GPU TECHNOLOGY CONFERENCE

Graph Cuts with CUDA

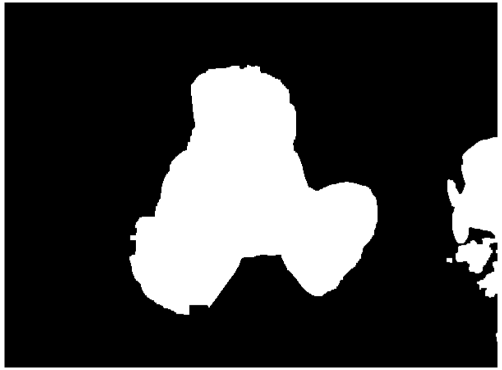
San Jose| 02/10/09 | Timo Stich

Outline

- Introduction
- Algorithms to solve Graph Cuts
- CUDA implementation
- Image processing application
- Summary

Problems solvable with Graphcuts

Stereo Depth Estimation



Binary Image Segmentation

Photo Montage (aka Image Stitching)

Source: MRF Evaluation, Middlebury College

Energy Minimization

- Graphcut finds global minimum

Sum over all Pixels of an Image

Sum over all neighborhoods

$$E(L) = \sum_{x} D_x(L_x) + \sum_{(x,y)\in \mathrm{N}} V(| L_x - L_y |)$$

Data Term:
Measures fitting of
label to pixel

Neighborhood Term:
Penalizes different labelings
for neighbors

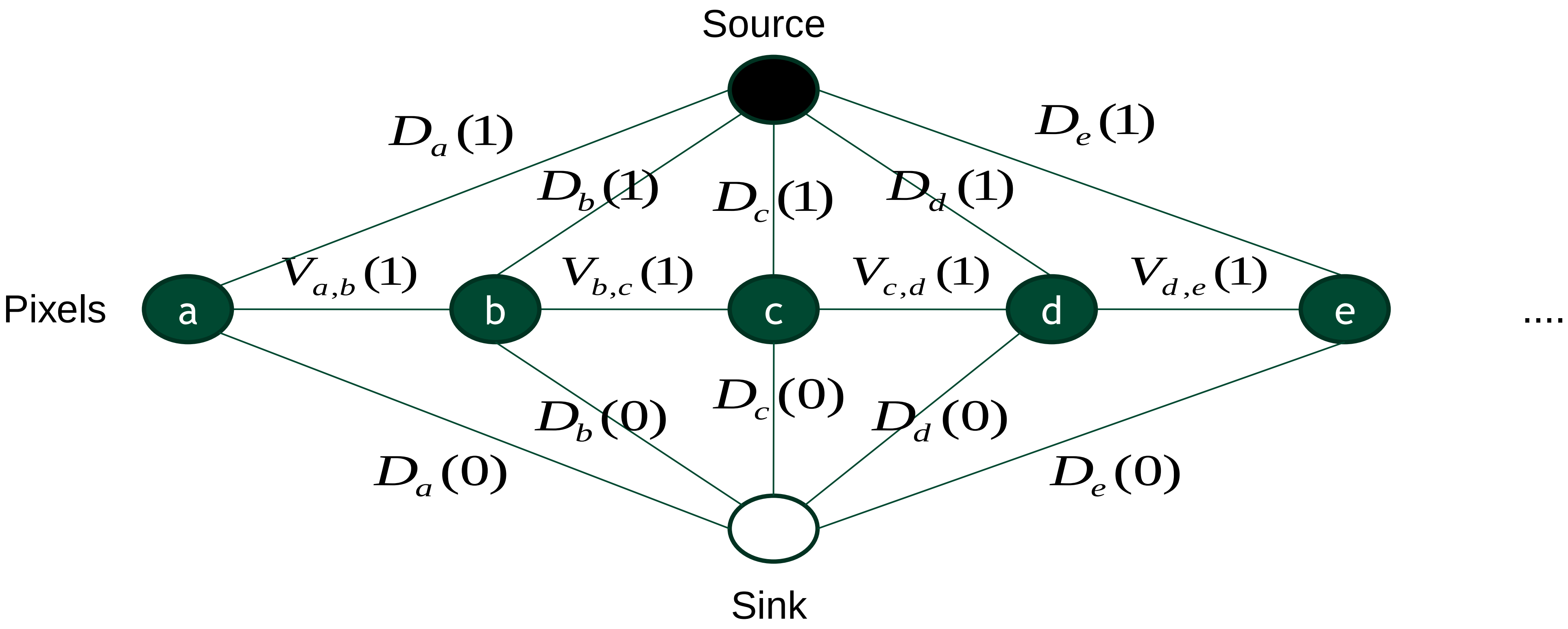
Example:
Binary Segmentation Problem

0	0	0	0
?	?	?	?
?	?	?	?
?	?	?	1

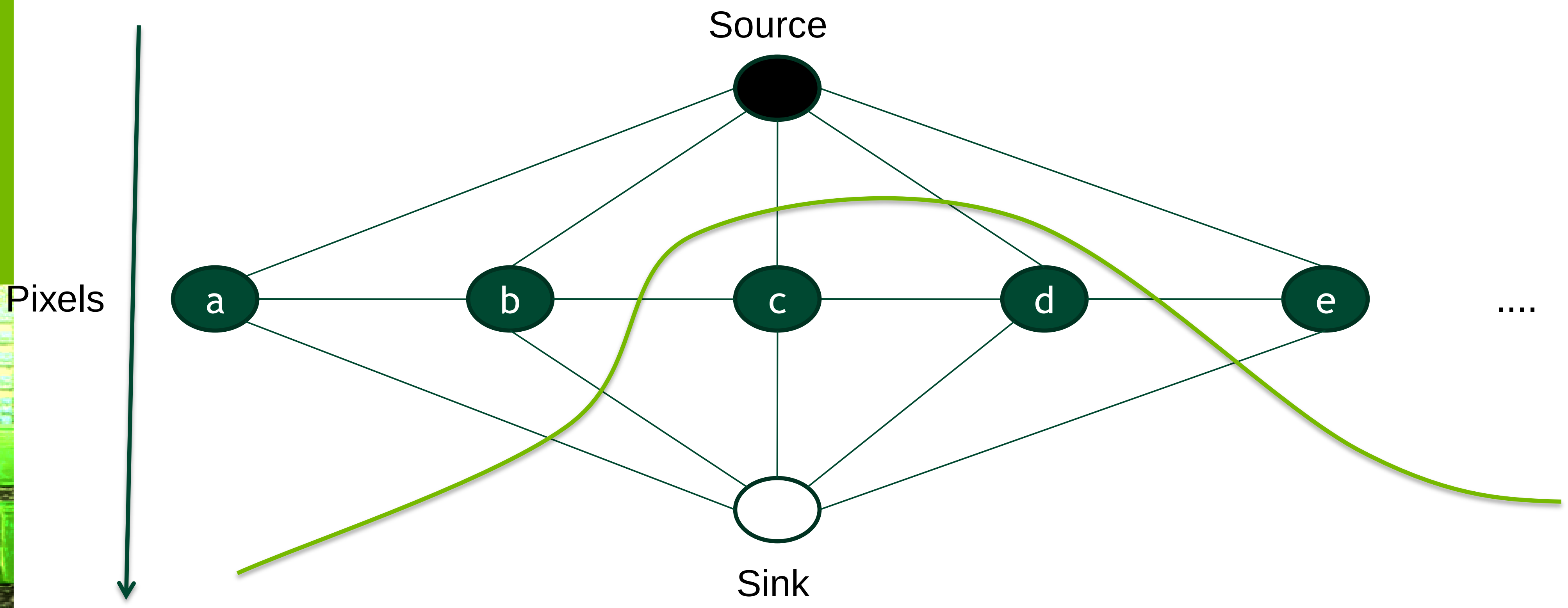
User marks ***some*** pixels as
Background and Foreground

Compute ***for all*** pixels if they
are Background or Foreground

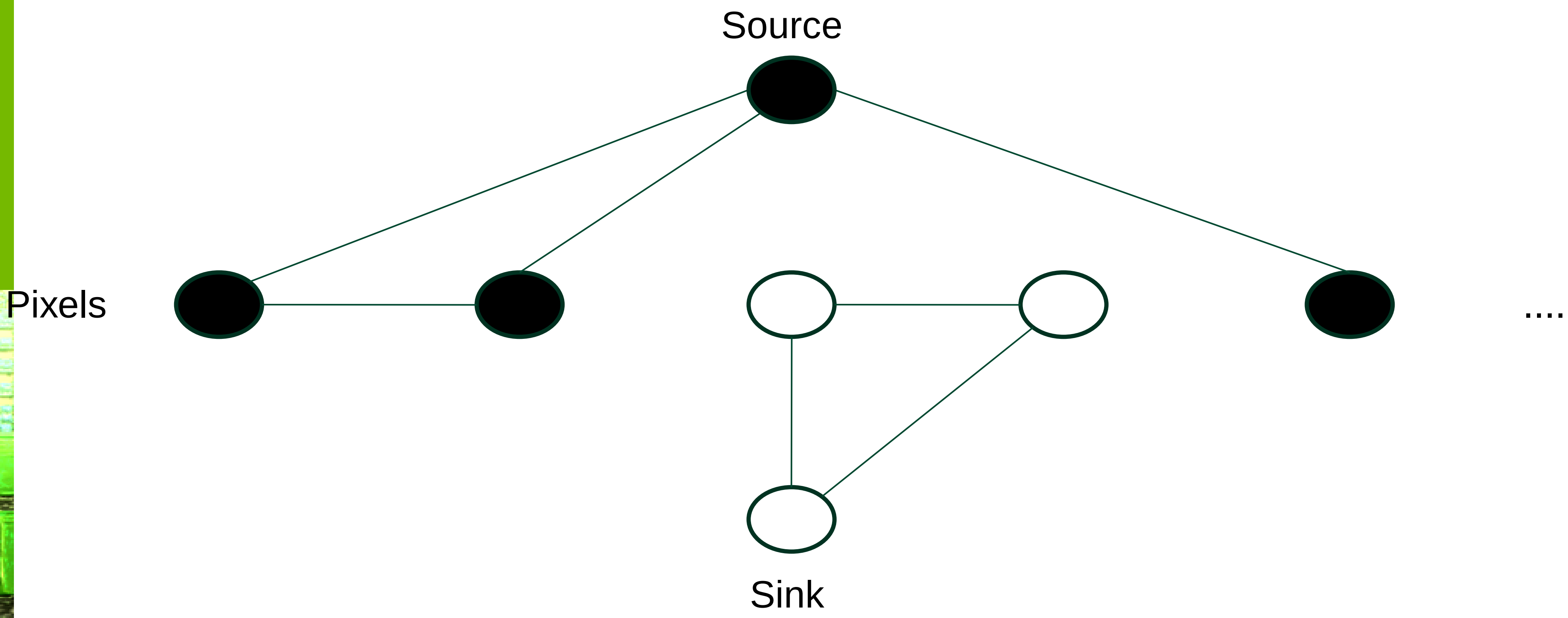
Building a Flow Graph for the Problem



Maximum Flow = Minimum Cut

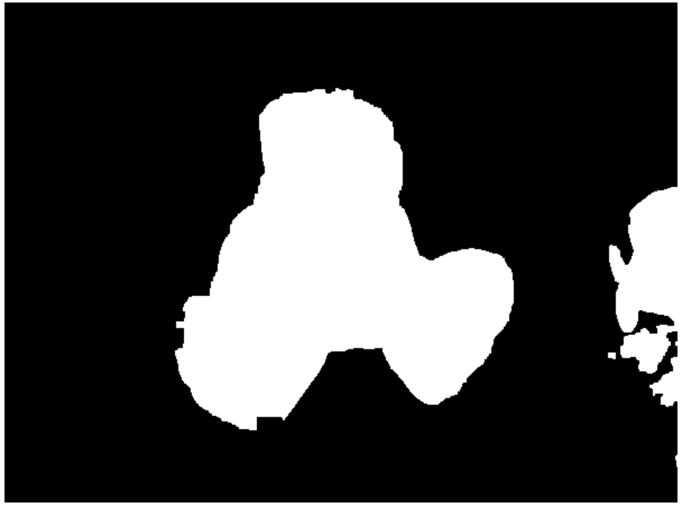


Graph Cut Solution



Graph Cut Solution

Input



Result

Graph Cut Algorithms

- Ford-Fulkerson
  - – Find augmenting paths from source to sink
  - – Global scope, based on search trees
  - – Most used implementation today by Boykov et al.
- Goldberg-Tarjan (push-relabel)
  - – Considers one node at a time
  - – Local scope, only direct neighbors matter
  - – Inherently parallel, good fit for CUDA

