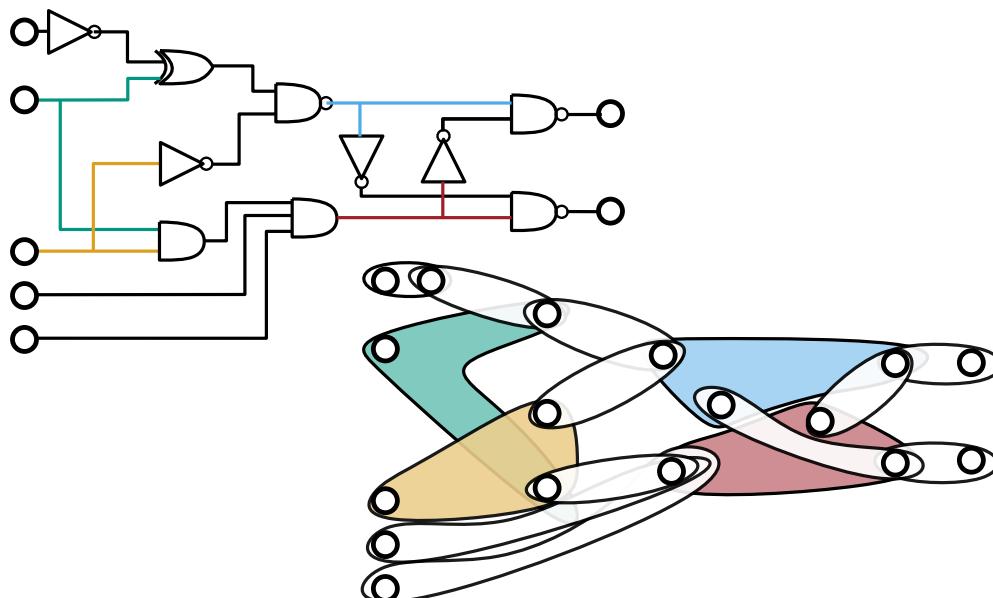


# High Quality Hypergraph Partitioning via Max-Flow-Min-Cut Computations

Master Thesis · February 16, 2018  
Tobias Heuer

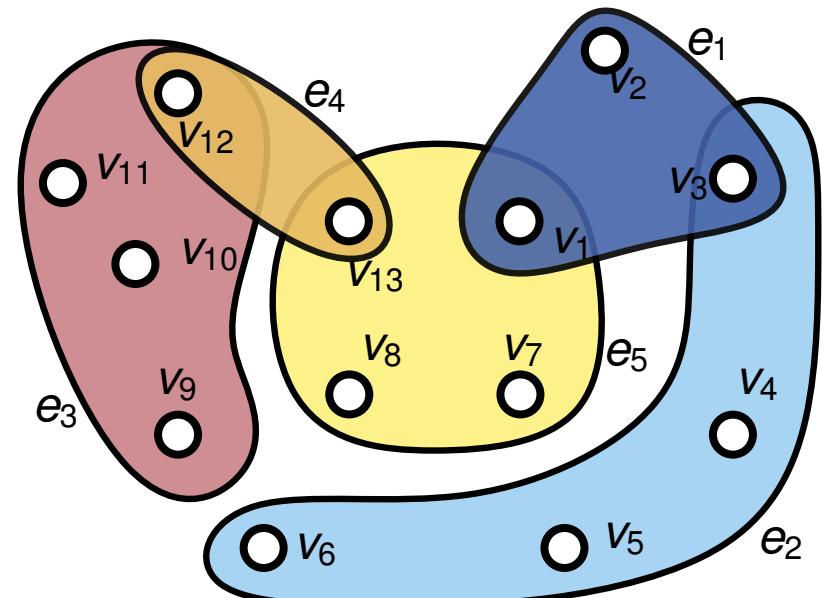
INSTITUTE OF THEORETICAL INFORMATICS · ALGORITHMIC GROUP



# Hypergraphs

[from SEA'17]

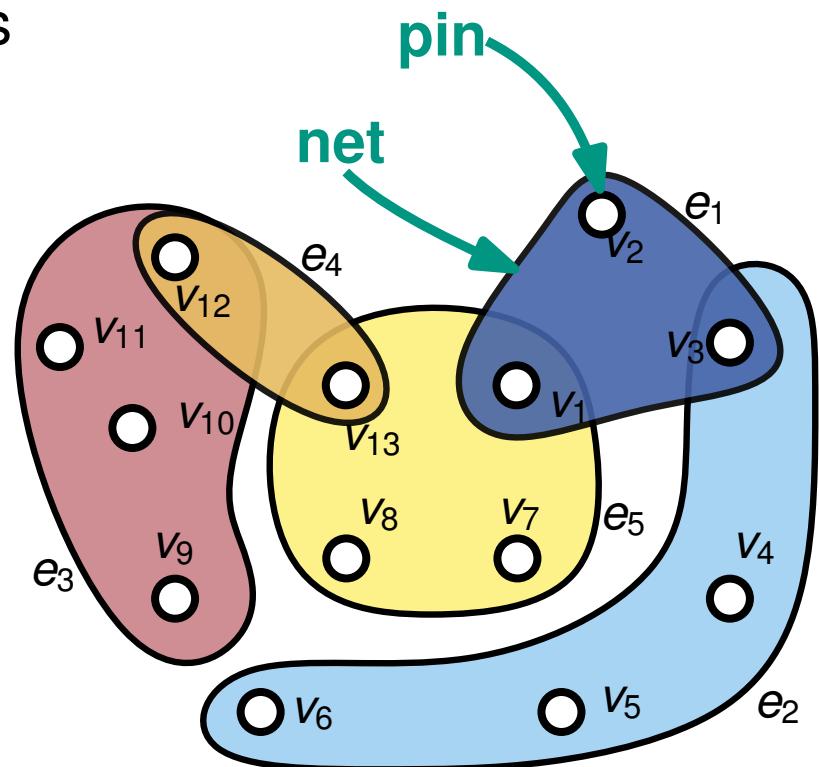
- Generalization of graphs  
⇒ hyperedges connect  $\geq 2$  nodes
- Graphs ⇒ dyadic (**2-ary**) relationships
- Hypergraphs ⇒ (**d-ary**) relationships
- Hypergraph  $H = (V, E, c, \omega)$ 
  - Vertex set  $V = \{1, \dots, n\}$
  - Edge set  $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - Node weights  $c : V \rightarrow \mathbb{R}_{\geq 1}$
  - Edge weights  $\omega : E \rightarrow \mathbb{R}_{\geq 1}$



# Hypergraphs

[from SEA'17]

- Generalization of graphs  
⇒ hyperedges connect  $\geq 2$  nodes
- Graphs ⇒ dyadic (**2-ary**) relationships
- Hypergraphs ⇒ (**d-ary**) relationships
- Hypergraph  $H = (V, E, c, \omega)$ 
  - Vertex set  $V = \{1, \dots, n\}$
  - Edge set  $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - Node weights  $c : V \rightarrow \mathbb{R}_{\geq 1}$
  - Edge weights  $\omega : E \rightarrow \mathbb{R}_{\geq 1}$
- $|P| = \sum_{e \in E} |e| = \sum_{v \in V} d(v)$



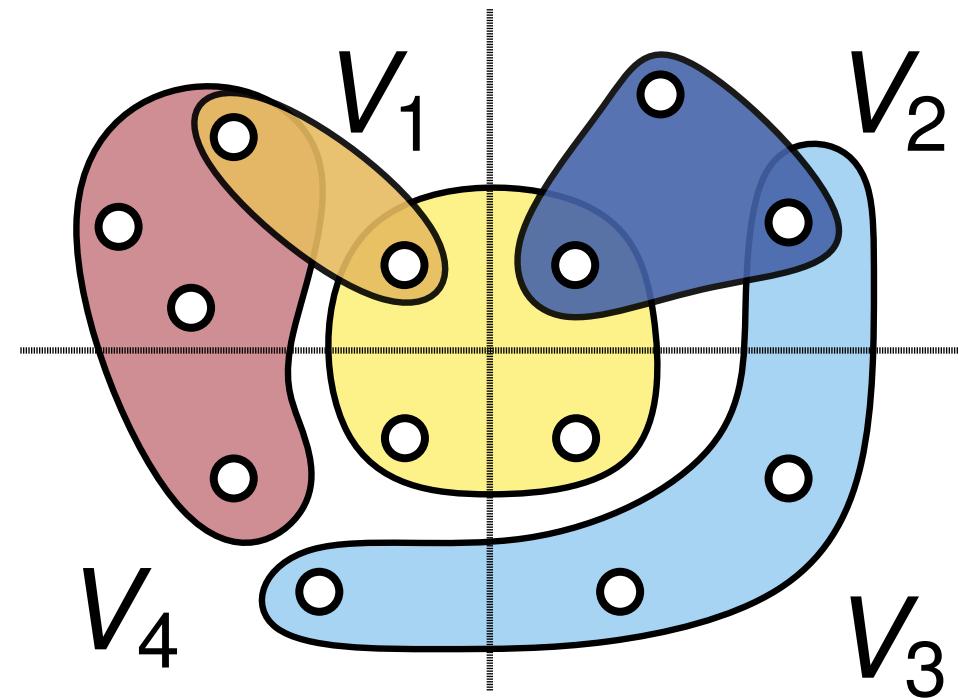
# Hypergraph Partitioning Problem

[from SEA'17]

Partition hypergraph  $H = (V, E, c, \omega)$  into  $k$  non-empty disjoint blocks  $\Pi = \{V_1, \dots, V_k\}$  such that:

- blocks  $V_i$  are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$



# Hypergraph Partitioning Problem

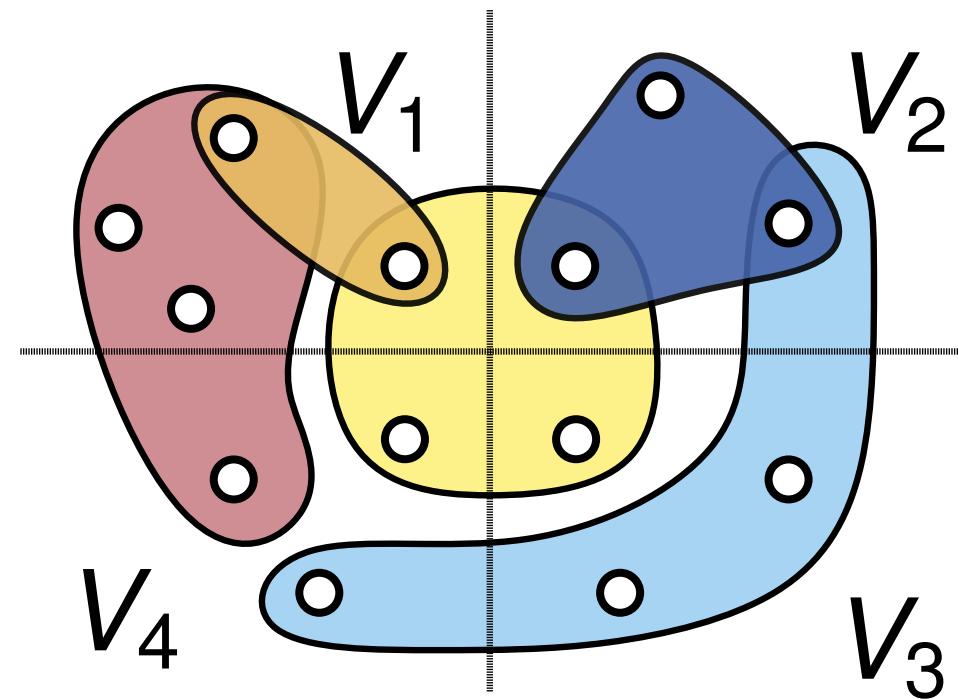
[from SEA'17]

Partition hypergraph  $H = (V, E, c, \omega)$  into  $k$  non-empty disjoint blocks  $\Pi = \{V_1, \dots, V_k\}$  such that:

- blocks  $V_i$  are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance  
parameter



# Hypergraph Partitioning Problem

[from SEA'17]

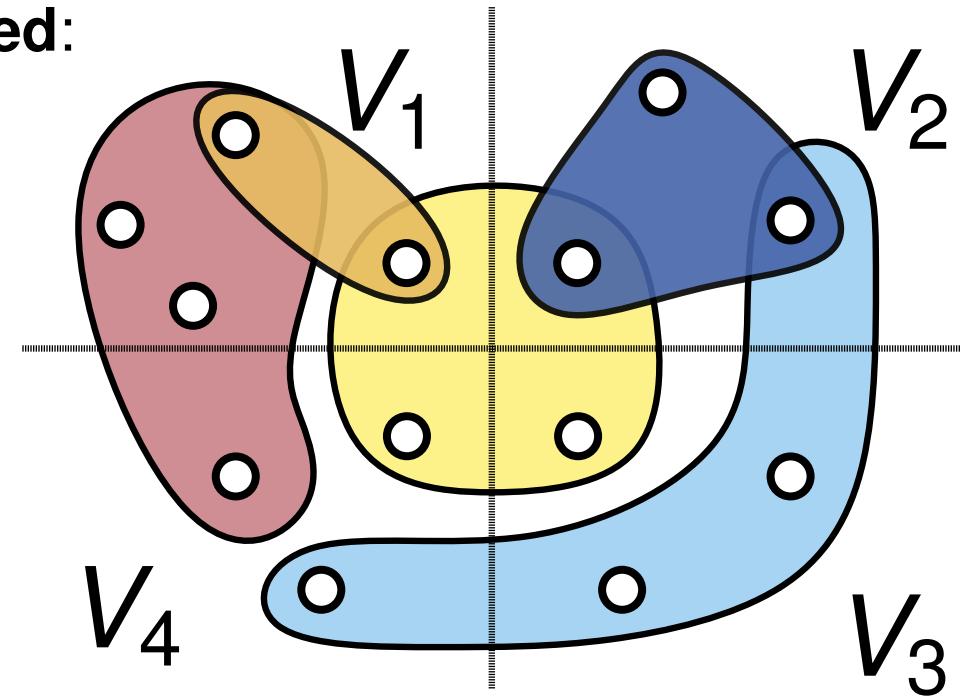
Partition hypergraph  $H = (V, E, c, \omega)$  into  $k$  non-empty disjoint blocks  $\Pi = \{V_1, \dots, V_k\}$  such that:

- blocks  $V_i$  are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance  
parameter

- **connectivity** objective is **minimized**:



# Hypergraph Partitioning Problem

[from SEA'17]

Partition hypergraph  $H = (V, E, c, \omega)$  into  $k$  non-empty disjoint blocks  $\Pi = \{V_1, \dots, V_k\}$  such that:

- blocks  $V_i$  are **roughly equal-sized**:

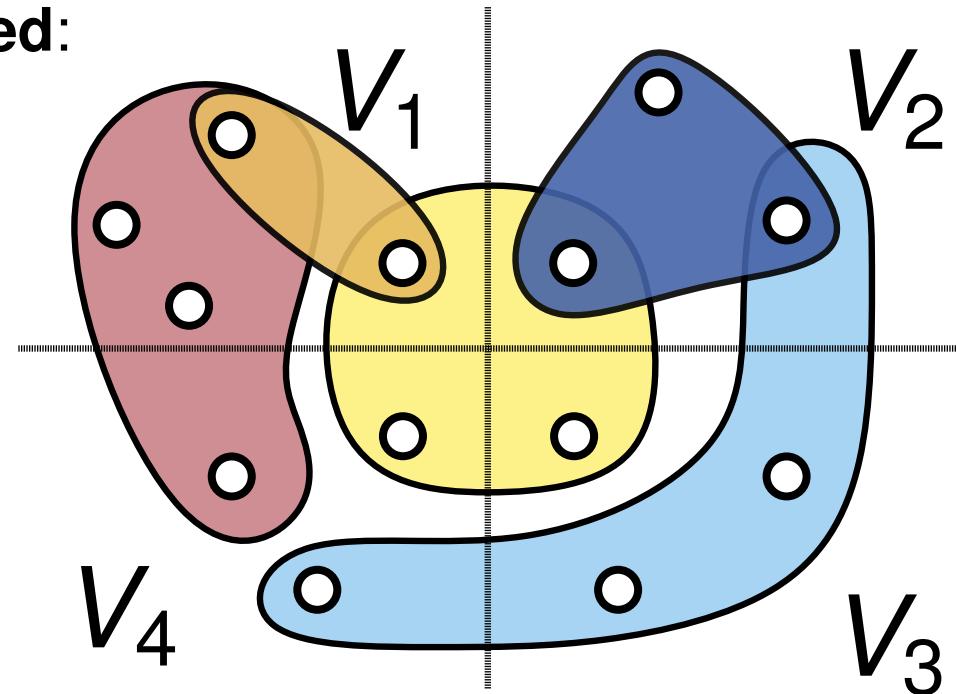
$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance  
parameter

- **connectivity** objective is **minimized**:

$$\sum_{e \in \text{cut}} (\lambda(e) - 1) \omega(e)$$

connectivity:  
# blocks connected by net  $e$



# Hypergraph Partitioning Problem

[from SEA'17]

Partition hypergraph  $H = (V, E, c, \omega)$  into  $k$  non-empty disjoint blocks  $\Pi = \{V_1, \dots, V_k\}$  such that:

- blocks  $V_i$  are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

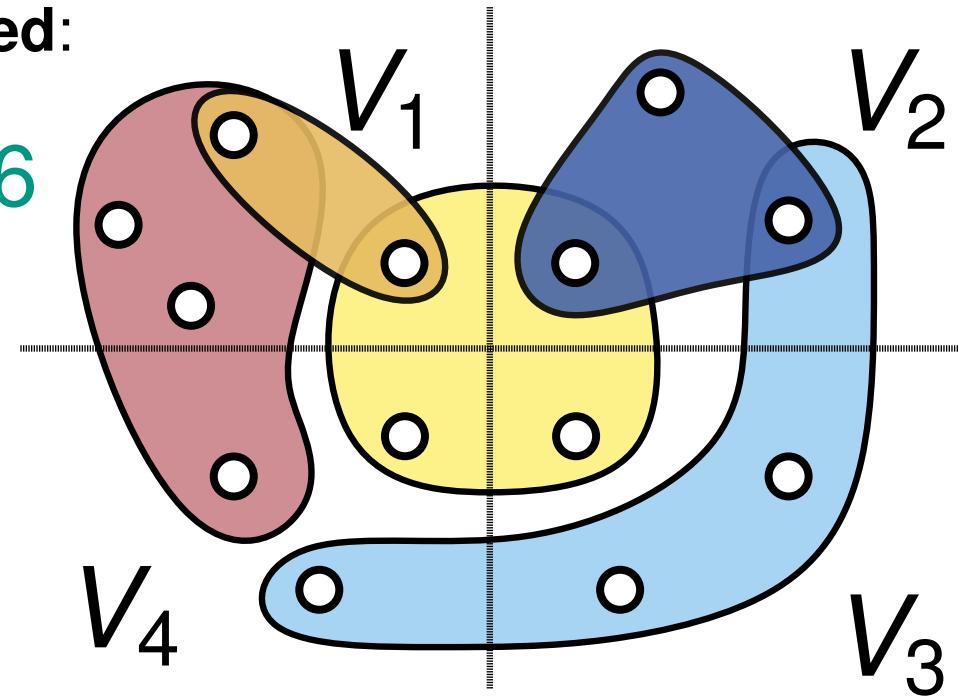
imbalance  
parameter

- **connectivity** objective is **minimized**:

$$\sum_{e \in \text{cut}} (\lambda(e) - 1) \omega(e) = 6$$

connectivity:

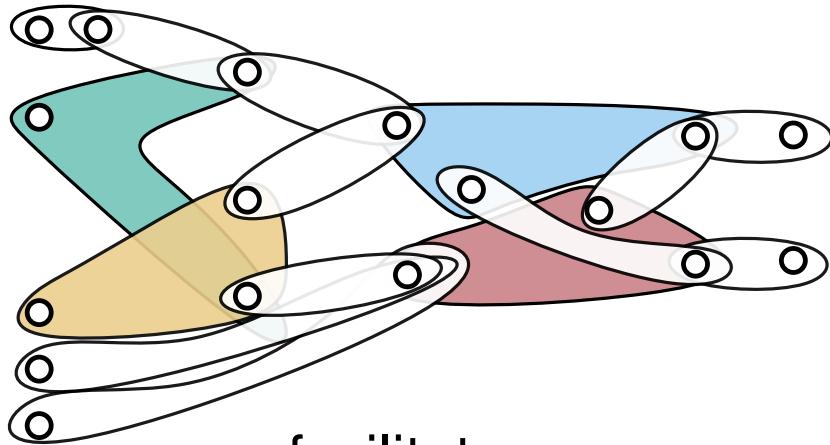
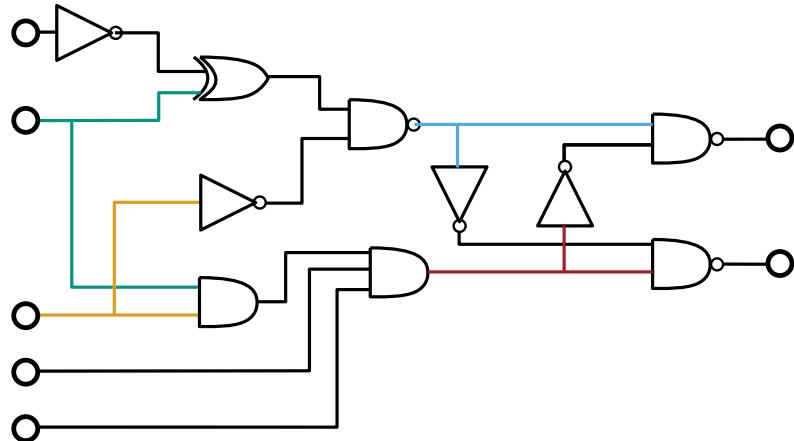
# blocks connected by net  $e$



# Applications

[from SEA'17]

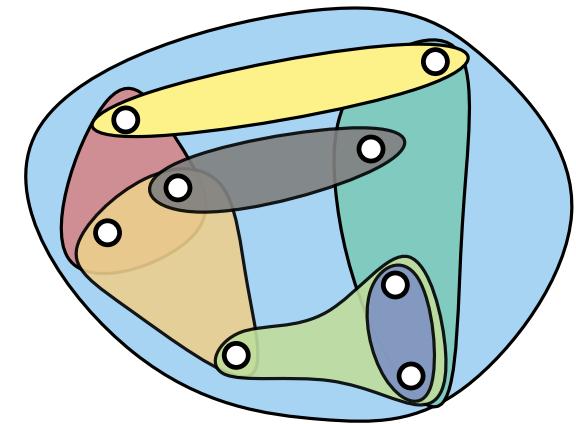
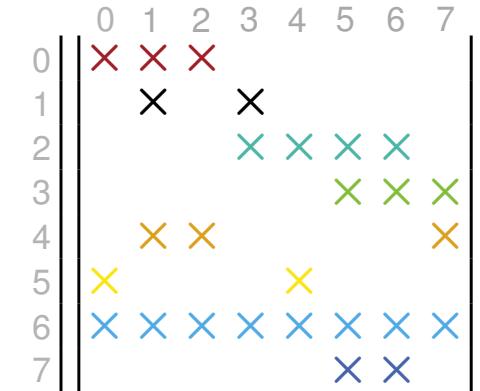
## VLSI Design



facilitate  
floorplanning & placement

Application  
Domain

## Scientific Computing



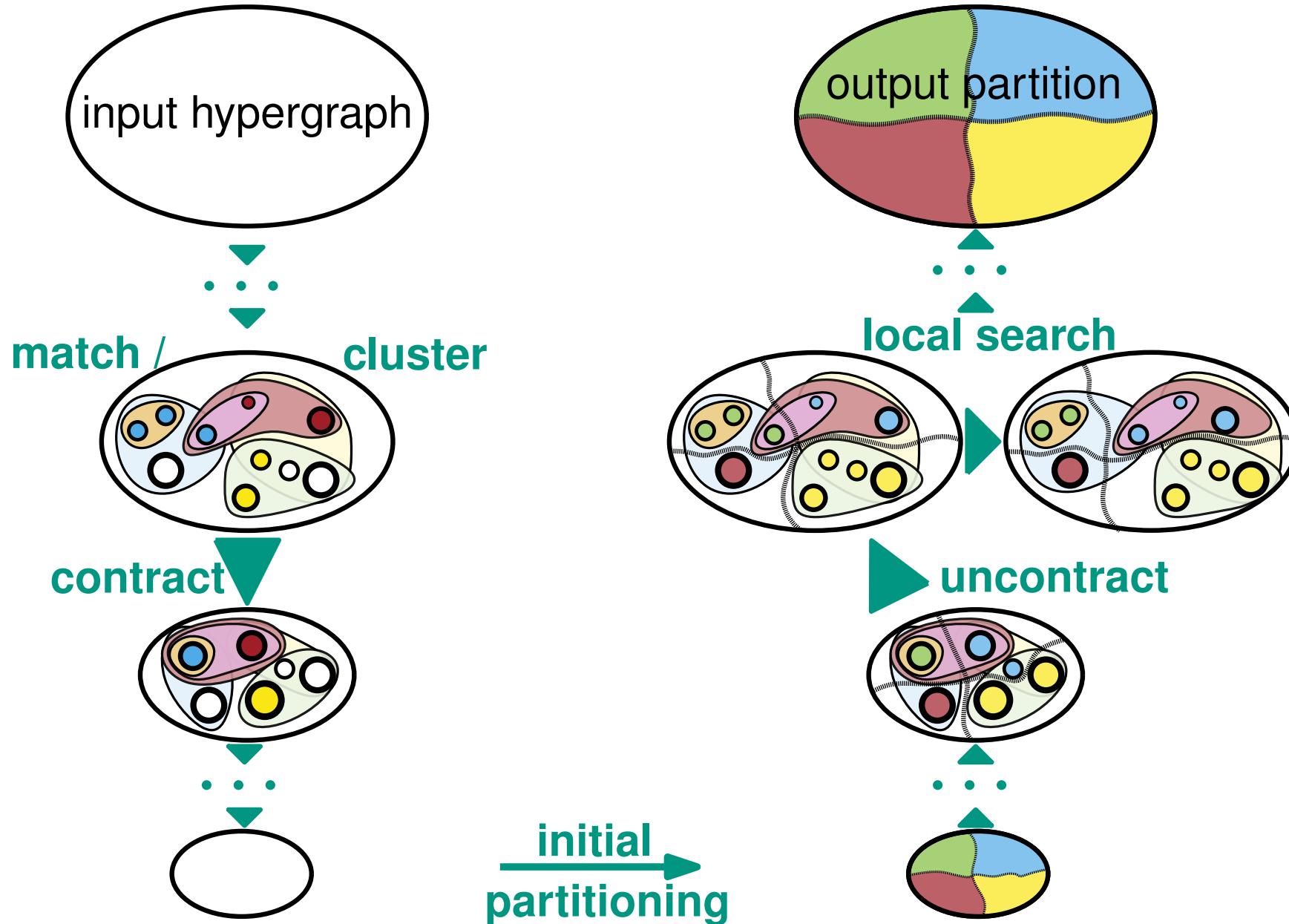
Hypergraph  
Model

Goal

minimize  
communication

# The Multilevel Framework

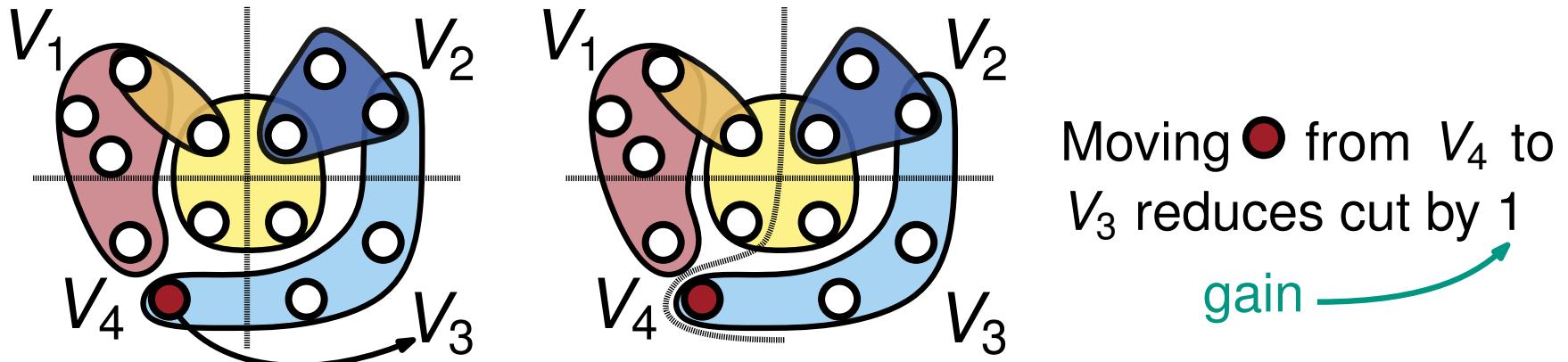
[from SEA'17]



# FM Algorithm

[Fiduccia, Mattheyses 88]

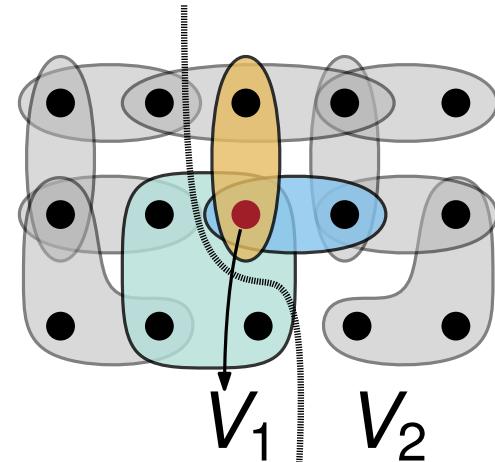
- All modern hypergraph partitioners implements variations of the *FM* algorithm
- **Move-based heuristic** that **greedily** move vertices between blocks based on **local** informations of incident nets



- Performs moves of vertices with **maximum gain** in each step

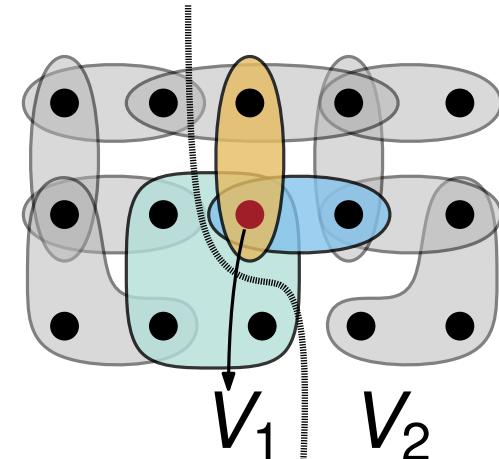
# FM Algorithm - Disadvantages

- Only incorporates **local** informations about the problem structure
  - Heavily depends on *initial partition*
  - In multilevel context: Depends on quality of *coarsening*

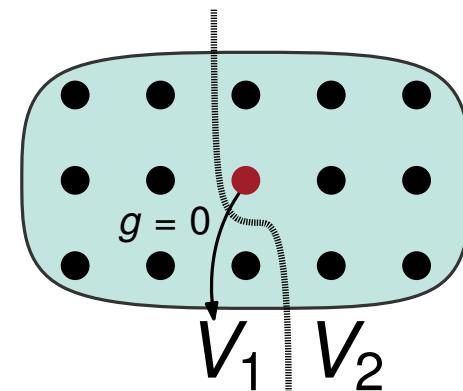


# FM Algorithm - Disadvantages

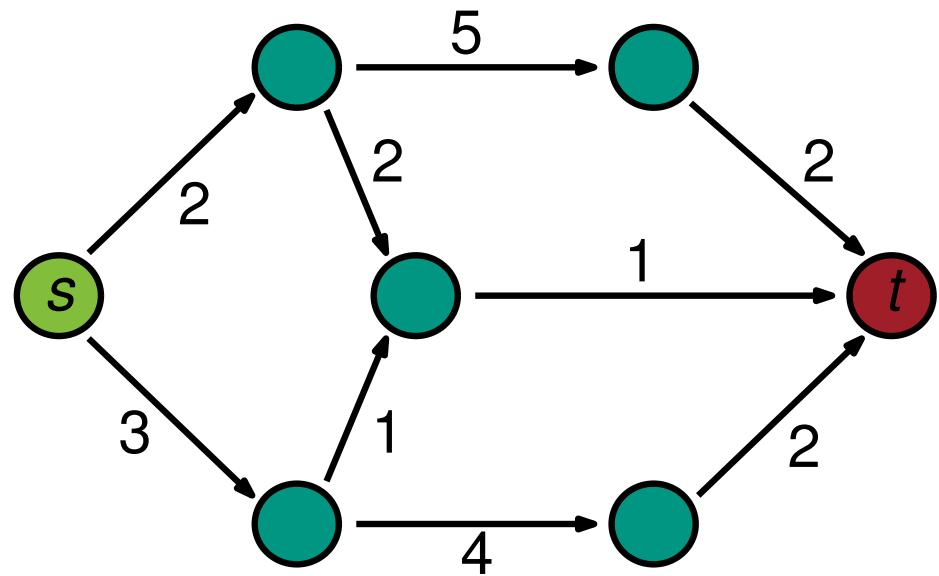
- Only incorporates **local** informations about the problem structure
  - Heavily depends on *initial partition*
  - In multilevel context: Depends on quality of *coarsening*



- Large hyperedges induce **Zero-Gain** moves
  - Quality mainly depends on random decisions made within the algorithm

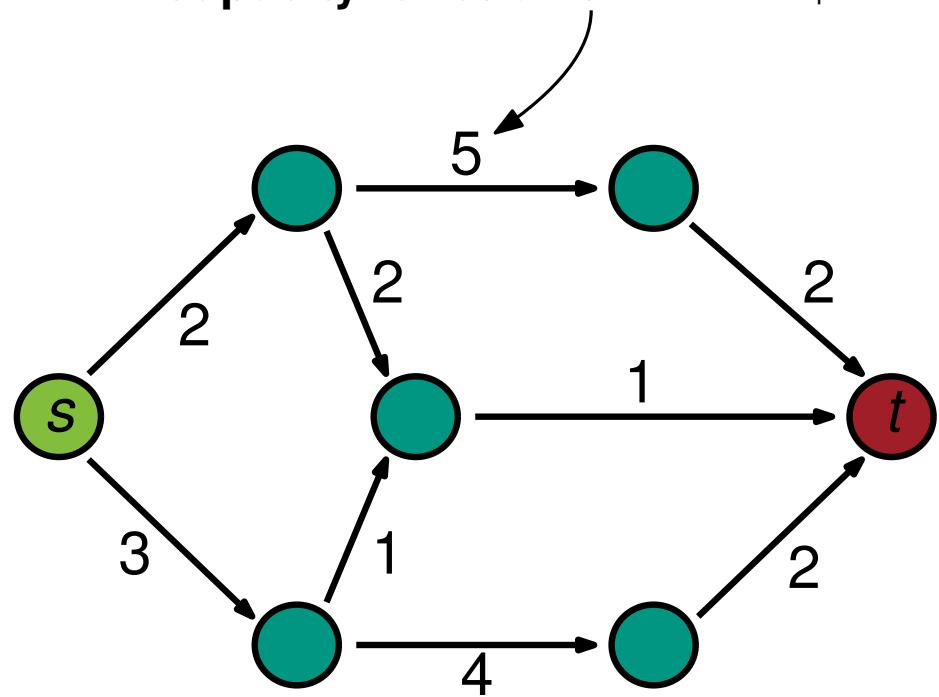


# Flows



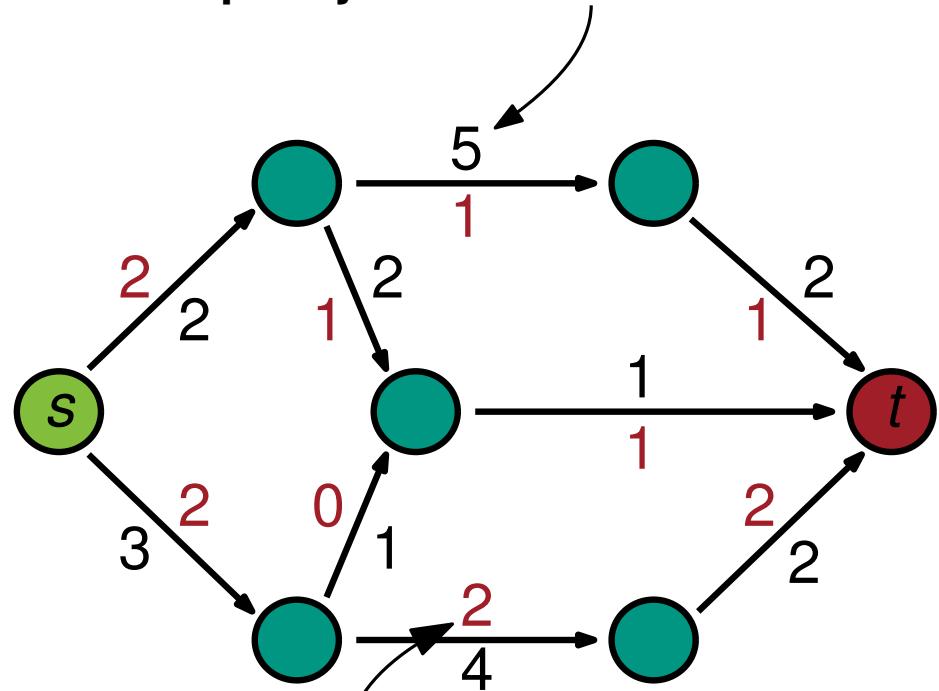
# Flows

Capacity function  $u : E \rightarrow \mathbb{R}_+$



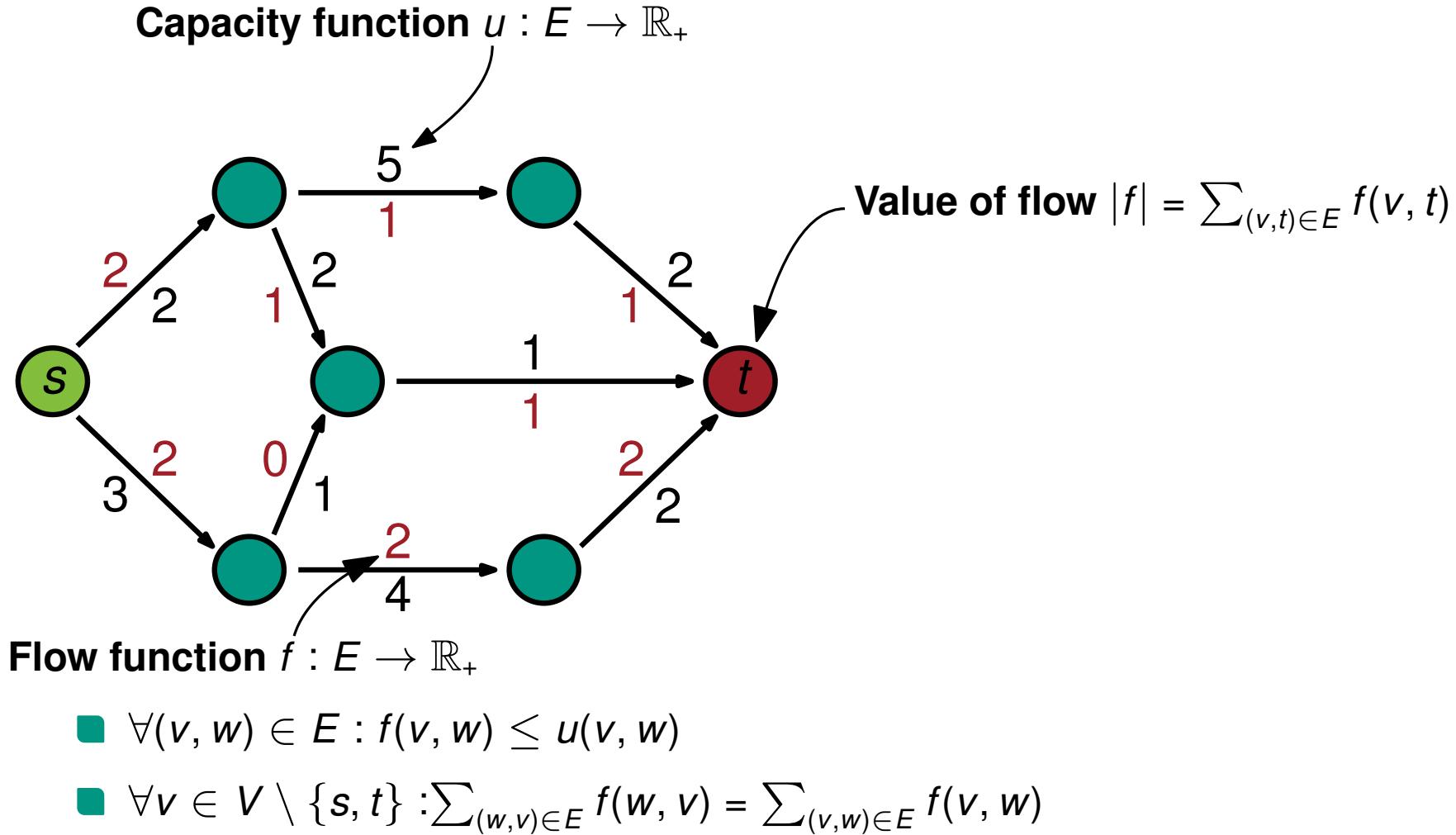
# Flows

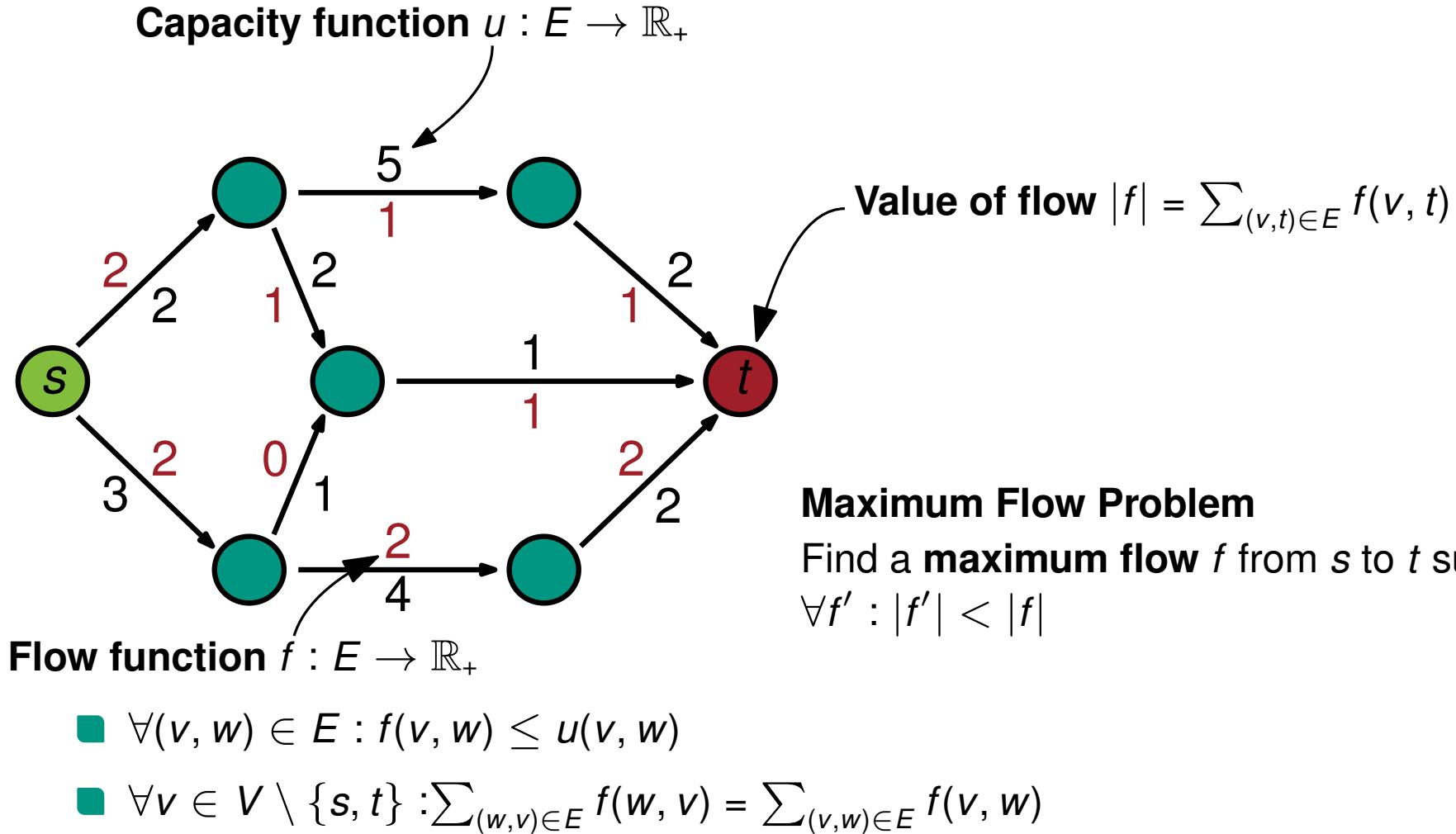
**Capacity function**  $u : E \rightarrow \mathbb{R}_+$



