



# Basic-block vectorization for graphics compilers

---

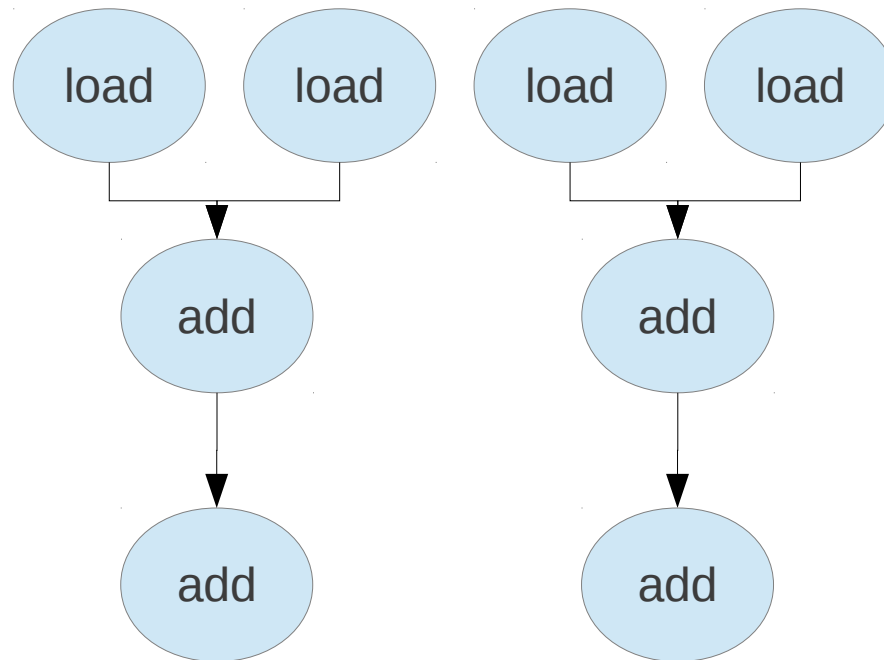
Robert Foss



# About vectorization

---

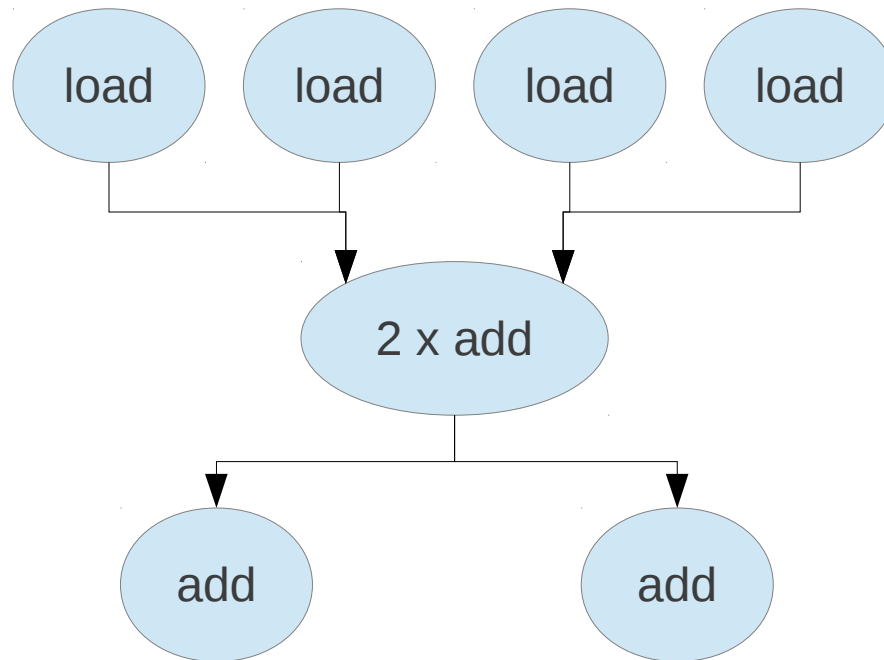
## Vectorization 101



# About vectorization

---

## Vectorization 101



# More about vectorization

---

## Techniques

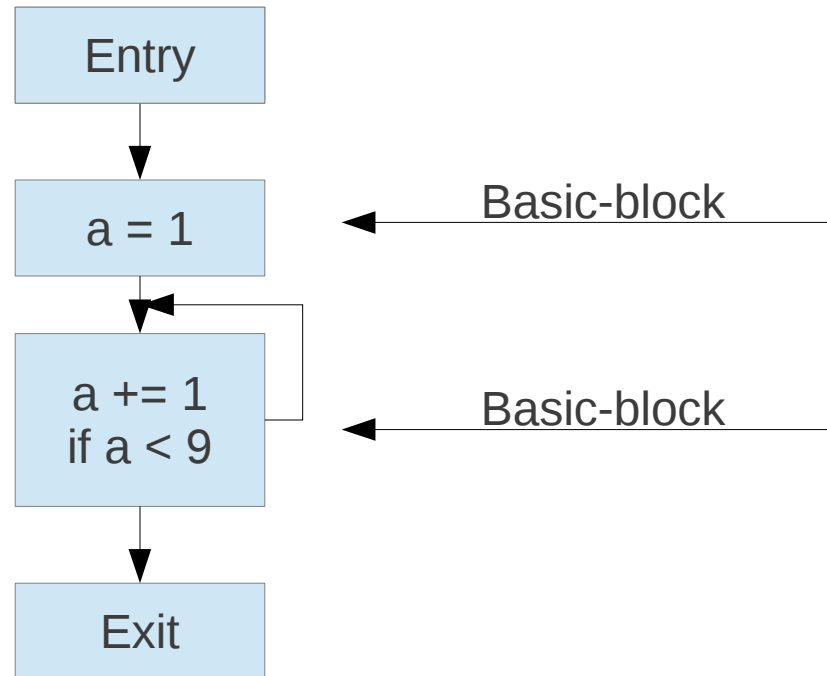
### Several techniques exist

- Loop-based vectorization
- Basic-block vectorization
- Superword Level Parallelism vectorization

# Basic-block vectorization

---

## Control Flow Graph



# Basic-block vectorization

---

## Basic-block

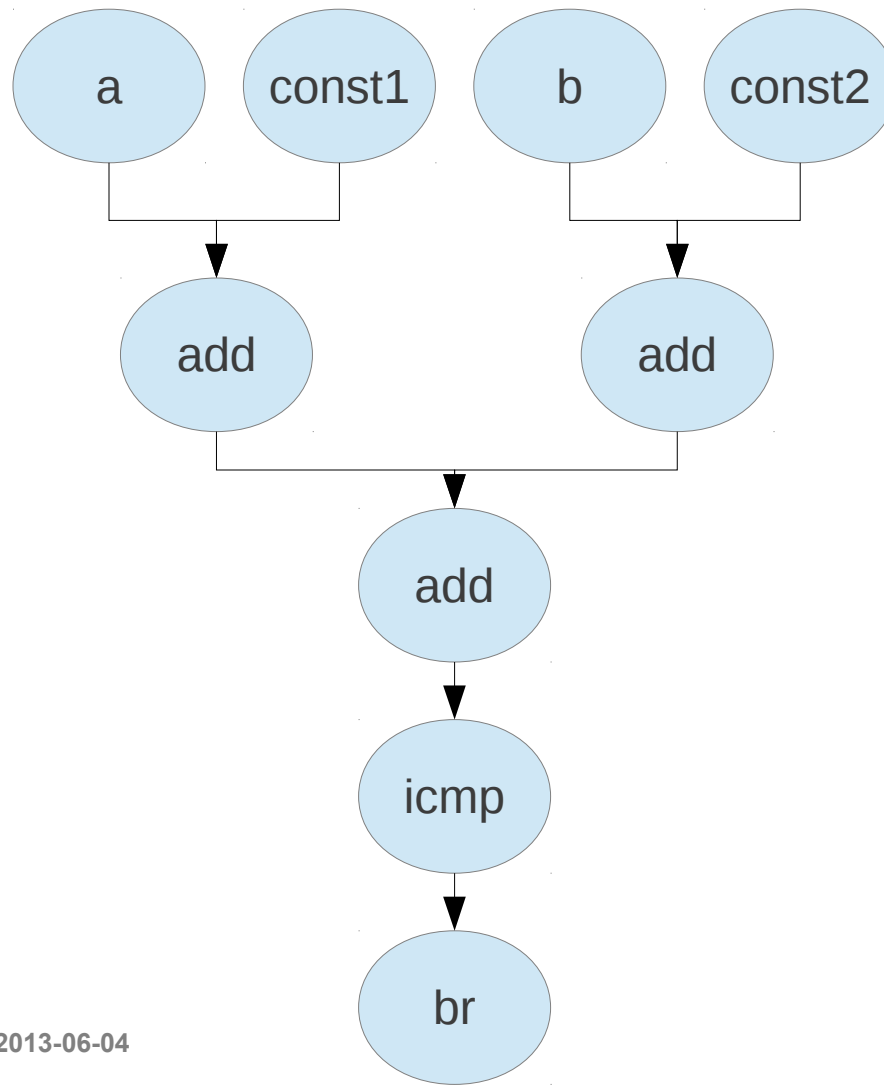
A diagram illustrating the mapping from a high-level basic block to its assembly representation. A small light blue box on the left contains high-level code, and a larger light blue box on the right contains the corresponding assembly code. Two lines connect the corners of the small box to the corners of the large box, indicating a one-to-one correspondence between the two representations.

```
a += 1  
b += 2  
c = a + b  
if c < 9
```

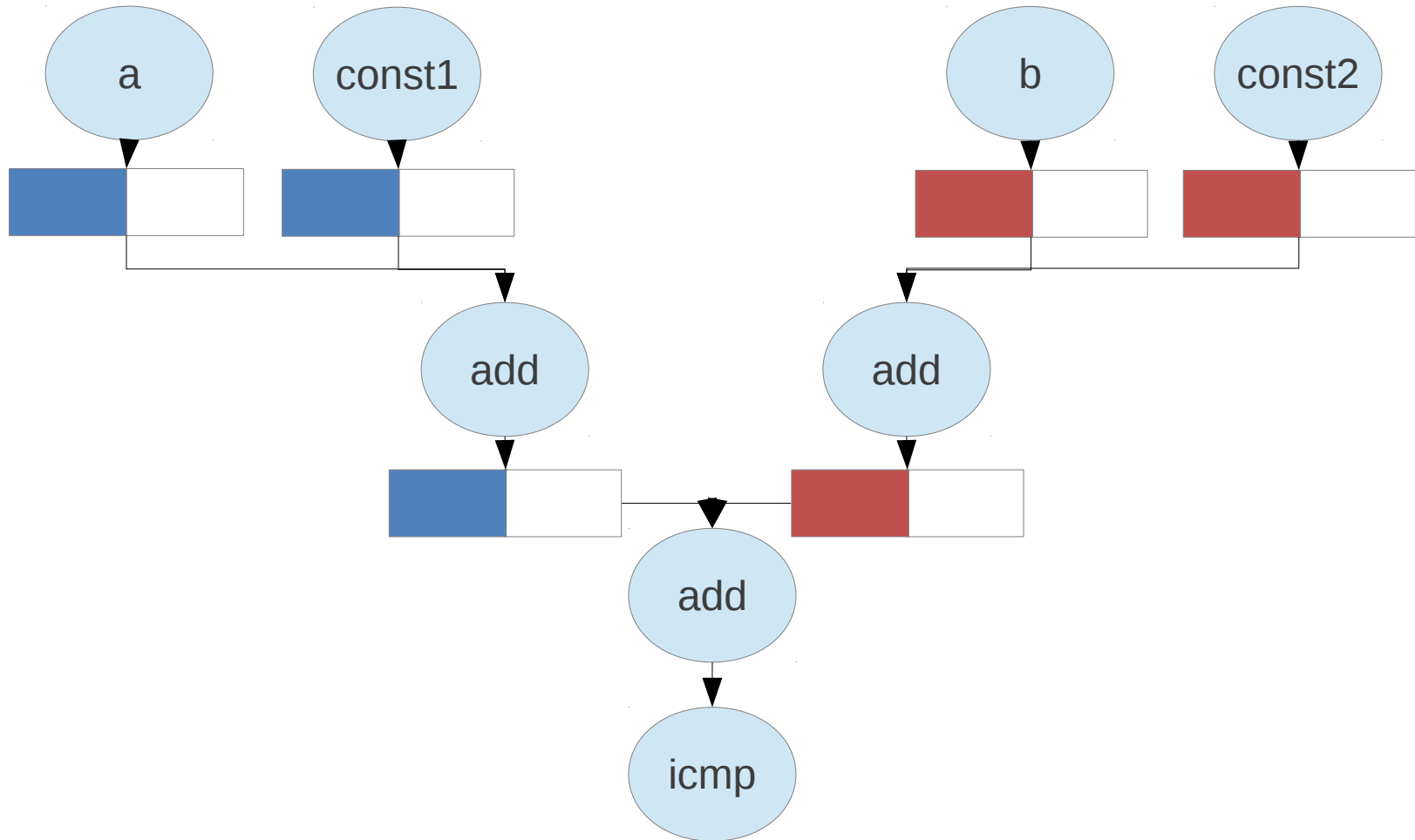
```
const1 = constant <1 x u32> 0x1  
const2 = constant <1 x u32> 0x2  
a      = add <1 x u32> a const1  
b      = add <1 x u32> b const2  
c      = add <1 x u32> a b  
const3 = constant <1 x u32> 0x9  
bool    = icmp _ult c c3  
br bool dest1 dest2
```