Push-Relabel in a nutshell

- Some definitions
  - Each node x:
    - Has excess flow u(x) and height h(x)
    - Outgoing edges to neighbors (x,*) with capacity c(x,*)
  - Node x is active: if u(x)> 0 and h(x)< HEIGHT_MAX
  - Active node x
    - can push to neighbor y: if c(x,y) > 0, h(y) = h(x) – 1
    - is relabeled: if for all c(x,*) > 0, h(*) >= h(x)

Push Pseudocode

```
void push(x, excess_flow, capacity, const height)
    if active(x) do
        foreach y=neighbor(x)
            if height(y) == height(x) – 1 do                // check height
                flow = min( capacity(x,y), excess_flow(x));    // pushed flow
                excess_flow(x) -= flow; excess_flow(y) += flow;  // update excess flow
                capacity(x,y) -= flow; capacity(y,x) += flow;   // update edge cap.
            done
        end
    done
```

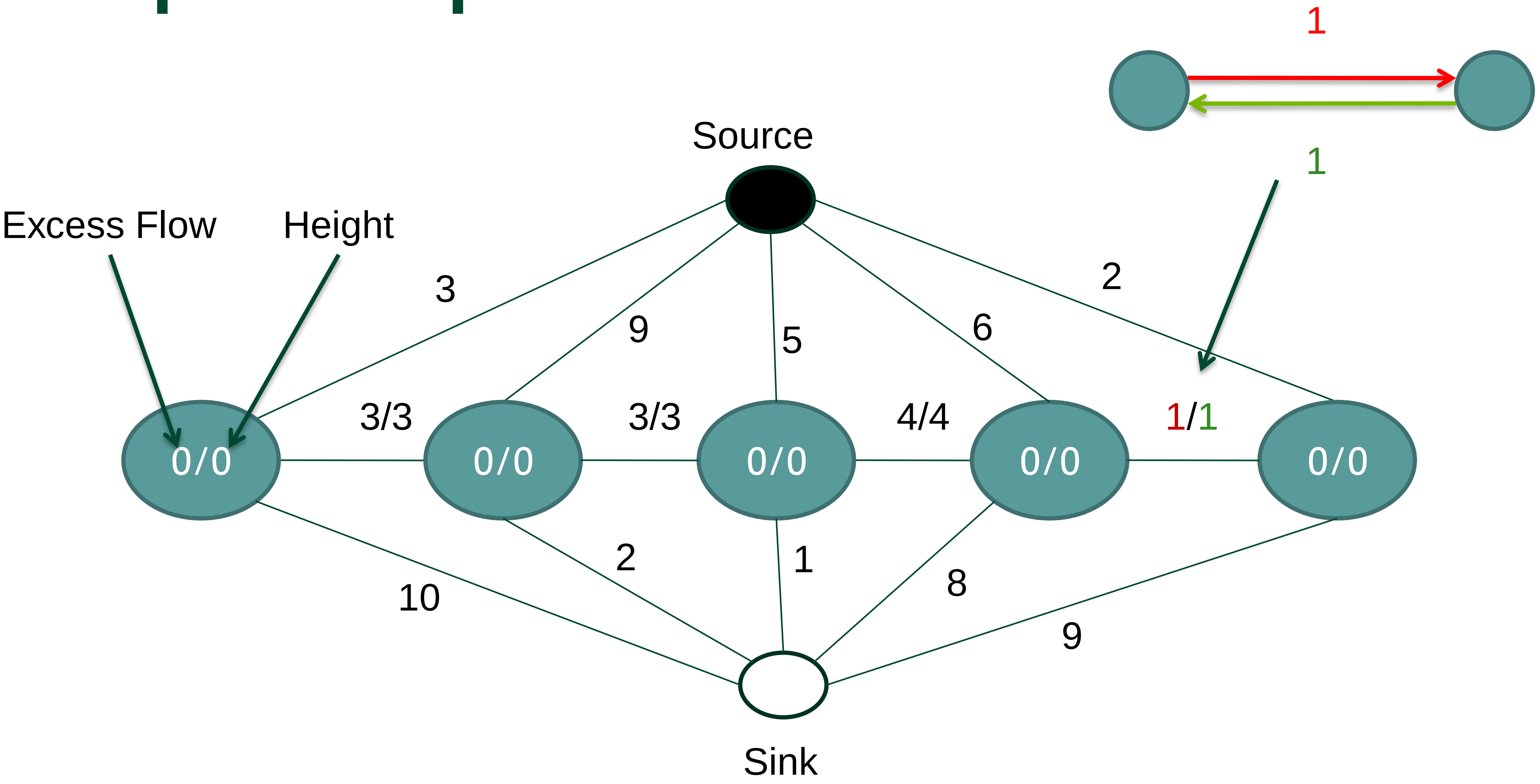
Relabel Pseudocode

```
void relabel(x, height, const excess_flow, const capacity)
    if active(x) do
        my_height = HEIGHT_MAX;                              // init to max height
        foreach y=neighbor(x)
            if capacity(x,y) > 0 do
                my_height = min(my_height, height(y)+1);     // minimum height + 1
            done
        end
        height(x) = my_height;                               // update height
    done
```

Push-Relabel Pseudocode

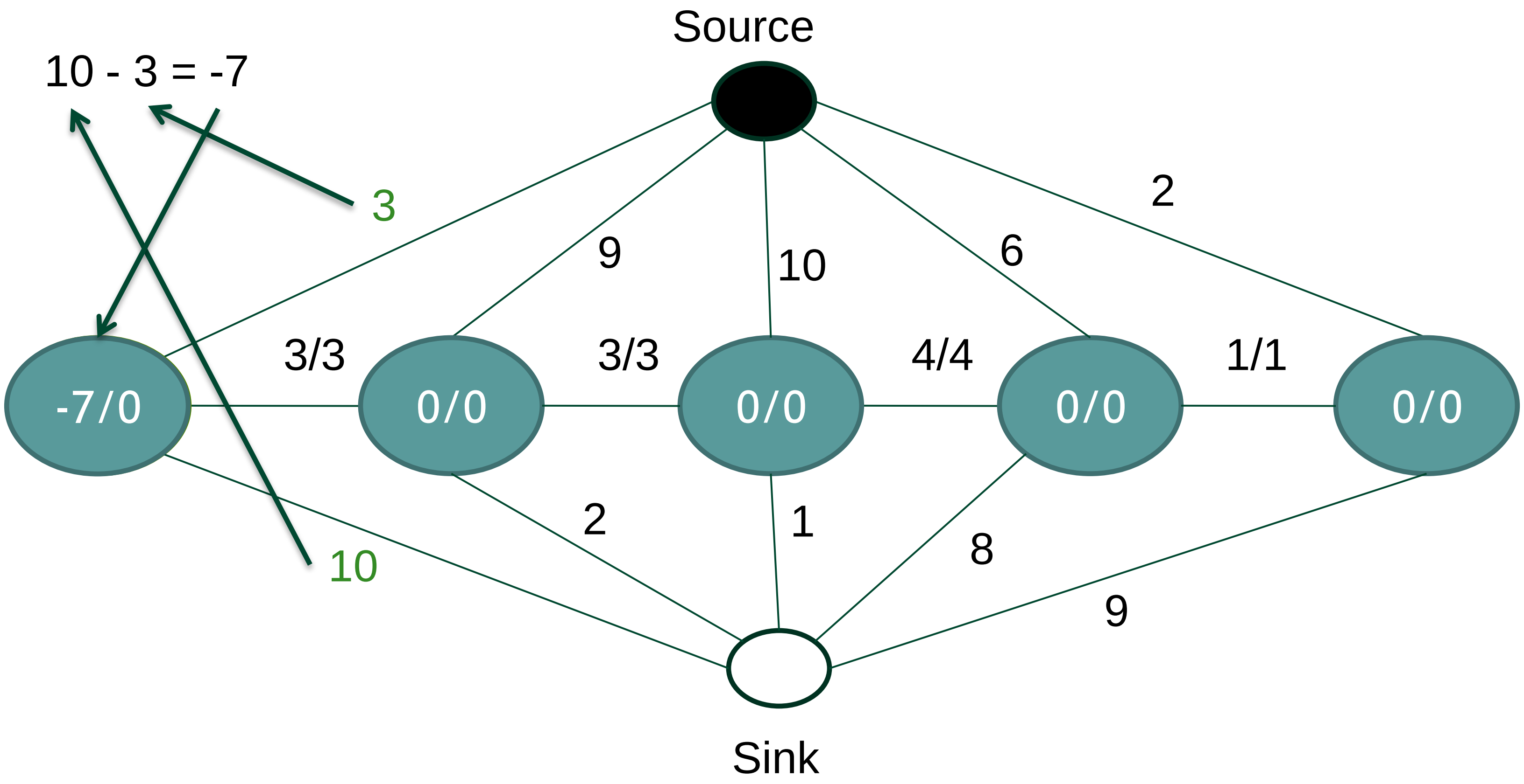
```
while any_active(x) do
        foreach x
            relabel(x);
        end
        foreach x
            push(x);
        end
done
```

Graph setup


1
1
Source
Excess Flow
Height
3
9
5
6
2
3/3
3/3
4/4
1/1
0/0
0/0
0/0
0/0
0/0
10
2
1
8
9
Sink

