Improved Sublinear-Time Algorithms for Testing Order Pattern Freeness

Ilan Newman

אוניברסיטת חיפה
University of Haifa
جامعة حيفا

Nithin Varma

max planck institut informatik

Ordered Patterns in Arrays

100	98	723	1.2	5.68	3	1

- Let A be an array of length n and $\pi : [k] \to [k]$ be a bijection

Ordered Patterns in Arrays

100	98	723	1.2	5.68	3	1

- Let A be an array of length n and $\pi : [k] \to [k]$ be a bijection
- Array A has a π-appearance if
  $\exists$ indices $i_1 < \ldots < i_k$ such that $A[i_a] > A[i_b]$ if $\pi(a) > \pi(b) \quad \forall a, b \in [k]$

Ordered Patterns in Arrays

100	98	723	1.2	5.68	3	1

- Let A be an array of length n and $\pi : [k] \to [k]$ be a bijection
- Array A has a π-appearance if
  $\exists$ indices $i_1 < \ldots < i_k$ such that $A[i_a] > A[i_b]$ if $\pi(a) > \pi(b) \quad \forall a, b \in [k]$
- A is π-free if it has no π-appearance

Ordered Patterns in Arrays

100	98	723	1.2	5.68	3	1

- Let A be an array of length n and $\pi : [k] \to [k]$ be a bijection
- Array A has a π-appearance if
  $\exists$ indices $i_1 < \ldots < i_k$ such that $A[i_a] > A[i_b]$ if $\pi(a) > \pi(b) \quad \forall a, b \in [k]$
- A is π-free if it has no π-appearance
- Above array has a (3,4,1,2)-appearance but is (1,2,3,4)-free

Pattern freeness

- Well studied notion in combinatorics [Bona; J. Comb. Theory '99], [Arratia; Elec. J. Of Comb. '99], [Alon & Friedgut; J. Comb. Theory '00], [Marcus & Tardos; J. Comb. Theory '04]

Pattern freeness

- Well studied notion in combinatorics [Bona; J. Comb. Theory '99], [Arratia; Elec. J. Of Comb. '99], [Alon & Friedgut; J. Comb. Theory '00], [Marcus & Tardos; J. Comb. Theory '04]

- Several classical algorithms to decide π-freeness [Albert, Aldred, Atkinson & Holton; ISAAC '01], [Ahal & Rabinovich; SIDMA '08], [Guillemot & Marx; SODA '14], [Berendson, Kozma & Marx; Algorithmica '21]

Pattern freeness

- Well studied notion in combinatorics [Bona; J. Comb. Theory '99], [Arratia; Elec. J. Of Comb. '99], [Alon & Friedgut; J. Comb. Theory '00], [Marcus & Tardos; J. Comb. Theory '04]
- Several classical algorithms to decide π-freeness [Albert, Aldred, Atkinson & Holton; ISAAC '01], [Ahal & Rabinovich; SIDMA '08], [Guillemot & Marx; SODA '14], [Berendson, Kozma & Marx; Algorithmica '21]
- Motivated by detecting motifs and patterns in time series analysis [Berndt & Clifford; AAAI '94], [Keogh, Lonardi & Chiu; SIGKDD '02]

Pattern freeness

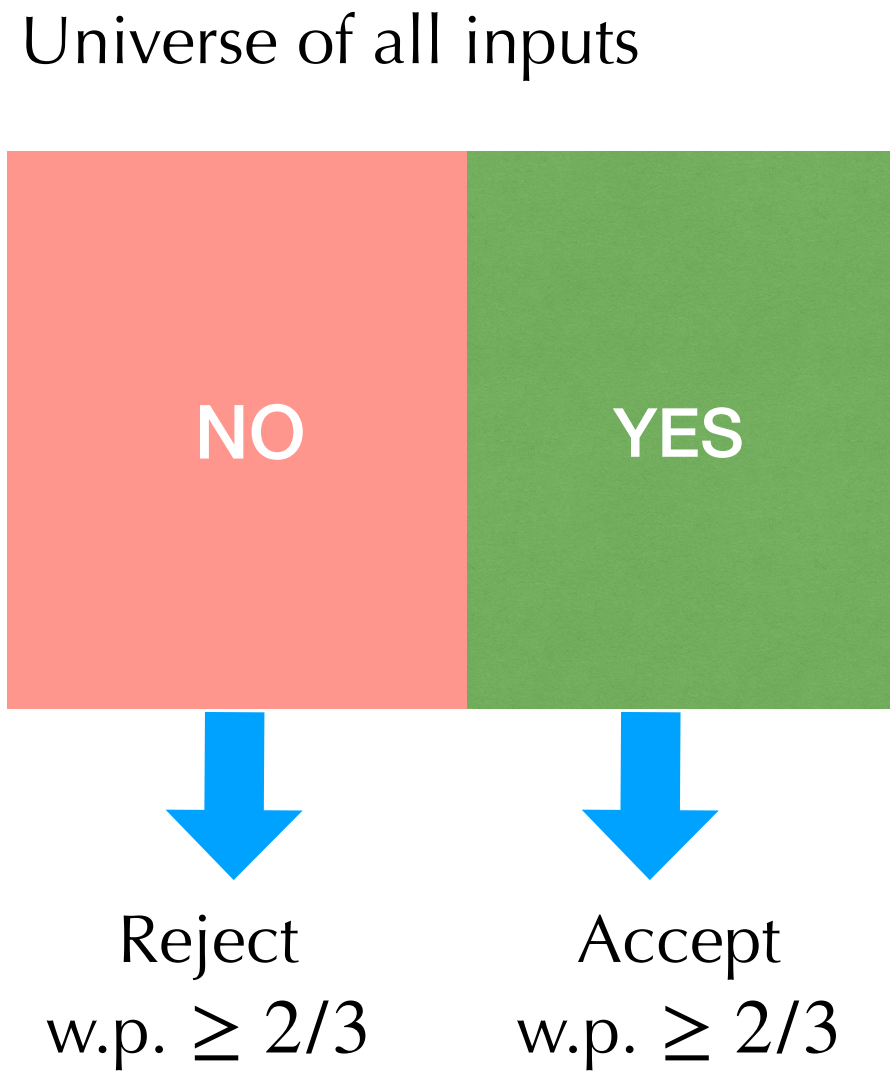
- Well studied notion in combinatorics [Bona; J. Comb. Theory '99], [Arratia; Elec. J. Of Comb. '99], [Alon & Friedgut; J. Comb. Theory '00], [Marcus & Tardos; J. Comb. Theory '04]
- Several classical algorithms to decide π-freeness[Albert, Aldred, Atkinson & Holton; ISAAC '01], [Ahal & Rabinovich; SIDMA '08], [Guillemot & Marx; SODA '14], in particular a linear time algorithm [Berendson, Kozma & Marx; Algorithmica '21]
- Motivated by detecting motifs and patterns in time series analysis [Berndt & Clifford; AAAI '94], [Keogh, Lonardi & Chiu; SIGKDD '02]
- Connections to Longest Increasing Subsequence [Newman & V.; ICALP '21]

Pattern freeness

- Well studied notion in combinatorics [Bona; J. Comb. Theory '99], [Arratia; Elec. J. Of Comb. '99], [Alon & Friedgut; J. Comb. Theory '00], [Marcus & Tardos; J. Comb. Theory '04]
- Several classical algorithms to decide π-freeness[Albert, Aldred, Atkinson & Holton; ISAAC '01], [Ahal & Rabinovich; SIDMA '08], [Guillemot & Marx; SODA '14], in particular a linear time algorithm [Berendson, Kozma & Marx; Algorithmica '21]
- Motivated by detecting motifs and patterns in time series analysis [Berndt & Clifford; AAAI '94], [Keogh, Lonardi & Chiu; SIGKDD '02]
- Connections to Longest Increasing Subsequence [Newman & **V.**; ICALP '21]

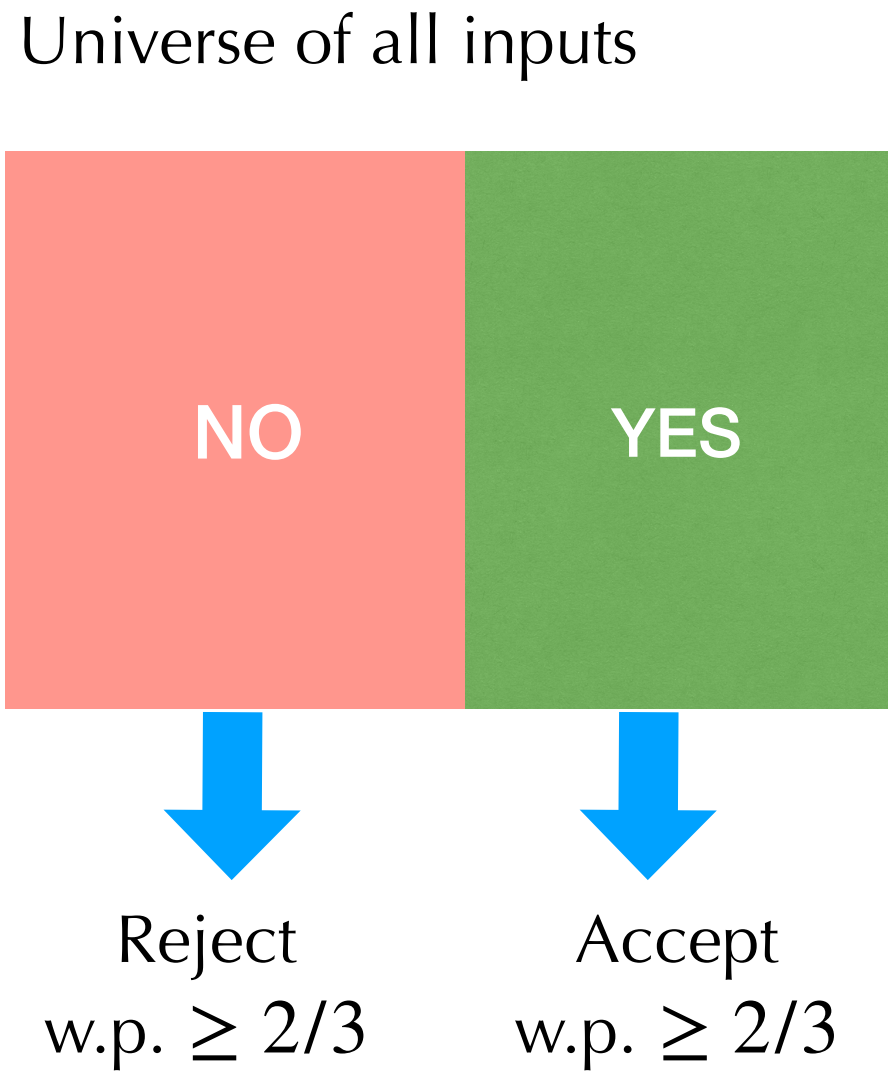
Today: Sublinear-Time Algorithms for (approximate) Pattern Freeness

