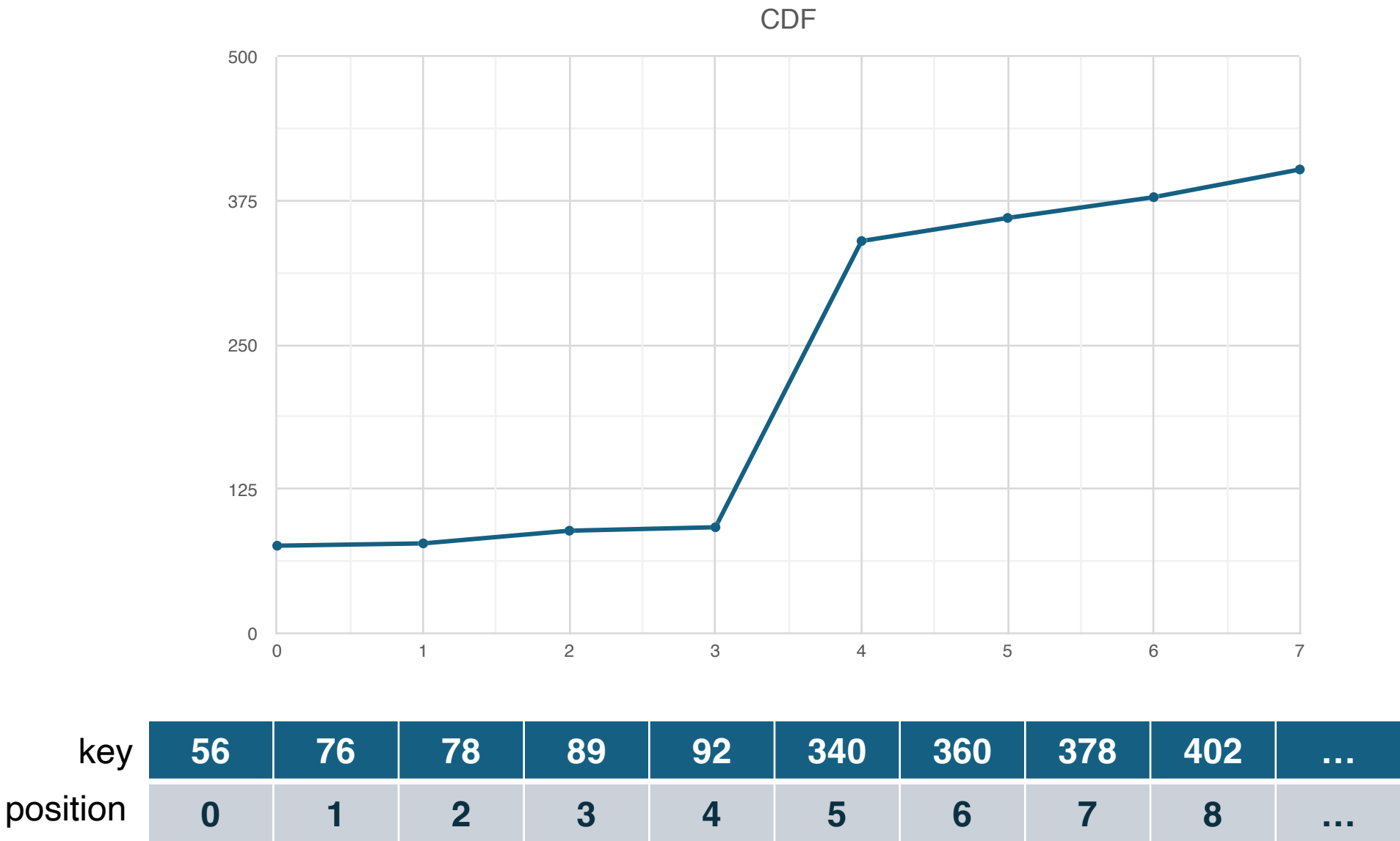


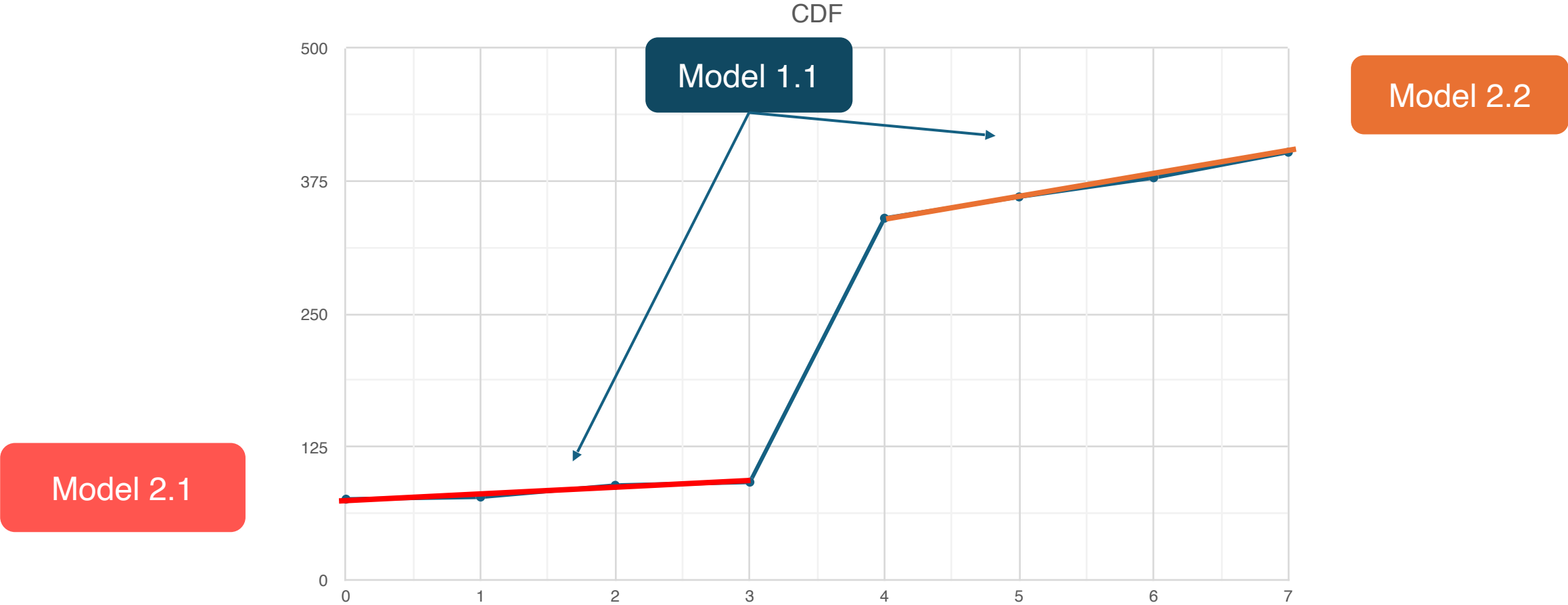
Alex: An Updatable Adaptive Learned Index

Presented by Owen Gombas
Data Management Data Structure
Swiss Joint Master in Computer Science
16.05.2024

Recursive Model Index (RMI)

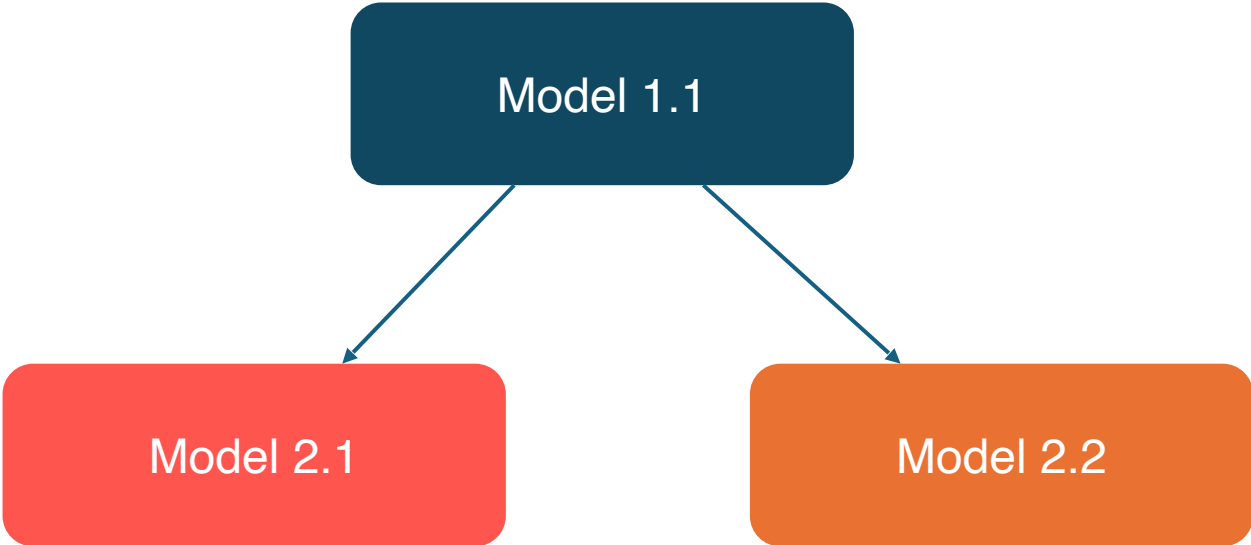


Recursive Model Index (RMI)



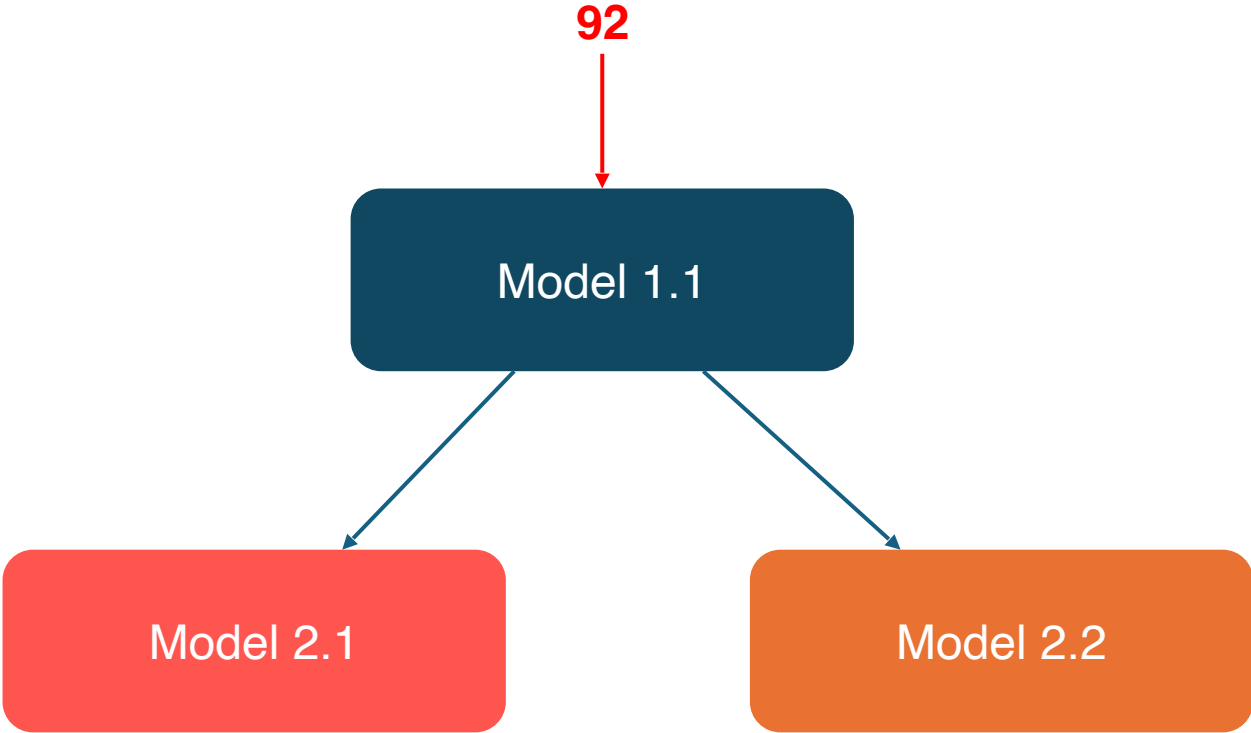
key	56	76	78	89	92	340	360	378	402	...
position	0	1	2	3	4	5	6	7	8	...

