Faster MPC Algorithms for Allocation in Uniformly Sparse Graphs

Jakub Łącki
Google Research

Slobodan Mitrović
UC Davis

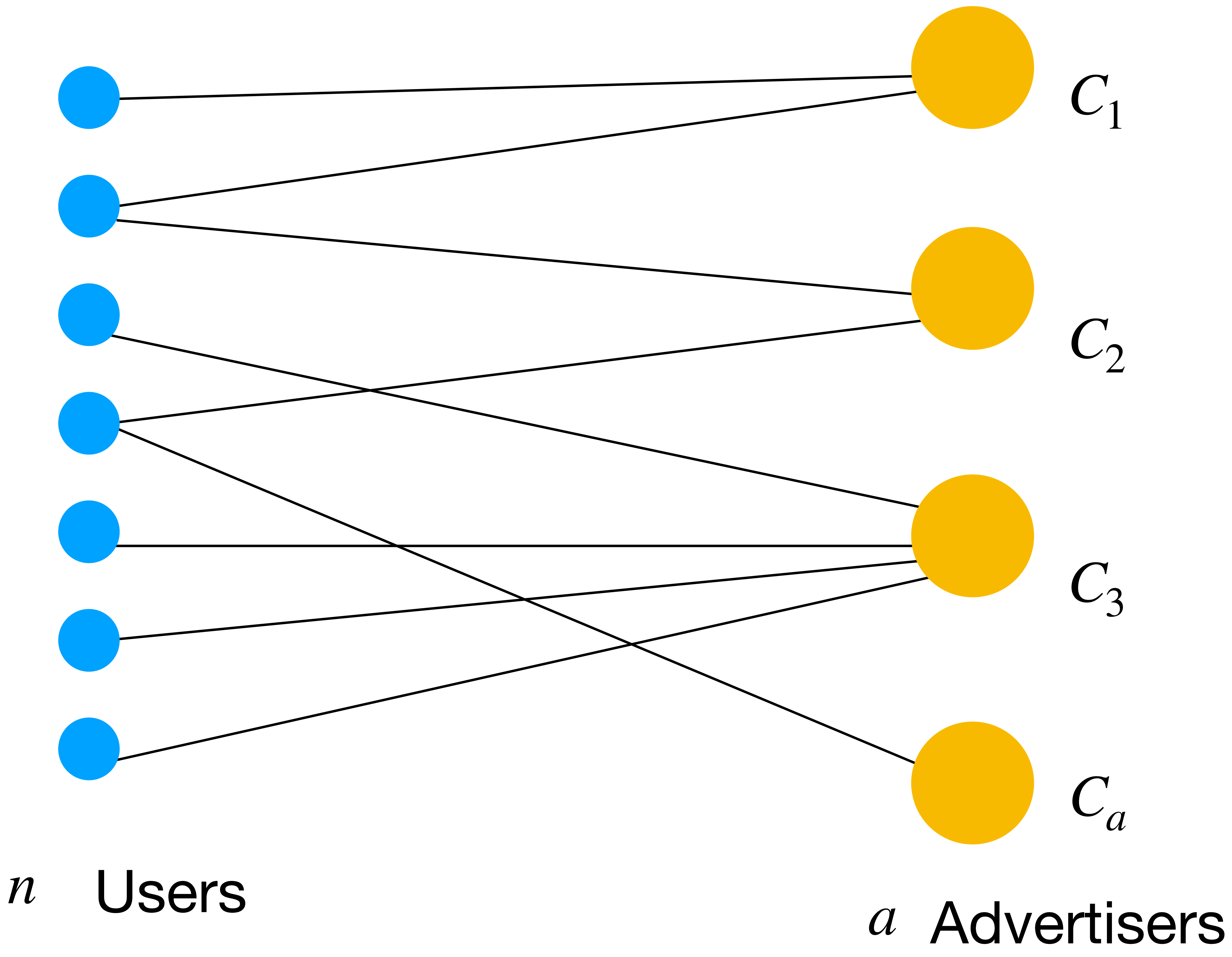
Wen-Horng Sheu
UC Davis

Srikkanth R
UC Davis

Overview of the Talk

1. Problem definition and notation
2. Overview of prior work and our results
3. A fast allocation algorithm in the LOCAL model
4. Fast implementation in sublinear MPC model

The Allocation Problem



I/p :

Bipartite graph with
node capacities $C_v \geq 1$

O/p :

Subset of edges M

Constraints:

(i) Each user has ≤ 1 edge

(i) Each ad v has $\leq C_v$ edges

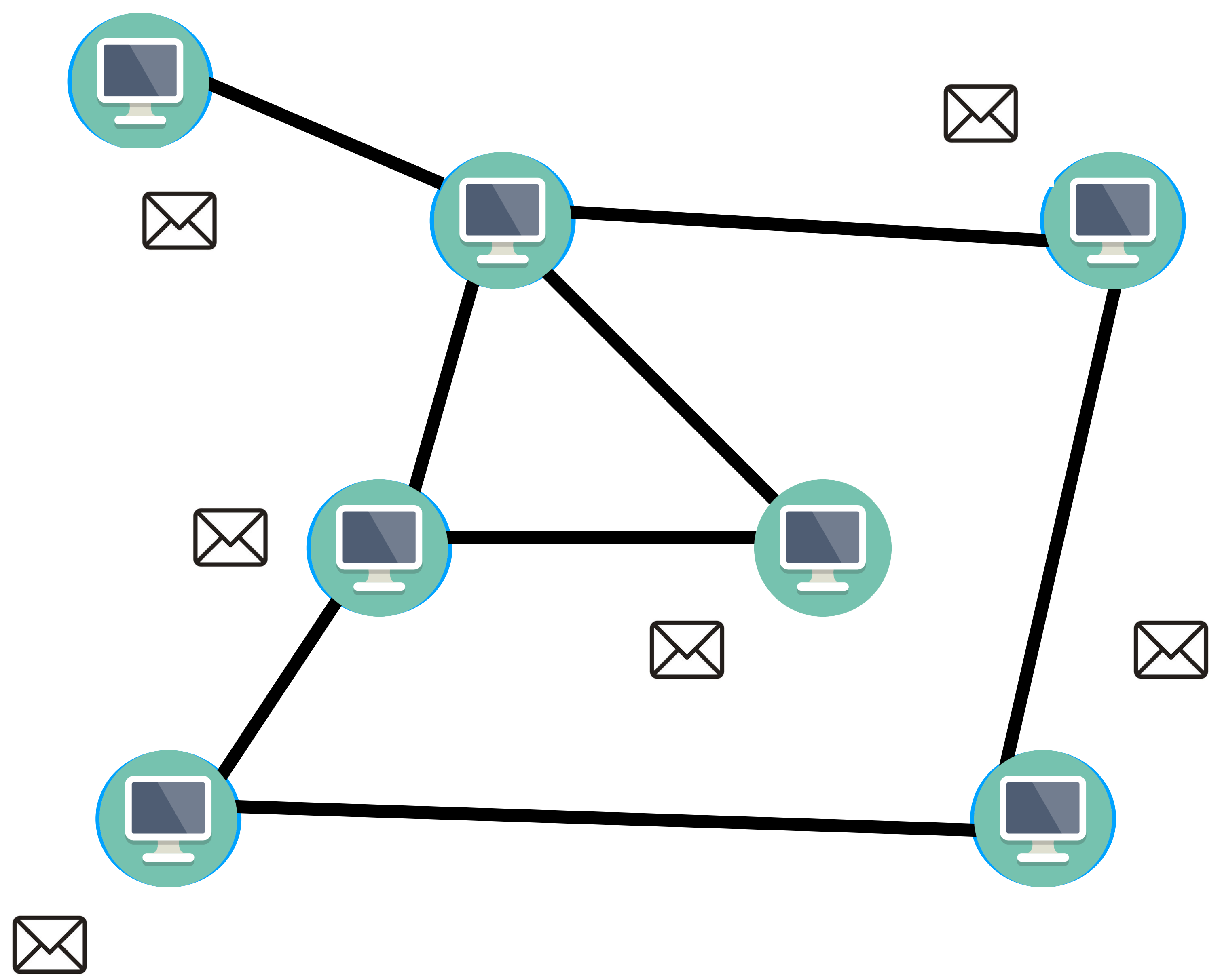
Objective:

Maximize $|M|$

Prior Work on Allocation

[Dhillon KDD ’01]	Co-clustering documents
[Mehta, Saberi, Vazirani, Vazirani JACM ’07]	Adsense problem and generalised online matching
[Agrawal, Mirrokni, Zadimoghaddam ICML ’18]	Distributed proportional allocation
	primitive
[Ahmadian, Liu, Peng, Zadimoghaddam ITCS ’22]	Distributed load balancing

LOCAL model



Nodes are computers

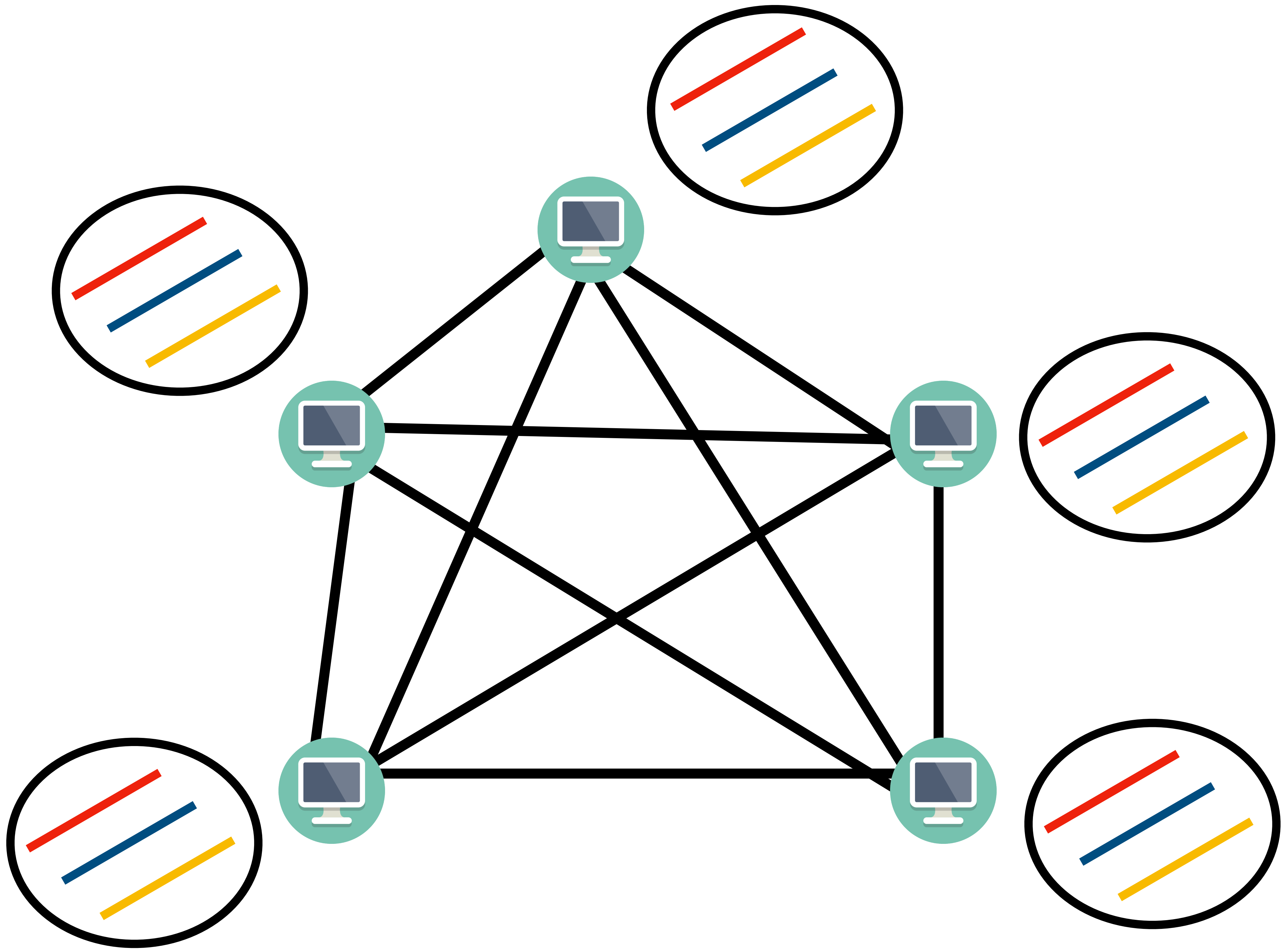
Topology of graph = communication network

Message passing

Synchronous rounds

Ideal parameters

sub-linear MPC model



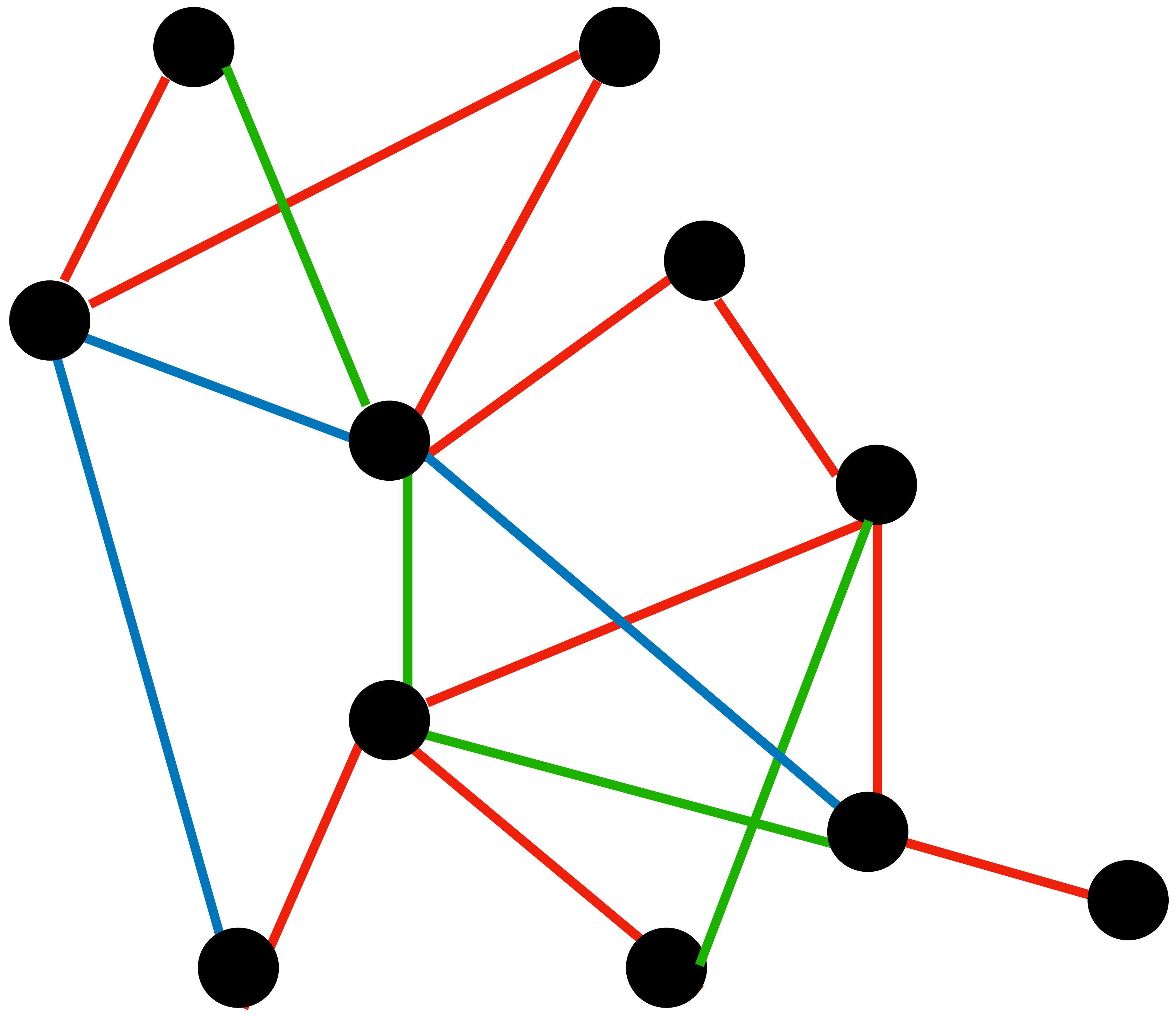
Message passing

Communication network is a clique

Nodes have limited memory $S = n^{\delta}$

Synchronous rounds

Arboricity of a graph



$\text{Arboricity}(G) = \lambda$

$\Leftrightarrow E(G)$ can be decomposed into λ forests

$\Rightarrow$ Every vertex induced subgraph has average degree at most 2λ

$\Rightarrow \Delta_{avg}/2 \leq \lambda \leq \Delta = \max_{v \in G} \deg(v)$

Prior Work on Matchings — Distributed and MPC

[Kapralov, Khanna, Sudhan SODA ’14]

$1 + \epsilon$ matching in $O_\epsilon(\log \Delta)$ LOCAL rounds

[Ghaffari and Uitto, SODA ’19]

Maximal matching in $\tilde{O}(\sqrt{\log \Delta})$ sub-linear MPC rounds

[Ghaffari, Grunau, Jin DISC ’21]

Maximal matching in $\tilde{O}(\sqrt{\log \lambda} + \log \log n)$ sub-linear MPC rounds

[Ghaffari, Grunau, Mitrovic SPAA ’22]

$1 + \epsilon$ approximate b-matching in $O(\log \log \Delta_{avg})$ near-linear MPC rounds

Our Results

Q1. Is there an efficient LOCAL algorithm for allocation that runs fast in sparse graphs?

Theorem 1

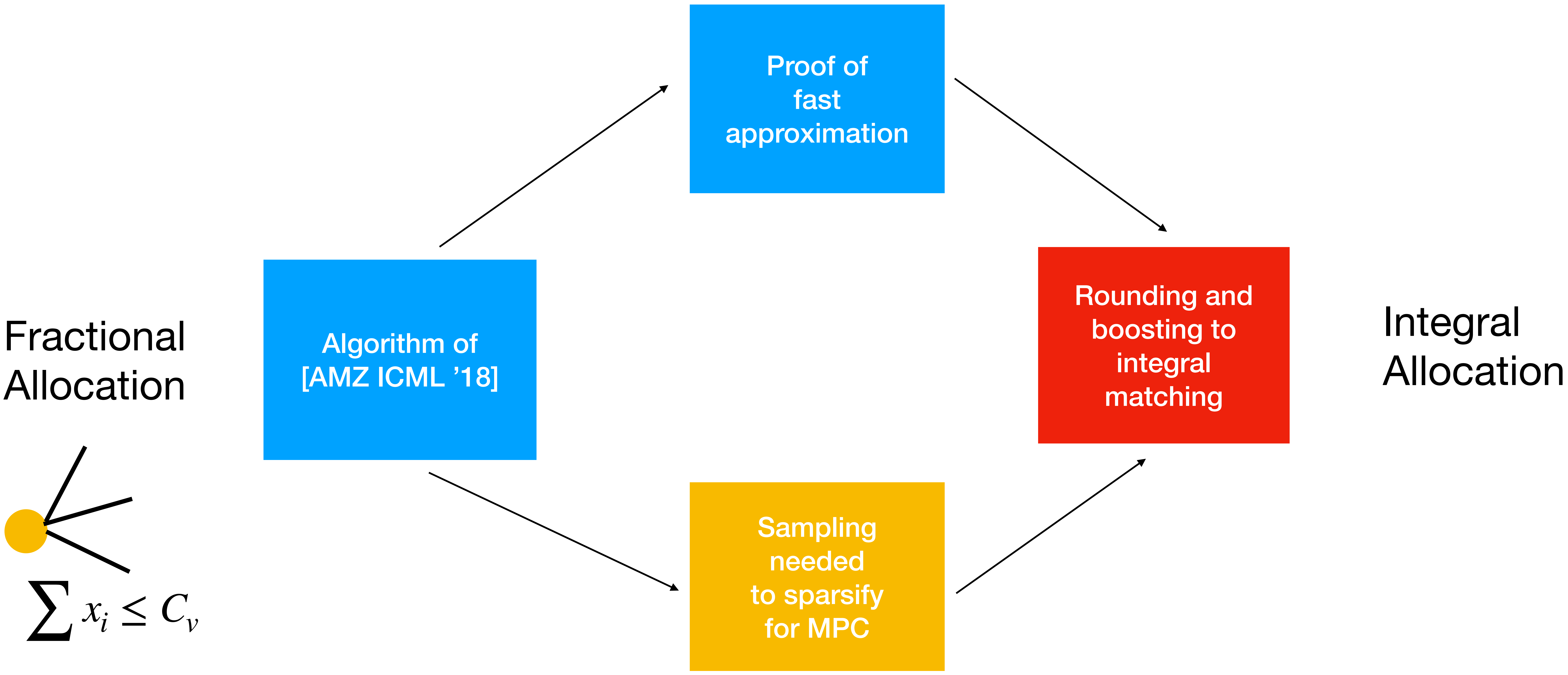
There exists a LOCAL algorithm for allocation that runs in $O_\epsilon(\log \lambda)$ rounds

Q2. Can such algorithms be implemented efficiently in sub-linear MPC?

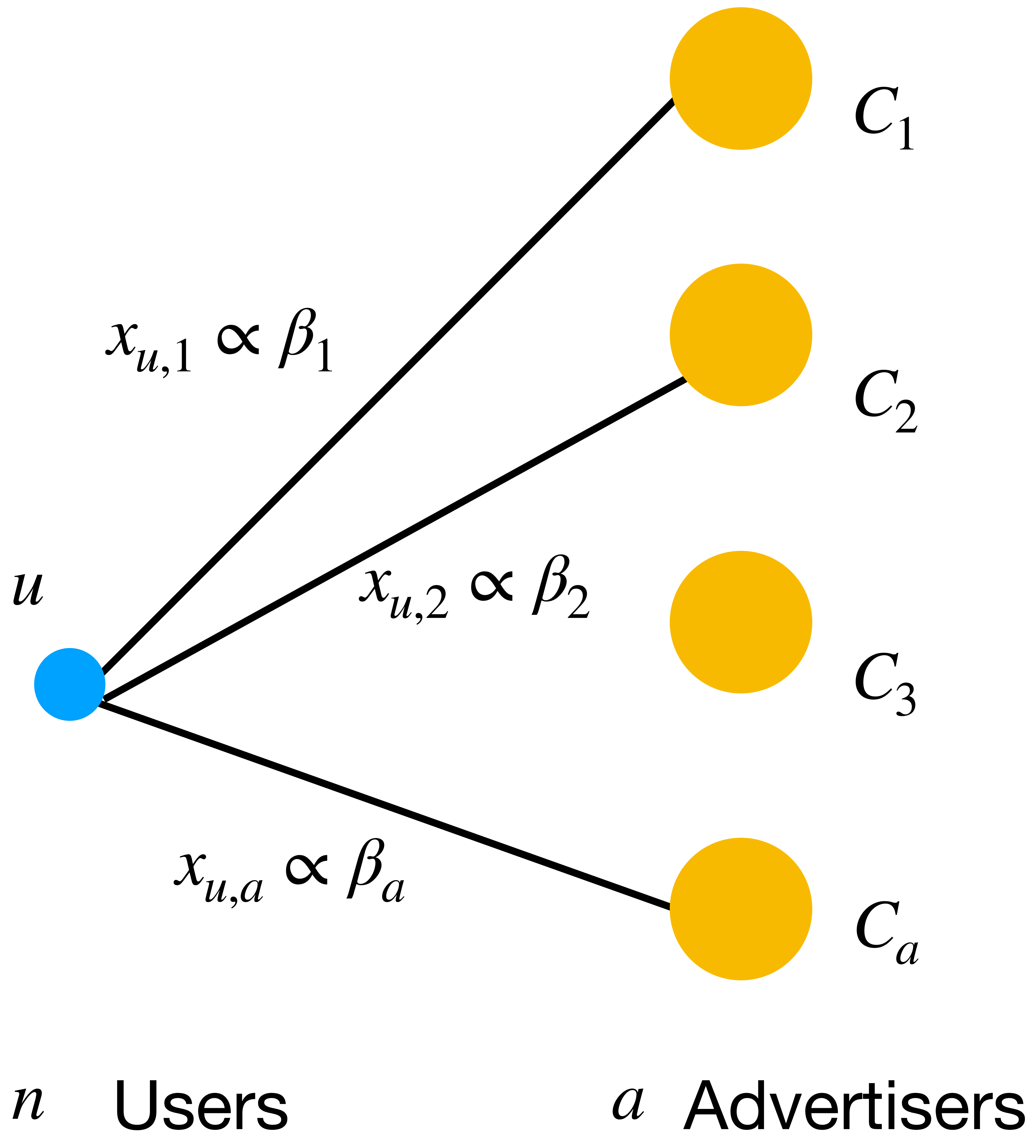
Theorem 2

The allocation algorithm can be implemented in $\tilde{O}_\epsilon(\sqrt{\log \lambda})$ sub-linear MPC rounds

