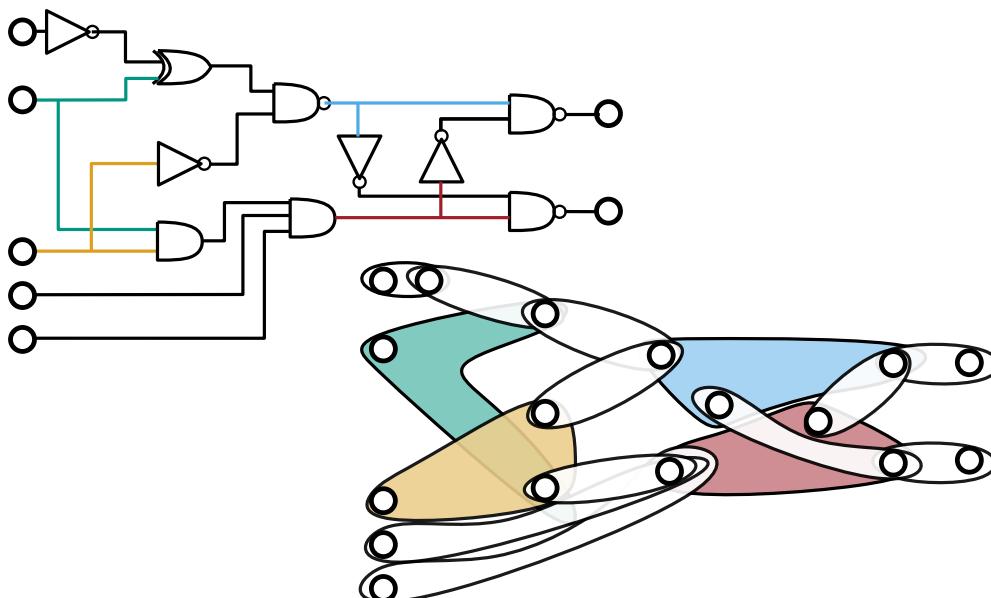


High Quality Hypergraph Partitioning via Max-Flow-Min-Cut Computations

Master Thesis · February 16, 2018
Tobias Heuer

INSTITUTE OF THEORETICAL INFORMATICS · ALGORITHMIC GROUP



Task

Developing a **refinement** algorithm based on **Max-Flow-Min-Cut** computations for the n -level hypergraph partitioner **KaHyPar**.

Task

Developing a **refinement** algorithm based on **Max-Flow-Min-Cut** computations for the n -level hypergraph partitioner **KaHyPar**.

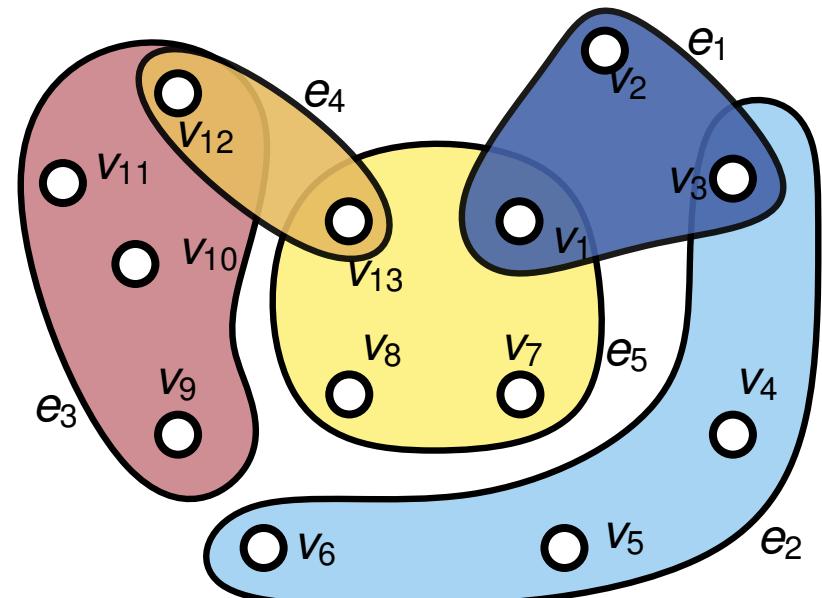
Contributions

- 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

Hypergraphs

[from SEA'17]

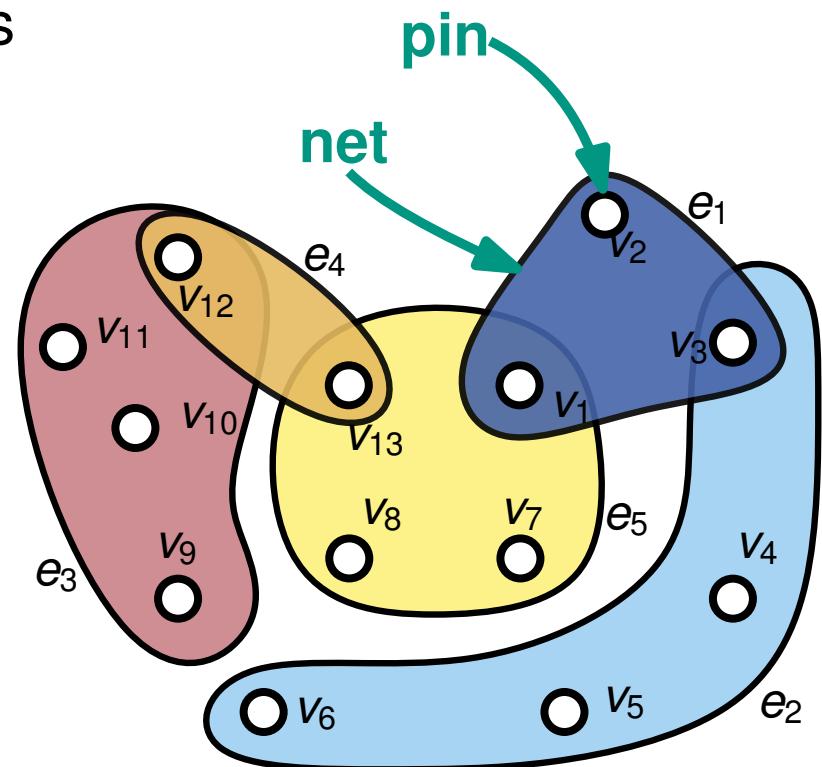
- Generalization of graphs
⇒ hyperedges connect ≥ 2 nodes
- Graphs ⇒ dyadic (**2-ary**) relationships
- Hypergraphs ⇒ (**d-ary**) relationships
- Hypergraph $H = (V, E, c, \omega)$
 - Vertex set $V = \{1, \dots, n\}$
 - Edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
 - Node weights $c : V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



Hypergraphs

[from SEA'17]

- Generalization of graphs
⇒ hyperedges connect ≥ 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 $\omega : E \rightarrow \mathbb{R}_{\geq 1}$
- $|P| = \sum_{e \in E} |e| = \sum_{v \in V} d(v)$



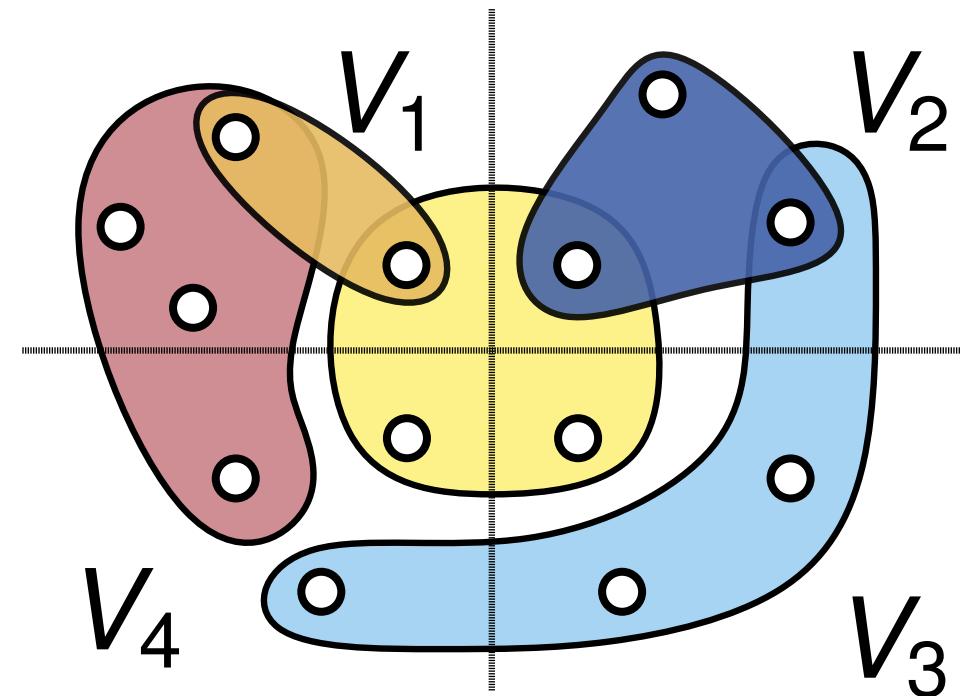
Hypergraph Partitioning Problem

[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

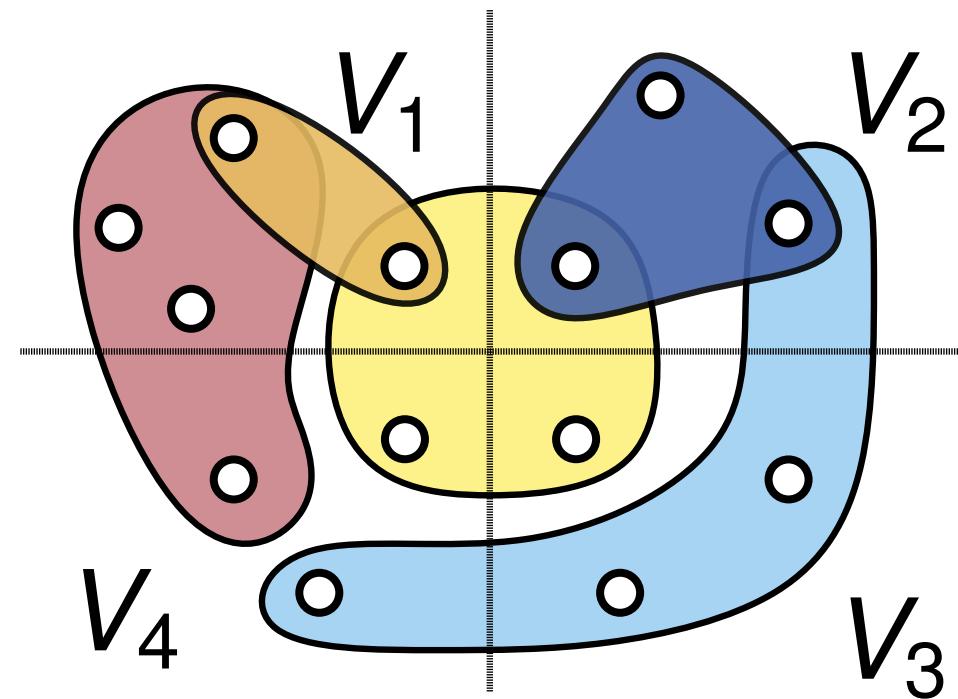
[from SEA'17]

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]

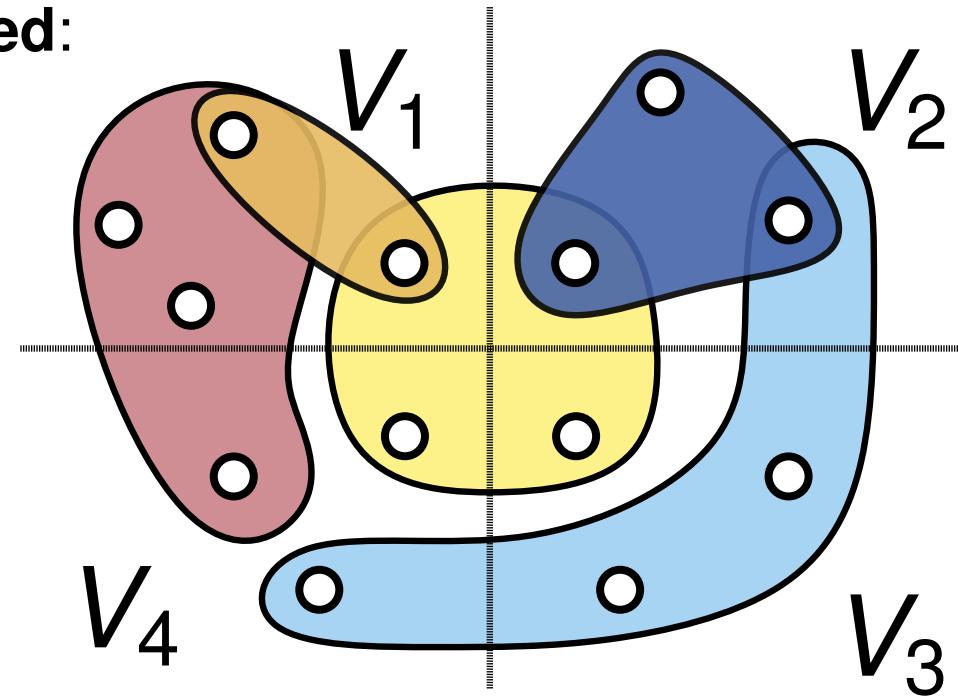
Partition hypergraph $H = (V, E, c, \omega)$ into k non-empty disjoint blocks $\Pi = \{V_1, \dots, V_k\}$ such that:

- 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**:

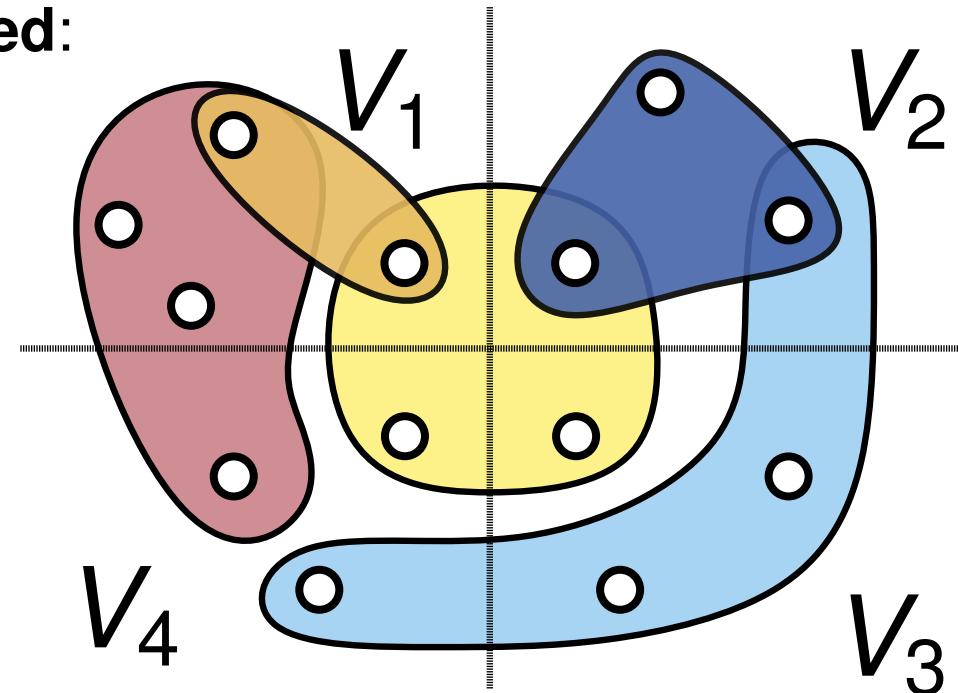
$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance
parameter

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

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

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**:

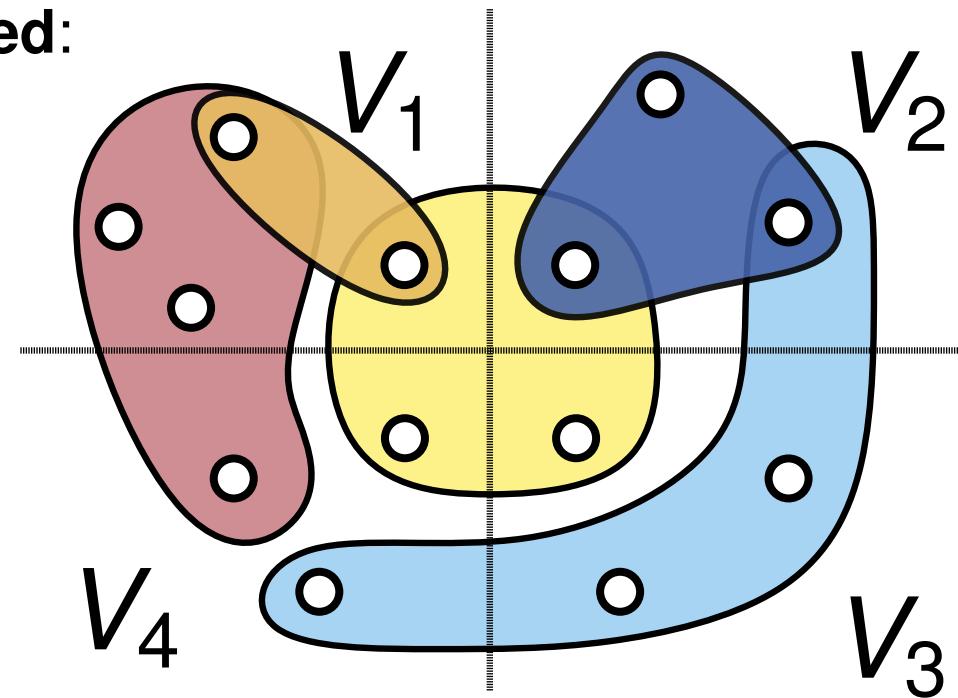
$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

imbalance
parameter

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

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

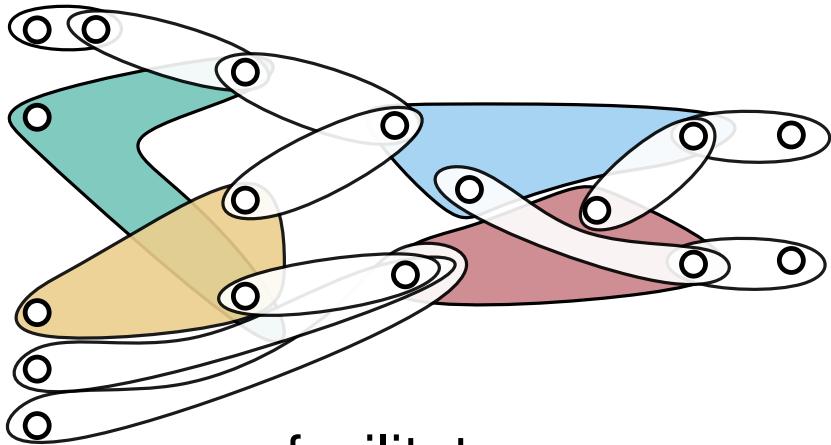
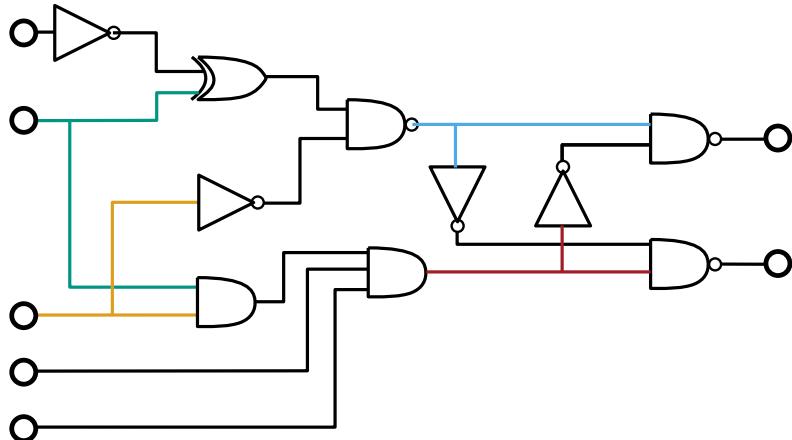
connectivity:
blocks connected by net e



Applications

[from SEA'17]

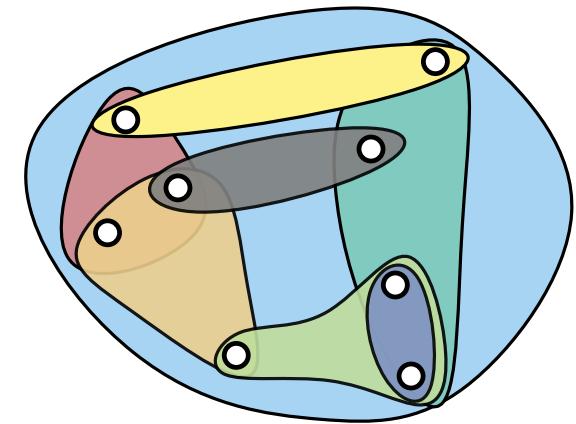
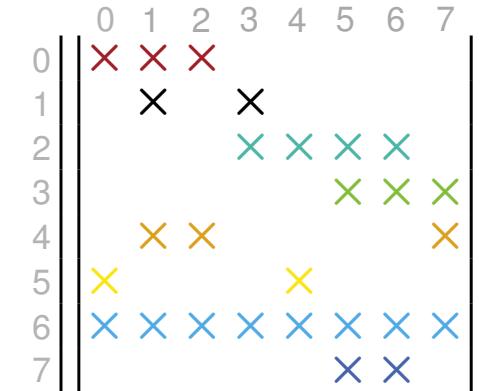
VLSI Design



facilitate
floorplanning & placement

Application
Domain

Scientific Computing



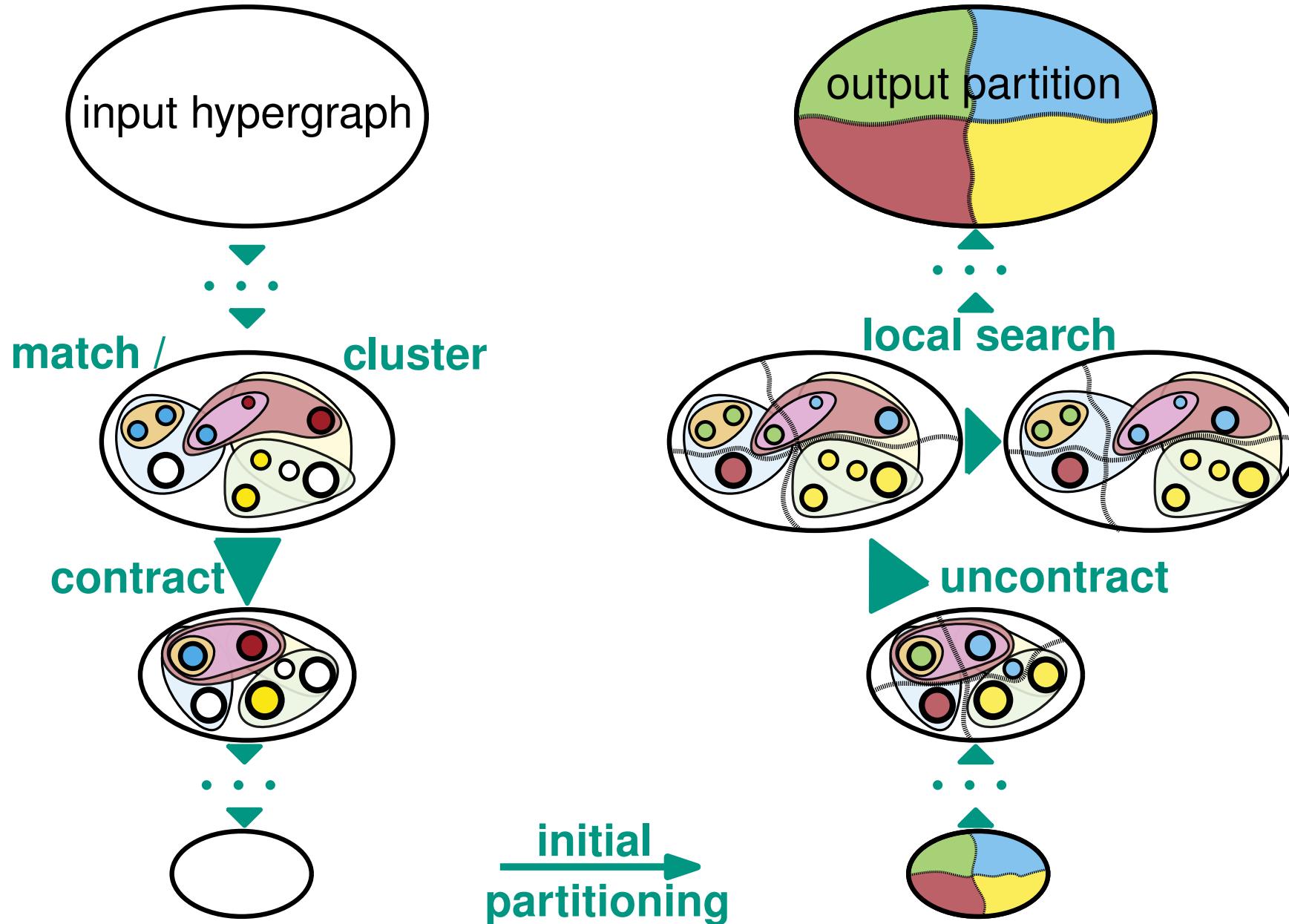
Hypergraph
Model

Goal

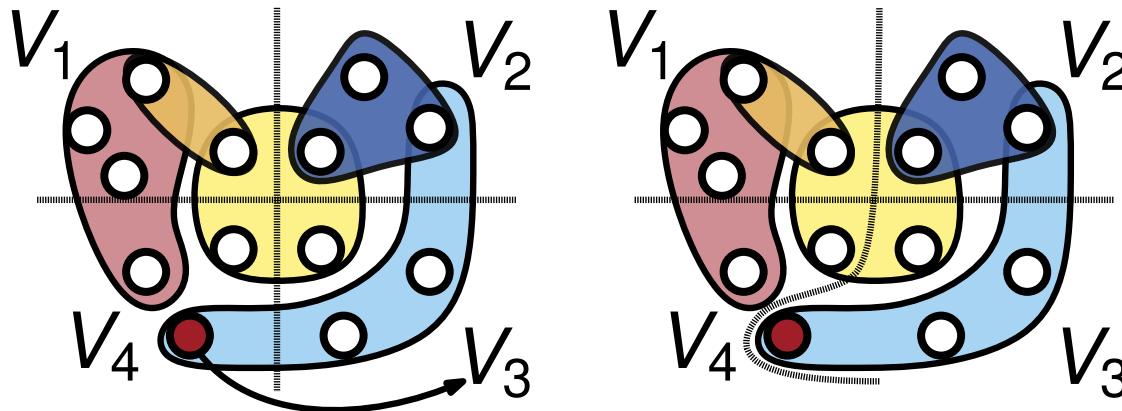
minimize
communication

The Multilevel Framework

[from SEA'17]



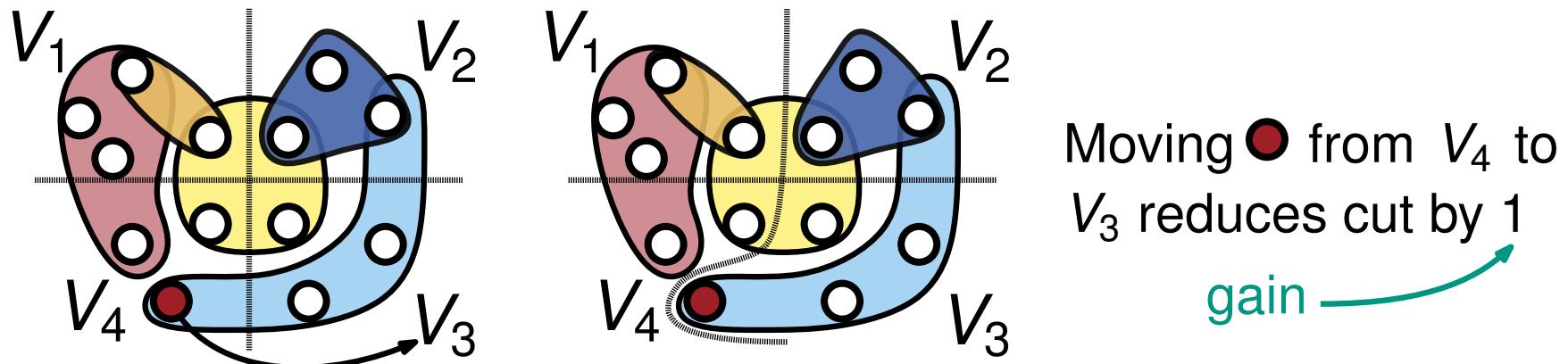
- Move-based heuristic that **greedily** move vertices between blocks based on **local** informations of incident nets



Moving \bullet from V_4 to V_3 reduces cut by 1

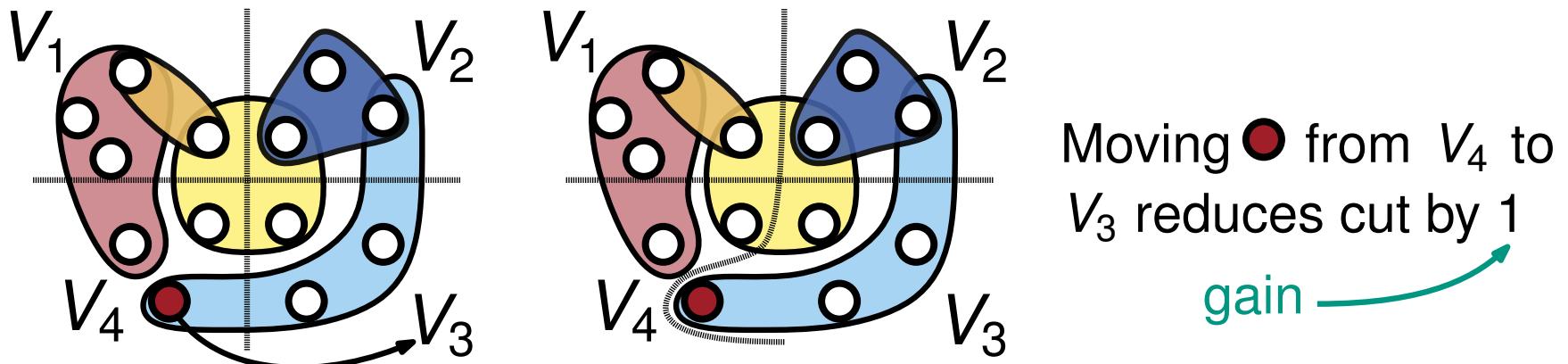
FM Algorithm

- Move-based heuristic that **greedily** move vertices between blocks based on **local** informations of incident nets



FM Algorithm

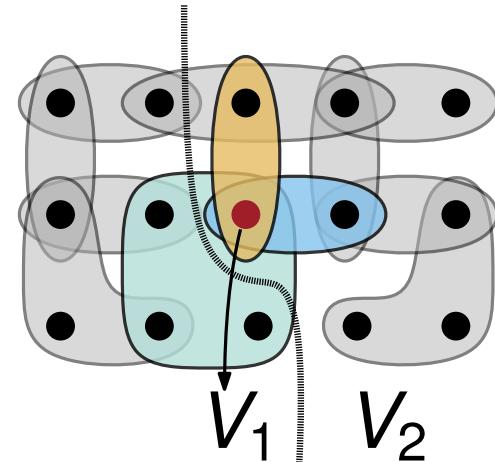
- Move-based heuristic that **greedily** move vertices between blocks based on **local** informations of incident nets



- Performs moves of vertices with **maximum gain** in each step
- All modern hypergraph partitioners implements variations of the *FM* algorithm

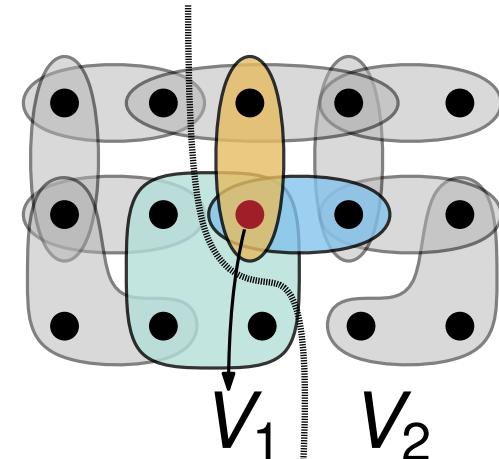
FM Algorithm - Disadvantages

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

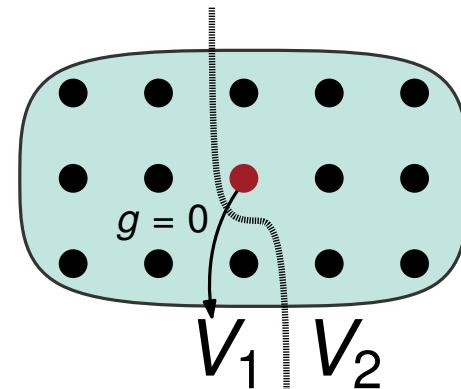


FM Algorithm - Disadvantages

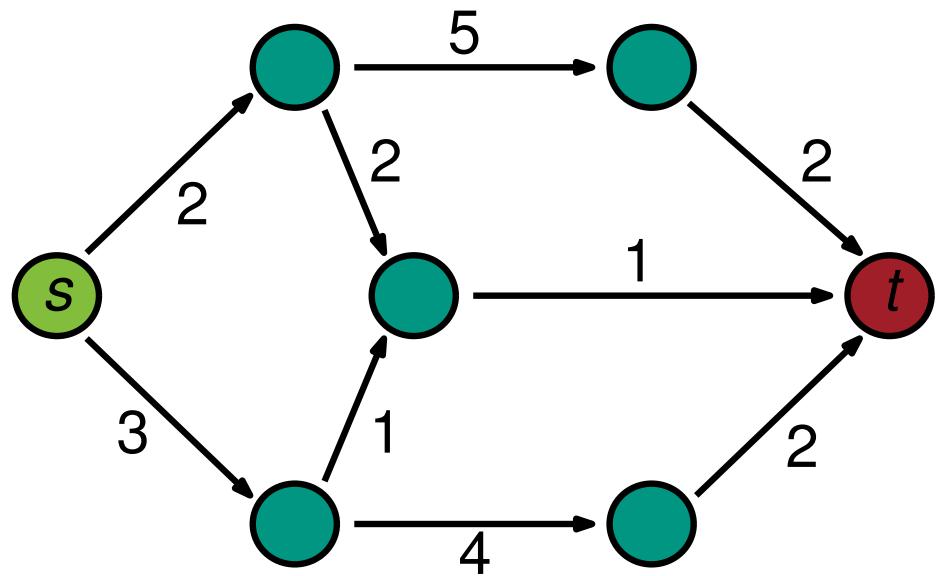
- Only incorporates **local** informations about the problem structure
 - Heavily depends on *initial partition*
 - In multilevel context: Depends on quality of *coarsening*



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

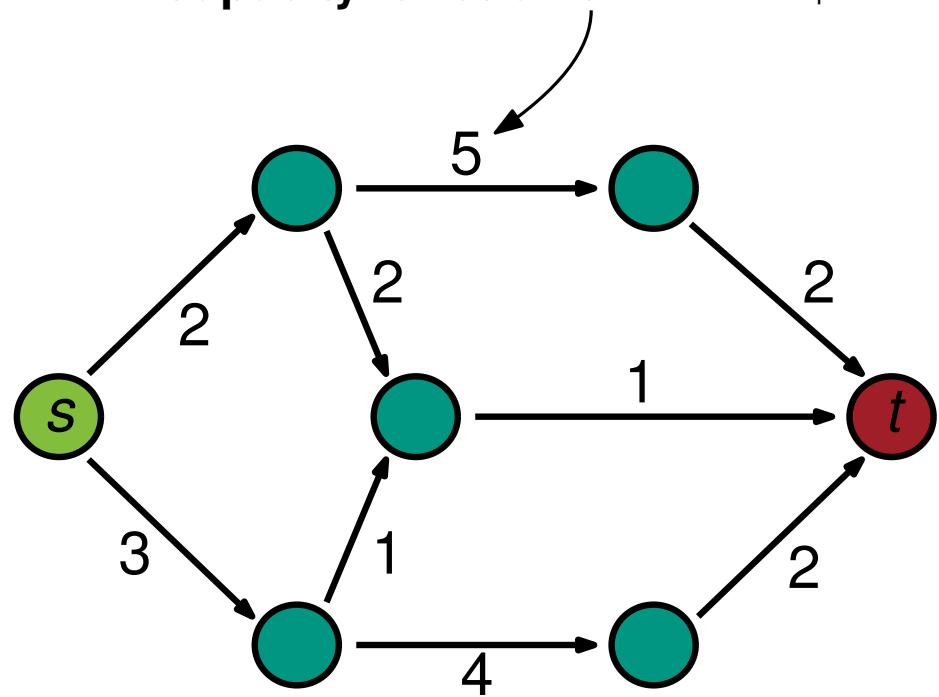


Flows



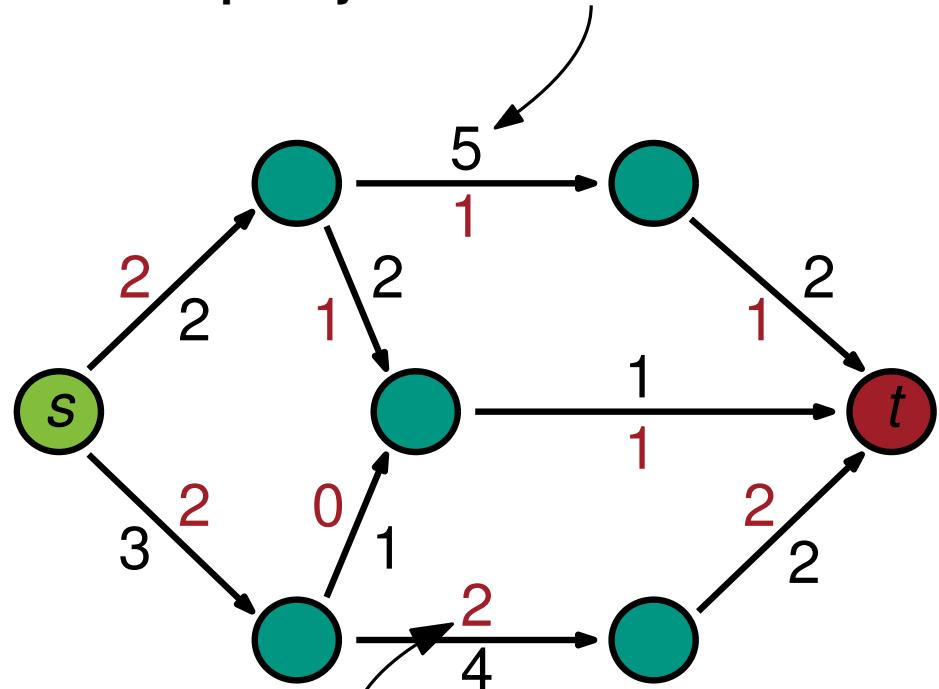
Flows

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



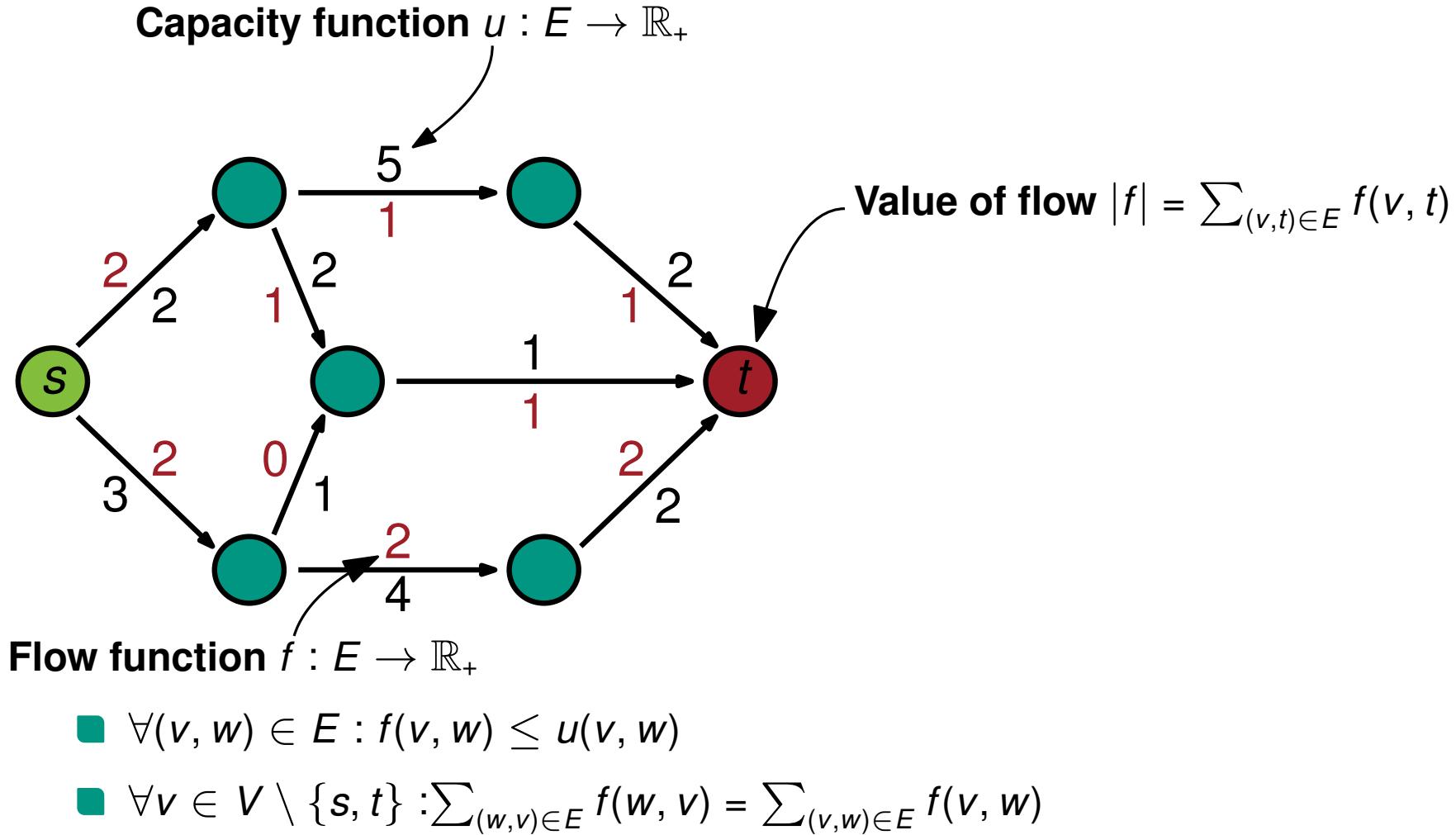
Flows

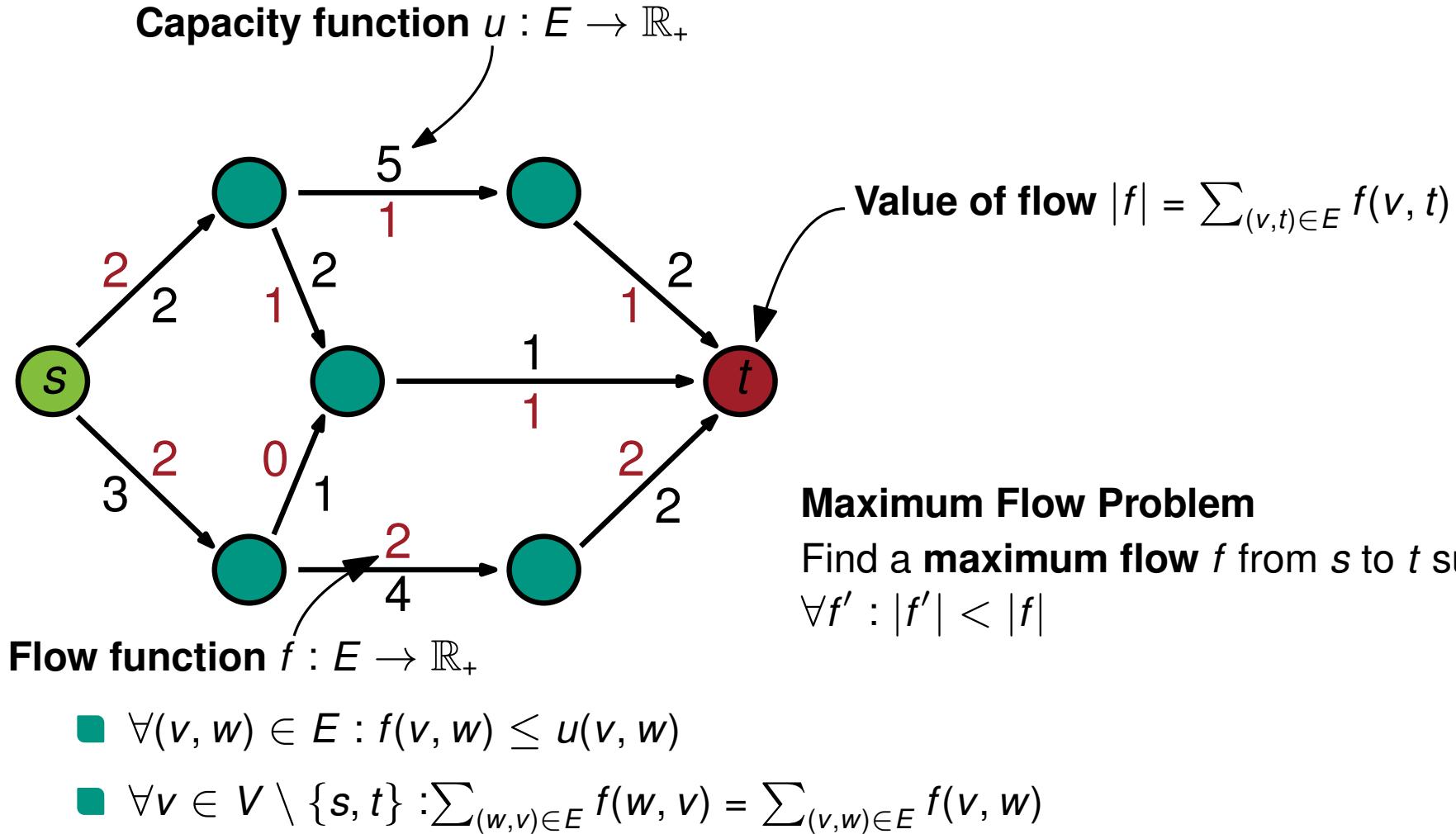
Capacity function $u : E \rightarrow \mathbb{R}_+$



