On the Analysis of Two Fundamental Randomized Algorithms

Multi-Pivot Quicksort and Efficient Hash Functions

Martin Aumüller

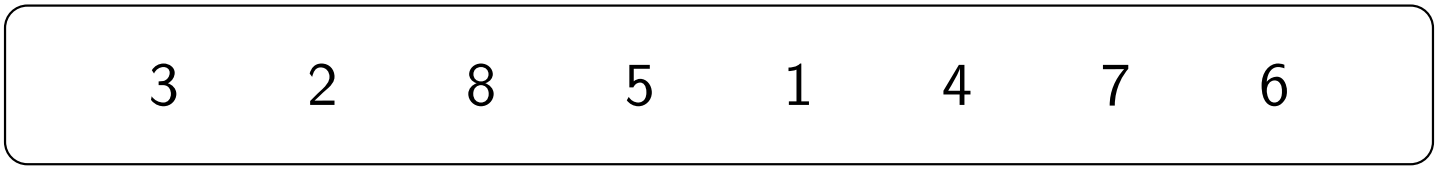
Technische Universität Ilmenau

Wissenschaftliche Aussprache
23. Juni 2015

Part I

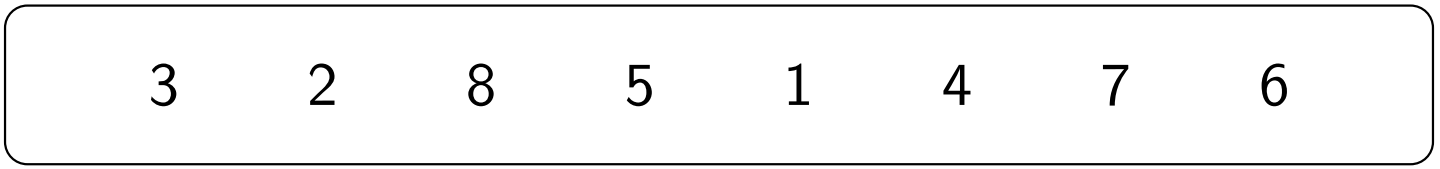
Multi-Pivot Quicksort

Classical Quicksort



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

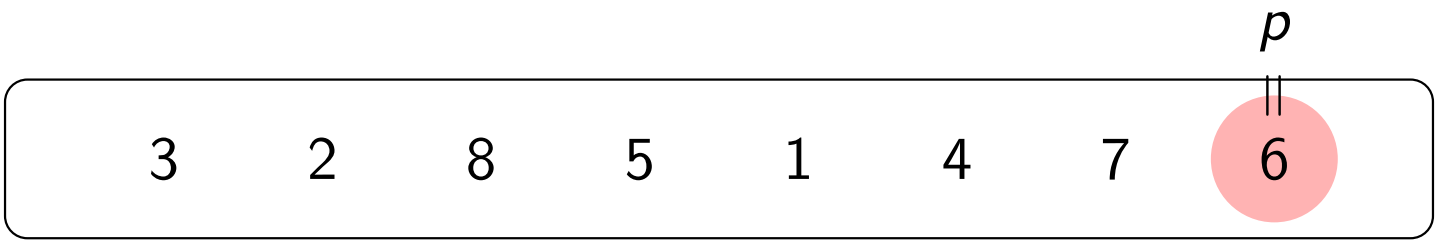
Classical Quicksort



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

1. Choose a pivot p.

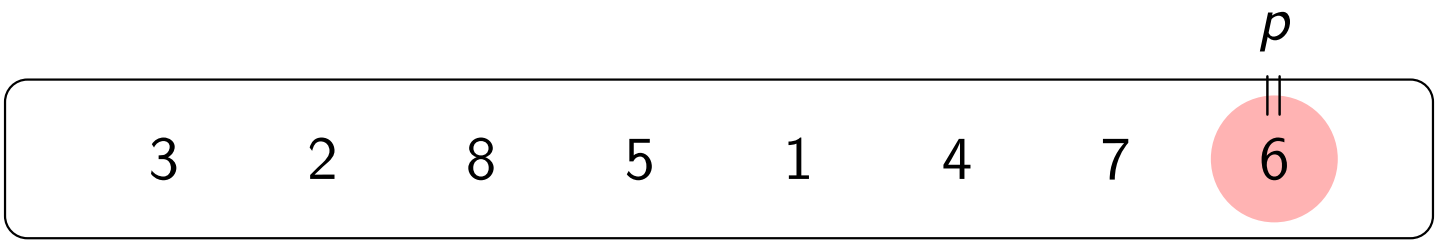
Classical Quicksort



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

1. Choose a pivot p.

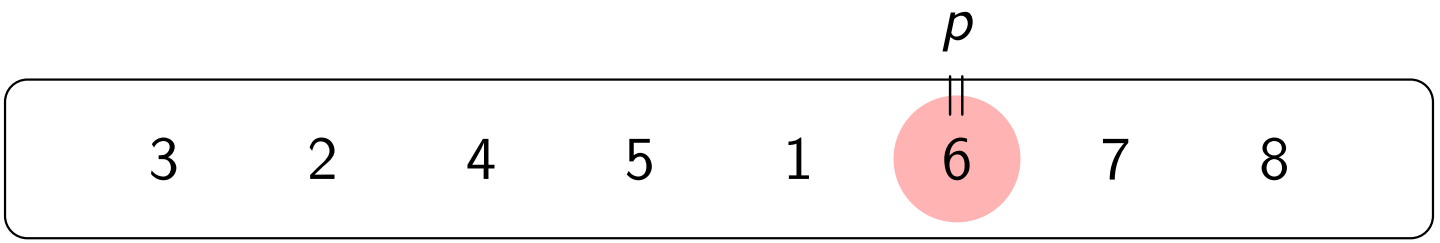
Classical Quicksort



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

1. Choose a pivot p.
2. Partition, i.e., re-arrange elements (e.g. by Hoare's method);

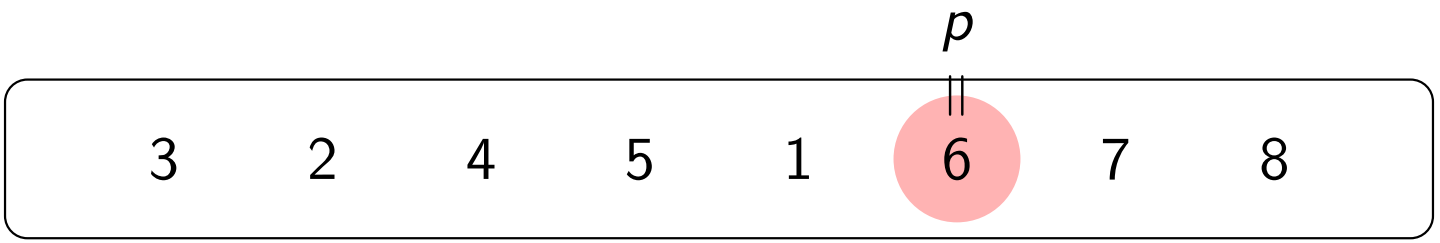
Classical Quicksort



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

1. Choose a pivot p.
2. Partition, i.e., re-arrange elements (e.g. by Hoare's method);

Classical Quicksort



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

1. Choose a pivot p.
2. Partition, i.e., re-arrange elements (e.g. by Hoare's method);

3. Use recursion to sort the two subarrays.

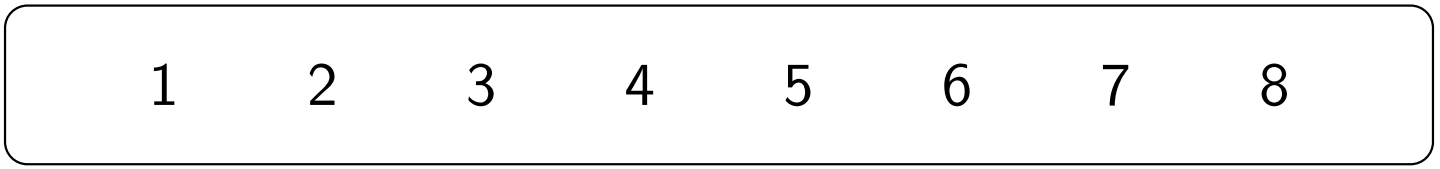
Classical Quicksort

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

1. Choose a pivot p.
2. Partition, i.e., re-arrange elements (e.g. by Hoare's method);

3. Use recursion to sort the two subarrays.

Classical Quicksort

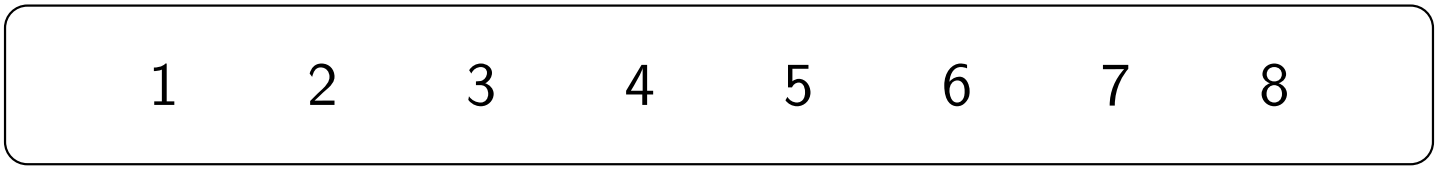


Input: Permutation of $\{1, \ldots, n\}$.

1. Choose a pivot p.
2. Partition, i.e., re-arrange elements (e.g. by Hoare's method);

3. Use recursion to sort the two subarrays.

Classical Quicksort



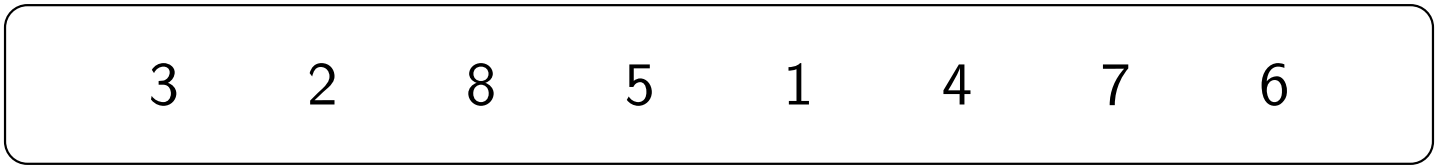
Input: Permutation of $\{1, \ldots, n\}$.

1. Choose a pivot p.
2. Partition, i.e., re-arrange elements (e.g. by Hoare's method);

3. Use recursion to sort the two subarrays.

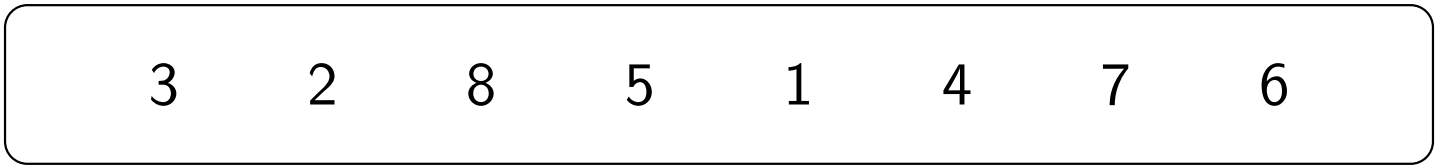
Expected number of comparisons: $2n \ln n - O(n)$.

Dual-Pivot Quicksort



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

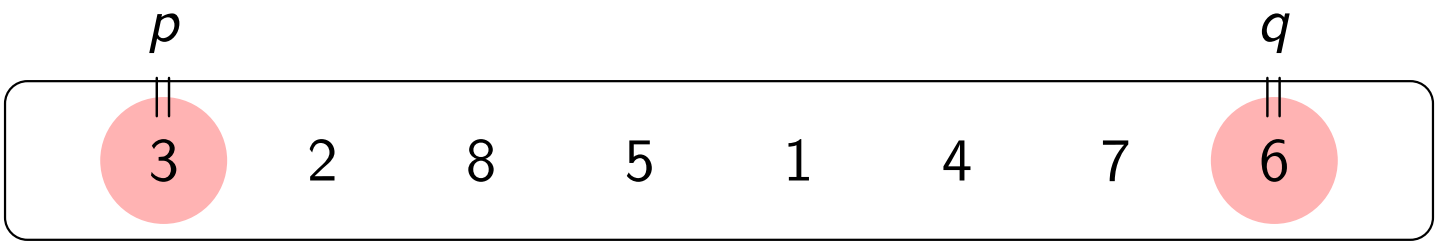
Dual-Pivot Quicksort



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

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

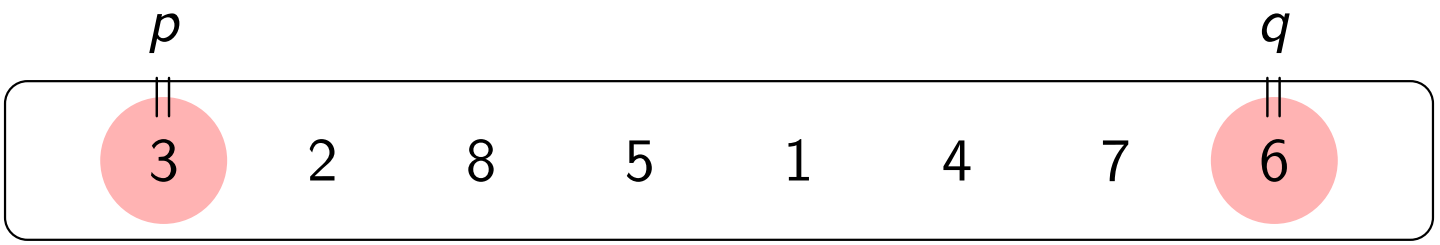
Dual-Pivot Quicksort



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

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

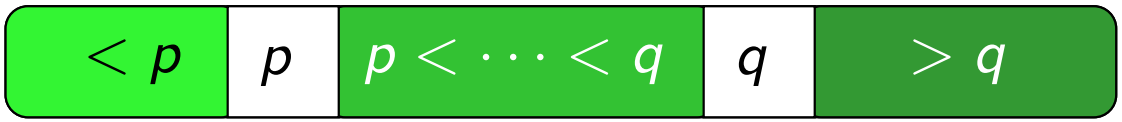
Dual-Pivot Quicksort



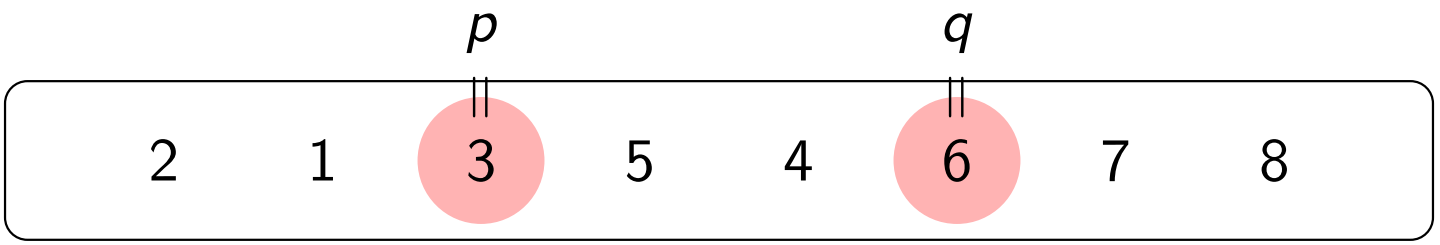
Input: Permutation of $\{1, \ldots, n\}$.

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

Partition:



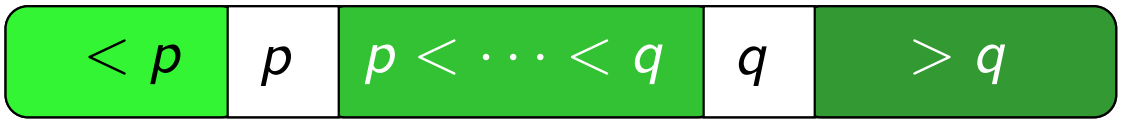
Dual-Pivot Quicksort



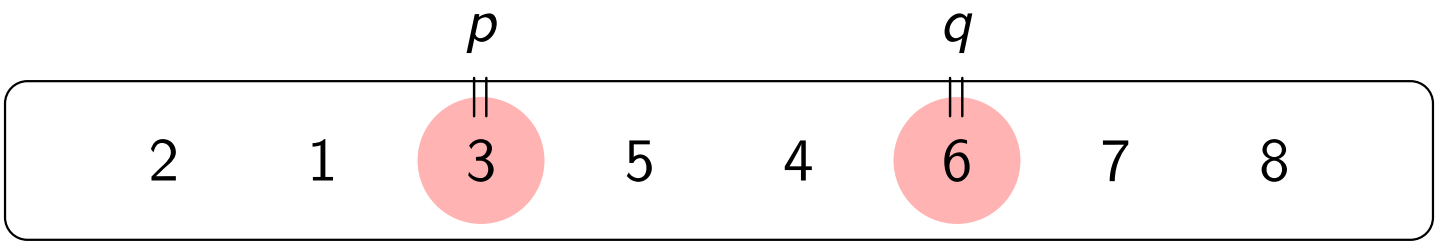
Input: Permutation of $\{1, \ldots, n\}$.

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

Partition:



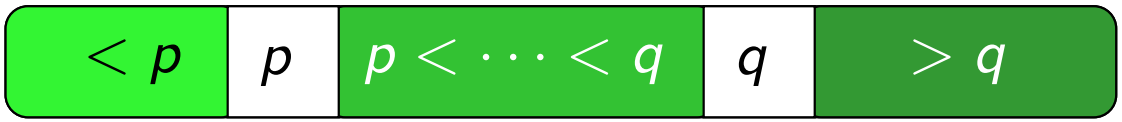
Dual-Pivot Quicksort



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

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

Partition:



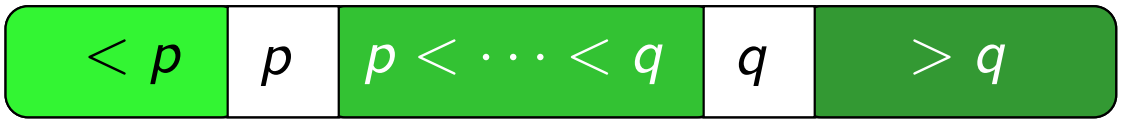
3. Use recursion to sort the **three** subarrays.

Dual-Pivot Quicksort

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

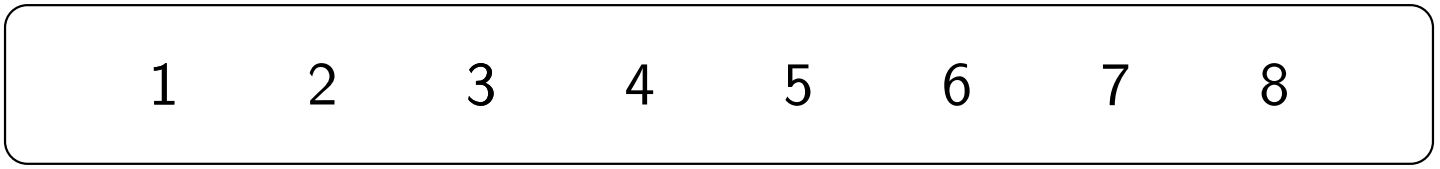
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



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

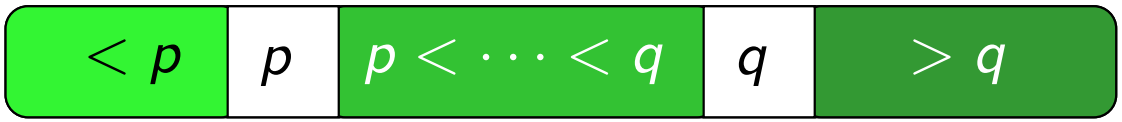
Dual-Pivot Quicksort



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

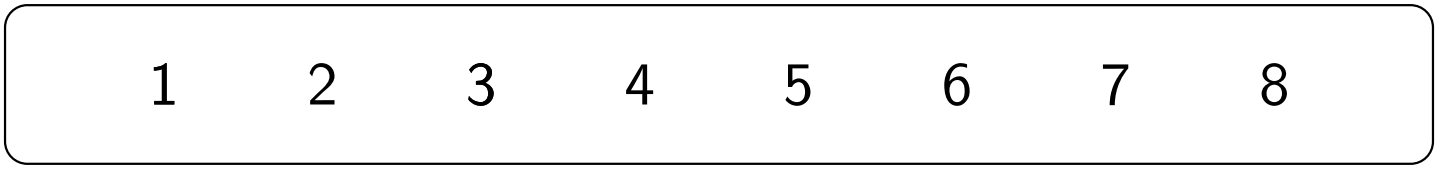
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

   Partition:



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

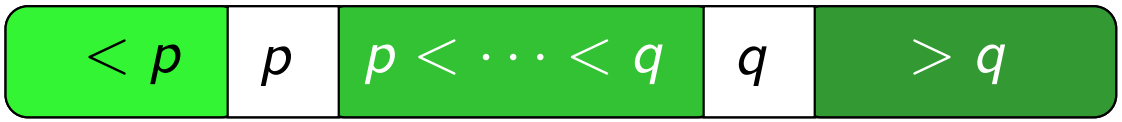
Dual-Pivot Quicksort



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.

Expected number of comparisons: **??**

Focus on comparisons and expectations.

Dual-Pivot Quicksort: Some History

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (Thesis, 1975)**: Analyzed a dual-pivot algorithm, found that it makes many more "**swaps**" than classical quicksort on average $\rightarrow$ no further investigation.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (Thesis, 1975)**: Analyzed a dual-pivot algorithm, found that it makes many more "**swaps**" than classical quicksort on average $\rightarrow$ no further investigation.
- **P. Hennequin (Thesis, 1991)**: Thorough analysis of quicksort with $k \geq 1$ pivots.

  For $k = 2$, **no improvements** over $2n \ln n$ found.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (Thesis, 1975)**: Analyzed a dual-pivot algorithm, found that it makes many more "**swaps**" than classical quicksort on average $\rightarrow$ no further investigation.
- **P. Hennequin (Thesis, 1991)**: Thorough analysis of quicksort with $k \geq 1$ pivots.

  For $k = 2$, **no improvements** over $2n \ln n$ found.

Topic went to sleep.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (Thesis, 1975)**: Analyzed a dual-pivot algorithm, found that it makes many more "**swaps**" than classical quicksort on average $\rightarrow$ no further investigation.
- **P. Hennequin (Thesis, 1991)**: Thorough analysis of quicksort with $k \geq 1$ pivots.

  For $k = 2$, **no improvements** over $2n \ln n$ found.

Topic went to sleep.

Java 7 (2009): Classical quicksort is replaced by a dual-pivot quicksort variant, proposed by Yaroslavskiy.

Experiments:
Yaroslavskiy's algorithm around 10% faster than classical QS.

Dual-Pivot Quicksort: Some History

Dual-Pivot Quicksort: Some History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding average comparison count:

- Yaroslavskiy: $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Dual-Pivot Quicksort: Some History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding average comparison count:

- Yaroslavskiy: $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Questions:

- Why different?
- Other possibilities?
- Best possible?