Overall pipeline



The Allocation Algorithm



There exists a global "preference" for the advertisers

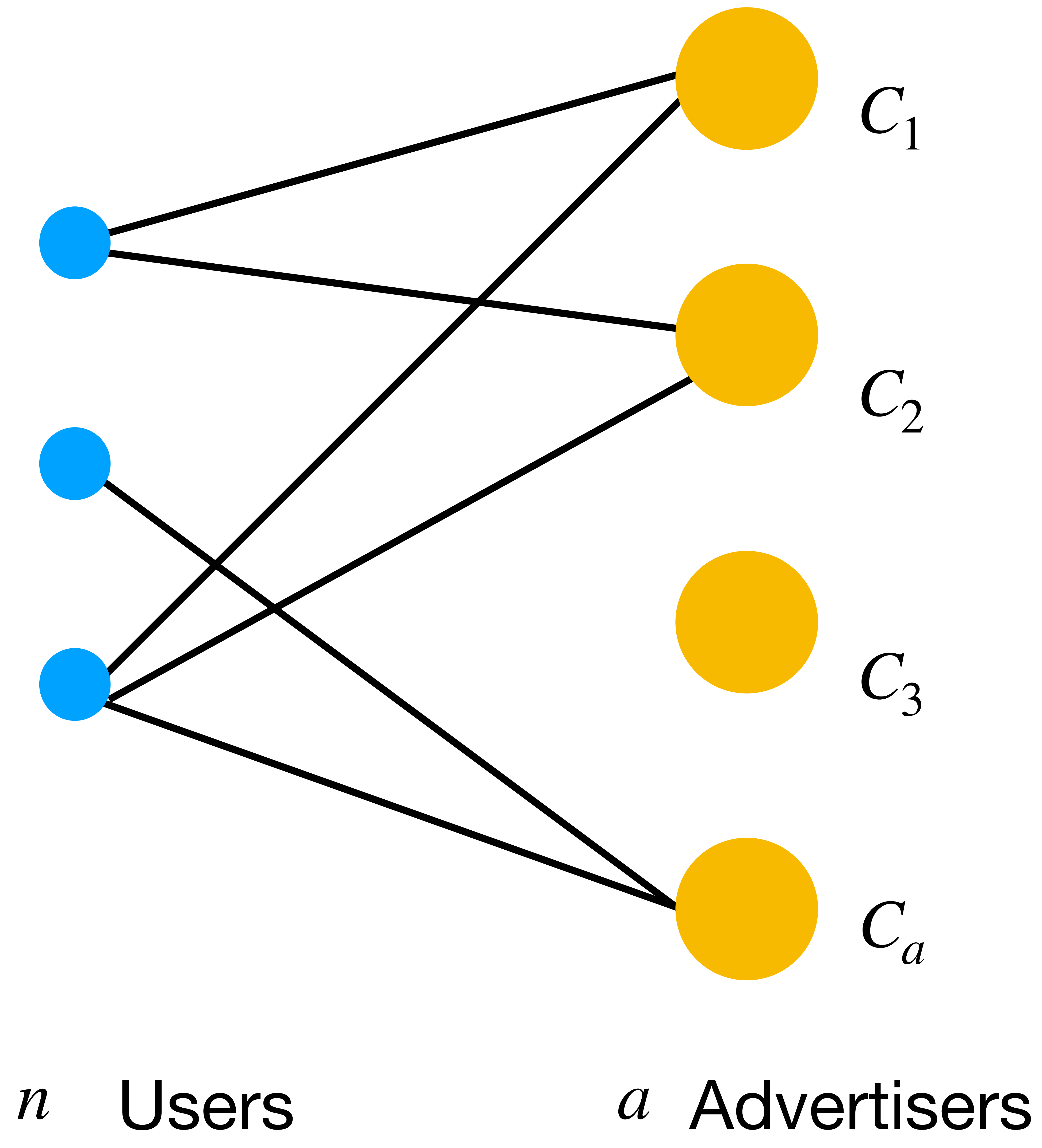
$\beta_v \in (0,\infty)$ for each advertiser v such that

assigning $x_{u,v} \propto \beta_v$

gives a $1+\epsilon$ approximate matching

$$x_{u,v} = \frac{\beta_v}{\sum_{(u,v')\text{exists}} \beta_{v'}}$$

The Allocation Algorithm



There exists a global "preference" for the advertisers

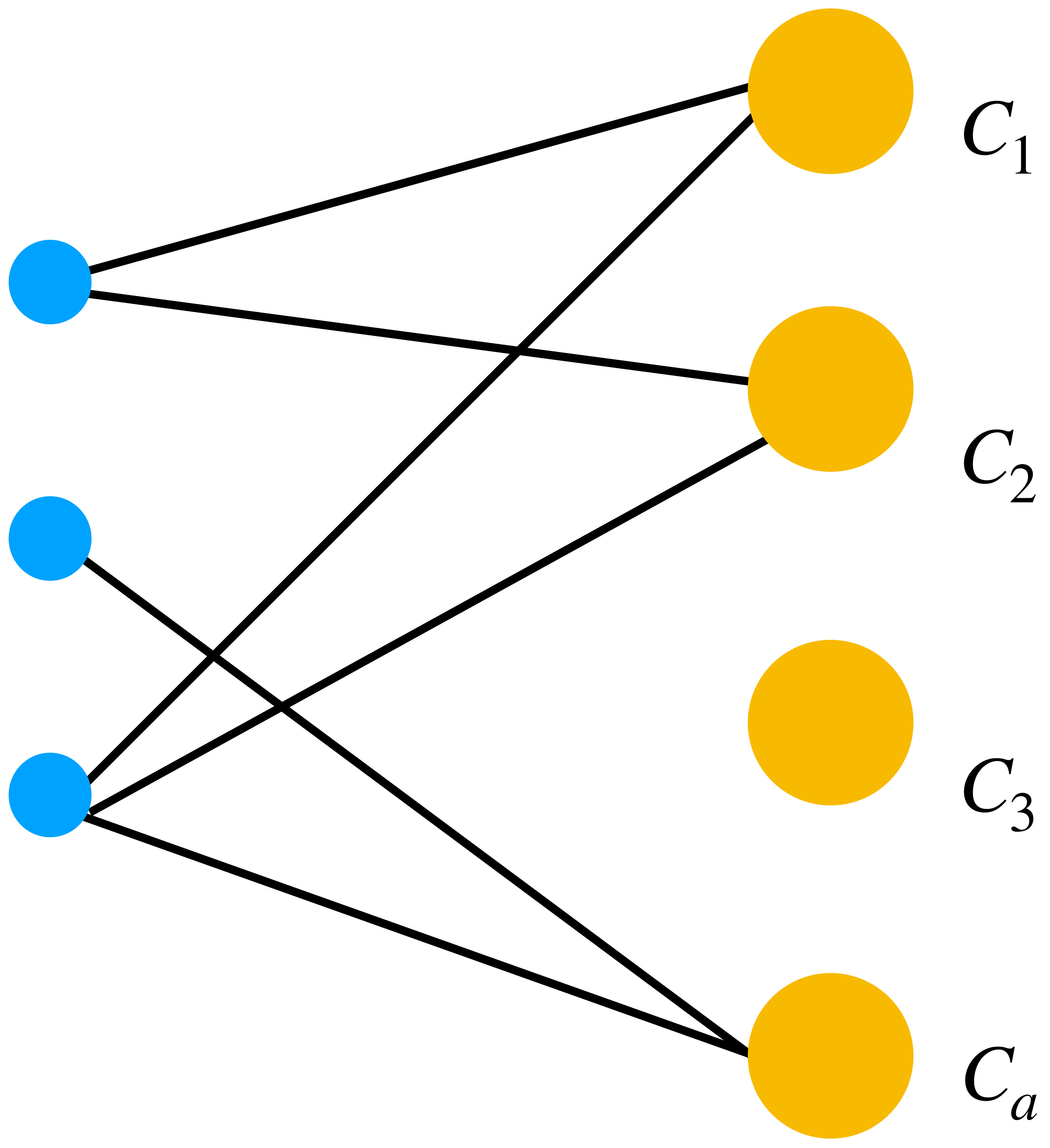
$\beta_v \in (0,\infty)$ for each advertiser v such that

assigning $x_{u,v} \propto \beta_v$

gives a $1+\epsilon$ approximate matching

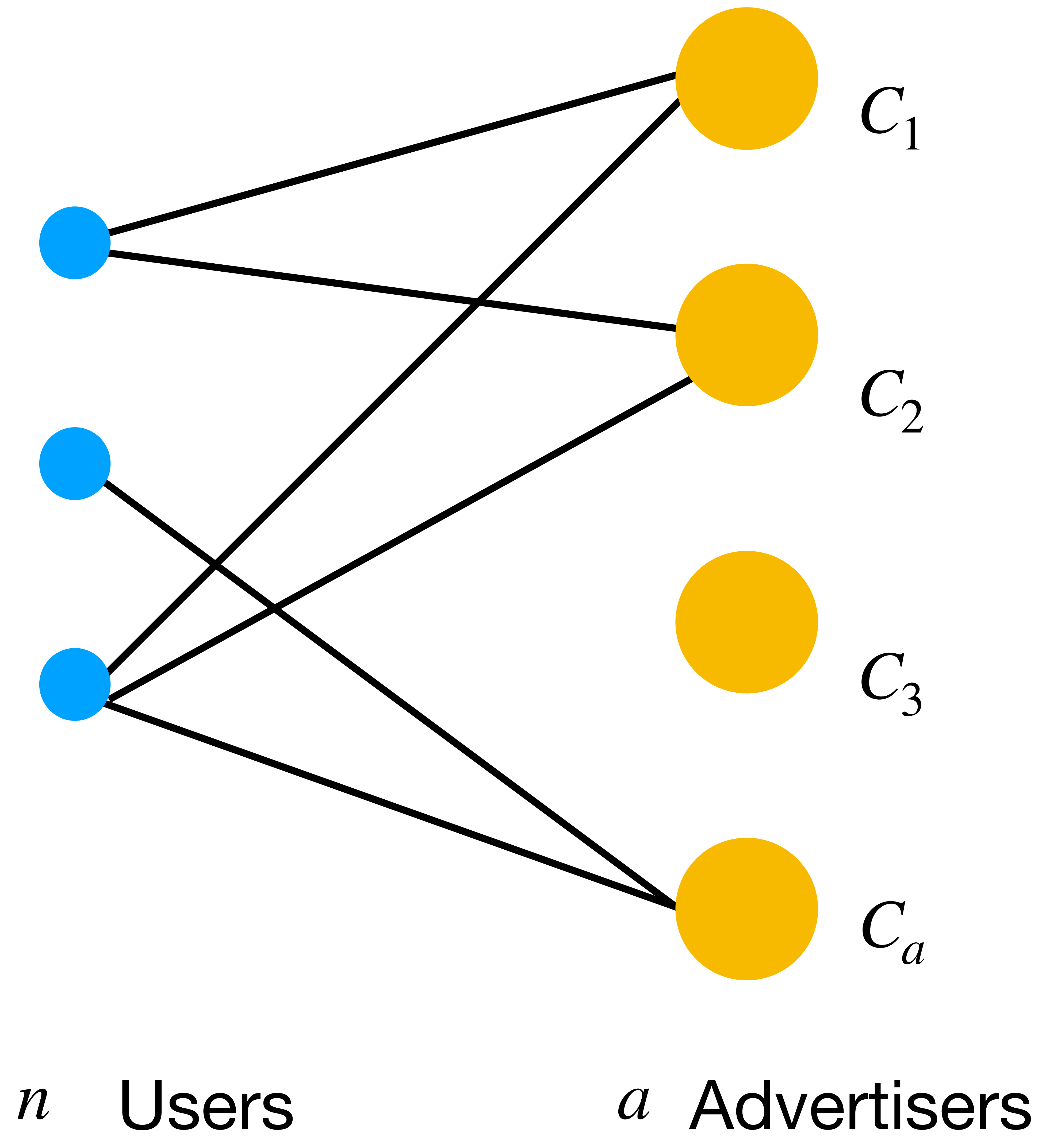
$$x_{u,v} = \frac{\beta_v}{\sum_{(u,v')\text{exists}} \beta_{v'}}$$

The Allocation Algorithm



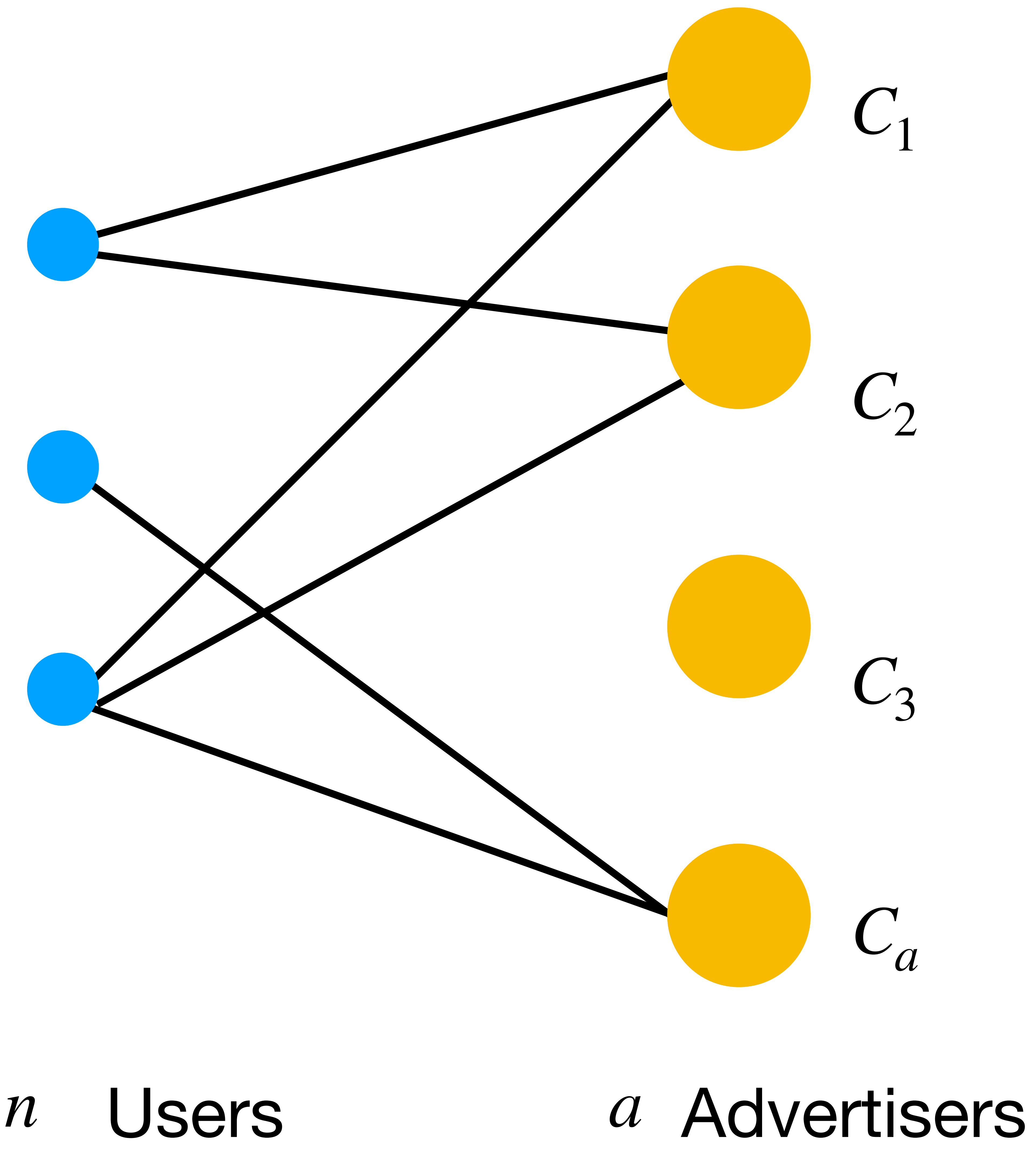
→ Start with $\beta_v = 1 \ \forall v$

The Allocation Algorithm



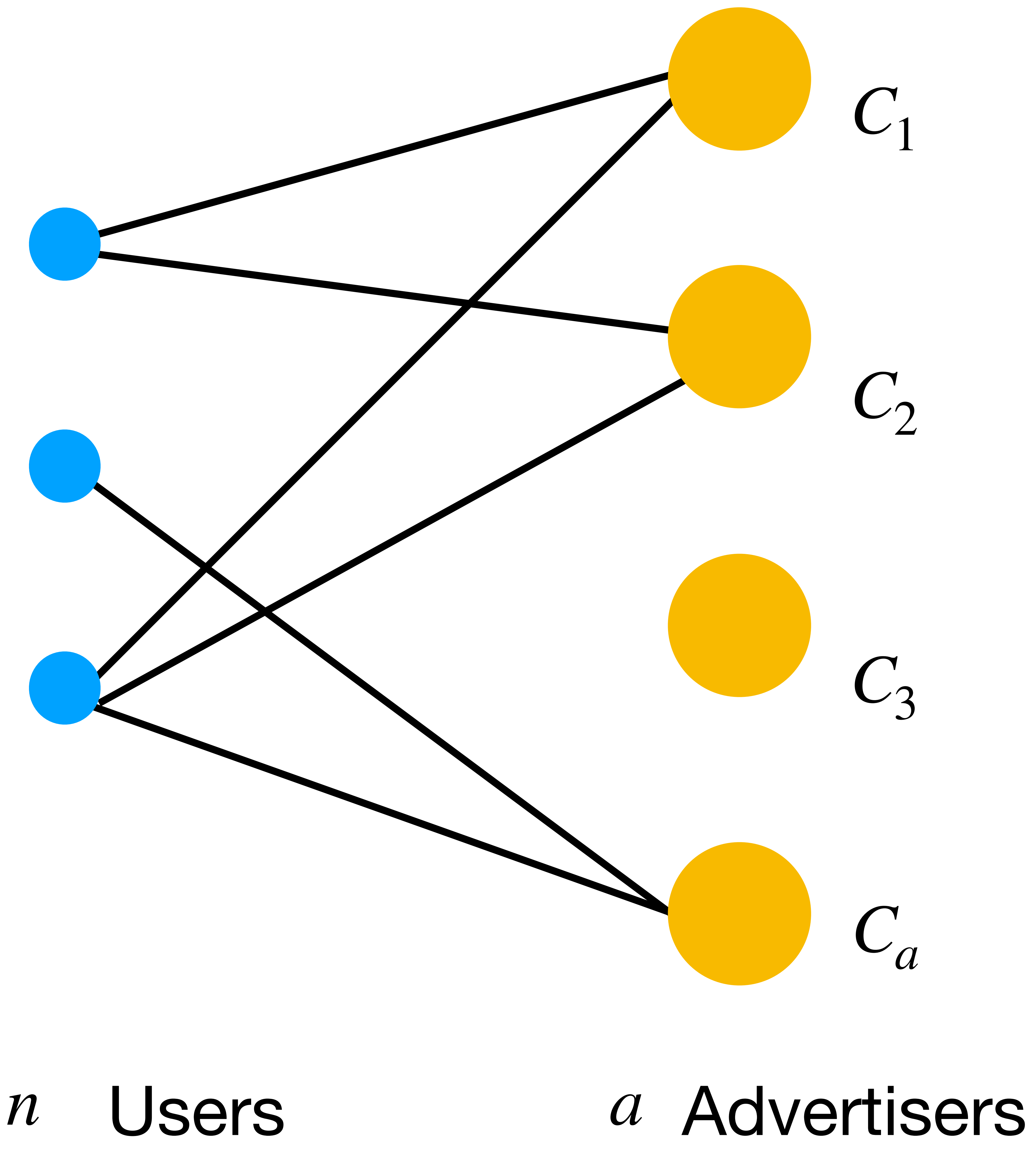
- Start with $\beta_v = 1 \ \forall v$
- Check how it does locally

The Allocation Algorithm



- Start with $\beta_v = 1 \ \forall v$
- Check how it does locally
- Change β_v by $1 + \epsilon$ factor

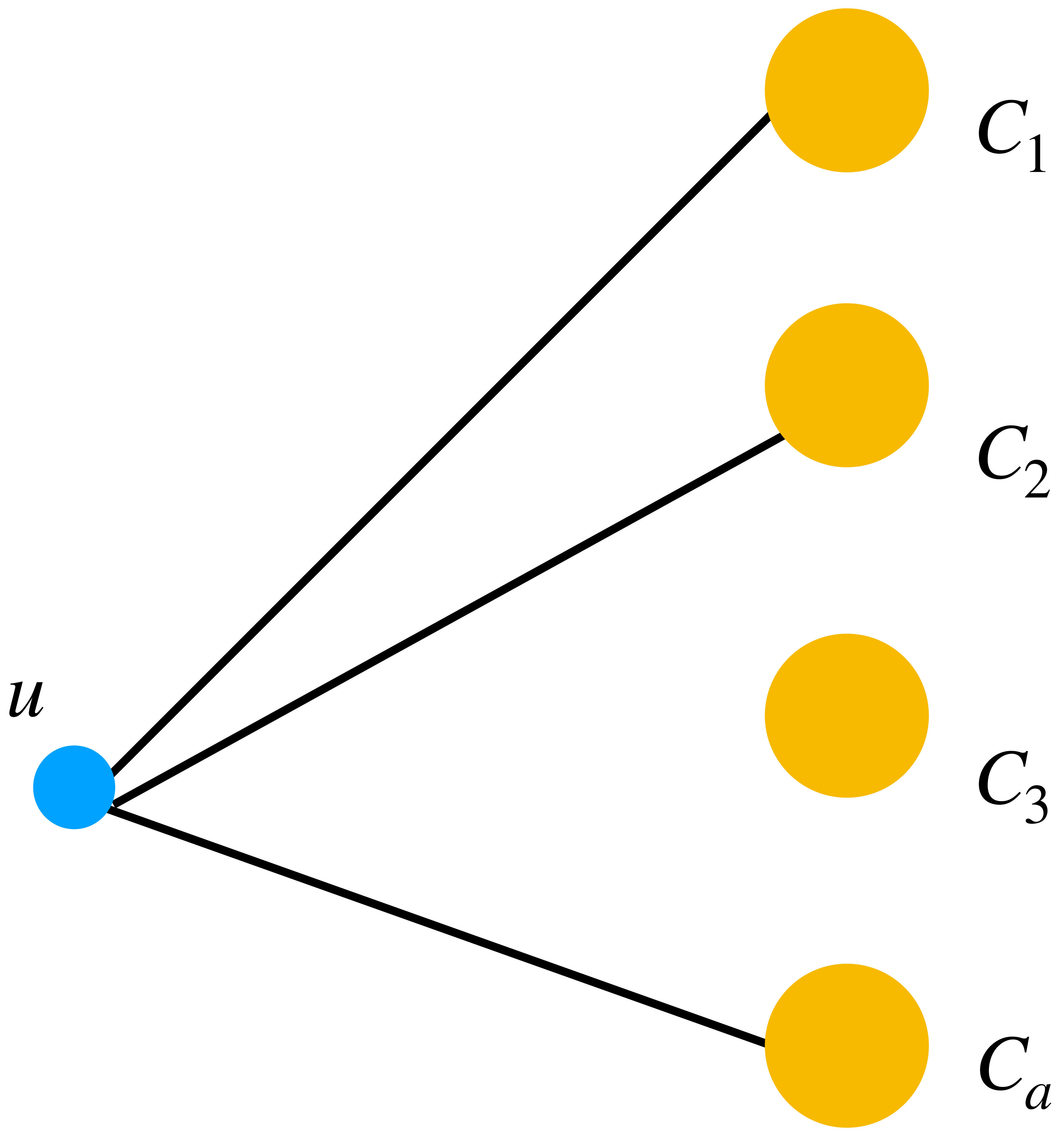
The Allocation Algorithm



- Start with $\beta_v = 1 \ \forall v$
- Check how it does locally
- Change β_v by $1 + \epsilon$ factor
- Repeat T times

