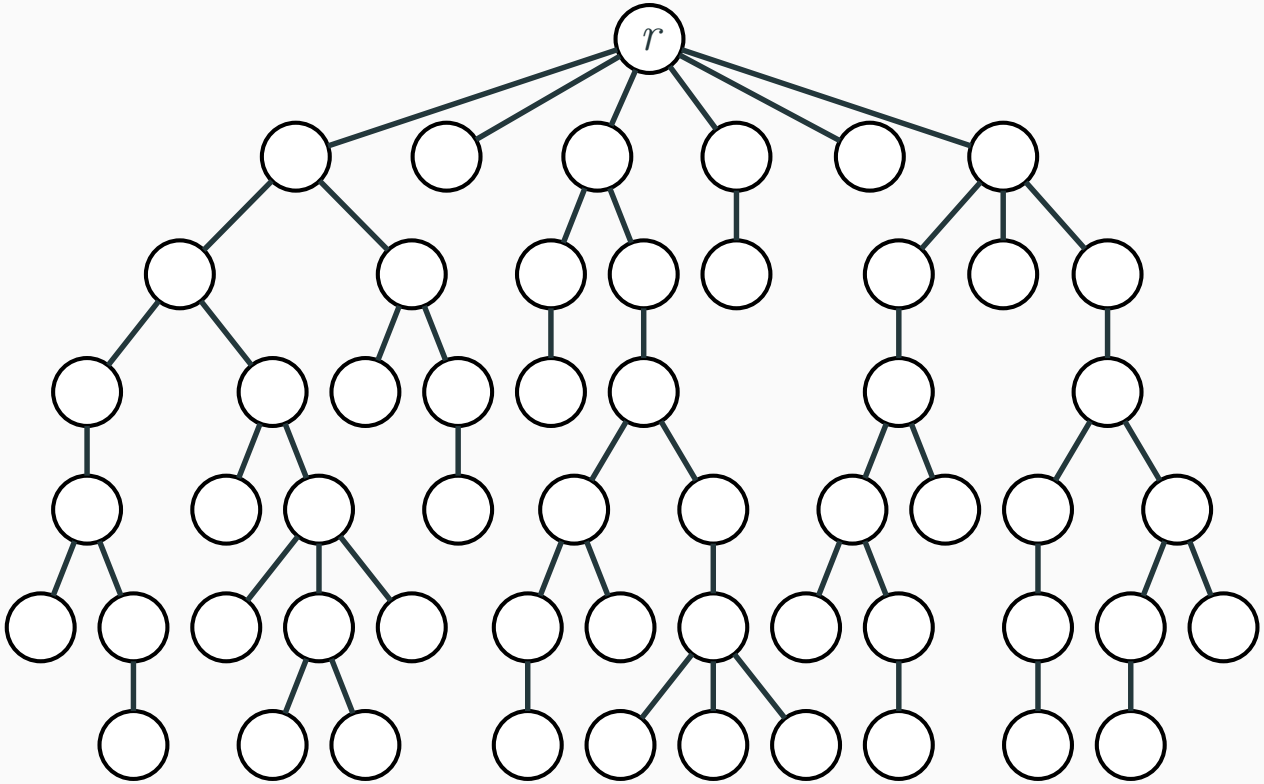
Firefighting on Trees Beyond Integrality Gaps