# Basic-block vectorization

---

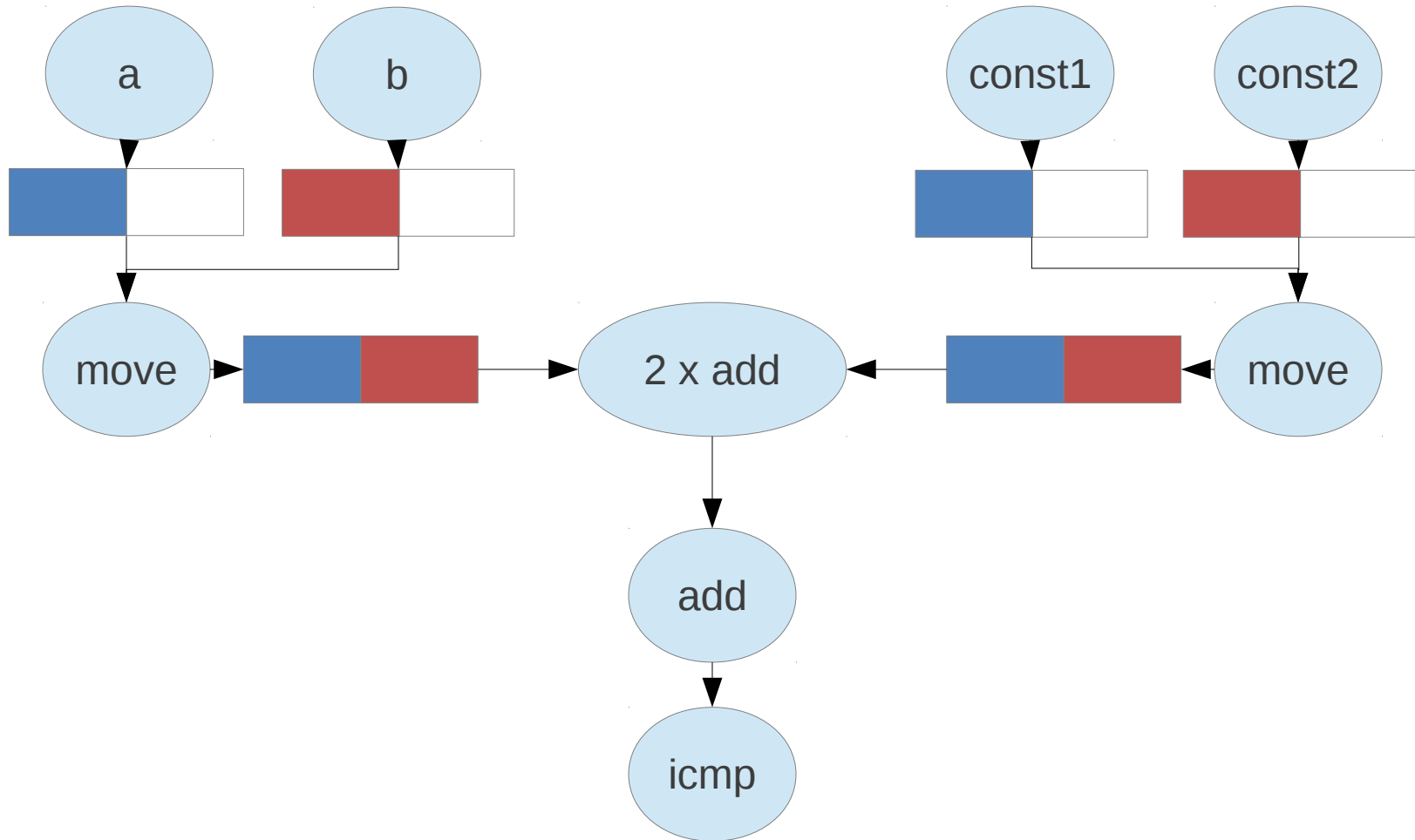


# Basic-block vectorization

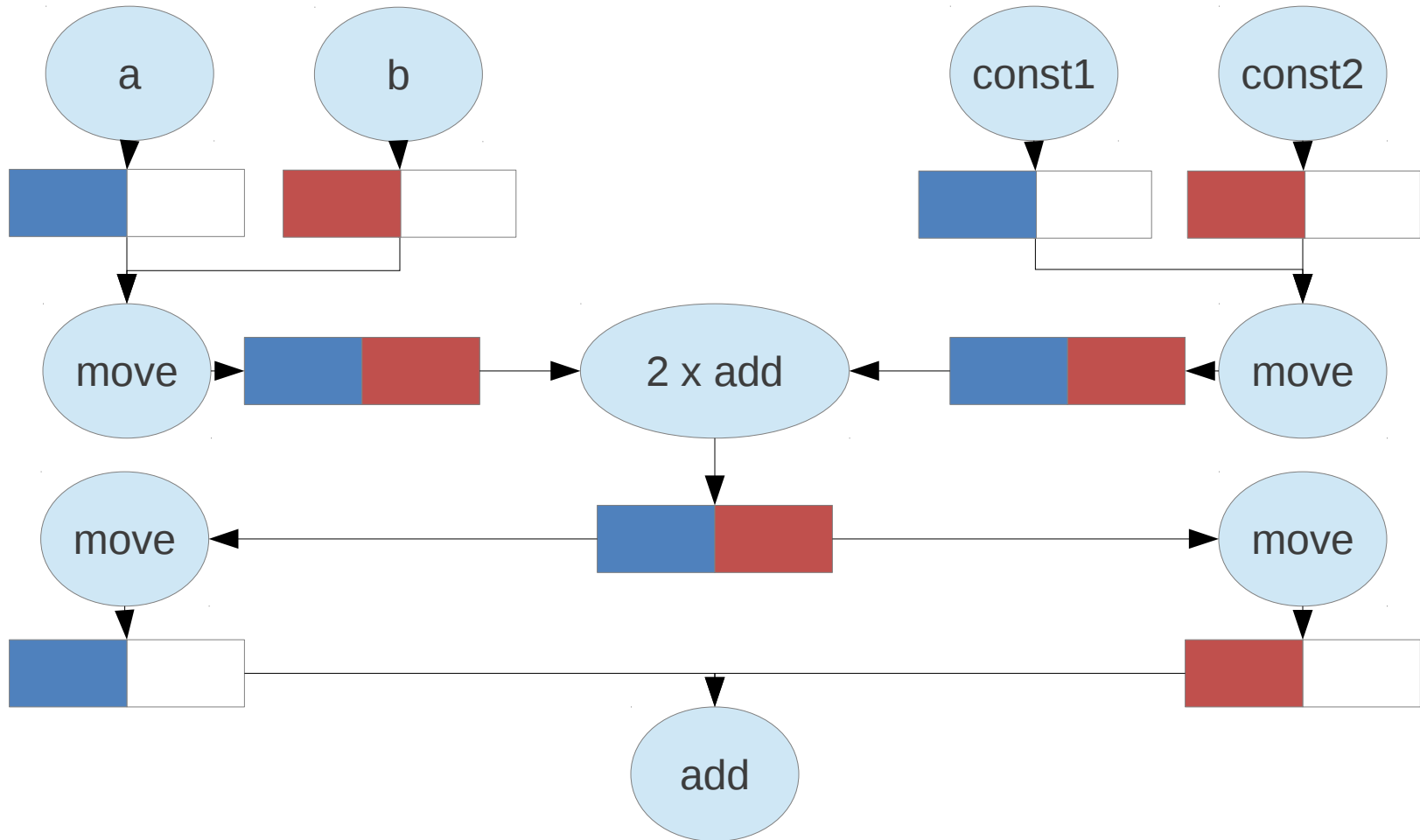




# Basic-block vectorization



# Basic-block vectorization



# Basic-block vectorization

---

## Alternatives

### Two algorithms were implemented

- LLVM-based basic-block vectorization
- Pair-based basic-block vectorization

# LLVM-based vectorization

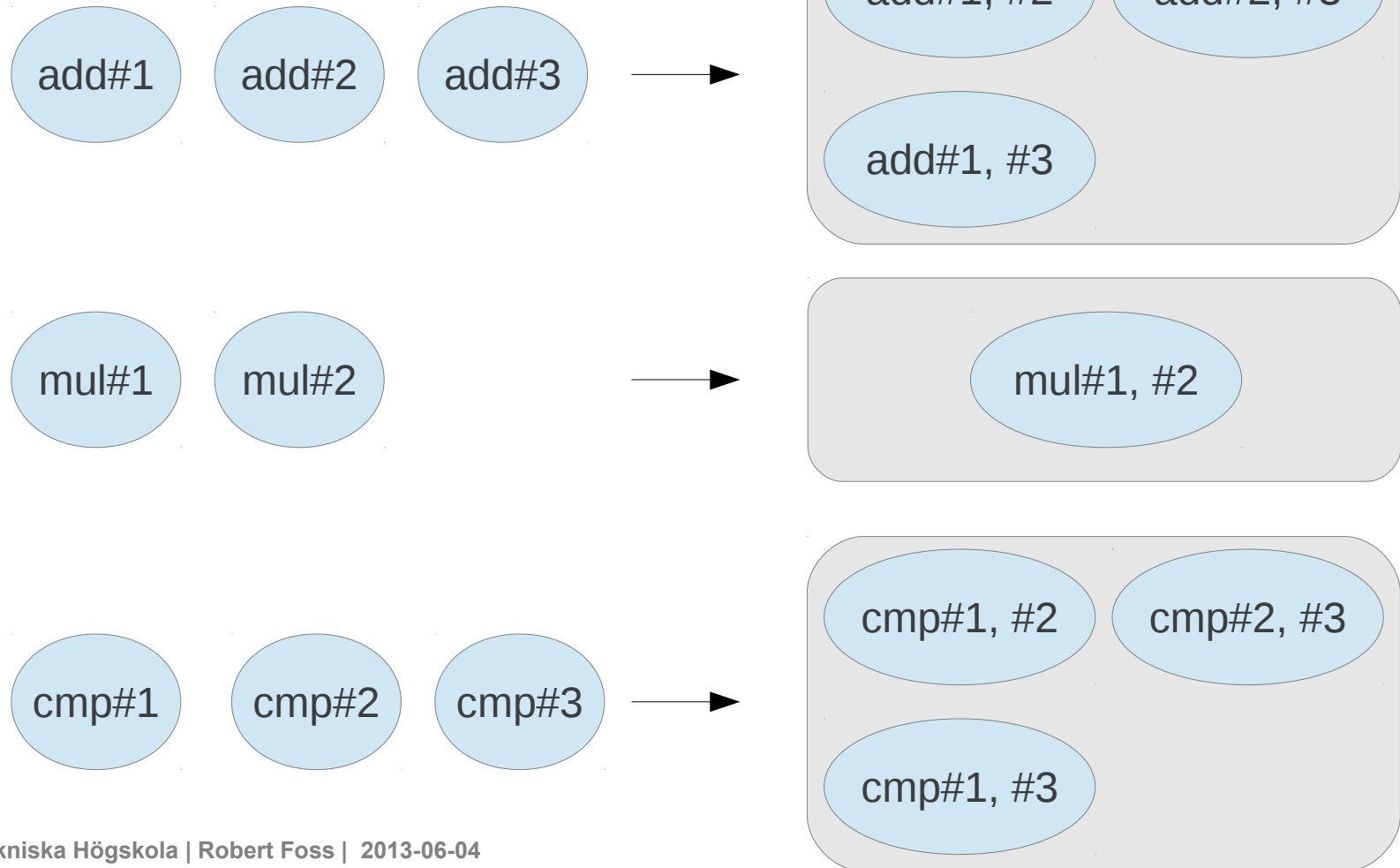
---

## Steps involved:

- Find potential pairs
- Find connections between pairs
- Pair selection
- Pair fusing
- Fixed-point iteration

# LLVM-based vectorization

Find potential pairs

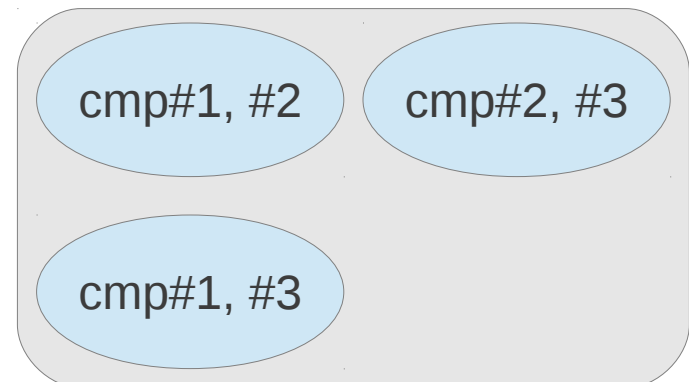
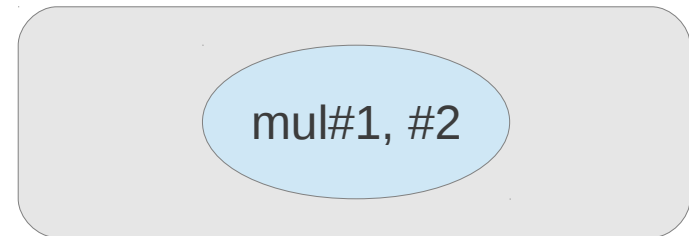
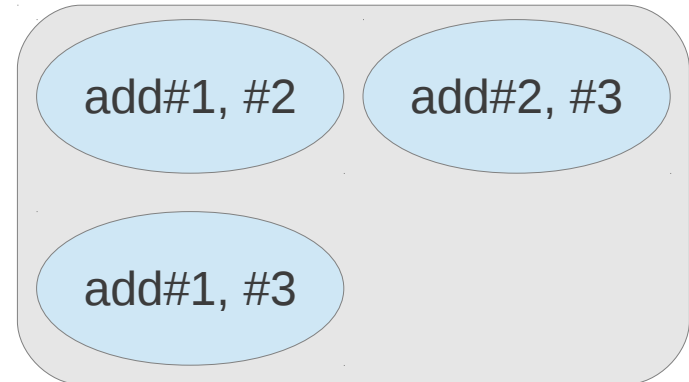


# LLVM-based vectorization

---

Find potential pairs

Steps involved:

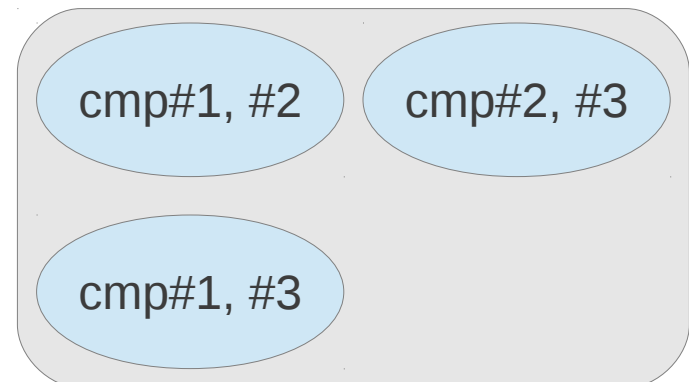
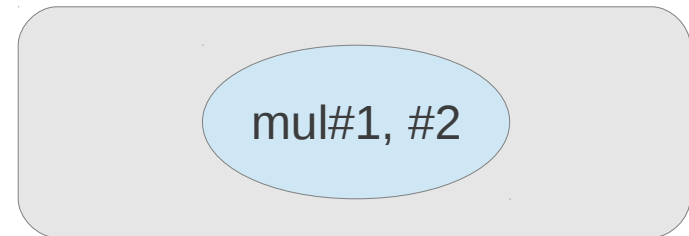
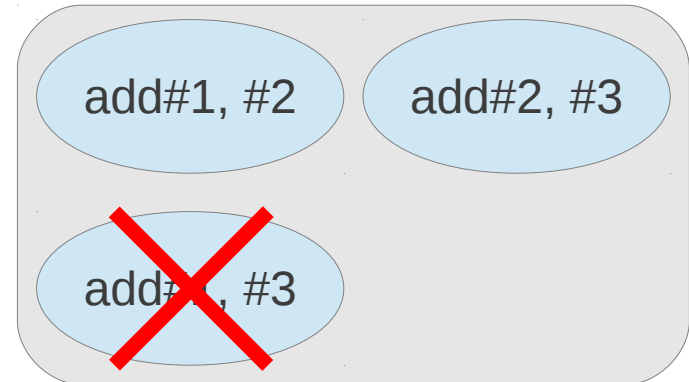


# LLVM-based vectorization

Find potential pairs

Steps involved:

- Remove intradependent pairs

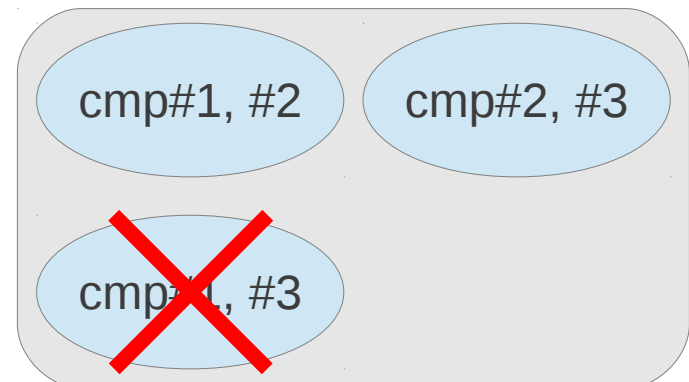
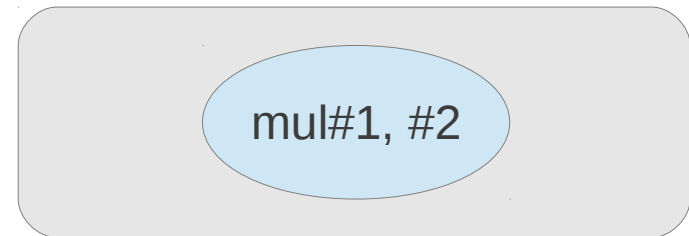
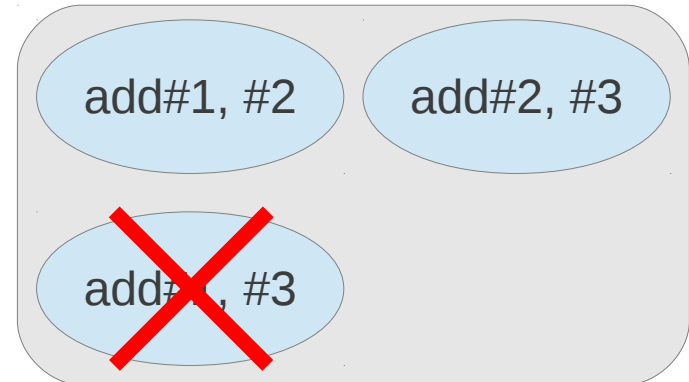


# LLVM-based vectorization

Find potential pairs

Steps involved:

- Remove intradependent pairs
- Remove illegal pairs

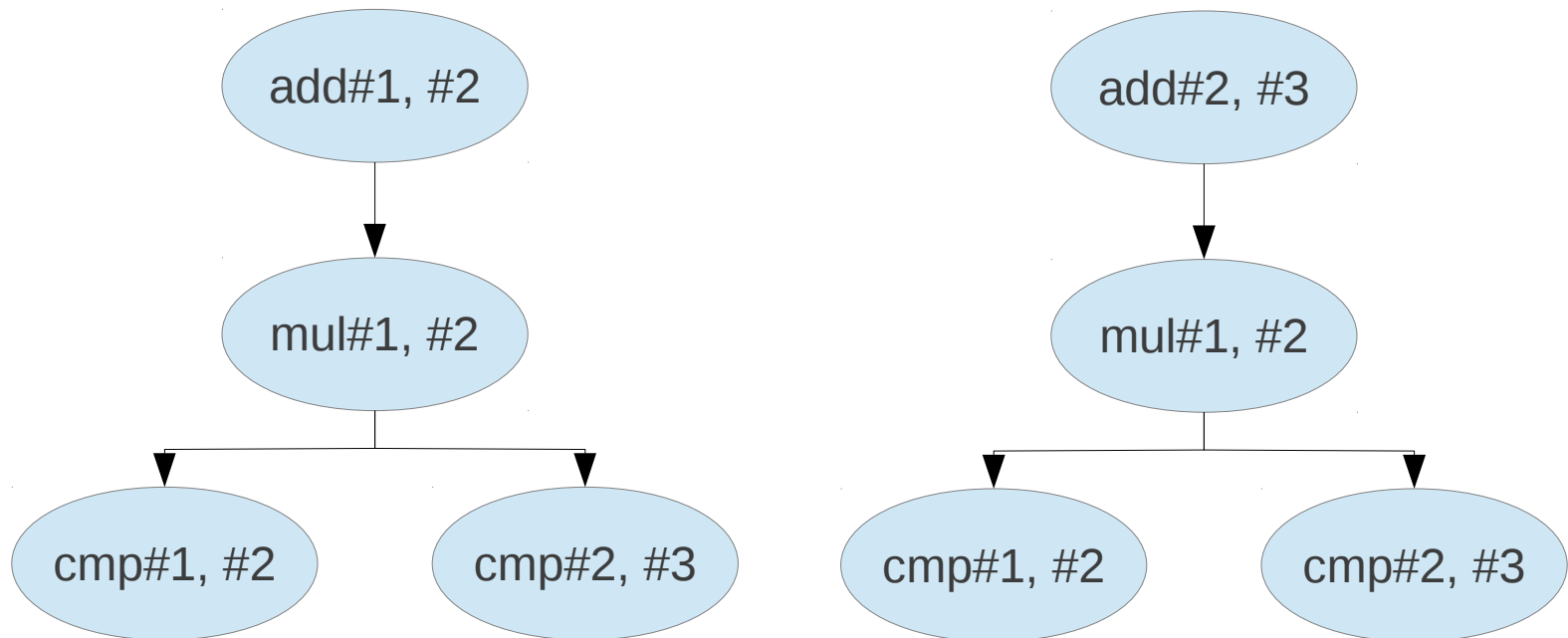




# LLVM-based vectorization

---

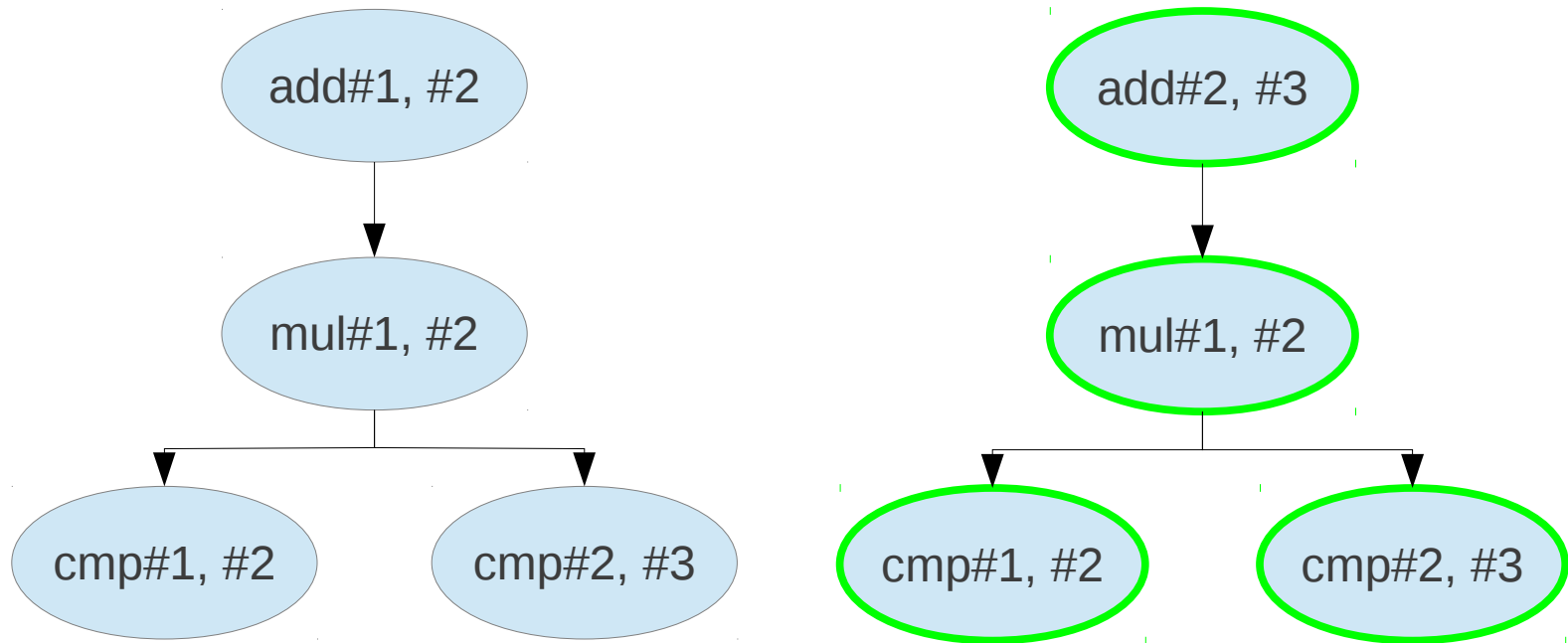
Find pair connections



# LLVM-based vectorization

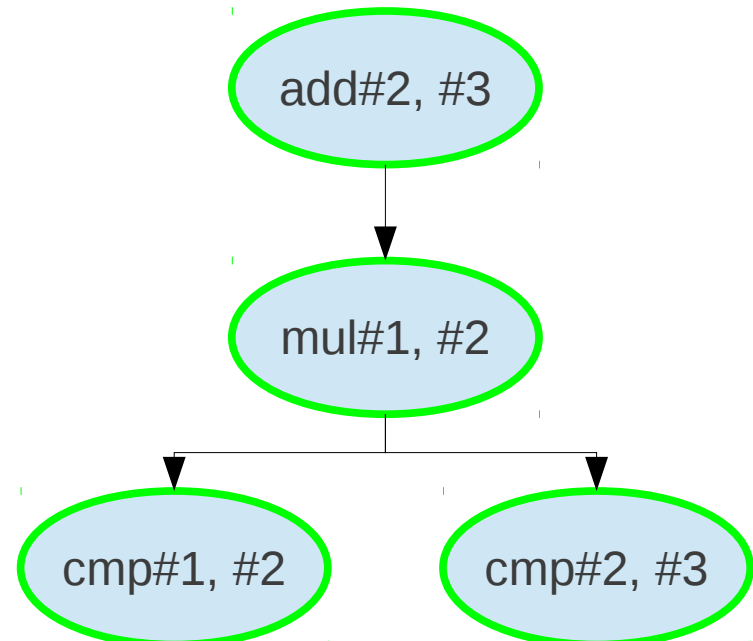
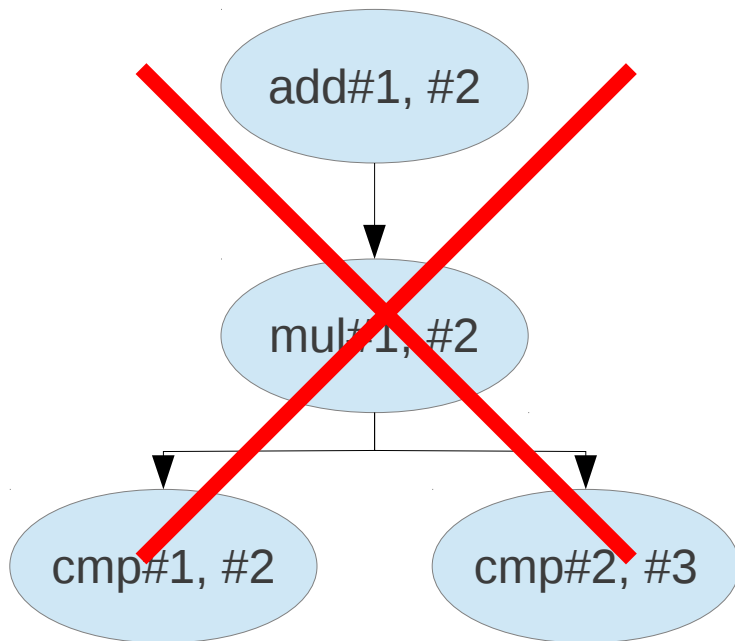
---