The Allocation Algorithm

User Round

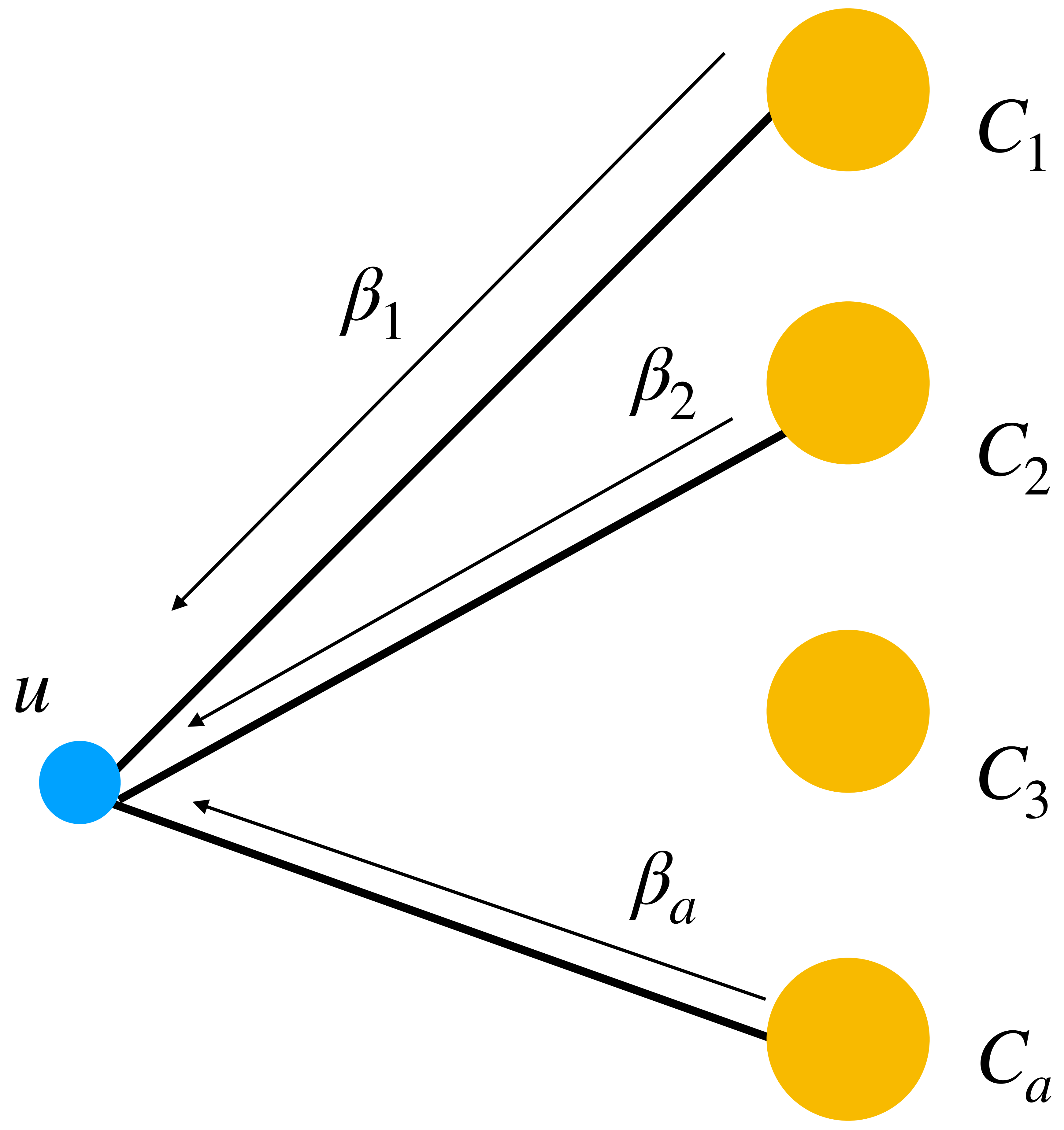


n Users

a Advertisers

The Allocation Algorithm

User Round

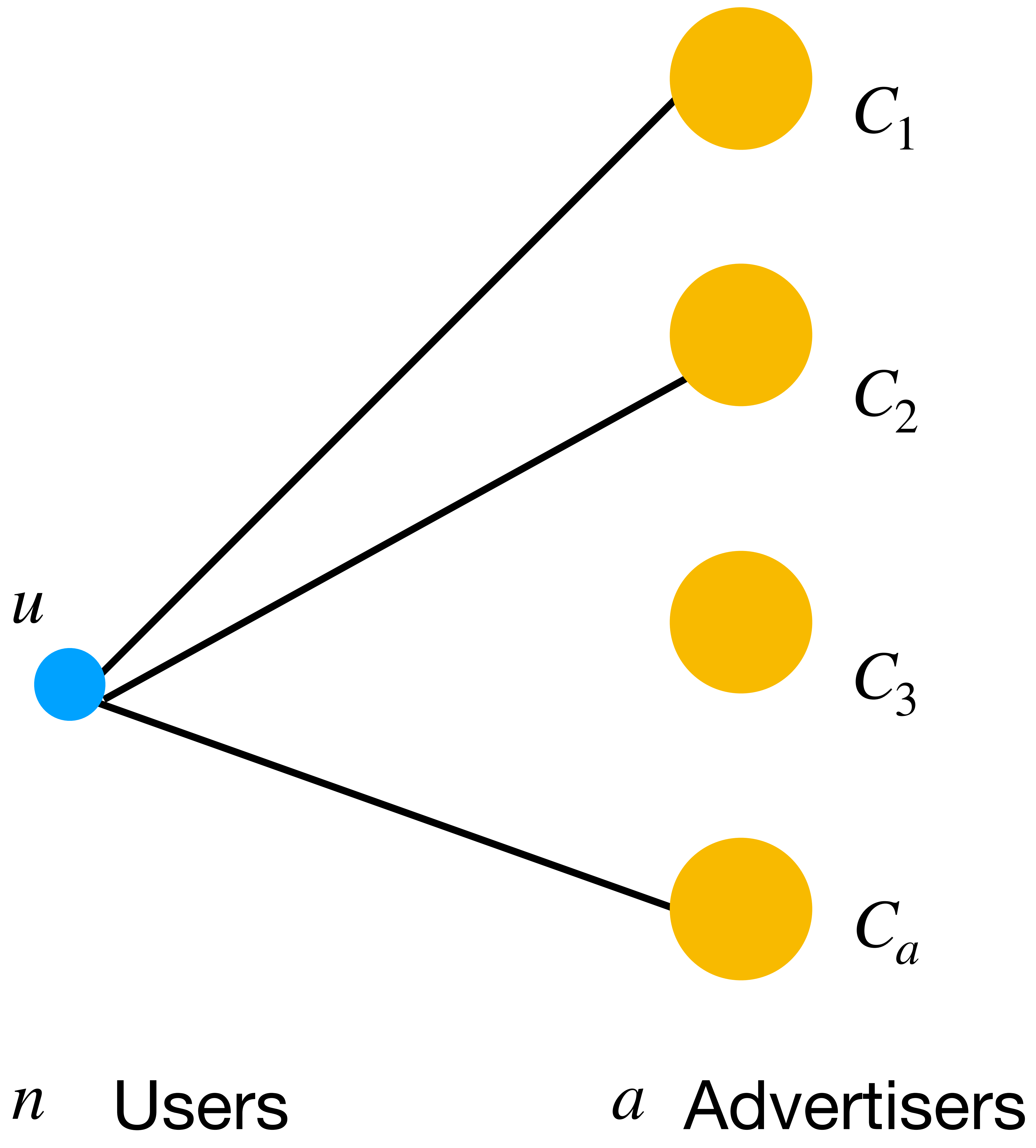


n Users a Advertisers

→ Each user u gets current β_v from neighbours

The Allocation Algorithm

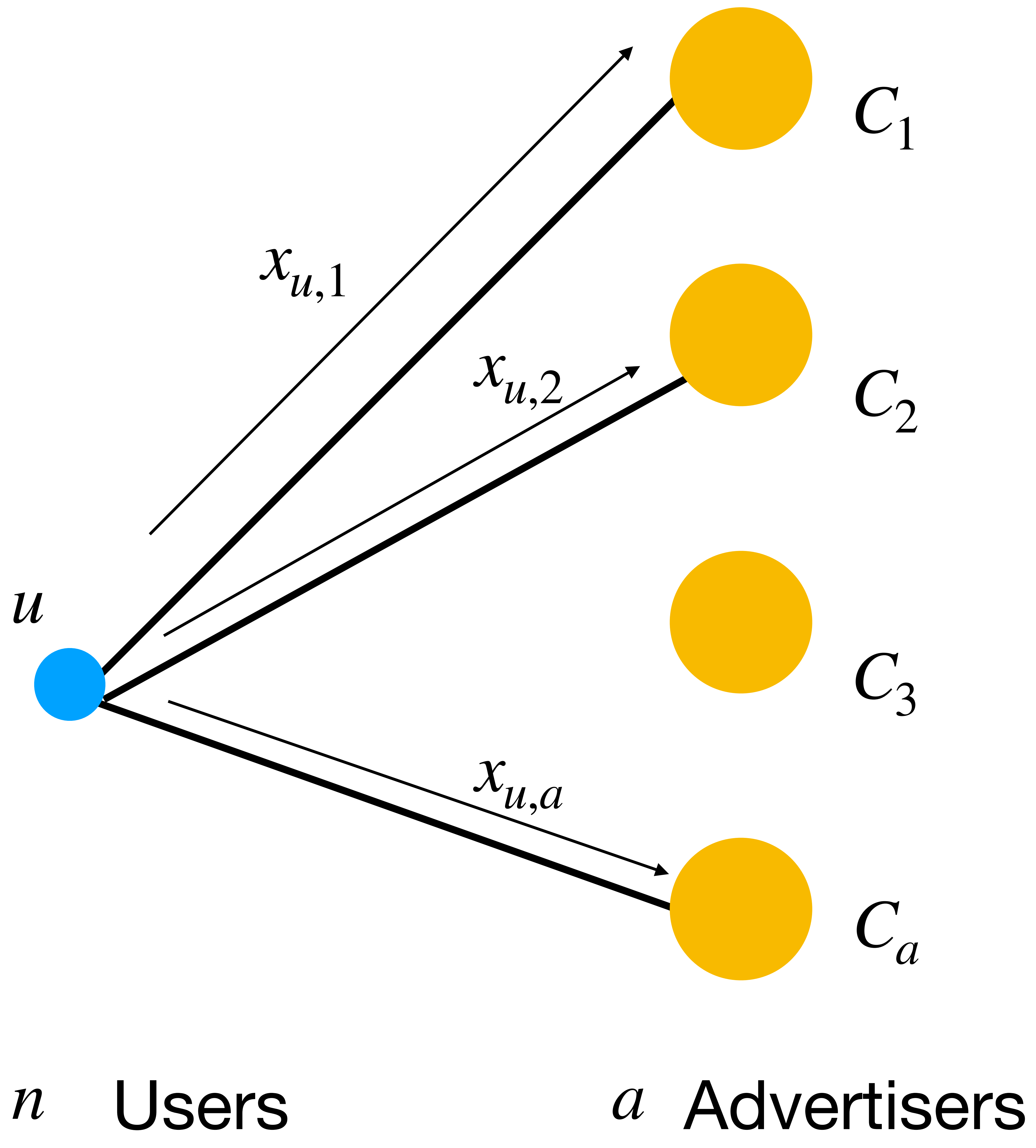
User Round



→ Each user u gets current β_v from neighbours

→ Compute $x_{u,v} = \dfrac{\beta_v}{\sum \beta_{v'}}$

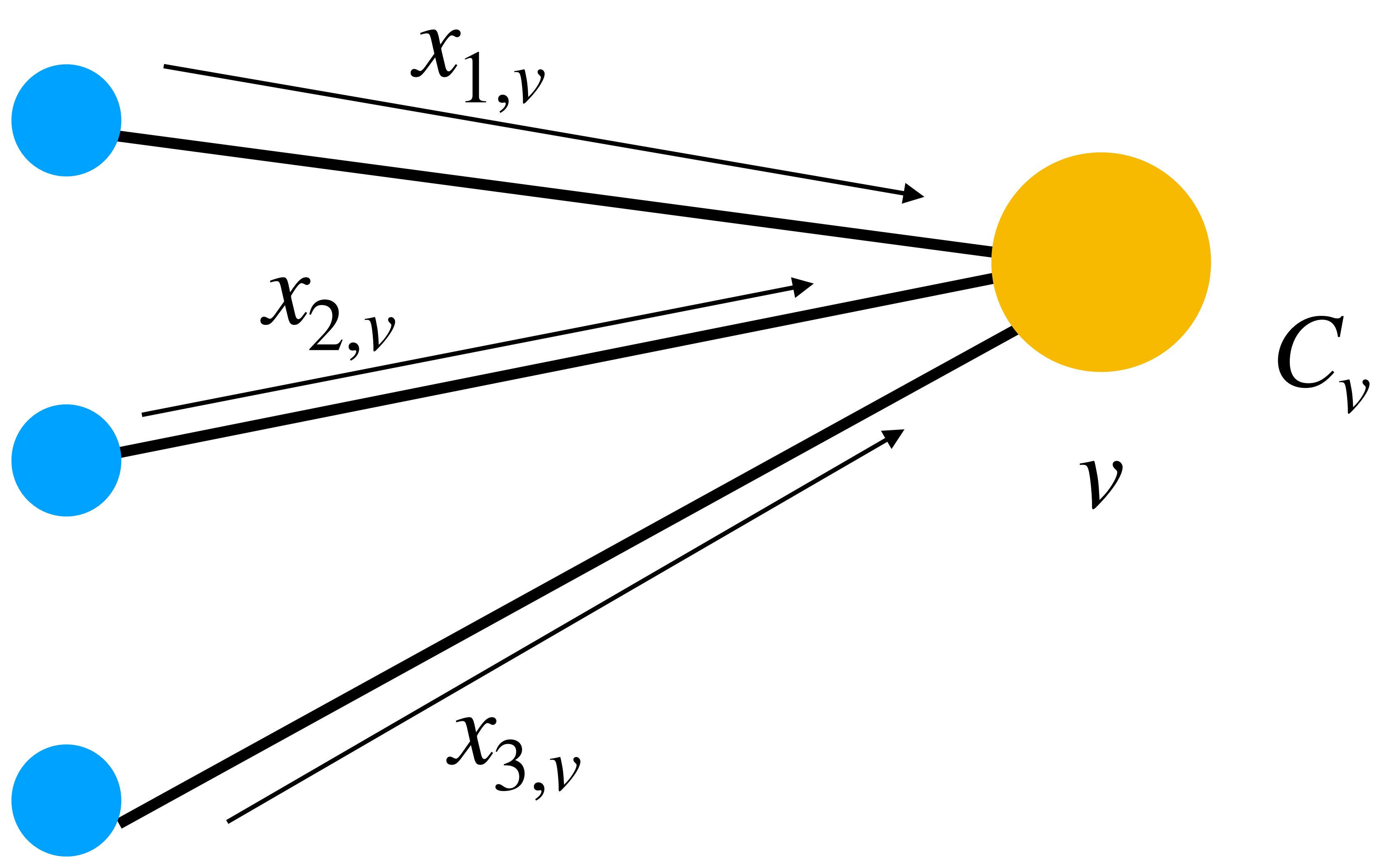
The Allocation Algorithm



User Round

- Each user u gets current β_v from neighbours
- Compute $x_{u,v} = \dfrac{\beta_v}{\sum \beta_{v'}}$
- Send $x_{u,v}$ to v

The Allocation Algorithm



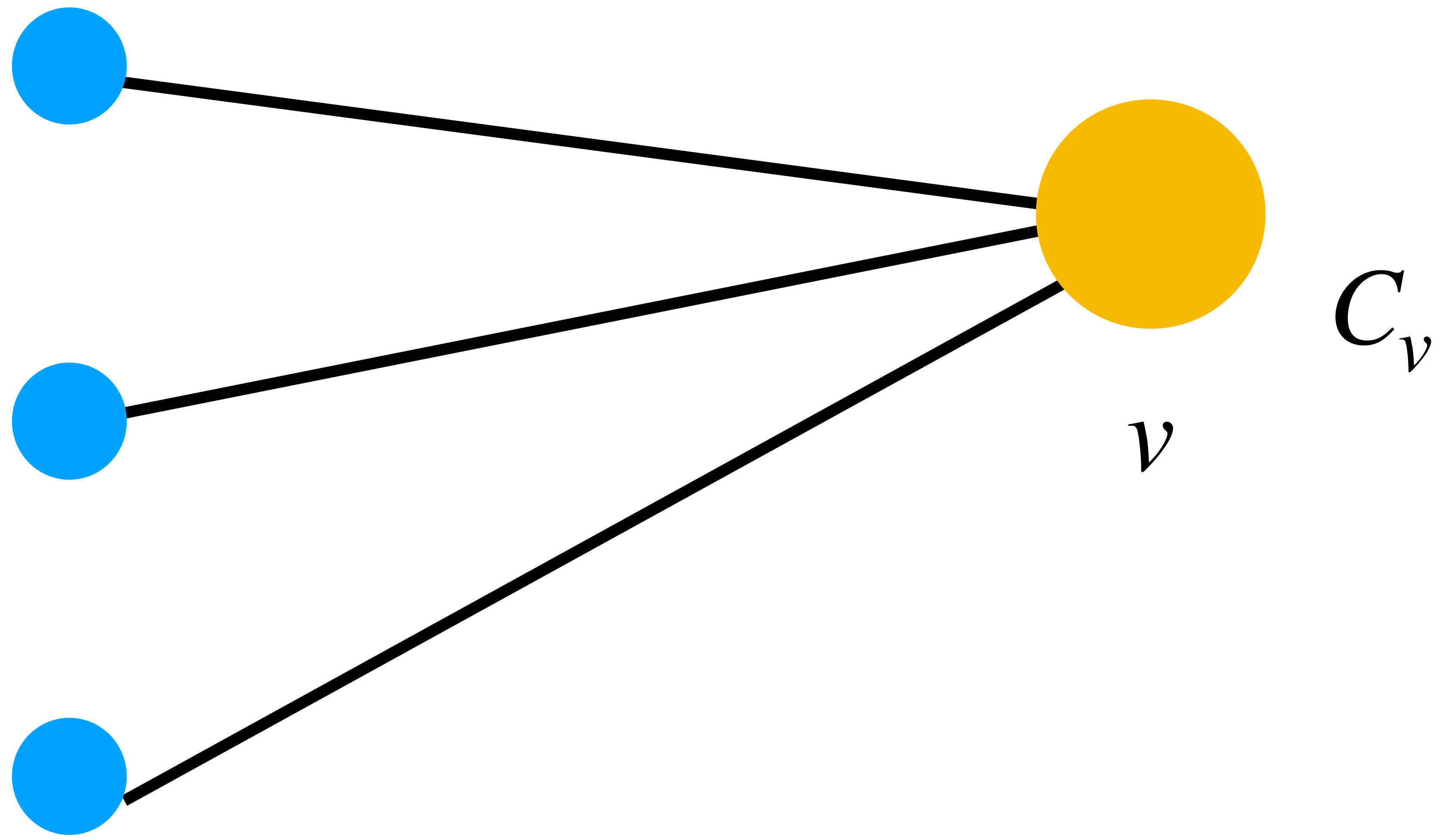
Advertiser Round

→ Receive $x_{u,v}$

n Users

a Advertisers

The Allocation Algorithm



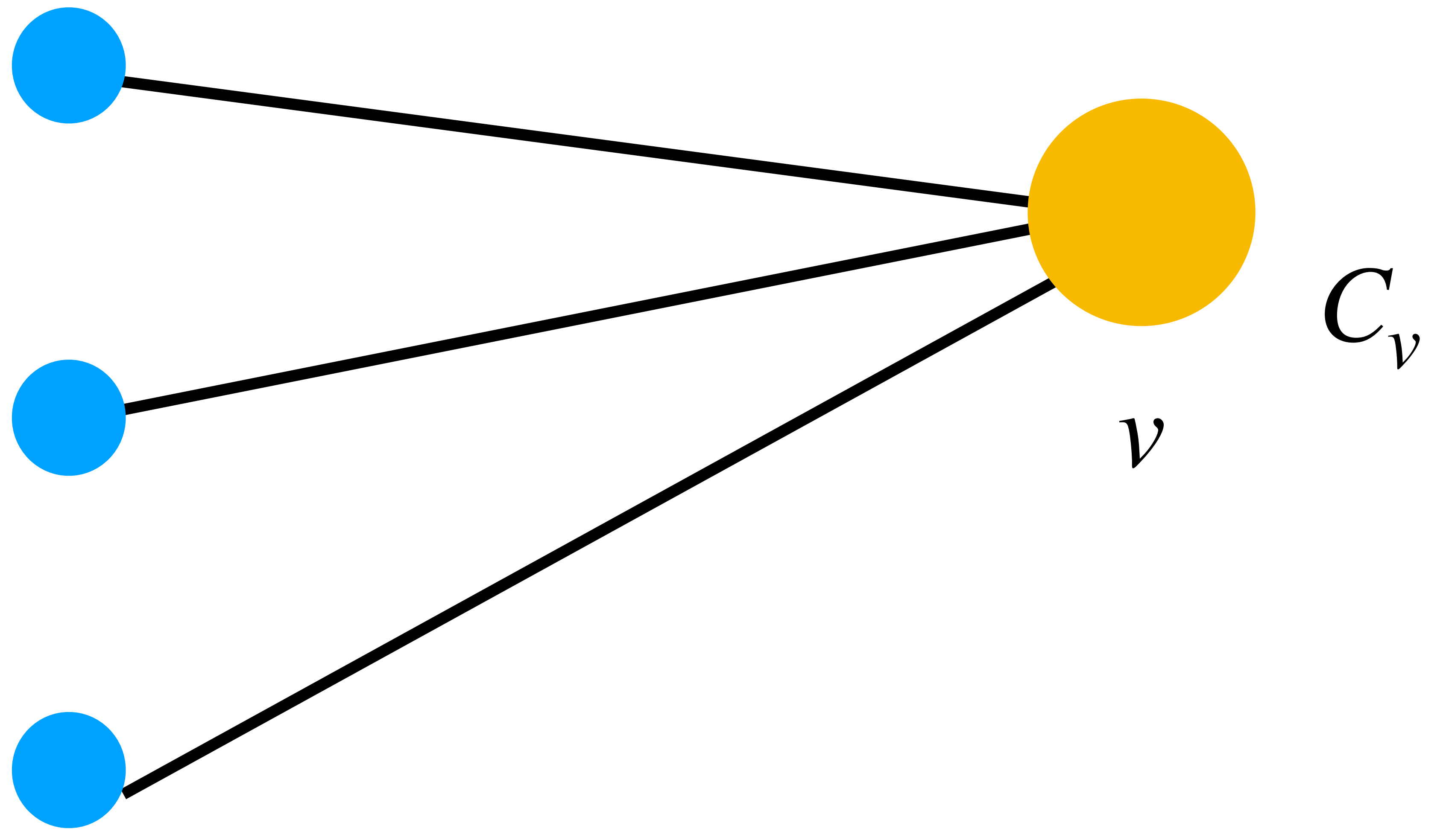
Advertiser Round

→ Receive $x_{u,v}$

→ Compute alloc$_v = \sum x_{u,v}$

n Users a Advertisers

The Allocation Algorithm



n Users a Advertisers

Advertiser Round

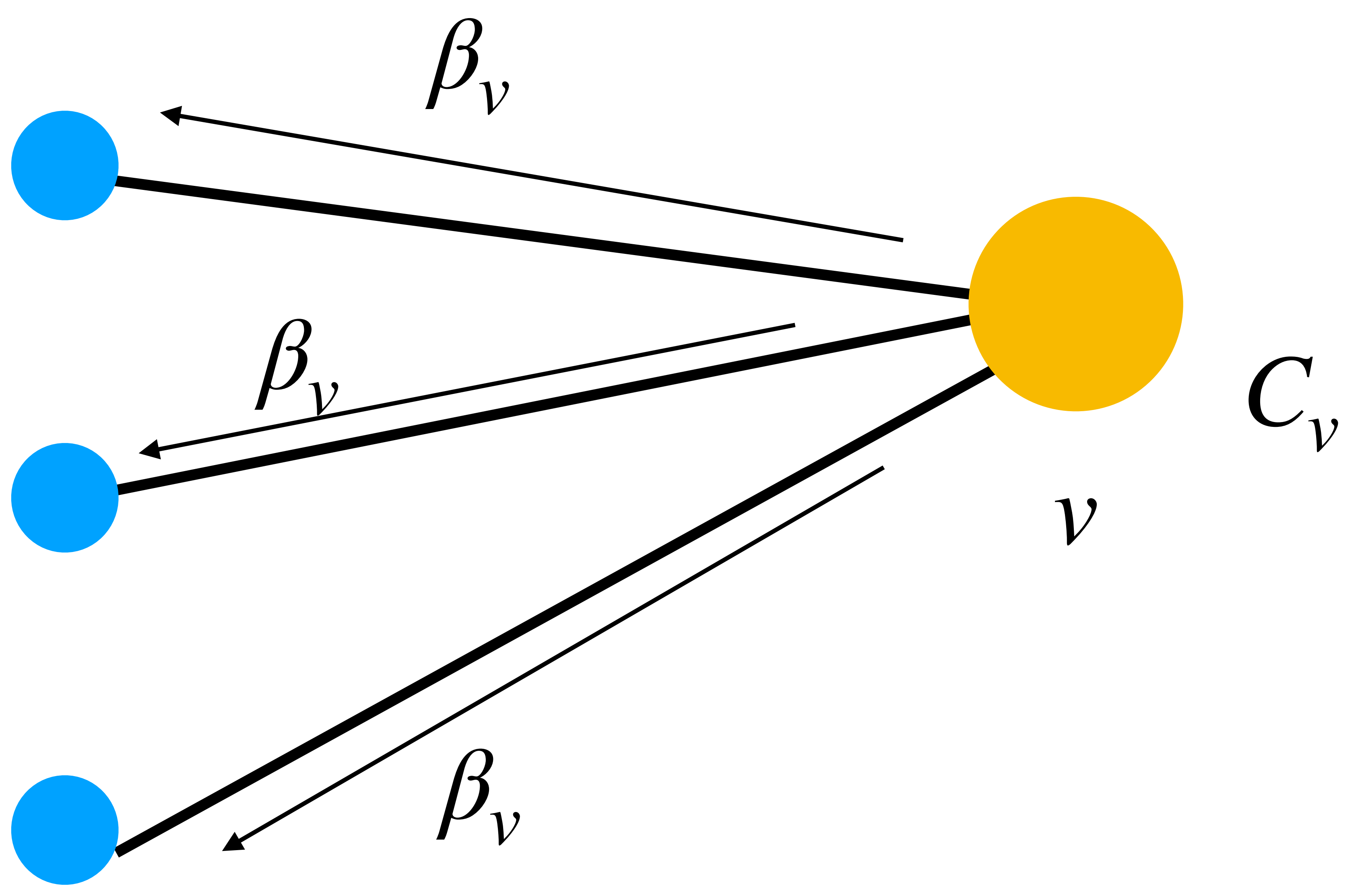
- Receive $x_{u,v}$
- Compute $\text{alloc}_v = \sum x_{u,v}$
- if $\text{alloc}_v < C_v/(1+\epsilon)$

  increase β_v by $1+\epsilon$ factor

  else if $\text{alloc}_v > C_v(1+\epsilon)$

  decrease β_v by $1+\epsilon$ factor

The Allocation Algorithm



n Users a Advertisers

Advertiser Round

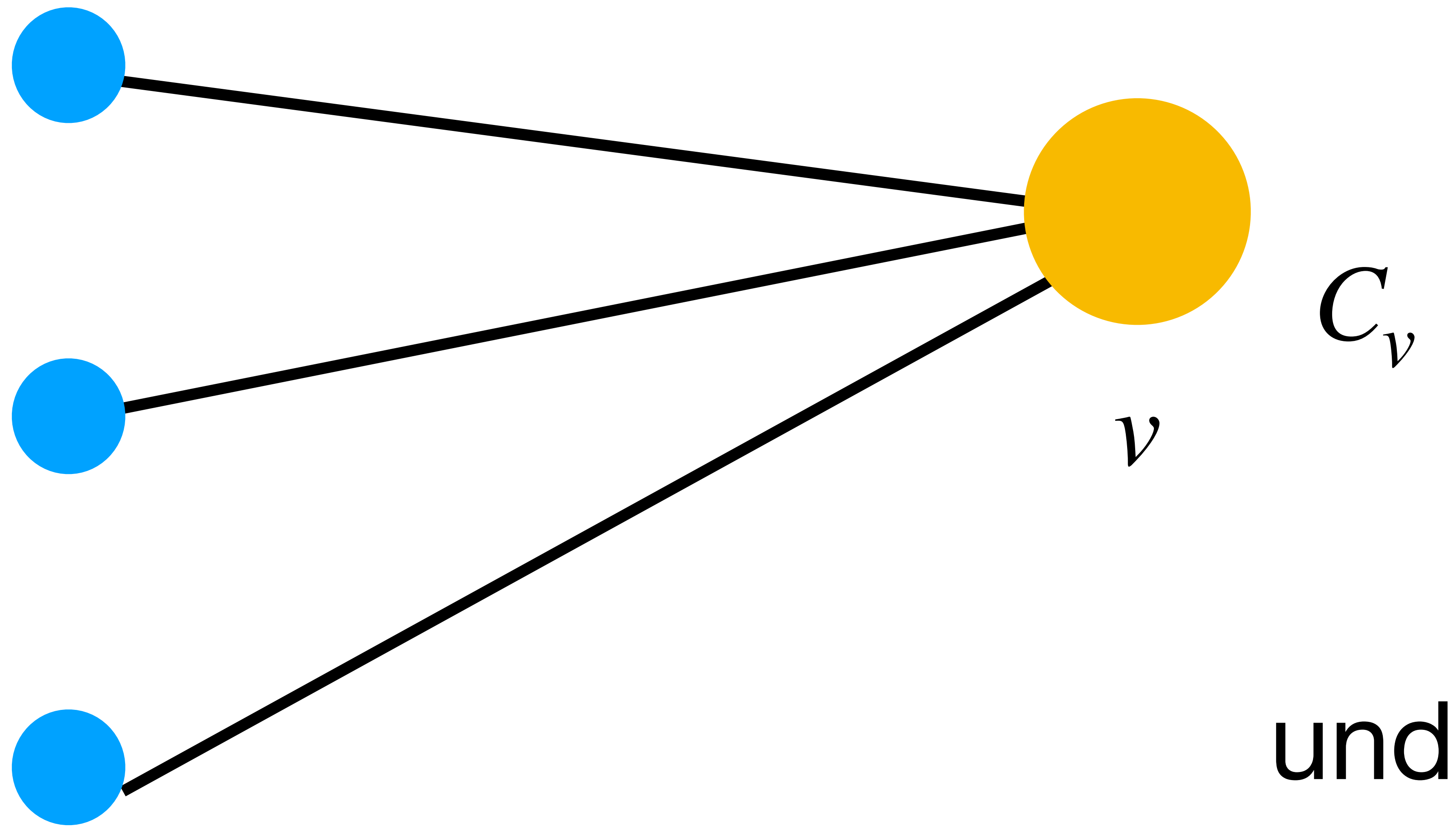
- Receive $x_{u,v}$
- Compute $\text{alloc}_v = \sum x_{u,v}$
- if $\text{alloc}_v < C_v/(1+\epsilon)$

  increase β_v by $1+\epsilon$ factor

  else if $\text{alloc}_v > C_v(1+\epsilon)$

  decrease β_v by $1+\epsilon$ factor
- Send β_v

The Allocation Algorithm



n Users a Advertisers

Advertiser Round

Receive $x_{u,v}$

Compute $\text{alloc}_v = \sum x_{u,v}$

under-allocation ← if $\text{alloc}_v < C_v/(1+\epsilon)$

increase β_v by $1+\epsilon$ factor

over-allocation ← else if $\text{alloc}_v > C_v(1+\epsilon)$

decrease β_v by $1+\epsilon$ factor

Send β_v

The Allocation Algorithm

User Round

- Each user u gets current β_v from neighbours
- Compute $x_{u,v} = \frac{\beta_v}{\sum \beta_{v'}}$
- Send $x_{u,v}$ to v

