logstor: A Log-structured Use-level GEOM Layer

Wuyang Chung

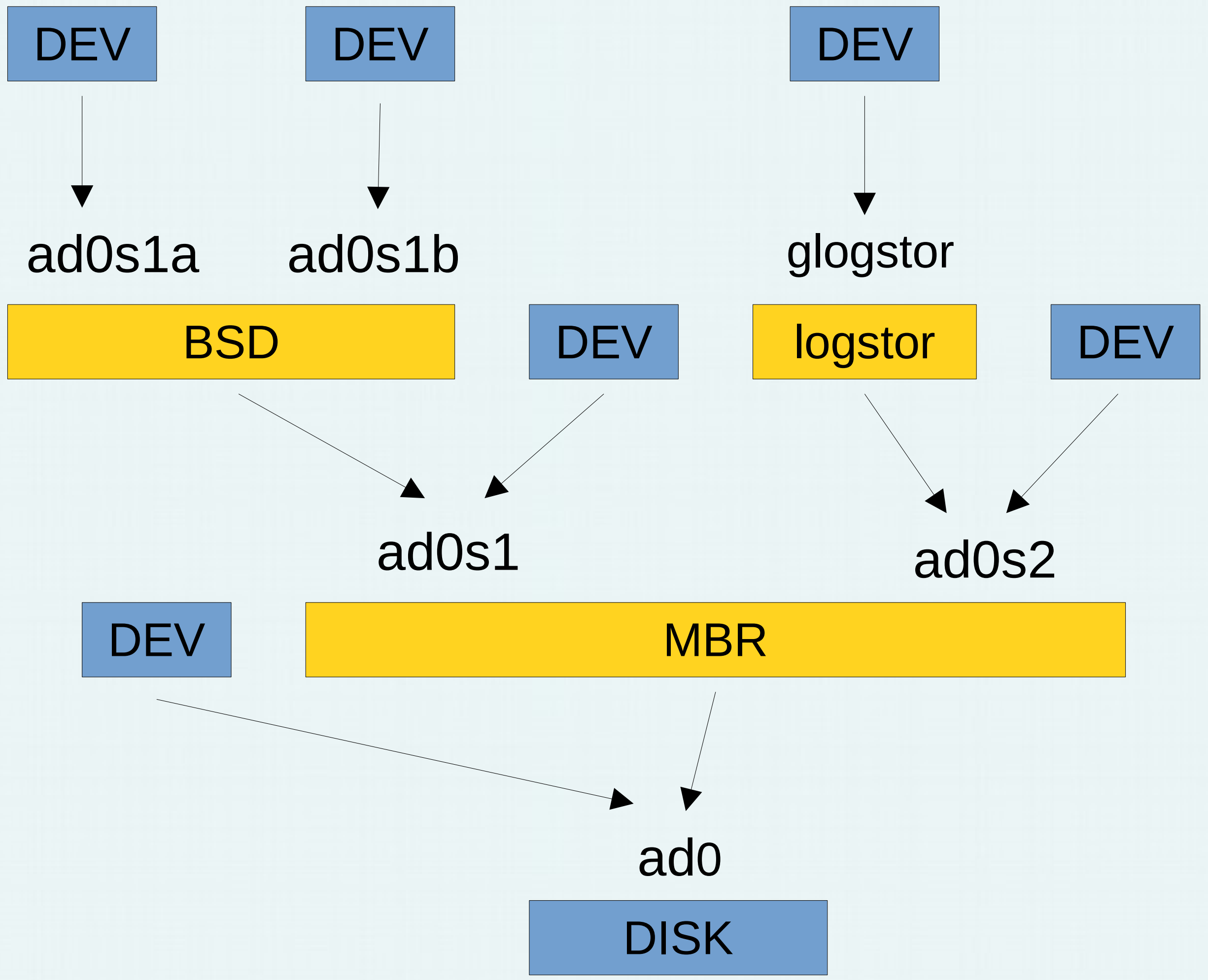
wy-chung@outlook.com
https://github.com/wy-chung/logstor.git

