Using Linear Algebra in Algorithms
Example 2: Maximum flow

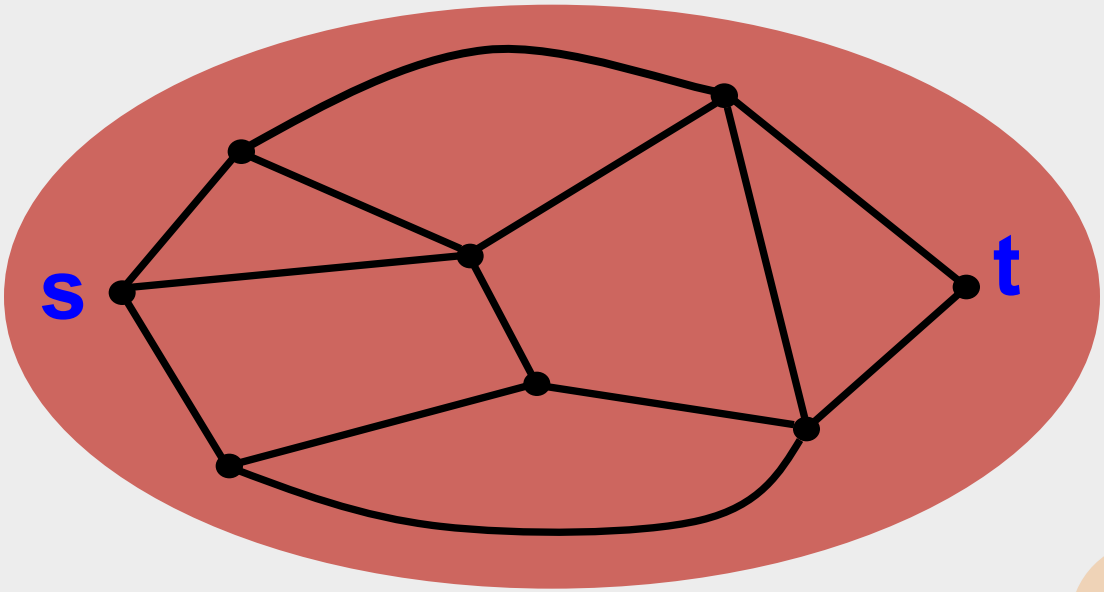
Abbas Mehrabian

a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

Acknowledgement: some of the following slides were prepared by Aleksander Madry and are downloaded from http://www.iasi.cnr.it/~ventura/Cargese15/Lectures.html

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**

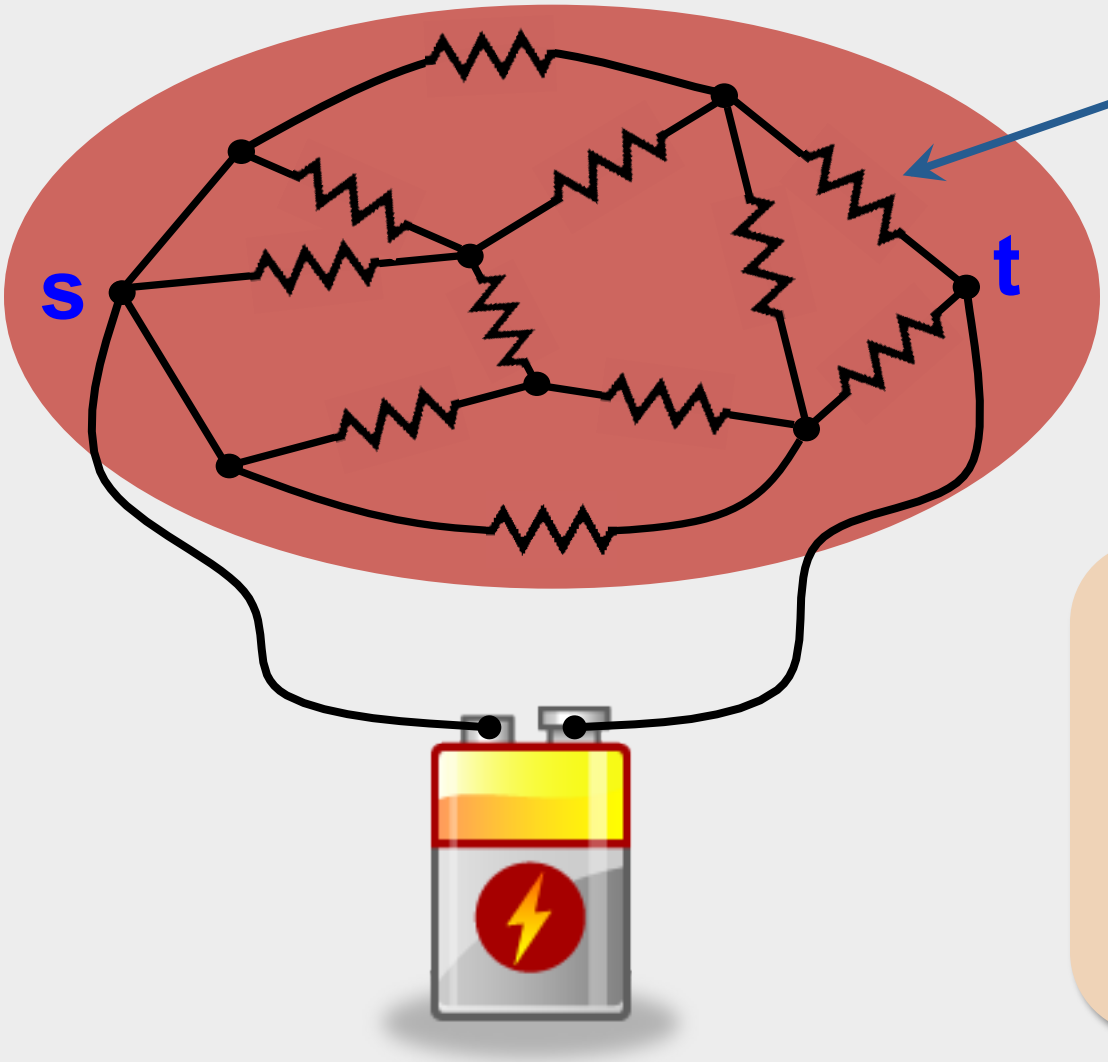


Recipe for elec. flow:

1) Treat edges as **resistors**

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**



resistance r_e

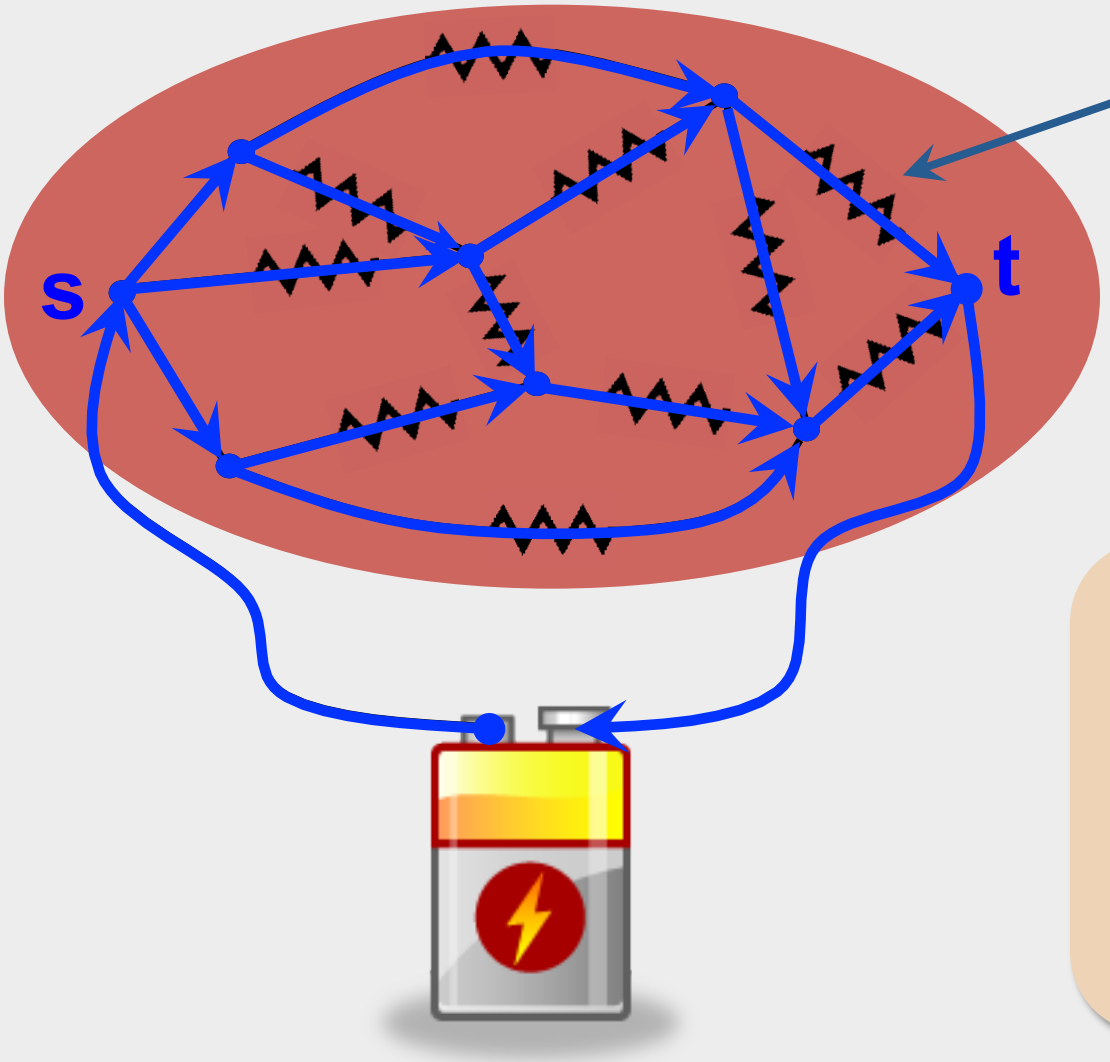
Recipe for elec. flow:

1) Treat edges as **resistors**

2) Connect a **battery** to **s** and **t**

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**



resistance r_e

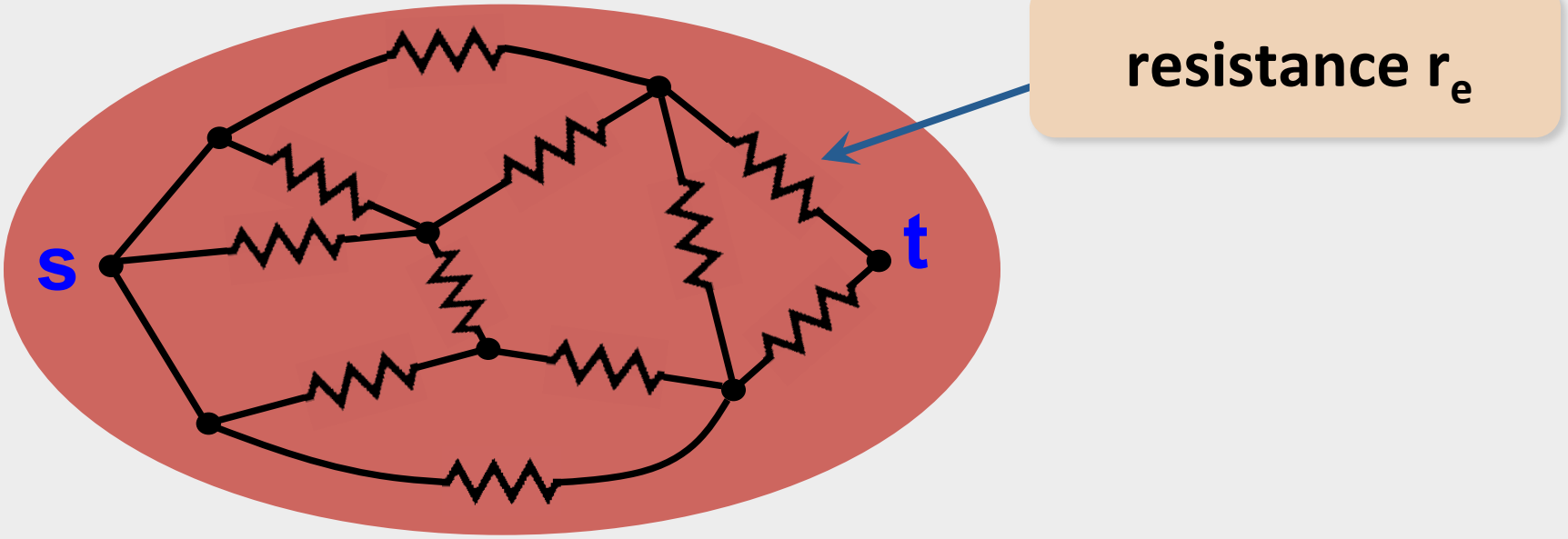
Recipe for elec. flow:

1) Treat edges as **resistors**

2) Connect a **battery** to **s** and **t**

Electrical flows (Take II)

