Online List Labeling with Predictions

Shikha Singh

Williams College

Joint work with

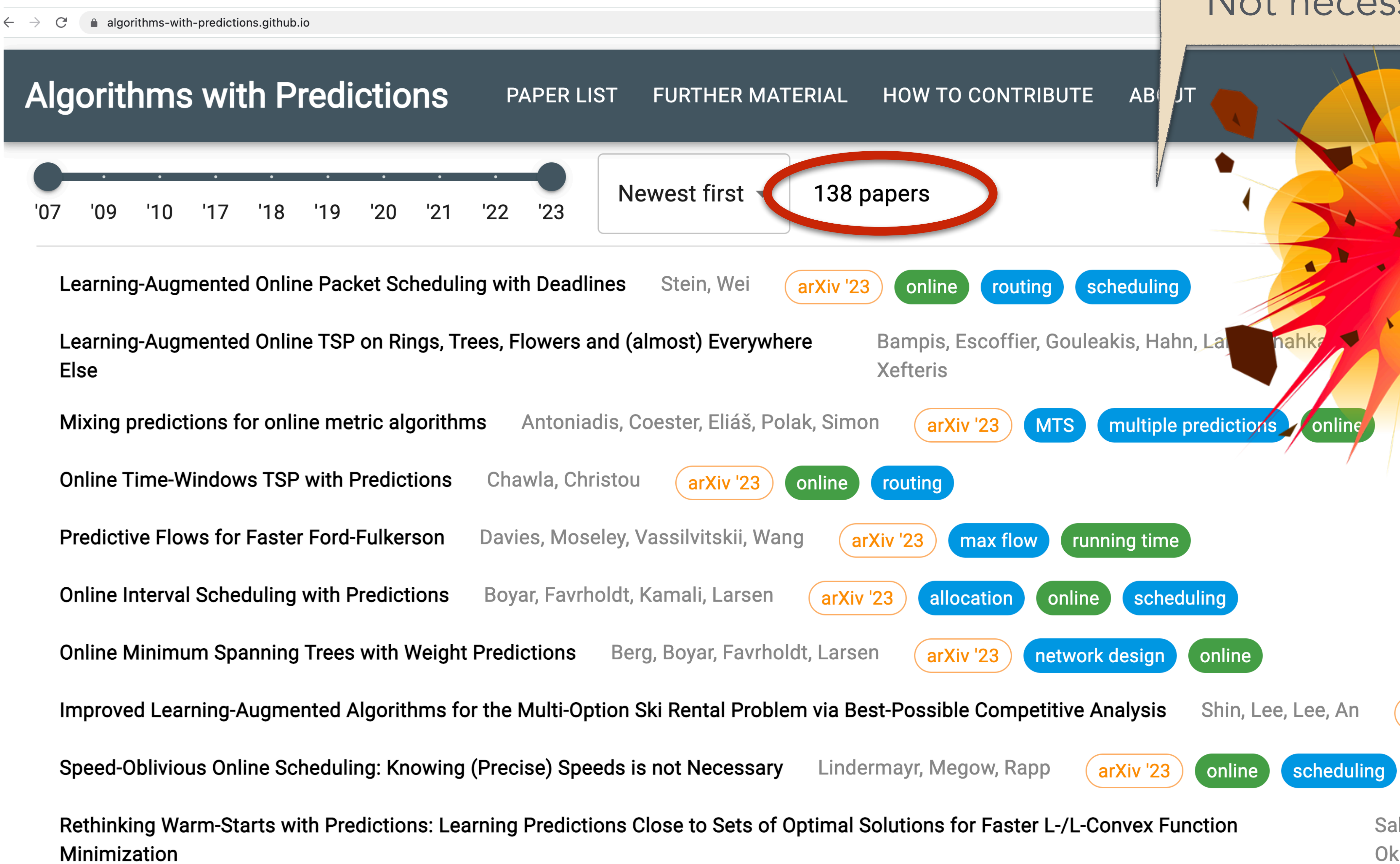
Samuel McCauley, Benjamin Moseley and Aidin Niaparast

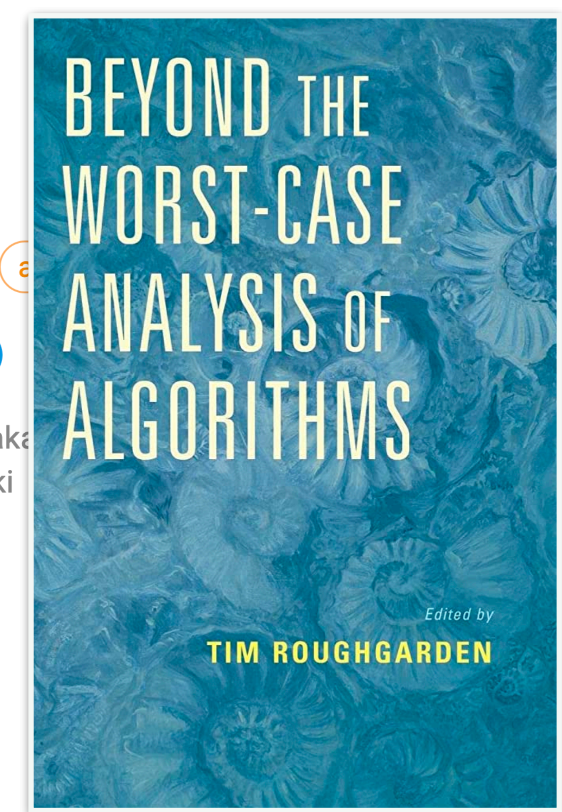
Algorithms with Predictions

- Worst-case is analysis often too pessimistic
- Growing line of work to analyze beyond-worst case performance of algorithms
- Focus on "instances we are more likely to see"
  - Future instances look like the past

Algorithms with Predictions

Not necessarily complete





Data Structures with Predictions

Not necessarily complete

Algorithms with Predictions PAPER LIST FURTHER MATERIAL HOW TO CONTRIBUTE ABOUT

'07 '09 '10 '17 '18 '19 '20 '21 '22 '23

Newest first 8 papers

On the Power of Learning-Augmented BSTs	Chen, Chen	arXiv '22, data structure, search
Learning-Augmented Binary Search Trees	Lin, Luo, Woodruff	arXiv '22, ICML '22, data structure, search
A learned approach to design compressed rank/select data structures	Boffa, Ferragina, Vinciguerra	ALENEX '21, TALG '22, data structure
On the performance of learned data structures	Ferragina, Lillo, Vinciguerra	Theor. Comput. Sci. '21, data structure
Repetition- and Linearity-Aware Rank/Select Dictionaries	Ferragina, Manzini, Vinciguerra	ISAAC '21, data structure
The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds	Ferragina, Vinciguerra	Proc. VLDB Endow. '20, data structure
Learned data structures	Ferragina, Vinciguerra	INNSBDDL '19, data structure
The Case for Learned Index Structures	Kraska, Beutel, Chi, Dean, Polyzotis	arXiv '17, SIGMOD Conference '18, data structure

Challenge: Initially no unified theoretical framework to reason about predictions

Learning-Augmented Model

- Introduced **recently** to give a general theoretical framework for analyzing learned algorithms
- **Design and Analysis Goal**:
    - Performance bounds as a function of **error in prediction**
    - Do well on both extremes (perfect and totally erroneous)
    - Degrade gracefully in between
- Essentially want low or no overhead of using predictions

Motivating Example [Kraska et al. SIGMOD '18]

- Binary search over a sorted array of n numbers
- Worst case: $O(\log n)$ time look up

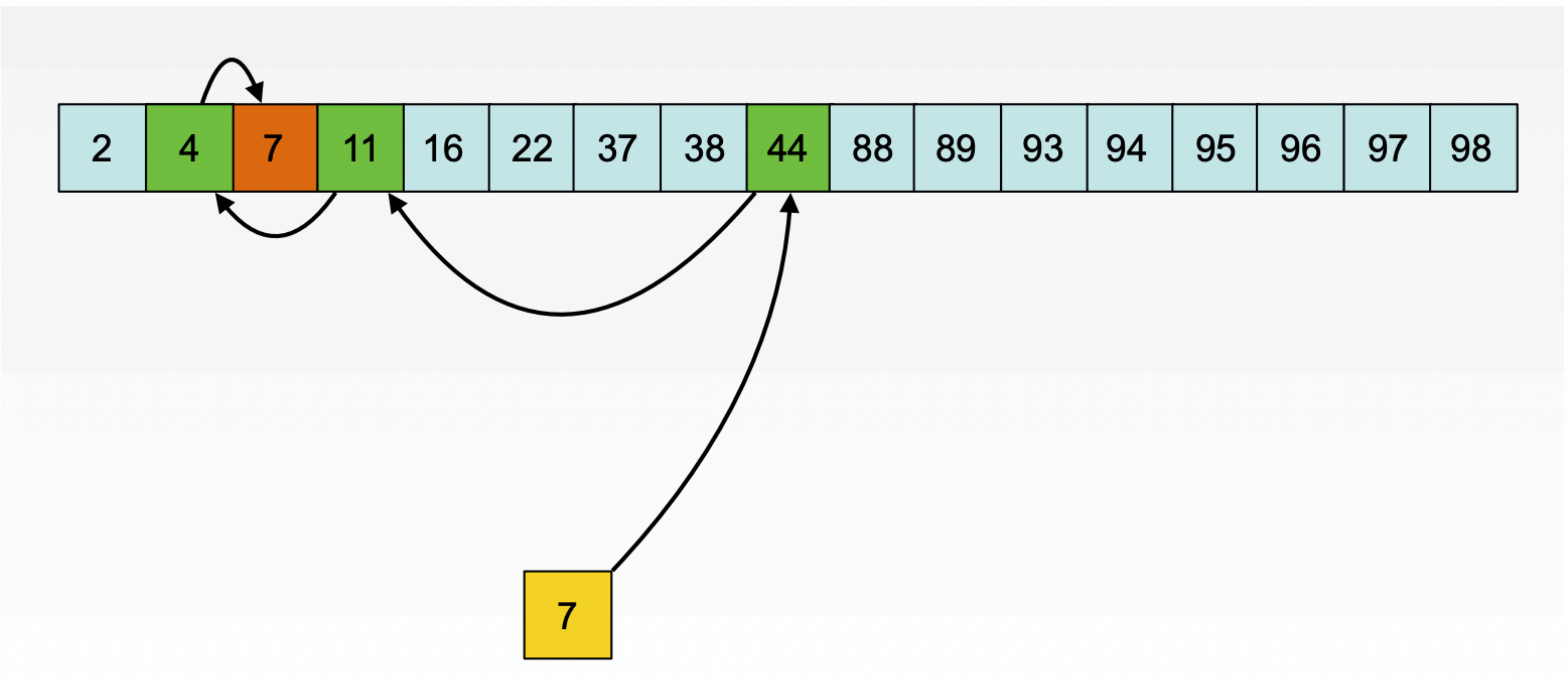


Figure credit: Ben Moseley

Motivating Example [Kraska et al. SIGMOD '18]

