Online List Labeling: Breaking the $\log^2 n$ Barrier

To Appear in FOCS 2022

Michael A. Bender
Stony Brook University

Alex Conway
VMWare Research

Martín Farach-Colton
Rutgers University

Hanna Komlós
Rutgers University

William Kuszmaul
MIT

Nicole Wein
DIMACS

The List Labeling Problem

Maintain n elements in **sorted order** in an array of size m, $m \geq n(1 + \Theta(1))$

Goal: minimize # elements moved per insertion/deletion (**cost**)

List Labeling — Library Analogy

A librarian wants to store books alphabetically on a shelf

No gaps between books $\Longrightarrow$ many books move after each insertion

Library with no gaps

Library with gaps

Formulations and (Re)Discoveries

Priority queues
[Itai, Konheim, Rodeh, ICALP'81]

File maintenance
[Willard, STOC'82, SIGMOD'86]

Order maintenance
[Dietz, STOC'82] [Dietz, Sleator, STOC'87]

Balanced binary trees & scapegoat trees
[Andersson, WADS'89] [Andersson, Lai, SWAT'90] [Galperin, Rivest, SODA'93]

Locality preserving dictionaries
[Raman, PODS'99]

Sample Applications

Cache-oblivious B-trees

[Bender, Demaine, Farach-Colton, FOCS'00] [Bender, Demaine, Iacono, Wu, SODA'02] [Brodal, Fagerberg, Jacob, SODA'02] [Bender, Fineman, Gilbert, Kuszmaul, SPAA'16]

Multi-dimensional searching

[Nekrich, Algorithmica'07] [Nekrich, Comp Geom'09] [Mortensen, SODA'03] [Toss, Pahins, Raffin, Comba Computers & Graphics'18]

Packed-memory arrays and density-control arrays

[Hofri, Konheim, SICOMP'87] [Katriel, '02] [Bender, Demaine, Farach-Colton, FOCS'00] [Bender, Hu, PODS'06] [Itai, Katriel, '07] [De Leo, Boncz, ICDE'19] [Bender, Fineman, Gilbert, Kopelowitz, Montes, SODA'17] [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage, PODS'16]

Particle simulation

[Durand, Raffin, Faure, RIPHYS'12]

Database join algorithms

[Khayyat, Lucia, Singh, Ouzzani, Papotti, Quiané-Ruiz, Tang, Kalnis, VLDB'17]

Managing dynamic sparse graphs

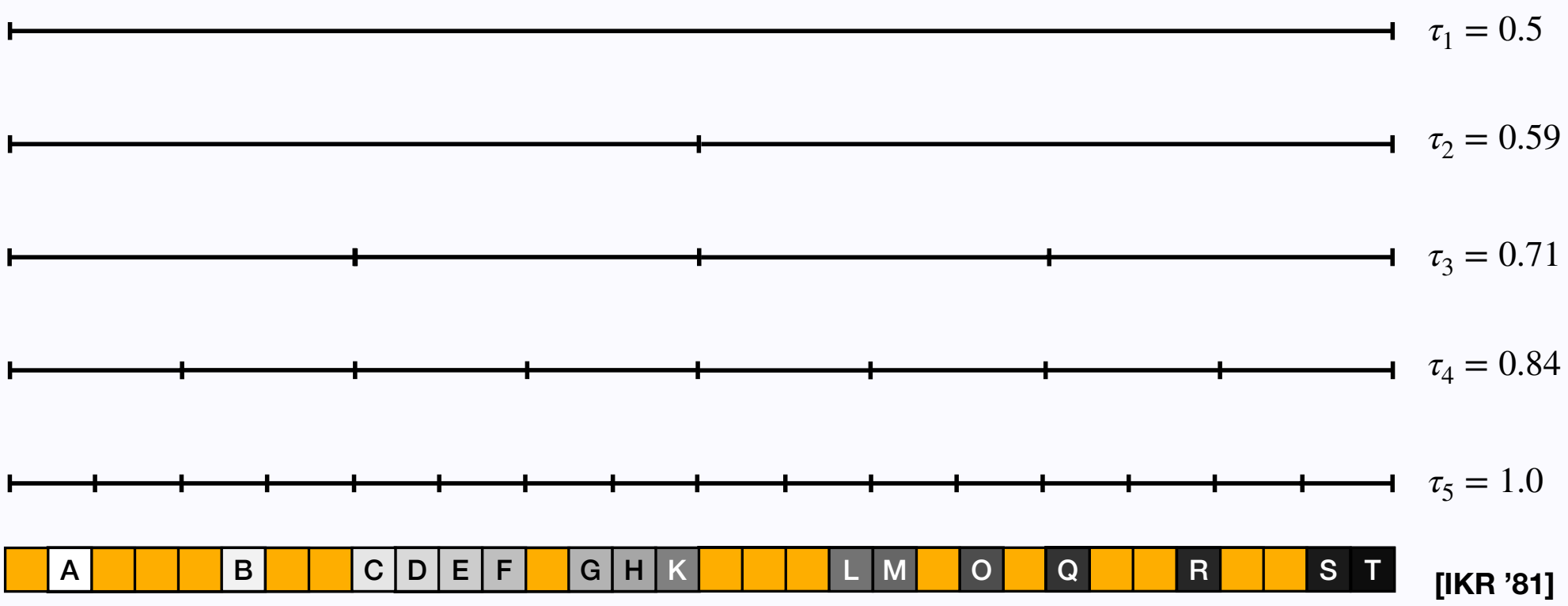
[Wheatman, Burns, IEEE BigData'21] [Wheatman, Xu, HPEC 2018, ALENEX'21] [Pandey, Wheatman, Xu, Buluc, SIGMOD'21] [De Leo, Boncz, GRADES and NDA'19, VLDB'21]

Controller problem

[Emek, Korman, Distributed Computing'11]

The Classical Solution [Itai, Konheim, Rodeh '81]

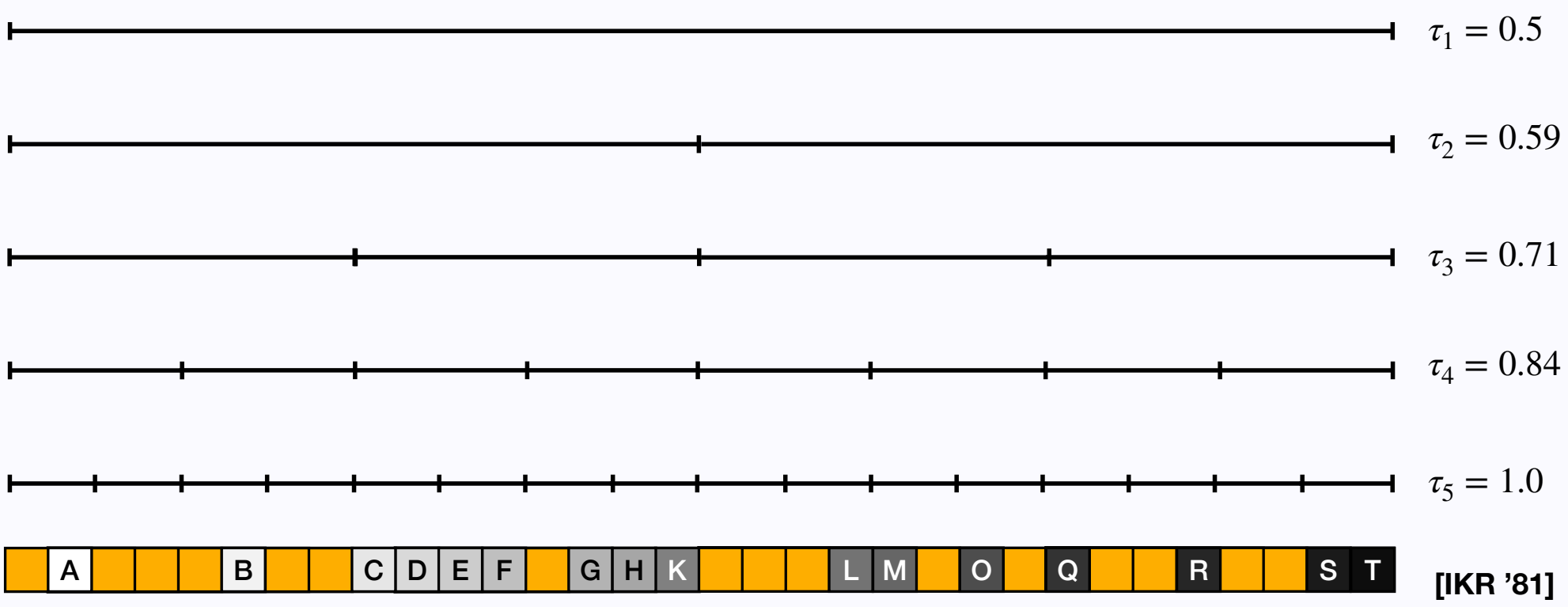
Step 1: Partition the array into nested intervals



