Non-minimal k-perfect hashing: Tight lower bounds and an application to fast static hash tables

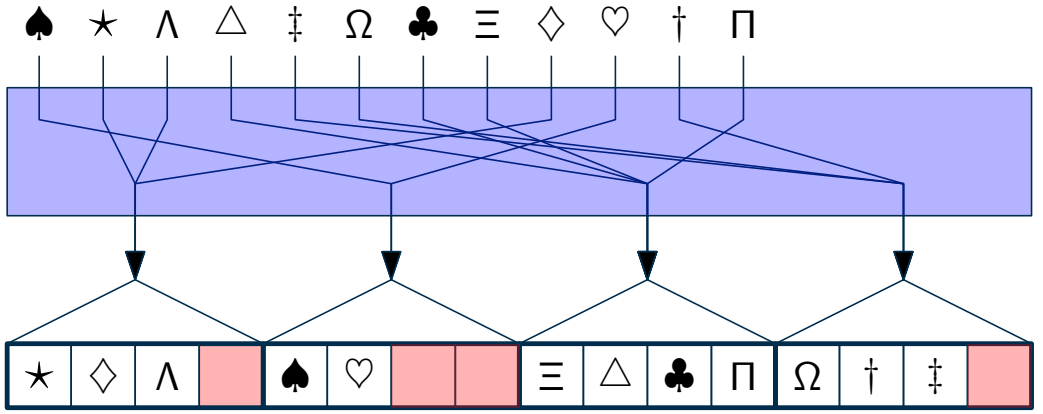
ESA 2026, L'Aquila

Ragnar Groot Koerkamp

Stefan Hermann

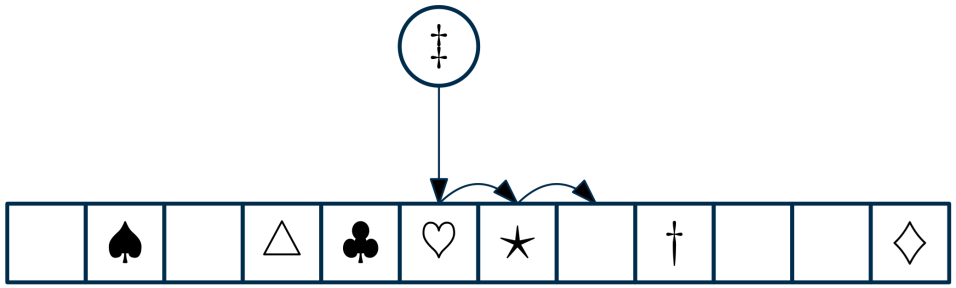
Peter Sanders

Stefan Walzer

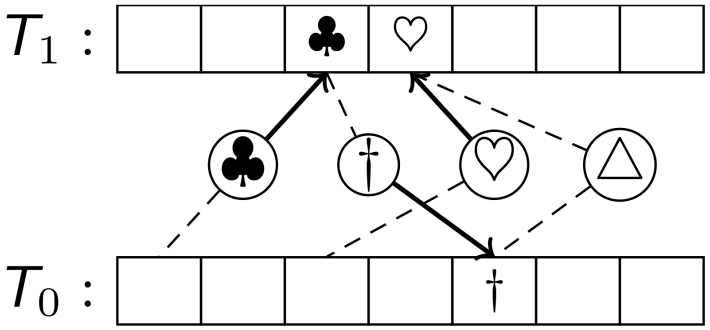


Families of Hash Tables

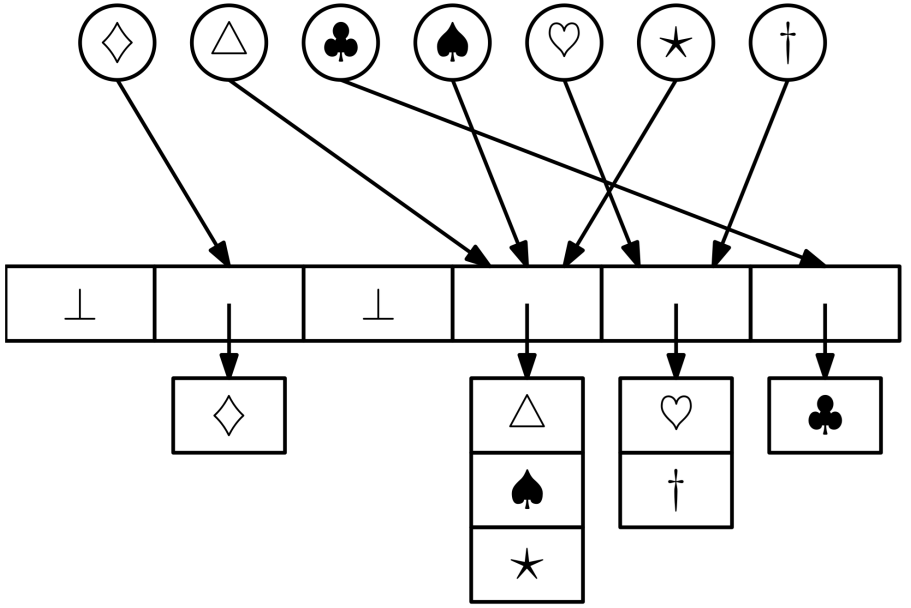
Probing



Cuckoo



Chaining

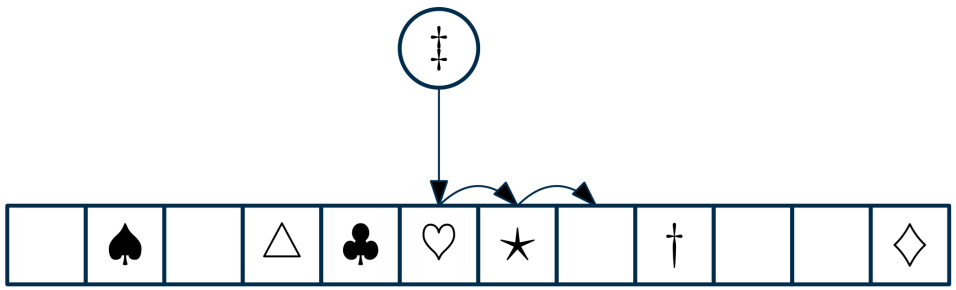


Perfect Hashing (for static key sets)

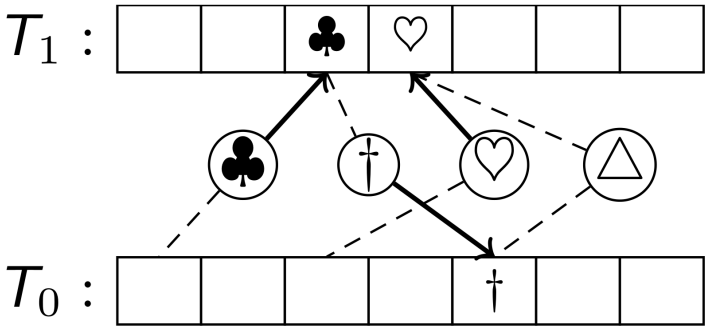
Hash without collisions.

Families of Hash Tables

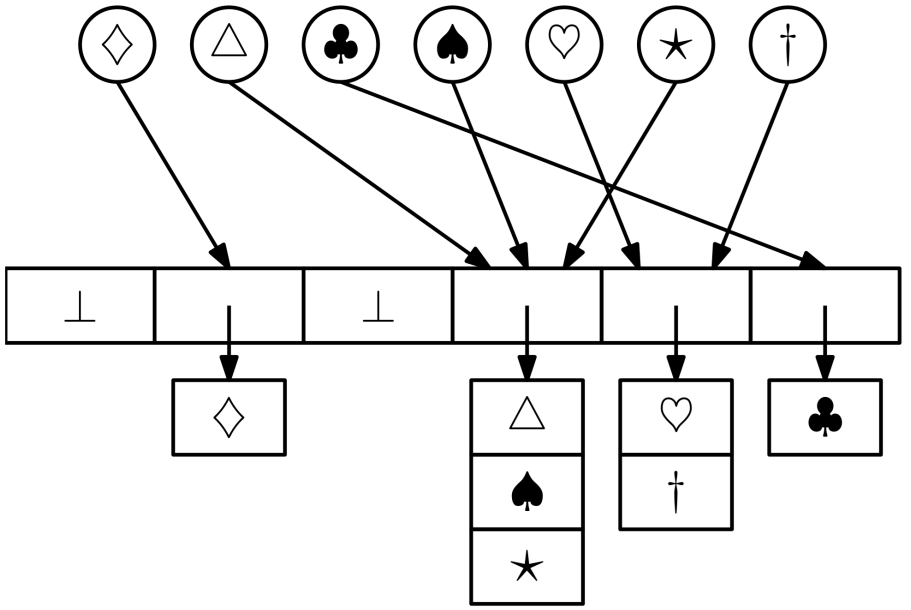
Probing



Cuckoo



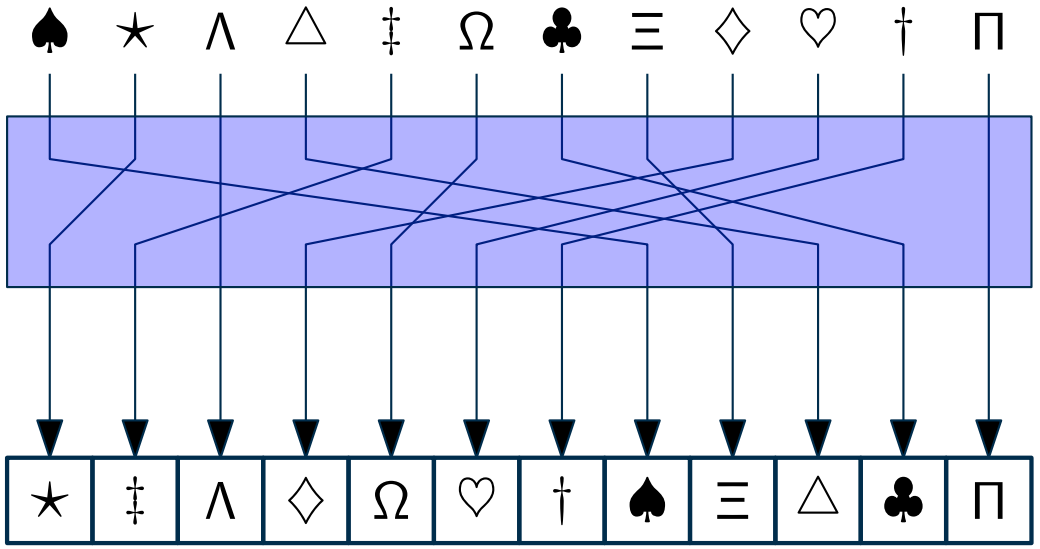
Chaining



Perfect Hashing (for static key sets)

Hash without collisions.

Perfect Hashing and its Variants



Observation

$$\Pr[\text{random hash function is bijection}] = \frac{n!}{n^n}$$

Implication

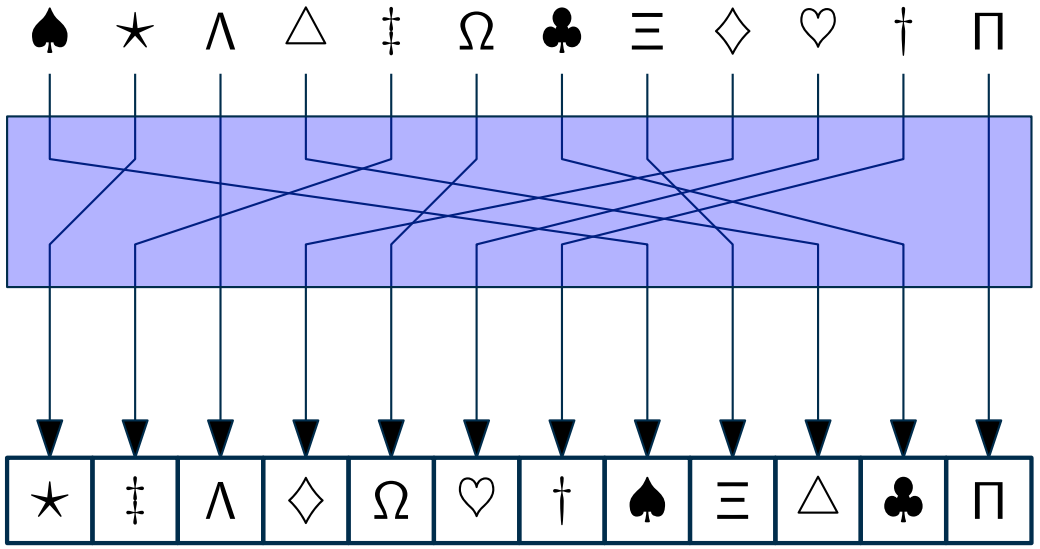
A (1,1)-perfect hash function requires space