Decision Problem



Decision Problem

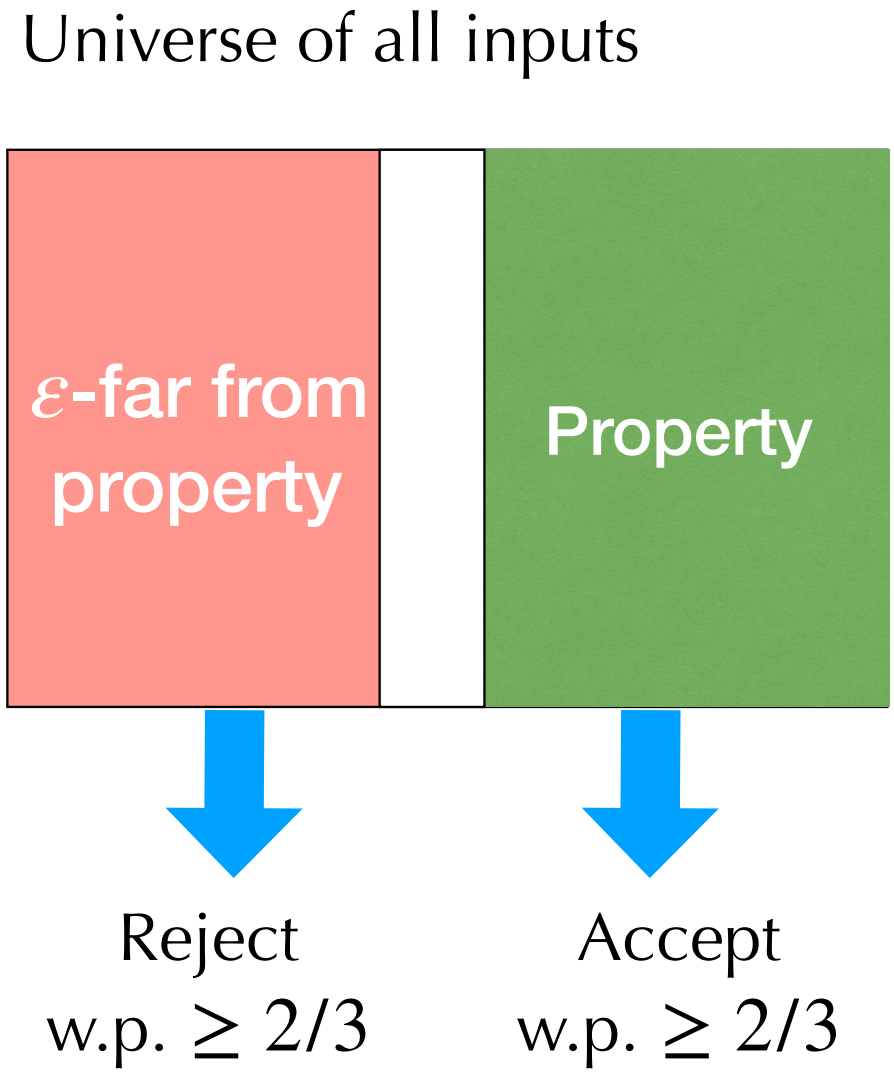
- Cannot exactly solve nontrivial decision problems without full access to the input



Property testing

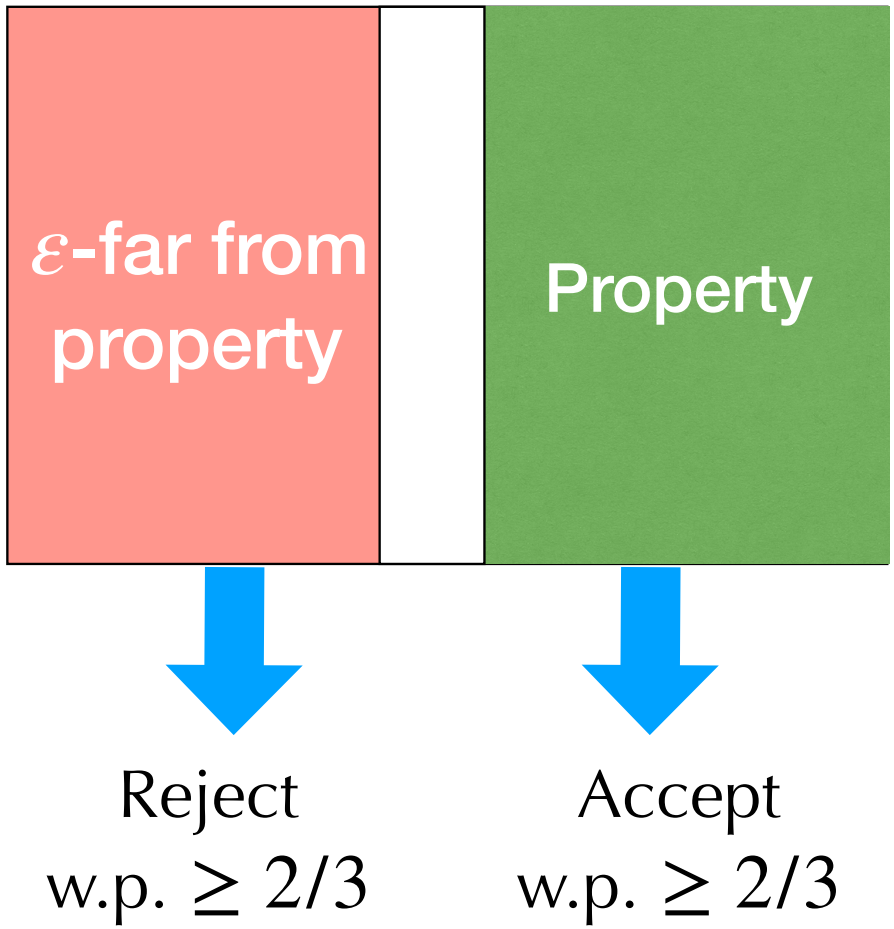
[Rubinfeld & Sudan; **SODA 92 & SICOMP 96**, Goldreich, Goldwasser & Ron; **FOCS 96 & JACM 98**]

- ε-far from property: At least ε fraction of input values need to be changed to satisfy the property



Property testing

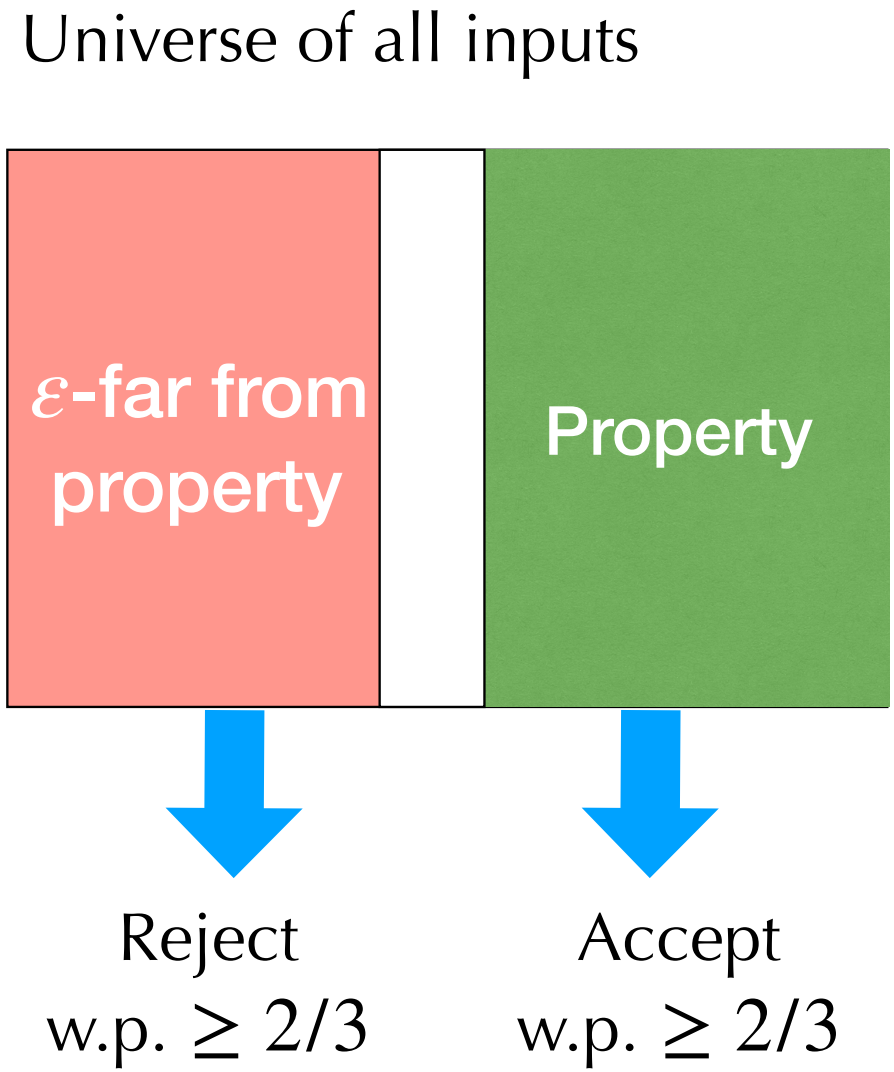
[Rubinfeld & Sudan; **SODA 92 & SICOMP 96**,
Goldreich, Goldwasser & Ron; **FOCS 96 & JACM 98**]