Recursive Model Index (RMI)



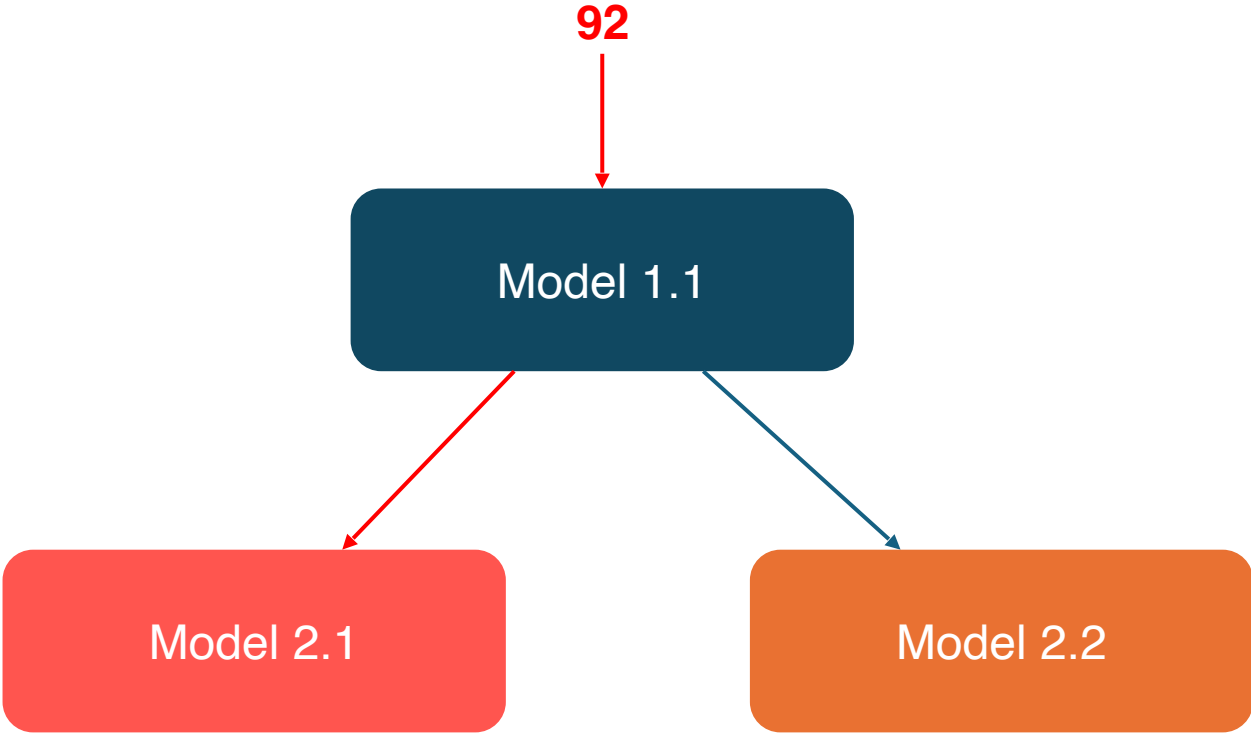
key	56	76	78	89	92	340	360	378	402	...
position	0	1	2	3	4	5	6	7	8	...

Recursive Model Index (RMI)



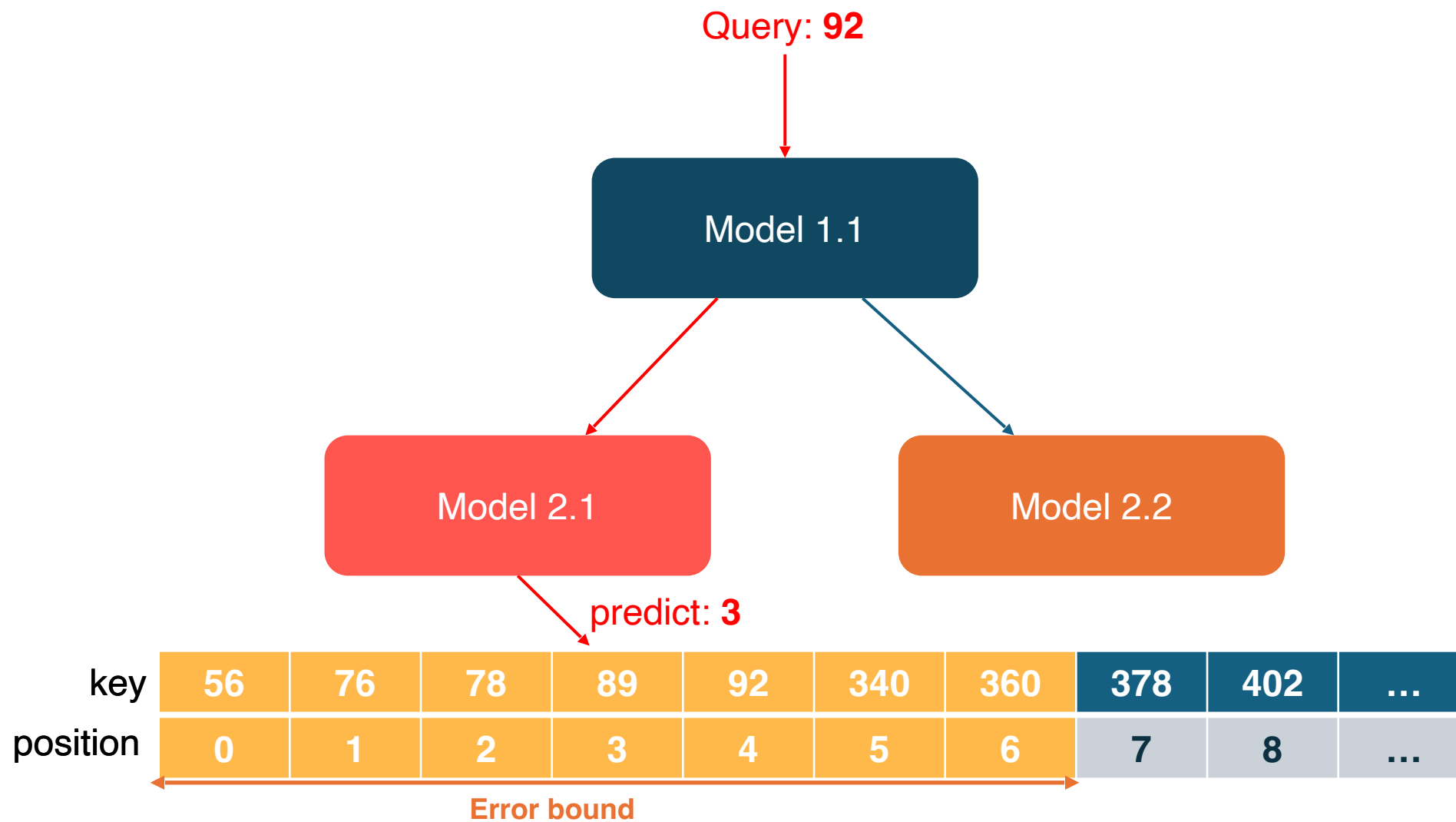
key	56	76	78	89	92	340	360	378	402	...
position	0	1	2	3	4	5	6	7	8	...

Recursive Model Index (RMI)

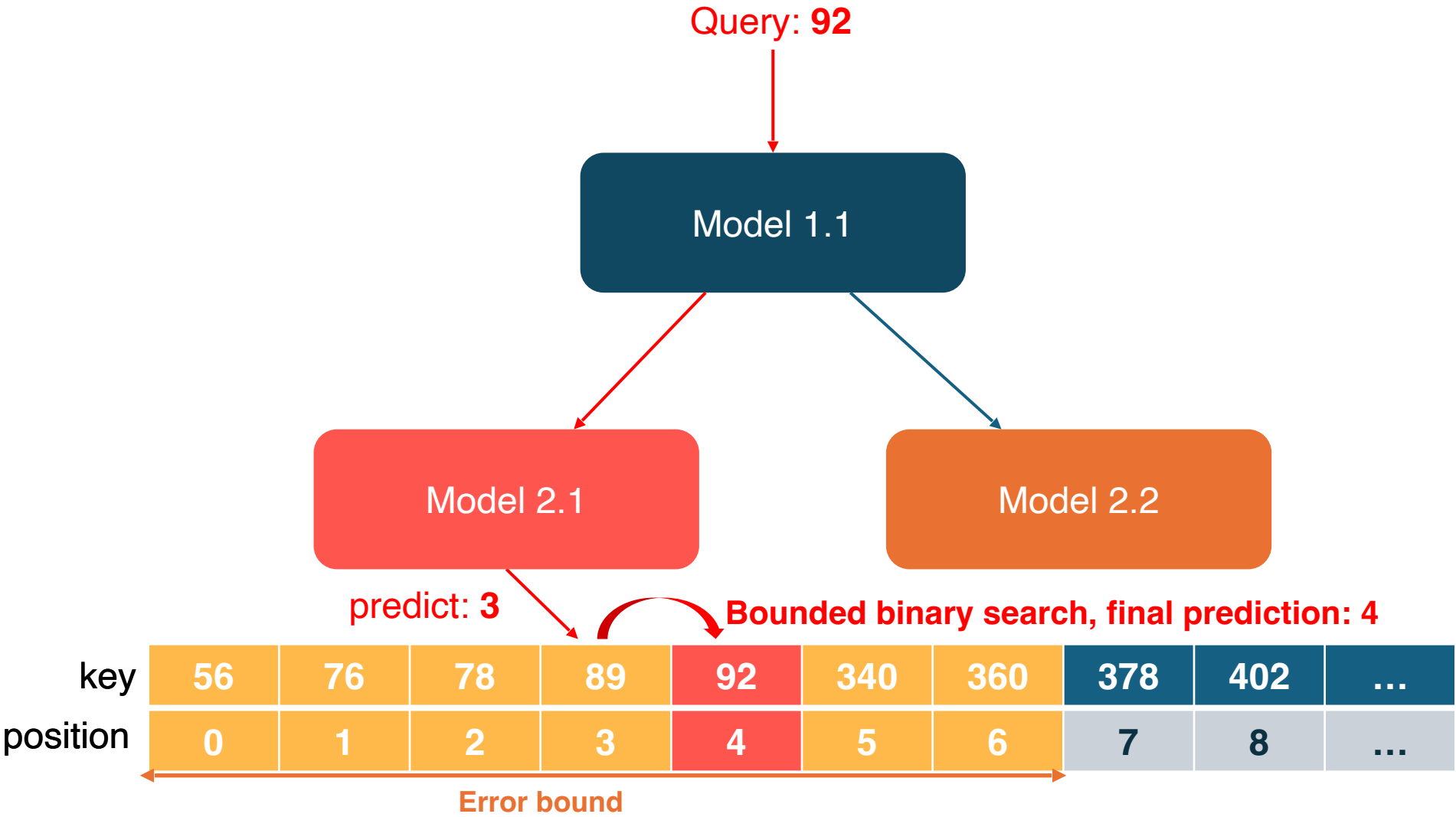


key	56	76	78	89	92	340	360	378	402	...
position	0	1	2	3	4	5	6	7	8	...

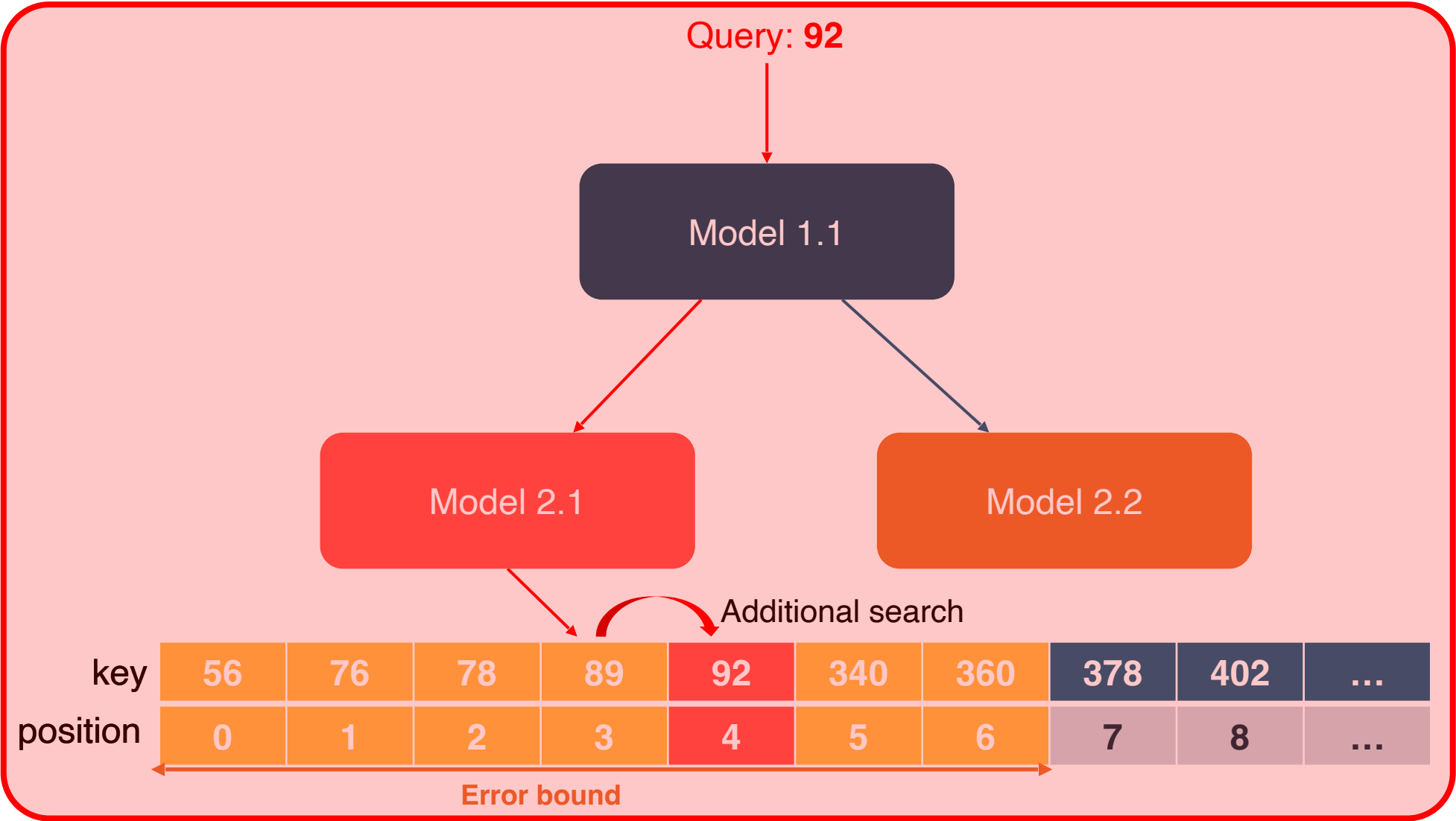
Recursive Model Index (RMI)