Advertiser Round

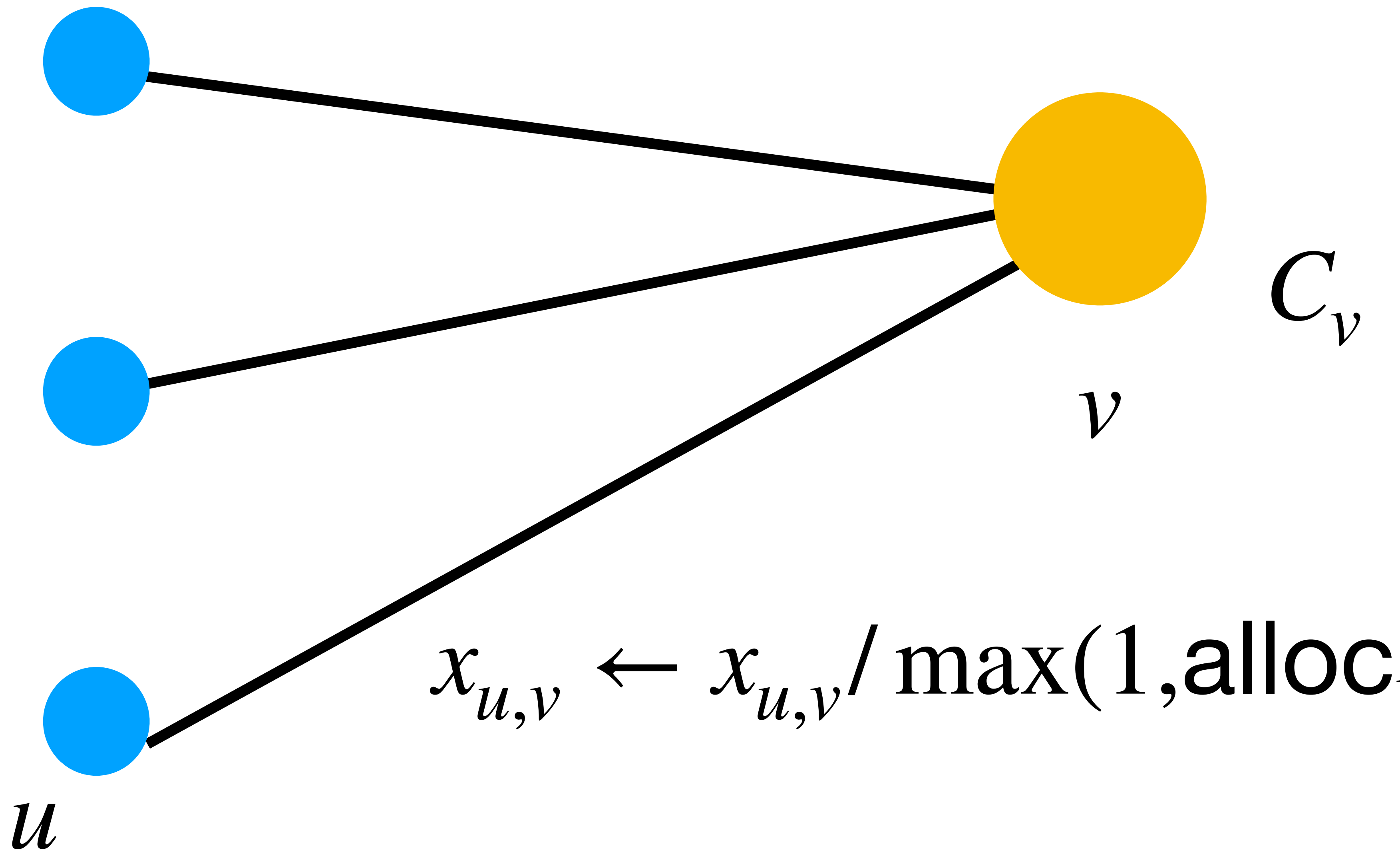
- Receive $x_{u,v}$
- Compute alloc$_v = \sum x_{u,v}$
- if alloc$_v < C_v/(1+\epsilon)$

  increase β_v by $1+\epsilon$ factor

  else if alloc$_v > C_v(1+\epsilon)$

  decrease β_v by $1+\epsilon$ factor
- Send β_v

The Allocation Algorithm



$x_{u,v} \leftarrow x_{u,v} / \max(1, \text{alloc}_v / C_v)$

n Users

a Advertisers

Last Round

If $\text{alloc}_v > C_v$

Rescale $x_{u,v}$ so that $\text{alloc}_v = C_v$

Feasible matching guaranteed

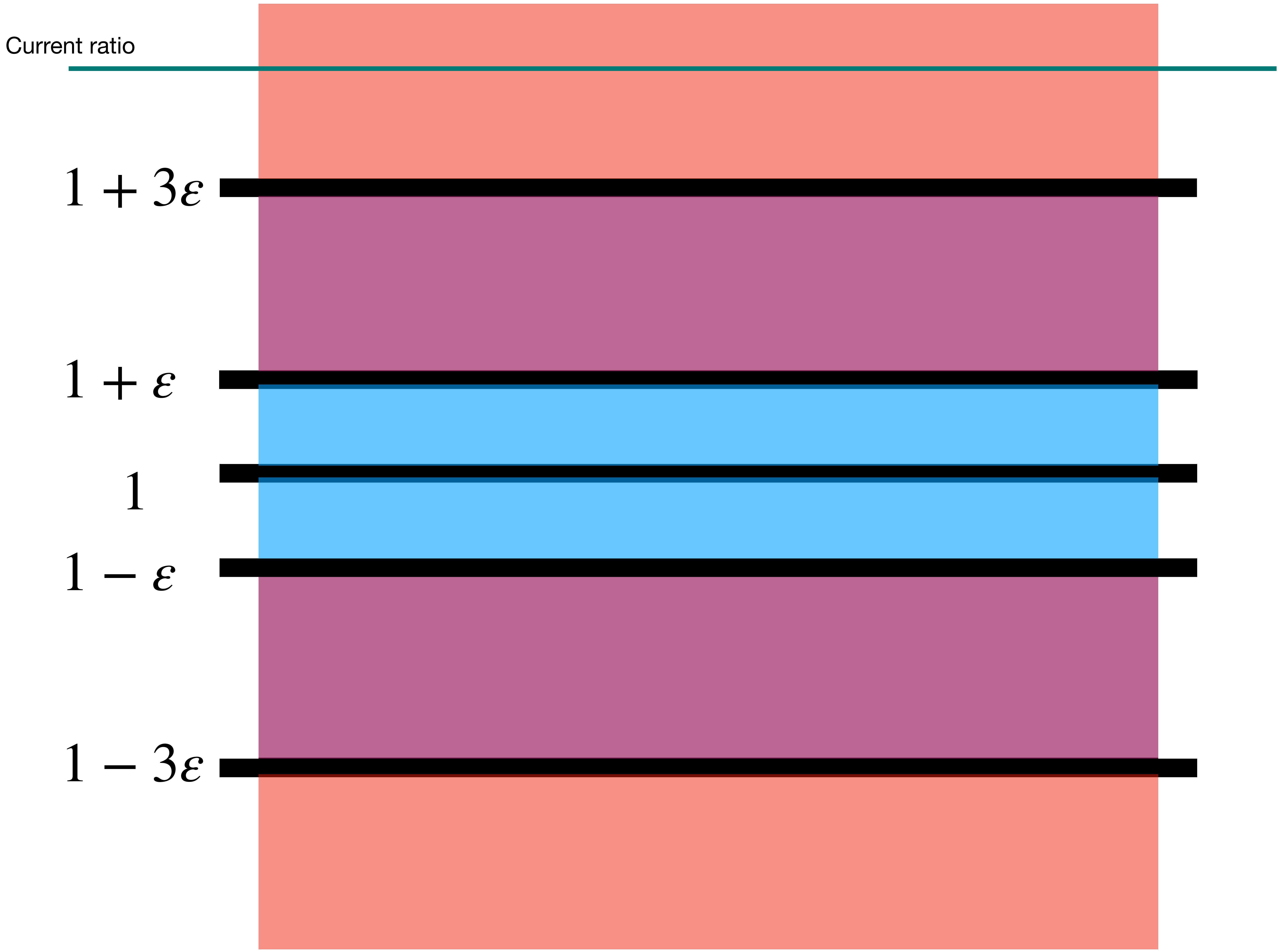
Proof of approximation

How does over/under allocation for a fixed advertiser change with time?

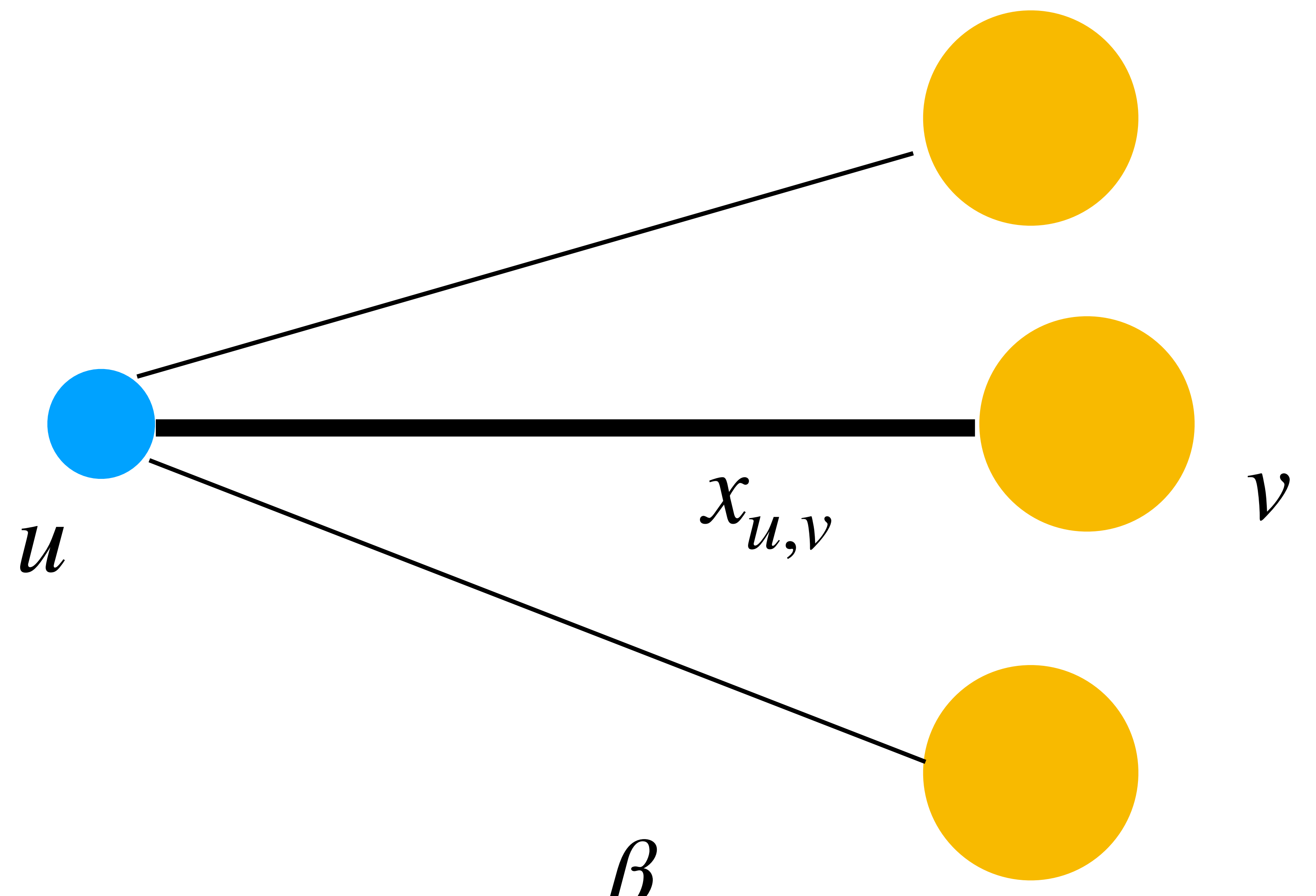
Claim

$\beta_v, x_{u,v}$, alloc_v / C_v change by $1 + O(\epsilon)$ factor each round

Study Allocation / Capacity ratio



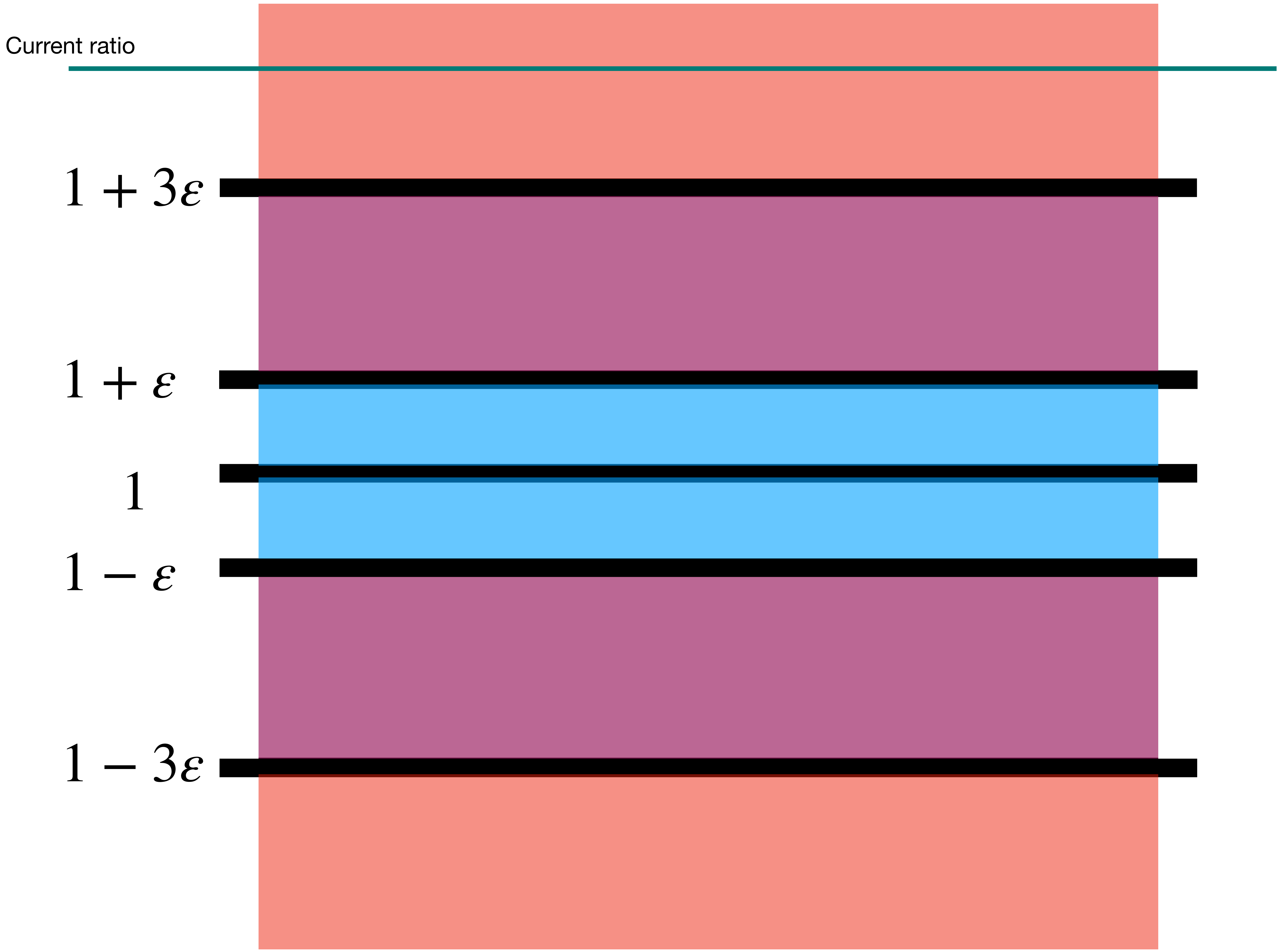
Proof of approximation



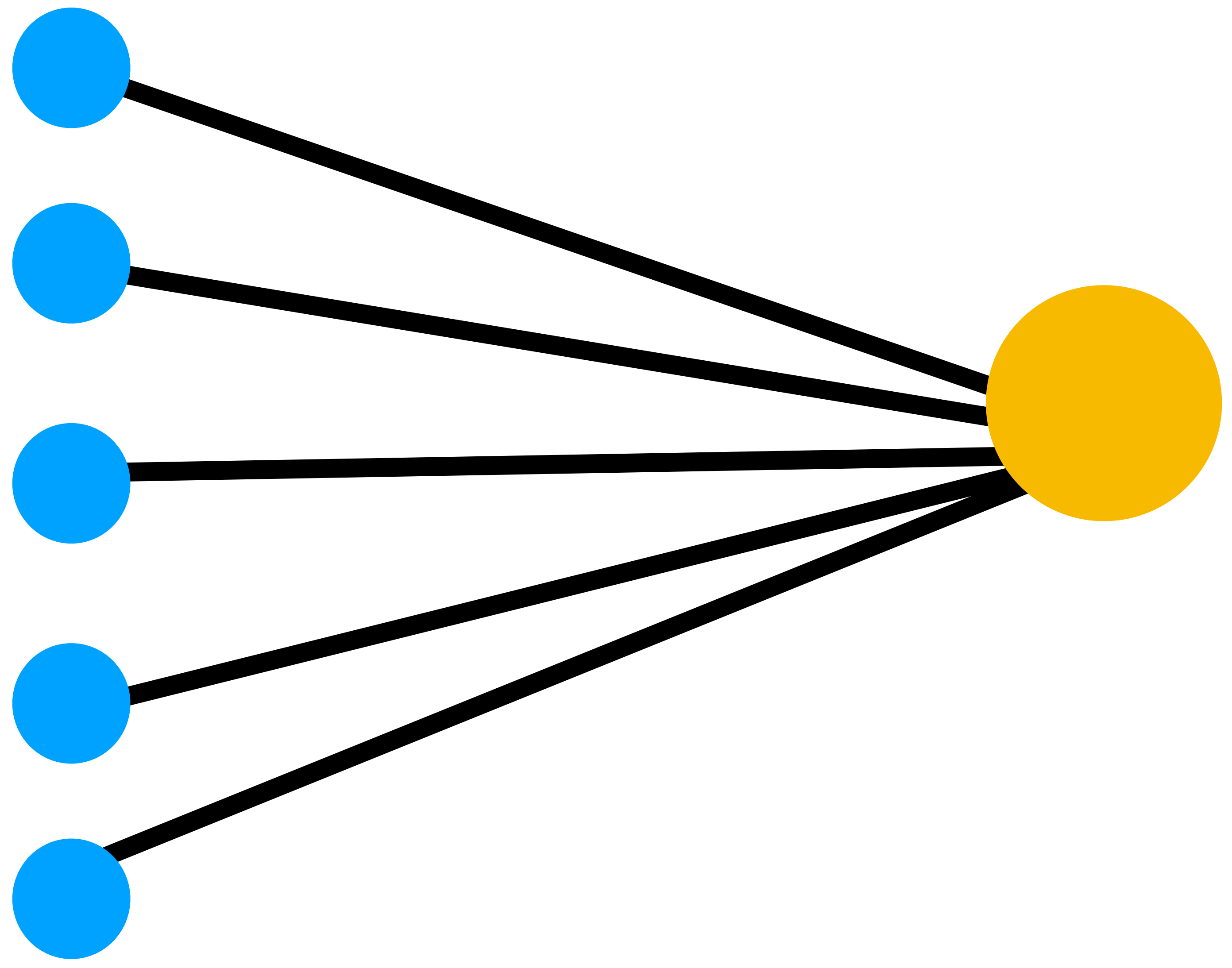
$$x_{u,v} = \frac{\beta_v}{\sum \beta_{v'}}$$

$$x'_{u,v} \in [\frac{1}{(1+\epsilon)^2}, (1+\epsilon)^2] \cdot x_{u,v}$$

Study Allocation / Capacity ratio

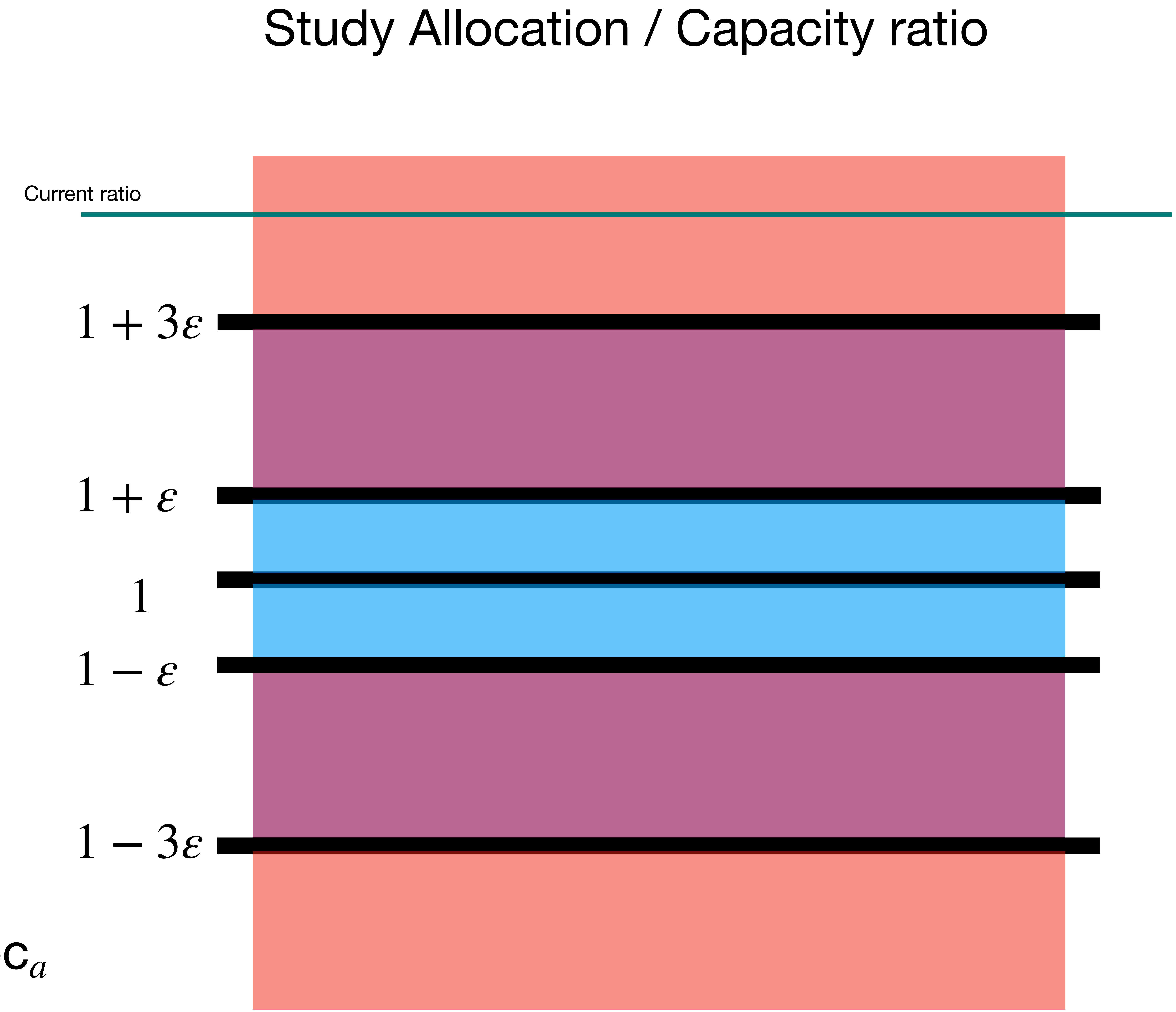


Proof of approximation



$$\text{alloc}_a = \sum x_{u',a}$$

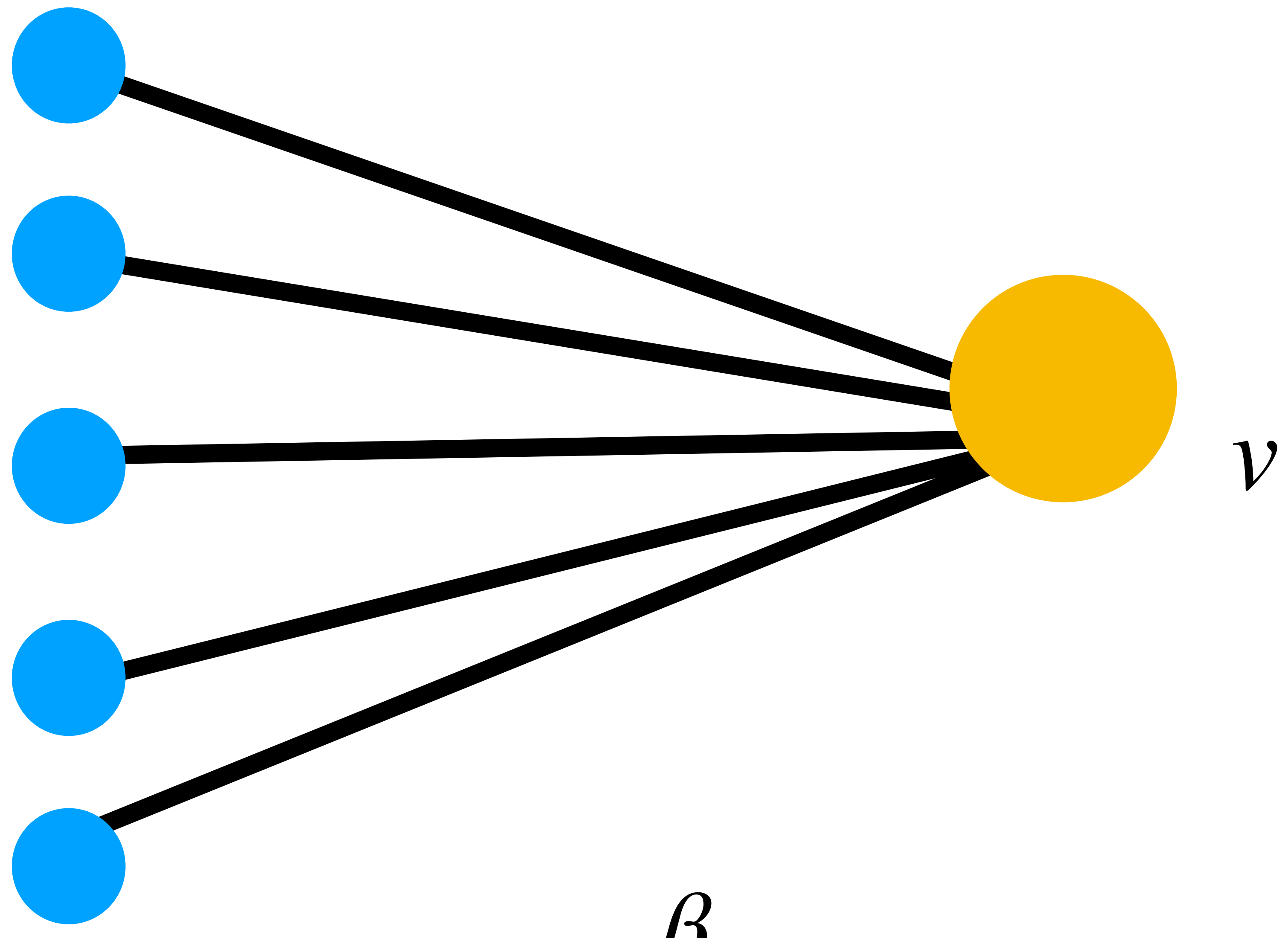
$$\text{alloc}_a^{new} \in [\frac{1}{(1+\epsilon)^2}, (1+\epsilon)^2] \cdot \text{alloc}_a$$

