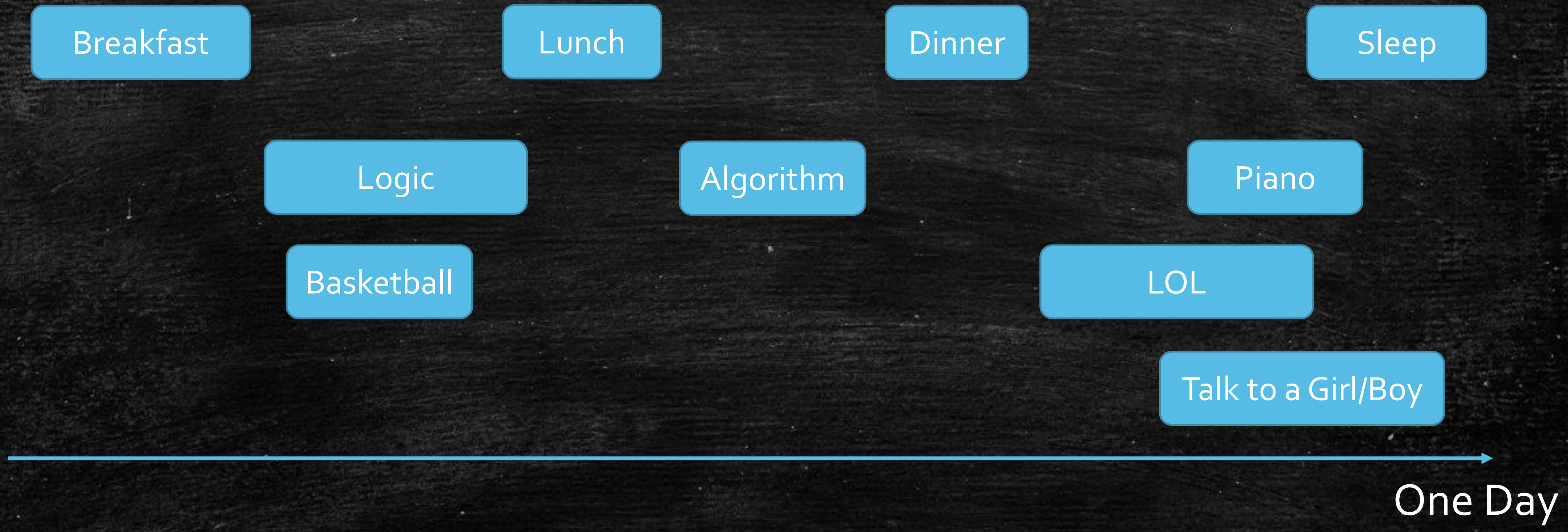
More Greedy Algorithms

Greedy Homework Scheduling

- **Input:** n homework, each homework j has a size s_j, and a deadline d_j.
- **Output:** output a time schedule of doing homework!
- Greedy Approach
  - Keep finishing the homework with the **closest** deadline
- We have prove it is optimal!

Let generalize the problem!



Let generalize the problem!

Each Activity has a fixed time schedule

How to do more activities a day?

Breakfast

Lunch

Dinner

Sleep

Logic

Algorithm

Piano

Basketball

LOL

Each Activity has a fixed time schedule

Talk to a Girl/Boy

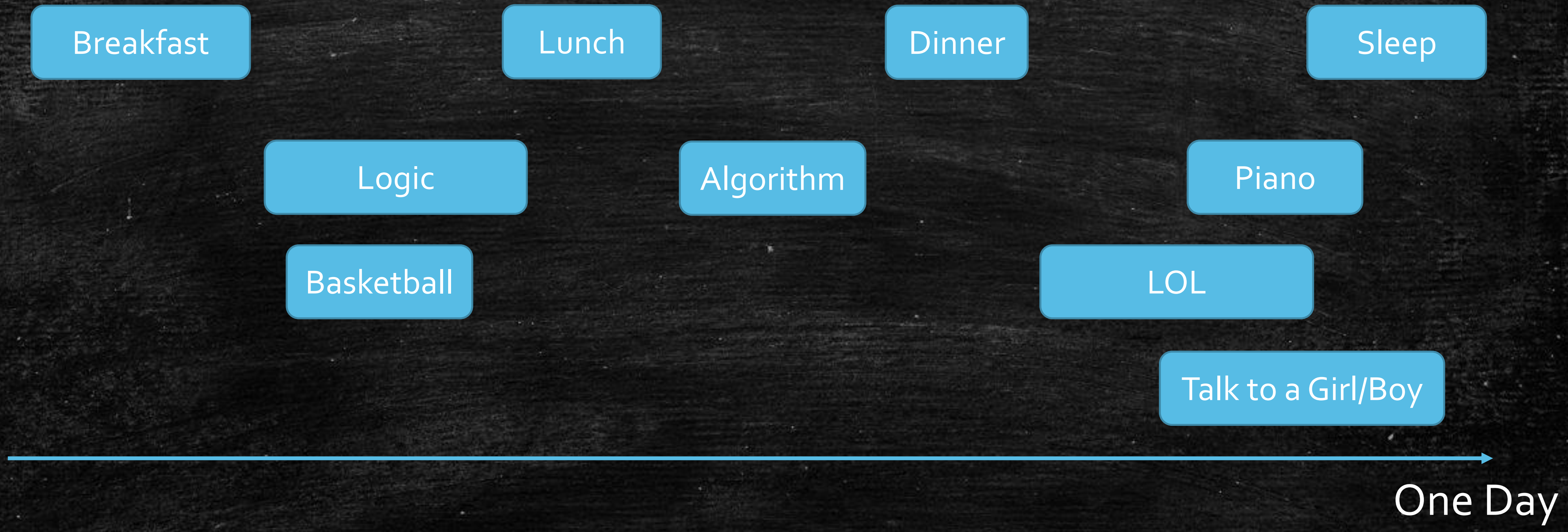
One Day

Three Greedy Ideas

A: Start Time First

B: Shortest Length First

C: Finish Time First



Which one is correct?

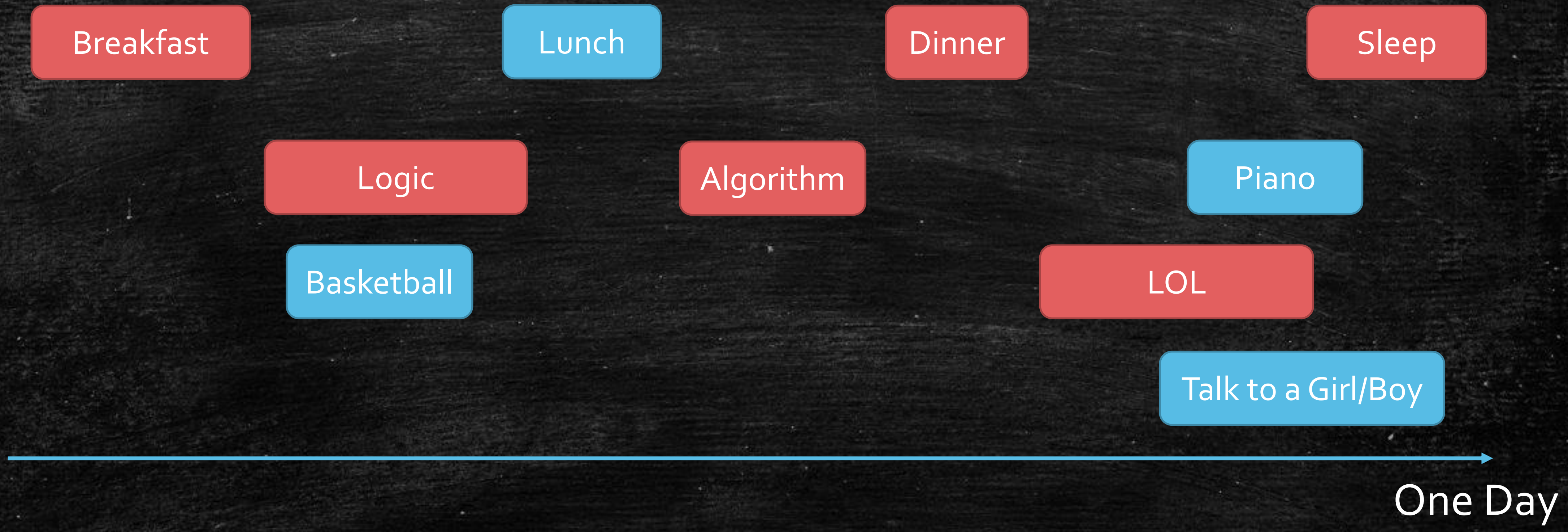
Three Greedy Ideas

A: Start Time First

Six!

B: Shortest Length First

C: Finish Time First



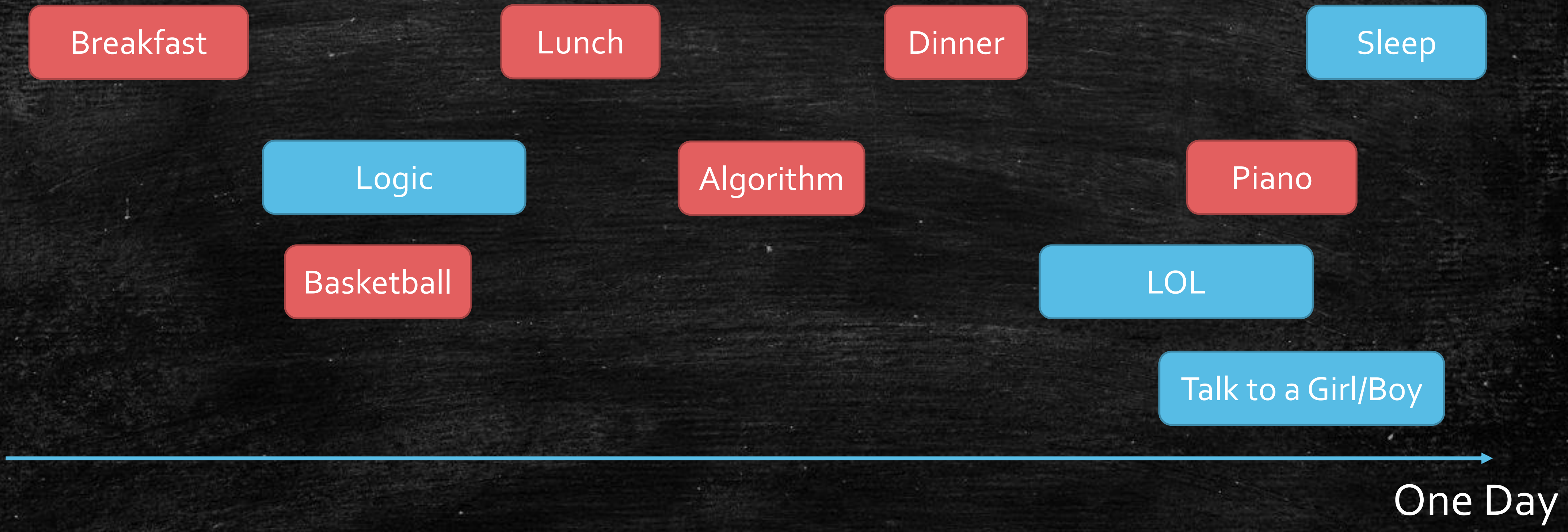
Three Greedy Ideas

A: Start Time First

B: Shortest Length First

Six!

C: Finish Time First



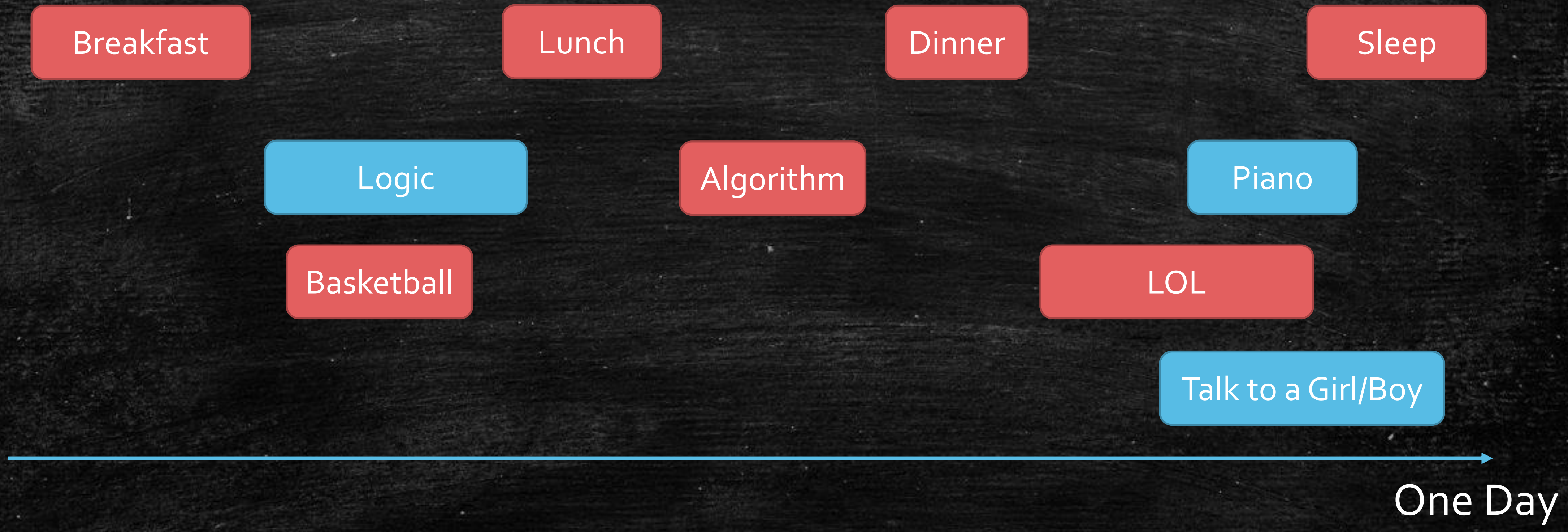
Three Greedy Ideas

A: Start Time First

B: Shortest Length First

C: Finish Time First

Seven!



Three Greedy Ideas

- Finish Time First is the only possible one!
- Is it correct?
- Intuition
  - Finish fast → Best for future
  - How to make a proof?

Recall

- Dijkstra
  - Grow from small SPT to larger SPT
- Prime & Kruskal
  - Grow from small P-MST to larger P-MST
- We are correct if we **never ruin out** OPT!
- Or say: we are still in an Optimal Tunnel!

The Big Idea

The local greedy choice do not ruin out OPT

Induction

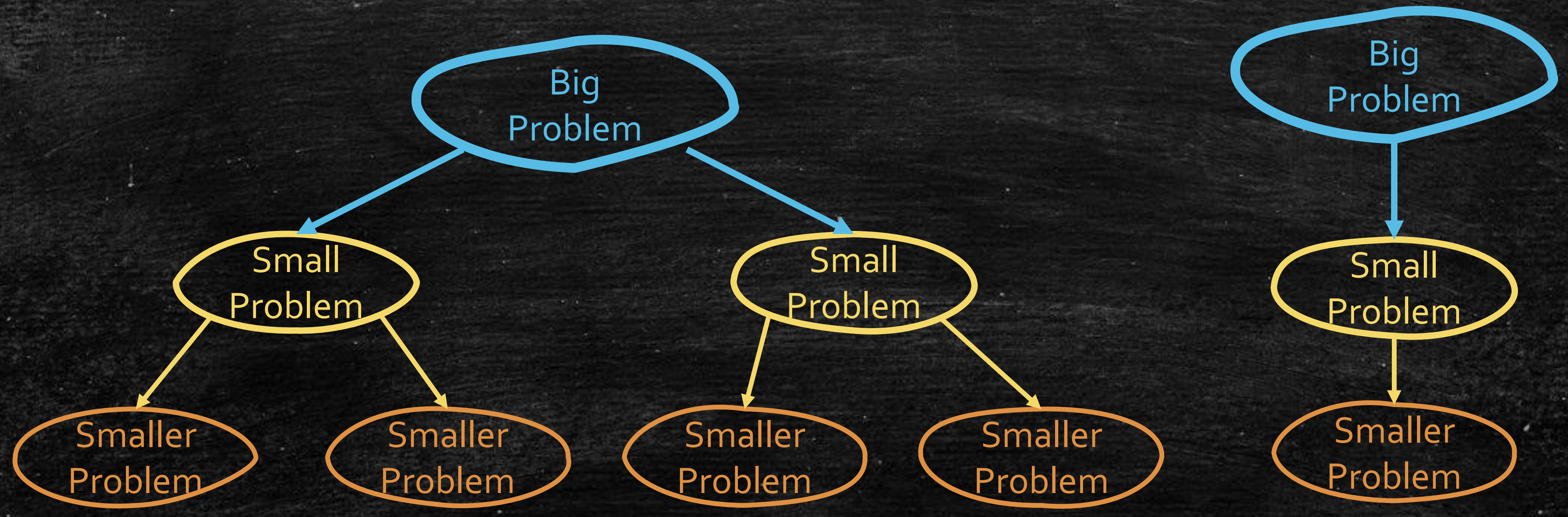
- Base step: $\emptyset$ is in an OPT.
- Assumptions: the selected $k - 1$ activities are in an OPT.
- Induction: After adding the k-th activity, we are still in an OPT.
- Conclusion: After adding the last activity, it is in an OPT. Nothing can be added, so it is OPT.

Proof of the Induction

- Assumptions: the selected $k-1$ activities are in an OPT.
- Induction: After adding the k-th activity, we are still in an OPT.
- Can you prove it?
- Discussion!

Summarize

Divide and Conquer vs. Greedy



One more interesting Greedy!

General Question

- How to encode a book?
- Two steps:
  - Give alphabet encoding policy
  - Encode all sentences in the book

i am good at algorithms

Alphabet: Naïve Approach

a	0000
d	0001
g	0010
h	0011
i	0100
l	0101
m	0110
o	0111
r	1000
s	1001
t	1010
space	1011

- **i am good at algorithms**
- Cost Analysis
  - Each character & space: 4 digit
  - Totally: $23 \times 4 = 92$

Improvement: Why not shorter?

a	0
d	1
g	10
h	11
i	100
l	101
m	110
o	111
r	1000
s	1001
t	1010
space	1011

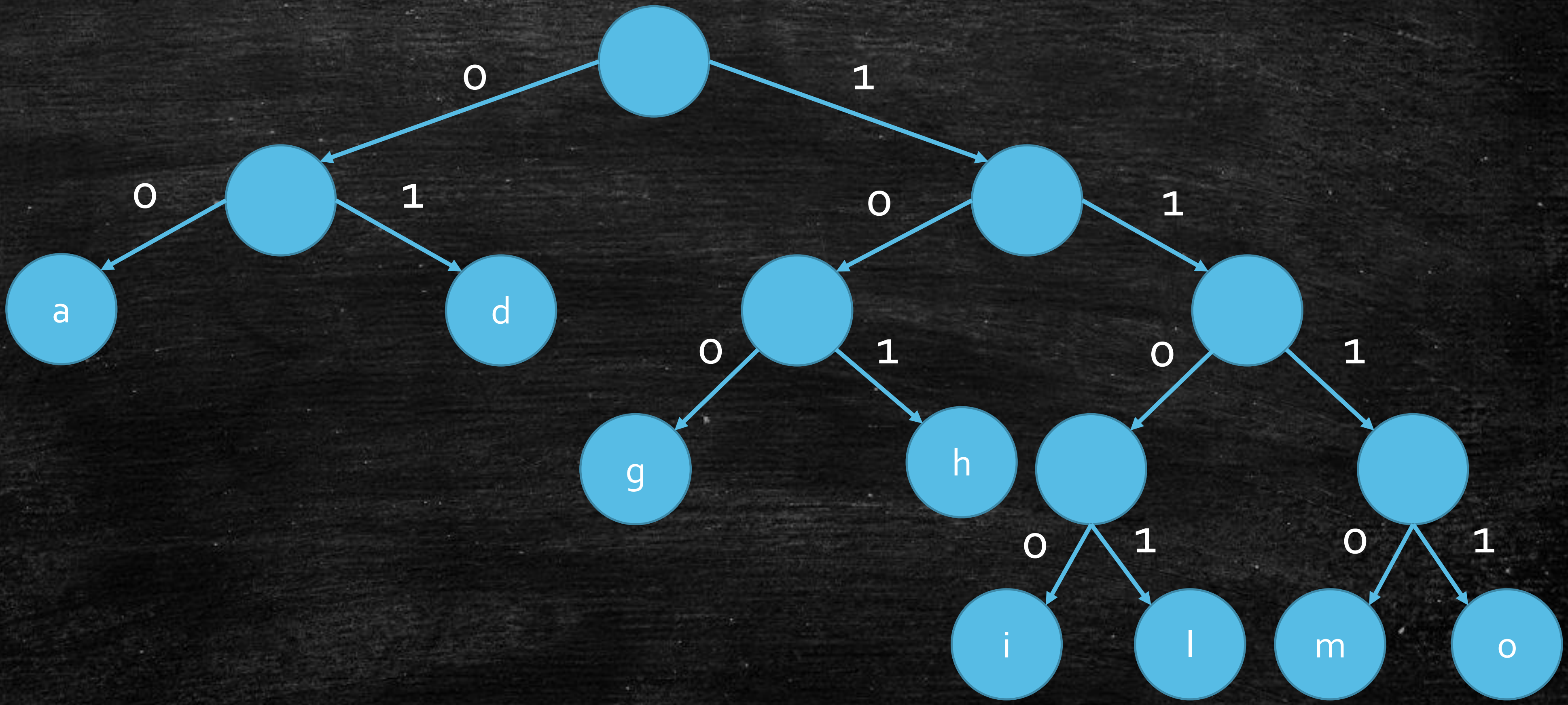
- **i am good at algorithms**
- Cost Analysis
  - Each character & space < 4 digit
  - Totally: $< 23 \times 4 = 92$
- Problem:
- When we decode
  - 10: is it 'g' or 'da' ?

