Artifacts Available

Artifacts Evaluated Functional

Results Replicated

XIndex: A Scalable Learned Index for Multicore Data Storage

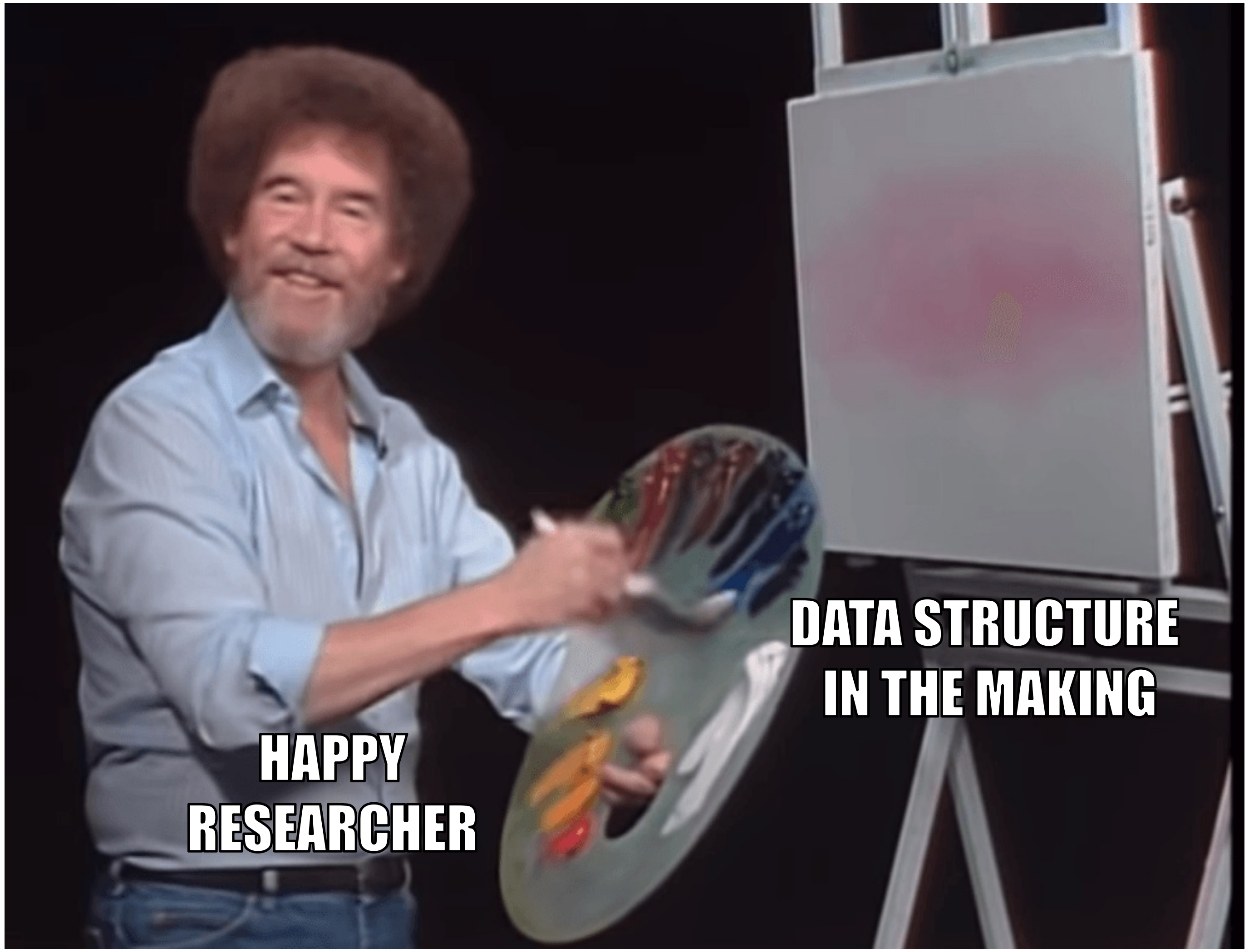
Chuzhe Tang, Youyun Wang, Zhiyuan Dong, Gansen Hu
Zhaoguo Wang, Minjie Wang, Haibo Chen

SHANGHAI JIAO TONG UNIVERSITY 1896

IPADS
INSTITUTE OF PARALLEL
AND DISTRIBUTED SYSTEMS

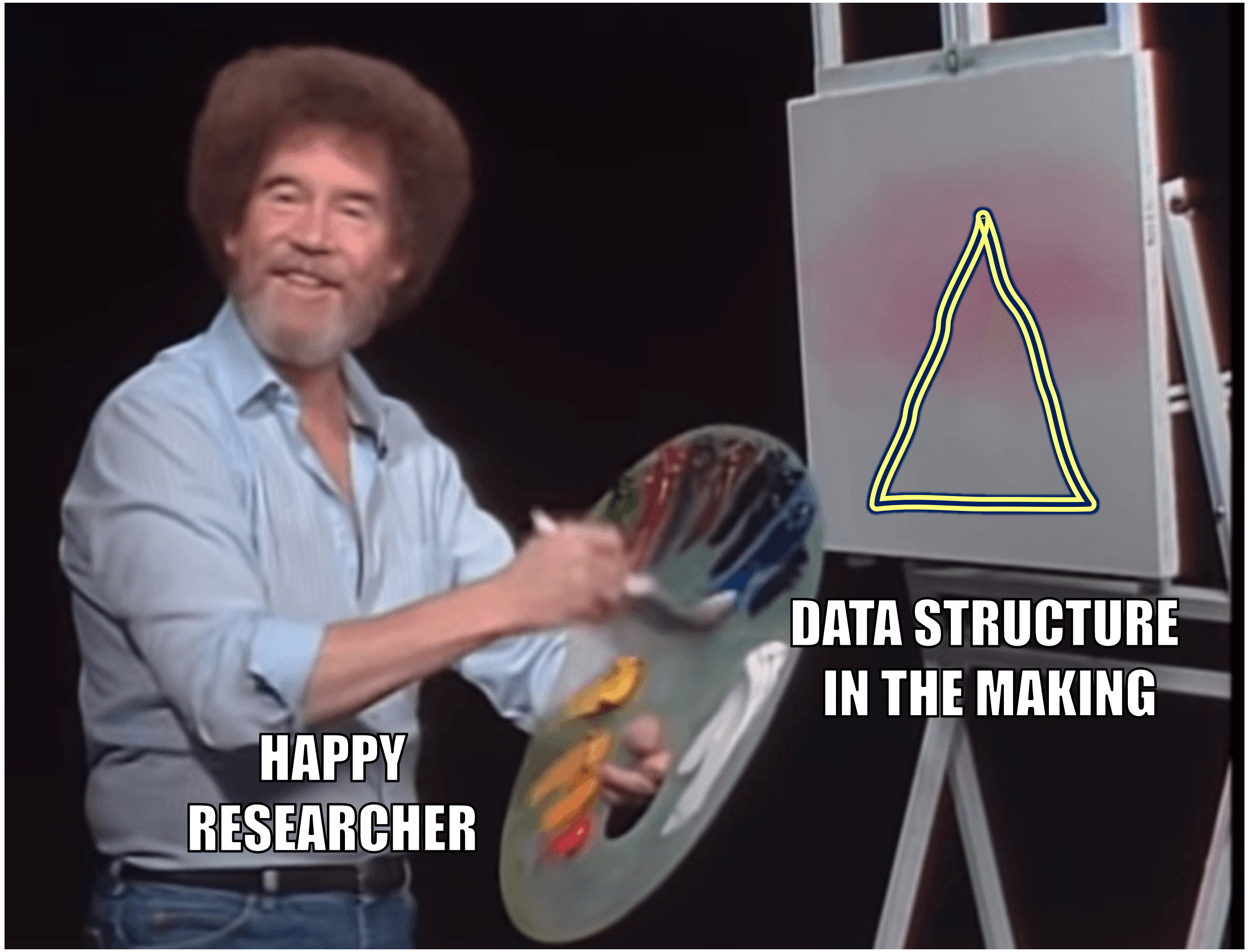
NYU

How to build a data structure



@ www.bobross.com

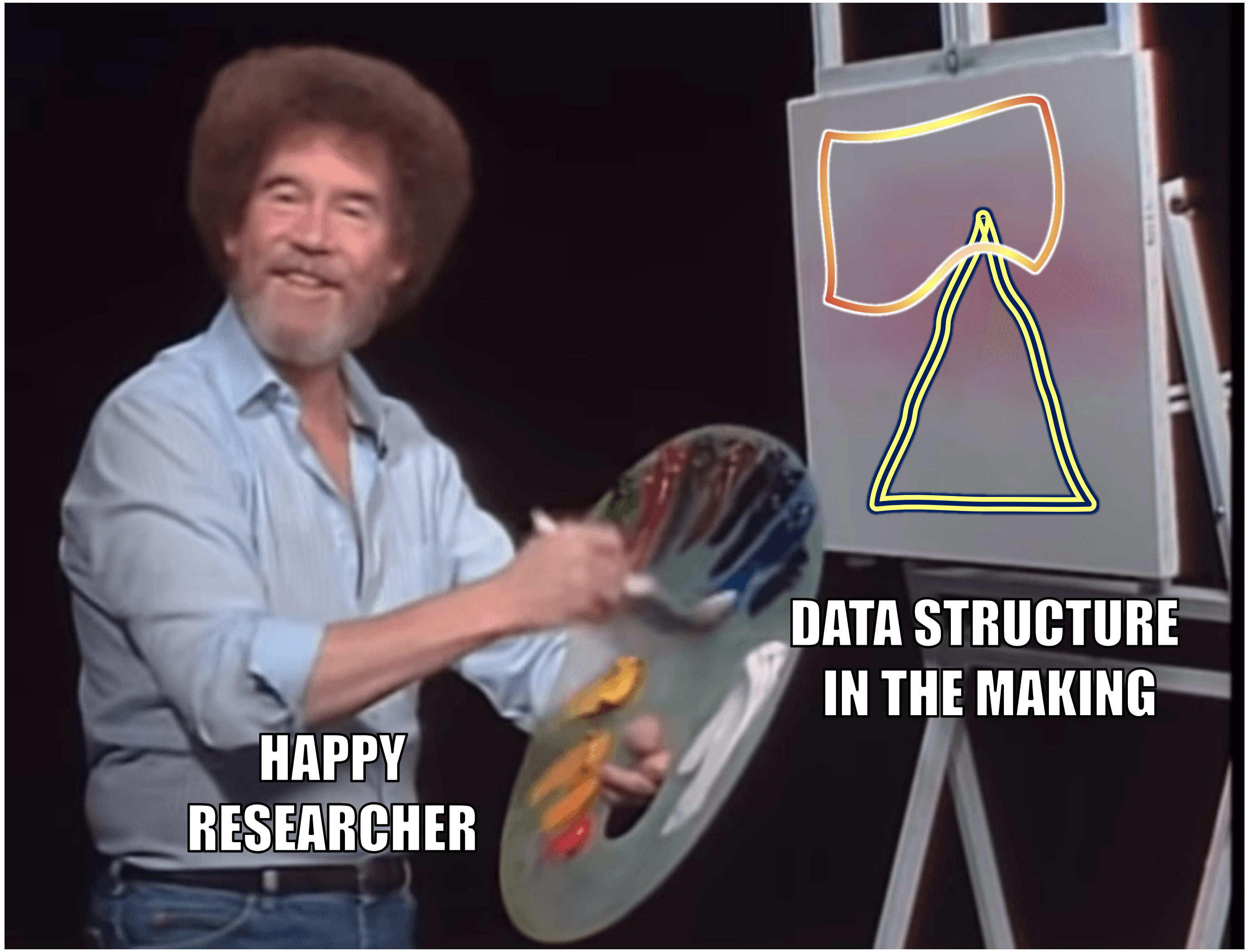
How to build a data structure



@ www.bobross.com

"Gotta have some low latency."

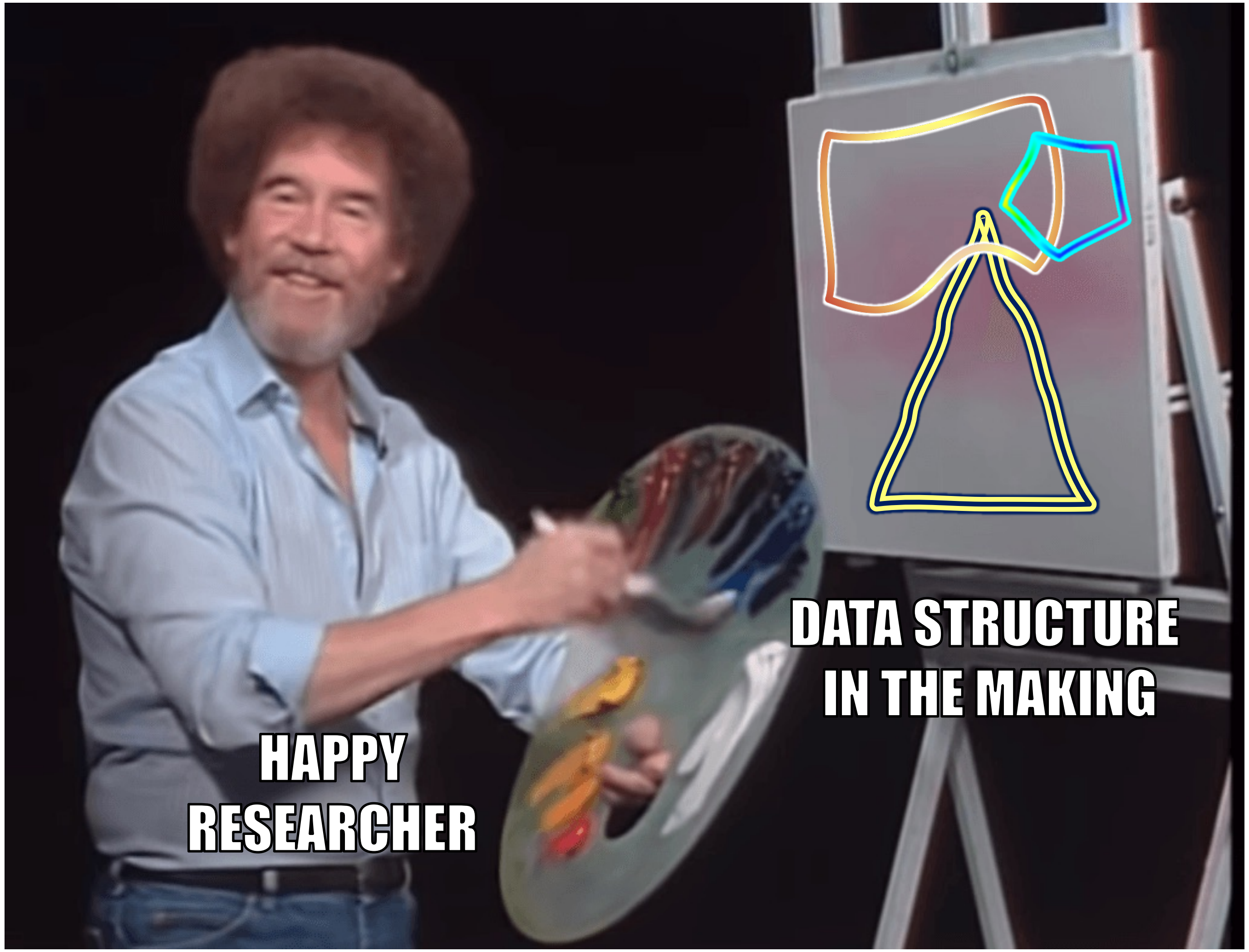
How to build a data structure



@ www.bobross.com

"Gotta have some scalability."

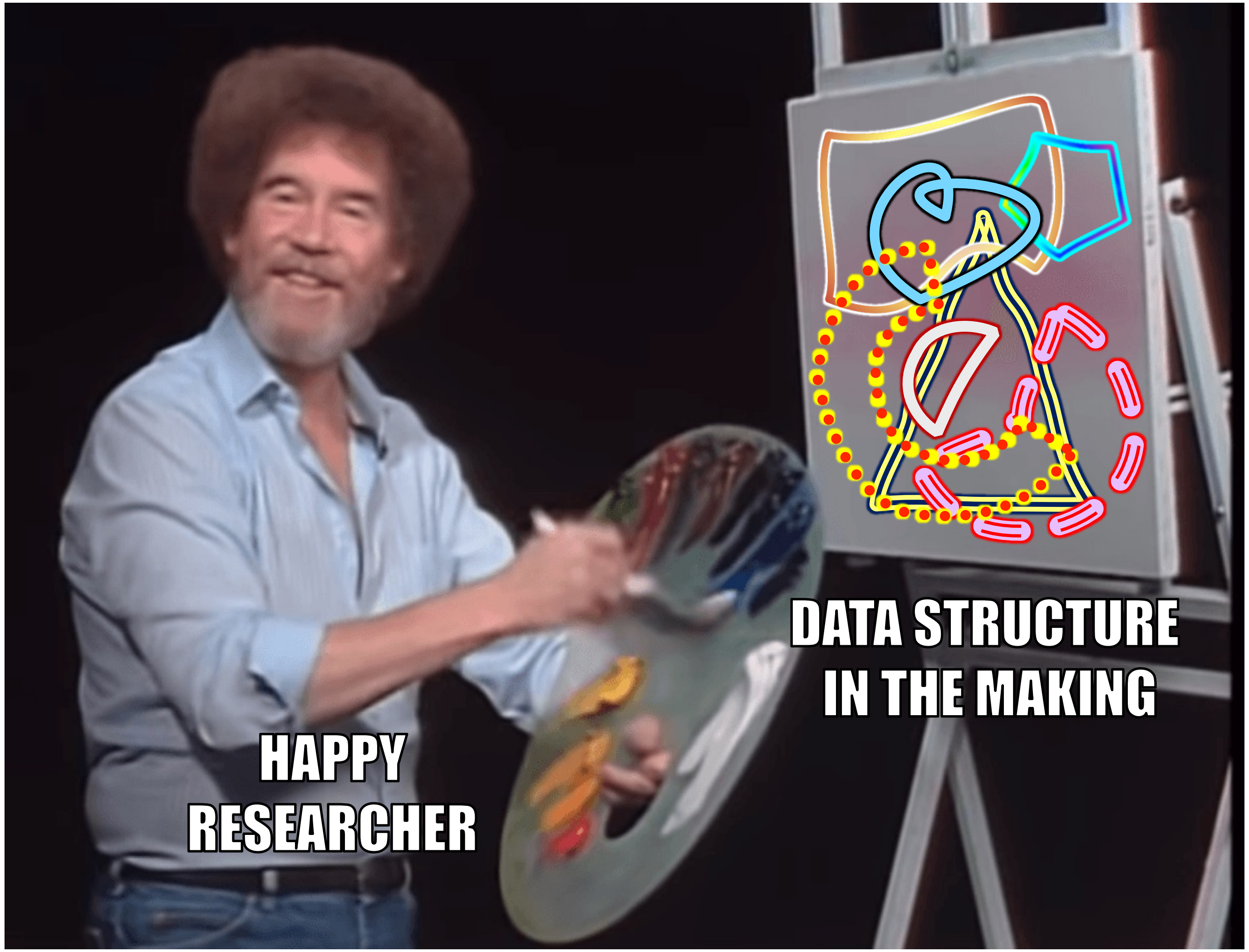
How to build a data structure



@ www.bobross.com

"Gotta have some consistency."

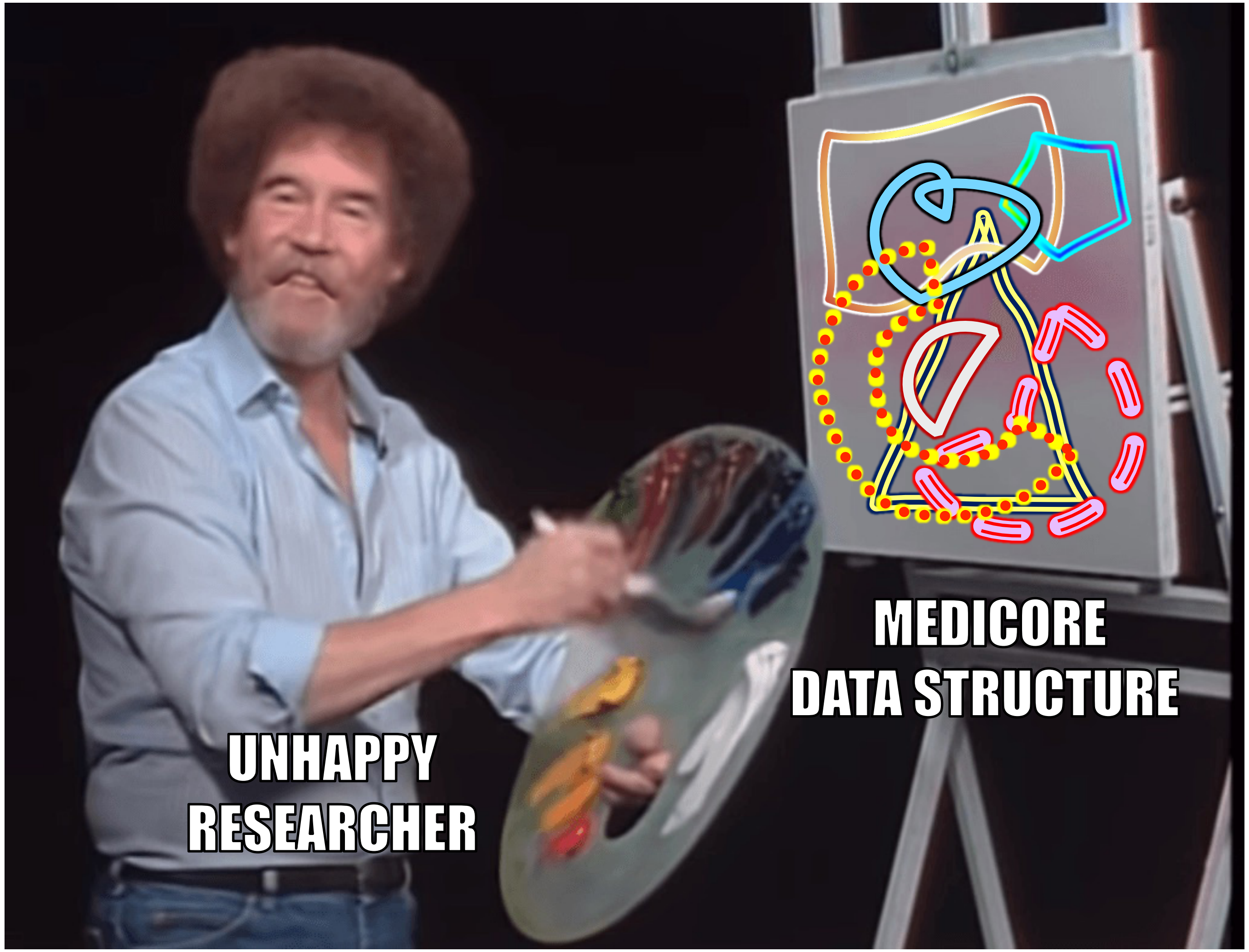
How to build a data structure



@ www.bobross.com

"Gotta have some small memory footprint, durability, adaptiveness, ..."

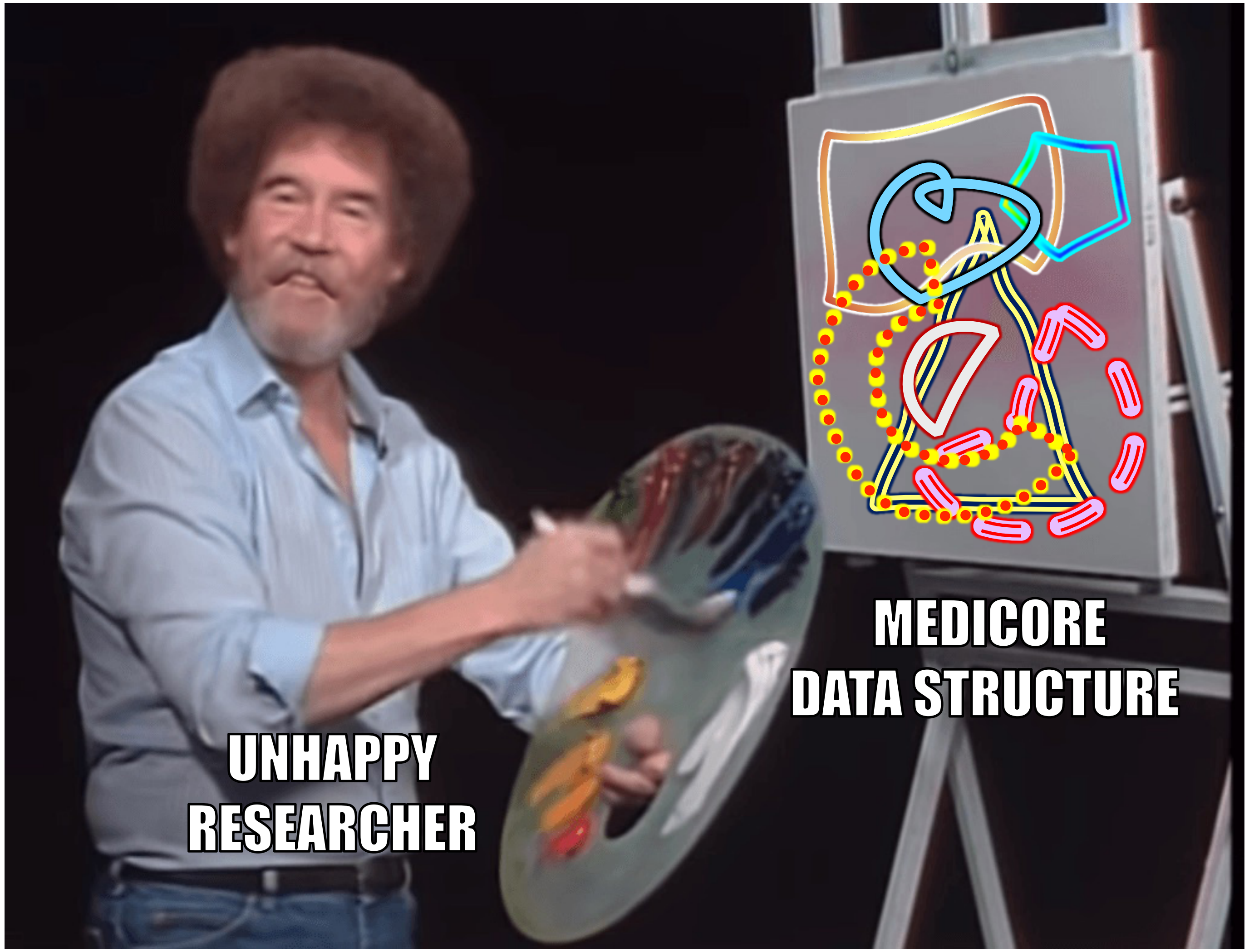
How to build a data structure



@ www.bobross.com

"Oops!"

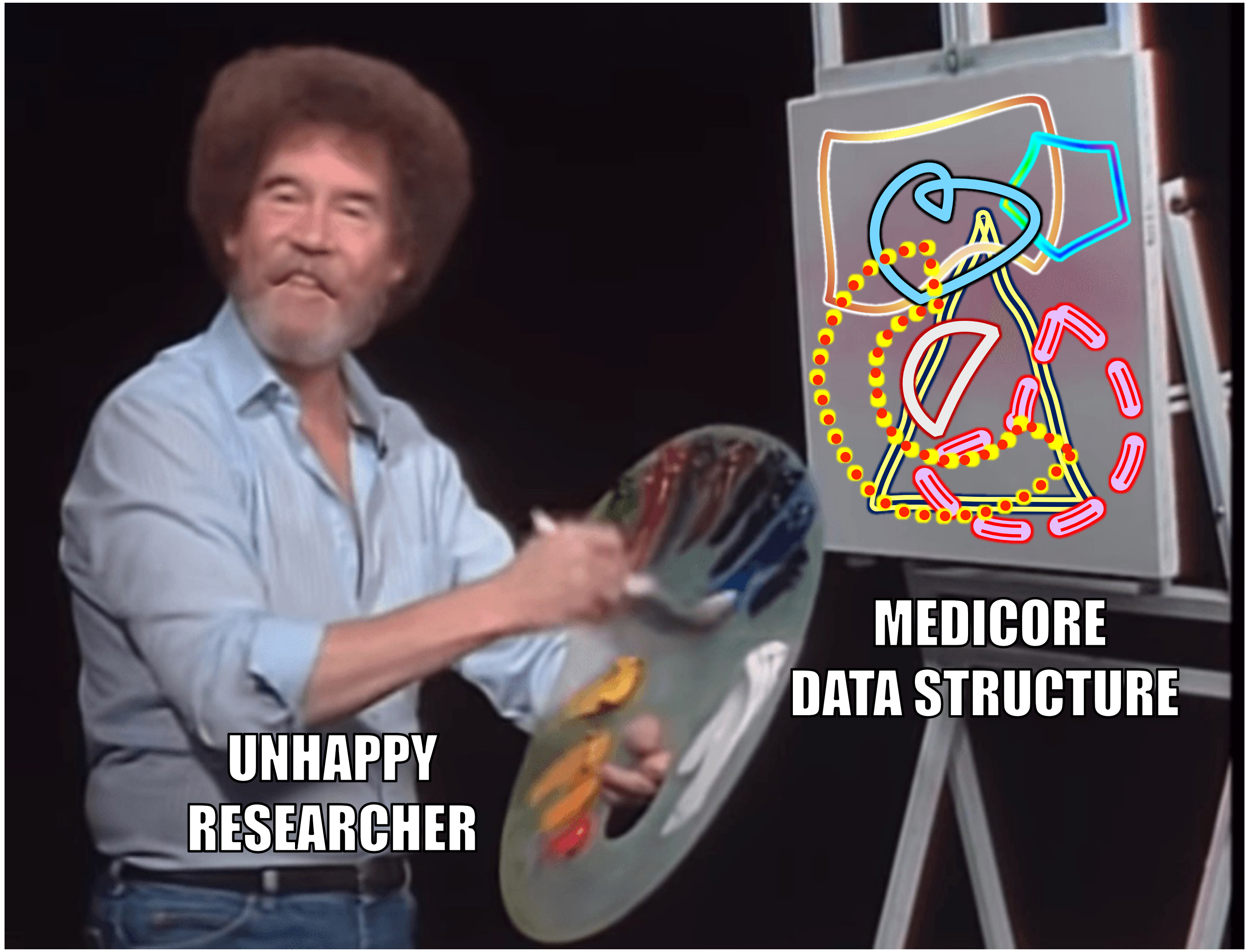
How to build a data structure



@ www.bobross.com

@ Jeff Dean

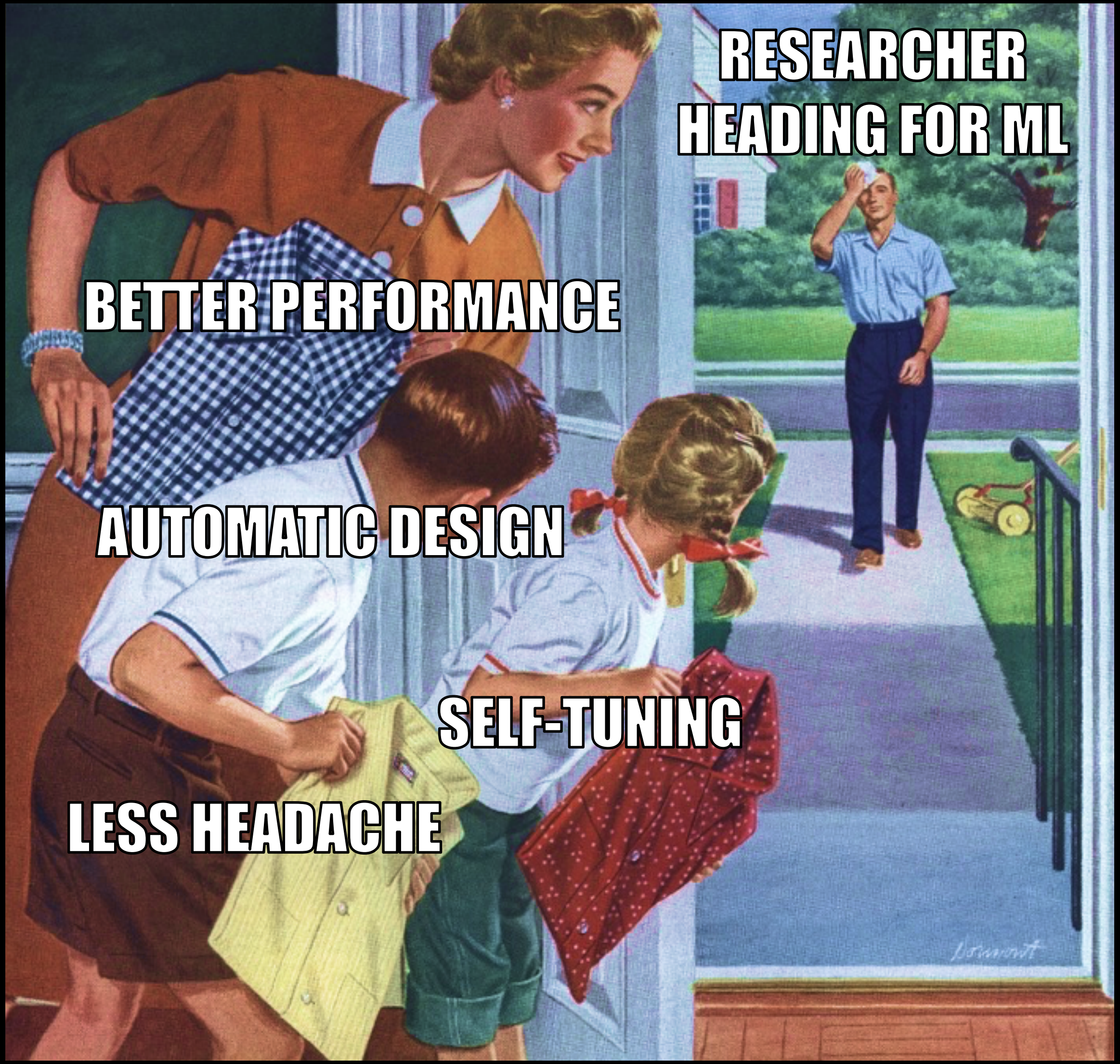
How to build a data structure



@ www.bobross.com

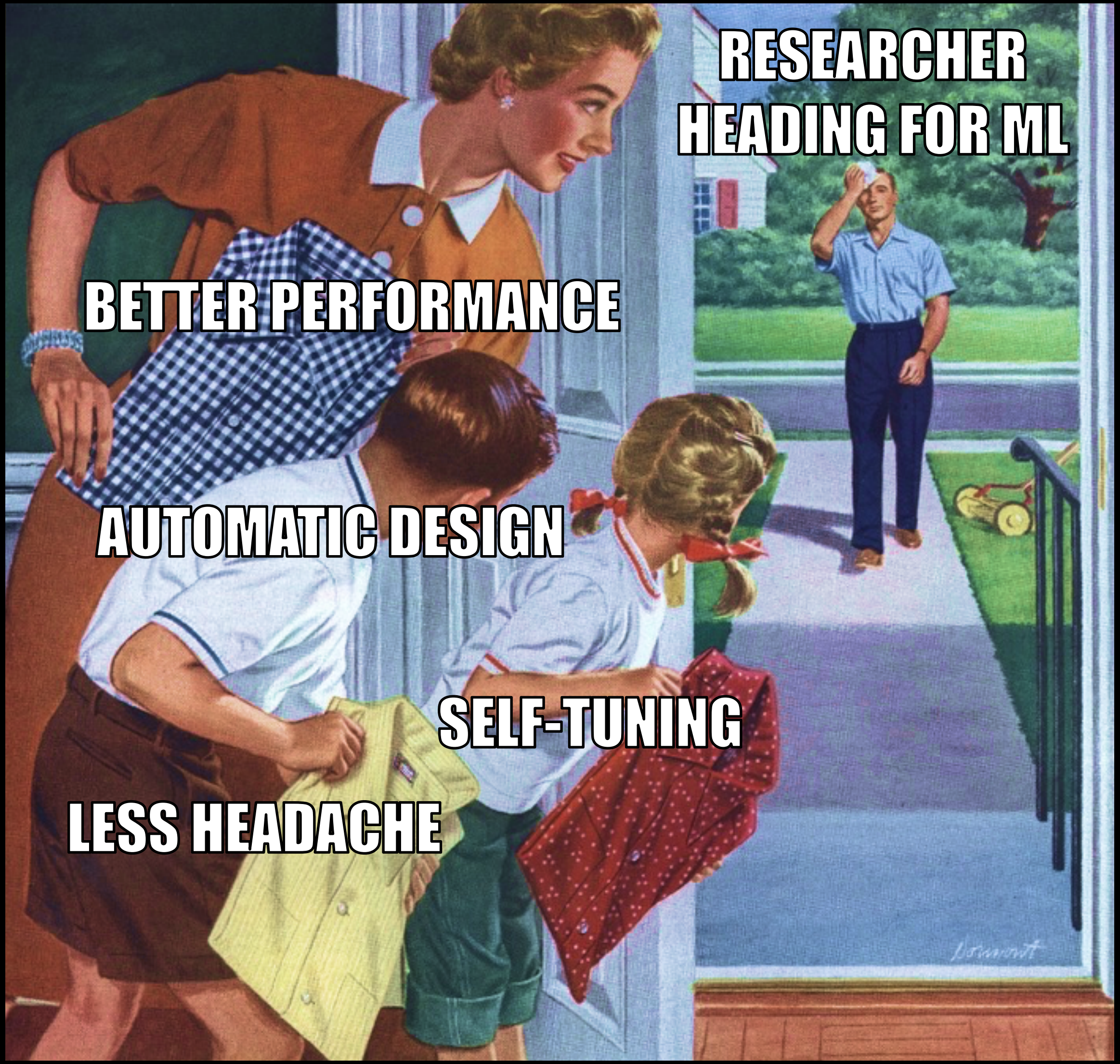
Let's do some Machine Learning!