Direct Push

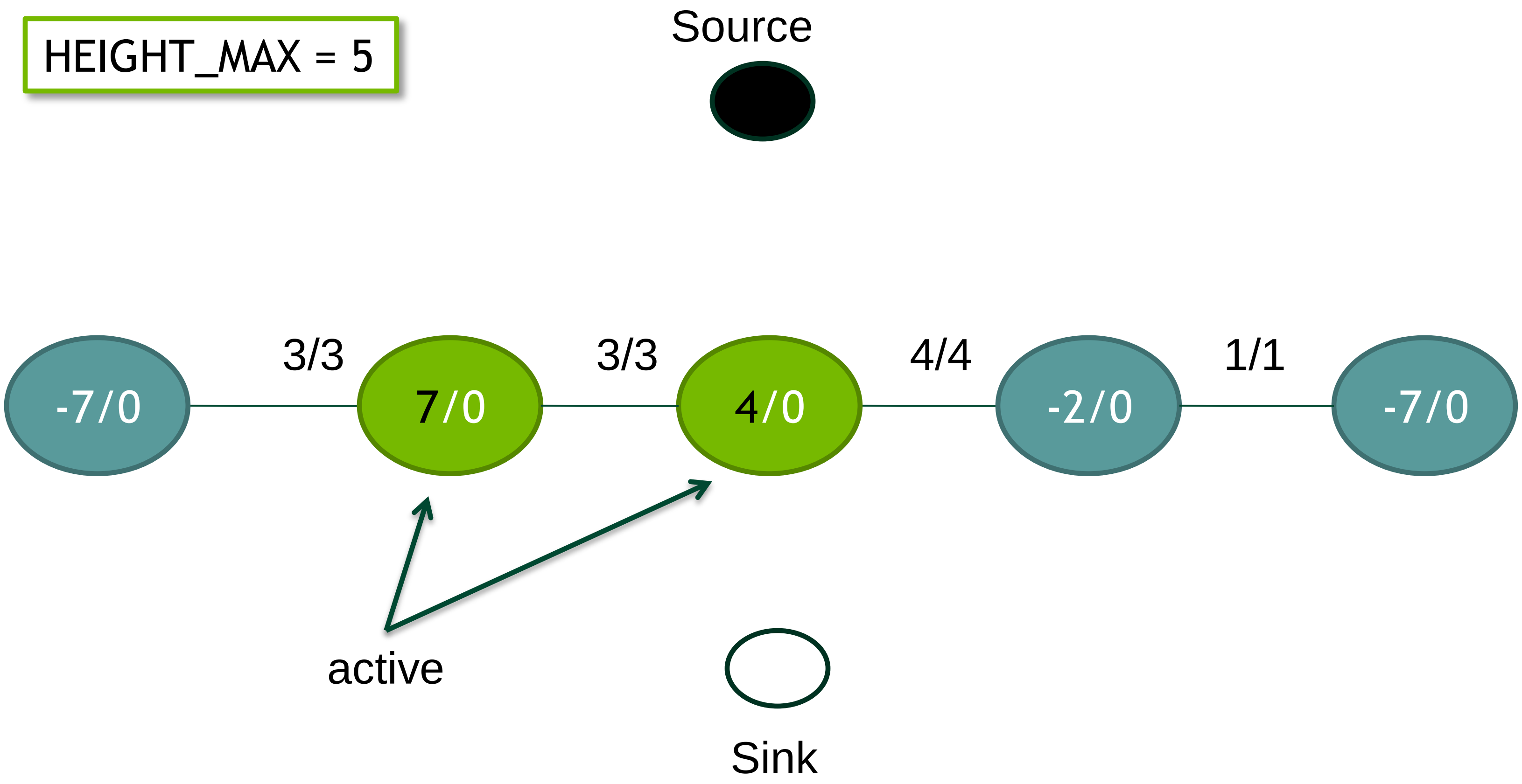
Total flow = 0



Initialized

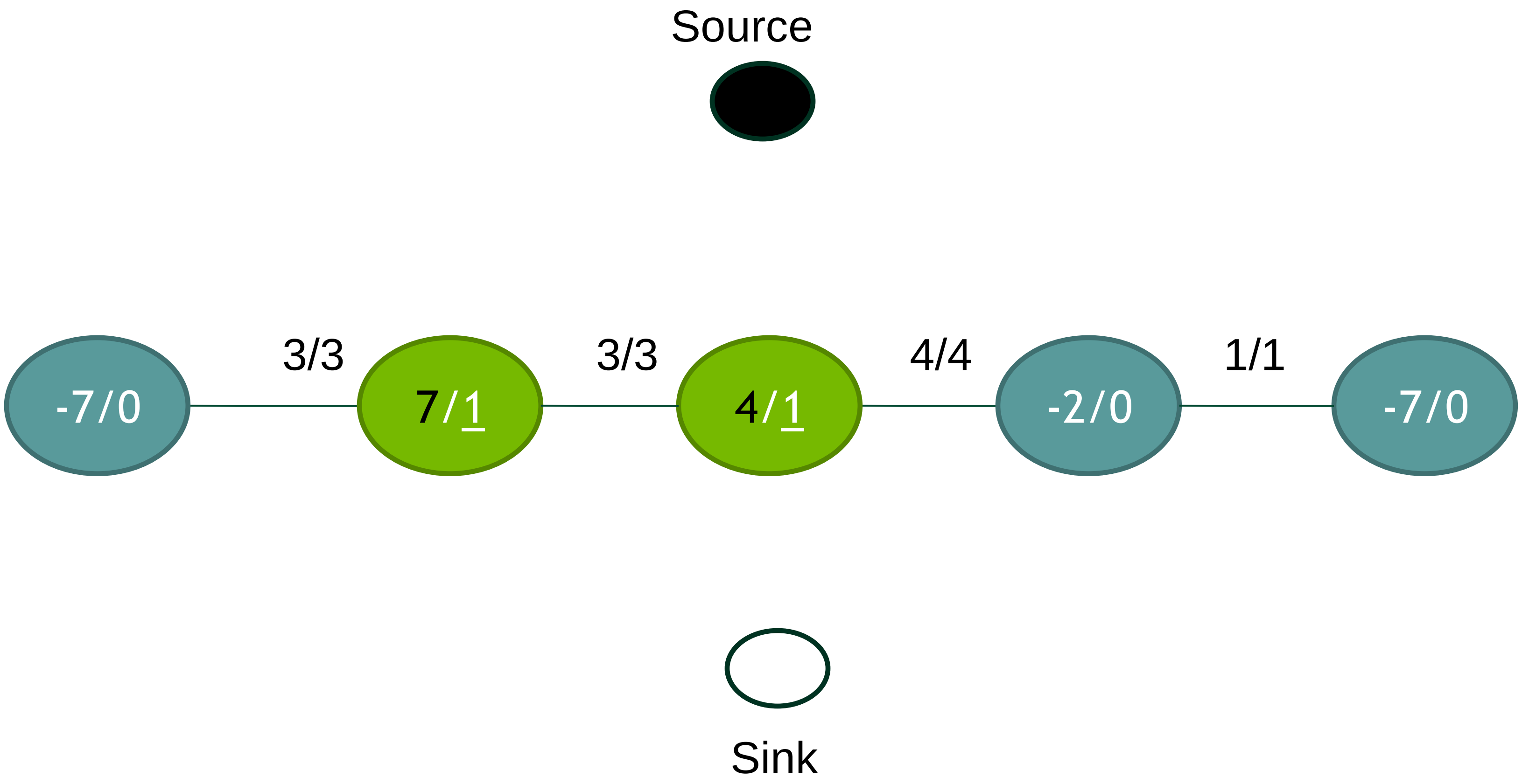
Total flow = 14

HEIGHT_MAX = 5



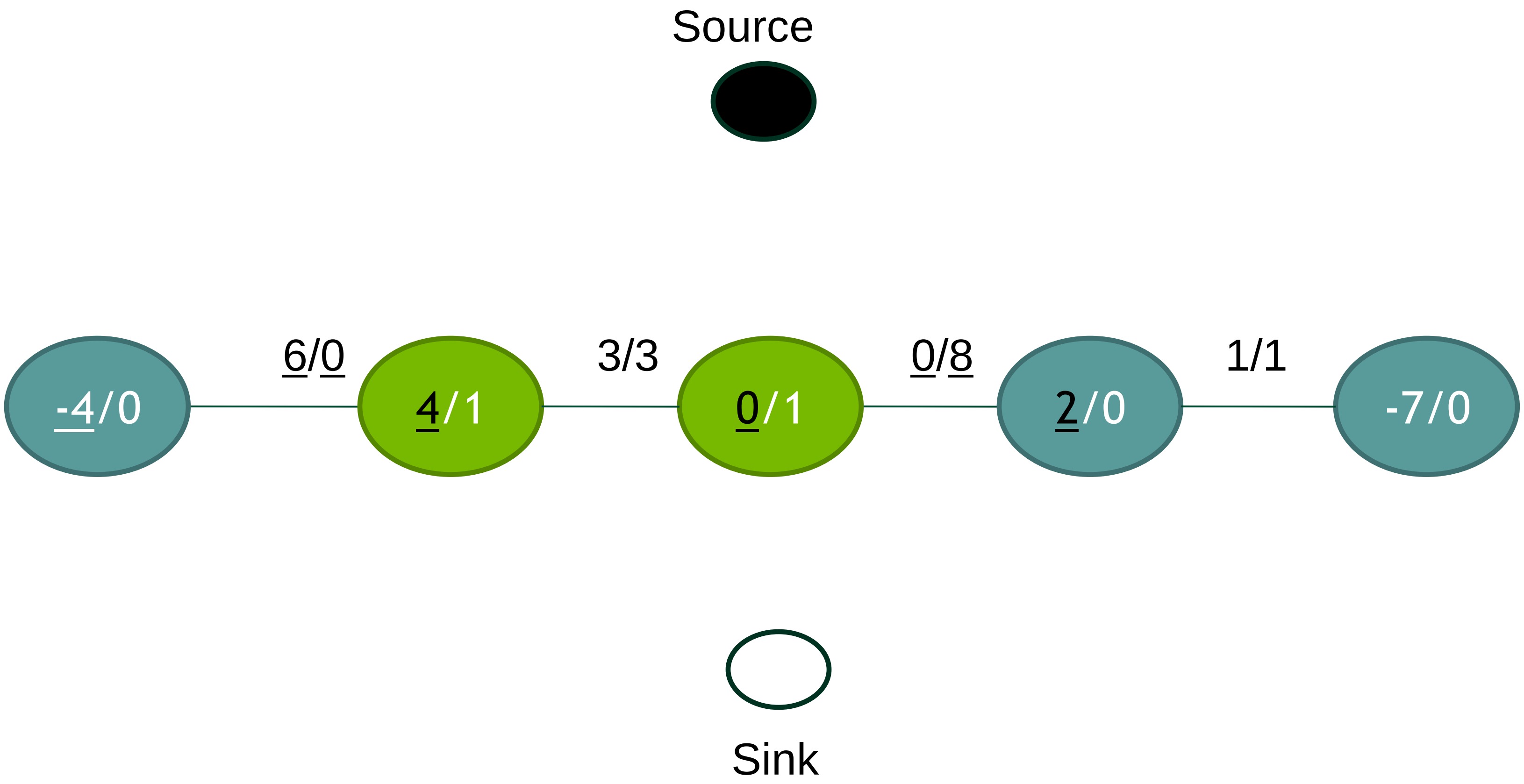
After Relabel

Total flow = 14



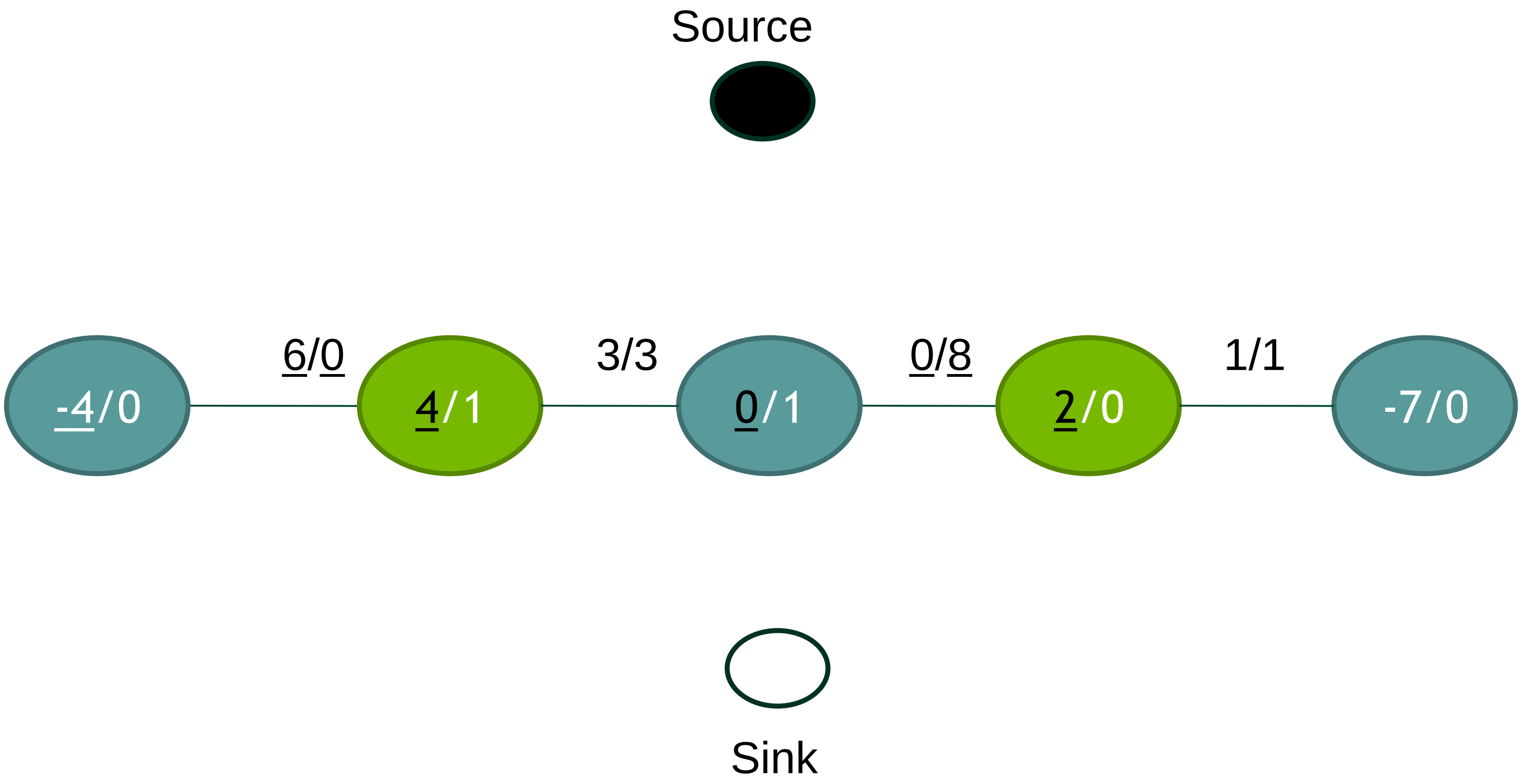
After Push

Total flow = 19



2nd iteration

Total flow = 19



Source

-4/0

6/0

4/1

3/3

0/1

0/8

2/0

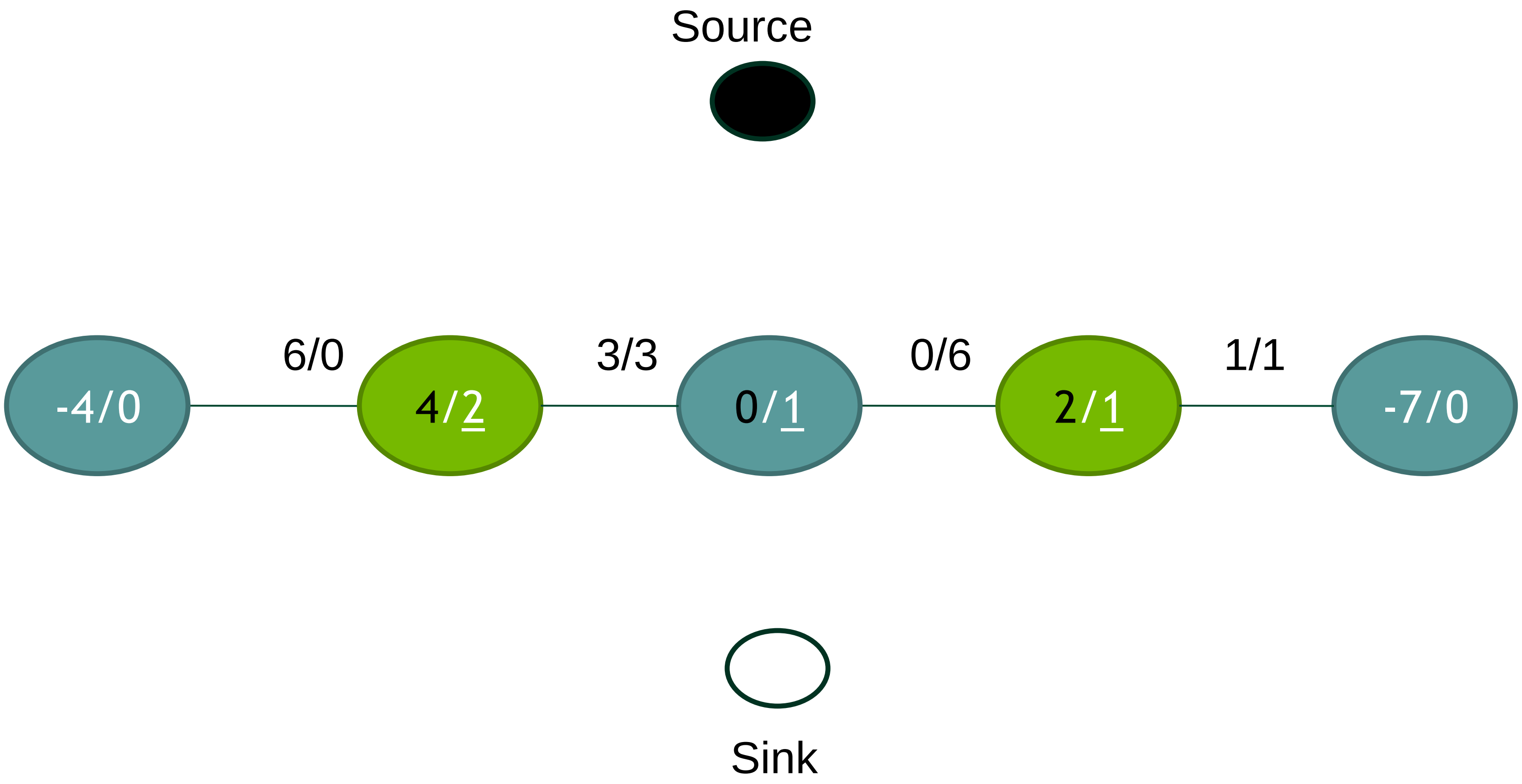
1/1

-7/0

Sink

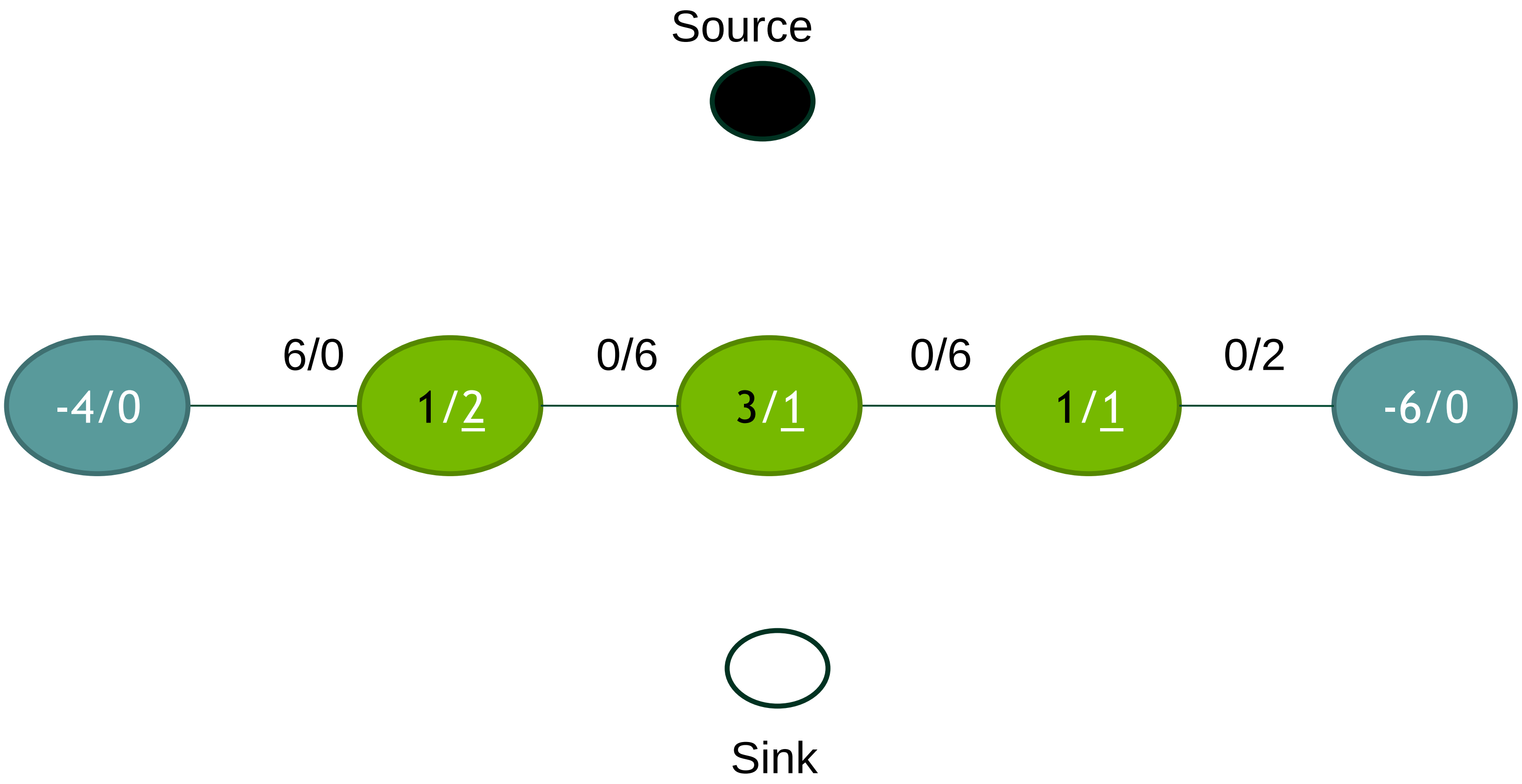
After Relabel

Total flow = 19



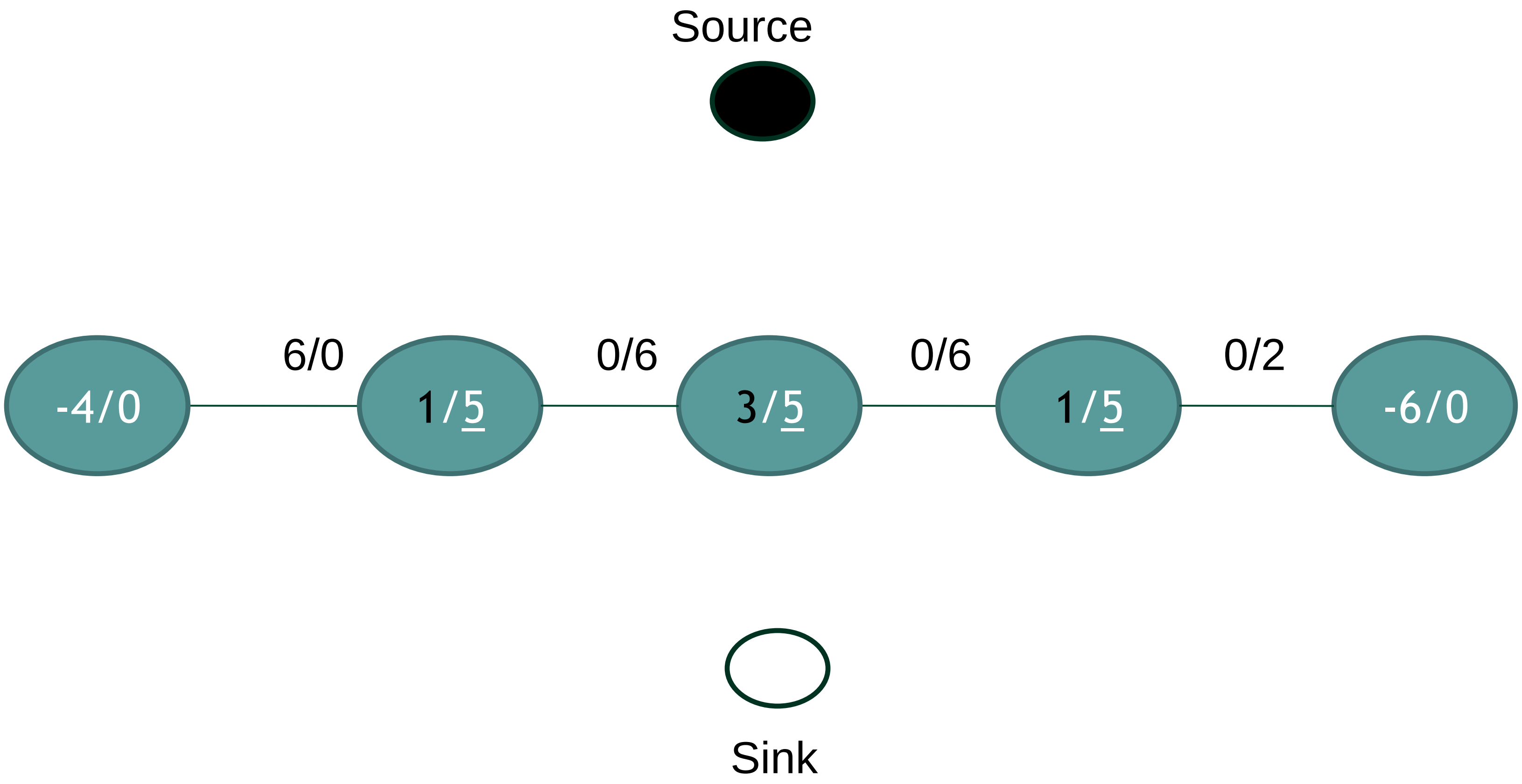
After Push

Total flow = 20



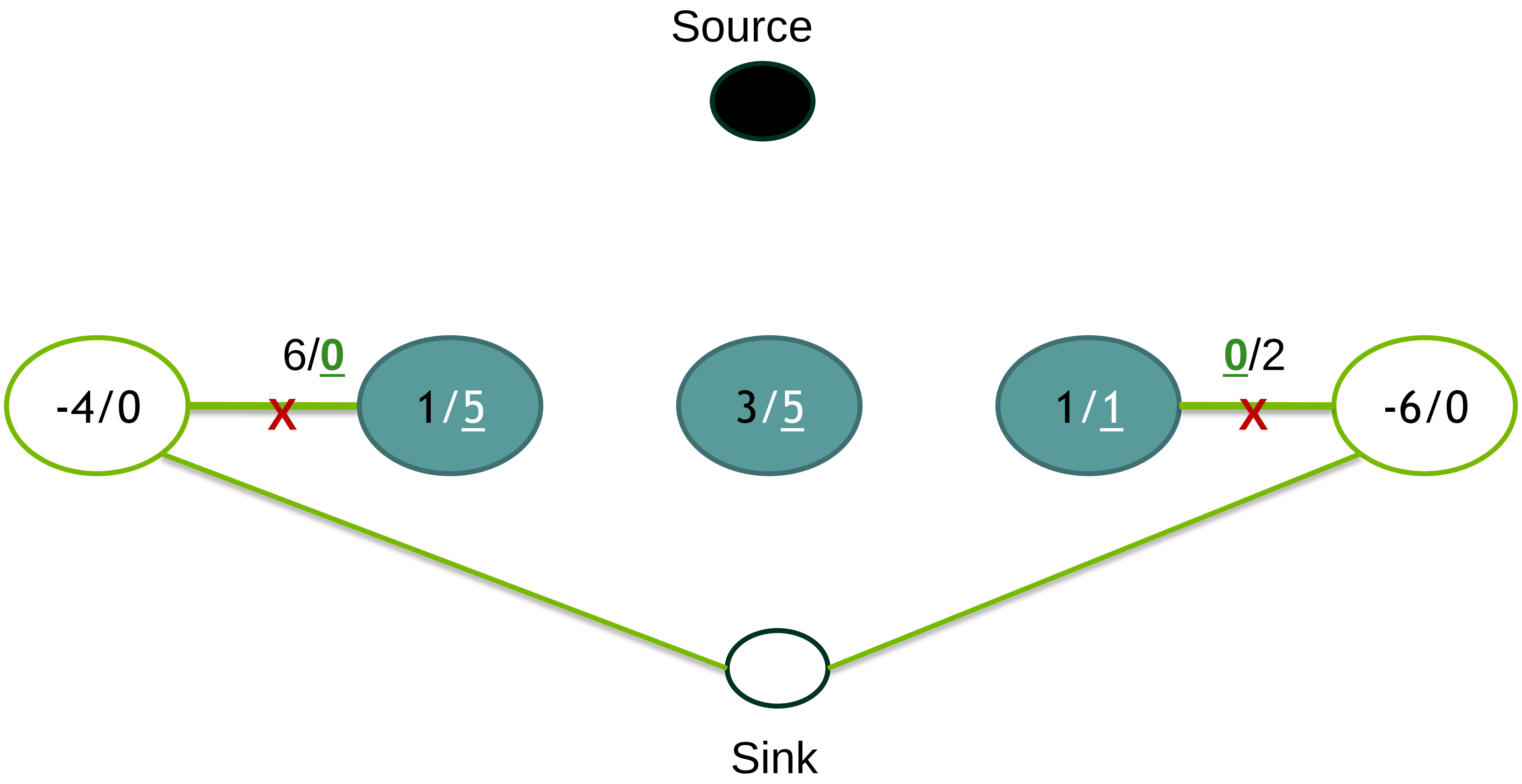
After 3 more Iterations, Terminated

Total flow = 20



