Modern hashing for alignment-free sequence analysis

Part 3: Multi-way bucketed cuckoo hashing for DNA *k*-mers

Jens Zentgraf & Sven Rahmann

GCB 2021

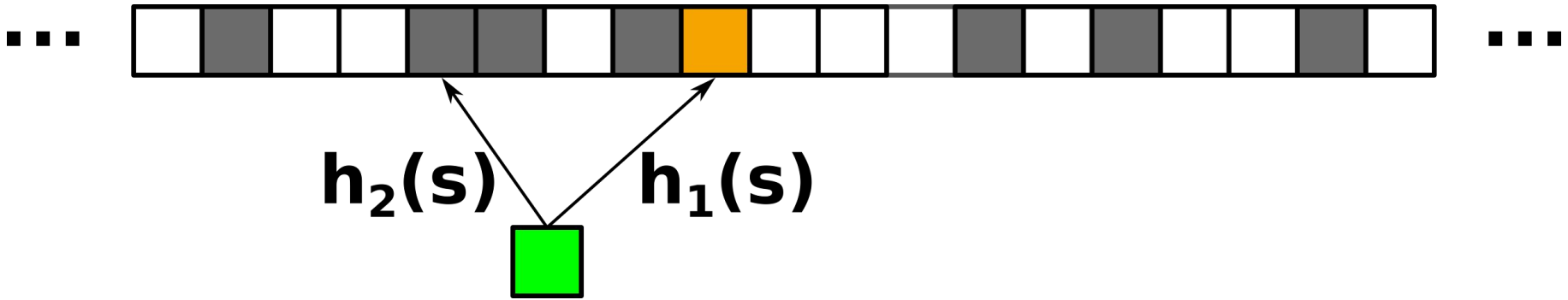
Classical Cuckoo hashing

- Two hash functions h_1 and h_2
- Insert new element x with h_1
  - If position is occupied with element *y*, displace element *y*
  - Insert y with alternative hash function
- Parameter *w*: Maximum number of displacements

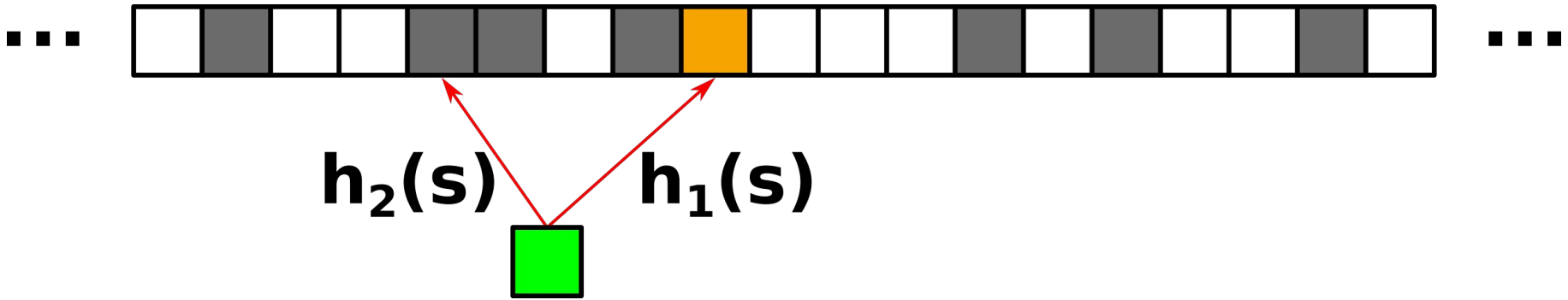
- Insert: O(*w*)
- Lookup: $2 \in O(1)$

Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. J. Algorithms, 51(2):122–144, 2004.

