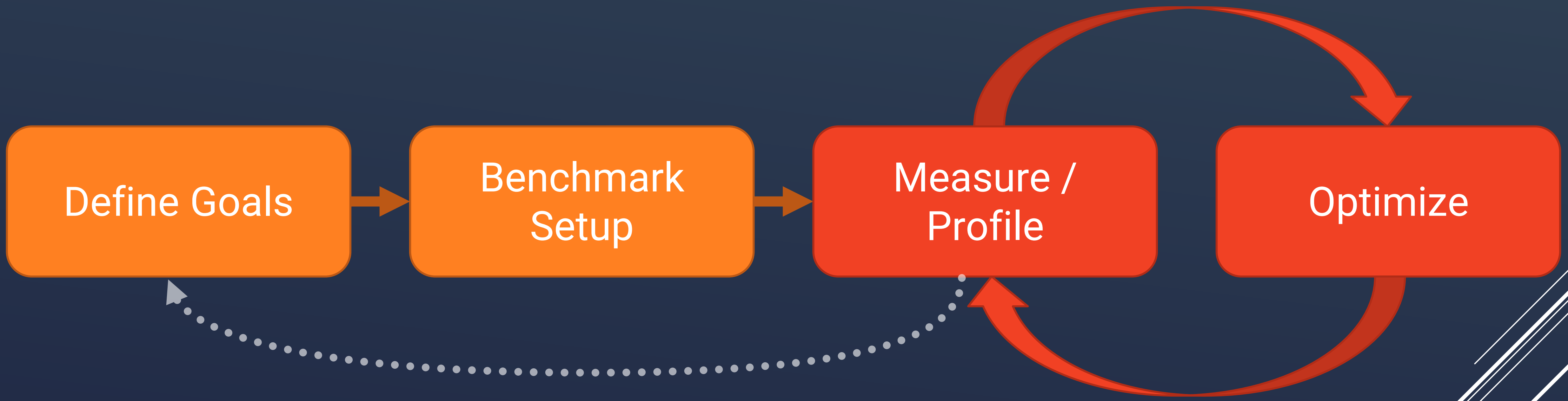
PERFORMANCE-ORIENTED COMPUTING

Optimization (0) - Overview

GOALS

- Before we get to the individual optimization topics, provide some general **definitions**
- Understand what “Optimization” **means**
- Be able to **classify optimizations** into 4 main categories

PERFORMANCE-ORIENTED DEVELOPMENT



- Optimization is a **cyclic** process, interleaving measuring / profiling with “actual” optimization work
- This aims to avoid *premature* or *ineffectual* optimization
  - We have the measurements to know **where** to effectively spend effort **before** we do so

DEFINING “OPTIMIZATION”

OPTIMIZATION – DEFINITION

- Exact definition varies with the Context of the optimization
- In the compiler context:

"Transformation of a program A into a program B, which has an identical input/output behavior but improves non-functional aspects."

- Examples:
  - Redundant subexpression elimination
  - Loop-invariant code motion
  - Lots of other "classic" compiler optimizations

