Multi-Pivot Quicksort
Comparison-Optimal Algorithms and Beyond

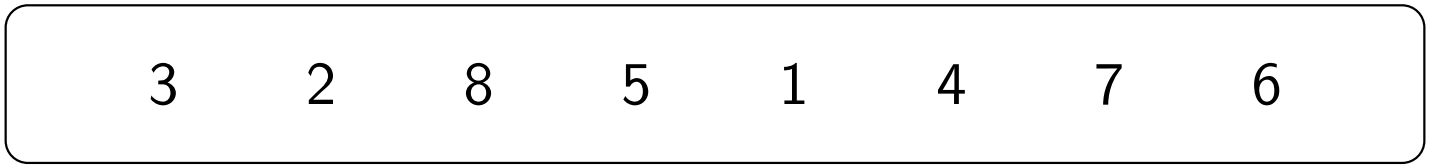
Martin Aumüller

IT University of Copenhagen

April 4, 2016

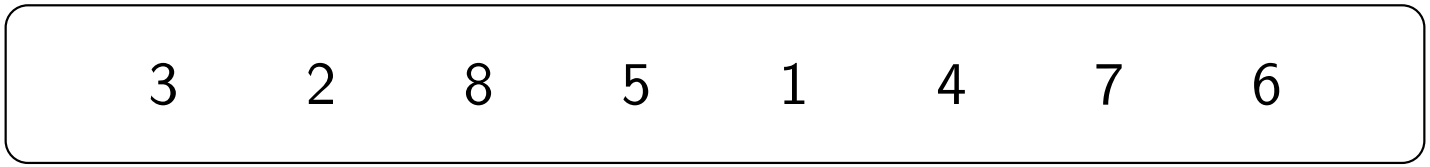
ARCO Workshop

Dual-Pivot Quicksort



Input: Permutation of $\{1, \ldots, n\}$.

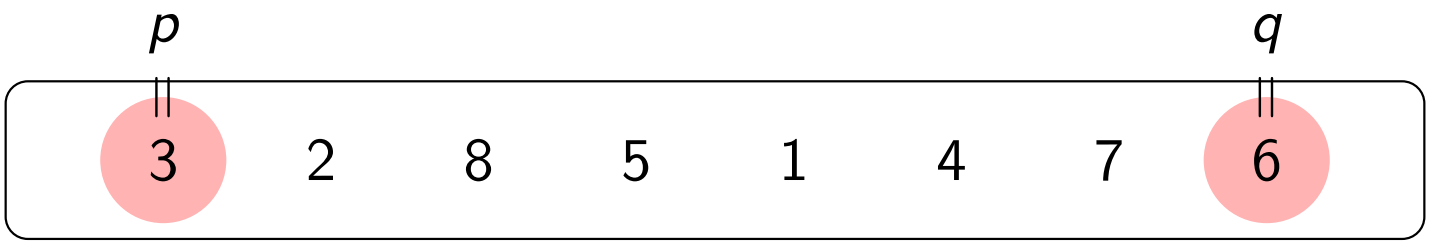
Dual-Pivot Quicksort



Input: Permutation of $\{1, \ldots, n\}$.

1. Choose **two** pivots p, q with $p < q$.

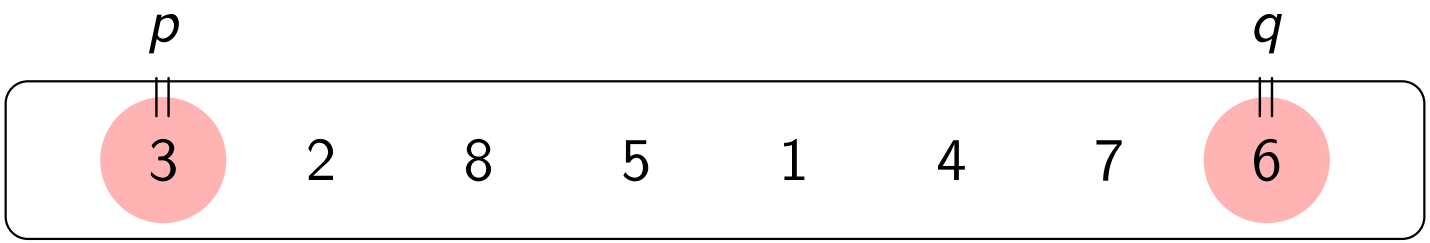
Dual-Pivot Quicksort



Input: Permutation of $\{1, \ldots, n\}$.

1. Choose **two** pivots p, q with $p < q$.

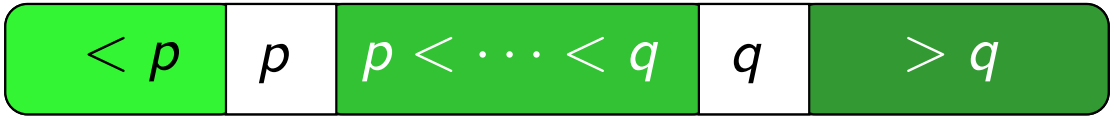
Dual-Pivot Quicksort



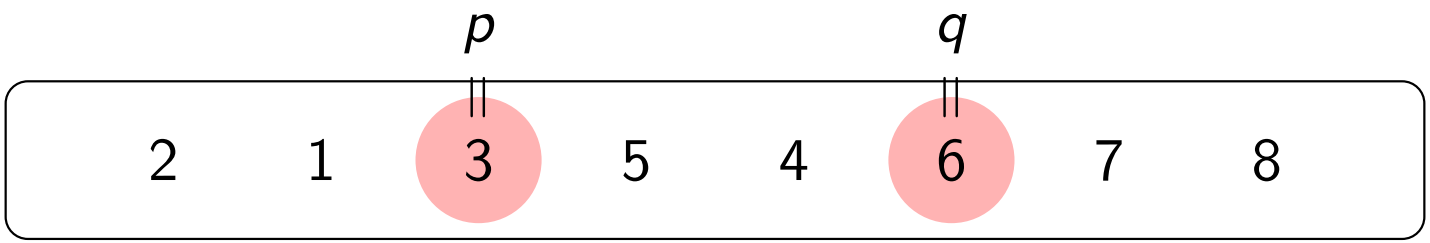
Input: Permutation of $\{1, \ldots, n\}$.

1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



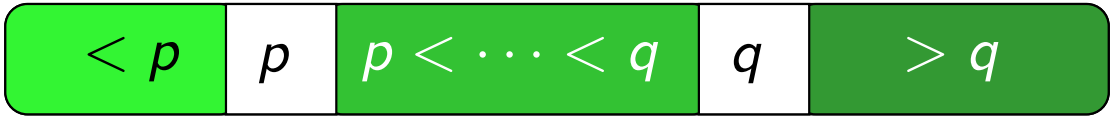
Dual-Pivot Quicksort



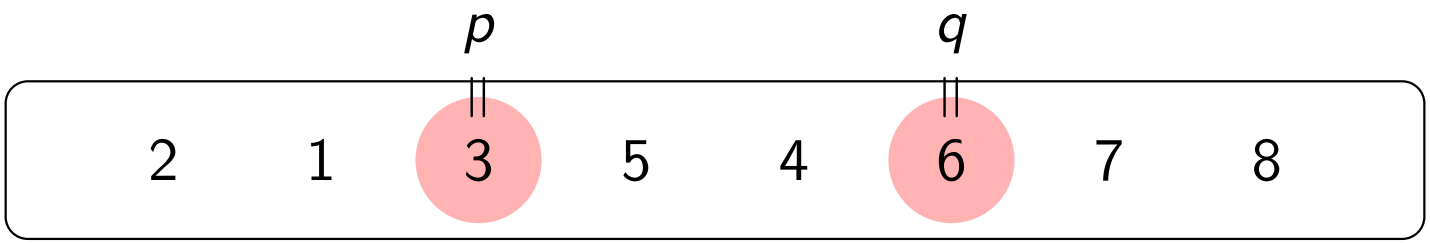
Input: Permutation of $\{1, \ldots, n\}$.

1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



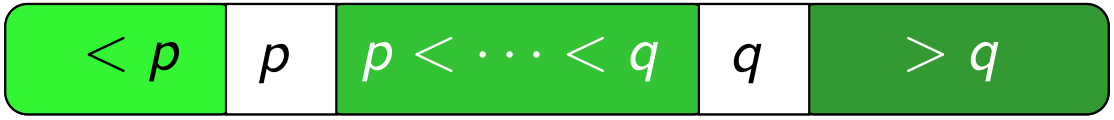
Dual-Pivot Quicksort



Input: Permutation of $\{1, \ldots, n\}$.

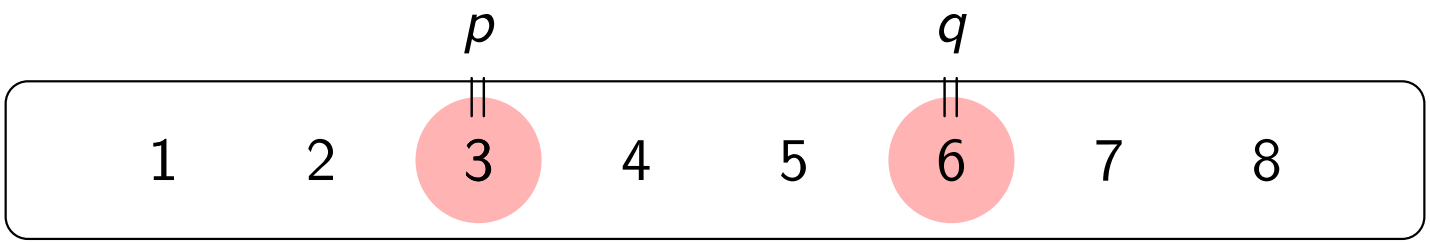
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



3. Use recursion to sort the **three** subarrays.

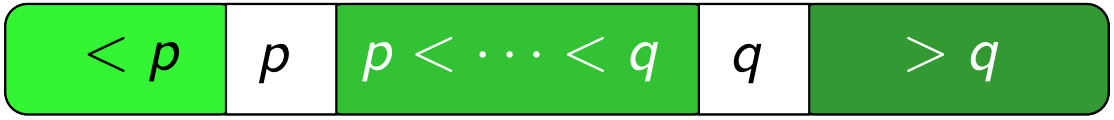
Dual-Pivot Quicksort



Input: Permutation of $\{1, \ldots, n\}$.

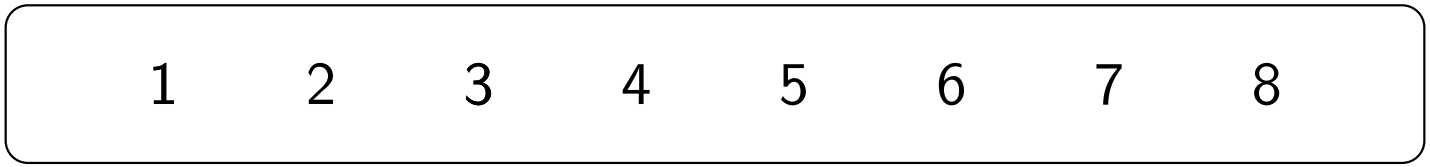
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



3. Use recursion to sort the **three** subarrays.

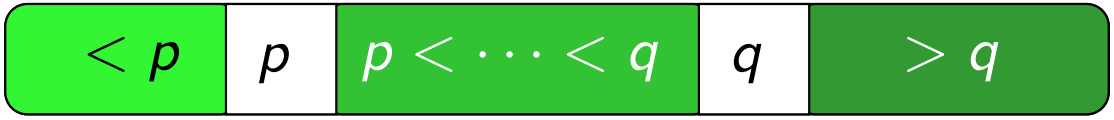
Dual-Pivot Quicksort



Input: Permutation of $\{1, \ldots, n\}$.

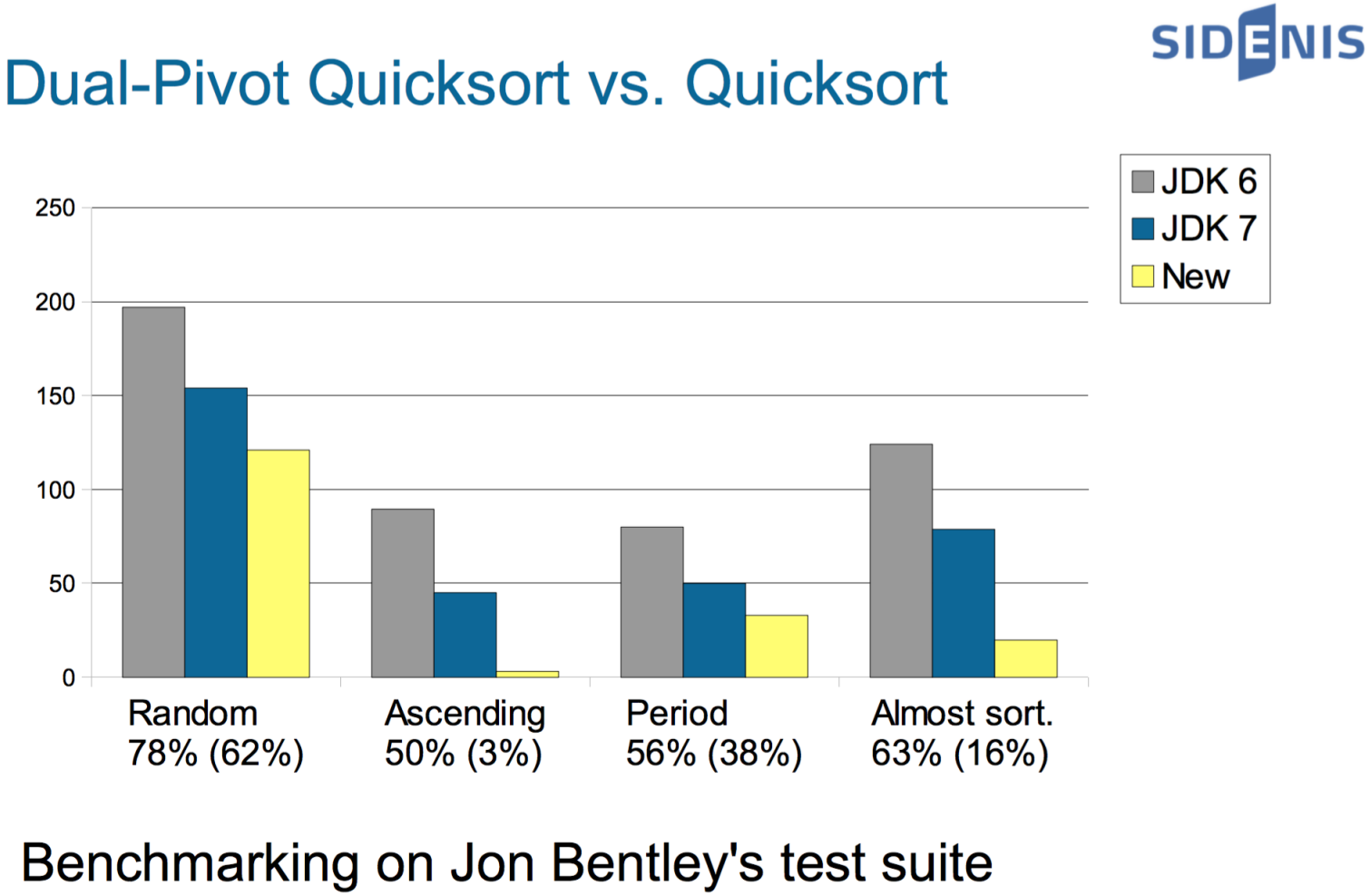
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

   Partition:



3. Use recursion to sort the **three** subarrays.

Why Consider It?



(Talk "Sorting in JDK", V. Yaroslavskiy, 2015.)

Part I

The Exact Average Comparison Count of Comparison-Optimal Dual-Pivot Quicksort

`http://arxiv.org/abs/1602.04031`

Joint work with Martin Dietzfelbinger (TU Ilmenau), Clemens Heuberger, Daniel Krenn (AAU Klagenfurt), and Helmut Prodinger (Stellenbosch University)

Goal

Analyze comparison-optimal dual-pivot quicksort algorithms **exactly**.

- P_n: cost for partitioning the input.
- C_n: cost for sorting the input.

Recurrence:

$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k)\mathbb{E}(C_k) \tag{1}$$

Goal

Analyze comparison-optimal dual-pivot quicksort algorithms **exactly**.

- P_n: cost for partitioning the input.
- C_n: cost for sorting the input.

Recurrence:

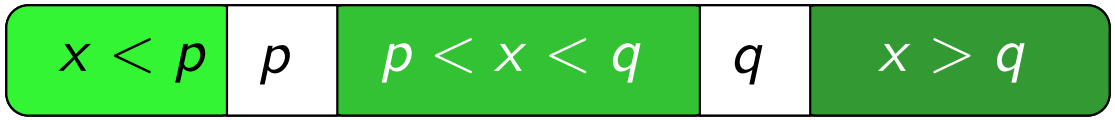
$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k)\mathbb{E}(C_k) \tag{1}$$

Need: exact analysis of partitioning cost $\mathbb{E}(P_n)$.

Then use known techniques (Generating functions, [Wild'13]) to solve (1).

How is the partitioning cost determined?

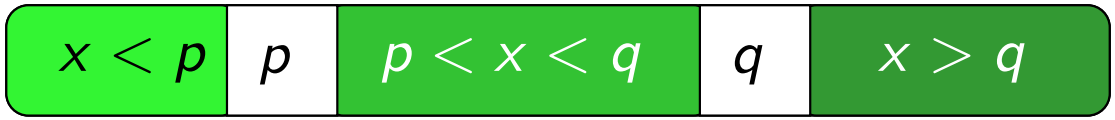
Must **classify** $n-2$ entries x into three parts:



small, medium, or large

How is the partitioning cost determined?

Must **classify** $n-2$ entries x into three parts:

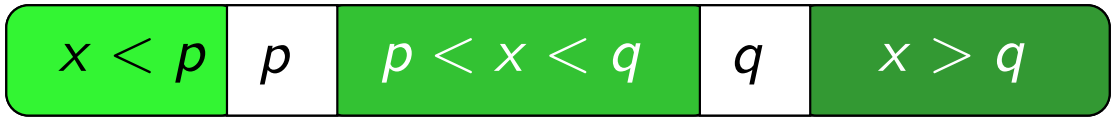


small, medium, or large

1 or 2 comparisons.
Unavoidable: 1 comparison for small/large x, 2 comparisons for medium x.

How is the partitioning cost determined?

Must **classify** $n-2$ entries x into three parts:



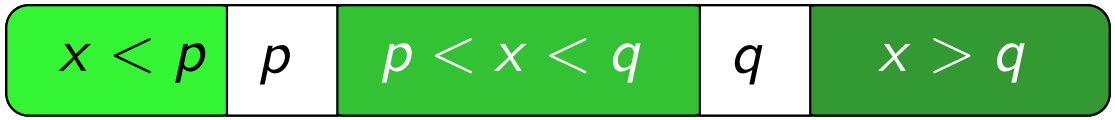
small, medium, or large

1 or 2 comparisons.
Unavoidable: 1 comparison for small/large x, 2 comparisons for medium x.

Extra: small x compared with q first and large x compared with p first.

How is the partitioning cost determined?

Must **classify** $n-2$ entries x into three parts:



small, medium, or large

1 or 2 comparisons.
Unavoidable: 1 comparison for small/large x, 2 comparisons for medium x.

Extra: small x compared with q first and large x compared with p first.

Partitioning strategy determines for next element x whether to compare x with p first or with q first.

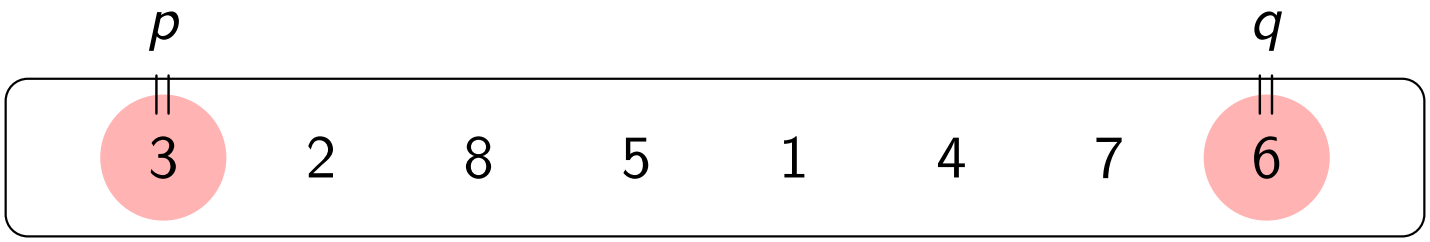
An implementation (Y./S./...) implicitly defines such a strategy.

Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ

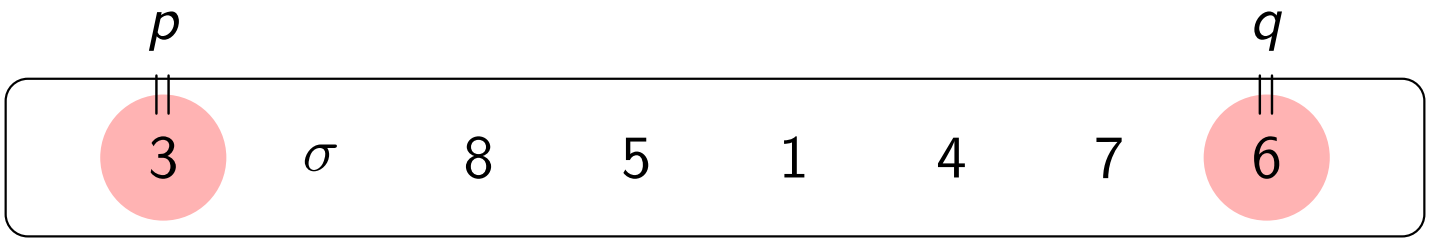
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



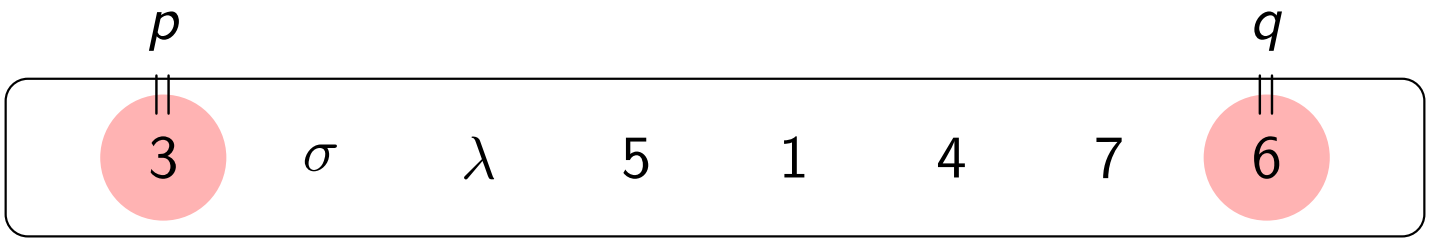
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



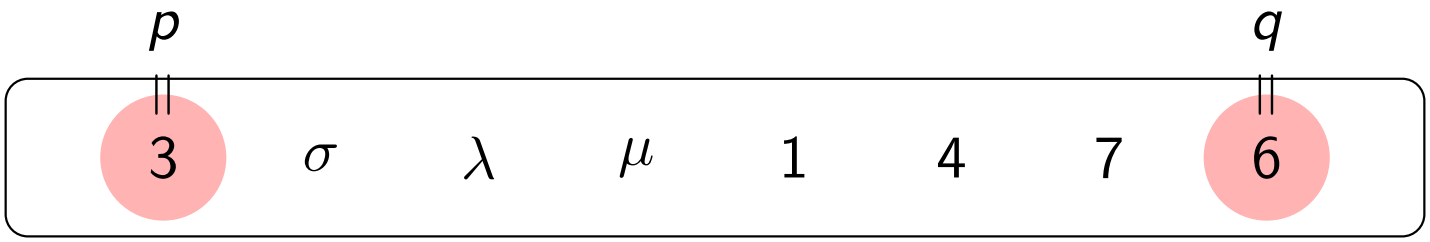
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



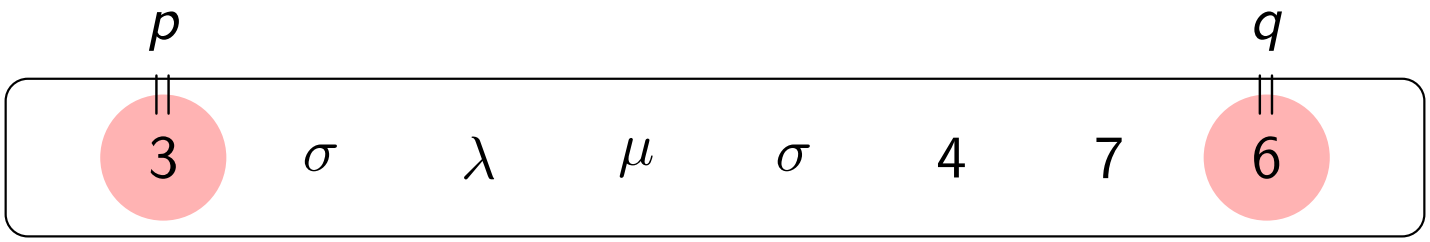
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \dots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



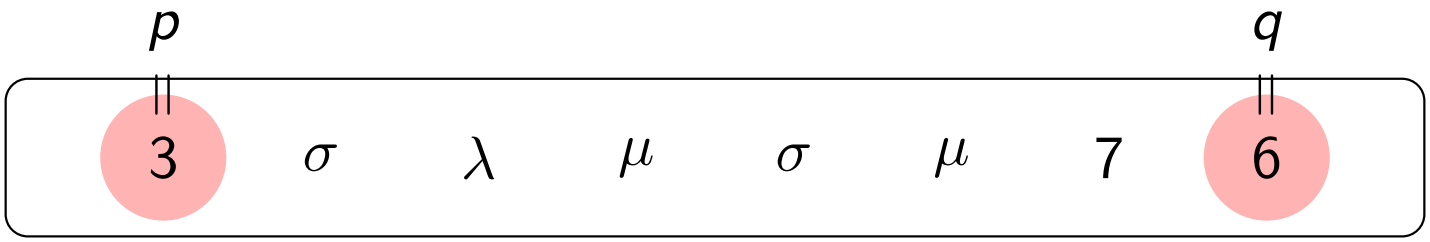
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



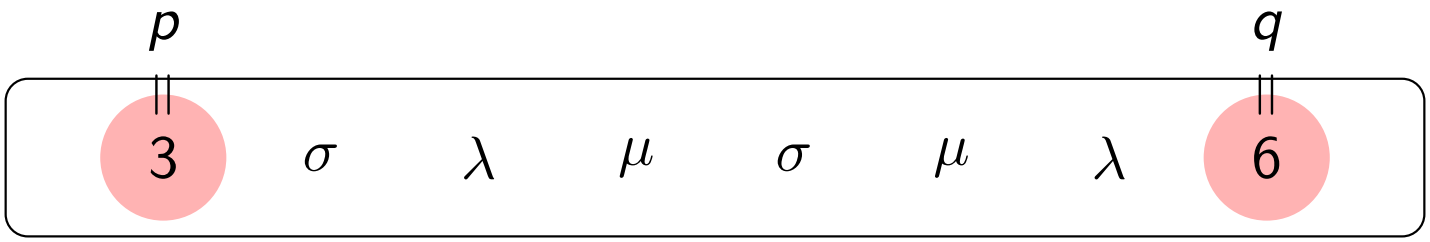
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



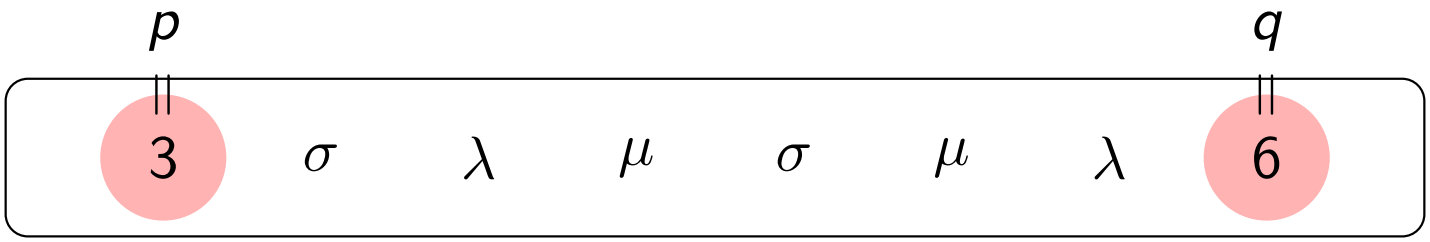
Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



Assumptions of the Analysis

- Input is **random permutation** of $\{1, \ldots, n\}$
- Choose two elements as **pivots**
- **Relabel** small elements as σ, medium elements as μ, and large elements as λ



Read input left to right, classify each element by one or two comparisons.

Average Cost

For the average partitioning/classification cost $\mathbb{E}(P_n)$ we get:

$$\mathbb{E}(P_n) = \frac{4}{3}(n-2) + 1 + \text{“average number of extra comparisons”}$$

For all classification algorithms:

- small [large] elements have to be compared to p [q]
- medium elements have to be compared to both

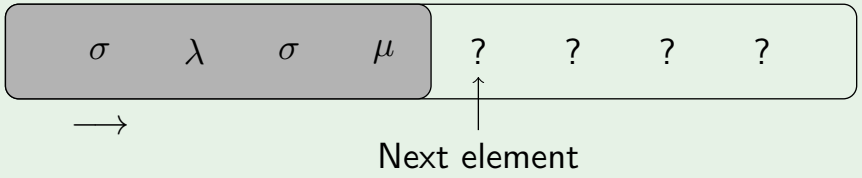
“Extra” comparisons:

- small x compared to q first
- large x compared to p first

Yaroslavskiy's Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Yaroslavskiy"

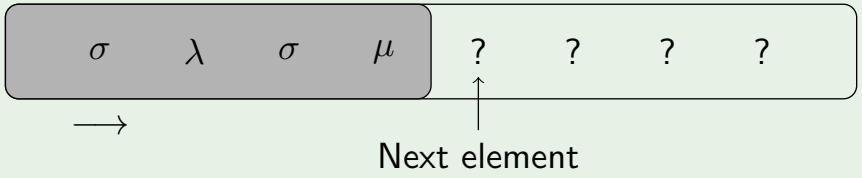


Seen elements in 

Yaroslavskiy's Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Yaroslavskiy"



Seen elements in 

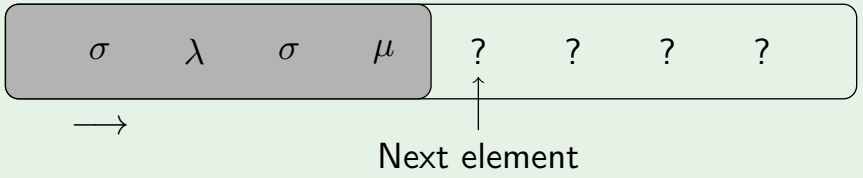
Classification Strategy:

- previous element large: compare with larger pivot first.
- otherwise: compare with smaller pivot first.

Yaroslavskiy's Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Yaroslavskiy"



Seen elements in 

Classification Strategy:

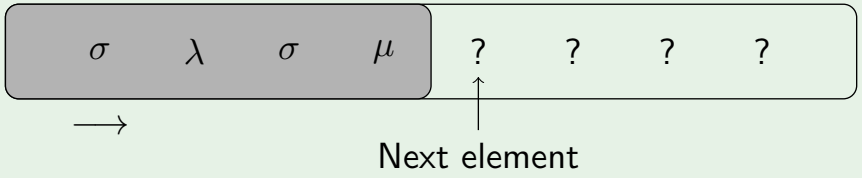
- previous element large: compare with larger pivot first.
- otherwise: compare with smaller pivot first.

[WN12]: Average sorting cost is $1.9n \ln n + O(n)$.

Yaroslavskiy's Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Yaroslavskiy"



Seen elements in 

Classification Strategy:

- previous element large: compare with larger pivot first.
- otherwise: compare with smaller pivot first.

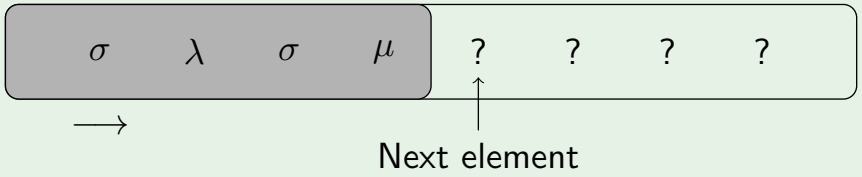
[WN12]: Average sorting cost is $1.9n \ln n + O(n)$.

[WNN15]: Analysis of the distribution of C_n (and other cost measures).

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Count"

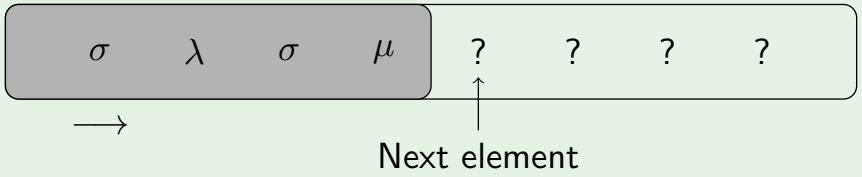


Seen s_{i-1} small and ℓ_{i-1} large elements in 

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Count"



Seen s_{i-1} small and ℓ_{i-1} large elements in 

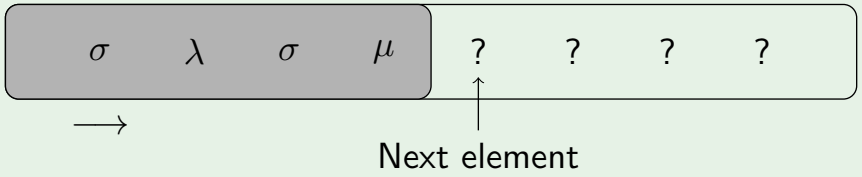
Classification Strategy:

- $\ell_{i-1} > s_{i-1}$: compare with larger pivot first.
- $\ell_{i-1} \leq s_{i-1}$: compare with smaller pivot first.

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Count"



Seen s_{i-1} small and ℓ_{i-1} large elements in 

Classification Strategy:

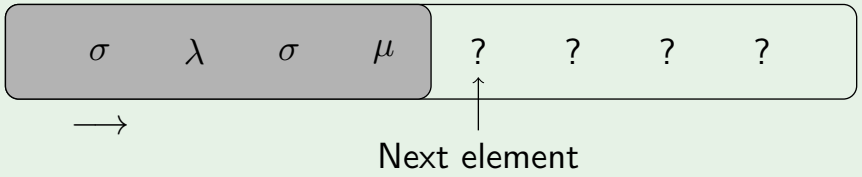
- $\ell_{i-1} > s_{i-1}$: compare with larger pivot first.
- $\ell_{i-1} \leq s_{i-1}$: compare with smaller pivot first.

[AD15]: Average sorting cost is $1.8n \ln n + O(n)$.

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i.

Strategy "Count"



Seen s_{i-1} small and ℓ_{i-1} large elements in 

Classification Strategy:

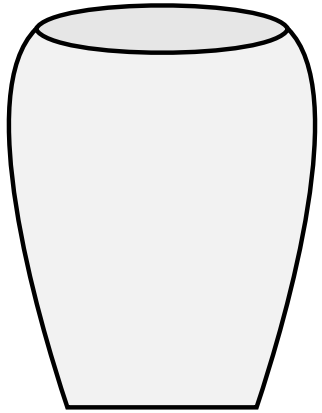
- $\ell_{i-1} > s_{i-1}$: compare with larger pivot first.
- $\ell_{i-1} \leq s_{i-1}$: compare with smaller pivot first.

[AD15]: Average sorting cost is $1.8n \ln n + O(n)$.

Here: "Count" is optimal + exact average comparison count.

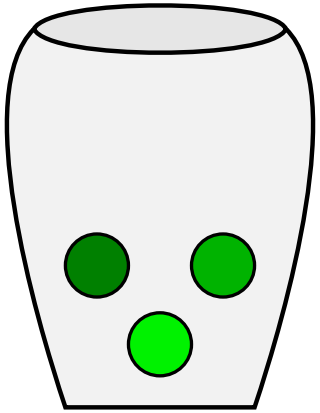
Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

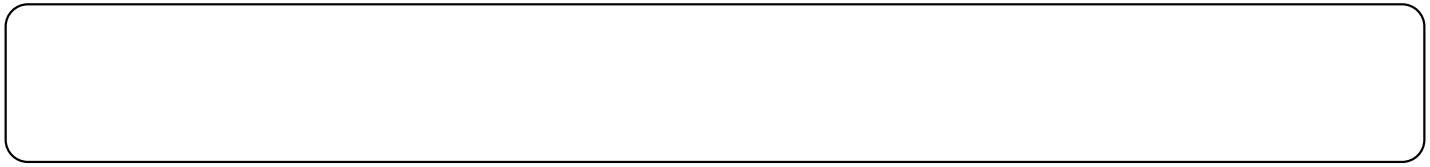


Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

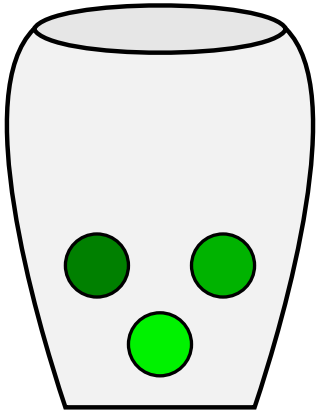


1. Put one light green ball, one green ball, one dark green ball in urn.

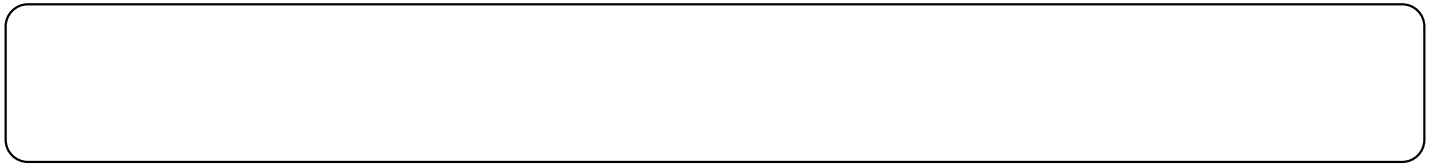


Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

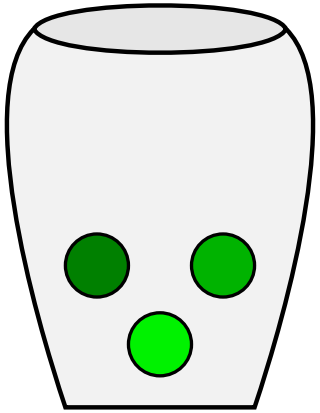


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:

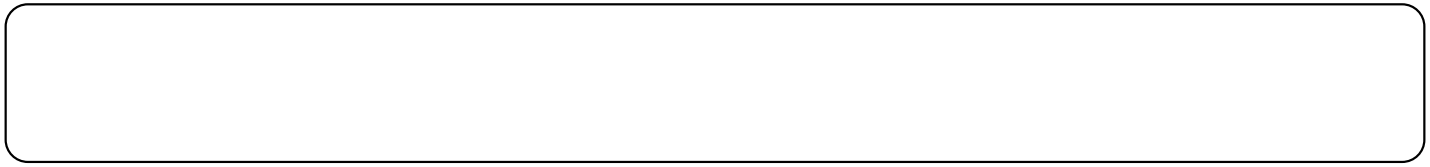


Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

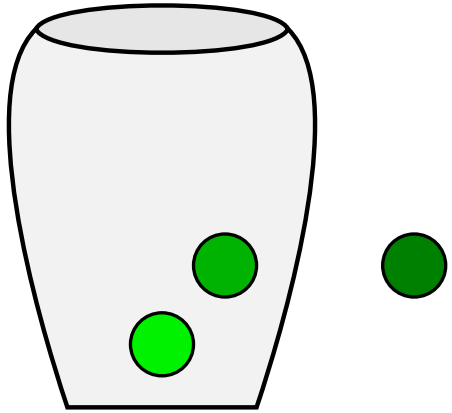


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random.

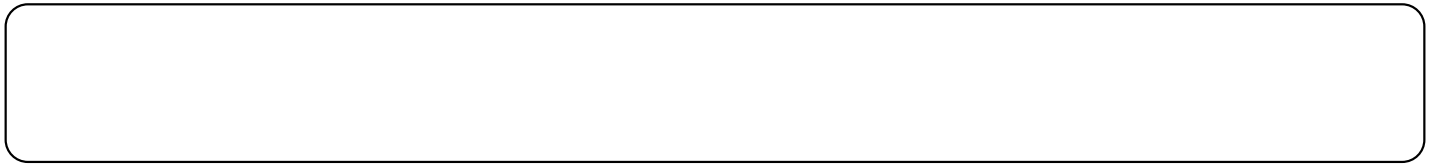


Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

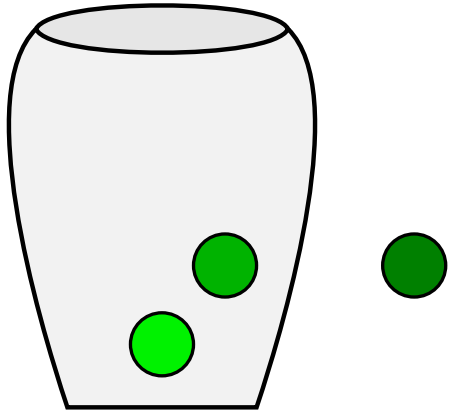


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random.



Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

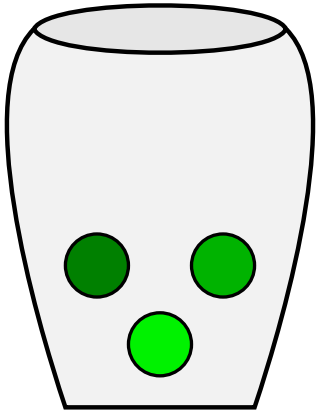


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back,

λ

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

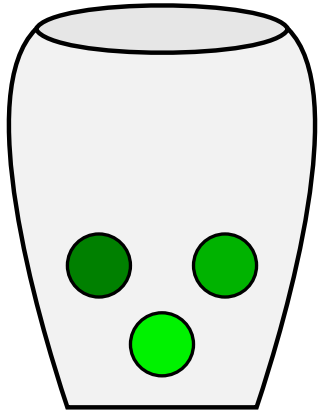


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back,

λ

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

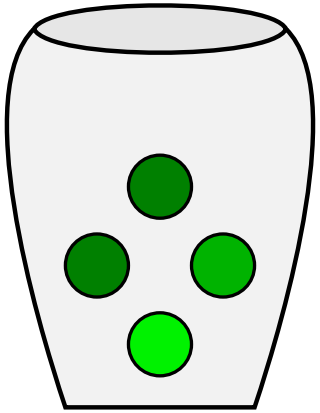


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

λ

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

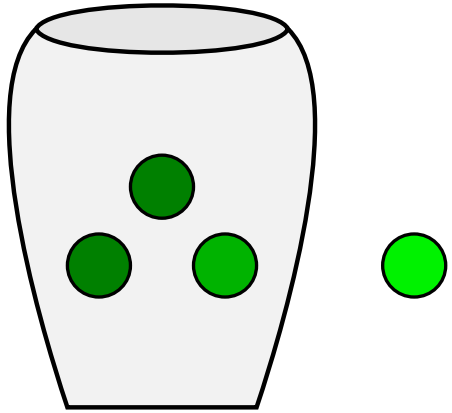


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

λ

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

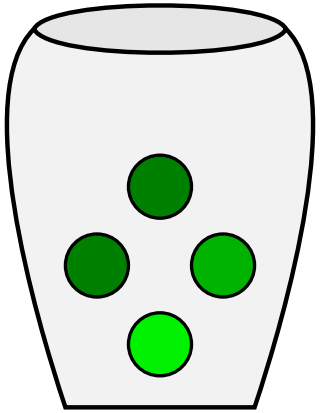


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$\lambda \quad \sigma$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

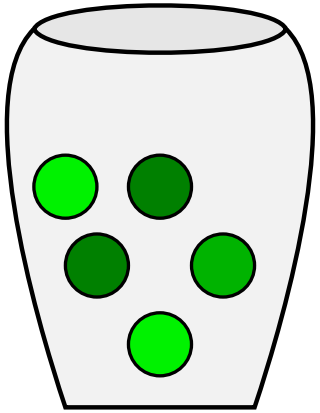


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$\lambda \quad \sigma$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

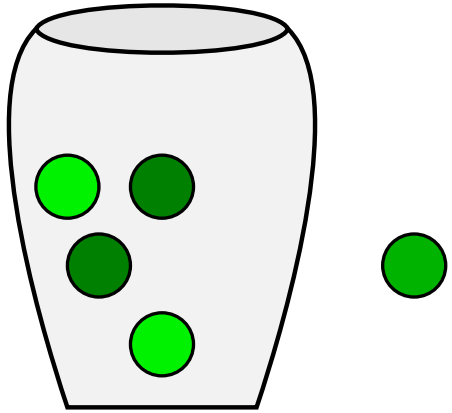


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$\lambda \quad \sigma$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

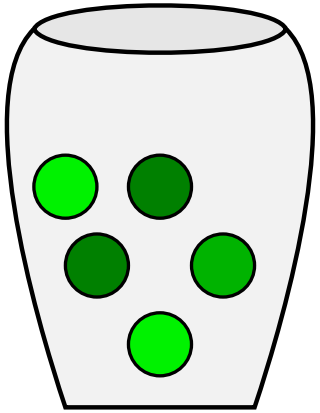


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$\lambda \qquad \sigma \qquad \mu$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

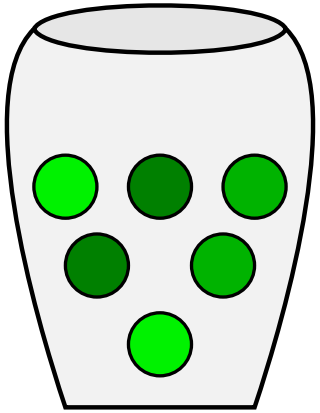


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$\lambda \quad \sigma \quad \mu$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

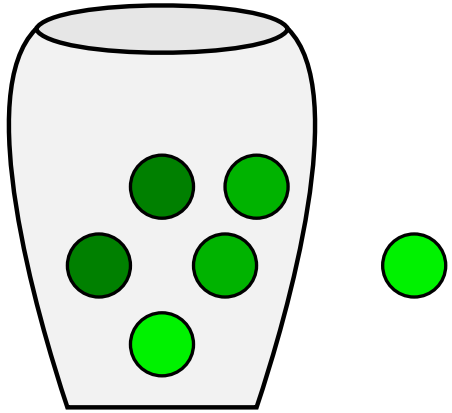


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$\lambda \quad \sigma \quad \mu$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

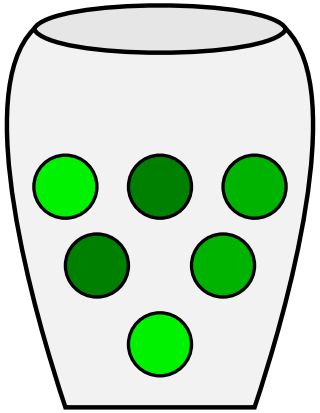


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$$\lambda \qquad \sigma \qquad \mu \qquad \sigma$$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

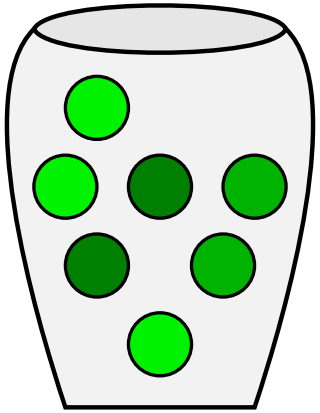


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$$\lambda \qquad \sigma \qquad \mu \qquad \sigma$$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.

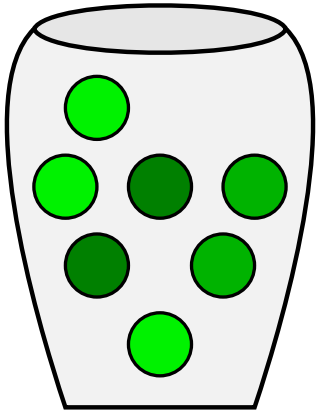


1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

$$\lambda \qquad \sigma \qquad \mu \qquad \sigma$$

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.



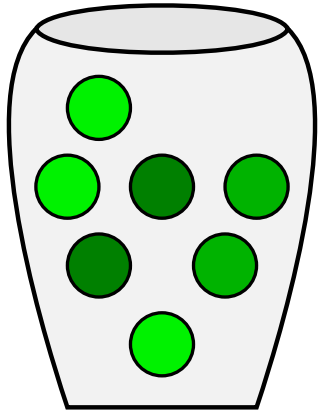
1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

λ	σ	μ	σ

Can show: This leads to the same distribution on $\{\sigma, \mu, \lambda\}$ sequences as the one generated by random permutation/pivots/relabel.

Proof idea: "Count" is optimal (among all algorithms)

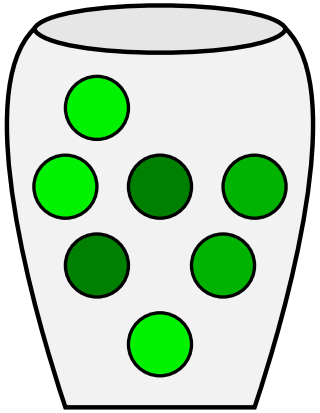
Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.



1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.



1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

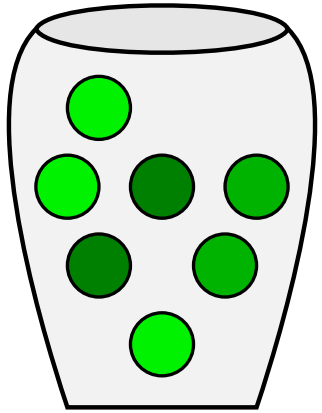
In Round i:

$$\Pr(\text{element is small}) = \frac{s_{i-1} + 1}{i + 2}.$$

(analogous formulæ for medium/large elements.)

Proof idea: "Count" is optimal (among all algorithms)

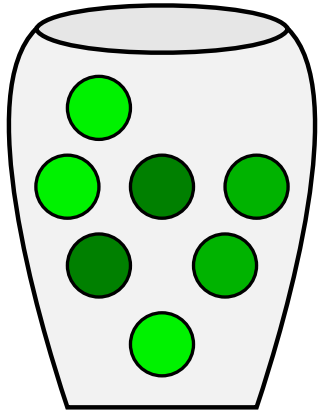
Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.



1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

Proof idea: "Count" is optimal (among all algorithms)

Pólya urn with three colors to model input sequence of $\{\sigma, \mu, \lambda\}$.



1. Put one light green ball, one green ball, one dark green ball in urn.
2. Round $i = 1, \ldots, n-2$:
   Choose ball from urn at random. Take down its color c, put it back, and put another ball of the same color in the urn.

Next step: Show that every partitioning strategy $\mathcal{S}$ with decision in round i based on the elements seen in rounds up to $i-1$ makes at least as many comparisons in round i as "Count" (on average).

"Count" is optimal (among all algorithms)

If $s_{i-1} \geq \ell_{i-1}$, "Count" compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{"Count" gets extra comparison in step } i) = \frac{\ell_{i-1}+1}{i+2}.$$

"Count" is optimal (among all algorithms)

If $s_{i-1} \geq \ell_{i-1}$, "Count" compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{"Count" gets extra comparison in step } i) = \frac{\ell_{i-1}+1}{i+2}.$$

If $\mathcal{S}$ uses small pivot first, no difference.

"Count" is optimal (among all algorithms)

If $s_{i-1} \geq \ell_{i-1}$, "Count" compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{"Count" gets extra comparison in step } i) = \frac{\ell_{i-1}+1}{i+2}.$$

If $\mathcal{S}$ uses small pivot first, no difference.

If $\mathcal{S}$ takes large pivot first:

$$\Pr(\mathcal{S} \text{ gets extra comparison in step } i) = \frac{s_{i-1}+1}{i+2},$$

at least as big as the probability for "Count".

"Count" is optimal (among all algorithms)

If $s_{i-1} \geq \ell_{i-1}$, "Count" compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{"Count" gets extra comparison in step } i) = \frac{\ell_{i-1}+1}{i+2}.$$

If $\mathcal{S}$ uses small pivot first, no difference.

If $\mathcal{S}$ takes large pivot first:

$$\Pr(\mathcal{S} \text{ gets extra comparison in step } i) = \frac{s_{i-1}+1}{i+2},$$

at least as big as the probability for "Count".

Conclusion: "Count" is the comparison-optimal strategy.

Next: Analysis of average comparison count.

Define “random walk” :

$$X_i = s_{i-1} - \ell_{i-1}, \quad \text{for } 1 \leq i \leq n-1.$$

Define "random walk" :

$$X_i = s_{i-1} - \ell_{i-1}, \quad \text{for } 1 \leq i \leq n-1.$$

Classification Strategy:

- $X_i \geq 0$: compare with smaller pivot first.
- $X_i < 0$: compare with larger pivot first.

Define "random walk" :

$$X_i = s_{i-1} - \ell_{i-1}, \quad \text{for } 1 \leq i \leq n-1.$$

Classification Strategy:

- $X_i \geq 0$: compare with smaller pivot first.
- $X_i < 0$: compare with larger pivot first.

Study simplified situation, ignore medium elements.

Define "random walk" :

$$X_i = s_{i-1} - \ell_{i-1}, \quad \text{for } 1 \le i \le n-1.$$

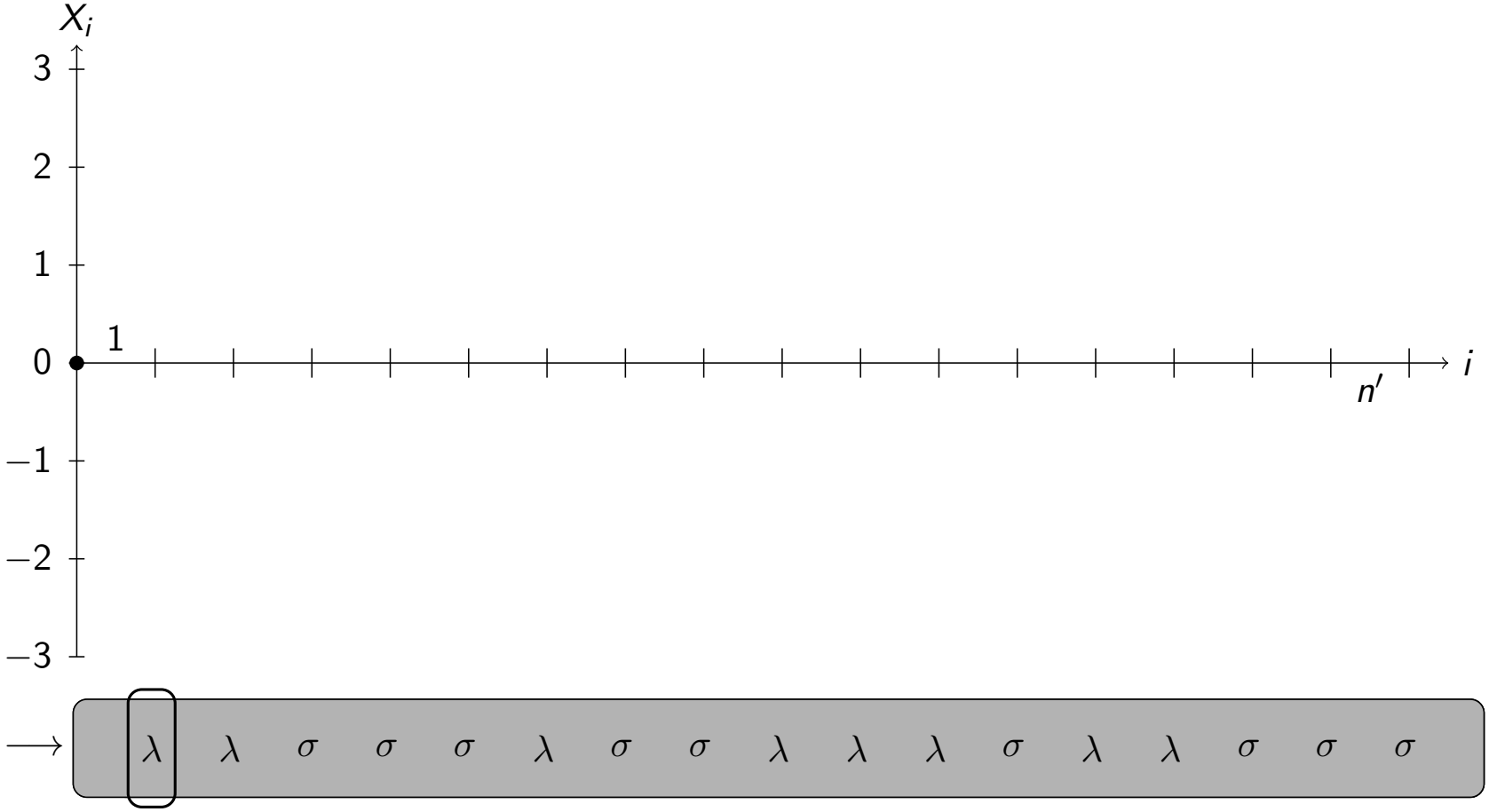
Classification Strategy:

- $X_i \ge 0$: compare with smaller pivot first.
- $X_i < 0$: compare with larger pivot first.

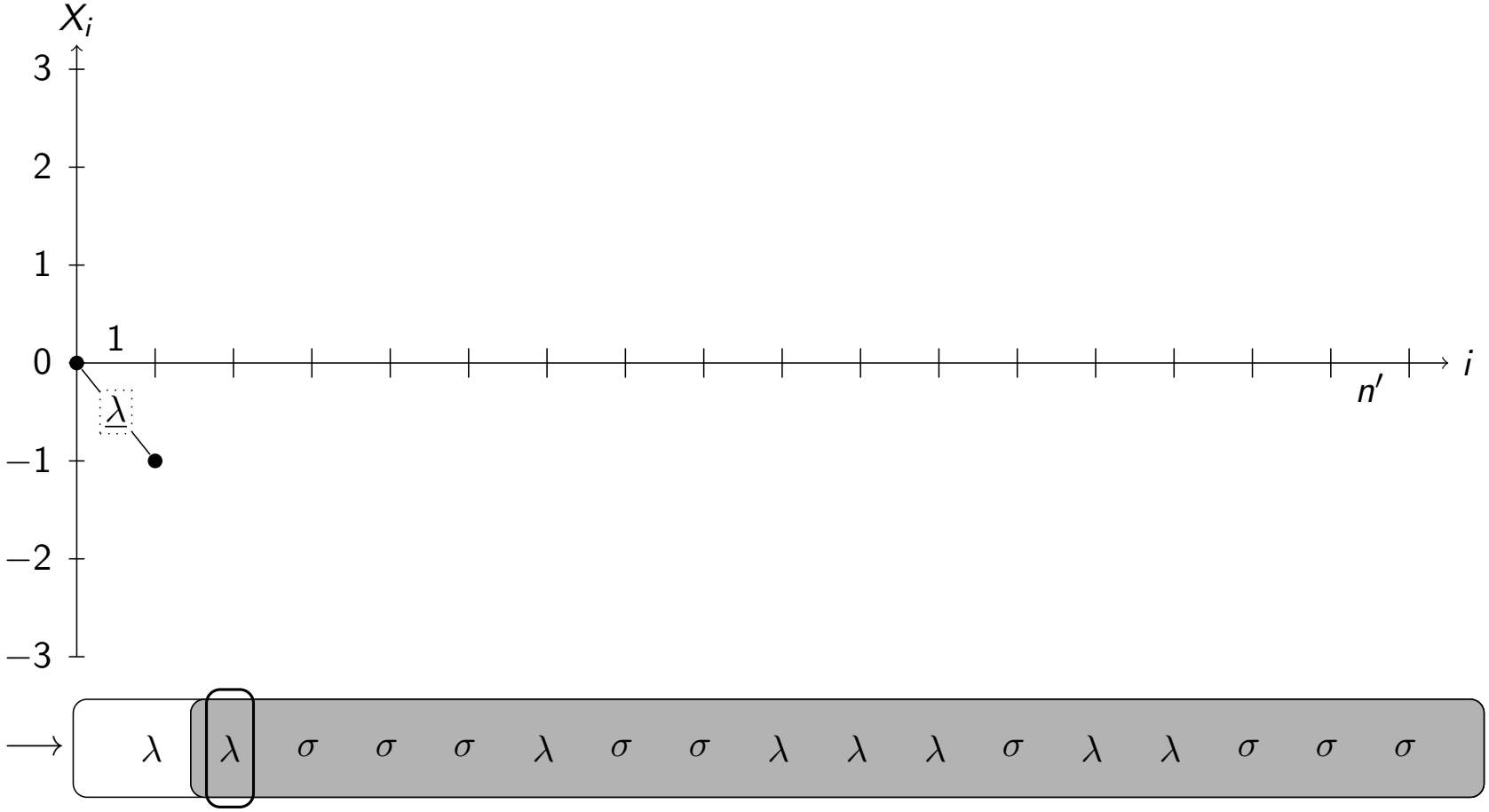
Study simplified situation, ignore medium elements.

We classify $n' = s + \ell$ elements.