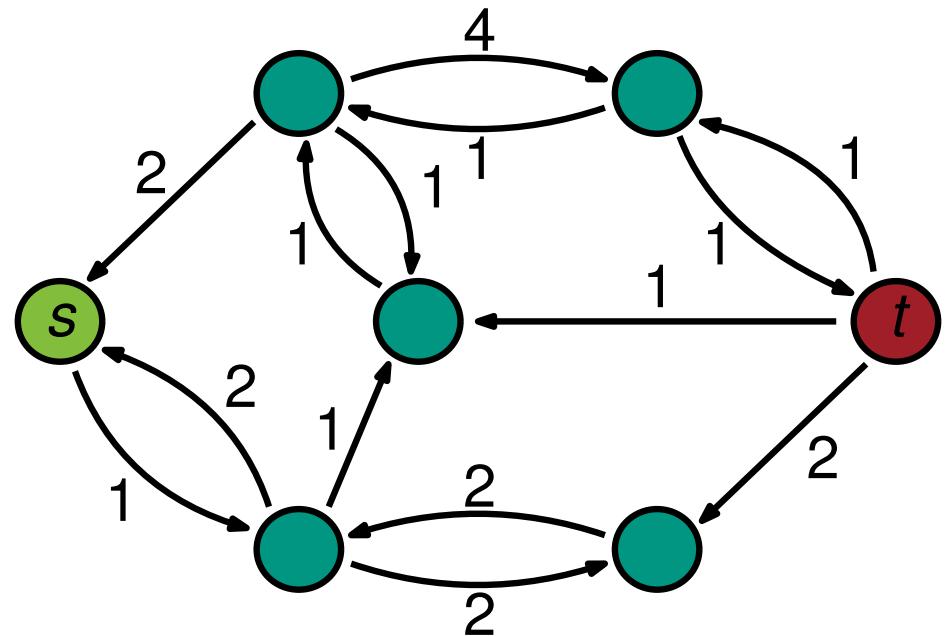
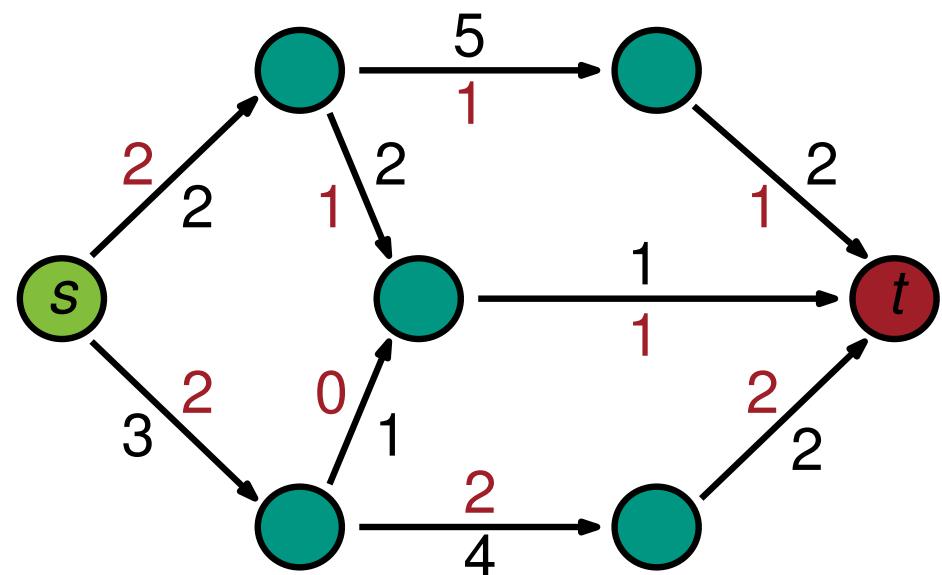
**Flow function**  $f : E \rightarrow \mathbb{R}_+$

- $\forall(v, w) \in E : f(v, w) \leq u(v, w)$
- $\forall v \in V \setminus \{s, t\} : \sum_{(w, v) \in E} f(w, v) = \sum_{(v, w) \in E} f(v, w)$



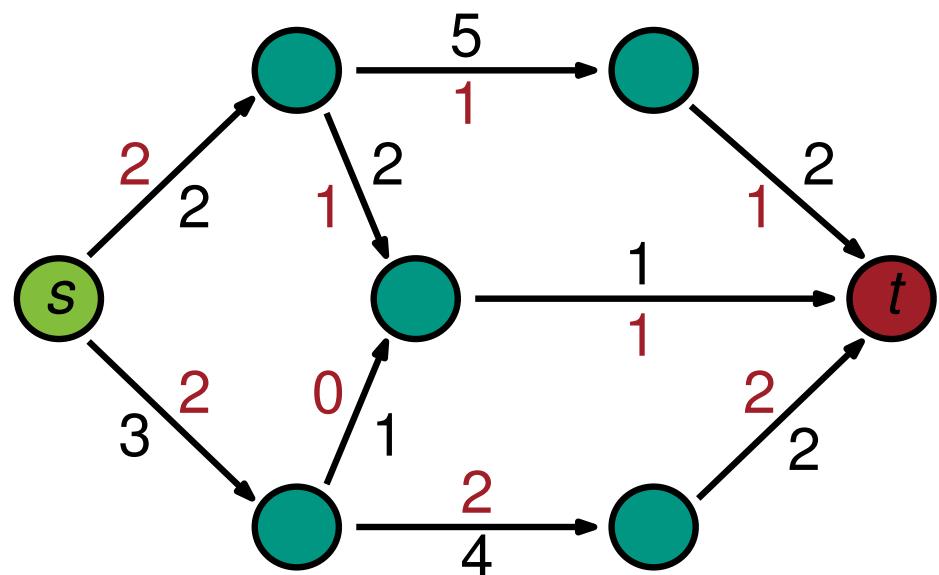


# Flows

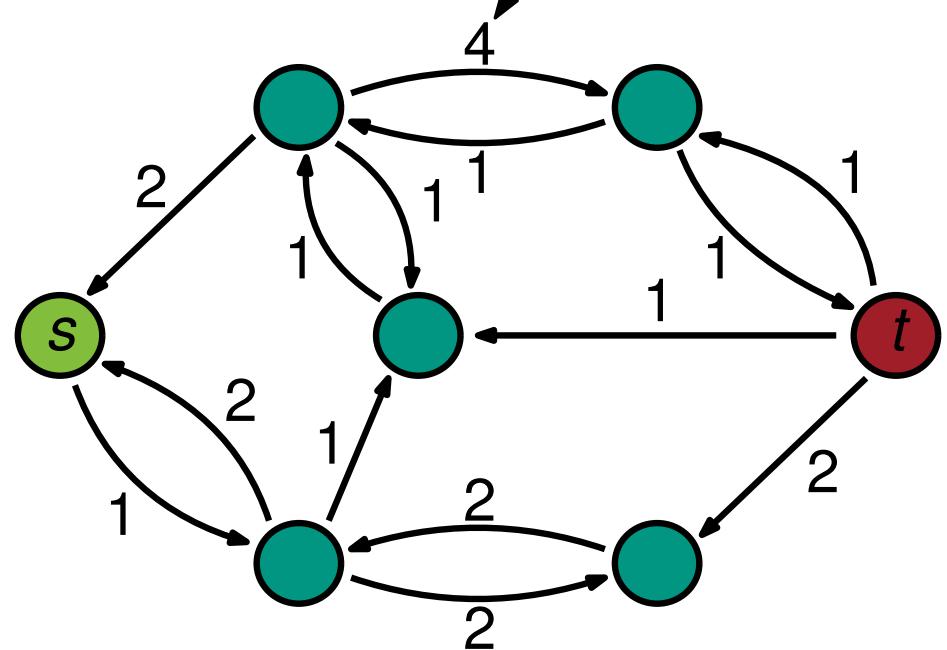


Residual Graph  $G_f$

# Flows

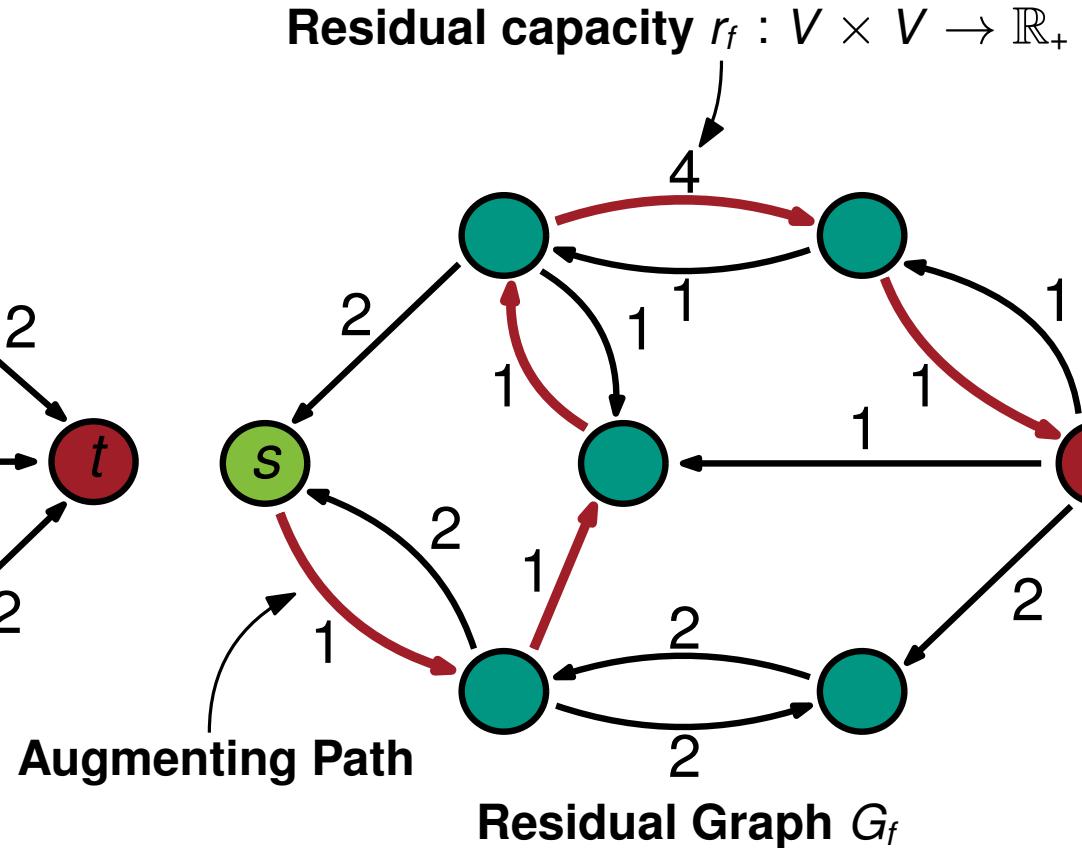
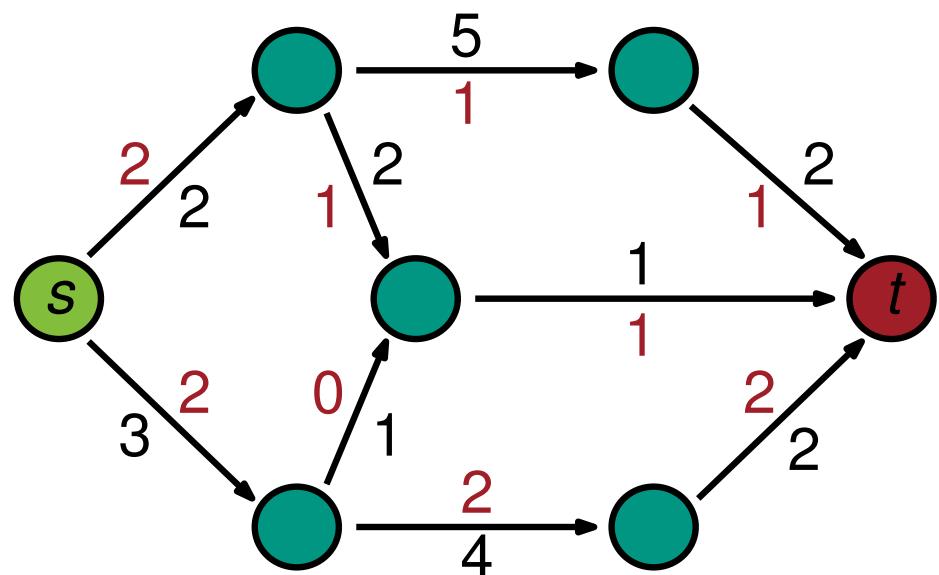


**Residual capacity**  $r_f : V \times V \rightarrow \mathbb{R}_+$



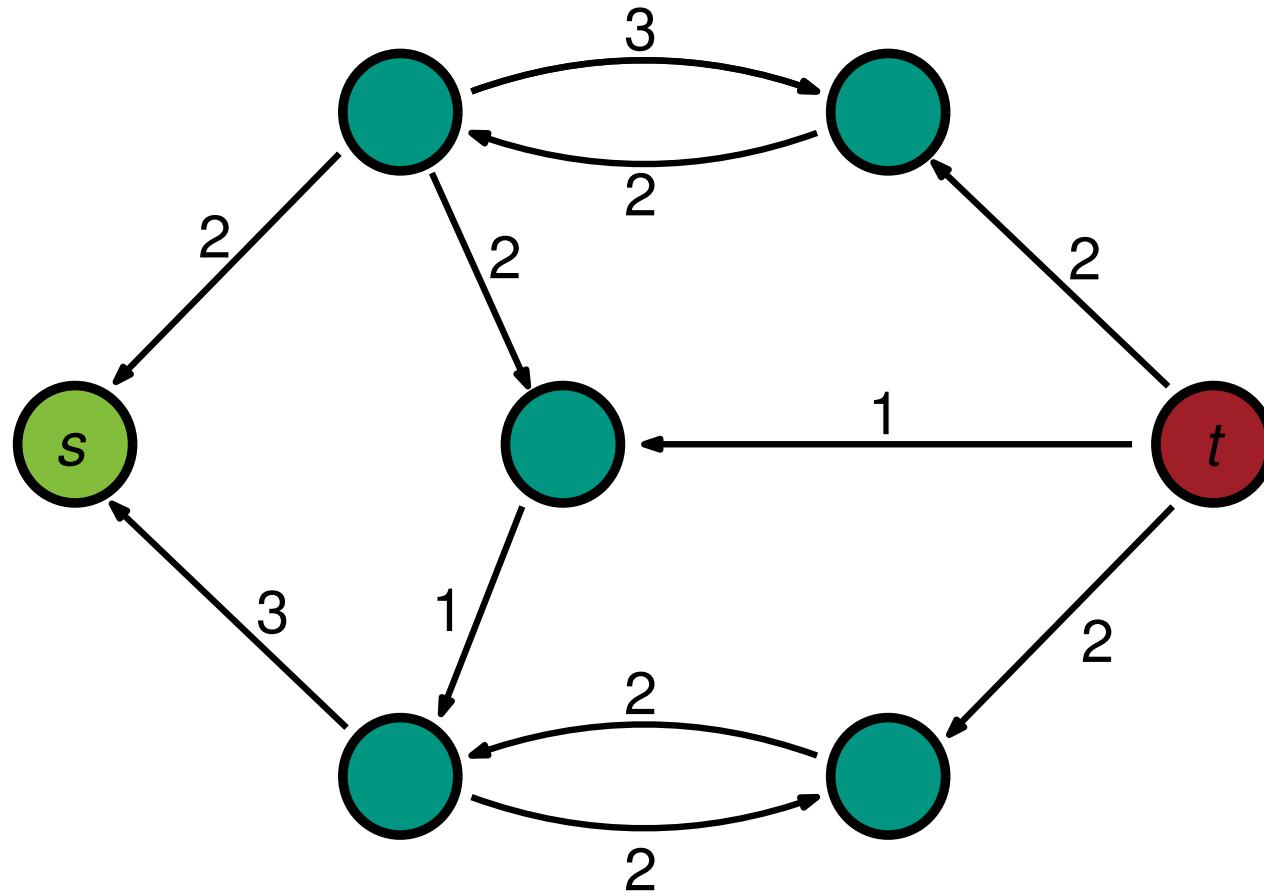
**Residual Graph**  $G_f$

# Flows



# Minimum $(s, t)$ -Bipartition

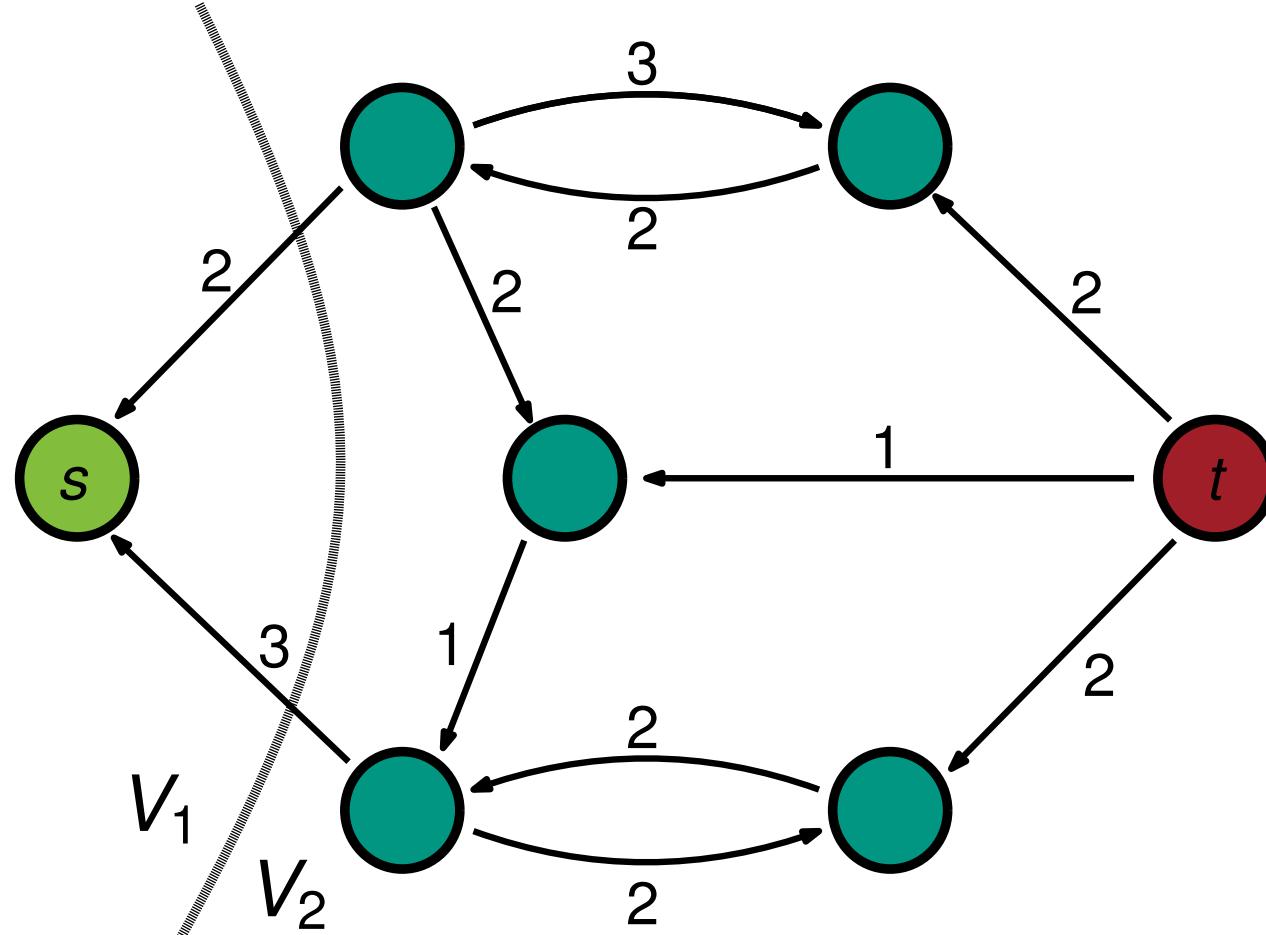
All nodes *reachable* from  $s$  are part of  $V_1$  and  $V_2 = V \setminus V_1$



Residual Graph  $G_f$  of a maximum flow  $f$

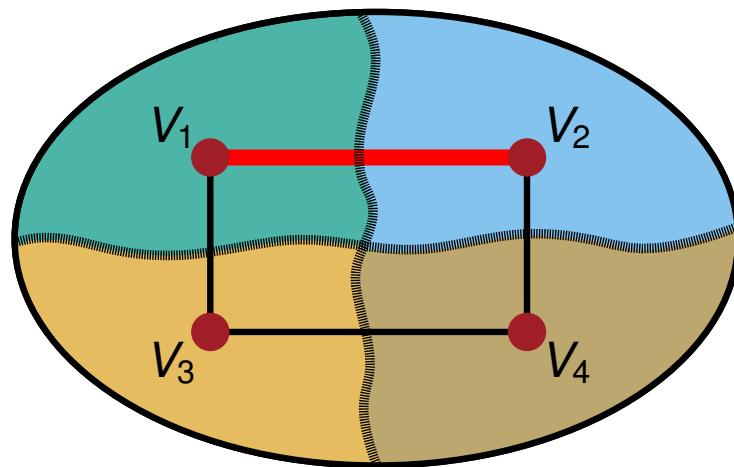
# Minimum $(s, t)$ -Bipartition

All nodes *reachable* from  $s$  are part of  $V_1$  and  $V_2 = V \setminus V_1$

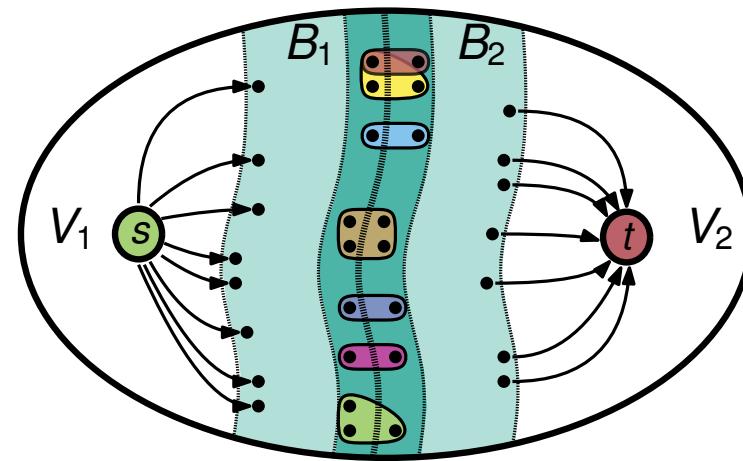


Residual Graph  $G_f$  of a maximum flow  $f$

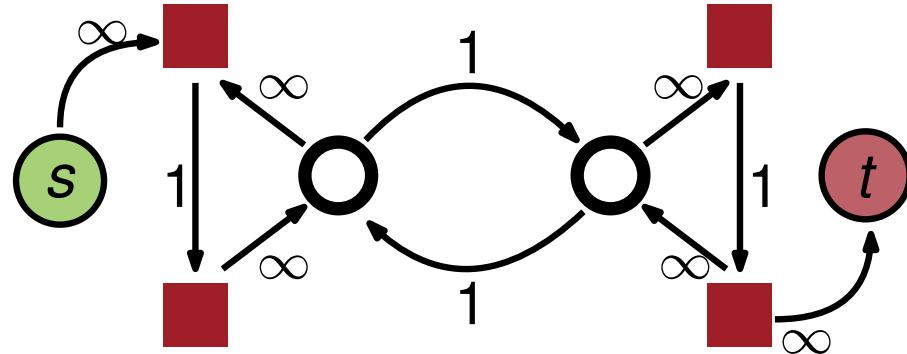
# Our Flow-Based Refinement Framework



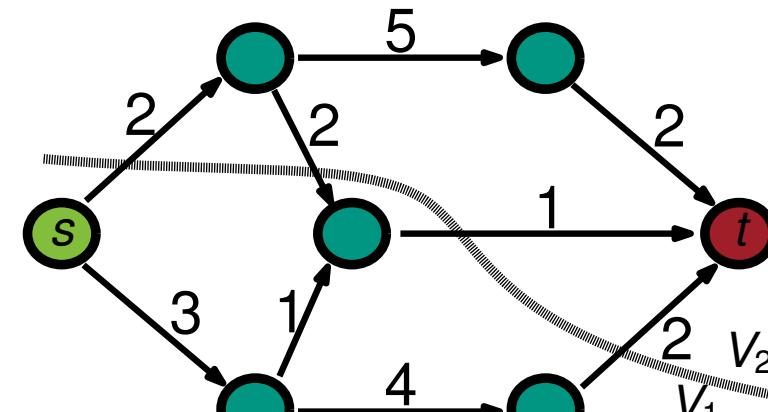
Select two adjacent blocks for refinement



Build Flow Problem

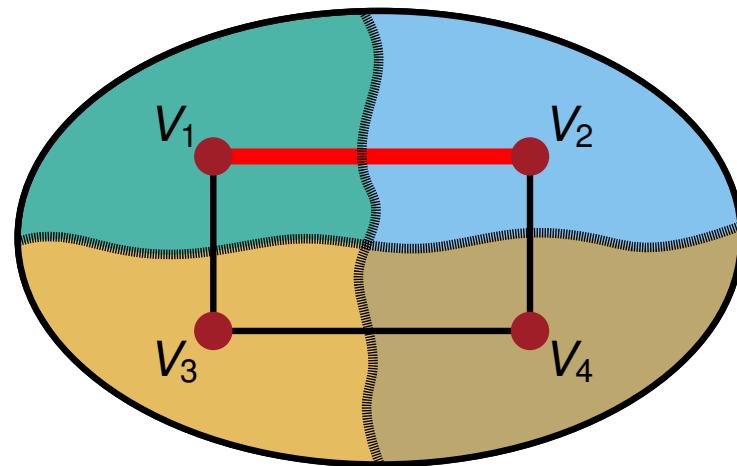


Solve Flow Problem

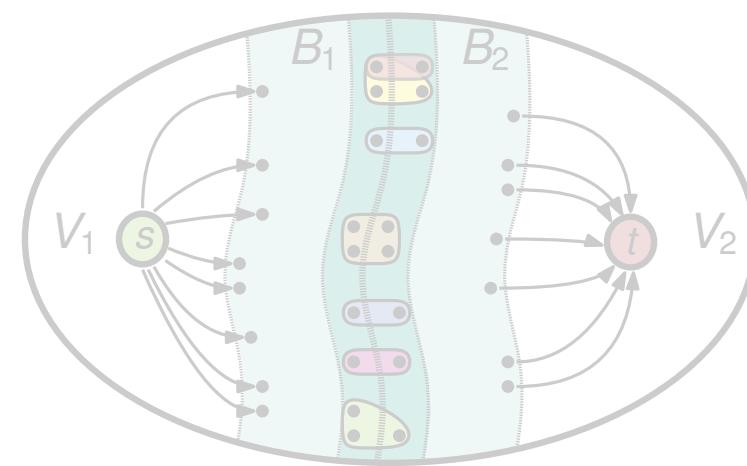


Find feasible minimum cut

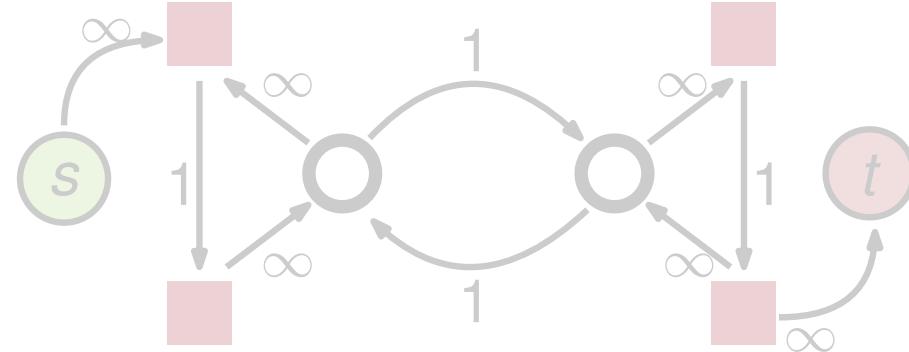
# Our Flow-Based Refinement Framework



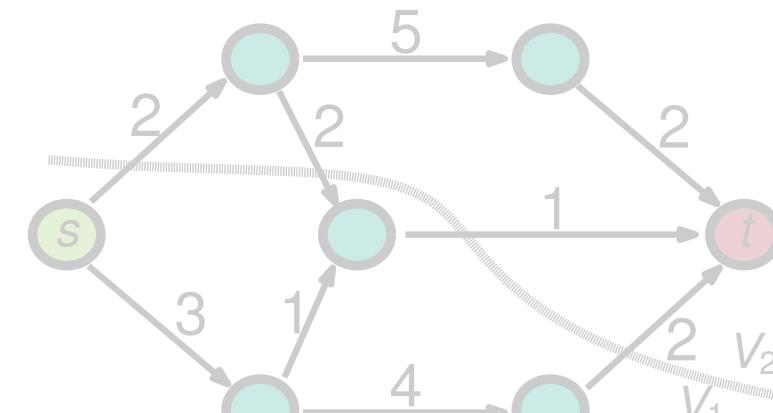
Select two adjacent blocks for refinement



Build Flow Problem



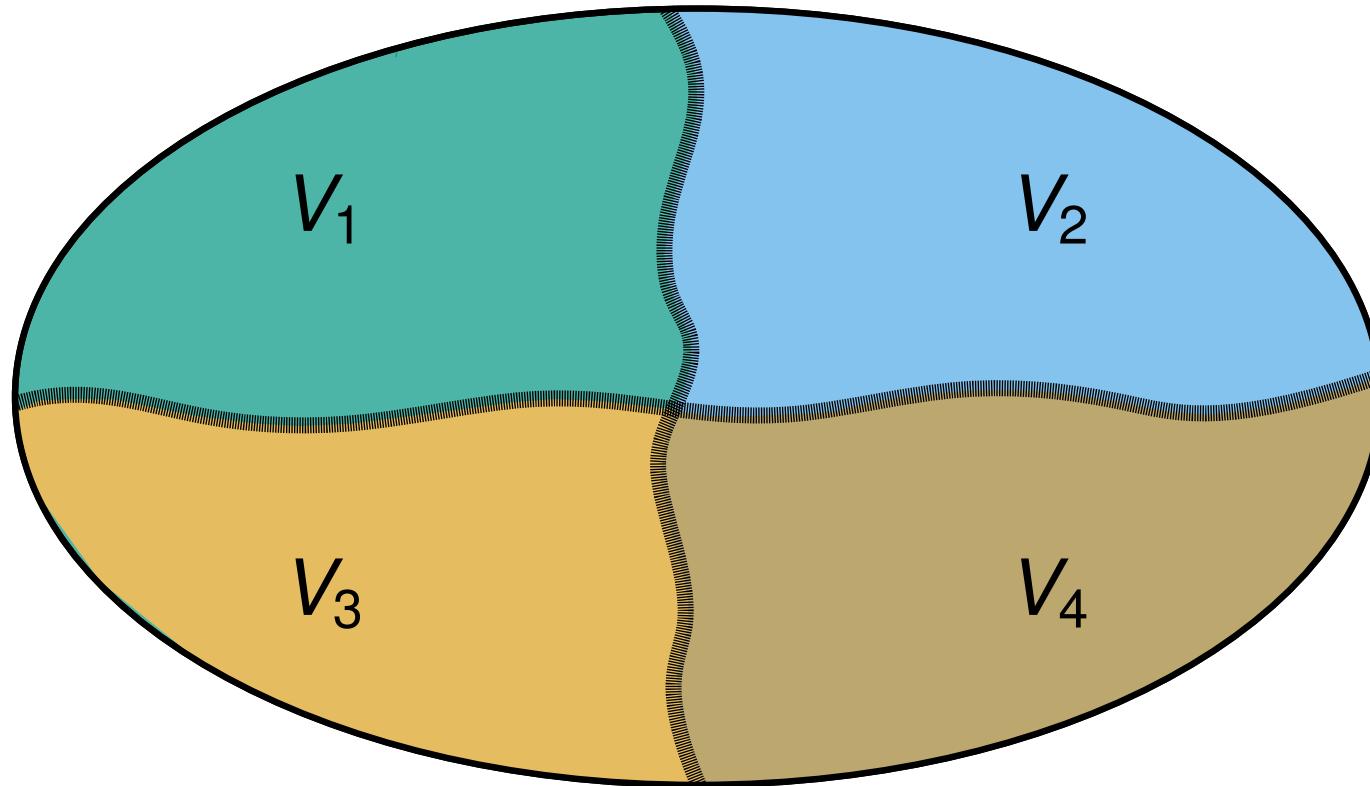
Solve Flow Problem



Find feasible minimum cut

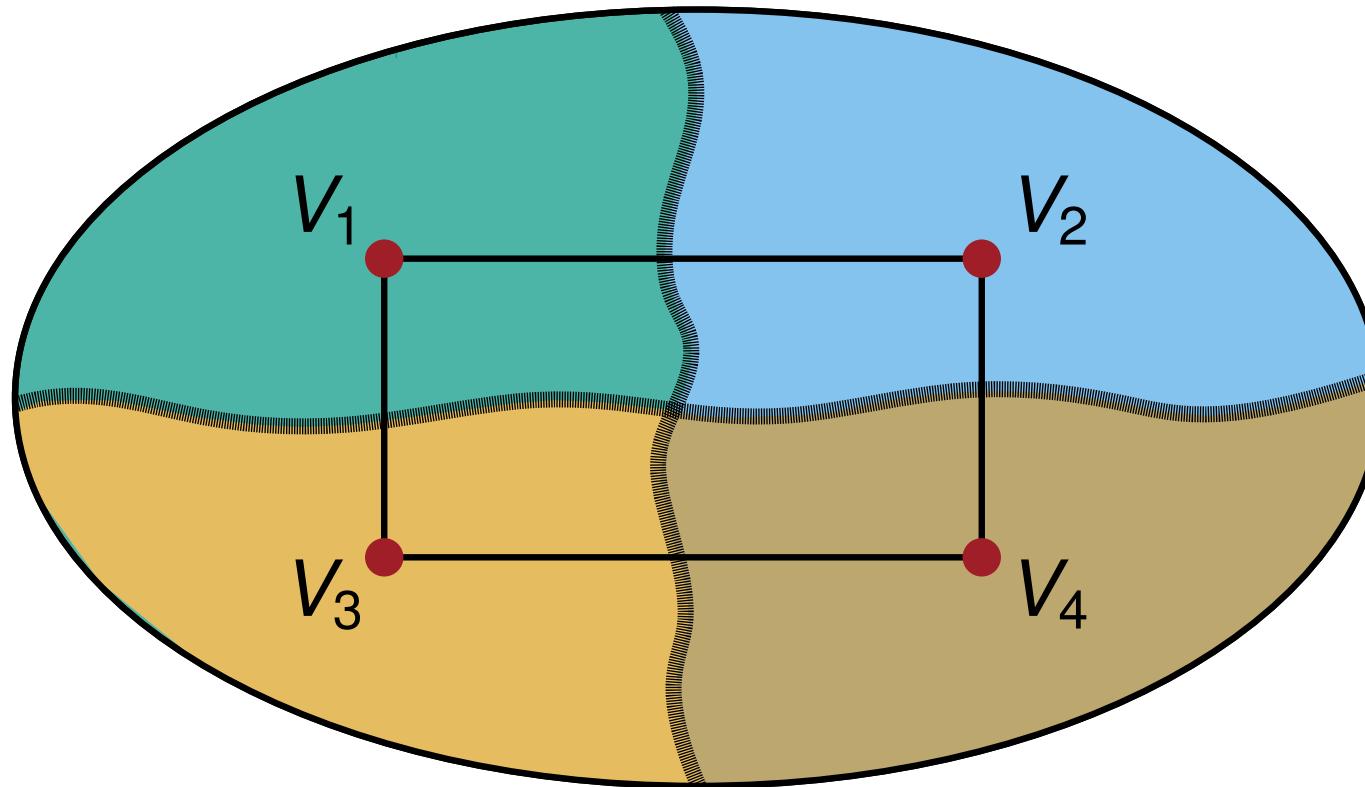
# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]



# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]



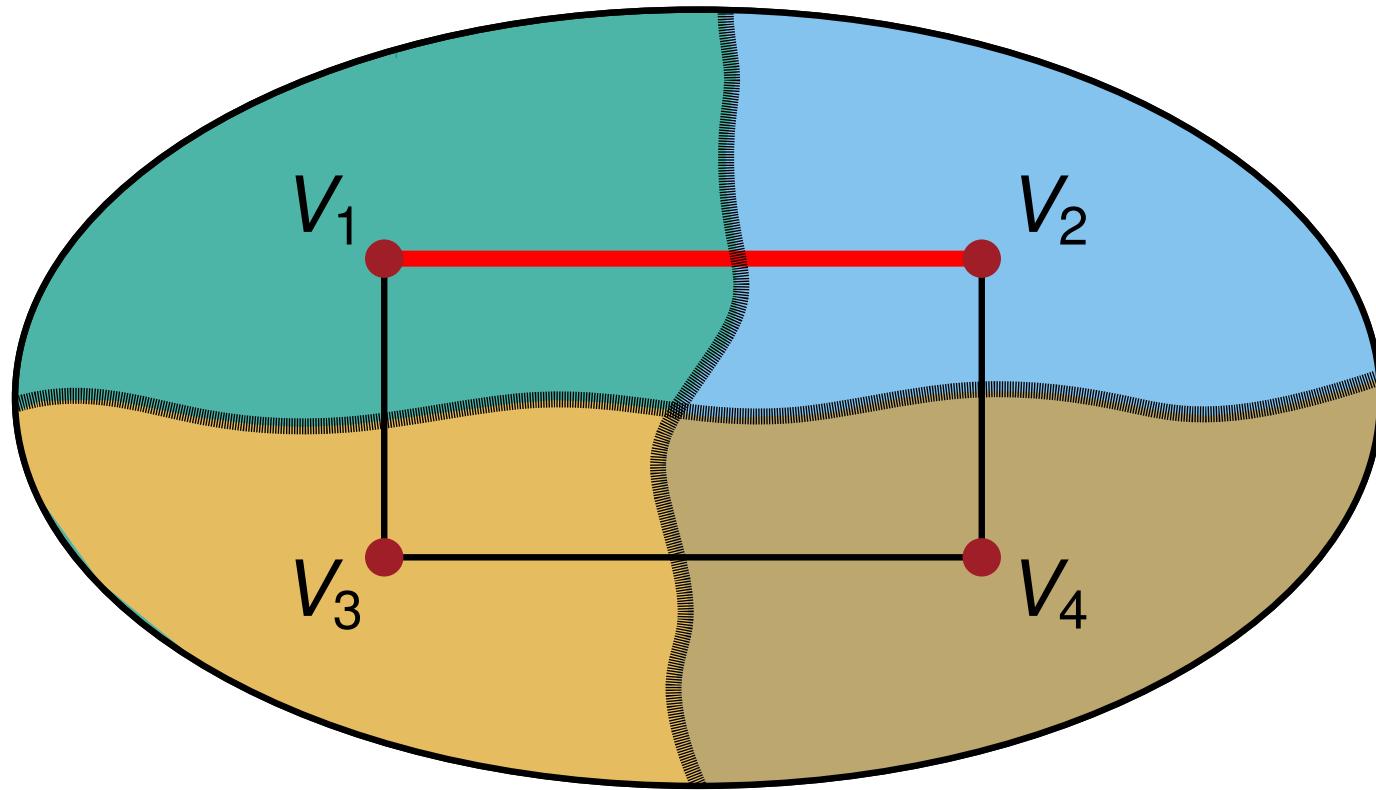
Build Quotient Graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 1**

$\text{refine}(V_1, V_2) = \text{Improvement!}$



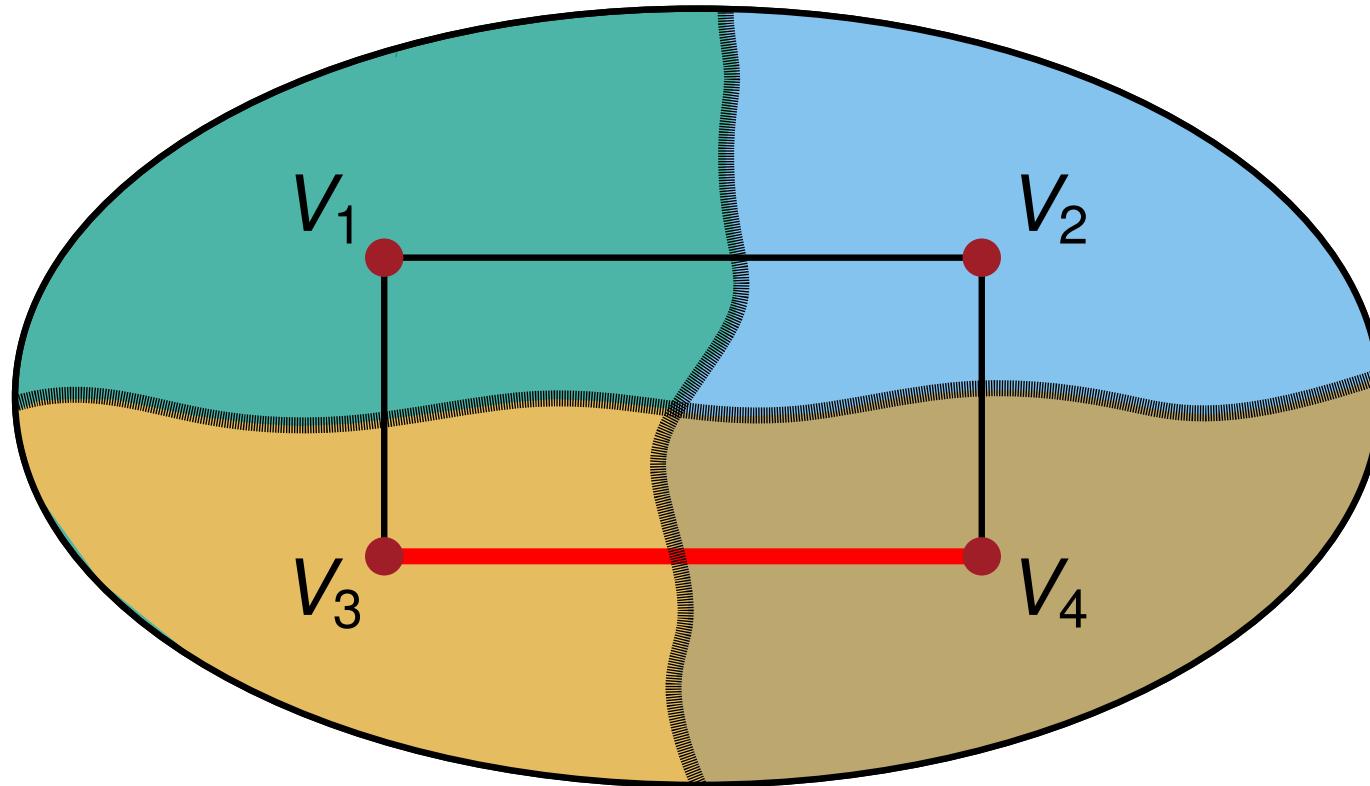
Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 1**