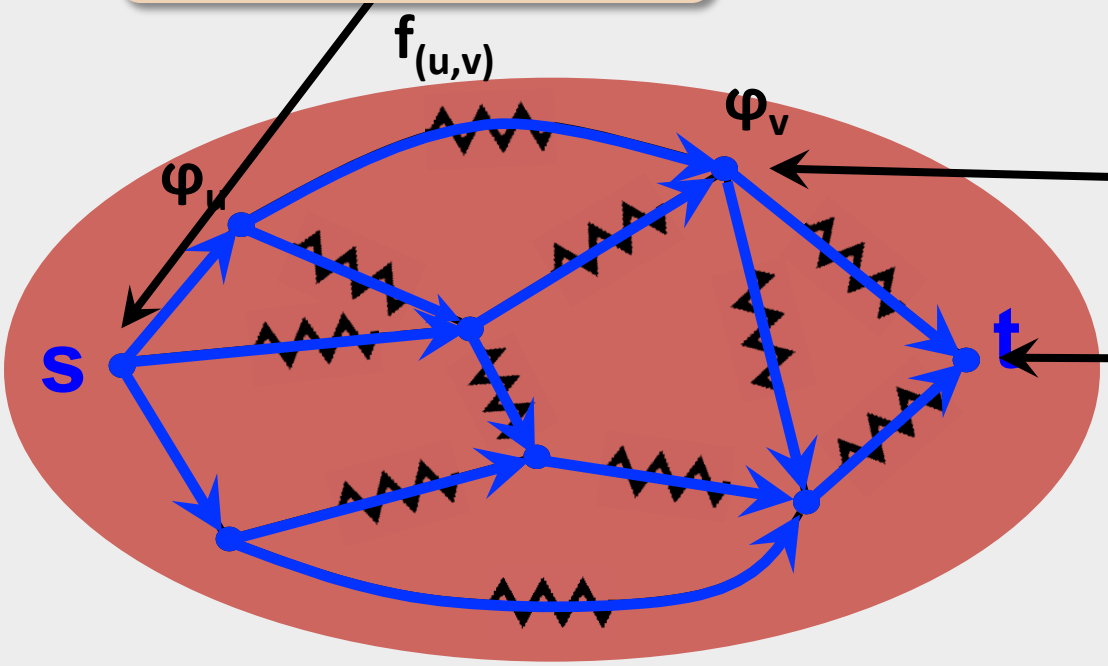
Input: **Undirected** graph **G**,
resistances r_e,
source s and **sink t**



(Another) recipe for electrical flow (of value F):

Ele…ake II)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**

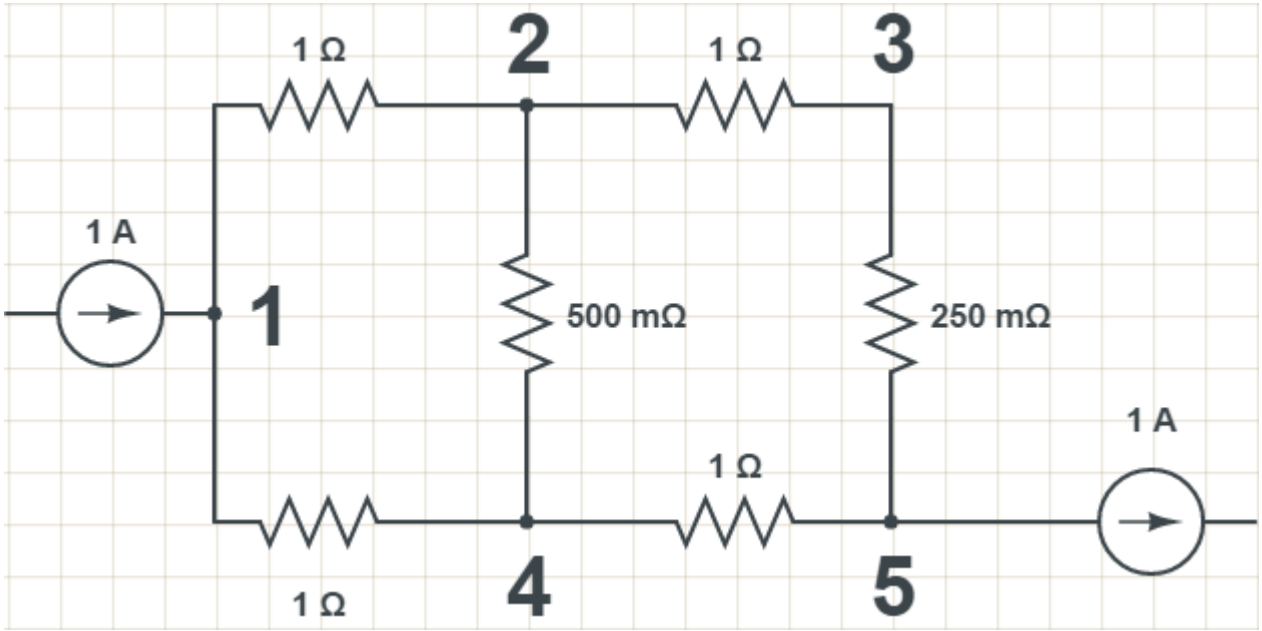


(Another) recipe for electrical flow (of value F):

Find **vertex potentials φ_v** such that setting, for all **(u,v)**

$$f_{(u,v)} \leftarrow (\varphi_v - \varphi_u)/r_{(u,v)}$$ **(Ohm's law)**

gives a **valid s-t flow** of **value F**

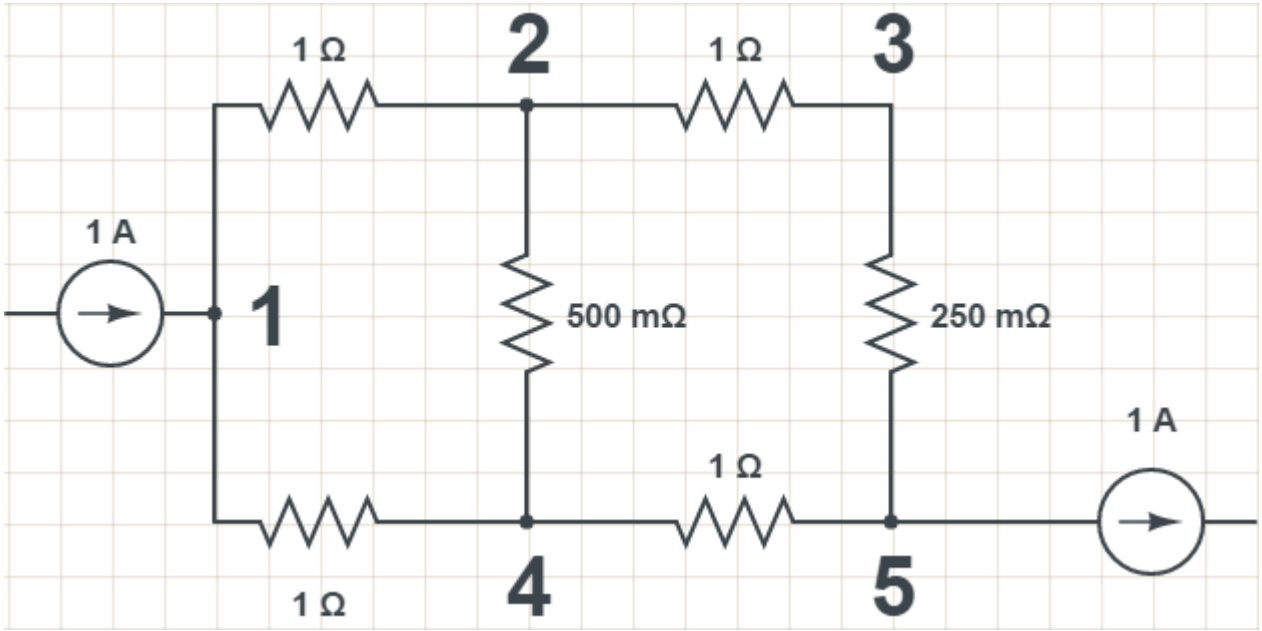


$$outgoing - current(1) = \frac{\varphi_1 - \varphi_2}{1} + \frac{\varphi_1 - \varphi_4}{1} = -1$$

$$2\varphi_1 - \varphi_2 - \varphi_4 = -1$$

$$outgoing - current(2) = \frac{\varphi_2 - \varphi_1}{1} + \frac{\varphi_2 - \varphi_3}{1} + \frac{\varphi_2 - \varphi_4}{1/2} = 0$$

$$4\varphi_2 - \varphi_1 - \varphi_3 - 2\varphi_4 = 0$$



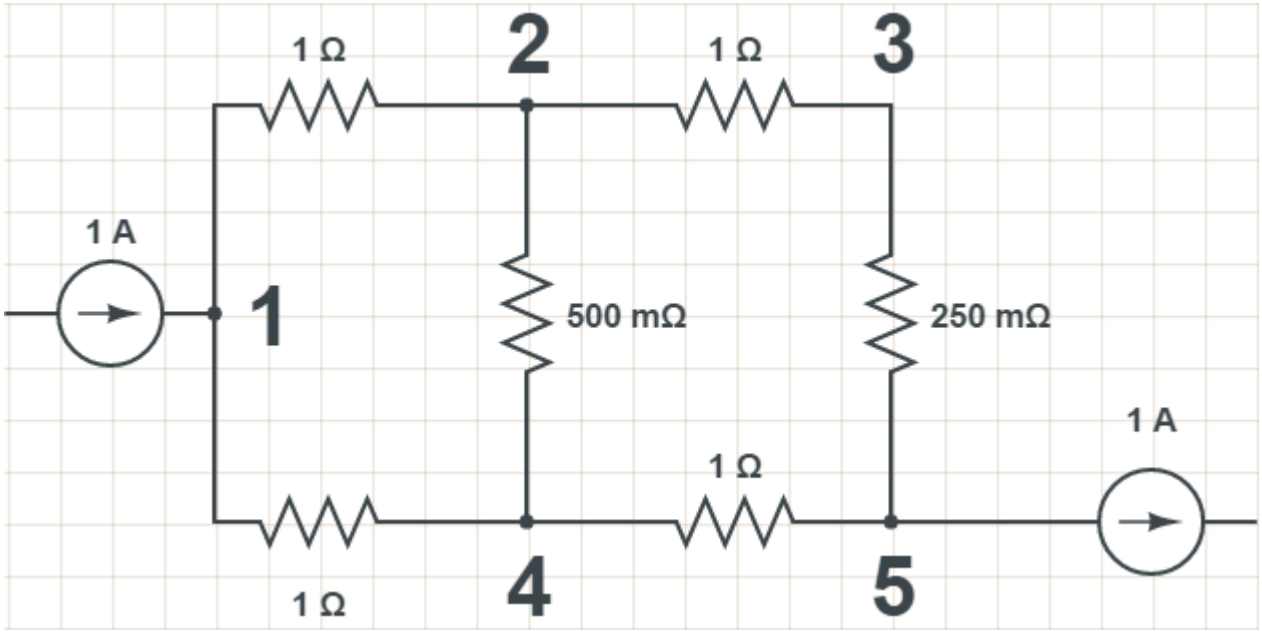
$$2\varphi_1-\varphi_2-\varphi_4=-1$$

$$4\varphi_2-\varphi_1-\varphi_3-2\varphi_4=0$$

$$4\varphi_3-\varphi_2-3\varphi_5=0$$

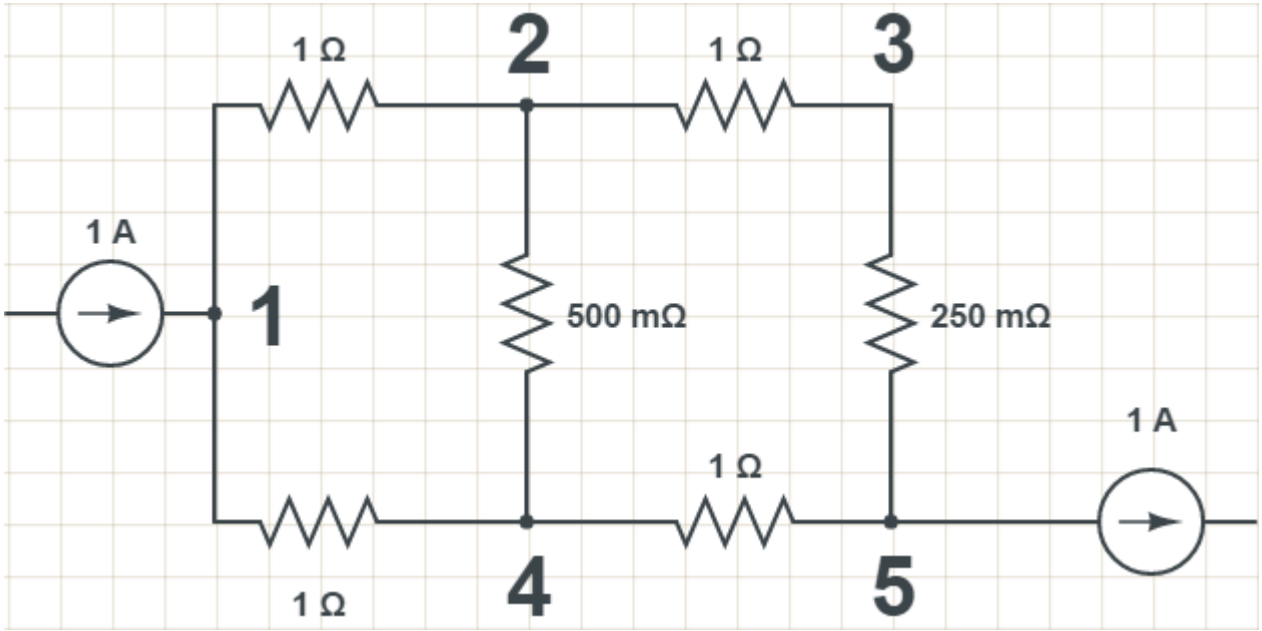
$$4\varphi_4-\varphi_1-\varphi_5-2\varphi_2=0$$

$$4\varphi_5-\varphi_4-3\varphi_3=1$$



$$\begin{bmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 4 & -1 & -2 & 0 \\ 0 & -1 & 4 & 0 & -3 \\ -1 & -2 & 0 & 4 & -1 \\ 0 & 0 & -3 & -1 & 4 \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$L \times \phi = b$$



$$\begin{bmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 4 & -1 & -2 & 0 \\ 0 & -1 & 4 & 0 & -3 \\ -1 & -2 & 0 & 4 & -1 \\ 0 & 0 & -3 & -1 & 4 \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \\ \varphi_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$L \times \phi = b$$

symmetric, positive semi-definite,
row-sum-zero, n+2m non-zero entries

How fast can we solve L ϕ = b ?