- Train a predictor $\tilde{r}(x)$ to predict x's location in array based on past data
  - $\tilde{r}(x)$ might be wrong, hopefully not too much
- **"Warm start"** your search at $\tilde{r}(x)$
  - Repeatedly double until you find x

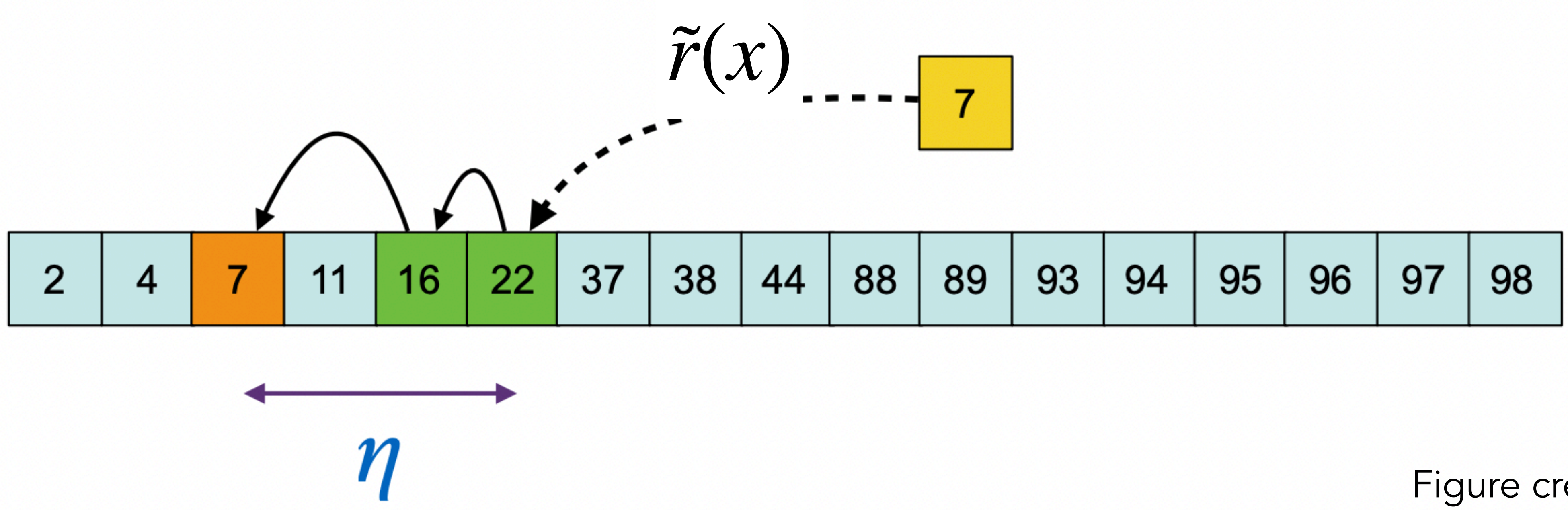


Figure credit: Ben Moseley

Motivating Example [Kraska et al. SIGMOD '18]

- Analysis: Define prediction error $\eta = |\tilde{r}(x) - r(x)|$
  - New lookup cost: $O(\log \eta)$
- **(Best).** Perfect prediction: $O(1)$ cost
- **(Worst).** Completely erroneous: $O(\log n)$
- **(Intermediate).** Degrades gracefully with error

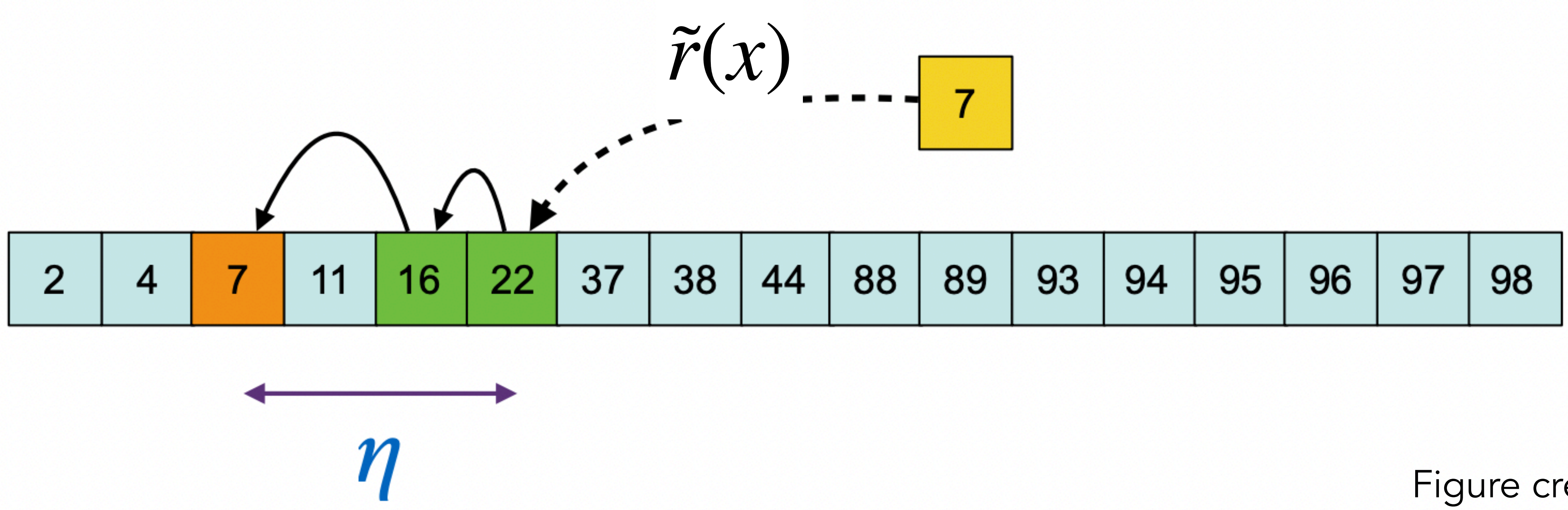
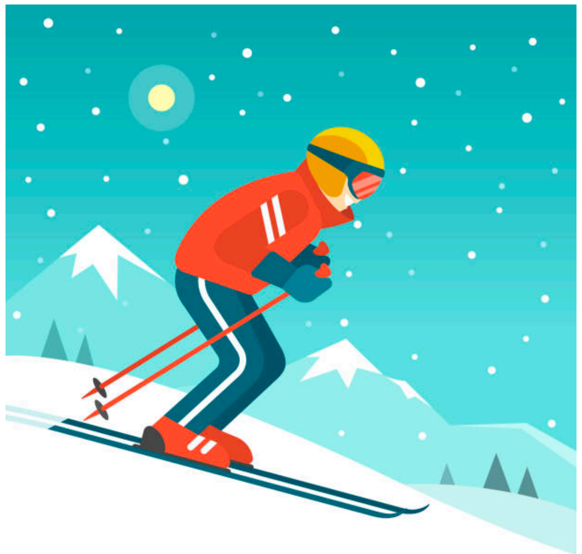
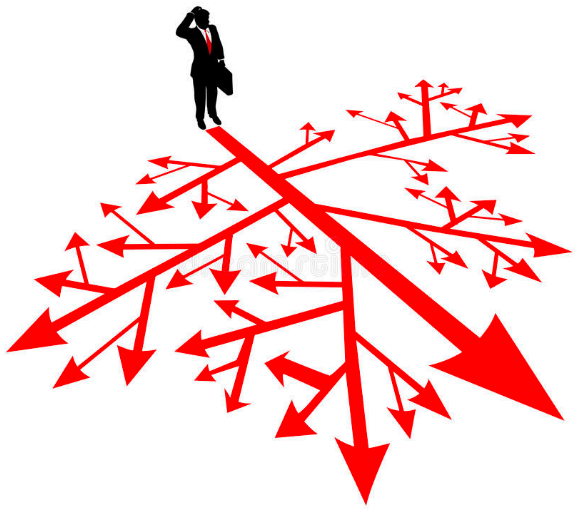


Figure credit: Ben Moseley

Problems Studied in this Model

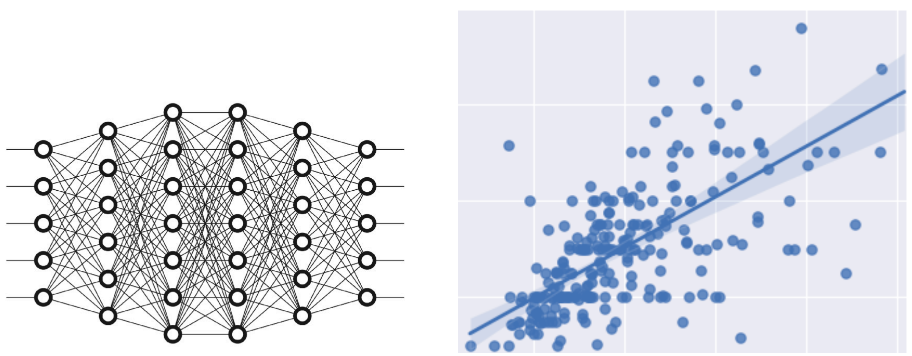
- Applied to **online algorithms** [Lavastida Moseley Vassilvitskii '20]
- Warm-starting offline optimization problems
  - Bipartite matchings [Dinitz Im Lavastida Moseley Vassilvitskii '21]
  - Shortest paths [Chen Silwal Vakilian Zhang '22]
  - Convex optimization [Sakaue Oki '22]
  - Flows [Davies Moseley Vassilvitskii Wang '23]