David Adjiashvili, Andrea Baggio, **Rico Zenklusen**

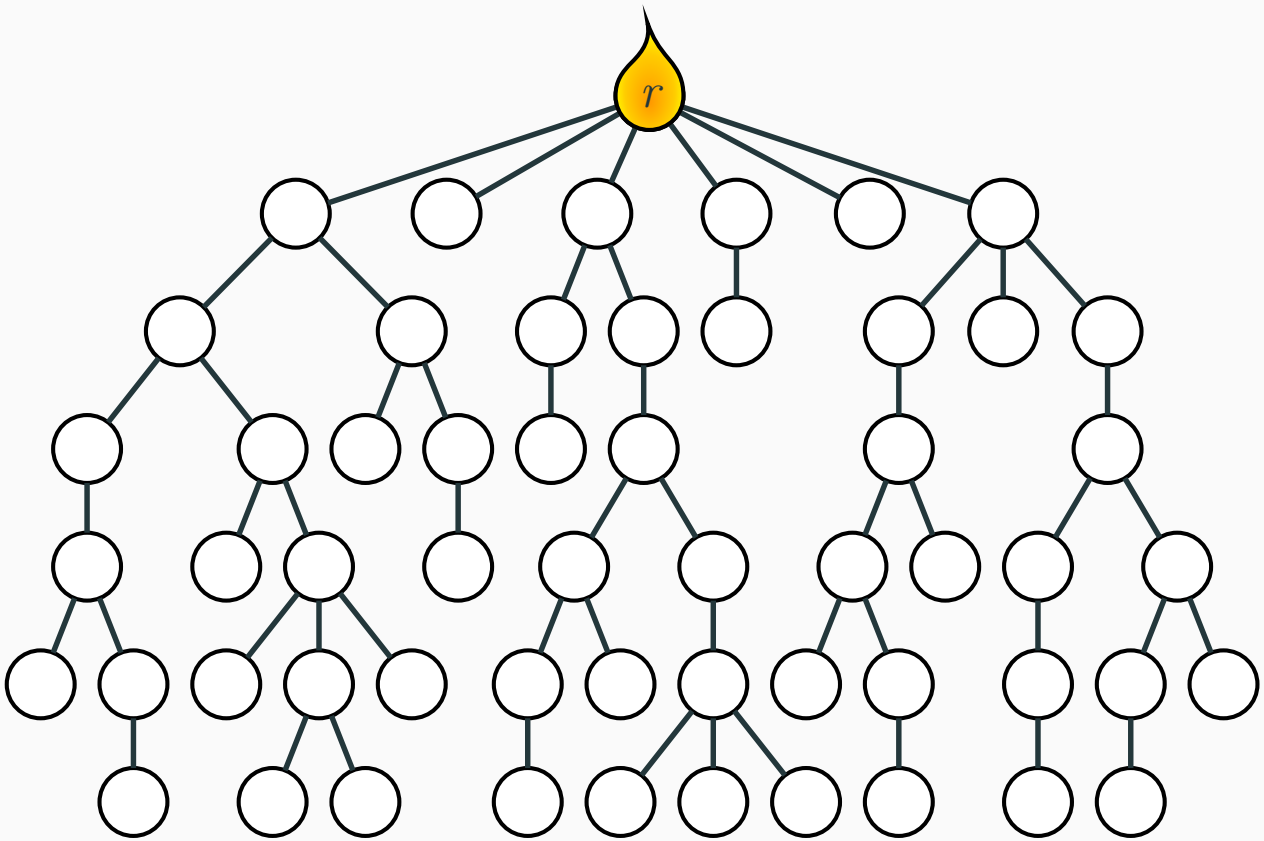
ETH Zurich

Introduction

Fire Spreading Model

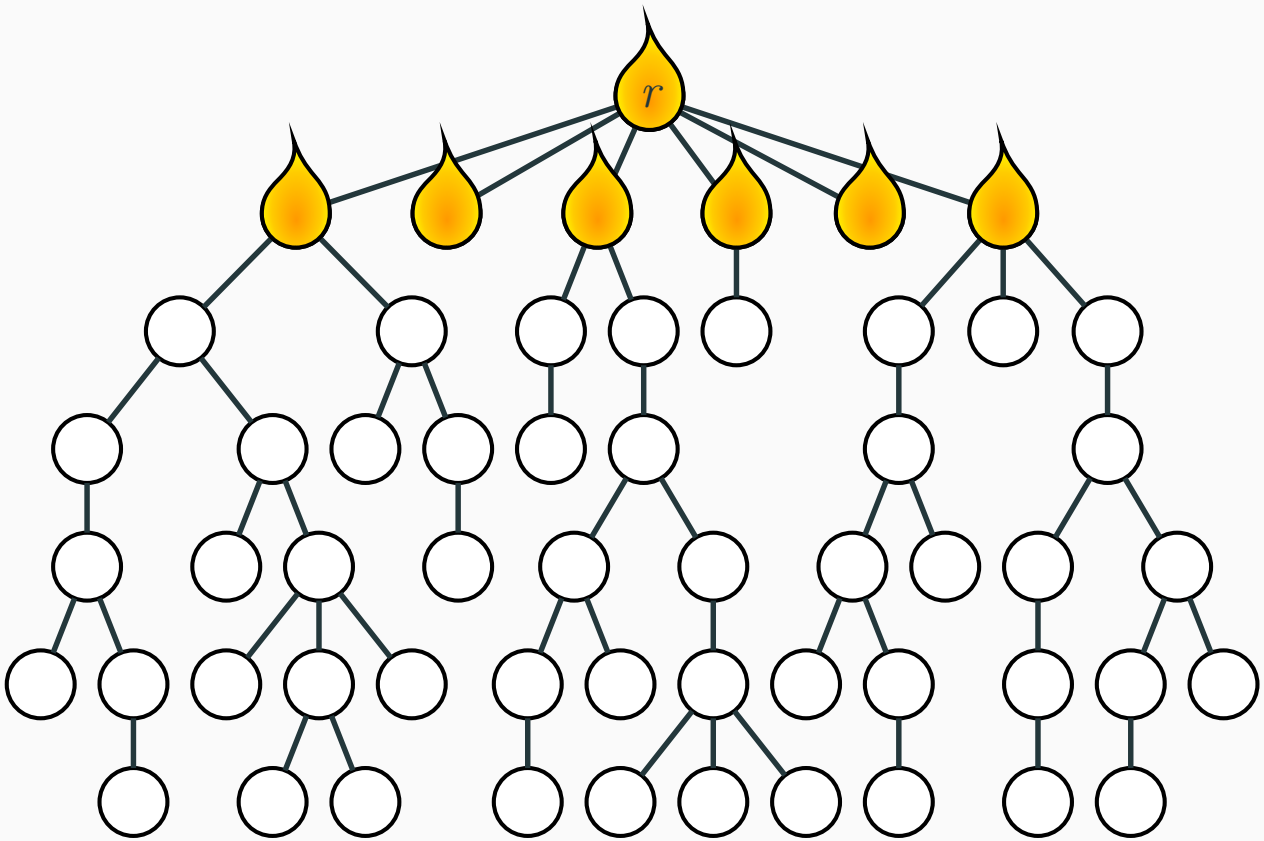


Fire Spreading Model



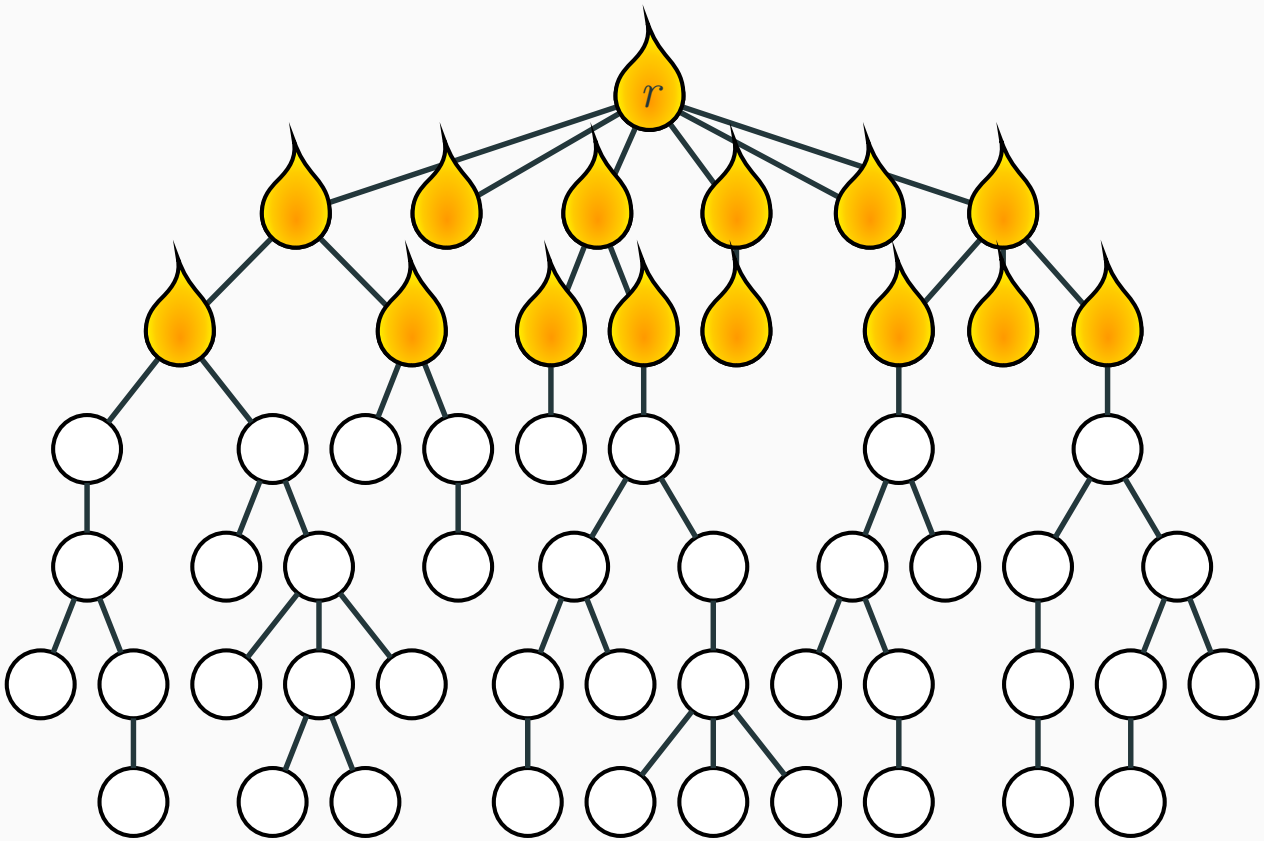
$t = 0$

Fire Spreading Model



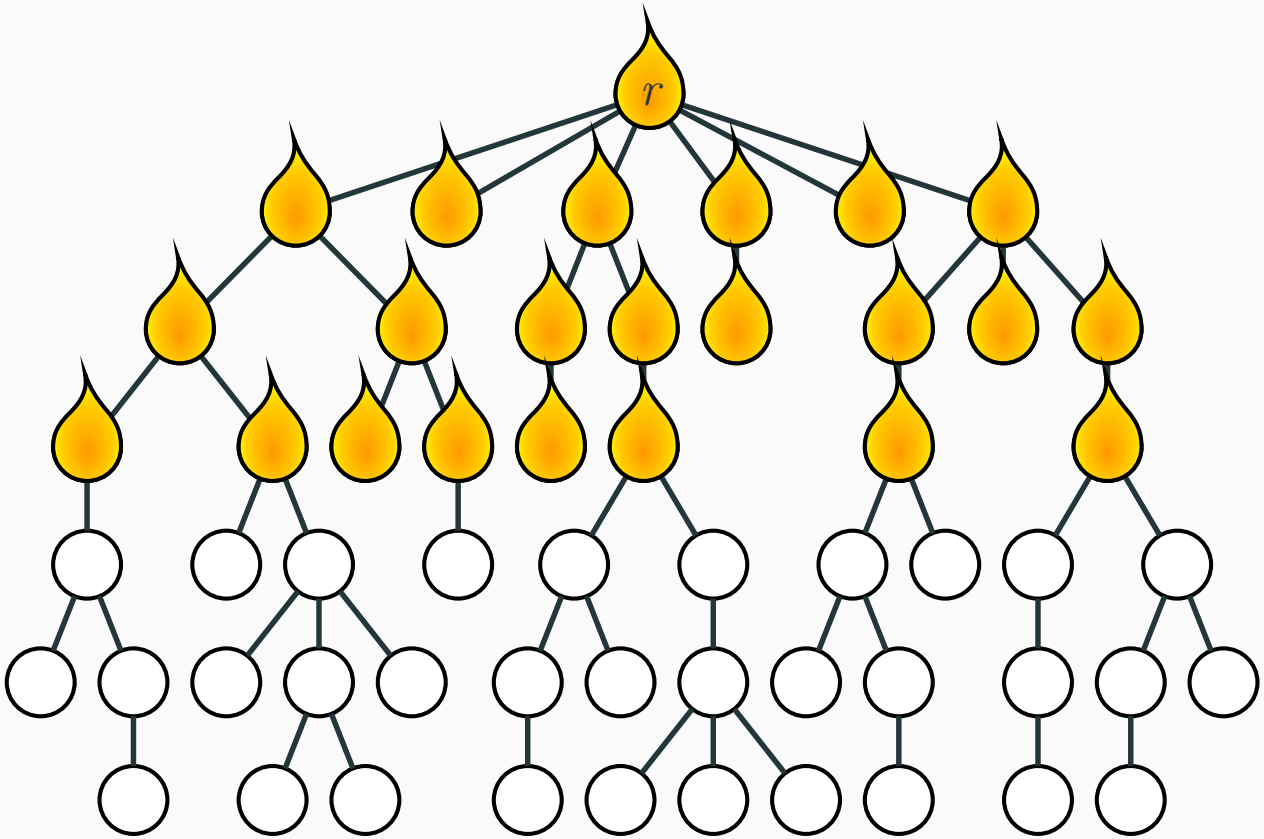
$t = 1$

Fire Spreading Model



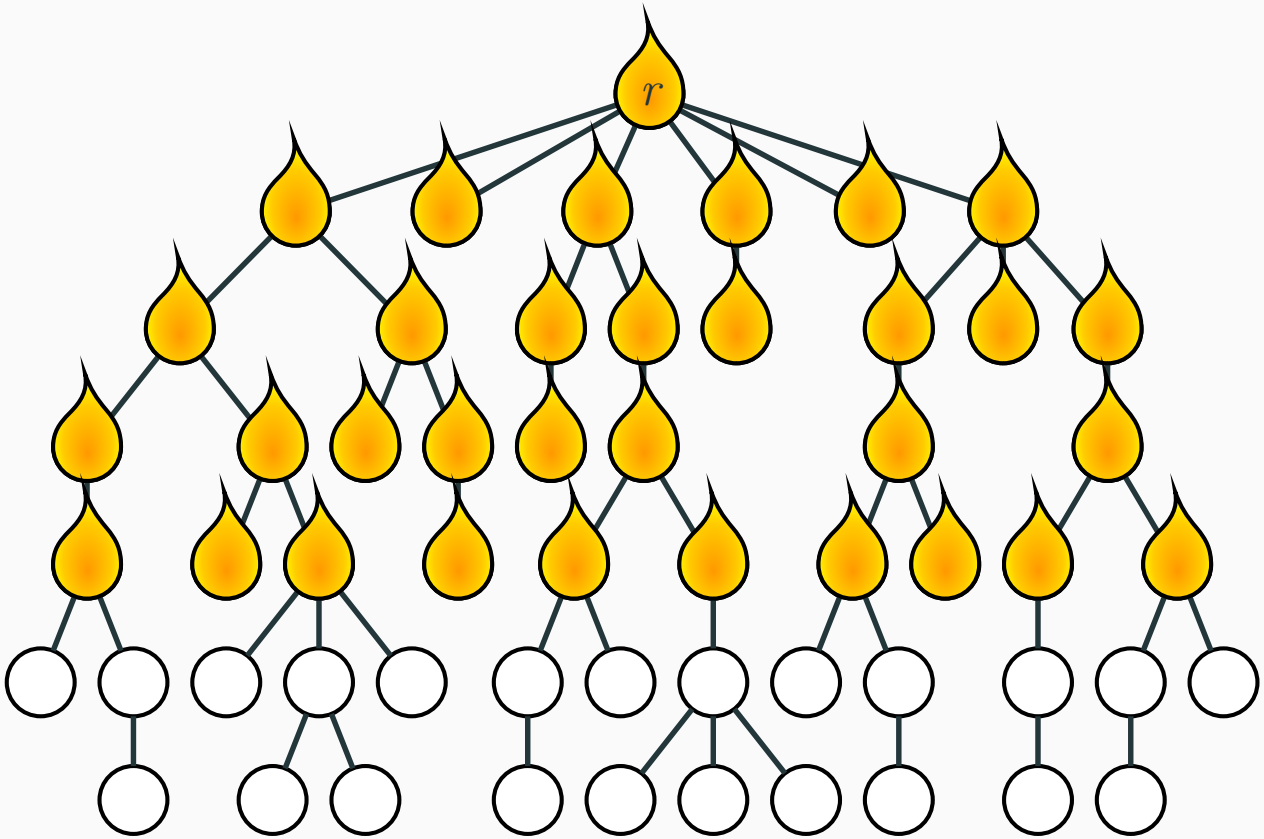
$t = 2$

Fire Spreading Model



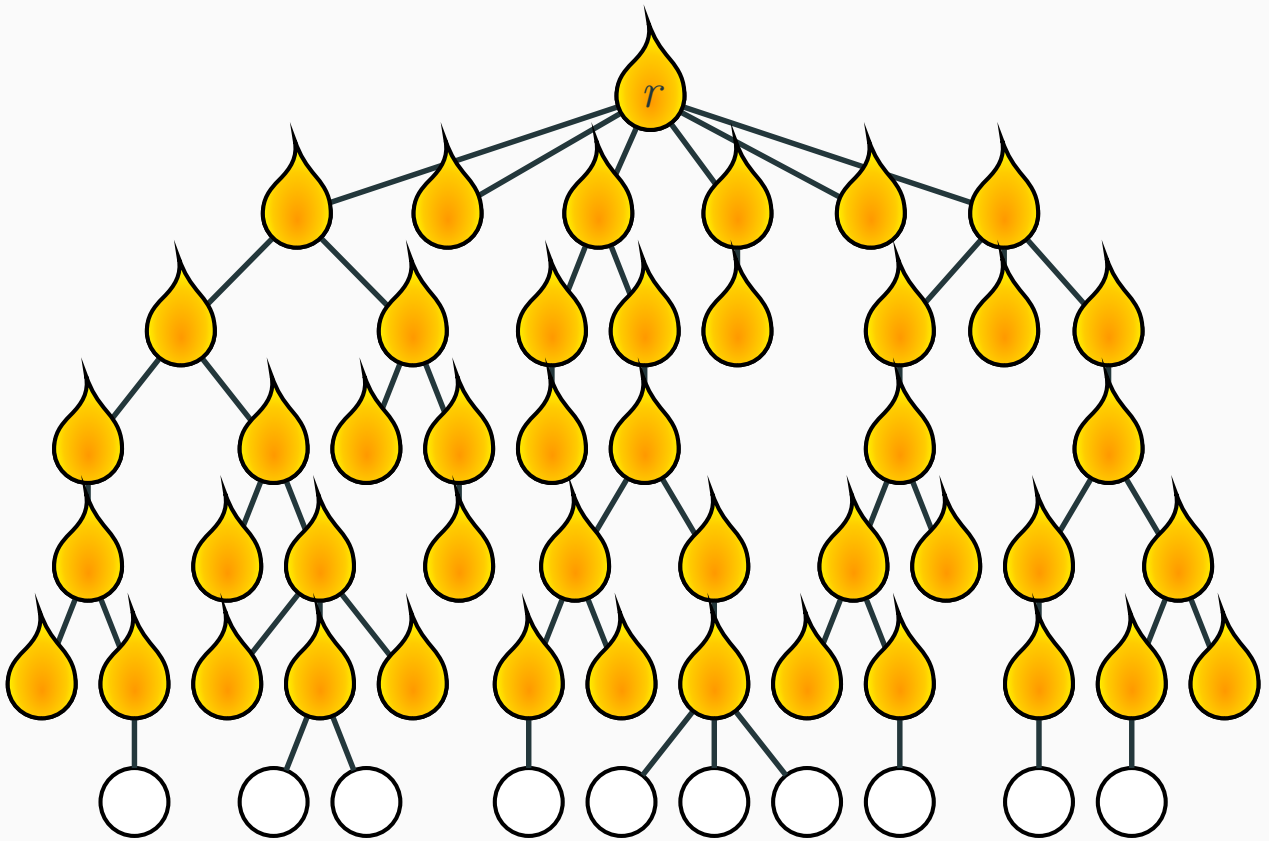
$t = 3$

Fire Spreading Model



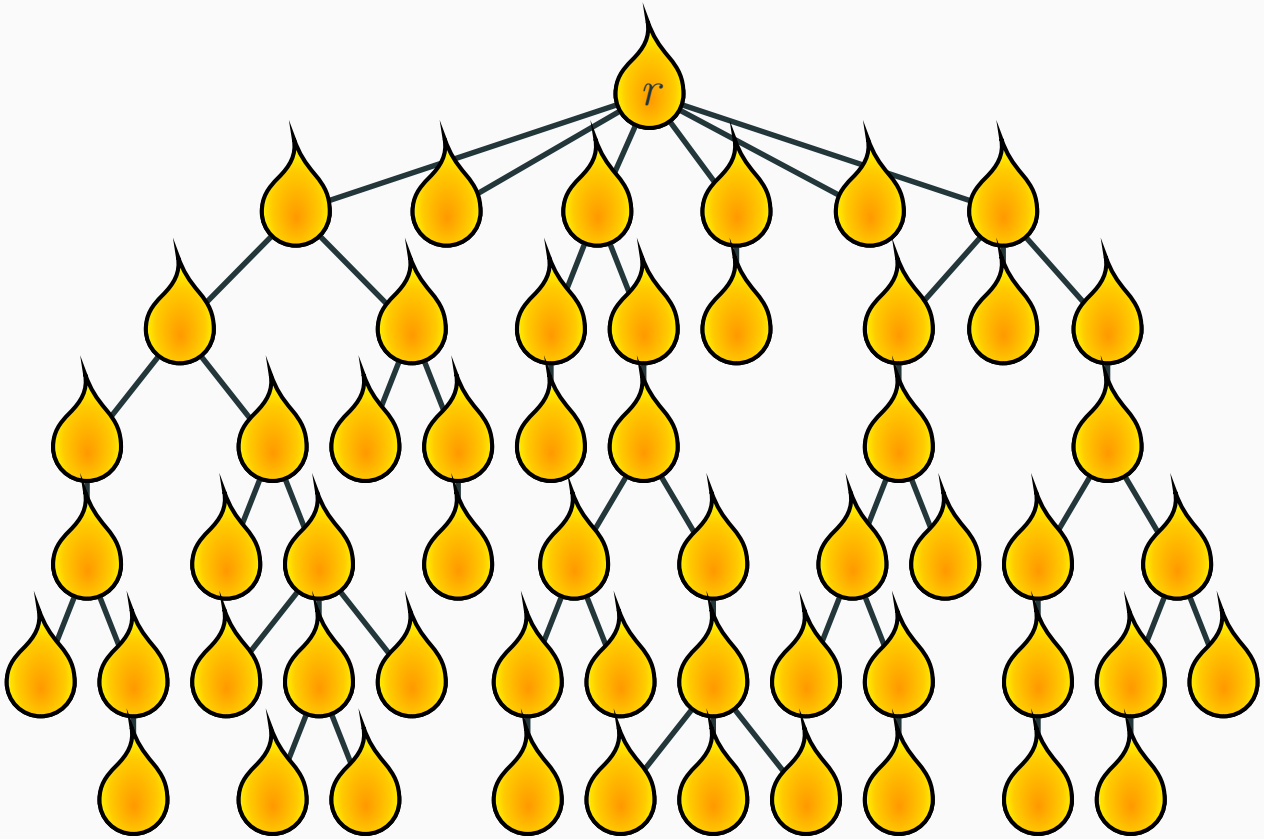
$t = 4$

Fire Spreading Model



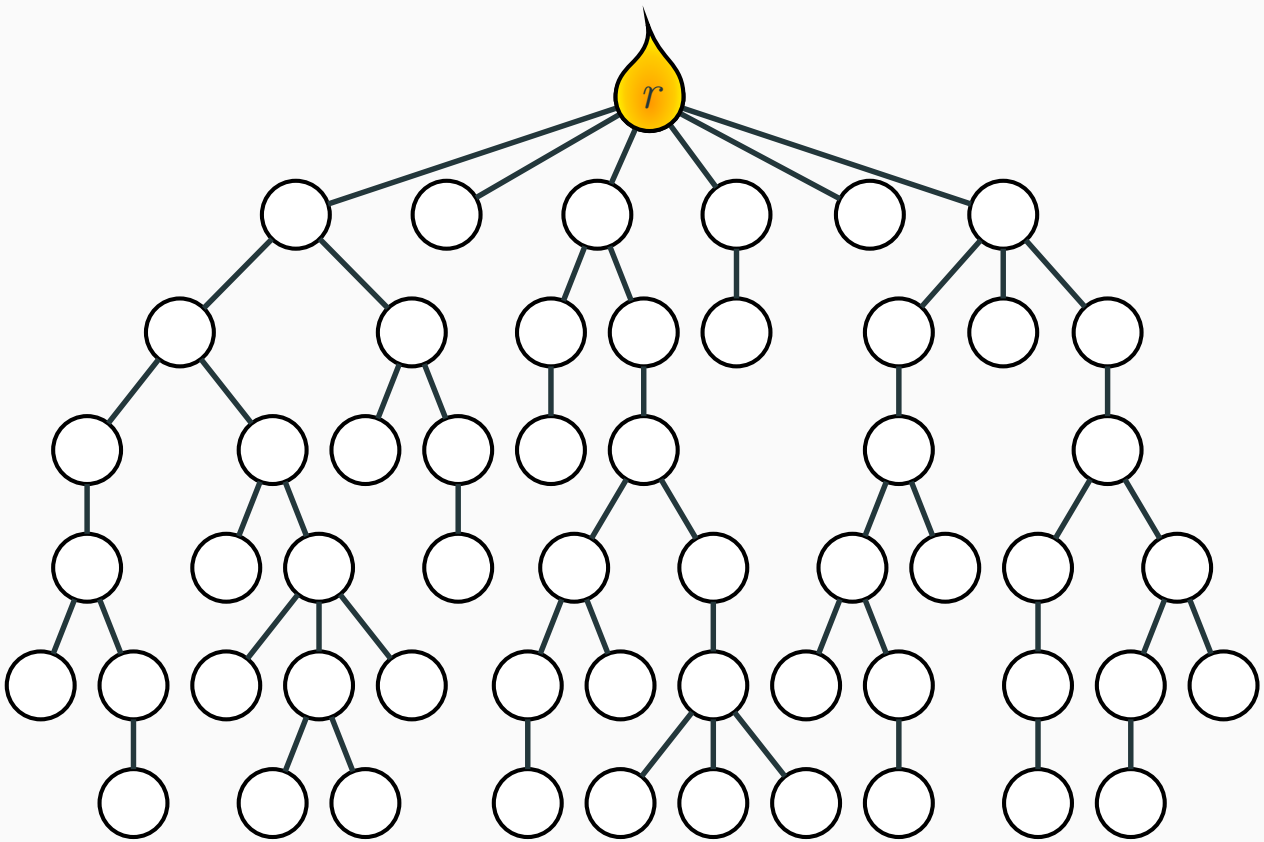
$t = 5$

Fire Spreading Model



$t = 6$

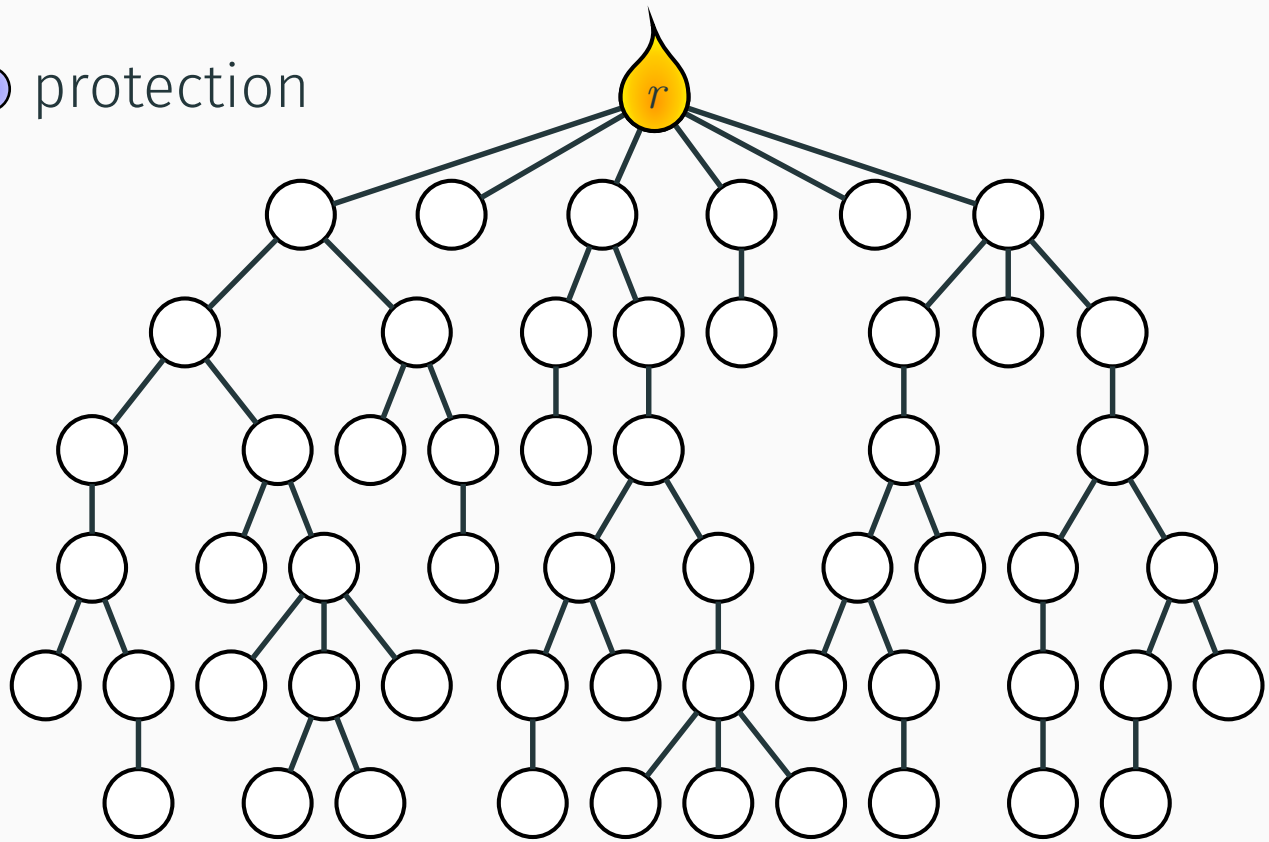
FireFighter Problem - FFP



$t = 0$

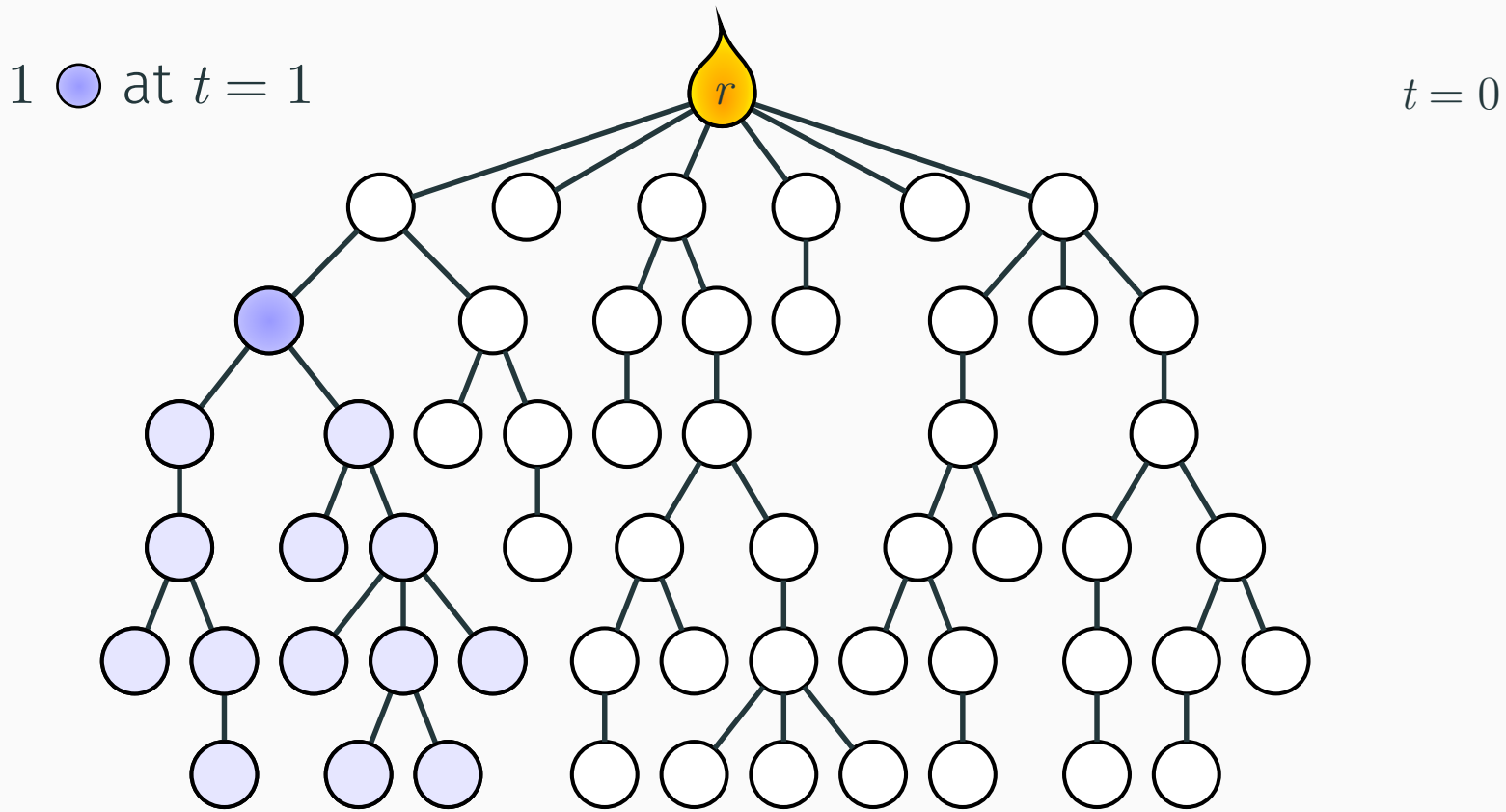