Inverse BFS from Sink

Total flow = 20



Source

-4/0

6/0

X

1/5

3/5

1/1

0/2

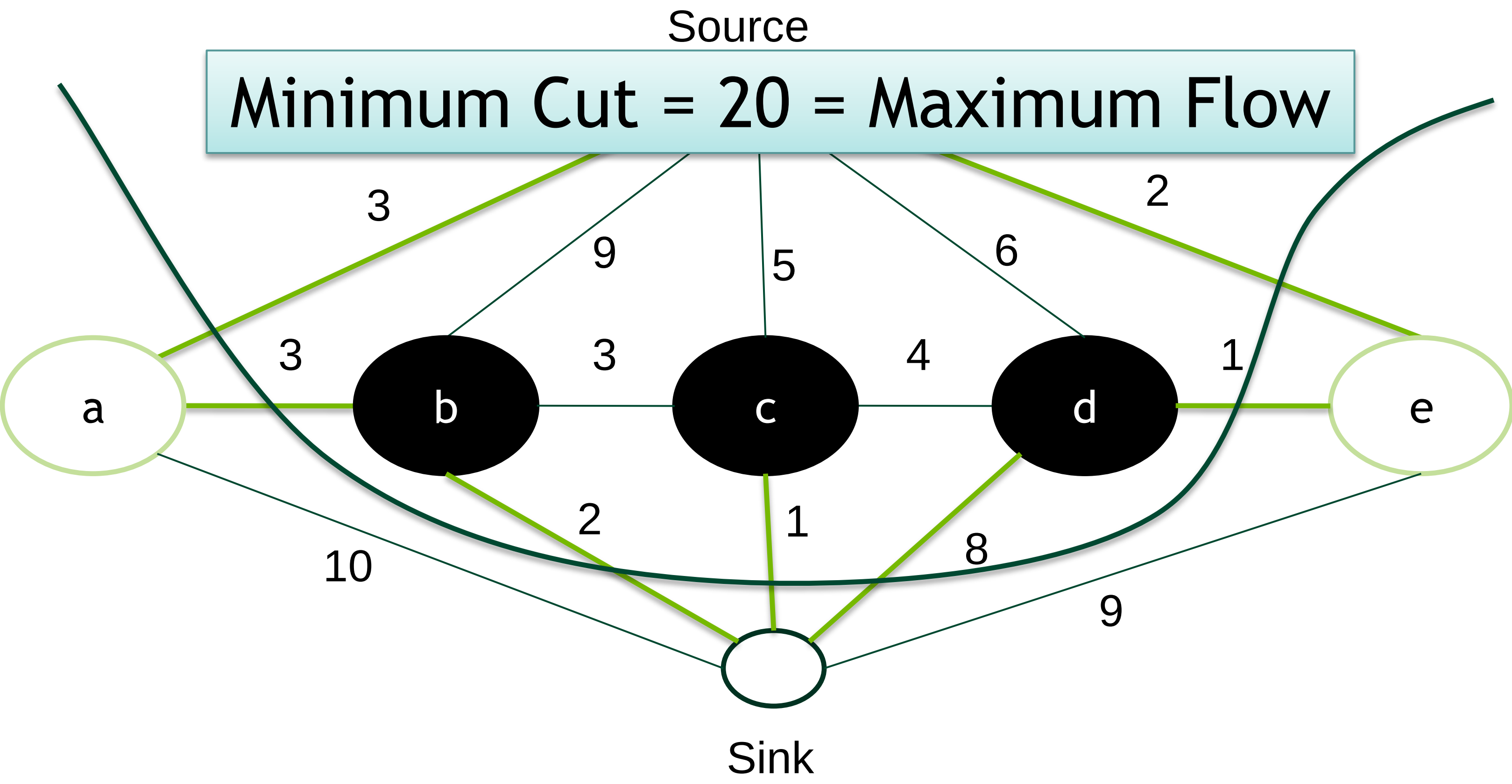
X

-6/0

Sink

Graph Cut and Solution

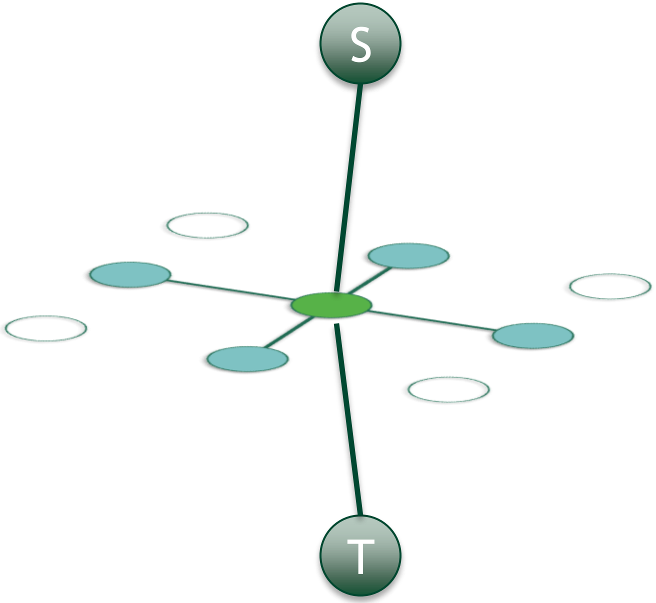
Total flow = 20



Minimum Cut = 20 = Maximum Flow

Graph Cuts for Image Processing

- Regular Graphs with 4-Neighborhood
- Integers
- Naive approach
  - One thread per node
  - Push Kernel
  - Relabel Kernel

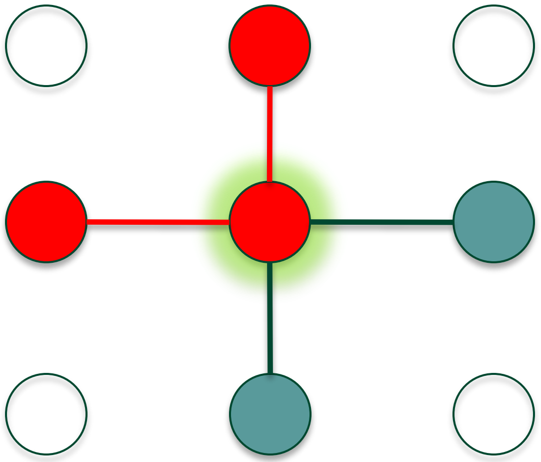


CUDA Implementation

- Datastructures
  - 4 WxH arrays for residual edge capacities
  - 2 WxH array for heights (double buffering)
  - WxH array for excess flow

Push Data Access Patterns

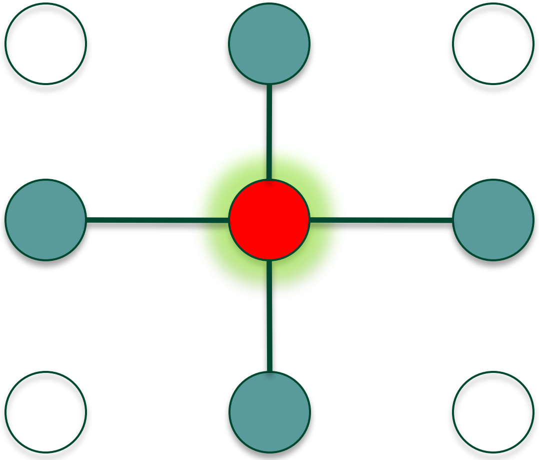