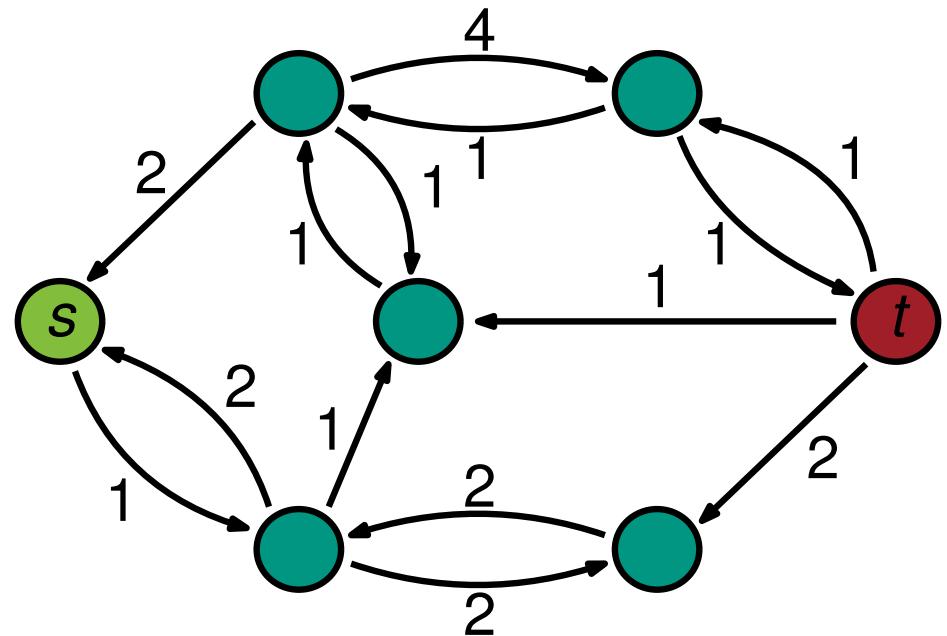
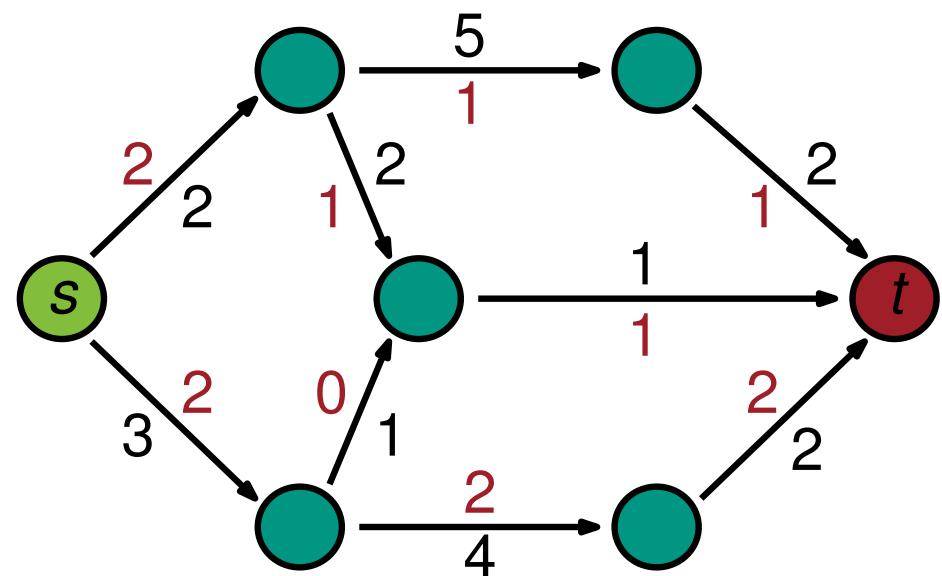
Flow function $f : E \rightarrow \mathbb{R}_+$

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



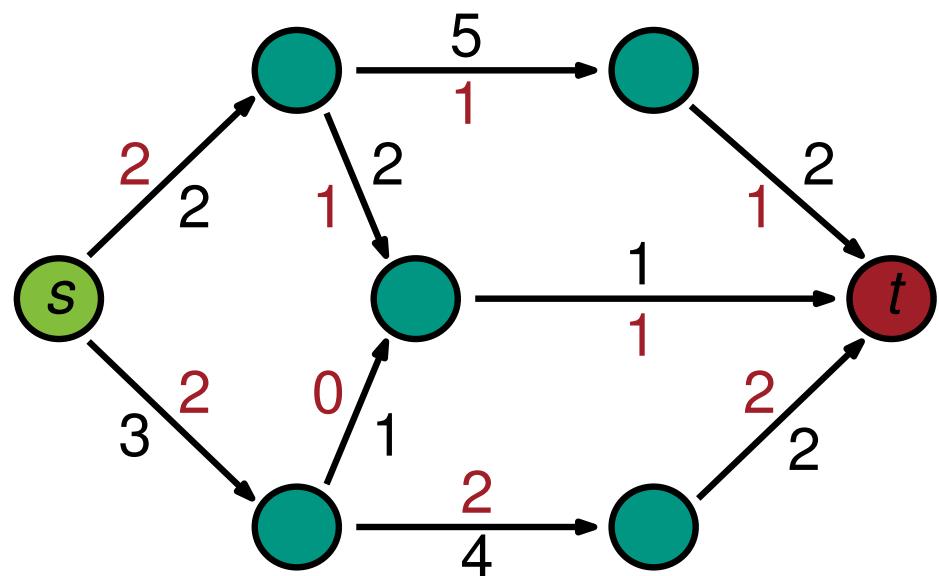


Flows

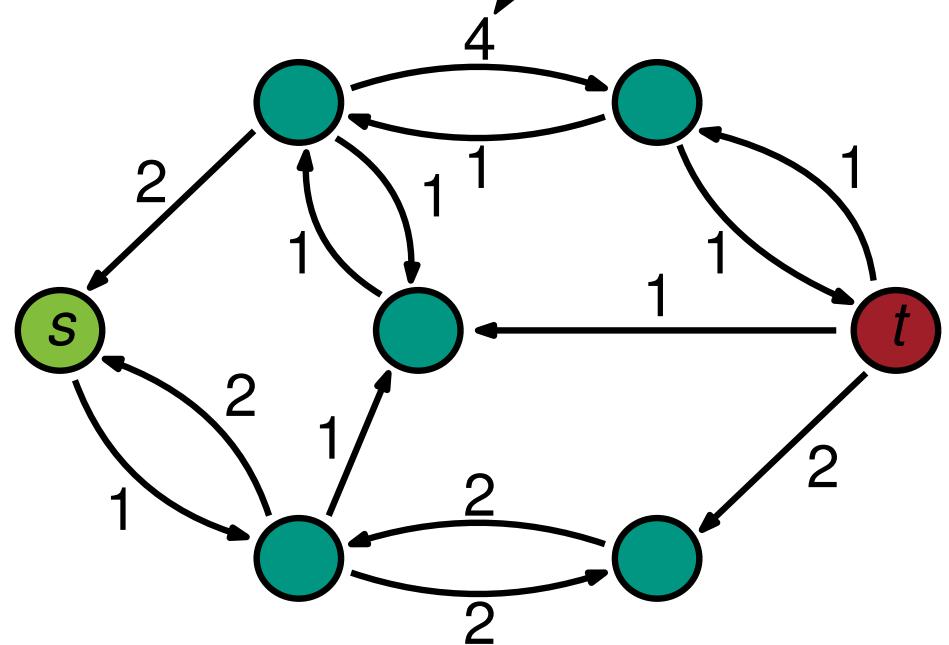


Residual Graph G_f

Flows

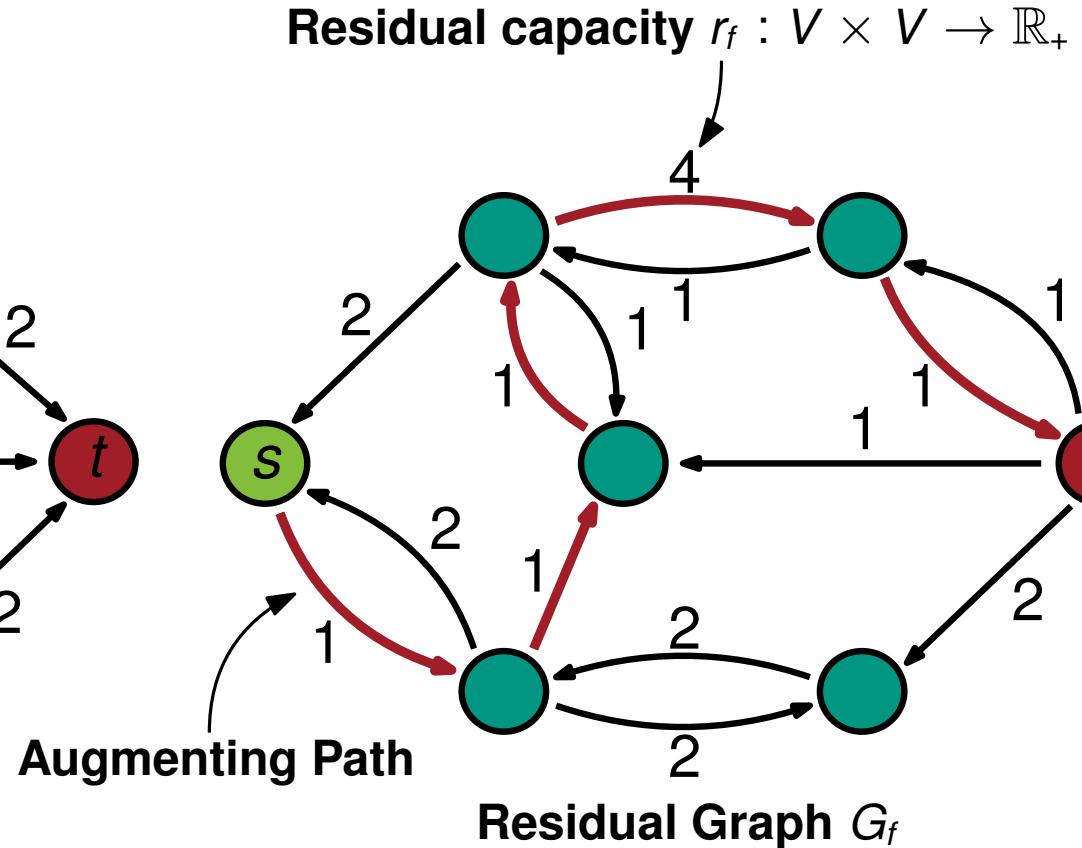
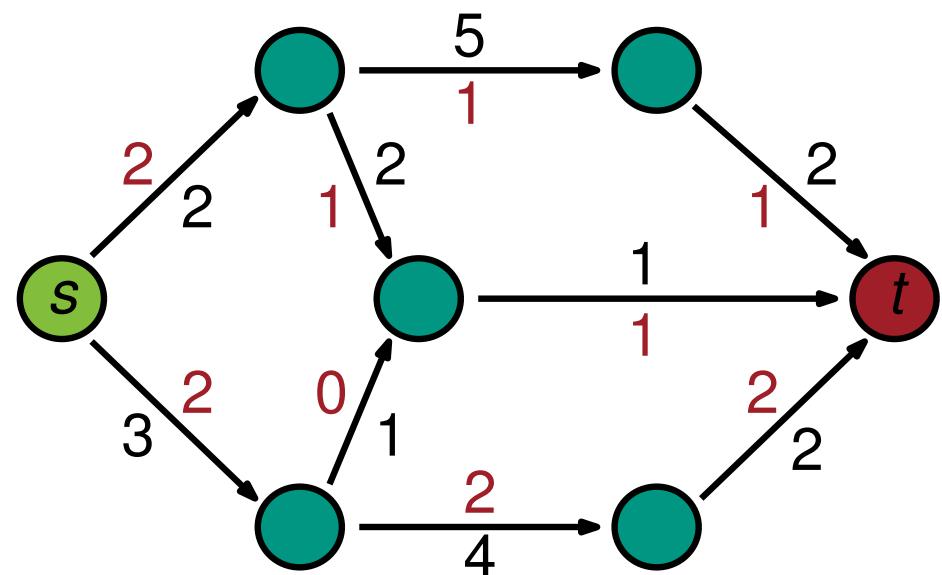


Residual capacity $r_f : V \times V \rightarrow \mathbb{R}_+$



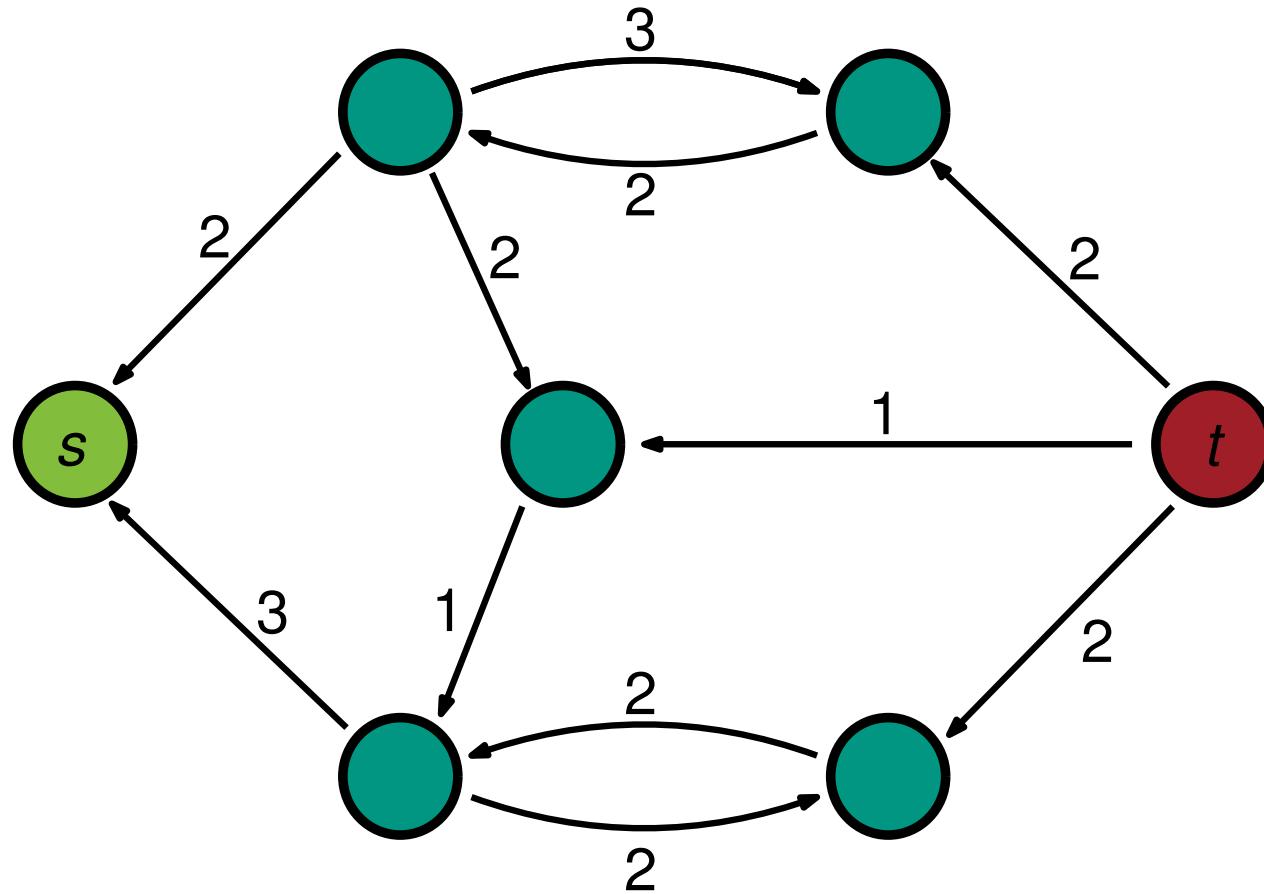
Residual Graph G_f

Flows



Minimum (s, t) -Bipartition

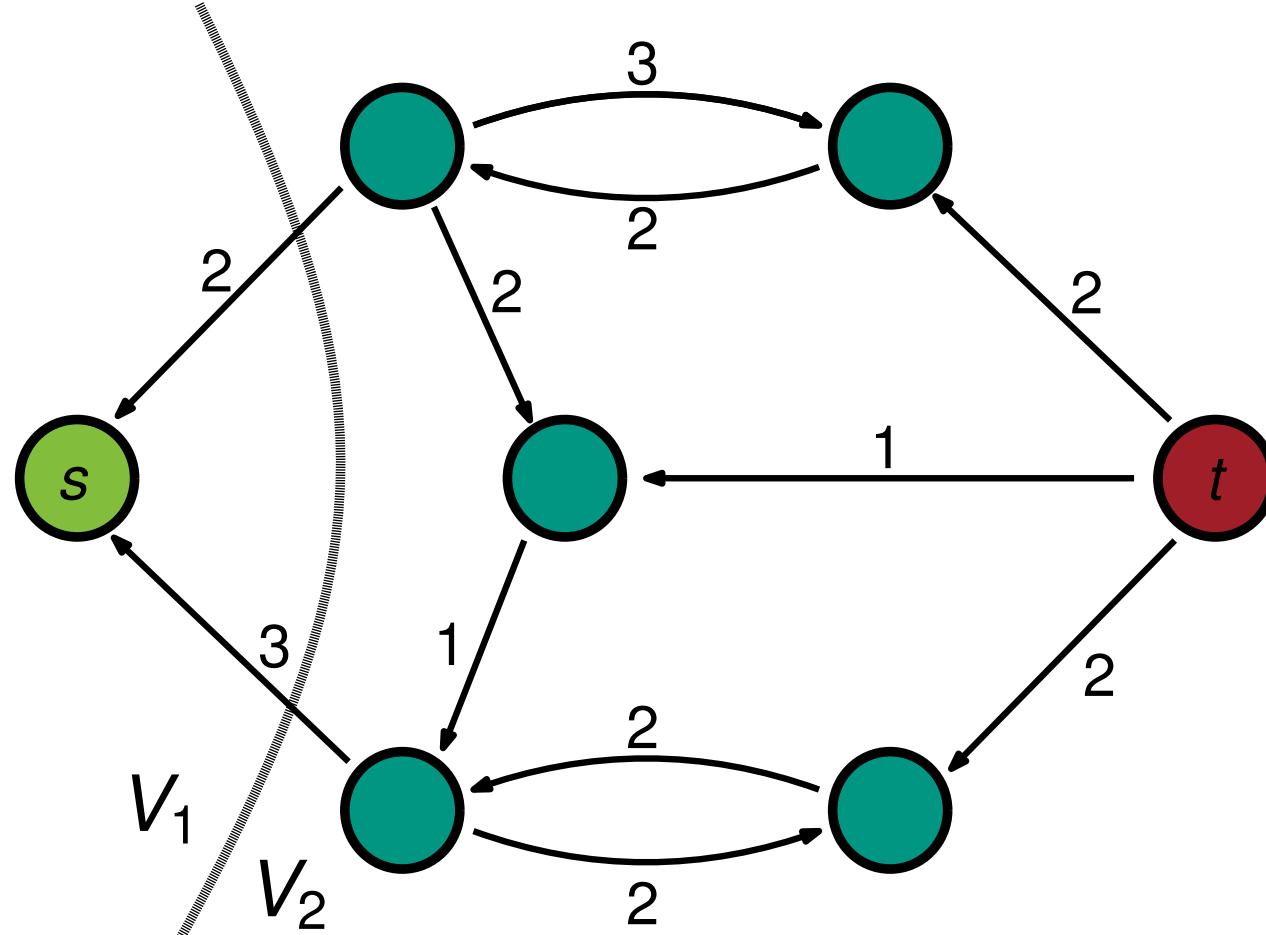
All nodes *reachable* from s are part of V_1 and $V_2 = V \setminus V_1$



Residual Graph G_f of a maximum flow f

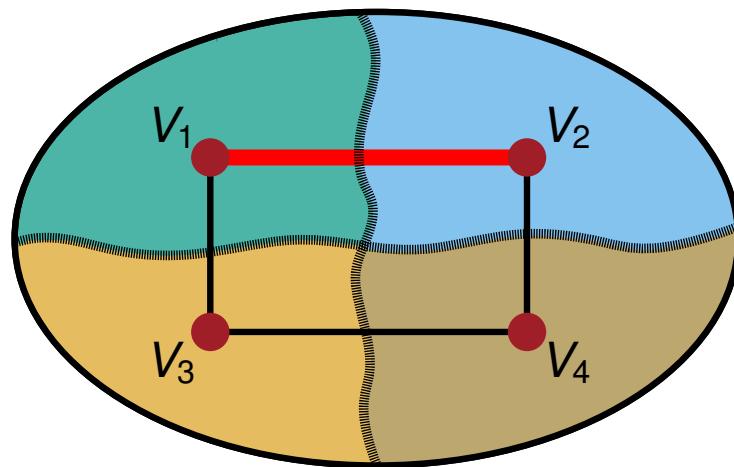
Minimum (s, t) -Bipartition

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

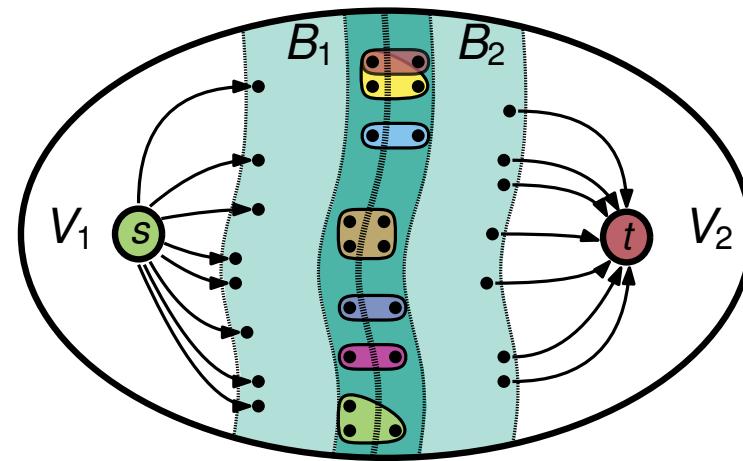


Residual Graph G_f of a maximum flow f

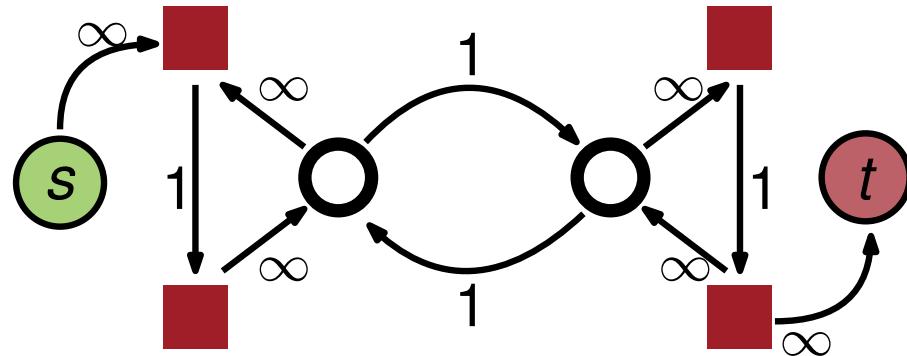
Our Flow-Based Refinement Framework



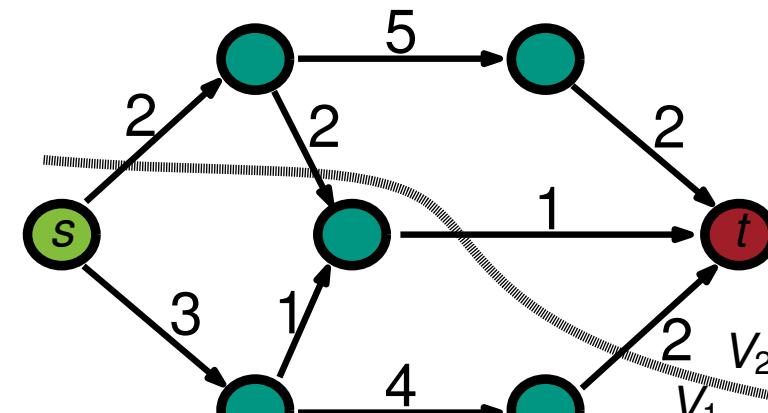
Select two adjacent blocks for refinement



Build Flow Problem

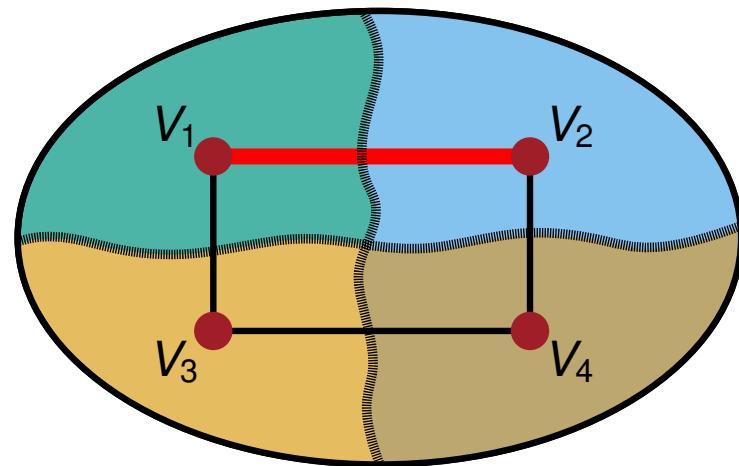


Solve Flow Problem

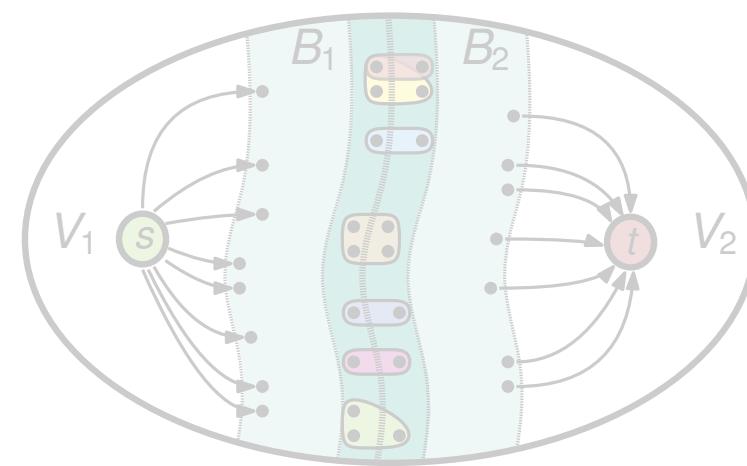


Find feasible minimum cut

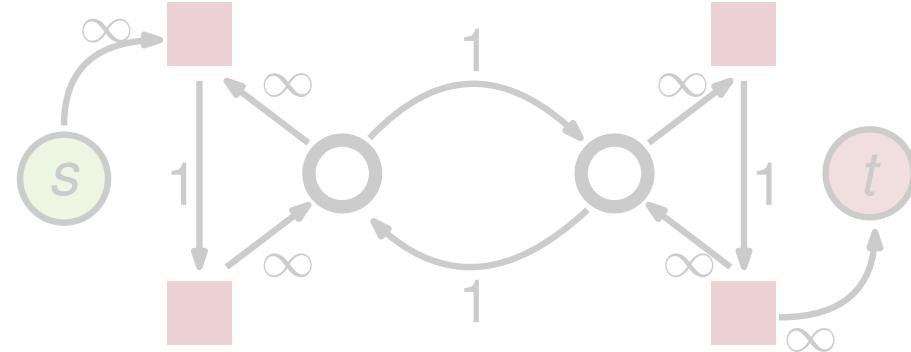
Our Flow-Based Refinement Framework



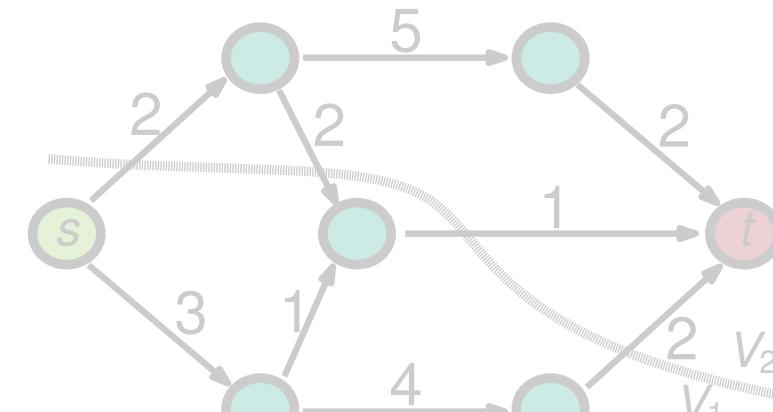
Select two adjacent blocks for refinement