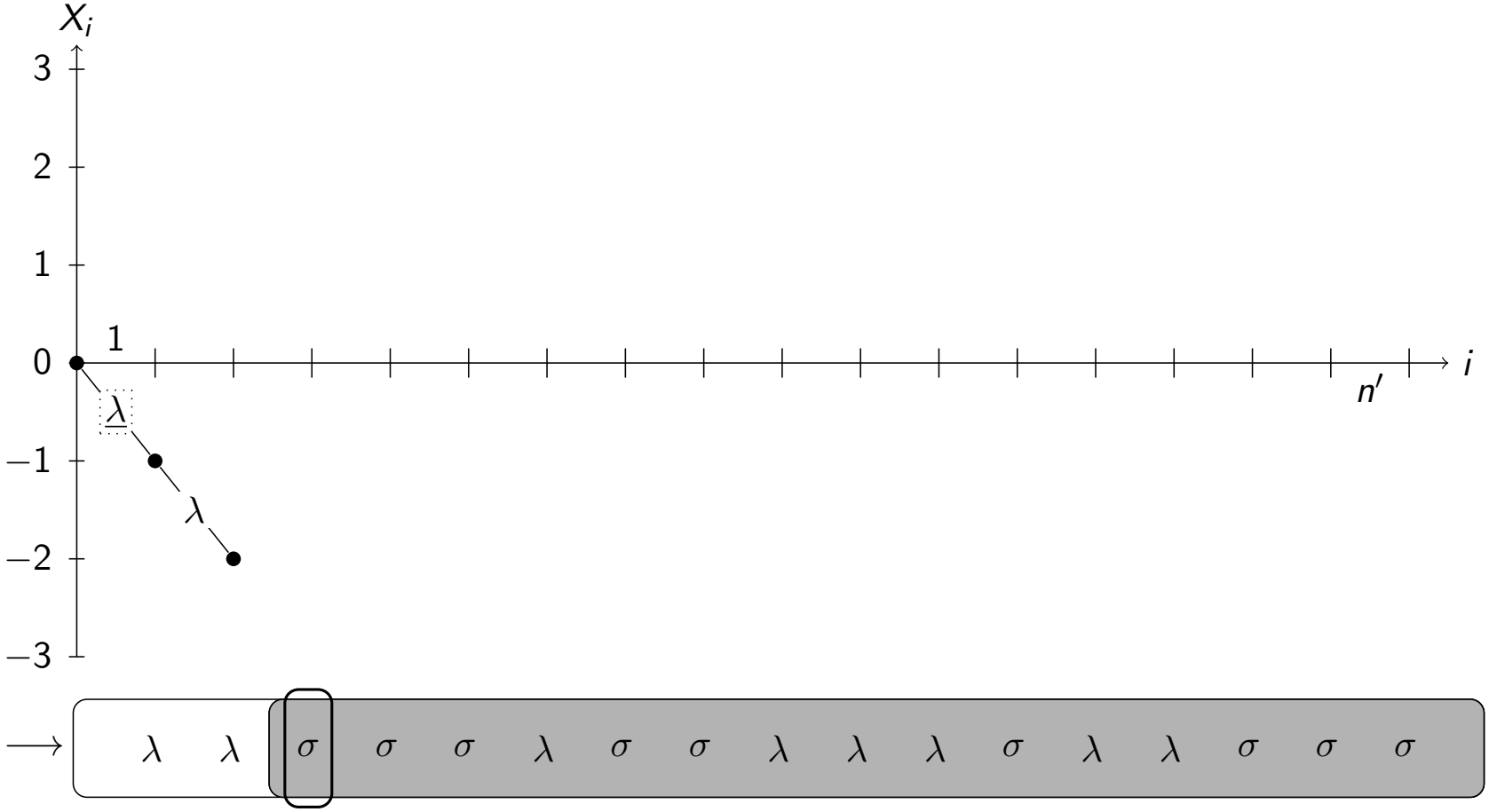
Random Walks & Analysis of Count



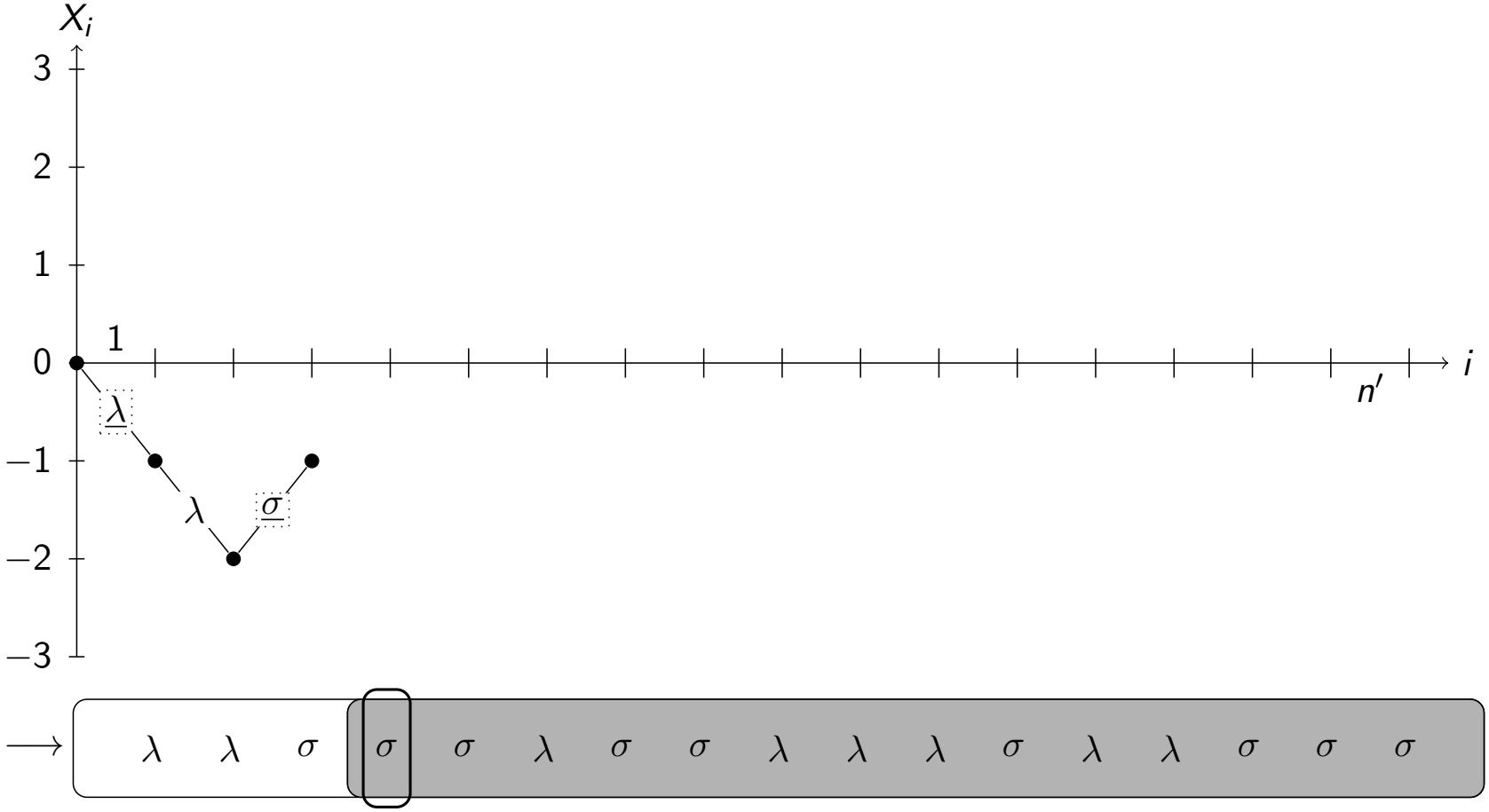
Random Walks & Analysis of Count



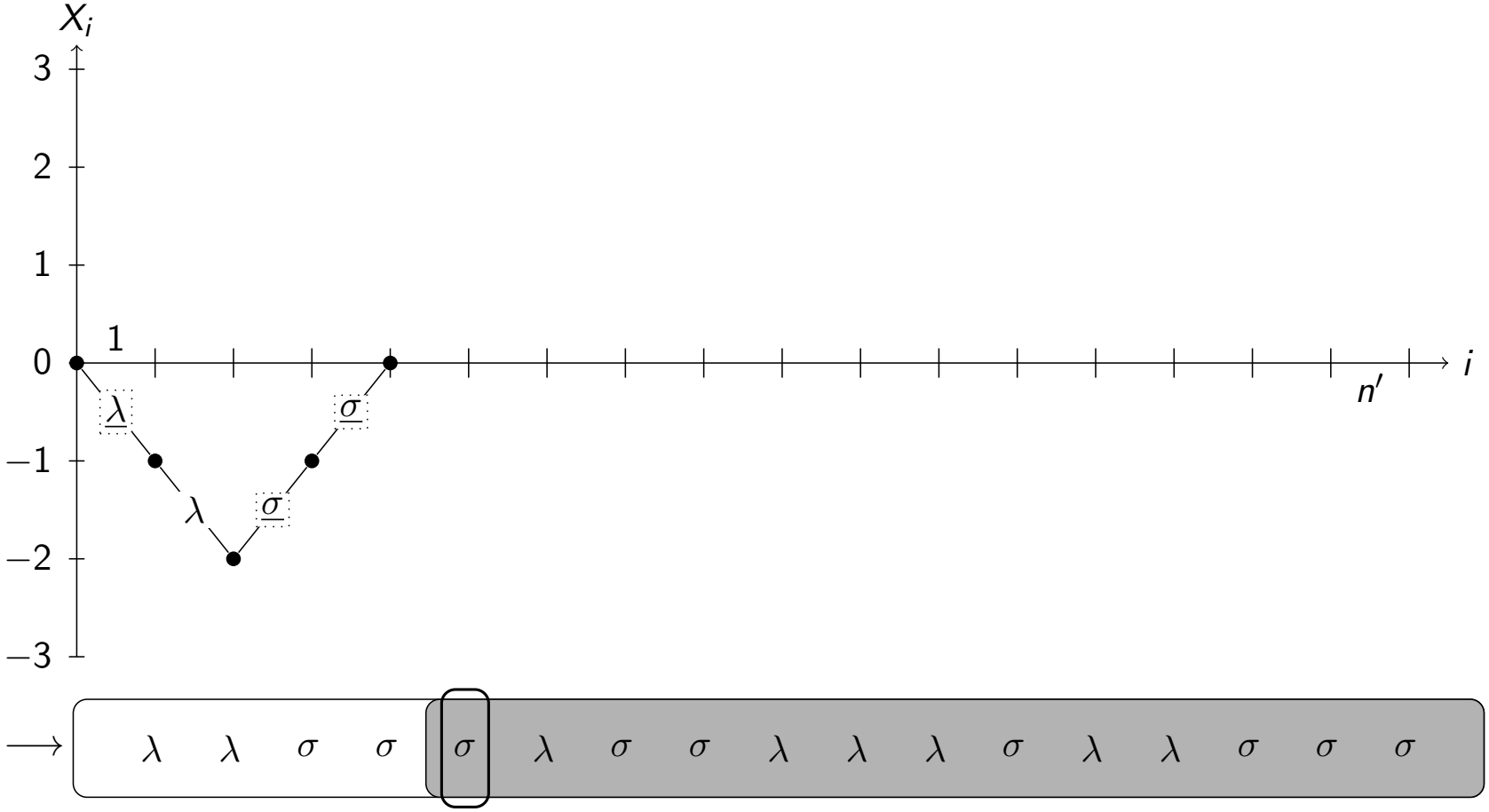
Random Walks & Analysis of Count



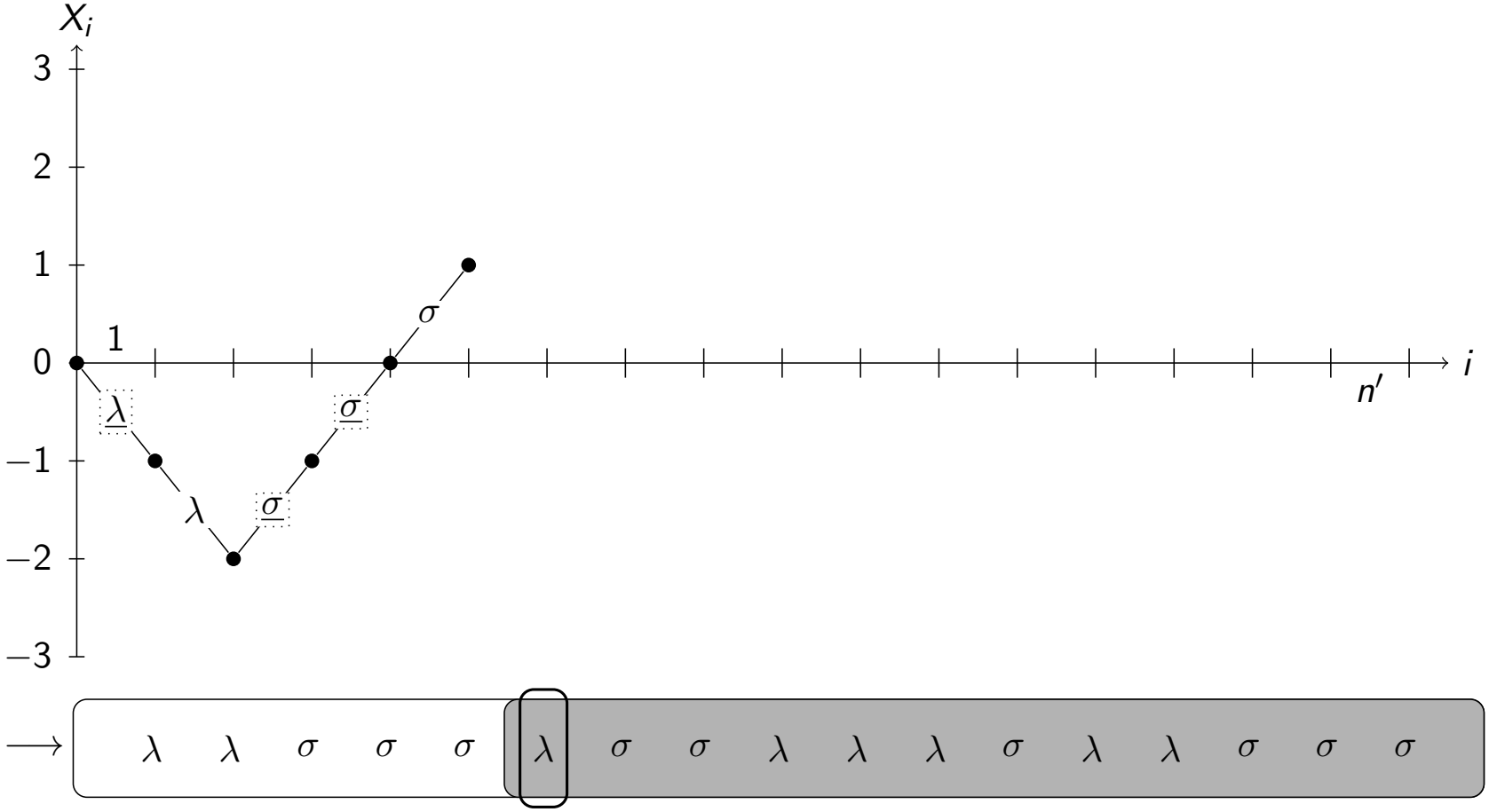
Random Walks & Analysis of Count



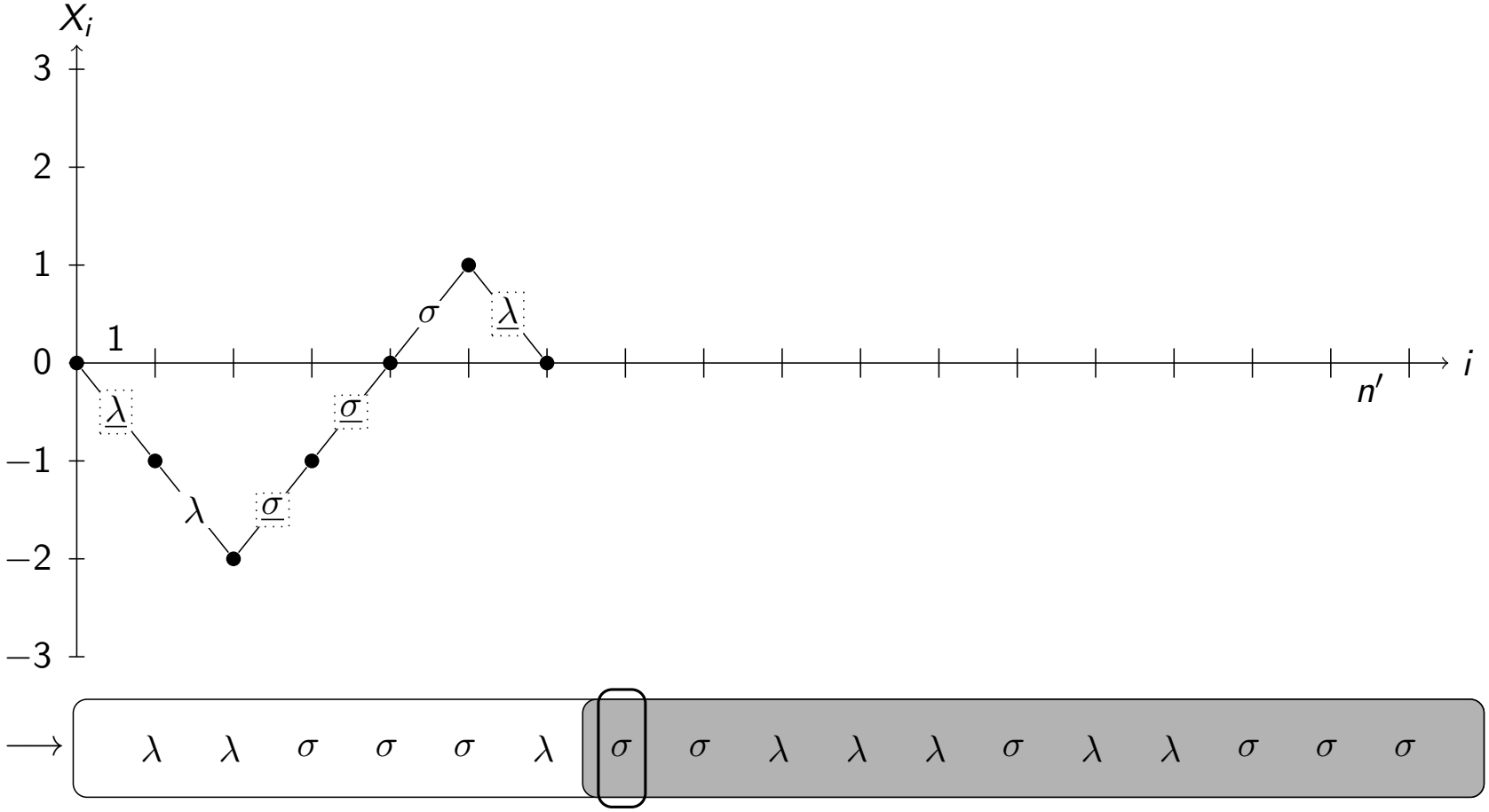
Random Walks & Analysis of Count



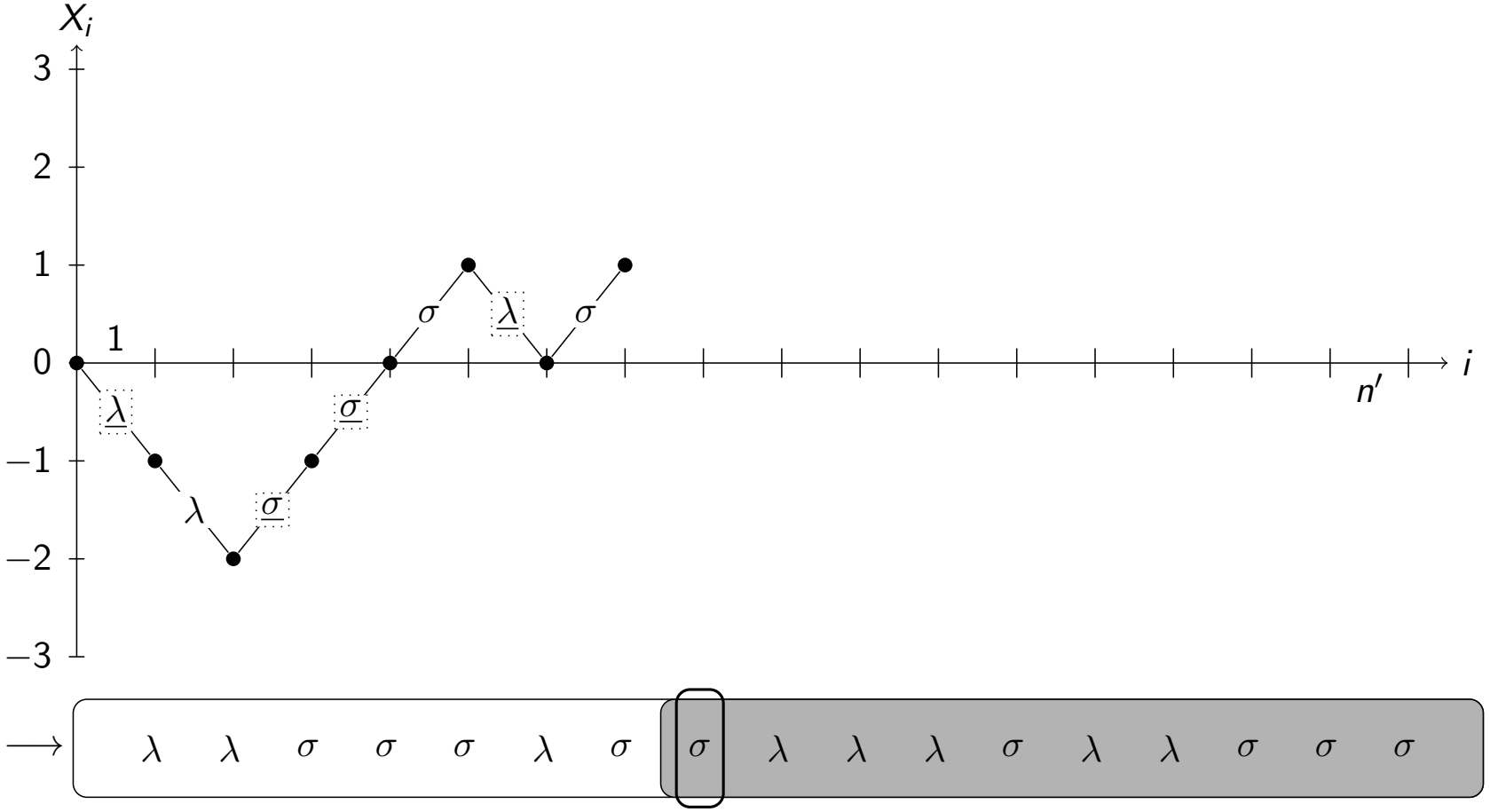
Random Walks & Analysis of Count



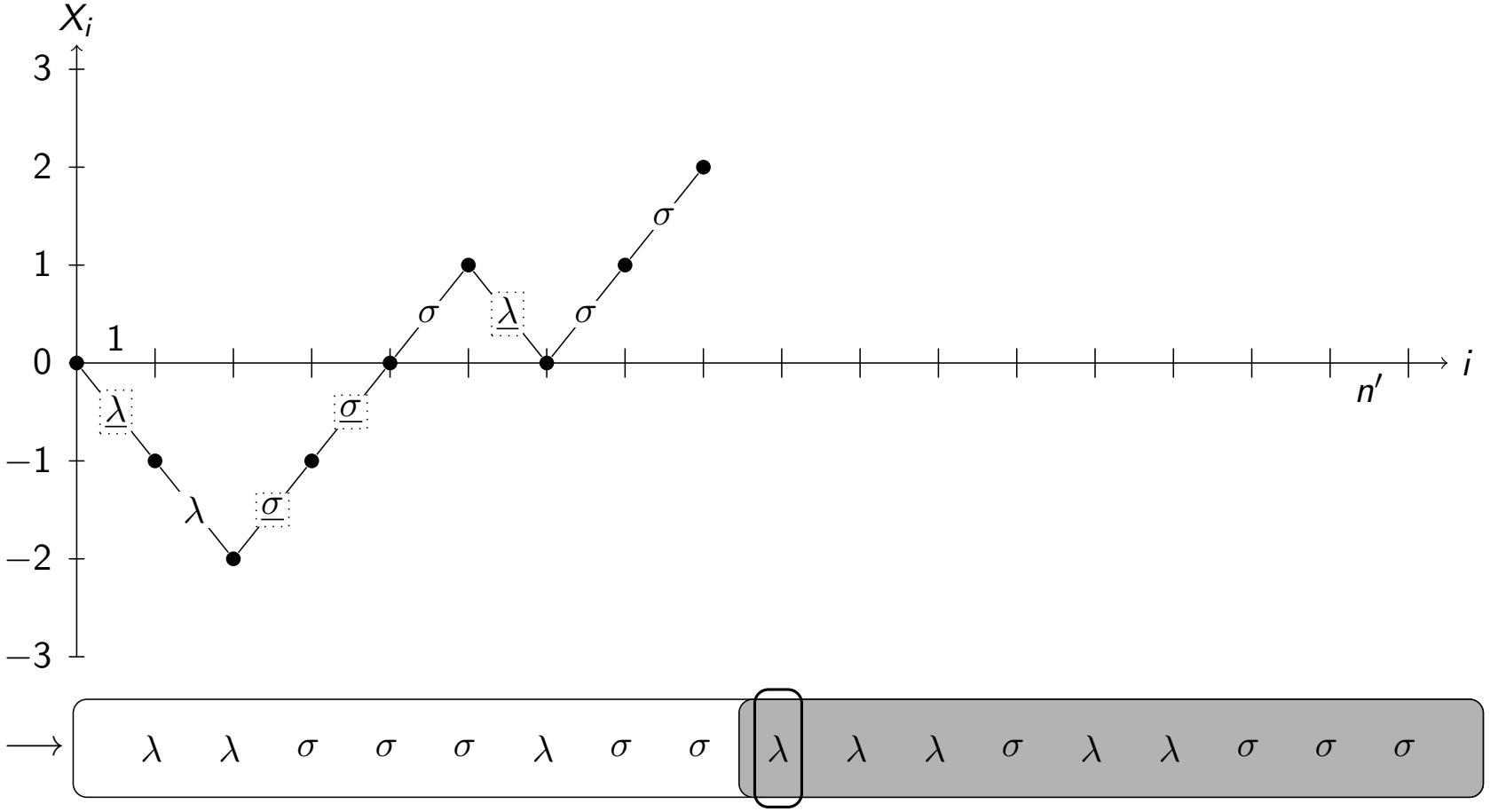
Random Walks & Analysis of Count



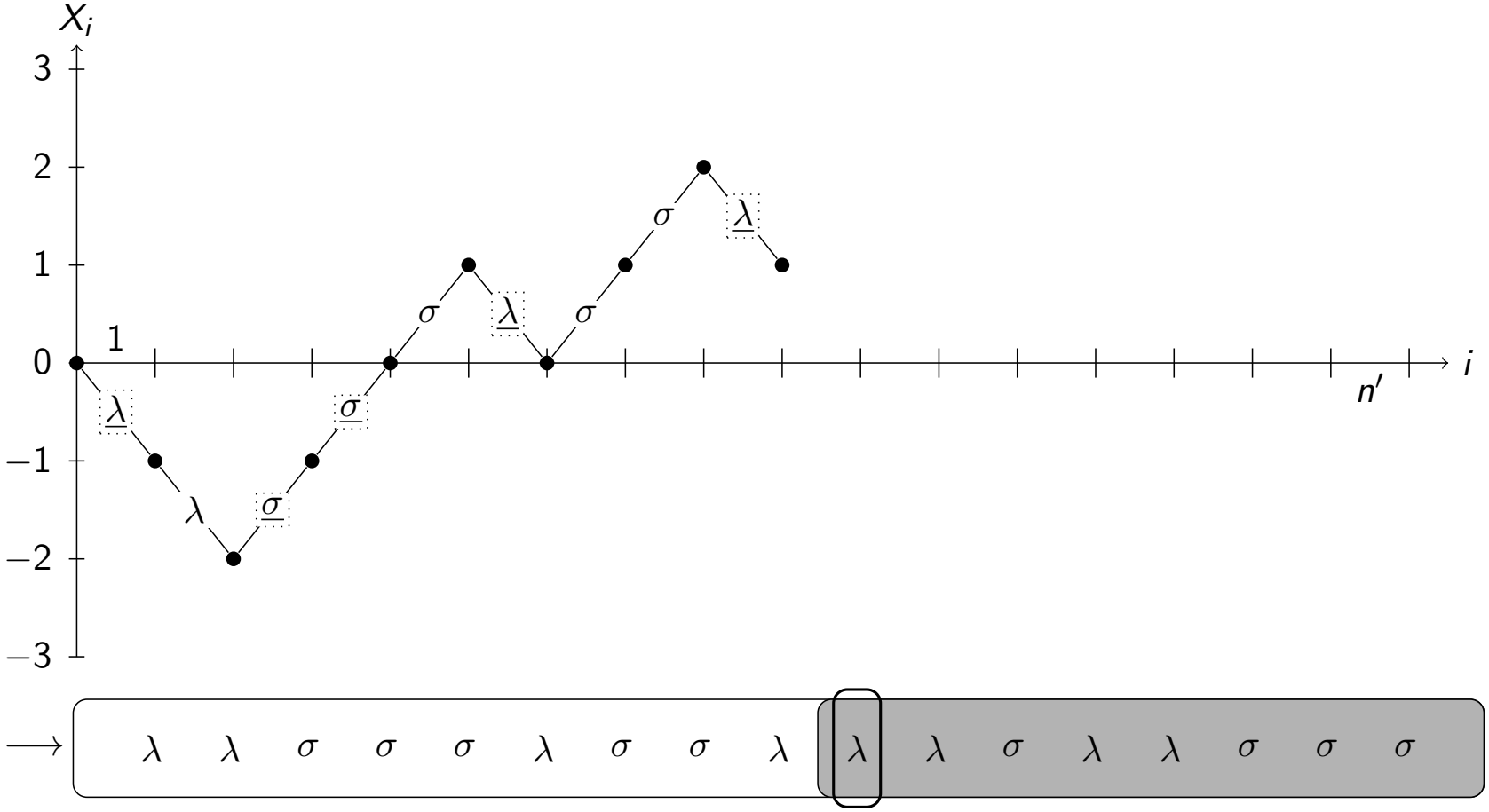
Random Walks & Analysis of Count



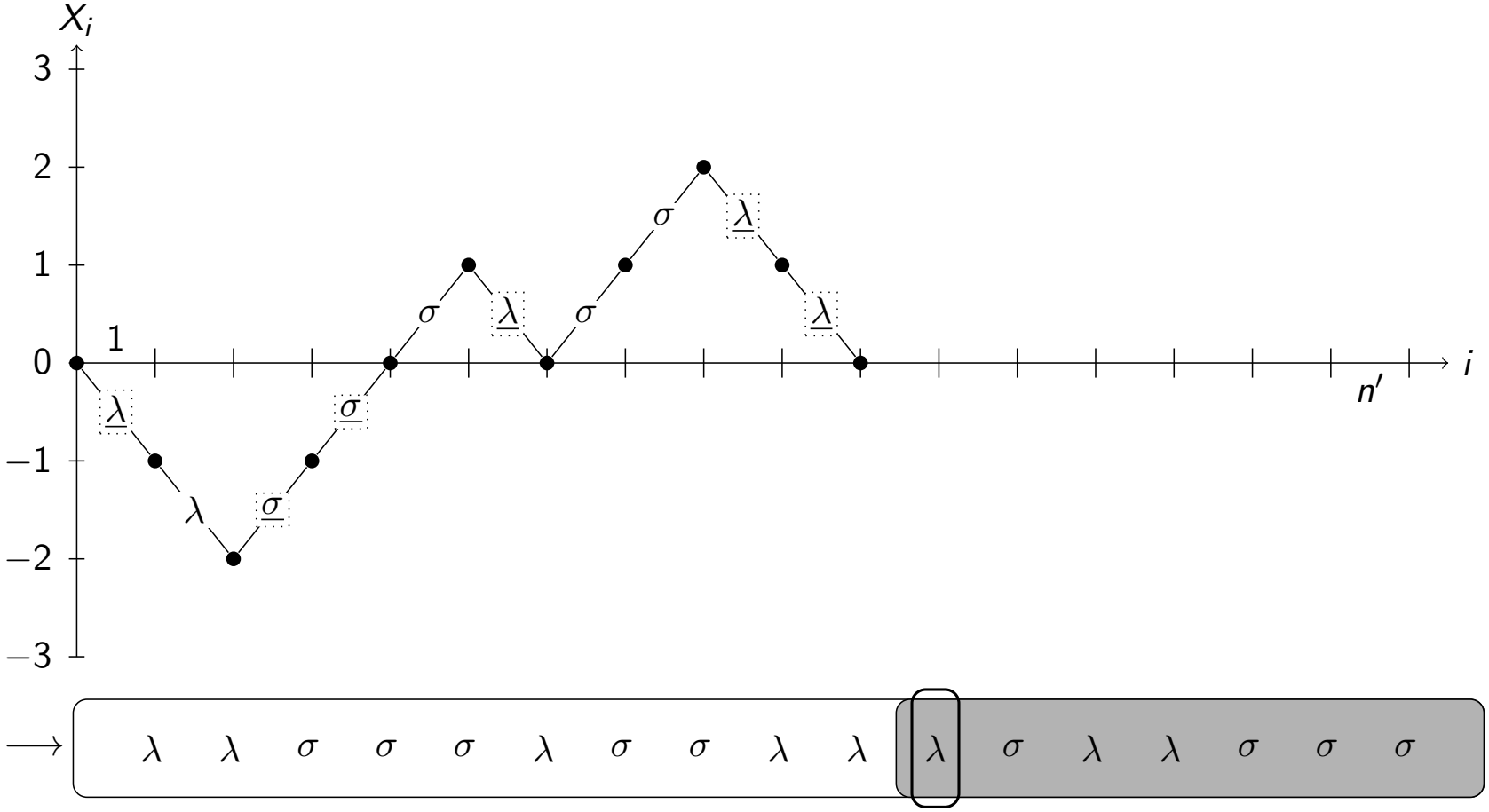
Random Walks & Analysis of Count



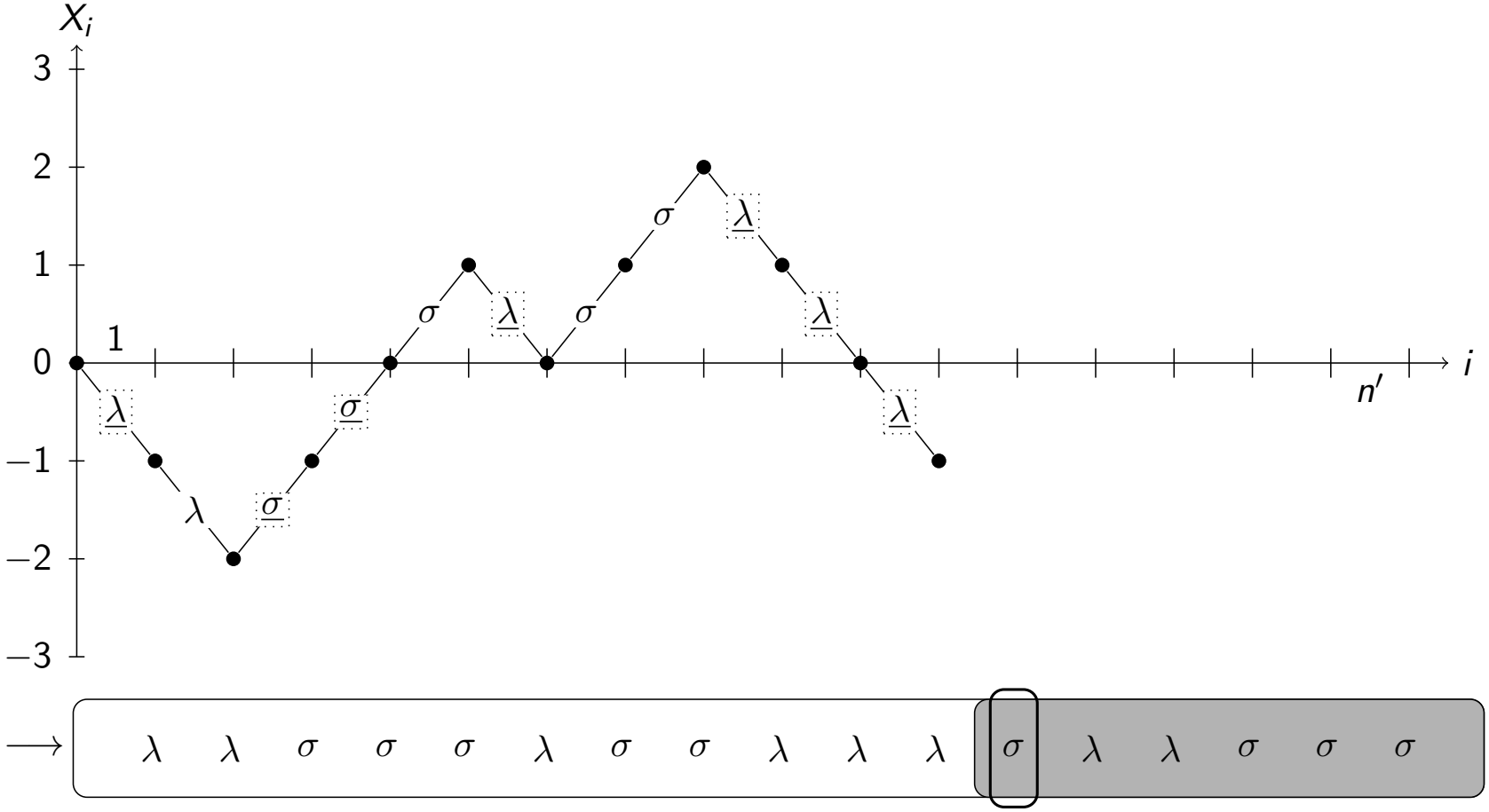
Random Walks & Analysis of Count



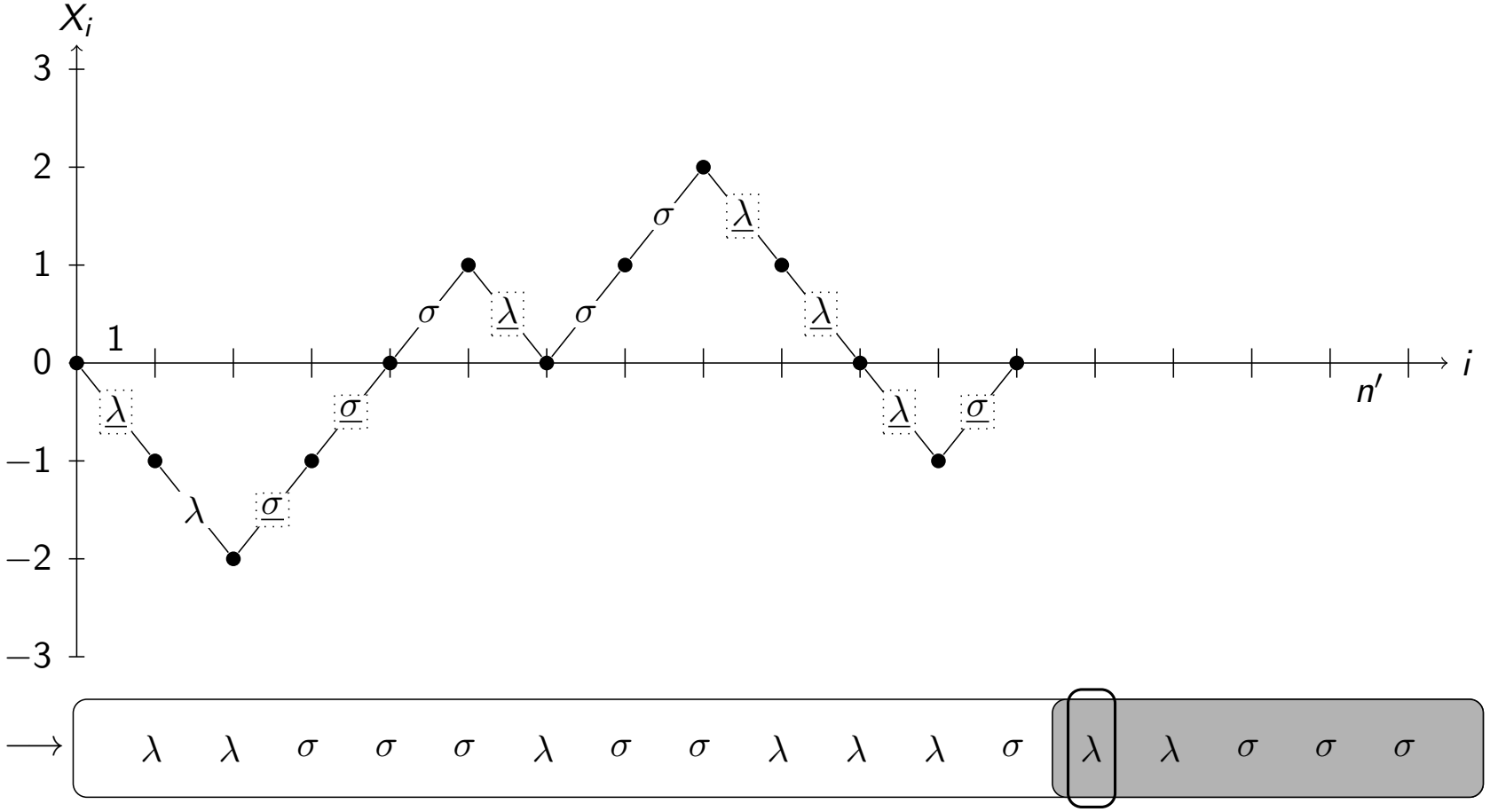
Random Walks & Analysis of Count



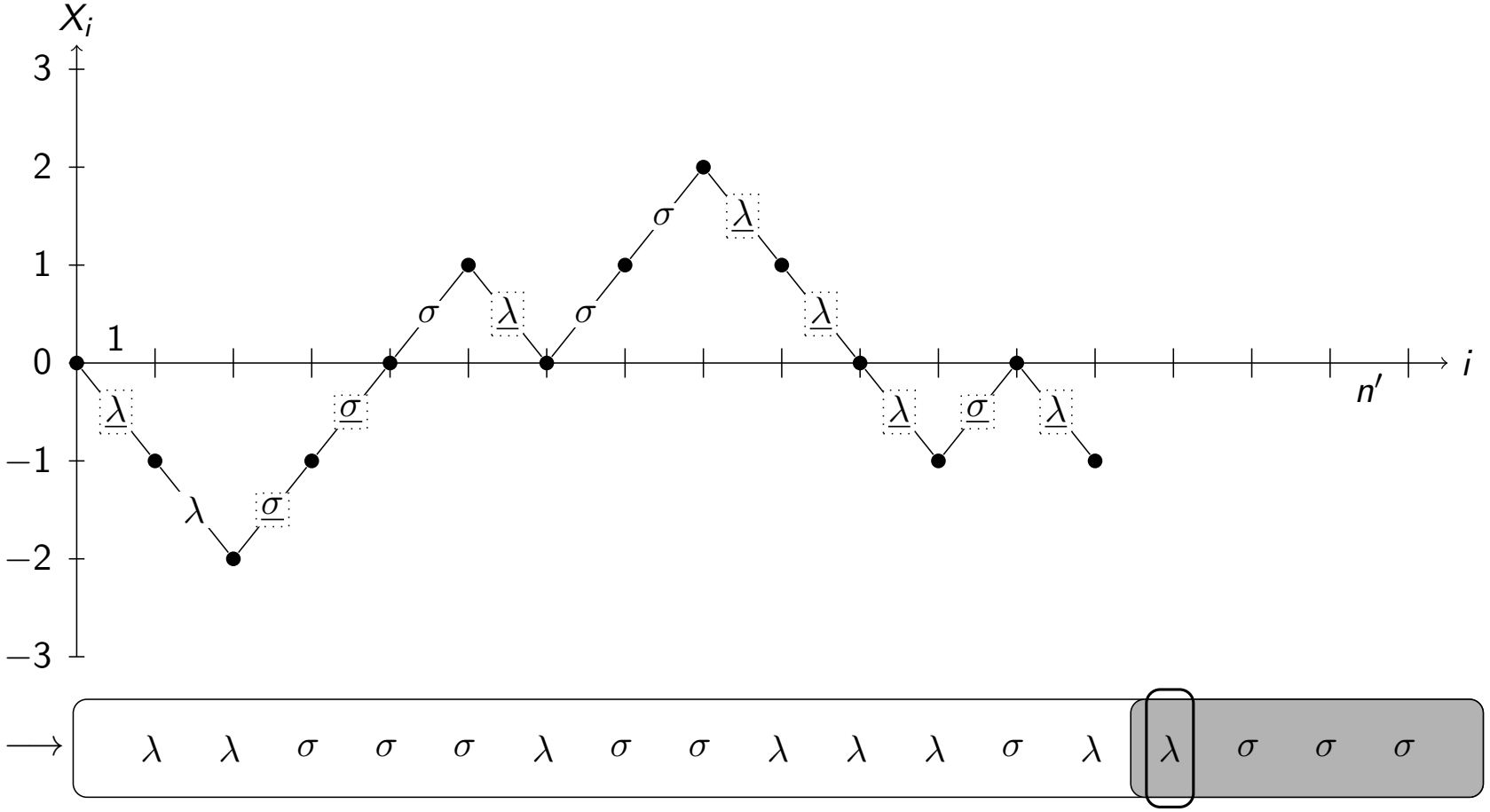
Random Walks & Analysis of Count



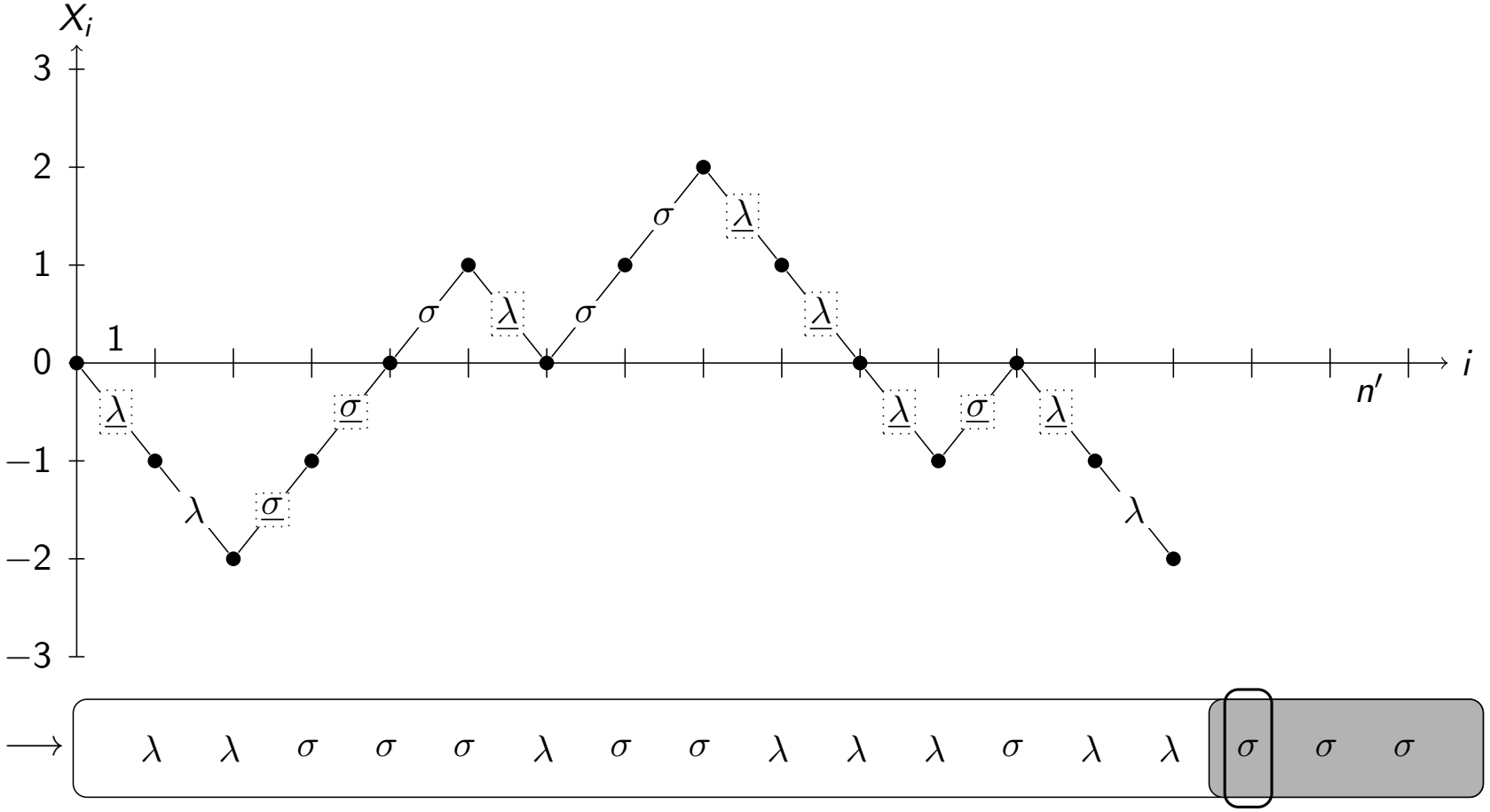
Random Walks & Analysis of Count



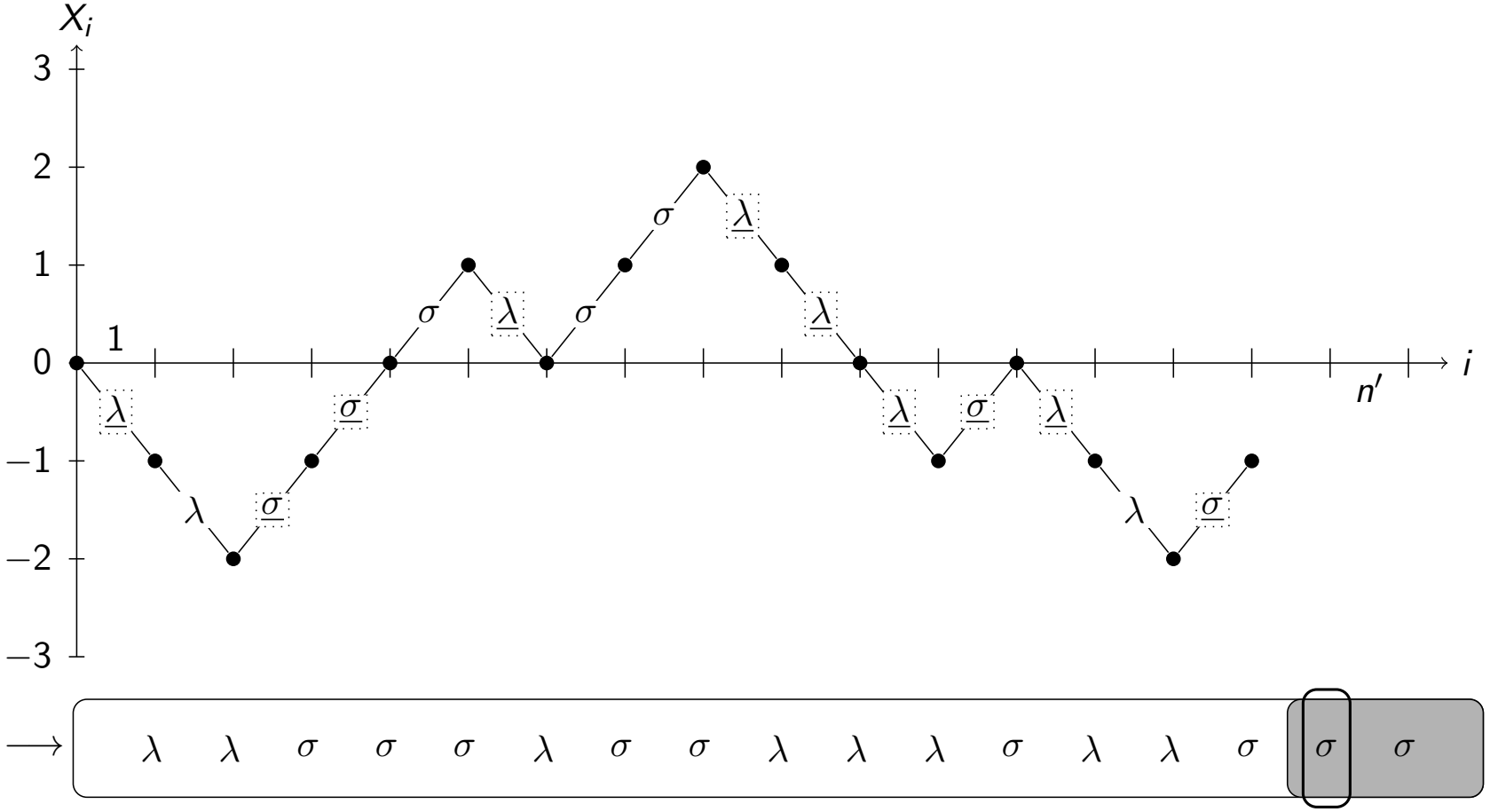
Random Walks & Analysis of Count



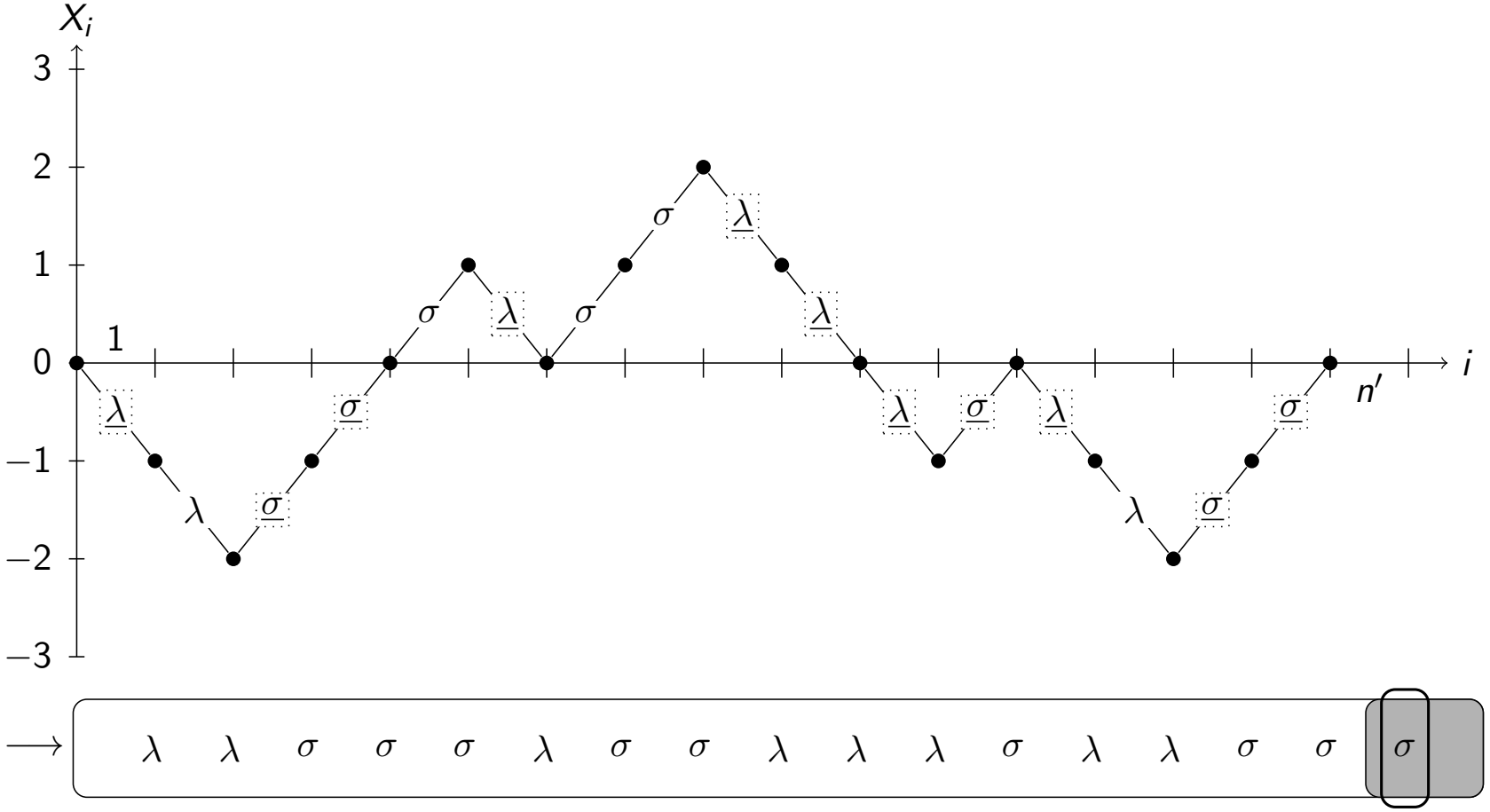
Random Walks & Analysis of Count



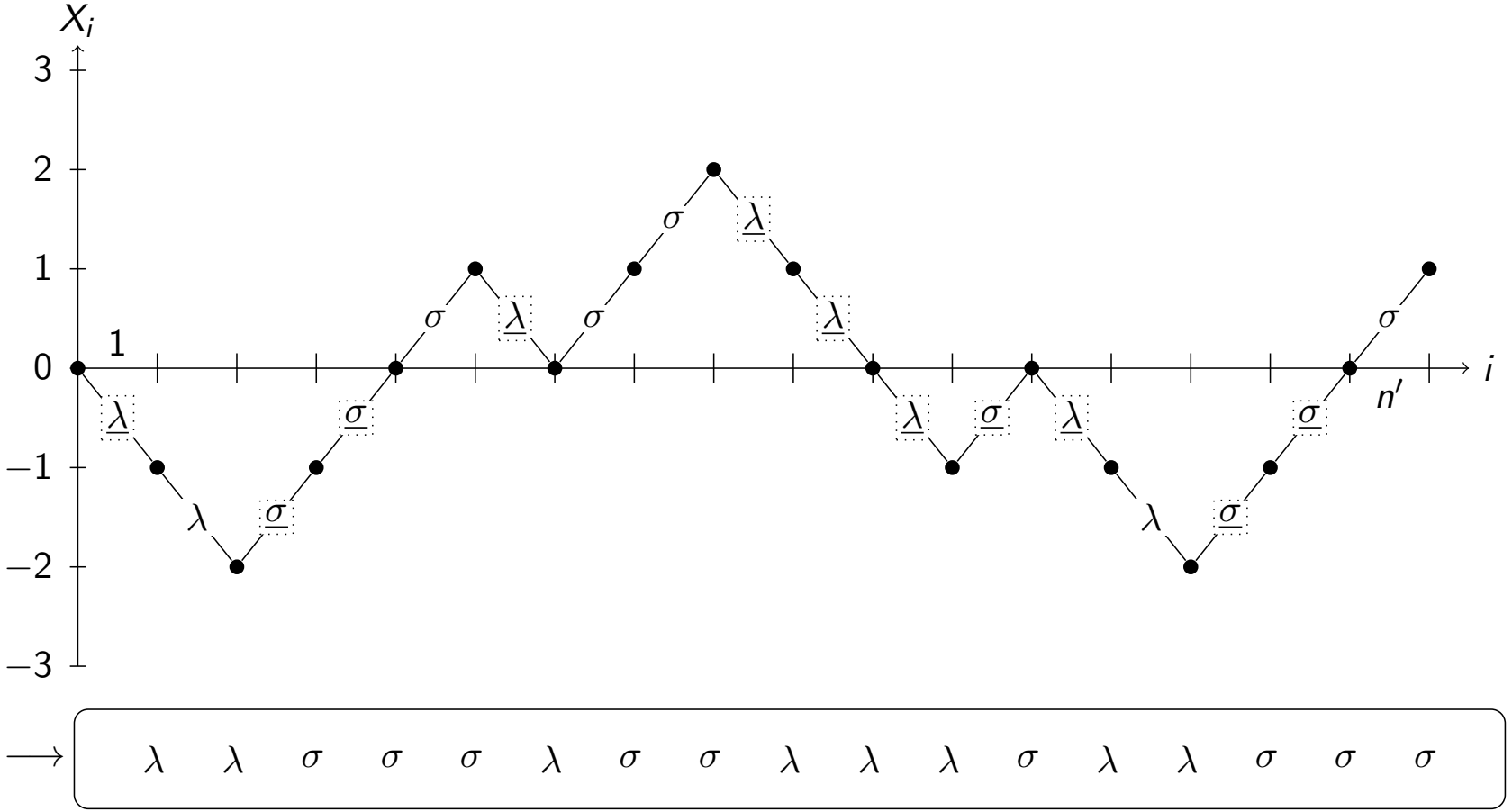
Random Walks & Analysis of Count