- *Ax=b* can be solved in $O(n^{2.38})$ (CLRS chapter 31)
  http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap31.htm

How fast can we solve L ϕ = b ?

- *Ax=b* can be solved in $O(n^{2.38})$ (CLRS chapter 31)
  http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap31.htm
- *A* Laplacian matrix of a graph:
  - An ε-approx. in time $O(m \log^{100} m \log(1/\varepsilon))$ [Spielman-Teng'04—11]
  - $O(m \log^{1/2} m \log(1/\varepsilon))$ [Cohen, Kyng, Miller, Pachocki, Peng, Rao, Xu'14]

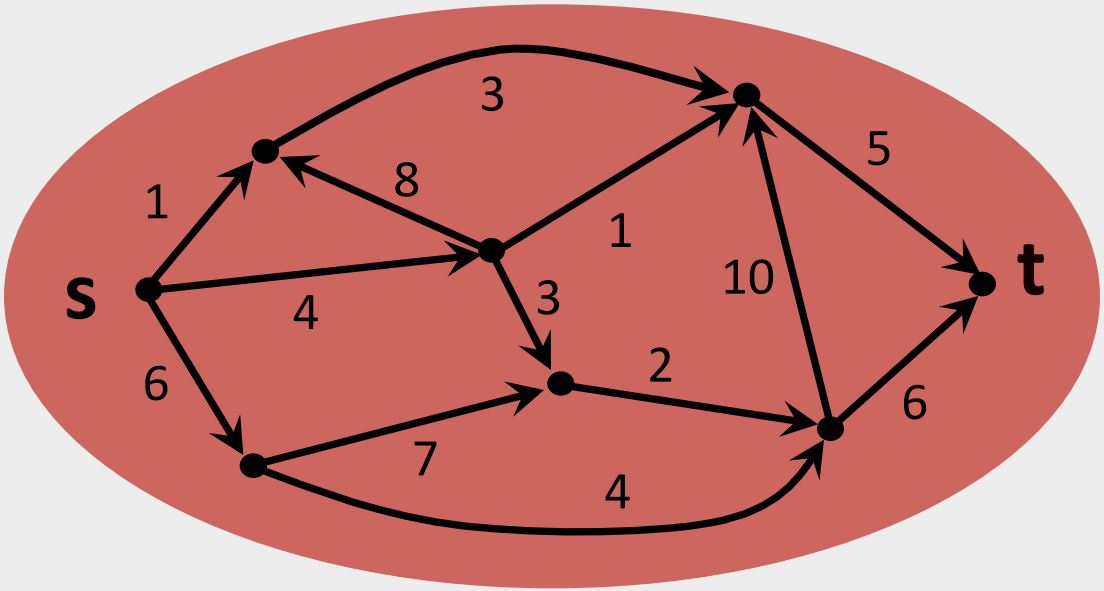
How fast can we solve L ϕ = b ?

- *Ax=b* can be solved in $O(n^{2.38})$ (CLRS chapter 31)
  http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap31.htm
- *A* Laplacian matrix of a graph:
  - An ε-approx. in time $O(m \log^{100} m \log(1/\varepsilon))$
    [Spielman-Teng'04—11]
  - $O(m \log^{1/2} m \log(1/\varepsilon))$
    [Cohen, Kyng, Miller, Pachocki, Peng, Rao, Xu'14]
- Numerous applications in graph algorithms, optimization, machine learning, numerical linear algebra
- Kelner's 3 talks in Berkeley (2013):
  https://simons.berkeley.edu/talks/laplacian-systems-and-electrical-flows
- Vishnoi's 2012 book "Laplacian solvers and their algorithmic applications" http://theory.epfl.ch/vishnoi/Lxb-Web.pdf

THE MAXIMUM FLOW PROBLEM

Maximum flow problem

Input: Directed graph **G**, integer **capacities u_e**, **source s** and **sink t**



Think: arcs = roads
capacities = # of lanes
s/t = origin/destination

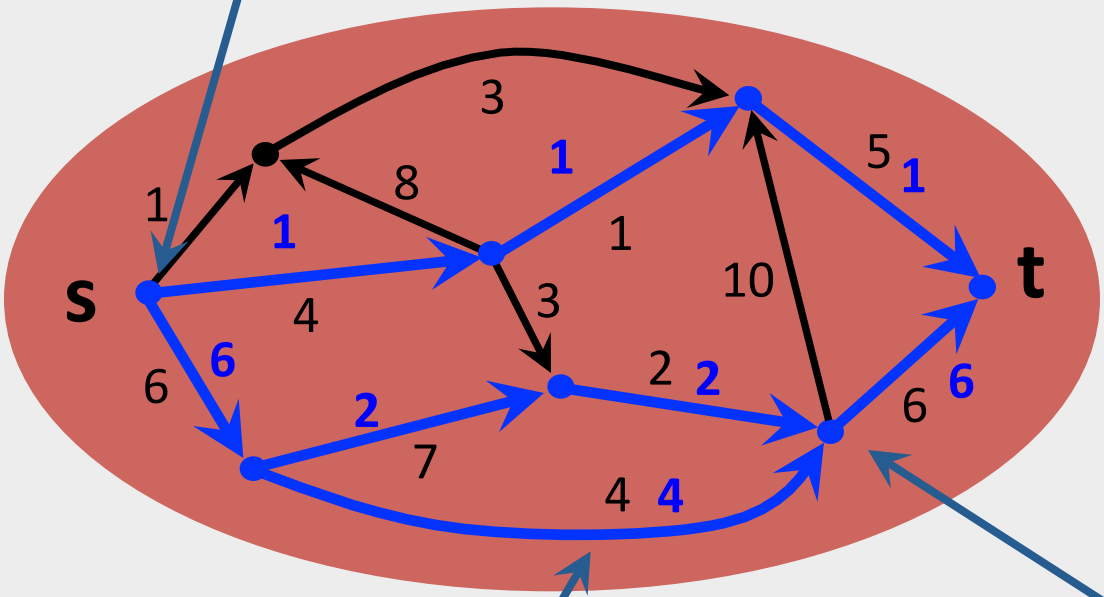
Task: Find a **feasible s-t flow** of **max value**

(**Think:** Estimate the **max** possible rate of traffic from **s** to **t**)

Maximum flow problem

Input: Directed graph **G**, integer **capacities u_e**, **source s** and **sink t**

value = net flow out of **s**



Think: arcs = roads
capacities = # of lanes
s/t = origin/destination

Max flow value
F*=10

no overflow on arcs:
$0 \le f(e) \le u(e)$

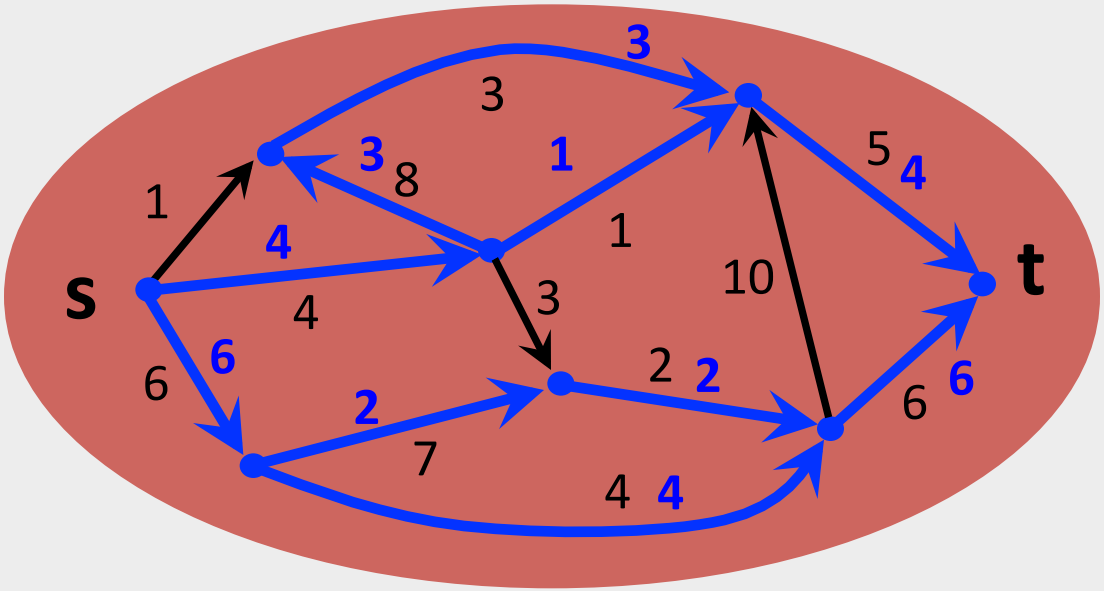
no leaks at all **v≠s,t**

Task: Find a **feasible s-t flow** of **max value**

(**Think:** Estimate the **max** possible rate of traffic from **s** to **t**)

Maximum flow problem

Input: Directed graph **G**, integer **capacities u_e**, **source s** and **sink t**



Think: arcs = roads
capacities = # of lanes
s/t = origin/destination

Max flow value
F*=10

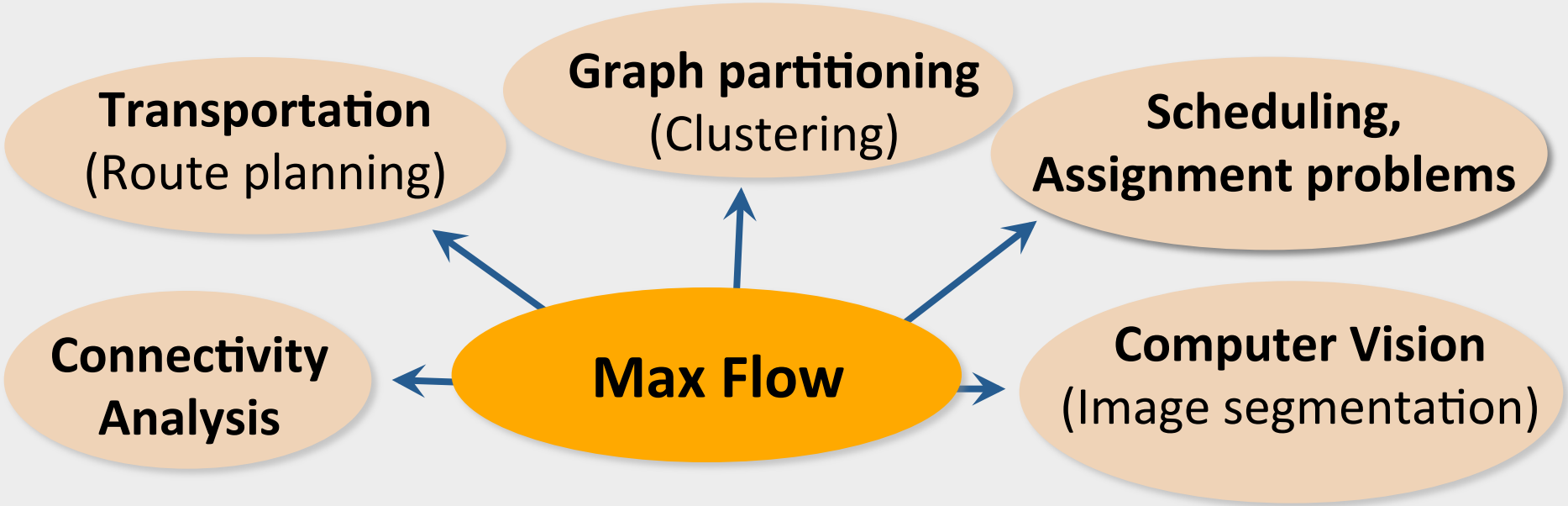
Task: Find a **feasible s-t flow** of **max value**

(**Think:** Estimate the **max** possible rate of traffic from **s** to **t**)

Why is this a good problem to study?

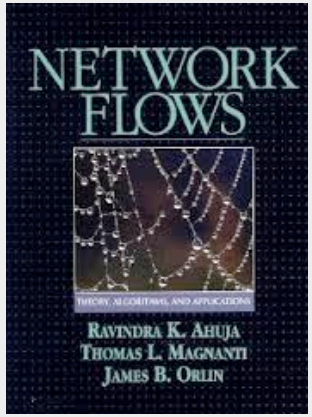
Max flow is a fundamental optimization problem

- **Extensively studied since 1930s** (classic 'textbook problem')
- **Surprisingly diverse set of applications**
- **Very influential in development of (graph) algorithms**



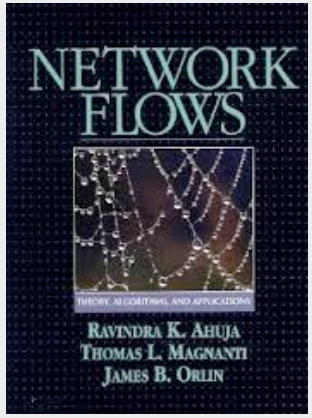
What is known about Max Flow?

A **LOT** of previous work



What is known about Max Flow?