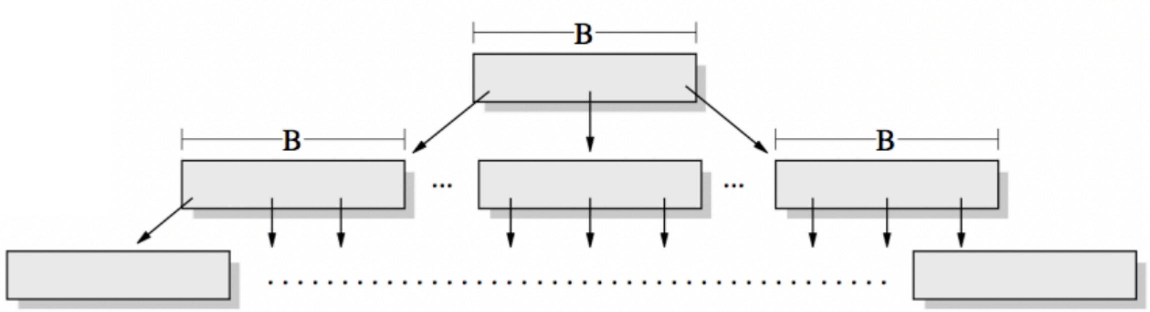




Learned Data Structures Literature

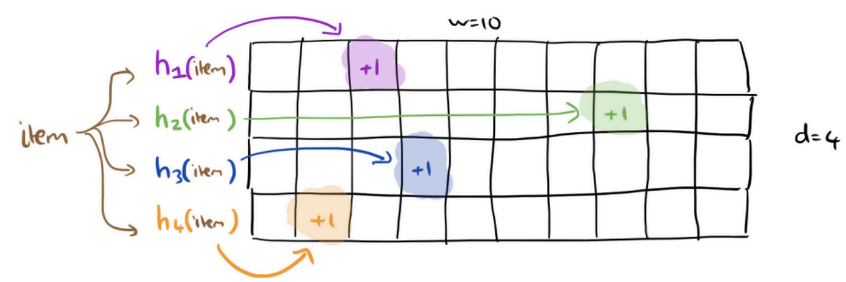
Learned Replacements of Data Structures



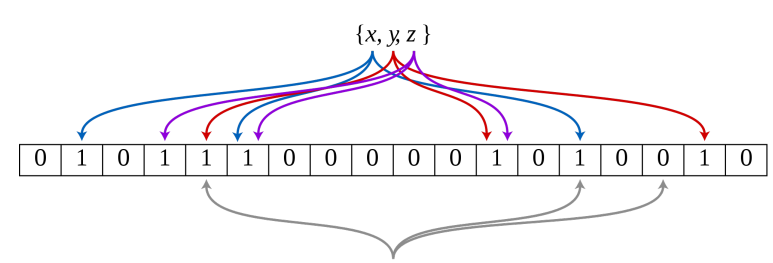


Learned indices:
[Kraska Beutel Chi Dean Polyzotis '18] & many follow ups, learned hashing [Ferragina Lehmann Sanders Vinciguerr '23], etc.

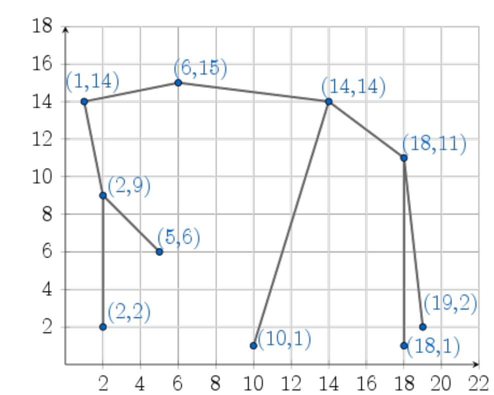
Learned Adaptations of Data Structures



Learned **count-min sketch**
[Hsu Indyk Katabi Vakilian '19]



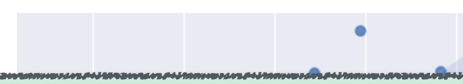
Learned **filters**
[Mitzenmacher '18]
& follow ups



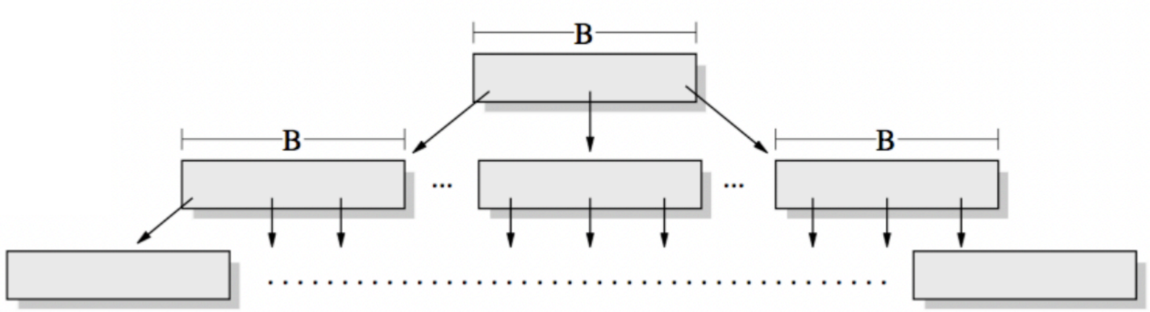
Learned **BSTs**
[Lin Luo Woodruff '22]

Learned Data Structures Literature

Learned **Replacements** of Data Structures



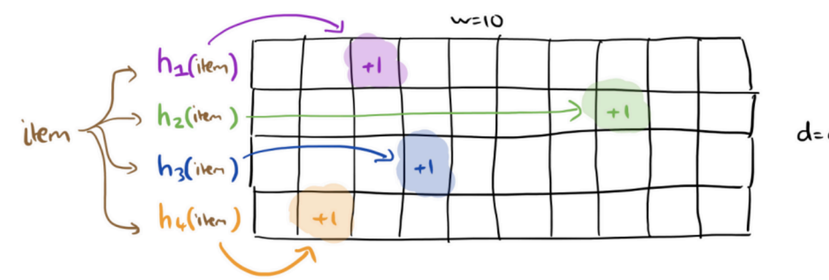
Exciting empirical results; limited analysis under assumptions



Learned indices:

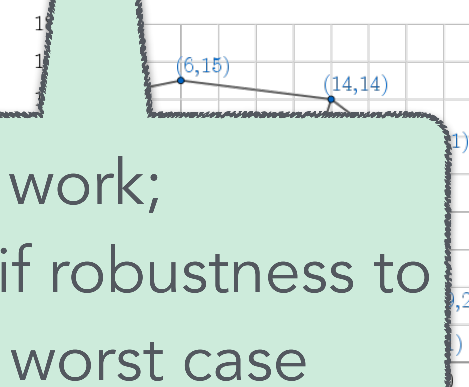
[Kraska Beutel Chi Dean Polyzotis '18] & many follow ups, learned hashing [Ferragina Lehmann Sanders Vinciguerr '23], etc.

Learned **Adaptations** of Data Structures

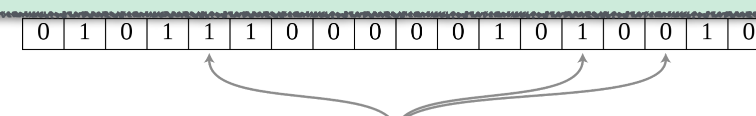


Learned **count-min sketch**

[Hsu Indyk Katabi Vakilian '19]



Exciting theoretical work;
Input distribution learned; if robustness to error present - reverts to worst case



Learned **filters**

[Mitzenmacher '18]

& follow ups

Learned **BSTs**

[Lin Luo Woodruff '22]

Our Goal

Apply the new **learning-augmented model** to data structures

Focus on a fundamental data structures problem:

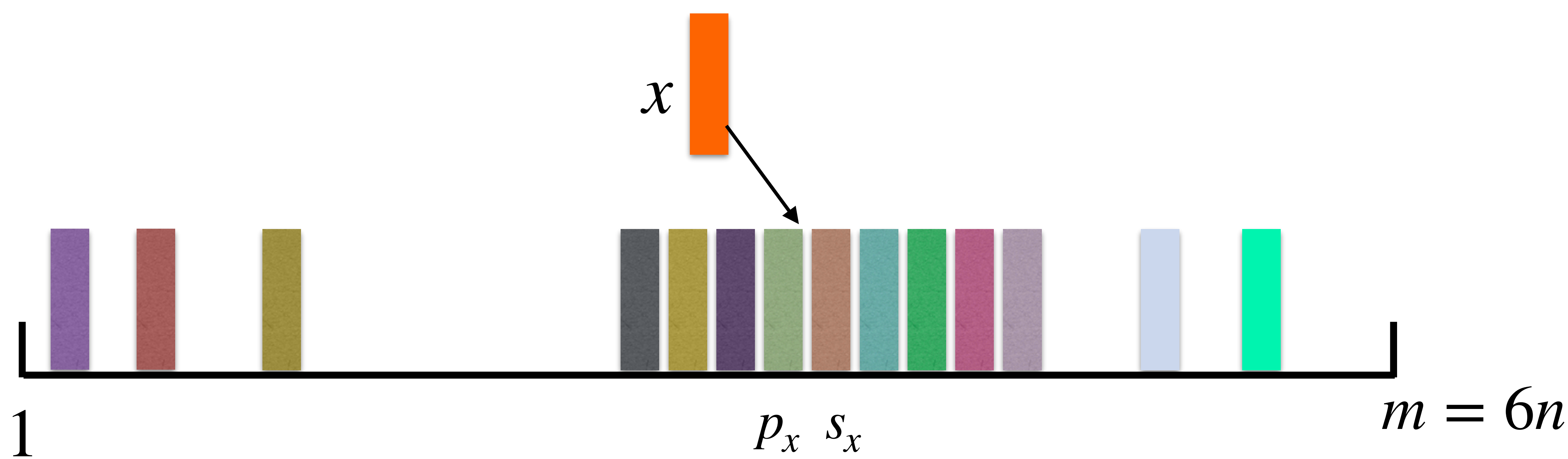
Online list labeling

Online List Labeling Problem

- n items arrive one by one (from a totally ordered universe)
- Must be stored in **sorted order** in an array of size $m = cn$
- Define label(x) as x's slot in array
- **Cost**: Minimize # relabels (**element movements**) per insert

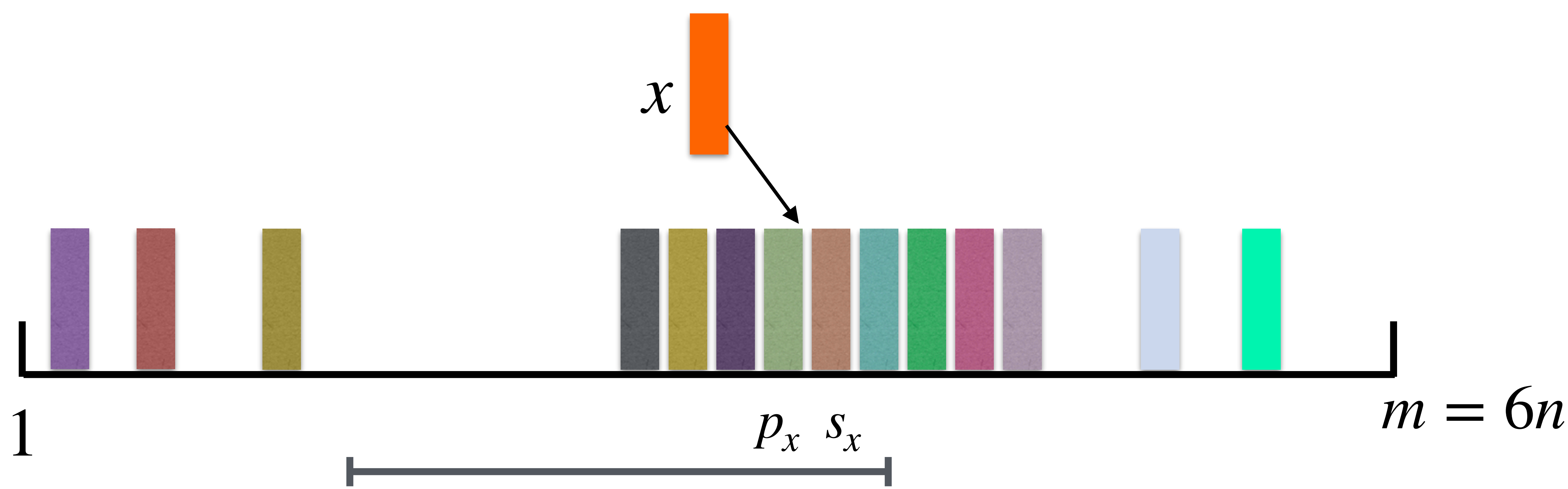
List Labeling Data Structure

- Insert(x) : must store it between pred p_x and succ s_x
- Might have to move things around to make room



