# Online bipartite matching with $O(\log^2 n)$ replacements.

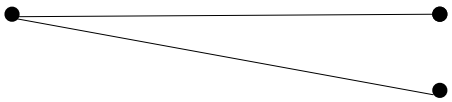
Aaron Bernstein – Technical University of Berlin
Jacob Holm – University of Copenhagen
Eva Rotenberg – University of Copenhagen

# Standard Online Matching

Bipartite Graph

- right side: **servers** are fixed from the beginning
- left side: **clients** arrive one a time with all incident edges

# Standard Online Matching



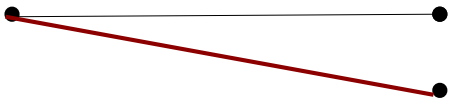
Bipartite Graph

- right side: **servers** are fixed from the beginning
- left side: **clients** arrive one a time with all incident edges

When a Client Arrives:

- Choose a neighboring server to match to (if available)
- *Can never take back choices*

# Standard Online Matching



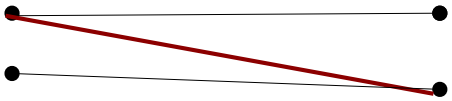
Bipartite Graph

- right side: **servers** are fixed from the beginning
- left side: **clients** arrive one a time with all incident edges

When a Client Arrives:

- Choose a neighboring server to match to (if available)
- *Can never take back choices*

# Standard Online Matching



Bipartite Graph

- right side: **servers** are fixed from the beginning
- left side: **clients** arrive one a time with all incident edges

When a Client Arrives:

- Choose a neighboring server to match to (if available)
- *Can never take back choices*

Mistakes are unavoidable: must settle for an approximation

# Allowing Recourse

## Example Application: Advertisements and Users

- When user arrives, select one of the advertisements
- "take backs" are clearly not possible

# Allowing Recourse

## Example Application: Advertisements and Users

- When user arrives, select one of the advertisements
- "take backs" are clearly not possible

## Example Application: Assigning Tasks to Machines

- When a task arrives, assign it to some machine
- Can later run the task on a different machine
  - incurs overhead

# Allowing Recourse

## Example Application: Advertisements and Users

- When user arrives, select one of the advertisements
- "take backs" are clearly not possible

## Example Application: Assigning Tasks to Machines

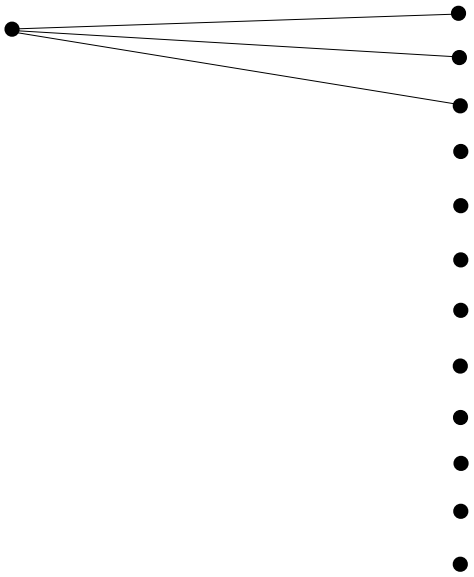
- When a task arrives, assign it to some machine
- Can later run the task on a different machine
  - incurs overhead

## Online Matching With Recourse

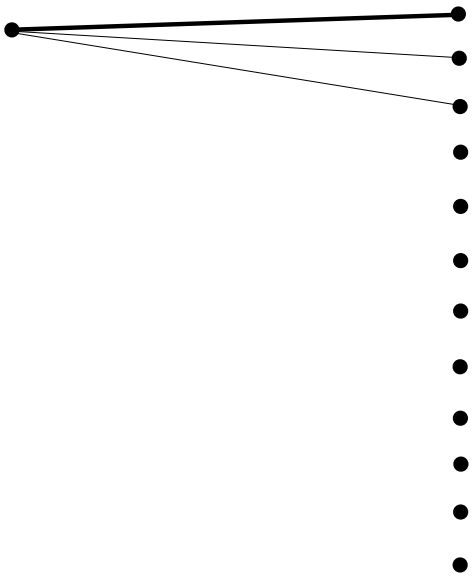
- Can change the matching as new clients arrive
- Must always maintain an *exact* maximum matching
- **Goal:** Minimize total number of changes

# Example of online matching with recourse

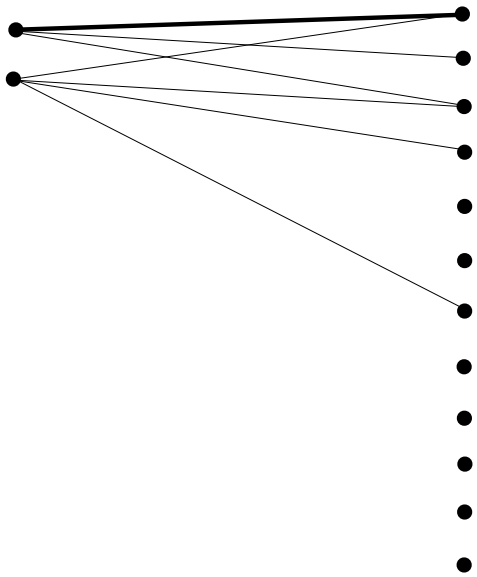
# Example of online matching with recourse



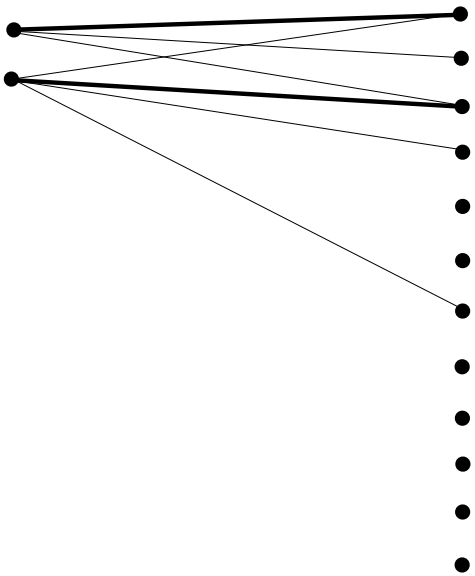
# Example of online matching with recourse



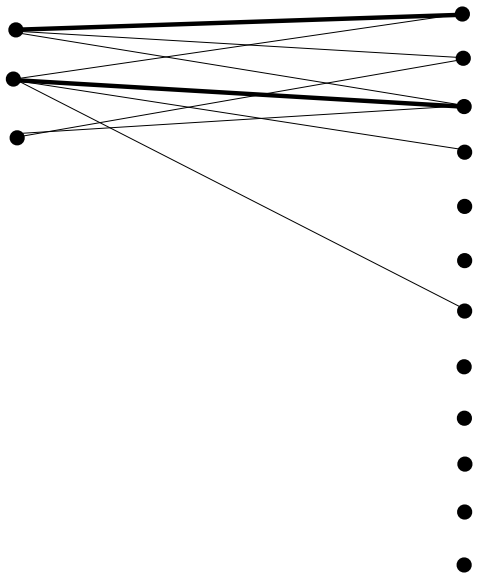
# Example of online matching with recourse



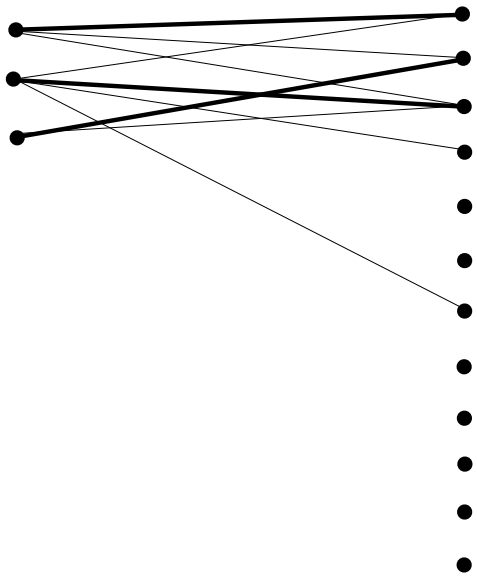
# Example of online matching with recourse



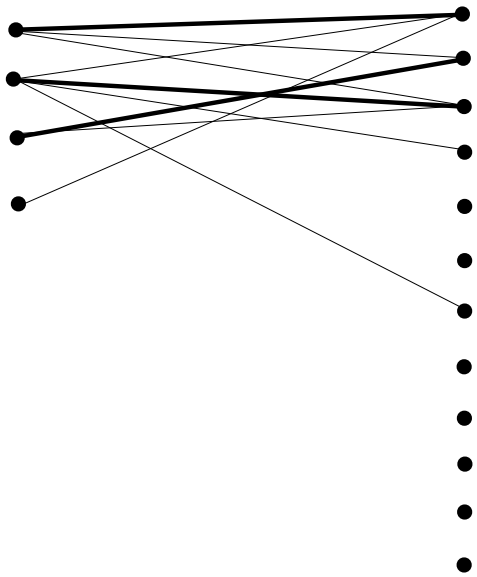
# Example of online matching with recourse



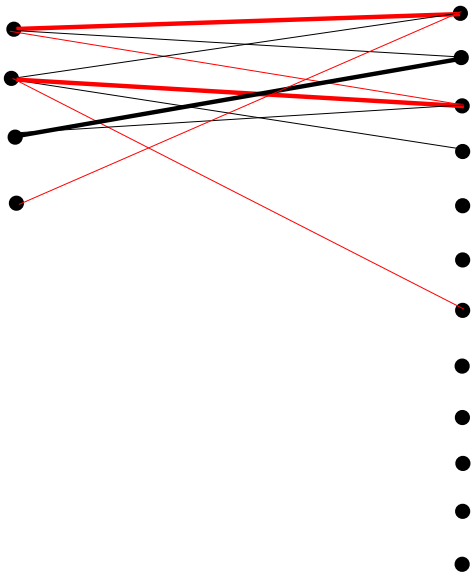
# Example of online matching with recourse



# Example of online matching with recourse

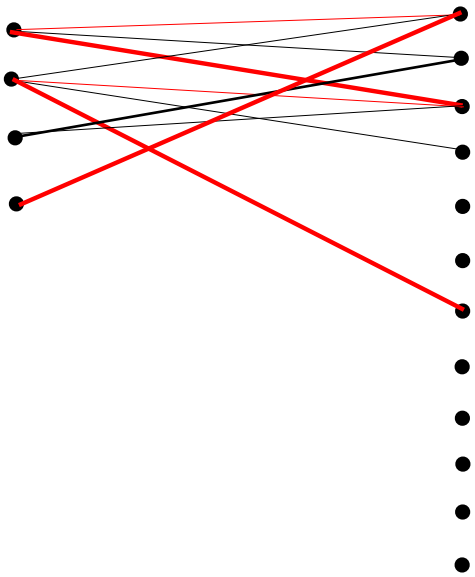


# Example of online matching with recourse



Must reassign 2 clients

# Example of online matching with recourse



## Task

- Find an augmenting path for each arriving client
- **Goal:** minimize total length of augmenting paths

# With and Without Recourse: Comparison

## Online Matching Without Recourse

- never reassign clients
- **Goal:** good approximation
  - Deterministic: .5-approx.
  - Randomized: .63 approx. [Karp, Vazirani, Vazirani, '90]

## Online Matching With Recourse

- Occasionally reassign clients
- **Requirement:** Must maintain *exact* maximum matching
- **Goal:** make as few reassignments as possible

# With and Without Recourse: Comparison

## Online Matching Without Recourse

- never reassign clients
- **Goal:** good approximation
  - Deterministic: .5-approx.
  - Randomized: .63 approx. [Karp, Vazirani, Vazirani, '90]

## Online Matching With Recourse

- Occasionally reassign clients
- **Requirement:** Must maintain *exact* maximum matching
- **Goal:** make as few reassignments as possible

Other Problems with Recourse

- MST, Max Flow, Scheduling, Steiner Tree, Set Cover
- **goal:** bound how often the optimal solution changes

# Outline

# Outline

# Shortest Augmenting Path

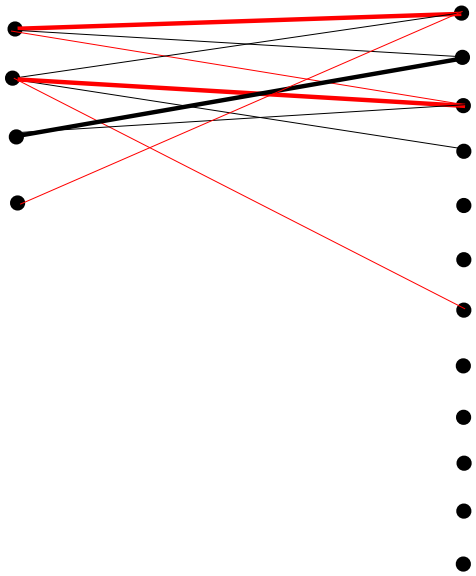
## Assumption for Remainder of Talk

- Always possible to match all clients

# Shortest Augmenting Path

## Assumption for Remainder of Talk

- Always possible to match all clients



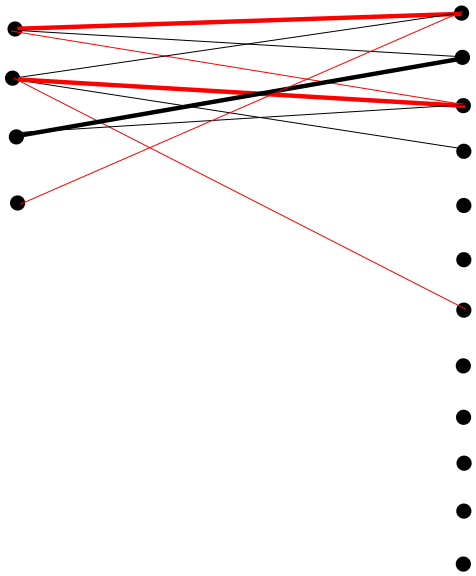
## Arrival of a New Client

- **Must** find augmenting path
- can choose which augmenting path
- Natural choice: **shortest** augmenting path
  - this is the greedy solution
  - denoted SAP

# Shortest Augmenting Path

## Assumption for Remainder of Talk

- Always possible to match all clients



## Arrival of a New Client

- **Must** find augmenting path
- can choose which augmenting path
- Natural choice: **shortest** augmenting path
  - this is the greedy solution
  - denoted SAP

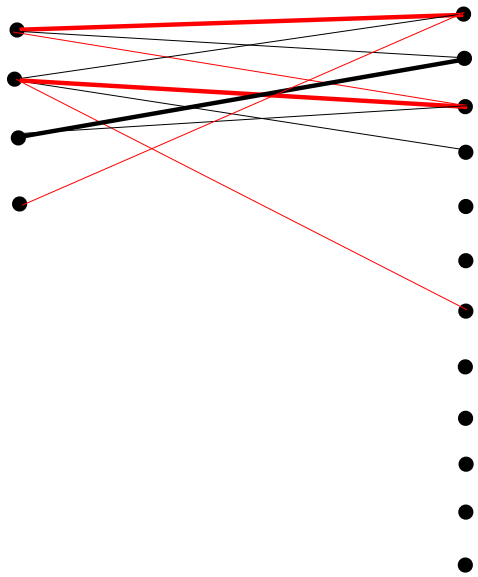
## Main Question

What is the total length of augmenting paths chosen by SAP?

# Shortest Augmenting Path

## Assumption for Remainder of Talk

- Always possible to match all clients



**Conjecture (2009):** SAP is asymptotically optimal

## Arrival of a New Client

- **Must** find augmenting path
- can choose which augmenting path
- Natural choice: **shortest** augmenting path
  - this is the greedy solution
  - denoted SAP

## Main Question

What is the total length of augmenting paths chosen by SAP?

