Allocating Goods to Maximize Fairness

Deeparnab Chakrabarty
U. of Waterloo

Julia Chuzhoy
TTI-C

Sanjeev Khanna
U. of Pennsylvania

Max Min Allocation

Input:

- Set ***A*** of m agents
- Set ***I*** of n items
- Utilities $u_{A,i}$ of agent A for item i.

Output: assignment of items to agents.

- Utility of agent A: $\sum u_{A,i}$ for items i assigned to A.

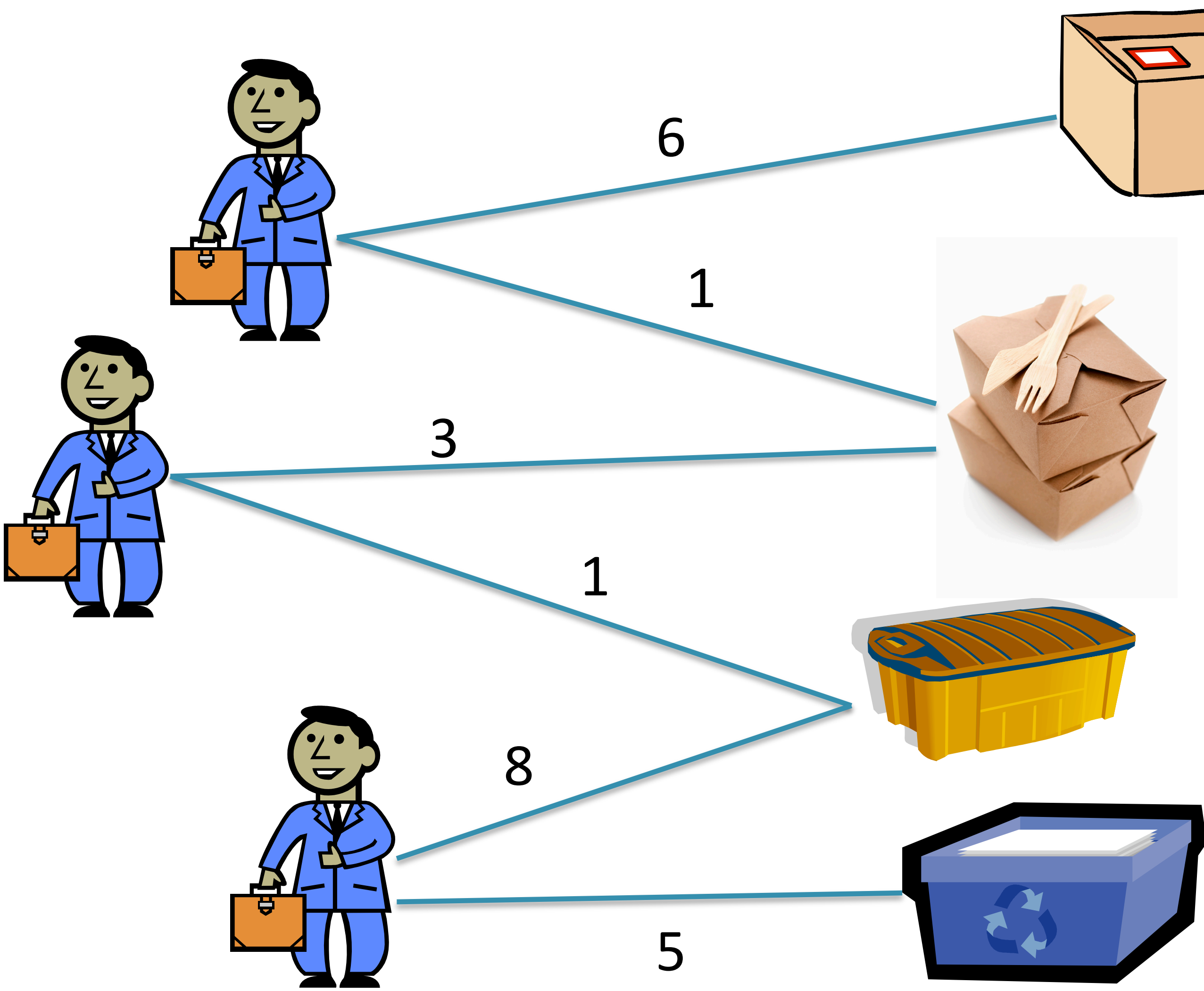
Goal: Maximize minimum utility of any agent.

Notation

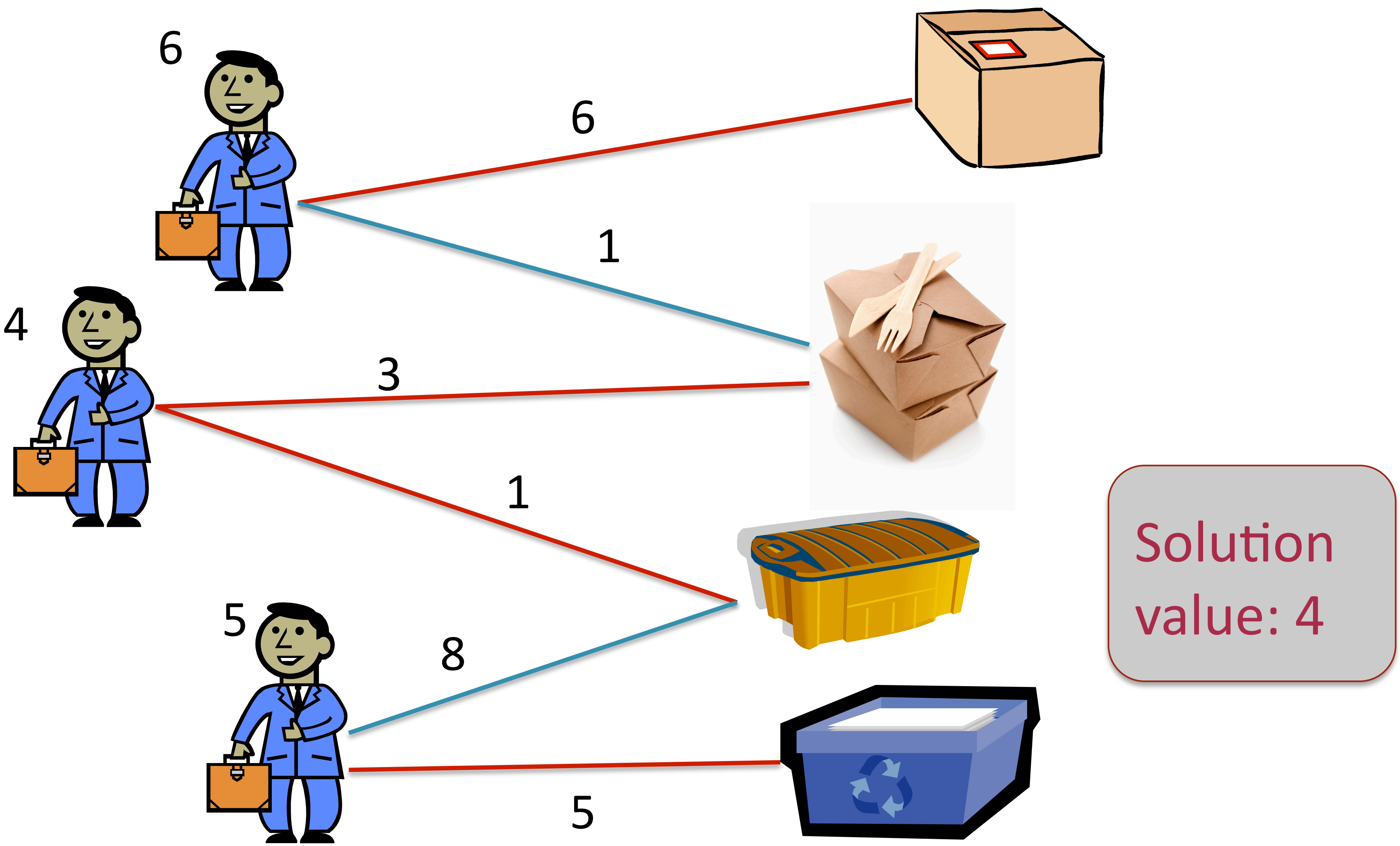
n - number of items

m - number of agents

Example



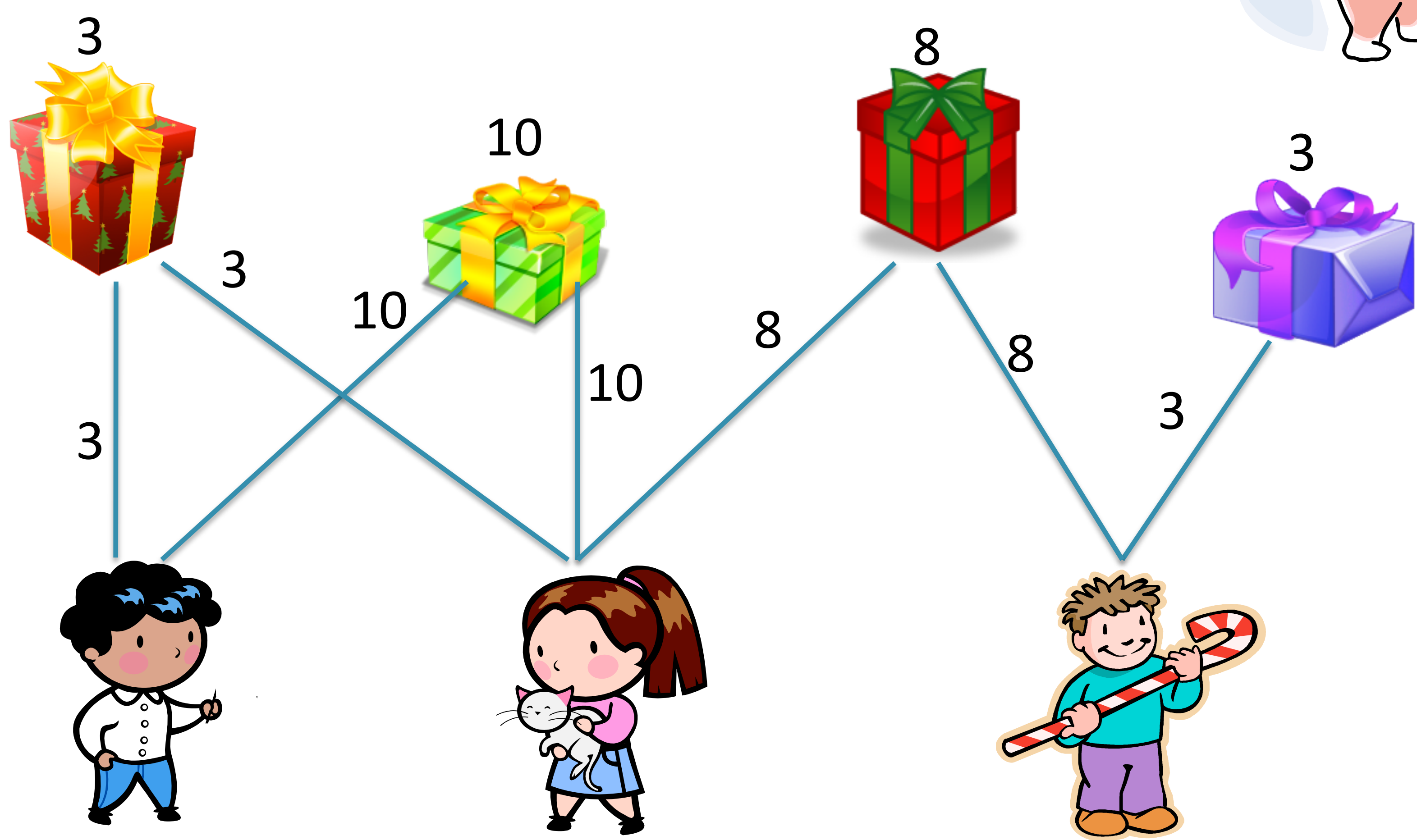
Example



Max-Min Allocation

- Captures a natural notion of fairness in allocation of indivisible goods.
- Approximation is still poorly understood.
- An interesting special case: Santa Claus problem.

The Santa Claus Problem



All edges adjacent to an item have identical utility

Santa Claus: Known Results

- Natural LP has $\Omega(m)$ integrality gap.
- [Bansal, Sviridenko '06]:
  - Introduced a new configuration LP
  - O(log log m/logloglog m)-approximation algorithm
- Non-constructive constant upper bounds on integrality gap of the LP [Feige '08], [Asadpour, Feige, Saberi '08].

Bad news: Configuration LP has $\Omega(\sqrt{m})$ integrality gap for Max-Min Allocation [Bansal, Sviridenko '06].

Known Results for Max Min Allocation

- (n-m+1)-approximation [Bezakova, Dani '05].
- $\tilde{O}(\sqrt{m})$-approximation via the configuration LP [Asadpour, Saberi '07].
- Configuration LP has $\Omega(\sqrt{m})$ integrality gap [Bansal, Sviridenko '06].
- Best current hardness of approximation factor: 2 [Bezakova, Dani '05]
  - Valid even in very restricted settings

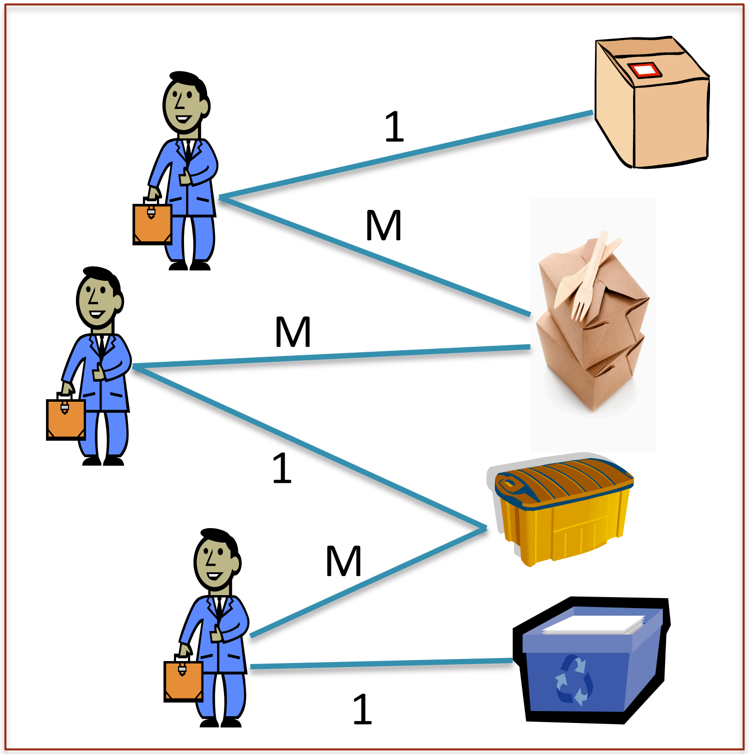
Our Main Result

- $\tilde{O}\left(n^{\epsilon}\right)$-approximation algorithm in time $n^{O(1/\epsilon)}$
  - Poly-logarithmic approximation in quasi-polynomial time.
  - n^{ϵ}-approximation in poly-time for any constant ϵ.
- We use an LP with $\Omega(\sqrt{m})$ integrality gap as a building block.

Independent Work

[Bateni, Charikar, Guruswami '09] obtained similar results for special cases of the problem:

- All utilities are in {0, 1, M}, where OPT=M.
- In the graph induced by utility-M edges:
  - All items have degree at most 2, or
  - Graph contains no cycles
- An $\tilde{O}\left(n^{\epsilon}\right)$-approximation in time $n^{O(1/\epsilon)}$ for these cases



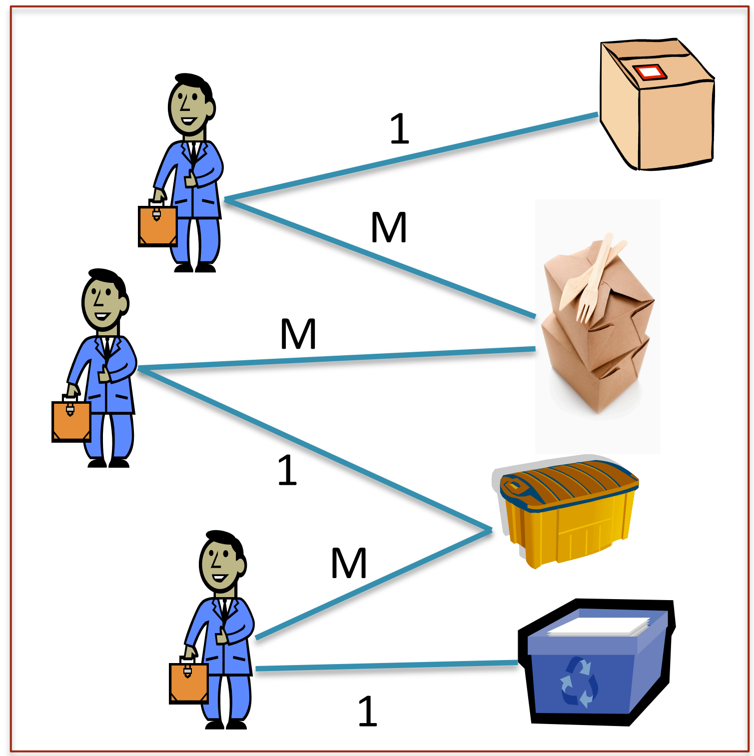
Independent Work

[Bateni, Charikar, Guruswam ... r

results for special case

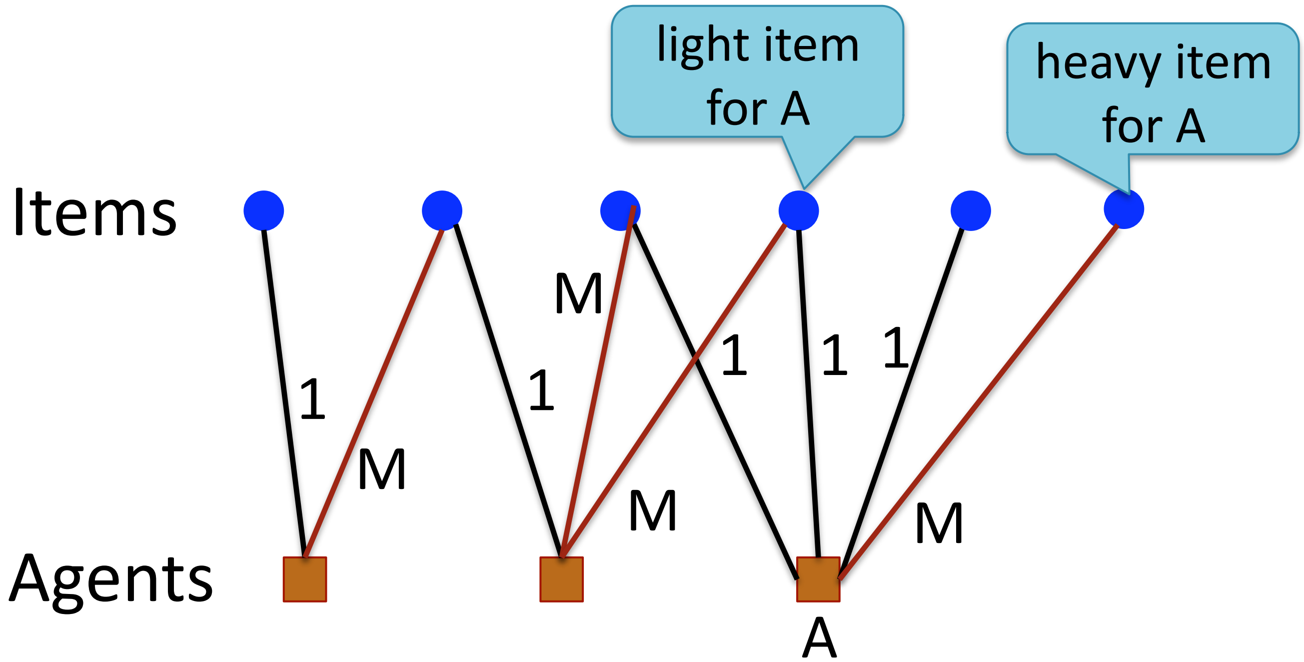
In this talk we also focus on the {0,1,M} setting but without the additional assumptions.

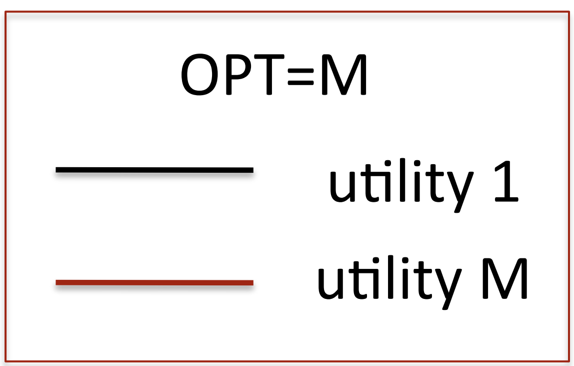
- All utilities are in {0, 1, M}, where OPT=M.
- In the graph induced by utility-M edges:
  - All items have degree at most 2, or
  - Graph contains no cycles
- An $\tilde{O}\left(n^{\epsilon}\right)$-approximation in time $n^{O(1/\epsilon)}$ for these cases



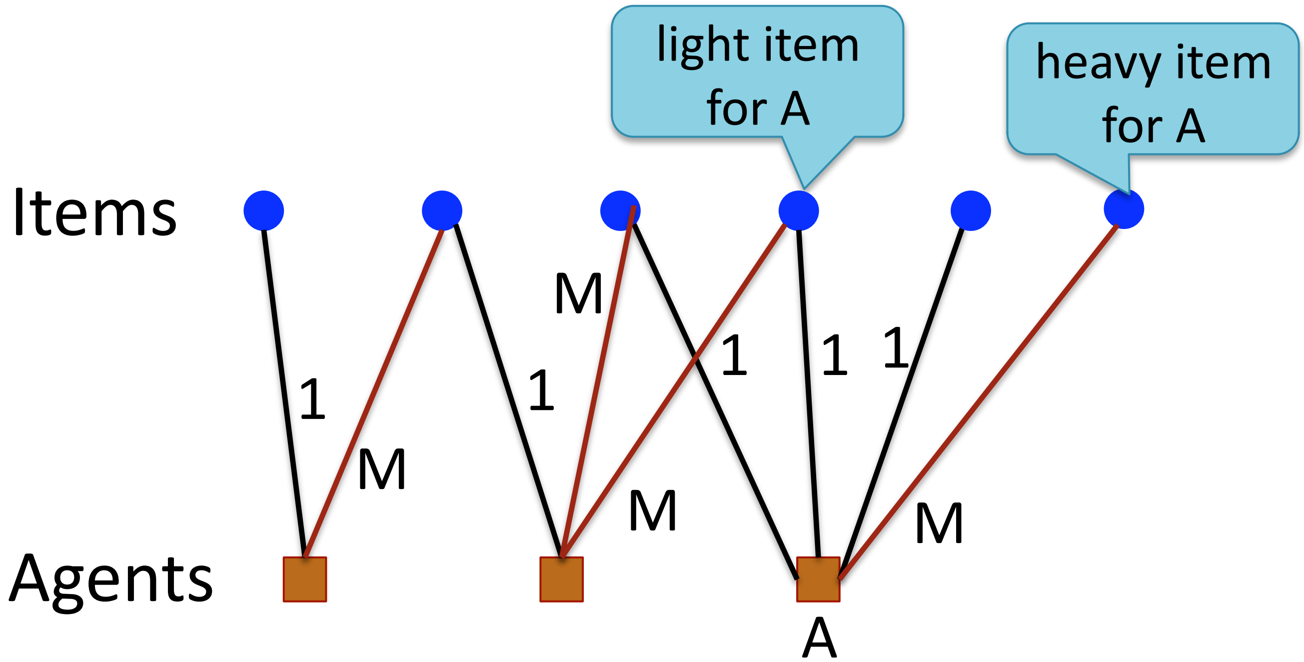
The $\tilde{O}\left(n^{\epsilon}\right)$-Approximation Algorithm

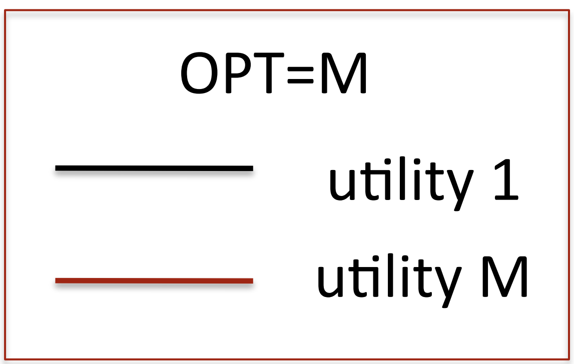
For simplicity, assume all utilities are in {0,1,M}, and OPT=M.





An item can be light for some agents and heavy for others.

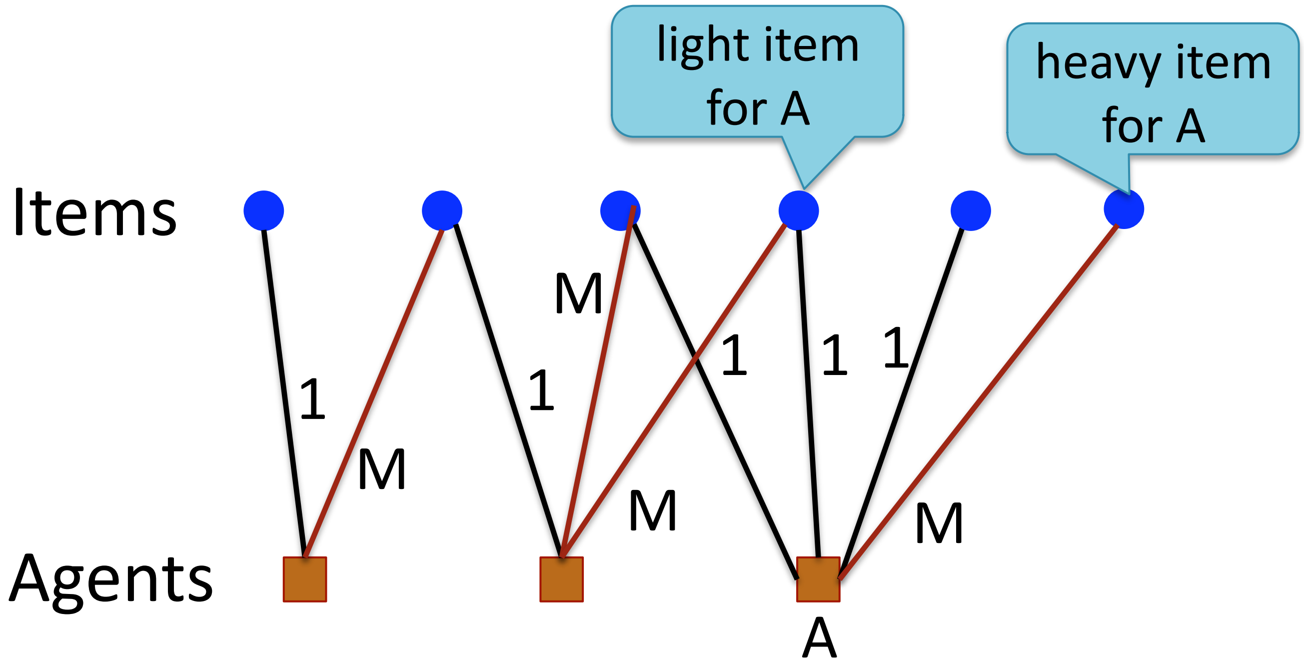


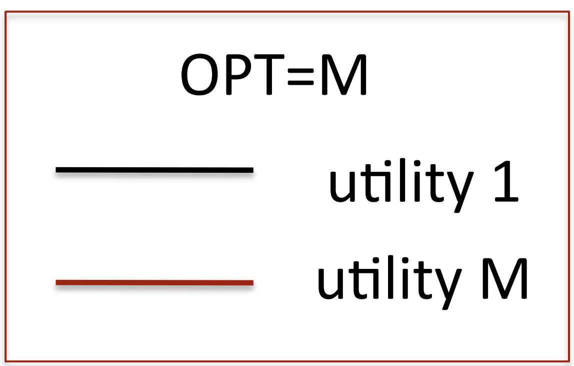


Optimal solution

Each agent A is assigned:

- One heavy item or
- M light items





α-approximate solution

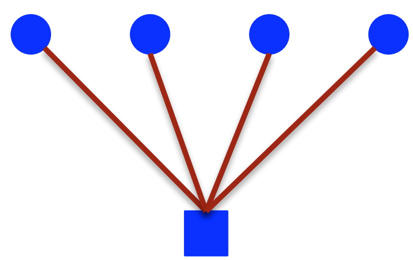
Each agent A is assigned:

- One heavy item or
- ~~M~~ M/α light items

Canonical Instances

All agents are either heavy or light.

Heavy Agent



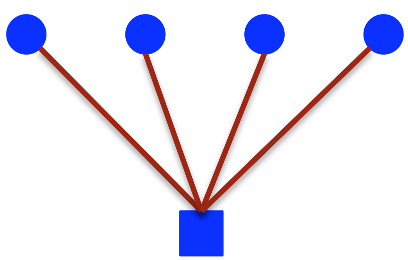
All adjacent items are heavy

Light Agent

Canonical Instances

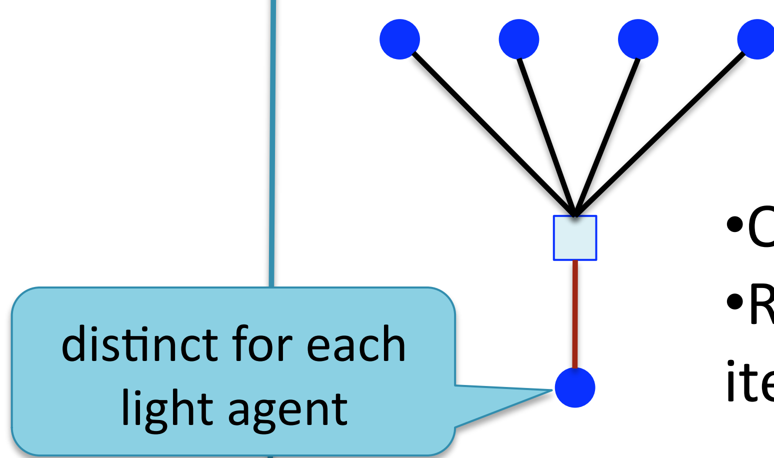
All agents are either heavy or light.