Property testing

[Rubinfeld & Sudan; **SODA 92 & SICOMP 96**, Goldreich, Goldwasser & Ron; **FOCS 96 & JACM 98**]

- Several well-studied and fundamental properties
  - Bipartiteness [Alon, Krivelevich; SICOMP '02, …]
  - Monotonicity [Chakrabarty Seshadhri; STOC '13, Khot, Minzer, Safra; SICOMP '18, …]
  - Convexity of images [Berman Murzabulatov Raskhodnikova; RSA '19, …]
  - Linearity and related properties [Blum Luby Rubinfeld; JCSS '93, …]



Testing Pattern Freeness

- Given query access to an array A of length n, a parameter $\epsilon \in (0,1)$, and permutation π, decide whether
  - A is π-free, OR
  - A is ϵ-far from π-free

Testing Pattern Freeness

- Given query access to an array A of length n, a parameter $\epsilon \in (0,1)$, and permutation π, decide whether
  - A is π-free, OR
  - A is ϵ-far from π-free

Hamming distance of A to every π-free array is at least ϵn

Testing Pattern Freeness

- Given query access to an array A of length n, a parameter $\epsilon \in (0,1)$, and permutation π, decide whether
  - A is π-free, OR
  - A is ϵ-far from π-free

Hamming distance of A to every π-free array is at least ϵn

Generalization of monotonicity testing of arrays [EKKRV00, DGLRRS99, F04, …]

Testing Pattern Freeness

- Given query access to an array A of length n, a parameter $\epsilon \in (0,1)$, and permutation π, decide whether
  - A is π-free, OR
  - A is ϵ-far from π-free

Hamming distance of A to every π-free array is at least ϵn

Monotonicity $\equiv (2,1)$-freeness

Generalization of monotonicity testing of arrays [EKKRV00, DGLRRS99, F04, …]

Testing Pattern Freeness

- Given query access to an array A of length n, a parameter $\epsilon \in (0,1)$, and permutation π, decide whether
  - A is π-free, OR
  - A is ϵ-far from π-free

Hamming distance of A to every π-free array is at least ϵn

Algorithm solving this problem is called ϵ-tester for π-freeness

Testing Pattern Freeness

- Given query access to an array A of length n, a parameter $\epsilon \in (0,1)$, and permutation π, decide whether
  - A is π-free, OR
  - A is ϵ-far from π-free

Hamming distance of A to every π-free array is at least ϵn

Algorithm solving this problem is called ϵ-tester for π-freeness

Tester adaptive if its queries depend on answers to previous queries

Tester nonadaptive otherwise

Testing Pattern Freeness

- **Special case**: Sortedness testing of arrays of length n [Ergun Kannan Kumar Rubinfeld Vishwanathan 00, Fischer 04, …]

Testing Pattern Freeness

- **Special case**: Sortedness testing of arrays of length n [Ergun Kannan Kumar Rubinfeld Vishwanathan 00, Fischer 04, …]

- [Newman Rabinovich Rajendraprasad Sohler 19] initiated the study of sublinear-time algorithms for testing pattern freeness for larger patterns

Testing Pattern Freeness

- **Special case**: Sortedness testing of arrays of length n [Ergun Kannan Kumar Rubinfeld Vishwanathan 00, Fischer 04, …]
- [Newman Rabinovich Rajendraprasad Sohler 19] initiated the study of sublinear-time algorithms for testing pattern freeness for larger patterns
- Studied in depth for monotone patterns $(1,2,\ldots,k)$ or $(k,k-1,\ldots,1)$

  [NRRS19, BenEliezer Canonne Letzter Waingarten 19, BenEliezer Letzter Waingarten 19]

Highlights of Past Work

- Sortedness testing of arrays of length n can be done using $\Theta(\log n)$ queries [EKKRV00, F04]

Highlights of Past Work

- Sortedness testing of arrays of length n can be done using $\Theta(\log n)$ queries [EKKRV00, F04]
- $O(\log n)$-query tester for monotone patterns of constant length [BLW19]
- polylog n query tester for arbitrary patterns of length 3 [NRRS19]

Highlights of Past Work

- Sortedness testing of arrays of length n can be done using $\Theta(\log n)$ queries [EKKRV00, F04]
- $O(\log n)$-query tester for monotone patterns of constant length [BLW19]
- polylog n query tester for arbitrary patterns of length 3 [NRRS19]

What about nonmonotone patterns of length > 3?

Highlights of Past Work

- Sortedness testing of arrays of length n can be done using $\Theta(\log n)$ queries [EKKRV00, F04]
- $O(\log n)$-query tester for monotone patterns of constant length [BLW19]
- polylog n query tester for arbitrary patterns of length 3 [NRRS19]

What about nonmonotone patterns of length > 3?

- Nonadaptive testers making $O(n^{1-\frac{1}{k-1}})$ queries [NRRS19]
- Nonadaptive testers cannot do better! [NRRS19, BenEliezer Canonne 18]

Highlights of Past Work

- Sortedness testing of arrays of length n can be done using $\Theta(\log n)$ queries [EKKRV00, F04]
- $O(\log n)$-query tester for monotone patterns of constant length [BLW19]
- polylog n query tester for arbitrary patterns of length 3 [NRRS19]

What about nonmonotone patterns of length > 3?

- Nonadaptive testers making $O(n^{1-\frac{1}{k-1}})$ queries [NRRS19]
- Nonadaptive testers cannot do better! [NRRS19, BenEliezer Canonne 18]

What about adaptive testers ?

Our Result

Let $\epsilon \in (0,1), k \in \mathbb{N}, \pi \in \mathbb{S}_k$. There is an ϵ-tester for π-freeness

- with query complexity $\tilde{O}(n^{o(1)})$
- that always accepts π-free arrays.

Our Result

Let $\epsilon \in (0,1), k \in \mathbb{N}, \pi \in \mathbb{S}_k$. There is an ϵ-tester for π-freeness

- with query complexity $\tilde{O}(n^{o(1)})$
- that always accepts π-free arrays.

- Improved sublinear-time guarantee
- Our techniques are general and do not rely on the structure of the pattern being considered (e.g., as opposed to [NRRS19])

Today

- $\tilde{O}(\sqrt{n})$-query algorithm to test π-freeness of $\pi \in \mathbb{S}_4$

Today

- $\tilde{O}(\sqrt{n})$-query algorithm to test π-freeness of $\pi \in \mathbb{S}_4$
- [BC18] Every nonadaptive algorithm has query complexity $\Omega(n^{2/3})$

Today

- $\tilde{O}(\sqrt{n})$-query algorithm to test π-freeness of $\pi \in \mathbb{S}_4$
- [BC18] Every nonadaptive algorithm has query complexity $\Omega(n^{2/3})$

Goal: Design an algorithm that makes $\tilde{O}(\sqrt{n})$ queries and finds a π-appearance in an array that is ϵ-far from π-free

First Useful Fact

- Array A is ϵ-far from being π-free $\Longrightarrow$ there are $\geq \frac{\epsilon n}{4}$ disjoint π-appearances

First Useful Fact

- Array A is ϵ-far from being π-free $\Longrightarrow$ there are $\geq \frac{\epsilon n}{4}$ disjoint π-appearances
- Used by all algorithms to test π-freeness

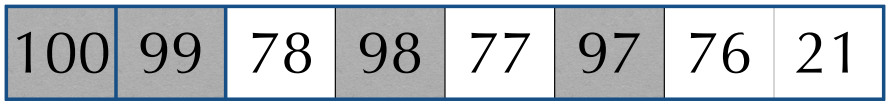
First Useful Fact

- Array A is ϵ-far from being π-free $\Longrightarrow$ there are $\geq \frac{\epsilon n}{4}$ disjoint π-appearances
- Used by all algorithms to test π-freeness

