Dynamic External Hashing: The Limit of Buffering

Qin Zhang

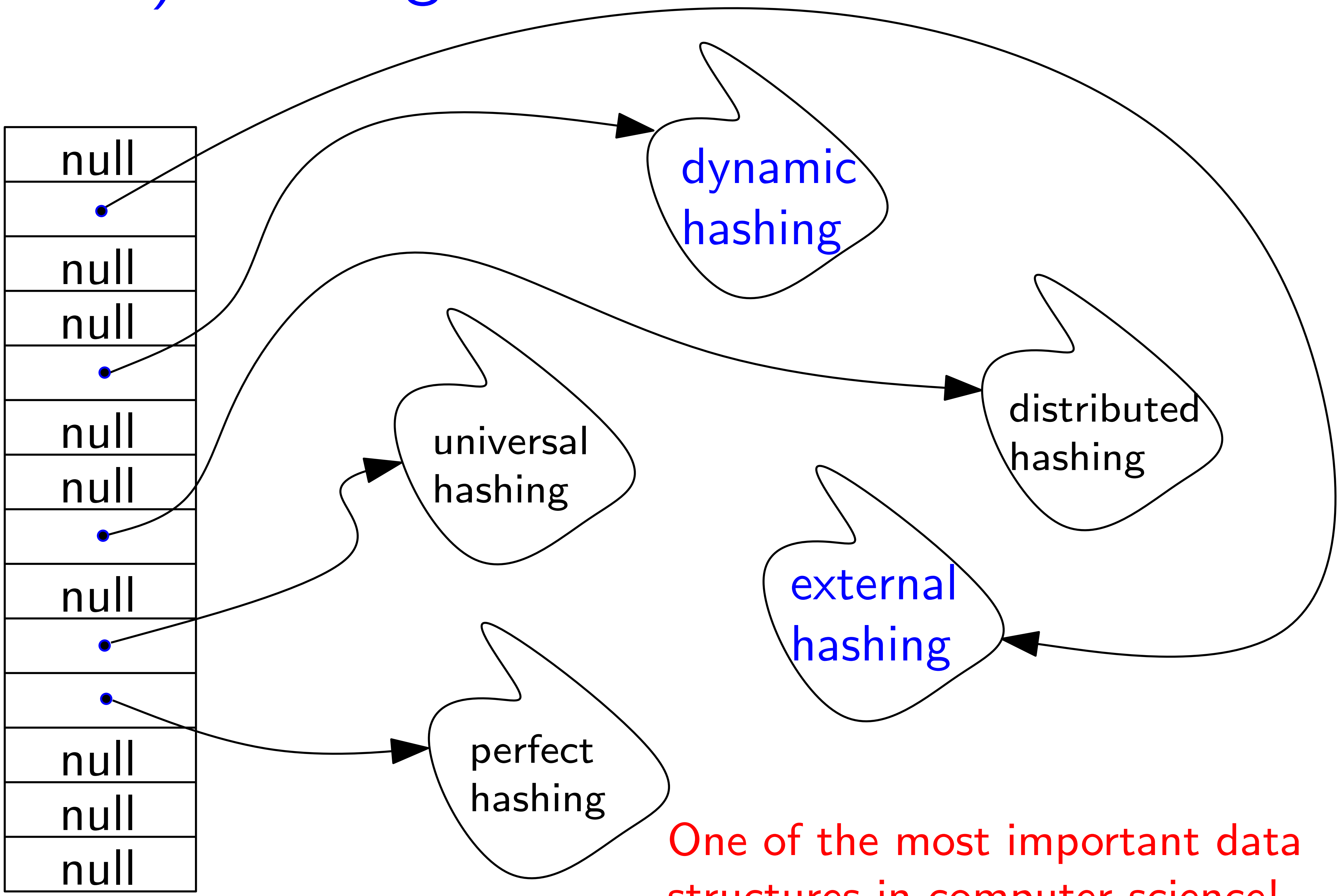
Hong Kong University of Science & Technology