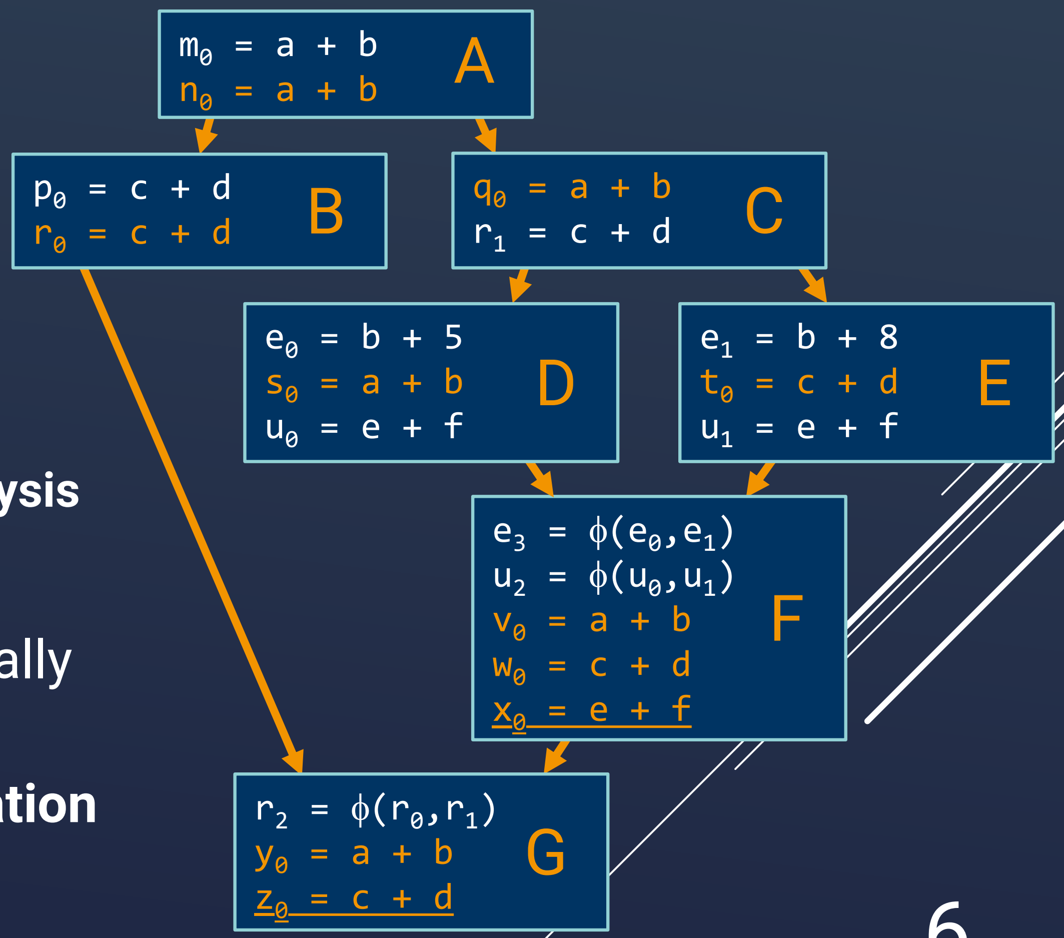
- This definition is very **safe**, but limits the space for optimizations

SIDENOTE:
COMMON SUBEXPRESSION ELIMINATION

*An expression **x (op) y** is redundant **if and only if**, along every path from the procedure's entry, it has been evaluated, and its constituent subexpressions (**x** & **y**) have **not** been redefined.*

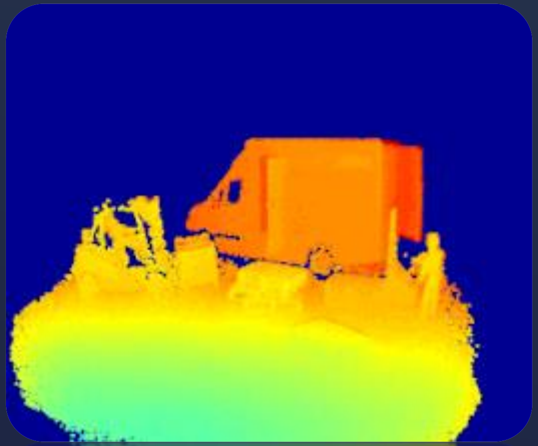
- *Example*: colored expressions are redundant
  - Some of these are relatively easy to find using **local** methods, while others require expensive **global analysis**
- Difficulty of applying optimization affects its utility
- Sometimes, performing this transformation is actually *slower*

→ Both of these points **also apply to manual optimization** efforts to some extent!