- Read/Write: Excess Flow, Edge capacities
- Read only : Height



Excess Flow Data

Relabel Data Access Patterns

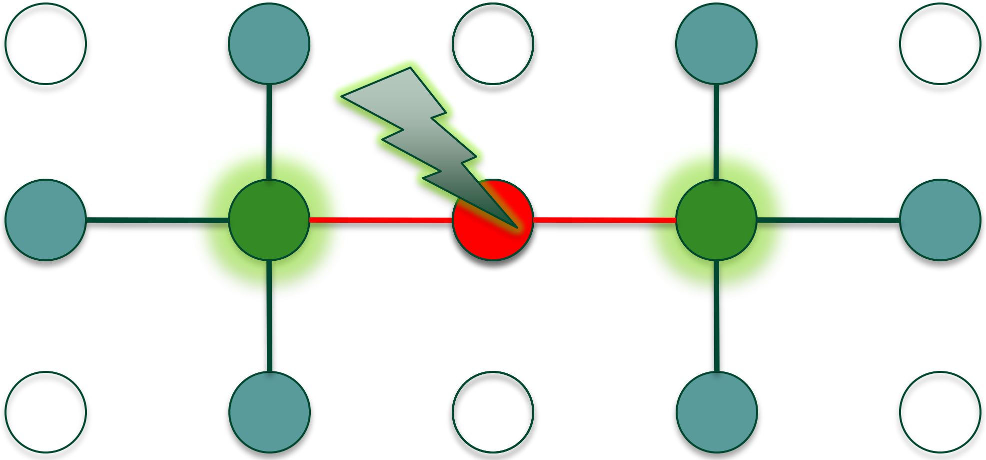
- Read/Write: Height (Texture, double buffered)
- Read only : Excess Flow, Edge capacities



Height Data

Data Access Patterns

- Push does scattered write:



Needs global atomics to avoid RAW Hazard!

Naive CUDA Implementation

- Iterative approach:
- Repeat
  - Push Kernel (Updates excess flow & edge capacities)
  - Relabel Kernel (Updates height)
- Until no active pixels are left

GPU TECHNOLOGY CONFERENCE

Naive CUDA Implementation

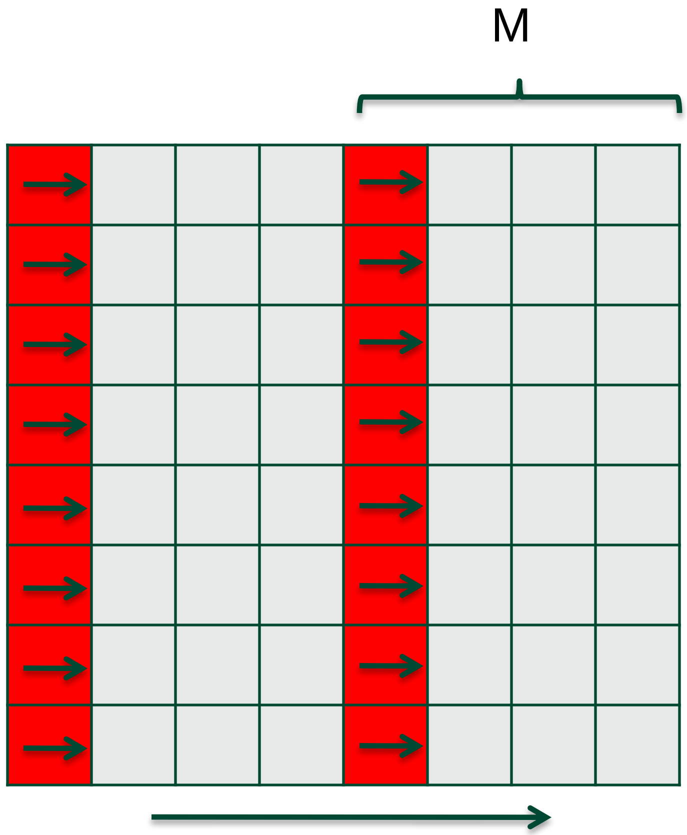
- Both kernels are memory-bound
