# RAMCloud Design Review

# Indexing

Ryan Stutsman

April 1, 2010

# Introduction

- **Should RAMCloud provide indexing?**
  - Leave indexes to client-side using transactions?
- **Many apps have similar indexing needs**
  - Or compose standard mechanisms to suit their needs
  - Can optimize for common needs on server-side

# Implementation Issues

- **Indexing on “opaque” data**
- **Partitioning Indexes**
- **Consistency**
- **Recovery/Availability of Indexes**

# Explicit Search Keys

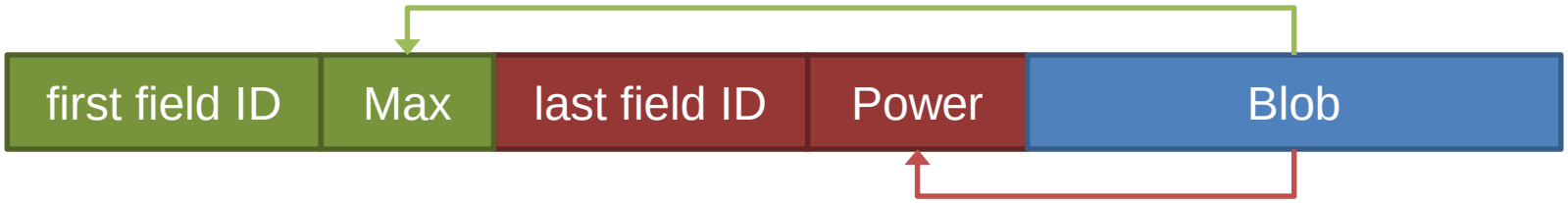
- **Problem: RAMCloud treats objects as opaque**
  - Server-side indexing without understanding the data?

```
put(tableId, person.objectId, person.pickle())
```

# Explicit Search Keys

- **Problem: RAMCloud treats objects as opaque**
  - Server-side indexing without understanding the data?
- **Idea: Apps provide search keys explicitly**
  - Apps understand the data

```
put(tableId, person.objectId, {'first': person.first,
                               'last': person.last},
                              person.pickle())
```



# Explicit Search Keys

- **Problem: RAMCloud treats objects as opaque**
  - Server-side indexing without understanding the data?
- **Idea: Apps provide search keys explicitly**
  - Apps understand the data

```
put(tableId, person.objectId, {'first': person.first,
                               'last': person.last},
                              person.pickle())
```


- **Can eliminate redundancy**
  - Search keys need not be repeated in object
  - Search keys + Blob are returned to app on get/lookup

# Explicit Search Keys

- **Lookups are distinct from gets**

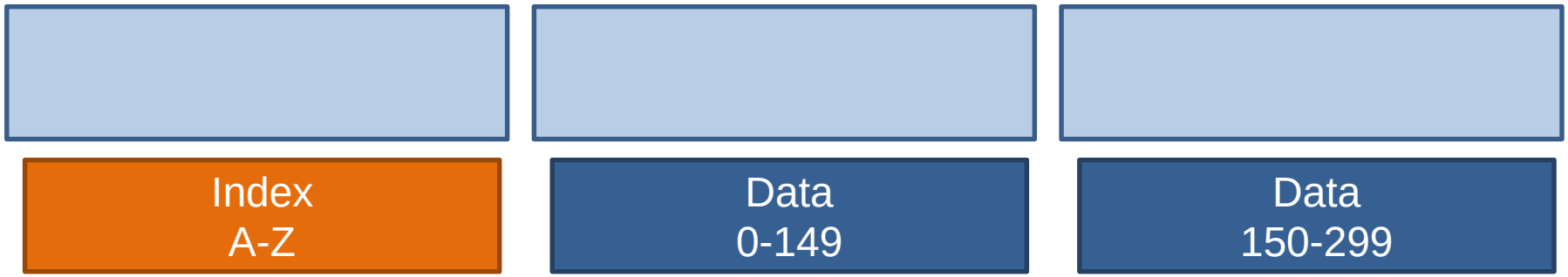
```
lookup(tableId, 'last', 'Power')
```

- **Put atomically updates indexes and object**
  - Details to follow

# Partitioning Indexes

- **Co-locate index and data**


- **Large tables?**
- **Large indexes?**
  - Can’t avoid multi-machine operations



# Partitioning Indexes

- **Split indexes on search key**


  - One extra access per lookup and put

- **Split indexes on object ID**


  - Lookups go to all index fragments
  - Puts are always local
  - Ordered enumeration of the index is problematic