Next

- Model that captures all dual-pivot algorithms
- Unified analysis of the average comparison count
- An optimal algorithm

Next

- Model that captures all dual-pivot algorithms
- Unified analysis of the average comparison count
- An optimal algorithm

Assumptions:

- input is random permutation of $\{1, \ldots, n\}$
- left-most and right-most elements are the two pivots
- cost of sorting: average comparison count

Reduce Sorting Cost to Partitioning Cost

Lemma (Martínez/Nebel/Wild 2014)

Average partitioning cost of $\mathrm{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$ leads to average sorting cost $\mathrm{E}(C_n) = \frac{6}{5} a \cdot n \ln n + O(n)$.

Reduce Sorting Cost to Partitioning Cost

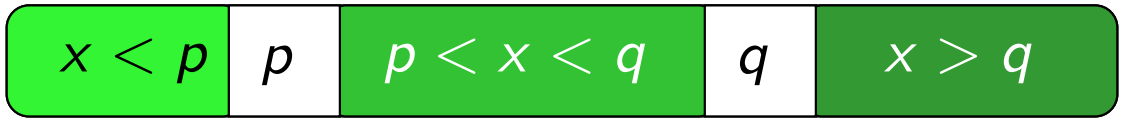
Lemma (Martínez/Nebel/Wild 2014)

Average partitioning cost of $\mathrm{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$ leads to average sorting cost $\mathrm{E}(C_n) = \frac{6}{5} a \cdot n \ln n + O(n)$.

So: What is the linear term in $\mathrm{E}(P_n)$?

Determining the Partitioning Cost

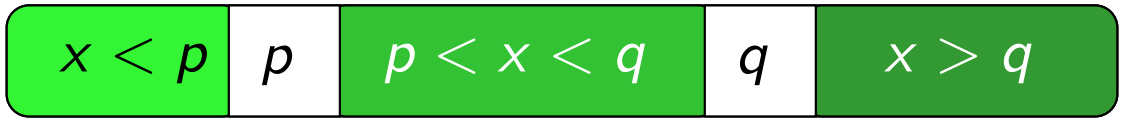
Must **classify** $n-2$ entries x into three parts:



small / σ, medium / μ, or large / λ

Determining the Partitioning Cost

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



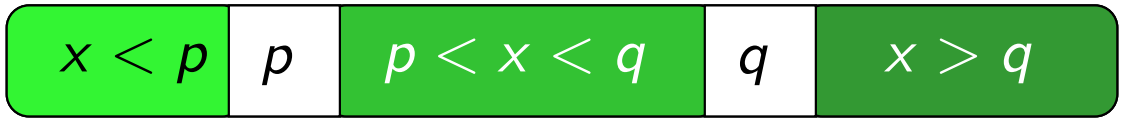
small / σ, medium / μ, or large / λ

1 or 2 comparisons.

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

Determining the Partitioning Cost

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



small / σ, medium / μ, or large / λ

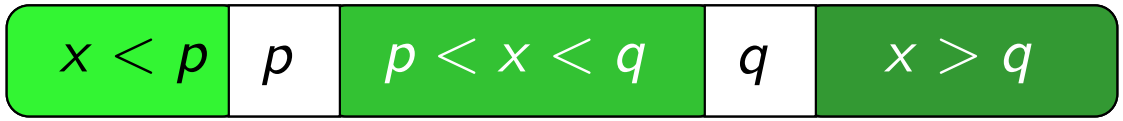
1 or 2 comparisons.

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

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

Determining the Partitioning Cost

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



small / σ, medium / μ, or large / λ

1 or 2 comparisons.

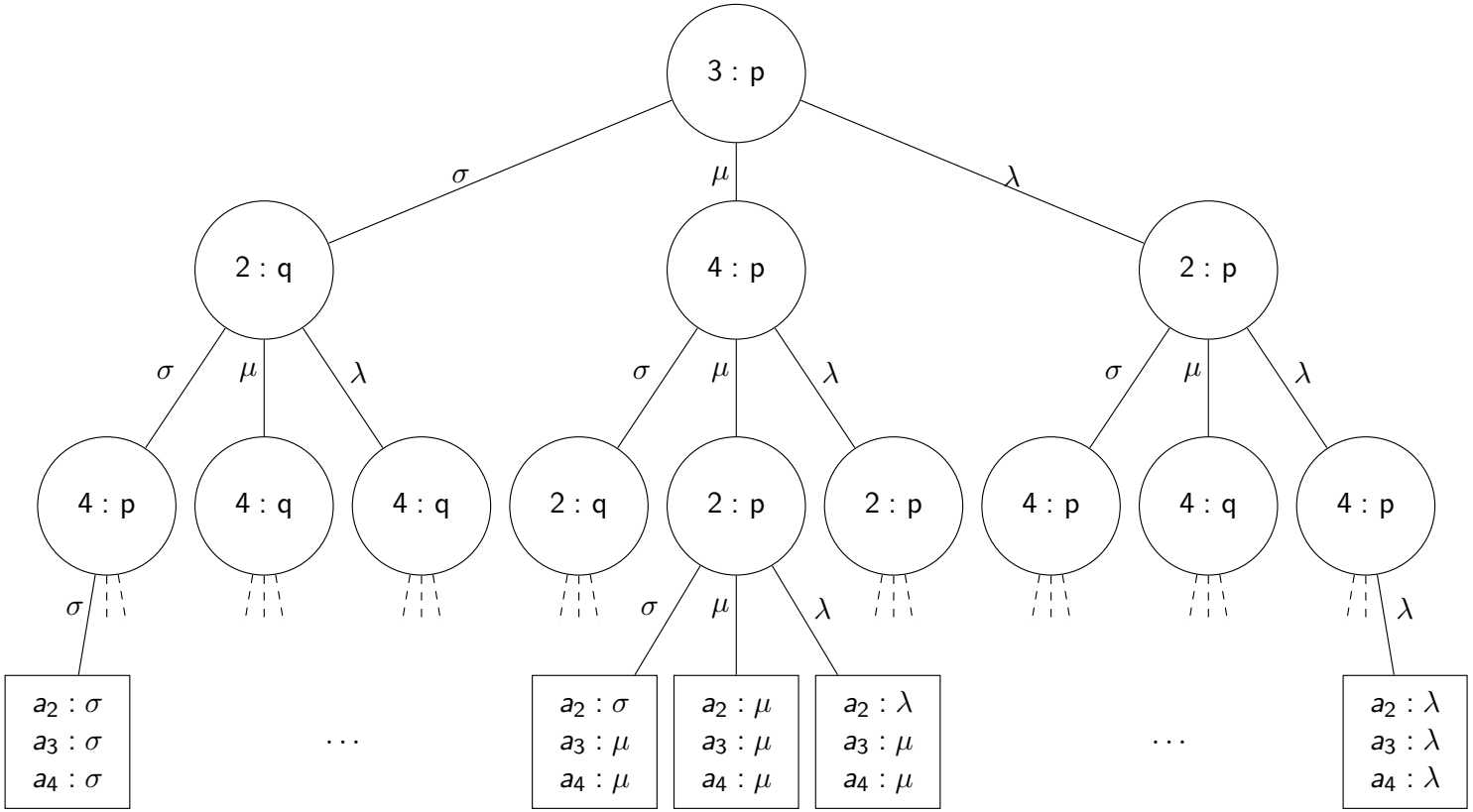
Unavoidable: 1 comparison for small/large x, 2 cmps. for medium x.

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.

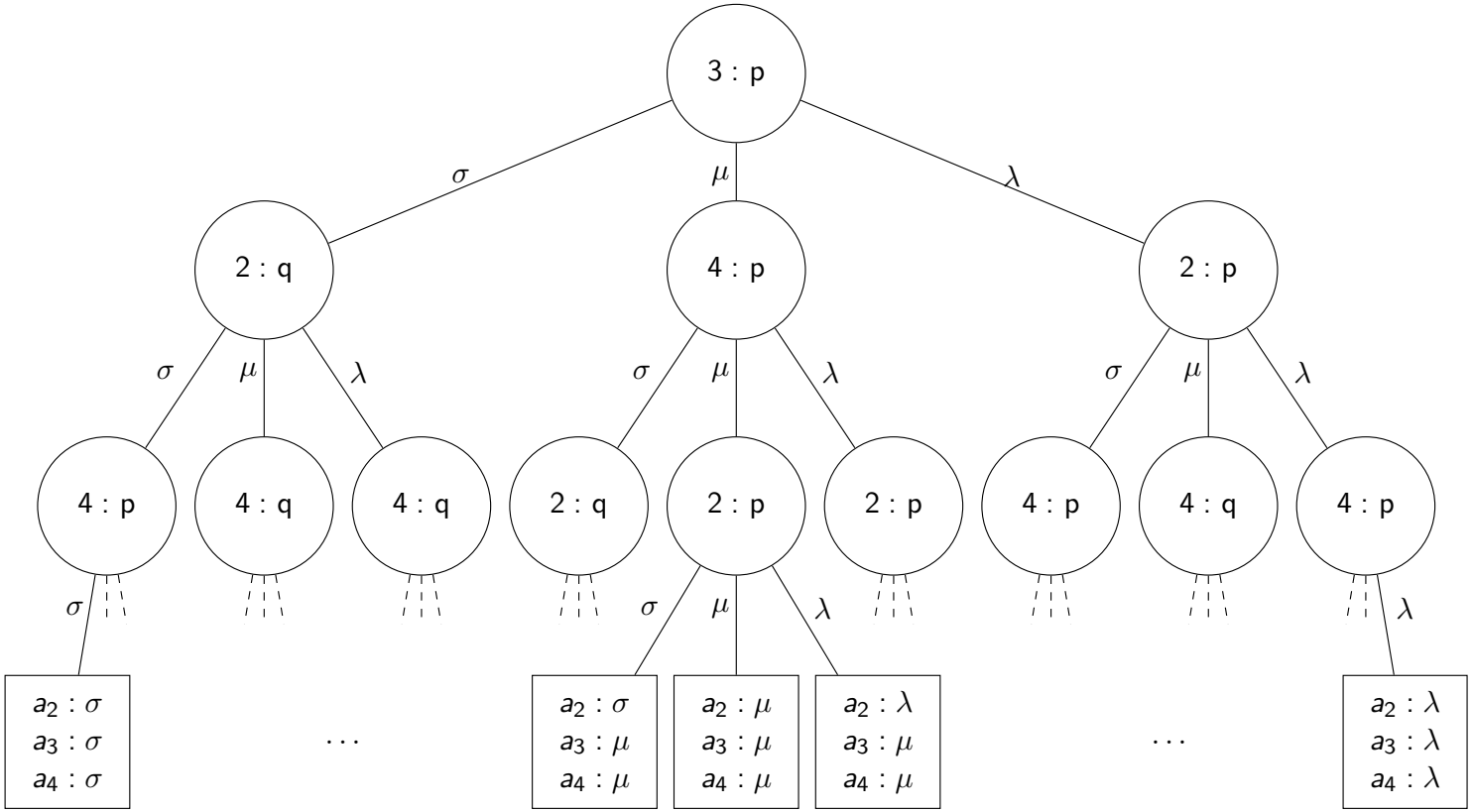
An implementation (Y./S./...) implicitly defines such a strategy.

Classification Tree, Models Strategy



Example: 3 4 2 1 5

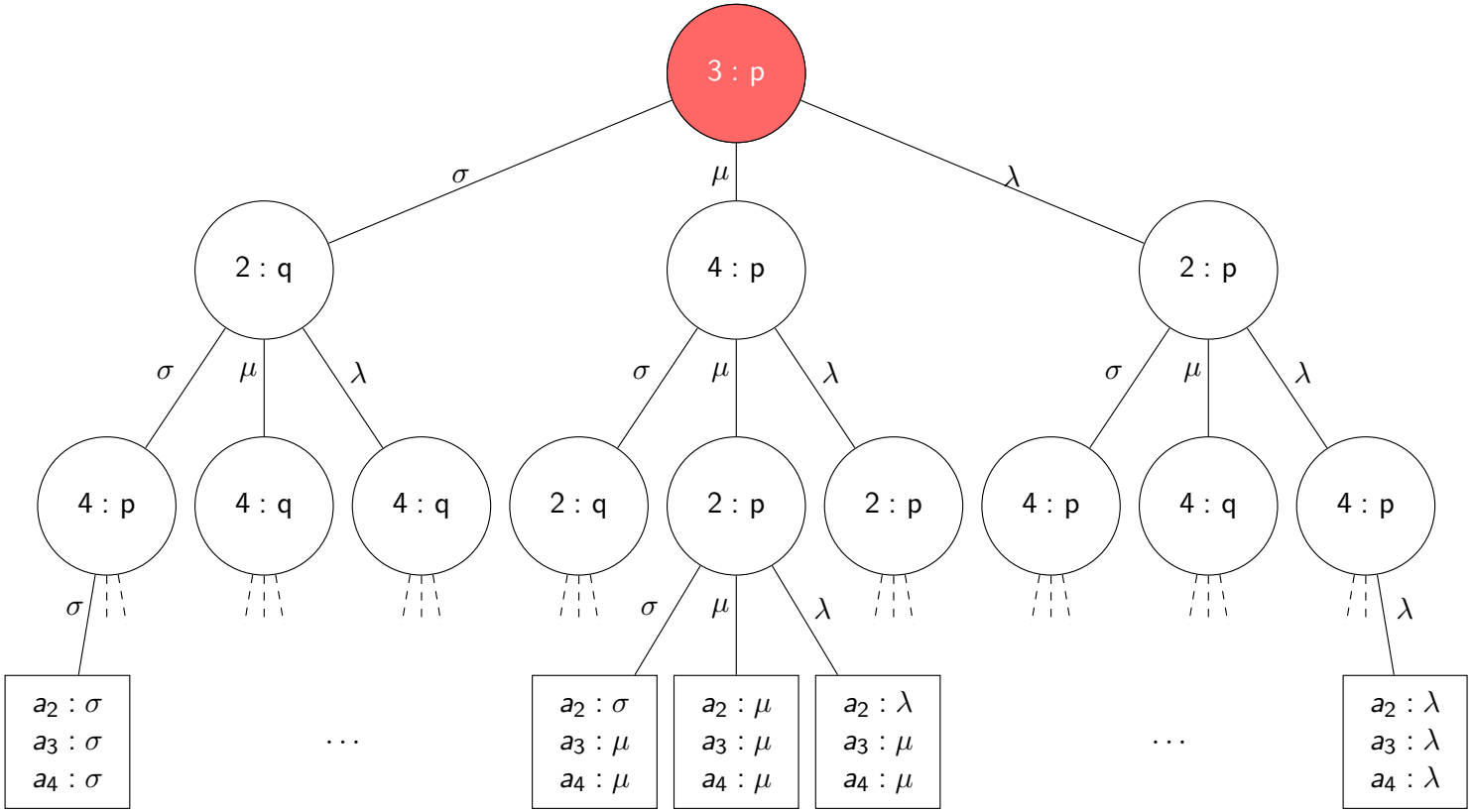
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p q

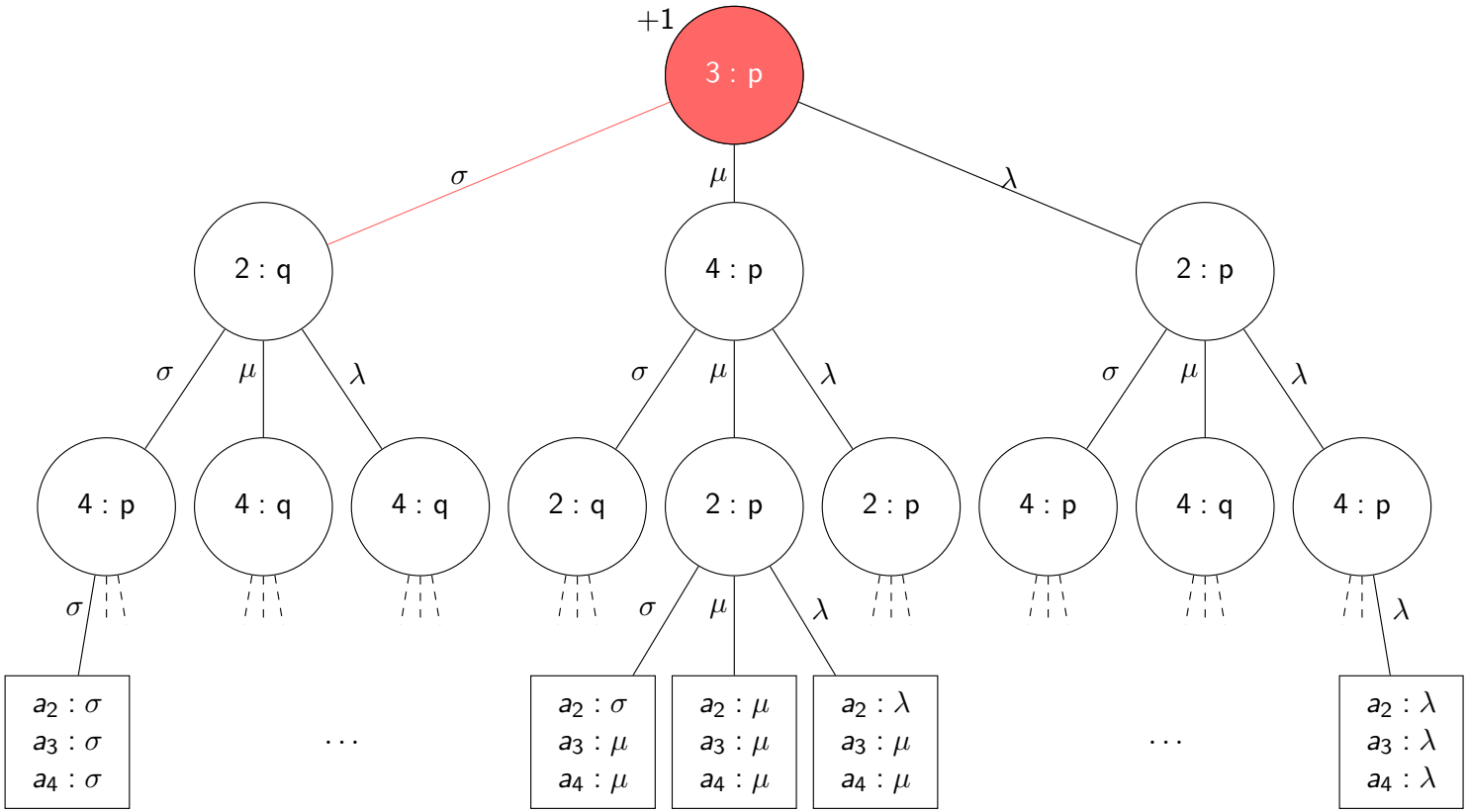
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p q

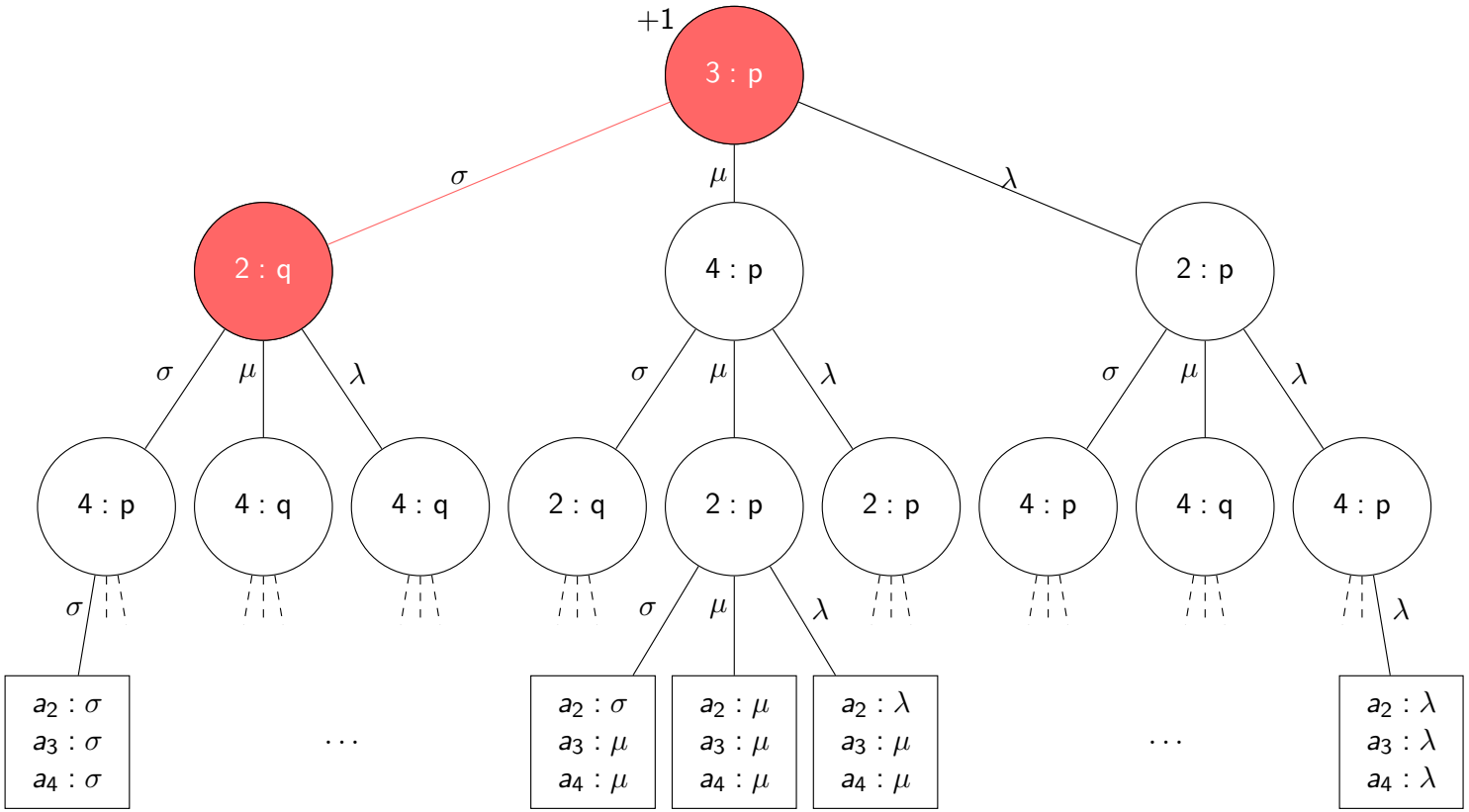
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p σ q