Heavy Agent



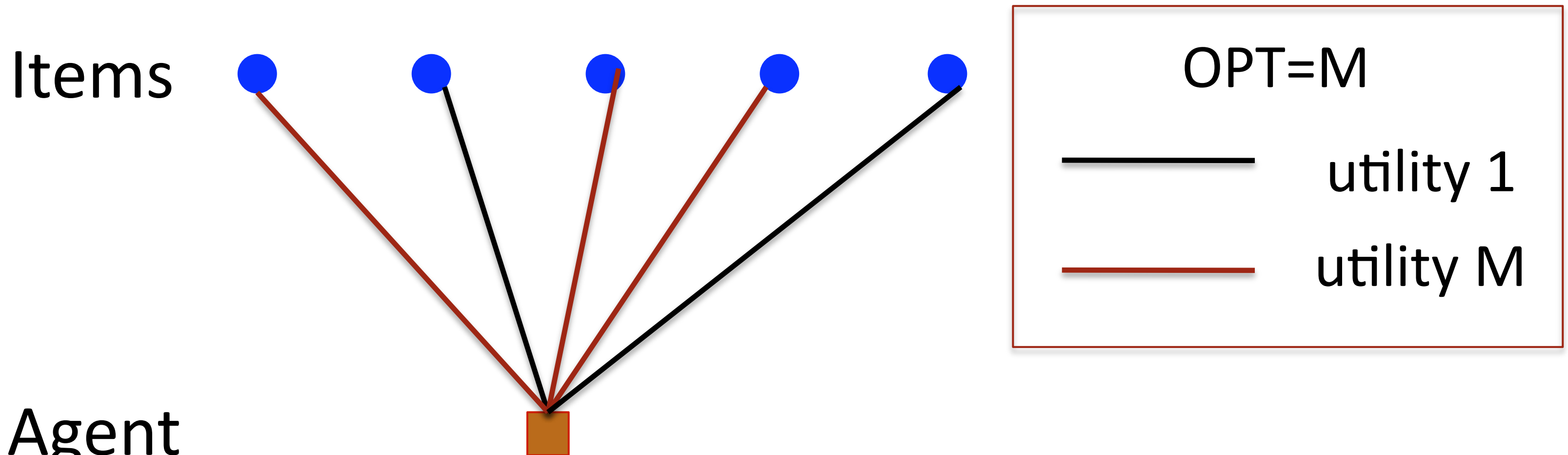
All adjacent items are heavy

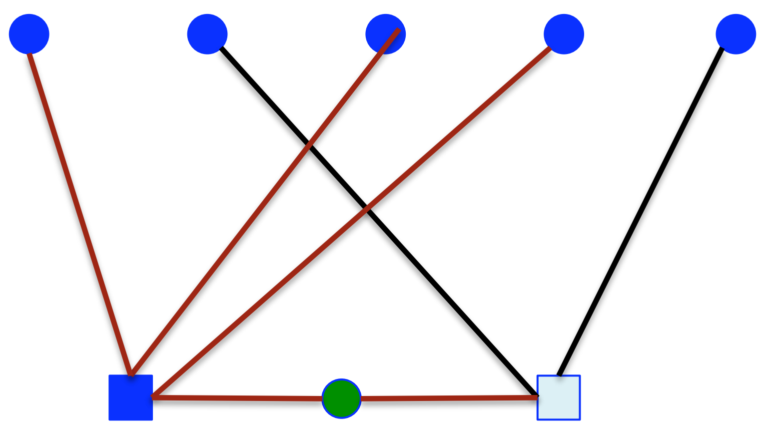
Light Agent



- One heavy item
- Rest of adjacent items are light.

Any Instance to Canonical Instance





From now on we assume w.l.o.g. that our instance is canonical

Notation

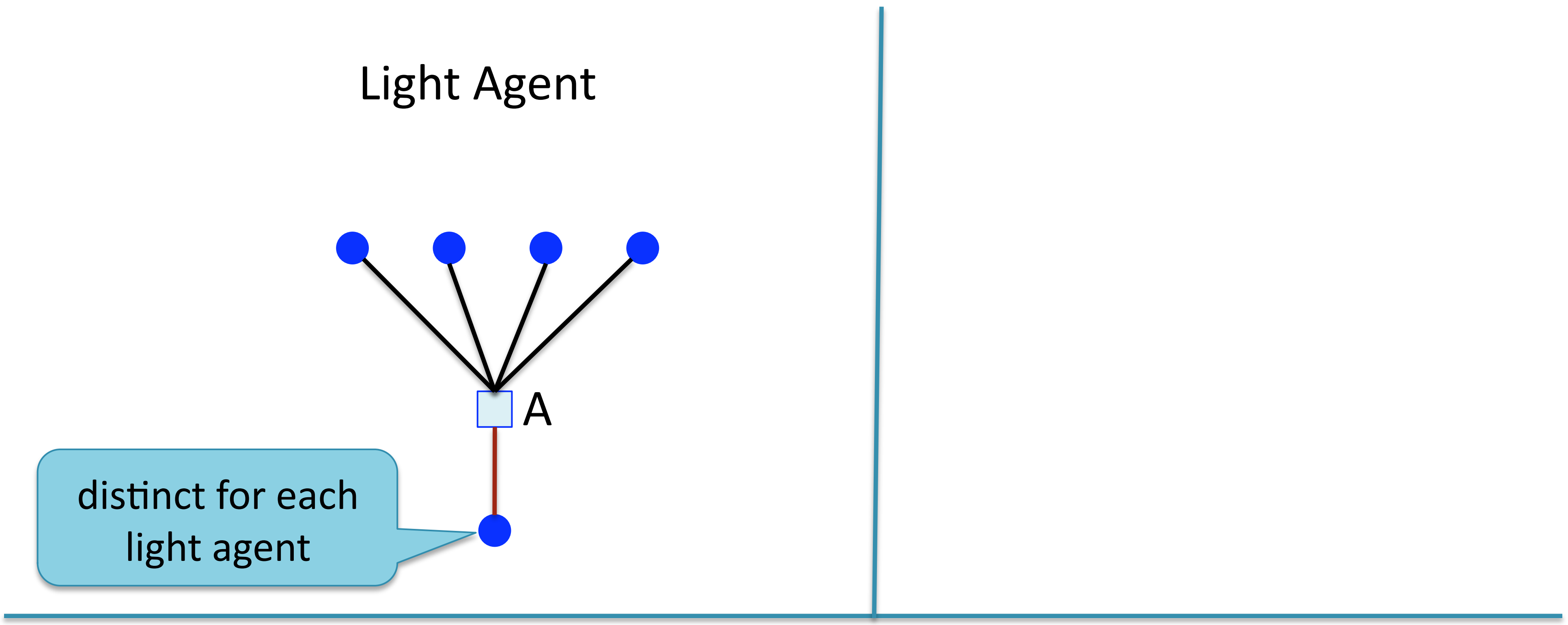
- □ Light agent
- ■ Heavy agent
- ● Item

Step 1: Turn the Assignment Problem into a Network Flow Problem!

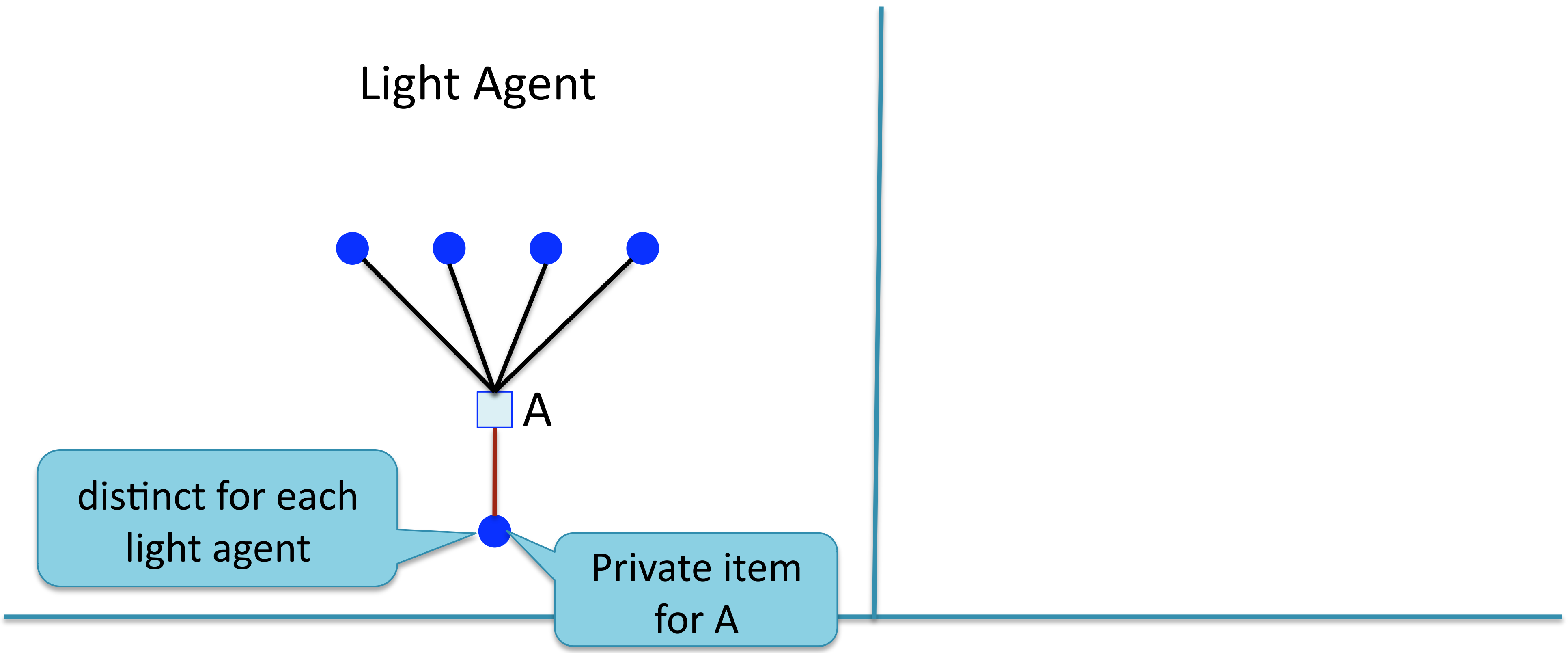
Main Idea

- Temporarily assign private items to agents
  - Item can be private for at most one agent
  - If i is private for A then $u_{A,i}=M$

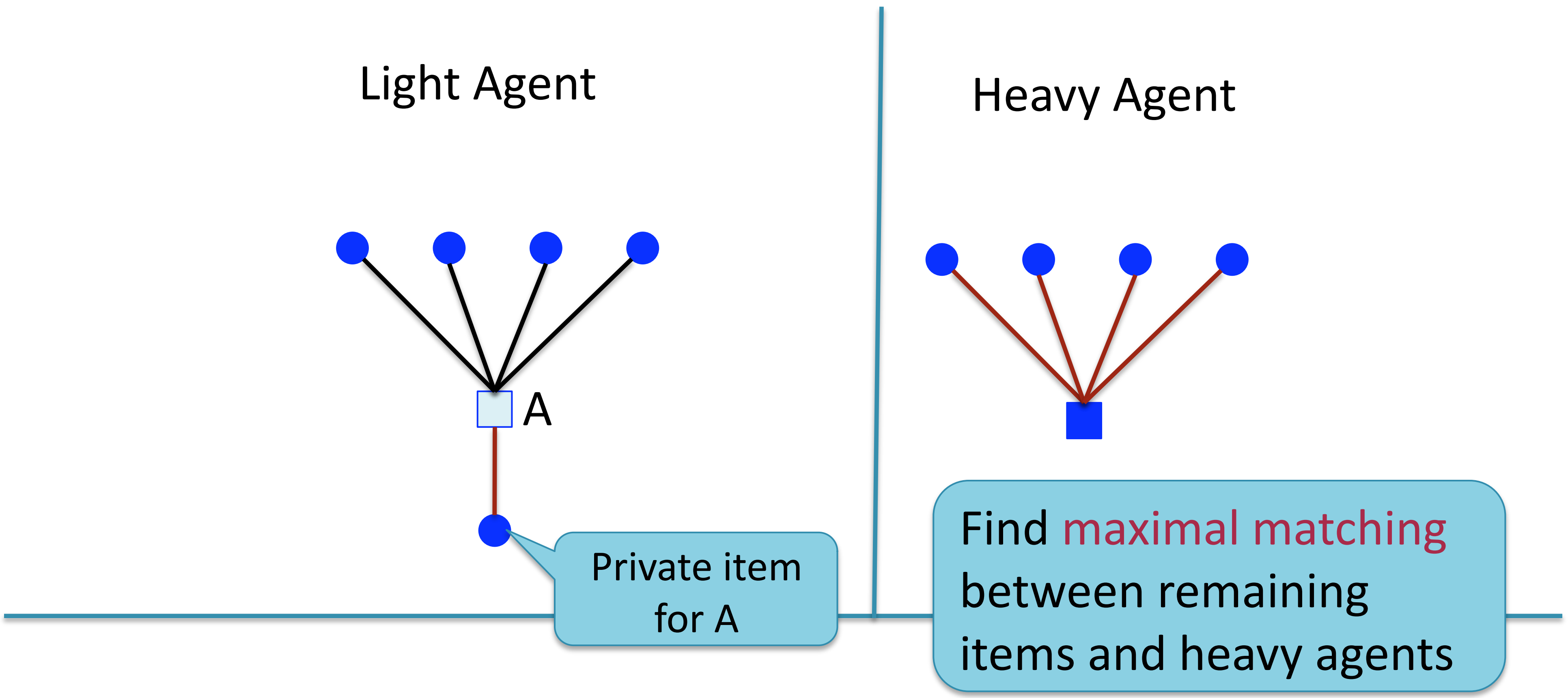
Assignment of Private Items



Assignment of Private Items



Assignment of Private Items



Main Idea

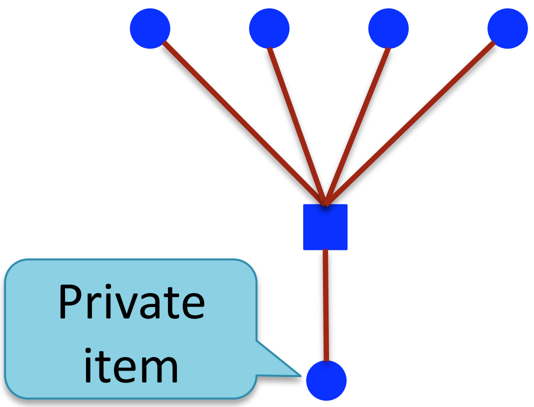
- Temporarily assign private items to agents
  - Item can be private for at most one agent
  - If i is private for A then $u_{A,i}=M$
- If every agent got a private item: done
  - terminals: heavy agents with no private item
  - S: set of items that are not assigned to any agent.
- Re-assignment of items:
  - An agent releases its private item iff it is satisfied by other items.
  - Can be simulated by flow.
  - Flow is sent from items in S towards the terminals.
  - Goal: find flow satisfying the terminals.

The Flow Network

- Start with the incidence graph of agents and items.
- Will build a directed flow network.
- We now go over pieces of the network, showing direction of edges, flow constraints, etc.

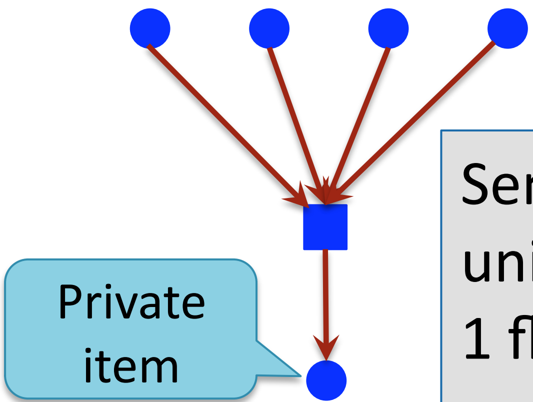
The Flow Network

Heavy agent w. private item



The Flow Network

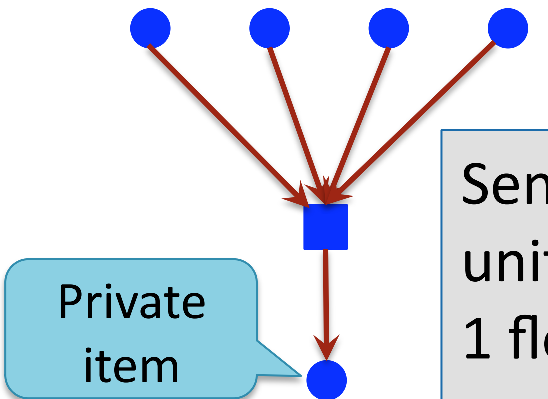
Heavy agent w. private item



Sends 1 flow unit iff receives 1 flow unit

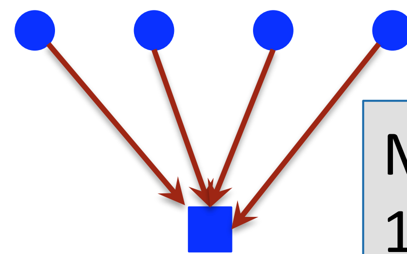
The Flow Network

Heavy agent w. private item



Sends 1 flow unit iff receives 1 flow unit

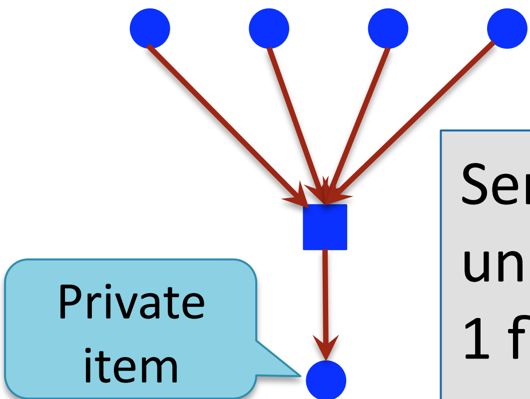
Terminal



Must receive 1 flow unit

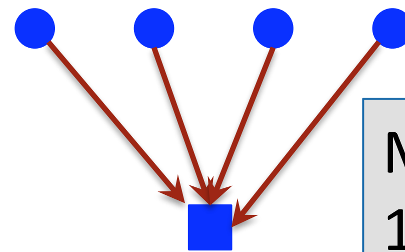
The Flow Network

Heavy agent w. private item



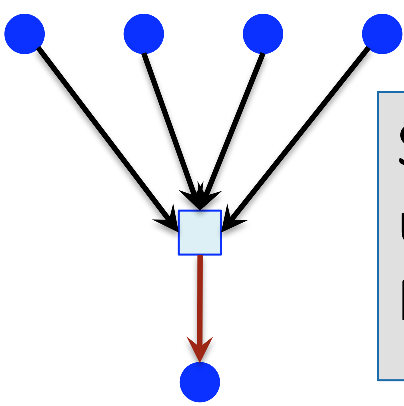
Sends 1 flow unit iff receives 1 flow unit

Terminal



Must receive 1 flow unit

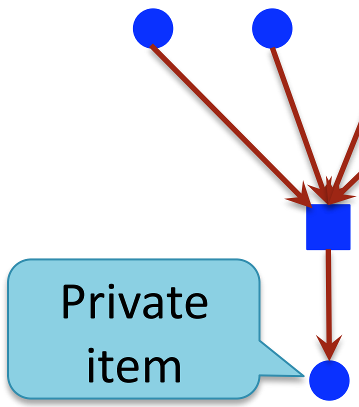
Light Agent



Sends 1 flow unit iff receives M flow units

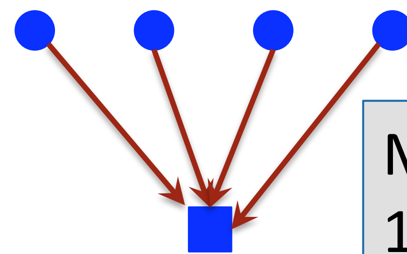
The Flow Network

Heavy agent w. private item



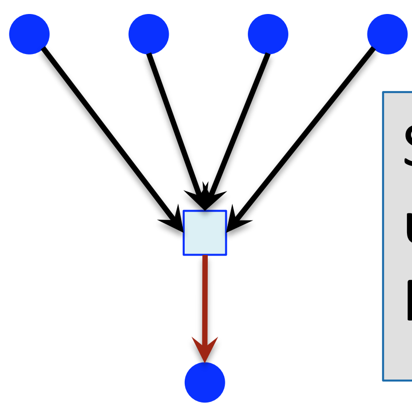
Sends 1 flow unit iff receives 1 flow unit

Terminal



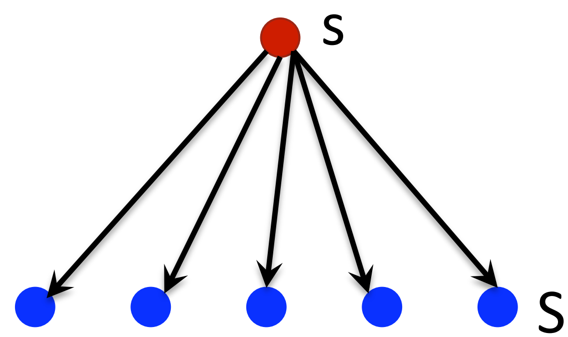
Must receive 1 flow unit

Light Agent

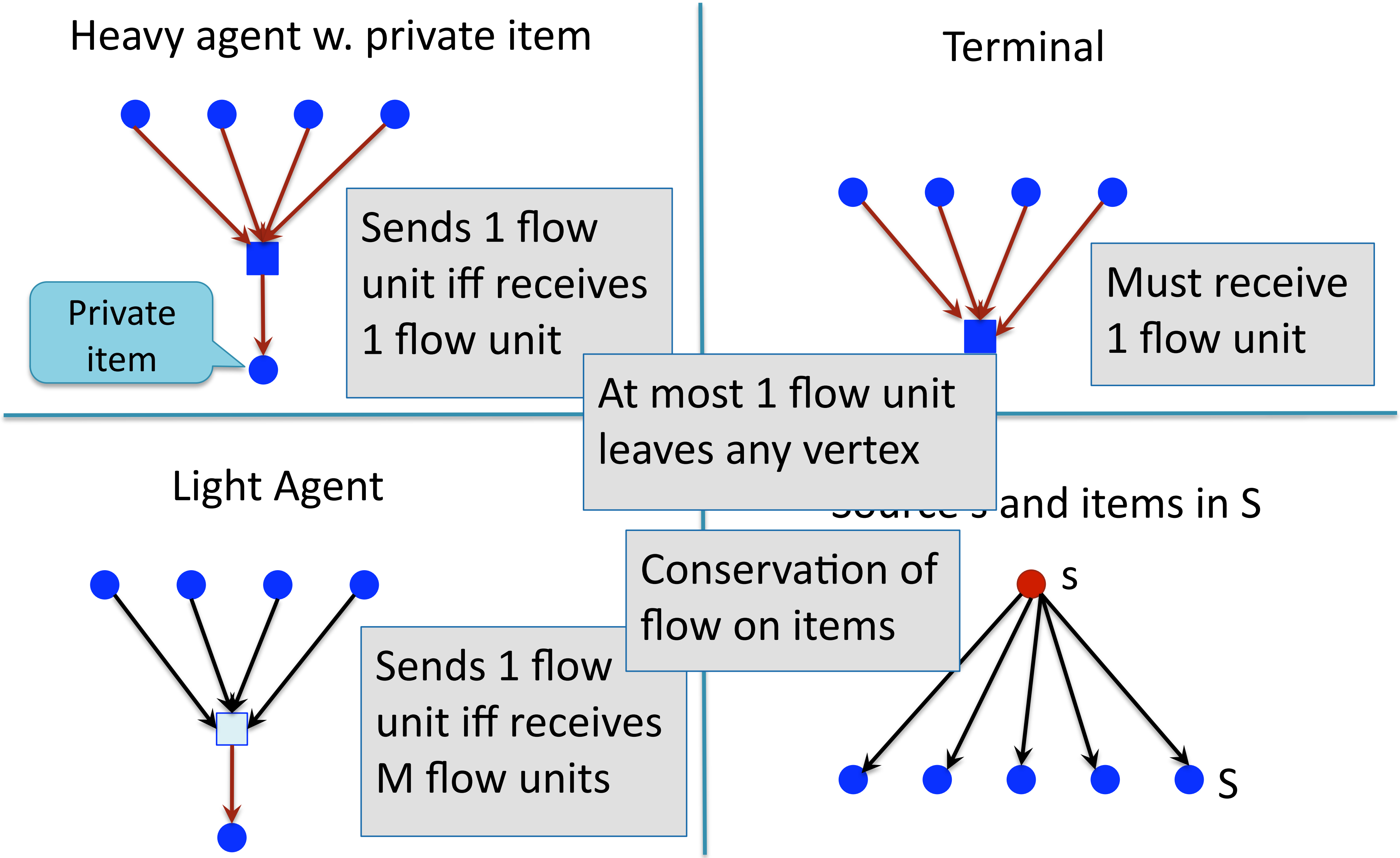


Sends 1 flow unit iff receives M flow units

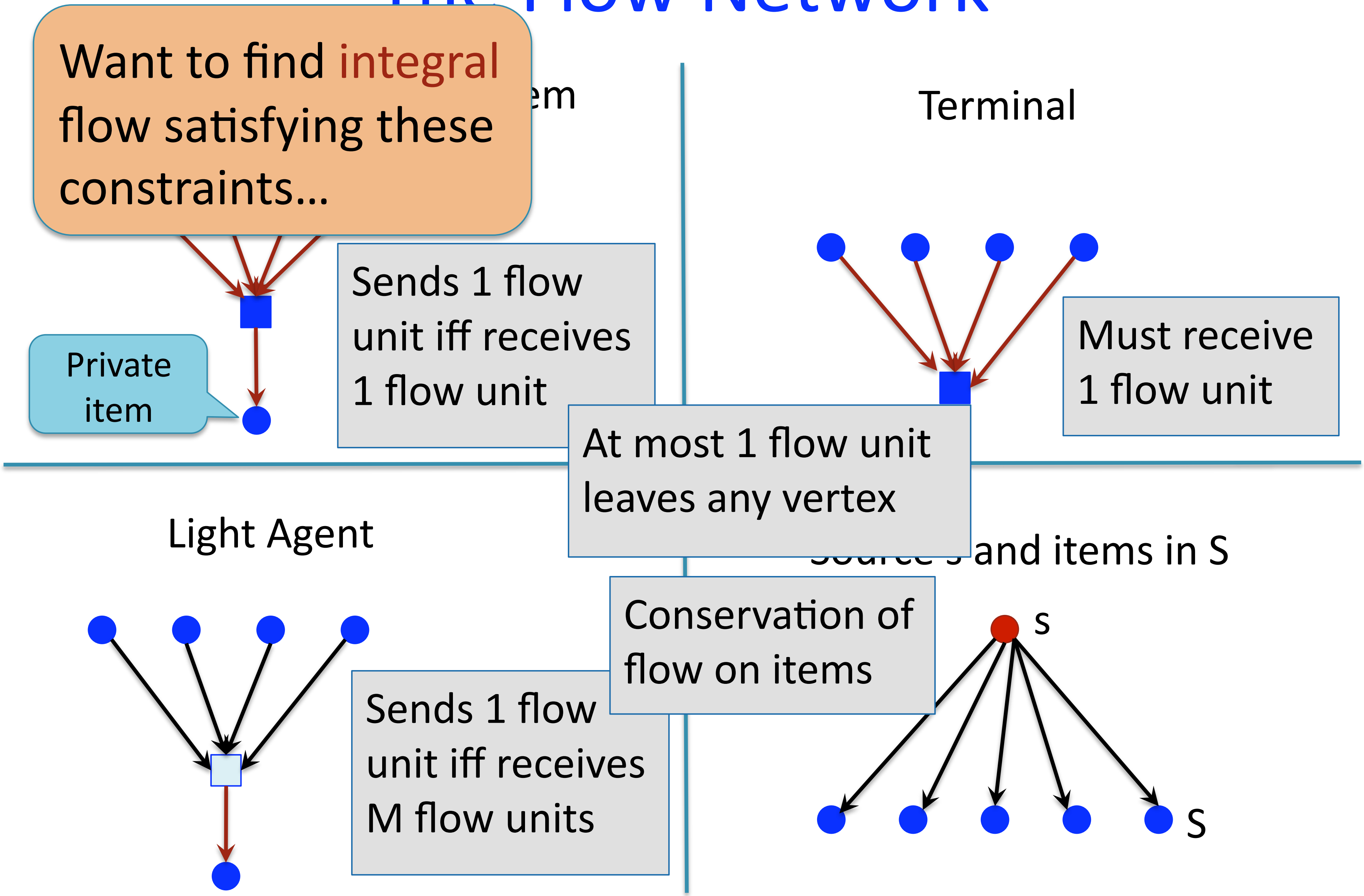
Source s and items in S



The Flow Network



The Flow Network



Interpretation of Flow

Edge e carries 1 flow unit ⟷ Lies in the symmetric difference of OPT and our assignment of private items

---

No flow sent through agent A ⟷ A is assigned its private item

Flow from item i to agent A ⟷ Item i is assigned to A

i is not private for A

Interpretation of Flow

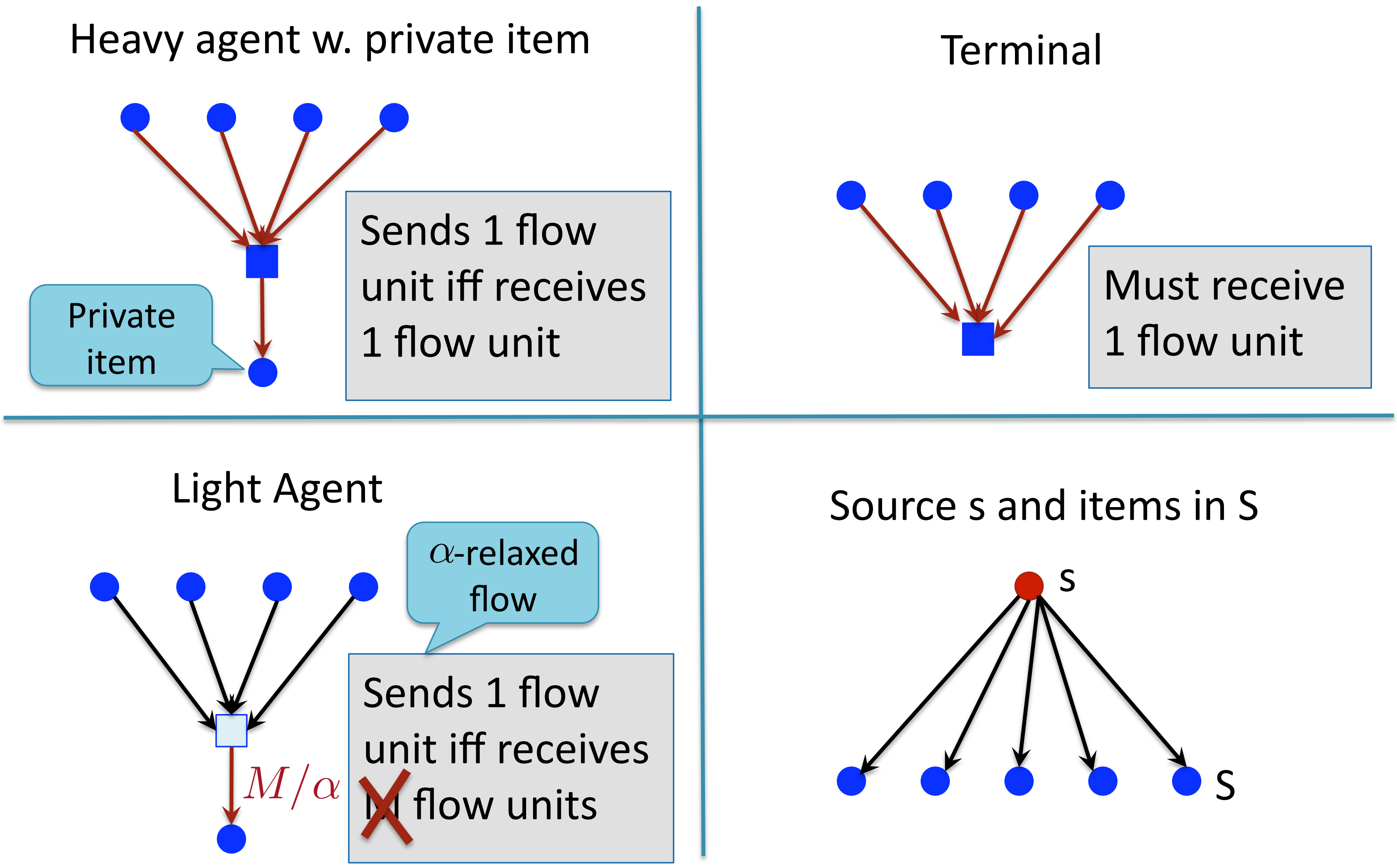
Edge e carries 1 flow unit

↔

Lies in the symmetric difference of OPT and our assignment of private items

- If OPT=M then such flow always exists!