[IKR '81]

The Classical Solution [Itai, Konheim, Rodeh '81]

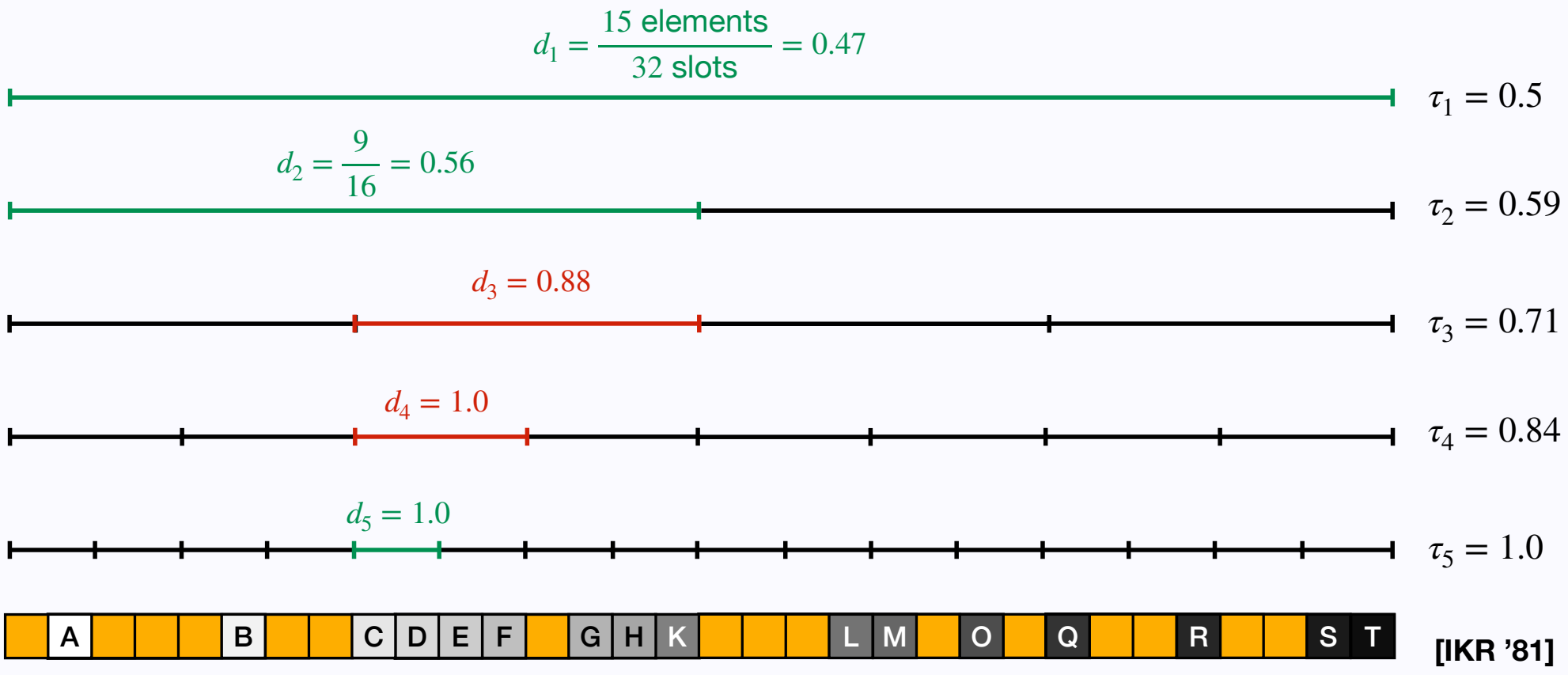
Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$



[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

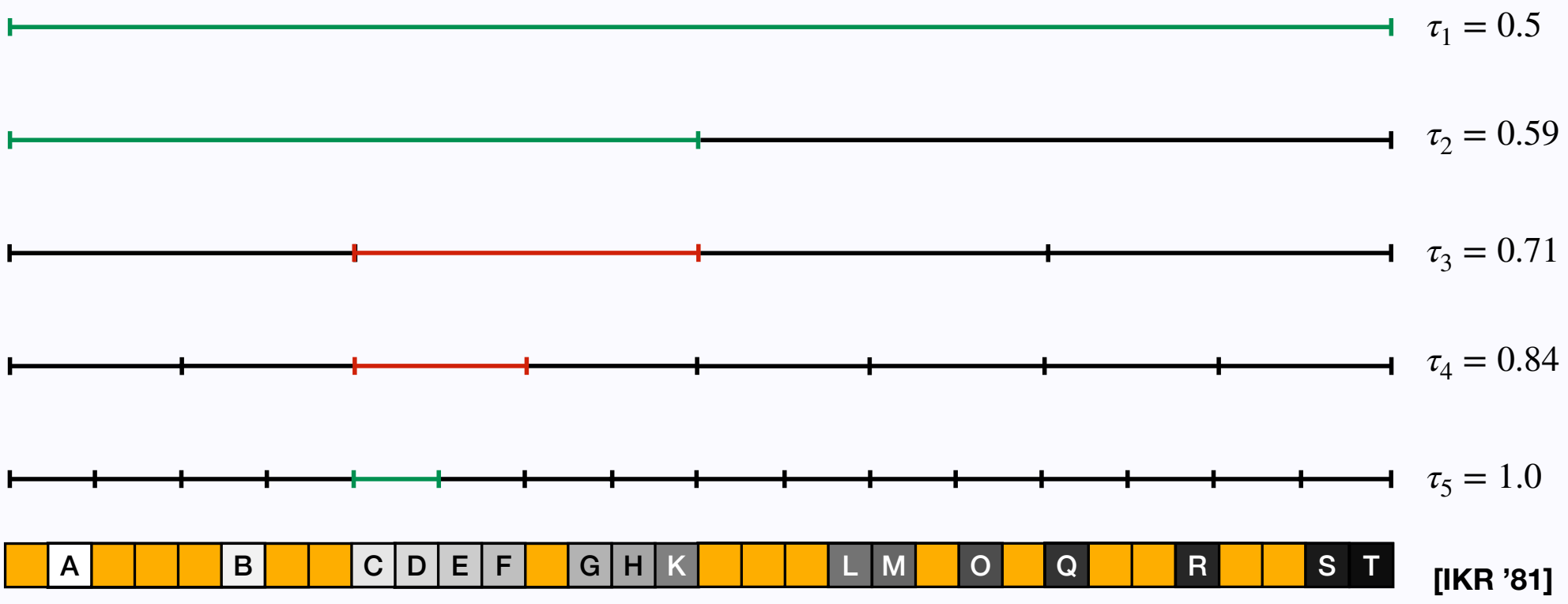


[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

➡ When an interval is over density, go up until $d_j \leq \tau_j$ and rebalance smoothly