$$\log_2 \frac{n^n}{n!} \approx \log_2(e^n) \approx 1.44n \text{ bits}$$

Generalisation: (α, k)-perfect hashing

- α: load factor
- k: bin size

Perfect Hashing and its Variants



Observation

$$\Pr[\text{random hash function is bijection}] = \frac{n!}{n^n}$$

Implication

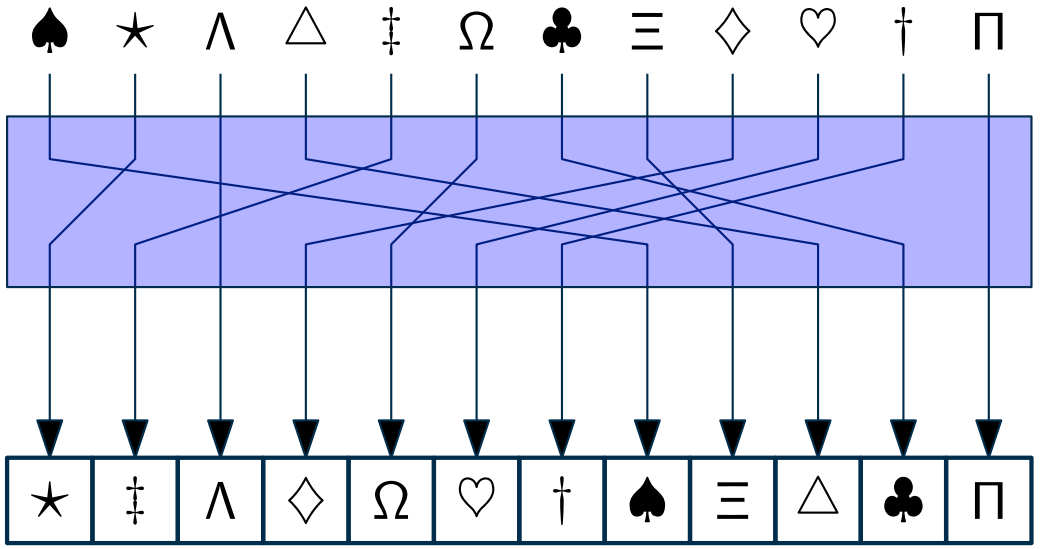
A (1,1)-perfect hash function requires space

$$\log_2 \frac{n^n}{n!} \approx \log_2(e^n) \approx 1.44n \text{ bits}$$

Generalisation: (α, k)-perfect hashing

- α: load factor
- k: bin size

Perfect Hashing and its Variants



Observation

$$\Pr[\text{random hash function is bijection}] = \frac{n!}{n^n}$$

Implication

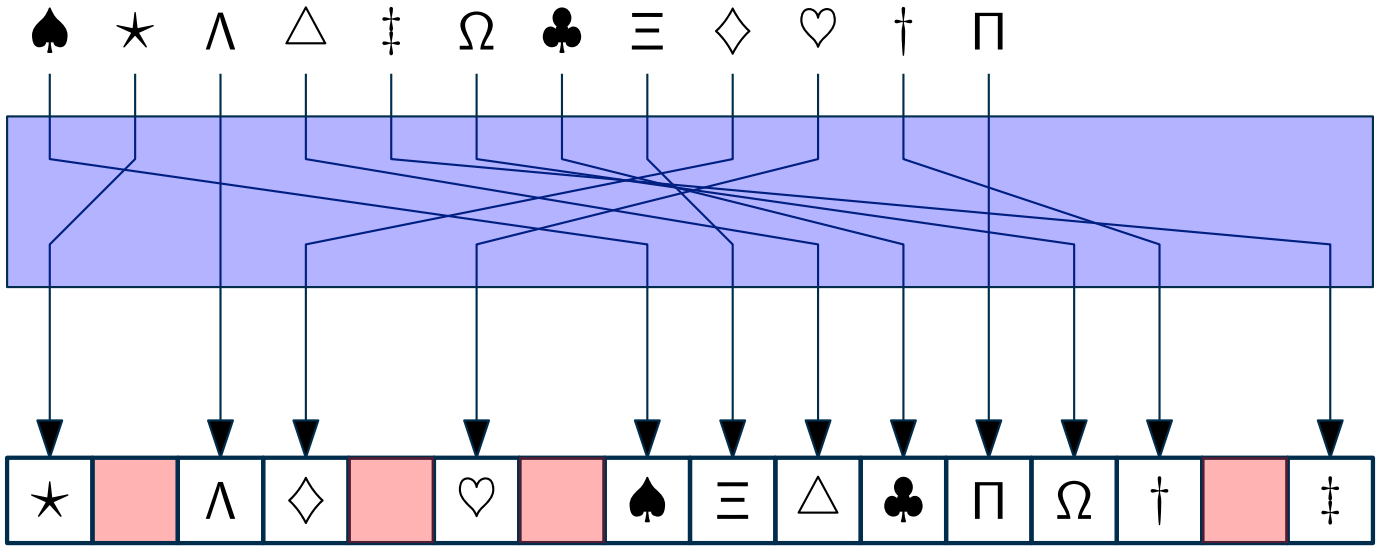
A (1,1)-perfect hash function requires space

$$\log_2 \frac{n^n}{n!} \approx \log_2(e^n) \approx 1.44n \text{ bits}$$

Generalisation: (α, k)-perfect hashing

- α: load factor
- k: bin size

Perfect Hashing and its Variants



Observation

$$\Pr[\text{random hash function is bijection}] = \frac{n!}{n^n}$$

Implication

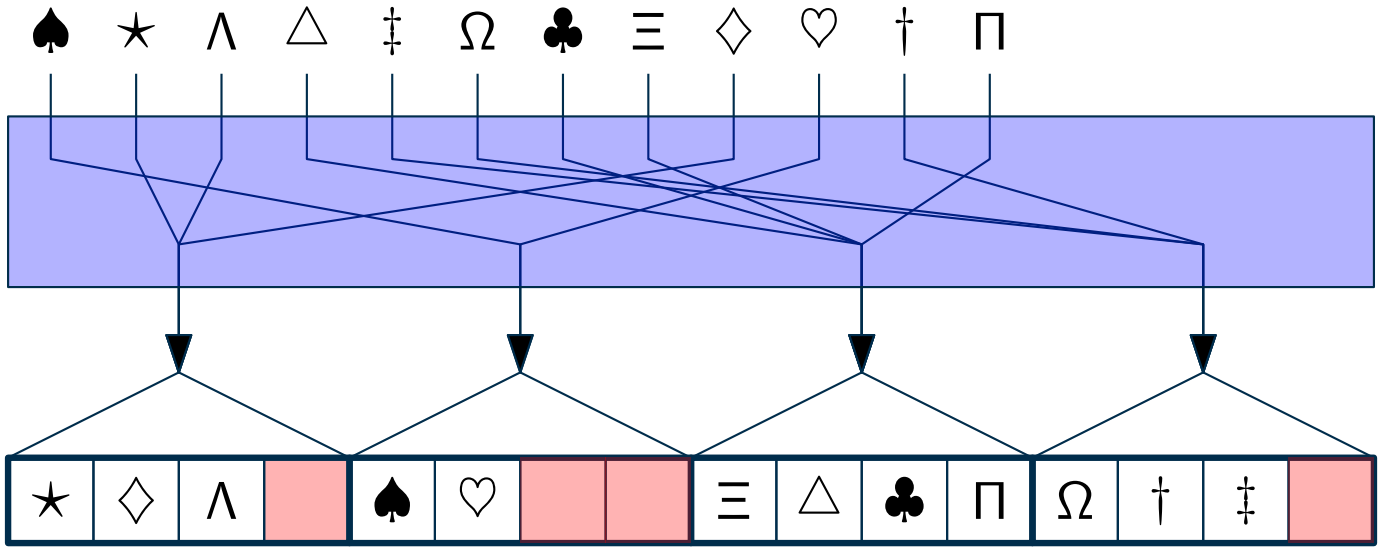
A (1,1)-perfect hash function requires space

$$\log_2 \frac{n^n}{n!} \approx \log_2(e^n) \approx 1.44n \text{ bits}$$

Generalisation: (α, k)-perfect hashing

- α: load factor
- k: bin size

Perfect Hashing and its Variants



Observation

$$\Pr[\text{random hash function is bijection}] = \frac{n!}{n^n}$$

Implication

A (1,1)-perfect hash function requires space

$$\log_2 \frac{n^n}{n!} \approx \log_2(e^n) \approx 1.44n \text{ bits}$$

Generalisation: (α, k)-perfect hashing

- α: load factor
- k: bin size

First Contribution: Space for (α, k)-Perfect Hashing

bits per key	$k = 1$	$k = 2$	$k = 4$	$k = 8$	...
$\alpha = 1.00$	1.443	0.943	0.589	0.355	...
$\alpha = 0.99$	1.376	0.881	0.533	0.307	...
$\alpha = 0.95$	1.215	0.740	0.417	0.217	...
$\alpha = 0.90$	1.074	0.622	0.327	0.154	...
$\alpha = 0.80$	0.862	0.456	0.210	0.083	...
⋮	⋮	⋮	⋮	⋮	

As before:

- space $= \log_2 \frac{1}{q} + O(1)$
- where $q = \Pr[\text{a random function is } (\alpha, k)\text{-perfect}]$.

Remaining Challenge

- compute q

First Contribution: Space for (α, k)-Perfect Hashing

bits per key	$k = 1$	$k = 2$	$k = 4$	$k = 8$	...
$\alpha = 1.00$	1.443	0.943	0.589	0.355	...
$\alpha = 0.99$	1.376	0.881	0.533	0.307	...
$\alpha = 0.95$	1.215	0.740	0.417	0.217	...
$\alpha = 0.90$	1.074	0.622	0.327	0.154	...
$\alpha = 0.80$	0.862	0.456	0.210	0.083	...
⋮	⋮	⋮	⋮	⋮	

As before:

- space $= \log_2 \frac{1}{q} + O(1)$
- where $q = \Pr[\text{a random function is } (\alpha, k)\text{-perfect}]$.

Remaining Challenge

- compute q

First Contribution: Space for (α, k)-Perfect Hashing

bits per key	$k = 1$	$k = 2$	$k = 4$	$k = 8$	...
$\alpha = 1.00$	1.443	0.943	0.589	0.355	...
$\alpha = 0.99$	1.376	0.881	0.533	0.307	...
$\alpha = 0.95$	1.215	0.740	0.417	0.217	...
$\alpha = 0.90$	1.074	0.622	0.327	0.154	...
$\alpha = 0.80$	0.862	0.456	0.210	0.083	...
⋮	⋮	⋮	⋮	⋮	

As before:

- space $= \log_2 \frac{1}{q} + O(1)$
- where $q = \Pr[\text{a random function is } (\alpha, k)\text{-perfect}]$.

Remaining Challenge

- compute q

First Contribution: Space for (α, k)-Perfect Hashing

bits per key	$k = 1$	$k = 2$	$k = 4$	$k = 8$	...
$\alpha = 1.00$	1.443	0.943	0.589	0.355	...
$\alpha = 0.99$	1.376	0.881	0.533	0.307	...
$\alpha = 0.95$	1.215	0.740	0.417	0.217	...
$\alpha = 0.90$	1.074	0.622	0.327	0.154	...
$\alpha = 0.80$	0.862	0.456	0.210	0.083	...
⋮	⋮	⋮	⋮	⋮	

As before:

- space $= \log_2 \frac{1}{q} + O(1)$
- where $q = \Pr[\text{a random function is } (\alpha, k)\text{-perfect}]$.

Remaining Challenge

- compute q

First Contribution: Space for (α, k)-Perfect Hashing

bits per key	$k = 1$	$k = 2$	$k = 4$	$k = 8$	...
$\alpha = 1.00$	1.443	0.943	0.589	0.355	...
$\alpha = 0.99$	1.376	0.881	0.533	0.307	...
$\alpha = 0.95$	1.215	0.740	0.417	0.217	...
$\alpha = 0.90$	1.074	0.622	0.327	0.154	...
$\alpha = 0.80$	0.862	0.456	0.210	0.083	...
⋮	⋮	⋮	⋮	⋮	

As before:

- space $= \log_2 \frac{1}{q} + O(1)$
- where $q = \Pr[\text{a random function is } (\alpha, k)\text{-perfect}]$.

