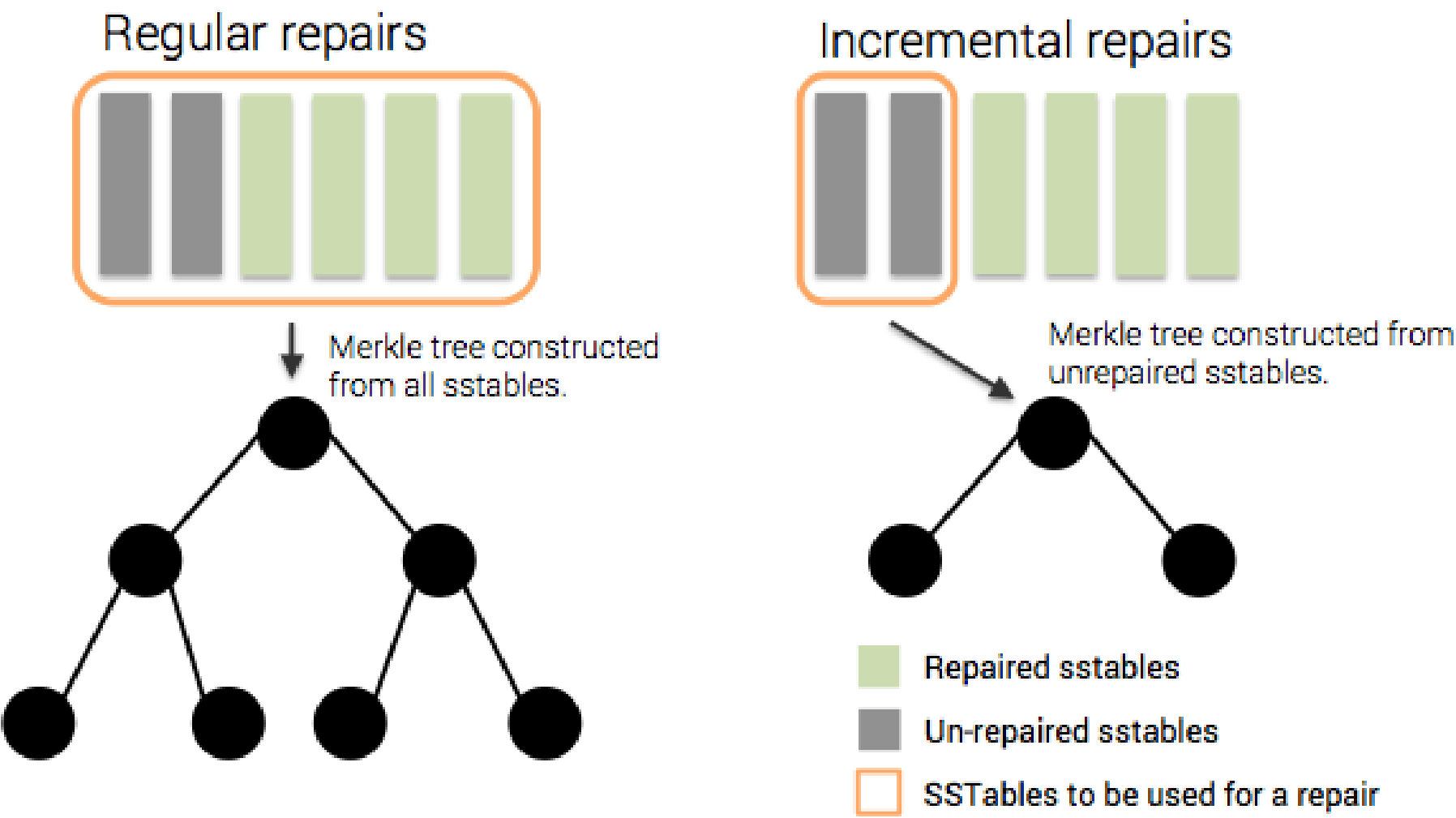


Repairs are important for every Cassandra cluster, especially when frequently deleting data. Running the `nodetool repair` command initiates the repair process on a specific node which in turn computes a Merkle tree for each range of data on that node. The [merkle tree](#) is a binary tree of hashes used by Cassandra for calculating the differences in datasets between nodes in a cluster. Every time a repair is carried out, the tree has to be calculated, each node that is involved in the repair has to construct its merkle tree from all the sstables it stores making the calculation very expensive. This allows for repairs to be network efficient as only targeted rows identified by the merkle tree as inconsistencies are sent across the network.