FireFighter Problem - FFP

protection

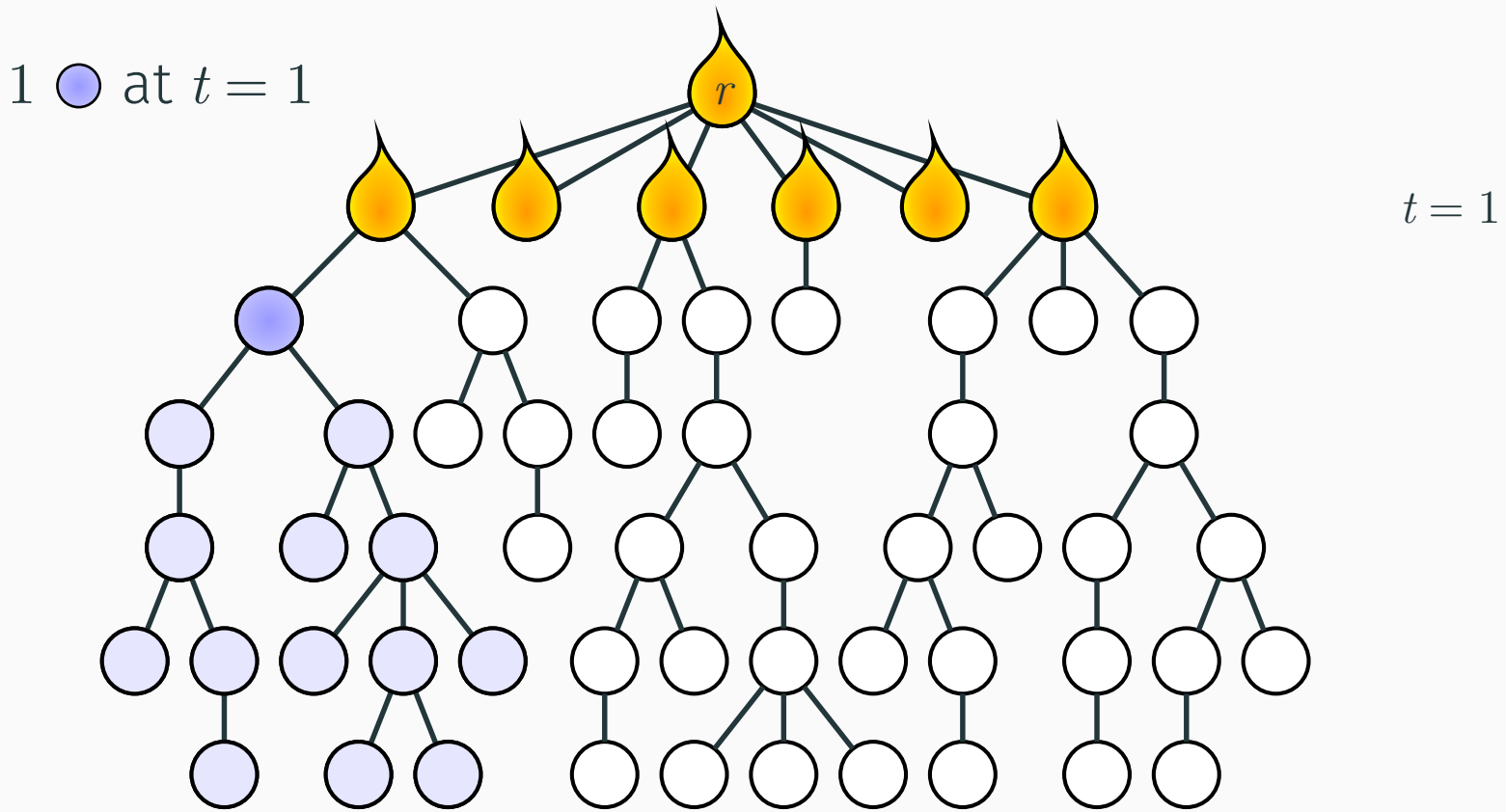


$t = 0$

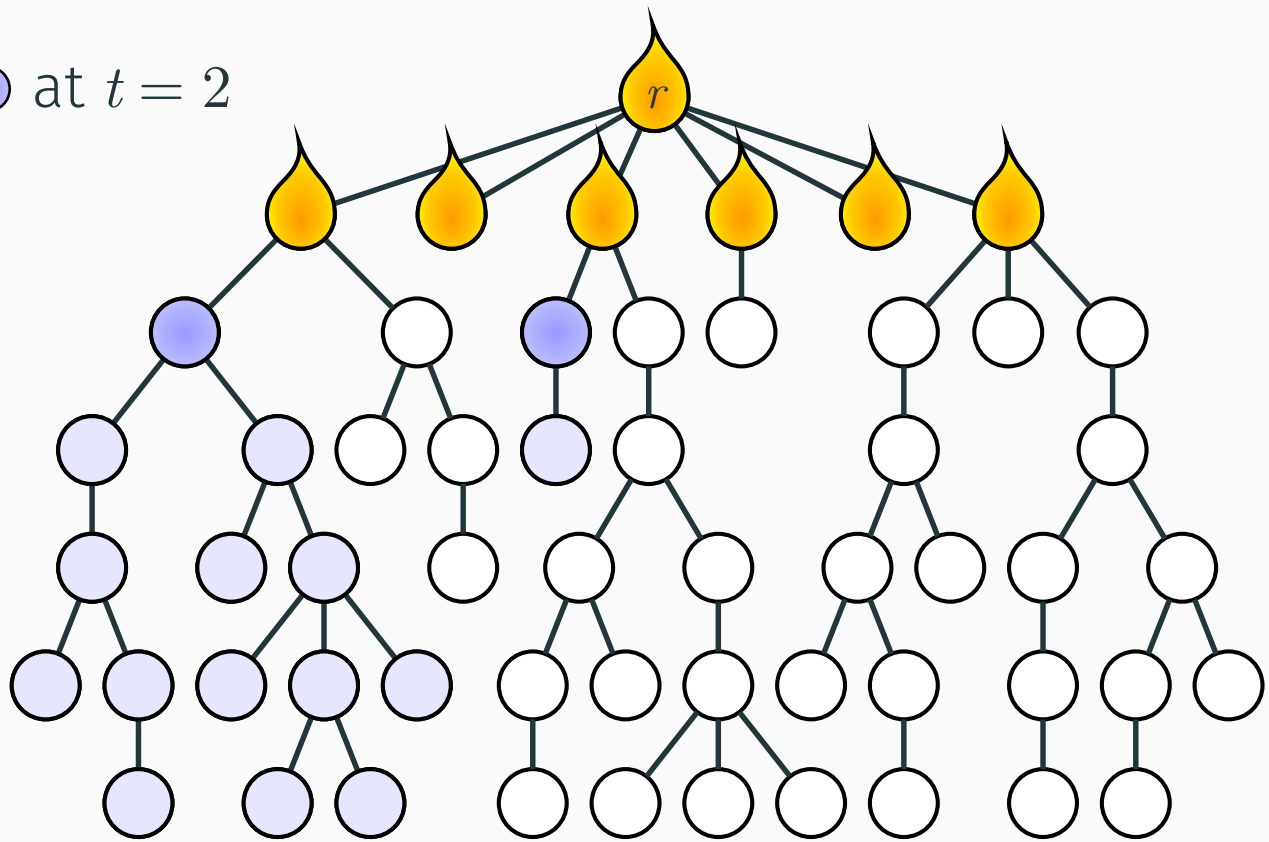
FireFighter Problem - FFP



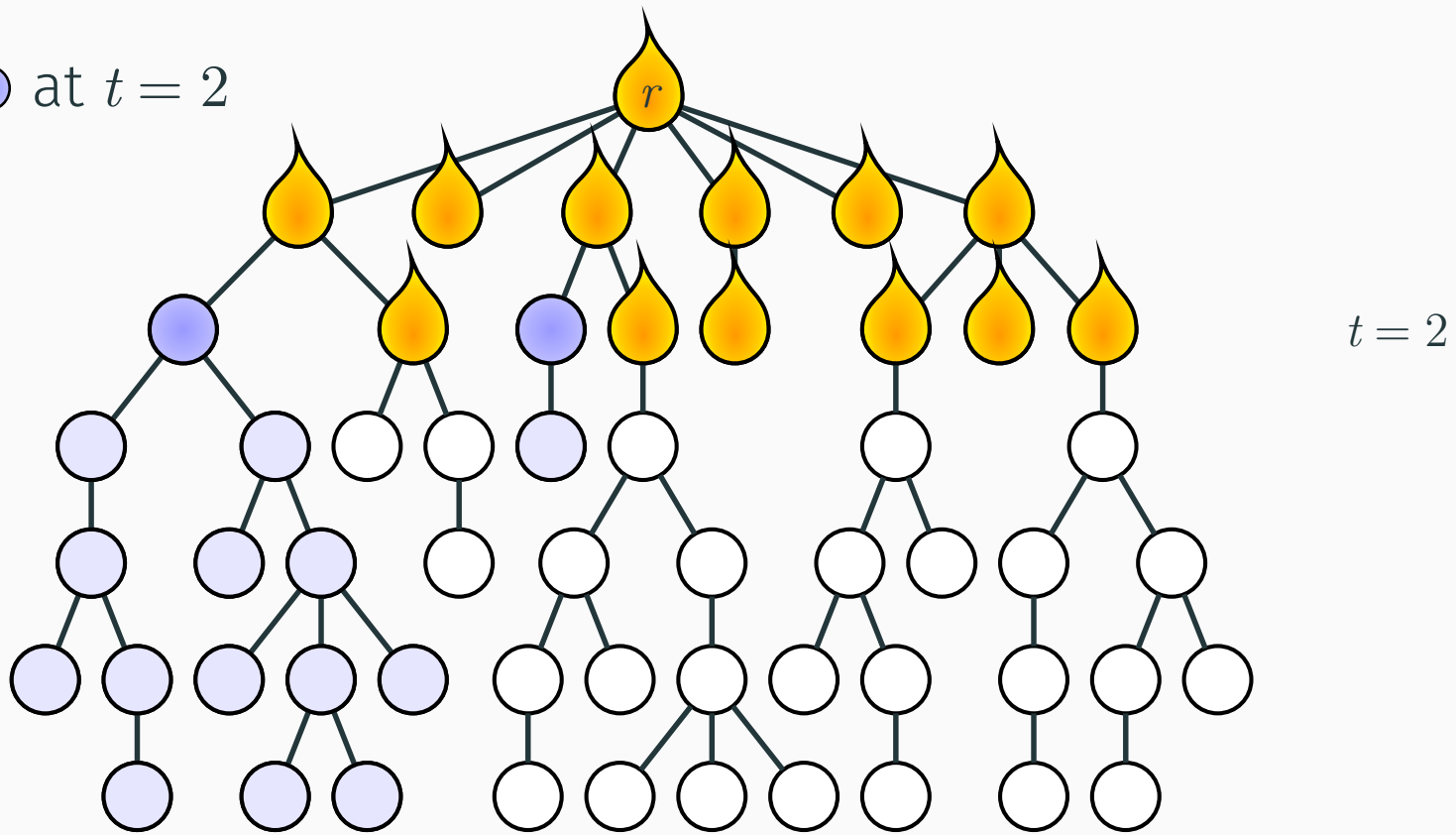
FireFighter Problem - FFP



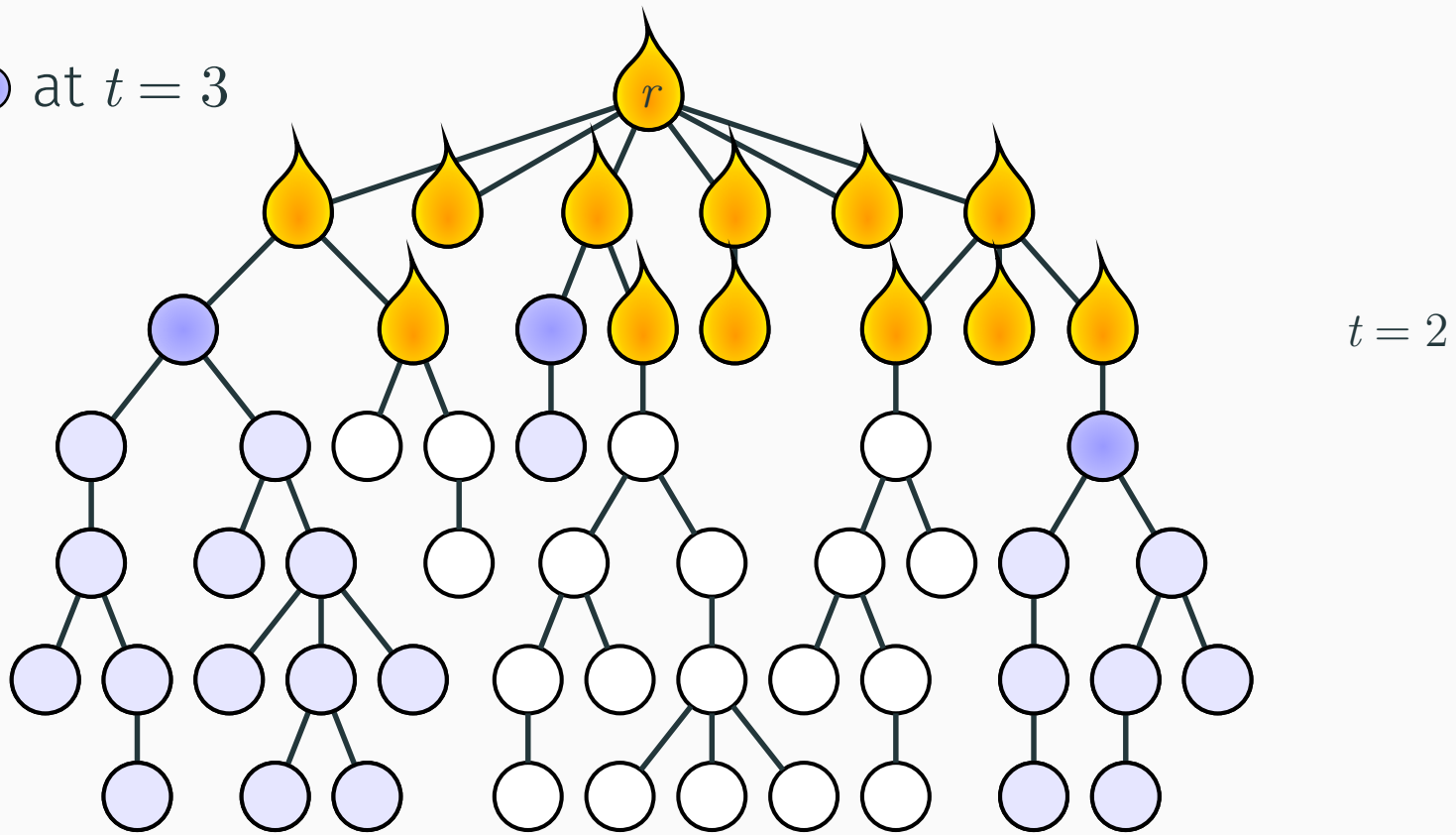
FireFighter Problem - FFP



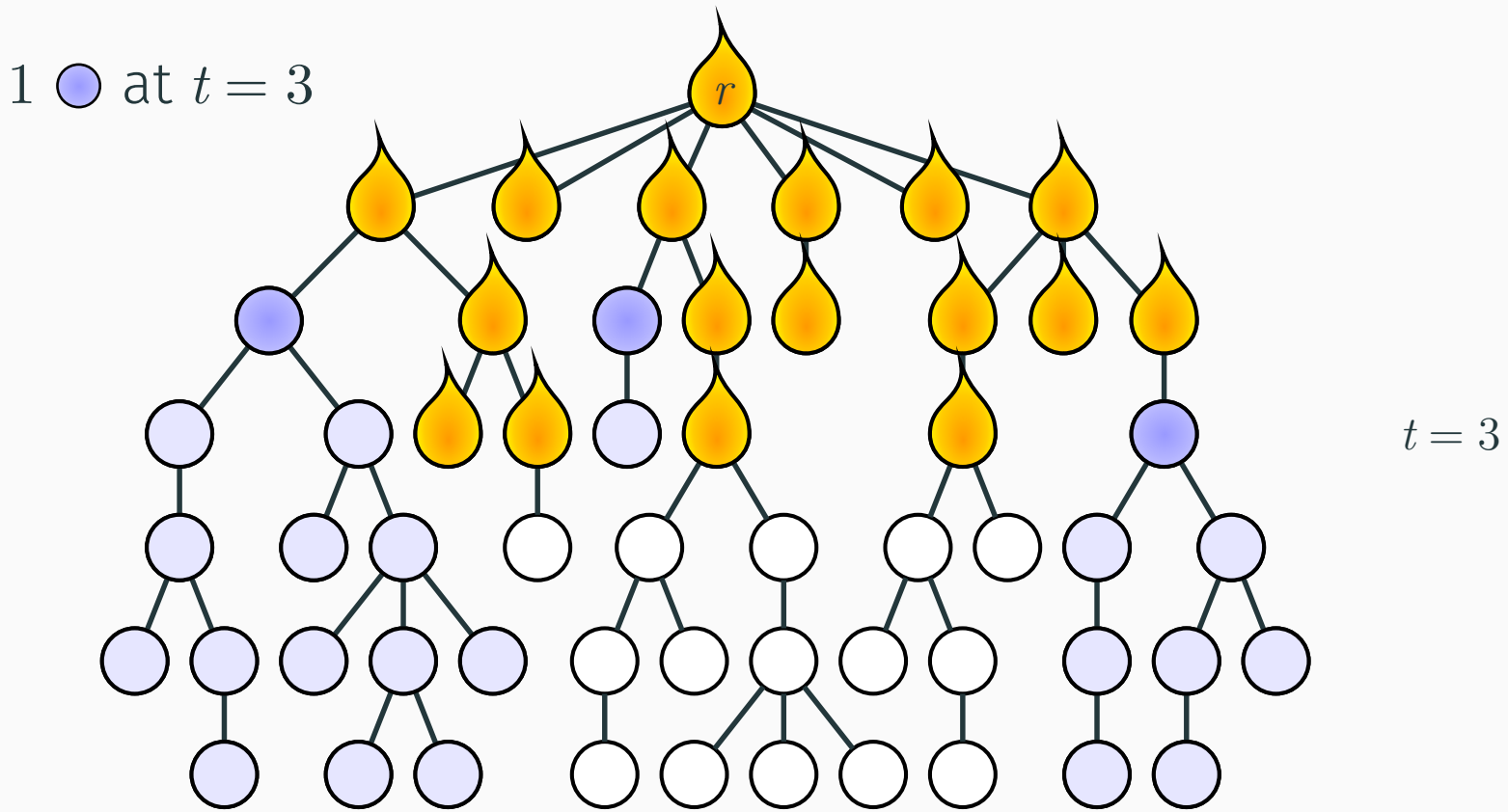
FireFighter Problem - FFP



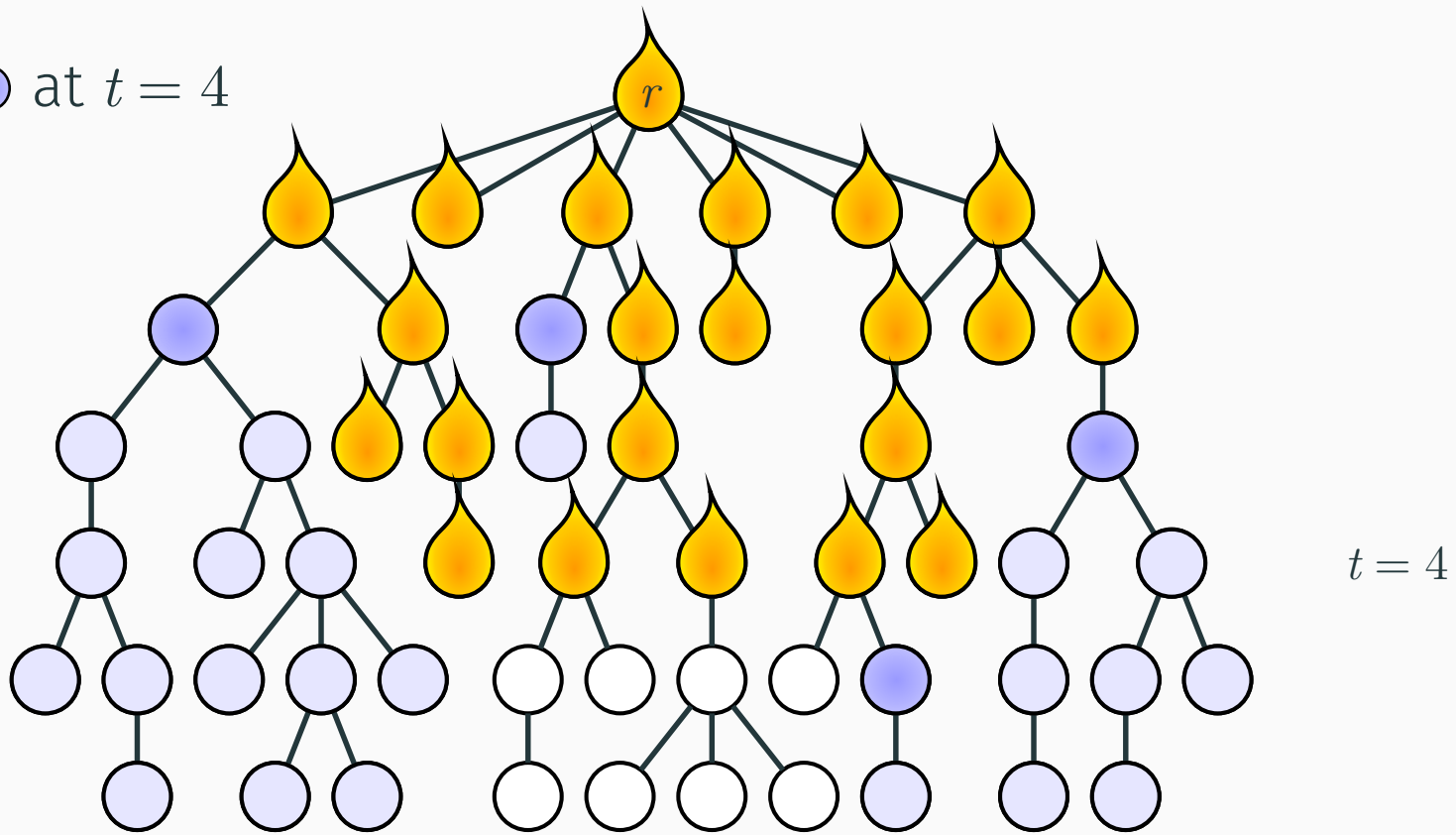
FireFighter Problem - FFP



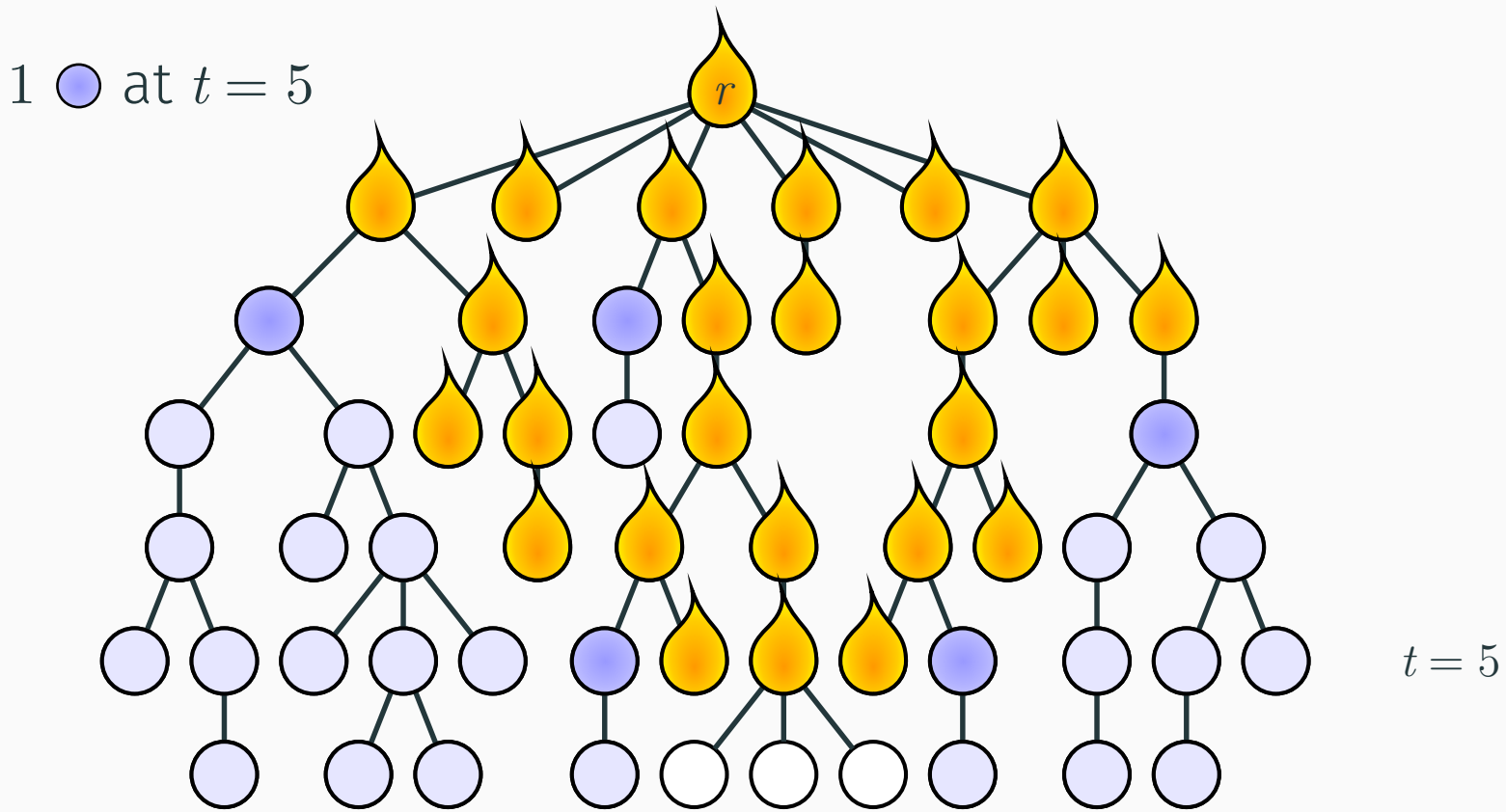
FireFighter Problem - FFP



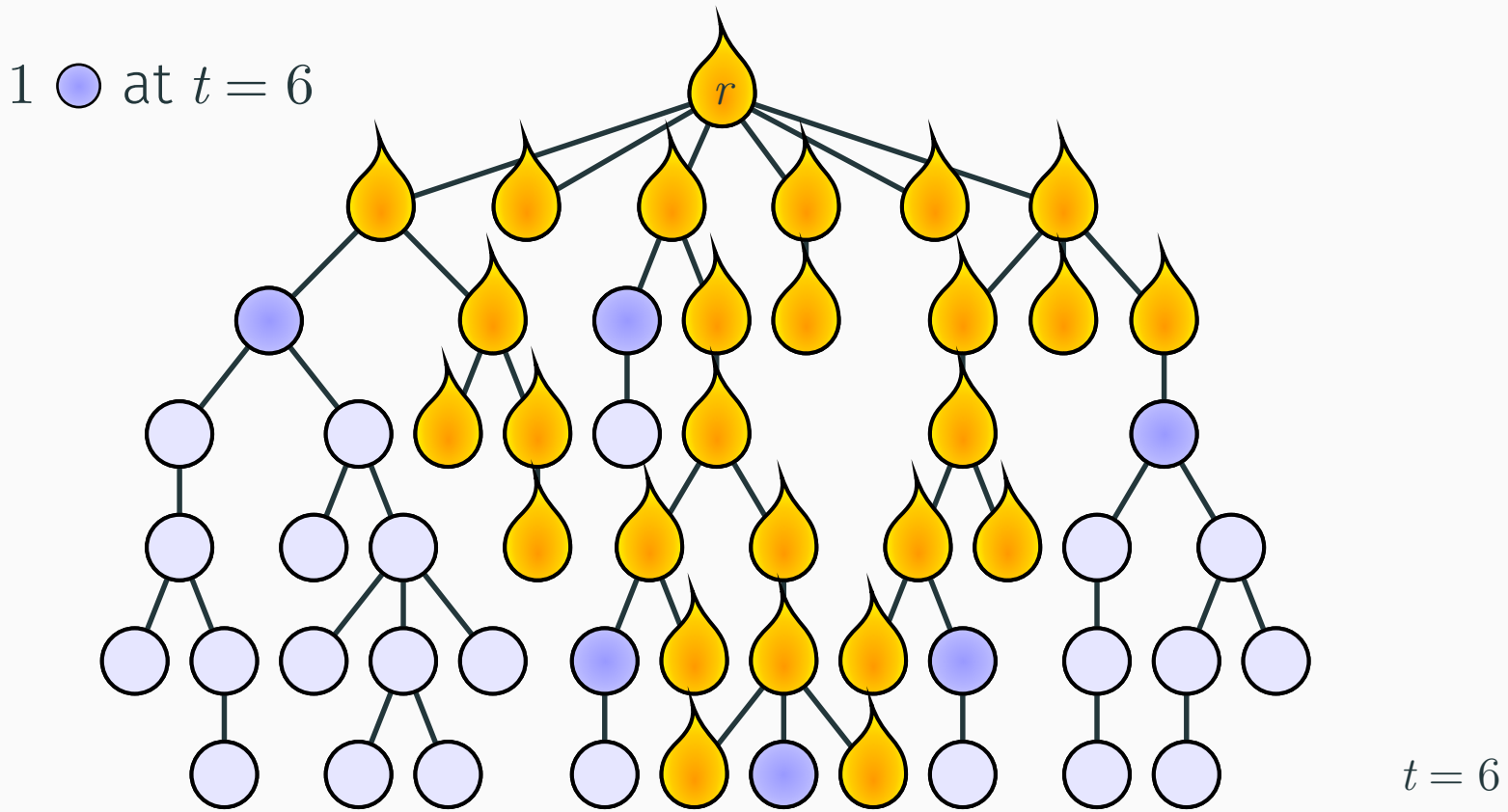
FireFighter Problem - FFP



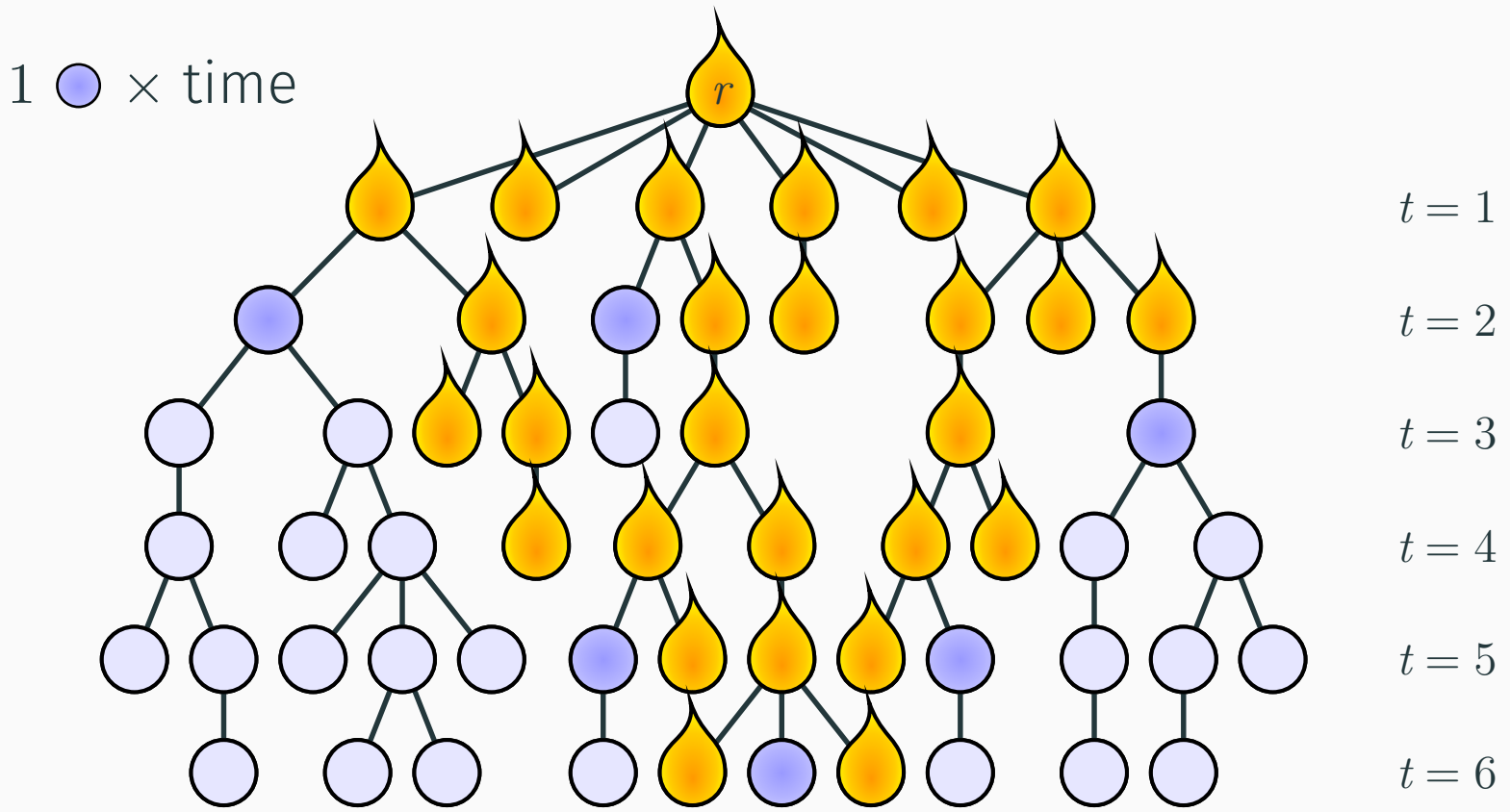
FireFighter Problem - FFP



FireFighter Problem - FFP



FireFighter Problem - FFP

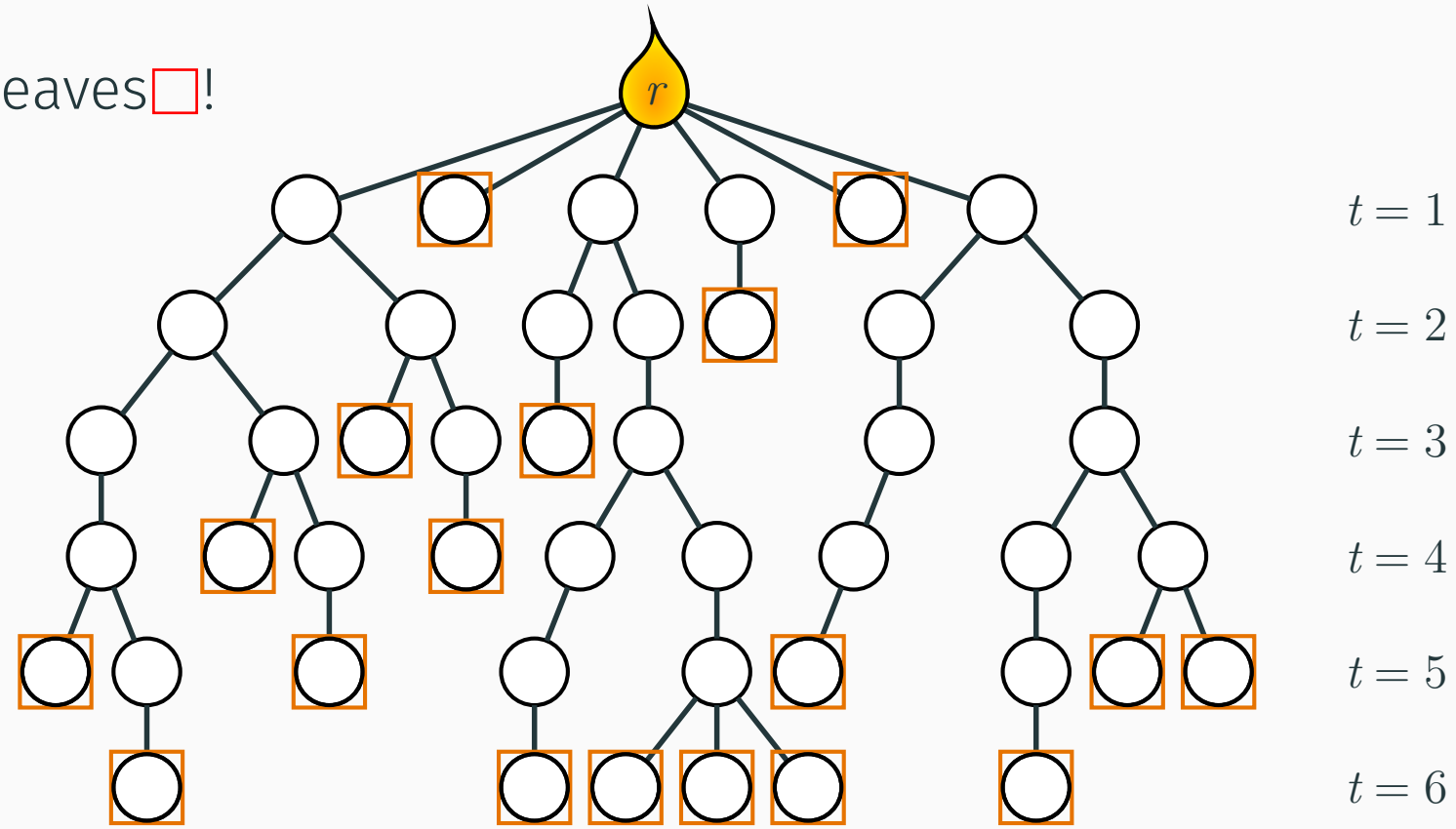


GOAL

Allocate one × time as to maximize saved nodes.