# Results (n is the total number of clients)

## Lower Bound (Grove, Kao, Krishnan, Vitter - 1995)

$\Omega(n \log n)$ *total number of matching changes (amortized* $\log n$*).*

# Results (n is the total number of clients)

## Lower Bound (Grove, Kao, Krishnan, Vitter - 1995)

$\Omega(n \log n)$ *total number of matching changes (amortized* $\log n$*).*

## Theorem (Chaudhuri, Daskalakis, Kleinberg, Lin - 2009)

*SAP:* $O(n \log(n))$ *total changes when clients arrive in a random order*

# Results (n is the total number of clients)

## Lower Bound (Grove, Kao, Krishnan, Vitter - 1995)

$\Omega(n \log n)$ *total number of matching changes (amortized* $\log n$*).*

## Theorem (Chaudhuri, Daskalakis, Kleinberg, Lin - 2009)

*SAP:* $O(n \log(n))$ *total changes when clients arrive in a random order*

## Theorem (Bosek, Leniowski, Sankowski, Zych - 2015)

*SAP:* $O(n \log^2(n))$ *total changes for trees*

- *very recently: improved to* $O(n \log(n))$ *for trees*

# Results (n is the total number of clients)

## Lower Bound (Grove, Kao, Krishnan, Vitter - 1995)

$\Omega(n \log n)$ *total number of matching changes (amortized* $\log n$*).*

## Theorem (Chaudhuri, Daskalakis, Kleinberg, Lin - 2009)

*SAP:* $O(n \log(n))$ *total changes when clients arrive in a random order*

## Theorem (Bosek, Leniowski, Sankowski, Zych - 2015)

*SAP:* $O(n \log^2(n))$ *total changes for trees*

- *very recently: improved to* $O(n \log(n))$ *for trees*

## Theorem (Bosek, Leniowski, Sankowski, Zych - 2014)

$O(n\sqrt{n})$ *total number of changes (not SAP).*

# Results (n is the total number of clients)

## Lower Bound (Grove, Kao, Krishnan, Vitter - 1995)

$\Omega(n \log n)$ *total number of matching changes (amortized* $\log n$*).*

## Theorem (Chaudhuri, Daskalakis, Kleinberg, Lin - 2009)

*SAP:* $O(n \log(n))$ *total changes when clients arrive in a random order*

## Theorem (Bosek, Leniowski, Sankowski, Zych - 2015)

*SAP:* $O(n \log^2(n))$ *total changes for trees*

- *very recently: improved to* $O(n \log(n))$ *for trees*

## Theorem (Bosek, Leniowski, Sankowski, Zych - 2014)

$O(n\sqrt{n})$ *total number of changes (not SAP).*

## Our Result

SAP: $O(n \log^2 n)$ total number of changes ($\log^2 n$ amortized)

# Outline

# Matching Obliviousness

Worst Case: cannot avoid $\Omega(n)$ changes for a single client insertion

# Matching Obliviousness

Worst Case: cannot avoid $\Omega(n)$ changes for a single client insertion

## Not Oblivious: Tracking the Matching

- Charge changes to some potential function $\Phi(G, M)$
  - Example: track how often each client has been reassigned
  - Example: track "heights" of vertices in the current matching.
- SAP hard to analyze because matching changes erratically.

# Matching Obliviousness

Worst Case: cannot avoid $\Omega(n)$ changes for a single client insertion

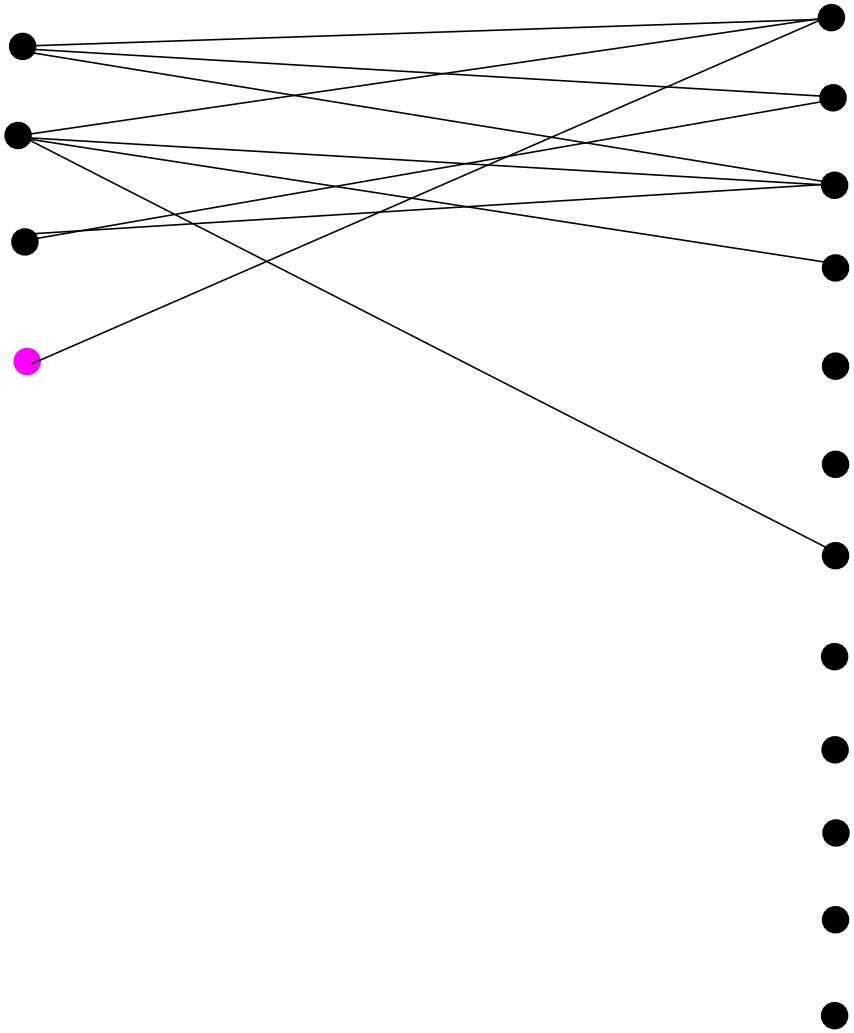
## Not Oblivious: Tracking the Matching

- Charge changes to some potential function $\Phi(G, M)$
  - Example: track how often each client has been reassigned
  - Example: track "heights" of vertices in the current matching.
- SAP hard to analyze because matching changes erratically.

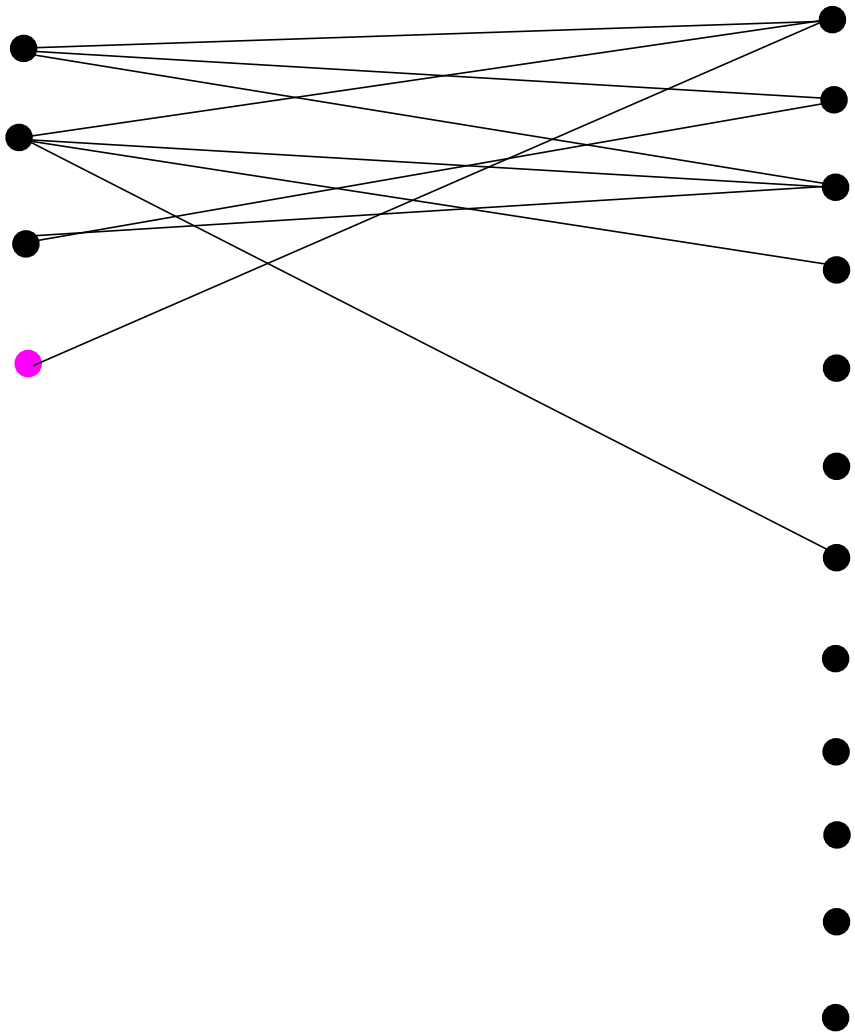
## Our Result: Matching Oblivious

- Does not need to "remember" matching between steps.
- Charges changes to potential $\Phi(G)$.

# Oblivious Augmenting Paths



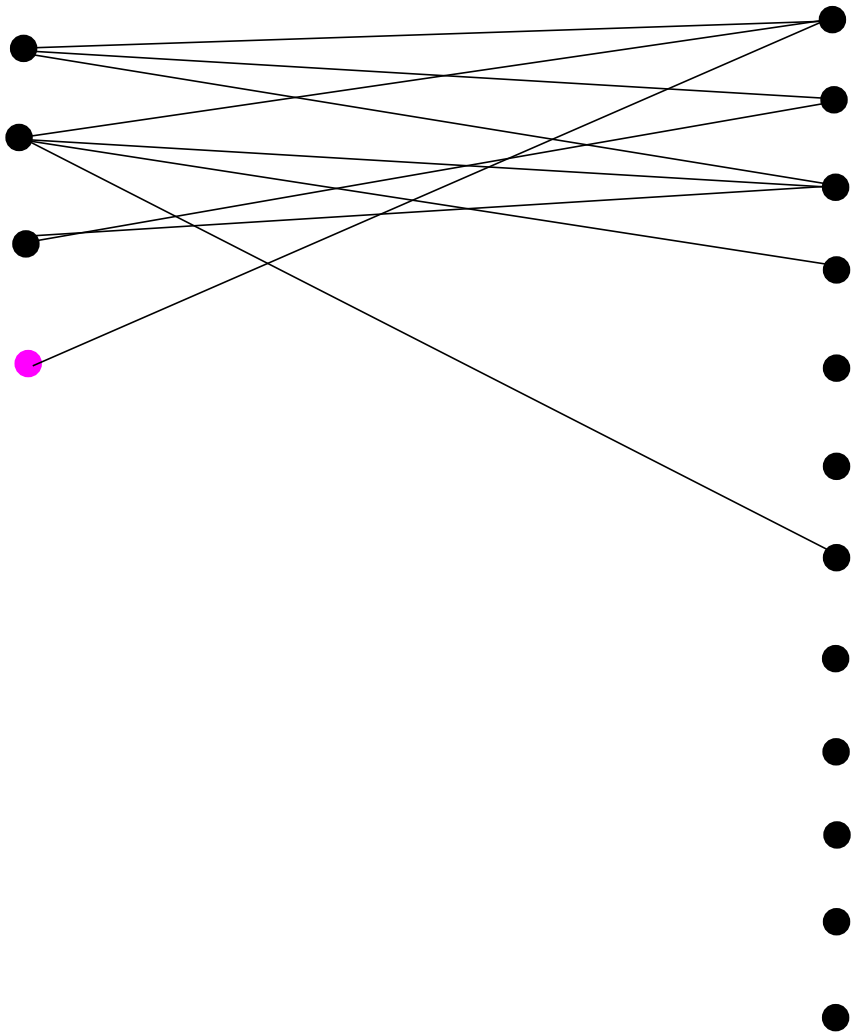
# Oblivious Augmenting Paths



**Cost of Inserted Client $c$**

length of shortest augmenting path from $c$ in the *current* matching

# Oblivious Augmenting Paths



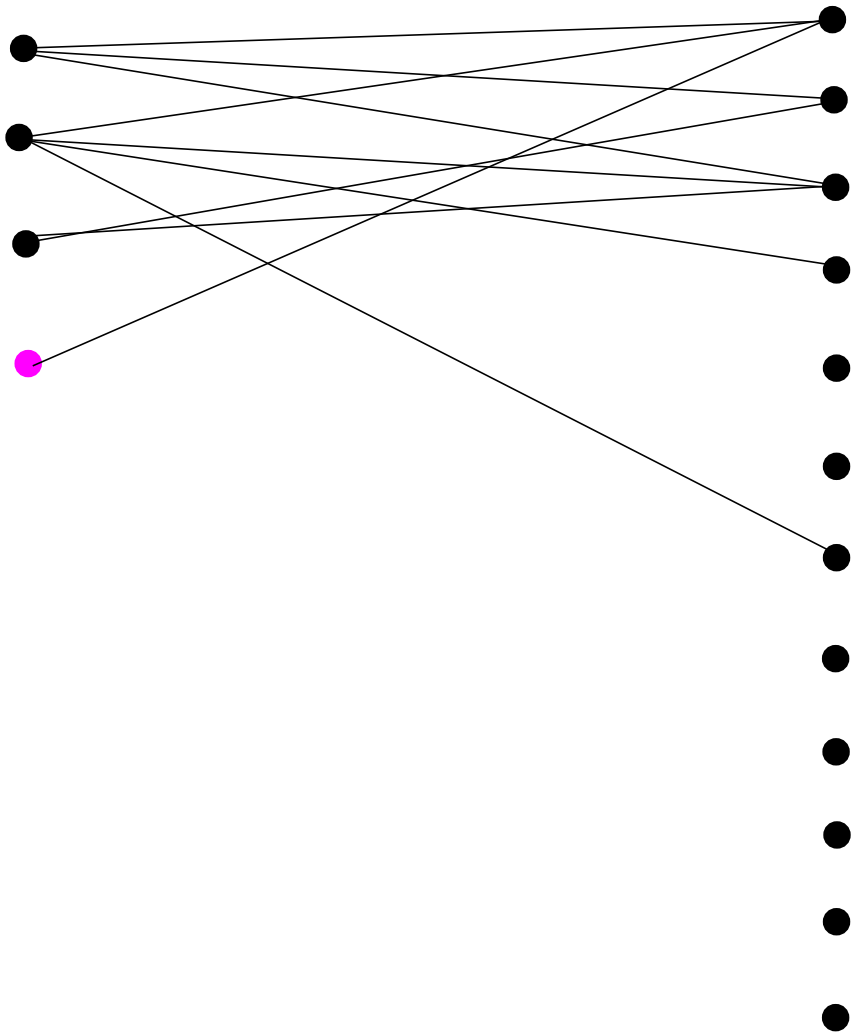
## Cost of Inserted Client $c$

length of shortest augmenting path from $c$ in the *current* matching

## Upper Bounding the Cost

$\max_{\text{all possible matchings M}}$ shortest augmenting path from $c$ in $M$

# Oblivious Augmenting Paths



## Cost of Inserted Client $c$

length of shortest augmenting path from $c$ in the *current* matching

## Upper Bounding the Cost

$\max_{\text{all possible matchings M}}$ shortest augmenting path from $c$ in $M$

## Main Analytic Question

Under what conditions does a newly inserted client $c$ have a short augmenting path in ALL possible matchings?