Remaining Challenge

- compute q

First Contribution: Space for (α, k)-Perfect Hashing

bits per key	$k = 1$	$k = 2$	$k = 4$	$k = 8$	...
$\alpha = 1.00$	1.443	0.943	0.589	0.355	...
$\alpha = 0.99$	1.376	0.881	0.533	0.307	...
$\alpha = 0.95$	1.215	0.740	0.417	0.217	...
$\alpha = 0.90$	1.074	0.622	0.327	0.154	...
$\alpha = 0.80$	0.862	0.456	0.210	0.083	...
⋮	⋮	⋮	⋮	⋮	

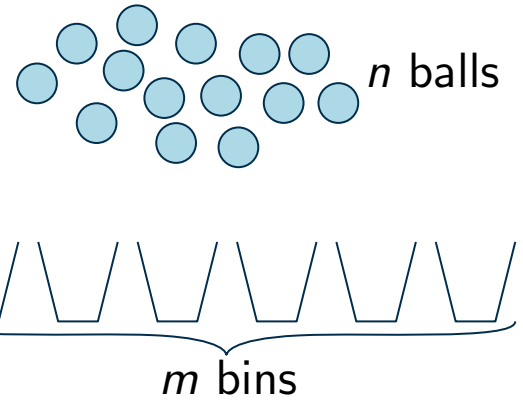
As before:

- space $= \log_2 \frac{1}{q} + O(1)$
- where $q = \Pr[\text{a random function is } (\alpha, k)\text{-perfect}]$.

Remaining Challenge

- compute q

Poissonisation with a Twist



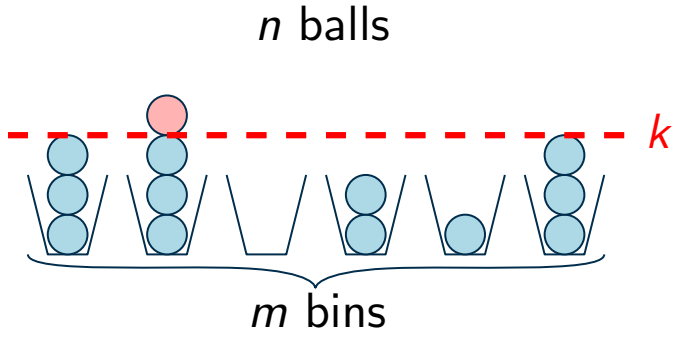
Problem

$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

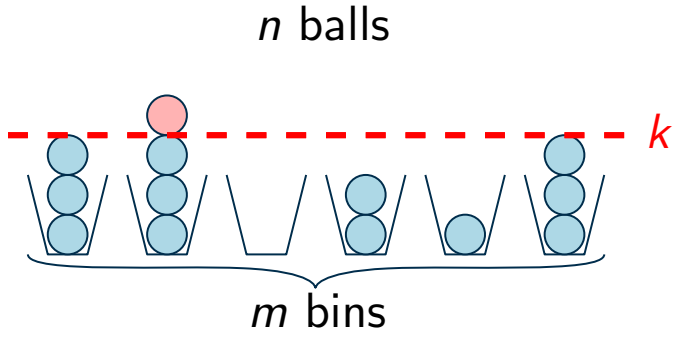
Poissonisation with a Twist

Problem

$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$



Poissonisation with a Twist



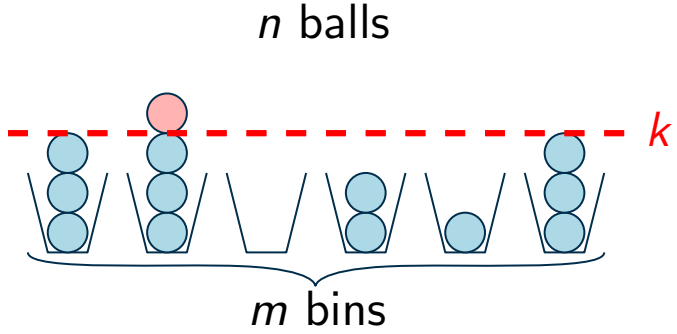
Problem

$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Can we pretent independence?

$$q \overset{?}{\approx} \Pr[\text{first bin load} \leq k]^m$$

Poissonisation with a Twist



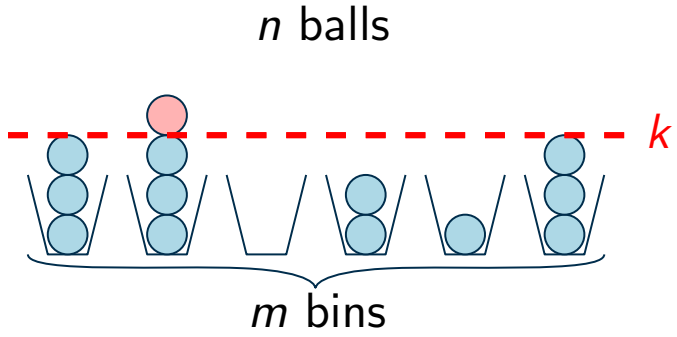
Problem

$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Can we pretent independence?

$$q \ll \underbrace{\Pr[\text{first bin load} \leq k]^m}_{\text{doesn't notice } n > mk}$$

Poissonisation with a Twist



Problem

$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

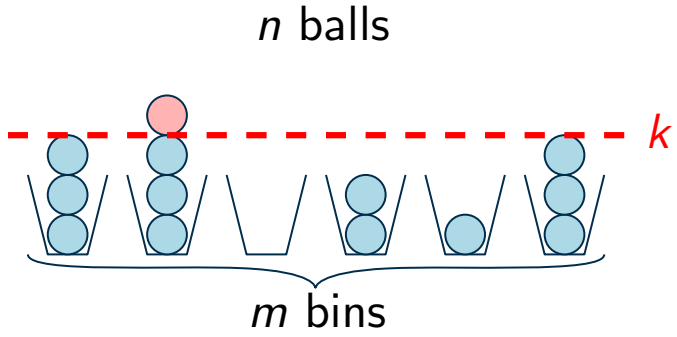
Auxiliary Setting

Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\Pr[\hat{X} \leq k \wedge S = n]$$

Poissonisation with a Twist



Problem

$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

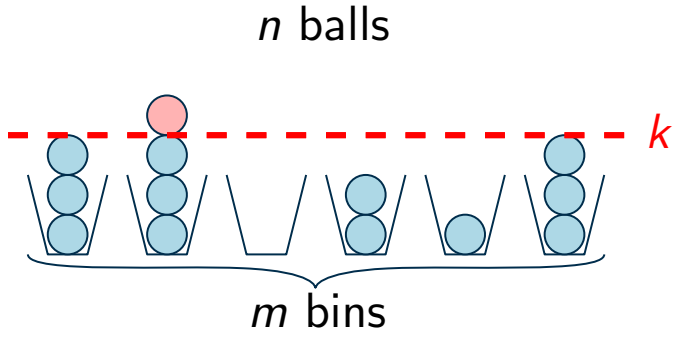
Auxiliary Setting

Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$

Poissonisation with a Twist



Problem

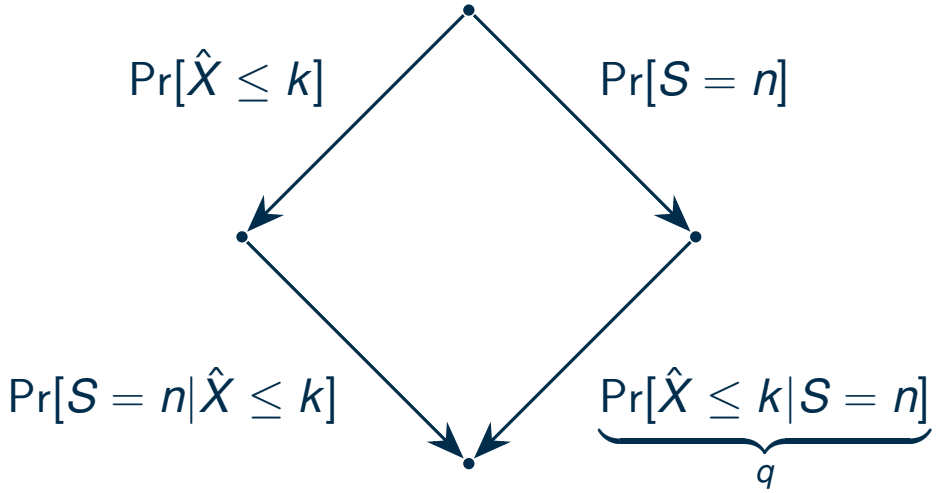
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

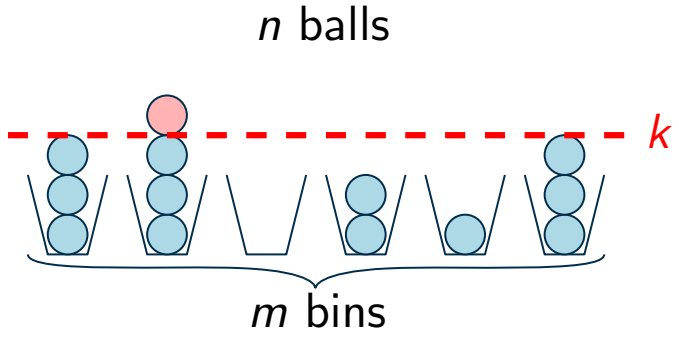
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

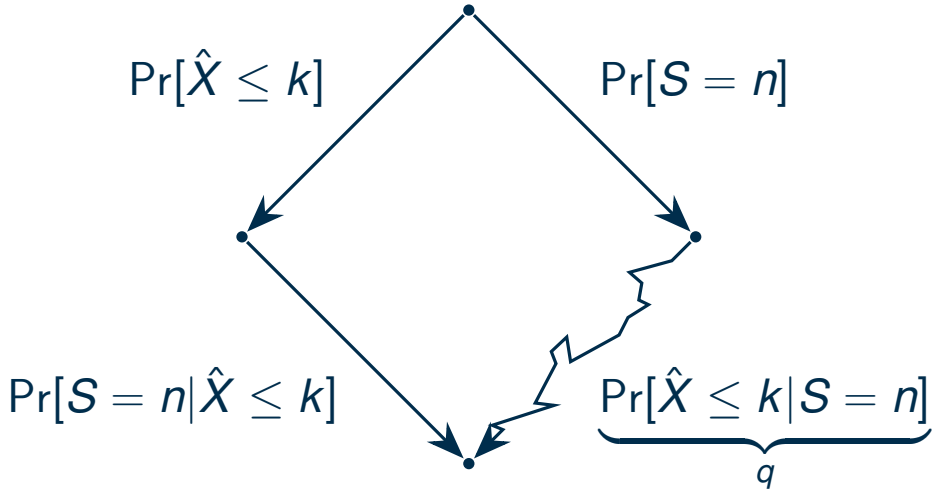
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

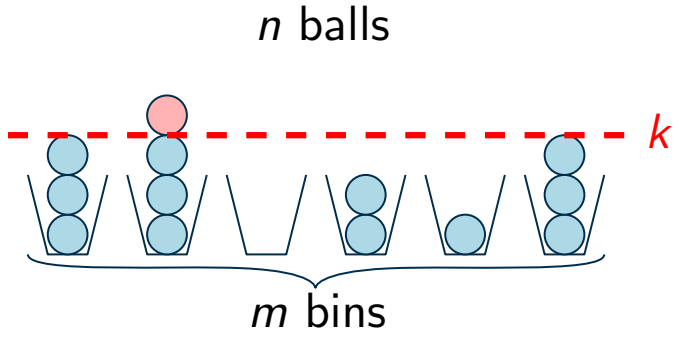
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

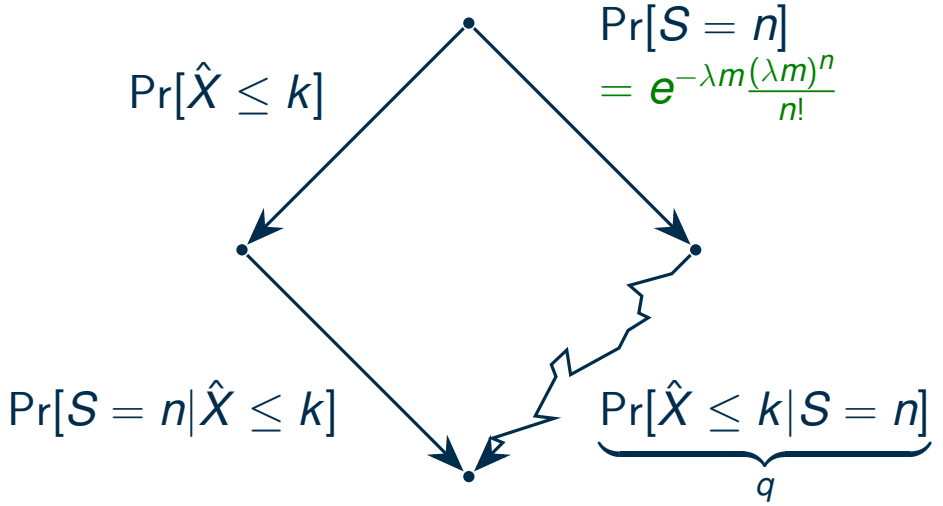
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