A (very) rough history outline



[Dantzig '51]	$O(mn^2 U)$
[Ford Fulkerson '56]	$O(mn\ U)$
[Dinitz '70]	$O(mn^2)$
[Dinitz '70] [Edmonds Karp '72]	$O(m^2n)$
[Dinitz '73] [Edmonds Karp '72]	$O(m^2 \log U)$
[Dinitz '73] [Gabow '85]	$O(mn \log U)$
[Goldberg Rao '98]	$\tilde{O}(m \min(m^{1/2}, n^{2/3}) \log U)$
[Lee Sidford '14]	$\tilde{O}(mn^{1/2} \log U)$

Our focus: Sparse graph (**m=O(n)**) and unit-capacity (**U=1**) regime

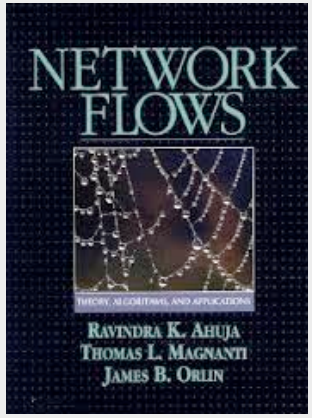
→ It is a good benchmark for combinatorial graph algorithms

→ Already captures interesting problems, e.g., **bipartite matching**

(**n** = # of vertices, **m** = # of arcs, **U** = max capacity, **$\tilde{O}$()** hides polylogs)

What is known about Max Flow?

A (very) rough history outline



[Dantzig '51]	$O(n^3)$
[Ford Fulkerson '56]	$O(n^2)$
[Dinitz '70]	$O(n^3)$
[Dinitz '70] [Edmonds Karp '72]	$O(n^3)$
[Dinitz '73] [Edmonds Karp '72]	$\tilde{O}(n^2)$
[Dinitz '73] [Gabow '85]	$\tilde{O}(n^2)$
[Goldberg Rao '98]	$\tilde{O}(n^{3/2})$
[Lee Sidford '14]	$\tilde{O}(n^{3/2})$

Our focus: Sparse graph (**m=O(n)**) and unit-capacity (**U=1**) regime

→ It is a good benchmark for combinatorial graph algorithms

→ Already captures interesting problems, e.g., **bipartite matching**

(**n** = # of vertices, **m** = # of arcs, **U** = max capacity, **$\tilde{O}$()** hides polylogs)

What is known about Max Flow?

Emerging barrier: $O(n^{3/2})$

[Even Tarjan '75, Karzanov '73]: Achieved this bound for **U=1** long time ago

Last 40 years: Matching this bound in increasingly more general settings, but **no improvement**

This indicates a fundamental limitation of our techniques

Our goal: Show a new approach finally breaking this barrier

(**n** = # of vertices, **m** = # of arcs, **U** = max capacity, **$\tilde{O}$()** hides polylogs)

Breaking the $\Omega(n^{3/2})$ barrier

Undirected graphs and approx. answers ($\Omega(n^{3/2})$ barrier still holds here)

[M '10]: **Crude approx. of** max flow **value** in **close to linear** time

[CKMST '11]: **(1-ε)-approx.** to max flow in $\tilde{O}(n^{4/3}\varepsilon^{-3})$ time

[LS '13, S '13, KLOS '14, P '14]: **(1-ε)-approx.** in $\tilde{O}(n\varepsilon^{-2})$ time

But: What about the **directed** and **exact** setting?

Today

[M '13]: Exact $\tilde{O}(n^{10/7})=\tilde{O}(n^{1.43})$-time alg.

(n = # of vertices, $\tilde{O}()$ hides polylog factors)

New approach:
Bring linear-algebraic techniques into play

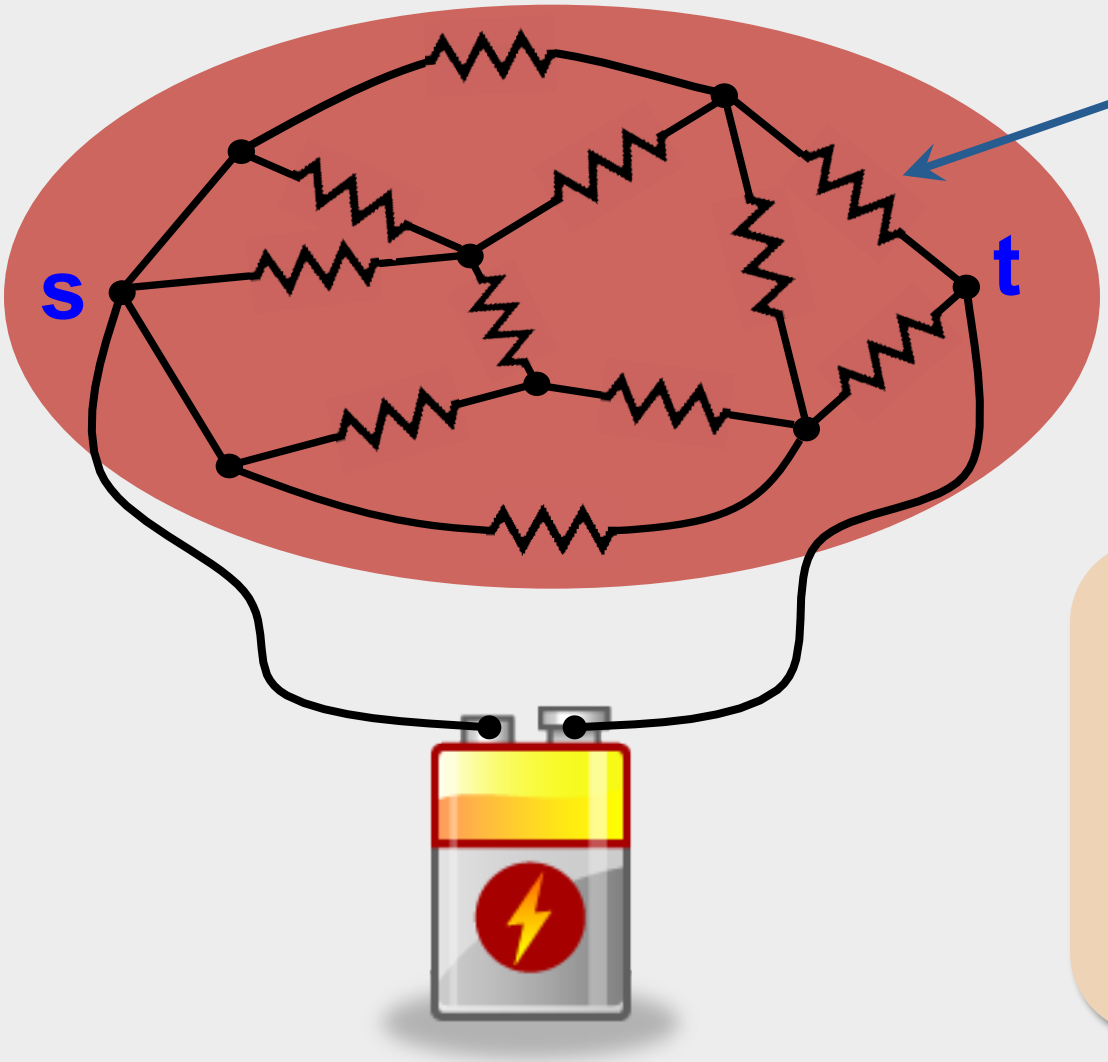
Idea: Probe the **global flow structure** of the graph by **solving linear systems**

How to relate **flow structure** to **linear algebra**?
(And why should it even help?)

Key object: Electrical flows

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**



resistance r_e

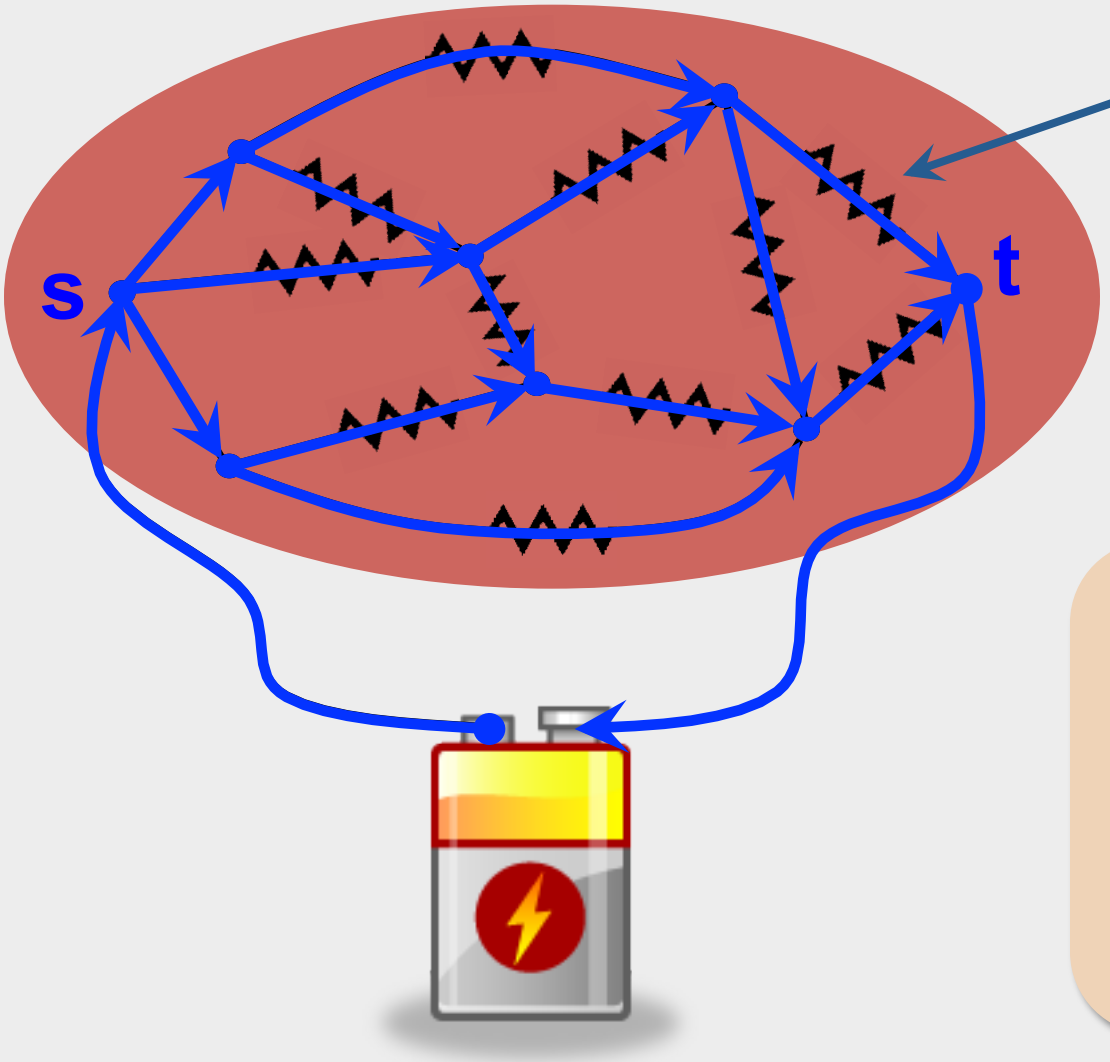
Recipe for elec. flow:

1) Treat edges as **resistors**

2) Connect a **battery** to **s** and **t**

Electrical flows (Take I)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**



resistance r_e

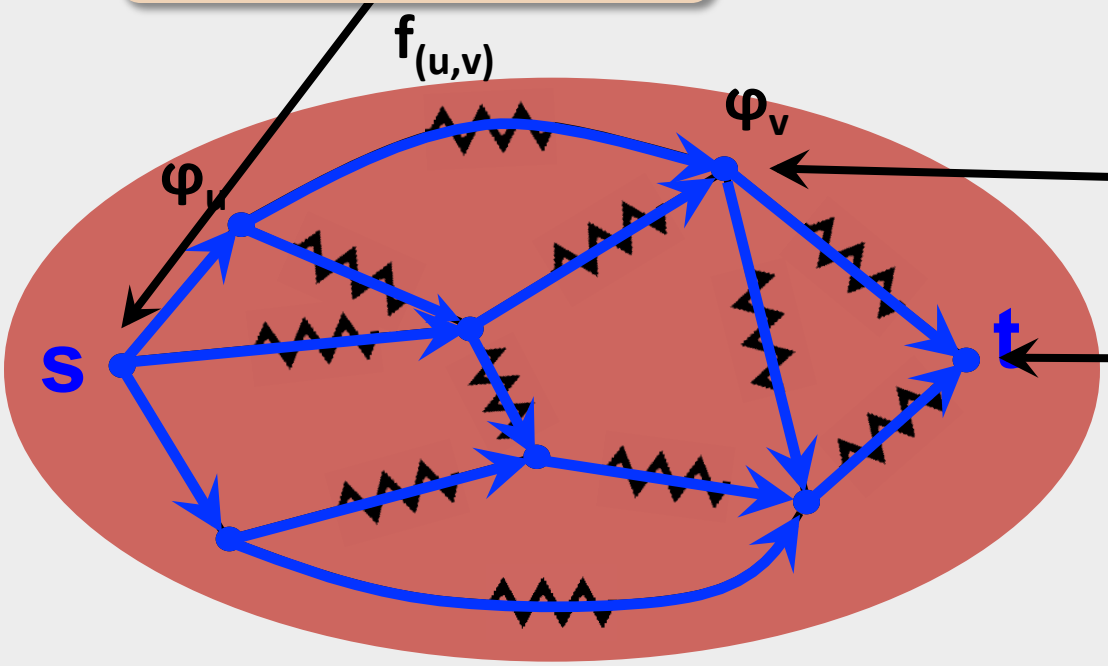
Recipe for elec. flow:

1) Treat edges as **resistors**

2) Connect a **battery** to **s** and **t**

Ele…ake II)

Input: **Undirected** graph **G**, **resistances r_e**, **source s** and **sink t**