# Outline

# Server Necessities: Vague Definition

Simplifying Assumption: It is always possible to match every client

## Server Necessities

Can ask about each server $s$: "how necessary is $s$ for ensuring that every client can be matched"

# Server Necessities: Vague Definition

Simplifying Assumption: It is always possible to match every client

## Server Necessities

Can ask about each server $s$: "how necessary is $s$ for ensuring that every client can be matched"

## Example: Complete Bipartite Graph with $n$ servers and $n$ clients

All servers have necessity 1

## Example: Complete Bipartite Graph with $2n$ servers and $n$ clients

All servers have necessity $1/2$

# The necessity $\alpha(s)$ of a server $s$.

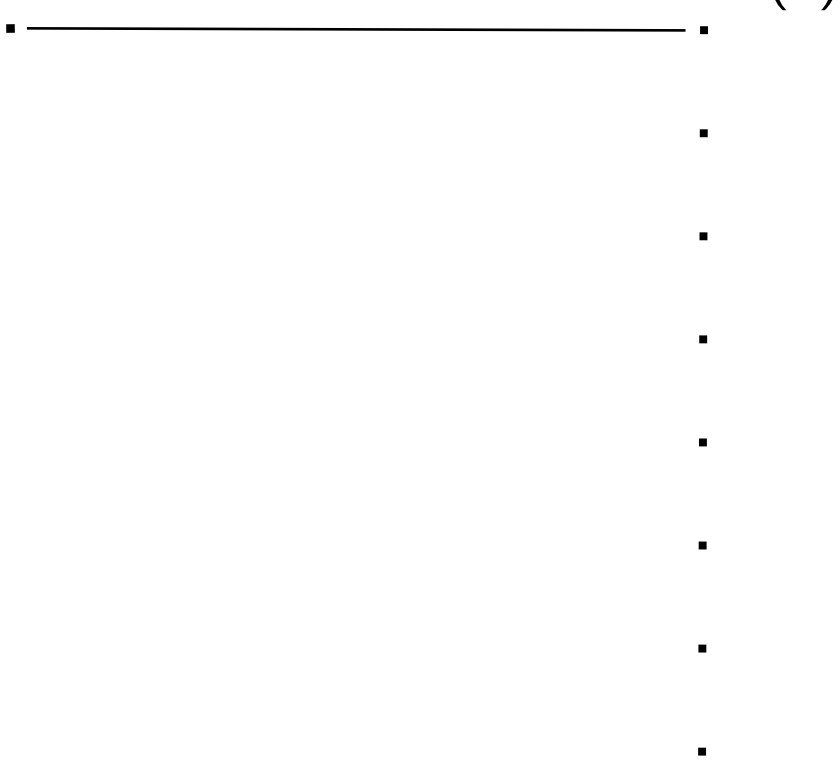
Client Server $\alpha(s)$



**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

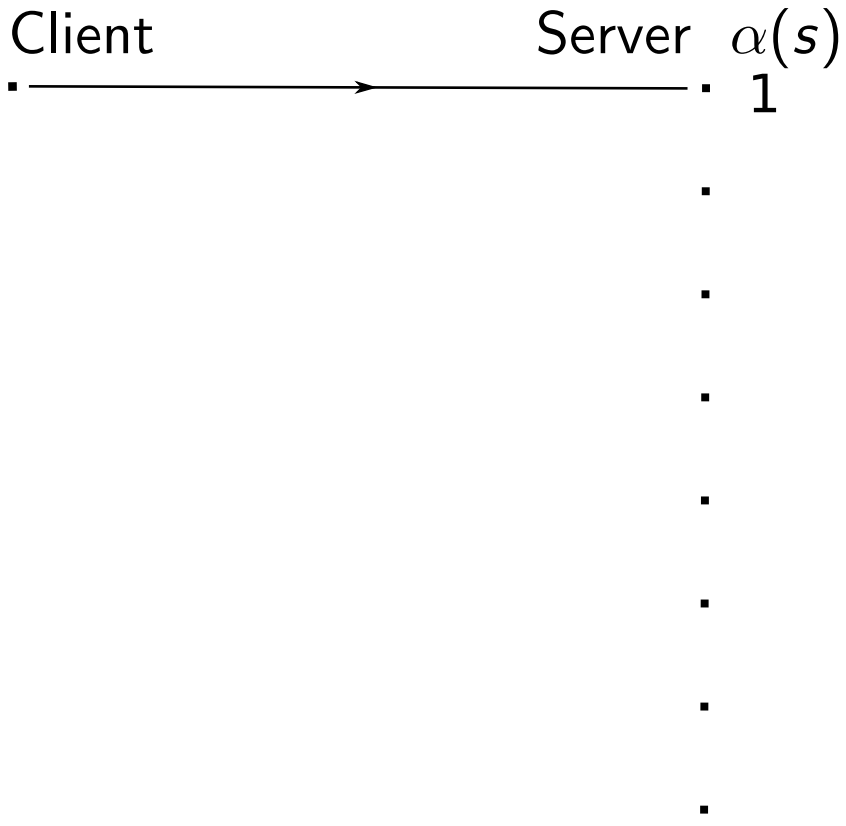
# The necessity $\alpha(s)$ of a server $s$.




**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

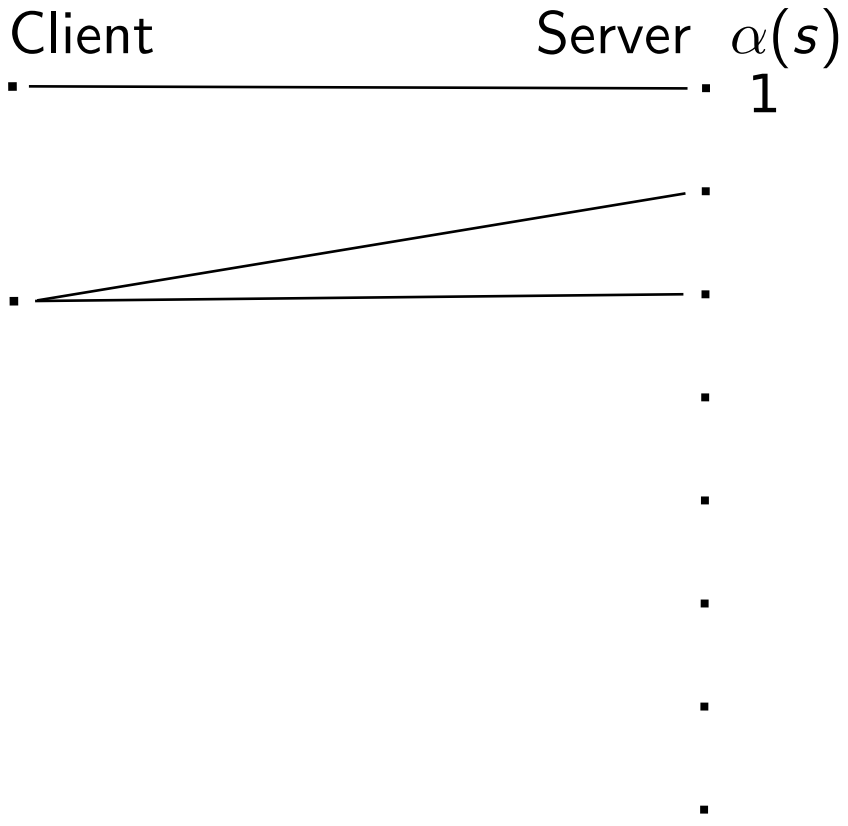
# The necessity $\alpha(s)$ of a server $s$.




**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

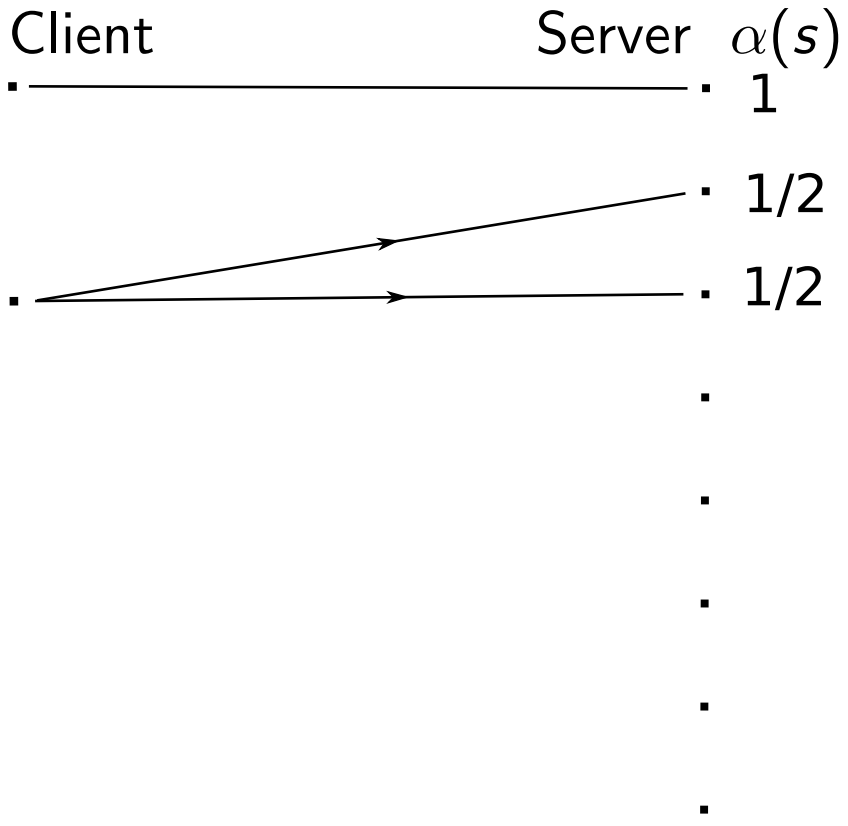
# The necessity $\alpha(s)$ of a server $s$.




**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

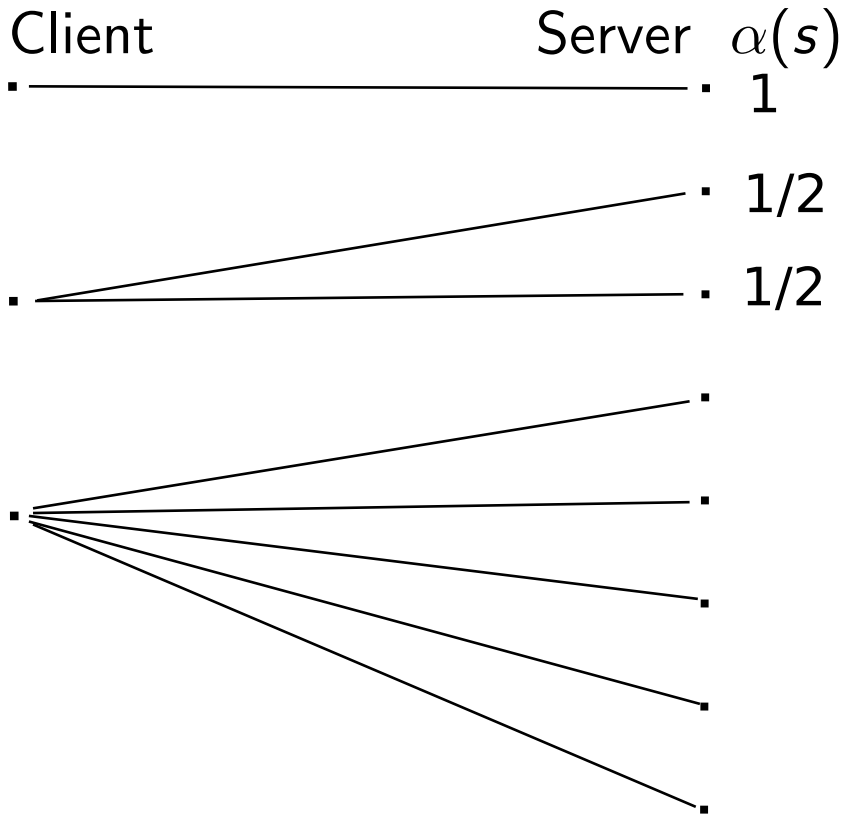
# The necessity $\alpha(s)$ of a server $s$.




**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

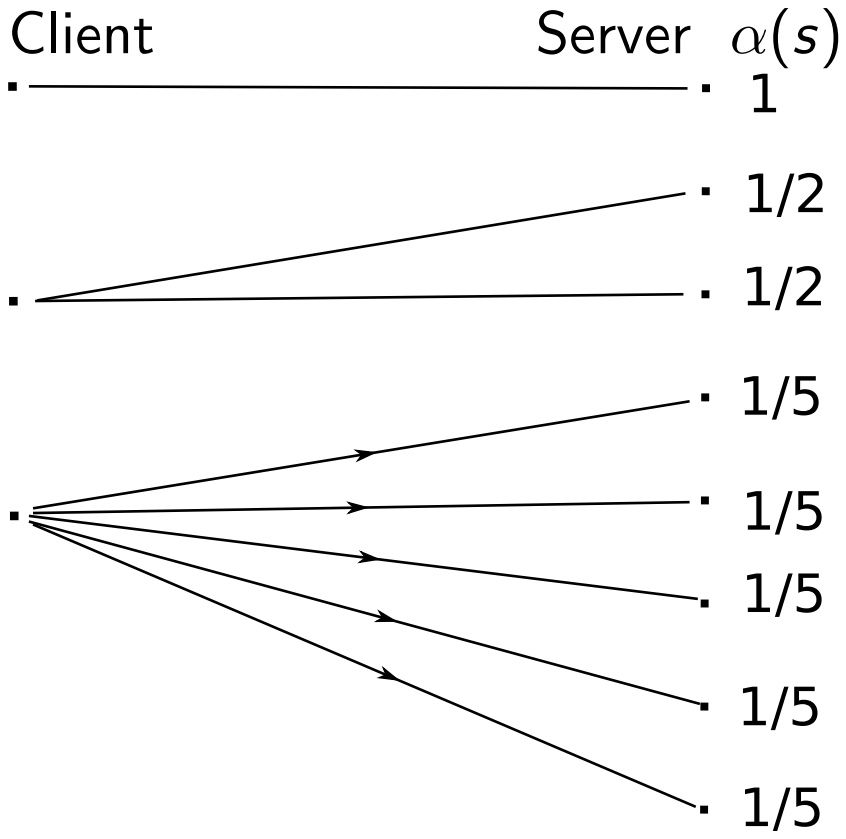
# The necessity $\alpha(s)$ of a server $s$.