# Partitioning Indexes: Thoughts

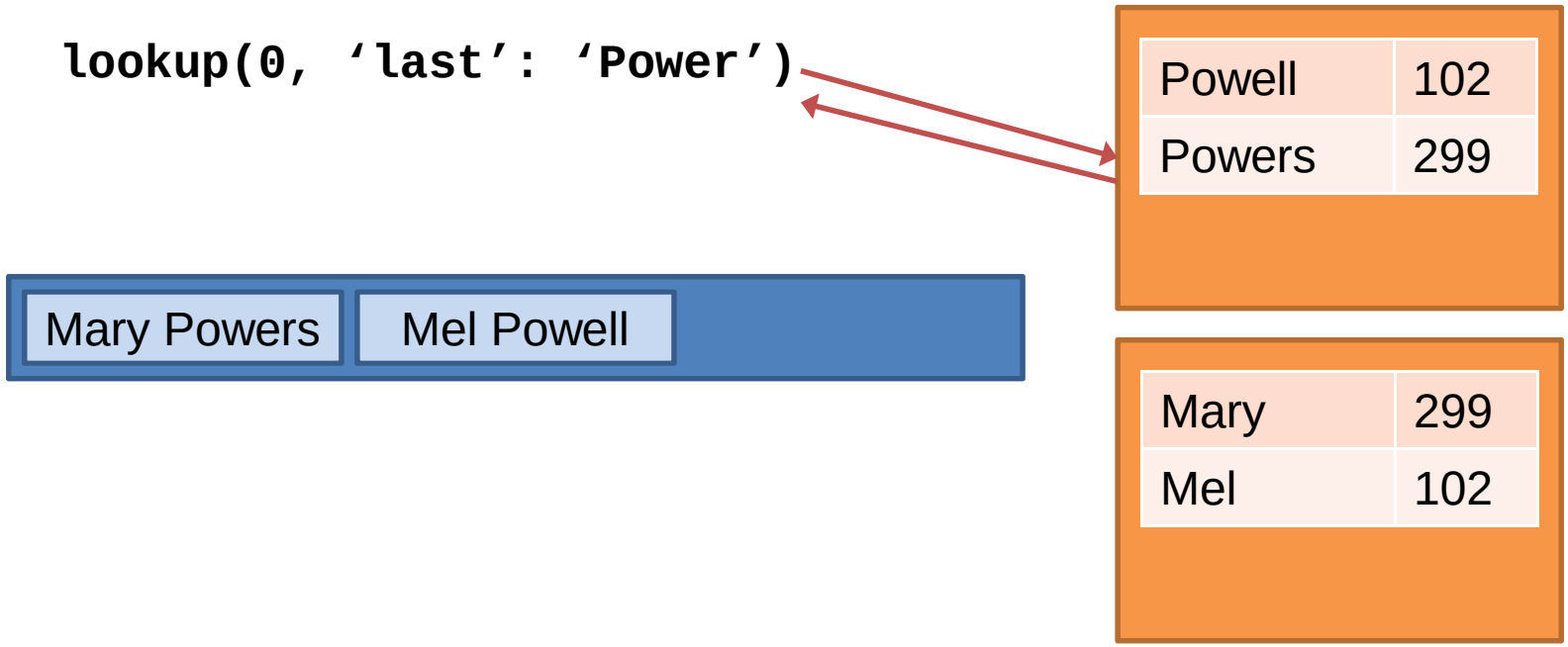
- **Our decision (for now): On search key**
  1. Don’t want weakest-link lookup performance
  2. To support enumerate and cursors for range queries

# Consistency

- **Problem: Index/Object inconsistency on puts**
  - Since object and index may reside on different hosts
  - Apps may see index entries for objects not yet written
- **Avoid fancy commit protocol, if possible**
- **Idea: Index entries “commit” on object put**
  - Object puts are atomic
  - Index entries invalid until corresponding put finishes

