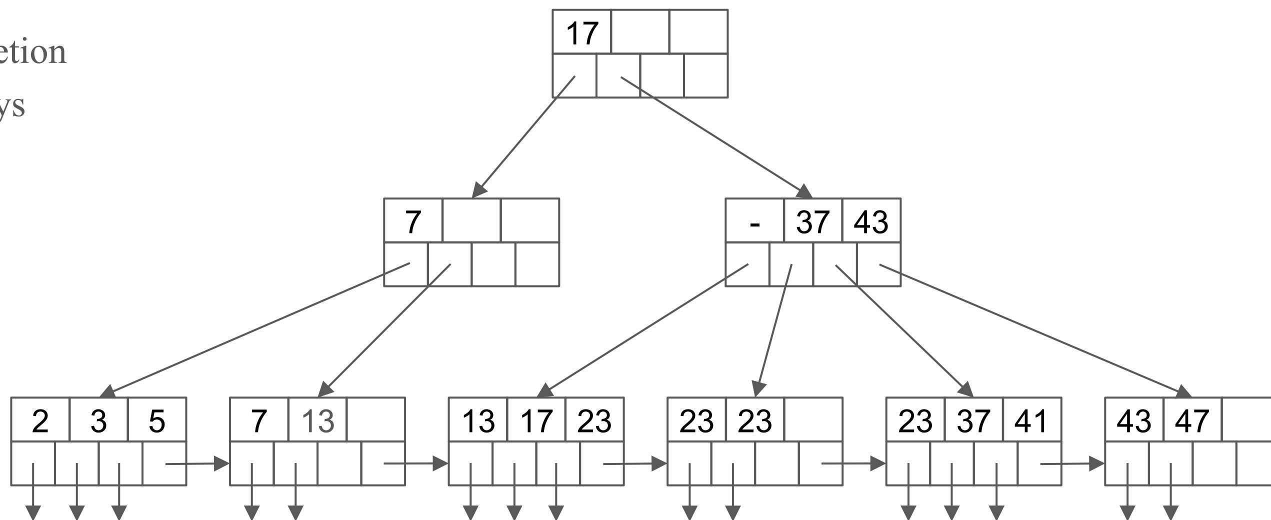
CS 4440 A

# Emerging Database Technologies

Lecture 7
01/31/24

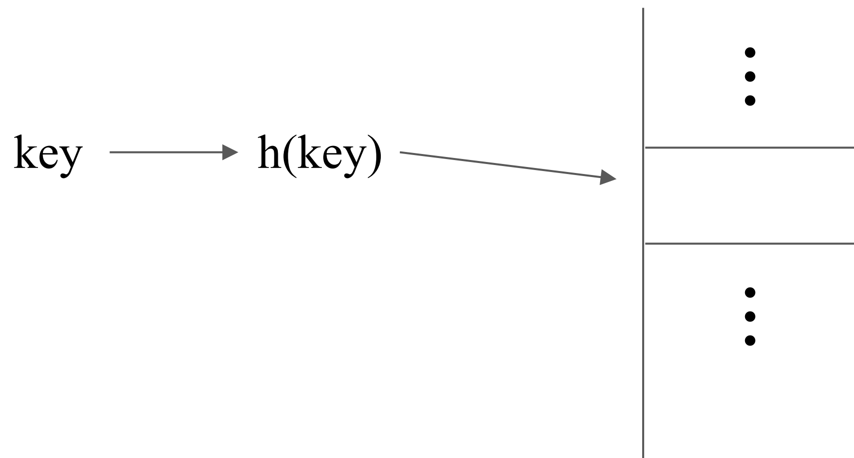
# Recap

- B+ tree
  - Lookup, insertion, deletion
  - Handling duplicate keys
  - I/O efficiency



# Hash table

- A hash function *h* takes a key and returns a block number from 0 to *B* - 1
- Blocks contain records and are stored in secondary storage
- Complexity:
  - O(1) operation complexity
  - O(n) storage complexity



# Hash table: Design Decisions

- Hash Function
  - How to map a large key space into a smaller domain of array offsets
  - Trade-off between fast execution vs. collision rate

- Hashing Scheme
  - How to handle key collisions after hashing
  - Trade-off between allocating a large hash table vs. extra steps to location/insert keys

# Hash function

- For any input key, return an integer representation of that key.
  - Output is deterministic
- Example:
  - Given a key that is a string, return the sum of the characters $x_i$ modulo $B$ (*i*.e., $\Sigma x_i \% B$)
    - This function is not idea since there might be many collisions
- We do NOT want to use a cryptographic hash function (e.g., SHA-256) for DBMS hash tables
- In general, we only care about the hash function's speed and collision rate.
- Current SOTA: xxHash

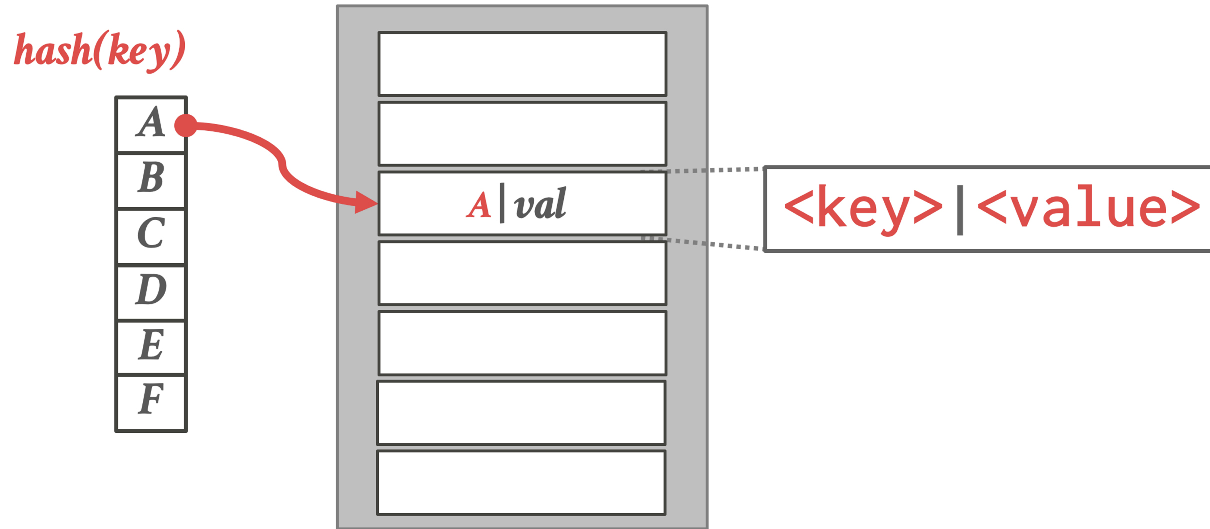
# Static hash table

- The number of buckets is fixed
- Often used during query execution because they are faster than dynamic hashing schemes.
- If the DBMS runs out of storage space in the hash table, it has to rebuild a larger hash table (usually 2x) from scratch, which is very expensive!
- Examples
  - Linear Probing Hashing
  - Robinhood Hashing (not covered)
  - Cuckoo Hashing

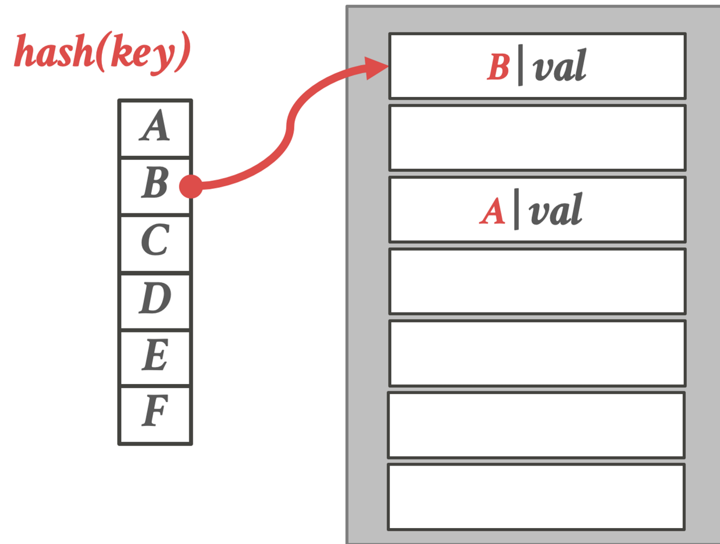
# Linear Probing Hashing

- Single giant table of slots
- Resolve collisions by linearly searching for the next free slot in the table.
    - To determine whether an element is present, hash to a location in the index and scan for it.
    - Has to store the key in the index to know when to stop scanning
    - Insertions and deletions are generalizations of lookups
- Example: Google's absl::flat_hash_map

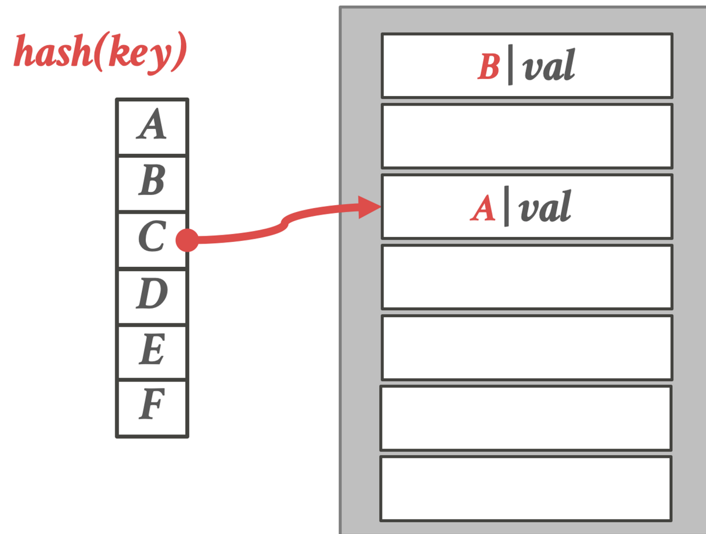
# Linear Probing Hashing



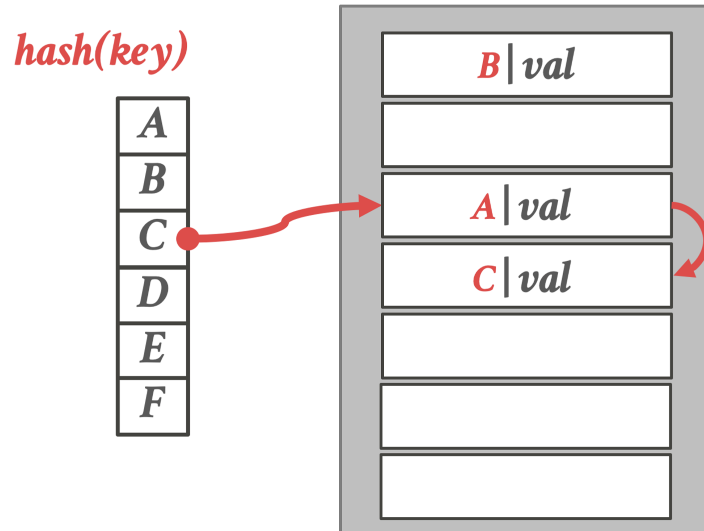
# Linear Probing Hashing



# Linear Probing Hashing



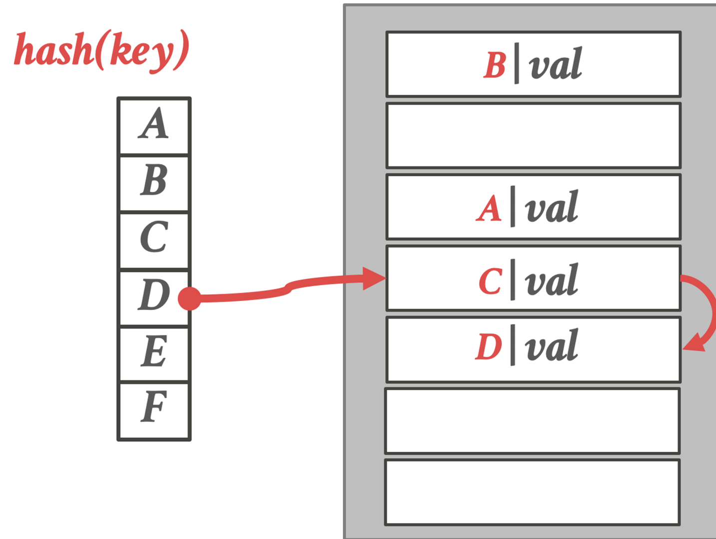
# Linear Probing Hashing





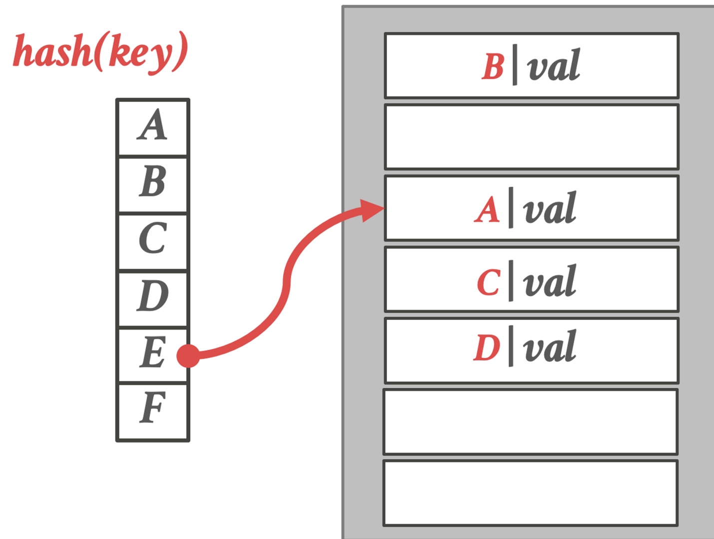

Slides adapted from CMU CS 15-445/645 by Andy Pavlo

# Linear Probing Hashing




Slides adapted from CMU CS 15-445/645 by Andy Pavlo

# Linear Probing Hashing

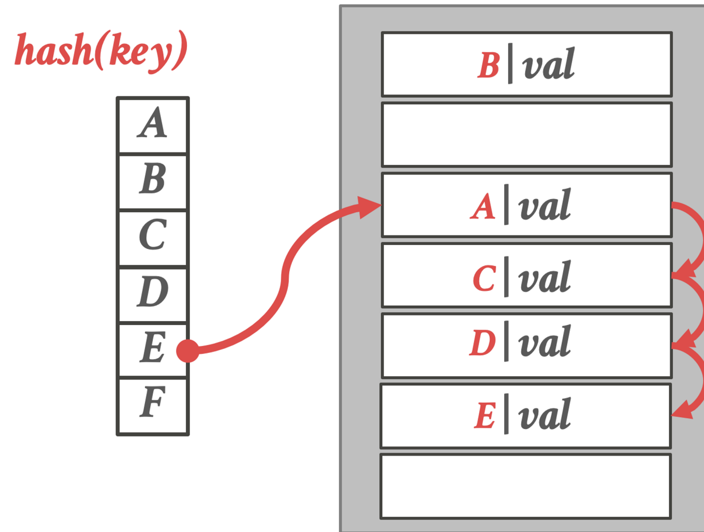



Q: What would happen in this case?