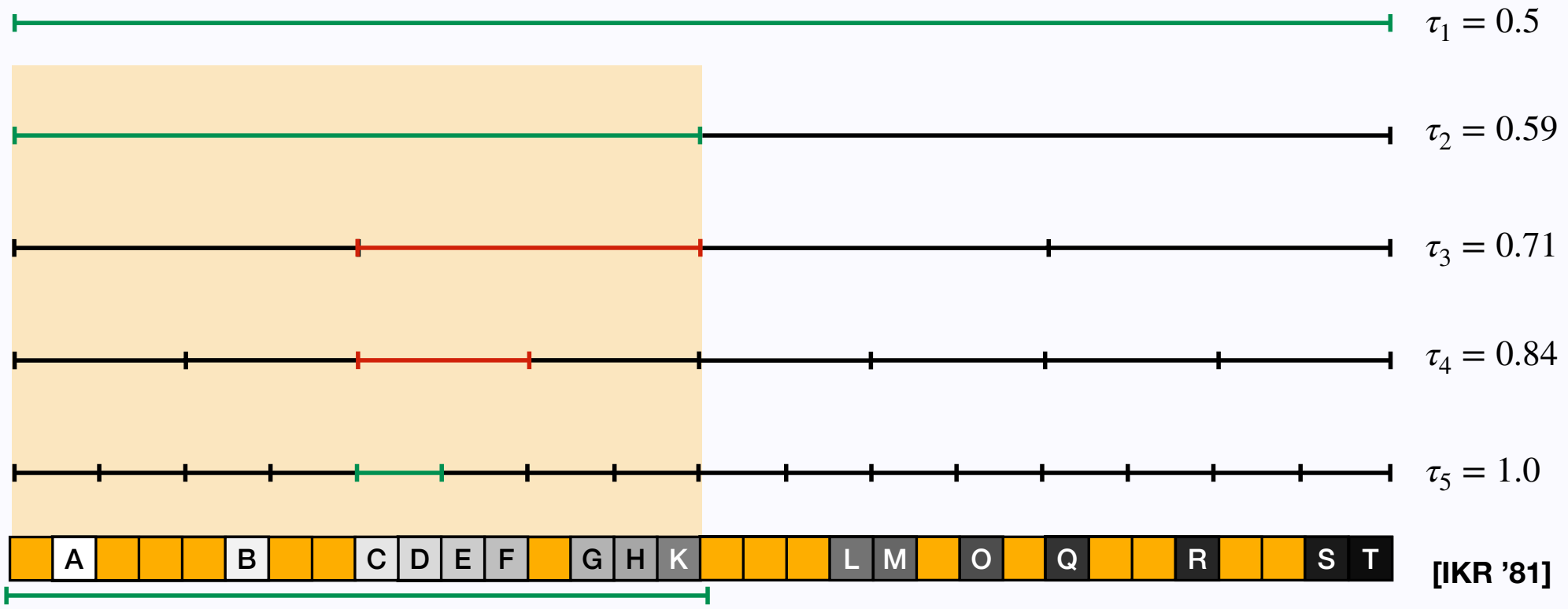


[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

➡ When an interval is over density, go up until $d_j \leq \tau_j$ and rebalance smoothly

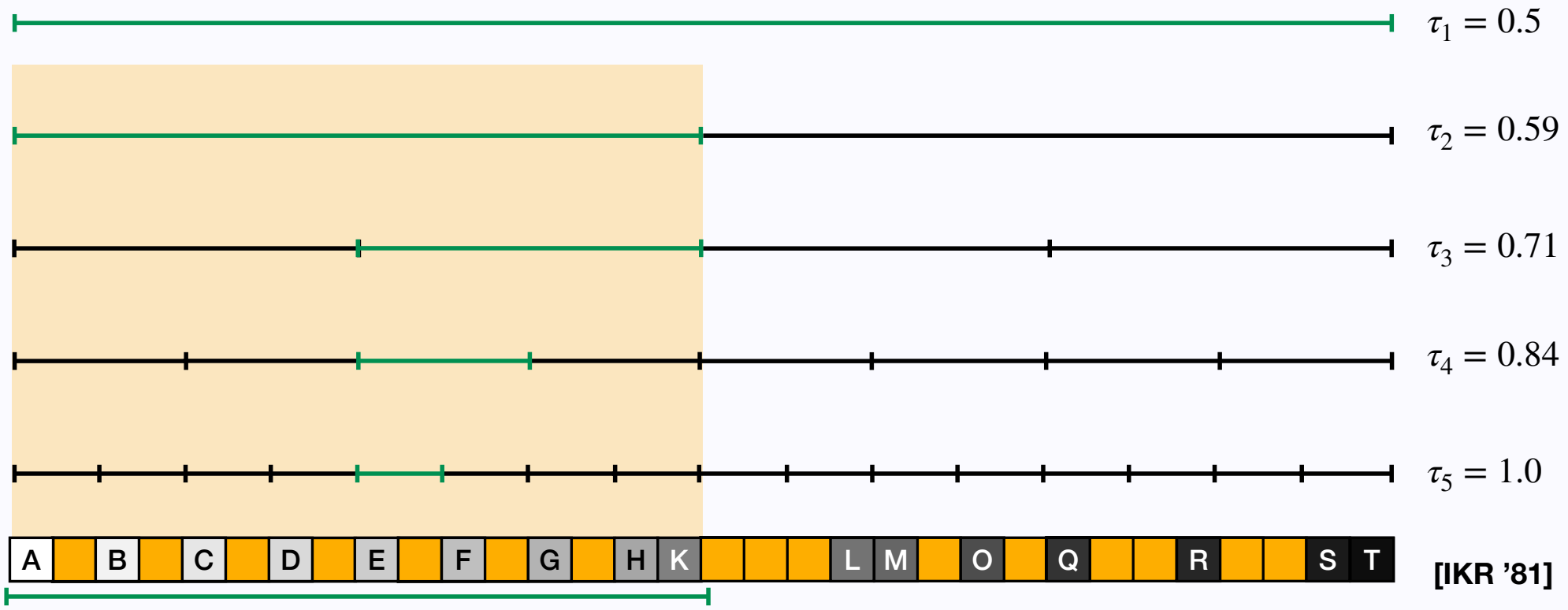


[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

➡ When an interval is over density, go up until $d_j \leq \tau_j$ and rebalance smoothly

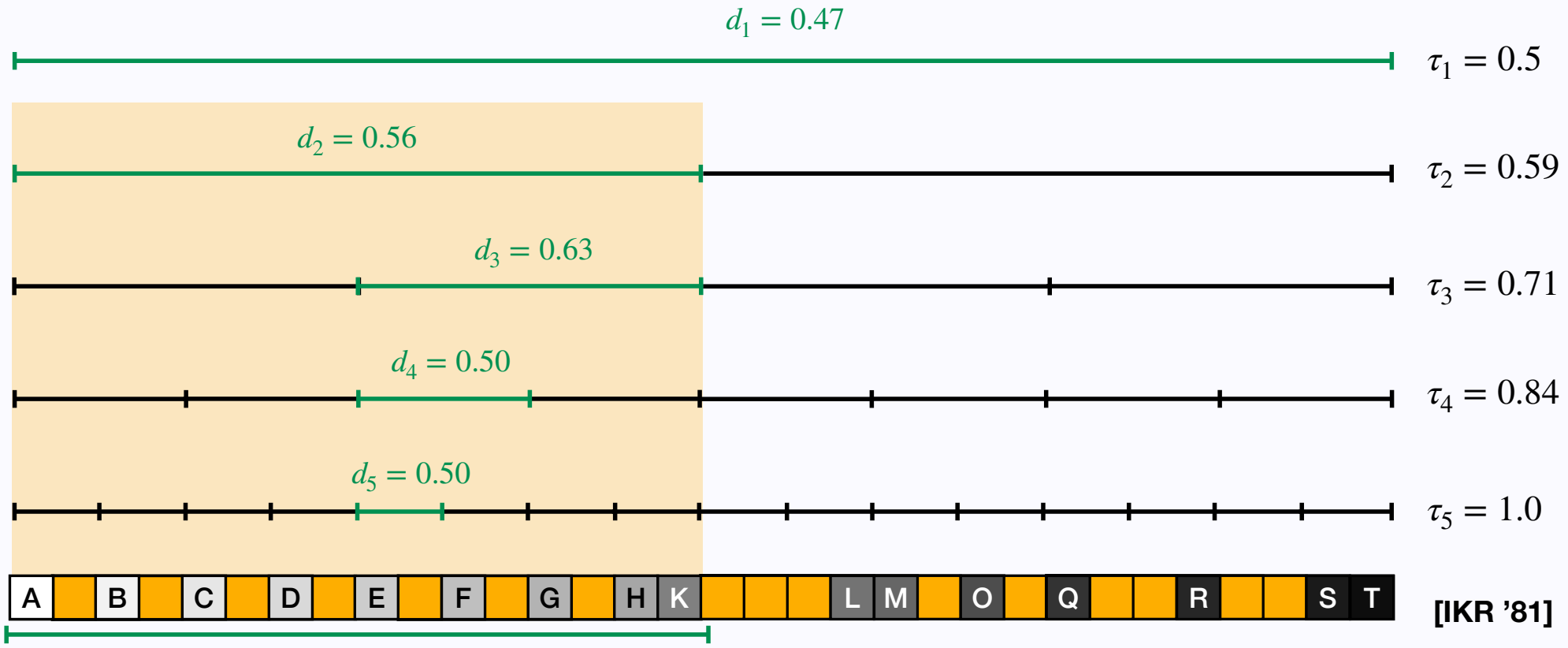


[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

➡ When an interval is over density, go up until $d_j \leq \tau_j$ and rebalance smoothly