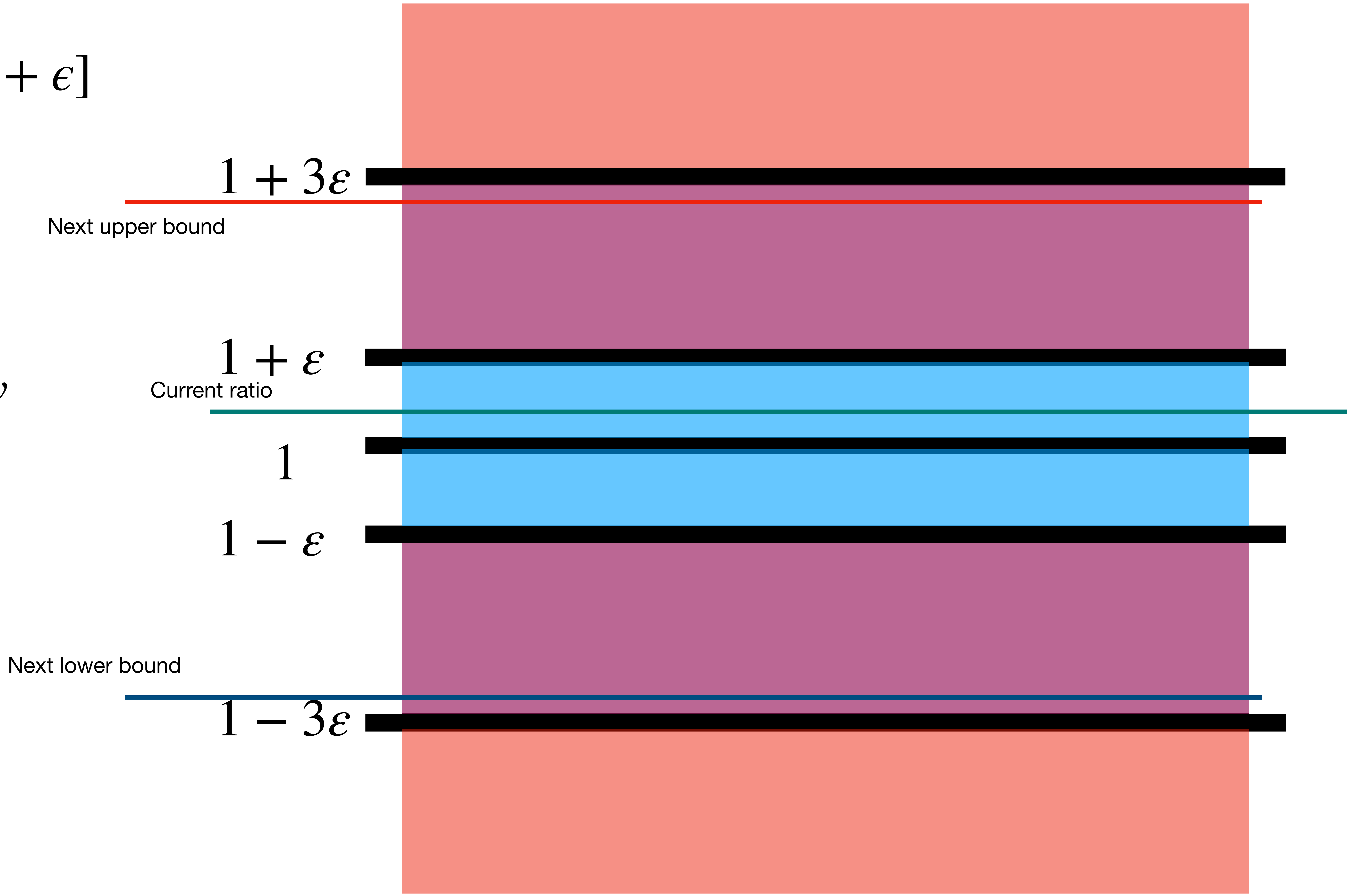


Proof of approximation

Case 1: Blue region

$$\text{alloc}_a / C_a \in [1-\epsilon, 1+\epsilon]$$





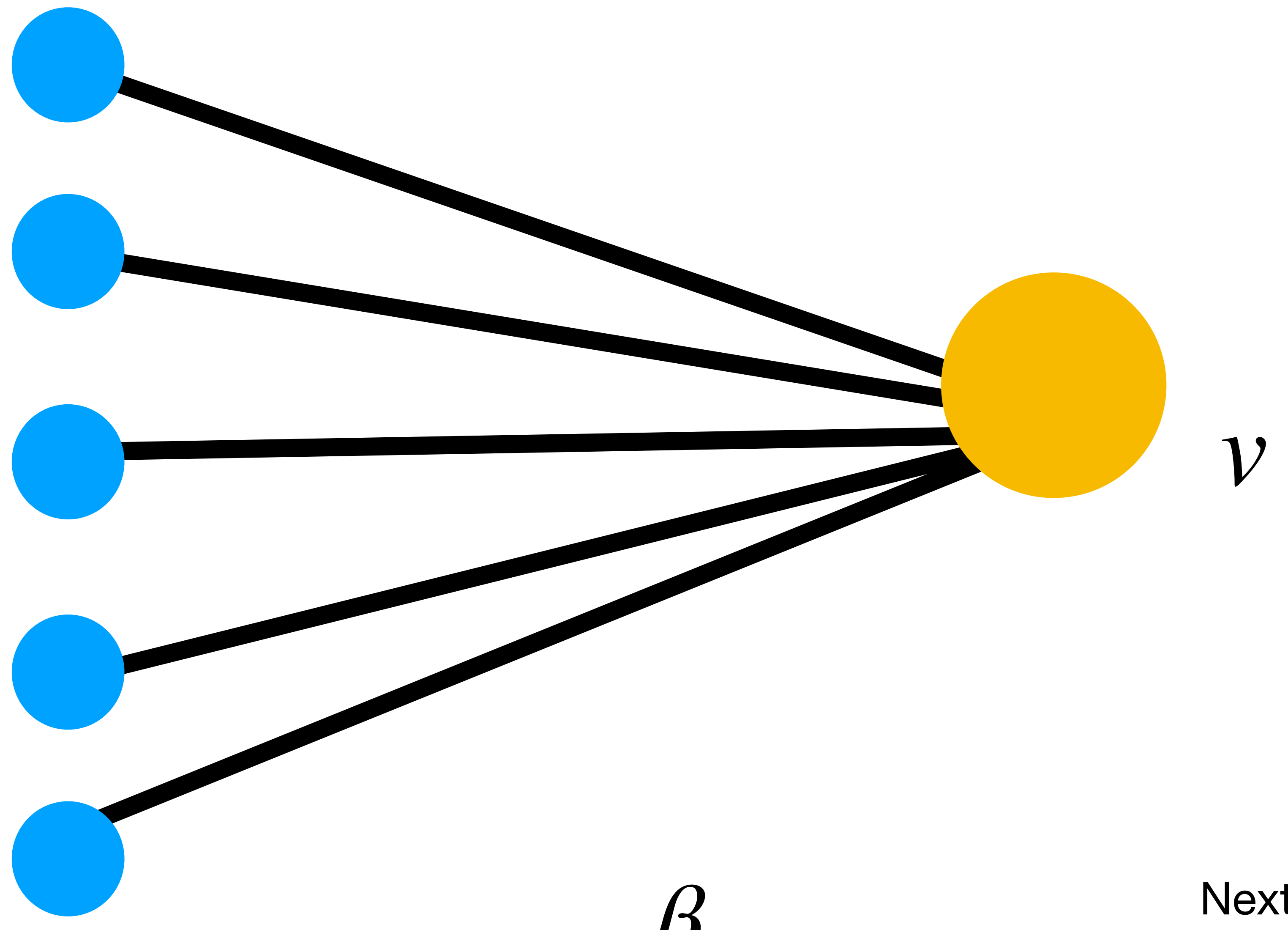
$$x_{u,v} = \frac{\beta_v}{\sum \beta_{v'}}$$

$$\text{alloc}_a = \sum x_{u',a}$$

Proof of approximation

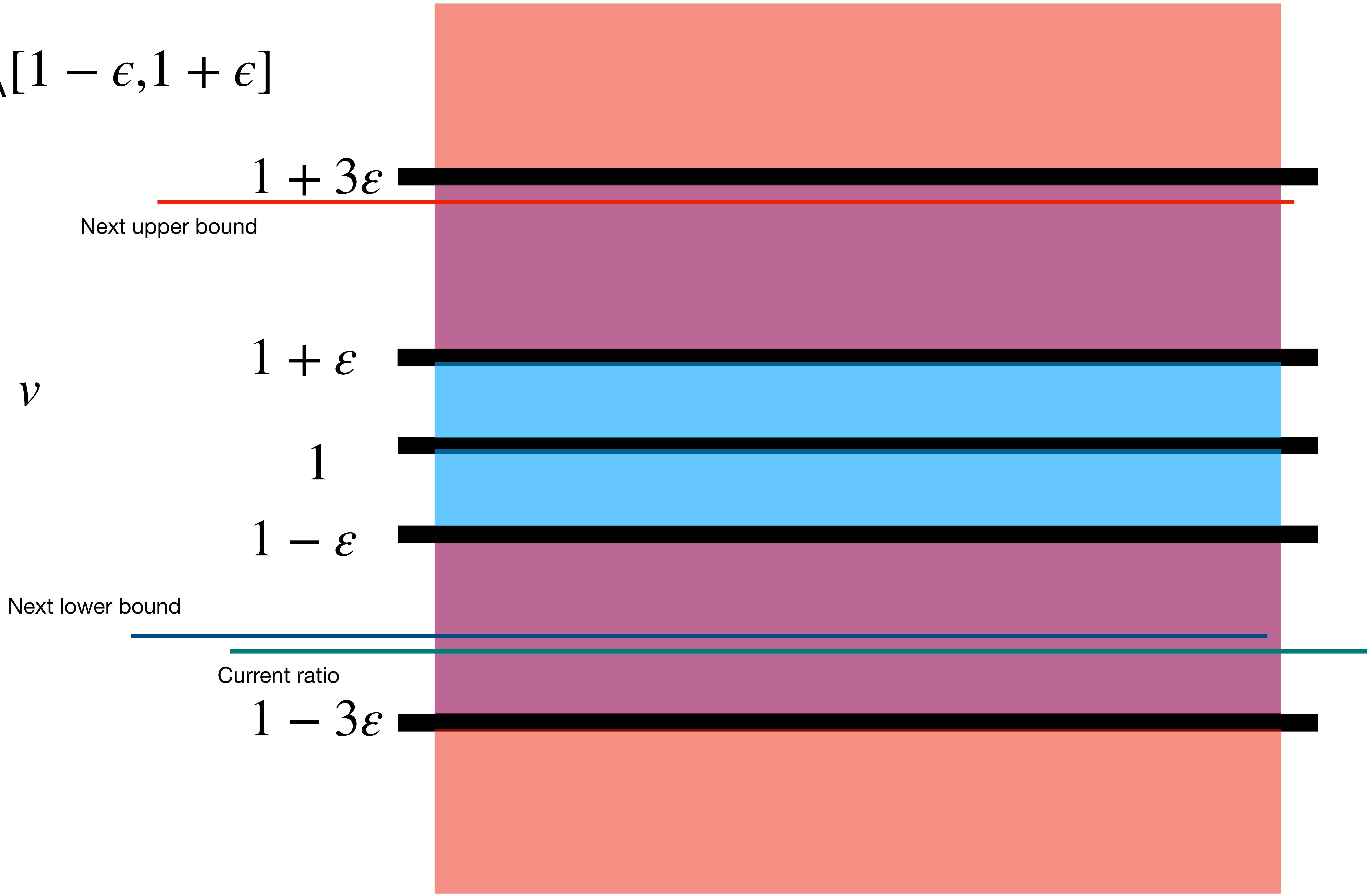
Case 2: Purple region

$\text{alloc}_a / C_a \in [1-3\epsilon, 1+3\epsilon] \backslash [1-\epsilon, 1+\epsilon]$



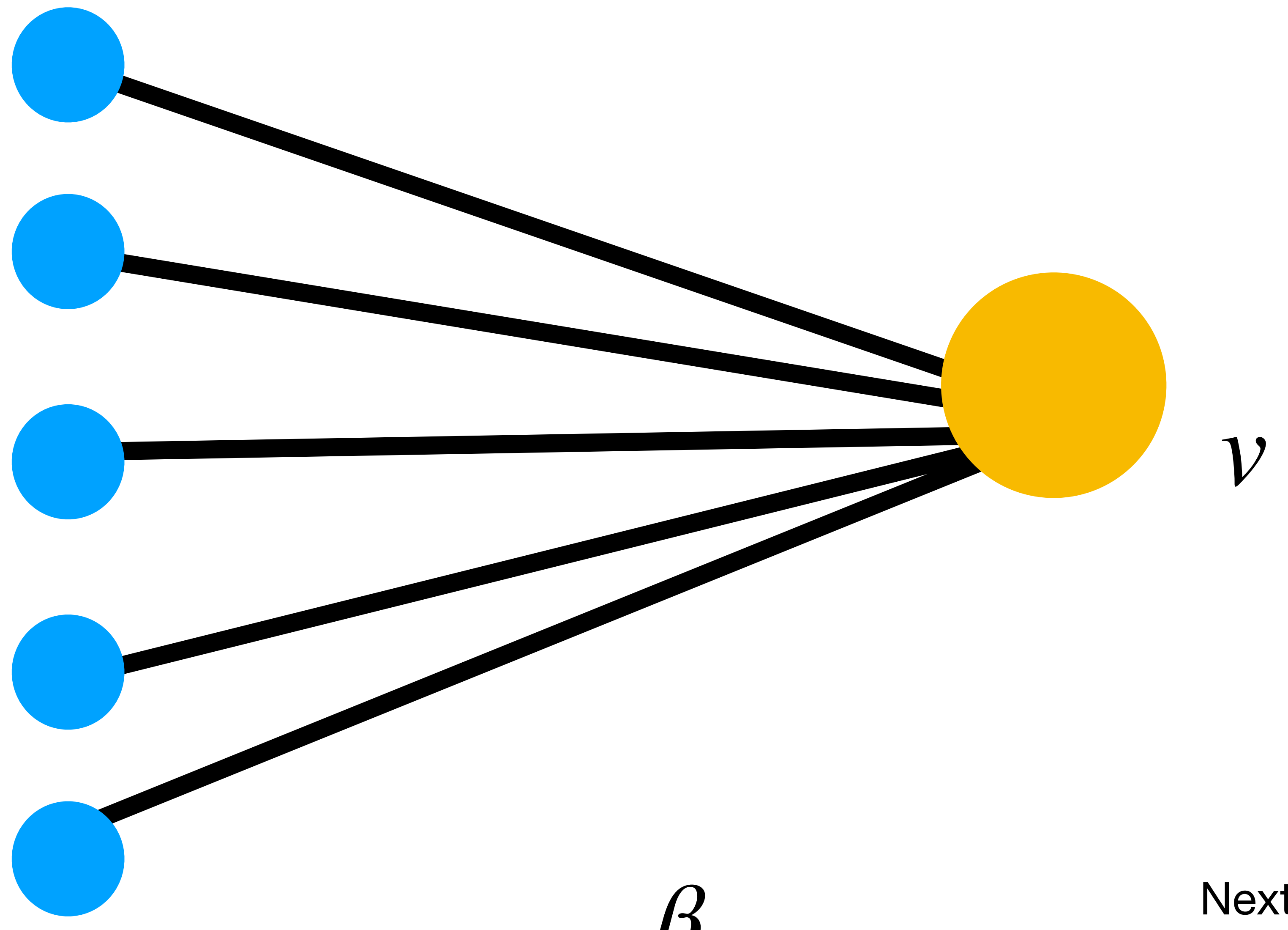
$$x_{u,v} = \frac{\beta_v}{\sum \beta_{v'}}$$

$$\text{alloc}_a = \sum x_{u',a}$$



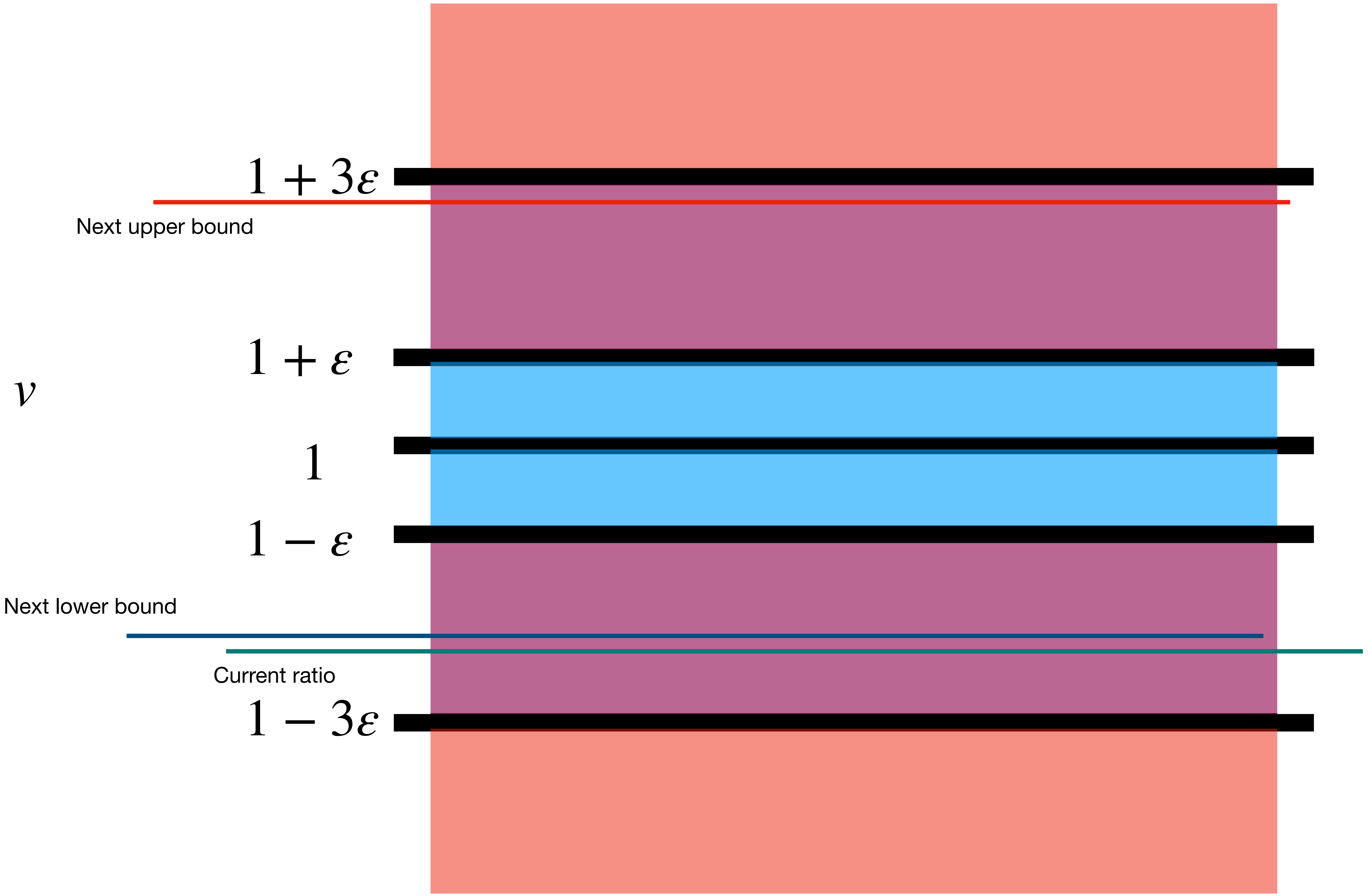
Proof of approximation

Case 2: Purple region
No escaping the good region!