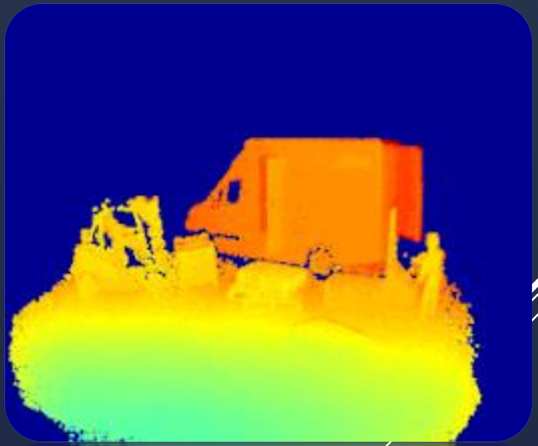
LIMITS OF STRICT DEFINITIONS

- Compilers need a **strict** definition, they cannot judge if a change is "important"
- However, in practice, such a definition would **limit** our optimization space
- Changes in output may be irrelevant for **actual purpose**
  - *Example*: Distance calculation in computer vision



Result, FP32

Result, FP16



→ Different results (bitwise), but equivalent in practice!

ALTERNATIVE DEFINITION

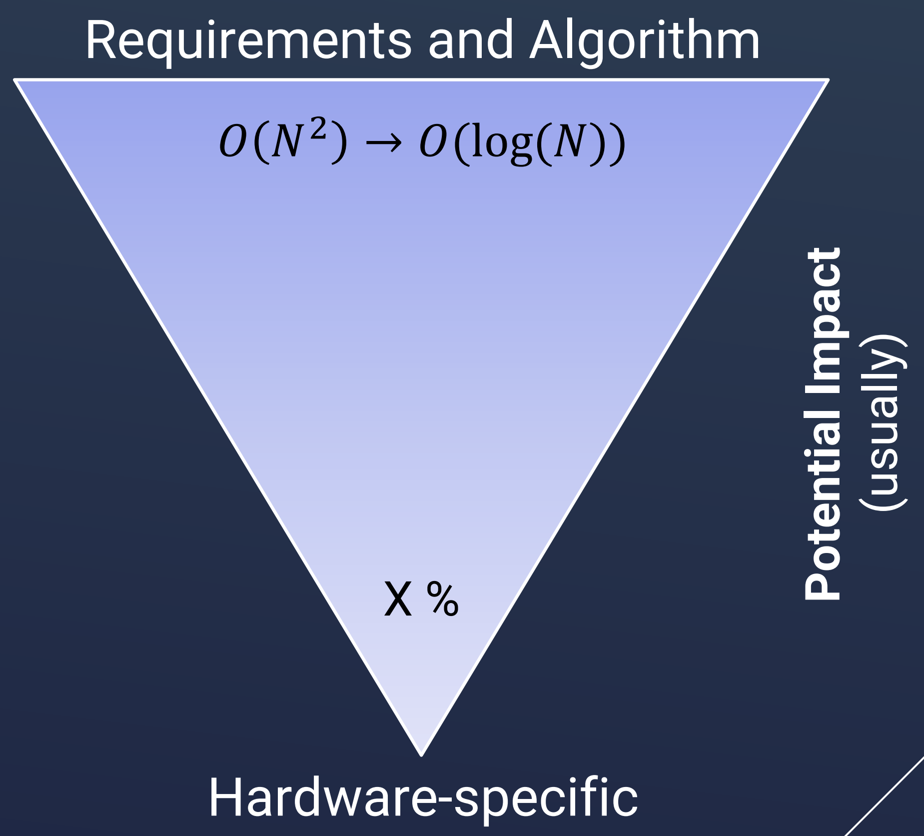
“Arbitrary changes in a program that improve non-functional aspects and still meet all other functional requirements.”

- I.e. exact equivalence on the individual bit level is **not** necessary
- Instead: Continued fulfillment of the specified **requirements**
  - Functional requirements on our program should be defined a priori (standard software engineering practices)
- *Example*: "Distance accuracy of the result is 1mm at a range of 10m".

