Tablet Profiling in RAMCloud

Steve Rumble
Stanford University

SEDCL Annual Retreat June 4, 2011

Recall: RAMCloud Data Model

- **RAMCloud stores objects in *tables***
  - Tables may span physical machines (if too large, or too hot)
  - A contiguous range of a table is called a *tablet*

- **Masters are responsible for serving *tablets***
  - Example:

Table #	First Key	Last Key
12	0	2^{64} - 1
47	63,742	5,723,742

- **Clients obtain a tablet map from the coordinator**
  - Associates tablets with the master servers that own them

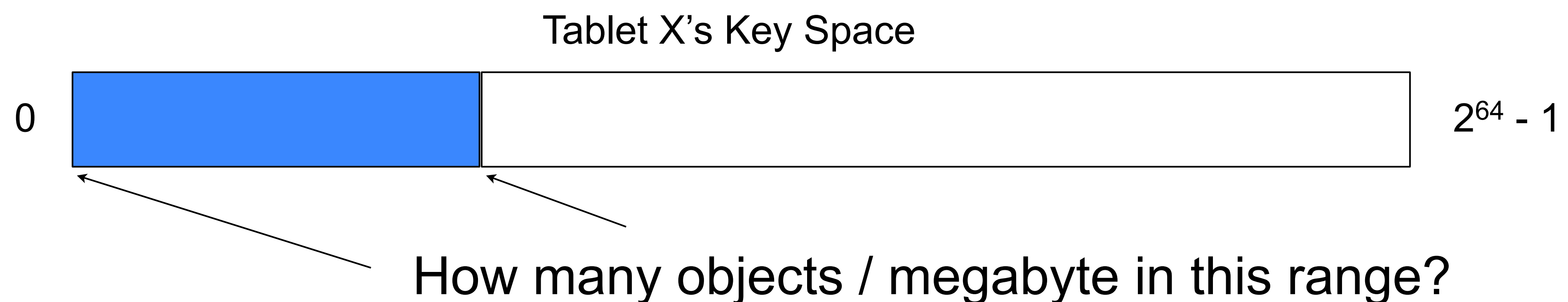
Even Distribution for Fast Recovery

- **When failures occur, need to recover quickly (1-2 sec)**

- **Problem:**
  - Recovery time is dependent on the amount of data each recovery master recovers
- **Can only recover about 600MB in 1-2 seconds**
  - 10GigE: ~1250MB/s max
  - 600MB in, 1200MB out (R = 2, without multicast)
- **Per-object CPU overheads also need to be balanced**

Dividing Tablets is Tricky

- **How does a server evenly divide its tablets / objects?**
- **Can easily count how many objects and bytes are in each tablet, but what about big tablets?**
  - Need to split them. But where to make the cut?
- **Must ensure each division isn't too big**
  - Cut the key space so each subrange contains <= 600MB of objects
- **Difficult because users store variable-sized objects wherever they want in the 64-bit space**
  - Densities are unpredictable
  - Key density needn't imply byte density



Solution 1: Randomness

- **Random distribution of keys should yield a uniform density of bytes**
  - E.g.: Consistent hashing
  - Key used by RAMCloud becomes hash(TableId, ObjectId)
- **Equal key subranges should have similar byte counts**
  - TabletSize / 600MB = number of equal splits to make
- **Problems with this idea**
  - Current design allows implicit locality of data based on locality of keys (objects in same table and nearby in key space are co-located)
  - Locality is useful for multi-object actions like range queries