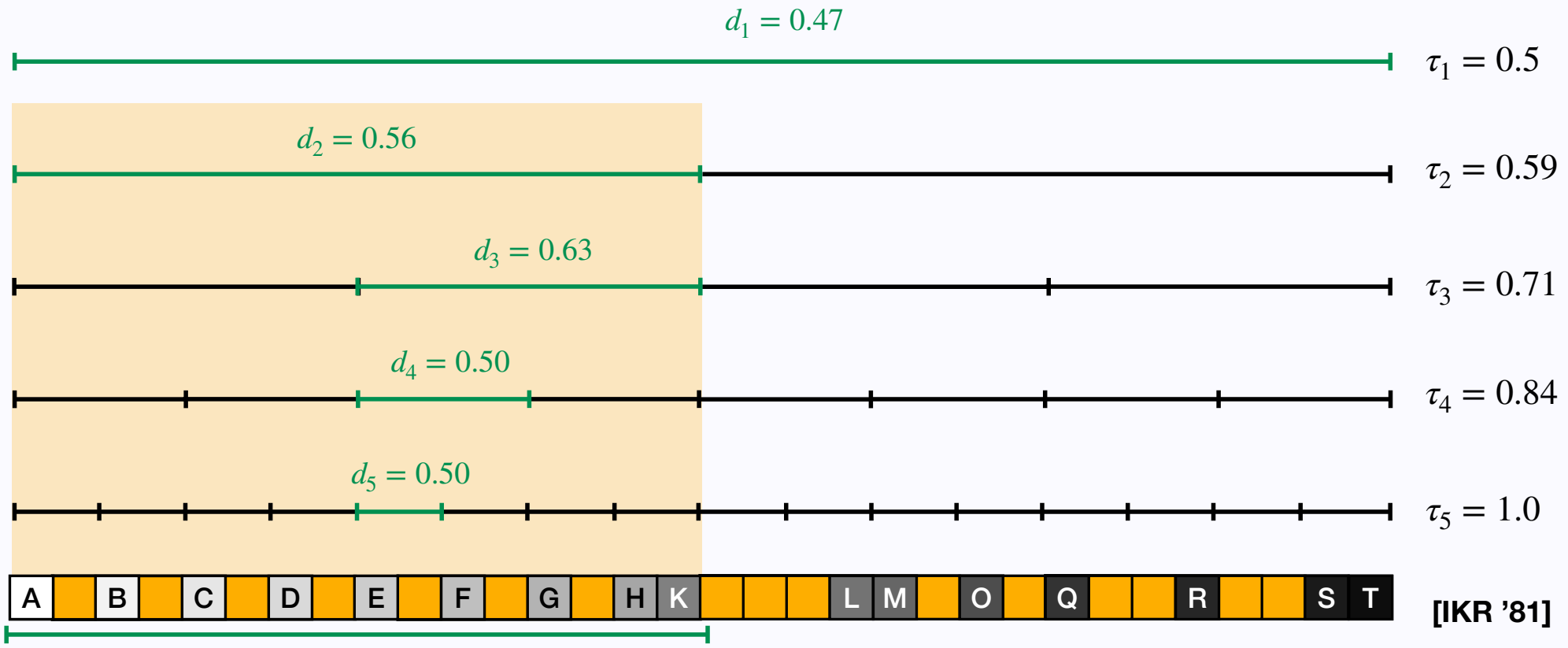


[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

➡ The density thresholds decrease for larger intervals



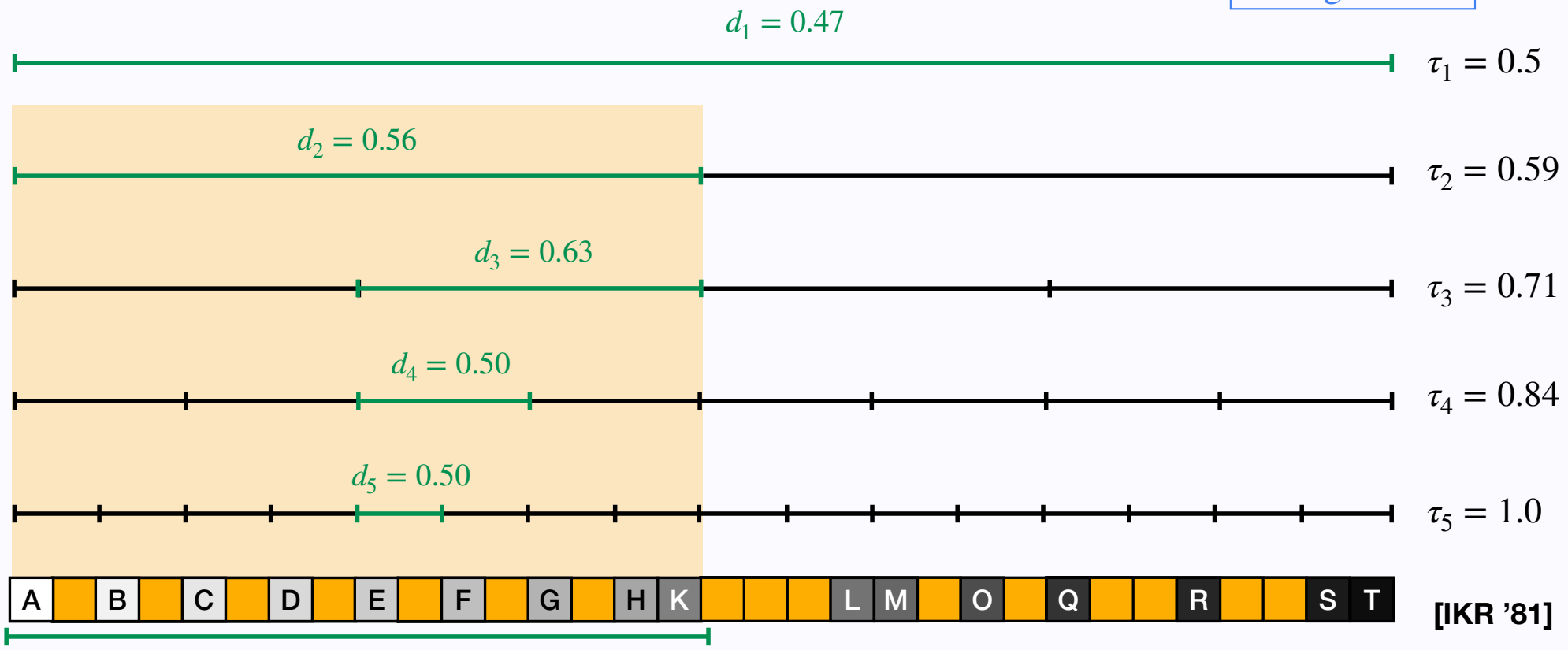
[IKR '81]

The Classical Solution

Invariant: The density of an interval at level i satisfies $d_i \leq \tau_i$

➡ The density thresholds decrease for larger intervals

$\Delta(\tau_i)$: factor of $(1 + 1/\log n)$ for $\log n$ levels



[IKR '81]