The Flow Network



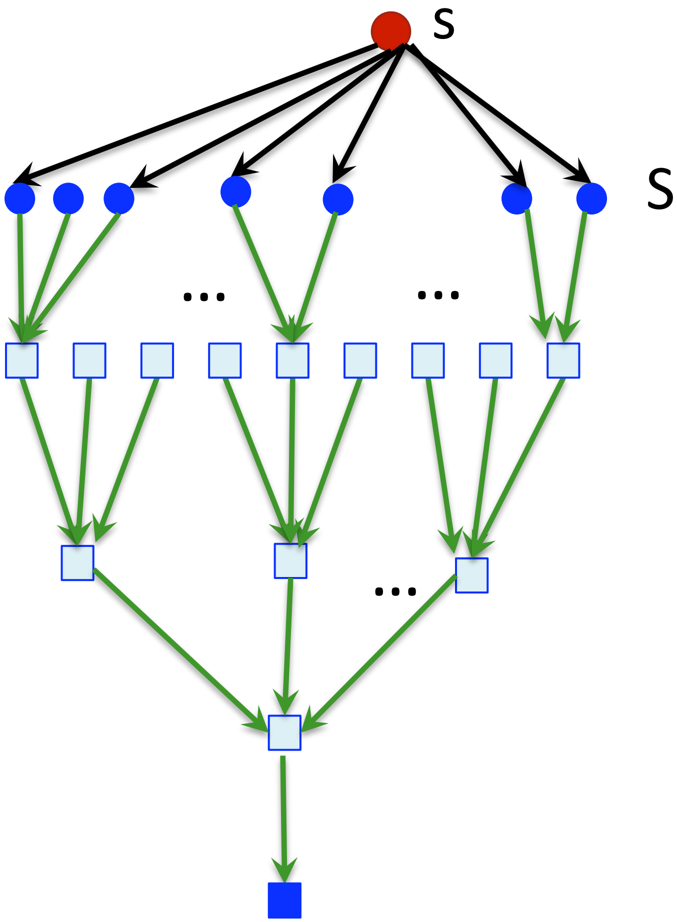
Interpretation of Flow

Edge e carries 1 flow unit ⟷ Lies in the symmetric difference of OPT and our assignment of private items

- If OPT=M then such flow always exists!
- An α-relaxed flow gives an α-approximation!

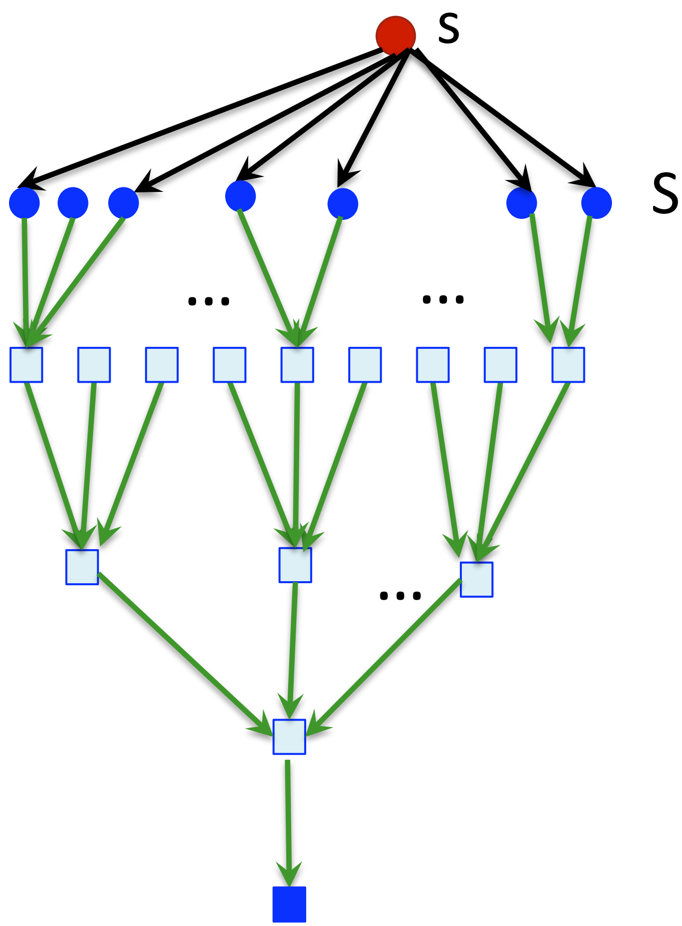
What Does a Feasible Flow Look Like?

A collection of disjoint structures like this:



What Does a Feasible Flow Look Like?

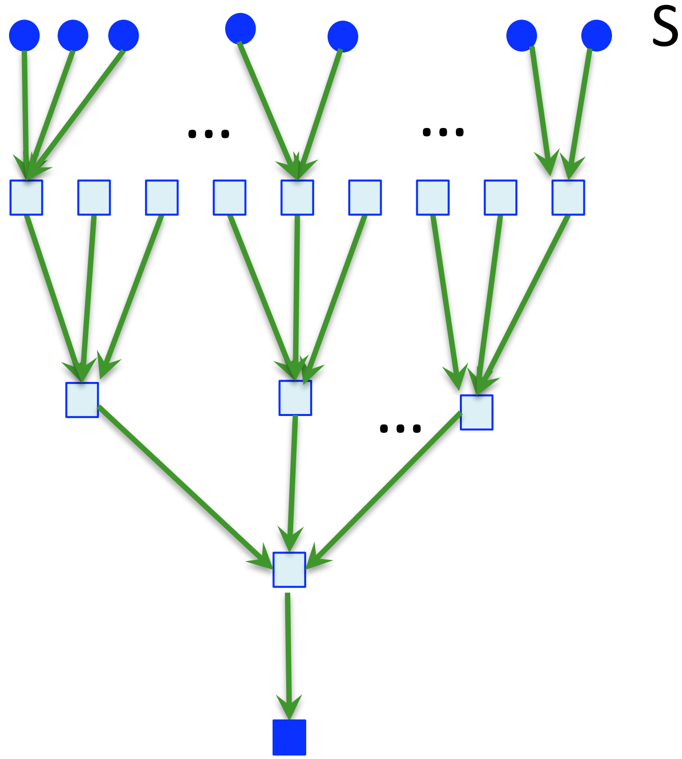
A collection of disjoint structures like this:



Ignore the source vertex s ...

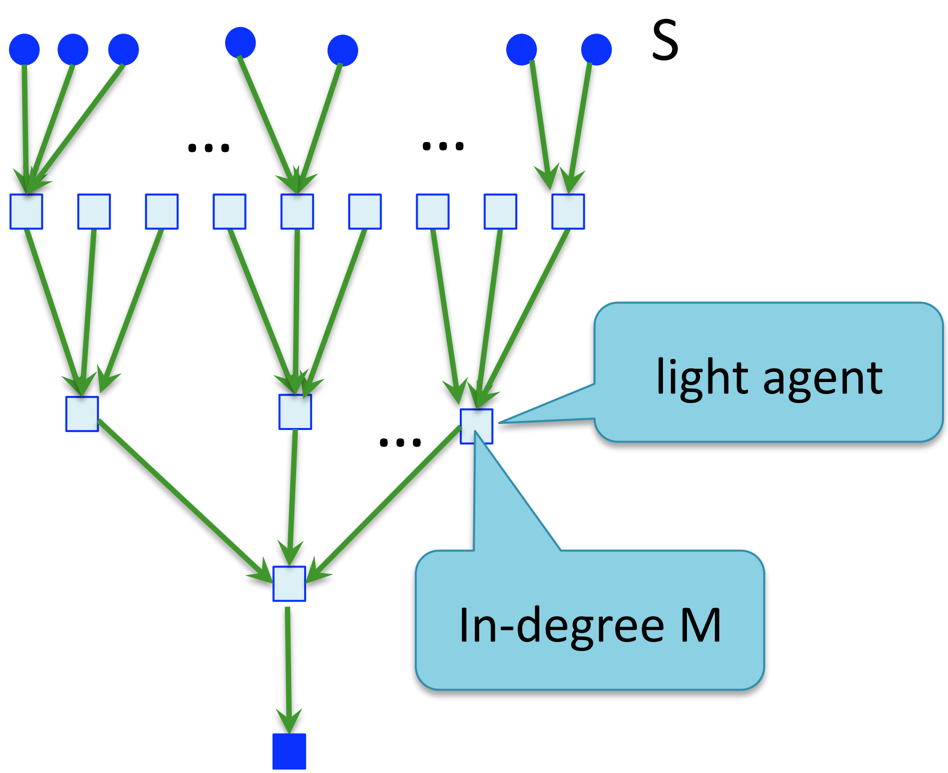
What Does a Feasible Flow Look Like?

A collection of disjoint trees like this:



What Does a Feasible Flow Look Like?

A collection of disjoint trees like this:

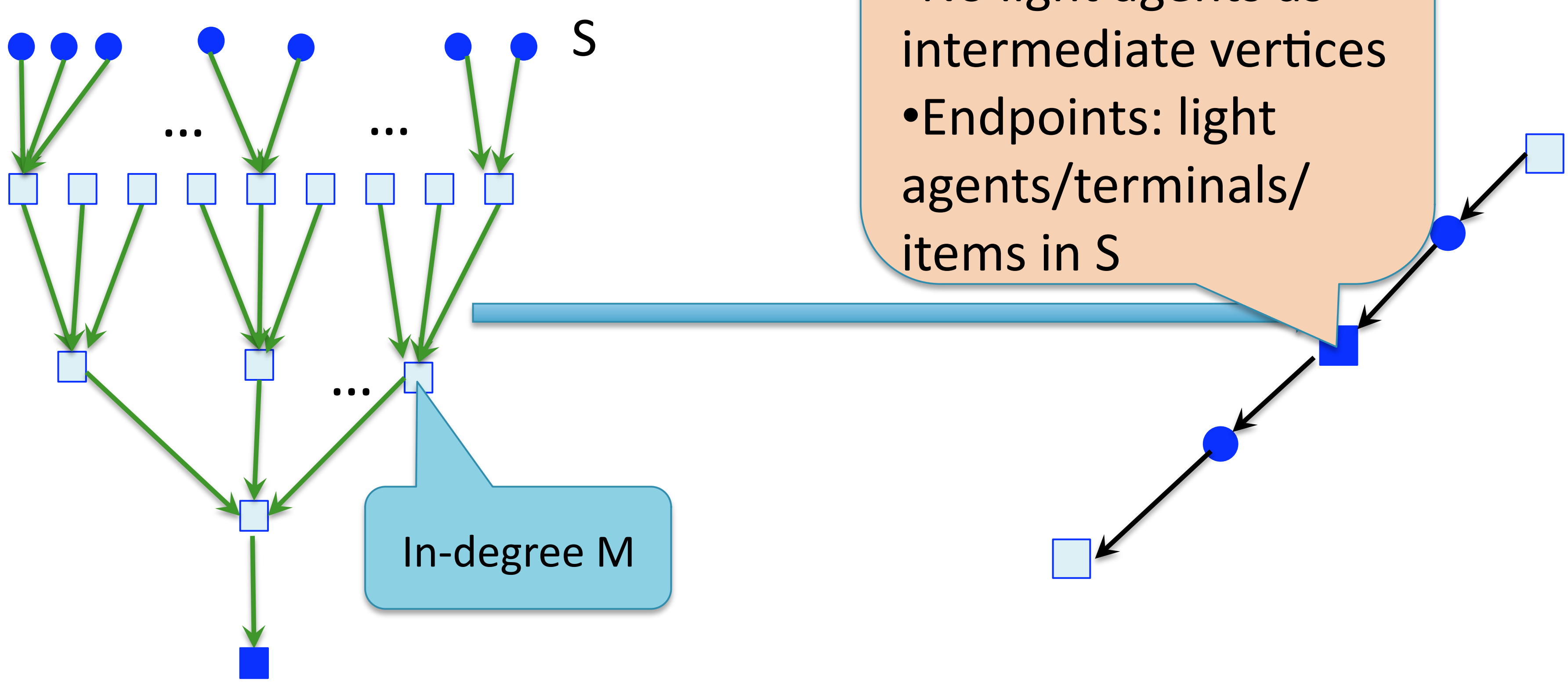


What Does a Feasible Flow Look Like?

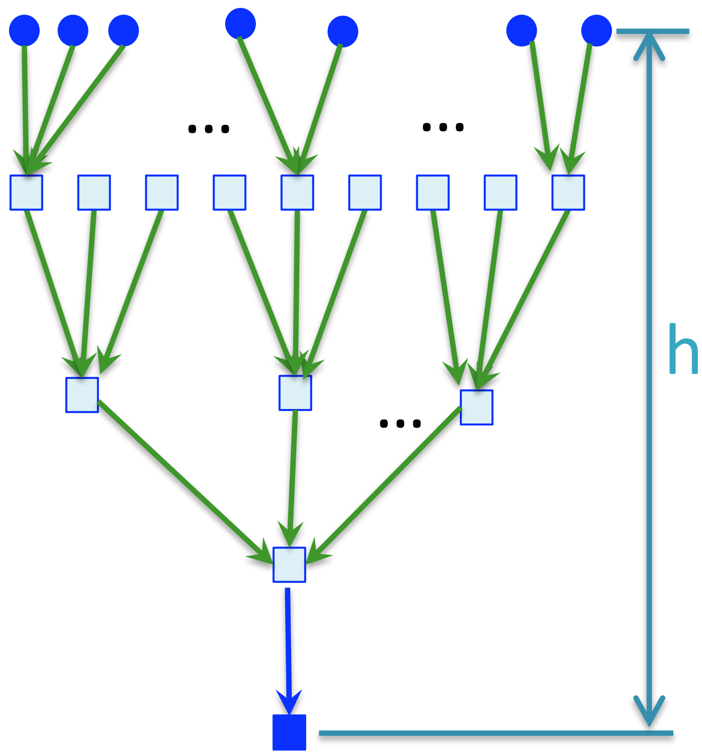
A collection of disjoint trees like this:

Every tree edge is an elementary path:

- No light agents as intermediate vertices
- Endpoints: light agents/terminals/ items in S



Equivalent Problem Statement



Find a collection of such disjoint trees!

- Solution cost = min degree of a light agent.
- If we only want $\tilde{O}\left(n^{\epsilon}\right)$ -approximation, can assume that $h \leq 1/\epsilon$ (by cutting the optimal trees).

Rest of the Algorithm

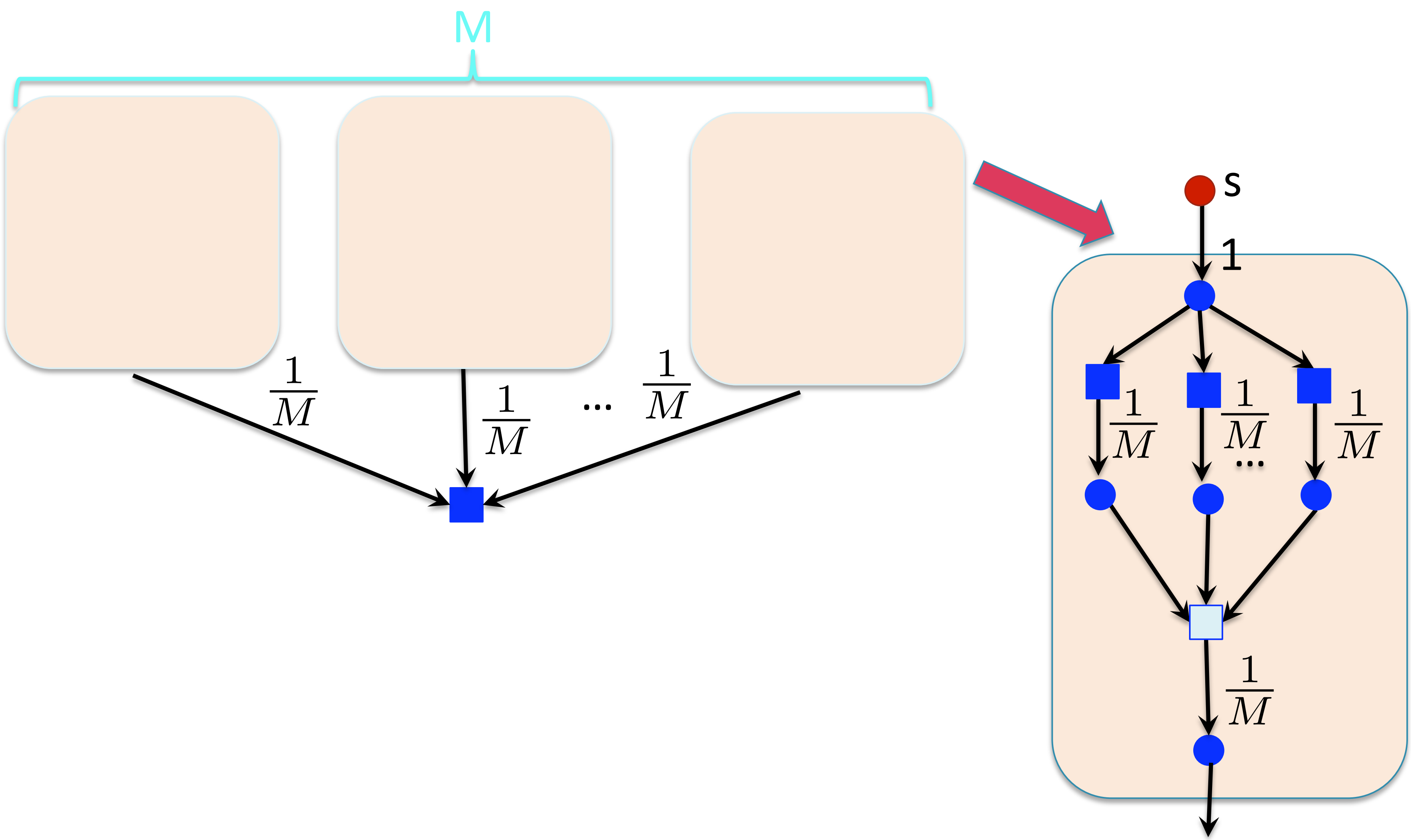
- Write an LP and perform LP-rounding
  - Our LP has $\Omega(\sqrt{m})$ integrality gap, size $n^{O(1/\epsilon)}$
  - LP-rounding gives poly-log n-approximate "almost feasible" solutions.
- Use LP-rounding as sub-routine to get final solution.

Part 1: LP and its Rounding

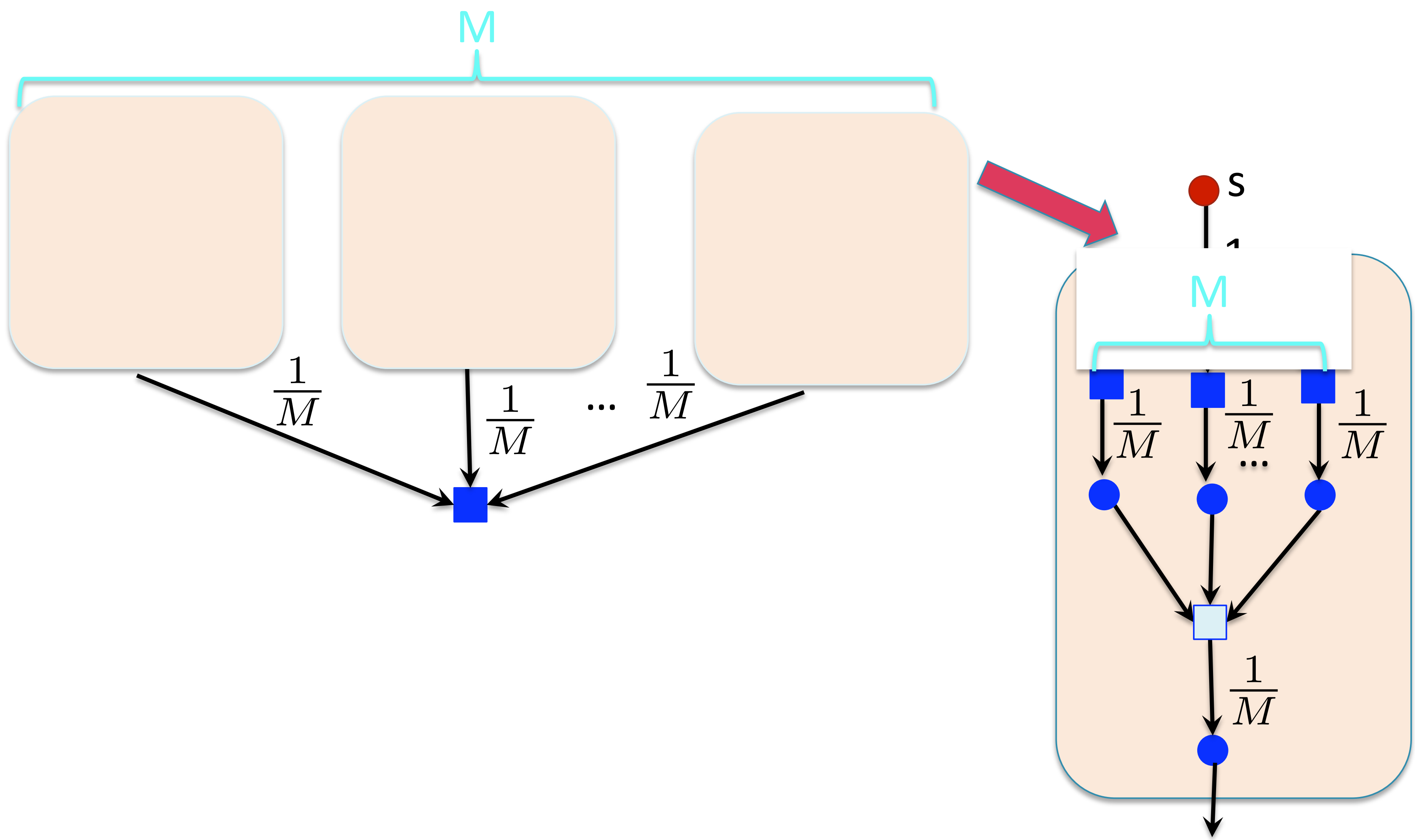
Natural LP

- Can write standard LP relaxation of flow constraints.
  - Easy to see that such an LP is too weak.

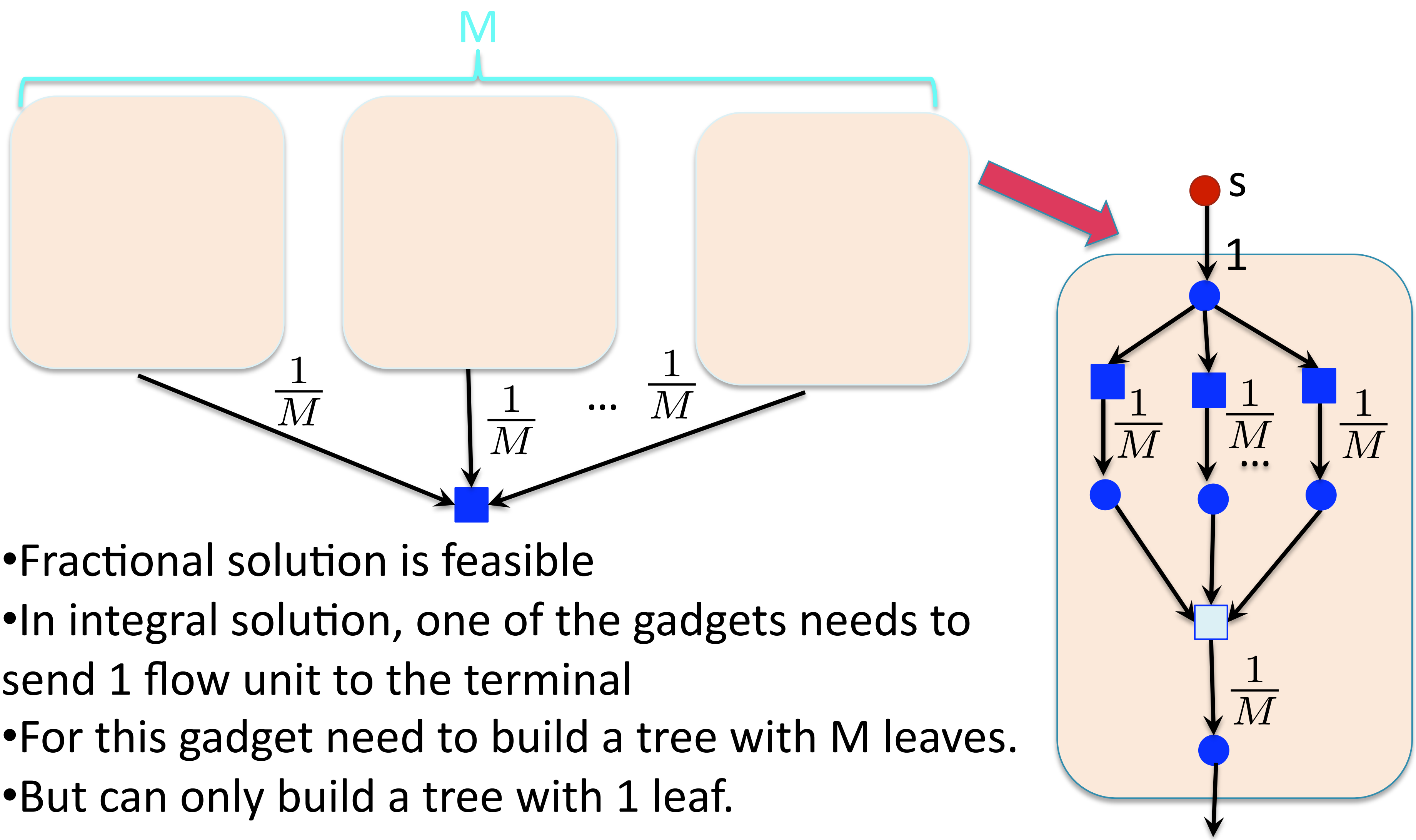
Why Standard Flow LP won't Work



Why Standard Flow LP won't Work



Why Standard Flow LP won't Work

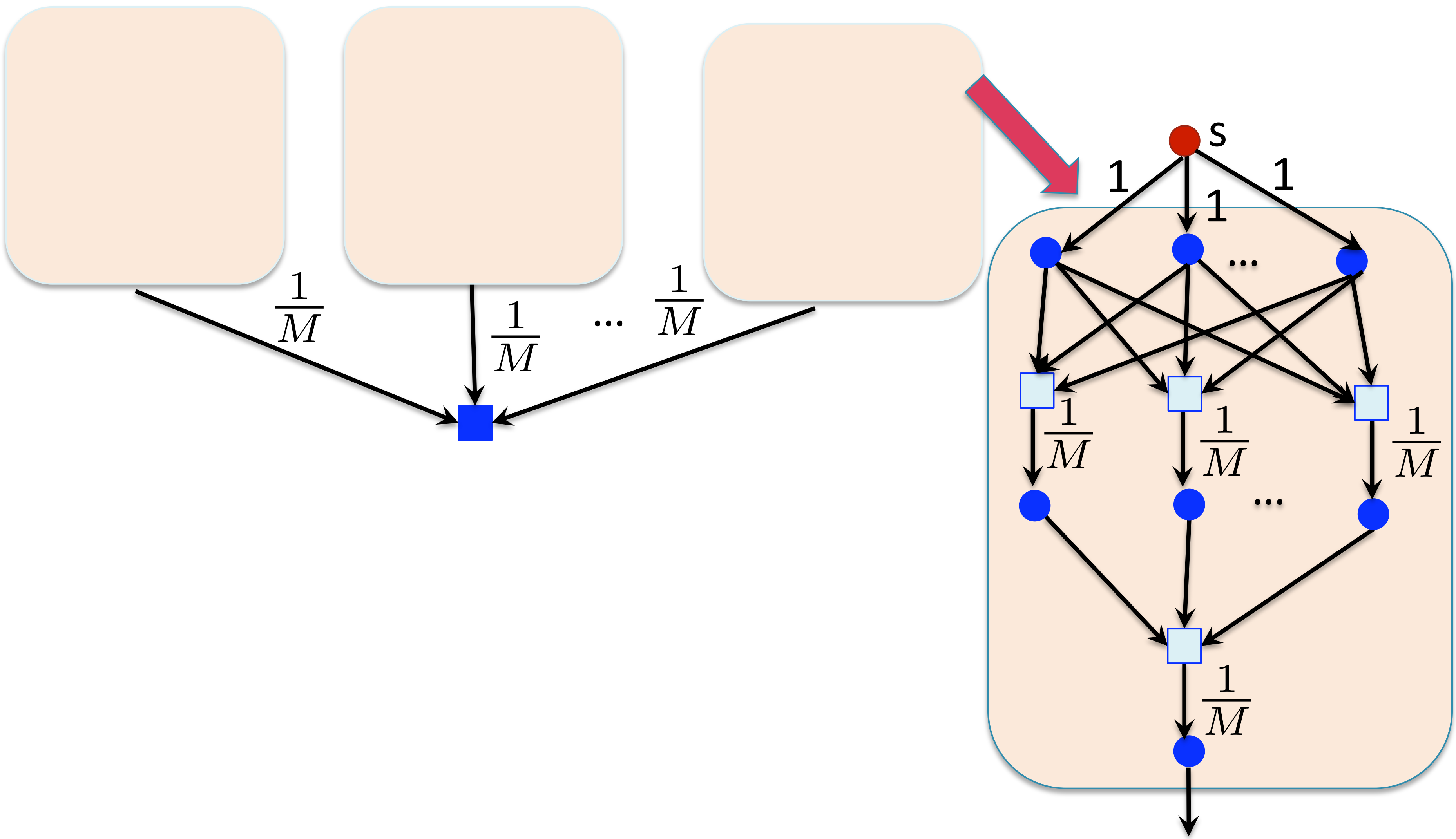


- Fractional solution is feasible
- In integral solution, one of the gadgets needs to send 1 flow unit to the terminal
- For this gadget need to build a tree with M leaves.
- But can only build a tree with 1 leaf.