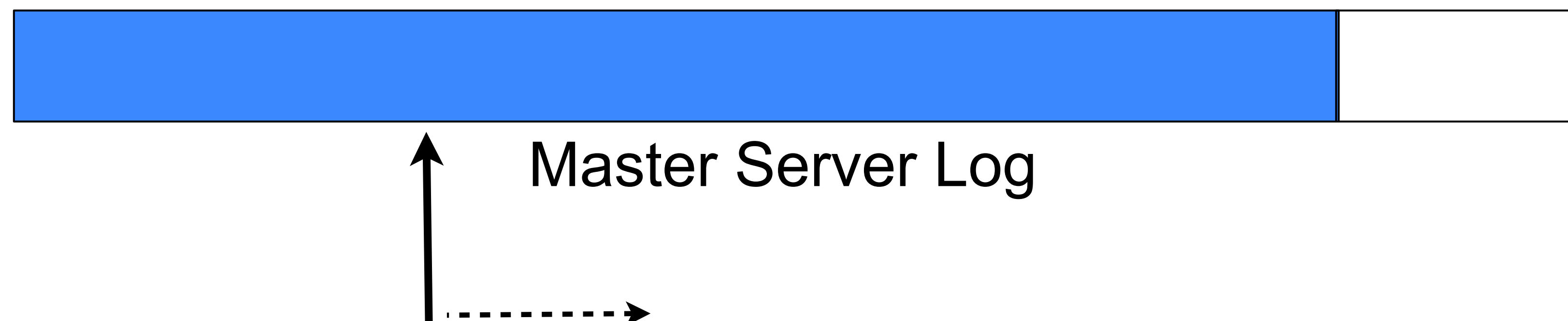
Prefer to Keep Tablet Model

- **How can we have our contiguous tablet ranges and still make recovery fast?**
- **Need some way to efficiently partition big tablets**
  - And combine tablets that are small
- **Let’s consider a few other options...**

Solution 2: Batch

- **Compute partitions in batch**
  - Masters periodically scan all of their data and determine reasonable partitions for their heirs

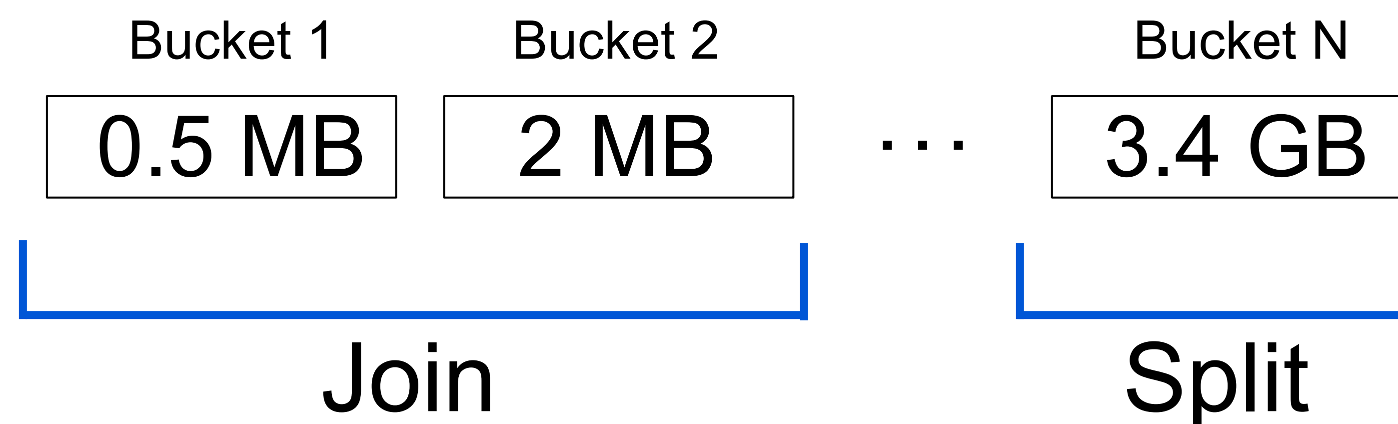
- **Multi-pass algorithm:**
  - Keep byte counts for N subranges
  - Walk log, updating counts
  - Split subranges that are too big, merge those that are too small



Tablet	Bucket 1	Bucket 2	Bucket 3	...	Bucket N
57	374MB	0	0	...	0
12	7MB	31MB	17MB	...	45KB