@ Jeff Dean



@ LIFE magazine

Expectation



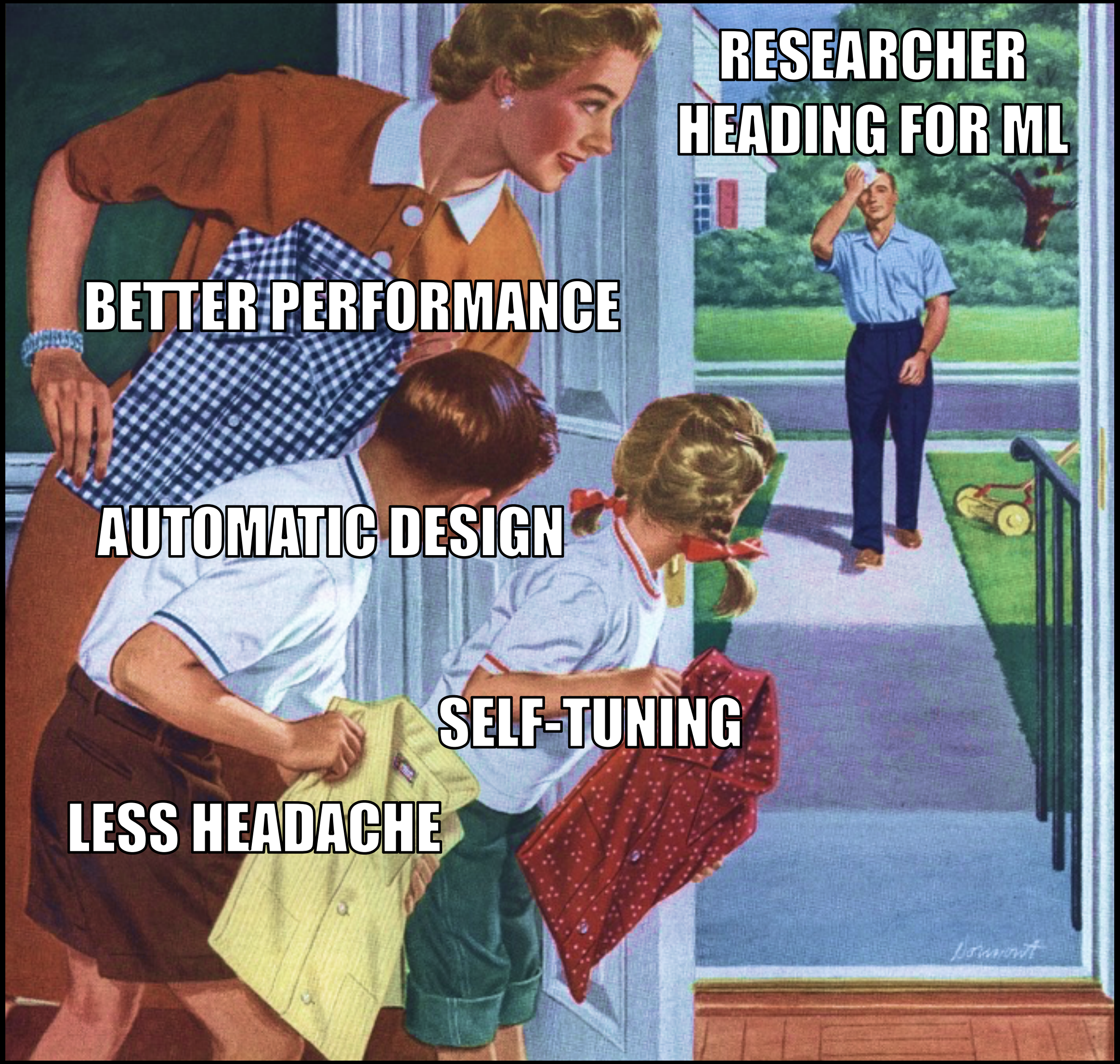
@ LIFE magazine

Expectation

RESEARCHER HEADING FOR ML

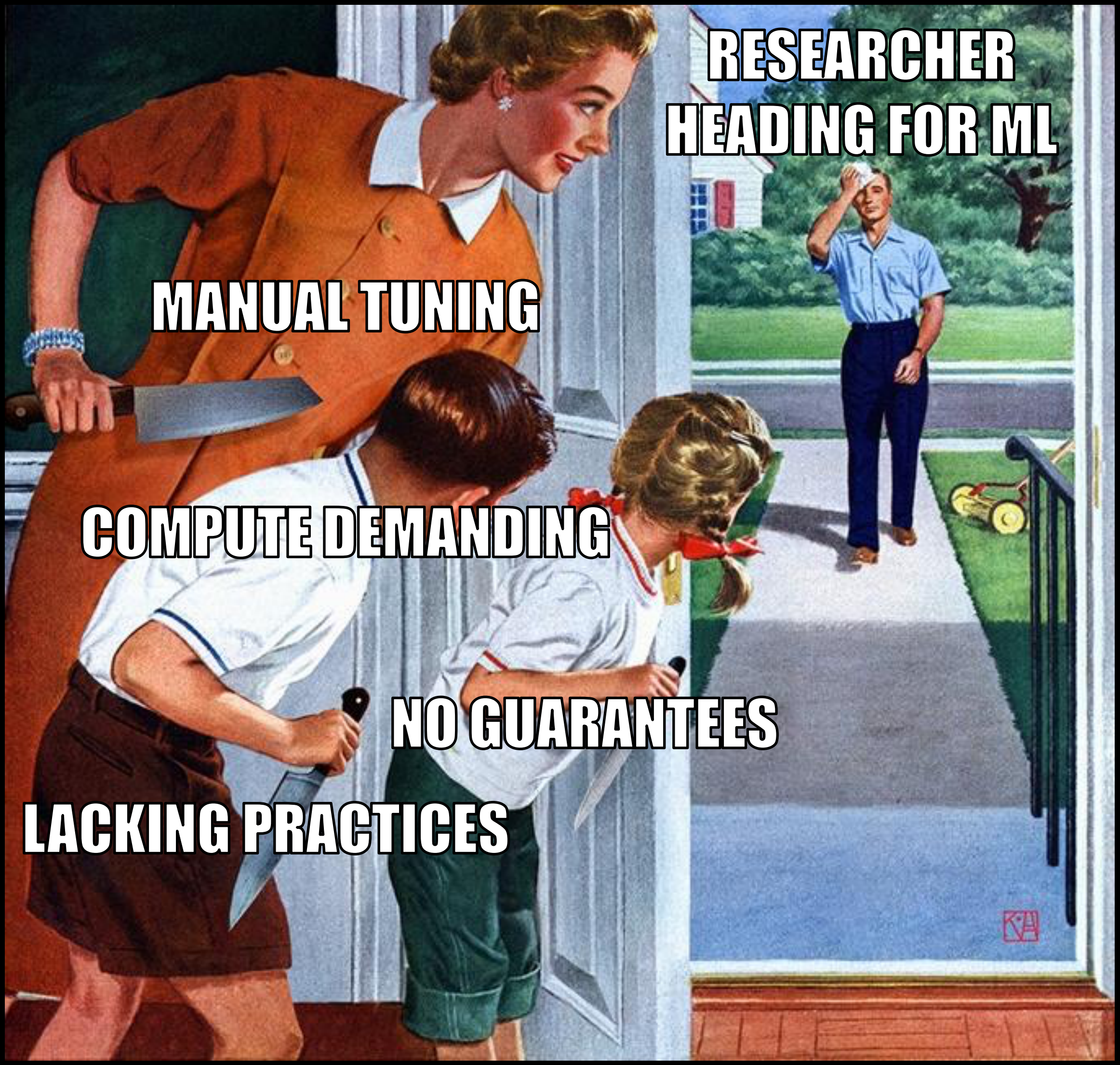
@ KARRRASKA

Reality



@ LIFE magazine

Expectation



@ KARRRASKA

Reality

Today's talk

Data Structure Design

Machine Learning

Today's talk

Question 1 **Does ML work?**

Data Structure Design

Machine Learning

Today's talk

Question 1 Does ML work?

Yes, but not perfectly

Data Structure Design

Machine Learning

Today's talk

Question 1 Does ML work?

Yes, but not perfectly

Question 2 How to make ML work?

Data Str
Desi

e Learning

Today's talk

Question 1 Does ML work?

Yes, but not perfectly

Question 2 How to make ML work?

Systematic approaches

Data Str
Desi

e Learning

Today's talk

Question 1 Does ML work?

Yes, but not perfectly

Data Str
Desi

e Learning

Question 2 How to make ML work?

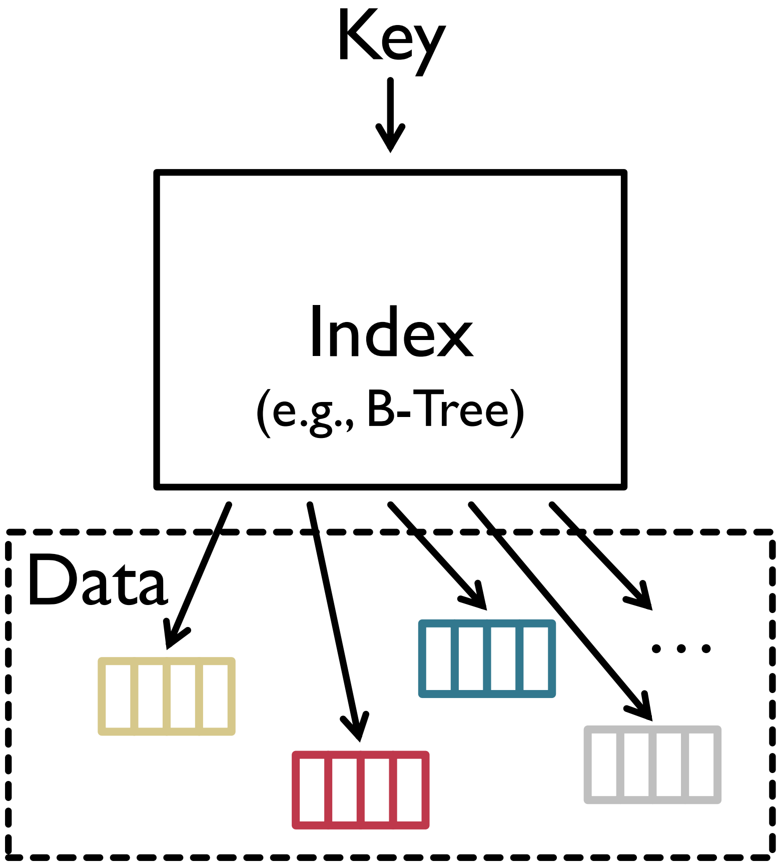
Systematic approaches

We answer with the learned index

Background: the learned index

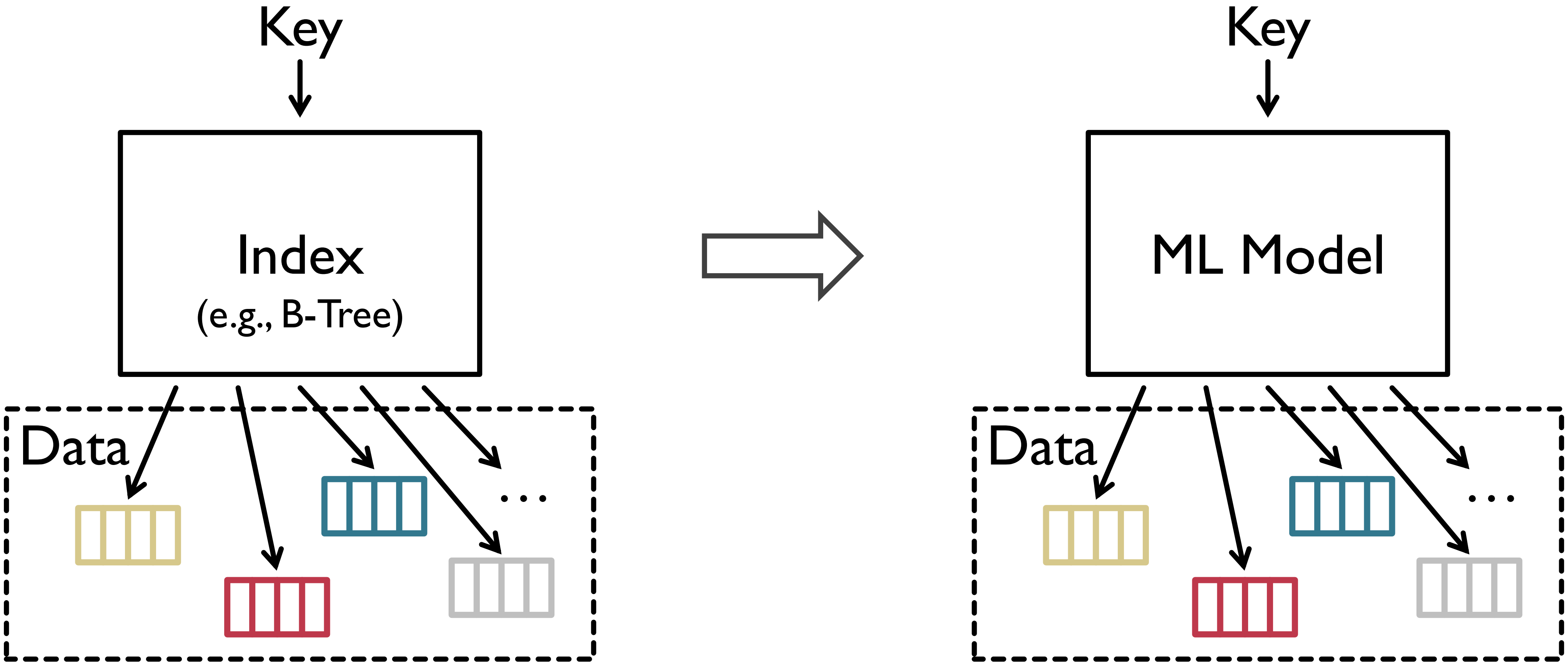
Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index



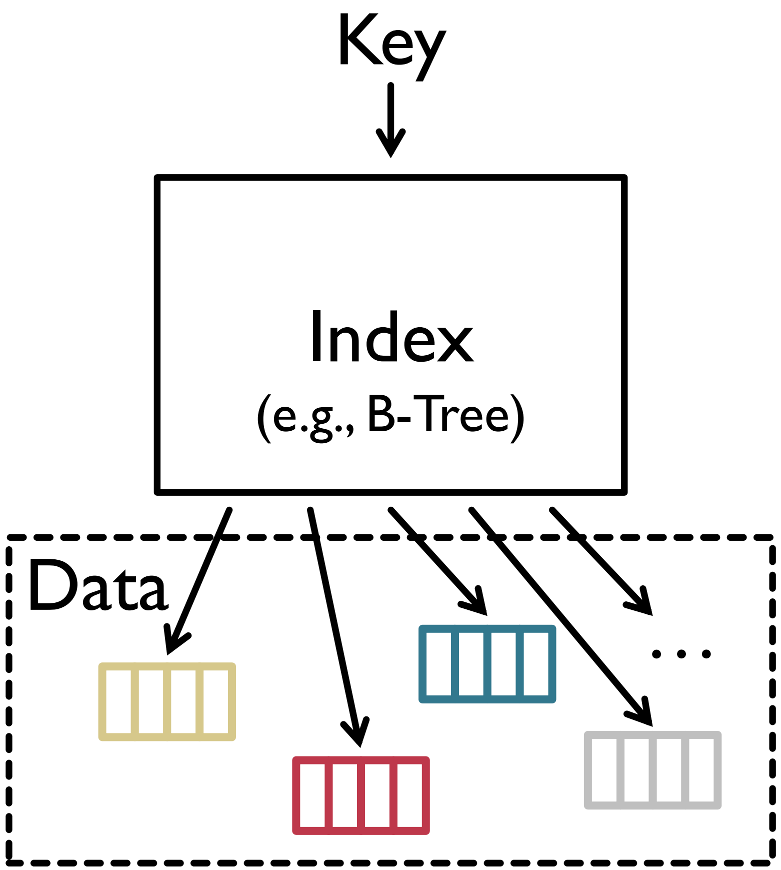
Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index



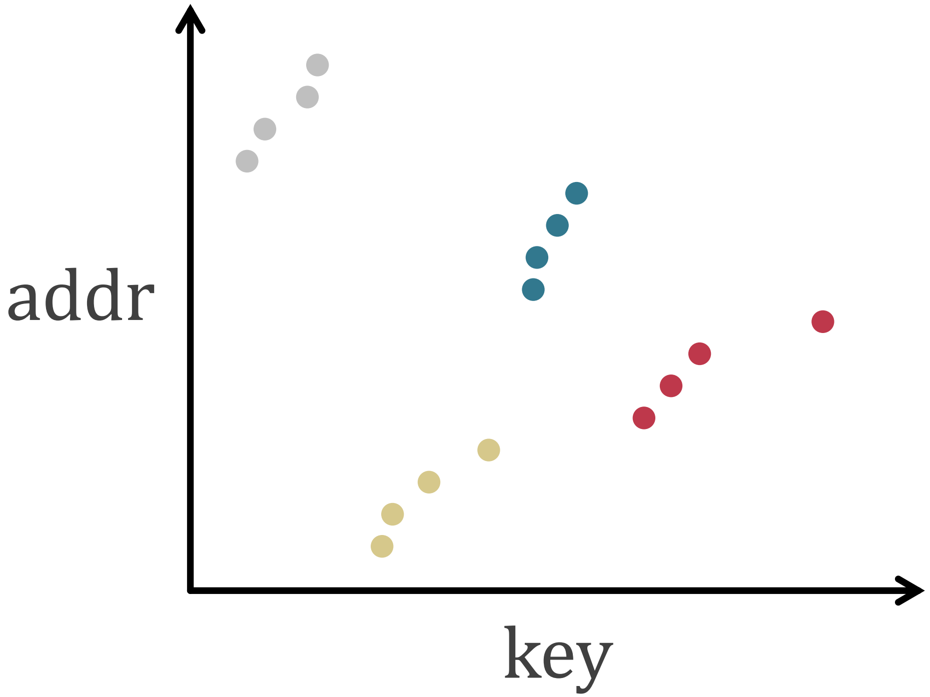
Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

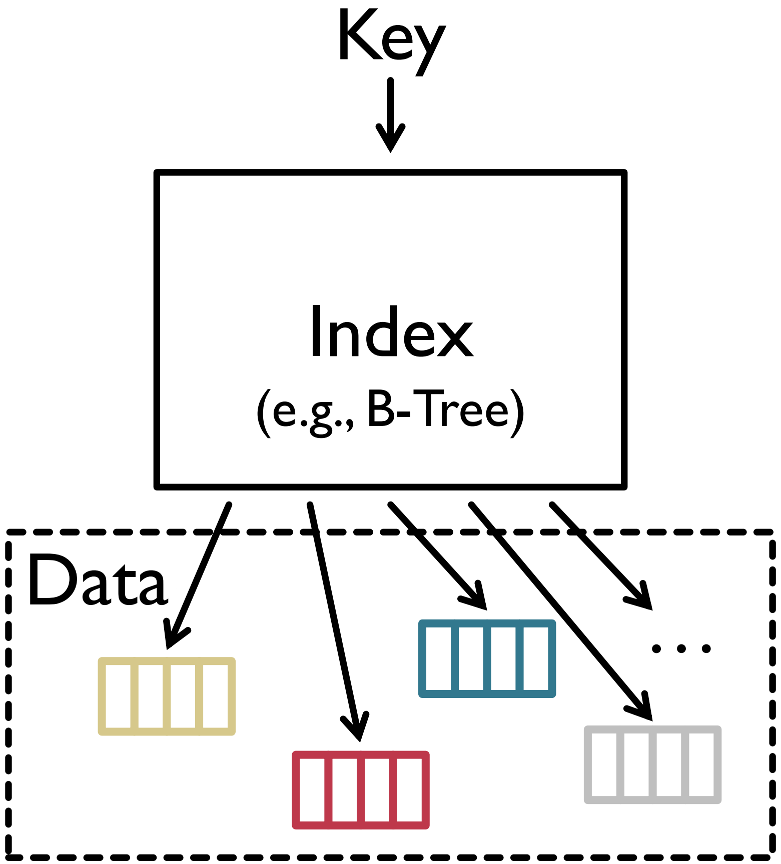
Background: the learned index



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

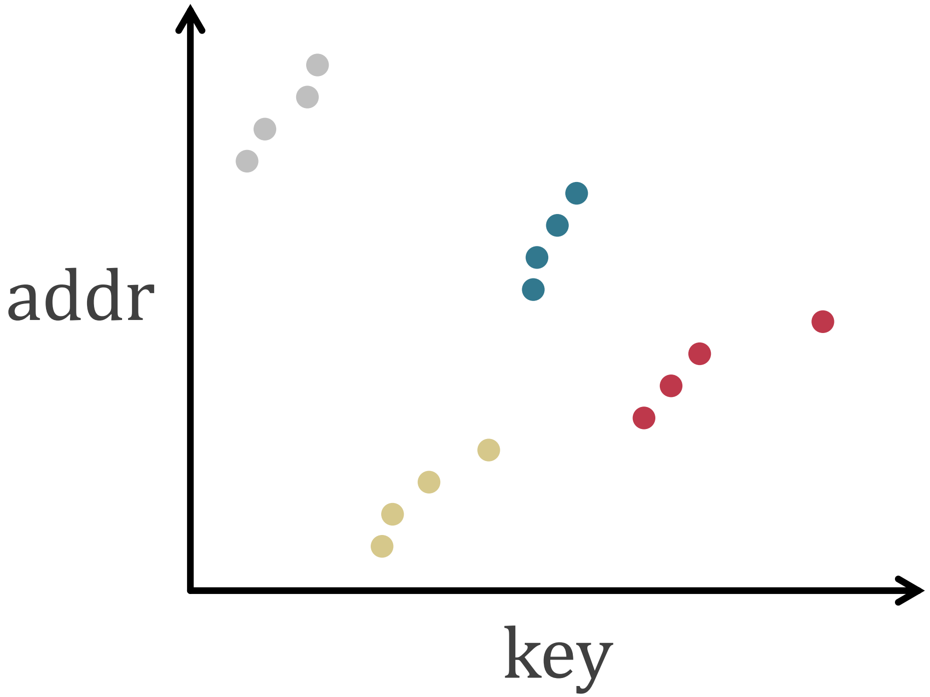


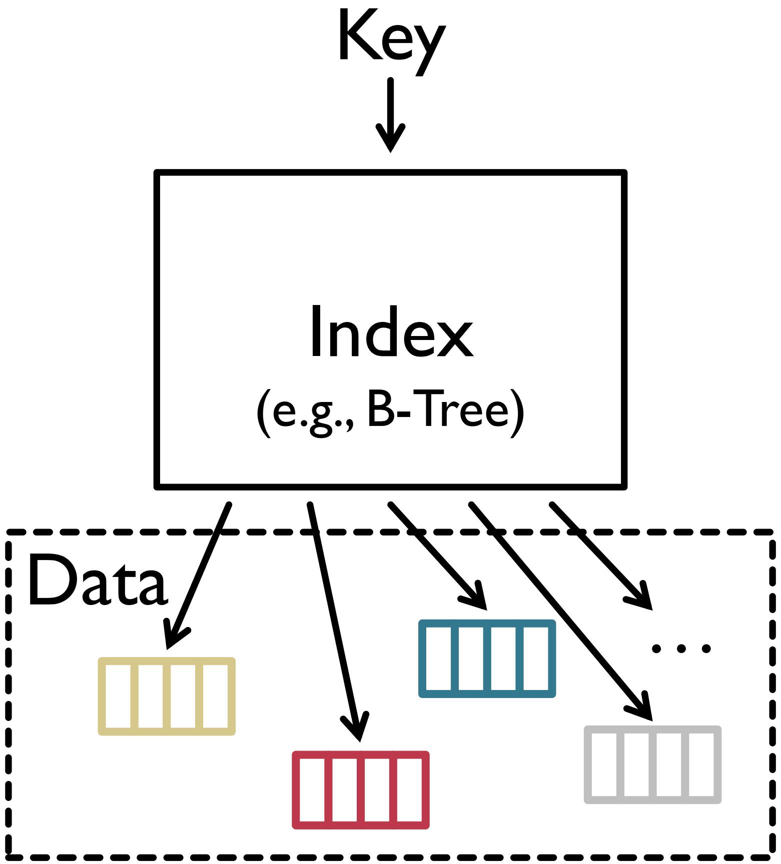


Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

- **With contiguously sorted data**

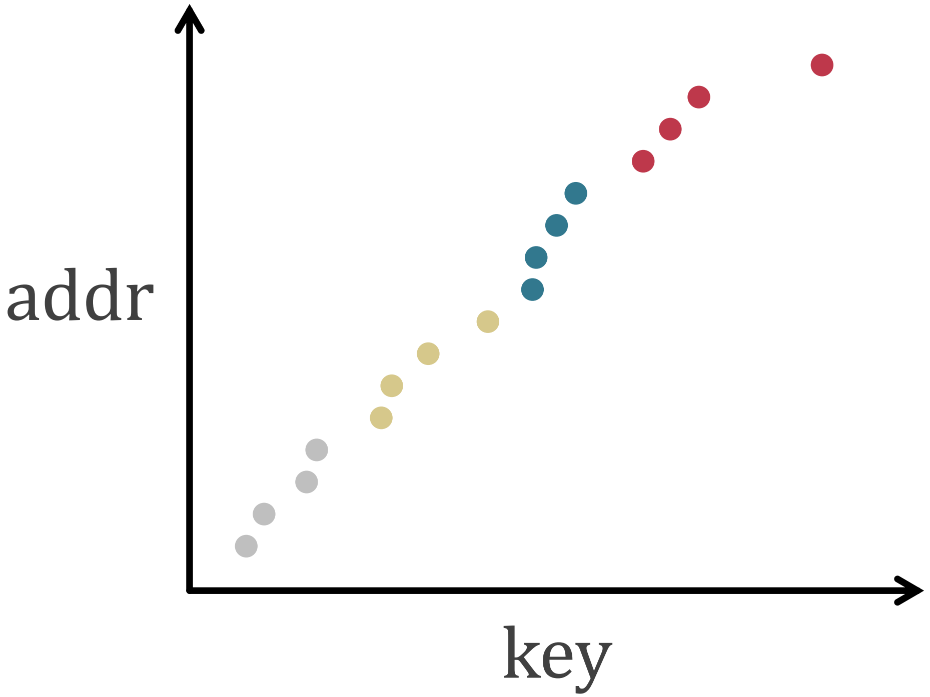


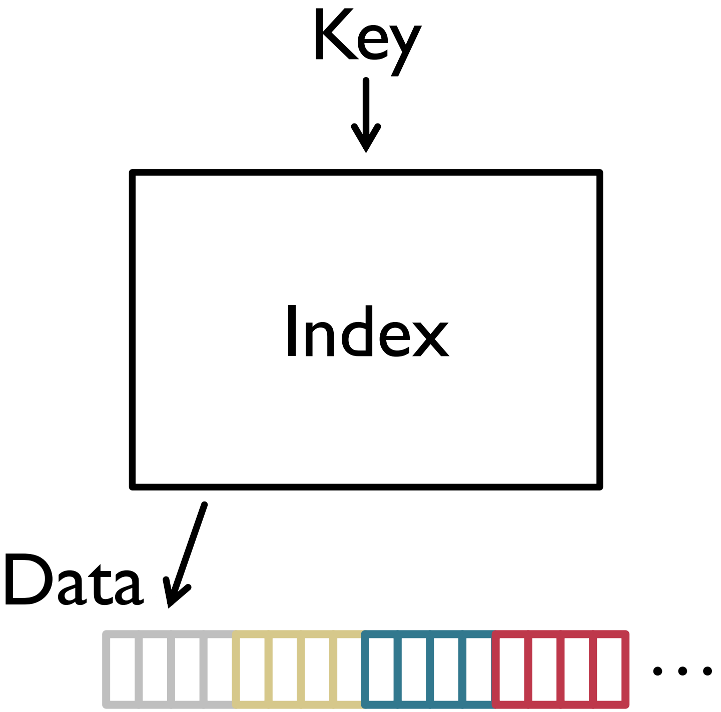


Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

- **With contiguously sorted data**

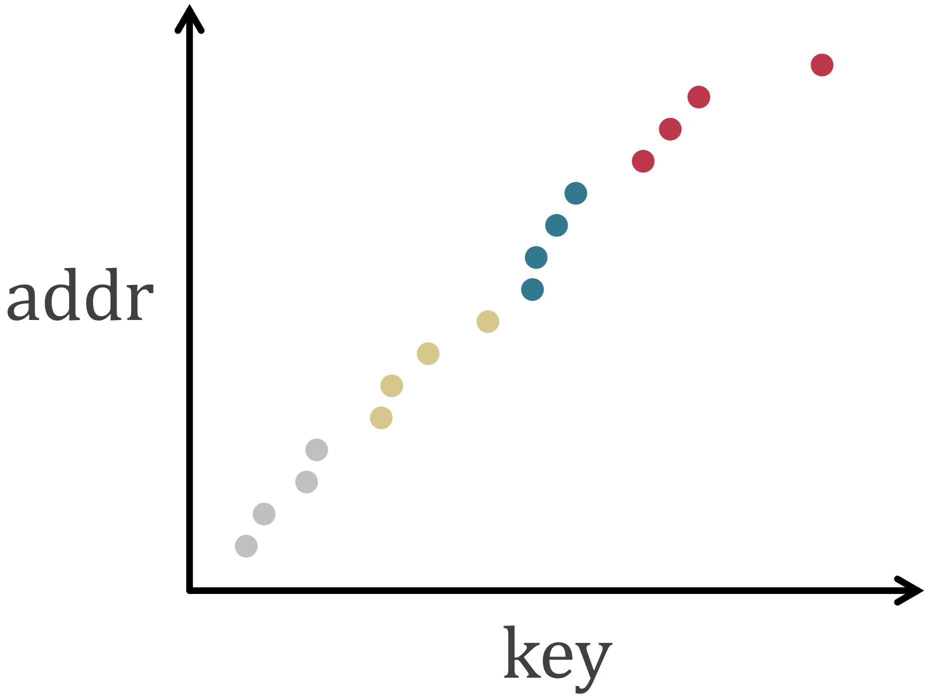


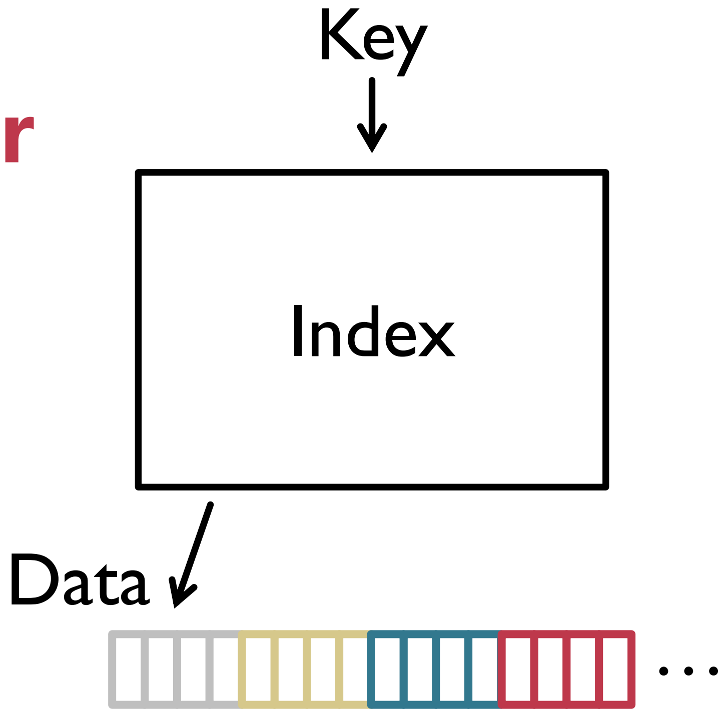


Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

- **With contiguously sorted data, index functions become simpler**

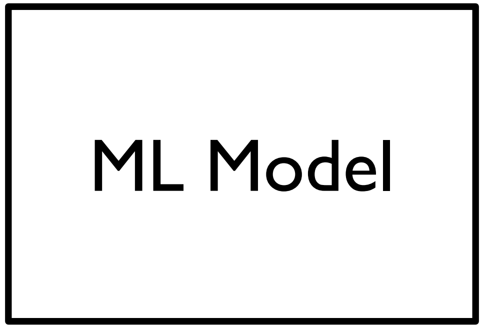




Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

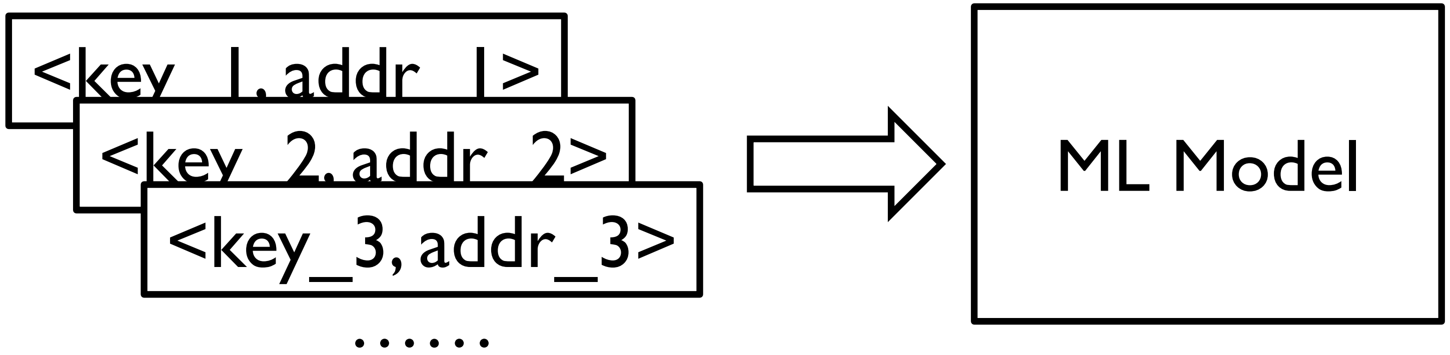
1. Sort data, train model with key-address mapping



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

1. **Sort data, train model with key-address mapping**



<key_1, addr_1>
<key_2, addr_2>
<key_3, addr_3>
……

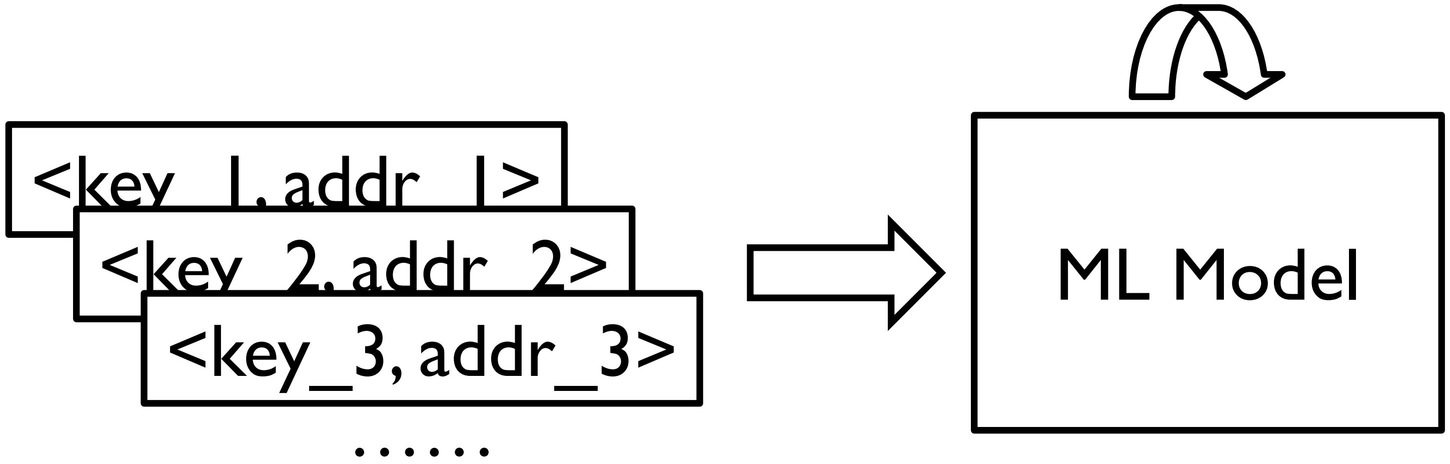
ML Model

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

1. **Sort data, train model with key-address mapping**

Train to overfit the data



<key_1, addr_1>

<key_2, addr_2>

<key_3, addr_3>

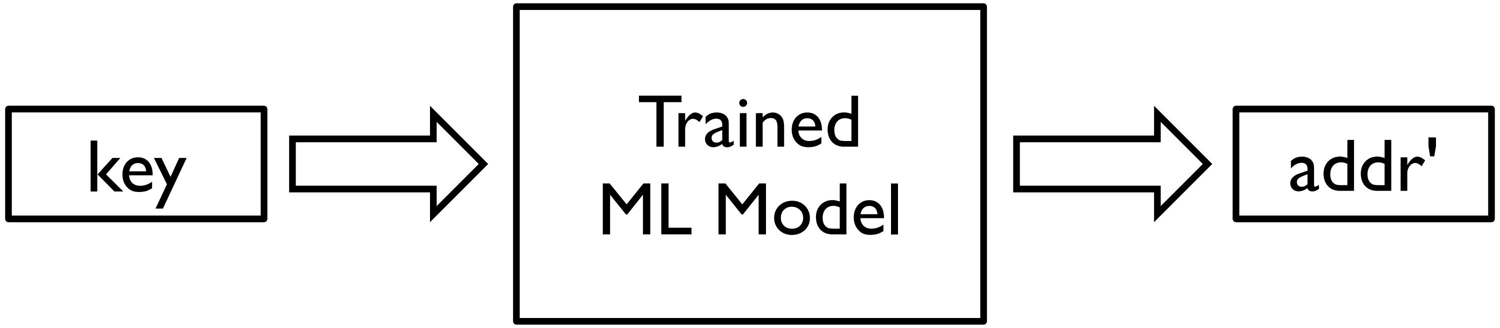
……

ML Model

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

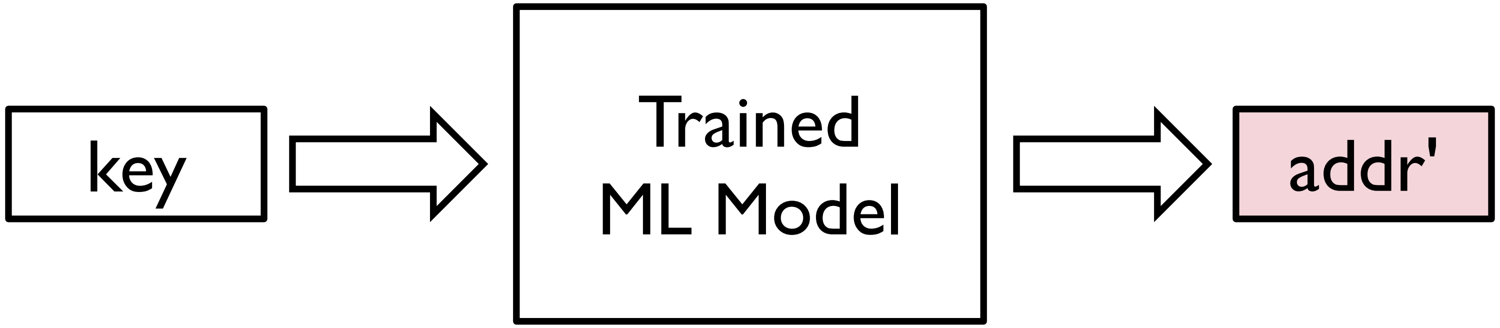
1. Sort data, train model with key-address mapping
2. Predict addresses with the trained model



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

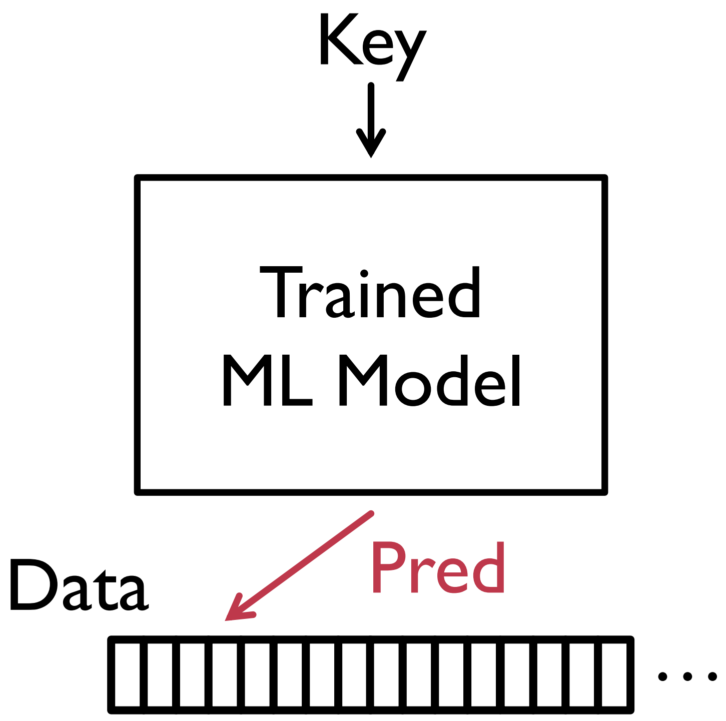
1. **Sort data, train model with key-address mapping**
2. **Predict addresses with the trained model**
   - Prediction is CLOSE, but NOT PRECISE