Build Flow Problem

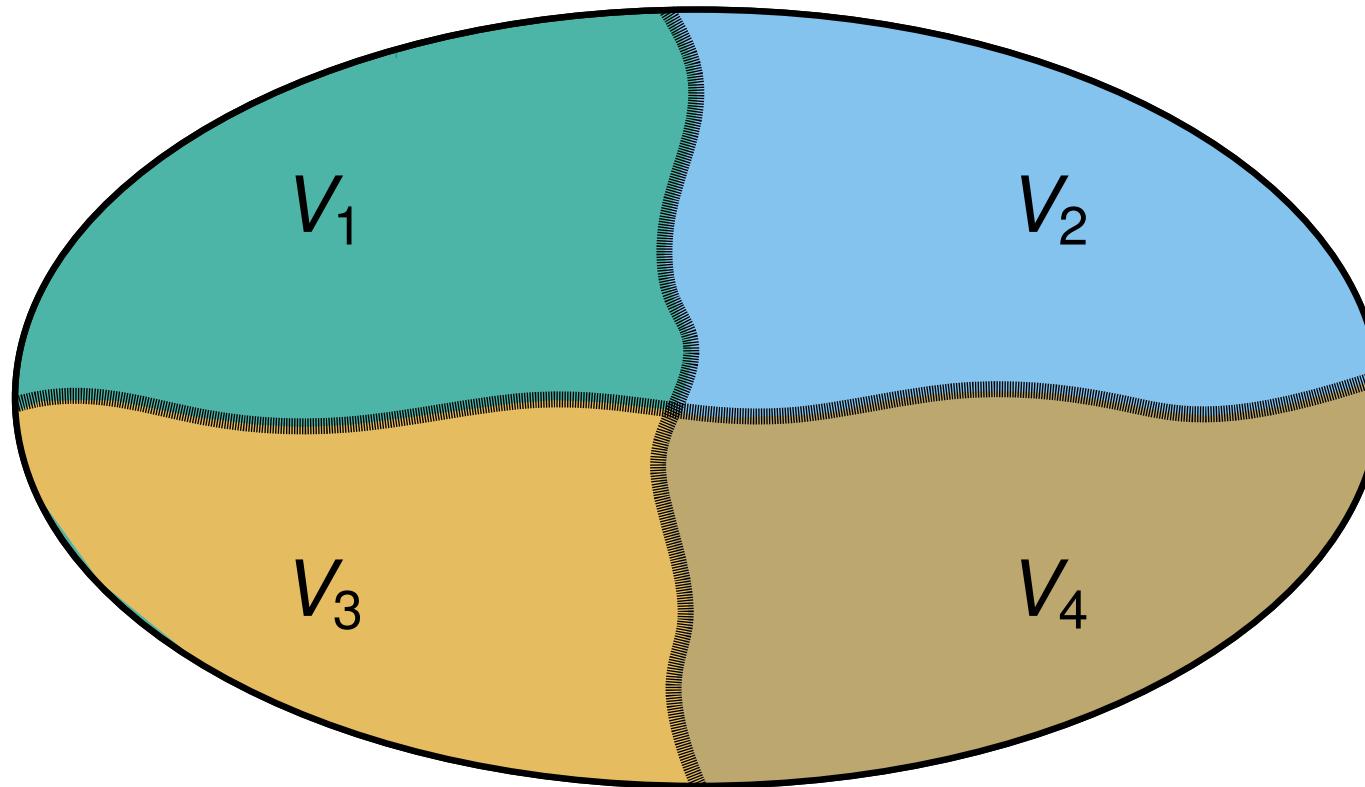


Solve Flow Problem

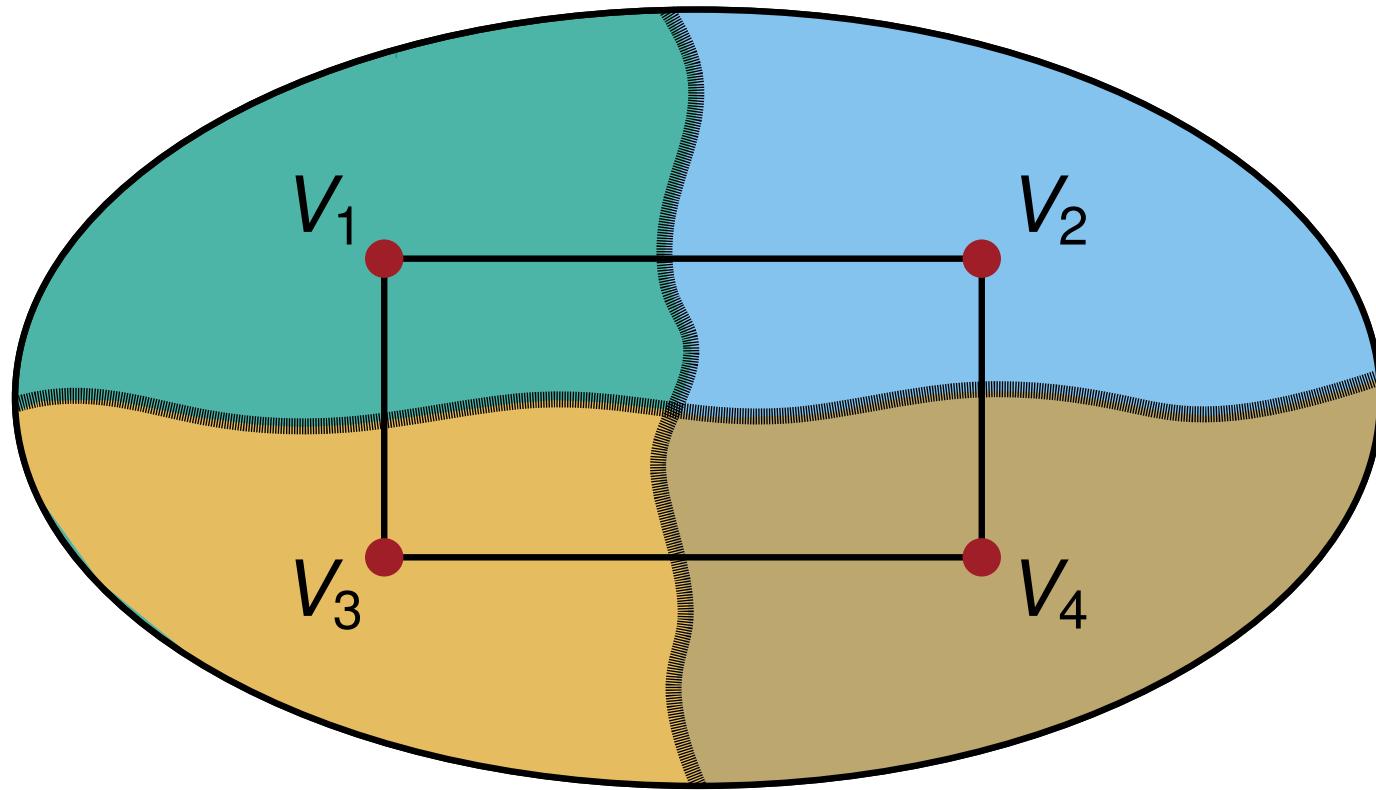


Find feasible minimum cut

Active Block Scheduling



Active Block Scheduling

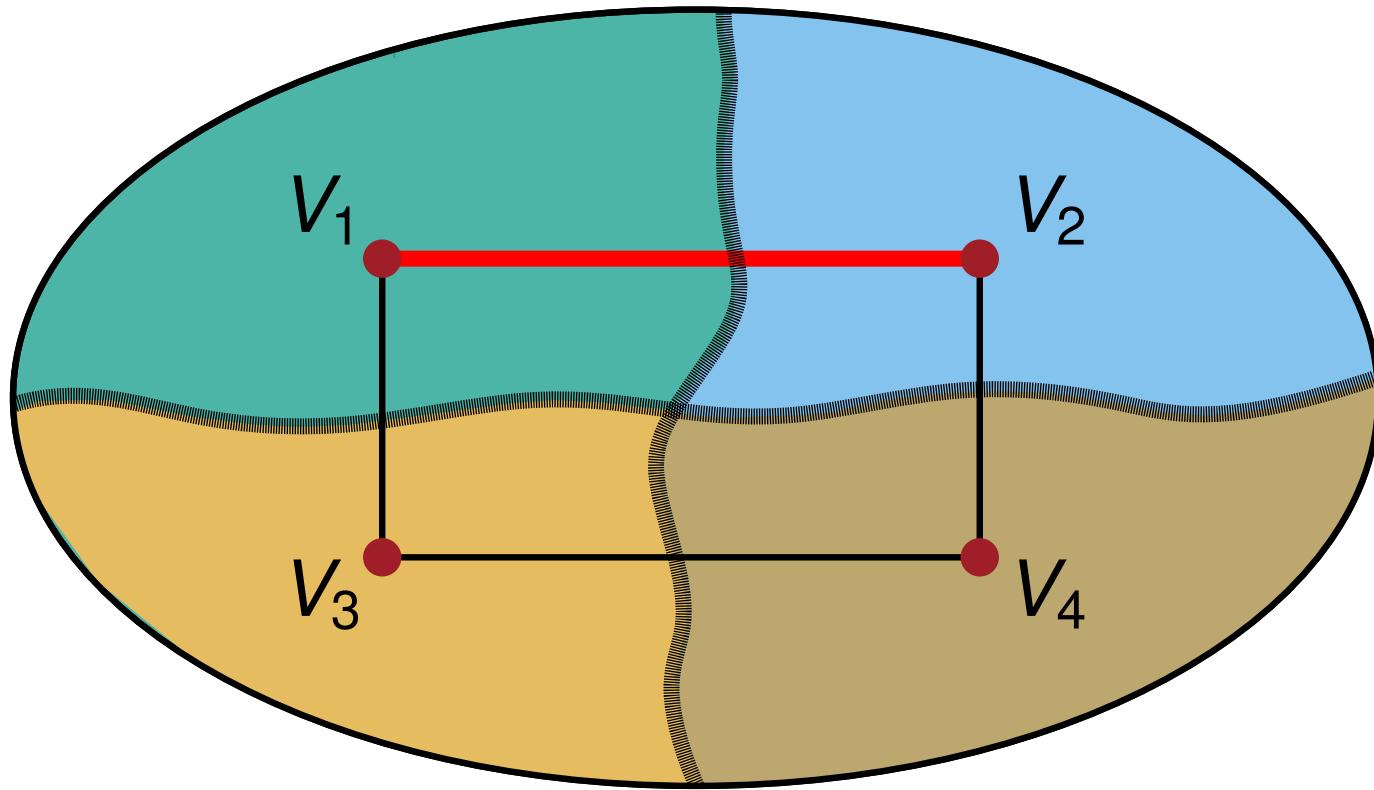


Build Quotient Graph

Active Block Scheduling

Round 1

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

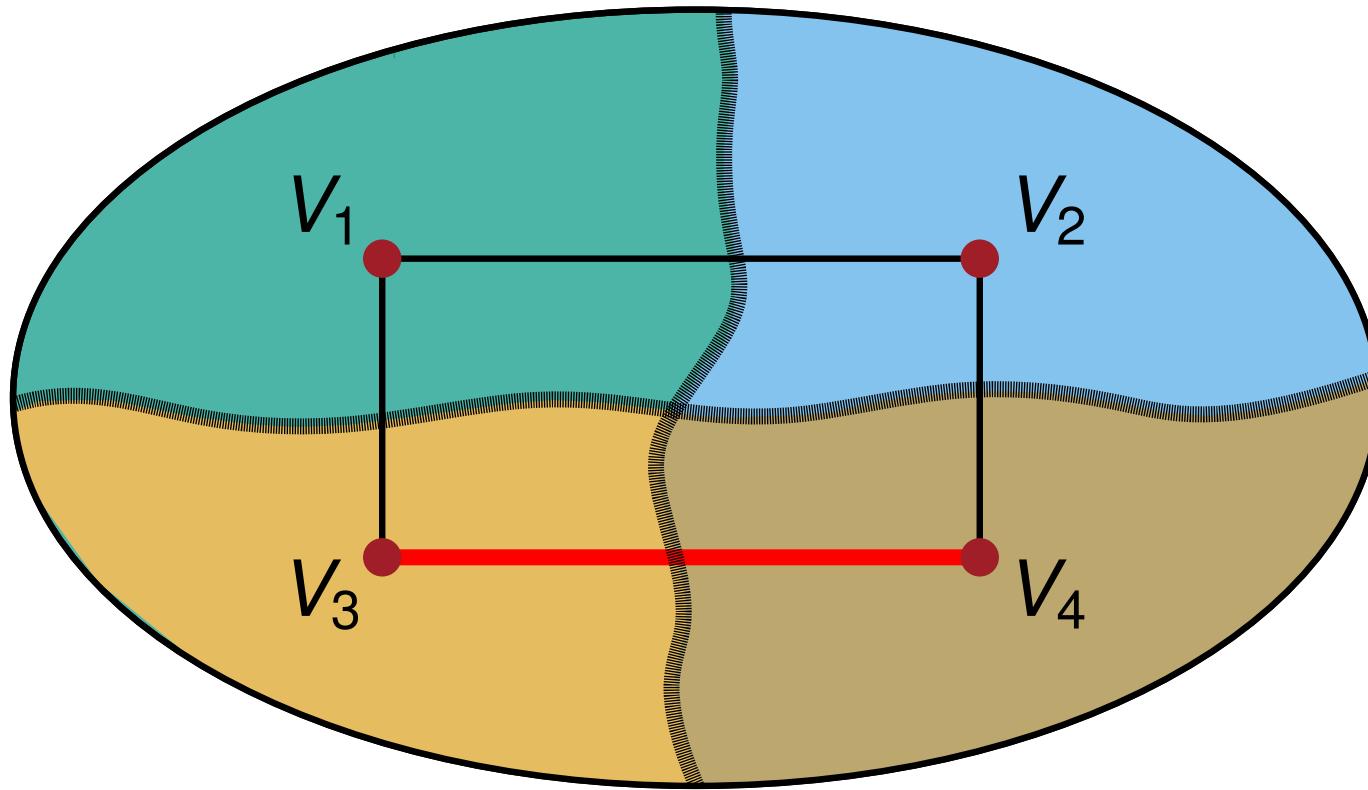


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1

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

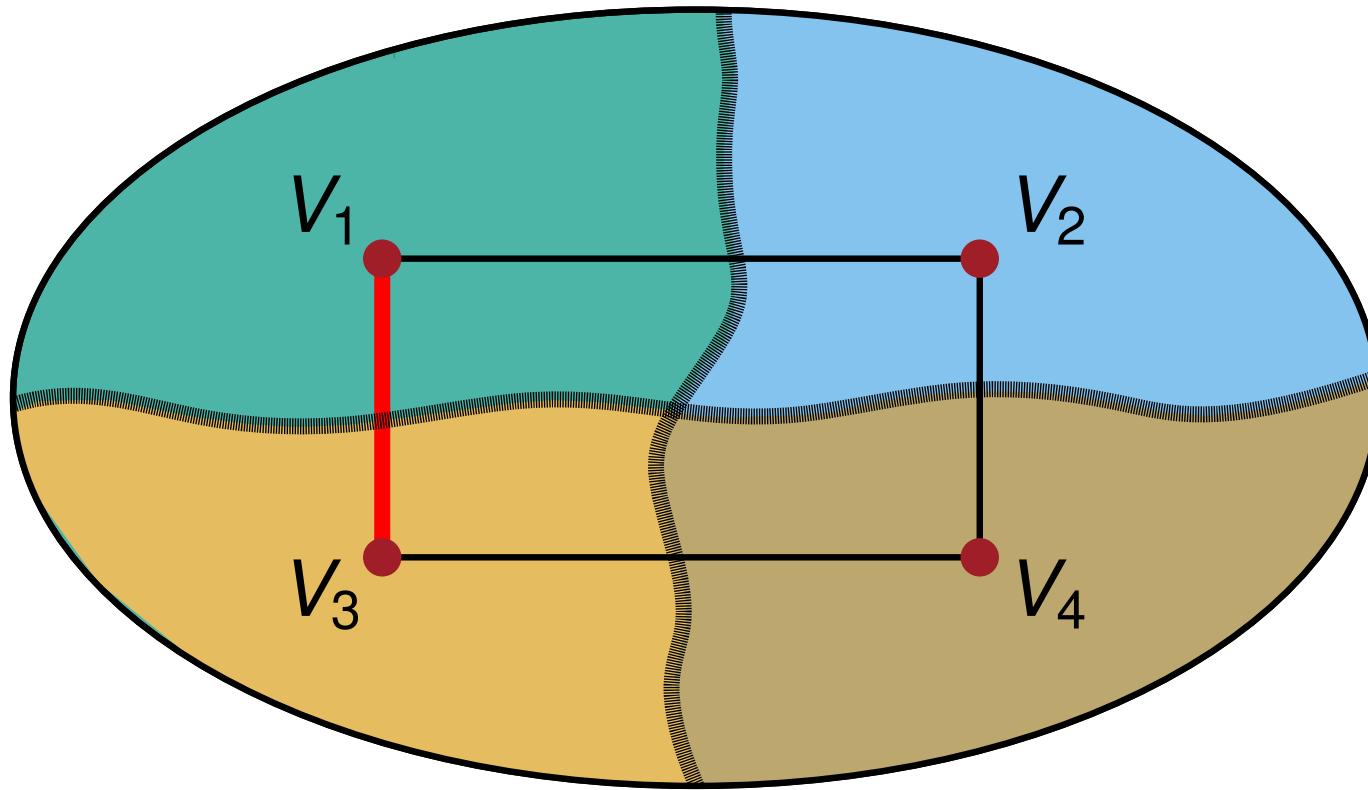


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1

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

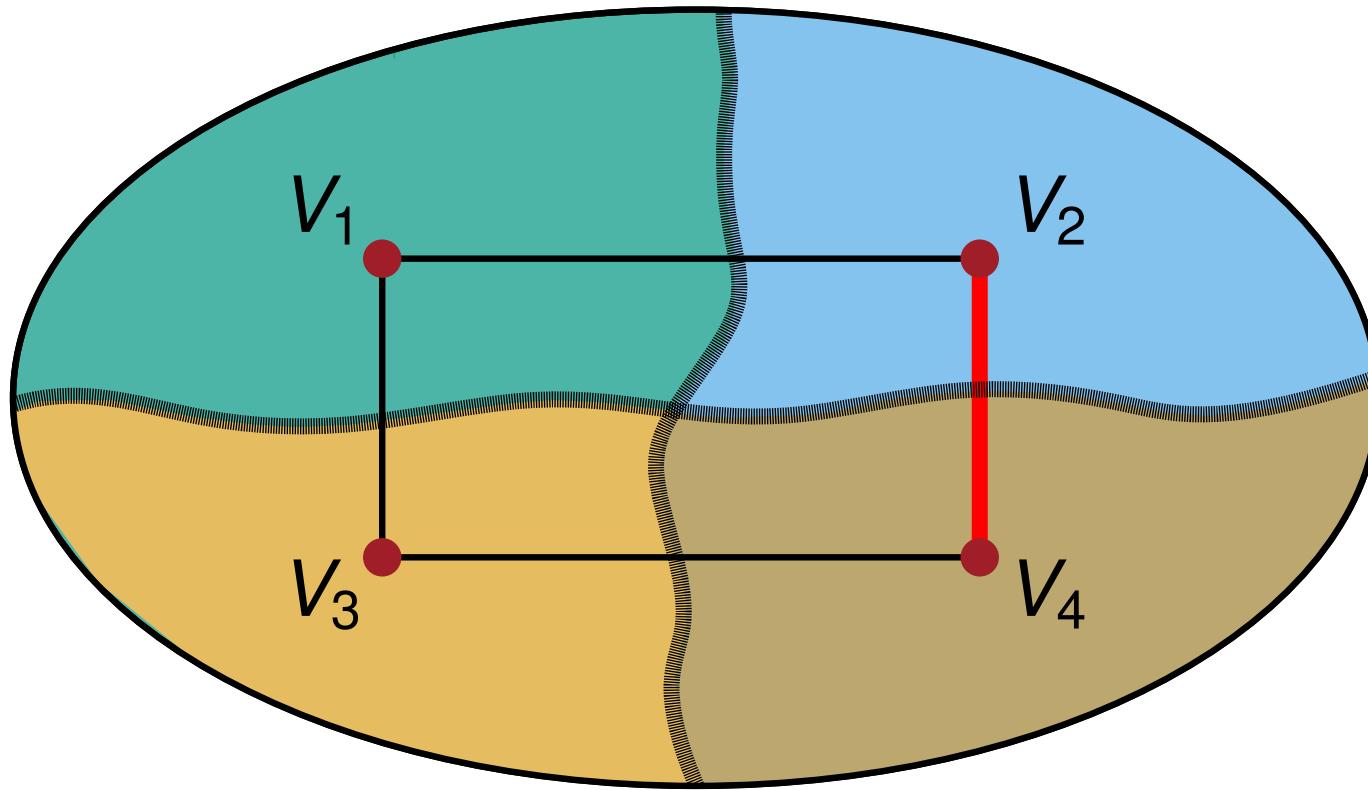


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1

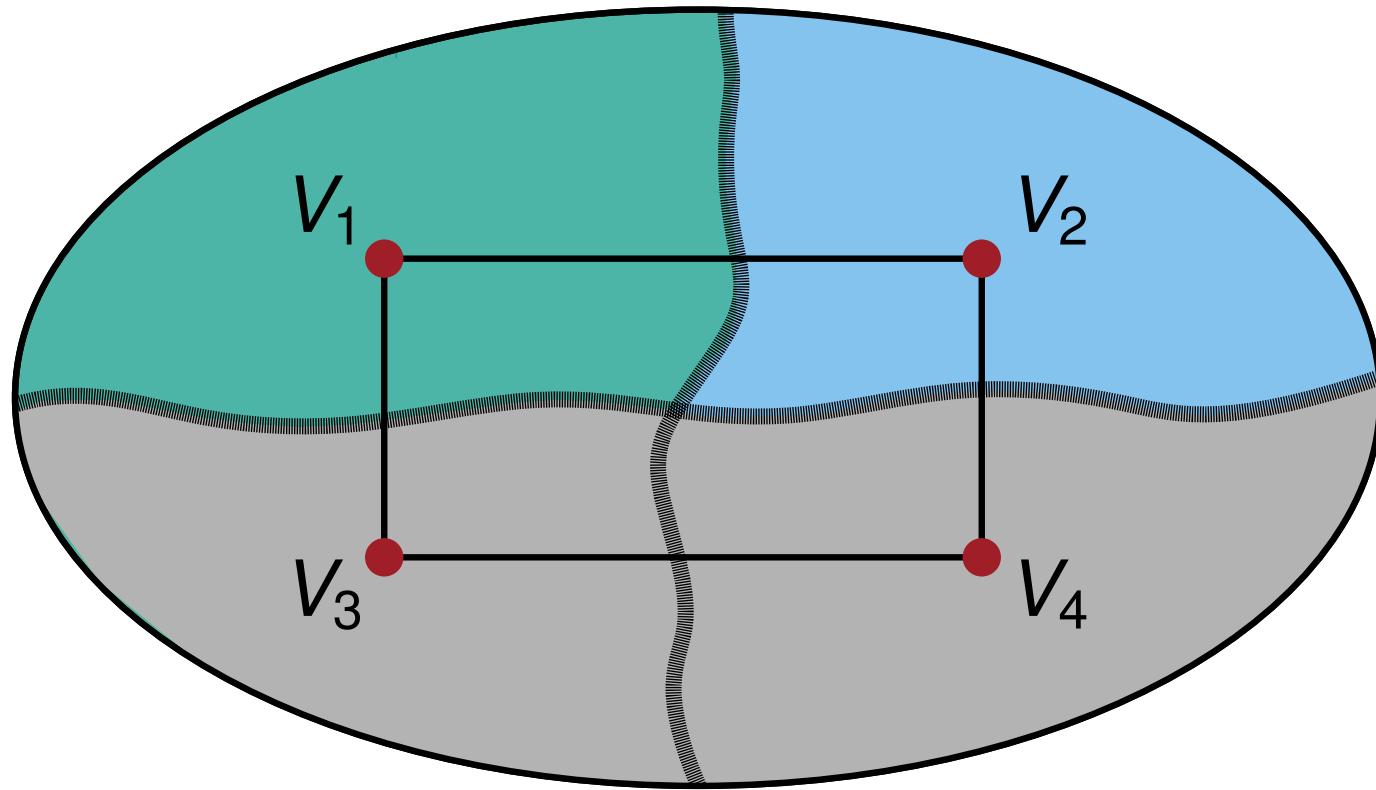
$\text{refine}(V_2, V_4) = \text{No Improvement!}$



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

Active Block Scheduling

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

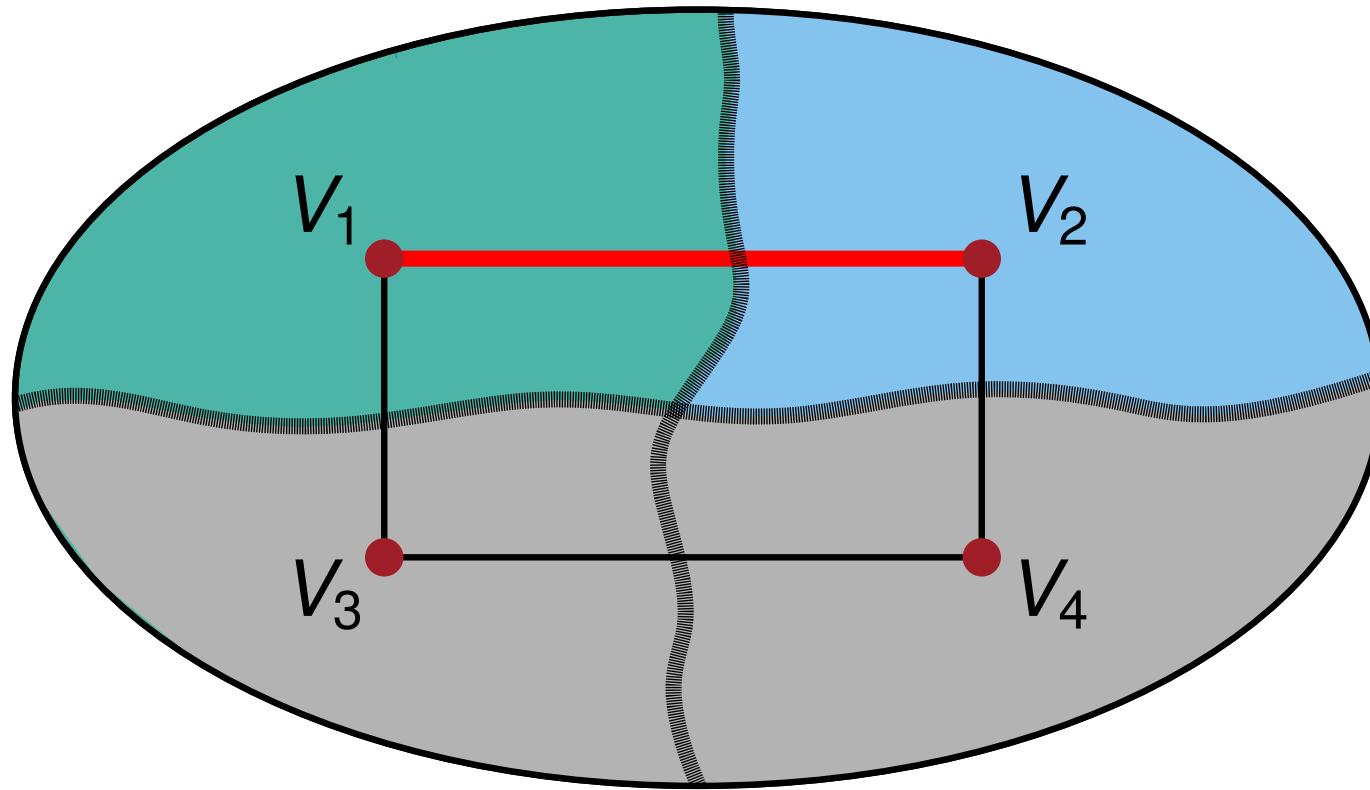


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 2

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

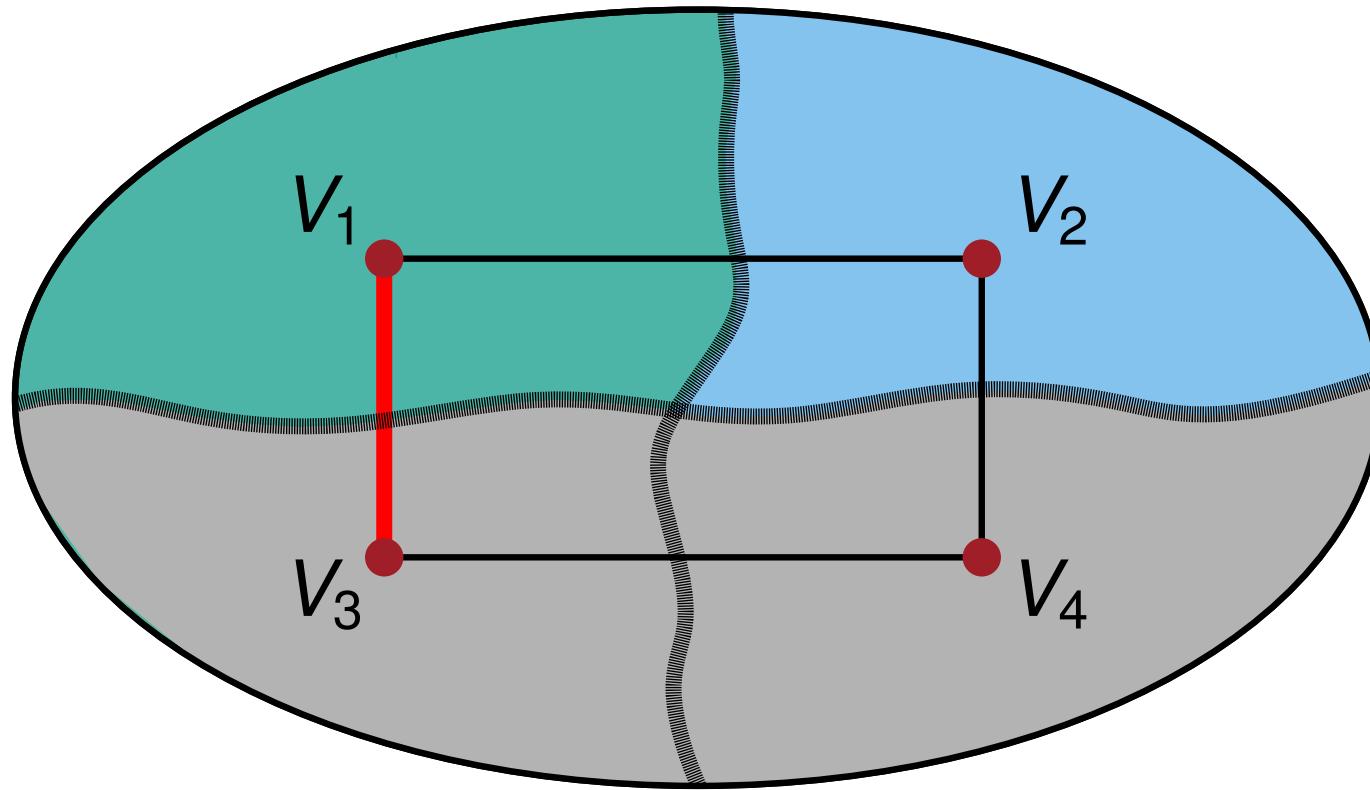


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 2

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

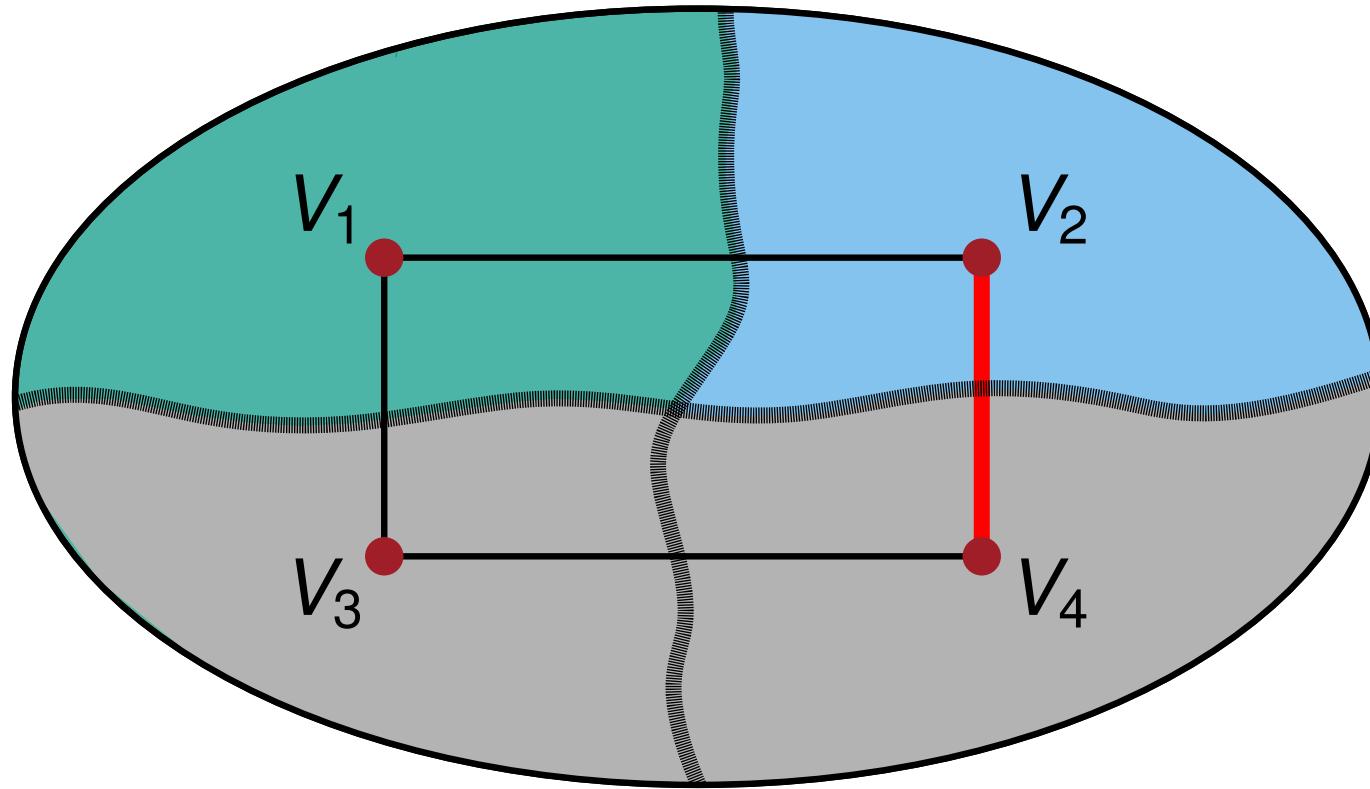


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 2

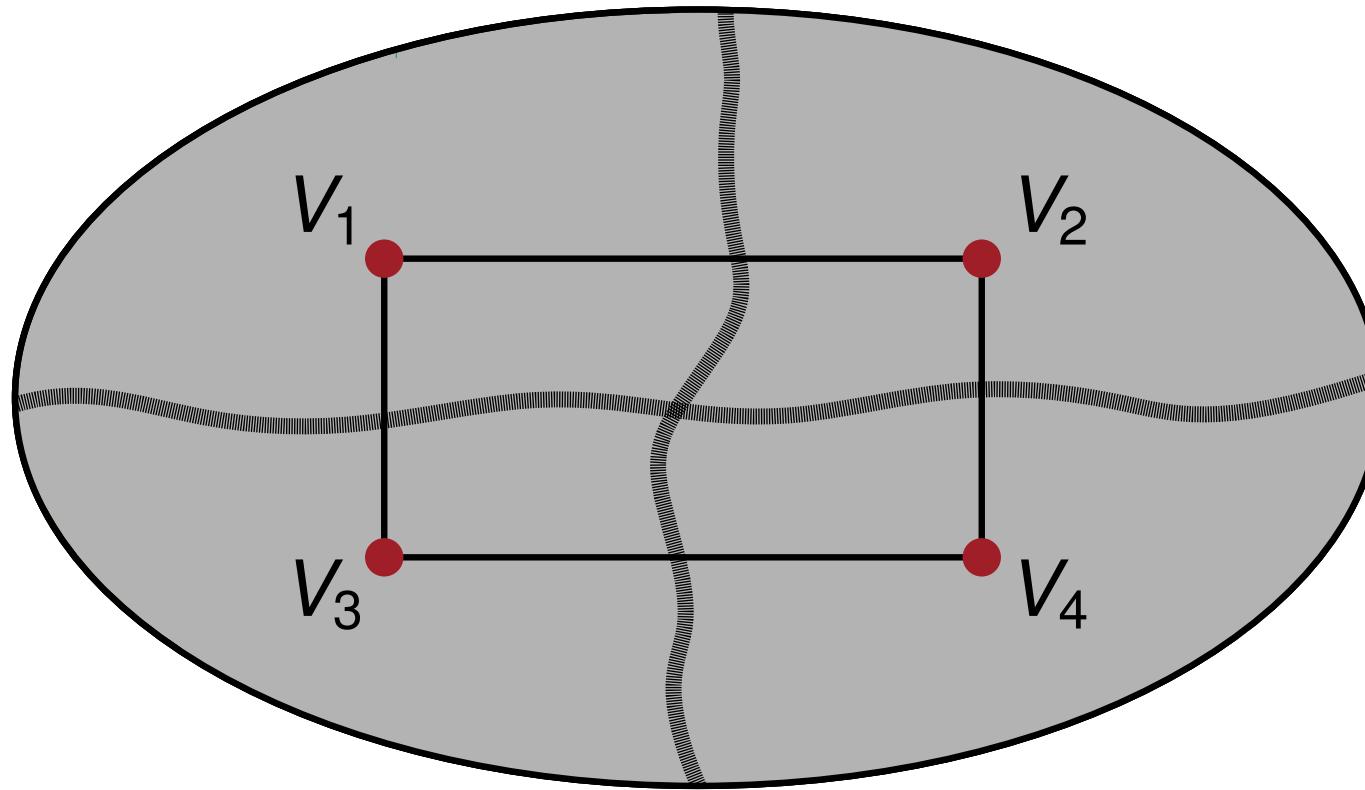
$\text{refine}(V_2, V_4) = \text{No Improvement!}$



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

Active Block Scheduling

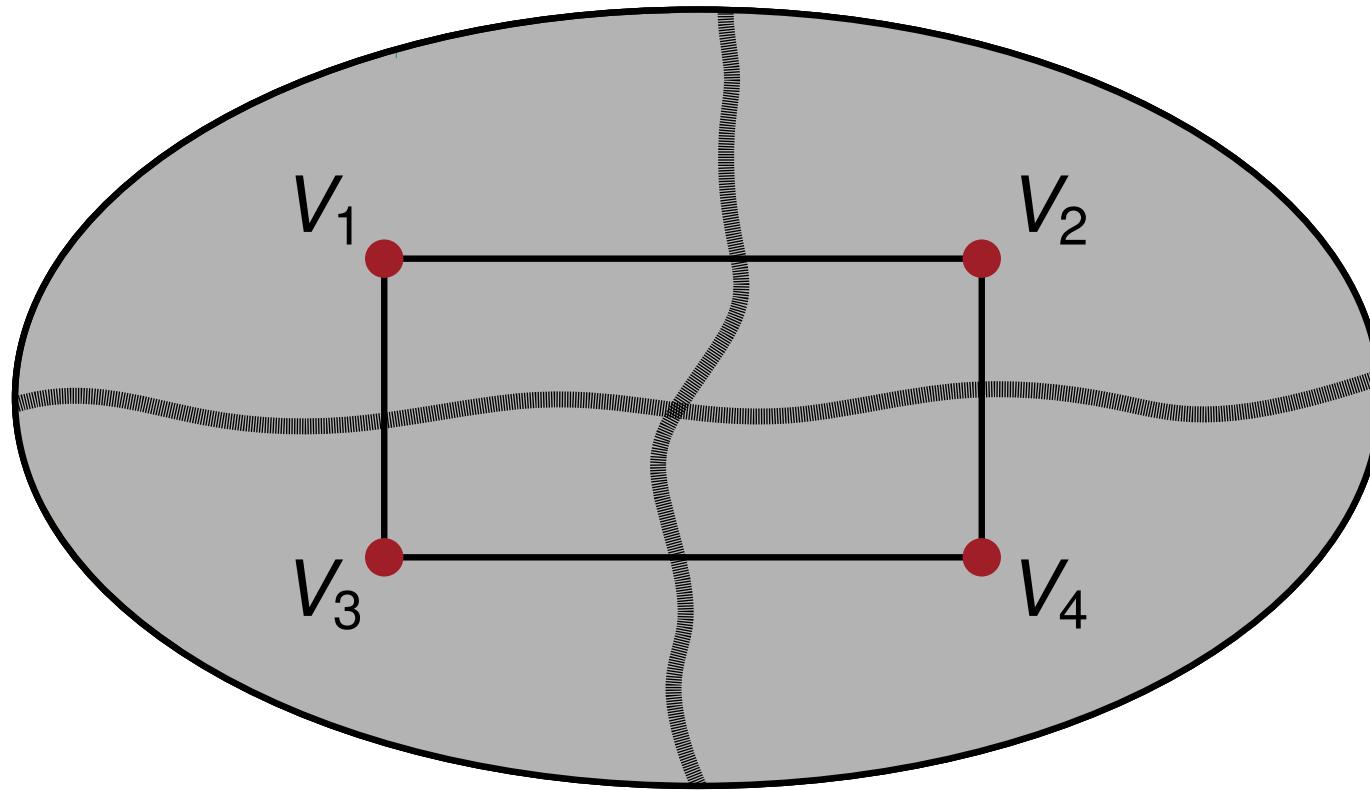
Round 2 Boundary did not change \Rightarrow Mark block as **inactive**



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

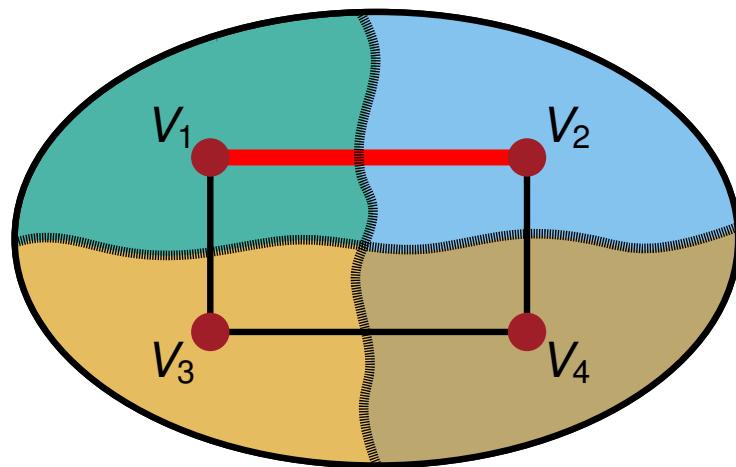
Active Block Scheduling

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

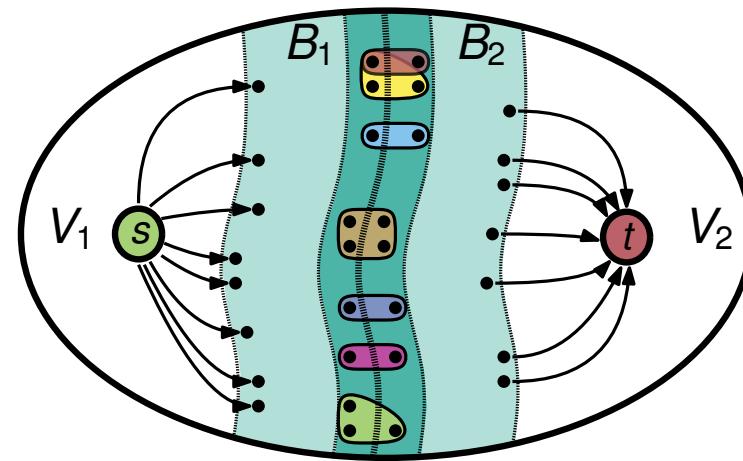


Using 2-way refinement algorithm for **active** blocks of the quotient graph

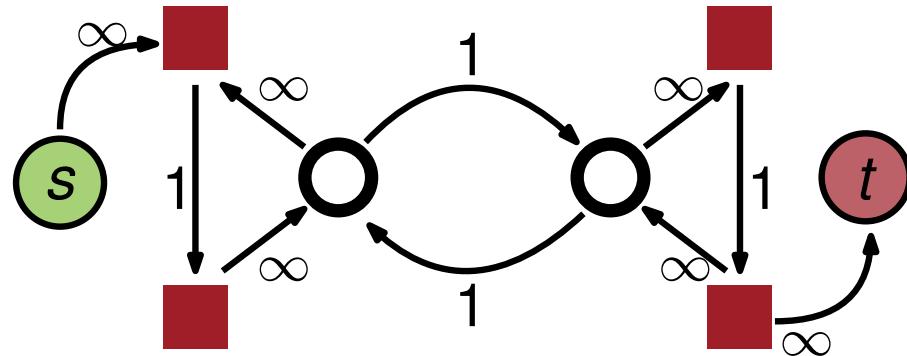
Our Flow-Based Refinement Framework



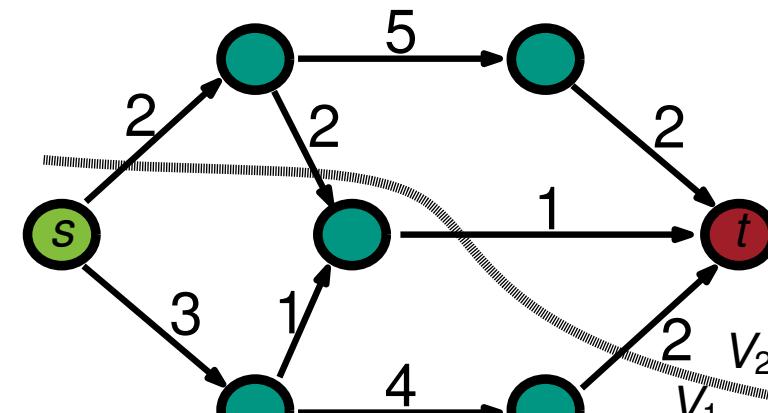
Select two adjacent blocks for refinement



Build Flow Problem

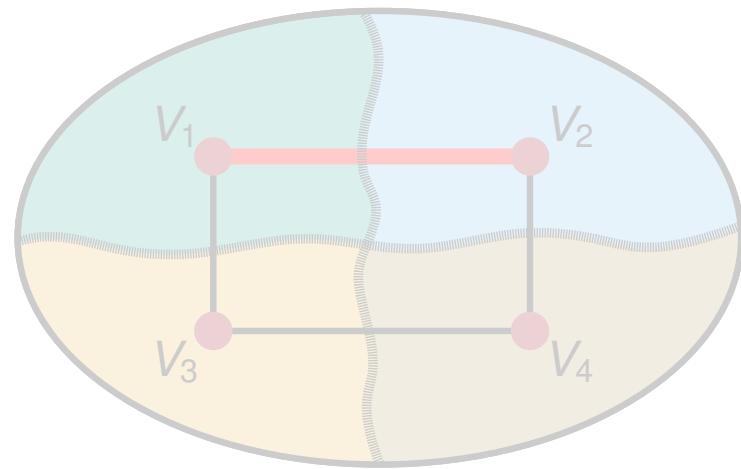


Solve Flow Problem

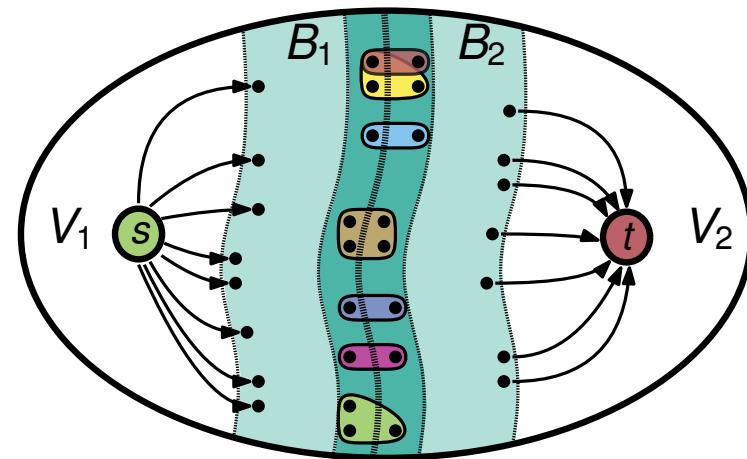


Find feasible minimum cut

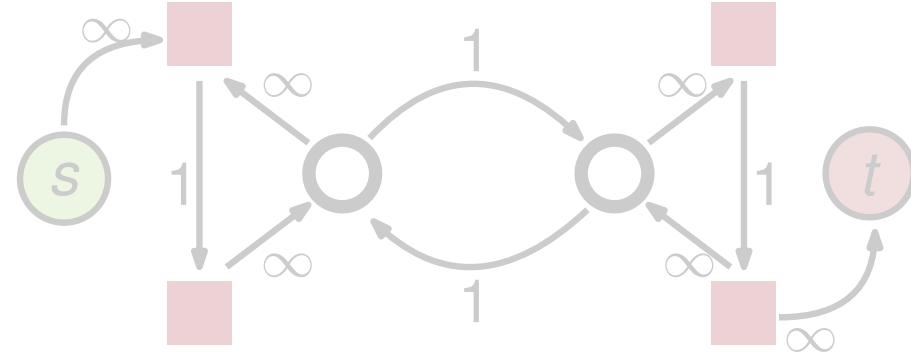
Our Flow-Based Refinement Framework



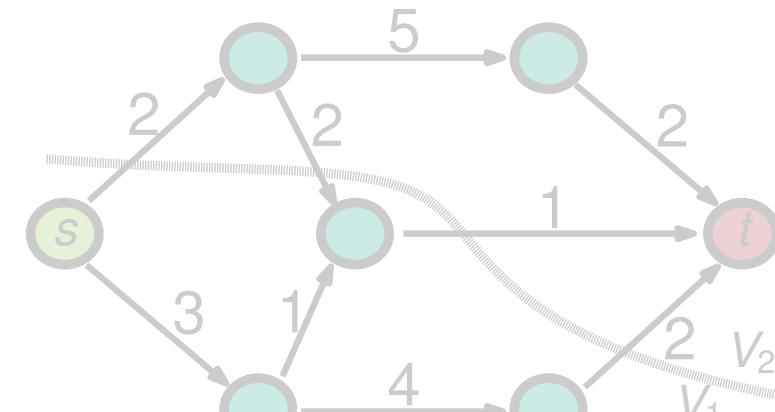
Select two adjacent blocks for refinement