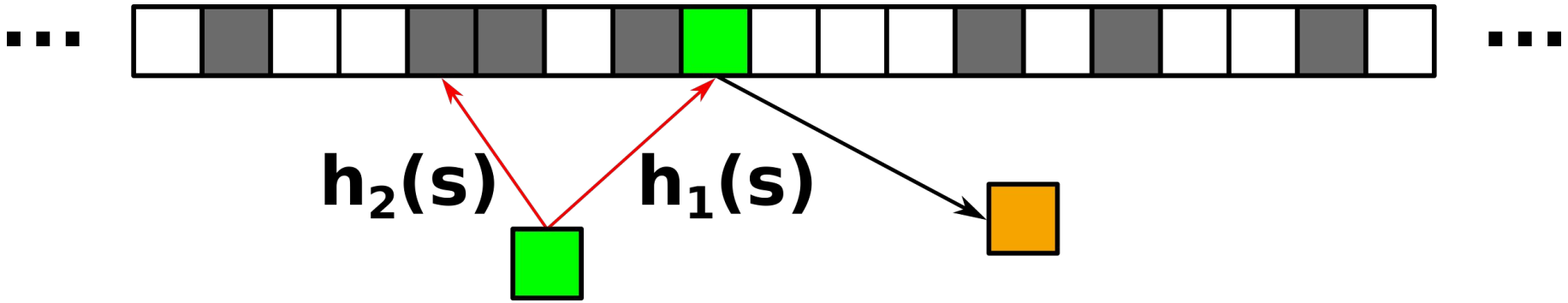
Cuckoo hashing



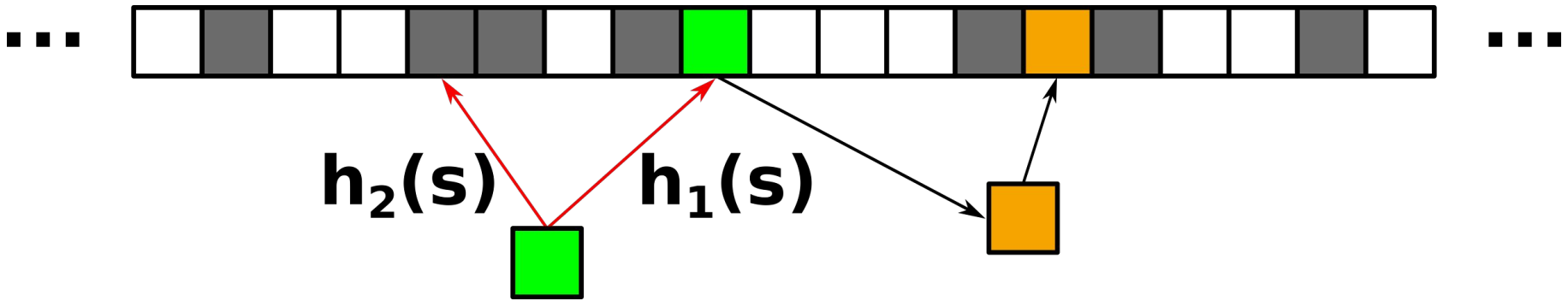
Cuckoo hashing



Cuckoo hashing

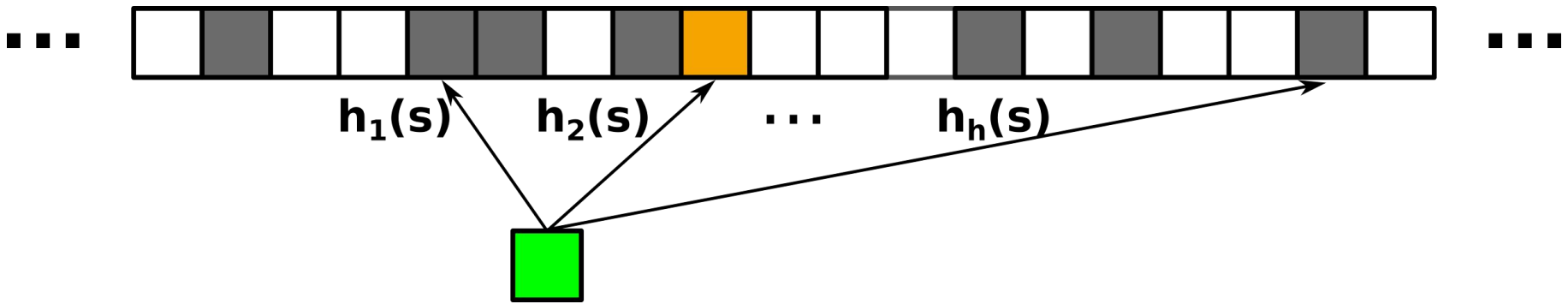


Cuckoo hashing



Cuckoo hashing with *h* hash functions

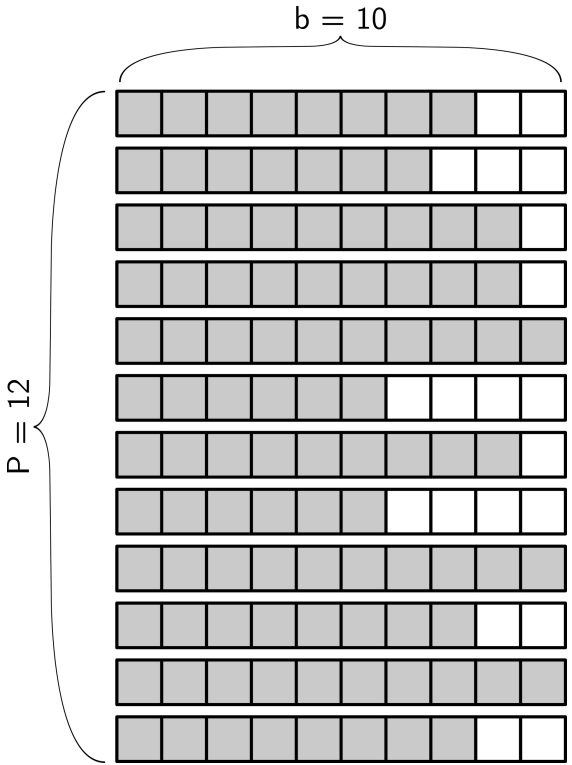
- Use $h \geq 2$ hash functions
- More choices:
  Better (more uniform) distribution of the elements



Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. Theory Comput. Syst., 38(2):229–248, 2005.

Cuckoo hashing with buckets

- Each position can store up to *b* elements ("**buckets**", "pages", "bins" of size *b*)
- Example: *P* = 12 buckets of size *b* = 10: hash table with 120 slots.
- Must search bucket to find element *x*, until found or empty slot encountered, or entire bucket has been searched



Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. Theor. Comput. Sci., 380(1-2):47–68, 2007.

(h,b) Cuckoo hashing

- Combination of
  - h hash functions
  - buckets with size b

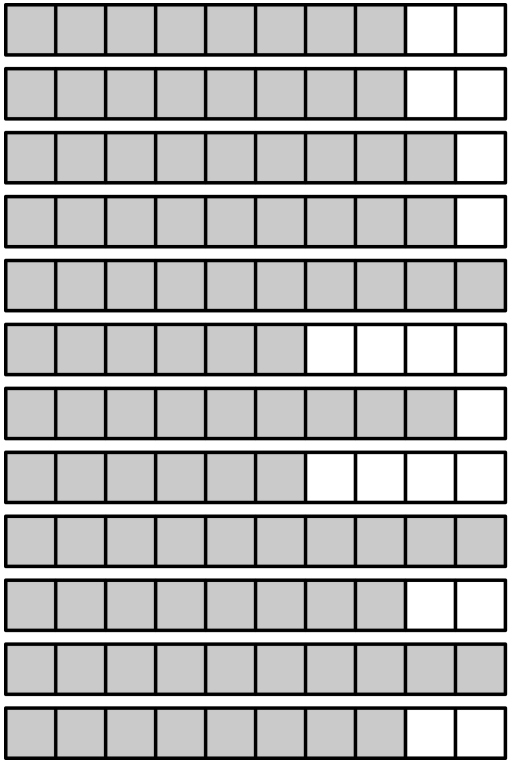
- Different insertion strategies
  - Random walk
  - Breadth first search
  - Solving a (huge, sparse) minimum weighted matching problem
  - LSA_{max}
  - ...

(*h*,*b*) Cuckoo hashing: Random walk

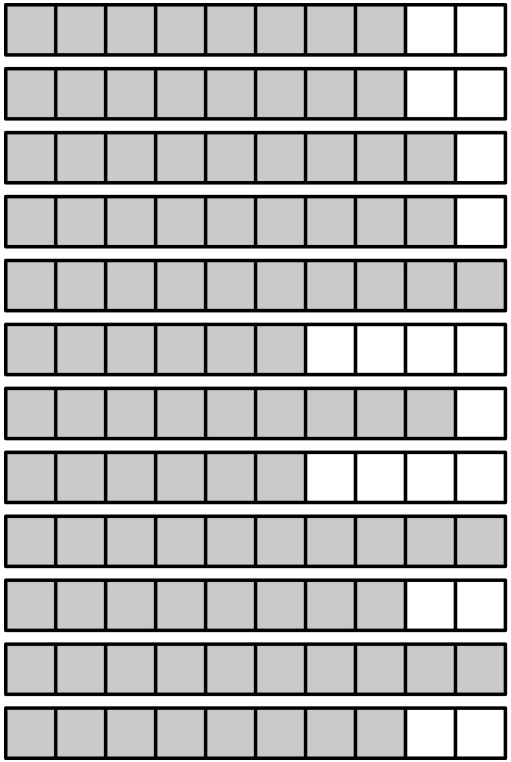
- Check all hash functions for an empty slot
- If no empty slot exits
  - Choose one element and replace it
  - Insert the replaced element
- Maximum of *w* replacements