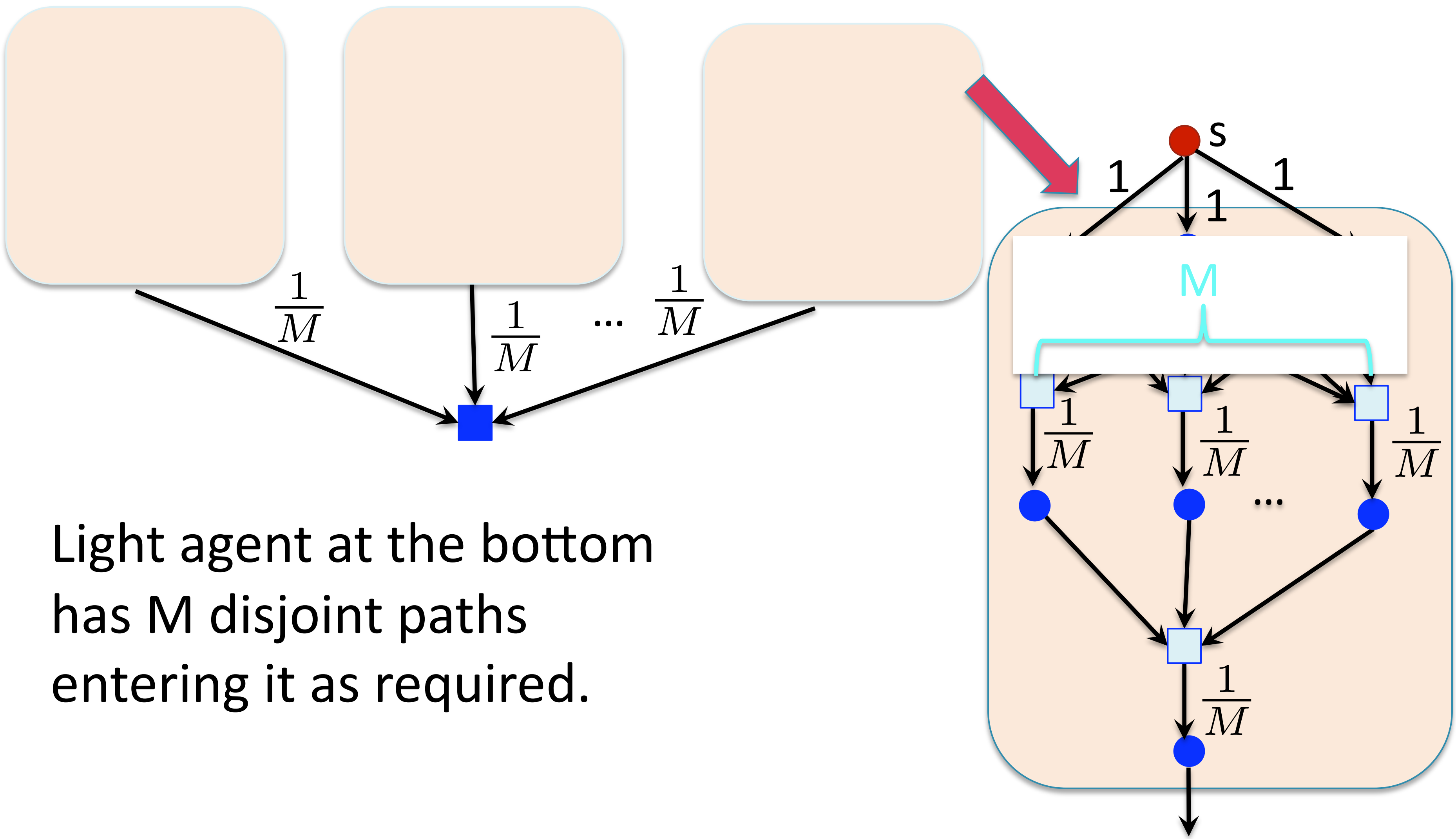
Easy Fix

- Need to keep track where the flow is going.
- For each light agent A, define flow type $f_{A.}$
  - Only flow of type f_A enters A.
  - x_A: amount of flow leaving A.
- New congestion constraints:
  - At most x_A units of flow of type f_A can go through any vertex.
- This will fix the problem in the example.
- But: can build harder examples…

Why Standard Flow LP won't Work

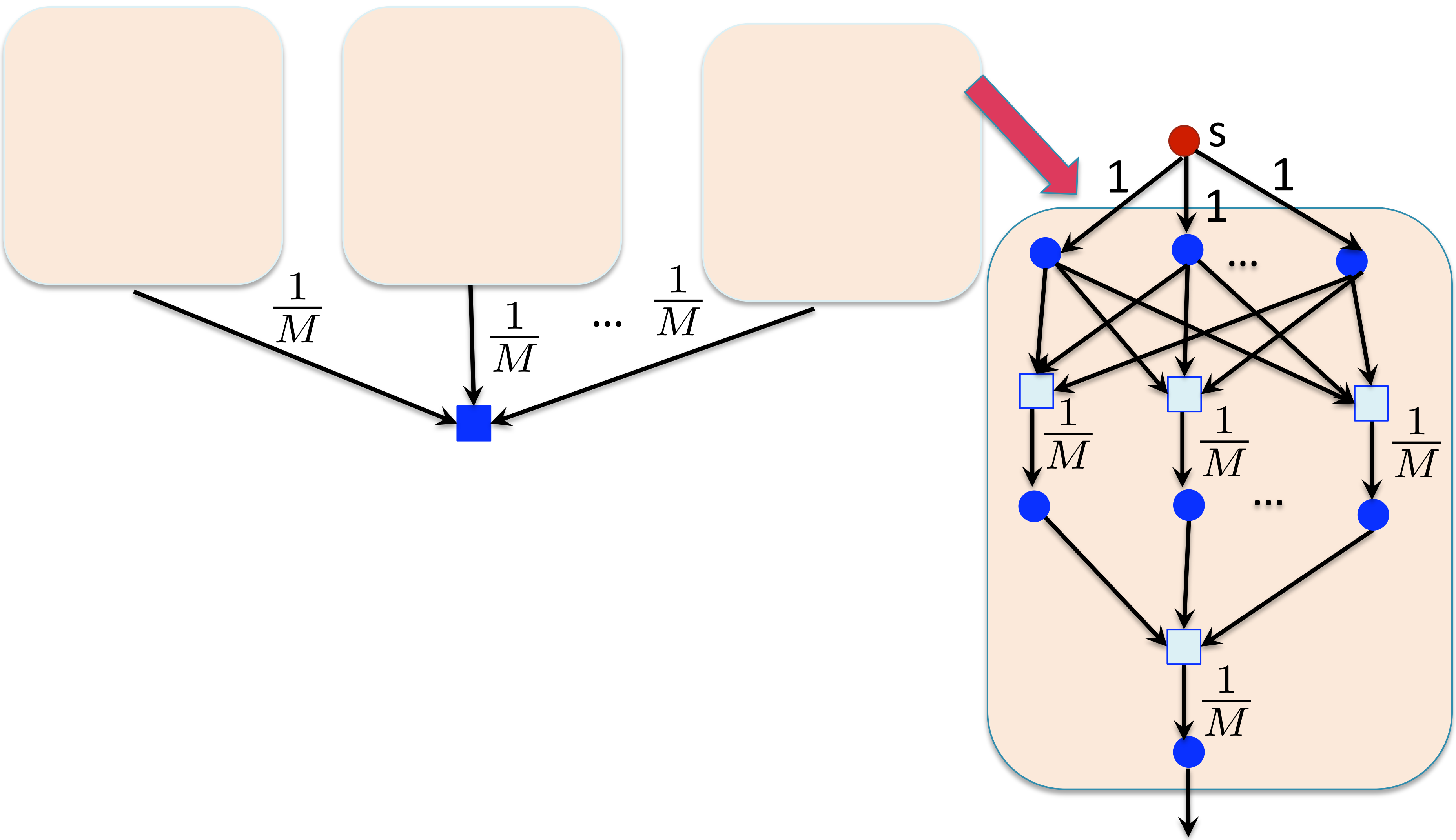


Why Standard Flow LP won't Work

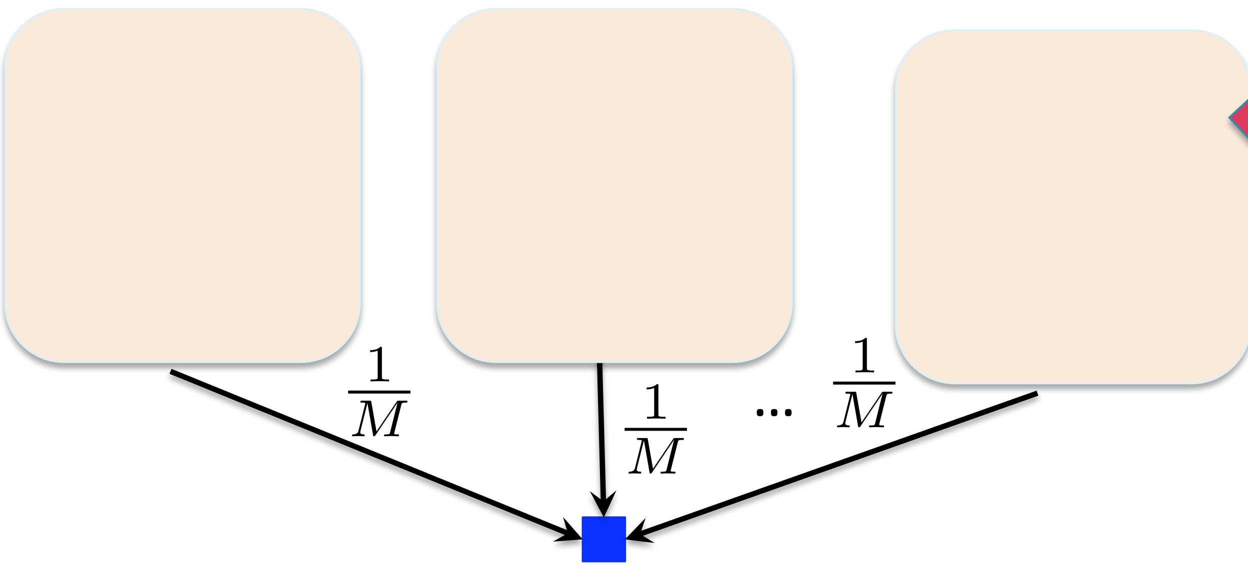


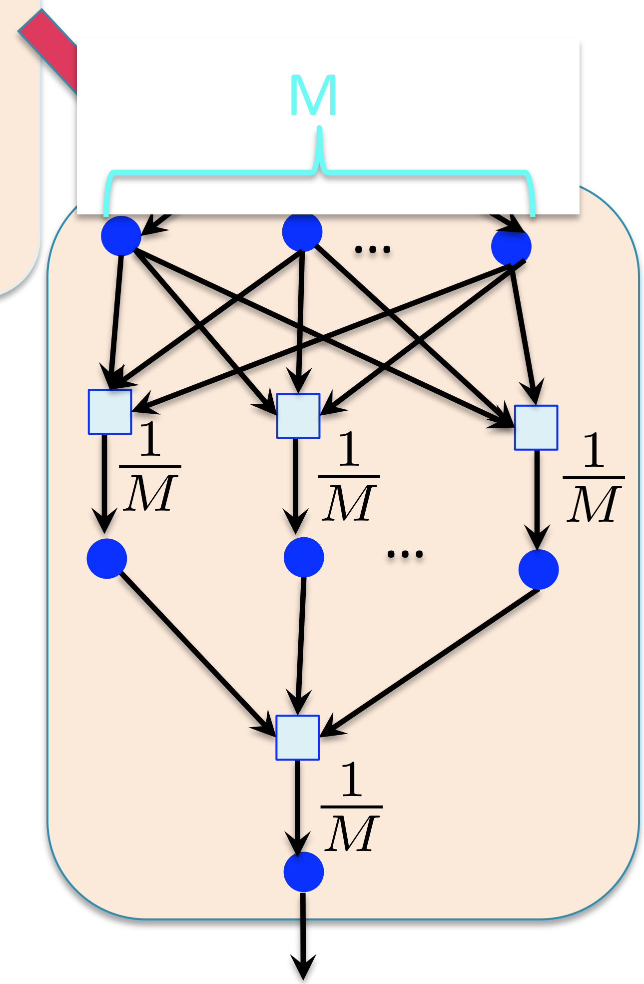
Light agent at the bottom has M disjoint paths entering it as required.

New Problem …



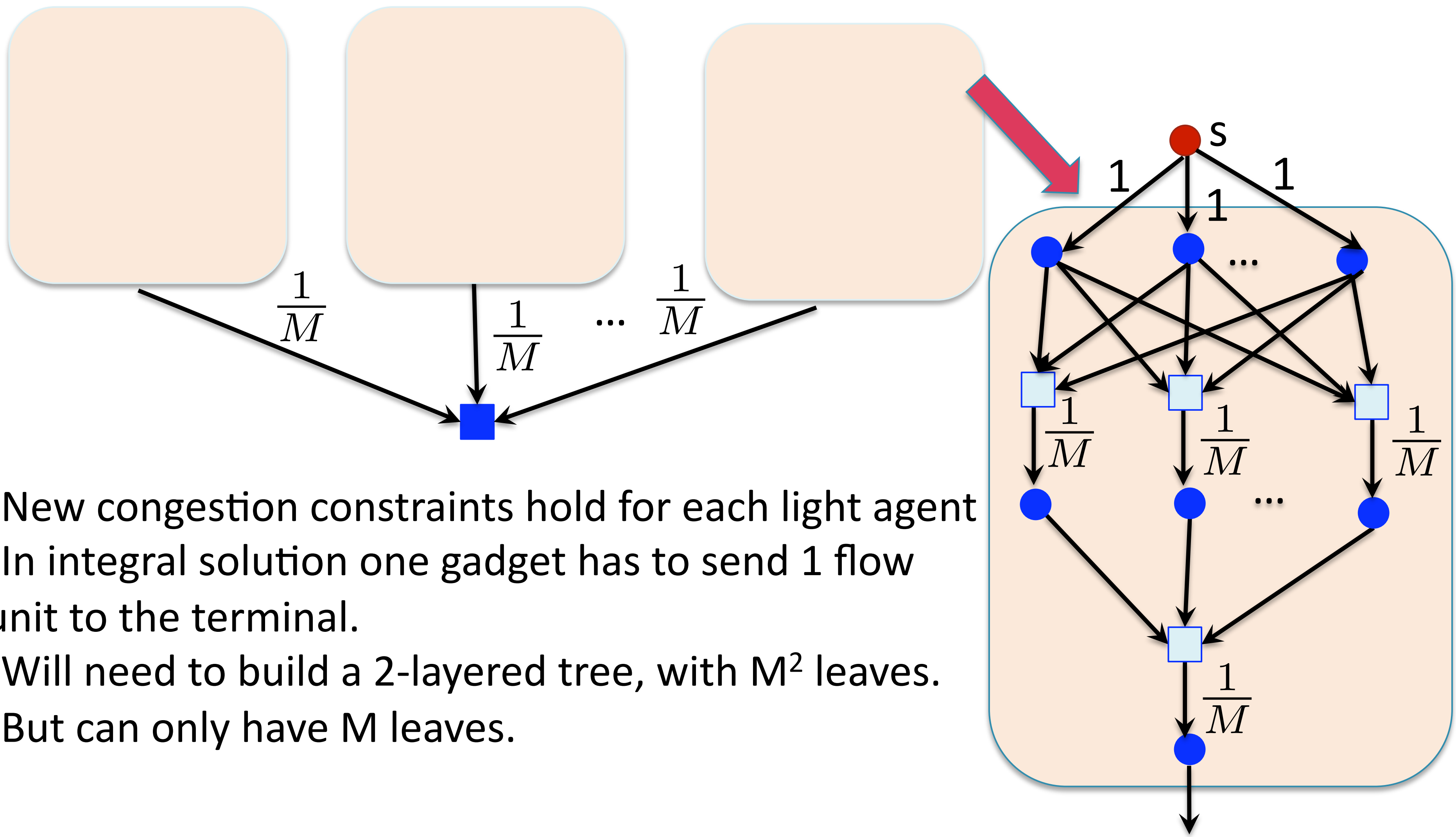
New Problem …





- Each light agent in the middle has M disjoint paths entering it as required.
- But: same M-tuple of items is used for each one of the agents.

New Problem …



- New congestion constraints hold for each light agent
- In integral solution one gadget has to send 1 flow unit to the terminal.
- Will need to build a 2-layered tree, with M^2 leaves.
- But can only have M leaves.

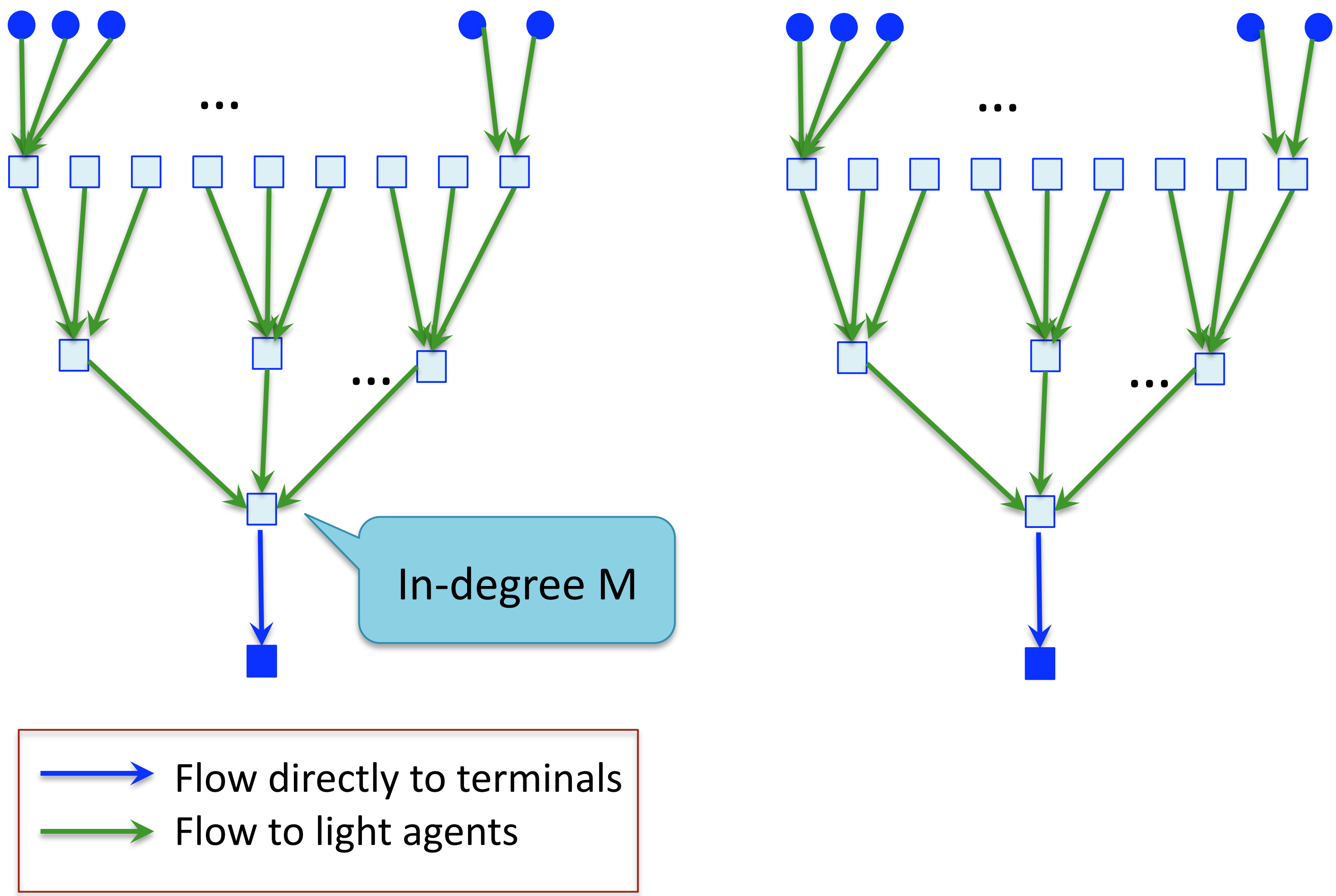
A Fix

- For each pair A,B of light agents define indicator variable $x_{A,B}$: whether or not there is a flow path containing A and B.
- Also define flow type $f_{A,B}$
- Keep the old variables x_A,x_B, that need to be coordinated with $x_{A,B}$
- New congestion constraints:
  - total amount of flow of types $f_{A,B}$ (summed over all A) going through any vertex is at most x_B
- This will fix the above example
- But can make harder examples…

Our LP Relaxation

- For each h'-tuple $(A_1, \ldots, A_{h'})$ of light agents, for each $h' \leq h$, define a variable $x(A_1, \ldots, A_{h'})$
  - indicator variable for having a flow-path containing these light agents
  - need to coordinate the variables across the different tuples
  - new capacity constraints
- Since $h \leq O(1/\epsilon)$, the LP-size is $n^{O(1/\epsilon)}$
- Integrality gap remains $\Omega(\sqrt{m})$
- But we can get polylog-approximate almost feasible solutions!

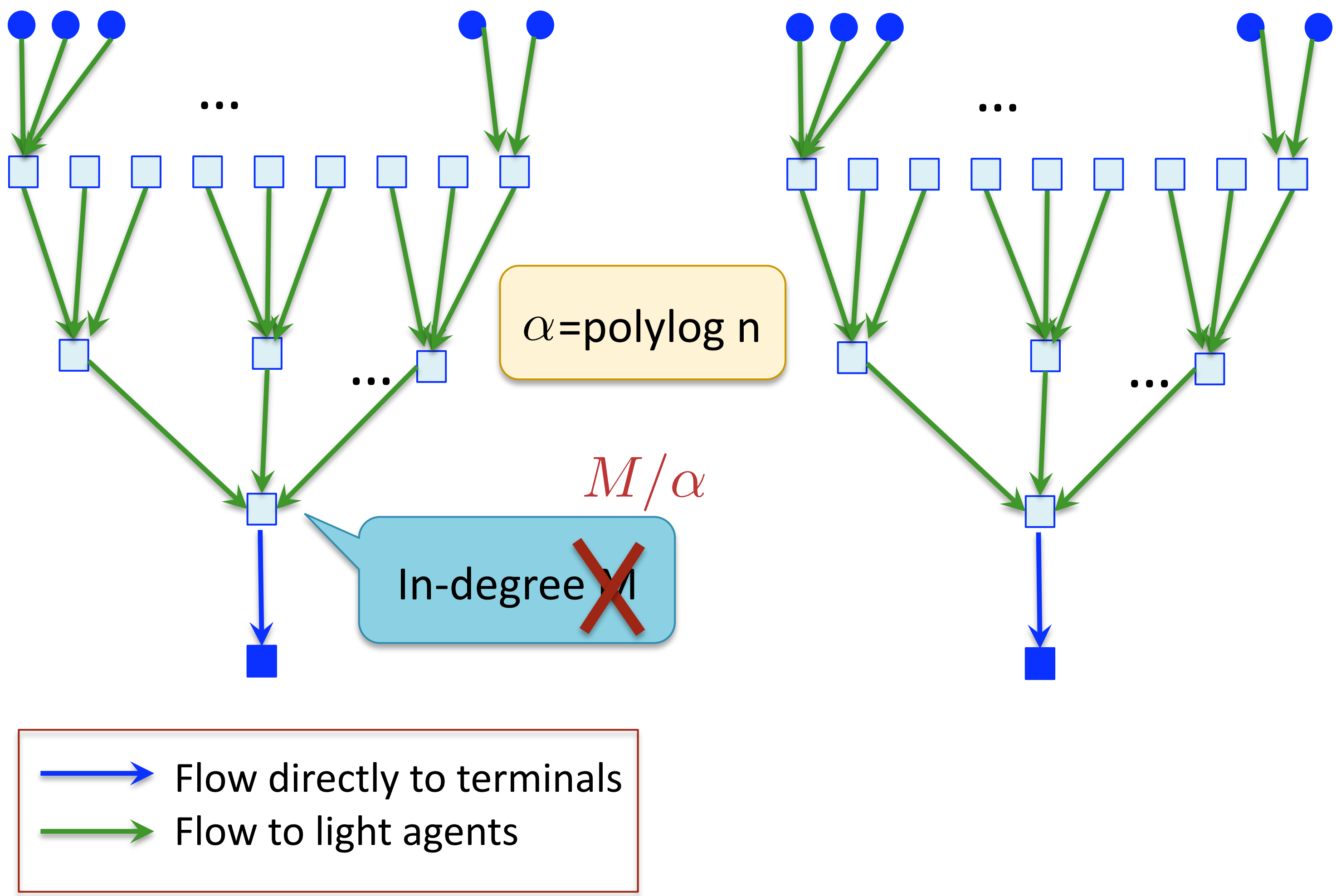
Almost Feasible Solutions



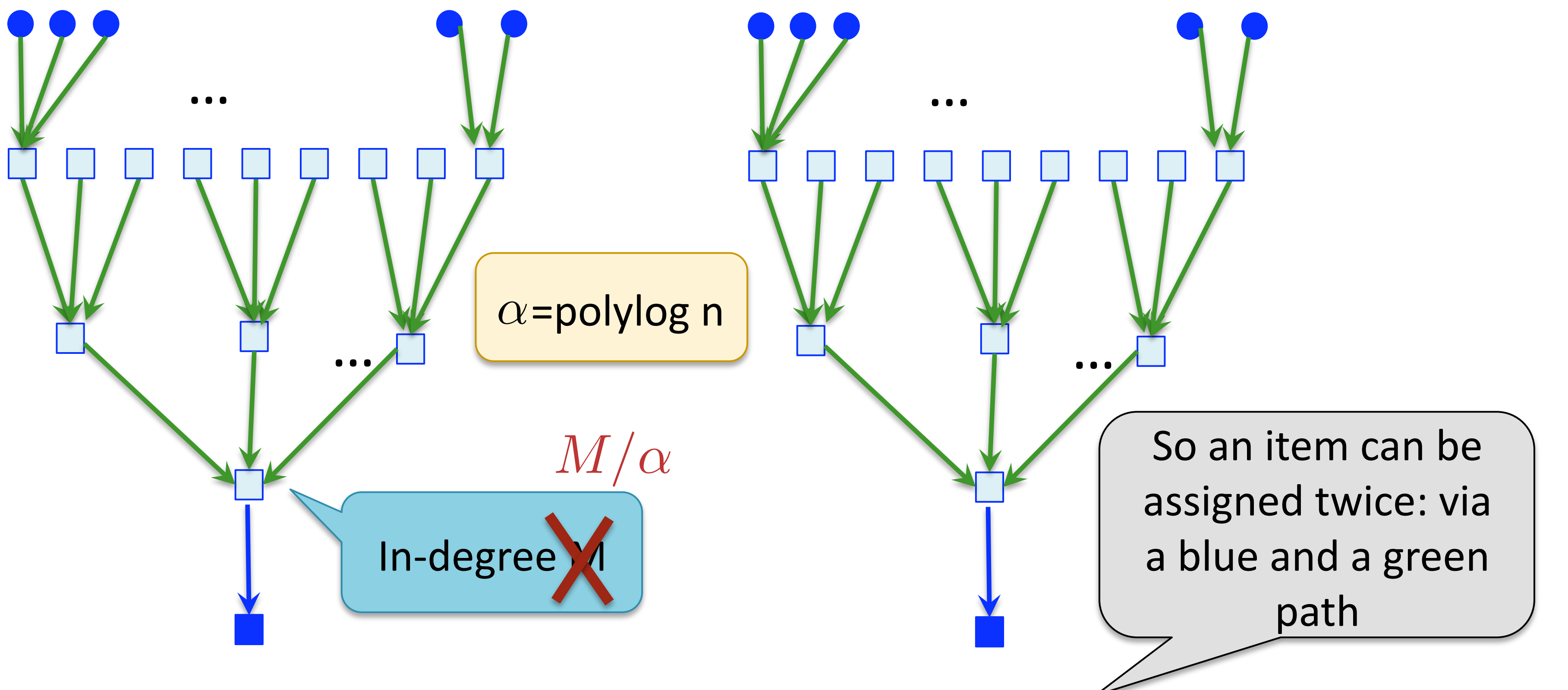
On Green and Blue Flow-Paths

- Behave very differently
- Green paths: a lot of flexibility
  - Even if we remove half the flow-paths entering every agent A, will still get a good solution.
- Can't do the same with blue flow-paths. Need to have 1 flow-path entering each terminal.

Almost Feasible Solutions



Almost Feasible Solutions

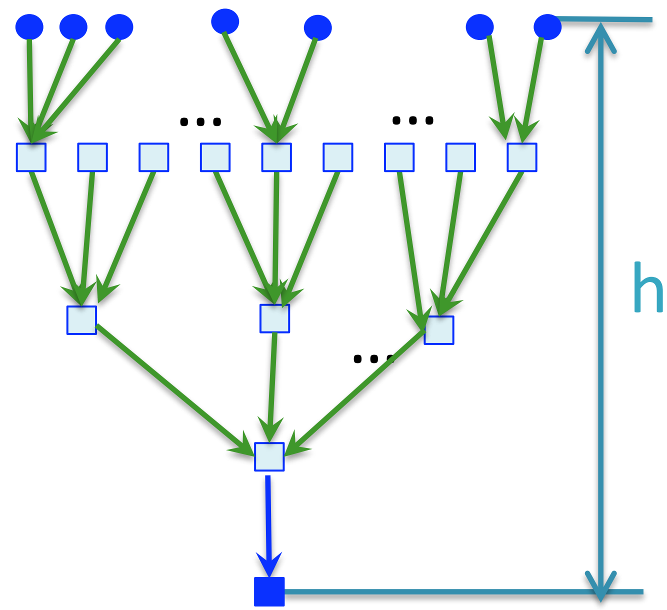


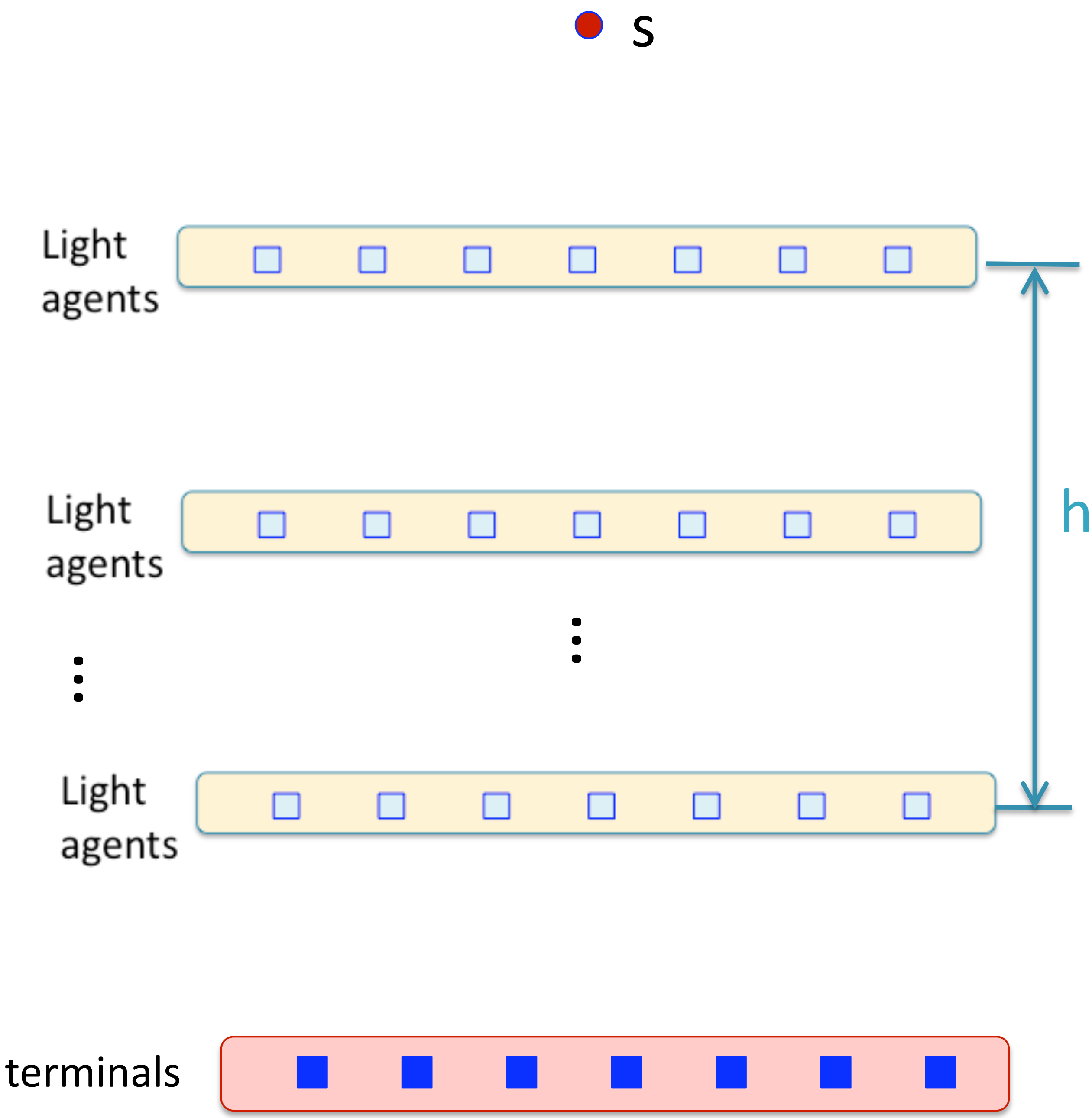
Flow directly to terminals
Flow to light agents

An item/heavy agent may appear on one blue and one green path.