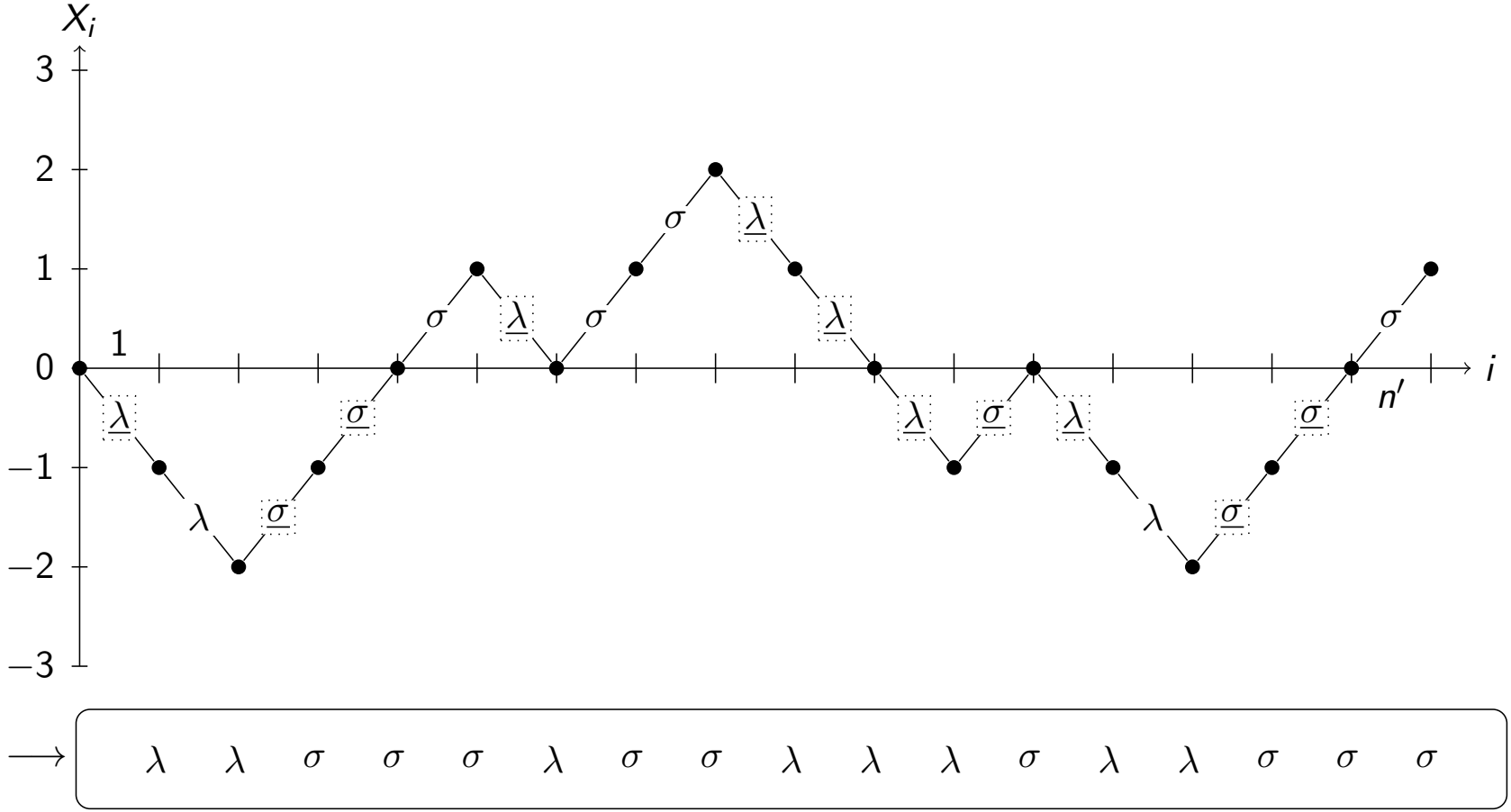
Random Walks & Analysis of Count



Random Walks & Analysis of Count

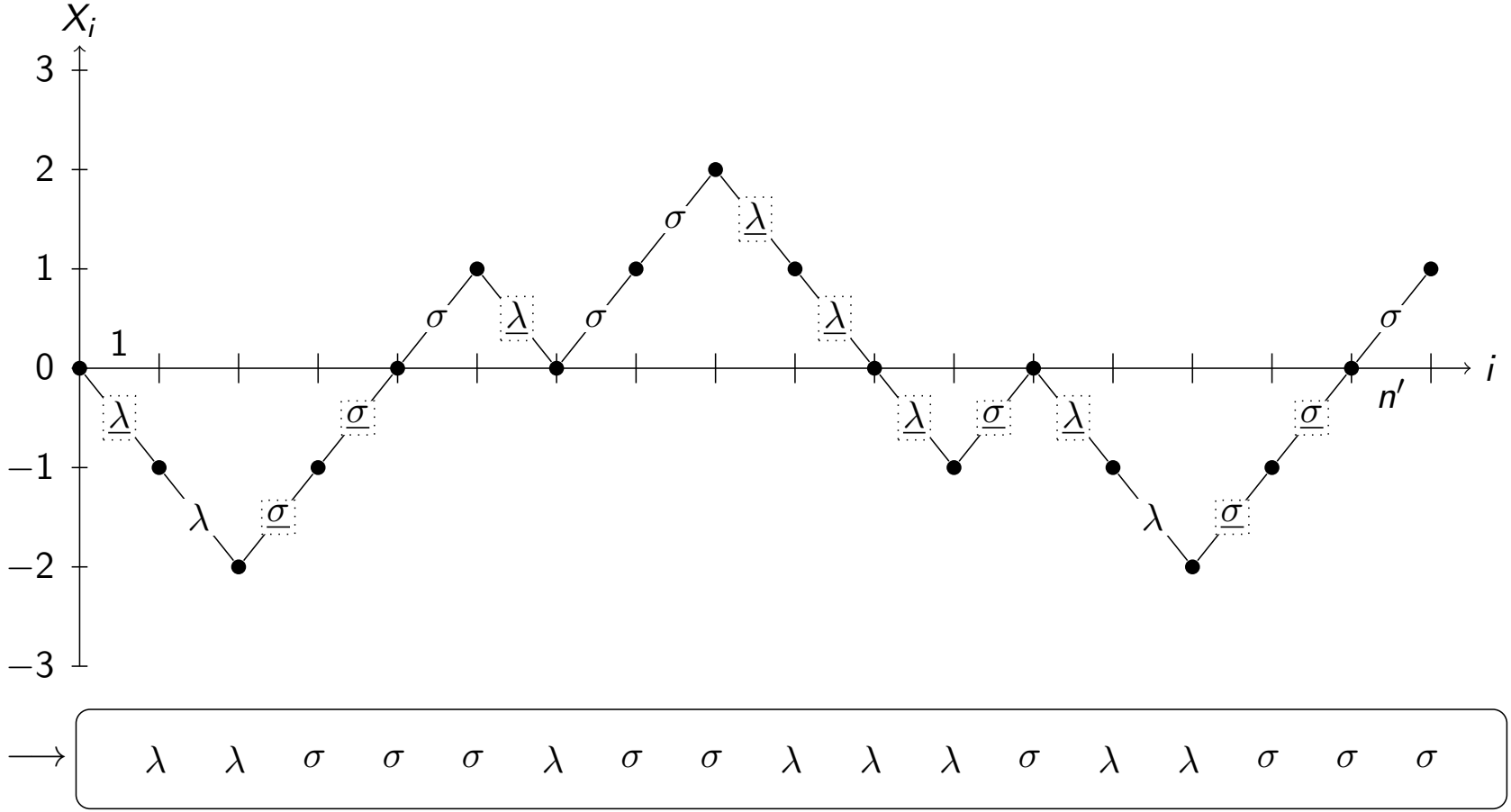


Random Walks & Analysis of Count



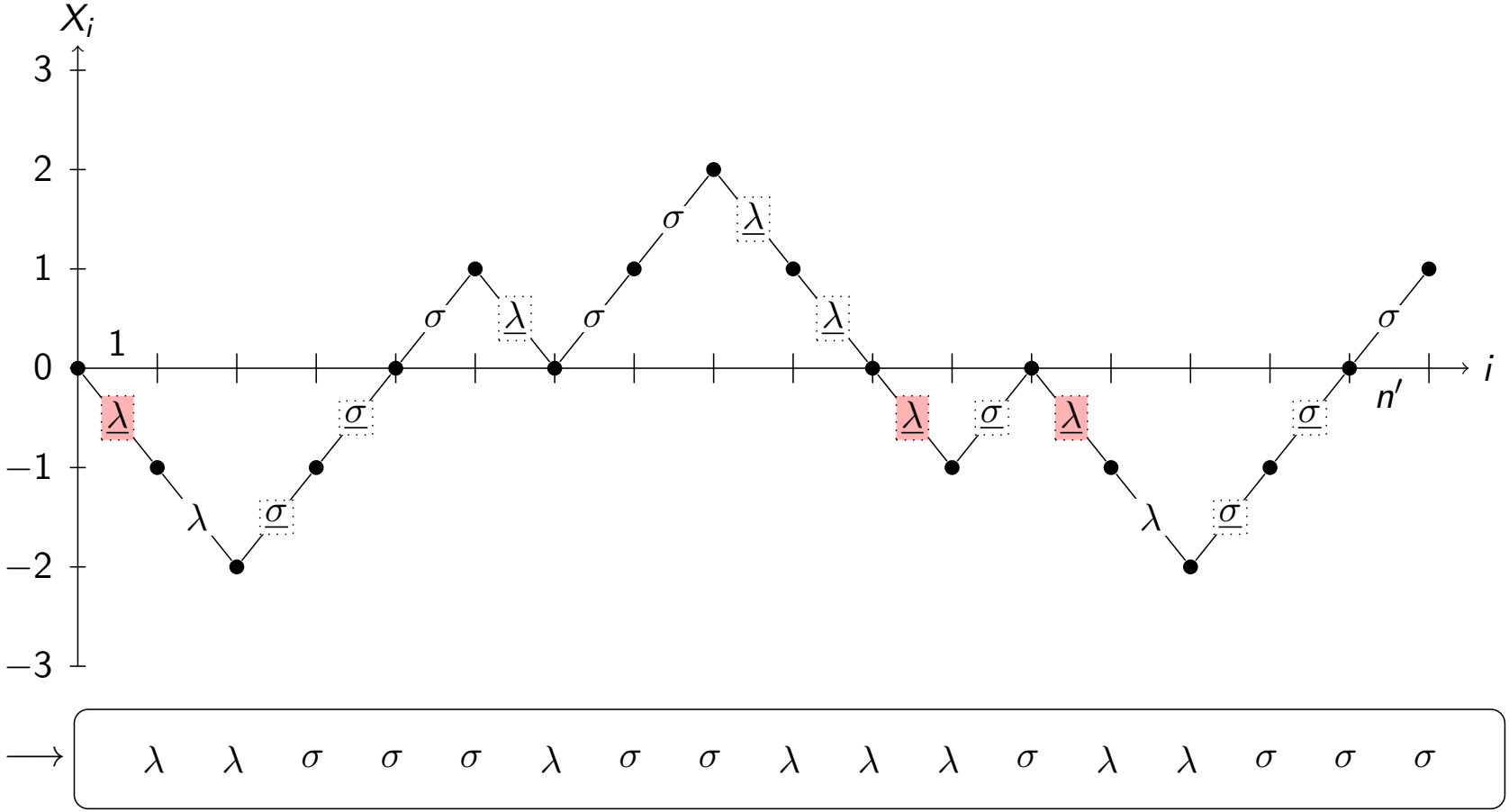
Observation: Extra comparison in round i $\Leftrightarrow$

Random Walks & Analysis of Count



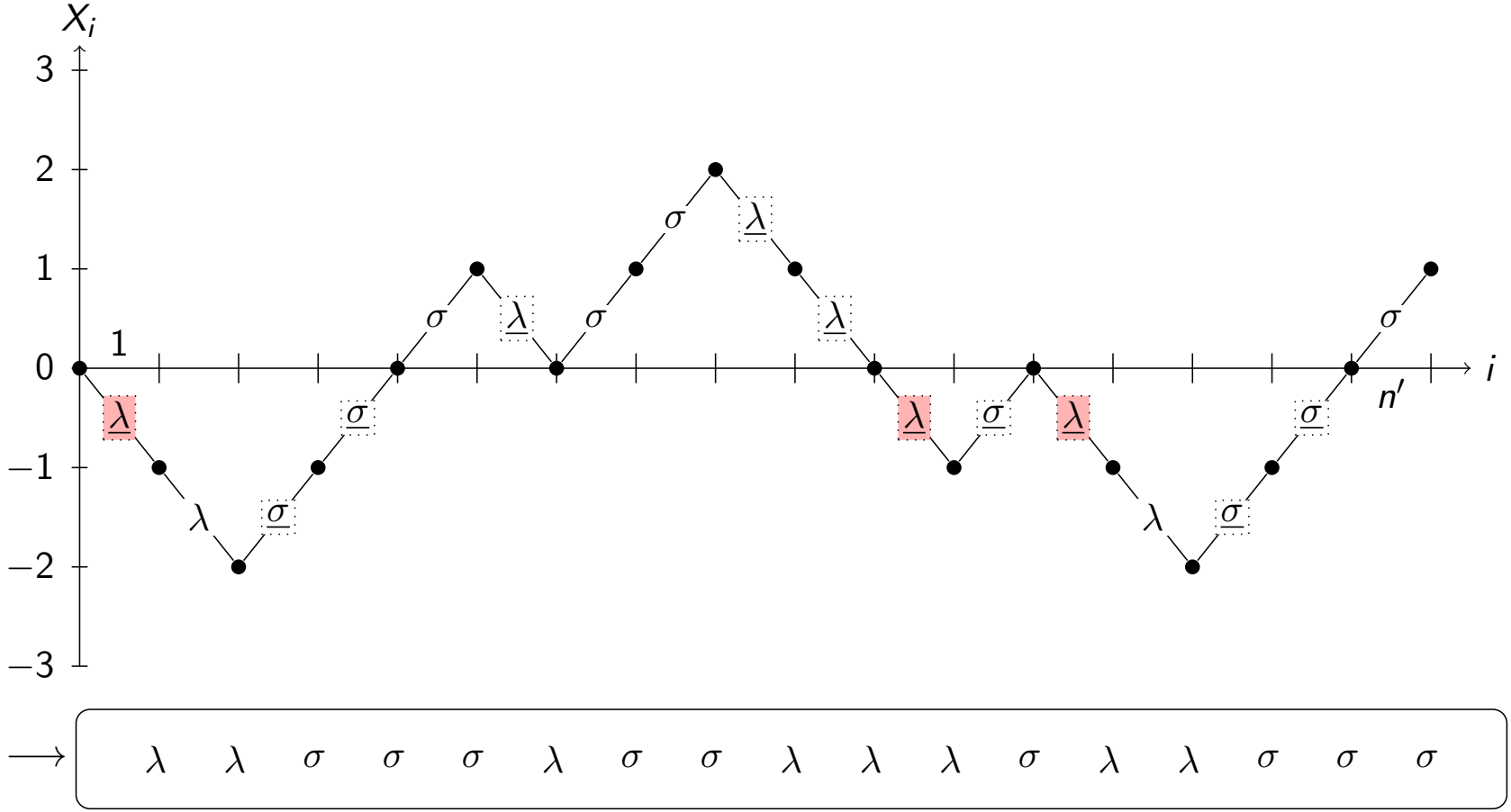
Observation: Extra comparison in round i $\Leftrightarrow$
move towards zero in round i,

Random Walks & Analysis of Count



Observation: Extra comparison in round i $\Leftrightarrow$
move towards zero in round i, plus move down from a zero.

Random Walks & Analysis of Count



Observation: Extra comparison in round i $\Leftrightarrow$
move towards zero in round i, plus move down from a zero.

Exactly $\min(s, \ell)$ many "move towards zero "situations.

Counting Zeros in Random Walks

Define the "number of zeros": $Z_{n'} := \#\{i \mid X_i = 0\}$.

Counting Zeros in Random Walks

Define the "number of zeros": $Z_{n'} := \#\{i \mid X_i = 0\}$.

Can only have zero at odd positions i.

Counting Zeros in Random Walks

Define the "number of zeros": $Z_{n'} := \#\{i \mid X_i = 0\}$.

Can only have zero at odd positions i.

Central argument