- Insert: O(*w*) cache misses
- Lookup: O(*h*) cache misses

(h,b) Cuckoo hashing: Random walk

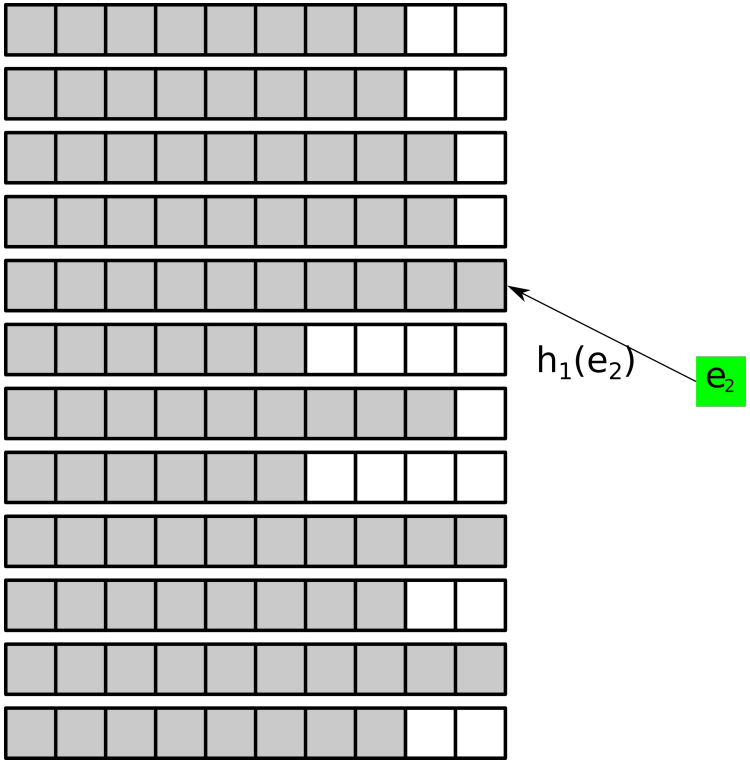


(h,b) Cuckoo hashing: Random walk

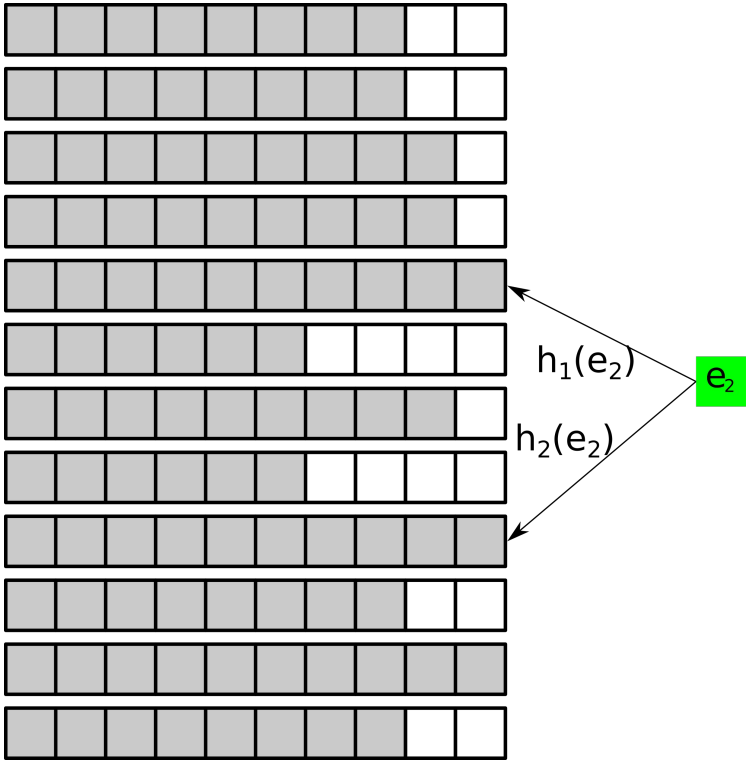


e_2

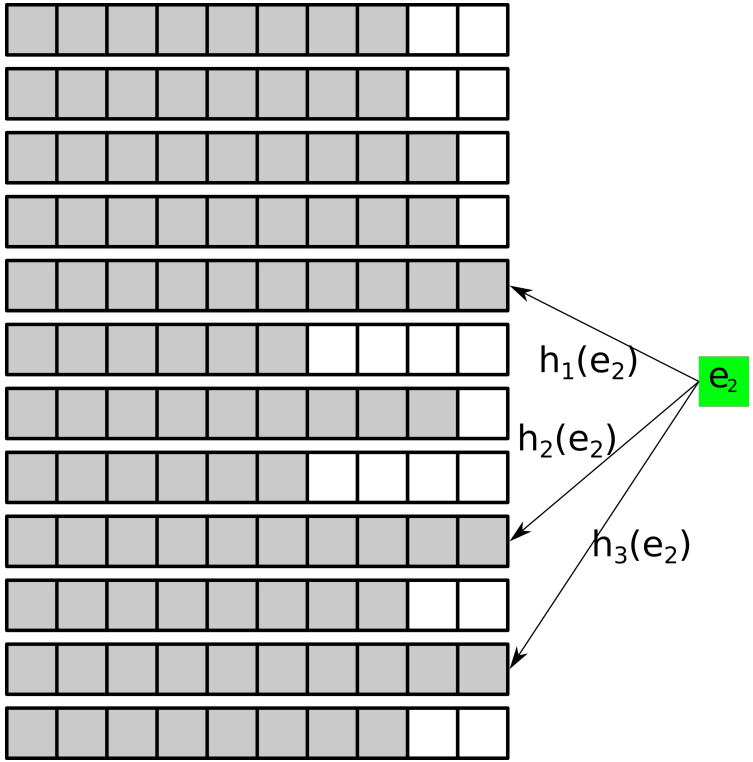
(h,b) Cuckoo hashing: Random walk



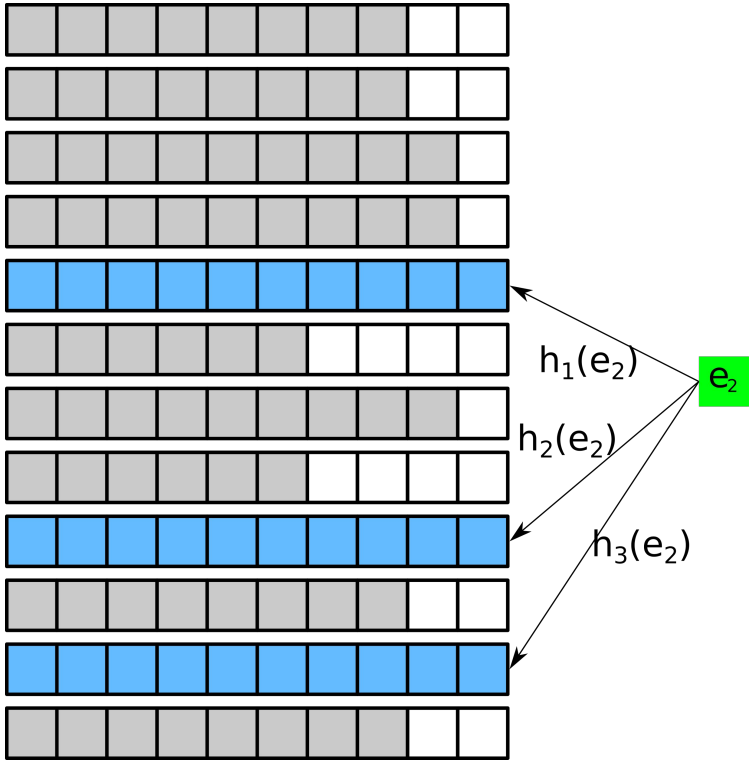
(h,b) Cuckoo hashing: Random walk



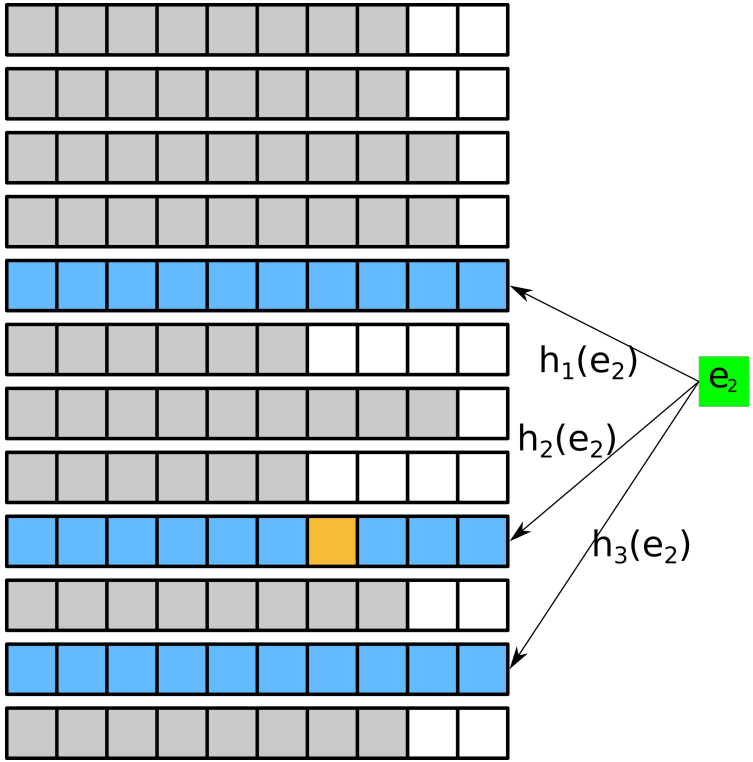
(h,b) Cuckoo hashing: Random walk



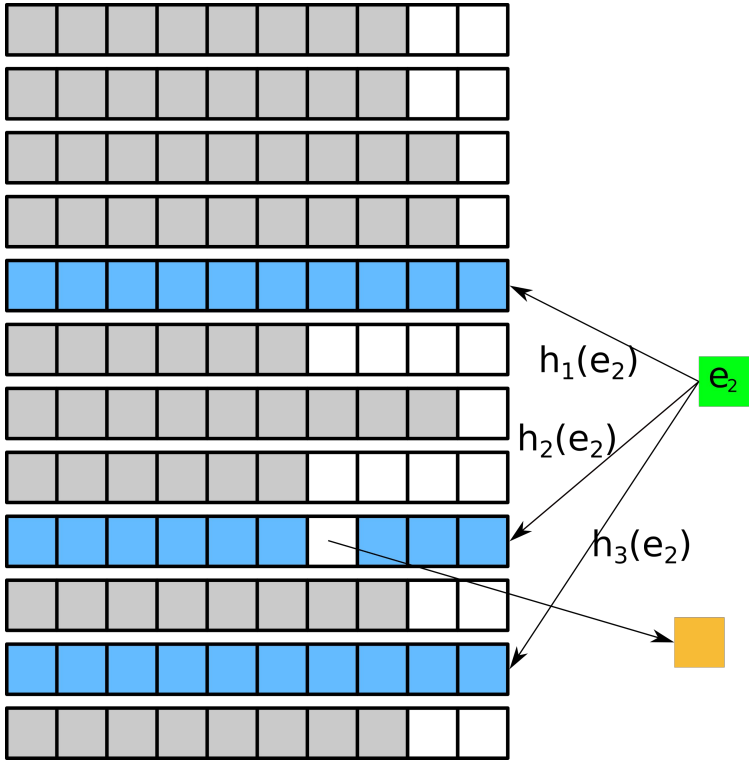
(h,b) Cuckoo hashing: Random walk



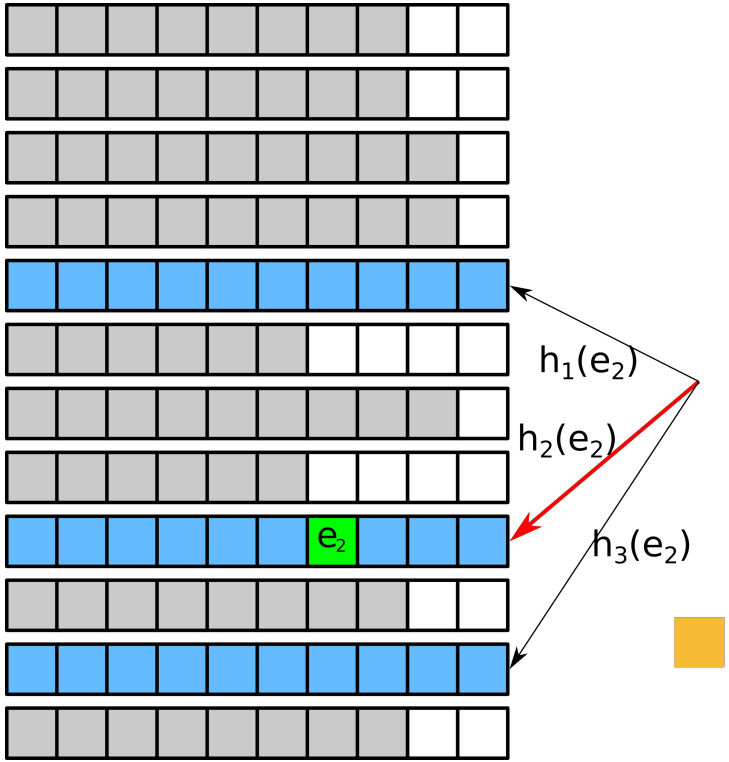