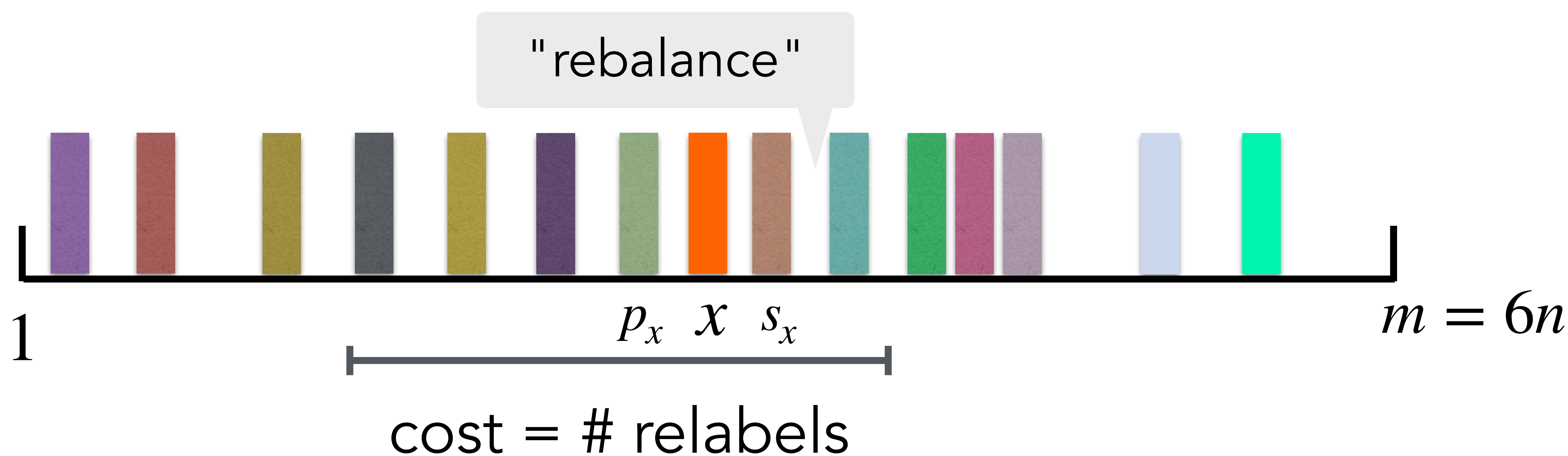
List Labeling Data Structure

- Insert(x) : must store it between pred p_x and succ s_x
- Might have to move things around to make room



List Labeling Data Structure

- Insert(x) : must store it between pred p_x and succ s_x
- Might have to move things around to make room
  - Must be careful: greedy approach $\Omega(n)$ per insert



Why List Labeling?

- Fundamental building block in many data structures
  - **Cache-oblivious B trees** [Bender Demaine Farach-Colton '00, Bender Demaine, Iacono Wu '02, Brodal Fagerberg Jacob '02] etc.
  - Graph data structures [Wheatman Burns '21, Wheatman Xu '18, Wheatman Xu '21, Pandey Wheatman Xu Buluc '21] etc.
- Studied for over four decades under various names
  - Sequential file maintenance [Willard '82, '86]
  - Order maintenance [Dietz '82, Dietz Slator '87]
  - Sparse tables [Itai, Konheim, Rodeh '81]
  - **Packed-memory arrays** [Bender, Demaine, Farach-Colton '00]
- We call any data structure for this problem **a list labeling array (LLA)**

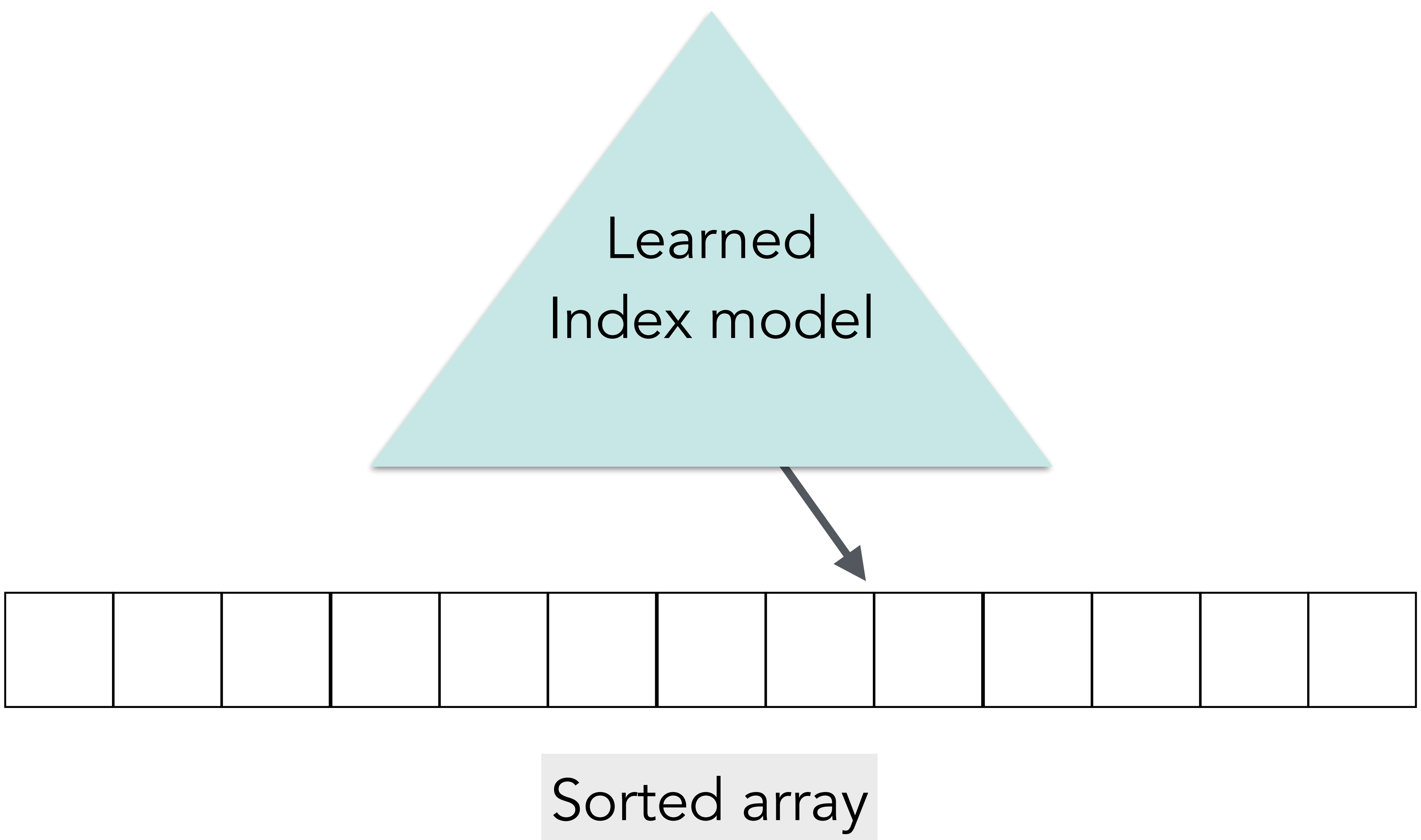
List Labeling: State of the Art

$\Omega(\log n)$ **lower bound** [Bulánek Koucky Saks '13]

- Deterministic LLAs:
  - $O(\log^2 n)$ amortized [Itai Konheim Rodeh '81] and worst-case LLA [Willard '82, '86], simplified by [Bender Cole Demaine Farach-Colton Zito '02], [Katriel '02], [Bender Fineman Gilbert Kopelowitz Montes '17]
  - Best possible for deterministic LLAs [Bulánek Koucky Saks '12]
- Randomized LLAs:
  - **Recent breakthrough**: $O(\log^{3/2} n)$ expected amortized [Bender Conway Farach-Colton Komlós Kuszmaul Wein '22] extends HI PMA [Bender Berry Johnson Kroeger McCauley Phillips Simon Singh Zage '16]
- Specialized LLAs:
  - Adaptive PMA [Bender Hu '07] and Rewired PMAs [DeLeo Boncz '19]