Recursive Model Index (RMI)



Recursive Model Index (RMI) Immutable



Recursive Model Index (RMI)

Immutable

Alex: An Updatable Adaptative Learned Index

Mutable

Recursive Model Index (RMI)

Immutable

Very slow build time

Alex: An Updatable Adaptative Learned Index

Mutable

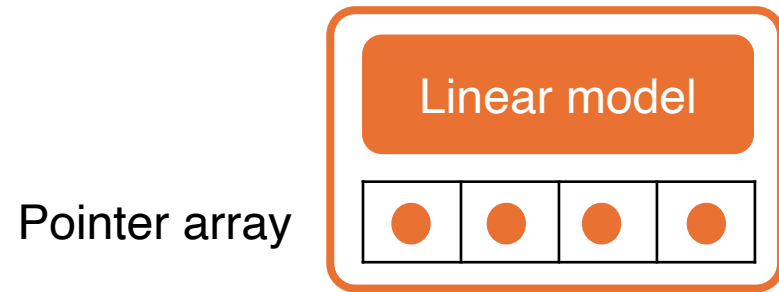
Insert time competitive to B+Tree

Lookup faster than B+Tree and Learned Index

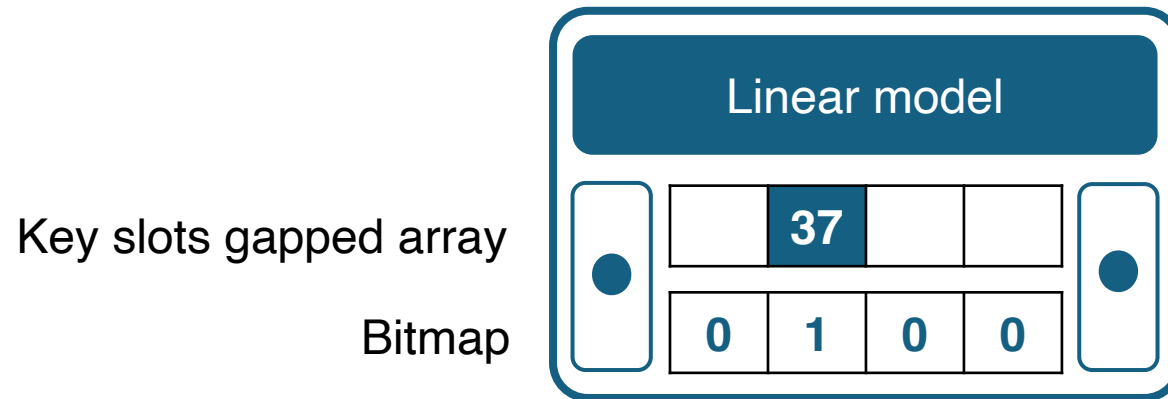
Index storage space smaller than B+Tree and Learned Index

Node layout

Model node (interior node)

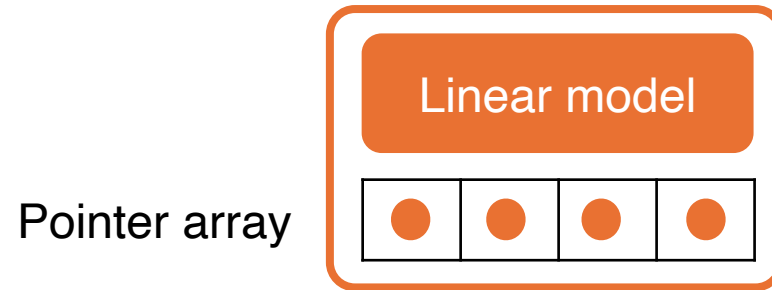


Data node (leaf node)

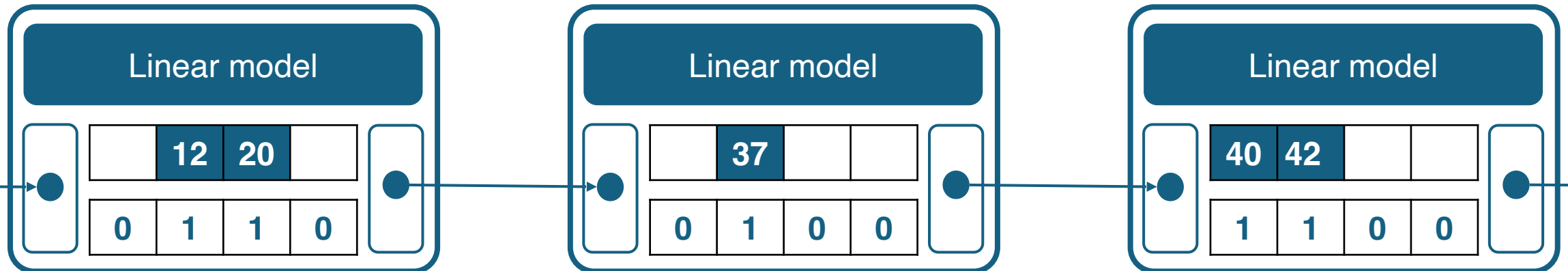


Node layout

Model node (interior node)



Data node (leaf node)