Solving the problem!

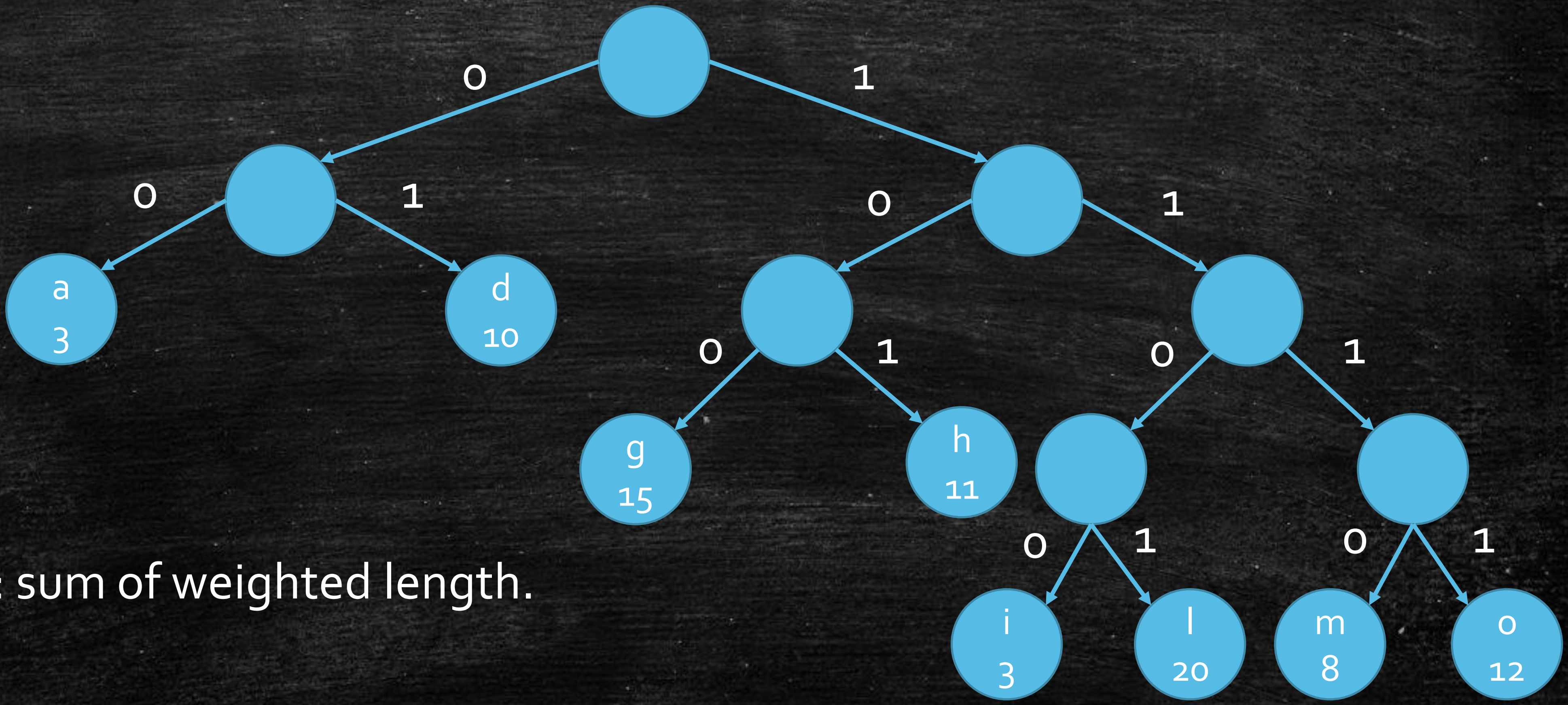
a	0
d	1
g	10
h	11
i	100
l	101
m	110
o	111
r	1000
s	1001
t	1010
space	1011

- **i am good at algorithms**
- Problem:
- When we decode
  - 10: is it 'g' or 'da' ?
- A **prefix-free** code
  - No one's code is the prefix of another one's code.
  - Example: 'd': 1 is the prefix of 'g': 10.
  - Think why it is good?
  - How to decode?

A prefix-free code is a tree!

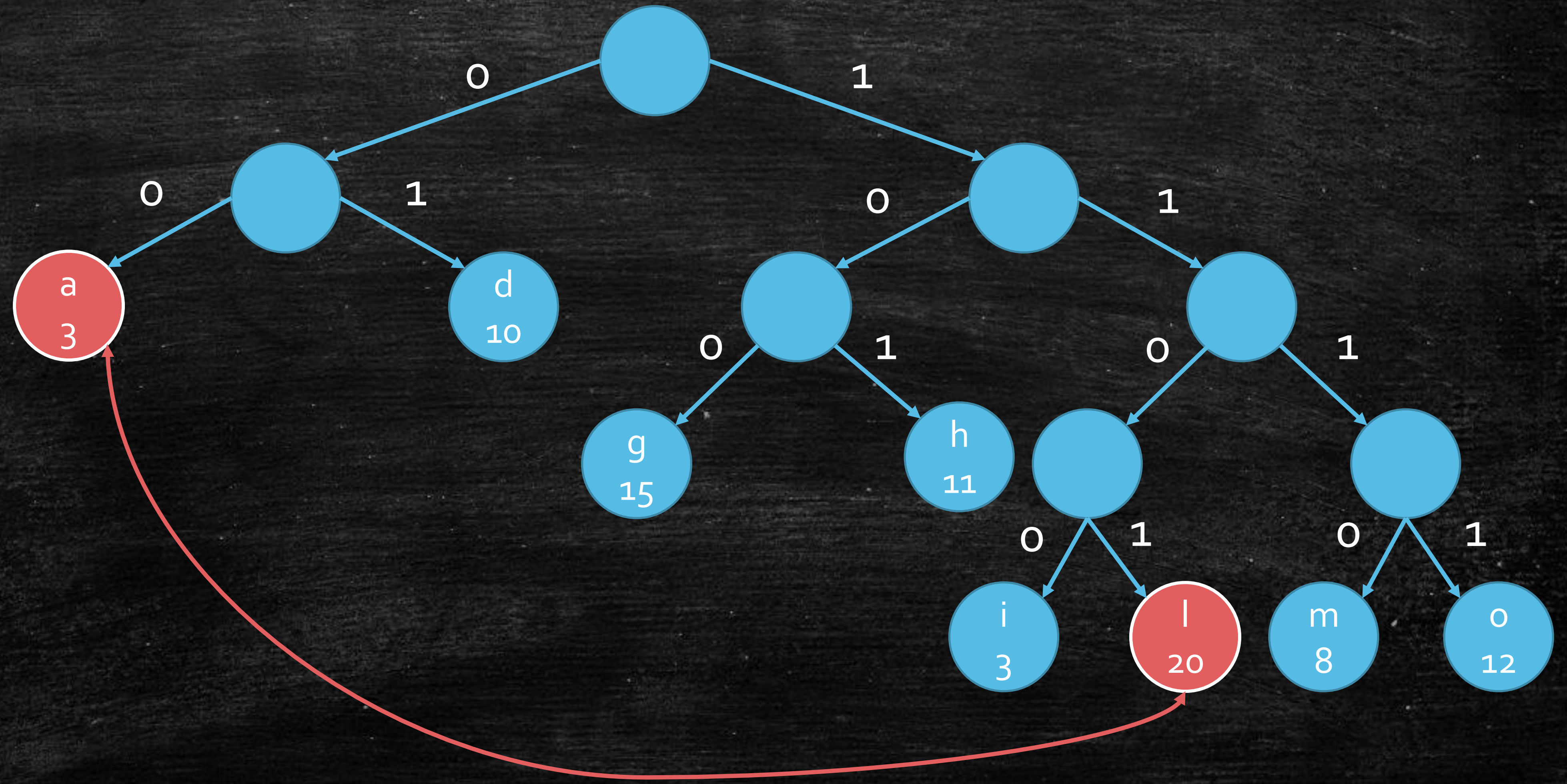


Cost of A prefix-free code is a tree!



Cost: sum of weighted length.

Minimize the cost?



What is the greedy approach now?

- Build a tree from bottom.
- Put small cost character to bottom.