Build Flow Problem

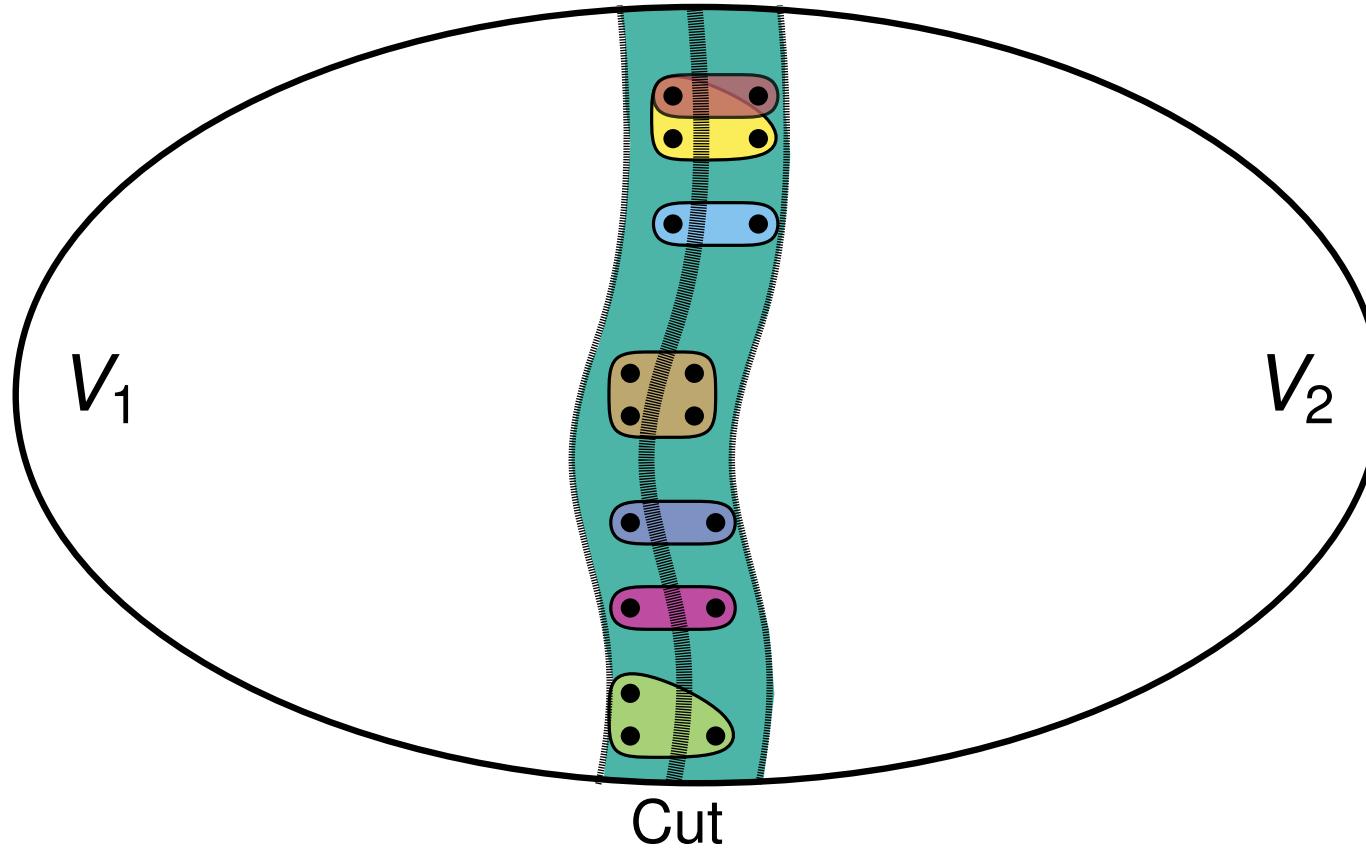


Solve Flow Problem

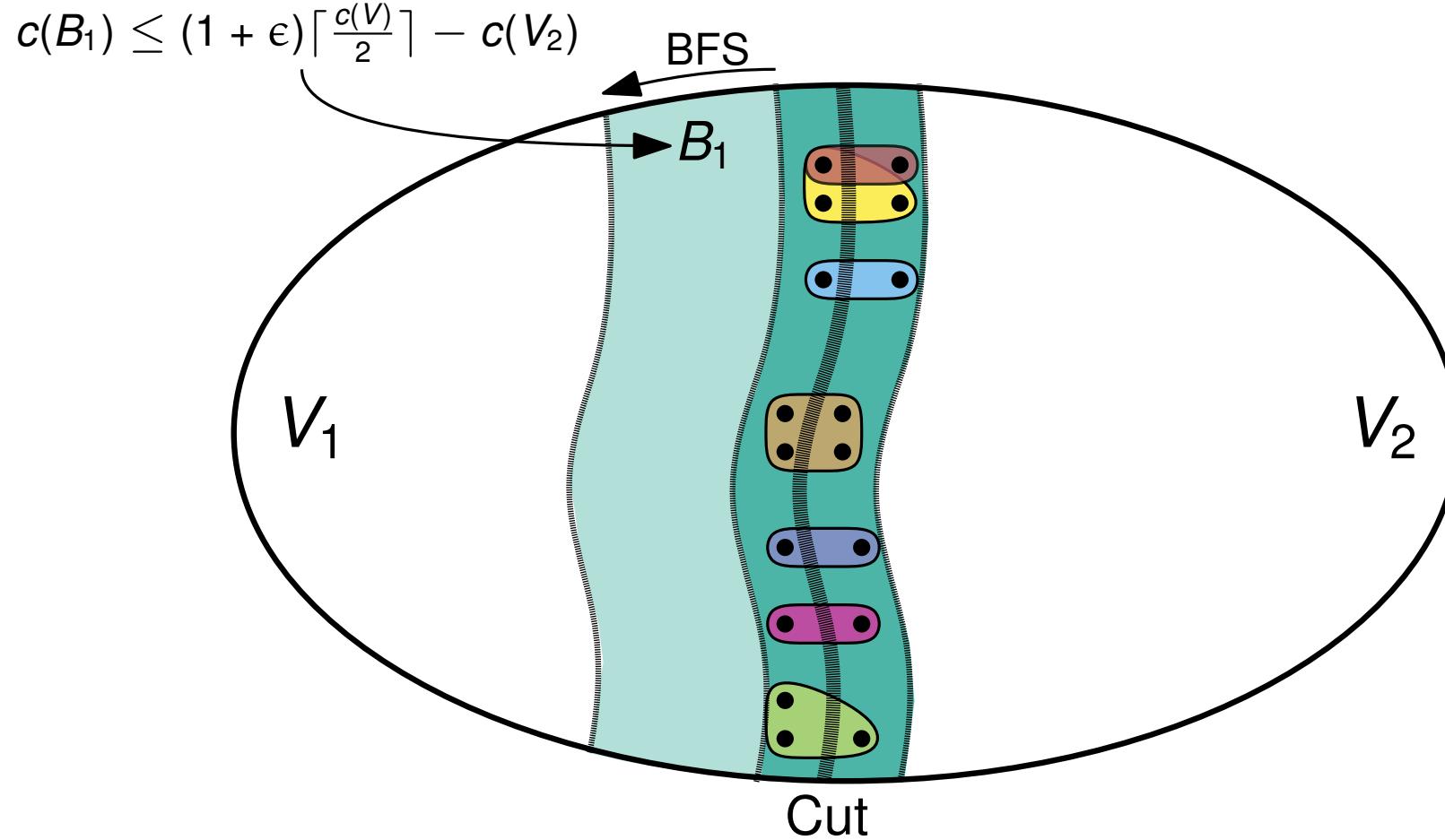


Find feasible minimum cut

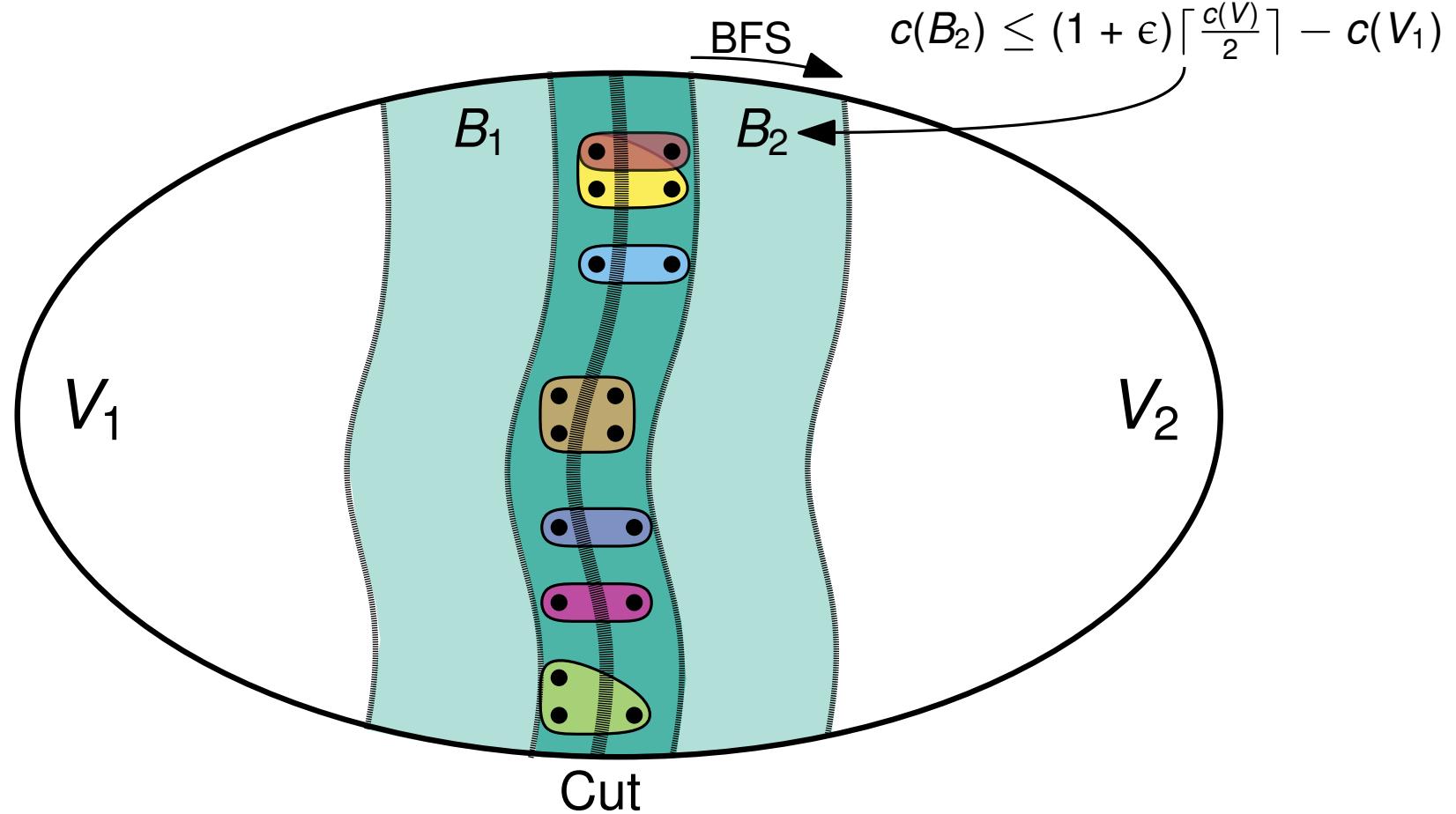
Adaptive Flow Iterations



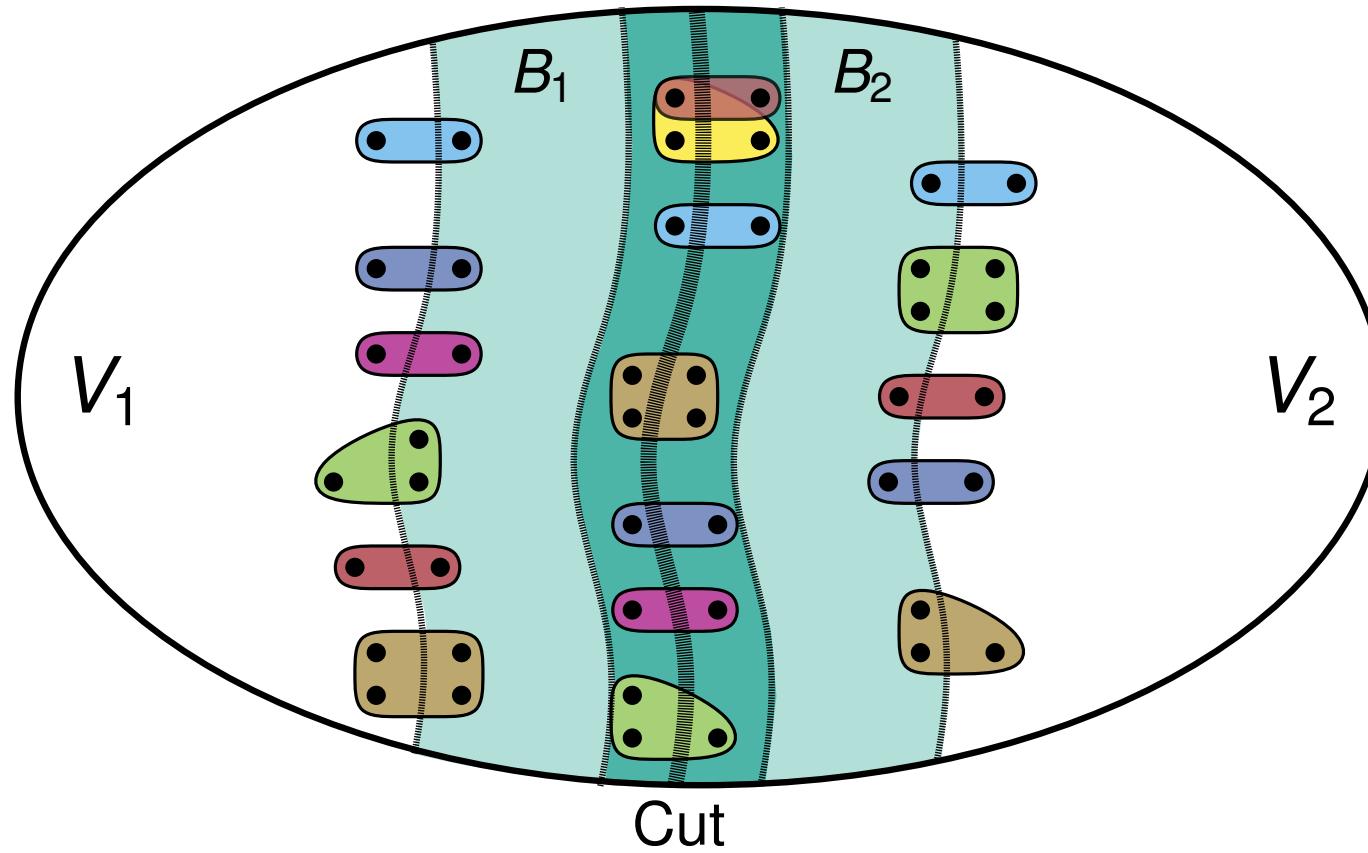
Adaptive Flow Iterations



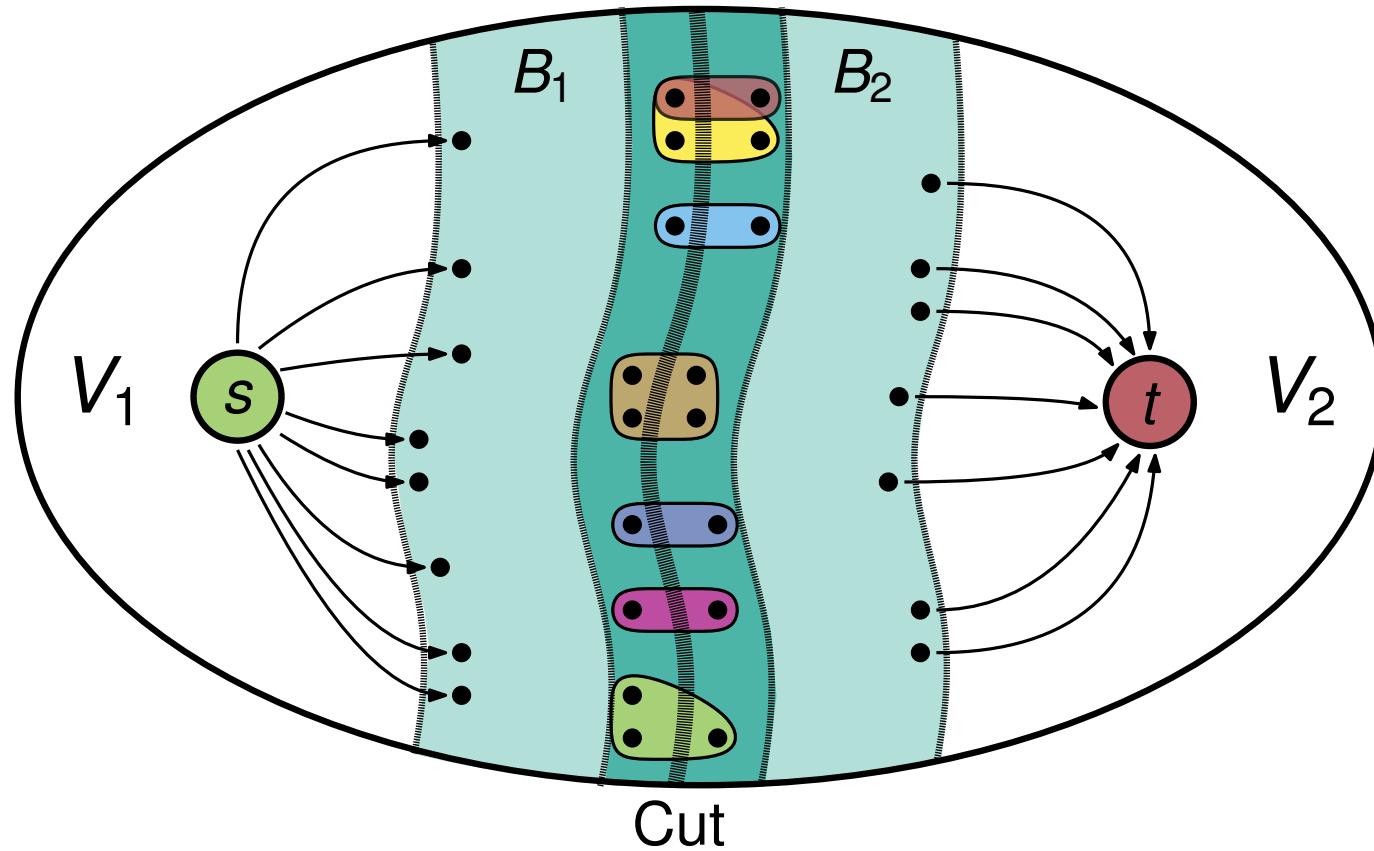
Adaptive Flow Iterations



Adaptive Flow Iterations

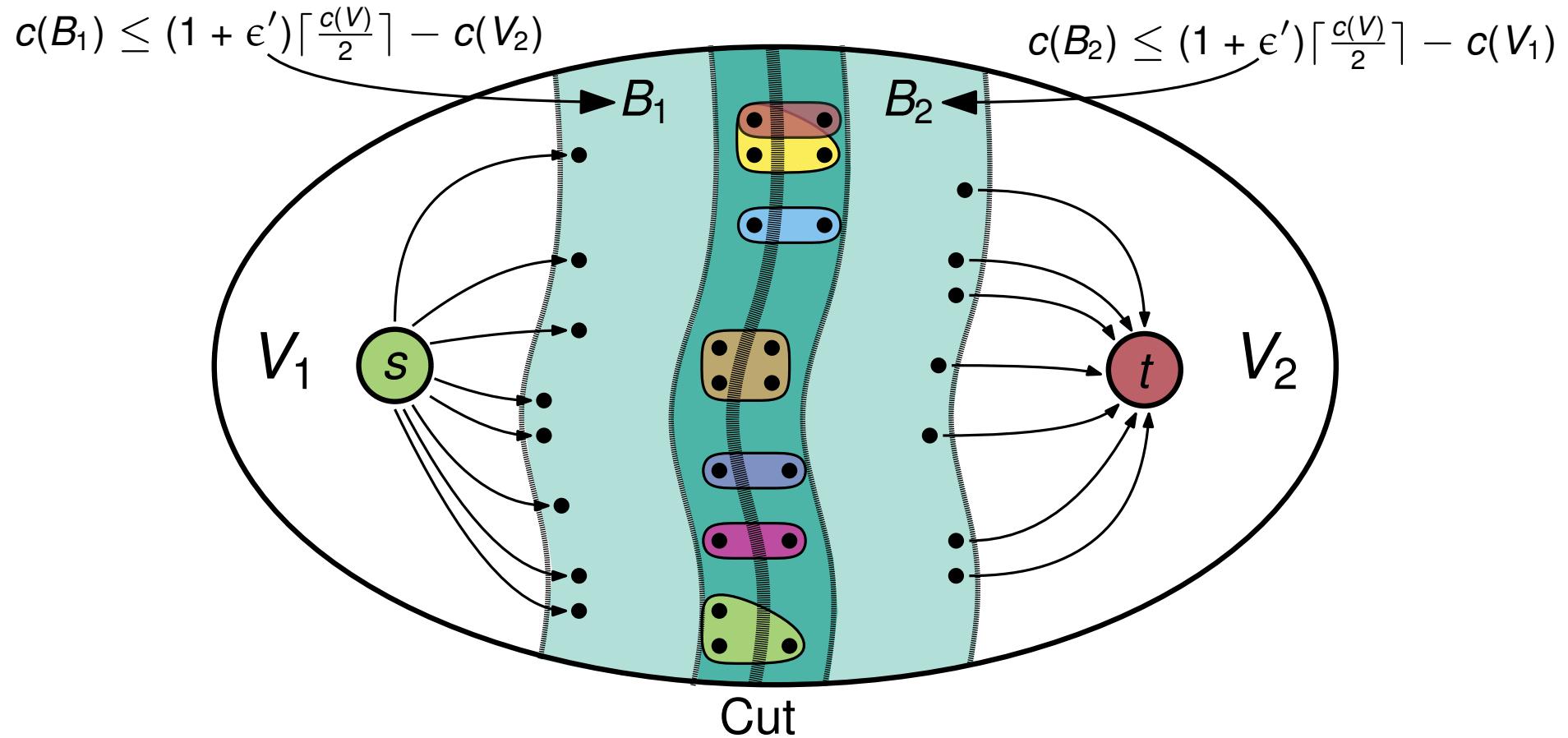


Adaptive Flow Iterations



Adaptive Flow Iterations

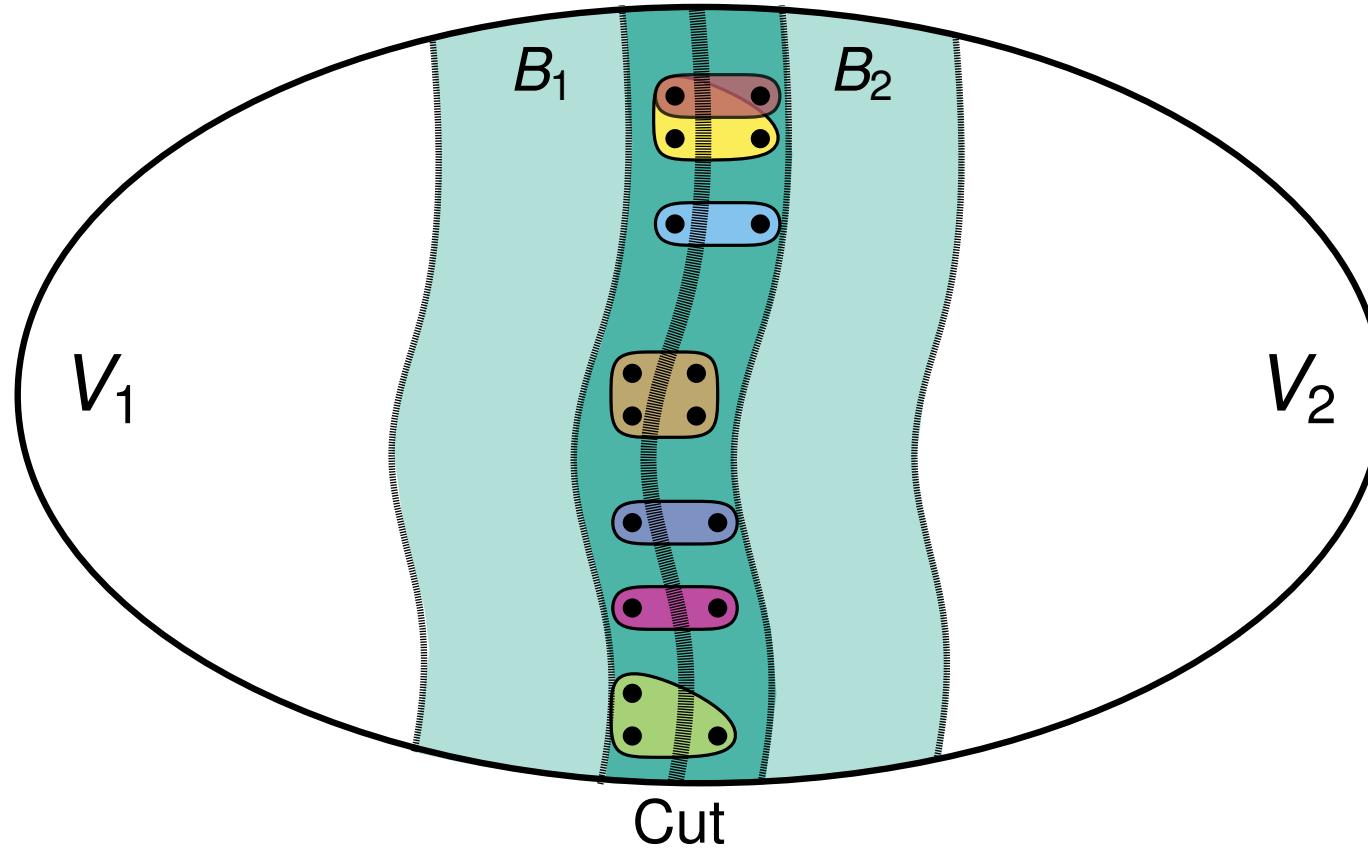
Use $\epsilon' = \alpha\epsilon$ instead of ϵ



Adaptive Flow Iterations

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

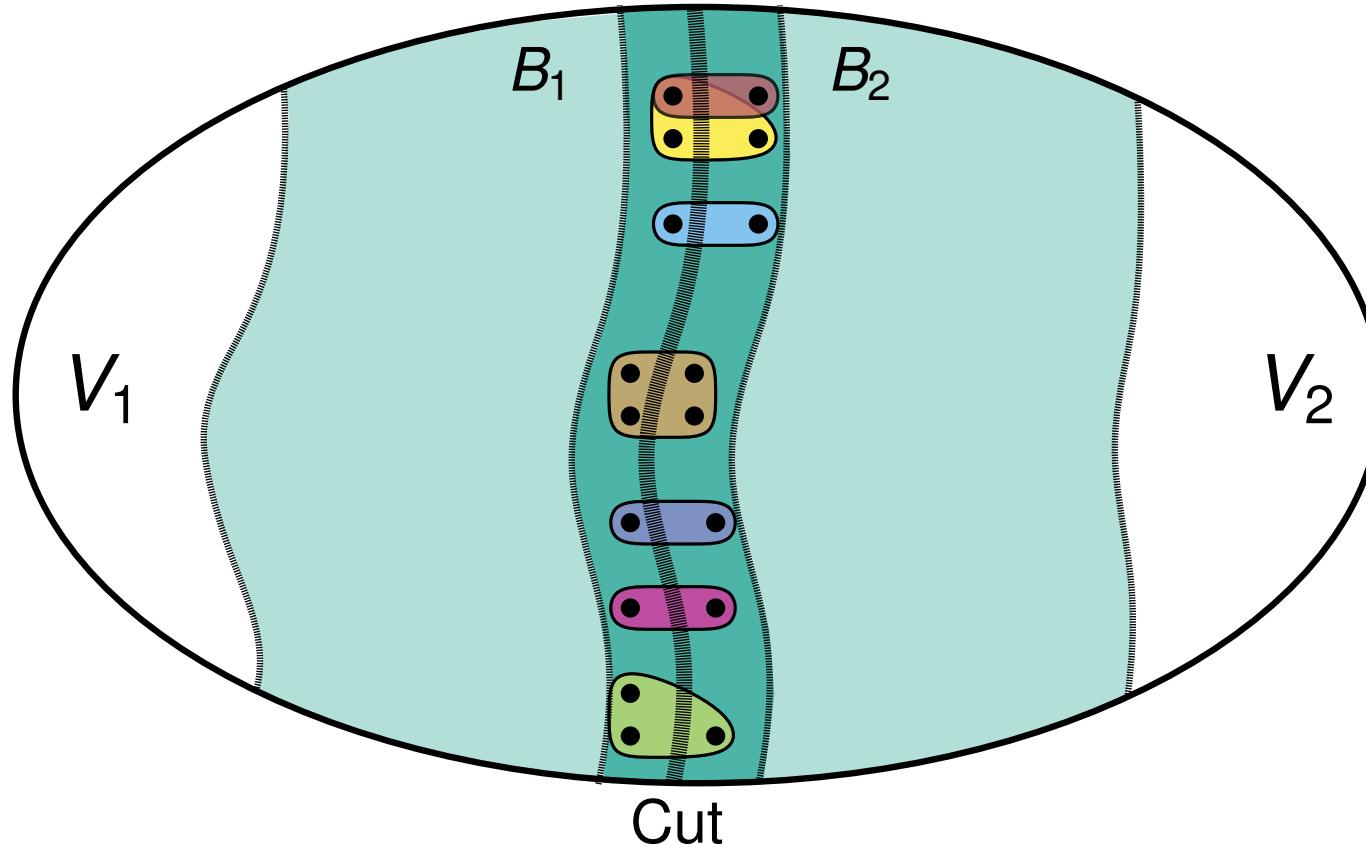
$\alpha = 1 \Rightarrow \text{Improvement Found} \Rightarrow \alpha = \max(2\alpha, \alpha') = 2$



Adaptive Flow Iterations

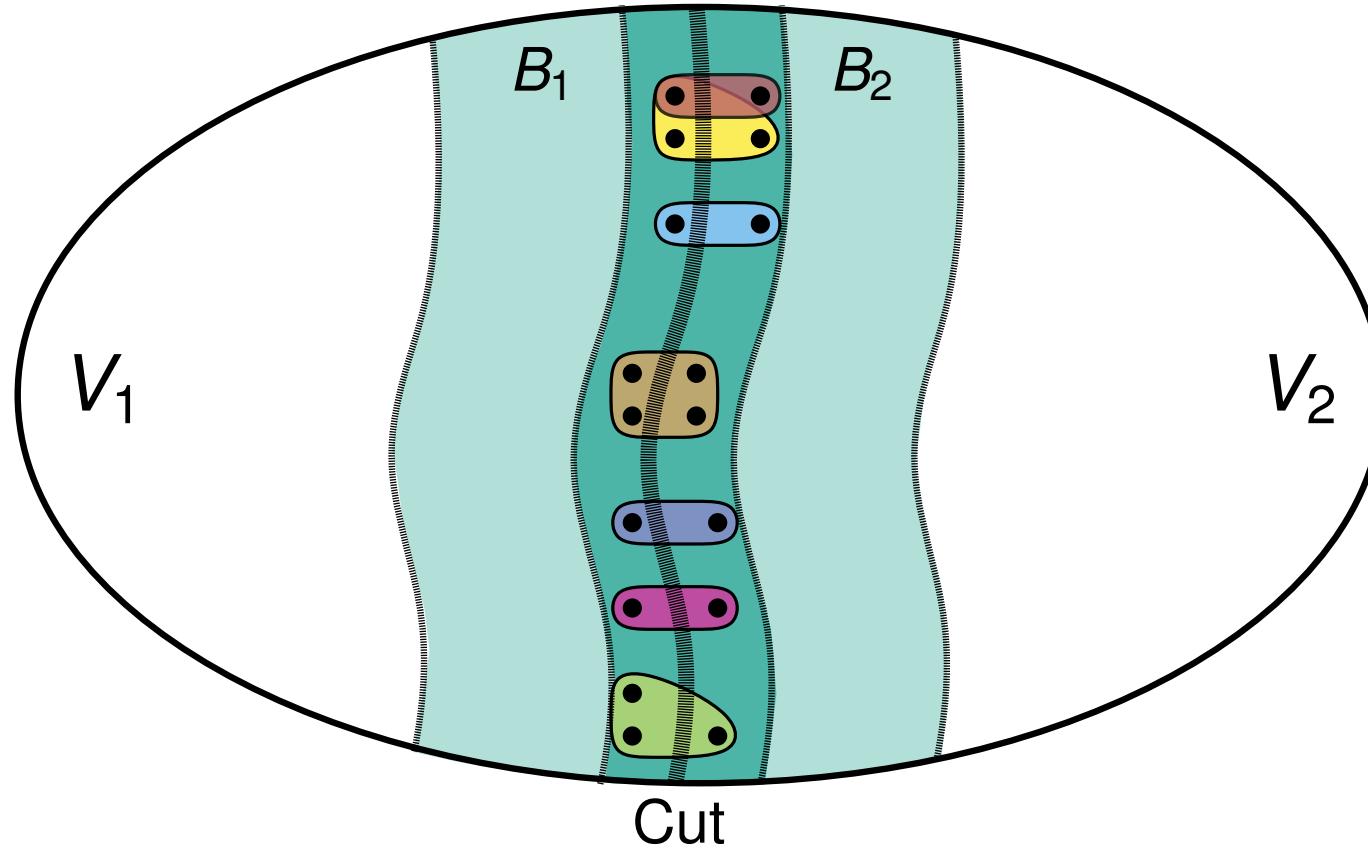
Use $\epsilon' = \alpha\epsilon$ instead of ϵ

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

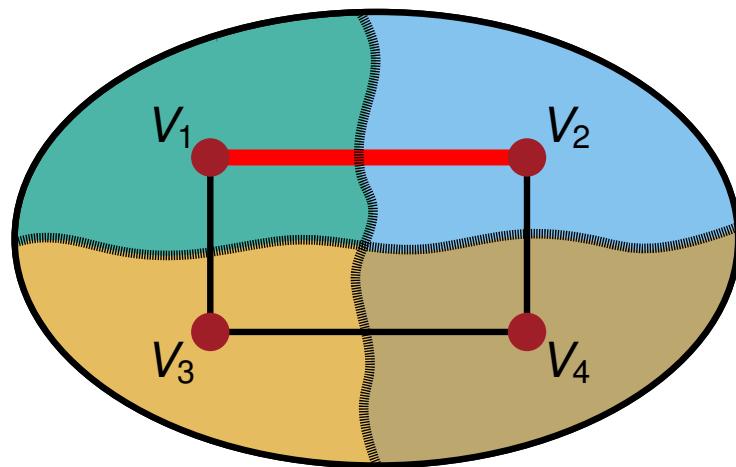


Adaptive Flow Iterations

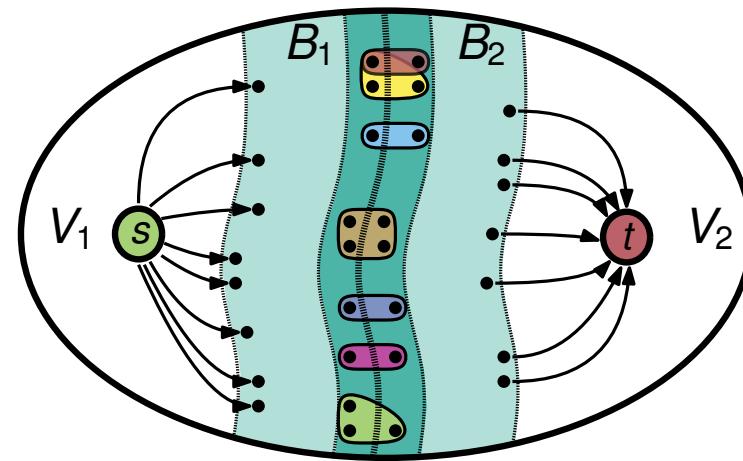
Use $\epsilon' = \alpha\epsilon$ instead of ϵ
 $\alpha = 1 \Rightarrow \text{No Improvement} \Rightarrow \text{Terminate}$



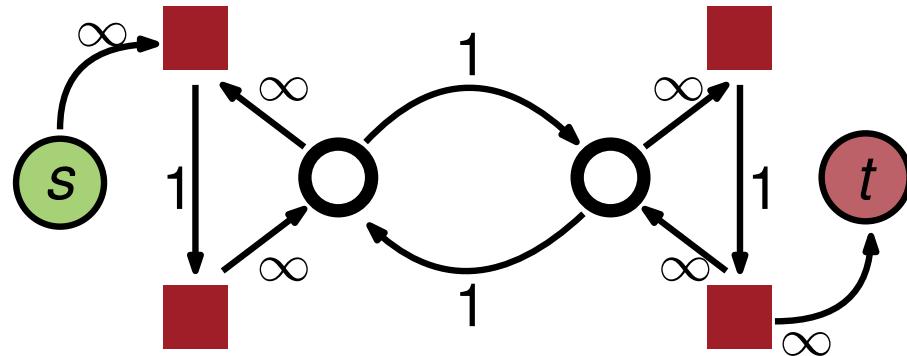
Our Flow-Based Refinement Framework



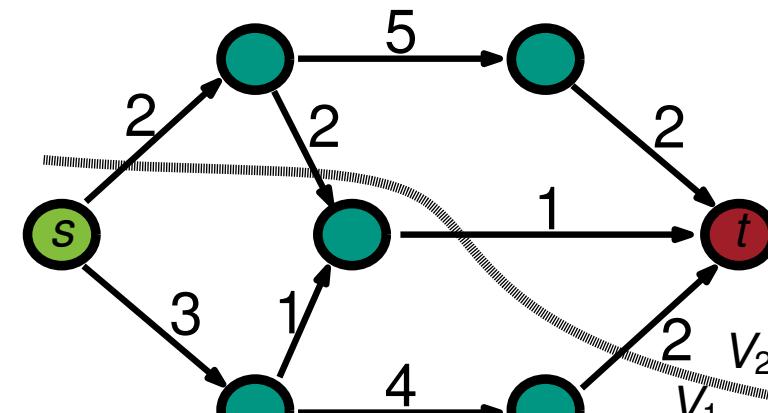
Select two adjacent blocks for refinement



Build Flow Problem

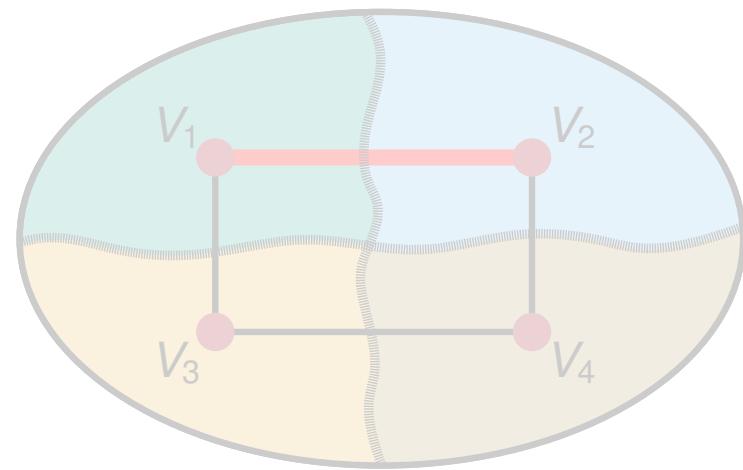


Solve Flow Problem

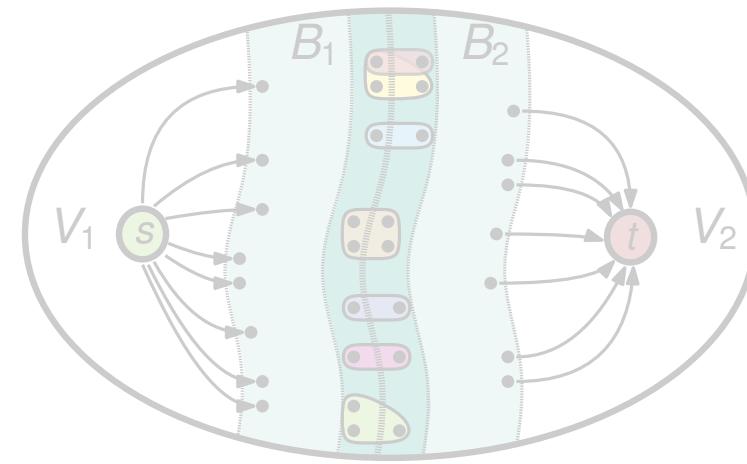


Find feasible minimum cut

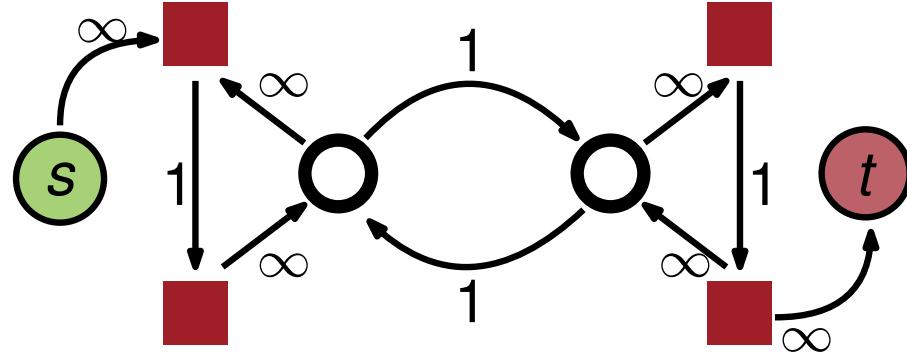
Our Flow-Based Refinement Framework



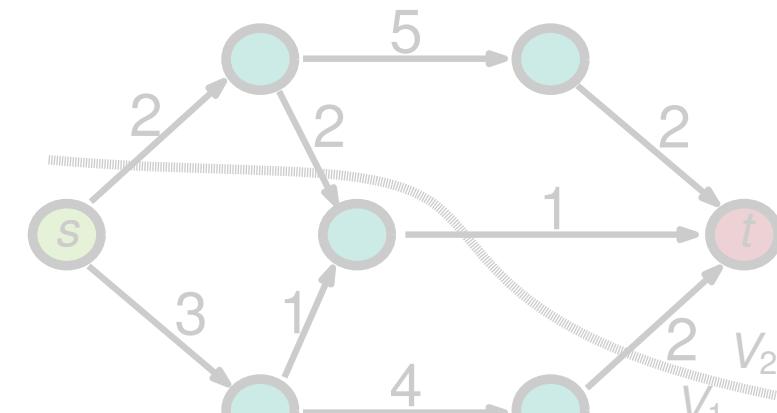
Select two adjacent blocks for refinement



Build Flow Problem

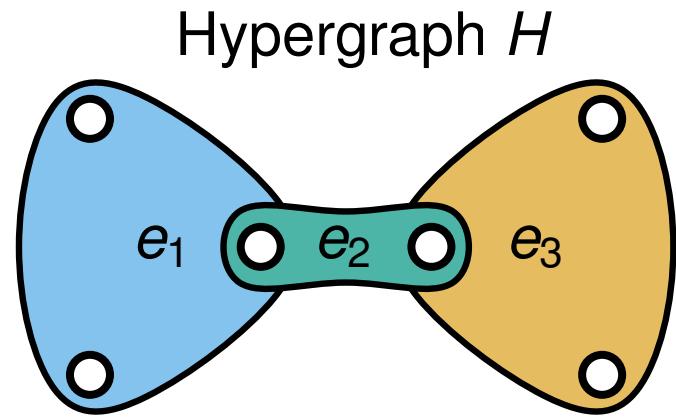


Solve Flow Problem

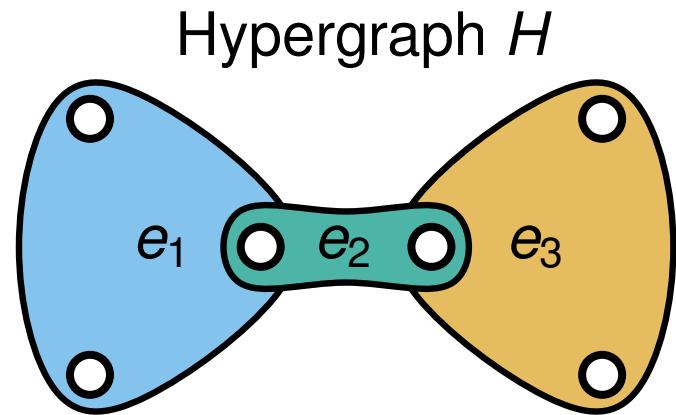


Find feasible minimum cut

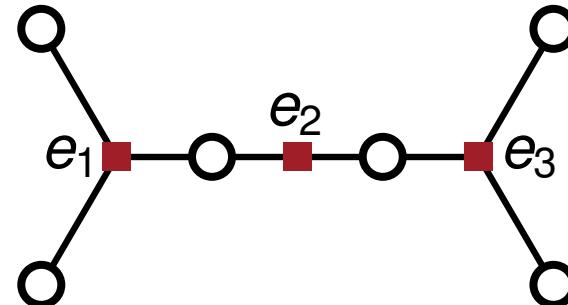
Hypergraph Flow Network



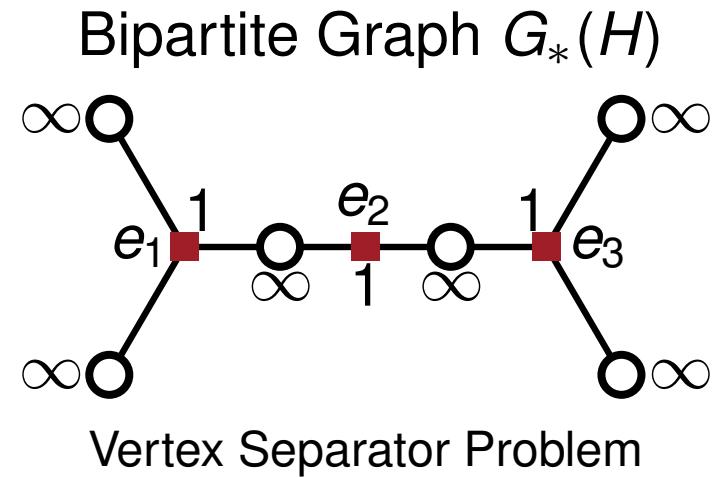
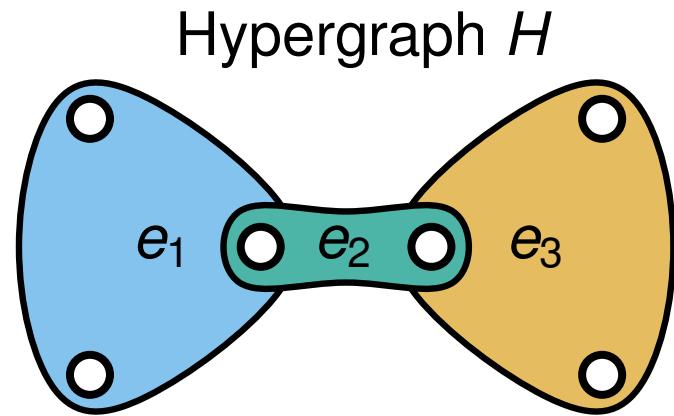
Hypergraph Flow Network



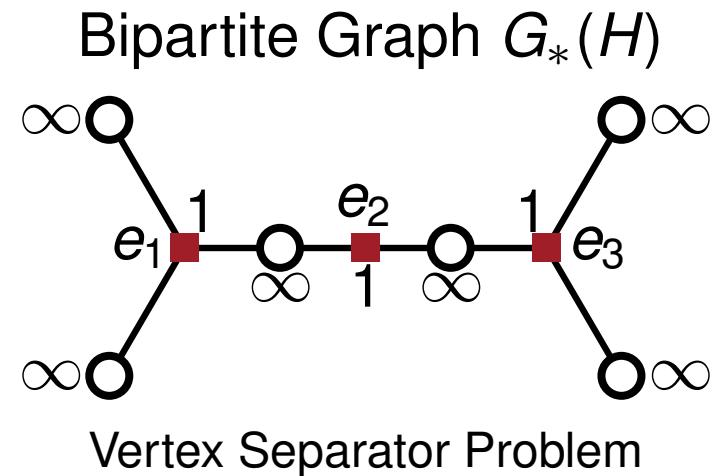
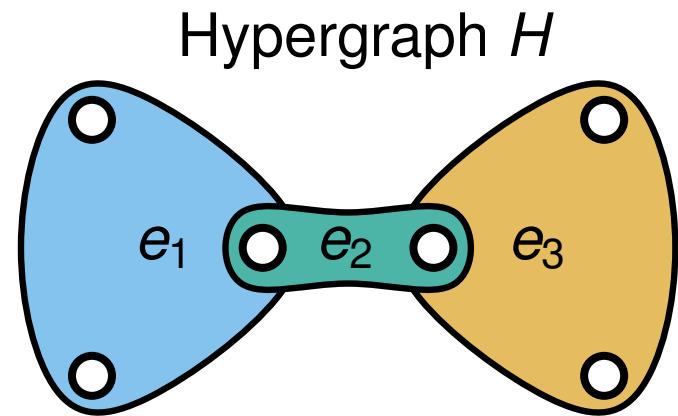
Bipartite Graph $G_*(H)$