(h,b) Cuckoo hashing: Random walk



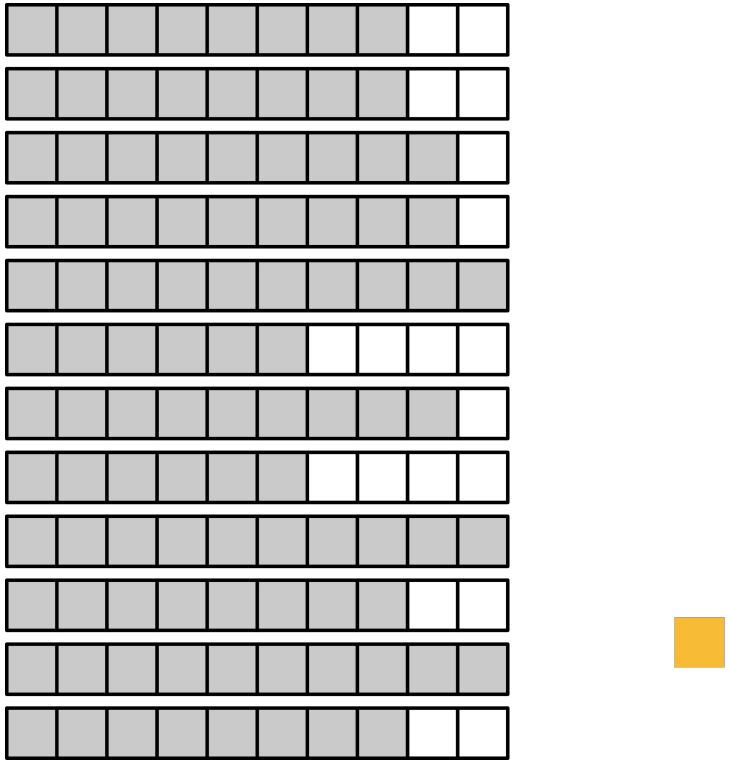
(h,b) Cuckoo hashing: Random walk



(h,b) Cuckoo hashing: Random walk



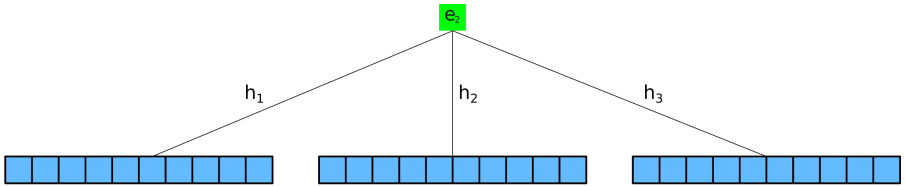
(h,b) Cuckoo hashing: Random walk

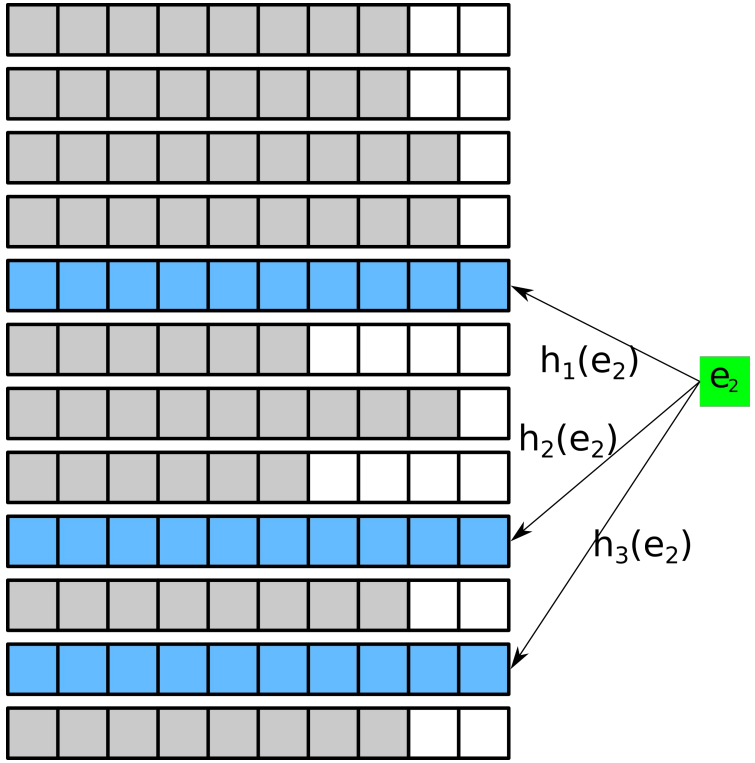


(h,b) Cuckoo hashing: BFS

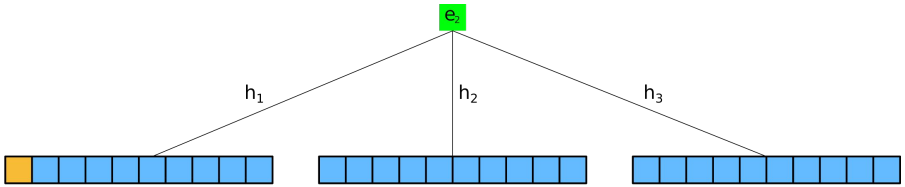
- BFS always finds a path to insert a new element (if one exists)
- BFS starts at the new elements
  - Check all reachable buckets
  - if there exists no empty slot:
    - Follow all elements in these buckets to their alternative locations