For each $i \in \{1, 3, 5, \ldots\}$, $i \leq n' + 1$,

$$\mathbb{P}(X_i = 0) = \frac{1}{i}.$$

Counting Zeros in Random Walks

Define the "number of zeros": $Z_{n'} := \#\{i \mid X_i = 0\}$.

Can only have zero at odd positions i.

Central argument

For each $i \in \{1, 3, 5, \ldots\}$, $i \leq n' + 1$,

$$\mathbb{P}(X_i = 0) = \frac{1}{i}.$$

Linearity of expectation yields:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{1 \leq i \leq n'+1 \\ i \text{ odd}}} \frac{1}{i}.$$

Counting Zeros in Random Walks

Define the "number of zeros": $Z_{n'} := \#\{i \mid X_i = 0\}$.

Can only have zero at odd positions i.

Central argument

For each $i \in \{1, 3, 5, \ldots\}$, $i \leq n' + 1$,

$$\mathbb{P}(X_i = 0) = \frac{1}{i}.$$

Linearity of expectation yields:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{1 \leq i \leq n'+1 \\ i \text{ odd}}} \frac{1}{i}.$$

$$\mathbb{E}(\#(\text{extra comps}) \mid s + \ell = n') = \min(s, \ell) + \frac{1}{2}\left(\mathbb{E}(Z_{n'}) - \frac{[n' \text{ even}]}{n' + 1}\right).$$

Exact Average Partitioning Cost

Back to general situation including medium elements.
Averaging over all pivot choices yields:

$$\mathbb{E}(P_n) = \frac{3n}{2} - \frac{19}{8} + \frac{2 + (-1)^n}{8(n - [n \text{ even}])} + \frac{1}{2} \sum_{\substack{1 \le i \le n-2 \\ i \text{ odd}}} \frac{1}{i}.$$