The Classical Solution

Consider an interval at level i of size s

- Last time it was rebalanced, it had density at most τ_{i-1}
- On next rebalance, it has density at least $\tau_i = (1 + 1/\log n)\tau_{i-1}$
- So we've added $\Omega(s/\log n)$ elements

[IKR '81]

The Classical Solution

Consider an interval at level i of size s

- Last time it was rebalanced, it had density at most τ_{i-1}
- On next rebalance, it has density at least $\tau_i = (1 + 1/\log n)\tau_{i-1}$
- So we've added $\Omega(s/\log n)$ elements

➡ The amortized cost of a rebalance at a specific level is $O(\log n)$

[IKR '81]

The Classical Solution

Consider an interval at level i of size s

- Last time it was rebalanced, it had density at most τ_{i-1}
- On next rebalance, it has density at least $\tau_i = (1 + 1/\log n)\tau_{i-1}$
- So we've added $\Omega(s/\log n)$ elements

➡ The amortized cost of a rebalance at a specific level is $O(\log n)$

➡ Each element can be rebalanced at each of its $\log n$ enclosing intervals

[IKR '81]

The Classical Solution

Consider an interval at level i of size s

- Last time it was rebalanced, it had density at most τ_{i-1}
- On next rebalance, it has density at least $\tau_i = (1 + 1/\log n)\tau_{i-1}$
- So we've added $\Omega(s/\log n)$ elements

➡ The amortized cost of a rebalance at a specific level is $O(\log n)$

➡ Each element can be rebalanced at each of its $\log n$ enclosing intervals

So the total cost is:

$\log n$	amortized cost per level
$\times \log n$	levels
$= \log^2 n$	total cost

[IKR '81]

Prior Work

$O(\log^2 n)$

Upper Bound

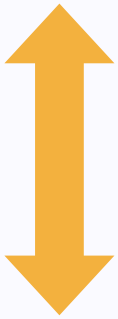
[Itai, Konheim, Rodeh ’81]

Prior Work

$O(\log^2 n)$

Upper Bound

[Itai, Konheim, Rodeh '81]

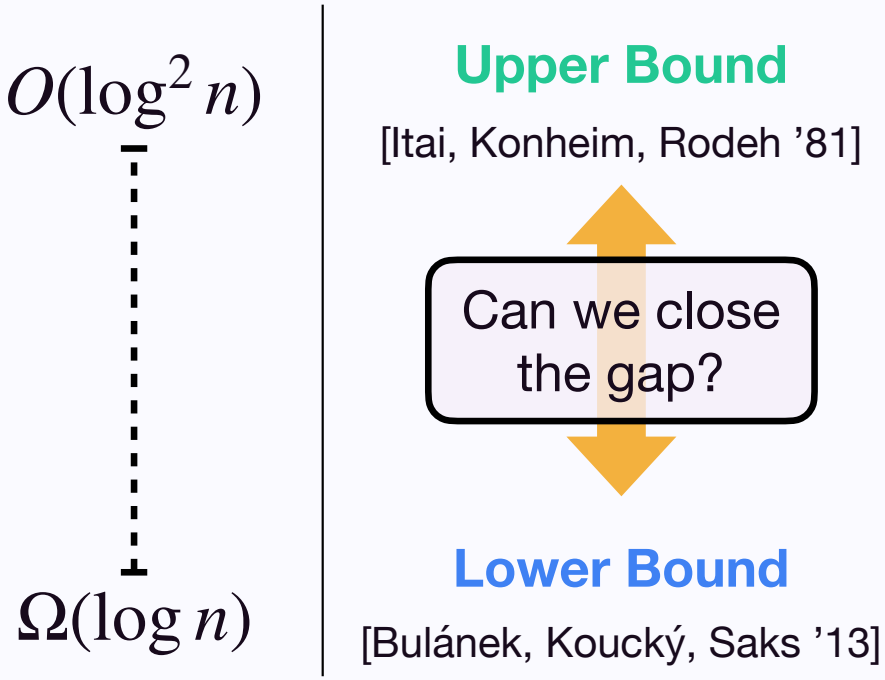


Lower Bound

[Bulánek, Koucký, Saks '13]

$\Omega(\log n)$

Open Problem



Open Problem

$O(\log^2 n)$

$\Omega(\log n)$

Upper Bound

[Itai, Konheim, Rodeh '81]

Can we close the gap?

Lower Bound

[Bulánek, Koucký, Saks '13]

Upper Bound

Unimproved

Formulations and (Re)Discoveries

Priority queues
[Itai, Konheim, Rodeh, ICALP'81]

Classical $O(\log^2 n)$ algorithm

File maintenance
[Willard, STOC'82, SIGMOD'86]

Deamortization of the classical algorithm

Order maintenance
[Dietz, STOC'82] [Dietz, Sleator, STOC'87]

Rediscovered algorithm, but on a larger array

Balanced binary trees & scapegoat trees
[Andersson, WADS'89] [Andersson, Lai, SWAT'90] [Galperin, Rivest, SODA'93]

Rediscovered algorithm, formulated as a tree

Locality preserving dictionaries
[Raman, PODS'99]

Rediscovered classical algorithm

Open Problem

$O(\log^2 n)$

$\Omega(\log n)$

Upper Bound

[Itai, Konheim, Rodeh '81]

Can we close the gap?

Lower Bound

[Bulánek, Koucký, Saks '13]

