CS 598CM: ML for Compilers and Architecture

Instructor: Charith Mendis

A Fast Fourier transform Compiler

Creation: fftgen

FFT Size

Input - X or output of another FFT

```
fftgen : int -> (int -> Complex.expr) ->
         int -> (int -> Complex.expr)
```

Backward or forward

Output - Y or output of another FFT

What is its' output?

Expression DAG (dependency graph)

What part of the compiler pipeline is this?

After Semantic Analysis

fftgen

$$Y[i_1 + i_2 n_1] = \sum_{j_2=0}^{n_2-1} \left[\left(\sum_{j_1=0}^{n_1-1} X[j_1 n_2 + j_2] \omega_{n_1}^{-i_1 j_1} \right) \omega_n^{-i_1 j_2} \right] \omega_{n_2}^{-i_2 j_2} . \tag{3}$$

```
let rec cooley_tukey n1 n2 input sign =
  let tmp1 j2 = fftgen n1
    (fun j1 -> input (j1 * n2 + j2)) sign in
  let tmp2 i1 j2 =
    exp n (sign * i1 * j2) @* tmp1 j2 i1 in
  let tmp3 i1 = fftgen n2 (tmp2 i1) sign
  in
   (fun i -> tmp3 (i mod n1) (i / n1))
```

Y[i] **i1** **i2**

fftgen

$$Y[i_1 + i_2 n_1] = \qquad (3)$$
$$\sum_{j_2=0}^{n_2-1} \left[\left(\sum_{j_1=0}^{n_1-1} X[j_1 n_2 + j_2] \omega_{n_1}^{-i_1 j_1} \right) \omega_n^{-i_1 j_2} \right] \omega_{n_2}^{-i_2 j_2} .$$

```
let rec cooley_tukey n1 n2 input sign =
  let tmp1 j2 = fftgen n1
    (fun j1 -> input (j1 * n2 + j2)) sign in
  let tmp2 i1 j2 =
    exp n (sign * i1 * j2) @* tmp1 j2 i1 in
  let tmp3 i1 = fftgen n2 (tmp2 i1) sign
  in
   (fun i -> tmp3 (i mod n1) (i / n1))
```

Y[i] **i1** **i2**

fftgen

$$Y[i_1 + i_2 n_1] = \sum_{j_2=0}^{n_2-1} \left[\left(\sum_{j_1=0}^{n_1-1} X[j_1 n_2 + j_2] \omega_{n_1}^{-i_1 j_1} \right) \omega_n^{-i_1 j_2} \right] \omega_{n_2}^{-i_2 j_2} . \quad (3)$$

```
let rec cooley_tukey n1 n2 input sign =
  let tmp1 j2 = fftgen n1
    (fun j1 -> input (j1 * n2 + j2)) sign in
  let tmp2 i1 j2 =
    exp n (sign * i1 * j2) @* tmp1 j2 i1 in
  let tmp3 i1 = fftgen n2 (tmp2 i1) sign
  in
   (fun i -> tmp3 (i mod n1) (i / n1))
```

Y[i] **i1** **i2**

fftgen

- How is it defined? **Mutually recursive definition**

fftgen n input sign =
If n %4 == 0: SR
If n is prime and gcd(n1,n2) = 1 PF….
If n is prime and gcd(n1,n2) != 1 cooley_tuckey …..
……..

```
let rec cooley_tukey n1 n2 input sign =
  let tmp1 j2 = fftgen n1
    (fun j1 -> input (j1 * n2 + j2)) sign in
  let tmp2 i1 j2 =
    exp n (sign * i1 * j2) @* tmp1 j2 i1 in
  let tmp3 i1 = fftgen n2 (tmp2 i1) sign
  in
   (fun i -> tmp3 (i mod n1) (i / n1))
```