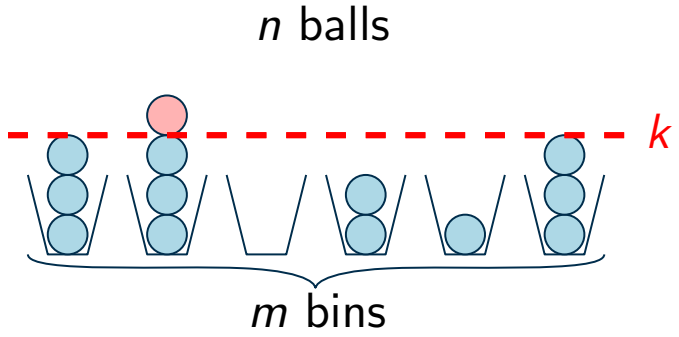
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

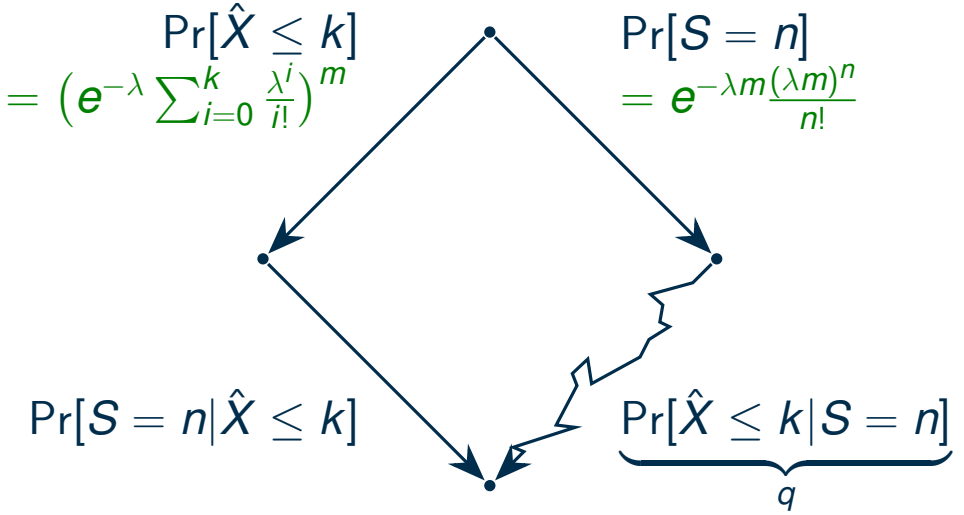
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

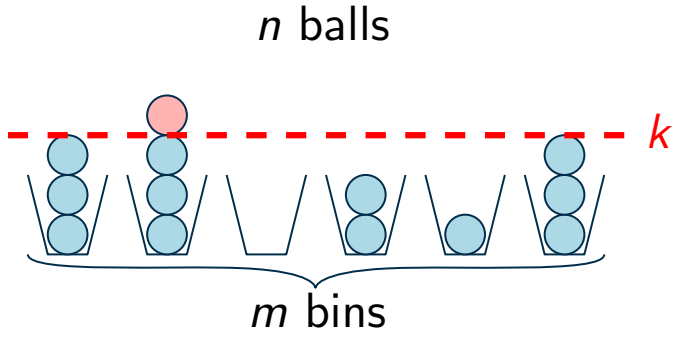
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

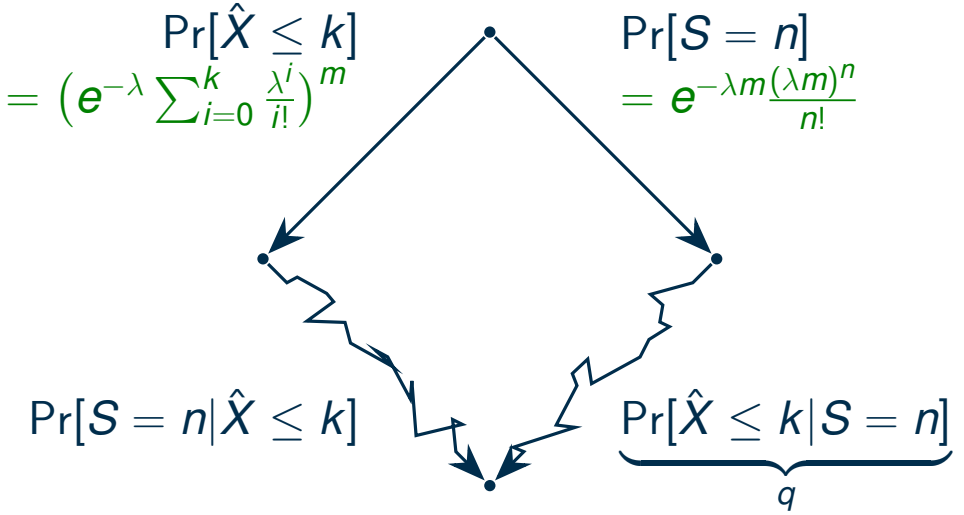
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

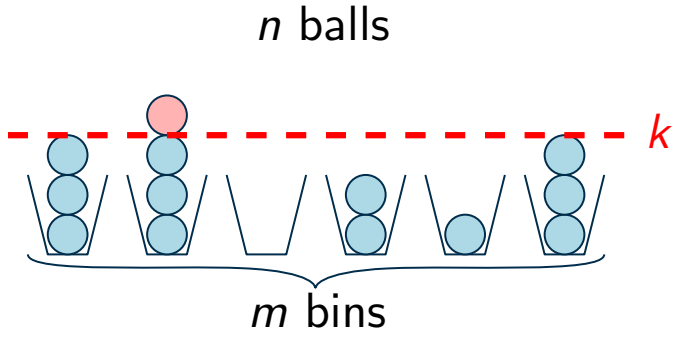
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

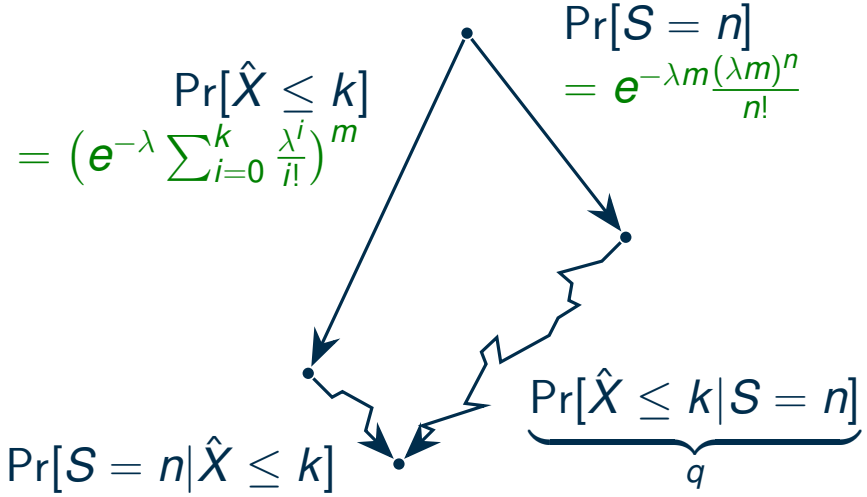
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

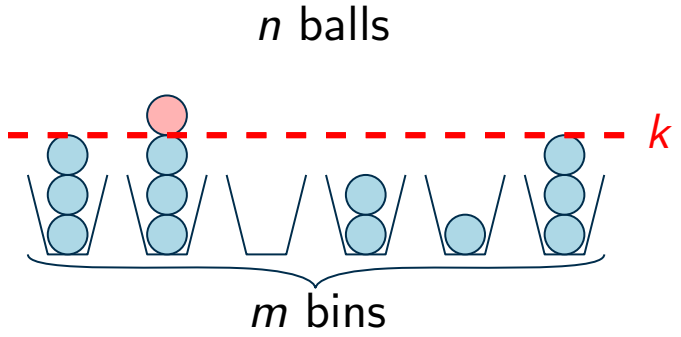
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

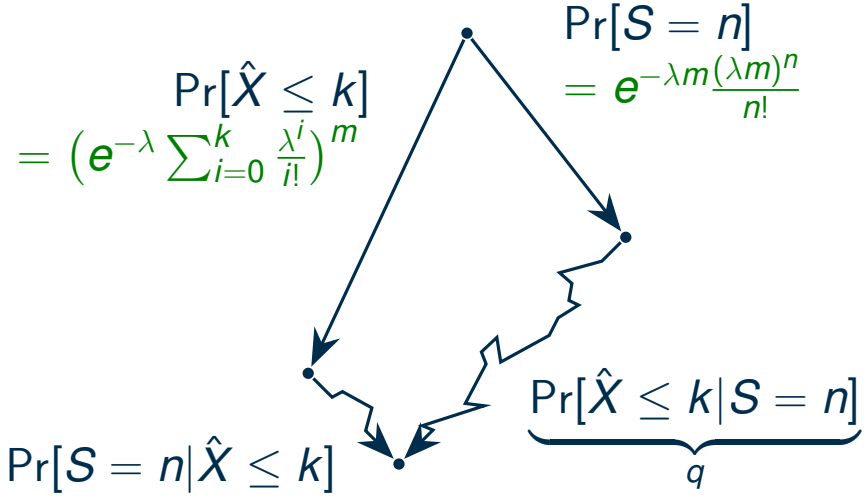
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

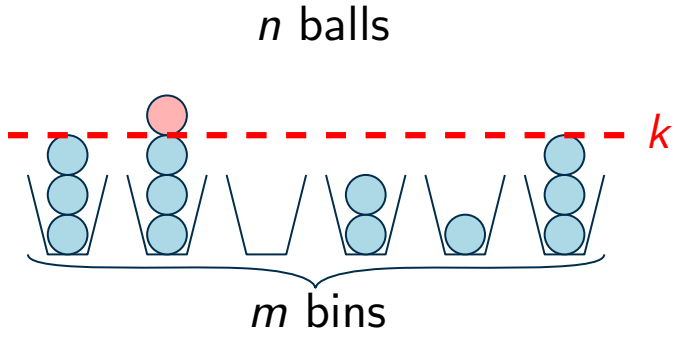
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- λ such that $\mathbb{E}_{X \sim \text{Poisson}(\lambda)}[X \mid X \leq k] = \frac{n}{m}$
- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



Poissonisation with a Twist



Problem

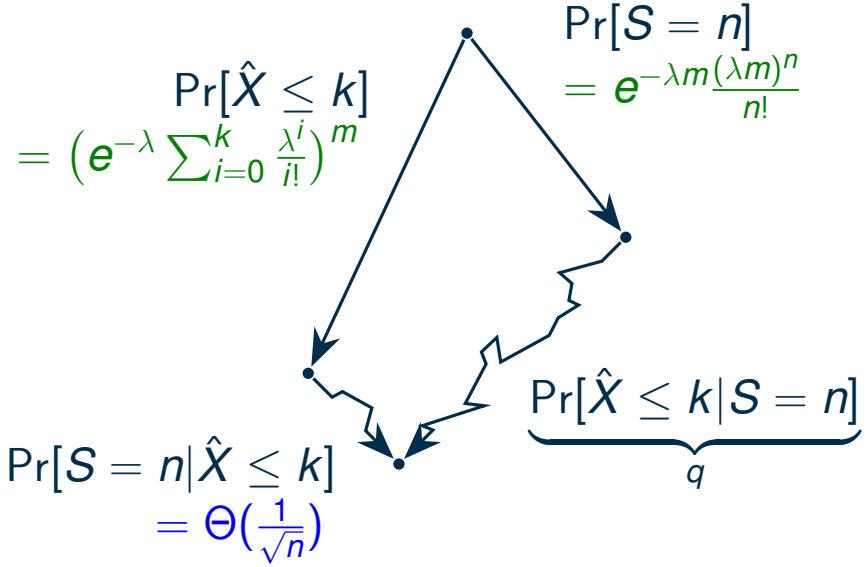
$q := \Pr[\text{all bin loads} \leq k] = ?$ // goal approximate $\log_2 \frac{1}{q}$

Auxiliary Setting

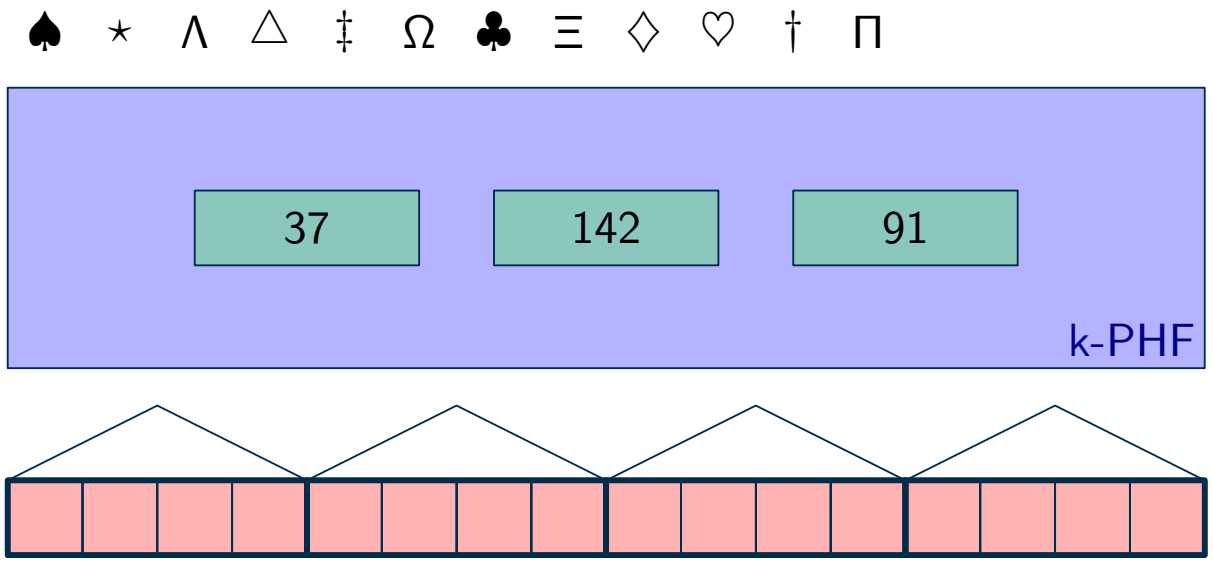
Consider bin loads $X_1, \ldots, X_m \sim \text{Poisson}(\lambda)$

- λ such that $\mathbb{E}_{X \sim \text{Poisson}(\lambda)}[X \mid X \leq k] = \frac{n}{m}$
- $\hat{X} := \max\{X_1, \ldots, X_m\}$
- $S := X_1 + \ldots + X_m$

$$\begin{aligned}\Pr[\hat{X} \leq k \wedge S = n] &= \Pr[S = n | \hat{X} \leq k] \cdot \Pr[\hat{X} \leq k] \\ &= \underbrace{\Pr[\hat{X} \leq k | S = n]}_{q} \cdot \Pr[S = n].\end{aligned}$$



k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.

Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

k-PHF-set

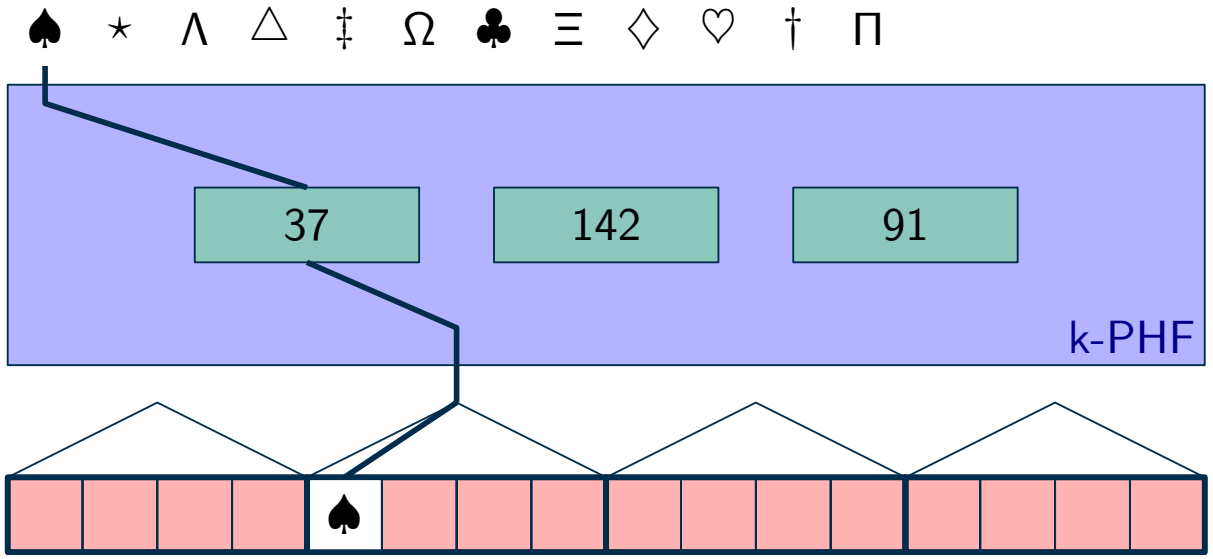
k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.

1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.

Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

k-PHF-set

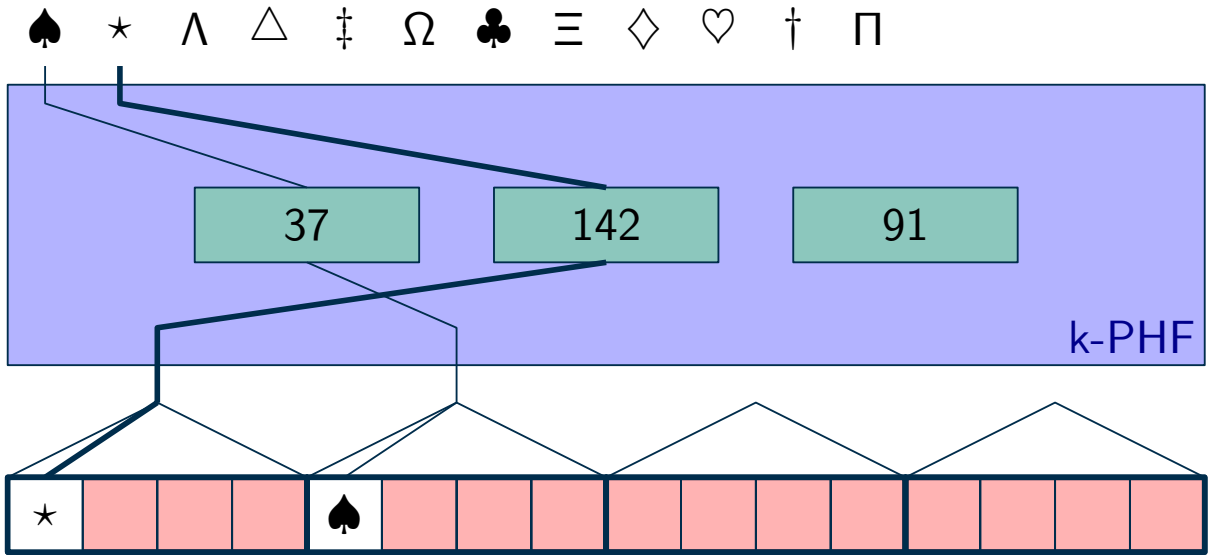
k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.

1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.
Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

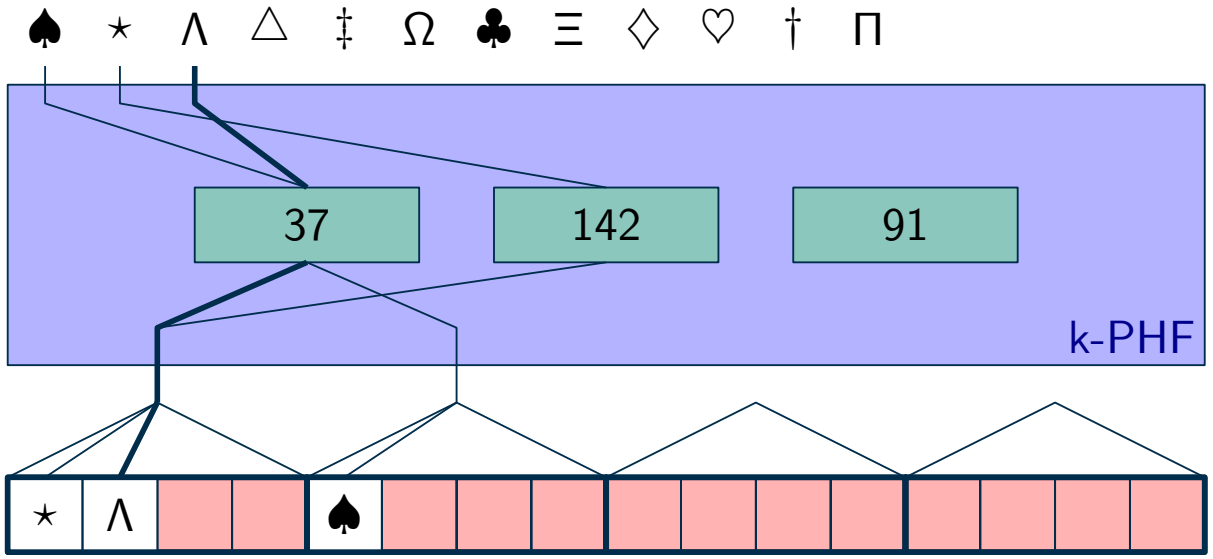
k-PHF-set

k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.
1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.
Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

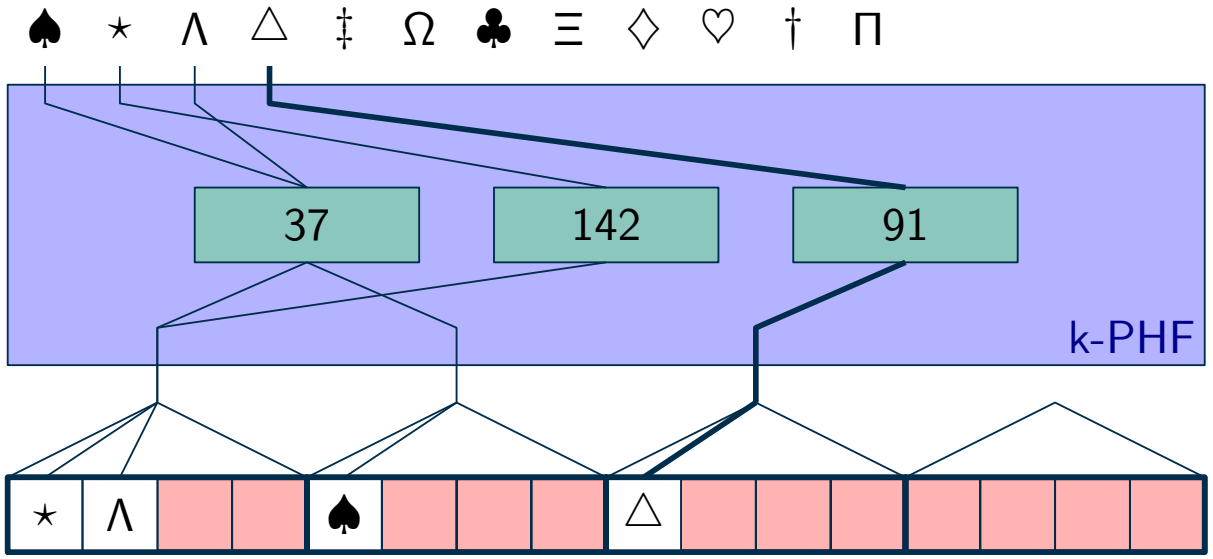
k-PHF-set

k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.
1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.
Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

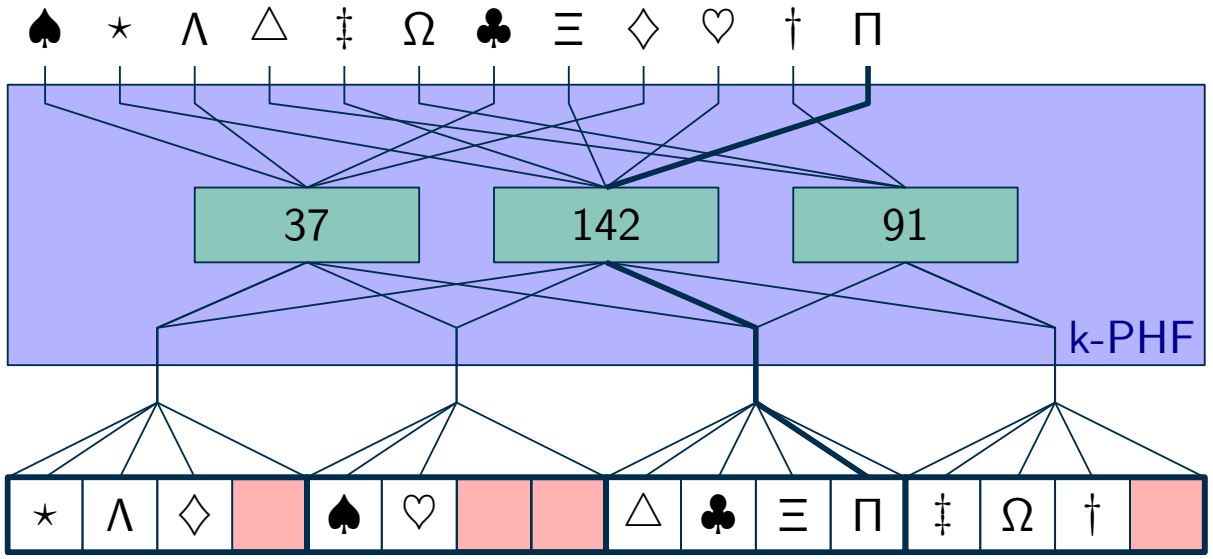
k-PHF-set

k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.
1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.

Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

k-PHF-set

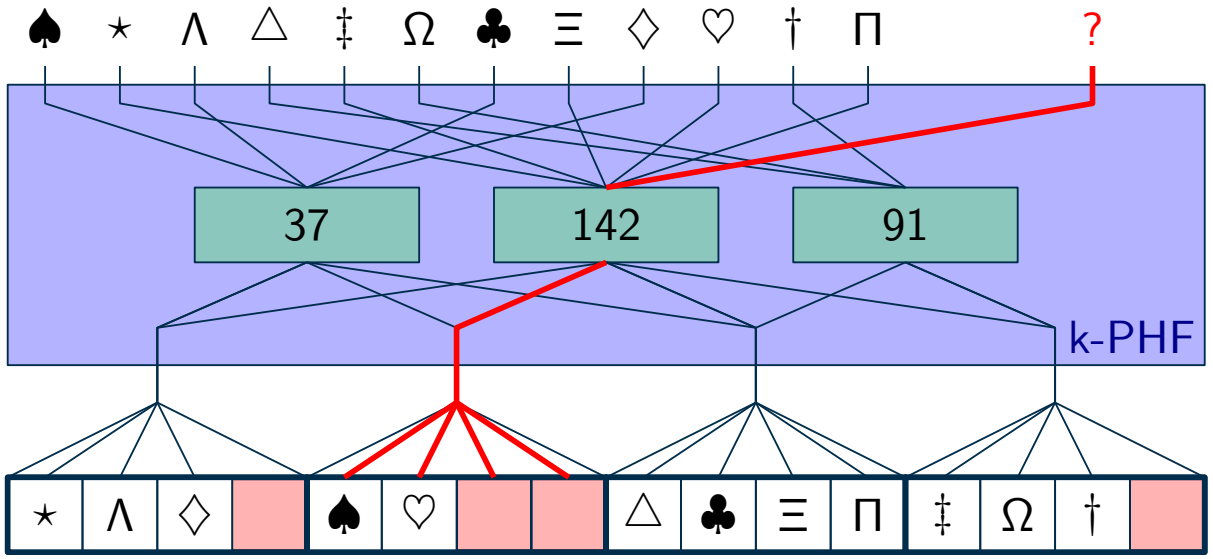
k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.

1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash & k-PHF-set



k-PtrHash data structure

An array of 8-bit seeds.
Target size: 1.5× the space lower bound.

k-PtrHash query

Hash the key to a seed. Then hash (key, seed) to a bin.

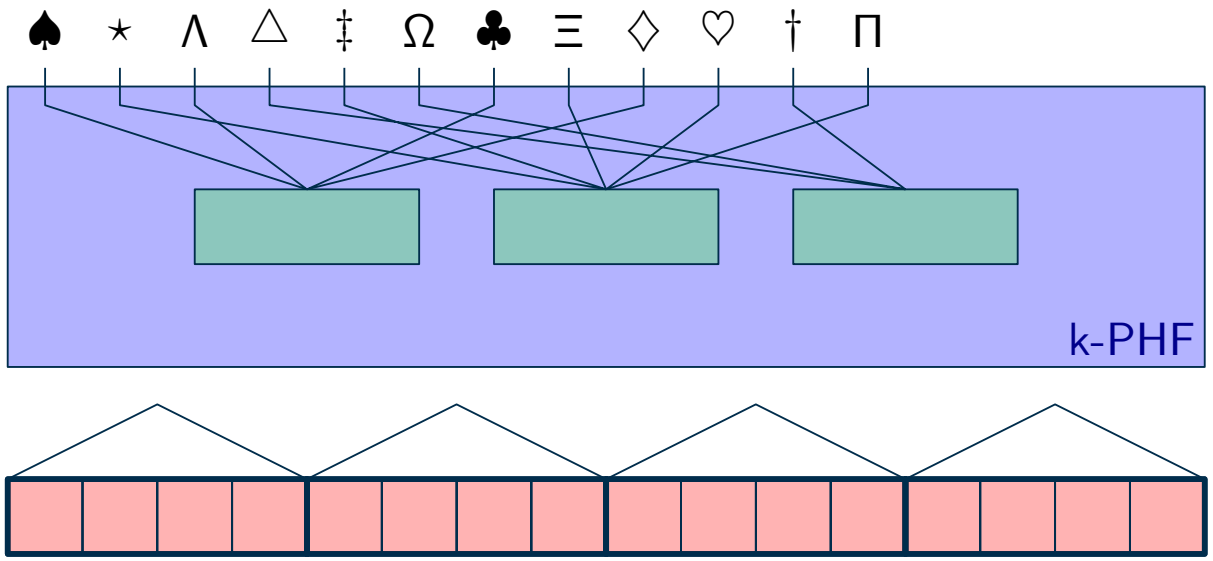
k-PHF-set

k-PHF + bins filled with keys.

k-PHF-set query

Check if the key is in the cache line indicated by the k-PHF using SIMD.
1 cache-miss if the k-PHF fits in cache; great for throughput when memory-bound.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

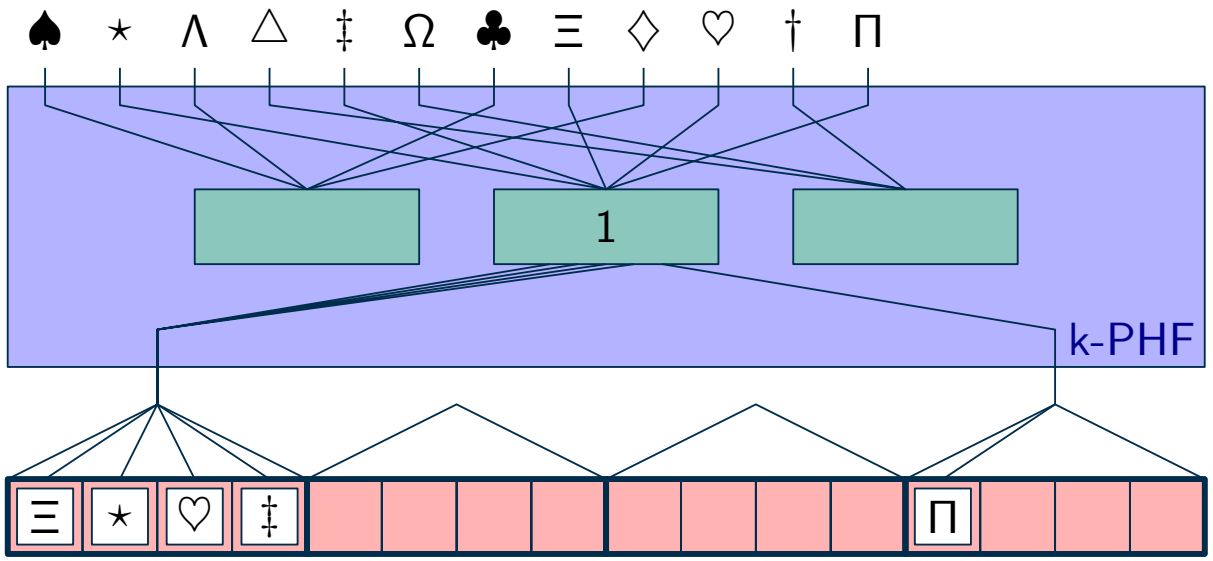
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the “best” seed $< 2^8$.

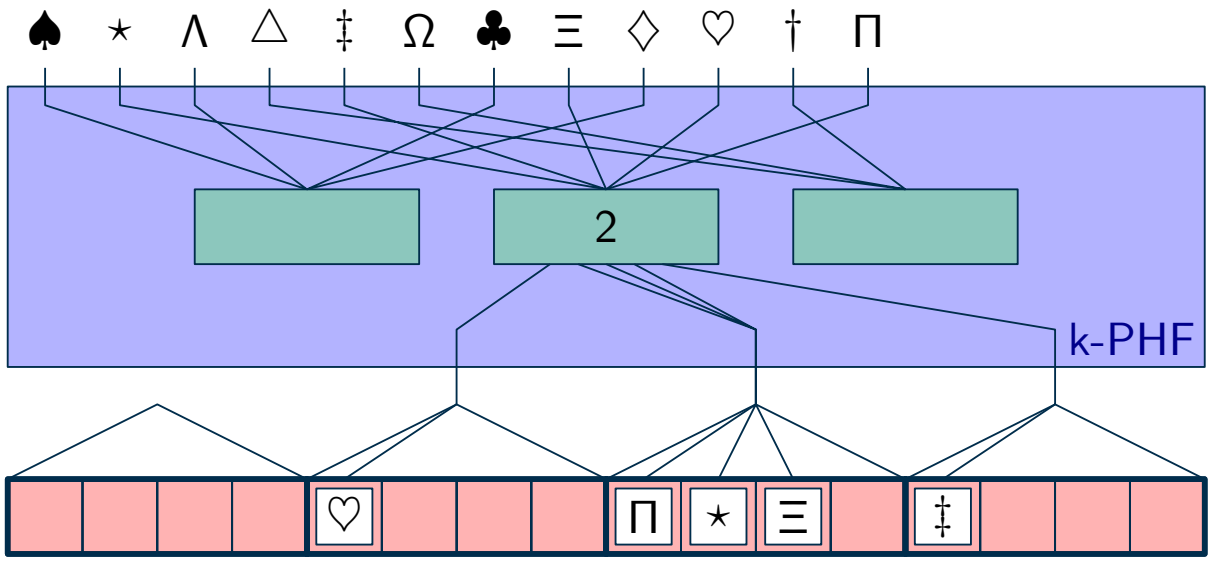
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the “best” seed $< 2^8$.

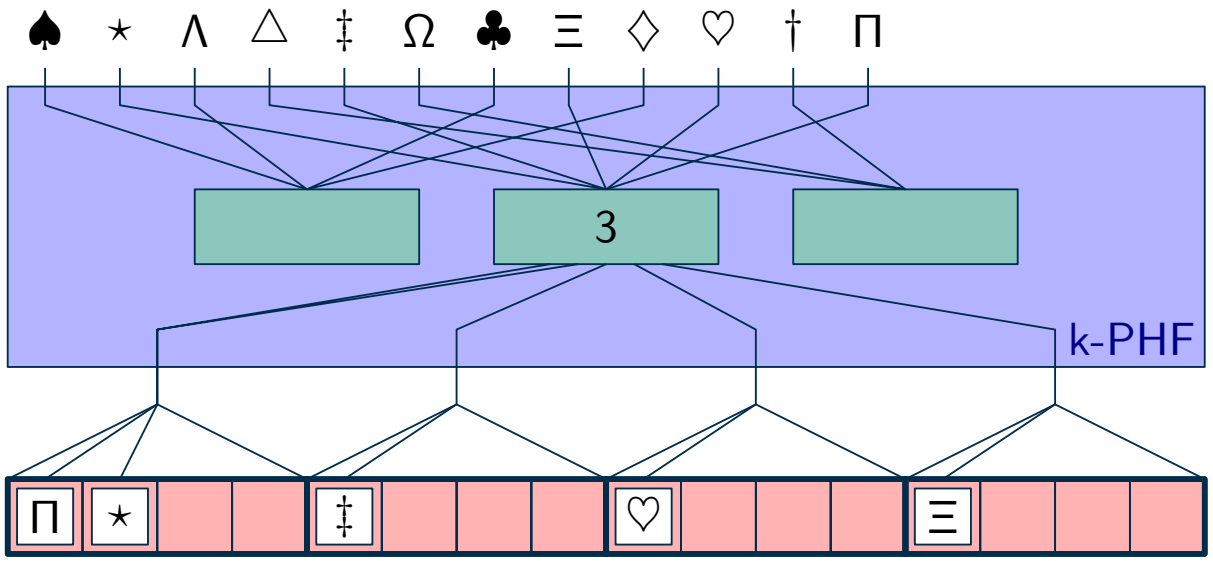
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

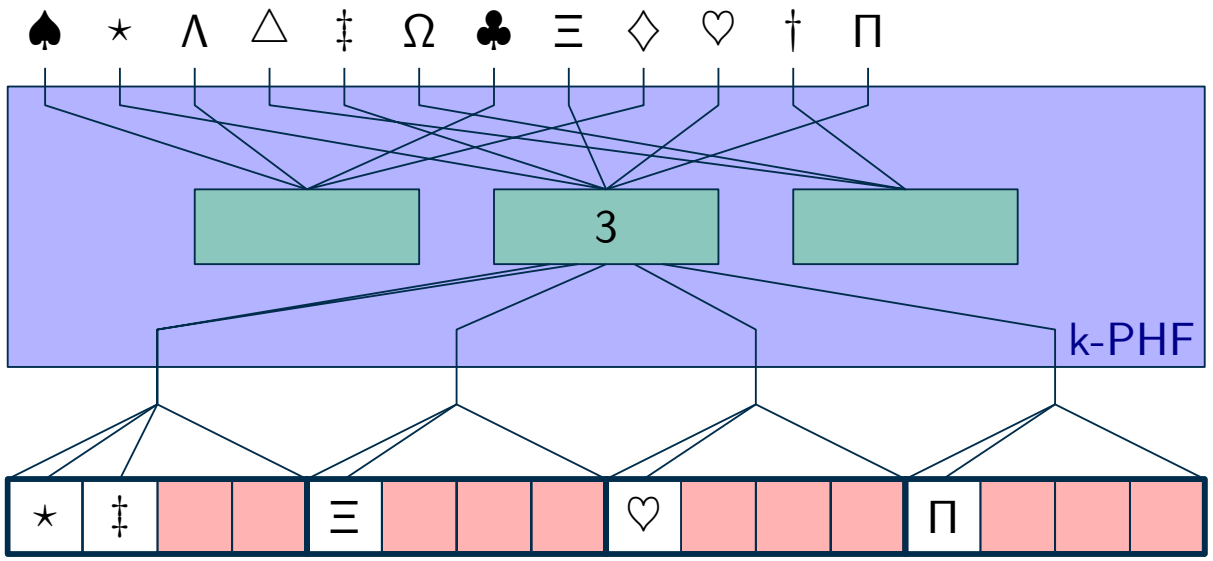
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