Solution 2: Batch

- **After one pass, have byte and object counts for each bucket of the key space**
- **Can join small buckets, but must split large ones into new buckets**
- **But, when we split, no idea where to draw the line. So, must break big buckets into another N**
  - Another pass is needed to update counts



Solution 2: Batch

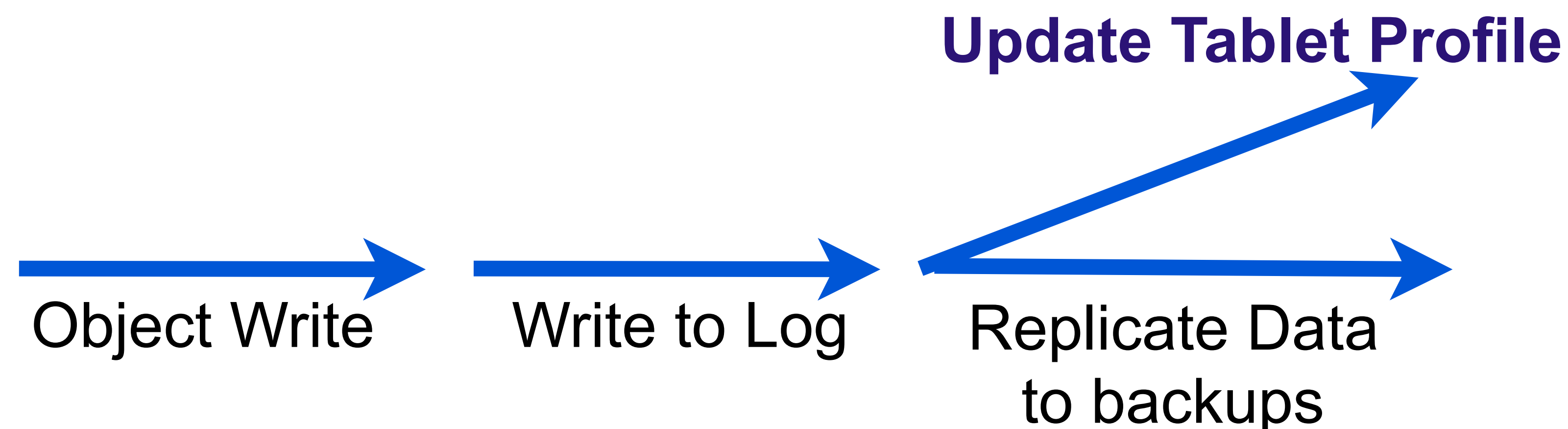
- **Batch is too expensive**

- **Need to be able to recalculate partitions fast, since they can grow at near-network speed (~500MB/s).**
  - Imbalance => longer recovery
  - Cannot afford seconds to recompute

- **64GB's worth of objects on a single server today**
  - May be 100s of millions of objects
  - And that's just for one pass (forget multiple passes)