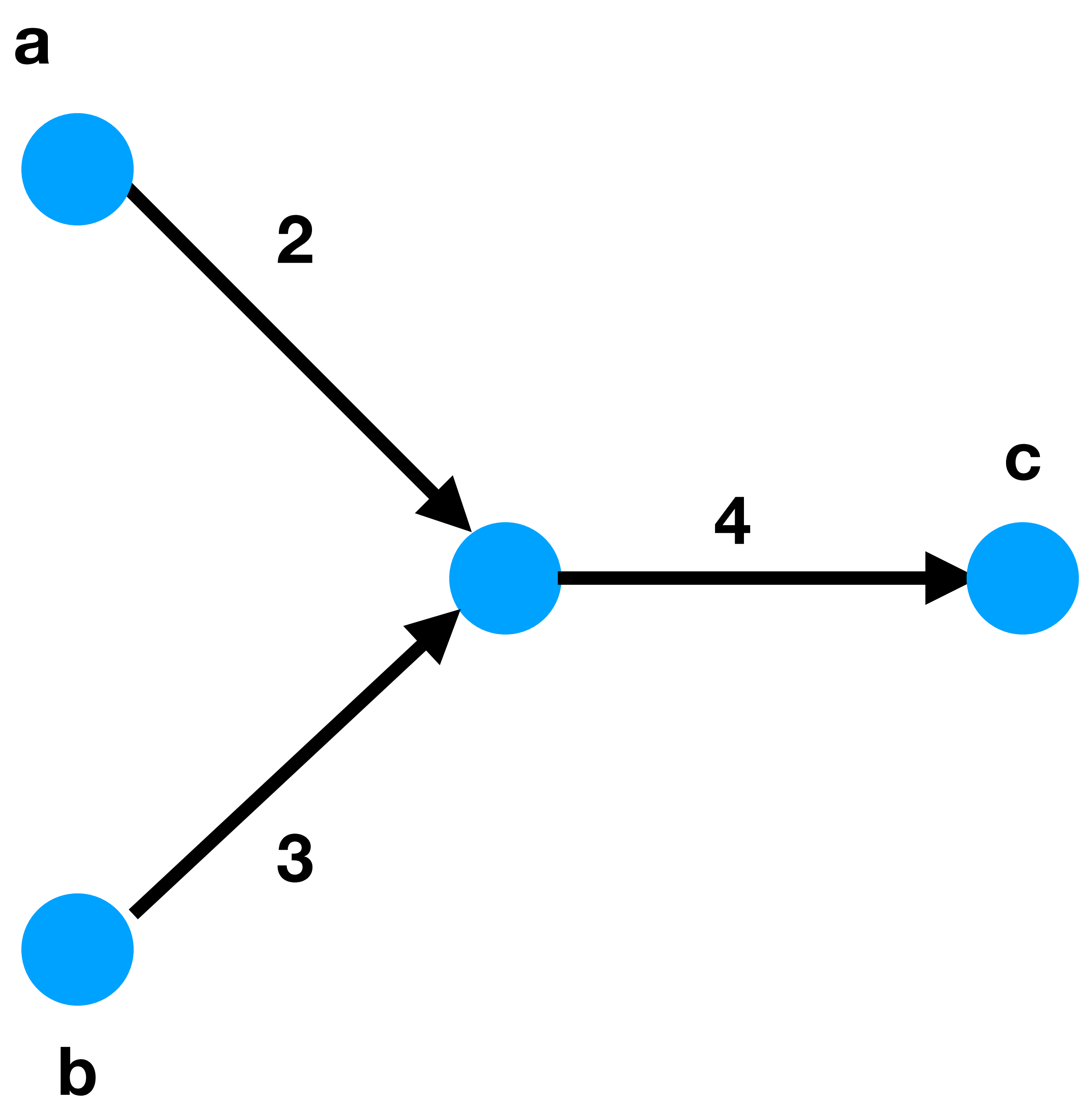
Simplifier

- Written in Monadic Style
- How is Common Subexpression Elimination implemented in modern compilers?