- Observations on the naive implementation
  - Push: Atomic memory bandwidth is lower
  - Relabel: 1-bit per edge would be sufficient

Addressing these bottlenecks improves overall performance

Push, improved

- Idea:
  - Work on tiles in shared memory
    - Share data between threads of a block
  - Each thread updates M pixels
    - Push first M times in first edge direction
    - Then M times in next edge direction

Wave Push



🟥 Active Thread

→ Push direction

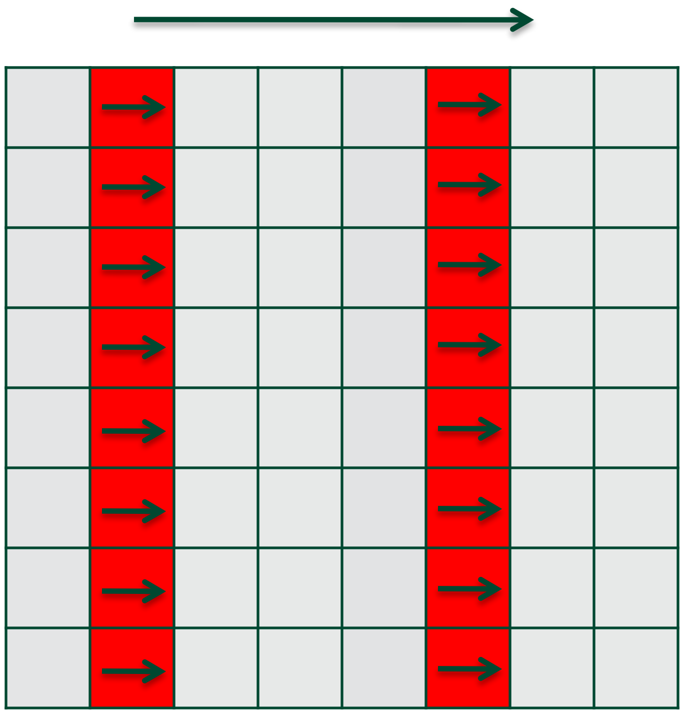
```
ef = 0;
for k=0...M-1
        ef += s_ef(k)
        flow = min(right(x+k),ef)
        right(x+k)-=flow;
        s_ef(k)=ef-flow;
        ef = flow;
end
```