Index structure

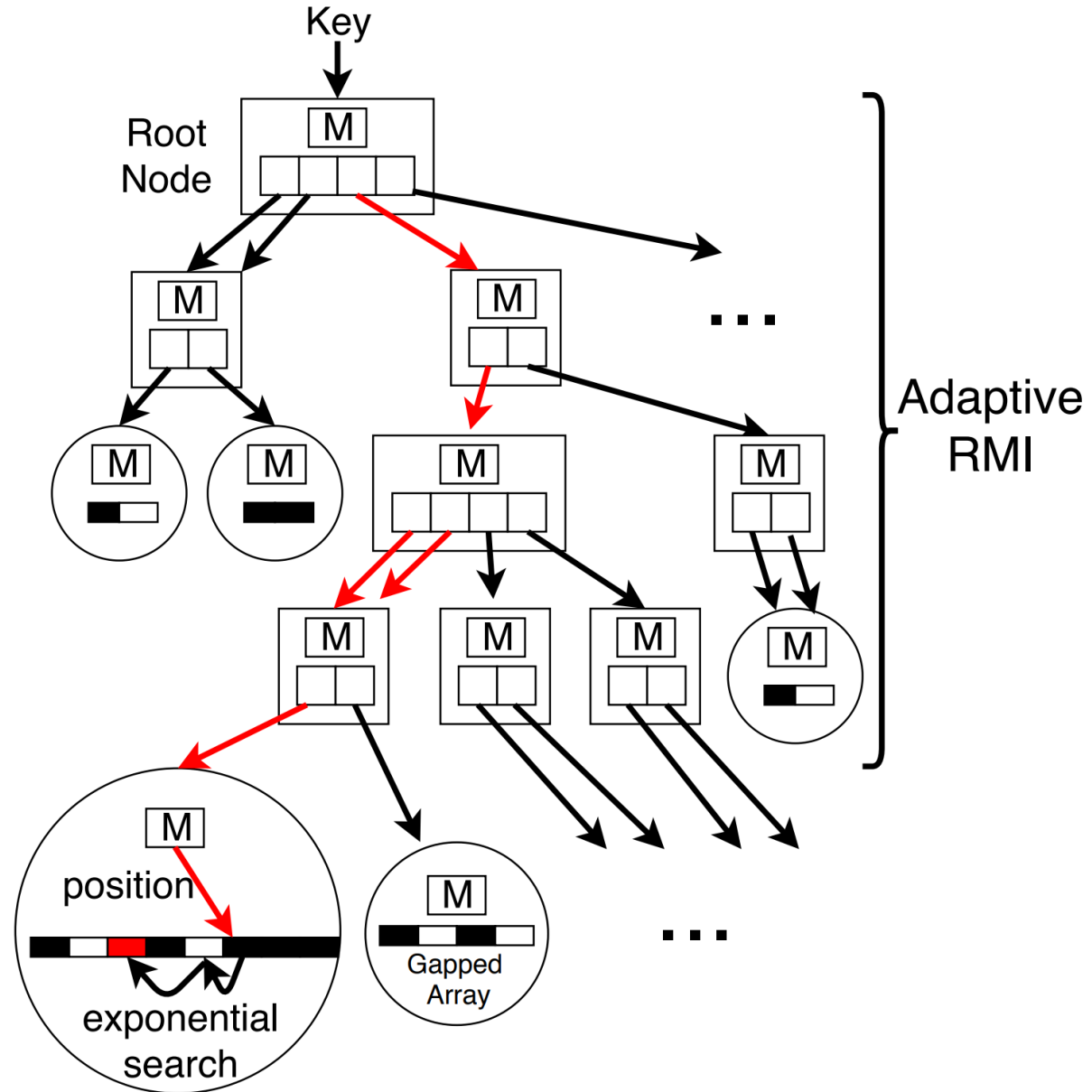
Legend

Internal
Node

Data
Node

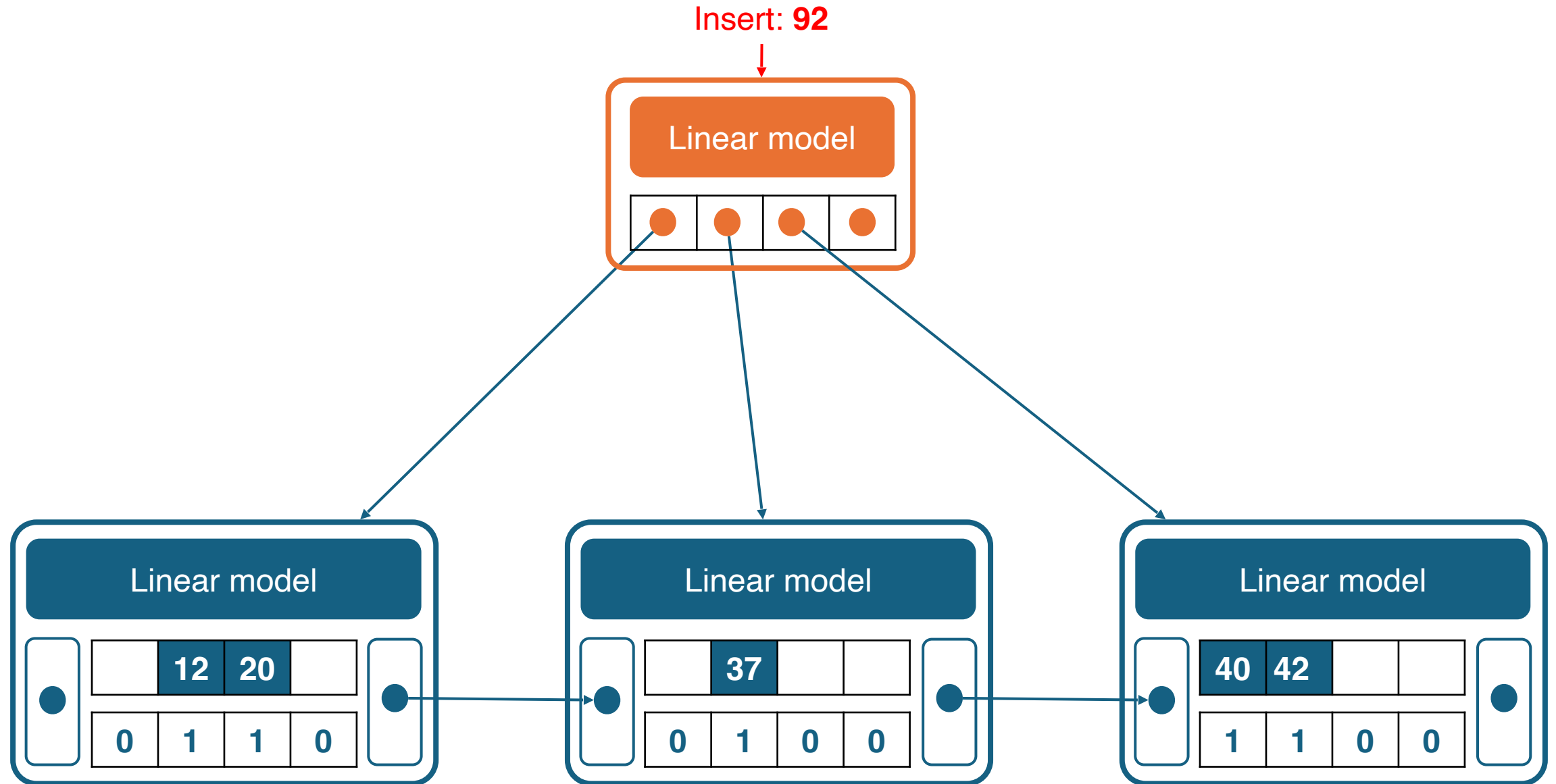
■ Key

□ Gap

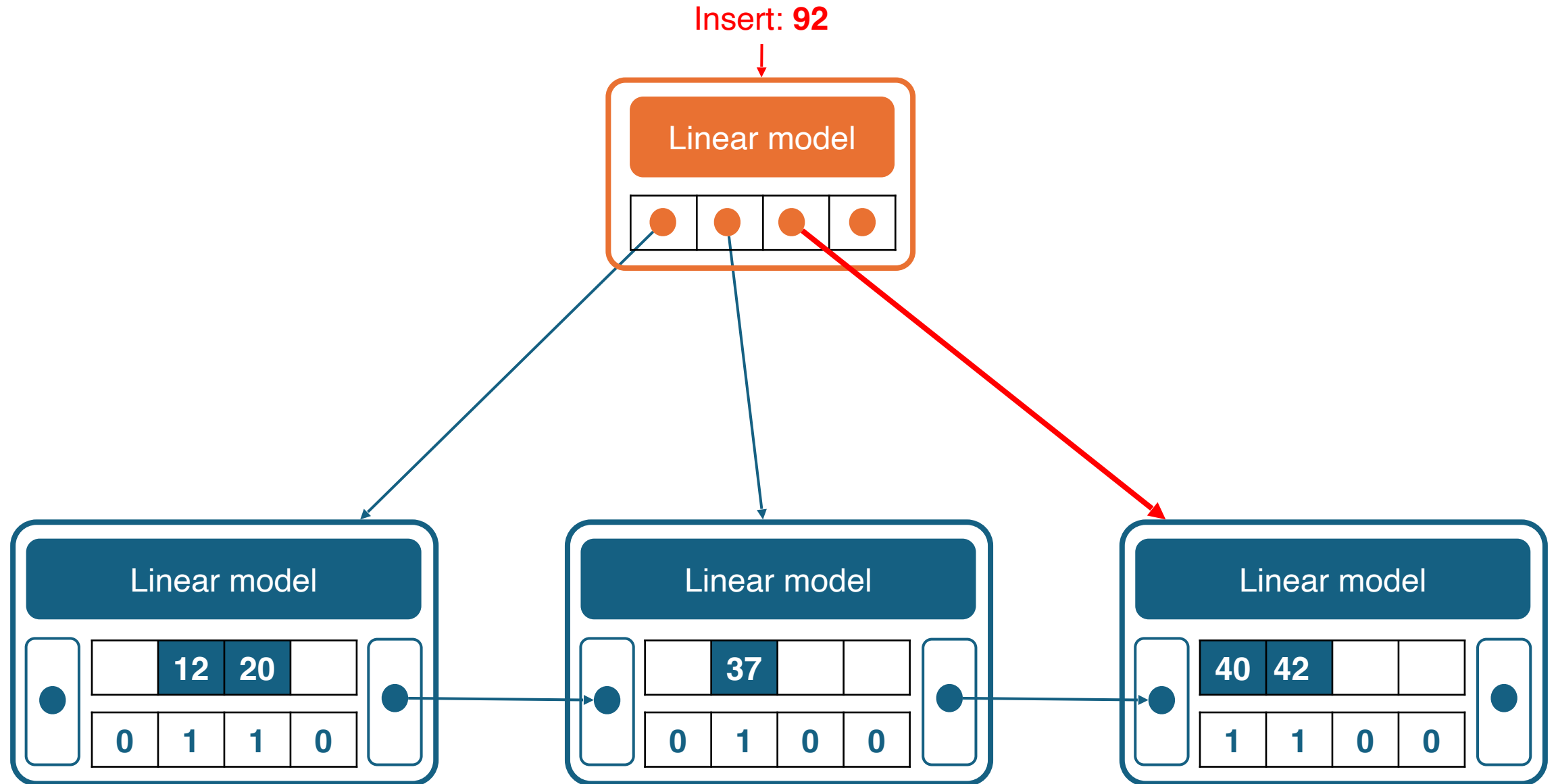


How to insert ?

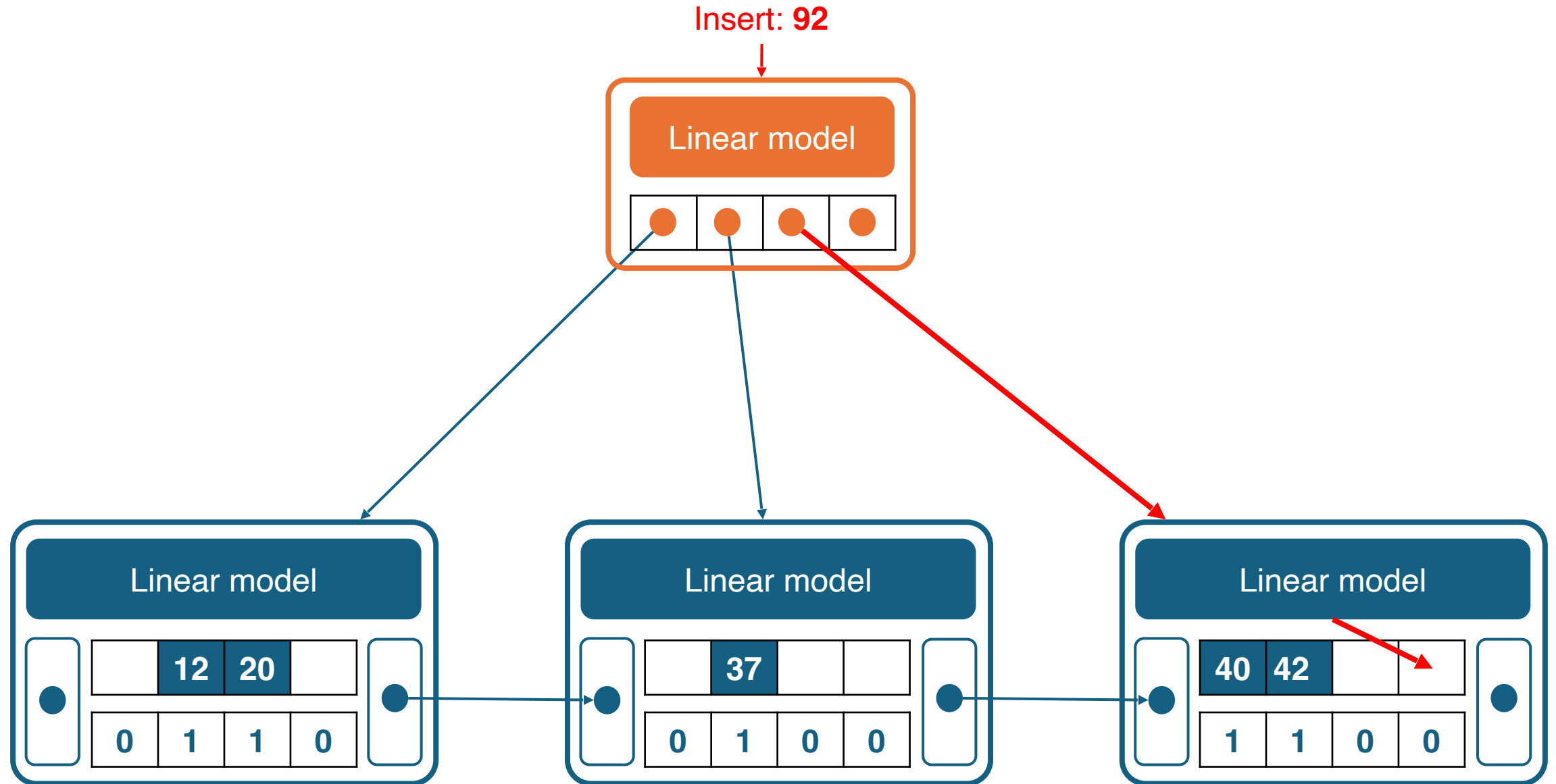
Insert - simple case



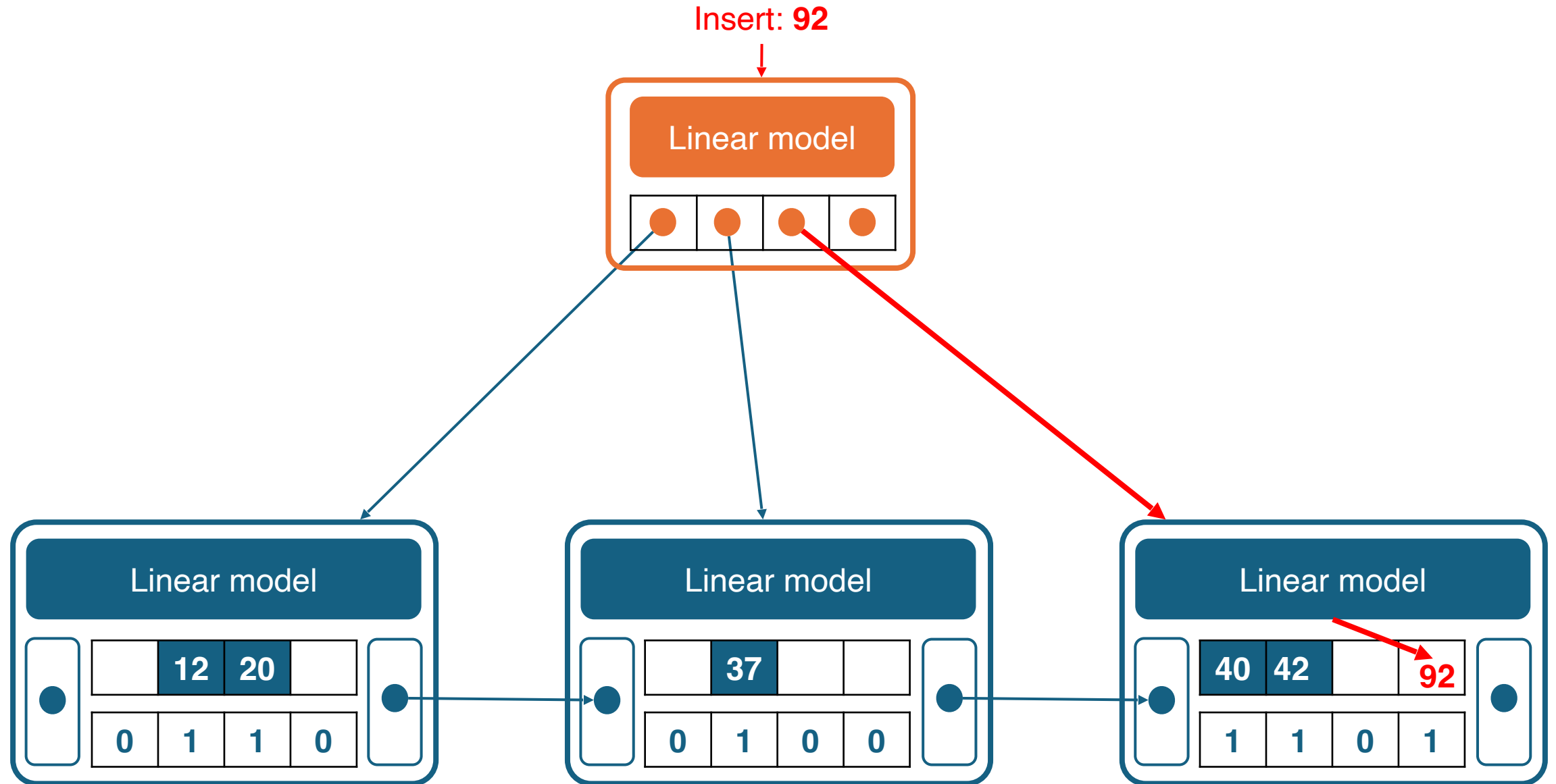
Insert - simple case



Insert - simple case



Insert - simple case




Insert - collision with preserved sorted order

10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

Insert - collision with preserved sorted order

Insert: **23**

$$f(23) = 2$$



10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

Insert - collision with preserved sorted order

Insert: **23**

$f(23) = 2$

Position isn't a gap → Shift to the closest gap direction

10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

10	15		24	46	49	51	78		
0	1	2	3	4	5	6	7	8	9

Insert - collision with preserved sorted order

Insert: **23**

$$f(23) = 2$$

Position isn't a gap → Shift to the closest gap direction

10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

10	15		24	46	49	51	78		
0	1	2	3	4	5	6	7	8	9

10	15	23	24	46	49	51	78		
0	1	2	3	4	5	6	7	8	9

Insert - collision without preserved sorted order

Insert: **32**

$$f(32) = 2$$




10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

Insert - collision without preserved sorted order

Insert: **32**

$f(32) = 2$

1. Predicted position do not preserve sorted order → Exponential search



10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

Insert - collision without preserved sorted order

Insert: 32

$f(32) = 2$

1. Predicted position do not preserve sorted order → Exponential search

Exponential search: 2 → 3

10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

Insert - collision without preserved sorted order

Insert: 32

$$f(32) = 2$$

1. Predicted position do not preserve sorted order → Exponential search
2. Position isn't a gap → Shift to the closest gap direction