$\text{refine}(V_3, V_4) = \text{No Improvement!}$



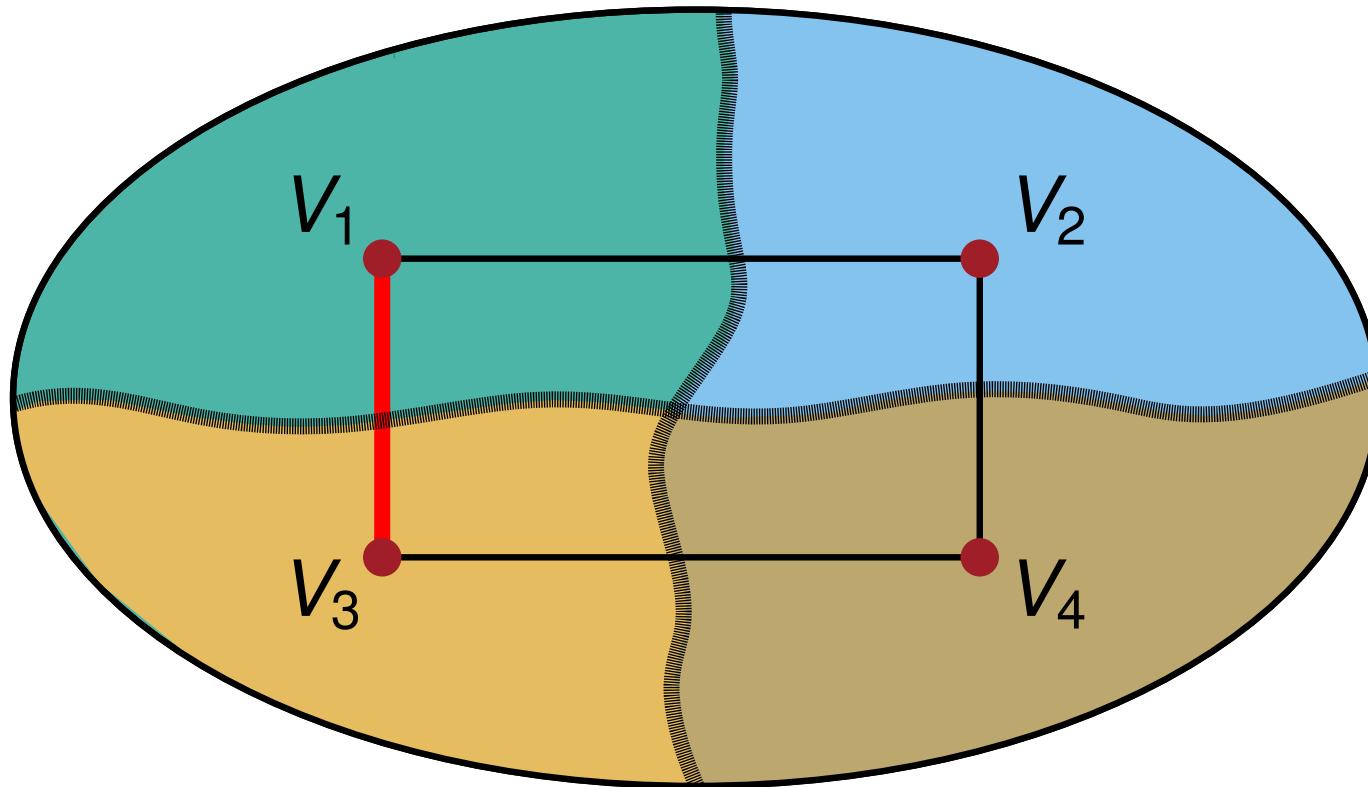
Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 1**

$\text{refine}(V_1, V_3) = \text{No Improvement!}$



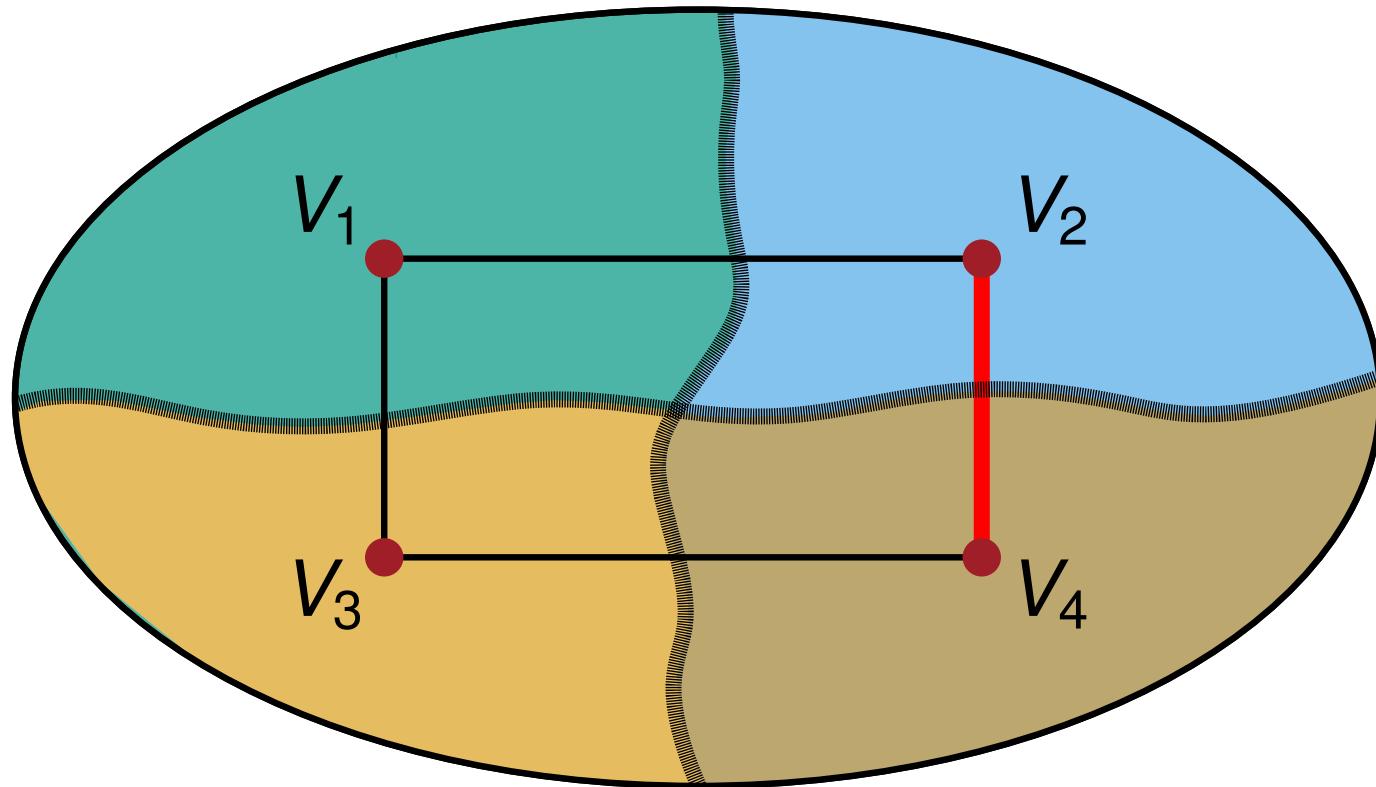
Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 1**

$\text{refine}(V_2, V_4) = \text{No Improvement!}$

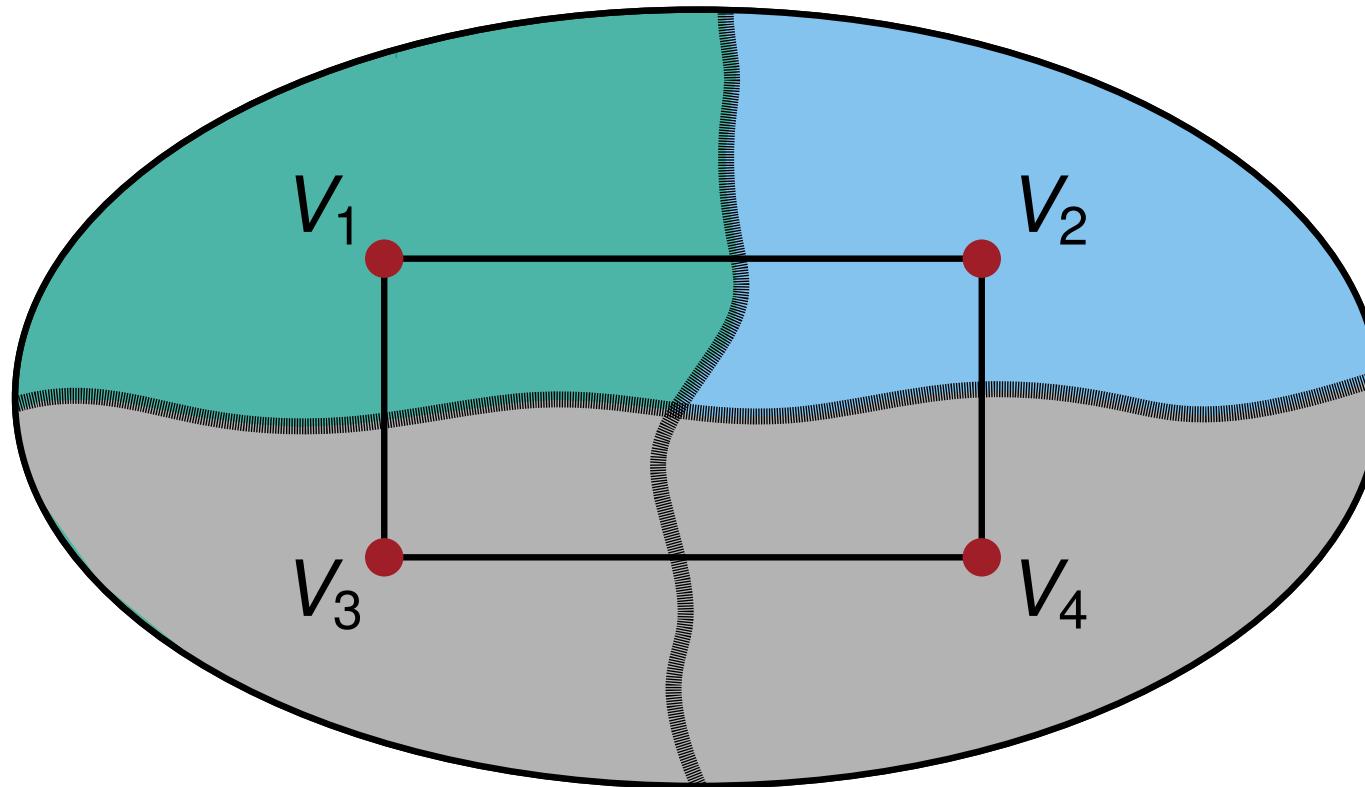


Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 1** Boundary did not change  $\Rightarrow$  Mark block as **inactive**



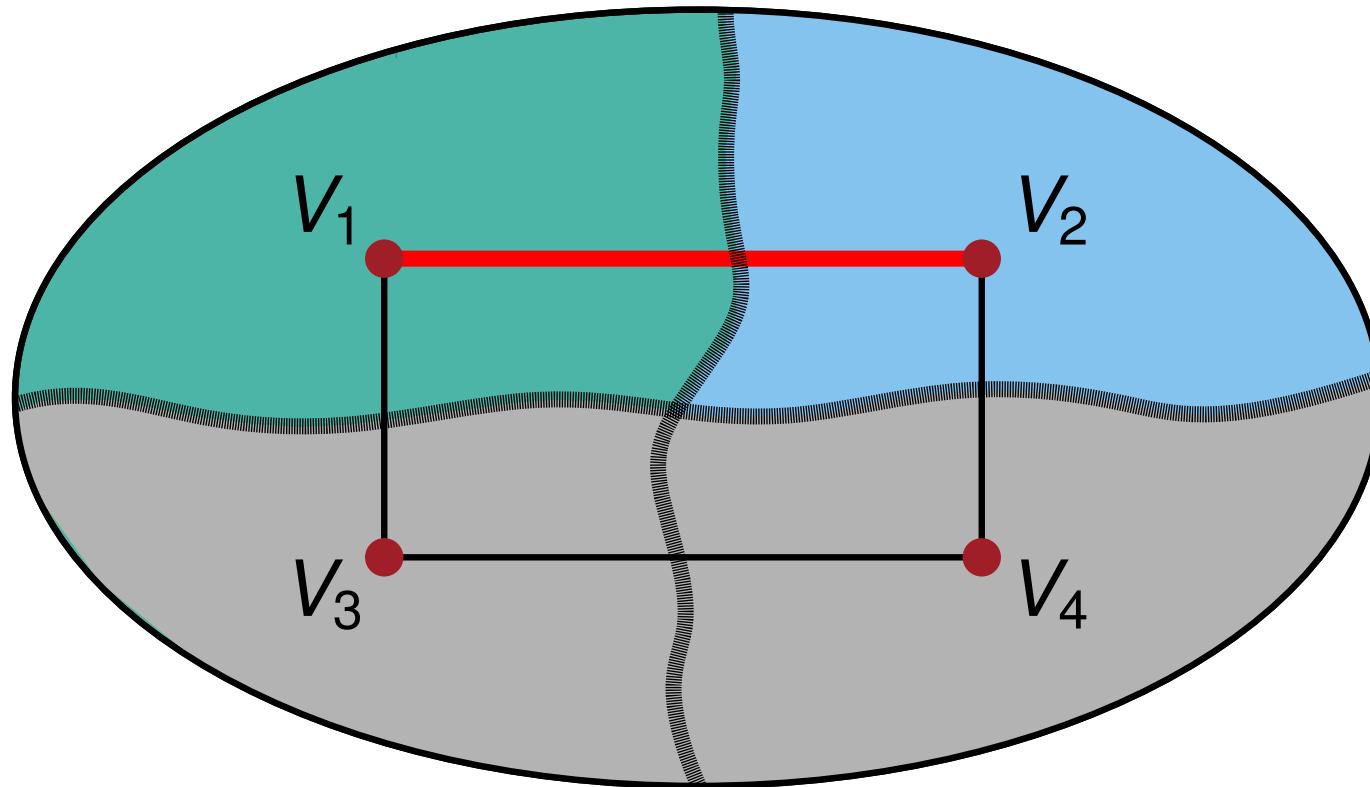
Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 2**

$\text{refine}(V_1, V_2) = \text{No Improvement!}$



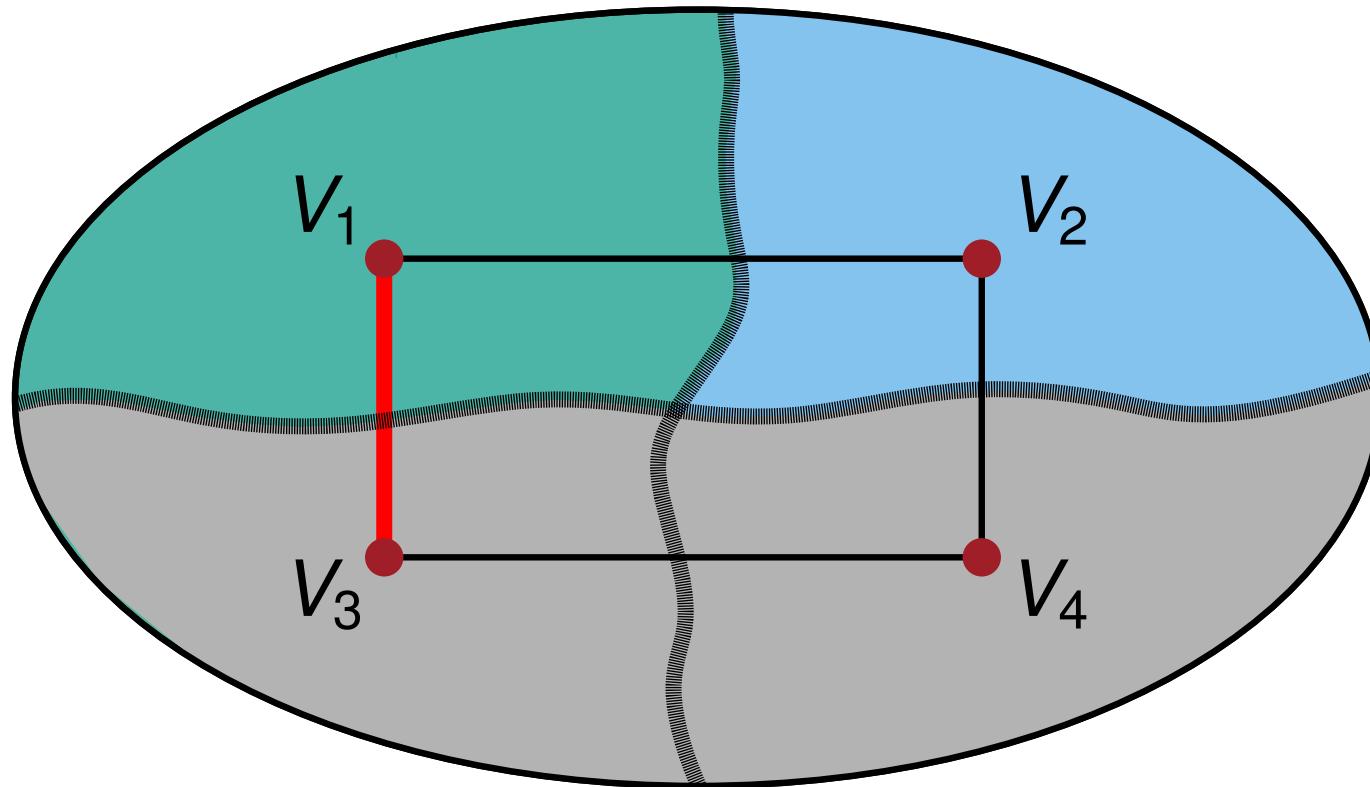
Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 2**

$\text{refine}(V_1, V_3) = \text{No Improvement!}$



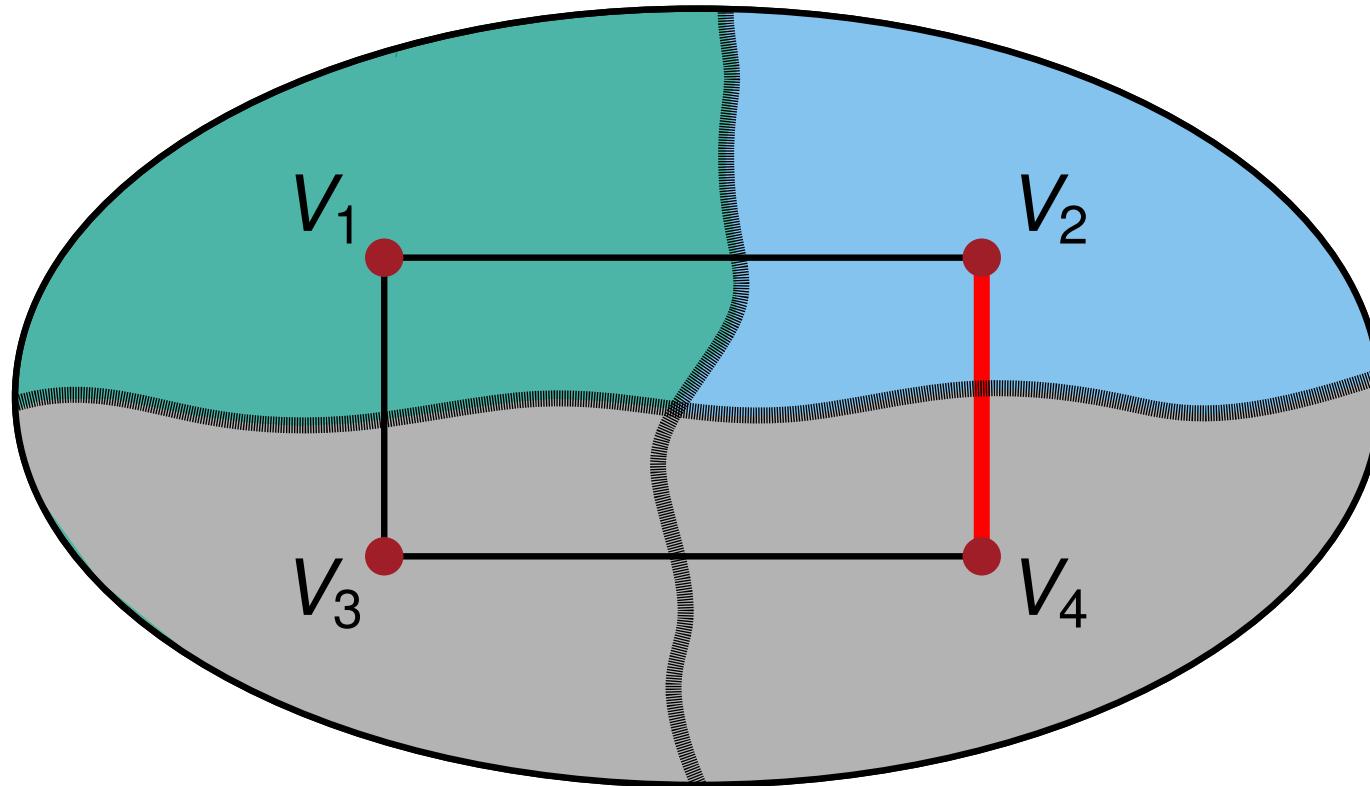
Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 2**

$\text{refine}(V_2, V_4) = \text{No Improvement!}$

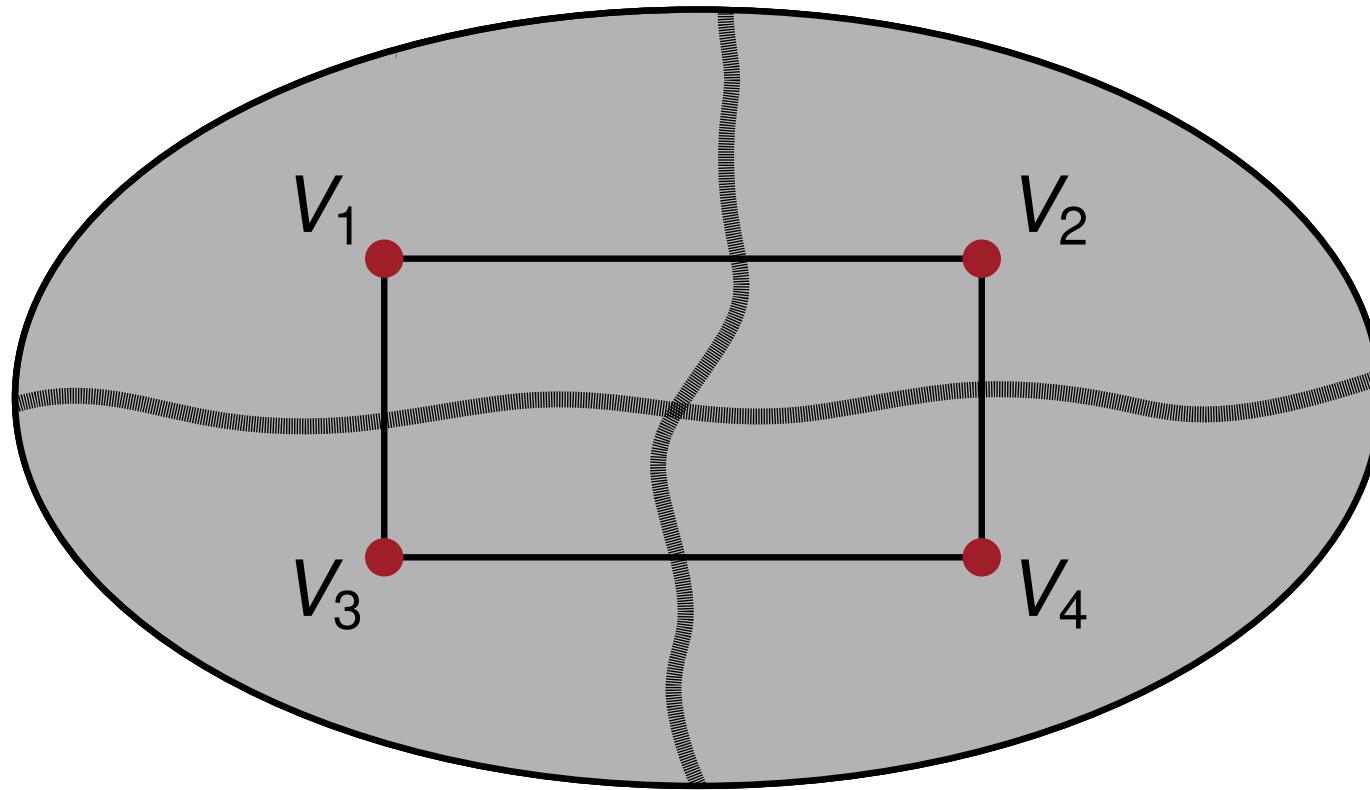


Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

KaFFPa [Sanders, Schulz 11]

**Round 2** Boundary did not change  $\Rightarrow$  Mark block as **inactive**

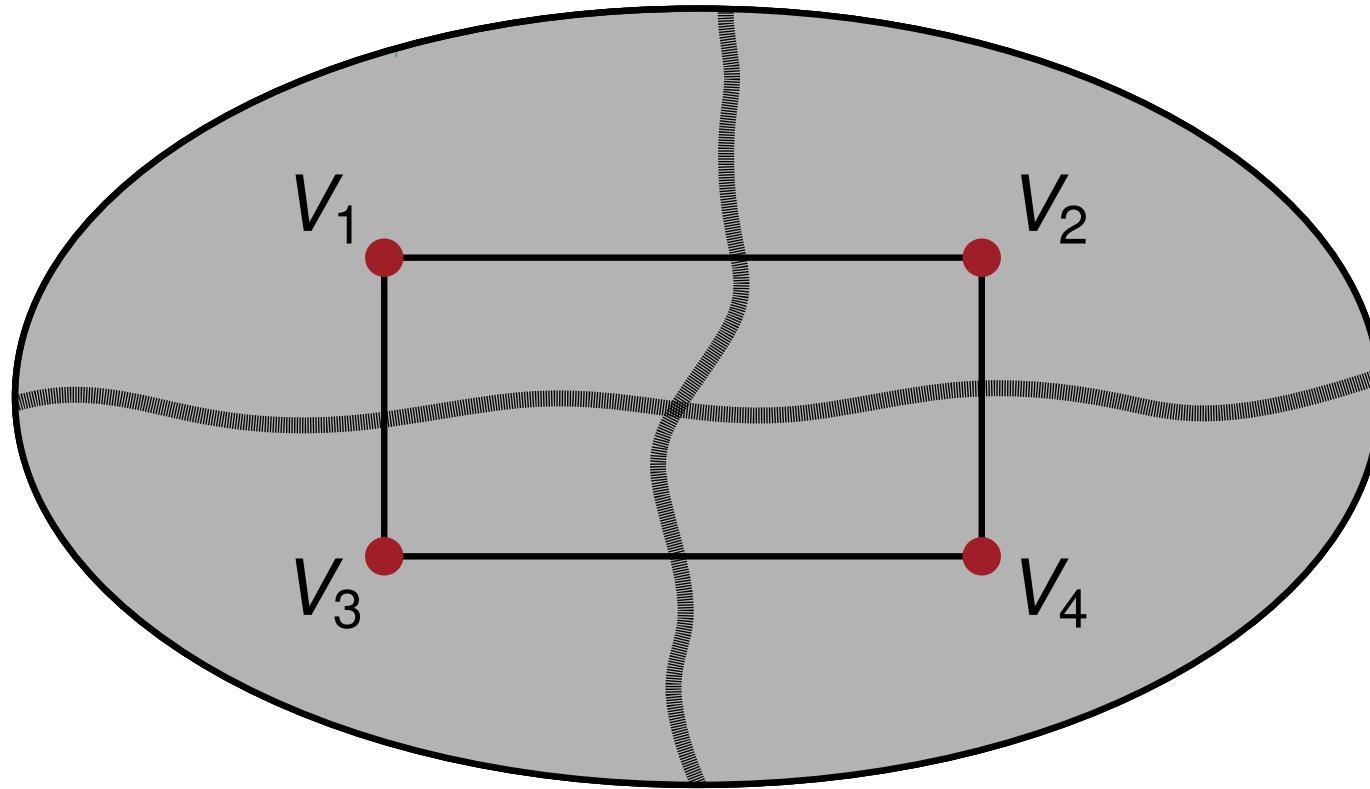


Using 2-way refinement algorithm for **active** blocks of the quotient graph

# Active Block Scheduling

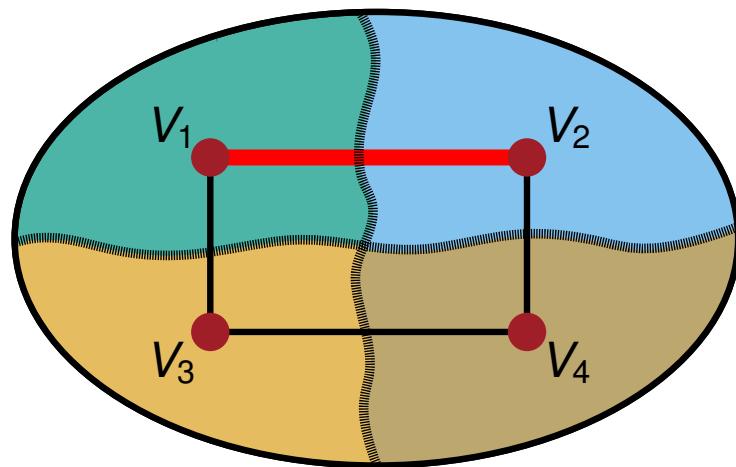
KaFFPa [Sanders, Schulz 11]

**Round 2** All blocks are **inactive**  $\Rightarrow$  Algorithm terminates

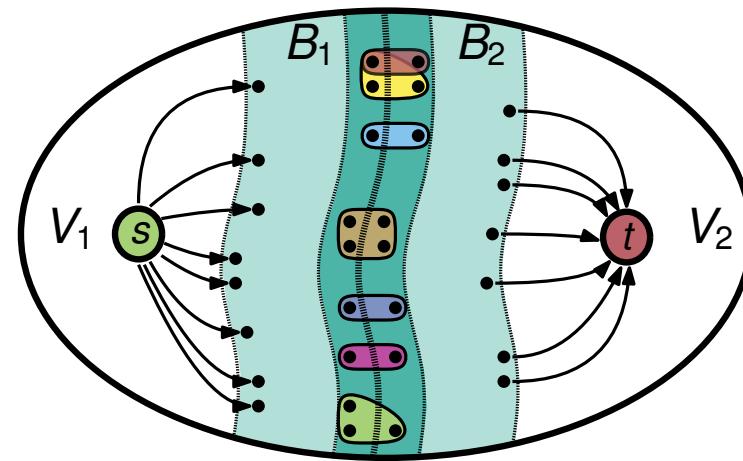


Using 2-way refinement algorithm for **active** blocks of the quotient graph

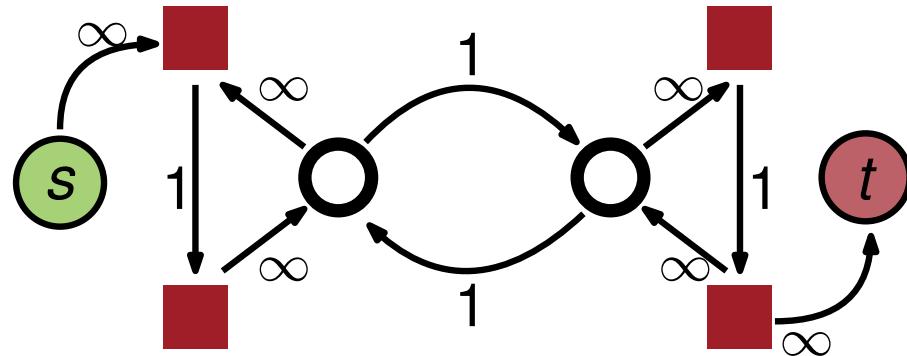
# Our Flow-Based Refinement Framework



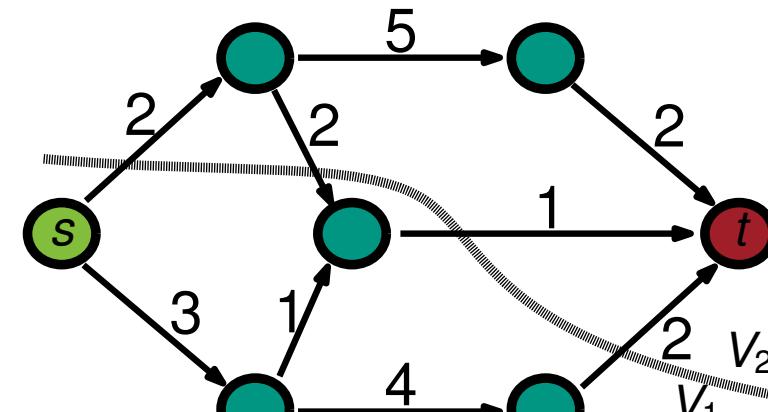
Select two adjacent blocks for refinement



Build Flow Problem

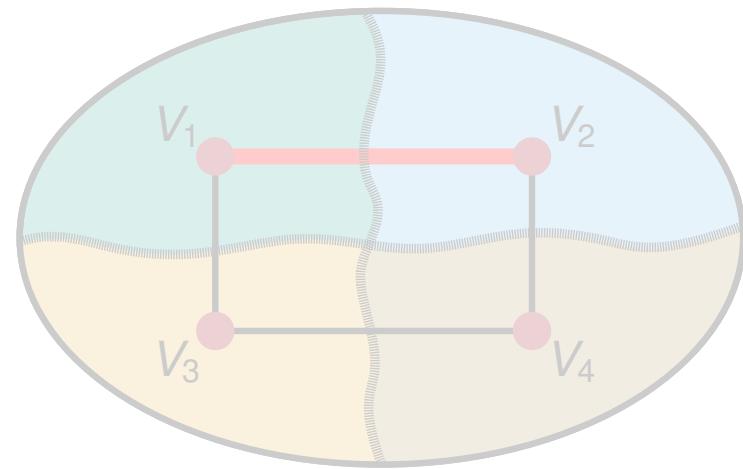


Solve Flow Problem

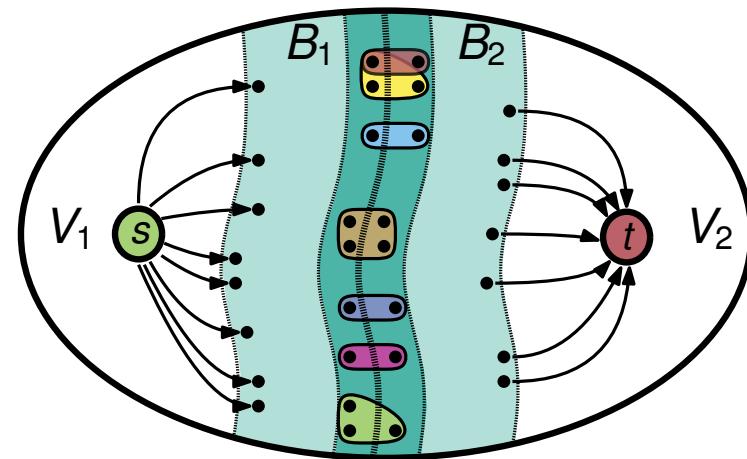


Find feasible minimum cut

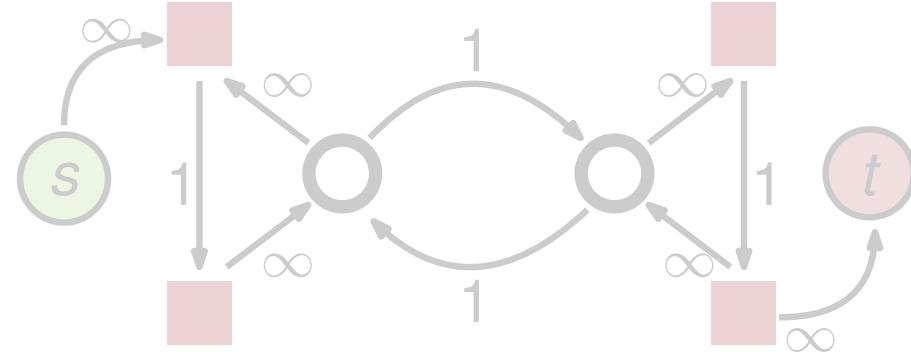
# Our Flow-Based Refinement Framework



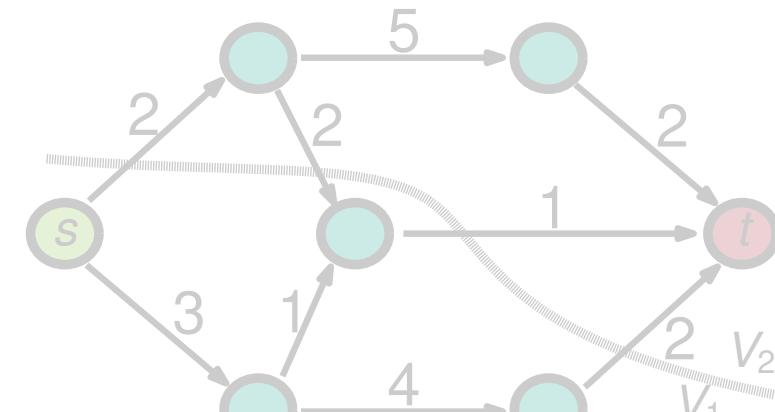
Select two adjacent blocks for refinement



Build Flow Problem



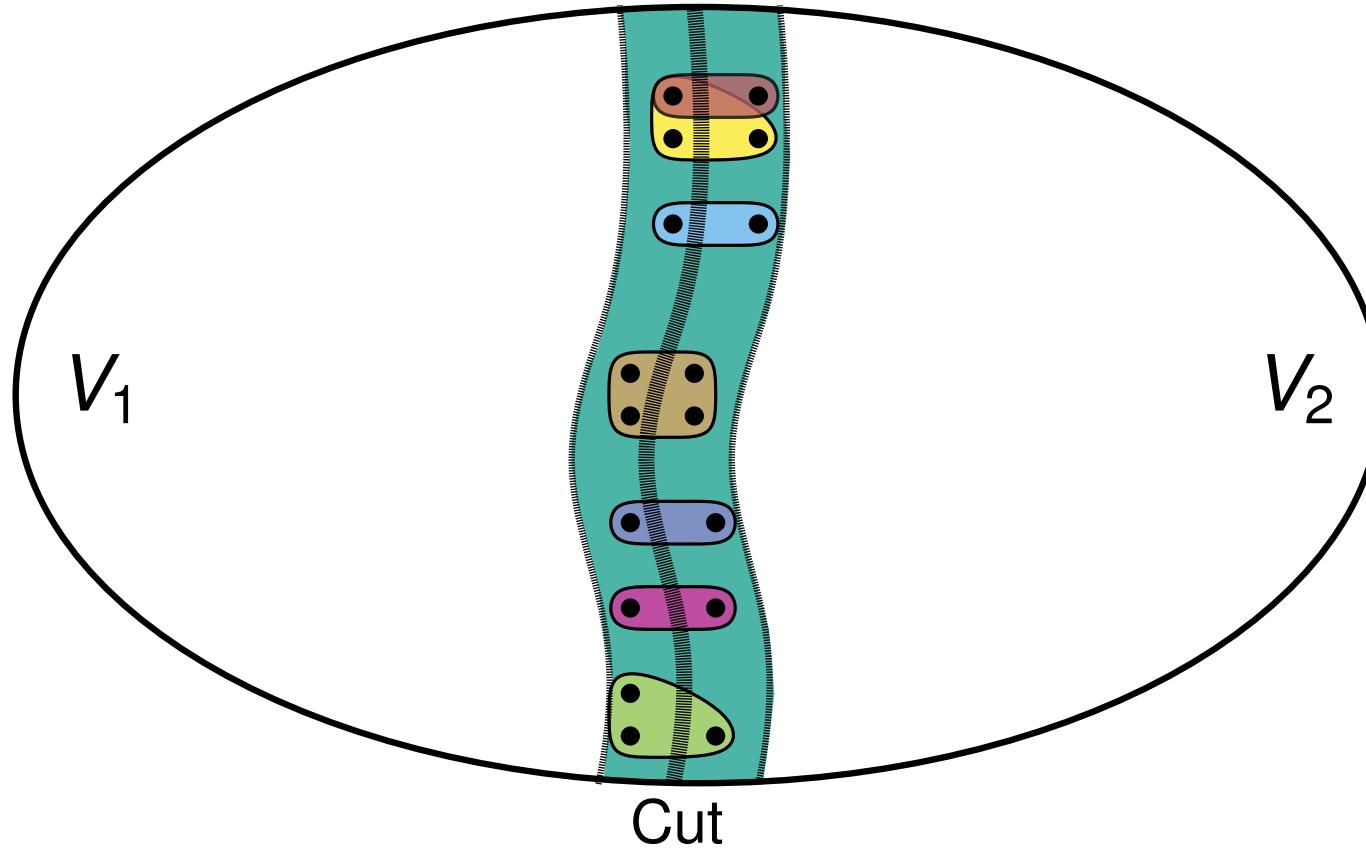
Solve Flow Problem



Find feasible minimum cut

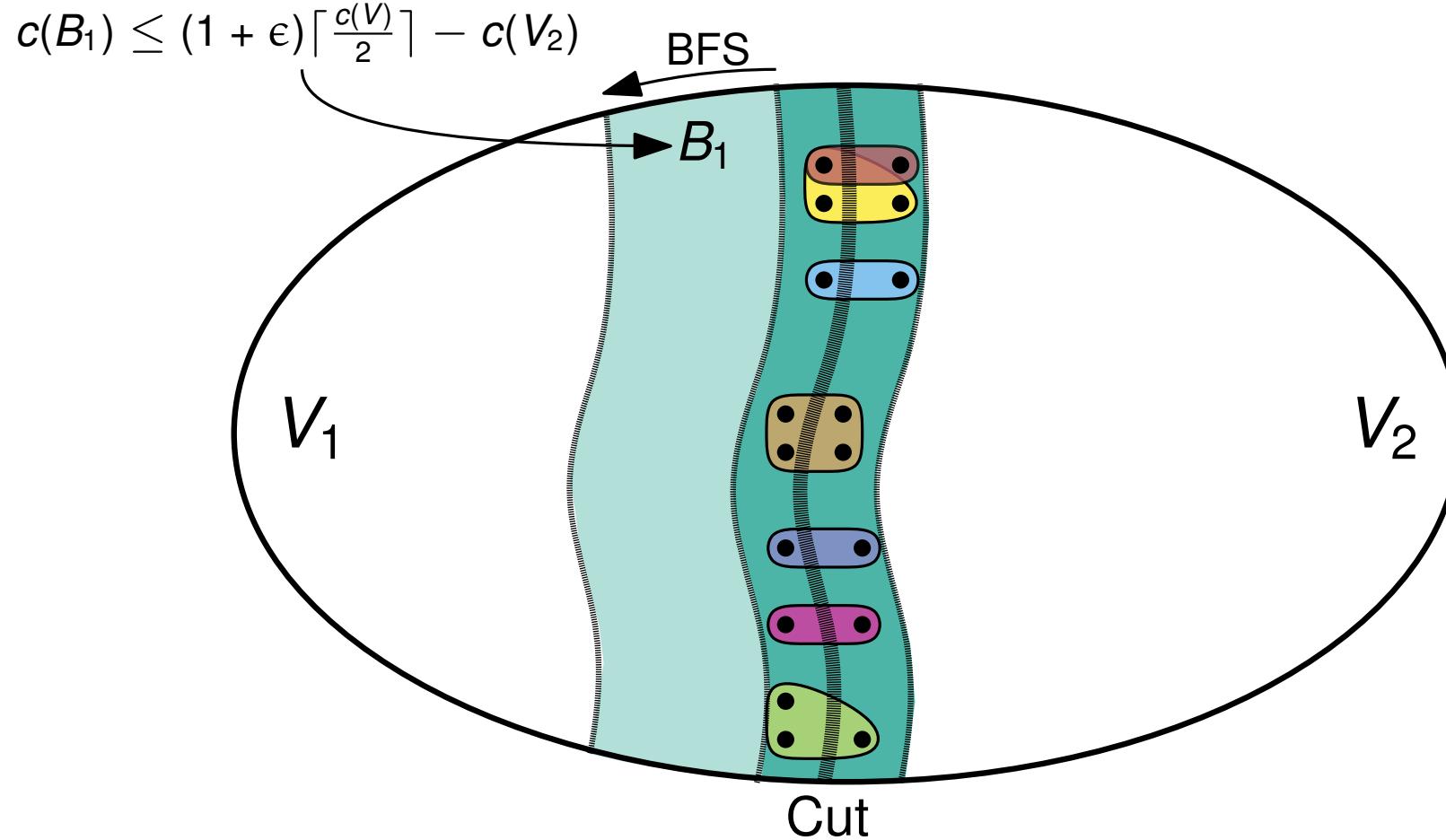
# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]



