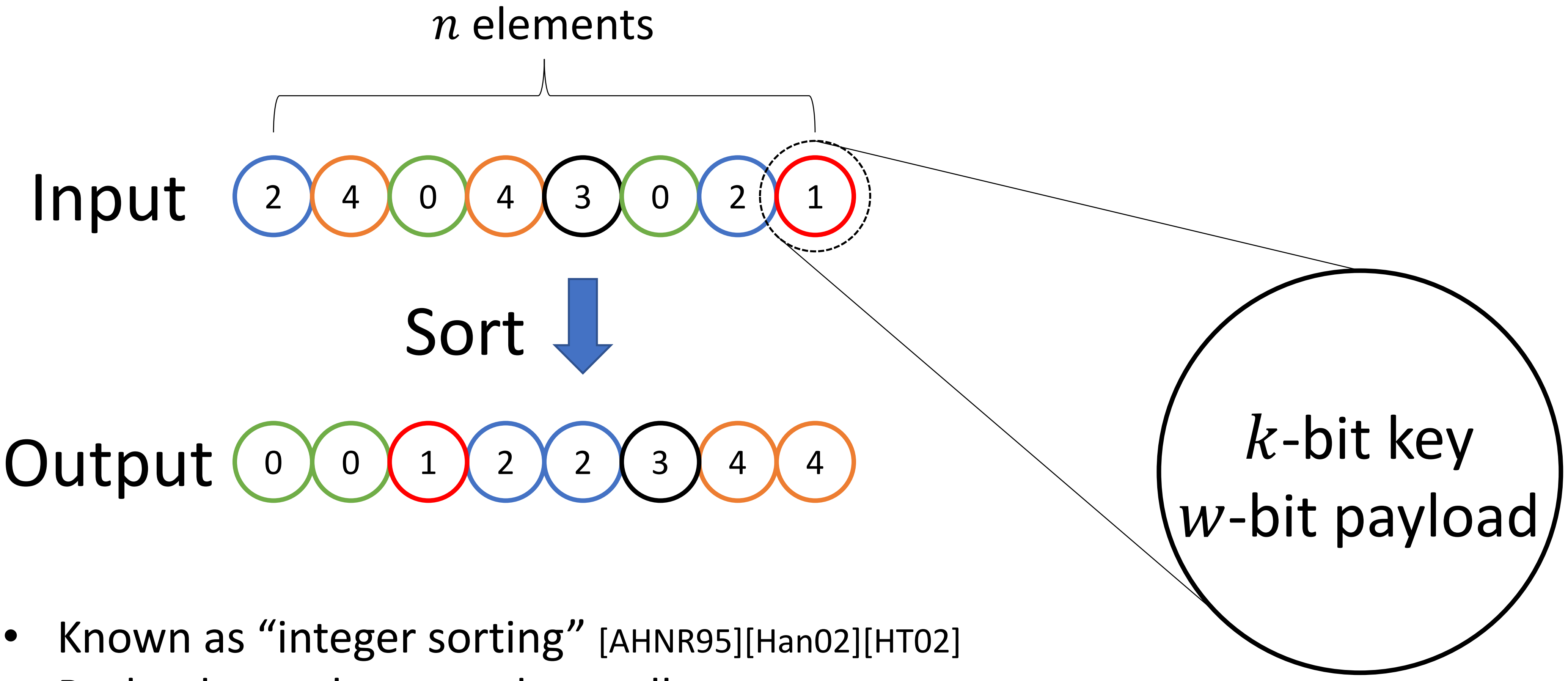
SIAM 2022 | ACM-SIAM Symposium on Discrete Algorithms

Optimal Sorting Circuits for Short Keys

Wei-Kai Lin

and Elaine Shi (Carnegie Mellon university)

Sorting, parameters: n, k, w



- Known as “integer sorting” [AHNR95][Han02][HT02]
- Payload must be moved as well
- Stability is not required

Input 2 4 0 4 3 0 2 1

k-bit key

w-bit payload

Circuit

- Input & output: $n \cdot (k + w)$ bits
- Const fan-in and fan-out gates (AND, OR, NOT)
- Efficiency metric (goal):
  small size (number of gates)
  small depth (length from input to output)

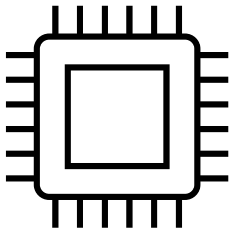
Output 0 0 1 2 2 3 4 4

Why circuit?

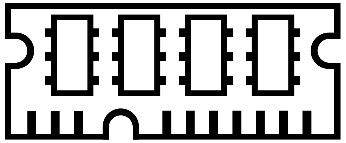
Random-Access Machine model (RAM)

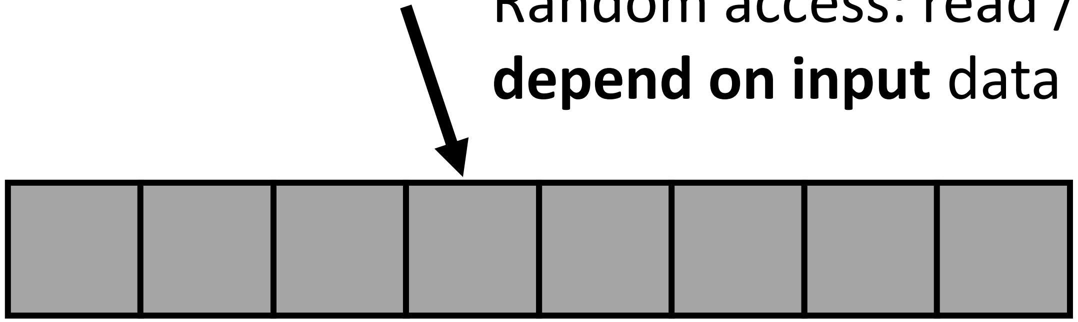
- Textbook counting sort and radix sort: $O(n \cdot k)$
- Sort n integers, nearly linear time (e.g. $O\left(n \cdot \sqrt{\log \log n}\right)$
  [Kirkpatrick-Reisch81][Andersson-Hagerup-Nilsson-Raman95] [Han-Thorup02] [Thorup02] [Han04] [Belazzougui-Brodal-Nielsen14]
  (word size > log n bits)
- Techniques: counting / hashing based ➔ need **random accesses**