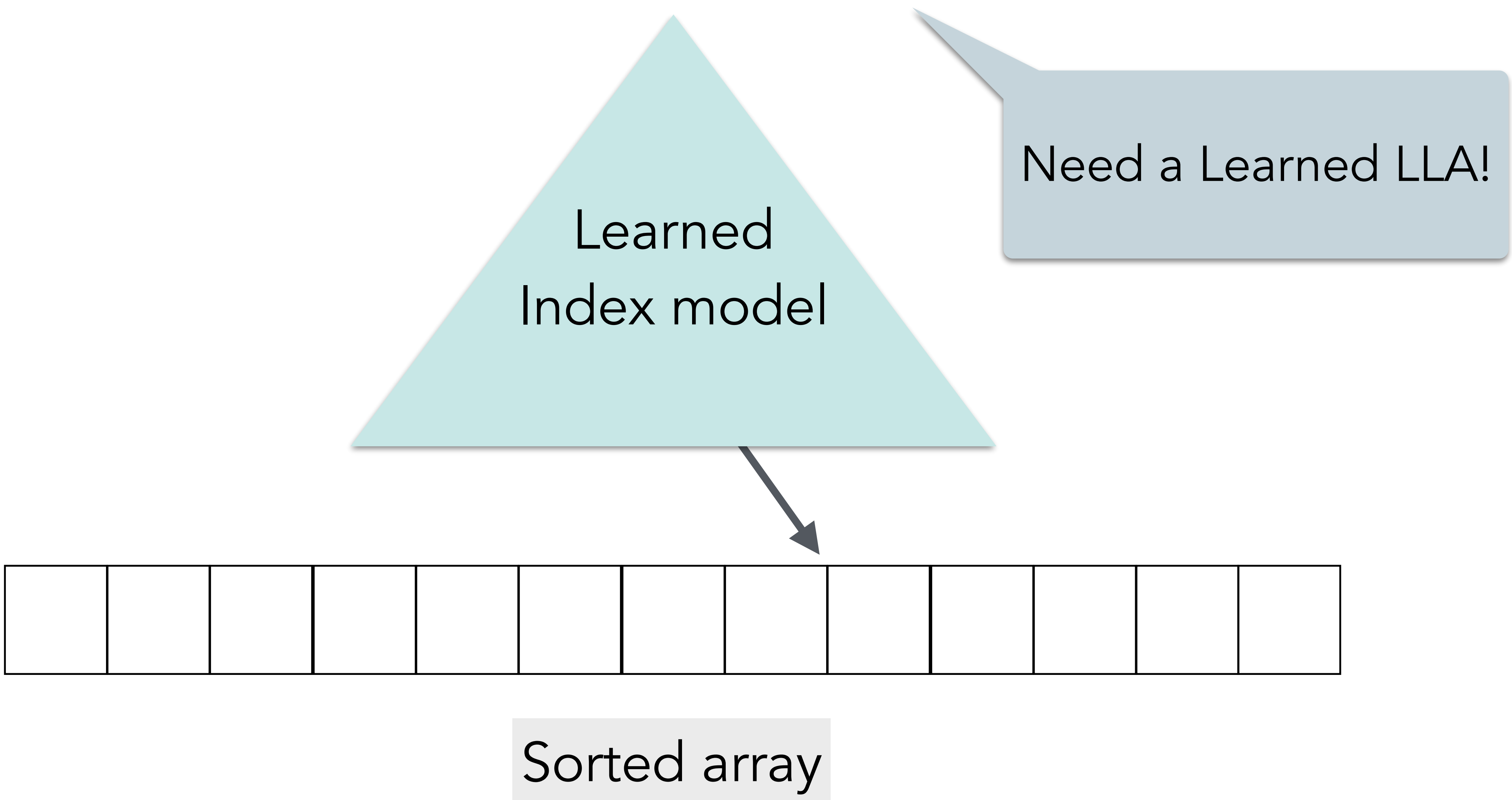
List Labeling in Learned Indices

- Directly motivated by work on learned indices
- Back to the original motivation from [Kraska et al. 2018]



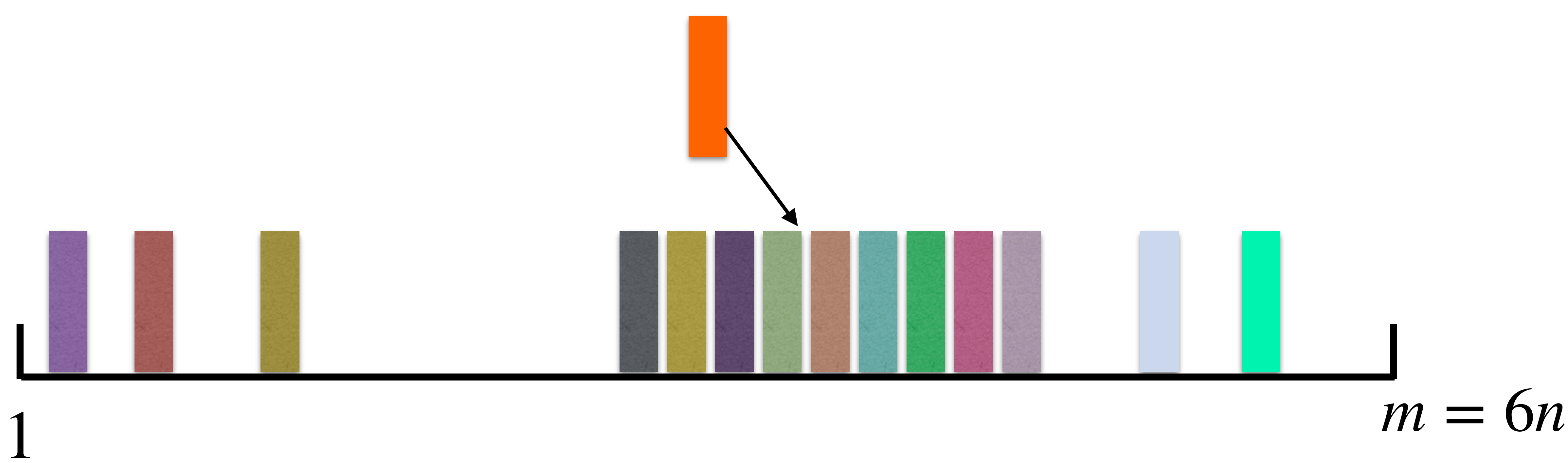
List Labeling in Learned Indices

- To support dynamic learned indices:
  - Need to efficiently maintain a dynamic sorted array!



Gapped Arrays

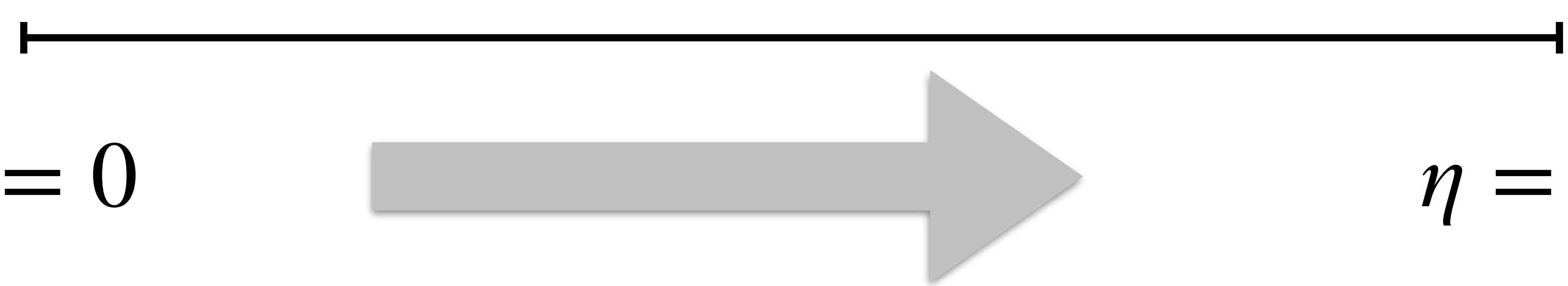
- Past work on learned indices used a **greedy** list labeling data structure: a gapped array [Ding et al. SIGMOD '20]
  - $\Omega(n)$ element movements per insert in worst case
- Assume uniform random insertions
  - $O(\log n)$ w.h.p. [Bender Farach-Colton Mosteiro '06]



Main Question

- How to leverage the **learning-augmented framework** to design a learned LLA that guarantees:
    - Best possible performance on extremes: best & worst predictions
    - Performance degrades gracefully with error

$\eta = 0$

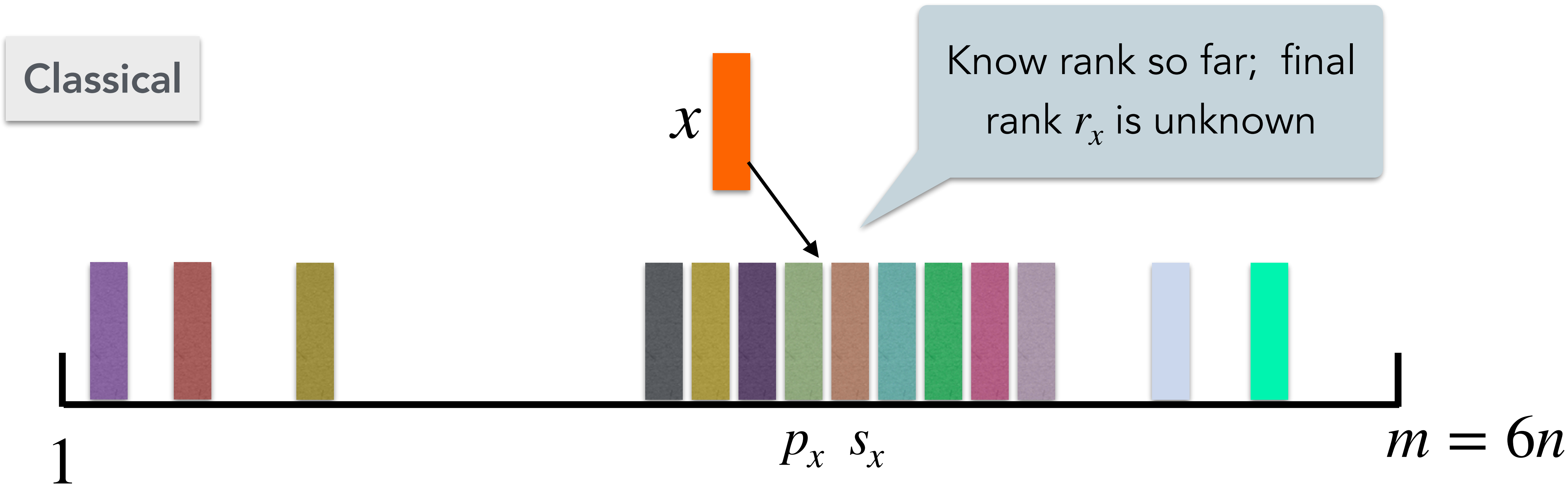


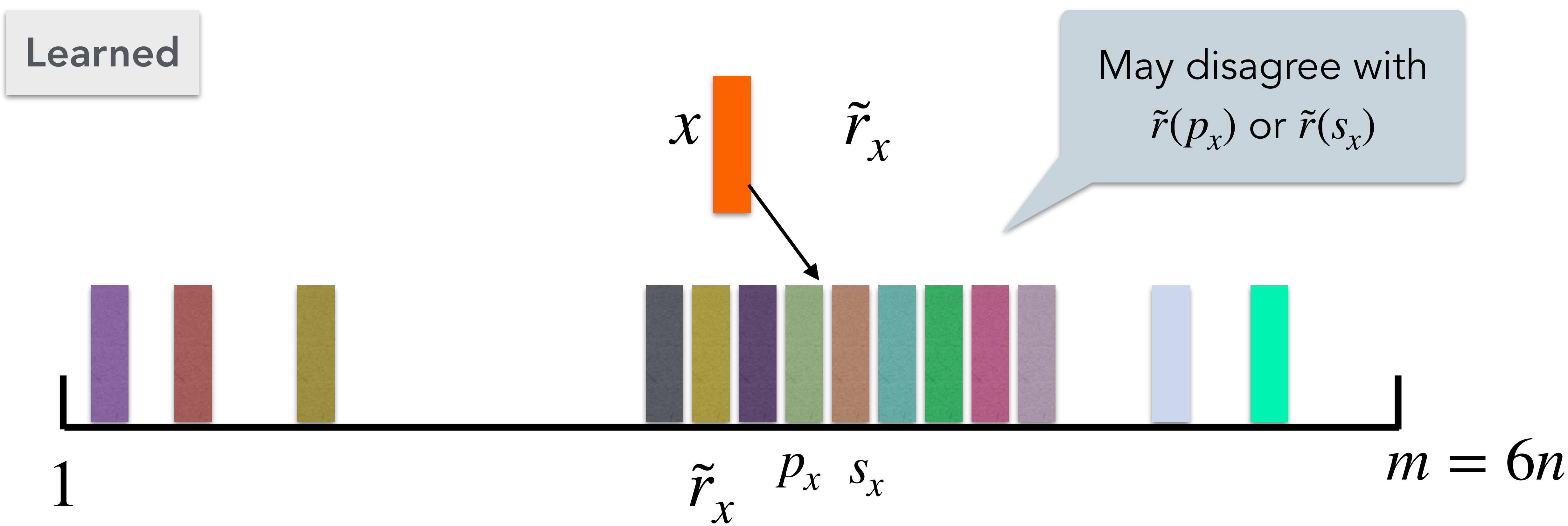
$\eta = \infty$

List Labeling Prediction Model

- n elements arrive one by one
  - For simplicity, ignore deletes for now
- **Final rank** of element x is $r(x)$ after all n elements arrive
- Each insert x arrives with a **predicted rank** $\tilde{r}(x)$
  - Assigned adversarially based on past inserts/predictions
- **Prediction error** $\eta_x = |r(x) - \tilde{r}(x)|$
- **Maximum error** as $\eta = \max \eta_x$