## Pair selection



# LLVM-based vectorization

## Pair selection



# Pair-based vectorization

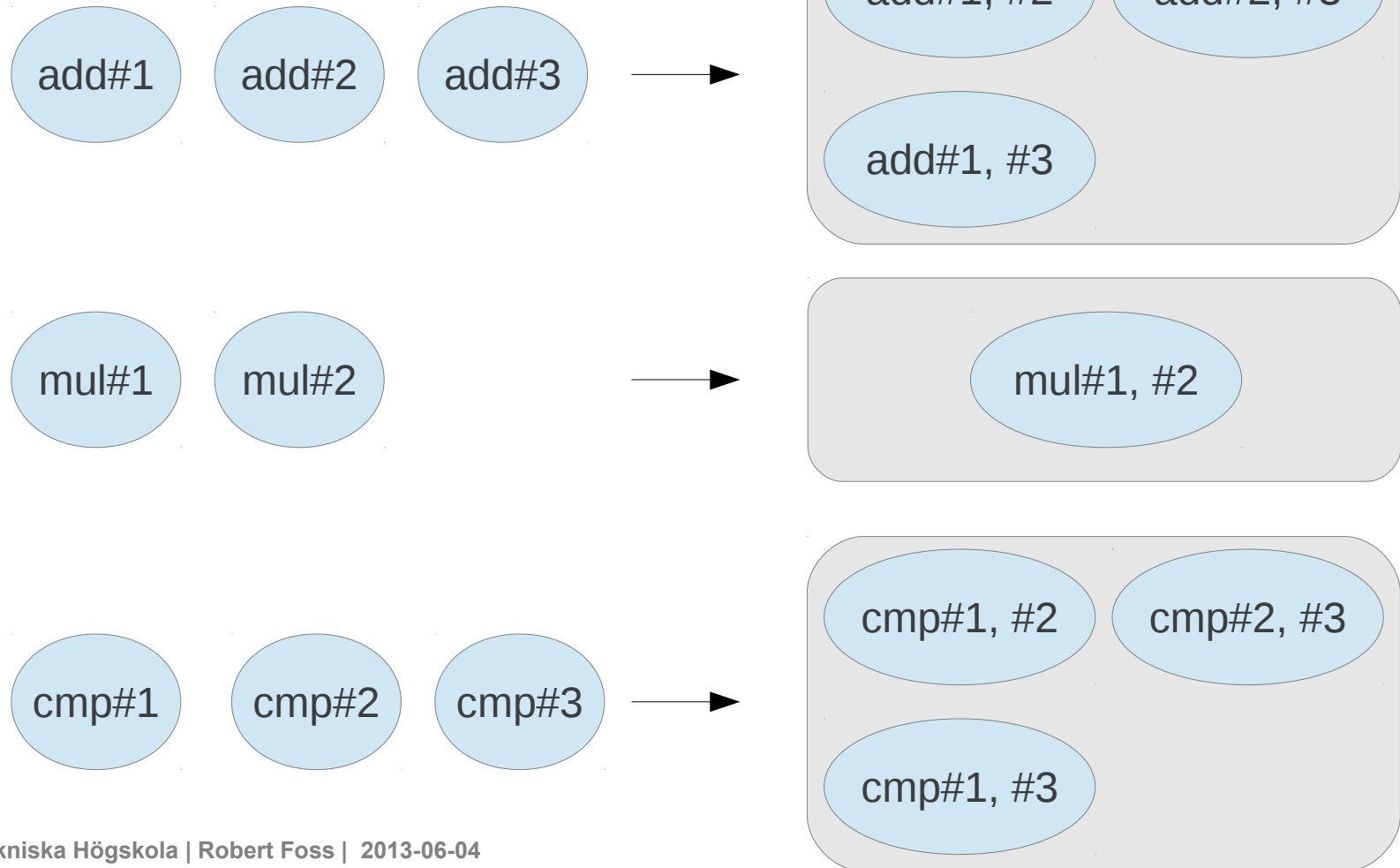
---

## Steps involved:

- Find potential pairs
- Pair selection
- Pair fusing

# LLVM-based vectorization

Find potential pairs

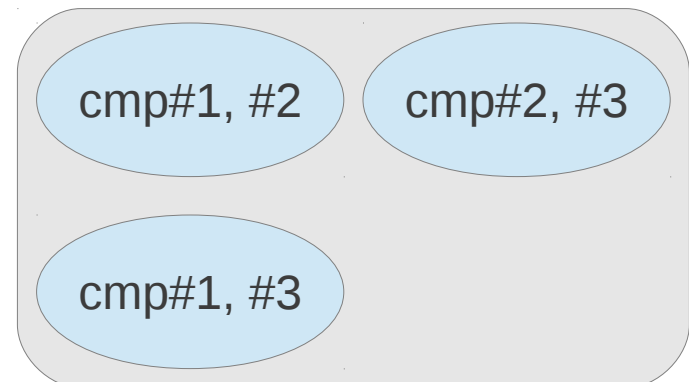
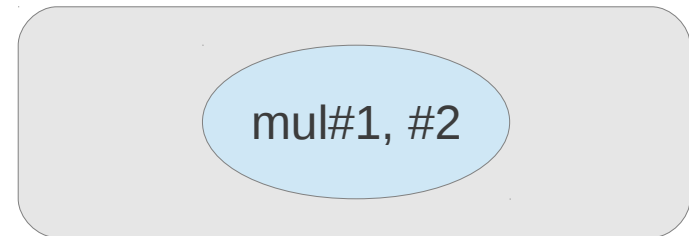
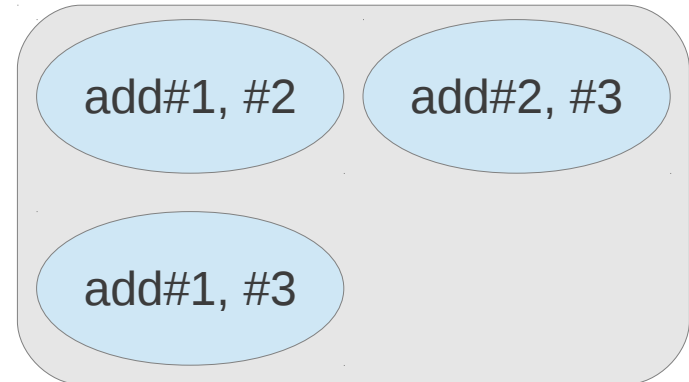


# LLVM-based vectorization

---

Find potential pairs

Steps involved:

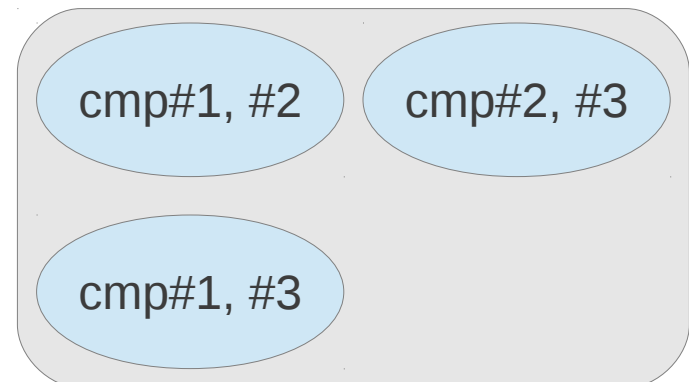
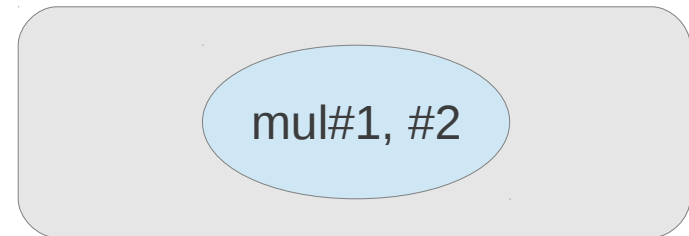
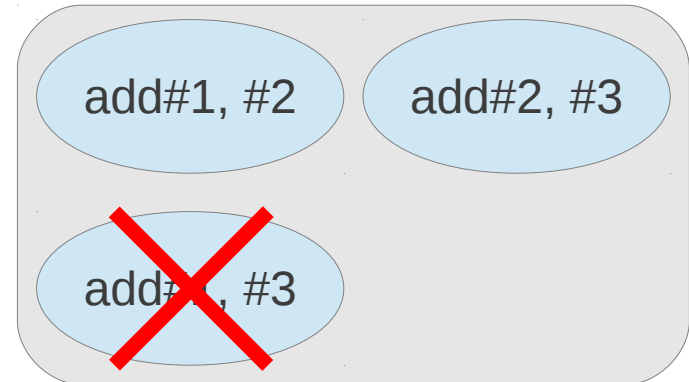


# LLVM-based vectorization

Find potential pairs

Steps involved:

- Remove intradependent pairs

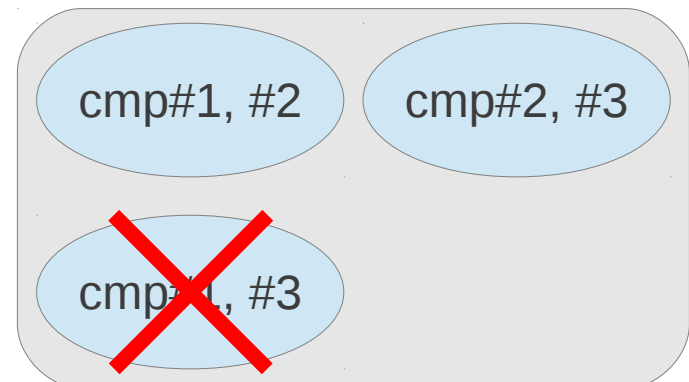
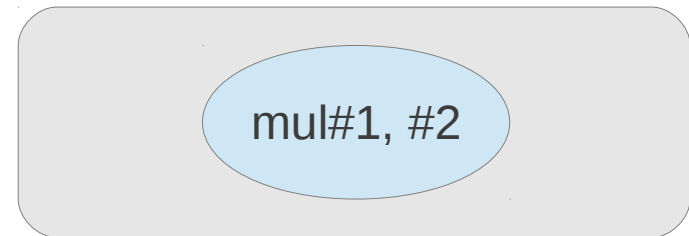
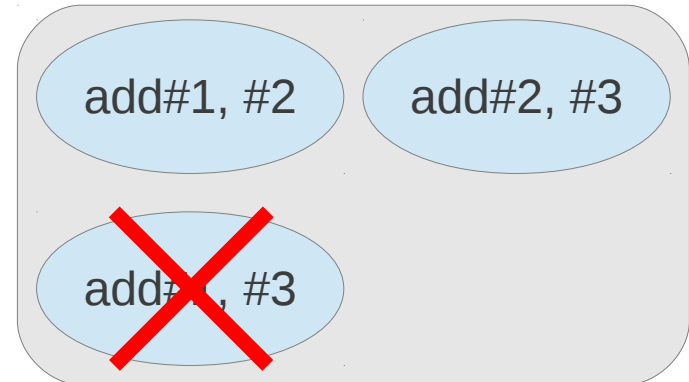


# LLVM-based vectorization

Find potential pairs

Steps involved:

- Remove intradependent pairs
- Remove illegal pairs





# Pair-based vectorization

---

## Pair selection

add#1, #2

add#2, #3

mul#1, #2

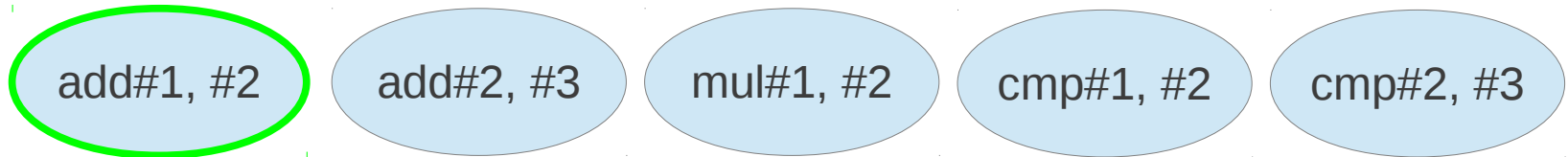
cmp#1, #2

cmp#2, #3

# Pair-based vectorization

---

## Pair selection

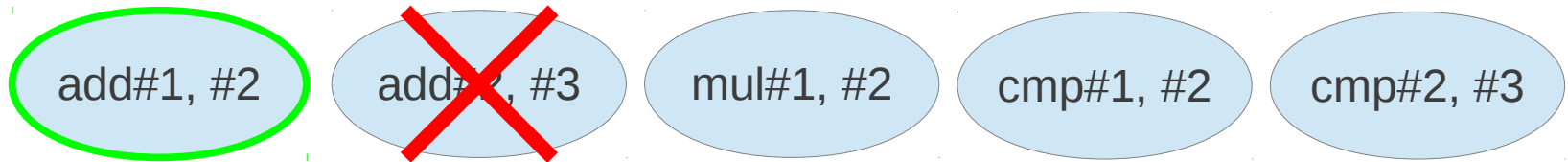


- Fuse most profitable pair

# Pair-based vectorization

---

## Pair selection

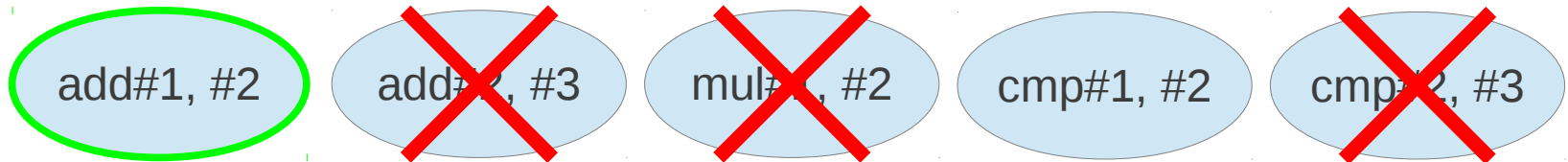


- Fuse most profitable pair
- Remove already fused operations

# Pair-based vectorization

---

## Pair selection

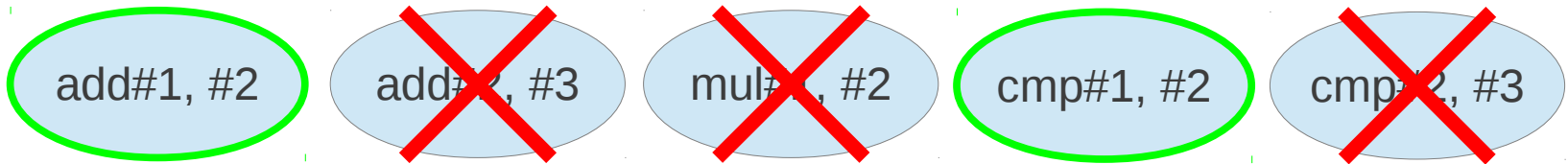


- Fuse most profitable pair
- Remove already fused operations
- Remove intradependent pairs