**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.



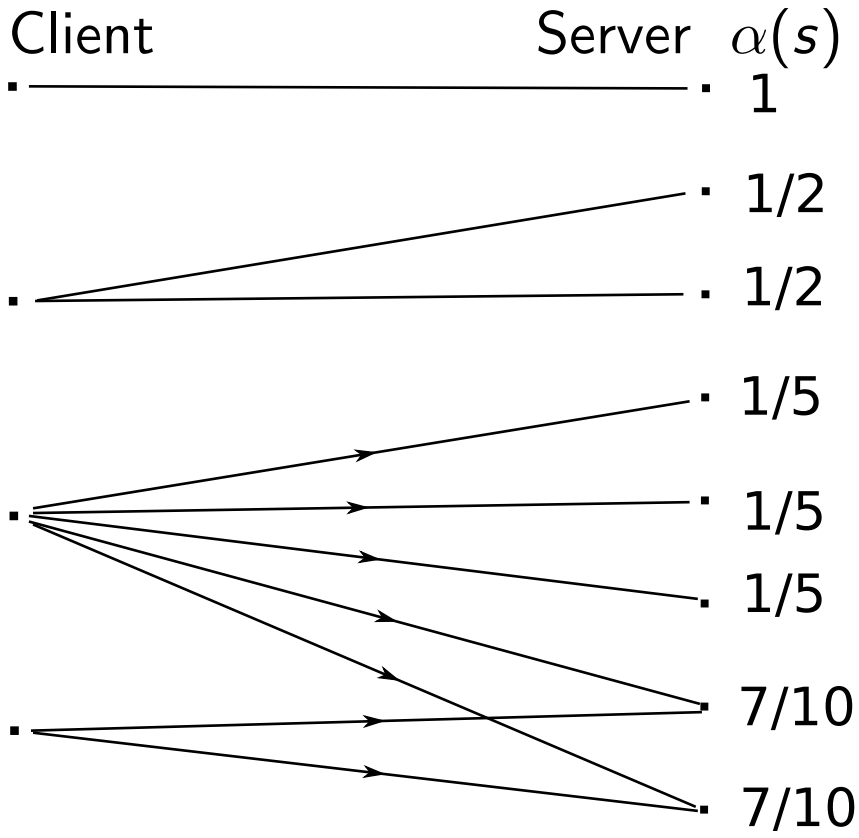

**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.



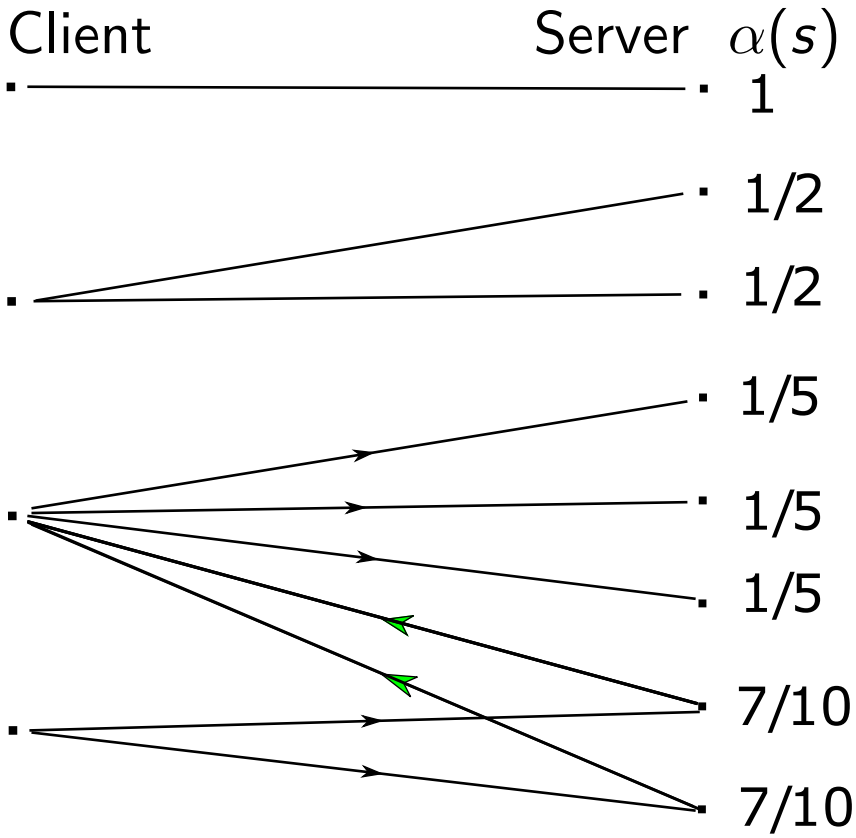

**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.



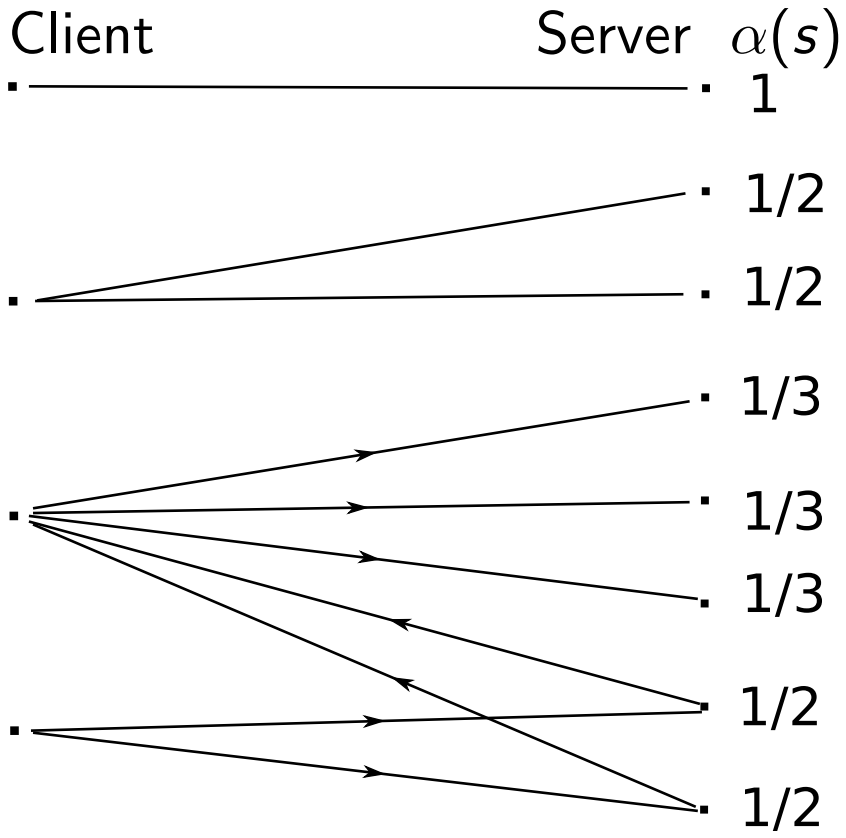

**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.



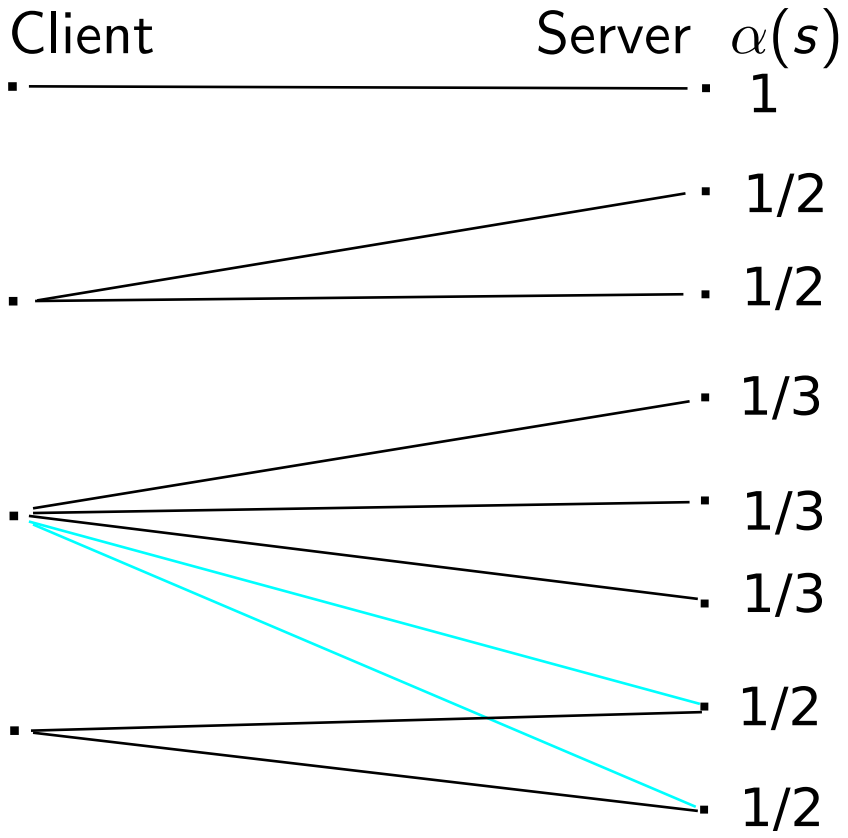

**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.



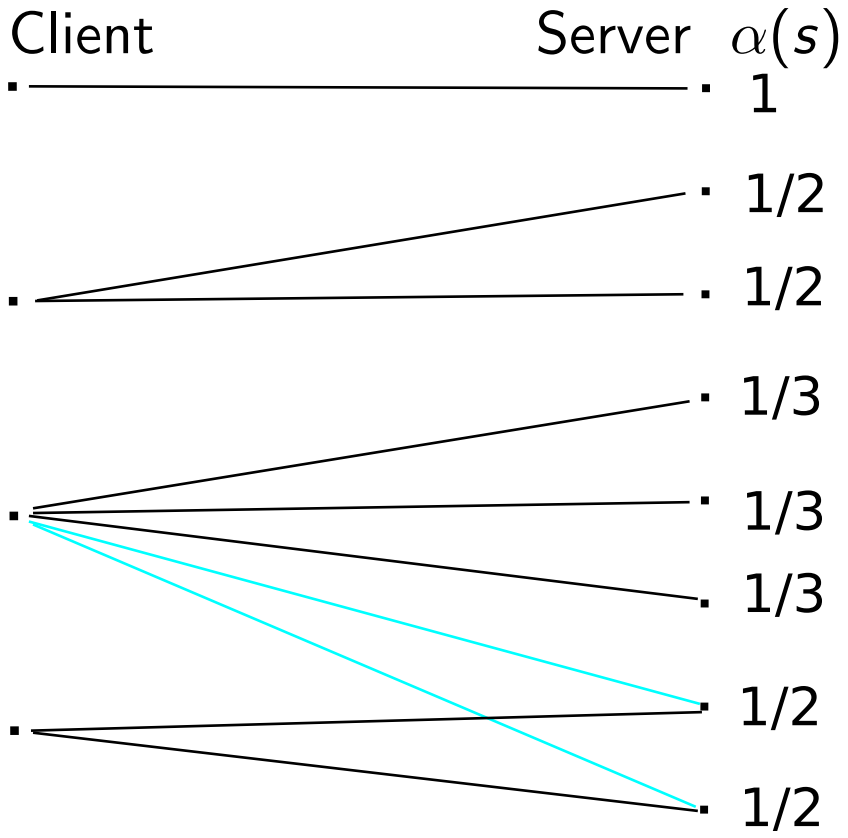

**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.



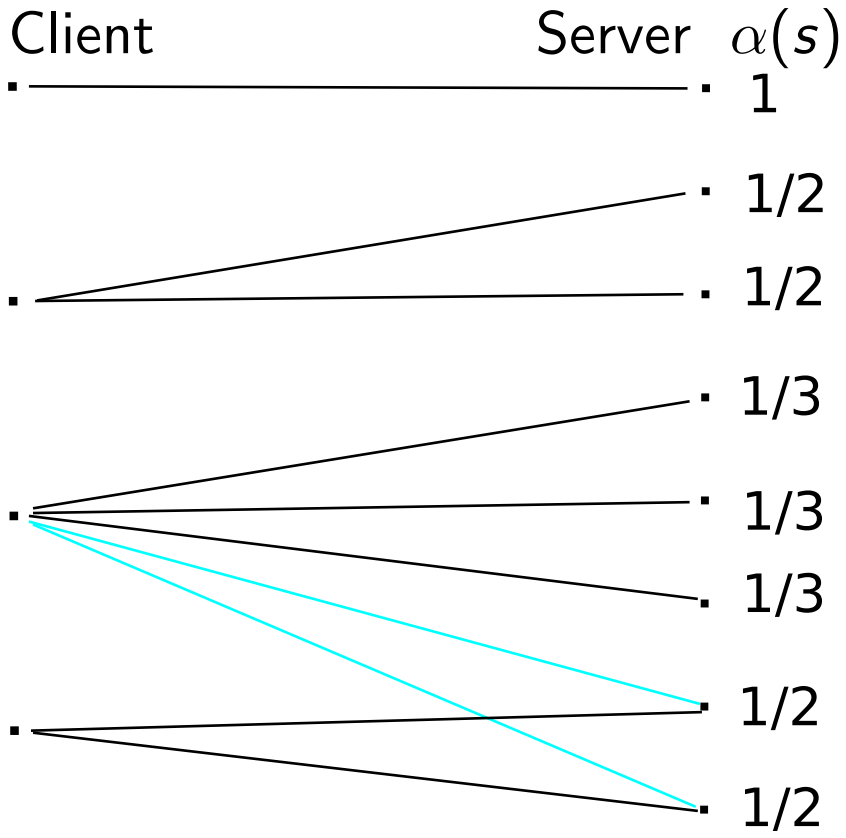

**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.




**Definition (Balanced flow)**

Minimising $\sum_{s \in S} (\alpha(s))^2$

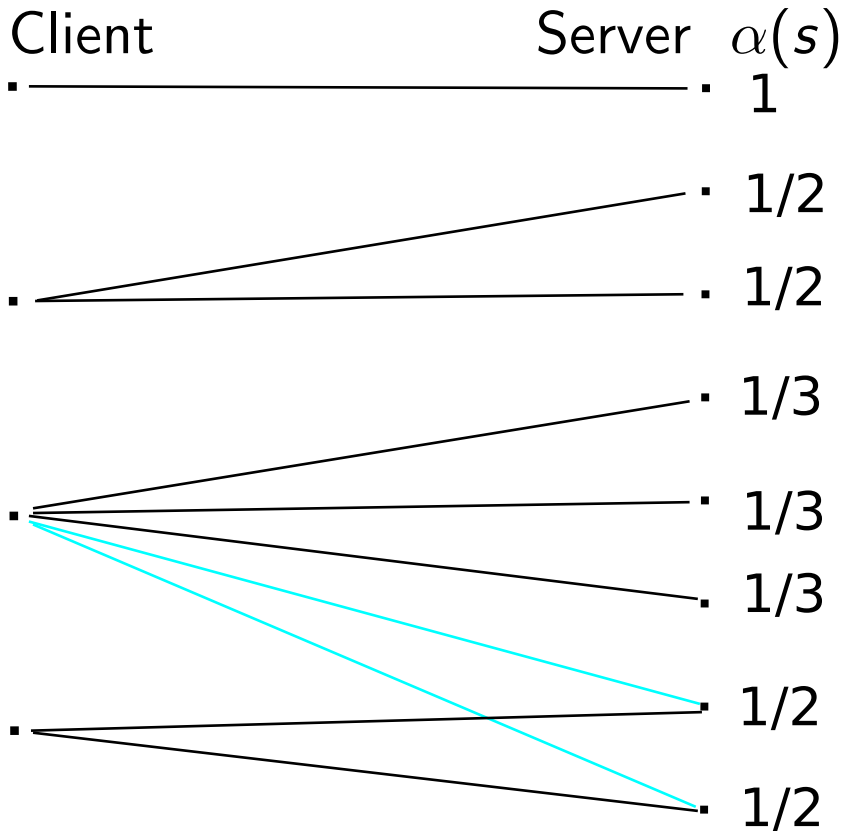
**Uniqueness Lemma**

All balanced flows have same $\alpha(s)$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

# The necessity $\alpha(s)$ of a server $s$.




**Definition (Balanced flow)**

Minimising $\sum_{s \in S}(\alpha(s))^2$

**Uniqueness Lemma**

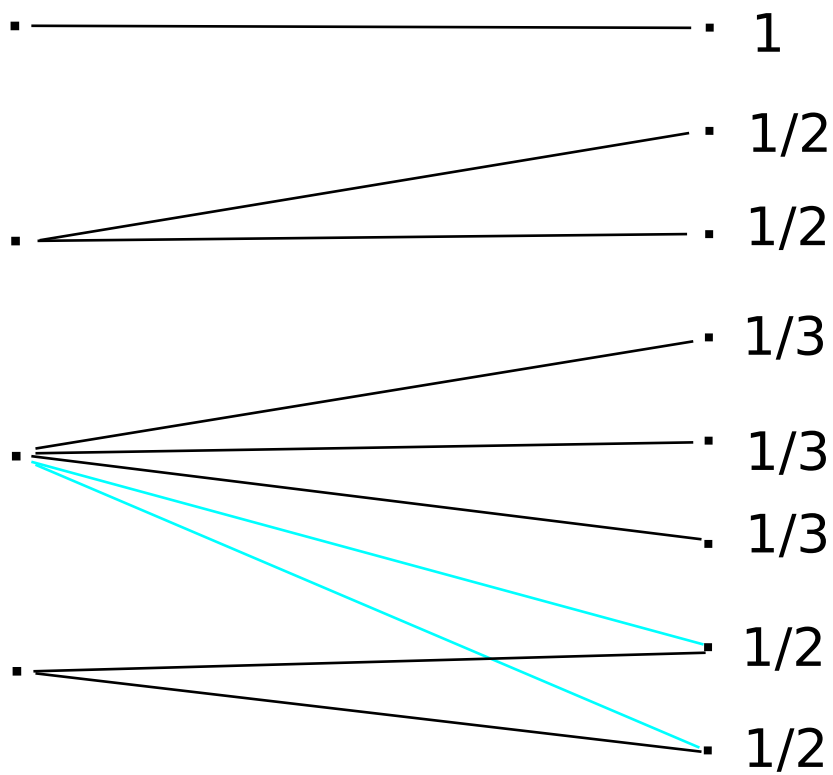
All balanced flows have same $\alpha(s)$

**Define a *flow* from each client to its neighbouring servers**

- Each client has out-flow 1
- the flow is *balanced*

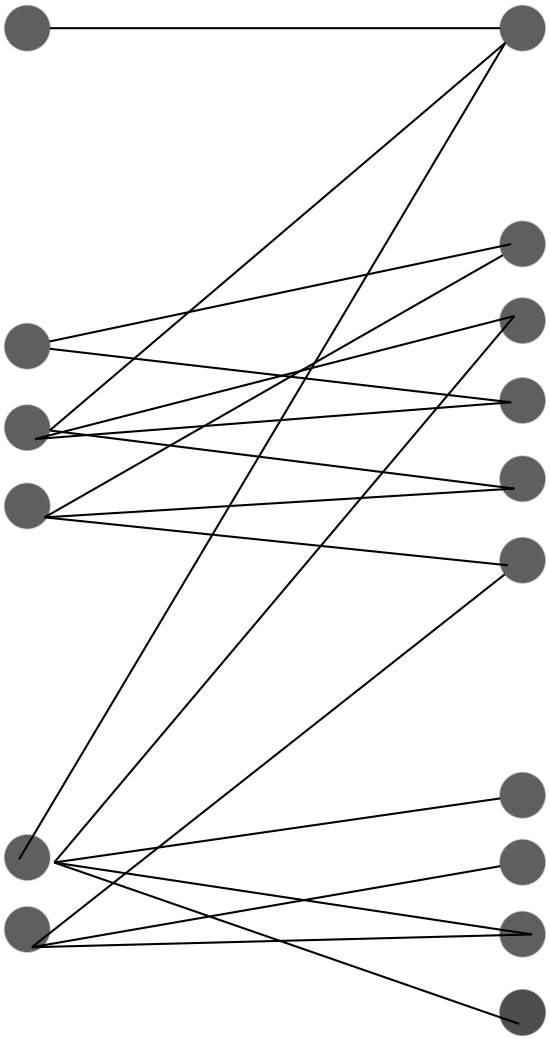
Observation: the necessities are independent of the current matching.

# Uniqueness $\rightarrow$ Structure?



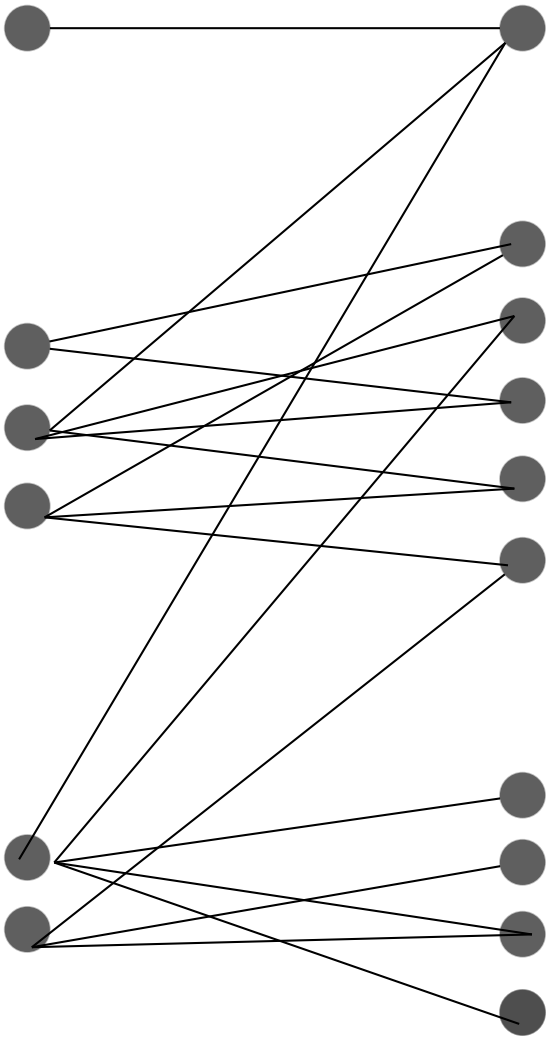

Similar to graph decomposition from Goel, Kapralov, Khanna SODA 2012.