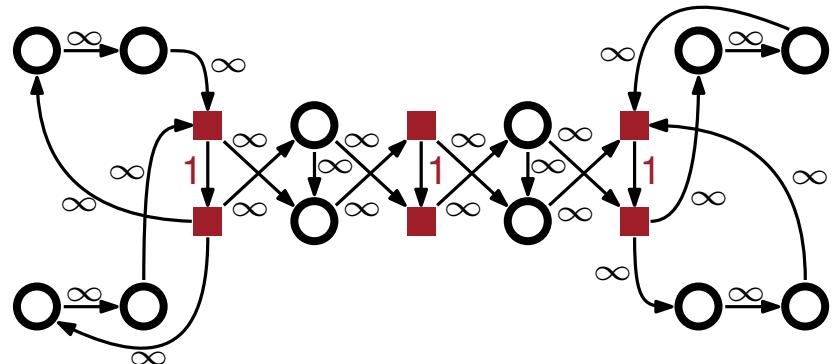
Hypergraph Flow Network



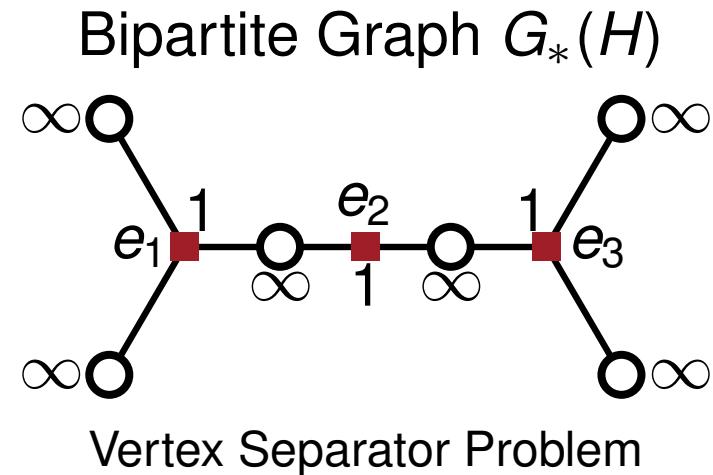
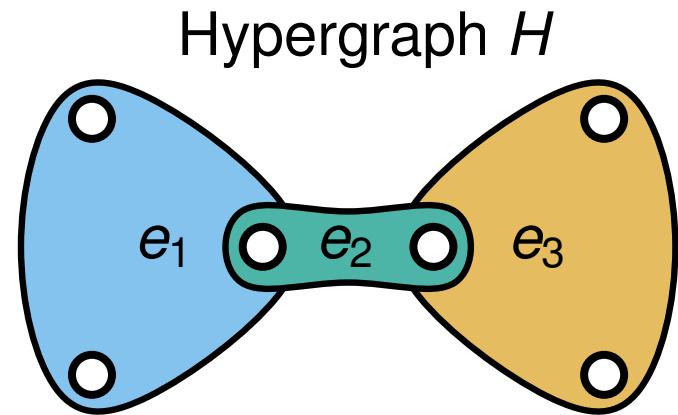
Hypergraph Flow Network



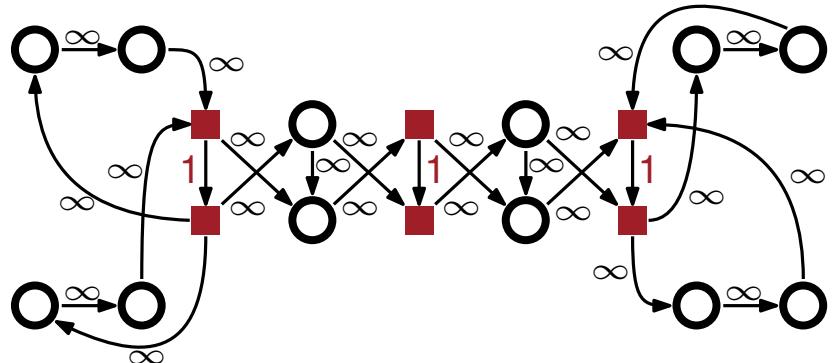
Vertex Separator Transformation



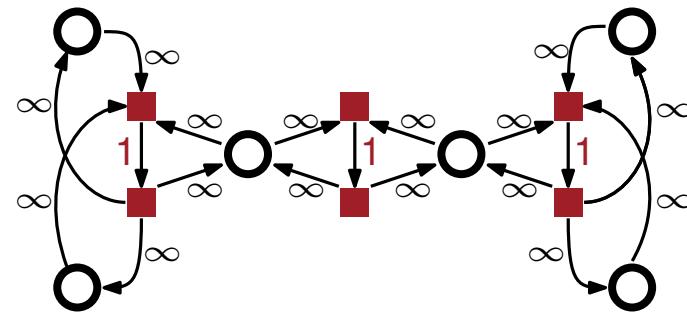
Hypergraph Flow Network



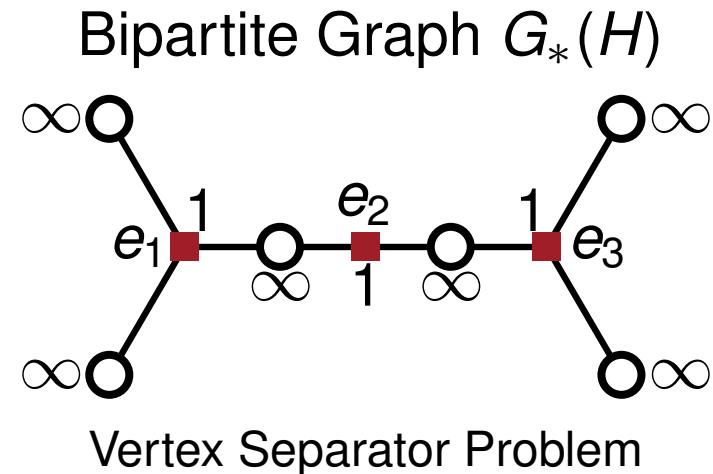
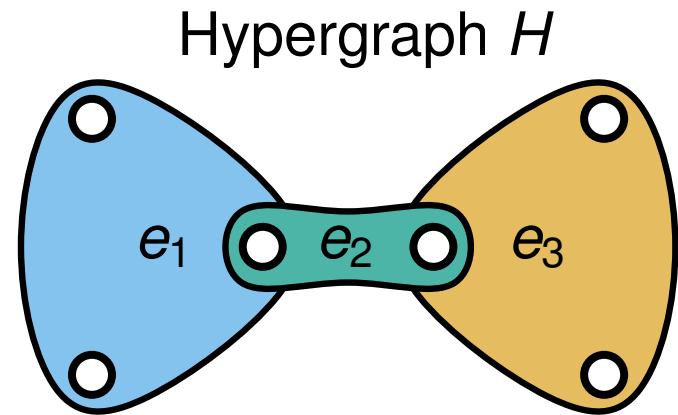
Vertex Separator Transformation



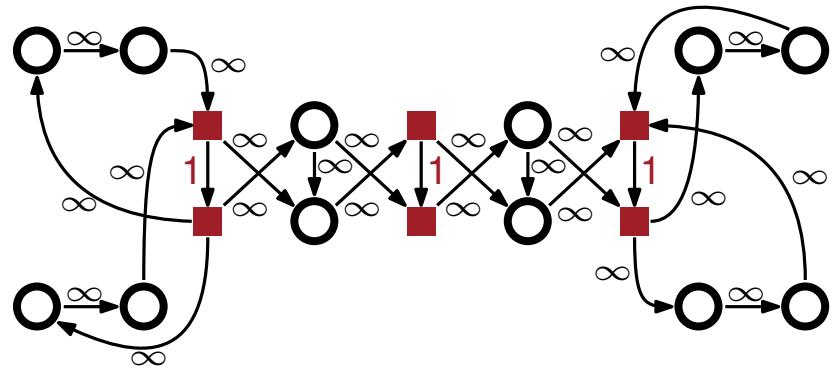
Lawler Network



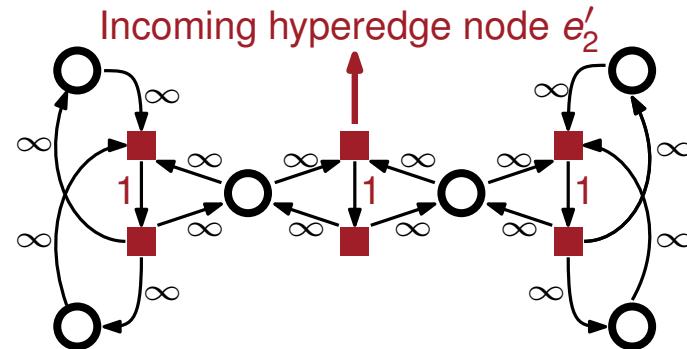
Hypergraph Flow Network



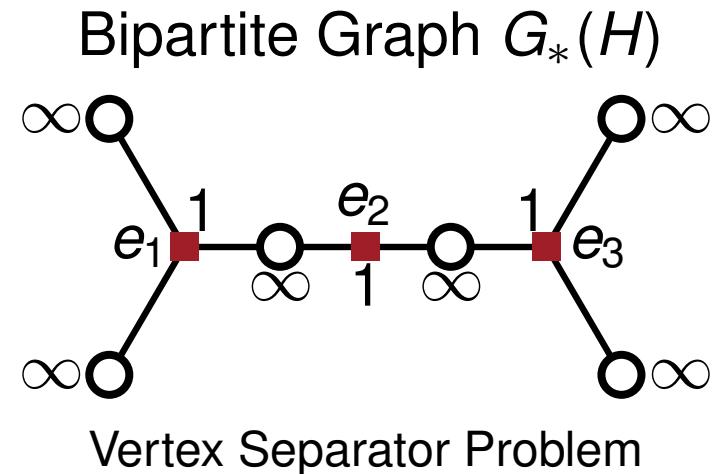
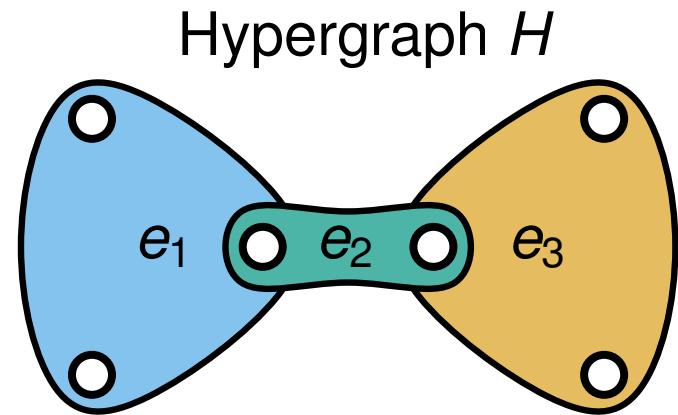
Vertex Separator Transformation



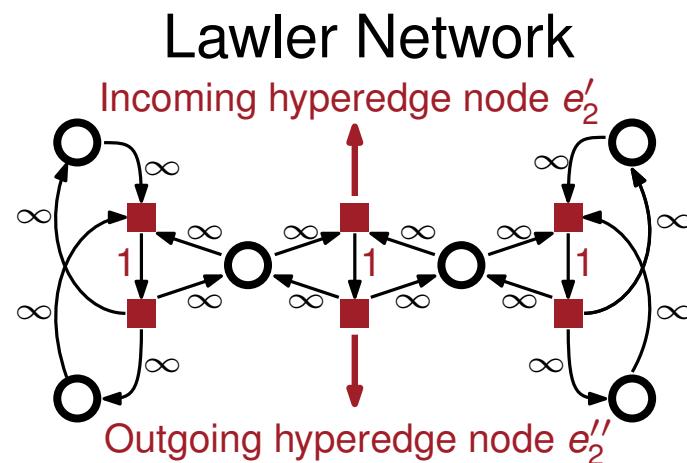
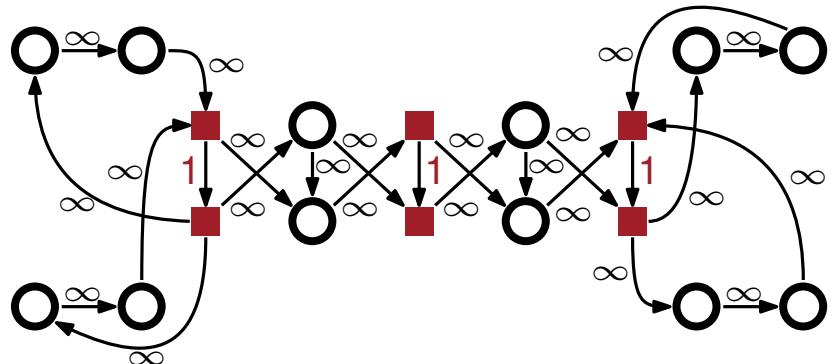
Lawler Network



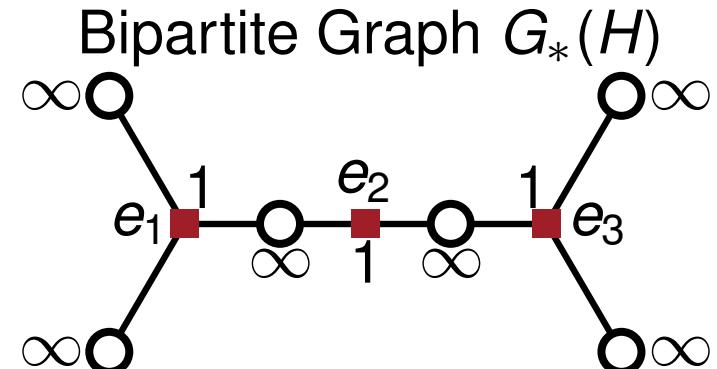
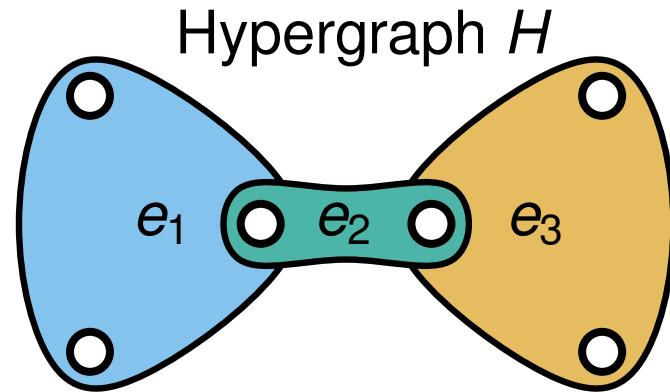
Hypergraph Flow Network



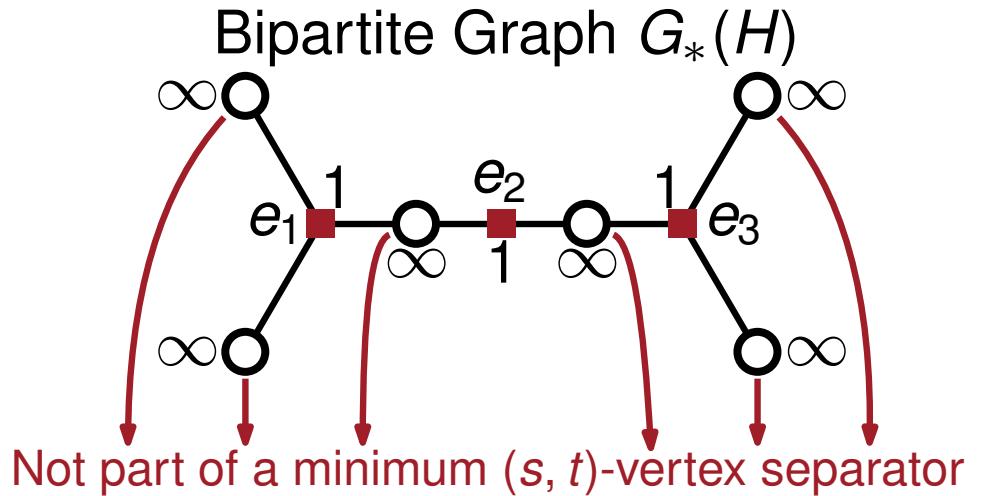
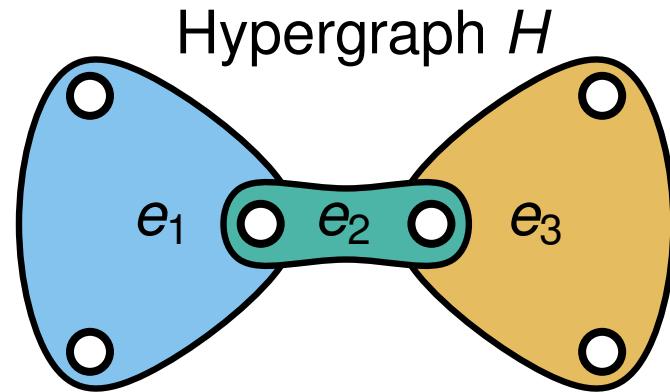
Vertex Separator Transformation



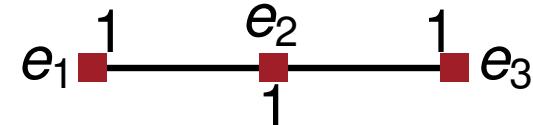
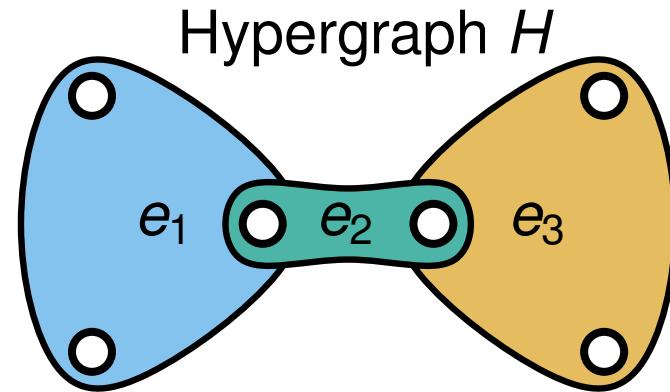
Hypergraph Flow Network - Low Degree Vertices



Hypergraph Flow Network - Low Degree Vertices

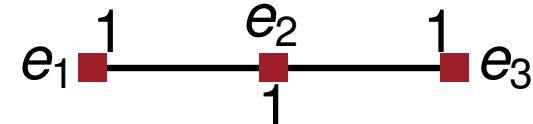
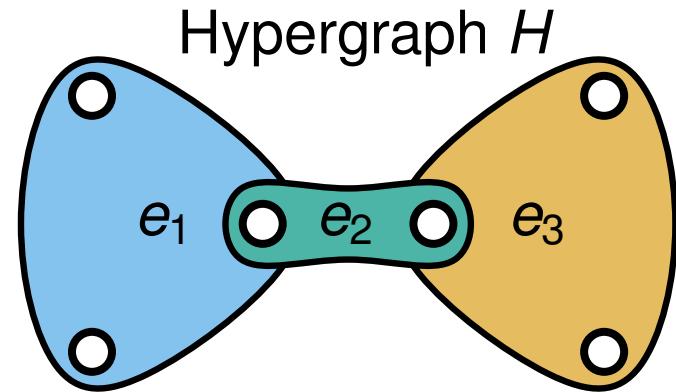


Hypergraph Flow Network - Low Degree Vertices



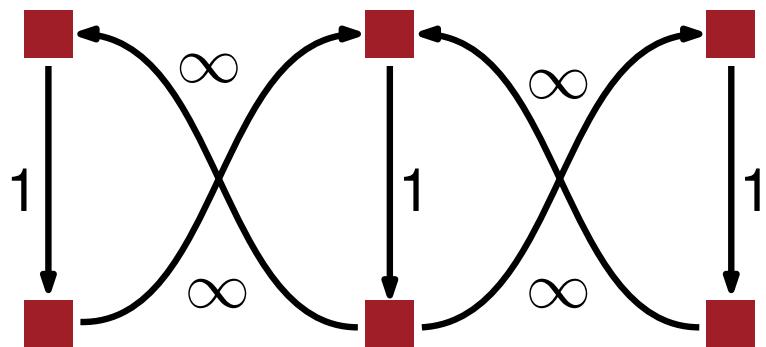
Remove all vertices by adding a clique

Hypergraph Flow Network - Low Degree Vertices

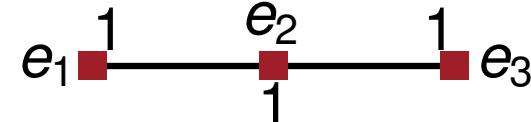
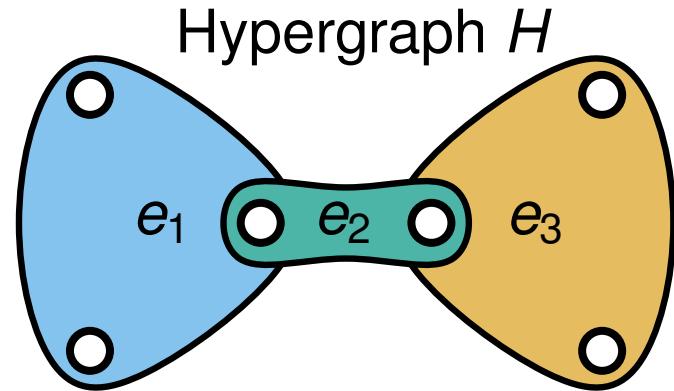


Remove all vertices by adding a clique

Our Network

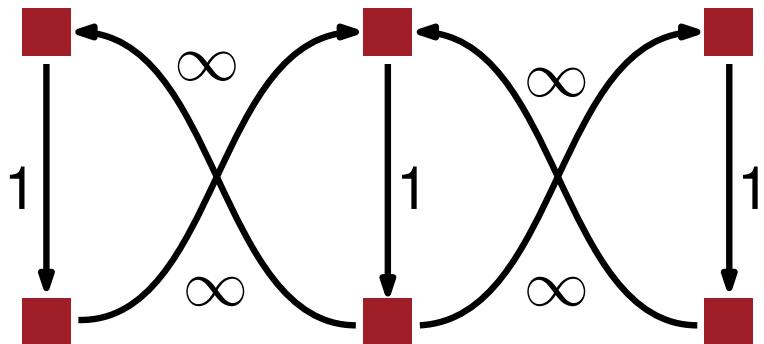


Hypergraph Flow Network - Low Degree Vertices



Remove all vertices by adding a clique

Our Network

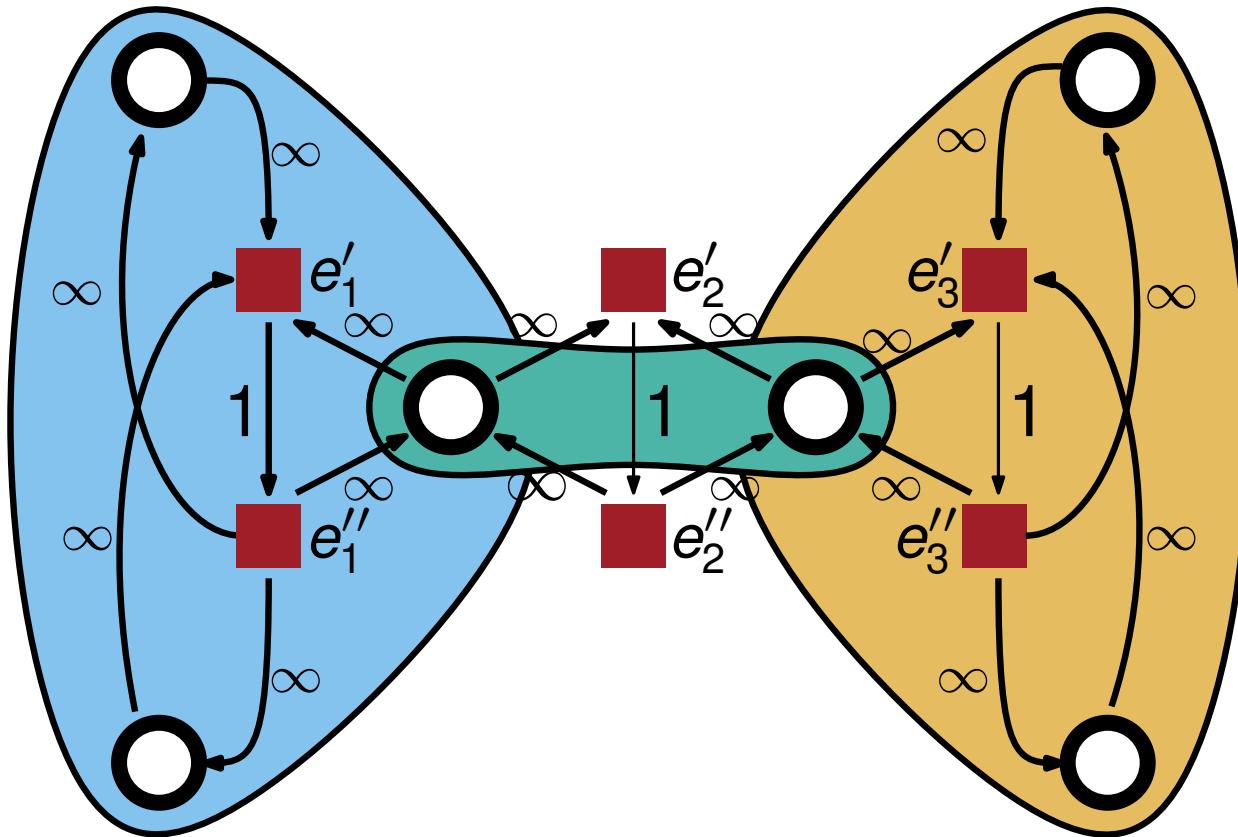


A hypernode v induces ...

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

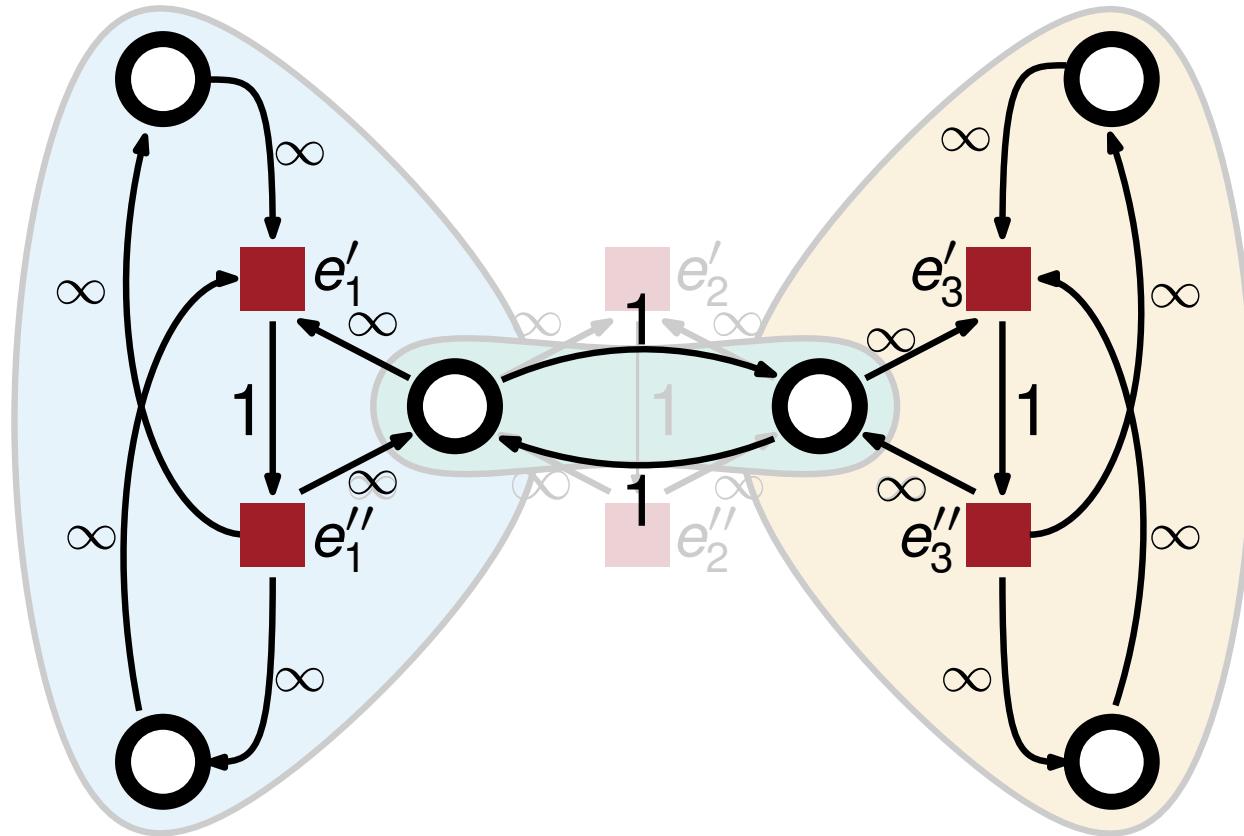
If $d(v) \leq 3$, then $d(v)(d(v) - 1) \leq 2d(v)$

Hypergraph Flow Network - Summary



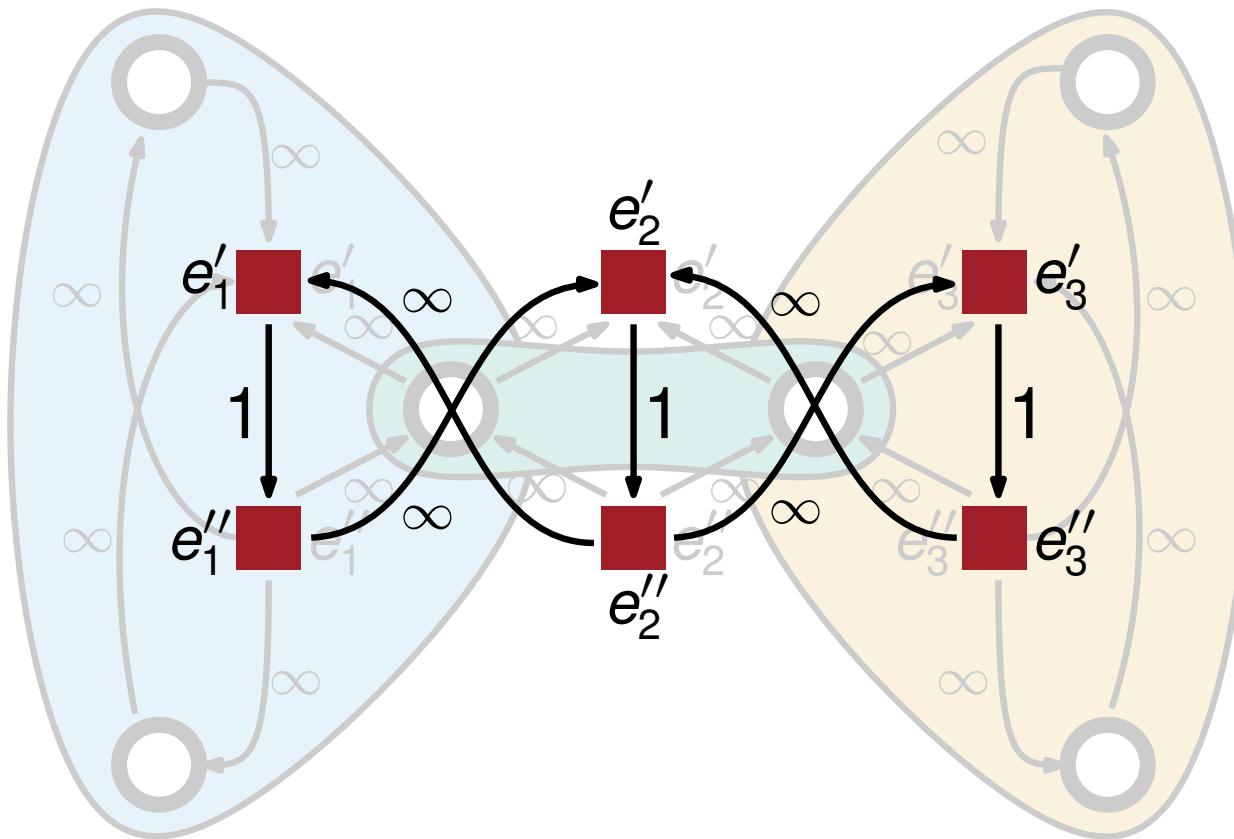
Lawler Network

Hypergraph Flow Network - Summary



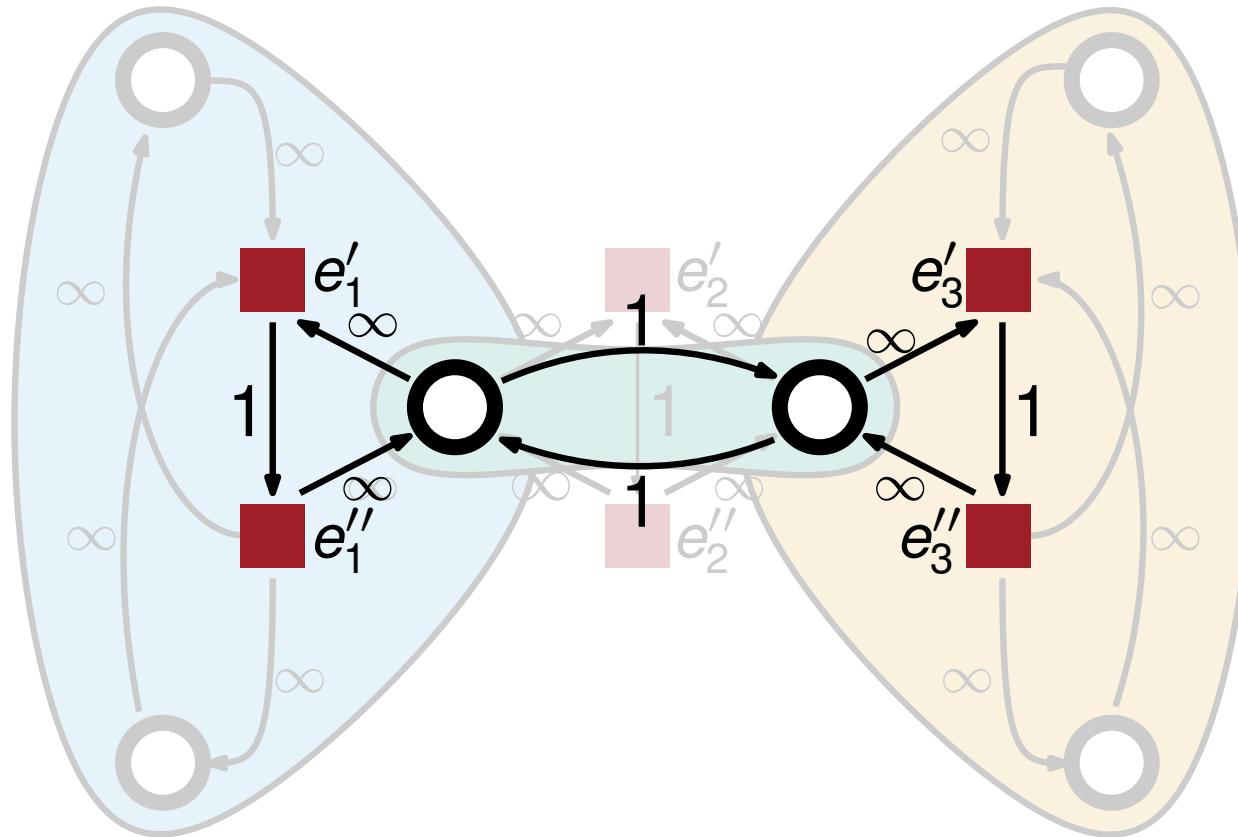
Wong Network

Hypergraph Flow Network - Summary



Our Network

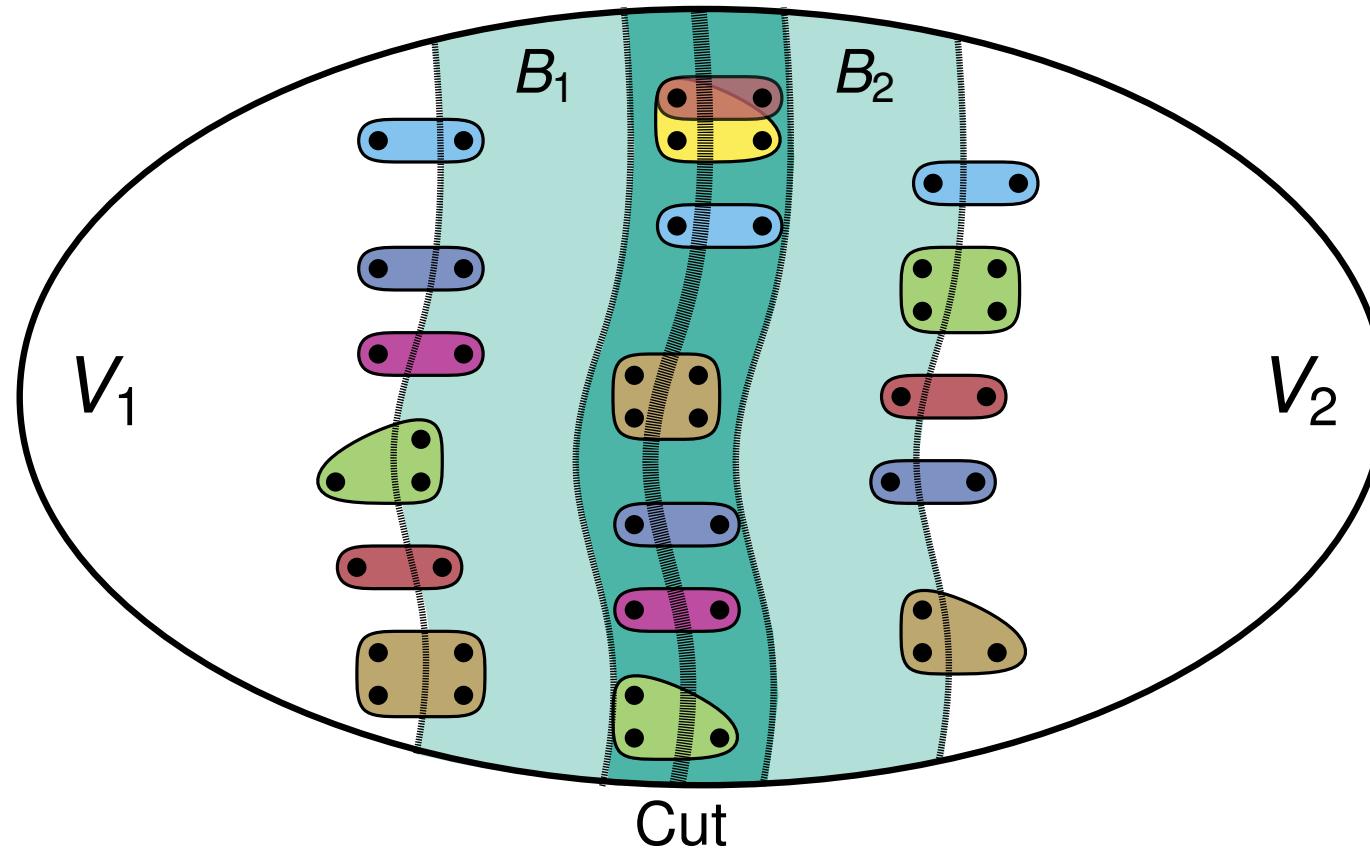
Hypergraph Flow Network - Summary



Hybrid Network

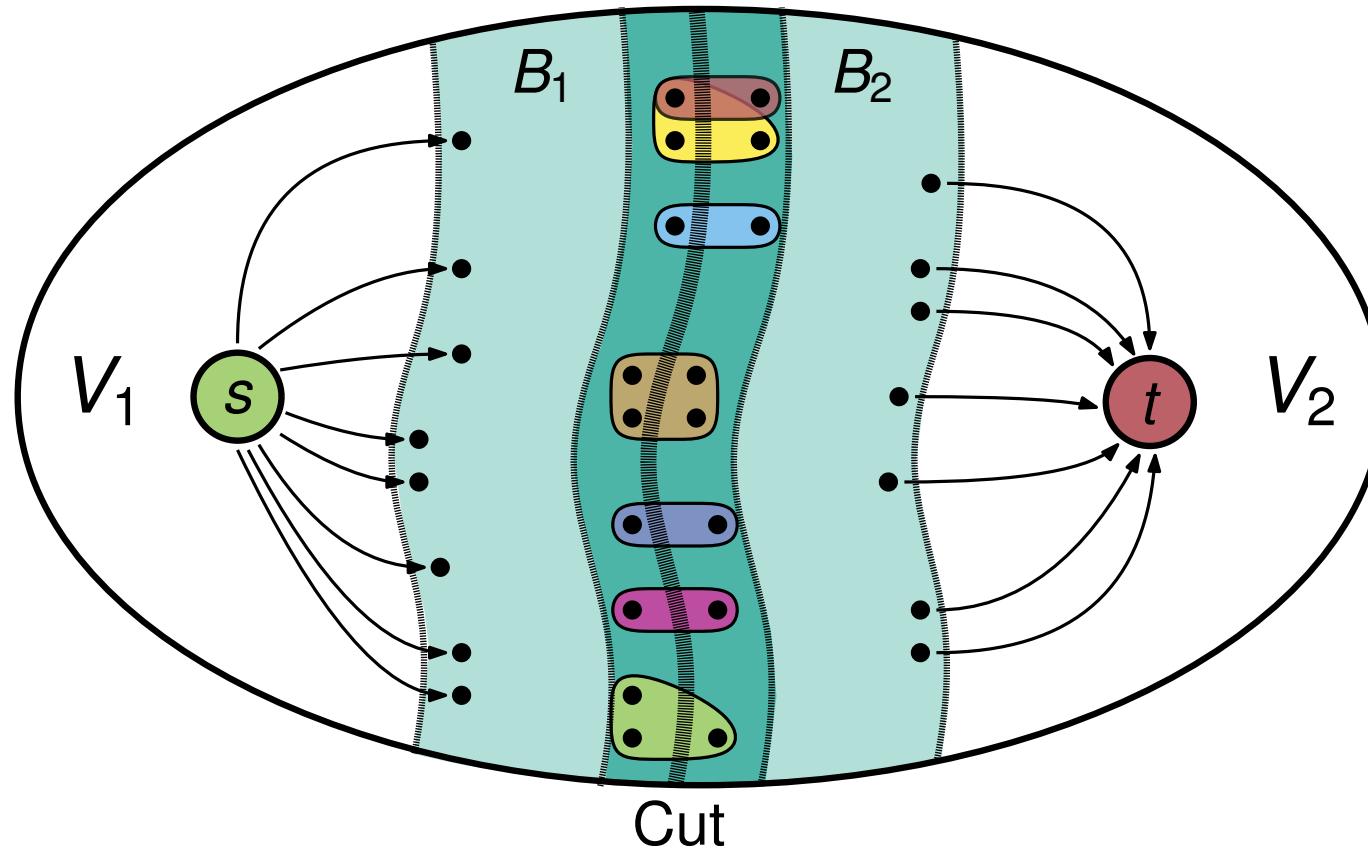
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaFFPa*