List Labeling with Predictions



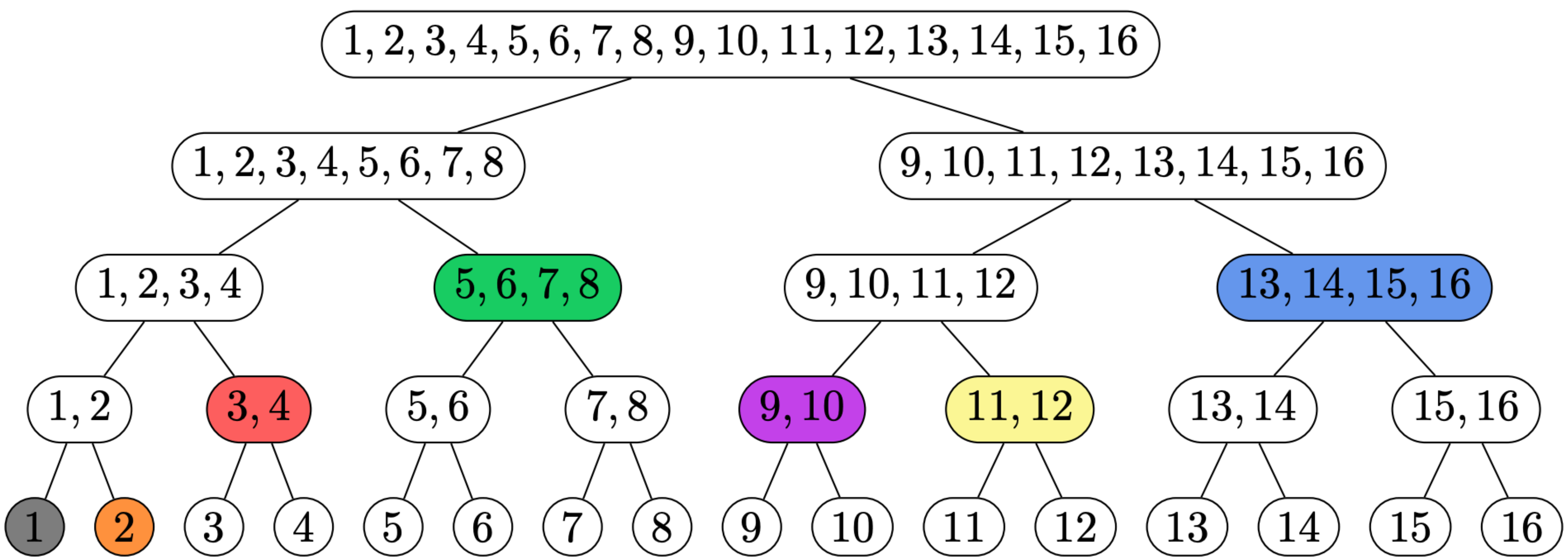


Our Results

- **[Today's talk]** A Learned List Labeling Array that
  - Uses existing worst-case LLAs as a blackbox
  - Guarantees $C(\eta)$ amortized cost where $C(n)$ is the amortized cost of black-box LLA
  - **Optimal** for **any error** η among deterministic LLAs
  - Empirically outperforms state-of-the-art LLAs
- **[Aside]** Stochastic predictions
  - Improved bounds in terms of mean and variance of unknown distribution from which error is sampled

learnedLLA: Description

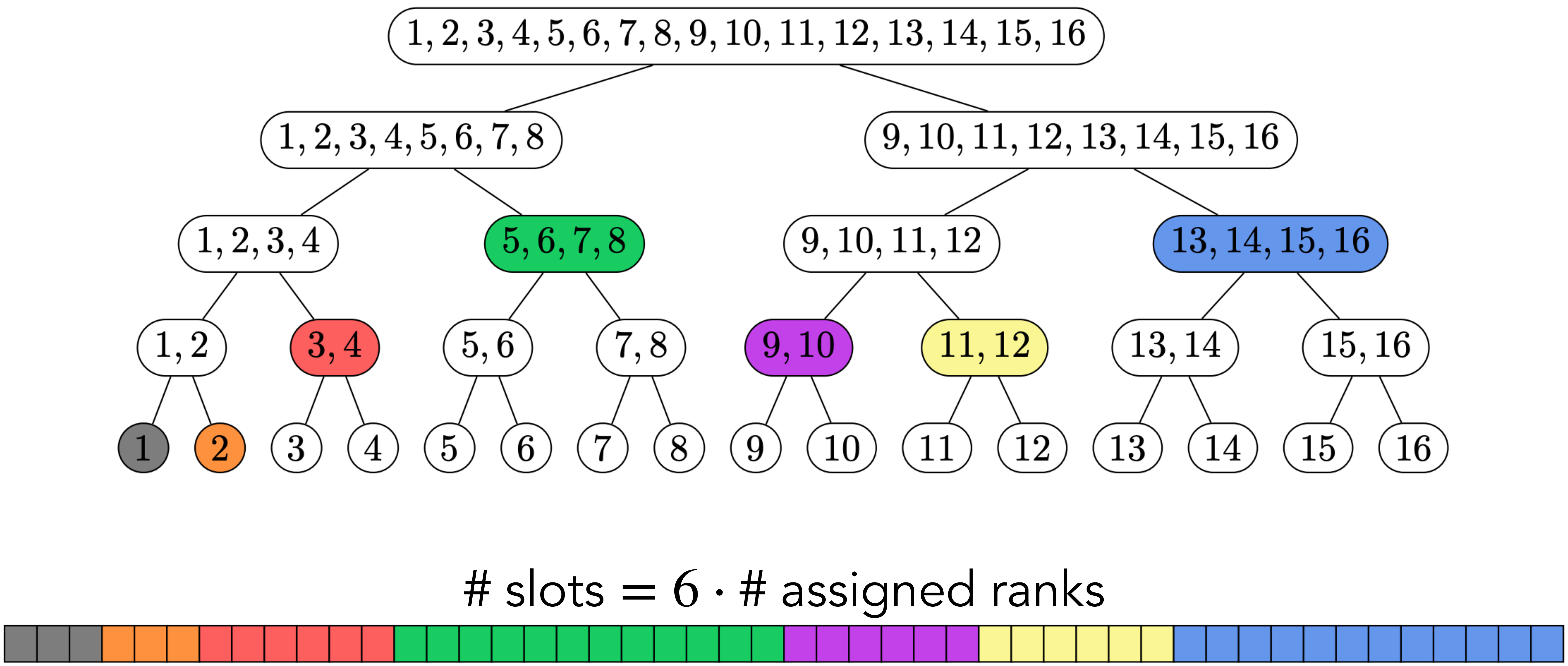
- At any time, partitioned into ℓ **actual LLAs** $P_1, \ldots, P_\ell$
- Each LLA is **assigned** contiguous **ranks** and **slots** that partition $\{1,\ldots,n\}$ and $\{1,\ldots,m\}$ respectively



\# slots = 6 · # assigned ranks

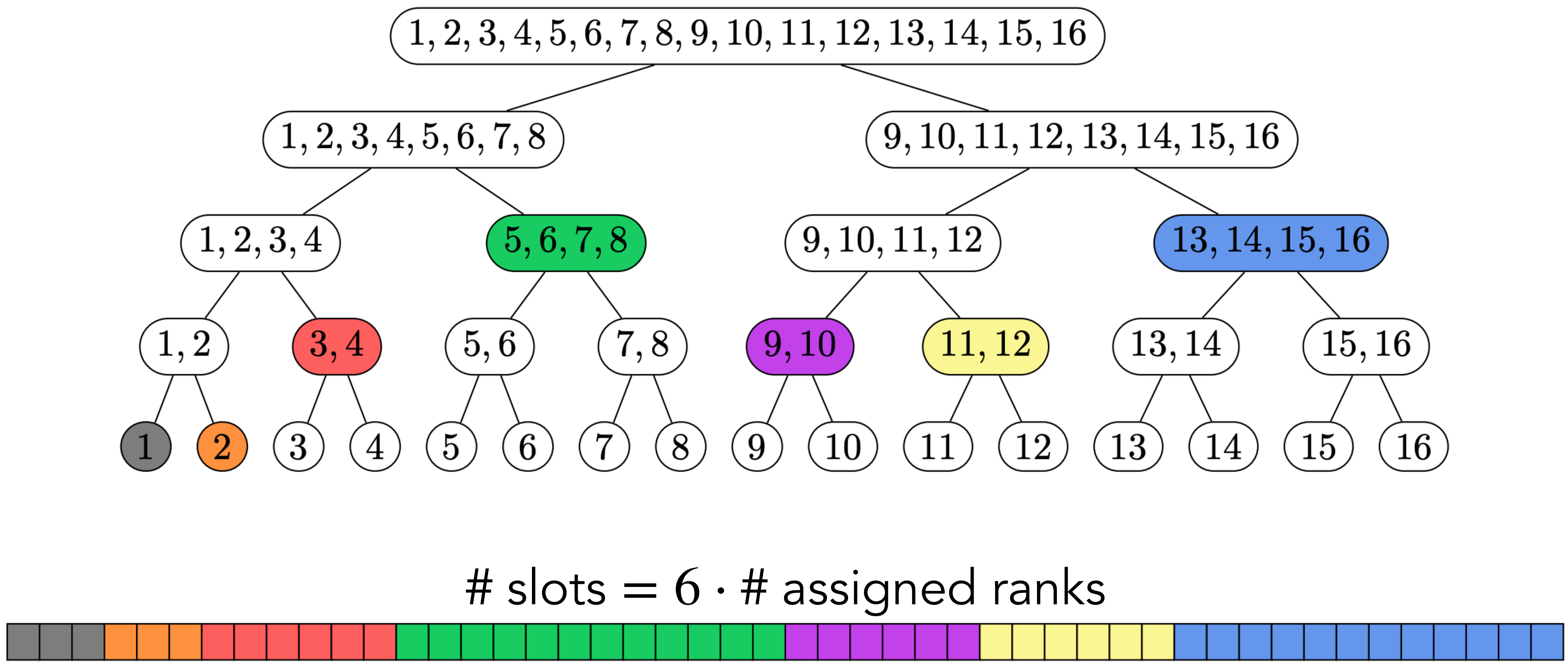
learnedLLA: Insert Idea

- If new insert's predicted rank **agrees with** pred and succ placement, insert into LLA containing predicted rank
- If it **conflicts** with pred (succ), insert into the LLA of pred/succ