100	99	78	98	77	97	76	21

First Useful Fact

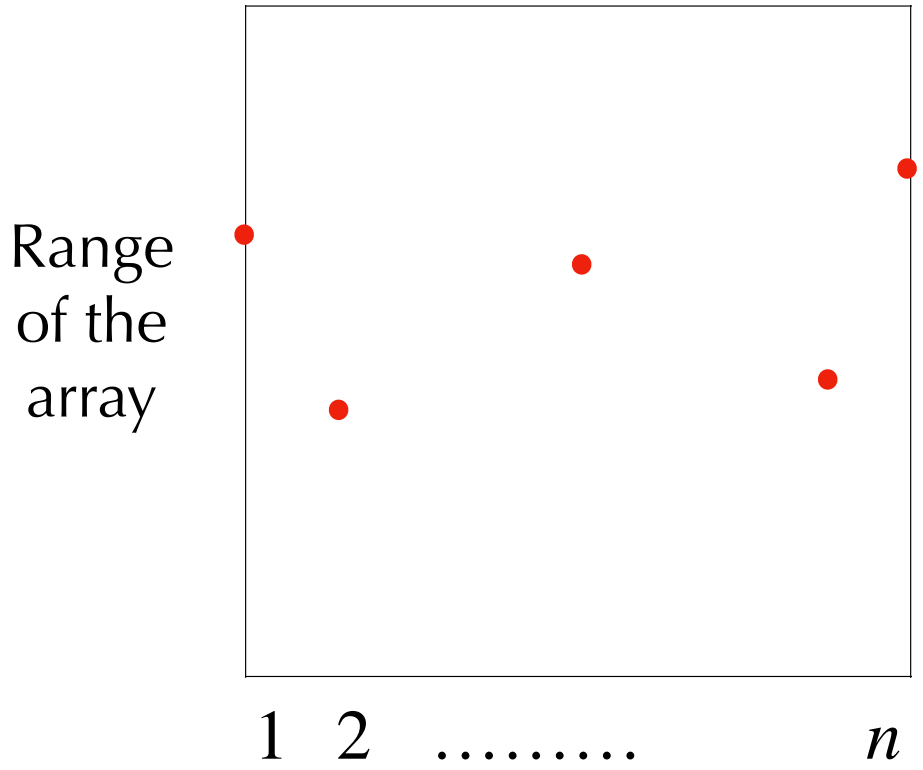
- Array A is ϵ-far from being π-free $\Longrightarrow$ there are $\geq \frac{\epsilon n}{4}$ disjoint π-appearances
- Used by all algorithms to test π-freeness



Disjoint (4,3,2,1)-appearances

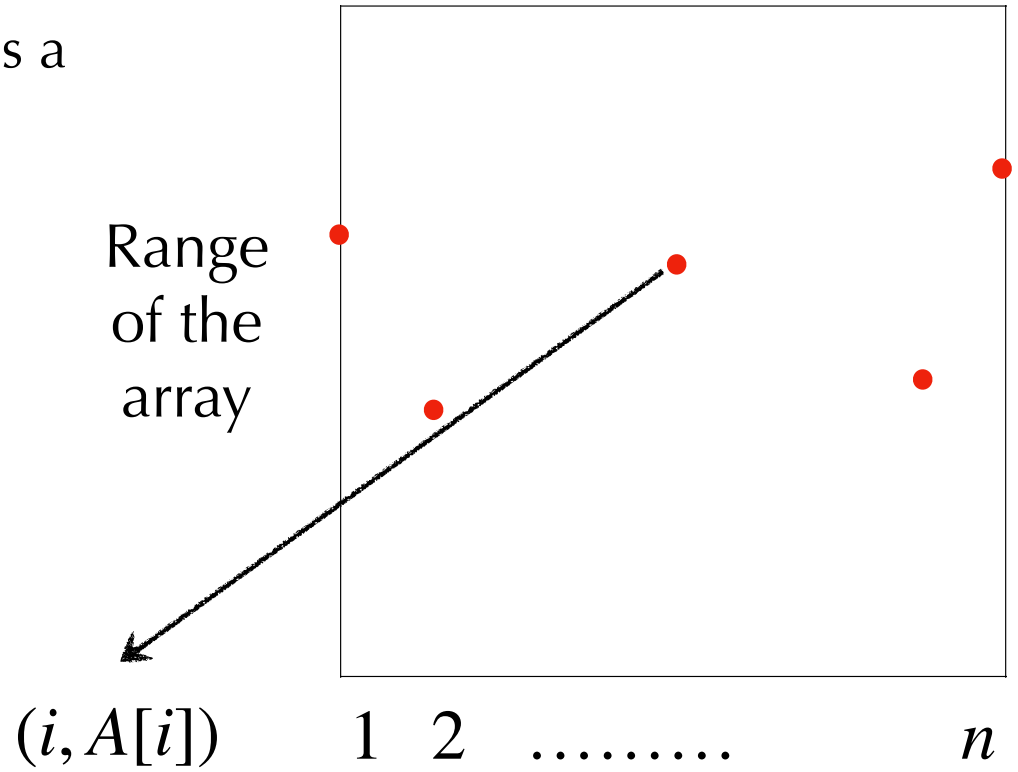
Gridding

- **(Conceptual) Step 1**: View the array as a box of n points in $[n] \times \mathbb{R}$



Gridding

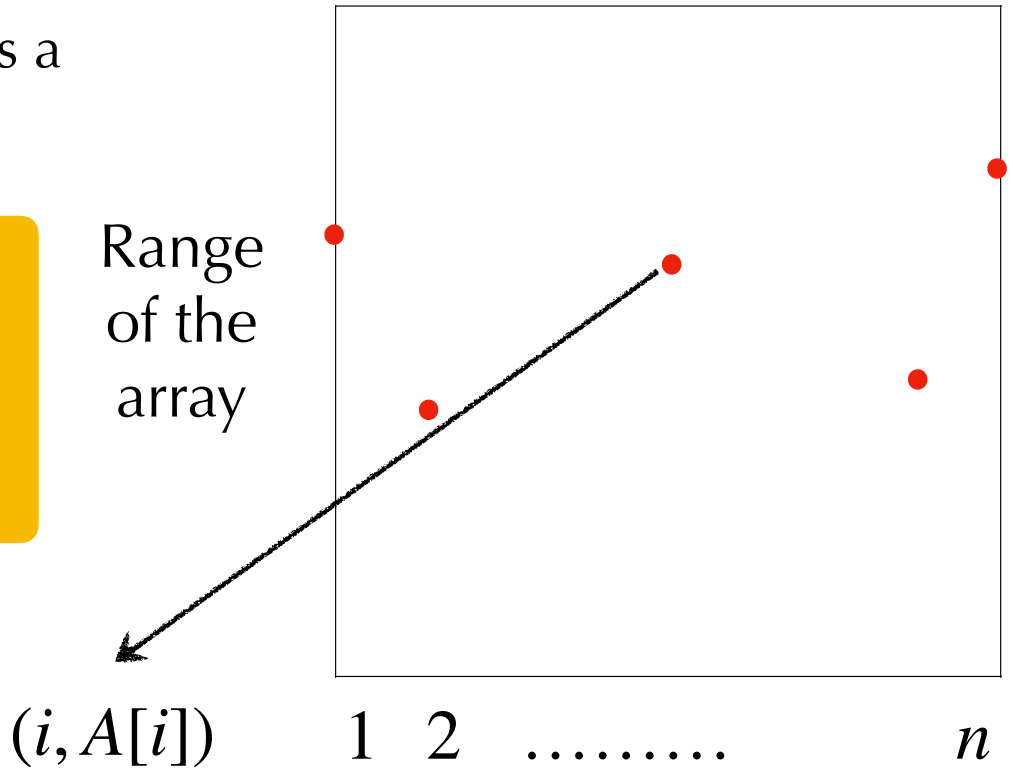
- **(Conceptual) Step 1**: View the array as a box of n points in $[n] \times \mathbb{R}$



Gridding

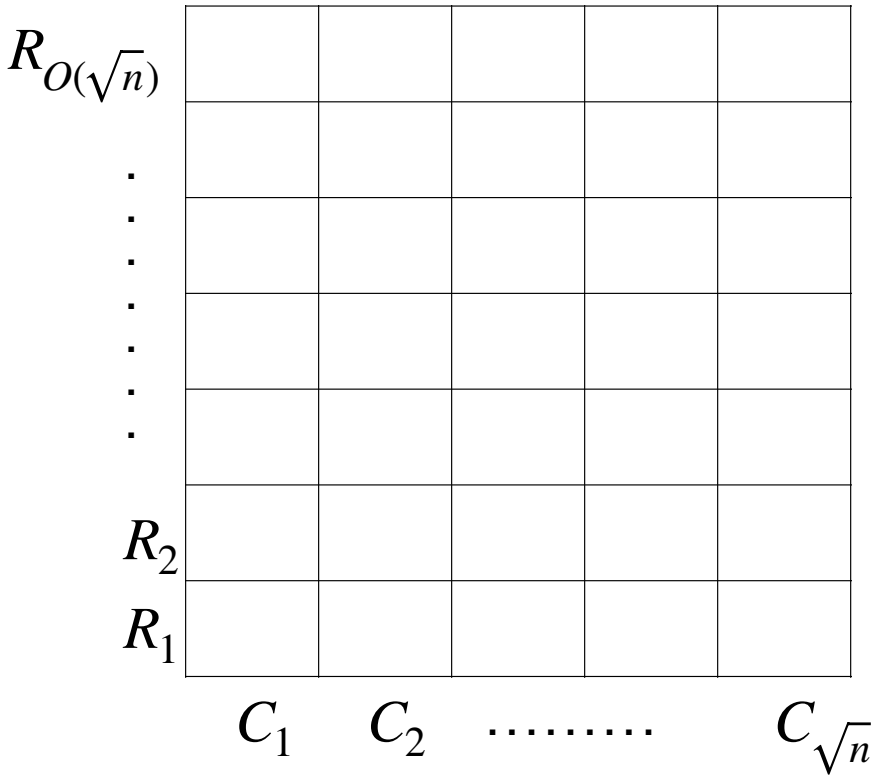
- **(Conceptual) Step 1**: View the array as a box of n points in $[n] \times \mathbb{R}$

Goal: Detect any one of the $\geq \frac{\epsilon n}{4}$ disjoint π-appearances present among the points