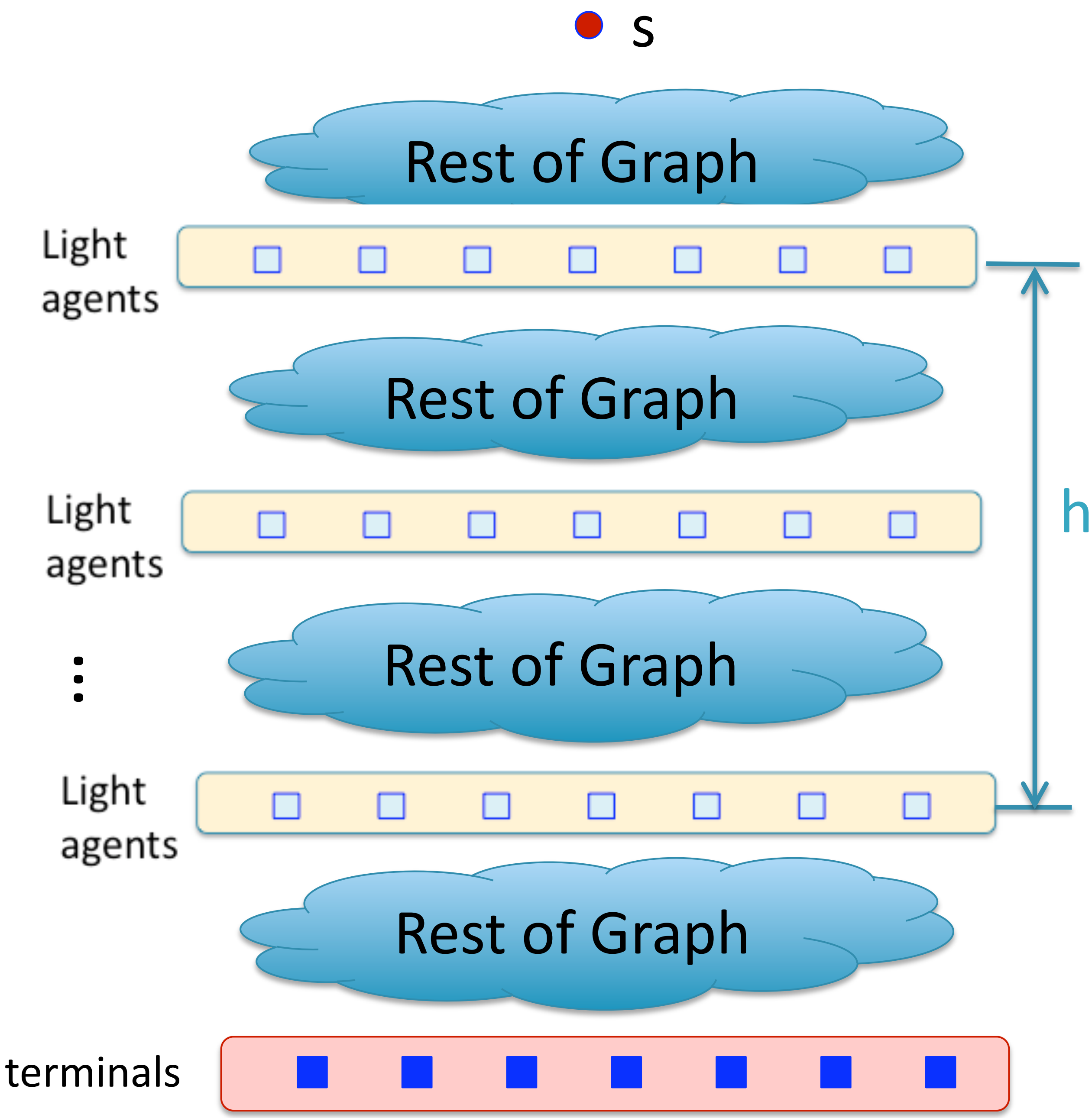
Our LP

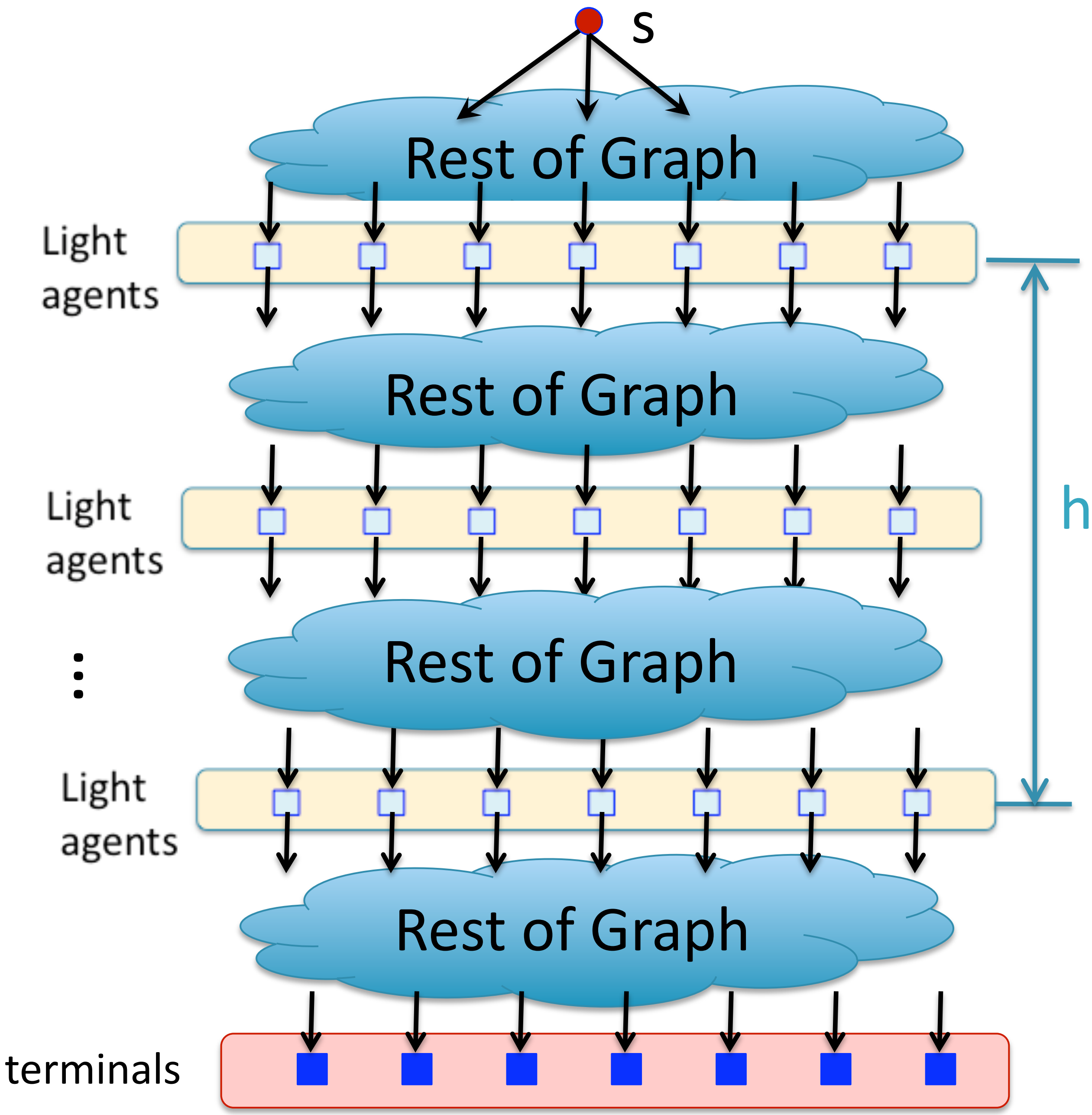
- We don't know which agents will appear in which layer
  - Make h copies of the graph





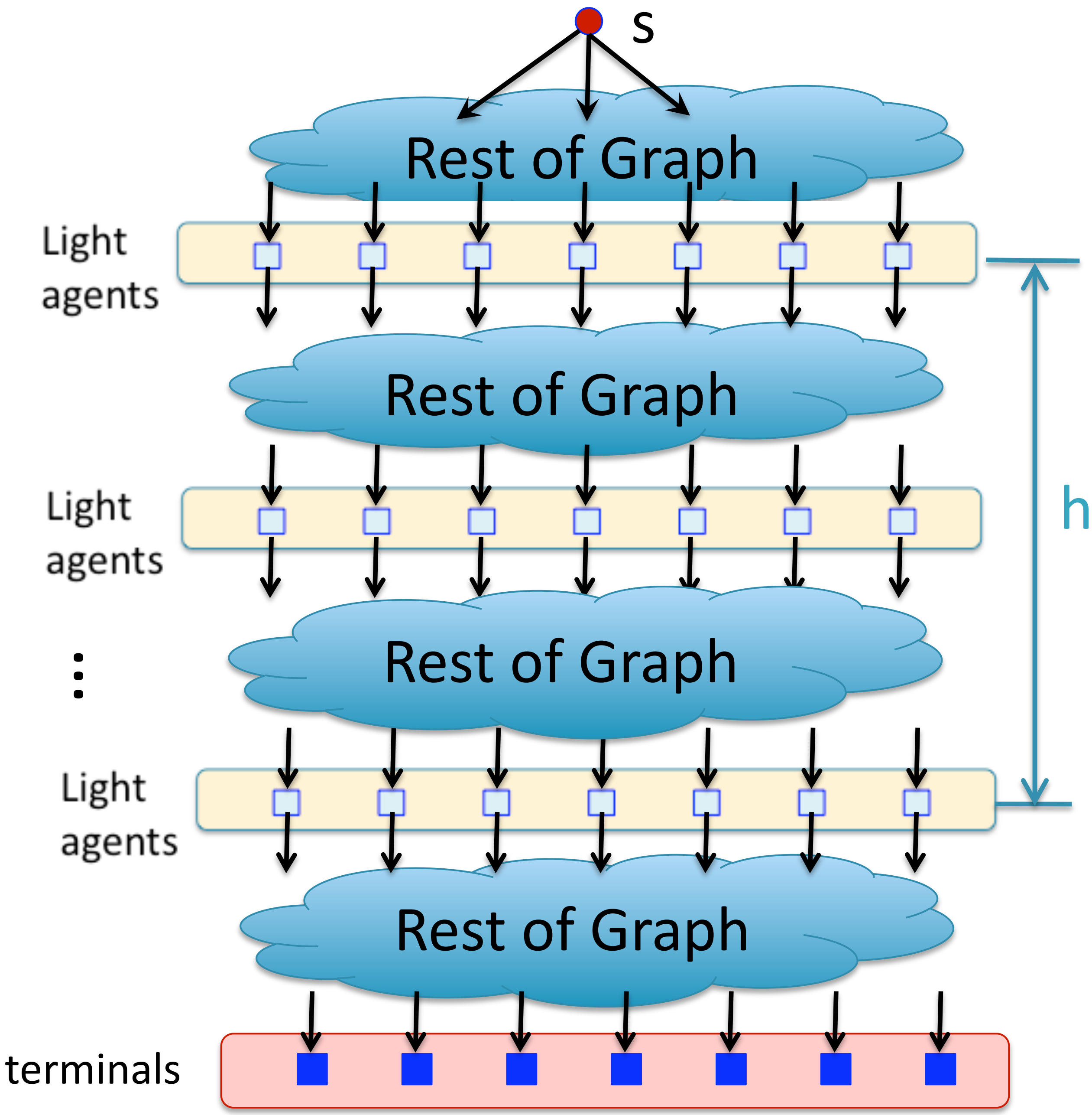
s
Light agents
Light agents
Light agents
h
terminals

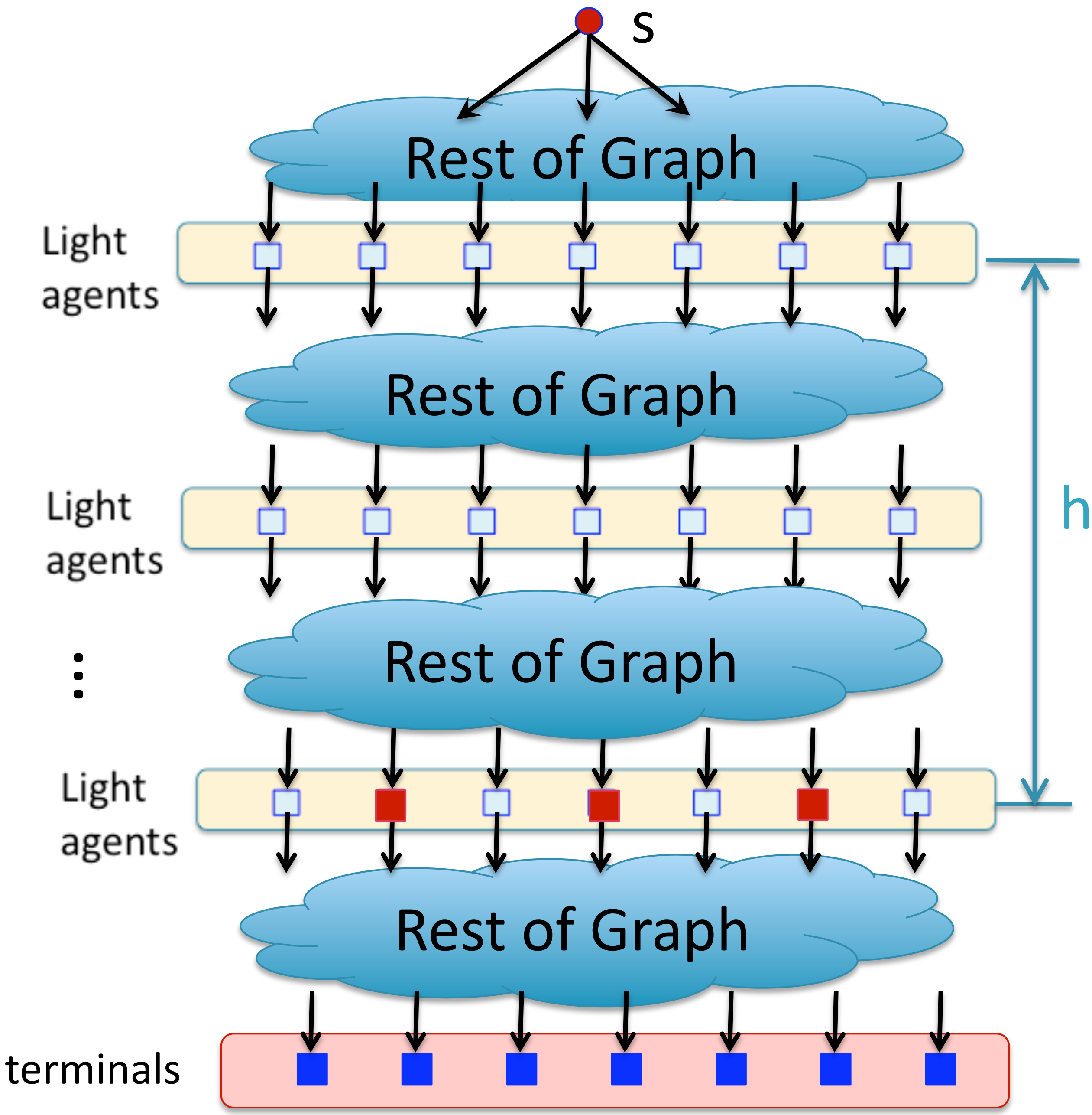

s
Rest of Graph
Light agents
Rest of Graph
Light agents
h
Rest of Graph
Light agents
Rest of Graph
terminals


s
Rest of Graph
Light agents
Rest of Graph
Light agents
Rest of Graph
Light agents
Rest of Graph
terminals
h

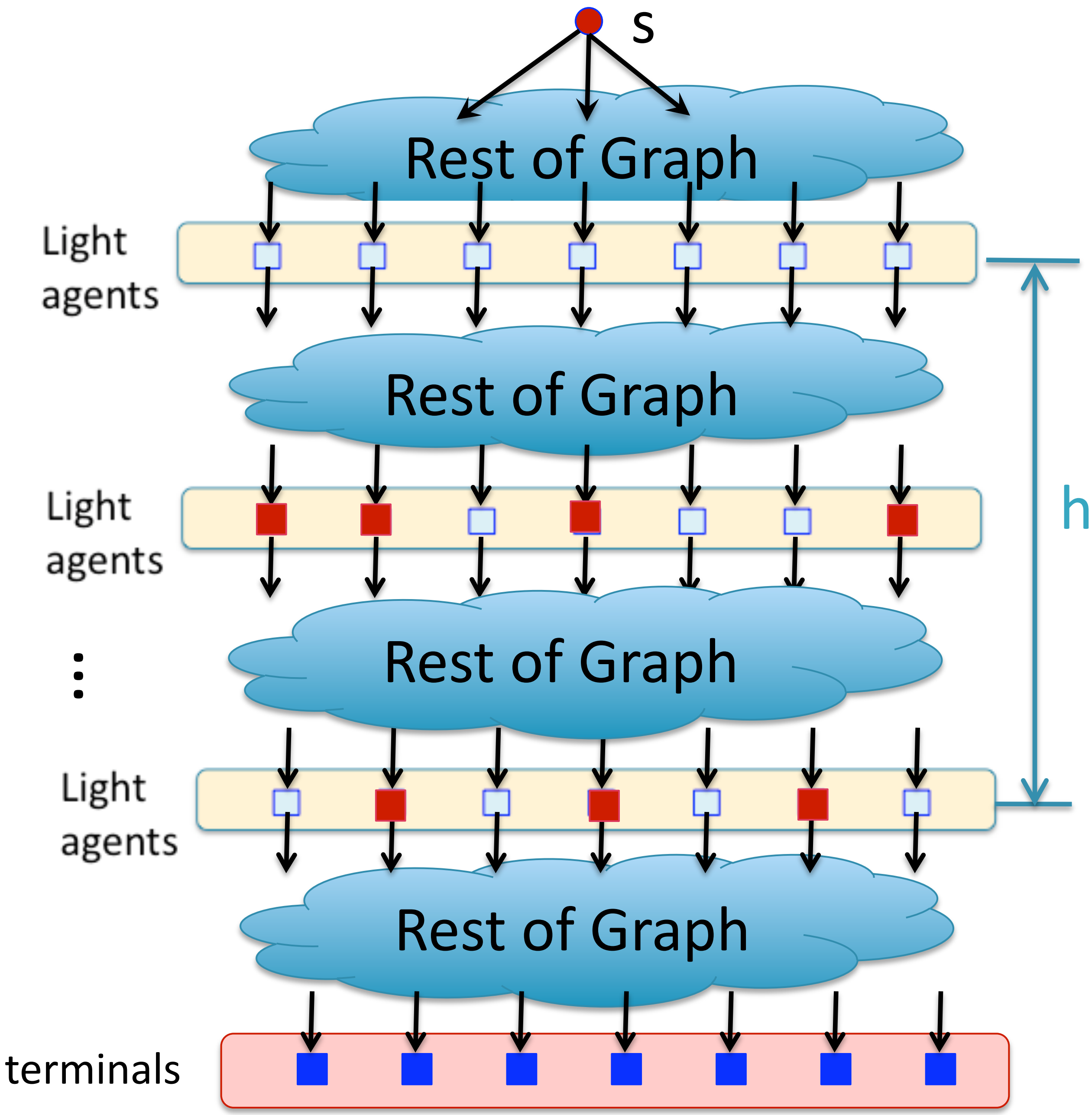
LP-rounding

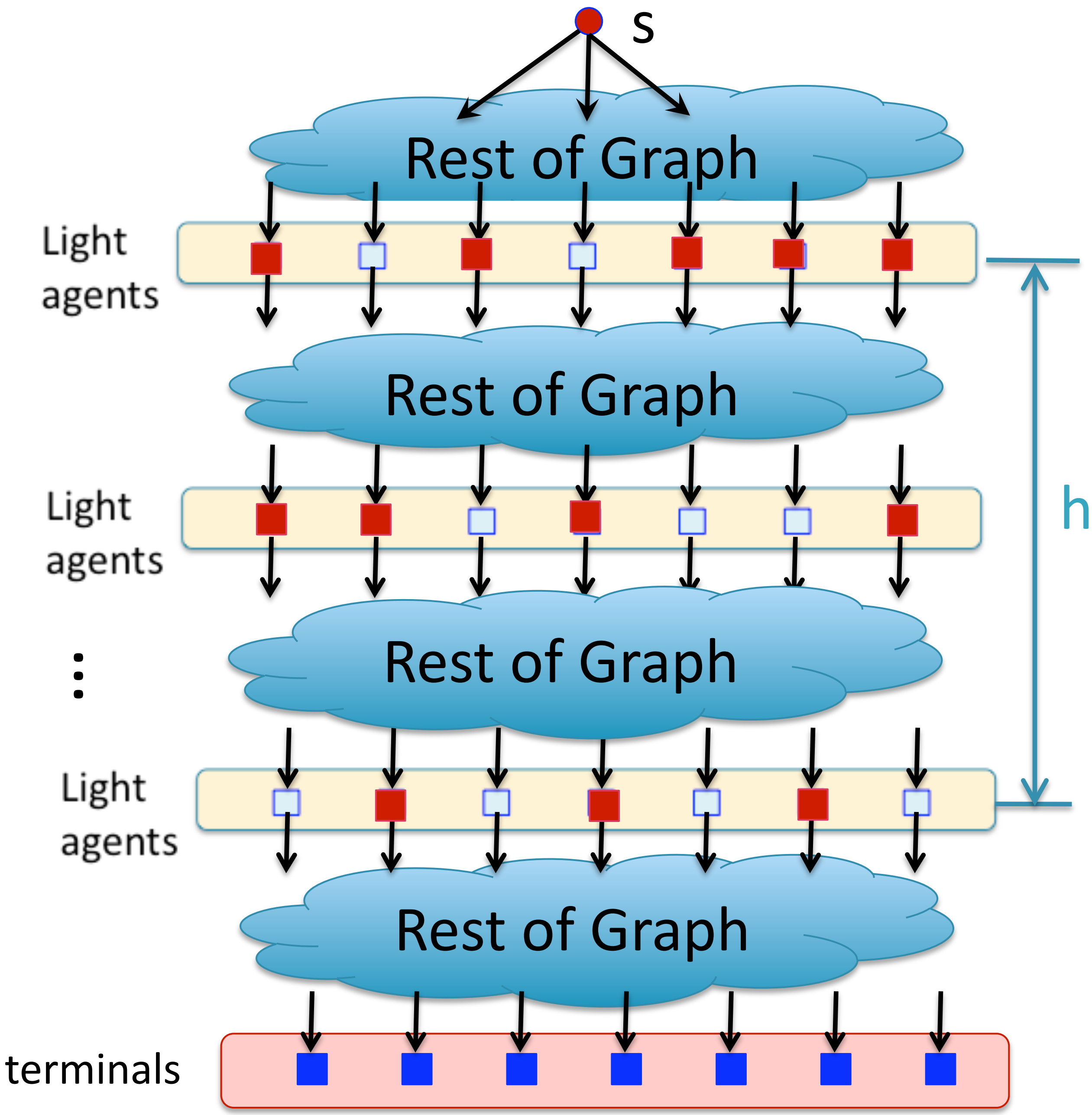
- Blue paths:
  - Can select via Randomized Rounding a set of disjoint paths connecting every terminal to a light agent
  - Use a procedure of Bansal and Sviridenko.
- Green paths:
  - Perform Randomized Rounding layer by layer.


s
Rest of Graph
Light agents
Rest of Graph
Light agents
Rest of Graph
Light agents
Rest of Graph
terminals
h



s
Rest of Graph
Light agents
Rest of Graph
Light agents
⋮
Rest of Graph
Light agents
Rest of Graph
terminals
h

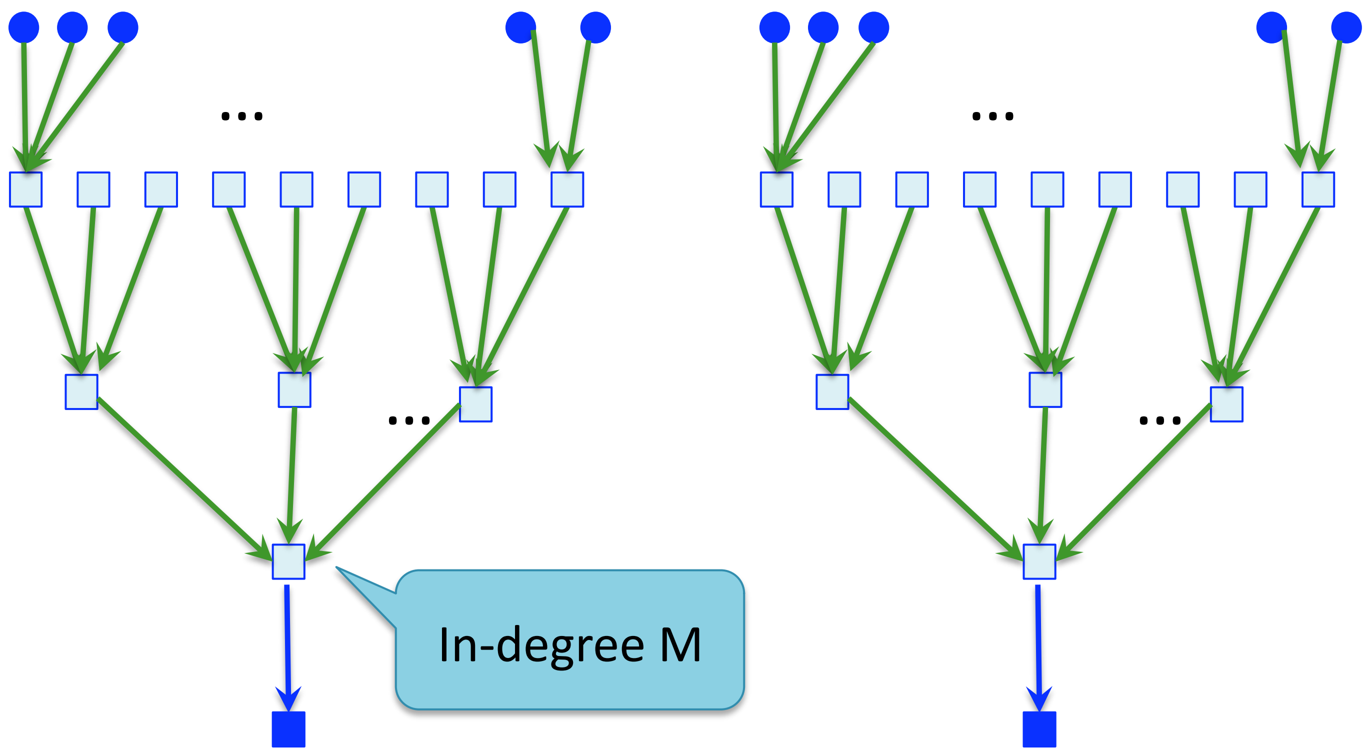

s
Rest of Graph
Light agents
Rest of Graph
Light agents
⋮
Rest of Graph
h
Light agents
Rest of Graph
terminals


s
Rest of Graph
Light agents
Rest of Graph
Light agents
Rest of Graph
Light agents
Rest of Graph
terminals
h

LP-rounding

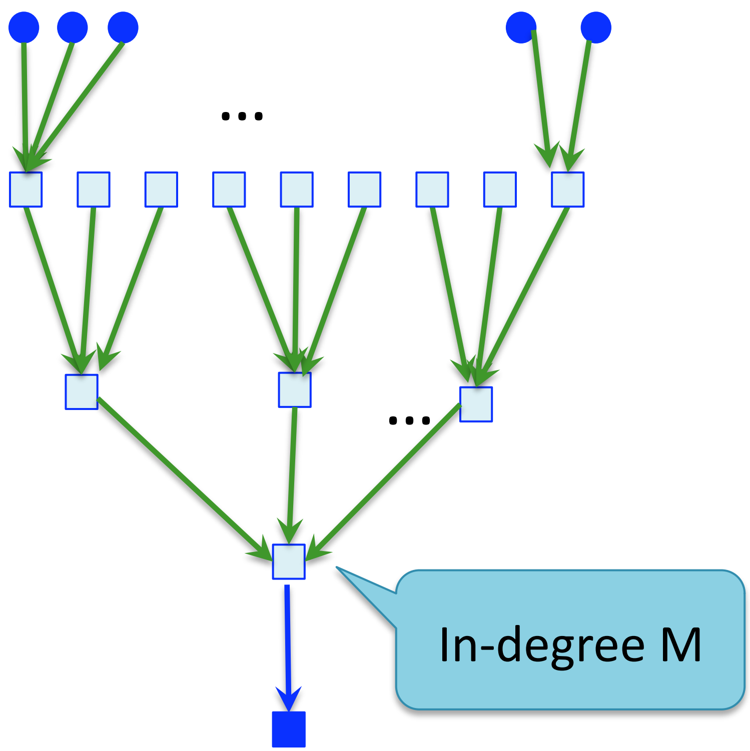
- Using the new capacity constraints, can show that congestion is bounded by polylog n, even when taking into account the h copies of every agent/item.
- So each item/heavy agent participates in at most polylog n green paths and at most one blue path w.h.p.
- Last step: get rid of congestion among green paths.
- Use a flow scaling trick.

Flow scaling trick



Problem: Some agents and items appear on poly(log n) green paths.

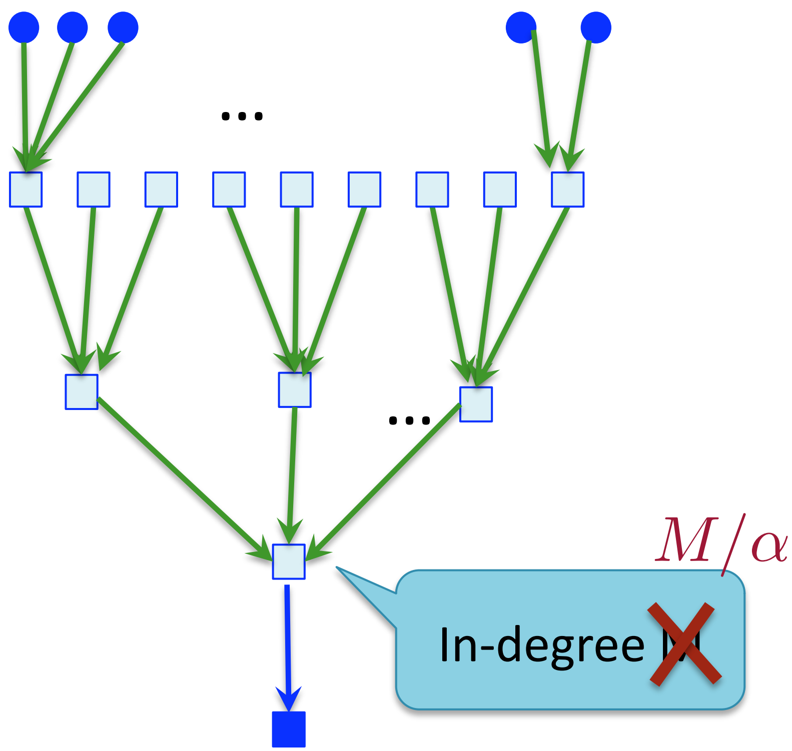
Flow scaling trick



Problem: Some agents and items appear on poly(log n) green paths.

- Scale flow down by α=polylog n factor.

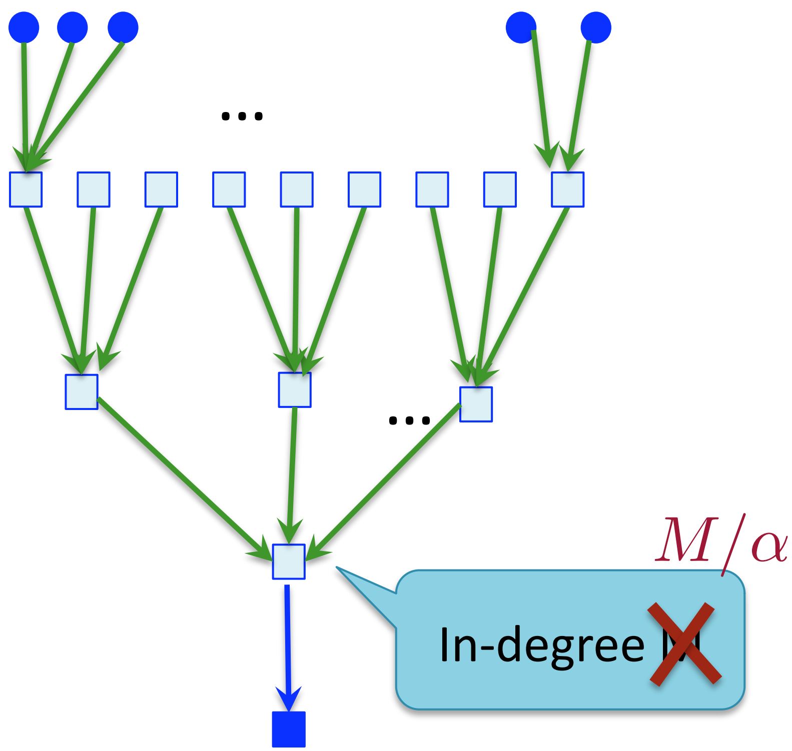
Flow scaling trick



Problem: Some agents and items appear on poly(log n) green paths.