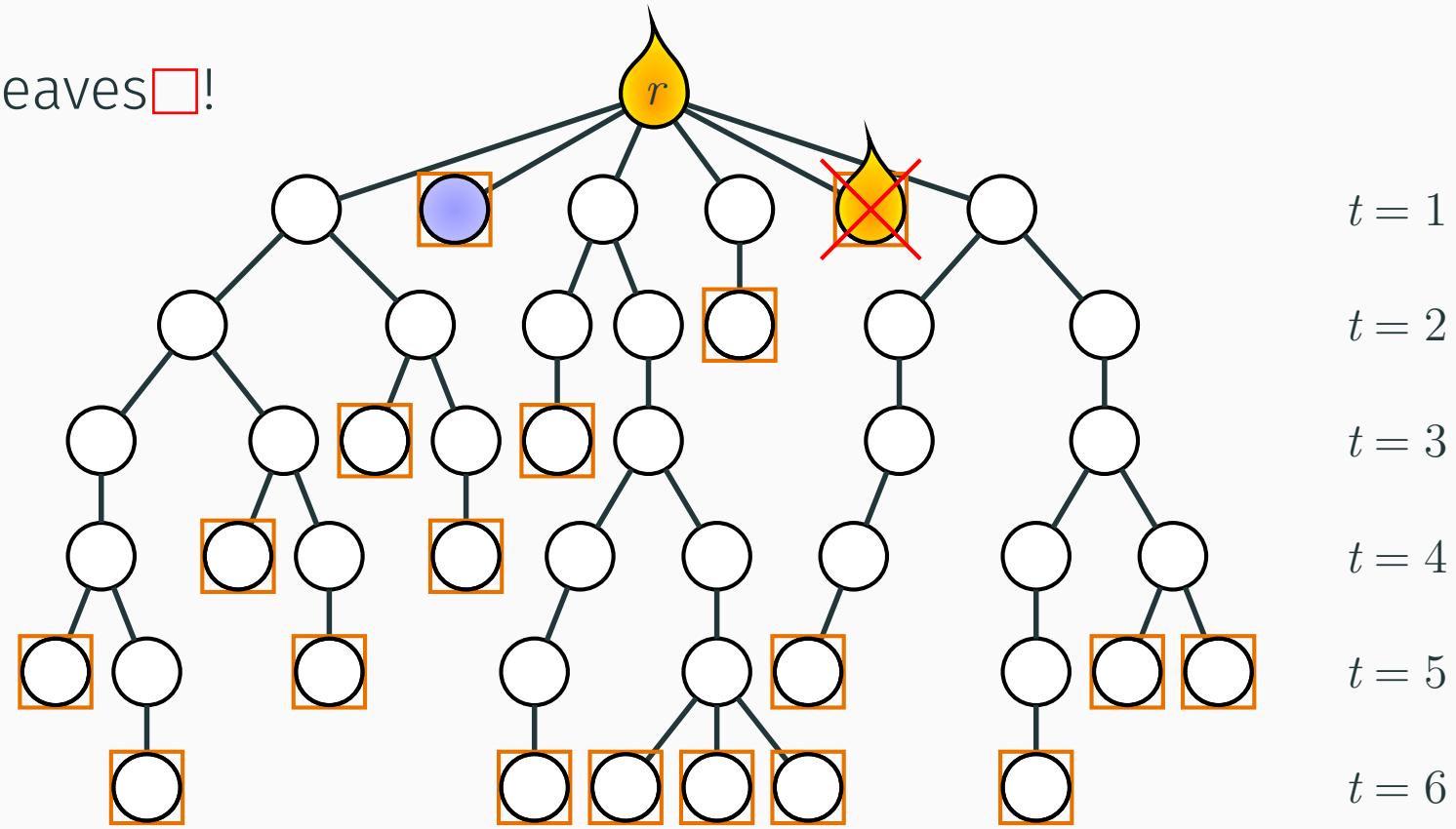
Resource Minimization for Fire Containment - RMFC

save all leaves□!



Resource Minimization for Fire Containment - RMFC

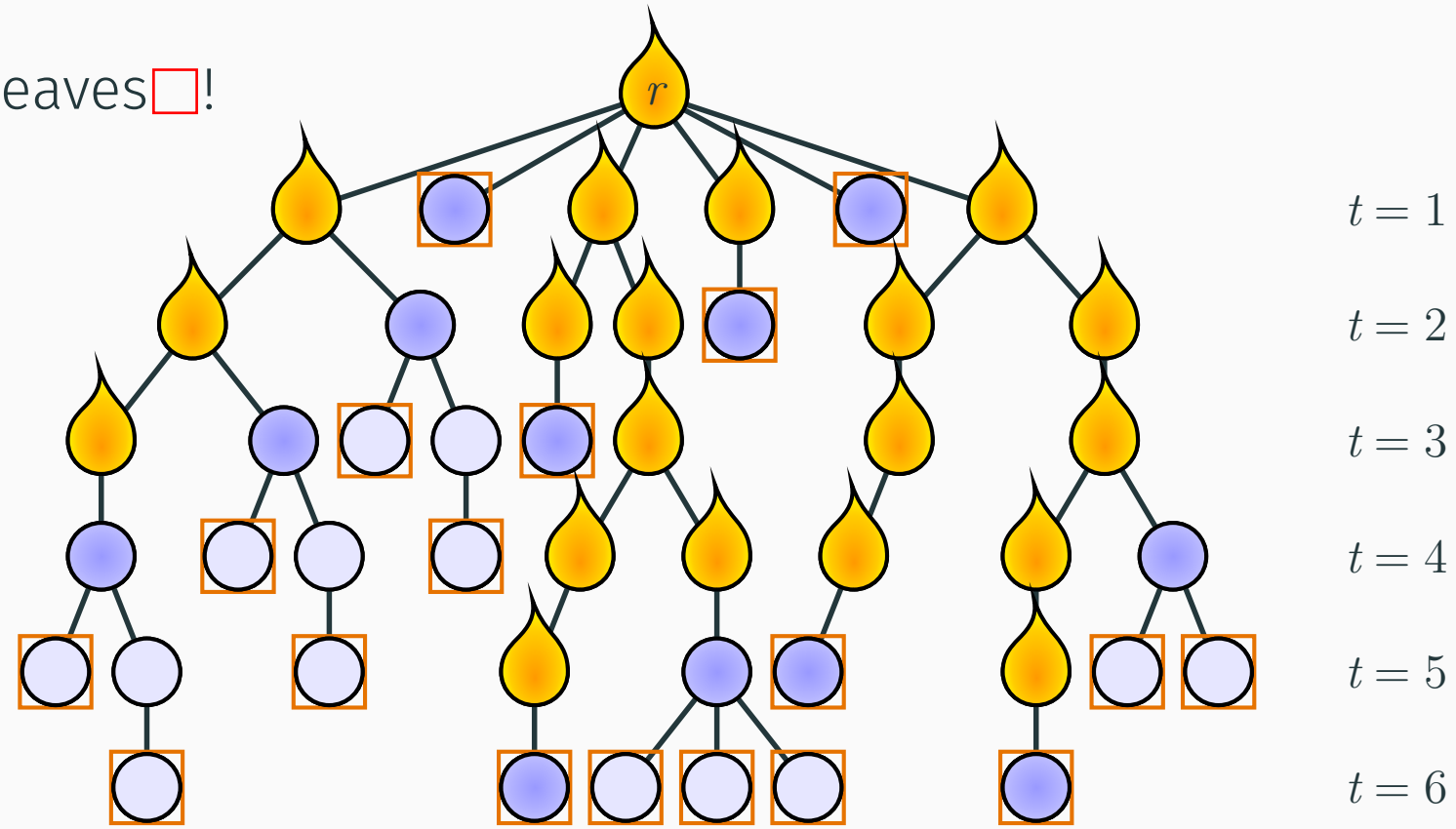
save all leaves □!



Using only one ● × time, impossible!

Resource Minimization for Fire Containment - RMFC

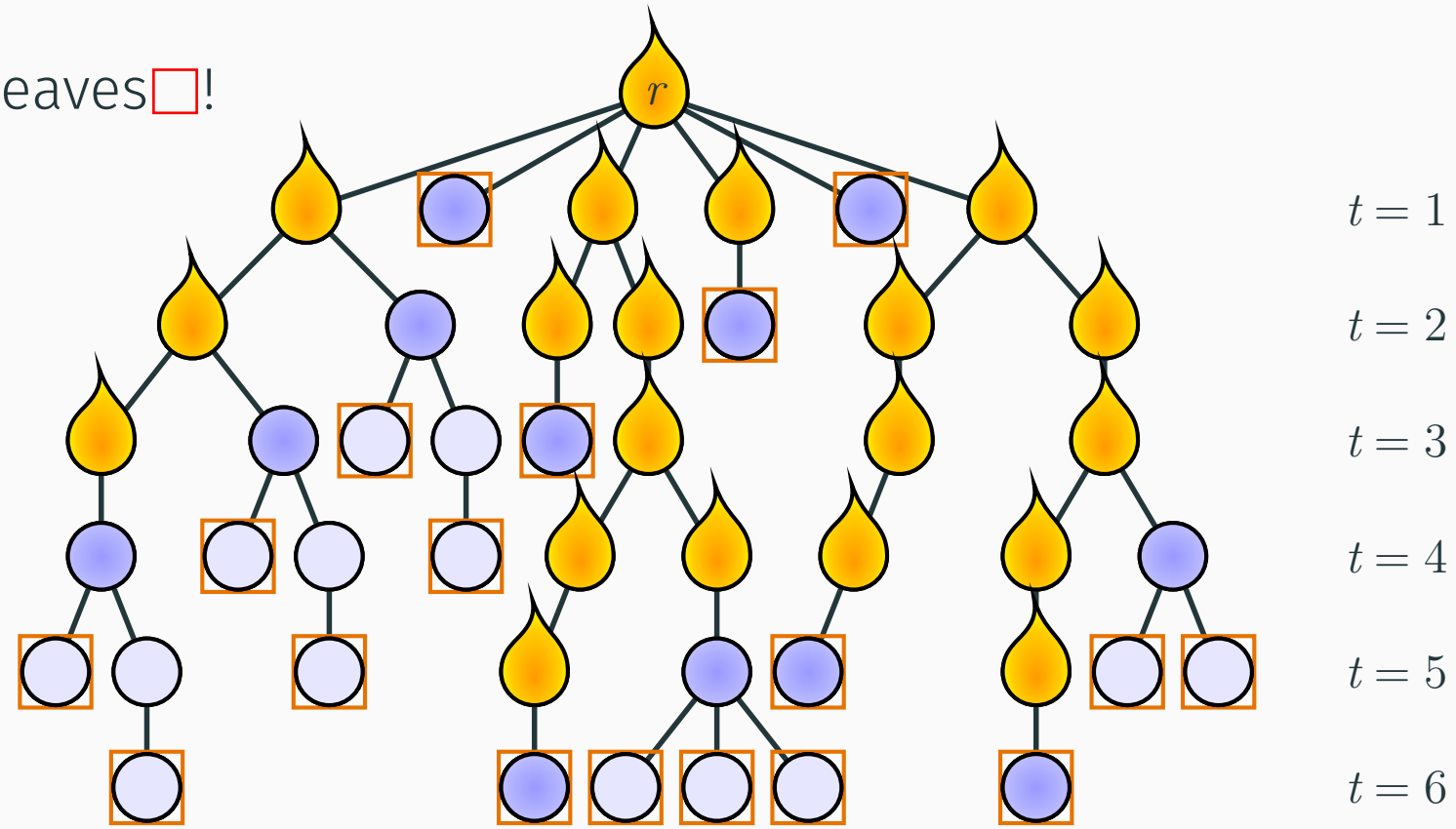
save all leaves □!



With two ● × time, saving all □ is possible!

Resource Minimization for Fire Containment - RMFC

save all leaves□!



GOAL

Minimize number of ○ × time needed to save all leaves.

Previous Results - FireFighter Problem On Trees

Previous Results - FireFighter Problem On Trees

Hardness - Finbow, King, MacGillivray, and Rizzi (2007)

The problem is NP-hard.

Approximation - Hartnell and Li (2000)

Simple greedy algorithm achieves $1/2$-approximation.

Approximation - Cai, Verbin, and Yang (2008)

$(1-1/e)$-approximation (LP based!).

Approx. - Anshelevich, Chakrabarty, Hate, and Swamy (2012)

$(1-1/e)$-approx. via monotone submodular function maximization subject to a partition matroid constraint.

Previous Results - RMFC On Trees

Previous Results - RMFC On Trees

Hardness - Finbow, King, MacGillivray, and Rizzi (2007)

NP-hard to approximate within any factor better than 2.

$O(\log n)$-approx. via constant-factor approximation for FFP.

Approximation - Chalermsook and Chuzhoy (2010)

$O(\log^* n)$-approximation (LP based!).

Do Integrality Gaps Reflect Approximation Hardness?

Current best algorithms for FFP and RMFC are LP based.

FFP

$(1-1/e)$ matches integr. gap.

(Chalermsook and Vaz (2016))

RMFC

$O(\log^* n)$ matches integr. gap up to constant-factor.

(Chalermsook & Chuzhoy (2010))

Answer not clear before!

Do Integrality Gaps Reflect Approximation Hardness?

Current best algorithms for FFP and RMFC are LP based.

FFP

$(1-1/e)$ matches integr. gap.

(Chalermsook and Vaz (2016))

RMFC

$O(\log^* n)$ matches integr. gap up to constant-factor.

(Chalermsook & Chuzhoy (2010))

Answer not clear before!

FFP

Special case of monotone submodular max. subject to partition matroid.
(no $(1-1/e+\epsilon)$-approx Fisher, Nemhauser and Wolsey [1978], Feige [1998]).

RMFC

Similar to the Asymmetric k-center problem.
(no $o(\log^* n)$-approx, unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\log\log n})$ (Chuzhoy, Guha, Halperin, Khanna, Kortsarz, Krauthgamer, and Naor (2005))).

Our Contributions

Theorem - Adjiashvili, Baggio, and Zenklusen (2016)

PTAS for the FireFighter Problem on trees.

Theorem - Adjiashvili, Baggio, and Zenklusen (2016)

$O(1)$-approximation for RMFC on trees.

Our Contributions

Theorem - Adjiashvili, Baggio, and Zenklusen (2016)

PTAS for the FireFighter Problem on trees.

Theorem - Adjiashvili, Baggio, and Zenklusen (2016)

$O(1)$-approximation for RMFC on trees.

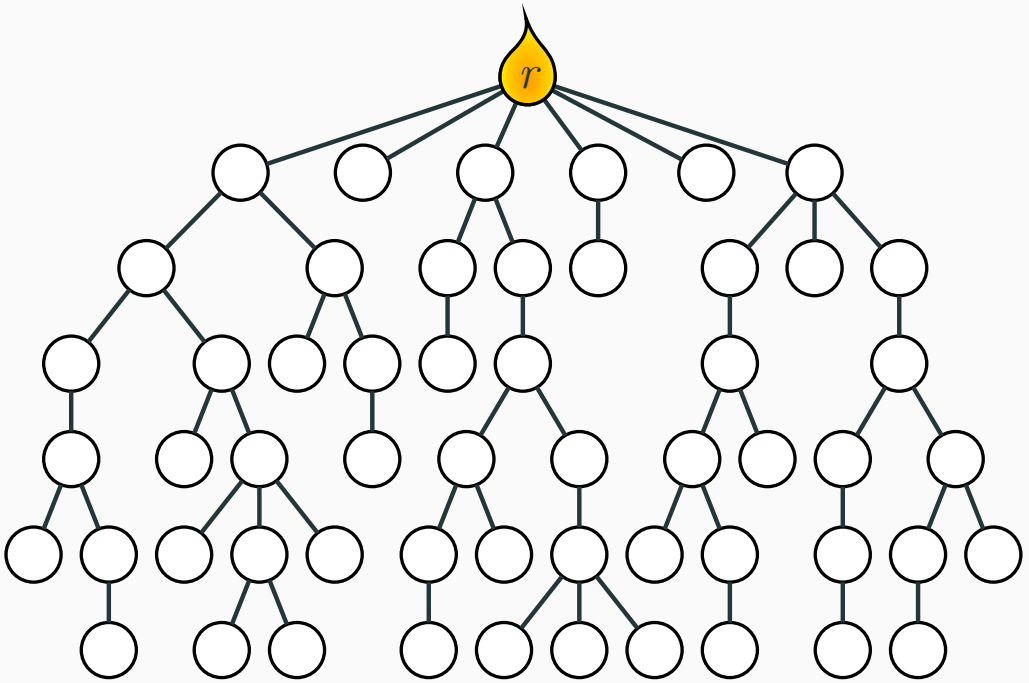
Both algorithms are guided by the same known LPs!

We introduce techniques to go beyond integrality gaps.

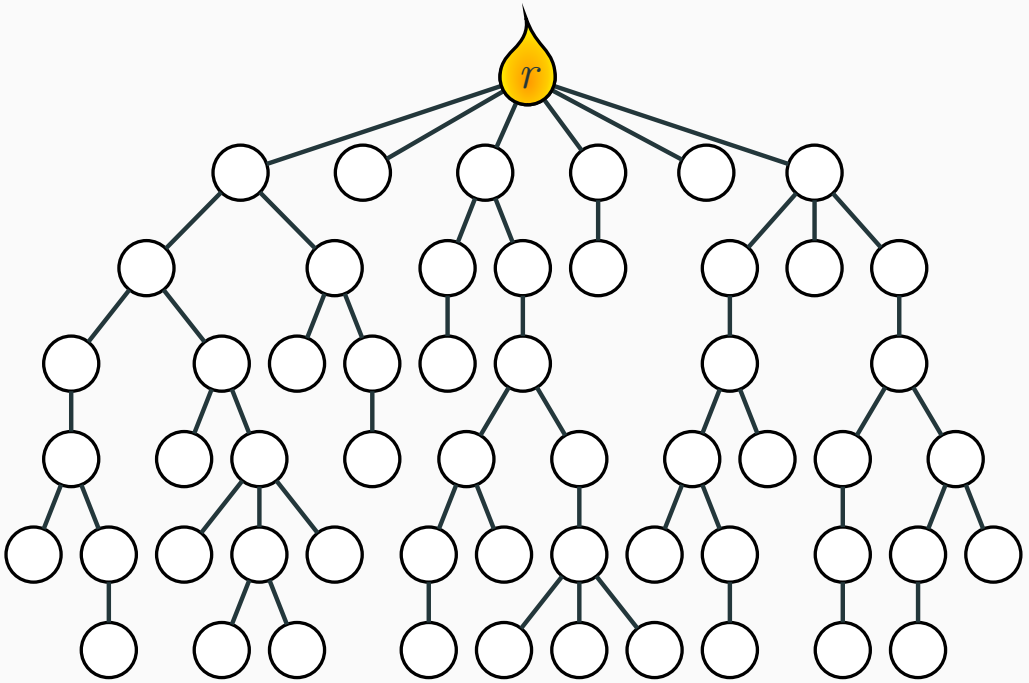
→ In particular: (partial) enumeration to identify super-constant size set of constraints to add to LP.

PTAS For FireFighter Problem

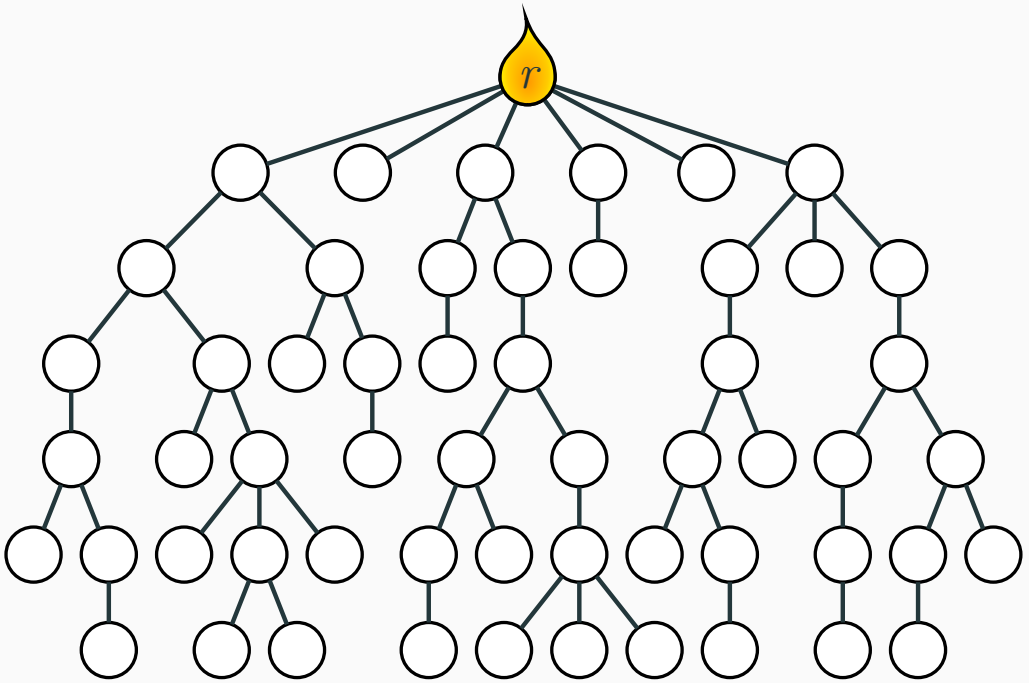
The Linear Program



The Linear Program

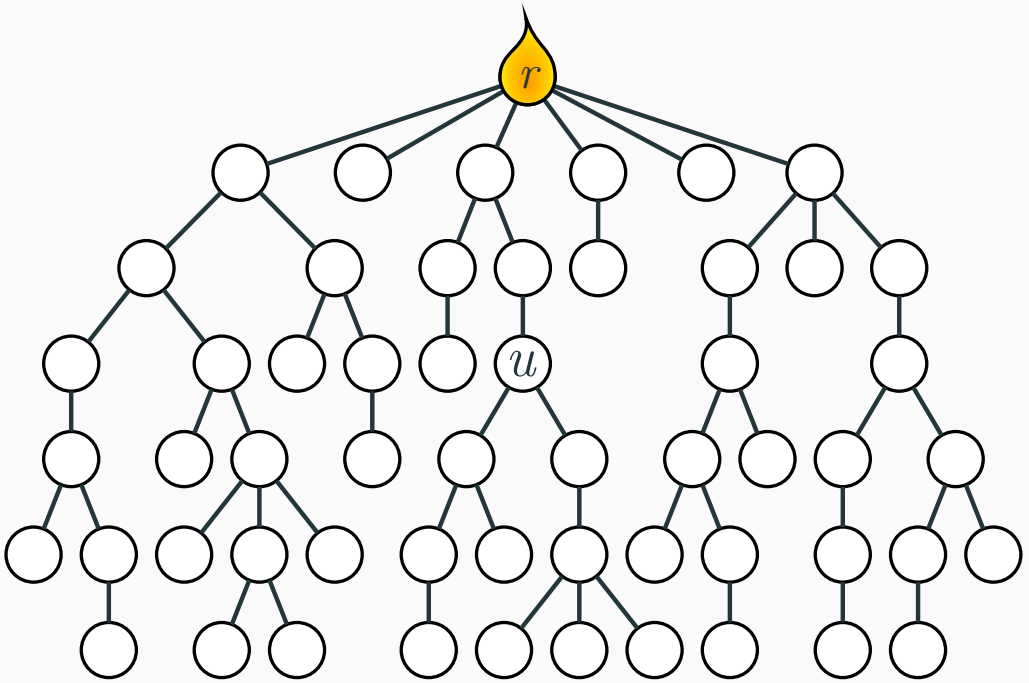


The Linear Program



$V=$ nodes

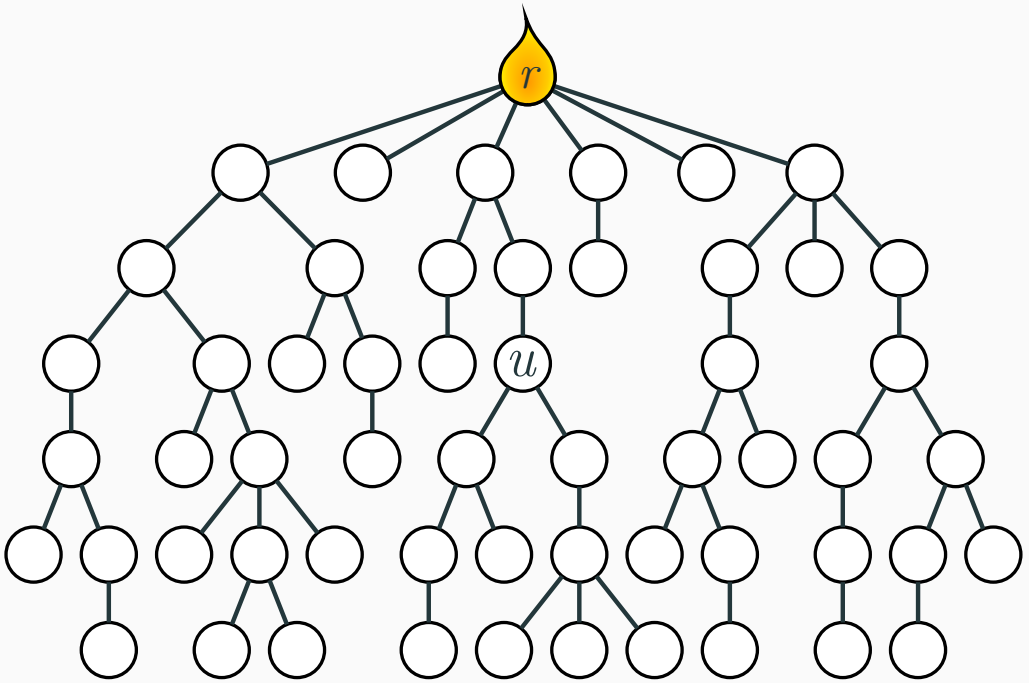
The Linear Program



$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bullet \\ 0 & \text{ow.} \end{cases}$$