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

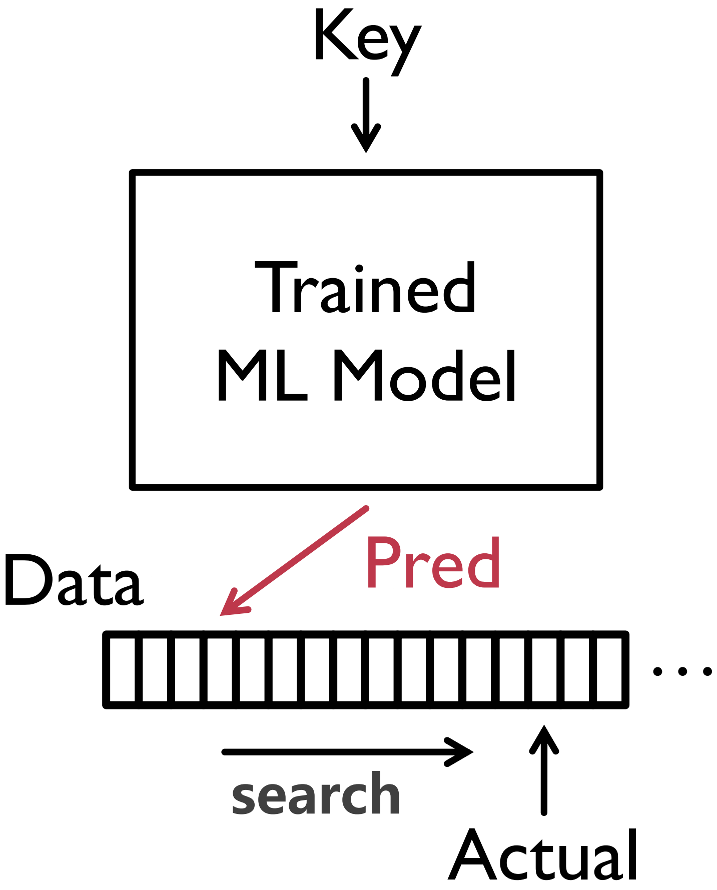
3. Search the correct position



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

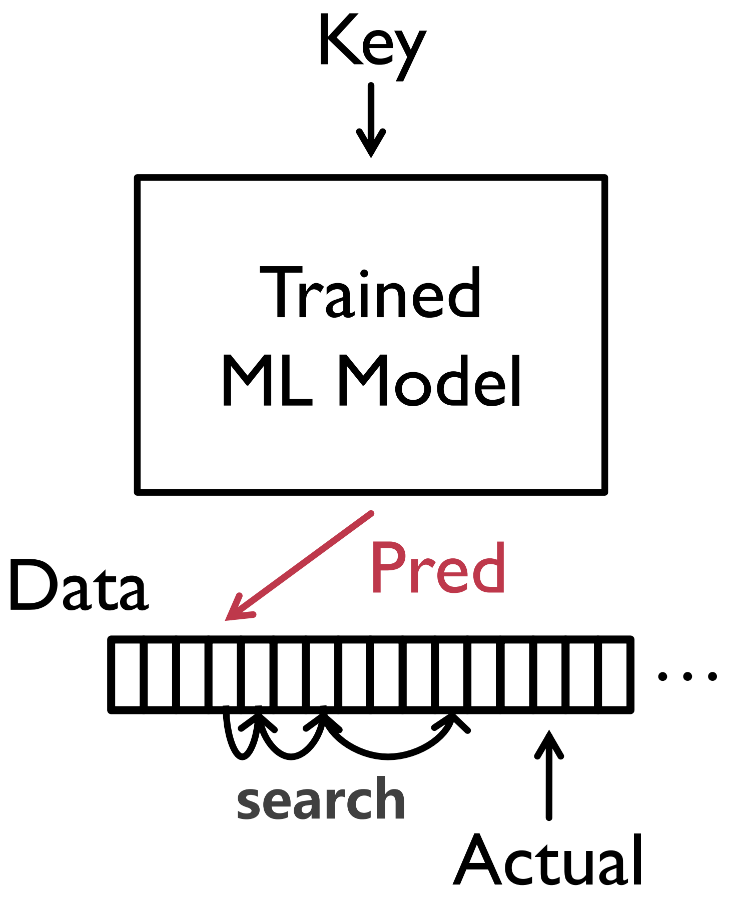
3. Search the correct position



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

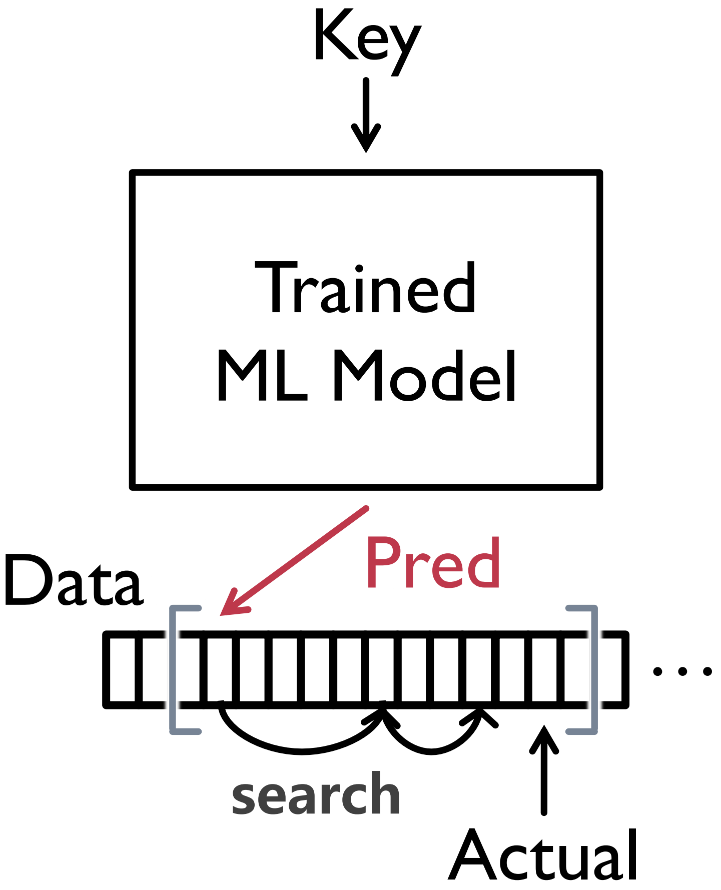
3. **Search the correct position**
   - Exponential search



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

3. **Search the correct position**
   - Exponential search
   - Binary search

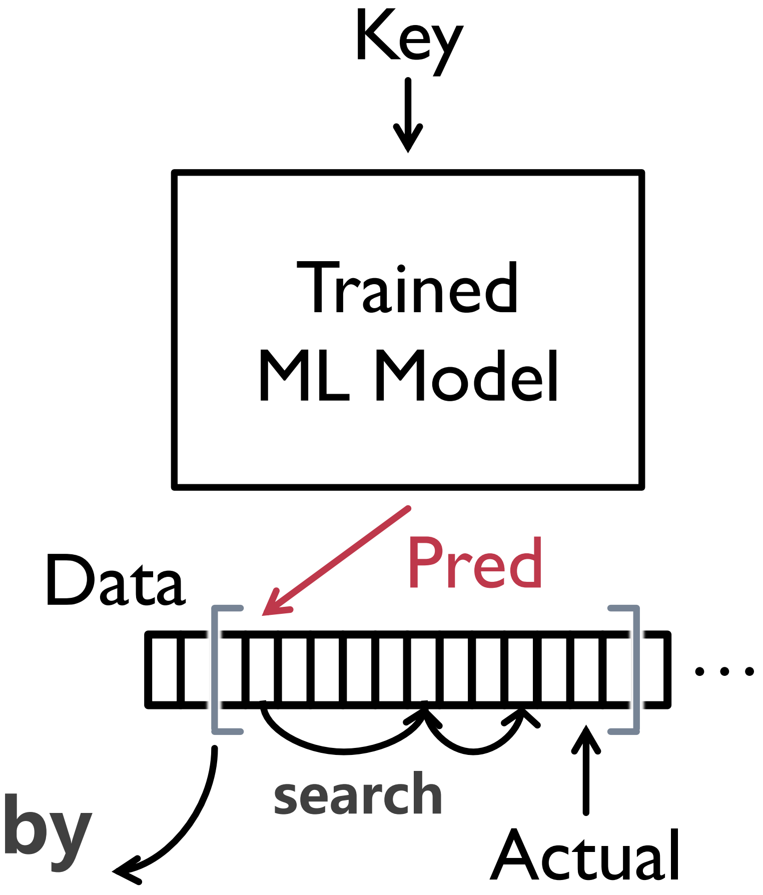


Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

3. Search the correct position

- Exponential search
- Binary search

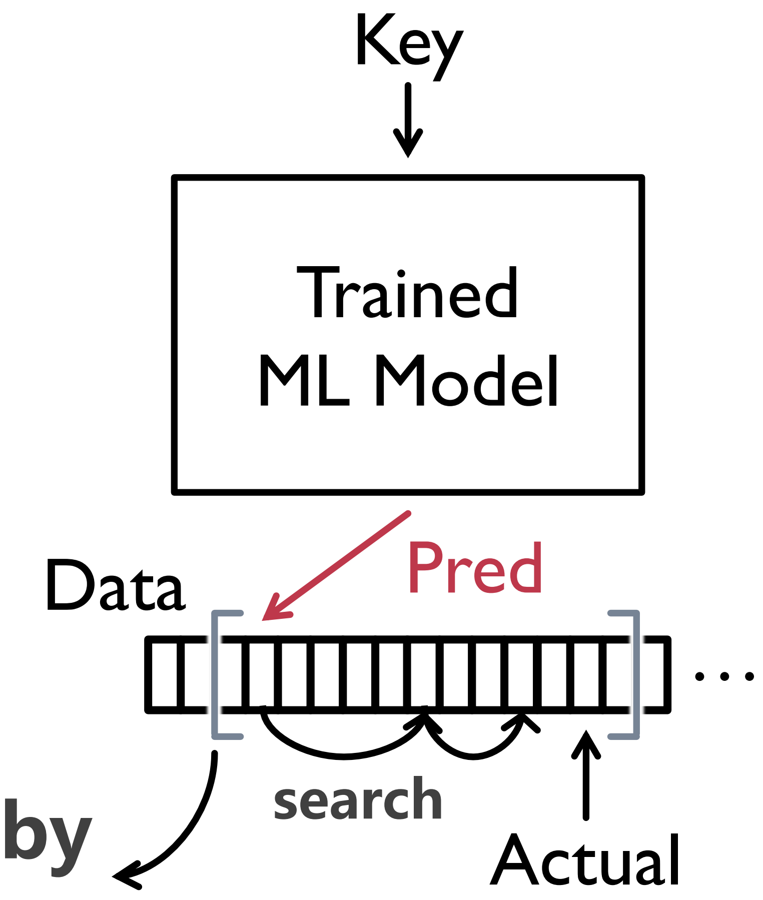


Range bounded by max/min errors

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

3. **Search the correct position**
   - Exponential search
   - Binary search
   - ……



Range bounded by max/min errors

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

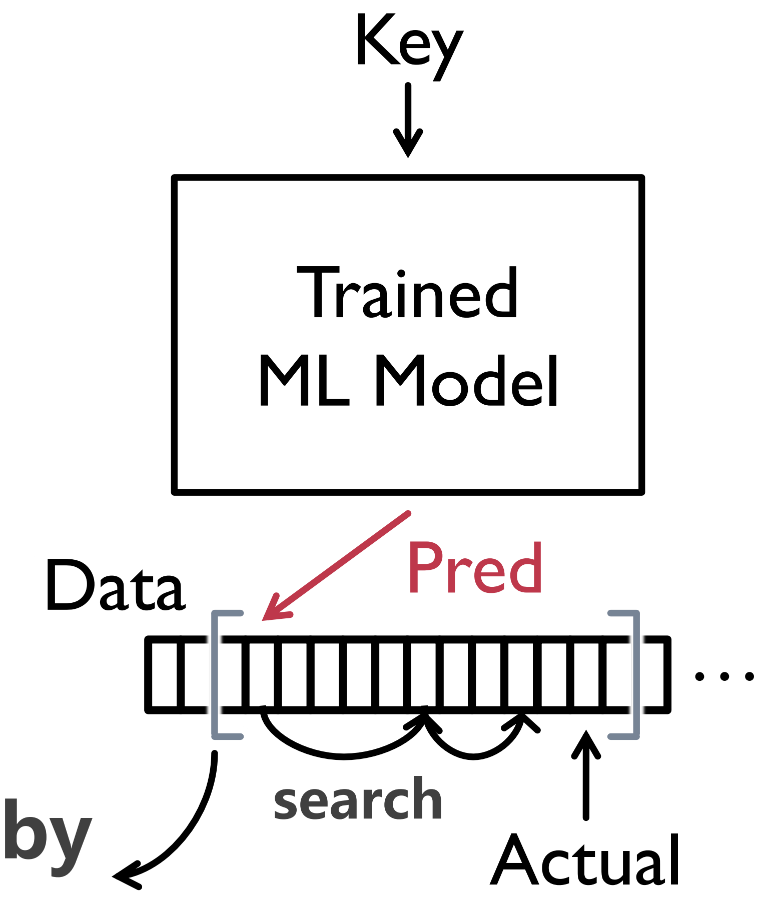
3. Search the correct position

- Exponential search
- Binary search

Smaller errors

⇩

Better search efficiency

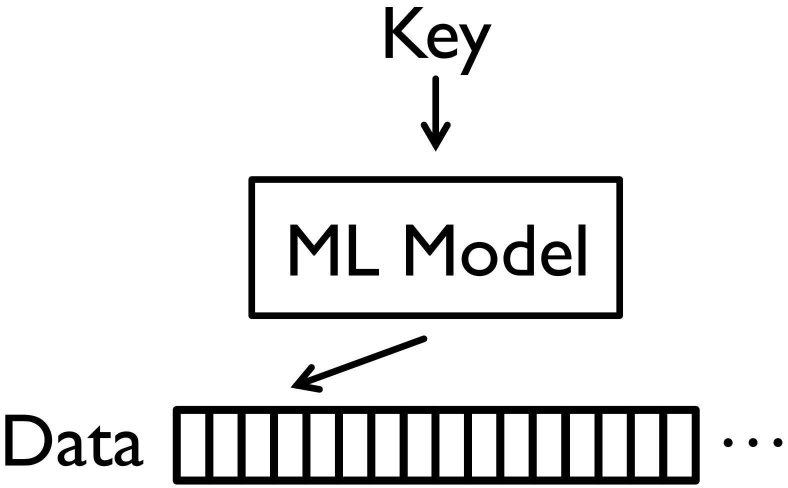


Range bounded by max/min errors

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

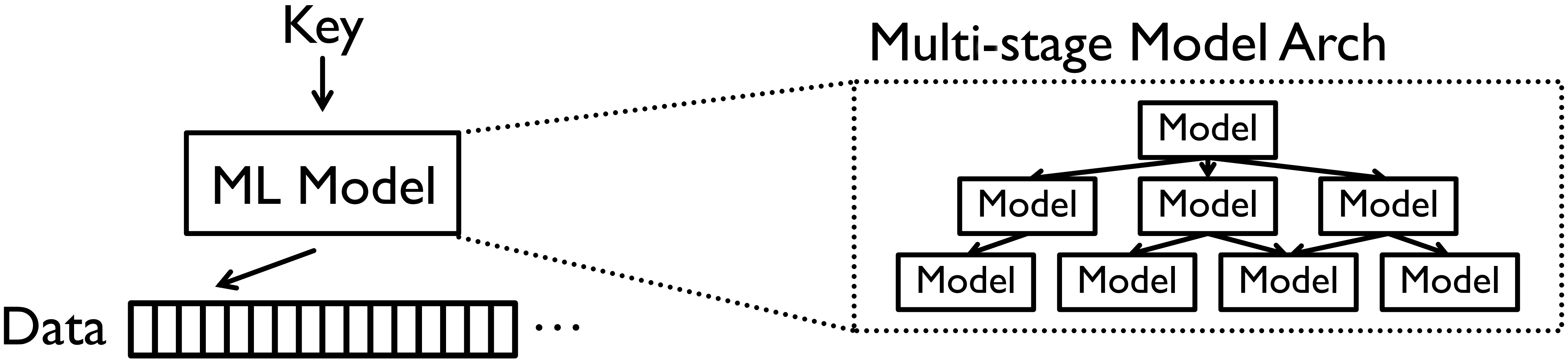
- **Multi-stage models learn indexes efficiently**



Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

- Multi-stage models learn indexes efficiently

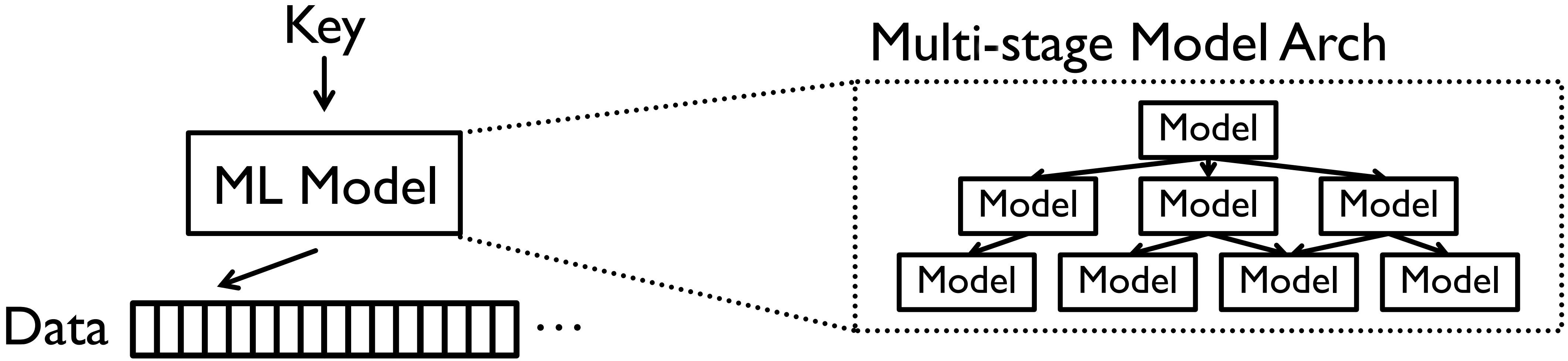


Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index

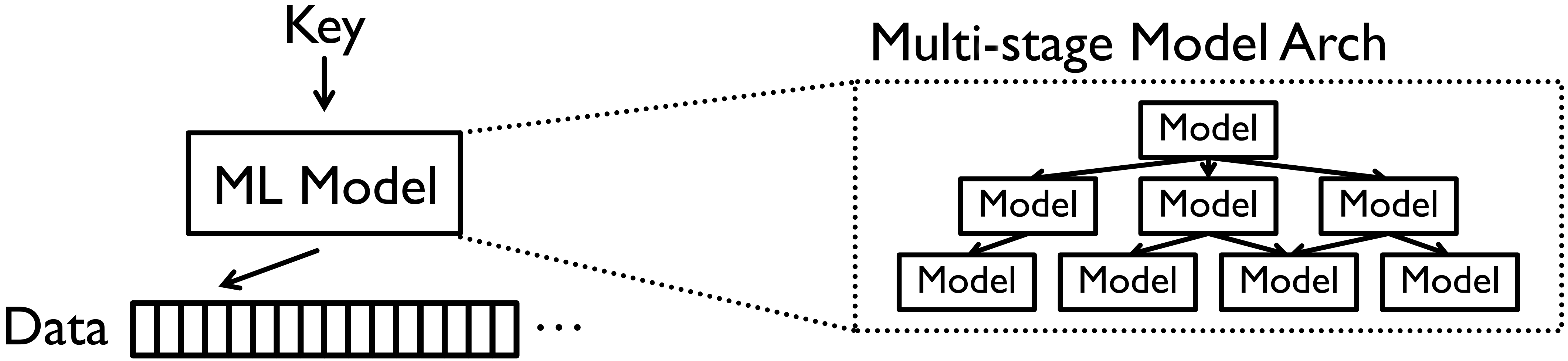
- Multi-stage models learn indexes efficiently

- Reduce 63% read latency and 99% memory usage



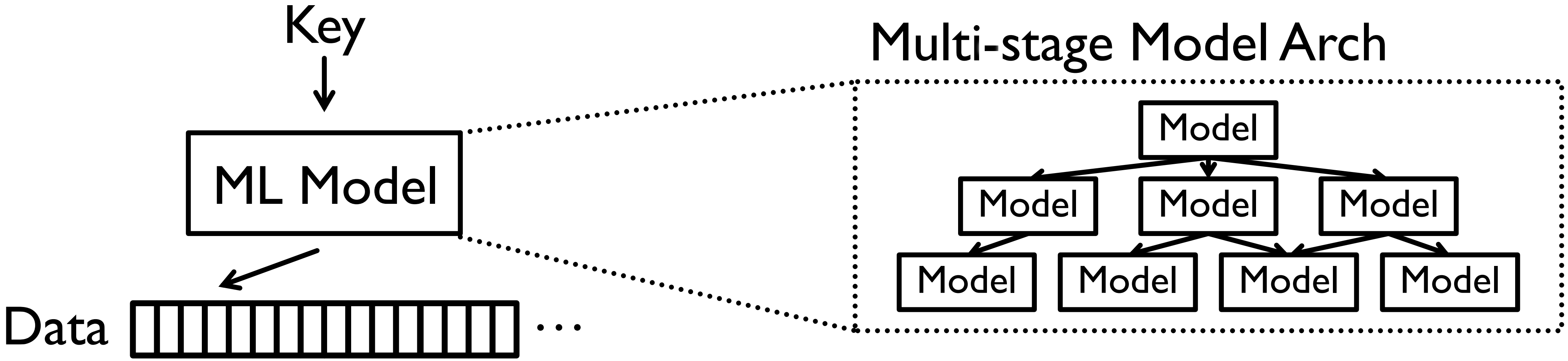
Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. SIGMOD '18

Background: the learned index



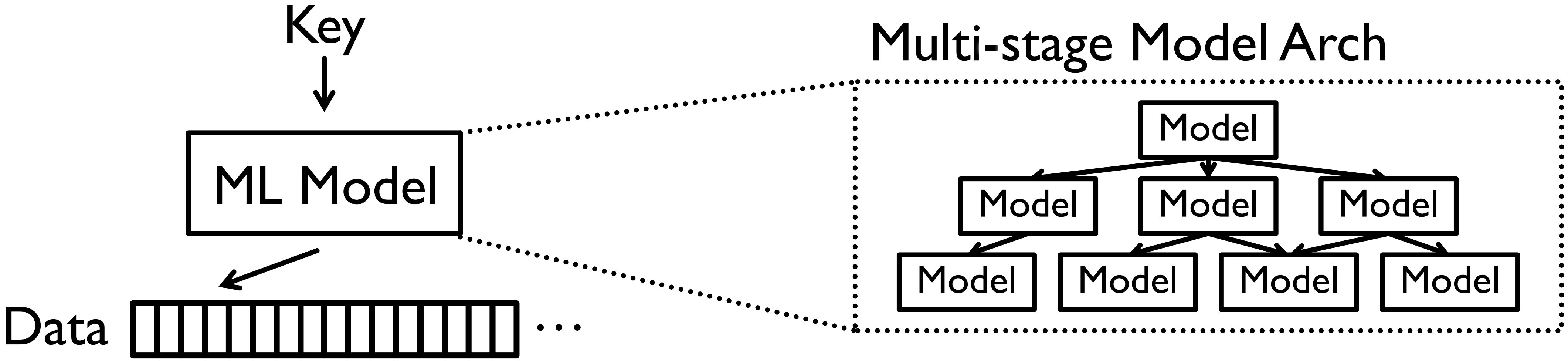
Background: the learned index

- **ISSUE 1 read-only, and non-trivial to support writes**
  - Takes several seconds to sort millions of records



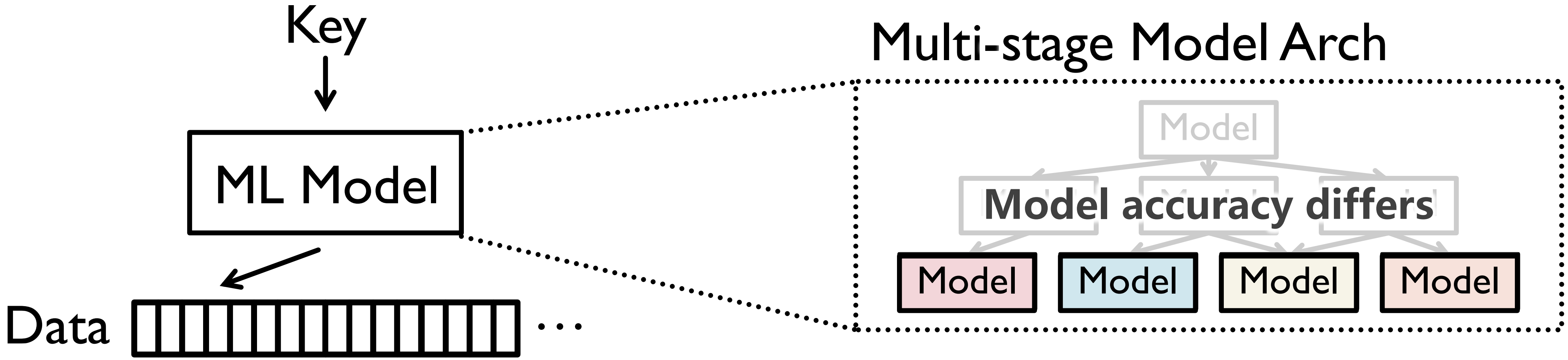
Background: the learned index

- **ISSUE 1 read-only, and non-trivial to support writes**
  - Takes several seconds to sort millions of records
- **ISSUE 2 performance degrades in certain workloads**
  - 23% worse than B-Tree in a specific access pattern



Background: the learned index

- ISSUE 1 **read-only, and non-trivial to support writes**
  - Takes several seconds to sort millions of records
- ISSUE 2 **performance degrades in certain workloads**
  - 23% worse than B-Tree in a specific access pattern



XIndex contribution

XIndex contribution

- How to efficiently support writes and concurrency?

XIndex contribution

- How to efficiently support writes and concurrency?

SOLUTION: buffer inserts and compact periodically

XIndex contribution

- **How to efficiently support writes and concurrency?**

SOLUTION: buffer inserts and compact periodically

Two-Phase Compaction for correctness and efficiency

XIndex contribution

- **How to efficiently support writes and concurrency?**

SOLUTION: buffer inserts and compact periodically

Two-Phase Compaction for correctness and efficiency

Fine-grained Synchronization for scalability

XIndex contribution

- How to efficiently support writes and concurrency?

SOLUTION: buffer inserts and compact periodically

Two-Phase Compaction for correctness and efficiency

Fine-grained Synchronization for scalability

- How to stay performant in dynamic workloads?

XIndex contribution

- How to efficiently support writes and concurrency?

SOLUTION: buffer inserts and compact periodically

Two-Phase Compaction for correctness and efficiency

Fine-grained Synchronization for scalability

- How to stay performant in dynamic workloads?

SOLUTION: adjust the structure at runtime

Heuristics for small model errors and buffer sizes

XIndex contribution

- How to efficiently support writes and concurrency?

SOLUTION: buffer inserts and compact periodically

Two-Phase Compaction for correctness and efficiency

Fine-grained Synchronization for scalability

- How to stay performant in dynamic workloads?

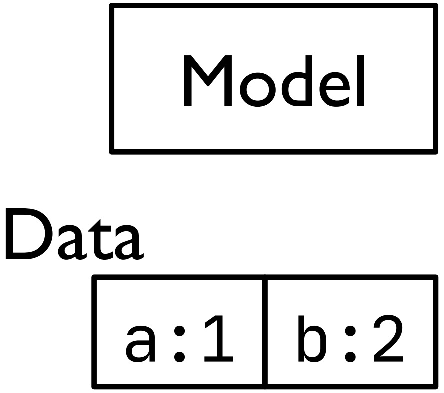
SOLUTION: adjust the structure at runtime

Heuristics for small model errors and buffer sizes

Up to 4.4× better perf than the state-of-the-arts

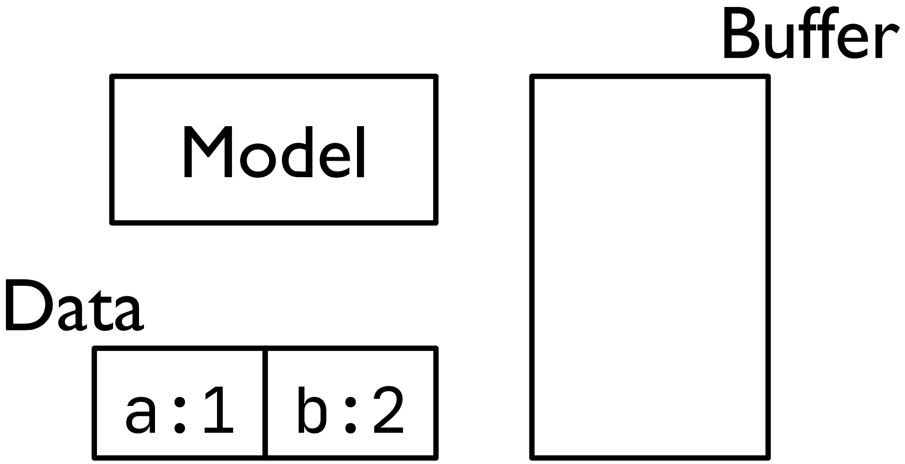
Handling writes: strawman solution

Handling writes: strawman solution



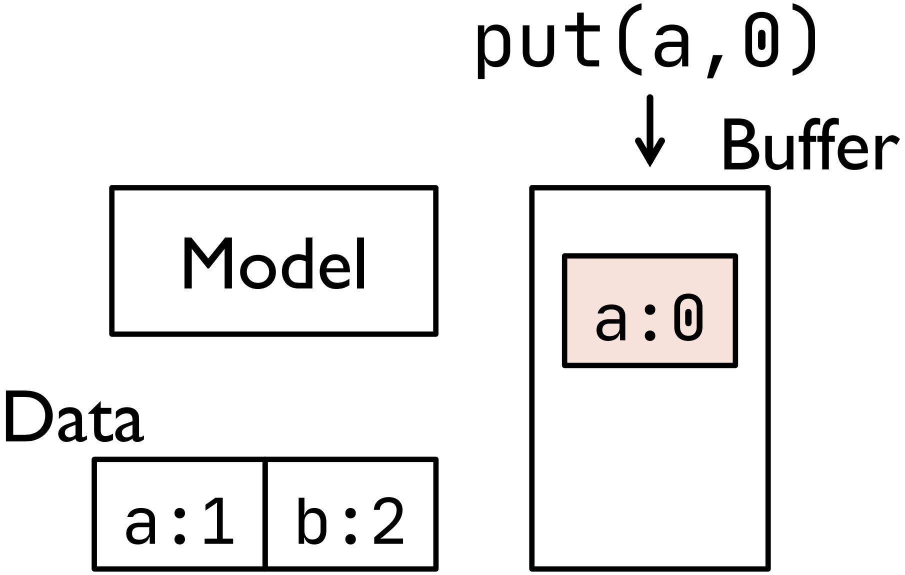
Handling writes: strawman solution

1. **Buffers all writes separately (e.g., in a B-Tree)**



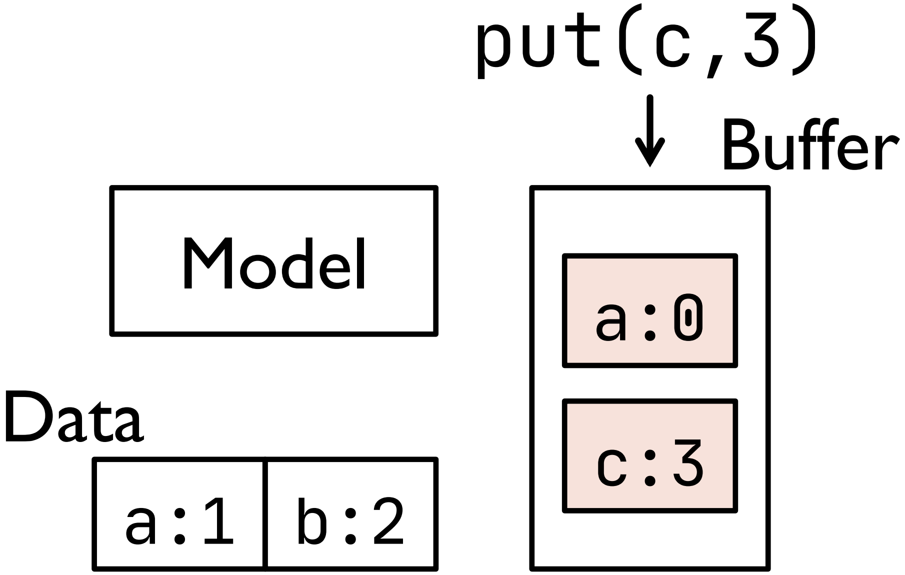
Handling writes: strawman solution

1. Buffers all writes separately (e.g., in a B-Tree)



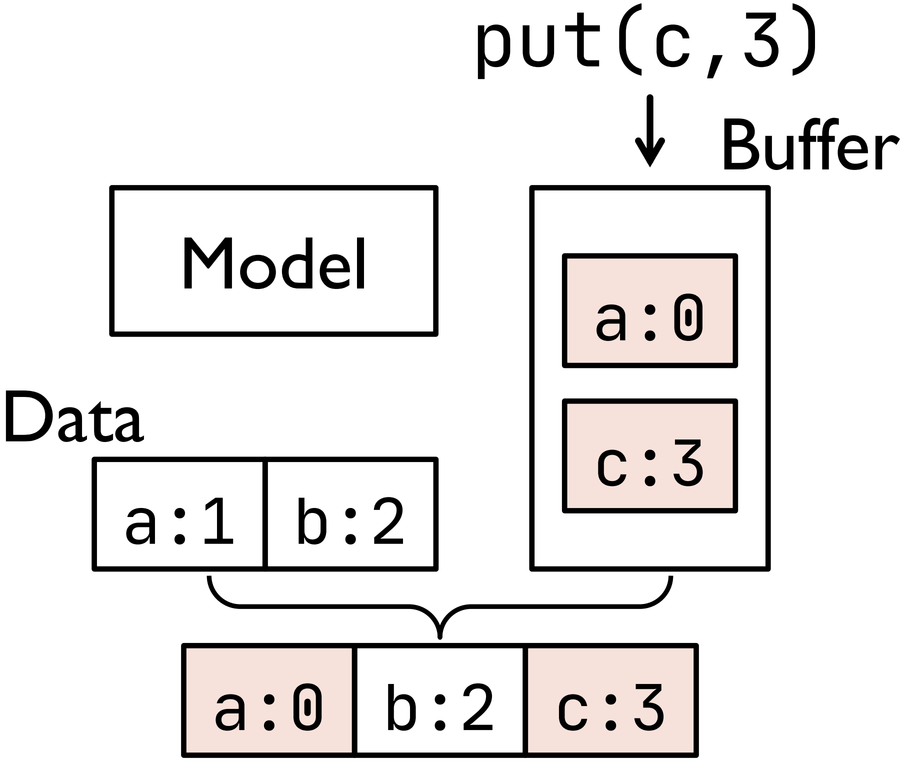
Handling writes: strawman solution

1. Buffers all writes separately (e.g., in a B-Tree)



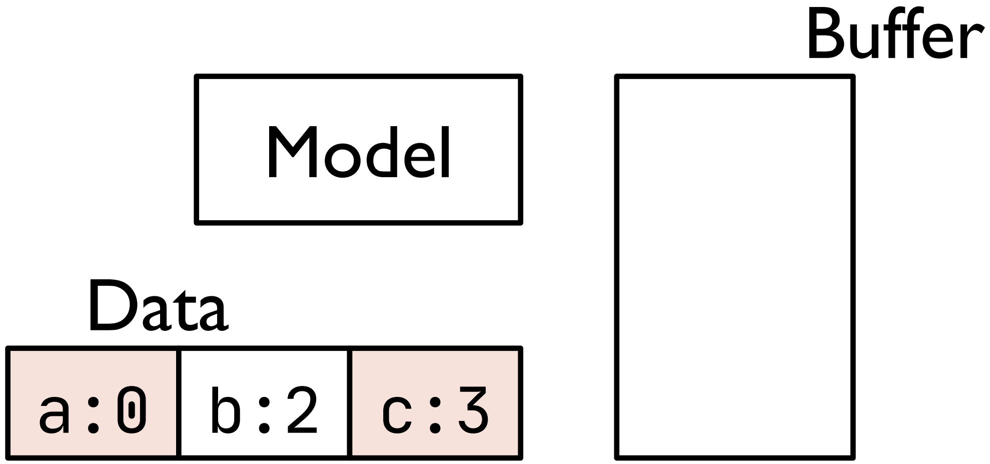
Handling writes: strawman solution

1. Buffers all writes separately (e.g., in a B-Tree)
2. Periodically compact the buffer



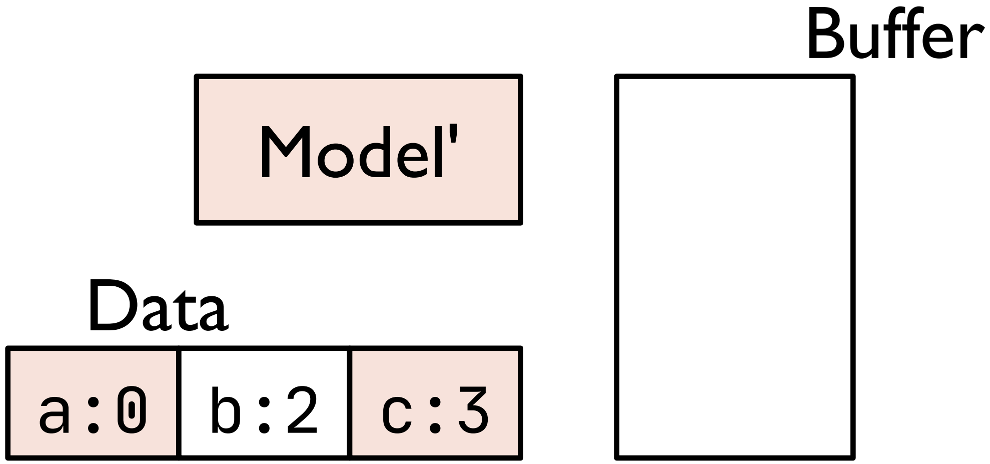
Handling writes: strawman solution

1. Buffers all writes separately (e.g., in a B-Tree)
2. Periodically compact the buffer



Handling writes: strawman solution

1. Buffers all writes separately (e.g., in a B-Tree)
2. Periodically compact the buffer and retrain the model



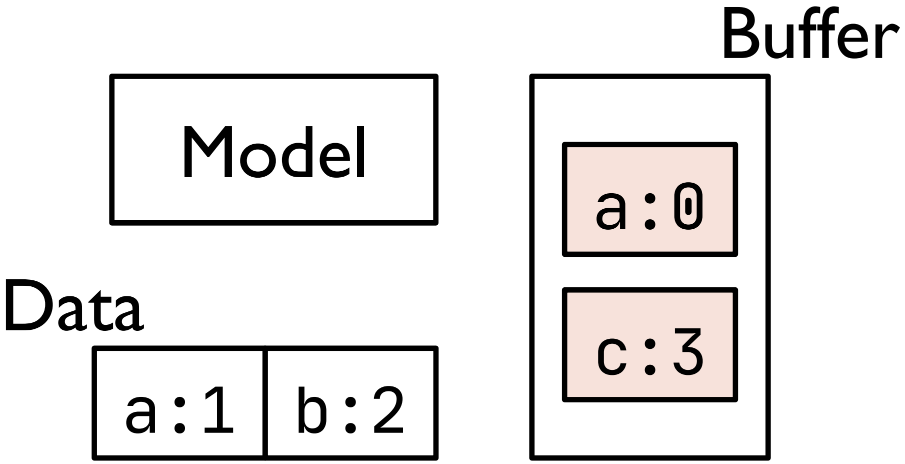
Handling writes: strawman solution

Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup

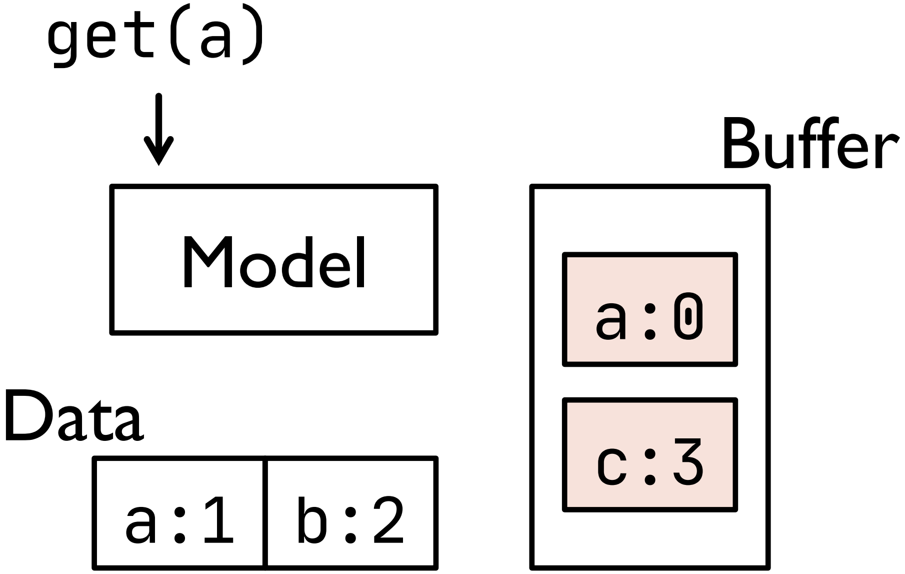
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup



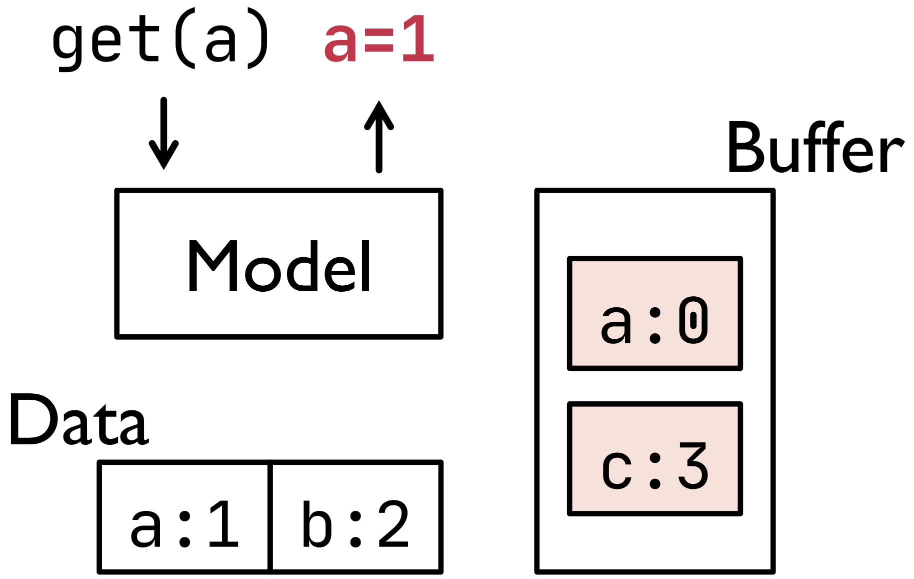
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup



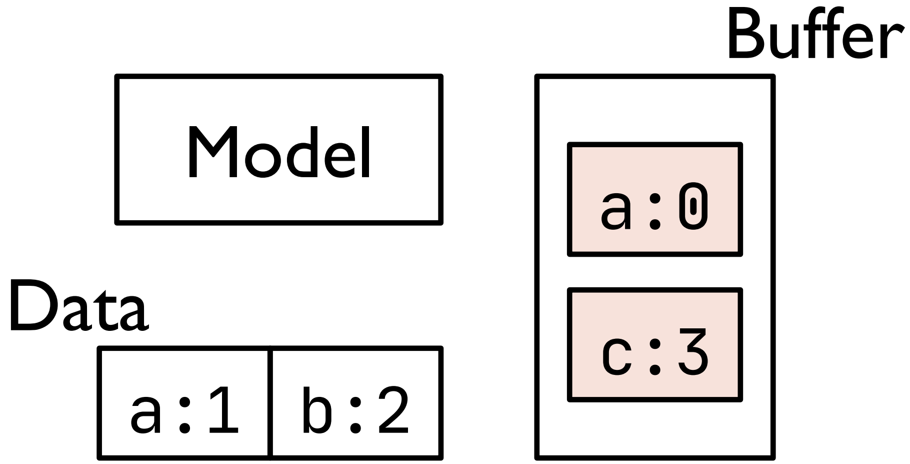
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup



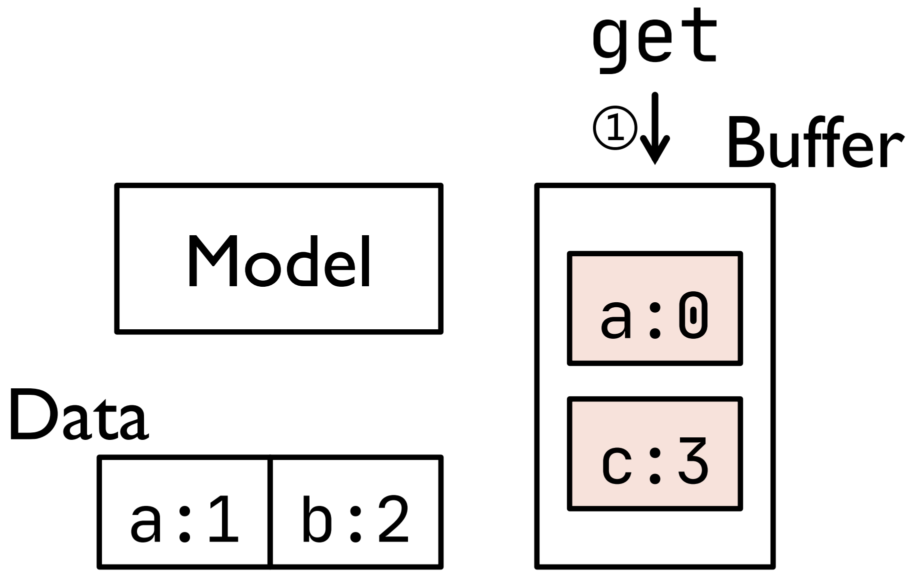
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup



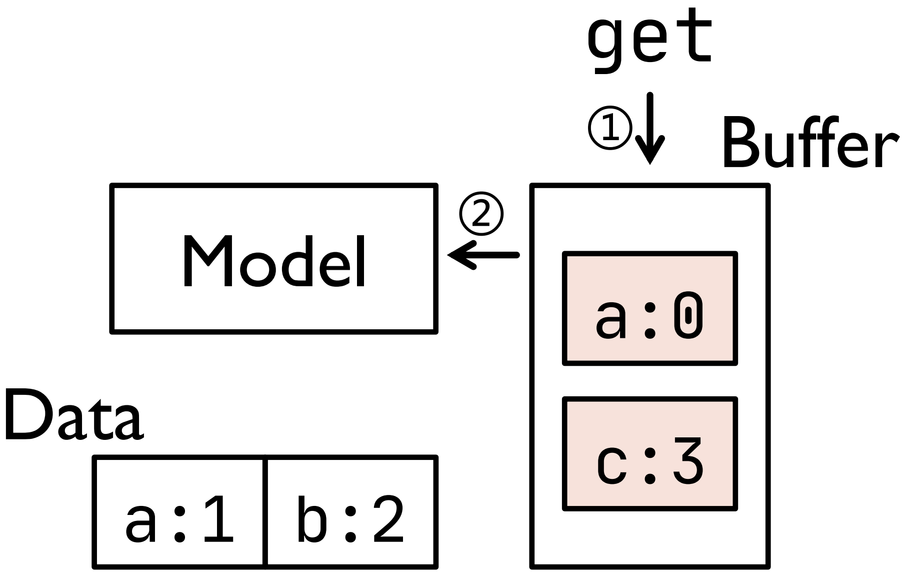
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup



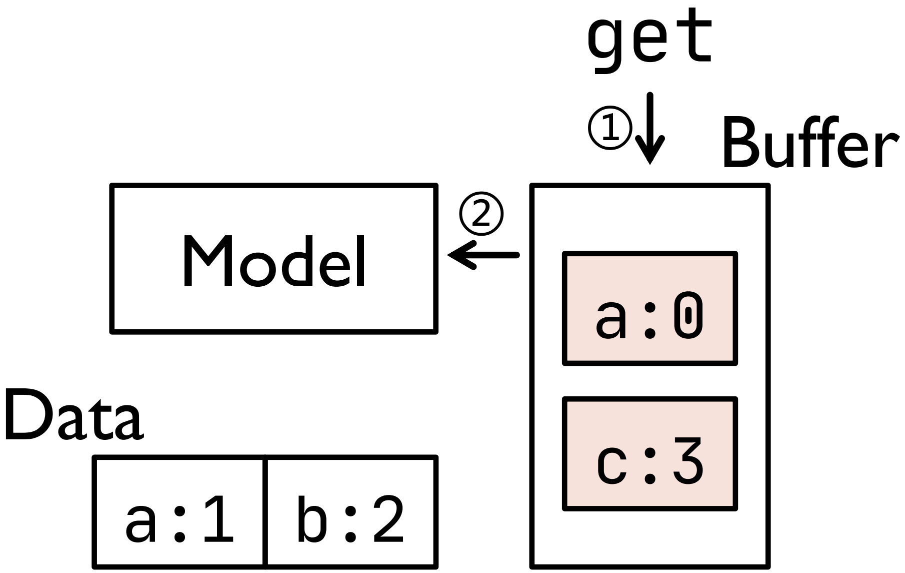
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup



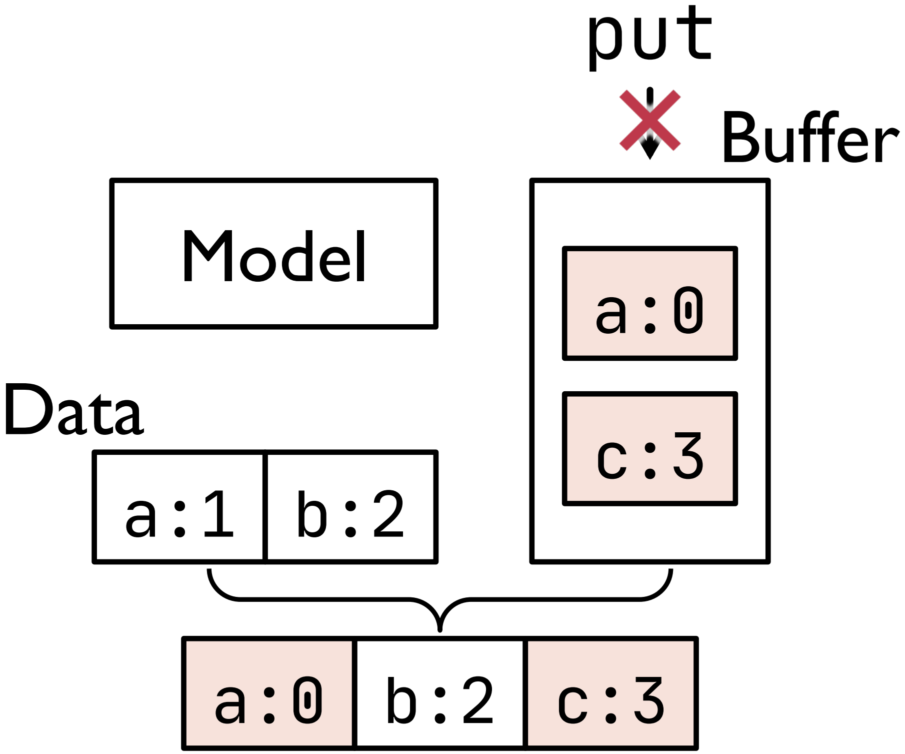
Handling writes: strawman solution

- **ISSUE 1: Reads get slower, due to buffer lookup**
  - More than 100% slow down



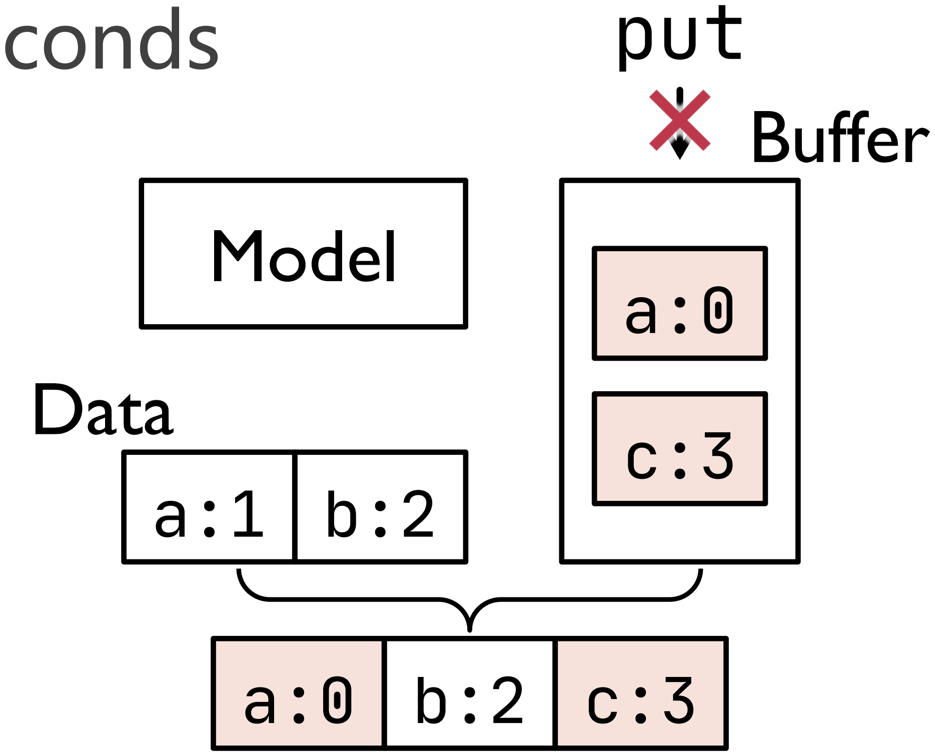
Handling writes: strawman solution

- **ISSUE 1**: Reads get slower, due to buffer lookup
- **ISSUE 2**: Compaction blocks writes to avoid races



Handling writes: strawman solution

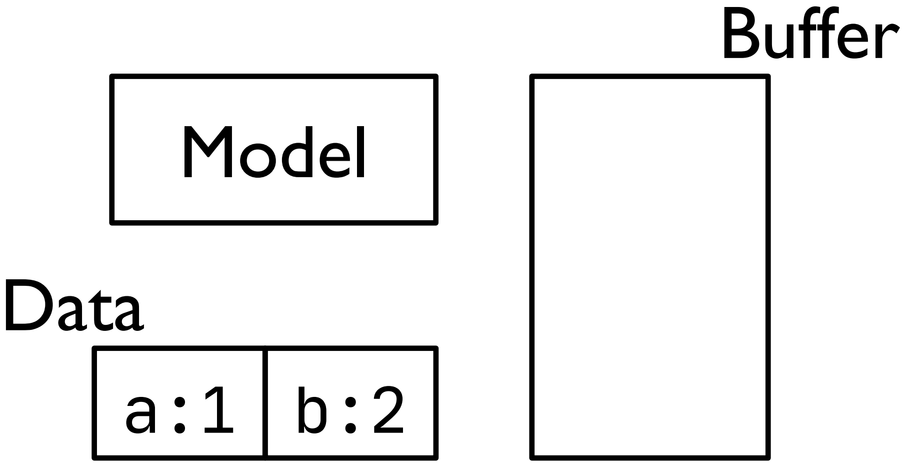
- **ISSUE 1**: Reads get slower, due to buffer lookup
- **ISSUE 2**: Compaction blocks writes to avoid races
  - – Up to 30+ seconds



Handling writes: improving strawman

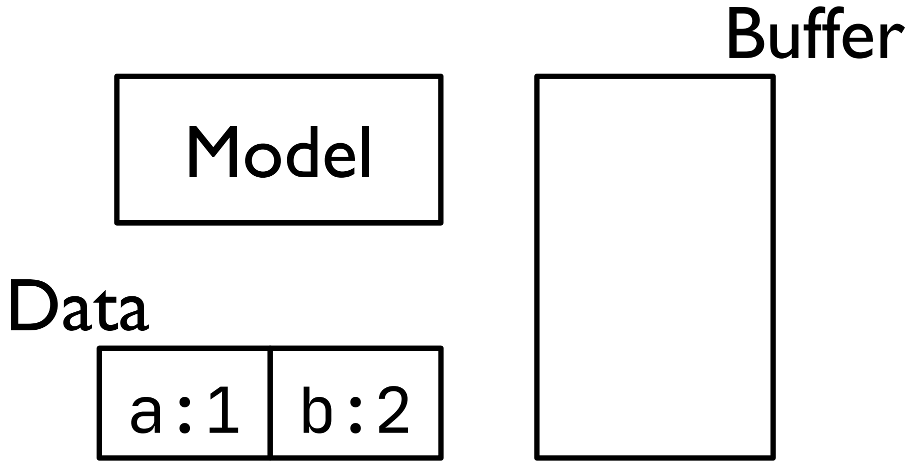
Handling writes: improving strawman

1. Avoid buffer lookups for reads



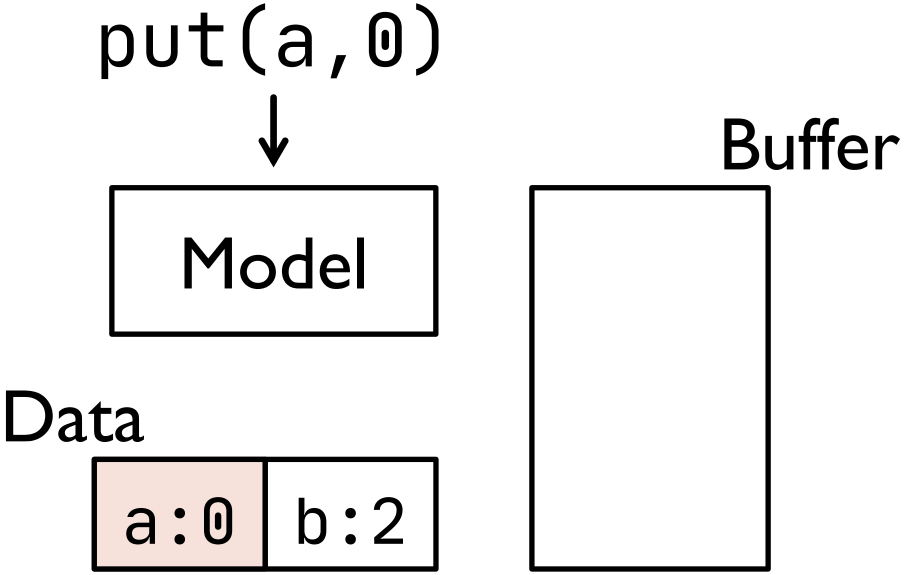
Handling writes: improving strawman

1. **Avoid buffer lookups for reads**
   - By performing updates in-place



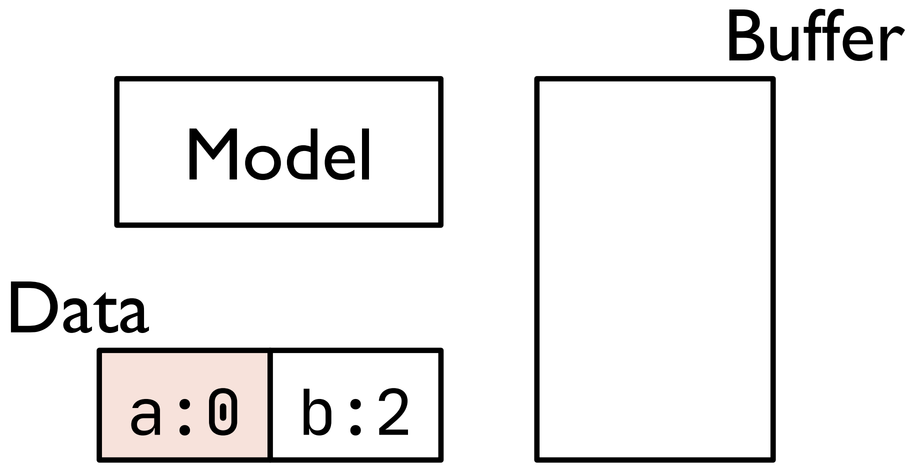
Handling writes: improving strawman

1. **Avoid buffer lookups for reads**
   - By performing updates in-place



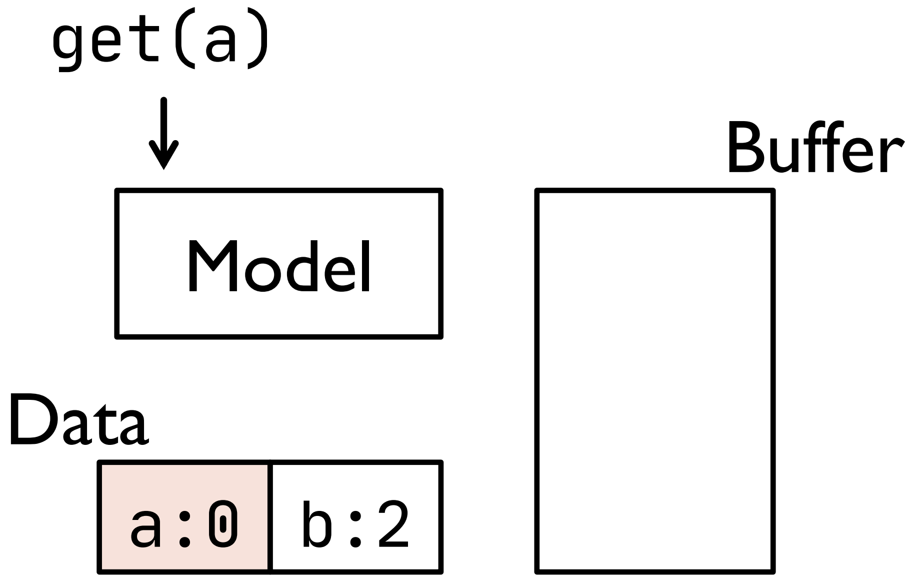
Handling writes: improving strawman

1. **Avoid buffer lookups for reads**
   - By performing updates in-place



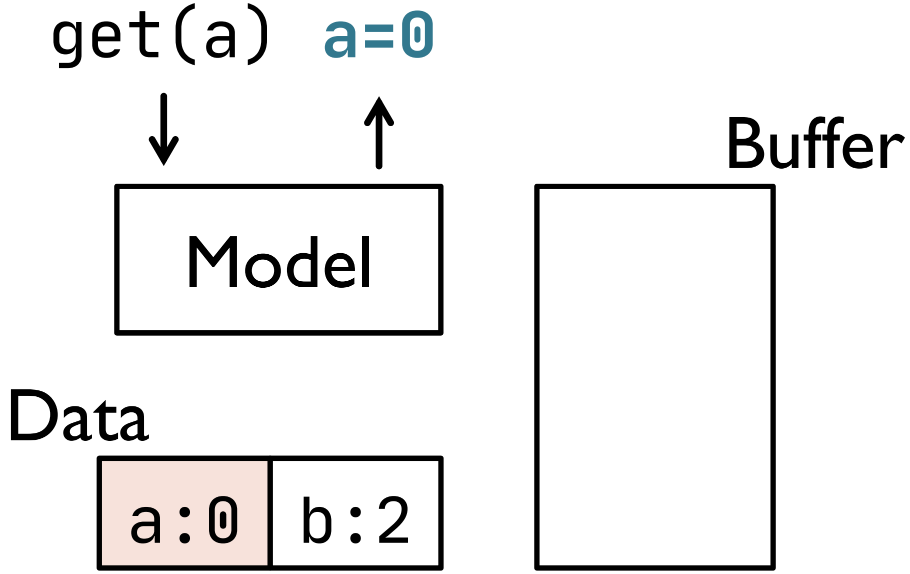
Handling writes: improving strawman

1. **Avoid buffer lookups for reads**
   - By performing updates in-place



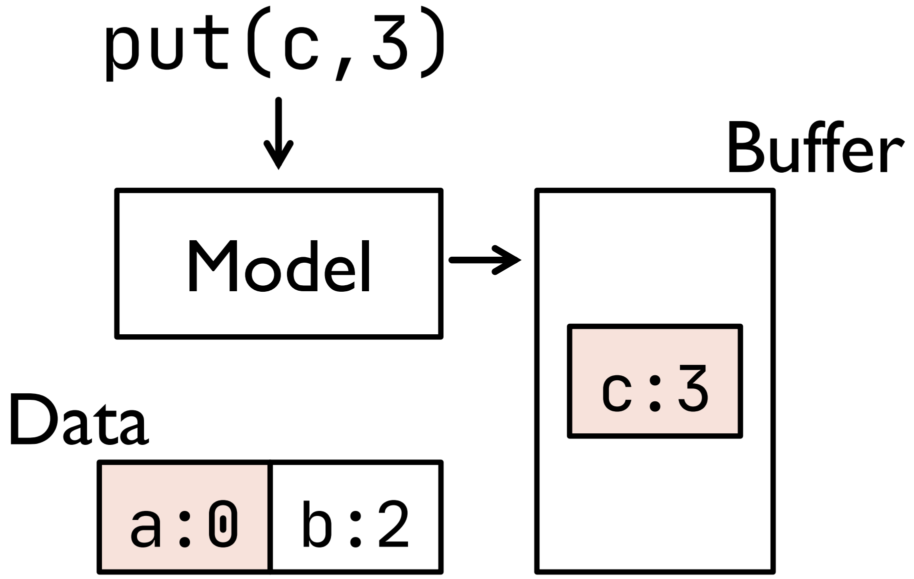
Handling writes: improving strawman

1. **Avoid buffer lookups for reads**
   - By performing updates in-place



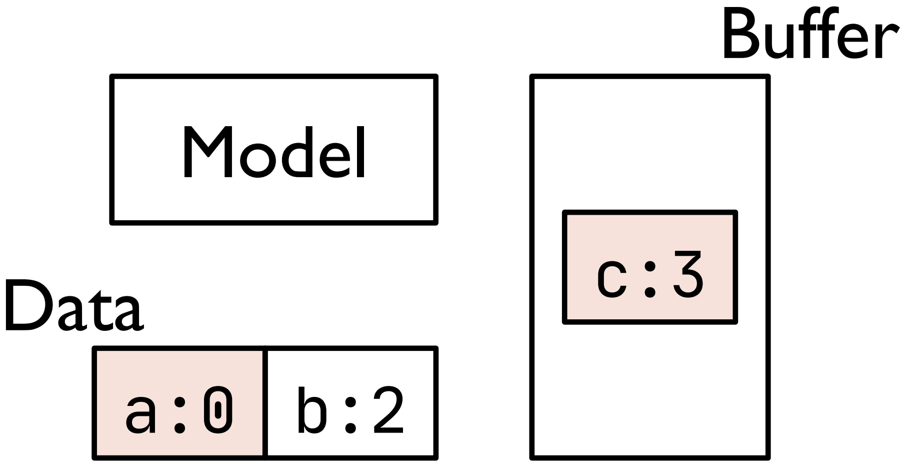
Handling writes: improving strawman

1. **Avoid buffer lookups for reads**
   - By performing updates in-place, and buffering only insertions



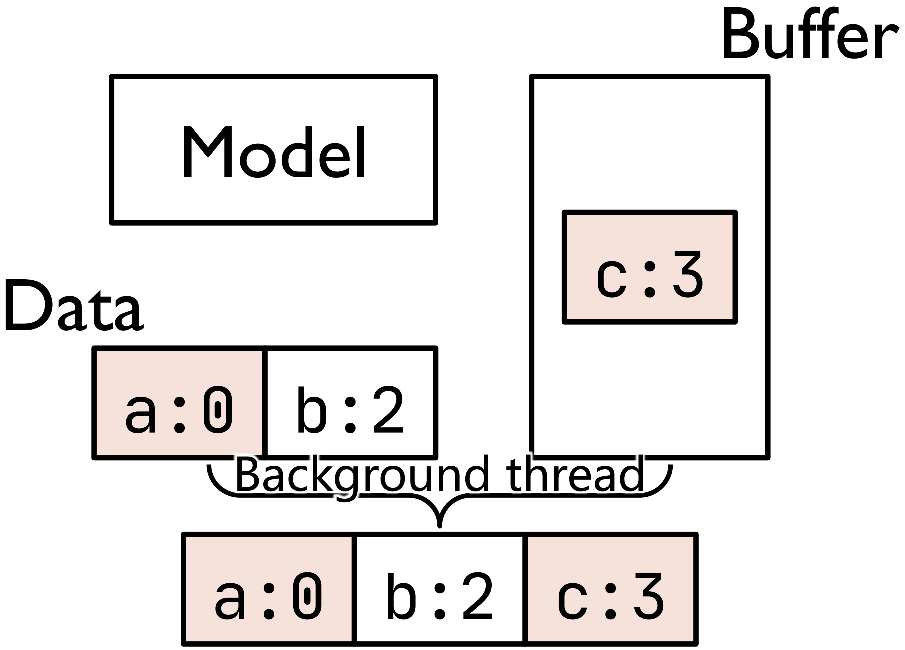
Handling writes: improving strawman

2. Avoid blocking writes



Handling writes: improving strawman

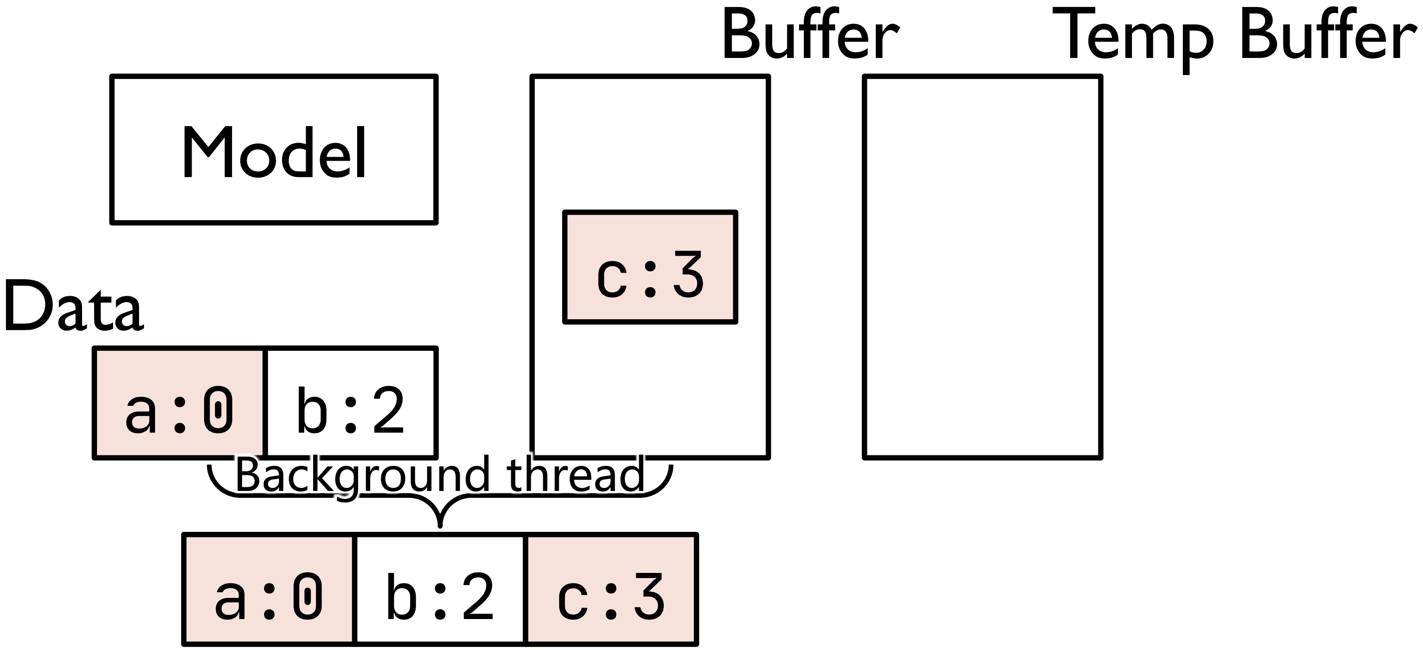
2. **Avoid blocking writes**
   - By compacting asynchronously



Handling writes: improving strawman

2. Avoid blocking writes

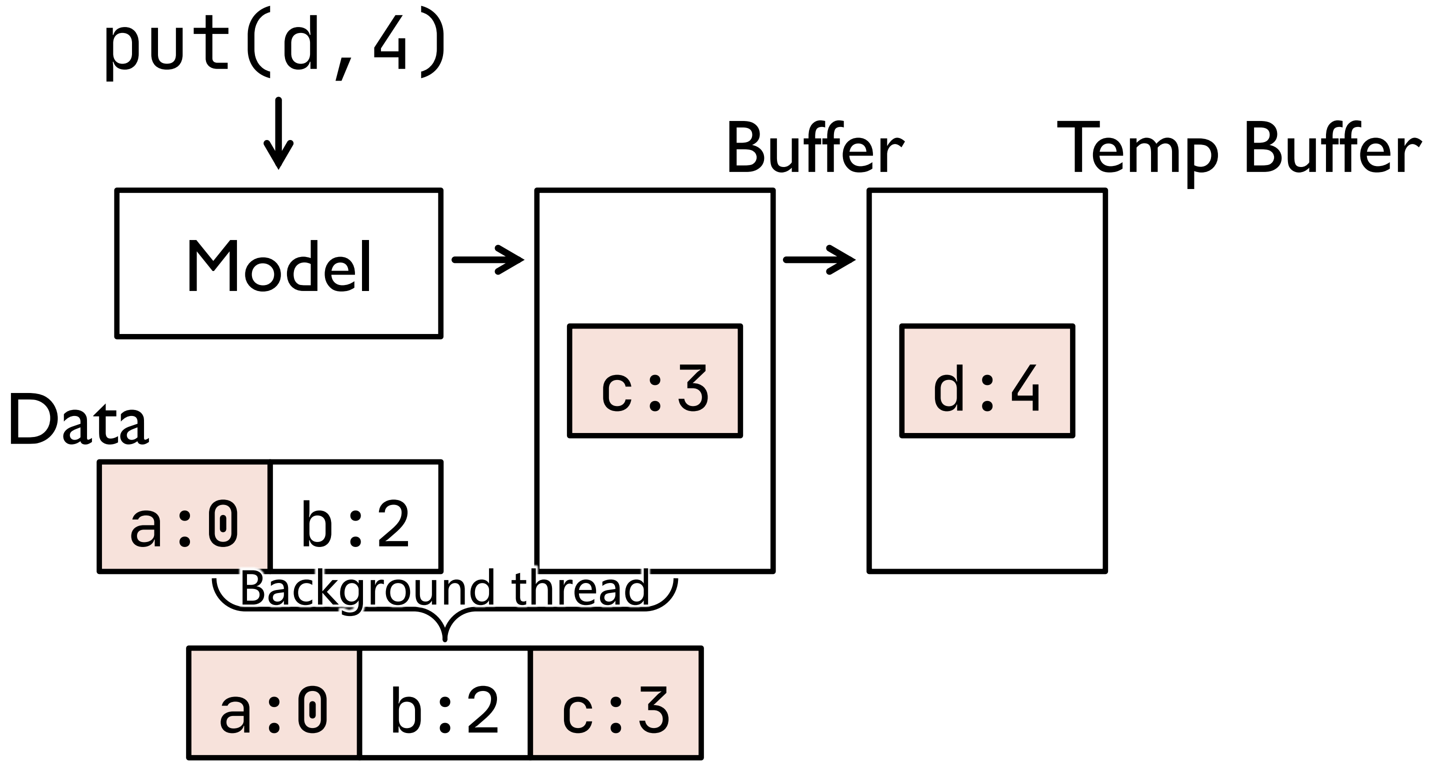
- By compacting asynchronously, and using a temporary buffer



Handling writes: improving strawman

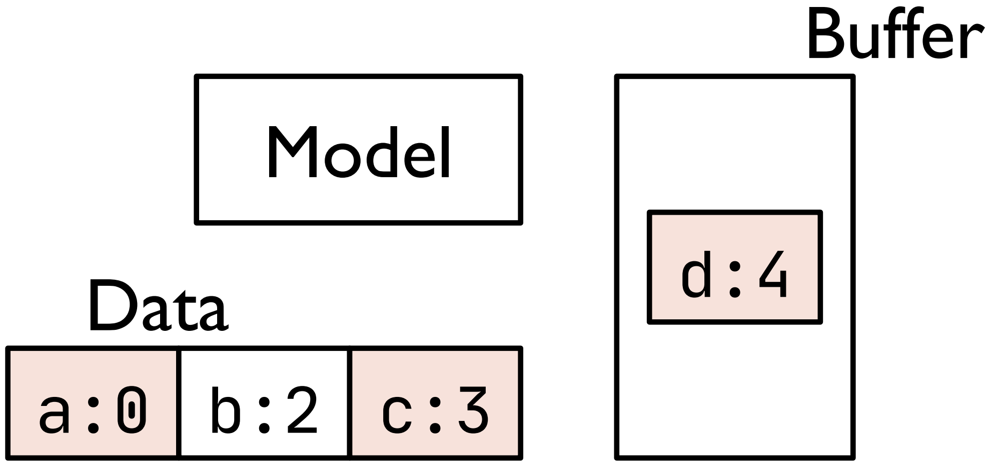
2. Avoid blocking writes

- By compacting asynchronously, and using a temporary buffer



Handling writes: improving strawman

2. **Avoid blocking writes**
   - By compacting asynchronously, and using a temporary buffer



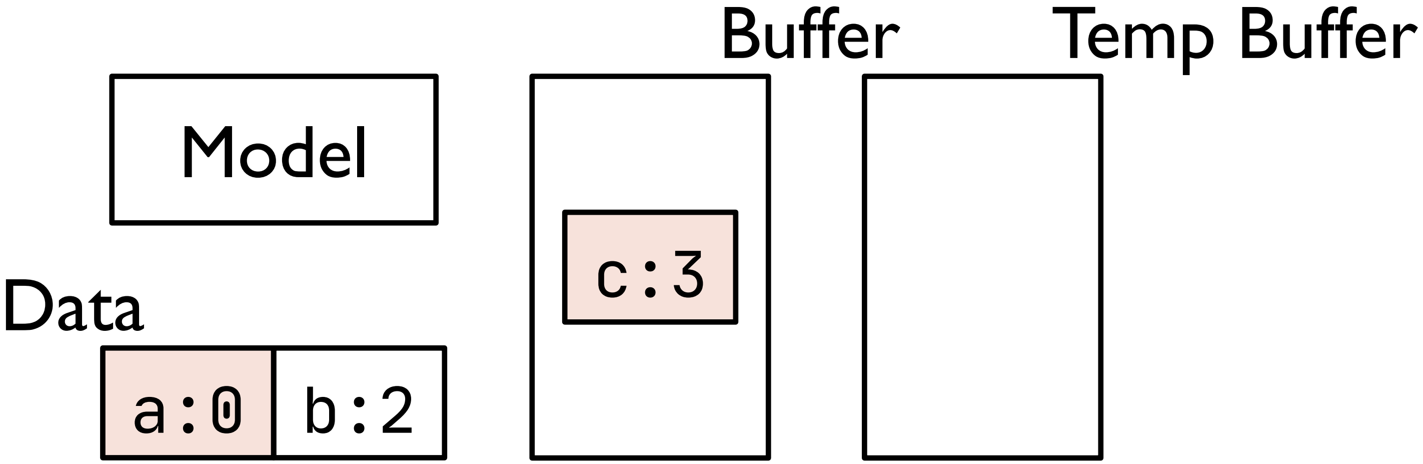
Handling writes: improving strawman

Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!

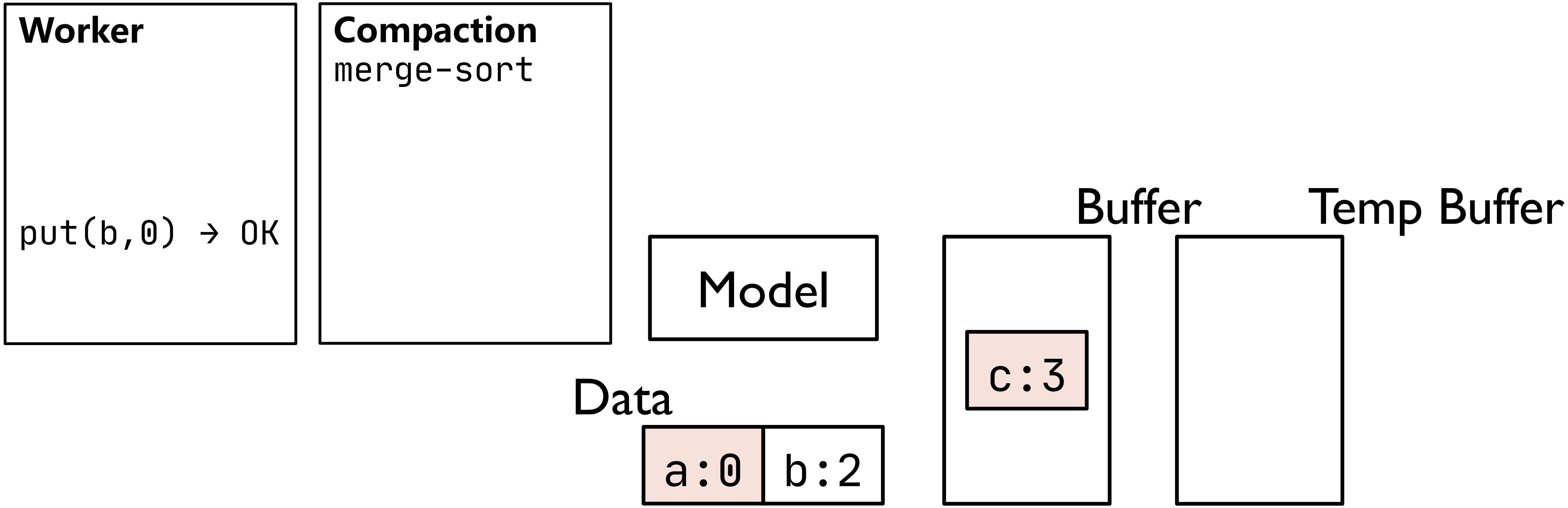
Handling writes: improving strawman

- CONSISTENCY ISSUE: Updates are lost!



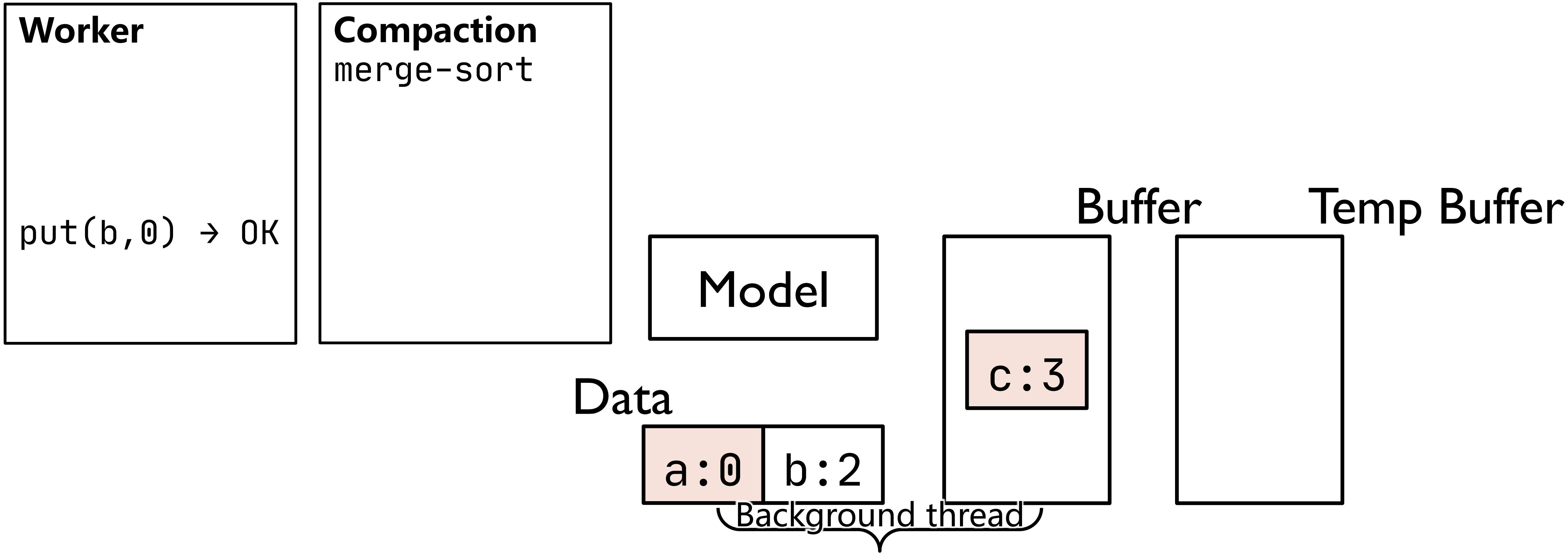
Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!



Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!



Handling writes: improving strawman

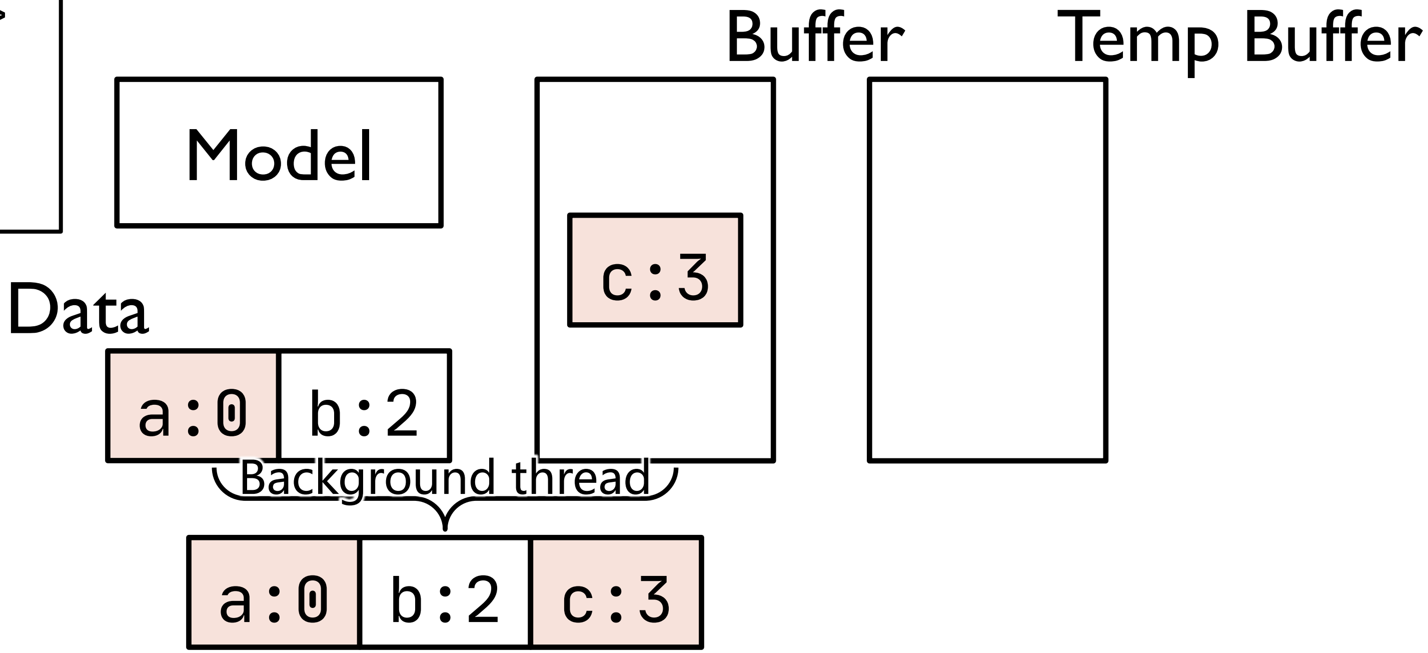
- **CONSISTENCY ISSUE**: Updates are lost!

Worker

```
put(b,0) → OK
```

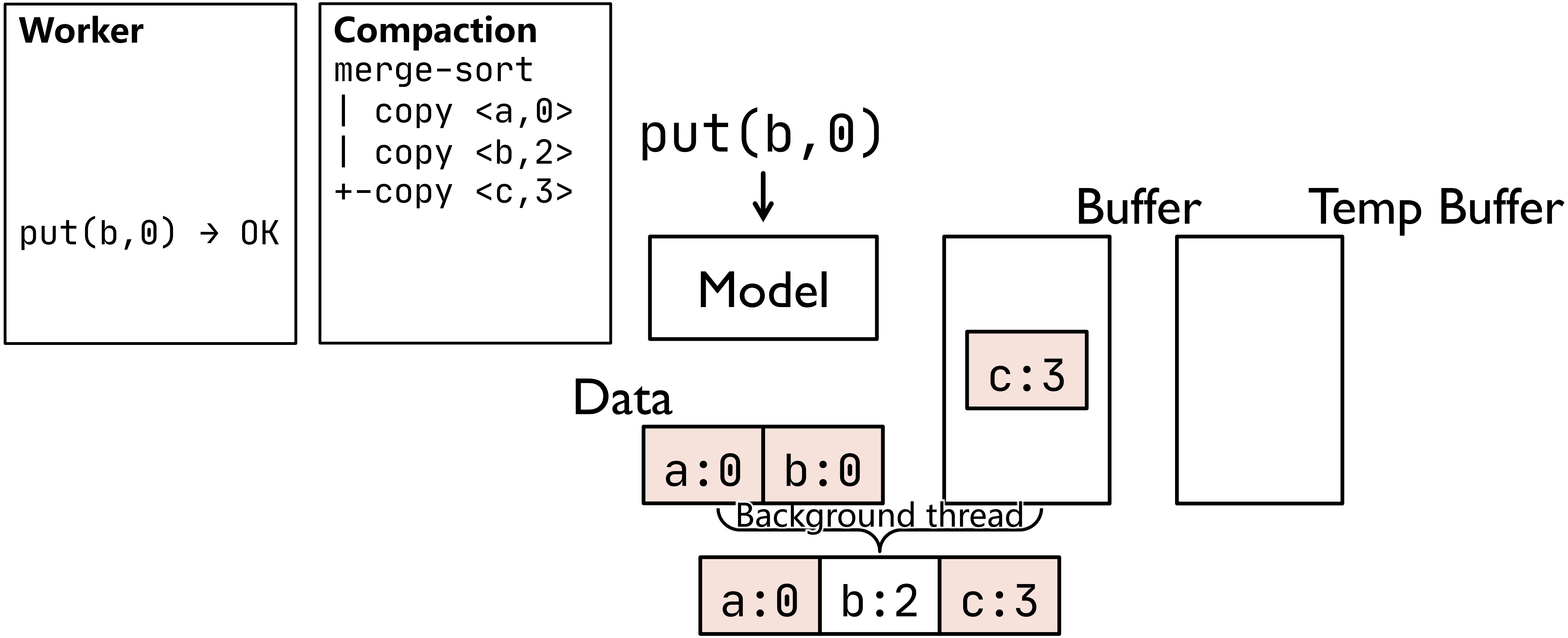
Compaction

```
merge-sort
| copy <a,0>
| copy <b,2>
+-copy <c,3>
```



Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!



Worker

```
put(b,0) → OK
```

Compaction

```
merge-sort
| copy <a,0>
| copy <b,2>
+-copy <c,3>
```

Handling writes: improving strawman

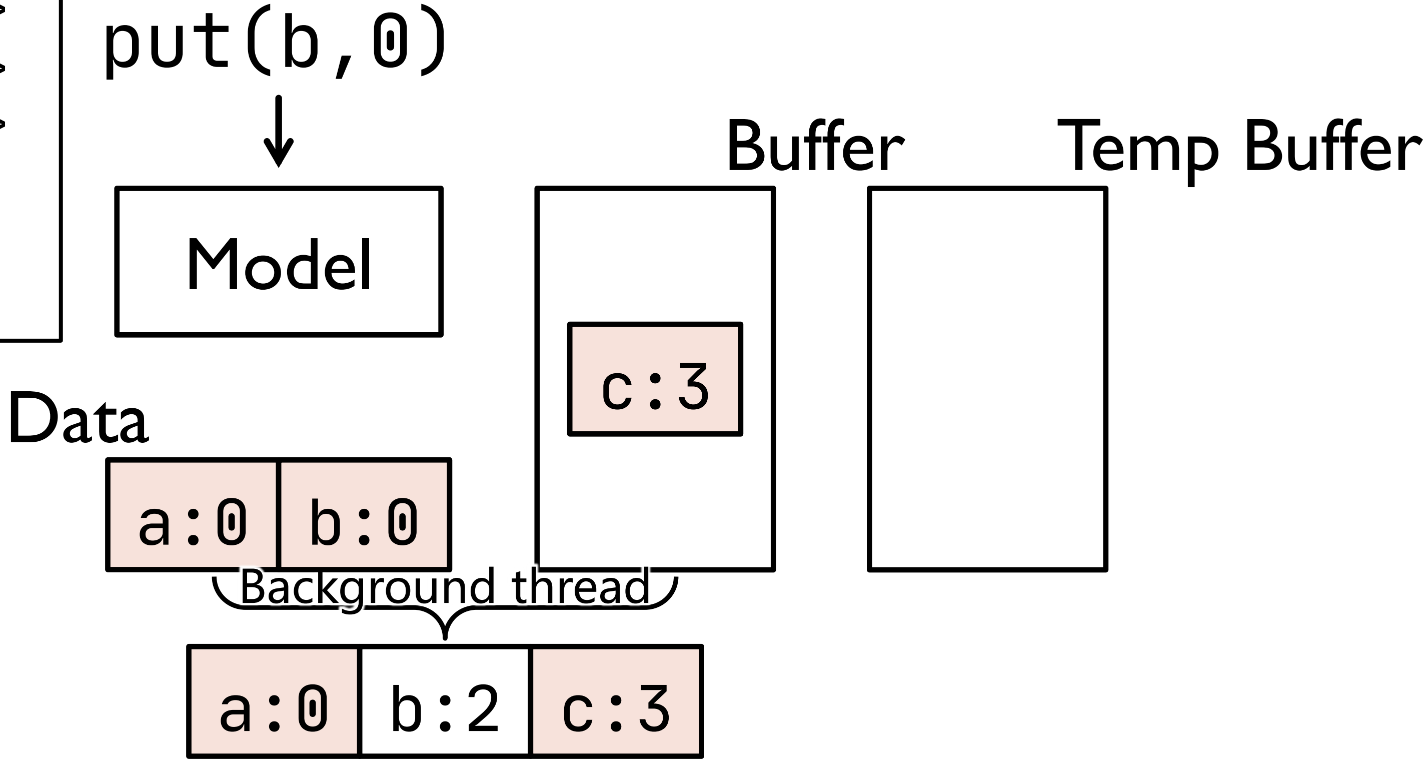
- **CONSISTENCY ISSUE**: Updates are lost!

Worker

```
put(b,0) → OK
```

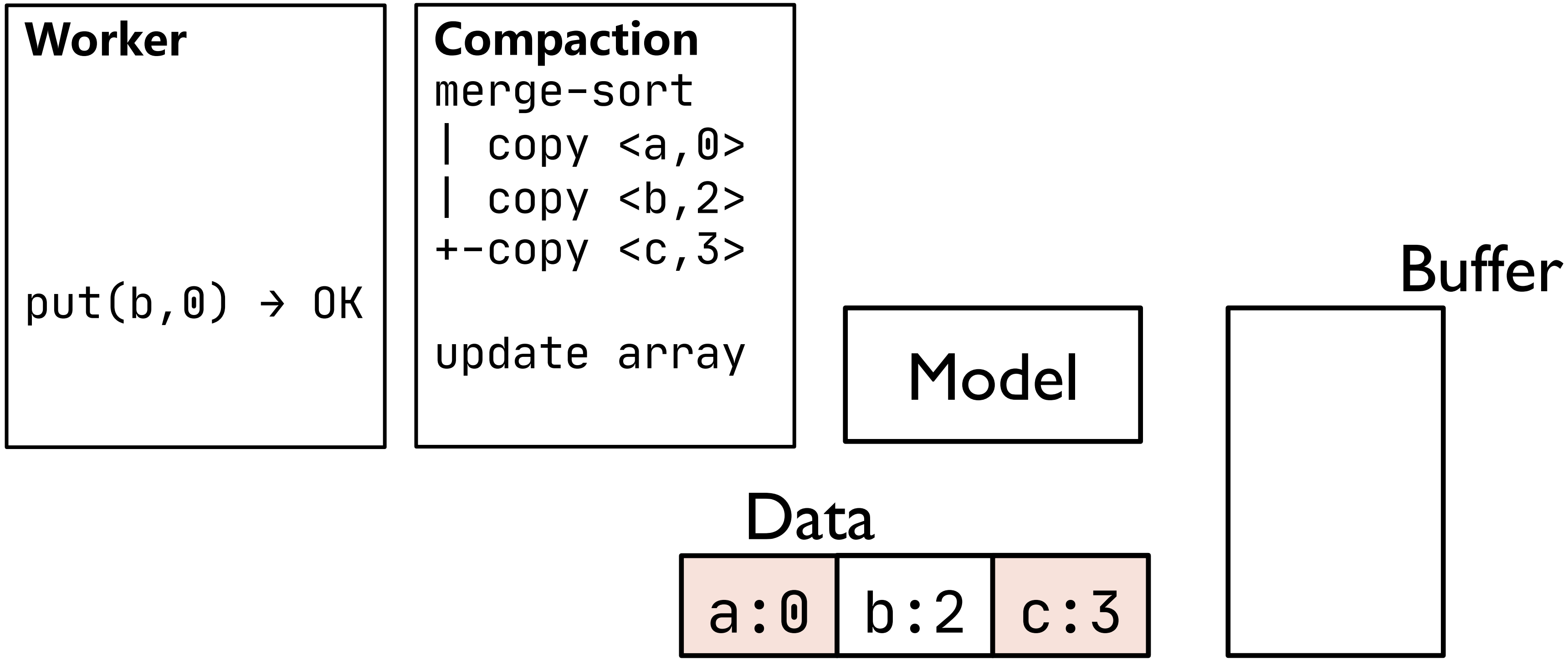
Compaction

```
merge-sort
| copy <a,0>
| copy <b,2>
+-copy <c,3>
```



Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!



Worker

```
put(b,0) → OK
```

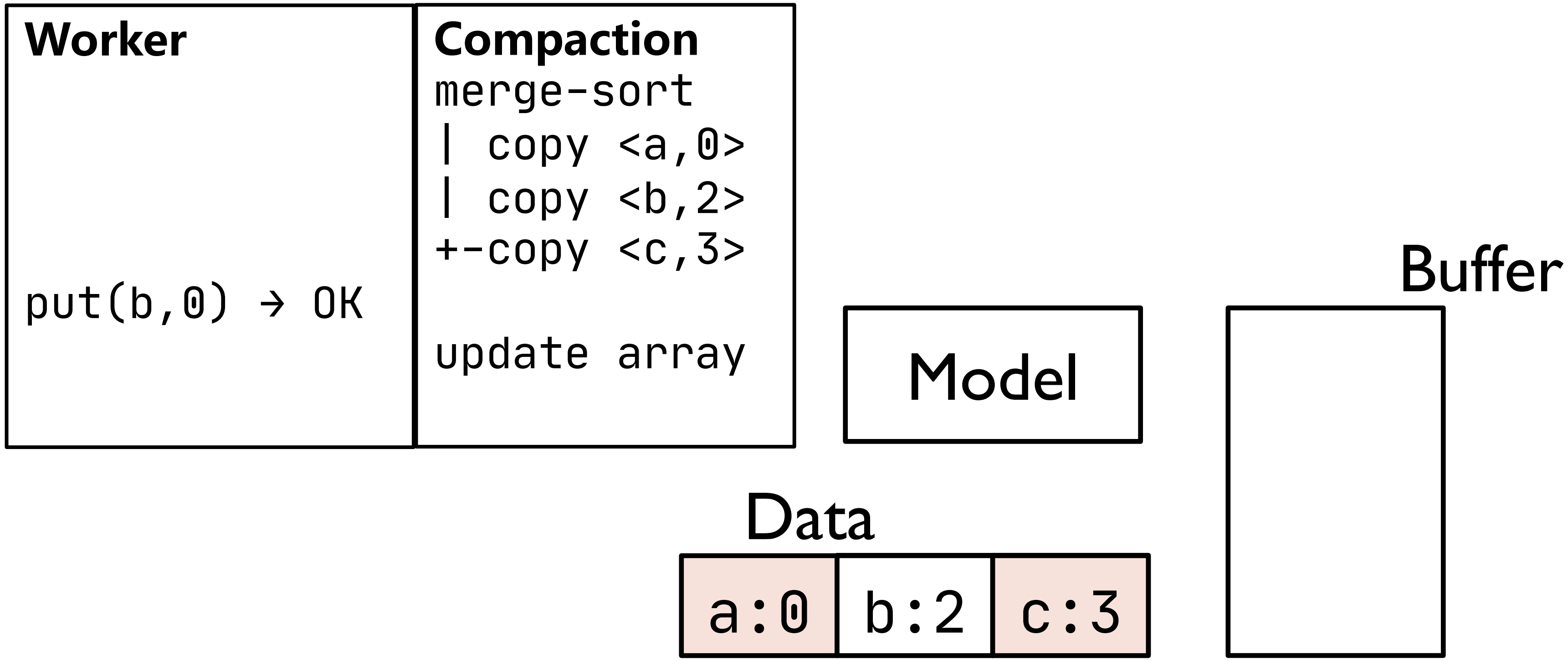
Compaction

```
merge-sort
| copy <a,0>
| copy <b,2>
+-copy <c,3>

update array
```

a:0	b:2	c:3

Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!



Worker

```
put(b,0) → OK
```

Compaction

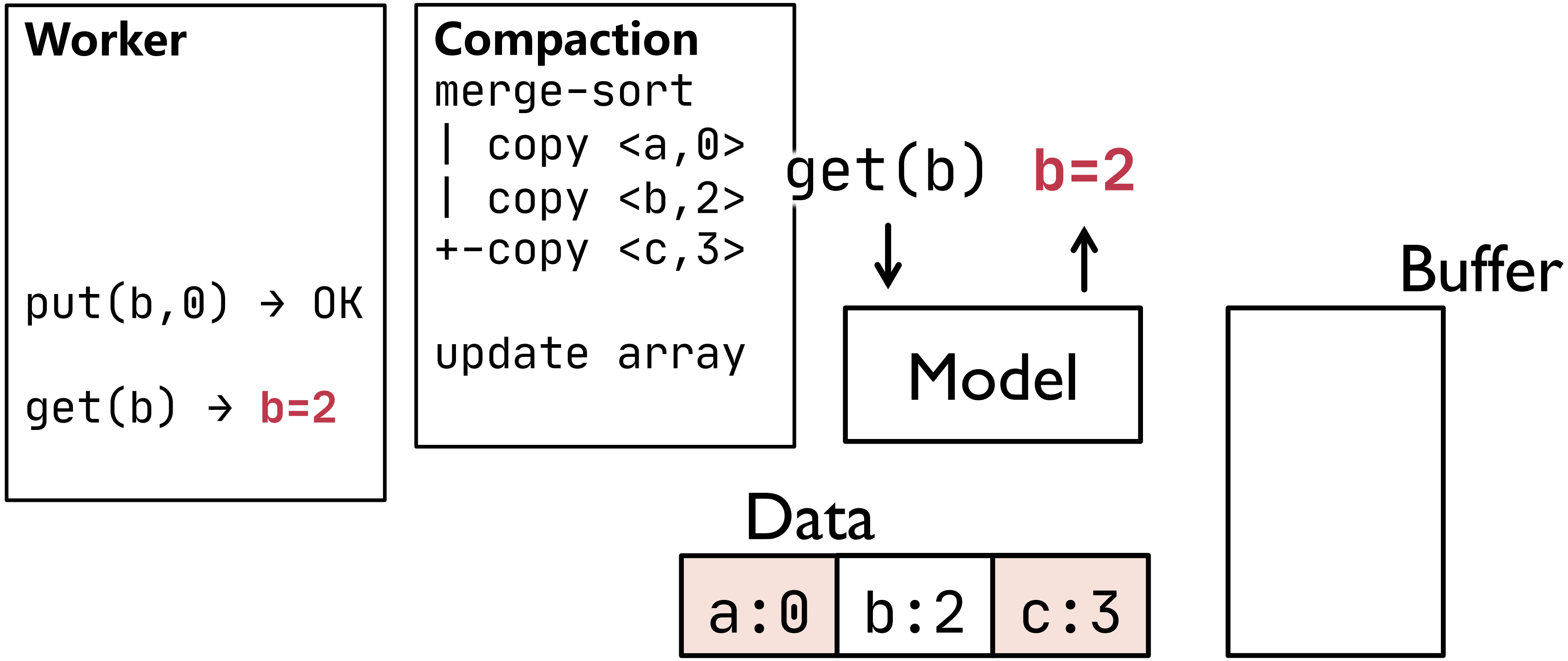
```
merge-sort
| copy <a,0>
| copy <b,2>
+-copy <c,3>

update array
```

a:0	b:2	c:3

keeps stale value (b=2)

Handling writes: improving strawman

- **CONSISTENCY ISSUE**: Updates are lost!



Worker

```
put(b,0) → OK

get(b) → b=2
```

Compaction

```
merge-sort
| copy <a,0>
| copy <b,2>
+-copy <c,3>

update array
```

keeps stale value (b=2)

Handling writes: the challenge

- How to efficiently and correctly handle writes?

Handling writes: the challenge

- How to **efficiently** and **correctly** handle writes?

Cannot slow down reads ✓

Handling writes: the challenge

- How to efficiently and correctly handle writes?

Cannot slow down reads ✓

Cannot block writes ✓

Handling writes: the challenge

- How to efficiently and correctly handle writes?

Cannot slow down reads ✓

Cannot block writes ✓

Must retain all updates ✗

Handling writes: the challenge

- How to efficiently and correctly handle writes?

Cannot slow down reads ✓

Cannot block writes ✓

Must retain all updates ✗

⇩

Two-Phase Compaction

Handling writes: Two-Phase Compaction

Handling writes: Two-Phase Compaction

- **OBSERVATION: duplicate records cause inconsistency**

Handling writes: Two-Phase Compaction

- **OBSERVATION: duplicate records cause inconsistency**

- **IDEA: not to create duplicates during compaction**

Handling writes: Two-Phase Compaction

- OBSERVATION: duplicate records cause inconsistency
- IDEA: not to create duplicates during compaction
- METHOD: 2-Phase Compaction — merge, then copy

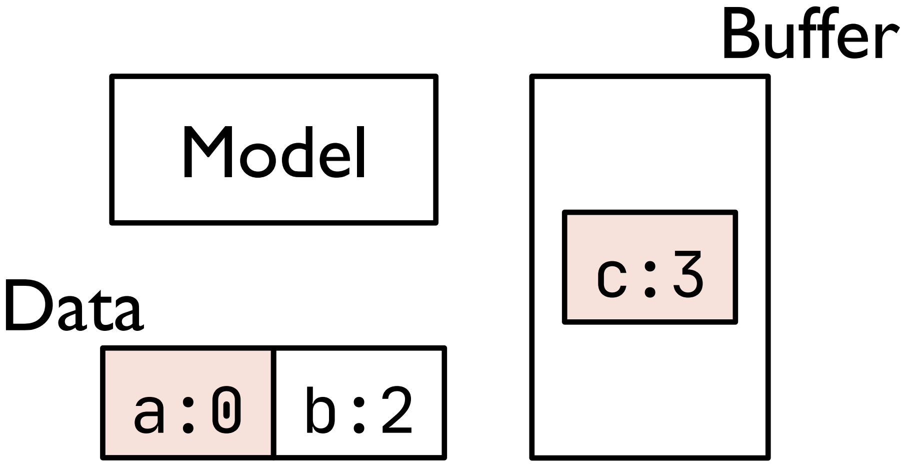
Handling writes: Two-Phase Compaction

- **OBSERVATION: duplicate records cause inconsistency**

- **IDEA: not to create duplicates during compaction**

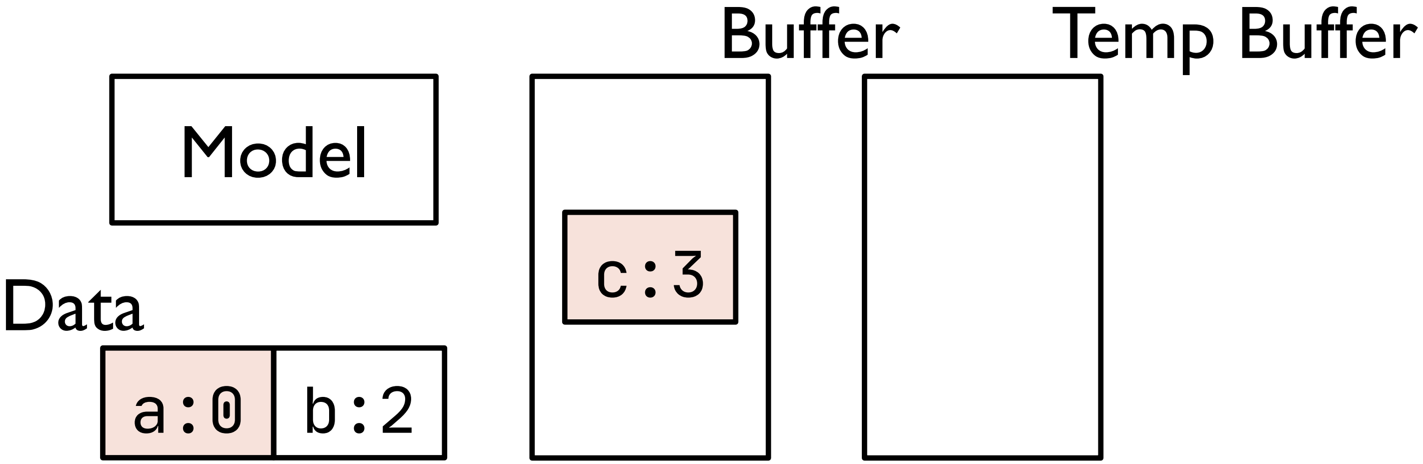
- **METHOD: 2-Phase Compaction — merge, then copy**
  - Still update in-place and compact asynchronously

Handling writes: Two-Phase Compaction



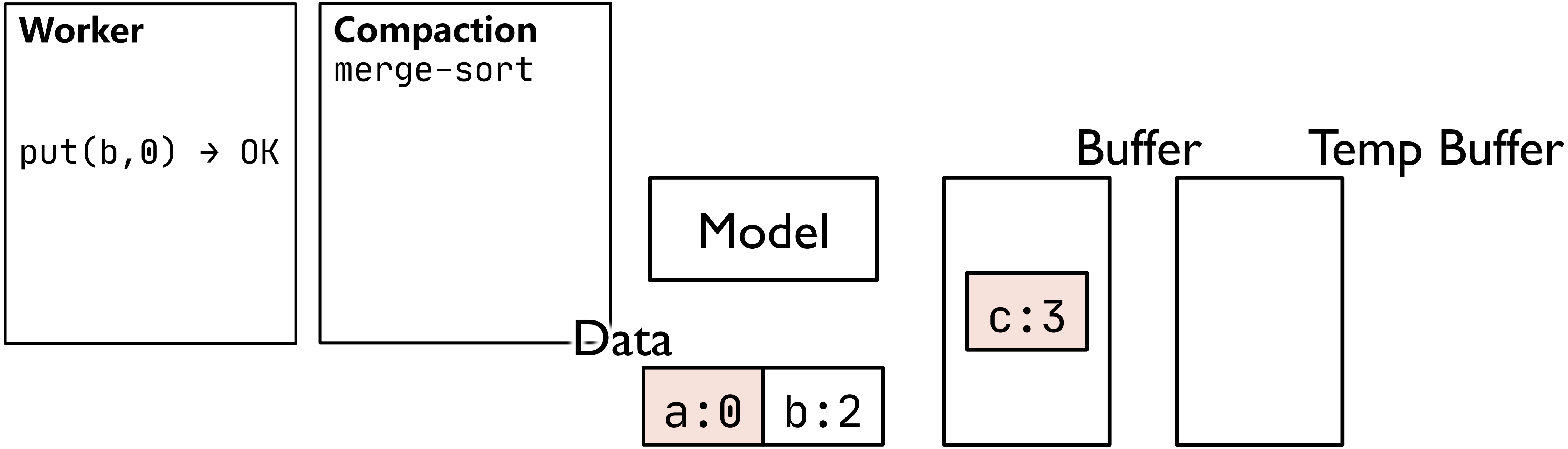
Handling writes: Two-Phase Compaction

1. **MERGE PHASE: merge-sort records on pointers**



Handling writes: Two-Phase Compaction

1. MERGE PHASE: merge-sort records on pointers



Worker

put(b,0) → OK

Compaction

merge-sort

Model

Data

a:0 | b:2

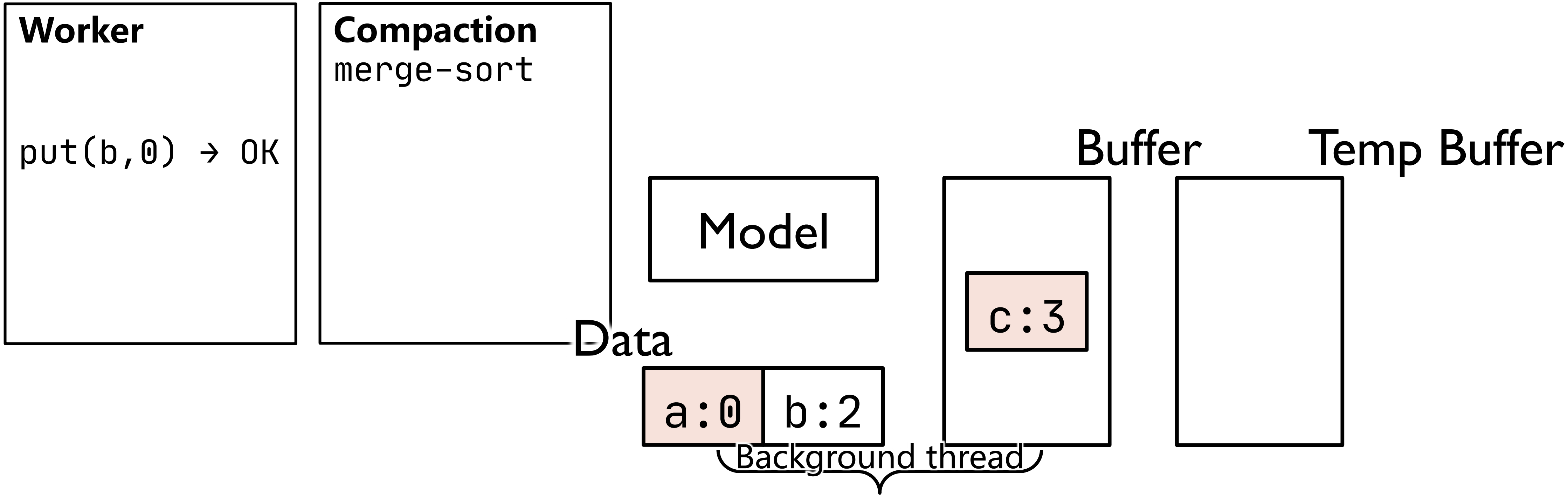
Buffer

c:3

Temp Buffer

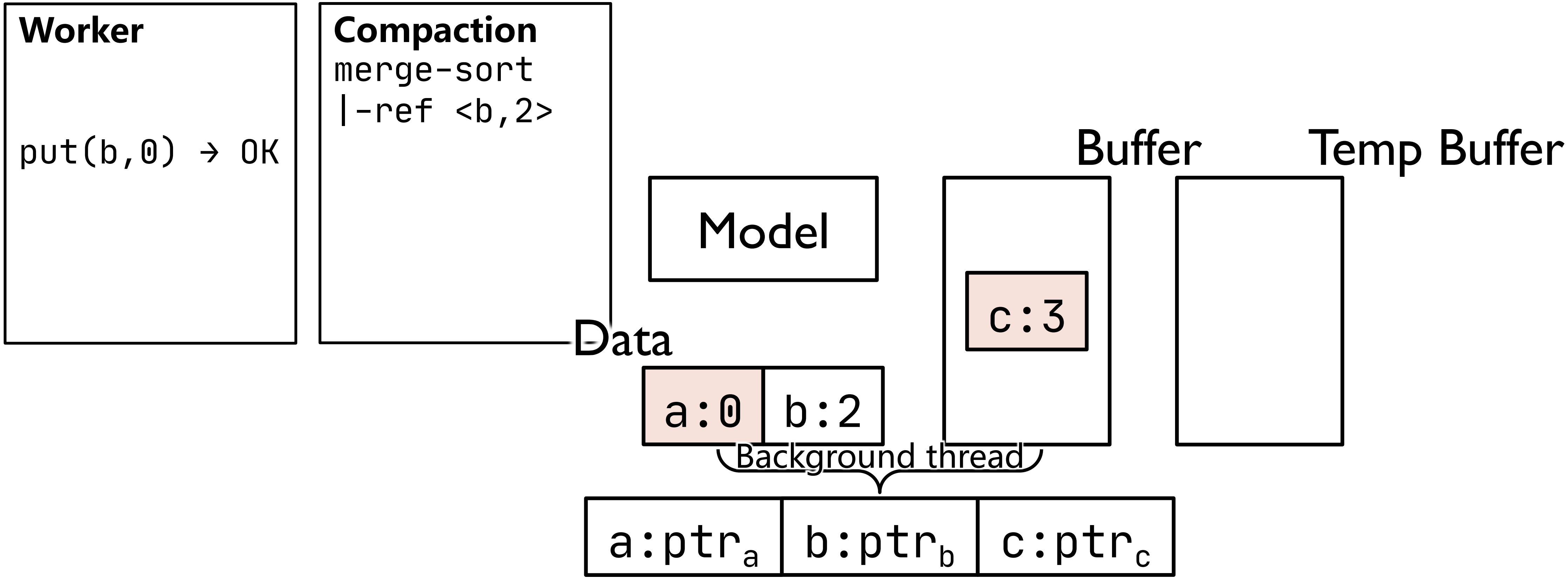
Handling writes: Two-Phase Compaction

1. MERGE PHASE: merge-sort records on pointers



Handling writes: Two-Phase Compaction

1. MERGE PHASE: merge-sort records on pointers



Worker

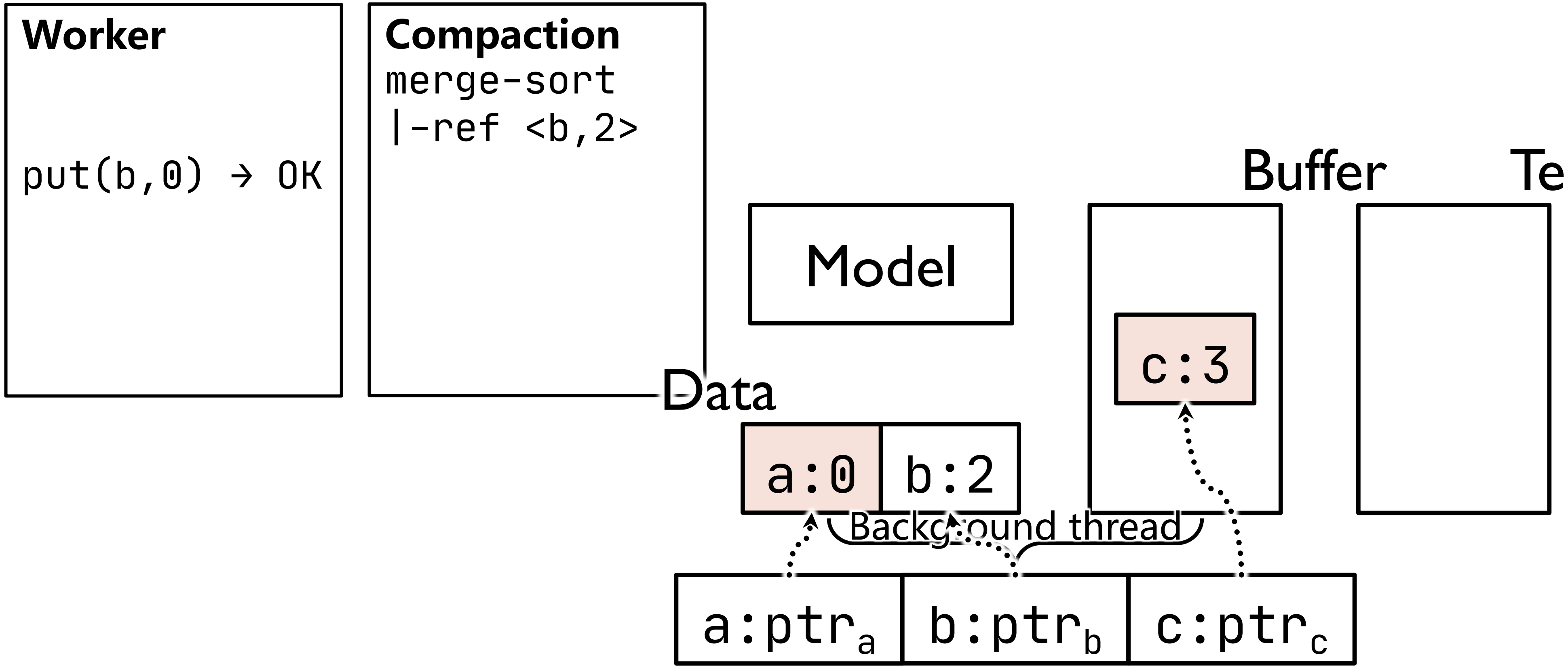
```
put(b,0) → OK
```

Compaction

```
merge-sort
|-ref <b,2>
```

Handling writes: Two-Phase Compaction

1. MERGE PHASE: merge-sort records on pointers



Worker

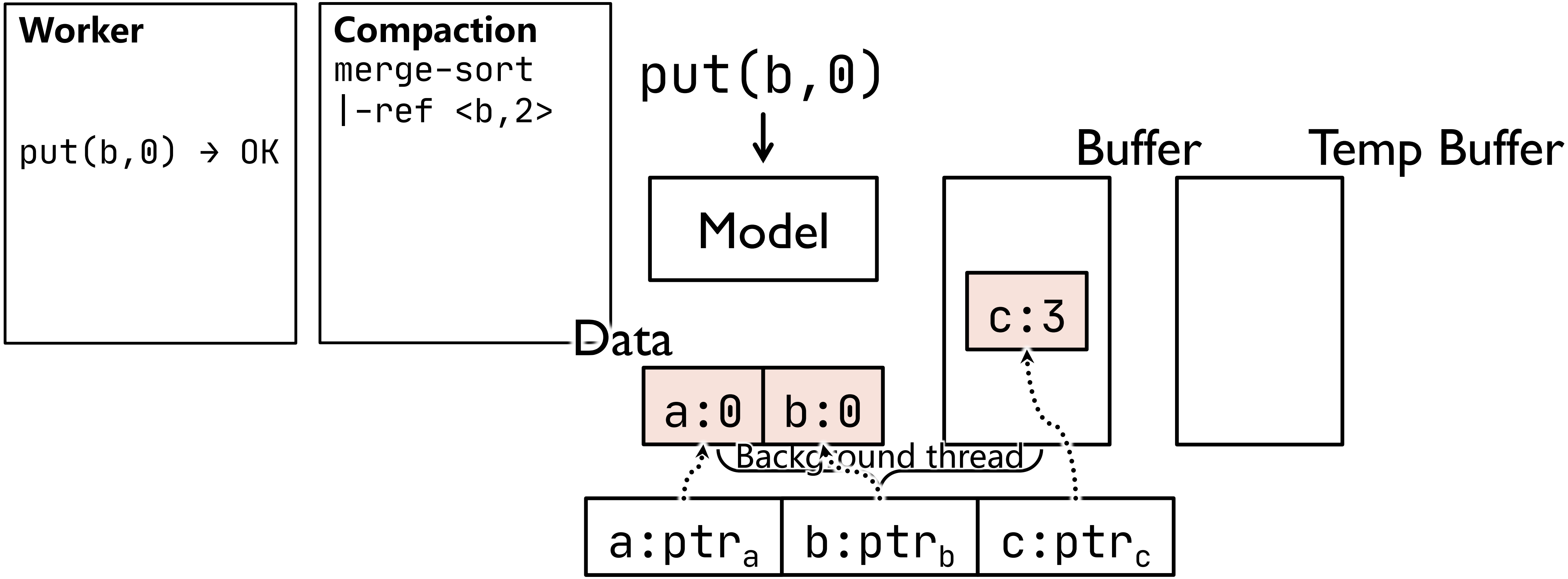
```
put(b,0) → OK
```

Compaction

```
merge-sort
|-ref <b,2>
```

Handling writes: Two-Phase Compaction

1. MERGE PHASE: merge-sort records on pointers



Worker

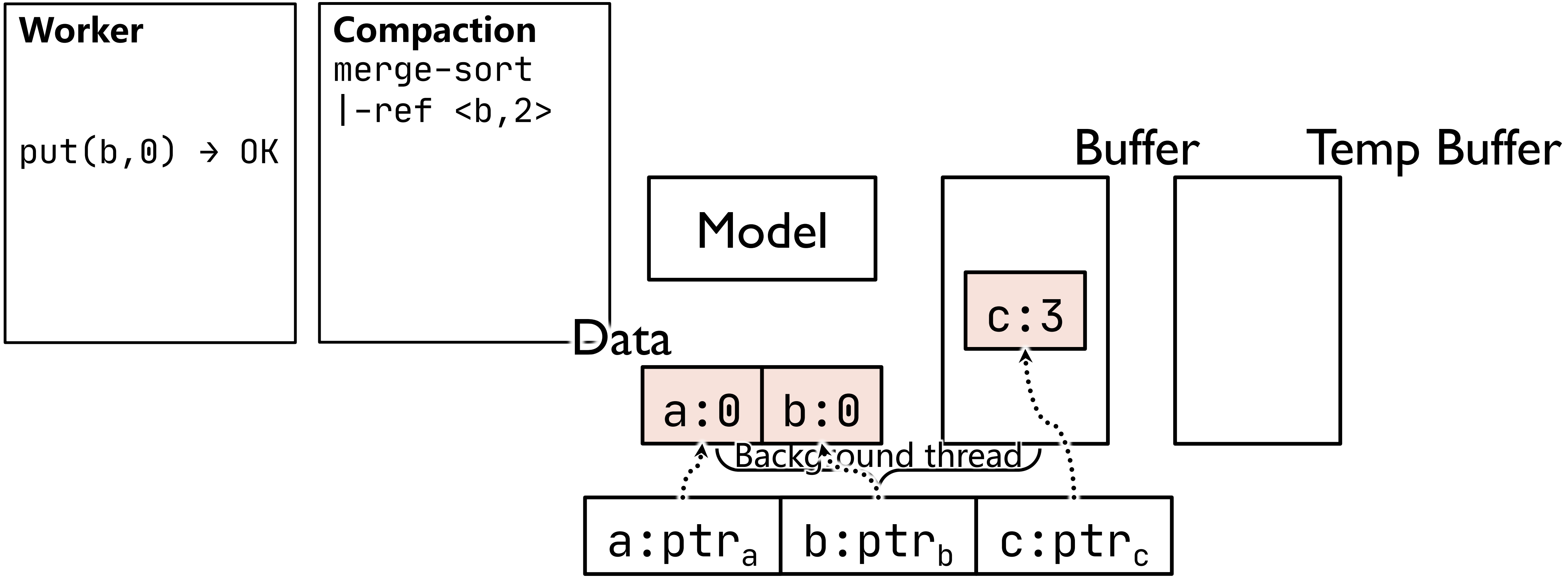
```
put(b,0) → OK
```

Compaction

```
merge-sort
|-ref <b,2>
```

Handling writes: Two-Phase Compaction

1. **MERGE PHASE: merge-sort records on pointers, update data array, and retrain model**