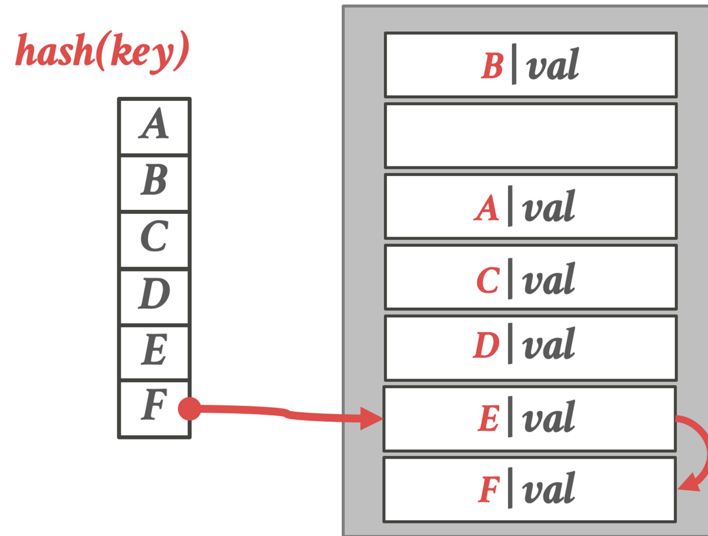


Slides adapted from CMU CS 15-445/645 by Andy Pavlo

# Linear Probing Hashing



# Linear Probing Hashing

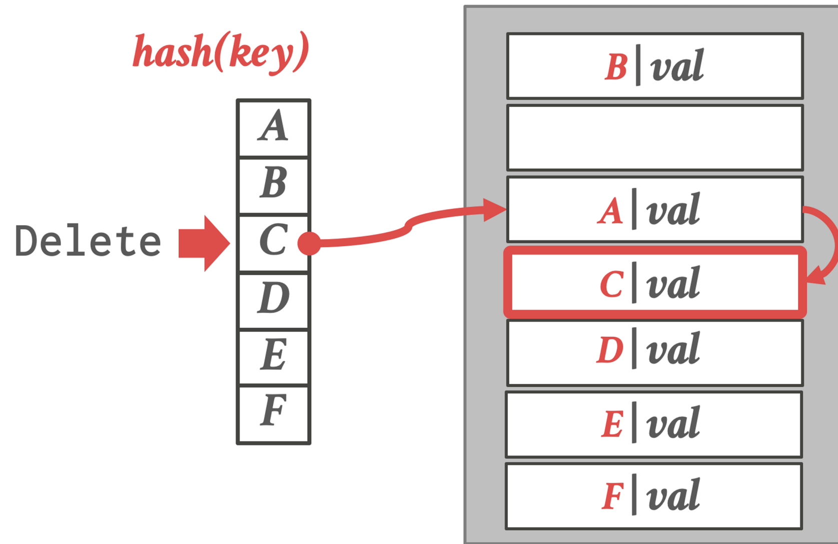


# Linear Probing Hashing - Delete

- It is not sufficient to simply delete the key
- This would affect searches for keys that have a hash value earlier than the emptied cell, but are stored in a position later than the emptied cell.
- Two solutions:
  - Tombstone
  - Movement (less common)


Slides adapted from CMU CS 15-445/645 by Andy Pavlo

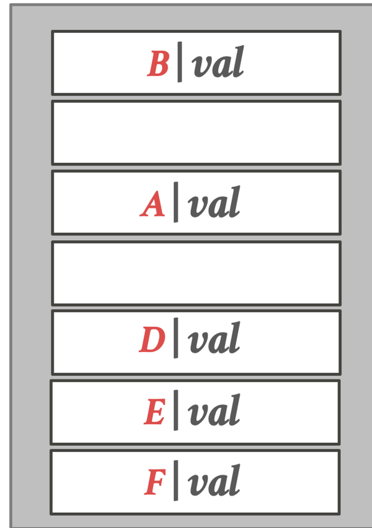
# Linear Probing Hashing



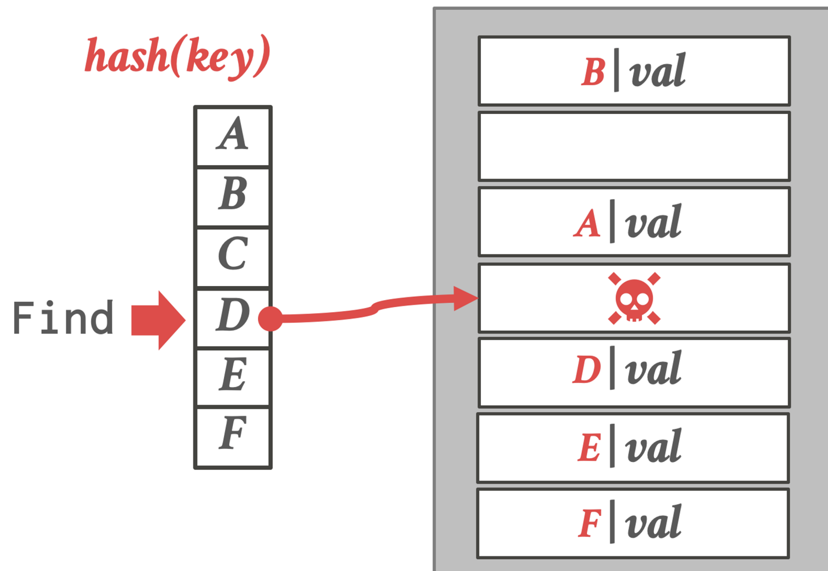
# Linear Probing Hashing

*hash(key)*

| A |
| --- |
| B |
| C |
| D |
| E |
| F |

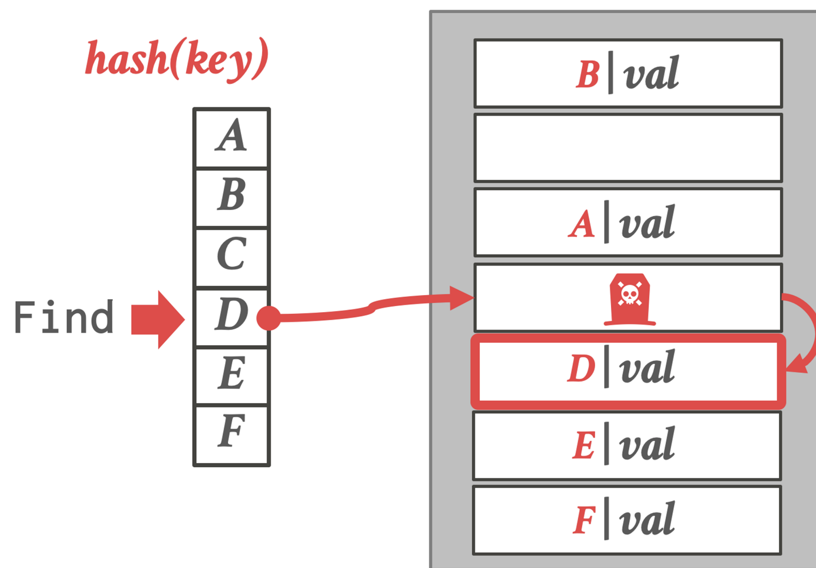


# Linear Probing Hashing



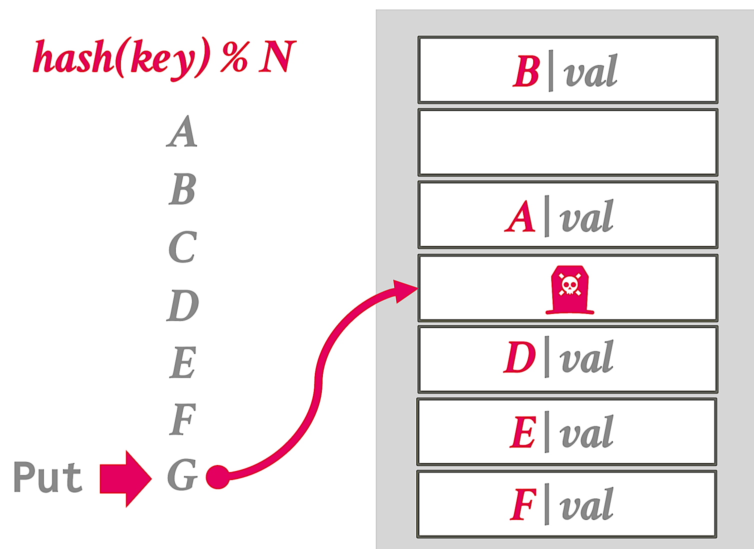

- Set a marker to indicate that the entry in the slot is logically deleted.

# Linear Probing Hashing



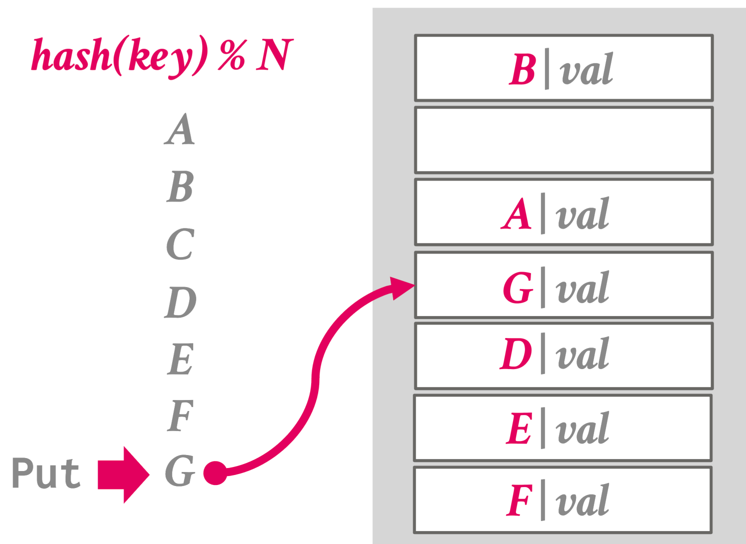

- Set a marker to indicate that the entry in the slot is logically deleted.