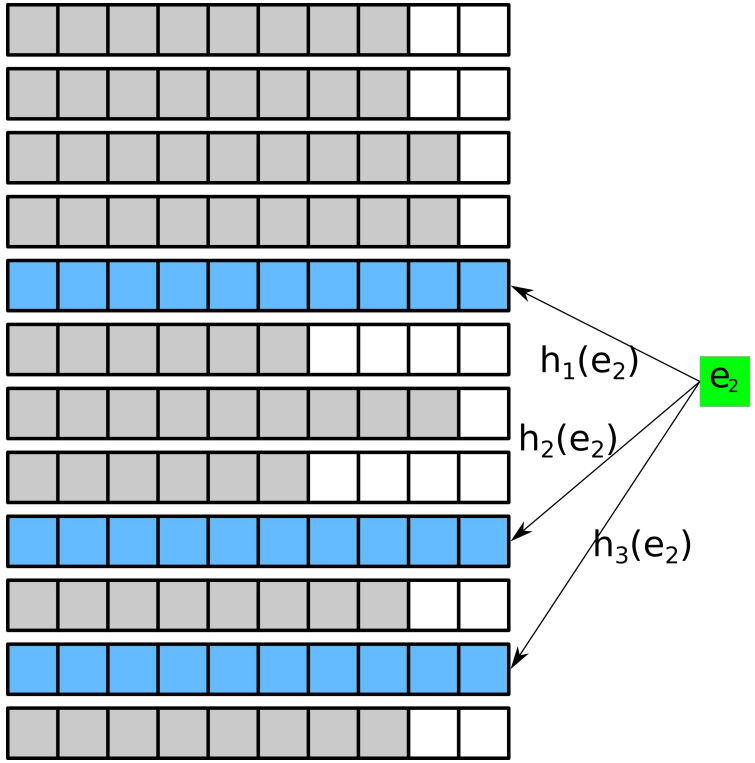
(h,b) Cuckoo hashing: BFS



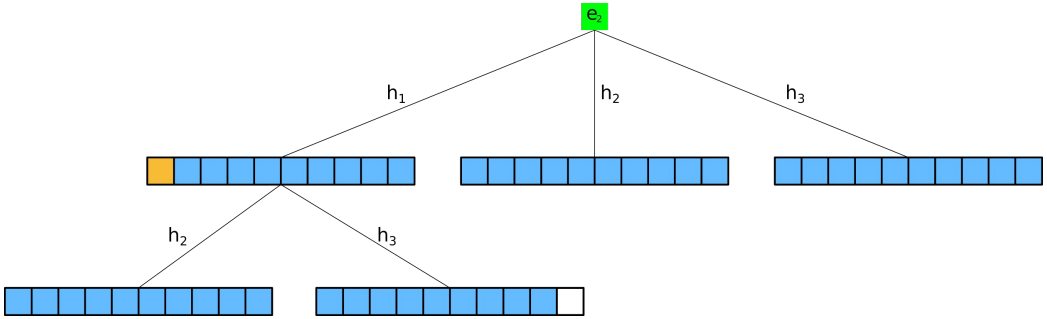


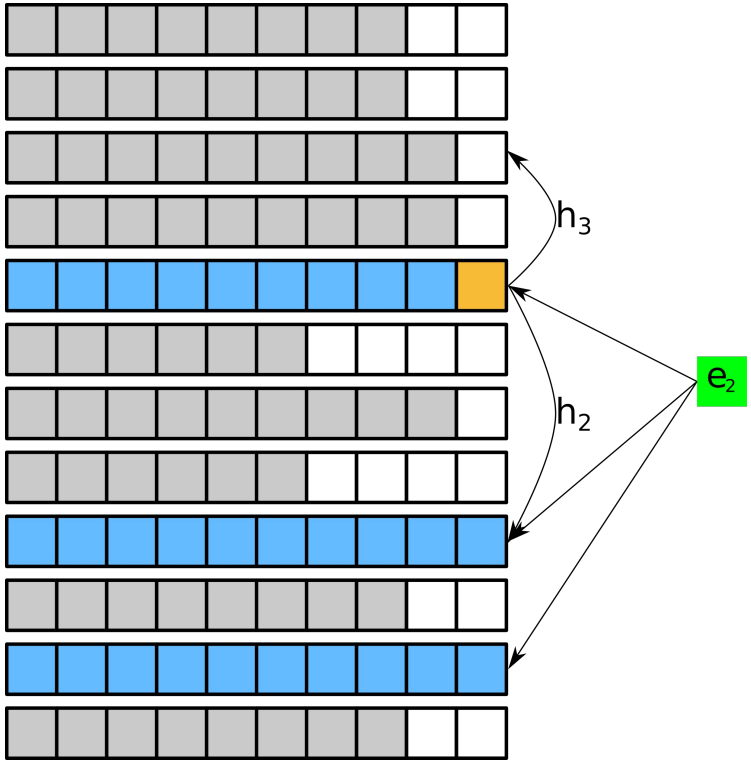
(h,b) Cuckoo hashing: BFS





(h,b) Cuckoo hashing: BFS





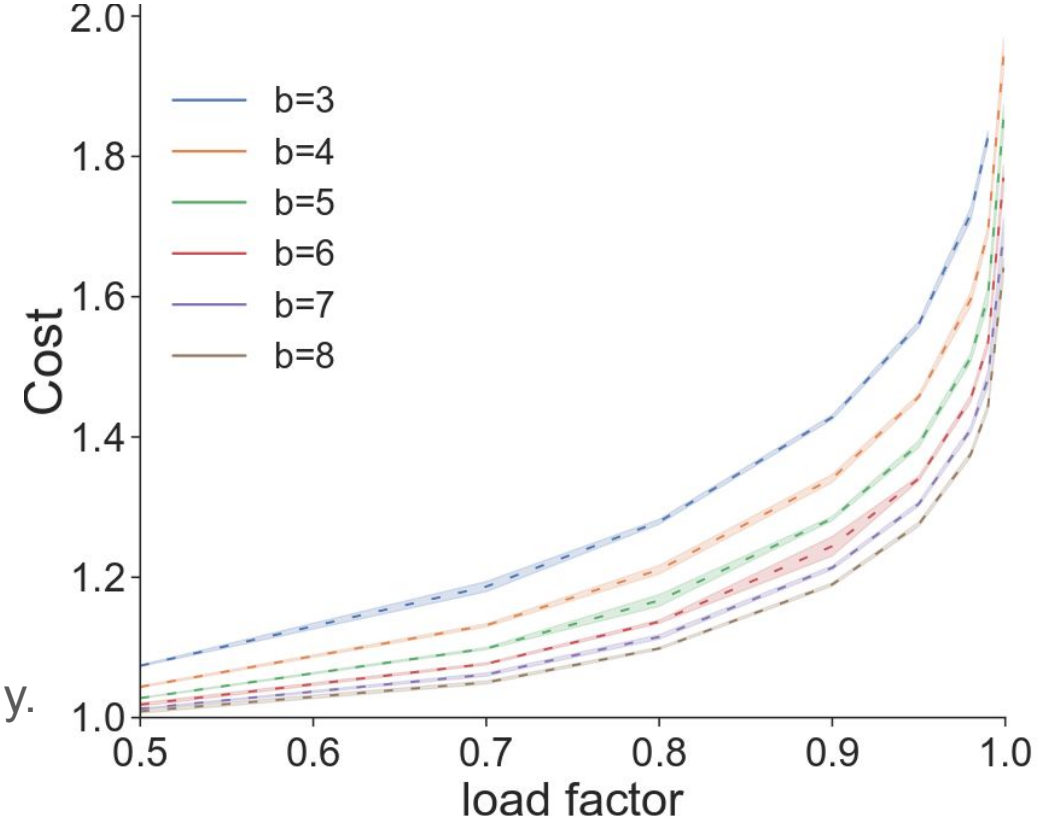
Achievable loads for (h,b) Cuckoo hashing

b **h**	**2**	**3**	**4**	**5**	**6**	**7**
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.8970	0.9882014140	0.9982414840	0.9997243601	0.9999568737	0.9999933439
3	0.95915	0.9972857393	0.9997951434	0.9999851453	0.9999989795	0.9999999329
4	0.98037	0.9992531564	0.9999720661	0.9999990737	0.9999999721	0.9999999992

S. Walzer. Load thresholds for cuckoo hashing with overlapping blocks. ICALP 2018.

Costs for (h=3, b) Cuckoo hashing with random walk

- Bucket size $b \in \{3, \ldots, 8\}$
- Load factor
  $\alpha \in \{0.5, 0.7, 0.8, 0.9, 0.95, 0.99, 0.999\}$
- Cost: expected number of memory lookups per lookup of a present key
- Even with $h = 3$, close to 1 lookup suffices on average (loads < 0.95). Worst case of 3 happens rarely.