Worker

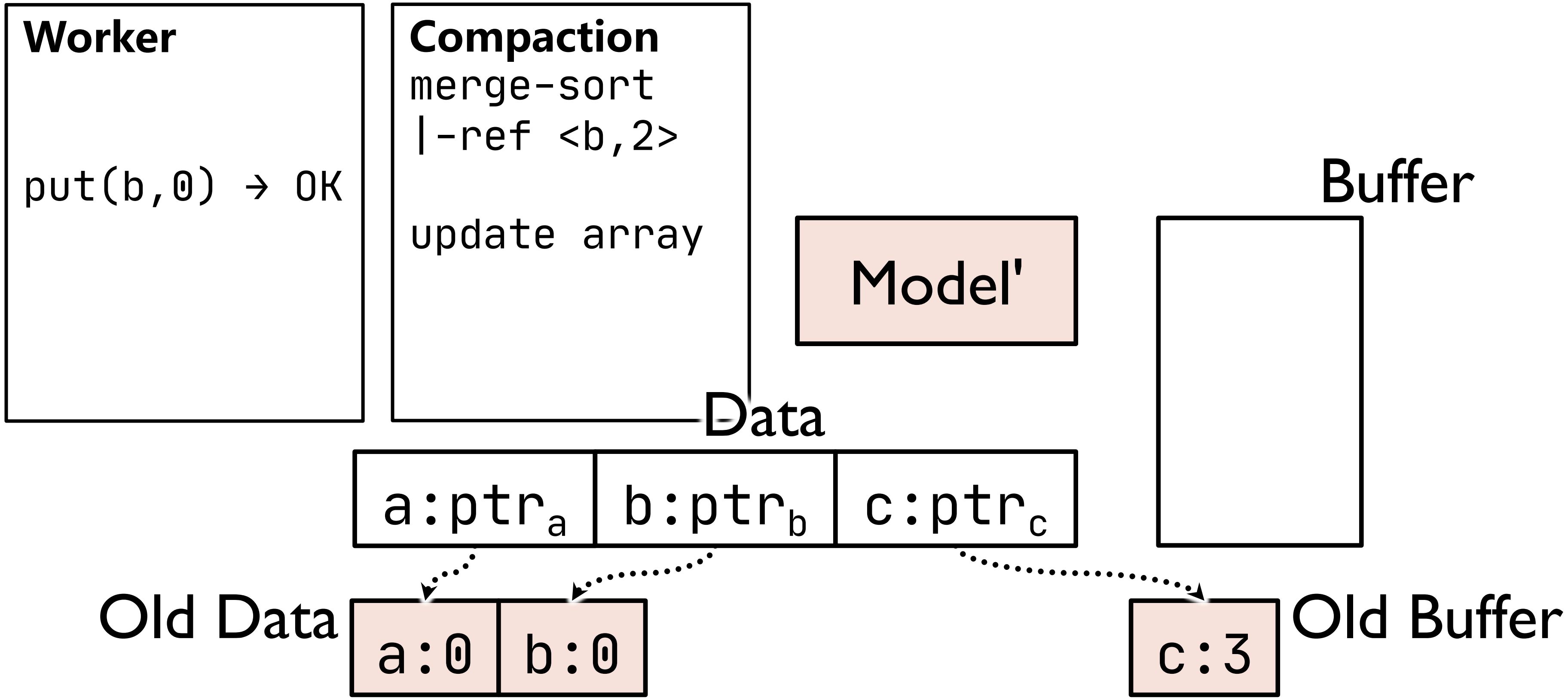
```
put(b,0) → OK
```

Compaction

```
merge-sort
|-ref <b,2>
```

Handling writes: Two-Phase Compaction

1. MERGE PHASE: merge-sort records on pointers, update data array, and retrain model



Worker

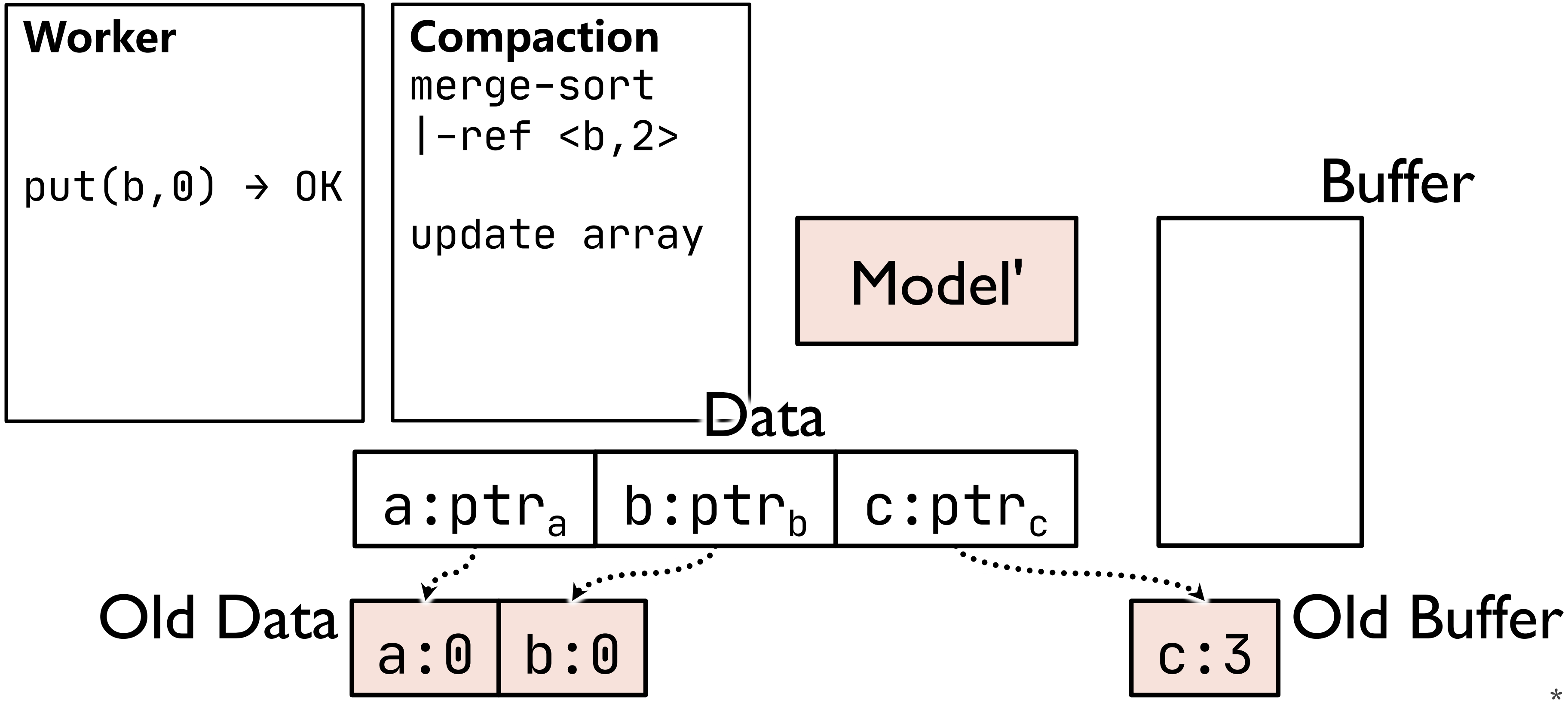
```
put(b,0) → OK
```

Compaction

```
merge-sort
|-ref <b,2>

update array
```

Handling writes: Two-Phase Compaction

2. WAIT: use RCU* barrier to ensure no direct access to old data/buffer



Worker:

```
put(b,0) → OK
```

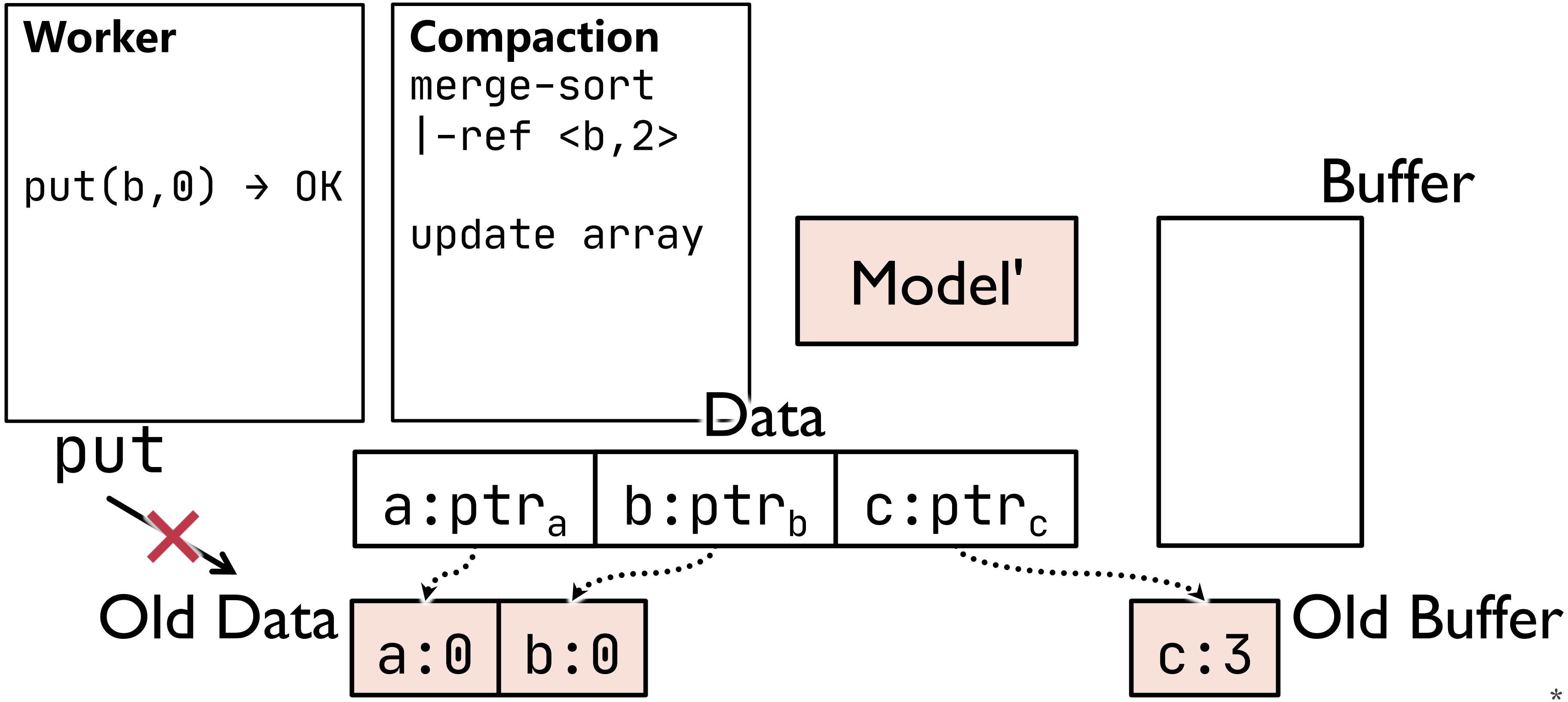
Compaction:

```
merge-sort
|-ref <b,2>

update array
```

*RCU stands for Read-Copy-Update

Handling writes: Two-Phase Compaction

2. **WAIT: use RCU* barrier to ensure no direct access to old data/buffer**



Worker

```
put(b,0) → OK
```

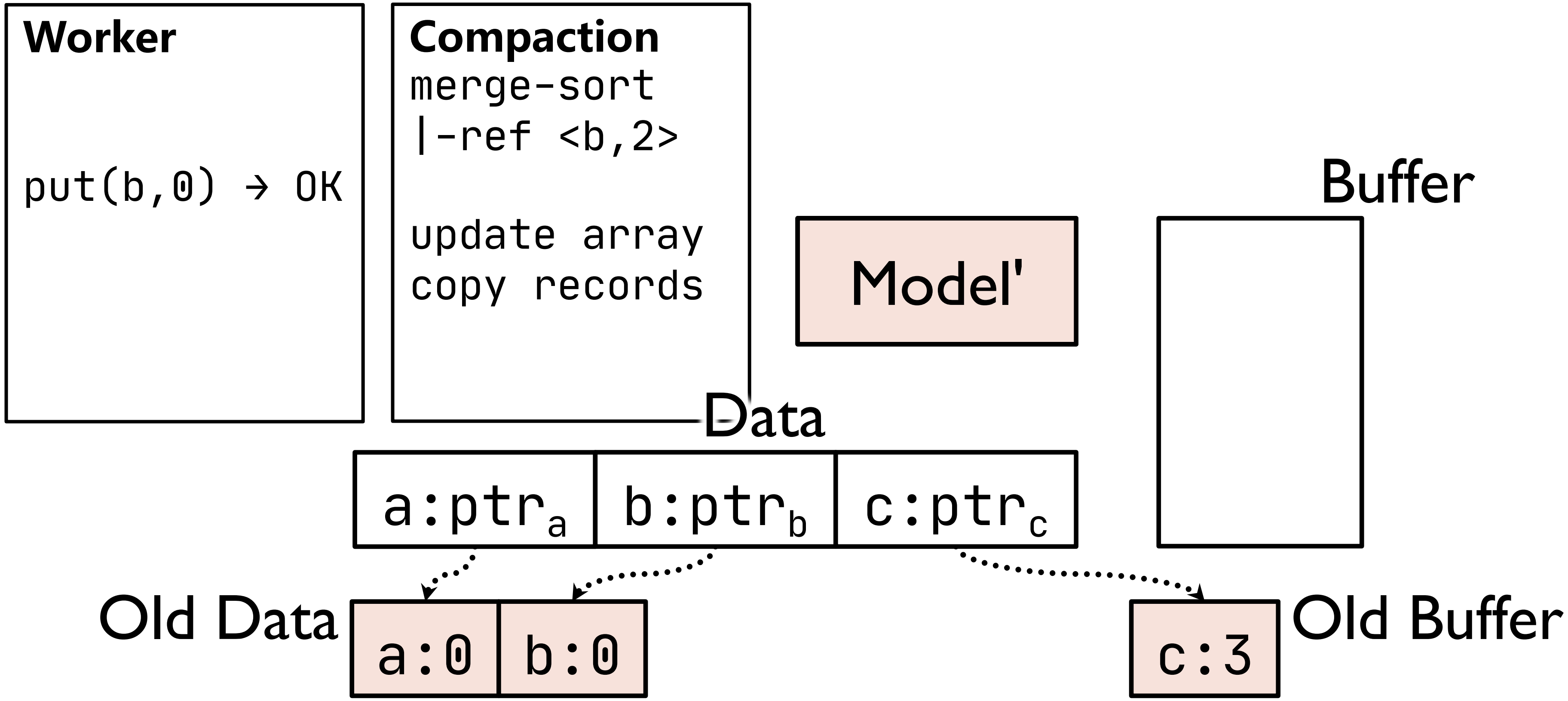
Compaction

```
merge-sort
|-ref <b,2>

update array
```

*RCU stands for Read-Copy-Update

Handling writes: Two-Phase Compaction

3. COPY PHASE: copy the latest records via pointers



Worker

```
put(b,0) → OK
```

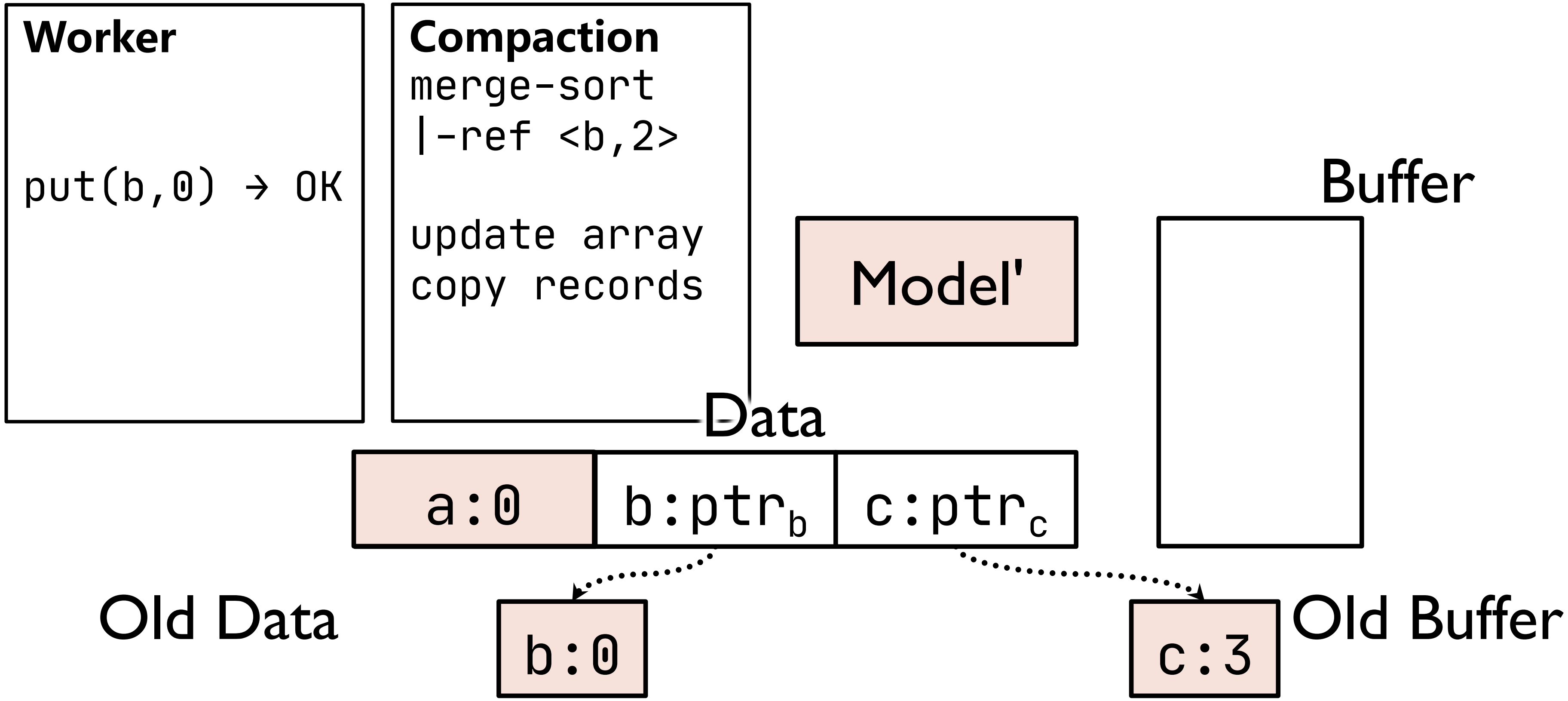
Compaction

```
merge-sort
|-ref <b,2>

update array
copy records
```

a:ptr_a	b:ptr_b	c:ptr_c

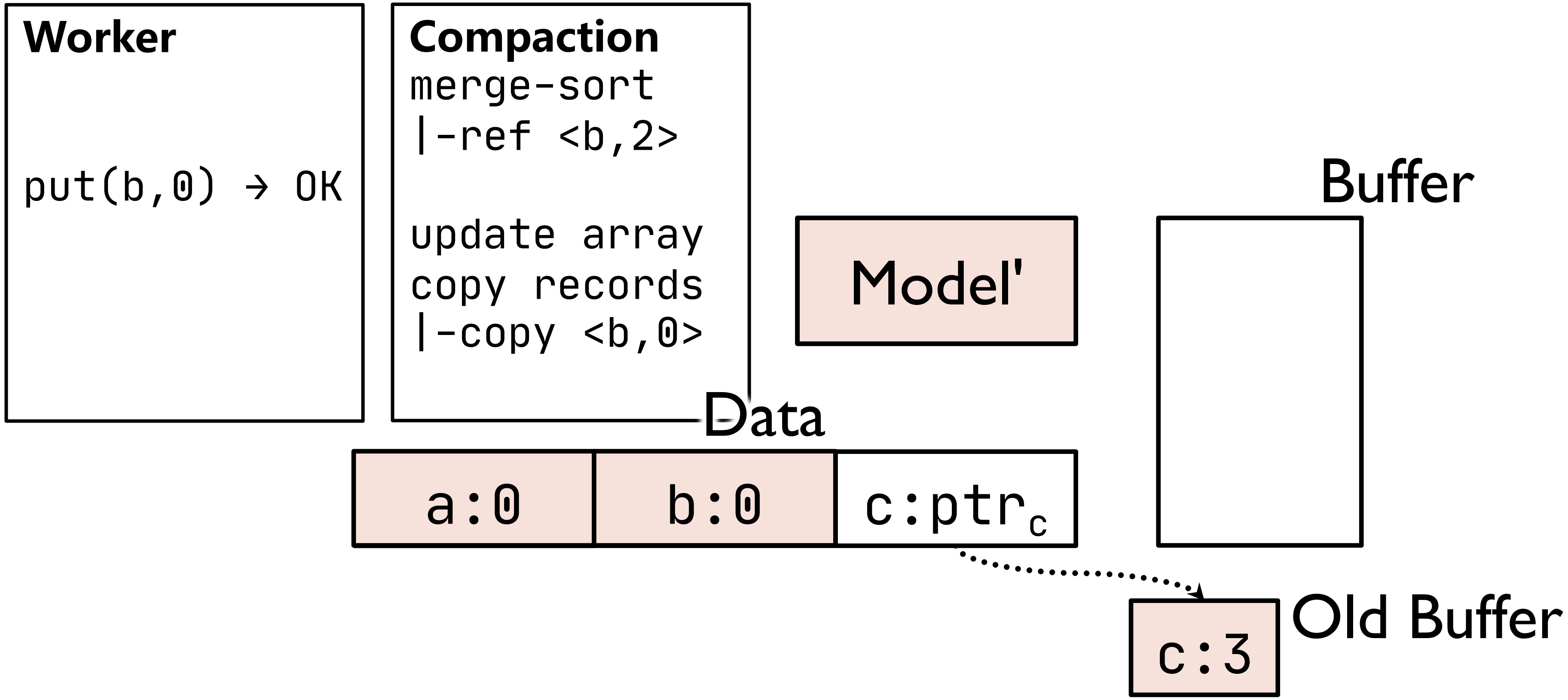
Handling writes: Two-Phase Compaction

3. COPY PHASE: copy the latest records via pointers



Handling writes: Two-Phase Compaction

3. COPY PHASE: copy the latest records via pointers



Worker

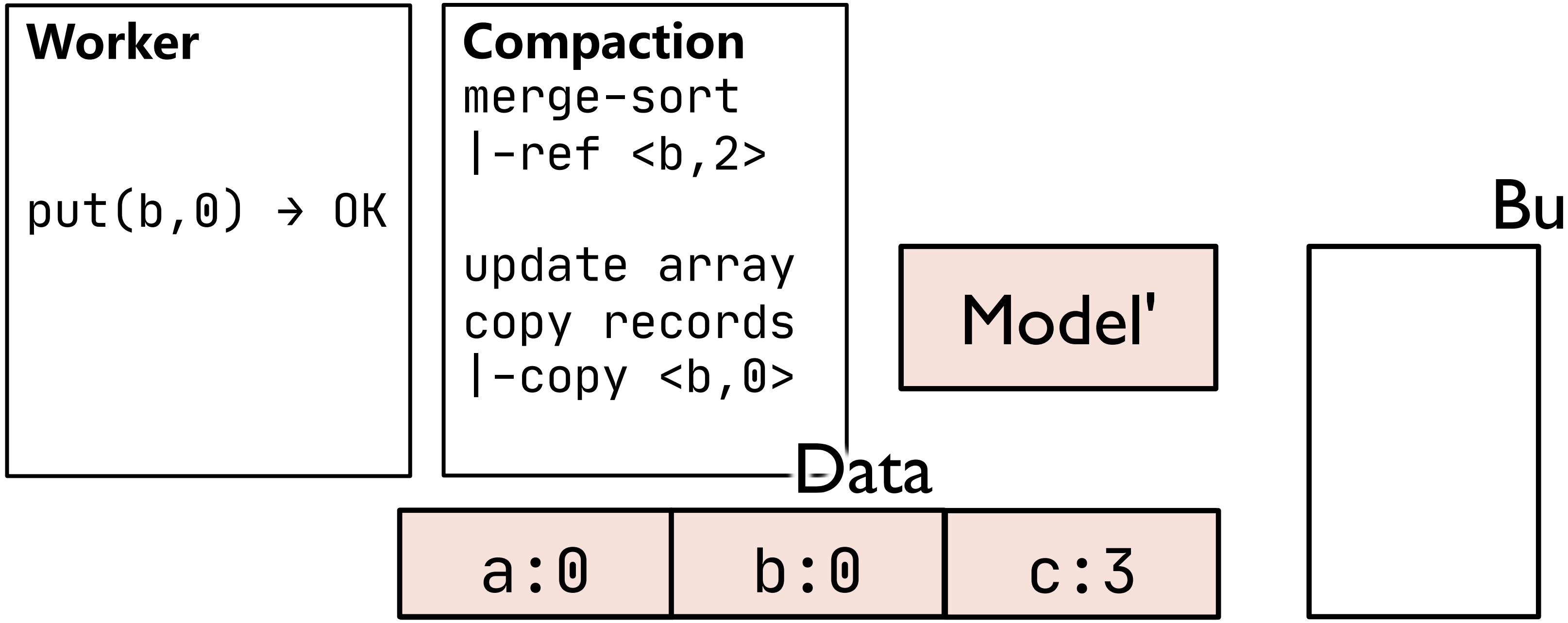
```
put(b,0) → OK
```

Compaction

```
merge-sort
|-ref <b,2>

update array
copy records
|-copy <b,0>
```

Handling writes: Two-Phase Compaction

3. COPY PHASE: copy the latest records via pointers



Worker

```
put(b,0) → OK
```

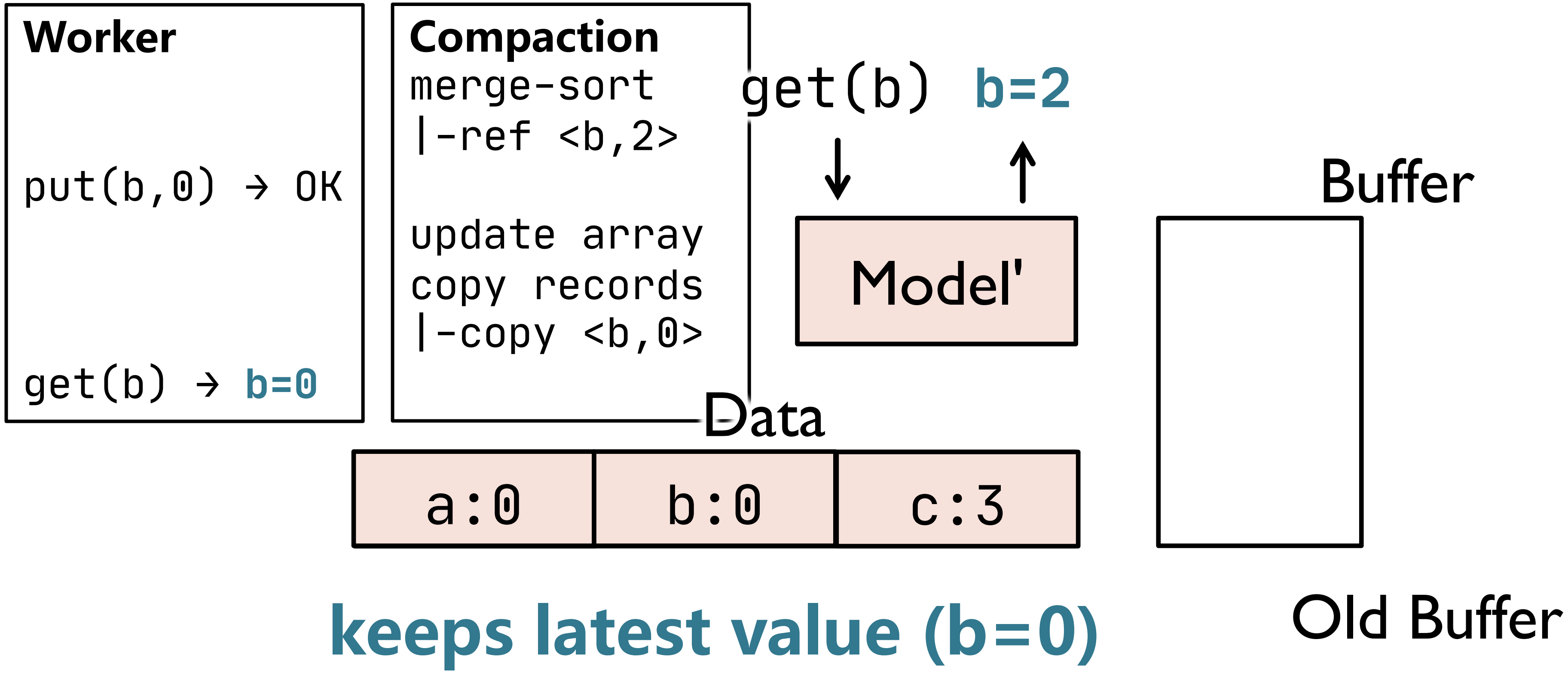
Compaction

```
merge-sort
|-ref <b,2>

update array
copy records
|-copy <b,0>
```

keeps latest value (b=0)

Handling writes: Two-Phase Compaction

3. COPY PHASE: copy the latest records via pointers



Worker

```
put(b,0) → OK

get(b) → b=0
```

Compaction

```
merge-sort
|-ref <b,2>

update array
copy records
|-copy <b,0>
```

keeps latest value (b=0)