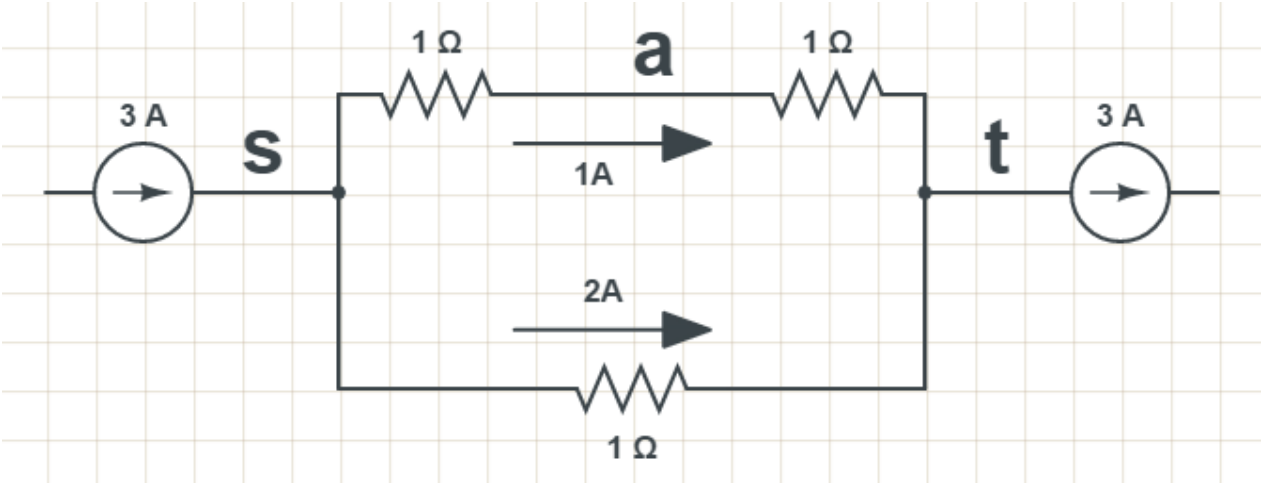
(Another) recipe for electrical flow (of value F):

Find **vertex potentials φ_v** such that setting, for all **(u,v)**

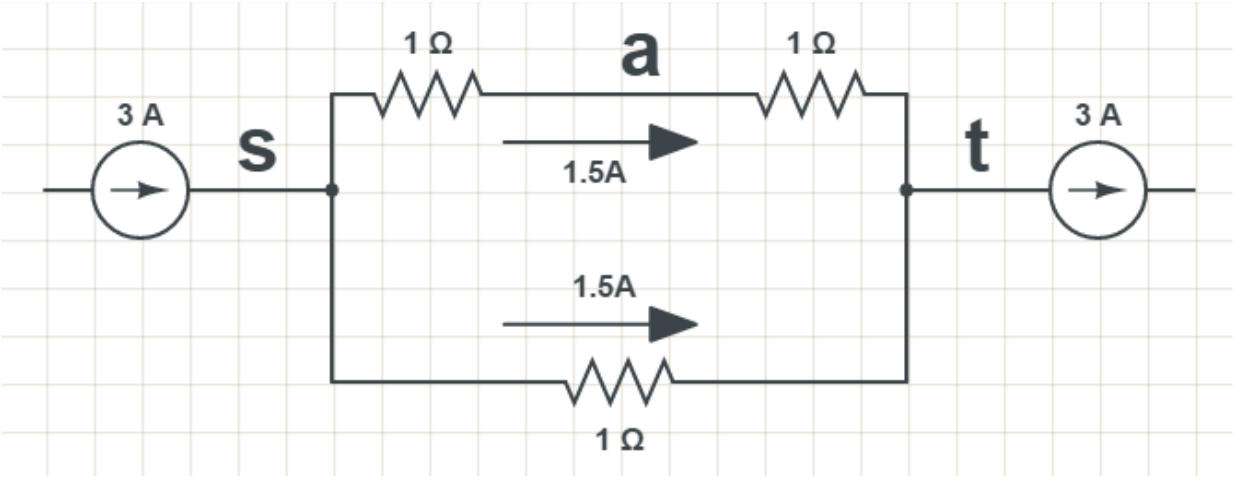
$$f_{(u,v)} \leftarrow (\varphi_v - \varphi_u)/r_{(u,v)}$$ **(Ohm's law)**

gives a **valid s-t flow** of **value F**

- Electrical 3-flow



- Arbitrary 3-flow



Electrical flows

Input: **Undirected** graph **G**,
resistances r_e,
source s and **sink t**

Principle of least energy

Electrical flow of value F:
The unique minimizer of the **energy**

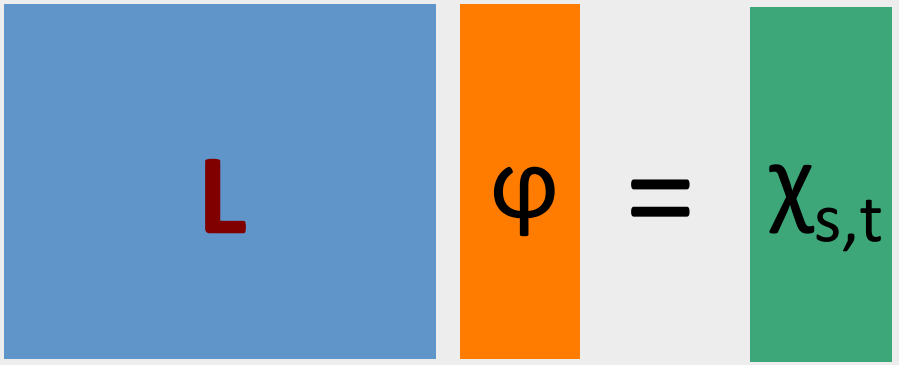
$$\mathbf{E(f) = \Sigma_e\, r_e\, f(e)^2}$$

among all **s-t** flows **f** of value **F**

Electrical flows = ℓ_2-minimization

How to compute an electrical flow?

Bottom line:



Electrical flow computation

Solving a **Laplacian system**

Bad news: Solving a linear system can take $\mathbf{O(n^{\omega})=O(n^{2.373})}$ (Prohibitive!)

Key observation:
$\mathbf{BR^{-1}B^{T}}$ is the **Laplacian** matrix **L** of the underlying graph

How to utilize it?

Result: Electrical flow is a **nearly-linear time** primitive

From electrical flows to undirected max flow

[CKMST '11]

- Suppose we have an algorithm, given F*, either finds a feasible flow of value F*, or decides that it does not exist

- Suppose we have an algorithm, given F*, either finds a feasible flow of value F*, or decides that it does not exist
- If this alg. has runtime T(n), we get an alg. with runtime T(n) log n for finding max flow

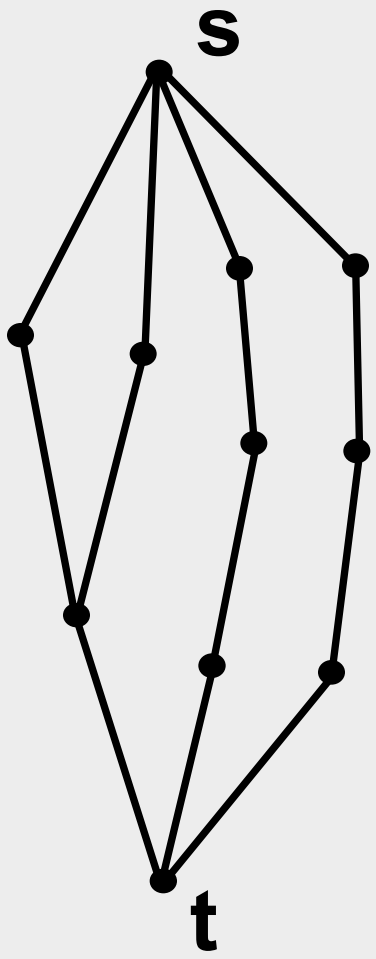
  Let LB = 0 and UB = n^2

  While UB – LB > ε
  - Let F* = (UB+LB)/2
  - run the algorithm for F*
  - If successful, LB = F* else UB = F*

Approx. undirected max flow via electrical flows

Assume: F^* known (via binary search)

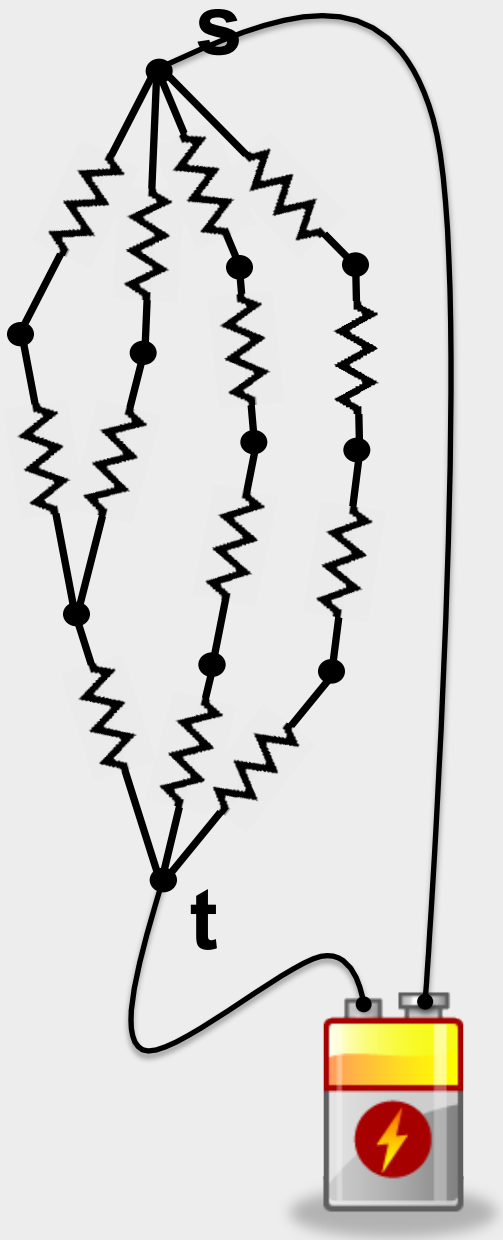
→ Treat edges as resistors of resistance **1**



Approx. undirected max flow via electrical flows

Assume: F^* known (via binary search)

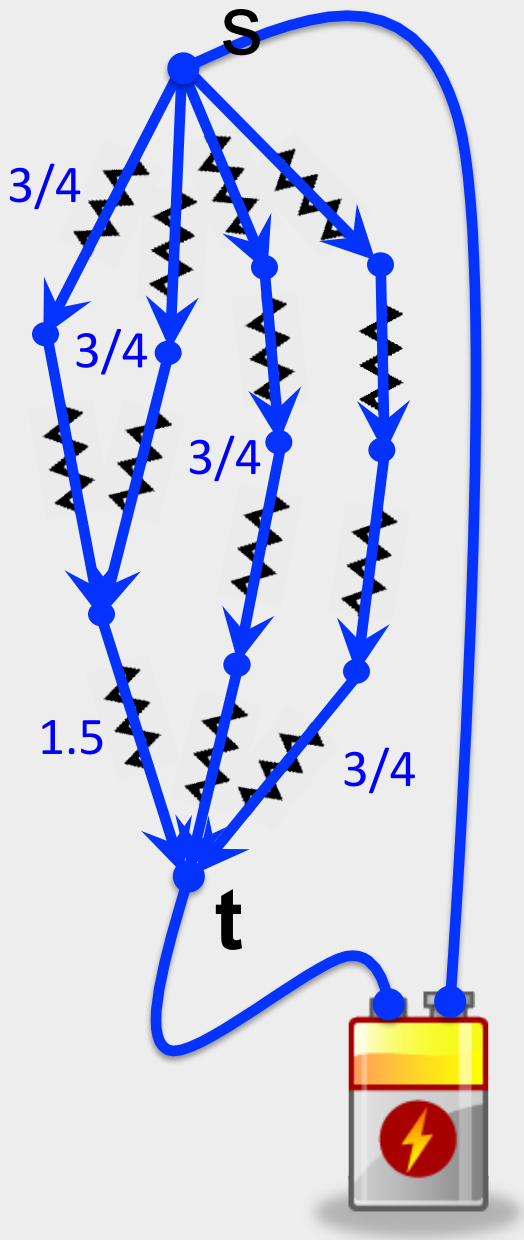
→ Treat edges as resistors of resistance **1**
→ Compute electrical flow of value **F^***



Approx. undirected max flow via electrical flows

Assume: F^* known (via binary search)

→ Treat edges as resistors of resistance **1**
→ Compute electrical flow of value **F^***
(This flow has **no leaks**, but **can overflow** some edges)



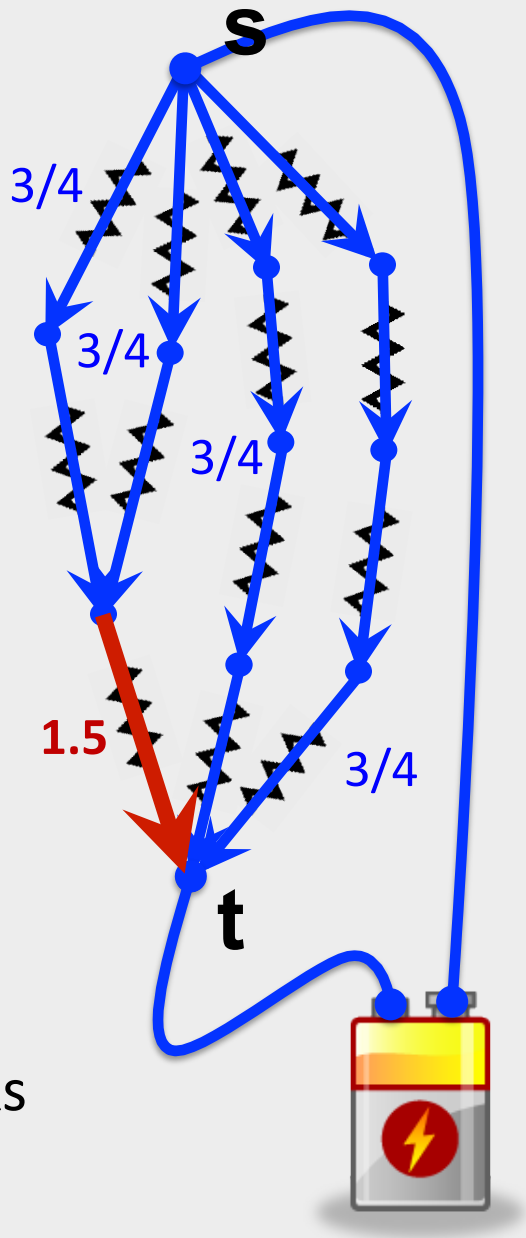
Approx. undirected max flow via electrical flows

Assume: F^* known (via binary search)

→ Treat edges as resistors of resistance **1**

→ Compute electrical flow of value **F^***
(This flow has **no leaks**, but **can overflow** some edges)

→ To fix that: **Increase resistances** on the overflowing edges
Repeat (**hope:** it doesn't happen too often)



Surprisingly: This approach can be made work!