Can also write down **generating functions**.

Exact Average Sorting Cost

Solving dual-pivot quicksort recurrence with generating functions [Wild'13] gives expected comparison count, explicitly, and then asymptotically:

$$\mathbb{E}(C_n) = 1.8 n \ln n + A' n + B' \ln n + C' + O\left(\frac{1}{n}\right),$$

where $A' \approx -2.38$, B' $= 1.675$, $C' \approx 1.82$.

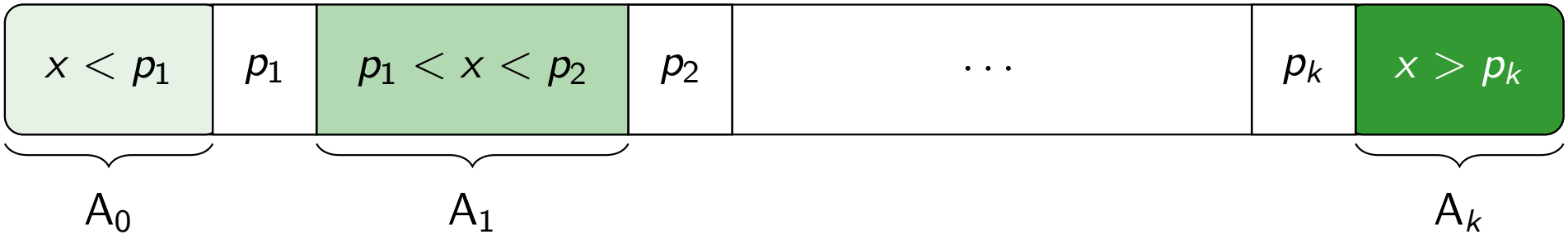
Part II

Memory Accesses in Multi-Pivot Quicksort

http://arxiv.org/abs/1510.04676

Quicksort with k Pivots

- Input: Random permutation of $\{1, \dots, n\}$.
- First k entries sorted are pivots $p_1, \dots, p_k$.
- Partitioning: Split $n - k$ remaining entries x into $k + 1$ classes



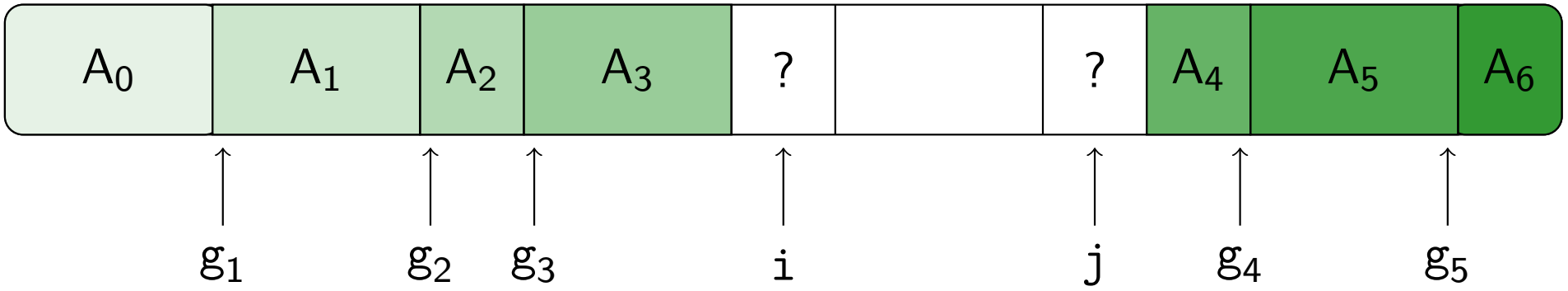
Then sort classes recursively.

Other Cost Measures

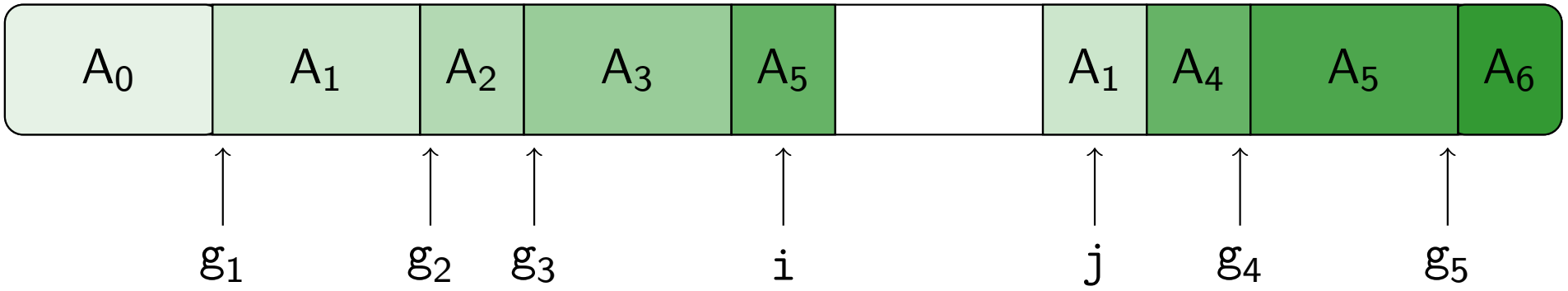
- Number of branch misses: Thorough analysis of classical QS and Yaroslavskiy's algorithm by Martínez/Nebel/Wild (ANALCO'15).

  Conclusion: No improvement over classical QS.
- Number of cache misses: (Kushagra et al., ALENEX'14):
  Dual-Pivot and three-pivot algorithm cause
  far fewer cache misses than classical QS.