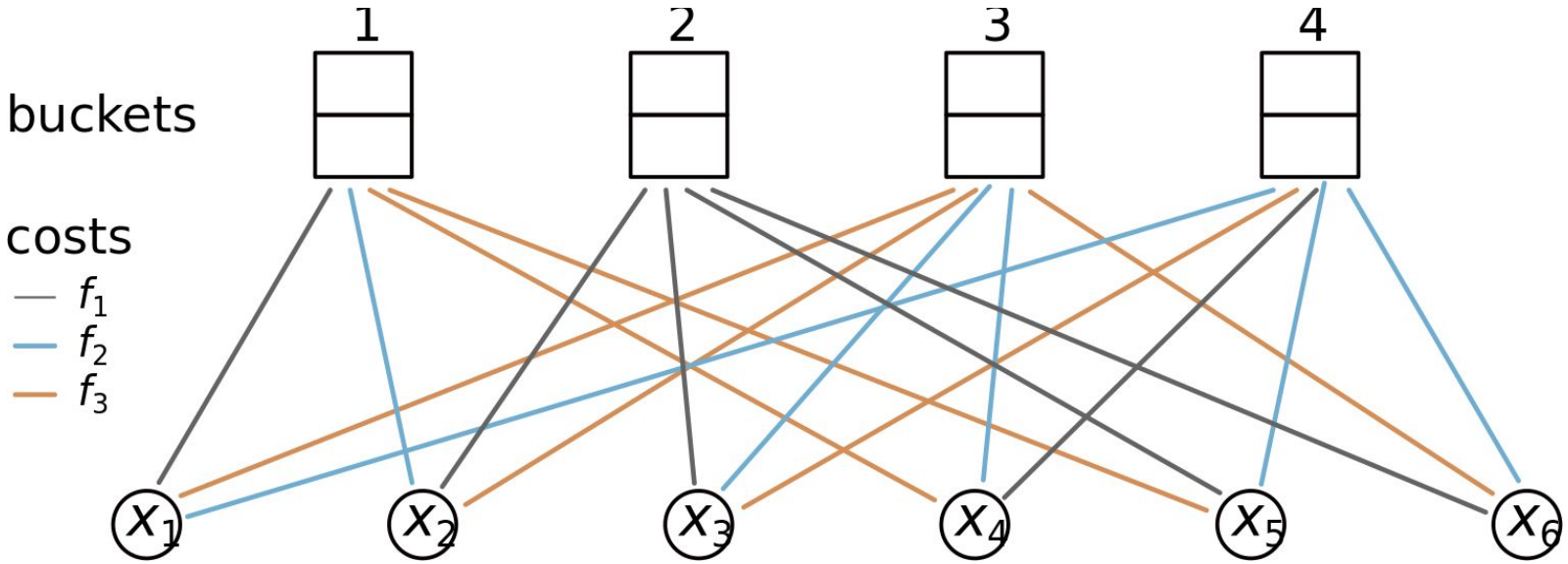
(h,b) Cuckoo hashing: Optimal

- Idea: Place many *k*-mers into the bucket of their **first** hash function.
- Can be written as a **minimum weighted bipartite matching problem**:

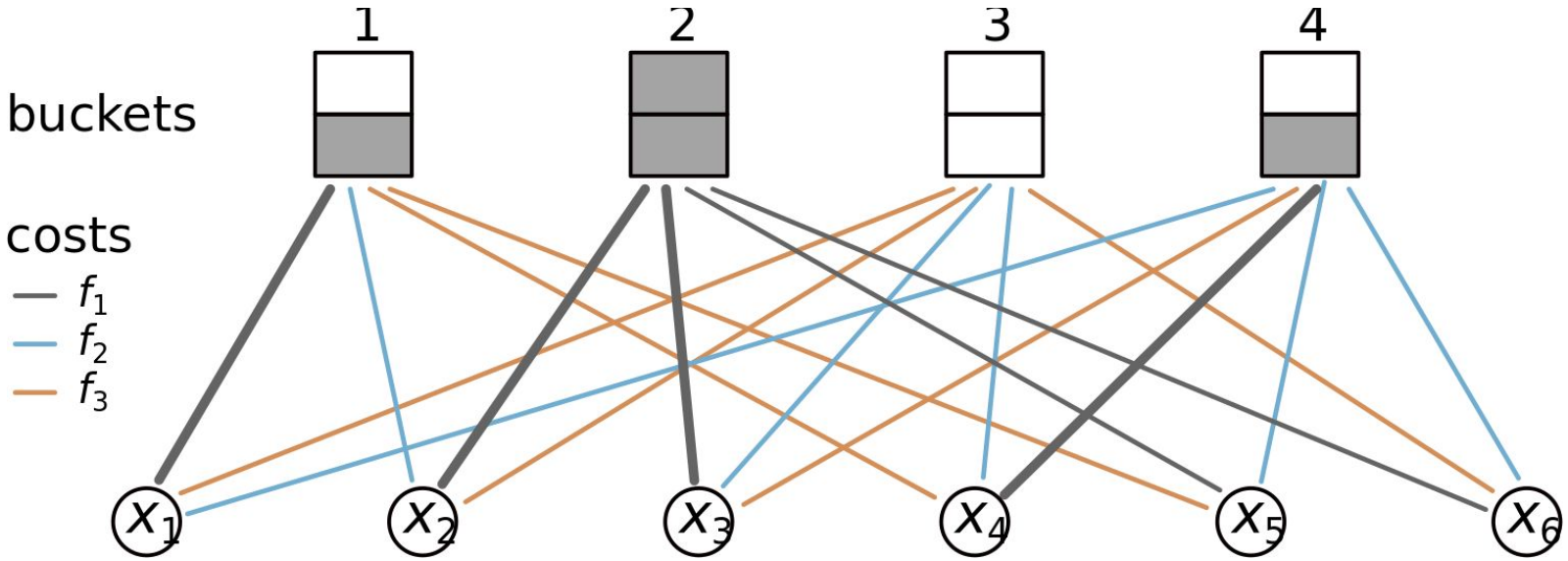
4.5 billions of keys ↔ 100s of millions of buckets

(3 buckets for each key; cost 1, 2, 3)