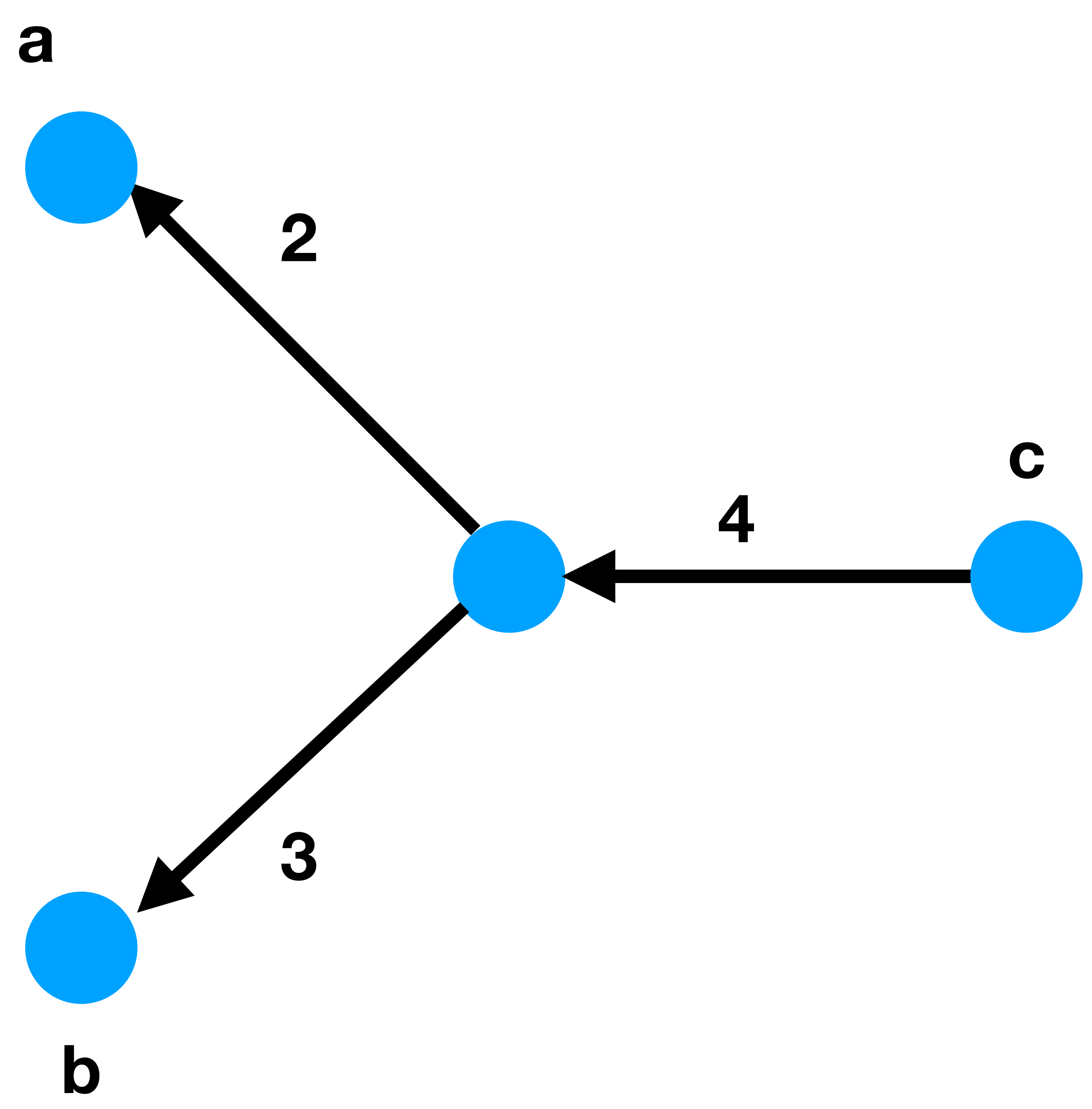
Available Expression Analysis - Data Flow Analysis (CS 526)