Gridding

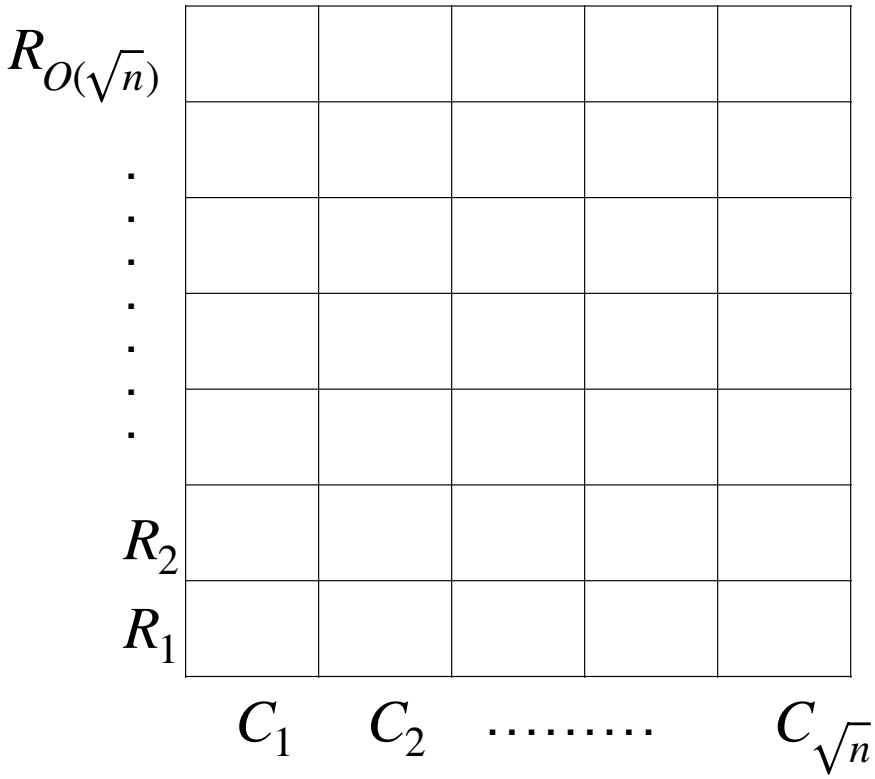
- **(Conceptual) Step 1**: View the array as a box of n points in $[n] \times \mathbb{R}$
- **Step 2**: Query $\tilde{O}(\sqrt{n})$ uniform indices to approximate the box by a coarse $O(\sqrt{n}) \times \sqrt{n}$ grid of sub-boxes



Gridding

- **(Conceptual) Step 1**: View the array as a box of n points in $[n] \times \mathbb{R}$
- **Step 2**: Query $\tilde{O}(\sqrt{n})$ uniform indices to approximate the box by a coarse $O(\sqrt{n}) \times \sqrt{n}$ grid of sub-boxes

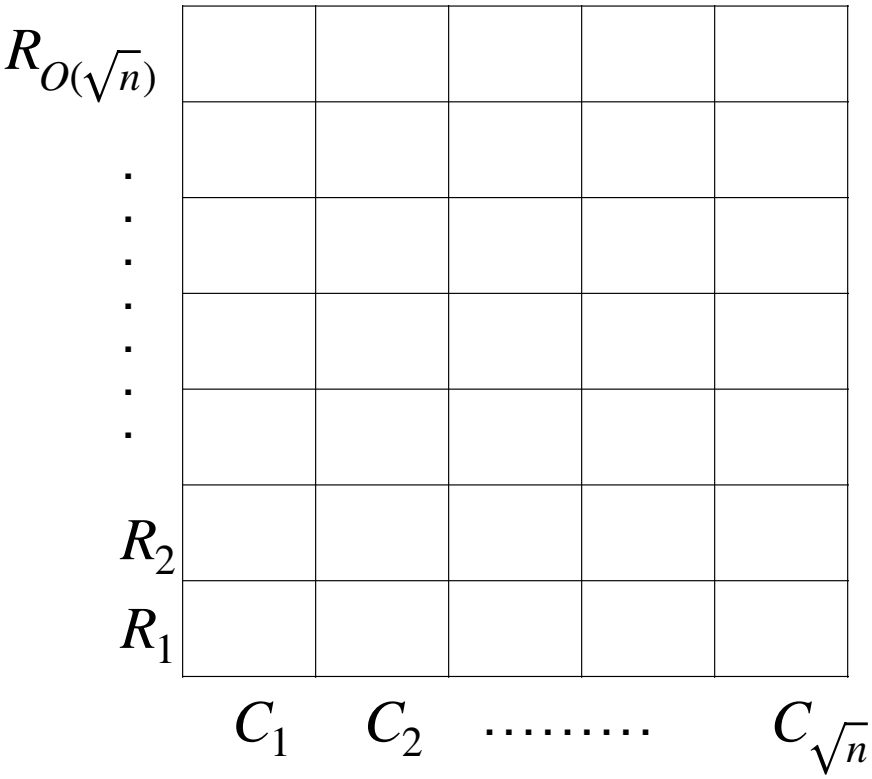
We ensure that rows and columns roughly equipartition the set of array points



Gridding

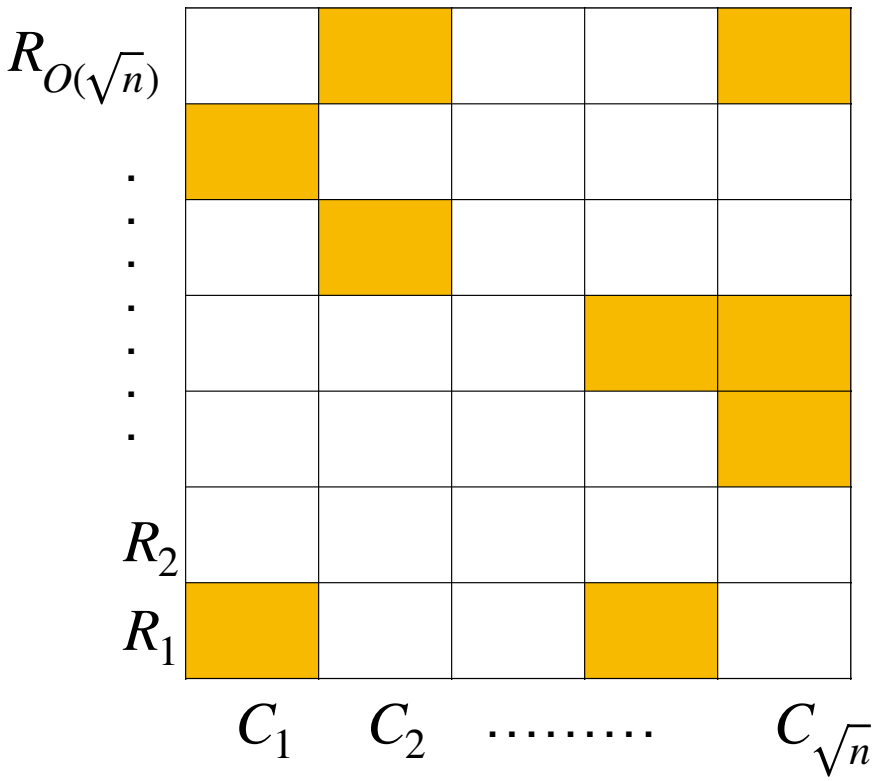
- **(Conceptual) Step 1**: View the array as a box of n points in $[n] \times \mathbb{R}$
- **Step 2**: Query $\tilde{O}(\sqrt{n})$ uniform indices to approximate the box by a coarse $O(\sqrt{n}) \times \sqrt{n}$ grid of sub-boxes

Each row has $\sim \sqrt{n}$ points
Each column has $\sqrt{n}$ points



Gridding: Part 2

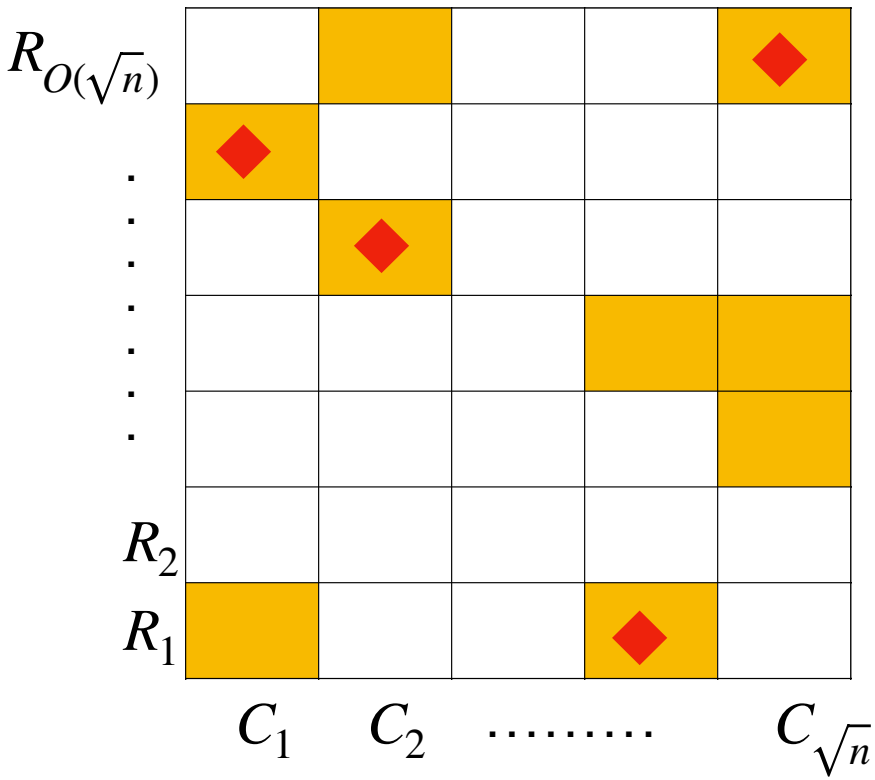
- From each column, sample $\tilde{O}(1)$ uniformly random points
- Tag a box as **nonempty** if it has at least one sampled point; tag it **dense** if it has a constant fraction of sampled points
- There are $O(n)$ boxes, out of which we only tag $\tilde{O}(\sqrt{n})$ boxes