Circuit is **fixed**
➔ more challenging



Random access: read / write memory word **depend on input** data





Sorting circuits imply *super* efficient algorithms

Implication

- Offline oblivious RAM [Boyle-Naor16]
- Function inversion / static non-adaptive data structures [Hellman80] [Corrigan-Gibbs&Kogan19] [Dvořák-Koucký-Král-Slívová21]
- Network coding conjecture [Ahlswede-Cai-Li-Yeung00] [Li-Li04] [Adler-Harvey-Jain-Kleinberg-Lehman06] [Afshani-Freksen-Kamma-Larsen19] [Asharov-**Lin**-Shi21]

Sorting circuit is “not easier” to construct

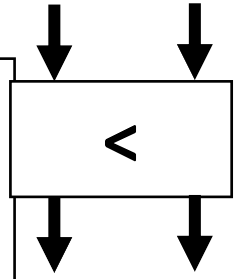
Lower bound for XXX is
“not easier than lower bound for sorting circuits”
(barrier for lower bound)

Question:
Best sorting in circuit size and depth?

Previous sorting circuits

Comparison-based “sorting networks”:

“Comparator”



- Bitonic sort [Batcher68]
  size $O\big((k+w)\cdot n\log^2 n\big)$, depth $O(\log^2 n)$ (practical)
- AKS [Ajtai-Komlos-Szemeredi83] [Patterson90] [Seiferas09] [Goodrich14]
  size $O\big((k+w)\cdot n\log n\big)$, depth $O(\log n)$

- Comparison-based:
  - $(k+w)\cdot n\log n$ is necessary
  - even when $k=1$
    (zero-one principle [Knuth98])

- “Indivisible” payloads:
  - $(k+w)\cdot n\cdot k$ is necessary
  - If stable (order preserving),
    $n\log n$ even when $k=1$
    [**Lin**-Shi-Xie19]

Previous sorting circuits

Non-comparison-based, indivisible payload, not stable:

- [Pippenger96], "self-routing superconcentrator"
  size $O\big((k+w)\cdot n + n\cdot\log n\big)$, depth $O(\log^2 n)$
- [Leighton-Ma-Suel95] [Mitchell-Zimmerman14] [**Lin**-Shi-Xie19] (randomized)
  [Asharov-Komargodski-**Lin**-Nayak-Peserico-Shi20] [Dittmer-Ostrovsky20]
  size $O\big((k+w)\cdot n\cdot k\cdot\log\log n\big)$, depth $poly\log n$
- [Asharov-**Lin**-Shi21]
  size $O\big((k+w)\cdot n\cdot k\cdot poly(\log^* n-\log^*(k+w))\big)$, depth $> poly\log n$

All $poly\log n$ depth

- [Koucký-Král21, concurrent]
  size $O\big((k+w)\cdot n\cdot k\cdot(\log^* n-\log^*(k+w))\big)$, depth $O(\log^3 n)$

Previous “small” sorting circuits

➔ All $poly \log n$ depth

Lower bound is $\log n$ [Cook-Dwork-Reischuk86]

AKS is $\mathrm{O}(\log n)$ depth

Some implication

need log depth

[Corrigan-Gibbs&Kogan19]

[Dvořák-Koucký-Král-Slívová21]

Main question:

Small size (<< n log n) and log depth?

Main Theorem:
Sort n elements, each consists of k-bit key and w-bit payload, in circuit size $O\big((k+w)\cdot n\cdot k\cdot poly(\log^* n-\log^*(k+w))\big)$, depth $O(\log n+\log w)$

Non-comparison, "indivisible" payload, not stable

Size = $O\big((k+w)\cdot n\cdot k\big)$ for any $(k+w)>\log^{(100)} n$ [optimal]

Intermediate result, Deterministic Oblivious Parallel RAM:
Sort n elements, each consists of k-bit key and w-bit payload, in total work $O(n\cdot k)$, parallel time $O(\log n)$ [optimal]

Application: To hide data from adversary that "observe accesses"
E.g. oblivious sorting is essential for oblivious RAM (ORAM) algorithms
[GO96] [Ajtai10] [DMN1] [GM11] [KLO12] [CG**L**S17] [PPRY18] [AK**L**NPS20] [DO20] …

Main Theorem:
Sort n elements, each consists of k-bit key and w-bit payload, in circuit size $O\big((k+w)\cdot n\cdot k\cdot poly(\log^* n - \log^*(k+w))\big)$, depth $O(\log n + \log w)$

Challenges

Achieve $\log n$ depth circuit for $k = 1$:

- All previous & concurrent results takes depth $poly \log n$
  [Pippenger96] [Asharov-**Lin**-Shi21] [Koucký-Král21]
  Based on Pippenger's "self-routing superconcentrator"
- Need depth $O(\log n)$

Novel 1-bit to k-bit upgrade:

- 1-bit is not stable ➔ "radix sort" not work
- "Quick sort" approach (using median) ➔ poly log factors
  [**Lin**-Shi-Xie19] [Asharov-**Lin**-Shi21] [Koucký-Král21]
- Need "additive" depth

1-bit to k-bit upgrade — Previous approaches

Radix sort?

n elements, k-bit keys

1-bit sorter (least significant bit)

1-bit sorter (2nd least significant bit)

output

✕ Need **stable**

If stable and indivisible, $n \log n$ is necessary [**Lin**-Shi-Xie19]

Quicksort?

n elements, k-bit keys

Find median, then 1-bit sorter

✕ Depth of median?