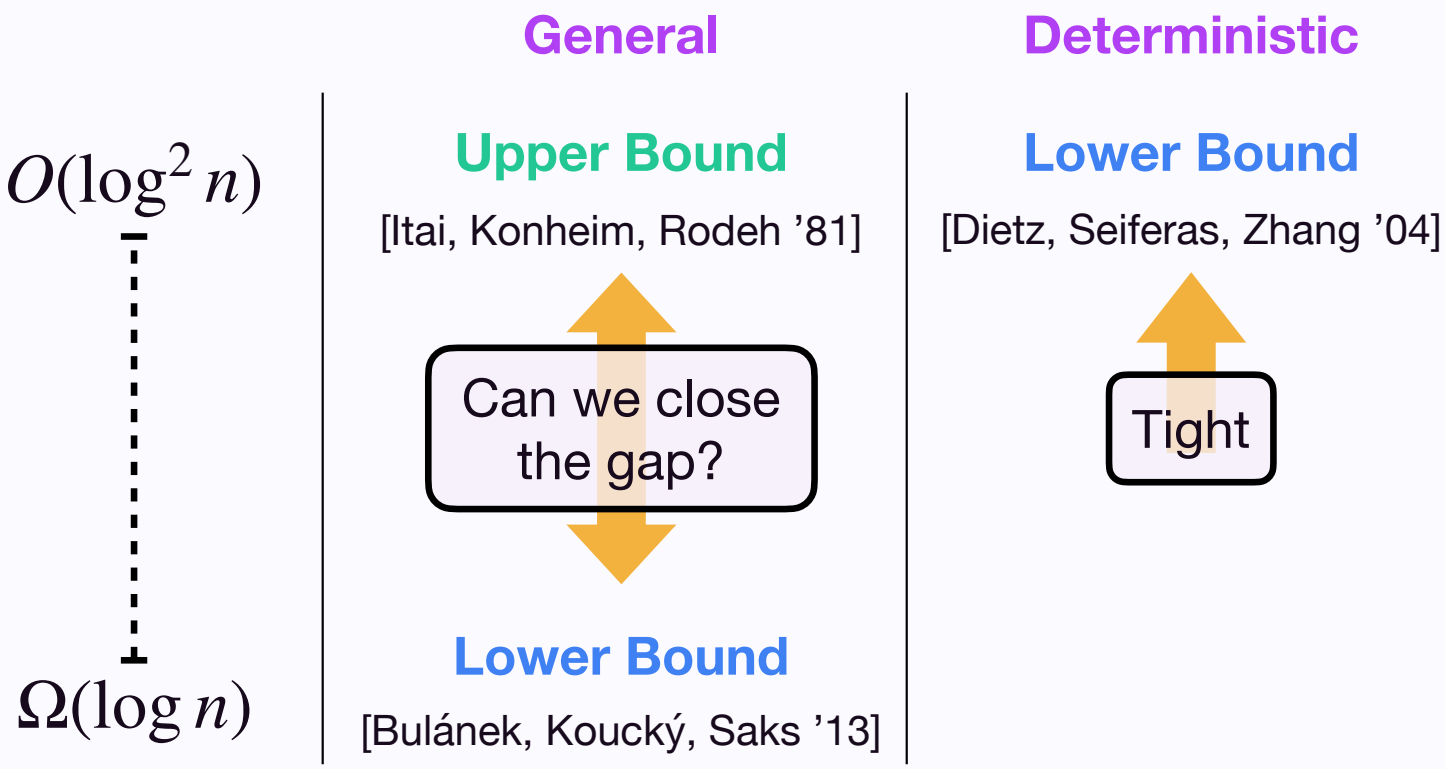
Upper Bound

Unimproved

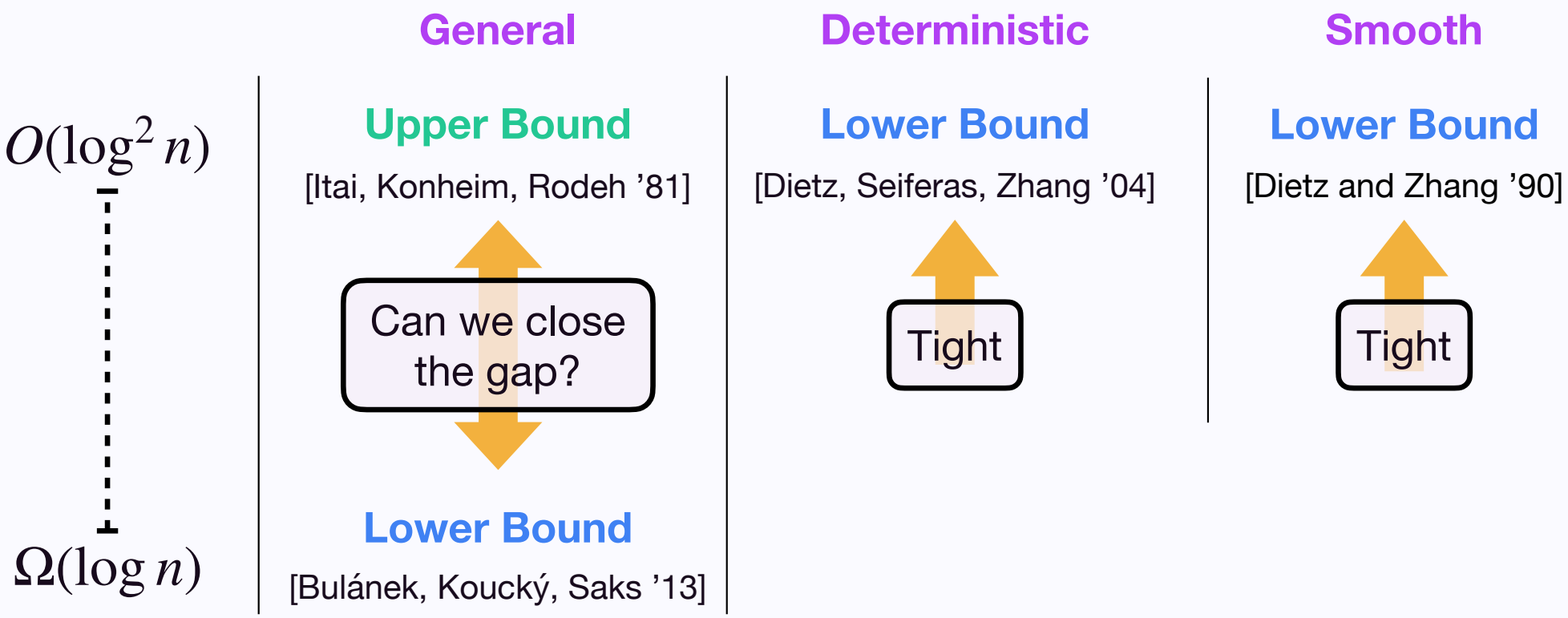
Lower Bound

Improved for specific cases

Prior Work


General
Deterministic
$O(\log^2 n)$
Upper Bound
[Itai, Konheim, Rodeh '81]
Lower Bound
[Dietz, Seiferas, Zhang '04]
Can we close the gap?
Tight
$\Omega(\log n)$
Lower Bound
[Bulánek, Koucký, Saks '13]

Prior Work


General
Deterministic
Smooth
$O(\log^2 n)$
$\Omega(\log n)$
Upper Bound
[Itai, Konheim, Rodeh '81]
Can we close the gap?
Lower Bound
[Bulánek, Koucký, Saks '13]
Lower Bound
[Dietz, Seiferas, Zhang '04]
Tight
Lower Bound
[Dietz and Zhang '90]
Tight

Prior Work

Evenly spaced rebalances only

General

Upper Bound

[Itai, Konheim, Rodeh '81]

Can we close the gap?

Lower Bound

[Bulánek, Koucký, Saks '13]

Deterministic

Lower Bound

[Dietz, Seiferas, Zhang '04]

Tight

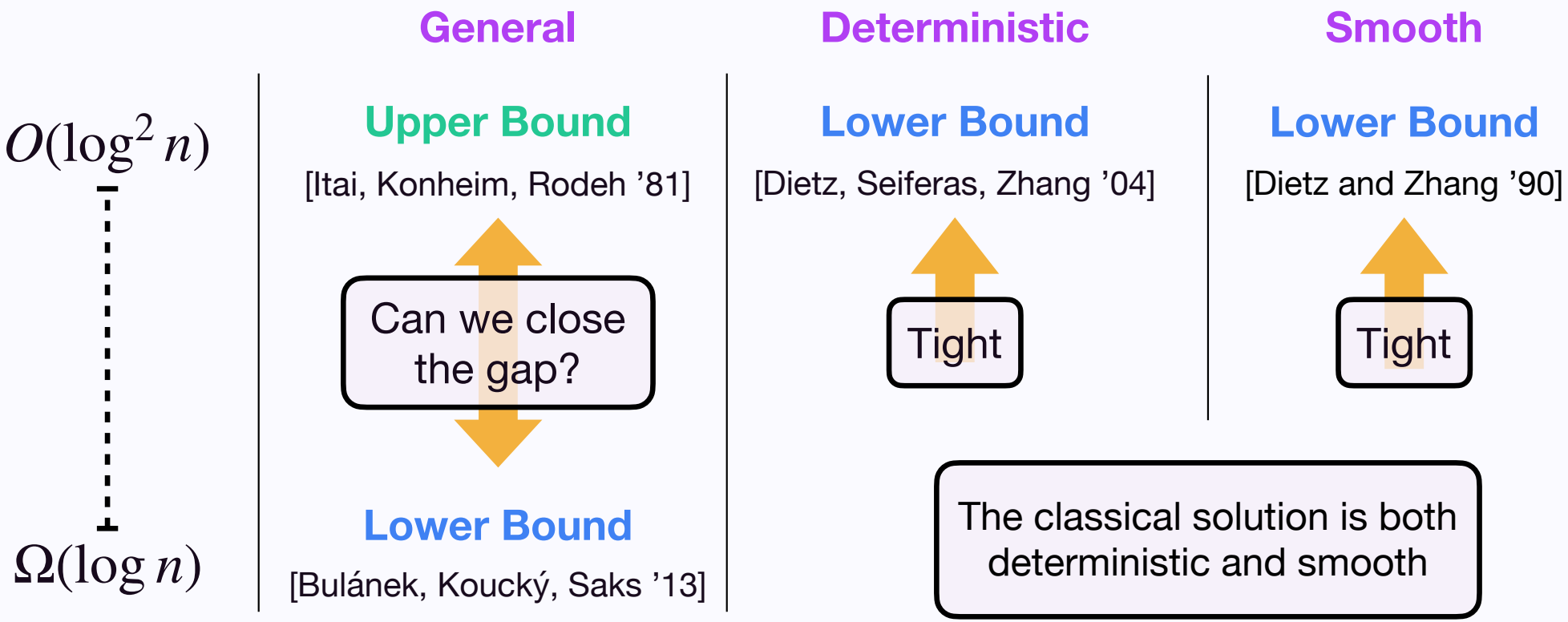
Smooth

Lower Bound

[Dietz and Zhang '90]

Tight

Prior Work


General
Deterministic
Smooth
$O(\log^2 n)$
$\Omega(\log n)$
Upper Bound
[Itai, Konheim, Rodeh '81]
Can we close the gap?
Lower Bound
[Bulánek, Koucký, Saks '13]
Lower Bound
[Dietz, Seiferas, Zhang '04]
Tight
Lower Bound
[Dietz and Zhang '90]
Tight
The classical solution is both deterministic and smooth

Our Result:

Theorem: Algorithm for list labeling with $O(\log^{3/2} n)$ expected cost per insertion/deletion

Our Results

$O(\log^2 n)$

$O(\log^{3/2} n)$

$\Omega(\log n)$

Prior Work

Upper Bound

[Itai, Konheim, Rodeh '81]

Can we close the gap?