Gridding: Part 2

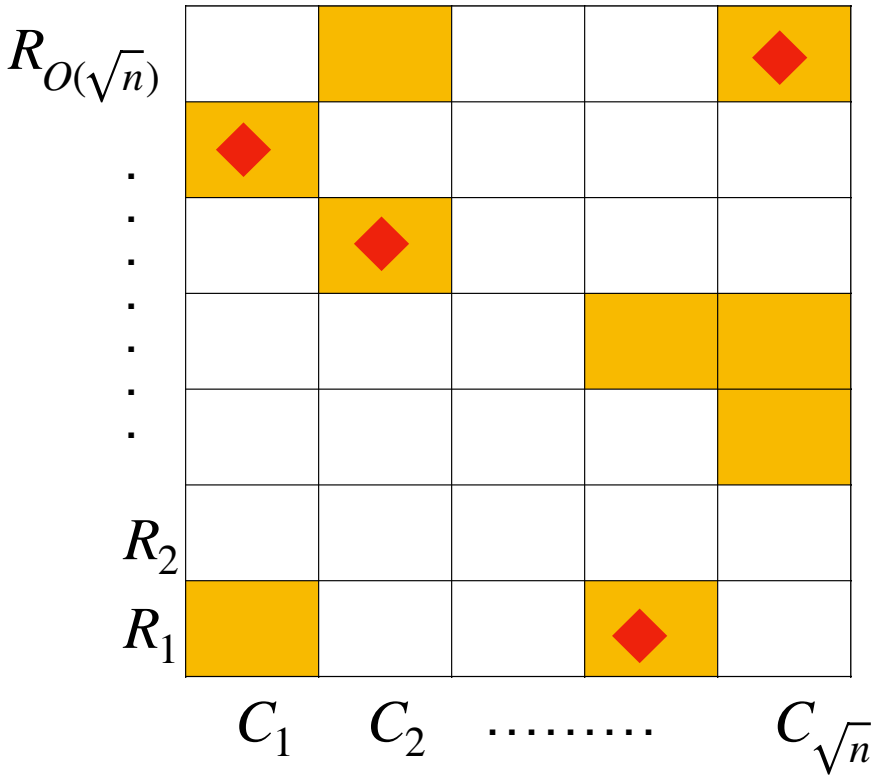
- From each column, sample $\tilde{O}(1)$ uniformly random points
- Tag a box as **nonempty** if it has at least one sampled point; tag it **dense** if it has a constant fraction of sampled points
- There are $O(n)$ boxes, out of which we only tag $\tilde{O}(\sqrt{n})$ boxes

If $\pi = (3,2,1,4)$, we already found a π-appearance



In general, if too many nonempty boxes…

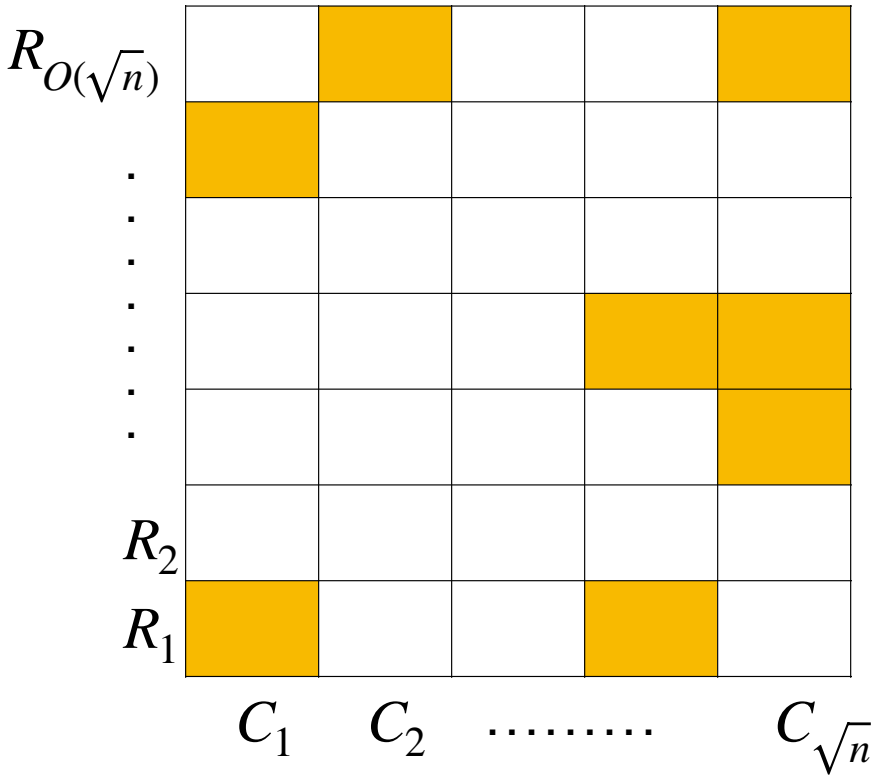
- **[Markus & Tardos '04]** For each permutation π, there exists a constant κ such that such that if the grid on the right has more than $\kappa\sqrt{n}$ nonempty boxes, then there is a π-appearance among the nonempty boxes



If gridding does not find a π-appearance

We show that:

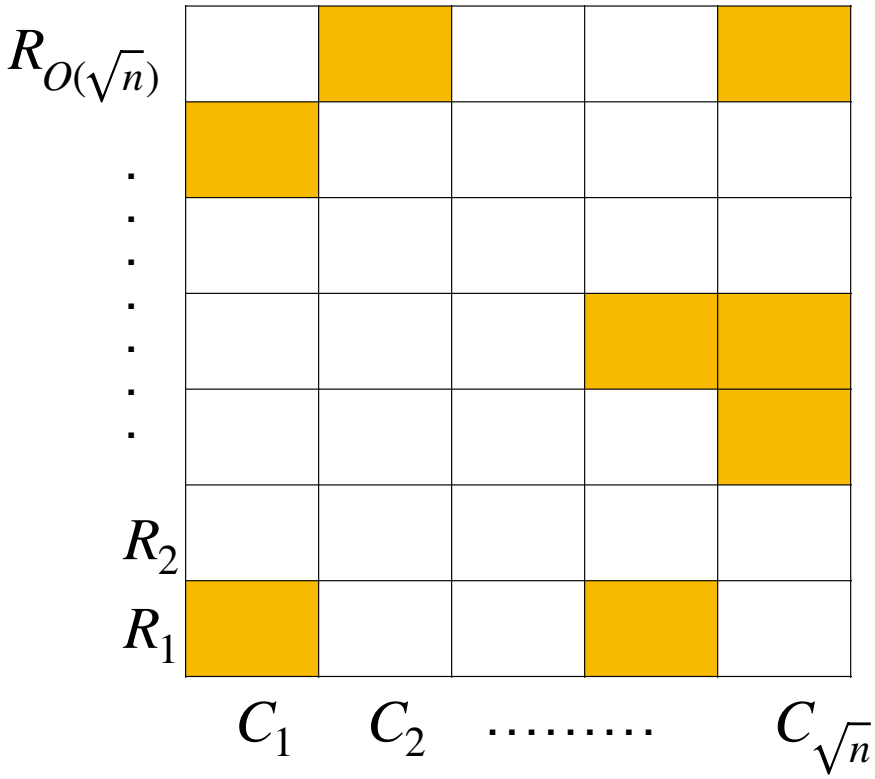
- Only $O(\sqrt{n})$ nonempty boxes. In particular, only $O(\sqrt{n})$ dense boxes



If gridding does not find a π-appearance

We show that:

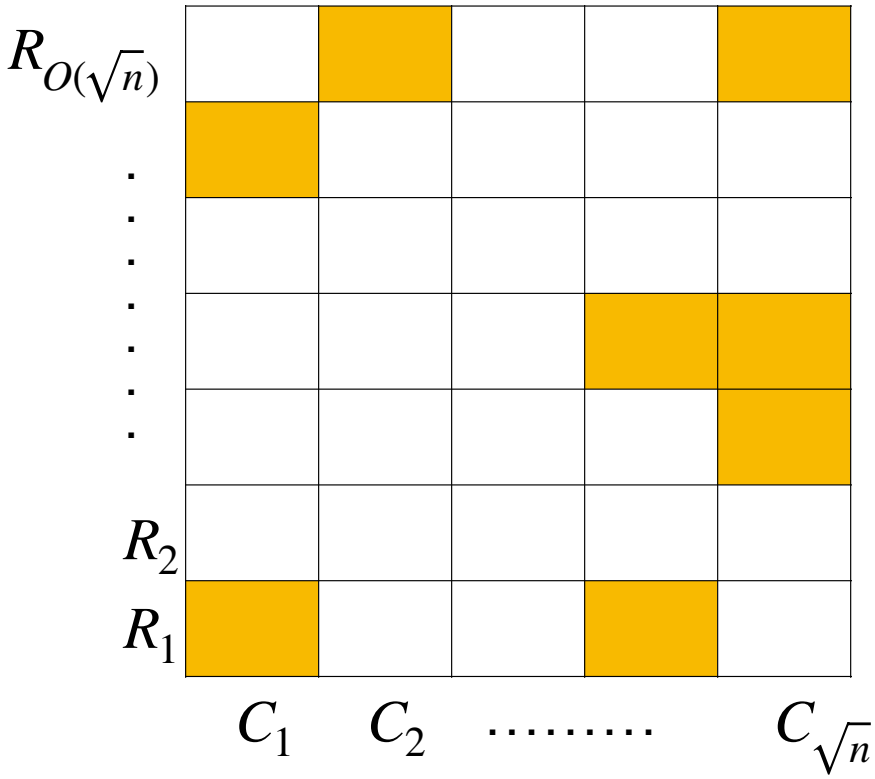
- Only $O(\sqrt{n})$ nonempty boxes. In particular, only $O(\sqrt{n})$ dense boxes
- Each row and column contains $O(1)$ dense boxes each