# Linear Probing Hashing




- Set a marker to indicate that the entry in the slot is logically deleted.
- Reuse the slot for new keys

# Linear Probing Hashing




- Set a marker to indicate that the entry in the slot is logically deleted.
- Reuse the slot for new keys

Slides adapted from CMU CS 15-445/645 by Andy Pavlo

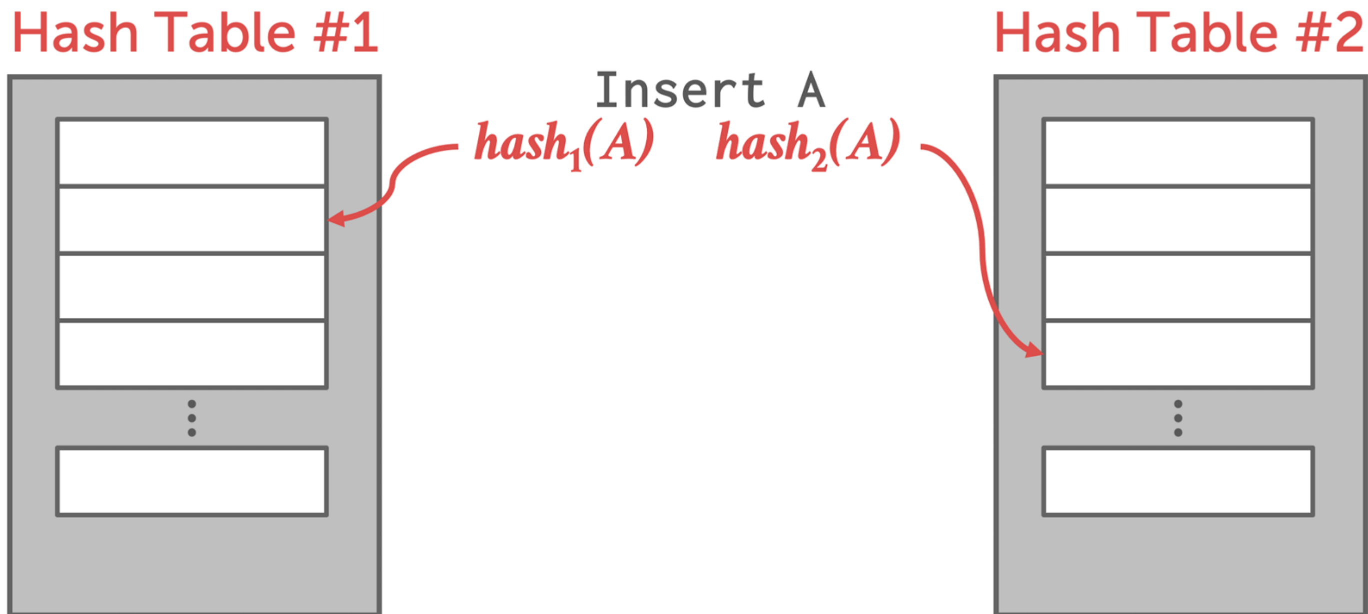
# Cuckoo Hashing

- Power of 2 choices: Use multiple hash tables with different seeds
  - On insert, check every table and pick one with a free slot
  - If no table has a free slot, evict the element from one of then and then re-hash it to find a new location
  - In rare cases, we may end up in a cycle. If this happens, we can rebuild using larger hash tables

- Look-ups and deletions are ~O(1) because only one location per hash table is checked.

Image source: https://theconversation.com/egg-colours-make-cuckoos-masters-of-disguise-34217

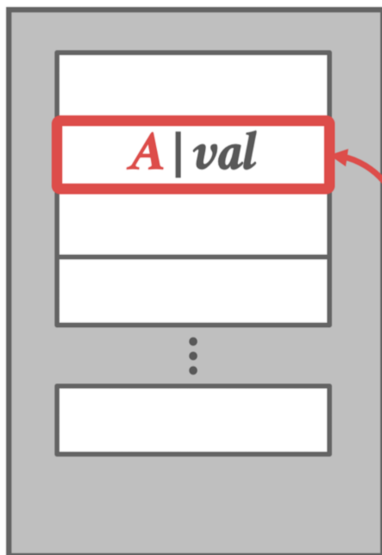
# Cuckoo Hashing




Slides adapted from CMU CS 15-445/645 by Andy Pavlo

# Cuckoo Hashing

Hash Table #1



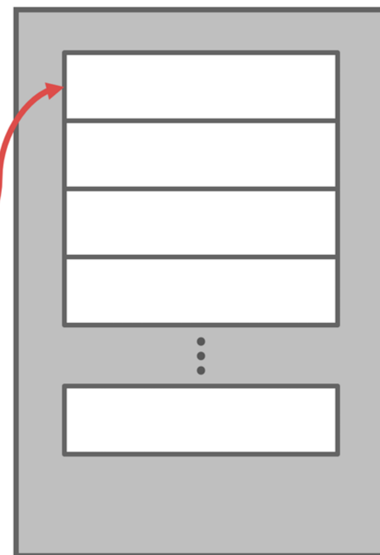

Insert A

*$hash_1(A)$* *$hash_2(A)$*

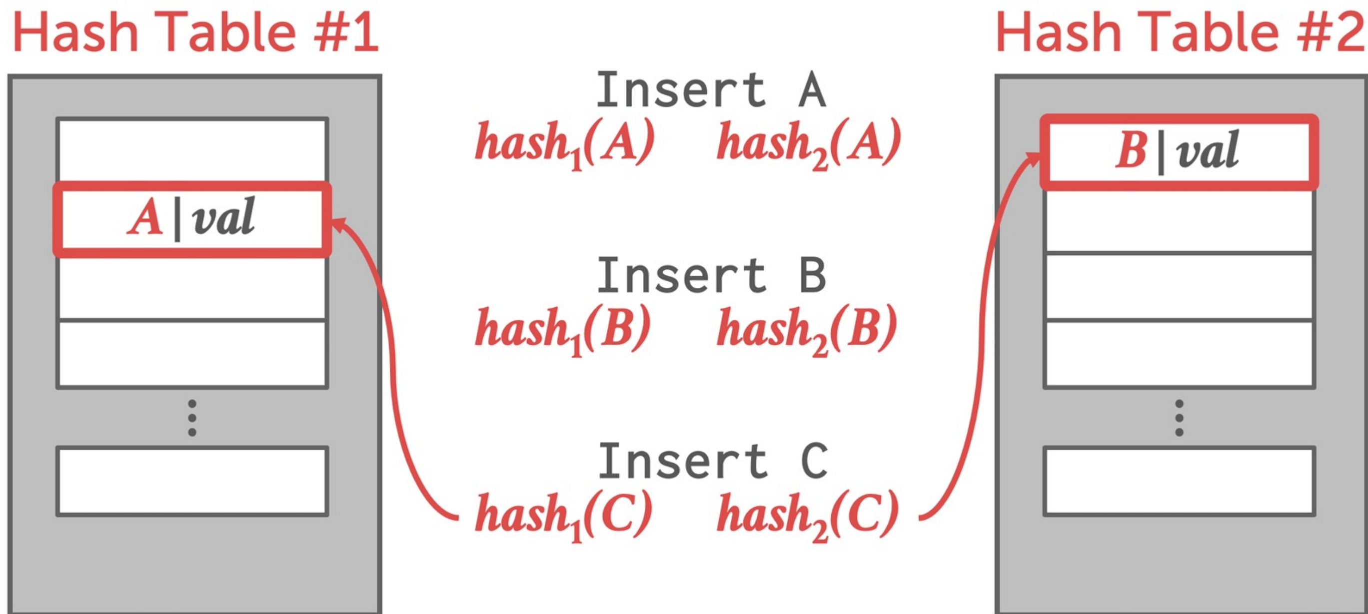
Insert B

*$hash_1(B)$* *$hash_2(B)$*

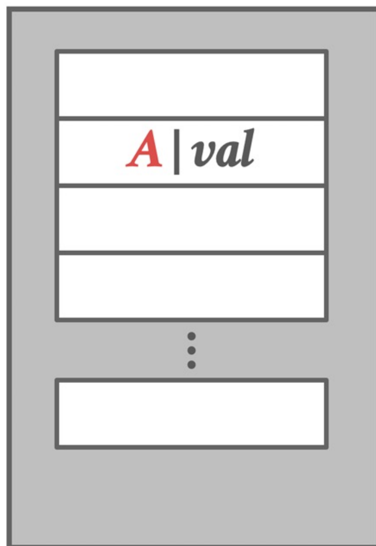
Hash Table #2



# Cuckoo Hashing



# Cuckoo Hashing




Insert A
$hash_1(A)$ $hash_2(A)$

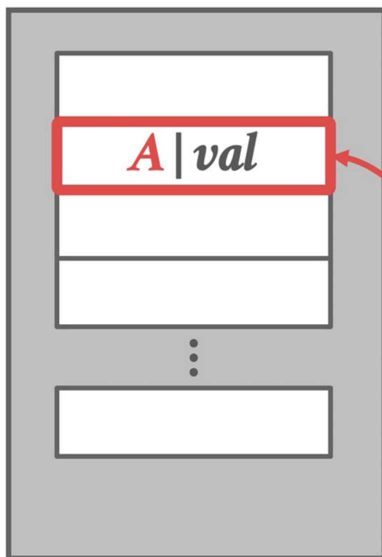
Insert B
$hash_1(B)$ $hash_2(B)$

Insert C
$hash_1(C)$ $hash_2(C)$

Slides adapted from CMU CS 15-445/645 by Andy Pavlo

# Cuckoo Hashing

Hash Table #1




Insert A

*$hash_1(A)$* *$hash_2(A)$*

Insert B

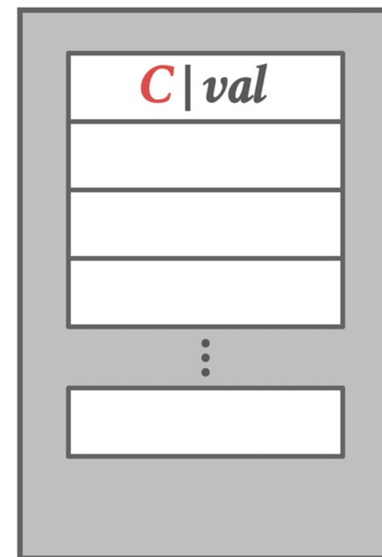
*$hash_1(B)$* *$hash_2(B)$*

Insert C

*$hash_1(C)$* *$hash_2(C)$*

*$hash_1(B)$*

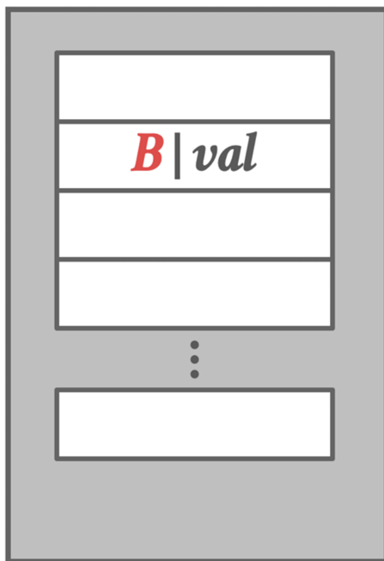
Hash Table #2






Slides adapted from CMU CS 15-445/645 by Andy Pavlo

# Cuckoo Hashing




Insert A

$hash_1(A)$ $hash_2(A)$

Insert B

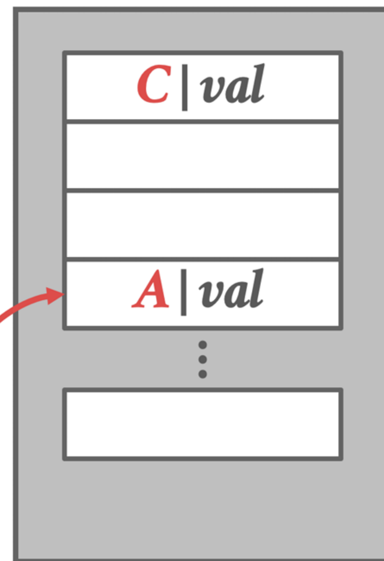
$hash_1(B)$ $hash_2(B)$

Insert C

$hash_1(C)$ $hash_2(C)$

$hash_1(B)$

$hash_2(A)$



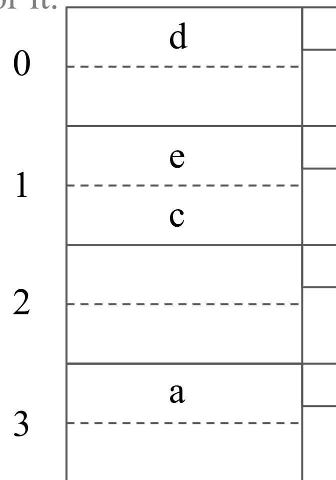
# Dynamic hash table

- The previous hash tables require the DBMS to know the number of elements it wants to store.
  - Otherwise it needs to rebuild the table to resize
- Dynamic hash tables incrementally resize the hash table on demand without needing to rebuild the entire table.
- Examples
  - Chained Hashing
  - Extensible Hashing
  - Linear Hashing

# Chained Hashing

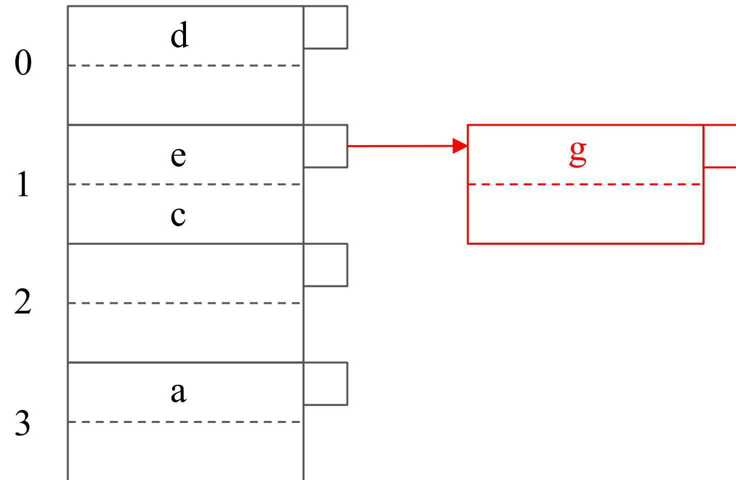
- Maintain a linked list of buckets for each slot in the hash table.

- Resolve collisions by placing all elements with the same hash key into the same bucket.
  - To determine whether an element is present, hash to its bucket and scan for it.
  - Insertions and deletions are generalizations of lookups.