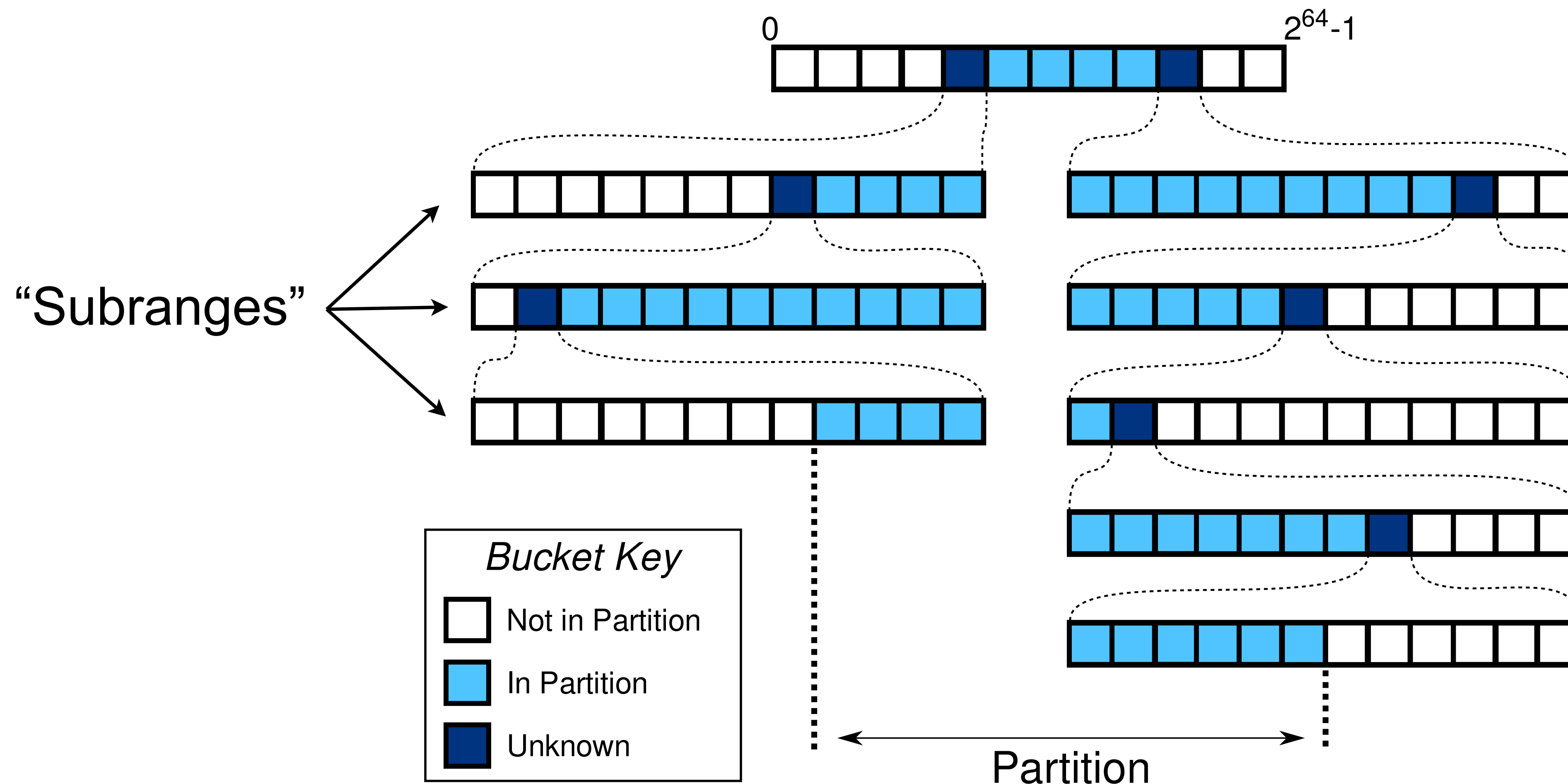
Solution 3: Online

- ***Tablet Profiler***
  - Profile each tablet to track the space usage
  - Do it online (i.e. during writes, while cleaning log)
  - Profiler can determine data density with bounded error
  - 2 operations maintain the data structure:
    - Tablet->Profiler->Track(Key, Bytes, Time) -- log writes
    - Tablet->Profiler->Untrack(Key, Bytes, Time) -- log cleaning
  - Can overlap tablet profile update with data replication
    - ~5 microsecond delay to hear back from backups