Scanning every sstable to allow for the creation of merkle trees is an expensive operation. To avoid the need for constant tree construction incremental repairs are being introduced in Cassandra 2.1. The idea is to persist already repaired data, and only calculate merkle trees for sstables that haven't previously undergone repairs allowing the repair process to stay performant and lightweight even as datasets grow so long as repairs are run frequently.



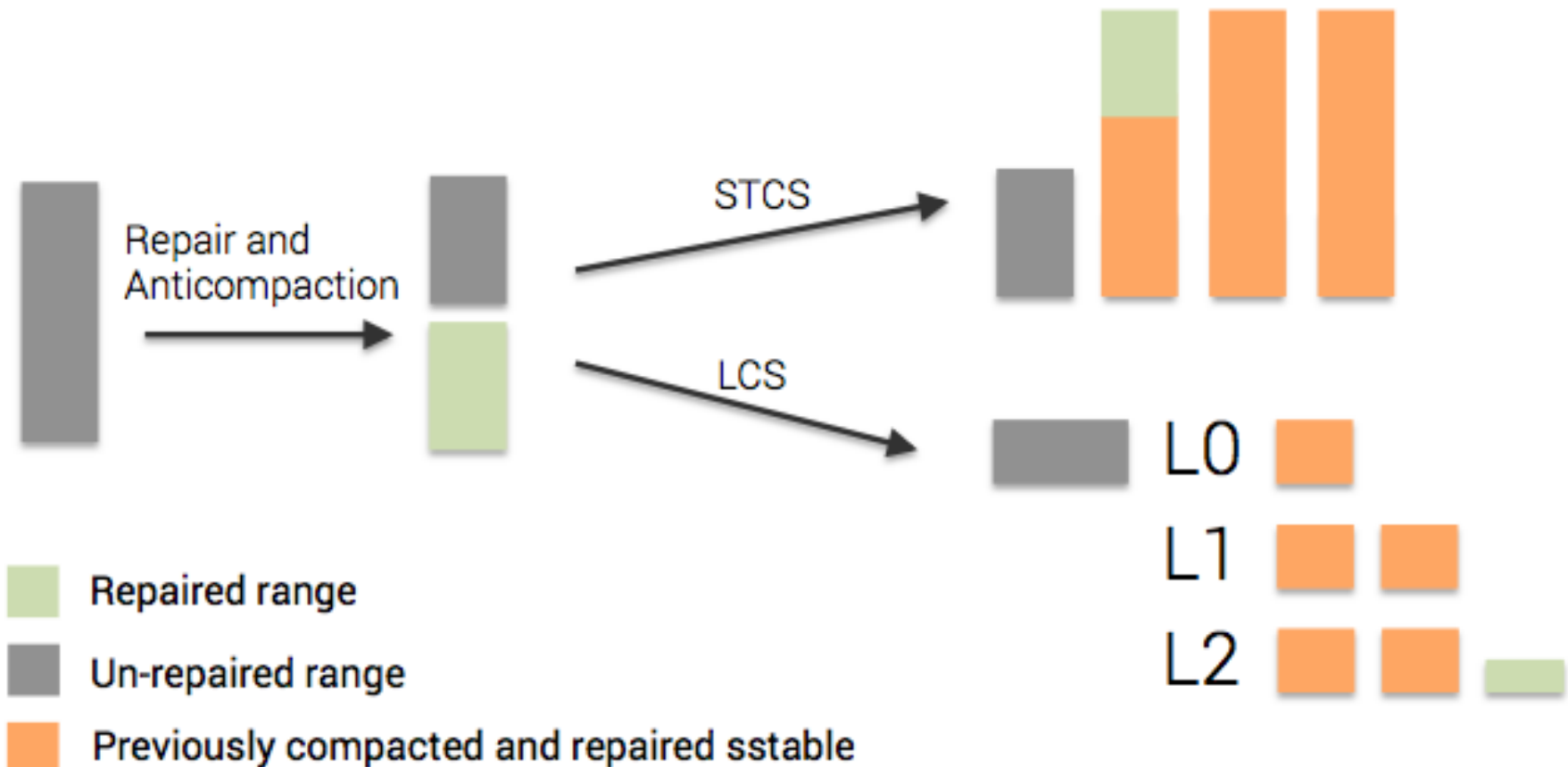
Incremental repairs begin with the repair leader sending out a prepare message to its peers. Each node builds a merkle tree from the un-repaired sstables, which it can distinguish by the new `repairedAt` field in each sstable's metadata. Once the leader receives a merkle tree from each node, it compares the trees and issues streaming requests, just as in the classic repair case.

