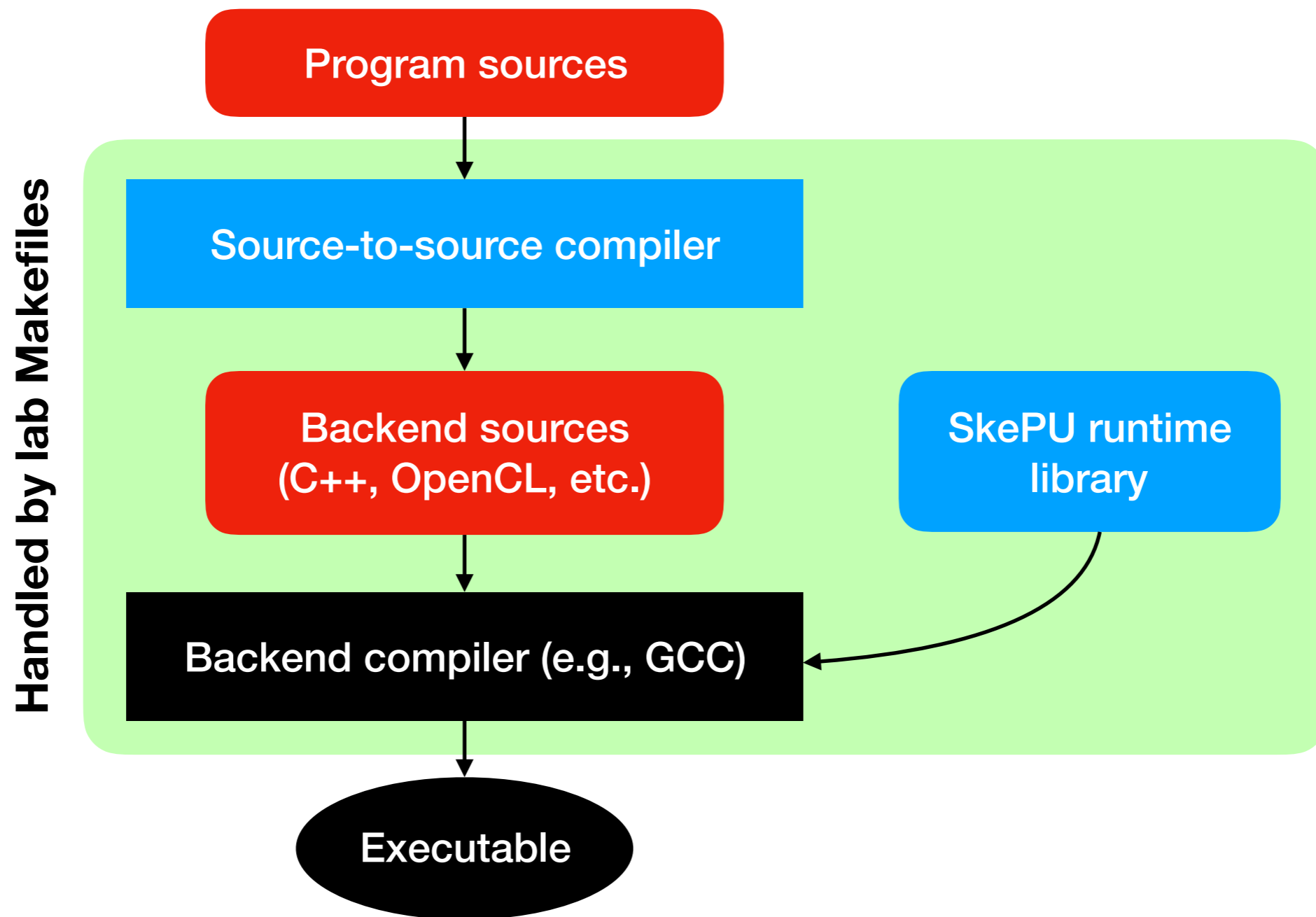
```
template<typename T>  
T abs(T input)  
{  
    return input < 0 ? -input : input;  
}
```

```
template<typename T>  
T mvmult(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 * m.cols + i] * v[i];  
  
    return abs(res);  
}
```

SkePU containers

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

SkePU build process



Lab structure

- Three exercises:
 1. Warm-up: dot product
 2. Averaging image filter + gaussian filter
 3. Median filter