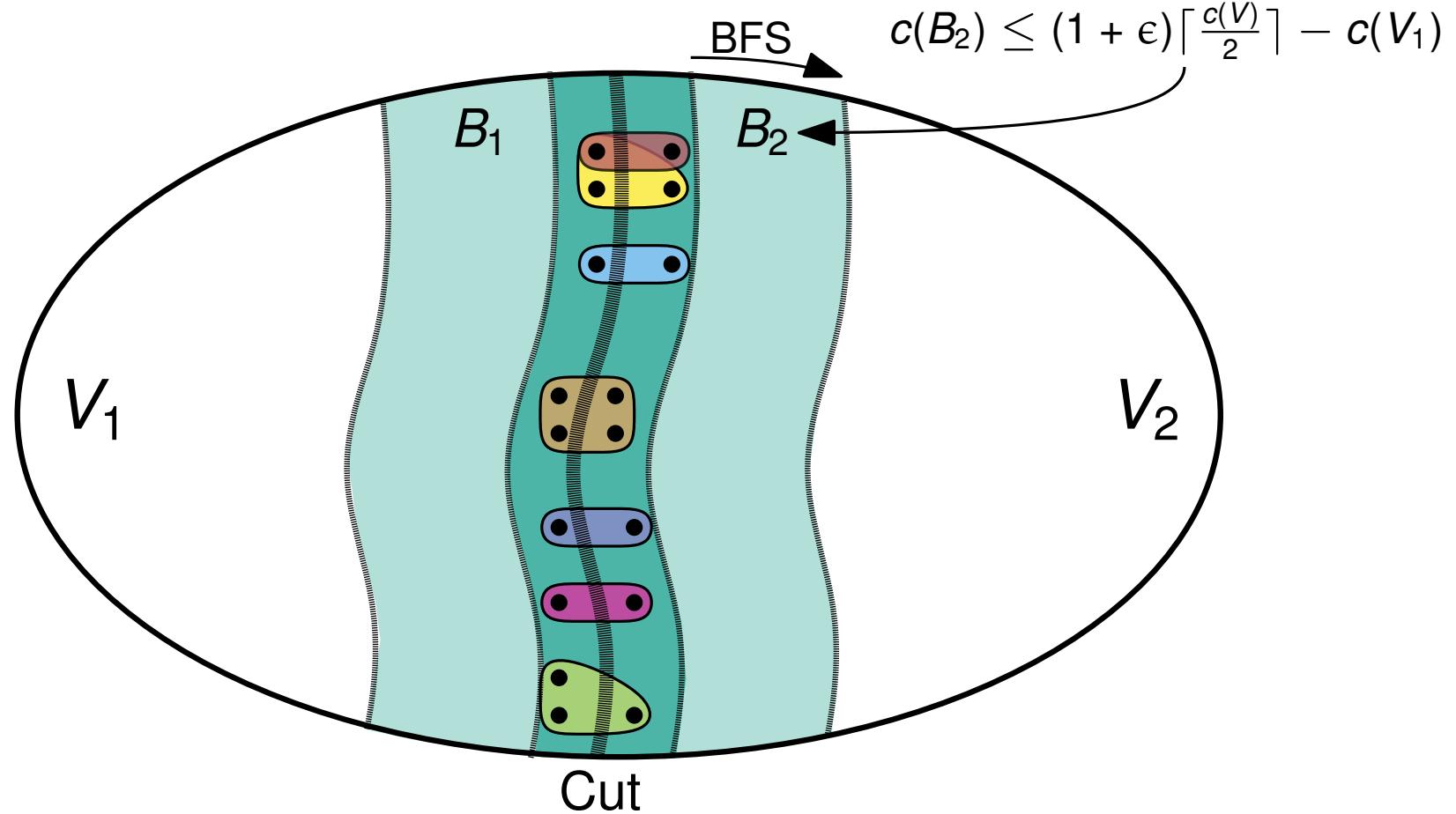
# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]



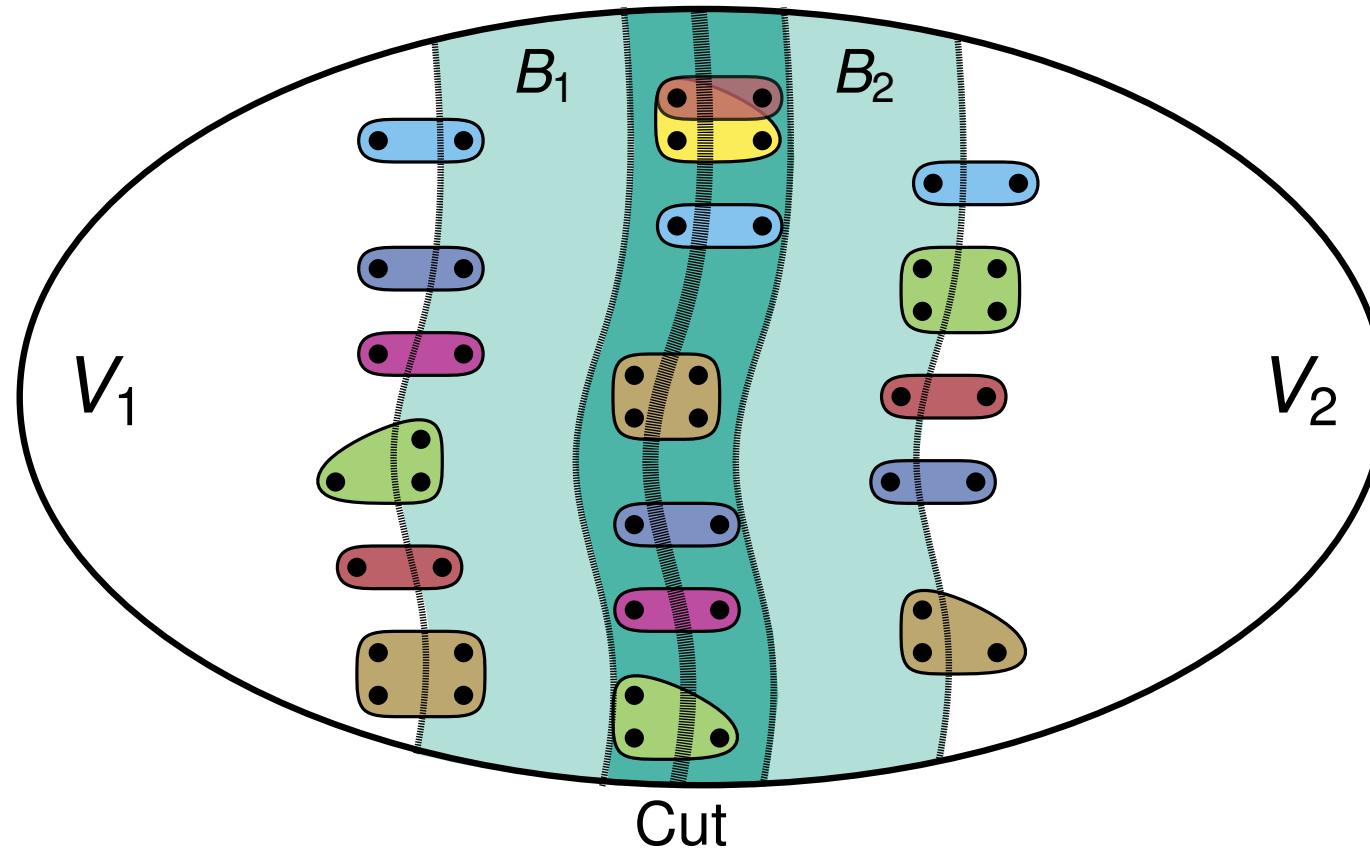
# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]



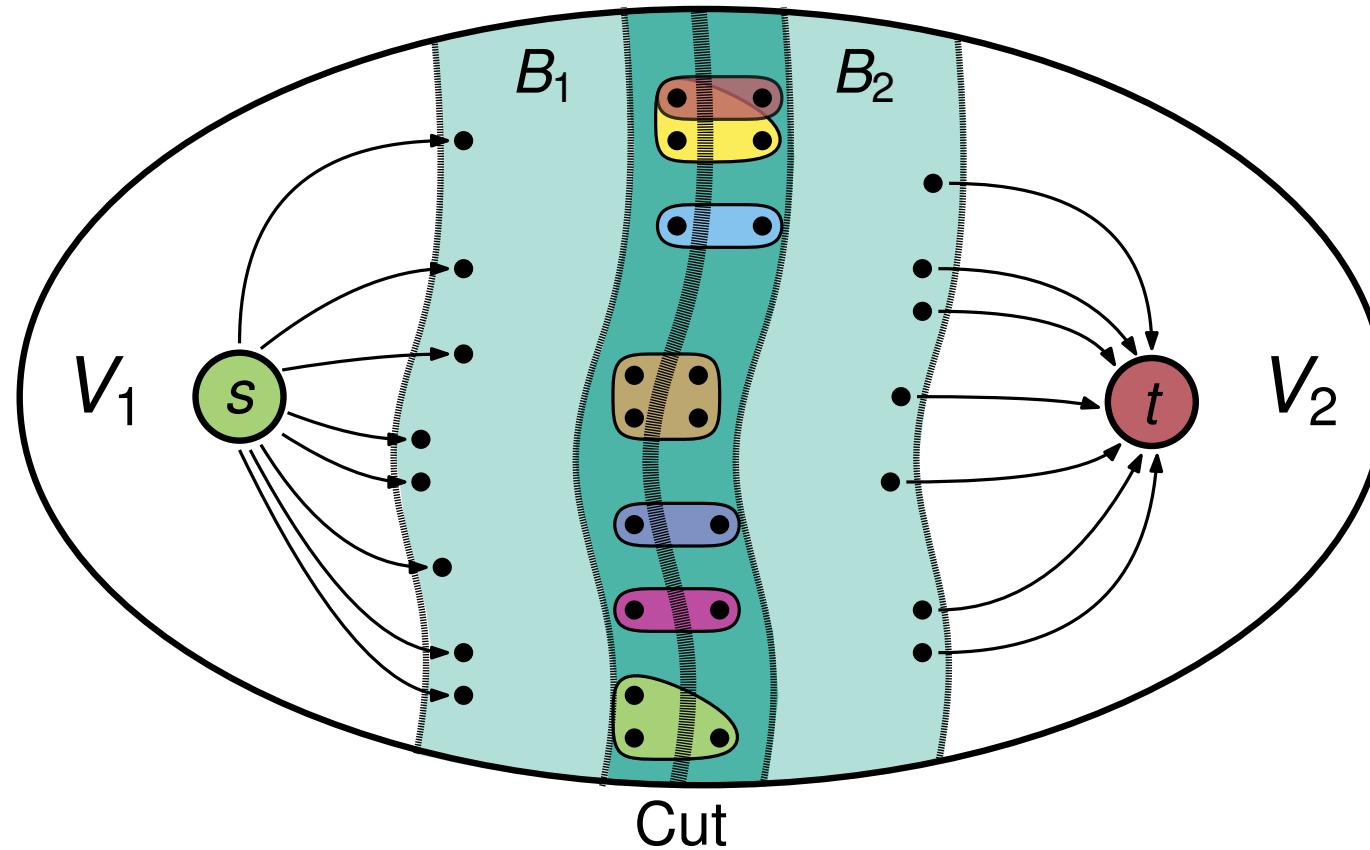
# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]



# Adaptive Flow Iterations

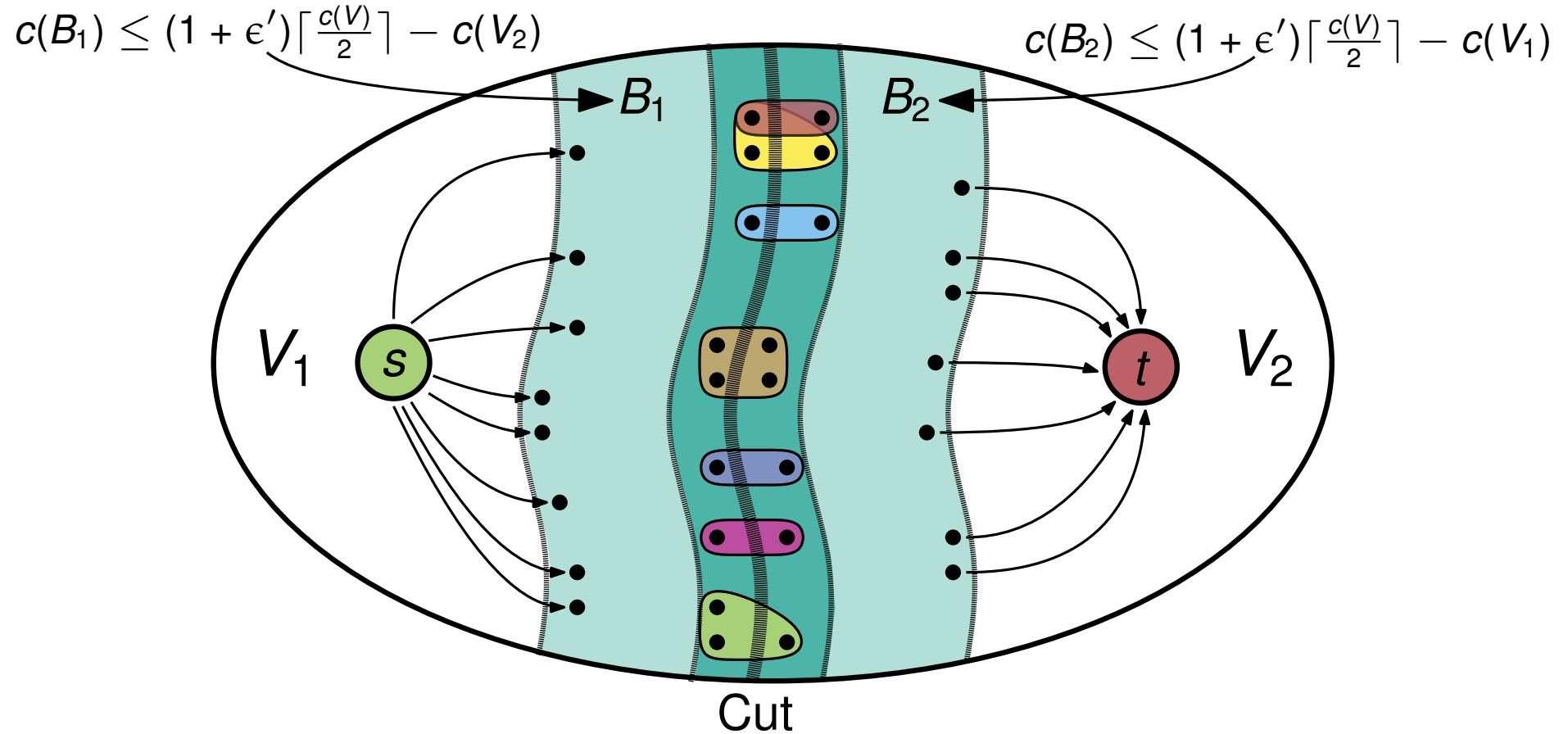
KaFFPa [Sanders, Schulz 11]



# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]

Use  $\epsilon' = \alpha\epsilon$  instead of  $\epsilon$

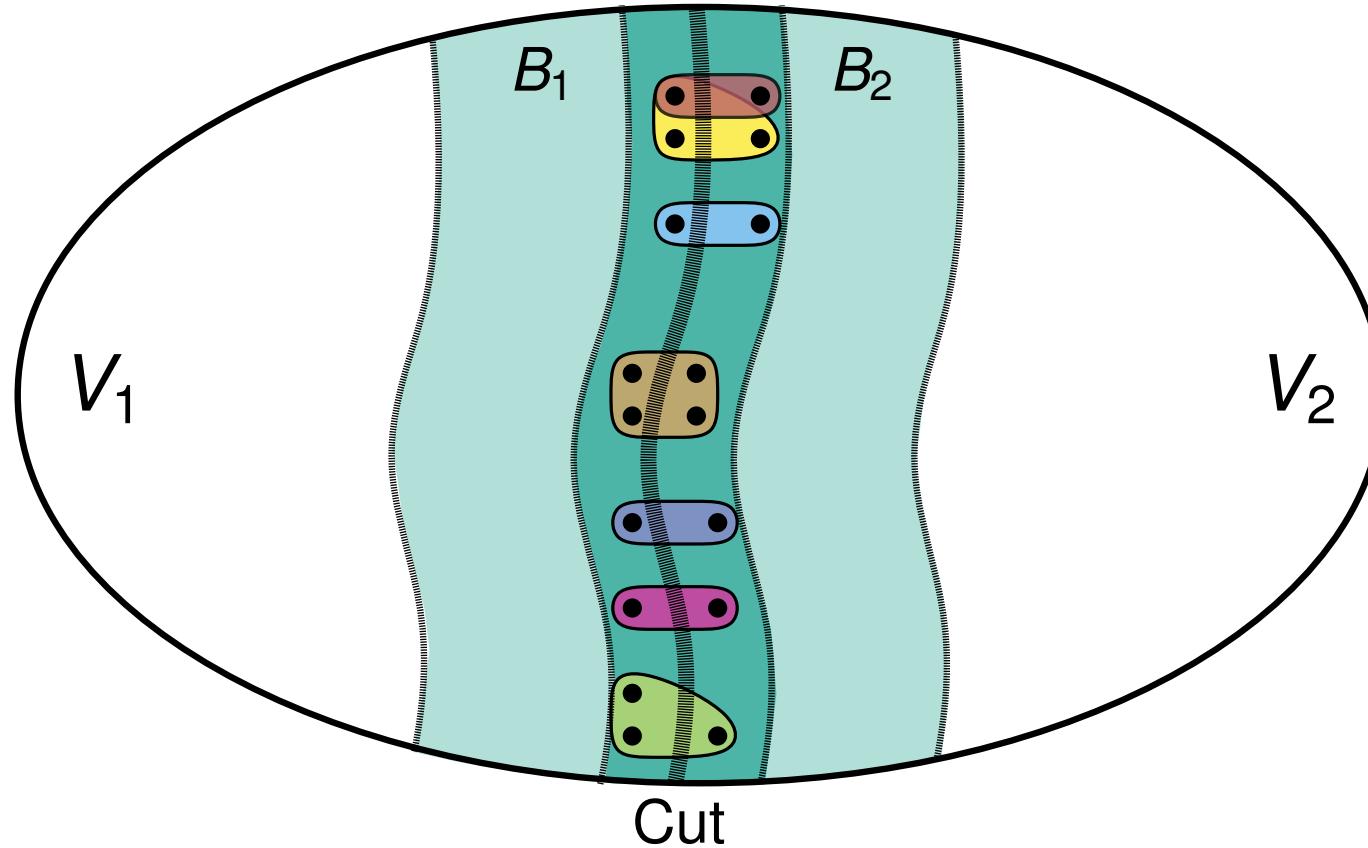


# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]

Use  $\epsilon' = \alpha\epsilon$  instead of  $\epsilon$

$\alpha = 1 \Rightarrow \text{Improvement Found} \Rightarrow \alpha = \max(2\alpha, \alpha') = 2$

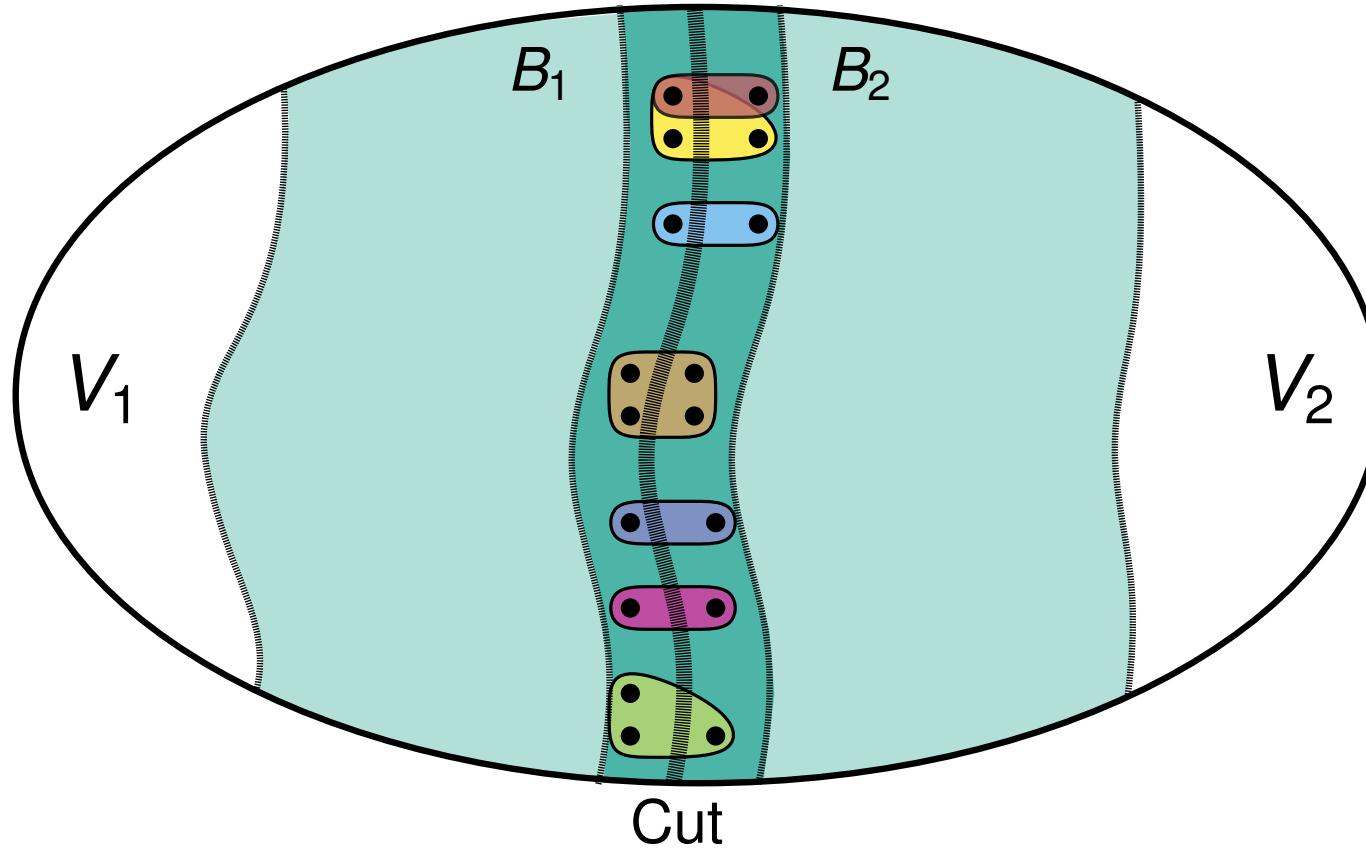


# Adaptive Flow Iterations

KaFFPa [Sanders, Schulz 11]

Use  $\epsilon' = \alpha\epsilon$  instead of  $\epsilon$

$\alpha = 2 \Rightarrow \text{No Improvement} \Rightarrow \alpha = \min(\frac{\alpha}{2}, 1) = 1$

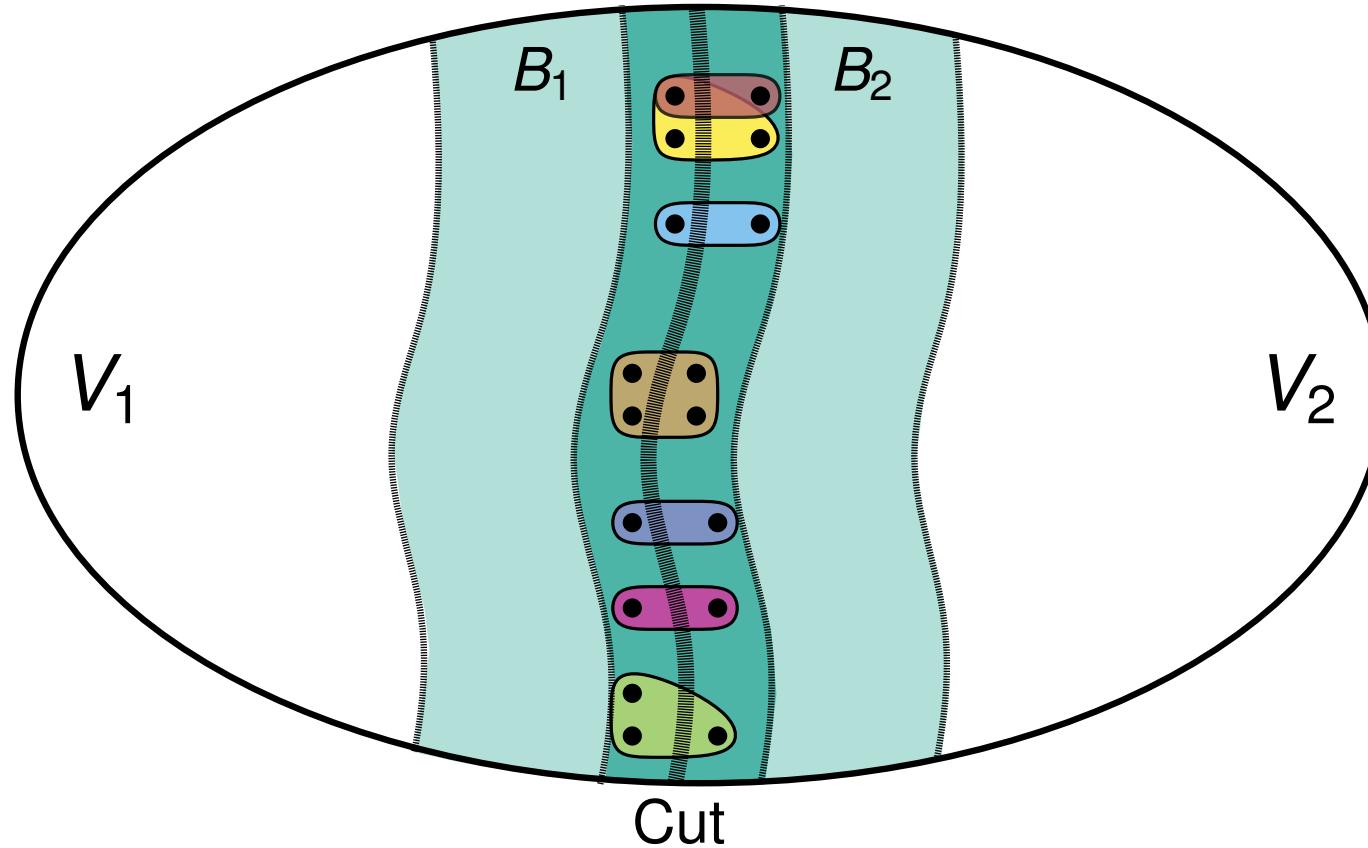


# Adaptive Flow Iterations

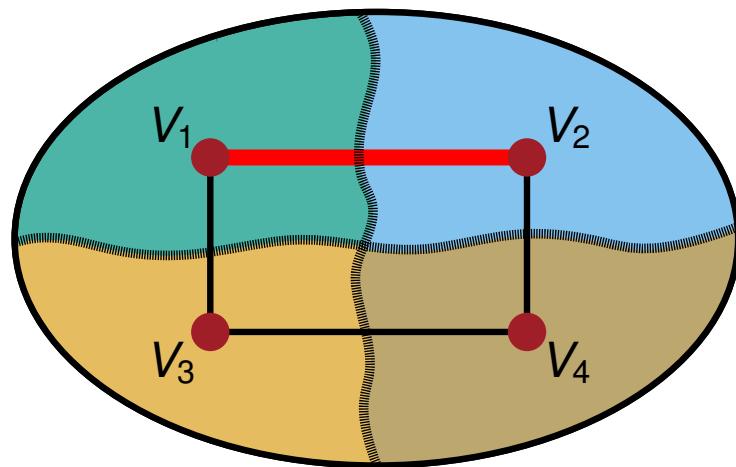
KaFFPa [Sanders, Schulz 11]

Use  $\epsilon' = \alpha\epsilon$  instead of  $\epsilon$

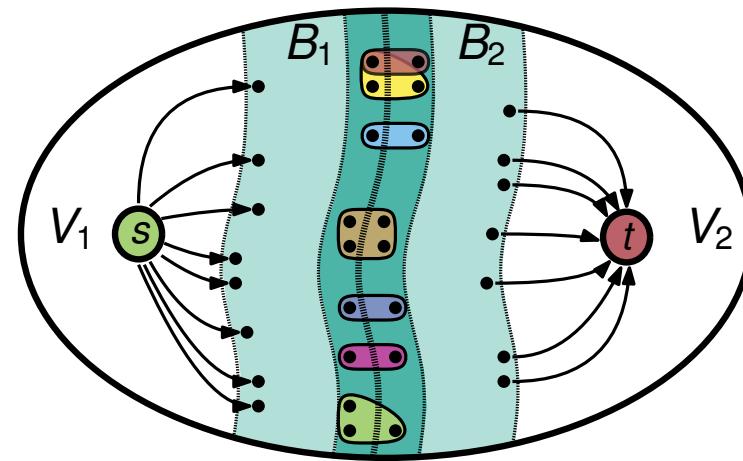
$\alpha = 1 \Rightarrow \text{No Improvement} \Rightarrow \text{Terminate}$



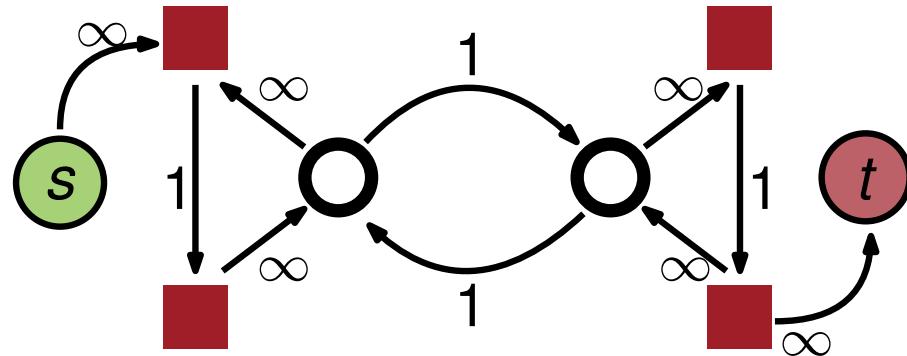
# Our Flow-Based Refinement Framework



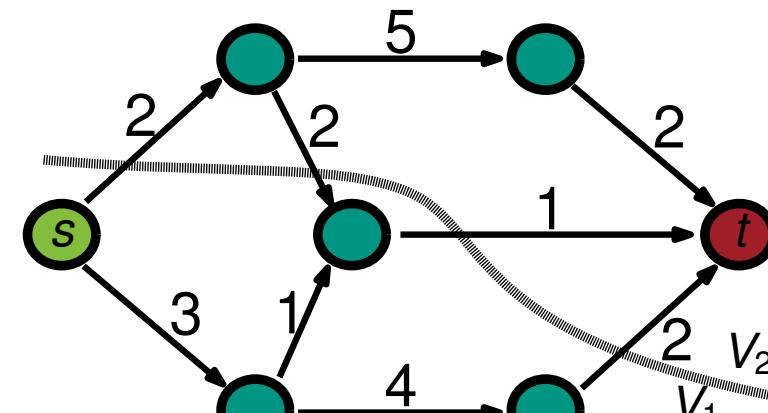
Select two adjacent blocks for refinement



Build Flow Problem

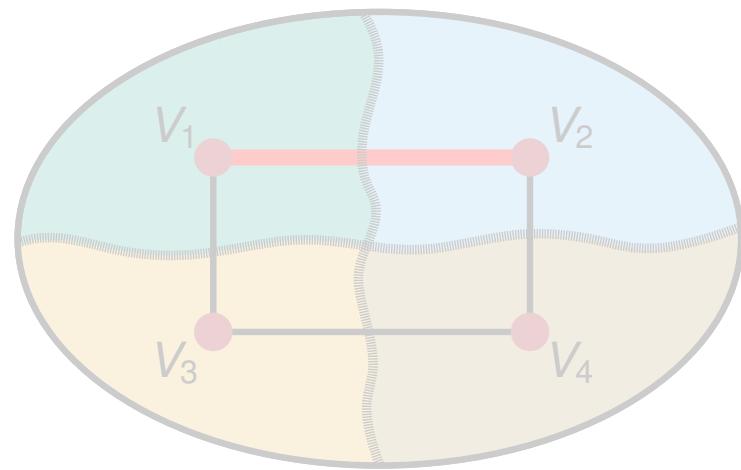


Solve Flow Problem

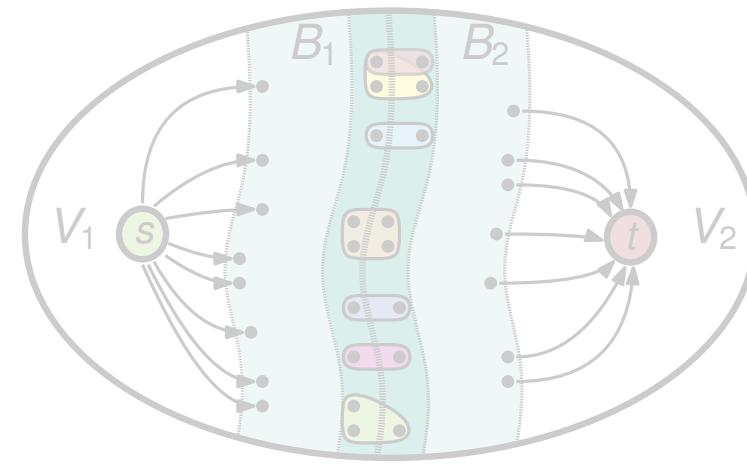


Find feasible minimum cut

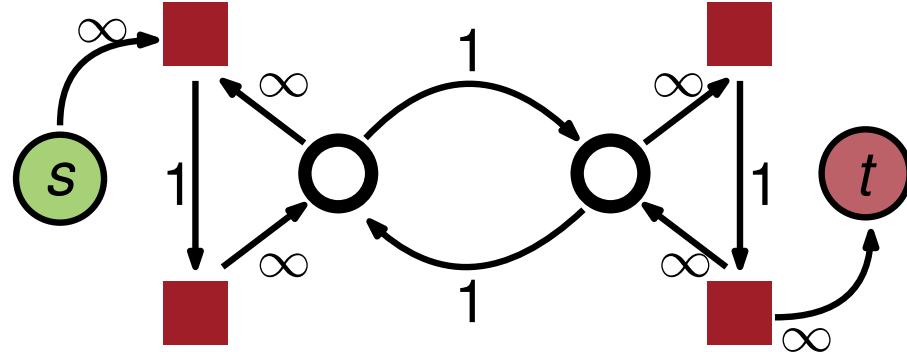
# Our Flow-Based Refinement Framework



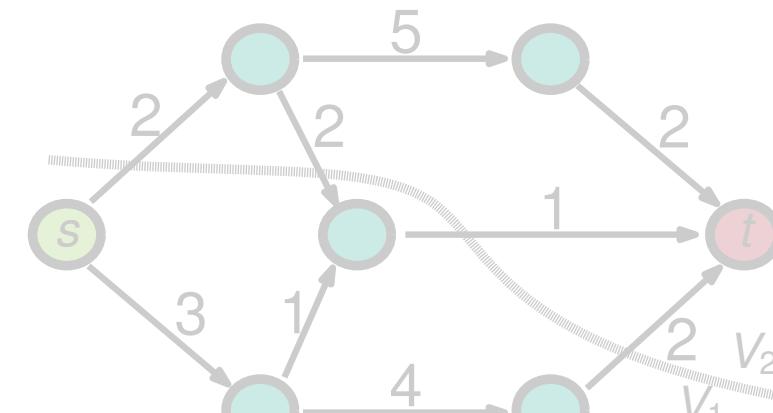
Select two adjacent blocks for refinement



Build Flow Problem

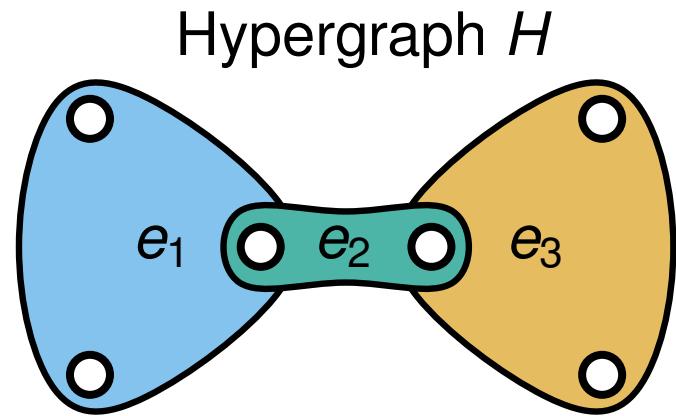


Solve Flow Problem

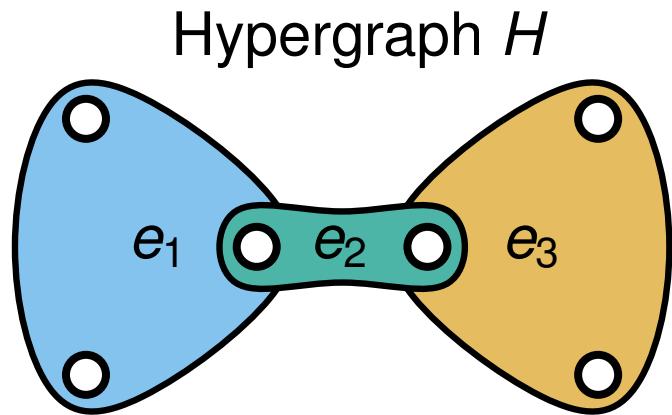


Find feasible minimum cut

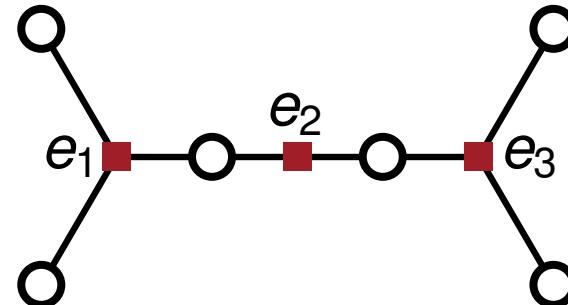
# Hypergraph Flow Network



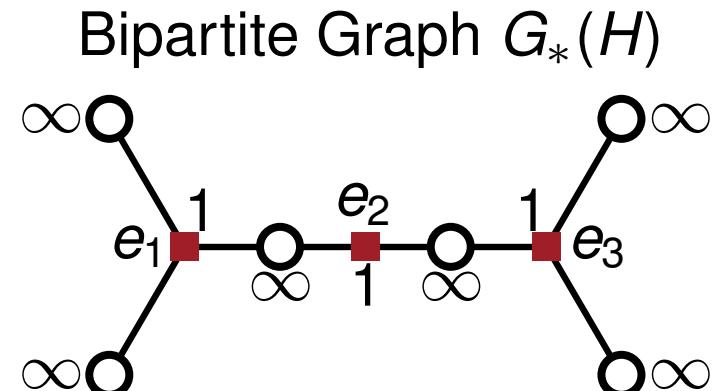
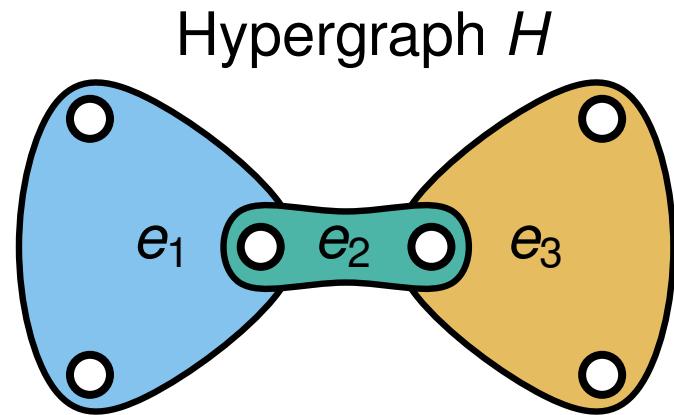
# Hypergraph Flow Network



Bipartite Graph  $G_*(H)$

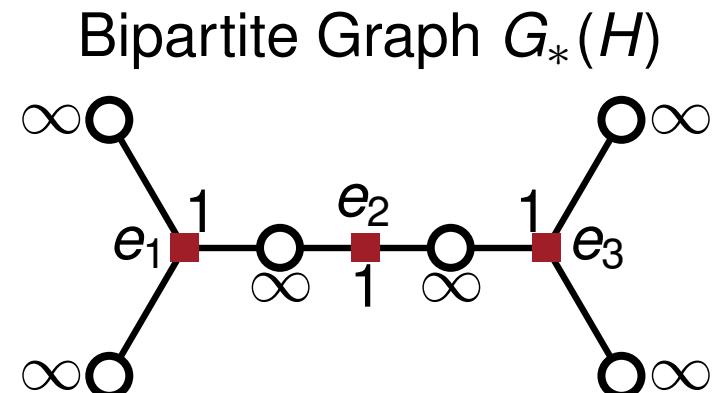
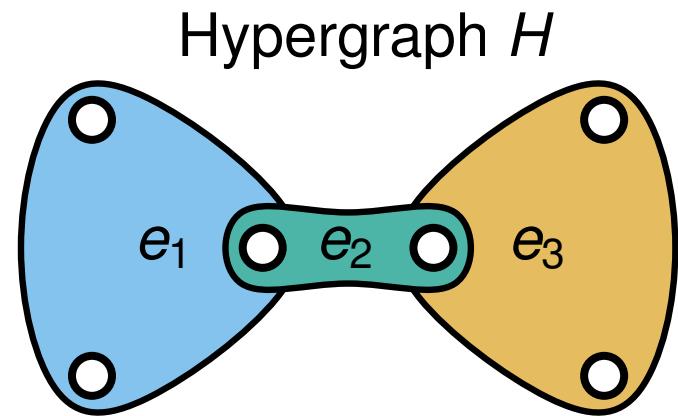


# Hypergraph Flow Network



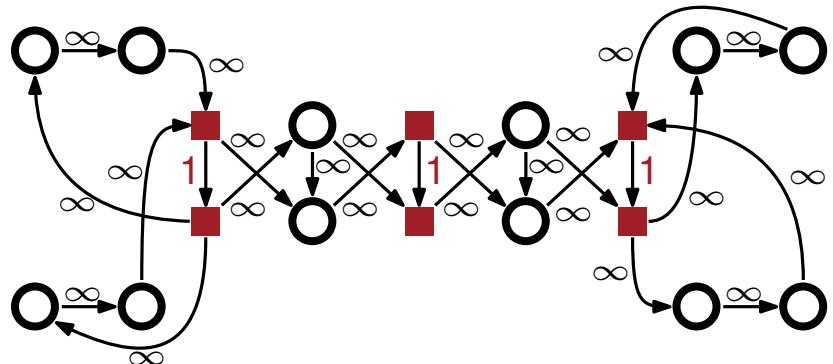
Vertex Separator Problem [Hu, Moerder 85]

# Hypergraph Flow Network

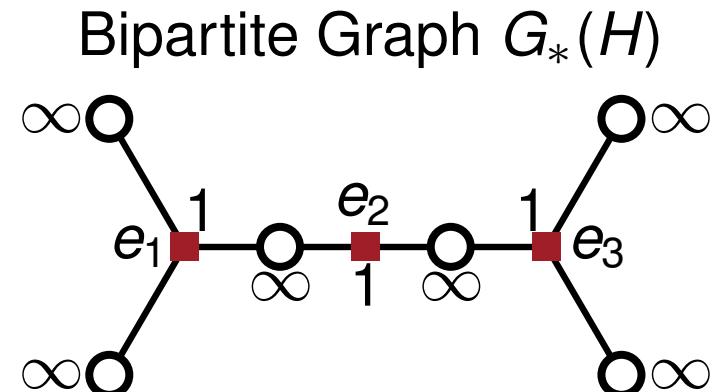
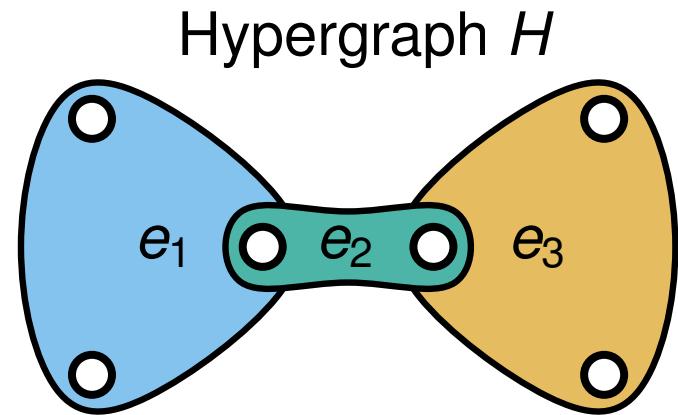


Vertex Separator Problem [Hu, Moerder 85]

## Vertex Separator Transformation

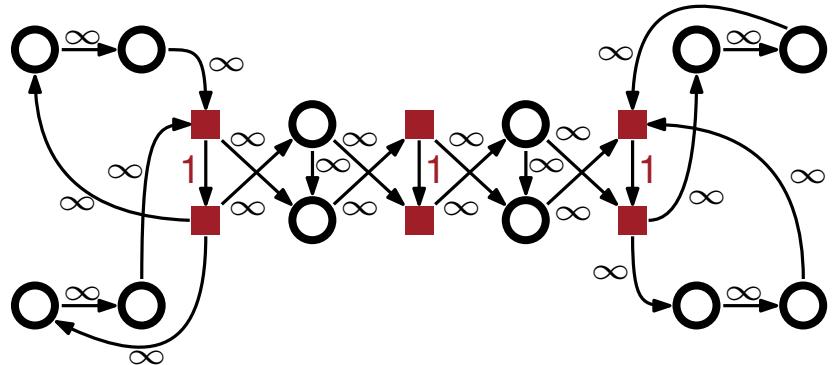


# Hypergraph Flow Network

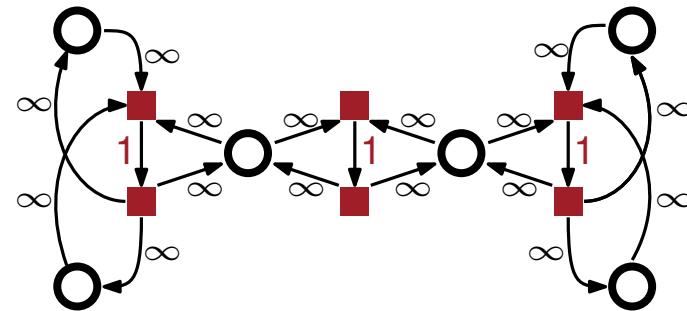


Vertex Separator Problem [Hu, Moerder 85]