The Linear Program

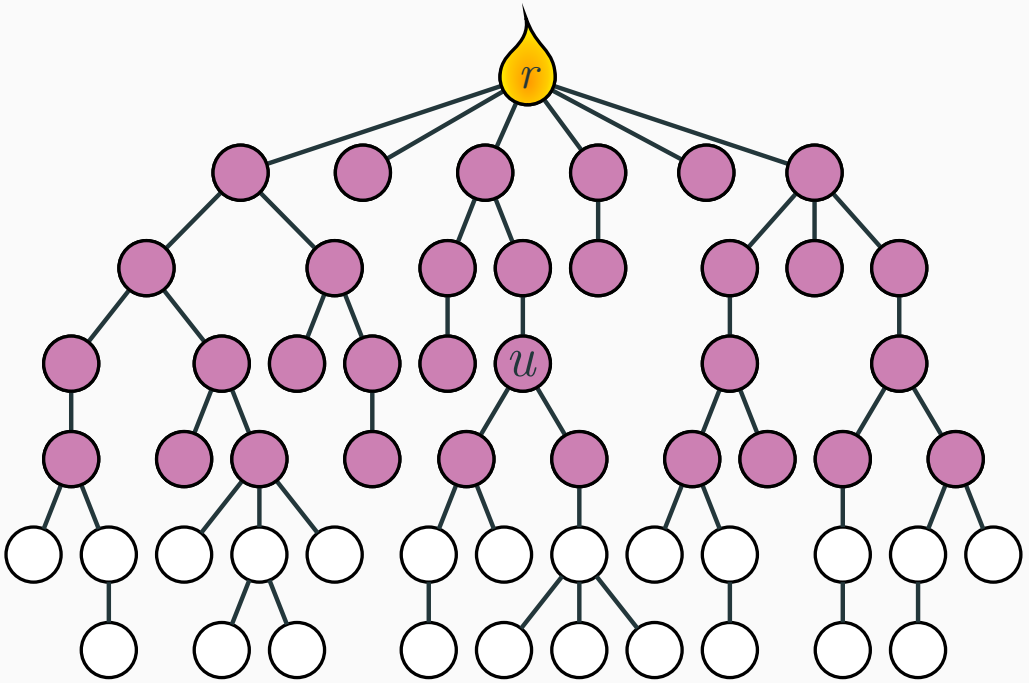


$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bullet \\ 0 & \text{ow.} \end{cases}$$

$$x_u \in \{0,1\} \qquad \forall\ u \in V \setminus \{r\}$$

The Linear Program



$V =$ nodes

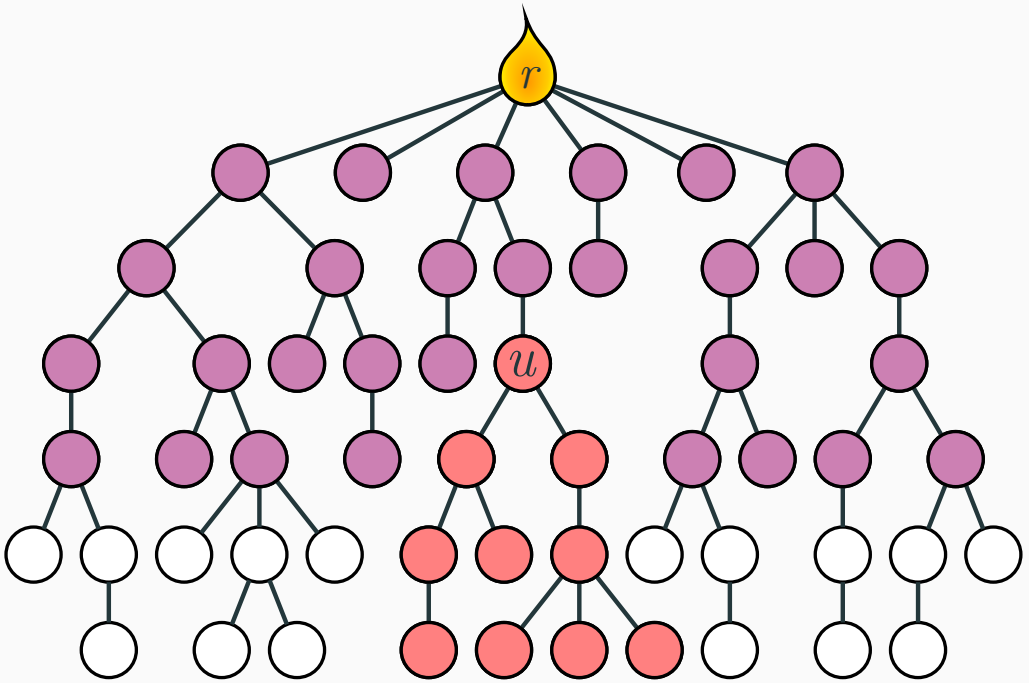
$$x_u = \begin{cases} 1 & \text{if } \bigcirc \\ 0 & \text{ow.} \end{cases}$$

$V_{\leq t} =$ levels $\leq t$

$$x(V_{\leq t}) \leq t \qquad \forall\ t = 1, \ldots, L$$

$$x_u \in \{0, 1\} \qquad \forall\ u \in V \setminus \{r\}$$

The Linear Program



$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bigcirc \\ 0 & \text{ow.} \end{cases}$$

$V_{\le t} =$ levels $\le t$

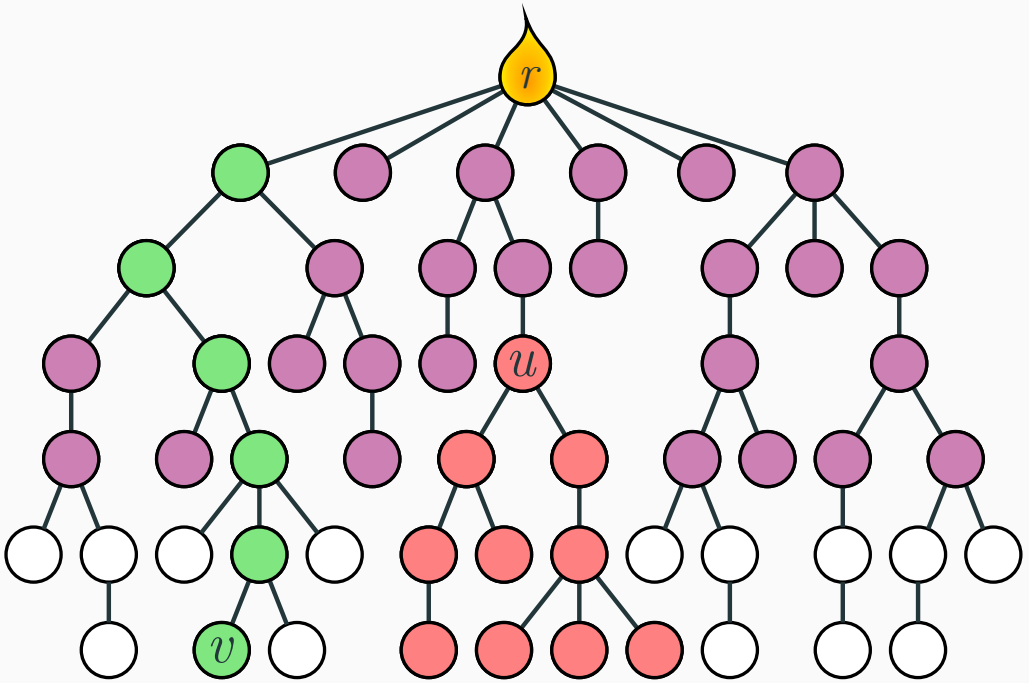
$c_u =$ |subtree of u|

$$\max \quad \sum_{u \in V} c_u \, x_u$$

$$x(V_{\le t}) \le t \qquad \forall\ t = 1, \ldots, L$$

$$x_u \in \{0, 1\} \qquad \forall\ u \in V \setminus \{r\}$$

The Linear Program



$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bigcirc \\ 0 & \text{ow.} \end{cases}$$

$V_{\le t} =$ levels $\le t$

$c_u =$ |subtree of u|

$P_v =$ path $r \to v$

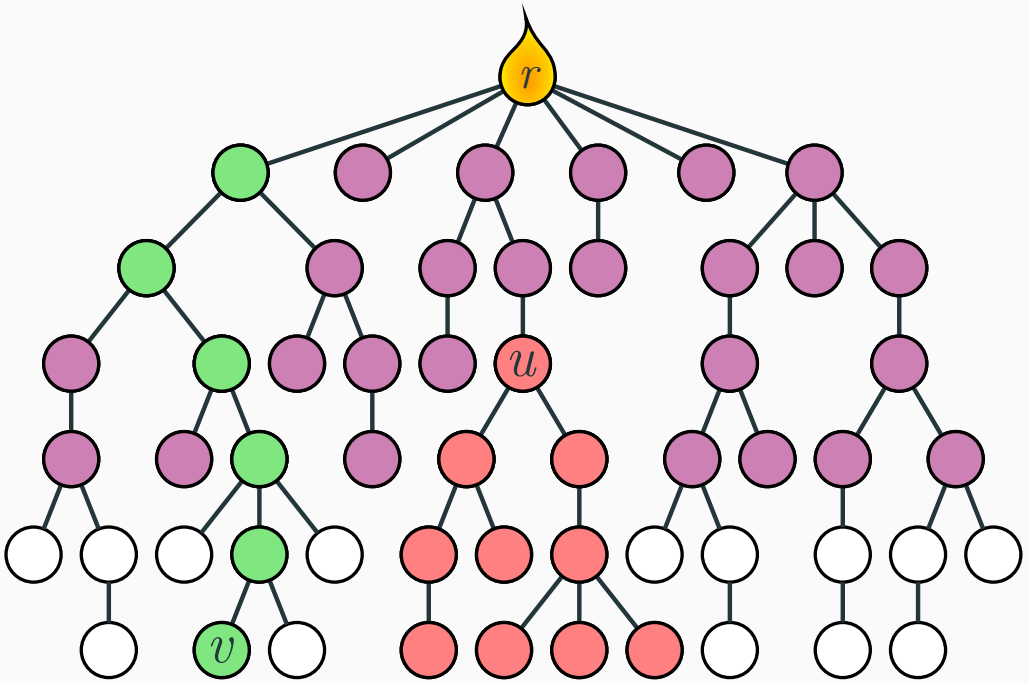
$$\max \quad \sum_{u \in V} c_u \, x_u$$

$$x(P_v) \le 1 \qquad \forall\ v \in \text{leaves}$$

$$x(V_{\le t}) \le t \qquad \forall\ t = 1, \ldots, L$$

$$x_u \in \{0, 1\} \qquad \forall\ u \in V \setminus \{r\}$$

The Linear Program



$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bigcirc \\ 0 & \text{ow.} \end{cases}$$

$V_{\leq t} =$ levels $\leq t$

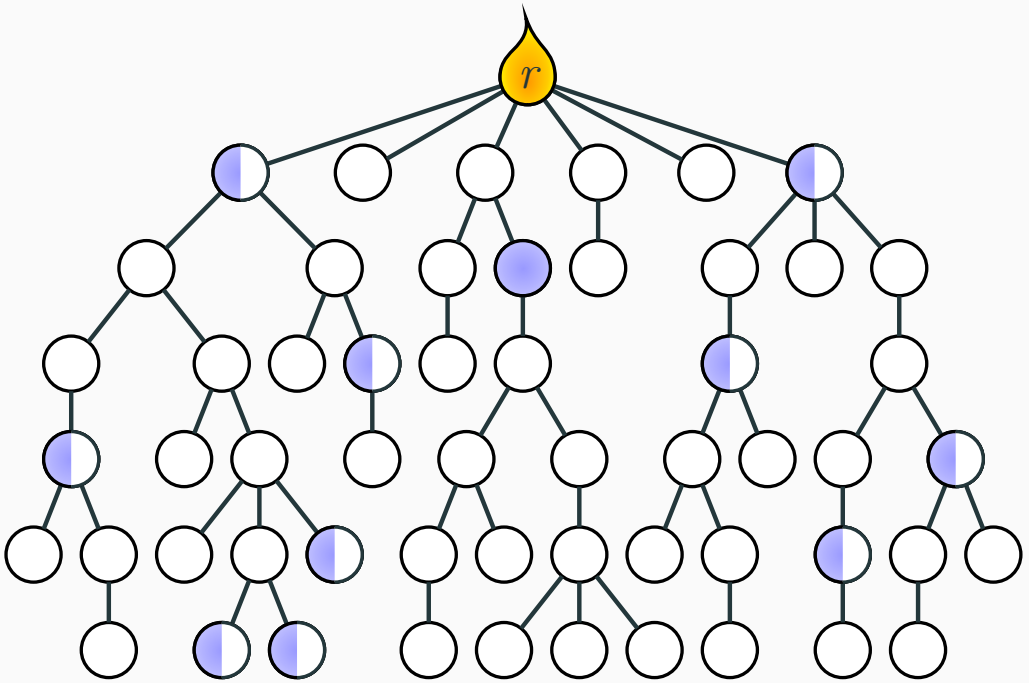
$c_u =$ |subtree of u|

$P_v =$ path $r \to v$

$$\max \sum_{u \in V} c_u \, x_u$$

$$(LP) \qquad \begin{aligned} x(P_v) &\leq 1 && \forall\ v \in \text{leaves} \\ x(V_{\leq t}) &\leq t && \forall\ t = 1, \ldots, L \\ x_u &\in [0,1] && \forall\ u \in V \setminus \{r\} \end{aligned}$$

The Linear Program



$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bullet \\ 0 & \text{ow.} \end{cases}$$

$V_{\le t} =$ levels $\le t$

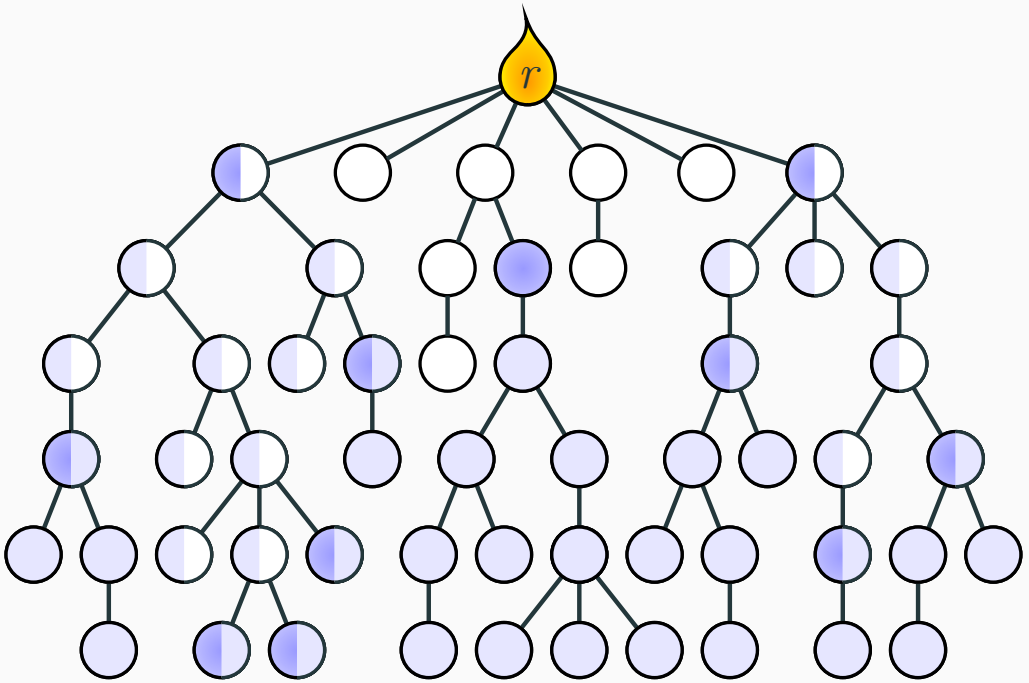
$c_u =$ |subtree of u|

$P_v =$ path $r \to v$

$$\max \sum_{u \in V} c_u \, x_u$$

$$(LP) \qquad \begin{aligned} x(P_v) &\le 1 && \forall\ v \in \text{leaves} \\ x(V_{\le t}) &\le t && \forall\ t = 1, \dots, L \\ x_u &\in [0,1] && \forall\ u \in V \setminus \{r\} \end{aligned}$$

The Linear Program



$V =$ nodes

$$x_u = \begin{cases} 1 & \text{if } \bullet \\ 0 & \text{ow.} \end{cases}$$

$V_{\leq t} =$ levels $\leq t$

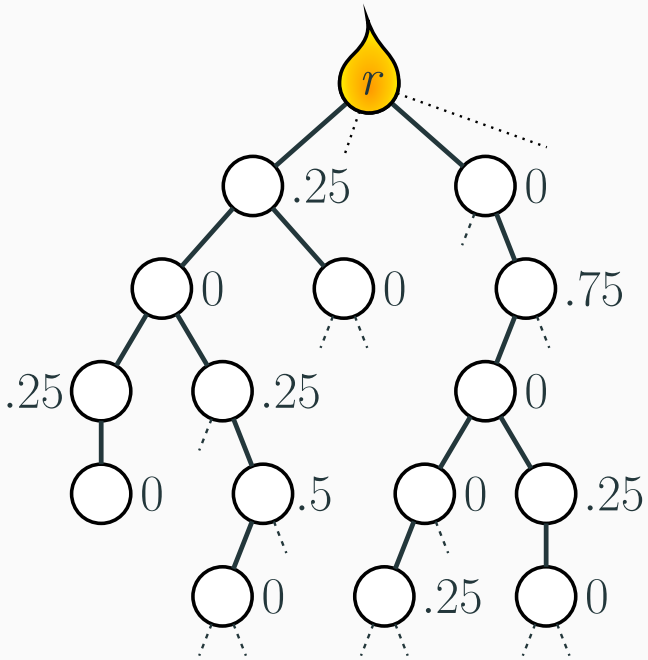
$c_u =$ |subtree of u|

$P_v =$ path $r \to v$

$$\max \quad \sum_{u \in V} c_u \, x_u$$

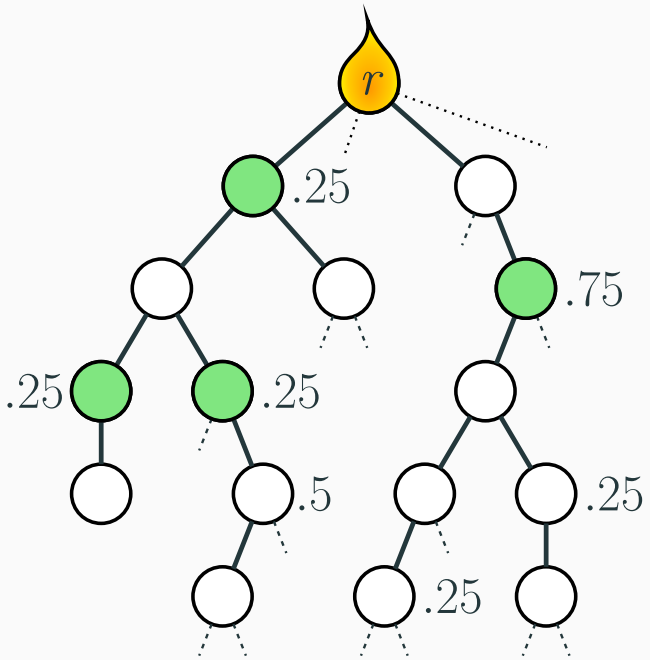
$$(LP) \qquad \begin{aligned} x(P_v) &\leq 1 && \forall\ v \in \text{leaves} \\ x(V_{\leq t}) &\leq t && \forall\ t = 1, \ldots, L \\ x_u &\in [0,1] && \forall\ u \in V \setminus \{r\} \end{aligned}$$

Loose and Tight Nodes



Let x^* be a vertex sol. of (LP).

Loose and Tight Nodes

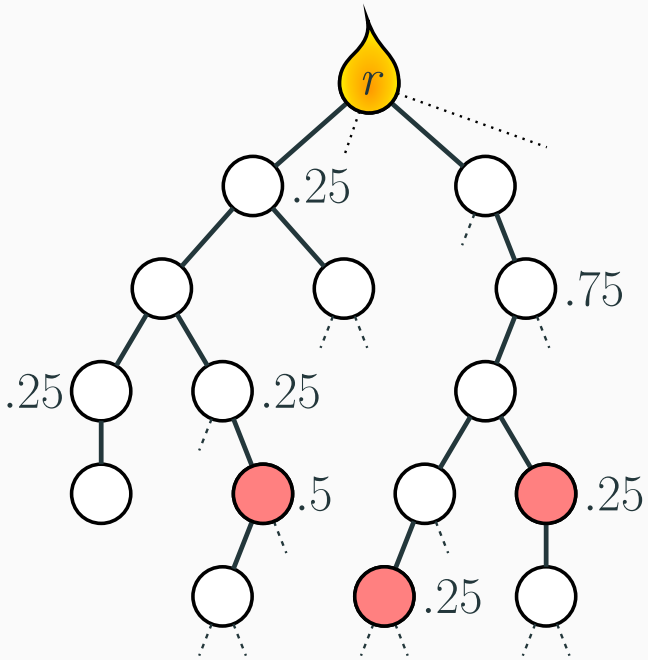


Let x^* be a vertex sol. of (LP).

A node u is loose w.r.t. x^* if

- $x^*_u > 0$
- $x^*(P_u) < 1$

Loose and Tight Nodes



Let x^* be a vertex sol. of (LP).

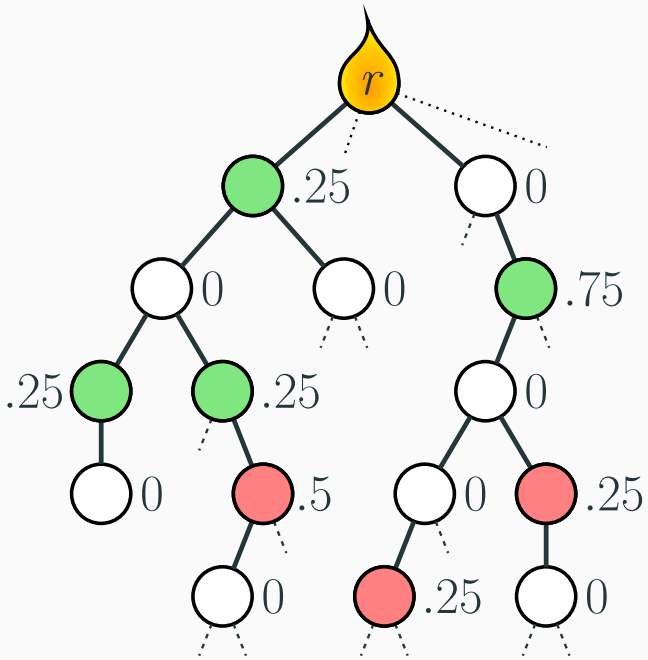
A node u is loose w.r.t. x^* if

- $x^*_u > 0$
- $x^*(P_u) < 1$

A node u is tight w.r.t. x^* if

- $x^*_u > 0$
- $x^*(P_u) = 1$

Loose and Tight Nodes



Let x^* be a vertex sol. of (LP).

A node u is loose w.r.t. x^* if
- $x^*_u > 0$
- $x^*(P_u) < 1$

A node u is tight w.r.t. x^* if
- $x^*_u > 0$
- $x^*(P_u) = 1$

Naive (but inspiring) plan: reallocate ● and protect 1-vertices

1. For each ●: either do nothing, or push fraction down to some ●.
2. Protect all 1-vertices.

PLAN - Reallocate Loose Nodes

PLAN - Reallocate Loose Nodes

We want loss from reallocating ● be at most $\epsilon \operatorname{val}(\mathrm{OPT})$.

PLAN - Reallocate Loose Nodes

We want loss from reallocating ● be at most $\epsilon \operatorname{val}(\mathrm{OPT})$.

● have to become few
and reallocation light!

PLAN - Reallocate Loose Nodes

We want loss from reallocating ● be at most $\epsilon \operatorname{val}(\mathrm{OPT})$.