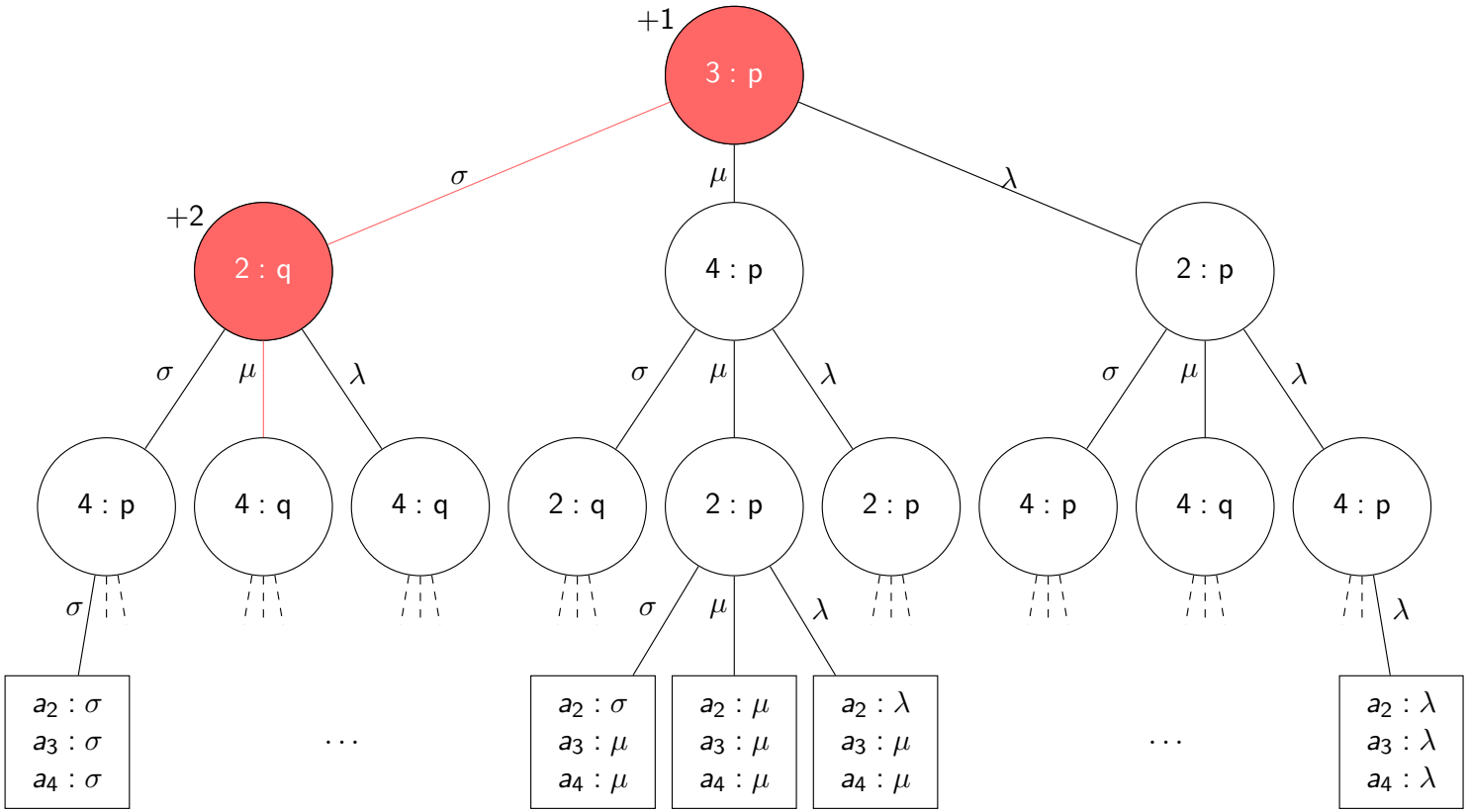
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p σ q

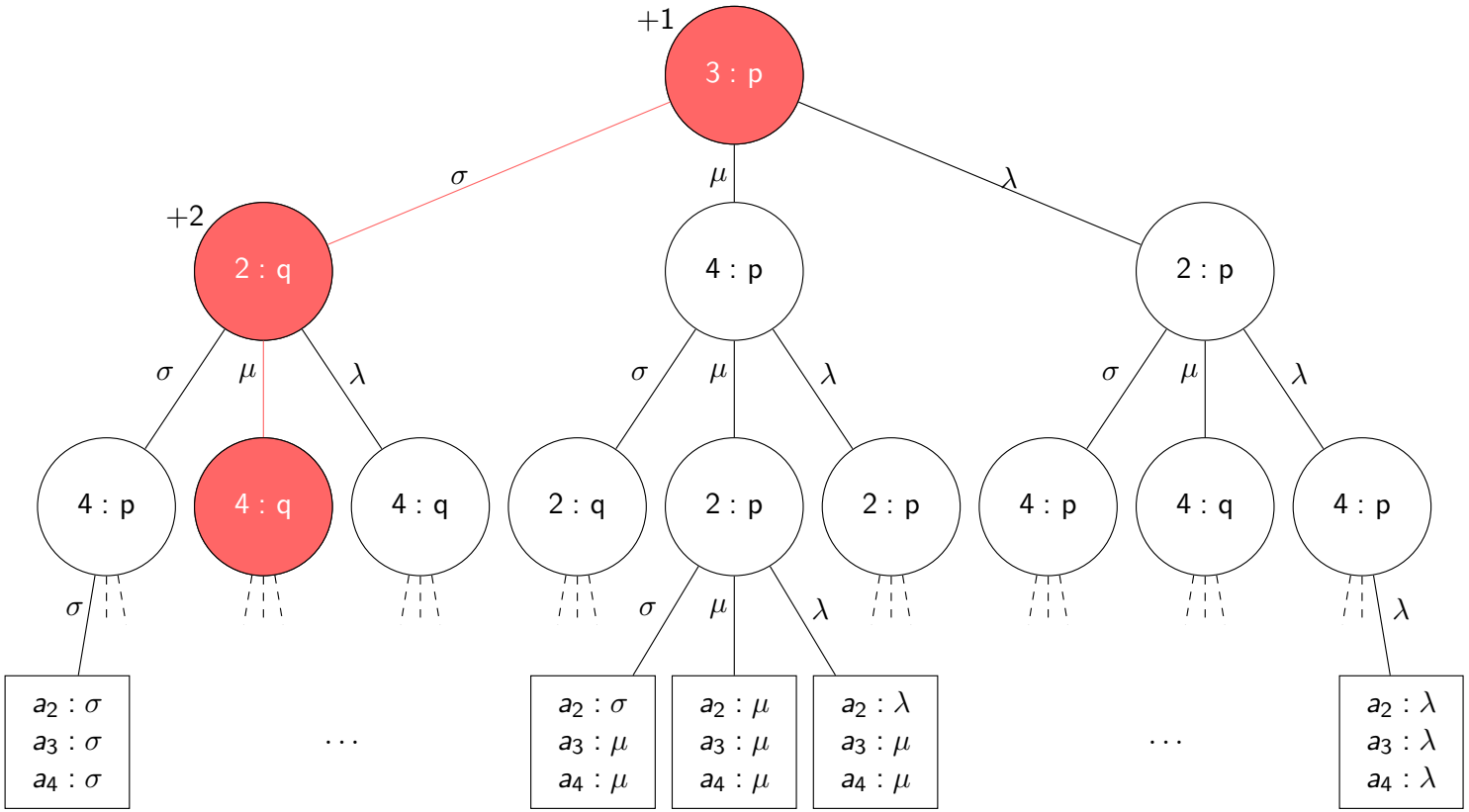
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p μ σ q

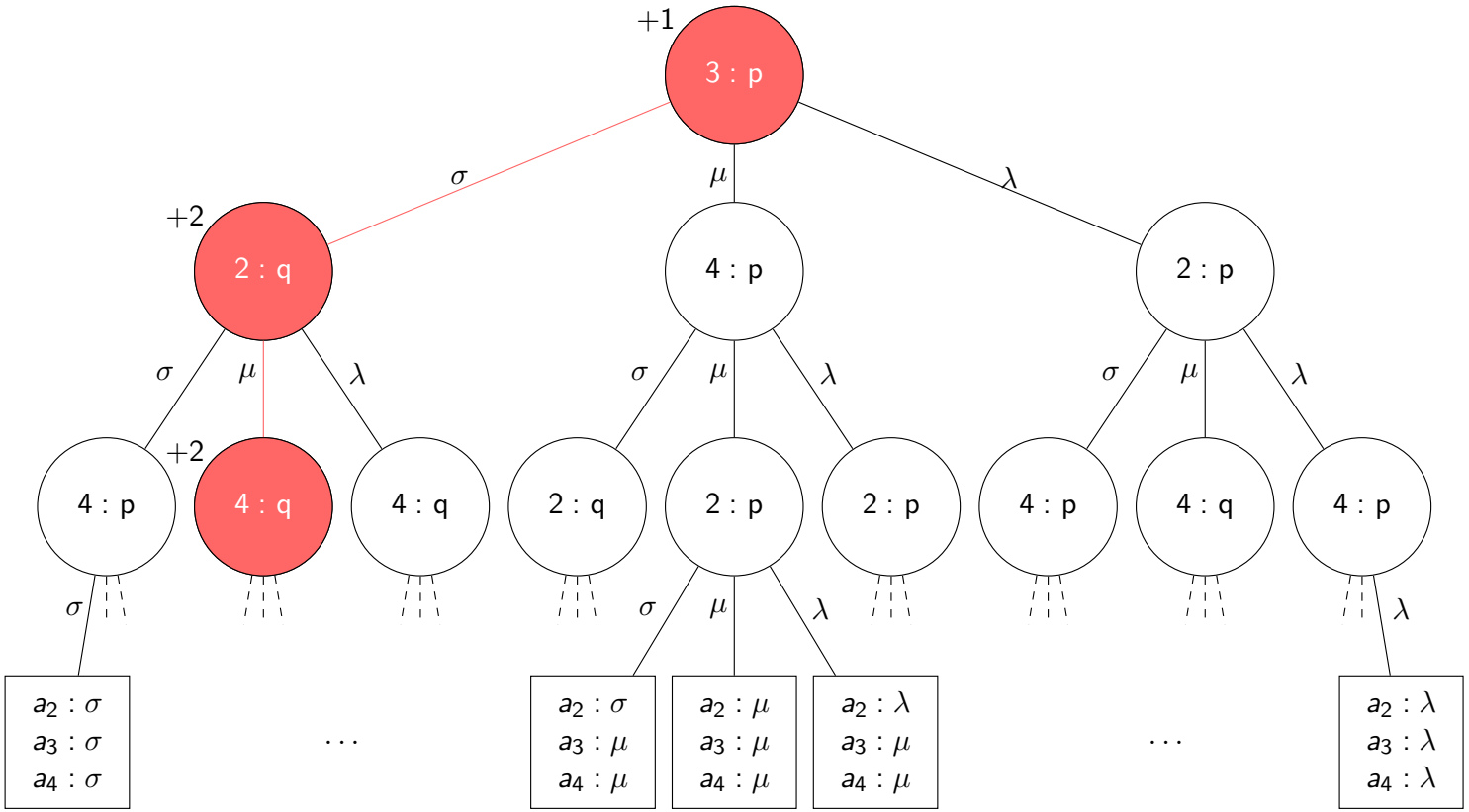
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p μ σ q

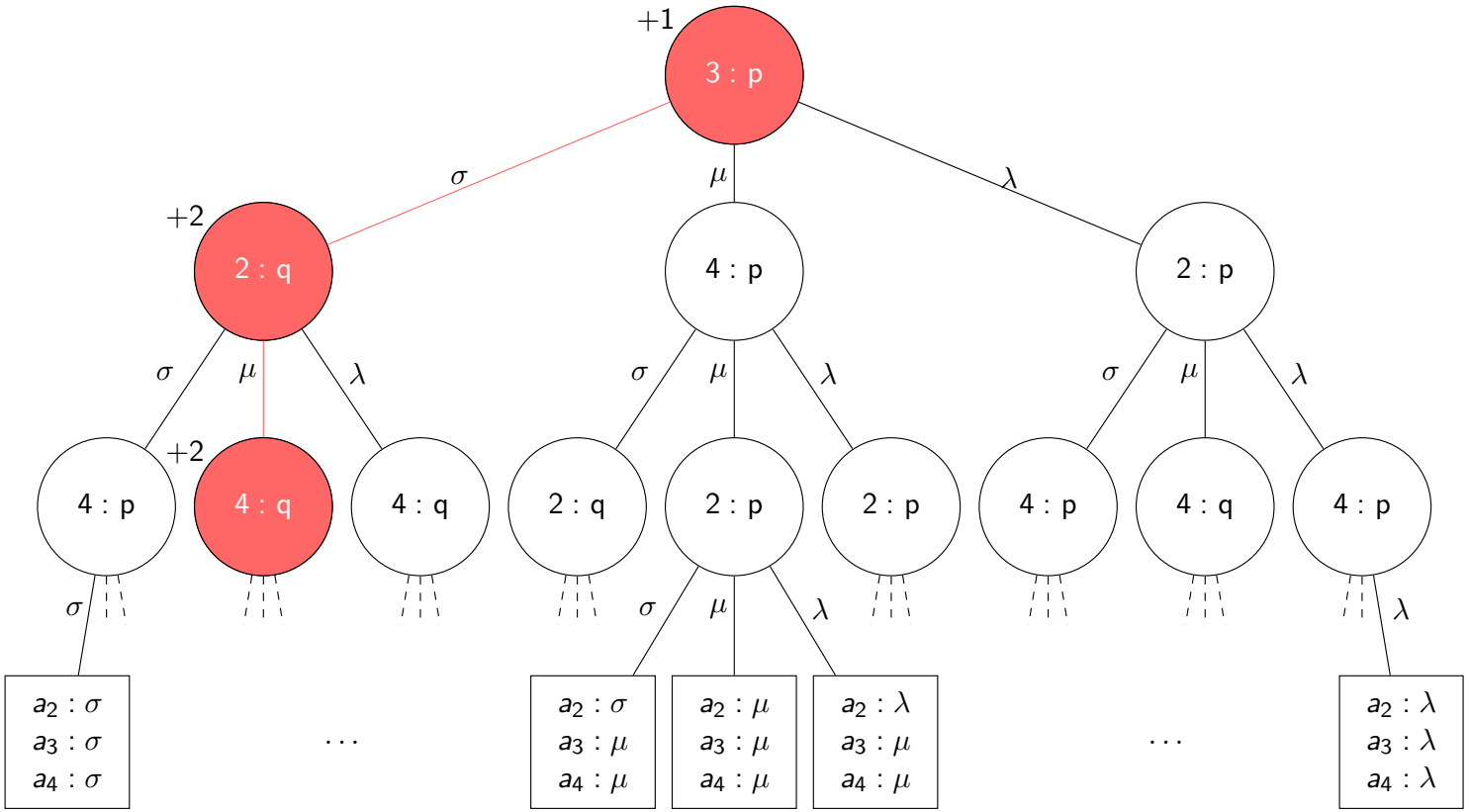
Classification Tree, Models Strategy



Example: 3 4 2 1 5

p μ σ σ q

Classification Tree, Models Strategy



Example: 3 4 2 1 5

$\Rightarrow$ 5 comparisons.

p μ σ σ q

Average Cost

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

$$\mathrm{E}(P_n) = 4/3n + \text{“average number of extra comparisons”}$$

Unavoidable comparisons.

“Extra” comparisons:

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

Cost of an Arbitrary Classification Tree

For fixed s, ℓ, let $f^{\mathrm{q}}_{s,\ell}$ be the average number of classifications "q first"

Lemma

For the average partition cost p_n of a classification tree T we have:

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{q}_{s,\ell} \cdot s + (n-2-f^{q}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Cost of an Arbitrary Classification Tree

For fixed s, ℓ, let $f^{q}_{s,\ell}$ be the average number of classifications "q first"

Lemma

For the average partition cost p_n of a classification tree T we have:

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{q}_{s,\ell} \cdot s + (n-2-f^{q}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Cost of an Arbitrary Classification Tree

For fixed s, ℓ, let $f^{\mathrm{q}}_{s,\ell}$ be the average number of classifications "q first"

Lemma

For the average partition cost p_n of a classification tree T we have:

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{q}_{s,\ell} \cdot s + (n-2-f^{q}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Cost of an Arbitrary Classification Tree

For fixed s, ℓ, let $f^{\mathrm{q}}_{s,\ell}$ be the average number of classifications "q first"

Lemma

For the average partition cost p_n of a classification tree T we have:

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{q}_{s,\ell} \cdot s + (n-2-f^{q}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

$\Rightarrow$ dependence introduces an error term of $O(n^{1-\varepsilon})$.

An Optimal Strategy Using An Oracle

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{\mathrm{q}}_{s,\ell} \cdot s + (n-2-f^{\mathrm{q}}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

An Optimal Strategy Using An Oracle

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{\mathrm{q}}_{s,\ell} \cdot s + (n-2-f^{\mathrm{q}}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Assume: Given the input, an oracle tells us whether or not $\ell > s$:

An Optimal Strategy Using An Oracle

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{\mathrm{q}}_{s,\ell} \cdot s + (n-2-f^{\mathrm{q}}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Assume: Given the input, an oracle tells us whether or not $\ell > s$:

Strategy

- $\ell > s$: Compare all elements to larger pivot first $(f^{\mathrm{q}}_{s,\ell} = n-2)$.

An Optimal Strategy Using An Oracle

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{\mathrm{q}}_{s,\ell} \cdot s + (n-2-f^{\mathrm{q}}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Assume: Given the input, an oracle tells us whether or not $\ell > s$:

Strategy

- $\ell > s$: Compare all elements to larger pivot first $(f^{\mathrm{q}}_{s,\ell} = n-2)$.
- $\ell \le s$: Compare all elements to smaller pivot first $(f^{\mathrm{q}}_{s,\ell} = 0)$.

An Optimal Strategy Using An Oracle

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \le n-2} \left(f^{\mathrm{q}}_{s,\ell} \cdot s + (n-2-f^{\mathrm{q}}_{s,\ell}) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Assume: Given the input, an oracle tells us whether or not $\ell > s$:

Strategy

- $\ell > s$: Compare all elements to larger pivot first $(f^{\mathrm{q}}_{s,\ell} = n-2)$.
- $\ell \le s$: Compare all elements to smaller pivot first $(f^{\mathrm{q}}_{s,\ell} = 0)$.

Average sorting cost: $1.8 n \ln n + O(n)$.

Remarks on Oracle Strategy

Idealized, but gives lower bound:

> No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons.

Implementation?

Random Sampling (read $n^{3/4}$ entries) to estimate if $s > \ell$ or $s \leq \ell$.

Remarks on Oracle Strategy

Idealized, but gives lower bound:

> No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons.

Implementation?

Random Sampling (read $n^{3/4}$ entries) to estimate if $s > \ell$ or $s \leq \ell$.

Average cost of random sampling algorithm: $1.8n \ln n + O(n)$.

Remarks on Oracle Strategy

Idealized, but gives lower bound:

No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons.

Implementation?

Random Sampling (read $n^{3/4}$ entries) to estimate if $s > \ell$ or $s \leq \ell$.

Average cost of random sampling algorithm: $1.8n \ln n + O(n)$.

In thesis: The **optimal** strategy, with an implementation.

So far...

- Unified model for dual-pivot quicksort algorithms.
- How to calculate average cost for general algorithms.
- Identified an optimal dual-pivot quicksort algorithm.
  Optimum: $1.8n \ln n$. (QS: $2n \ln n$, Yaroslavskiy: $1.9n \ln n$.)

So far...

- Unified model for dual-pivot quicksort algorithms.
- How to calculate average cost for general algorithms.
- Identified an optimal dual-pivot quicksort algorithm.
  Optimum: $1.8n \ln n$. (QS: $2n \ln n$, Yaroslavskiy: $1.9n \ln n$.)

In thesis:

- generalization to $k \geq 3$ pivots.
- optimal bounds: formula for all $k \geq 3$.
- solution for $k = 3$: $\approx 1.705n \ln n$ (Master's thesis, P. Klaue, 2014)
- open: solution for $k > 3$.

So far...

- Unified model for dual-pivot quicksort algorithms.
- How to calculate average cost for general algorithms.
- Identified an optimal dual-pivot quicksort algorithm.
  Optimum: $1.8n \ln n$. (QS: $2n \ln n$, Yaroslavskiy: $1.9n \ln n$.)

In thesis:

- generalization to $k \geq 3$ pivots.
- optimal bounds: formula for all $k \geq 3$.
- solution for $k = 3$: $\approx 1.705n \ln n$ (Master's thesis, P. Klaue, 2014)
- open: solution for $k > 3$.

Next: Other cost measures.

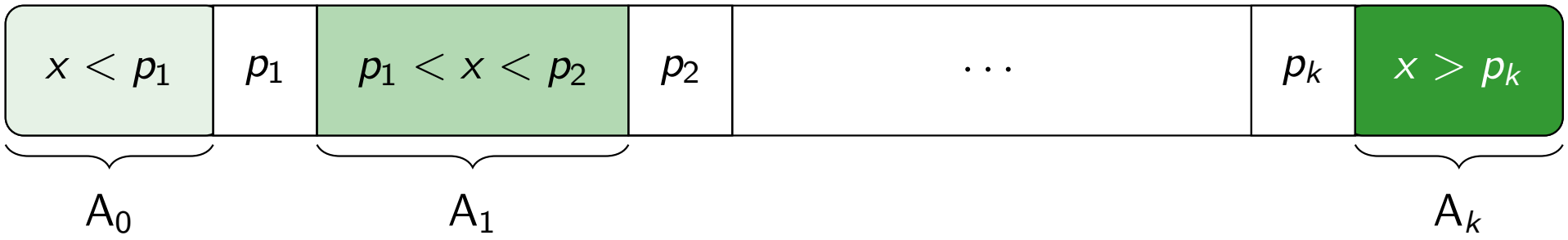
Other Cost Measures

- 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.

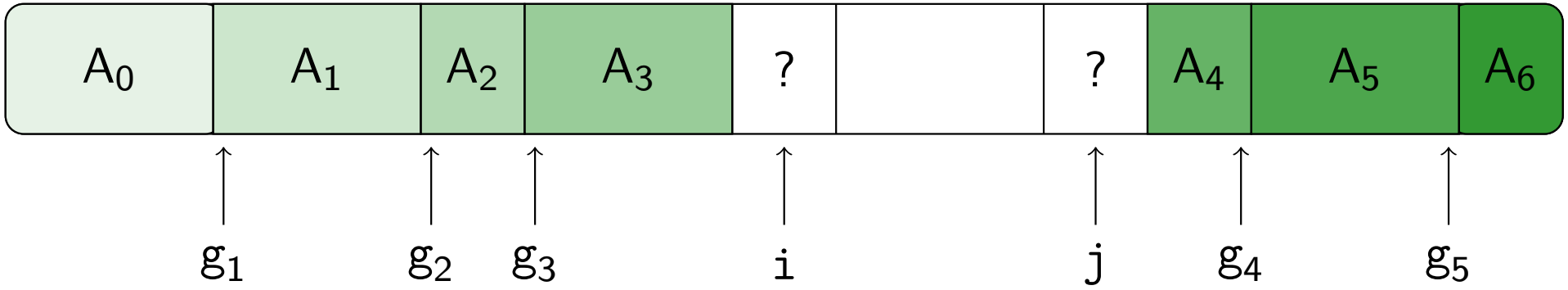
Quicksort with k Pivots

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

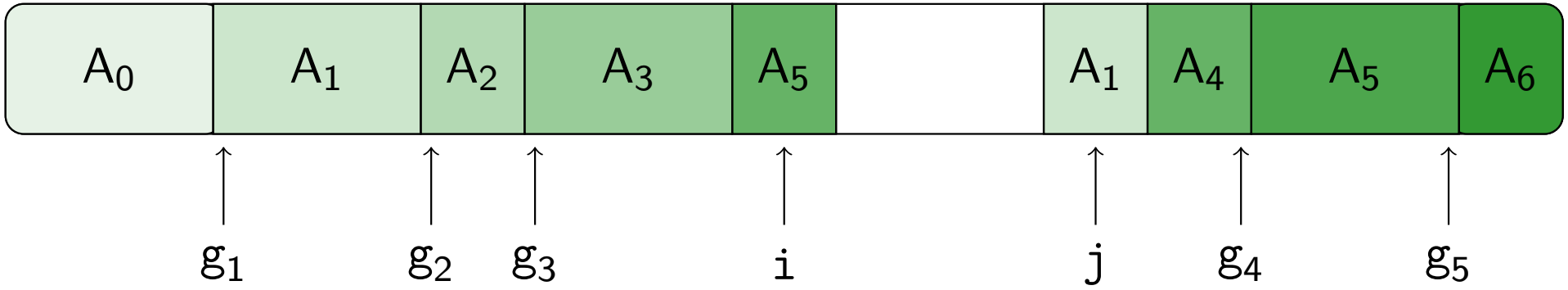


Then sort classes recursively.