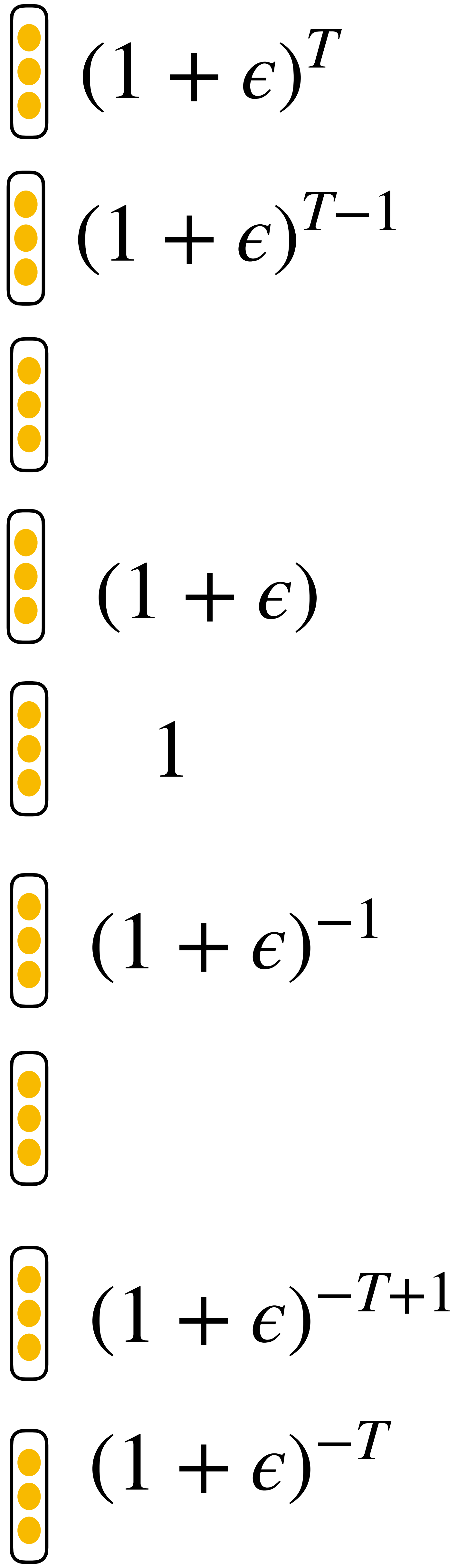


$$x_{u,v} = \frac{\beta_v}{\sum \beta_{v'}}$$

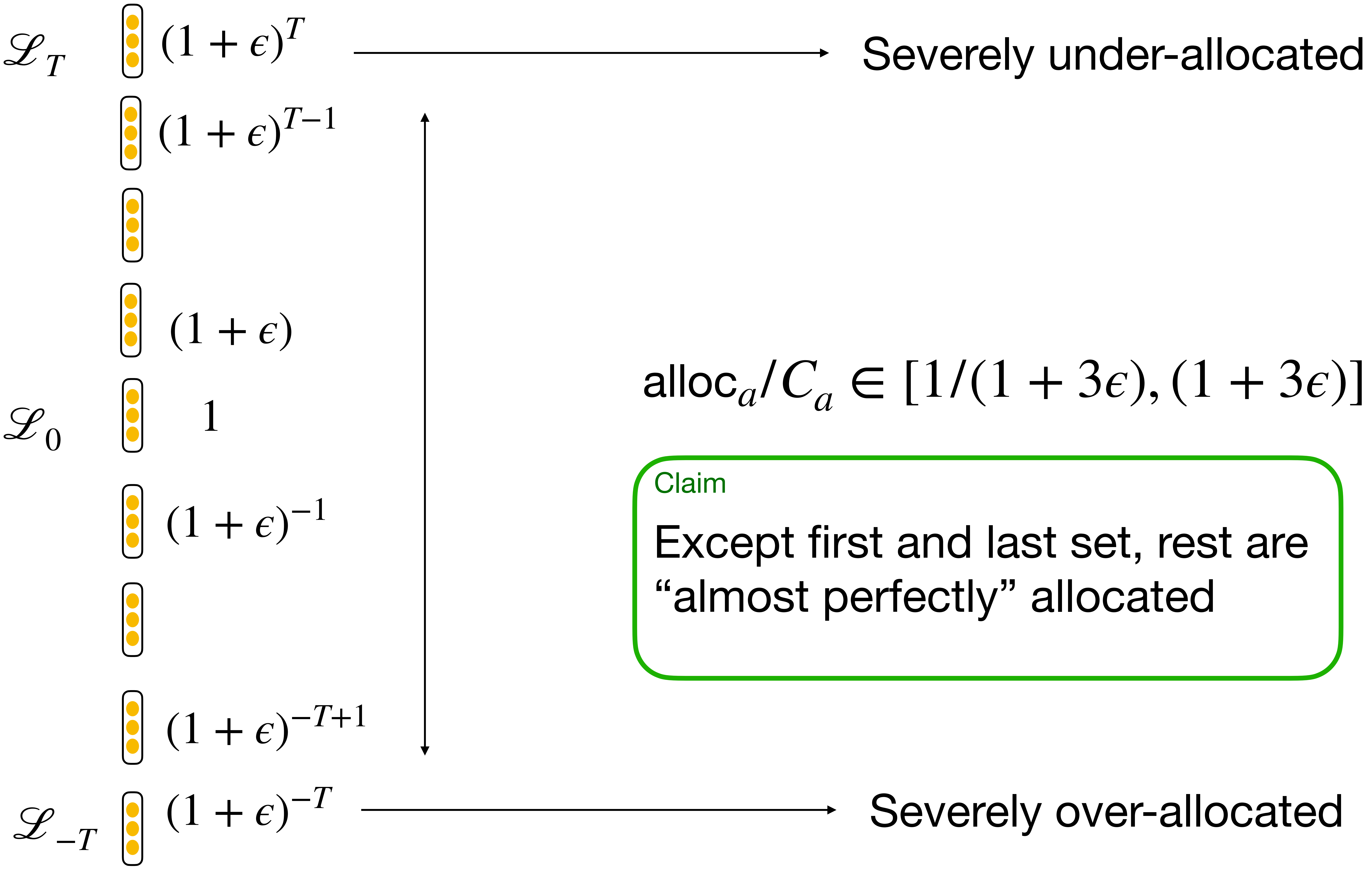
$$\text{alloc}_a = \sum x_{u',a}$$



Partition the advertisers according to their final priority values



Partition the advertisers according to their final priority values

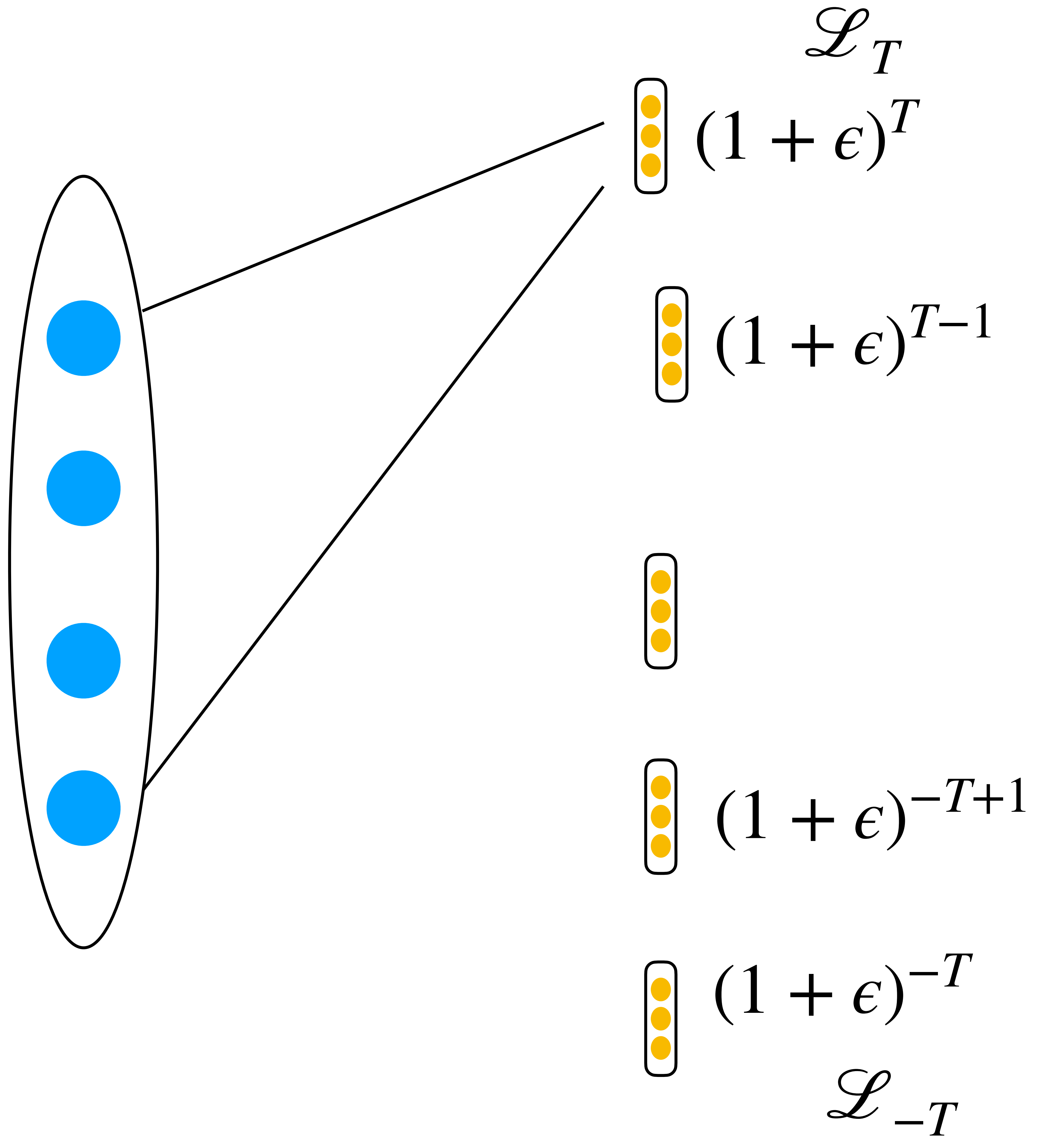


$\text{alloc}_a / C_a \in [1/(1+3\epsilon), (1+3\epsilon)]$

Claim

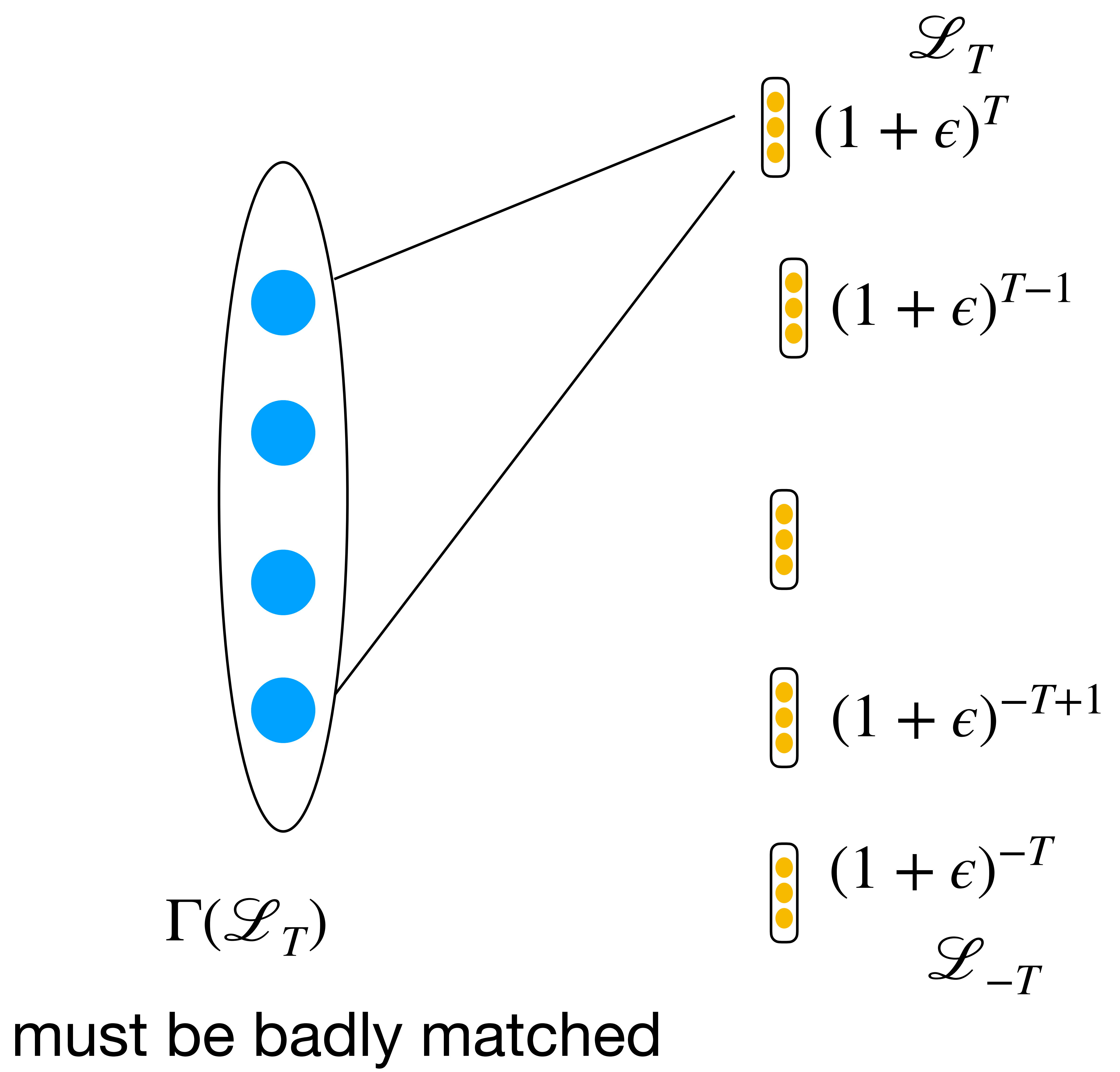
Except first and last set, rest are "almost perfectly" allocated

Analysis of approximation



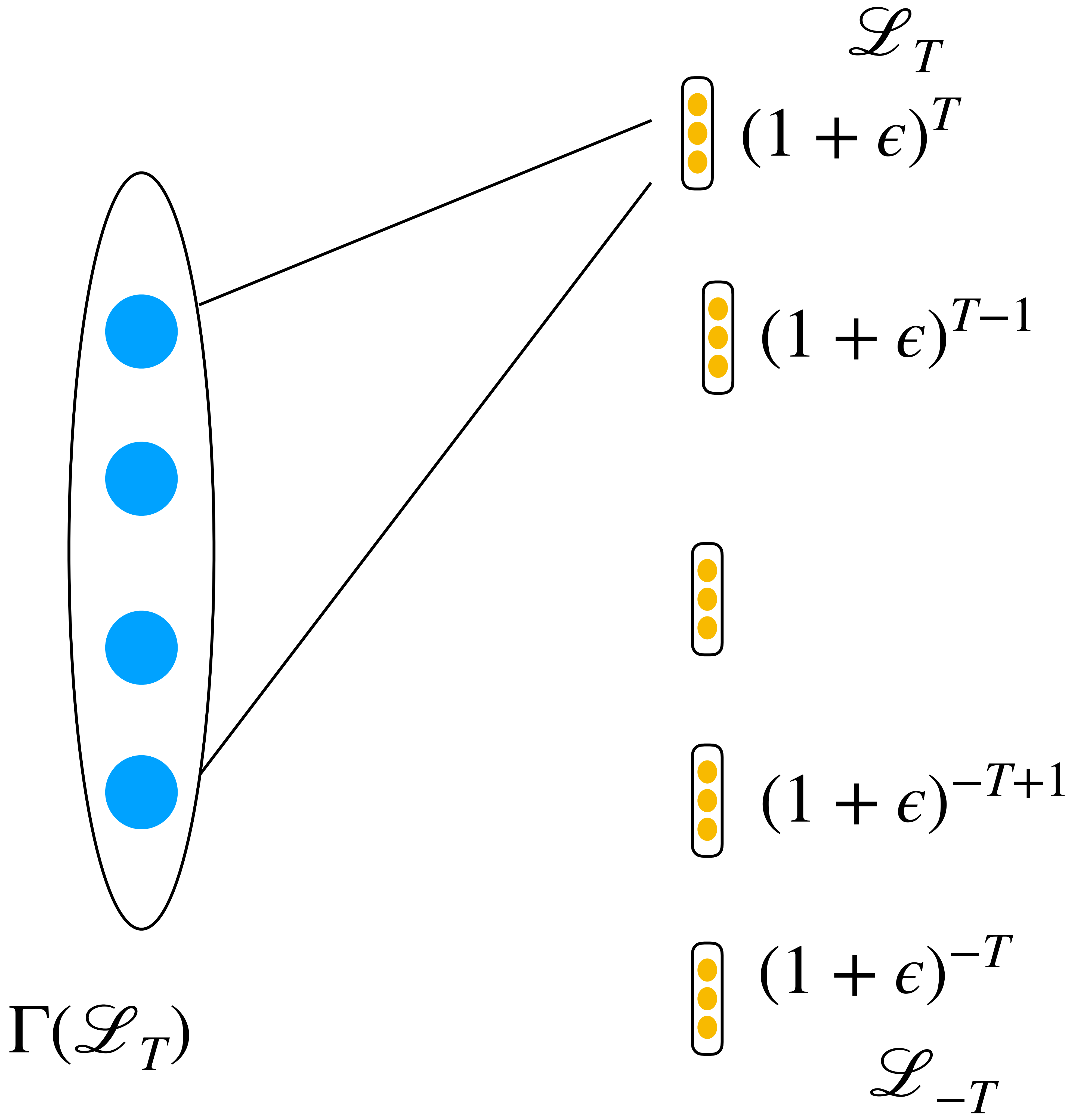
How could this matching be bad?

Analysis of approximation



How could this matching be bad?

Analysis of approximation

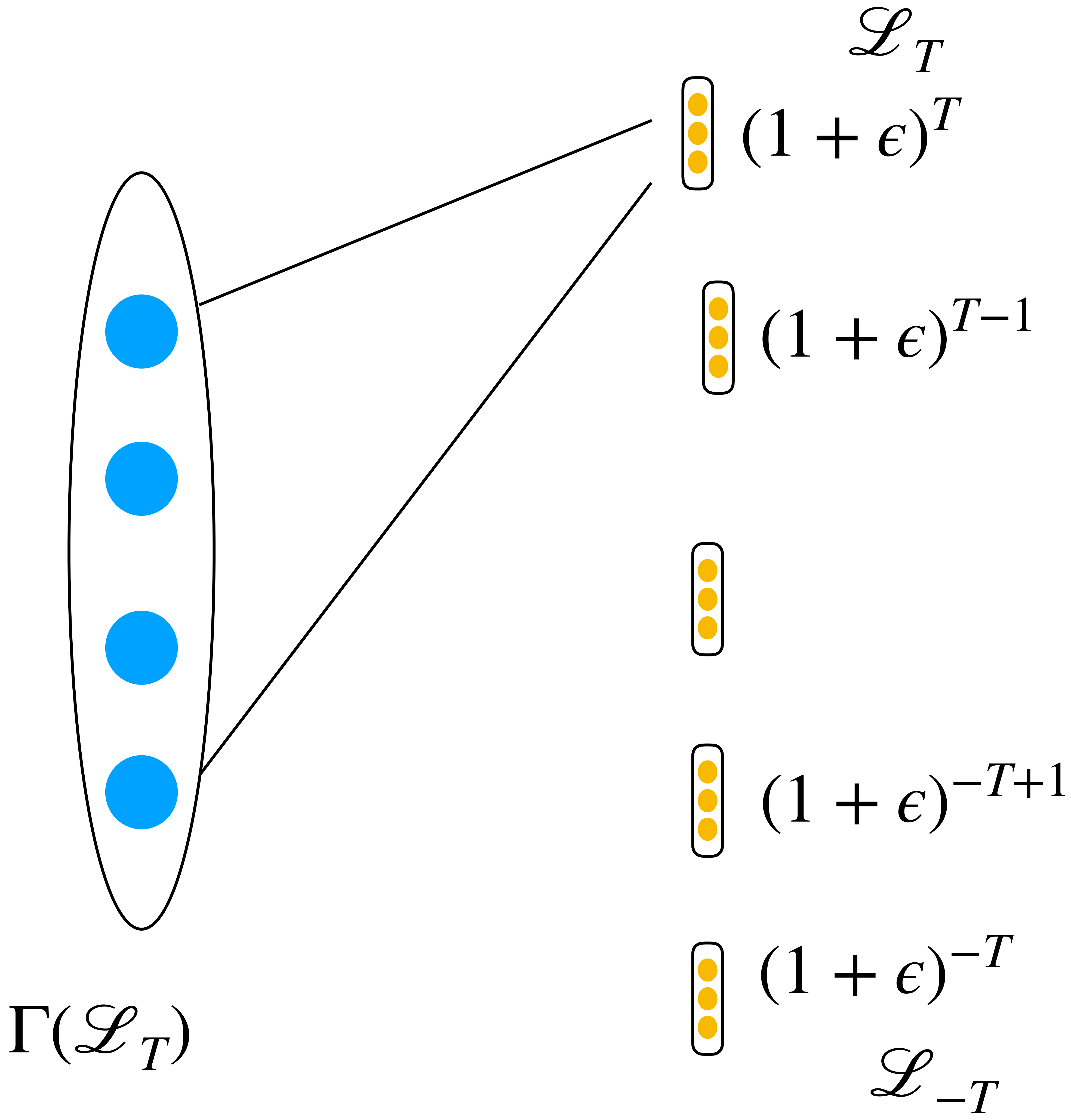


must be badly matched

How could this matching be bad?

$k = |\Gamma(\mathscr{L}_T)|$

Analysis of approximation



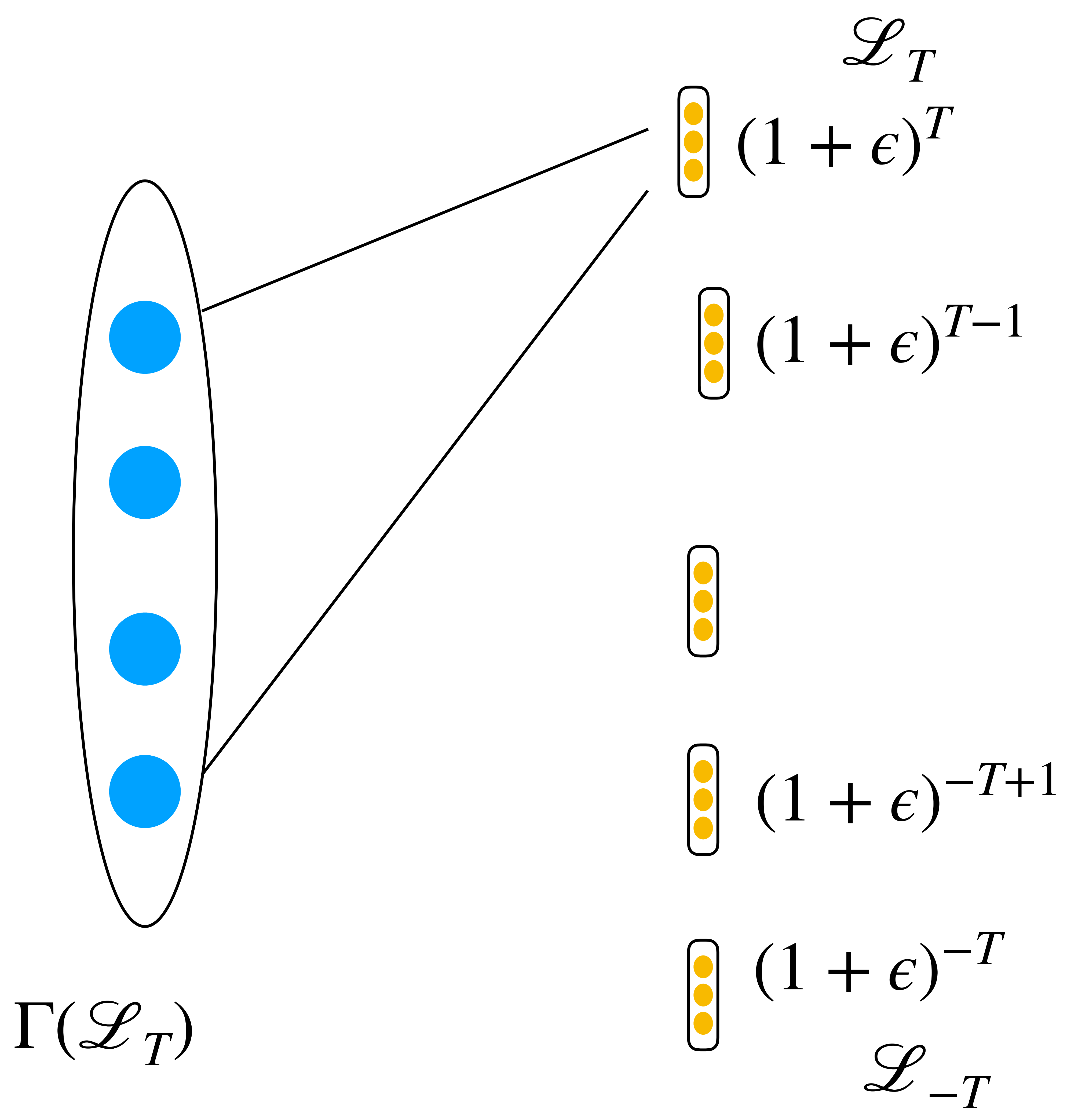
must be badly matched

How could this matching be bad?

$k = |\Gamma(\mathcal{L}_T)|$

GOT = matching of the algorithm

Analysis of approximation



must be badly matched

How could this matching be bad?

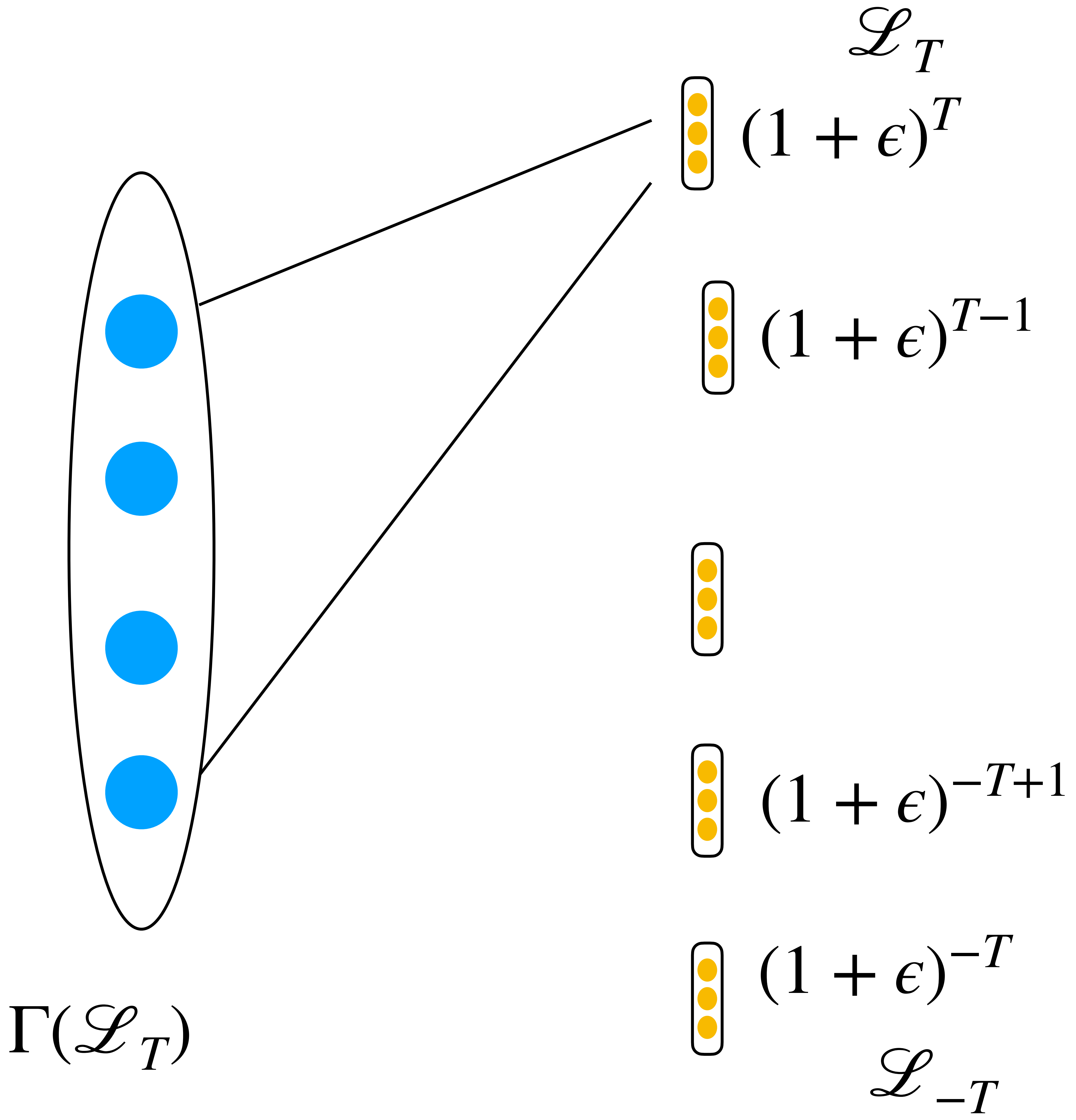
$k = |\Gamma(\mathscr{L}_T)|$

GOT = matching of the algorithm

Claim

GOT $\geq k(1-\epsilon)$ gets $2+O(\epsilon)$-approx

Analysis of approximation



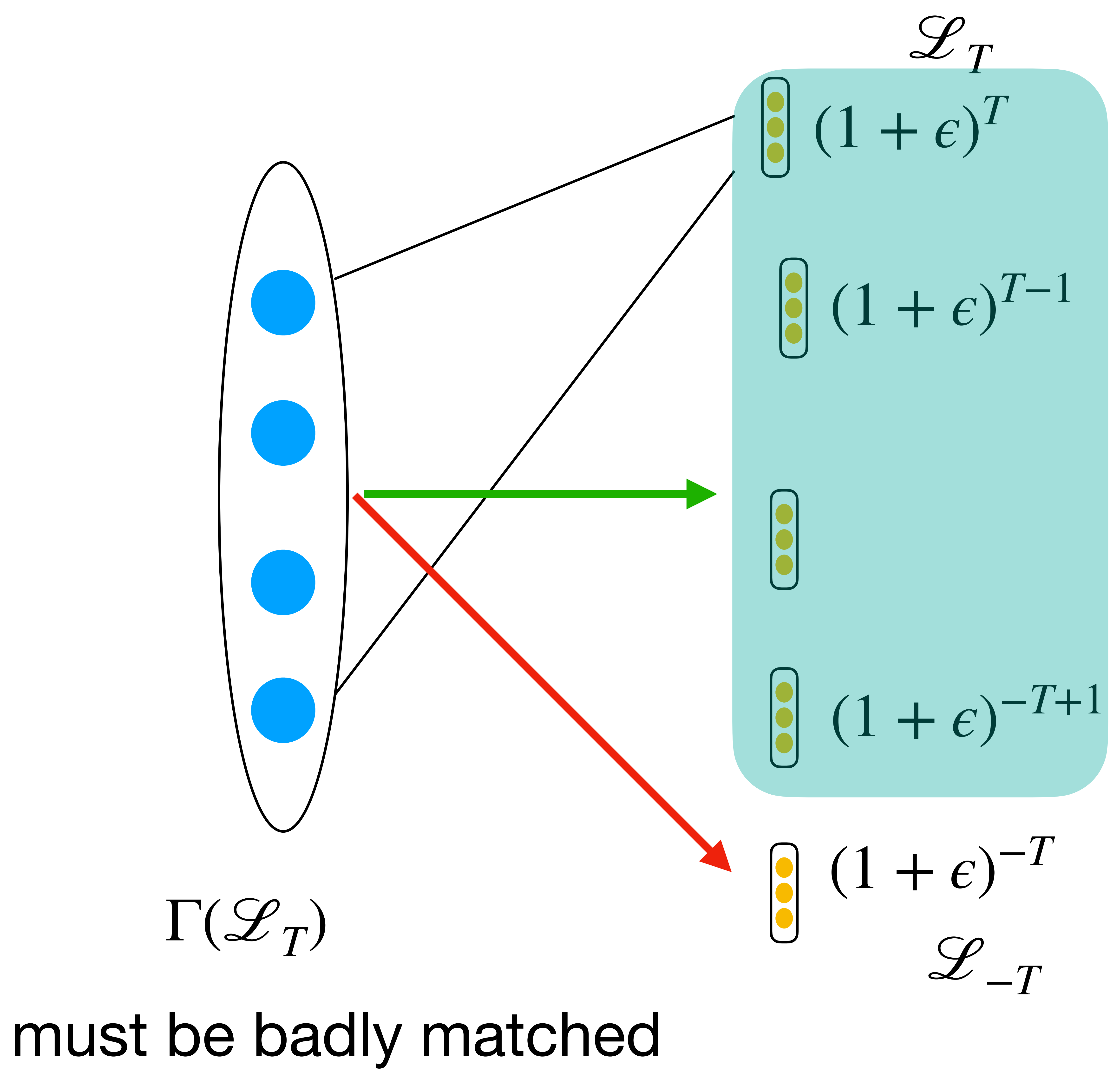
must be badly matched

Claim

GOT $\geq k(1-\epsilon)$ gets $2+O(\epsilon)$-approx

Where did our algorithm assign $\Gamma(\mathscr{L}_T)$?