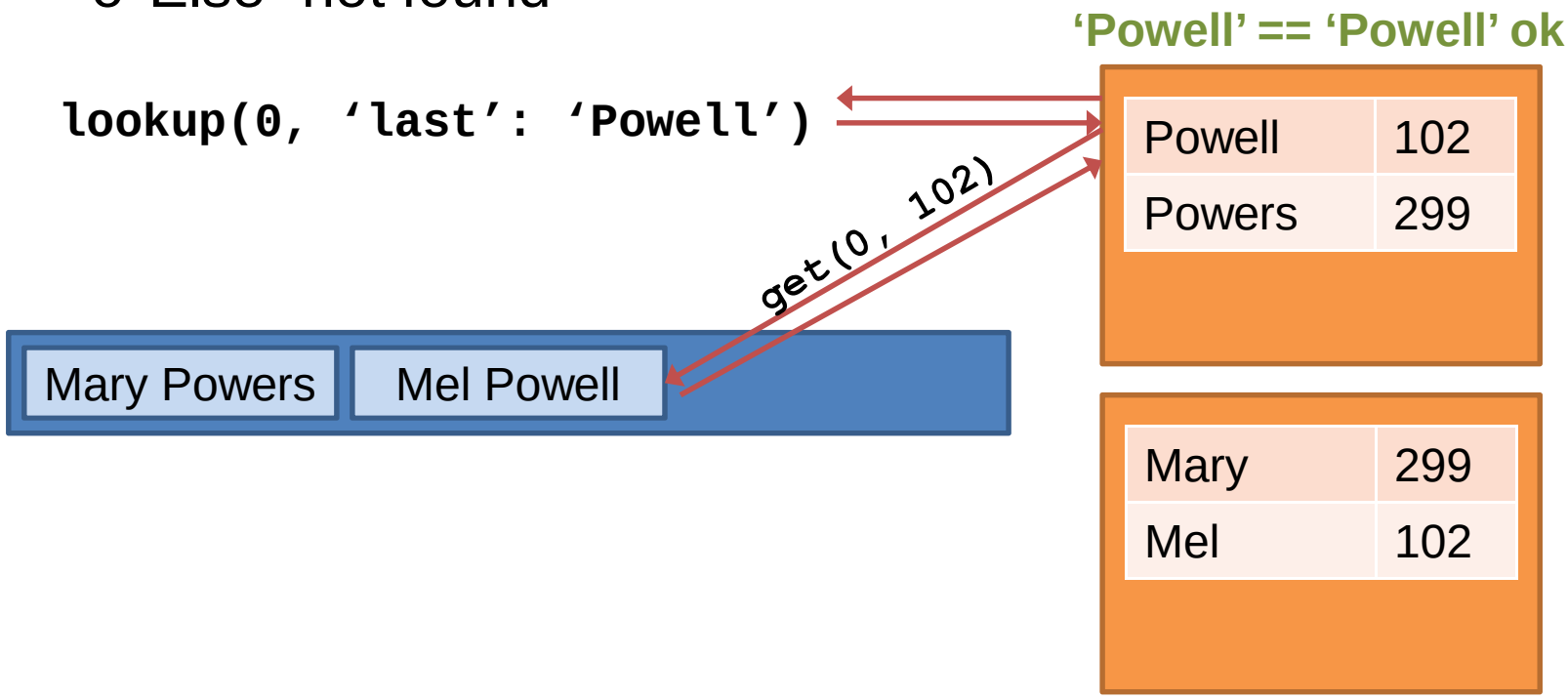
# Consistency: Lookup

- **Request goes directly to correct index partition**
  - “Not found” returns immediately



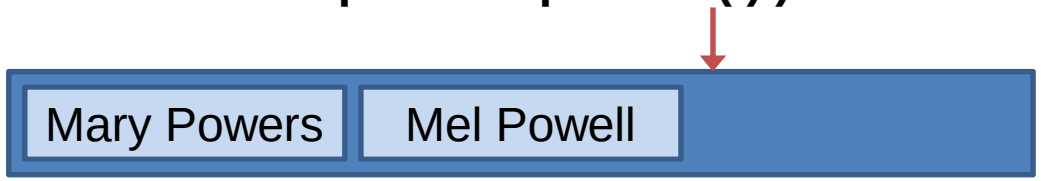
# Consistency: Lookup

- **Consistency is checked on hit**
  - If table and index agree the return the object
  - Else “not found”