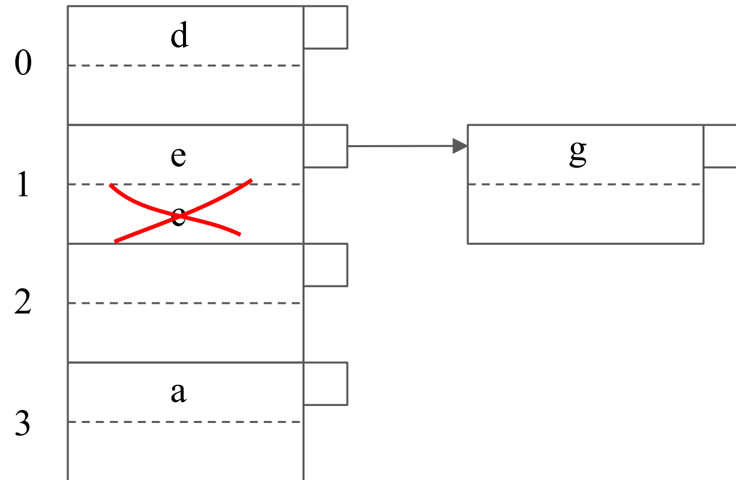
# Chained Hashing

- Add g where $h(g) = 1$



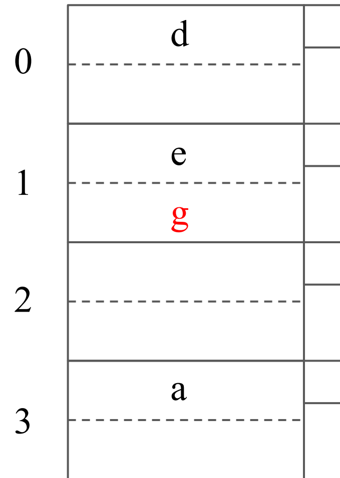
# Chained Hashing

- Remove c where $h(c) = 1$



# Chained Hashing

- Remove c where $h$(c) = 1



# Extendible Hashing

- Chained-hashing approach that splits buckets incrementally instead of letting the linked list grow forever.
  - Long chains of blocks -> many disk I/Os
- Multiple slot locations can point to the same bucket chain.
- Reshuffle bucket entries on split and increase the number of bits to examine.
  - Data movement is localized to just the split chain.

# Extensible hash table

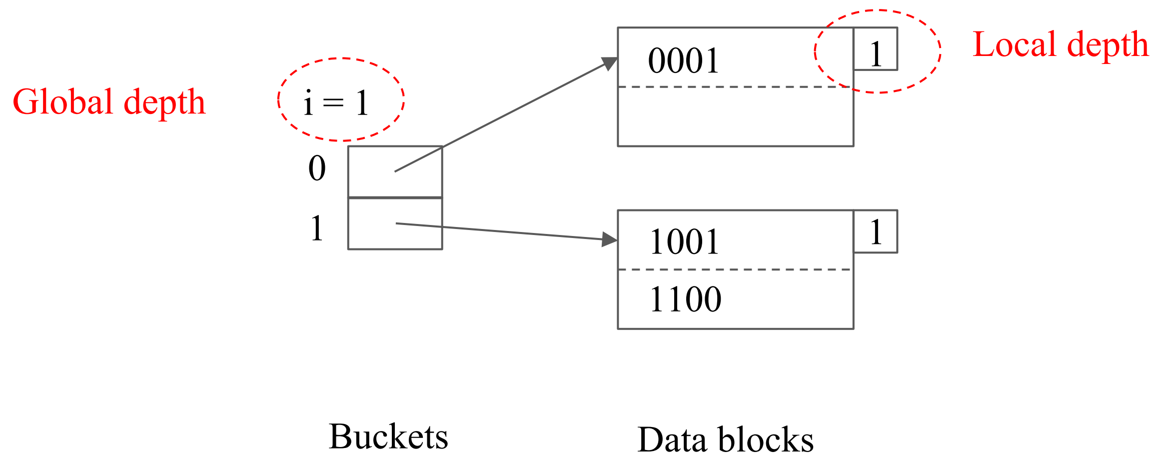
- Use first $i$ bits of hash value to locate block
  - $i$ grows over time

$i = 3$

h(key): 00101100

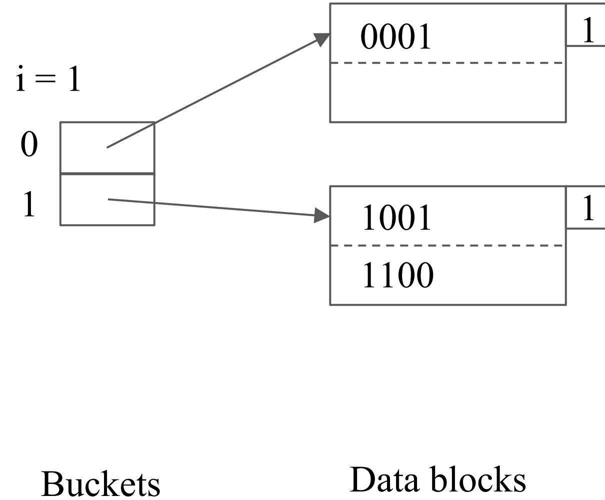
# Extensible hash table

- Use level of indirection where buckets are pointers to blocks



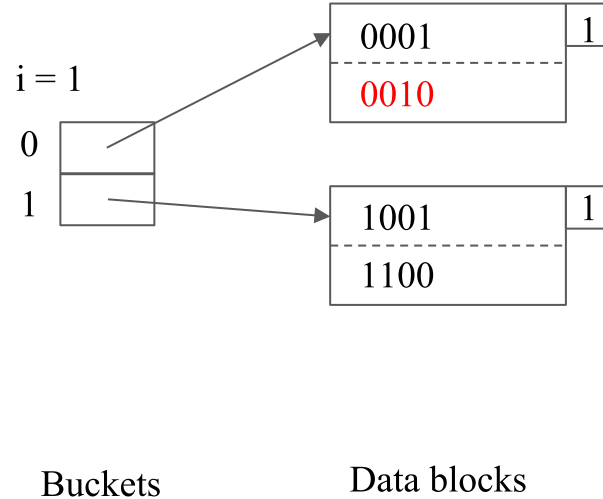
# Extensible hash table

- Add 0010



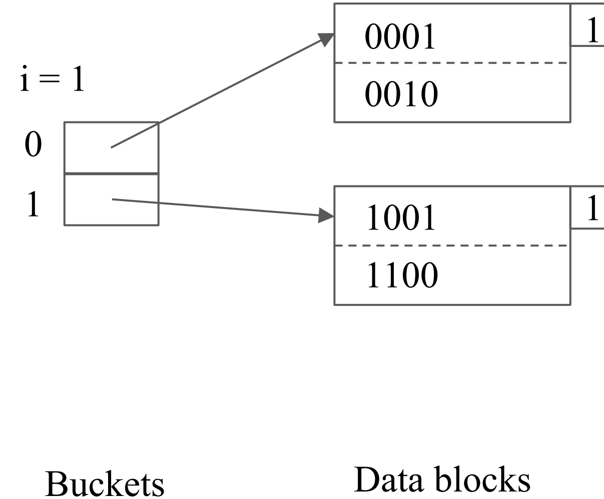
# Extensible hash table

- Add 0010



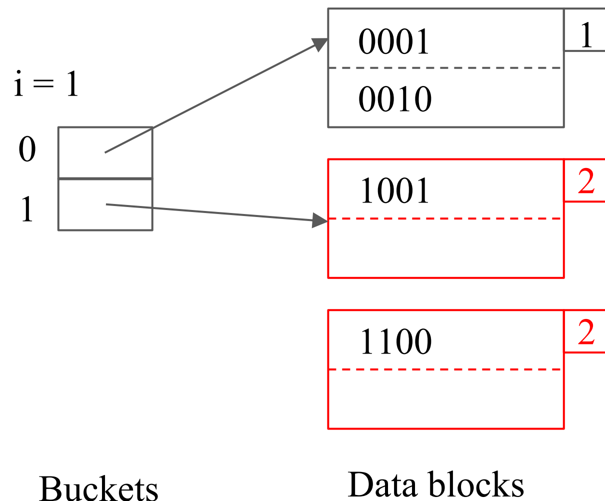
# Extensible hash table

- Add 1010



# Extensible hash table

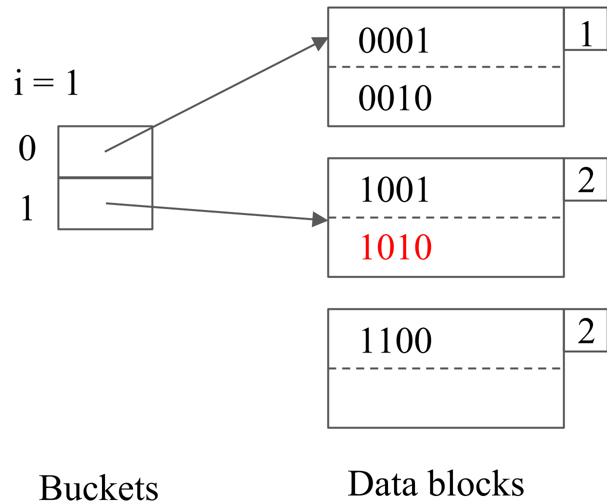
- Add 1010




May need to repeat splitting until there is space

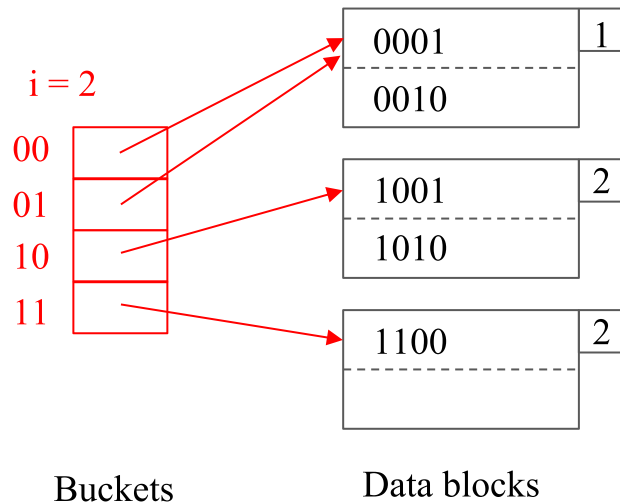
# Extensible hash table

- Add 1010



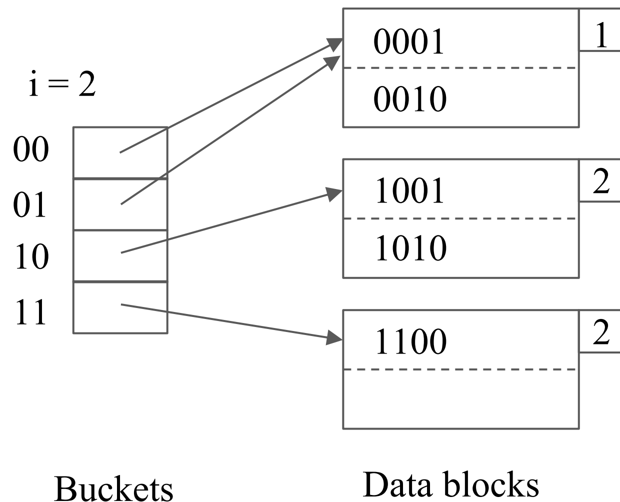
# Extensible hash table

- Add 1010



# Extensible hash table

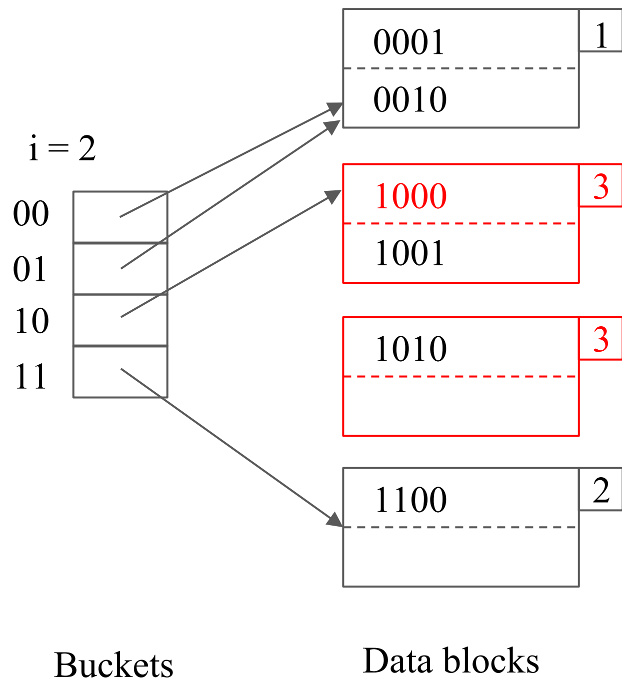
- Add 1000




Q: What will happen in this case?