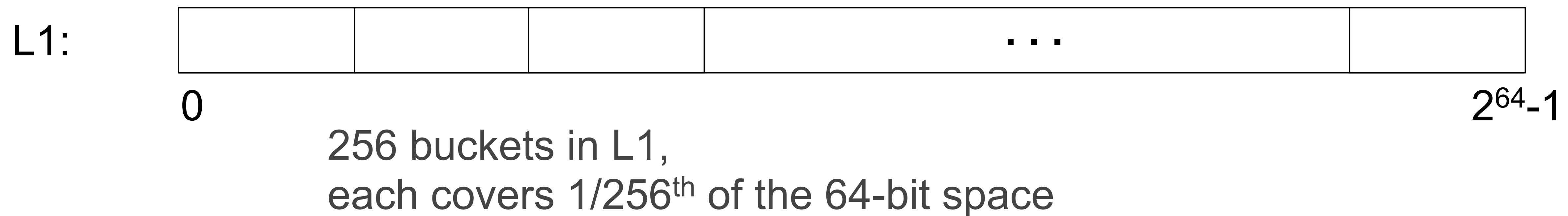
Tablet Profile Structure

- **Tree structure - of key ranges and buckets**
  - Somewhat like a page table structure.
  - Parameters:
    - *B*, the number of bits per level (Affects number of buckets)
    - *S*, maximum bytes in a bucket before splitting
- **Each level "zooms in" on a subrange of the key space. Buckets keep byte tallies.**



Tablet Profile Structure

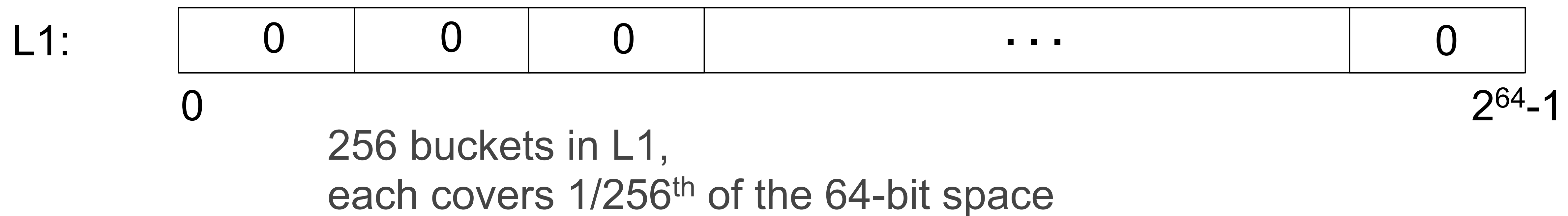
- **Example: B = 8, S = 8MB**



- **With B = 8, there are up to 8 levels**
  - NumLevels = ceil(64 / B)

Tablet Profile Structure

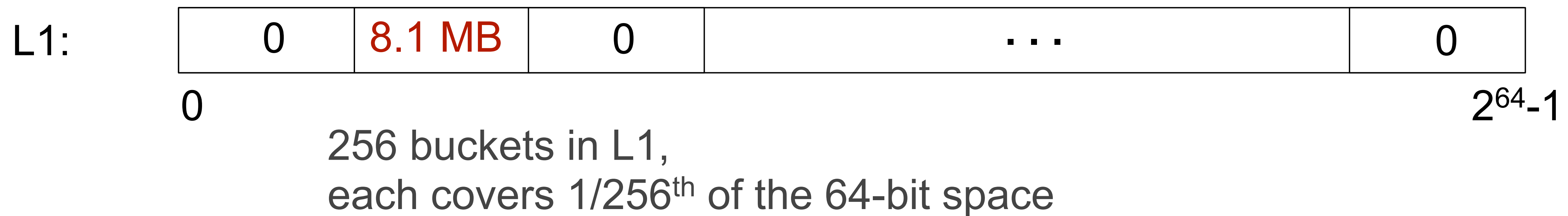
- **Example: B = 8, S = 8MB**



- **With B = 8, there are up to 8 levels**
  - NumLevels = ceil(64 / B)