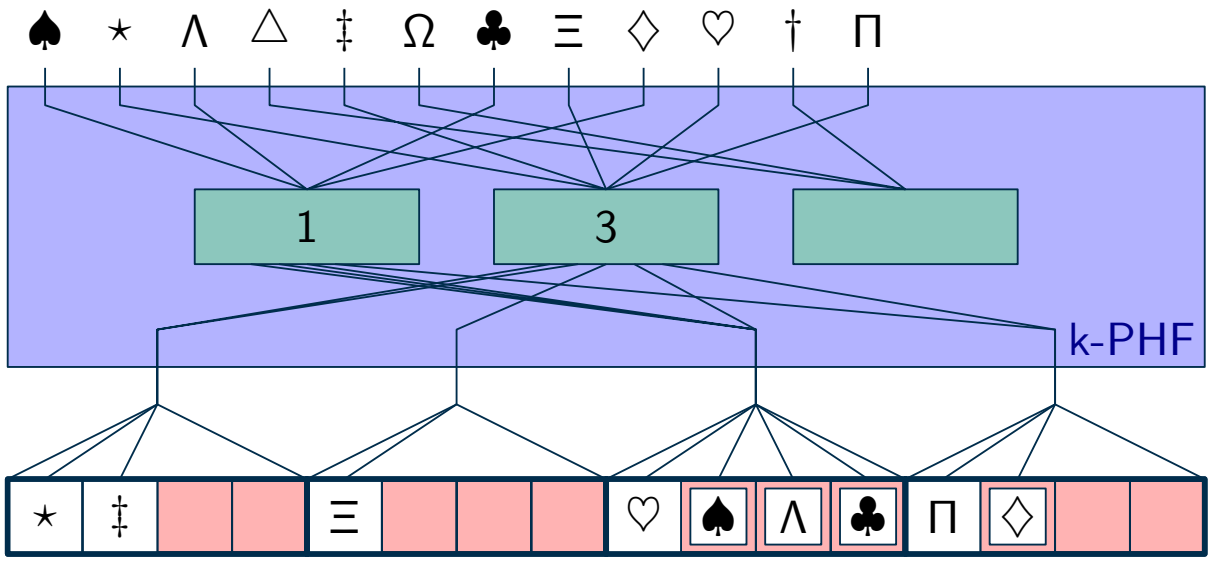
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

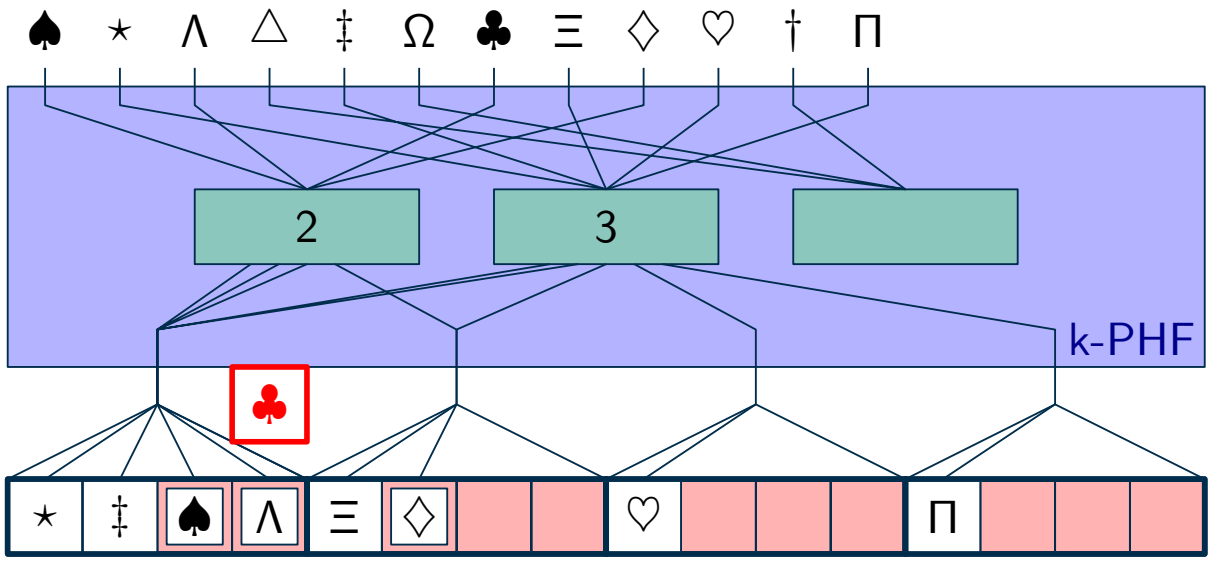
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

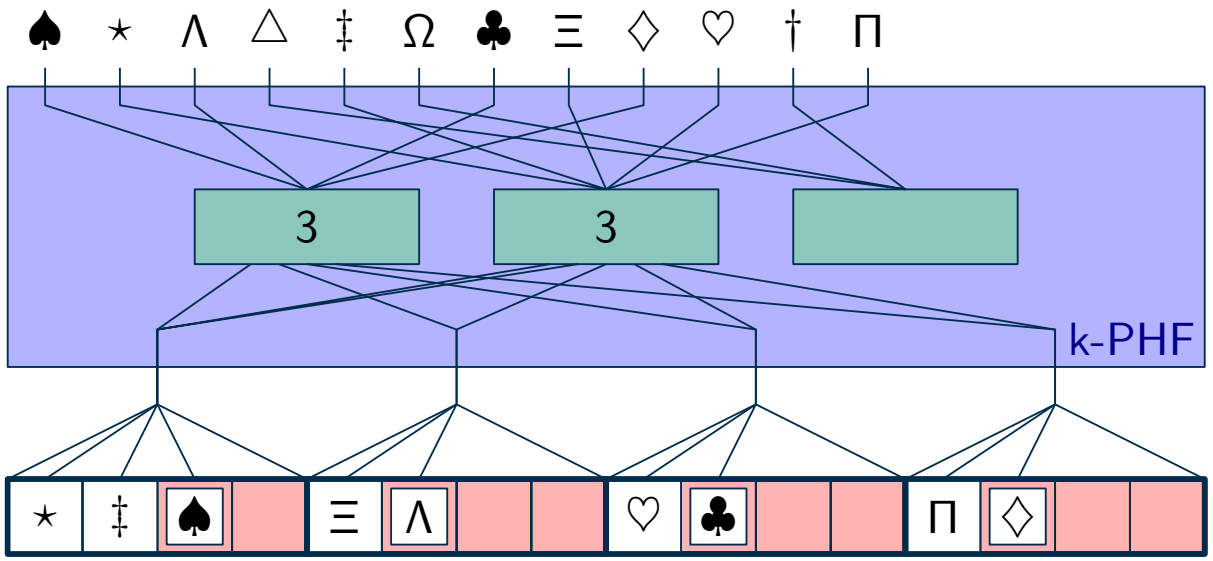
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

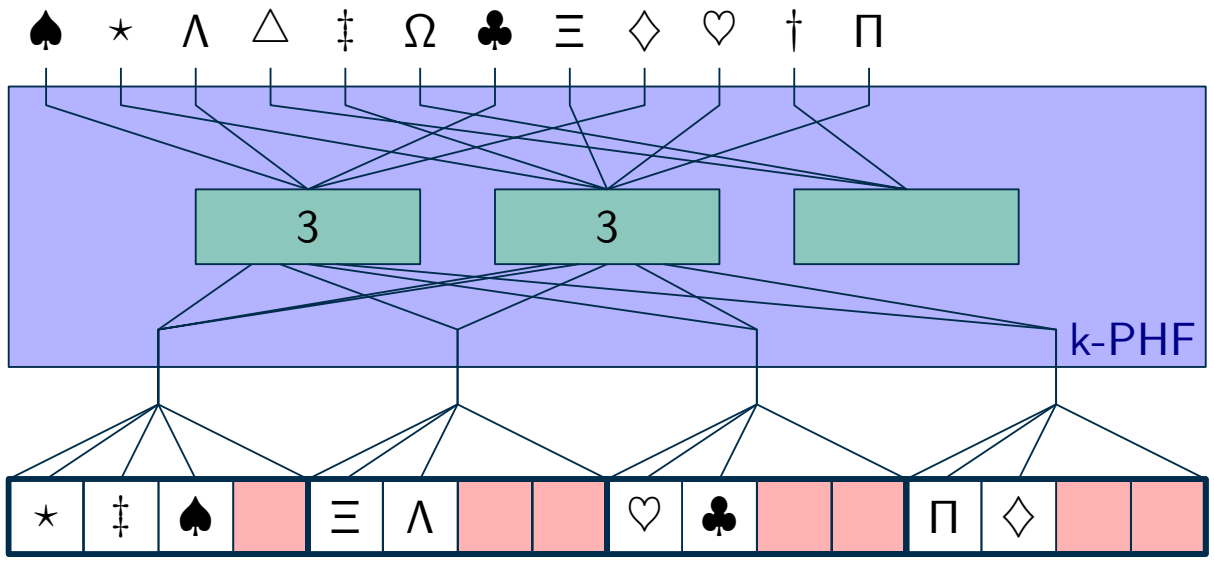
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

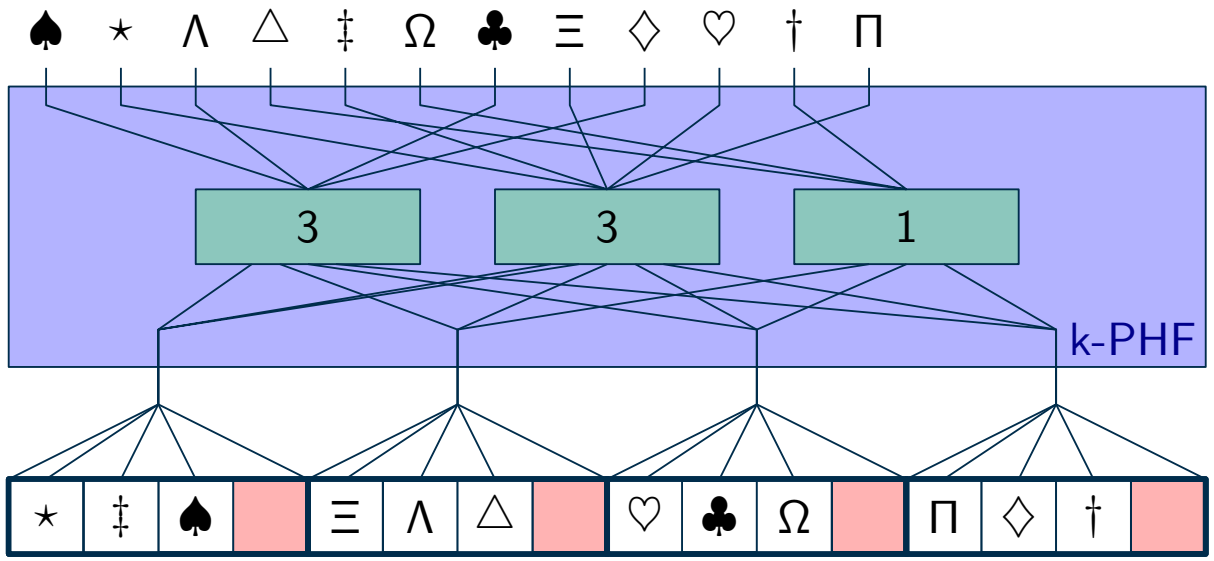
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

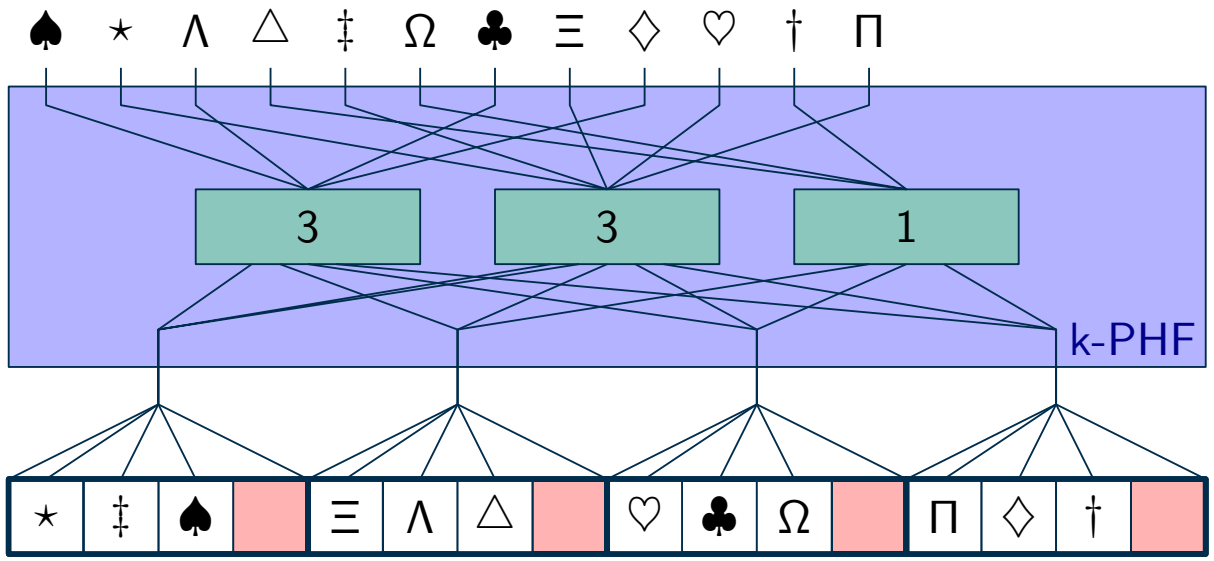
Best seed

Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PtrHash construction: find good seeds



Order of buckets

Like PTHash, PHOBIC, PtrHash: from large to small.