# Pair-based vectorization

---

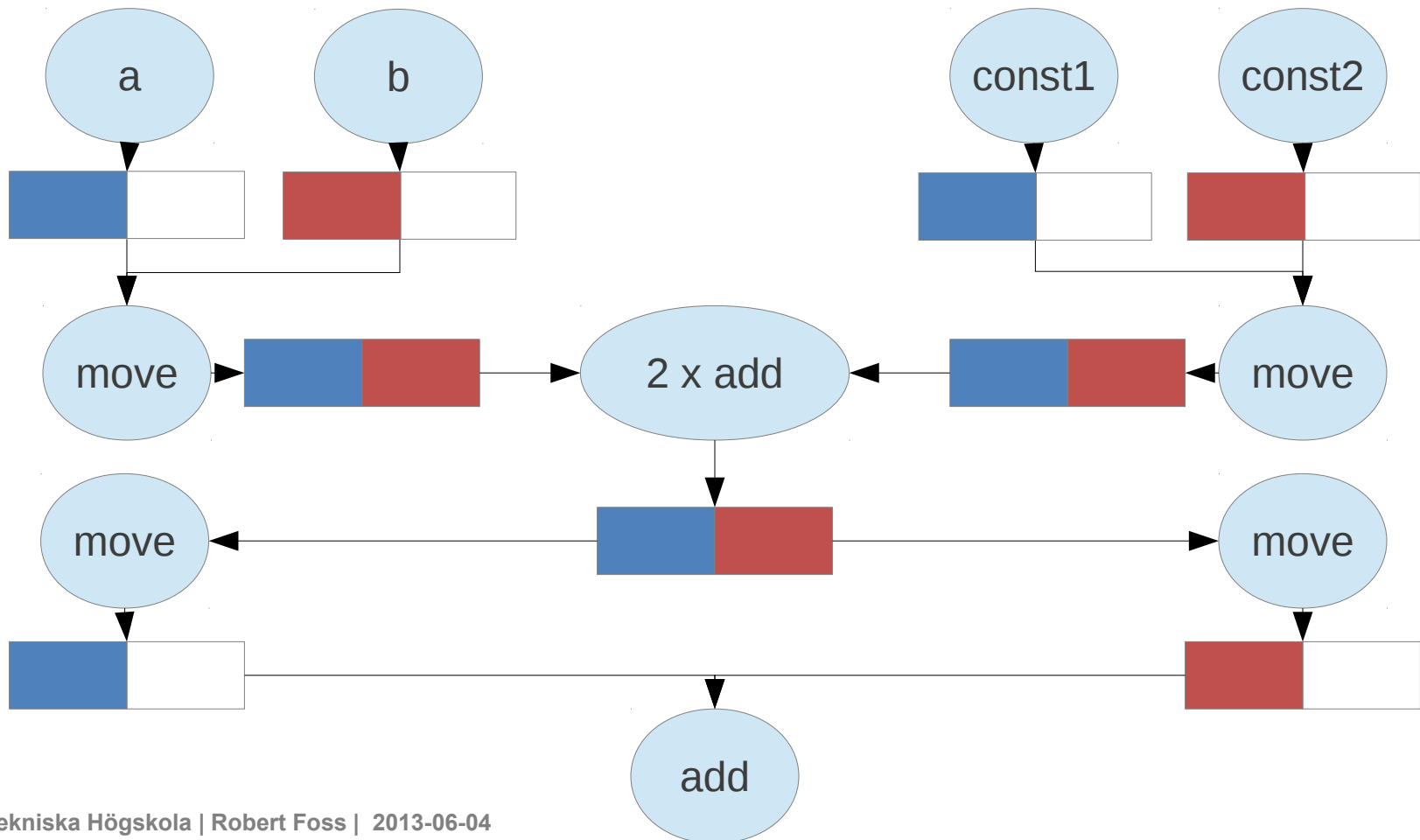
## Pair selection



- Fuse most profitable pair
- Remove already fused operations
- Remove intradependent pairs
- Fixed-point iteration

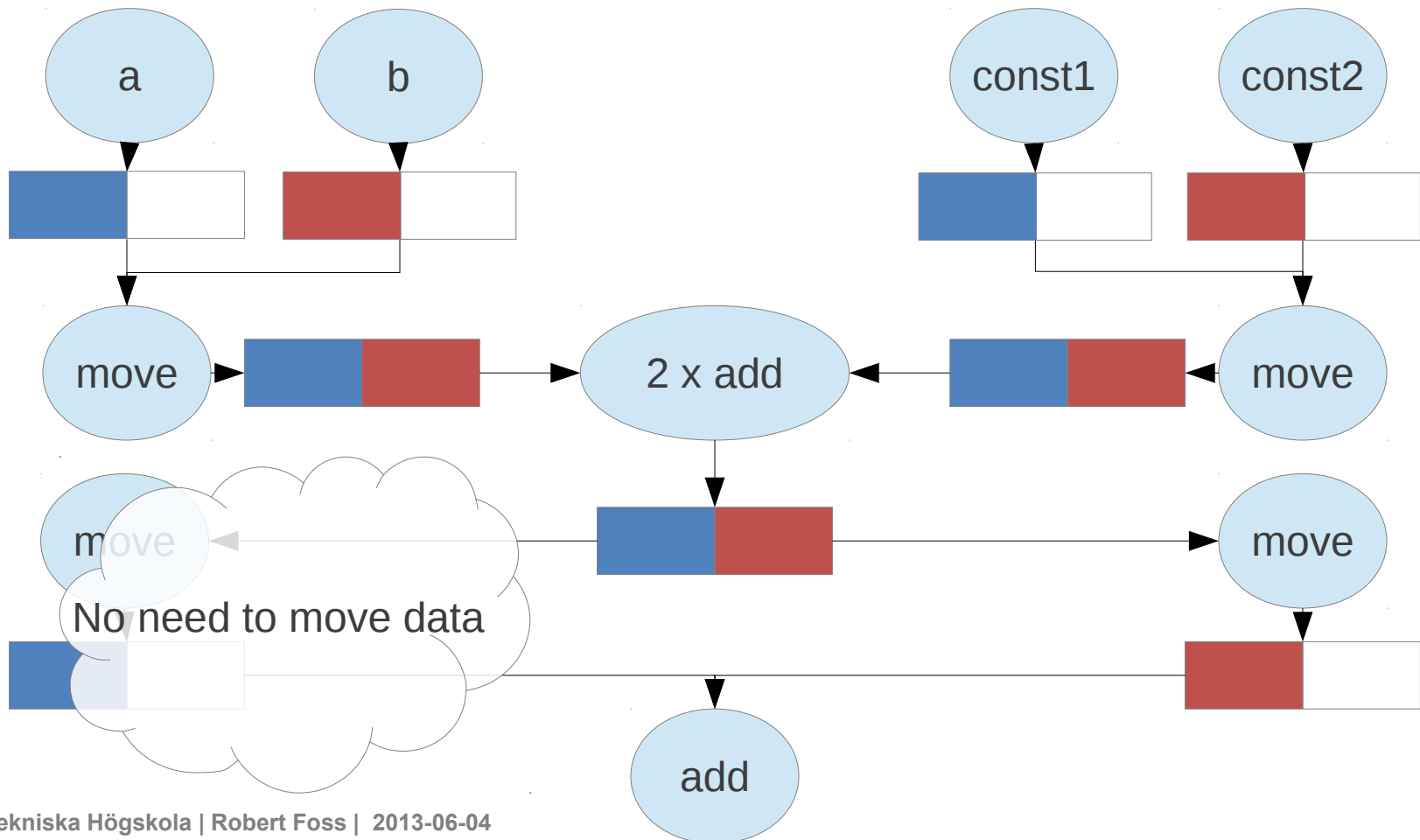
# Heuristics

Which? How?



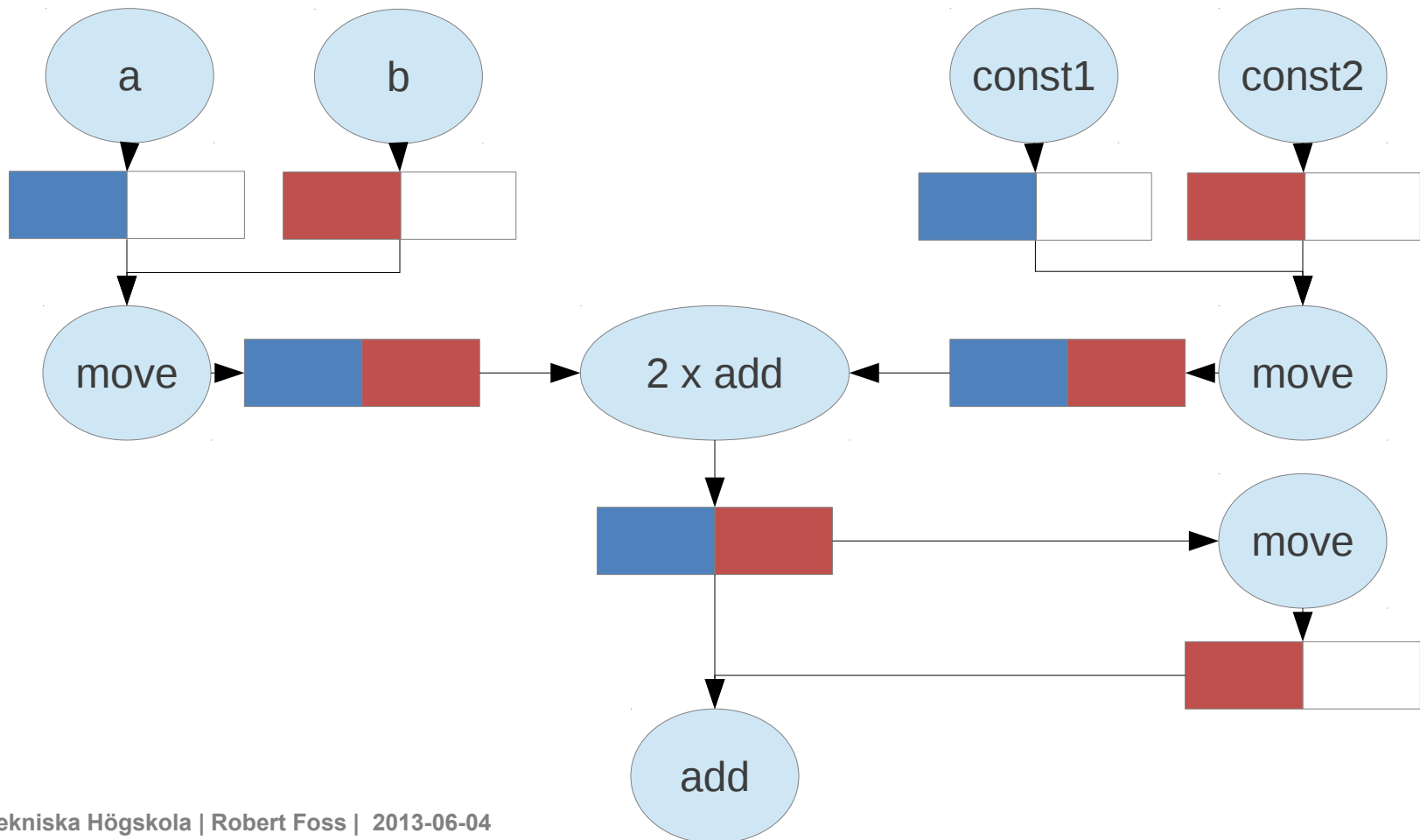
# Heuristics

Which? How?



# Heuristics

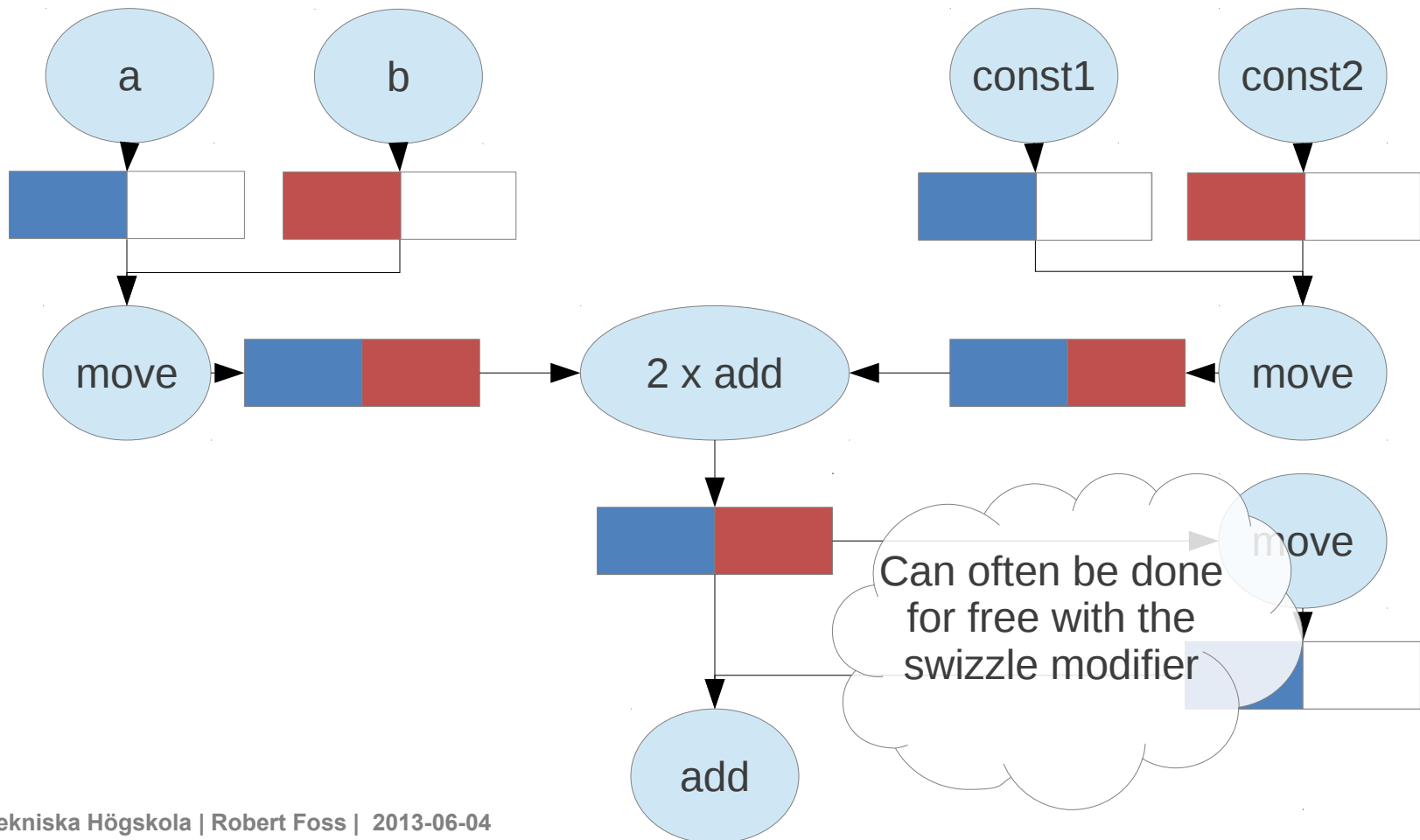
Which? How?





