setting environment

after every flush, perform the following check

```
//Update d[i] if a new level is added to the tree

// T = size ratio; level_count = levels in tree
// D_th = delete persistence threshold
double x = D_th * T / pow ( T, level_count - 1 );
if ( new_level_added ( ) ) {
   for ( int i = 0; i < level_count; ++i ) {
      d[i] = x * pow ( T, i );
      if ( i > 0 )
         d[i] += d[i-1];
   }
}
```

tombstones

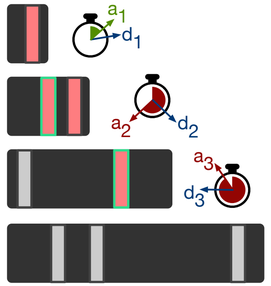
TTL-expired tombstones

logically deleted entries

FADE: enforcing a finite bound for delete persistence latency

initial state*

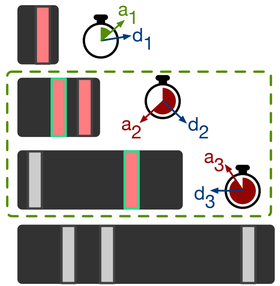
Levels 1, 2, and 3 all have files that contain tombstone



Choose level that has files with expired TTL

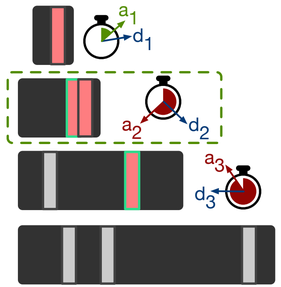
selecting files for compaction

Levels 2 and 3 both have files with expired TTL



Chose the file that has the highest tombstone count

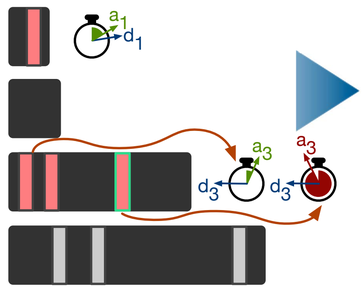
Level 2 has the file with highest tombstone count



Initiate compaction with chosen file and update age of files

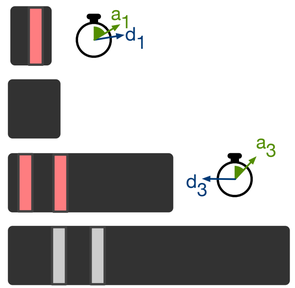
performing compaction

After compaction, Level 3 has 2 files with tombstones



Initiate compaction with the file that has expired TTL

All files with expired TTL are compacted — repeat routine



This ensures a finite upper-bound for delete persistence latency

* timers are shown only for files containing tombstones