- Domain Specific Optimizations

DAG transposition

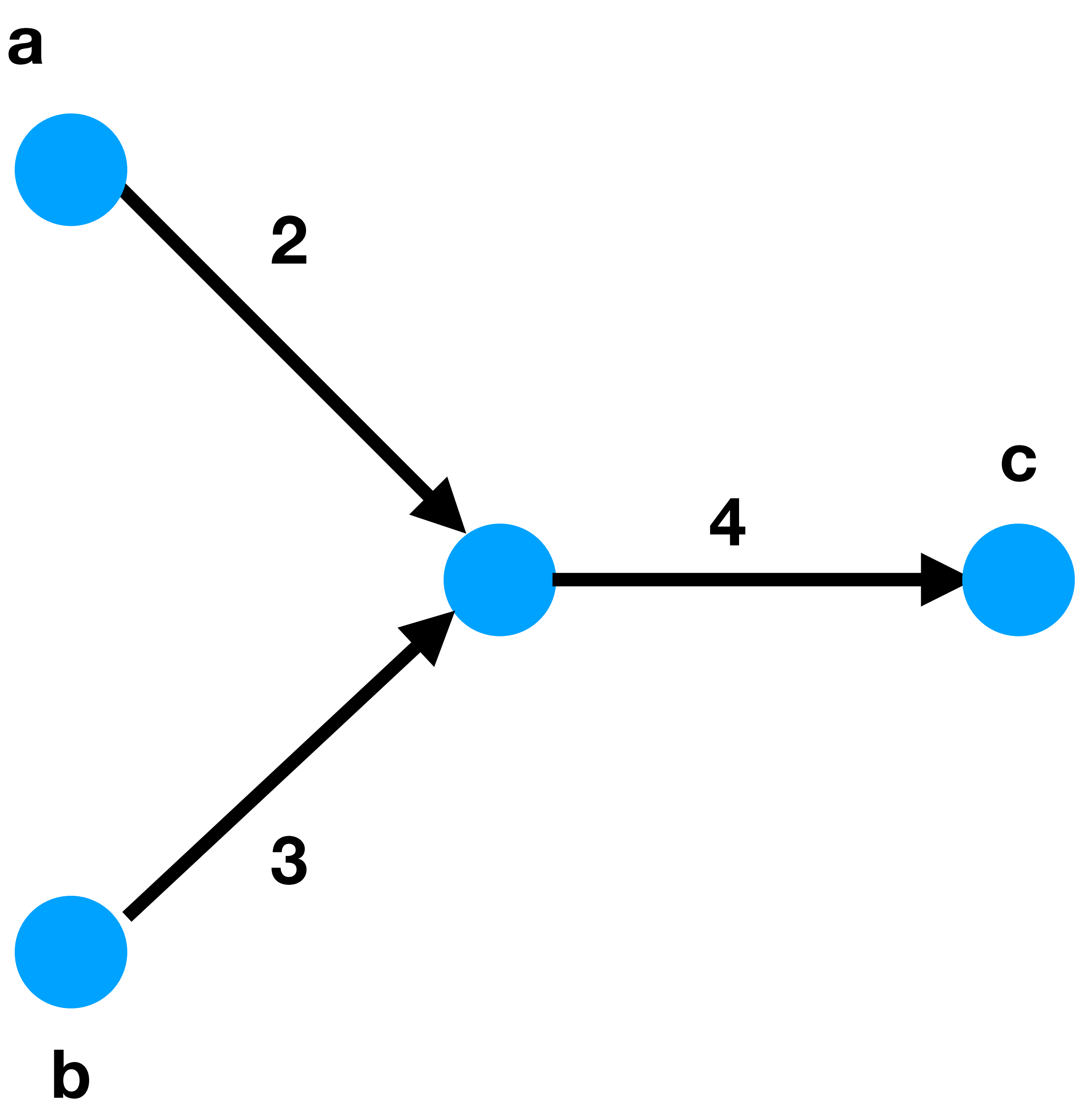


Original