Lower Bound

[Bulánek, Koucký, Saks '13]

Our Results

Upper Bound

Algorithm with $O(\log^{3/2} n)$ expected cost per update

Our Results

$O(\log^2 n)$

$O(\log^{3/2} n)$

$\Omega(\log n)$

Prior Work

Upper Bound

[Itai, Konheim, Rodeh '81]

Can we close the gap?

Lower Bound

[Bulánek, Koucký, Saks '13]

Our Results

Upper Bound

Algorithm with $O(\log^{3/2} n)$ expected cost per update

Matching lower bound for History-Independent algorithms

Lower Bound

Our Results

$O(\log^2 n)$

$O(\log^{3/2} n)$

$\Omega(\log n)$

Prior Work

Upper Bound

[Itai, Konheim, Rodeh '81]

Can we close the gap?

Lower Bound

[Bulánek, Koucký, Saks '13]

Our Results

Upper Bound

Algorithm with $O(\log^{3/2} n)$ expected cost per update

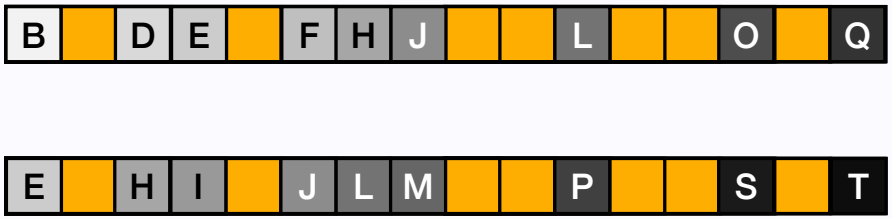
Matching lower bound for History-Independent algorithms

Lower Bound

So what is history independence?

History Independence — Definition

History Independence: For our problem, the distribution of occupied slots depends **only** on number of elements currently present



History Independence — Prior Work

Surprising result:

- $O(\log^2 n)$ history-independent algorithm
  [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage, PODS'16]

History Independence — Prior Work

Surprising result:

- $O(\log^2 n)$ history-independent algorithm
  [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage, PODS'16]

➡ No increase in cost/runtime to achieve history independence

History Independence — Prior Work

Surprising result:

- $O(\log^2 n)$ history-independent algorithm
  [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage, PODS'16]

➡ No increase in cost/runtime to achieve history independence

- Previously viewed as a desirable property to achieve **in its own right** (e.g. for security guarantees)
- In contrast, we **use** history independence to achieve a **faster** algorithm

History Independence — Prior Work

Surprising result:

- $O(\log^2 n)$ history-independent algorithm
  [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage, PODS'16]

➡ No increase in cost/runtime to achieve history independence

Their main idea: Use randomness to determine left/right split of elements

➡ Reformulate density constraint as an imbalance constraint between the left/right subintervals

➡ Maintains uniform distribution of left/right imbalances satisfying the constraint

➡ In contrast, classical algorithm always splits evenly when rebuilding

Our Main Idea: Rebalance Less Frequently

Looser balance thresholds $\Longrightarrow$ Less frequent rebalancing $\Longrightarrow$ Lower Cost

Our Main Idea: Rebalance Less Frequently

Looser balance thresholds $\Longrightarrow$ Less frequent rebalancing $\Longrightarrow$ Lower Cost

Roughly speaking, densities differ by a $\log^{-1/2} n$ factor instead of $\log^{-1} n$

Our Main Idea: Rebalance Less Frequently

Looser balance thresholds $\Longrightarrow$ Less frequent rebalancing $\Longrightarrow$ Lower Cost

Roughly speaking, densities differ by a $\log^{-1/2} n$ factor instead of $\log^{-1} n$

Pro: Cost of maintaining our new invariant: ~~$O(\log n)$~~ $O(\log^{1/2} n)$ at each level

$\Longrightarrow$ Final cost of $O(\log^{3/2} n)$ 😀

Our Main Idea: Rebalance Less Frequently

Looser balance thresholds $\Longrightarrow$ Less frequent rebalancing $\Longrightarrow$ Lower Cost

Roughly speaking, densities differ by a $\log^{-1/2} n$ factor instead of $\log^{-1} n$

Pro: Cost of maintaining our new invariant: ~~$O(\log n)$~~ $O(\log^{1/2} n)$ at each level

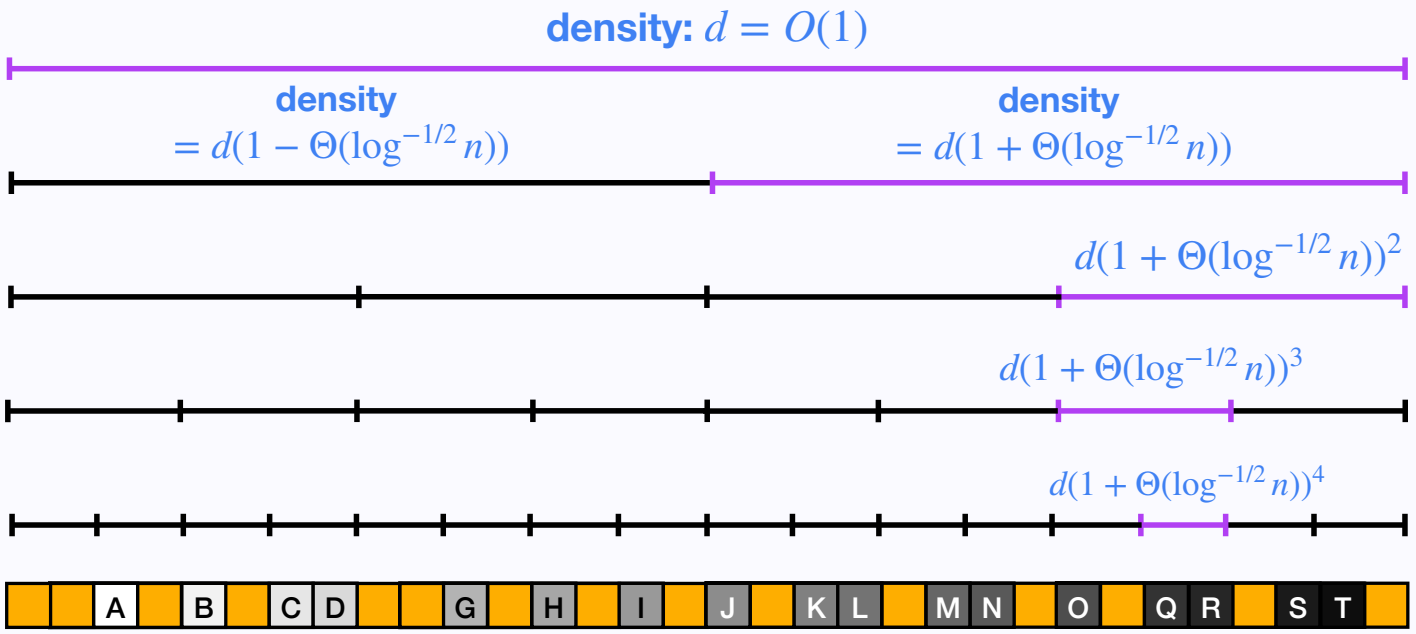
$\Longrightarrow$ Final cost of $O(\log^{3/2} n)$ 😀