Seed value

Like PHAST: choose the "best" seed $< 2^8$.

Best seed

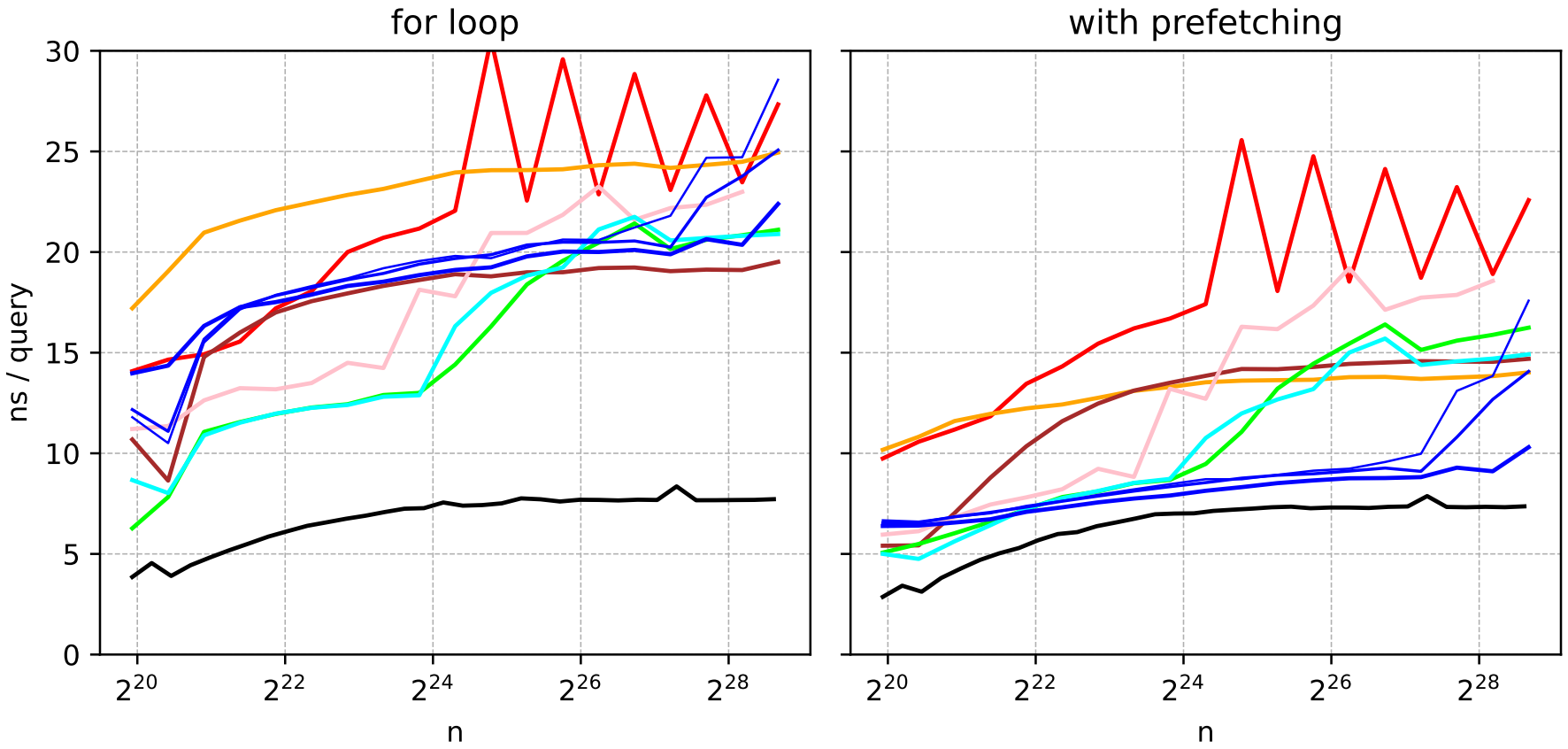
Heuristic: minimize $\sum_i 2^{\text{load of bin } i}$.

Bumping

If there is no collision-free seed, use seed 0 to *bump* keys to a fallback structure. Usually $< 1\%$ of keys.

k-PHF-set: near-optimal with prefetching

Hashset<u64> query throughput (50% positive queries, Intel i7-10750H laptop, 1 thread)



Probing
- HashSet ($0.44 \leq \alpha \leq 0.88$)
- CuckooSet<Lazy> ($\alpha = 0.7$)
- CuckooSet<Eager> ($\alpha = 0.99$)

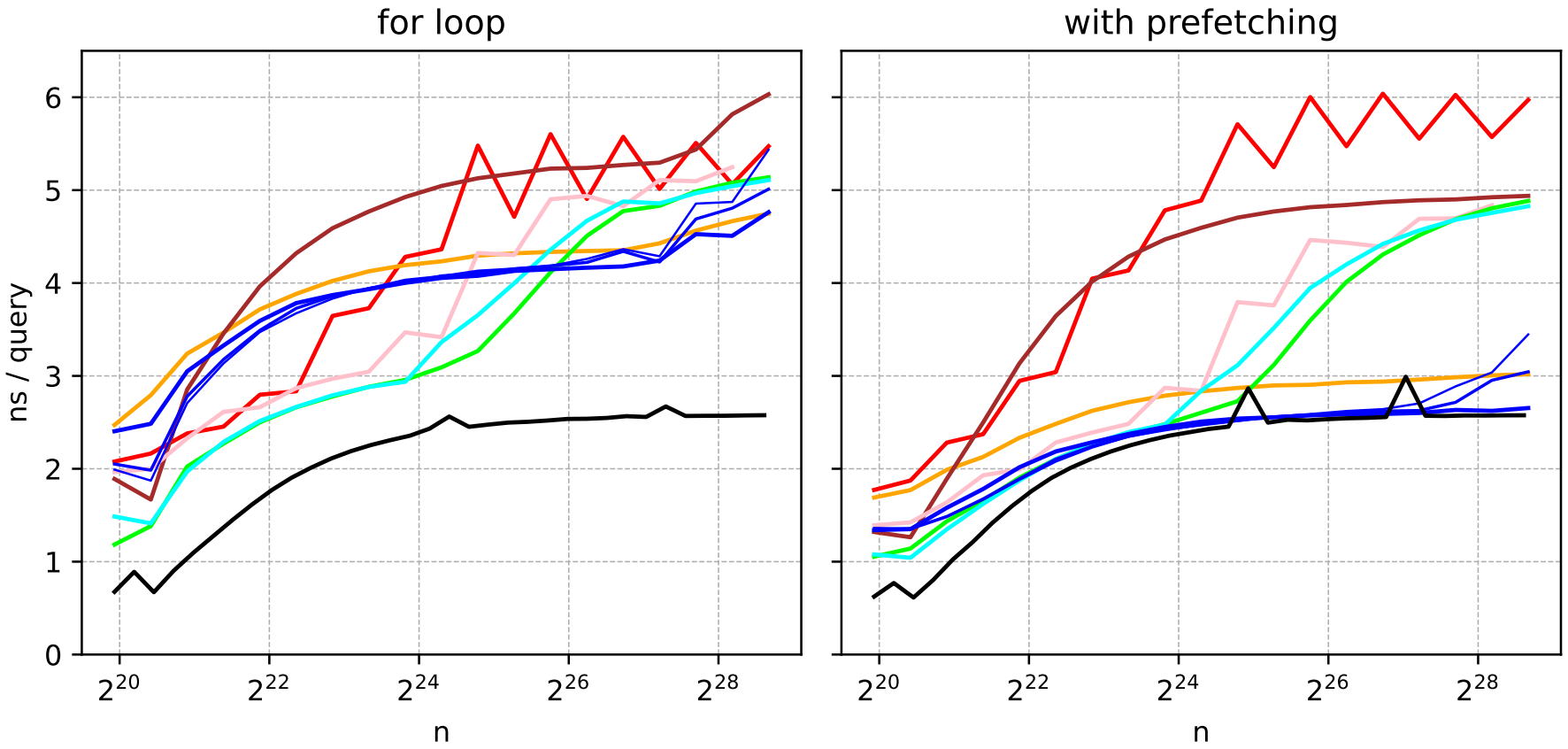
1-PHF-set
- FPH ($\alpha = 0.95$)
- PHF-set<PHast+> ($\alpha = 0.98$)
- PHF-set<PtrHash> ($\alpha = 0.99$)

***k*-PHF-set**
- *k*-PHF-set<*k*-PtrHash> ($\alpha = 0.95$)
- *k*-PHF-set<*k*-PtrHash> ($\alpha = 0.9$)
- *k*-PHF-set<*k*-PtrHash> ($\alpha = 0.7$)

Lower bound
- MockHashSet ($\alpha = 1.0$)

k-PHF-set: near-optimal with prefetching (12 threads)

Hashset<u64> query throughput (50% positive queries, Intel i7-10750H laptop, 12 threads)



Probing
- HashSet ($0.44 \leq \alpha \leq 0.88$)
- CuckooSet<Lazy> ($\alpha = 0.7$)
- CuckooSet<Eager> ($\alpha = 0.99$)

1-PHF-set
- FPH ($\alpha = 0.95$)
- PHF-set<PHast+> ($\alpha = 0.98$)
- PHF-set<PtrHash> ($\alpha = 0.99$)

k-PHF-set
- k-PHF-set<k-PtrHash> ($\alpha = 0.95$)
- k-PHF-set<k-PtrHash> ($\alpha = 0.9$)
- k-PHF-set<k-PtrHash> ($\alpha = 0.7$)

Lower bound
- MockHashSet ($\alpha = 1.0$)

Fin.

Theory
Space lower bound for non-minimal k-PHF.

k-PtrHash
A simple & fast k-PHF.

k-PHF-set
A static hashset using k-PtrHash.

Result
Up to 2× higher query throughput.

k-PtrHash results

k	Load factor α target	Load factor α final	Space (bits/key) target	Space (bits/key) final	Bumped (%)	Build time (ns/key)	Query time (ns/key)
4	0.70	≈ 0.700	0.204	0.204	0.001	191	3.2
4	0.80	≈ 0.800	0.316	0.316	0.001	190	3.4
4	0.90	≈ 0.900	0.489	0.490	0.005	206	3.7
4	0.99	0.980	0.800	0.808	0.53	277	3.9
8	0.70	≈ 0.700	0.065	0.065	0.0002	398	2.6
8	0.80	≈ 0.800	0.124	0.124	0.0001	473	3.0
8	0.90	≈ 0.900	0.231	0.231	0.002	450	3.2
8	0.99	0.976	0.460	0.467	0.72	500	3.6
16	0.70	≈ 0.700	0.015	0.015	0.13	450	2.5
16	0.80	0.798	0.040	0.041	0.11	480	2.6
16	0.90	0.896	0.097	0.098	0.23	443	2.9
16	0.99	0.968	0.252	0.258	1.12	499	3.4

Results of k-PtrHash on $n = 10^8$ keys, targeting $1.5\times$ the space lower bound. *Final* load factor and space usage take into account the additional space for bumping. Timing are single threaded, and the query time is measured as the average time per iteration of a for loop.

k-PtrHash results