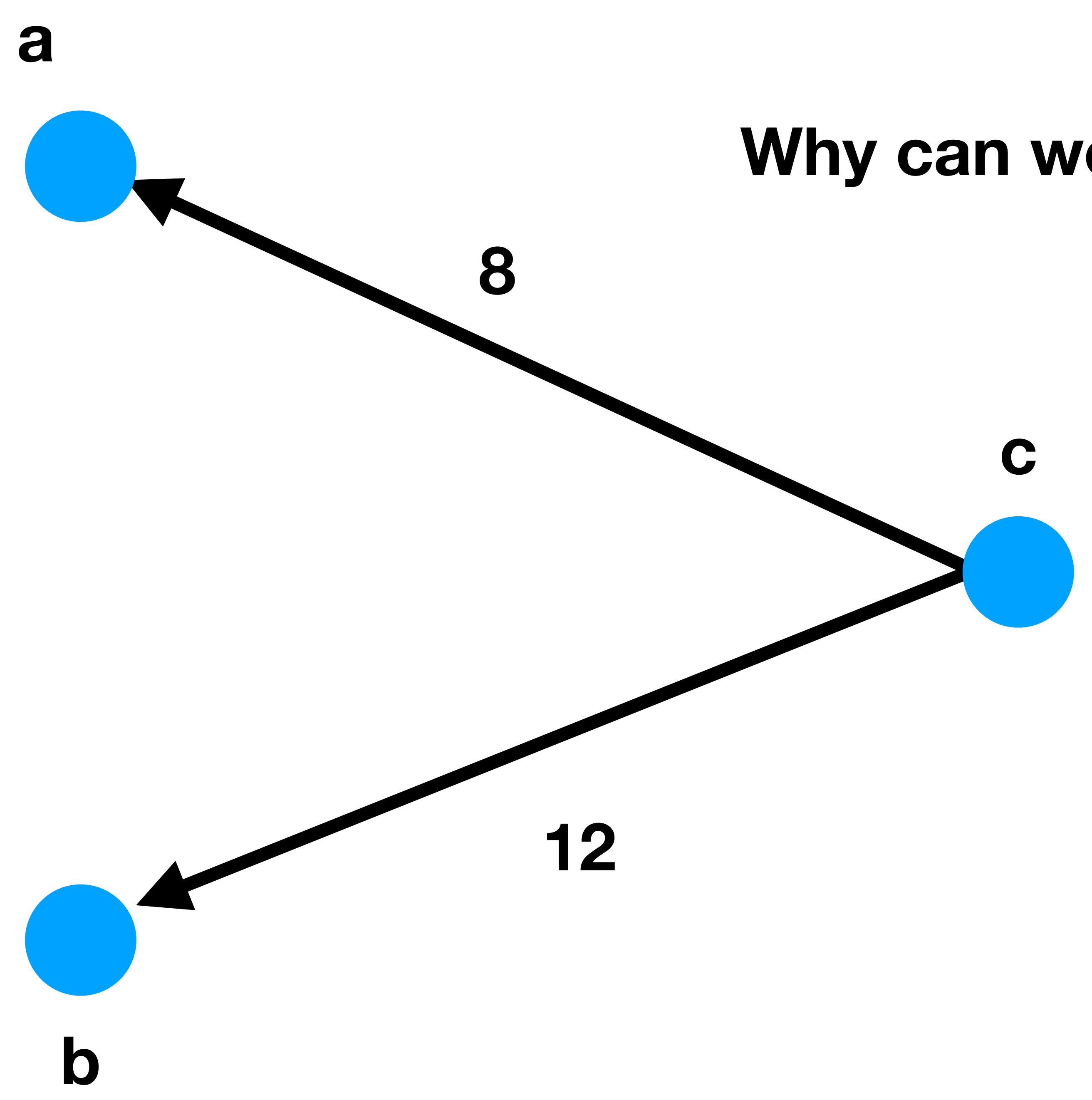
Transposed

DAG transposition



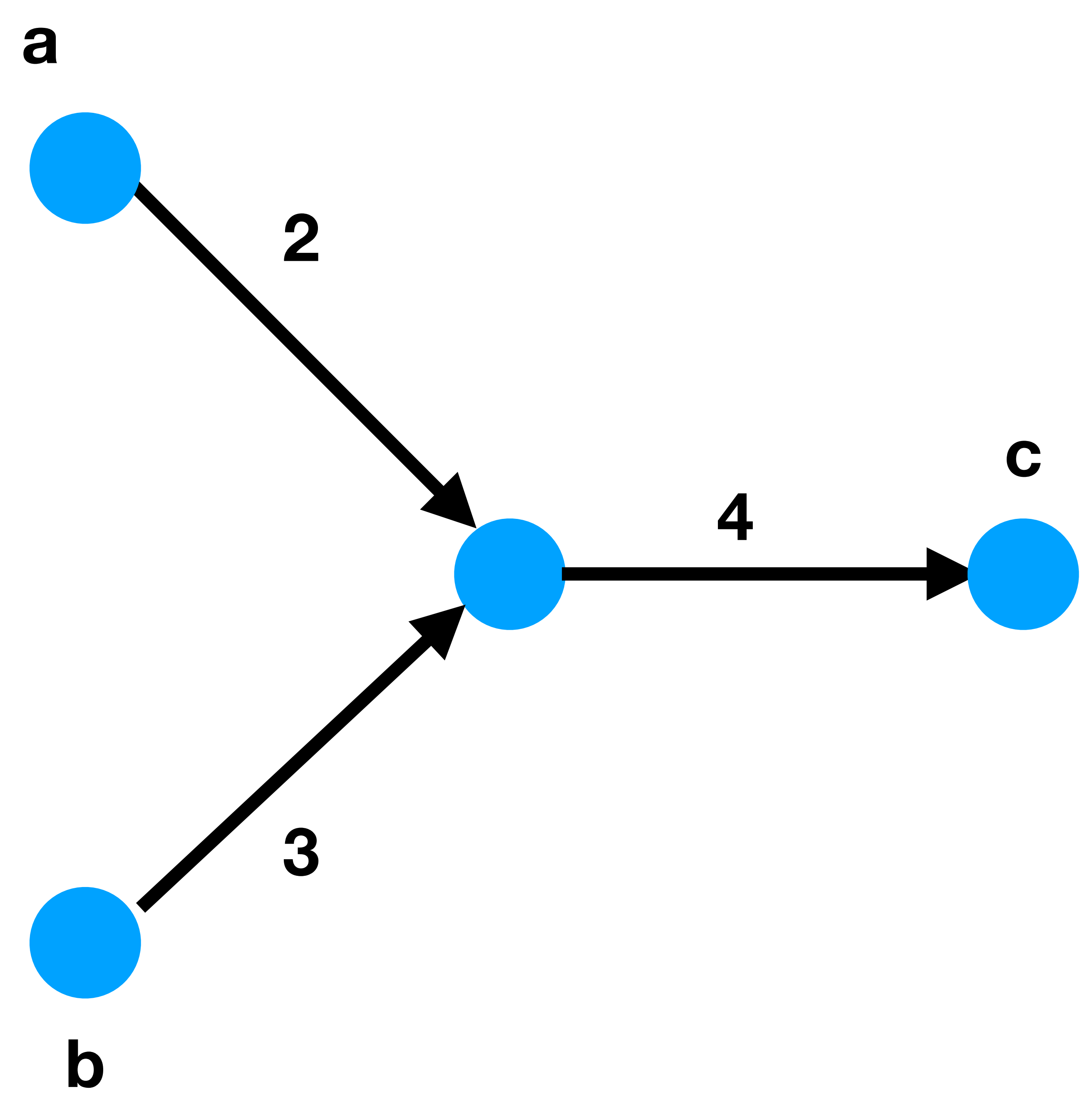
Original

Why can we do this?