Exponential search: 2 → 3

10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

10	15	24		46	49	51	78		
0	1	2	3	4	5	6	7	8	9

Insert - collision without preserved sorted order

Insert: **32**

$$f(32) = 2$$

1. Predicted position do not preserve sorted order → Exponential search
2. Position isn't a gap → Shift to the closest gap direction

Exponential search: 2 → 3

10	15	24	46	49	51		78		
0	1	2	3	4	5	6	7	8	9

10	15	24		46	49	51	78		
0	1	2	3	4	5	6	7	8	9

10	15	24	32	46	49	51	78		
0	1	2	3	4	5	6	7	8	9



Maximal node capacity reached

Maximal node capacity reached

Data Node reach fullness criteria (80% full)

Linear model									
10	15	24	32	46	49	51			78
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	0	0	1

What do we do ?

Maximal node capacity reached

Full Data Node

Linear model									
10	15	24	32	46	49	51			78
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	0	0	1

What do we do ?

Expansion

Splits

Split sideways

Split downwards

Expansion mechanism

[illegible]

Expansion mechanism

[illegible]

Expansion mechanism

Linear model									
10	15	24	32	46	49	51			78
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	0	0	1

Expands key slots capacity

Linear model													
↓	↓	inner gaps		↓	inner gaps		↓	↓	↓	↓	remaining gaps		
10	12		24	32		46	49	51	51	78			
0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	0	1	1	0	1	1	1	1	1	0	0	0

Expansion mechanism

Linear model									
10	15	24	32	46	49	51			78
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	0	0	1

Expands key slots capacity

Linear model													
↓	↓	inner gaps		↓	inner gaps		↓	↓	↓	↓	remaining gaps		
10	12	12	24	32	32	46	49	51	51	78	INF	INF	INF
0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	0	1	1	0	1	1	1	1	1	0	0	0

Split sideways mechanism



Linear model									
10	15	24	32	46	49	51			78
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	0	0	1

Becomes

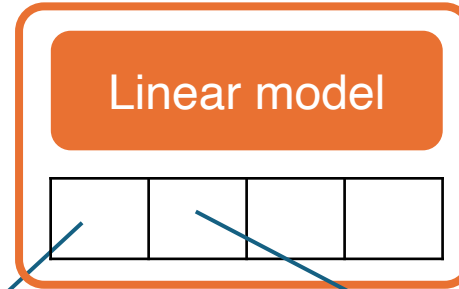
Linear model

10	15	24	32	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

Linear model

46	49	51	78	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

Split sideways mechanism



Reconnect to initial node ancestor

Linear model

10	15	24	32	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

Linear model

46	49	51	78	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

Split downwards mechanism

Linear model									
10	15	24	32	46	49	51			78
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	0	0	1

Becomes

Linear model							
10	15	24	32	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

Linear model							
46	49	51	78	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

Split downwards mechanism

Reconnect to initial node ancestor

Linear model

New internal node created with a fanout of 2

Linear model

10	15	24	32	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

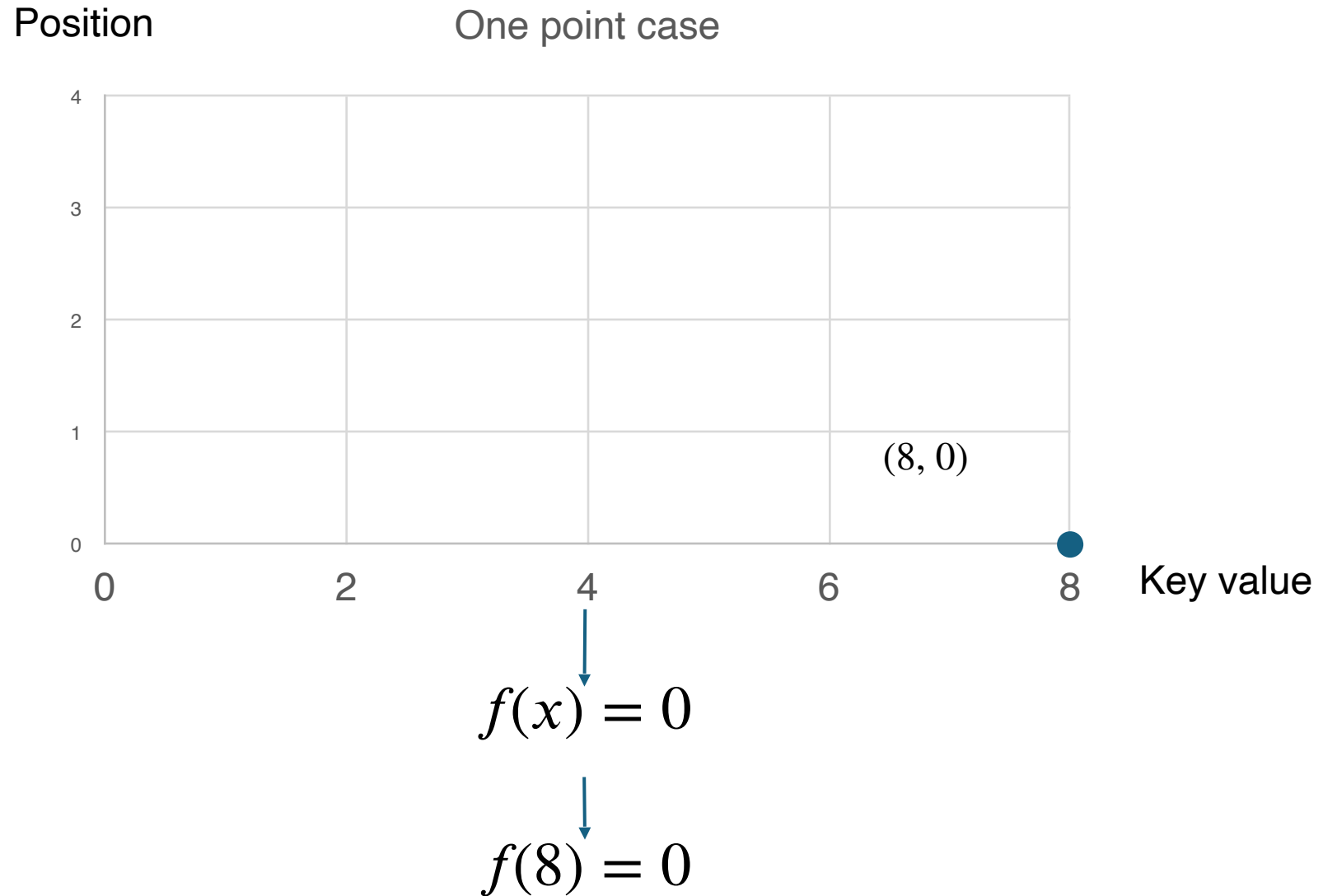
Linear model

46	49	51	78	INF	INF	INF	INF
0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

How to fit the linear models

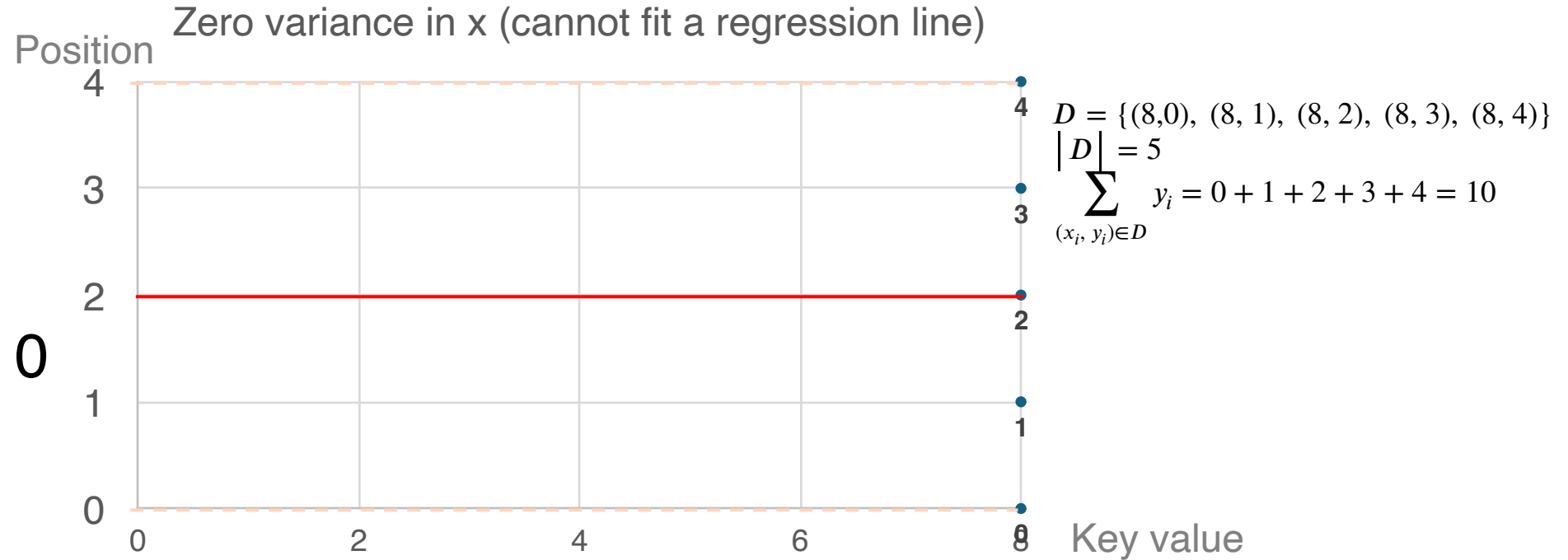
Fitting the Data Node model - corner case 1