Joint work with Zhewei Wei and Ke Yi

SPAA 2009

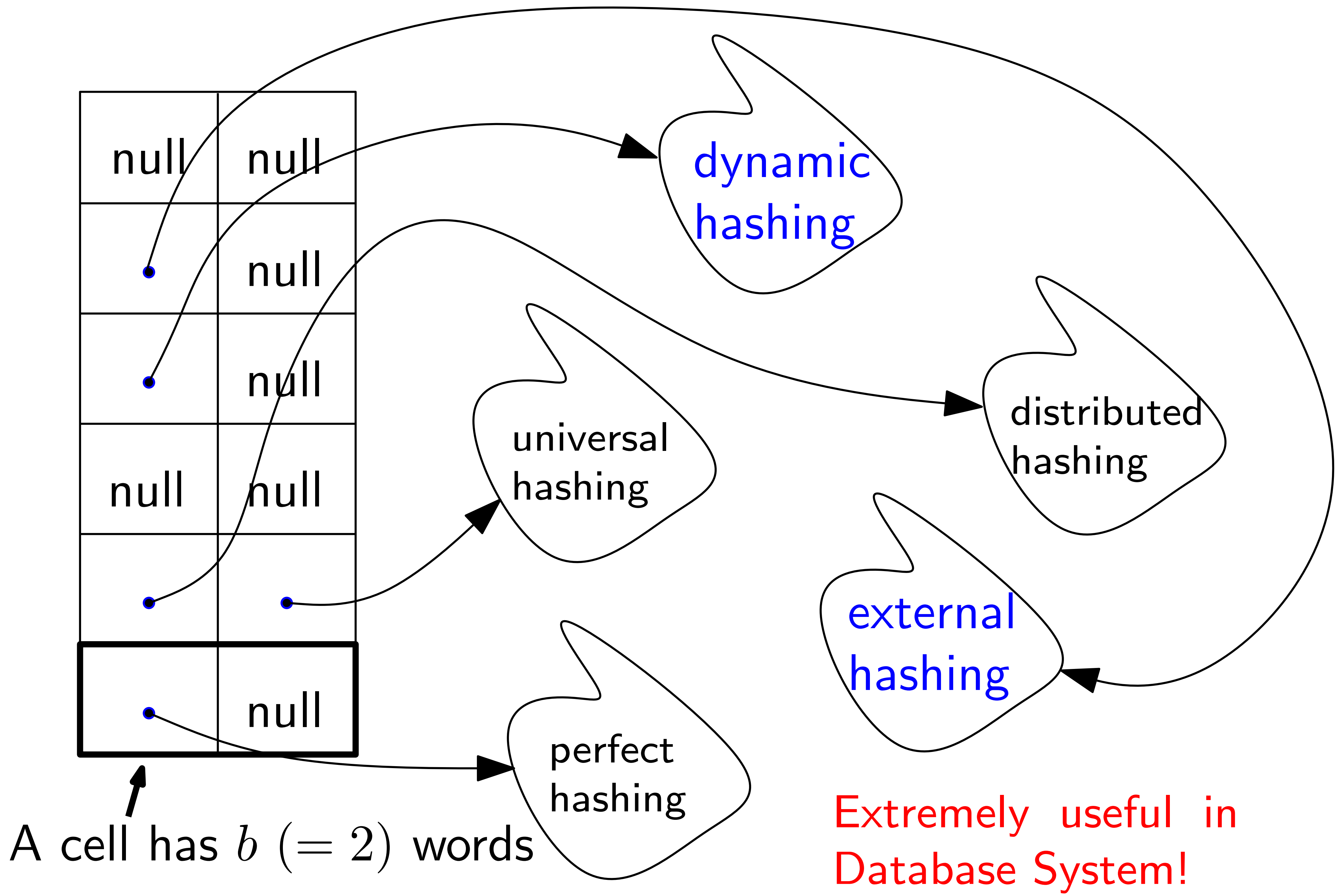
August 13, 2009

(internal) Hashing!



One of the most important data structures in computer science!

External hashing!



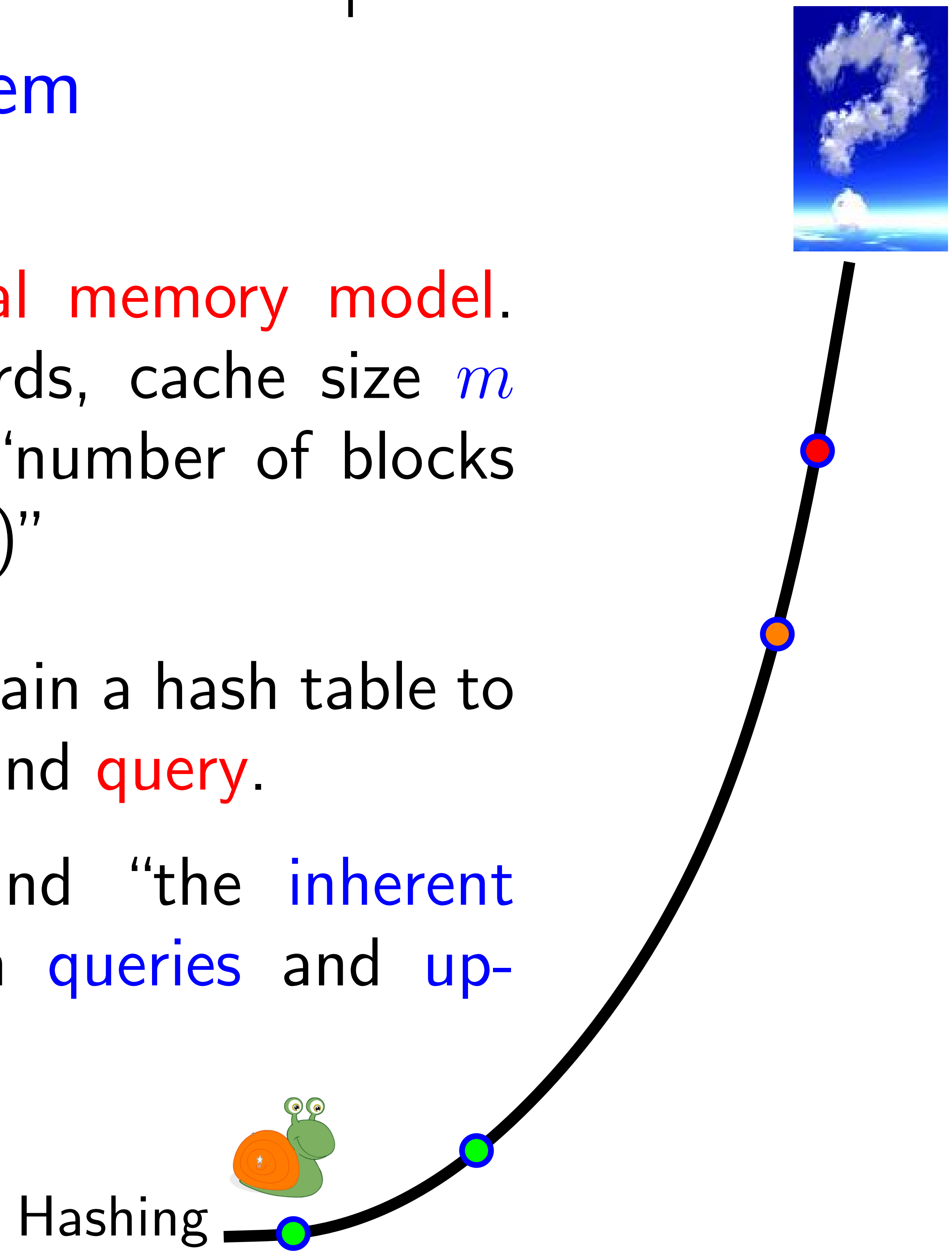
Extremely useful in Database System!

Model and problem

Model: External memory model. Block size b words, cache size m words. Cost is "number of blocks read/write (I/Os)"

Problem: Maintain a hash table to support update and query.

Try to understand "the inherent tradeoff between queries and updates"



Results

Previous results

- Hashing in the internal memory is well understood (under random inputs).

  Knuth, 1970s: $t_q = \frac{1}{2}(1+1/(1-\alpha)), t_u = 1+\frac{1}{2}(1+1/(1-\alpha)^2)$. α: load factor: minimum storage should be use/storage actually used

Previous results

- Hashing in the internal memory is well understood (under random inputs).

  Knuth, 1970s: $t_q = \frac{1}{2}(1+1/(1-\alpha)), t_u = 1+\frac{1}{2}(1+1/(1-\alpha)^2)$. α: load factor: minimum storage should be use/storage actually used

- In external memory (random inputs)

  Knuth, 1970s: Expected average cost of a query is $1 + 1/2^{\Omega(b)}$ I/Os, provided the load factor α is less than a constant smaller than 1. Update has a similar bound.