# Consistency: Create

```
put(0, 301, {‘first’: ‘Max’,
             ‘last’: ‘Power’},
          person.pickle())
```

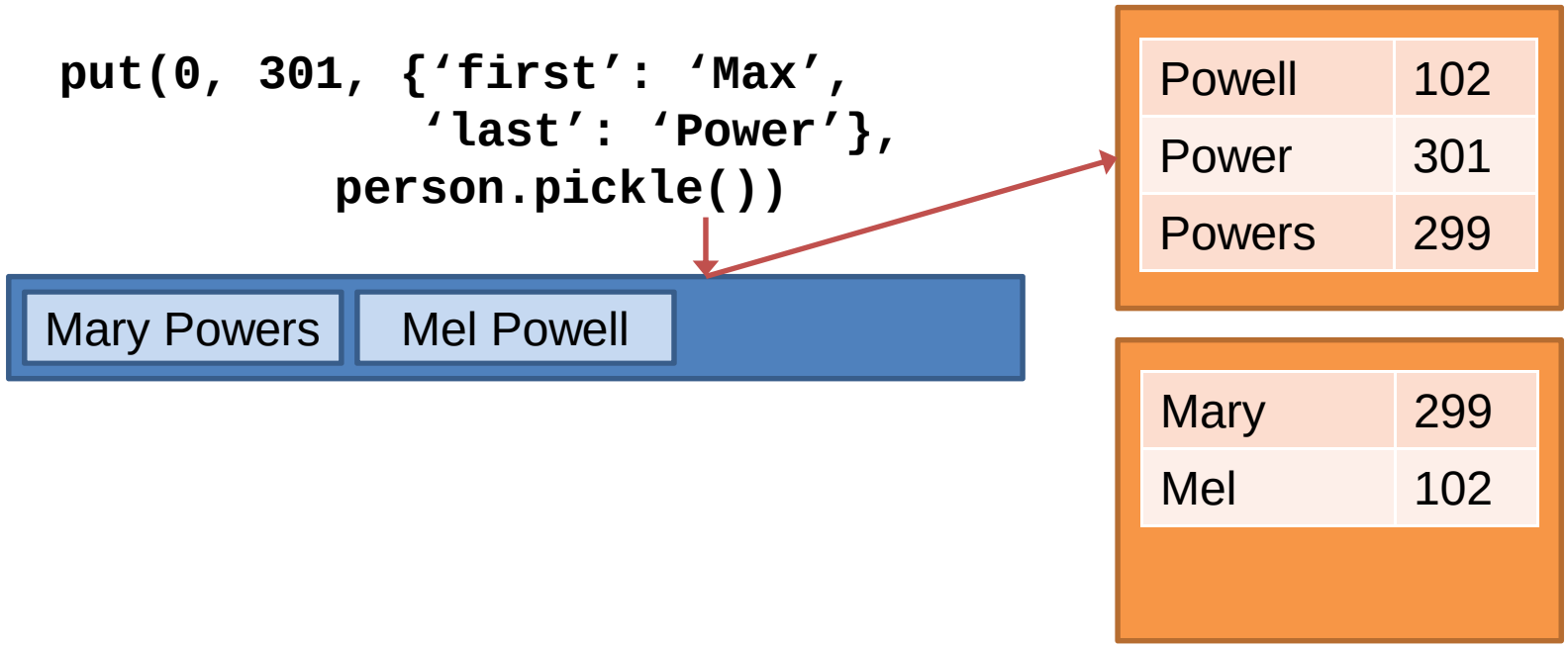



| | |
|---|---|
| Powell | 102 |
| Powers | 299 |

| | |
|---|---|
| Mary | 299 |
| Mel | 102 |

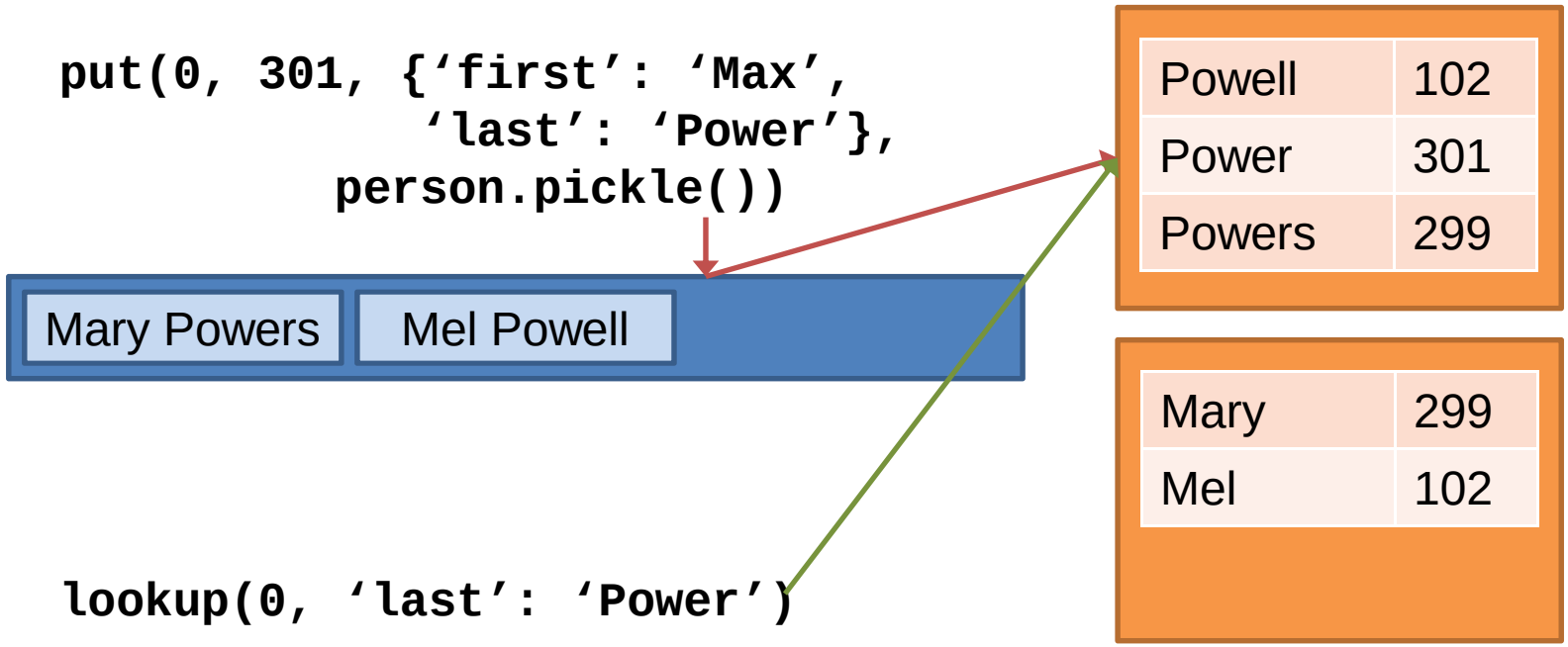
# Consistency: Create

- **Insert index entries before writing object**
  - What happens if a lookup happens in the meantime?