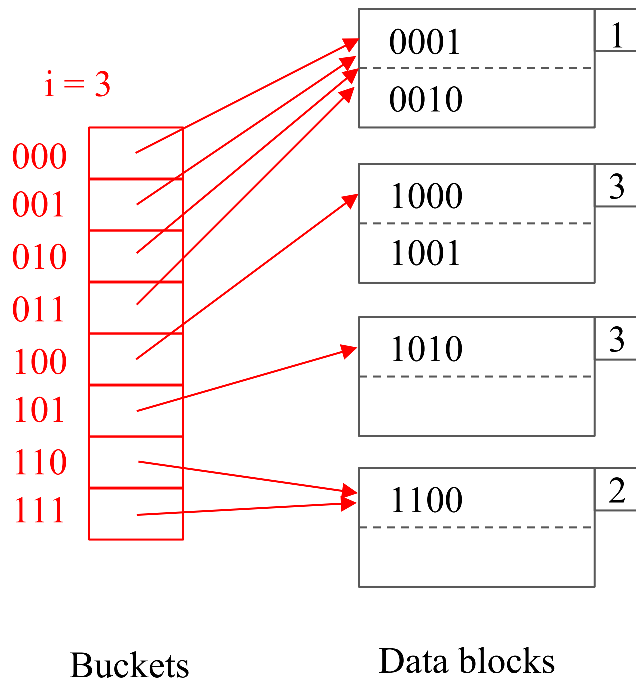
# Extensible hash table

- Add 1000



# Extensible hash table

- Add 1000



# Extensible hash table

- Deletion: the “reverse” of insertion
  - However, merging blocks and reducing the buckets is optional

# Extensible hashing summary

- If bucket array fits in memory, lookup is always 1 disk I/O
- Can grow table with little wasted space and avoiding full reorganizations
- However, doubling the bucket array is expensive
  - Splitting can occur frequently if the number of records per block is small
  - At some point, the bucket array may not fit in memory
- Linear hashing (covered next) grows the number of buckets more slowly

# Linear hashing

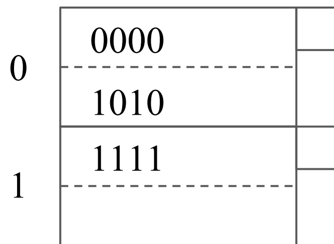
- The hash table maintains a pointer that tracks the next bucket to split.
  - When any bucket overflows, split the bucket at the pointer location.
- Use multiple hashes to find the right bucket for a given key.
- Can use different overflow criterion:
  - Space Utilization
  - Average Length of Overflow Chains

# Linear hash tables

- Use last *i* bits of hash value to locate block
- Hash table grows linearly

| | |
|---|---|
| # bits used | i = 1 |
| # buckets | n = 2 |
| # records | r = 3 |

Policy: limit $r \leq 1.7n$



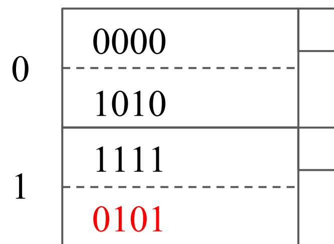
# Linear hash tables

- Add 0101

| | |
|---|---|
| # bits used | $i = 1$ |
| # buckets | $n = 2$ |
| # records | $r = 4$ |

Policy: limit $r \leq 1.7n$

Violation

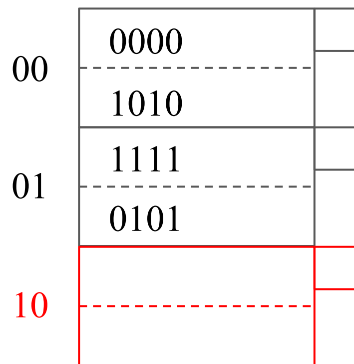


# Linear hash tables

- Add 0101

| # bits used | $i = 2$ |
|---|---|
| # buckets | $n = 3$ |
| # records | $r = 4$ |

Policy: limit $r \leq 1.7n$

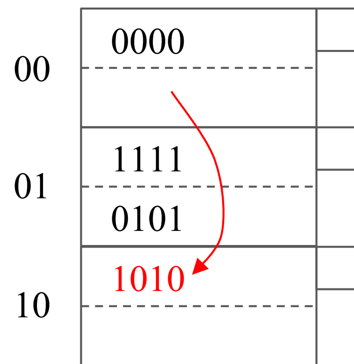


# Linear hash tables

- Add 0101

| | |
|---|---|
| # bits used | i = 2 |
| # buckets | n = 3 |
| # records | r = 4 |

Policy: limit $r \leq 1.7n$

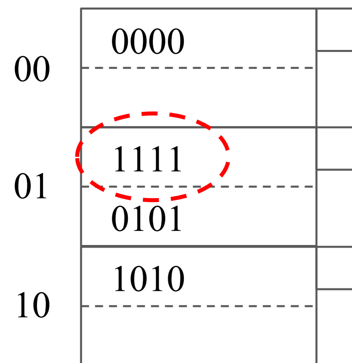


# Linear hash tables

- Add 0101

| # bits used | i = 2 |
| --- | --- |
| # buckets | n = 3 |
| # records | r = 4 |

Policy: limit $r \leq 1.7n$



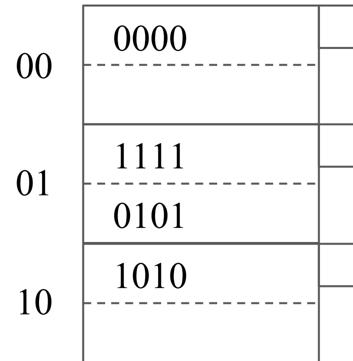

1111 stays here because there is no 11 bucket yet

# Linear hash tables

- Add 0001

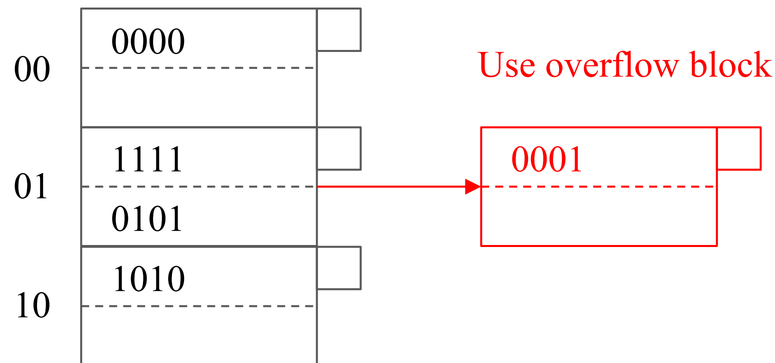
| # bits used | i = 2 |
|---|---|
| # buckets | n = 3 |
| # records | r = 4 |

Policy: limit $r \leq 1.7n$



# Linear hash tables

- Add 0001



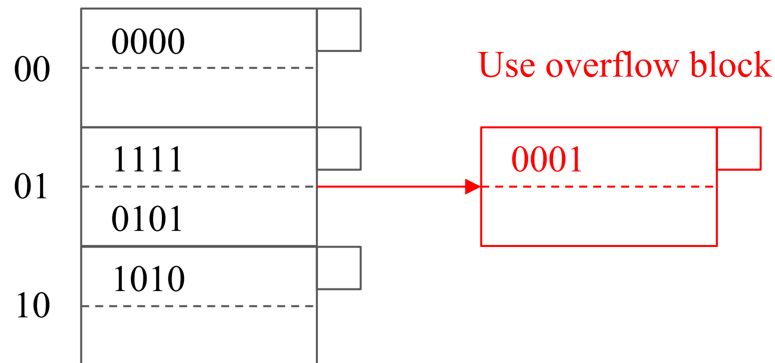
# Linear hash tables

- Add 0001

# bits used | i = 2
# buckets | n = 3
# records | r = 5

Policy: limit r ≤ 1.7n

No violation

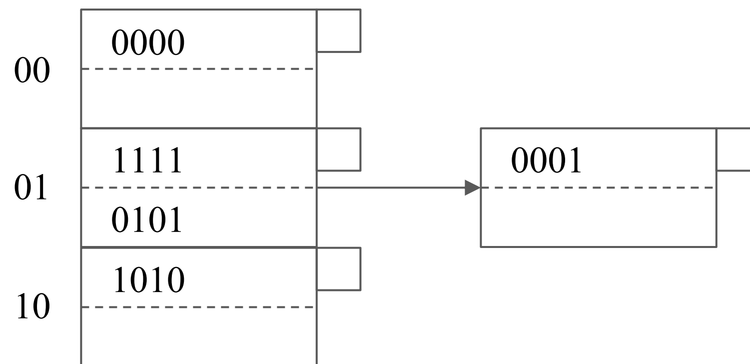


# Exercise #1

- Continuing with example, add 0111

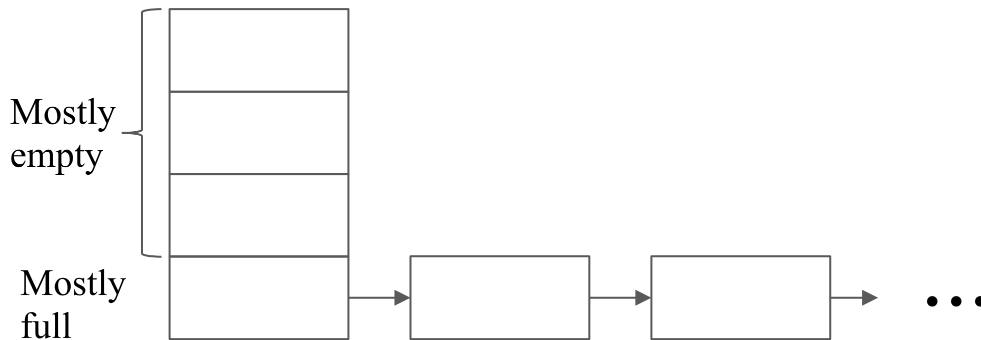
| | |
|---|---|
| # bits used | i = 2 |
| # buckets | n = 3 |
| # records | r = 5 |

Policy: limit r ≤ 1.7n



# Linear hashing summary

- Can grow table with little wasted space and avoiding full reorganizations
- Compared to extensible hashing, there is no array of buckets
- However, there can be a long chain of overflow blocks



# Indexing vs hashing

- Indexing (including B trees) is good for range lookups
- Hashing is good for equality-based point lookups

```
SELECT *
FROM Movies
WHERE year >= 2000;
```

```
SELECT *
FROM Movies
WHERE title = ‘Ponyo’;
```

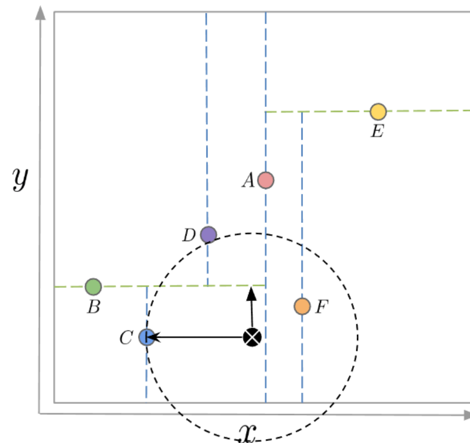
# Multidimensional Indexes (14.4)

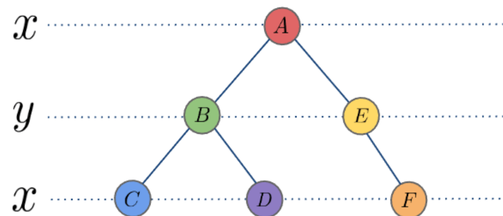
All the index structures discussed so far are one dimensional

- Assume a single search key, and they retrieve records that match a given search key value.
- The key can contain multiple attributes

Examples:

- Kd-tree, r-tree

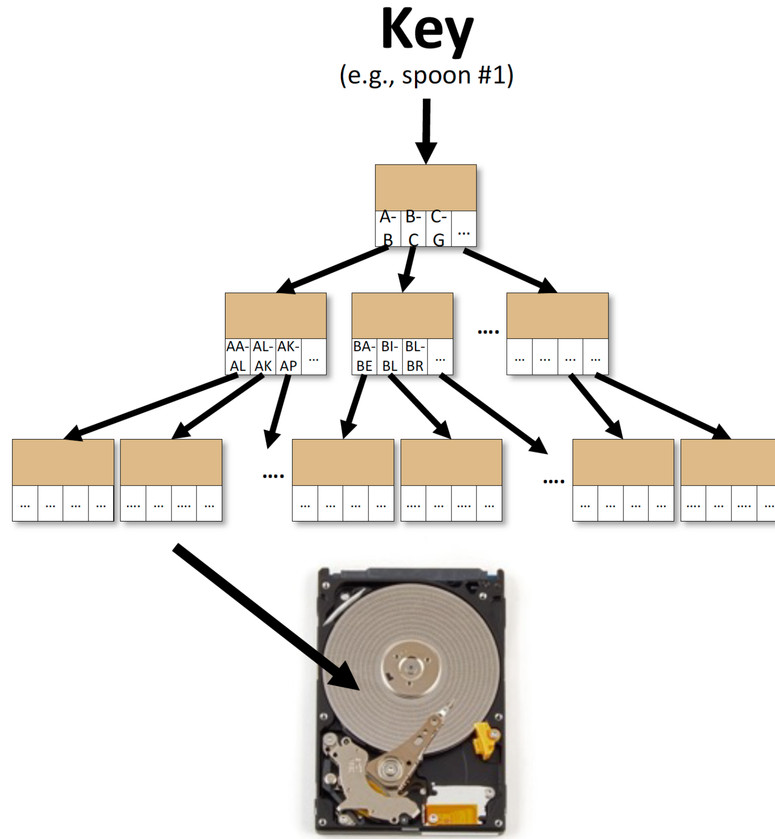


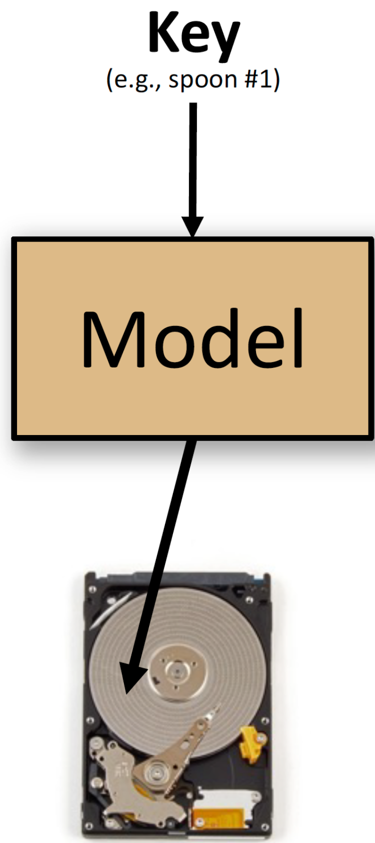

* We will discuss this more in the nearest neighbor search lecture

# A brief intro to learned index structures

Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska



Key
(e.g., spoon #1)
A-B
B-C
C-G
AA-AL
AL-AK
AK-AP
BA-BE
BI-BL
BL-BR


Key
(e.g., spoon #1)
Model

# How To Build Models To Predict the Location

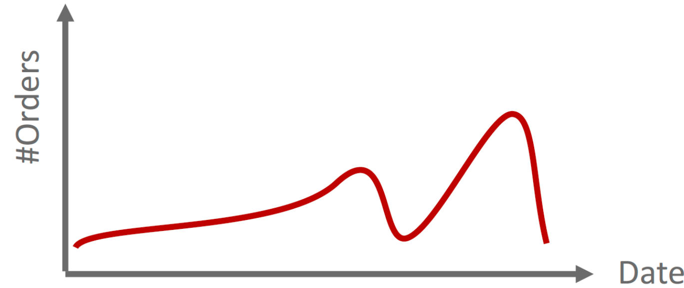
| id | date | first_name | last_name | email | address | zip | state | credit_card_nb | amount |
|---|---|---|---|---|---|---|---|---|---|
| 1000 | 2017-01-01 | Hobart | Spracklin | hspracklin0@dailymotion.com | 20565 High Crossing Plaza | 56372 | Minnesota | 4405-6975-7285-5160 | $ 611.00 |
| 1001 | 2017-01-02 | Billye | Binnion | bbinnion1@123-reg.co.uk | 3698 Upham Point | 20260 | District of Columbia | 3533-7150-7728-9850 | $ 244.00 |
| 1002 | 2017-01-02 | Johann | Brockley | jbrockley2@bizjournals.com | 23844 Artisan Place | 98516 | Washington | 67597-1193-7985-5100 | $ 233.00 |
| 1003 | 2017-01-03 | Artie | MacMenami | amacmenamin3@hao123.com | 6276 Toban Trail | 78759 | Texas | 3537-4829-6134-5000 | $ 210.00 |
| 1004 | 2017-01-03 | Delilah | O'Currigan | docurrigan4@chron.com | 86016 New Castle Avenue | 72199 | Arkansas | 3555-2017-2226-5780 | $ 286.00 |
| 1005 | 2017-01-04 | Gretta | Will | gwill5@yelp.com | 0 Dottie Circle | 68524 | Nebraska | 503844-1984-2085-5000 | $ 870.00 |
| 1006 | 2017-01-04 | Gordon | Kirsopp | gkirsopp6@utexas.edu | 64060 Scott Park | 20370 | District of Columbia | 633332-1895-2414-5000 | $ 687.00 |
| 1007 | 2017-01-05 | Bendick | Fagg | bfagg7@army.mil | 94 Florence Hill | 45440 | Ohio | 3528-9673-1815-8420 | $ 733.00 |
| 1008 | 2017-01-05 | Dimitry | Boyet | dboyet8@sakura.ne.jp | 35886 Golf Plaza | 30066 | Georgia | 3576-6991-4041-3170 | $ 382.00 |
| 1009 | 2017-01-06 | Ailsun | Beinke | abeinke9@si.edu | 1 Badeau Place | 46295 | Indiana | 56022-2011-8072-1400 | $ 854.00 |
| 1010 | 2017-01-07 | Lou | Hallows | lhallowsa@theguardian.com | 1 Twin Pines Junction | 91125 | California | 5602-2364-4079-0250 | $ 150.00 |
| 1011 | 2017-01-09 | Tiffani | Mathew | tmathewb@seattletimes.com | 0456 Meadow Vale Lane | 75260 | Texas | 6387-6943-8910-4580 | $ 313.00 |
| 1012 | 2017-01-09 | Perl | Bridie | pbridiec@hubpages.com | 07 Bluestem Junction | 33124 | Florida | 3539-8662-2397-5880 | $ 558.00 |
| 1013 | 2017-01-09 | Rosabelle | Blasik | rblasikd@delicious.com | 7 Fairfield Pass | 79699 | Texas | 5602-2297-6599-8560 | $ 941.00 |
| 1014 | 2017-01-10 | Meggi | Belamy | mbelamye@ask.com | 0995 Manufacturers Street | 10170 | New York | 3557-5094-7405-8340 | $ 875.00 |
| 1015 | 2017-01-10 | Tadio | Balderston | tbalderstonf@apache.org | 80 Novick Road | 75260 | Texas | 60485-3728-7119-9300 | $ 954.00 |
| 1016 | 2017-01-11 | Gianina | Oxteby | goxtebyg@google.pl | 72674 Fuller Avenue | 89505 | Nevada | 4-0415-9268-2397 | $ 239.00 |
| 1017 | 2017-01-12 | Brendan | Doody | bdoodyh@craigslist.org | 87414 Golden Leaf Street | 11480 | New York | 201-6348-4121-1314 | $ 308.00 |
| 1018 | 2017-01-13 | Conway | Coombs | ccoombsi@blogger.com | 2810 Oakridge Park | 32859 | Florida | 3529-1514-0357-9120 | $ 60.00 |
| 1019 | 2017-01-14 | Germaine | Bere | gberej@bravesites.com | 82802 Oakridge Park | 20041 | District of Columbia | 670961-0240-4054-9000 | $ 95.00 |
| 1020 | 2017-01-15 | Davide | Tolcharde | dtolchardek@redcross.org | 89 Continental Avenue | 79165 | Texas | 5018-7748-4325-9510 | $ 137.00 |
| 1021 | 2017-01-16 | Nigel | Artharg | nartharg l@gizmodo.com | 31 Mcbride Point | 22301 | Virginia | 560225-6965-2870-0000 | $ 496.00 |
| 1022 | 2017-01-17 | Rickard | Trenholm | rtrenholmm@cbslocal.com | 93 Hoepker Parkway | 70593 | Louisiana | 3541-5241-5383-9970 | $ 760.00 |
| 1023 | 2017-01-18 | Juditha | Dwane | jdwanen@vk.com | 7914 Eliot Lane | 14276 | New York | 5456-4410-0914-3180 | $ 474.00 |
| 1024 | 2017-01-19 | Susan | Ilden | sildeno@aol.com | 25204 Huxley Road | 21684 | Maryland | 3574-8586-6367-9920 | $ 83.00 |
| 1025 | 2017-01-20 | Abbey | Triggle | atrigglep@google.com.au | 47 Debra Pass | 74184 | Oklahoma | 3538-6047-6315-7710 | $ 513.00 |
| 1026 | 2017-01-21 | Zsazsa | Dunster | zdunsterq@nature.com | 7 Gerald Alley | 40576 | Kentucky | 3562-0325-7709-3490 | $ 952.00 |
| 1027 | 2017-01-22 | Grantham | Friatt | gfriattr@seattletimes.com | 774 Prairieview Circle | 29225 | South Carolina | 3571-1171-9476-8780 | $ 942.00 |
| 1028 | 2017-01-22 | Ross | Gaudin | rgaudins@samsung.com | 3102 Loeprich Trail | 68197 | Nebraska | 5108-7578-4665-2710 | $ 572.00 |
| 1029 | 2017-01-22 | Aluino | Drover | adrovert@dagondesign.com | 2717 Northridge Avenue | 72199 | Arkansas | 670999-3171-8848-0000 | $ 318.00 |
| 1030 | 2017-01-23 | Shurlock | Braker | sbrakeru@huffingtonpost.com | 30783 Jenna Alley | 80945 | Colorado | 6331106-1894-9878-0000 | $ 166.00 |
| 1031 | 2017-01-24 | Glenda | Goodbody | ggoodbodyv@economist.com | 720 Pierstorff Way | 7522 | New Jersey | 36-0593-2719-1684 | $ 412.00 |



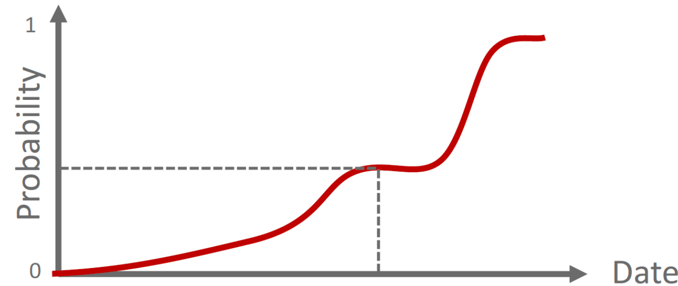
Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# How To Build Models To Predict the Location

Frequency Distribution




Cumulative Distribution Function (CDF)



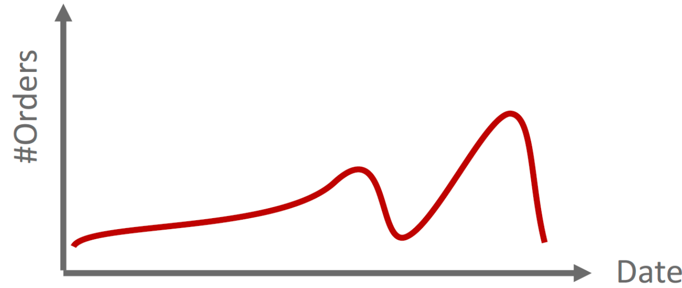

| date | 2017-01-01 | 2017-01-02 | 2017-01-02 | 2017-01-03 | 2017-01-03 | 2017-01-04 | 2017-01-04 | 2017-01-05 | 2017-01-05 | 2017-01-06 | 2017-01-07 | 2017-01-09 | 2017-01-09 | 2017-01-09 | 2017-01-10 | 2017-01-10 | 2017-01-11 | 2017-01-12 | 2017-01-13 | 2017-01-14 | 2017-01-15 | 2017-01-16 | 2017-01-17 | 2017-01-18 | 2017-01-19 | 2017-01-20 | 2017-01-21 | 2017-01-22 | 2017-01-22 | 2017-01-22 | 2017-01-23 | 2017-01-24 | 2017-01-24 | 2017-01-26 | 2017-01-26 | 2017-01-26 | 2017-01-28 | 2017-01-29 | 2017-01-30 | 2017-01-30 | 2017-01-30 | 2017-01-31 | 2017-01-31 | 2017-02-01 | 2017-02-01 | 2017-02-02 | 2017-02-04 | 2017-02-05 | 2017-02-05 | 2017-02-06 | 2017-02-06 | 2017-02-06 | 2017-02-07 | 2017-02-07 | 2017-02-08 | 2017-02-08 | 2017-02-08 | 2017-02-09 | 2017-02-10 | 2017-02-10 | 2017-02-11 | 2017-02-12 | 2017-02-13 | 2017-02-13 | 2017-02-14 | 2017-02-14 | 2017-02-15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



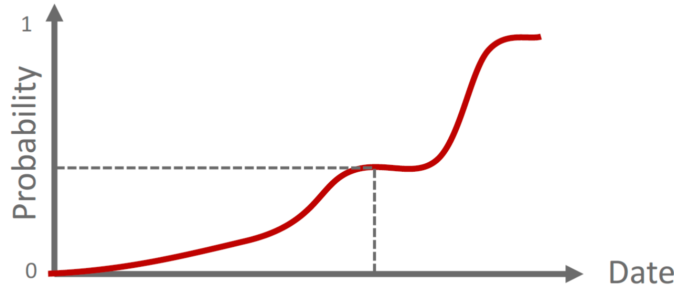
Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# How To Build Models To Predict the Location

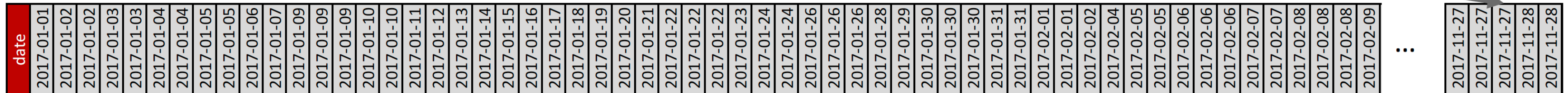
Frequency Distribution




Cumulative Distribution Function (CDF)