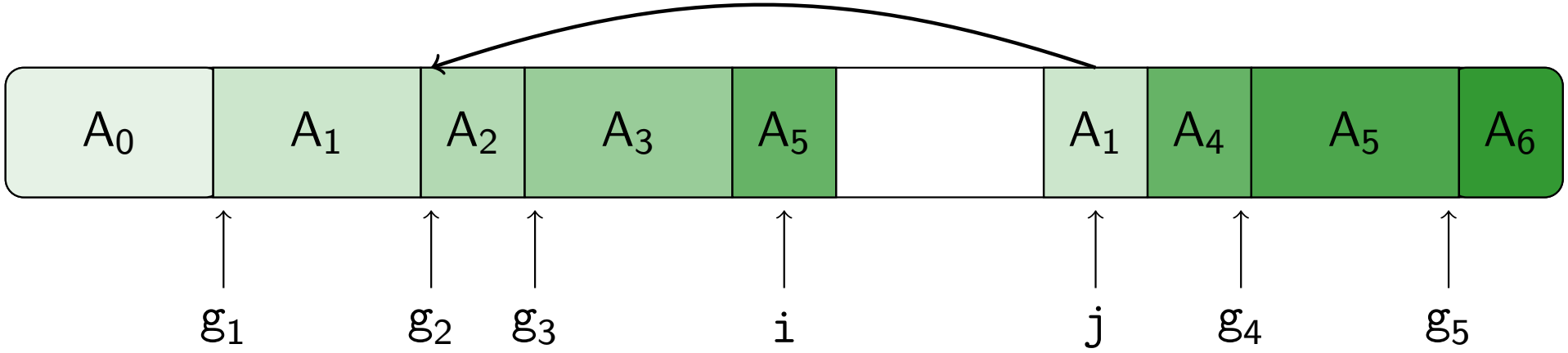
A Generalized Partitioning Algorithm



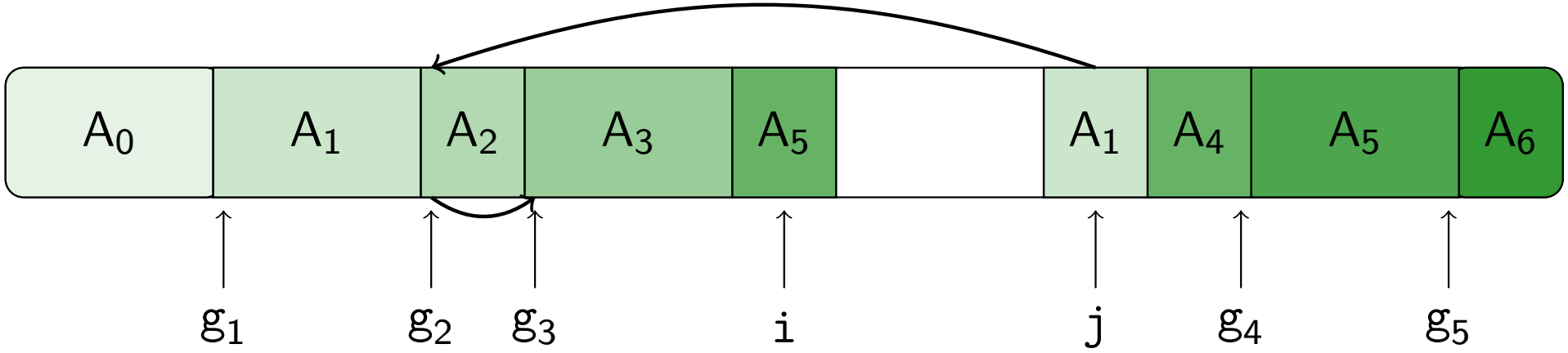
A Generalized Partitioning Algorithm



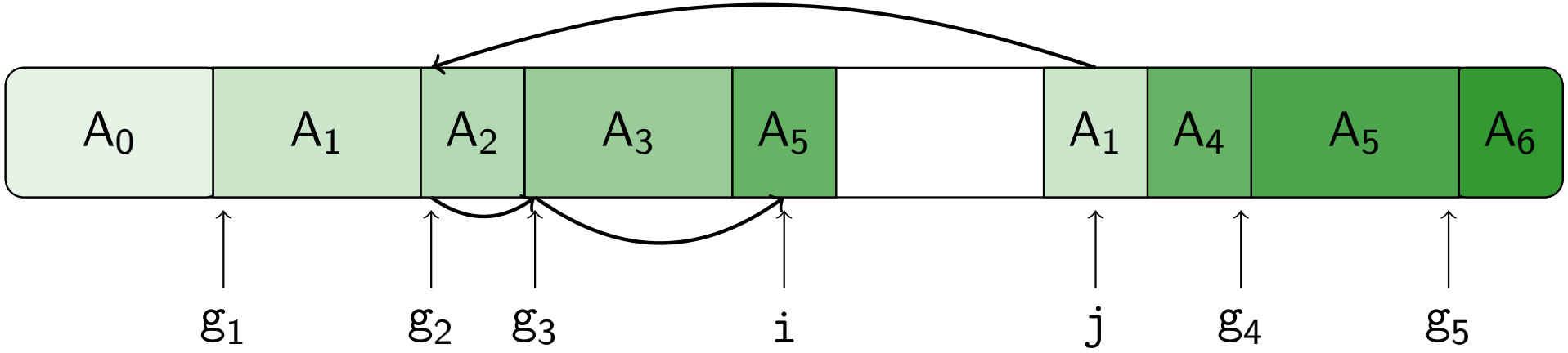
A Generalized Partitioning Algorithm



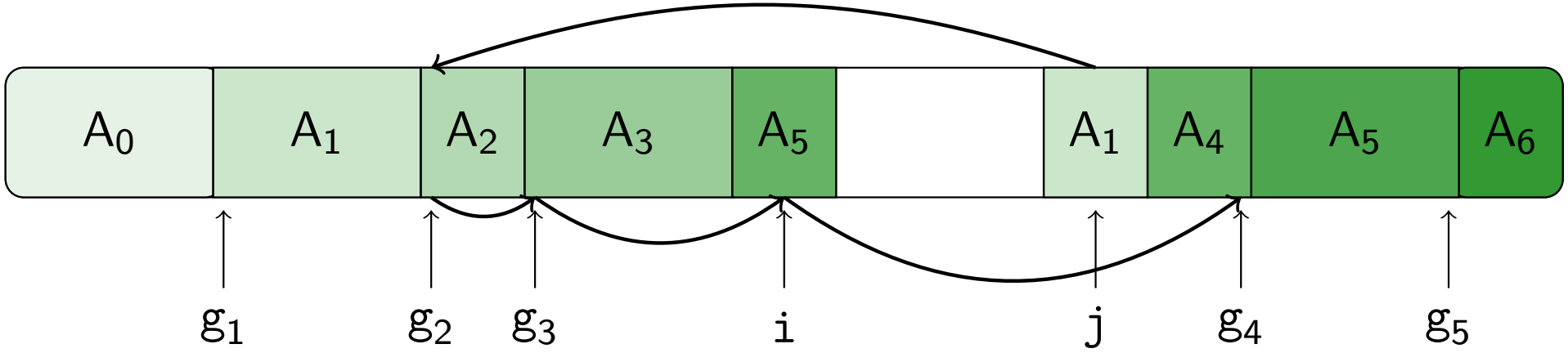
A Generalized Partitioning Algorithm



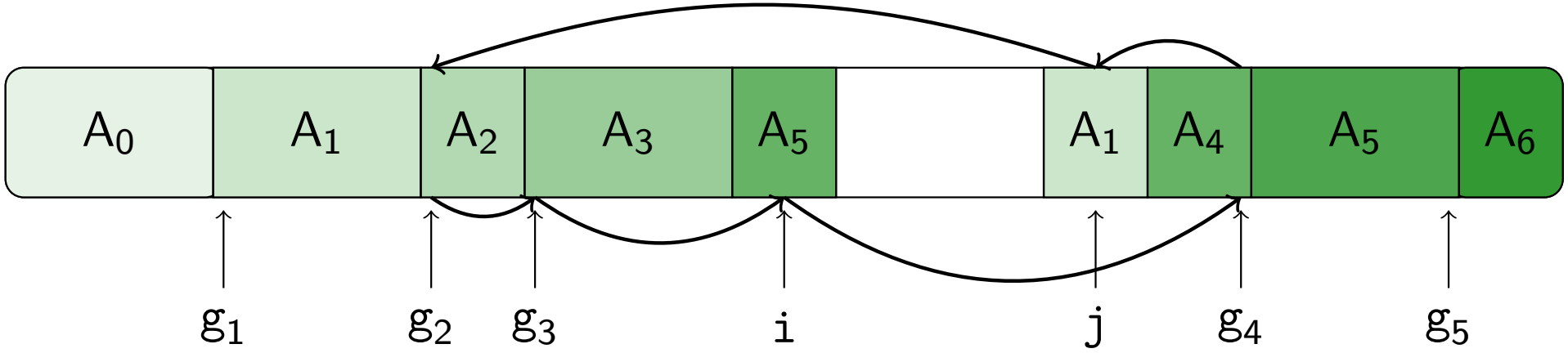
A Generalized Partitioning Algorithm



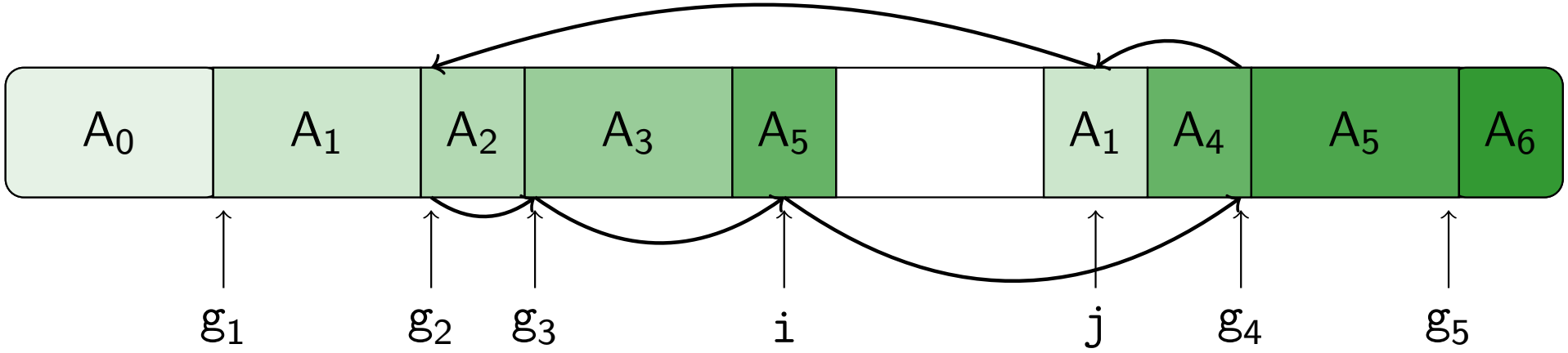
A Generalized Partitioning Algorithm



A Generalized Partitioning Algorithm

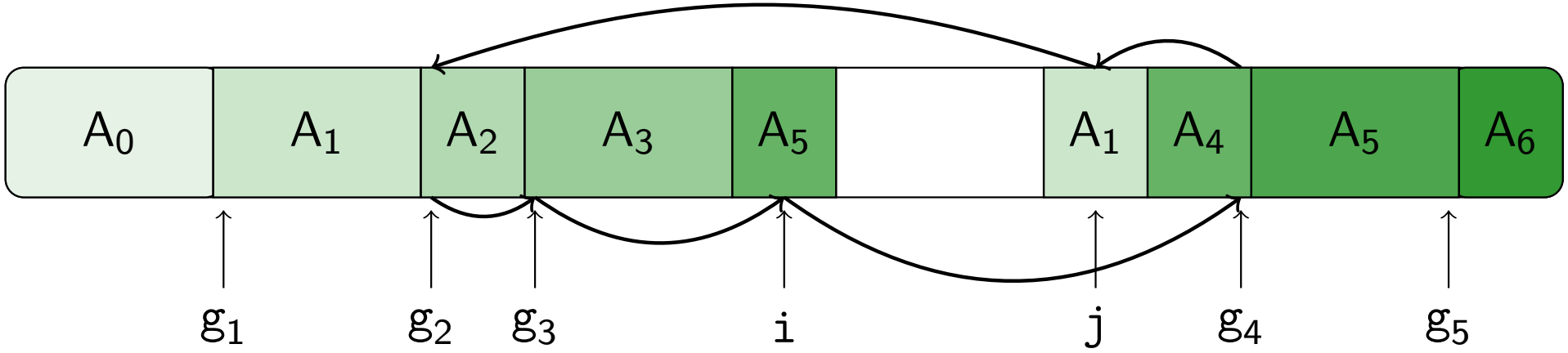


A Generalized Partitioning Algorithm



5 pointer visits.

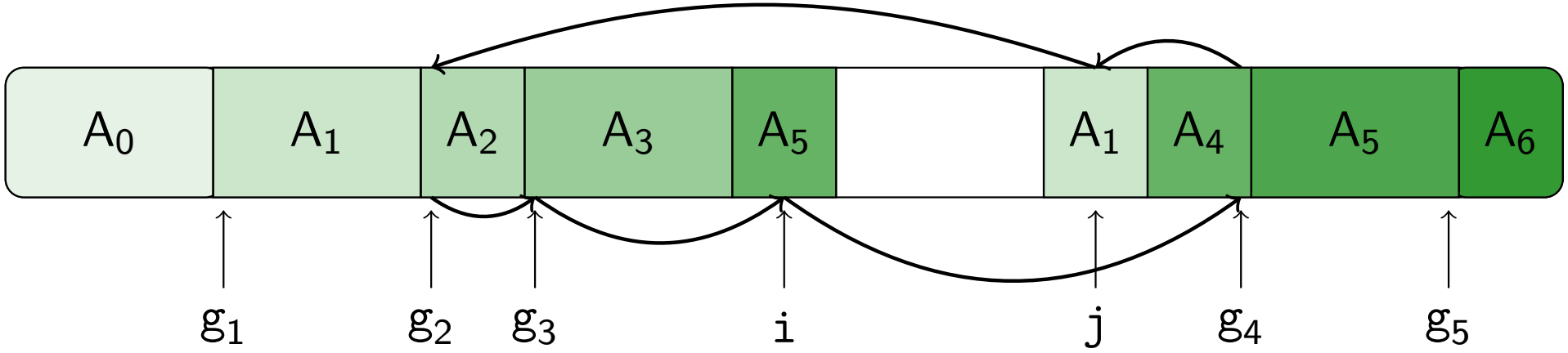
A Generalized Partitioning Algorithm



5 pointer visits.

Analysis for partitioning yields:

A Generalized Partitioning Algorithm



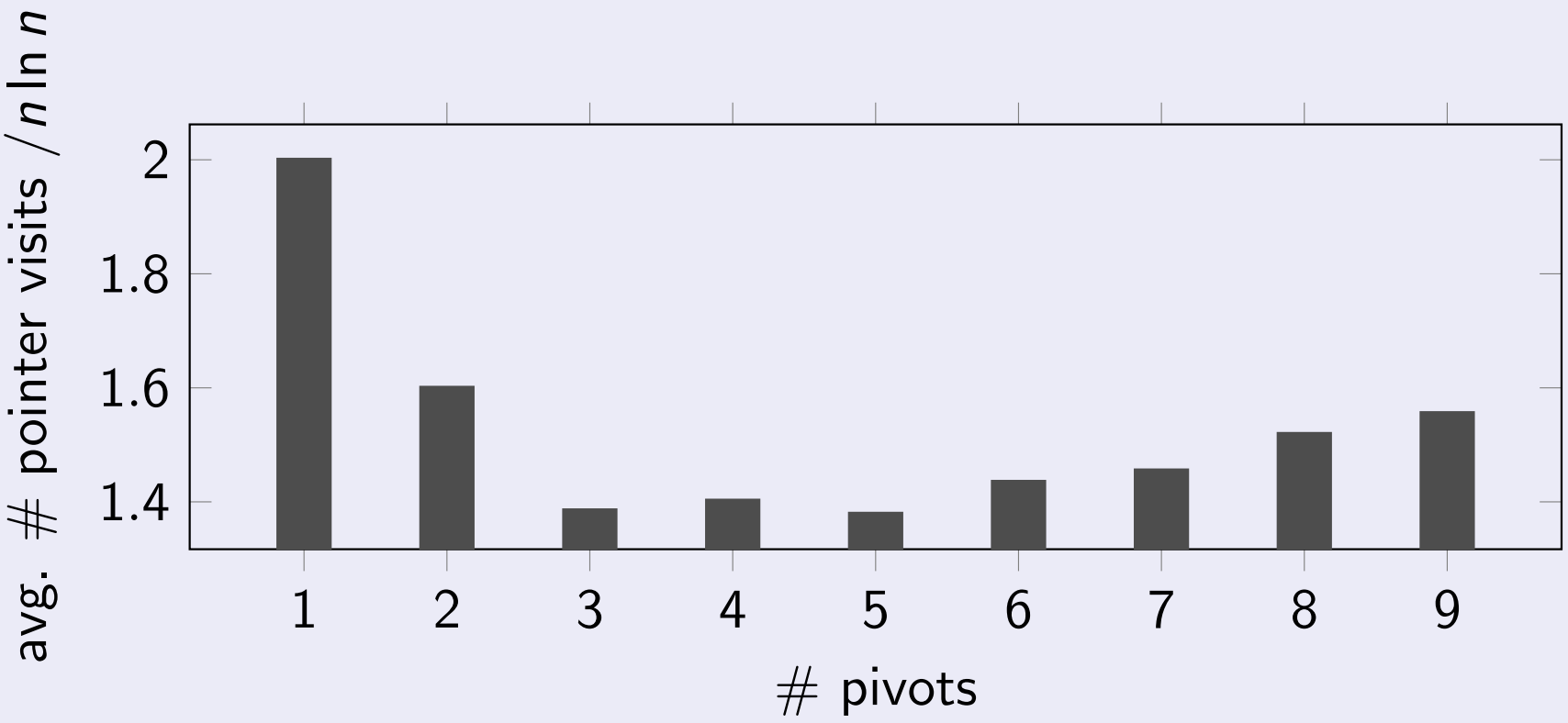
5 pointer visits.

Analysis for partitioning yields:

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

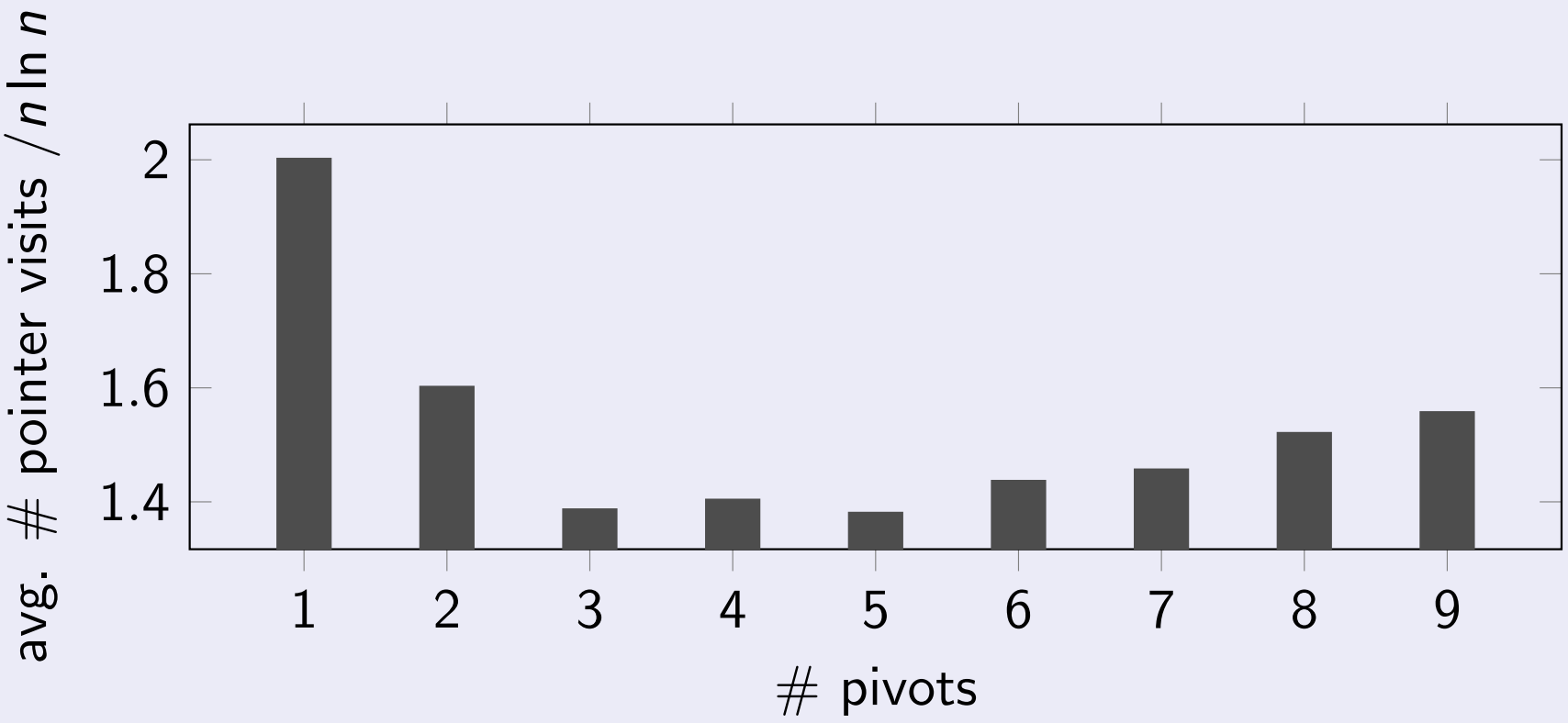
A Generalized Partitioning Algorithm

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



A Generalized Partitioning Algorithm

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



- In thesis: Thorough experimental study
- Result 1: "pointer visits" predict L1 cache misses accurately
- Result 2: closest w.r.t. predicting empirical running time

Summary and Open Problems

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- An optimal classification strategy
- $1.8n \ln n + O(n)$ is optimal
- $k > 2$ pivots: Mentioned other cost measures that might determine running time

Further Directions:

- Choosing pivots from a sample
- Optimal algorithms w.r.t. other cost measures
- Dual-Pivot Quicksort with equal elements
- Smoothed Analysis for Multi-Pivot Quicksort