A bottom-up building

a 3

d 10

g 15

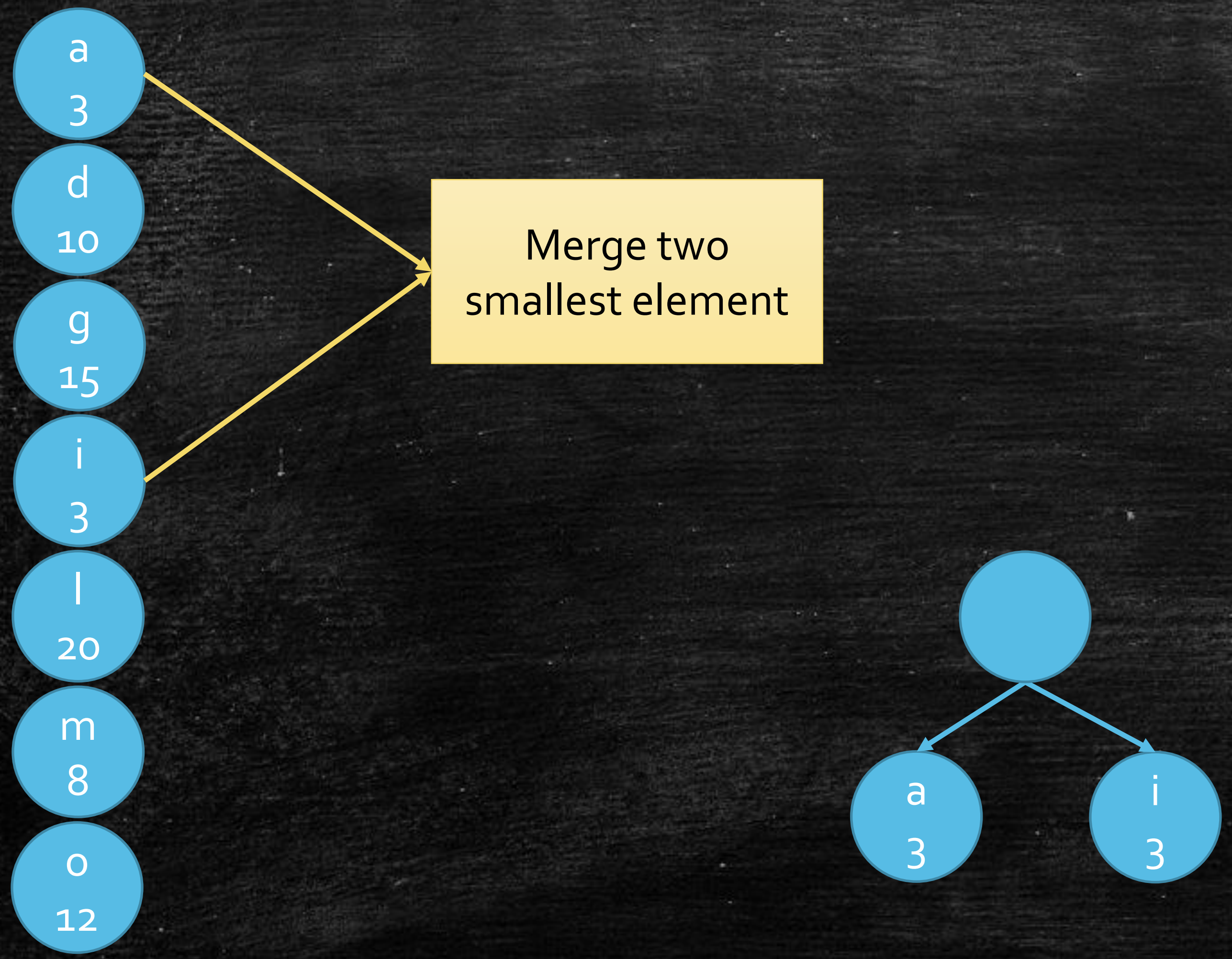
i 3

l 20

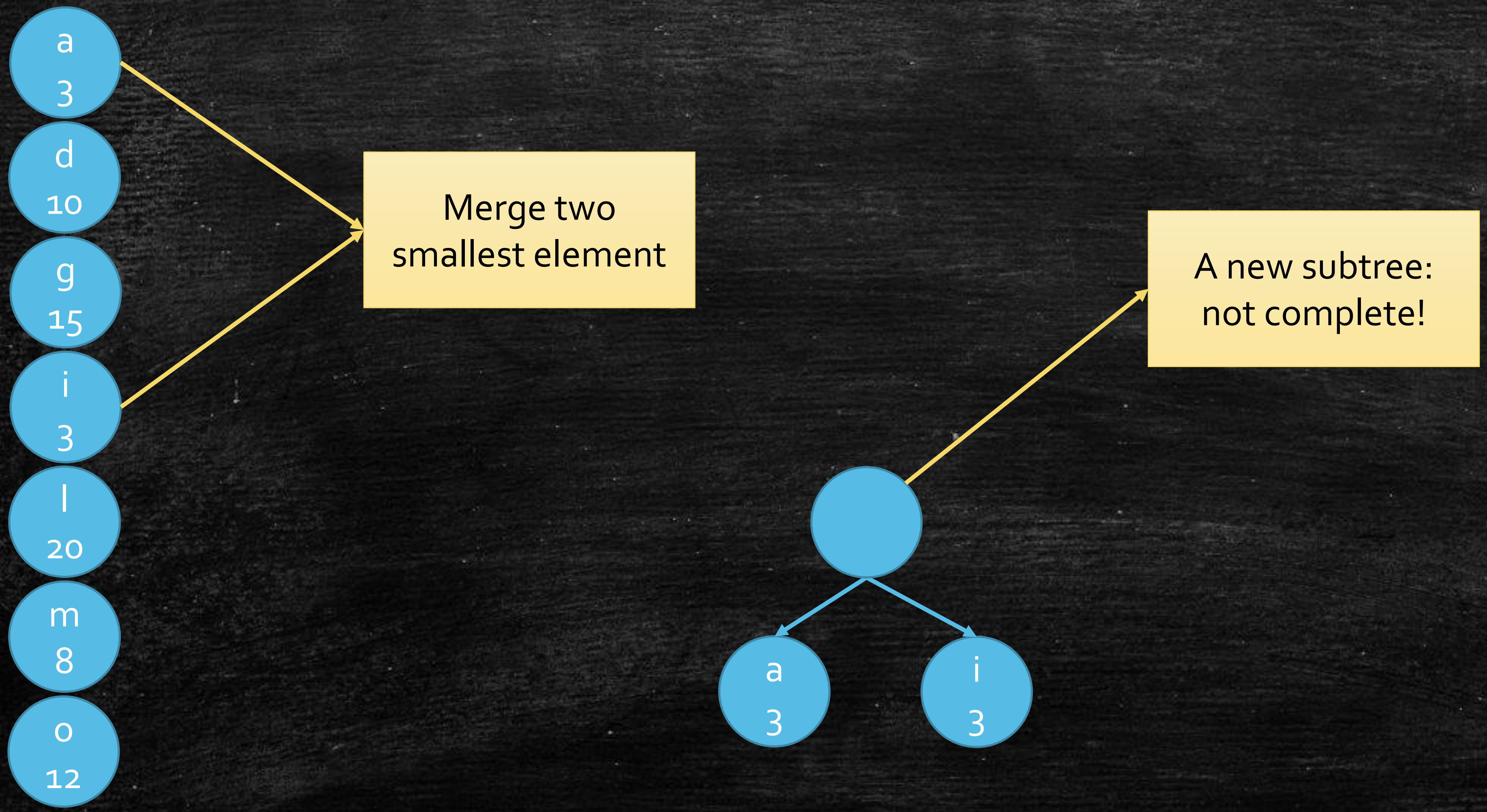
m 8

o 12

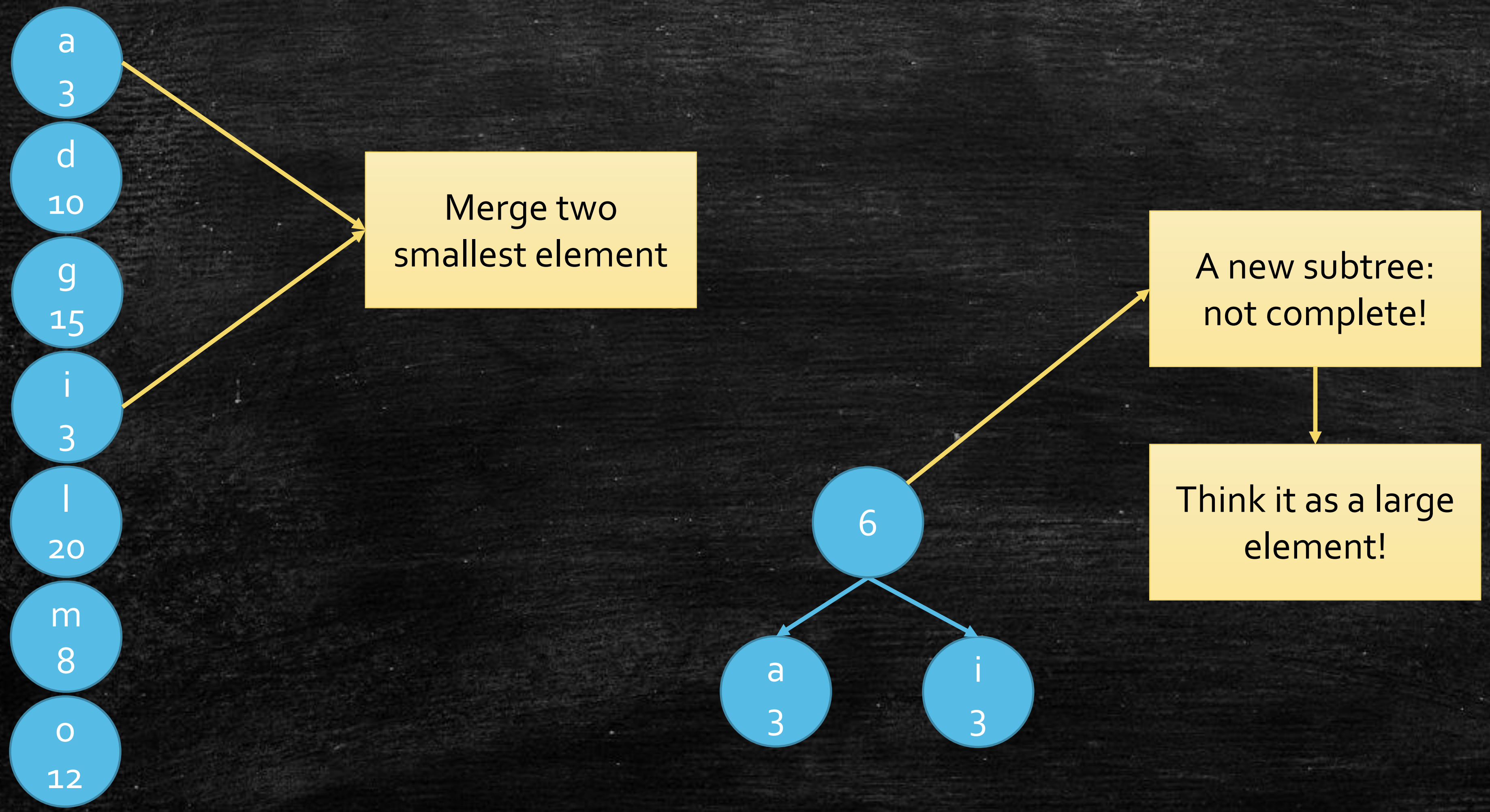
A bottom-up building



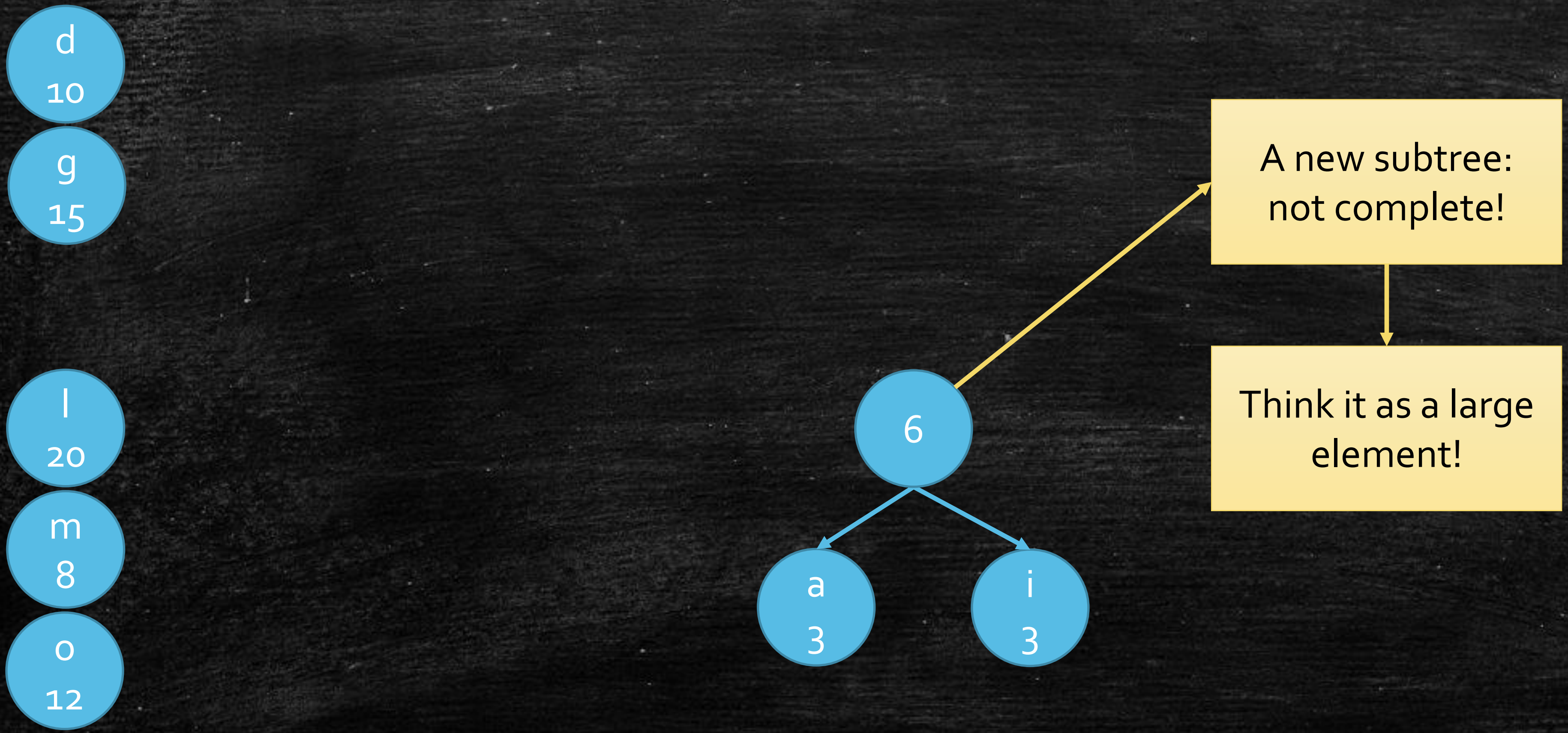
A bottom-up building



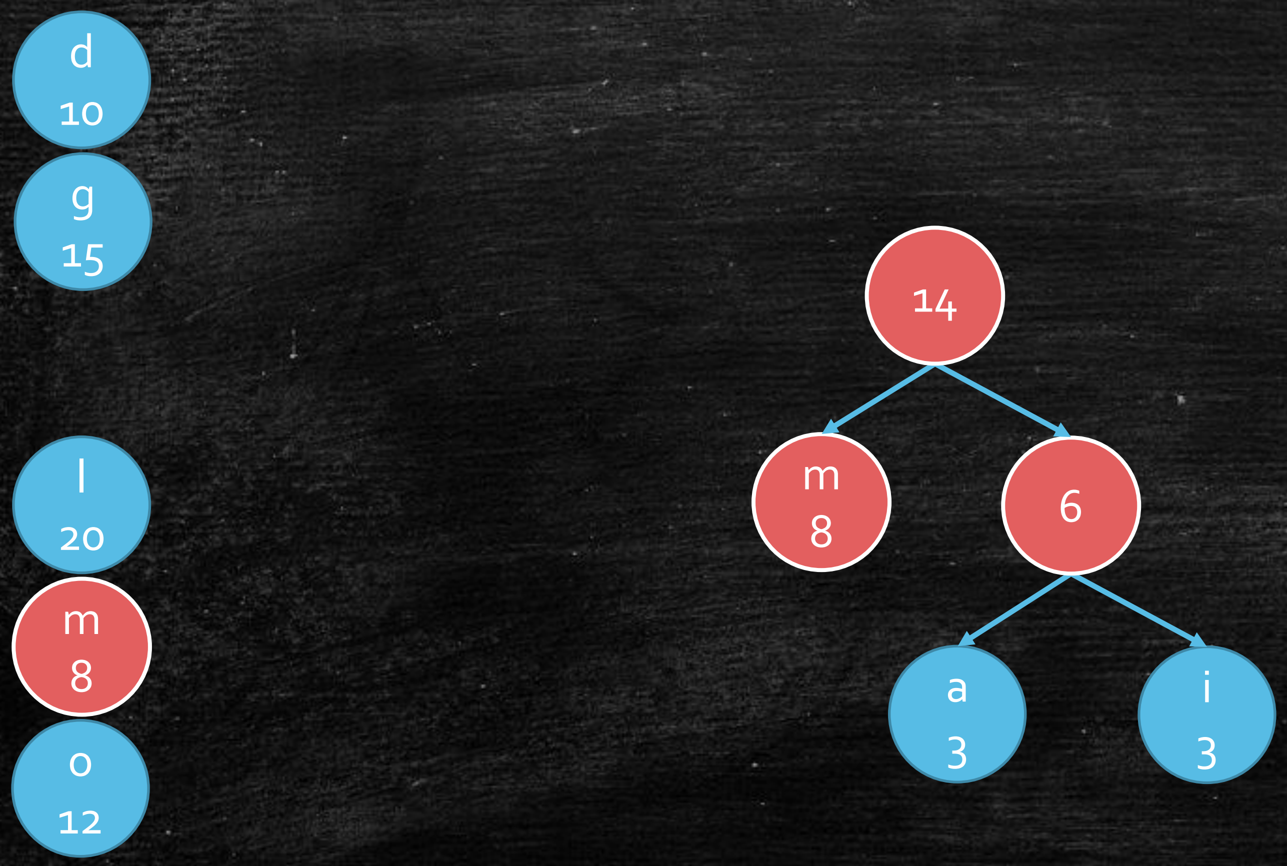
A bottom-up building



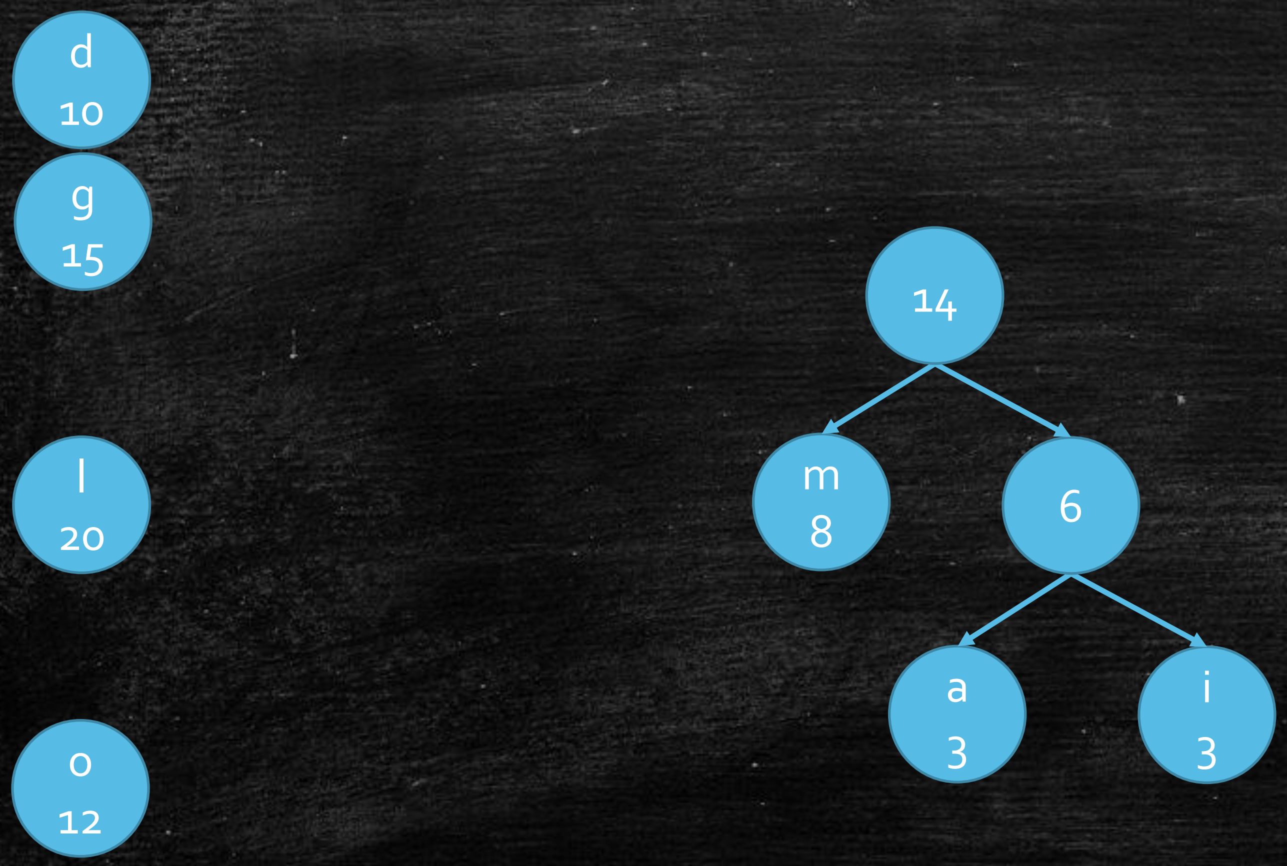
A bottom-up building



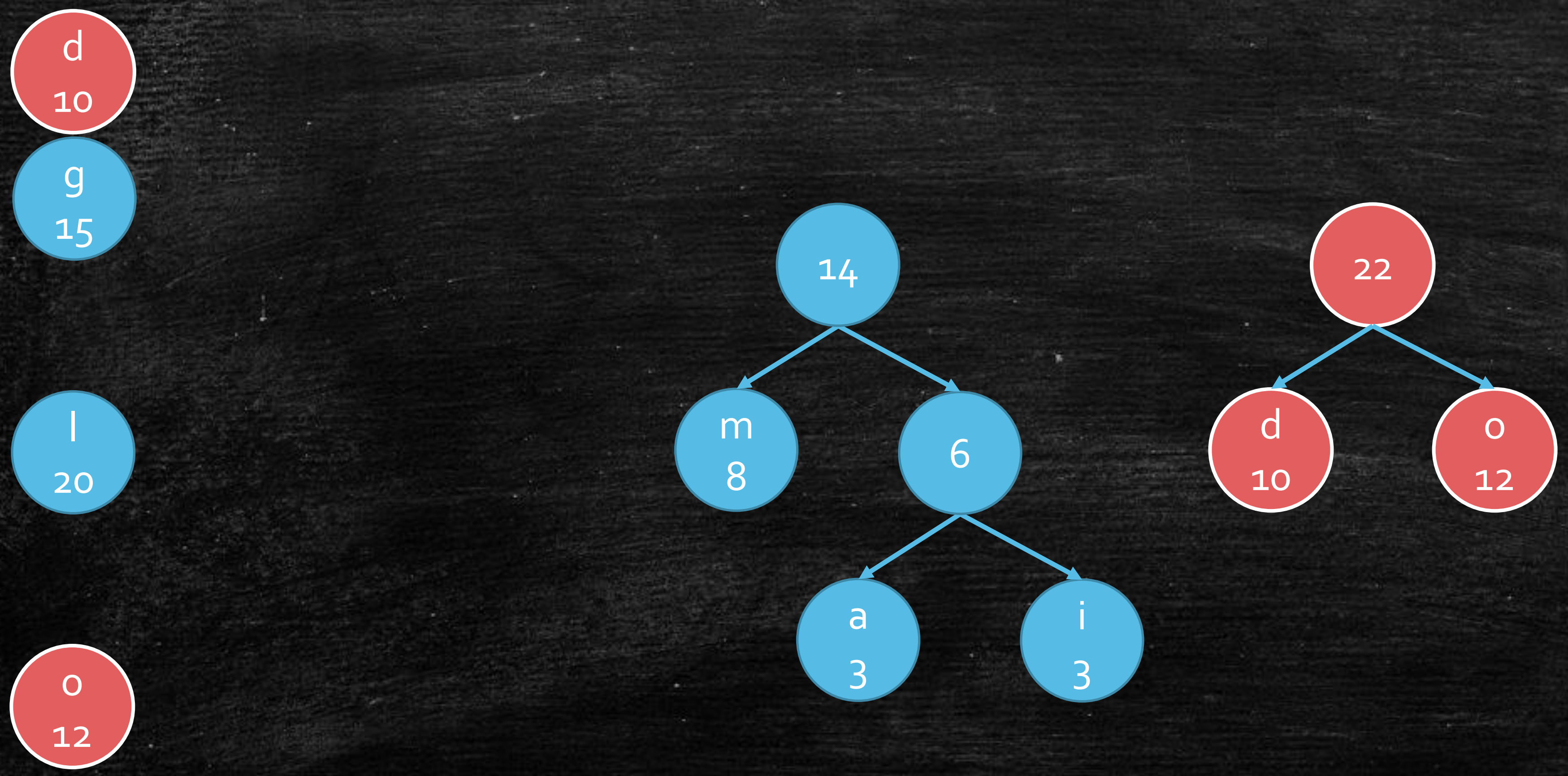
A bottom-up building



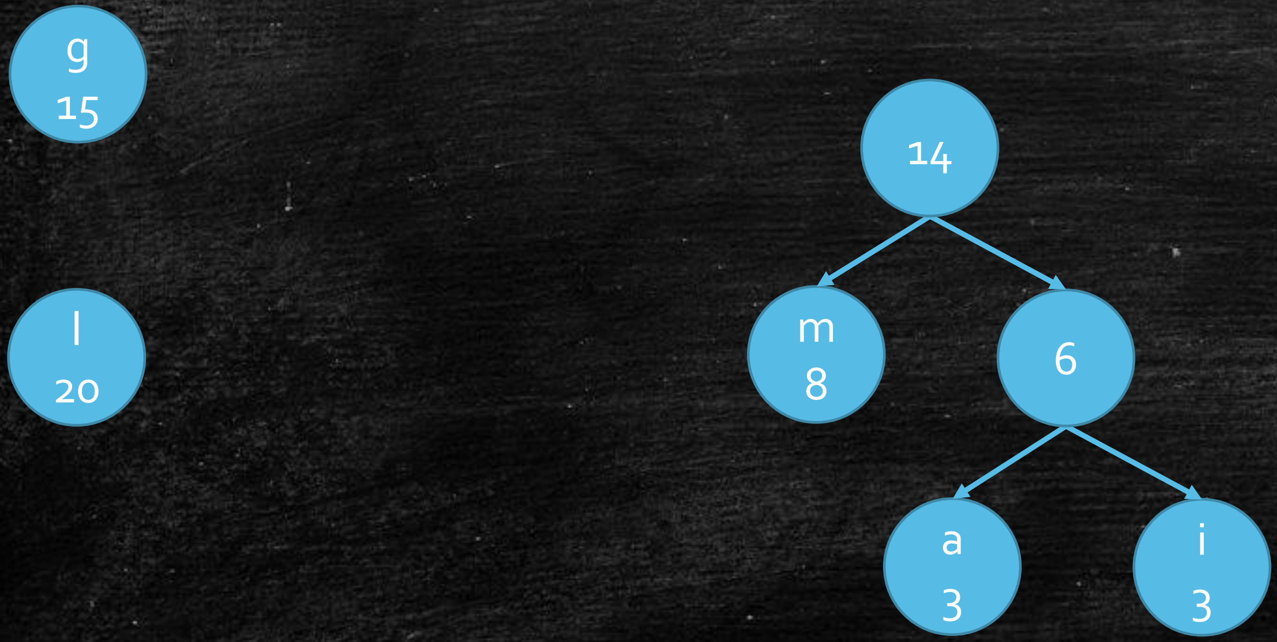
A bottom-up building



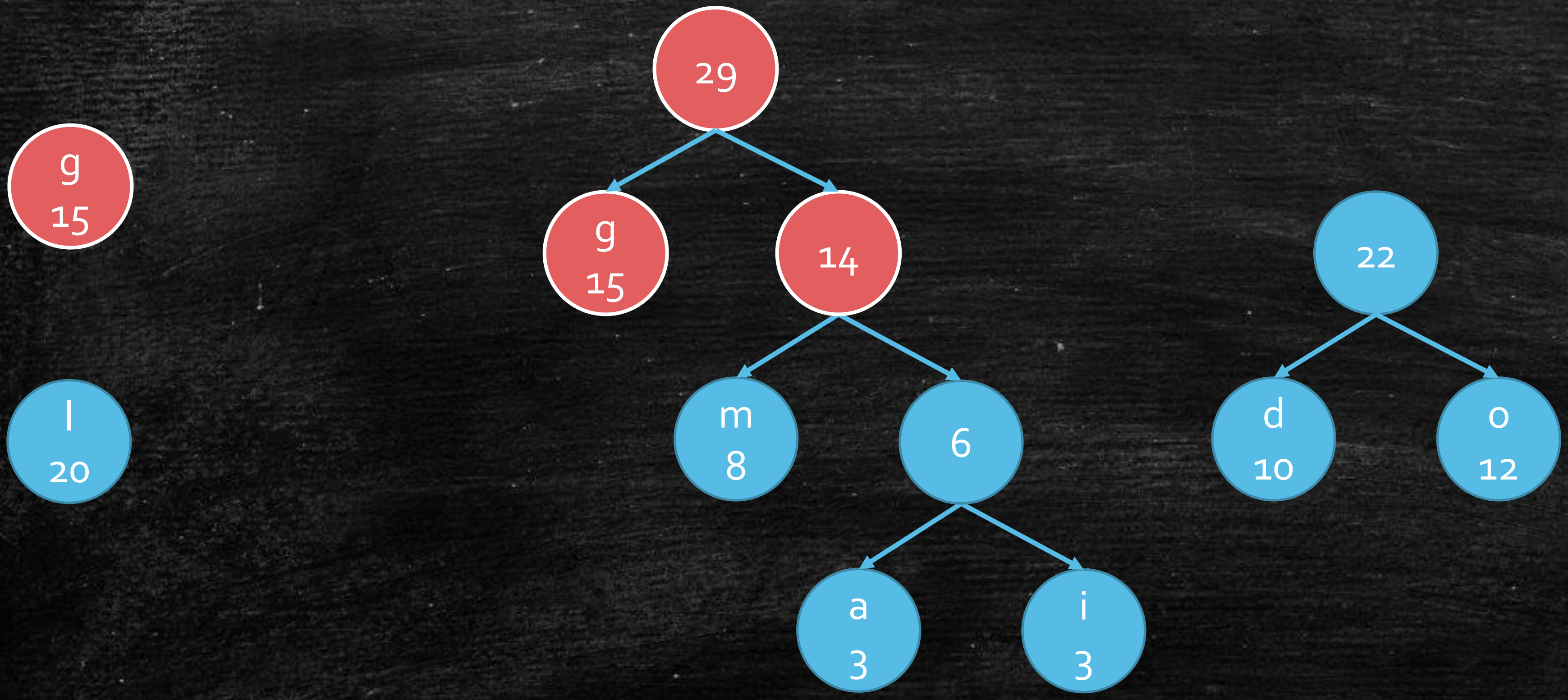
A bottom-up building