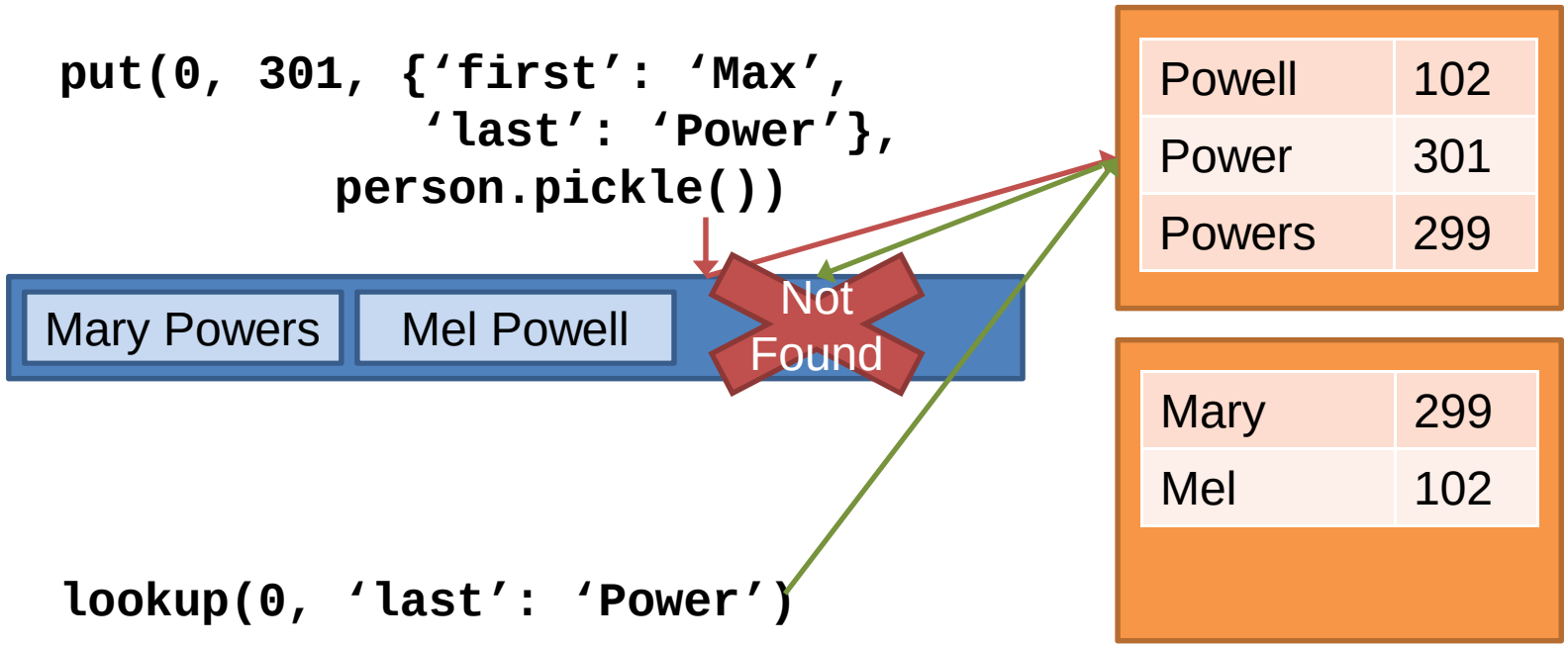
# Consistency: Concurrent Lookup

- **Concurrent ops ignore inconsistent entries**



# Consistency: Concurrent Lookup

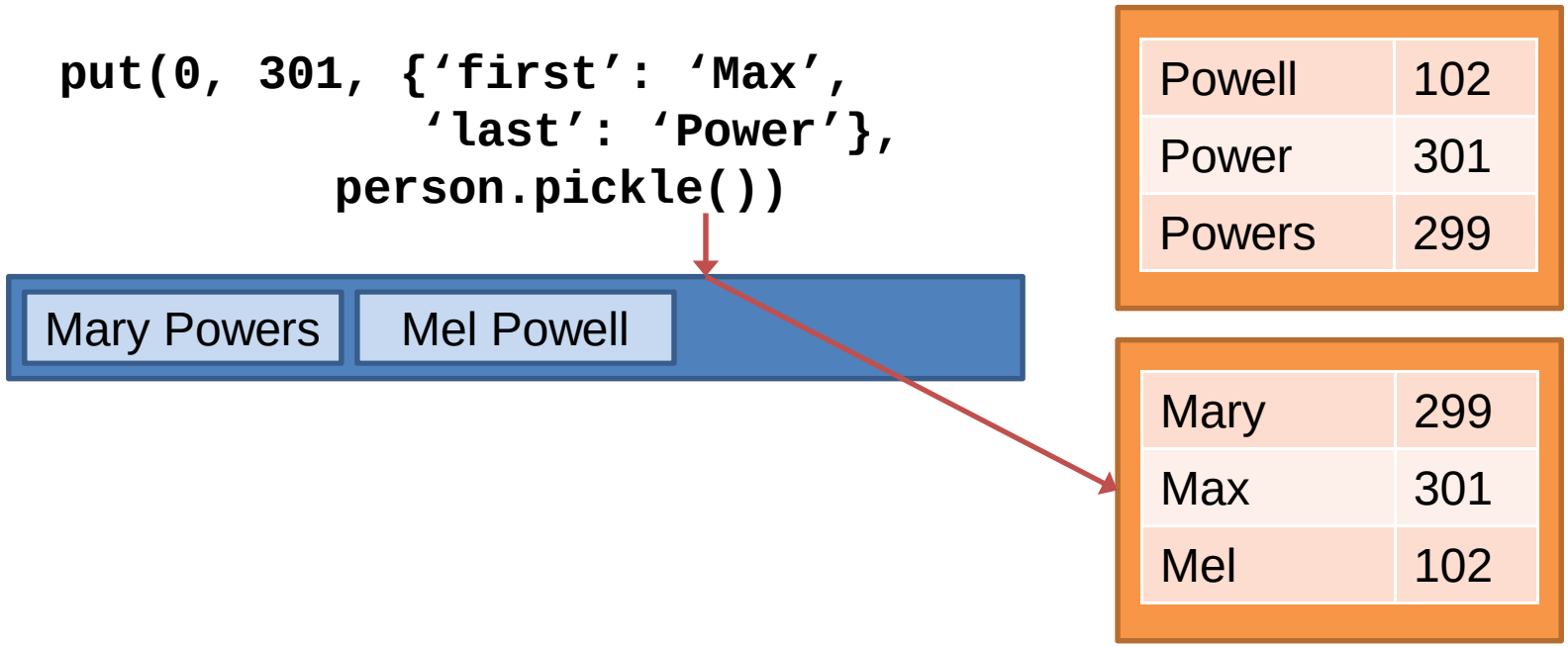
- **Concurrent ops ignore inconsistent entries**