Part II

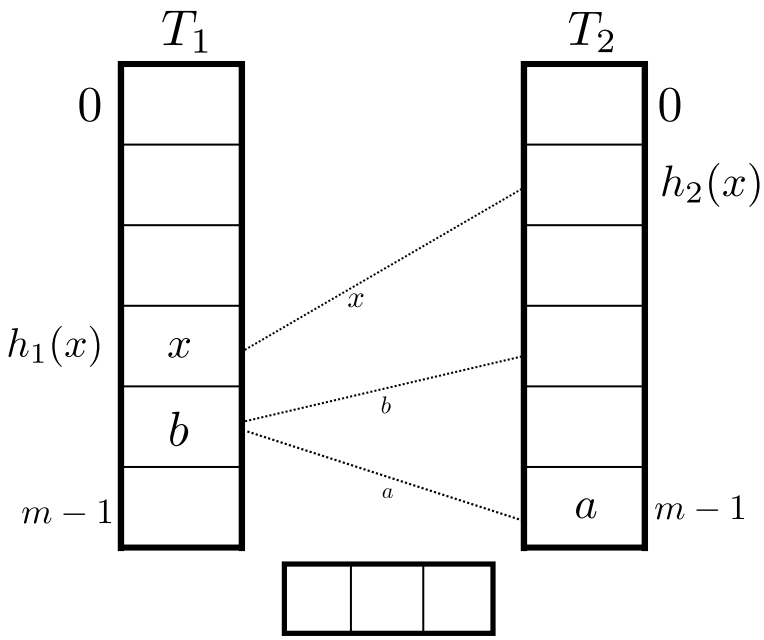
Efficient Hash Functions

Ex.: Cuckoo Hashing with a Stash (PR '01/KMW '08)

A hashing-based implementation of the **dictionary** data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- $s = O(1)$ additional cells ("stash")
- two (hash) functions h_1, h_2 with $h_i : U \to [m]$

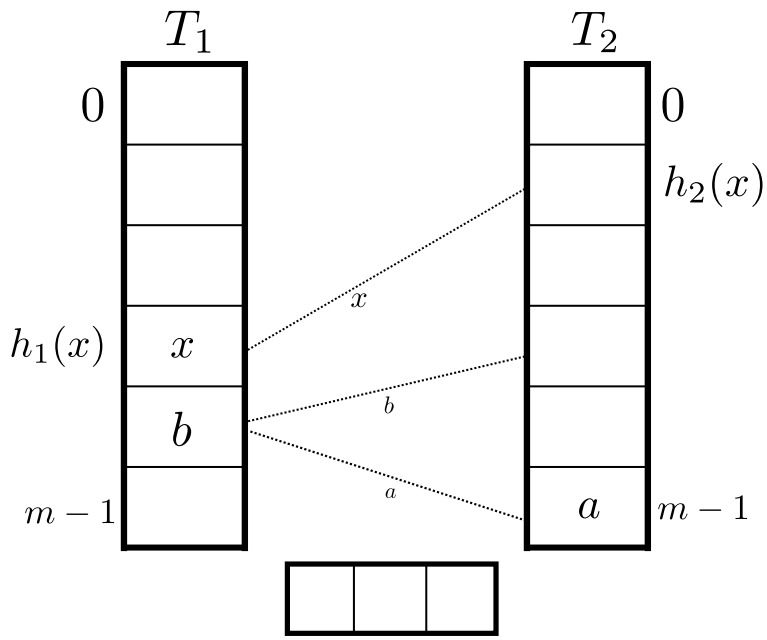


Ex.: Cuckoo Hashing with a Stash (PR '01/KMW '08)

A hashing-based implementation of the **dictionary** data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- $s = O(1)$ additional cells ("stash")
- two (hash) functions h_1, h_2 with $h_i : U \to [m]$



Rules:

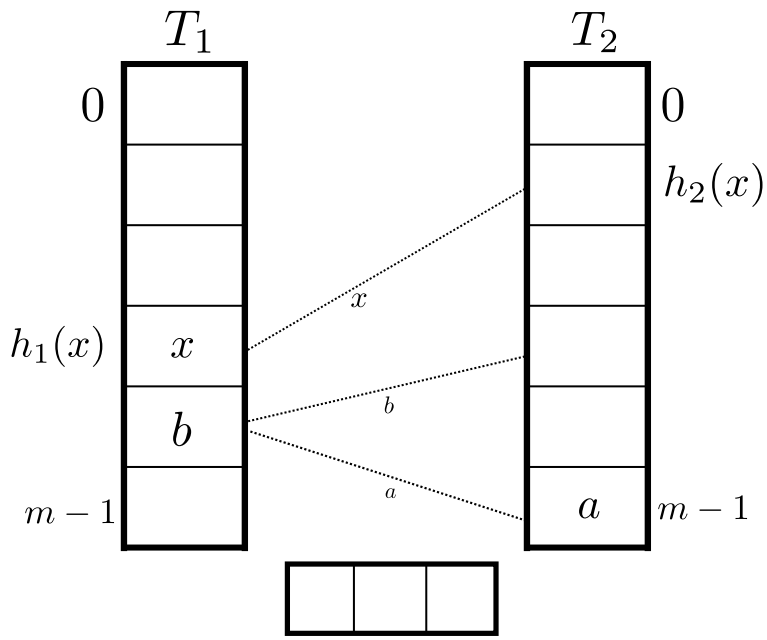
- each table/stash cell can hold exactly one key
- a key x must be stored either in $T_1[h_1(x)]$, $T_2[h_2(x)]$, or in the stash

Ex.: Cuckoo Hashing with a Stash (PR '01/KMW '08)

A hashing-based implementation of the **dictionary** data type.

Setting:

- set $S \subseteq U$ of n keys
- two tables $T_1[0..m-1]$ and $T_2[0..m-1]$, $m \geq (1+\varepsilon)n$
- $s = O(1)$ additional cells ("stash")
- two (hash) functions h_1, h_2 with $h_i : U \to [m]$



Rules:

- each table/stash cell can hold exactly one key
- a key x must be stored either in $T_1[h_1(x)]$, $T_2[h_2(x)]$, or in the stash

Definition

If S can be stored according to these rules, we call (h_1, h_2) **suitable for** S.

Failure Probability of C.H. with a Stash

Theorem (KMW '08)

Let $S \subseteq U$ with $|S| = n$. If (h_1, h_2) are **fully random**, then

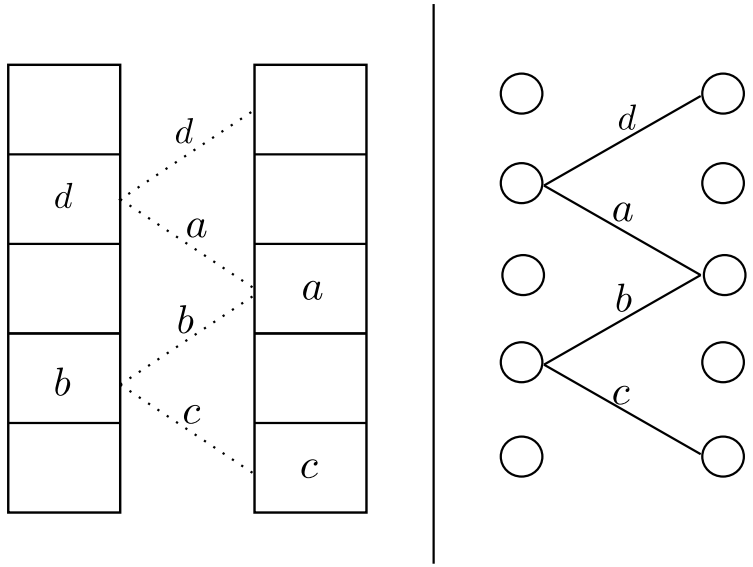
$$\Pr((h_1, h_2) \text{ unsuitable for } S \text{ with stash size } s) = O(1/n^{s+1}).$$

Tight result: $\Theta(1/n^{s+1})$. (Kutzelnigg '10)

Analysis of Cuckoo Hashing with a Stash

What is a criteria for (h_1, h_2) being unsuitable for stash size s?

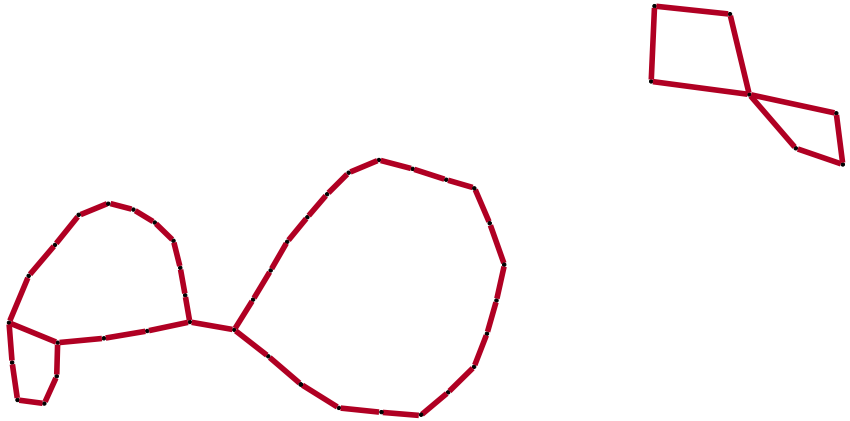
Tool: Cuckoo graph $G(S, h_1, h_2)$ (Devroye/Morin '03)



Analysis of Cuckoo Hashing with a Stash

What is a criteria for (h_1, h_2) being unsuitable for stash size s?

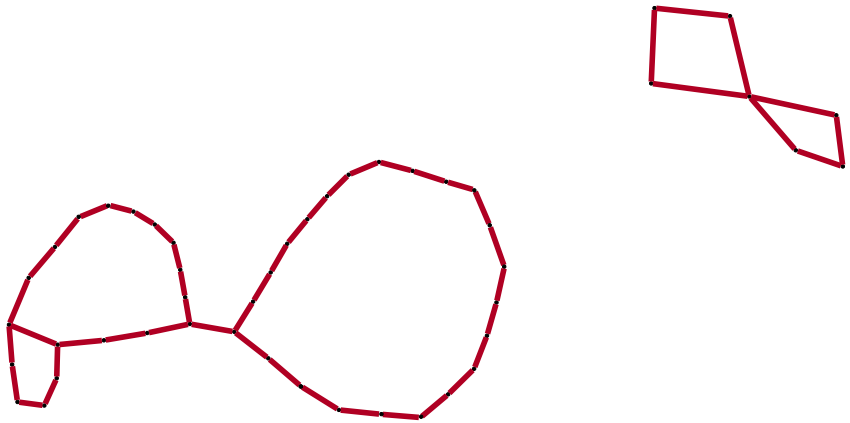
Tool: Cuckoo graph $G(S, h_1, h_2)$ (Devroye/Morin '03)



Analysis of Cuckoo Hashing with a Stash

What is a criteria for (h_1, h_2) being unsuitable for stash size s?

Tool: Cuckoo graph $G(S, h_1, h_2)$ (Devroye/Morin '03)



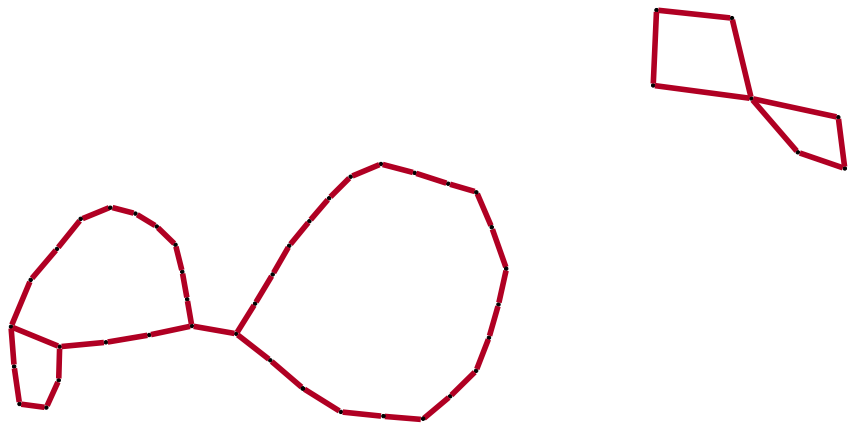
MOG_s: No leaves, each conn. comp. > 1 cycles, #edges - #vert.$=s+1$.

Example: MOG_2

Analysis of Cuckoo Hashing with a Stash

What is a criteria for (h_1, h_2) being unsuitable for stash size s?

Tool: Cuckoo graph $G(S, h_1, h_2)$ (Devroye/Morin '03)



MOG_s: No leaves, each conn. comp. > 1 cycles, #edges - #vert.$=s+1$.

Example: MOG_2

Lemma (A. '10/ ADW '12 (Journal: ADW '14))

(h_1, h_2) unsuitable for S with stash size $s \Leftrightarrow G(S, h_1, h_2)$ contains a MOS_s.

Next

Next

- An explicit and efficient hash function construction.
- General approach to bound probability that certain subgraphs exist.
- No focus on applications.

Next

- An explicit and efficient hash function construction.
- General approach to bound probability that certain subgraphs exist.
- No focus on applications.

Other approaches: Thorup/Pătraşcu'11, Reingold/Rothblum/Wieder'14.

Next

- An explicit and efficient hash function construction.
- General approach to bound probability that certain subgraphs exist.
- No focus on applications.

Other approaches: Thorup/Pătraşcu'11, Reingold/Rothblum/Wieder'14.

Based on manuscript of Woelfel'05

Next

- An explicit and efficient hash function construction.
- General approach to bound probability that certain subgraphs exist.
- No focus on applications.

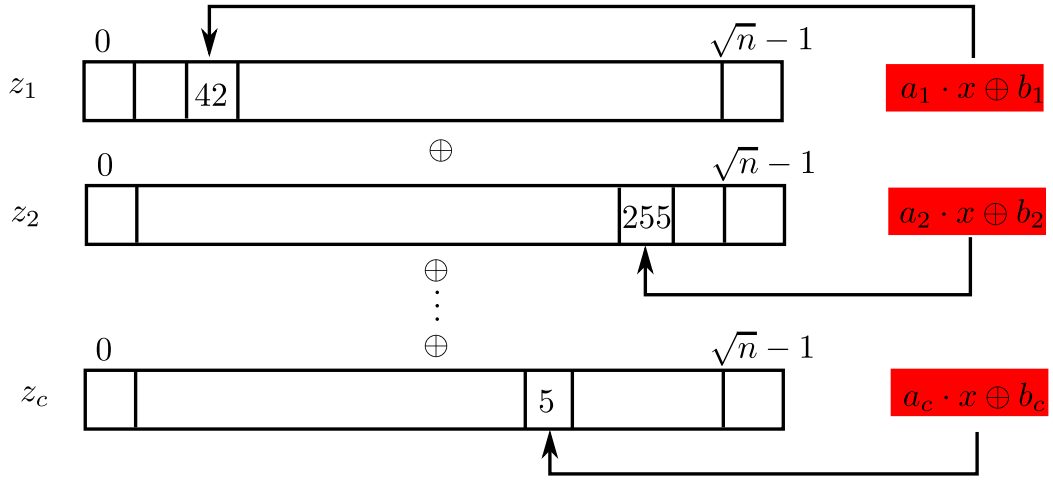
Other approaches: Thorup/Pătraşcu'11, Reingold/Rothblum/Wieder'14.

Based on manuscript of Woelfel'05

and Dietzfelbinger/Woelfel'03, Woelfel'06.

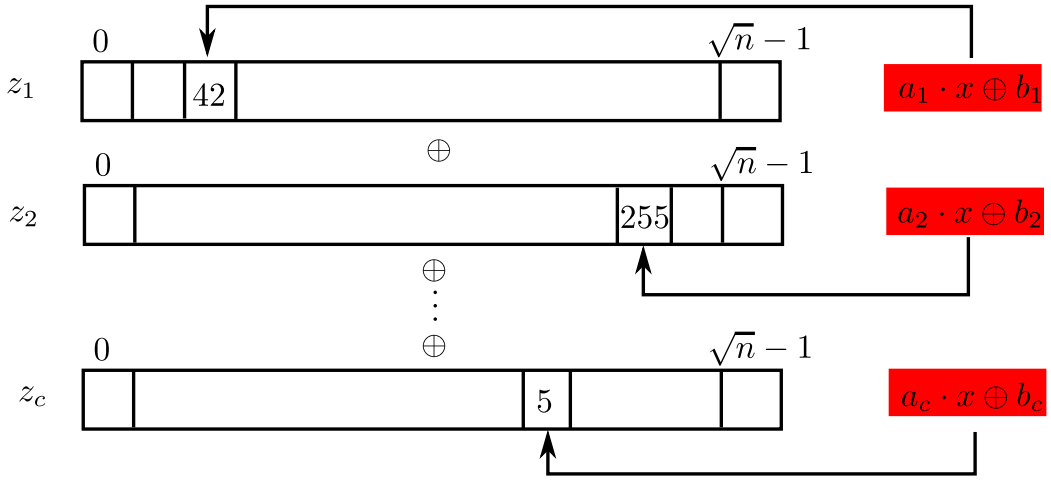
Sketch of The Hash Class (Modification of DW '03)

$$h(x) = a_0 \cdot x \oplus b_0 \bigoplus$$



Sketch of The Hash Class (Modification of DW '03)

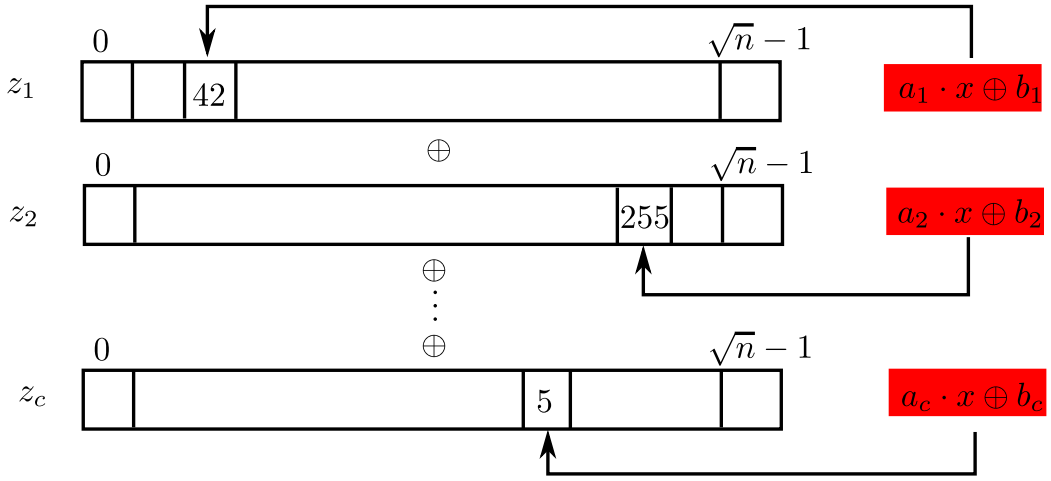
$$h(x) = a_0 \cdot x \oplus b_0 \bigoplus$$



$$h_i(x) = f_i(x) \oplus \bigoplus_{j=1}^{c} z_j^{(i)}[\, g_j(x) \,], \qquad i = 1, 2, \ldots$$

Sketch of The Hash Class (Modification of DW '03)

$$h(x) = a_0 \cdot x \oplus b_0 \bigoplus$$

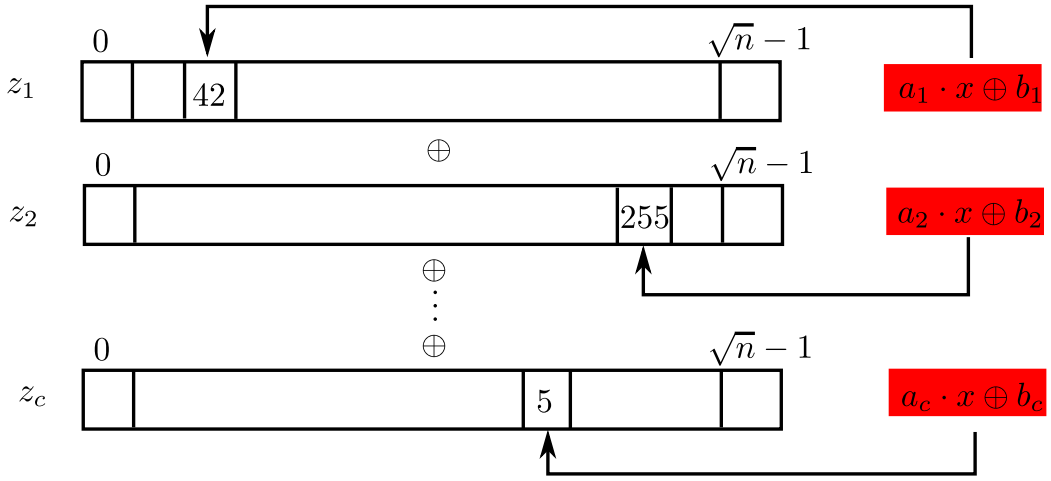


$$h_i(x) = f_i(x) \oplus \bigoplus_{j=1}^{c} z_j^{(i)}[\, g_j(x) \,], \qquad i = 1, 2, \ldots$$

Class of all these pairs (h_1, h_2) of hash functions: $\mathcal{Z}$.

Sketch of The Hash Class (Modification of DW '03)

$$h(x) = a_0 \cdot x \oplus b_0 \bigoplus$$



$$h_i(x) = f_i(x) \oplus \bigoplus_{j=1}^{c} z_j^{(i)}[\, g_j(x) \,], \qquad i = 1, 2, \ldots$$

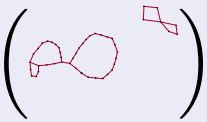
Class of all these pairs (h_1, h_2) of hash functions: $\mathcal{Z}$.

Let $T \subseteq U$. If there is a g_j function such that at most one pair of keys in T collides under g_j (i.e., $g_j(x) = g_j(y)$), then h_1, h_2 are fully random on T.

If this is the case: (h_1, h_2) **T-good**, otherwise (h_1, h_2) **T-bad**.

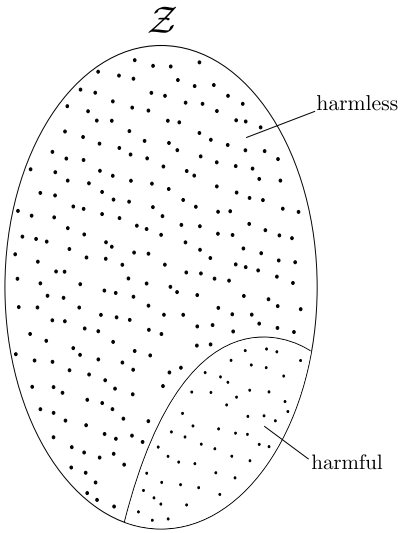
Approach / Collecting "Harmful" Hash Functions

Main Task

Let $\mathcal{A}$ be a graph property, e. g., MOG_s  . Obtain bounds on

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}(G(S,h_1,h_2) \text{ has subgraph with property } \mathcal{A}).$$

We split our set of hash functions into "harmful" and "harmless" ones.



(h_1, h_2) are harmful, if there exists $T \subseteq S$ such that

- $G(T, h_1, h_2)$ has property $\mathcal{A}$,
- (h_1, h_2) is T-bad.

$B_S^{\mathcal{A}}$:= set of all harmful pairs (h_1, h_2).

A Central Lemma

Lemma

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}(\exists \textit{ subgraph with property } \mathcal{A}) \leq$$
$$\mathrm{E}^*(\#\textit{subgraphs with property } \mathcal{A}) + \Pr_{(h_1,h_2)\in\mathcal{Z}}(B_S^{\mathcal{A}}).$$

First summand: May use full randomness.