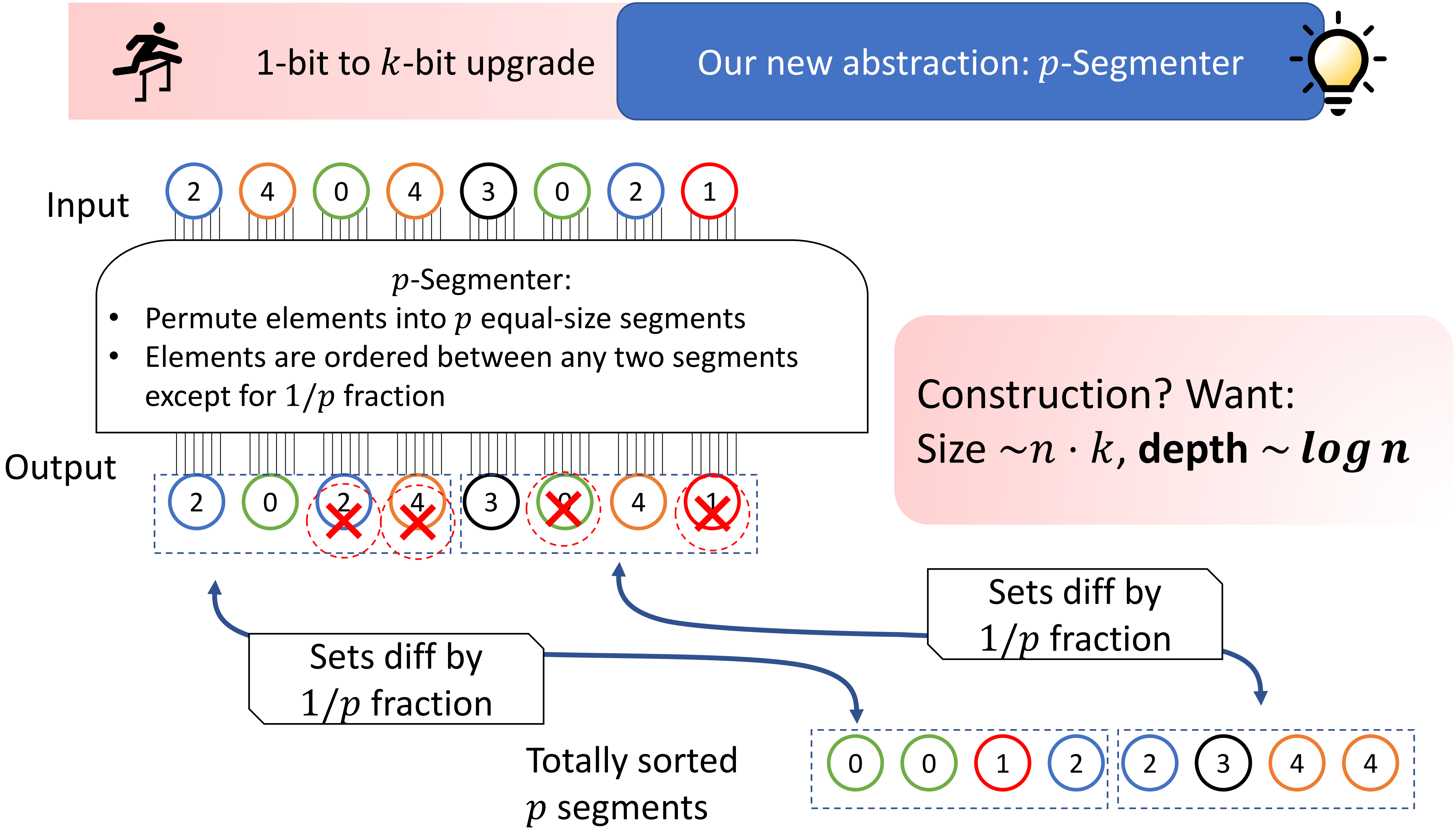
$n/2$ elem, smaller | $n/2$ elem, greater