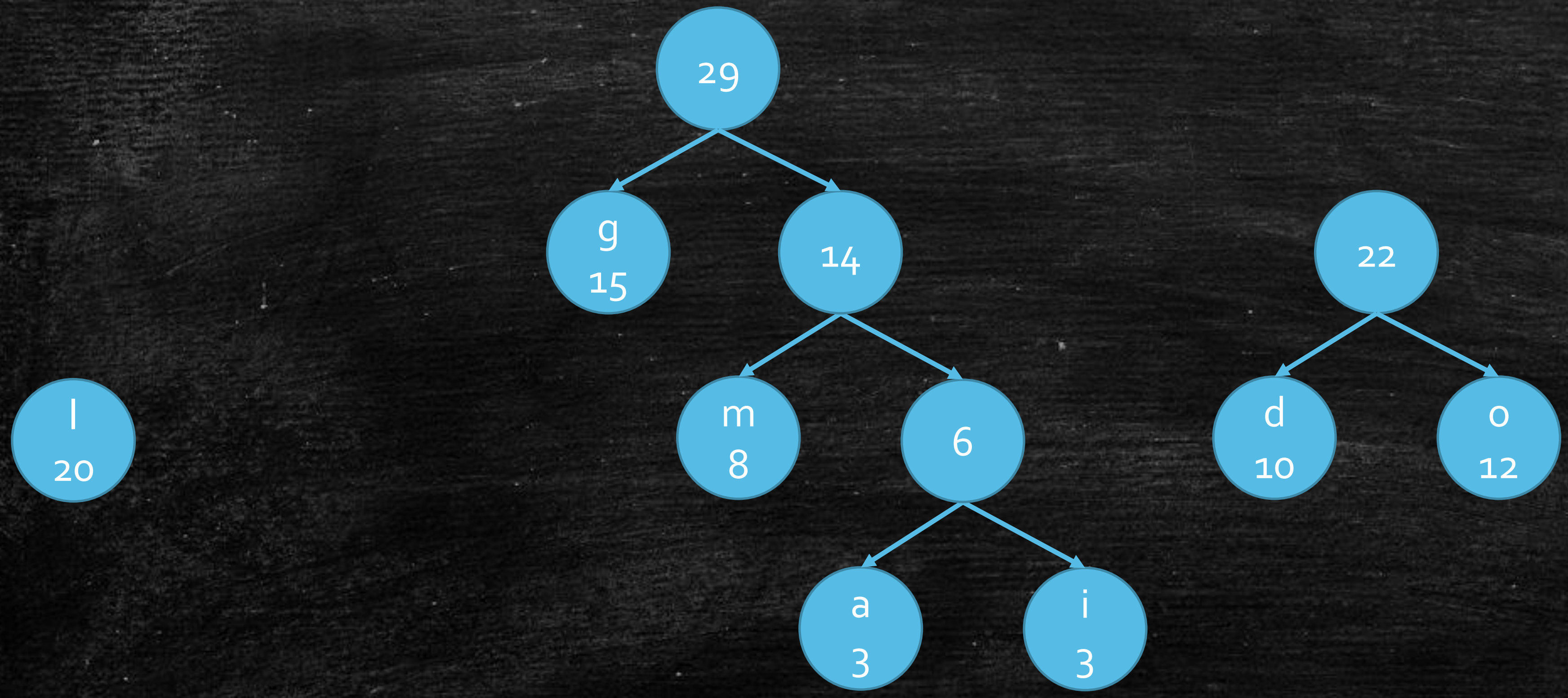
A bottom-up building



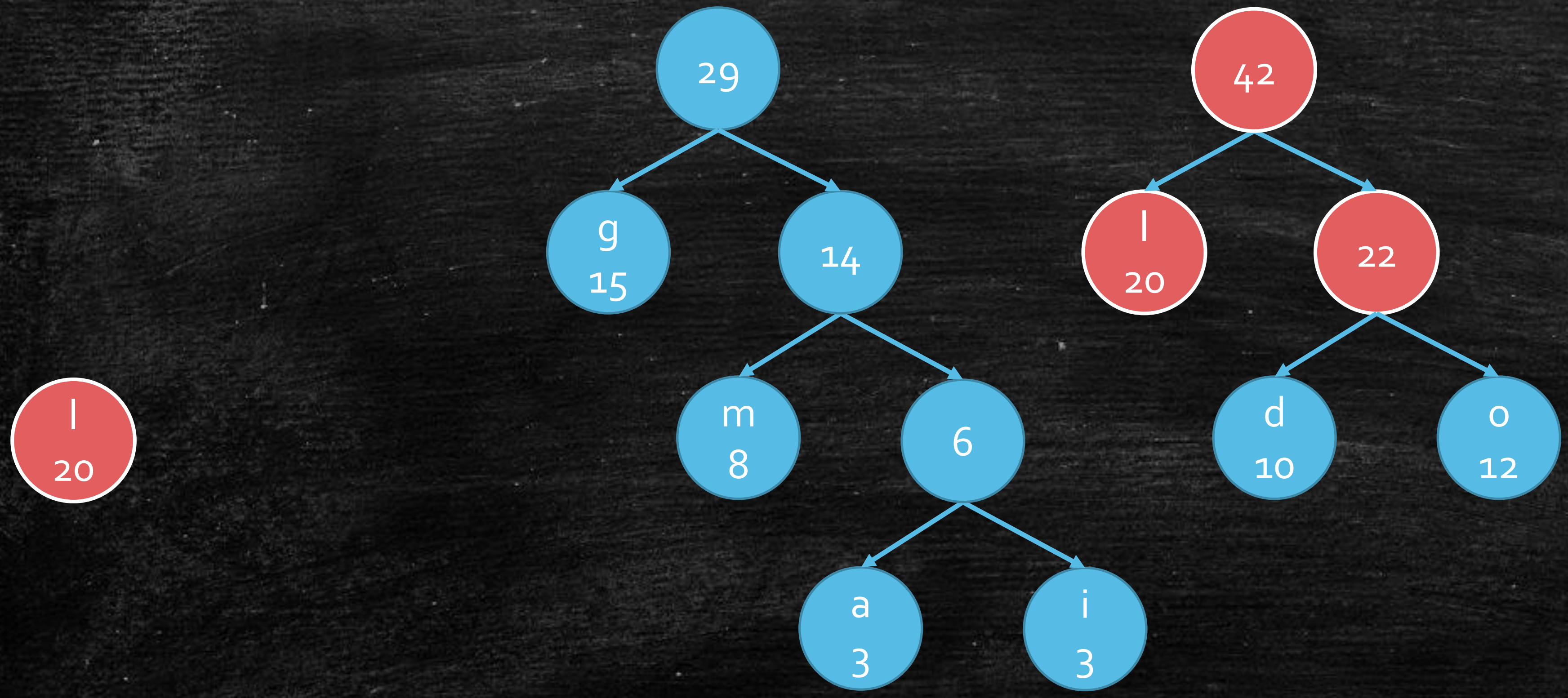
A bottom-up building



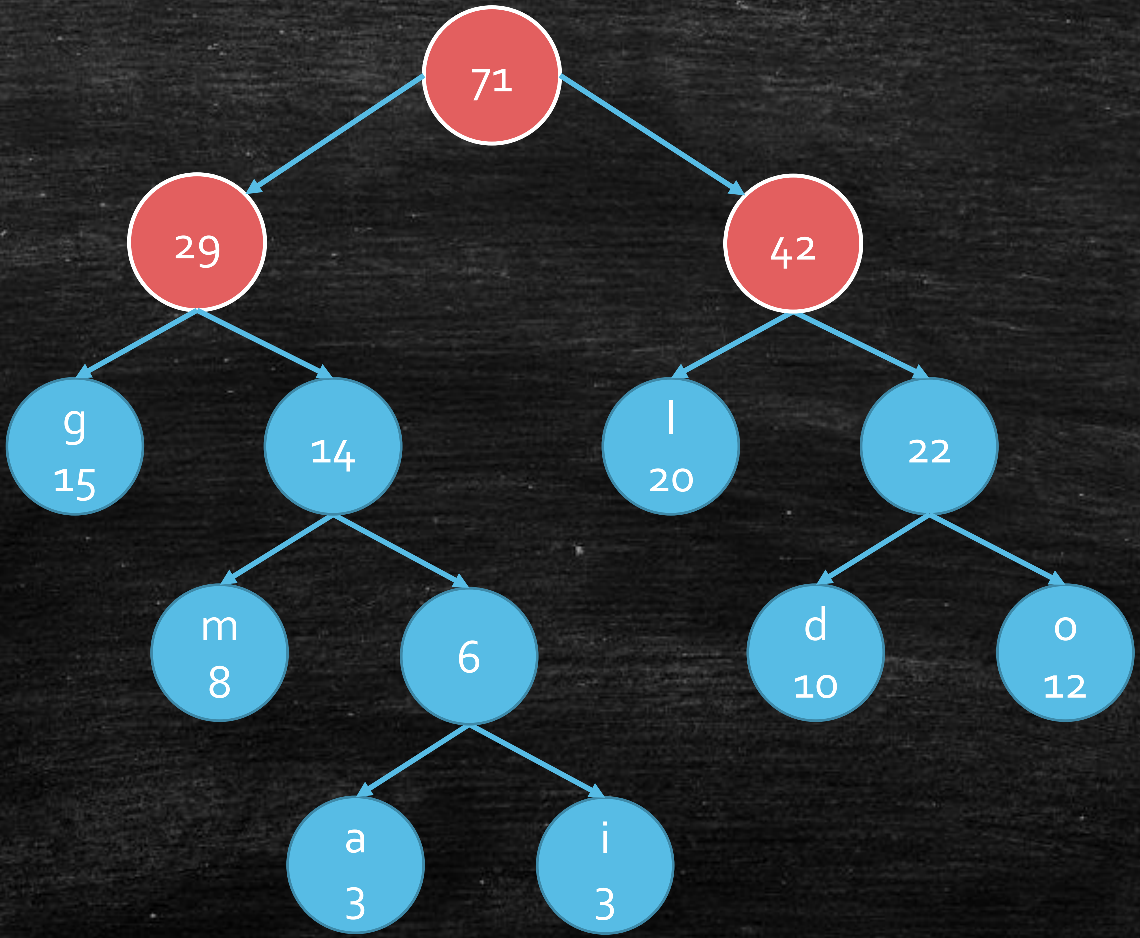
A bottom-up building



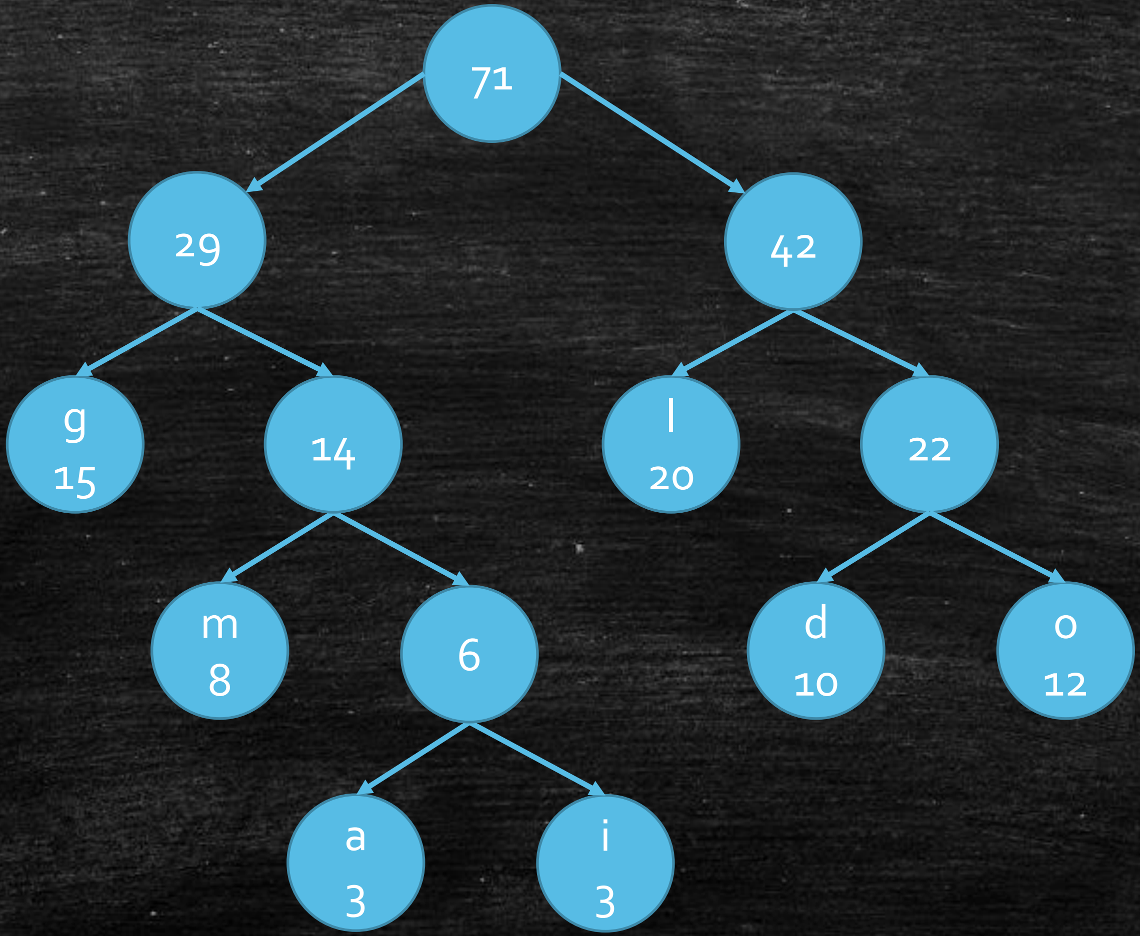
A bottom-up building



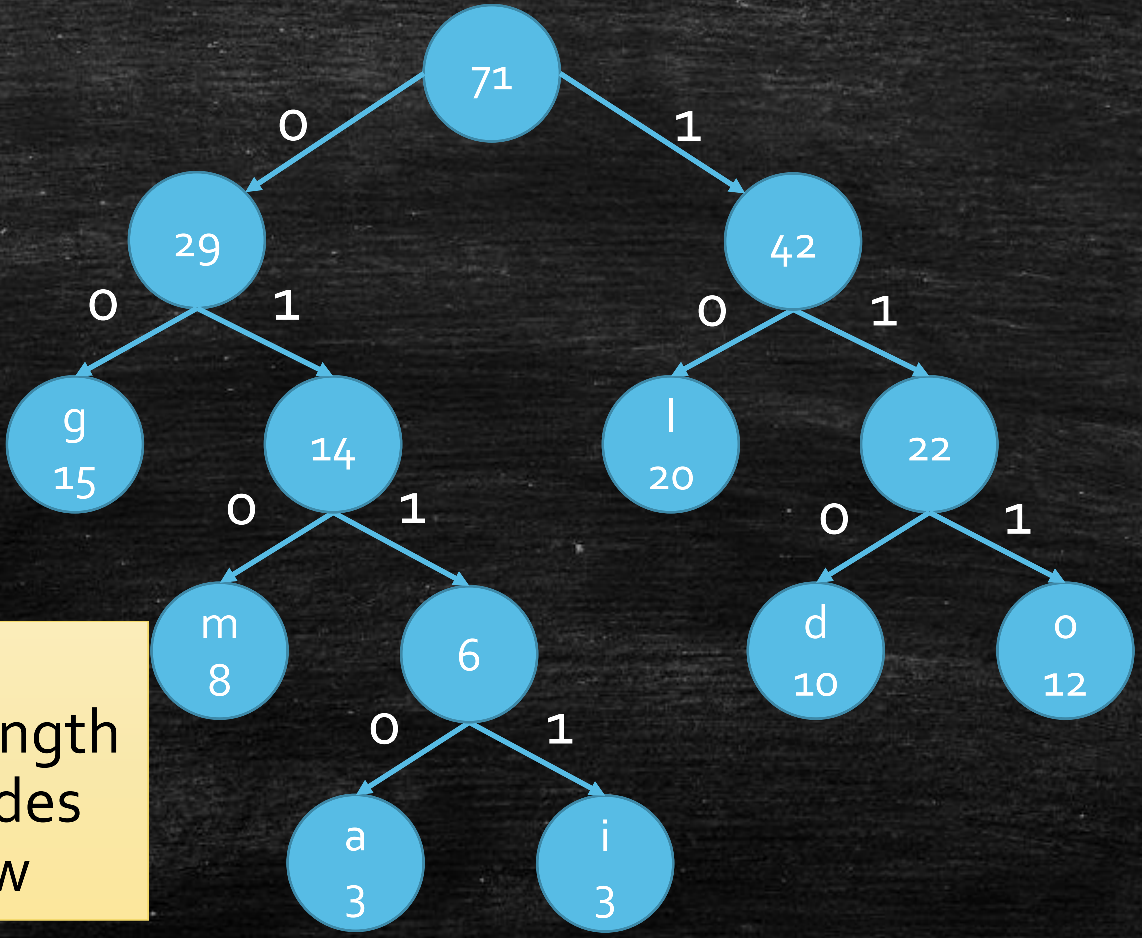
A bottom-up building



A bottom-up building



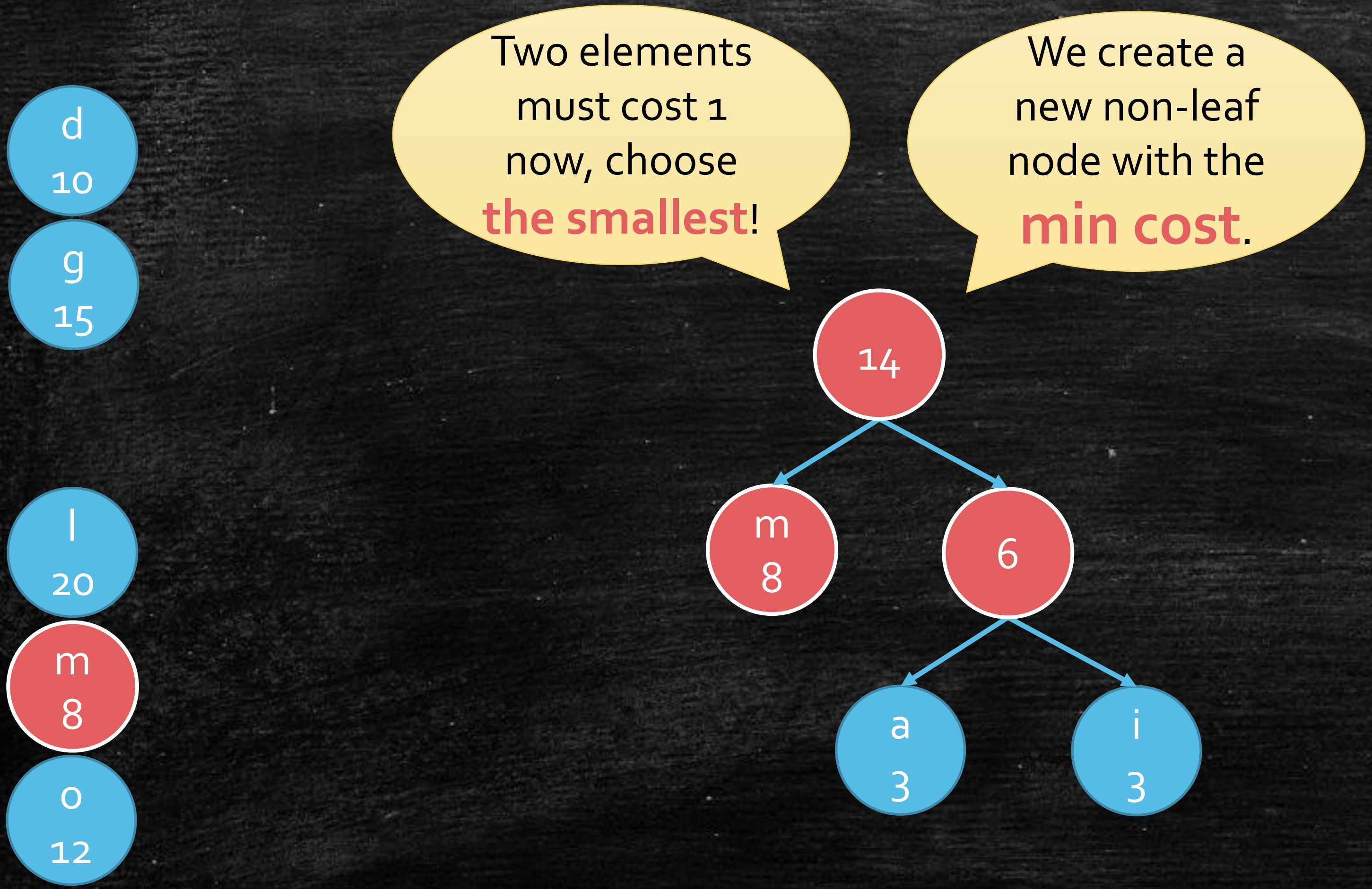
A bottom-up building



Cost

- Sum of weighted length
- Sum of non-leaf nodes
- An augmenting view

Is the cost minimized?

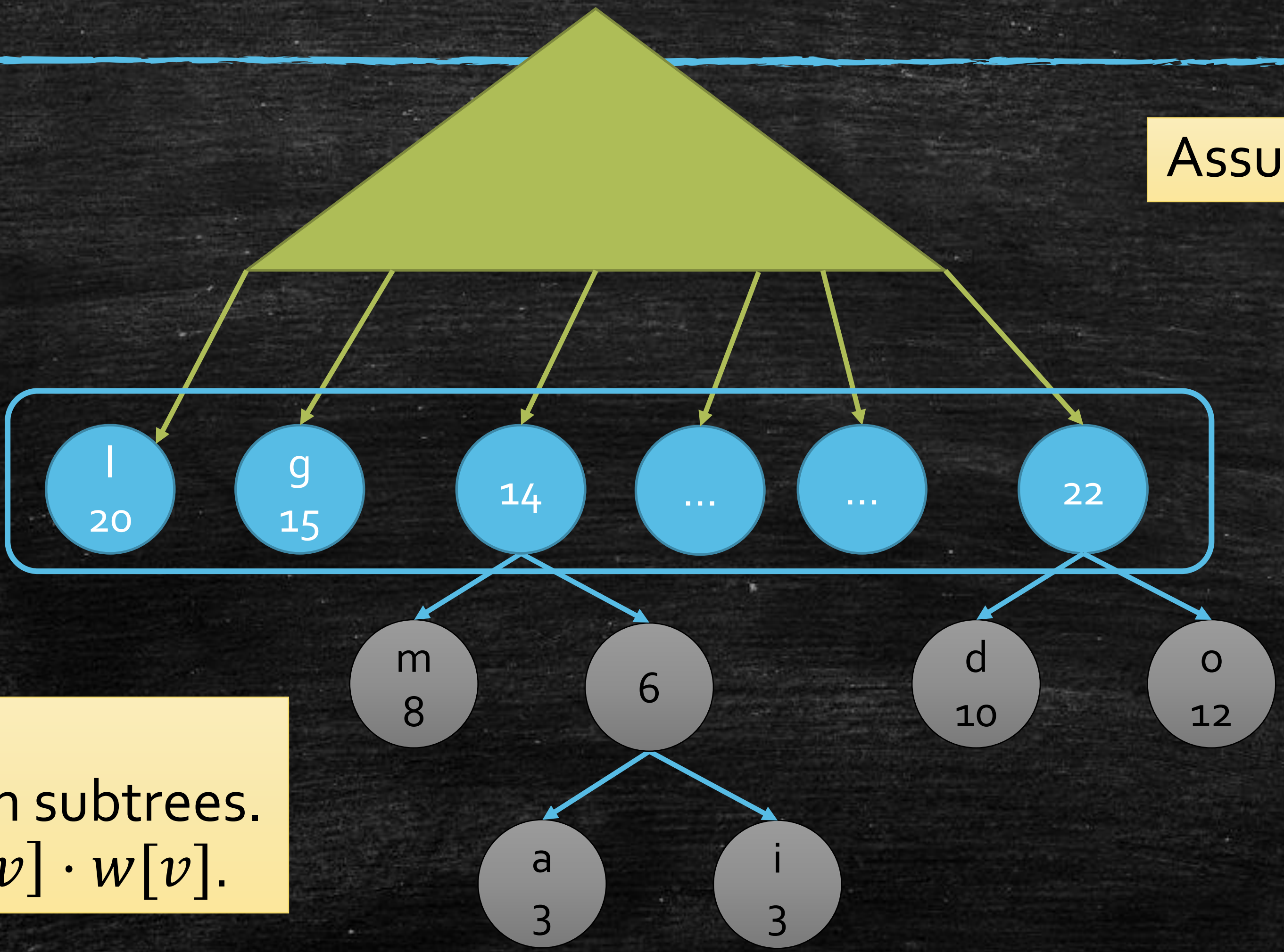


But these are intuitions, how to prove?