1. Dot product

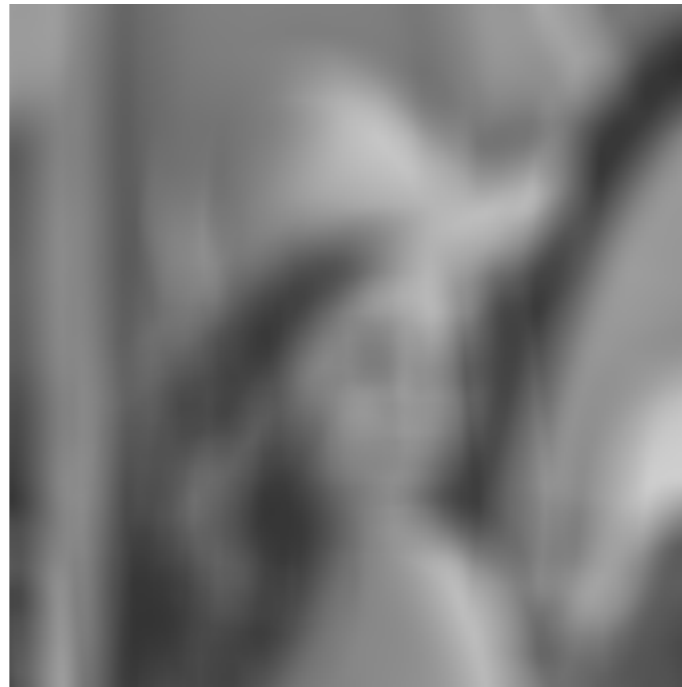
- Implement two variants of dot product:
 - With **MapReduce** skeleton
 - With **Map + Reduce** skeletons
- Compare and contrast the variants
 - Why does SkePU have the MapReduce skeleton?
- Measure with different backends and problem sizes

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



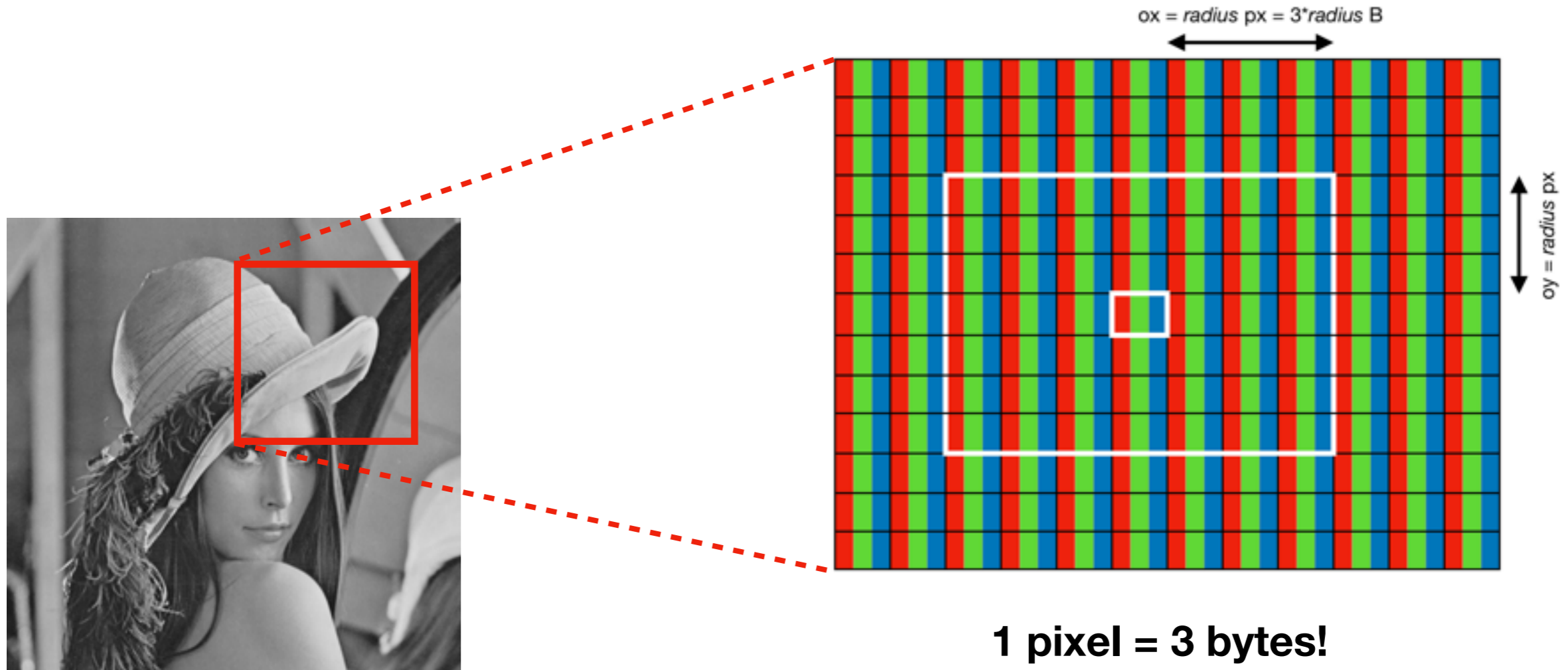
Original



Median

Image filters

- Layout of image data in memory




Lab build process

Build lab program:

```
> make bin/addone
```

Run lab program:

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



CPU: Use sequential backend
OpenMP: Use multithreaded backend
OpenCL: Use GPU backend

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?