Median + 1-bit sort | Median + 1-bit sort

$n/4$ | $n/4$ | $n/4$ | $n/4$

⋮

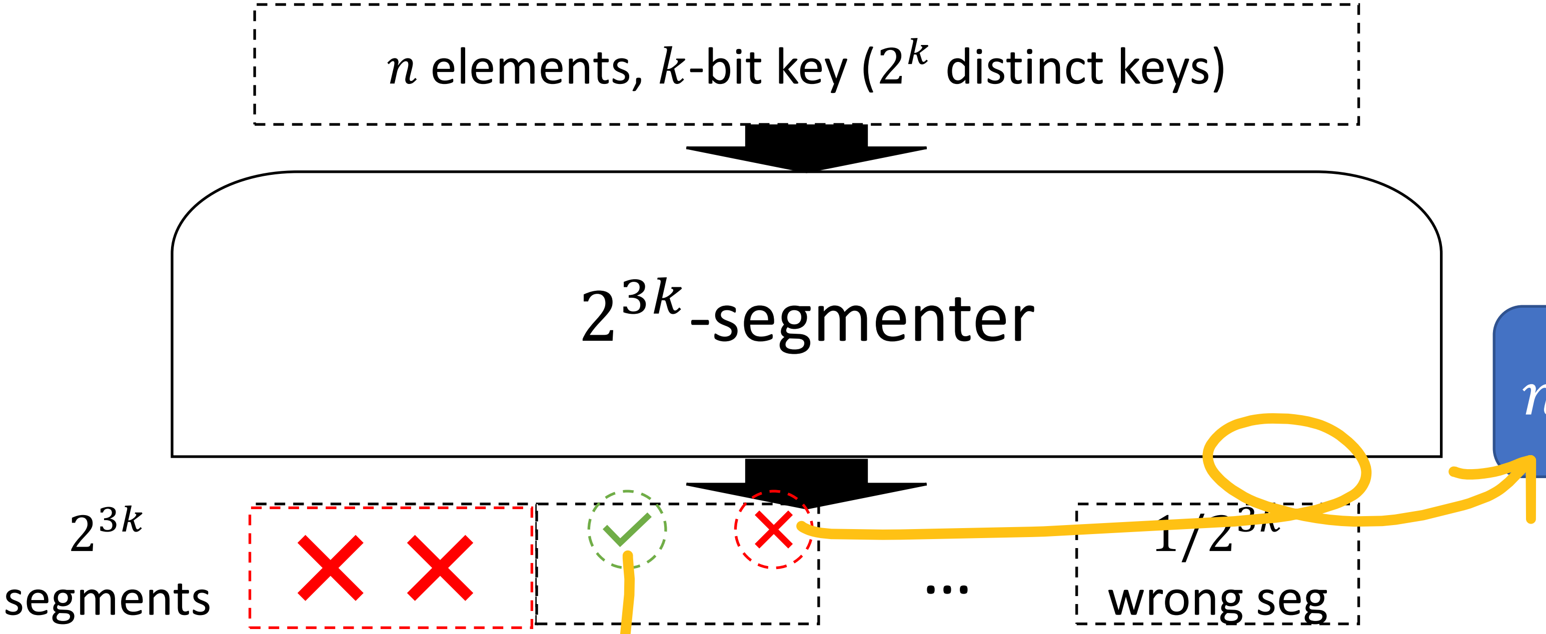
✕ Total depth = poly log


1-bit to k-bit upgrade
Our new abstraction: p-Segmenter
Input
2 4 0 4 3 0 2 1
p-Segmenter:
• Permute elements into p equal-size segments
• Elements are ordered between any two segments except for $1/p$ fraction
Output
2 0 2 4 3 0 4 1
Construction? Want:
Size $\sim n \cdot k$, **depth** $\sim \boldsymbol{\log n}$
Sets diff by $1/p$ fraction
Sets diff by $1/p$ fraction
Totally sorted p segments
0 0 1 2 2 3 4 4

1-bit to k-bit upgrade

Segmenter “errors”