Tablet Profile Structure

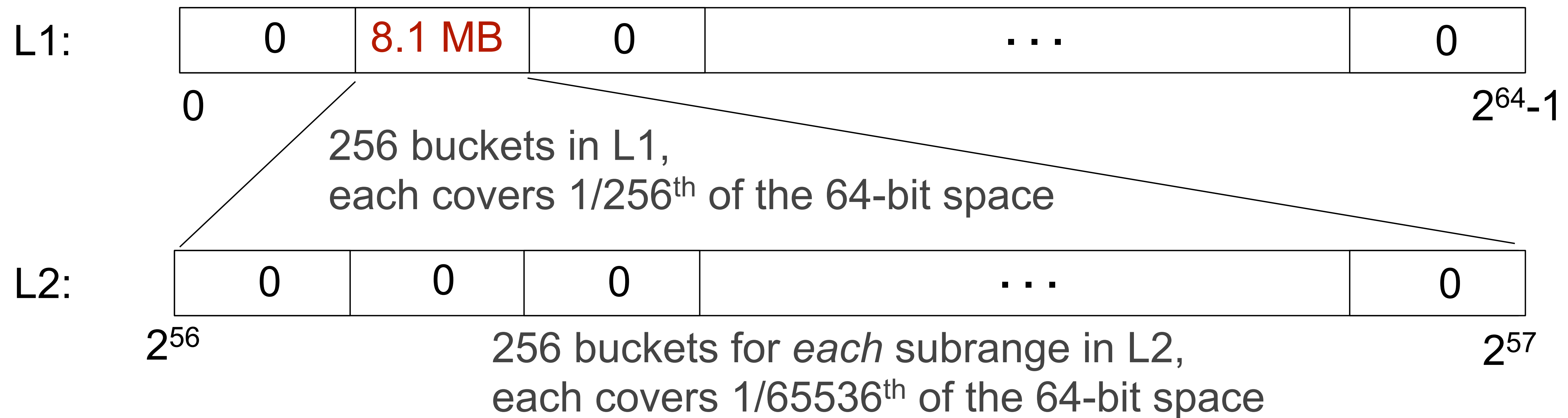
- **Example: B = 8, S = 8MB**



- **With B = 8, there are up to 8 levels**
  - NumLevels = ceil(64 / B)