Excess Flow Data-Tile in Shared Memory

Wave Push

🟥 Active Thread

→ Push direction



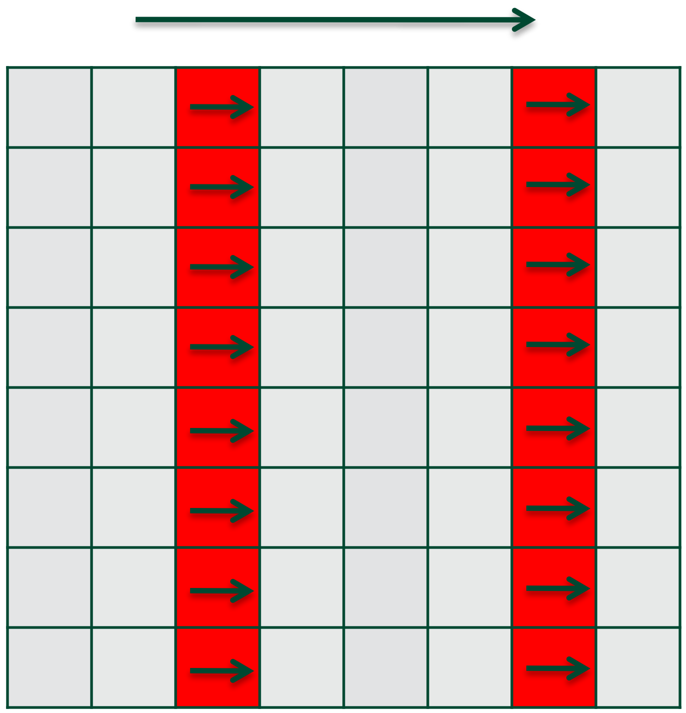
```
ef = 0;
for k=0...M-1
    ef += s_ef(k)
    flow = min(right(x+k),ef)
    right(x+k)-=flow;
    s_ef(k)=ef-flow;
    ef = flow;
end
```

Flow is carried along by each thread

Wave Push

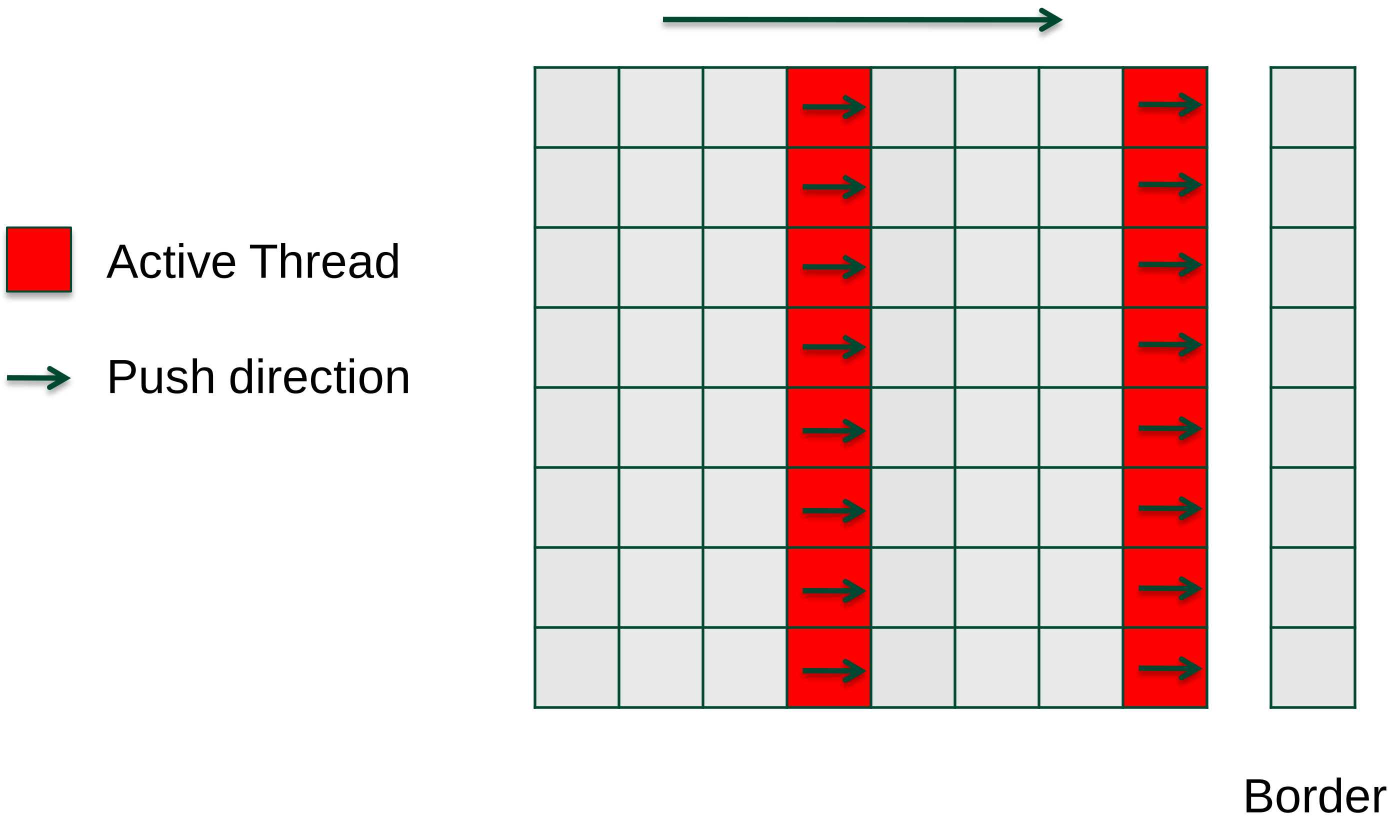
🟥 Active Thread

→ Push direction



```
ef = 0;
for k=0...M-1
    ef += s_ef(k)
    flow = min(right(x+k),ef)
    right(x+k)-=flow;
    s_ef(k)=ef-flow;
    ef = flow;
end
```

Wave Push



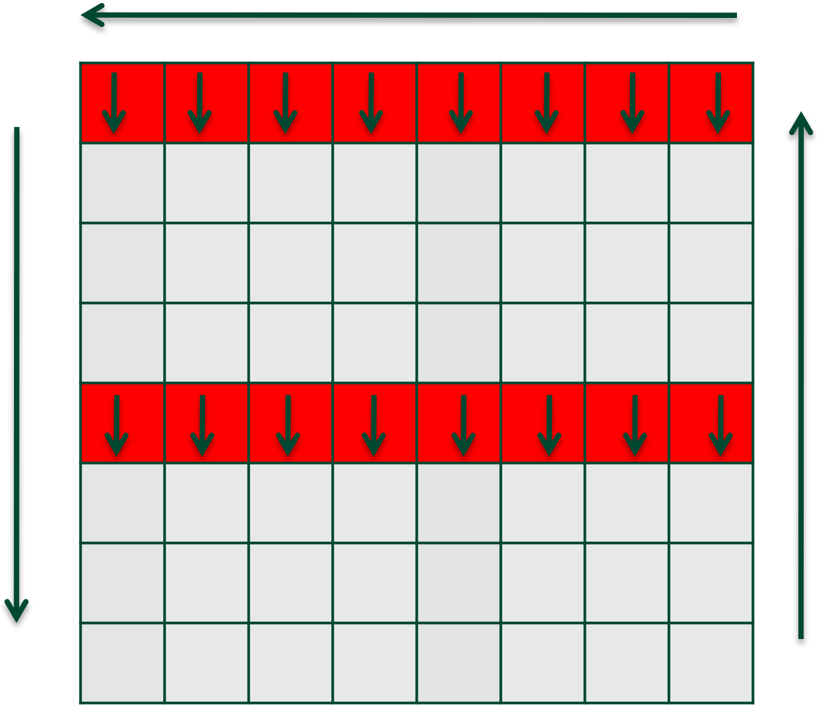
🟥 Active Thread

→ Push direction

Wave Push

🟥 Active Thread

→ Push direction



Do the same for other directions

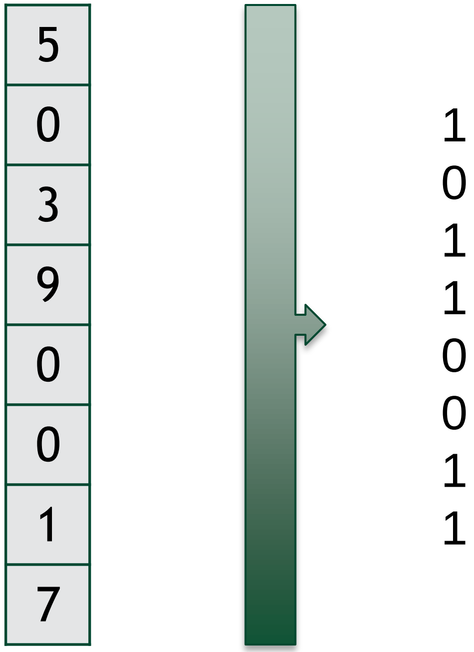
Wave Push

- After tile pushing, border is added
- Benefits
  - No atomics necessary
  - Share data between threads
  - Flow is transported over larger distances

Relabel

- Binary decision: capacity > 0 ? 1 : 0
- Idea: Compress residual edges as bit-vectors
  - Compression computed during push

Relabel



5
0
3
9
0
0
1
7

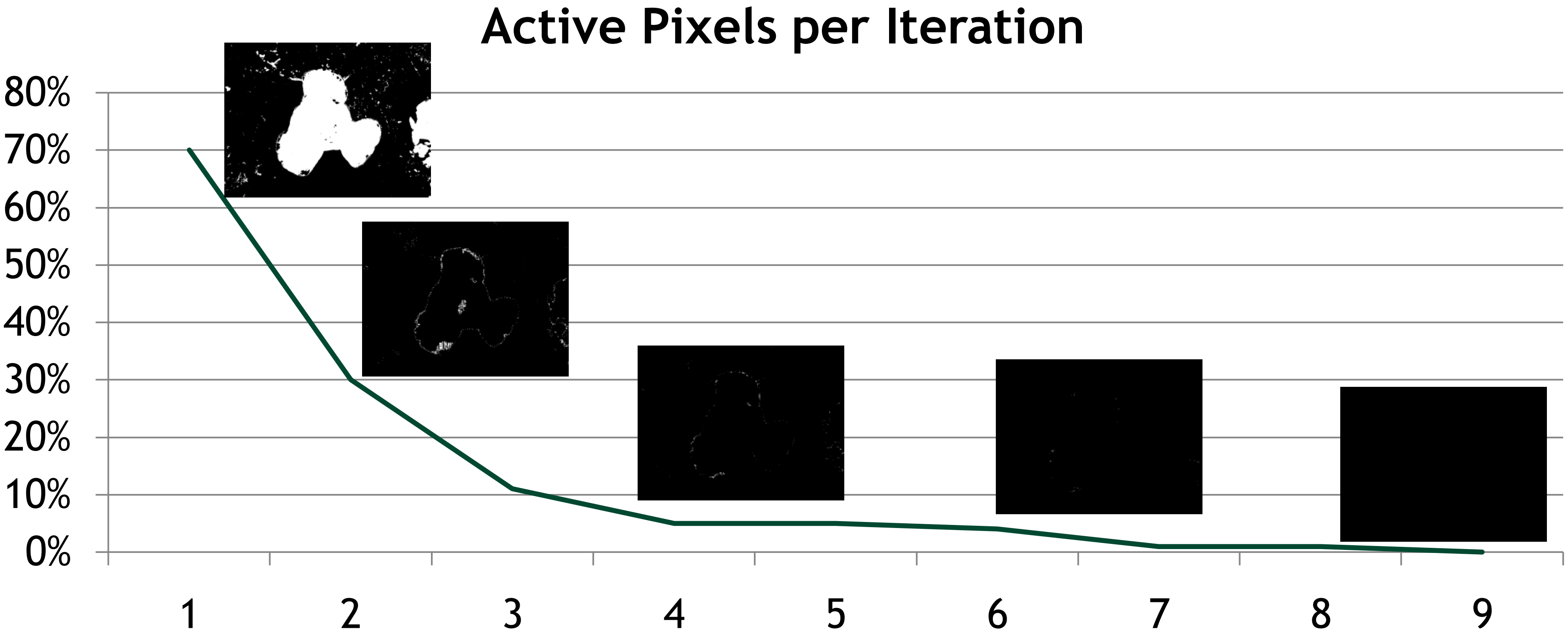
1
0
1
1
0
0
1
1

- Compression Ratio: 1:32 (int capacities)