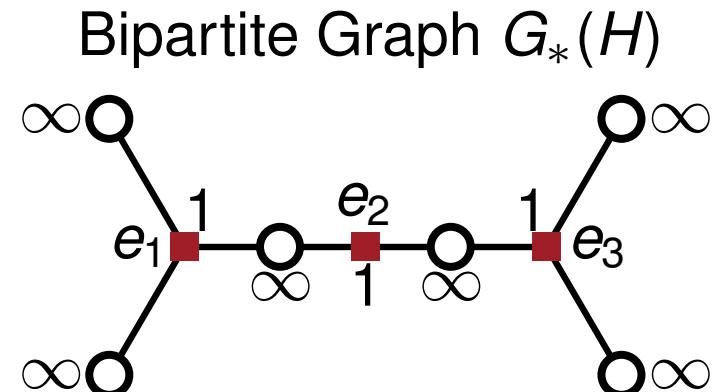
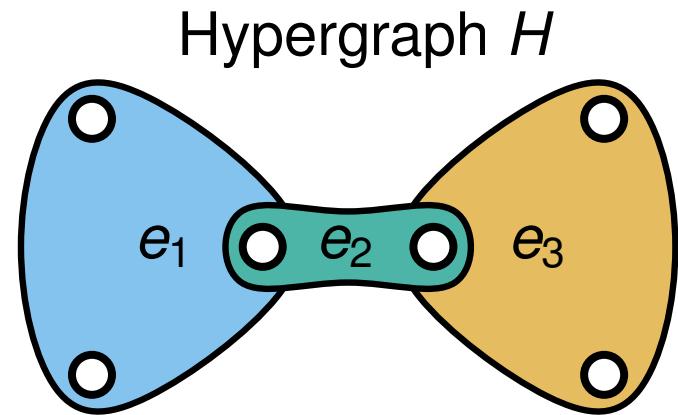
## Vertex Separator Transformation



## Lawler Network [Lawler 73]

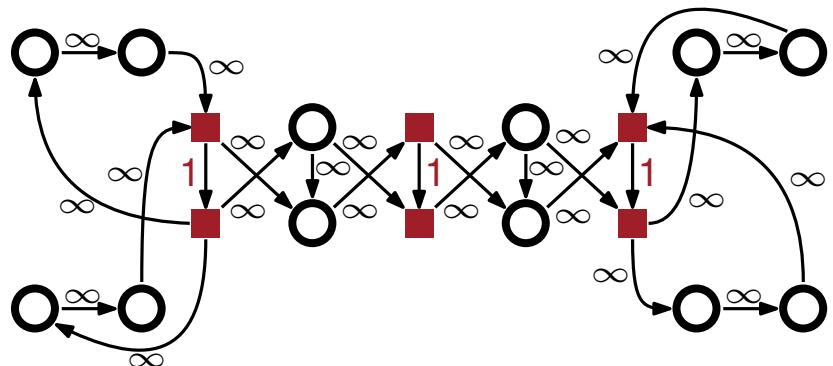


# Hypergraph Flow Network

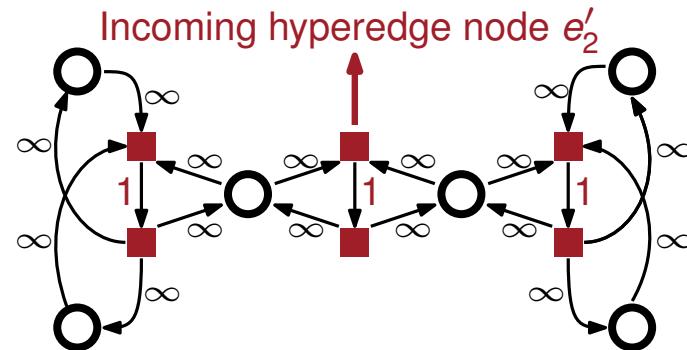


Vertex Separator Problem [Hu, Moerder 85]

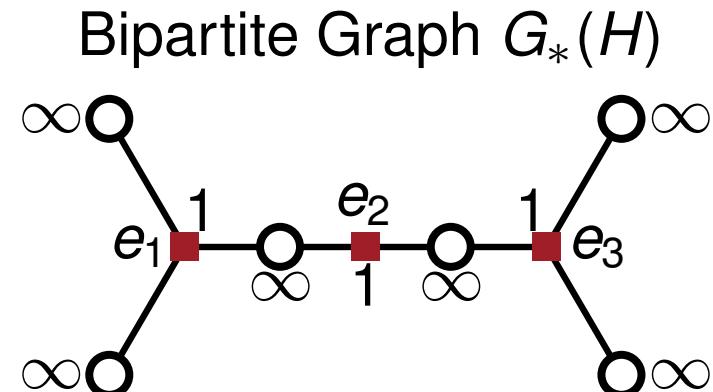
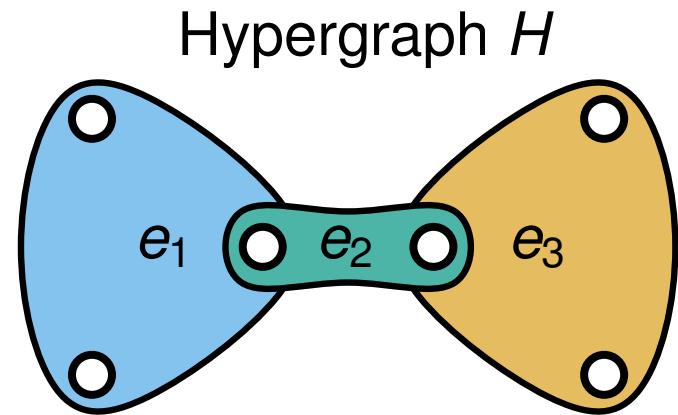
## Vertex Separator Transformation



## Lawler Network [Lawler 73]

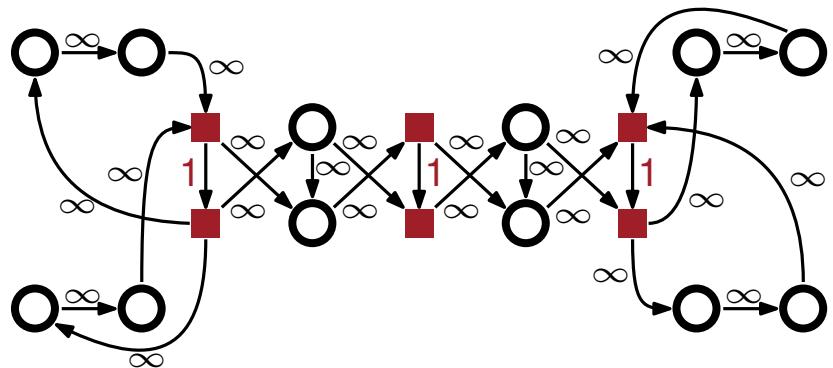


# Hypergraph Flow Network

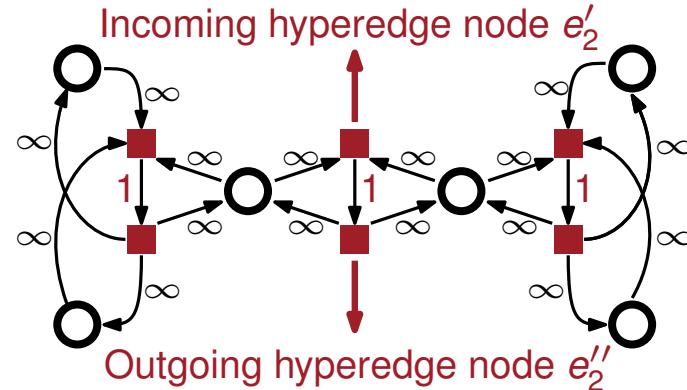


Vertex Separator Problem [Hu, Moerder 85]

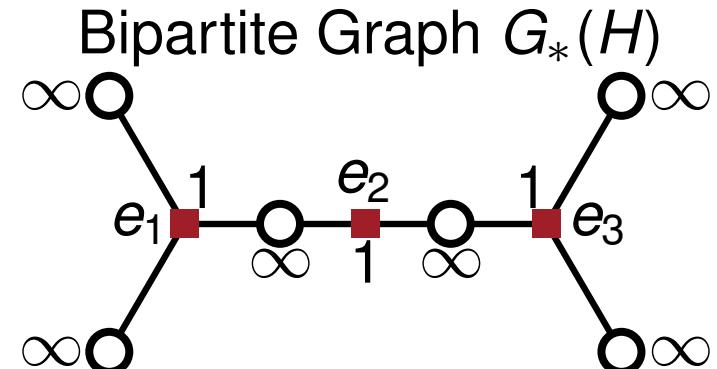
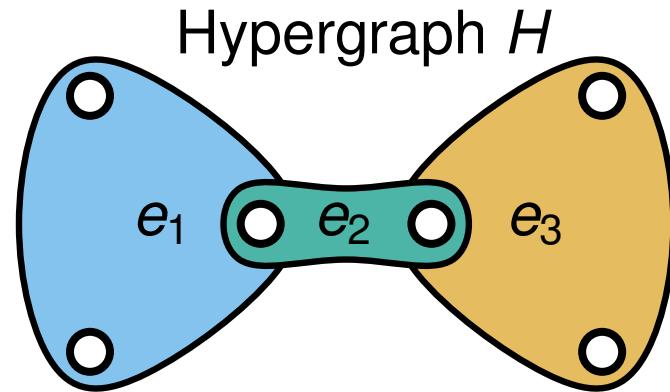
## Vertex Separator Transformation



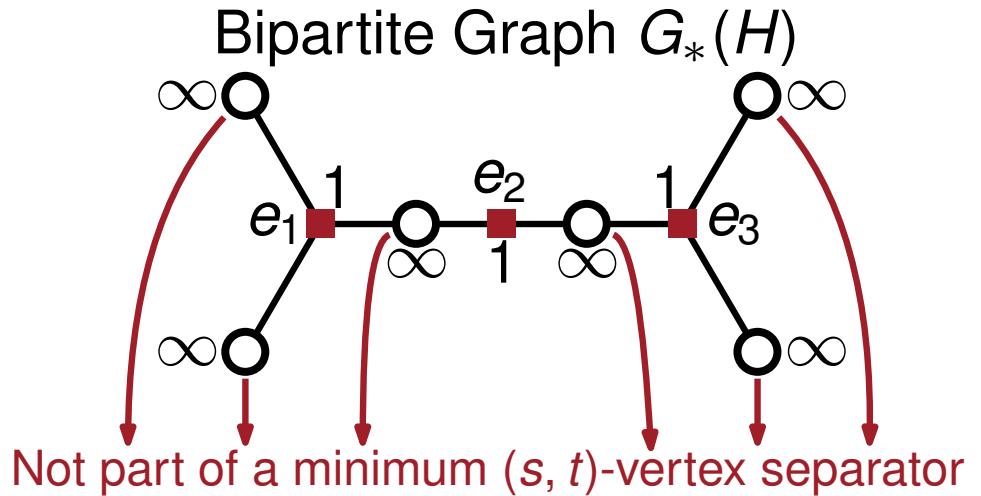
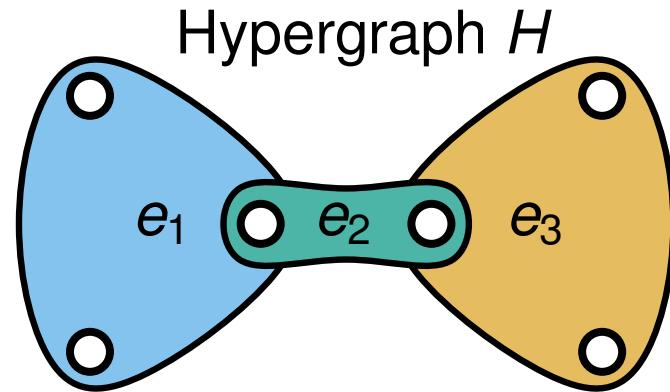
## Lawler Network [Lawler 73]



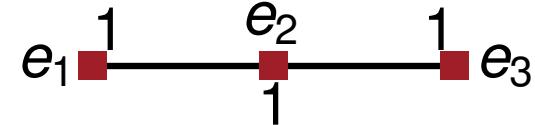
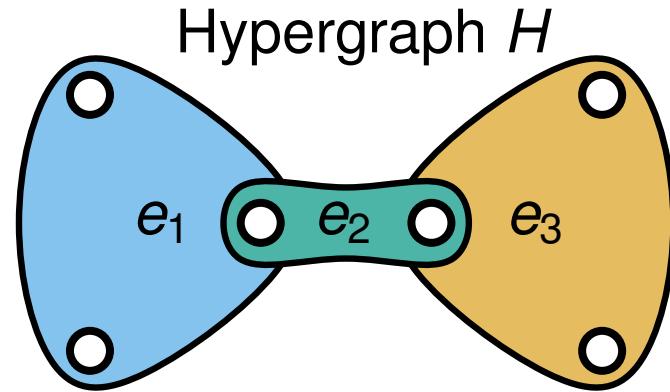
# Hypergraph Flow Network - Low Degree Vertices



# Hypergraph Flow Network - Low Degree Vertices

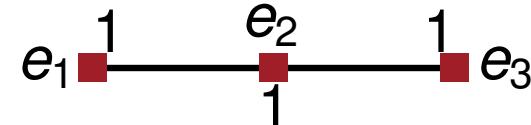
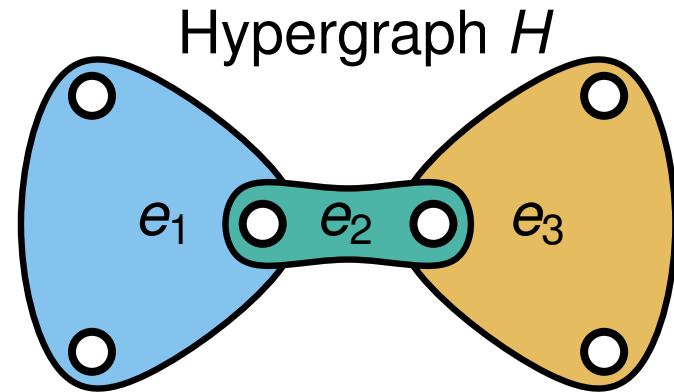


# Hypergraph Flow Network - Low Degree Vertices



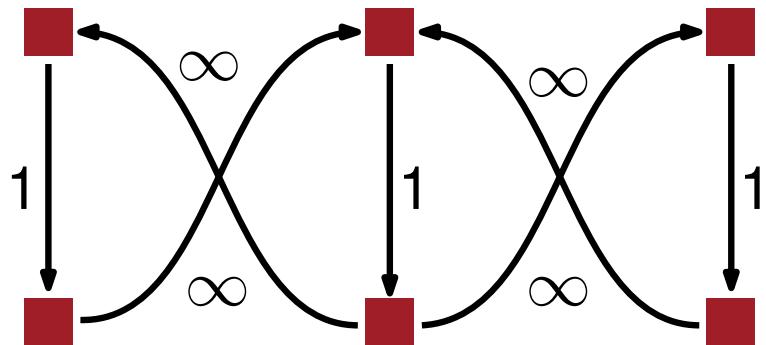
Remove all vertices by adding a clique

# Hypergraph Flow Network - Low Degree Vertices

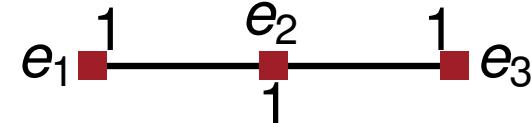
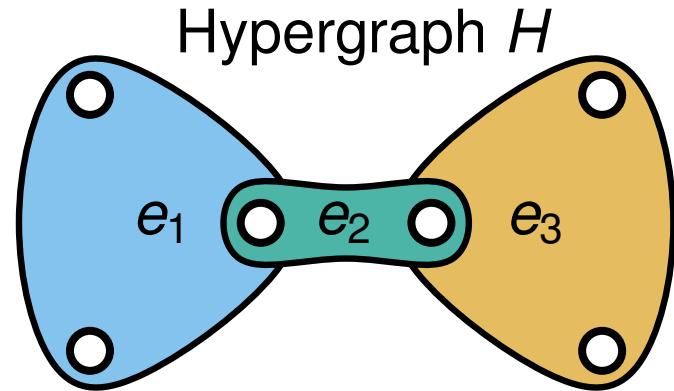


Remove all vertices by adding a clique

Our Network

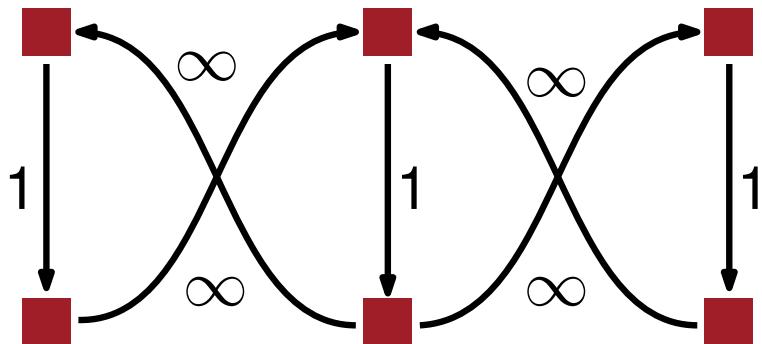


# Hypergraph Flow Network - Low Degree Vertices



Remove all vertices by adding a clique

Our Network

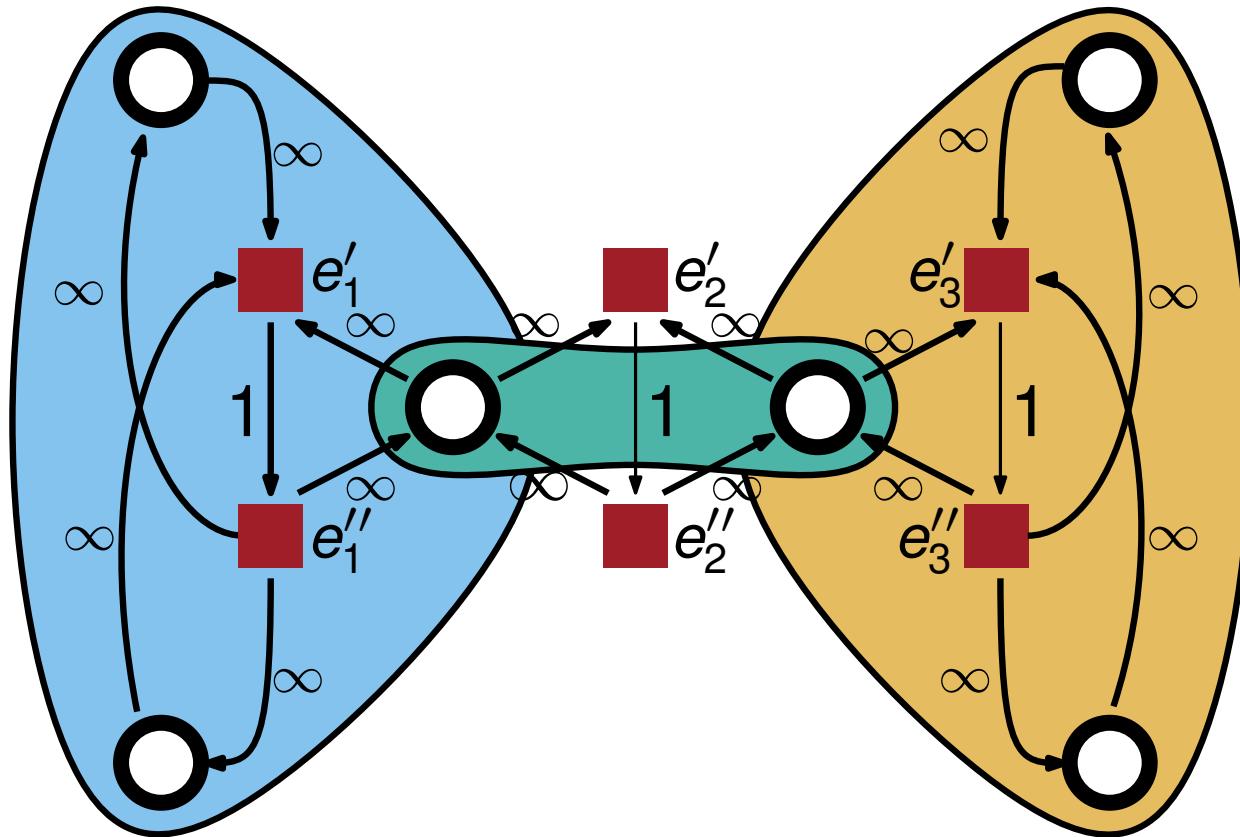


A hypernode  $v$  induces ...

- ...  $2d(v)$  edges in the Lawler Network
- ...  $d(v)(d(v) - 1)$  edges in our network

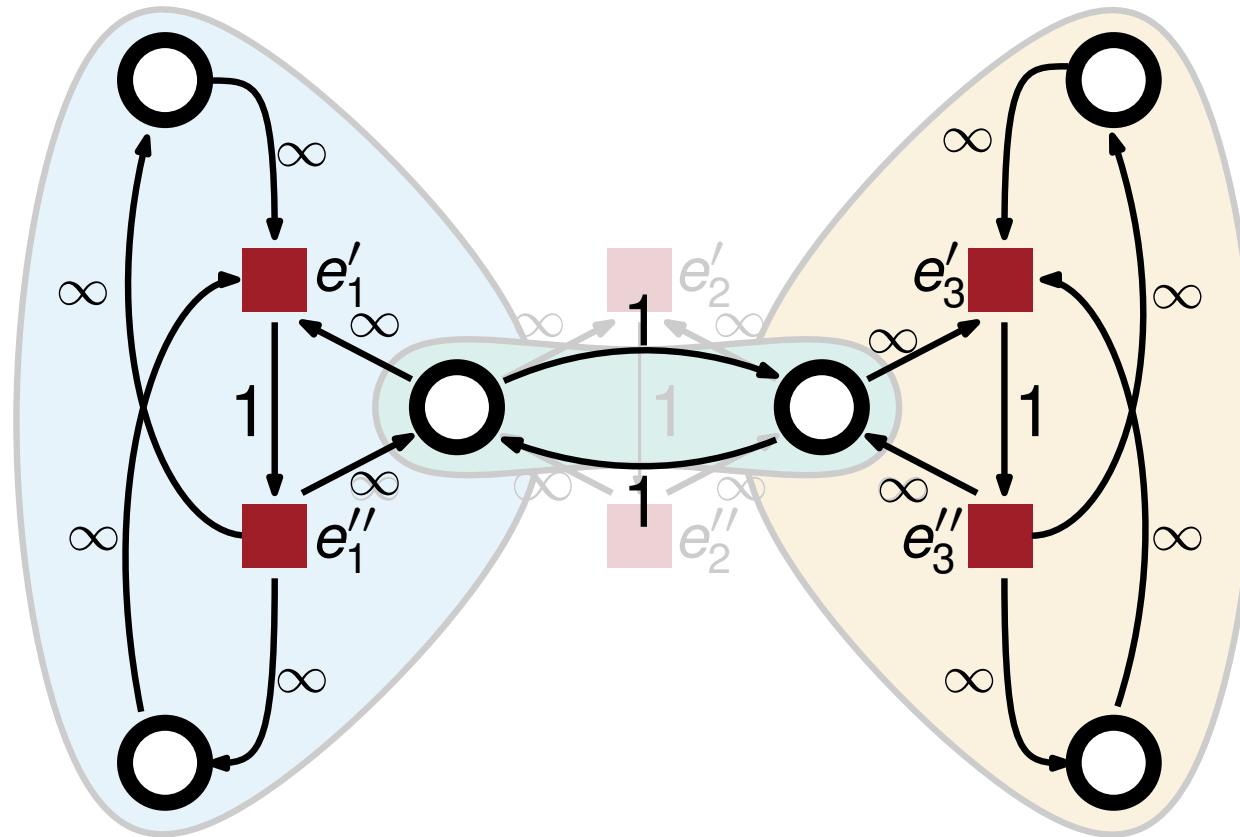
If  $d(v) \leq 3$ , then  $d(v)(d(v) - 1) \leq 2d(v)$

# Hypergraph Flow Network - Summary



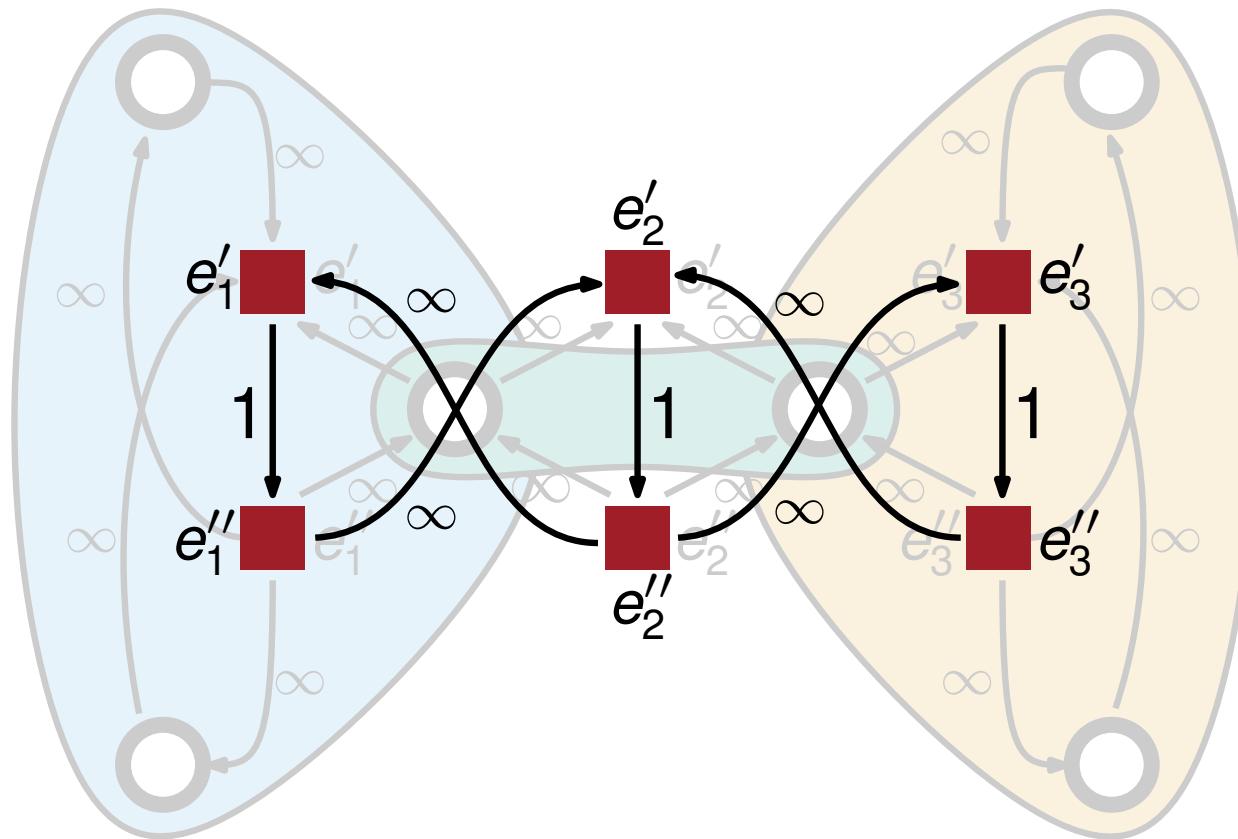
## Lawler Network

# Hypergraph Flow Network - Summary



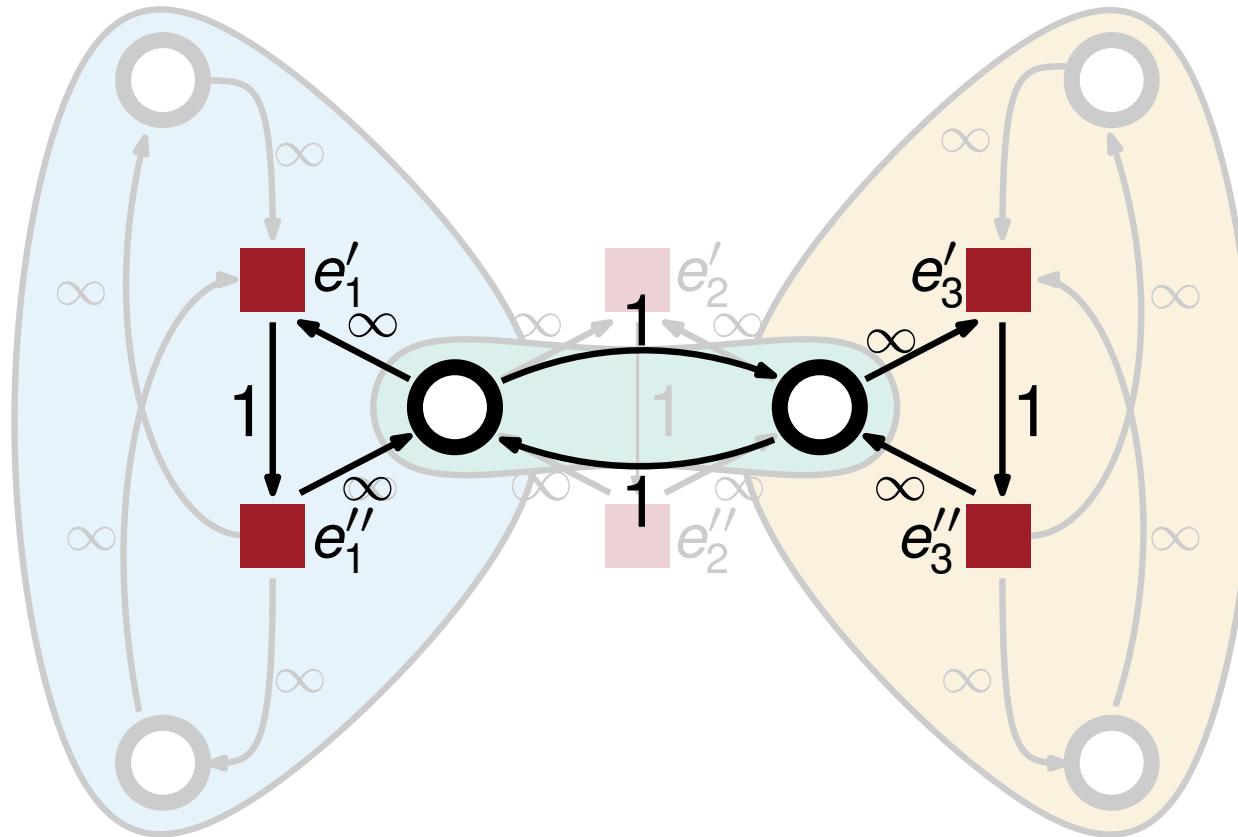
**Wong Network [Liu, Wong 98]**

# Hypergraph Flow Network - Summary



**Our Network**

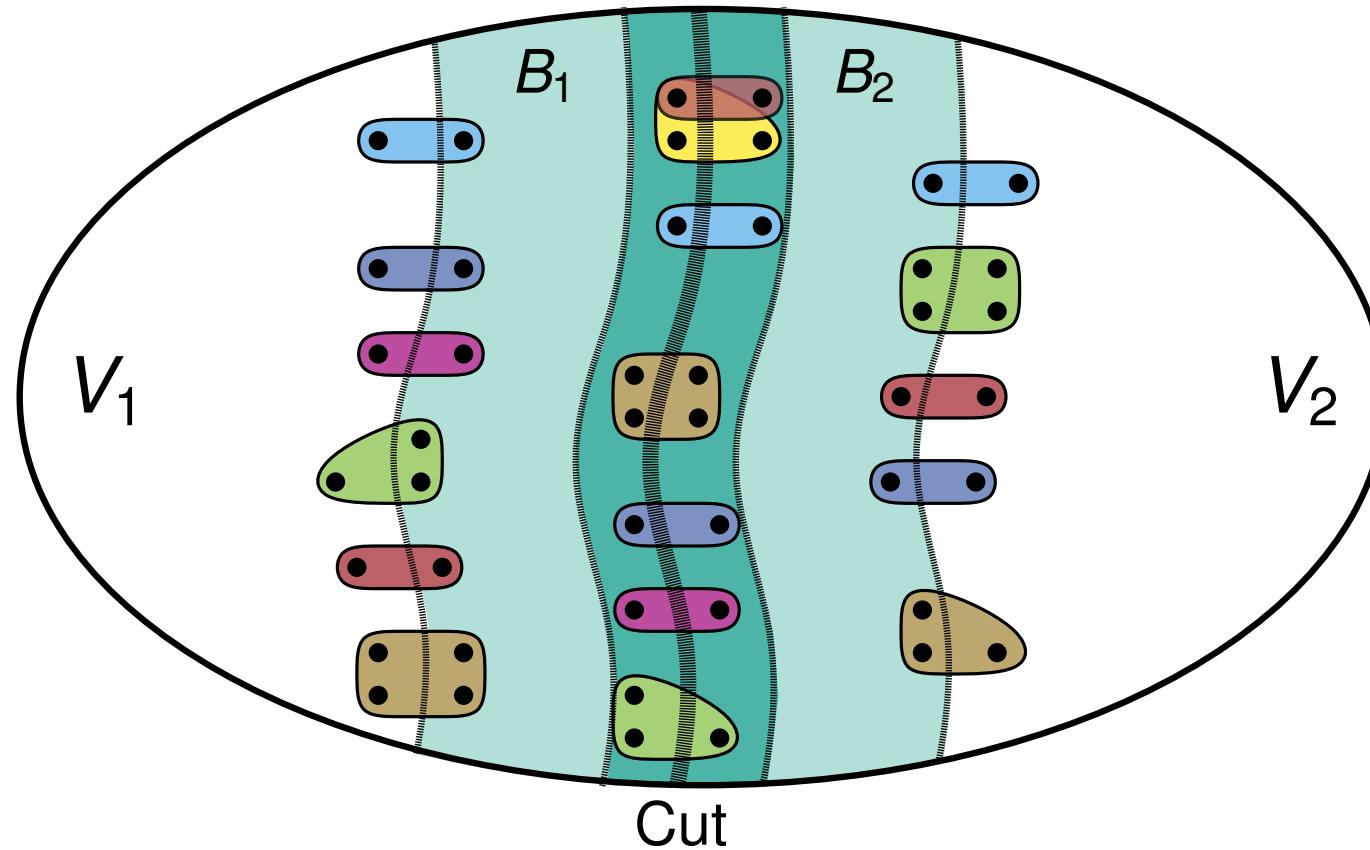
# Hypergraph Flow Network - Summary



## Hybrid Network

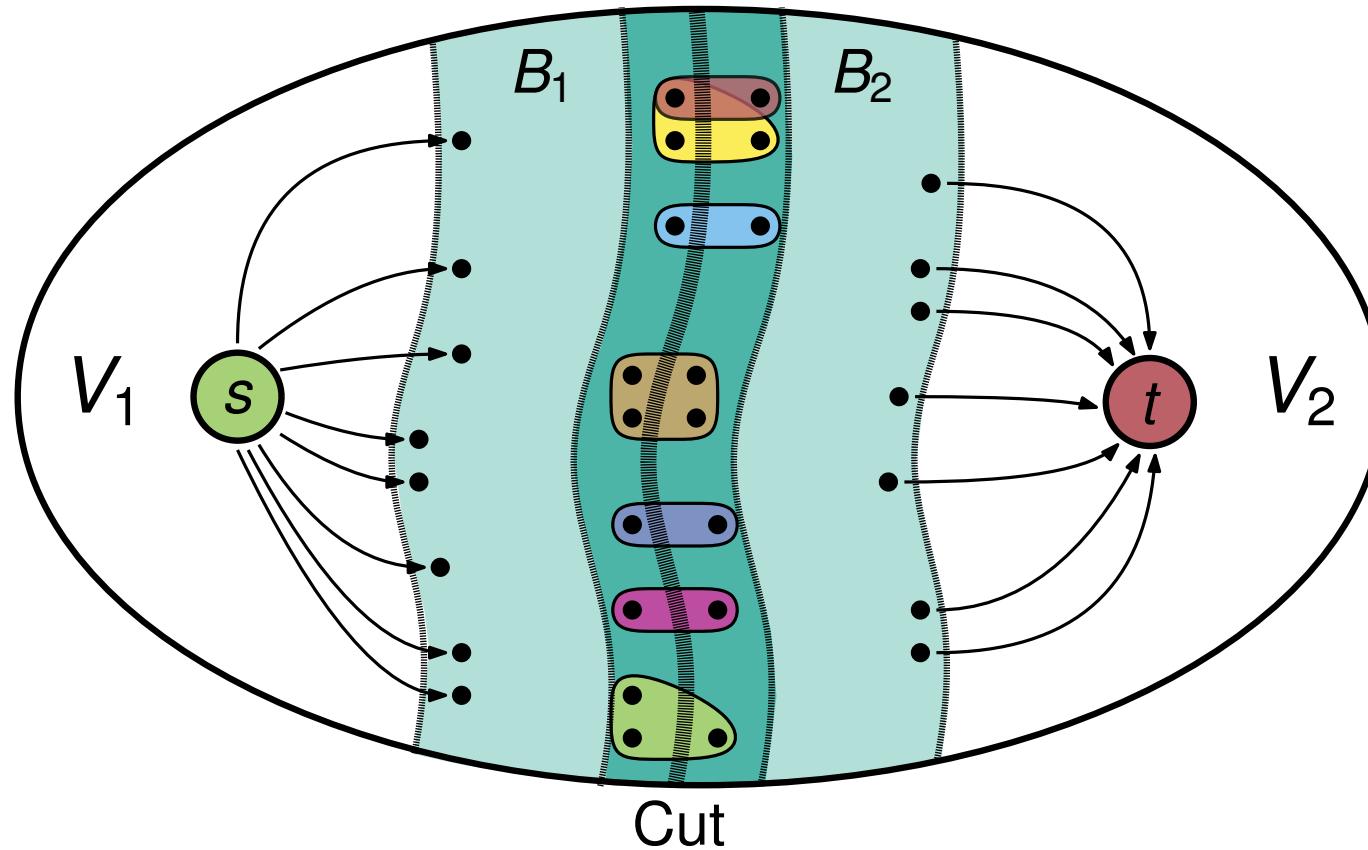
# Optimized Flow Problem Modeling Approach