Proof of The Correctness

- The Big Idea
  - The local greedy choice do not ruin out OPT.
- Assume we are still in a partial-OPT,
- after Merging two smallest elements, are we still in?

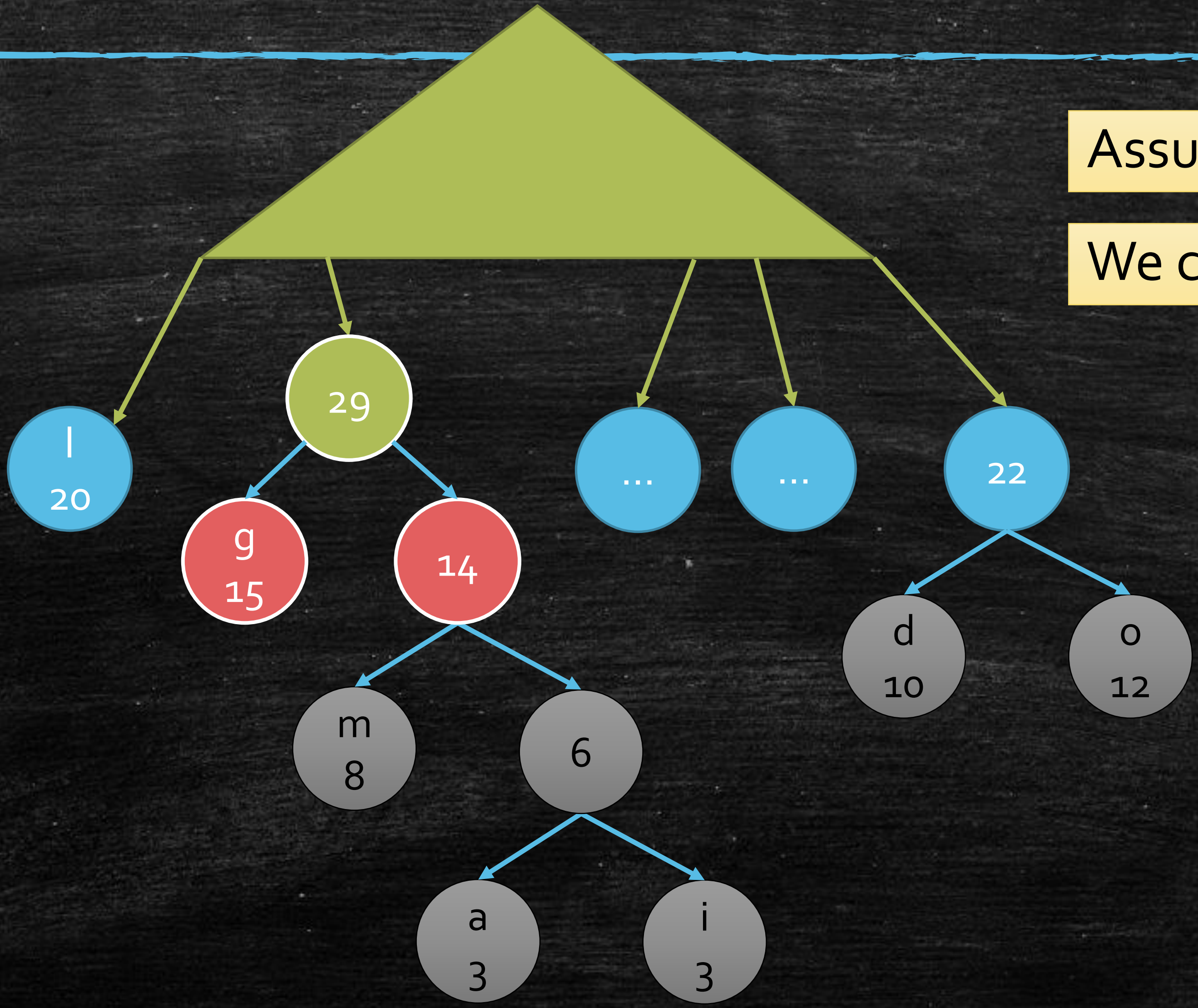
Induction



Assume we are in T^*.

Cost
1. Inside cost in subtrees.
2. Sum of $\mathrm{lev}[v] \cdot w[v]$.

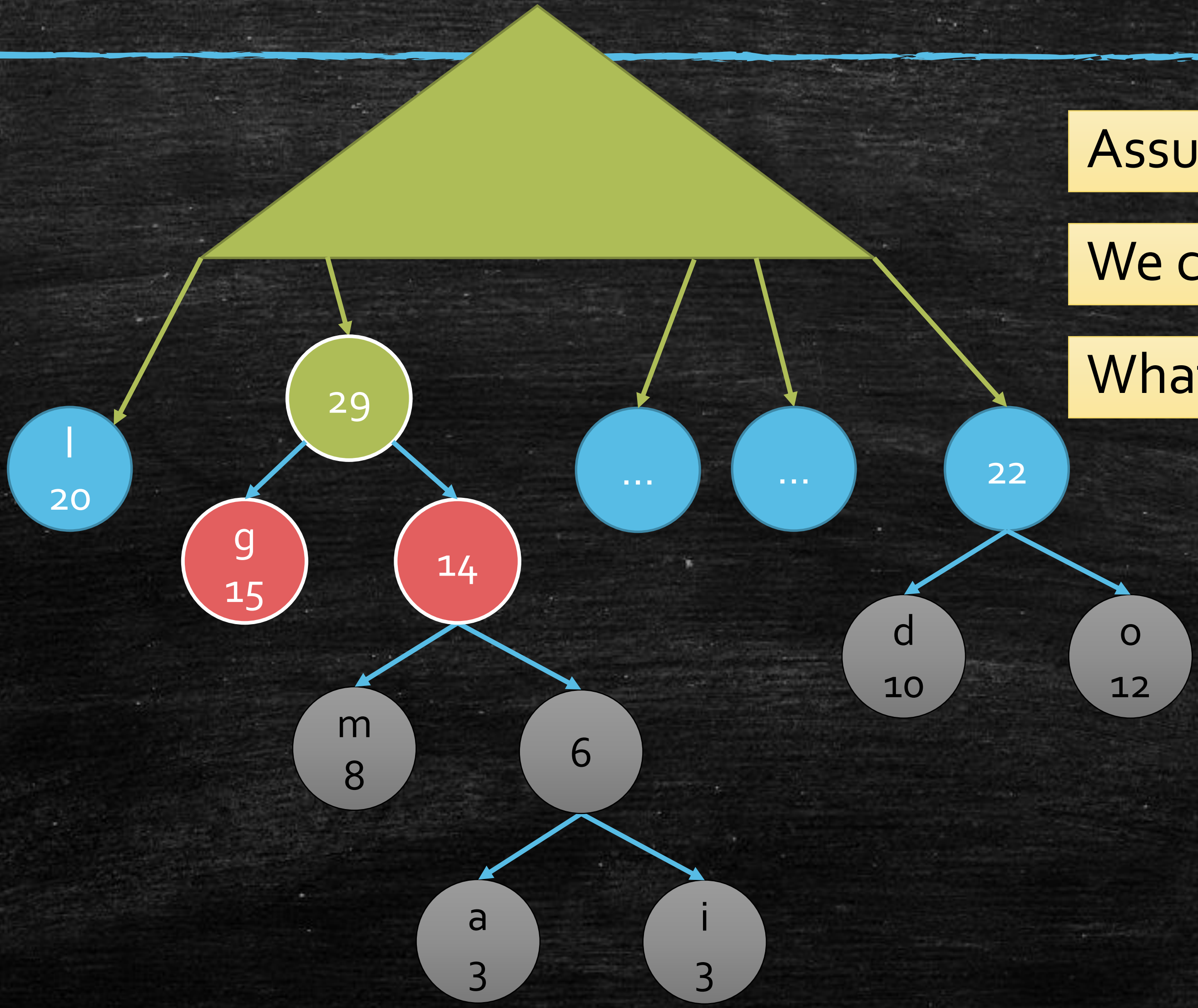
Induction



Assume we are in T^*.

We choose 14 and 15.

Induction

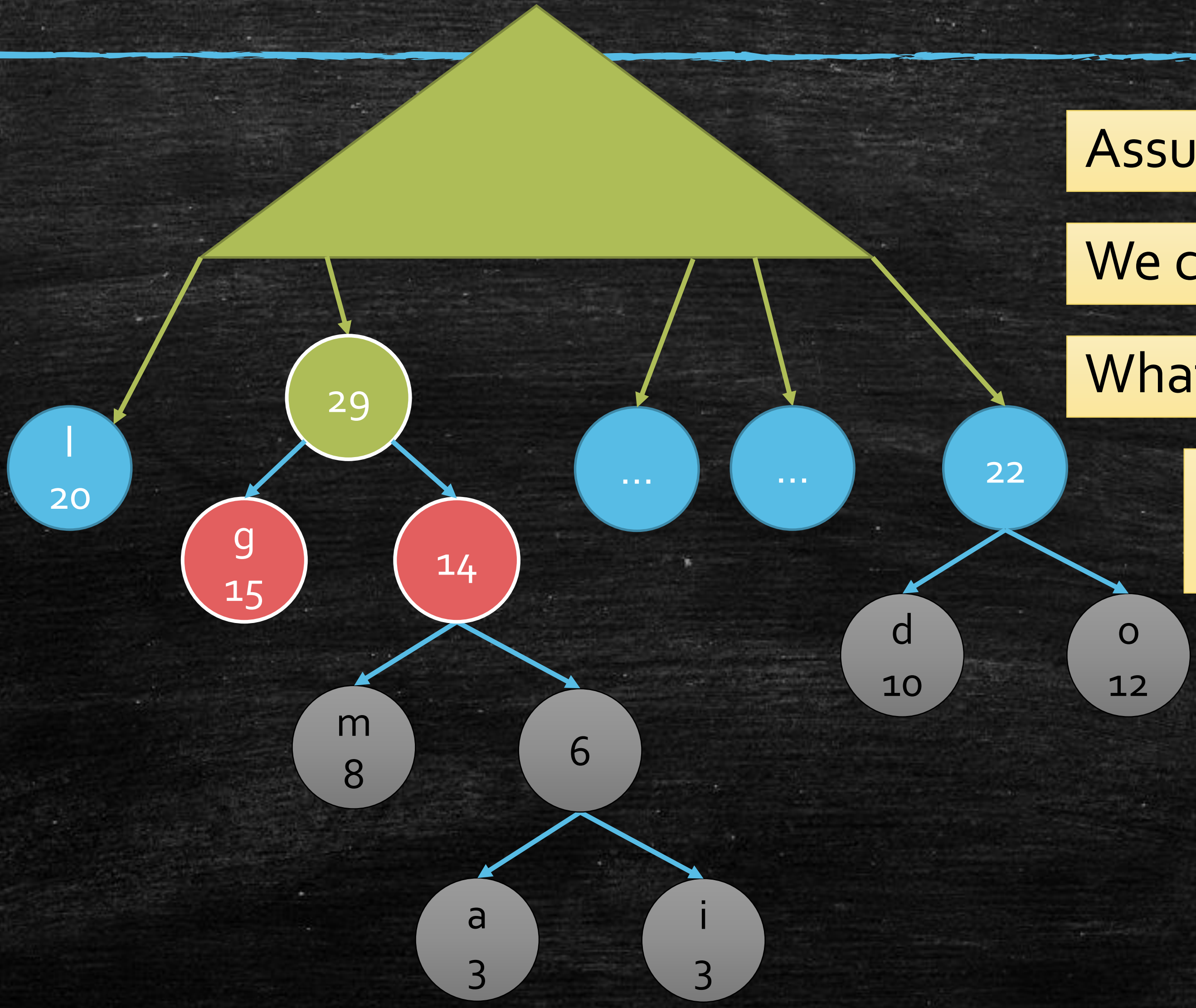


Assume we are in T^*.

We choose 14 and 15.

What if we are in trouble?

Induction



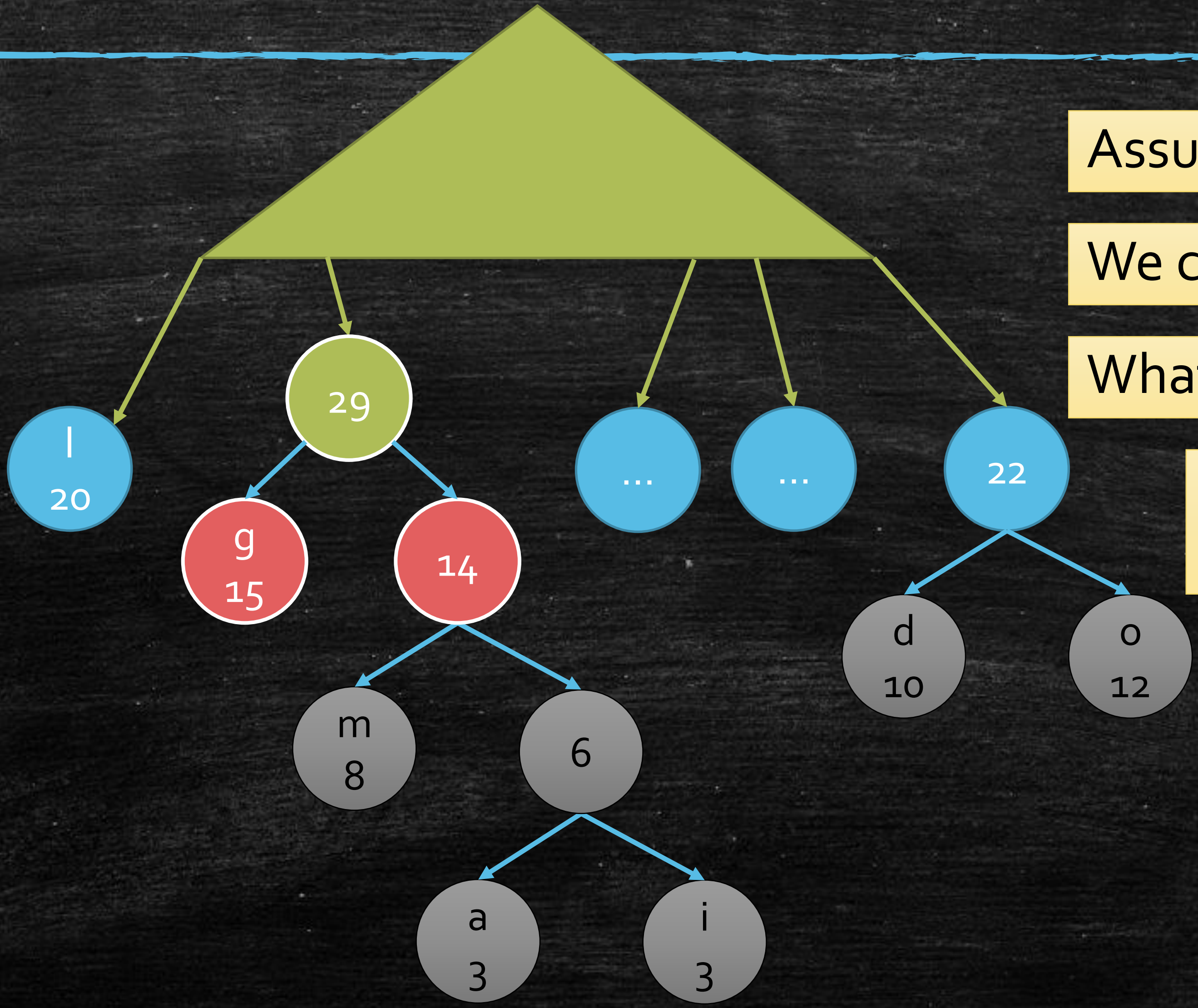
Assume we are in T^*.

We choose 14 and 15.

What if we are in trouble?

14 and 15 are not sibling in T^*

Induction



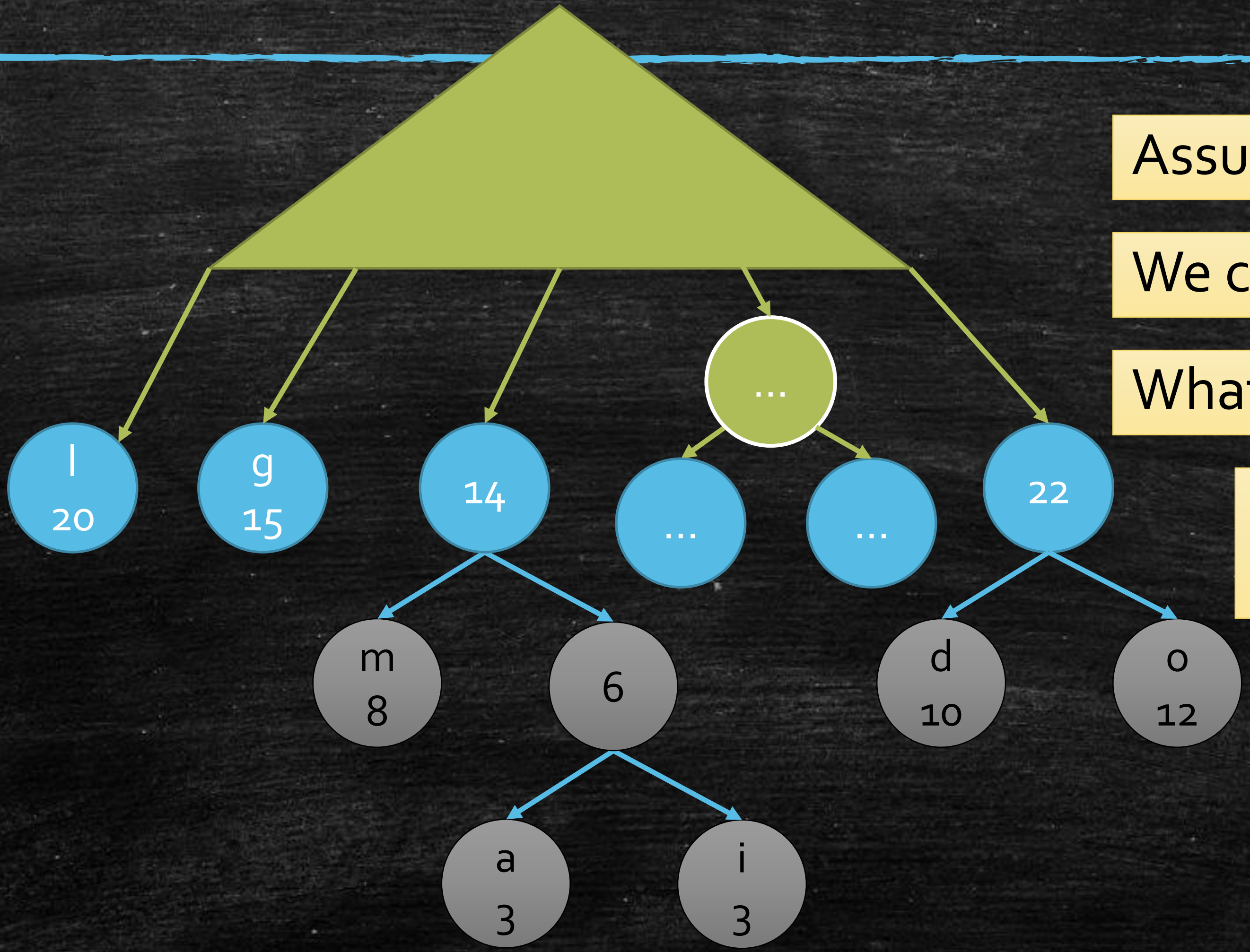
Assume we are in T^*.

We choose 14 and 15.

What if we are in trouble?

14 and 15 are not sibling in T^*

Induction



Assume we are in T^*.

We choose 14 and 15.