# Unique Server Necessities: combinatorial proof sketch



Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

# Unique Server Necessities: combinatorial proof sketch



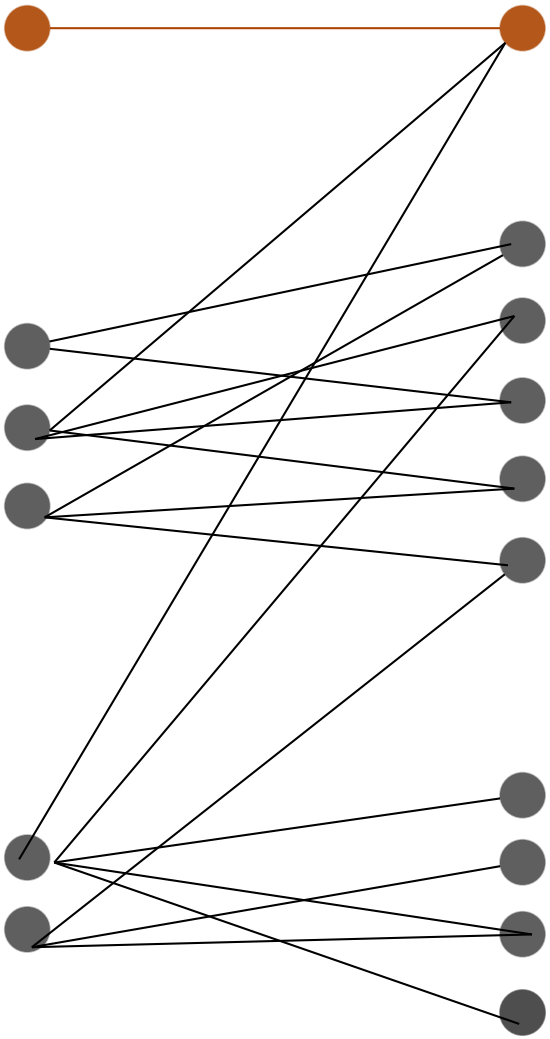
Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

- That is, $|K| \leq |N(K)|$

# Unique Server Necessities: combinatorial proof sketch



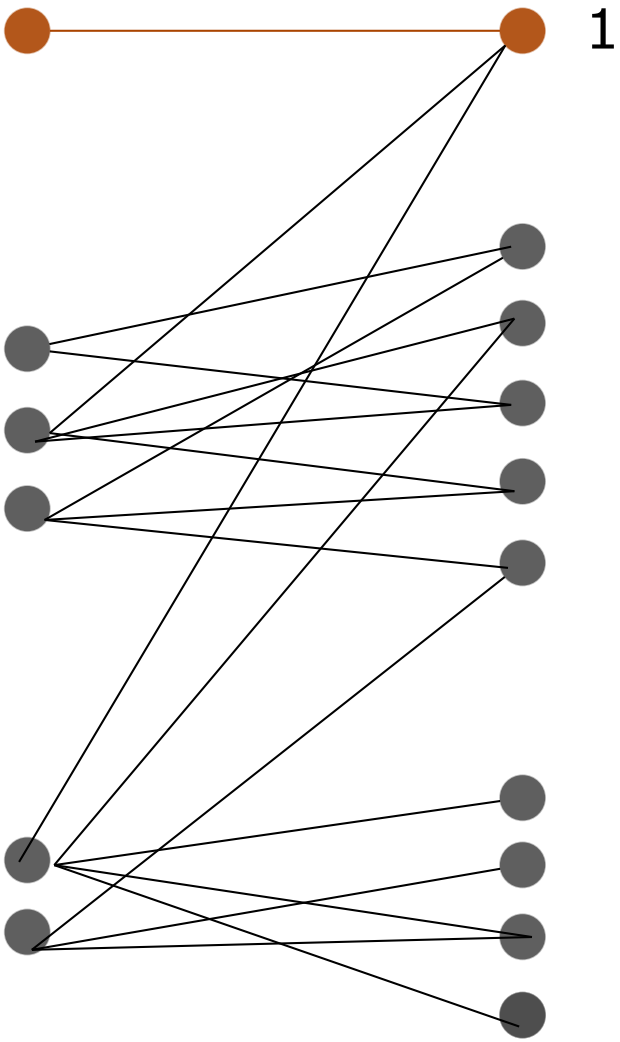
Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

- That is, $|K| \leq |N(K)|$

# Unique Server Necessities: combinatorial proof sketch



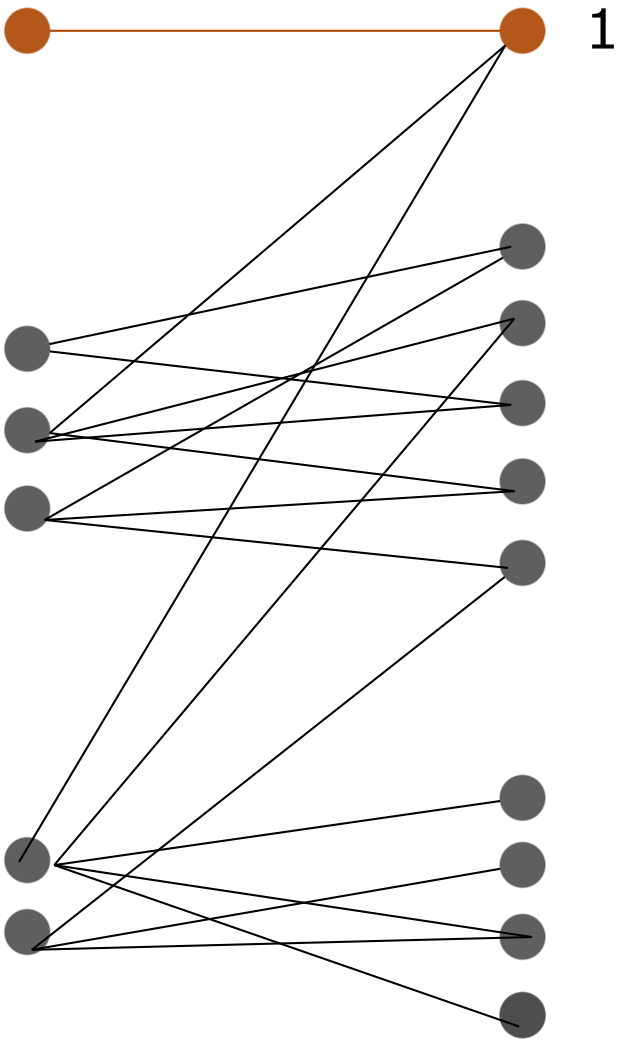

Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

- That is, $|K| \leq |N(K)|$

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

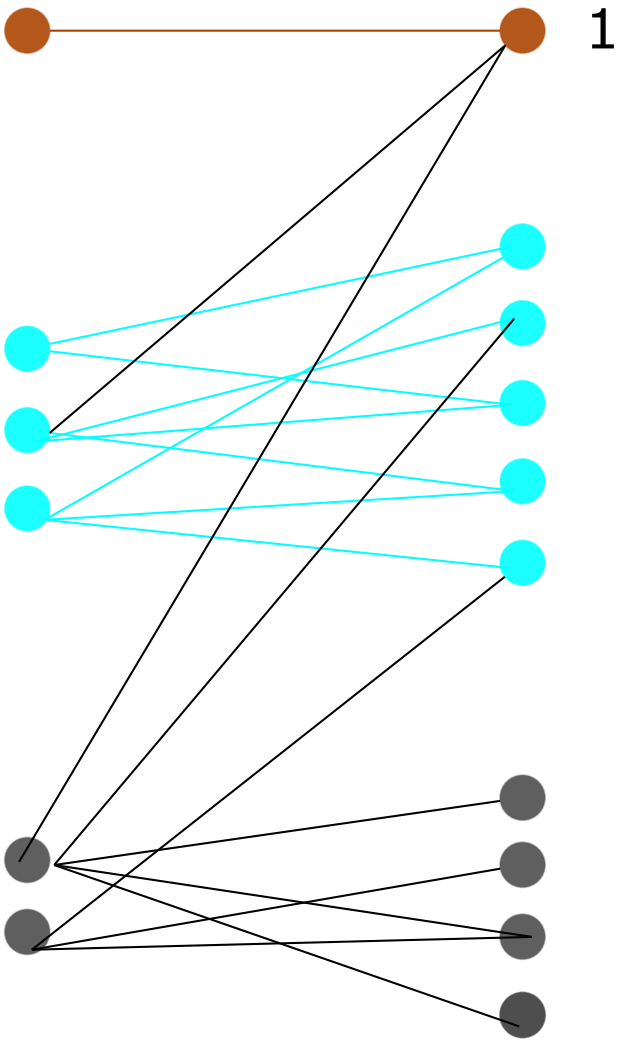
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

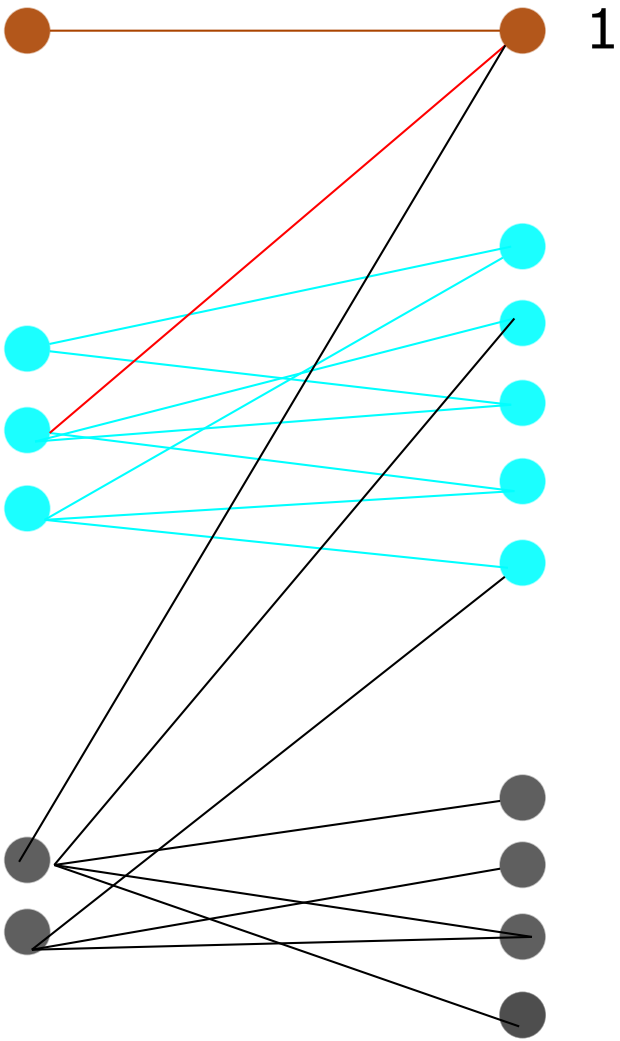
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

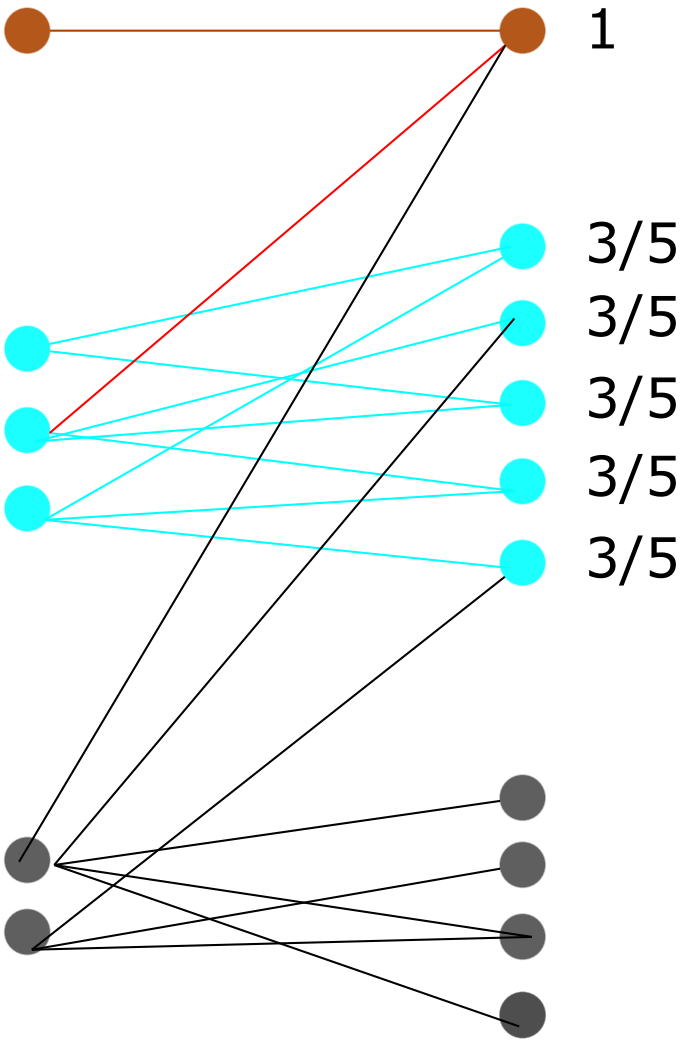
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

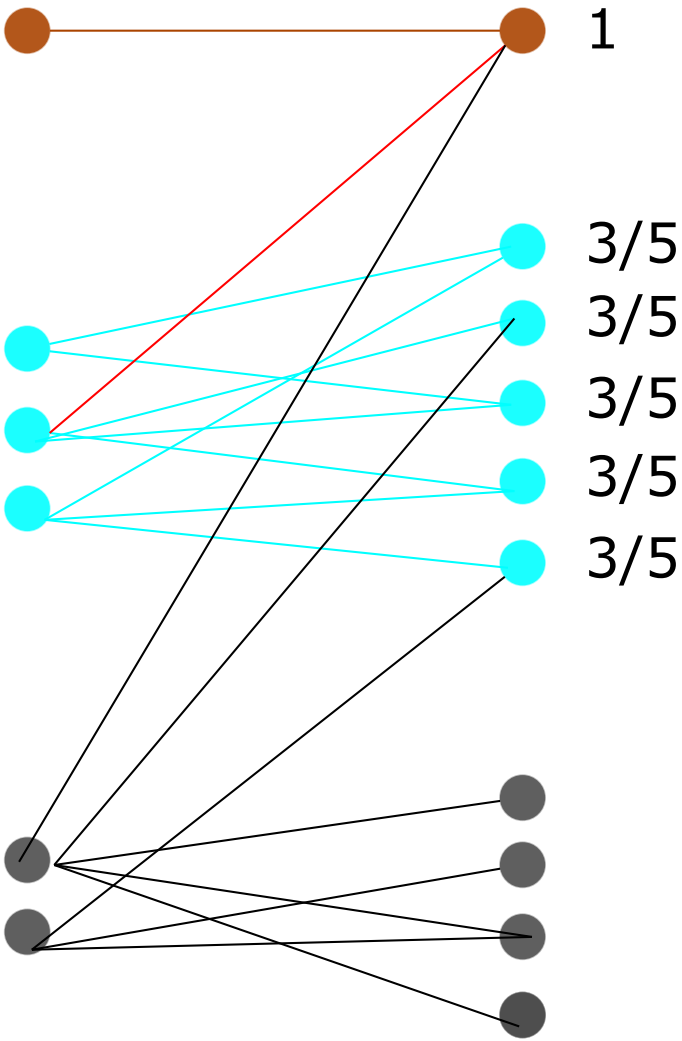
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