CUDA Implementation

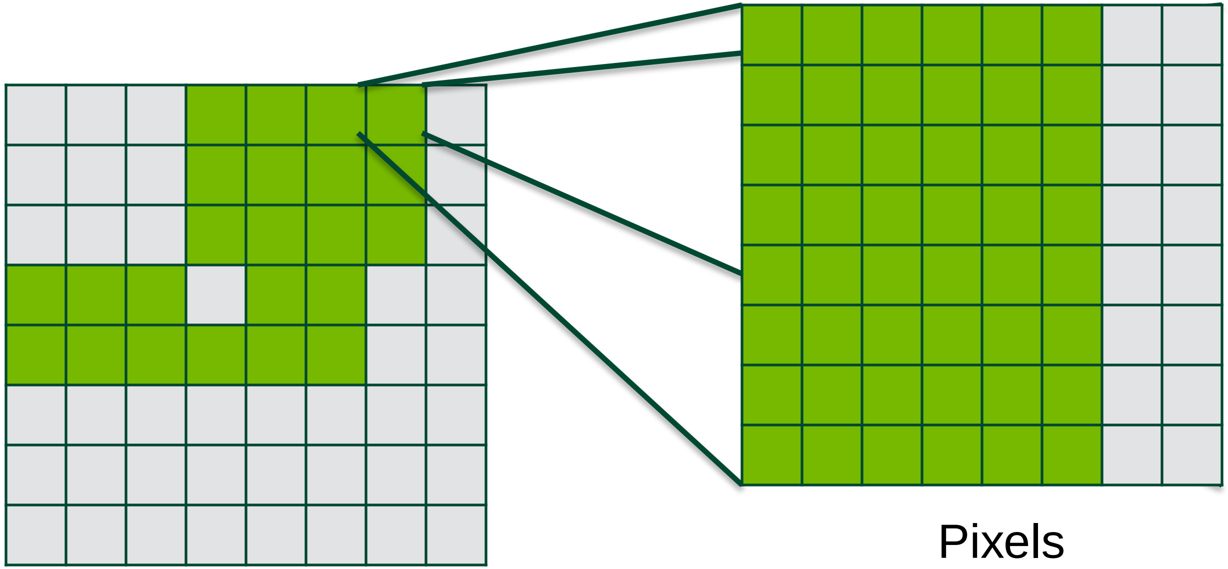
- Algorithmic observations
  - Most parts of the graph will converge early
  - Periodic global relabeling significantly reduces necessary iterations

Tile based push-relabel



Tile based push-relabel

- Split graph in NxN pixel tiles (32x32)
- If any pixel is active, the tile is active



Tile based push-relabel

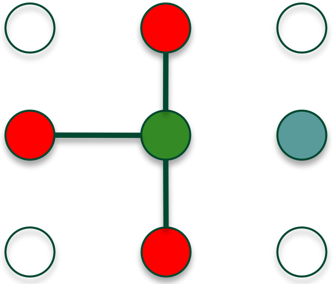
- Repeat
  - Build list of active tiles
  - For each active tile
    - Push
    - Relabel
- Until no active tile left

Global Relabel

- Local relabel is a bad heuristic for long distance flow transportation
  - Unnecessary pushing of flow back and forth
- Global relabel is exact
  - Computes the correct geodesic distances
  - Flow will be pushed in the correct direction
  - Downside: costly operation

Global Relabel

- BFS from sink
  - First step implicit -> multi-sink BFS
- Implemented as local operator:

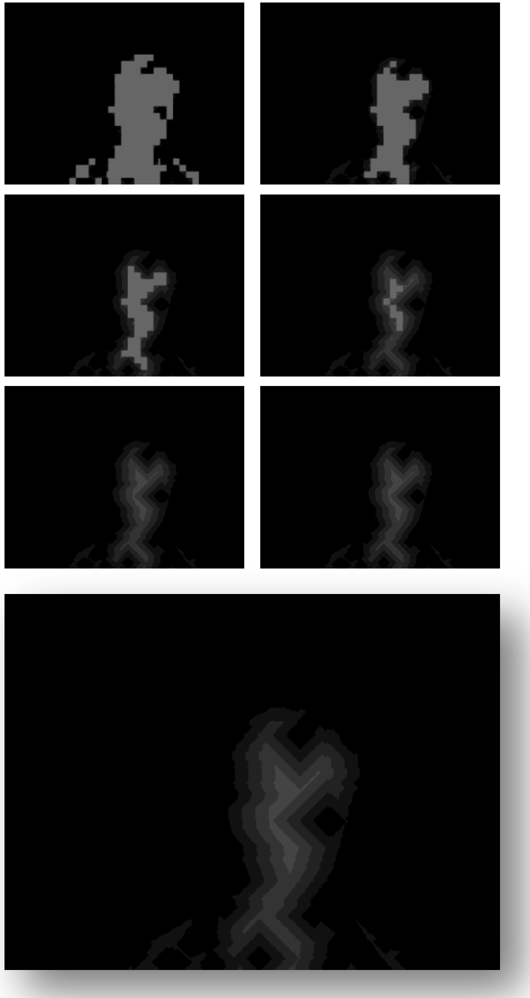


Global Relabel

- Mechanisms from Push-Relabel can be reused:
  - Wave Updates
  - Residual Graph Compression
  - Tile based

Global Relabel

- Initialize all pixels:
  - with flow < 0 to 0 (multi-sink BFS)
  - with flow >= 0 to infinity
- Compress residual graph
- Build active tile list
- Repeat
  - Wave label update
- Until no label changed



Final CUDA Graphcut

- Repeat
  - Global Relabel
  - For H times do
    - Build active tile list
    - For each tile do push-relabel
- Until no active tile

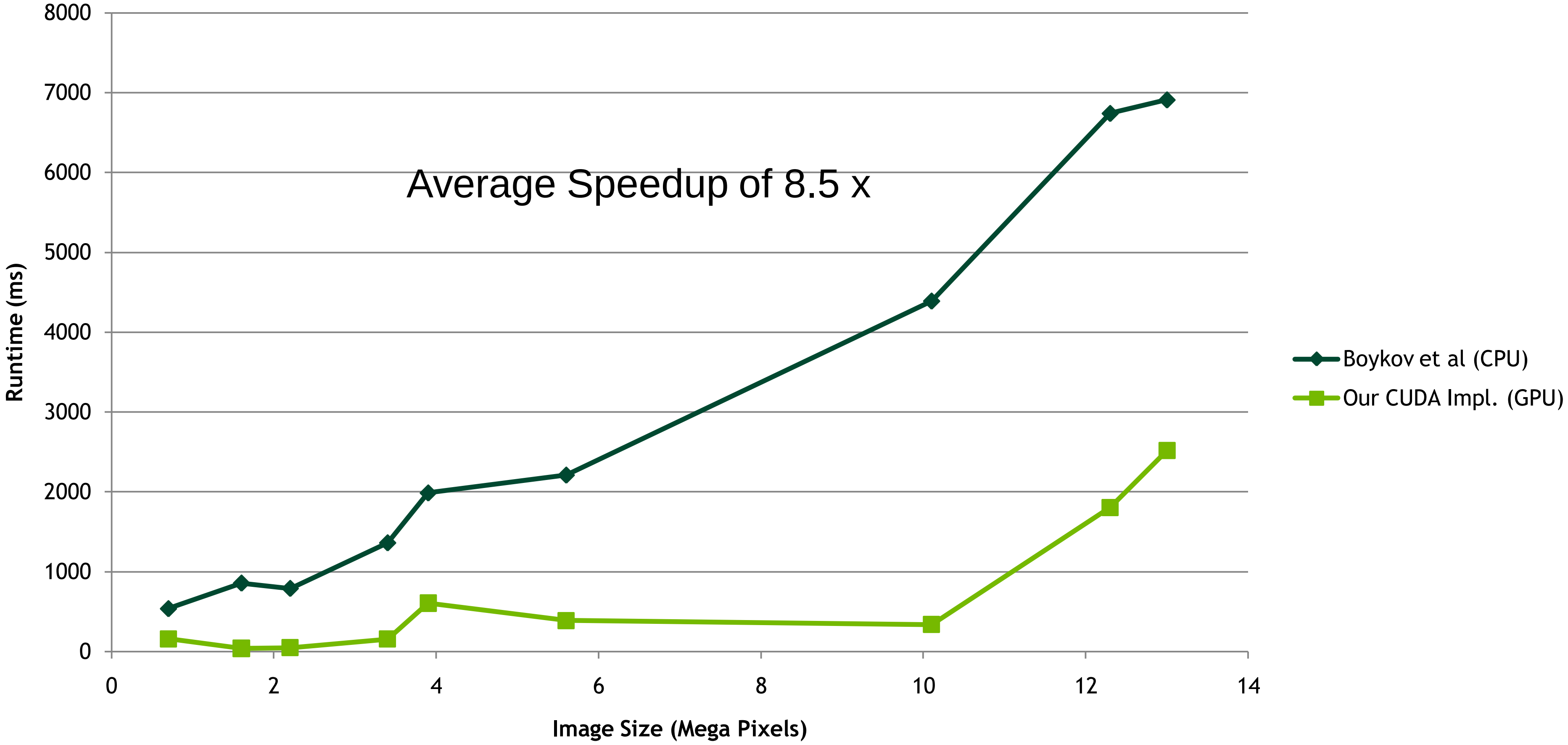
Results

- Comparison between Boykov et al. (CPU), CudaCuts and our implementation
  - Intel Core2 Duo E6850 @ 3.00 GHz
  - NVIDIA Tesla C1060

Dataset	Boykov (CPU)	CudaCuts (GPU)	Our (GPU)	Speedup Our vs CPU
Flower (600x450)	191 ms	92 ms	20 ms	**9.5x**
Sponge (640x480)	268 ms	59 ms	14 ms	**19x**
Person (600x450)	210 ms	78 ms	35 ms	**6x**

Average speedup over CPU is 10x

Results



Average Speedup of 8.5 x

Example Application: GrabCut

GrabCut Application (Siggraph 2004 paper)

- Based on Color models for FG and BG
  - User specifies a rectangle around the object to cut
  - Initialize GMM model of FG and BG colors
  - Graph Cut to find labeling
  - Use new labeling to update GMM
  - Iterate until convergence
- Full CUDA implementation
- Total runtime: ~25 ms per iteration -> 500 ms

Summary

- Introduction to Graph Cuts
- Push-Relabel CUDA implementation
  - Beats CPU by 8.5 x on average
- Makes full CUDA implementation of many image processing applications possible