Tablet Profile Structure

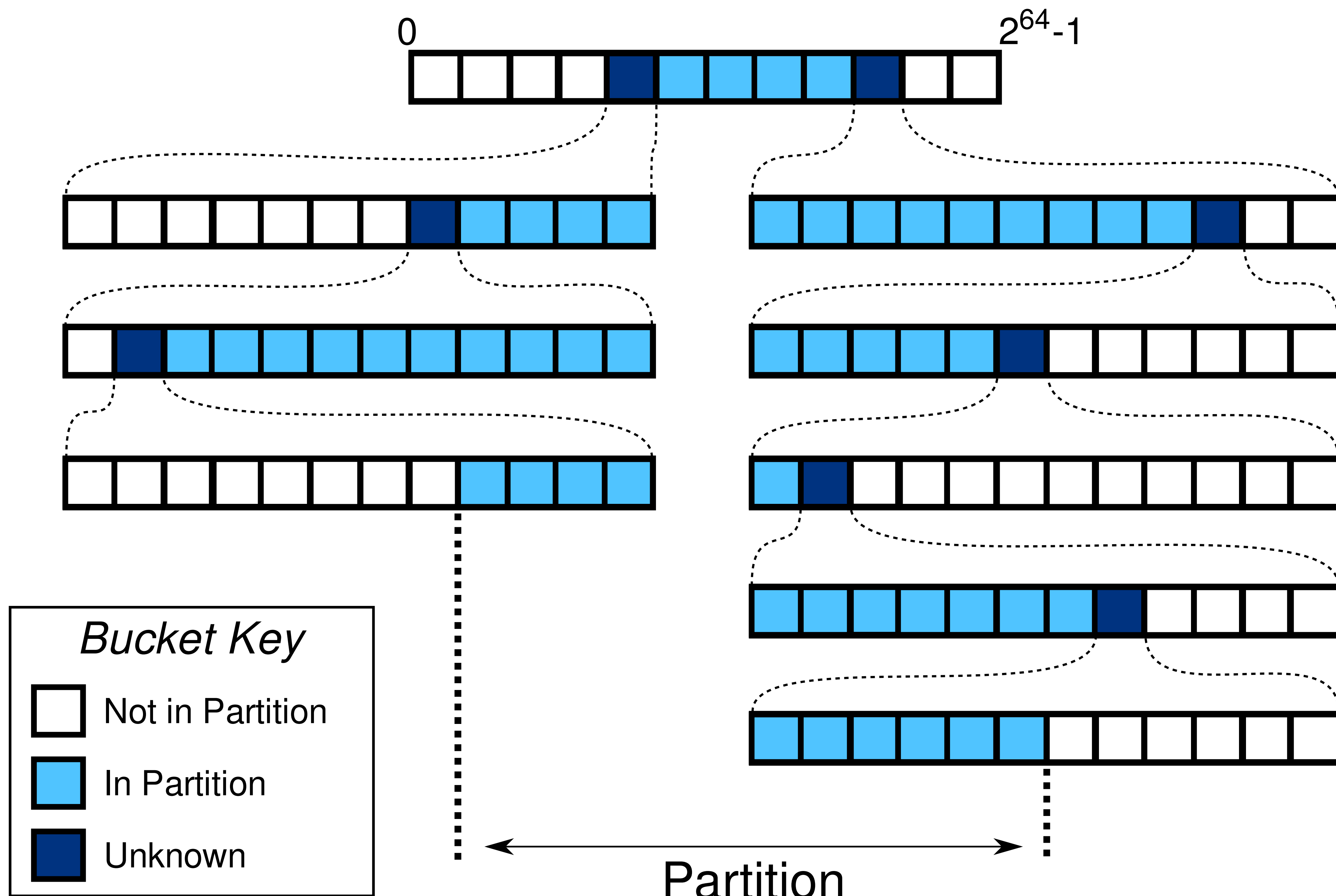
- **Example: B = 8, S = 8MB**



- **With B = 8, there are up to 8 levels**
  - NumLevels = ceil(64 / B)

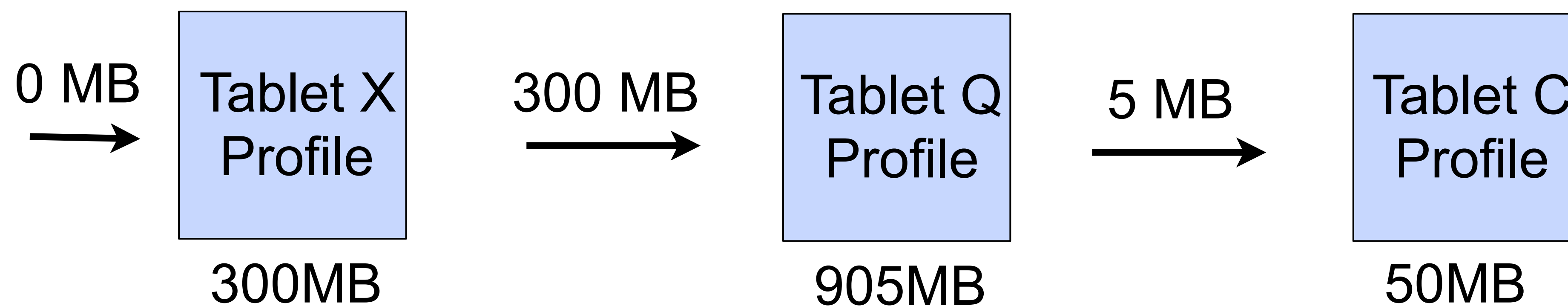
Bounding Error

- **How does this bound the error?**
  - Lines are drawn in leaf subranges
  - We have exact counts for all data between the border subranges.