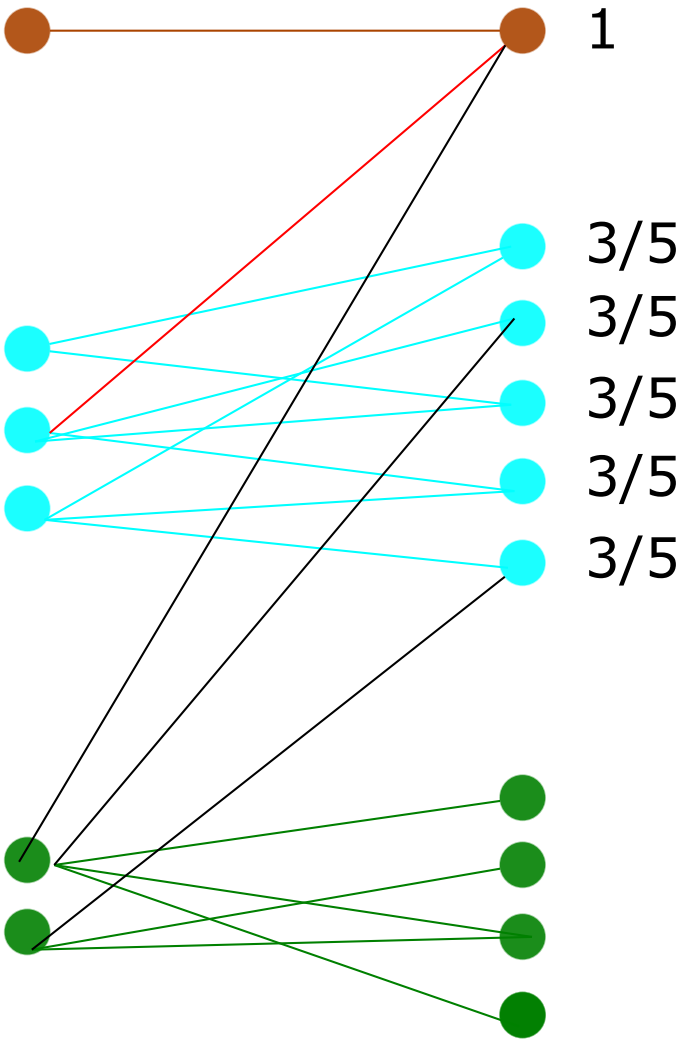
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

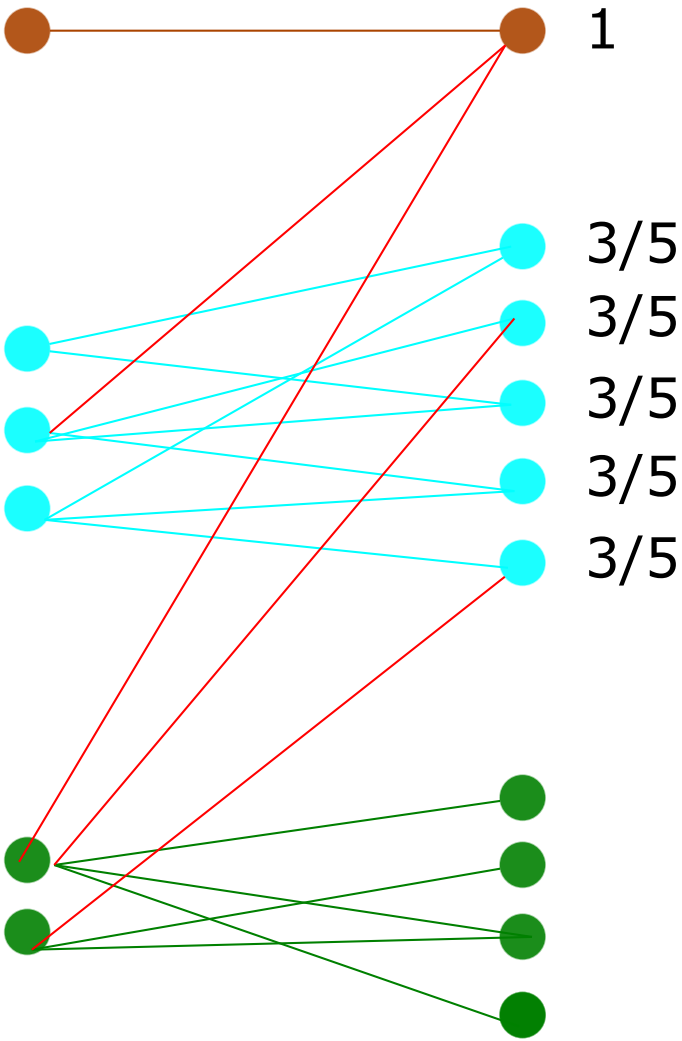
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

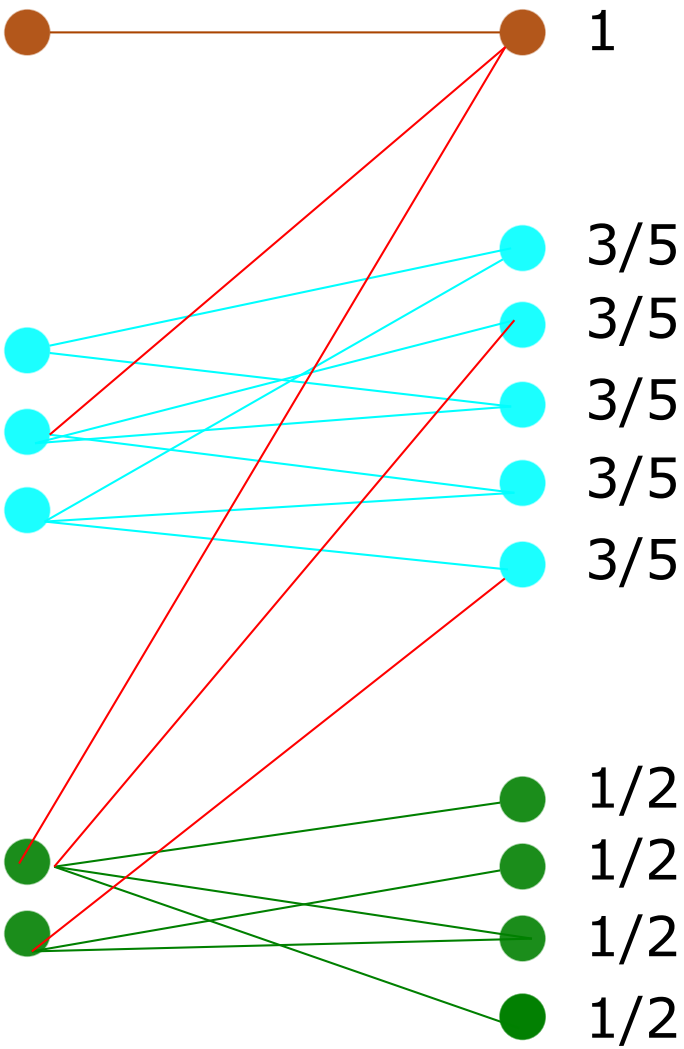
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

# Unique Server Necessities: combinatorial proof sketch




Server Necessity of $s$: how much do we need $s$ to ensure that every client can be matched.

## Hall's Condition

All clients can be matched if every subset $K \subseteq C$ has expansion at least 1

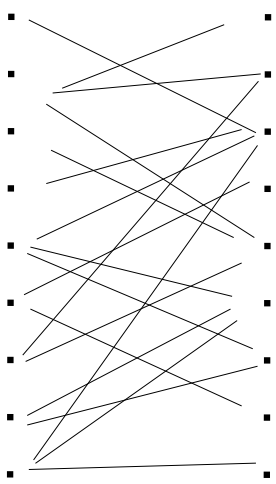
- That is, $|K| \leq |N(K)|$

## Our Decomposition

Successively Remove least expansive subset

- i.e. subset $K \subseteq C$ that minimizes $\frac{|N(K)|}{|K|}$.
- More expansion $\rightarrow$ lower necessity

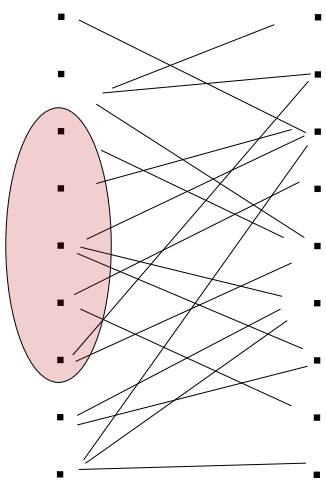
# Necessities and Hall's Theorem



## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

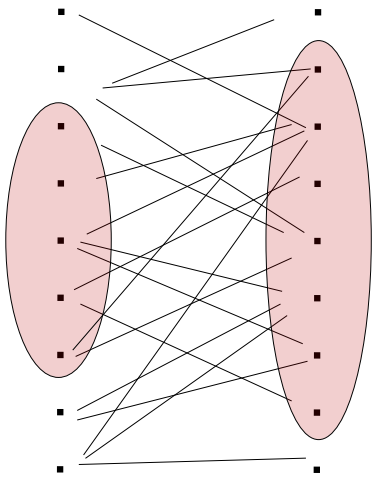
# Necessities and Hall's Theorem



## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

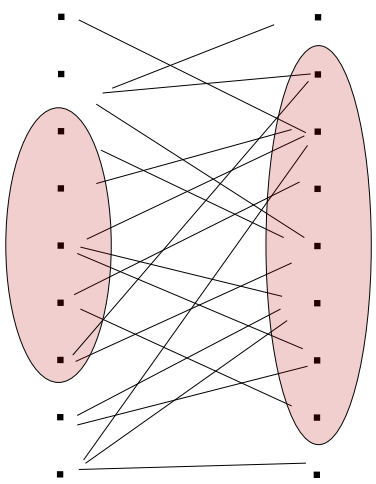
# Necessities and Hall's Theorem



## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

# Necessities and Hall's Theorem



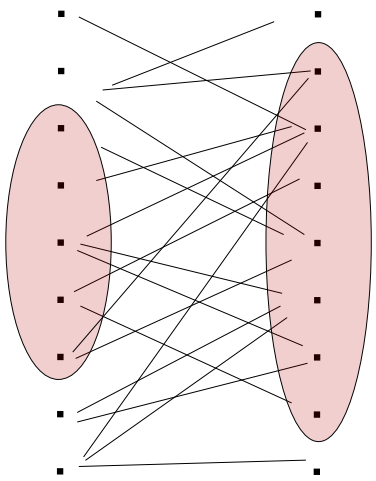
$\alpha(s) \leq 1$

## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

- *Equivalently: can match all clients iff if $\alpha(s) \leq 1 \quad \forall s \in S$*

# Necessities and Hall's Theorem



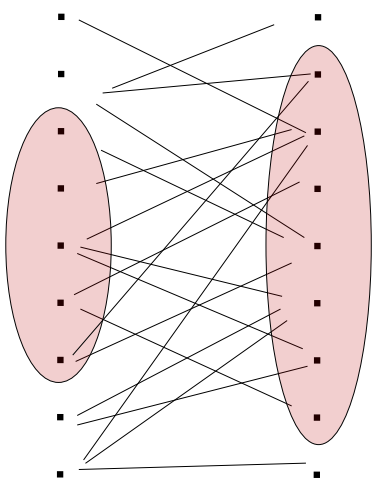
$$\alpha(s) \leq 3/4 \implies \frac{|K|}{|N(K)|} \leq 3/4$$

## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

- *Equivalently: can match all clients iff if $\alpha(s) \leq 1 \quad \forall s \in S$*

# Necessities and Hall's Theorem



$$\alpha(s) \leq 3/4 \implies \frac{|K|}{|N(K)|} \leq 3/4$$

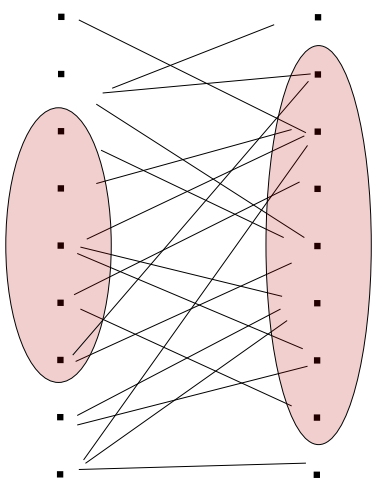
Expansion factor: $4/3$

## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

- *Equivalently: can match all clients iff if $\alpha(s) \leq 1 \quad \forall s \in S$*

# Necessities and Hall's Theorem



$$\alpha(s) \leq 3/4 \implies \frac{|K|}{|N(K)|} \leq 3/4$$

Expansion factor: $4/3$

## Theorem (Hall's Theorem)

*Can match all clients iff for every $K \subseteq C$, the neighbourhood $N(K)$ has size $|N(K)| \geq |K|$. In other words, $\frac{|K|}{|N(K)|} \leq 1$.*

- *Equivalently: can match all clients iff if $\alpha(s) \leq 1 \quad \forall s \in S$*

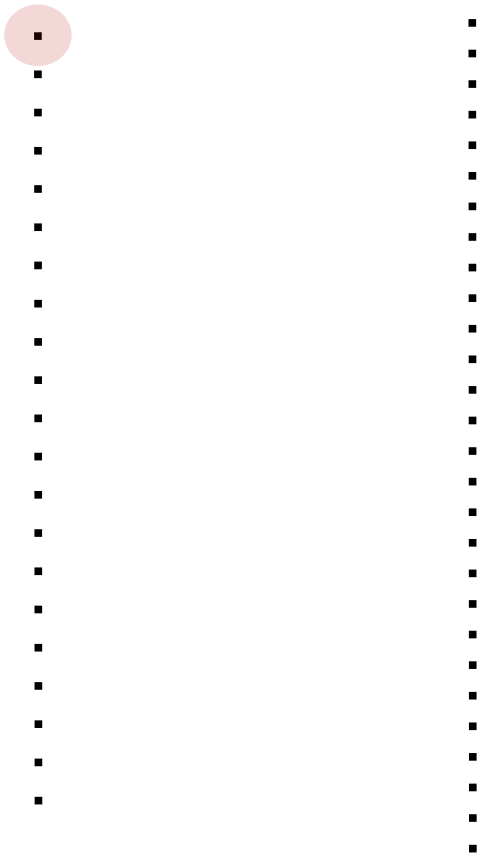
## Lemma

*Say that for $K \subset C$, $\alpha(s) \leq \alpha_K^{max}$ for all $s \in N(K)$. Then $\frac{|K|}{|N(K)|} \leq \alpha_K^{max}$*

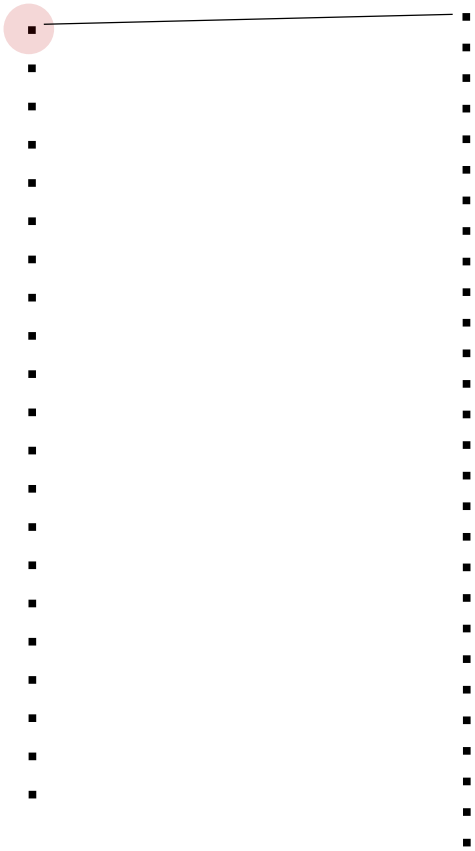
- *Expansion Factor:* $1/\alpha_K^{max}$