Handling writes: Two-Phase Compaction

- Cannot slow down reads ✓
- Cannot block writes ✓
- Must retain all updates ✗

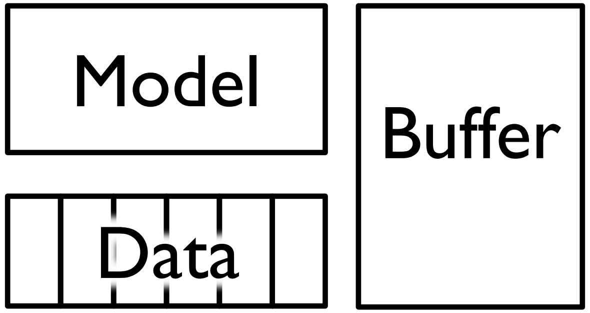
Handling writes: Two-Phase Compaction

Cannot slow down reads ✓

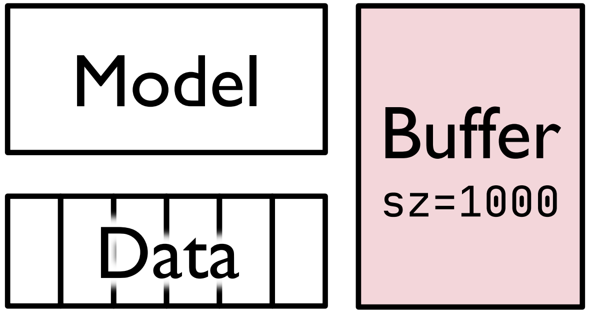
Cannot block writes ✓

Must retain all updates ✓

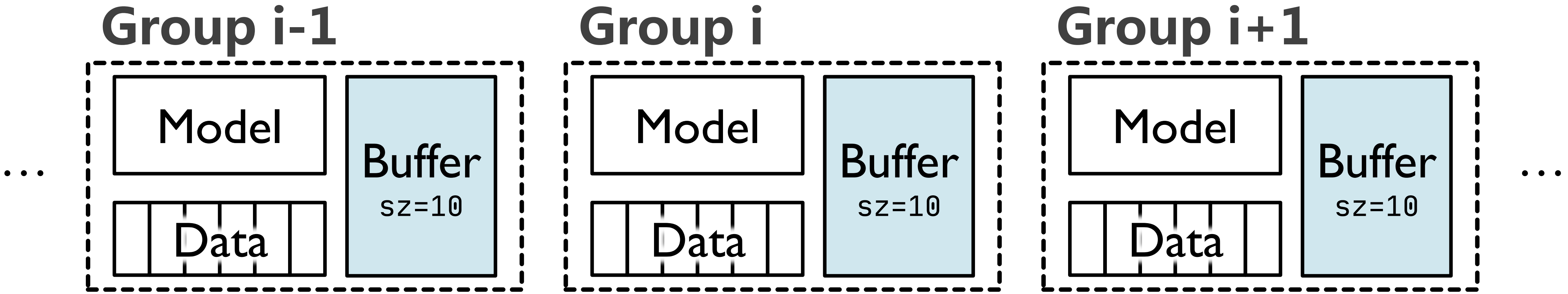
Handling writes: range-partitioning



Handling writes: range-partitioning

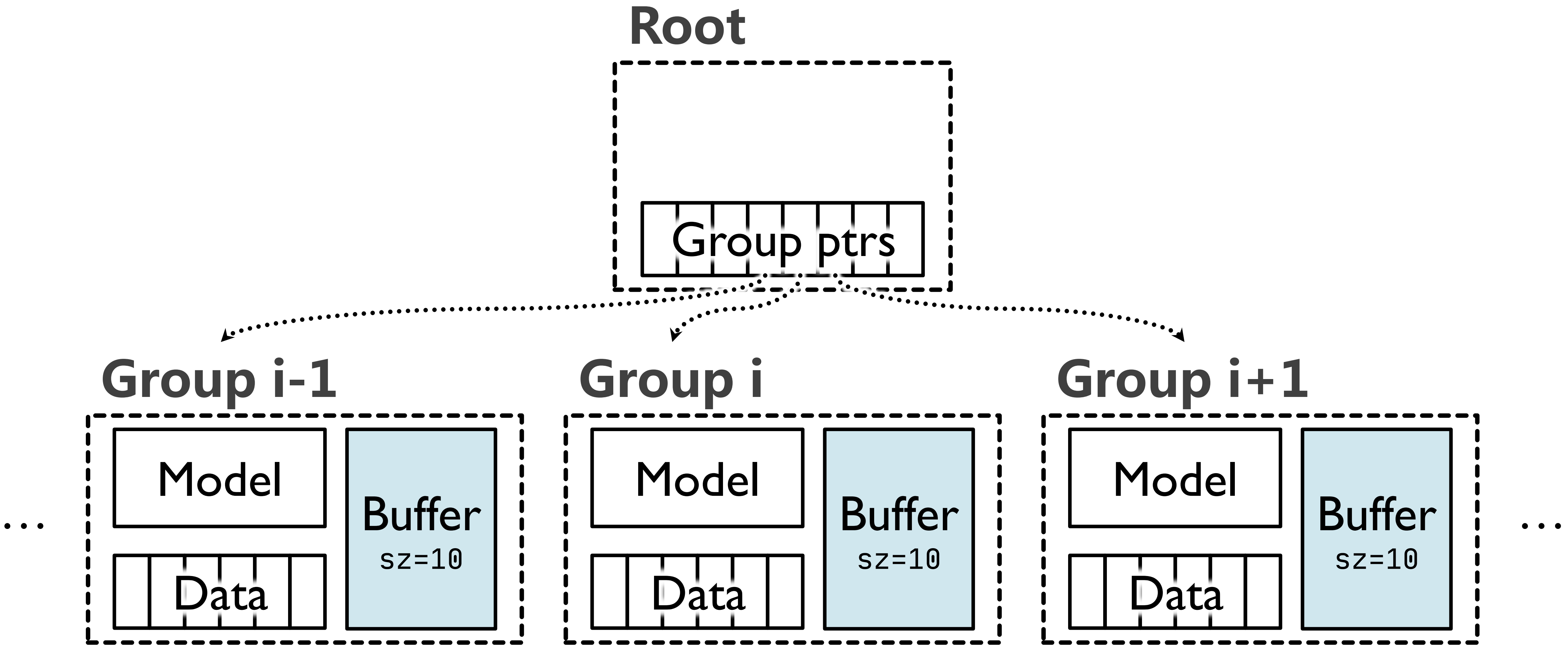


Handling writes: range-partitioning



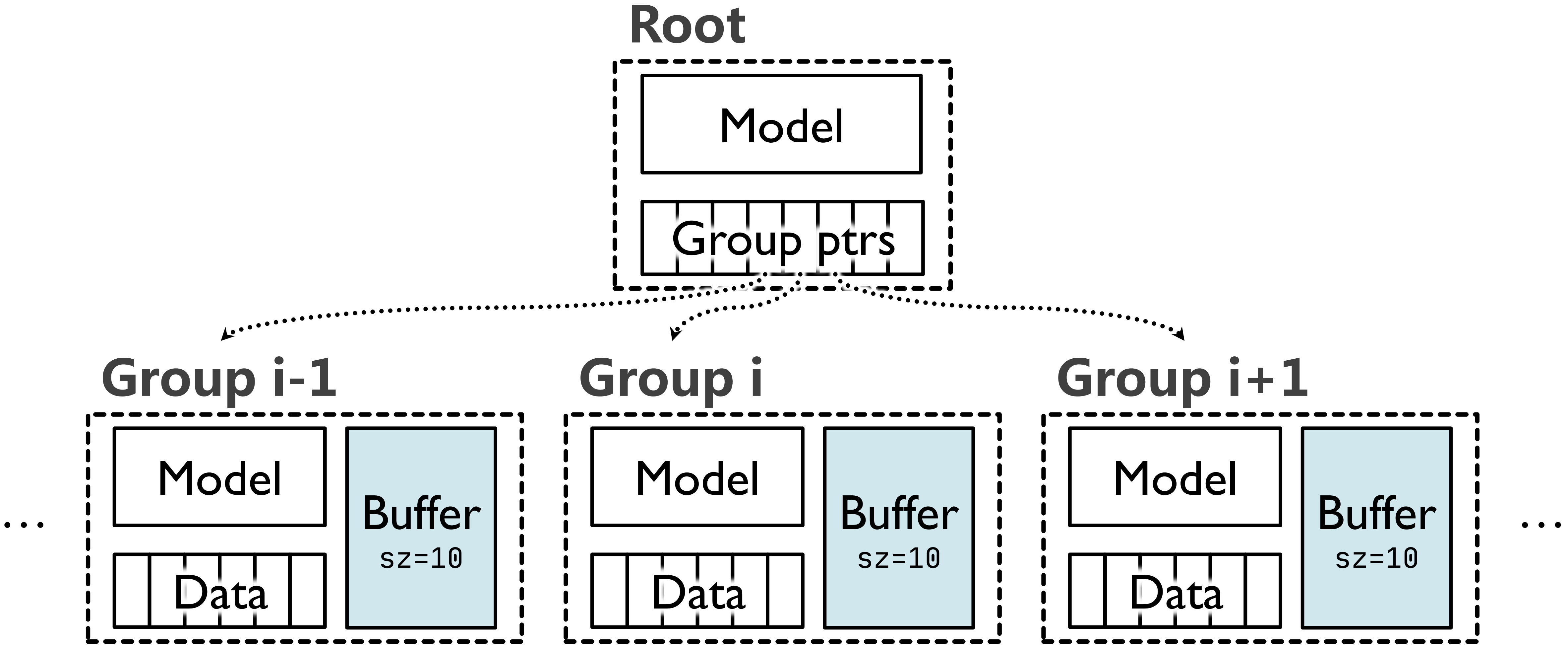
Range-partition data into group nodes

Handling writes: range-partitioning



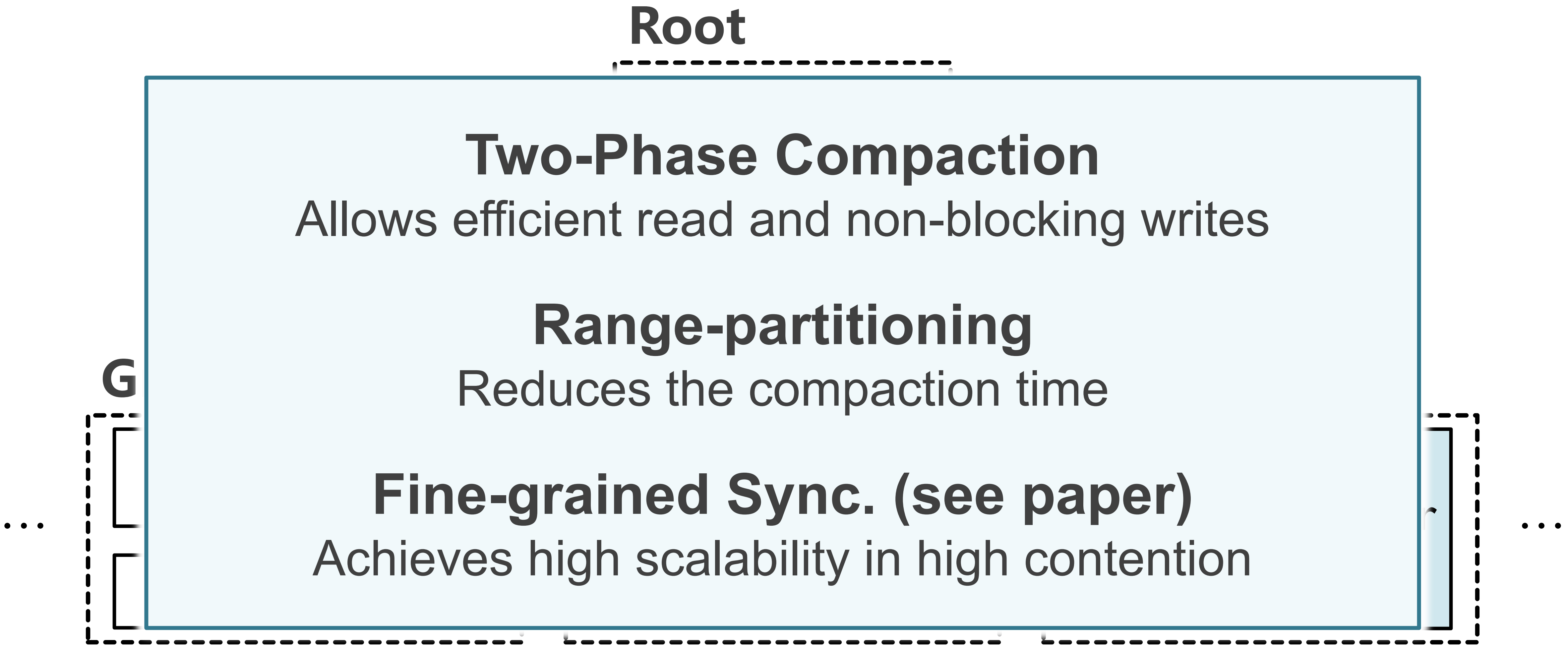
Range-partition data into group nodes

Handling writes: range-partitioning



Range-partition data into group nodes

Handling writes: range-partitioning



Two-Phase Compaction

Allows efficient read and non-blocking writes

Range-partitioning

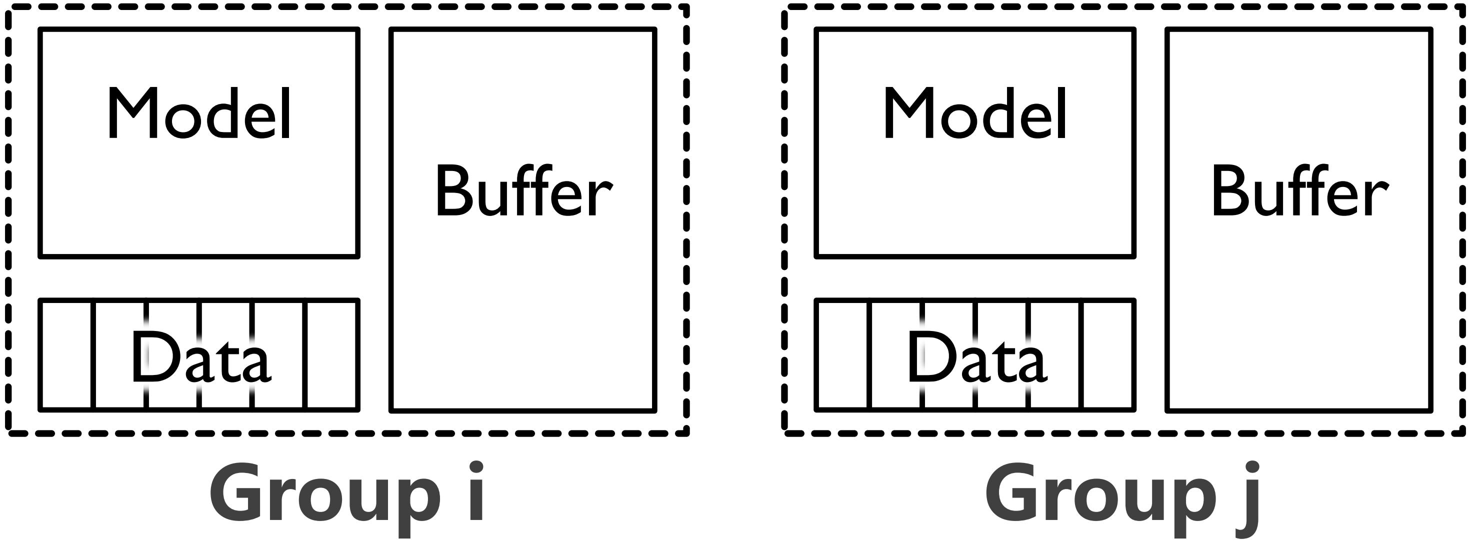
Reduces the compaction time

Fine-grained Sync. (see paper)

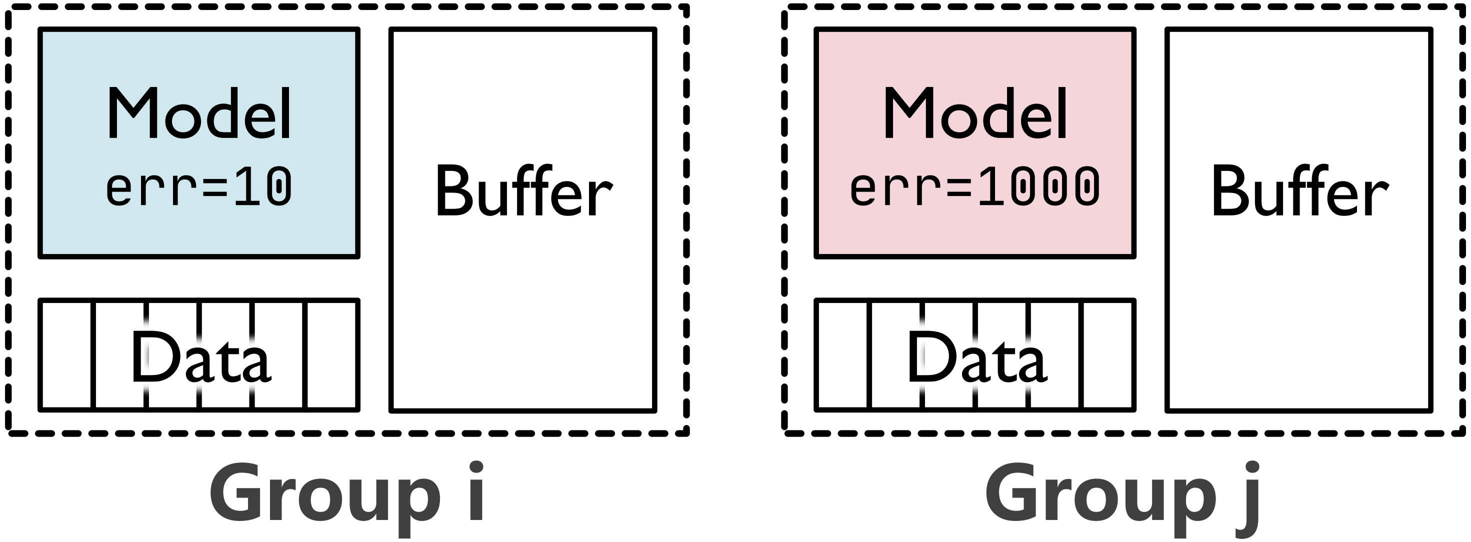
Achieves high scalability in high contention

Range-partition data into group nodes

Dynamic workloads: the problem



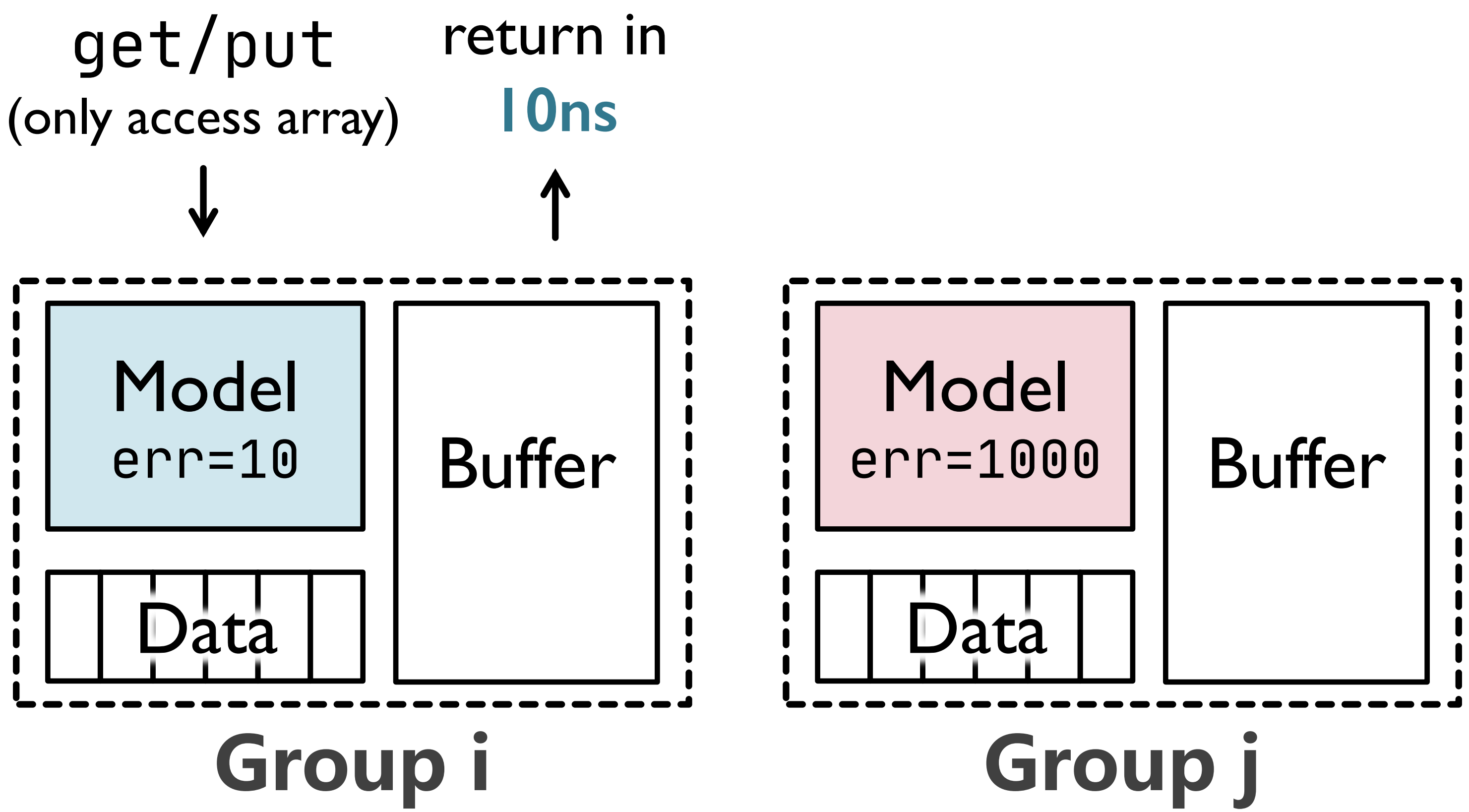
Dynamic workloads: the problem



🟥 Large model error

🟦 Small model error

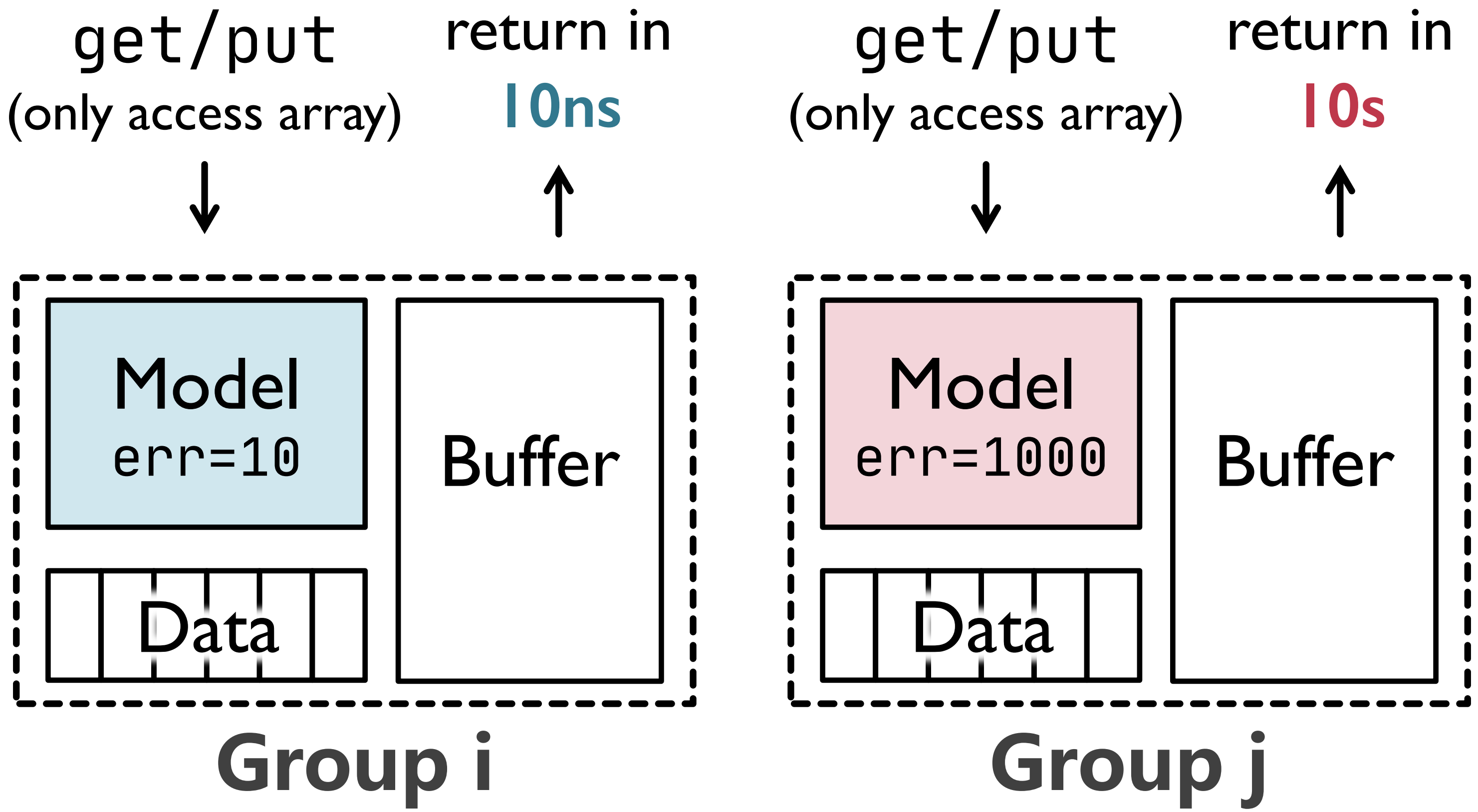
Dynamic workloads: the problem



🟥 Large model error

🟦 Small model error

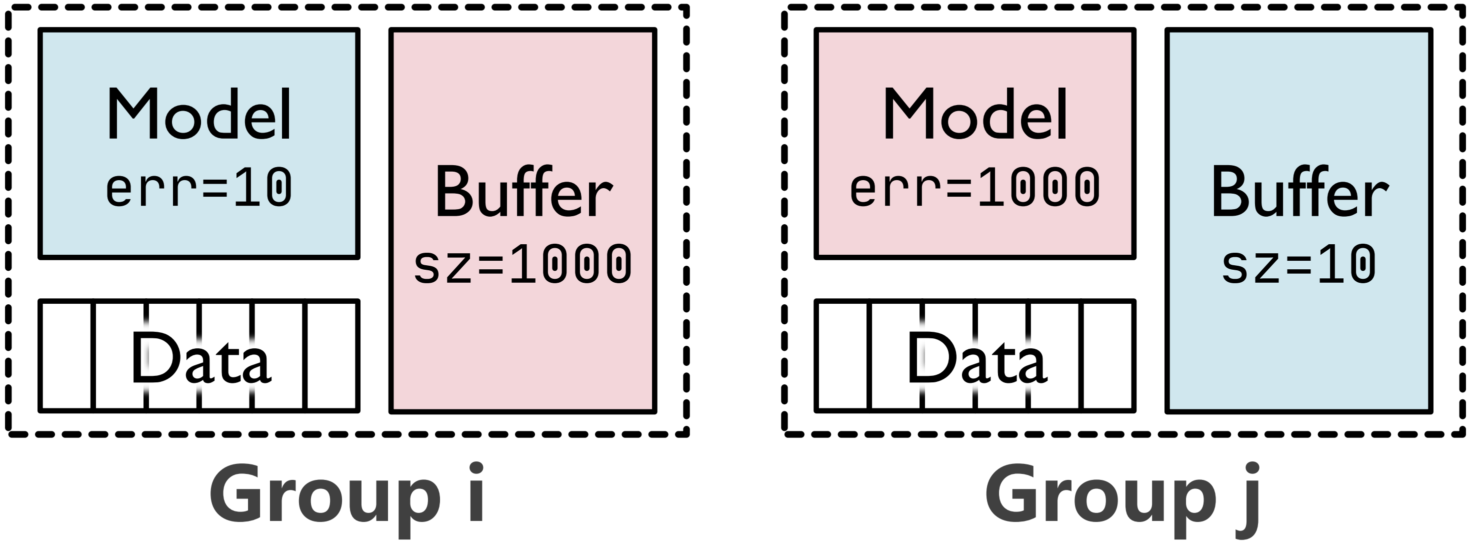
Dynamic workloads: the problem



🟥 Large model error

🟦 Small model error

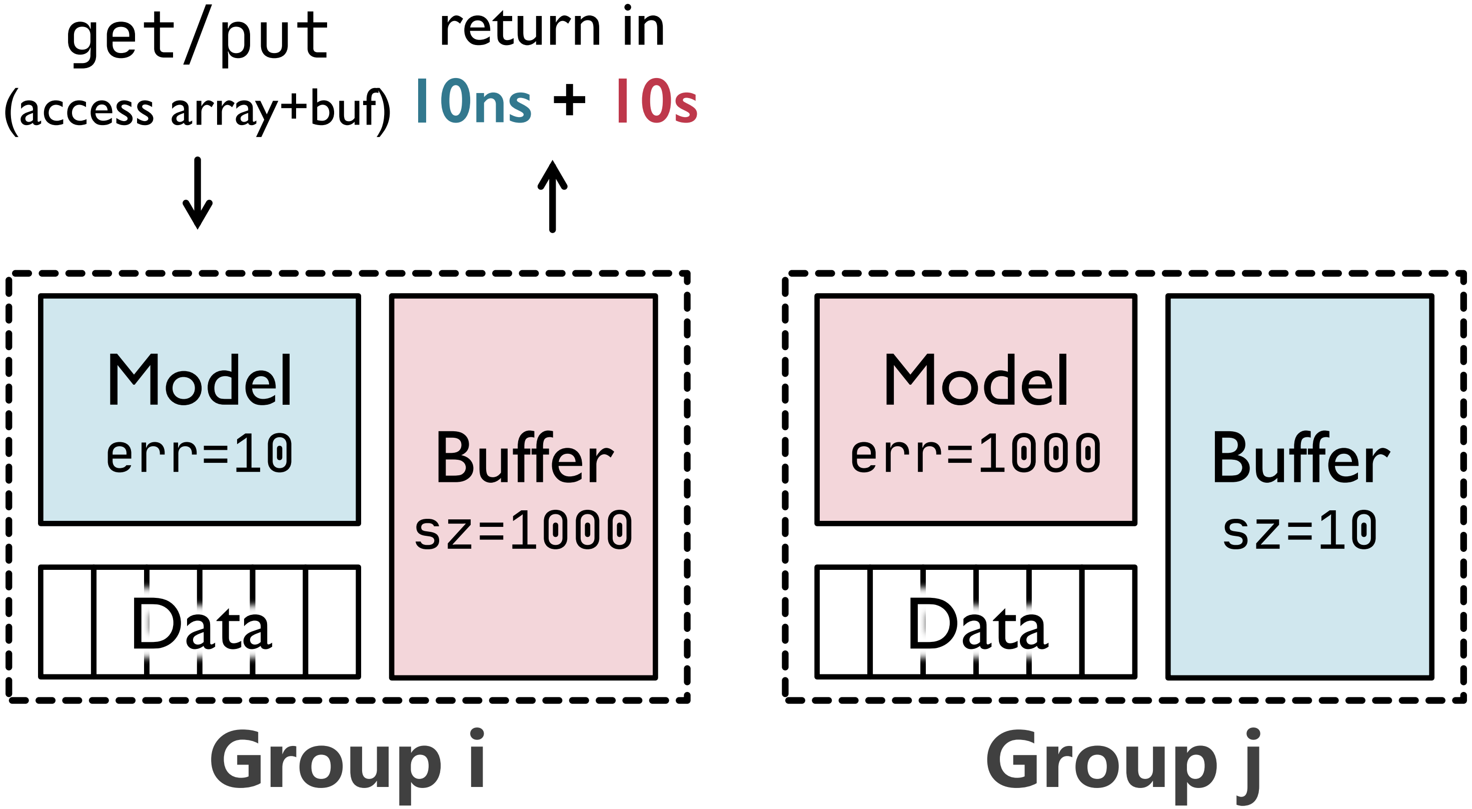
Dynamic workloads: the problem



🟥 Large model error/buffer size

🟦 Small model error/buffer size

Dynamic workloads: the problem



`get/put`
(access array+buf)

return in
10ns + 10s

Model
`err=10`

Buffer
`sz=1000`

Data

Group i

Model
`err=1000`

Buffer
`sz=10`

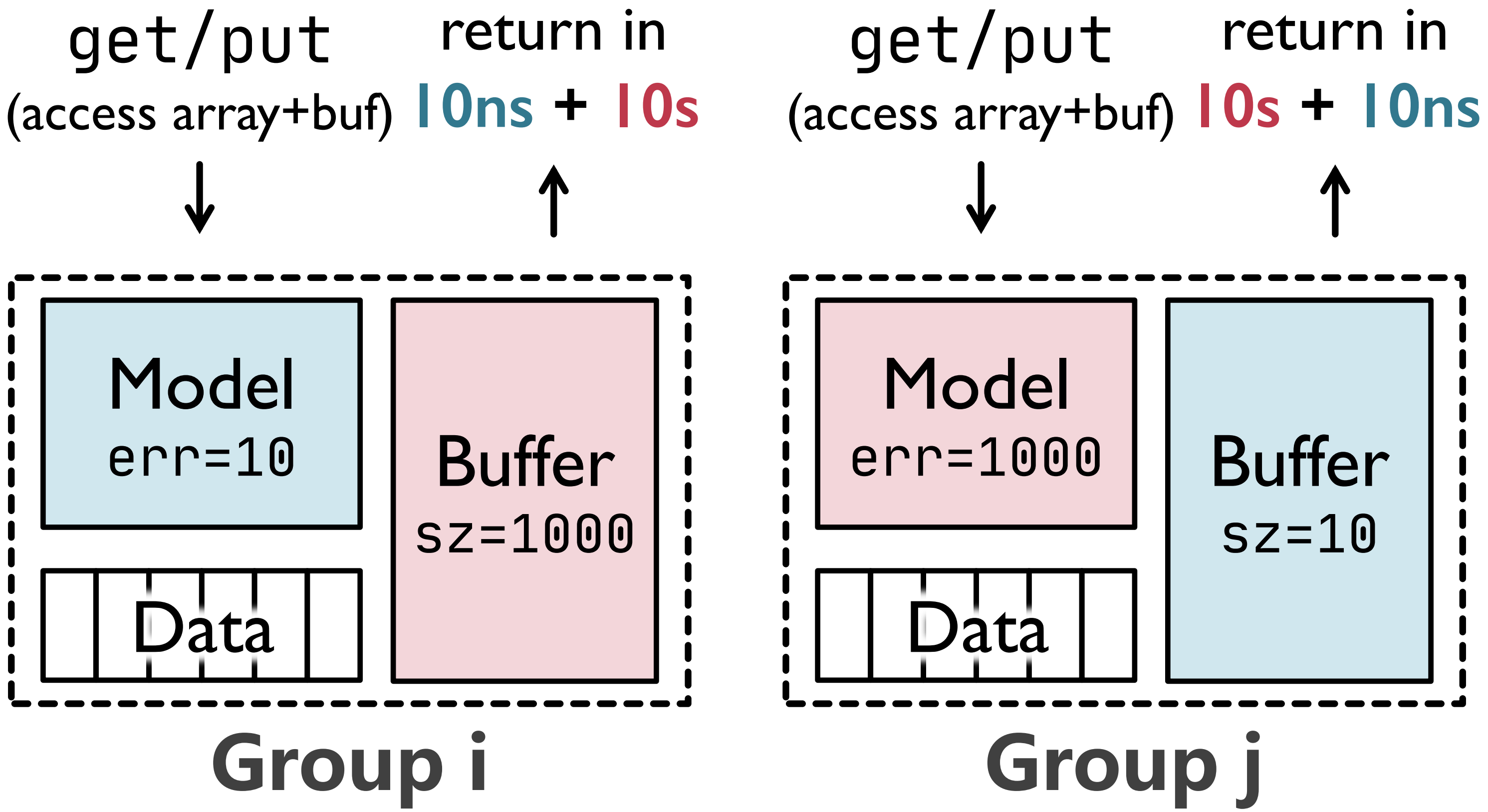
Data

Group j

🟥 Large model error/buffer size

🟦 Small model error/buffer size

Dynamic workloads: the problem



get/put
(access array+buf)

return in
10ns + 10s

Model
err=10

Buffer
sz=1000

Data

Group i

get/put
(access array+buf)