A Central Lemma

Lemma

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}(\exists \textit{ subgraph with property } \mathcal{A}) \leq$$
$$\mathrm{E}^*(\#\textit{subgraphs with property } \mathcal{A}) + \Pr_{(h_1,h_2)\in\mathcal{Z}}(B_S^{\mathcal{A}}).$$

First summand: May use full randomness.

How to bound $\Pr(B_S^{\mathcal{A}})$?

$\Pr(B_S^{\mathcal{A}}) = \Pr(\exists T \subseteq S\colon G(T, h_1, h_2) \in \mathcal{A} \cap (h_1, h_2)$ are T-bad $)$

A Central Lemma

Lemma

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}(\exists \textit{ subgraph with property } \mathcal{A}) \leq$$
$$\mathrm{E}^*(\#\textit{subgraphs with property } \mathcal{A}) + \Pr_{(h_1,h_2)\in\mathcal{Z}}(B_S^{\mathcal{A}}).$$

First summand: May use full randomness.

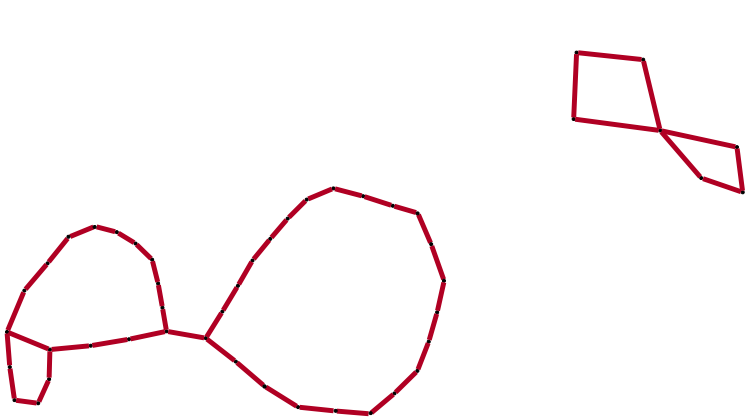
How to bound $\Pr(B_S^{\mathcal{A}})$?

$\Pr(B_S^{\mathcal{A}}) = \Pr(\exists T \subseteq S\colon G(T, h_1, h_2) \in \mathcal{A} \cap (h_1, h_2)$ are T-bad $)$

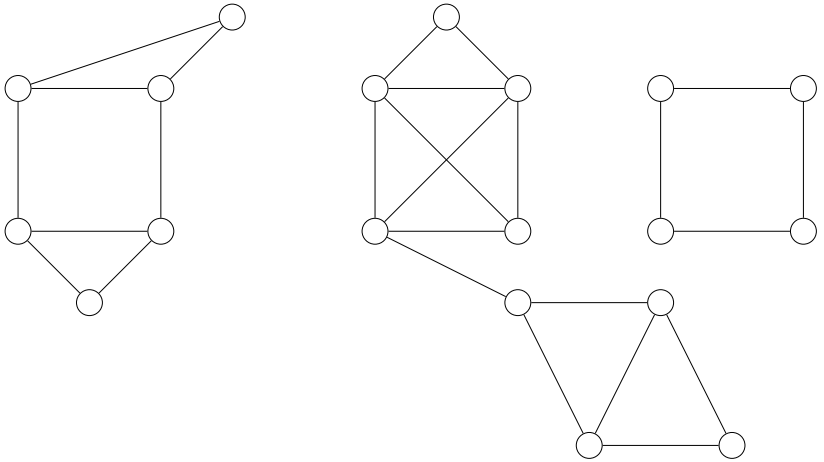
Approach: Generalize, Peel, Reduce.

First Step: Generalize $\mathcal{A}$

Goal: Find suitable graph property $\mathcal{B} \supseteq \mathcal{A}$.

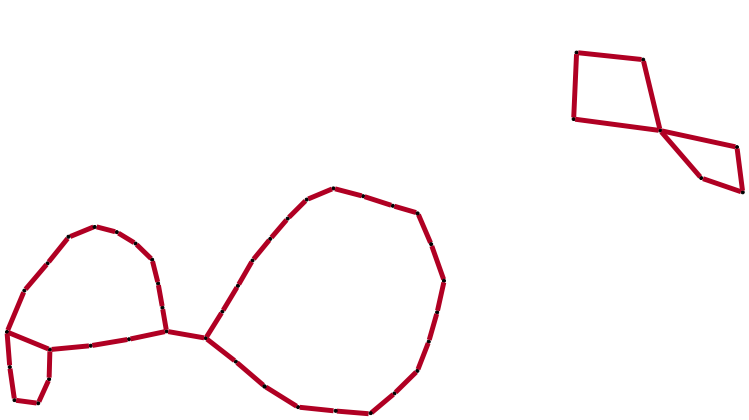


MOG_s $\subseteq$ "leafless"

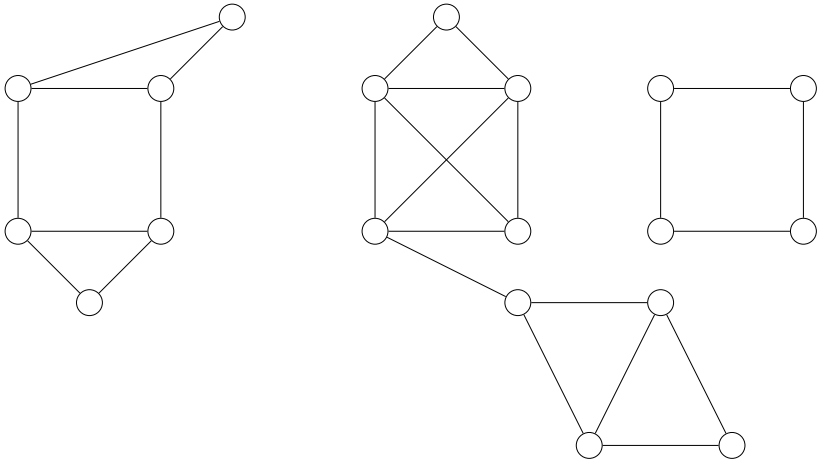


First Step: Generalize $\mathcal{A}$

Goal: Find suitable graph property $\mathcal{B} \supseteq \mathcal{A}$.



MOG_s $\subseteq$

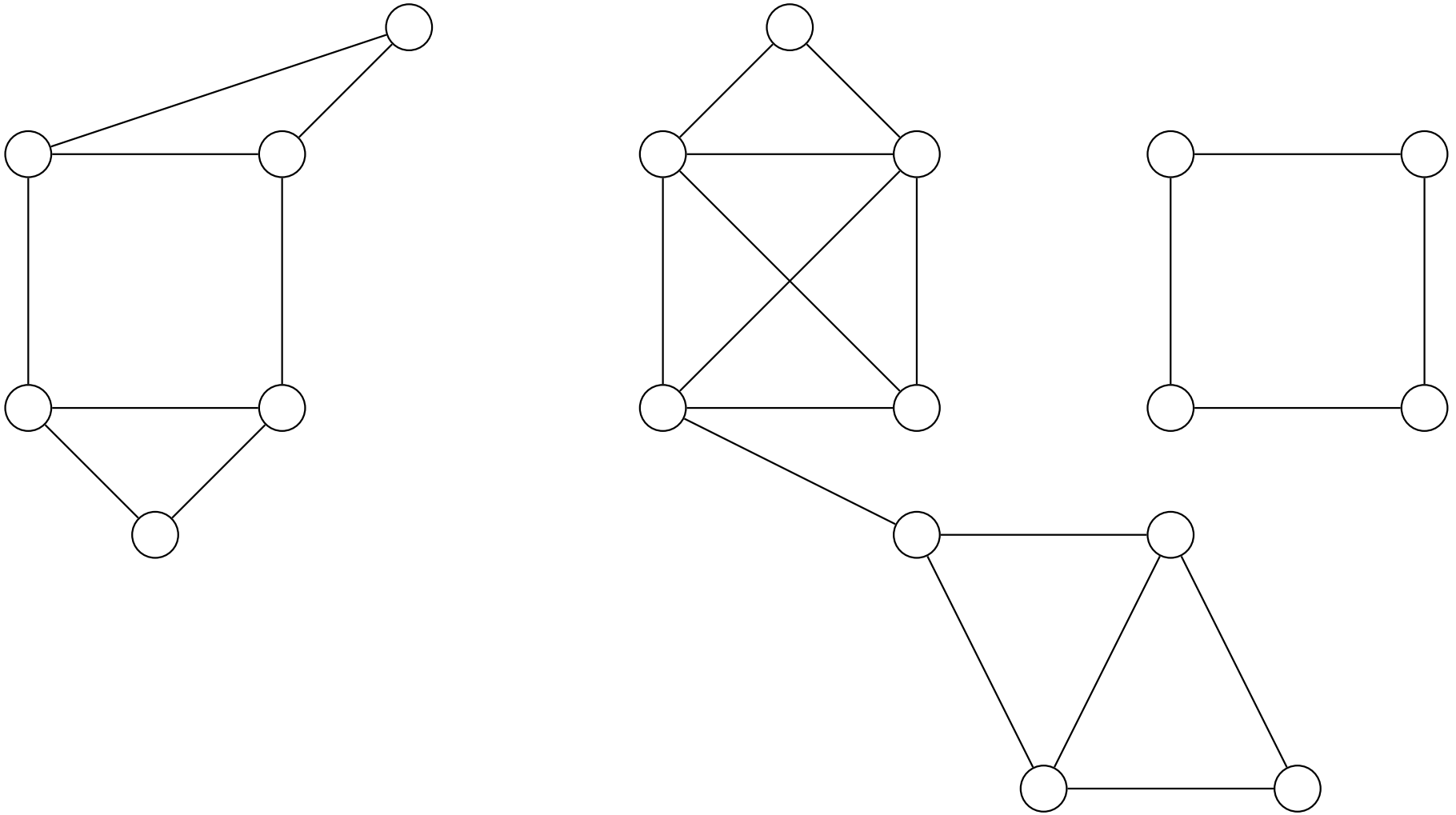


"leafless"

May concentrate on bounding failure term on "leafless" graphs.

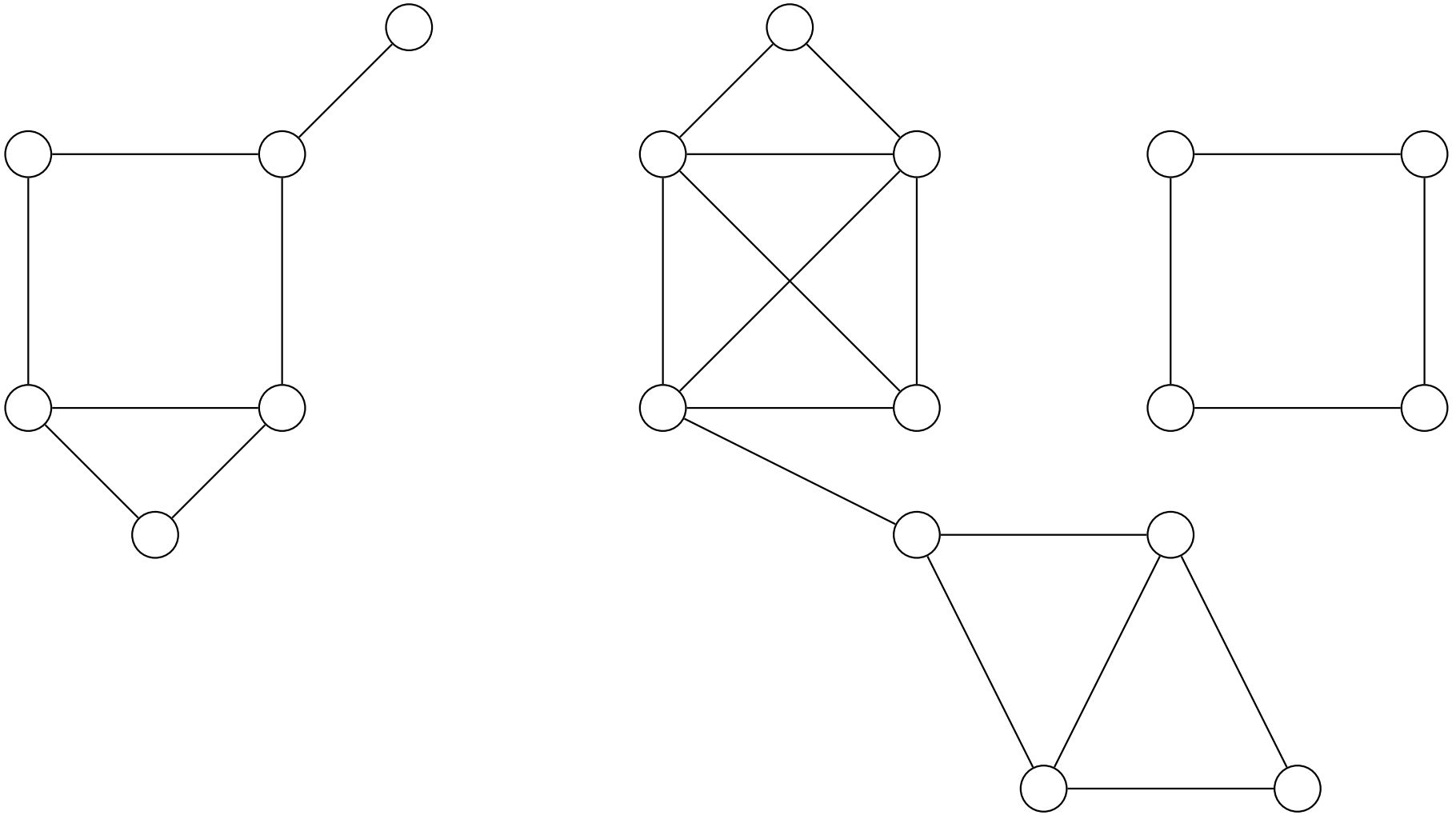
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



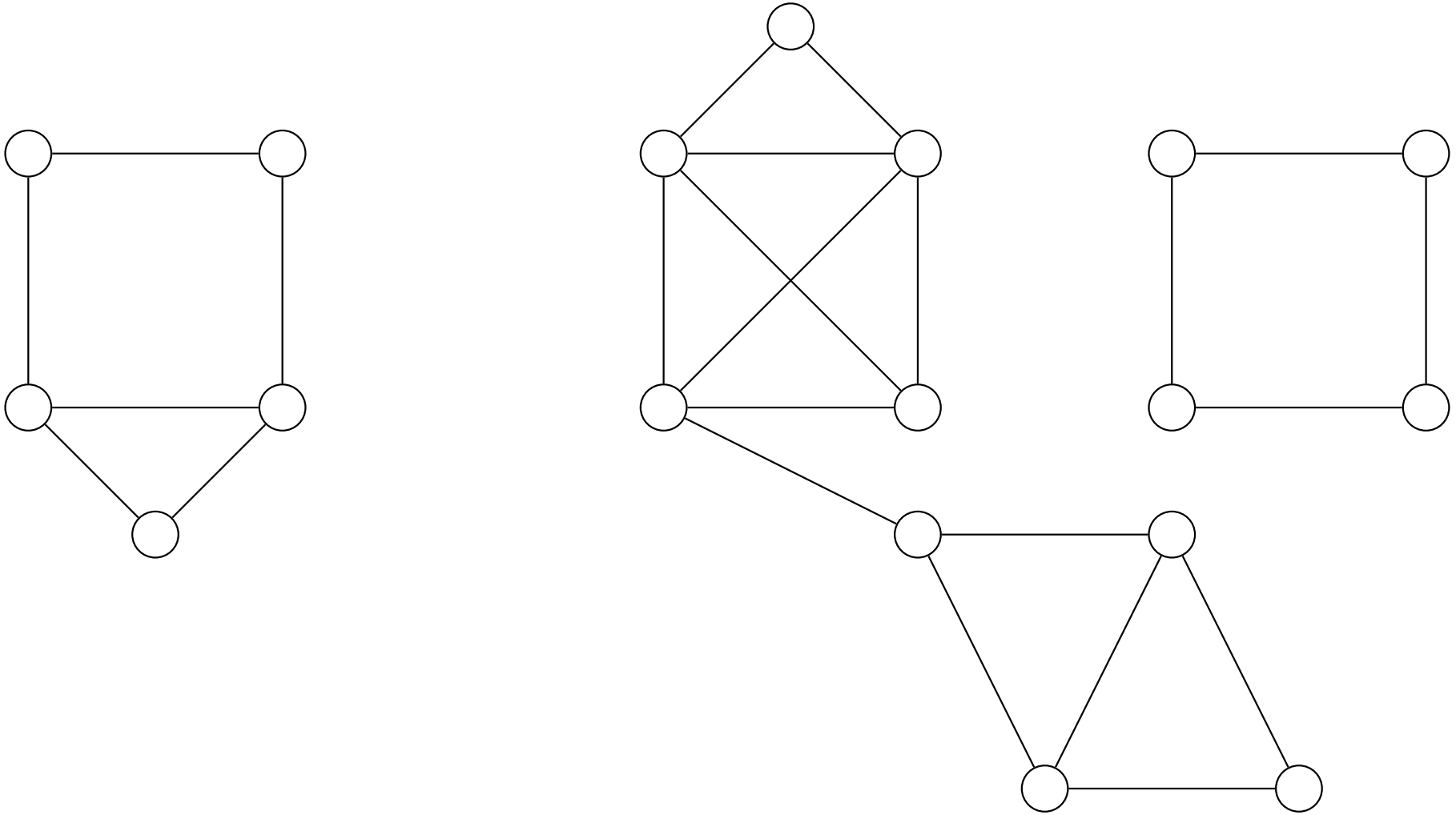
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S\colon G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



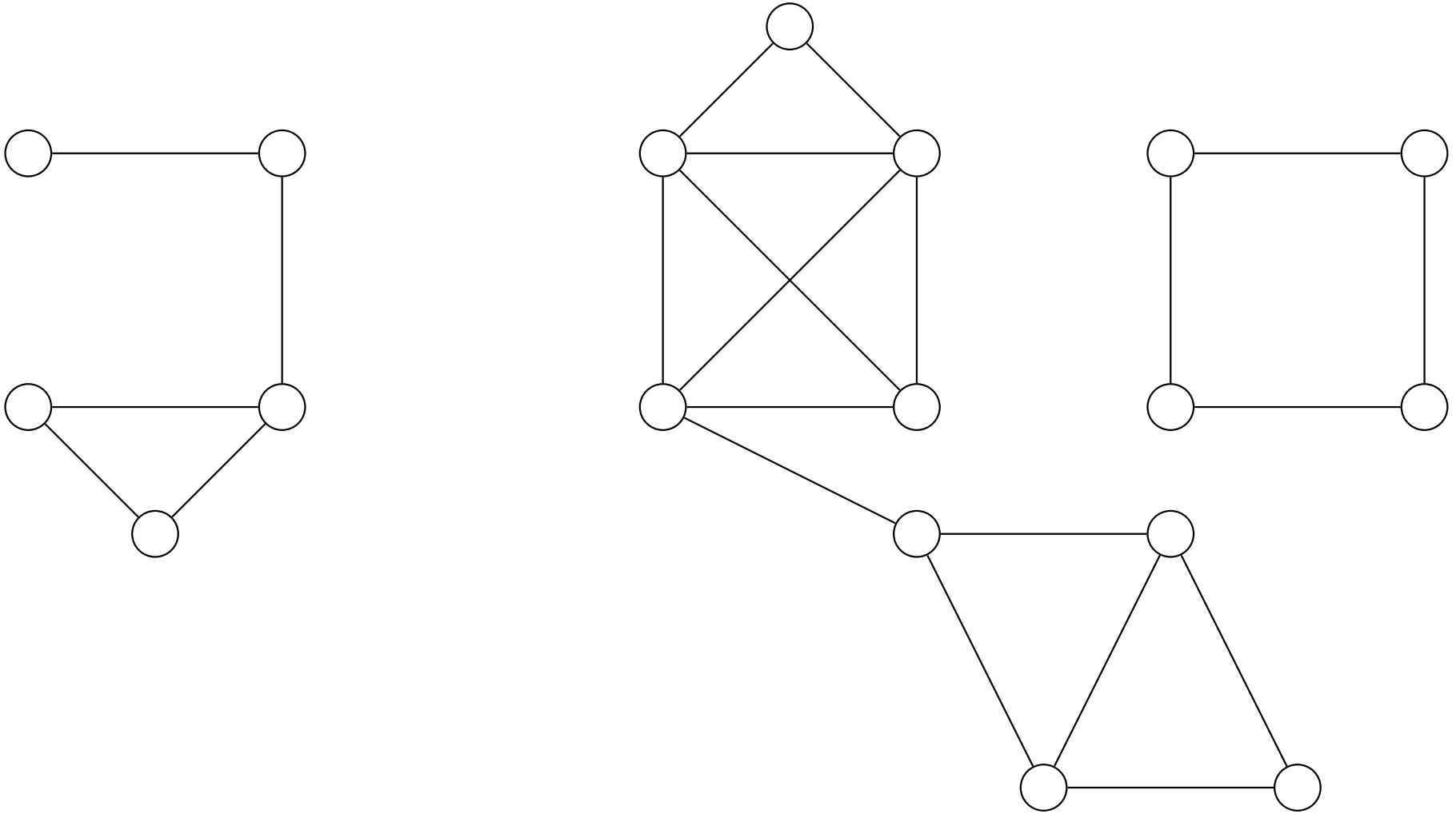
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap$ (h_1, h_2) are T-bad ".