1. Wrong segment
2. Correct segment, wrong position



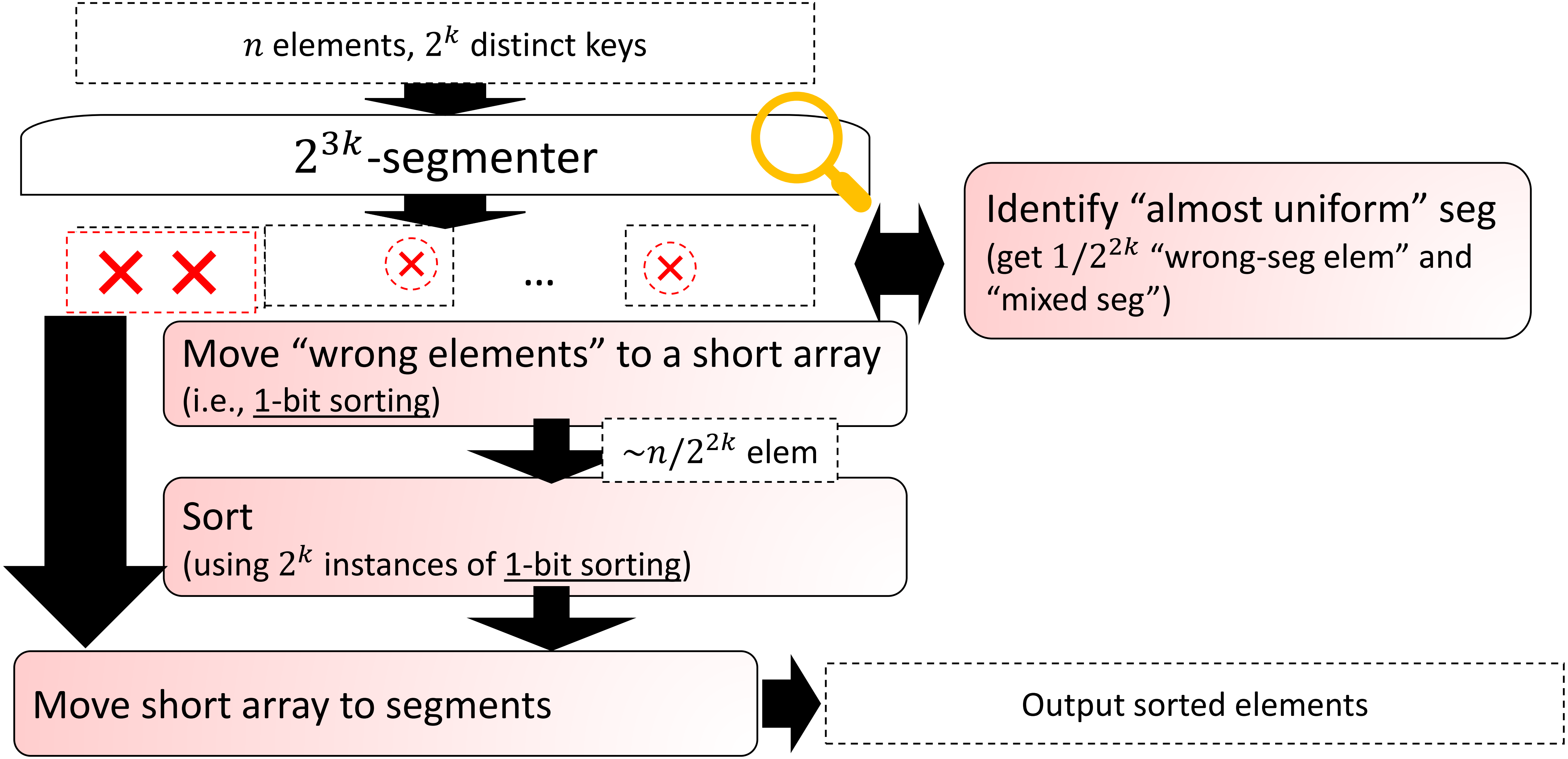
$n/2^{3k}$ elements in wrong seg

Element in correct seg, but **wrong position**? Sorting every seg is **too expensive**

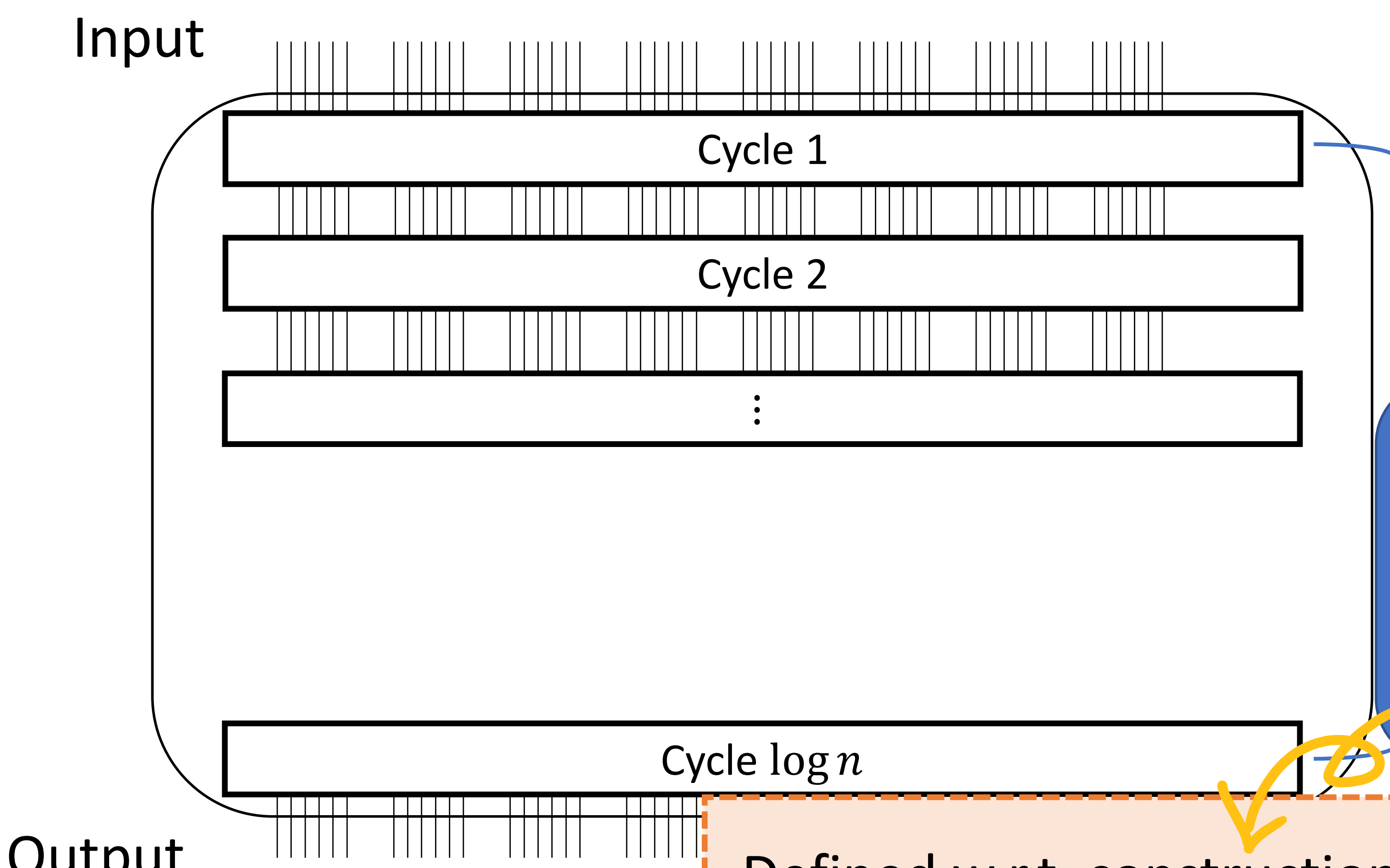
- 2^k distinct keys in 2^{3k} segments
  ➔ Almost all seg consist identical keys when totally sorted ☺
- 2^k (out of 2^{3k}) seg “mixed” ➔ wrong position ☹