● have to become few
and reallocation light!

$\Leftarrow$

- Tree transformations.
- Strengthen LP.

PLAN - Reallocate Loose Nodes

We want loss from reallocating ● be at most $\epsilon \operatorname{val}(\mathrm{OPT})$.

● have to become few
and reallocation light!

$\Leftarrow$

- Tree transformations.
- Strengthen LP.

Bound on number of ●:

Sparsity Lemma

Number of loose nodes at most L (depth of tree).

PLAN - Reallocate Loose Nodes

We want loss from reallocating ● be at most $\epsilon \operatorname{val}(\mathrm{OPT})$.

● have to become few
and reallocation light!

$\Leftarrow$

- Tree transformations.
- Strengthen LP.

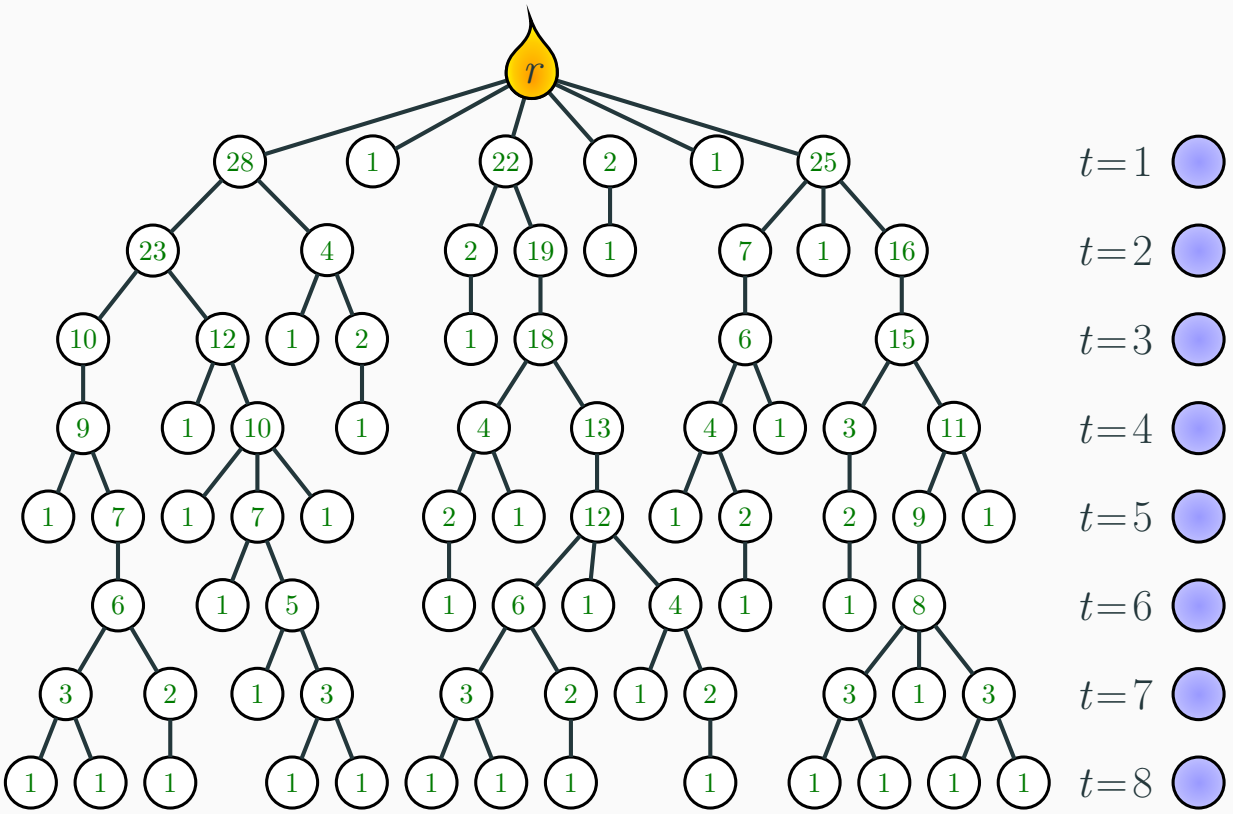
Bound on number of ●:

Sparsity Lemma

Number of loose nodes at most L (depth of tree).

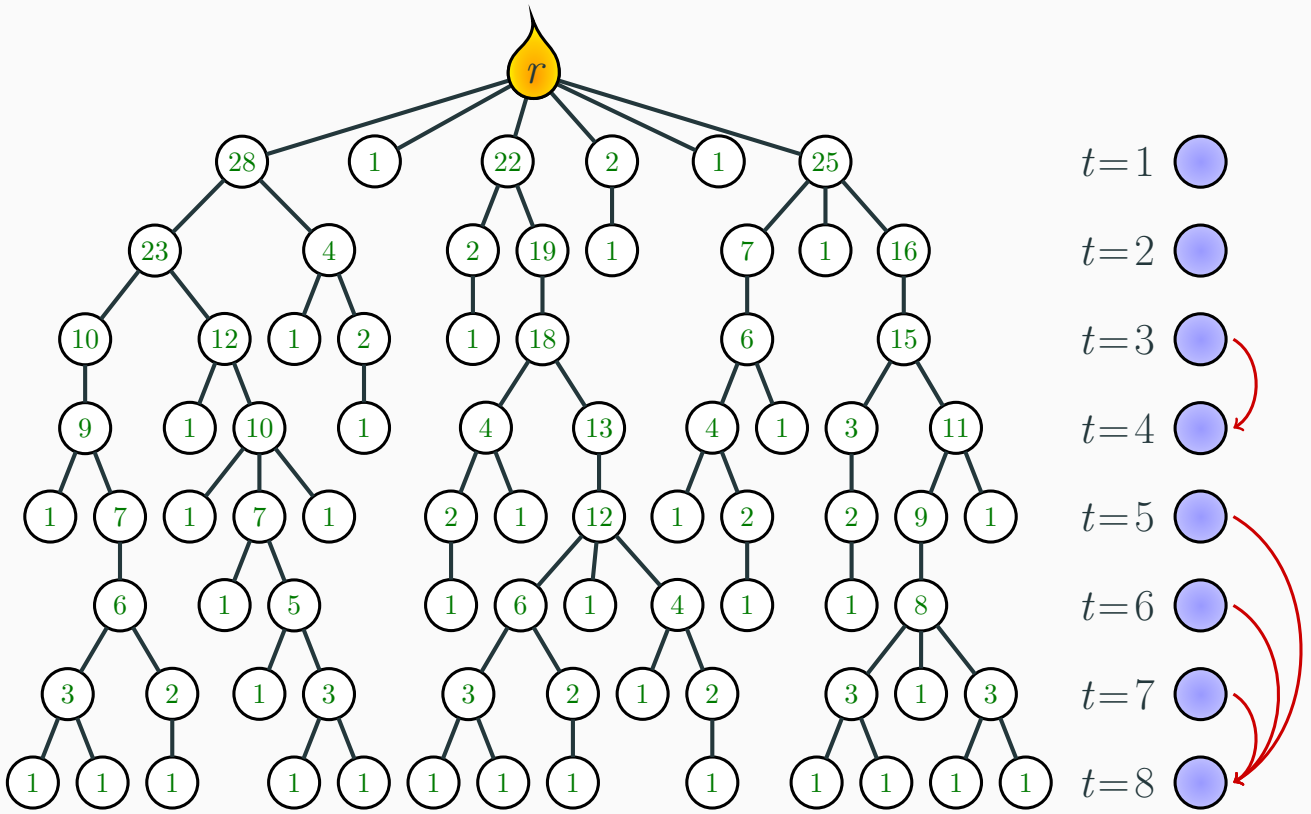
We want to reduce L!

Compression - Reducing Depth L



c_u

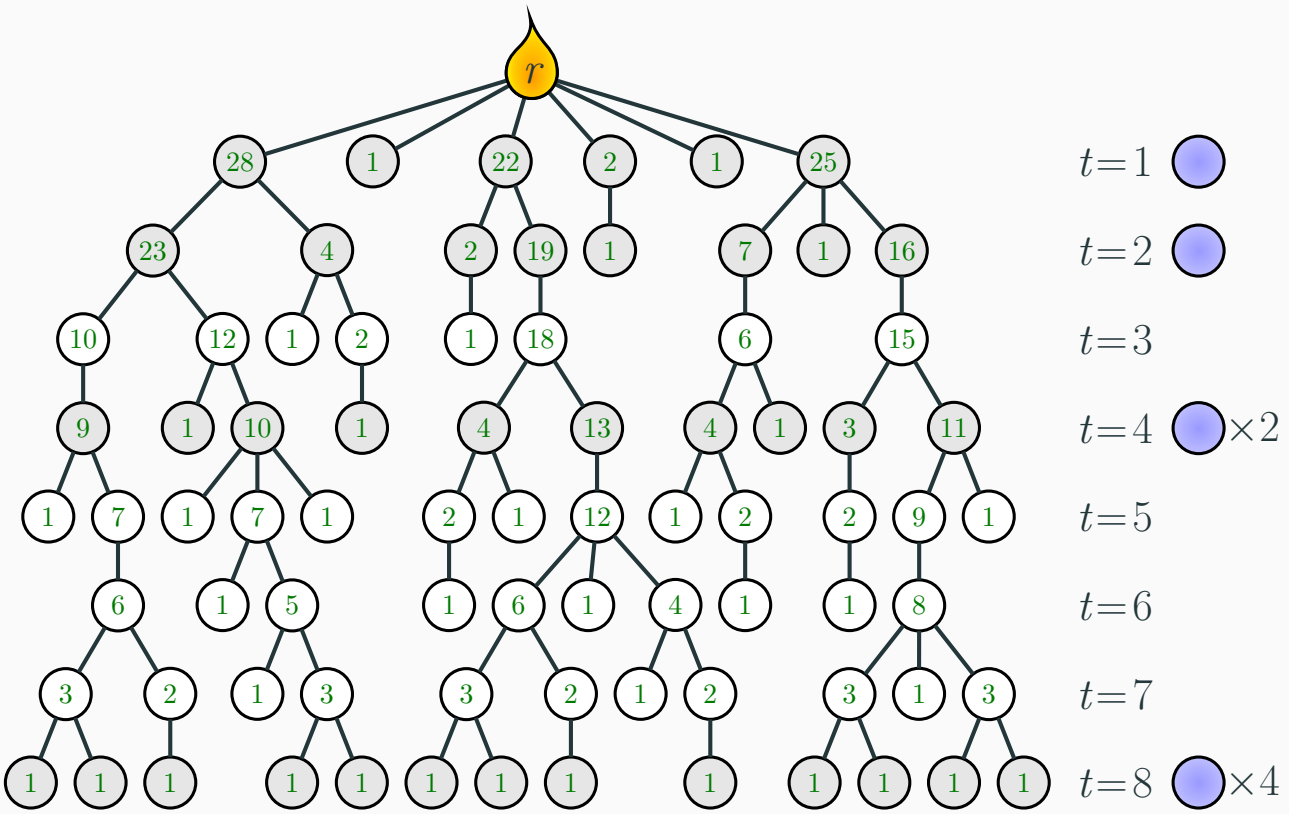
Compression - Reducing Depth L



c_u

push down

Compression - Reducing Depth L

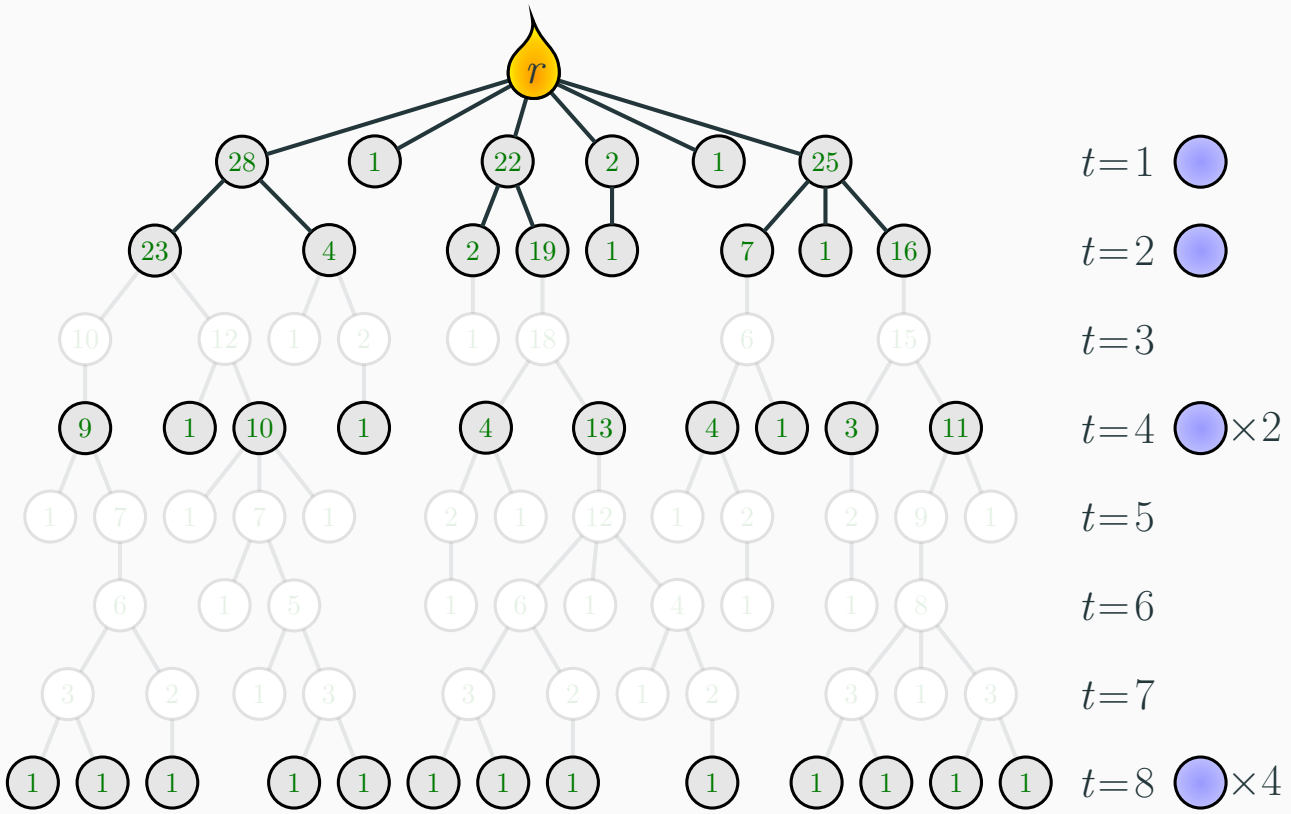


c_u

push down

$l = 2^i,\ i = 0,1,\ldots$

Compression - Reducing Depth L



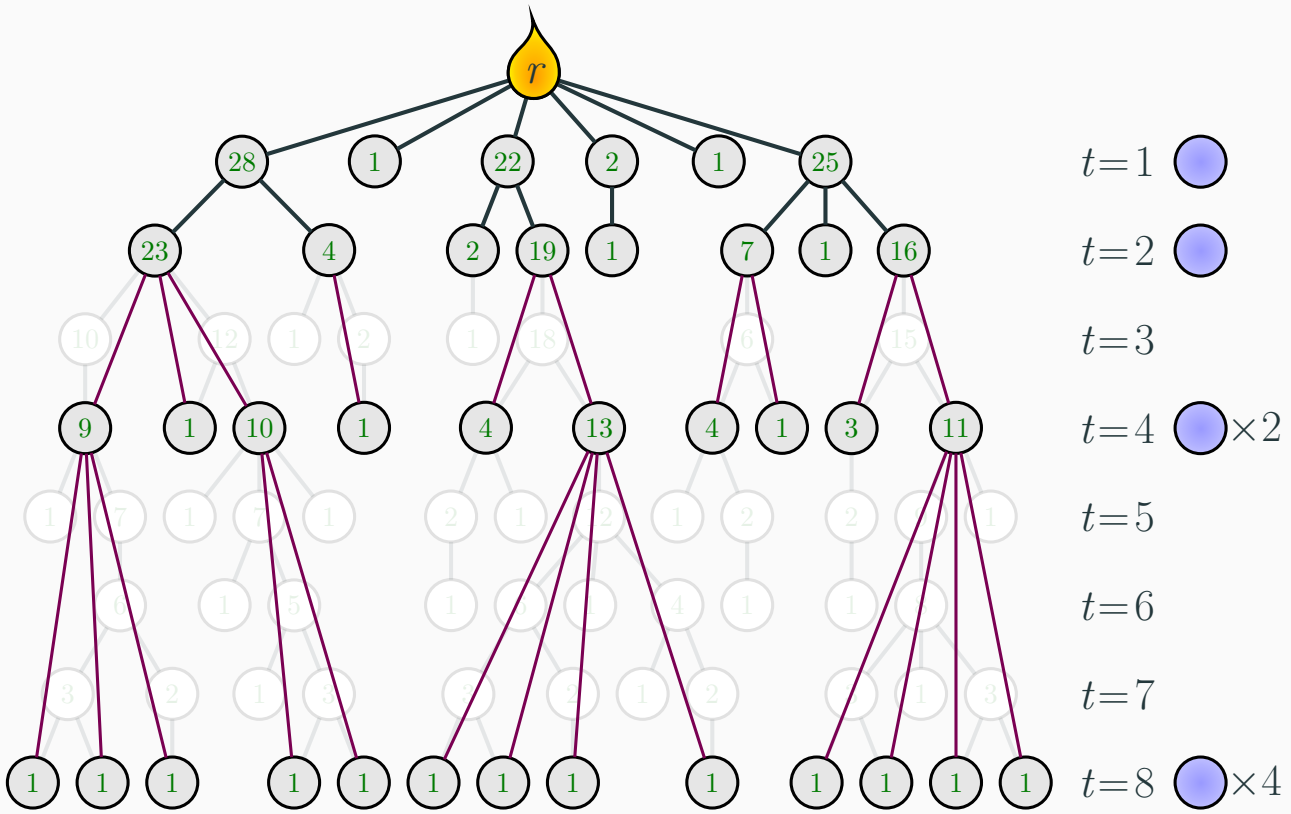
c_u

push down

$l=2^i,\ i=0,1,\ldots$

erase

Compression - Reducing Depth L



c_u

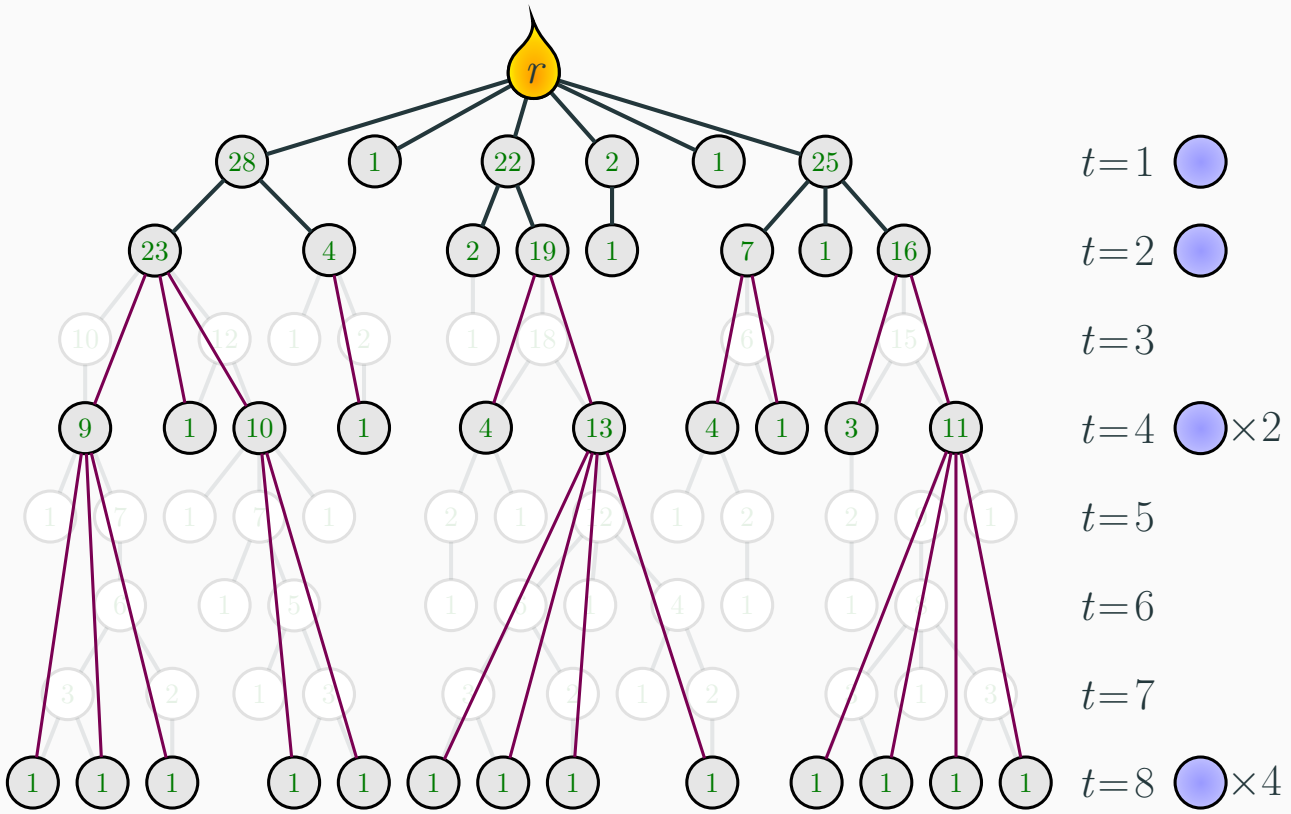
push down

$l=2^i,\ i=0,1,\ldots$

erase

connect

Compression - Reducing Depth L



c_u

push down

$l = 2^i,\ i = 0,1,\ldots$

erase

connect

- $\mathrm{val}(\mathrm{OPT}_{comp}) \geq 1/2\, \mathrm{val}(\mathrm{OPT}_{orig})$.
- Sol. to compressed instance has same value in orig. instance.
- Compressed tree has $O(\log(L))$ levels.

δ-COMPRESSION

Previously, we selected $l = 2^i$, $i = 0, 1, \ldots$

What if we select $l = \lfloor (1+\delta)^n \rfloor$, $n = 0, 1, \ldots$, $\delta \in]0, 1[$?

δ-COMPRESSION

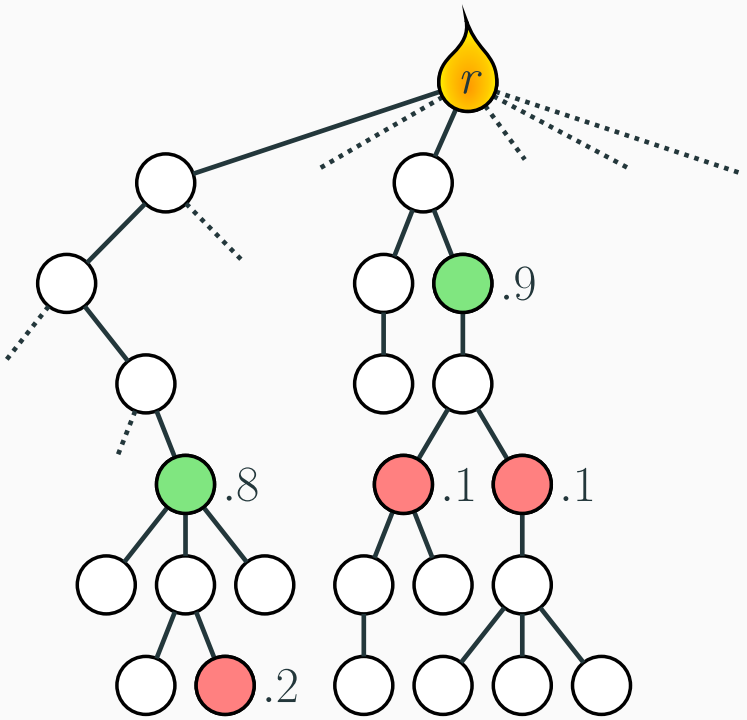
Previously, we selected $l = 2^i$, $i = 0, 1, \ldots$

What if we select $l = \lfloor (1+\delta)^n \rfloor$, $n = 0, 1, \ldots$, $\delta \in]0, 1[$?

Lemma

- $\mathrm{val}(\mathrm{OPT}_{comp}) \geq (1-\delta)\,\mathrm{val}(\mathrm{OPT}_{orig})$.
- Sol. to compressed instance has same value in orig. instance.
- Compressed tree has $O\left(\log L/\delta\right)$ levels.