Exact Numbers Calculated by D. E. Knuth

Table 2

AVERAGE ACCESSES IN AN UNSUCCESSFUL SEARCH BY SEPARATE CHAINING

Bucket size, b	Load factor, α 10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0048	1.0187	1.0408	1.0703	1.1065	1.1488	1.197	1.249	1.307	1.34
2	1.0012	1.0088	1.0269	1.0581	1.1036	1.1638	1.238	1.327	1.428	1.48
3	1.0003	1.0038	1.0162	1.0433	1.0898	1.1588	1.252	1.369	1.509	1.59
4	1.0001	1.0016	1.0095	1.0314	1.0751	1.1476	1.253	1.394	1.571	1.67
5	1.0000	1.0007	1.0056	1.0225	1.0619	1.1346	1.249	1.410	1.620	1.74
10	1.0000	1.0000	1.0004	1.0041	1.0222	1.0773	1.201	1.426	1.773	2.00
20	1.0000	1.0000	1.0000	1.0001	1.0028	1.0234	1.113	1.367	1.898	2.29
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.018	1.182	1.920	2.70

Table 3

AVERAGE ACCESSES IN A SUCCESSFUL SEARCH BY SEPARATE CHAINING

Bucket size, b	Load factor, α 10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0500	1.1000	1.1500	1.2000	1.2500	1.3000	1.350	1.400	1.450	1.48
2	1.0063	1.0242	1.0520	1.0883	1.1321	1.1823	1.238	1.299	1.364	1.40
3	1.0010	1.0071	1.0215	1.0458	1.0806	1.1259	1.181	1.246	1.319	1.36
4	1.0002	1.0023	1.0097	1.0257	1.0527	1.0922	1.145	1.211	1.290	1.33
5	1.0000	1.0008	1.0046	1.0151	1.0358	1.0699	1.119	1.186	1.268	1.32
10	1.0000	1.0000	1.0002	1.0015	1.0070	1.0226	1.056	1.115	1.206	1.27
20	1.0000	1.0000	1.0000	1.0000	1.0005	1.0038	1.018	1.059	1.150	1.22
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.015	1.083	1.16

The Art of Computer Programming, volume 3, 1998, page 542

Can we batch updates?

- However, in external memory model, disk reads/writes are expensive and powerful. Can we hope for lower than 1 I/O per update?

Can we batch updates?

- However, in external memory model, disk reads/writes are expensive and powerful. Can we hope for lower than 1 I/O per update?
  - No, if there is no space in main memory for buffering. But, not the case in reality!

Can we batch updates?

- However, in external memory model, disk reads/writes are expensive and powerful. Can we hope for lower than 1 I/O per update?
  - No, if there is no space in main memory for buffering. But, not the case in reality!
  - Maybe yes, if we have an $\Omega(b)$ main memory for buffering! Like numerous problems in external memory, e.g. stack. More: priority queue, buffer tree ...

    Can the amortized update cost be something like $O(1/b^c)$ (for some $0 < c \leq 1$) for hashing?

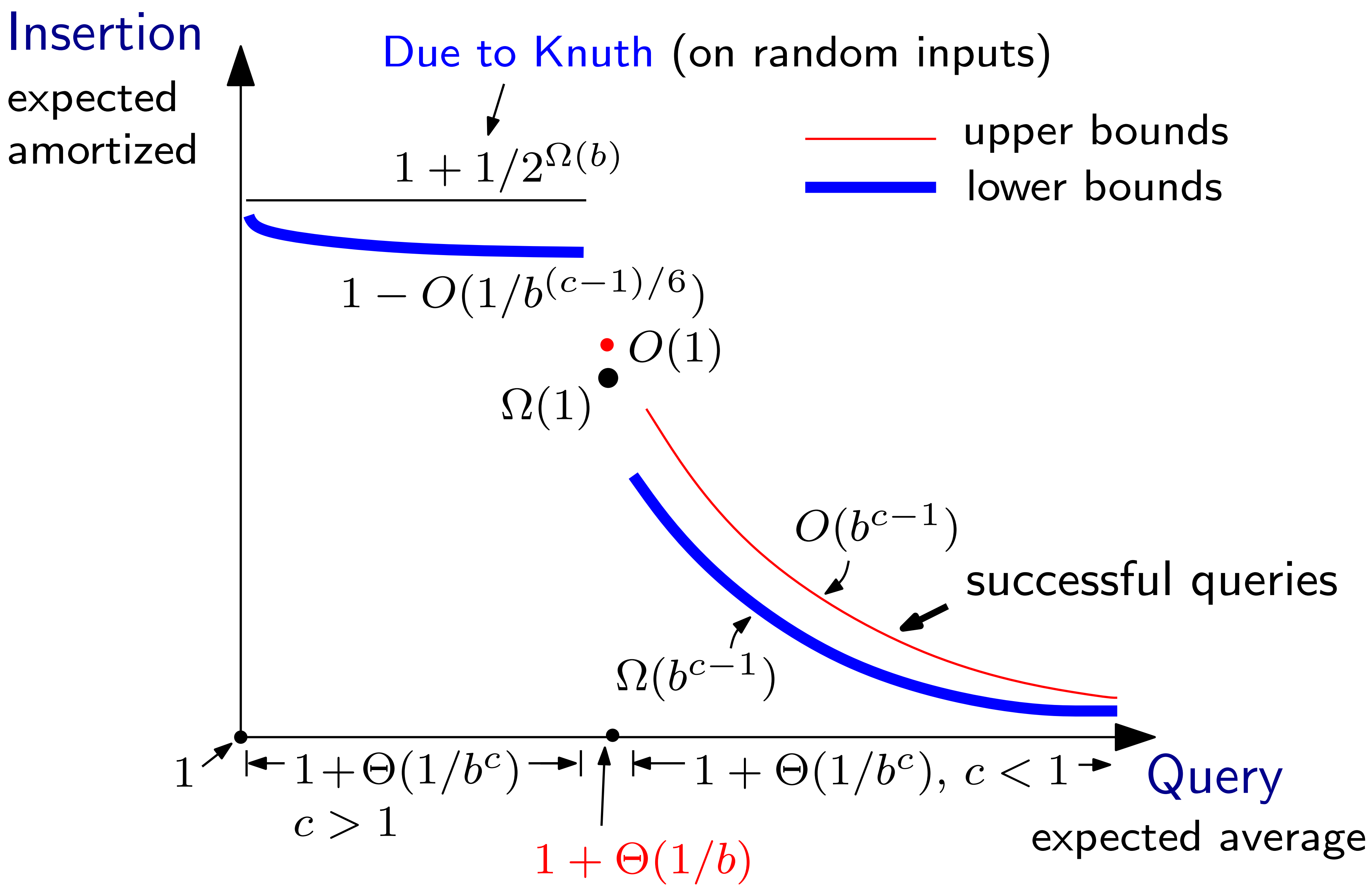
Can we batch updates?

- However, in external memory model, disk reads/writes are expensive and powerful. Can we hope for lower than 1 I/O per update?
  - No, if there is no space in main memory for buffering. But, not the case in reality!
  - Maybe yes, if we have an $\Omega(b)$ main memory for buffering! Like numerous problems in external memory, e.g. stack. More: priority queue, buffer tree ...