Computing Partitions

- Walk each tablet profile
- Once we reach 600MB, we've found a partition
- If we don't reach 600MB, pass the previous count into the next profile



Partition #	Table	First Key	Last Key
0	X	0	2^{64} - 1
0	Q	0	5,723,742
1	Q	5,723,742	2^{64} - 50
2	Q	2^{64} - 49	2^{64} - 1
2	C	0	2^{64} - 1

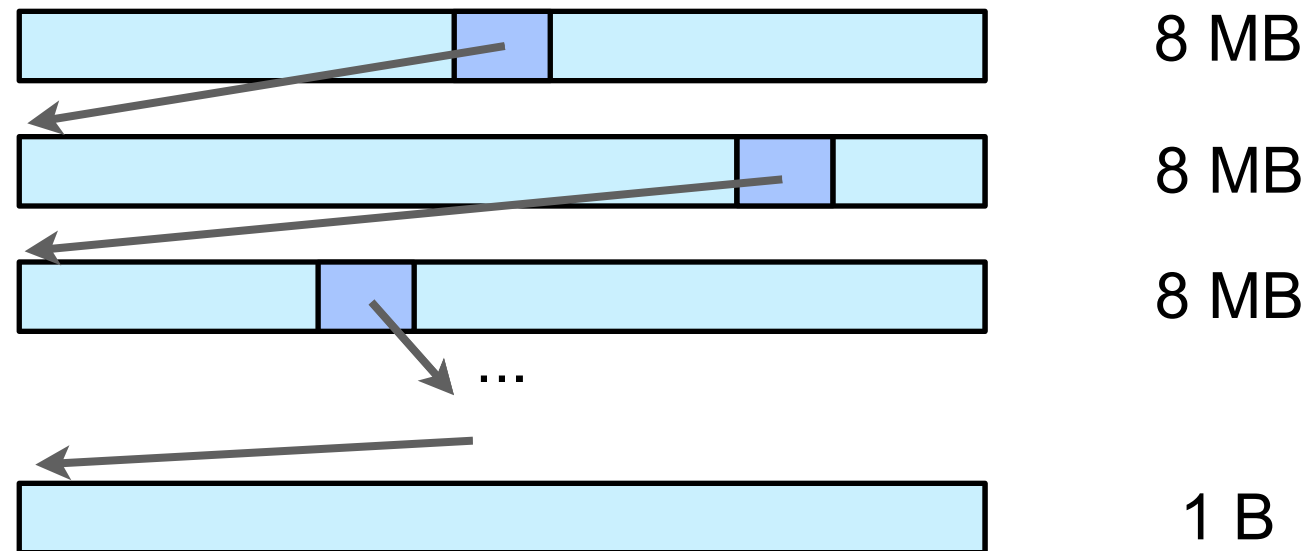
The Last Will and Testament

- **While still alive, each server maintains a *will***
  - Groups tablets into even, well-sized chunks (~600MB)
- **The will is synchronized with the coordinator**
  - Needn't be strictly consistent, so long as it's complete (describes all objects owned)
- **When a server crashes, the coordinator uses its will to determine what data each recovery master inherits**
  - Recovers data according to *partitions* listed in the will

Partition #	Table #	First Key	Last Key
0	12	0	$2^{64} - 1$
0	47	63,742	5,723,742

Space Complexity

- **Worst case: Lots of small tablets that look like this:**



- **For 8-level trees, that's 8 subrange structures for 8MB*7 + 1B ~= 56MB of data**
  - Each subrange (256 buckets) consumes about 16 * 256 bytes = 4K.
    - 16 bytes for object and byte counts
  - 64GB / 54MB = 1200
  - 1200 * 4K ~= 4.7MB, or .007% overhead.

Time Complexity

- **Updates require walking tree structure**
  - Follow logarithmic number of pointers
  - < 1 microsecond
  - Subranges can be pooled for quick allocation if we need to expand
- **B = 8: 8 levels in the tree**
  - Currently have plenty of time while waiting for backups to acknowledge replicated writes

Outstanding Issues

- **A *single* object can occupy arbitrary log space and we can't split a single key...**
  - E.g. Update, Update, Update, ...
  - Can mitigate by eliding previous objects in same segment
    - With 1MB objects and 8192 segments/server, can still have 8GB of old objects in different segments.

- **Wills aren't yet computed online**
  - Batch processing too inefficient with 10,000+ tables

Questions

?