CATEGORIES OF OPTIMIZATION

CATEGORIES OF OPTIMIZATION

1. **Adaptation of requirements**
2. **Algorithmic** optimization
3. **Procedural** optimization
4. **Hardware-specific** optimization



ADAPTATION OF REQUIREMENTS

- Improving performance or reducing resource consumption by **adapting requirements**
- The goal is to identify **non-essential requirements** that have a large impact on performance (compared to their necessity)
- Applicability is very application specific, and must be done in coordination with **domain experts**
- In contrast to other forms of optimization, **usually not objectively measurable and assessable**
  - Performance improvement can be measured, but not the influence on the overall quality – and thus the *meaningfulness* of the optimization

ADAPTATION OF REQUIREMENTS
- EXAMPLE -

- Alpha-blending has significant performance impact on Nintendo Switch (memory bandwidth), relatively minor visual impact

→ Requirement adaptation: **reduction of background vegetation**

ALGORITHMIC OPTIMIZATION

- **Most important** category of optimizations
  - By far the largest potential of possible performance improvements
  - Wide range of subcategories
- Includes all optimizations that achieve the same result in a *fundamentally different* way
- Algorithmic optimizations often also require **changes to data structures**

Potentially enables reduction in complexity Class!

(Usually not possible with procedural or HW-specific optimization)

ALGORITHMIC OPTIMIZATION
- (SILLY) EXAMPLE -

Sorting

- If a program uses a Bubblesort or other sub-optimal sorting algorithm
- Use Quicksort or other efficient sort instead
- $\boldsymbol{O}(\boldsymbol{N}^2) \rightarrow \boldsymbol{O}(\boldsymbol{N}\ \mathbf{log}(\boldsymbol{N}))$ complexity! (average case)

- This particular case is probably not a problem in most applications (nobody uses bubblesort)
- But general scenario extremely relevant!
  - Also in production software you frequently encounter ad-hoc solutions for problems for which there are **categorically better** algorithms

PROCEDURAL OPTIMIZATION

- Adaptations in the **implementation** of an **existing algorithm** to make the execution more efficient
- Distinction to Algorithmic Optimization not always 100% clear/binary
  - E.g. caching of existing results could be seen as procedural or algorithmic optimization

- Many of these are implemented as common compiler optimizations
  - Inlining, loop unrolling, …
  - Common Subexpression Elimination (seen before) is another example

PROCEDURAL OPTIMIZATION
- EXAMPLE -

- Loop-invariant code motion

```
for(int i=0;…) {
  …
  x[i] = y[i]
    + (a*sin(b));
  …
}
```

```
double c = a*sin(b);
for(int i=0;…) {
  …
  x[i] = y[i] + c;
  …
}
```

- Implemented in all modern compilers in some form
- Like with many of these basic optimizations, challenge today is determining when to actually perform it
  - Alternative/opposite is called **rematerialization**

HARDWARE-SPECIFIC OPTIMIZATION

2 basic categories:

1. Changing program flow and/or data structures to work *better* on certain hardware
2. Leveraging special hardware units or features

- Important difference:
  - **Category 1** *tunes* for certain target hardware, but remains generally executable
  - **Category 2** *restricts* the portability of the program

Affects *performance portability*

Affects plain old portability

HARDWARE-SPECIFIC OPTIMIZATION - EXAMPLES -

- For **category 1** – rather common: tuning for *cache hierarchies*
  - → we'll discuss this in detail in the next chapter!
- For **category 2**: Counting the set bits in a given integer **"n"**

Performance

Architecture	Latency	Throughput (CPI)
Icelake	3	1
Skylake	3	1
Broadwell	3	1
Haswell	3	1
Ivy Bridge	3	1