    Can the amortized update cost be something like $O(1/b^c)$ (for some $0 < c \leq 1$) for hashing?
- Conjectured by Jensen and Pagh (2007):

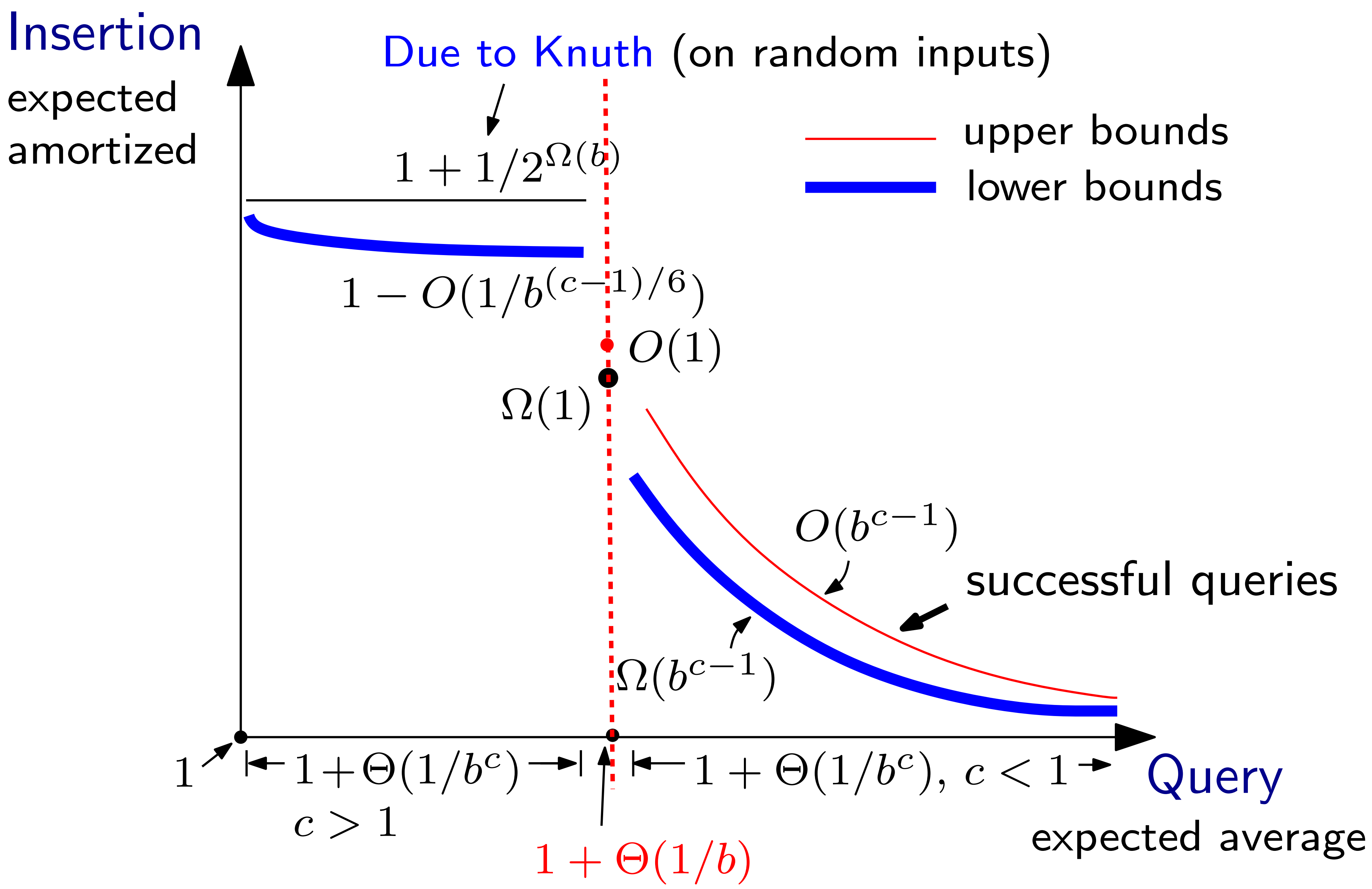
  The insertion cost must be $\Omega(1)$ I/Os if the query cost is required to be $O(1)$ I/Os.

Our results



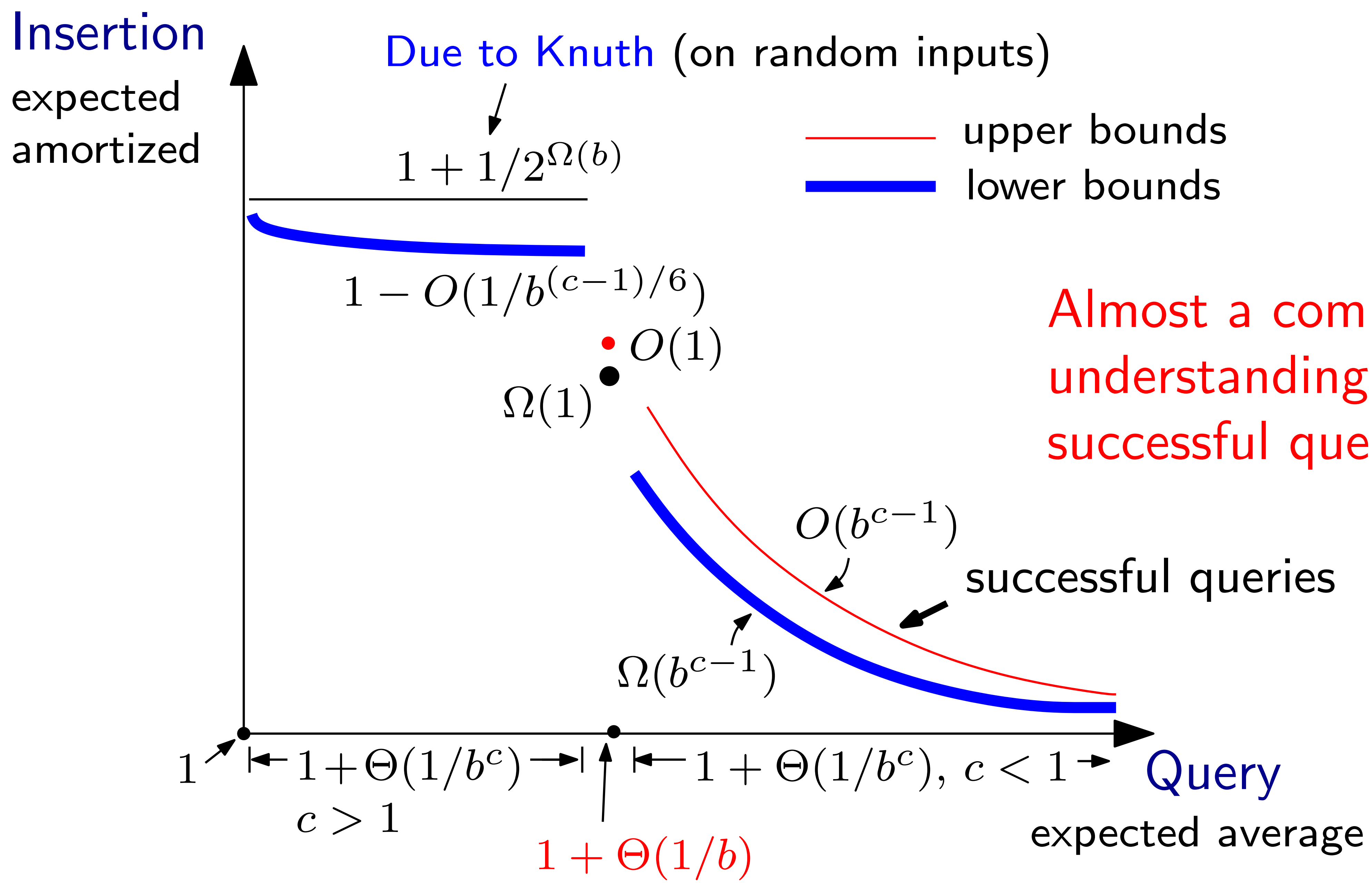
Insertion
expected
amortized
Due to Knuth (on random inputs)
$1 + 1/2^{\Omega(b)}$
upper bounds
lower bounds
$1 - O(1/b^{(c-1)/6})$
$O(1)$
$\Omega(1)$
$O(b^{c-1})$
successful queries
$\Omega(b^{c-1})$
1
$1+\Theta(1/b^c)$
$c > 1$
$1+\Theta(1/b^c)$, $c < 1$
$1+\Theta(1/b)$
Query
expected average