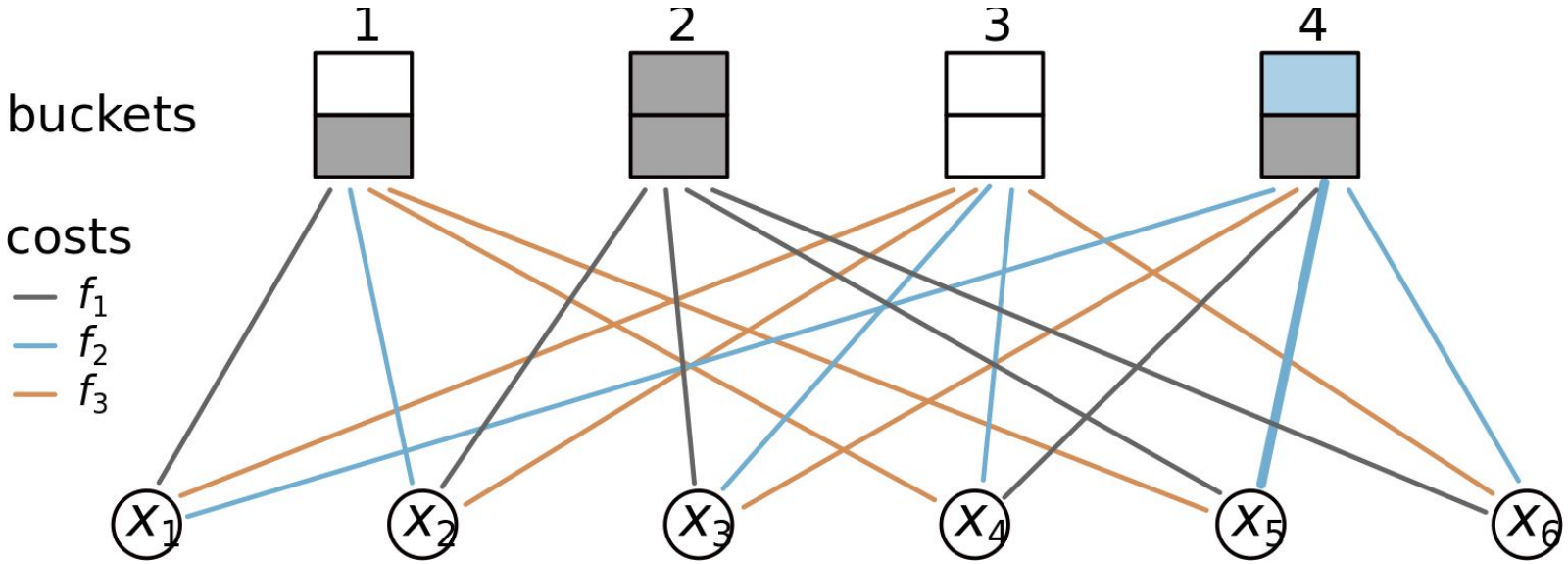
Initialization



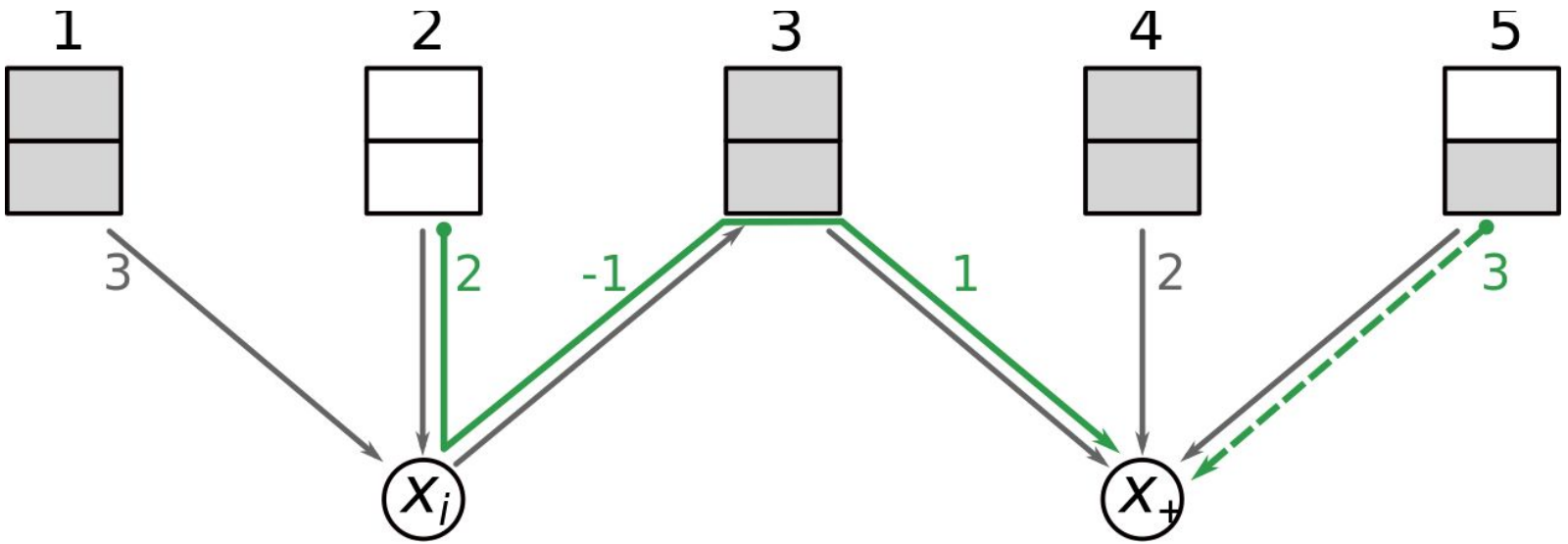
Initialization



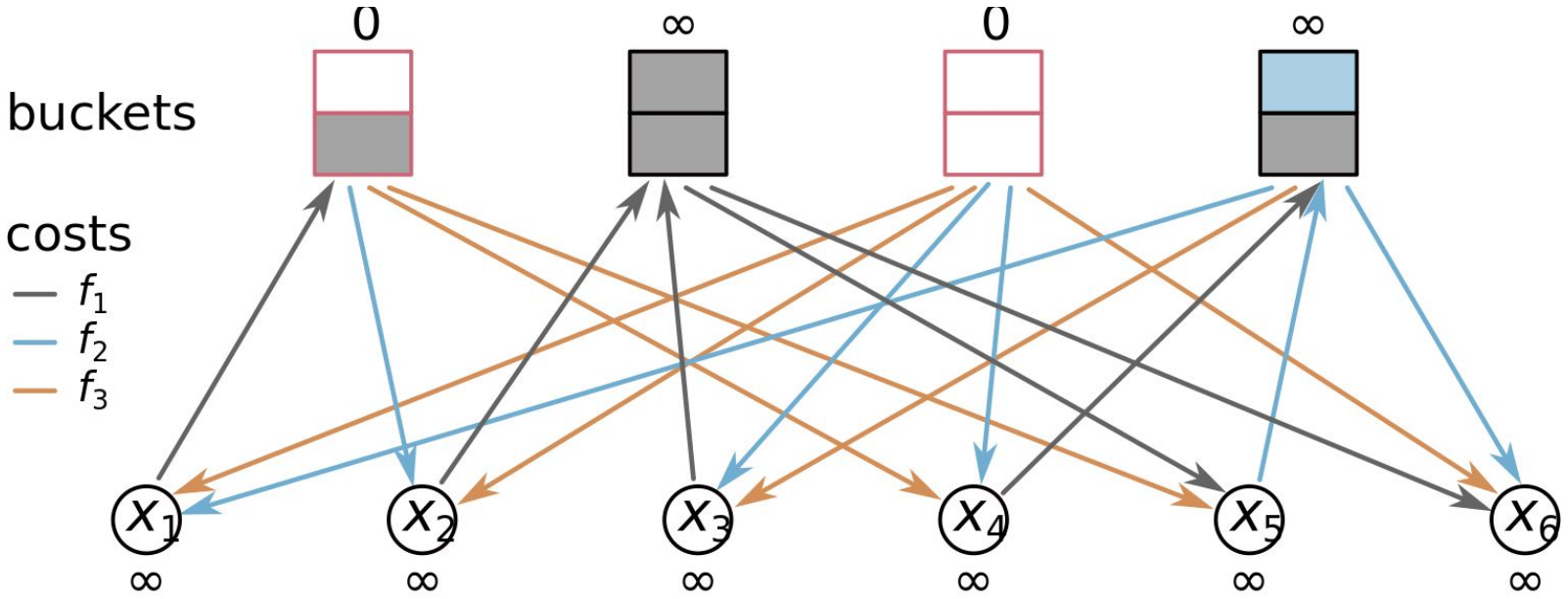
Initialization



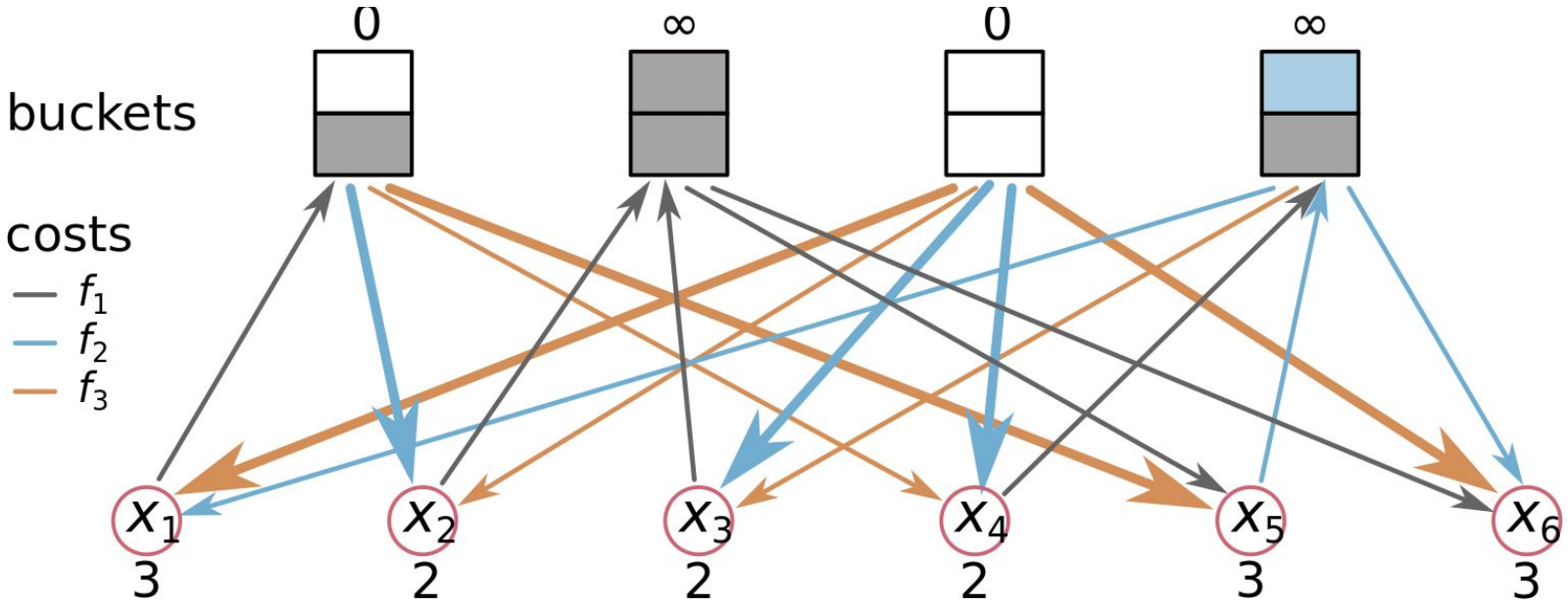