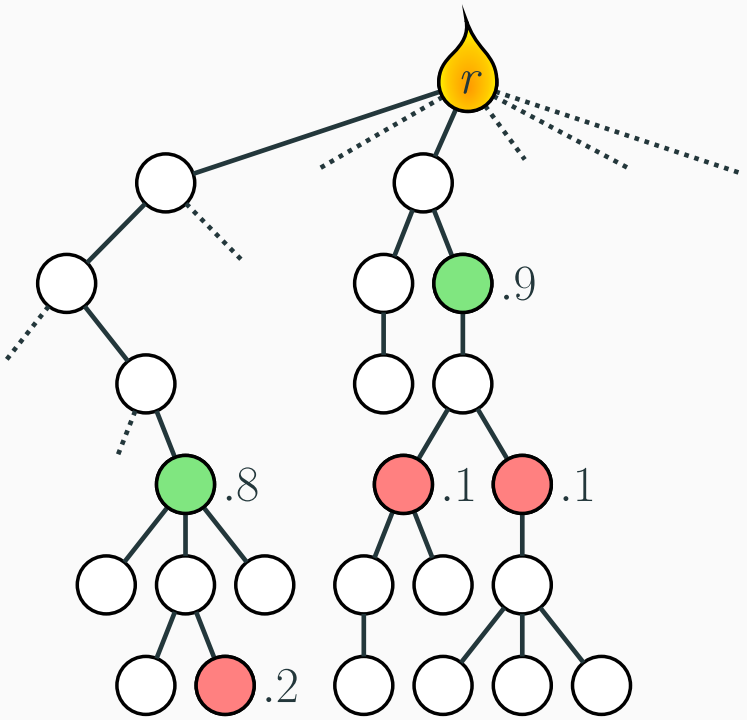
Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

Critical Nodes - Bounding Loss From Singular Reallocation



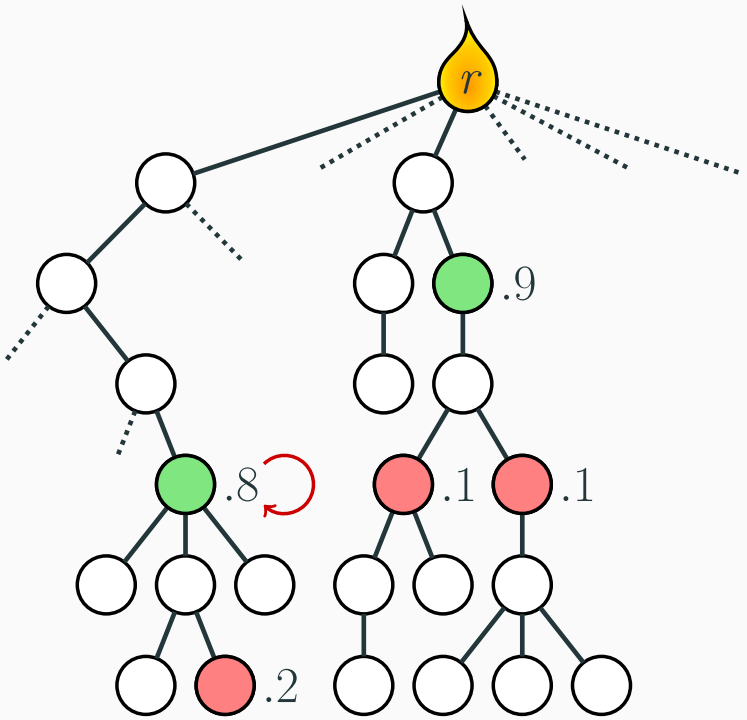
loose

tight

Plan: reallocate ● and protect 1-vertices

1. For each ●: {

Critical Nodes - Bounding Loss From Singular Reallocation



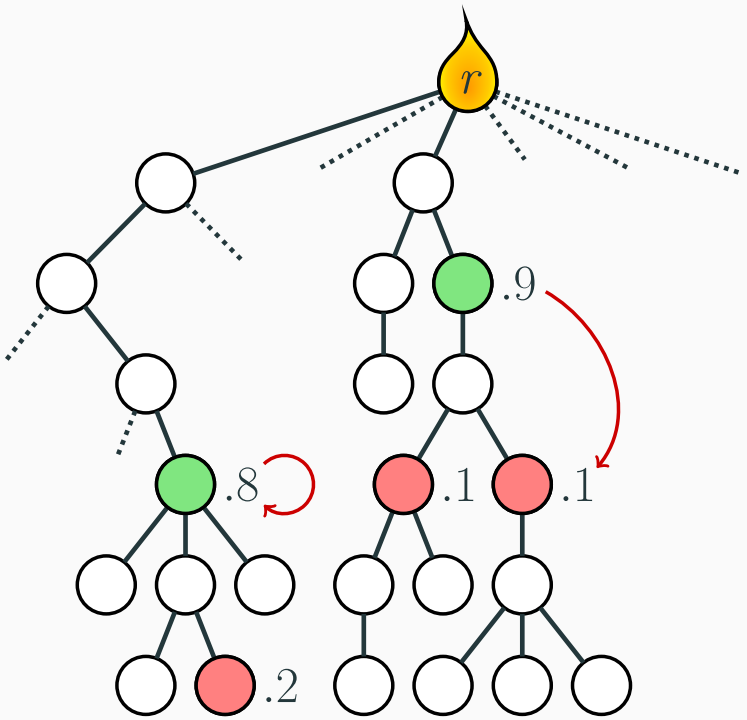
loose

tight

Plan: reallocate ● and protect 1-vertices

1. For each ●: { either do nothing,

Critical Nodes - Bounding Loss From Singular Reallocation



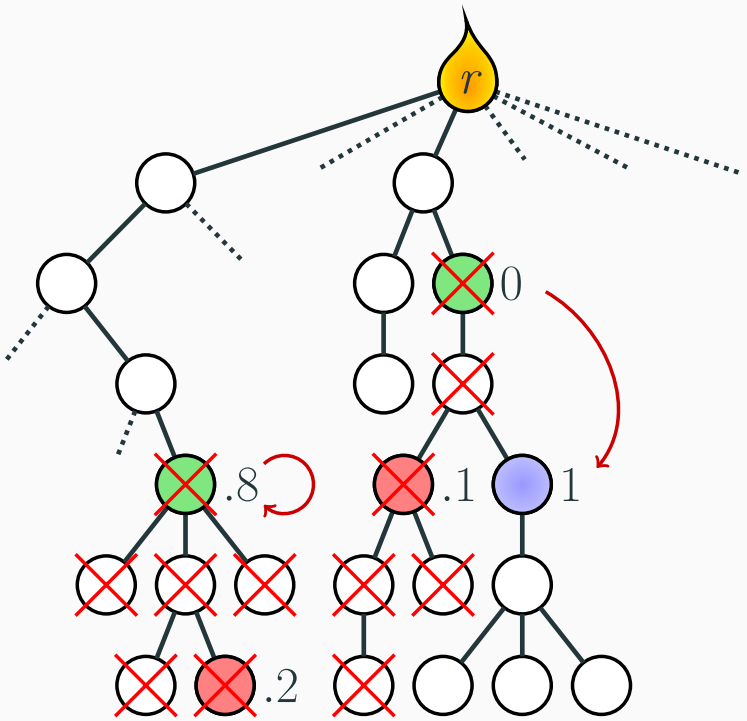
loose

tight

Plan: reallocate ● and protect 1-vertices

1. For each ●: { either do nothing, / or push fraction down to some ●.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

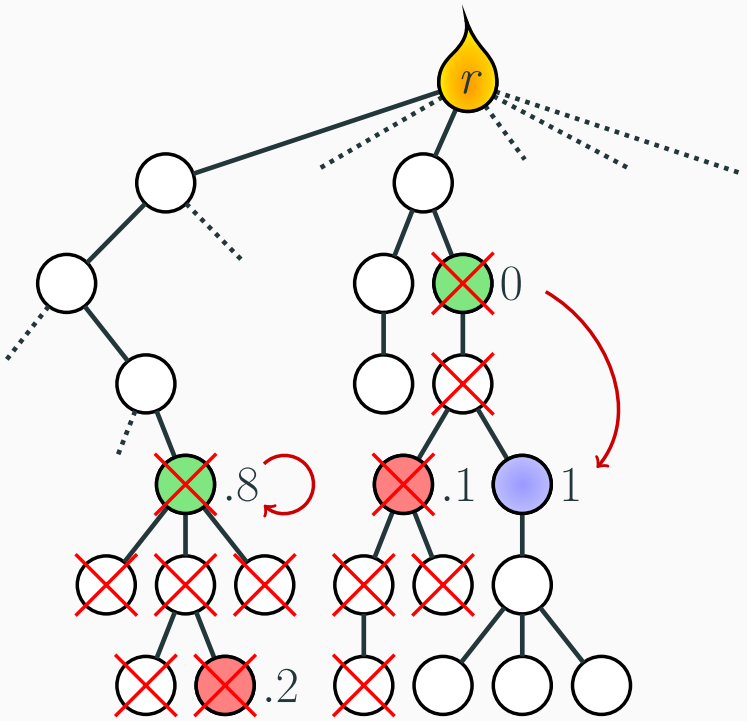
tight

loss

Plan: reallocate ● and protect 1-vertices

1. For each ●: { either do nothing, / or push fraction down to some ●.
2. Return 1-vertices.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

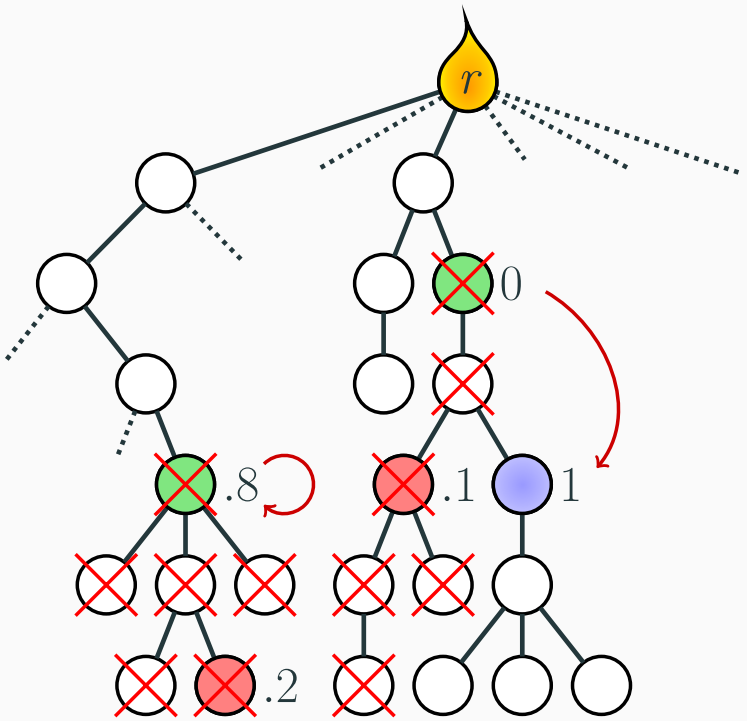
loss

How to enforce little loss from reallocation?

- Either ● covers little,
- or (● → ●) loses little.

Loss measure = number of × nodes.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

loss

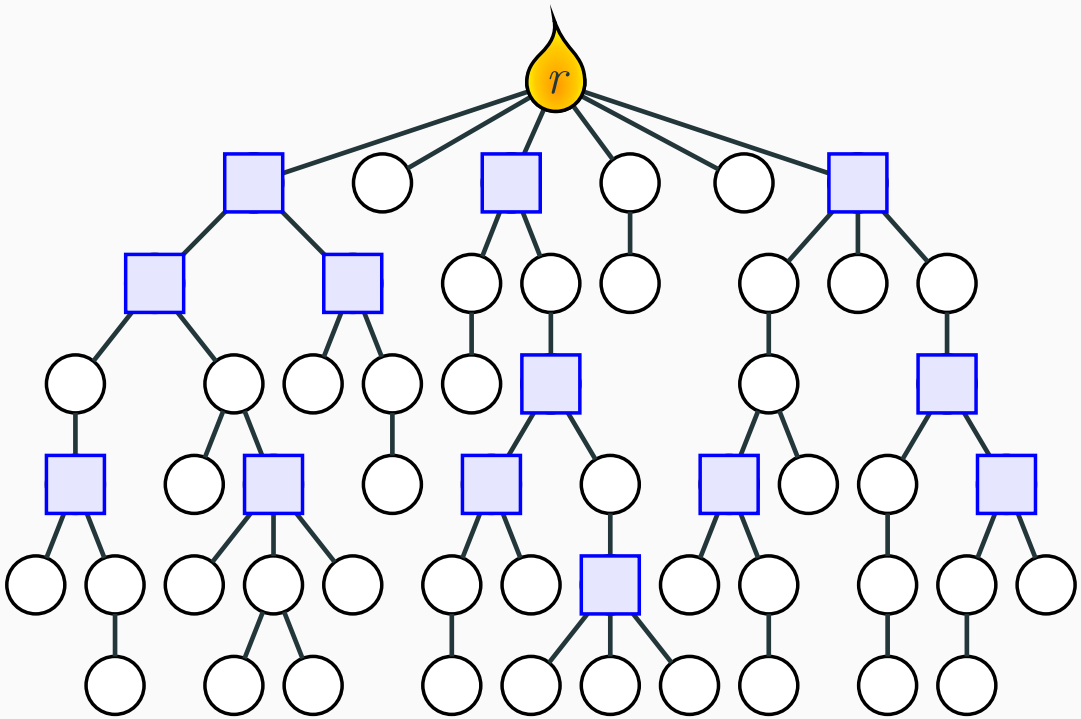
How to enforce little loss from reallocation?

- Either ● covers little,
- or (● → ●) loses little.

Loss measure = number of × nodes.

Frac. ● always covered by ● ⇒ × always covered by ●.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

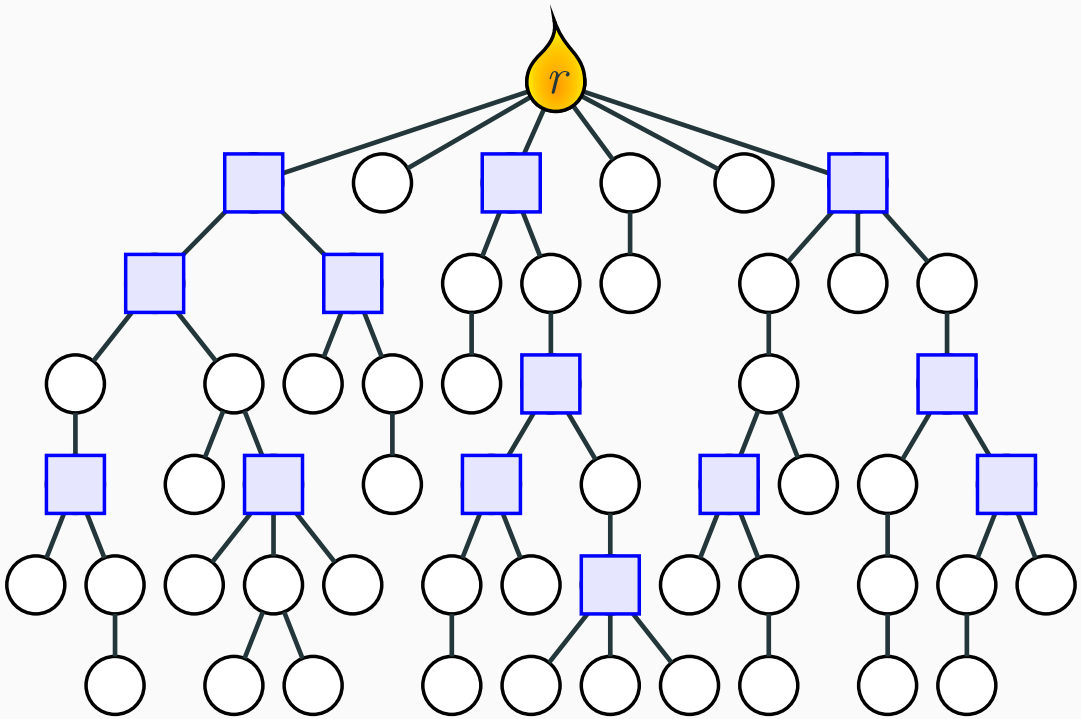
tight

loss

critical

Idea: Force ● and ● to appear only in delimited regions
by guessing a “critical” node set w.r.t. optimal solution.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

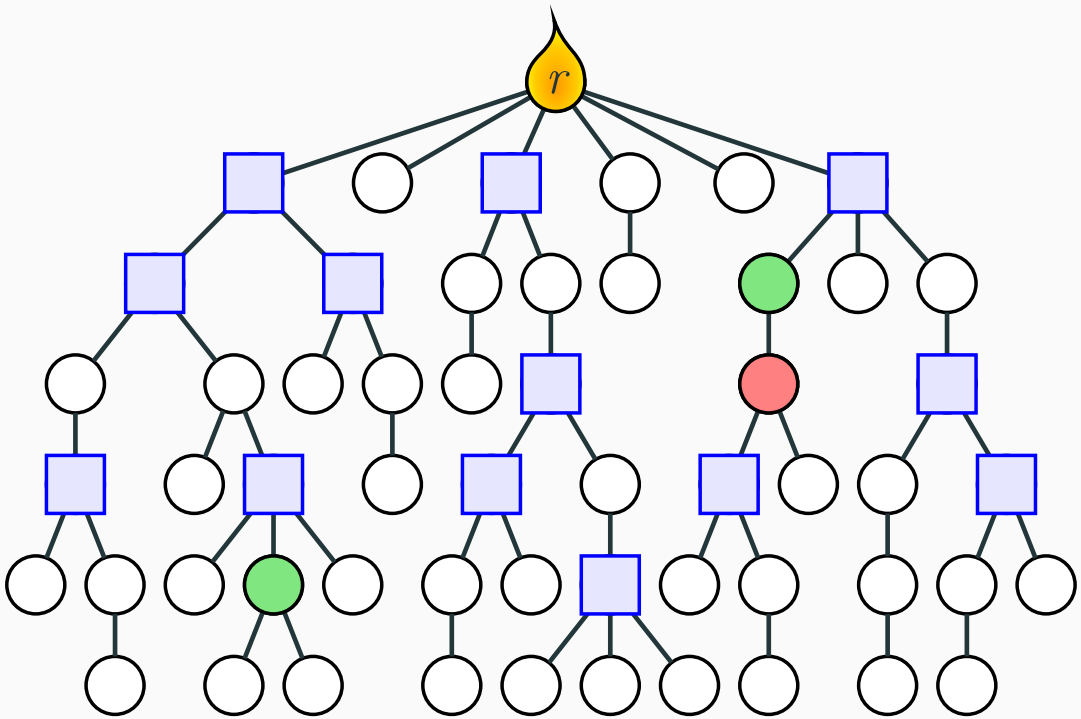
loss

critical

Guessing

For any u: $\begin{cases} \text{add } x(P_u) = 1 \text{ to } (LP) & \text{if } u \text{ saved by OPT,} \\ \text{add } x(P_u) = 0 \text{ to } (LP) & \text{if } u \text{ burning in OPT.} \end{cases}$

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

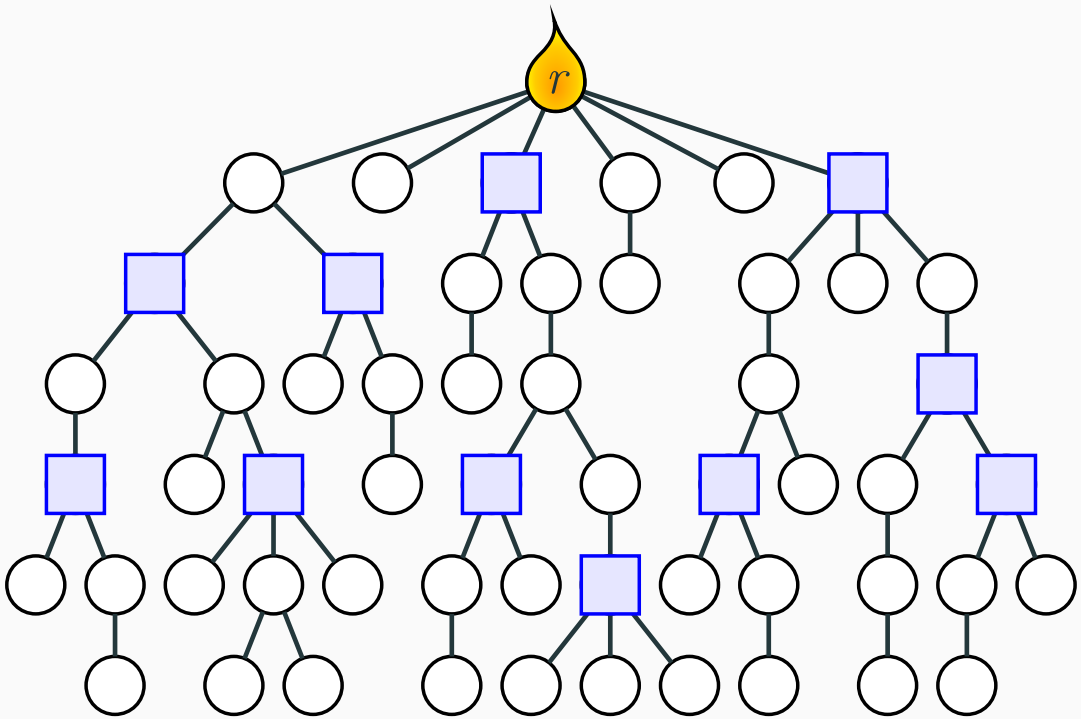
loss

critical

Solve (LP) updated with guessed contraints.

Observe:
- Either ● does not have □ below,
- or ● covers some ● of the same component.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

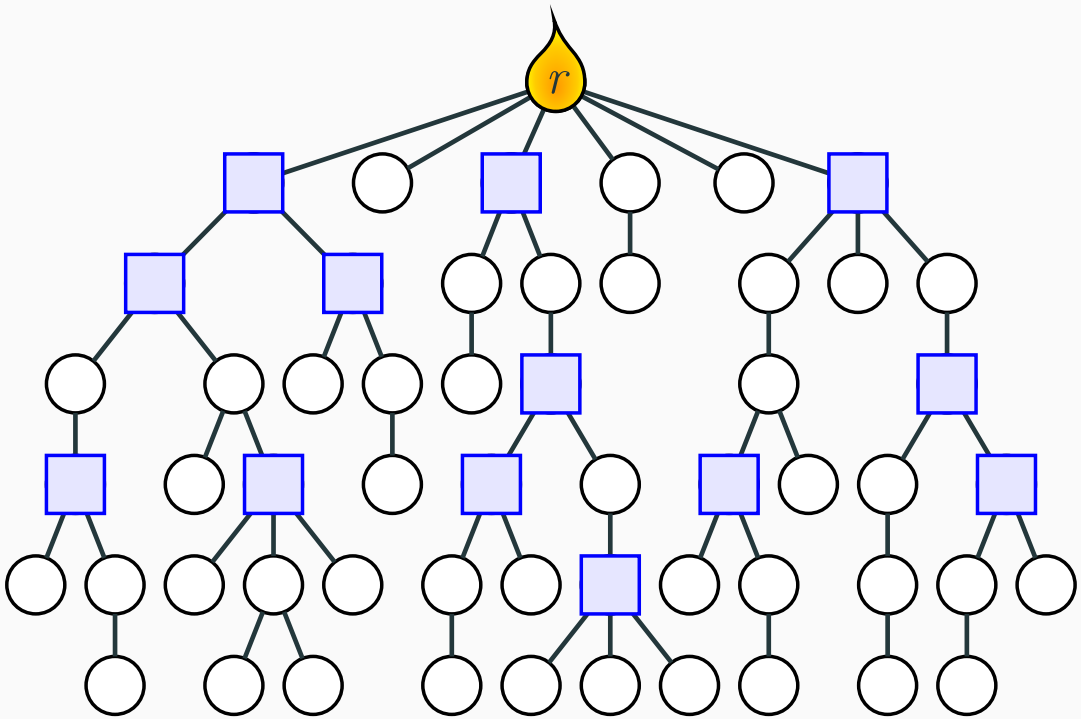
loss

critical

How to place □ as to contain loss?

- □ have to isolate small components.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

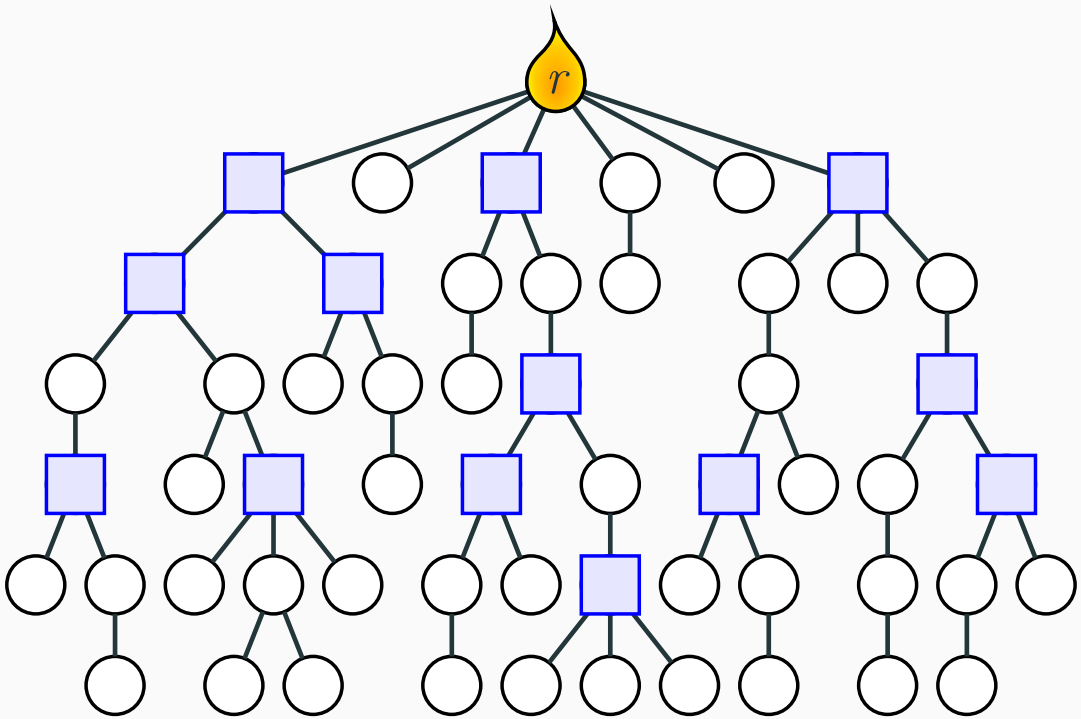
loss

critical

How to place □ as to contain loss?