Our results



Insertion
expected
amortized
Due to Knuth (on random inputs)
$1 + 1/2^{\Omega(b)}$
upper bounds
lower bounds
$1 - O(1/b^{(c-1)/6})$
$O(1)$
$\Omega(1)$
$O(b^{c-1})$
successful queries
$\Omega(b^{c-1})$
1
$1+\Theta(1/b^c)$
$c > 1$
$1+\Theta(1/b^c)$, $c < 1$
$1 + \Theta(1/b)$
Query
expected average

Our results



Almost a complete understanding for successful queries!

Other related results

- Upper bounds
  - Remove *ideal hash function* assumption [Carter and Wegman 1979], making query worst-case [i.e. Fredman, Komlos and Szemeredi, 1984] ... (internal)
  - Queries and updates in $1 + O(1/b^{\frac{1}{2}})$ I/Os with $\alpha = 1 - O(1/b^{\frac{1}{2}})$ [Jensen and Pagh, 2007]. (external, no memory)

Other related results

- Upper bounds
  - Remove *ideal hash function* assumption [Carter and Wegman 1979], making query worst-case [i.e. Fredman, Komlos and Szemeredi, 1984] ... (internal)
  - Queries and updates in $1 + O(1/b^{\frac{1}{2}})$ I/Os with $\alpha = 1 - O(1/b^{\frac{1}{2}})$ [Jensen and Pagh, 2007]. (external, no memory)
- Lower bounds (internal)

  Very sparse, only with some strong requirements, e.g., the algorithm is deterministic and query is worst-case [Dietzfelbinger et. al. 1994].

Other related results

- Upper bounds
  - Remove *ideal hash function* assumption [Carter and Wegman 1979], making query worst-case [i.e. Fredman, Komlos and Szemeredi, 1984] ... (internal)
  - Queries and updates in $1 + O(1/b^{\frac{1}{2}})$ I/Os with $\alpha = 1 - O(1/b^{\frac{1}{2}})$ [Jensen and Pagh, 2007]. (external, no memory)
- Lower bounds (internal)

  Very sparse, only with some strong requirements, e.g., the algorithm is deterministic and query is worst-case [Dietzfelbinger et. al. 1994].
- Lower bounds in other dynamic external memory problems

  Only known are query-update tradeoffs for the *predecessor* [Fagerberg and Brodal 2003], *range reporting* [Yi 2009].

Technical details:

Lowerbounds

Preliminaries

- $U = \{0, 1, \ldots, u-1\}$: universe. $|U| = u$.
- m: size of main memory. n: total number of items.
  b: size of one block. All in words.

Preliminaries

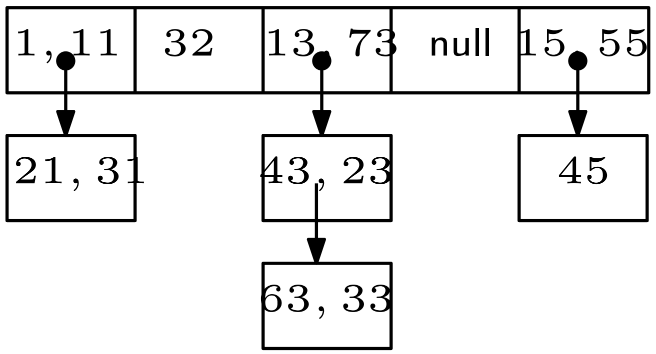
- $U = \{0, 1, \ldots, u-1\}$: universe. $|U| = u$.
- m: size of main memory. n: total number of items. b: size of one block. All in words.
- Some mild assumptions
  - Atomic elements
  - $n \geq \Omega\left(m \log u \cdot b^{2c}\right)$ for some constant $c > 0$

Preliminaries

- $U = \{0, 1, \ldots, u-1\}$: universe. $|U| = u$.
- m: size of main memory. n: total number of items.
  b: size of one block. All in words.
- Some mild assumptions
  - Atomic elements
  - $n \geq \Omega\left(m \log u \cdot b^{2c}\right)$ for some constant $c > 0$
- Deterministic data structure $+$ a random distrib. of inputs
  (Via a method similar to Yao's Minimax Principle) $\Longrightarrow$
  Randomized data structure

Observations

- Two extreme cases
  - One exterme: only use a fixed mapping for all items.

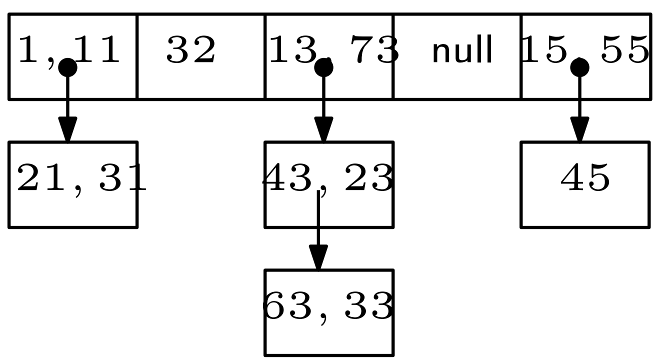


$b = 2$

Update is expensive!