return in
10s + 10ns

Model
err=1000

Buffer
sz=10

Data

Group j

🟥 Large model error/buffer size

🟦 Small model error/buffer size

Dynamic workloads: controlling errors

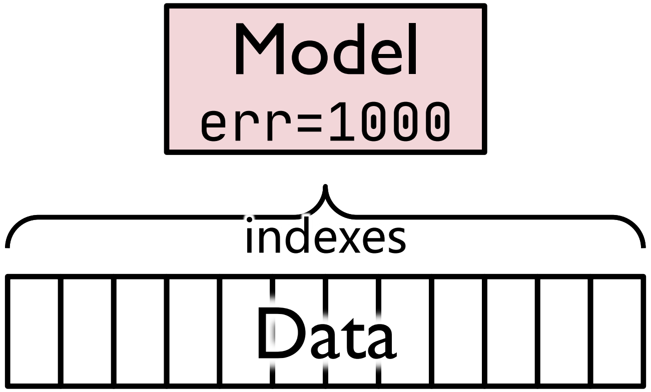
Model Split

Model Merge

Dynamic workloads: controlling errors

Model Split

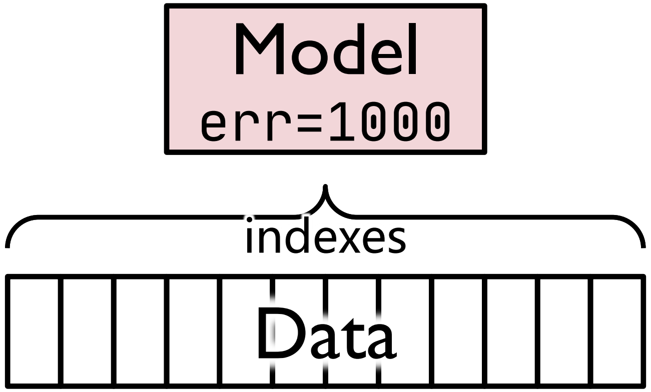
to reduce model error



Model Merge

Dynamic workloads: controlling errors

Model Split
to reduce model error

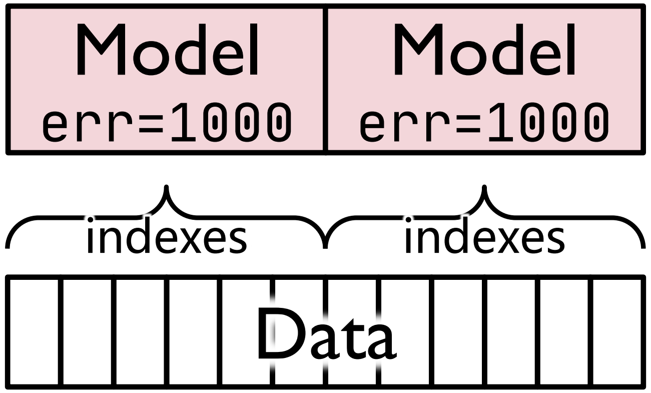


Model Merge

Dynamic workloads: controlling errors

Model Split

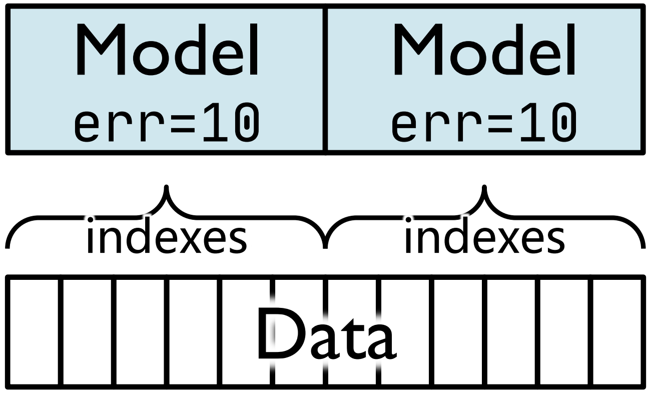
to reduce model error



Model Merge

Dynamic workloads: controlling errors

Model Split
to reduce model error

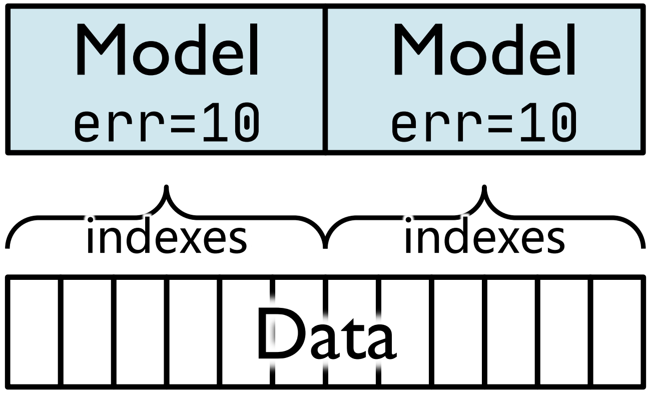


Model Merge

Dynamic workloads: controlling errors

Model Split

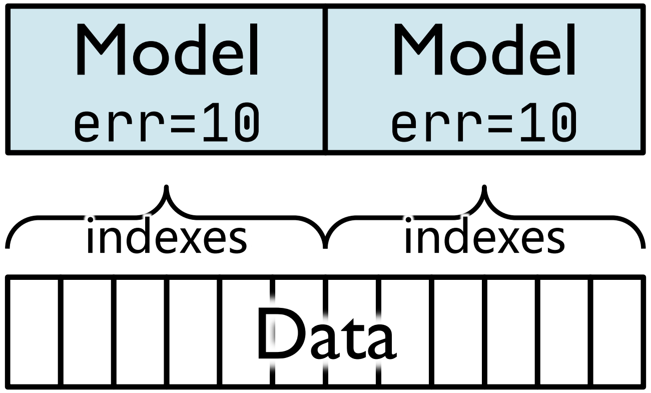
to reduce model error



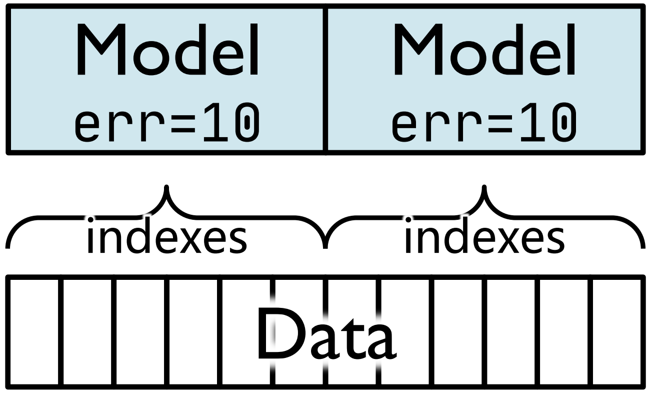
Model Merge

Dynamic workloads: controlling errors

Model Split
to reduce model error

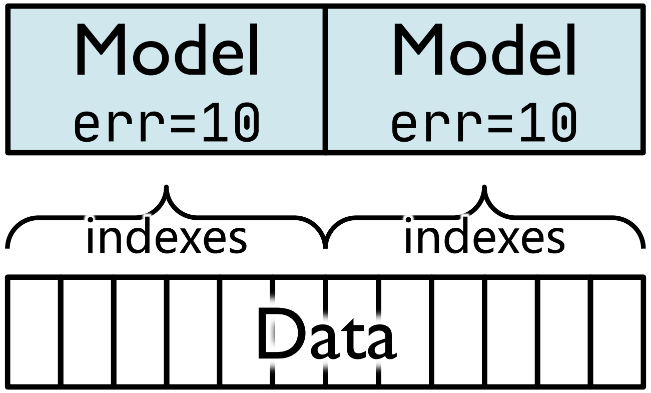


Model Merge
to reduce model #

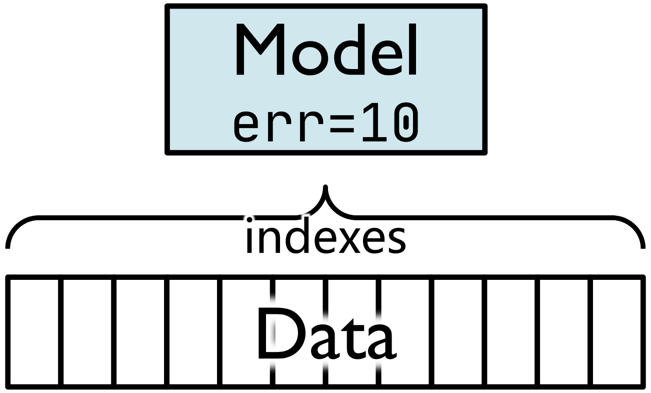


Dynamic workloads: controlling errors

Model Split
to reduce model error

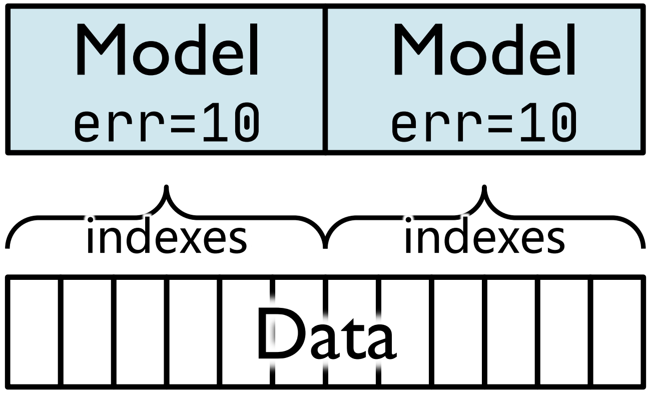


Model Merge
to reduce model #

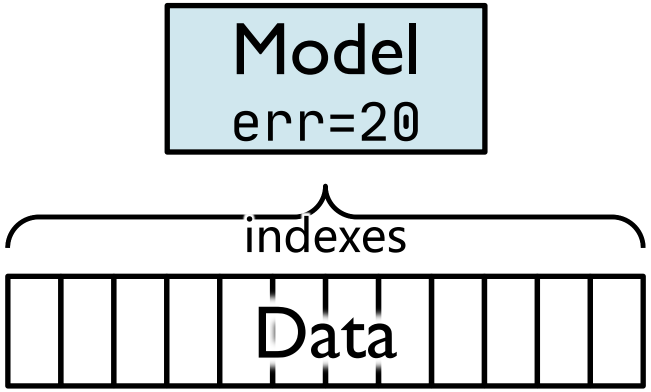


Dynamic workloads: controlling errors

Model Split
to reduce model error



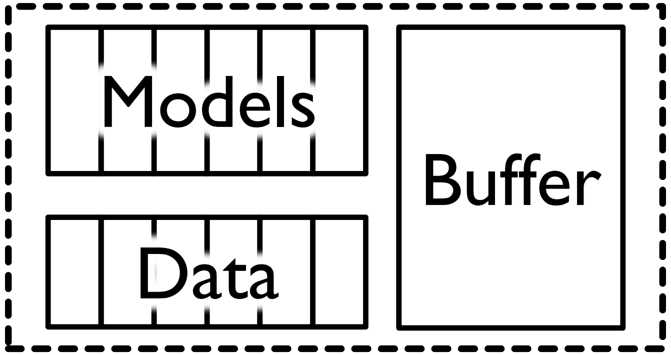
Model Merge
to reduce model #



Dynamic workloads: controlling buffer size

Group Split

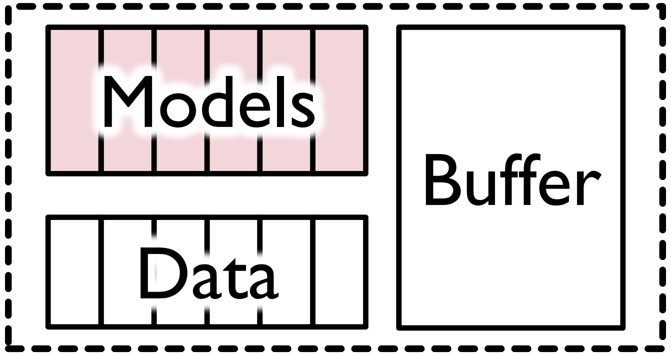
to reduce model error and buffer size



Dynamic workloads: controlling buffer size

Group Split

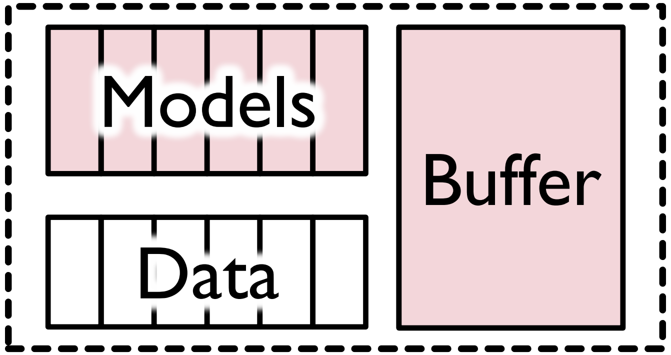
to reduce model error and buffer size



Dynamic workloads: controlling buffer size

Group Split

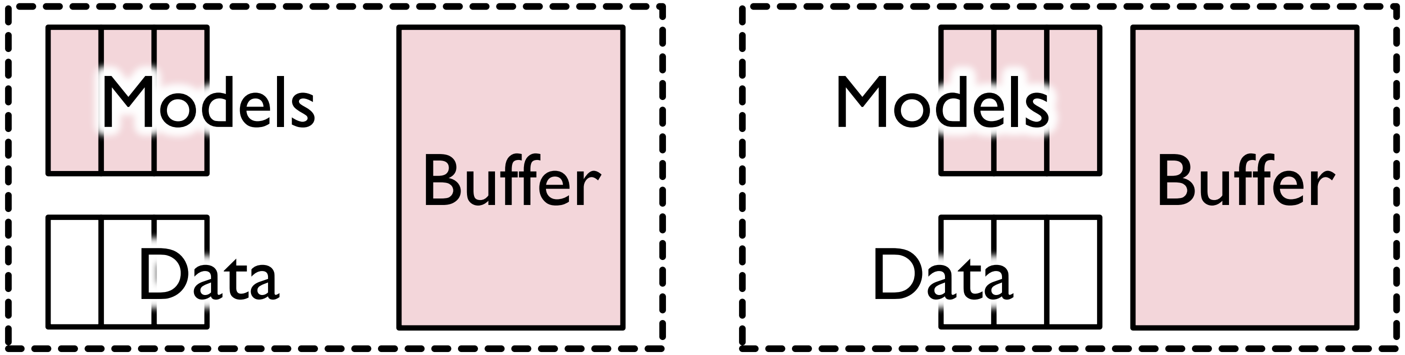
to reduce model error and buffer size



Dynamic workloads: controlling buffer size

Group Split

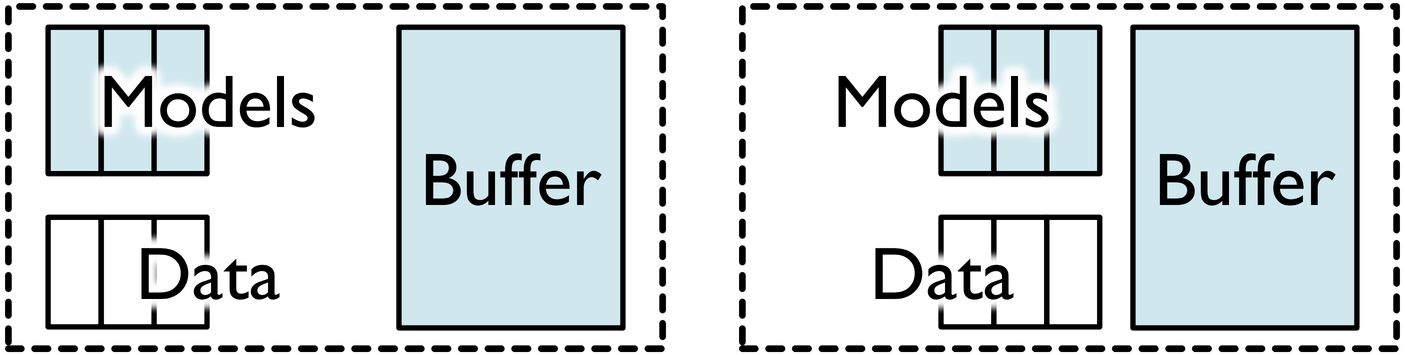
to reduce model error and buffer size



Dynamic workloads: controlling buffer size

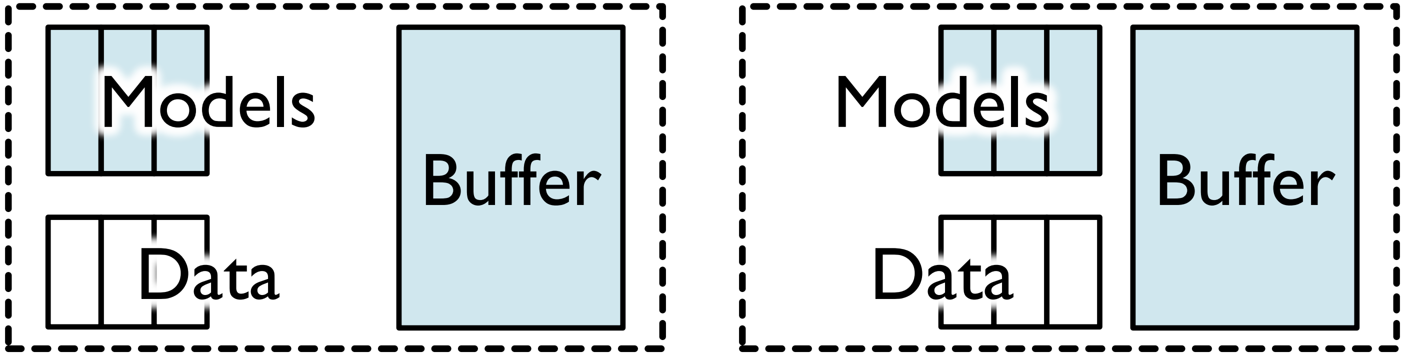
Group Split

to reduce model error and buffer size



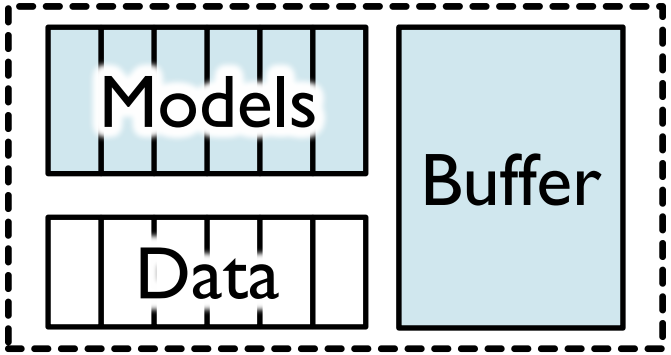
Dynamic workloads: controlling buffer size

Group Merge
to reduce group #

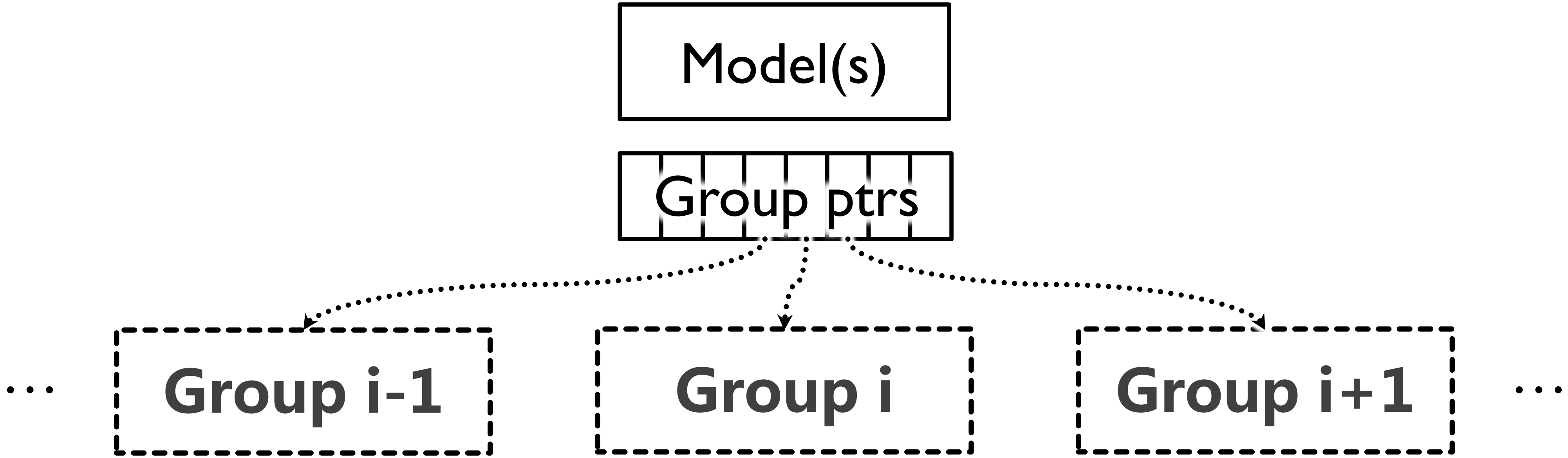


Dynamic workloads: controlling buffer size

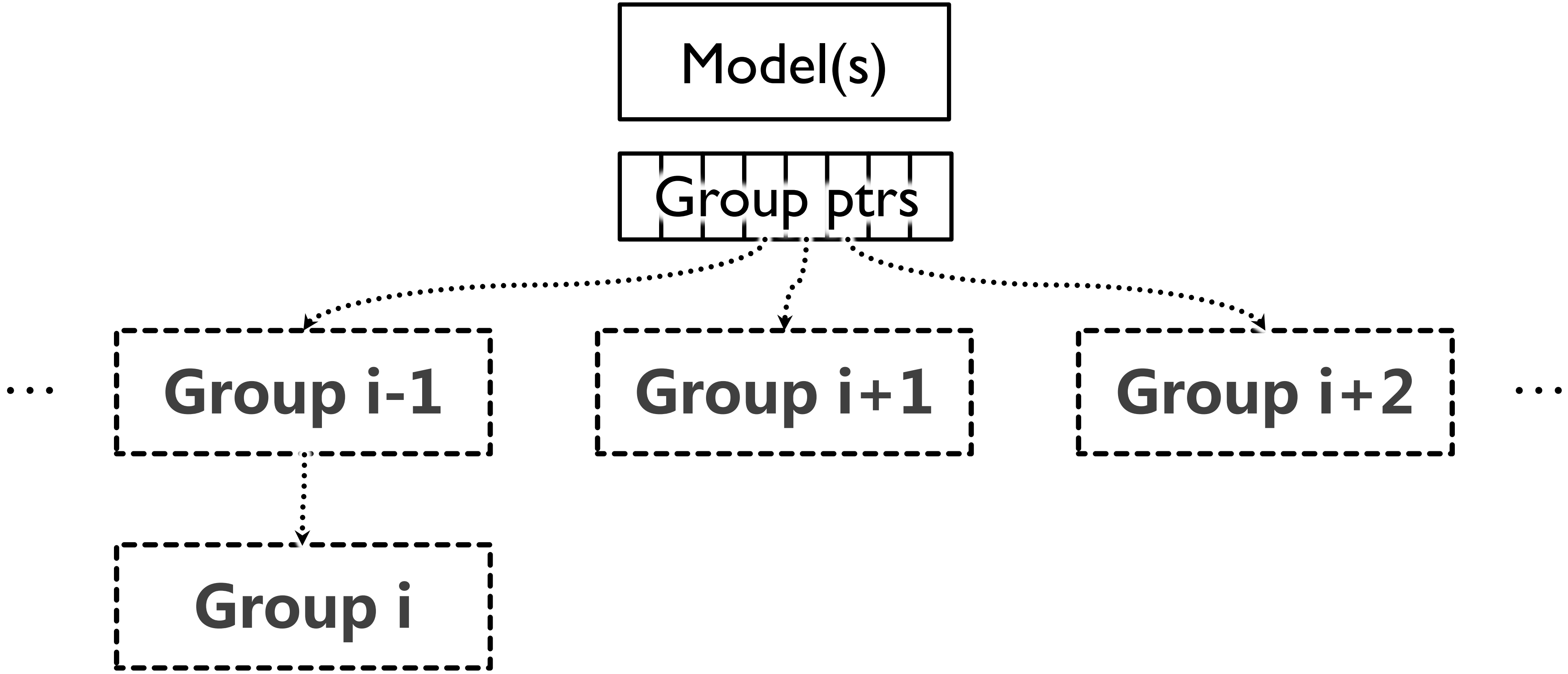
Group Merge
to reduce group #



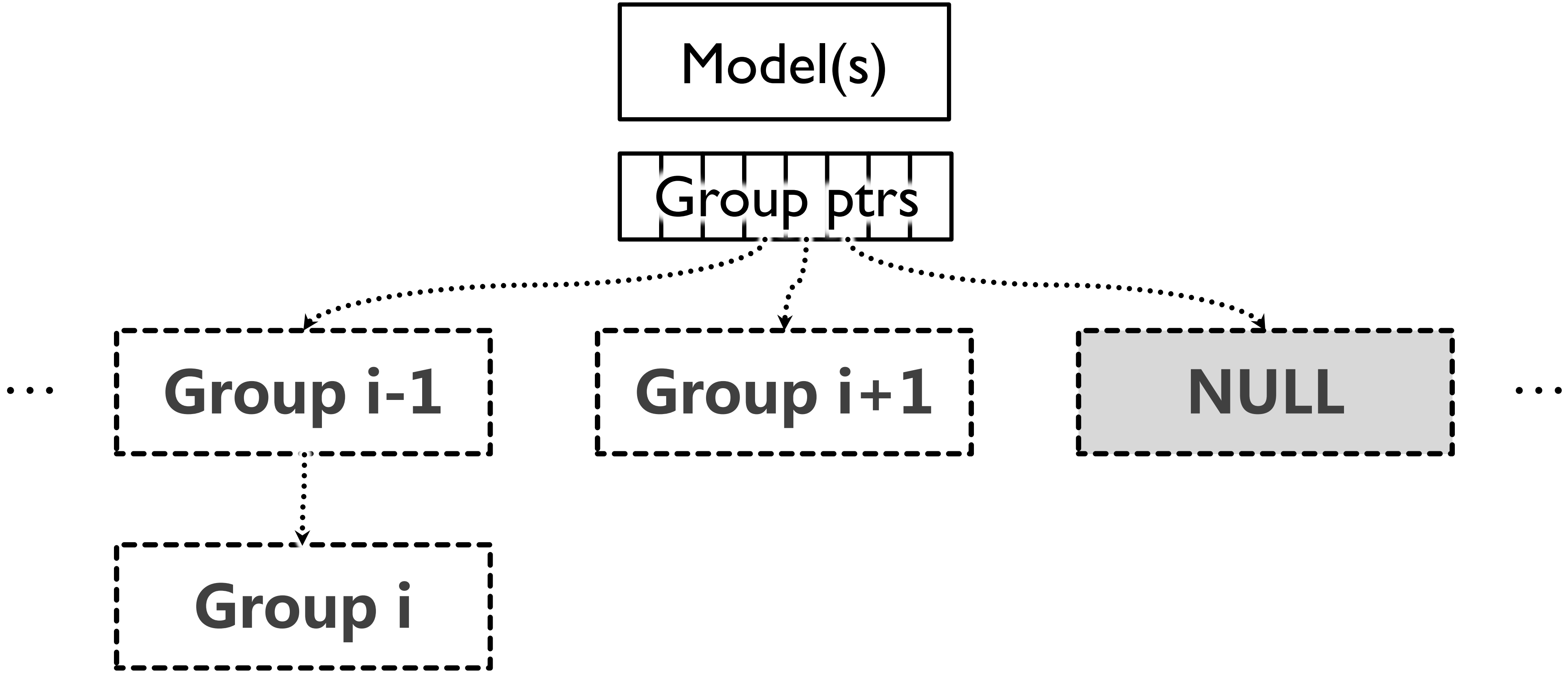
Dynamic workloads: root maintenance



Dynamic workloads: root maintenance



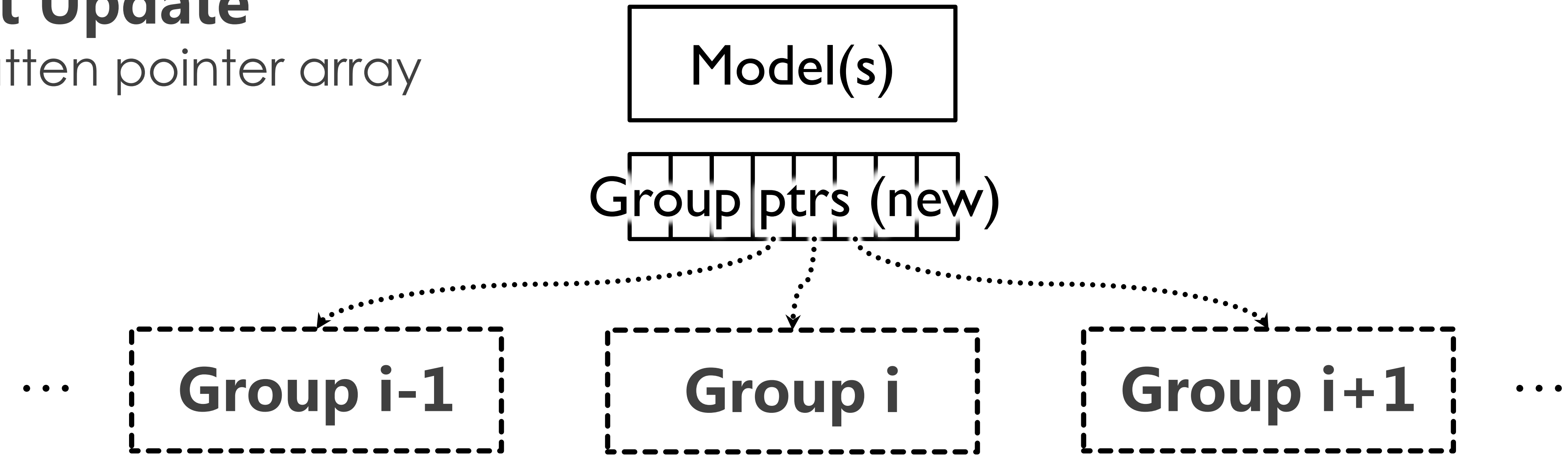
Dynamic workloads: root maintenance



Dynamic workloads: root maintenance

Root Update

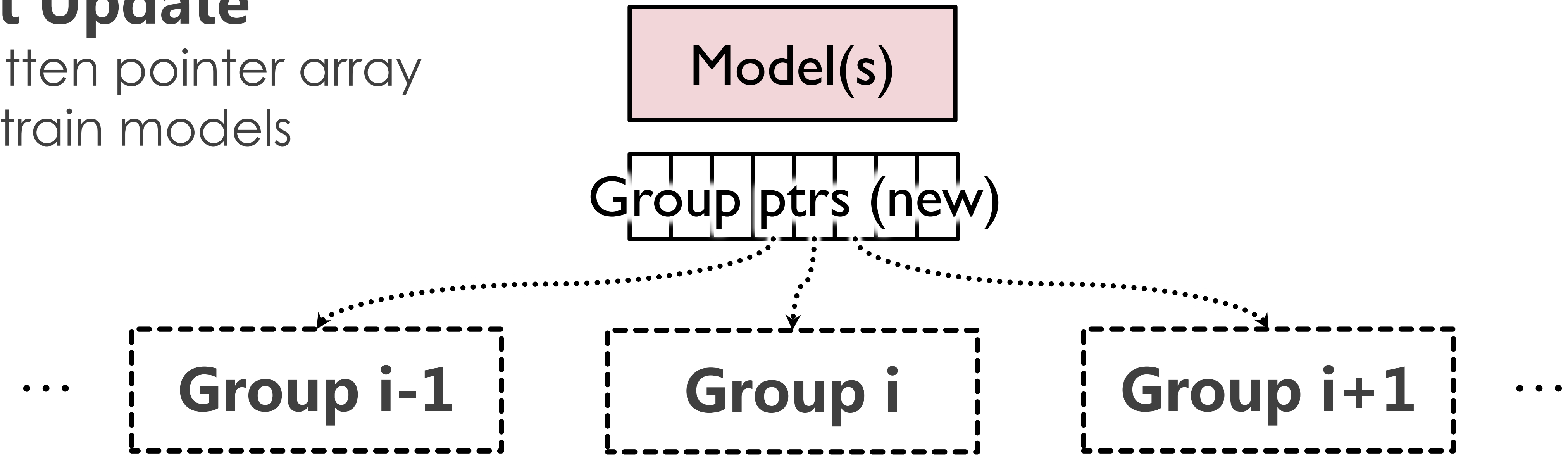
1. Flatten pointer array



Dynamic workloads: root maintenance

Root Update

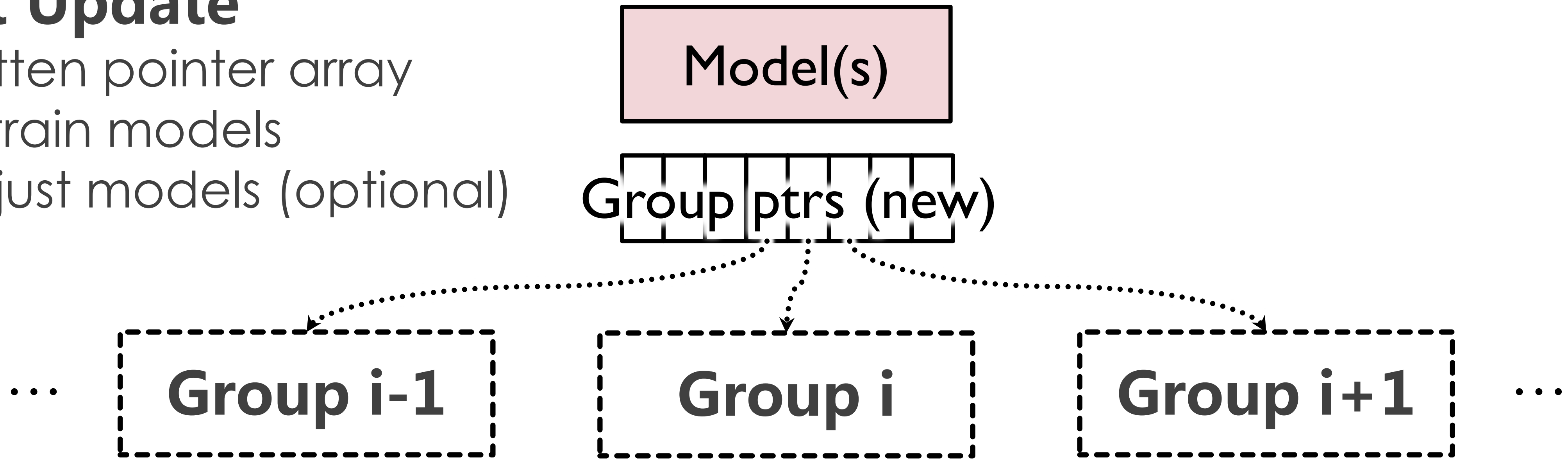
1. Flatten pointer array
2. Retrain models



Dynamic workloads: root maintenance

Root Update

1. Flatten pointer array
2. Retrain models
3. Adjust models (optional)



See the paper for

- **Detailed pseudocode**
- **Fine-grained synchronization protocols**
- **Optimizations**
- **A proof sketch on linearizability**
  - Formal proof in the extended version*
- ……

*https://ipads.se.sjtu.edu.cn/_media/publications/xindex_extended.pdf

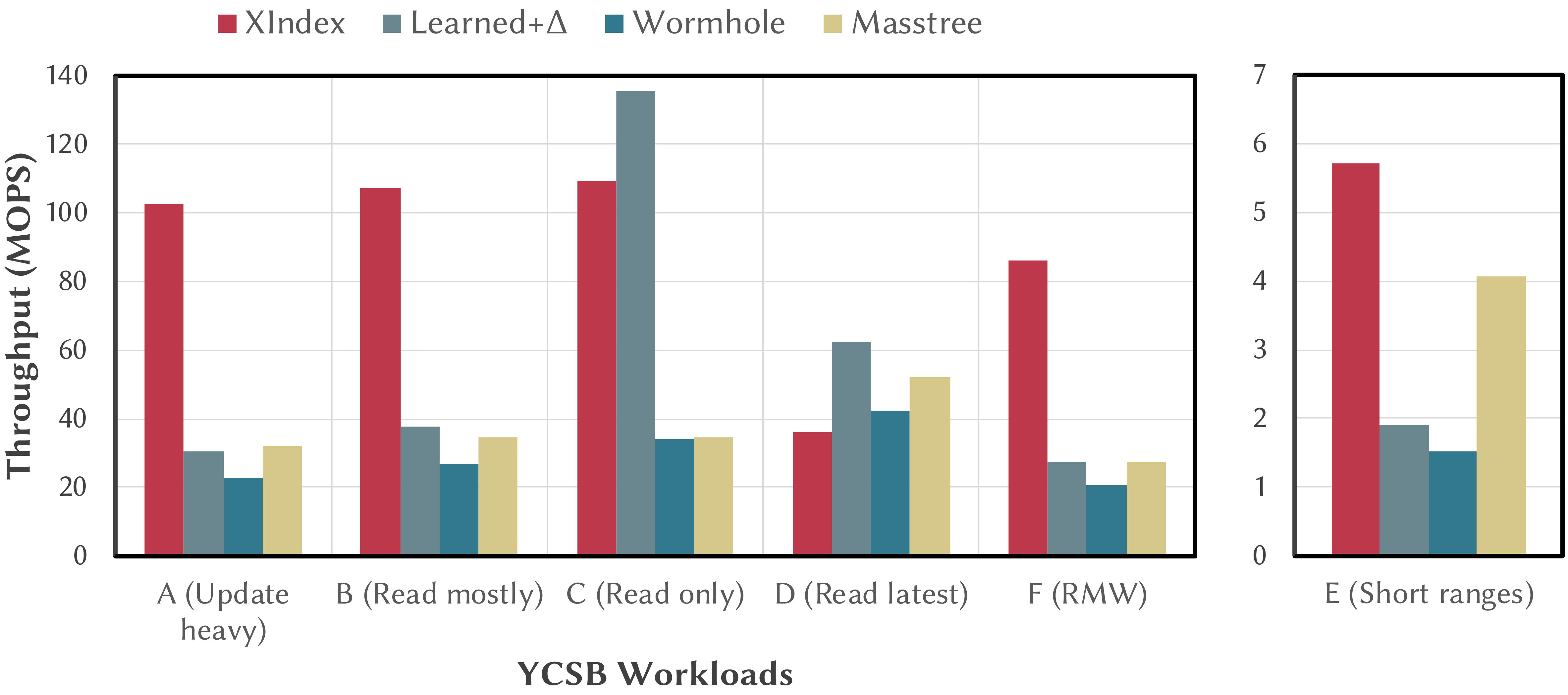
Evaluation

Evaluation Questions

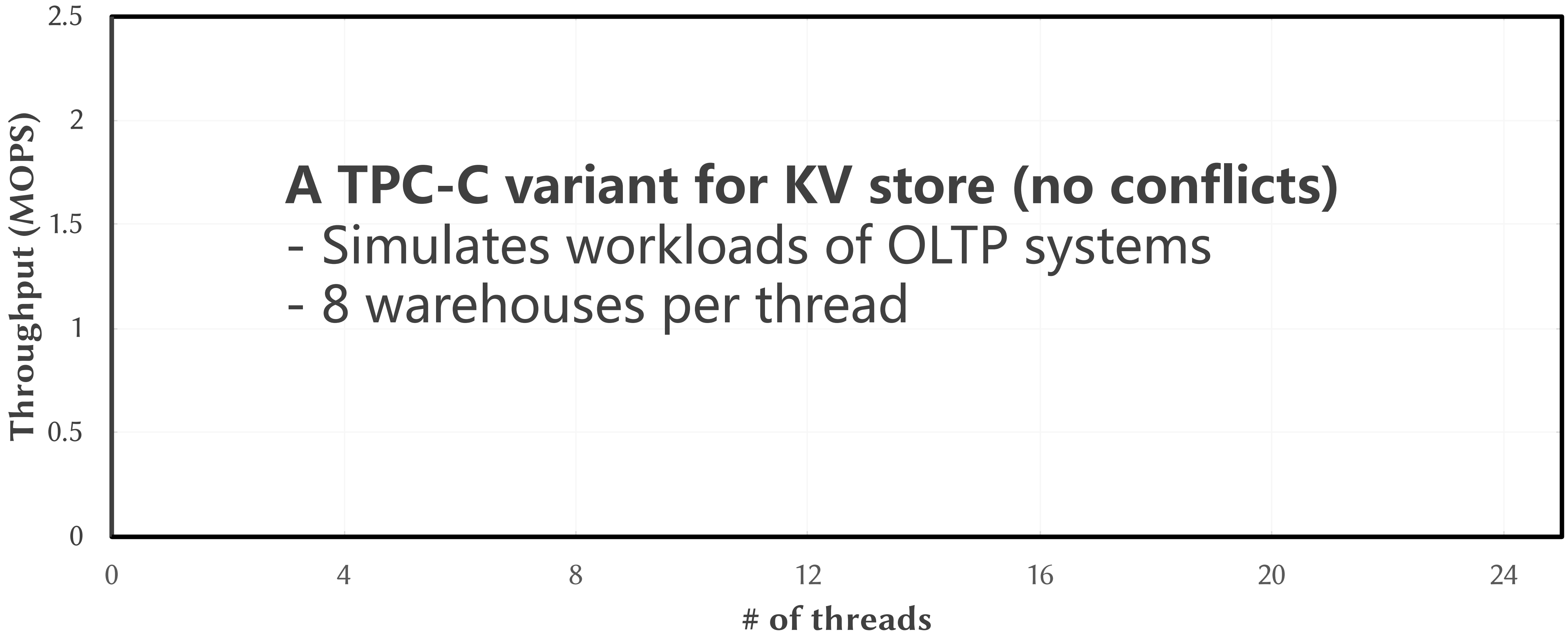
How does XIndex compare with the state-of-the-arts?
Can real systems benefit from XIndex?

- 2 sockets, each has 12 2.20GHz cores; 126GB Ram
- Masstree [EuroSys '12], Wormhole [EuroSys '19], baseline learned index [SIGMOD '18]
- 1:11 background-foreground thread ratio

Throughput in YCSB

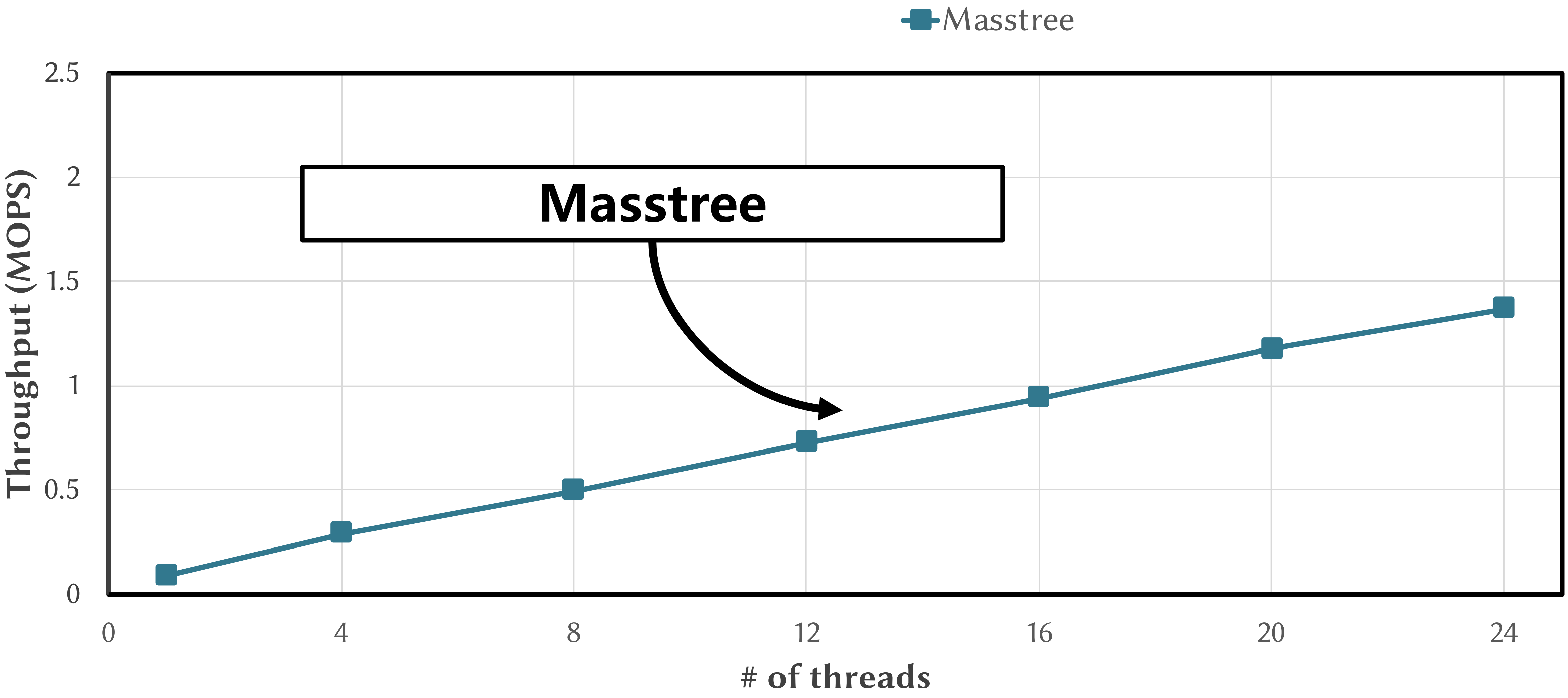


Throughput in TPC-C (KV)

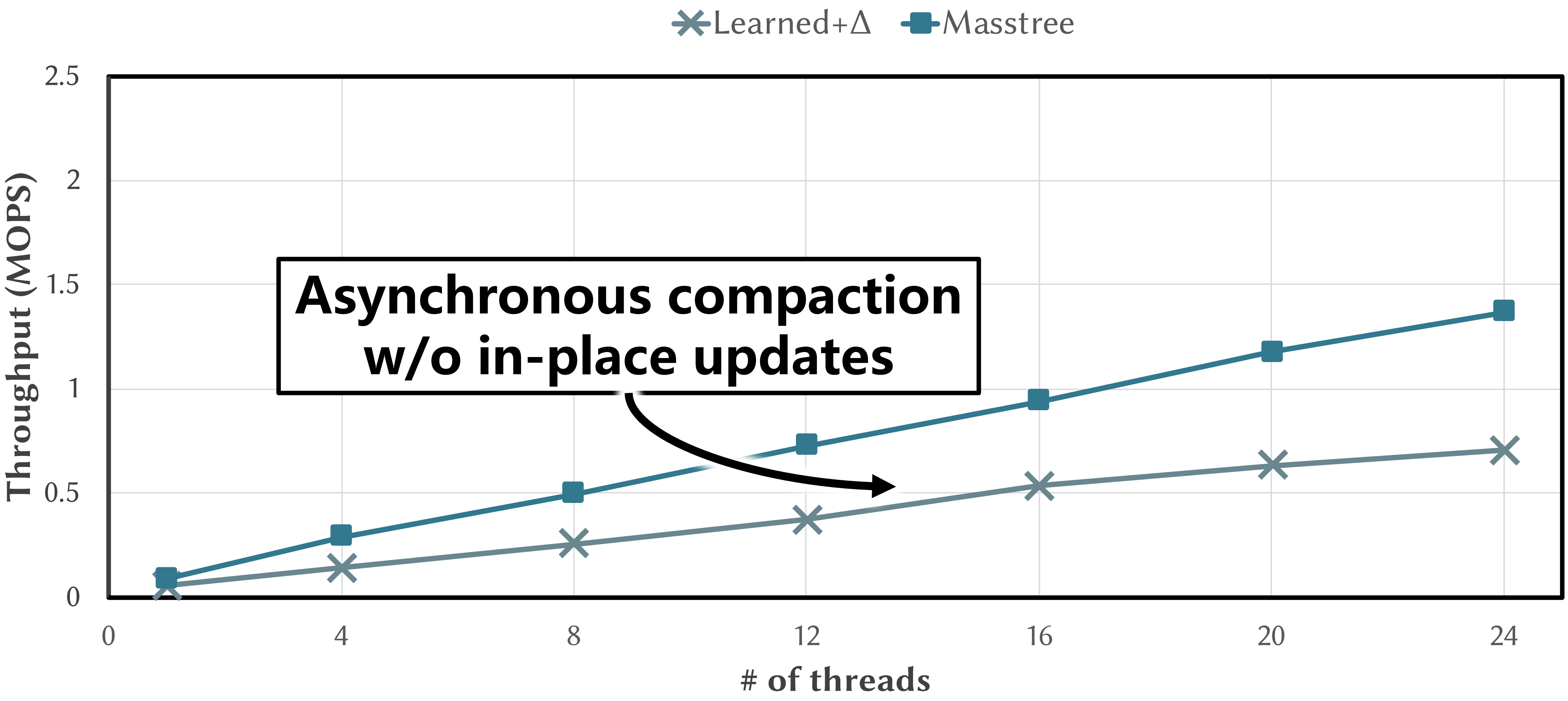


A TPC-C variant for KV store (no conflicts)
- Simulates workloads of OLTP systems
- 8 warehouses per thread

Throughput in TPC-C (KV)

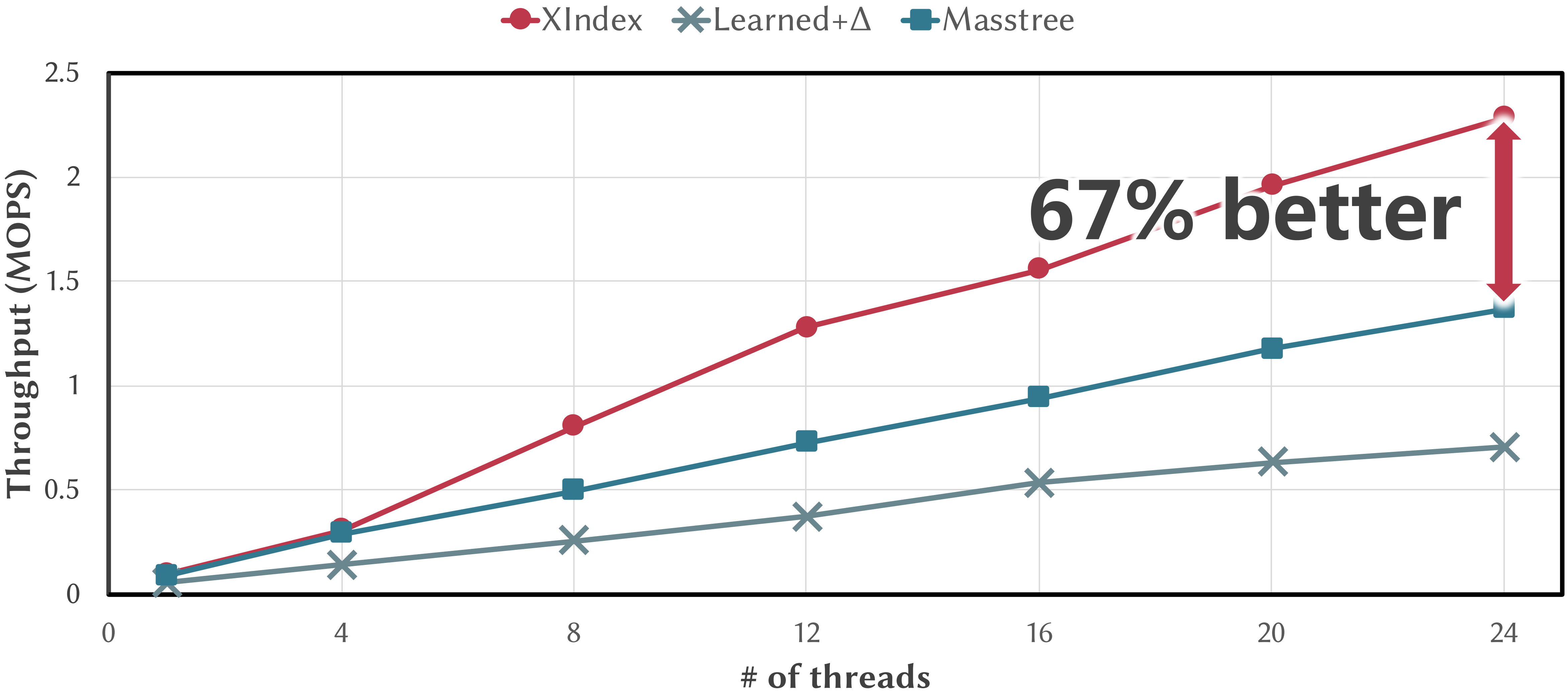


Throughput in TPC-C (KV)



Asynchronous compaction w/o in-place updates

Throughput in TPC-C (KV)



XIndex

- **ML has limitations in data structure design**
- **To make ML work, we need a systematics approach**
  - Two-Phase Compaction for correctness and efficiency
  - Fine-grained Synchronization for scalability
  - Structure Adjustment at runtime for stable performance

Open-sourced at

https://ipads.se.sjtu.edu.cn:1312/opensource/xindex