# Consistency: Create (continued)

- **Insert index entries before writing object**

```
put(0, 301, {'first': 'Max',
             'last': 'Power'},
            person.pickle())
```



# Consistency: Create

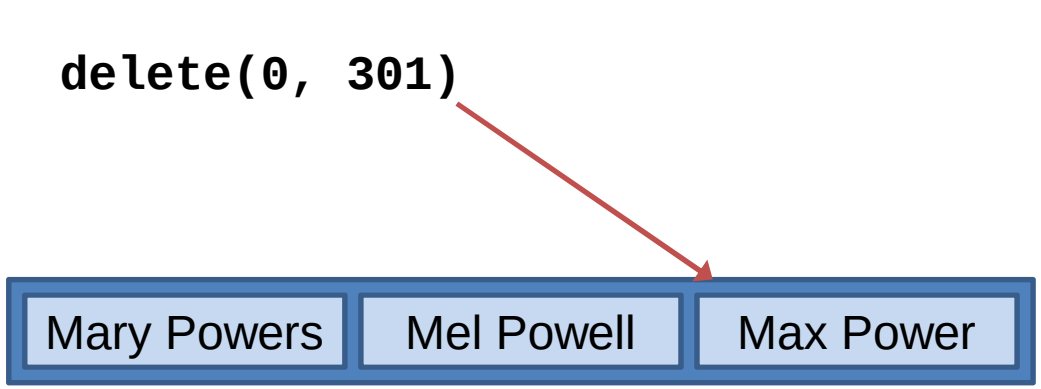
- **Put completes; index entries now valid**

```
put(0, 301, {‘first’: ‘Max’,
             ‘last’: ‘Power’},
          person.pickle())
```