Standard Pass



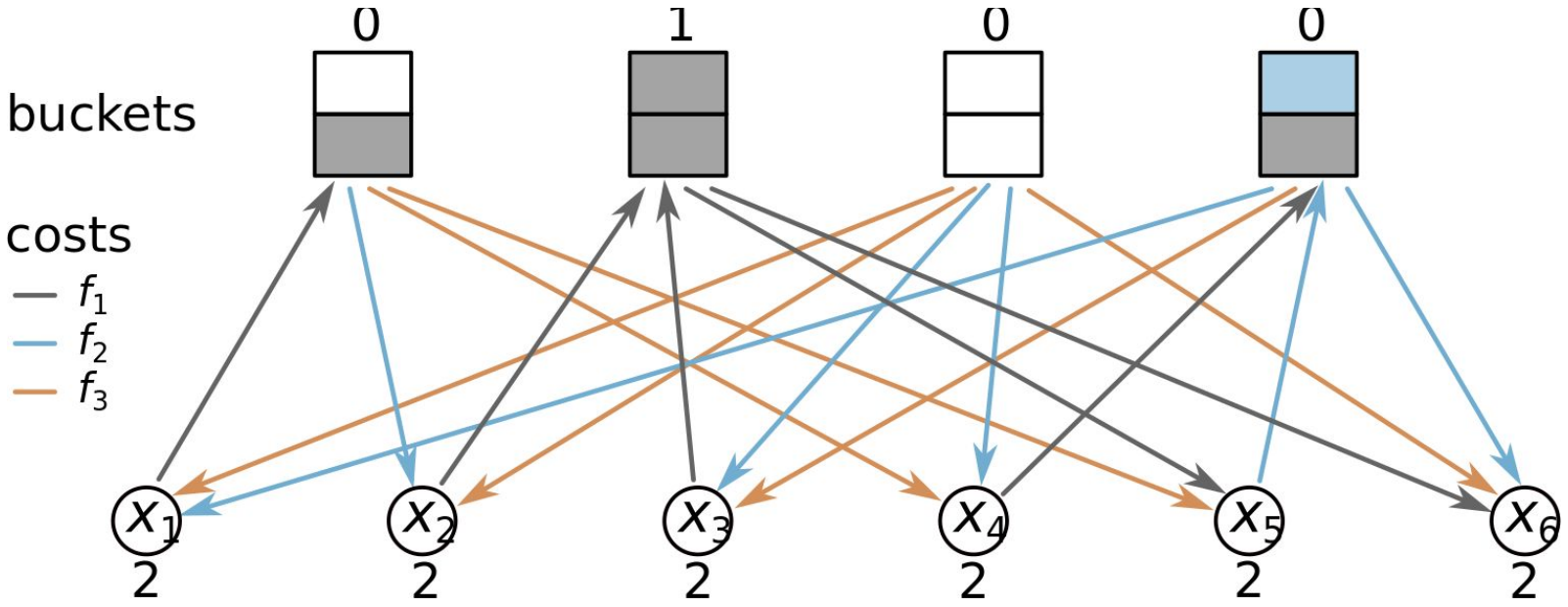
Standard Pass



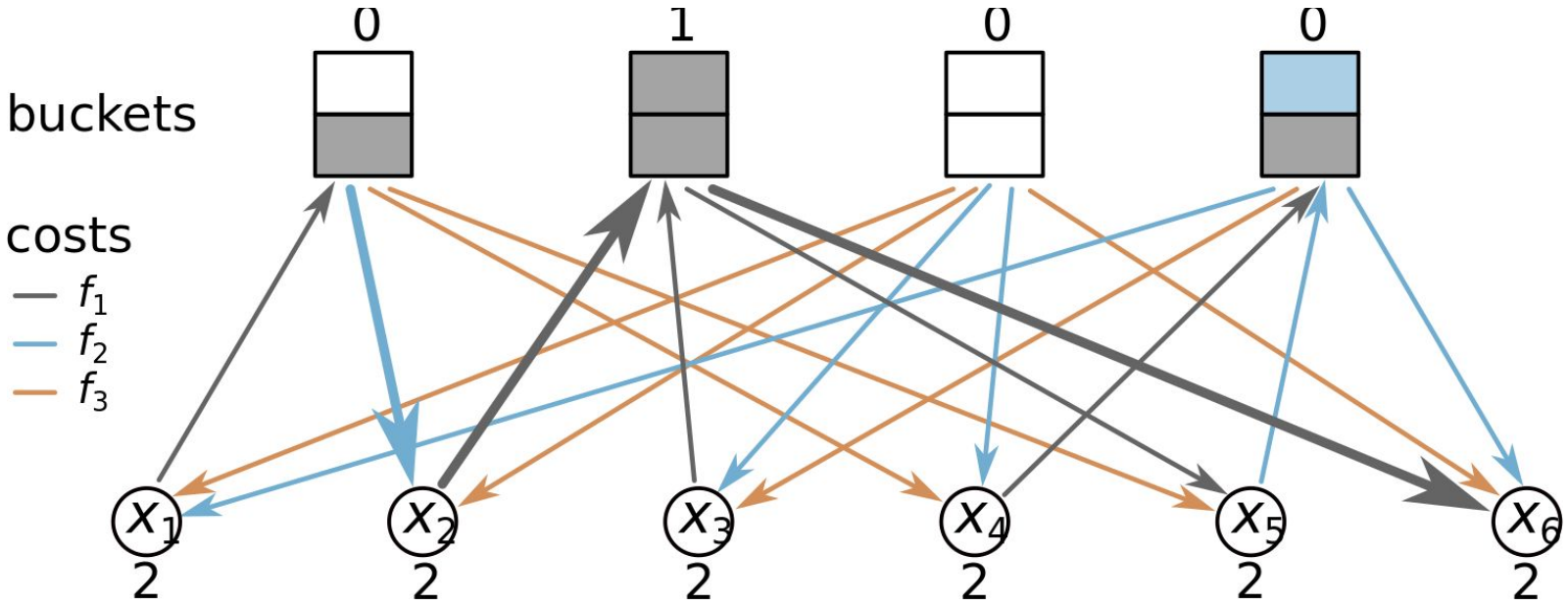
Standard Pass



Standard Pass



Standard Pass



Next part:
Performance Engineering