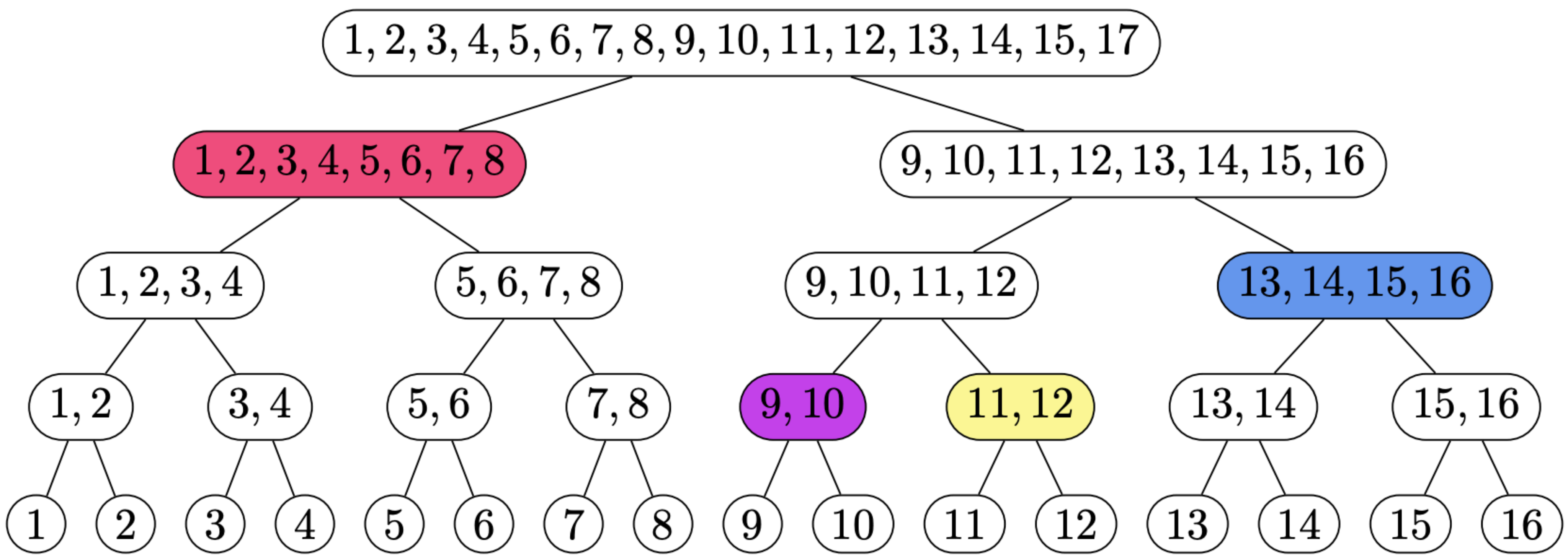
learnedLLA: Example Insert

- $\tilde{r}(x)$ is ***assigned to*** red LLA, but pred(x) is ***stored in*** green LLA
  - Insert x to green LLA
- If green LLA more than half full, merge with grey, orange, red



learnedLLA: Example Insert

- $\tilde{r}(x)$ is ***assigned to*** red LLA, but pred(x) is ***stored in*** green LLA
    - Insert x to green LLA
- If green LLA more than half full, merge with grey, orange, red



slots = $6 \cdot$ # assigned ranks

learnedLLA: Insertion

- $i_x \leftarrow$ LLA whose assigned ranks contain $\tilde{r}_x$
- i_p (resp i_s) $\leftarrow$ LLA whose assigned ranks contain p_x (resp s_x)

Insertion:

- If $i_p > i_x$: insert x into i_p
- Else if $i_s < i_x$: insert x into i_s
- Else: insert x into i_x
- If LLA inserted to more than half full
    - Merge with "sibling" LLA

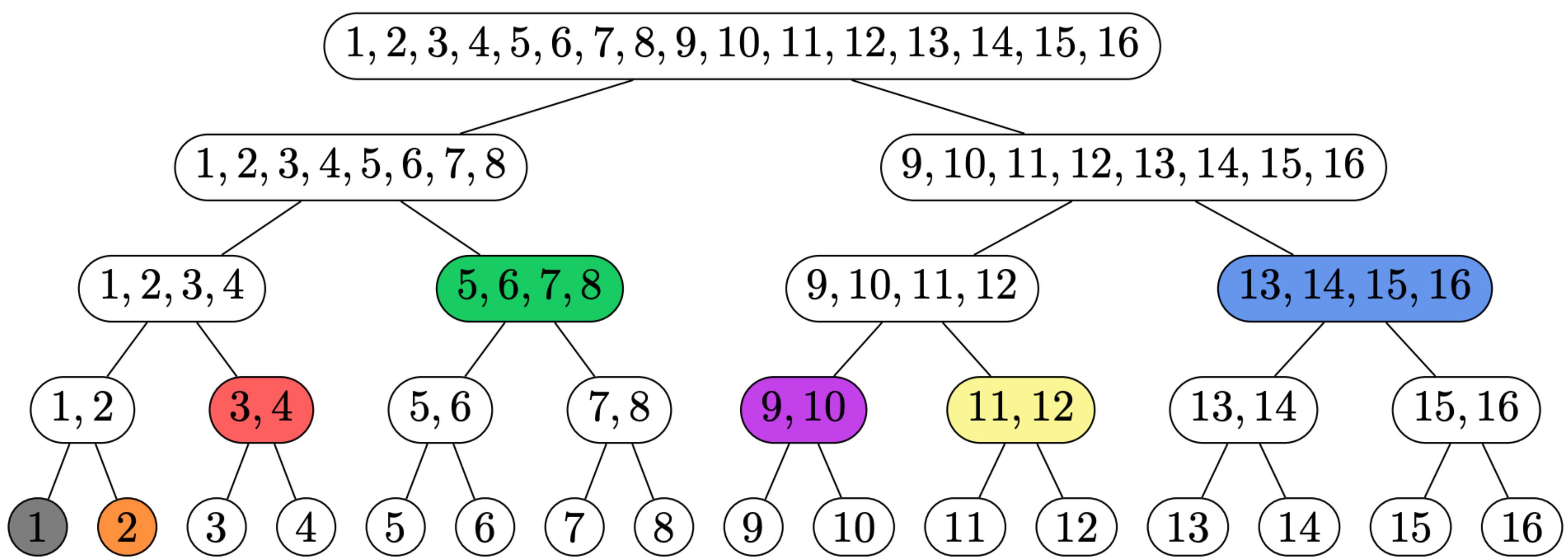
Let **blackbox LLA** handle actual slot within

Only case that uses the **predicted rank**

Idea: Some element must have error $\propto$ size of overfull LLA

learnedLLA: Analysis Idea

- **Key lemma:** If P is an actual LLA, then it contains some element x with high enough error: $\eta_x \geq |P|/2$
  - 3x elements as # assigned ranks
  - Some elem responsible for pushing others to this LLA

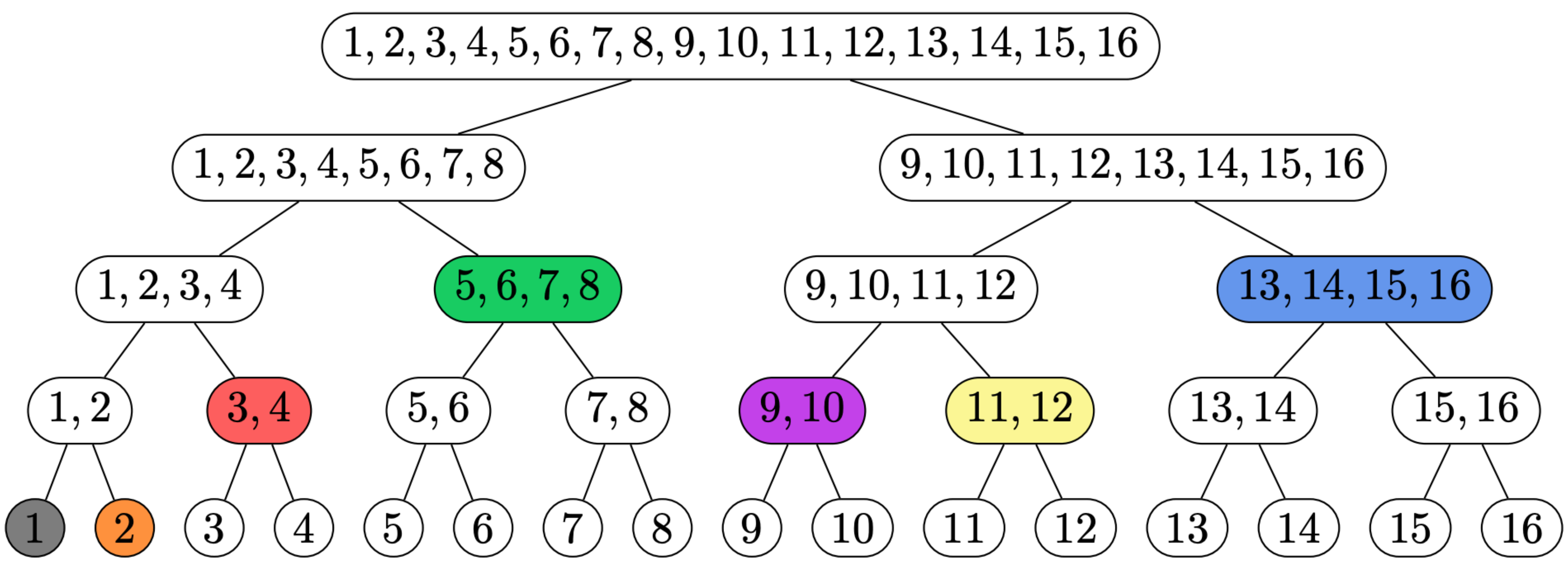


slots = 6 · # assigned ranks

learnedLLA: Analysis Idea

- **Key lemma:** If P is an actual LLA, then it contains some element x with high enough error: $\eta_x \geq |P|/2$
  - 3x elements as # assigned ranks
  - Responsible for pushing other