- Scale flow down by α=polylog n factor.
- We get α-approximate fractional solution with no congestion.

Flow scaling trick



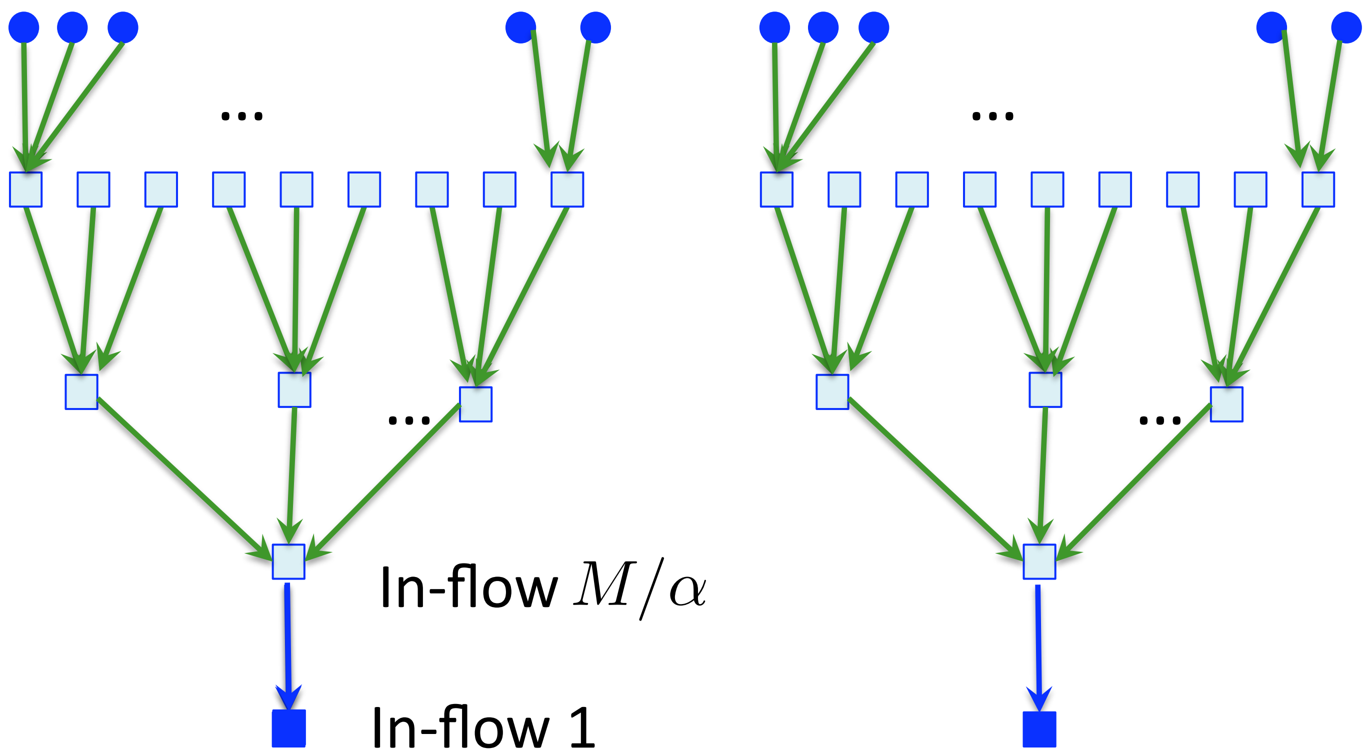
Problem: Some agents and items appear on poly(log n) green paths.

- Scale flow down by α=polylog n factor.
- We get α-approximate fractional solution with no congestion.
- From integrality of flow can find such integral solution.

(Need to set up a single source-single sink flow network).

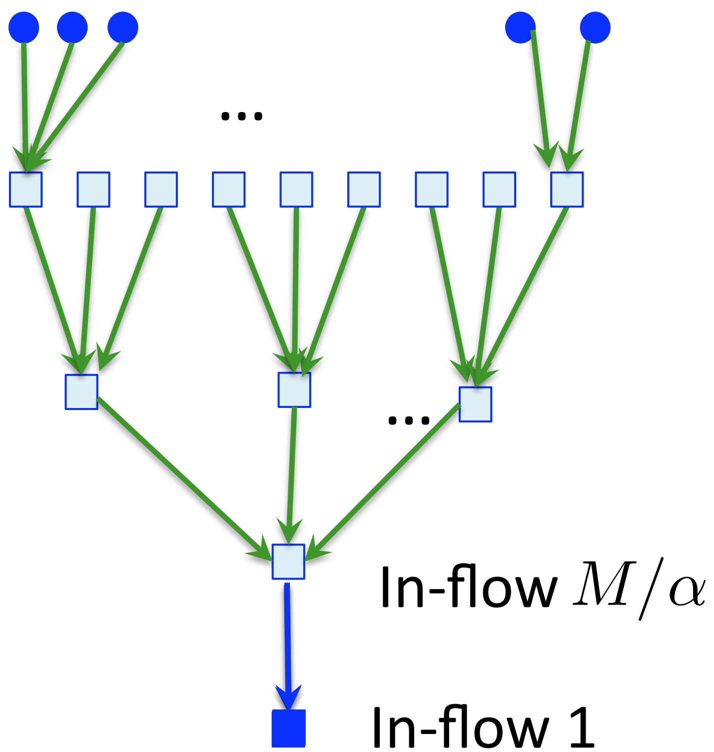
Why can't we use the flow scaling trick to get a feasible solution from an almost-feasible one?

Flow Scaling for Almost-Feasible Solutions



Problem: heavy agent/item may appear on a blue and a green path

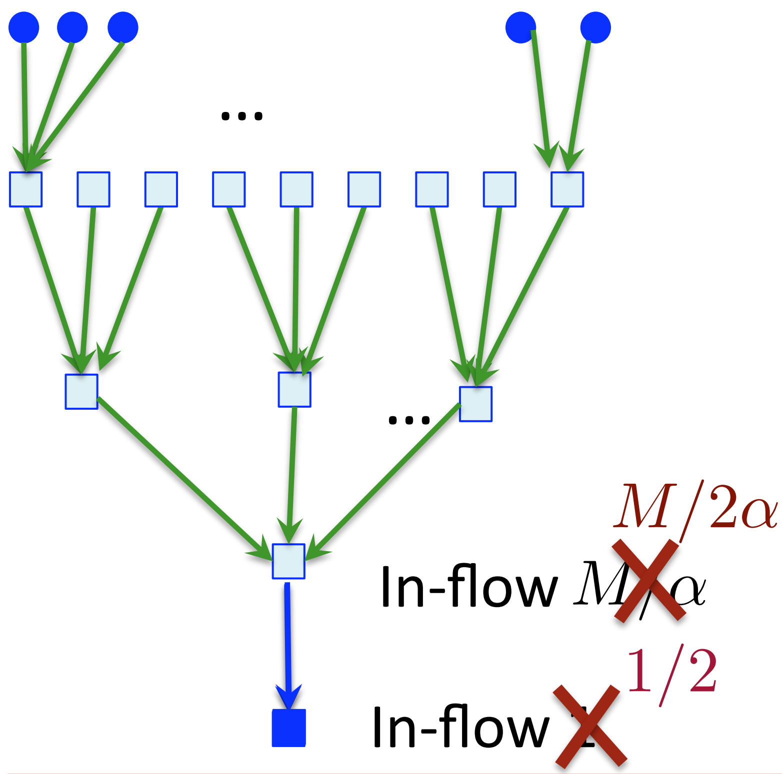
Flow Scaling for Almost-Feasible Solutions



Problem: heavy agent/ item may appear on a blue and a green path

- Scale the flow down by factor 2.
- We get “2-approximate” fractional solution with no congestion.
- From integrality of flow can find such integral solution.

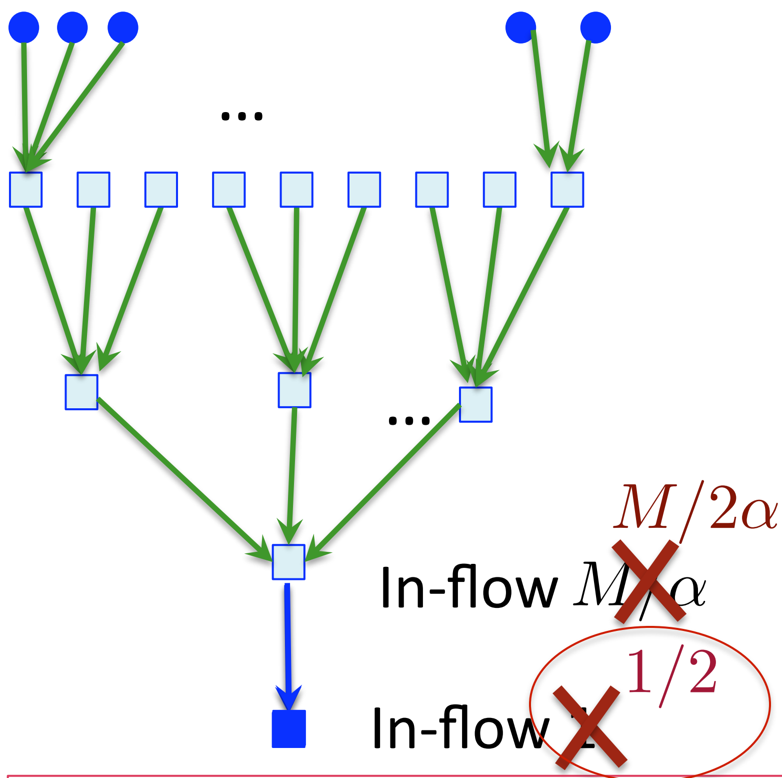
Flow Scaling for Almost-Feasible Solutions



- Scale the flow down by factor 2.
- We get "2-approximate" fractional solution with no congestion.
- From integrality of flow can find such integral solution.

Problem: heavy agent/item may appear on a blue and a green path

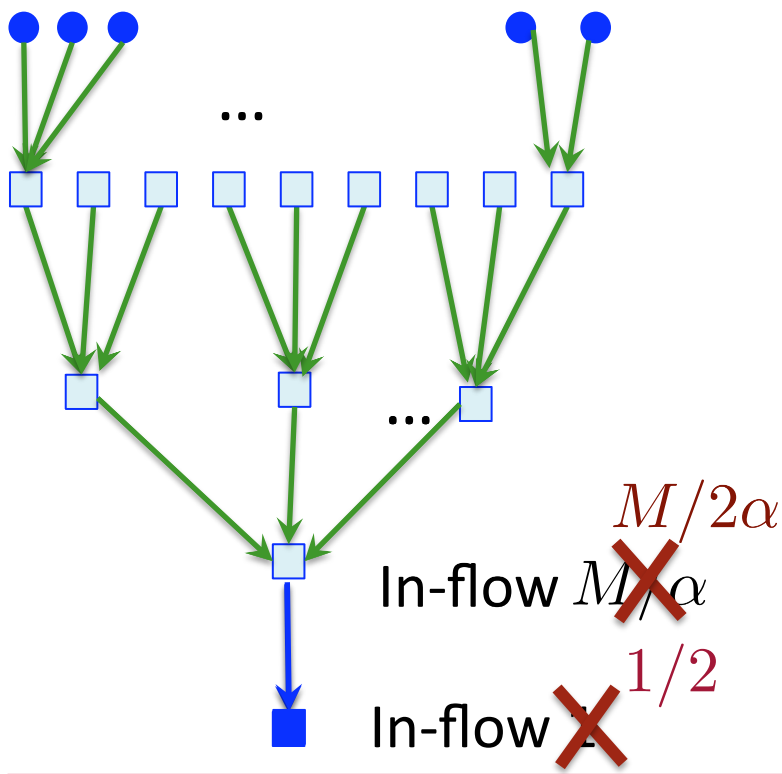
Flow Scaling for Almost-Feasible Solutions



- Scale the flow down by factor 2.
- We get "2-approximate" fractional solution with no congestion.
- ~~From integrality of flow can find such integral solution.~~

Problem: heavy agent/item may appear on a blue and a green path

Flow Scaling for Almost-Feasible Solutions

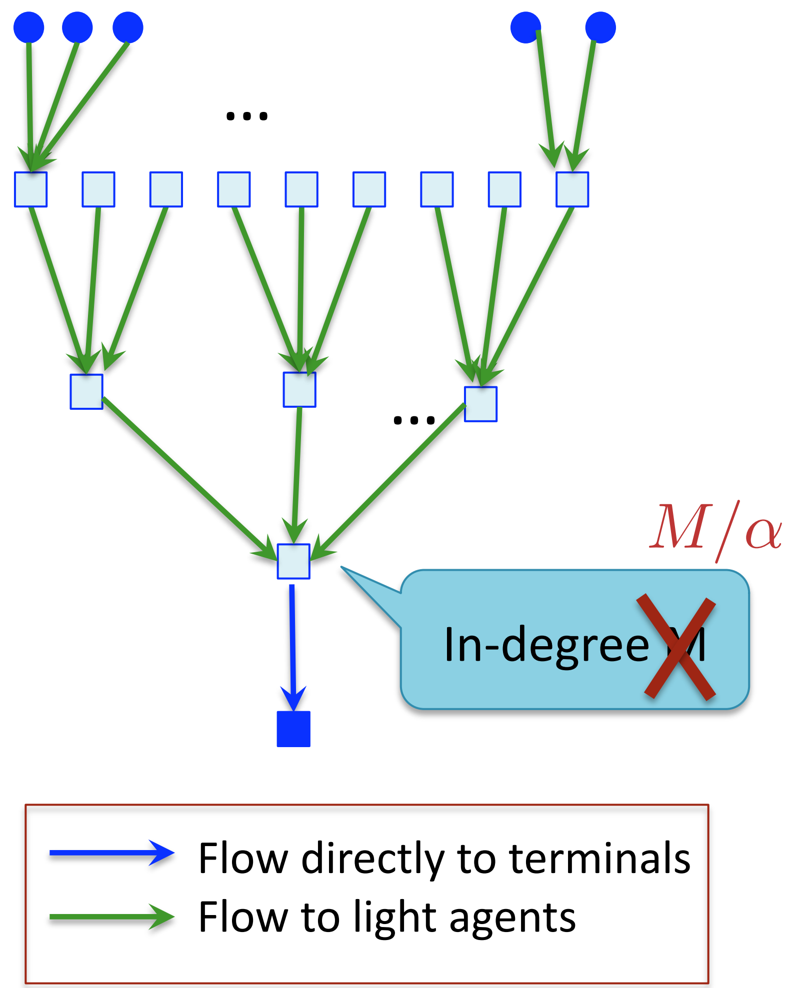


- Scale the flow down by factor 2.
- We get "2-approximate" fractional solution with no congestion.
- ~~From integrality of flow can find such integral solution.~~

Problem: heavy agent/item may appear on a blue and a green path

The LP's integrality gap is $\sqrt{m}$

Summary of LP-Rounding



We get almost-feasible solution:

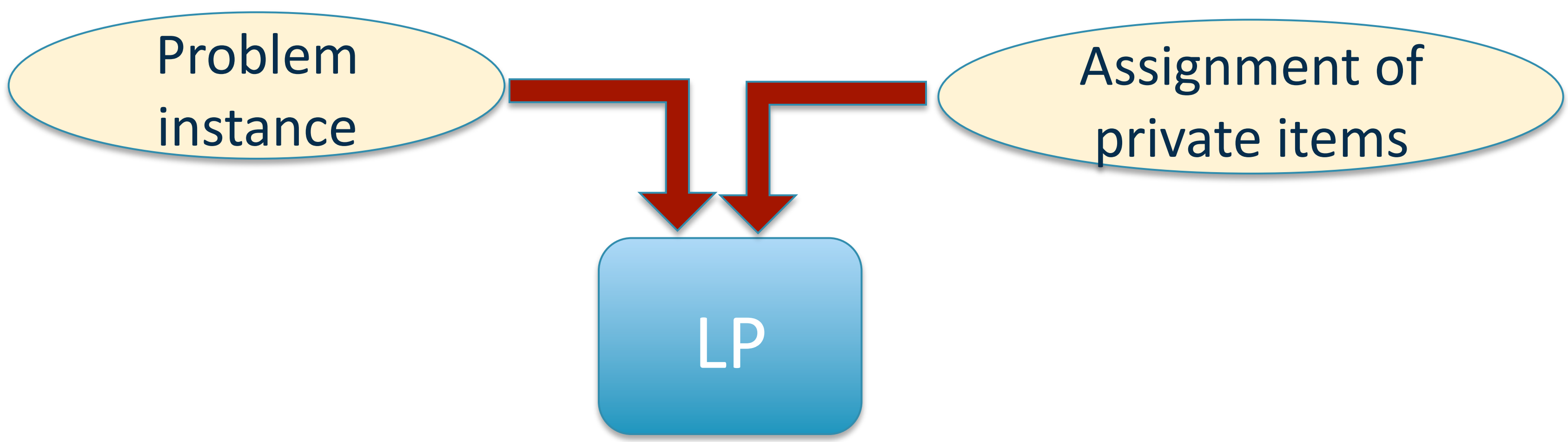
- An item/heavy agent may appear on one blue and one green path.
- Approximation factor: $\alpha = \text{poly} \log n$

Part 2: Getting around the Integrality Gap

Getting around the Integrality Gap

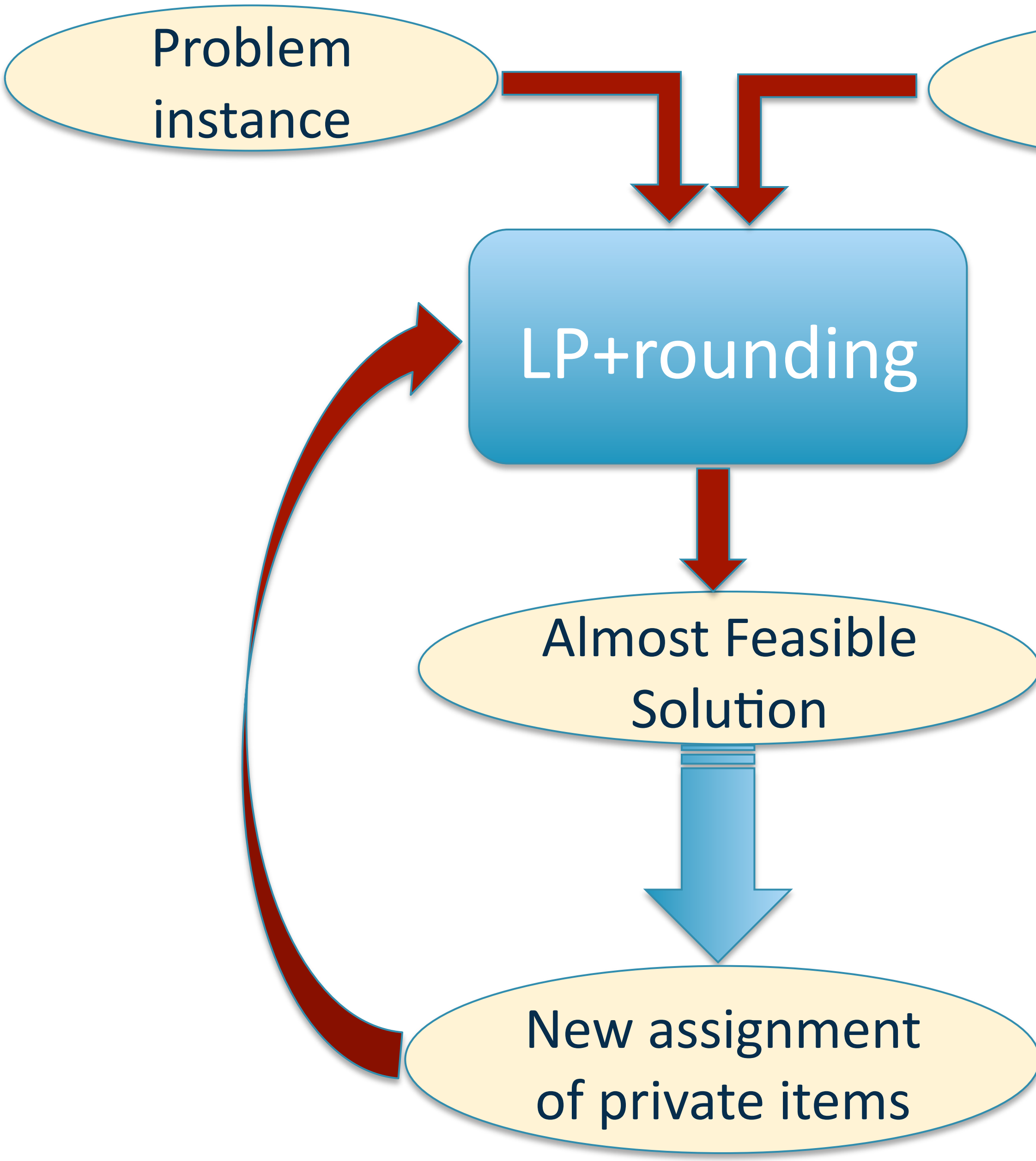
Integrality gap of the LP is $\Omega(\sqrt{m})$

$\Rightarrow$For some inputs to LP the gap is large

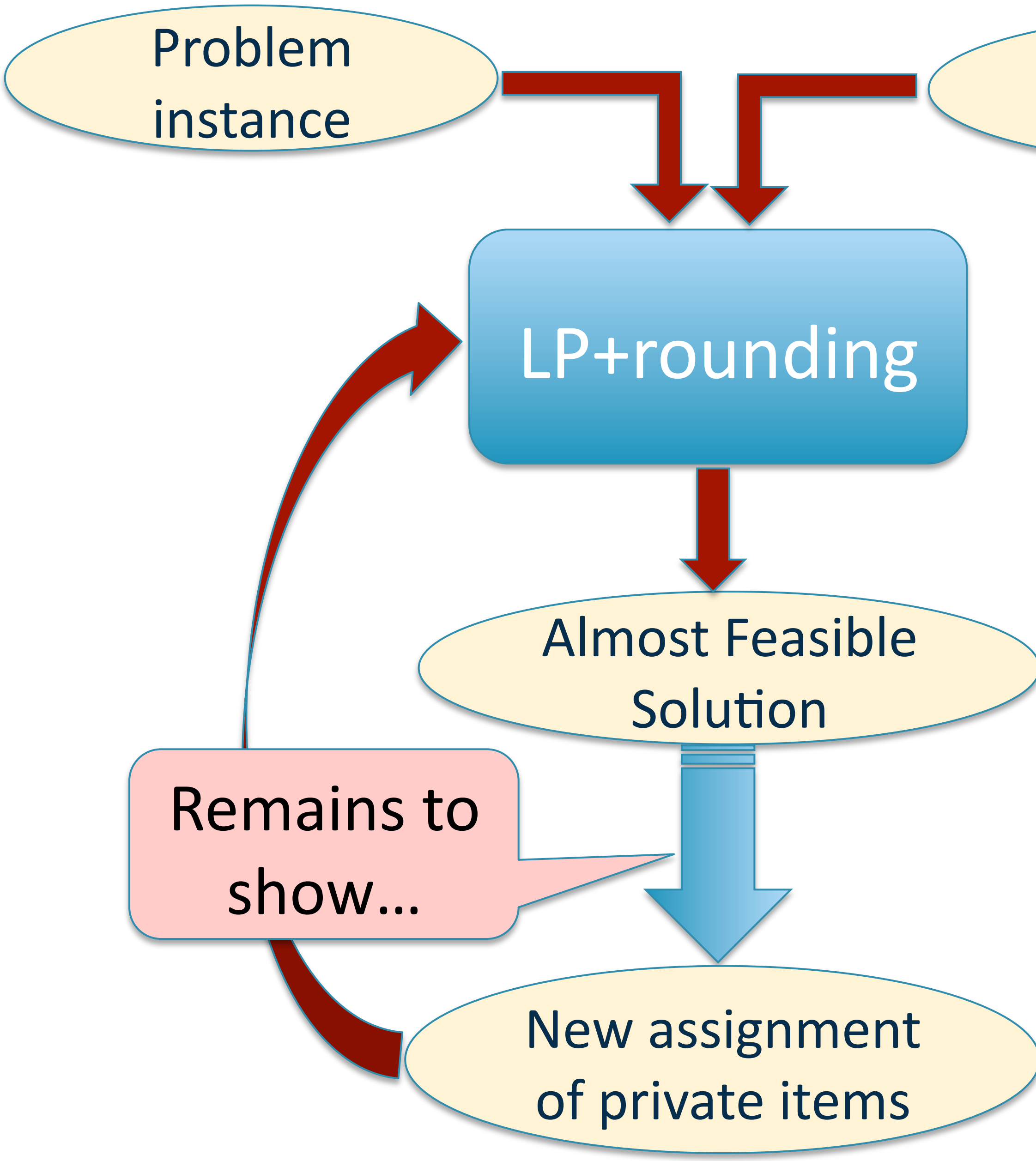


We'll try to find better assignments of private items, so integrality gap goes down.

•LP-rounding is used to find the new assignment!

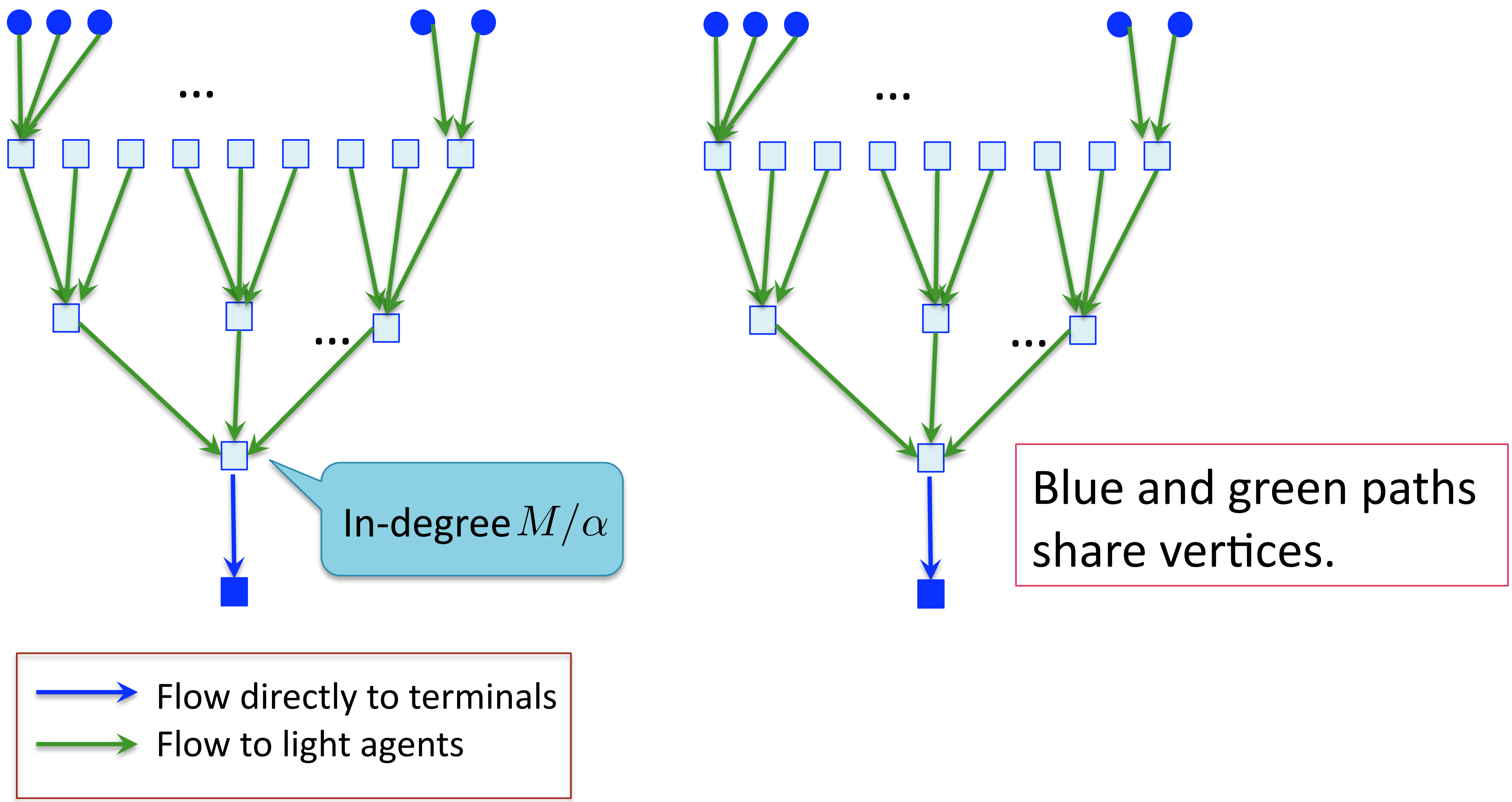


- Repeat $1/\epsilon$ times.
- Number of terminals goes down with each iteration
- Once we have few terminals, LP-rounding gives good solution.

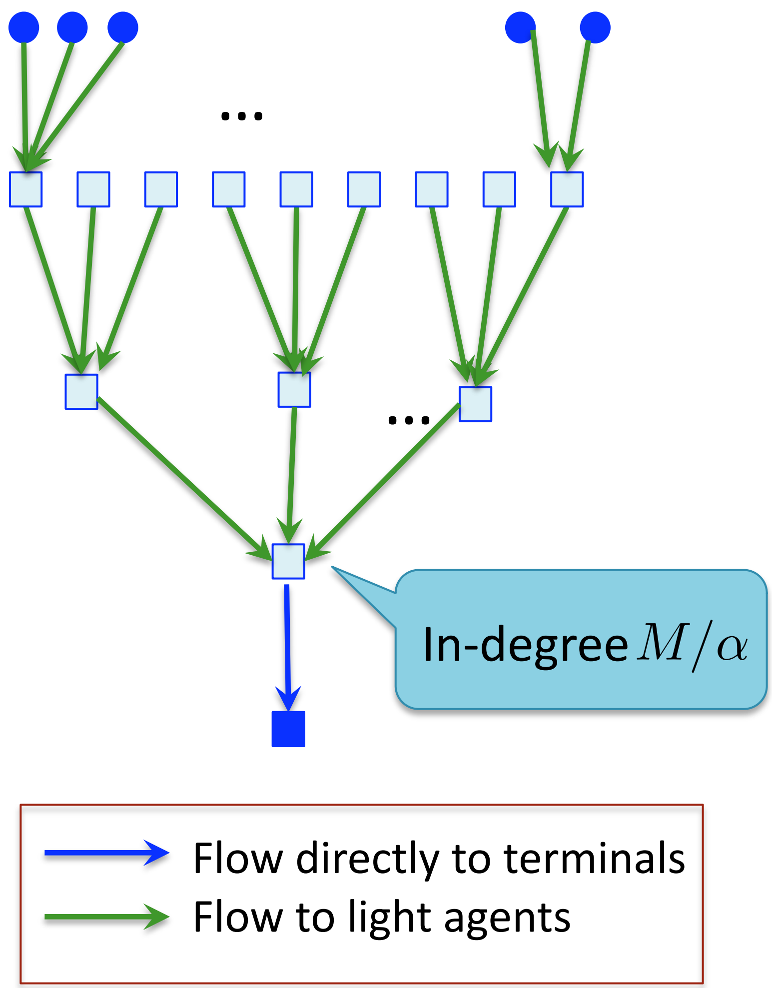


- Repeat $1/\epsilon$ times.
- Number of terminals goes down with each iteration
- Once we have few terminals, LP-rounding gives good solution.