Modeling Approach in *KaFFPa*



# Optimized Flow Problem Modeling Approach

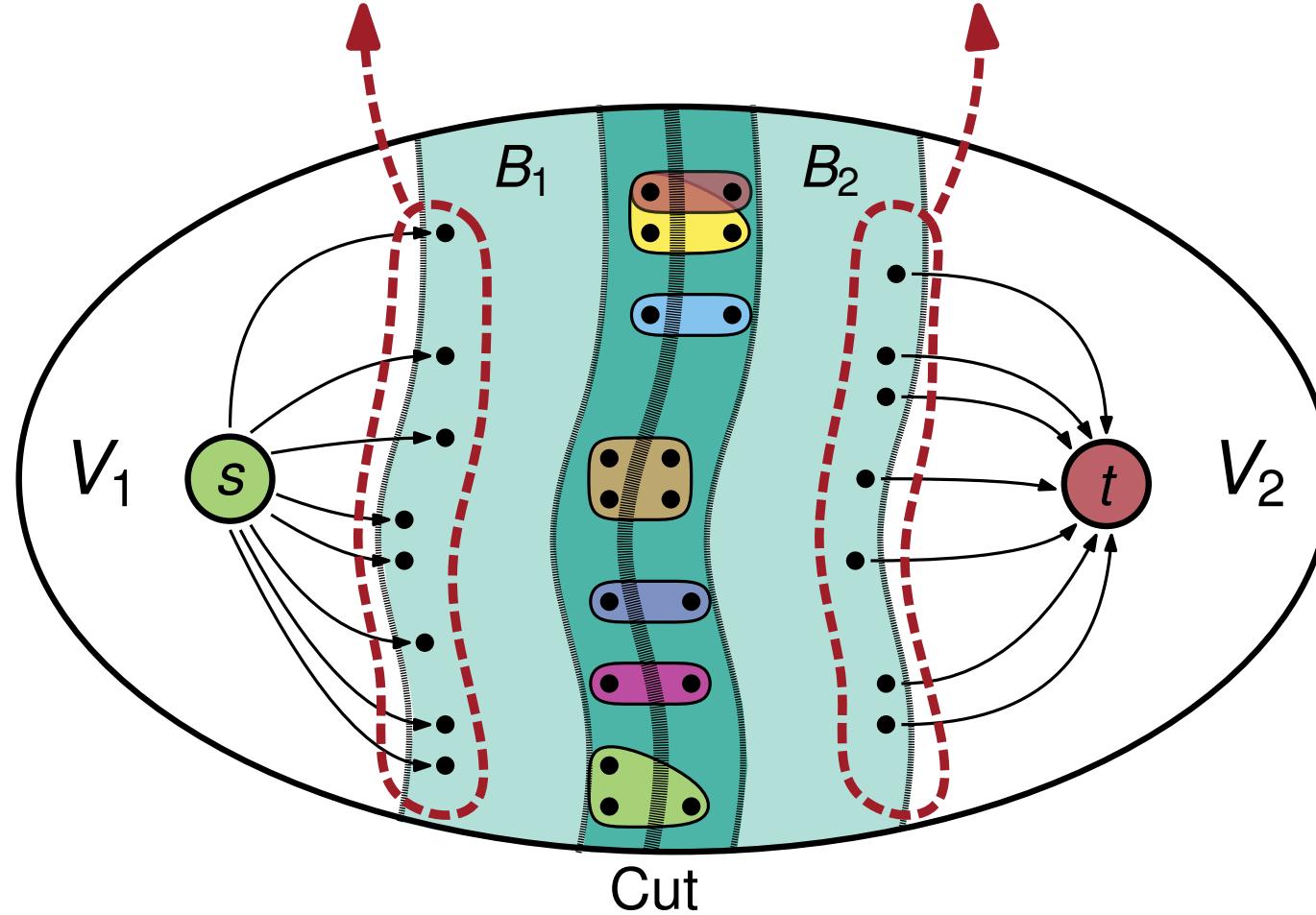
Modeling Approach in *KaFFPa*



# Optimized Flow Problem Modeling Approach

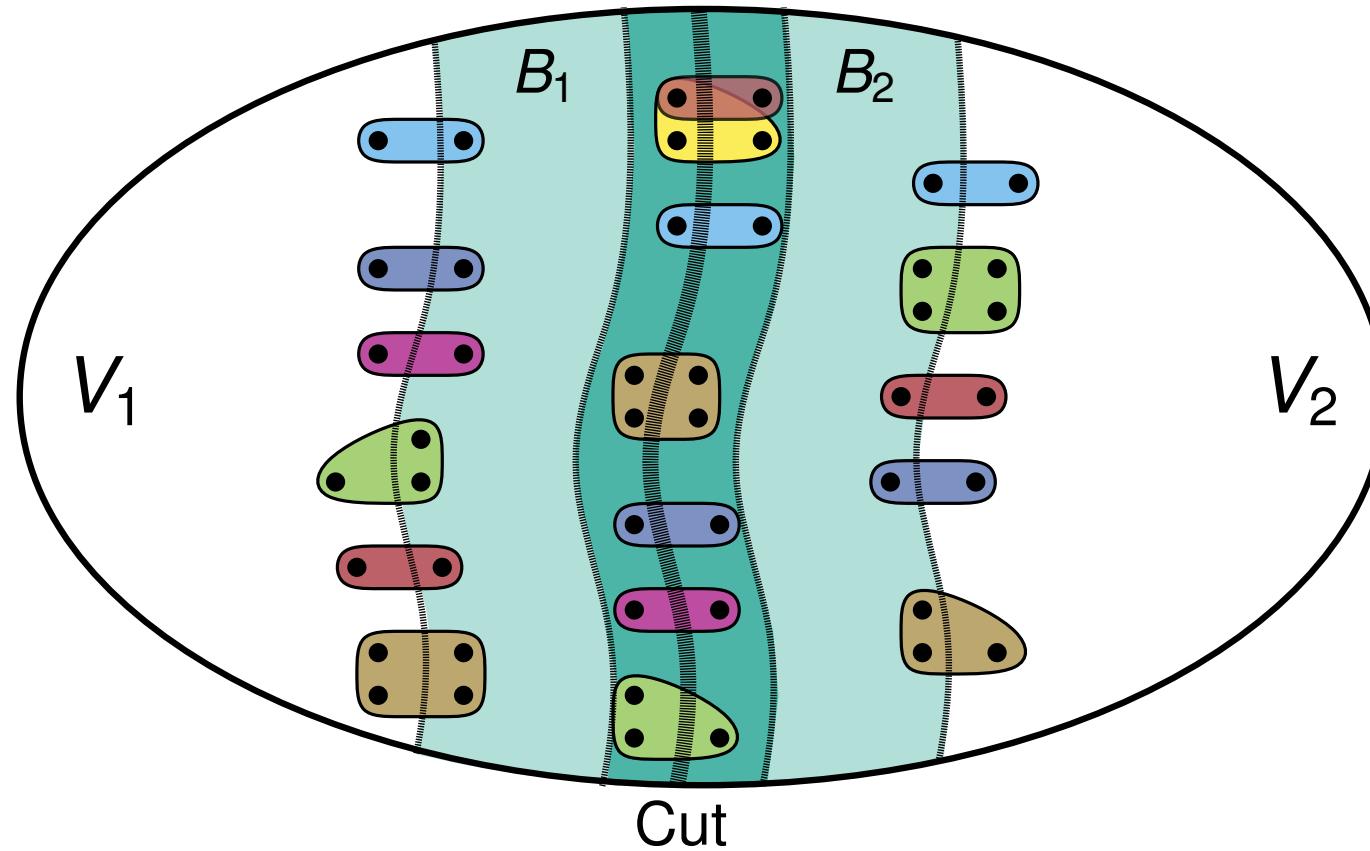
Modeling Approach in *KaFFPa*

**Not moveable after Max-Flow-Min-Cut computation**



# Optimized Flow Problem Modeling Approach

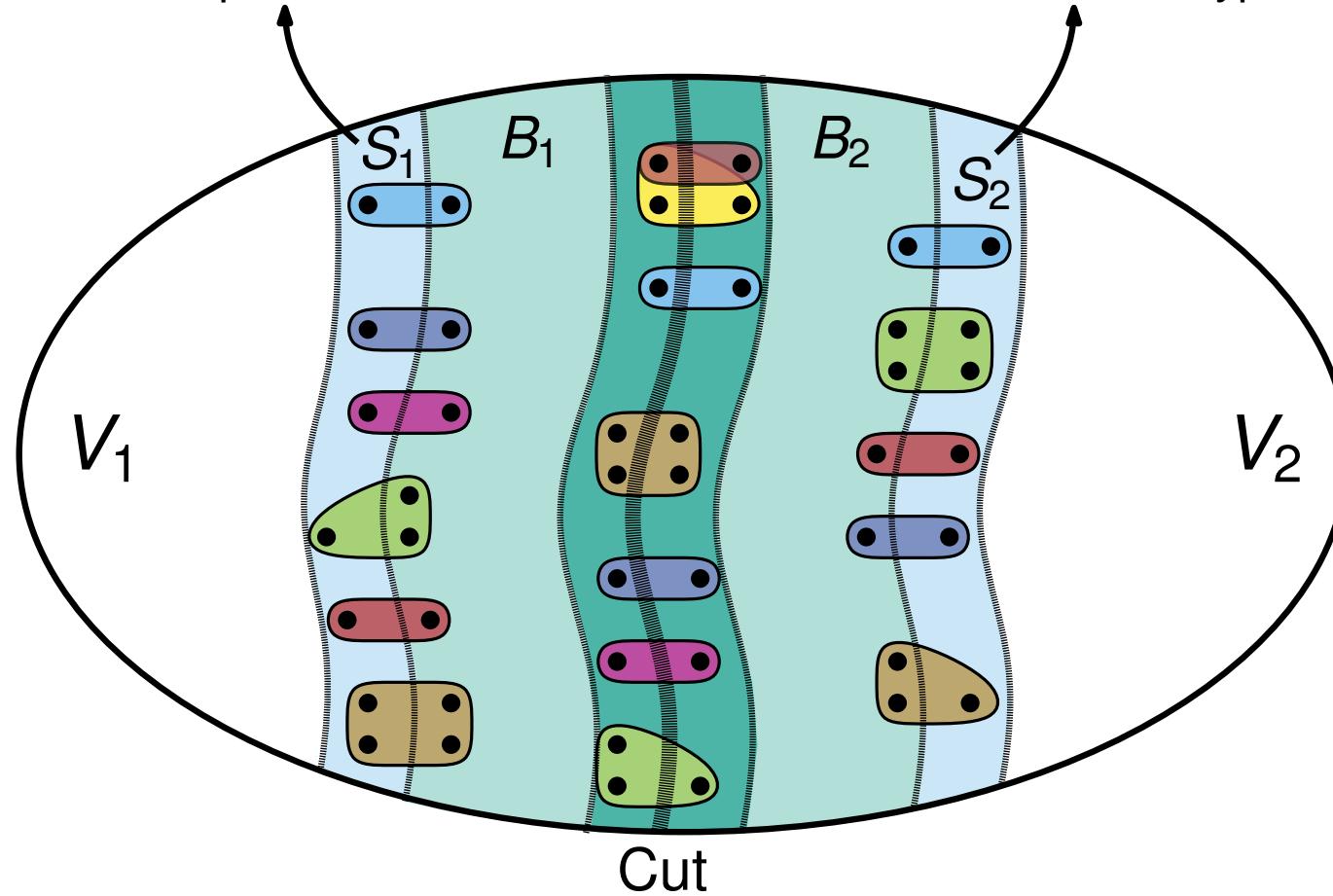
Modeling Approach in *KaHyPar*



# Optimized Flow Problem Modeling Approach

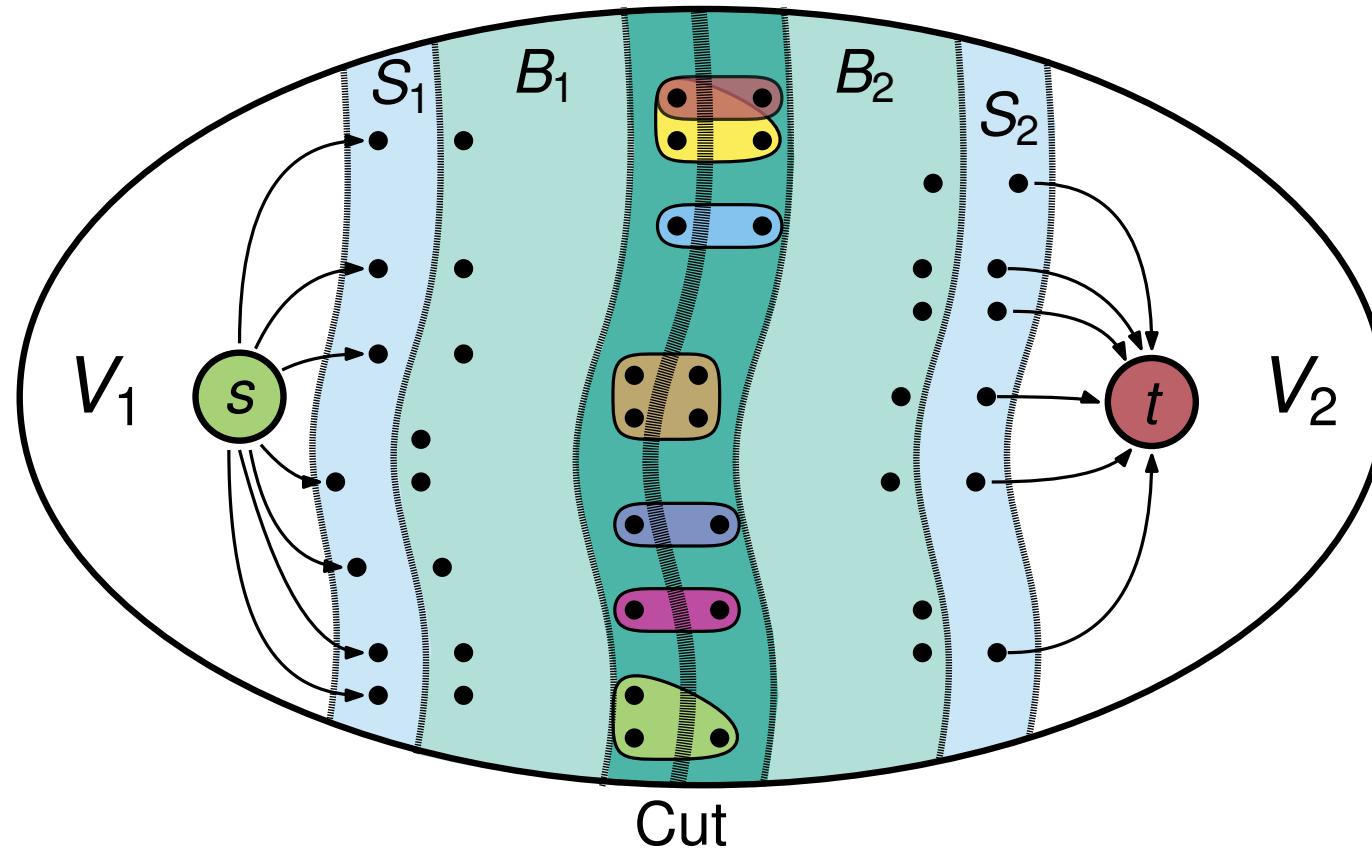
Modeling Approach in *KaHyPar*

Extend flow problem with all vertices contained in a border hyperedge



# Optimized Flow Problem Modeling Approach

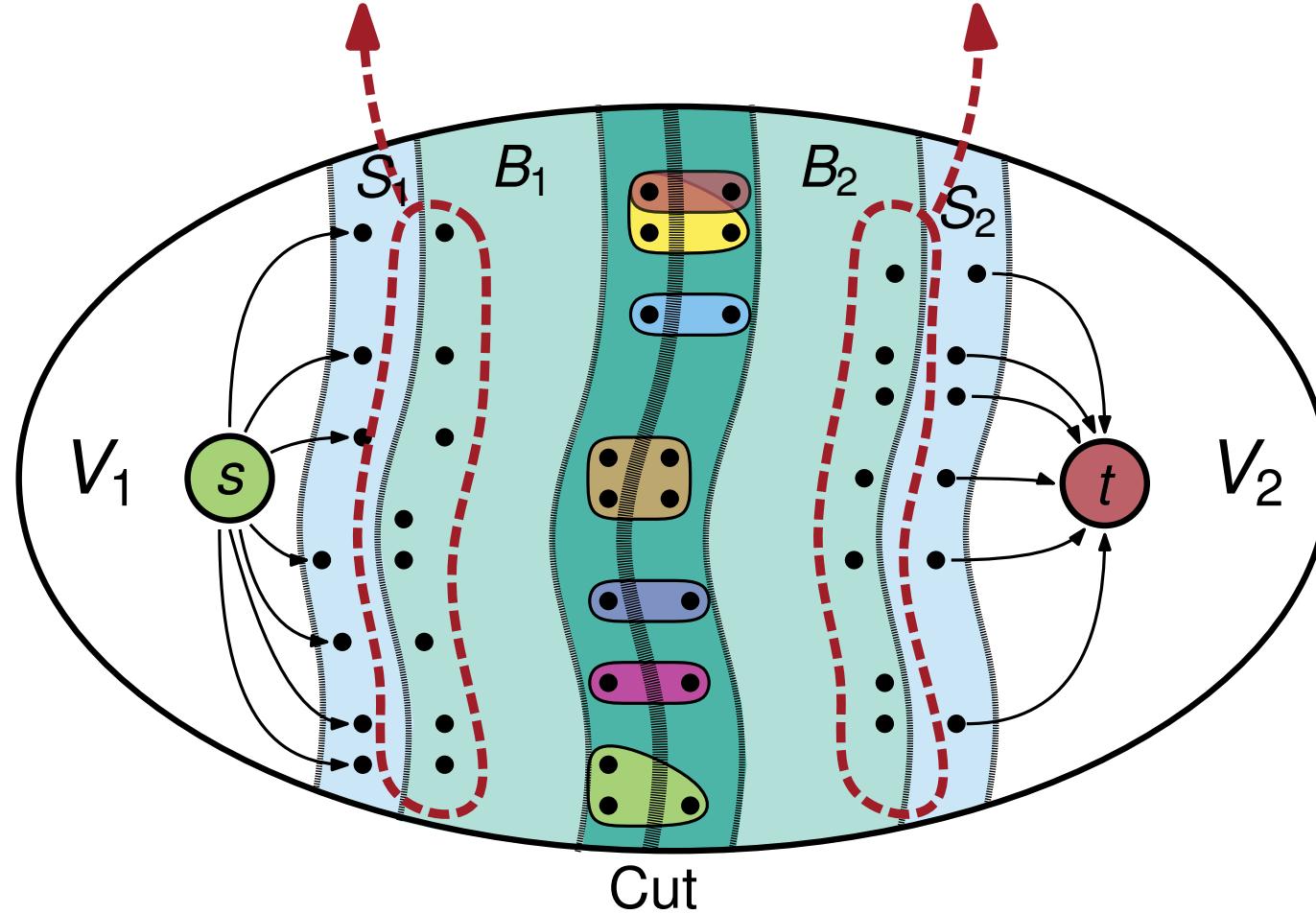
Modeling Approach in *KaHyPar*



# Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*

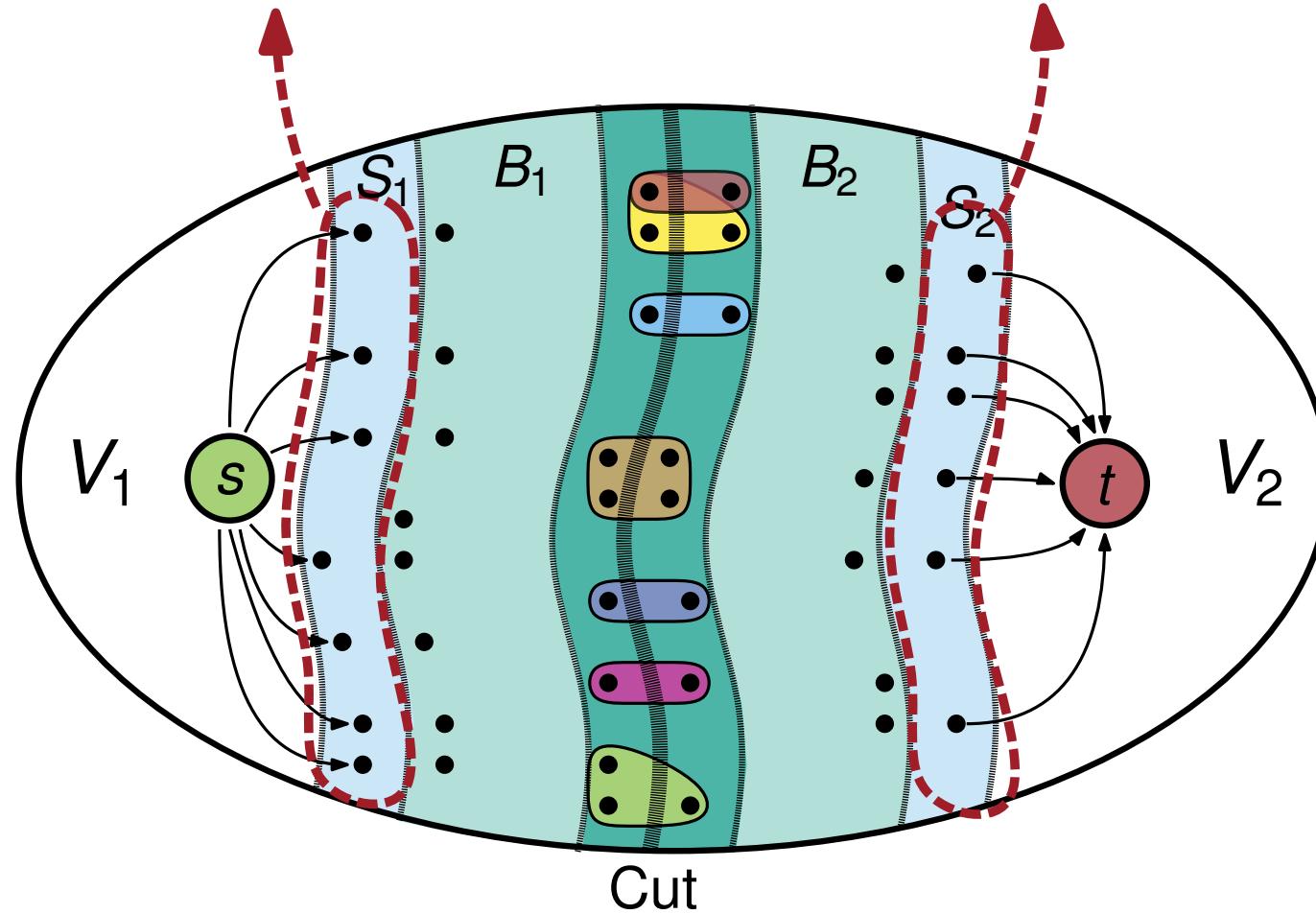
**Moveable** after Max-Flow-Min-Cut computation, but . . .



# Optimized Flow Problem Modeling Approach

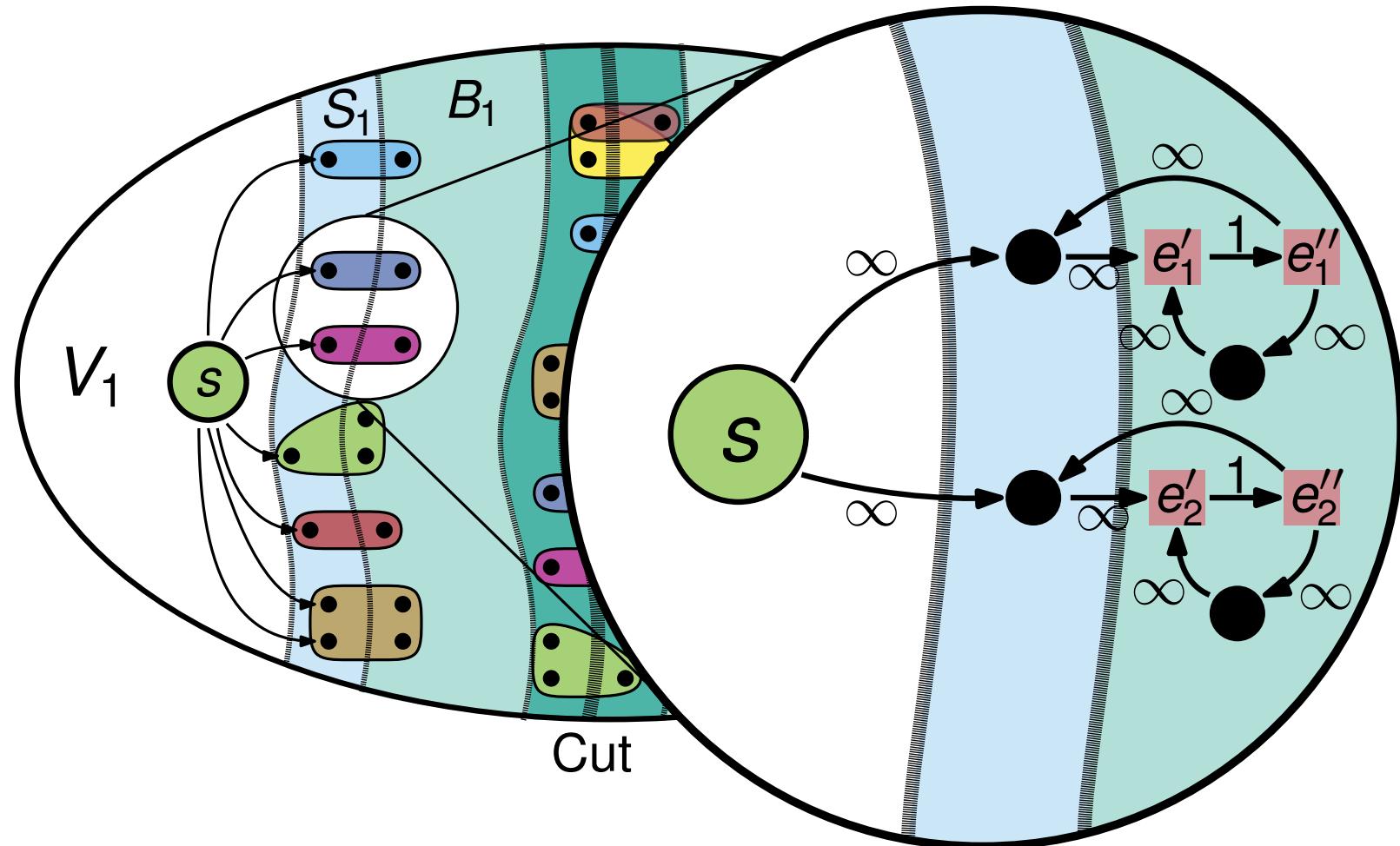
Modeling Approach in *KaHyPar*

... flow problem has significantly more nodes and edges.



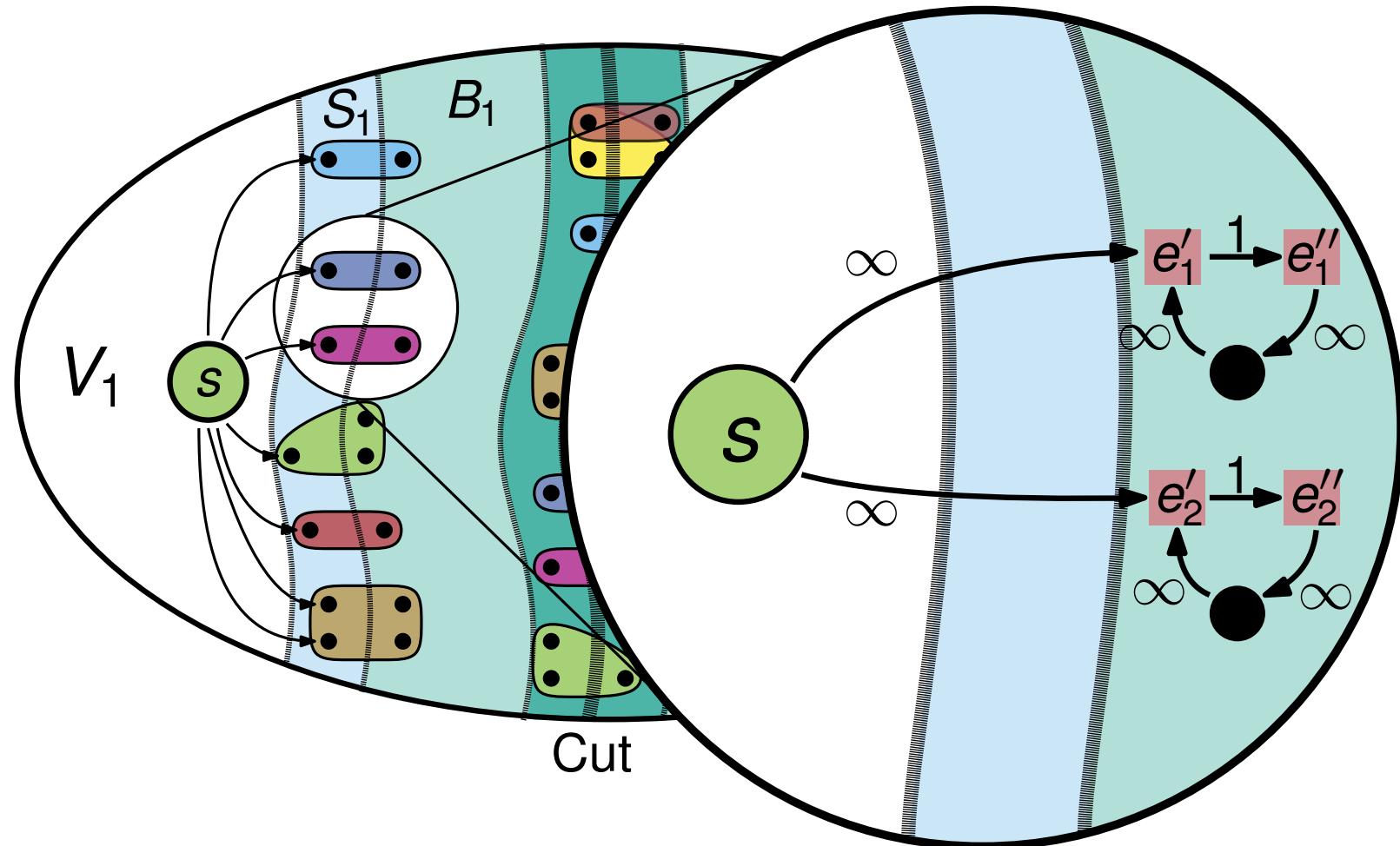
# Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*



# Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*

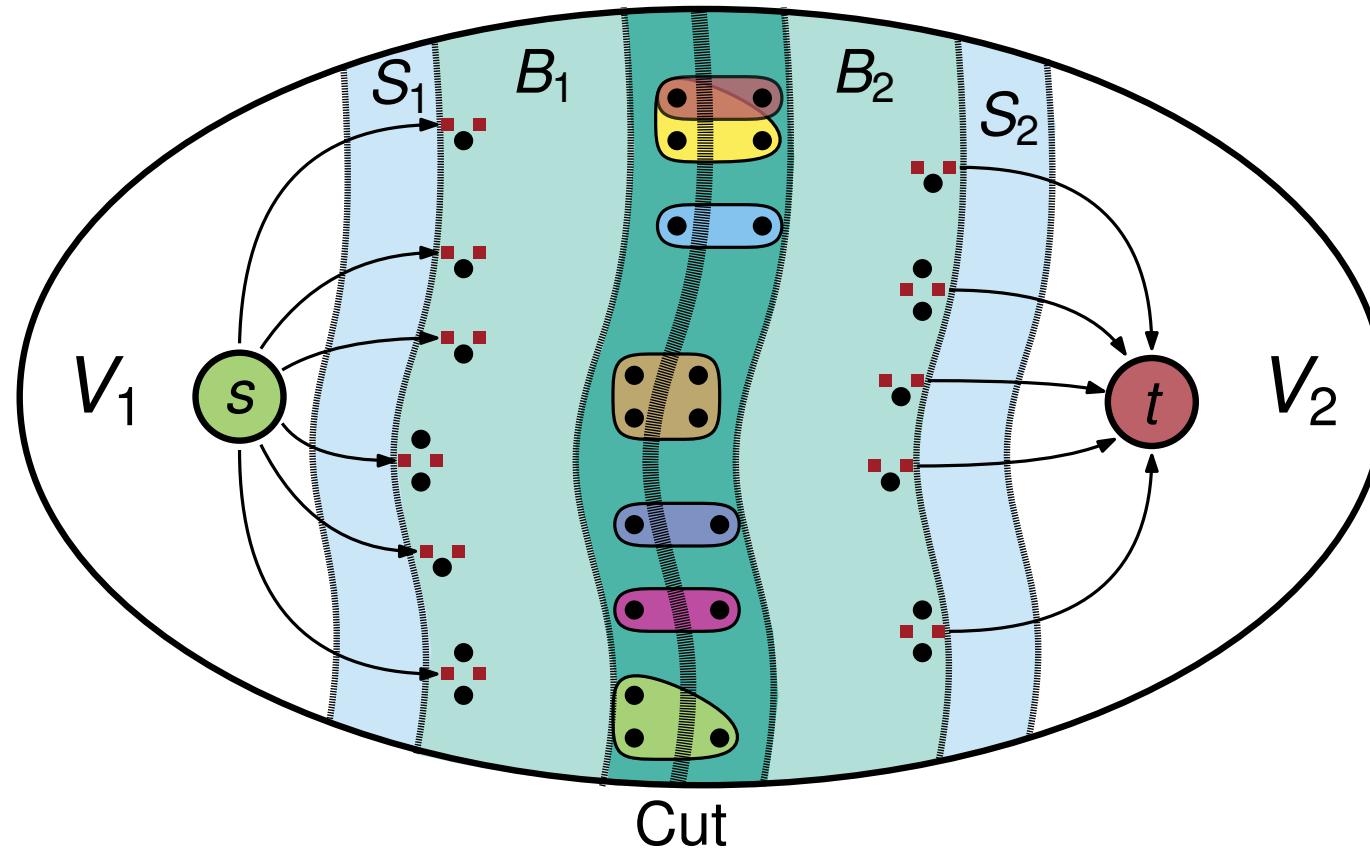


# Optimized Flow Problem Modeling Approach

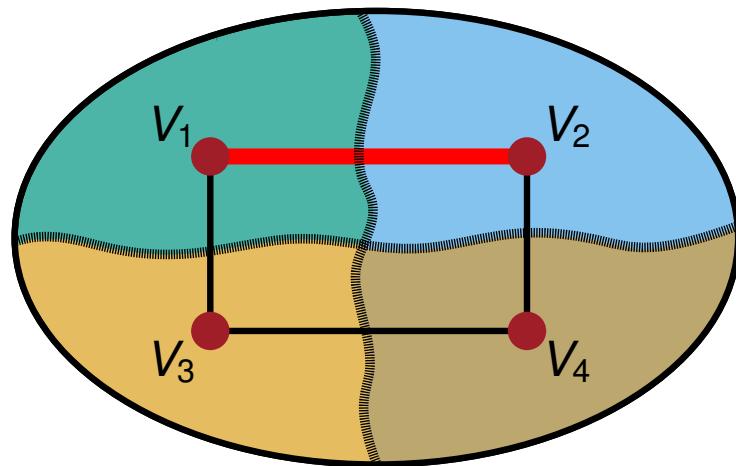
Modeling Approach in *KaHyPar*

$$S = \{e' \mid e \in I(S_1)\}$$

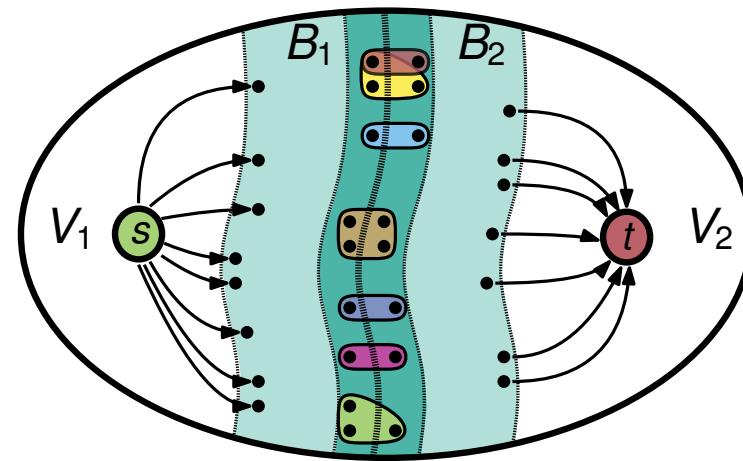
$$T = \{e'' \mid e \in I(S_2)\}$$



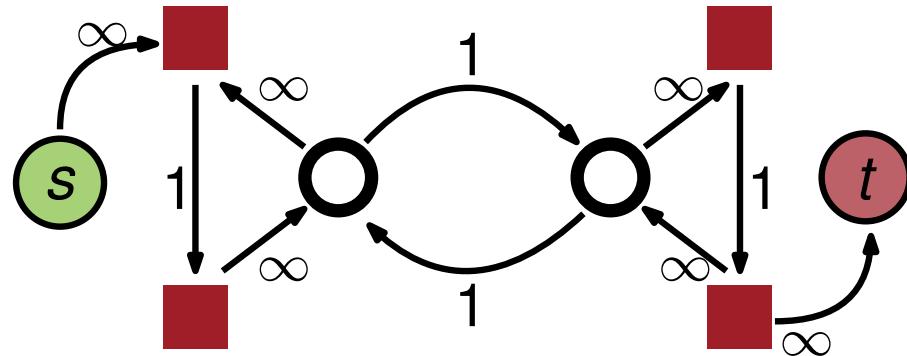
# Our Flow-Based Refinement Framework



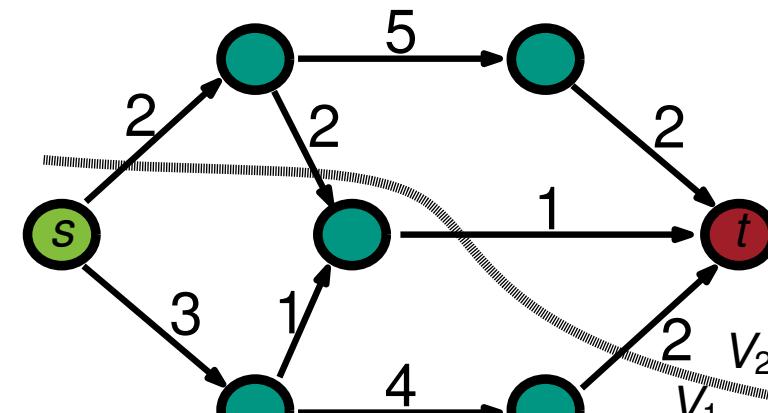
Select two adjacent blocks for refinement



Build Flow Problem

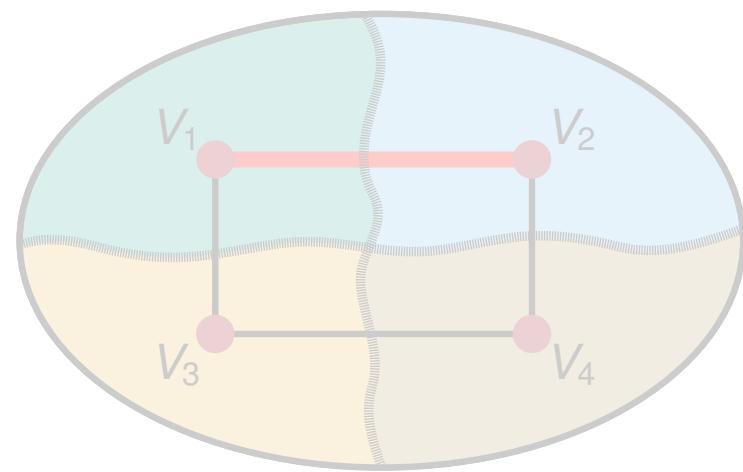


Solve Flow Problem

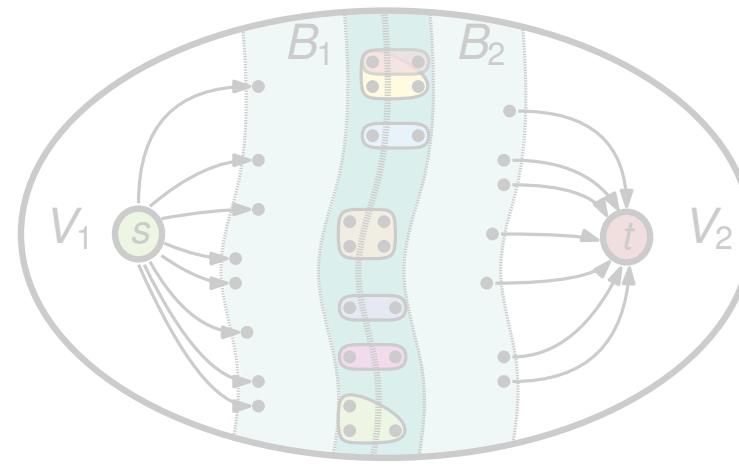


Find feasible minimum cut

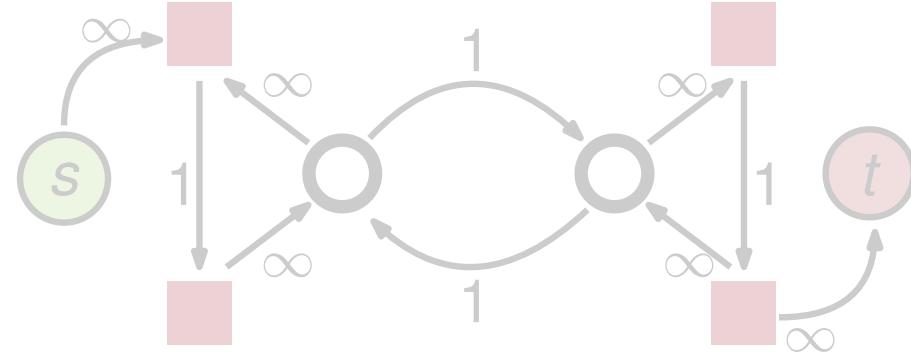
# Our Flow-Based Refinement Framework



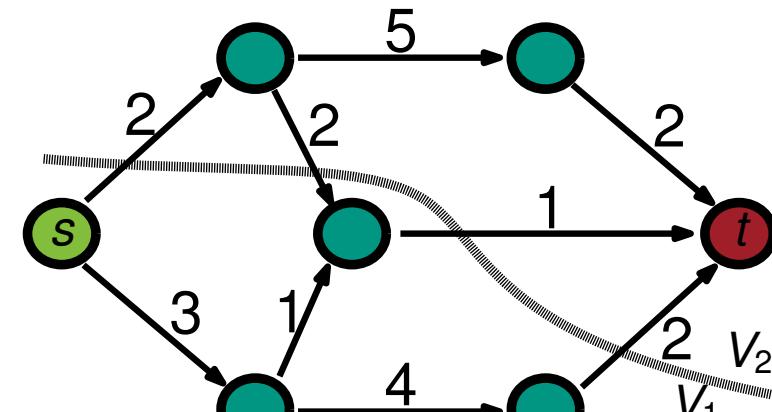
Select two adjacent blocks for refinement



Build Flow Problem



Solve Flow Problem



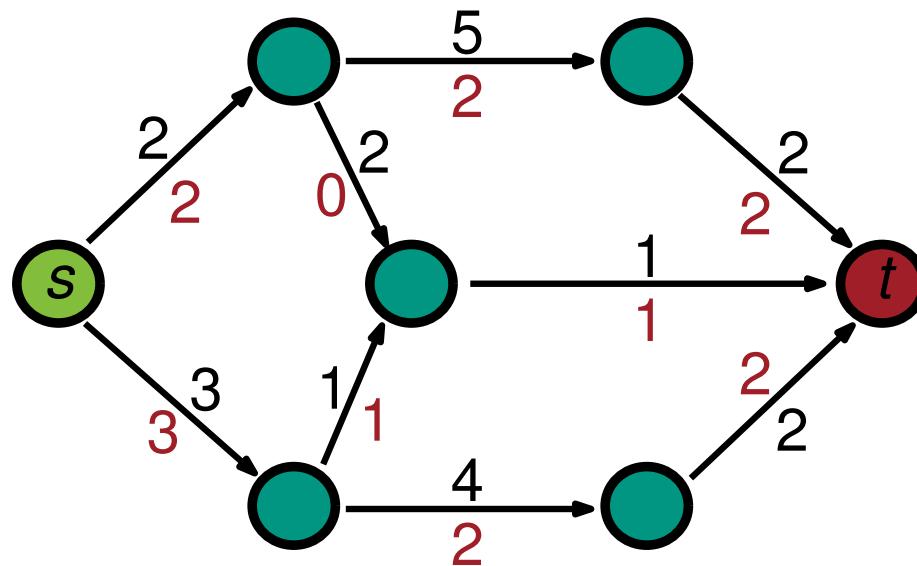
Find feasible minimum cut

# Most Balanced Minimum Cut

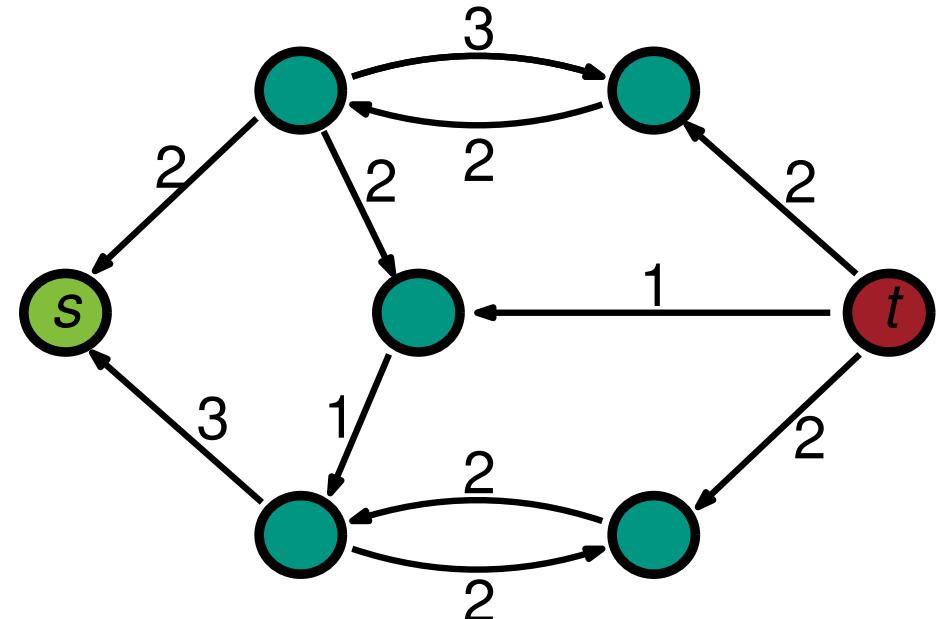
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Residual Graph

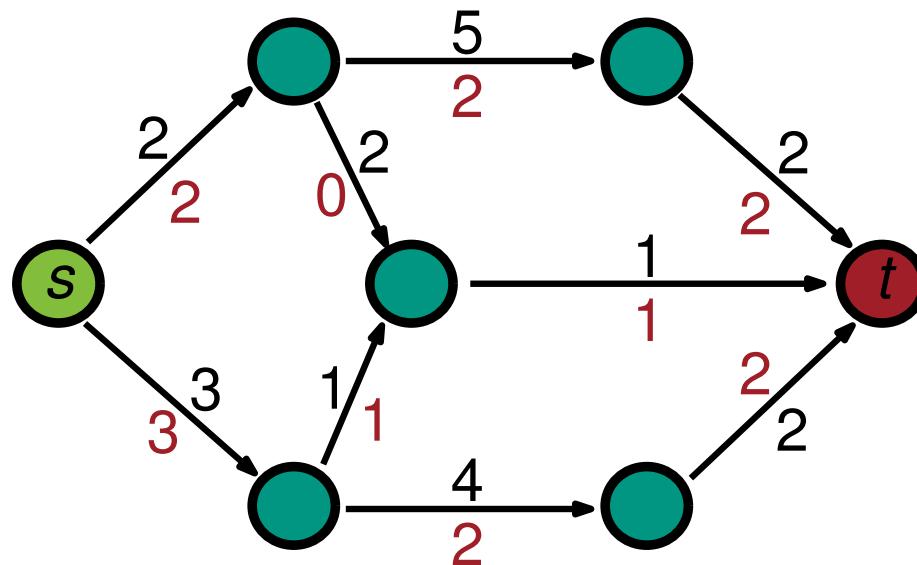


# Most Balanced Minimum Cut

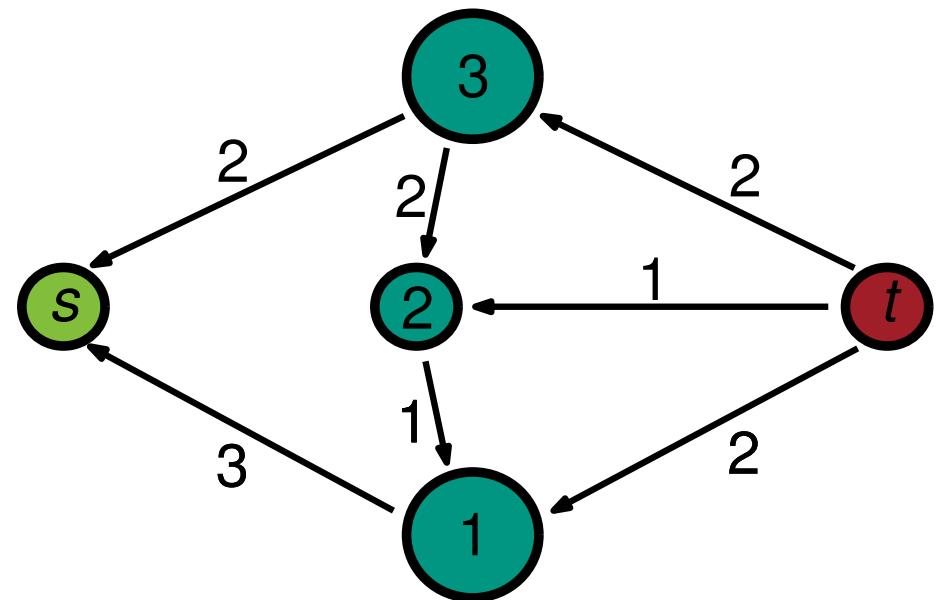
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



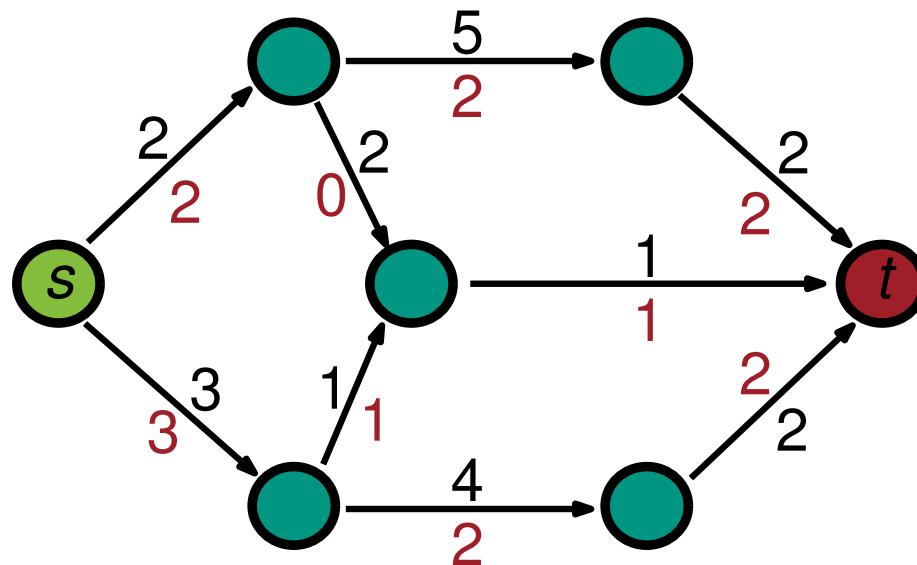
Contract all *strongly connected components* in the residual graph