But: One needs to be careful how to fill in the blanks

We will do this now

Filling in the blanks

Recall: We are dealing with **undirected** graphs

From now on: All capacities are **1**, **m=O(n)**
and the value $\mathbf{F}^*$ of max flow is known

Electrical vs. maximum flows

Fix some resistances **r** and consider the elect. flow $\mathbf{f_E}$ of value **F***

We don't expect $\mathbf{f_E}$ to obey **all** capacity constraints
(i.e., we can have $|\mathbf{f_E(e)}| >> \mathbf{1}$ for some edge **e**)

Still, $\mathbf{f_E}$ obeys these constraints in a certain sense...

We have:

$$\Sigma_e\, r_e\, |f_E(e)| \leq \Sigma_e\, r_e$$

In other words: Capacity constraints are preserved on **average** (weighted wrt to $\mathbf{r_e}$s)

Proof:

Electrical vs. maximum flows

This gives rise to a **very fast** algorithm for the following task:

'Feasibility on average':

Given weights **w** compute a flow **f** of value **F*** s.t.

$$\Sigma_e w_e |f(e)| \leq \Sigma_e w_e$$

Key point: We already know how to make such a crude algorithm useful to us!

Multiplicative weights update method
[FS '97, PST '95, AHK '05]

'Technique for turning weak algorithms into strong ones'

In our setting:

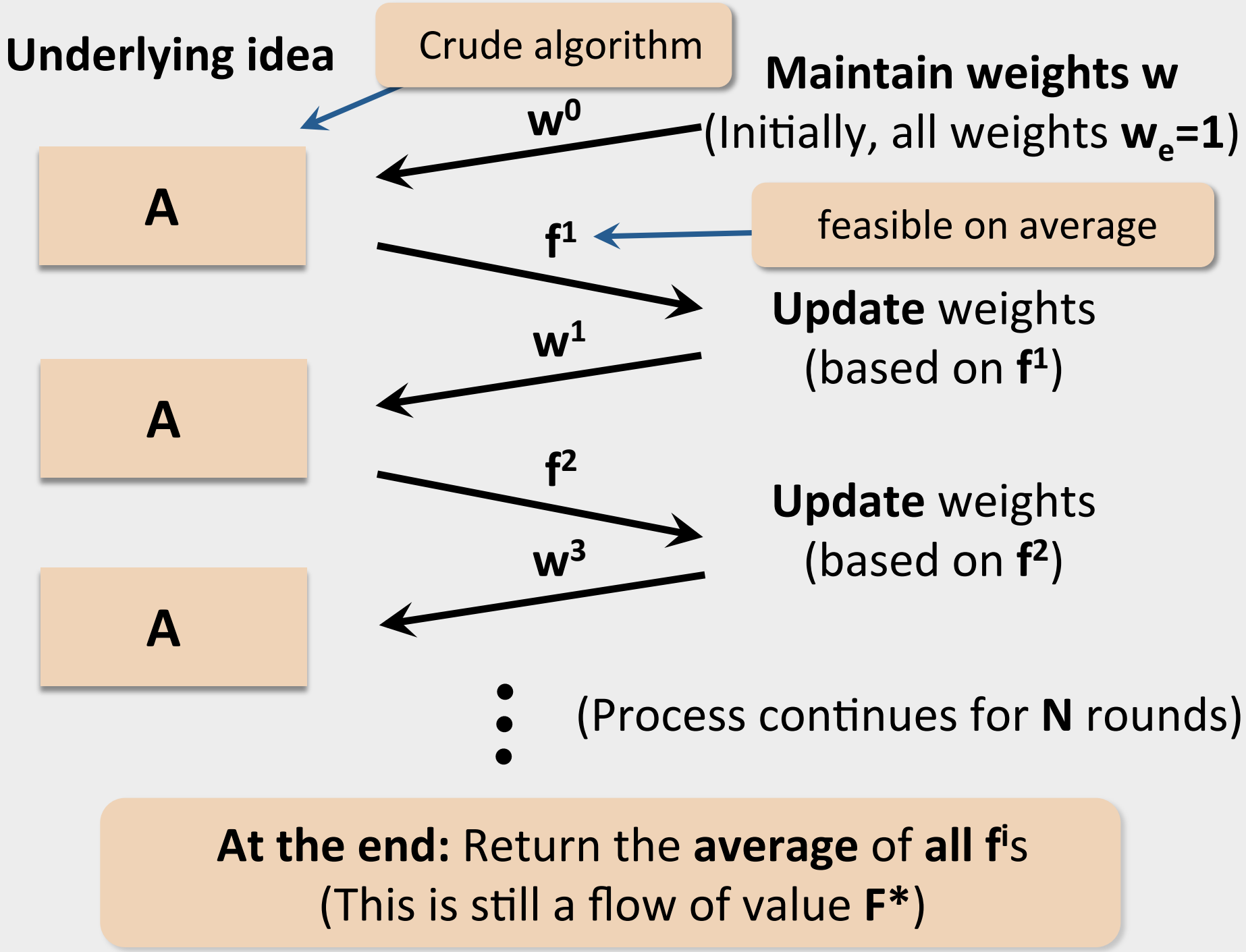
Crude algorithm computing **'feasible on average'** flows

↓

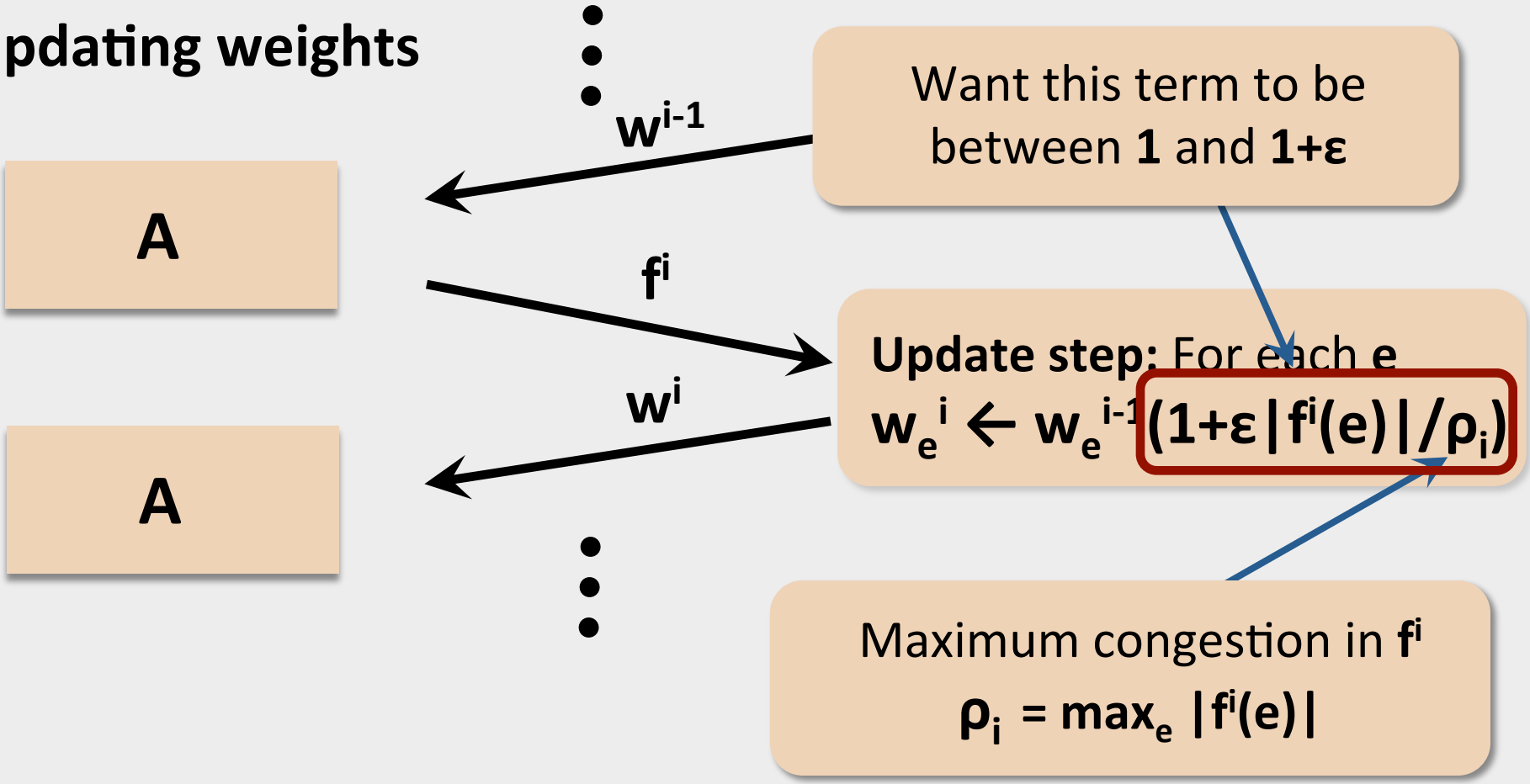
(1-ε)-approx. max flow

[(**1+ε)-approx**. feasibility **everywhere**]

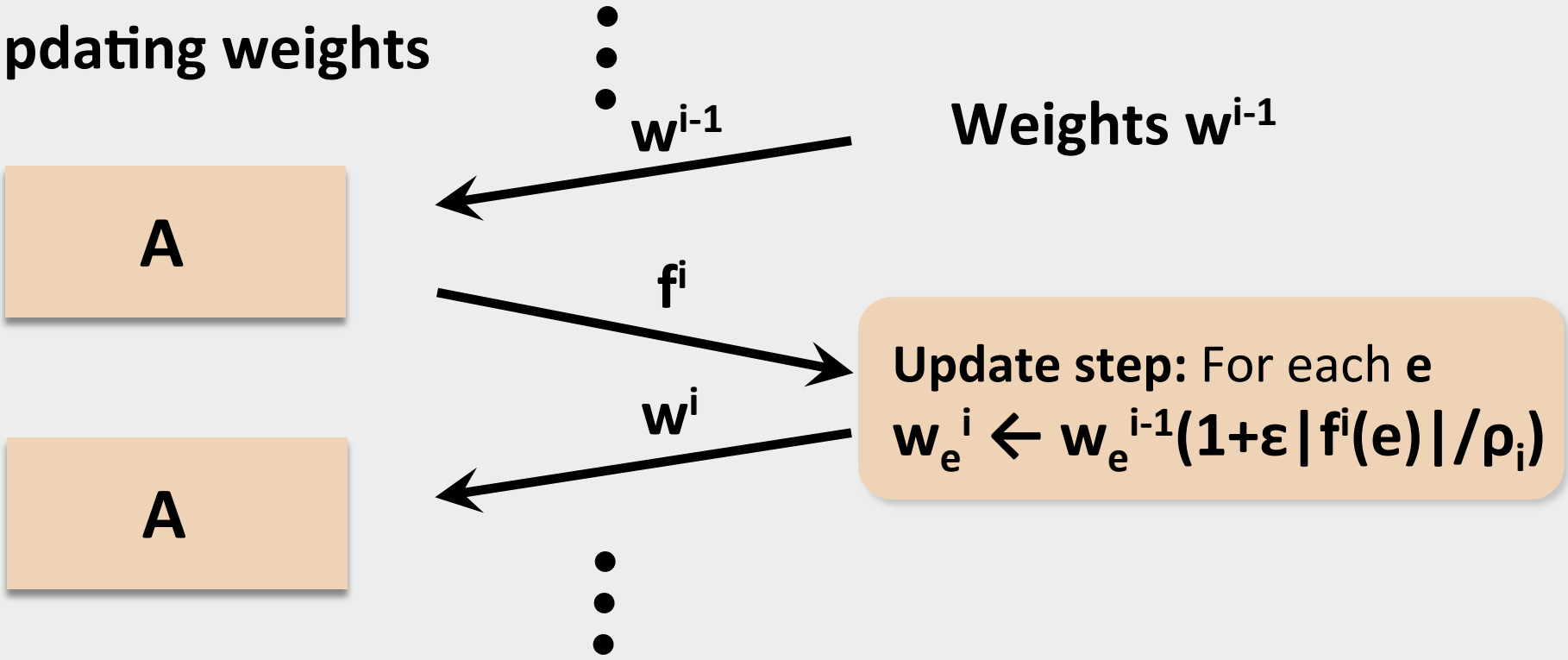
How does this method work?