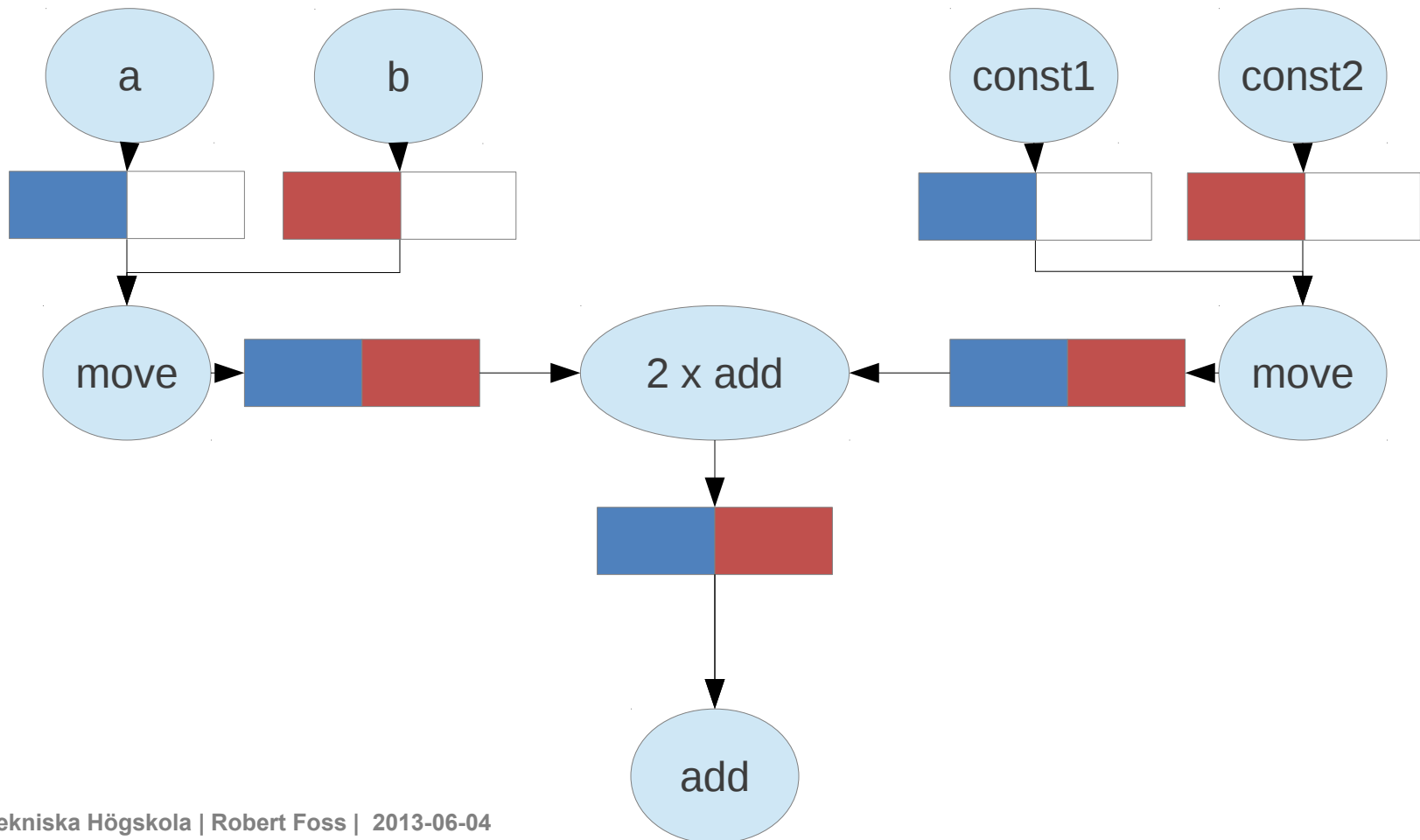
# Heuristics

Which? How?



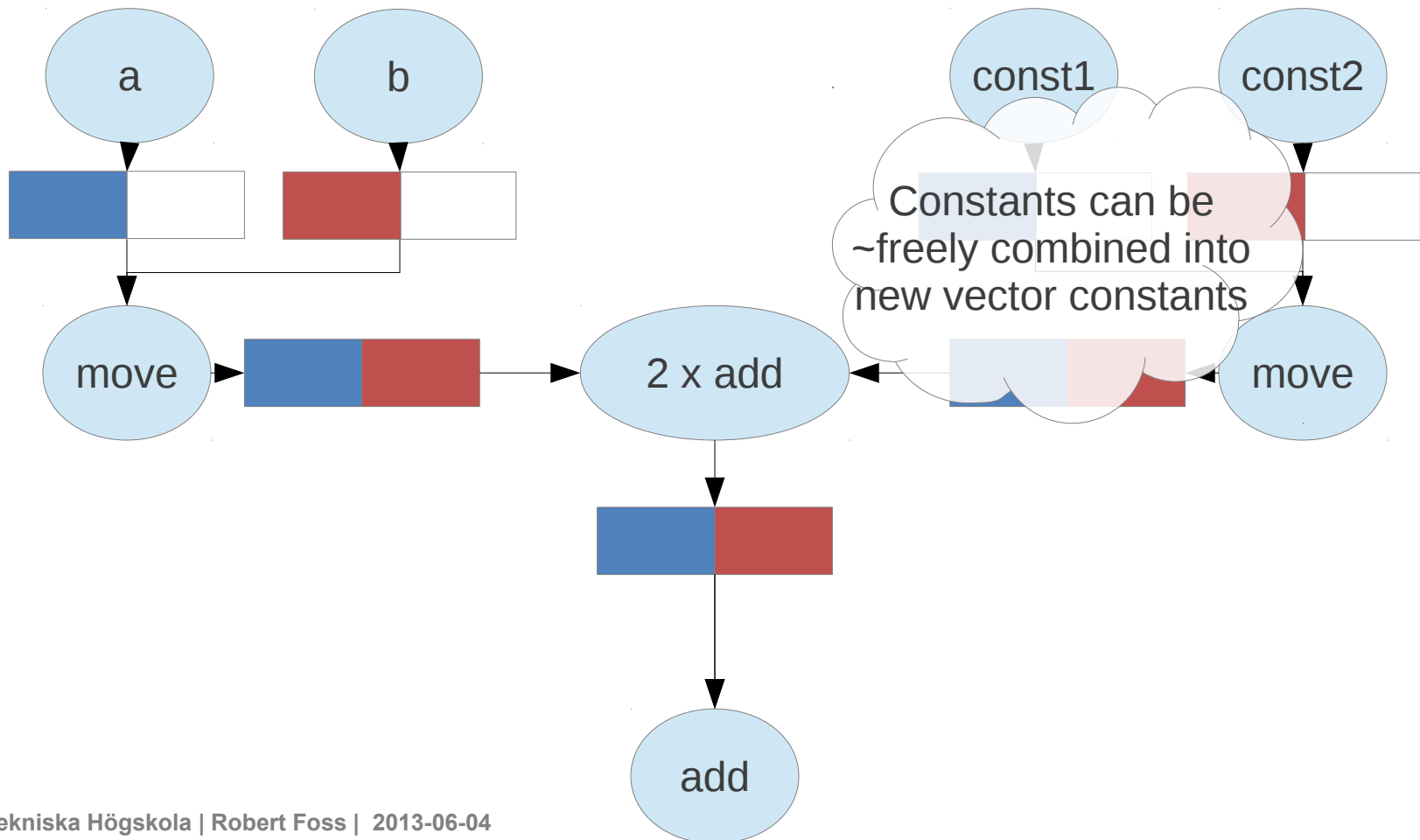
# Heuristics

Which? How?



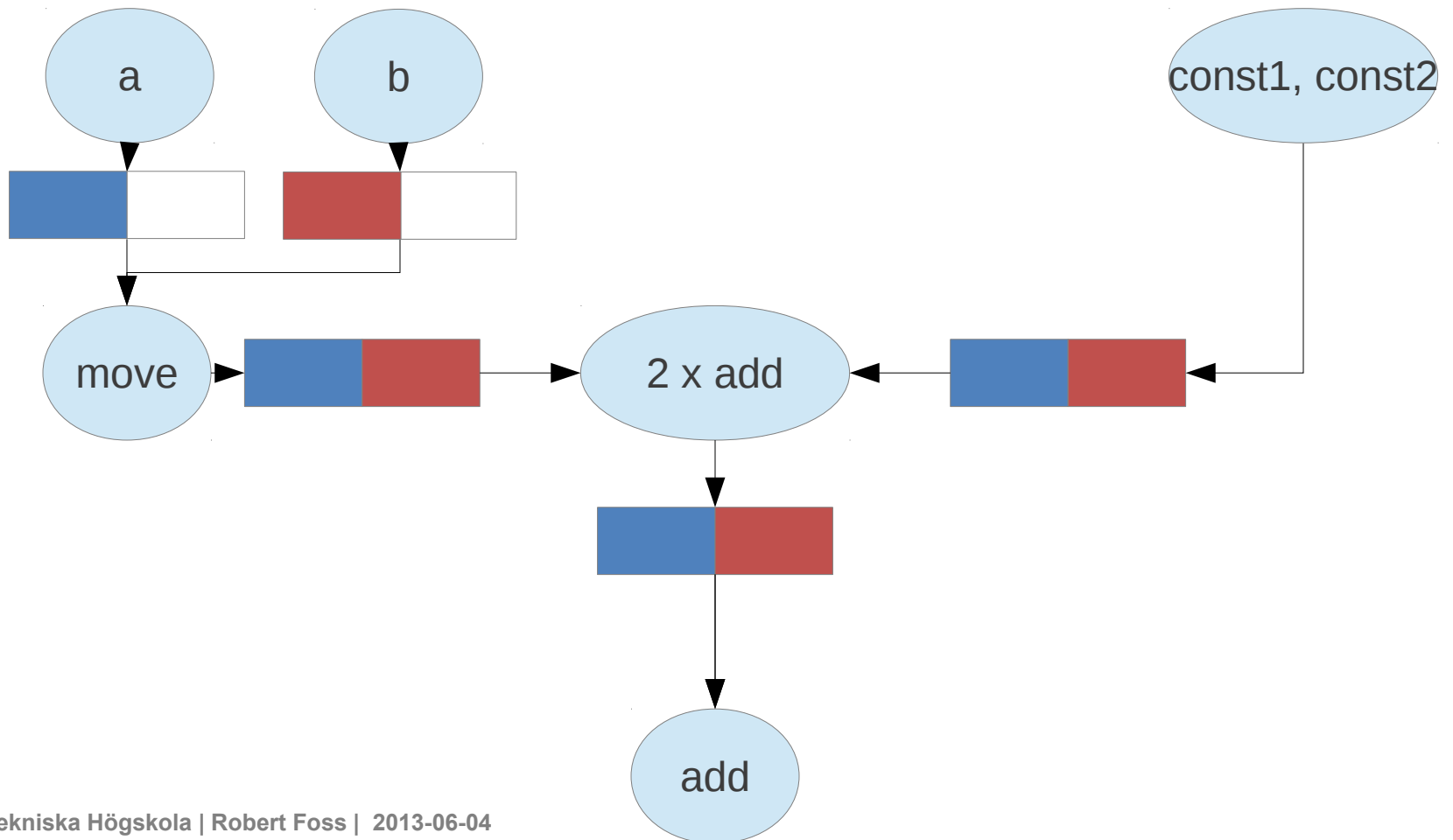
# Heuristics

Which? How?



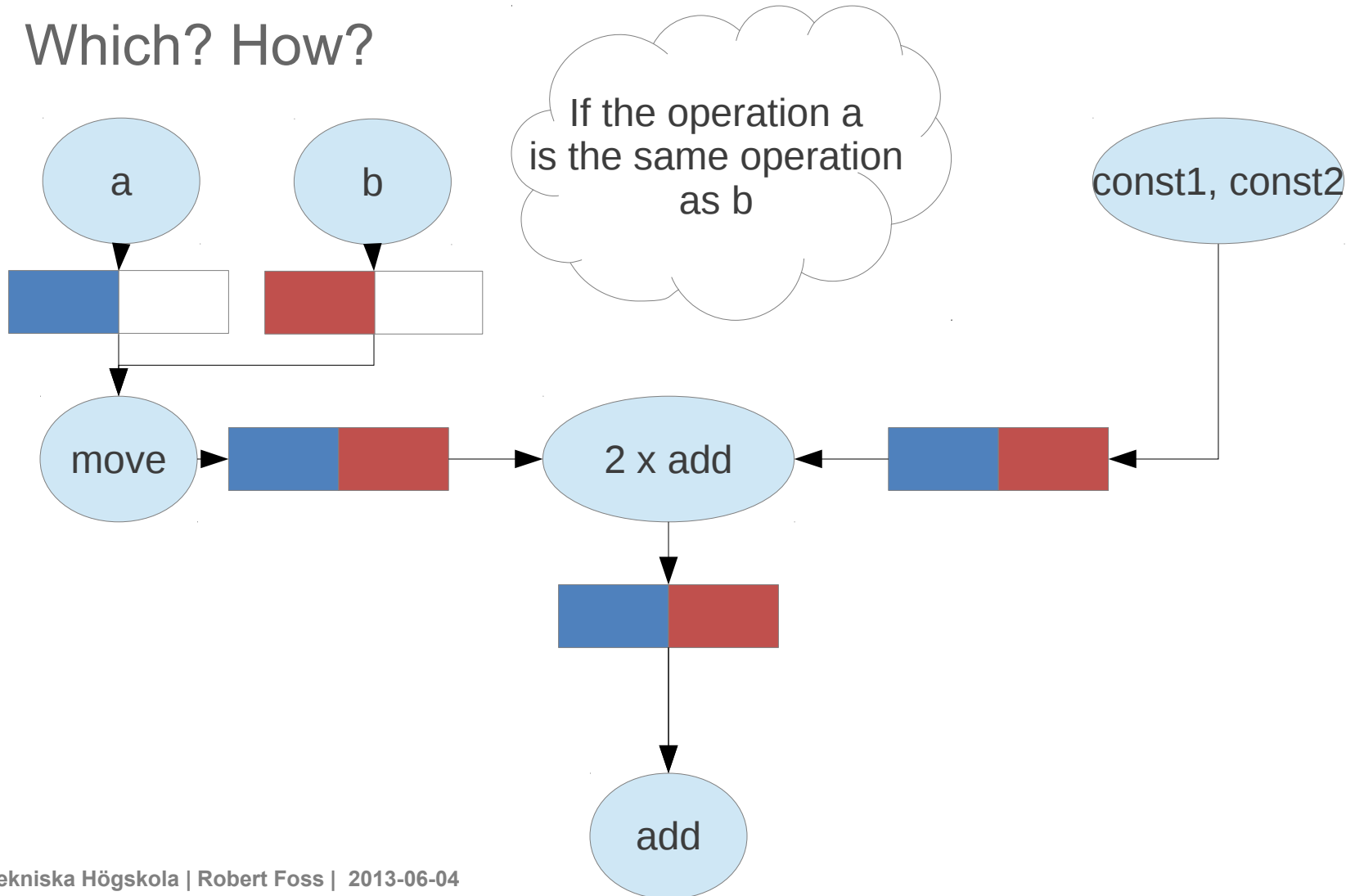
# Heuristics

Which? How?