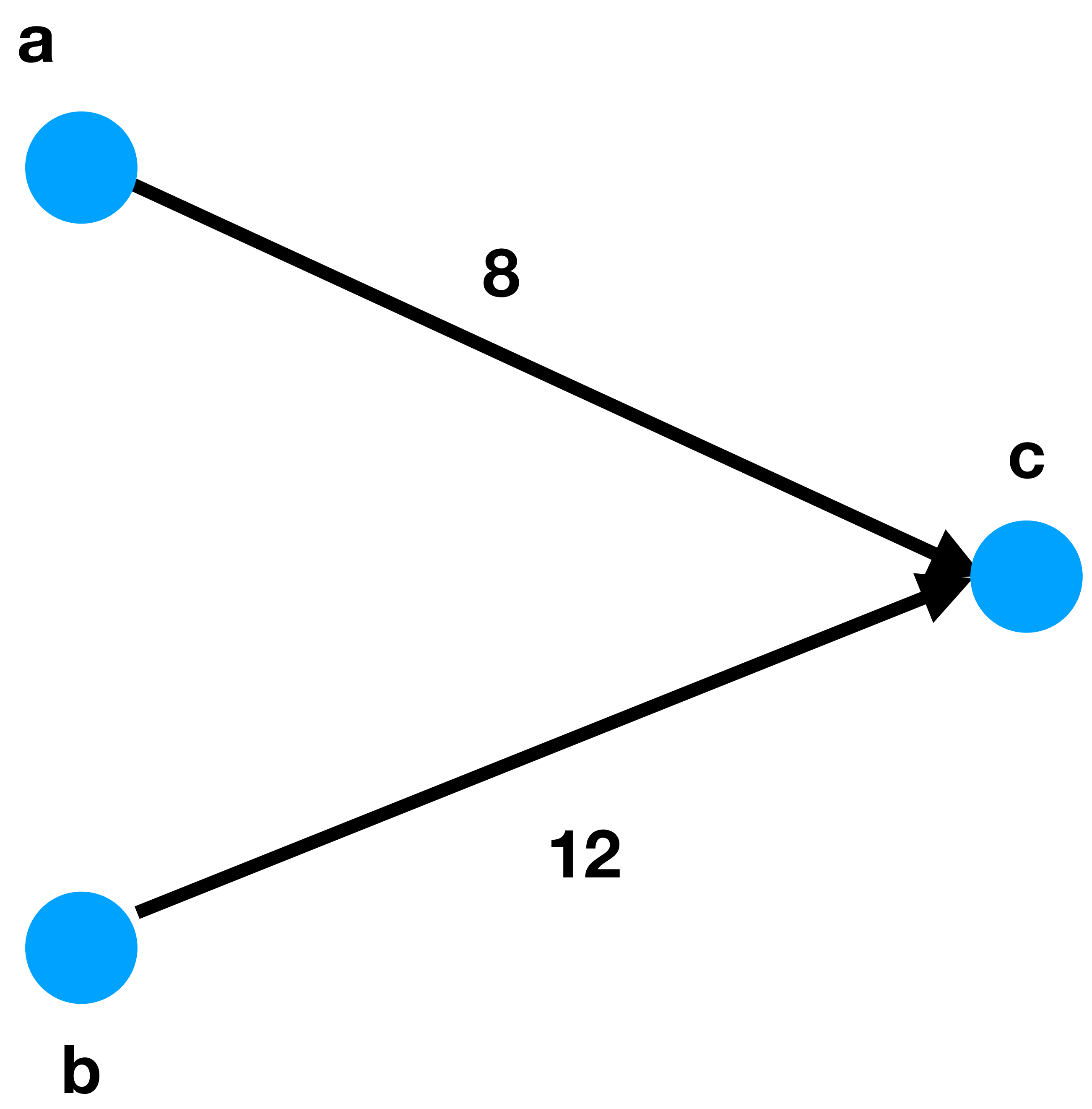


Transposed

DAG transposition

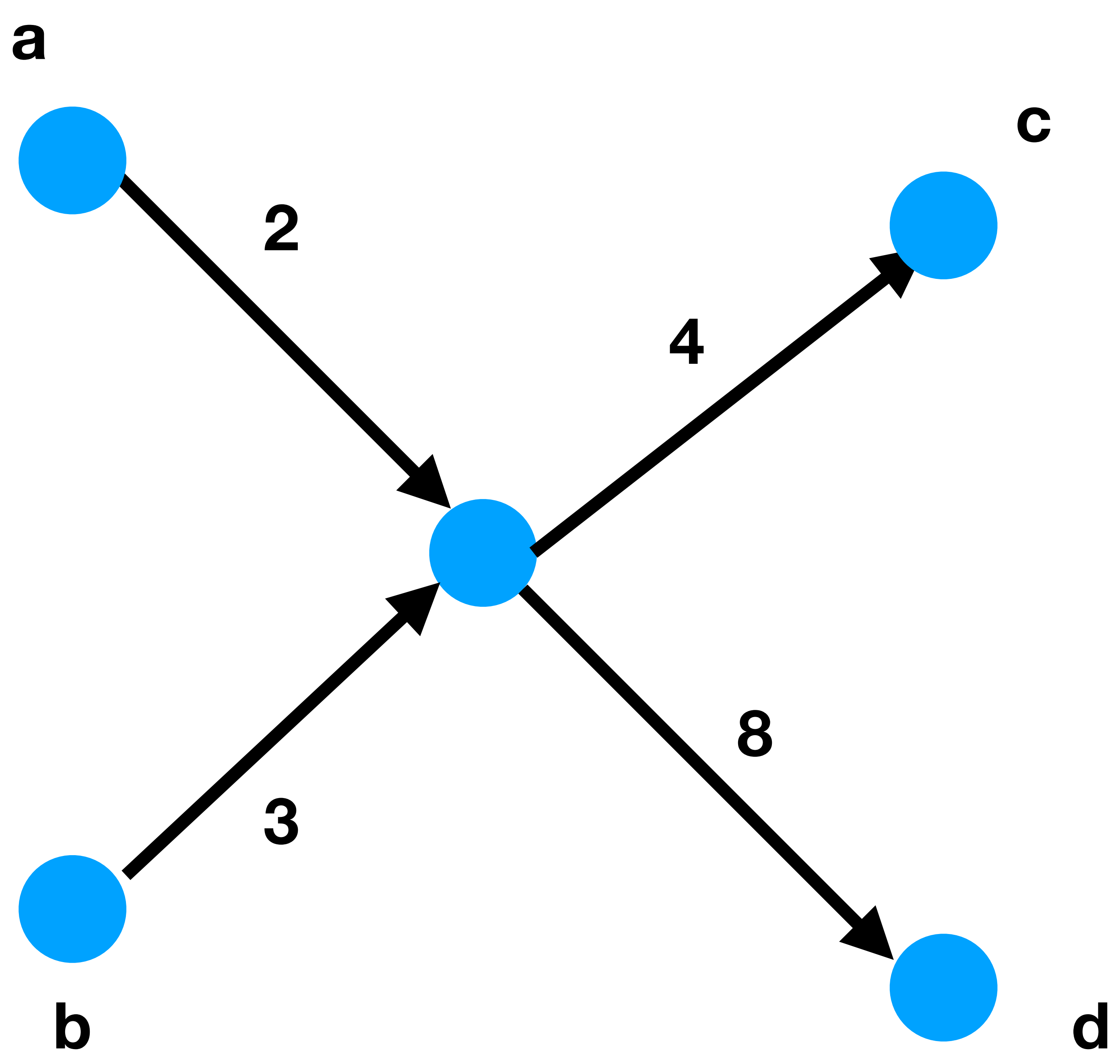


Original



Transpose again

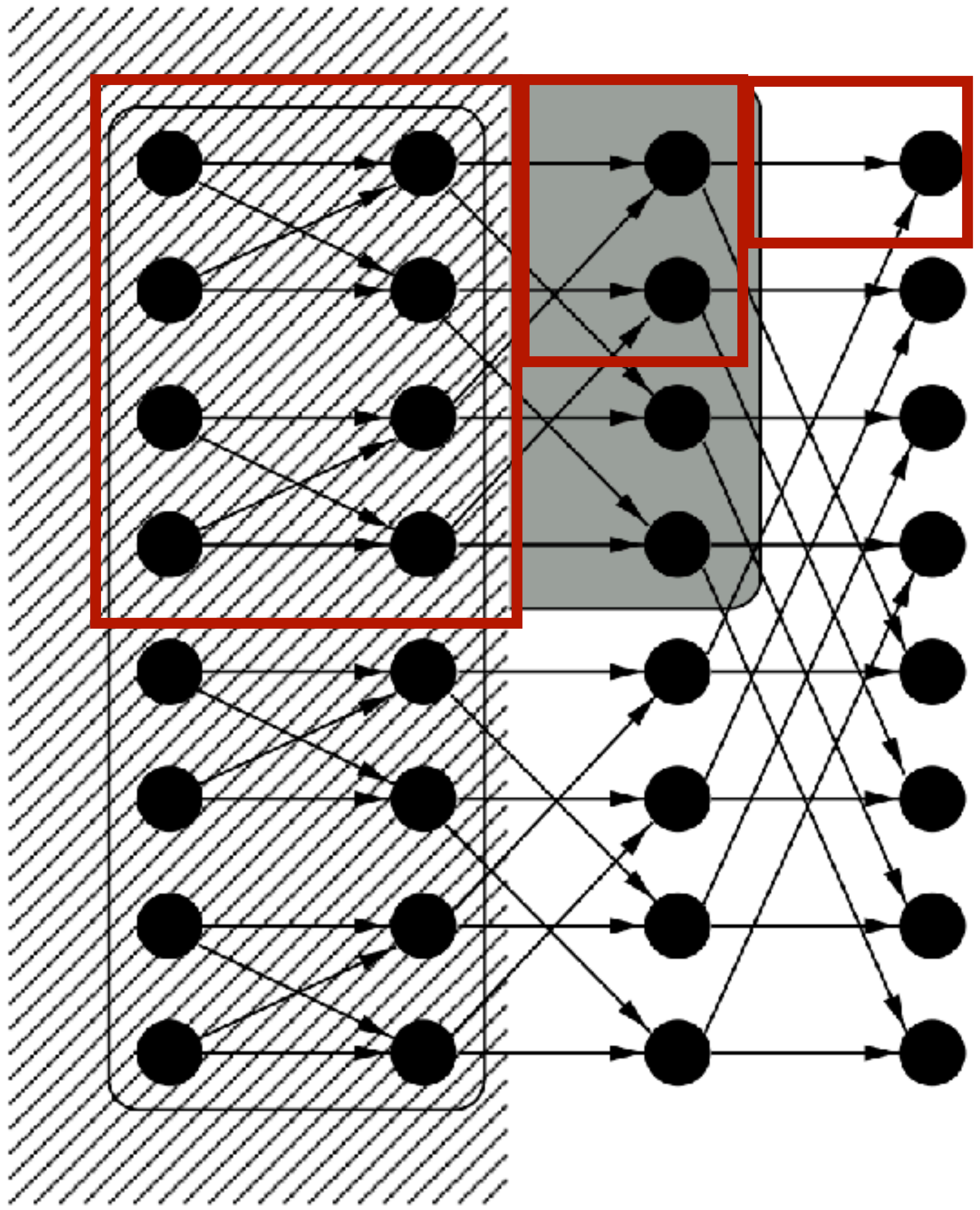
DAG transposition



Can we optimize this?

Scheduler

Why is the proposed algorithm cache oblivious (work for any C)?



What optimizations are missing?

$$Y[i_1 + i_2 n_1] = \sum_{j_2=0}^{n_2-1} \left[\left(\sum_{j_1=0}^{n_1-1} X[j_1 n_2 + j_2] \omega_{n_1}^{-i_1 j_1} \right) \omega_n^{-i_1 j_2} \right] \omega_{n_2}^{-i_2 j_2} \, . \tag{3}$$

- Vectorization - how?
- Parallelization - how?

Putting it into perspective

Simplifier

Common expressions
Algebraic identities

Heuristics

Number of addition and multiplication Instructions

Scheduler

Dependency preserving Schedules

Optimal*

Number of register spills

Optimizations are designed manually :(

Where is auto-tuning?

- Even in this paper there are opportunities to remove heuristics - What?
- Now that you understand the domain specific compiler and optimization design, you are ready to read the planning paper
- Exercise: Read Frigo and Johnson “The Design and Implementation of FFTW3” 2015
- Space: N-D FFTs, how to decide which slices are computed in what order?
- Approach: Decide plans beforehand and exhaustively run all decided plans on hardware to determine the best performing one

Extensions (From Your Responses)

- GPUs, Vectorization, Distributed - new hardware targets
- Instead of C lower it to LLVM IR - Why?
- Better and newer optimizations - What?
- Algorithm selection learned
- Better Cost Function