| | |
|---|---|
| Powell | 102 |
| Power | 301 |
| Powers | 299 |

| | |
|---|---|
| Mary | 299 |
| Max | 301 |
| Mel | 102 |

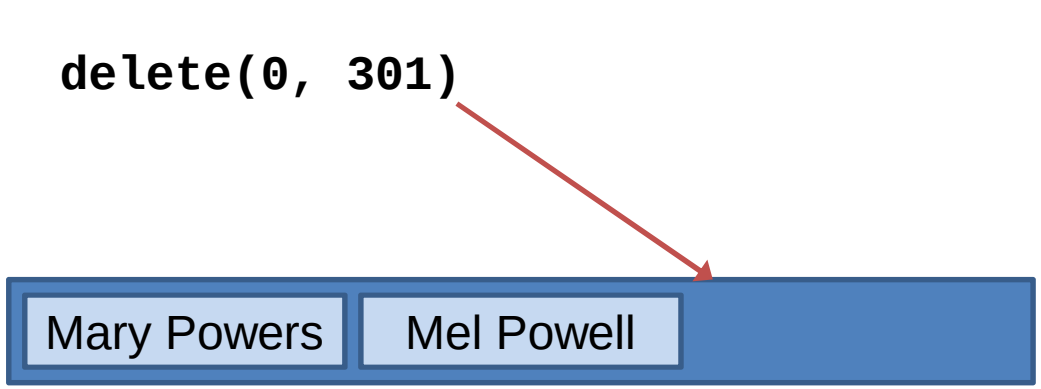
# Consistency: Delete




| | |
|---|---|
| Powell | 102 |
| Power | 301 |
| Powers | 299 |

| | |
|---|---|
| Mary | 299 |
| Max | 301 |
| Mel | 102 |

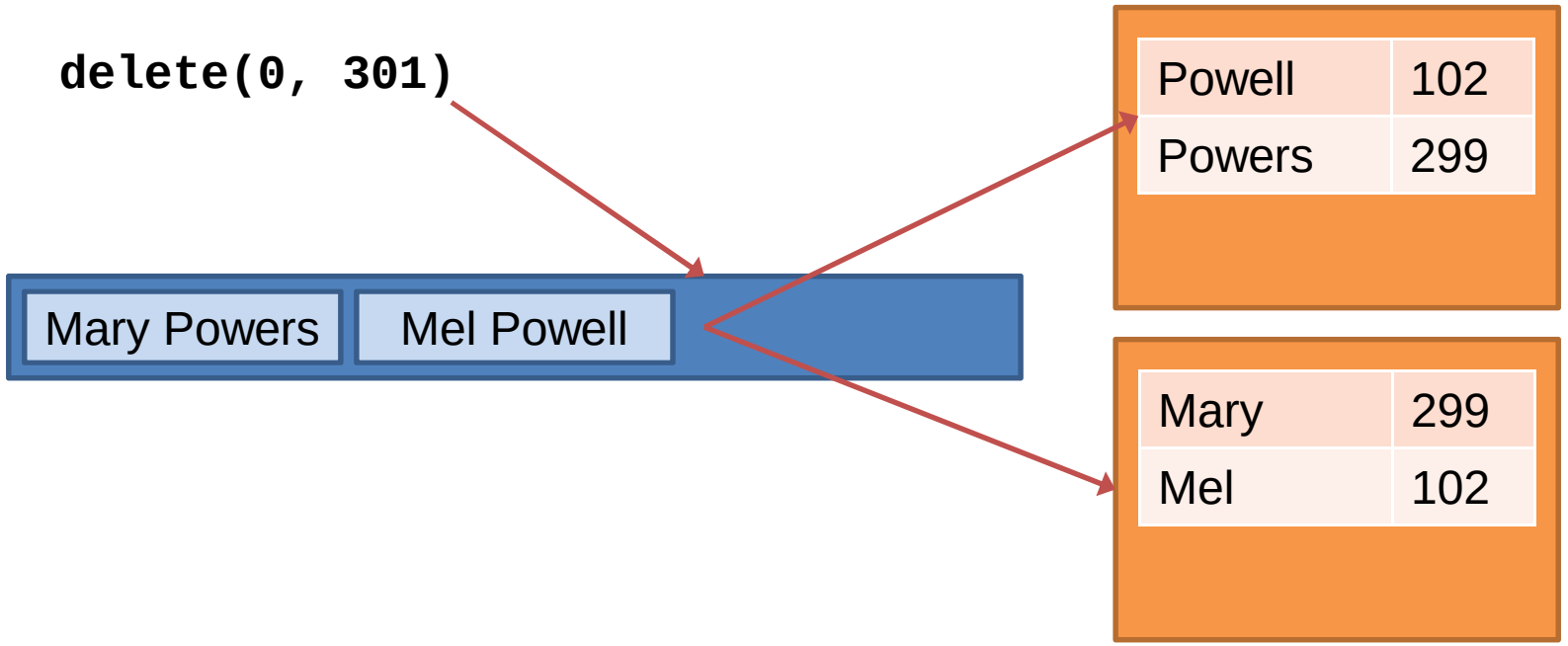
# Consistency: Delete




| | |
|---|---|
| Powell | 102 |
| Power | 301 |
| Powers | 299 |

| | |
|---|---|
| Mary | 299 |
| Max | 301 |
| Mel | 102 |

# Consistency: Delete



# Consistency: Update

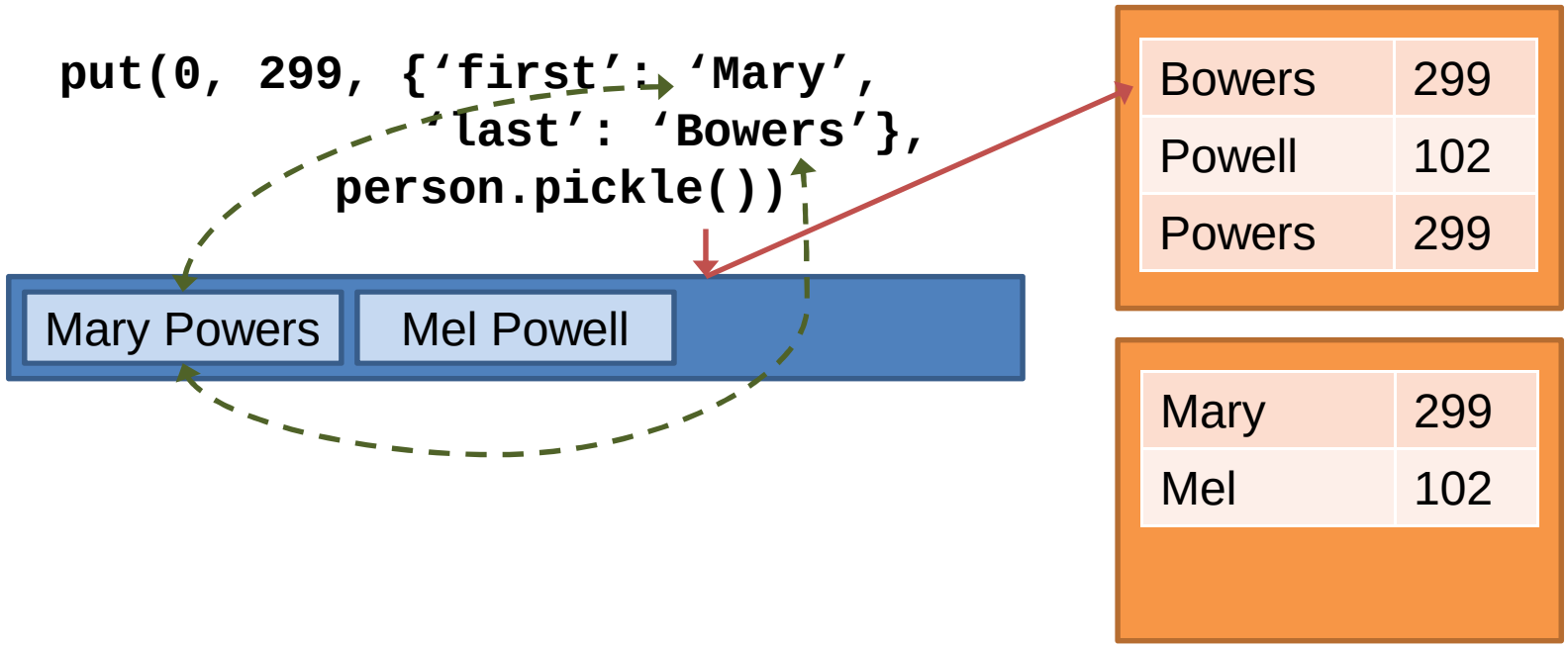
```
put(0, 299, {‘first’: ‘Mary’,
             ‘last’: ‘Bowers’},
            person.pickle())
```