# Outline

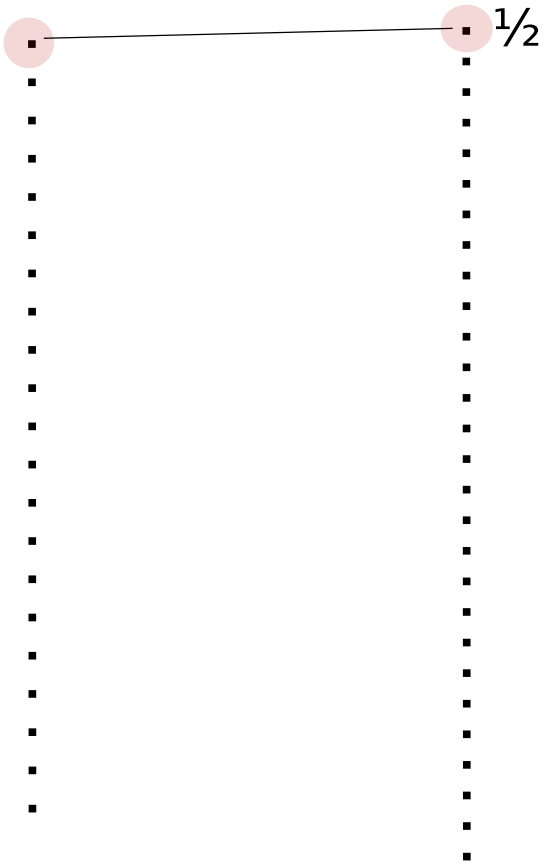
# Expansion Lemma



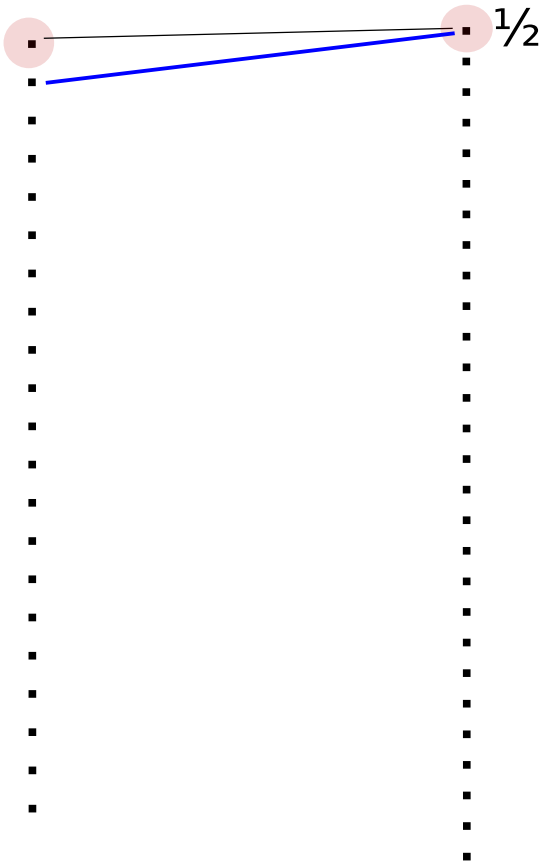
# Expansion Lemma



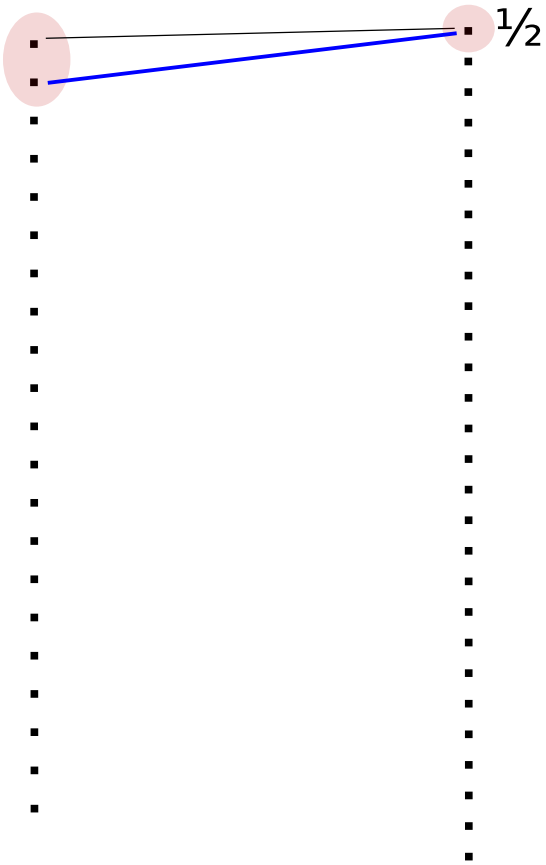
# Expansion Lemma



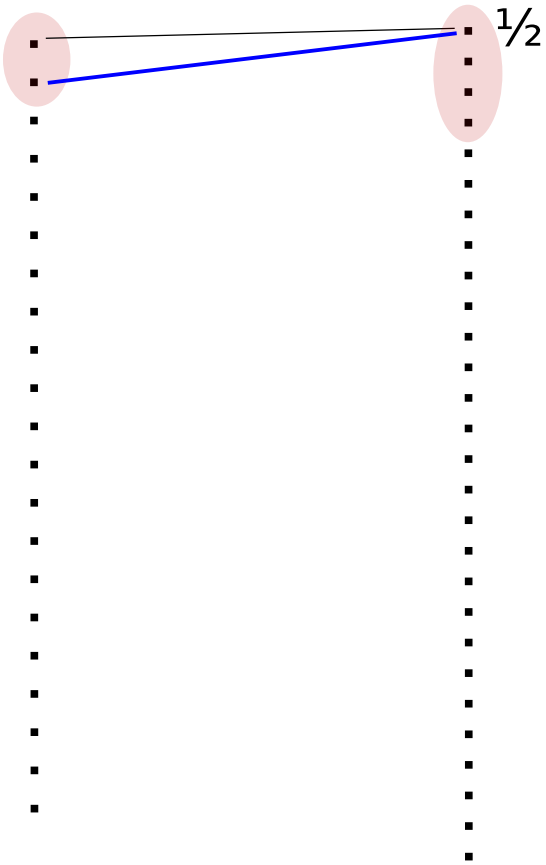
# Expansion Lemma



# Expansion Lemma



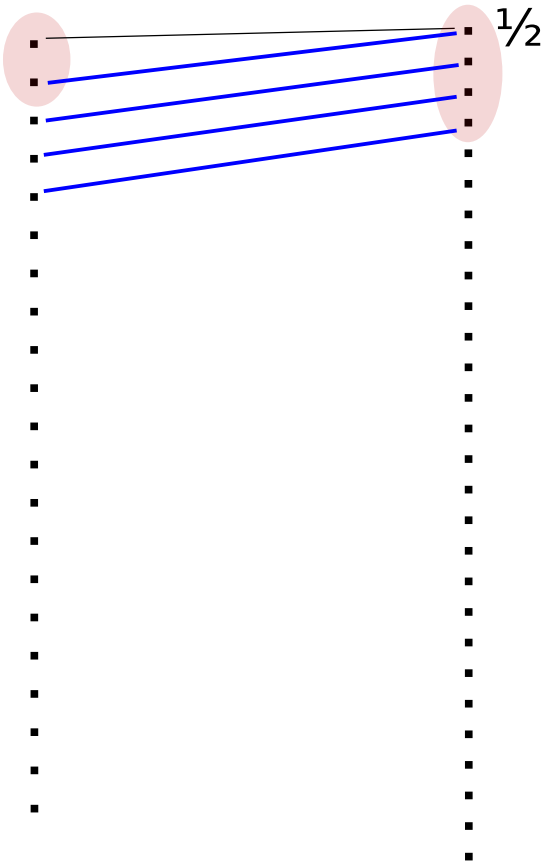
# Expansion Lemma




## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

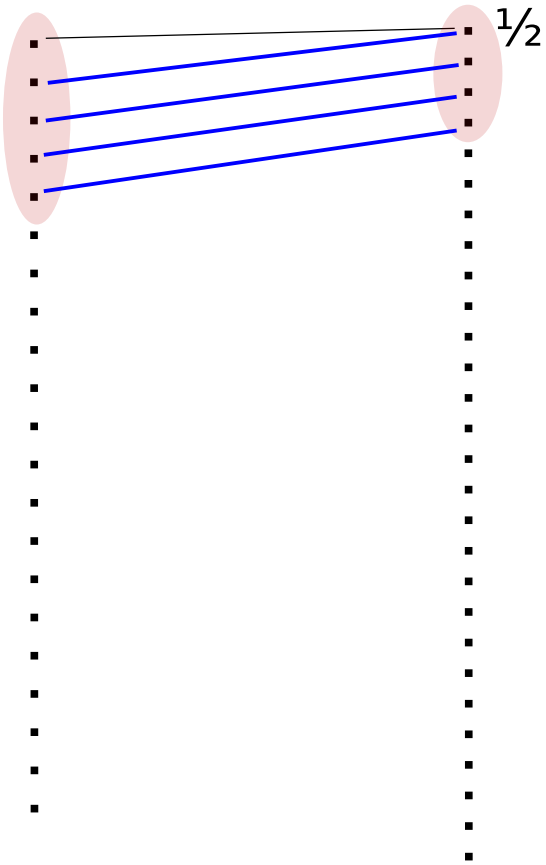
# Expansion Lemma




## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

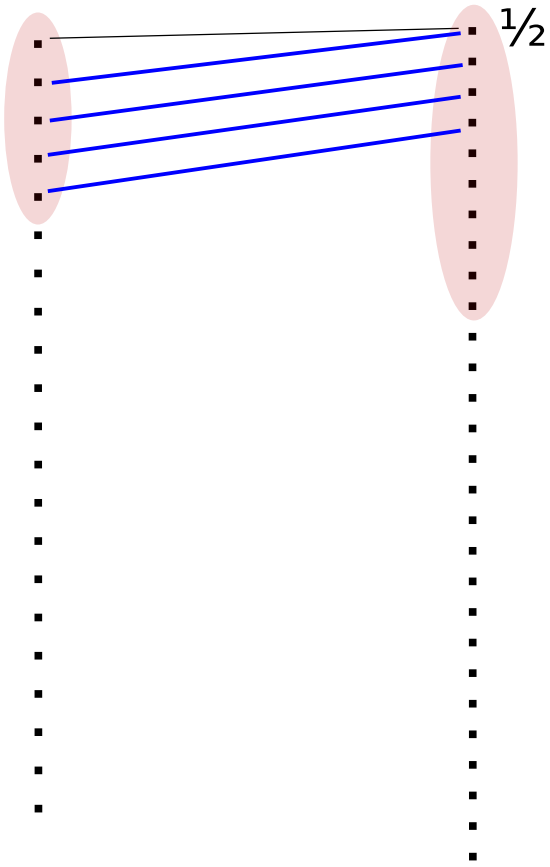
# Expansion Lemma




## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

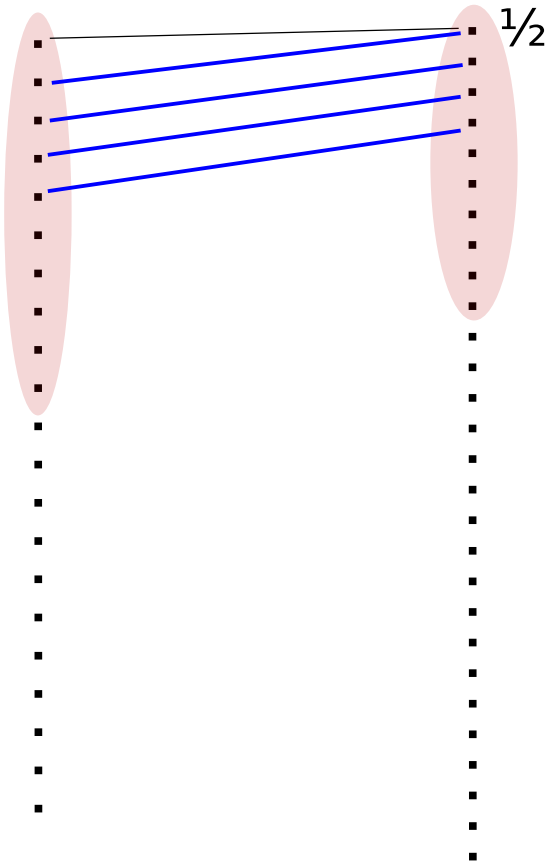
# Expansion Lemma




## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

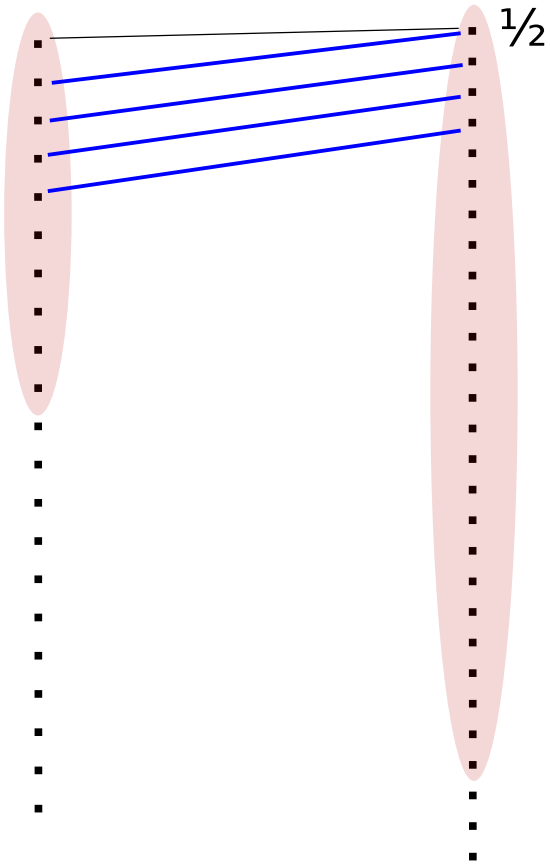
# Expansion Lemma




## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

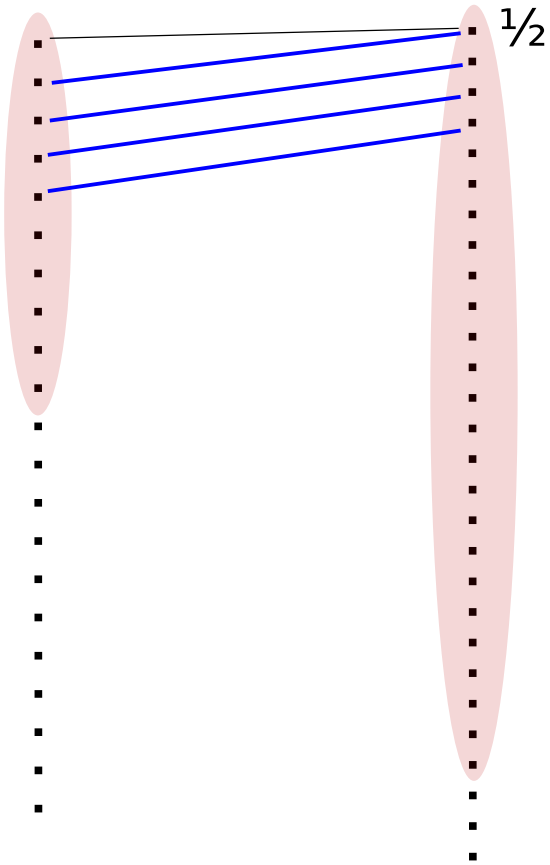
# Expansion Lemma




## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

# Expansion Lemma




## Expansion Lemma

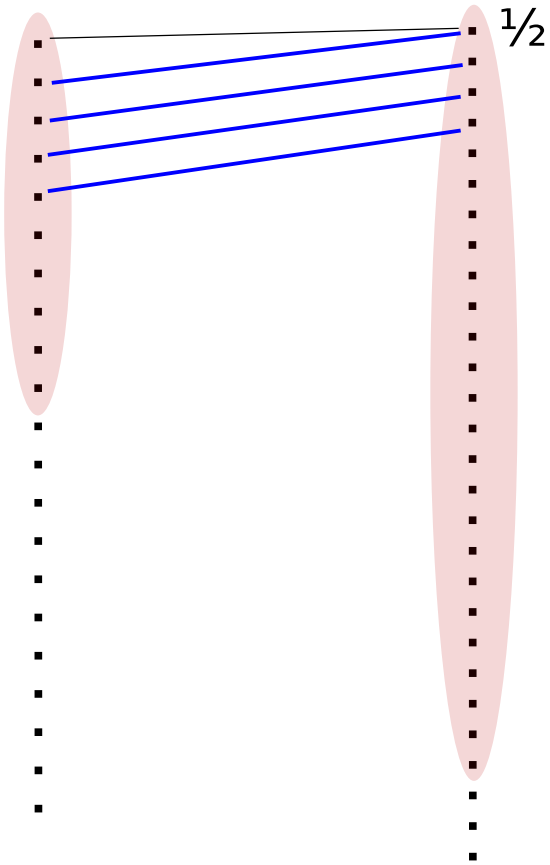
Server $s$ has $\alpha(s) = 1 - \epsilon$

$\Longrightarrow$ there is an augmenting path through $s$ of length $O(\frac{\log n}{\epsilon})$

## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

# Expansion Lemma




## Expansion Lemma

Server $s$ has $\alpha(s) = 1 - \epsilon$
$\Longrightarrow$ there is an augmenting path through $s$ of length $O(\frac{\log n}{\epsilon})$

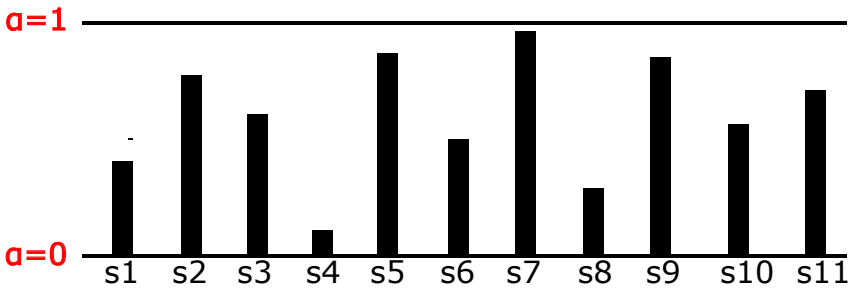
## Proof.