Back to Almost Feasible Solutions

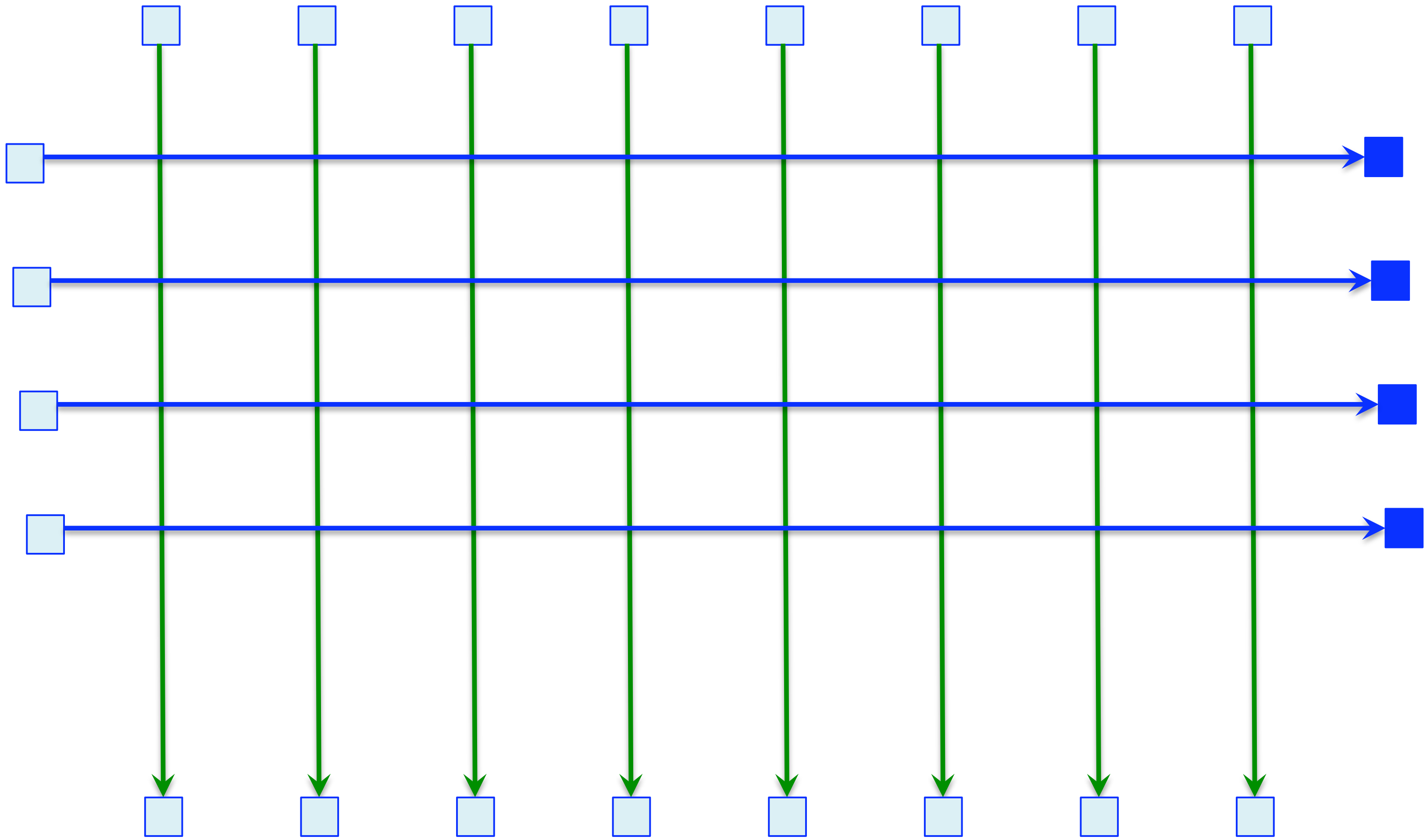


Intuition

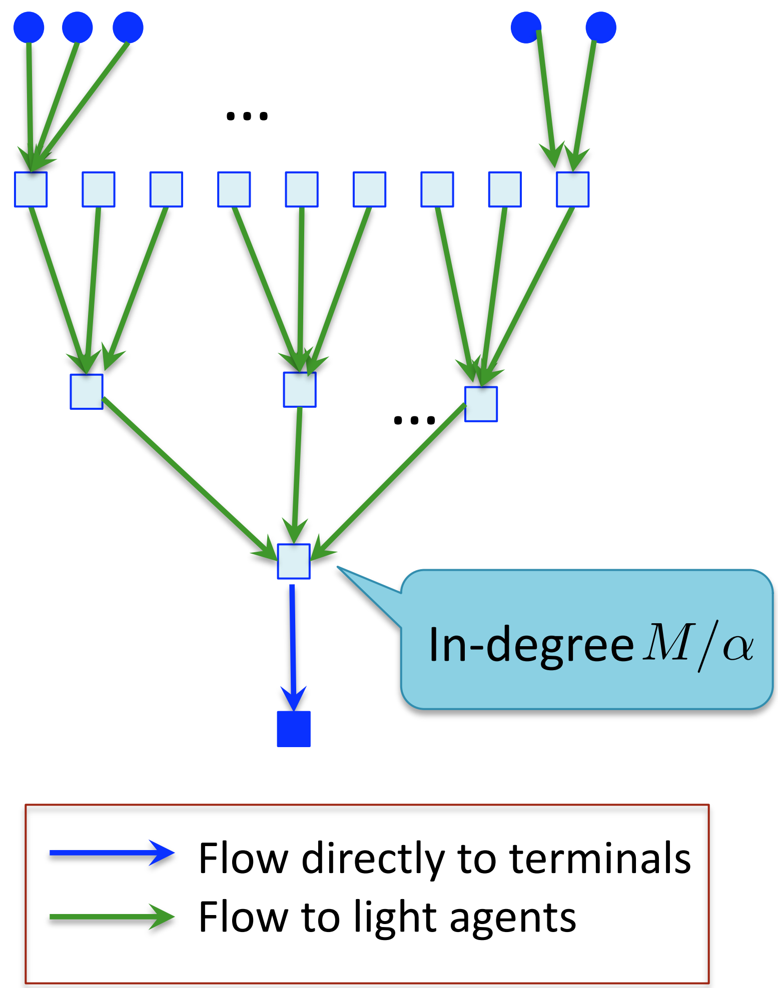


- There are much fewer blue paths than green paths.
- But still there could be many intersections between them.

Example



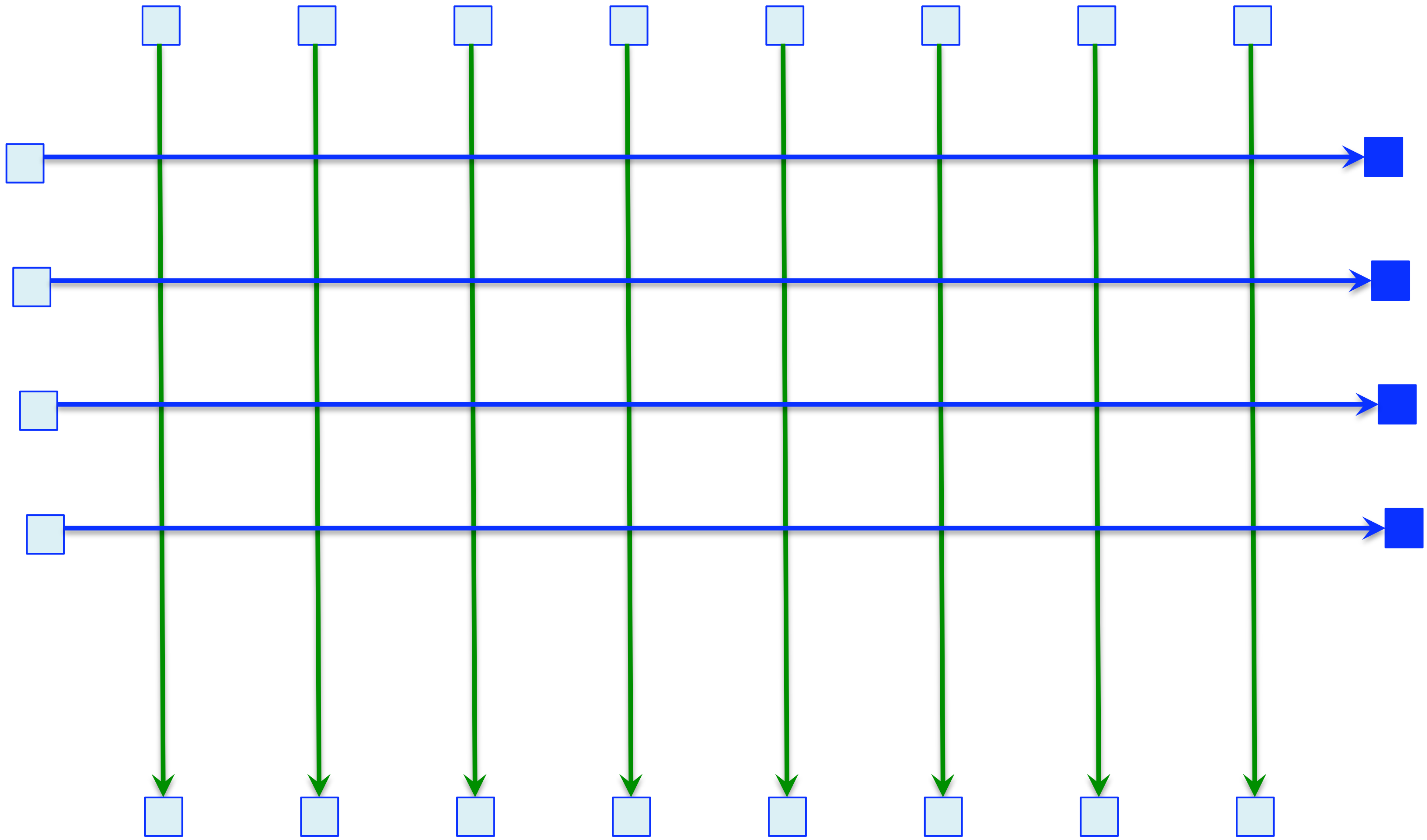
Intuition



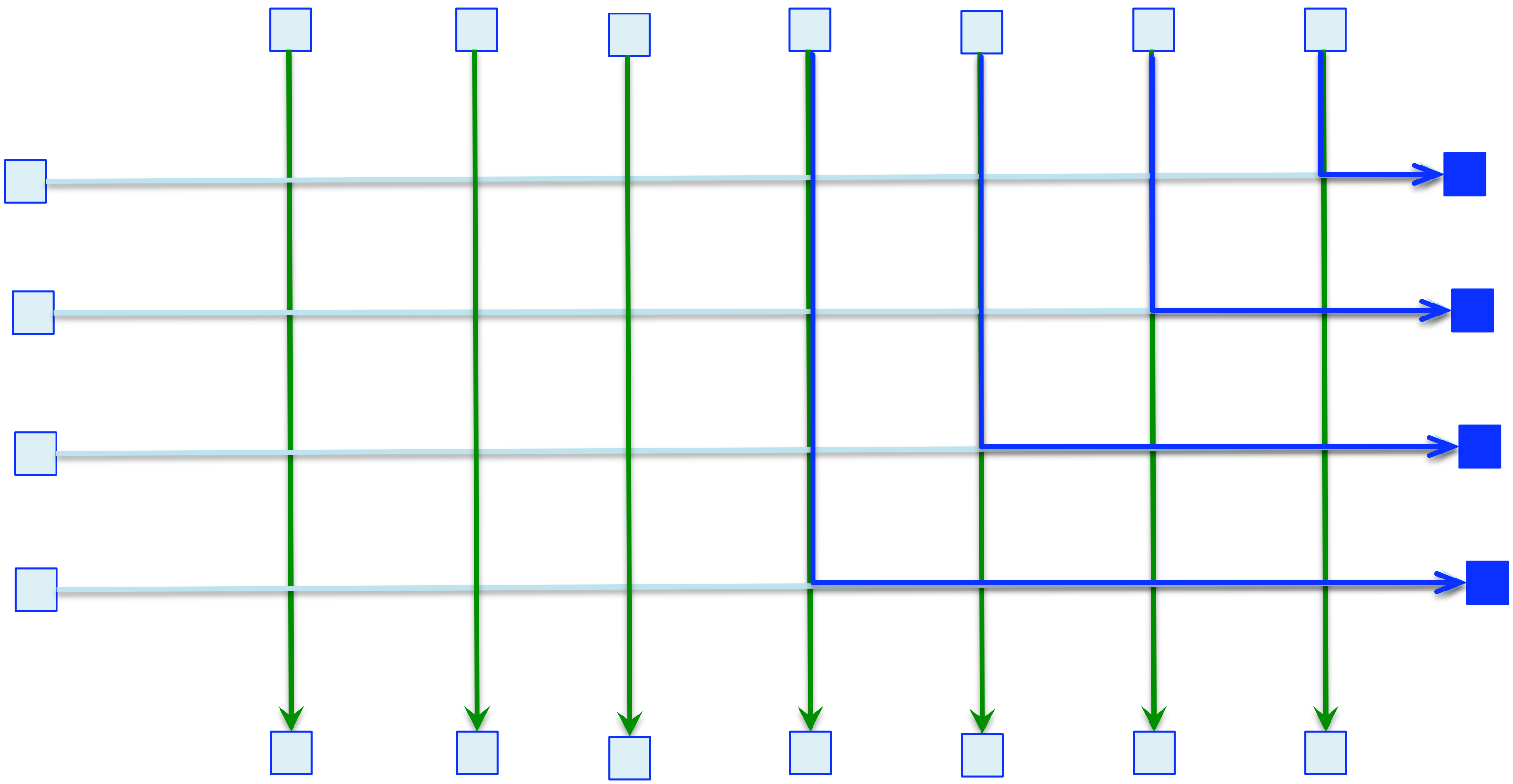
- There are much fewer blue paths than green paths.
- But still there could be many intersections between them.
- Step 1: Re-route blue paths so they intersect few green path.

Notice: it's a single-source flow. Each terminal needs to get a blue flow-path originating at some light agent, doesn't matter which.

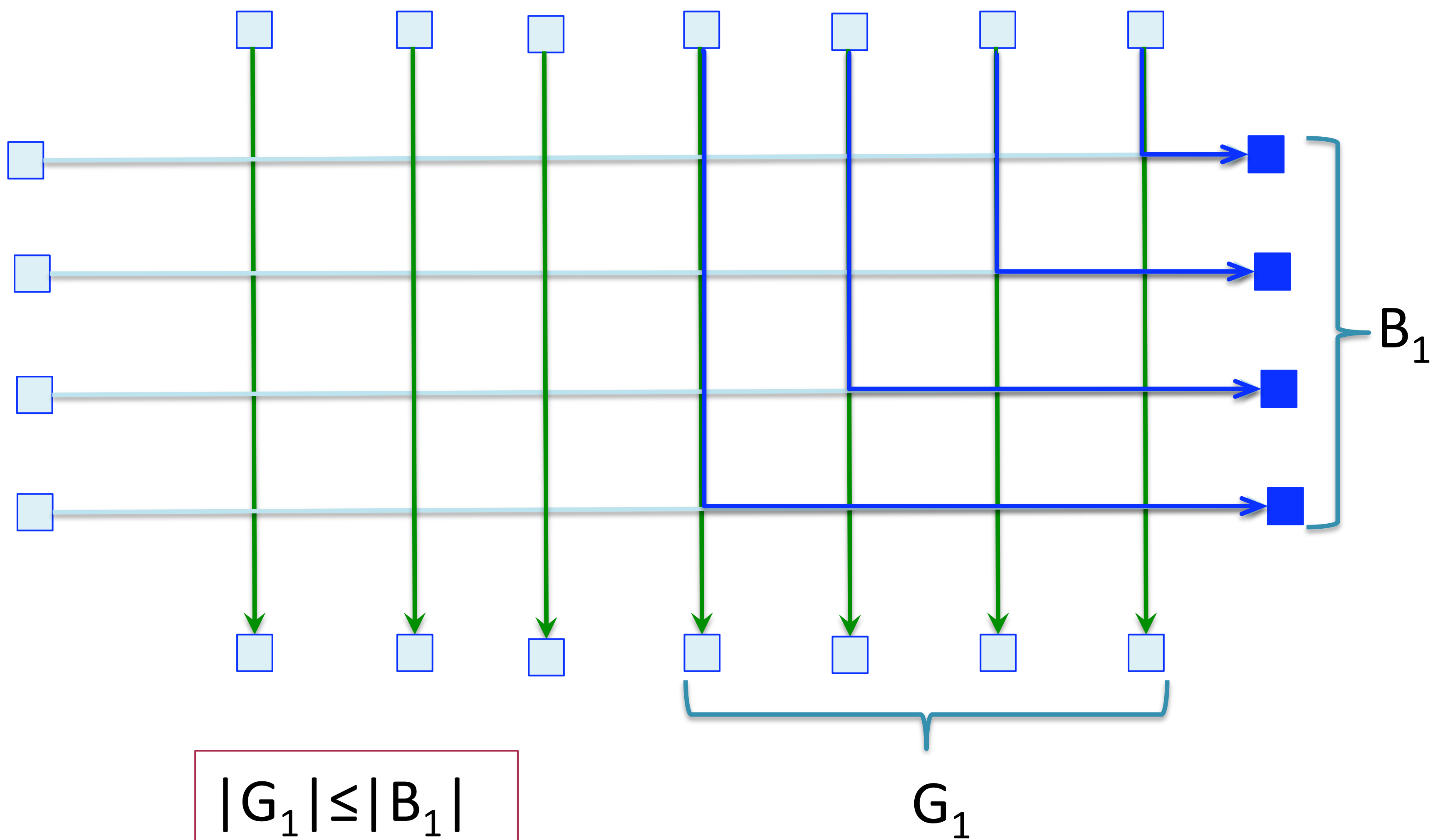
Example



Example

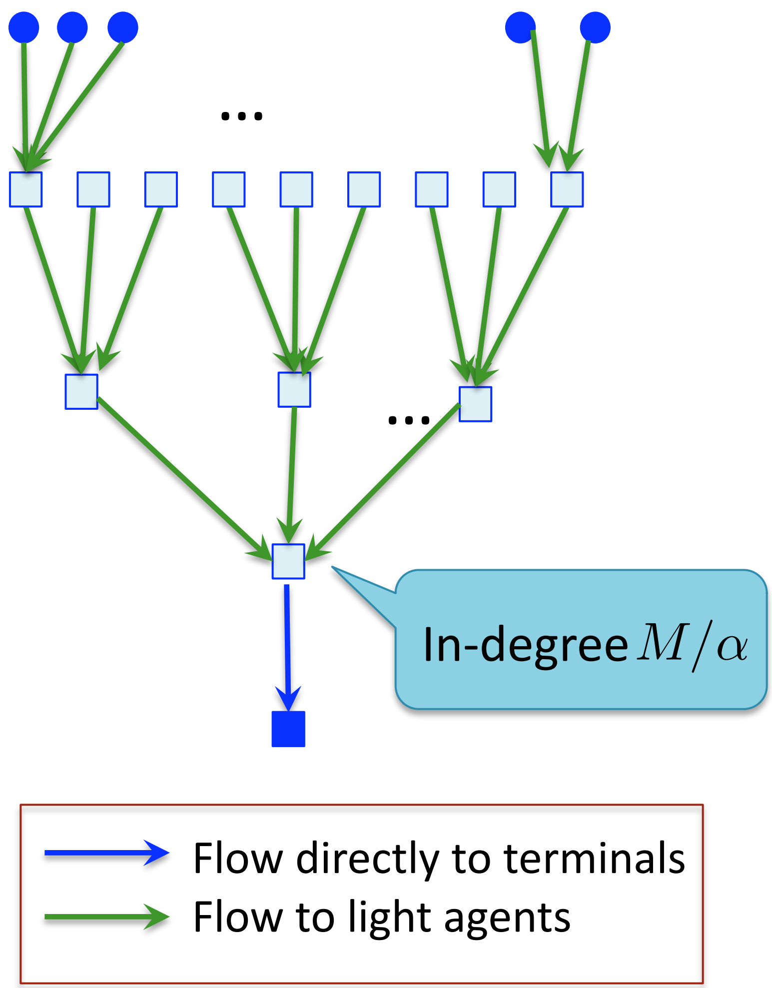


Example



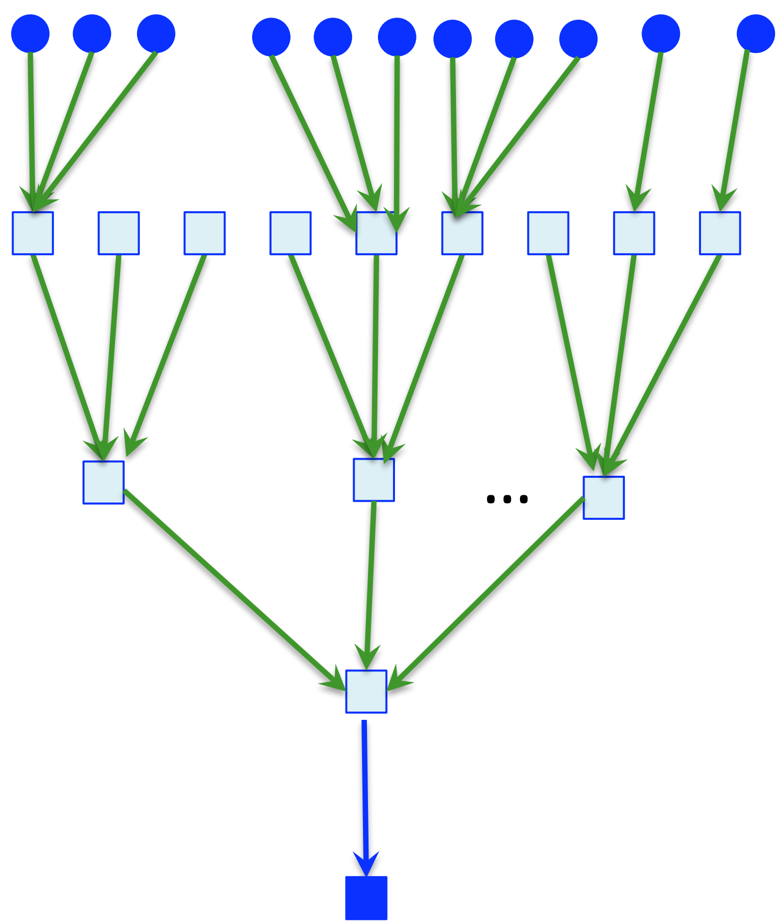
$|G_1| \leq |B_1|$

Intuition



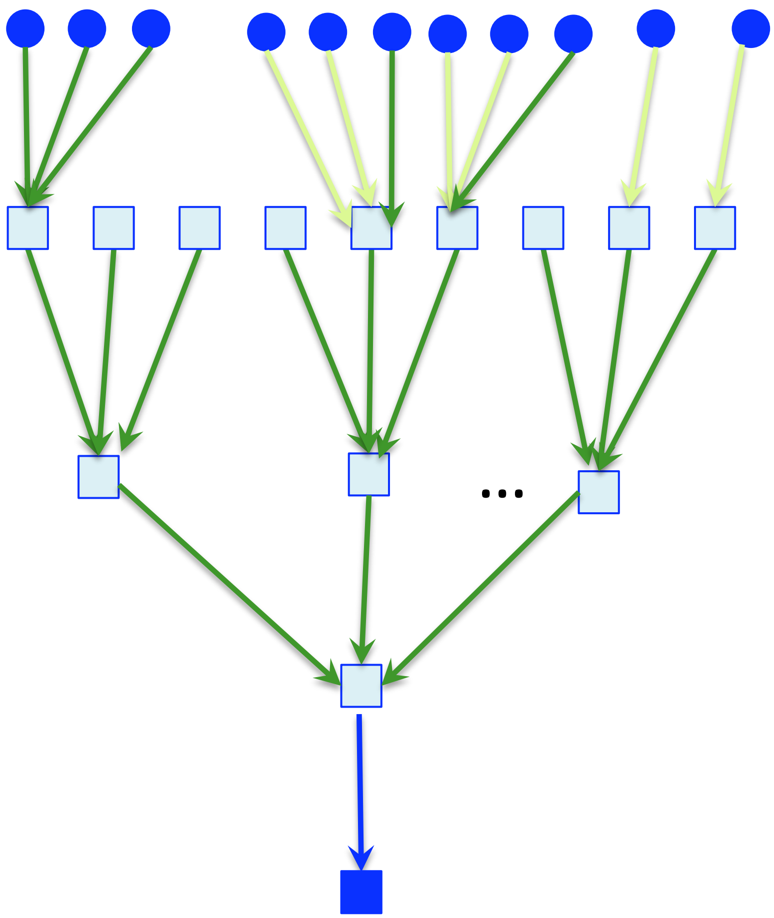
- There are much fewer blue paths than green paths.
- But still there could be many intersections between them.
- Step 1: Re-route blue paths so they intersect few green path.
- Step 2: Remove all green paths in G_1.
  - Few paths are deleted.
  - If each light agent has less than half its paths deleted then we are done.

Intuition



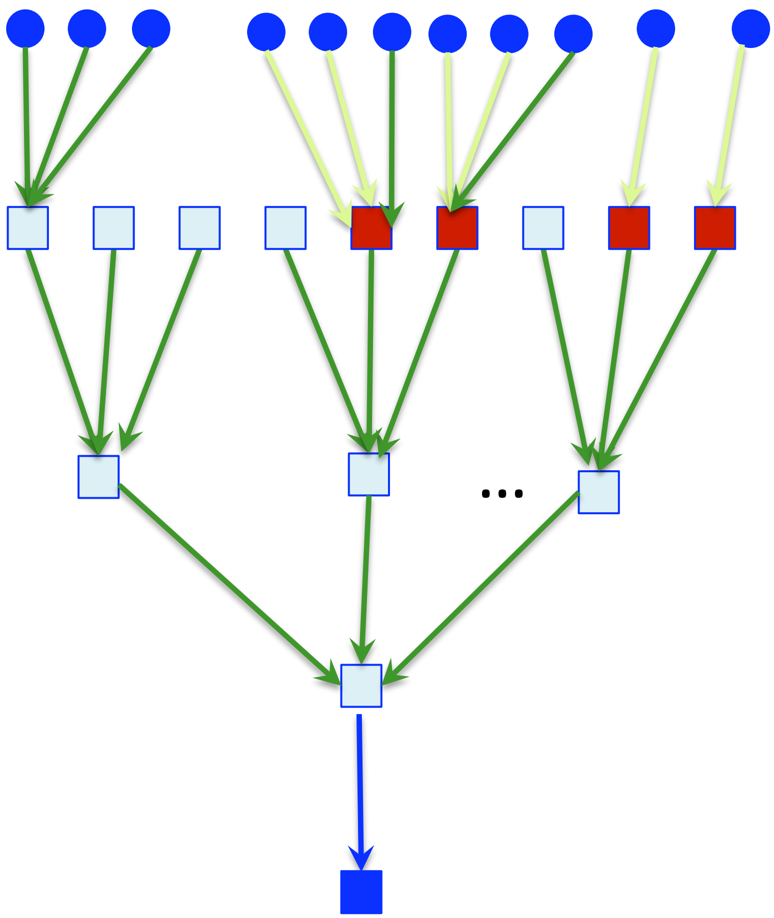
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



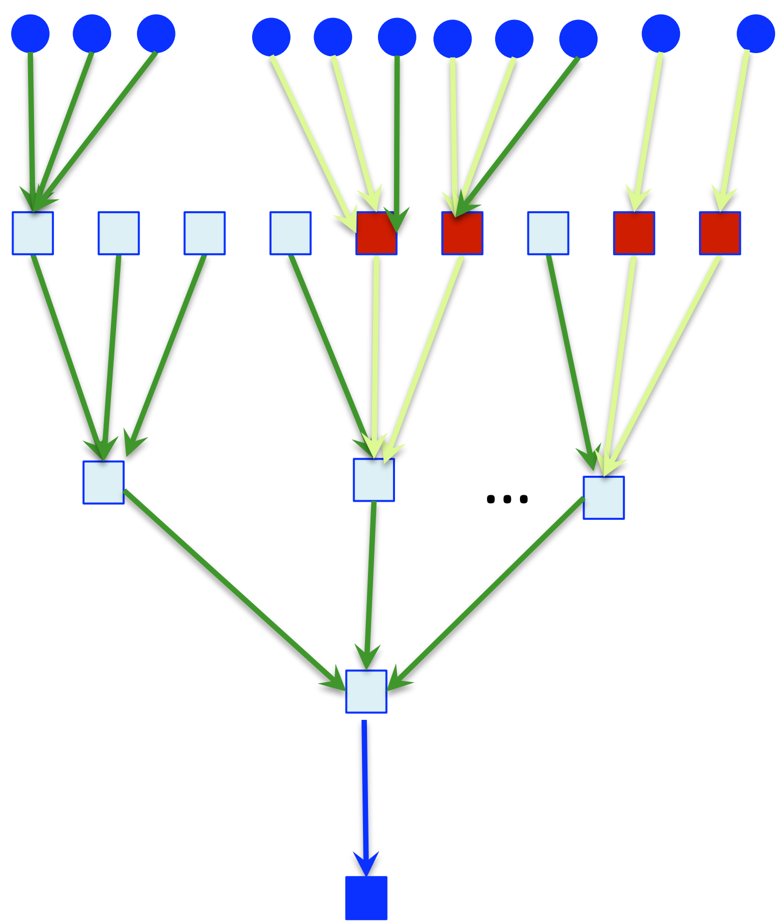
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



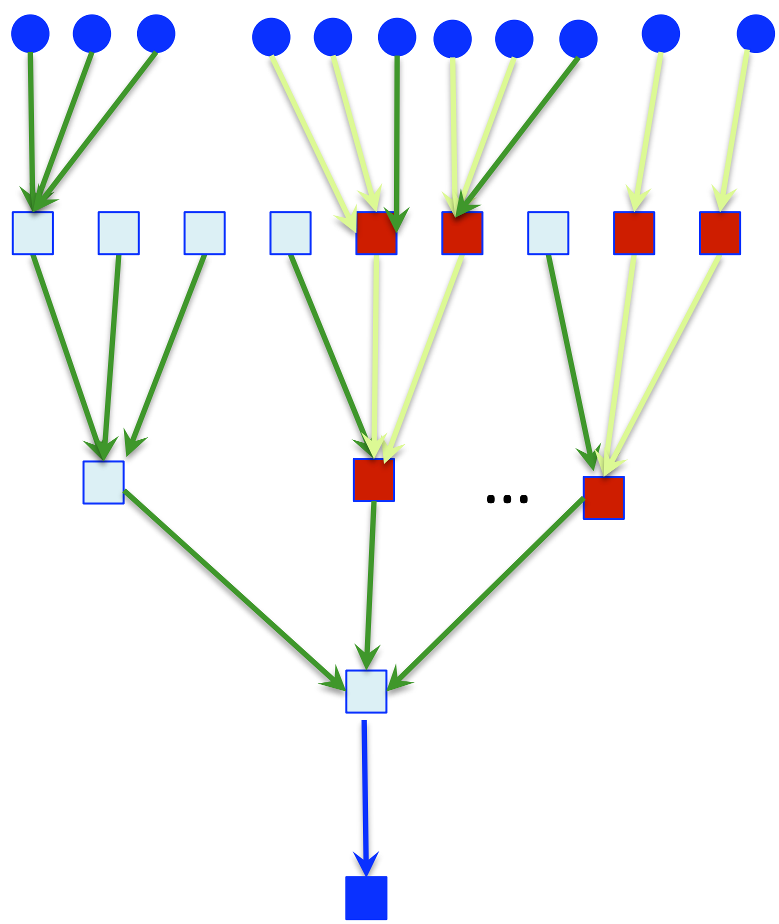
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