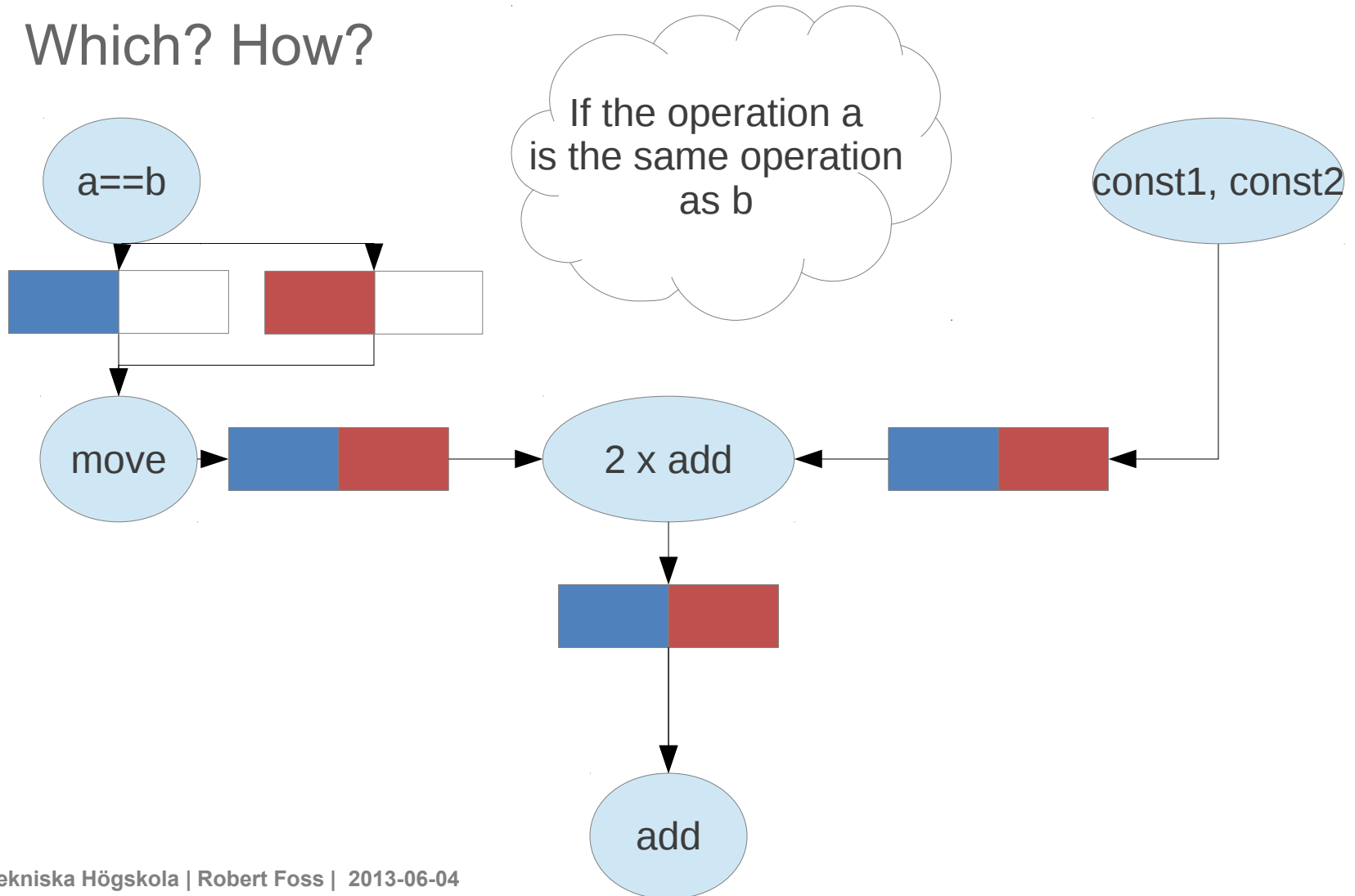
# Heuristics

Which? How?



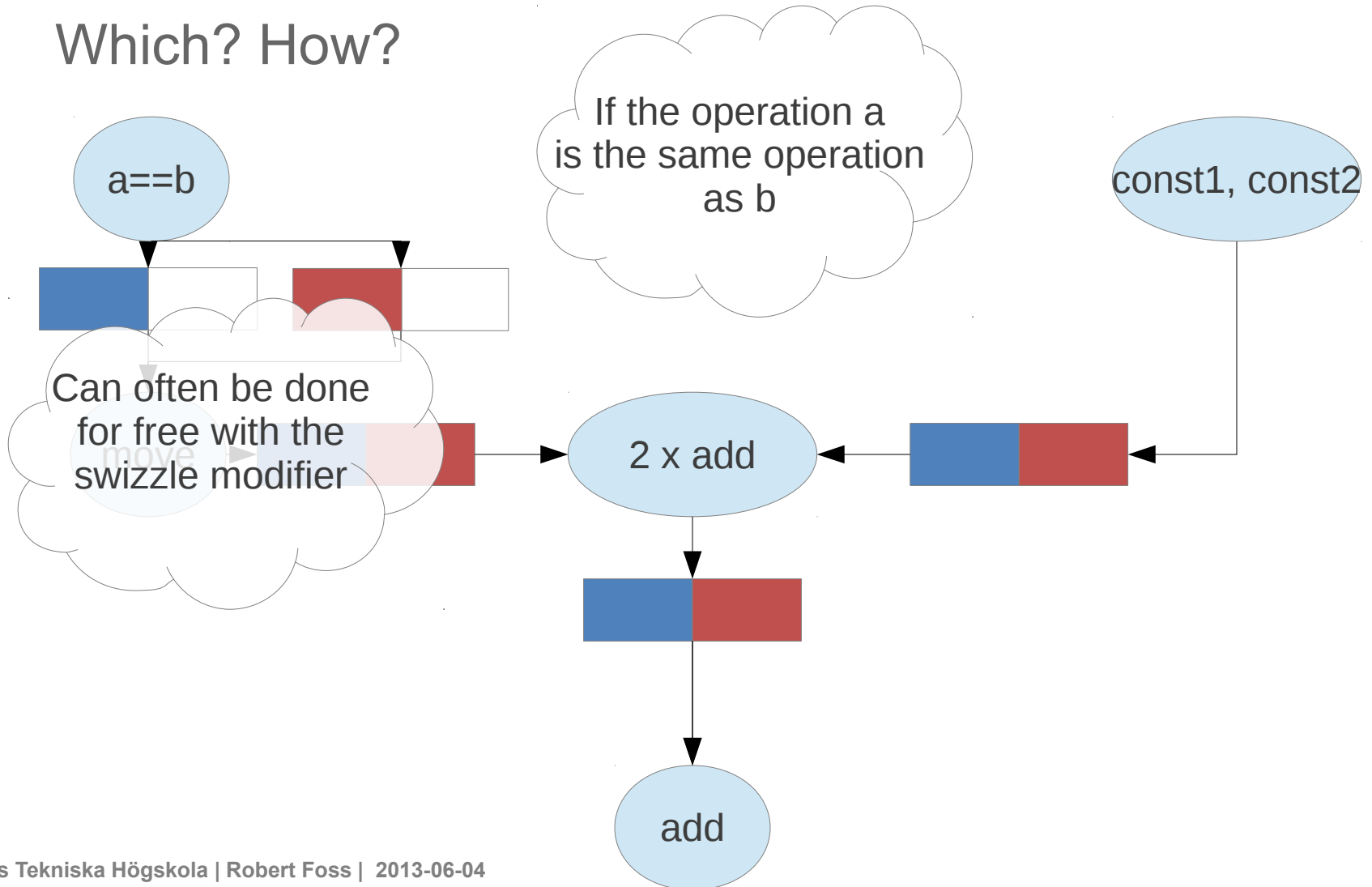
# Heuristics

Which? How?



# Heuristics

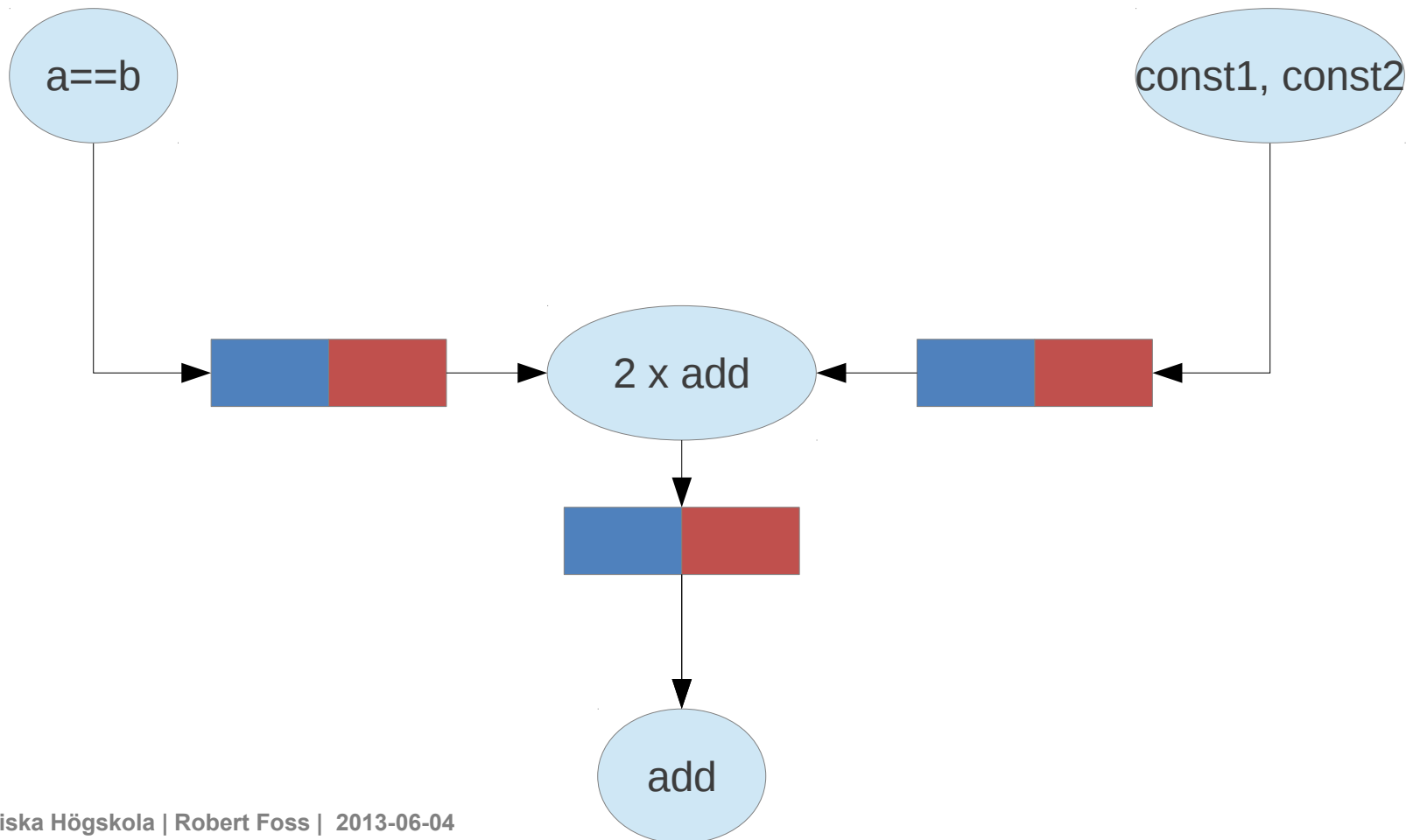
Which? How?