Analysis of approximation

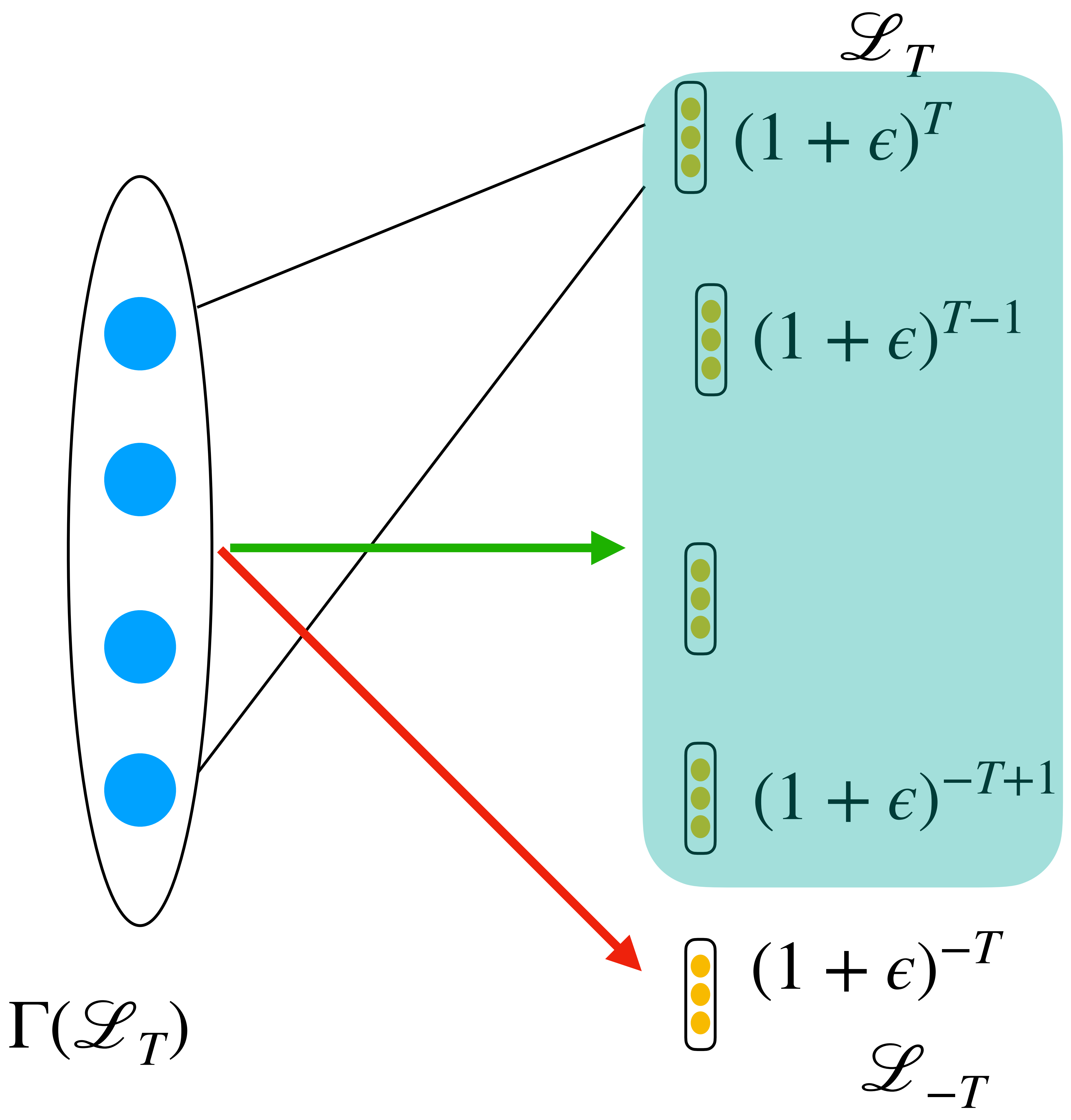


Claim

GOT $\geq k(1-\epsilon)$ gets $2+O(\epsilon)$-approx

Where did our algorithm assign $\Gamma(\mathscr{L}_T)$?

Analysis of approximation



must be badly matched

Claim

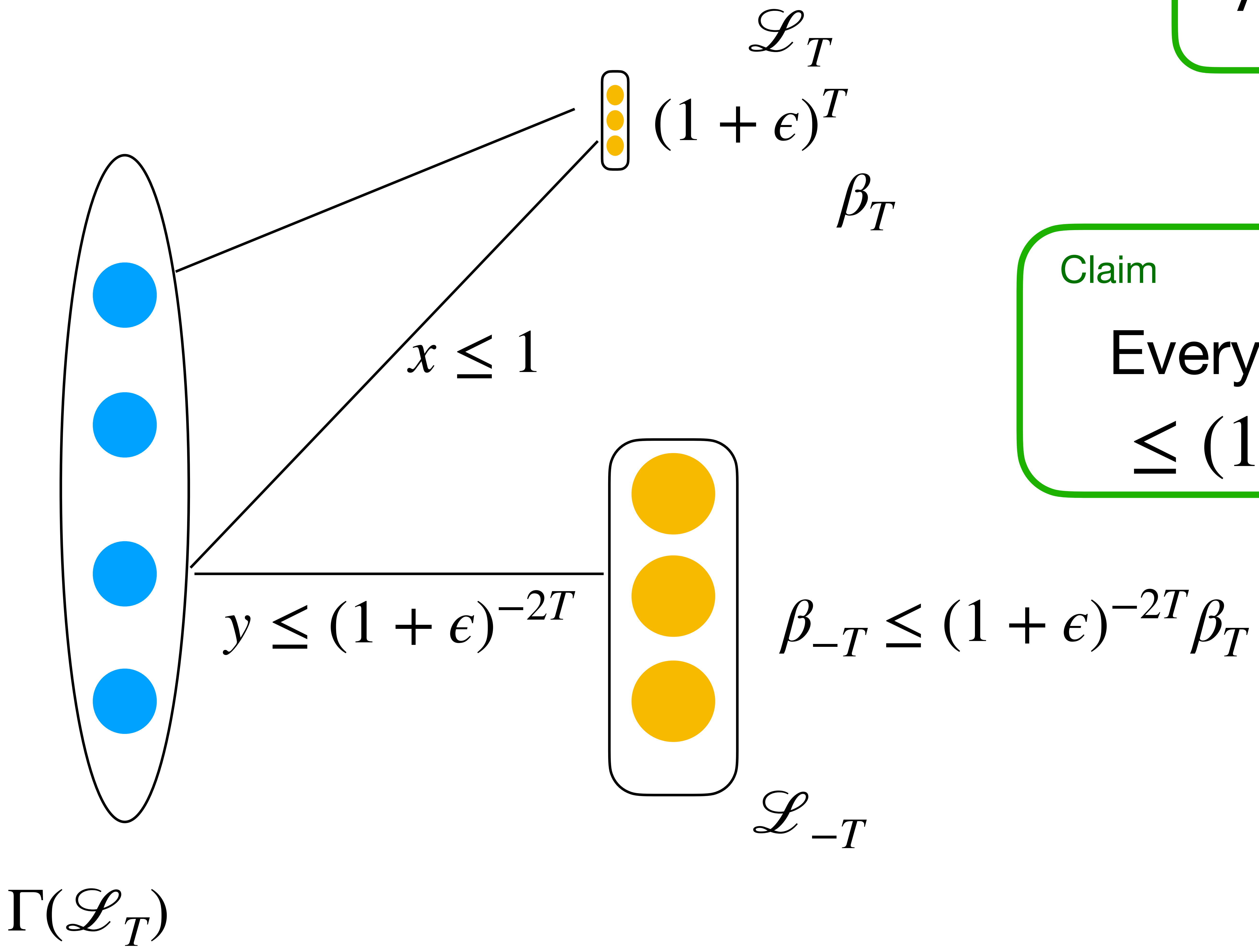
GOT $\geq k(1-\epsilon)$ gets $2+O(\epsilon)$-approx