Underlying idea
Crude algorithm
Maintain weights w
(Initially, all weights w_e=1)
w^0
A
f^1
feasible on average
Update weights
(based on f^1)
w^1
A
f^2
Update weights
(based on f^2)
w^3
A
(Process continues for N rounds)
At the end: Return the average of all f^is
(This is still a flow of value F*)

Updating weights



Updating weights



Underlying dynamics:

Edge **e suffers large** overflow → $\mathbf{w_e}$ grows **rapidly**

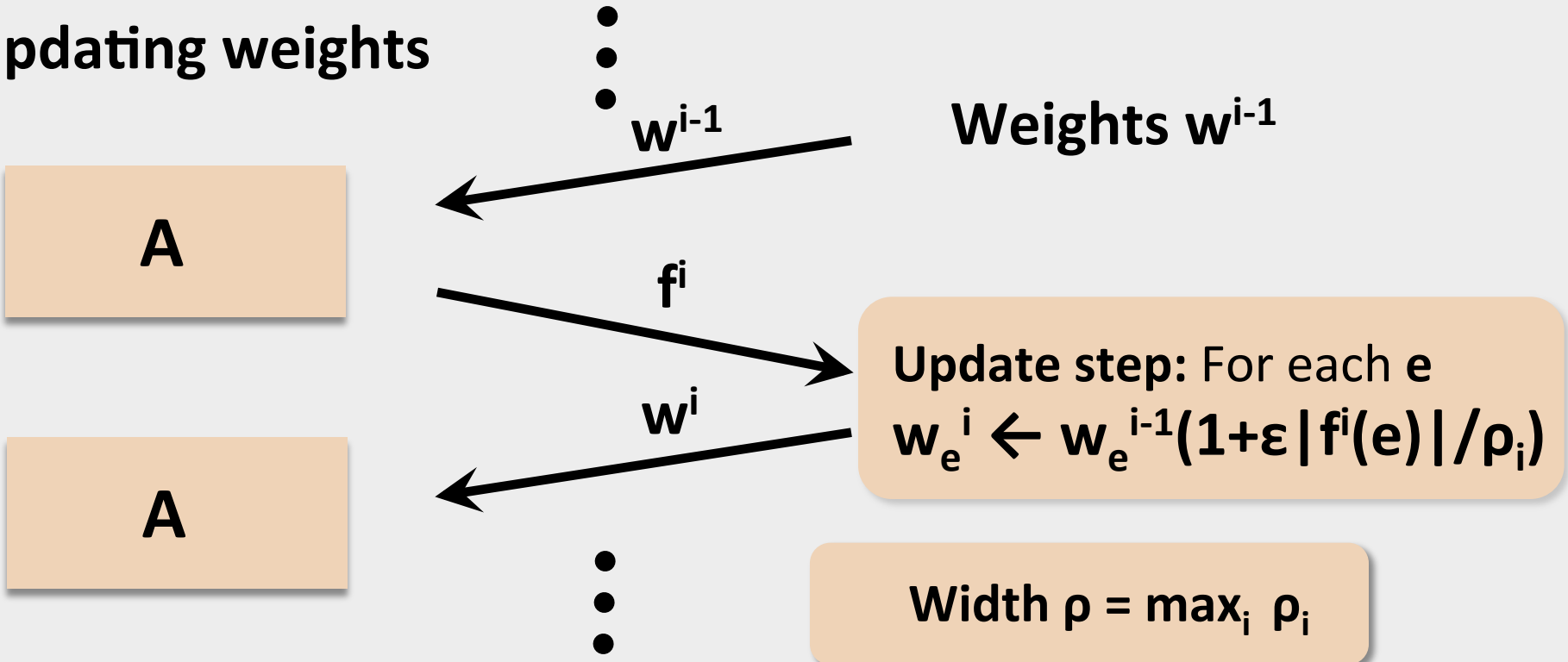
Average overflow **small** → $\mathbf{\Sigma_e w_e}$ grows **slowly**

↓

No edge suffers large overflow **too often**

→ **averaging out** yields (almost) no overflow

Updating weights

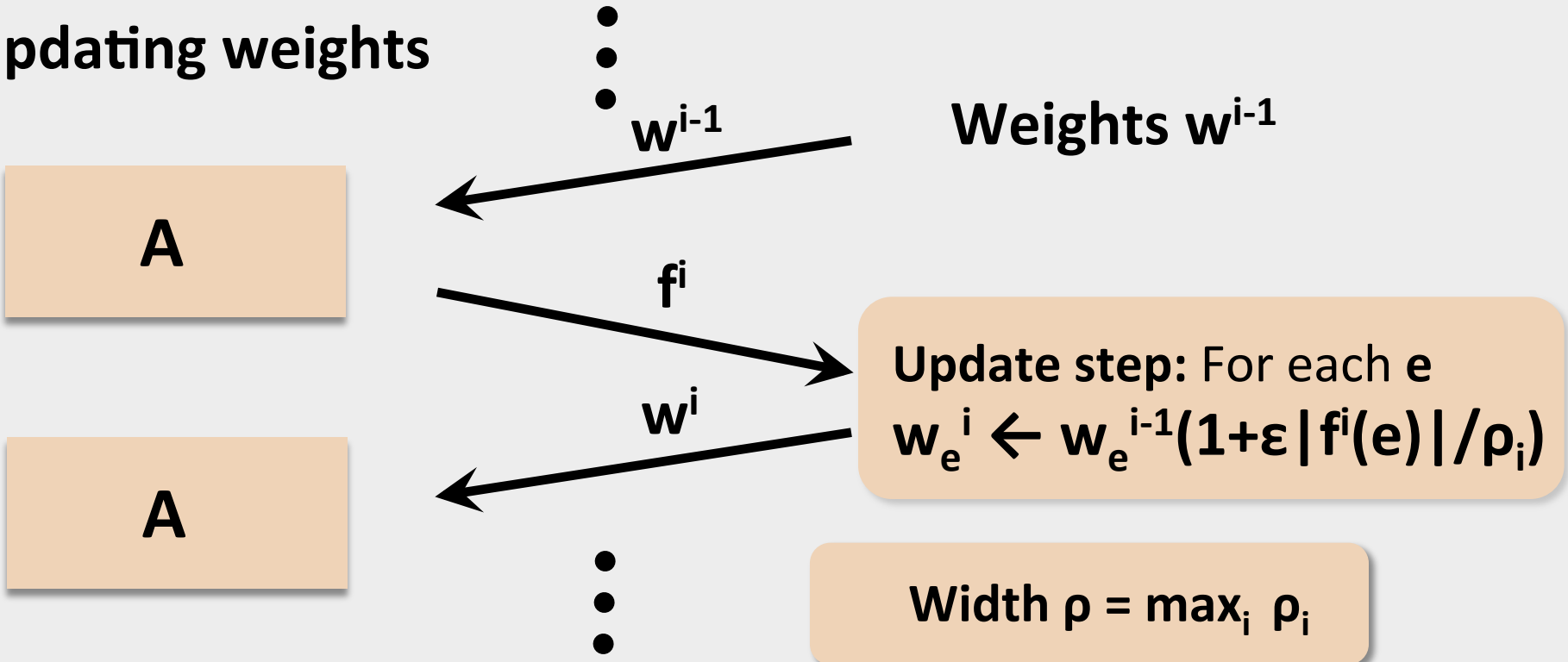


[AHK '05]: It suffices to repeat this step $N=\tilde{O}(\rho\varepsilon^{-2})$ times to get a **(1-ε)-approx** to max flow

Think: ρ measures the **electrical vs. max** flow discrepancy

Note: Linear dependence on **ρ** is unavoidable

Updating weights



[AHK '05]: It suffices to repeat this step $N=\tilde{O}(\rho\varepsilon^{-2})$ times to get a **(1-ε)-approx** to max flow

Bottom line:

A = 

Electrical flow primitive gives us the crude algorithm

We can use MWU framework
to fill in our blanks!

Multiplicative weight update method

- Arora, Hazan, Kale'12, "The multiplicative weight update method: a meta-algorithm and applications"
  http://theoryofcomputing.org/articles/v008a006/
- Madry'12, "How to get rich (if you have good advice", EPFL course notes
  http://thl.epfl.ch/gems/notes/lecture1.pdf

  [I am preparing a Persian translation of this article for http://sharifmathjournal.ir/]

Our algorithm

→ Treat edges as resistors of resistance $\mathbf{r_e=1}$

→ Compute electrical flow **f** of value $\mathbf{F^*}$

→ **Increase resistances** on overflowing edges

Repeat

Our algorithm

→ Treat edges as resistors of resistance $\mathbf{r_e=1}$

→ Compute electrical flow **f** of value $\mathbf{F^*}$

→ **Increase resistances:** for each **e**,

$$r_e^i \leftarrow r_e^{i-1}(1+\varepsilon|f^i(e)|/\rho_i)$$

Repeat $\mathbf{N=\tilde{O}(\rho\varepsilon^{-2})}$ **times**

→ **At the end:** Take an **average** of all the flows as the final answer

→ Resistances $\mathbf{r_e}$ evolve as weights $\mathbf{w_e}$

→ Convergence condition: "execute **N** rounds"

Our algorithm

→ Treat edges as resistors of resistance $\mathbf{r_e=1}$

→ Compute electrical flow **f** of value $\mathbf{F^*}$

→ **Increase resistances:** for each **e**,

$$r_e^i \leftarrow r_e^{i-1}(1+\varepsilon|f^i(e)|/\rho_i)$$

Repeat $\mathbf{N=\tilde{O}(\rho\varepsilon^{-2})}$ **times**

→ **At the end:** Take an **average** of all the flows as the final answer

Result: This algorithm gives us an **(1-ε)**-approx. max flow in $\mathbf{\tilde{O}(\rho\varepsilon^{-2})\cdot\tilde{O}(n) = \tilde{O}(n\rho\varepsilon^{-2})}$ time

Crucial question: How large the worst-case overflow **ρ** can be?

Our question: Let **f** be an elect. flow of value **F*** wrt resist. $\mathbf{r_e}$
How large $\boldsymbol{\rho = \max_e |f(e)|}$ can be?

In general: ρ can be very large
(**Think:** one edge having an extremely small resistance)

Fix: Regularize the resistances with a uniform distribution
$\mathbf{r_e' \leftarrow r_e + \varepsilon |r|_1 / m}$

Can show: ρ is bounded by $\mathbf{O(n^{½} \varepsilon^{-1})}$ then

Proof:

This gives a **(1-ε)-approx. $\tilde{O}(n^{3/2}\varepsilon^{-3})$-**time algorithm

Going beyond the $\tilde{O}(n^{3/2})$ Barrier

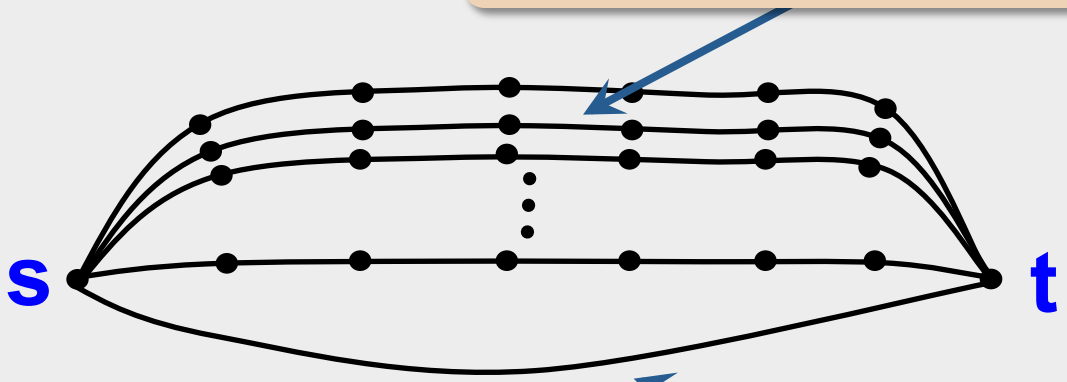
Speeding up our algorithm

Running time is dominated by $\approx\rho$ elect. flow computations

Can we improve our $O(n^{1/2}\varepsilon^{-1})$ bound on ρ?

Not really…

$\approx n^{1/2}$ paths with $\approx n^{1/2}$ vertices each



one edge

Speeding up our algorithm

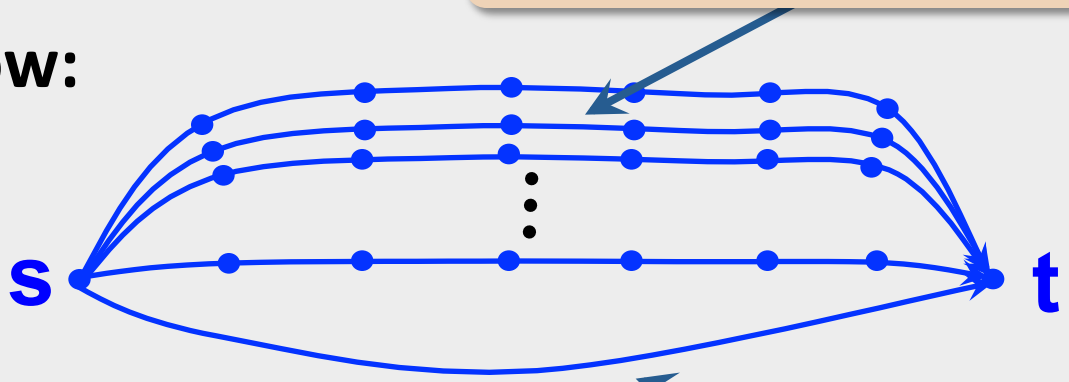
Running time is dominated by $\approx\rho$ elect. flow computations

Can we improve our $O(n^{1/2}\varepsilon^{-1})$ bound on ρ?