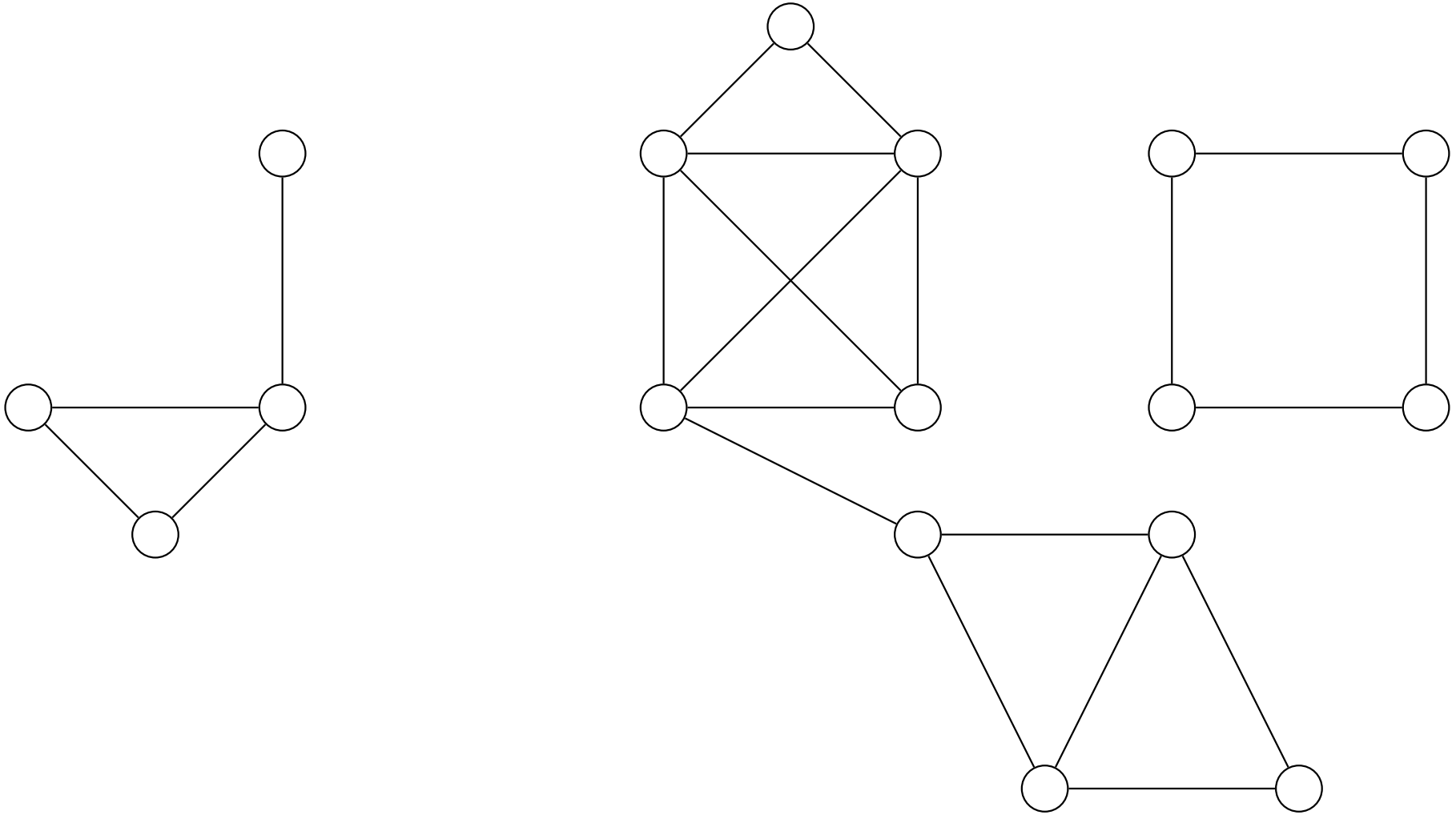
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



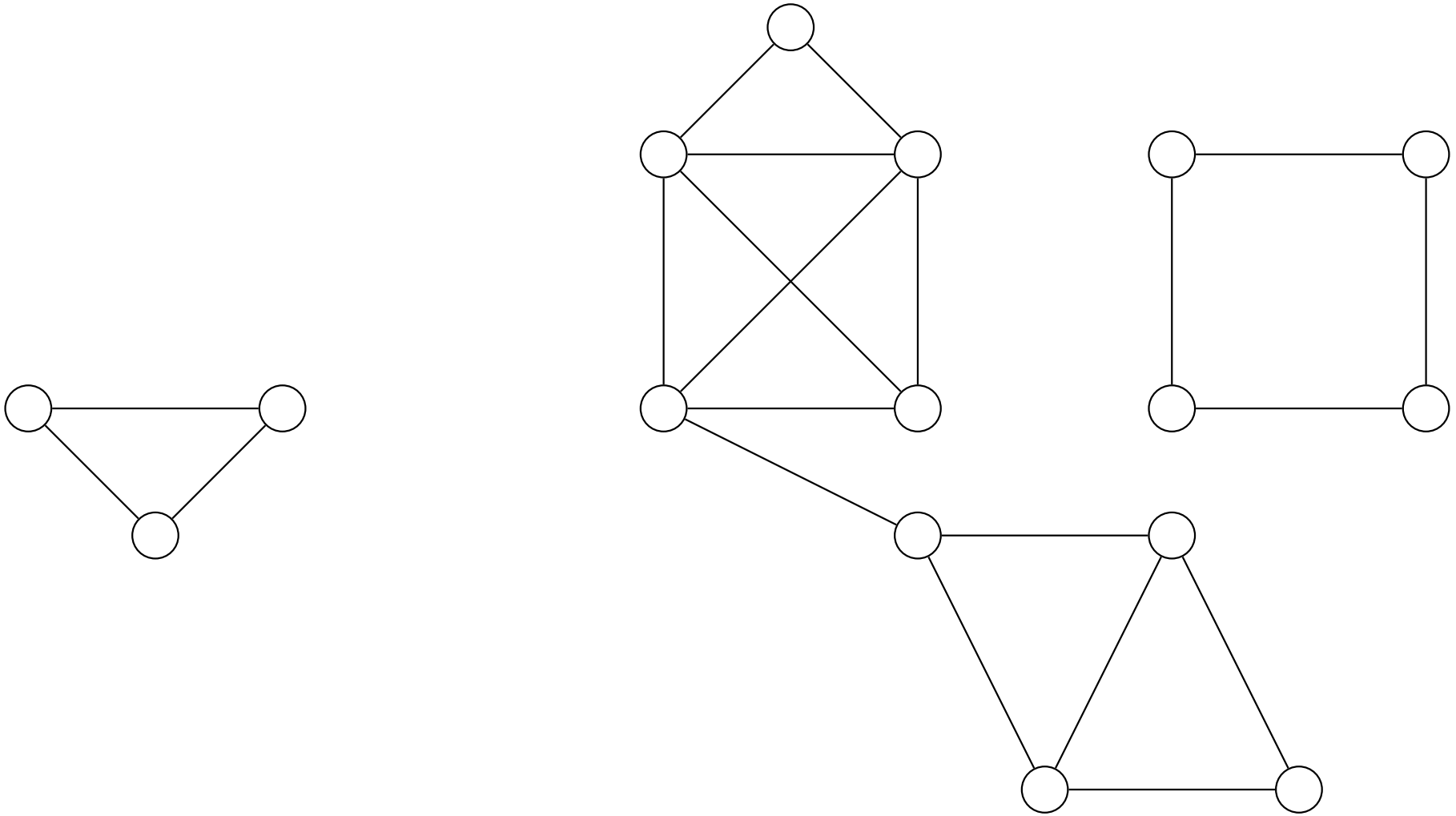
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



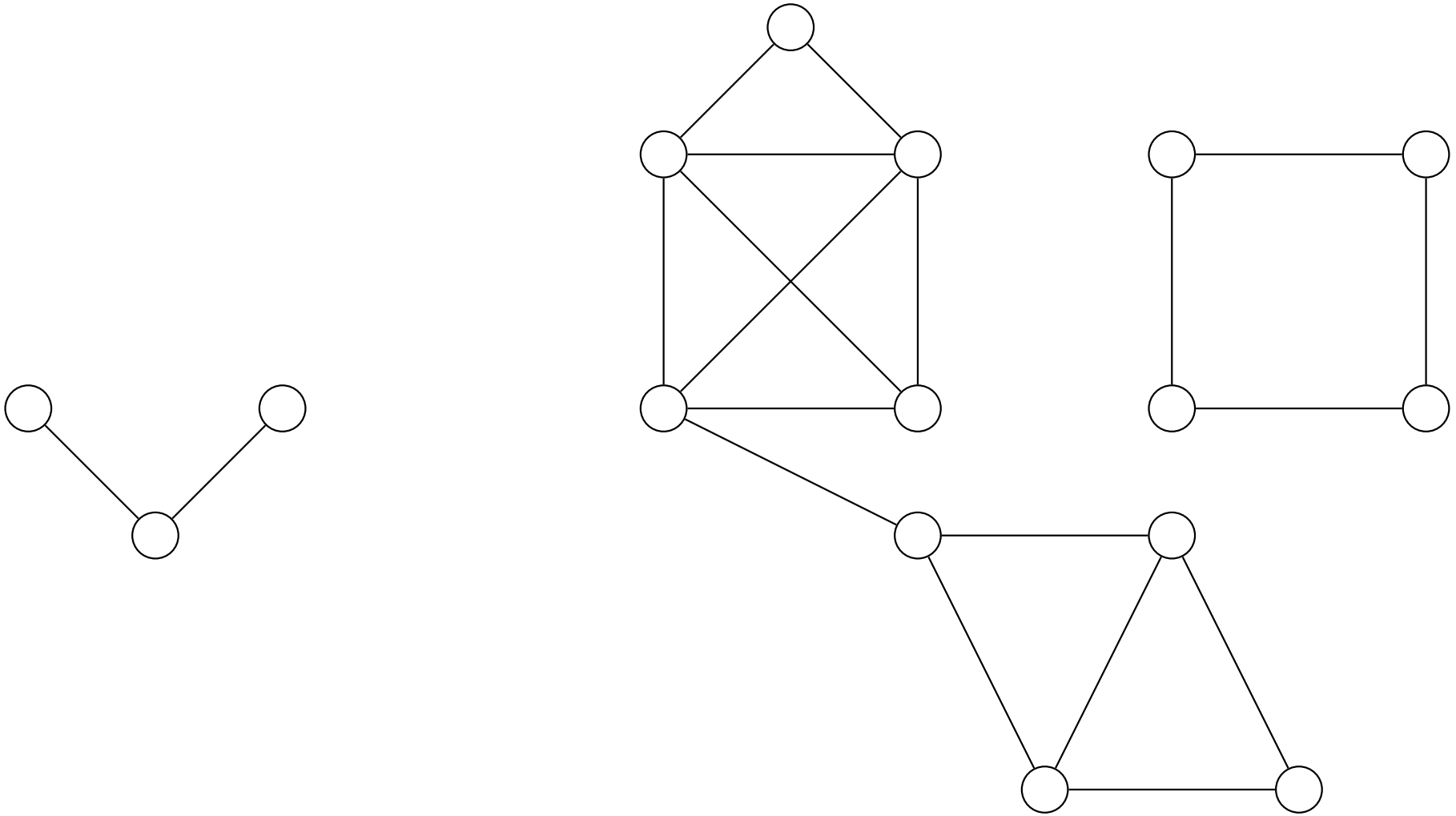
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap (h_1, h_2)$ are T-bad ".



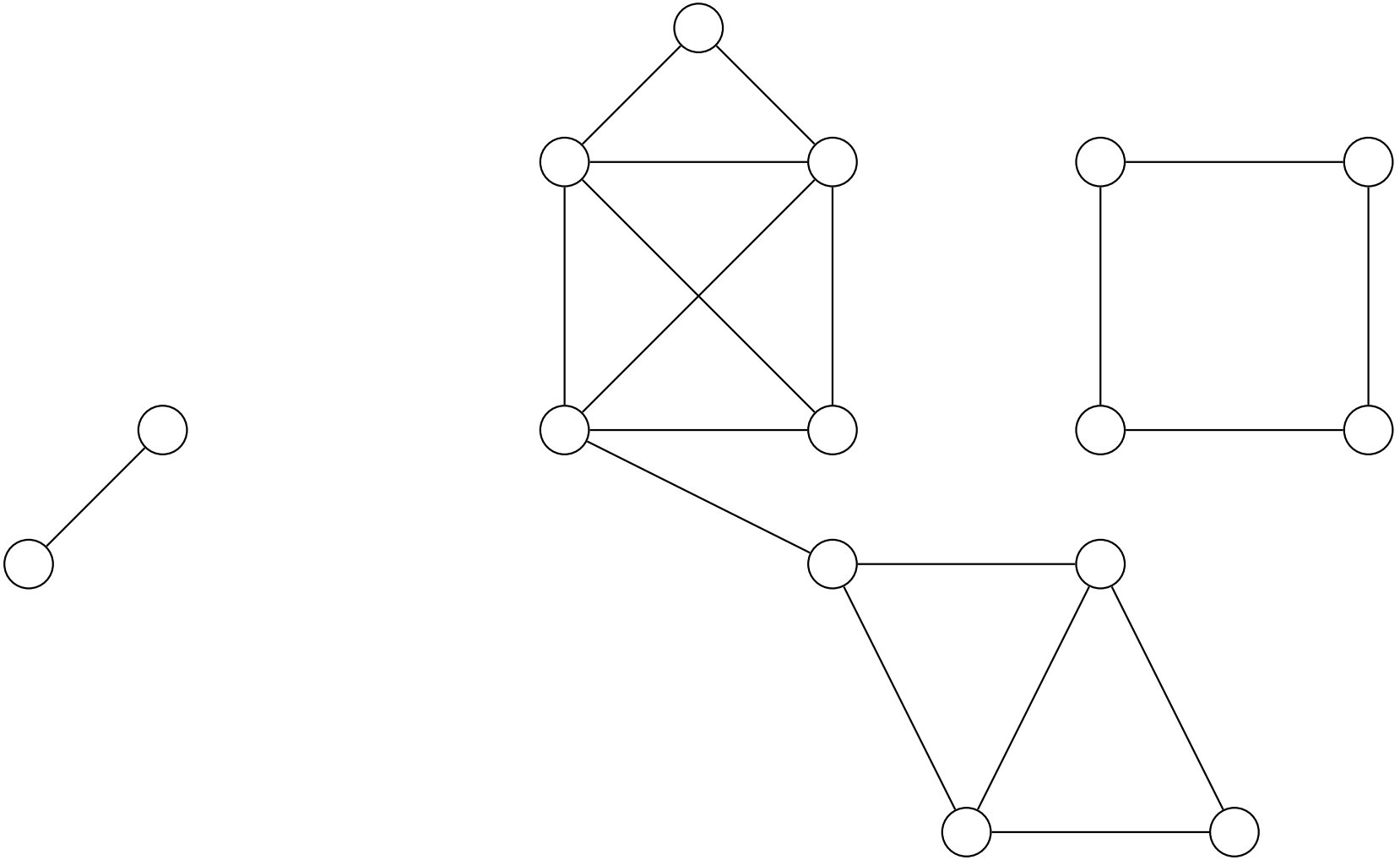
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap$ (h_1, h_2) are T-bad ".



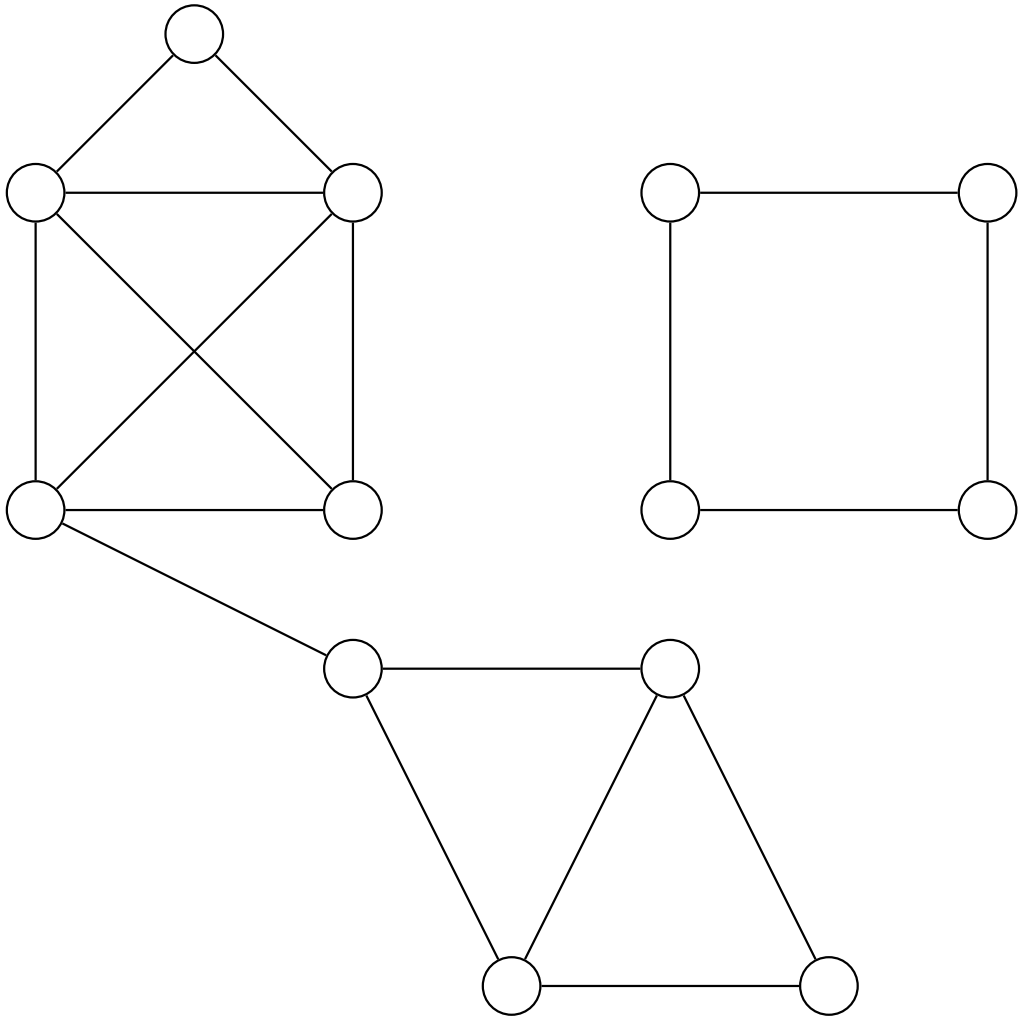
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S\colon G(T, h_1, h_2)$ is leafless $\cap\,(h_1, h_2)$ are T-bad ".



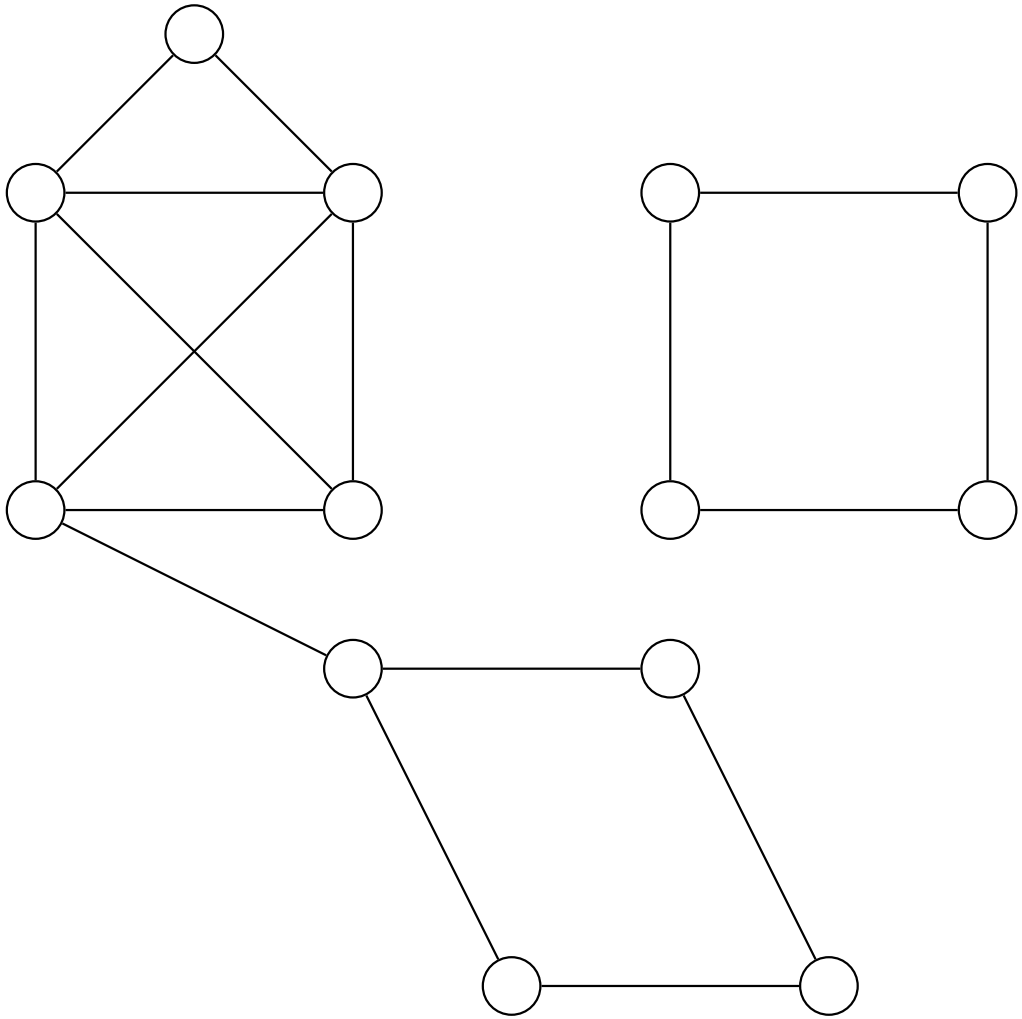
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



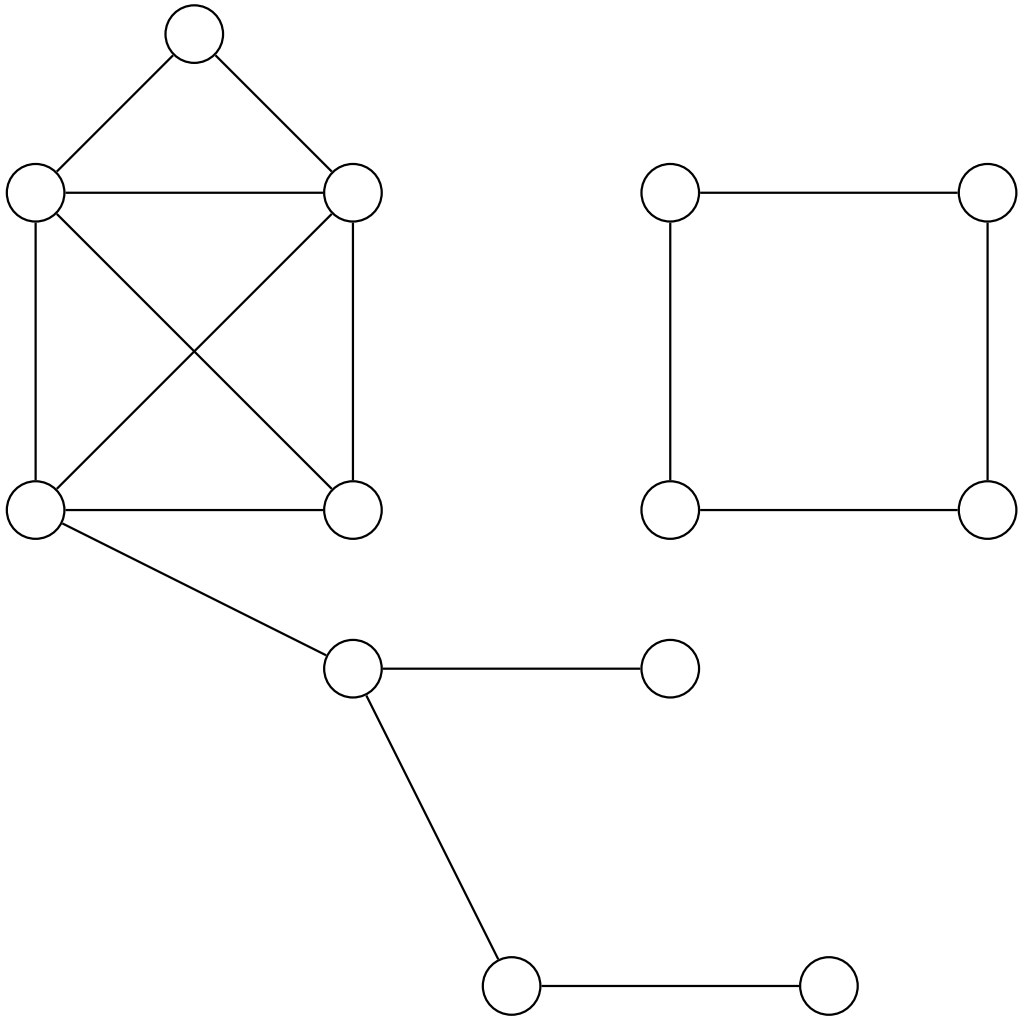
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