Optimized Flow Problem Modeling Approach

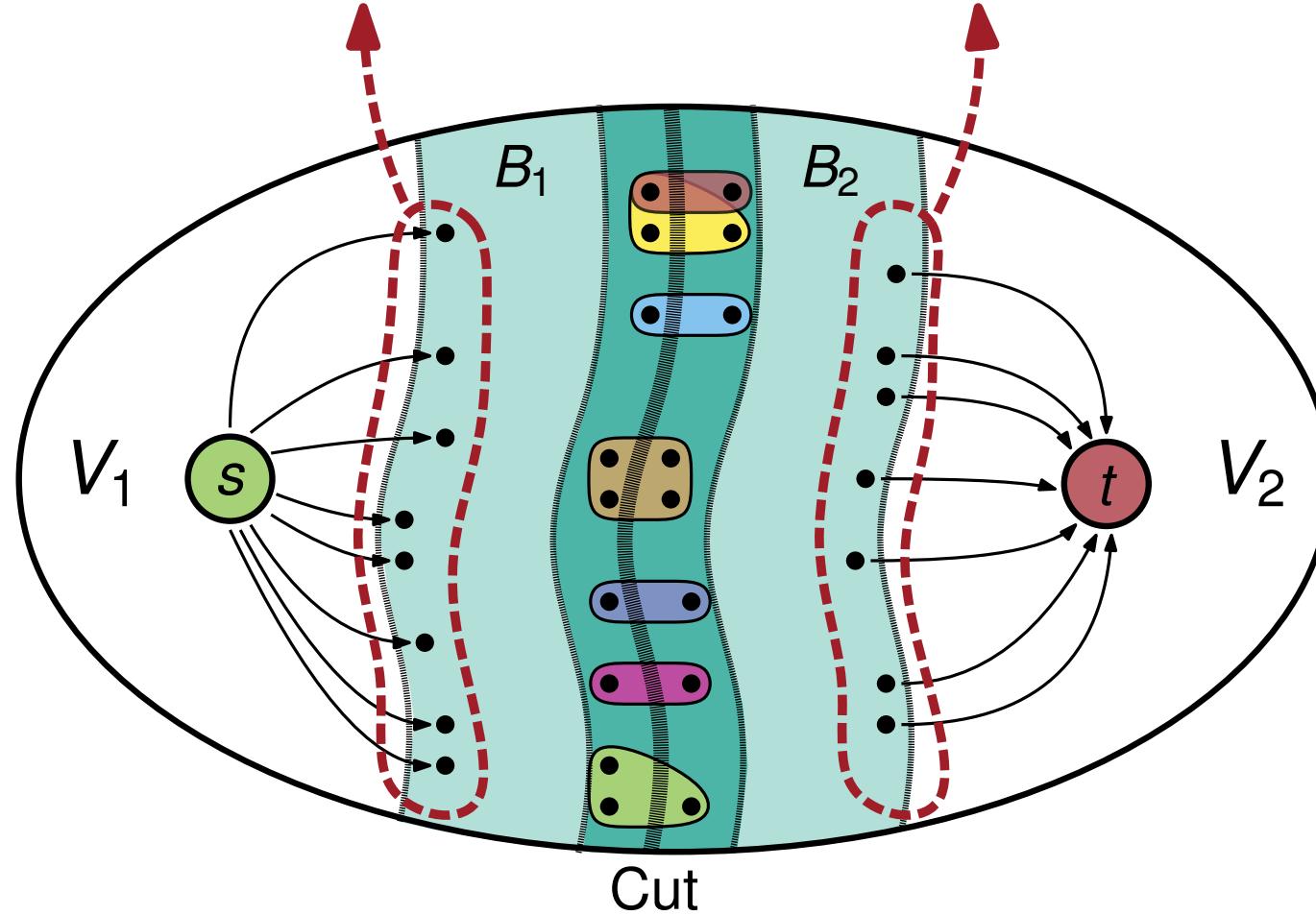
Modeling Approach in *KaFFPa*



Optimized Flow Problem Modeling Approach

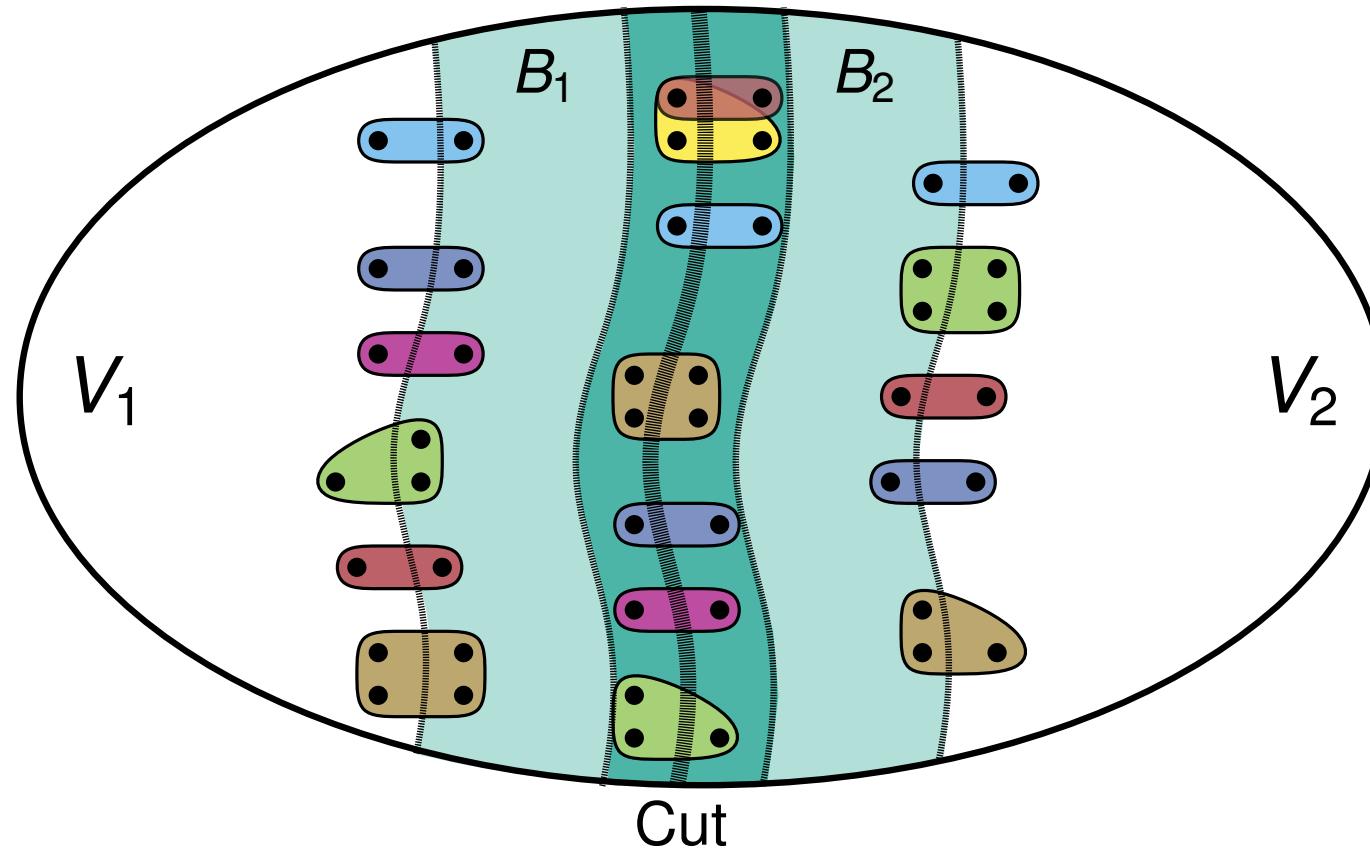
Modeling Approach in *KaFFPa*

Not moveable after Max-Flow-Min-Cut computation



Optimized Flow Problem Modeling Approach

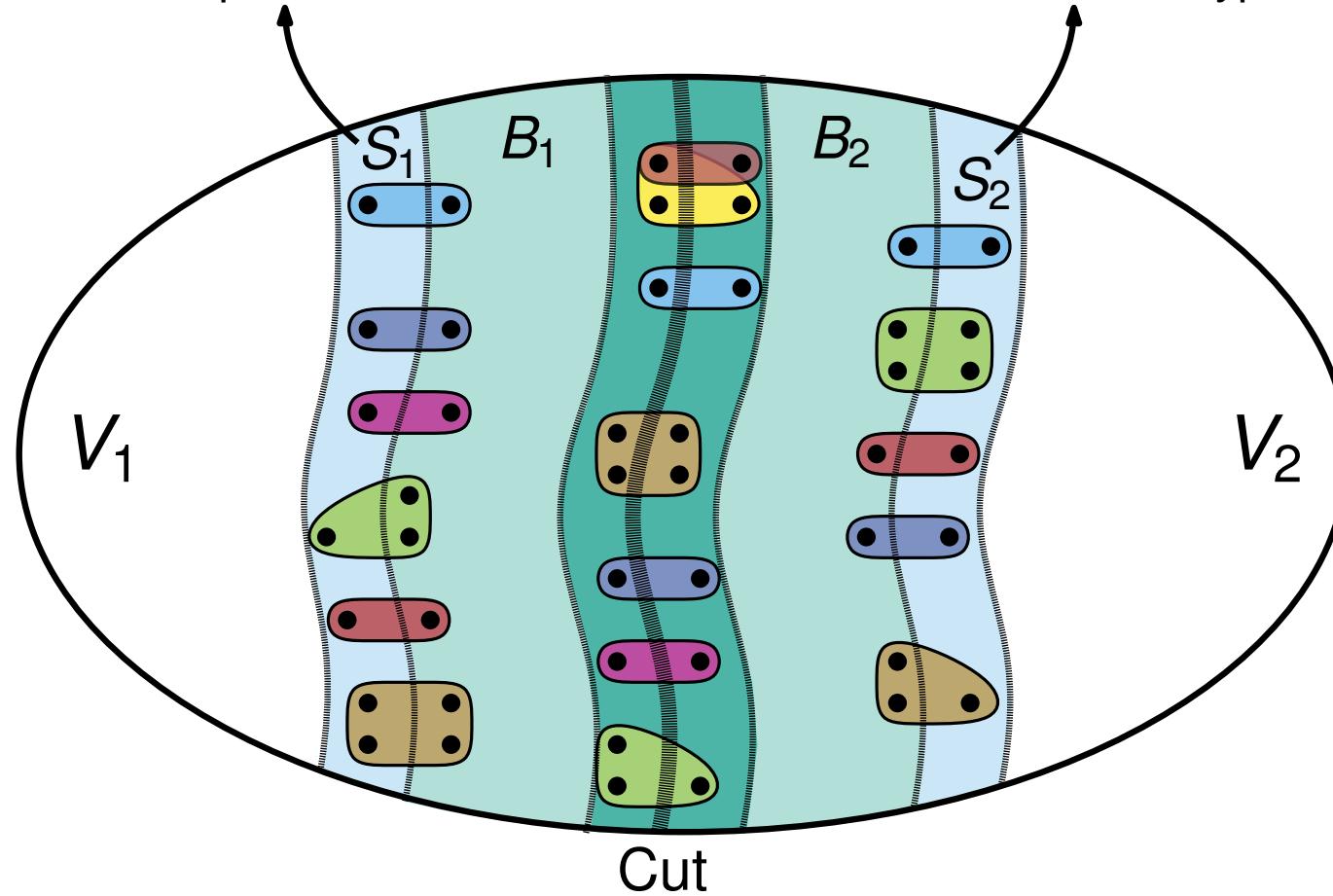
Modeling Approach in *KaHyPar*



Optimized Flow Problem Modeling Approach

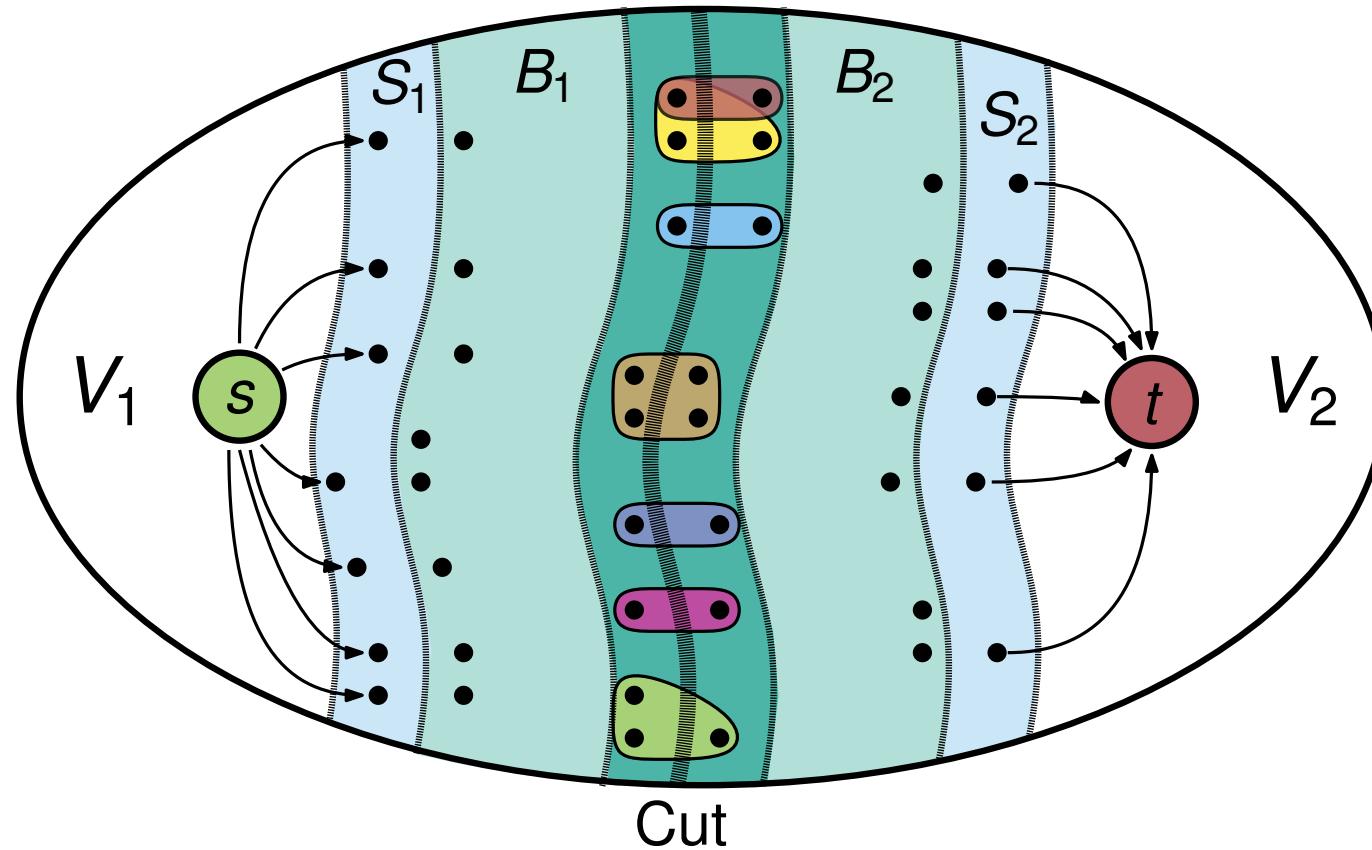
Modeling Approach in *KaHyPar*

Extend flow problem with all vertices contained in a border hyperedge



Optimized Flow Problem Modeling Approach

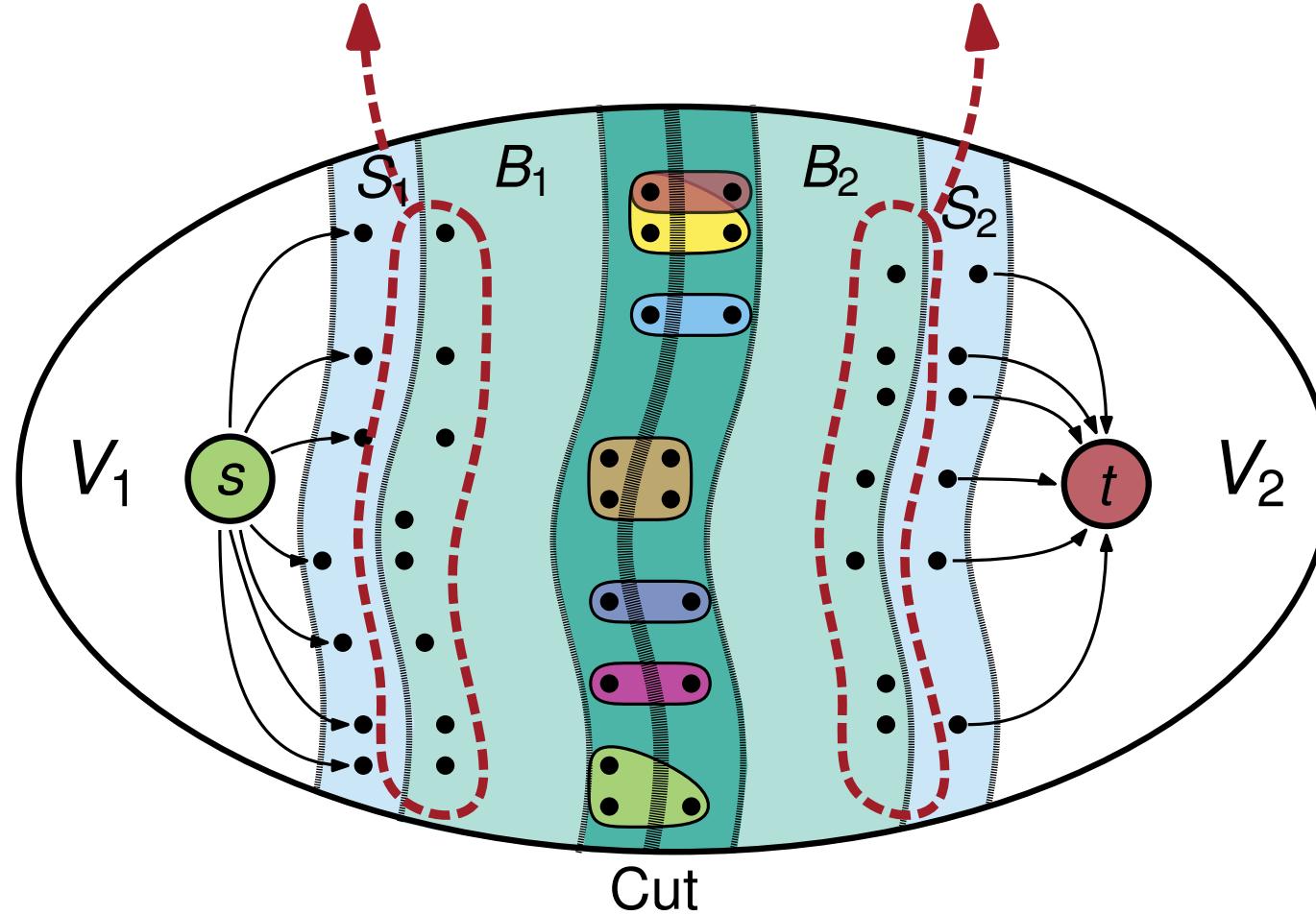
Modeling Approach in *KaHyPar*



Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*

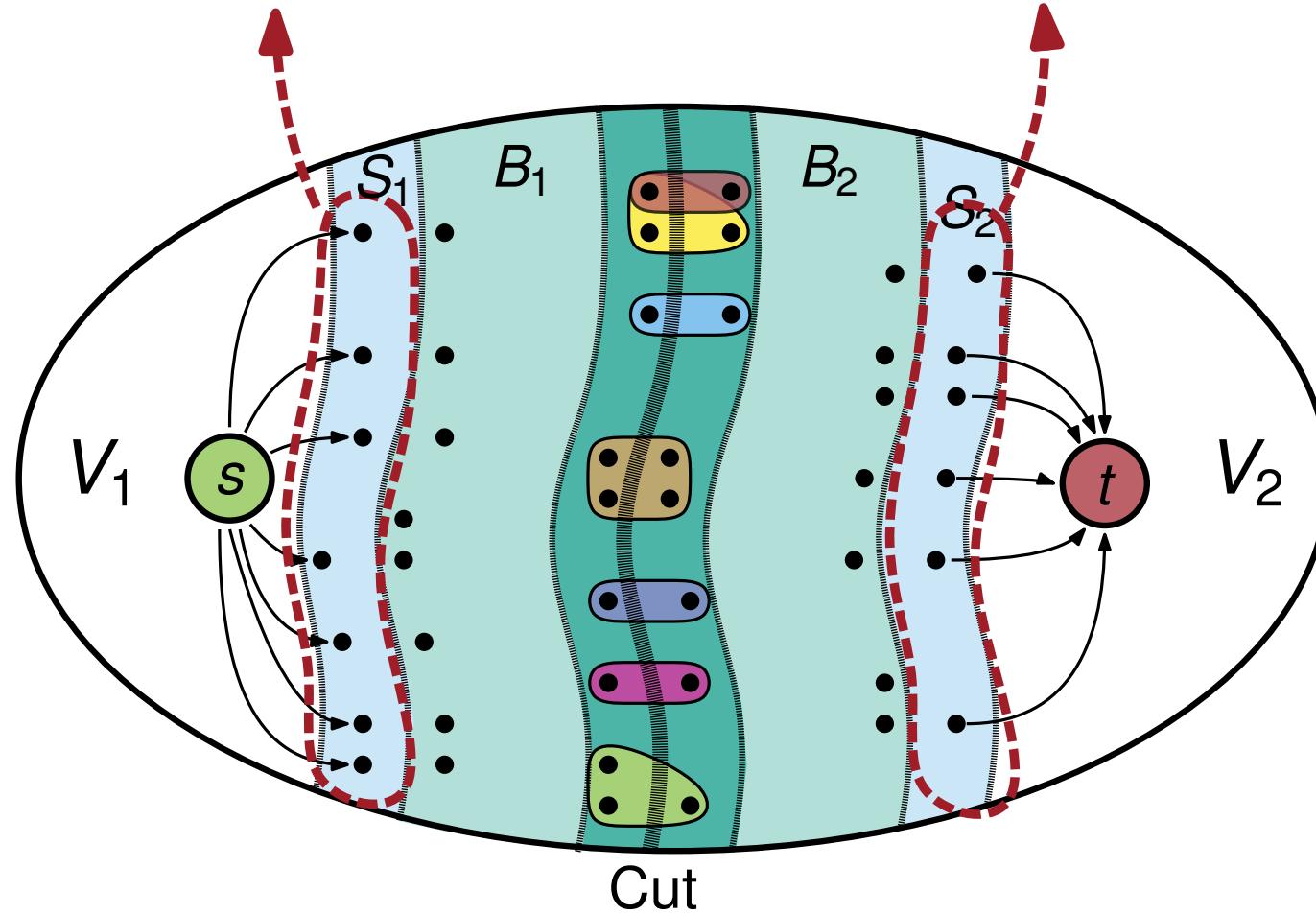
Moveable after Max-Flow-Min-Cut computation, but . . .



Optimized Flow Problem Modeling Approach

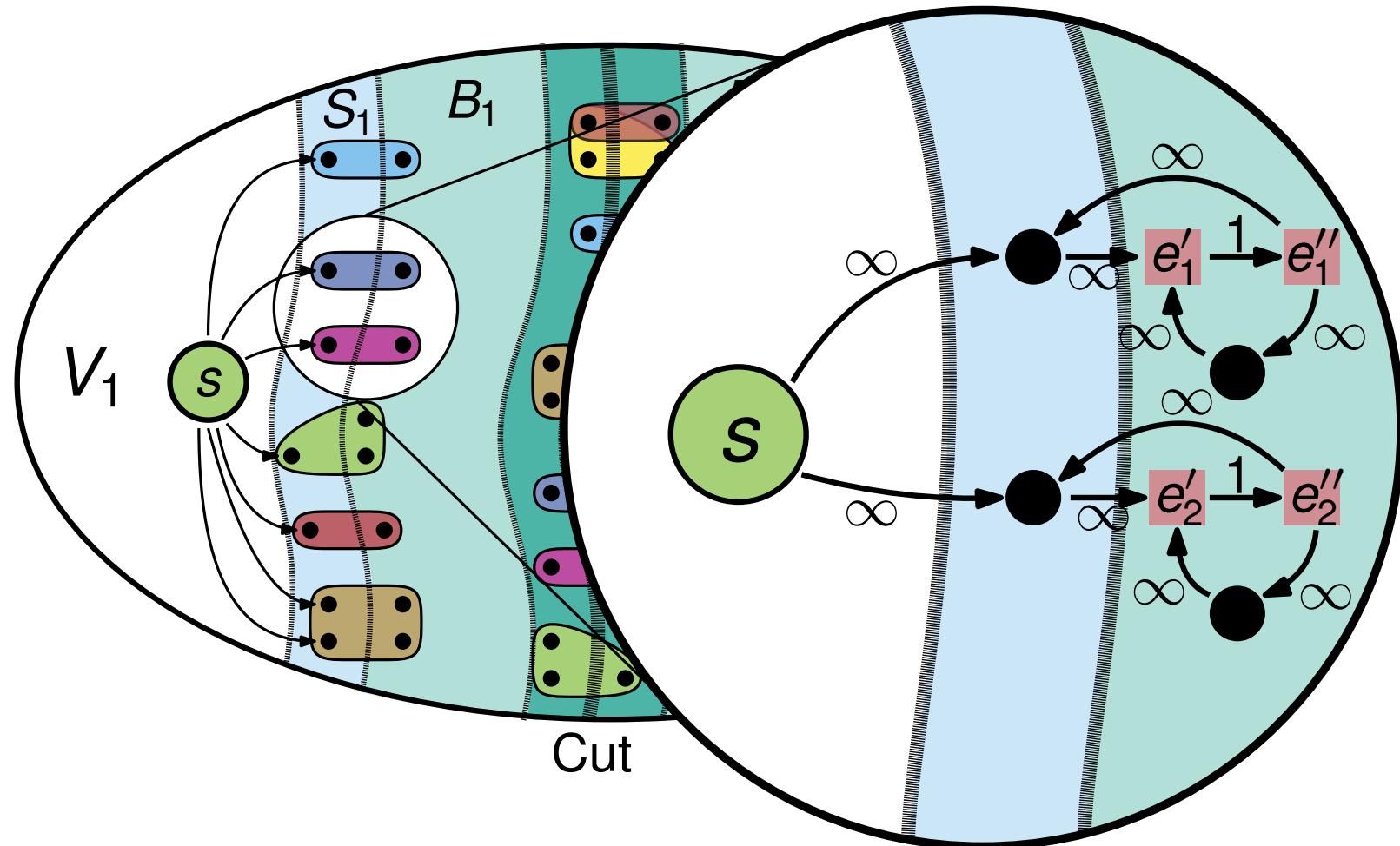
Modeling Approach in *KaHyPar*

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



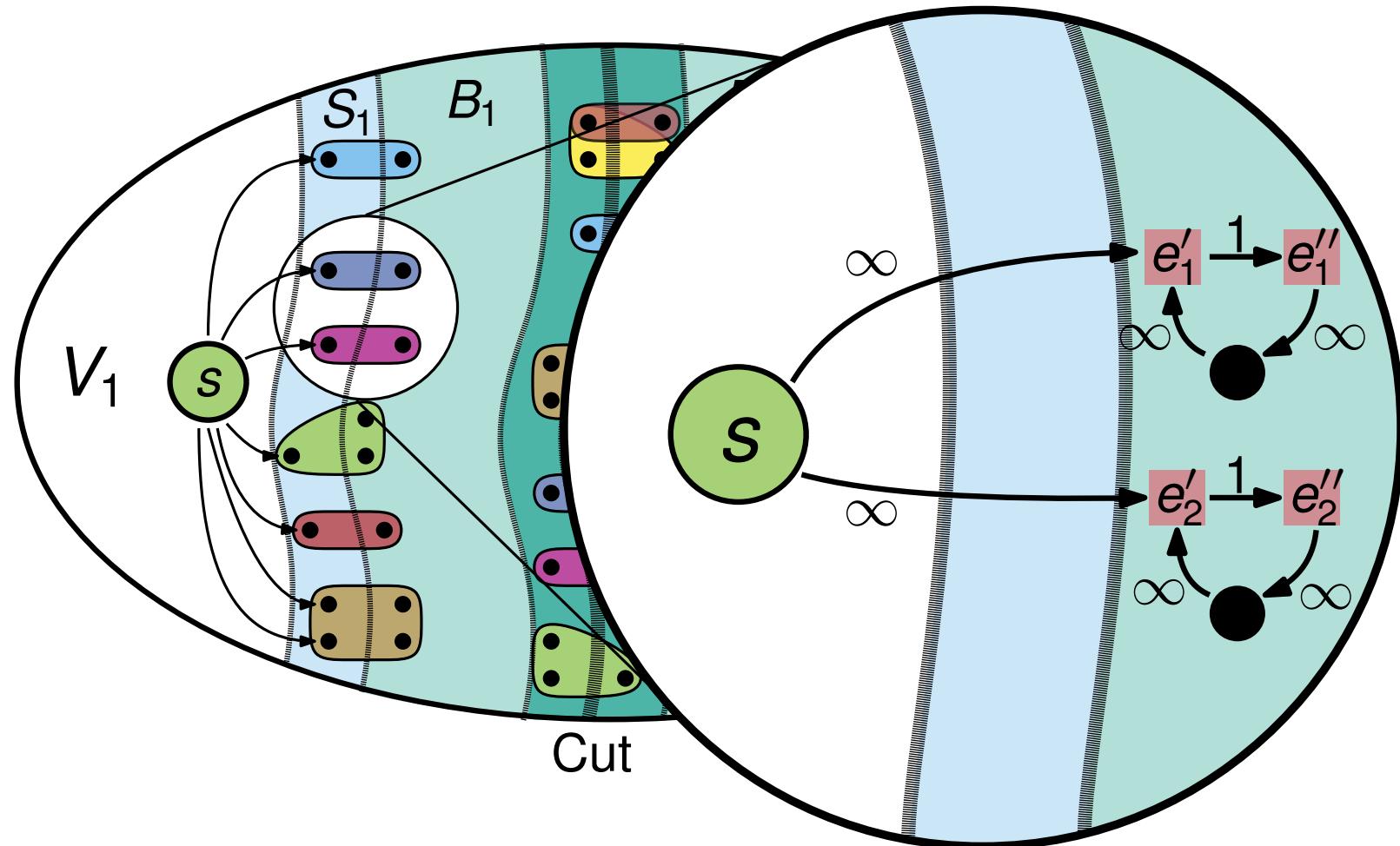
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*



Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*

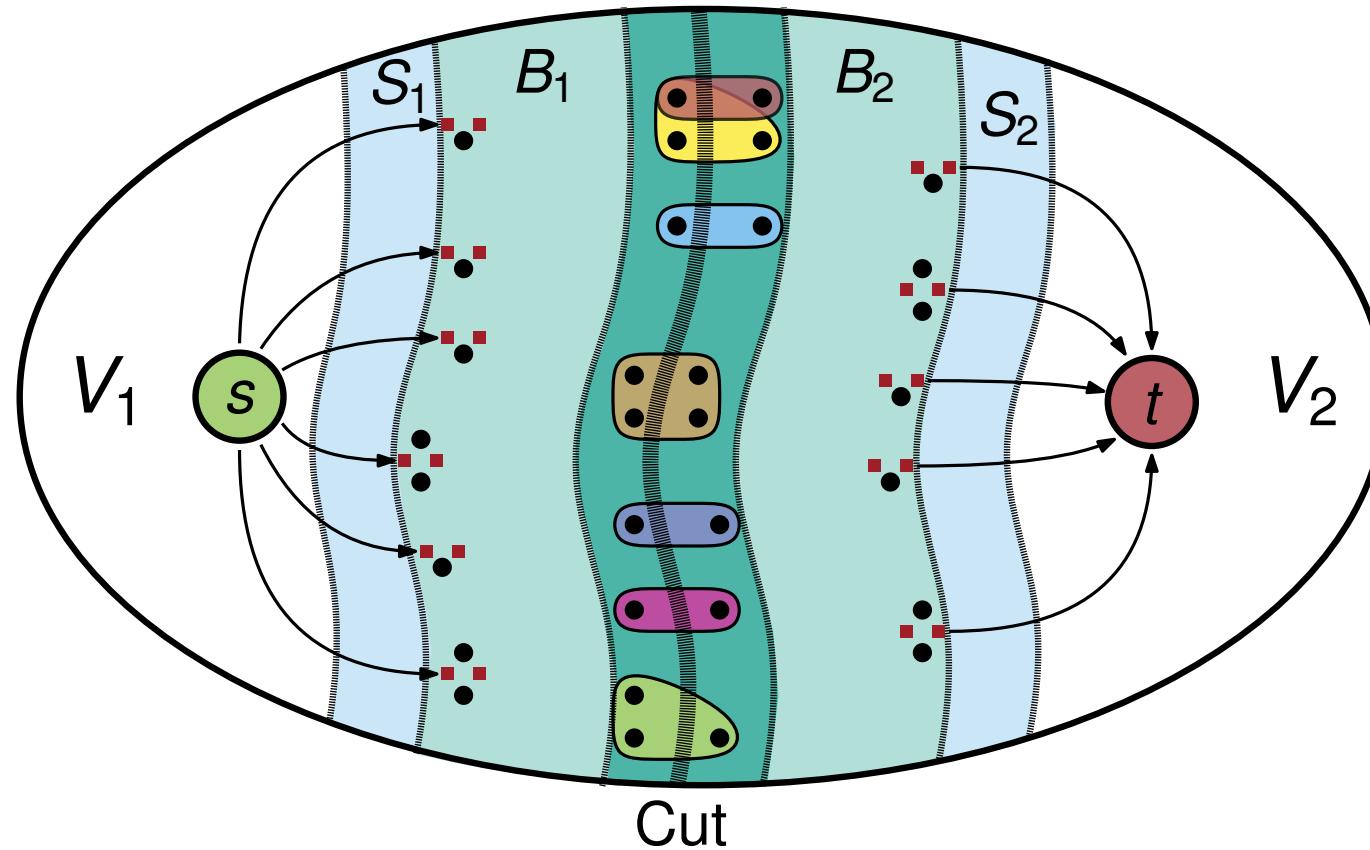


Optimized Flow Problem Modeling Approach

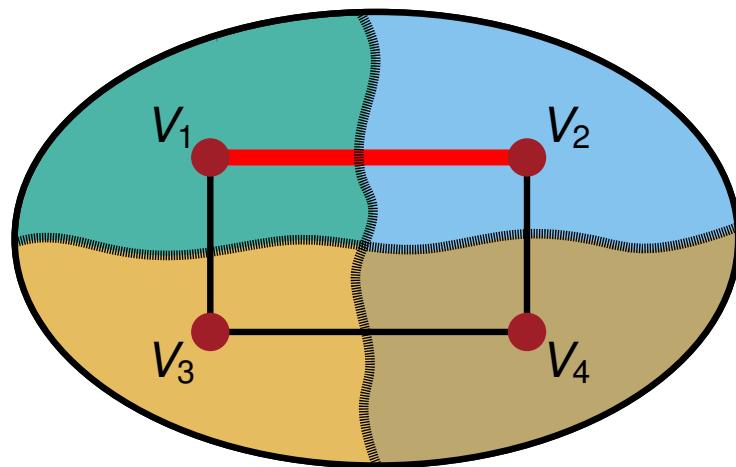
Modeling Approach in *KaHyPar*

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

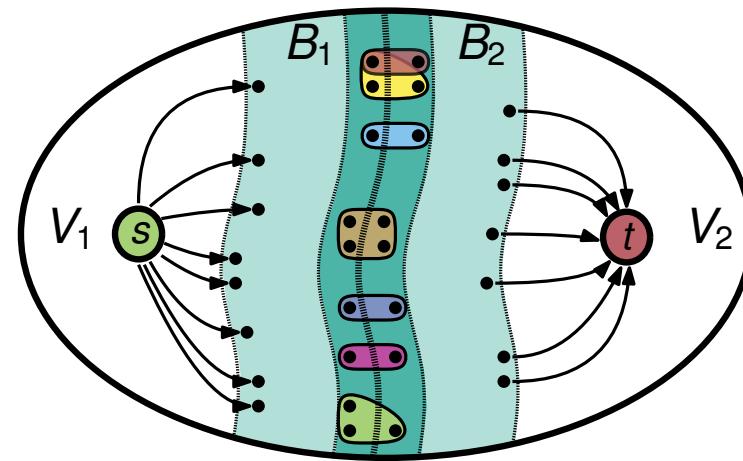
$$T = \{e'' \mid e \in I(S_2)\}$$



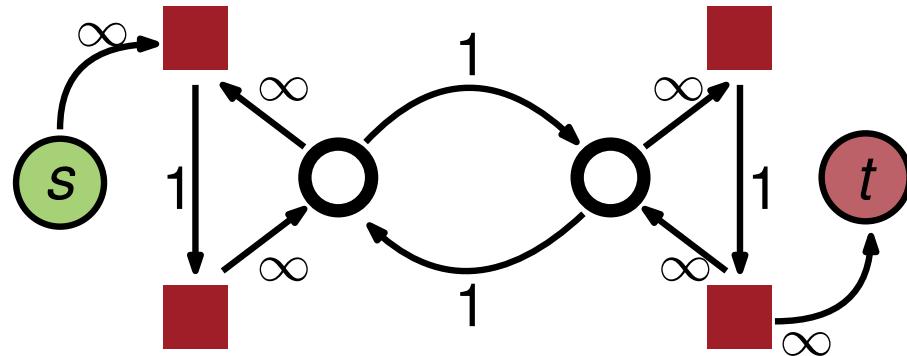
Our Flow-Based Refinement Framework



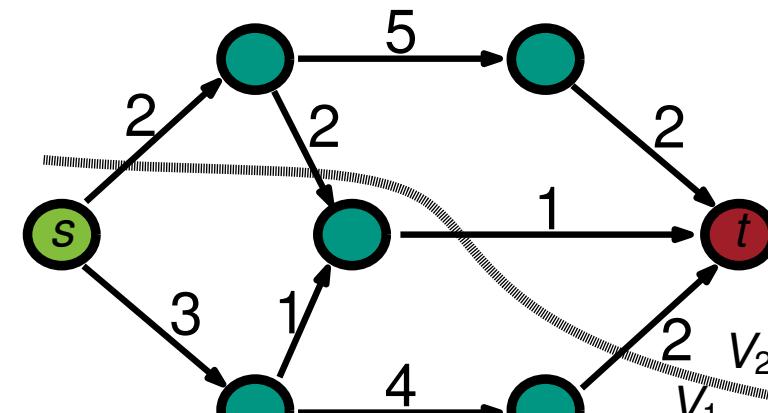
Select two adjacent blocks for refinement



Build Flow Problem

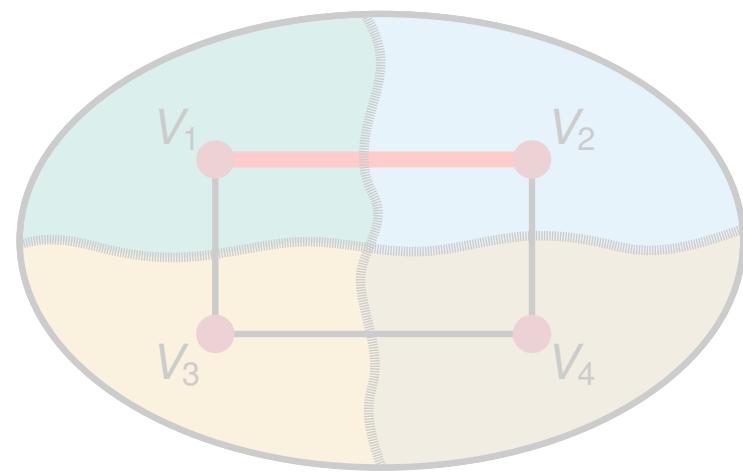


Solve Flow Problem

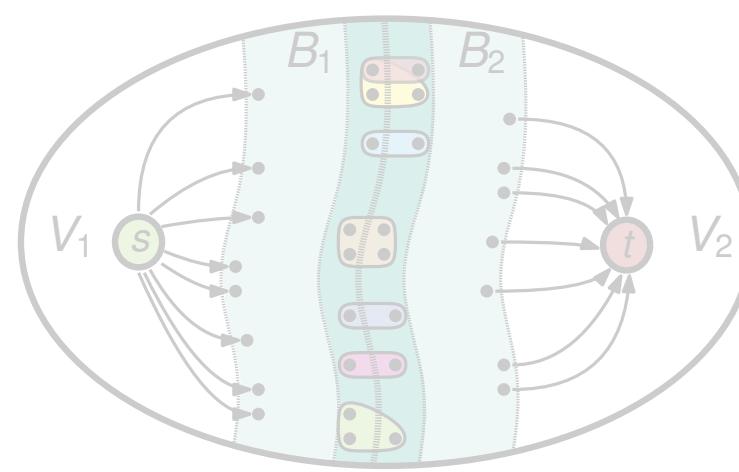


Find feasible minimum cut

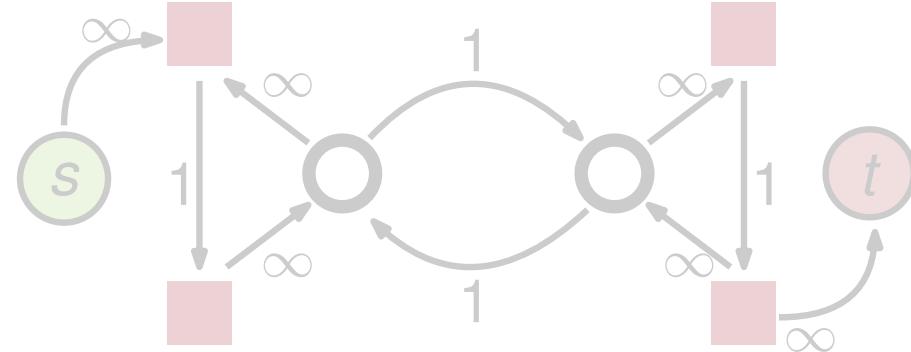
Our Flow-Based Refinement Framework



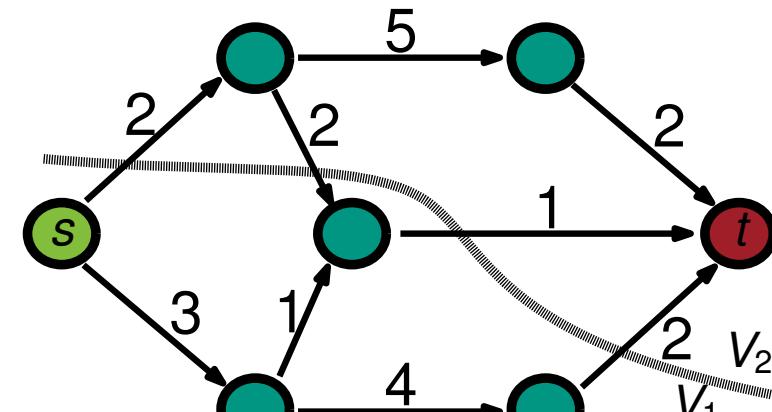
Select two adjacent blocks for refinement



Build Flow Problem



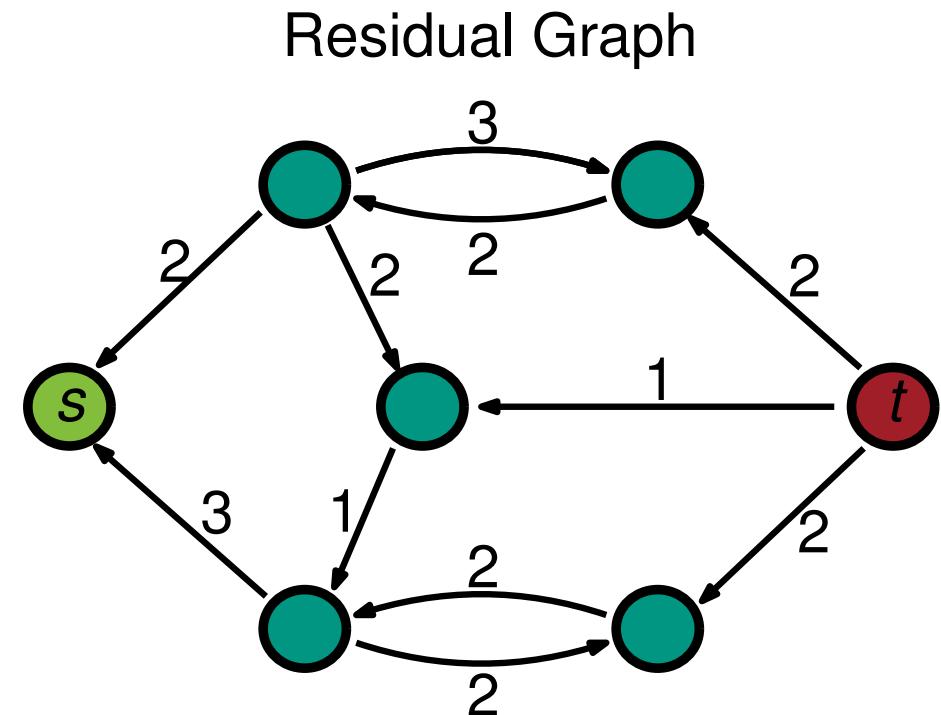
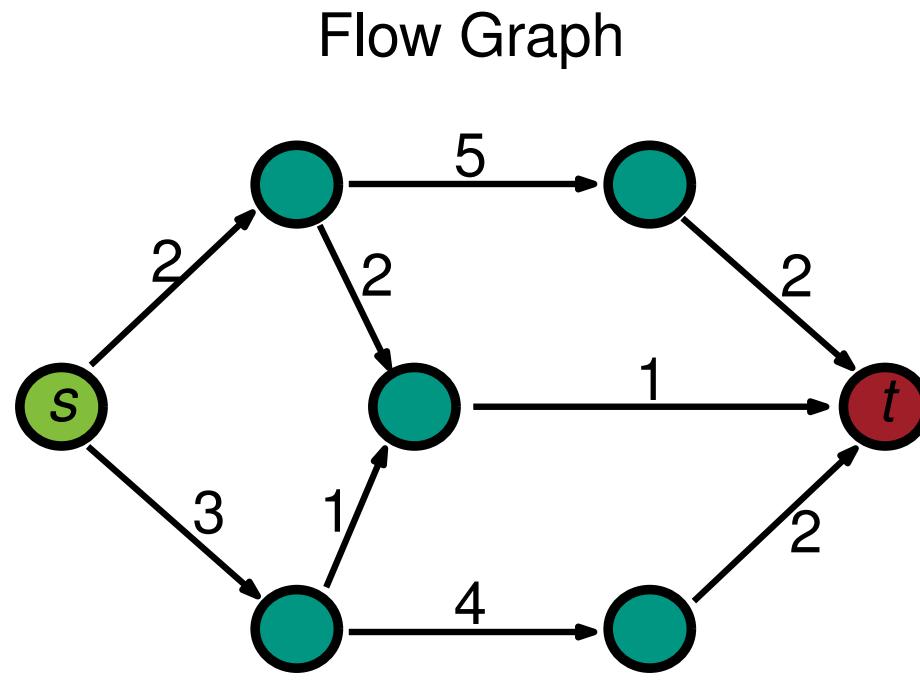
Solve Flow Problem