Finally, the leader issues an anticompaaction command. Anticompaaction is the process of segregating repaired and unrepaired ranges into separate sstables; repaired sstables are written with a new `repairedAt` field denoting the time of repair. Since sstable are not locked against compaction during the repair, they might get removed via compaction before the process completes. This costs us some efficiency, since they will be repaired again later, but does not harm correctness.

## Compaction with incremental repairs

Maintaining separate pools of repaired and unrepaired sstable causes some extra complexity for compaction to deal with. For example, in the diagram below we repair a range covering half of the initial sstable. After repair, anticompaaction splits it into a set of repaired and unrepaired sstables, at which point leveled and size-tiered compaction strategies handle segregation of the data differently.

Size-Tiered compaction takes a simple approach of splitting repaired and unrepaired sstables into separate pools, each of which is compacted independently. Leveled compaction simply performs size-tiered compaction on unrepaired data, moving into the proper levels after repair. This cuts down on write amplification compared to maintaining two leveling pools.



## Migrating to incremental repairs

Full repairs remain the default, largely so Cassandra doesn't have to guess the repaired state of existing sstables. Guessing that everything is fully repaired is obviously problematic; guessing that nothing is repaired is less obviously so: LCS would start size-tiering everything, since that is what it does now with unrepaired data! To avoid this, compaction remains unchanged until incremental repair is first performed and compaction detects sstables with the `repairedAt` flag.

Incremental repairs can be opted into via the `-inc` option to `nodetool repair`. This is compatible with both sequential and parallel (`-par`) repair, e.g., `bin/nodetool -par -inc <ks> <cf>`. When an sstable is fully covered by a repaired range, no anticomaction will occur, it will just rewrite the `repairedAt` field in sstable metadata. Recovering from missing data or corrupted sstables will require a non-incremental full repair. (For more on anticomaction, see Marcus's post [here](#).)

## Effect of tools / commands on repair status

Since the sstable's repair status is now tracked via it's metadata, understanding how the set of tools provided with open-source Cassandra can impact this repair status becomes important.

Bulk Loading - even if repaired in a different cluster, loaded tables will be unrepaired.

Scrubbing - if scrubbing results in dropping rows, new sstables will be become unrepaired, however if no bad rows are detected, the sstable will keep its original repairedAt field.

Major compaction - STCS will combine each of its pools into a single sstable, one repaired and one not. Major compaction continues to have no effect under LCS.

Setting Repaired Status - a new tool added in 2.1 beta 2 can be found in `tools/bin/sstablerepairedset` that allows users to mark an sstable as repaired manually allowing for an easy migration to using incremental repairs by using the `sstablerepairedset --is-repaired <sstable>` command. It's important to only use this tool on repaired sstables, the status of an sstable can be checked via the `/tools/bin/sstablemetadata` tool by looking at the repairedAt field.

Resources

Webinars

Datasheets

Case Studies

Whitepapers

Reports

Videos

Podcasts

Blog

Company

About Us

Apache Cassandra™

Leadership

Events

Press Releases

In The News

Careers

Academy

Online Courses

Short Courses

Certifications

Instructor Led Training

Public Training

Developer Blog

Contact Us


+1 (650) 389-6000

info@datastax.com

Subscribe to Newsletter

Email Address

By subscribing you confirm you agree to the processing of information as described in our [website privacy policy](#) and agree to our [website terms of use](#). If you prefer not to receive marketing emails from us you can opt-out of marketing communications at any time by using the unsubscribe link provided in our marketing emails.



Do not sell my personal data

Powered by OneTrust