➔➔ At most $n/2^{2k}$ wrong position

1-bit to k-bit upgrade | Segmenter + 1-bit sorting



Construct p-Segmenter | Revisit AKS sorting (log depth, parallel)



- $\log n$ "**cycles**"

Each cycle:

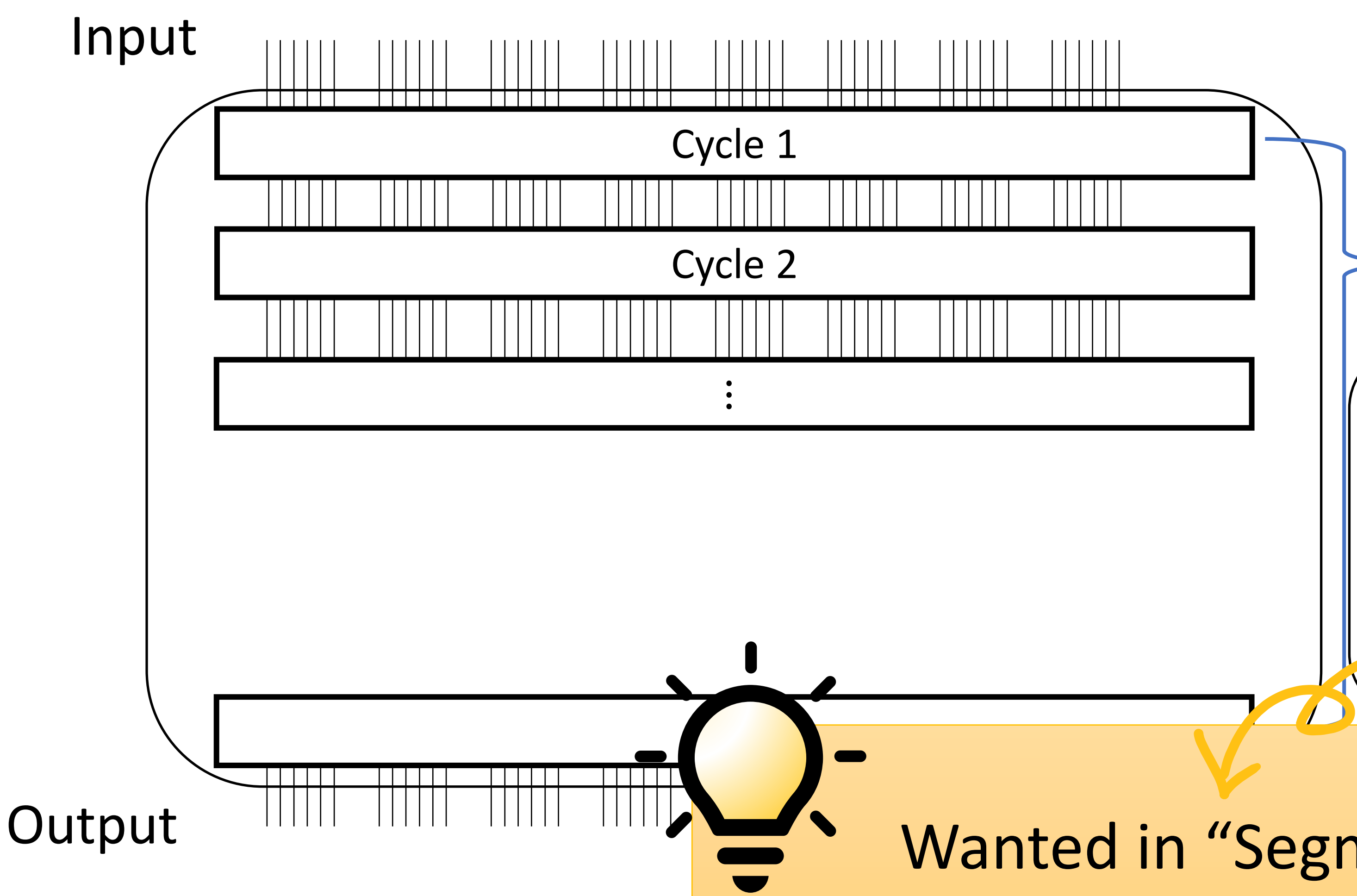
- Comparators
- Size $O(n)$, const depth

- Cycles diff in construction
- Each cycle "refine" outcome of previous cycle into "**more sorted**"

Defined w.r.t. construction of each cycle [AKS83, main lemma]

(skip here)

Construct p-Segmenter | Revisit AKS sorting (log depth, parallel)



- $\log n$ **"cycles"**

Each cycle:

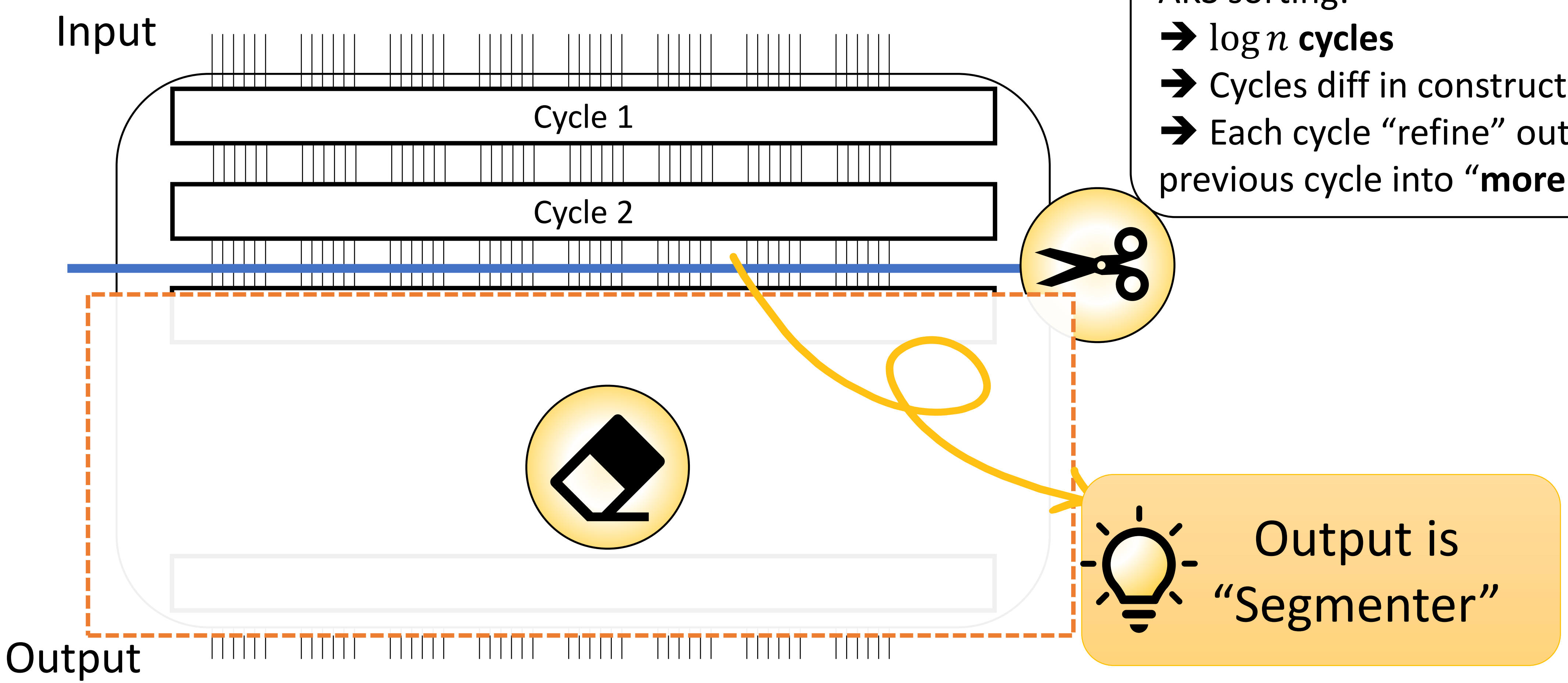
- Comparators
- Size $O(n)$, const depth

- Cycles diff in construction
- Each cycle "refine" outcome of previous cycle into "**more sorted**"

Wanted in "Segmenter"

Construct p-Segmenter

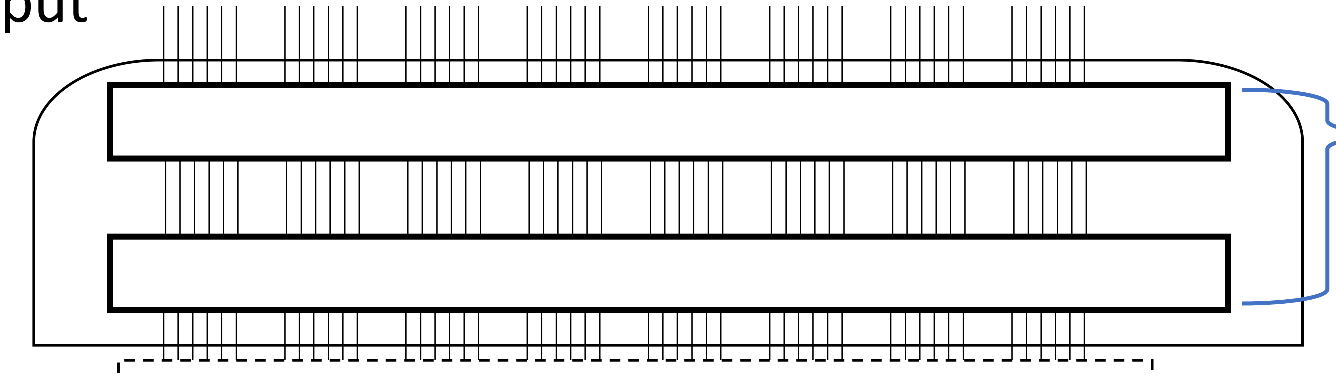
Segmenter based on AKS



AKS sorting:
- ➔ $\log n$ **cycles**
- ➔ Cycles diff in construction
- ➔ Each cycle “refine” outcome of previous cycle into “**more sorted**”