Case 1: $|\mathscr{L}_{-T}| \geq k$

GOT $\geq k$ ✓

$\text{alloc}_v = C_v$ after rescaling

Bounding the optimum



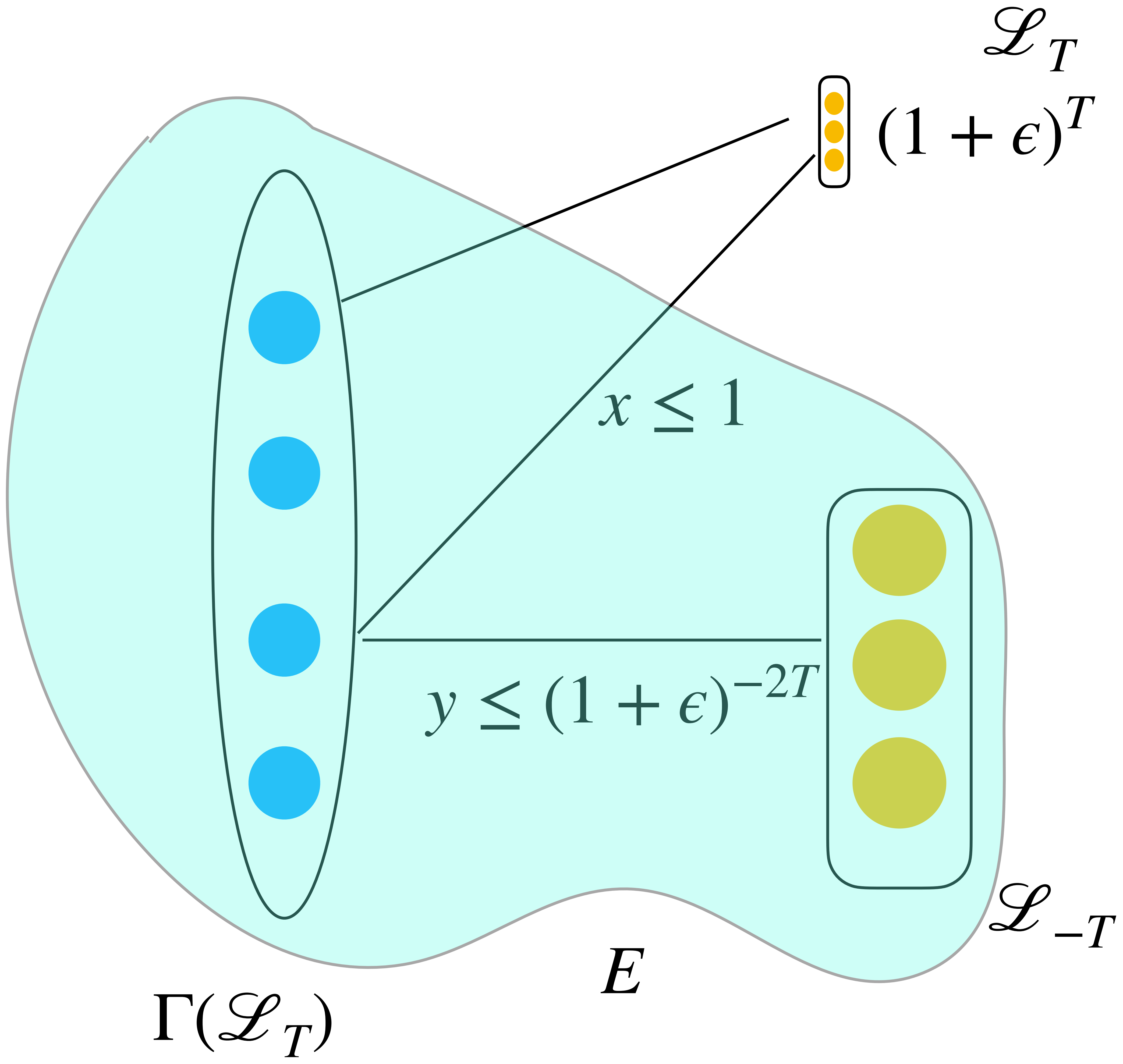
must be badly matched

Assume : $|\mathscr{L}_{-T}| \leq k$

Claim

Every edge to $\mathscr{L}_{-T}$ has weight $\leq (1+\epsilon)^{-2T}$

Bounding the optimum

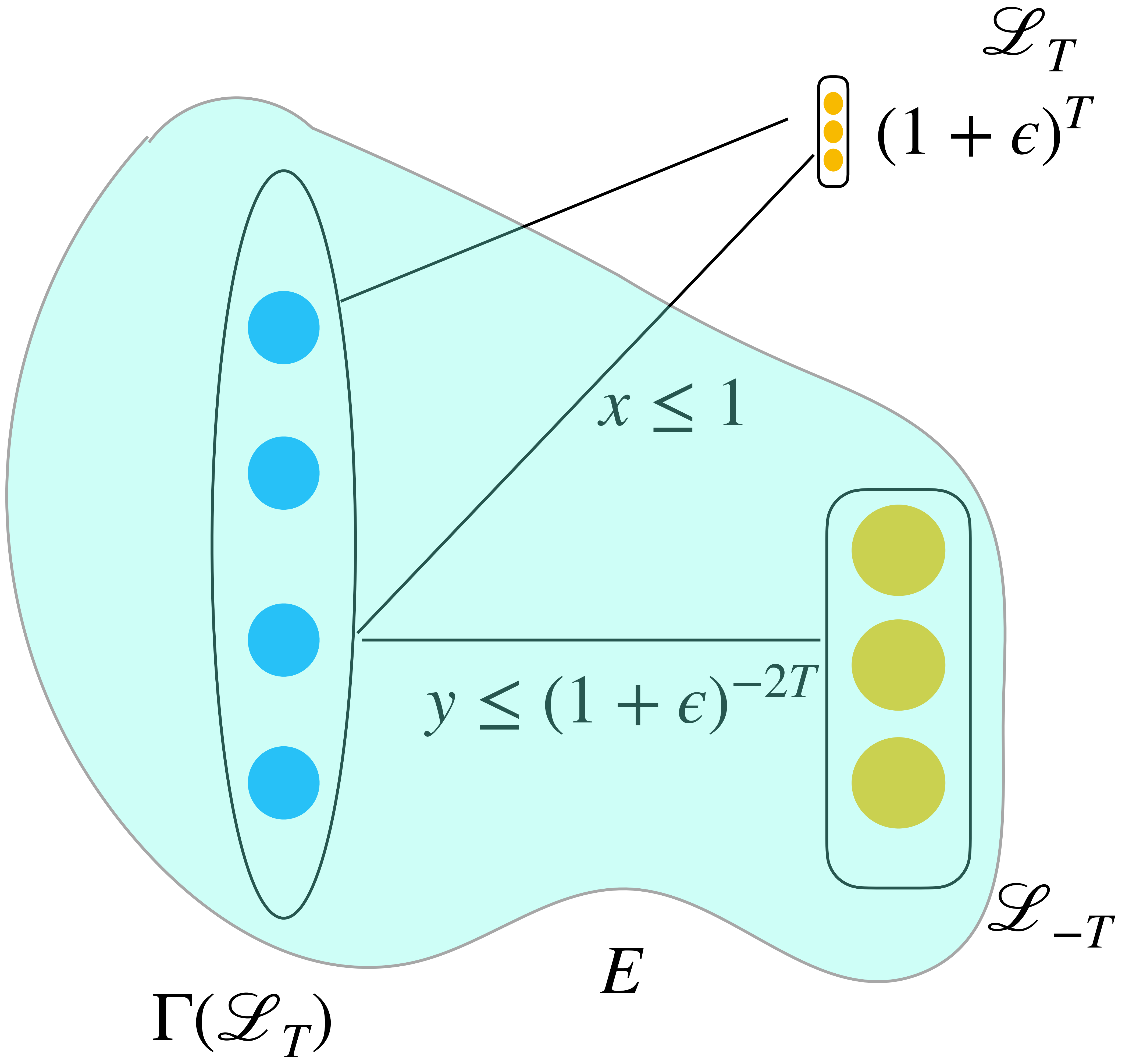


must be badly matched

Assume : $|\mathscr{L}_{-T}| \leq k$

Matching sent to $\mathscr{L}_{-T} \leq |E|(1+\epsilon)^{-2T}$

Bounding the optimum



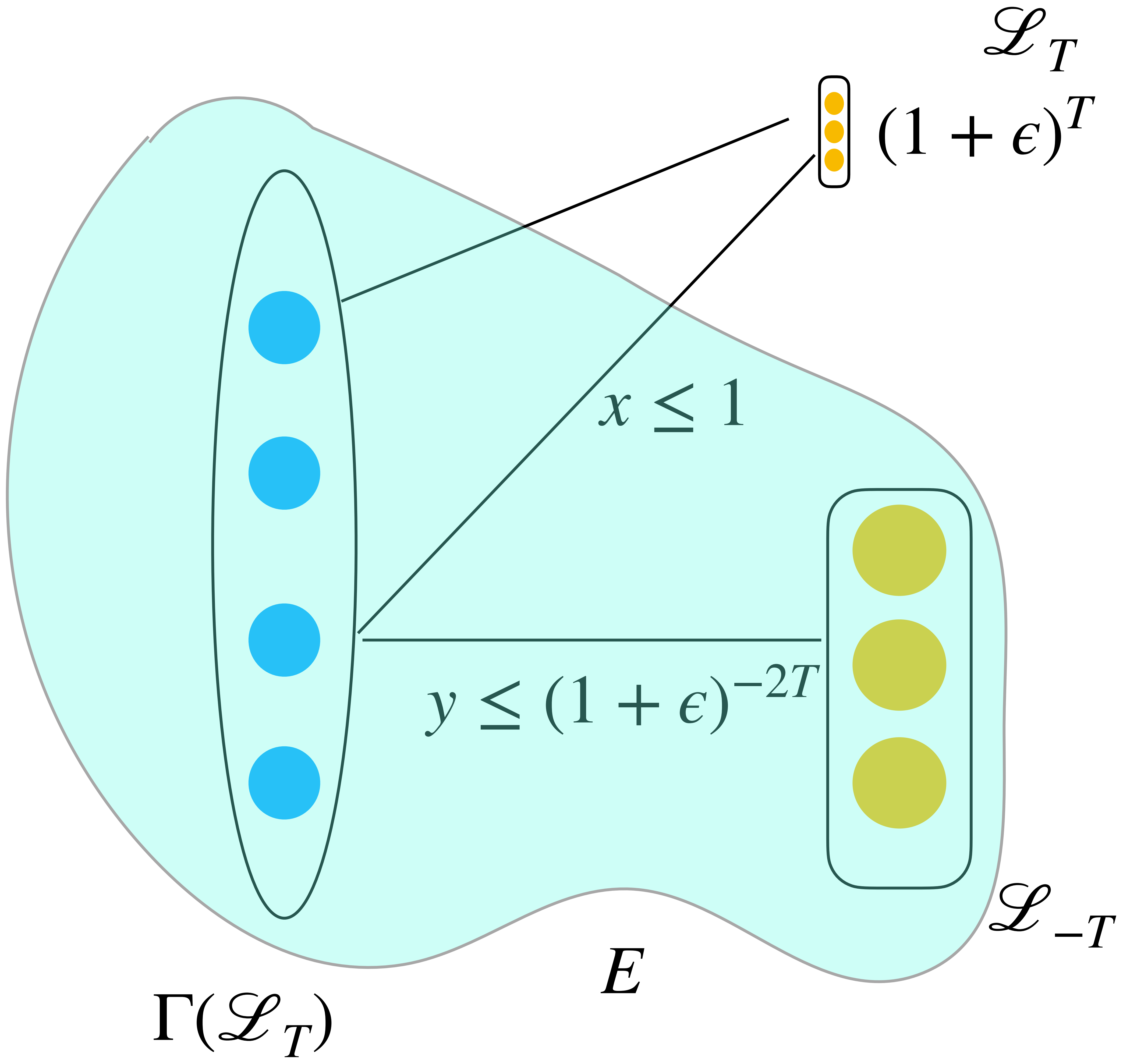
must be badly matched

Assume : $|\mathcal{L}_{-T}| \leq k$

Matching sent to $\mathcal{L}_{-T} \leq |E|(1+\epsilon)^{-2T}$

$$|E| \leq 4k\lambda$$

Bounding the optimum



must be badly matched

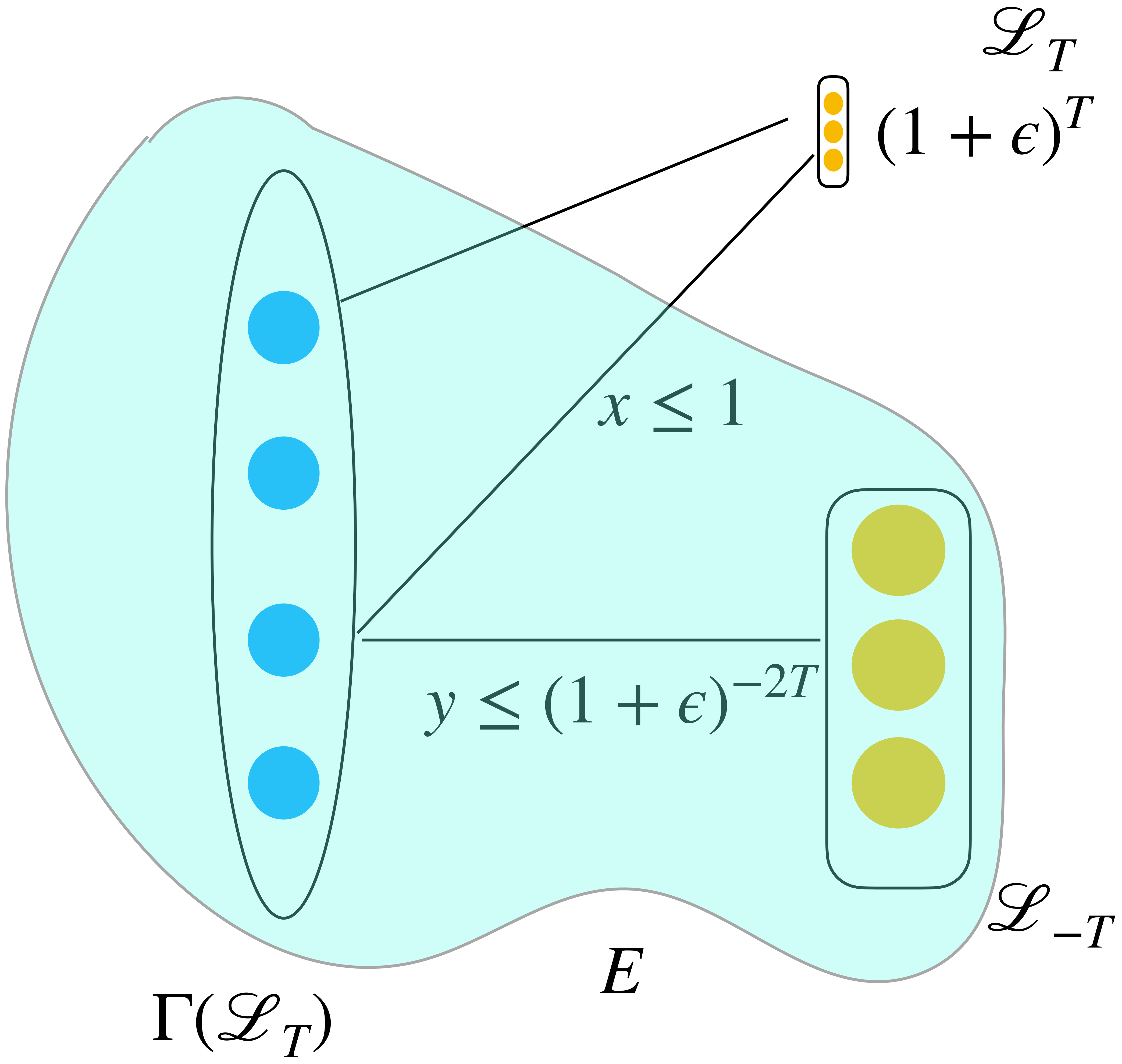
Assume : $|\mathscr{L}_{-T}| \leq k$

Matching sent to $\mathscr{L}_{-T} \leq |E|(1+\epsilon)^{-2T}$

$$|E| \leq 4k\lambda$$

Matching sent to $\mathscr{L}_{-T} \leq 4k\lambda(1+\epsilon)^{-2T}$

Bounding the optimum



must be badly matched

Assume : $|\mathscr{L}_{-T}| \leq k$

Matching sent to $\mathscr{L}_{-T} \leq |E|(1+\epsilon)^{-2T}$

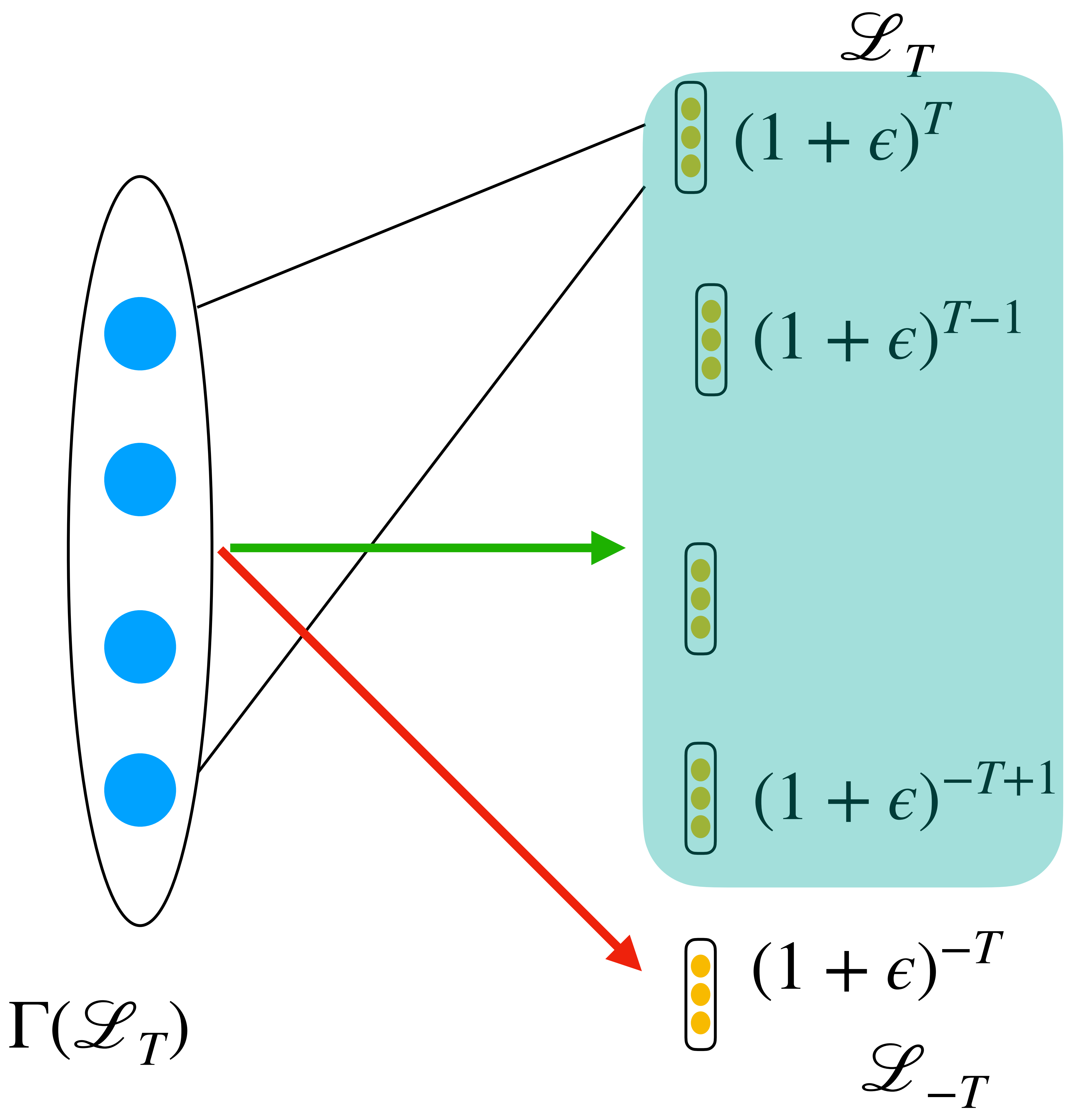
$$|E| \leq 4k\lambda$$

Matching sent to $\mathscr{L}_{-T} \leq 4k\lambda(1+\epsilon)^{-2T}$

$$\leq k\epsilon$$

$$T = O_\epsilon(1 + \log \lambda)$$

Analysis of approximation



must be badly matched

Claim

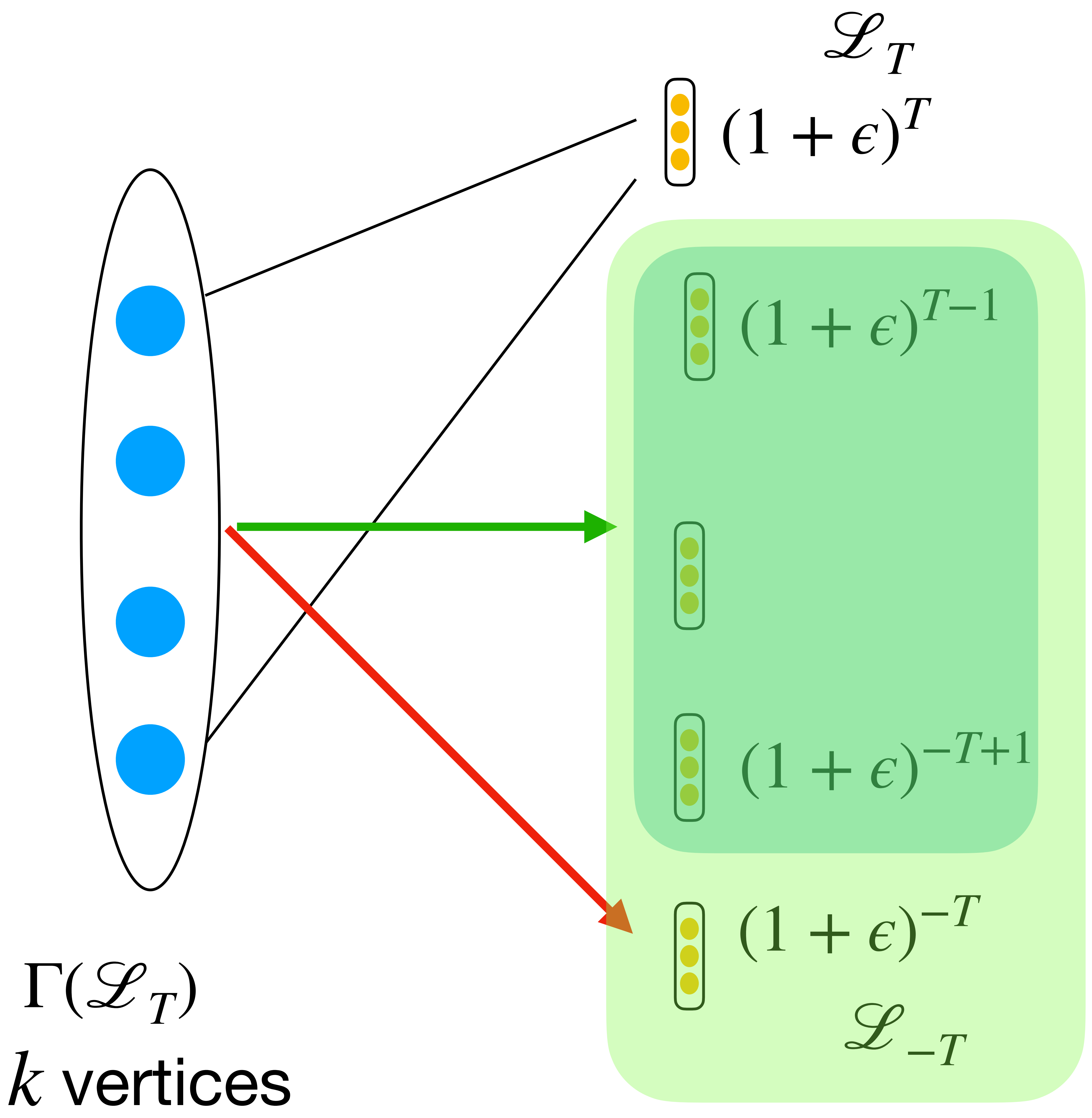
GOT $\geq k(1-\epsilon)$ gets $2+O(\epsilon)$-approx

Assume : $|\mathscr{L}_{-T}| \leq k$

Matching sent to $\mathscr{L}_{-T} \leq k\epsilon$

$$\text{GOT} \geq \frac{k(1-\epsilon)}{(1+3\epsilon)}$$

Bounding the optimum



TIGHT = Total capacity excluding $\mathscr{L}_T$

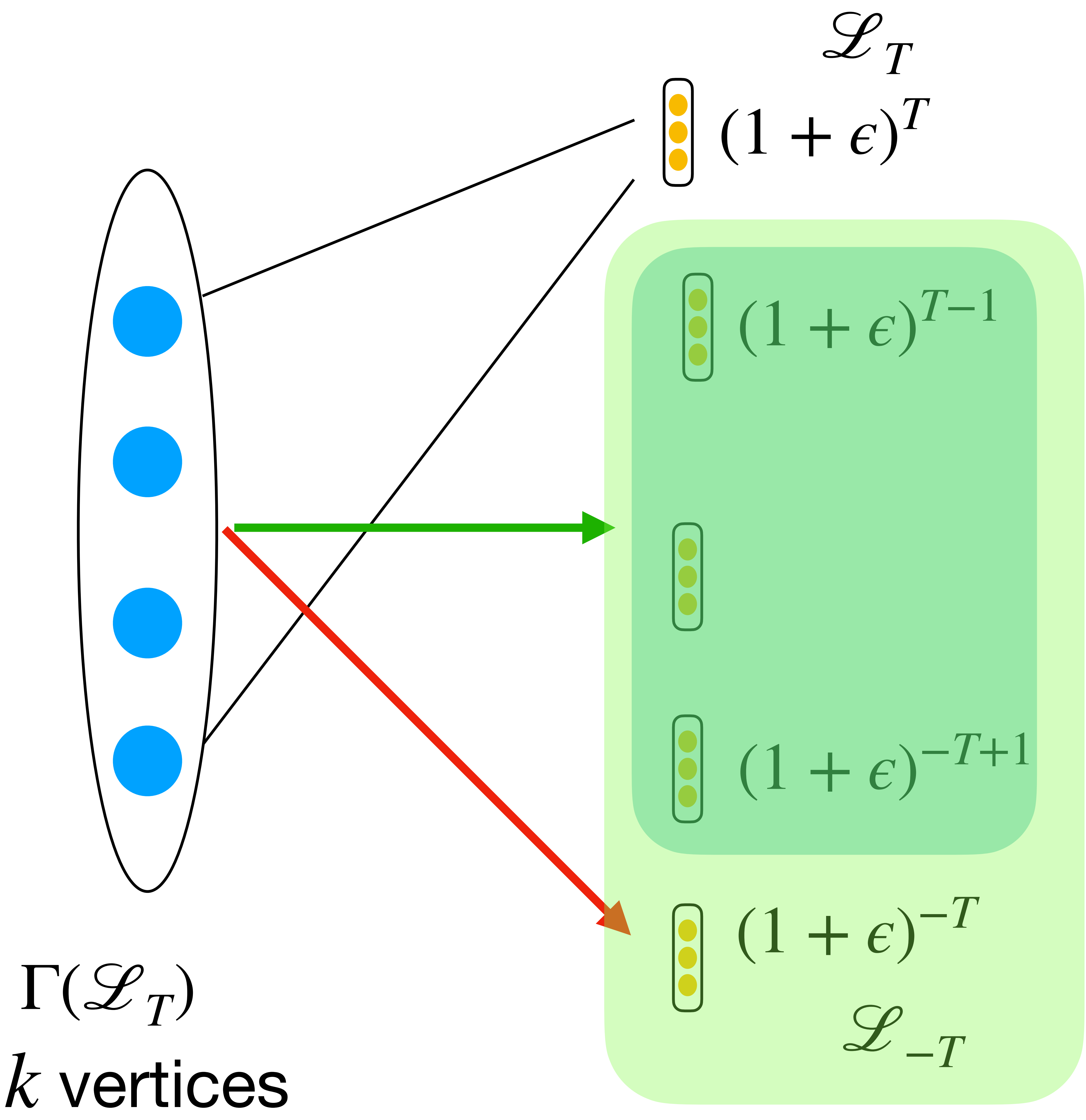
OPT $\leq$ TIGHT $+ k$

GOT $\geq$ TIGHT$/(1+3\epsilon)$

2GOT $\geq$ (TIGHT $+ k)/(1+O(\epsilon))$

GOT $\geq$ OPT$/(2+O(\epsilon))$

Bounding the optimum



TIGHT = Total capacity excluding $\mathscr{L}_T$

OPT $\leq$ TIGHT $+ k$

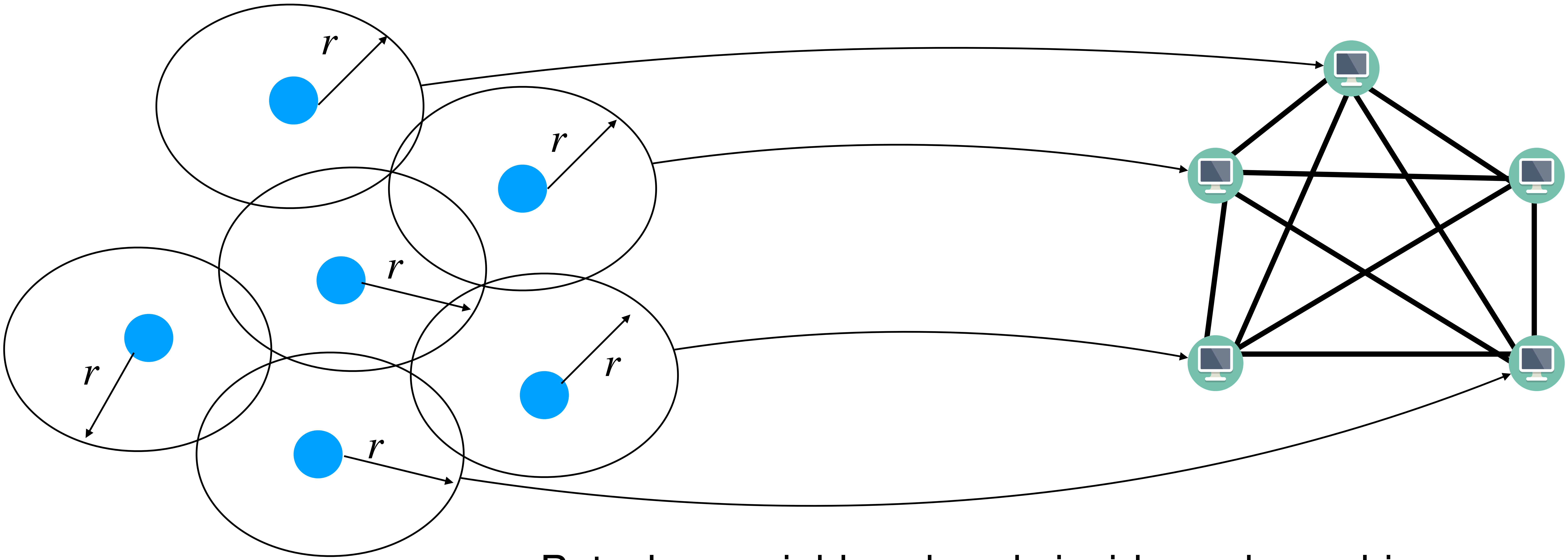
GOT $\geq$ TIGHT$/(1+3\epsilon)$

2GOT $\geq$ (TIGHT $+ k)/(1+O(\epsilon))$

GOT $\geq$ OPT$/(2+O(\epsilon))$

Can be boosted to $1+\epsilon$ using [GGM18]

Simulation in MPC



Put r hop-neighbourhoods inside each machine

r rounds of LOCAL can be simulated!

Size > n ?

Random Thresholding

Approximation argument perspective

if $\text{alloc}_v < C_v/(1+\epsilon)$

increase β_v by $1+\epsilon$ factor

Random Thresholding

Approximation argument perspective

if alloc$_v < C_v/(1+\epsilon)$

increase β_v by $1+\epsilon$ factor

Does this need to be strictly ϵ?

$$\epsilon \leftarrow [\epsilon/2, \epsilon]$$

Does it need to be same for all vertices and rounds?

$$\epsilon_{v,t} \leftarrow [\epsilon/2, \epsilon]$$

Random Thresholding

Approximation argument perspective

if $\text{alloc}_v < C_v/(1+\epsilon)$

increase β_v by $1+\epsilon$ factor

Does this need to be strictly ϵ?

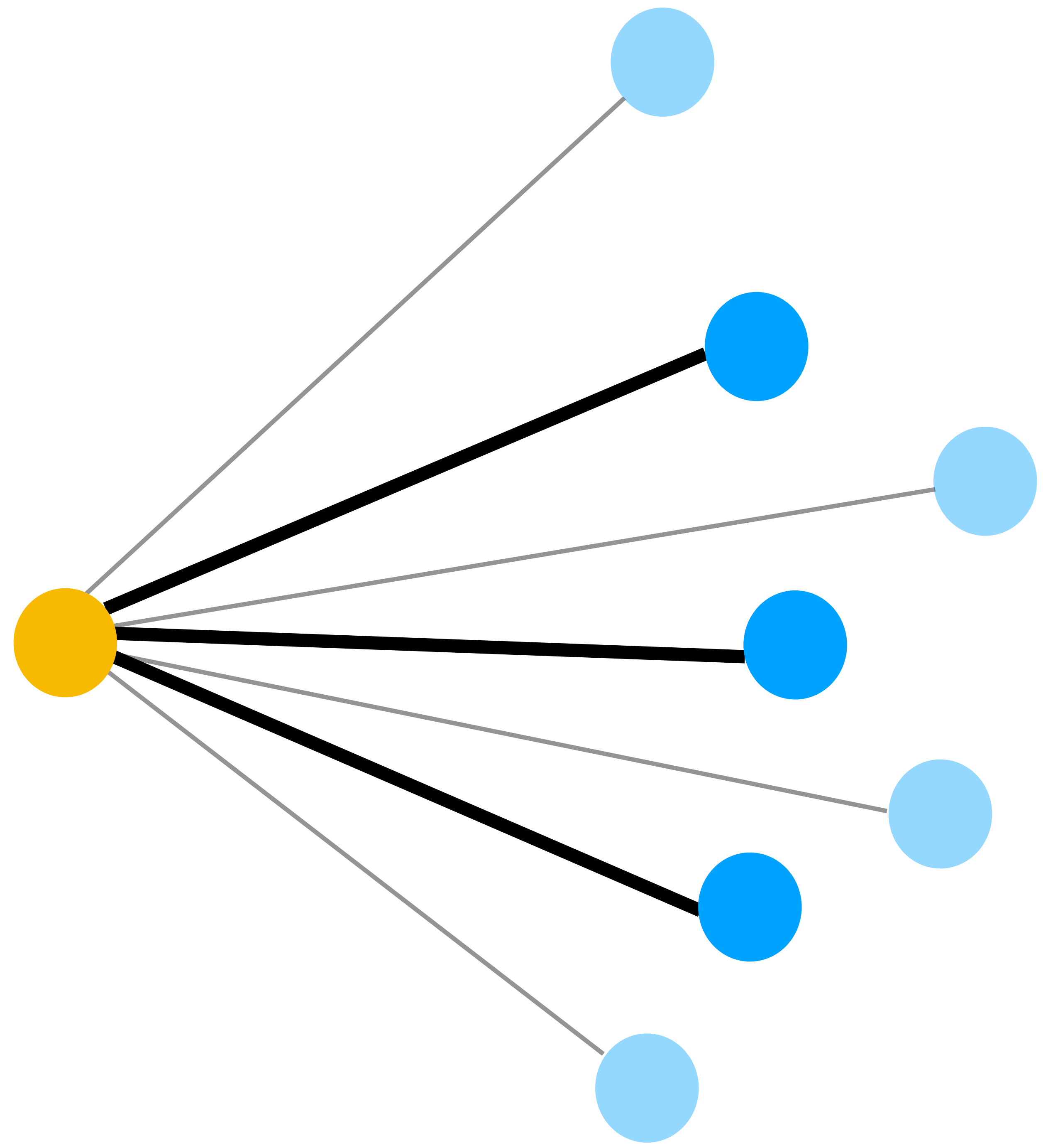
$\epsilon \leftarrow [\epsilon/2, \epsilon]$

Does it need to be same for all vertices and rounds?

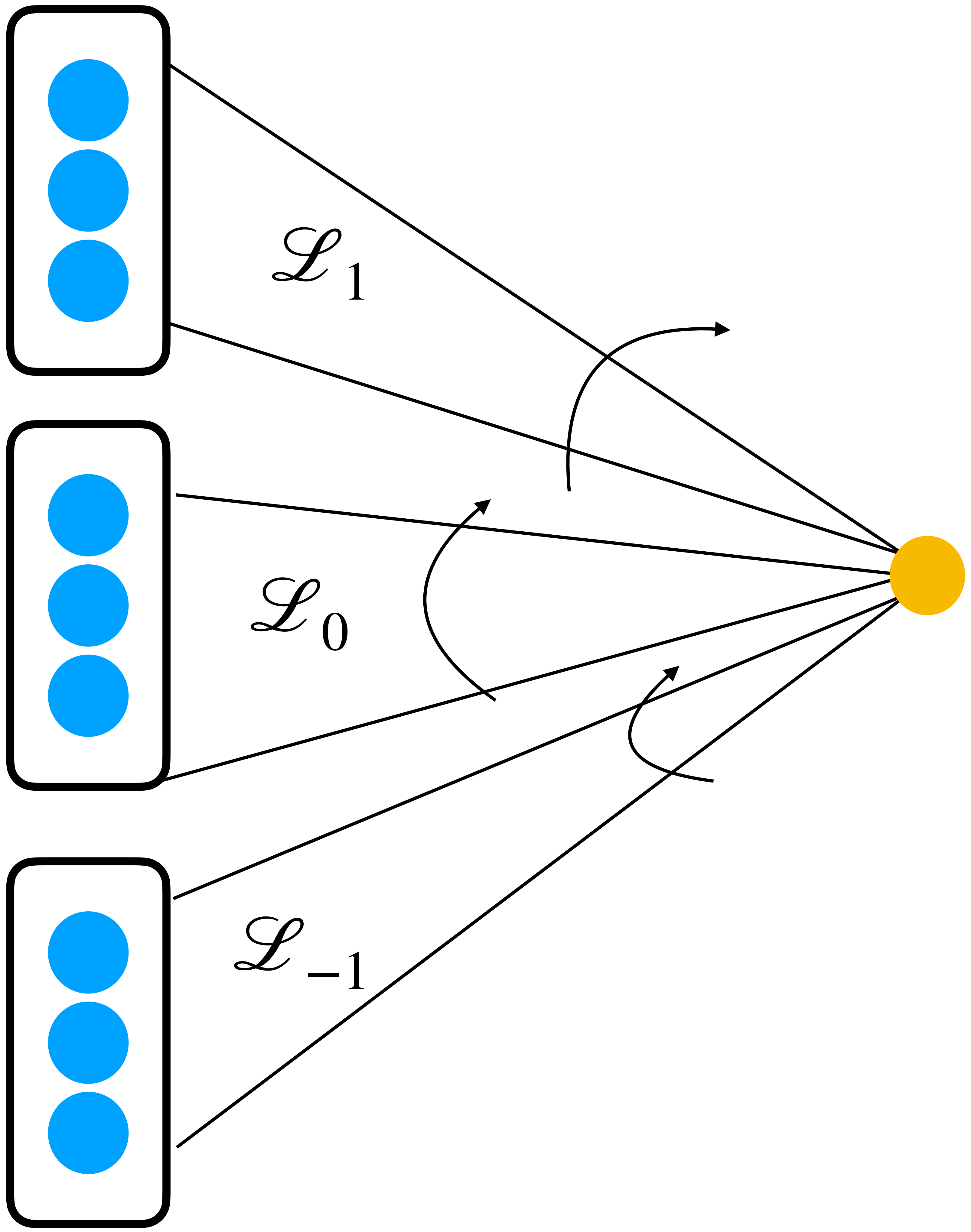
$\epsilon_{v,t} \leftarrow [\epsilon/2, \epsilon]$

Algorithm design perspective

$\hat{\text{alloc}}_v \leftarrow 1+\epsilon$ approx of alloc_v



Bucketing + uniform sampling does the trick



$$\beta_u = \sum x_{u,v}$$

β_u changes by factor $(1+\epsilon)$

$$\mathscr{L}_i = \{v \mid \beta_v \in [(1+\epsilon)^i, (1+\epsilon)^{i-1})\}$$

Uniformly sampling from $\mathscr{L}_v$ is enough!

r – hop neighbourhood has size $2^{O(r^2)}$

Open Problems

Open Problem 1

Can our results be extended to $b-$matching problem?

Open Problem 2

Can we get dependence on average degree instead?