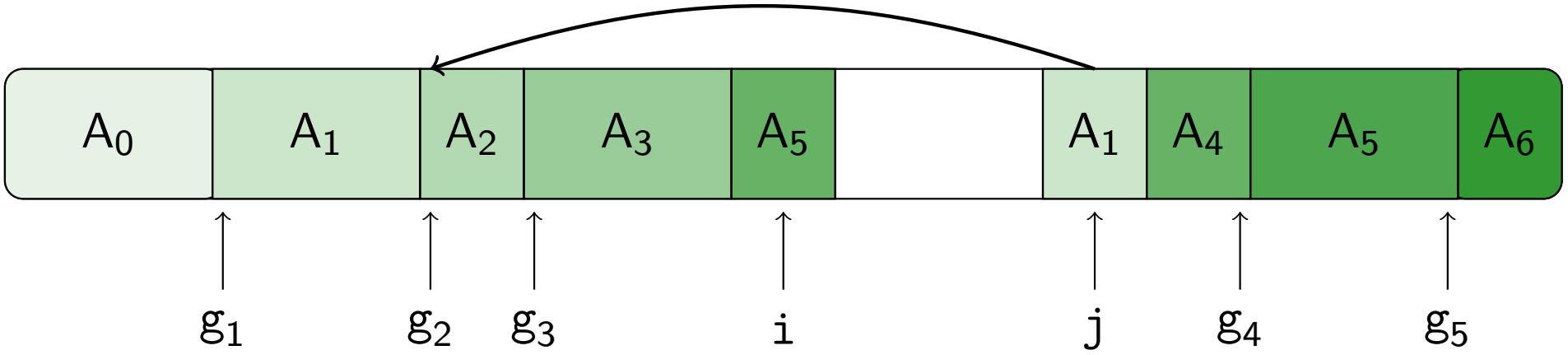
A Generalized Partitioning Algorithm



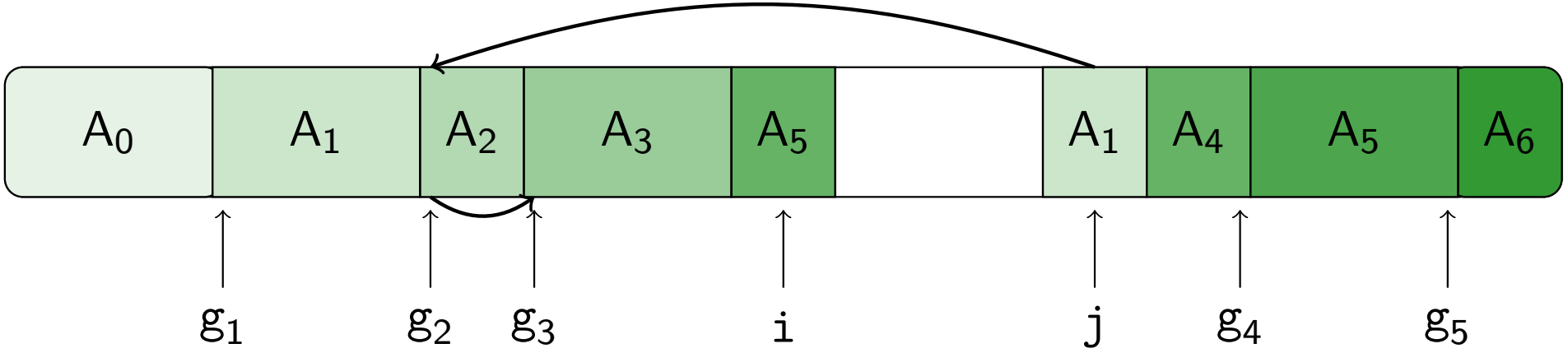
A Generalized Partitioning Algorithm



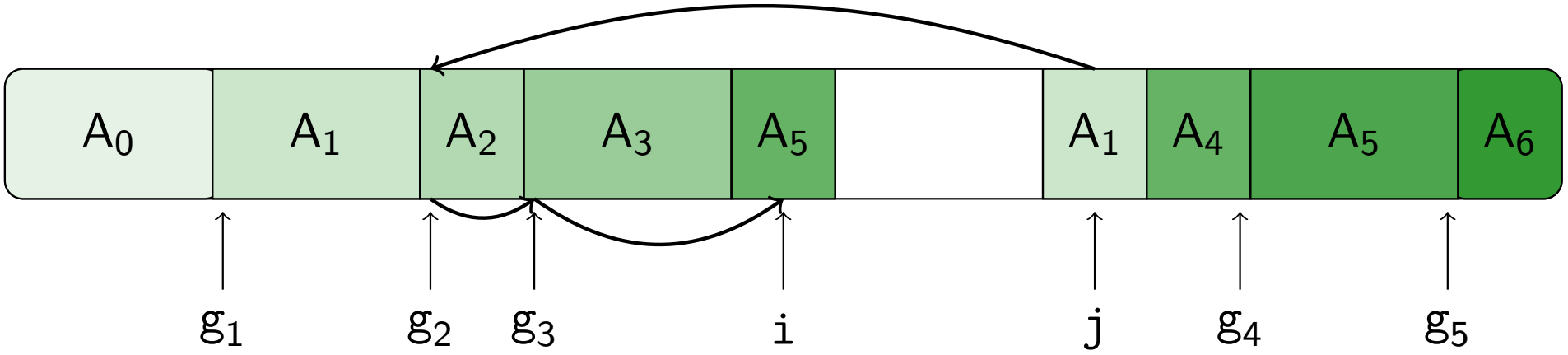
A Generalized Partitioning Algorithm



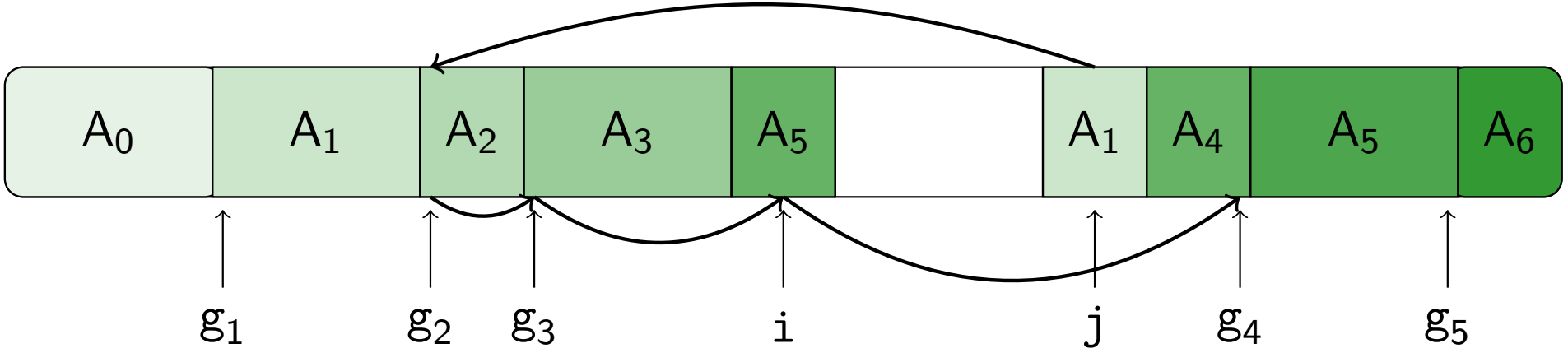
A Generalized Partitioning Algorithm



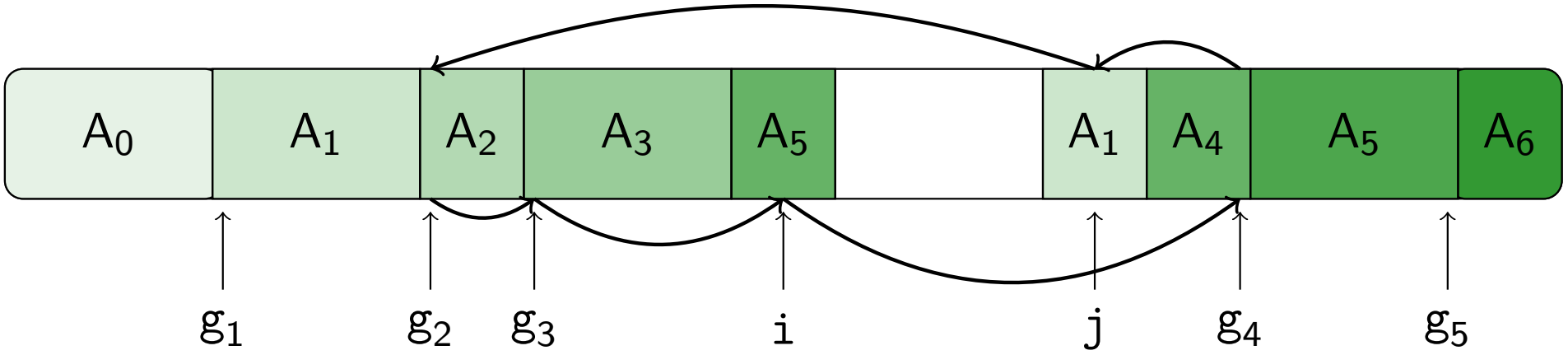
A Generalized Partitioning Algorithm



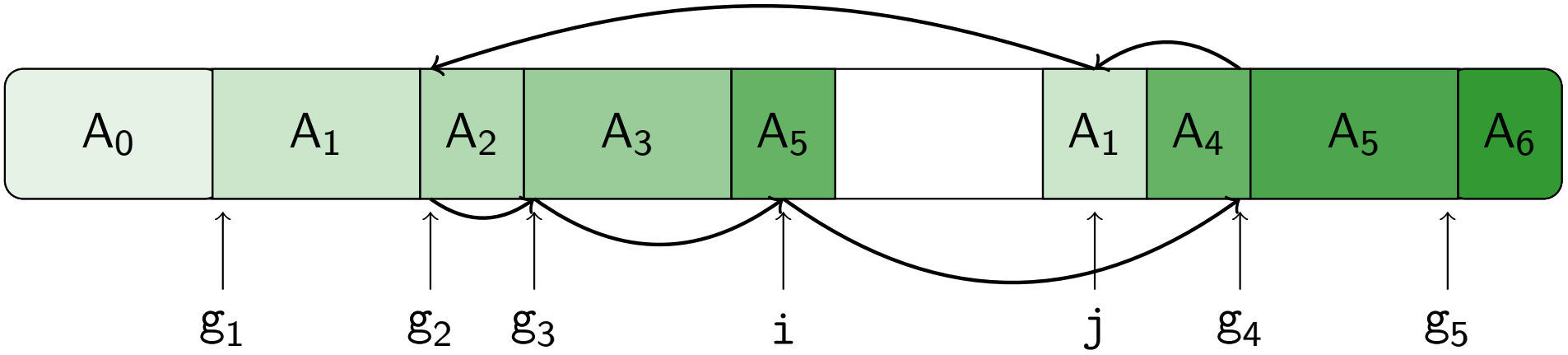
A Generalized Partitioning Algorithm



A Generalized Partitioning Algorithm

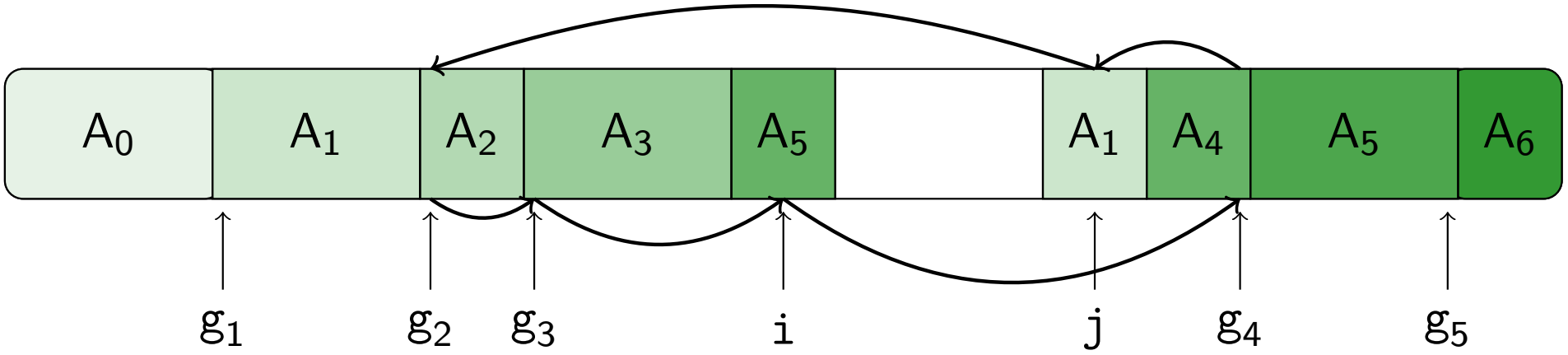


A Generalized Partitioning Algorithm



5 pointer visits.

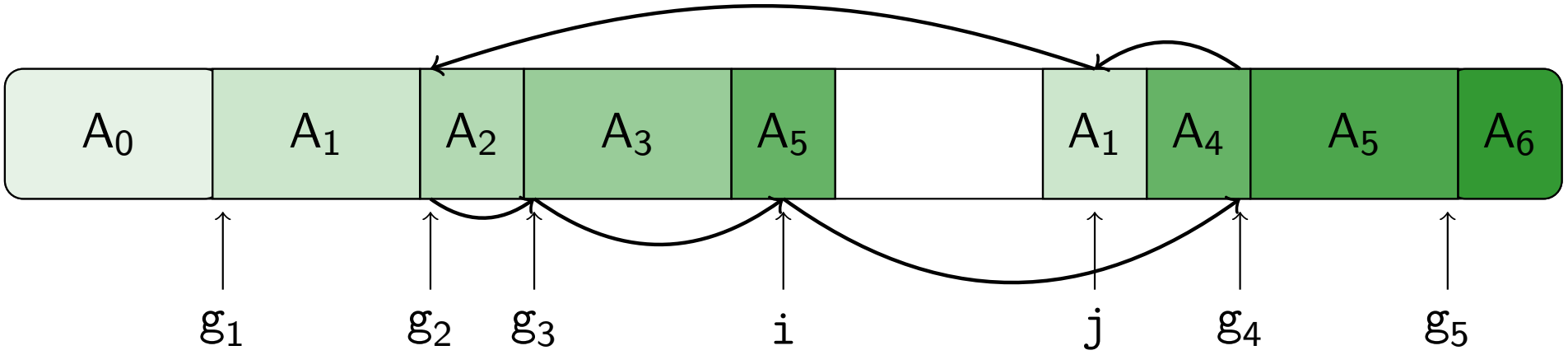
A Generalized Partitioning Algorithm



5 pointer visits.

Analysis for partitioning yields:

A Generalized Partitioning Algorithm



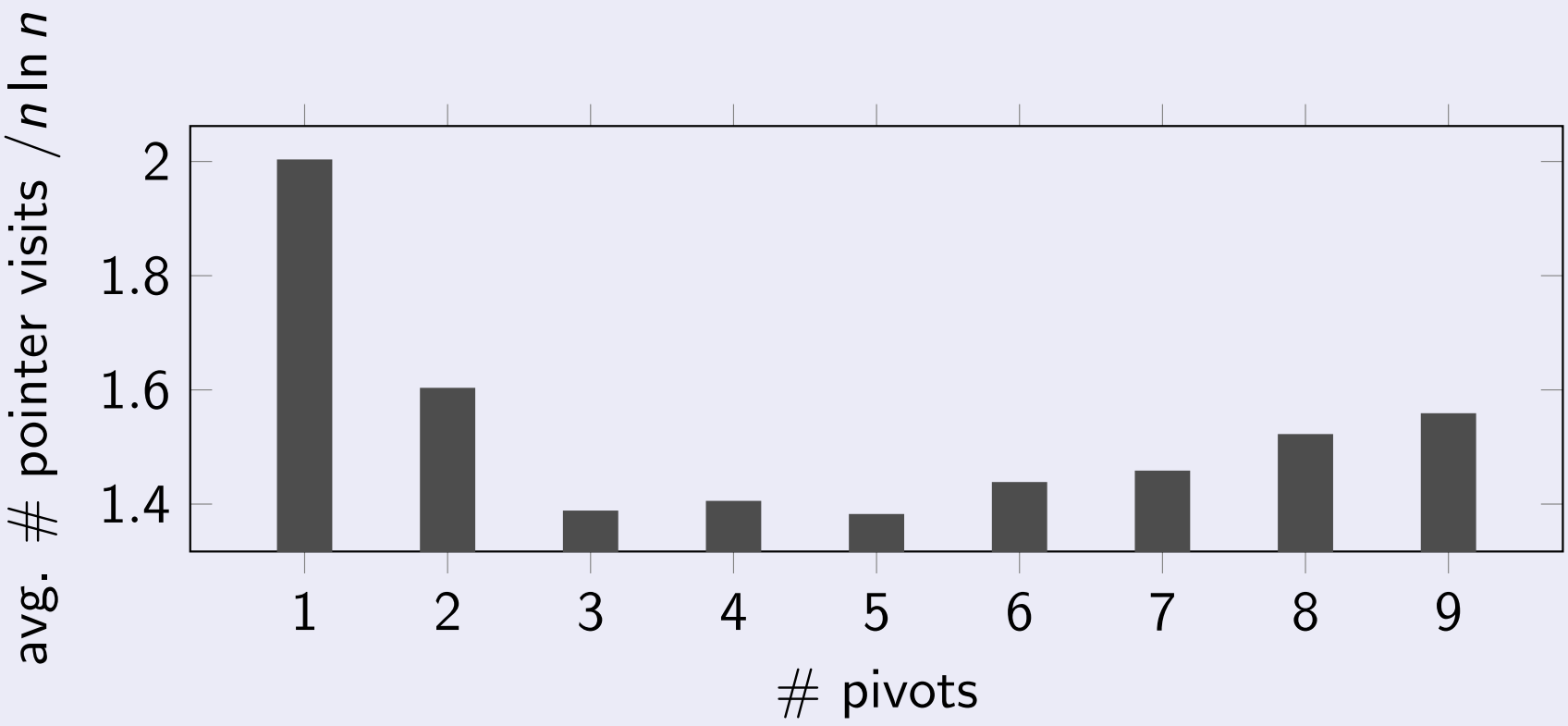
5 pointer visits.

Analysis for partitioning yields:

$$\mathbb{E}(P_n^{\text{ptr-visit}}) = \begin{cases} \dfrac{k+3}{4} \cdot (n-k), & \text{for odd } k, \\ \left(\dfrac{k+3}{4} + \dfrac{3}{k+1}\right) \cdot (n-k), & \text{for even } k. \end{cases}$$

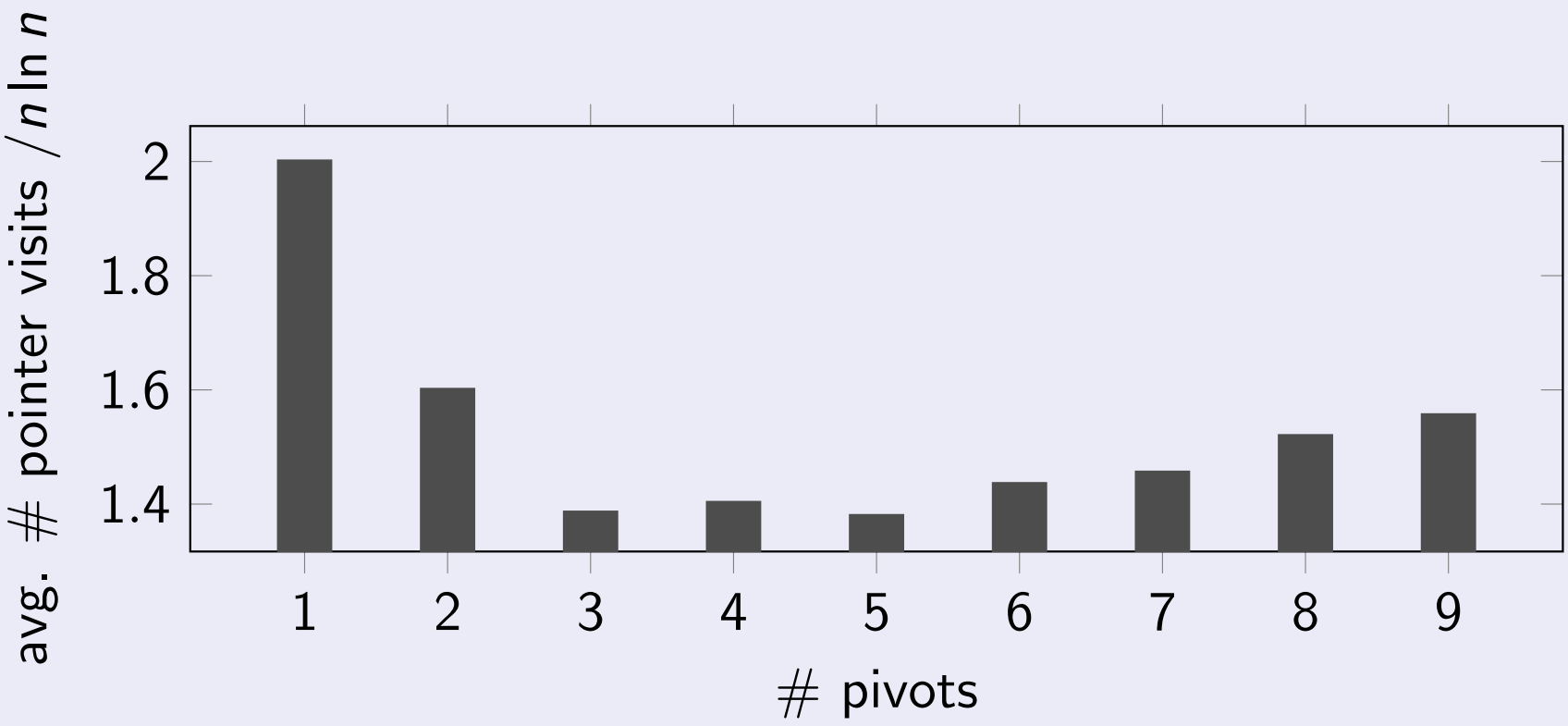
A Generalized Partitioning Algorithm

"Pointer visits" for sorting with k pivots: Minimum at $k = 5$.



A Generalized Partitioning Algorithm

"Pointer visits" for sorting with k pivots: Minimum at $k = 5$.



Observations:

- "pointer visits" predict L1 cache misses accurately
- closest w.r.t. predicting empirical running time

Conclusion & Open Problems

Theory:

- Intuitive algorithm "Count" is comparison-optimal
- Exact average comparison count for "Count"
- "Count" variant for multi-pivot quicksort also optimal
- But: Average comparison count unknown for $k \geq 5$!

Theory & Practice:

- Multi-Pivot quicksort has advantages in cache behavior
- Does "pointer visits" / "scanned elements" cost measure produce reliable running time predictions for other algorithms?

Conclusion & Open Problems

Theory:

- Intuitive algorithm "Count" is comparison-optimal
- Exact average comparison count for "Count"
- "Count" variant for multi-pivot quicksort also optimal
- But: Average comparison count unknown for $k \geq 5$!

Theory & Practice:

- Multi-Pivot quicksort has advantages in cache behavior
- Does "pointer visits" / "scanned elements" cost measure produce reliable running time predictions for other algorithms?

Thank you!

Running Time Experiments