Construct p-Segmenter

Input



Output

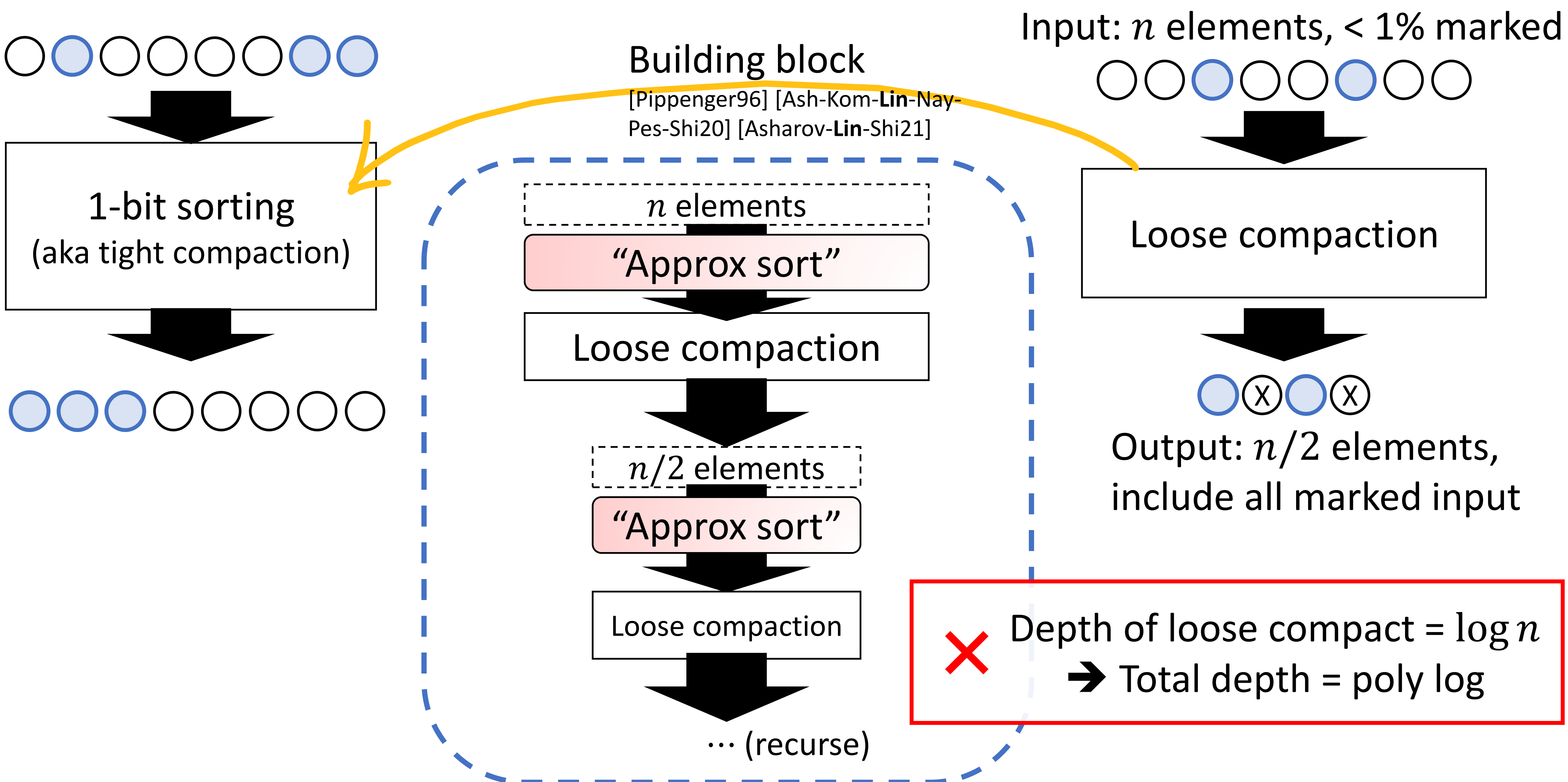
- 2^k equal-sized segments
- Ordered elements except for $1/2^k$ fraction

Construction of Segmenter:
First k "cycles" of AKS sorting
➔ 2^k-segmenter

2^k-segmenter (comparator network), taking size $O(n \cdot k)$, depth $O(k)$
("wrapping lemma" of [AKS83])

1-Bit sorting in $\log n$ depth circuit

Previous approaches



1-Bit sorting in $\log n$ depth circuit

Low depth "sparse" loose compaction

1-bit sorting
(aka tight compaction)

Building block
[Pippenger96]
[Ash-Kom-**Lin**-Pes-Shi20]

Improved construct:

- $poly \log n$-degree expander graph (const degree in Pippenger)
- Size: $O(n)$
- Depth: $O(\log n)$

+

"Repeated bootstrapping"
[Asharov-**Lin**-Shi21]

Input: n elements, $n/poly \log n$ marked

Sparse
Loose compaction

Output: $n/\log n$ elements, include all marked input

Previous:
Apply several loose compact…
➔ Depth > $\log n$

Epilogue: Reducing $\text{poly}\ \log^*$ to $\log^*$

This work:
size $O\big((k+w)\cdot n\cdot k\cdot poly(\log^* n-\log^*(k+w))\big)$, depth $O(\log n)$

[Koucký-Král21, concurrent]
size $O\big((k+w)\cdot n\cdot k\cdot (\log^* n-\log^*(k+w))\big)$, depth $O(\log^3 n)$

Better "repeated bootstrapping" technique

Putting together:
Sort n elements, k-bit key and w-bit payload,
circuit size $O\big((k+w)\cdot n\cdot k\cdot (\log^* n-\log^*(k+w))\big)$, depth $O(\log n)$

Conclusion and Open Problems

This talk:
Sort n elements, k-bit key and w-bit payload,
circuit size $O\big((k+w)\cdot n\cdot k\cdot(\log^* n-\log^*(k+w))\big)$, depth $O(\log n)$

Open problems:

- Get rid of $\log^*$? (better recursion?)
- Get rid of k? (beyond "indivisible"?)
- Conditional lower bounds?
- Improve segmenter / AKS cycles?

Thank you!