P(X<2017-11-27) * N

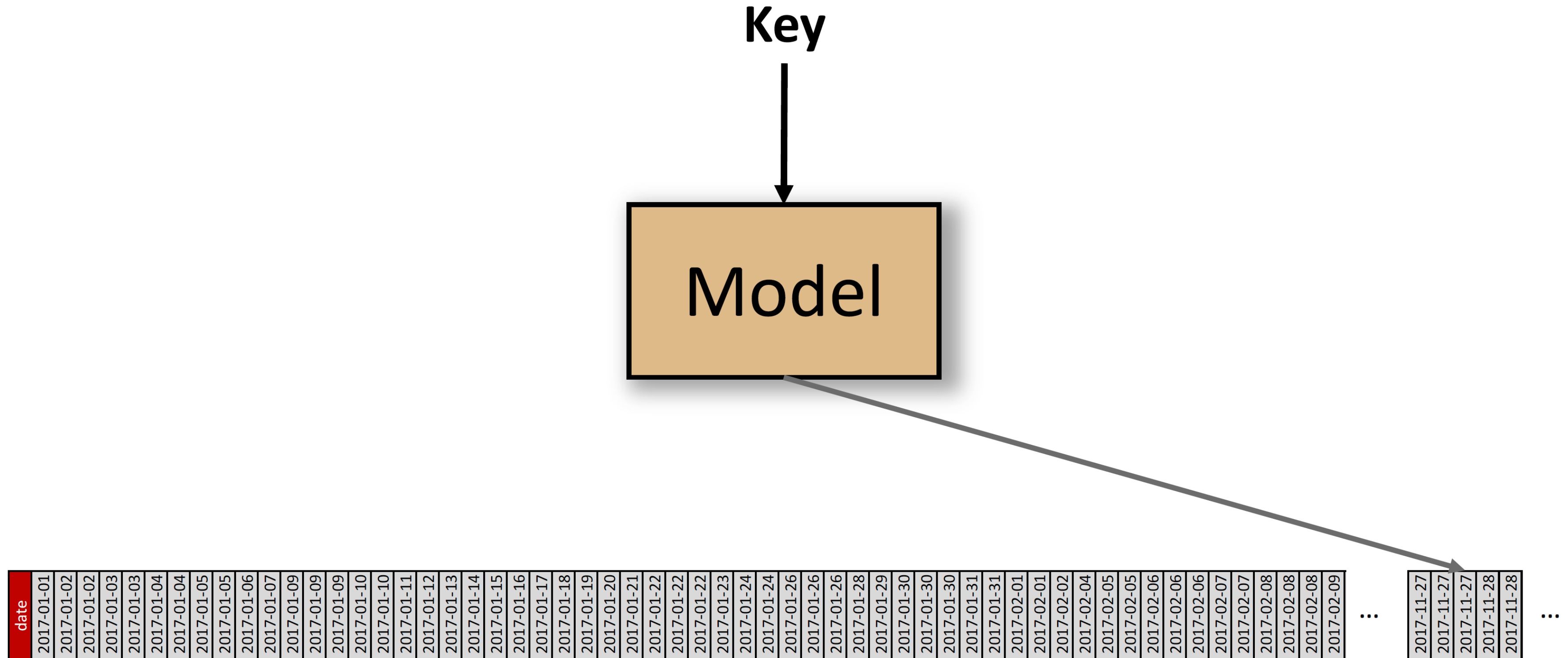






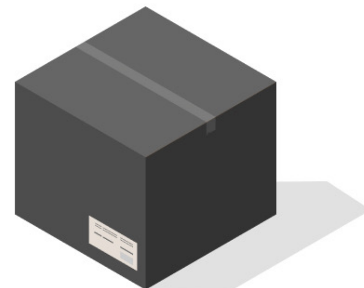
Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# CDF Model



# Key Insight

- Traditional data structures (typically) make no assumptions about the data
- But knowing the data distribution might allow for significant performance gains and might even change the complexity of data structures (e.g., O(log n)→O(1) for lookups)



Black box - we do not know anything

White box - we know everything

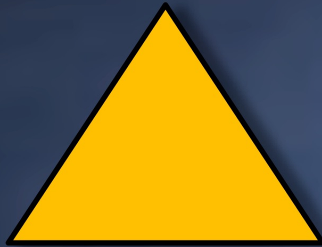
Image source: https://www.careerist.com/insights/a-guide-to-white-box-black-box-and-gray-box-testing

# Does it work? A first attempt

TensorFlow

- 200M web-server log records by timestamp-sorted
- 2 layer NN, 32 width, ReLU activated
- Prediction task: timestamp → position within sorted array

# Does It Work? A First Attempt

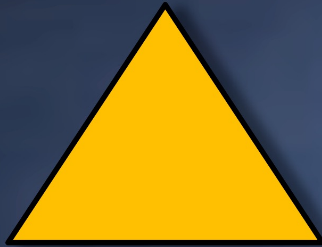


State-Of-The-Art B-Tree

**260ns**

**???**

# Does It Work? A First Attempt



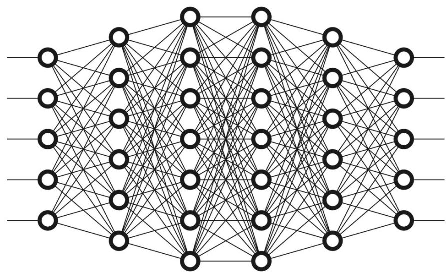
State-Of-The-Art
B-Tree

**260ns**

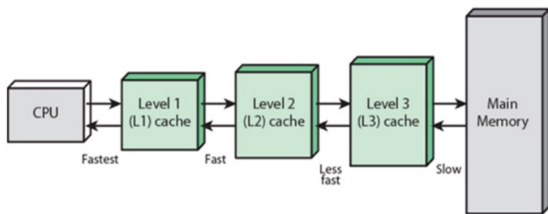
**>80,000ns**

# Reasons

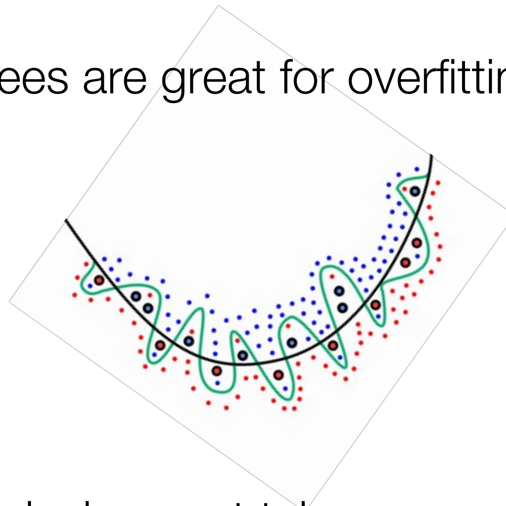
Tensorflow are not designed for nan—second execution



B-Trees are cache-efficient




B-Trees are great for overfitting



Search does not take advantage of the prediction

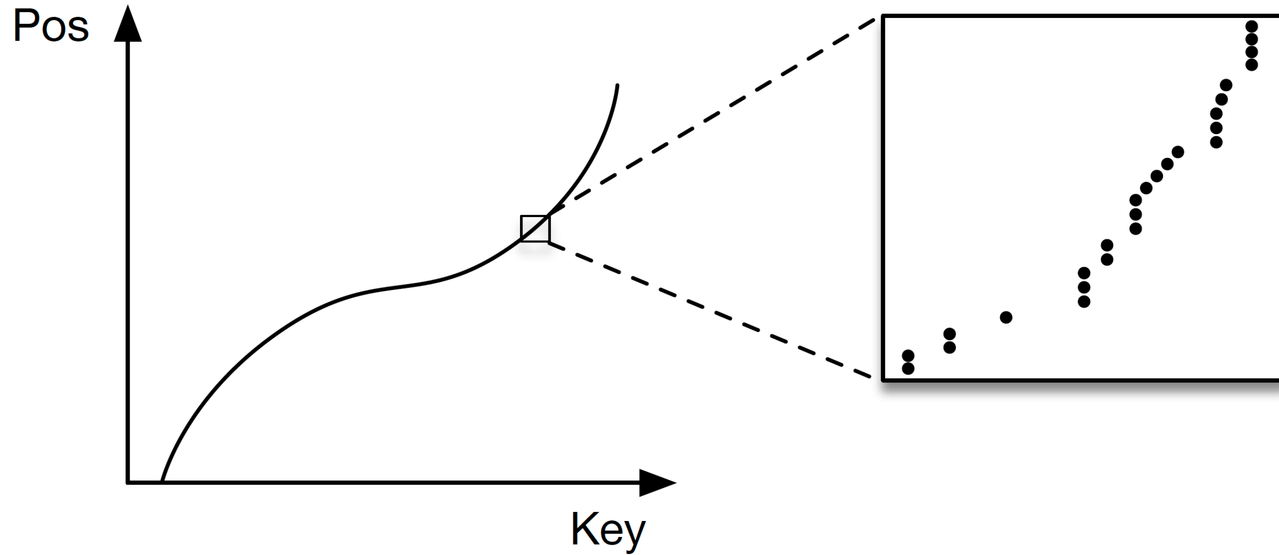
Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# The Learning Index Framework (LIF)

- An index synthesis system
- Given an index configuration generate the best possible code
- Uses ideas from Tupleware [VLDB15]
- Simple models are trained “on-the-fly”, whereas for complex models we use Tensorflow and extract weights afterwards (i.e., no Tensorflow during inference time)
- Best index configuration is found using auto-tuning (e.g., see TuPAQ [SOCC15]

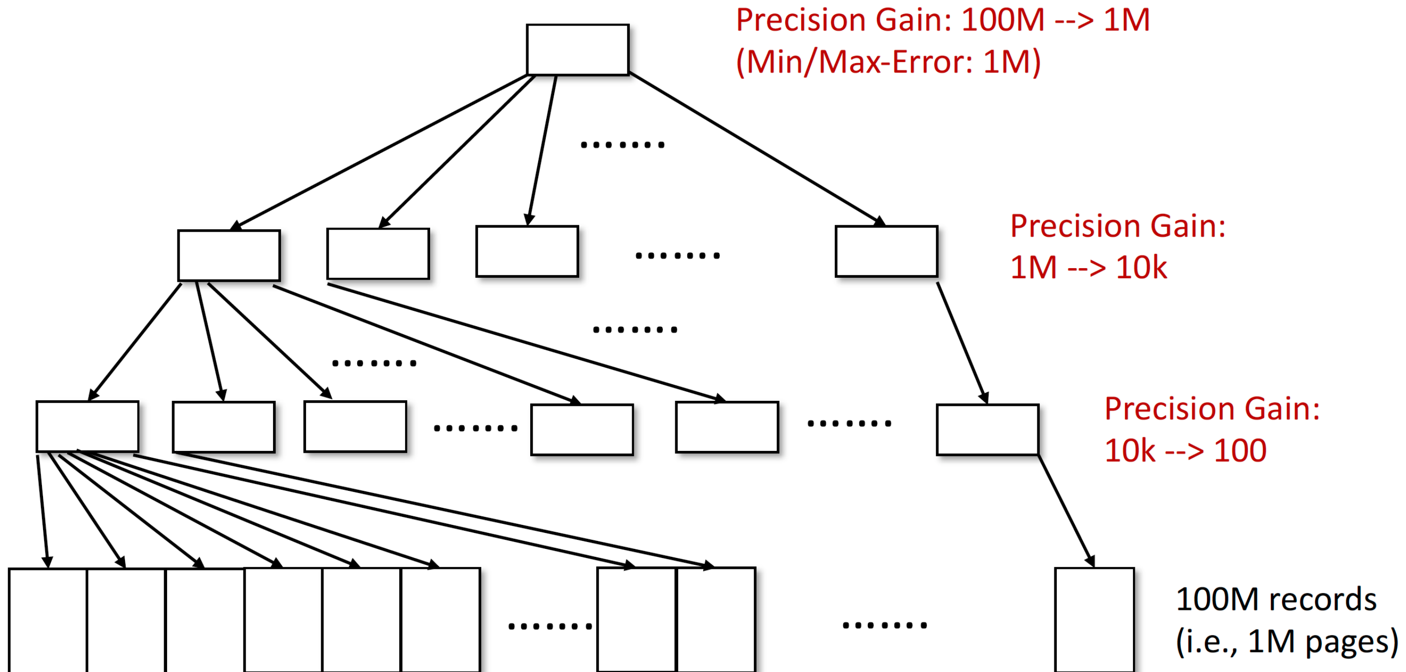
# Overfitting is a Good Thing

## *The Last Mile Problem*



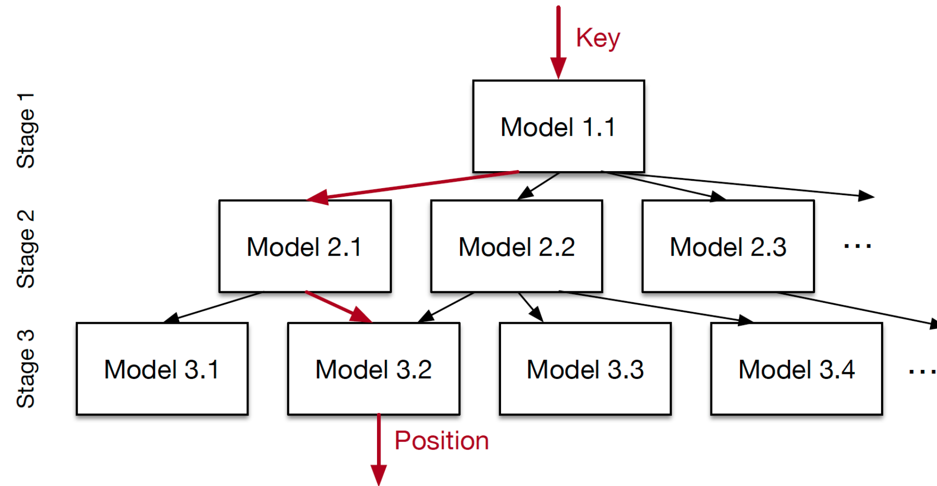
# Overfitting is a Good Thing

Index over 100M records. Page-size: 100




Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# Recursive-Model Index (RMI)