Con: Some intervals may overflow 😟

Problem: Overflows Can Happen

Con: Some intervals may overflow

- Density can increase by a factor of $1 + \Theta(\log^{-1/2} n)$ at each level:



Problem: Overflows Can Happen

Con: Some intervals may overflow

- Density can increase by a factor of $1 + \Theta(\log^{-1/2} n)$ at each level:
  - ➡ Density at bottom level: can be $d(1 + \Theta(\log^{-1/2} n))^{\log n} = \omega(1)$

Next Idea: Rebalance Condition Depends on Density

Intuition: Revert to classical solution in regions that get “too” dense

Next Idea: Rebalance Condition Depends on Density

Intuition: Revert to classical solution in regions that get “too” dense

- Higher density $\Longrightarrow$ More stringent balance condition $\Longrightarrow$ Prevents overflow

Next Idea: Rebalance Condition Depends on Density

Intuition: Revert to classical solution in regions that get “too” dense

- Higher density $\Longrightarrow$ More stringent balance condition $\Longrightarrow$ Prevents overflow
- Lower density $\Longrightarrow$ Less stringent balance condition $\Longrightarrow$ Less work

Next Idea: Rebalance Condition Depends on Density

Intuition: Revert to classical solution in regions that get “too” dense

- Higher density $\Longrightarrow$ More stringent balance condition $\Longrightarrow$ Prevents overflow
  - ➡ $O(\log^2 n)$ cost
- Lower density $\Longrightarrow$ Less stringent balance condition $\Longrightarrow$ Less work
  - ➡ $O(\log^{3/2} n)$ cost

Next Idea: Rebalance Condition Depends on Density

Intuition: Revert to classical solution in regions that get "too" dense

- Higher density $\Longrightarrow$ More stringent balance condition $\Longrightarrow$ Prevents overflow
  - ➡ $O(\log^2 n)$ cost
- Lower density $\Longrightarrow$ Less stringent balance condition $\Longrightarrow$ Less work
  - ➡ $O(\log^{3/2} n)$ cost

We show: Vast majority of the array is sparse $\Longrightarrow$ $O(\log^{3/2} n)$ expected total cost

Last Idea: Eliminate Potential Vulnerabilities

Hide the dense regions from the adversary

Last Idea: Eliminate Potential Vulnerabilities

Hide the dense regions from the adversary

- Our strategy may create some dense regions in the array
  - ➡ An adversary could target and exploit them to drive up cost

Last Idea: Eliminate Potential Vulnerabilities

Hide the dense regions from the adversary

- Our strategy may create some dense regions in the array
  - ➡ An adversary could target and exploit them to drive up cost

- **Solution:** History independence!
  - ➡ The adversary can neither find nor create dense regions
  - ➡ Additional idea: Apply a random shift — See the paper

The Final $O(\log^{3/2} n)$ Algorithm

The Final $O(\log^{3/2} n)$ Algorithm

Prior $O(\log^2 n)$ History-Independent Algorithm

[BBJKMPSSZ '16]

The Final $O(\log^{3/2} n)$ Algorithm

Prior $O(\log^2 n)$ History-Independent Algorithm

[BBJKMPSSZ '16]

Loosen rebalancing rule
— Speeds up the algorithm

The Final $O(\log^{3/2} n)$ Algorithm

Prior $O(\log^2 n)$ History-Independent Algorithm

[BBJKMPSSZ '16]

Loosen rebalancing rule
— Speeds up the algorithm

Stricter rebalancing rule in dense regions
— Prevents overflow

The Final $O(\log^{3/2} n)$ Algorithm

Prior $O(\log^2 n)$ History-Independent Algorithm

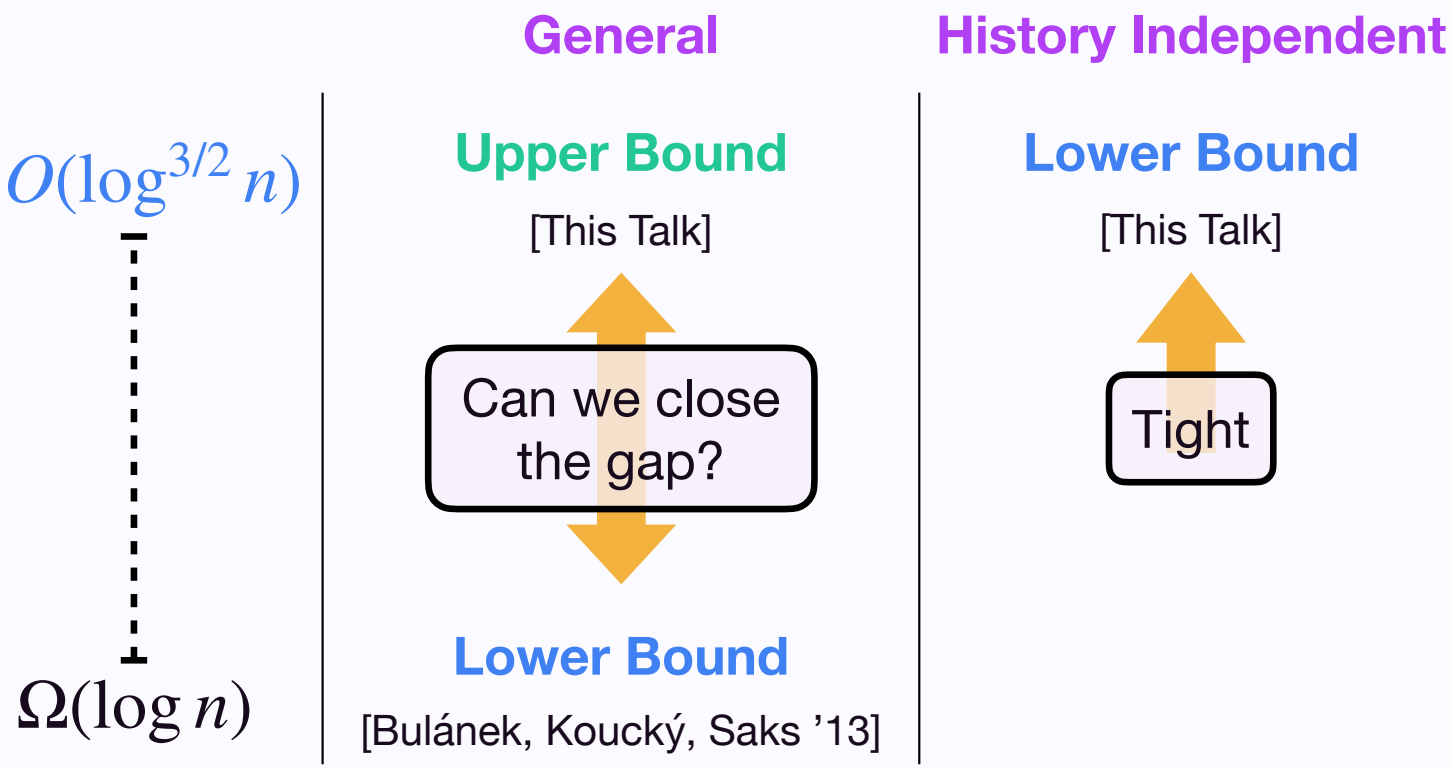
[BBJKMPSSZ '16]

Loosen rebalancing rule
— Speeds up the algorithm

Stricter rebalancing rule in dense regions
— Prevents overflow

Smoothly vary between rebalance rules
— Reduces cost from $\tilde{O}(\log^{3/2} n)$ to $O(\log^{3/2} n)$

Summary of Results & New Open Problem



Thank you!

- Michael A. Bender, Stony Brook University — bender@cs.stonybrook.edu
- Alex Conway, VMWare Research — aconway@vmware.com
- Martín Farach-Colton, Rutgers University — martin@farach-colton.com
- Hanna Komlós, Rutgers University —hkomlos@gmail.com
- William Kuszmaul, MIT — william.kuszmaul@gmail.com
- Nicole Wein, DIMACS — nicole.wein@rutgers.edu