Find feasible minimum cut

Most Balanced Minimum Cut

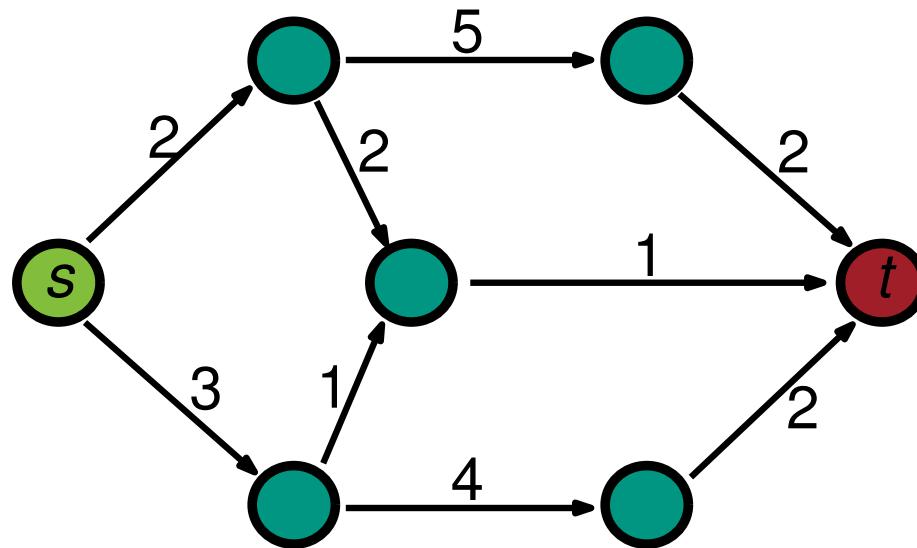
One maximum flow f has enough information to enumerate all minimum (s, t) -cuts



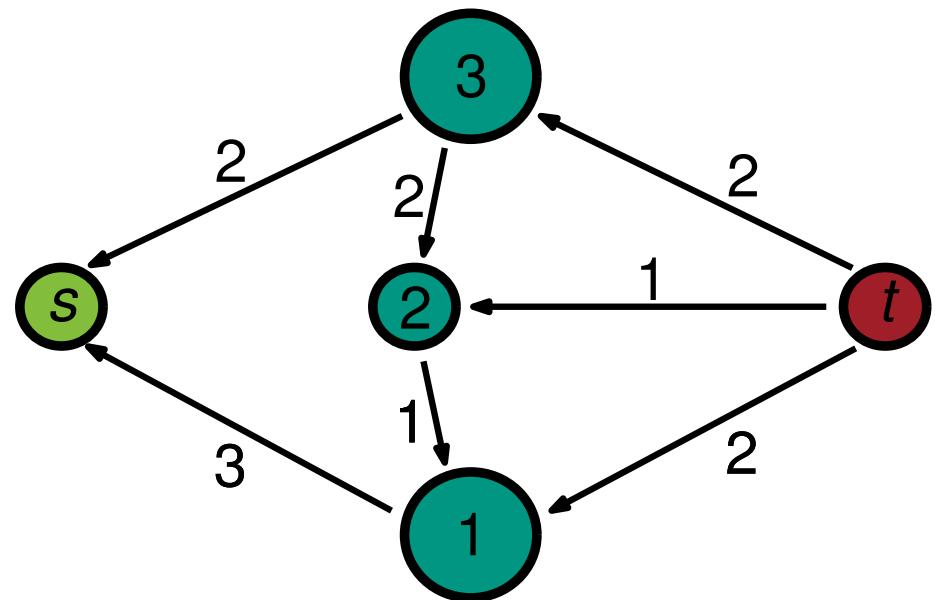
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC

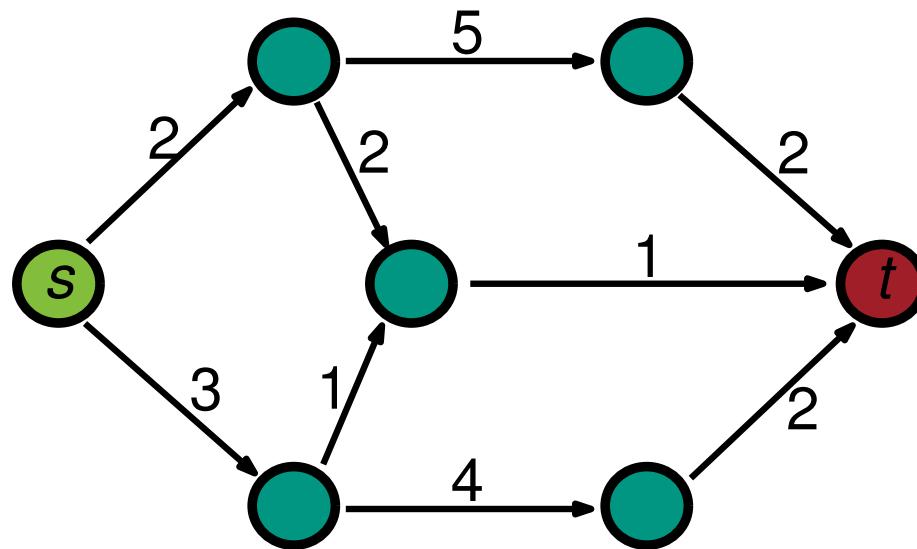


Contract all *strongly connected components* in the residual graph

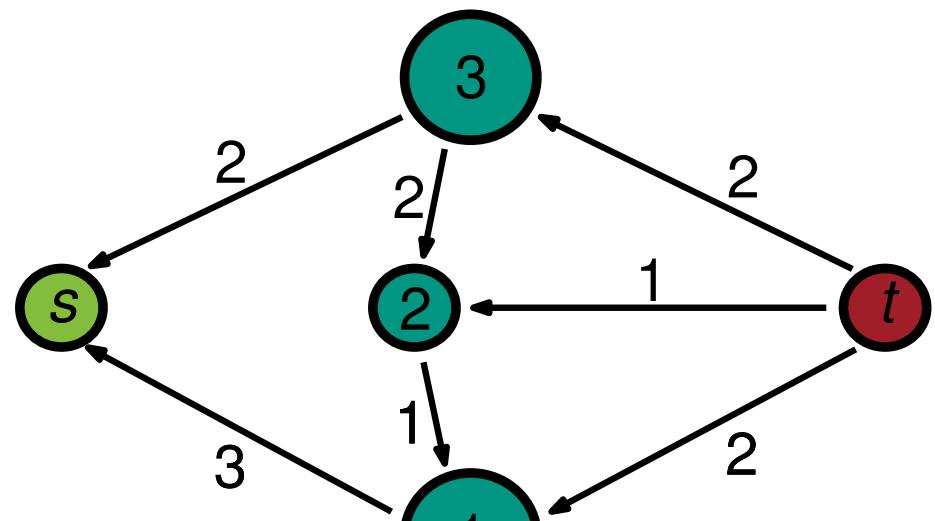
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC



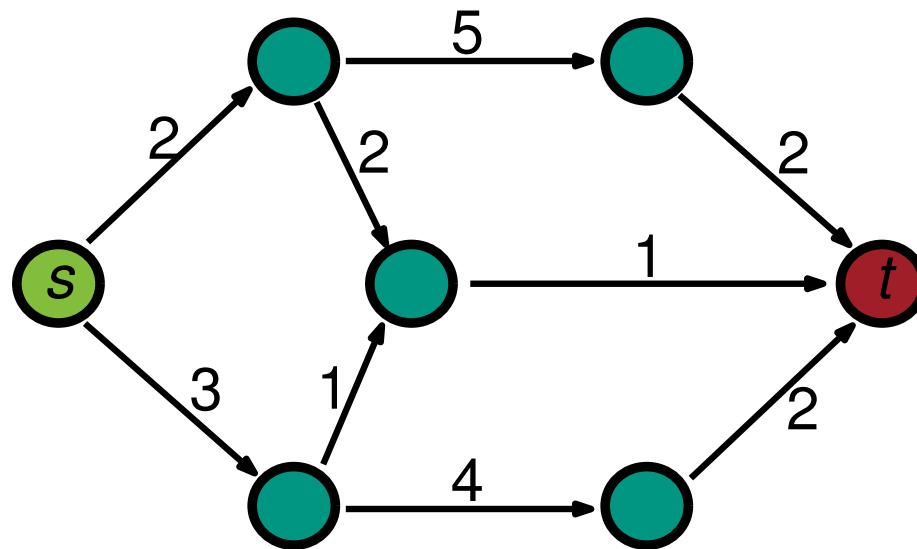
$$\langle t, 3, 2, 1, s \rangle$$

Find *topological order*

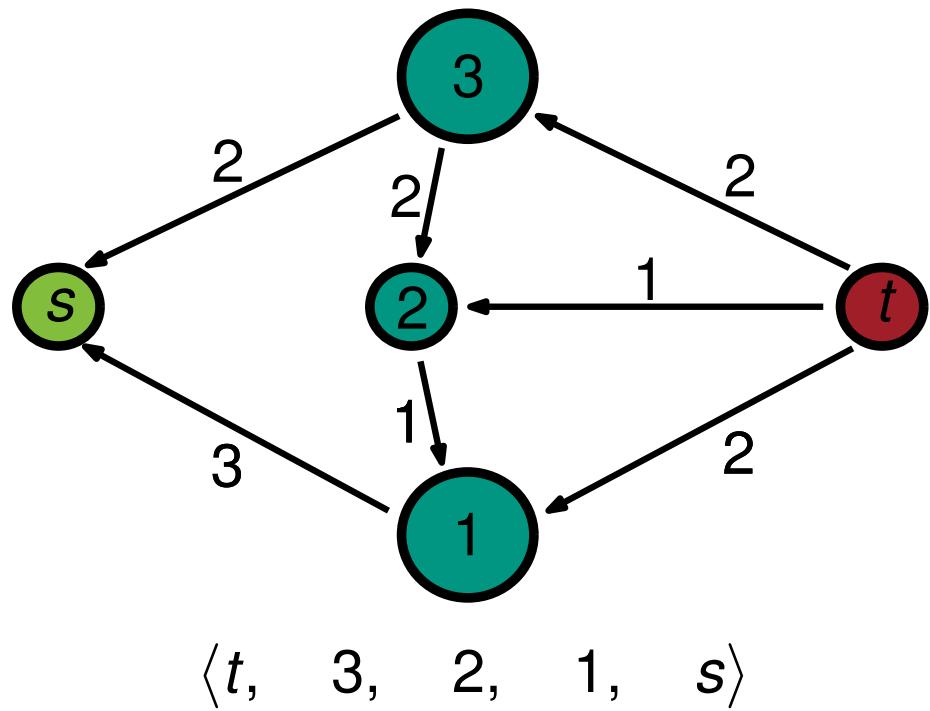
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC

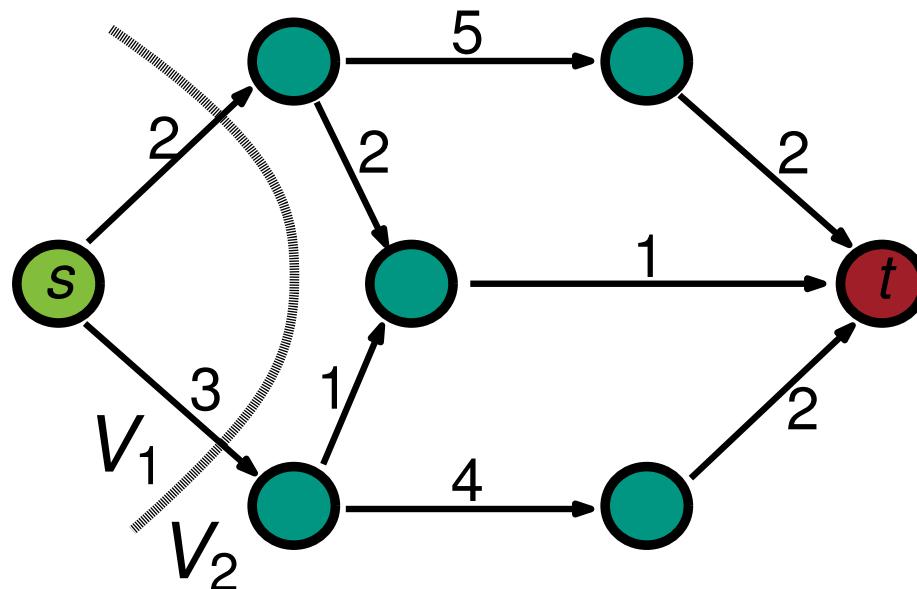


Sweep through **reverse** topological order

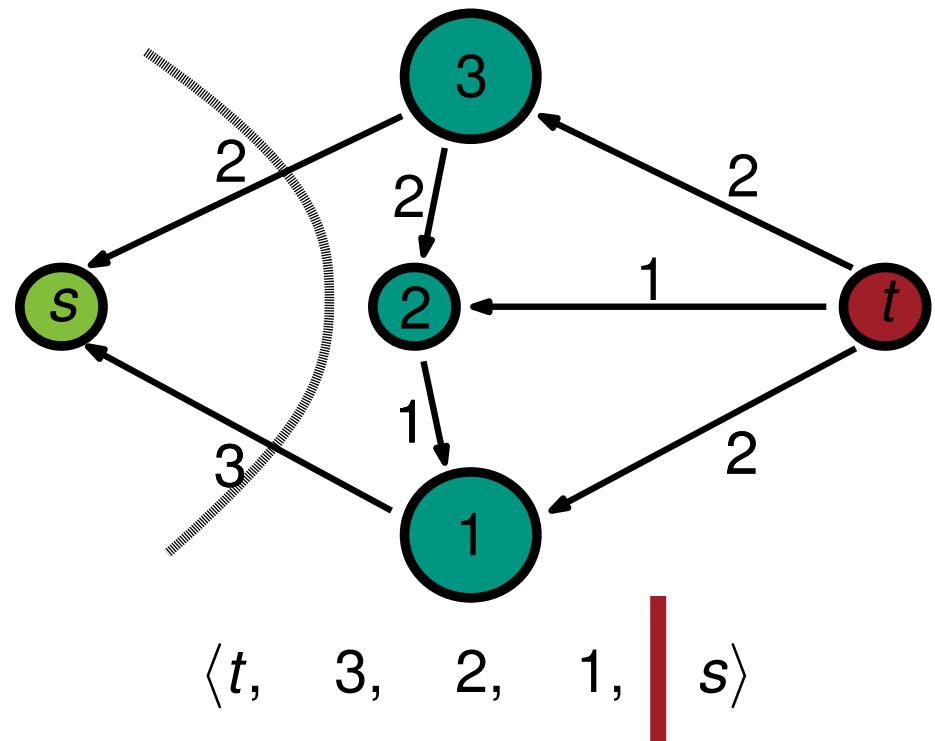
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC

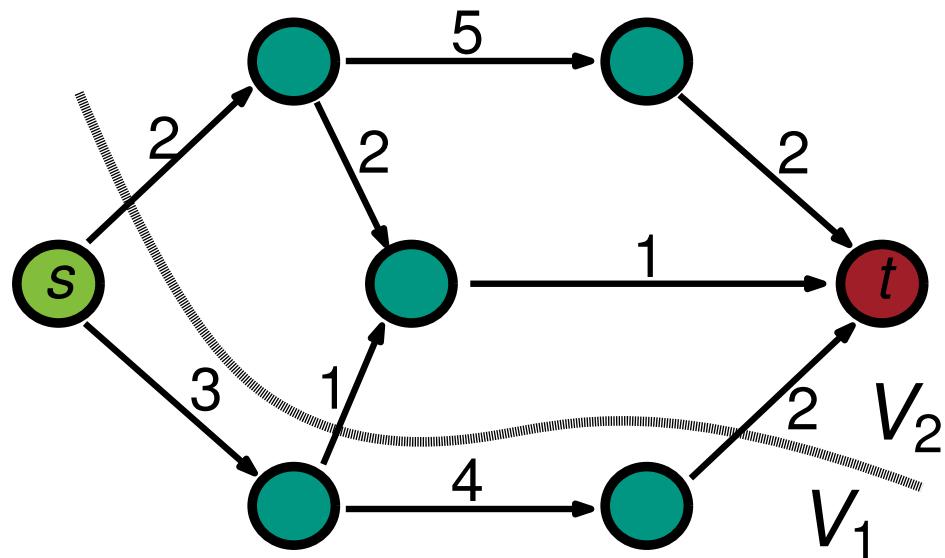


Sweep through **reverse** topological order

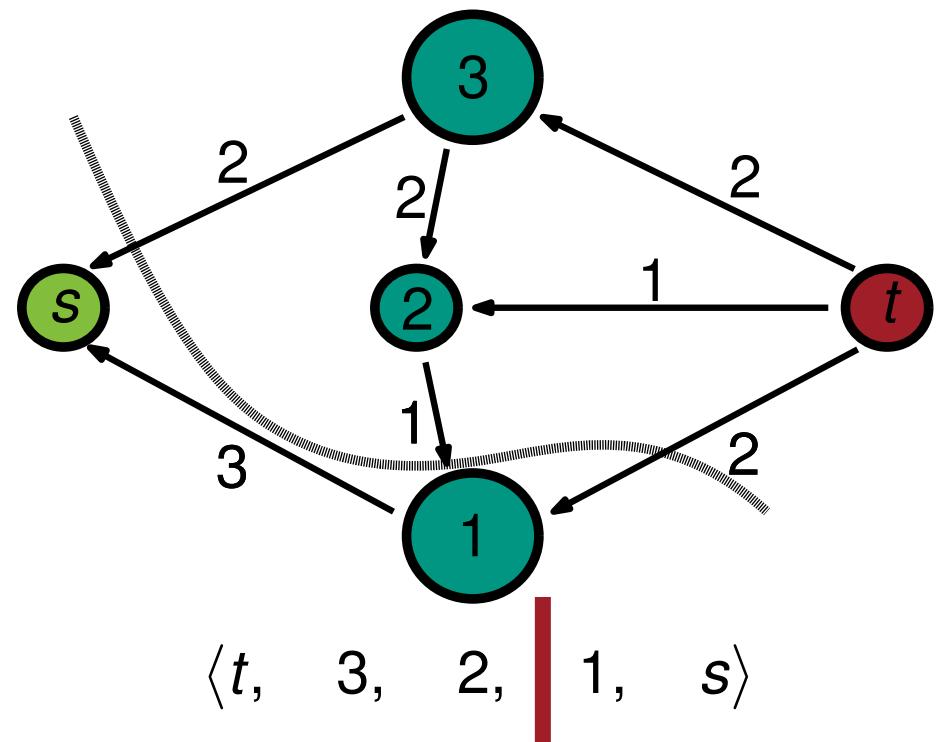
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC

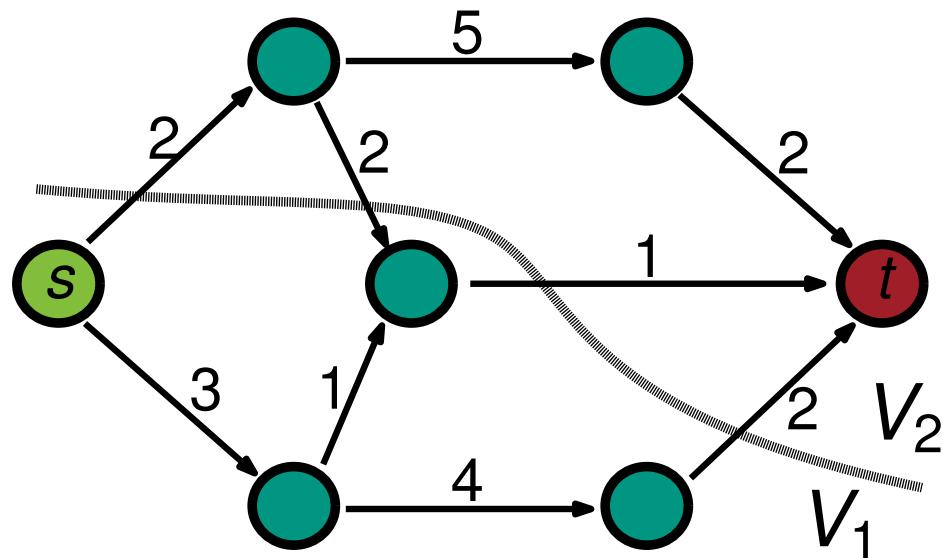


Sweep through **reverse** topological order

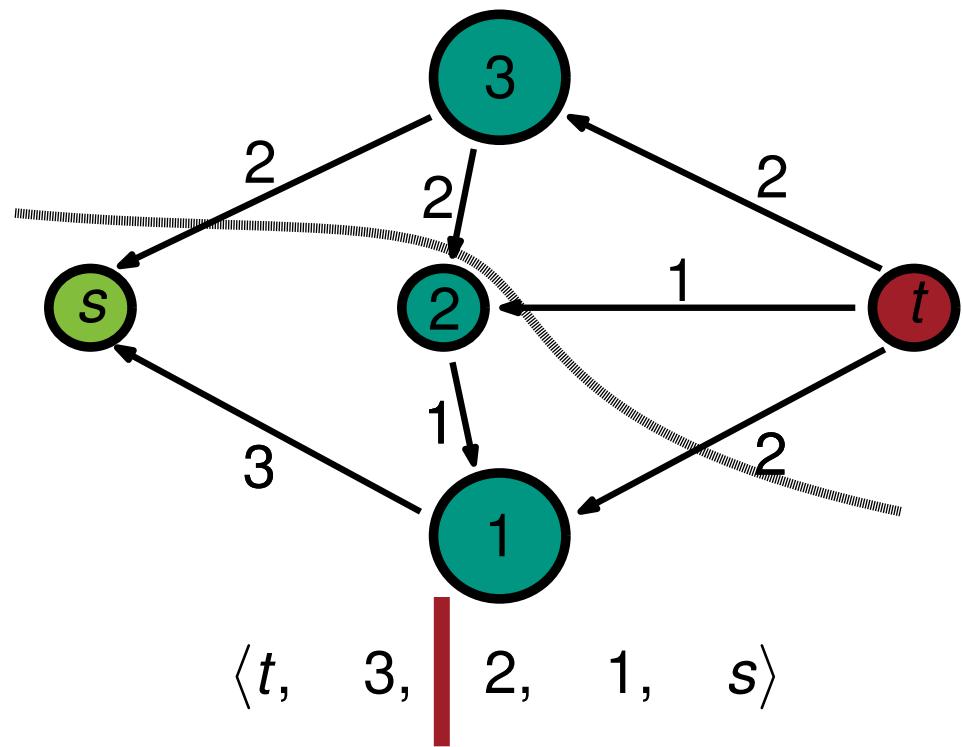
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC

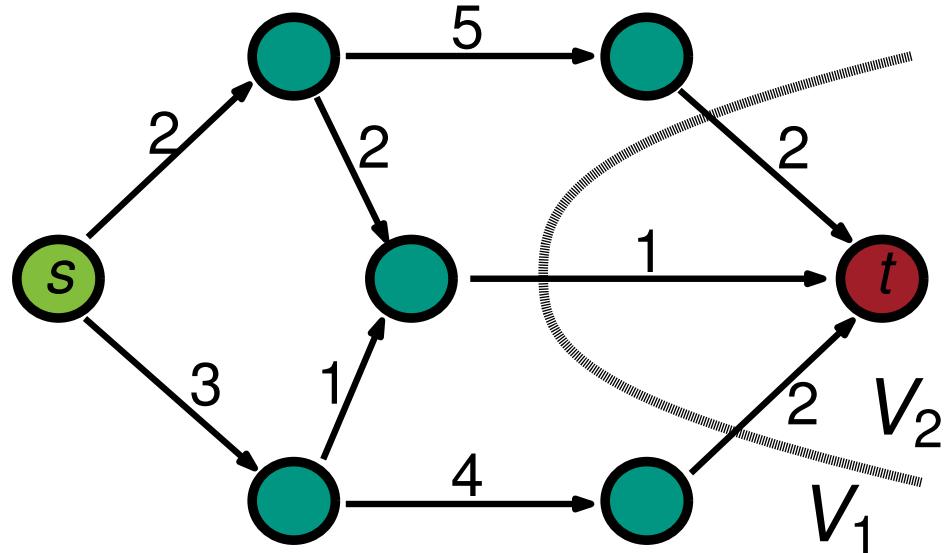


Sweep through **reverse** topological order

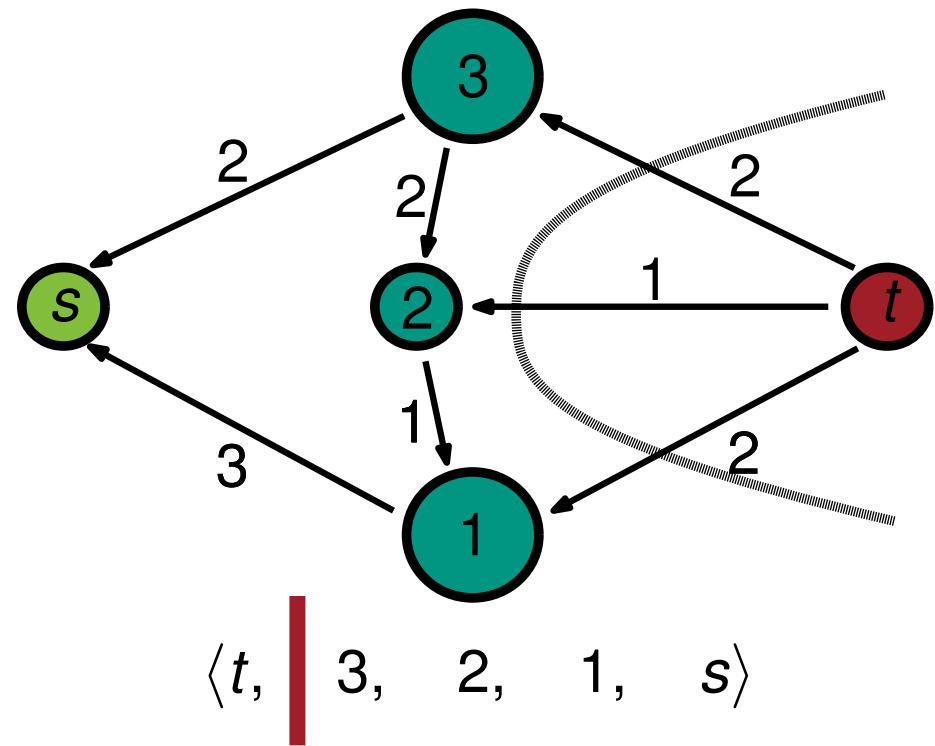
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t) -cuts

Flow Graph



Picard-Queryanne DAC



Sweep through **reverse** topological order

Integration into KaHyPar

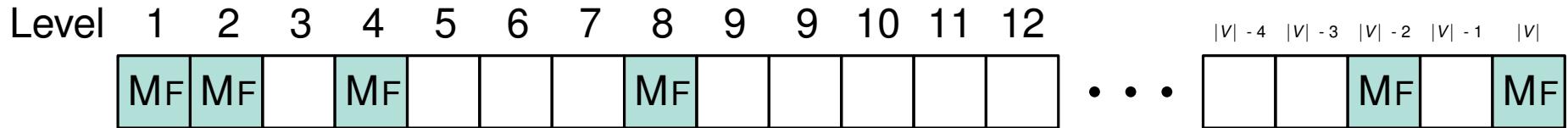
- KaHyPar is a n -level hypergraph partitioner

Integration into KaHyPar

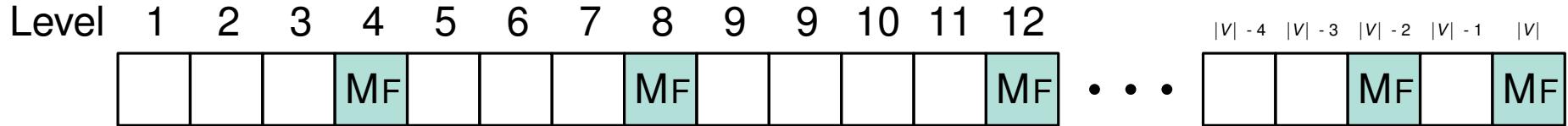
- KaHyPar is a n -level hypergraph partitioner

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 $i = \beta \cdot j$

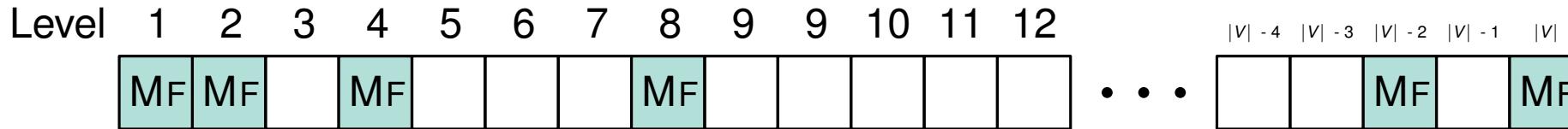


Integration into KaHyPar

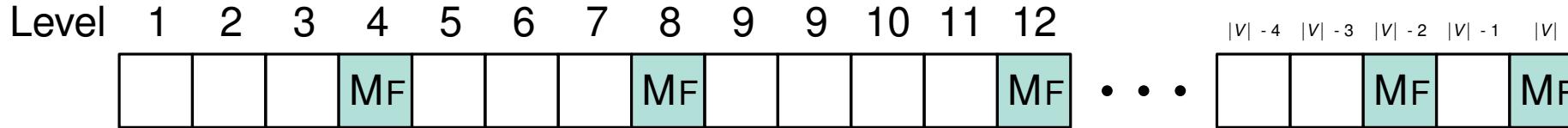
- KaHyPar is a n -level hypergraph partitioner

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 $i = \beta \cdot j$



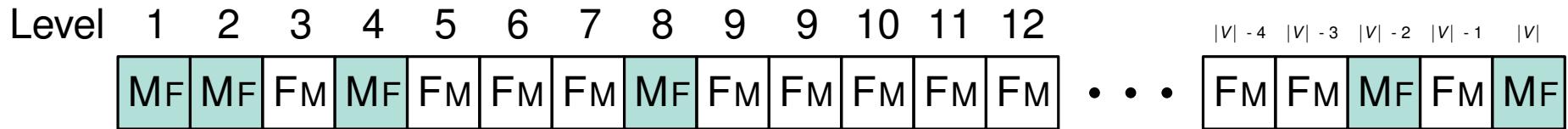
Note, each policy uses *flow-based refinement* on the **last level** 

Integration into KaHyPar

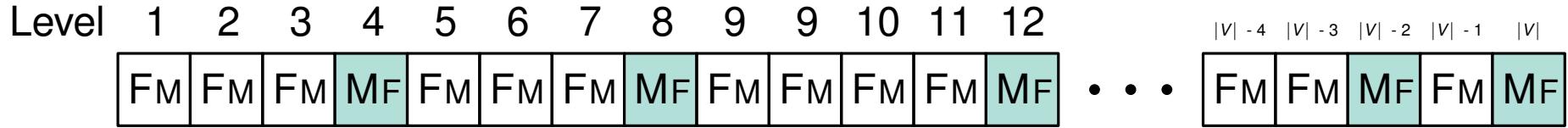
- KaHyPar is a n -level hypergraph partitioner

Flow Execution Policies

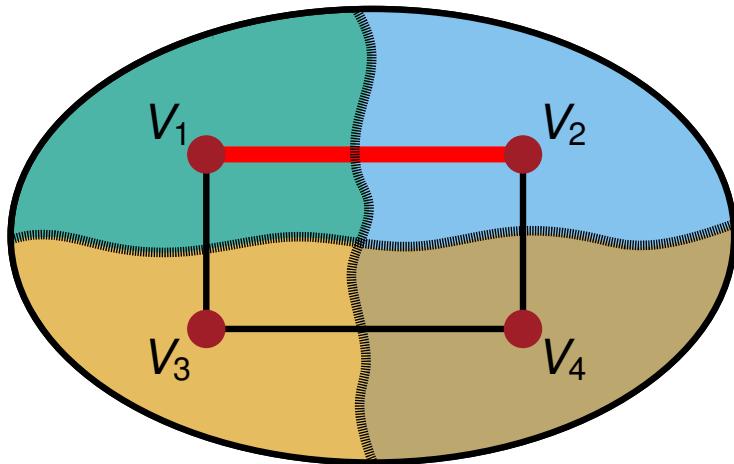
Exponential: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = 2^j$



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

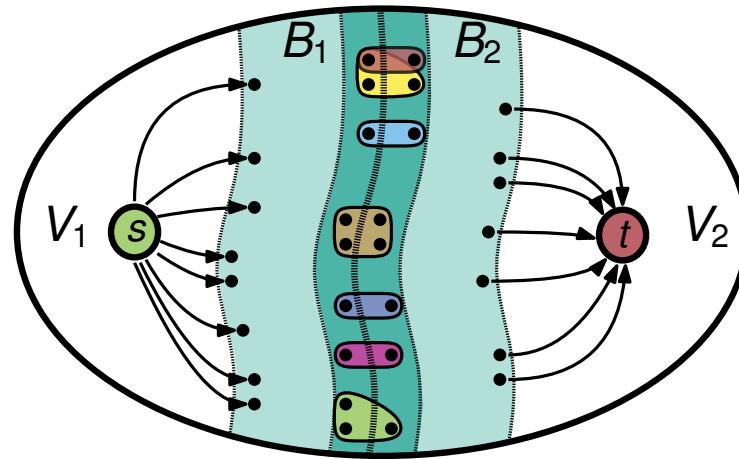


Speed-Up Heuristics



Active Block Scheduling

- (R1) If cut between two blocks is small (e.g. ≤ 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, 8×256 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, 8×256 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)

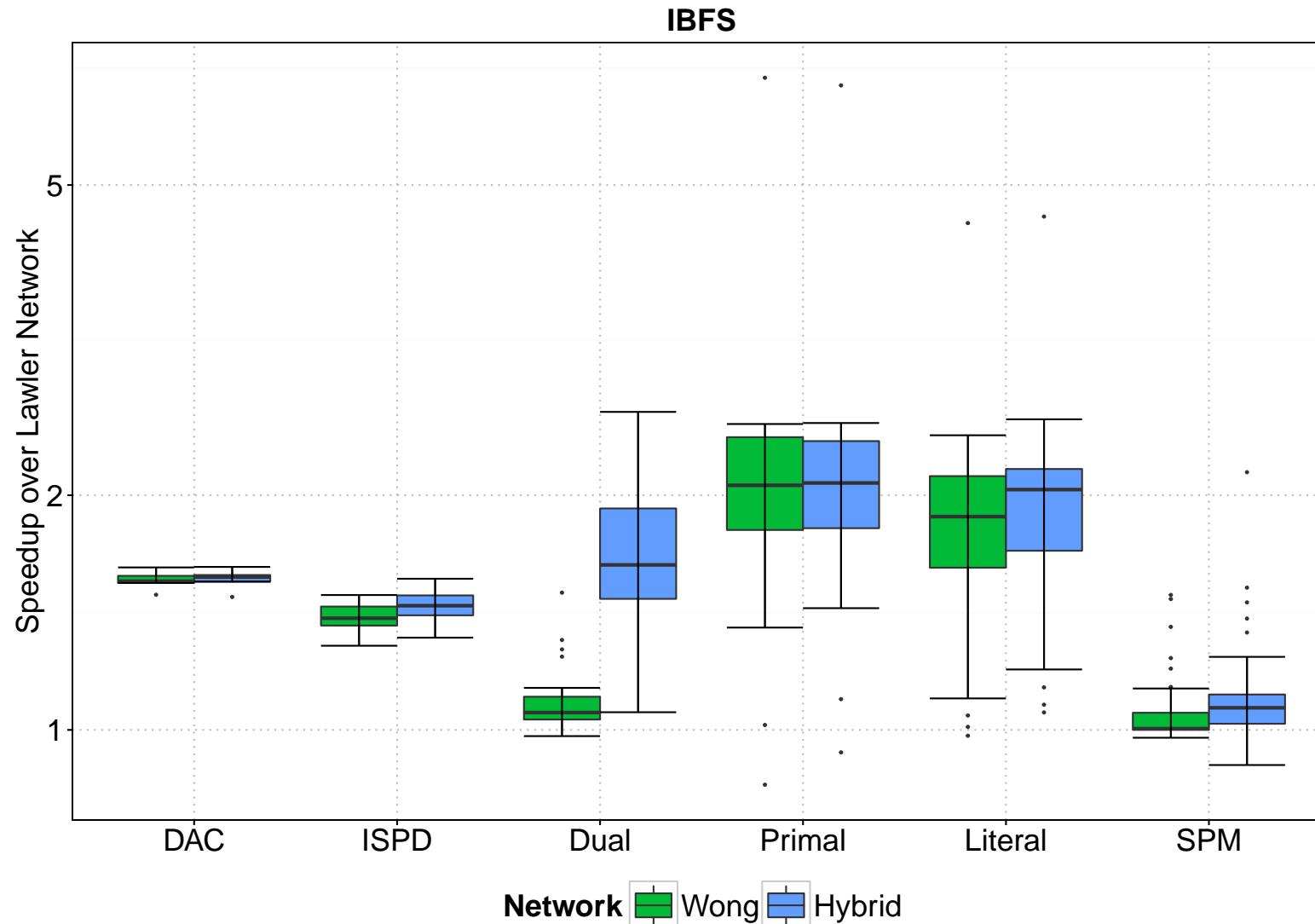
Benchmark Type

- 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.	$(+F,-M,-FM)$		$(+F,+M,-FM)$		$(+F,+M,+FM)$		CONSTANT128	
α'	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	−6.09	12.91	−5.60	13.40	0.23	15.37	0.53	55.75
2	−3.19	15.75	−2.07	16.74	0.74	18.06	1.09	87.93
4	−1.82	20.37	−0.19	21.88	1.21	22.49	1.61	144.42
8	−0.85	28.49	0.98	30.67	1.71	30.23	2.16	257.41
16	−0.19	43.32	1.75	46.66	2.21	43.53	2.69	498.29
Ref.	KaHyPar-CA	6373.88	13.73					

Flow Configuration

Improvement of **MBMC** more significantly for large α' :

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

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
α'	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	-6.09	12.91	-5.60	13.40	0.23	15.37	0.53	55.75
2	-3.19	15.75	-2.07	16.74	0.74	18.06	1.09	87.93
4	-1.82	20.37	-0.19	21.88	1.21	22.49	1.61	144.42
8	-0.85	28.49	0.98	30.67	1.71	30.23	2.16	257.41
16	-0.19	43.32	1.75	46.66	2.21	43.53	2.69	498.29
Ref.	KaHyPar-CA	6373.88	13.73					

Flow Configuration

Flows with FM are faster than flows on its own for large α'

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(-F,+M,+FM)		CONSTANT128	
α'	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	−6.09	12.91	−5.60	13.40	0.23	15.37	0.53	55.75
2	−3.19	15.75	−2.07	16.74	0.74	18.06	1.09	87.93
4	−1.82	20.37	−0.19	21.88	1.21	22.49	1.61	144.42
8	−0.85	28.49	0.98	30.67	1.71	30.23	2.16	257.41
16	−0.19	43.32	1.75	46.66	2.21	43.53	2.69	498.29
Ref.	KaHyPar-CA	6373.88	13.73					

Flow Configuration

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

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128		
	α'	Avg [%]	$t[s]$	α'	Avg [%]	$t[s]$	α'	Avg [%]	$t[s]$
1	−6.09	12.91		−5.60	13.40	0.23	15.37	0.53	55.75
2	−3.19	15.75		−2.07	16.74	0.74	18.06	1.09	87.93
4	−1.82	20.37		−0.19	21.88	1.21	22.49	1.61	144.42
8	−0.85	28.49		0.98	30.67	1.71	30.23	2.16	257.41
16	−0.19	43.32		1.75	46.66	2.21	43.53	2.69	498.29
Ref.	KaHyPar-CA		6373.88	13.73					

Flow Configuration

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
α'	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	−6.09	12.91	−5.60	13.40	0.23	15.37	0.53	55.75
2	−3.19	15.75	−2.07	16.74	0.74	18.06	1.09	87.93
4	−1.82	20.37	−0.19	21.88	1.21	22.49	1.61	144.42
8	−0.85	28.49	0.98	30.67	1.71	30.23	2.16	257.41
16	−0.19	43.32	1.75	46.66	2.21	43.53	2.69	498.29
Ref.	KaHyPar-CA	6373.88	13.73					

Flow Configuration

α'	Config.	(+F,-M,-FM)		(+F,+M,-FM)	
		KaFFPa	Our	KaFFPa	Our
1		-15.48	-6.10	-15.26	-5.62
2		-10.50	-3.20	-10.12	-2.08
4		-5.98	-1.82	-5.08	-0.20
8		-3.22	-0.85	-1.64	0.98
16		-1.52	-0.20	0.51	1.75
Ref.		KaHyPar-CA		6373.88	

Flow Configuration

Quality improvement around 5% to 10% for small α'

α'	Config.	(+F,-M,-FM)		(+F,+M,-FM)	
		KaFFPa	Our	KaFFPa	Our
1		-15.48	-6.10	-15.26	-5.62
2		-10.50	-3.20	-10.12	-2.08
4		-5.98	-1.82	-5.08	-0.20
8		-3.22	-0.85	-1.64	0.98
16		-1.52	-0.20	0.51	1.75
Ref.		KaHyPar-CA		6373.88	

Flow Configuration

Quality improvement around 1.5% to 2.5% for large α'

α'	Config.	(+F,-M,-FM)		(+F,+M,-FM)	
		KaFFPa	Our	KaFFPa	Our
1		-15.48	-6.10	-15.26	-5.62
2		-10.50	-3.20	-10.12	-2.08
4		-5.98	-1.82	-5.08	-0.20
8		-3.22	-0.85	-1.64	0.98
16		-1.52	-0.20	0.51	1.75
Ref.		KaHyPar-CA		6373.88	

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 _(R1)	-2.41	-2.06	33.89	63.15
KaHyPar-MF _(R1,R2)	-2.40	-2.05	28.52	57.78
KaHyPar-MF _(R1,R2,R3)	-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 _(R1)	-2.41	-2.06	33.89	63.15
KaHyPar-MF _(R1,R2)	-2.40	-2.05	28.52	57.78
KaHyPar-MF _(R1,R2,R3)	-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 _(R1)	-2.41	-2.06	33.89	63.15
KaHyPar-MF _(R1,R2)	-2.40	-2.05	28.52	57.78
KaHyPar-MF _(R1,R2,R3)	-2.41	-2.06	21.23	50.49