Observations

- Two extreme cases
  - One exterme: only use a fixed mapping for all items.



$b = 2$

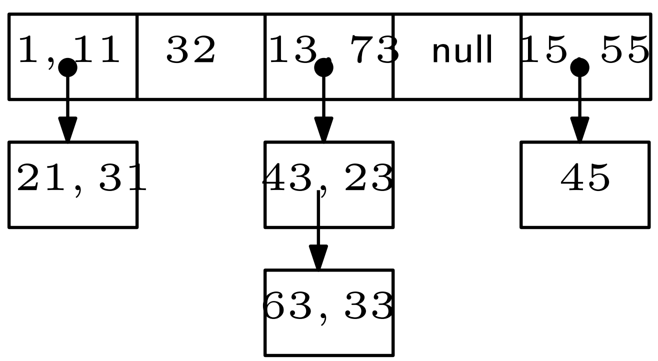
Update is expensive!

  - Another exterme: for every b items come, write to a new block.

Too many possible mappings.

Observations

- Two extreme cases
  - One exterme: only use a fixed mapping for all items.



$b = 2$

Update is expensive!

  - Another exterme: for every b items come, write to a new block.

Too many possible mappings.

- Also easy to see

If with only the information in memory, the hash table cannot locate the item, then querying it takes at least 2 I/Os.

The abstraction

- Consider the layout of a hash table at any snapshot. Denote all the blocks on disk by $B_0, B_1, B_2, \ldots, B_d$ ($B_0 = M$). Let $f : U \rightarrow \{0, 1, \ldots, d\}$ be any function computable within memory.

  When querying $x \in U$, $f(x)$: index of the first block the DS will probe. If $f(x) = 0$, the DS will still probe the memory.

The abstraction

- Consider the layout of a hash table at any snapshot. Denote all the blocks on disk by $B_0, B_1, B_2, \ldots, B_d$ $(B_0 = M)$. Let $f : U \to \{0, 1, \ldots, d\}$ be any function computable within memory.

  When querying $x \in U$, $f(x)$: index of the first block the DS will probe. If $f(x) = 0$, the DS will still probe the memory.

- We divide items inserted into 3 zones with respect to f.

Memory

Memory zone M: set of items stored in memory. $t_q = 0$.

Disk

Fast zone F: set of items x such that $x \in B_{f(x)}$. $t_q = 1$.

Slow zone S: The rest of items. $t_q \geq 2$.

The key idea

The hash table can employ a family $\mathcal{F}$ of at most $2^{m \log u}$ distinct f's.

Note that the current f adopted by the hash table is dependent upon the already inserted items, but the family $\mathcal{F}$ has to be fixed beforehand.

Size of the slow zone is small.

- Suppose the hash table answers a successful query with an expected average cost of $t_q = 1 + \delta$ I/Os. Consider the snapshot when k random items have been inserted.

$$E[|S|] \leq m + \delta k.$$

Memory zone M: $t_q = 0$

Fast zone F: $t_q = 1$

Slow zone S: $t_q \geq 2$

small!

Size of the slow zone is small.

- Suppose the hash table answers a successful query with an expected average cost of $t_q = 1 + \delta$ I/Os. Consider the snapshot when k random items have been inserted.

$$E[|S|] \leq m + \delta k.$$

Memory zone M: $t_q = 0$

Fast zone F: $t_q = 1$

Slow zone S: $t_q \geq 2$

small!

- The high-probability version.

  LEMMA 1. Let $\phi \geq 1/b^{(c-1)/4}$ and let $k \geq \phi n$. At the snapshot when k items have been inserted, with probability at least $1 - 2\phi$, $|S| \leq m + \frac{\delta}{\phi}k$.

Basic idea of the lower bound proof

- Consider any $f : U \to \{0, 1, \ldots, d\}$. For $i = 0, \ldots, d$, let $\alpha_i = |f^{-1}(i)|/u$, and we call $(\alpha_0, \alpha_1, \ldots, \alpha_d)$ the characteristic vector of f.

Basic idea of the lower bound proof

- Consider any $f : U \rightarrow \{0, 1, \ldots, d\}$. For $i = 0, \ldots, d$, let $\alpha_i = |f^{-1}(i)|/u$, and we call $(\alpha_0, \alpha_1, \ldots, \alpha_d)$ the characteristic vector of f.
- After ϕn random insertions. Pick a fixed threshold ρ. Assume $\alpha_0 \geq \alpha_1 \geq \ldots \geq \alpha_d$

f	α_0	α_1	α_2	•••	α_k	•••	α_d

ρ

(A) If $\exists$ too many large α_i's, S too large, violating the query requirement (LEMMA 1).

Basic idea of the lower bound proof

- Consider any $f : U \rightarrow \{0, 1, \ldots, d\}$. For $i = 0, \ldots, d$, let $\alpha_i = |f^{-1}(i)|/u$, and we call $(\alpha_0, \alpha_1, \ldots, \alpha_d)$ the characteristic vector of f.
- After ϕn random insertions. Pick a fixed threshold ρ. Assume $\alpha_0 \geq \alpha_1 \geq \ldots \geq \alpha_d$

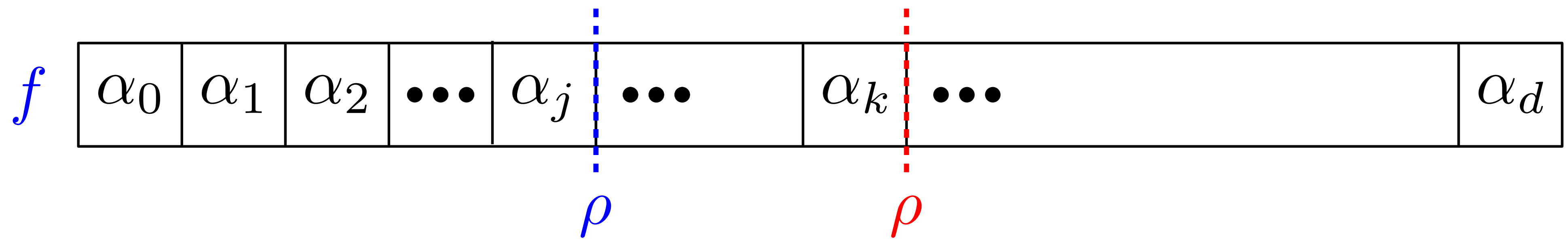
f	α_0	α_1	α_2	•••	α_j	•••	α_k	•••	α_d

ρ

(B) Else f is likely to distribute recent randomly inserted items evenly, leading to a high insertion cost.