| | |
|---|---|
| Powell | 102 |
| Powers | 299 |

| | |
|---|---|
| Mary | 299 |
| Mel | 102 |

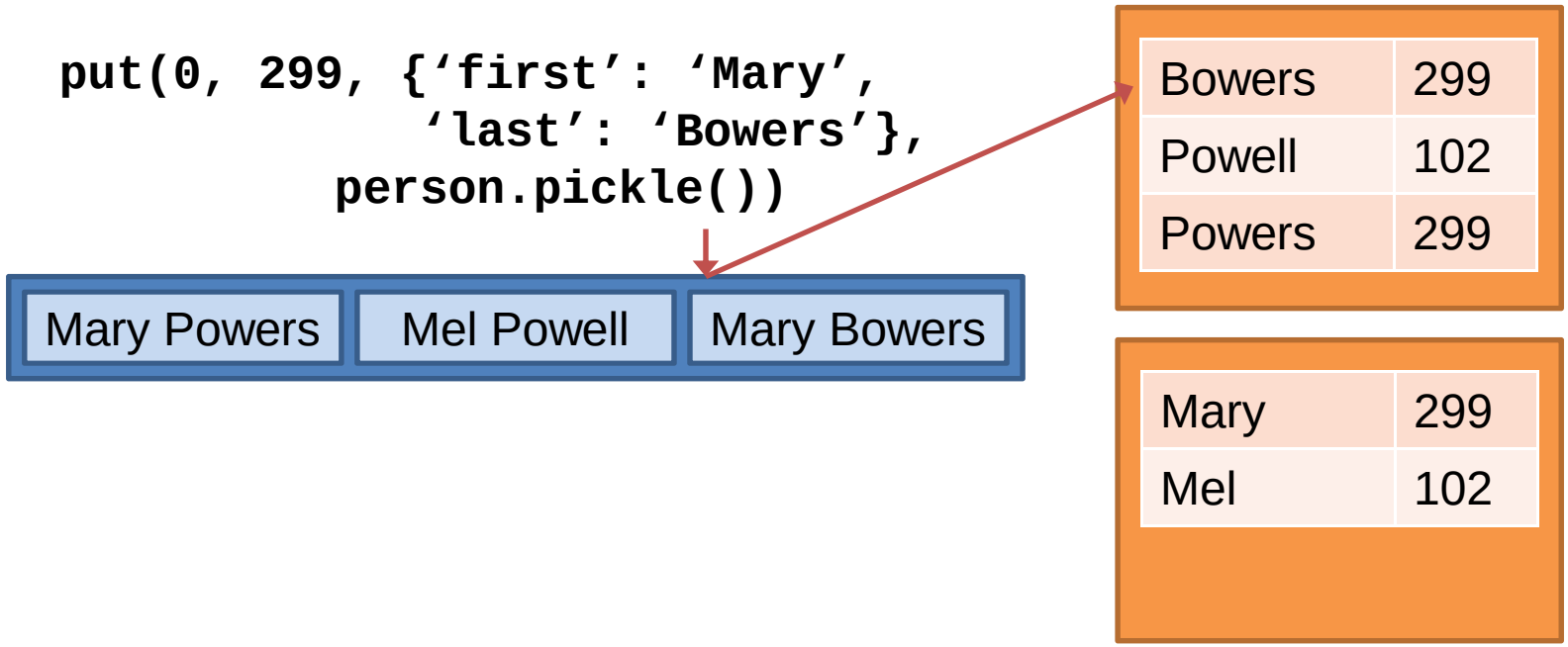
# Consistency: Update

- **Compare previous index entries**
  - Insert new value if updated



# Consistency: Update

- **Commit by writing the new value**
  - Old index entries ignored by lookup since inconsistent



# Consistency: Update

- **Cleanup old, inconsistent entries**

```
put(0, 299, {‘first’: ‘Mary’,
             ‘last’: ‘Bowers’},
          person.pickle())
```


| | |
|---|---|
| Bowers | 299 |
| Powell | 102 |

| | |
|---|---|
| Mary | 299 |
| Mel | 102 |

# Consistency: Thoughts

- **Low-latency gives simplified consistency**
- **Turn atomic puts into atomic index updates**
  - All index updates for an object go through master
  - Index entries invalid until corresponding put completes

# Index Recovery

- **Problem: Unavailable until indexes recover**
  - Many requests will be lookups
  - These will block unless indexes are recovered
- **Rebuild from other masters?**
  - TODO why this fails
  - TODO Doesn’t fail with sharding?
- **Rebuild from backups?**
  - TODO

# Index Recovery: Sharding

- **Split on object ID**
  - Can always co-locate index with data
  - Index chunk at most 320 MB
  - Each new master can rebuild in a fraction of a second
- **Split on search key**
  - Entire shards composed only of index data
  - At most 640 MB apiece
    - 0.6s to gather data, fraction of a second to rebuild
  - Part or all of 640 MB may come from shards in recovery
    - 0.6s + 0.6s = 1.2s upper bound

# Index Recovery: Replication

- **Idea: Replicate indexes once in RAM**
  - Threat is only to availability, not data loss
- **Idea: Only preserve the shape of the index**
  - The search keys are stored in the log
- **TODO**

# Index Recovery: Logging

- **TODO**

# Summary

- **Apps provide search keys explicitly on put**
- **Partition indexes on search key for easy lookup/enumeration**
- **Atomic indexes from atomic puts**
- **Fast index recovery for high-availability**