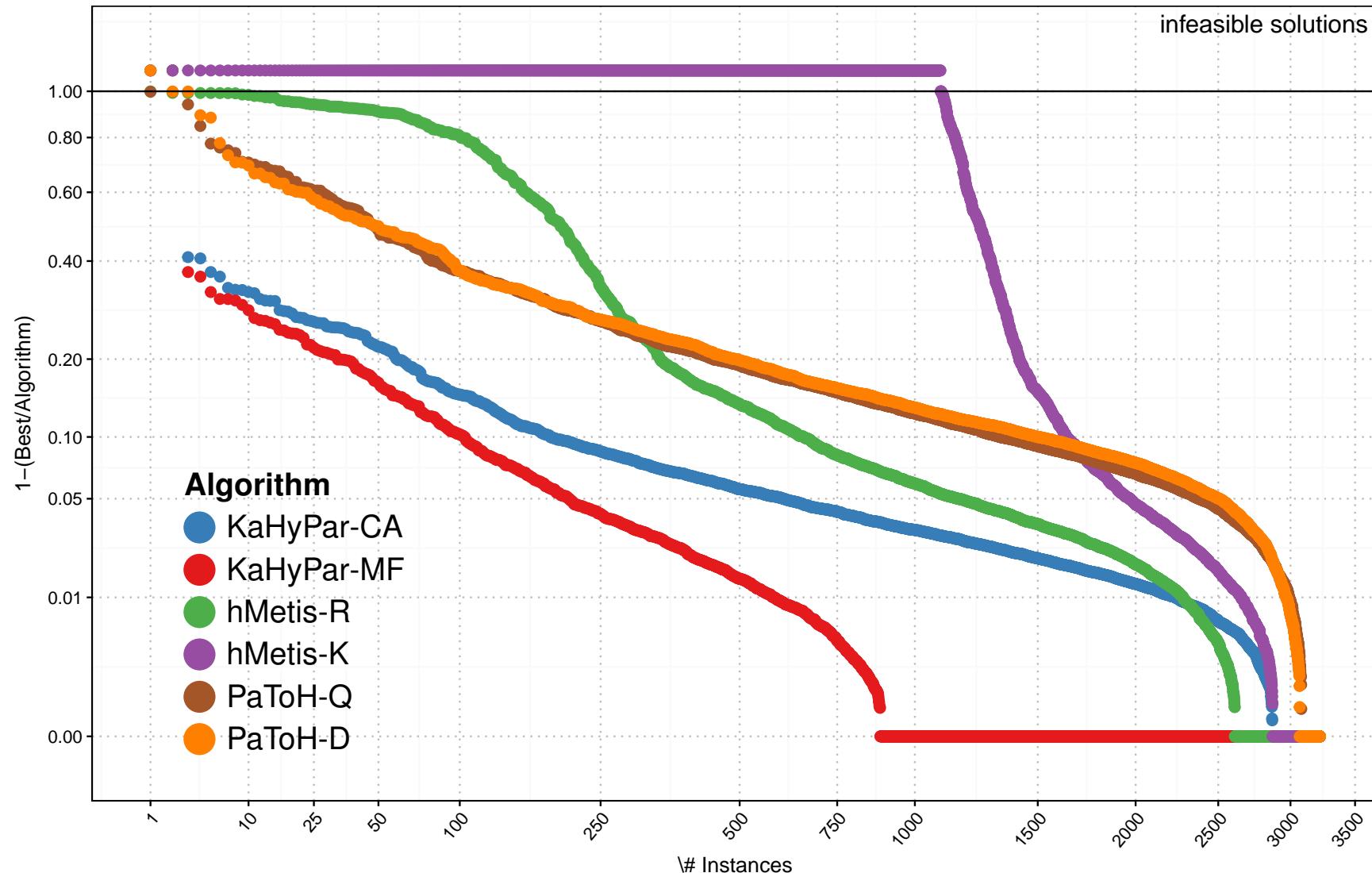
Speed-up by a factor of 2

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 _(R1)	-2.41	-2.06	33.89	63.15
KaHyPar-MF _(R1,R2)	-2.40	-2.05	28.52	57.78
KaHyPar-MF _(R1,R2,R3)	-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

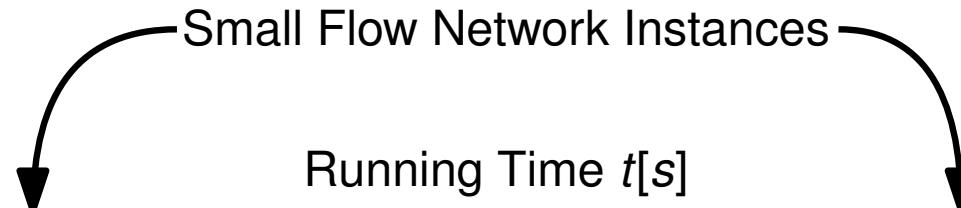
Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	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

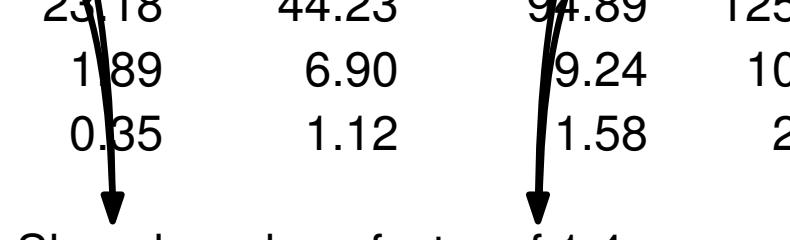
Small Flow Network Instances



Running Time $t[s]$

Algorithm	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

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

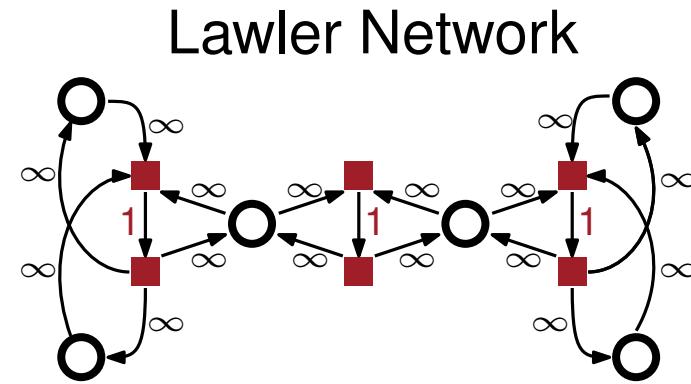
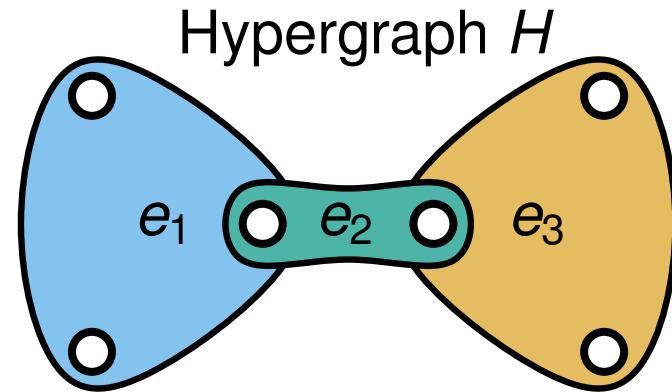
Large Flow Network Instances

Slow-down by a factor of 1.85

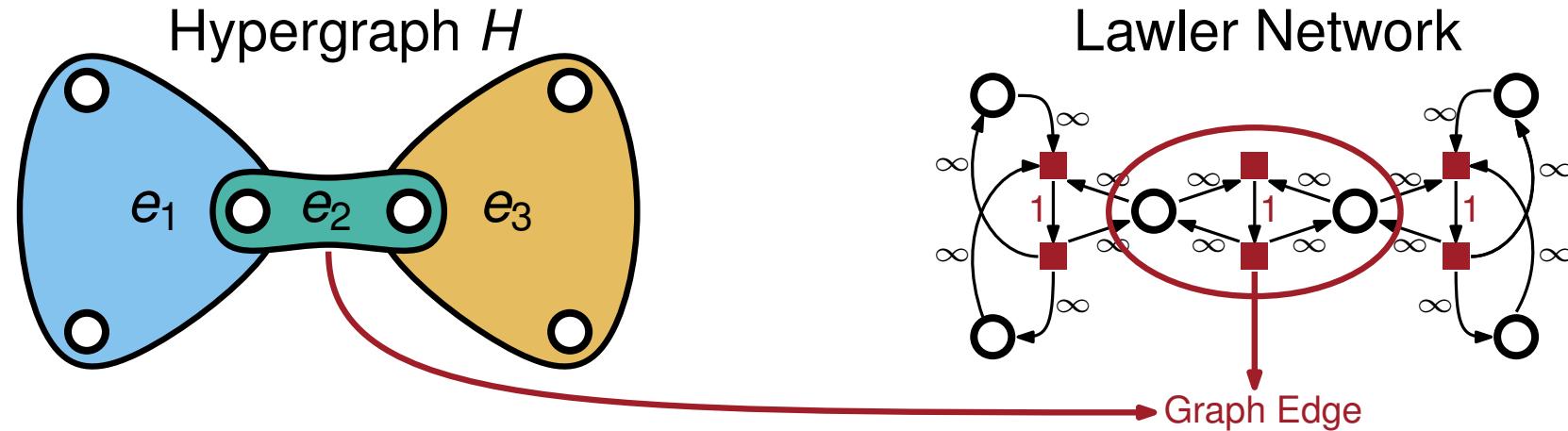
Conclusion

Appendix

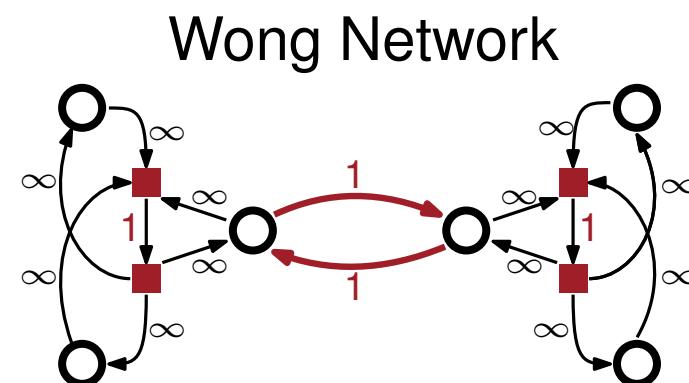
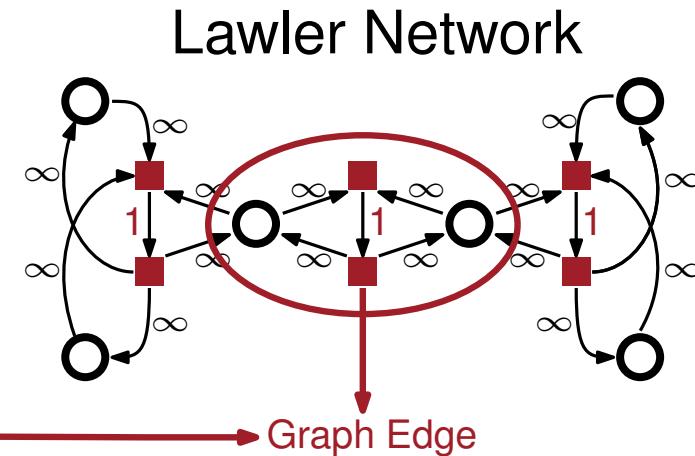
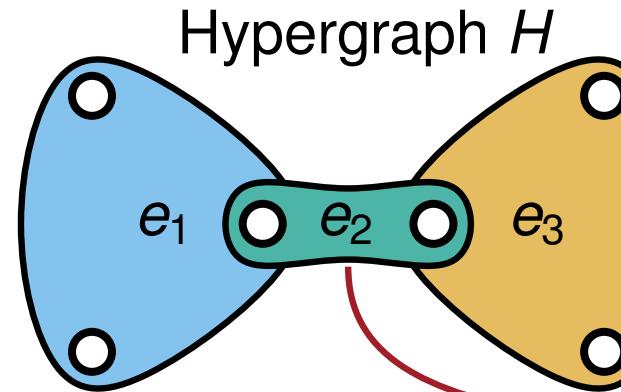
Hypergraph Flow Network - Graph Edges



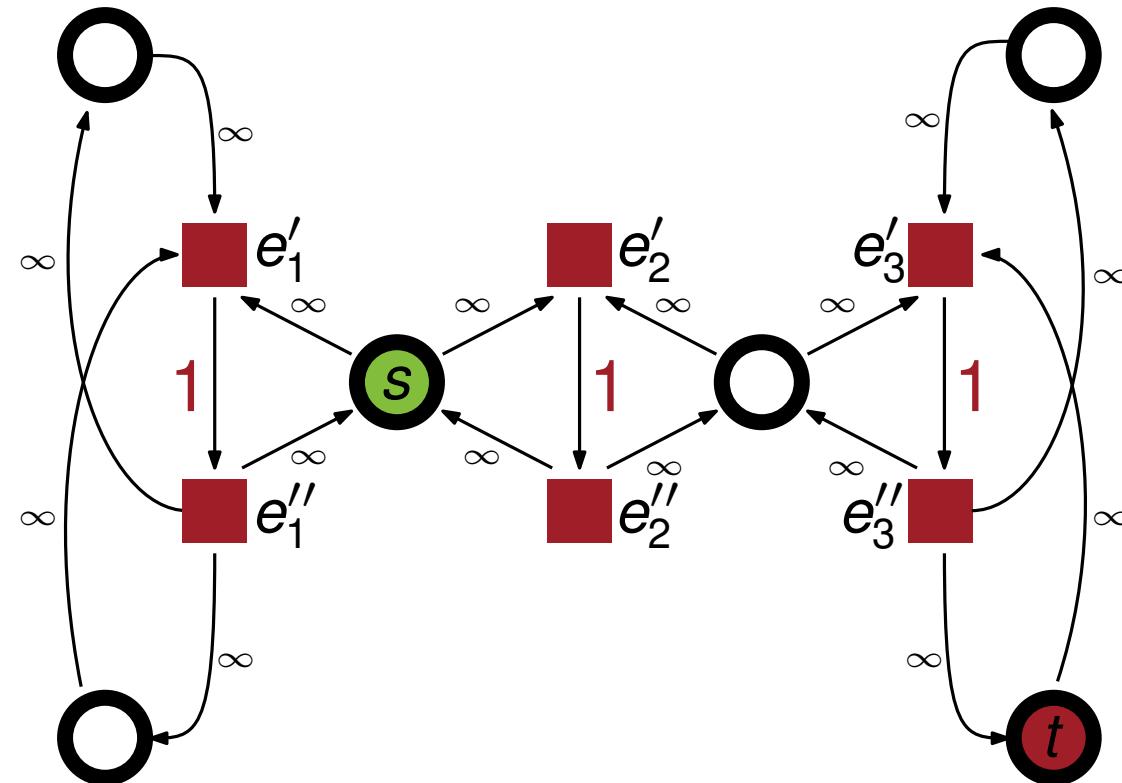
Hypergraph Flow Network - Graph Edges



Hypergraph Flow Network - Graph Edges

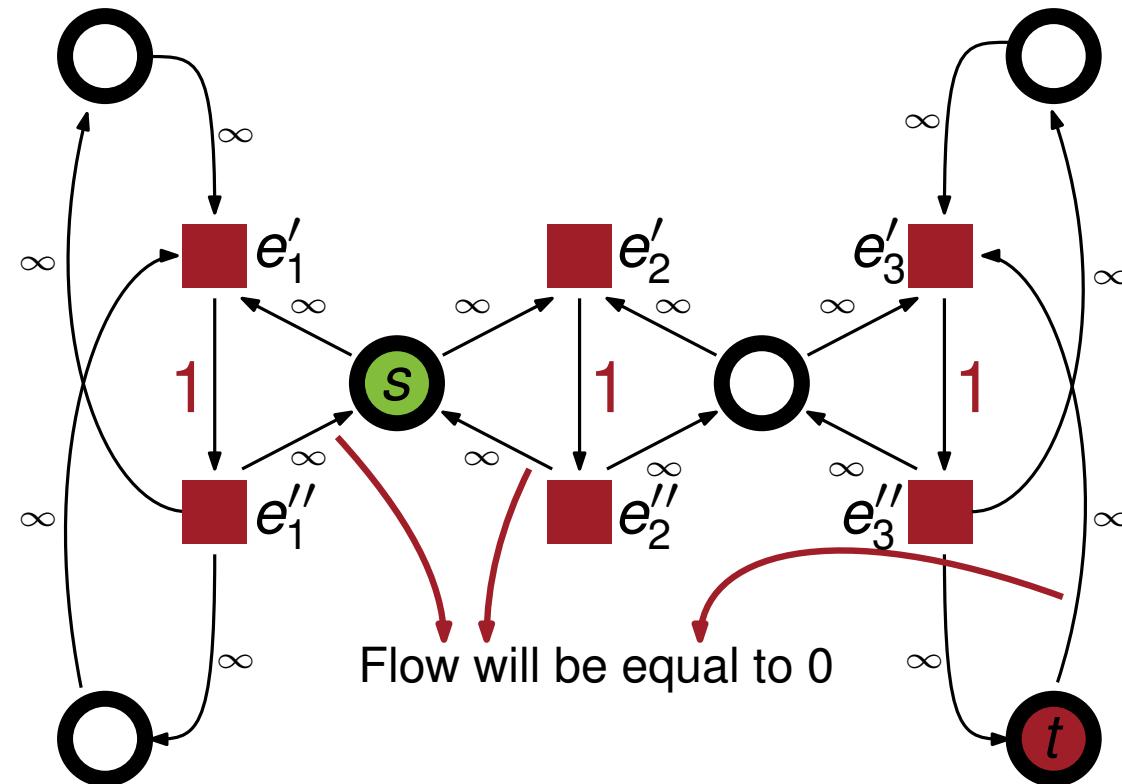


Removing Source and Sink Vertices



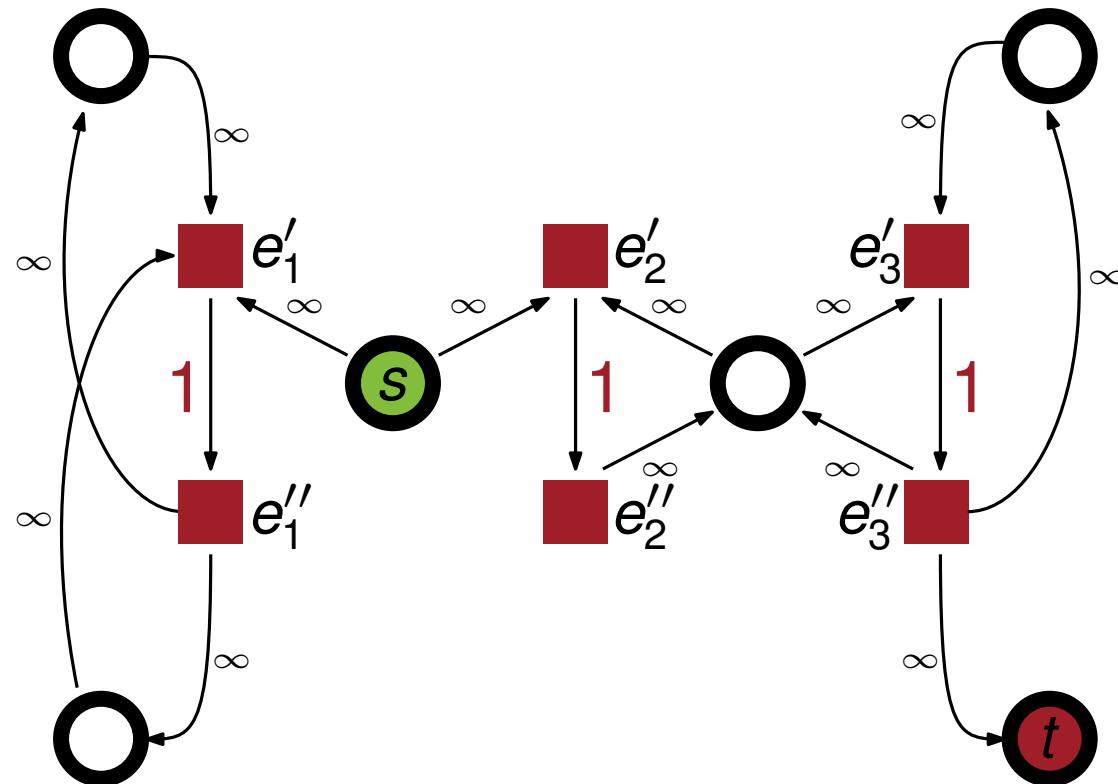
Lawler Network

Removing Source and Sink Vertices



Lawler Network

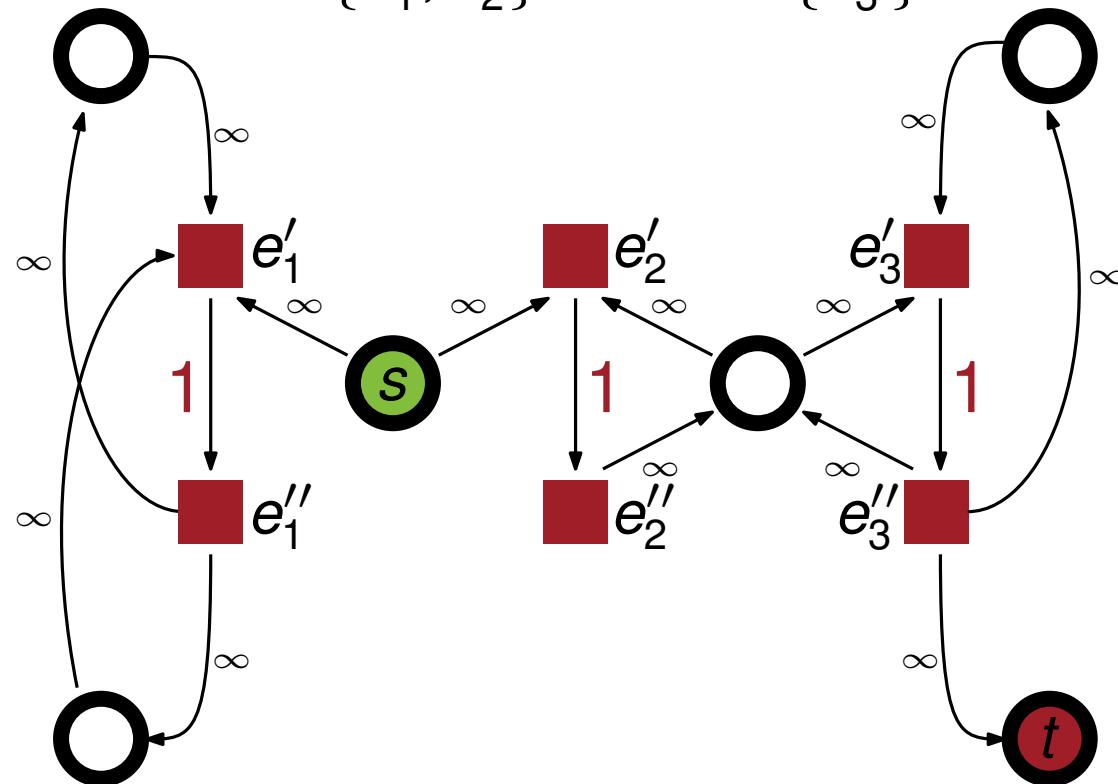
Removing Source and Sink Vertices



Lawler Network

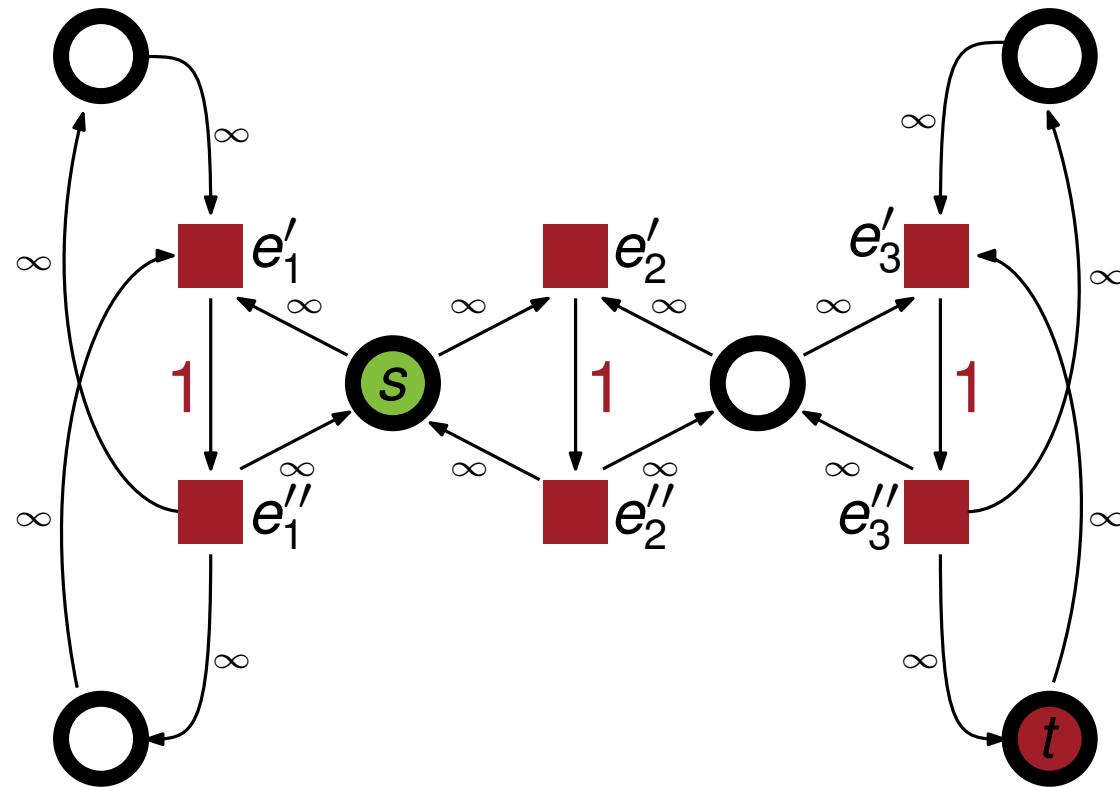
Removing Source and Sink Vertices

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

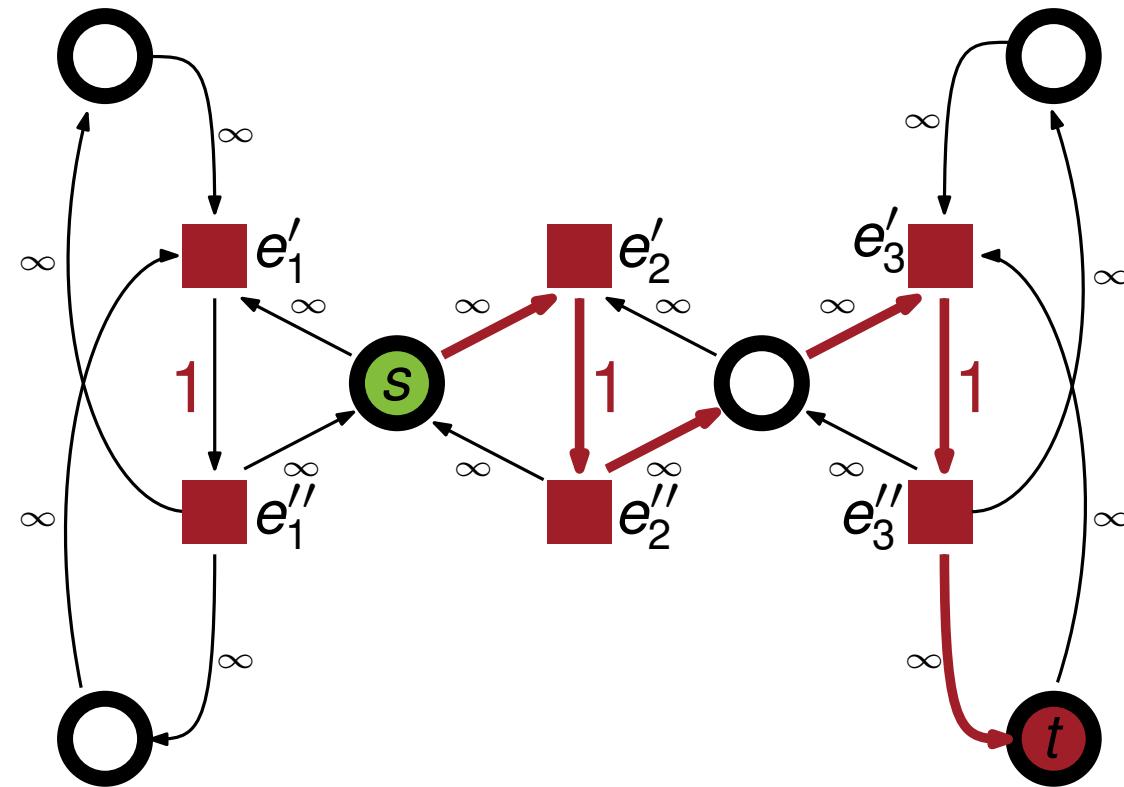


Lawler Network

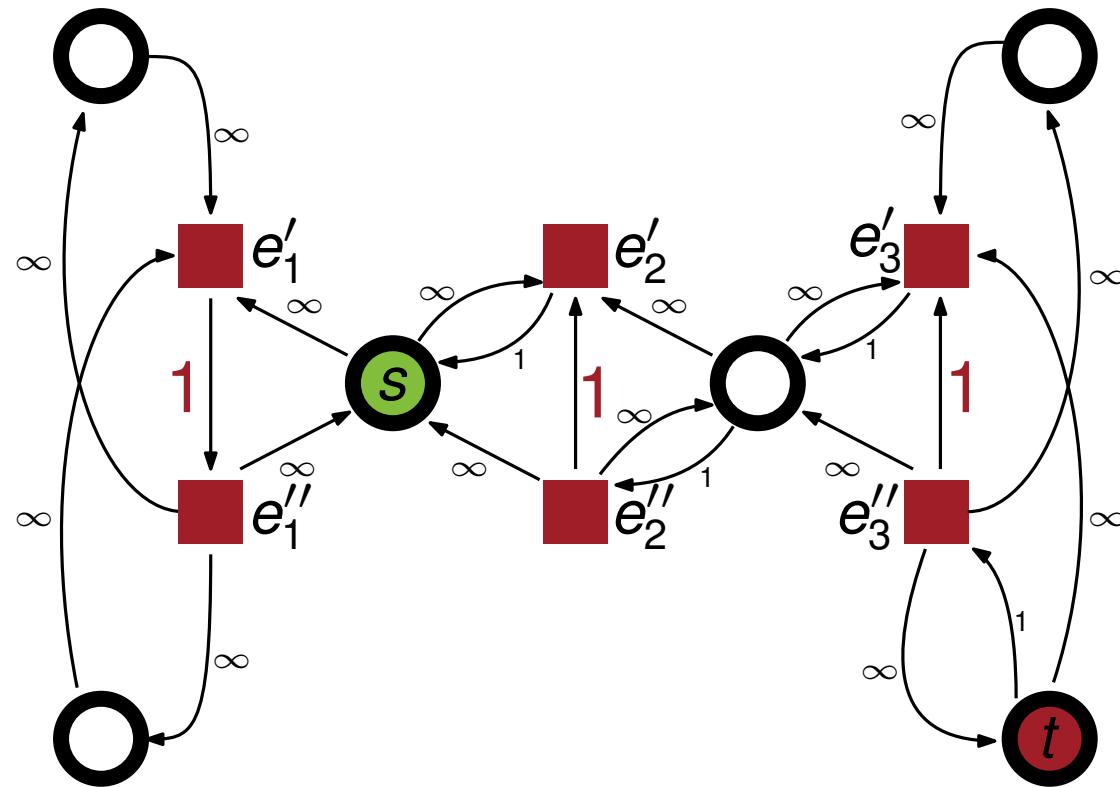
Minimum (s, t) -Bipartition



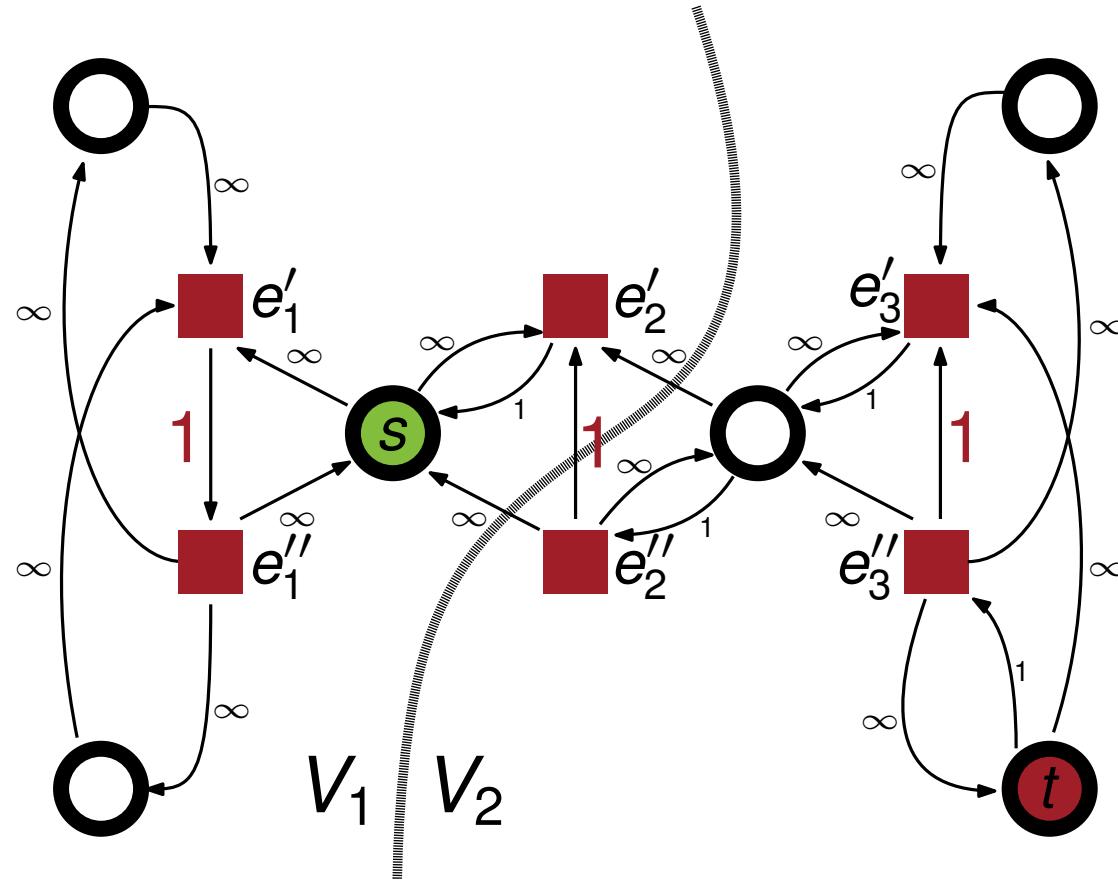
Minimum (s, t) -Bipartition



Minimum (s, t) -Bipartition



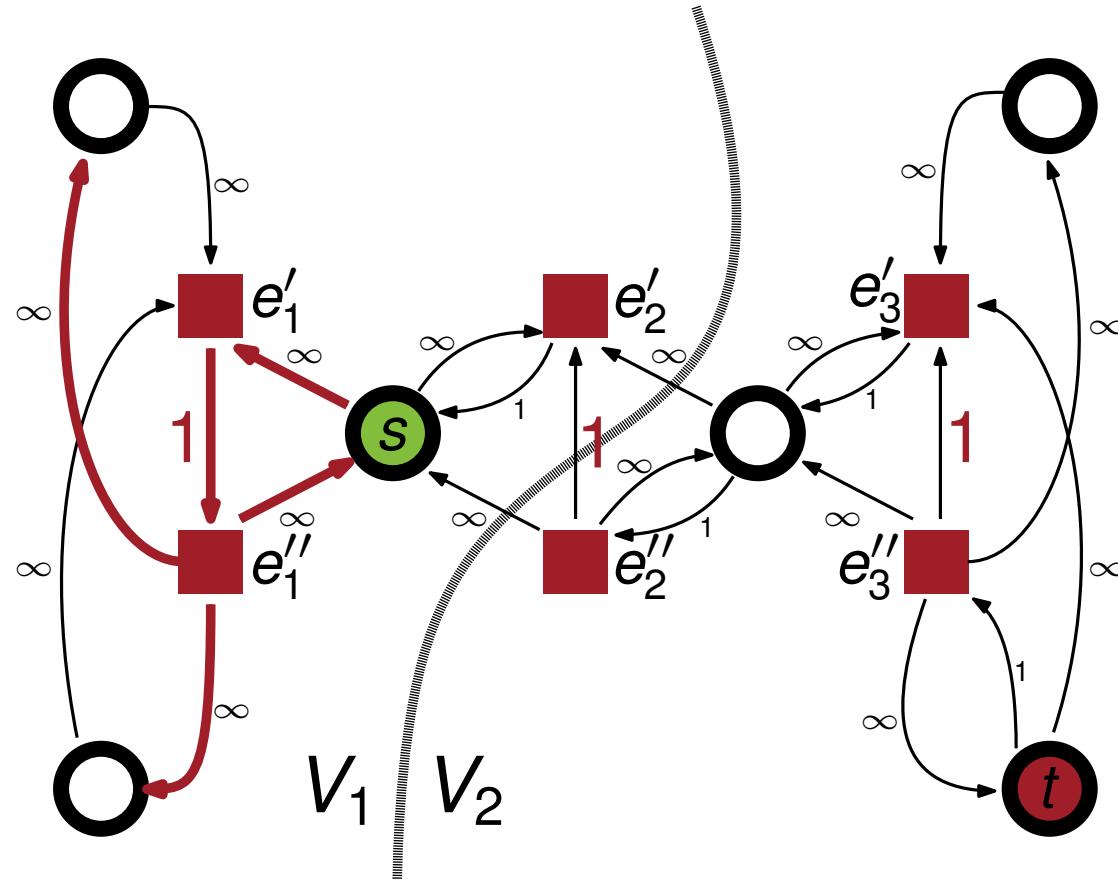
Minimum (s, t) -Bipartition



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

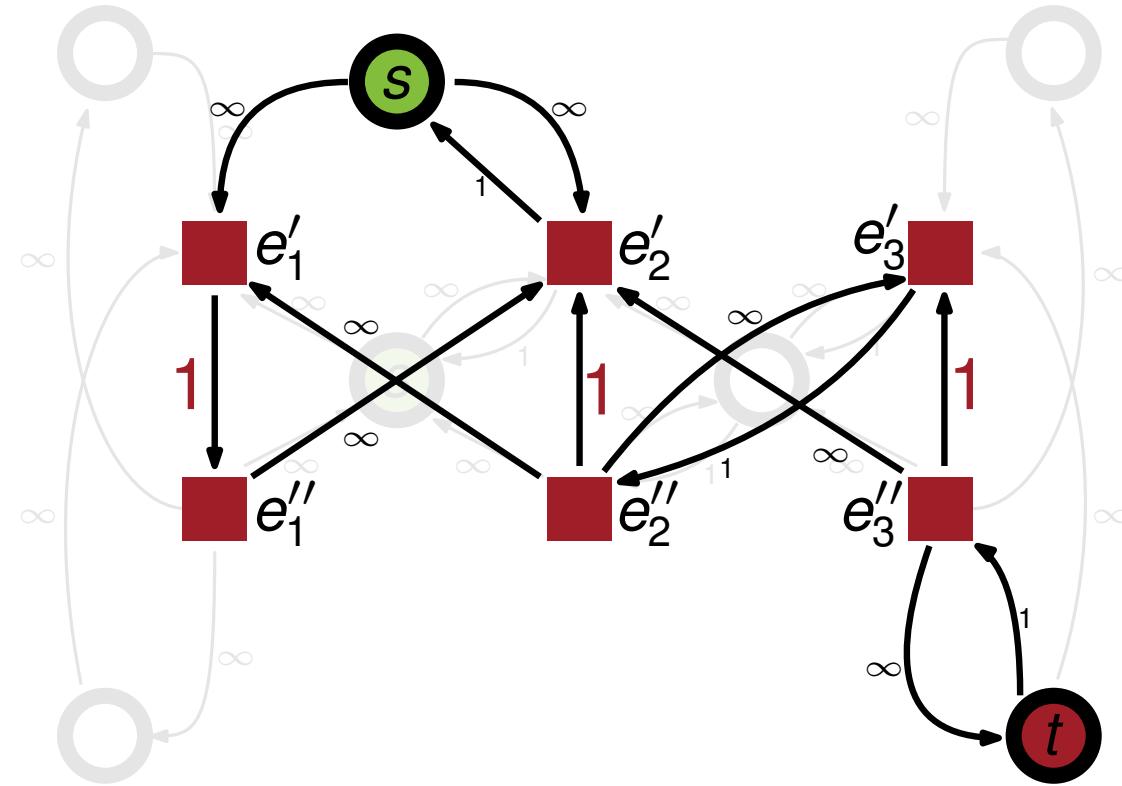
Minimum (s, t) -Bipartition

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

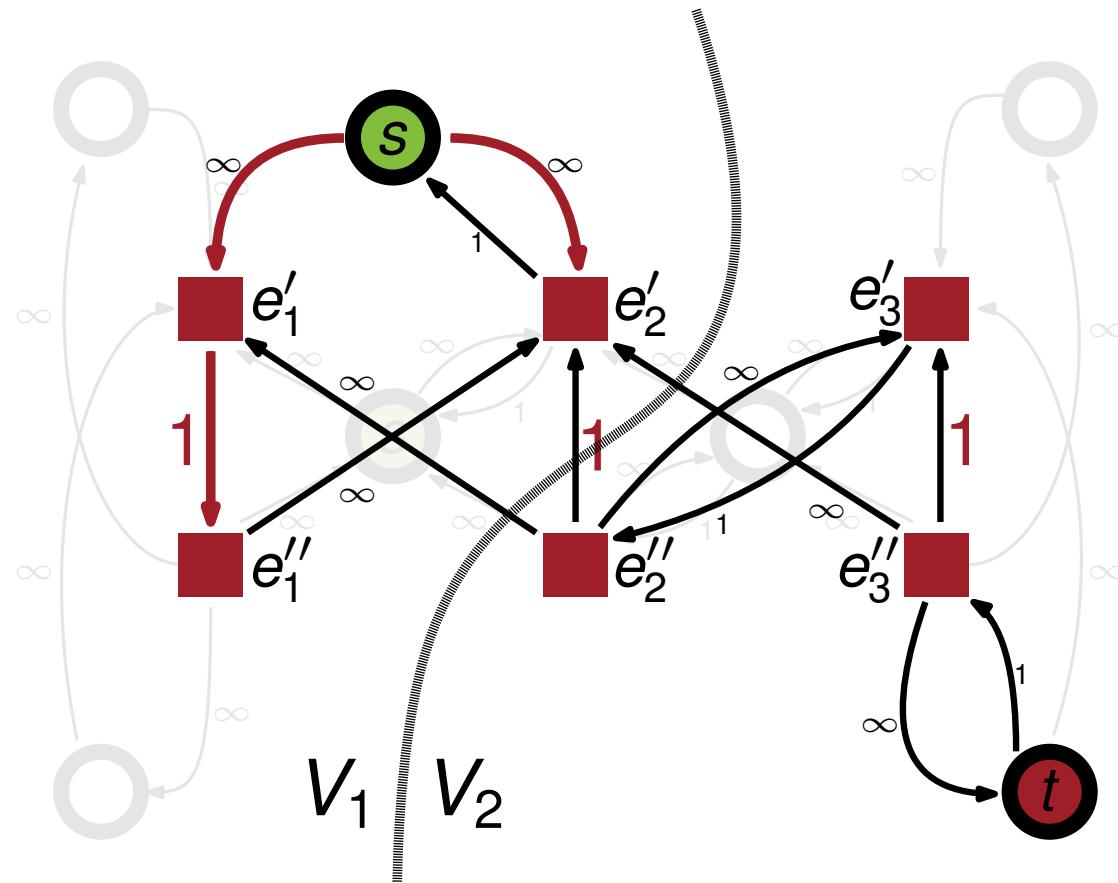


All nodes *reachable* from s are part of V_1 and $V_2 = V \setminus V_1$

Minimum (s, t) -Bipartition



Minimum (s, t) -Bipartition



Minimum (s, t) -Bipartition