If $N \leq 1$



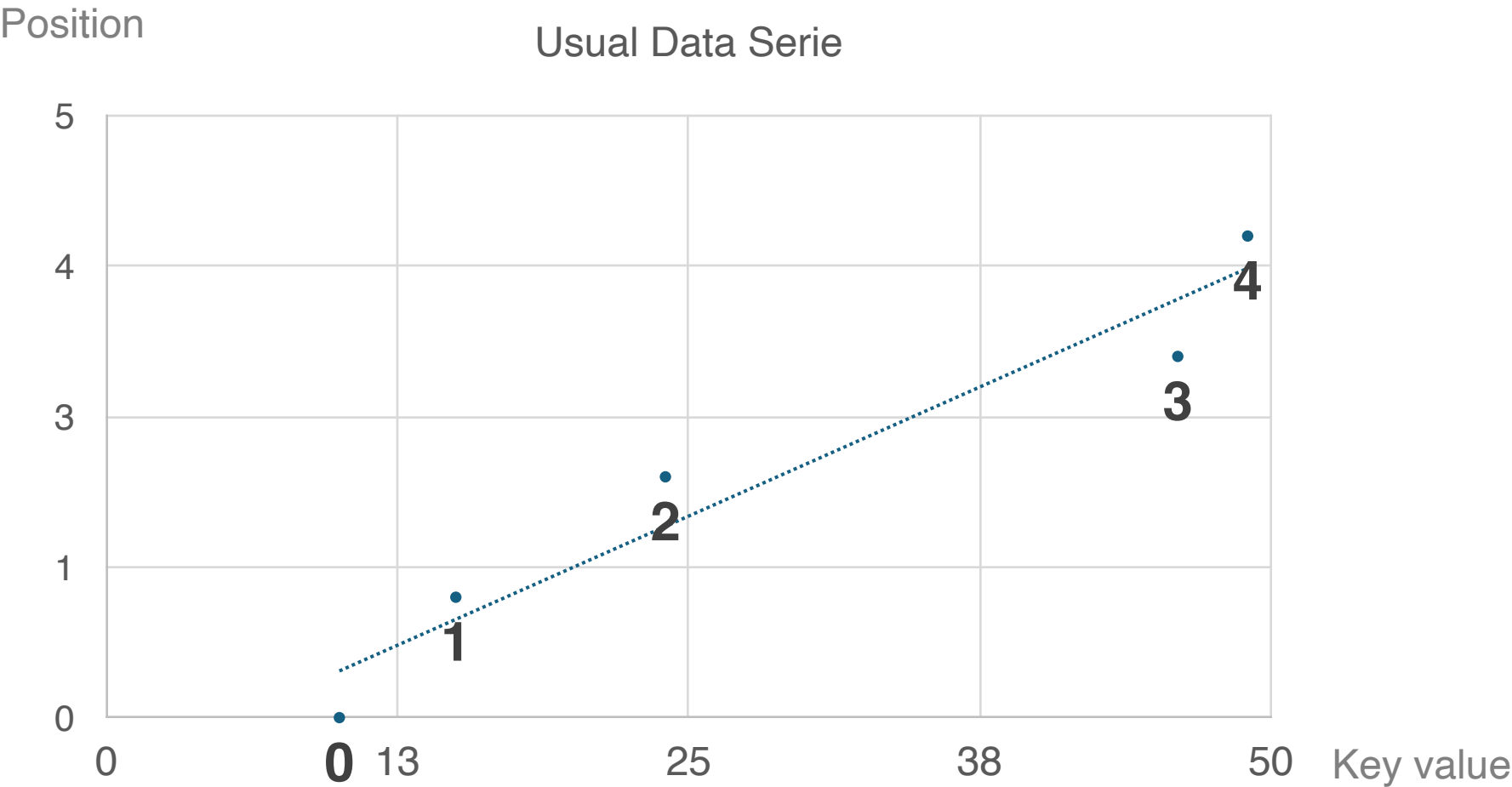
Fitting the Data Node model - corner case 2

If $N > 1$ and
variance in x is 0



$$f(x) = \frac{1}{|D|} \sum_{(x_i, y_i) \in D} y_i = 2 \quad (\text{mean of the } y \text{ values})$$
$$f(8) = 2$$

Fitting the Data Node model - usual case

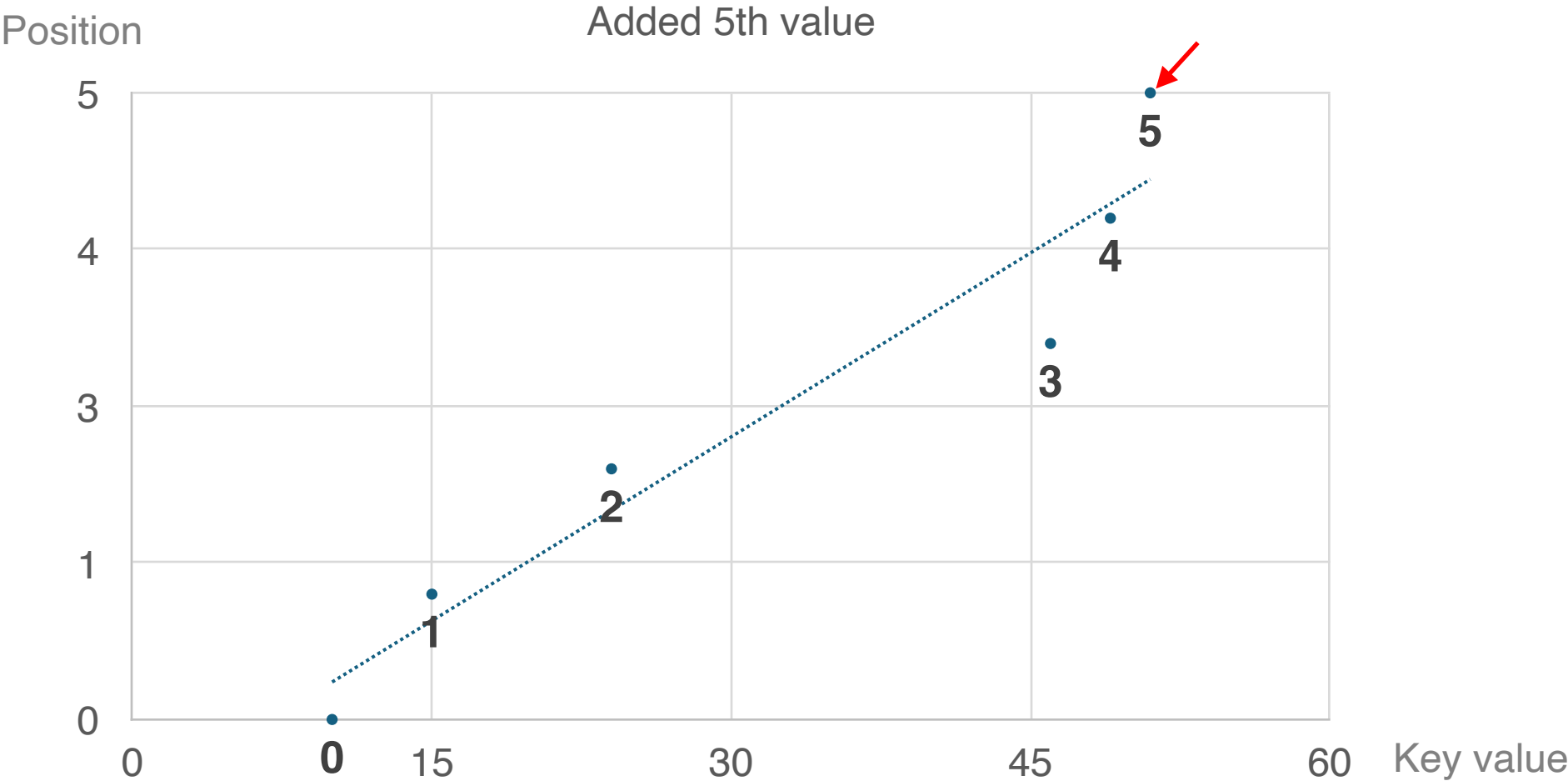


x

10	15	24	46	49
0	1	2	3	4

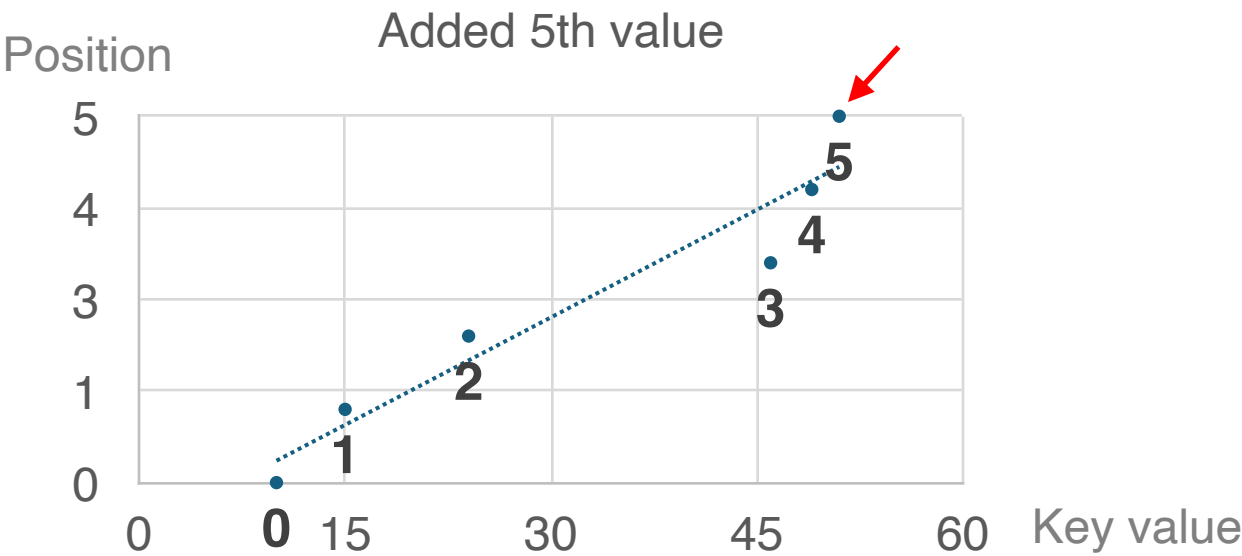
$\text{Int}[f(x)]$

Fitting the Data Node model - model error



x	10	15	24	46	49	51
$\text{Int}[f(x)]$	0	1	2	3	4	4

Fitting the Data Node model



x	10	15	24	46	49	51
$\text{Int}[f(x)]$	0	1	2	3	4	4

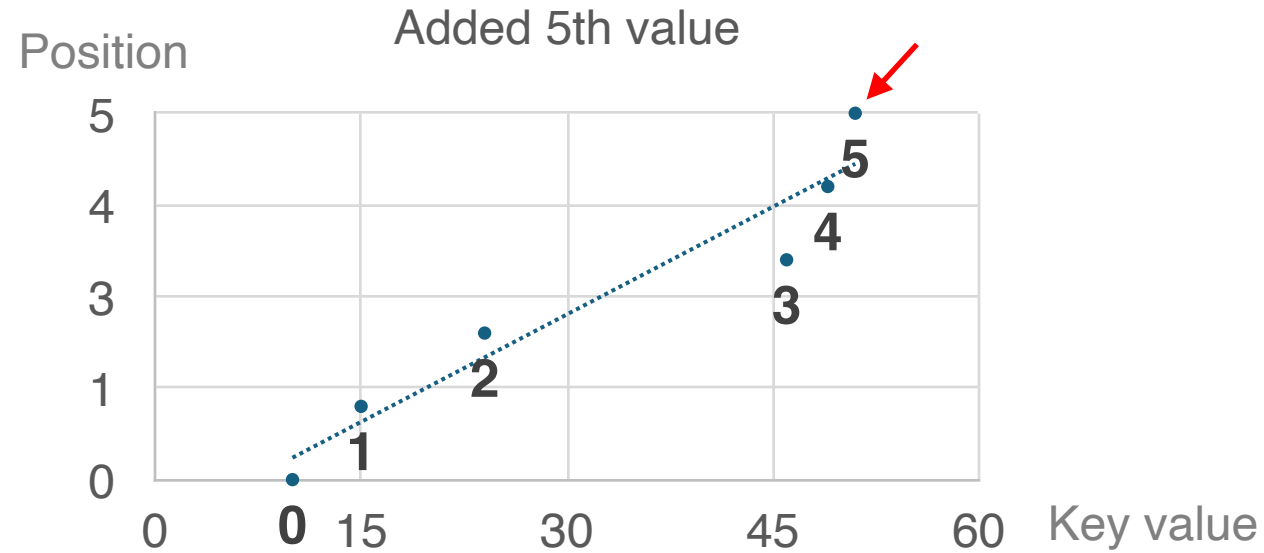
$\text{Int}[f(51)]$ ↓

Key slots

Position

10	15	24	46	49					
0	1	2	3	4	5	6	7	8	9

Fitting the Data Node model



x	10	15	24	46	49	51
$\text{Int}[f(x)]$	0	1	2	3	4	4

$\text{Int}[f(51)]$ ↓ Exponential search: 4 → 5

Key slots

Position

10	15	24	46	49	51				
0	1	2	3	4	5	6	7	8	9

In addition

- Alex has a mechanism to handle corner cases where the keys are inserted in an increasing/decreasing order to avoid poor performances
- Model nodes are built in such way that they do not produce any errors
- Alex uses exponential search because:
 - Experimentally verified that exponential is faster than bounded binary search
 - Exponential search remove the need for storing the error bounds
(because it is as fast without error bounds as bounded binary search)

How to choose between expansion and splits ?