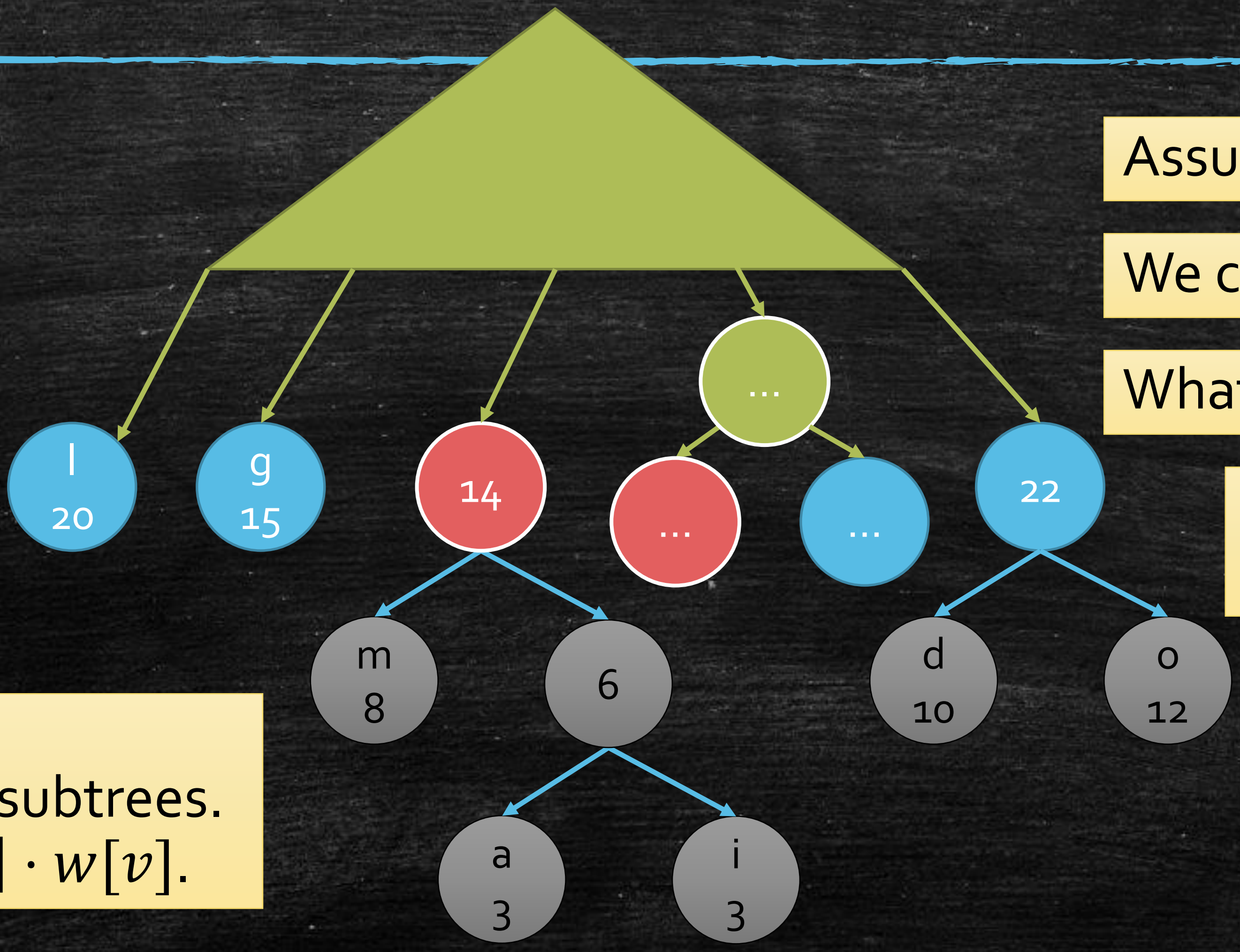
What if we are in trouble?

14 and 15 are not sibling in T^*

Check two lowest (largest lev) Siblings.

What if swap 14 or 15 in?

Induction



Assume we are in T^*.

We choose 14 and 15.

What if we are in trouble?

14 and 15 are not sibling in T^*

Check two lowest (largest lev) Siblings.

What if swap 14 or 15 in?

Cost

1. Inside cost in subtrees.
2. Sum of $lev[v] \cdot w[v]$.

Time Complexity

- Sort the characters by their appearance.
  - $O(n \log n)$
- Repeat n rounds
  - Find two minimized appearance elements.
  - Delete two minimized element.
  - Insert a super node into the list.

Time Complexity

- Sort the characters by their appearance.
  - $O(n \log n)$
- Repeat n rounds
  - Find two minimized appearance elements.
  - Delete two minimized element.
  - Insert a super node into the list.
- Use a Heap?
  - Each round: $\boldsymbol{O(1) + O(\log n) + O(\log n)}$.
- Totally: $O(n \log n)$ even if the characters are sorted.

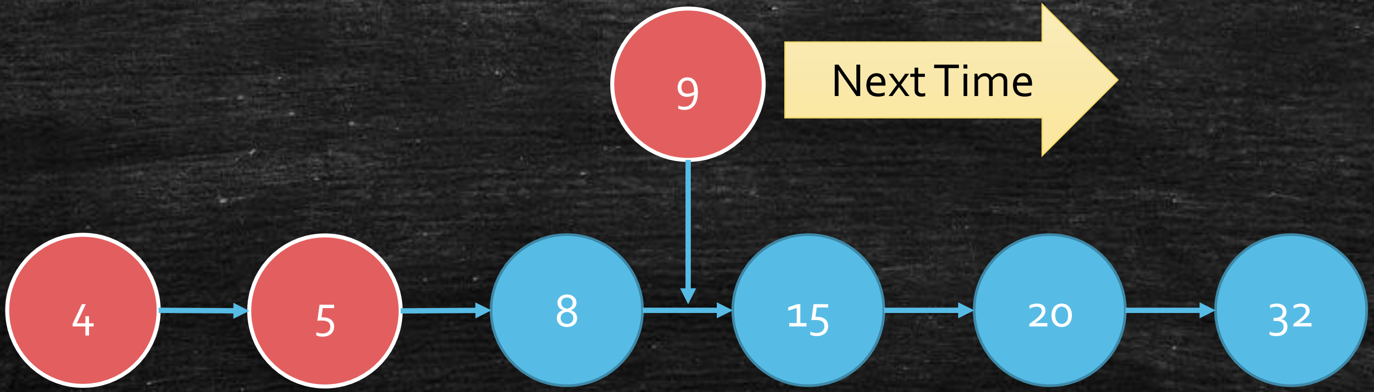
Can we Improve?

- Given a sorted list.
- Repeat n rounds
  - Find two minimized appearance elements.
  - Delete two minimized element.
  - Insert a super node into the list.
- Observation: Go back to the construction.

Can we Improve?

- Given a sorted list.
- Repeat n rounds
  - Find two minimized appearance elements.
  - Delete two minimized element.
  - Insert a super node into the list.
- Observation:
  - Inserted super nodes become **larger** and **larger**.

What if using only a sorted linked list?



- You can recall Merge Sort → Totally $o(n)$ in n rounds.
- You can also use two priority queue to implement.

Super Fun Story!!!

The story of the invention of Huffman codes is a great story that demonstrates that students can do better than professors. David Huffman (1925-1999) was a student in an electrical engineering course in 1951. His professor, Robert Fano, offered students a choice of taking a final exam or writing a term paper. Huffman did not want to take the final so he started working on the term paper. The topic of the paper was to find the most efficient (optimal) code. What Professor Fano did not tell his students was the fact that it was an open problem and that he was working on the problem himself. Huffman spent a lot of time on the problem and was ready to give up when the solution suddenly came to him. The code he discovered was optimal, that is, it had the lowest possible average message length. The method that Fano had developed for this problem did not always produce an optimal code. Therefore, Huffman did better than his professor. Later Huffman said that likely he would not have even attempted the problem if he had known that his professor was struggling with it.

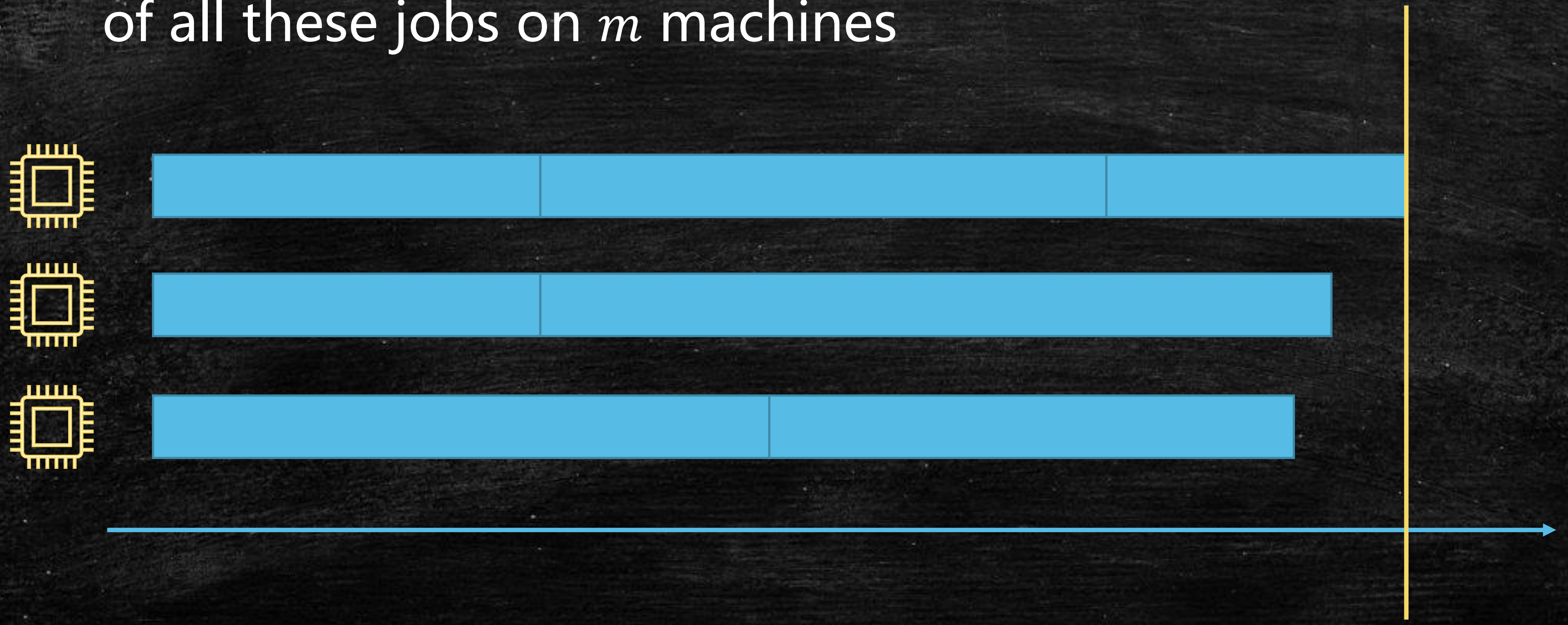
From:

https://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes

Even more Greedy!

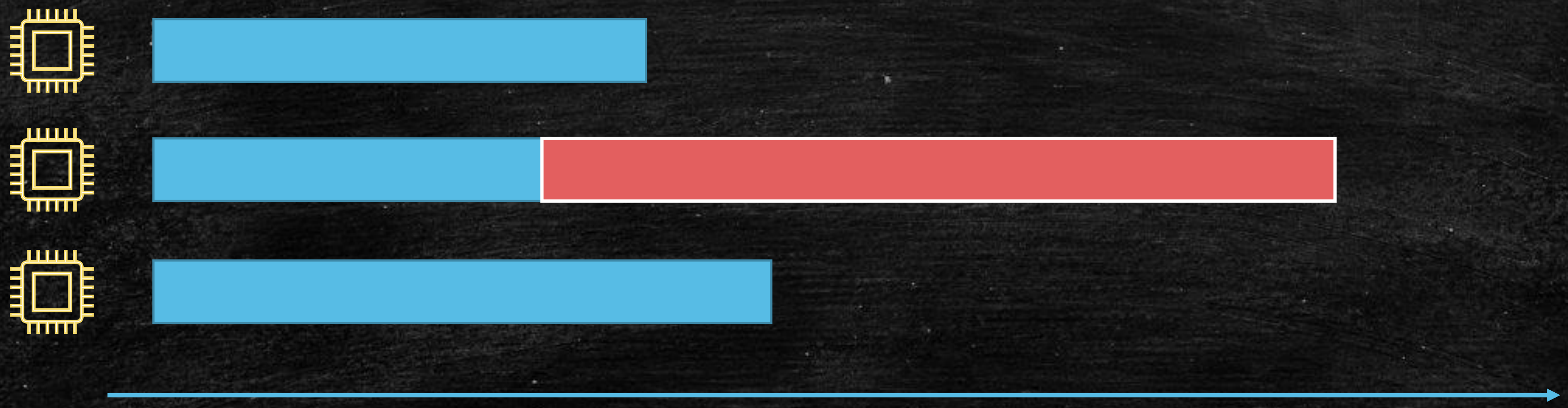
Makespan Minimization

- **Input:** m identical machines, n jobs with size p_i.
- **Output:** the minimized max completion time (**Makespan**) of all these jobs on m machines



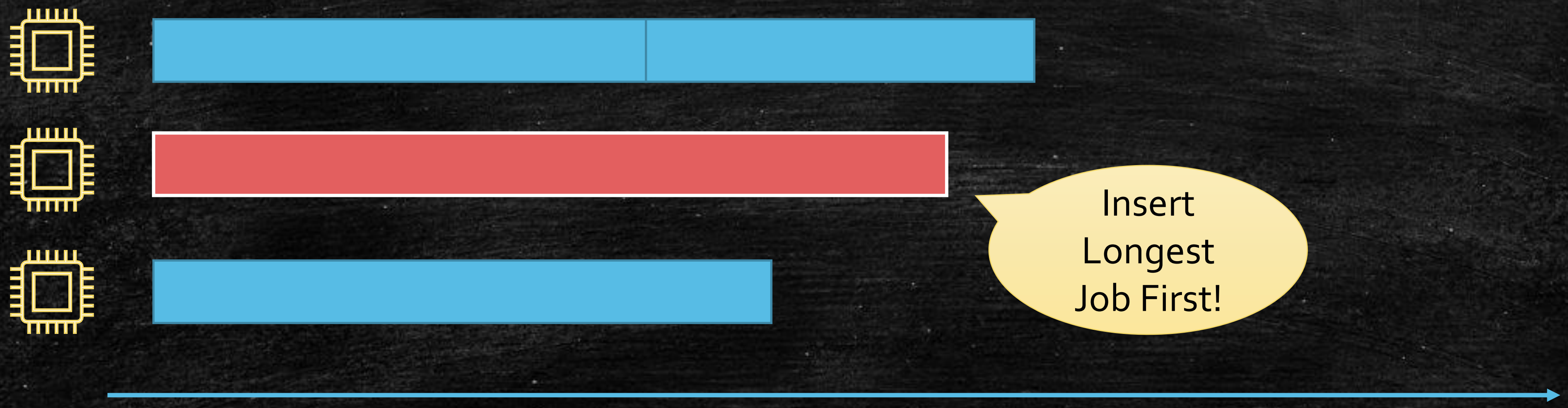
Greedy Attempt

- Schedule jobs to the earliest finished machine.
- In local view, it is optimal!



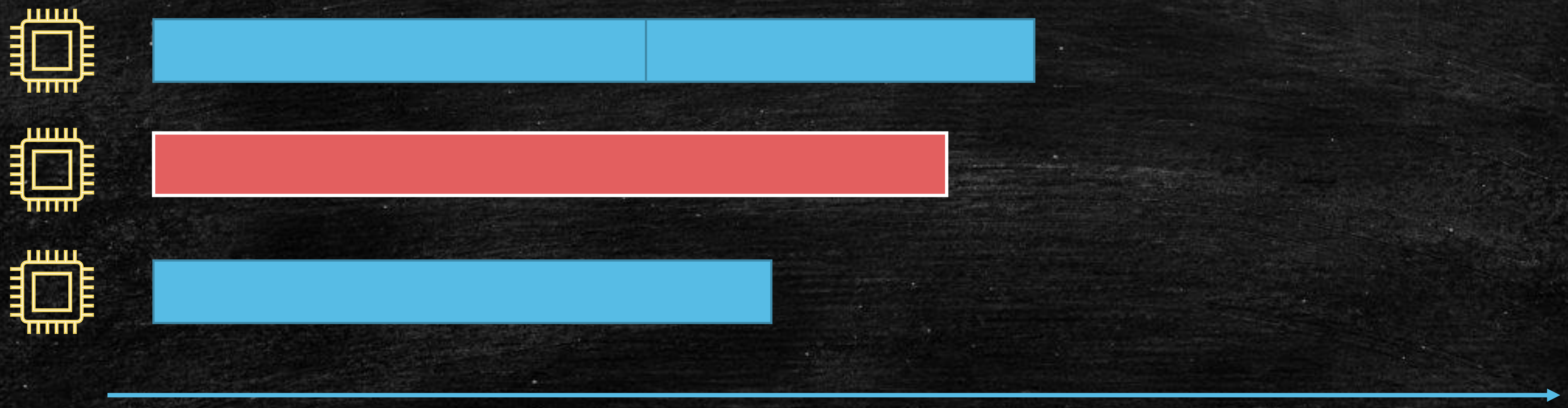
Is it optimal globally?

- No!
- Problem: the insertion order matters!



Greedy Attempt 2

- **LPT** Algorithm
  - Longest Processing Time First.
  - Insert jobs into the earliest finished machine.

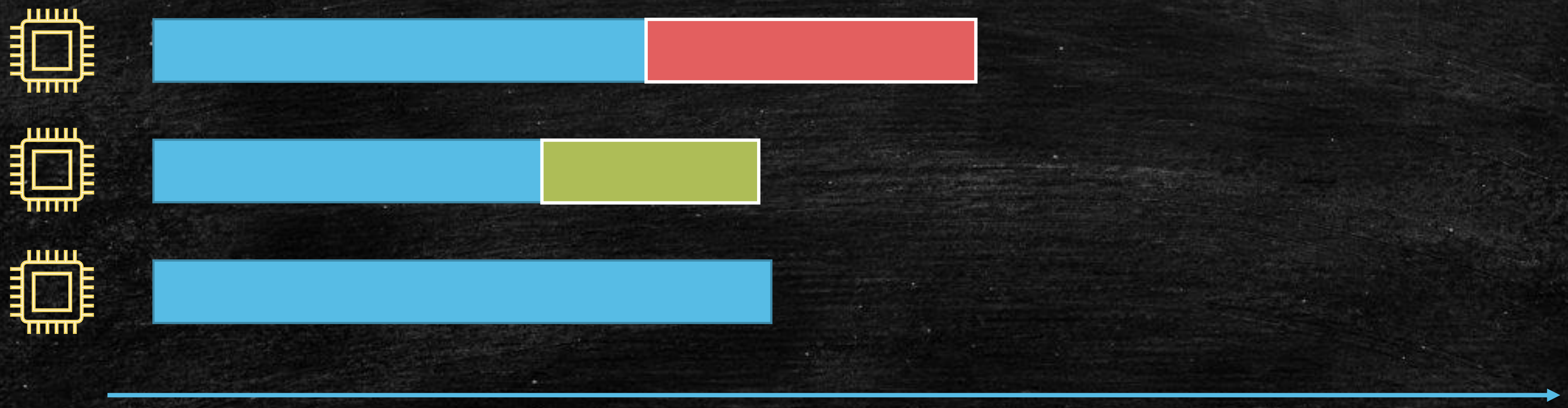


Proof of the correctness

- Assume we are still in OPT.
- We put the longest left job onto the earliest finished machine.
- **Discussion! Are we still in an OPT?**

Proof of the correctness

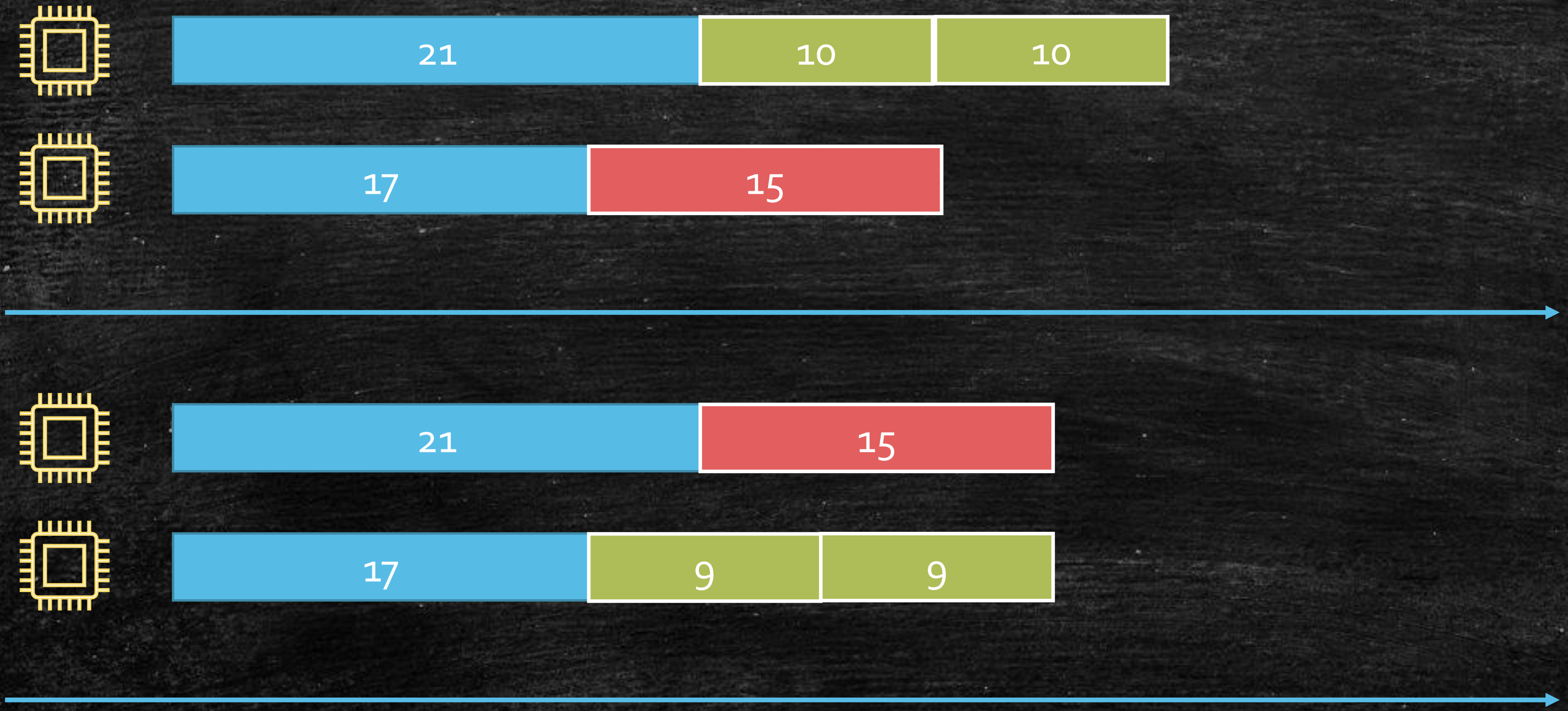
- **Discussion! Are we still in an OPT?**
- Suppose not.
- We can swap red and green!