# Most Balanced Minimum Cut

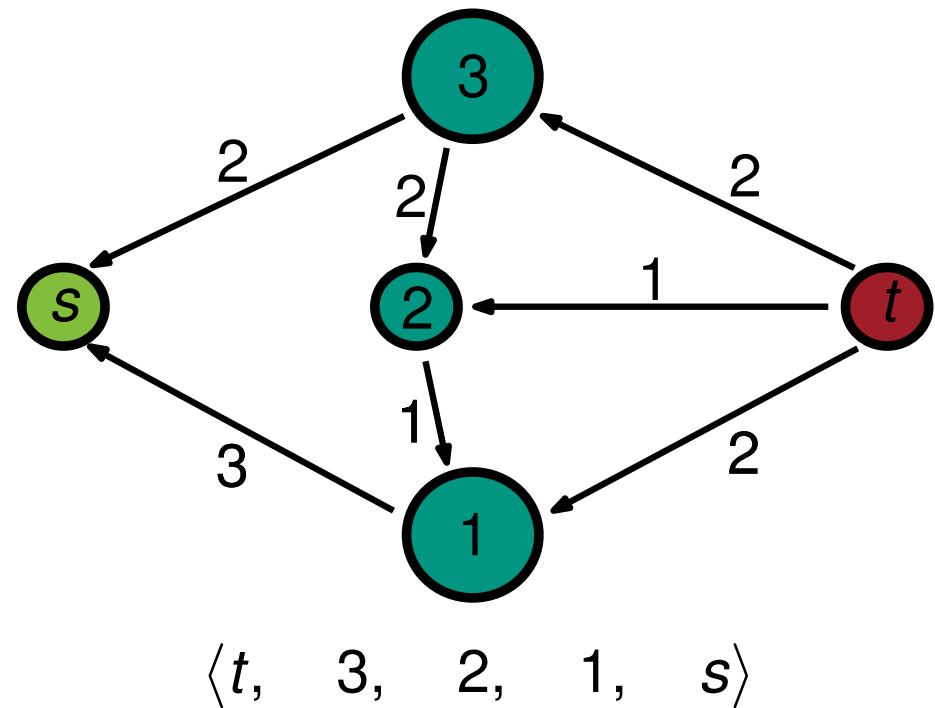
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



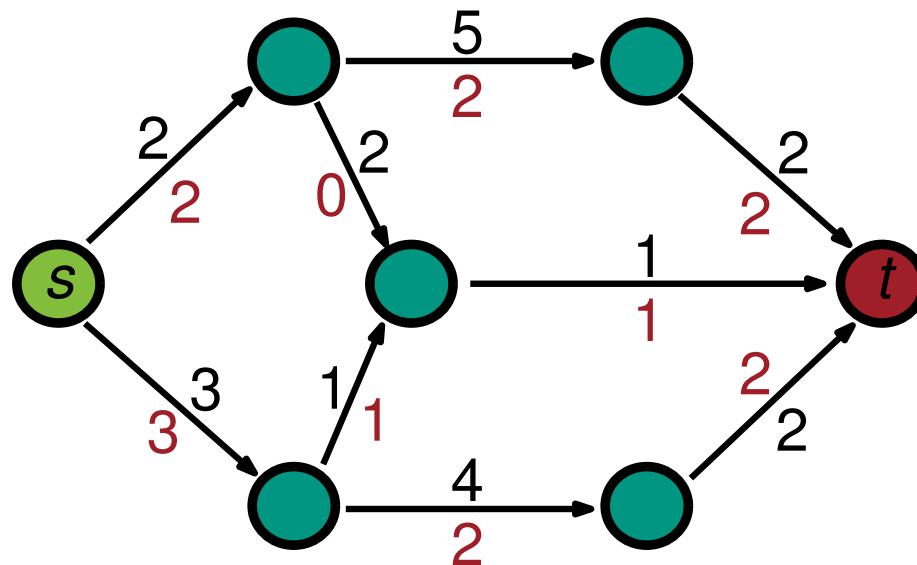
Find *topological order*

# Most Balanced Minimum Cut

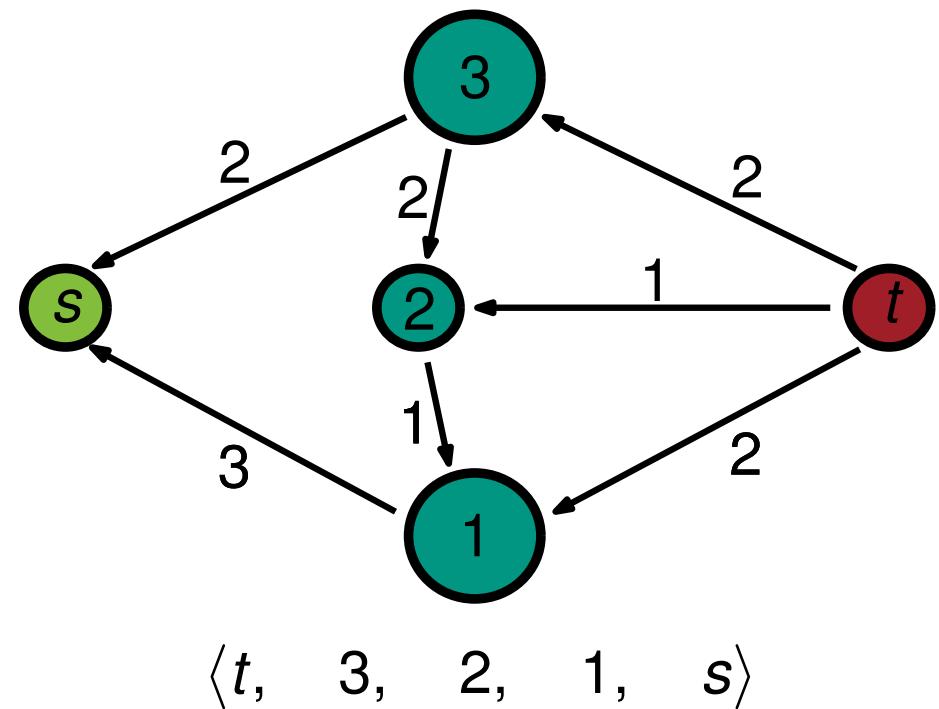
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



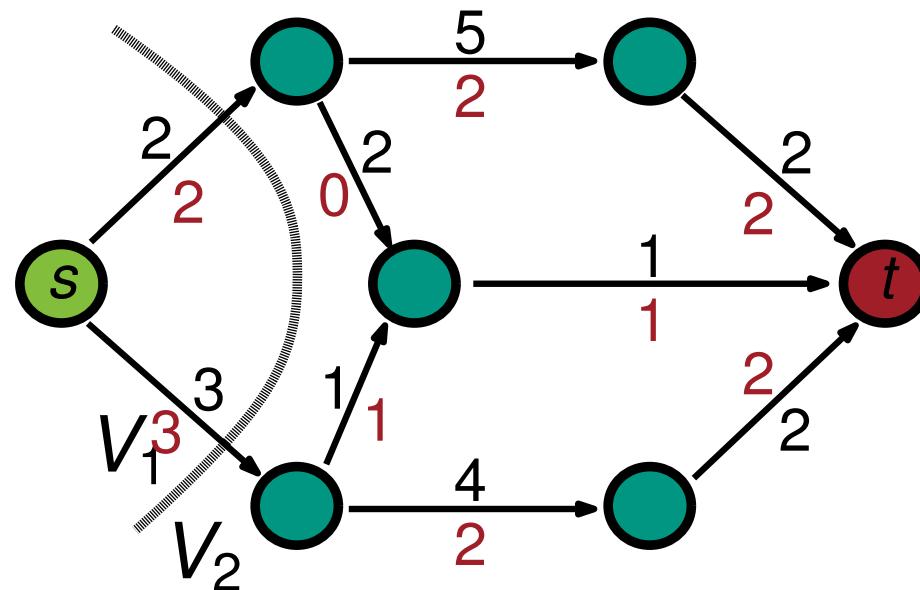
Sweep through **reverse** topological order

# Most Balanced Minimum Cut

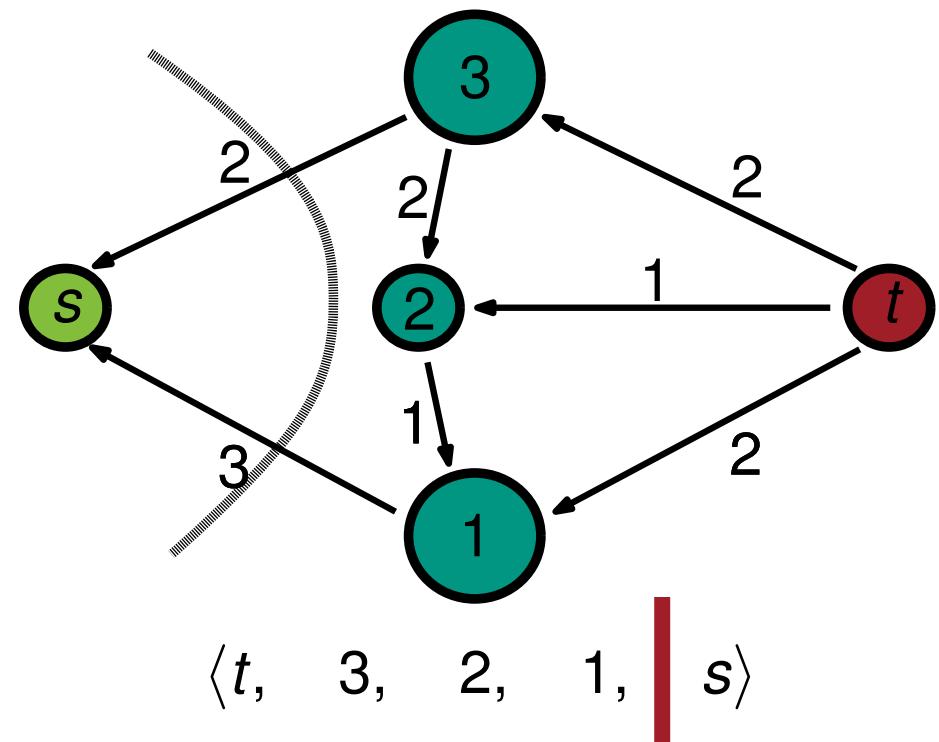
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



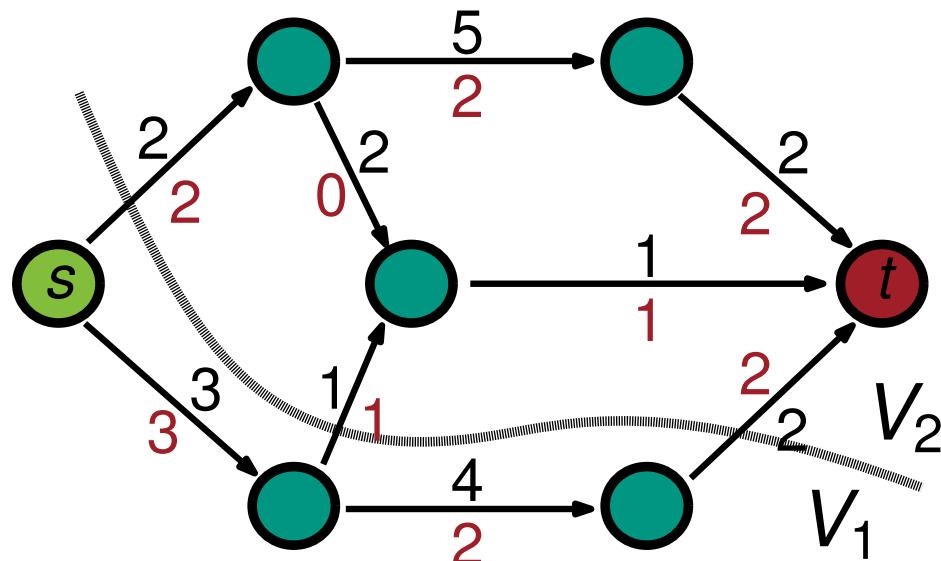
Sweep through **reverse** topological order

# Most Balanced Minimum Cut

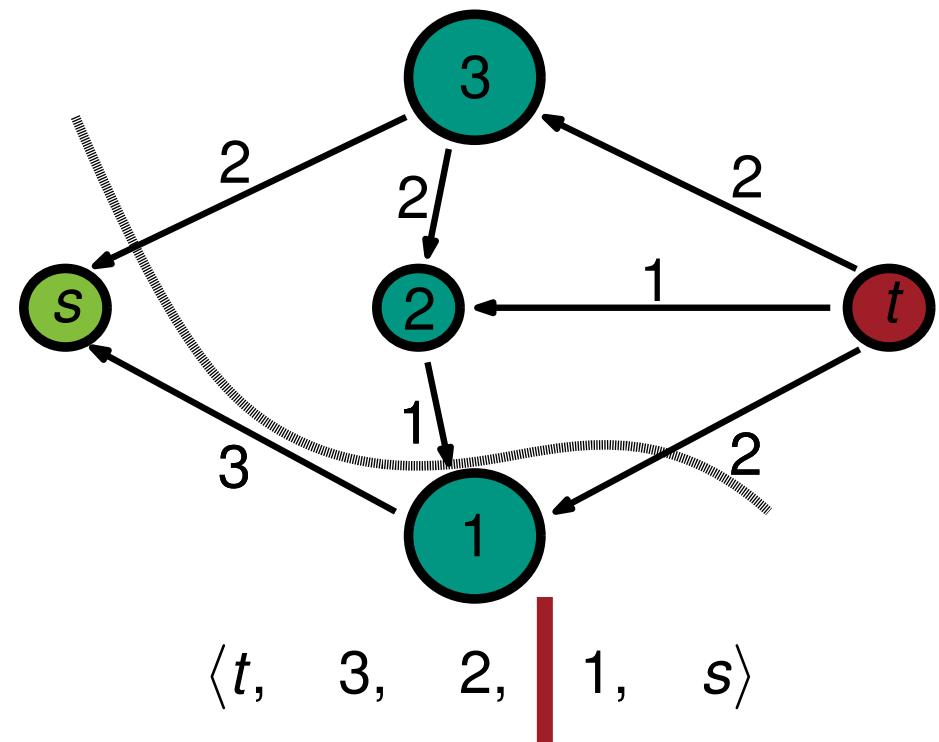
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



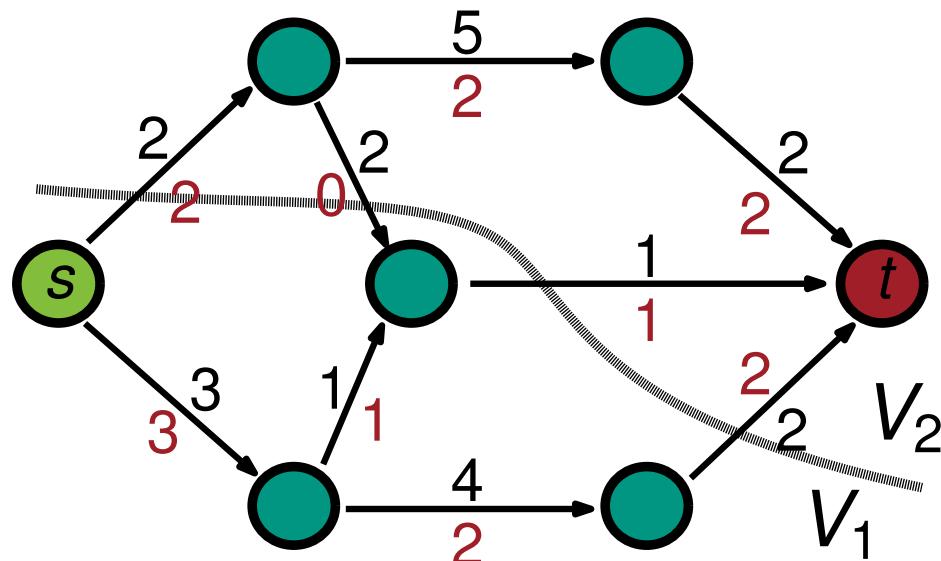
Sweep through **reverse** topological order

# Most Balanced Minimum Cut

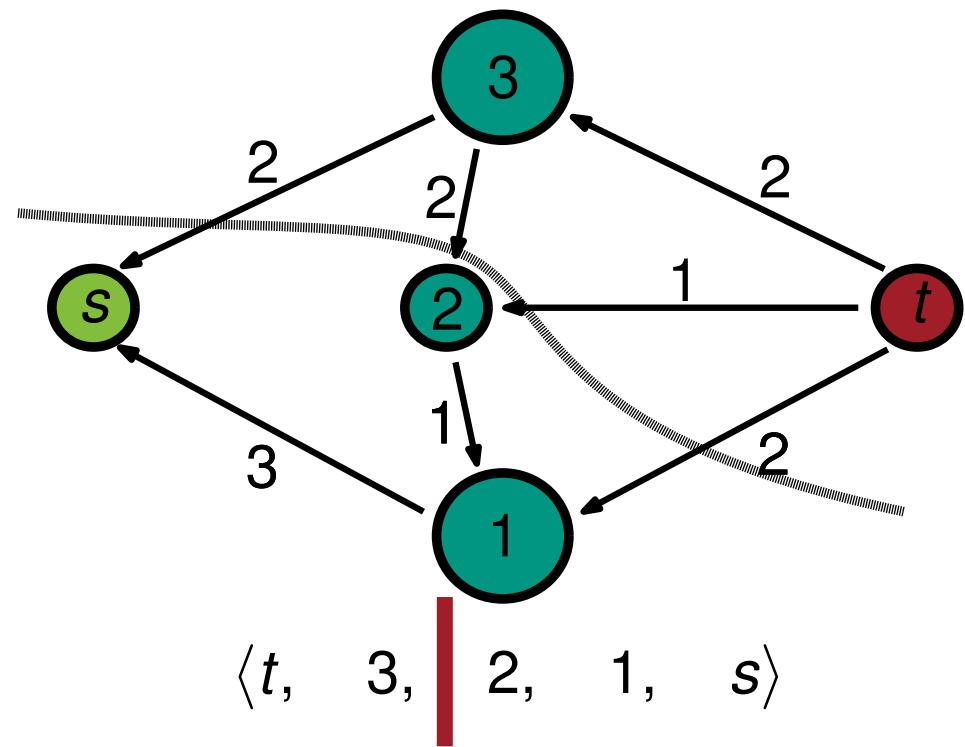
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



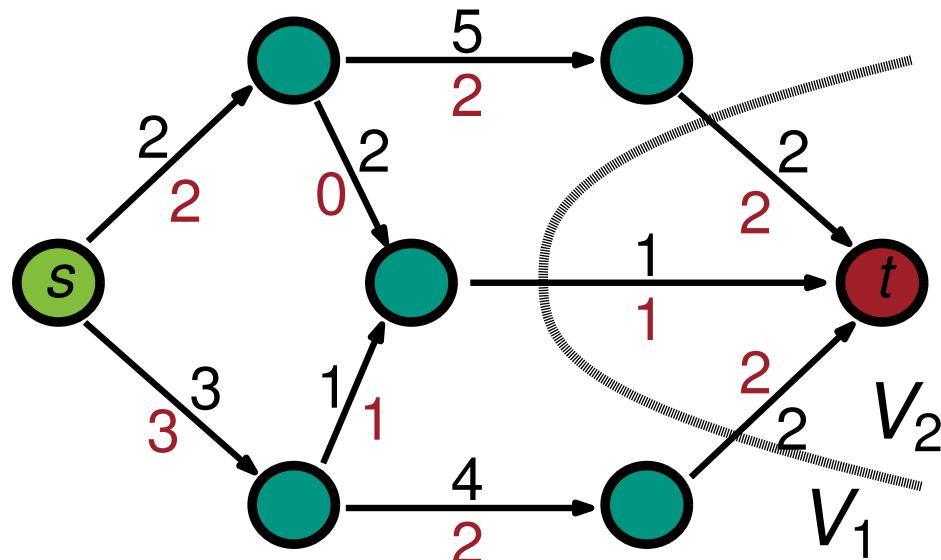
Sweep through **reverse** topological order

# Most Balanced Minimum Cut

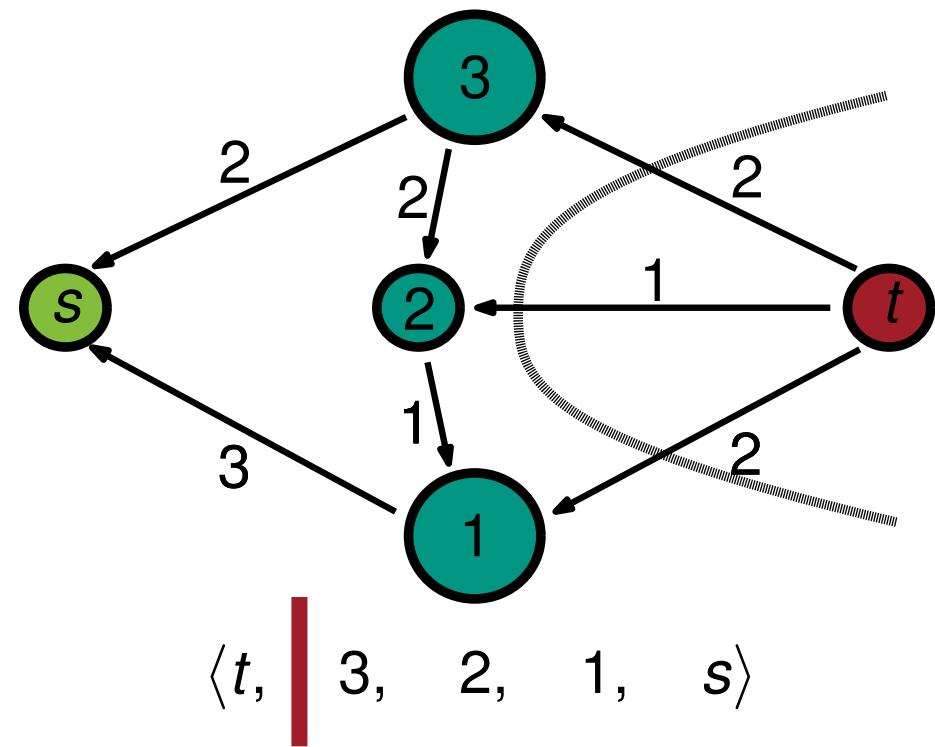
KaFFPa [Sanders, Schulz 11]

One maximum flow  $f$  has enough information to enumerate all minimum  $(s, t)$ -cuts [Picard, Queyranne 80]

Flow Graph



Picard-Queryanne DAC



Sweep through **reverse** topological order

# Integration into KaHyPar

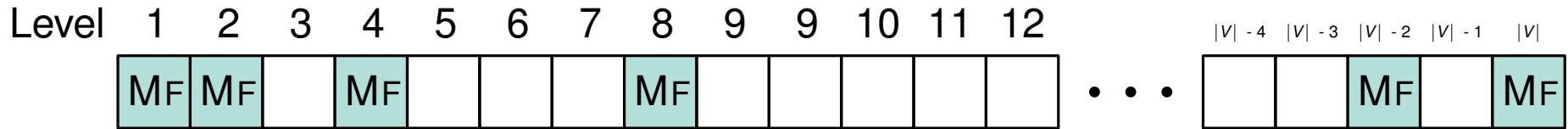
- KaHyPar is a  $n$ -level hypergraph partitioner

# Integration into KaHyPar

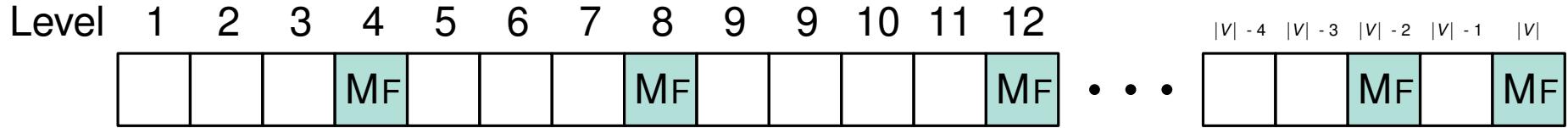
- KaHyPar is a  $n$ -level hypergraph partitioner

## Flow Execution Policies

**Exponential:** Execute *Max-Flow-Min-Cut* computations (MF) on each level  $i$  with  $i = 2^j$



**Constant:** Execute *Max-Flow-Min-Cut* computations (MF) on each level  $i$  with  $i = \beta \cdot j$

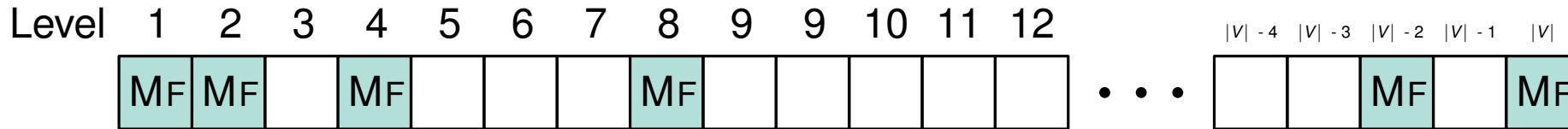


# Integration into KaHyPar

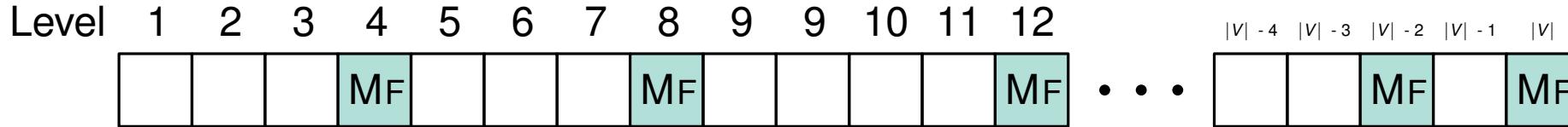
- KaHyPar is a  $n$ -level hypergraph partitioner

## Flow Execution Policies

**Exponential:** Execute *Max-Flow-Min-Cut* computations (MF) on each level  $i$  with  $i = 2^j$



**Constant:** Execute *Max-Flow-Min-Cut* computations (MF) on each level  $i$  with  $i = \beta \cdot j$



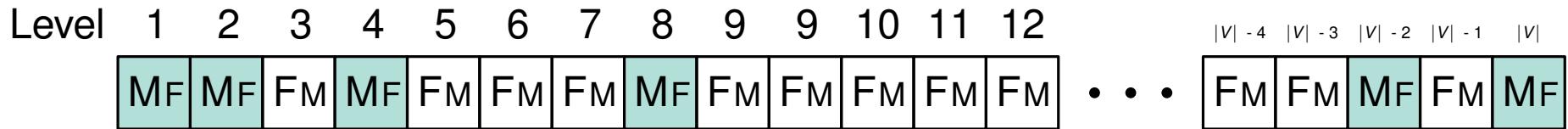
Note, each policy uses *flow-based refinement* on the **last level** 

# Integration into KaHyPar

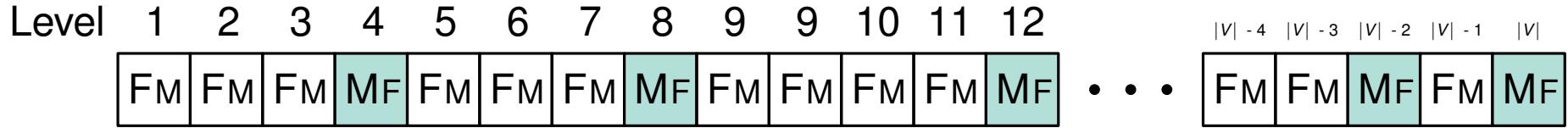
- KaHyPar is a  $n$ -level hypergraph partitioner

## Flow Execution Policies

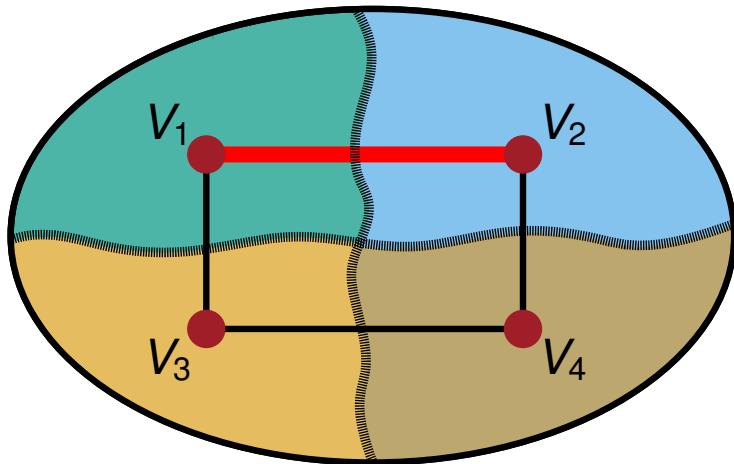
**Exponential:** Execute *Max-Flow-Min-Cut* computations (MF) on each level  $i$  with  $i = 2^j$



**Constant:** Execute *Max-Flow-Min-Cut* computations (MF) on each level  $i$  with  $i = \beta \cdot j$

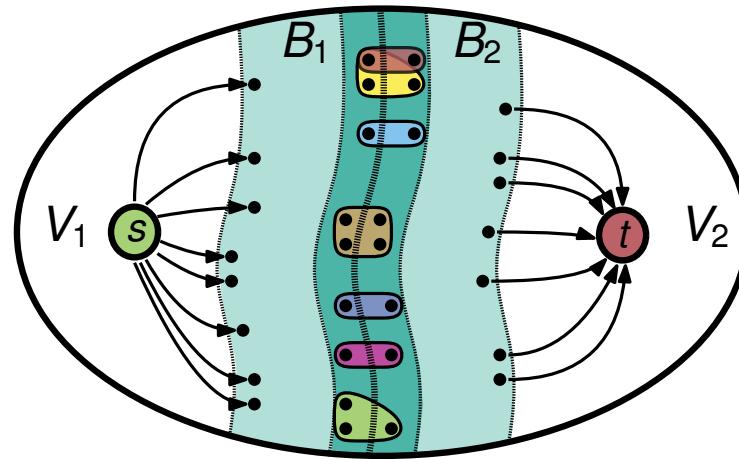


# Speed-Up Heuristics



**Active Block Scheduling**

- (R1) If cut between two blocks is small (e.g.  $\leq 10$ ) skip flow-based refinement, except on the last level
- (R2) Only execute flow-based refinement if previous computations lead to an improvement (except in first round)



**Adaptive Flow Iterations**

- (R3) If no hypernode change its block after *Max-Flow-Min-Cut* computation, then break

# Experimental Setup

## System

- Intel Xeon E5-2670 Octa-Core (2.6 GHz)
- 64 GB Main Memory
- 20 MB L3-Cache,  $8 \times 256$  KB L2-Cache
- **Compiler:** g++-5.2
- **Flags:**  $-O3 -mtune=native -march=native$

## Flow Algorithms

- EDMOND KARP
- GOLDBERG TARJAN
- BOYKOV KOLMOGOROV
- IBFS (fastest in our experiments)

# Experimental Setup

## System

- Intel Xeon E5-2670 Octa-Core (2.6 GHz)
- 64 GB Main Memory
- 20 MB L3-Cache,  $8 \times 256$  KB L2-Cache
- **Compiler:** g++-5.2
- **Flags:**  $-O3 -mtune=native -march=native$

## Flow Algorithms

- EDMOND KARP
- GOLDBERG TARJAN

Own Implementations

- BOYKOV KOLMOGOROV
- IBFS (fastest in our experiments)

Third-Party Implementations

# Experimental Setup

## Benchmarks

- Parameter Tuning Benchmark Set (25 Hypergraphs)
- Benchmark Subset (165 Hypergraphs)
- Full Benchmark Set (488 Hypergraphs)

## Benchmark Type

- DAC (10 Hypergraphs)
- ISPD98 (18 Hypergraphs)
- PRIMAL (92 Hypergraphs)
- LITERAL (92 Hypergraphs)
- DUAL (92 Hypergraphs)
- SPM (184 Hypergraphs)

# Experimental Setup

## Benchmarks

- Parameter Tuning Benchmark Set (25 Hypergraphs)
- Benchmark Subset (165 Hypergraphs)
- Full Benchmark Set (488 Hypergraphs)

## Benchmark Type

- DAC (10 Hypergraphs)
- ISPD98 (18 Hypergraphs)

VLSI Design

- PRIMAL (92 Hypergraphs)
- LITERAL (92 Hypergraphs)
- DUAL (92 Hypergraphs)

SAT Formulas

- SPM (184 Hypergraphs)

Sparse Matrices

# Experimental Setup

## Benchmarks

- Parameter Tuning Benchmark Set (25 Hypergraphs)
- Benchmark Subset (165 Hypergraphs)
- Full Benchmark Set (488 Hypergraphs)

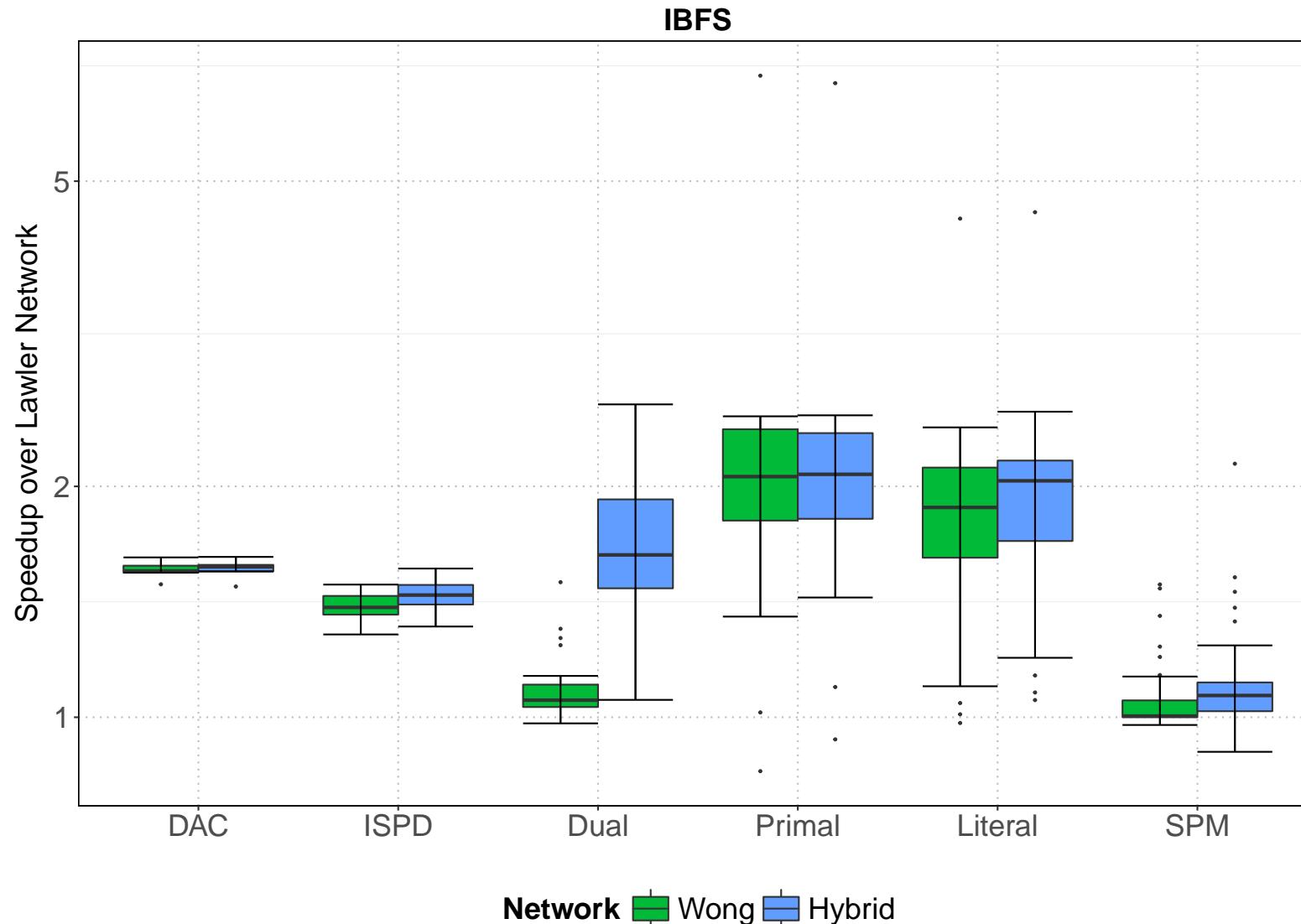
## Benchmark Type

- DAC (10 Hypergraphs)
- ISPD98 (18 Hypergraphs)
- PRIMAL (92 Hypergraphs)
- LITERAL (92 Hypergraphs)
- DUAL (92 Hypergraphs)
- SPM (184 Hypergraphs)

## Methodology

- $\varepsilon = 3\%$
- $k \in \{2, 4, 8, 16, 32, 64, 128\}$
- 10 seeds

# Flow Networks



# Flow Configuration

- $+/- F$  = Enabled/Disabled Flow-based refinement
- $+/- M$  = Enabled/Disabled Most Balanced Minimum Cut
- $+/- FM$  = Enabled/Disabled FM Heuristic
- CONSTANT128 =  $(+F,+M,+FM)$  with **constant** flow execution policy and  $\beta = 128$

Config. $\alpha'$	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	-6.09	12.91	-5.60	13.40	0.23	15.16	0.54	67.90
2	-3.19	15.75	-2.07	16.74	0.73	17.51	1.11	140.24
4	-1.82	20.37	-0.19	21.88	1.21	21.53	1.66	269.63
8	-0.85	28.49	0.98	30.67	1.71	28.68	2.20	512.12
16	-0.19	43.32	1.75	46.66	2.21	41.31	2.76	973.91
Ref.	(-F,-M,+FM)	6373.88	13.73					

# Flow Configuration

Improvement of **MBMC** more significantly for large  $\alpha'$ :

- $\alpha' \in \{1, 2\}$ : Improvement 0.5% to 1.12%
- $\alpha' \in \{4, 8, 16\}$ : Improvement 1.63% to 1.94%

Config. $\alpha'$	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
	Avg [%]	t[s]	Avg [%]	t[s]	Avg [%]	t[s]	Avg [%]	t[s]
1	-6.09	12.91	-5.60	13.40	0.23	15.16	0.54	67.90
2	-3.19	15.75	-2.07	16.74	0.73	17.51	1.11	140.24
4	-1.82	20.37	-0.19	21.88	1.21	21.53	1.66	269.63
8	-0.85	28.49	0.98	30.67	1.71	28.68	2.20	512.12
16	-0.19	43.32	1.75	46.66	2.21	41.31	2.76	973.91
Ref.	(-F,-M,+FM)	6373.88	13.73					

# Flow Configuration

Flows with FM are faster than flows on its own for large  $\alpha'$

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
	$\alpha'$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]
1	-6.09	12.91	-5.60	13.40	0.23	15.16	0.54	67.90
2	-3.19	15.75	-2.07	16.74	0.73	17.51	1.11	140.24
4	-1.82	20.37	-0.19	21.88	1.21	21.53	1.66	269.63
8	-0.85	28.49	0.98	30.67	1.71	28.68	2.20	512.12
16	-0.19	43.32	1.75	46.66	2.21	41.31	2.76	973.91
Ref.	(-F,-M,+FM)		6373.88	13.73				

# Flow Configuration

Quality improvement around 0.3% to 0.5%, but running time is impractical

Config. $\alpha'$	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
	Avg [%]	t[s]	Avg [%]	t[s]	Avg [%]	t[s]	Avg [%]	t[s]
1	-6.09	12.91	-5.60	13.40	0.23	15.16	0.54	67.90
2	-3.19	15.75	-2.07	16.74	0.73	17.51	1.11	140.24
4	-1.82	20.37	-0.19	21.88	1.21	21.53	1.66	269.63
8	-0.85	28.49	0.98	30.67	1.71	28.68	2.20	512.12
16	-0.19	43.32	1.75	46.66	2.21	41.31	2.76	973.91
Ref.	(-F,-M,+FM)	6373.88	13.73					

# Flow Configuration

Config. $\alpha'$	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
	Avg [%]	t[s]	Avg [%]	t[s]	Avg [%]	t[s]	Avg [%]	t[s]
1	−6.09	12.91	−5.60	13.40	0.23	15.16	0.54	67.90
2	−3.19	15.75	−2.07	16.74	0.73	17.51	1.11	140.24
4	−1.82	20.37	−0.19	21.88	1.21	21.53	1.66	269.63
8	−0.85	28.49	0.98	30.67	1.71	28.68	2.20	512.12
16	−0.19	43.32	1.75	46.66	2.21	41.31	2.76	973.91
Ref.	(−F,−M,+FM)	6373.88	13.73					

# Flow Problem Modeling

KaFFPa vs. KaHyPar

$\alpha$	HYPERGRAPHS			GRAPHS		
	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$
1	8.3	8.9	8.2	13.3	12.7	11.8
2	8.5	7.1	5.1	12.4	10.1	8.5
4	7.4	4.1	2.8	11.0	7.9	5.7
8	5.4	2.4	1.6	9.4	5.6	4.0
16	3.5	1.3	1.2	7.5	4.2	3.6

# Flow Problem Modeling

KaFFPa vs. KaHyPar

Test Configuration: (+F,-M,-FM)

25 Hypergraph Instances

15 Graph Instances

$\alpha$	HYPERGRAPHS			GRAPHS		
	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$
1	8.3	8.9	8.2	13.3	12.7	11.8
2	8.5	7.1	5.1	12.4	10.1	8.5
4	7.4	4.1	2.8	11.0	7.9	5.7
8	5.4	2.4	1.6	9.4	5.6	4.0
16	3.5	1.3	1.2	7.5	4.2	3.6

**Improvement in percentage** of our flow problem modeling approach  
relative to the approach of KaFFPa

# Flow Problem Modeling

KaFFPa vs. KaHyPar

Graphs profit more than hypergraphs from new modeling approach

$\alpha$	HYPERGRAPHS			GRAPHS		
	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$
1	8.3	8.9	8.2	13.3	12.7	11.8
2	8.5	7.1	5.1	12.4	10.1	8.5
4	7.4	4.1	2.8	11.0	7.9	5.7
8	5.4	2.4	1.6	9.4	5.6	4.0
16	3.5	1.3	1.2	7.5	4.2	3.6

# Flow Problem Modeling

KaFFPa vs. KaHyPar

The smaller the flow problem the bigger the improvement

$\alpha$	HYPERGRAPHS			GRAPHS		
	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$
1	8.3	8.9	8.2	13.3	12.7	11.8
2	8.5	7.1	5.1	12.4	10.1	8.5
4	7.4	4.1	2.8	11.0	7.9	5.7
8	5.4	2.4	1.6	9.4	5.6	4.0
16	3.5	1.3	1.2	7.5	4.2	3.6

# Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	-2.47	-2.12	43.04	72.30
KaHyPar-MF <sub>(R1)</sub>	-2.41	-2.06	33.89	63.15
KaHyPar-MF <sub>(R1,R2)</sub>	-2.40	-2.05	28.52	57.78
KaHyPar-MF <sub>(R1,R2,R3)</sub>	-2.41	-2.06	21.23	50.49

# Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	-2.47	-2.12	43.04	72.30
KaHyPar-MF <sub>(R1)</sub>	-2.41	-2.06	33.89	63.15
KaHyPar-MF <sub>(R1,R2)</sub>	-2.40	-2.05	28.52	57.78
KaHyPar-MF <sub>(R1,R2,R3)</sub>	-2.41	-2.06	21.23	50.49

Comparable Quality

# Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	-2.47	-2.12	43.04	72.30
KaHyPar-MF <sub>(R1)</sub>	-2.41	-2.06	33.89	63.15
KaHyPar-MF <sub>(R1,R2)</sub>	-2.40	-2.05	28.52	57.78
KaHyPar-MF <sub>(R1,R2,R3)</sub>	-2.41	-2.06	21.23	50.49

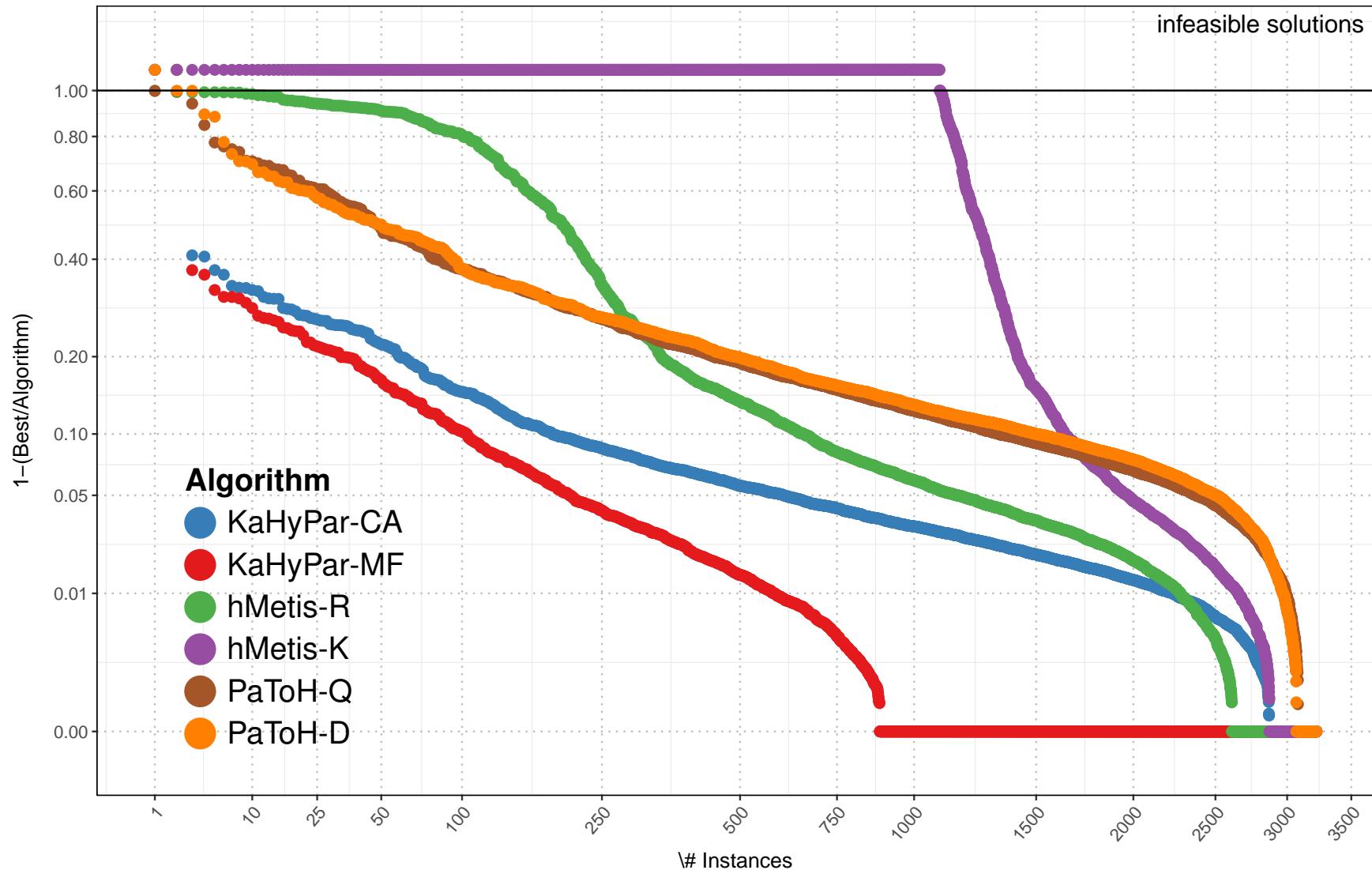
Speed-up by a factor of 2

# Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	-2.47	-2.12	43.04	72.30
KaHyPar-MF <sub>(R1)</sub>	-2.41	-2.06	33.89	63.15
KaHyPar-MF <sub>(R1,R2)</sub>	-2.40	-2.05	28.52	57.78
KaHyPar-MF <sub>(R1,R2,R3)</sub>	-2.41	-2.06	21.23	50.49

Slow-down compared to KaHyPar-CA by factor of 1.72

# Quality - Full Benchmark Set



# Running Time - Full Benchmark Set

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

# Running Time - Full Benchmark Set

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Overall Running Time

Slow-down by a factor of 1.8

# Running Time - Full Benchmark Set

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Overall Running Time

Comparable running time to hMetis-K

# Running Time - Full Benchmark Set

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Small Flow Network Instances

Slow-down by a factor of 1.4

# Running Time - Full Benchmark Set

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Large Flow Network Instances

Slow-down by a factor of 1.85

# Conclusion

## Contributions

- **Generalize framework** of Sanders and Schulz to the hypergraph partitioning context
- **Sparsification techniques** for the hypergraph flow network
  - Maximum flow algorithms are up to a factor of 2 faster
- Optimized **flow problem modeling** approach
  - Quality improvement for small flow problems around 7.5% to 13.3%
- Integration of **speedup heuristics** in framework
  - Comparable quality and speedup of framework by a factor of 2

## Experimental Results

- Outperforms 5 different systems on 73% of 3216 benchmark instances
- Improve quality of *KaHyPar* by 2.5%, while only incurring a slowdown by a factor of 1.8
- Comparable running time to *hMetis* and outperforms it on 84% of the instances

# References

[Fiduccia, Mattheyses 88]

C. M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for improving Network Partitions. In *Papers on 25 Years of Electronic Design Automation 1988*, ACM, pp.241-247.