Compute expected cost of a primitive

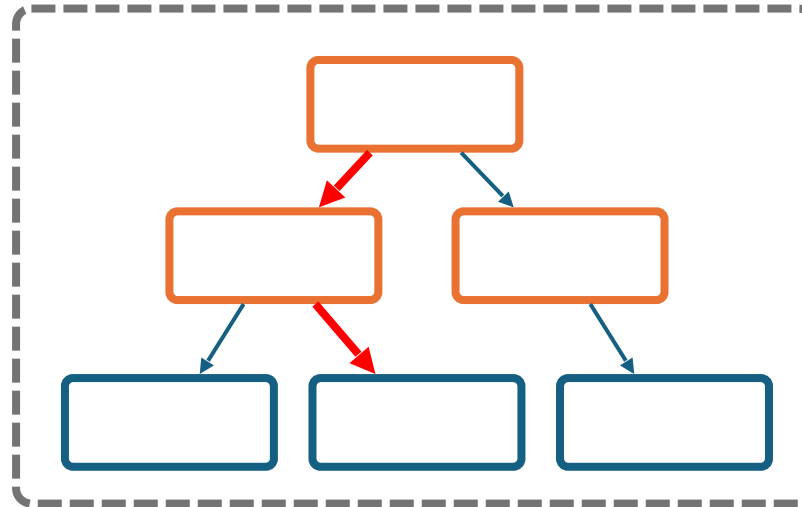
Try to predict the future, computed at node creation time

Intra-node



1. Average lookup time
2. Average insert time

Traverse to Leaf



Empirical cost

Uses the actual collected statistics at time t and uses it at time t

Intra-node cost model

How:

1. Average lookup time: average log of model prediction error
(average number of exponential search steps)
2. Average Insert time: average distance to the closest gap for all existing keys
(average number of shifts for inserts)

Intra-node cost model

How:

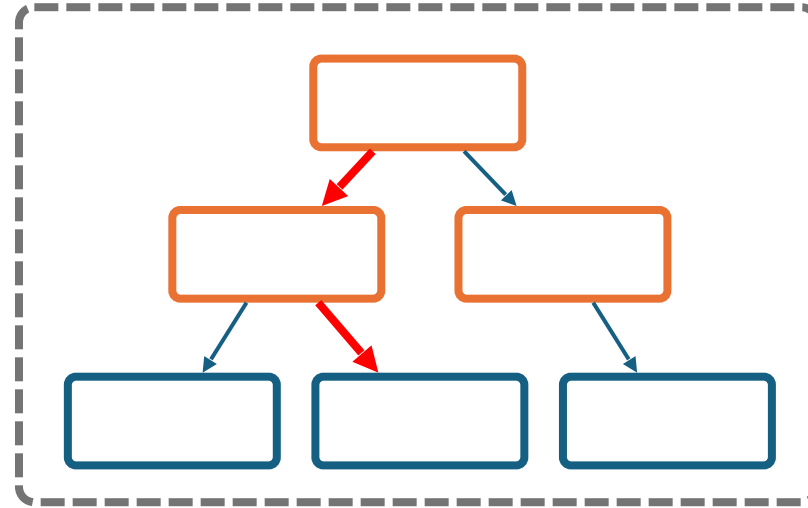
1. Average lookup time: average log of model prediction error
(Average number of exponential search steps)
2. Average Insert time: average distance to the closest gap for all existing keys
(Average number of shifts for inserts)

Because:

- Lookup performance is directly correlated with 1.
- Insert performance is directly correlated with both 1. and 2.
(since inserts also perform an exponential search to find the correct insert position)

TraverseToLeaf cost model

Traverse to Leaf



TraverseToLeaf cost model

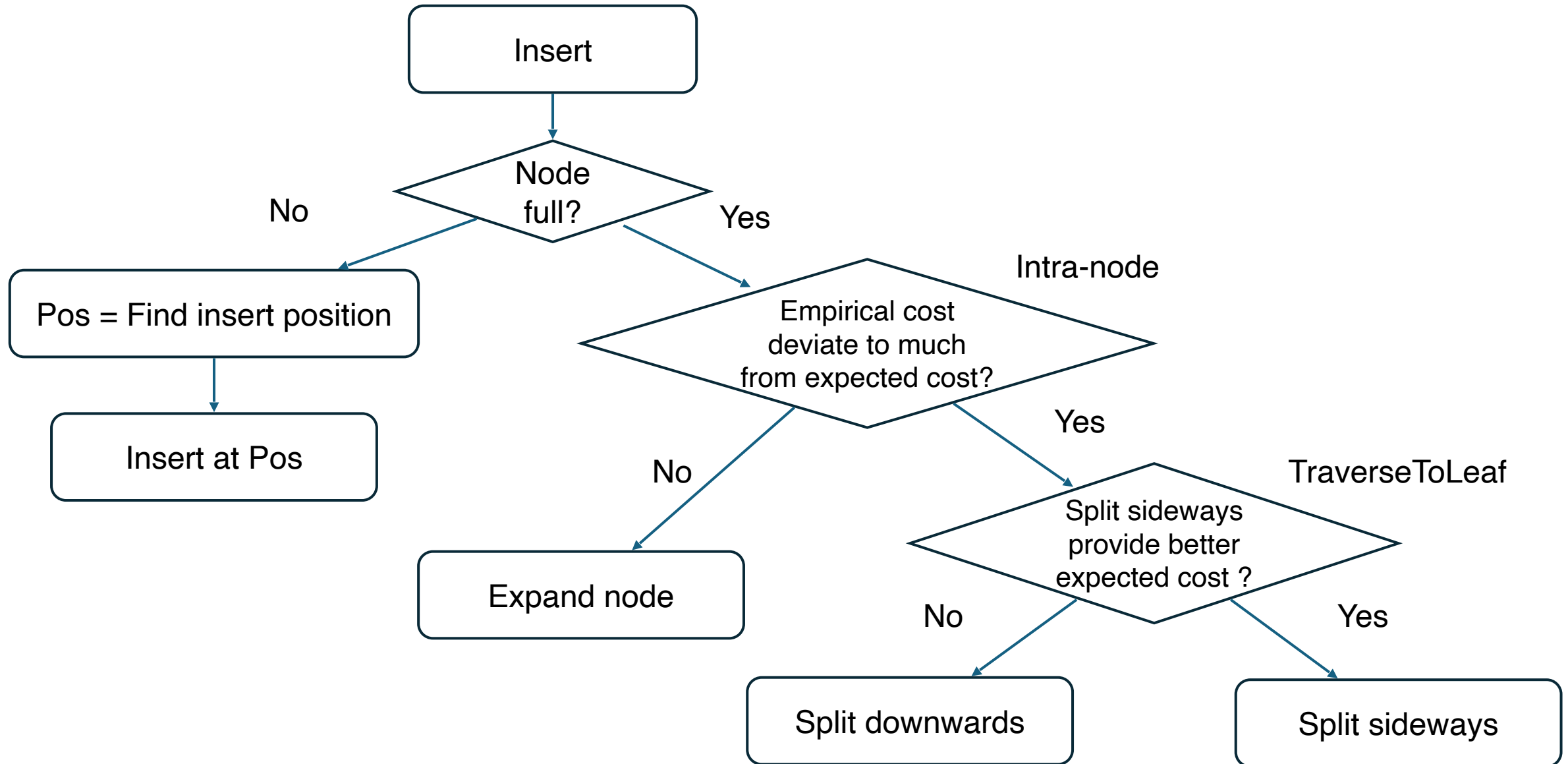
Use the following values:

1. The depth of the Data Node being traversed to
2. The total size (in bytes) of all inner nodes and data node metadata

Because:

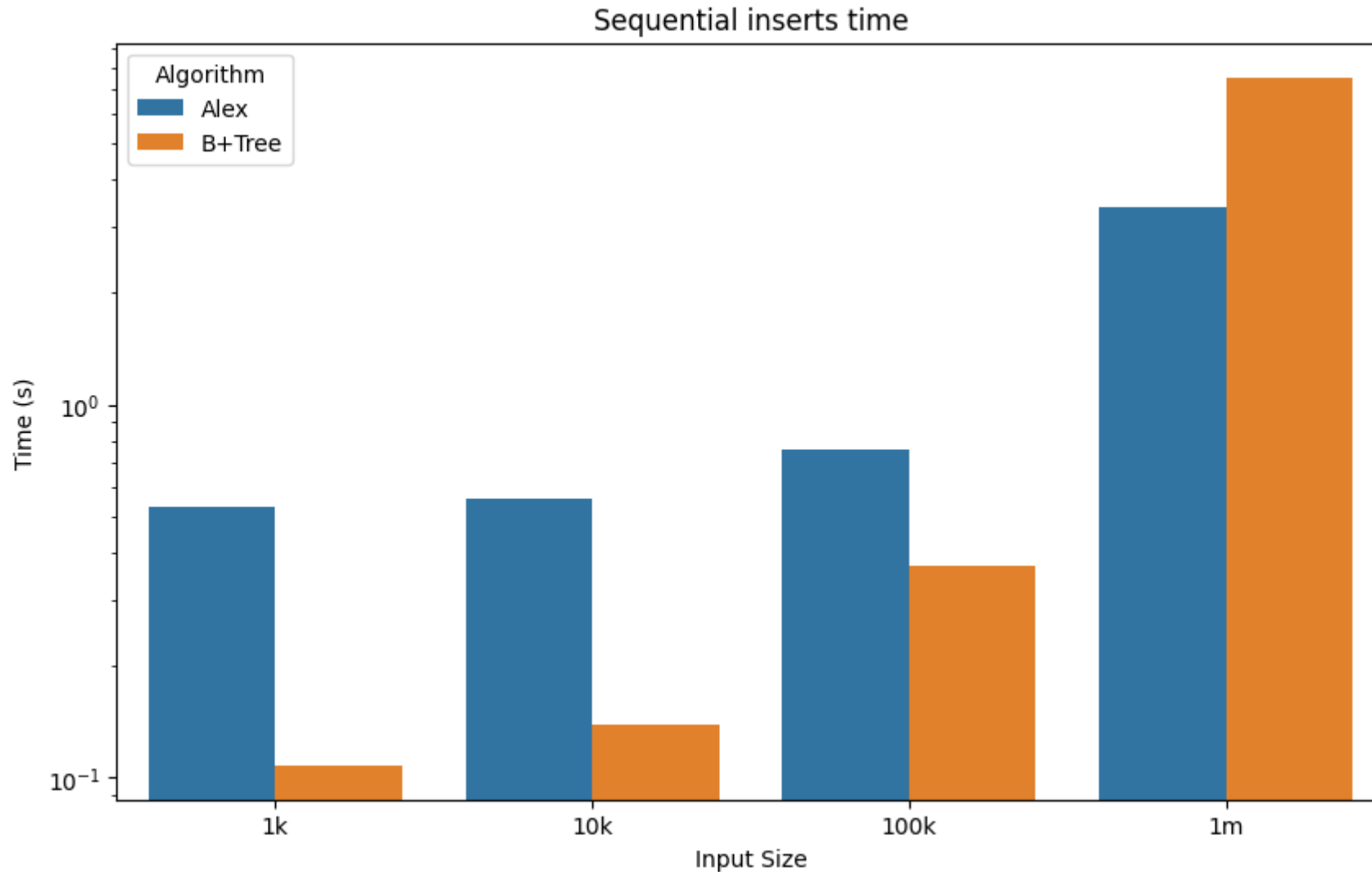
- Deeper Data Node is, the more pointer chasing are required
- Larger size reduces CPU cache locality inducing bad performances

How to choose between expansion and splits ?