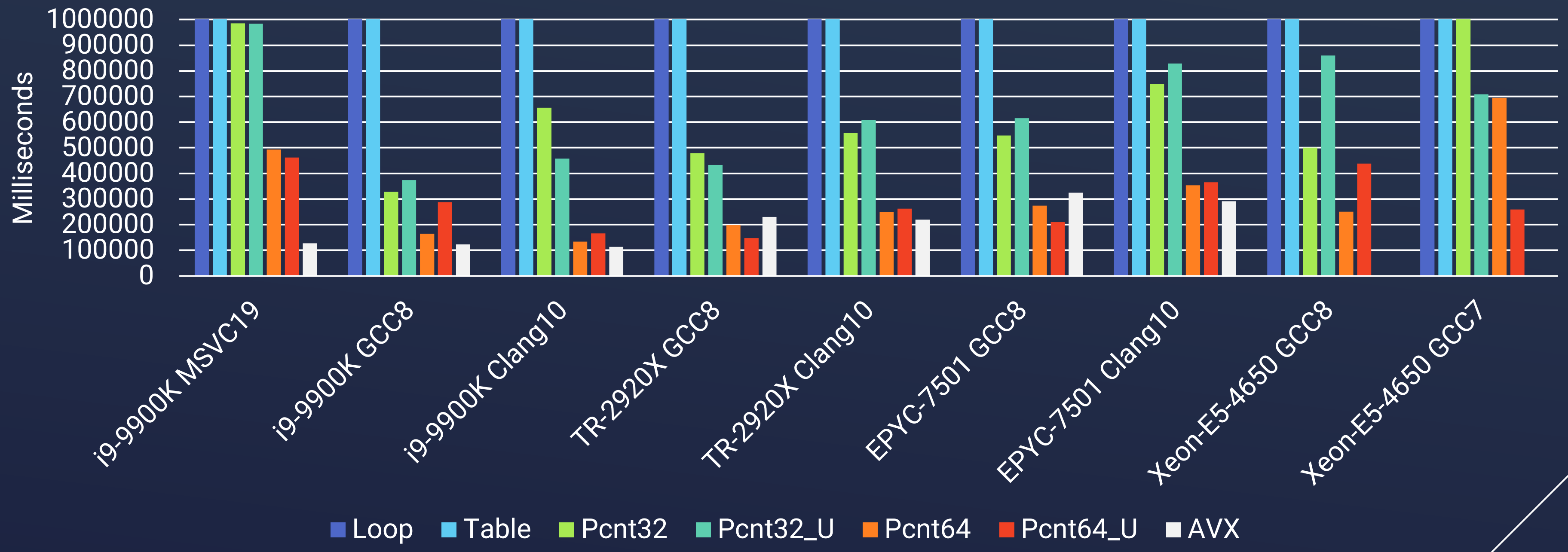
```
unsigned int count = 0;
while(n) {
  count += n & 1;
  n >>= 1;
}
```

```
unsigned int count =
  TBL[(n >>  0) & 0xff] +
  TBL[(n >>  8) & 0xff] +
  TBL[(n >> 16) & 0xff] +
  TBL[(n >> 24)])
```

```
unsigned int count =
        _mm_popcnt_u32(n);
```

HARDWARE-SPECIFIC OPTIMIZATION

- Performance of hardware-optimized code is very **hardware-dependent**
- For explicitly hardware optimized code this seems obvious, but **also** optimizations that are *not obviously hardware specific* can show this behavior



CATEGORIES OF OPTIMIZATION AND TOOL SUPPORT

1. **Adaptation of requirements**
2. **Algorithmic** optimization
3. **Procedural** optimization
4. **Hardware-specific** optimization

Items 1–2: Must largely be done **manually**

Items 3–4: Can be (at least partially) **automated**

It makes sense to familiarize yourself with the relevant compiler features of your respective development platform.
(Not only standard optimizations, but also e.g. vectorization).

CONCLUSION

APPLICABILITY OF OPTIMIZATIONS

- **Adaptation of requirements** and **algorithmic** optimizations are **largely independent** of programming languages and software/HW platforms
  - But requirements adaptation is very domain-specific
- **Procedural** and **hardware-focused** optimization differ depending on the programming environment and HW/SW stack

➔ We will primarily **focus on algorithmic optimization** in the remainder of this course!

SUMMARY

- The cycle of **performance-oriented development**
- **Definition** of “Optimization”
  - Strict compiler definition
  - More broadly applicable alternative & reasoning
- **Categories** of optimization
  - Adaptation of requirements
  - **Algorithmic optimization**
  - Procedural optimization
  - Hardware-specific optimization

QUESTIONS ?