[Hu, Moerder 85]

T.C. Hu and K. Multiterminal Flows in a Hypergraph. In VLSI Circuit Layout: Theory and Design. In *IEEE Press 1985*, pp.1-12.

[Lawler 73]

E. Lawler. Cutsets and Partitions of Hypergraphs. *Networks 3 1973*, pp.275-285.

[Liu, Wong 98]

H. Liu and D. Wong. Network Flow Based Multi-way Partitioning with Area and Pin Constraints. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 1998*, pp.50-59.

[Picard, Queyranne 80]

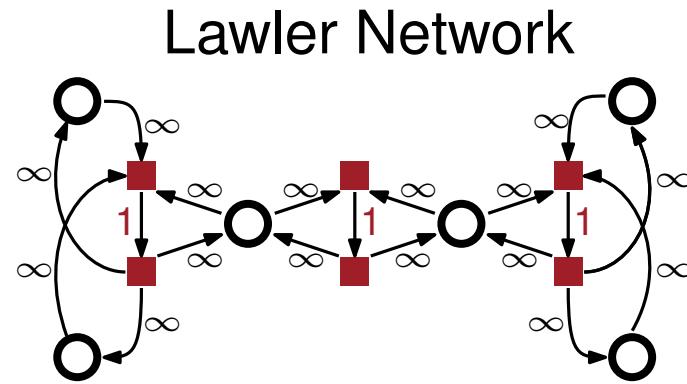
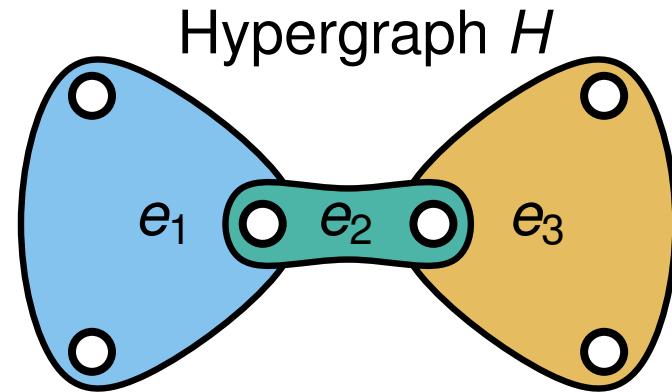
J.-C. Picard and M. Queyranne. On the Structure of all Minimum Cuts in a Network and Applications. In *Combinatorial Optimization II 1980*, pp.8-16.

[Sanders, Schulz 11]

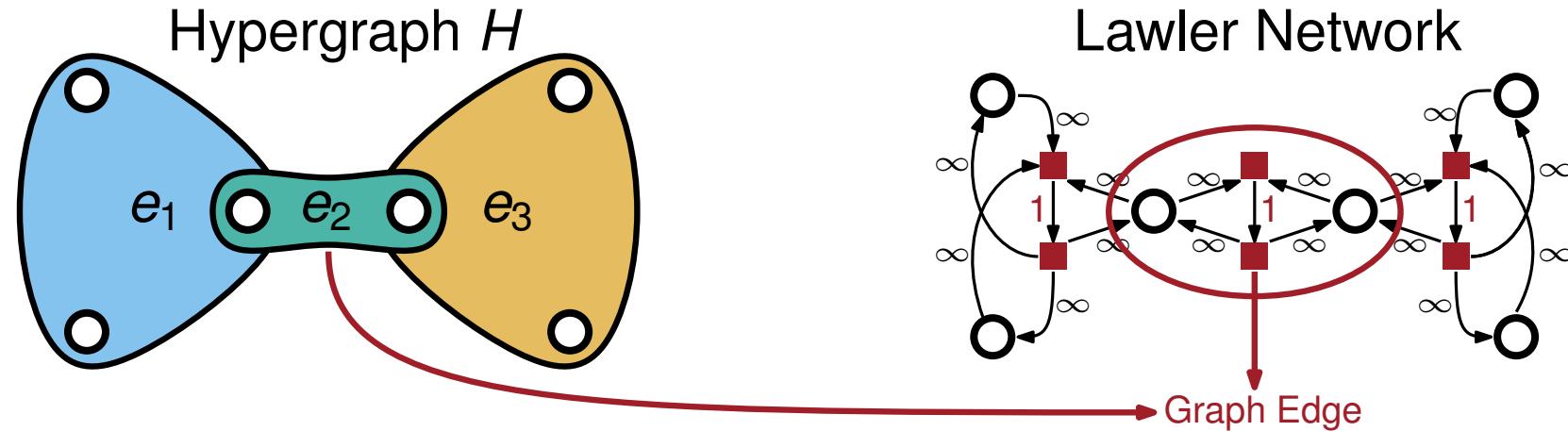
P. Sanders and C. Schulz. Engineering Multilevel Graph Partitioning Algorithms. *ESA 2011*, vol. 6942, Springer, pp.469-480.

# Appendix

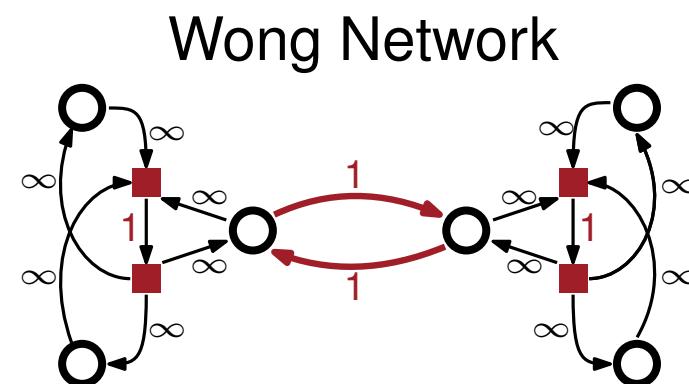
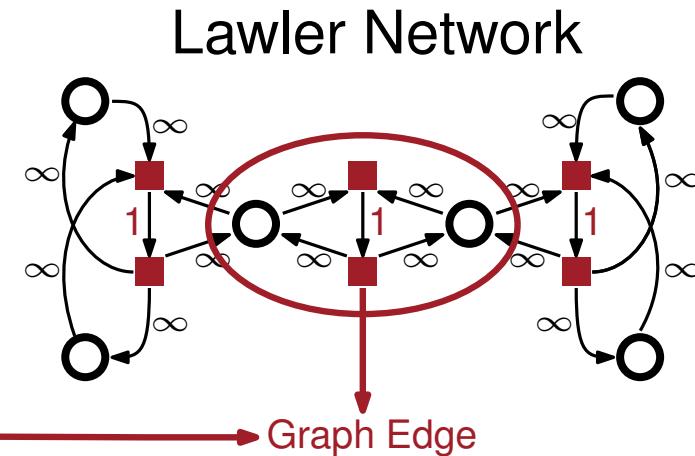
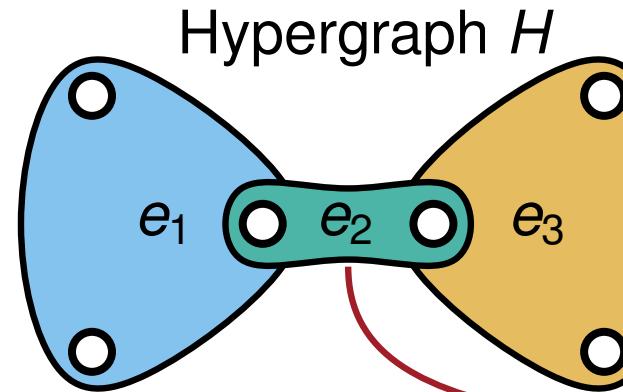
# Hypergraph Flow Network - Graph Edges



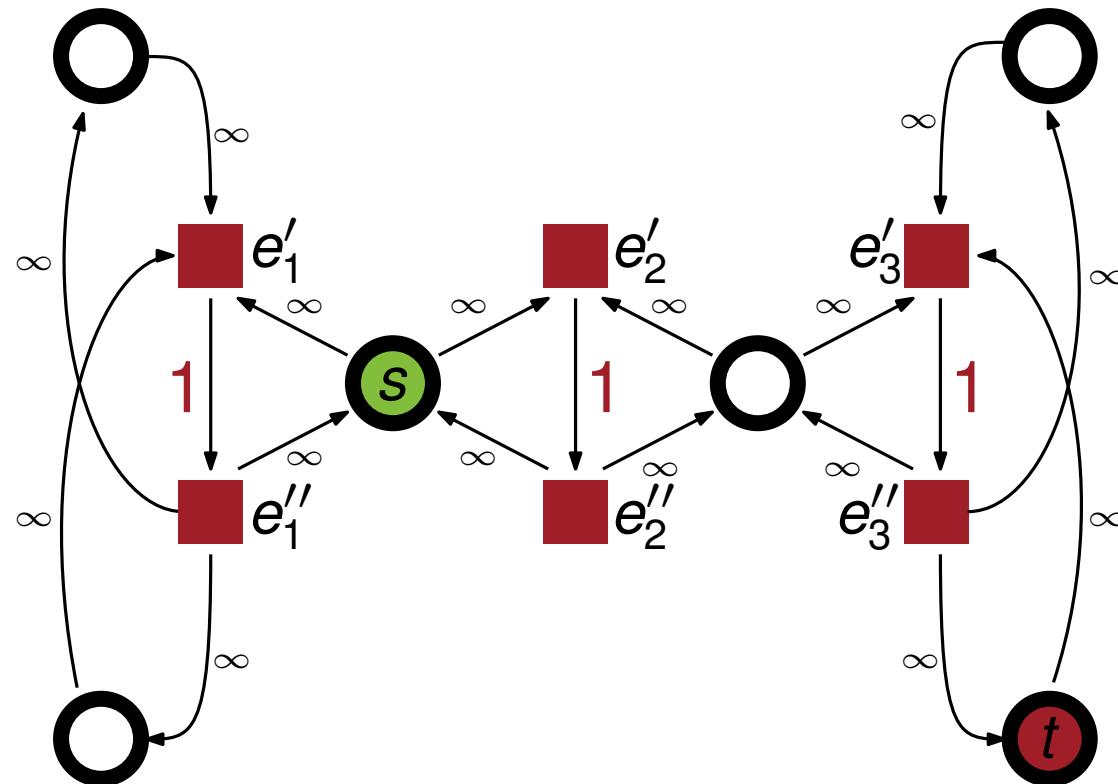
# Hypergraph Flow Network - Graph Edges



# Hypergraph Flow Network - Graph Edges

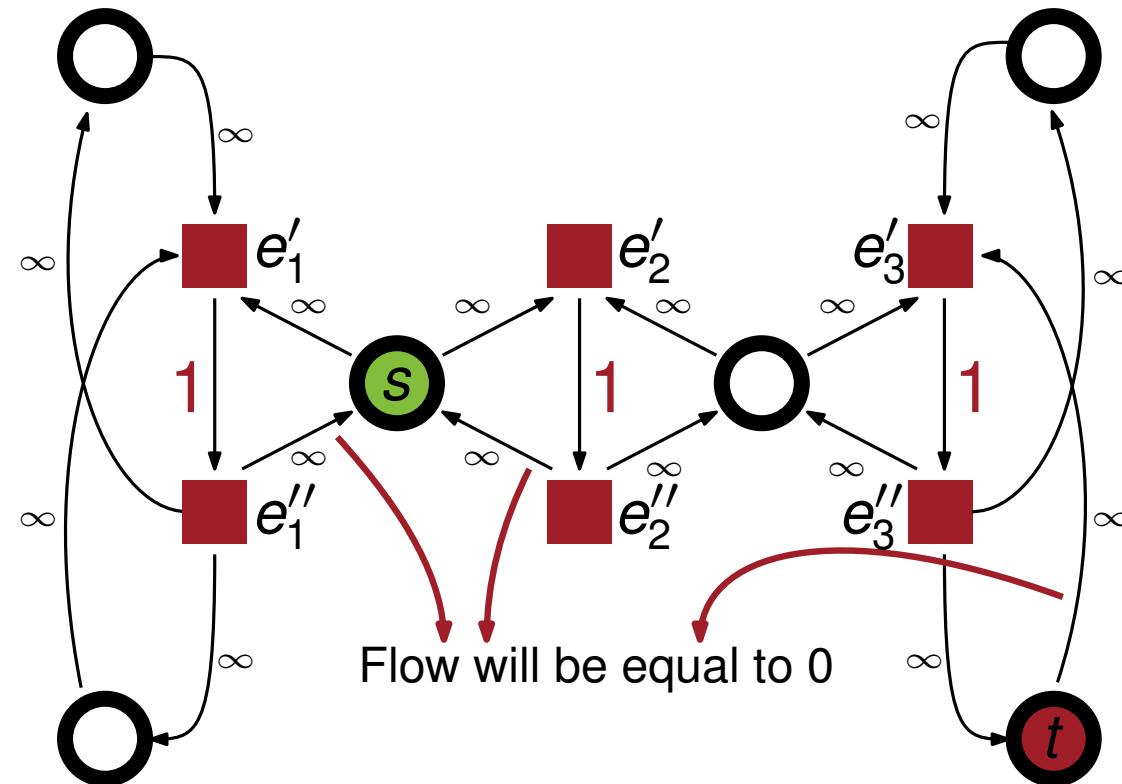


# Removing Source and Sink Vertices



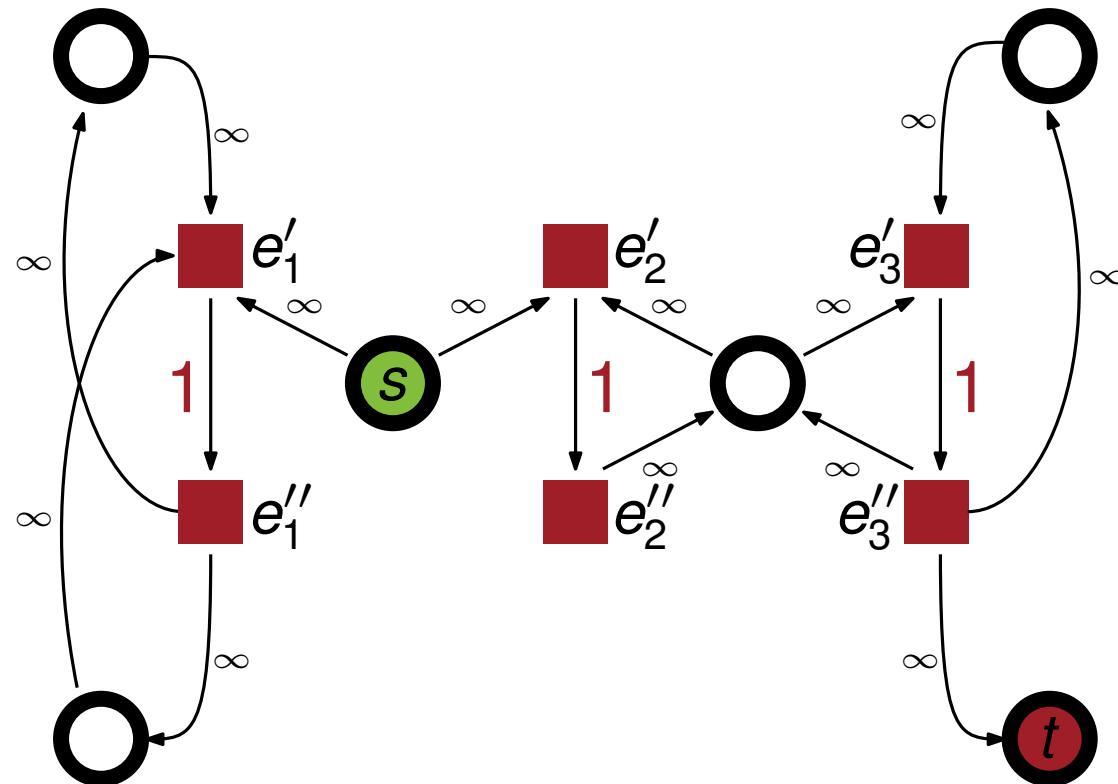
Lawler Network

# Removing Source and Sink Vertices



Lawler Network

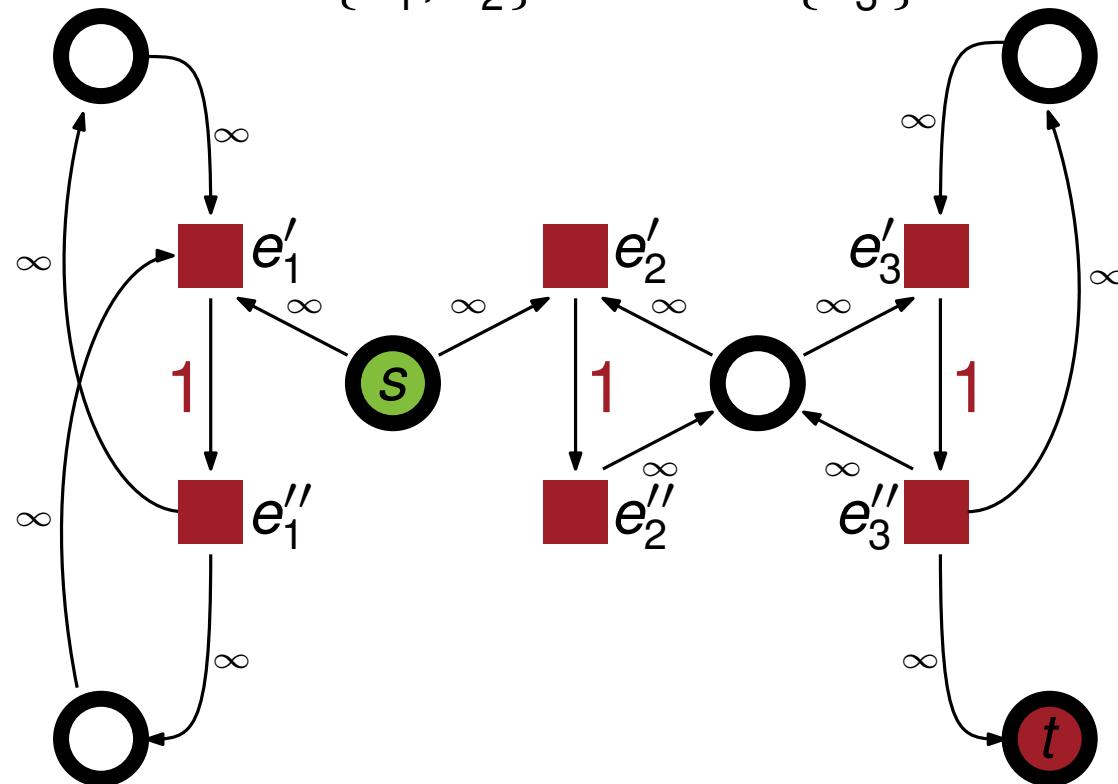
# Removing Source and Sink Vertices



Lawler Network

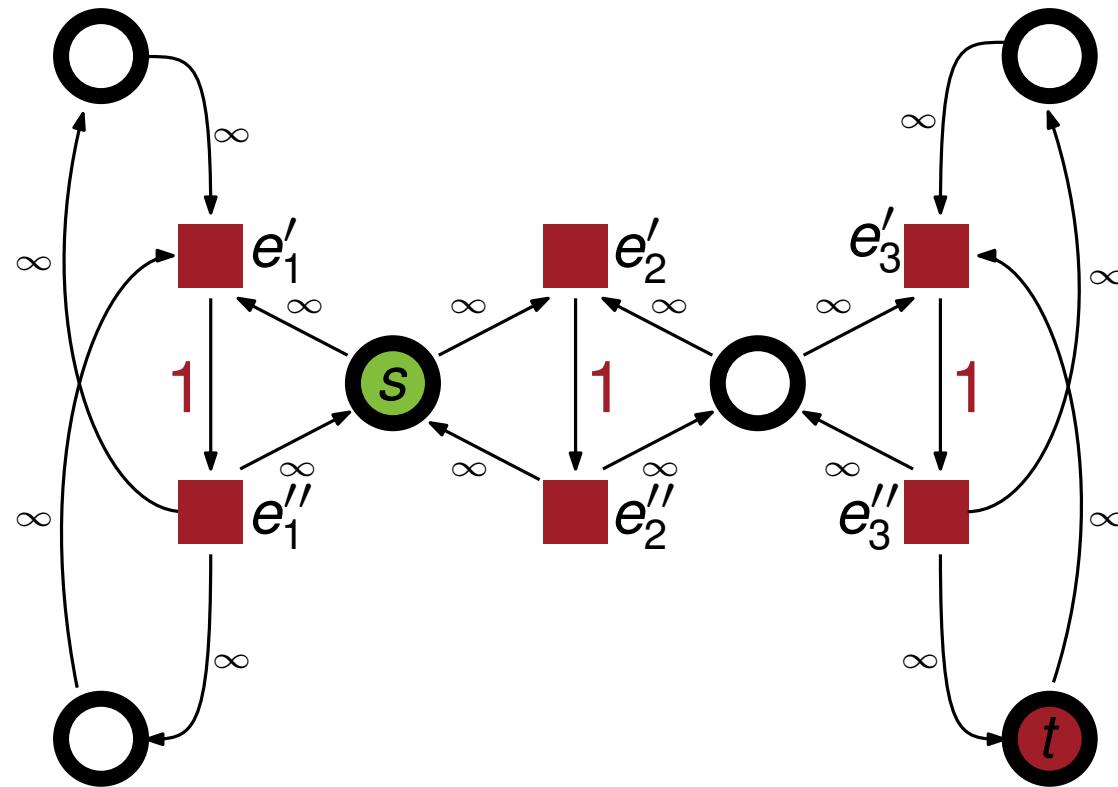
# Removing Source and Sink Vertices

Corresponds to *Multi-Source Multi-Sink* problem with  
 $S = \{e'_1, e'_2\}$  and  $T = \{e''_3\}$

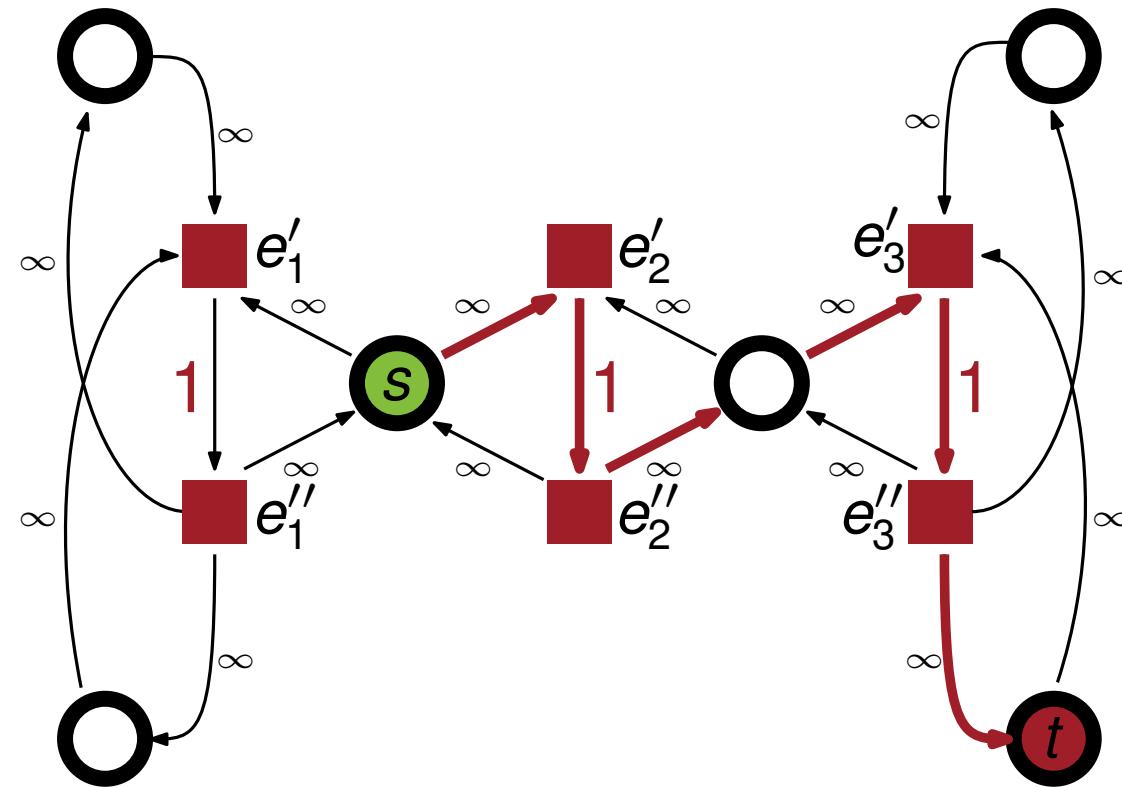


Lawler Network

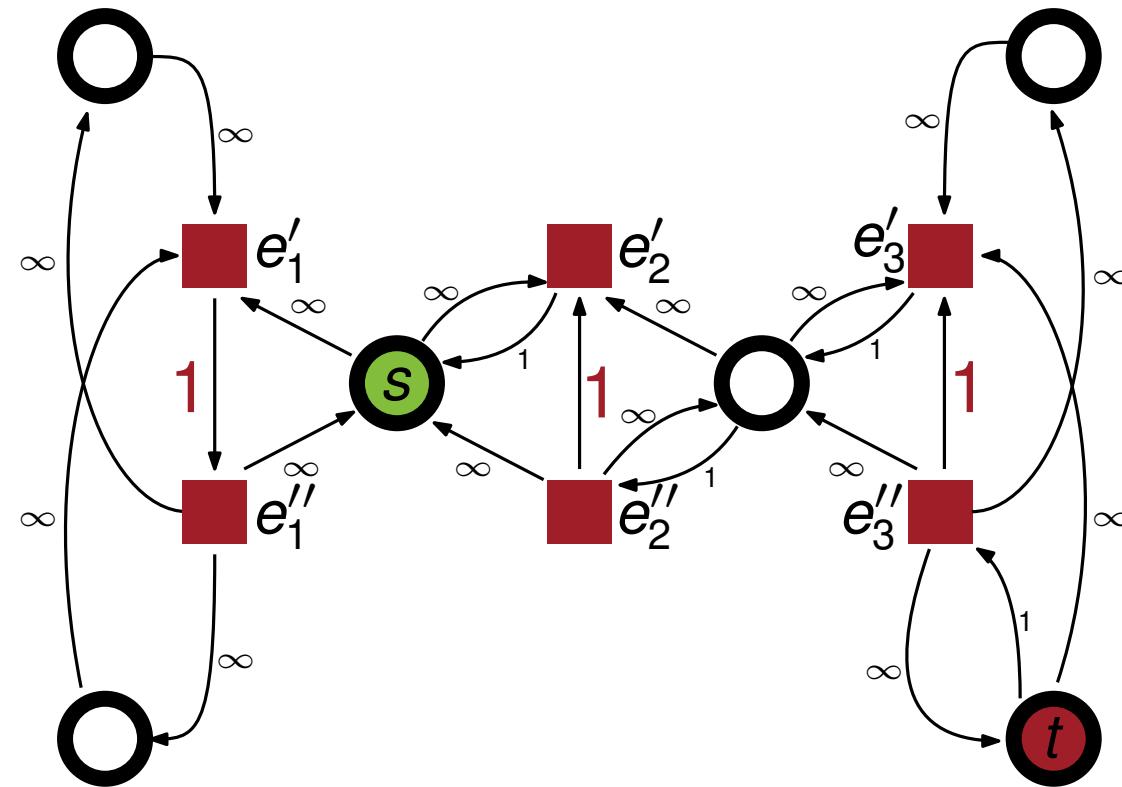
# Minimum $(s, t)$ -Bipartition



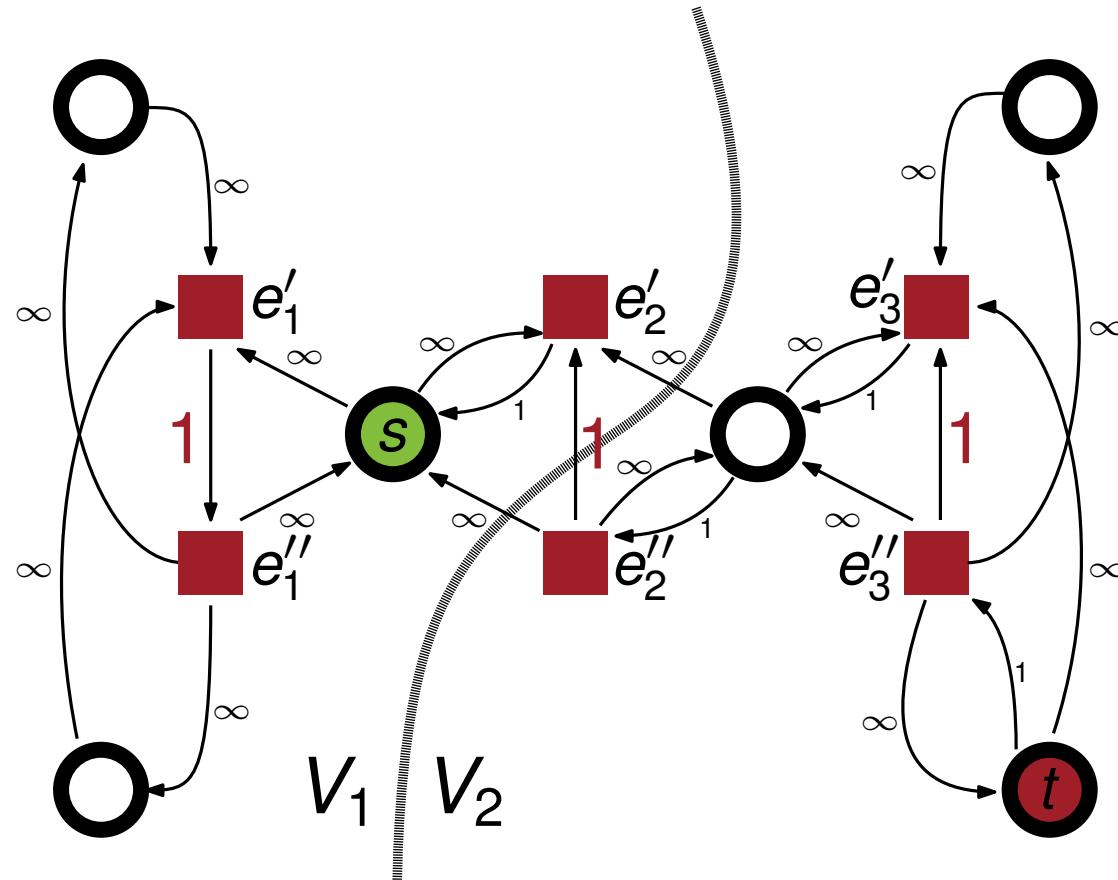
# Minimum $(s, t)$ -Bipartition



# Minimum $(s, t)$ -Bipartition



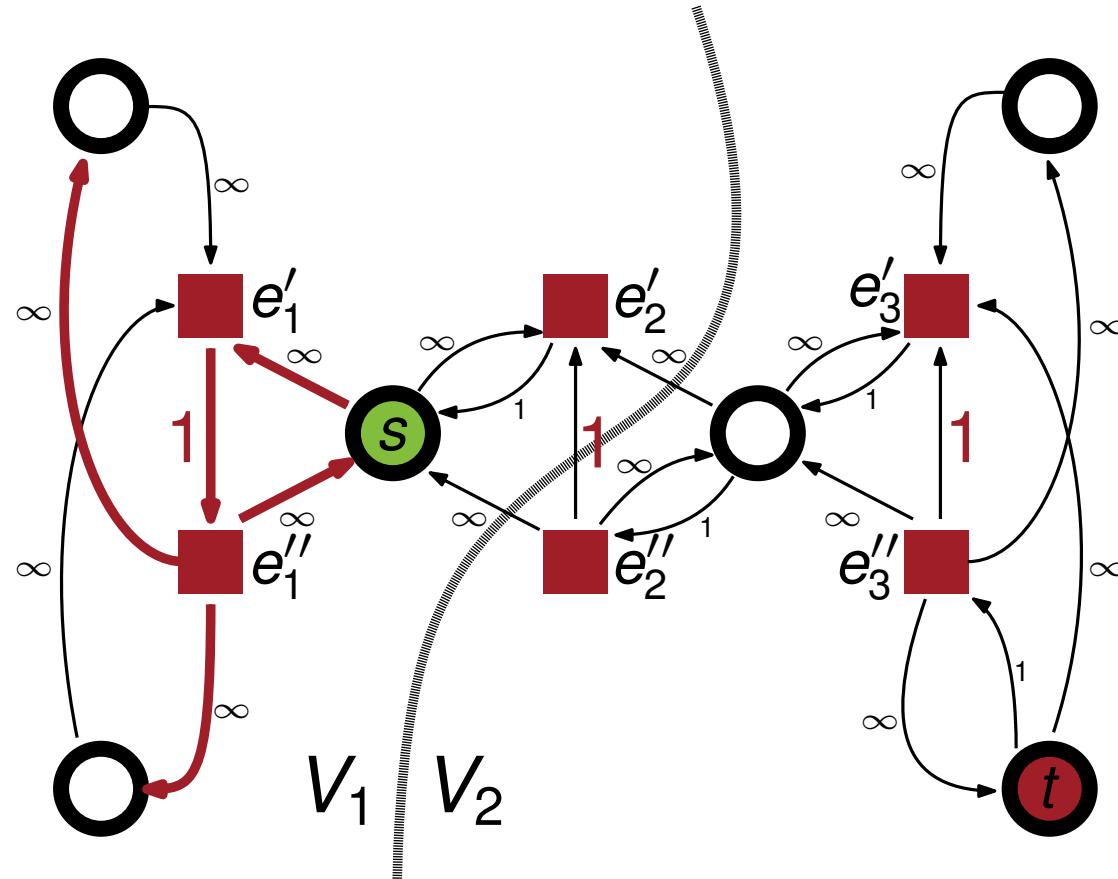
# Minimum $(s, t)$ -Bipartition



All nodes *reachable* from  $s$  are part of  $V_1$  and  $V_2 = V \setminus V_1$

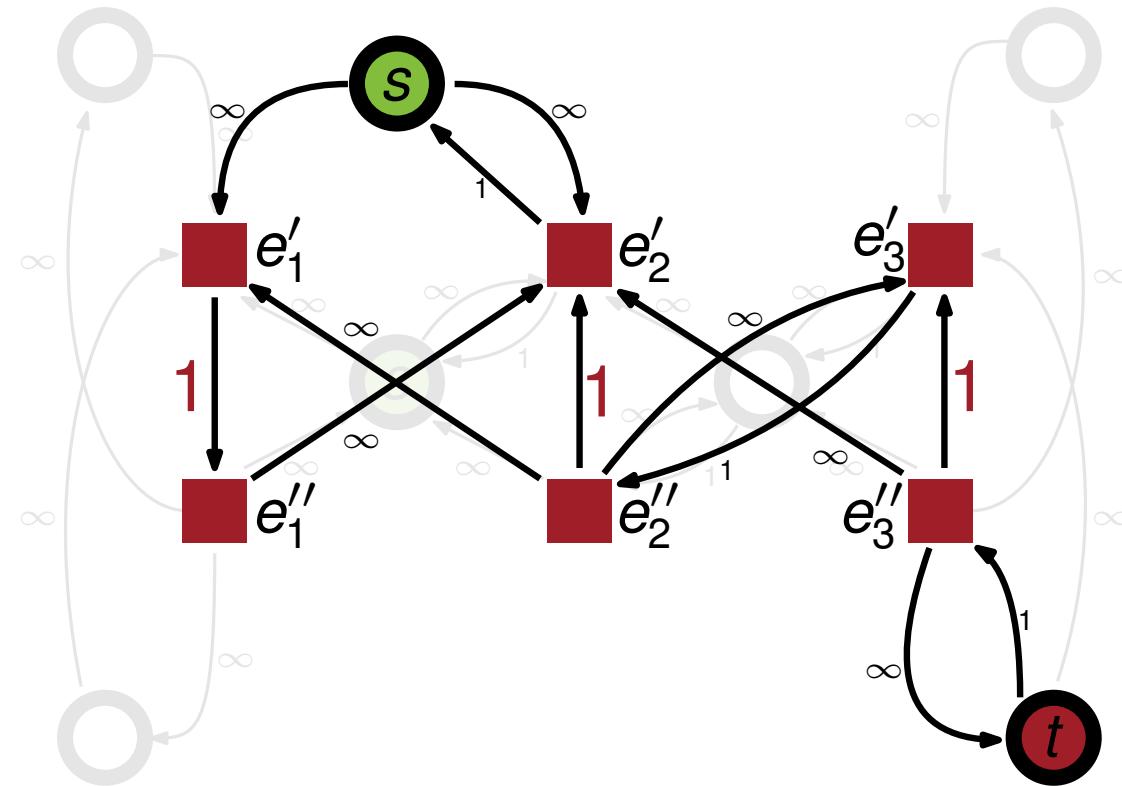
# Minimum $(s, t)$ -Bipartition

For each hypernode  $v \in V_1$ , there exists at least one  $e \in I(v)$  with  $e'' \in V_1$

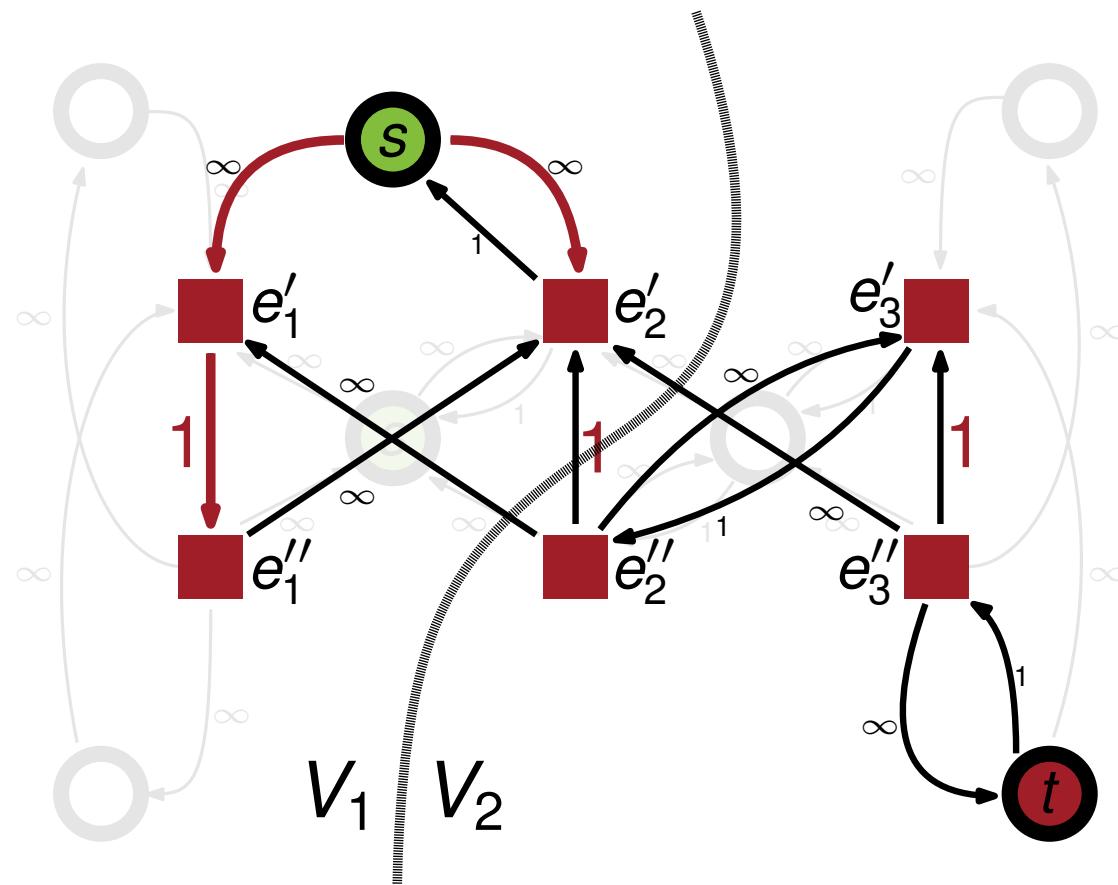


All nodes *reachable* from  $s$  are part of  $V_1$  and  $V_2 = V \setminus V_1$

# Minimum $(s, t)$ -Bipartition



# Minimum $(s, t)$ -Bipartition



# Minimum $(s, t)$ -Bipartition