Proof of the correctness

- **Discussion! Are we still in an OPT?**
- Suppose not.
- But what if we have two green jobs?

Thinking: is it really bad?



Proof is a way for us find problems!

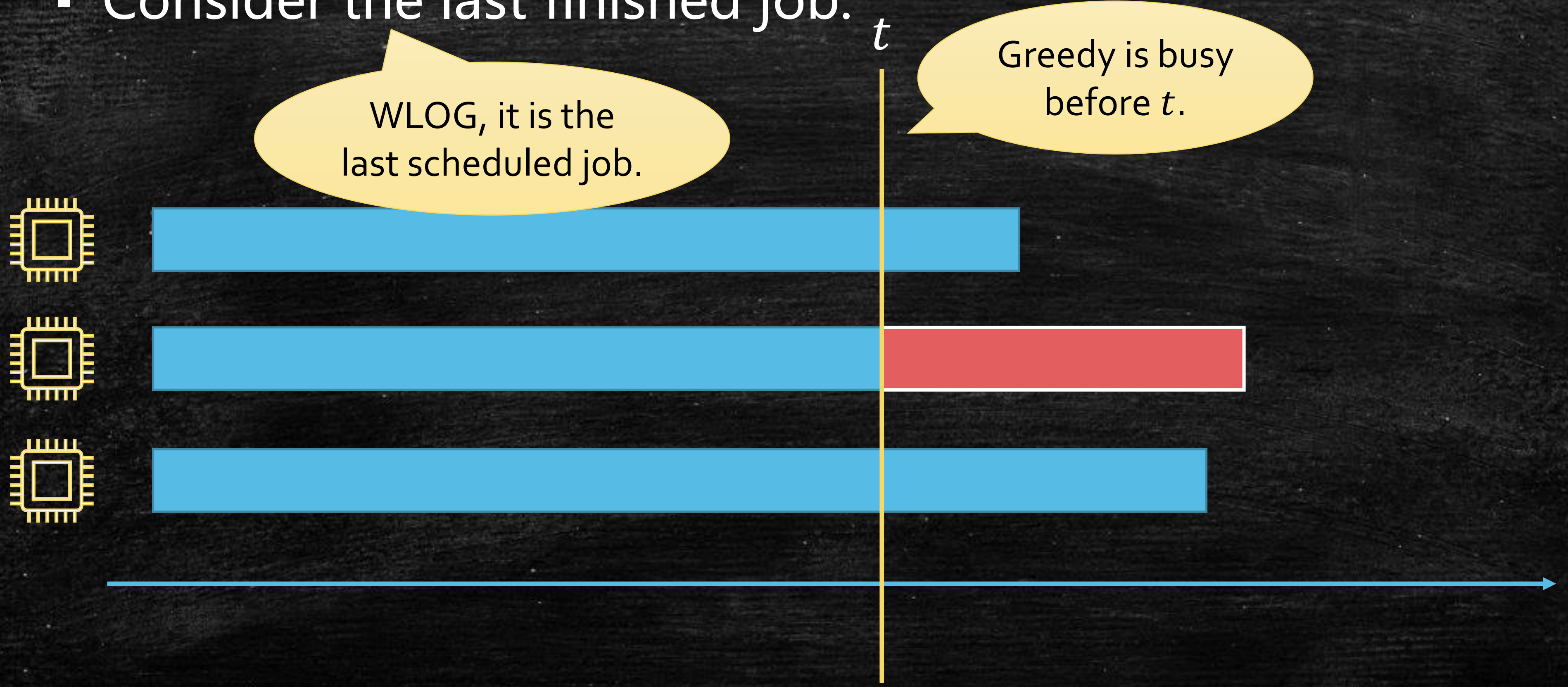
How to find a correct greedy algorithm?

Makespan Minimization

- Makespan Minimization is a NP-hard problem.
- Find a poly time algorithm for it means $P = NP$.
- Is Simple Greedy or LPT very bad?
- At least, they are better than arbitrary scheduling.

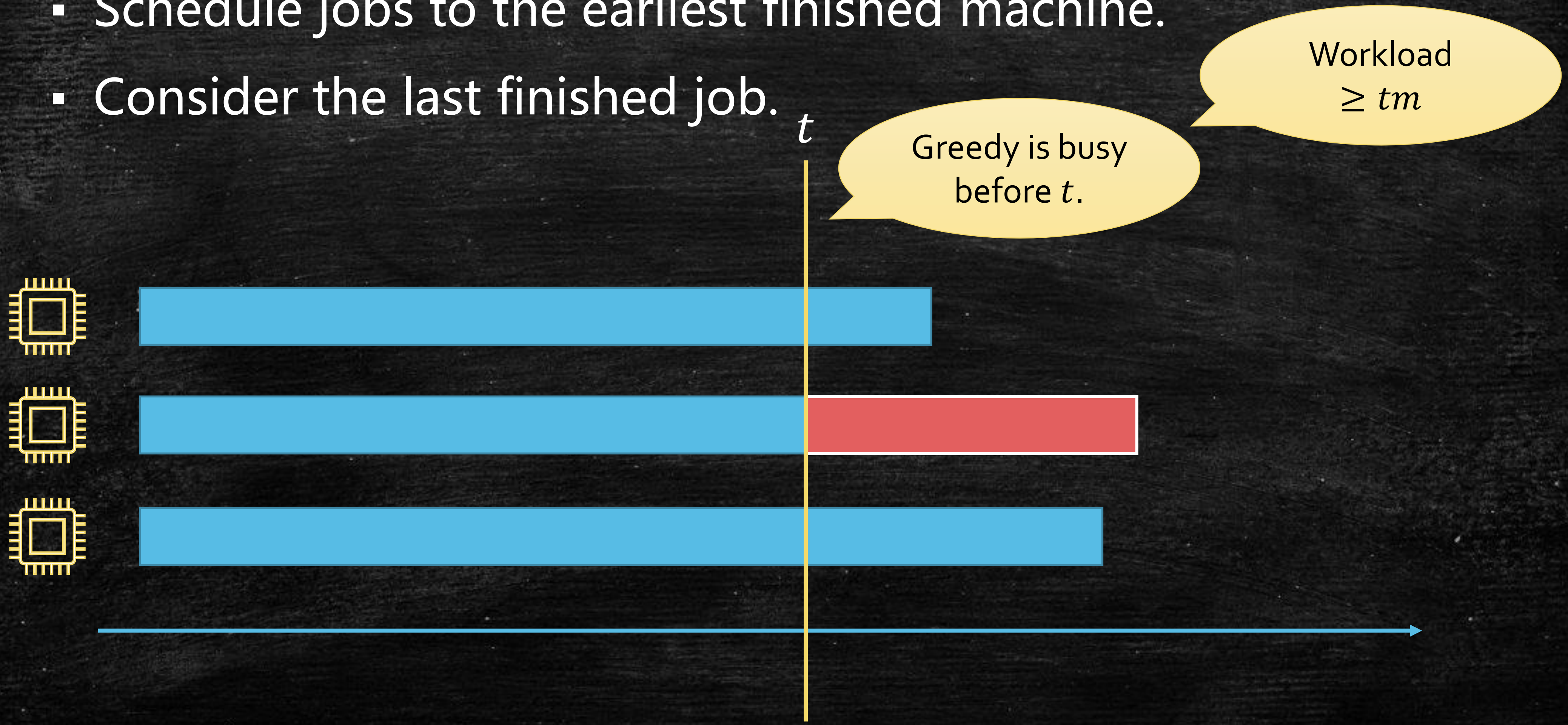
Connect Greedy Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



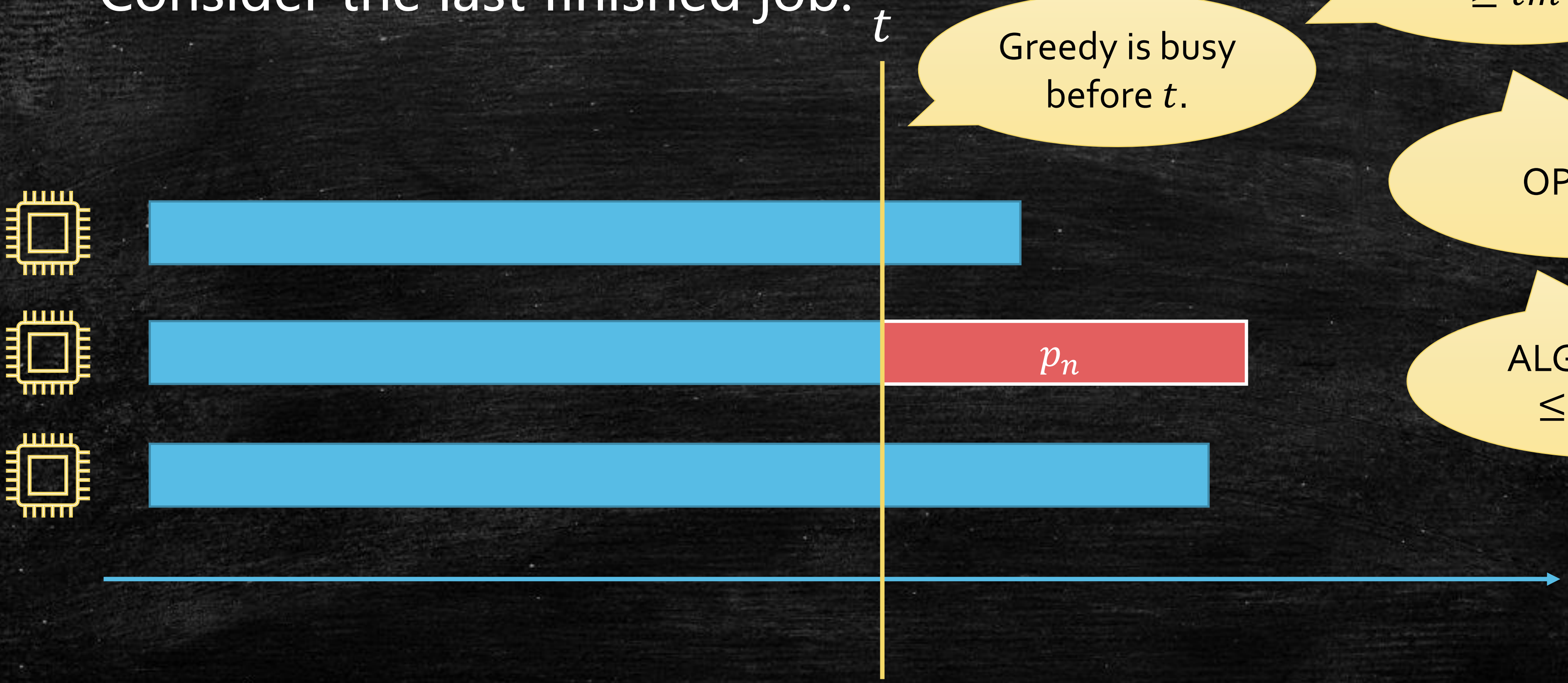
Connect Greedy Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



Connect Greedy Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



Greedy is busy before t.

Workload $\geq tm$

OPT$\geq t$

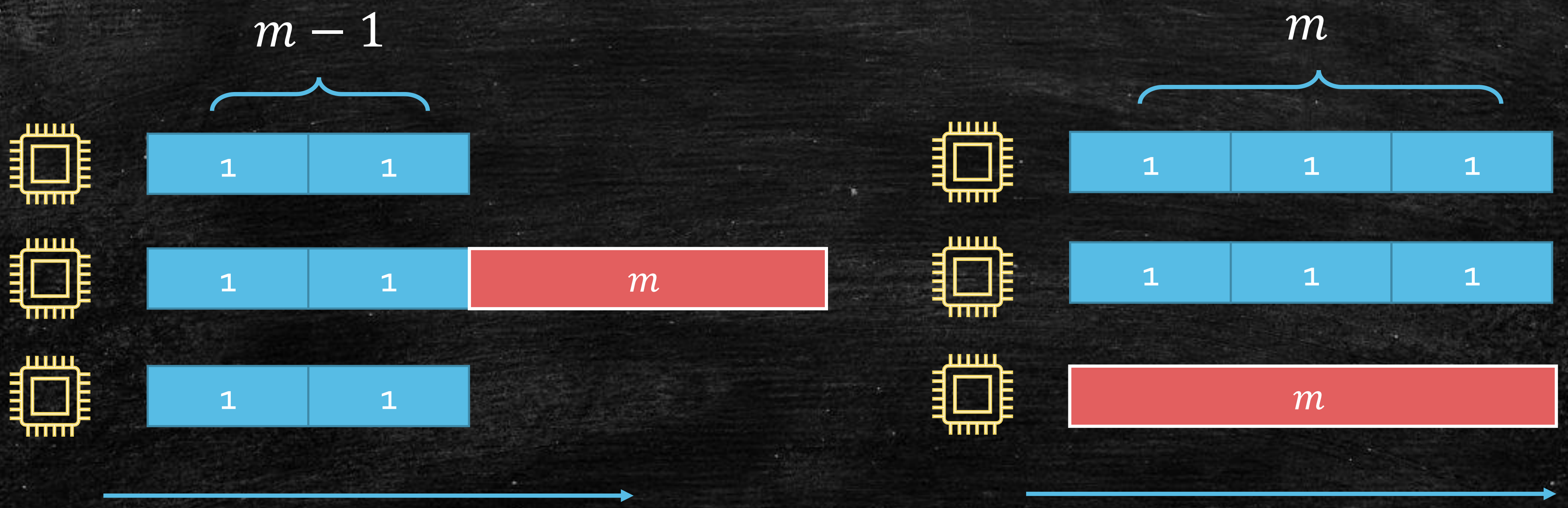
ALG$= t + p_n$
$\leq 2 \cdot$OPT

Connect Greedy Solution to OPT

- Greedy is at most 2 times of OPT!
- We call Greedy is an Approximation Algorithm
  - Approximation Ratio is 2.
  - 2-approximate Algorithm.
- It seems not tight, can we reduce the ratio?

Counter-Example

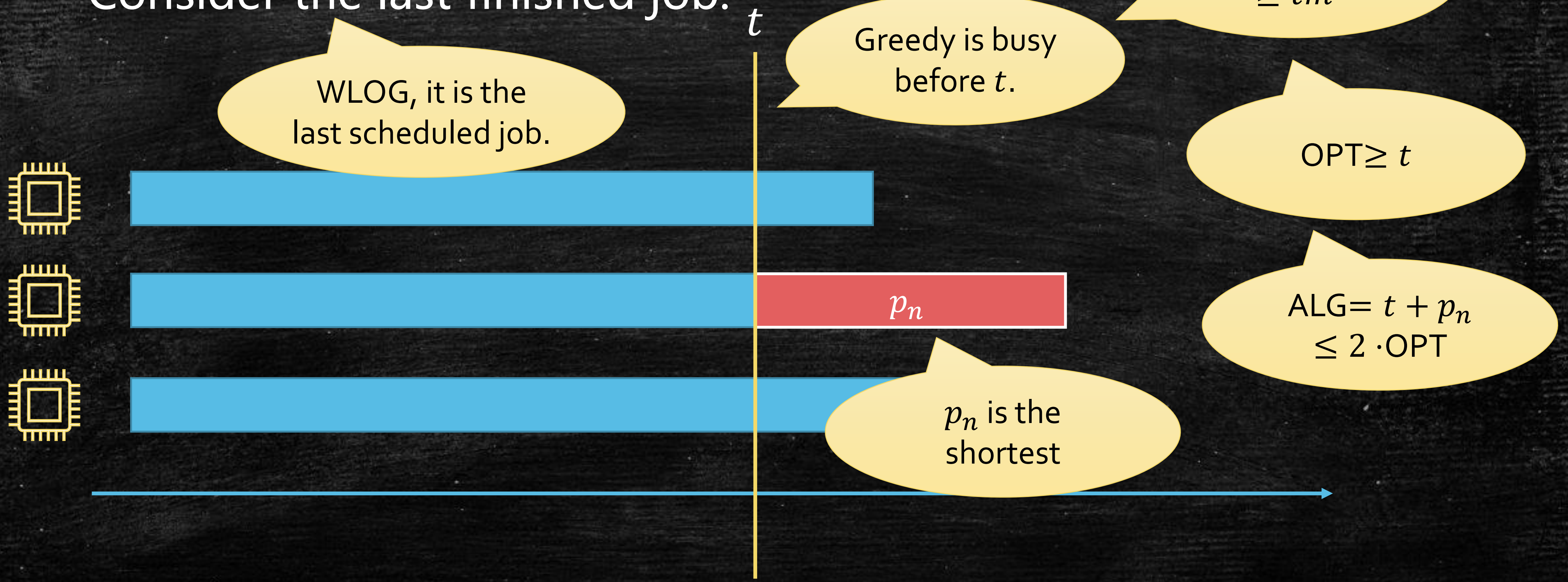
Greedy get $2m-1$, OPT get m.
Ratio $\rightarrow 2$ when m is large enough.



Can we improve the approximation ratio by **LPT**?

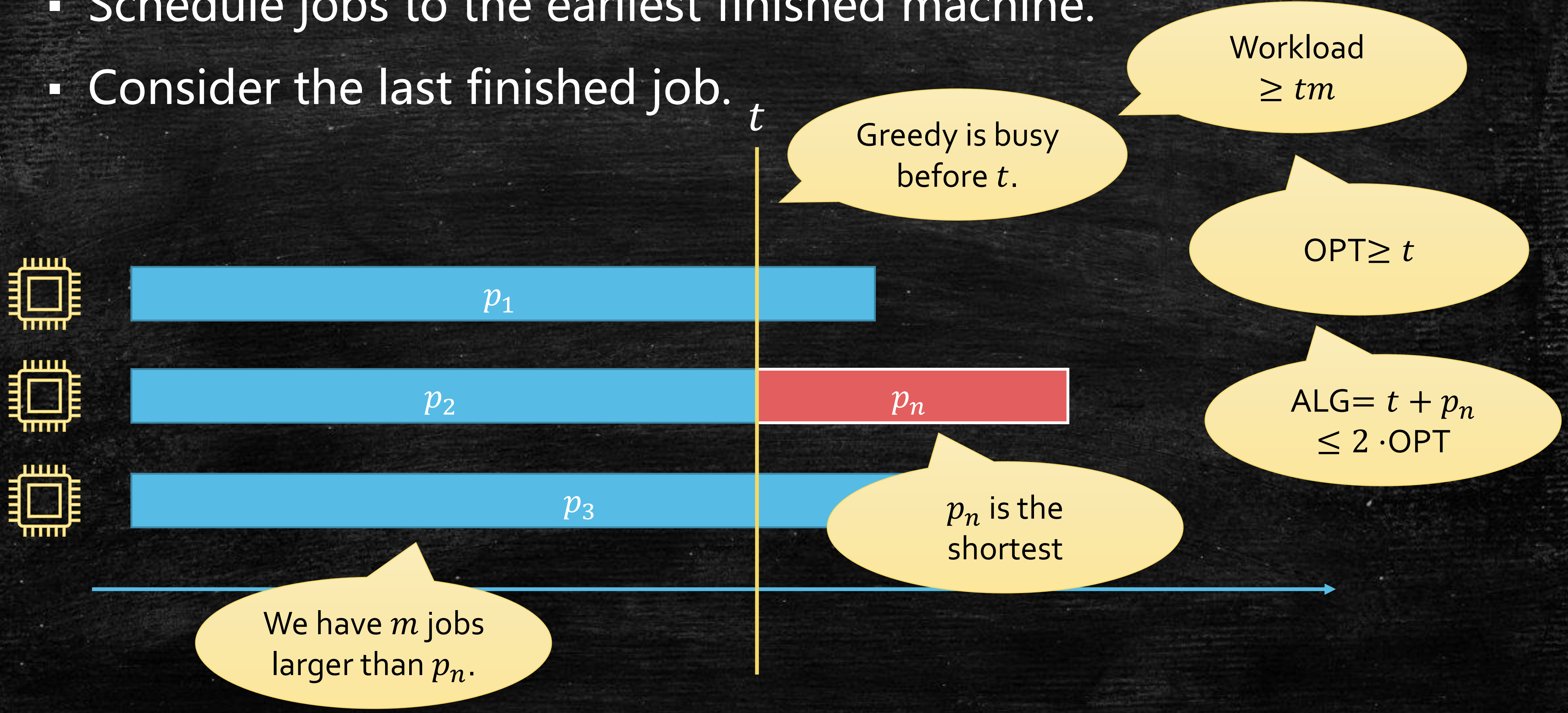
Connect LPT Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