# Heuristics

---

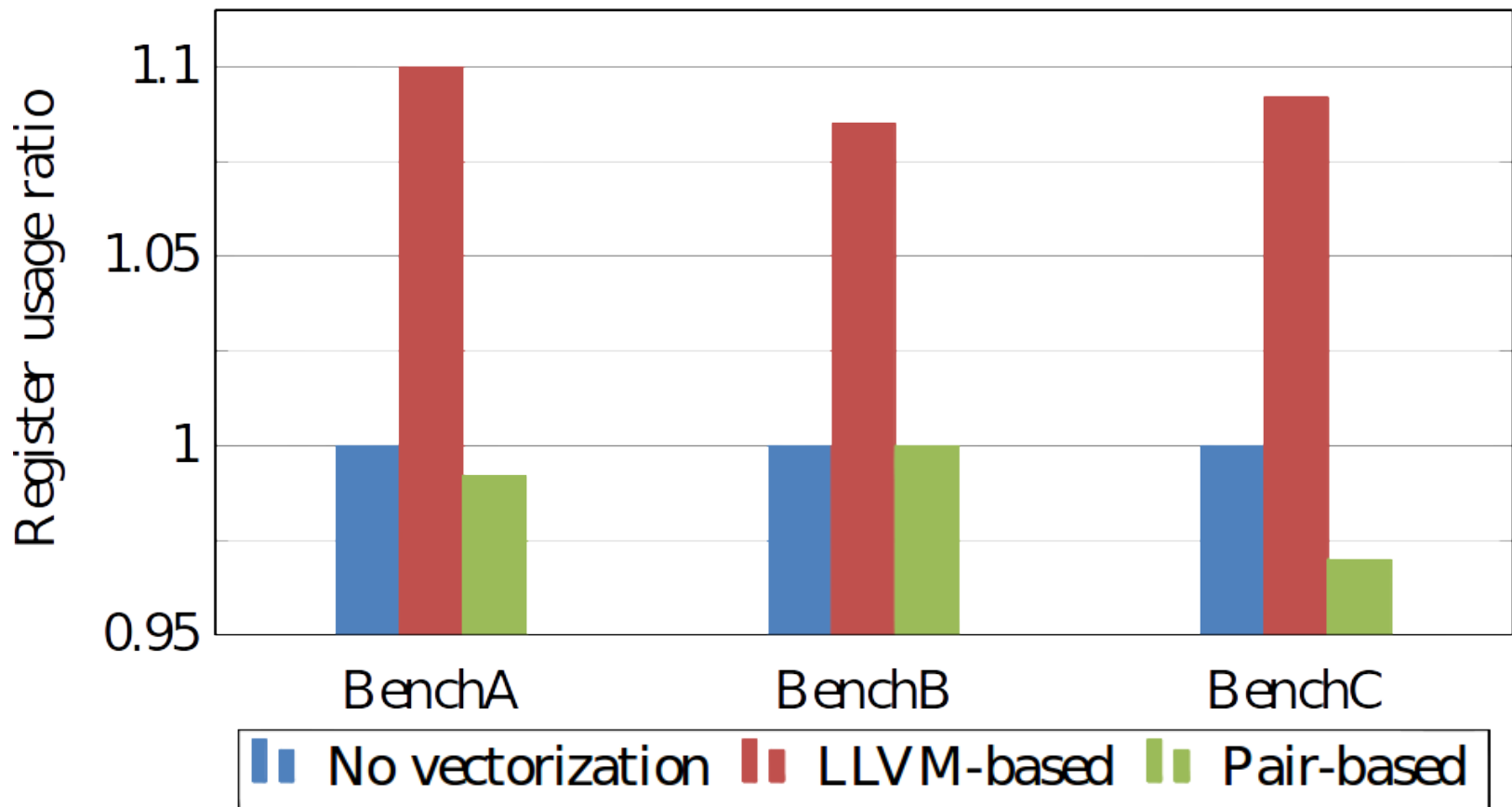
Which? How?





# Results

## Work Register Usage

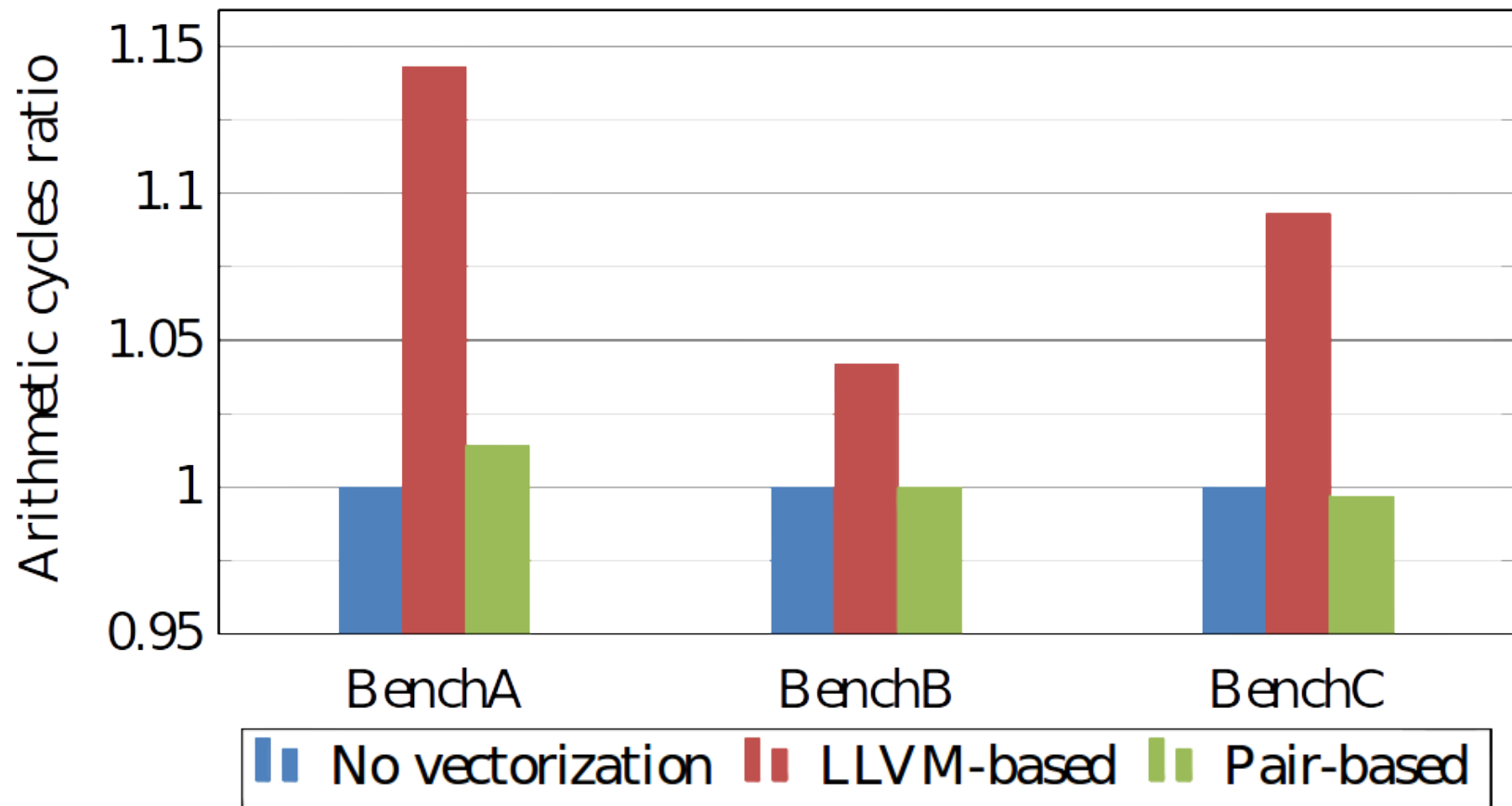


# Work Register Usage



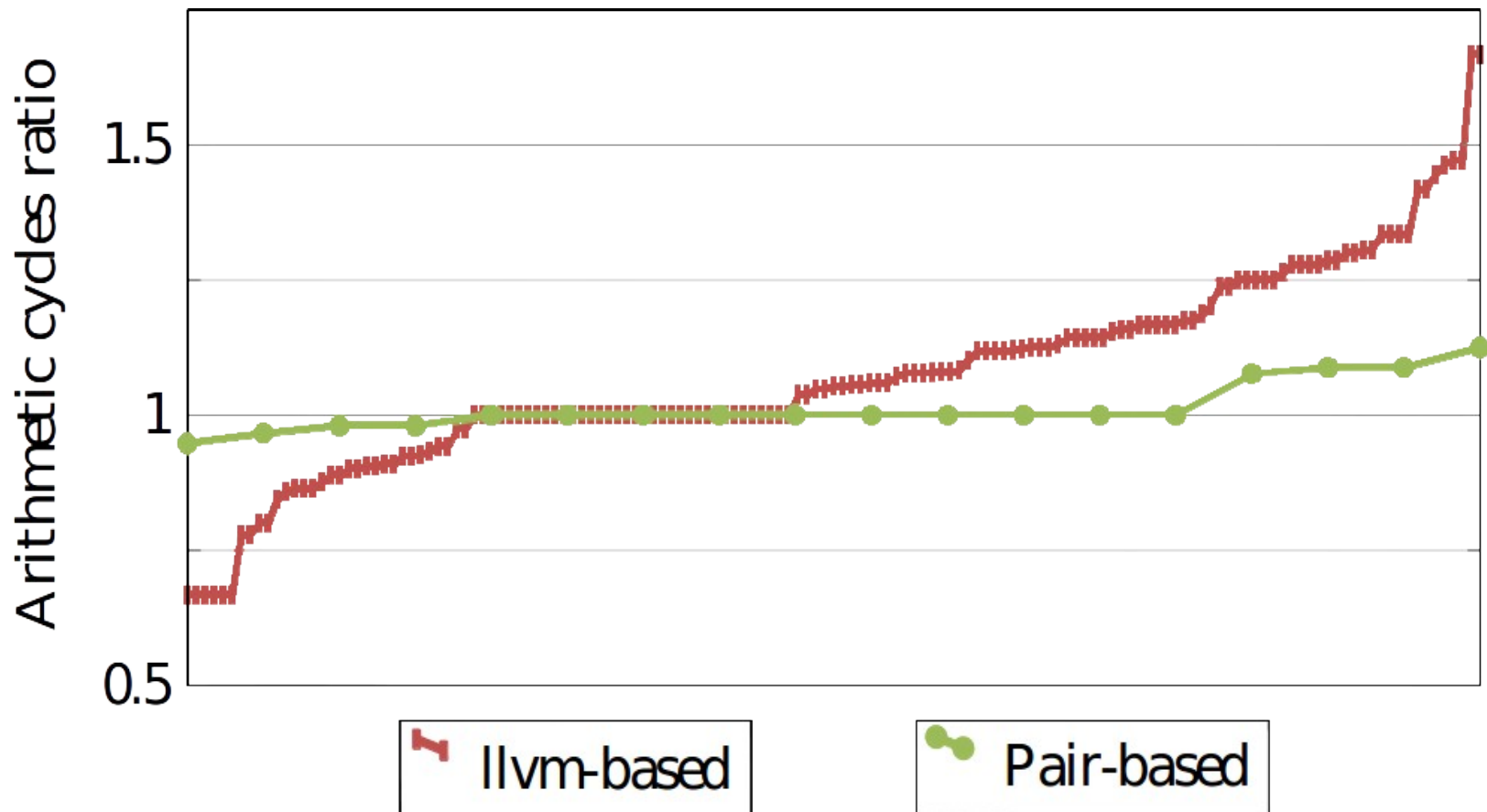
# Results

## Arithmetic Cycles



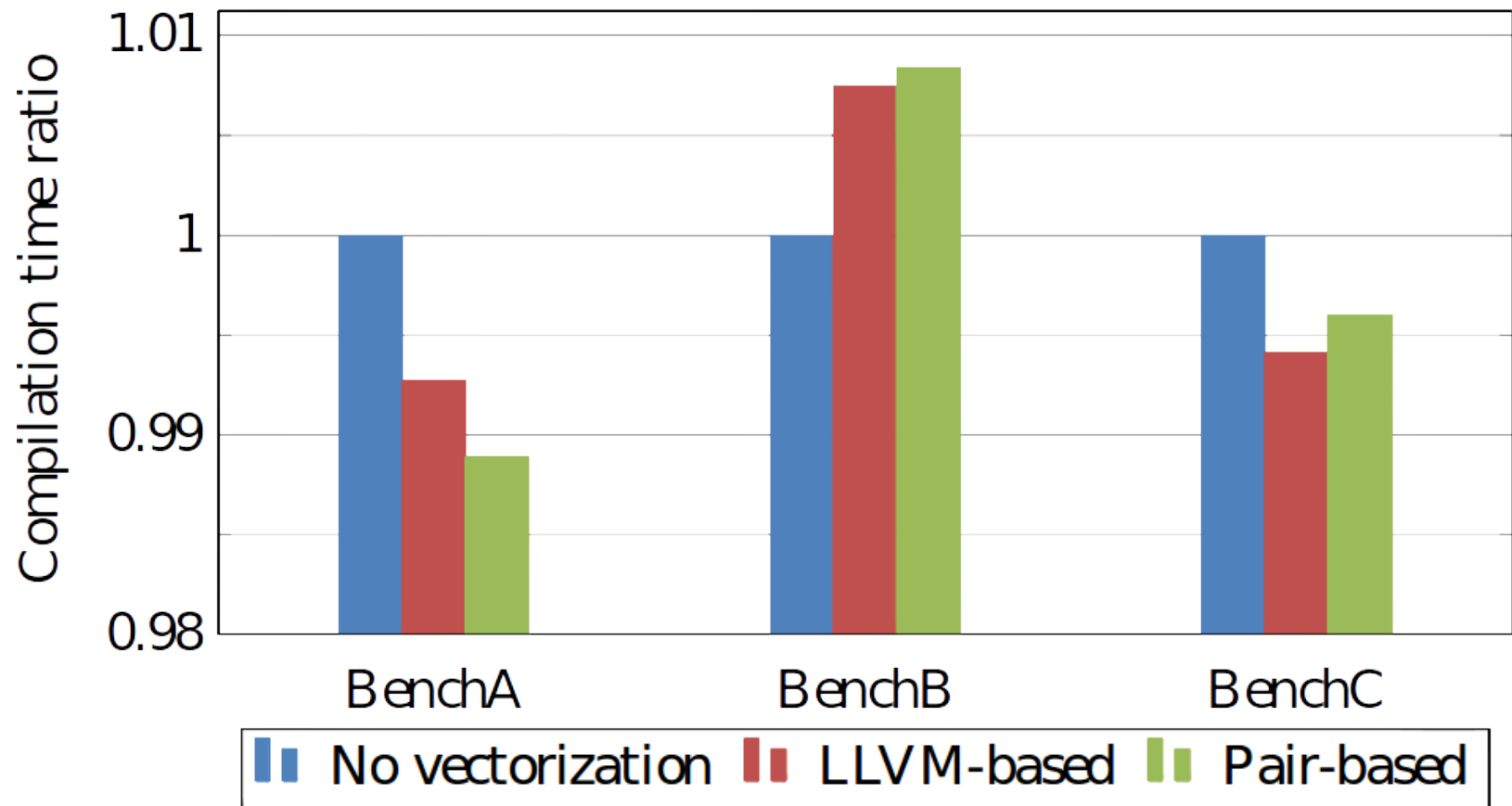
# Results

## Arithmetic Cycles



# Results

## Compilation Time



# Results

---

Why?

# Results

---

Why?

- Increased register pressure

# Results

---

## Why?

- Increased register pressure
- Increased scheduling tightness
- The cost of moving data around