Expansion factor of $\frac{1}{1-\epsilon} \sim 1 + \epsilon \Longrightarrow$
path has length $O(\log_{1+\epsilon} n) = O(\frac{\log n}{\epsilon})$ □

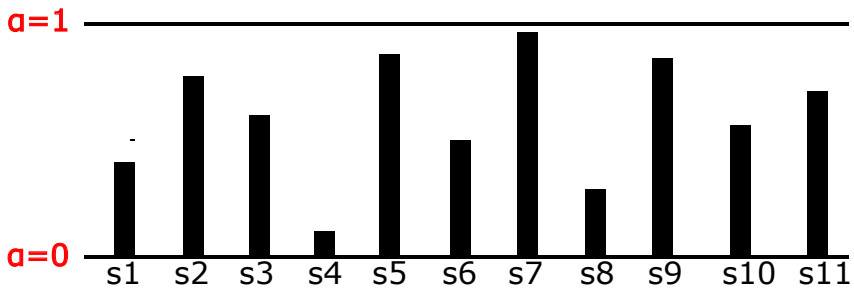
## Expansion Properties

- **Expansion Factor:** $\frac{1}{\alpha^{max}} = 2$
- **Balanced Flow:** $\alpha^{max}$ never increases as we search for the augmenting path

# Putting it All Together



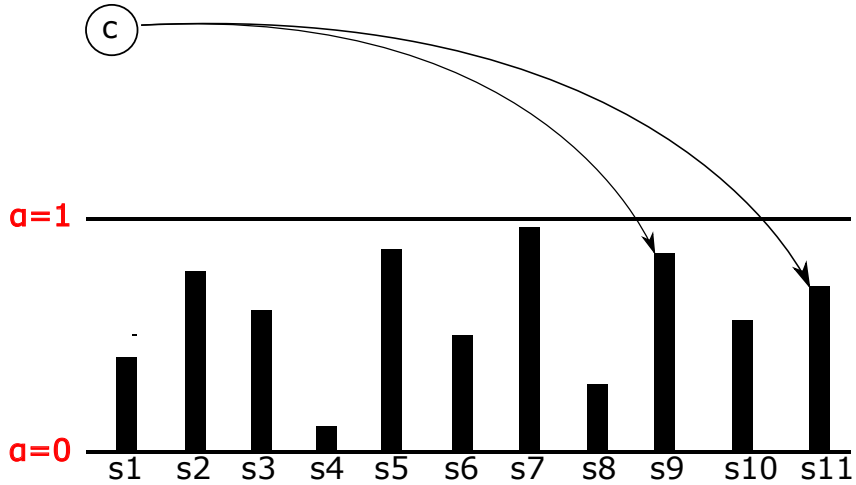
# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing

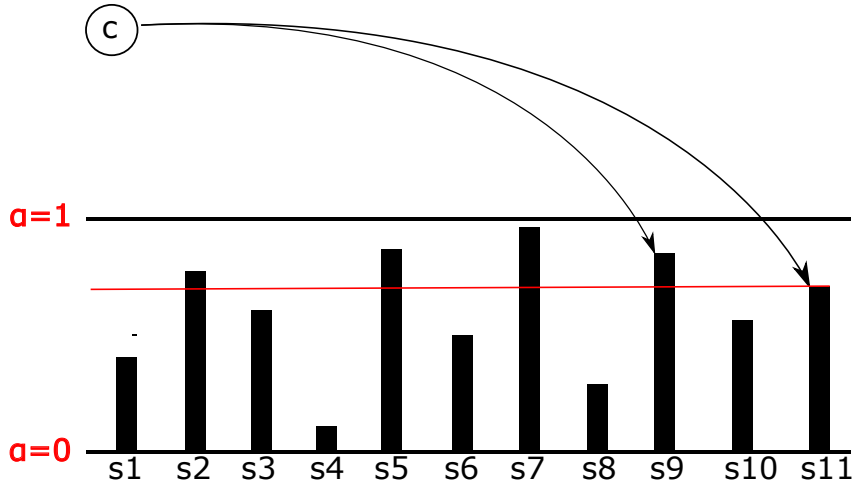
# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing

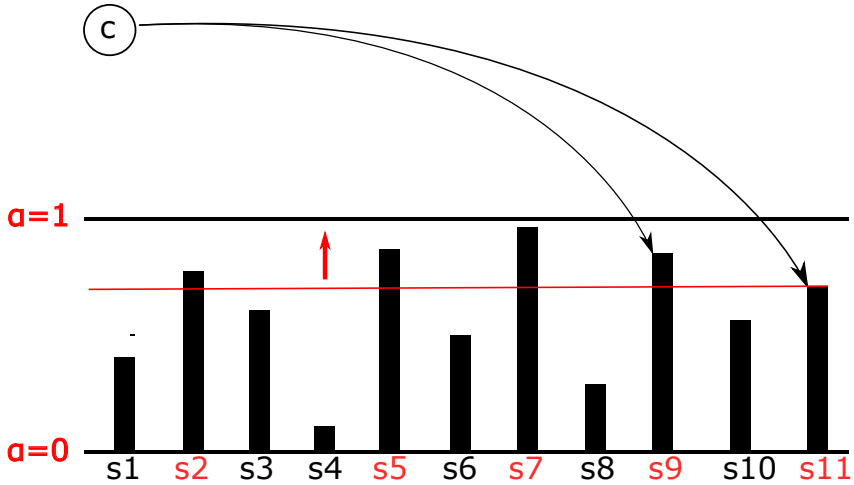
# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

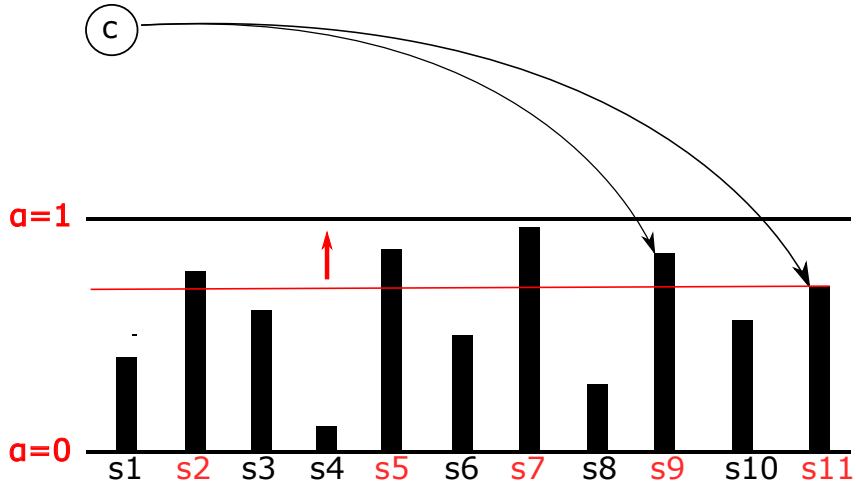
# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

# Putting it All Together



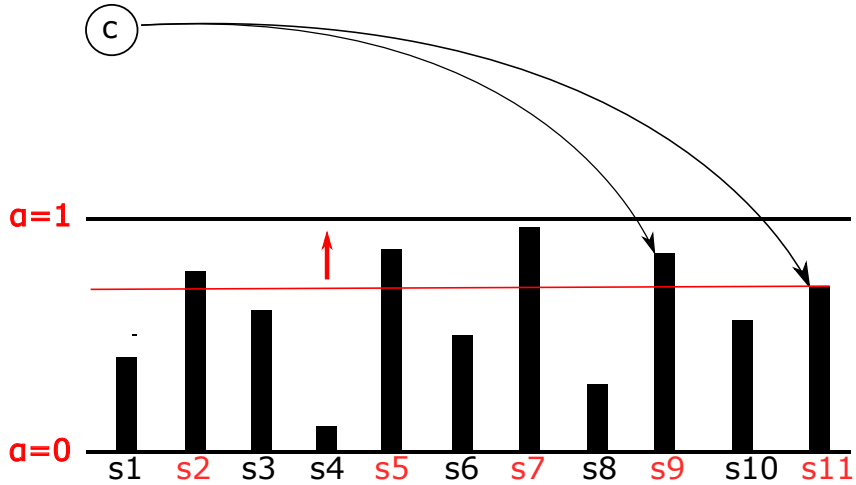

## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

## Expansion Lemma

$\alpha(s) = 1 - \epsilon \rightarrow$ path length $O(\frac{\log n}{\epsilon})$

# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

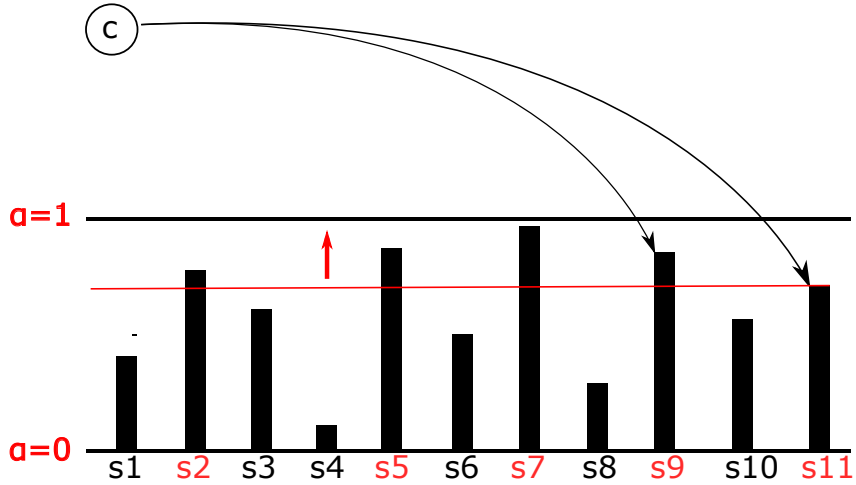
## Expansion Lemma

$\alpha(s) = 1 - \epsilon \rightarrow$ path length $O(\frac{\log n}{\epsilon})$

## Example: $\alpha(s11) \sim \frac{9}{10}$

- Expansion Lemma: path length $= O(\log n/10) = O(\log(n))$
- at most $n$ augmenting paths in total (trivial)
- total number of changes: $O(n \log n)$

# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

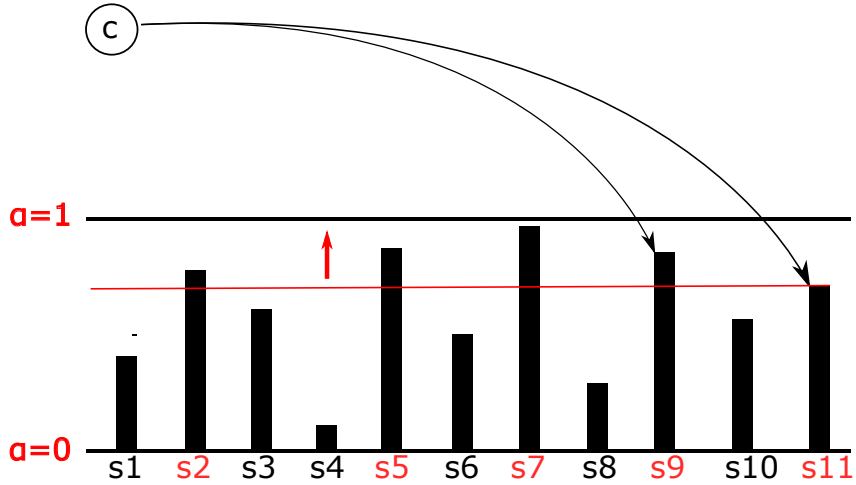
## Expansion Lemma

$\alpha(s) = 1 - \epsilon \rightarrow$ path length $O(\frac{\log n}{\epsilon})$

## Example: $\alpha(s11) \sim 1 - \frac{1}{n^{1/3}}$

- augmenting path length $= O(n^{1/3} \log n)$
- all new flow goes to servers with $\alpha(s) \geq 1 - \frac{1}{n^{1/3}}$

# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

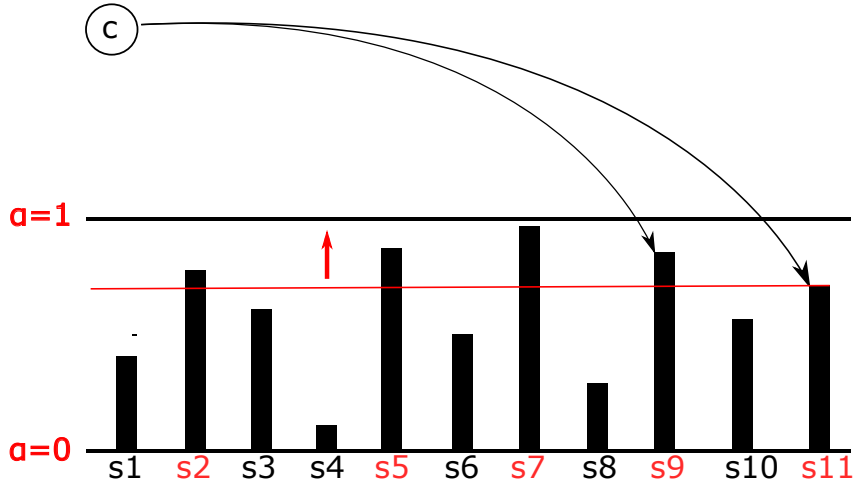
## Expansion Lemma

$\alpha(s) = 1 - \epsilon \rightarrow$ path length $O(\frac{\log n}{\epsilon})$

## Example: $\alpha(s11) \sim 1 - \frac{1}{n^{1/3}}$

- augmenting path length $= O(n^{1/3} \log n)$
- all new flow goes to servers with $\alpha(s) \geq 1 - \frac{1}{n^{1/3}}$
- 1 unit of flow from client $\rightarrow n^{1/3}$ servers "saturated"

# Putting it All Together




## $\alpha(s)$ changes over time

- each client adds 1 to $\sum \alpha(s)$
- $\alpha(s)$ non-decreasing
- Flow Only Goes Upwards

## Expansion Lemma

$\alpha(s) = 1 - \epsilon \rightarrow$ path length $O(\frac{\log n}{\epsilon})$

## Example: $\alpha(s11) \sim 1 - \frac{1}{n^{1/3}}$

- augmenting path length $= O(n^{1/3} \log n)$
- all new flow goes to servers with $\alpha(s) \geq 1 - \frac{1}{n^{1/3}}$
- 1 unit of flow from client $\rightarrow n^{1/3}$ servers "saturated"
- at most $n^{2/3}$ such paths $\rightarrow (n^{2/3})(n^{1/3} \log n) = O(n \log(n))$ changes

# Outline

Lower Bound: $\Omega(n \log n)$ total number of server changes (amortized $\log n$).

## Theorem (number of changes)

*SAP makes at most $O(n \log^2 n)$ replacements in total (amortized $\log^2 n$).*

- *Previous: SAP makes $O(n^2)$ replacements (trivial)*
- *Previous: SAP makes $O(n \log n)$ replacements in special cases*
- *Previous: non-SAP protocol with $O(n^{1.5})$ replacements*

Lower Bound: $\Omega(n \log n)$ total number of server changes (amortized $\log n$).

## Theorem (number of changes)

*SAP makes at most $O(n \log^2 n)$ replacements in total (amortized $\log^2 n$).*

- *Previous: SAP makes $O(n^2)$ replacements (trivial)*
- *Previous: SAP makes $O(n \log n)$ replacements in special cases*
- *Previous: non-SAP protocol with $O(n^{1.5})$ replacements*

## Theorem (time to compute the changes)

*There is an implementation of SAP with $O(m\sqrt{n}\sqrt{\log n})$ total time.*

- *Previous: $O(m\sqrt{n})$ but with $n^{1.5}$ replacements (not SAP)*

Lower Bound: $\Omega(n \log n)$ total number of server changes (amortized $\log n$).

## Theorem (number of changes)

*SAP makes at most $O(n \log^2 n)$ replacements in total (amortized $\log^2 n$).*

- *Previous: SAP makes $O(n^2)$ replacements (trivial)*
- *Previous: SAP makes $O(n \log n)$ replacements in special cases*
- *Previous: non-SAP protocol with $O(n^{1.5})$ replacements*

## Theorem (time to compute the changes)

*There is an implementation of SAP with $O(m\sqrt{n}\sqrt{\log n})$ total time.*

- *Previous: $O(m\sqrt{n})$ but with $n^{1.5}$ replacements (not SAP)*

## Theorem: Minimizing Max Server Load

- Let $L$ be maximum server load in final graph
- Lower Bound: $\Omega(\min\{nL, n^{1.5}\})$ server changes
- Upper Bound: $O(\log^2 n \min\{nL, n^{1.5}\})$ server changes

# Open Problems

Lower Bound: $\Omega(n \log n)$ total number of server changes (amortized $\log n$).

## Conjecture

*SAP makes at most* $O(n \log n)$ *server changes in total.*

## Conjecture

*There is an implementation of SAP that runs in* $O(m\sqrt{n})$ *total time.*