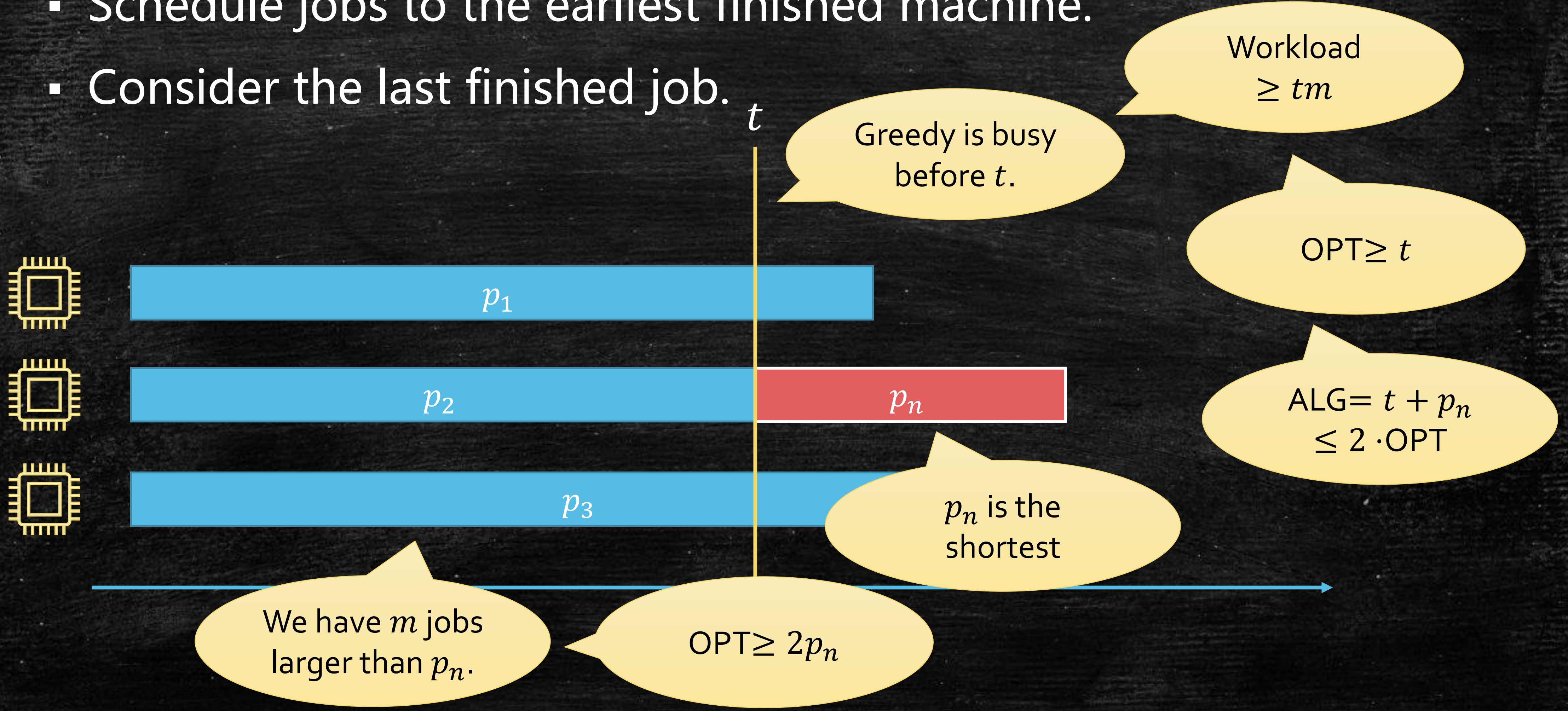
Connect LPT Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



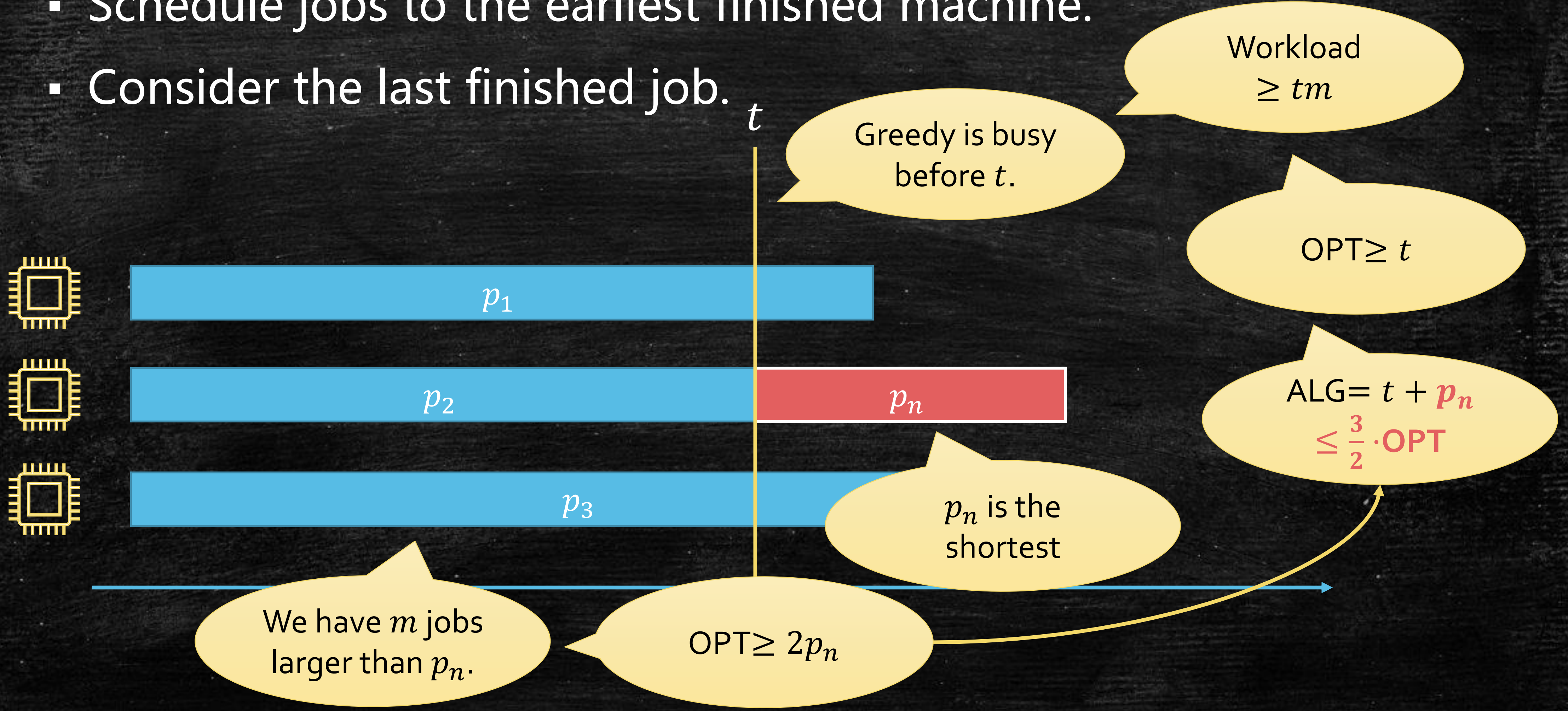
Connect LPT Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



Connect LPT Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



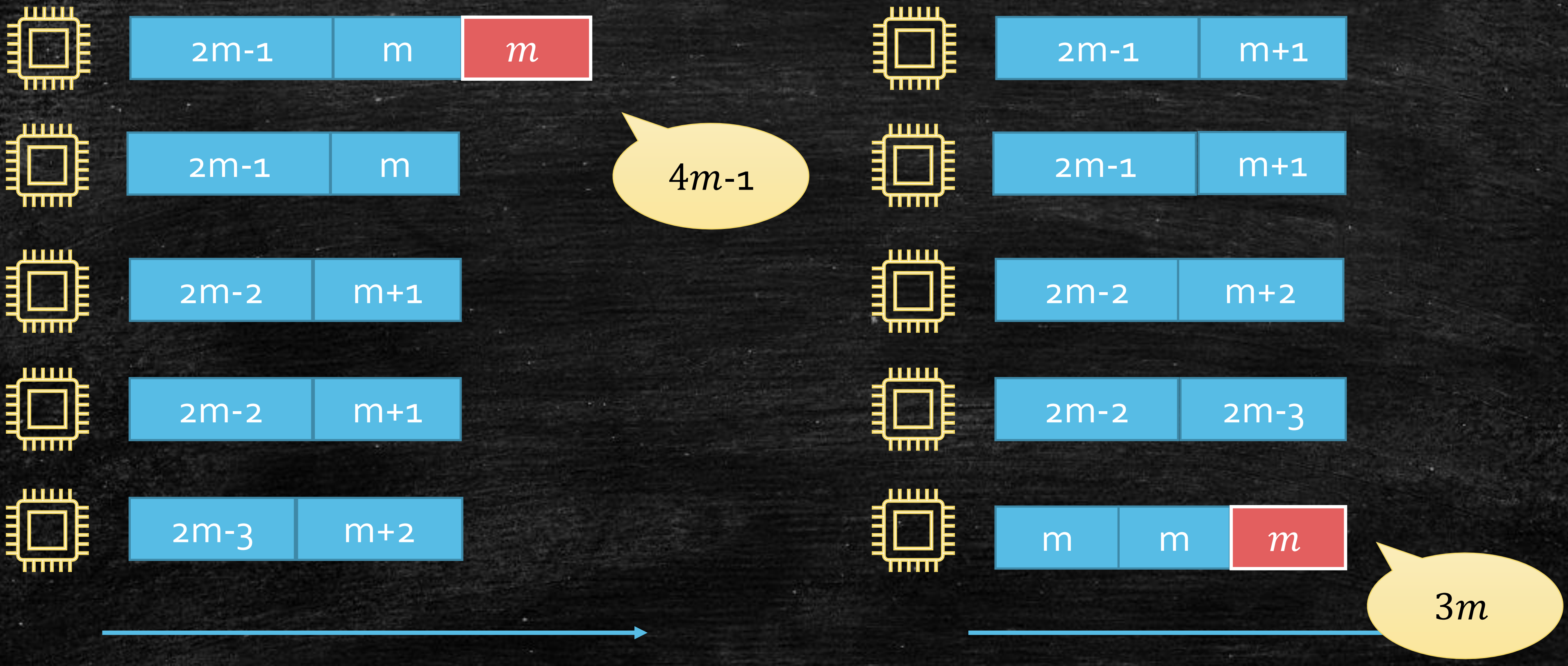
Are we done?

Is it finished?

- Not enough larger than p_n jobs?

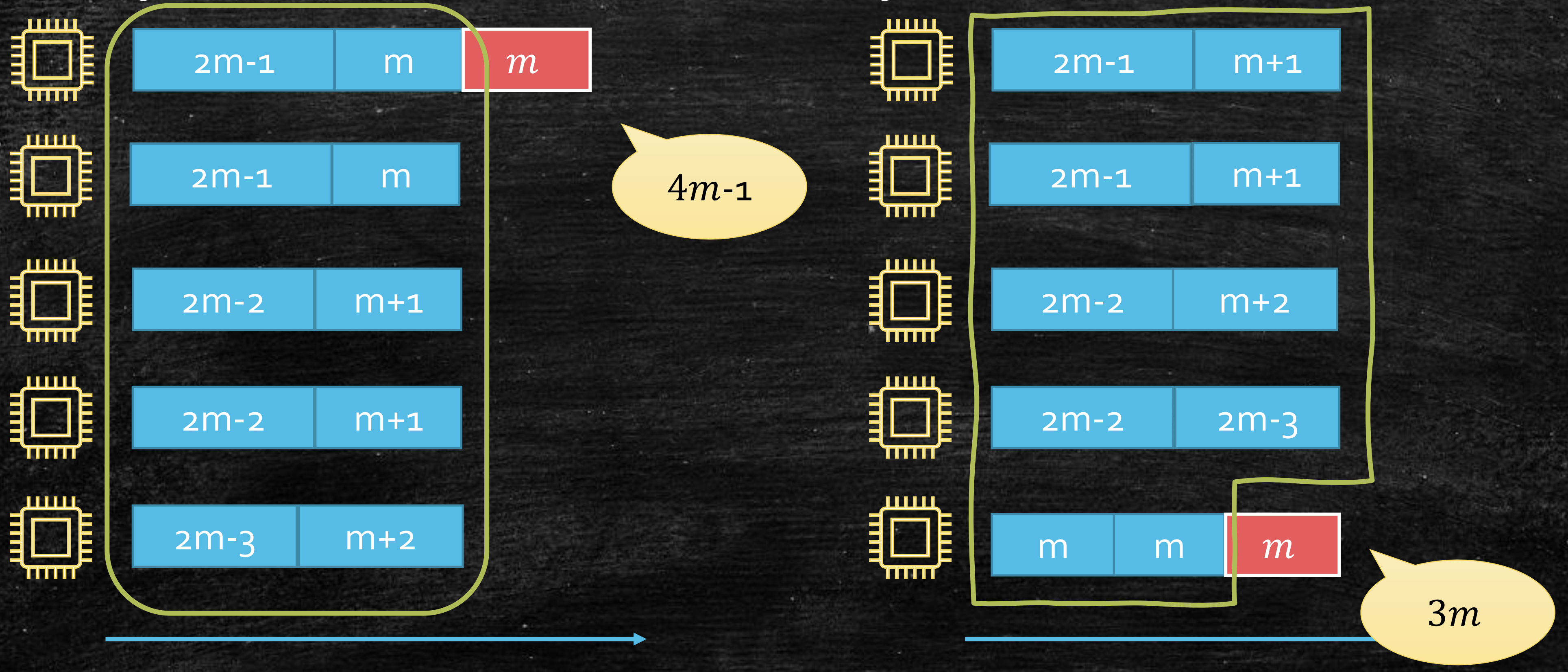
A counter example for LPT

- 2 jobs with size m+1~$2m-1$, 1 job with m.



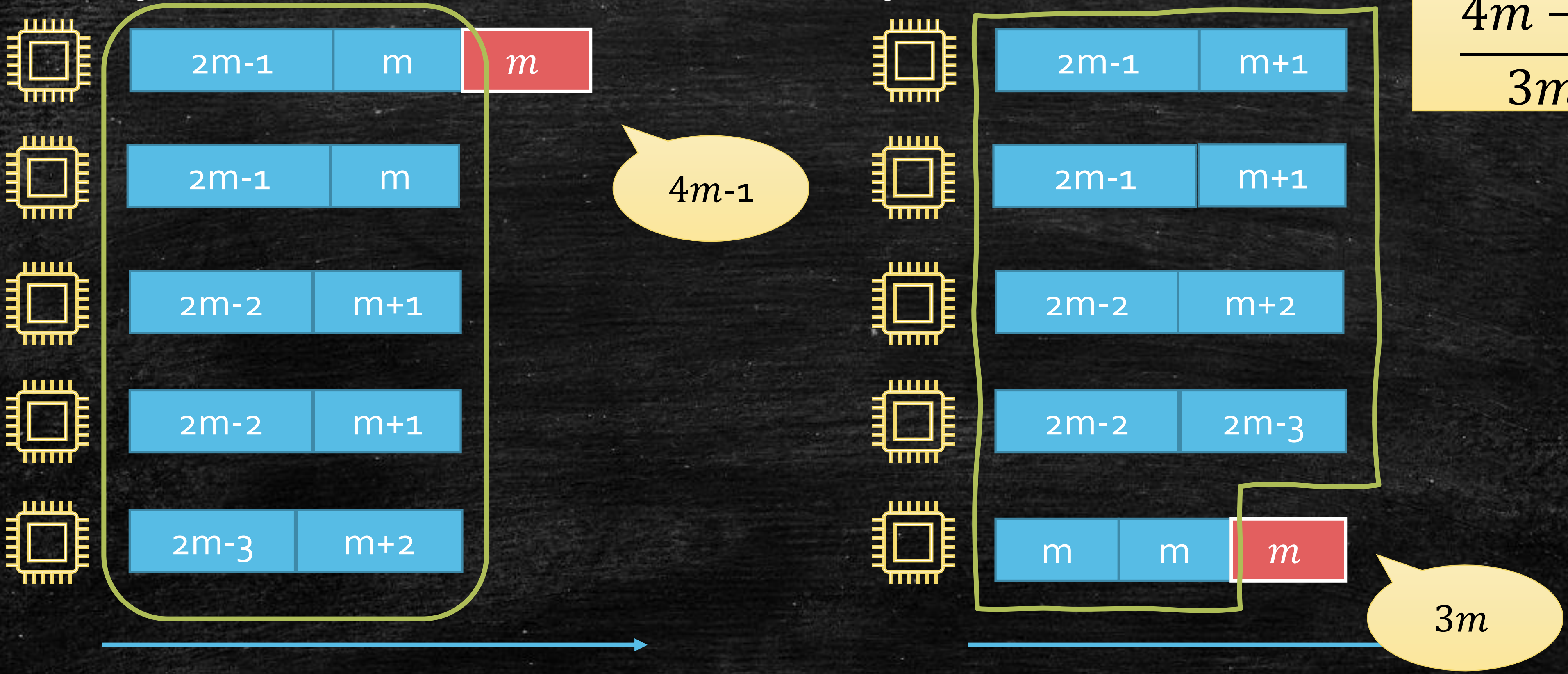
A counter example for LPT

- 2 jobs with size m+1~$2m-1$, 1 job with m.



A counter example for LPT

- 2 jobs with size m+1~$2m-1$, 1 job with m.

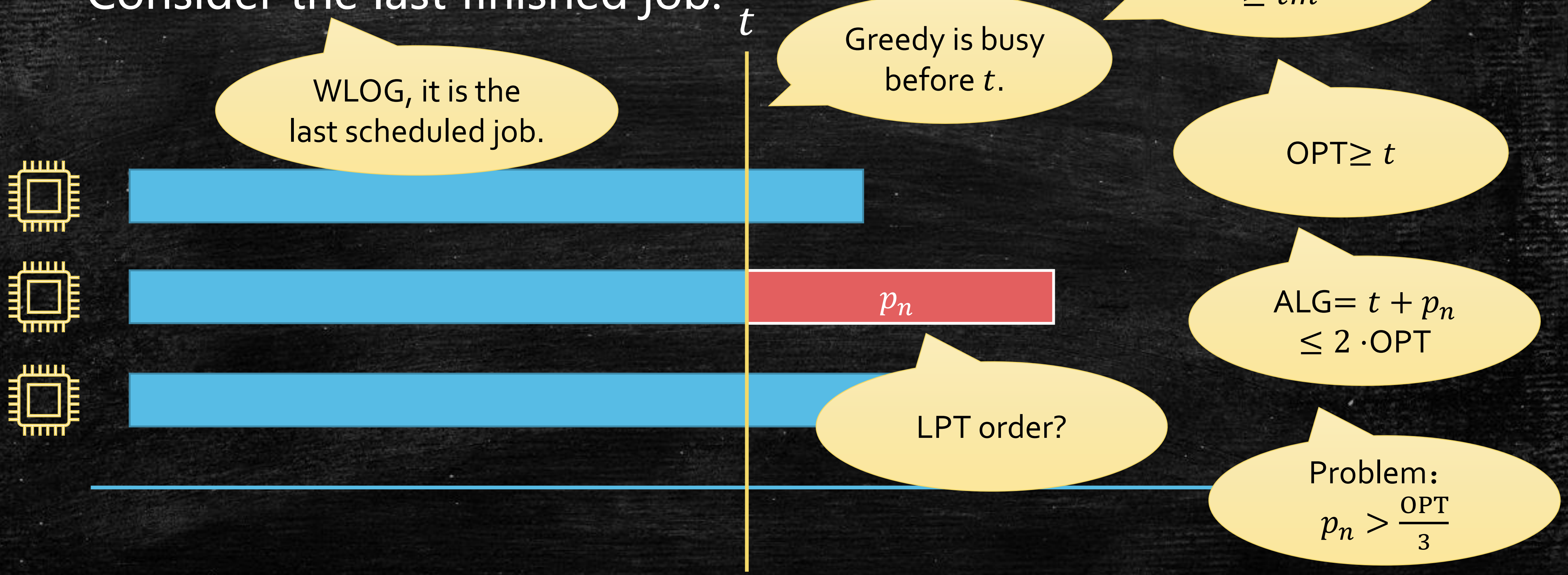


$$\frac{4m-1}{3m} \to 4/3$$

Can we improve the ratio to $4/3$?

Connect LPT Solution to OPT

- Schedule jobs to the earliest finished machine.
- Consider the last finished job.



Tips!

- Fact: we have n jobs larger than OPT/3
- Fact: OPT can have at most 2 jobs on one machine.
- Thinking: How to prove **LPT** $=$ **OPT** with the Facts.
- More Questions
  - How to make the 2-analysis of **Greedy** to $2 - \frac{1}{m}$?
  - How to make the $\frac{4}{3}-$analysis of **LPT** to $\frac{4}{3} - \frac{1}{3m}$?
  - Tips: they use the same technique.

Today’s goal

- Learn what is **Greedy**!
- Recap the difference of **Greedy** and **Divide and Conquer**.
- Learn to find the **problems** in a **Greedy attempt**.
- Learn to analyze Greedy Algorithm when it is **not optimal**.