Largest LLA size = $O(\eta)$



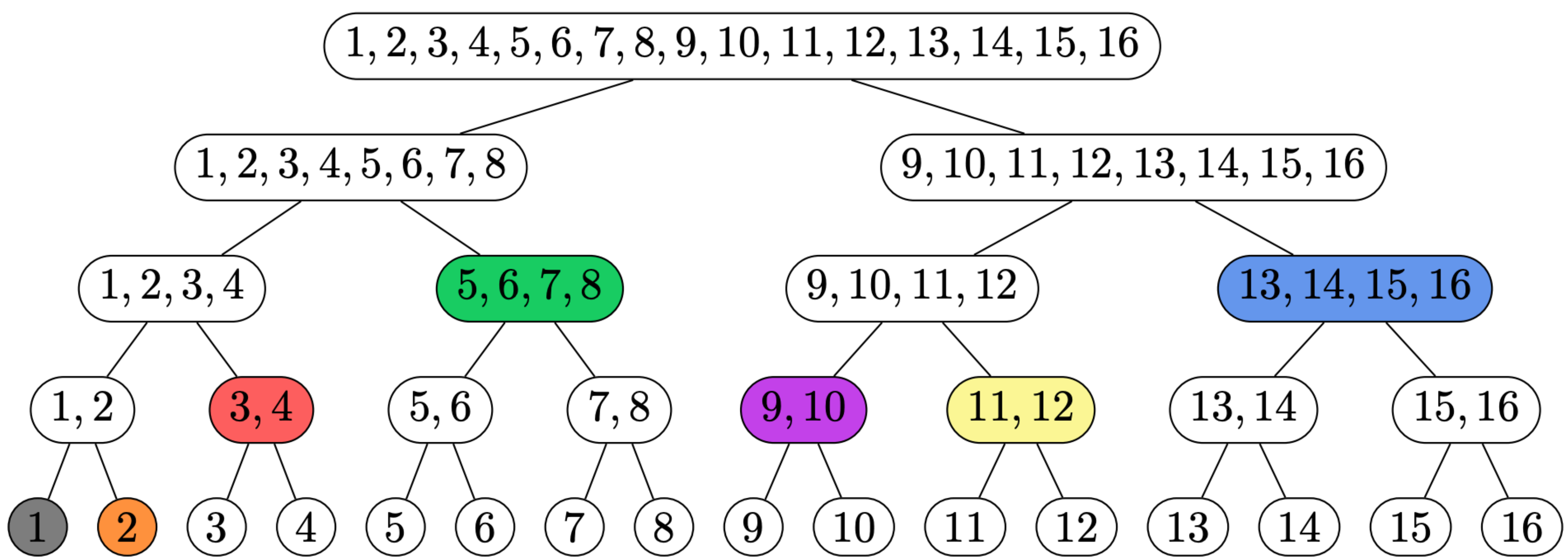
slots = 6 · # assigned ranks

learnedLLA: Analysis Idea

- **Total cost** = Relabels within LLAs + Relabels during merges

Dominated by cost of final LLAs

Linear cost; lower order term

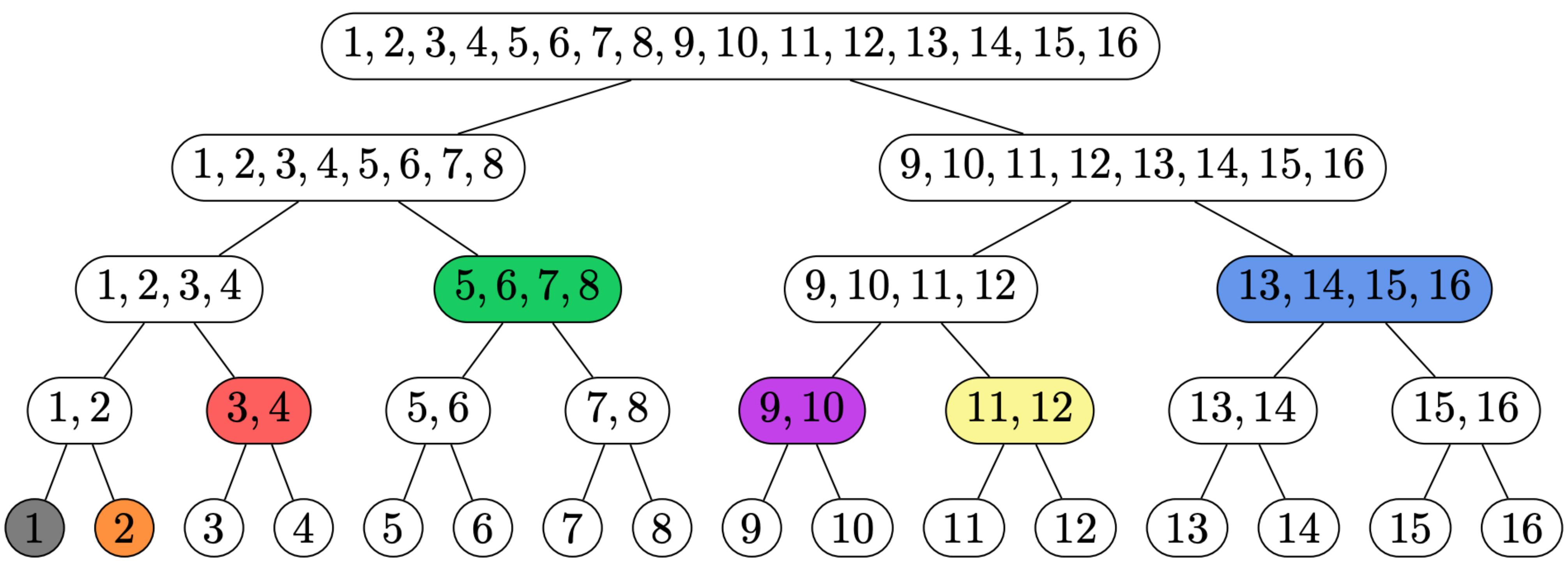


slots = 6 · # assigned ranks

learnedLLA: Analysis Idea

- Total cost = Relabels within LLAs + Relabels during merges

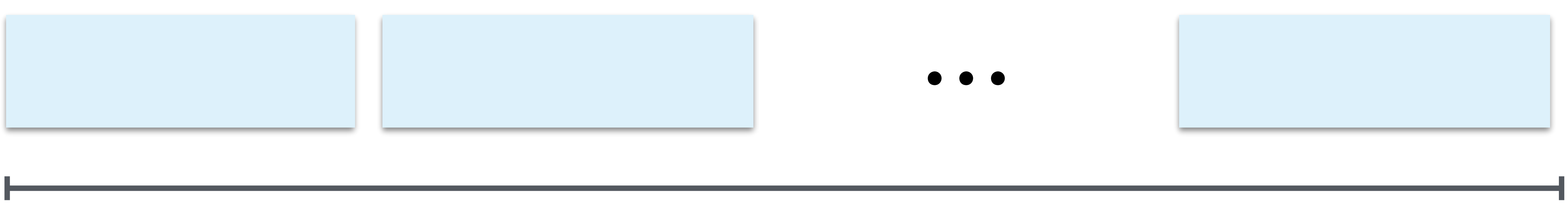
At most $C(\eta)$ amortized cost of each final LLA; all n partitioned across final LLAs



slots = 6 · # assigned ranks

Lower Bound Idea

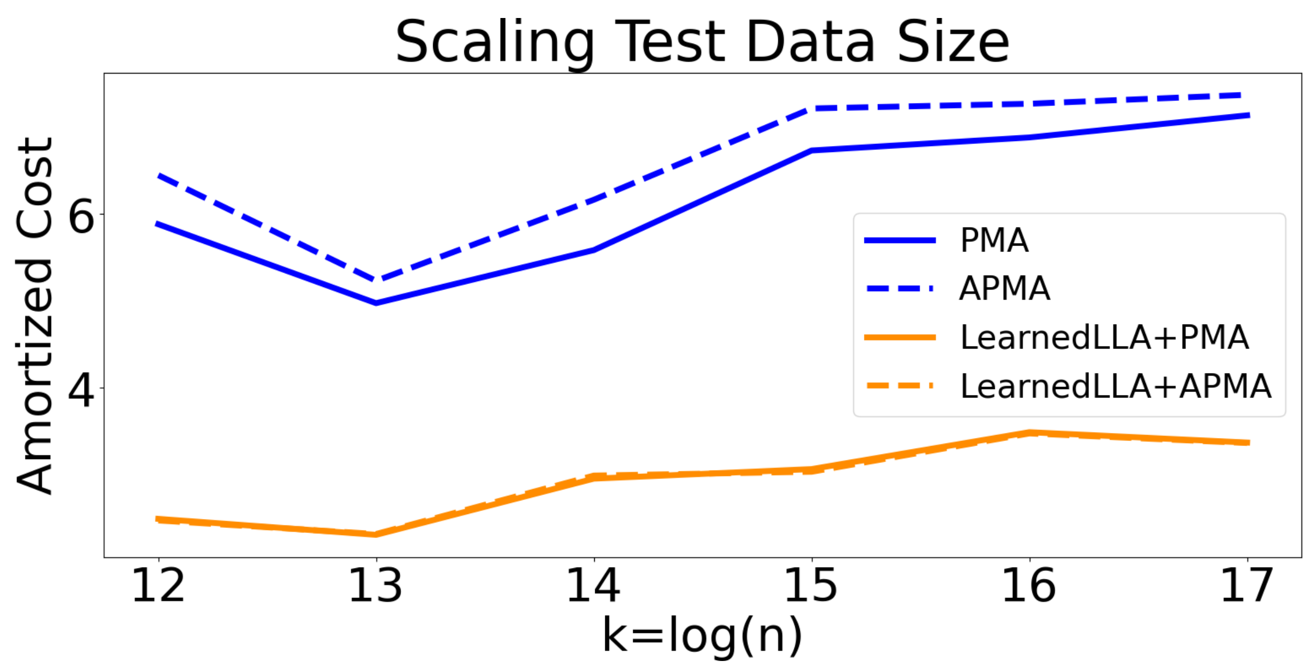
- Easier to see that can't do better when $\eta = 0$ or $\eta = n$
- **Question.** How to prove optimality for intermediate error?
- **Idea.** Apply classic lower bound [Bulánek Koucky Saks '12] to each n/η subproblems of size η
- **Challenge.** Can't force big LLA to only use assigned slots



n/η subproblems

Experiments

- LearnedLLA outperforms PMA, APMA on numerous real data
- Inherits performance of APMA when using it as blackbox



Amortized cost

LLAs	Gowalla (LocationID)	Gowalla (Latitude)	MOOC	AskUbuntu	email-Eu-core
PMA	7.14	14.56	19.22	24.56	21.49
APMA	7.38	15.63	16.70	10.84	21.43
LearnedLLA + PMA	**3.36**	**6.06**	**11.99**	14.27	**16.55**
LearnedLLA + APMA	**3.36**	6.15	12.13	**8.49**	**16.55**

Conclusion

- Learning-augmented framework provides a **"worst case"** way to reason about learned algorithms
- Only been applied to online algorithms/optimization problems
- Online list labeling structure was very amenable to this model
- **Exciting future direction**: opportunity to exploit this model for other data structures
  - **Main challenge:** what to predict and what not to predict
    - E.g. how to handle insert + query workloads?
- **Open problem from earlier**: How to tackle average case error?