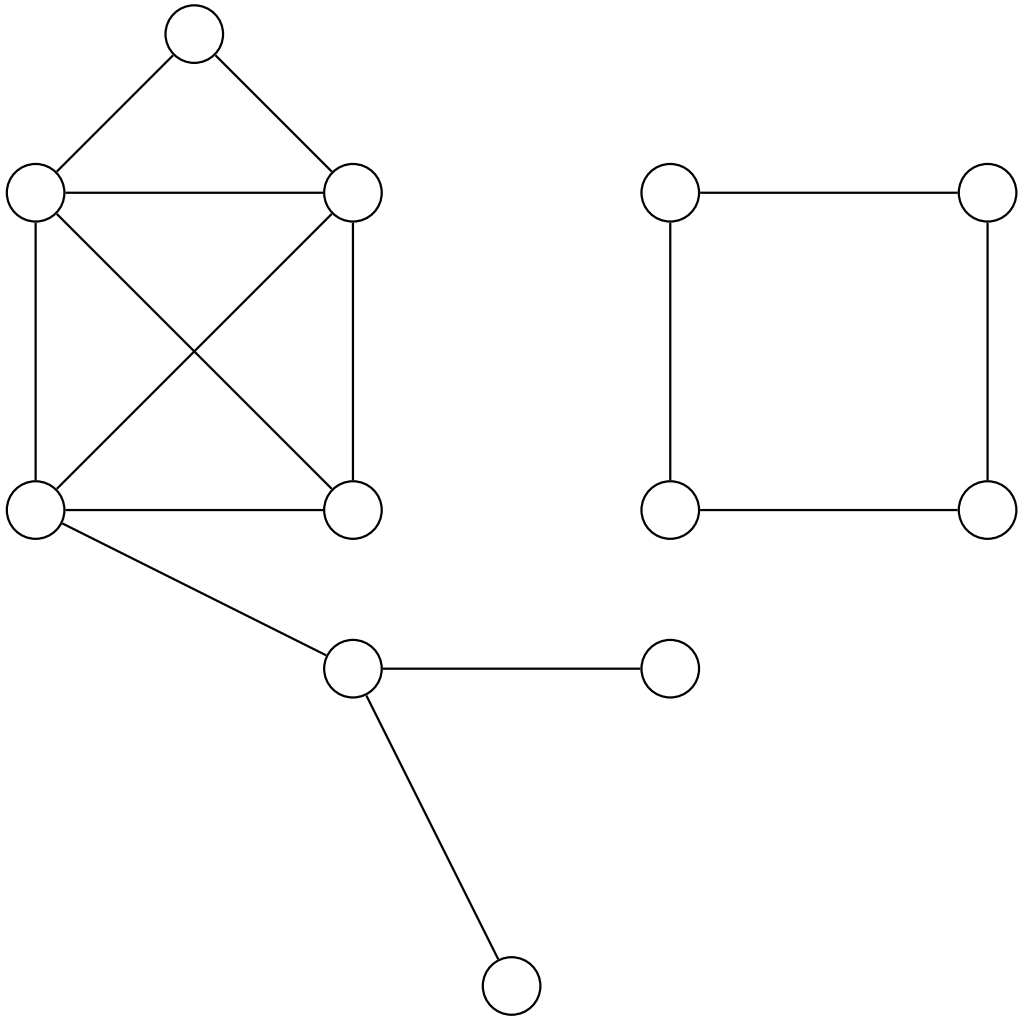
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S\colon G(T, h_1, h_2)$ is leafless $\cap\, (h_1, h_2)$ are T-bad ".



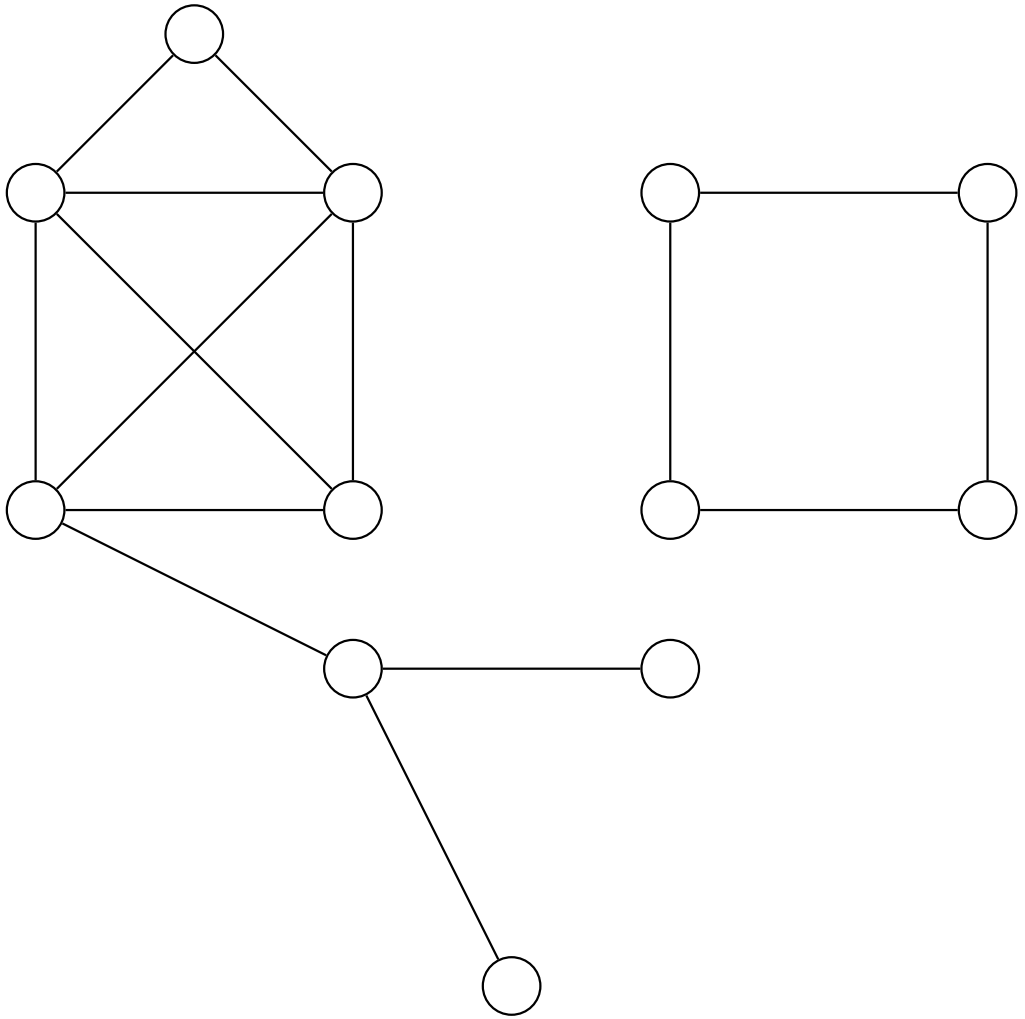
Second Step: Peel (DW '03)

Assume "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap (h_1, h_2)$ are T-bad ".



Second Step: Peel (DW '03)

If "$\exists T \subseteq S$: $G(T, h_1, h_2)$ is leafless $\cap$ (h_1, h_2) are T-bad "



then "$\exists T' \subseteq S$: $G(T', h_1, h_2)$ forms "peeled graph" $\cap$ (h_1, h_2) are T'-good"

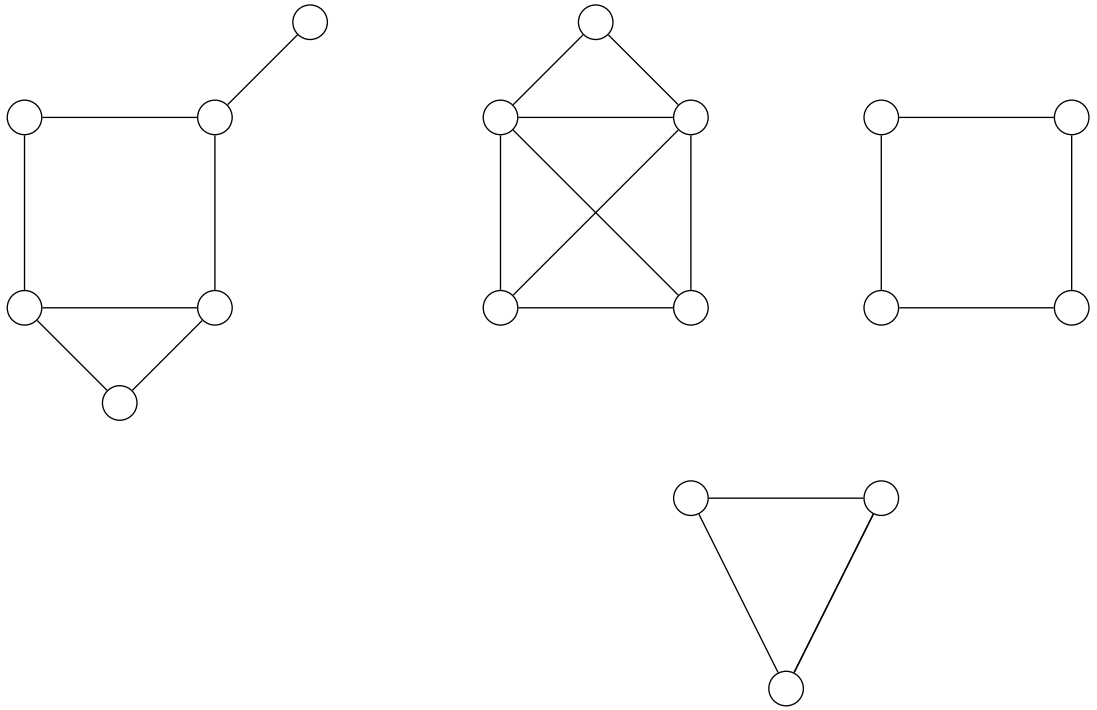
Third Step: Reduce

Goal: Reduce number of graphs that must be considered.

"Reducing:"

- preserve collisions (mark two colliding edges/keys per g_j function)

  Mandatory to obtain low failure probabilities.
- make graphs smaller



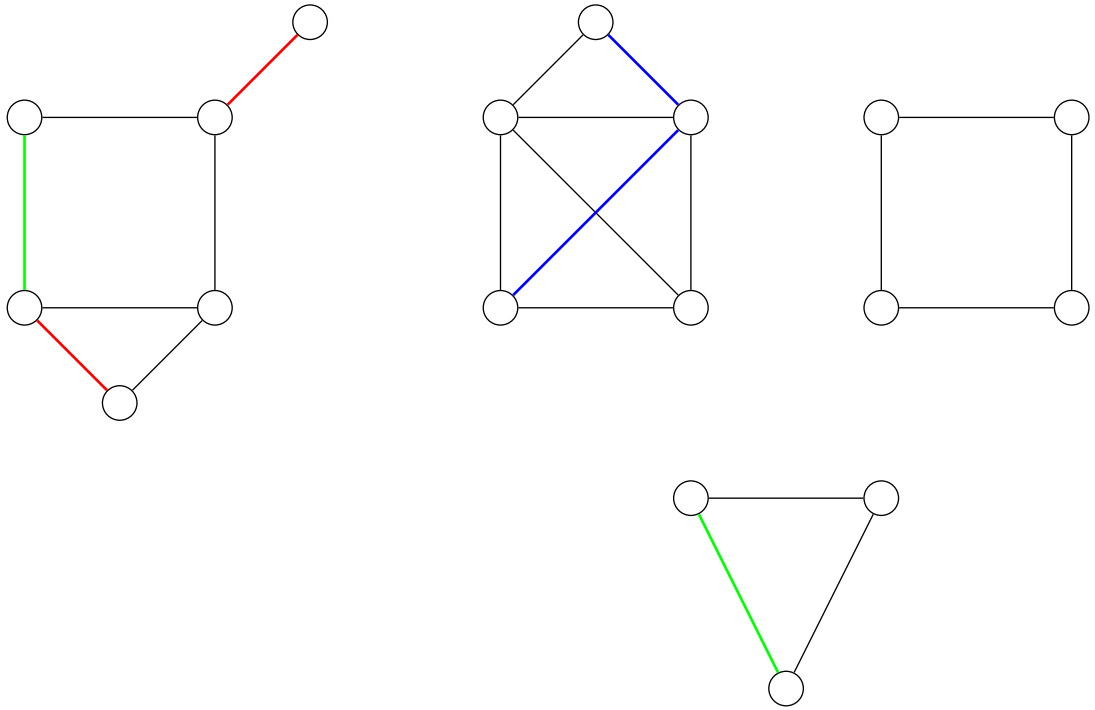
Third Step: Reduce

Goal: Reduce number of graphs that must be considered.

"Reducing:"

- preserve collisions (mark two colliding edges/keys per g_j function)

  Mandatory to obtain low failure probabilities.
- make graphs smaller



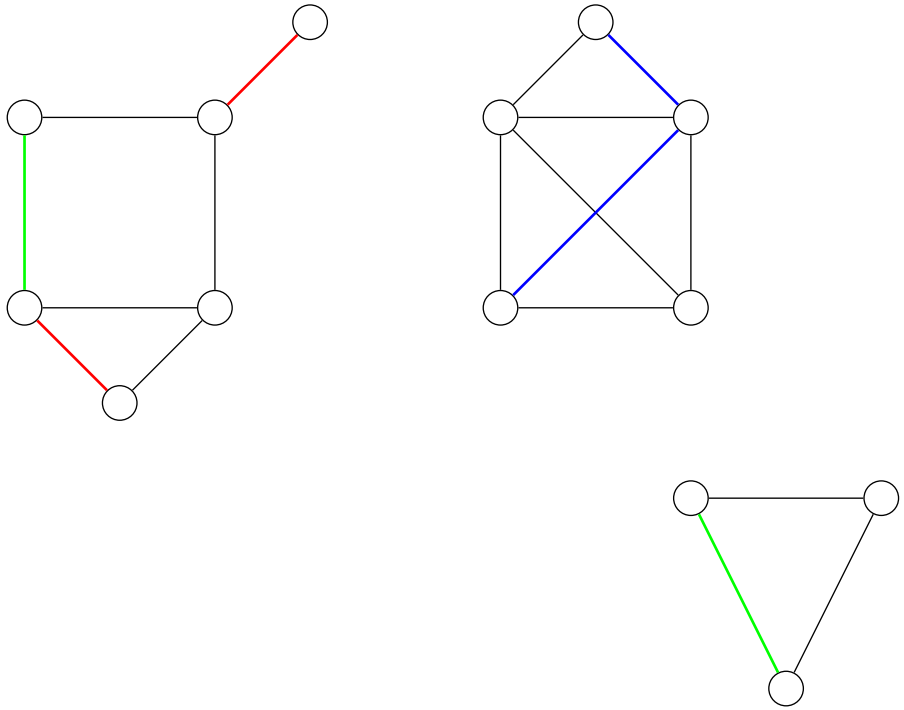
Third Step: Reduce

Goal: Reduce number of graphs that must be considered.

"Reducing:"

- preserve collisions (mark two colliding edges/keys per g_j function)

  Mandatory to obtain low failure probabilities.
- make graphs smaller



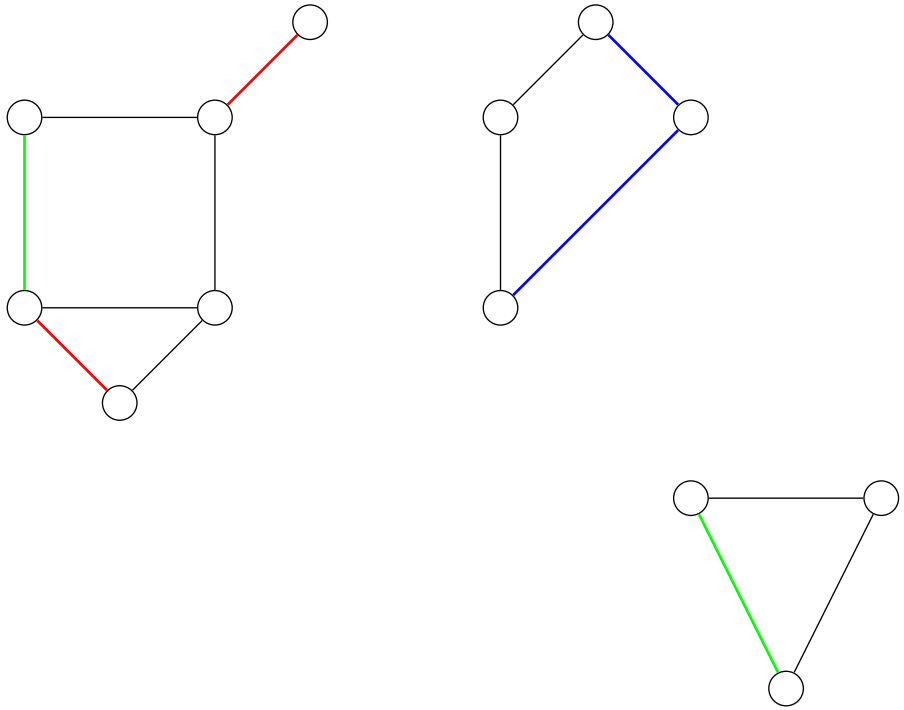
Third Step: Reduce

Goal: Reduce number of graphs that must be considered.

"Reducing:"

- preserve collisions (mark two colliding edges/keys per g_j function)

  Mandatory to obtain low failure probabilities.
- make graphs smaller



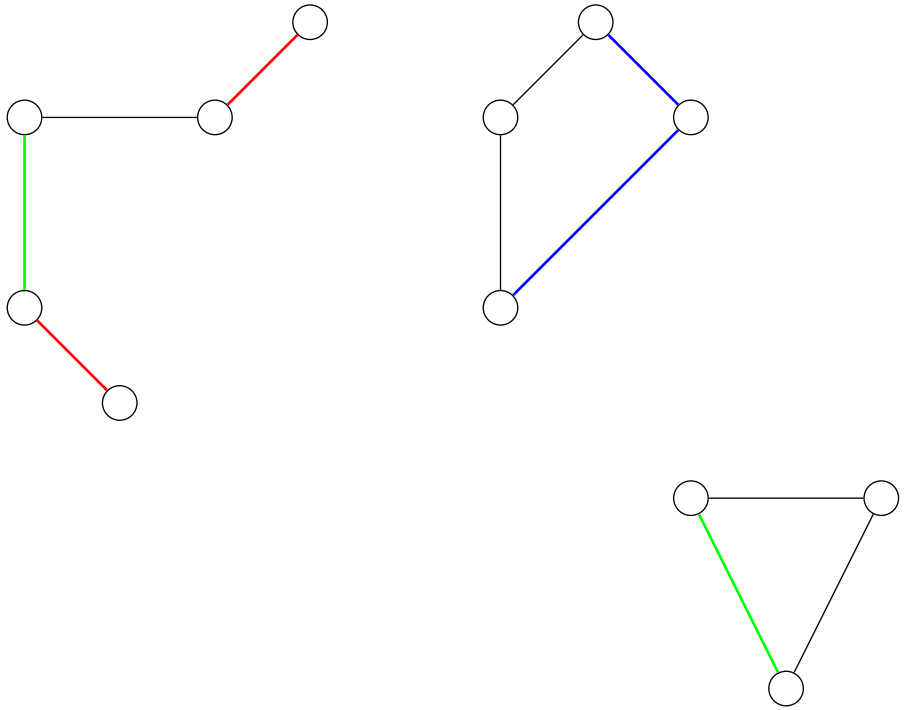
Third Step: Reduce

Goal: Reduce number of graphs that must be considered.

"Reducing:"

- preserve collisions (mark two colliding edges/keys per g_j function)

  Mandatory to obtain low failure probabilities.
- make graphs smaller



Result

- Graph property of interest: $\mathcal{A}$, know/calculate:

$$\mathrm{E}^*(\#\text{subgraphs with property } \mathcal{A}) = O\left(n^{-\alpha}\right).$$

- Generalize, Peel, Reduce. Suppose for "reduced" graph property $\mathcal{B}$:

$$\sum_{t=2}^{n} t^{O(1)} \mathrm{E}^*(\#\text{subgraphs with property } \mathcal{B} \text{ with } t \text{ edges}) = O\left(n^{\beta}\right).$$

Result

- Graph property of interest: $\mathcal{A}$, know/calculate:

$$\mathrm{E}^*(\#\text{subgraphs with property } \mathcal{A}) = O\left(n^{-\alpha}\right).$$

- Generalize, Peel, Reduce. Suppose for "reduced" graph property $\mathcal{B}$:

$$\sum_{t=2}^{n} t^{O(1)} \mathrm{E}^*(\#\text{subgraphs with property } \mathcal{B} \text{ with } t \text{ edges}) = O\left(n^{\beta}\right).$$

Trick: Generalized property can be re-used, e. g., "leafless".

Result

- Graph property of interest: $\mathcal{A}$, know/calculate:

$$\mathrm{E}^*(\#\text{subgraphs with property } \mathcal{A}) = O\left(n^{-\alpha}\right).$$

- Generalize, Peel, Reduce. Suppose for "reduced" graph property $\mathcal{B}$:

$$\sum_{t=2}^{n} t^{O(1)} \mathrm{E}^*(\#\text{subgraphs with property } \mathcal{B} \text{ with } t \text{ edges}) = O\left(n^{\beta}\right).$$

Trick: Generalized property can be re-used, e. g., "leafless".

Using $c \geq 2(\alpha + \beta)$ g-functions and tables gives

$$\Pr_{(h_1,h_2)\in\mathcal{Z}}(\text{Graph contains subgraph with property } \mathcal{A}) = O(n^{-\alpha}).$$

Checking conditions only based on random graph theory!

Results in Thesis

Graphs ("Plug-and-play"):

Results in Thesis

Graphs ("Plug-and-play"):

- Cuckoo hashing (with a stash)
- Simulation of a uniform hash function (Pagh/Pagh '03)
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani '13)

Results in Thesis

Graphs ("Plug-and-play"):

- Cuckoo hashing (with a stash)
- Simulation of a uniform hash function (Pagh/Pagh '03)
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani '13)

Hypergraphs:

Results in Thesis

Graphs ("Plug-and-play"):

- Cuckoo hashing (with a stash)
- Simulation of a uniform hash function (Pagh/Pagh '03)
- Constructing a perfect hash function (Bothelo/Pagh/Ziviani '13)

Hypergraphs:

- Parallel/Sequential Load Balancing (Schickinger/Steger '00).

  Generalizes ad-hoc analysis by Woelfel '06.
- Generalized cuckoo hashing (≥ 3 hash functions, $\ell \geq 2$ keys per cell):
  Could not obtain suitable bounds

Conclusion: Hashing

We have seen:

- a construction for efficient hash functions
- method to bound probability that certain subgraphs exist
- some applications of this hash class

Further Directions:

- initialization with more practical parameters?
- more applications?
- compare performance to new "high-independence" hash classes (Thorup '13, Christiani/Pagh/Thorup '15)
- method: achieve bounds beyond "first moment" (e. g., Galton-Watson Branching Processes)

Vielen Dank für Ihre Aufmerksamkeit!

Thank you for your attention!

Mange tak!