Basic idea of the lower bound proof

- Consider any $f : U \to \{0, 1, \ldots, d\}$. For $i = 0, \ldots, d$, let $\alpha_i = |f^{-1}(i)|/u$, and we call $(\alpha_0, \alpha_1, \ldots, \alpha_d)$ the characteristic vector of f.
- After ϕn random insertions. Pick a fixed threshold ρ. Assume $\alpha_0 \geq \alpha_1 \geq \ldots \geq \alpha_d$



(A) If $\exists$ too many large α_i's, S too large, violating the query requirement (LEMMA 1).

(B) Else f is likely to distribute recent randomly inserted items evenly, leading to a high insertion cost.

Both hold with very high probability, even after taking union of all $O(2^{m \log u})$ different f.

Upper bounds

Easy!

Logarithmic method

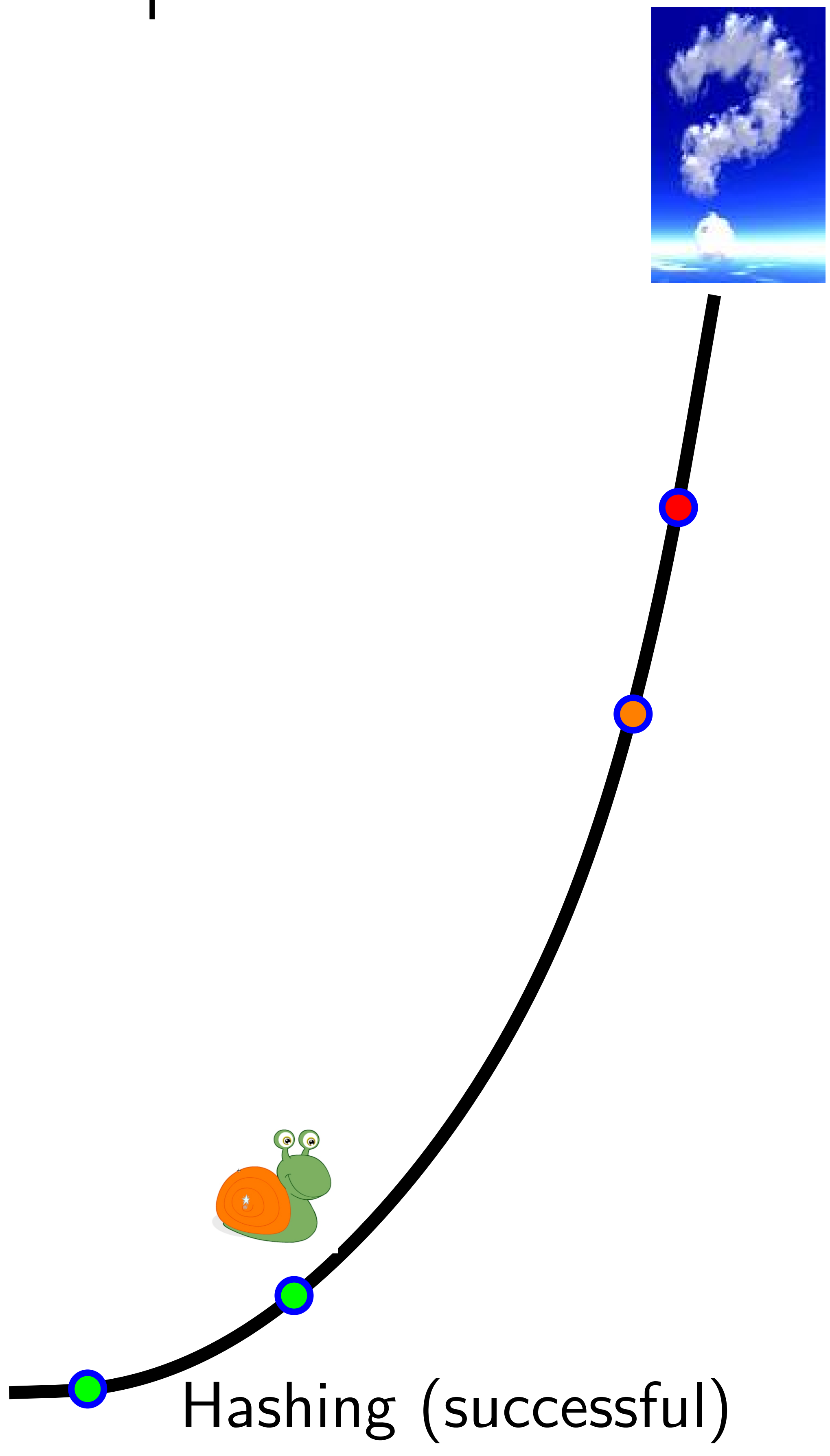
+ Query start from the last (biggest) layer

  + Tricks to keep the last layer large

Beyond hashing:

Subsequent and future work

Beyond hashing



Beyond hashing

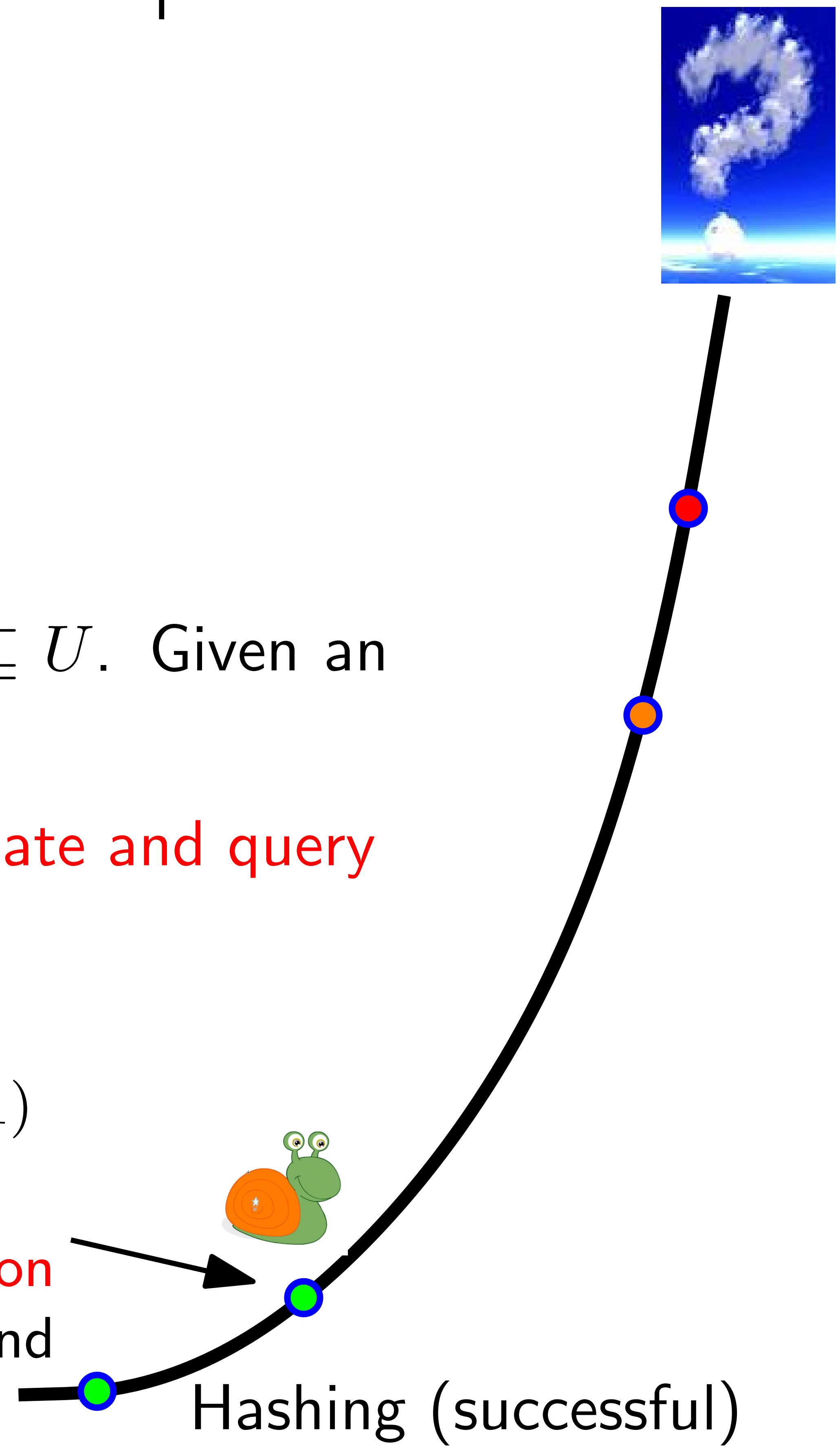
Membership

Problem: Maintain a set $S \subseteq U$. Given an $x \in U$, answer $x \in S$?

Again, tradeoffs between update and query

Membership: if $t_q \leq 1 + \delta$ $(0 \leq \delta < 1/2)$, then $t_u \geq \Omega(1)$ [Yi and Zhang 2009]

(1) Without atomic assumption
(2) Consider both successful and unseccessful query



Beyond hashing

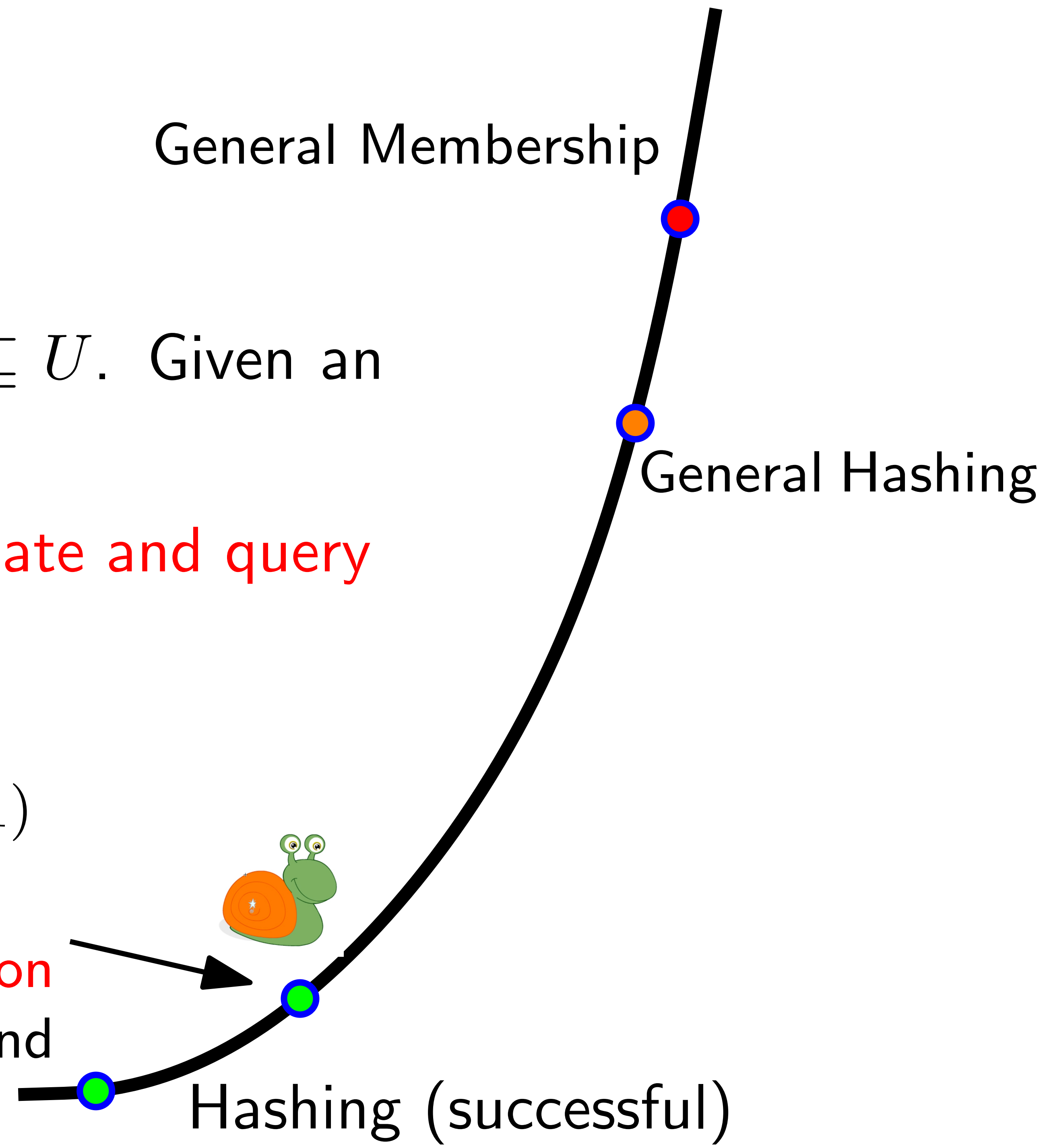
Membership

Problem: Maintain a set $S \subseteq U$. Given an $x \in U$, answer $x \in S$?

Again, tradeoffs between update and query

Membership: if $t_q \leq 1 + \delta$ $(0 \leq \delta < 1/2)$, then $t_u \geq \Omega(1)$ [Yi and Zhang 2009]

(1) Without atomic assumption
(2) Consider both successful and unseccessful query



More problems

Lower bounds of other dynamic problems in the cell probe with cache setting.

1. predecessor, range-sum
2. union-find
3. . . .

The End

THANK YOU

Q and A

The Banff National Park