Not really…

$\approx n^{1/2}$ paths with $\approx n^{1/2}$ vertices each

Max flow:



$F^* \approx n^{1/2}$

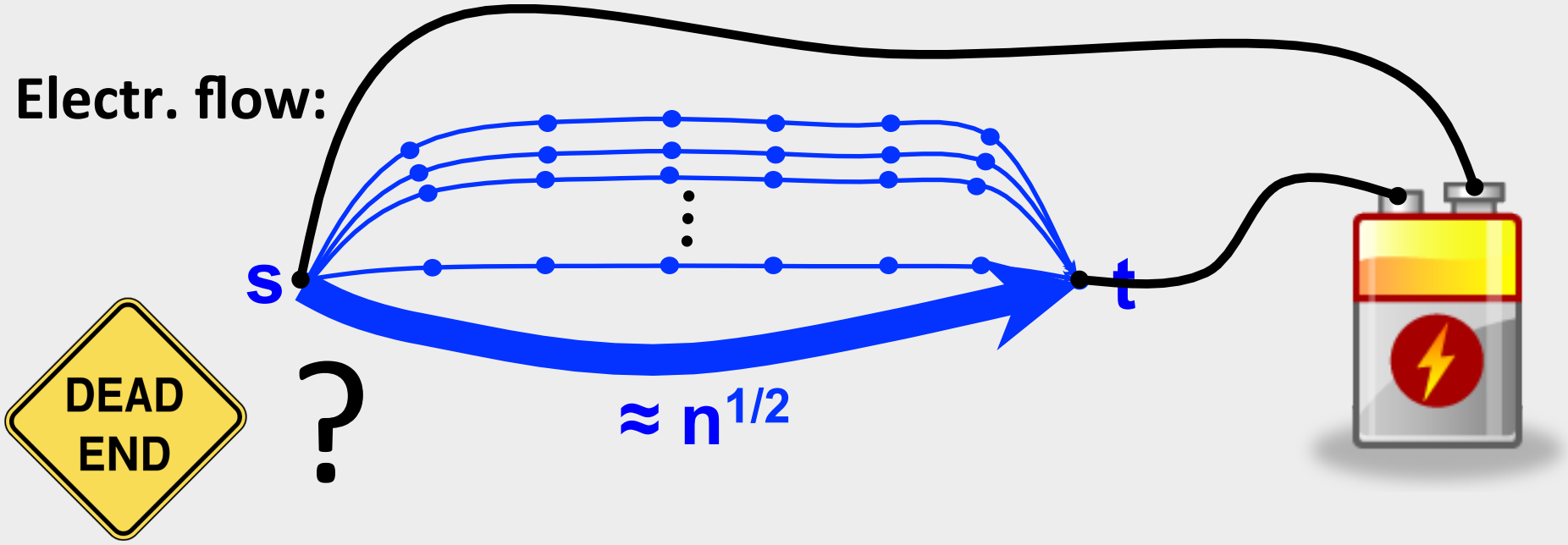
one edge

Speeding up our algorithm

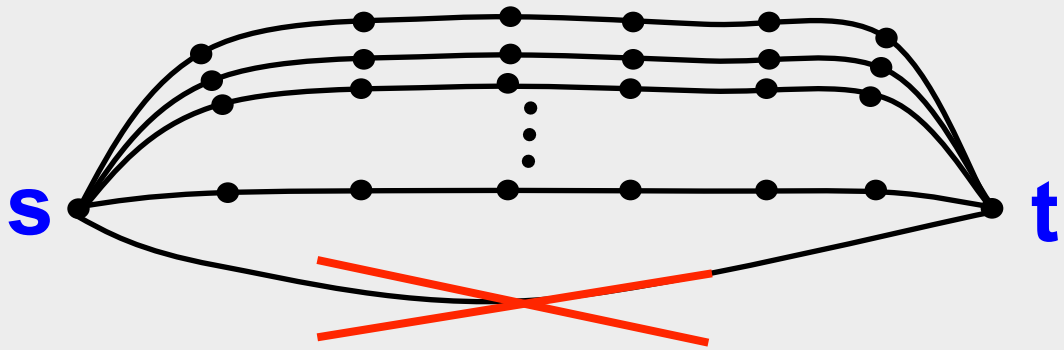
Running time is dominated by $\approx \rho$ elect. flow computations

Can we improve our $O(n^{1/2} \varepsilon^{-1})$ bound on ρ?

Not really…



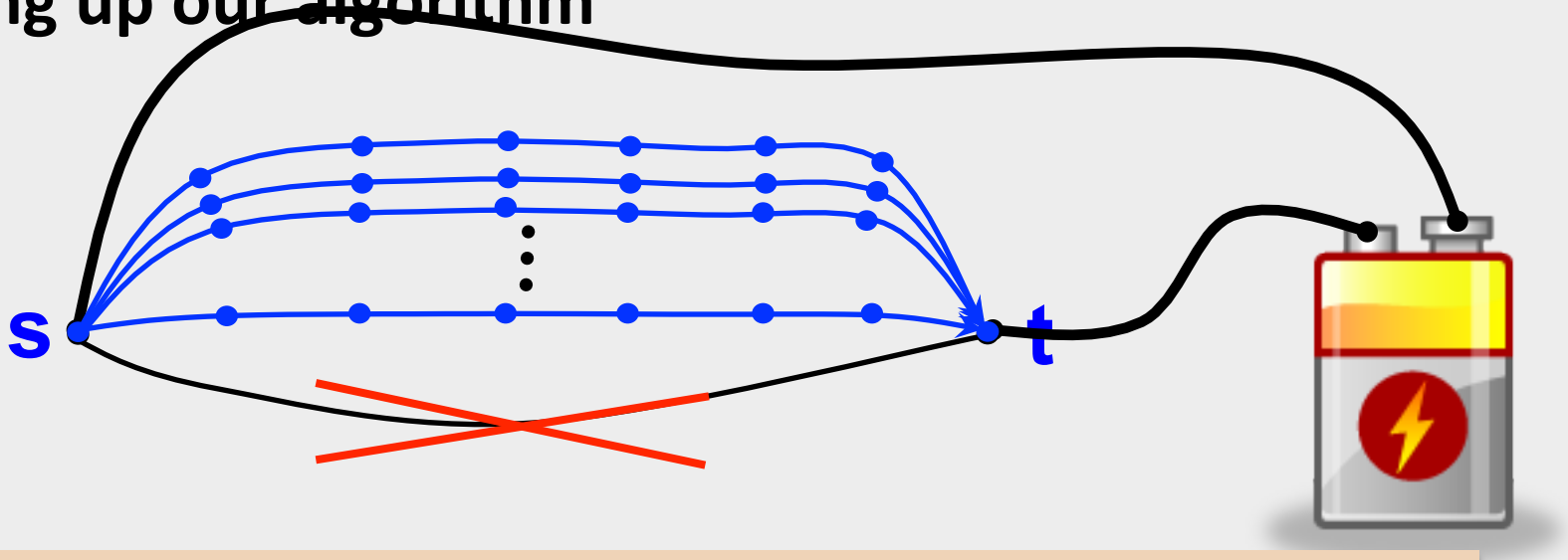
Speeding up our algorithm



Key observation: If we remove this bad edge…

→ The **max flow** does not change much

Speeding up our algorithm



Key observation: If we remove this bad edge…

→ The **max flow** does not change much

→ But the resulting **electrical flow** is much better behaved!

Can we turn this observation into an algorithmic idea?

Speeding up our algorithm

Idea: Let our electrical flow oracle **self-enforce** a smaller overflow **ρ' << ρ**

Modification of the oracle: If the computed electrical flow has some edge **e** flow more than **ρ':**

→ **Remove** this edge from the graph (permanently)

→ **Recompute** the electrical flow

Note: If this oracle always **successfully** terminates, its effective overflow is **ρ'**

Speeding up our algorithm

Crucial question: What is the right setting of **ρ'**?

→ We want **ρ'** to be as small as possible
→ But if it becomes too small the edge removal might be too aggressive and cut too many of them

Sweet spot: $\rho' \approx n^{1/3}$

Key reason: Removal of edges that flow a lot
→ significantly increases the **energy** of the electr. flow
→ But perturbs the **max flow** only slightly

Speeding up our algorithm

Our potential: The **energy $E_r(f)$** of the electrical flow **f** wrt current resistances **r**

Can show:

→ **$E_r(f)$** is not too small initially and cannot become too large (as long as we remove no more than ≈ **εF^*** edges)

→ As the resistances only increase, **$E_r(f)$** never decreases

This makes $E_r(f)$ a convenient potential

Need to show: Removal of an overflowing edge increases **$E_r(f)$** significantly

Speeding up our algorithm

Need to show: Removal of an overflowing edge increases $\mathbf{E_r(f)}$ significantly

Fact: If an edge **e** contributes a **δ**-fraction of energy then removing it increases $\mathbf{E_r(f)}$ by a factor of $\mathbf{1+\Omega(\delta)}$

Further: If an edge **e** flows at least **ρ'** in **f** then its energy contribution is $\mathbf{\Delta=\Omega(\varepsilon(\rho')^2/n)}$

Speeding up our algorithm

Need to show: Removal of an overflowing edge increases $\mathbf{E_r(f)}$ significantly

Fact: If an edge **e** contributes a **δ**-fraction of energy then removing it increases $\mathbf{E_r(f)}$ by a factor of $\mathbf{1+\Omega(\delta)}$

Further: If an edge **e** flows at least **ρ'** in **f** then its energy contribution is $\mathbf{\Delta=\Omega(\varepsilon(\rho')^2/n)}$

Putting it all together: We can have $\mathbf{\leq \tilde{O}(\Delta^{-1})=\tilde{O}(n/\varepsilon(\rho')^2)}$ edge removals before $\mathbf{E_r(f)}$ grows by too much

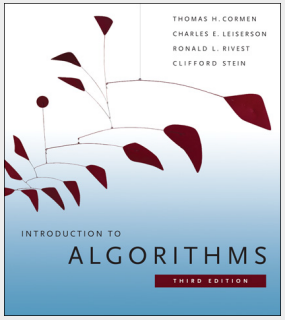
Taking $\mathbf{\rho' \approx n^{1/3}\varepsilon^{-1}}$ makes $\mathbf{\tilde{O}(\Delta^{-1})=\tilde{O}(\varepsilon n^{1/3})}$ be smaller than $\mathbf{\varepsilon F^* \geq \varepsilon\rho'}$ as needed

This gives the $\mathbf{\tilde{O}(n^{4/3}\varepsilon^{-3})}$**-time (1-ε)-approx.** algorithm

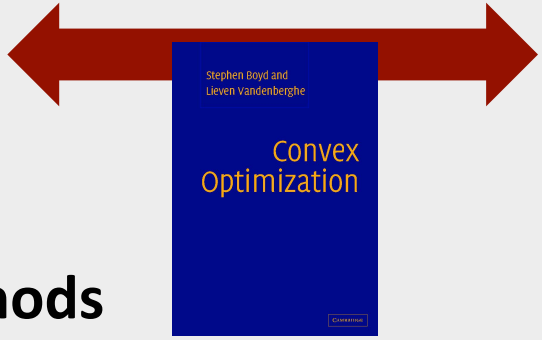
Main theorem

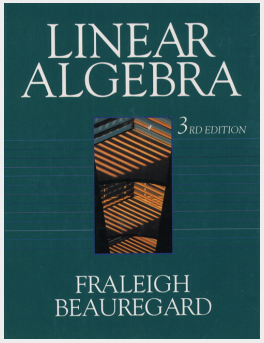
- **Theorem.** There is an algorithm such that, given an undirected, capacitated graph, two vertices s, t, and $\varepsilon>0$, with maximum s,t-flow value F*, outputs a feasible flow of value at least $(1-\varepsilon)F^*$ and has running time $O(m^{4/3}\varepsilon^{-3})$.
  [Christiano, Kelner, Madry and Spielman'11]
- Running time improved to $O(m(\log m)^{11}\varepsilon^{-3})$
  [Peng'16] http://arxiv.org/abs/1411.7631

Our goal: Forging the next generation of tools for graph algorithms



Combinatorial methods
(trees, paths, partitions, matchings, routings,…)





Linear-algebraic tools
(eigenvalues, electrical flows, linear systems,…)

Cont. opt. primitives
(gradient-descent, interior-point methods,…)

Underlying theme: Merging combinatorial and continuous methods

Linear Algebra & Algorithms

- Shayan Oveis Gharan 2015, “Asymmetric TSP”
  (with Nima Ahmadipour, gave O(log log n) approximation for asymmetric travelling salesman problem)
  https://www.msri.org/workshops/754/schedules/19574
- Workshop in Berkeley 2014: algorithmic spectral graph theory
  https://simons.berkeley.edu/workshops/spectral2014-boot-camp
  - James Lee,“The Unreasonable Effectiveness of Spectral Graph Theory” https://simons.berkeley.edu/events/openlectures2014-fall-4
  - Aleksander Madry (3 talks): Maximum flow
    https://simons.berkeley.edu/talks/electrical-flows-optimization-and-new-approaches-maximum-flow-problem
  - Jon Kelner: Multicommodity flow
    https://simons.berkeley.edu/talks/jon-kelner-2014-12-01
  - Nikhil Srivastava (3 talks): graph sparsification via random matrices
    https://simons.berkeley.edu/talks/graph-sparsification
- More examples on Spielman’s page on Laplacian solvers (eg. Clustering algorithms) https://sites.google.com/a/yale.edu/laplacian/
- Vishnoi 2012 book “Laplacian solvers and their algorithmic applications” (eg graph partitioning) http://theory.epfl.ch/vishnoi/Lxb-Web.pdf
- Kannan-Vempala 2013 book “Spectal algorithms” http://www.cc.gatech.edu/~vempala/spectral/