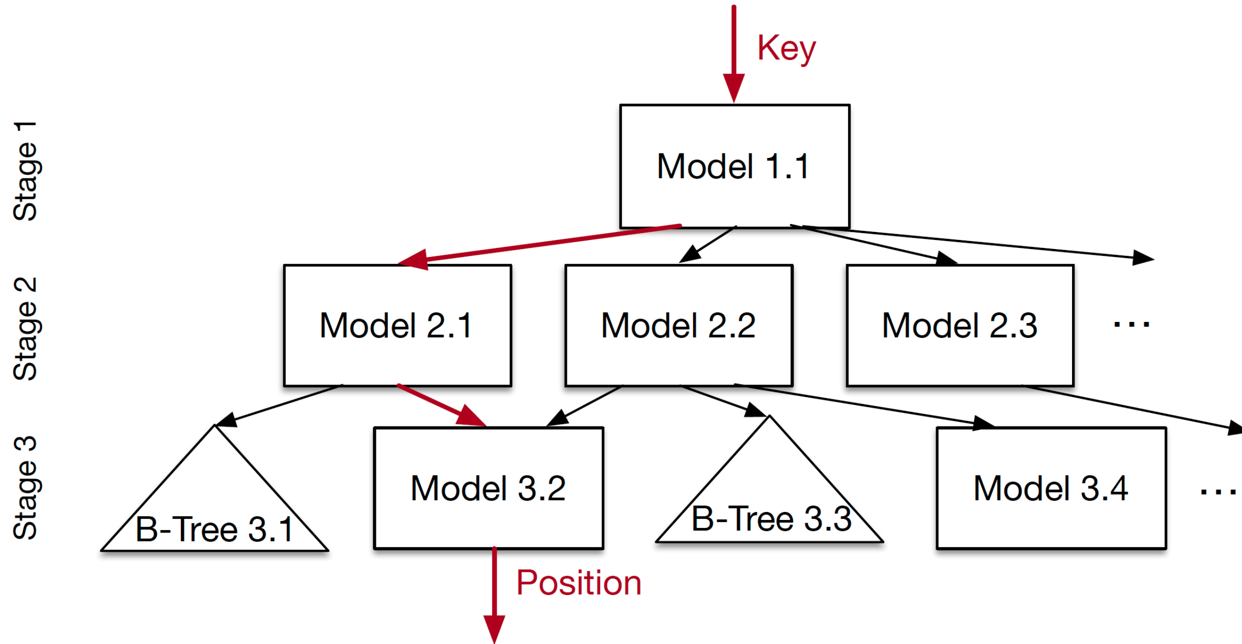

## 2-Stage RMI with Linear Model

```
pos₀ = a₀ + b₀ * key
pos₁ = m₁[pos₀].a + m₁[pos₀].b * key
record = local-search(key, pos₁)
```



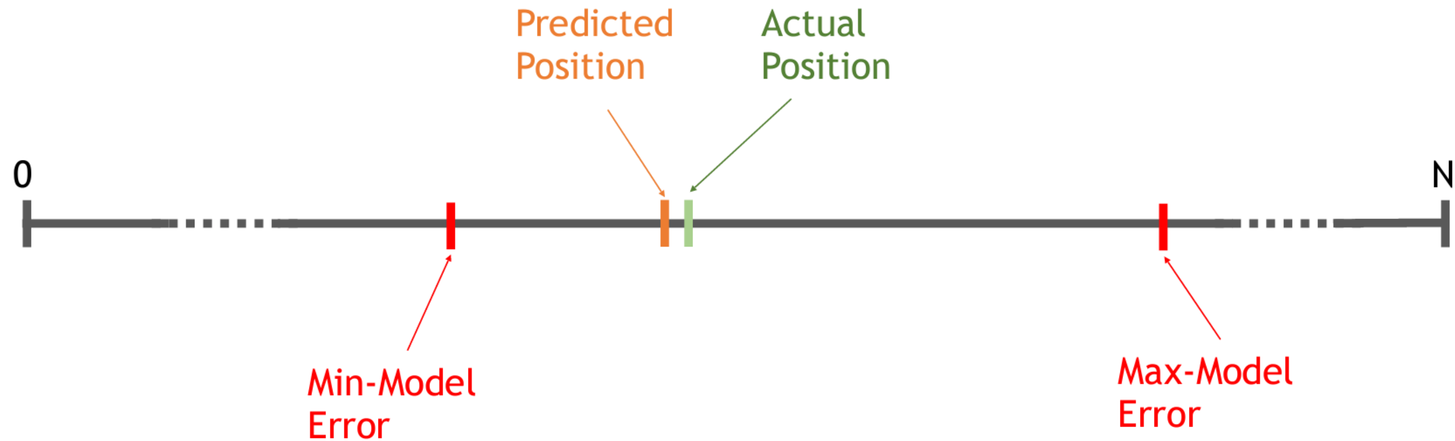
Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# Hybrid RMI




**Worst-Case Performance is the one of a B-Tree**

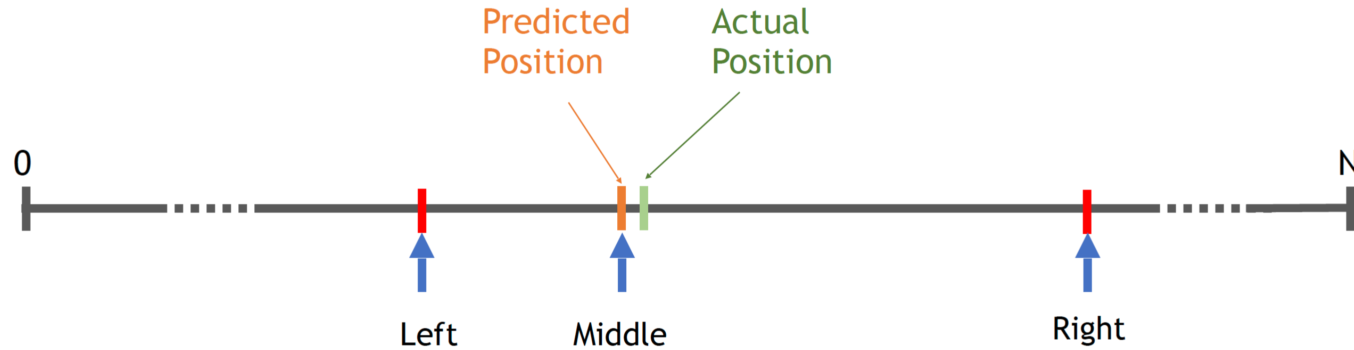
# Taking advantage of the prediction quality






Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

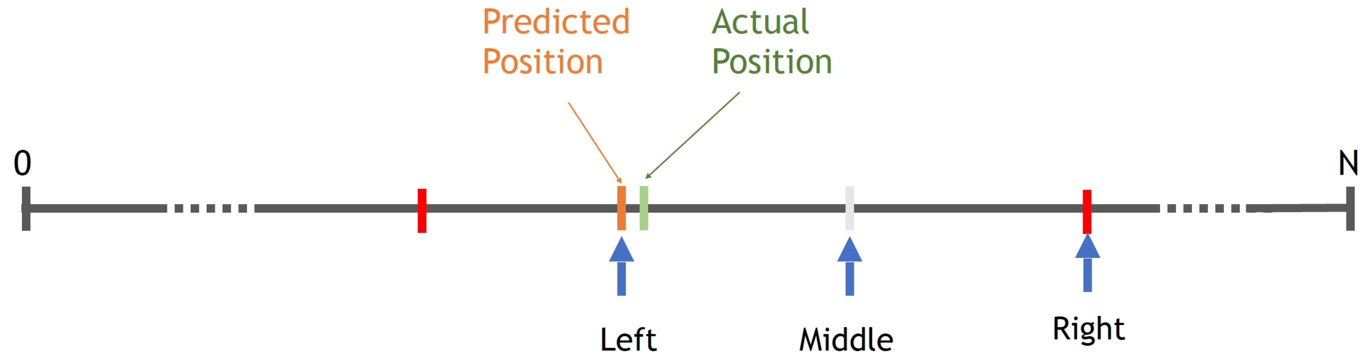
# Binary Search



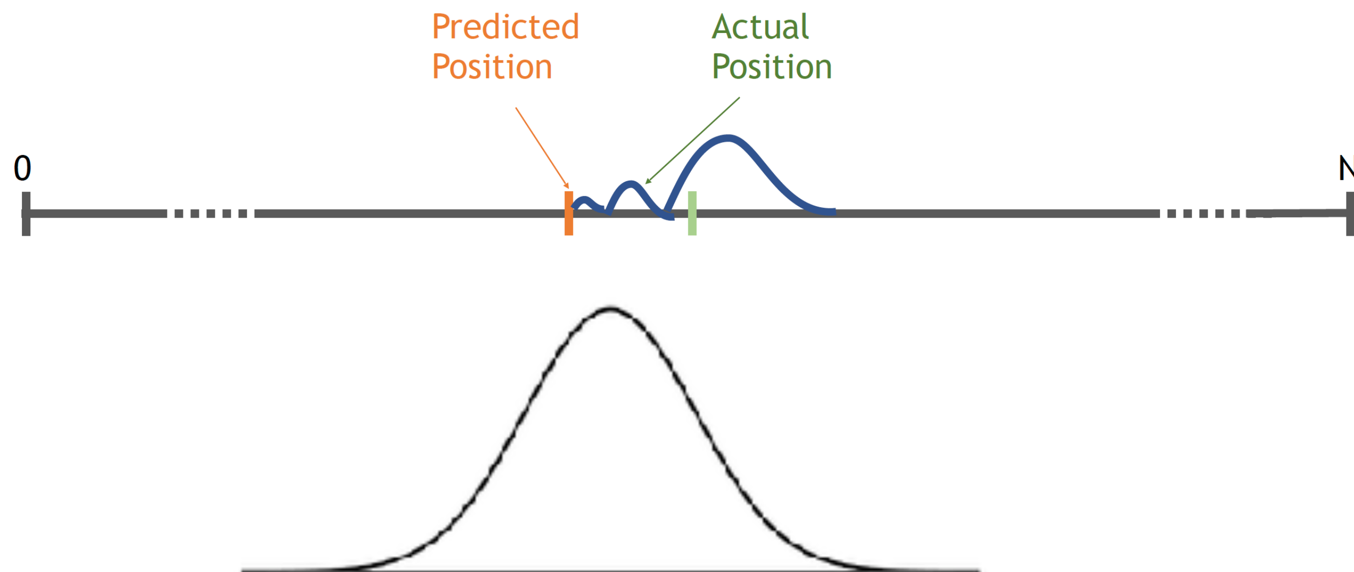



Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# Binary Search



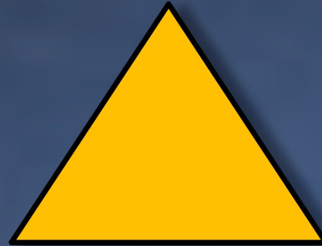
# Exponential Search






Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# Initial Results



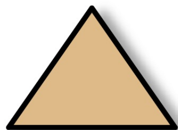
State-Of-The-Art B-Tree

Learned Index

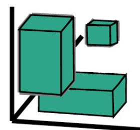
| TensorFlow | State-Of-The-Art B-Tree | Learned Index |
| --- | --- | --- |
| >80,000ns | 265ns | 85ns |
| | 13MB | 0.7MB |

# Fundamental Algorithms & Data Structures

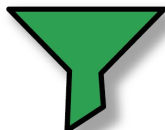
Tree



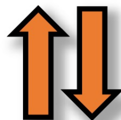
Multi-Dim Index



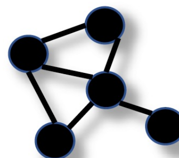
Bloom-Filter



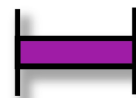
Sorting



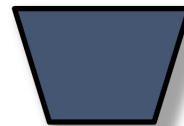
Scheduling



Range-Filter



Hash-Map

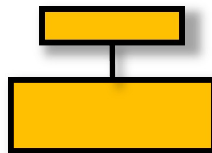


Data Cubes

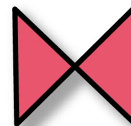
DNA-Search

SQL Query Optimizer

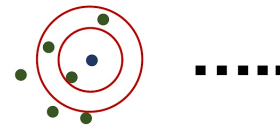
Cache Policy



Join



Nearest Neighbor



......

Slides adapted from SIGMOD19 Tutorial Learned Data Structures and Algorithms (part 2) by Tim Kraska

# Readings

- Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis. The Case for Learned Index Structures [SIGMOD'18]

- Jialin Ding, Umar Farooq Minhas, Hantian Zhang, Yinan Li, Chi Wang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet. ALEX: An Updatable Adaptive Learned Index [SIGMOD'20]