If gridding does not find a π-appearance

We show that:

- Only $O(\sqrt{n})$ nonempty boxes. In particular, only $O(\sqrt{n})$ dense boxes
- Each row and column contains $O(1)$ dense boxes each
- The union of points in all the dense boxes contains $\Omega(\epsilon n)$ disjoint π-appearances

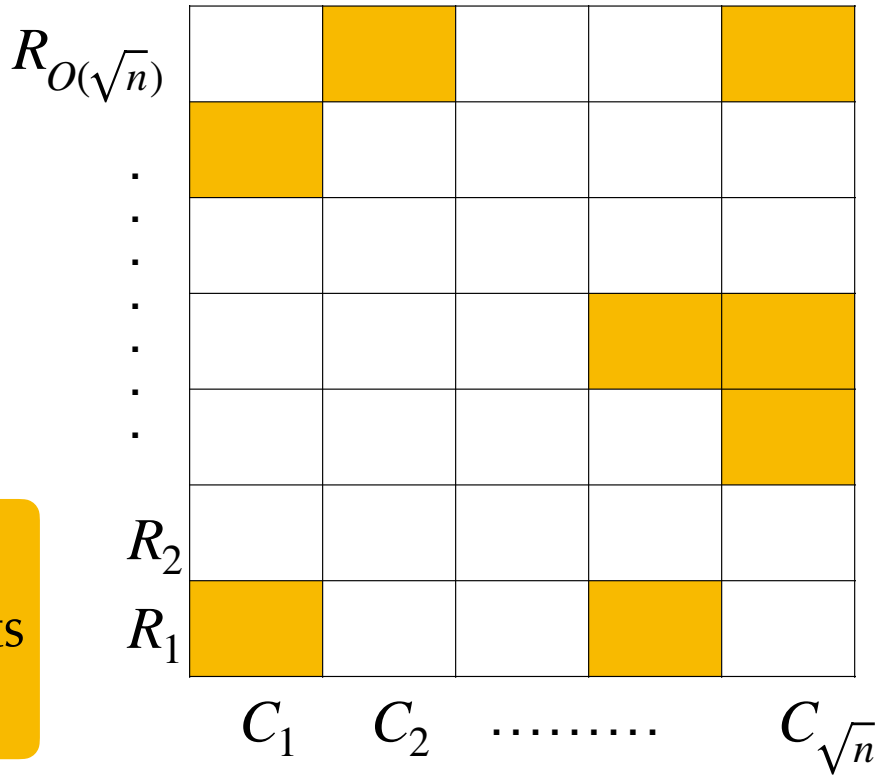


If gridding does not find a π-appearance

We show that:

- Only $O(\sqrt{n})$ nonempty boxes. In particular, only $O(\sqrt{n})$ dense boxes
- Each row and column contains $O(1)$ dense boxes each
- The union of points in all the dense boxes contains $\Omega(\epsilon n)$ disjoint π-appearances

Ideas in Proof: Combinatorial lemma by [MarcusTardos04] + Averaging arguments + Properties of the sampling procedure

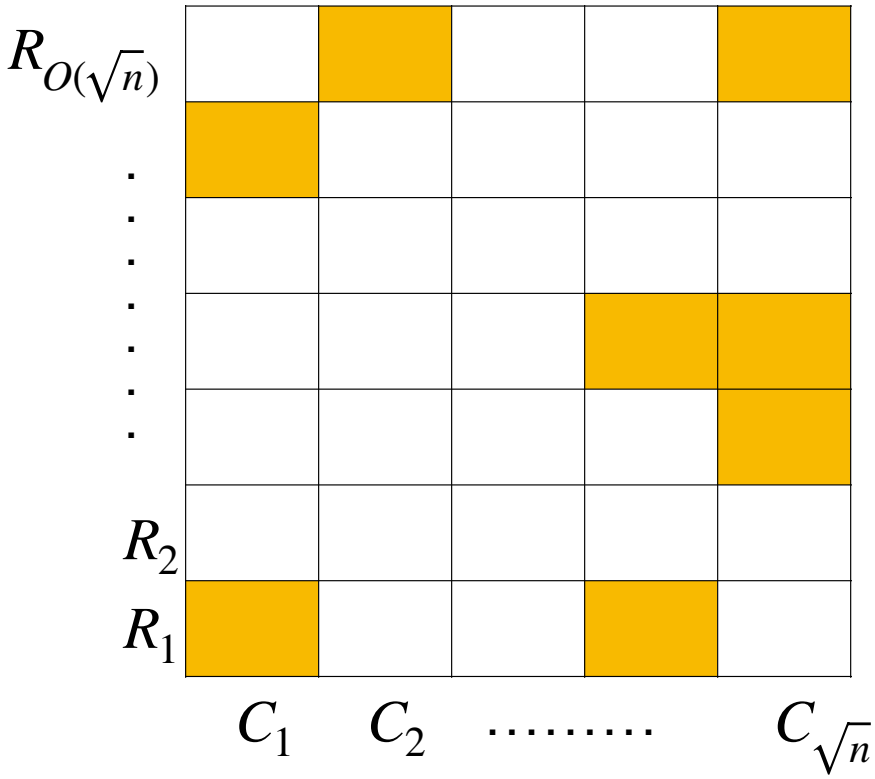


If gridding does not find a π-appearance

We show that:

- Only $O(\sqrt{n})$ nonempty boxes. In particular, only $O(\sqrt{n})$ dense boxes
- Each row and column contains $O(1)$ dense boxes each
- The union of points in all the dense boxes contains $\Omega(\epsilon n)$ disjoint π-appearances

Where in the grid do these appear?

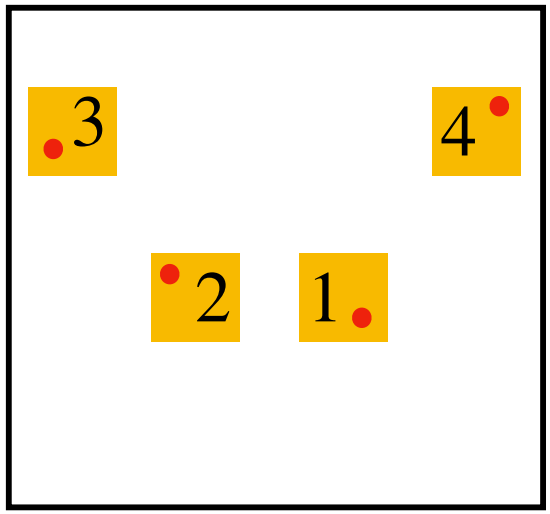


If gridding does not find a π-appearance

$\Omega(\epsilon n)$ disjoint π-appearances have their "**legs**" in dense boxes arranged according to any one of the constantly many types of "**configurations**"

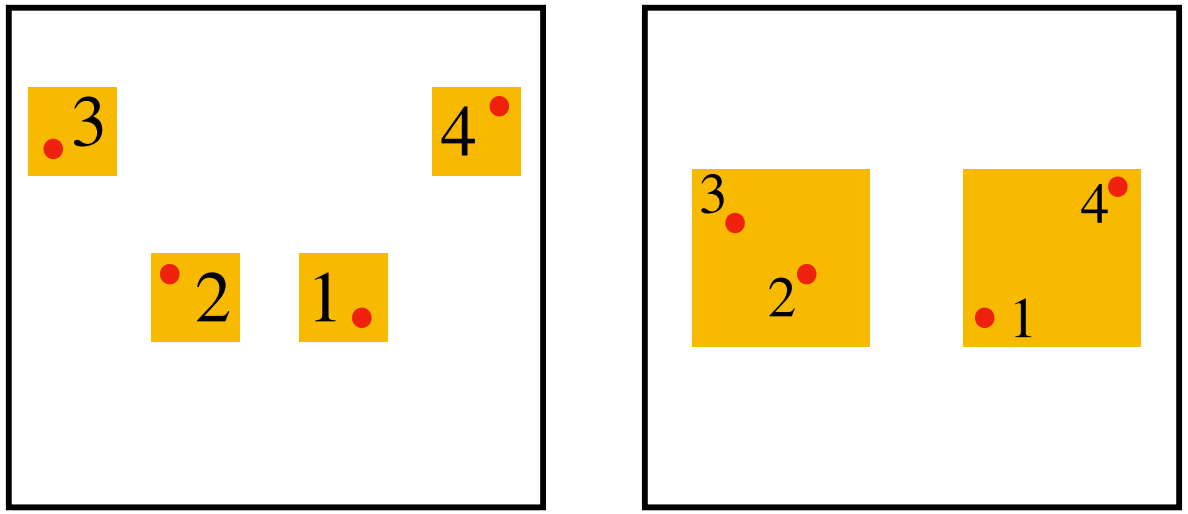
If gridding does not find a π-appearance

$\Omega(\epsilon n)$ disjoint π-appearances have their "**legs**" in dense boxes arranged according to any one of the constantly many types of "**configurations**"