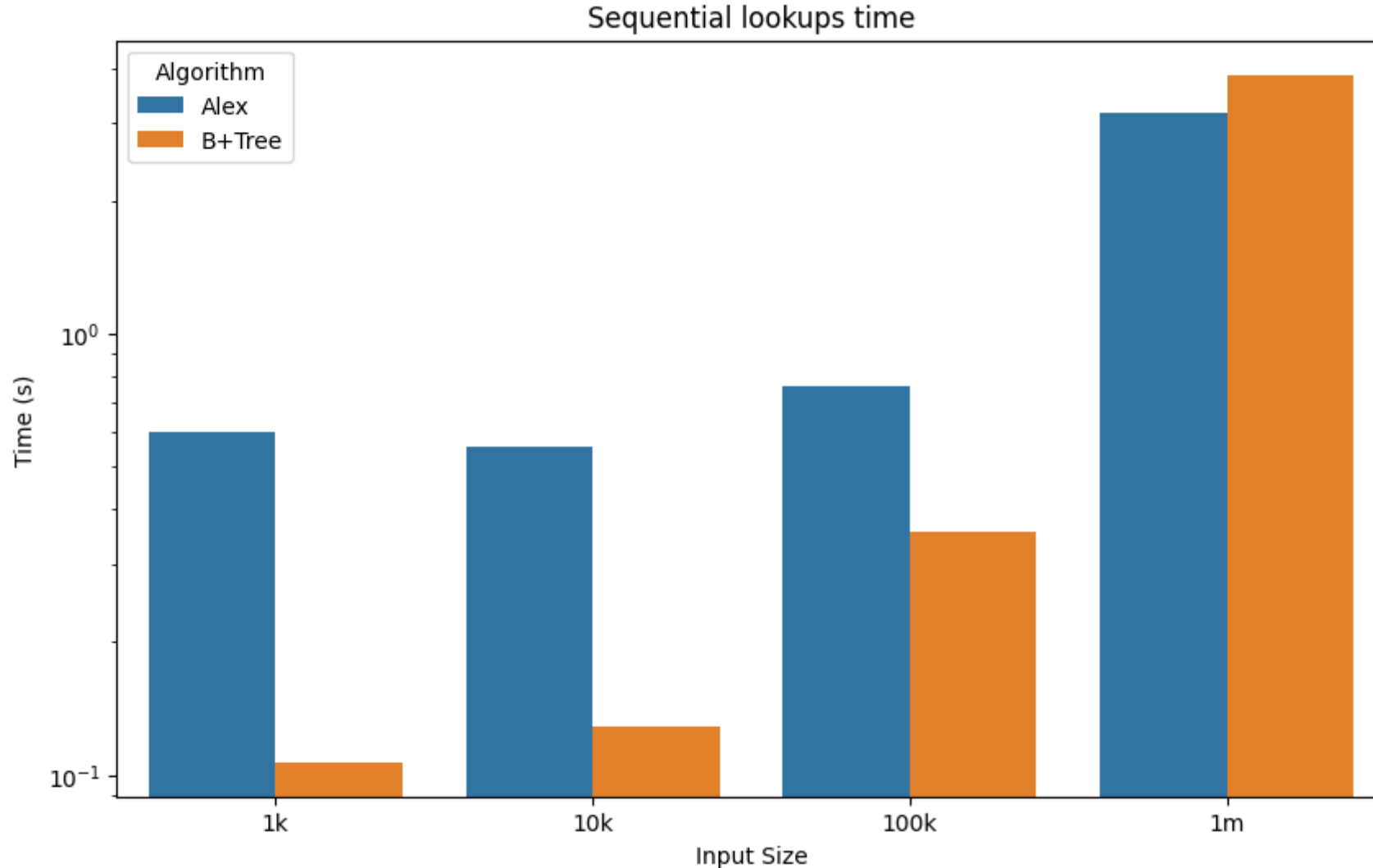
Benchmarks sequential inserts (vs B+Tree)

Sequential insert of 1k to 1m keys

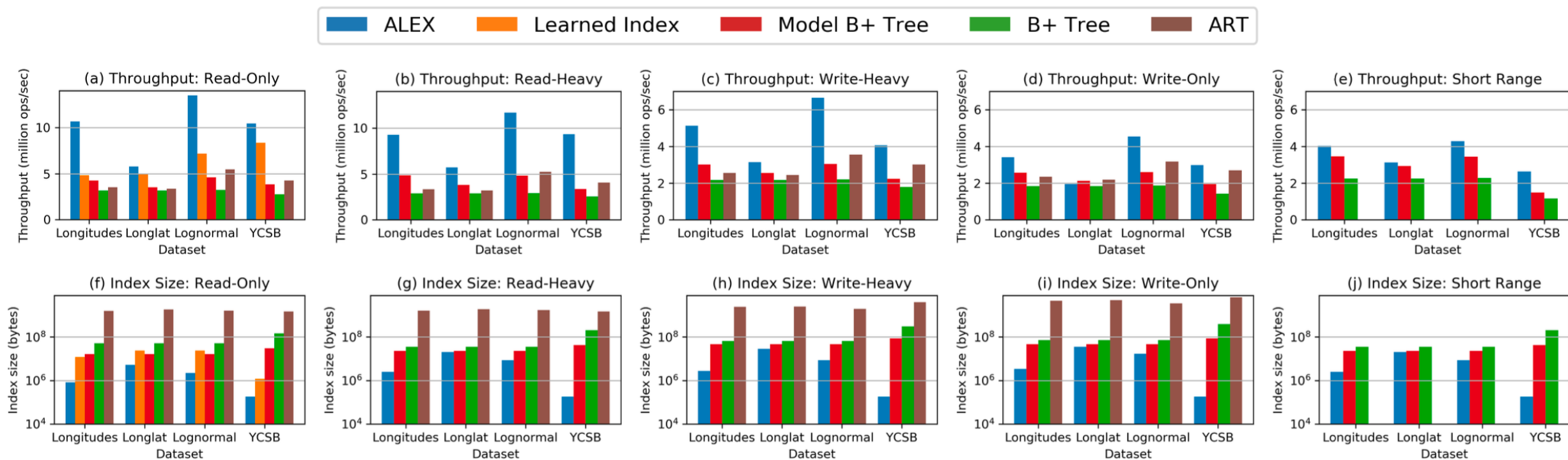


Benchmarks sequential lookups (vs B+Tree)

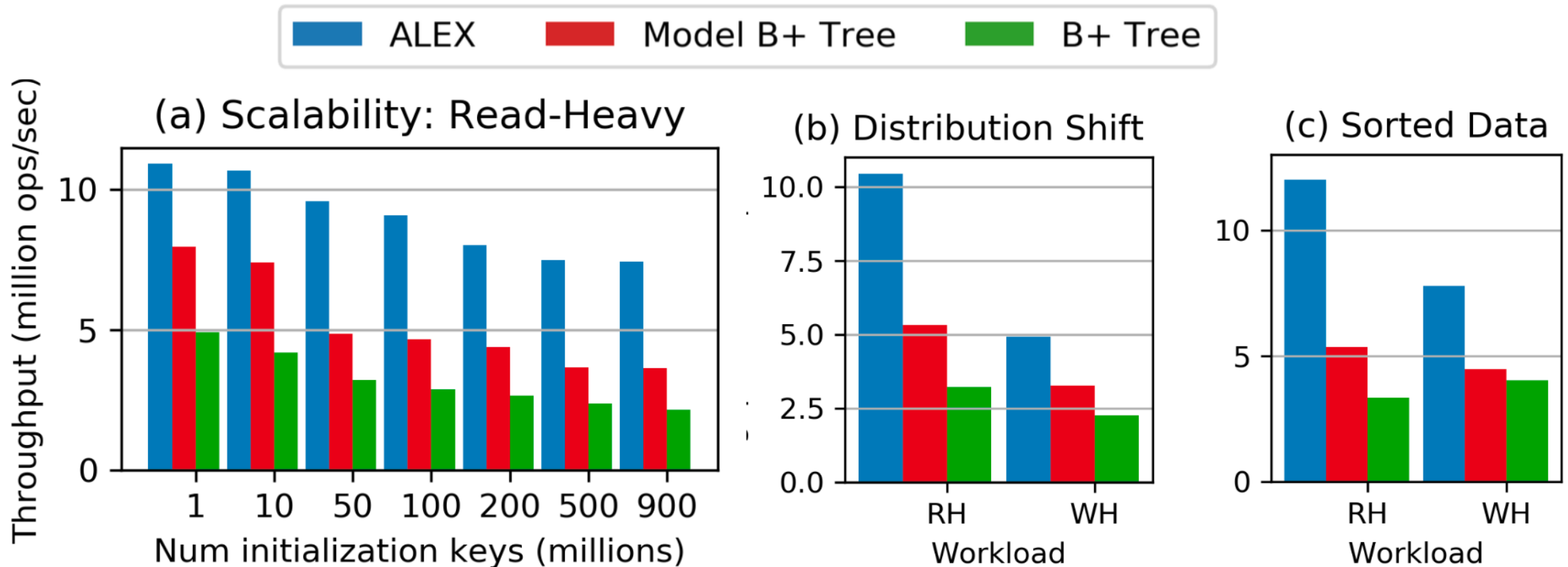
Sequential lookups of 1k to 1m keys



Benchmarks (from the paper)

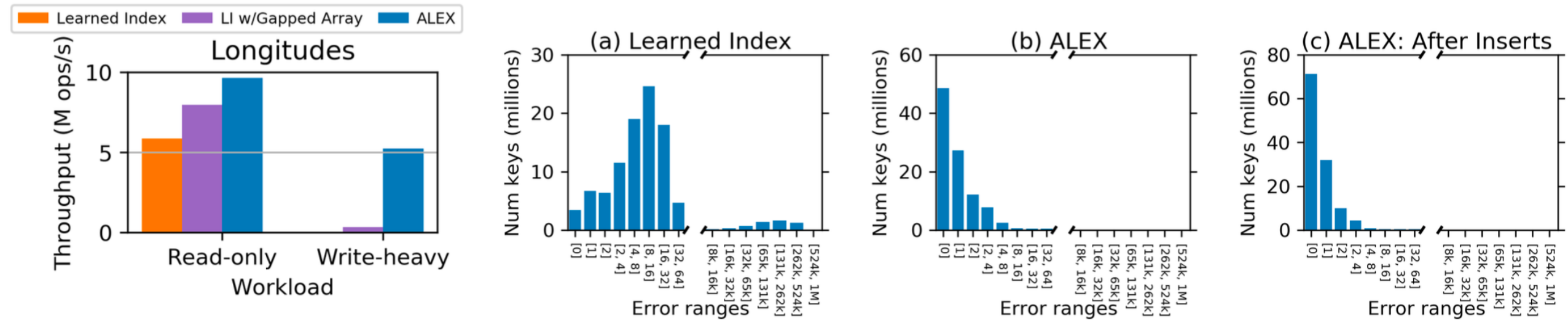


Benchmarks (from the paper)



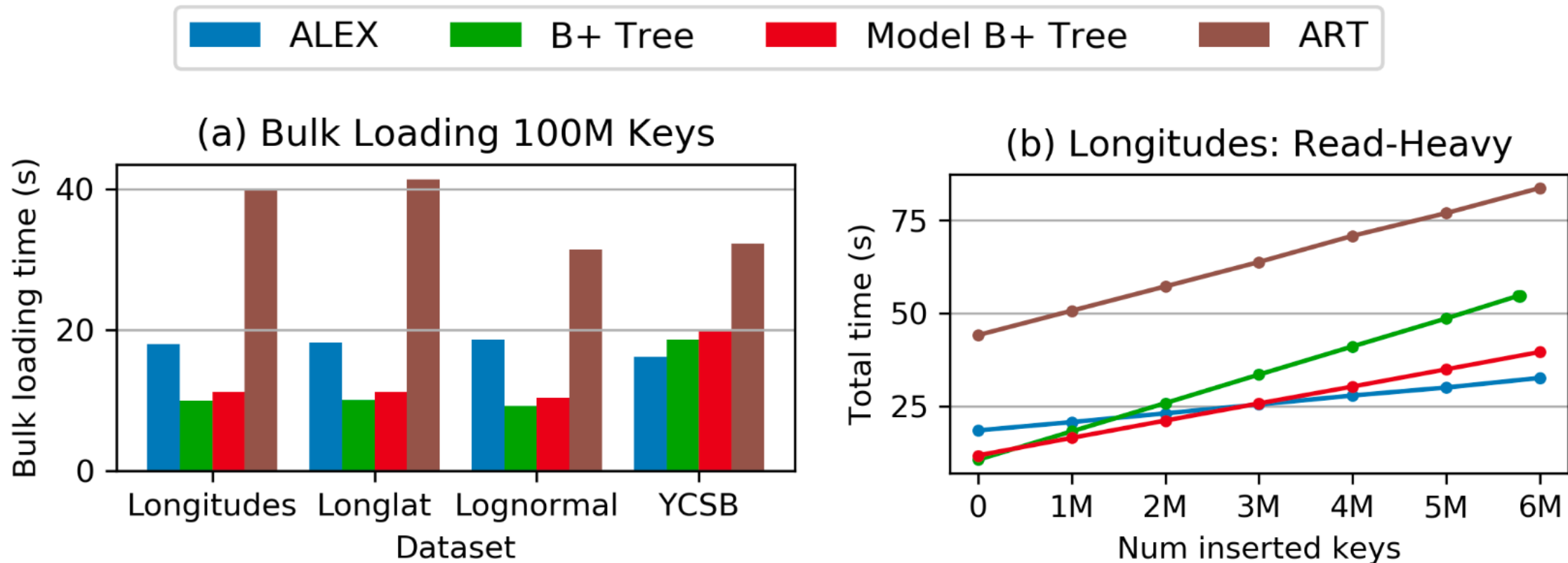
Alex maintains high throughput when scaling to very large datasets

Benchmarks (from the paper)



Alex achieve smaller prediction error than Learned Index

Any caveats ?



Alex takes more time to bulk load than B+Tree and a Model B+Tree

Any caveats ?

Be careful of very large keys!

-> Use the “largest” `float` type available while computing the linear models' parameters

```
func (self *LinearModelBuilder) Add(x float64, y float64) {  
    self.count++  
    self.xSum += x  
    self.ySum += y  
  
    self.xxSum += x * x  
    self.xySum += x * y  
  
    self.xMin = min(x, self.xMin)  
    self.xMax = max(x, self.xMax)  
    self.yMin = min(y, self.yMin)  
    self.yMax = max(y, self.yMax)  
}
```

Insights and opinions

- Hard to implement, we have to be very careful to all possible corner cases (3k lines of code just for inserting)
- Space for more research/improvement
- Wouldn't use in production yet