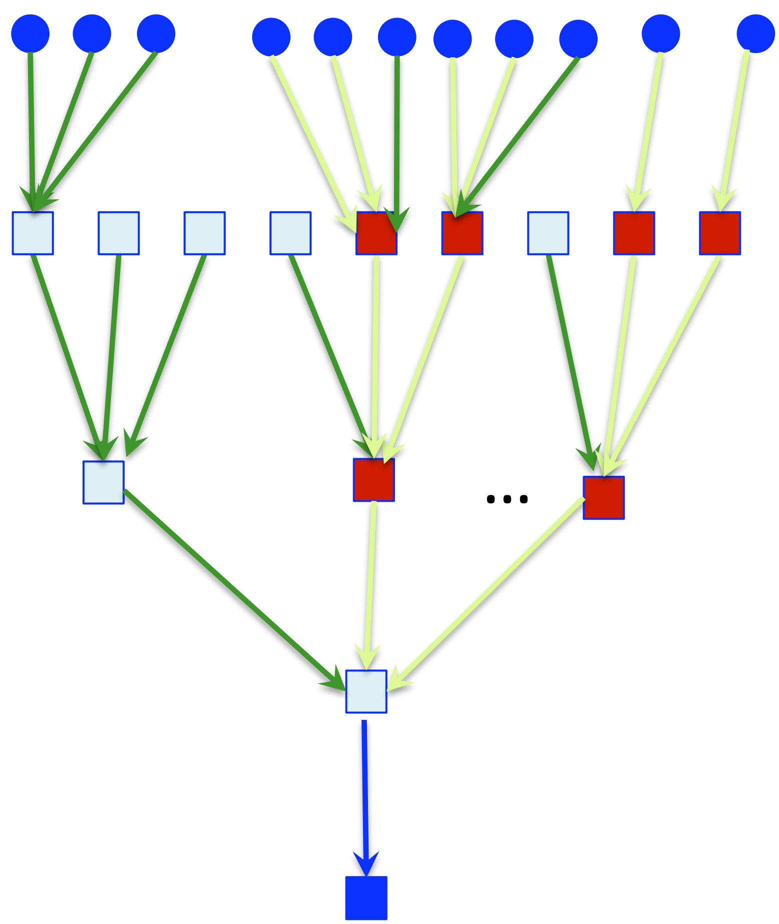
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



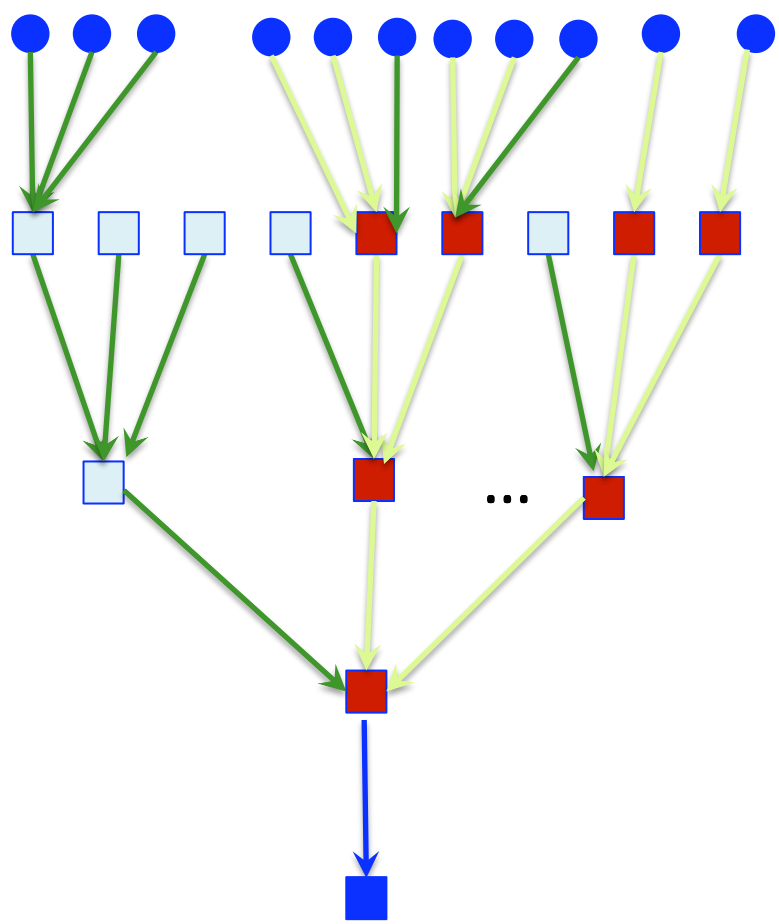
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



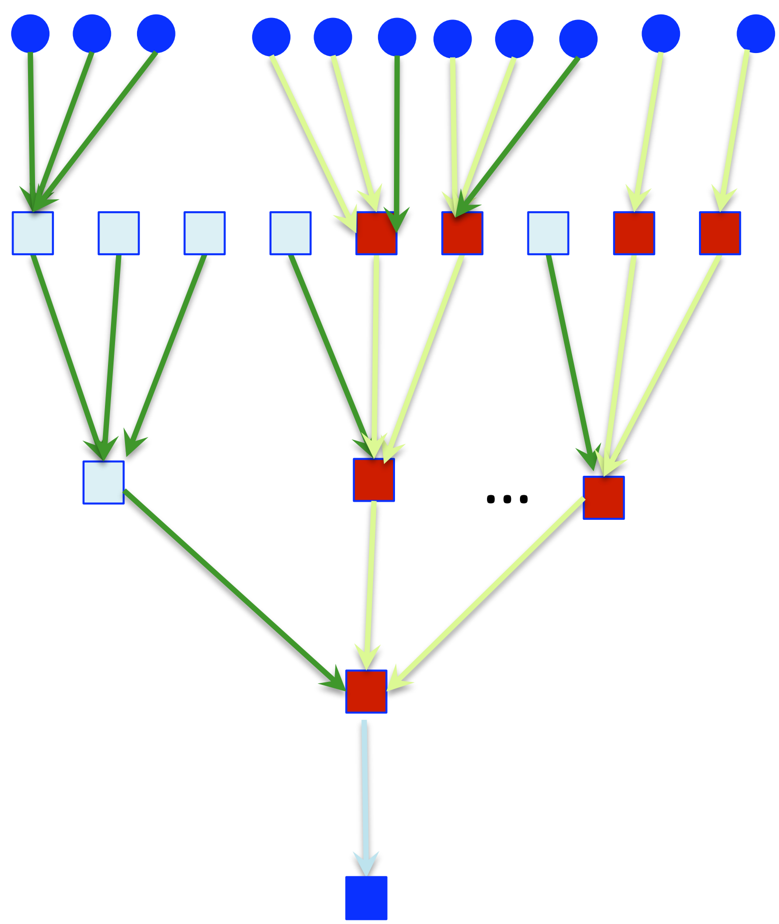
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



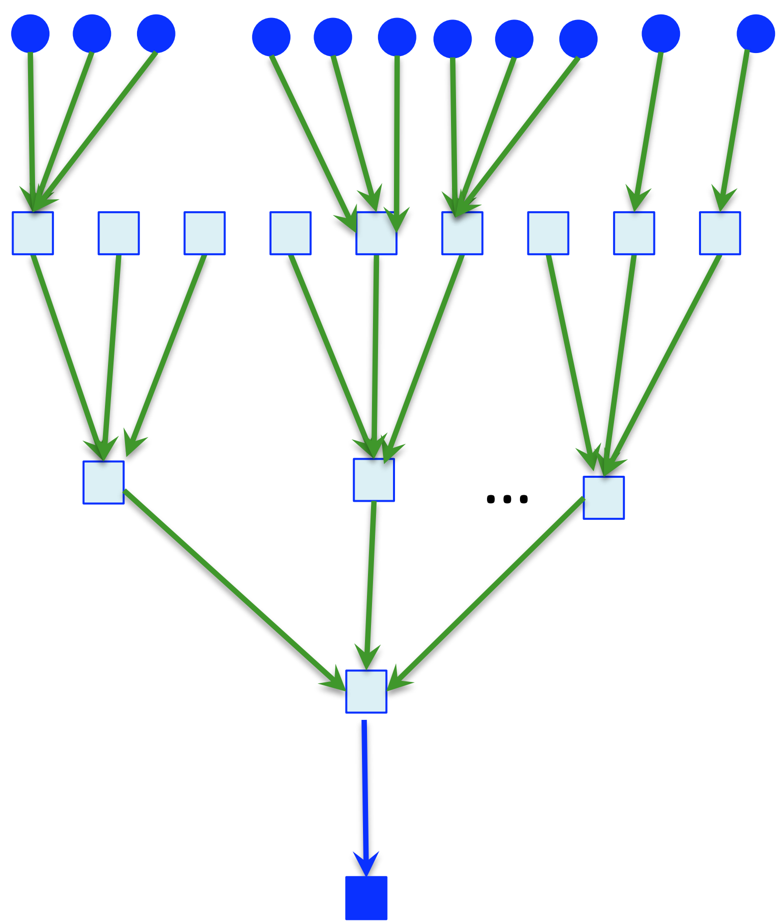
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition



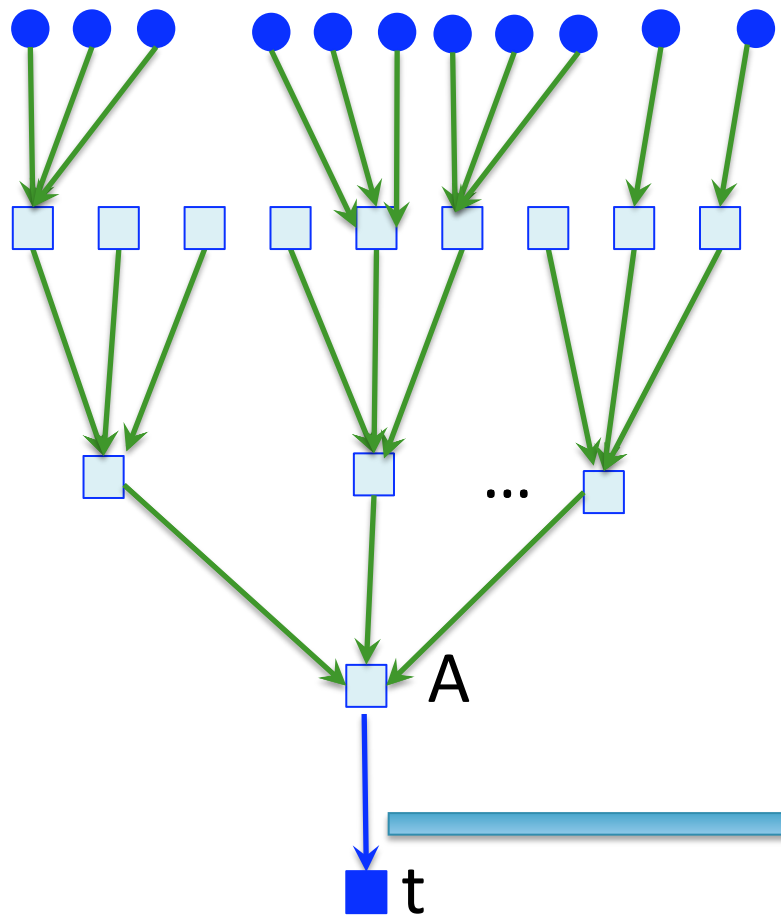
- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.

Intuition

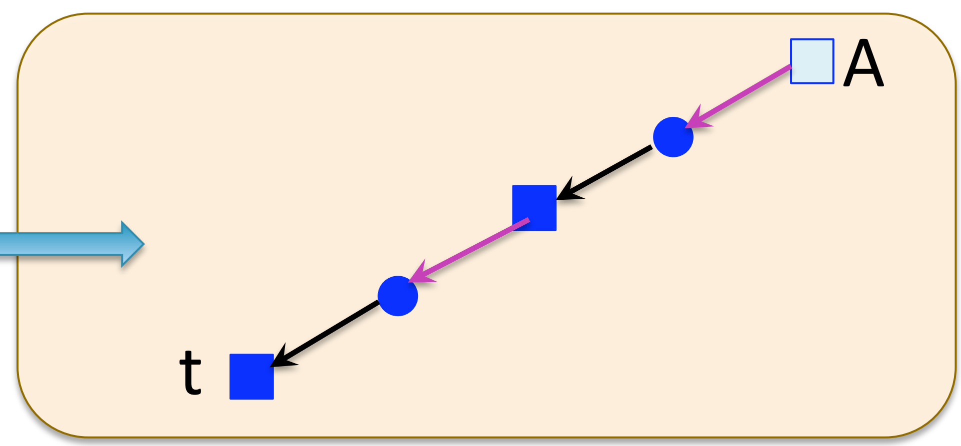


- A light agent is bad if more than half its incoming paths were deleted.
- Iteratively remove all bad light agents with their sub-trees and adjacent paths.
- A tree survives iff the blue path entering its terminal is not deleted.
- Only a small fraction of trees do not survive.

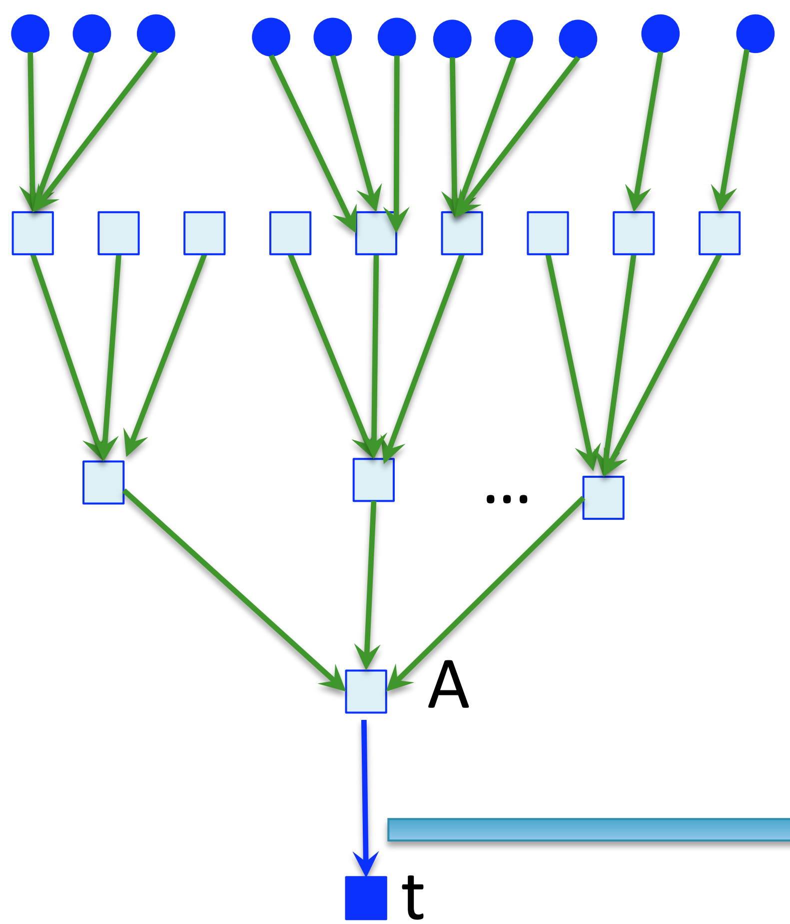
Trees that Survive



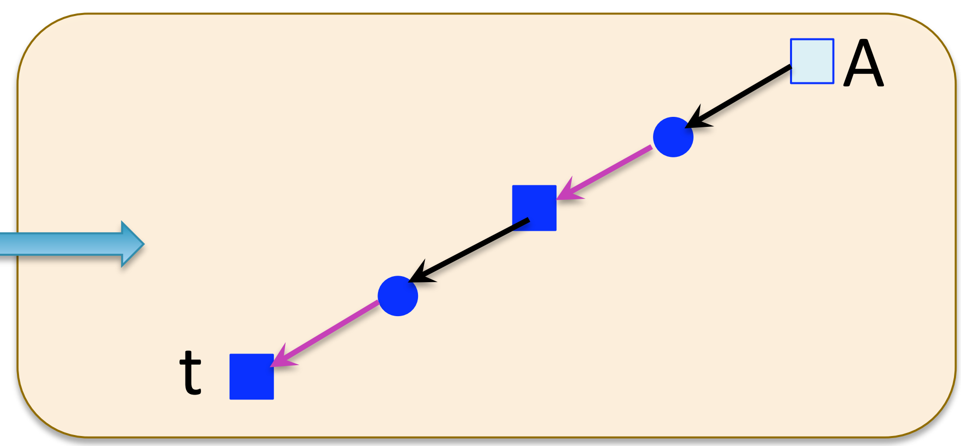
- There is a flow satisfying A by light items.
- Commit to satisfying A by light items.
- Remove A from the graph.
- Add A to set L_1
- Re-assign private items along the blue path.



Trees that Survive

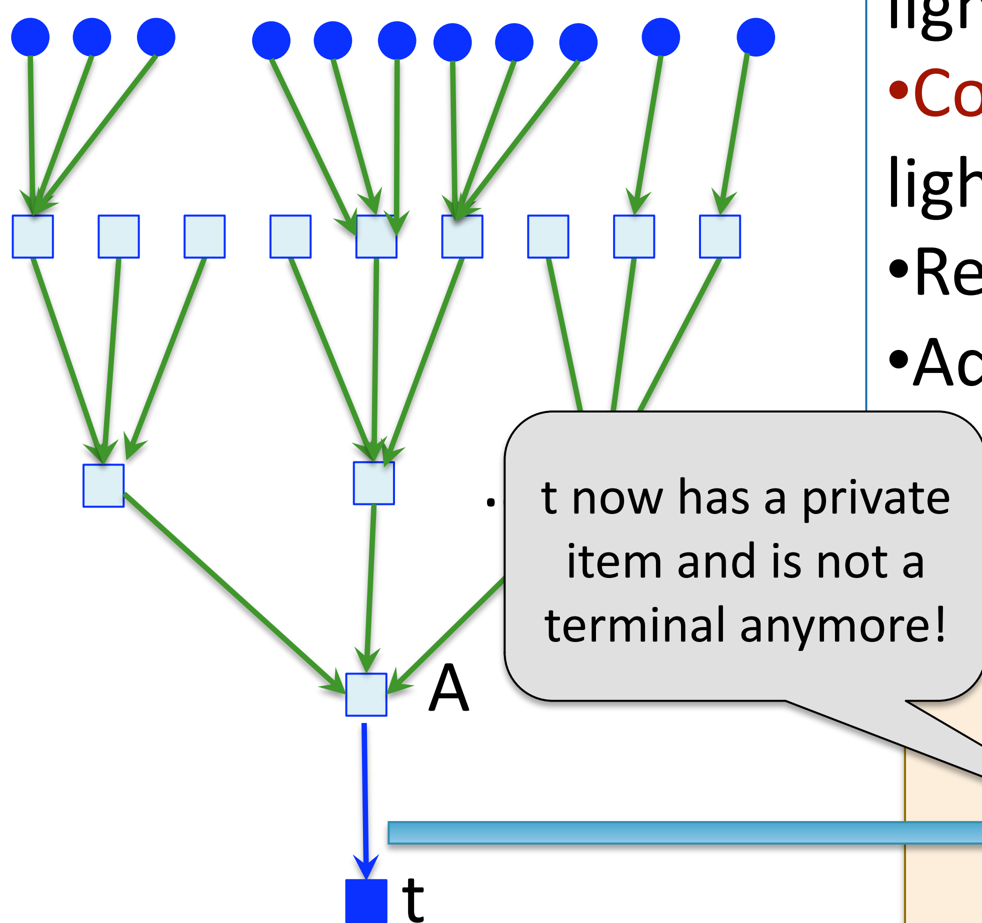


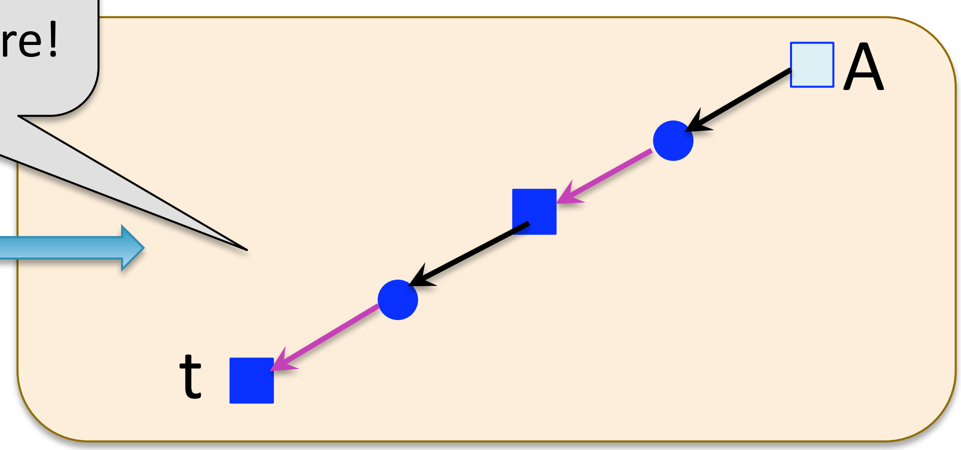
- There is a flow satisfying A by light items.
- Commit to satisfying A by light items.
- Remove A from the graph.
- Add A to set L_1
- Re-assign private items along the blue path.



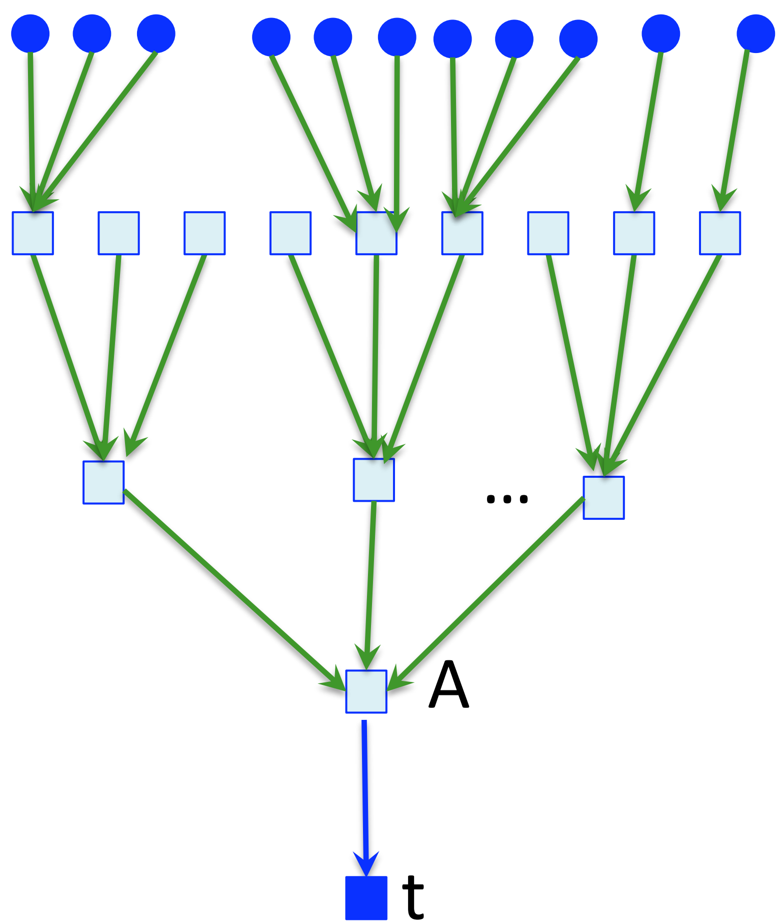
Trees that Survive

- There is a flow satisfying A by light items.
- Commit to satisfying A by light items.
- Remove A from the graph.
- Add A to set L_1
- -assign private items along blue path.





Trees that don't Survive

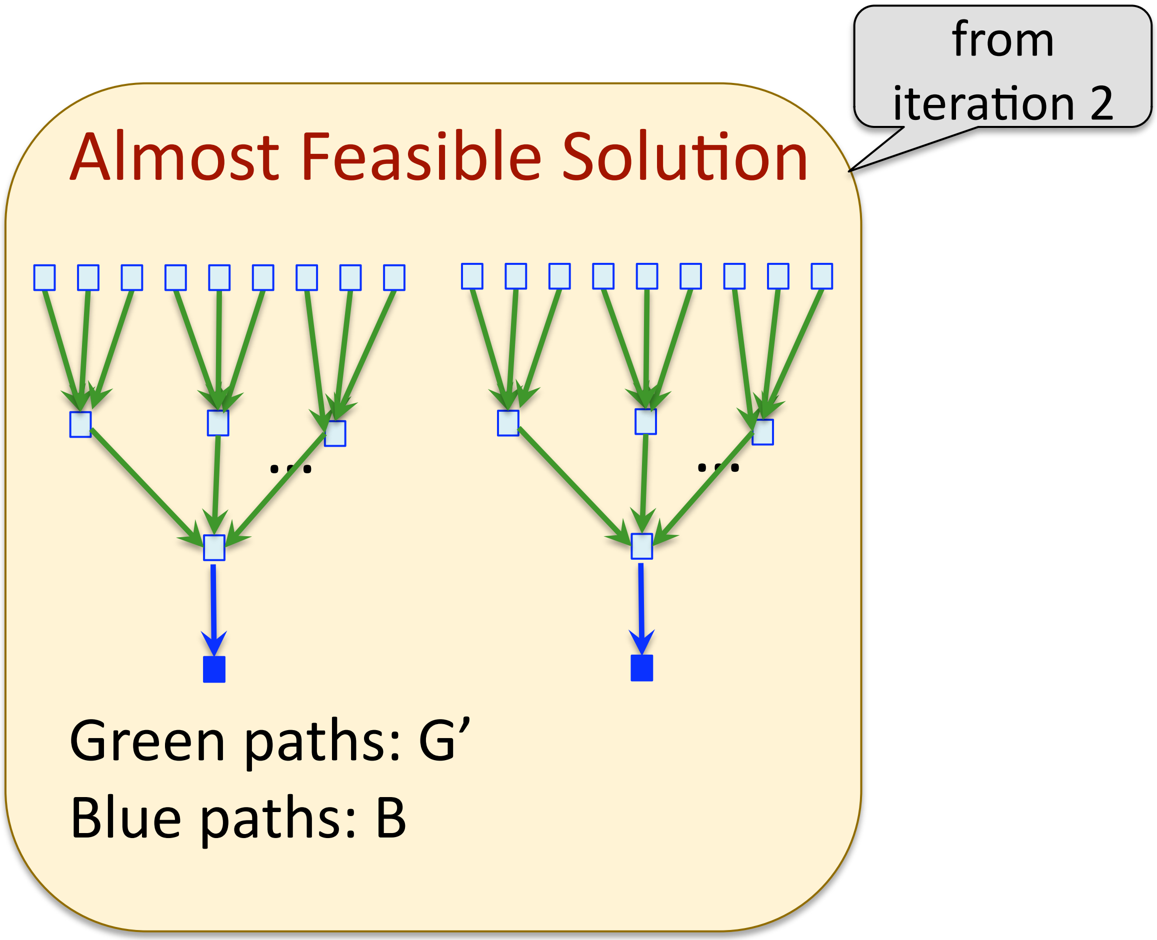


- t remains a terminal for the next iteration.
- Only small fraction of trees don't survive
- So number of terminals is much smaller now.

Iteration 2

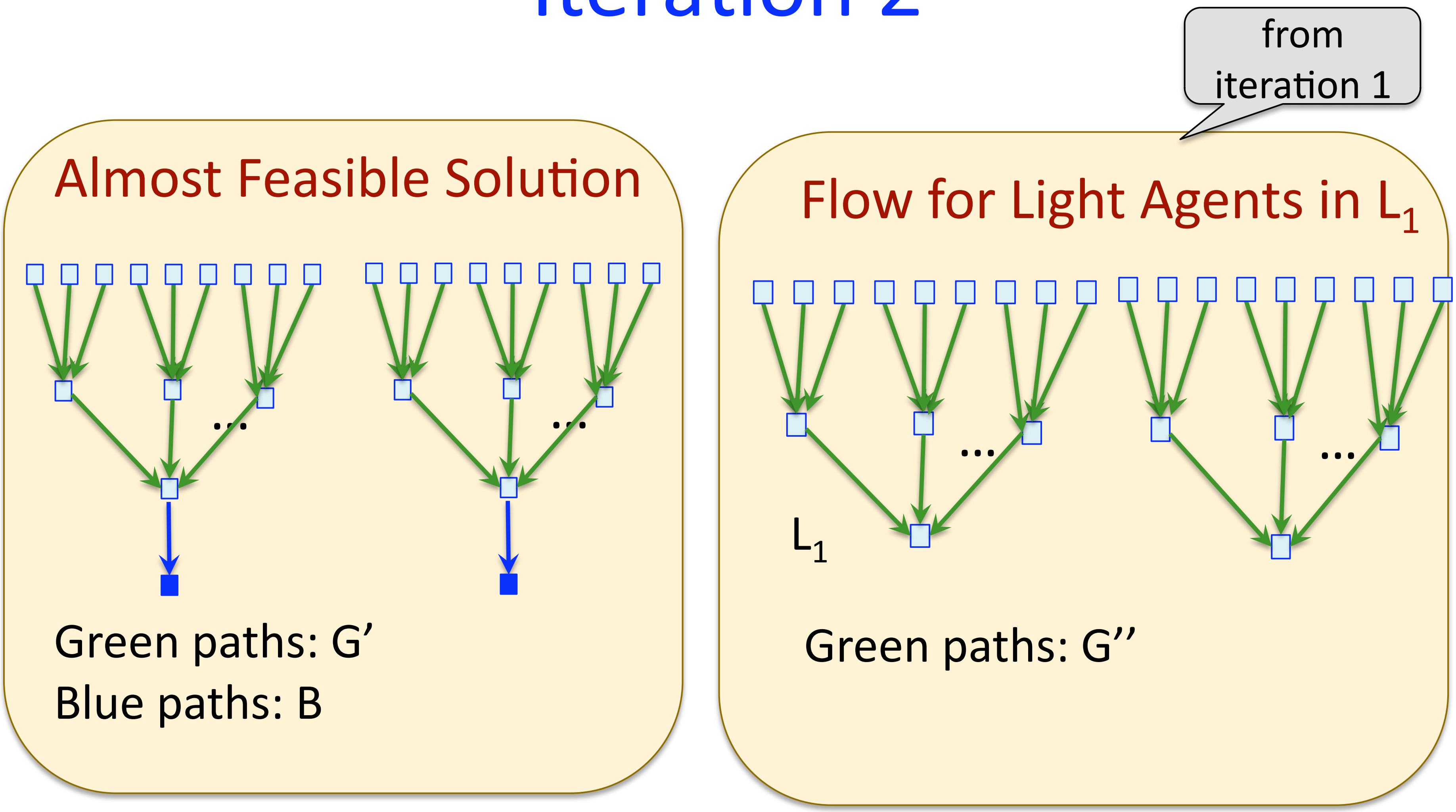
- Obtain almost-feasible polylog-approximate solution for remaining instance.

Iteration 2



A vertex may appear on one path in each of G' and B.

Iteration 2



A vertex may appear on one path in each of G', G'' and B.

Iteration 2

- Obtain almost-feasible solution for remaining instance.
- Combine G' and G'' using the scaling trick to get a set G of green paths.
- Re-route paths in B so they intersect a small number of paths in G.
- Remove from G all paths intersecting paths in B.
- Take care of bad agents.
- Produce input for next iteration as before.

Iteration 2

- Obtain almost-feasible solution for remaining instance.
- Combine G' and G'' using the scaling trick to get a set G of green paths.
- Re-route paths in B so they intersect a small number of paths in G
- Remove from G all p
- Take care of bad ag
- Produce input for next iteration as before.

•Number of terminals goes down by almost n^{ϵ} factor in each iteration.
•After $O(1/\epsilon)$ iterations we will be done.

Summary

- We have shown $\tilde{O}\left(n^{\epsilon}\right)$ -approximation for Max Min Allocation, in $n^{O(1/\epsilon)}$ running time
  - poly-logarithmic approximation in quasi-polynomial time
- Best current hardness of approximation is 2.
- Santa Claus problem: best current approximation is O(log log m/log log log m), same hardness of approximation

Thank you!