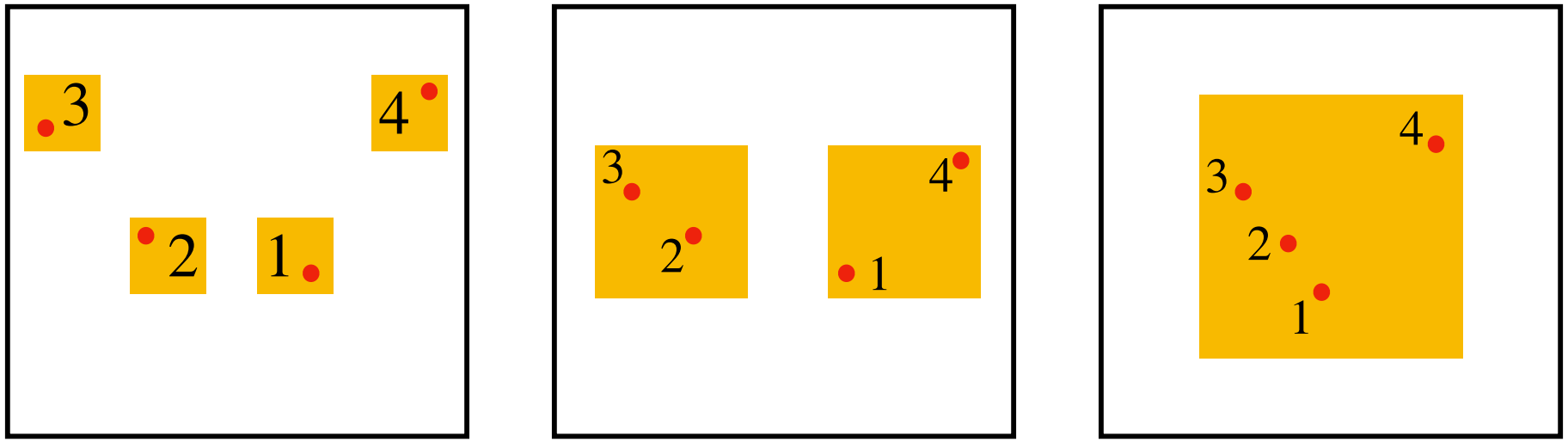
If gridding does not find a π-appearance

$\Omega(\epsilon n)$ disjoint π-appearances have their "**legs**" in dense boxes arranged according to any one of the constantly many types of "**configurations**"



If gridding does not find a π-appearance

$\Omega(\epsilon n)$ disjoint π-appearances have their "**legs**" in dense boxes arranged according to any one of the constantly many types of "**configurations**"

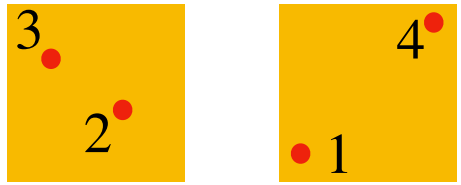


How to find π-appearances in these configurations?

- For each configuration $\mathscr{C}$:
  - Check for π-appearances whose legs are in boxes forming the configuration $\mathscr{C}$

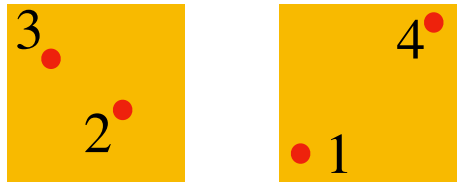
A 2-boxed configuration

- Suppose $\Omega(\epsilon n)$ disjoint π-appearances have their legs distributed according to the following 2-boxed configuration



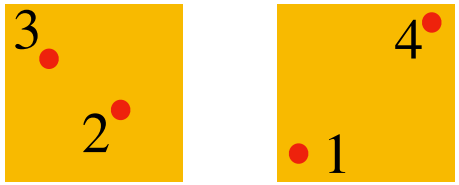
A 2-boxed configuration

- Suppose $\Omega(\epsilon n)$ disjoint π-appearances have their legs distributed according to the following 2-boxed configuration
- At most $\sqrt{n}$ dense boxes & each dense box has $\leq \sqrt{n}$ points



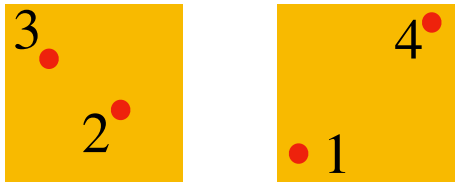
A 2-boxed configuration

- Suppose $\Omega(\epsilon n)$ disjoint π-appearances have their legs distributed according to the following 2-boxed configuration
- At most $\sqrt{n}$ dense boxes & each dense box has $\leq \sqrt{n}$ points
- A uniformly random dense box B contains the $(3,2)$ legs of $\Omega(\epsilon\sqrt{n})$ disjoint π -appearances



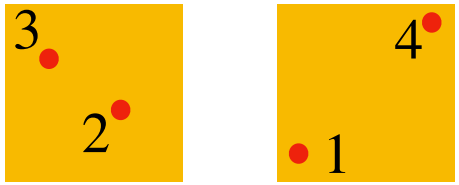
A 2-boxed configuration

- Suppose $\Omega(\epsilon n)$ disjoint π-appearances have their legs distributed according to the following 2-boxed configuration
- At most $\sqrt{n}$ dense boxes & each dense box has $\leq \sqrt{n}$ points
- A uniformly random dense box B contains the $(3,2)$ legs of $\Omega(\epsilon\sqrt{n})$ disjoint π -appearances
- There are $O(1)$ dense boxes sharing a row with B, by our Gridding



A 2-boxed configuration

- Suppose $\Omega(\epsilon n)$ disjoint π-appearances have their legs distributed according to the following 2-boxed configuration



- At most $\sqrt{n}$ dense boxes & each dense box has $\leq \sqrt{n}$ points
- A uniformly random dense box B contains the $(3,2)$ legs of $\Omega(\epsilon\sqrt{n})$ disjoint π -appearances
- There are $O(1)$ dense boxes sharing a row with B, by our Gridding
- Querying all points in the subarrays corresponding to B and the other dense boxes in its row is sufficient to detect such a π-appearance

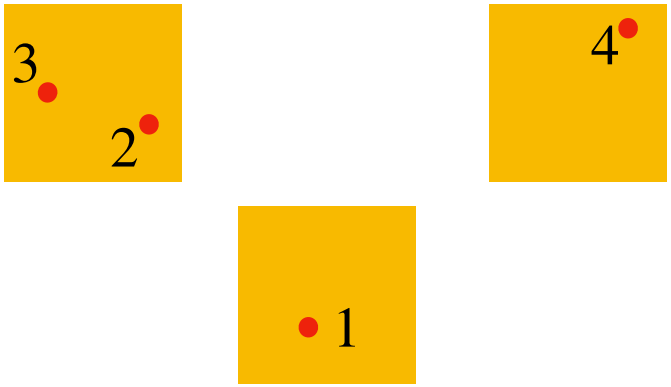
  $\tilde{O}(\sqrt{n})$ queries overall, since we want a high success probability

Other proof ideas

- Most other configurations are dealt with by similar averaging arguments
- However, more complicated arguments needed for some specific configurations

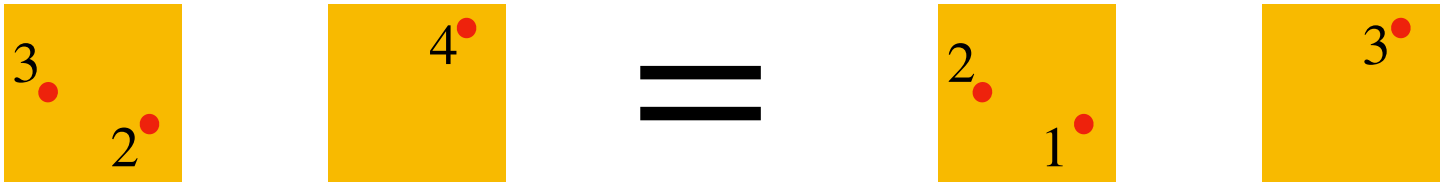
Other proof ideas

- Most other configurations are dealt with by similar averaging arguments
- However, more complicated arguments needed for some specific configurations



Other proof ideas

- Most other configurations are dealt with by similar averaging arguments
- However, more complicated arguments needed for some specific configurations
- Our algorithm works for patterns of all constant length k
  - For patterns of length $k > 4$, our recursive algorithm ends up solving a more general problem
  - Detect a ν-appearance with a specific leg distribution, where ν is some subpattern of π



Other proof ideas

- Most other configurations are dealt with by similar averaging arguments
- However, more complicated arguments needed for some specific configurations
- Our algorithm works for patterns of all constant length k
  - For patterns of length $k > 4$, our recursive algorithm ends up solving a more general problem
  - Detect a ν-appearance with a specific leg distribution, where ν is some subpattern of π
- The algorithm is called recursively in order to improve the query complexity to $\tilde{O}(n^{o(1)})$ from $O(\sqrt{n})$

Open problems

- What is the true complexity of testing π-freeness?

Open problems

- What is the true complexity of testing π-freeness?
  - Lower bounds for adaptive algorithms?

Open problems

- What is the true complexity of testing π-freeness?
  - Lower bounds for adaptive algorithms?
- What about patterns of superconstant length?

Open problems

- What is the true complexity of testing π-freeness?
  - Lower bounds for adaptive algorithms?
- What about patterns of superconstant length?
- Approximating the distance of arrays to π-freeness

Open problems

- What is the true complexity of testing π-freeness?
  - Lower bounds for adaptive algorithms?
- What about patterns of superconstant length?
- Approximating the distance of arrays to π-freeness

Thank you!