- □ have to isolate small components.
- Any nearest common ancestor of □ has to be □.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

loss

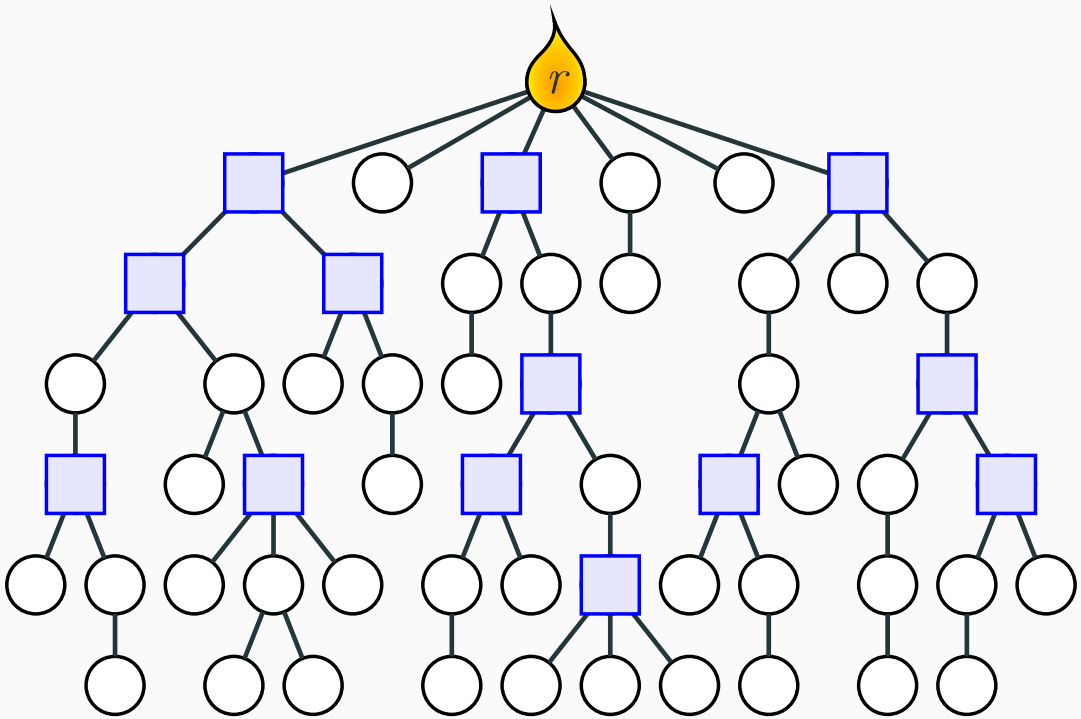
critical

How to place □ as to contain loss?

- □ have to isolate small components.
- Any nearest common ancestor of □ has to be □.

$\Rightarrow$ Loss by reallocation bounded by size of component.

Critical Nodes - Bounding Loss From Singular Reallocation



loose

tight

loss

critical

ROUNDING: reallocate ● and return 1-vertices

1. For each ●: { do nothing if no □ below, / push down to the ● covering □.
2. Return all 1-vertices.

On Choosing Critical Nodes

On Choosing Critical Nodes

Let k : Max loss for single reallocation.

$\Rightarrow$ $\square$ must decompose graph into components of size $\leq k$.

On Choosing Critical Nodes

Let k : Max loss for single reallocation.

$\Rightarrow$ $\square$ must decompose graph into components of size $\leq k$.

Recall:
- # of reallocations $= O(\log L)$.
- Target for max total loss $= \epsilon \cdot \mathrm{val}(\mathrm{OPT})$.

On Choosing Critical Nodes

Let k : Max loss for single reallocation.

$\Rightarrow$ $\square$ must decompose graph into components of size $\leq k$.

Recall:
- # of reallocations $= O(\log L)$.
- Target for max total loss $= \epsilon \cdot \mathrm{val}(\mathrm{OPT})$.

$$\text{We need } k \leq \frac{\epsilon \, \mathrm{val}(\mathrm{OPT})}{O(\log L)}.$$

On Choosing Critical Nodes

Let k : Max loss for single reallocation.

$\Rightarrow$ $\square$ must decompose graph into components of size $\leq k$.

Recall:
- # of reallocations $= O(\log L)$.
- Target for max total loss $= \epsilon \cdot \mathrm{val}(\mathrm{OPT})$.

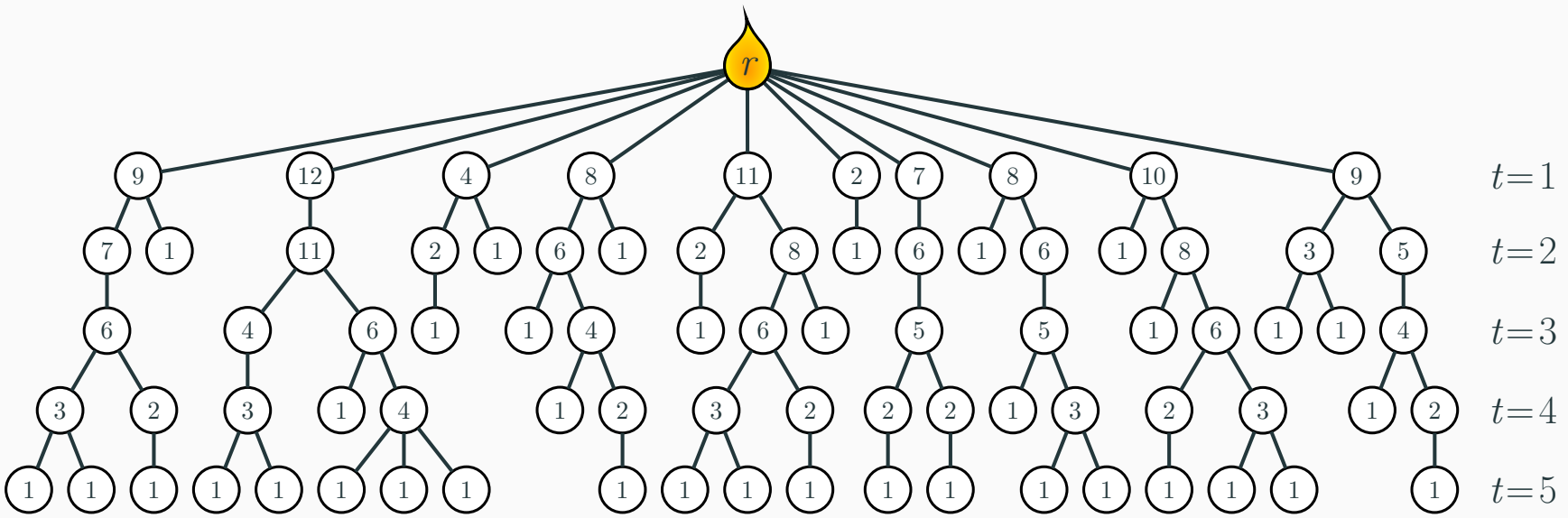
$$\text{We need } k \leq \frac{\epsilon\, \mathrm{val}(\mathrm{OPT})}{O(\log L)}.$$

Number of $\square$ needed is $O\left(\frac{|V|}{k}\right) = O\left(\frac{|V| \log L}{\epsilon\, \mathrm{val}(\mathrm{OPT})}\right)$.

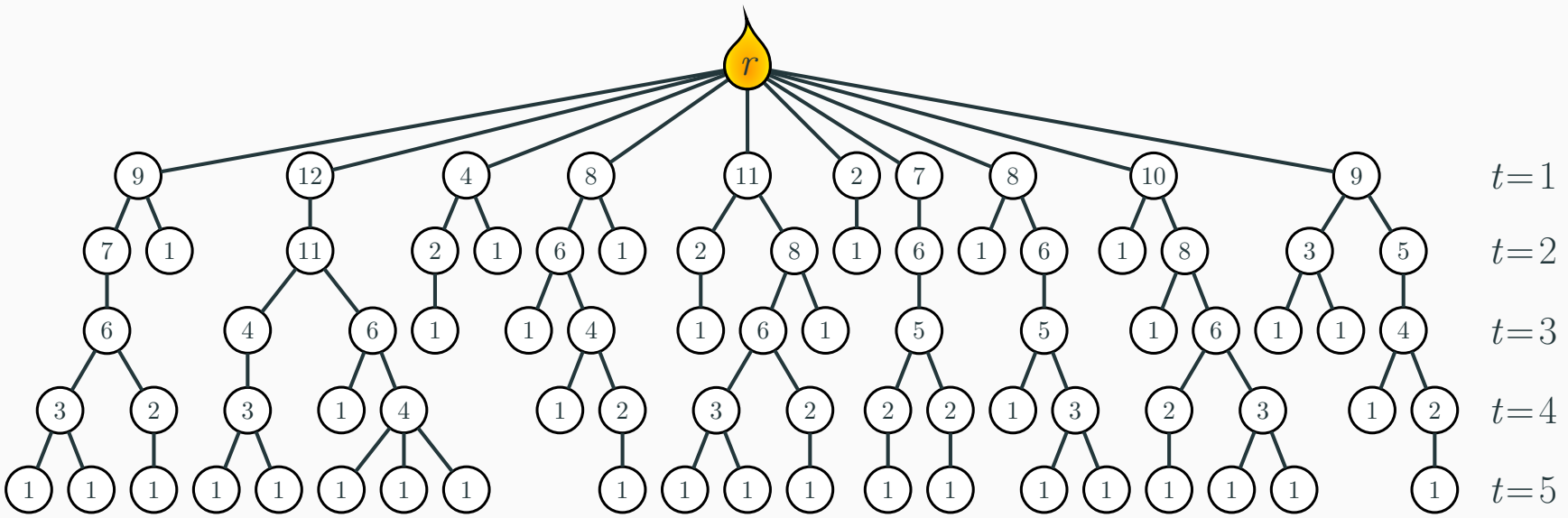
However, guessing costs $= 2^{|\square|}$.

For poly-time enumeration, we need $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$!

λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$



λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$

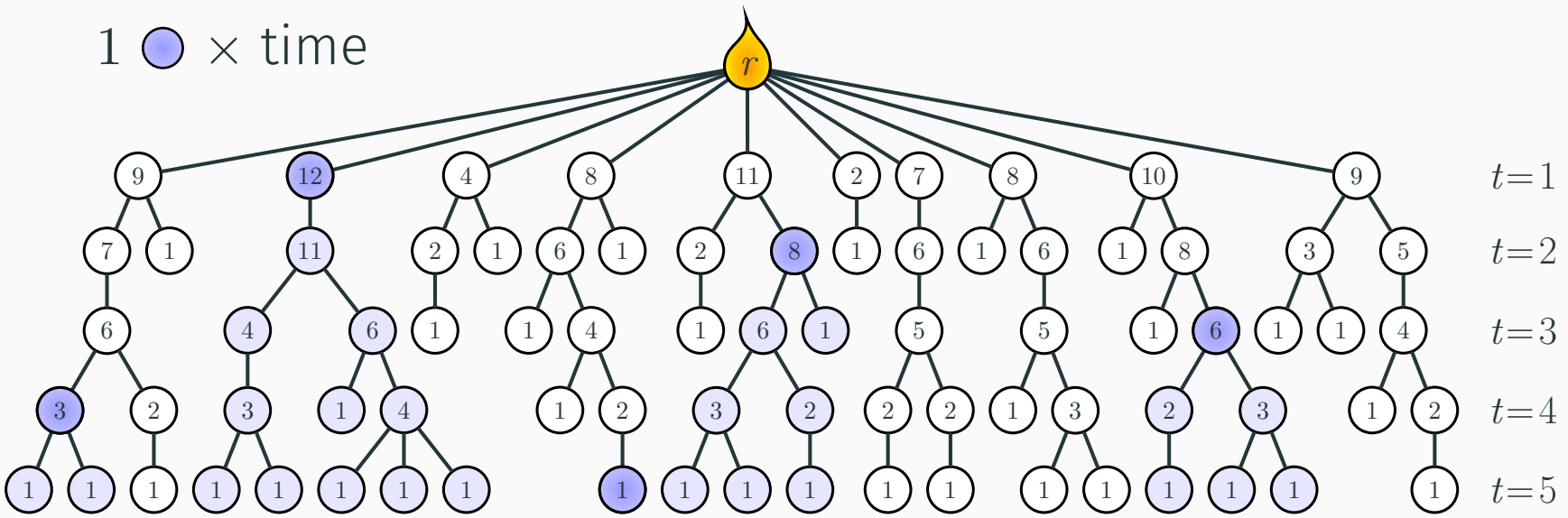


Greedy Algorithm

From the first to the last level,

place ● at the heaviest remaining subtree.

λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$

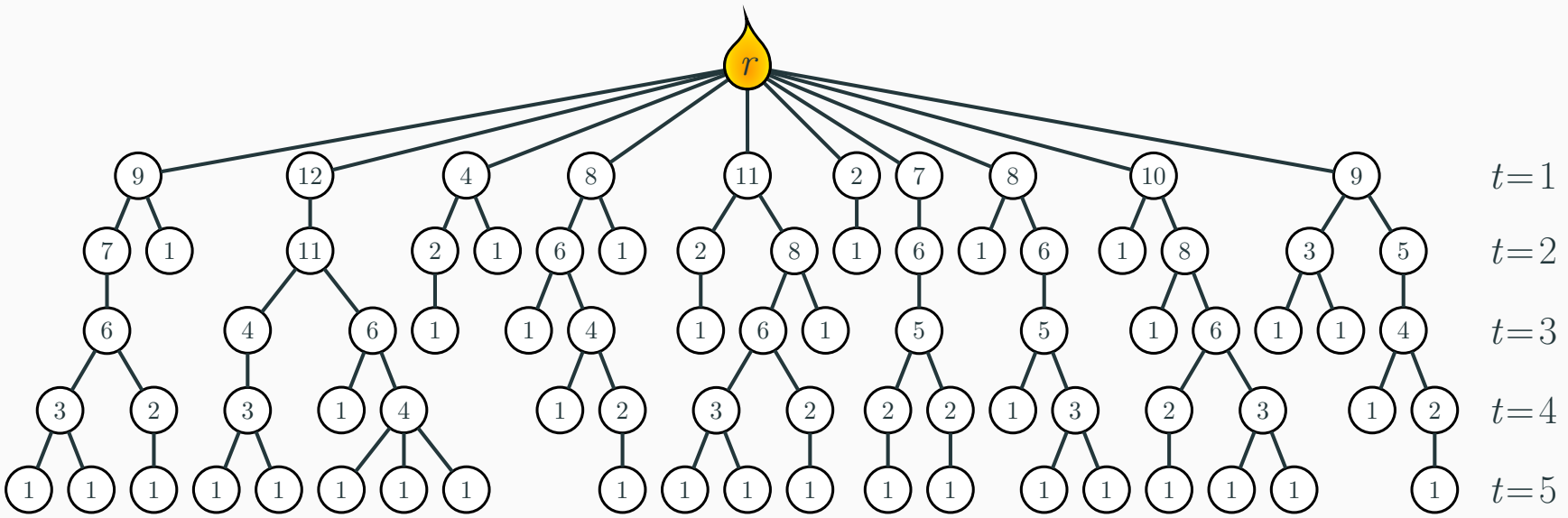


Greedy Algorithm

From the first to the last level,

place ● at the heaviest remaining subtree.

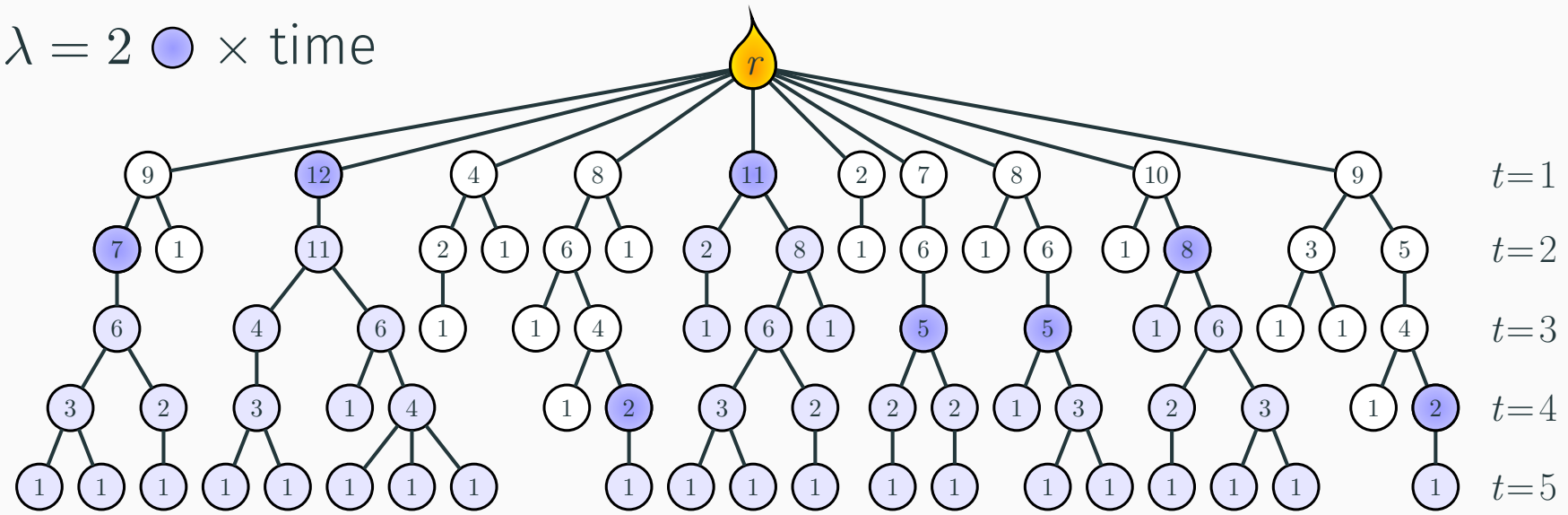
λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$



Greedy Pruning

1. From the first to the last level,
   place λ ● at the heaviest remaining subtrees.

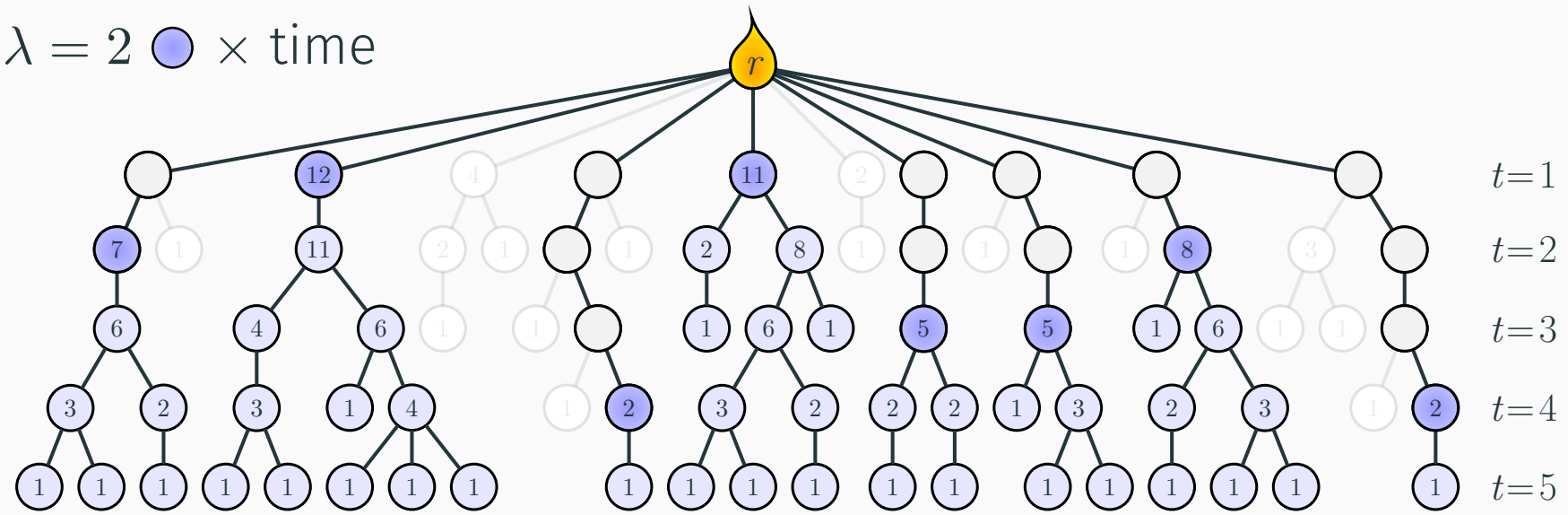
λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$



Greedy Pruning

1. From the first to the last level,
   place λ ● at the heaviest remaining subtrees.

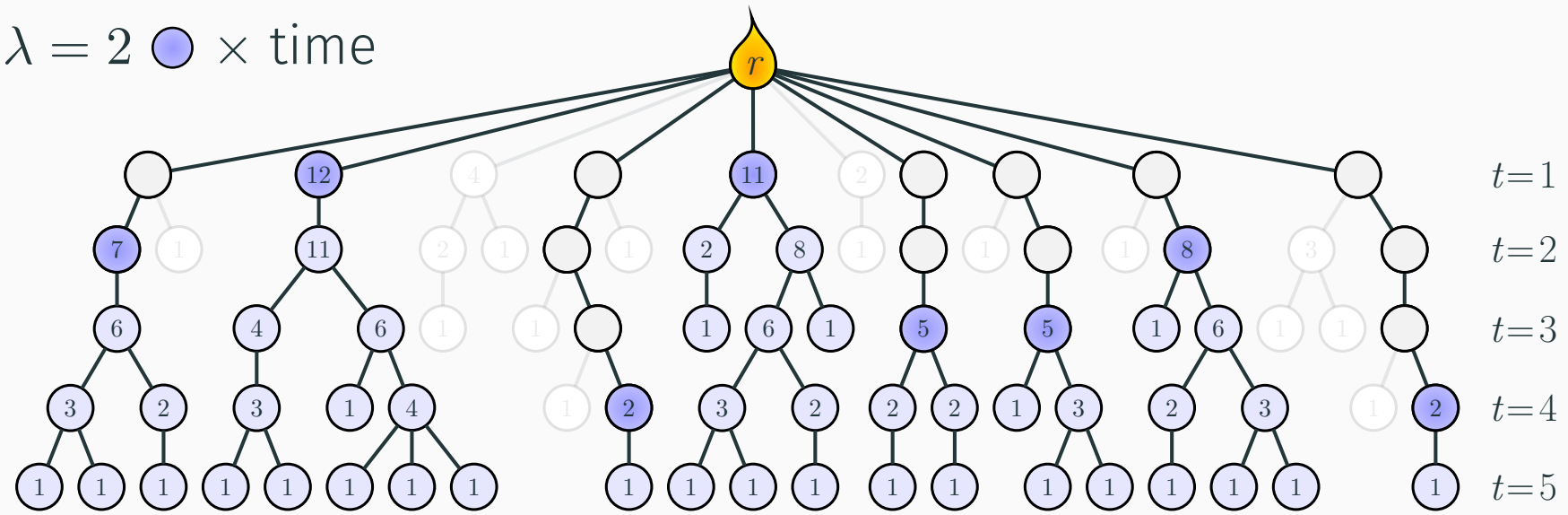
λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$



Greedy Pruning

1. From the first to the last level,
   place λ ● at the heaviest remaining subtrees.
2. Remove all the nodes that are not saved
   but leave the path from ● to r.

λ-Greedy Pruning: Preprocessing to get $\mathrm{val}(\mathrm{OPT}) = \Omega(|V|)$



Lemma

- Optimal value in pruned tree $\geq 1/\lambda$ size of pruned tree.
- Optimal value in pruned tree $\geq$
  $\geq (1 - 1/\lambda)$ optimal value in original tree.

Algorithm - PTAS for FFP

INPUT: Tree of size n, $\epsilon > 0$.

DO:
1. Apply δ-compression.
2. Apply λ-greedy pruning (for well-chosen $\lambda = O(1)$).
3. Determine k-critical nodes.
4. Guess which k-critical nodes to save.
5. Solve $(LP) \rightarrow$ determine 🟢 and 🔴.
6. Reallocate 🟢.

RETURN: 1-vertices.

Conclusions

Conclusions

Theorem

PTAS for the FireFighter Problem on trees.

Conclusions

Theorem

PTAS for the FireFighter Problem on trees.

Theorem

$O(1)$-approximation for RMFC on trees.

We use:

- Variant of δ-compression.
- LP-guided recursive enumeration algorithm to guess super-constant set of good vertices to protect.

OPEN: 2-approximation for RMFC.
(We have a quasi-polynomial 2-approximation.)