Sept. 19 2021

Outline

- Introduction
  - GEOM
  - logstor
- Implementation
- Performance
- Future Work

Introduction - GEOM

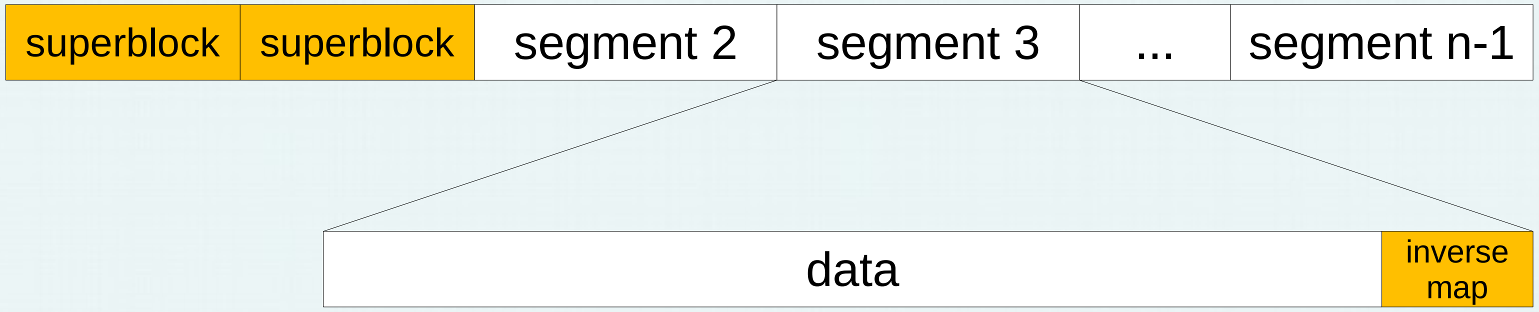


Introduction - logstor

- Use the idea from log-structured file system
  - Data written are appended to the end of the log
    - Can transform random write to sequential write
  - Implement a simple file system to store the forward map information
  - Also need a inverse map information for garbage collection
    - Stored at the end of every segment
  - The algorithm used for garbage collection is hot-cold separation with aging for wear leveling

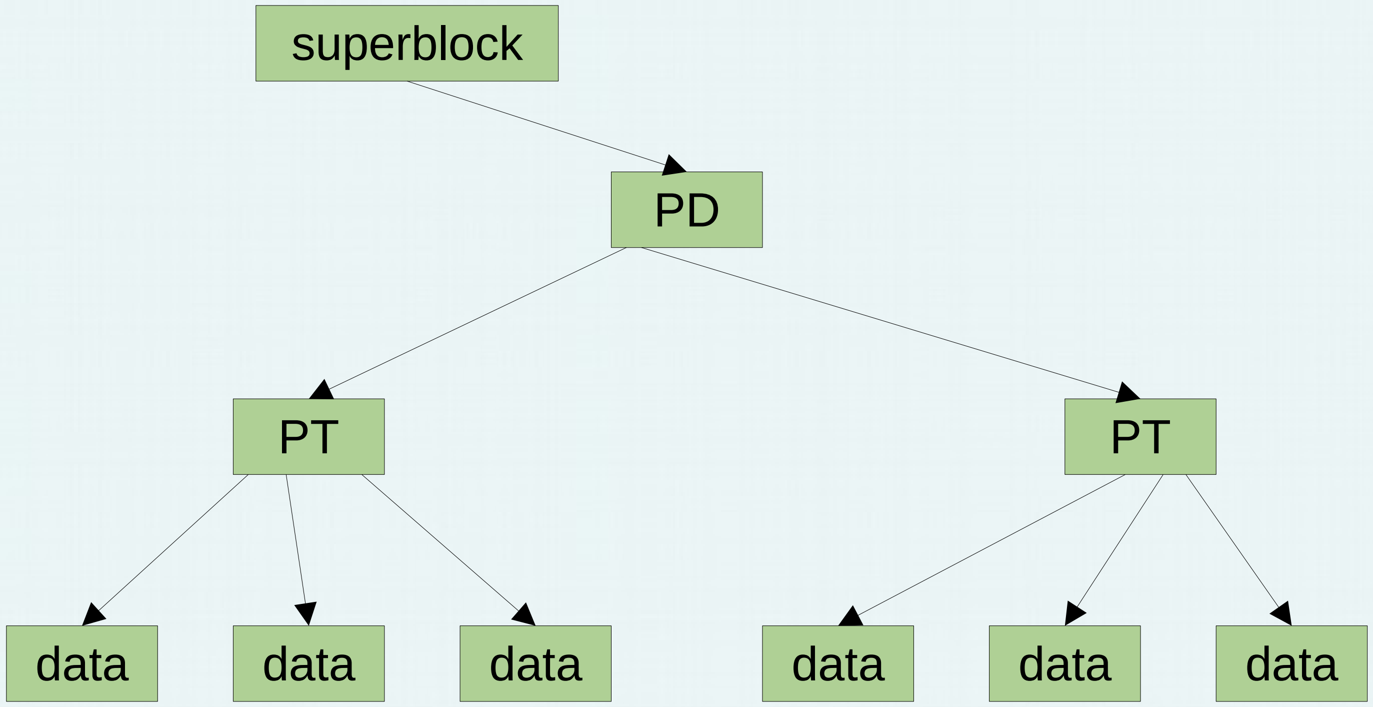
Implementation

- Disk is divided into segments
- The first two segments are for superblock
- A inverse map is stored at the end of each segment
- A “simple file system” is used to store the forward map information
  - Its data and metadata are also stored in logstor's data areas



Implementation

A simple file on disk



Implementation

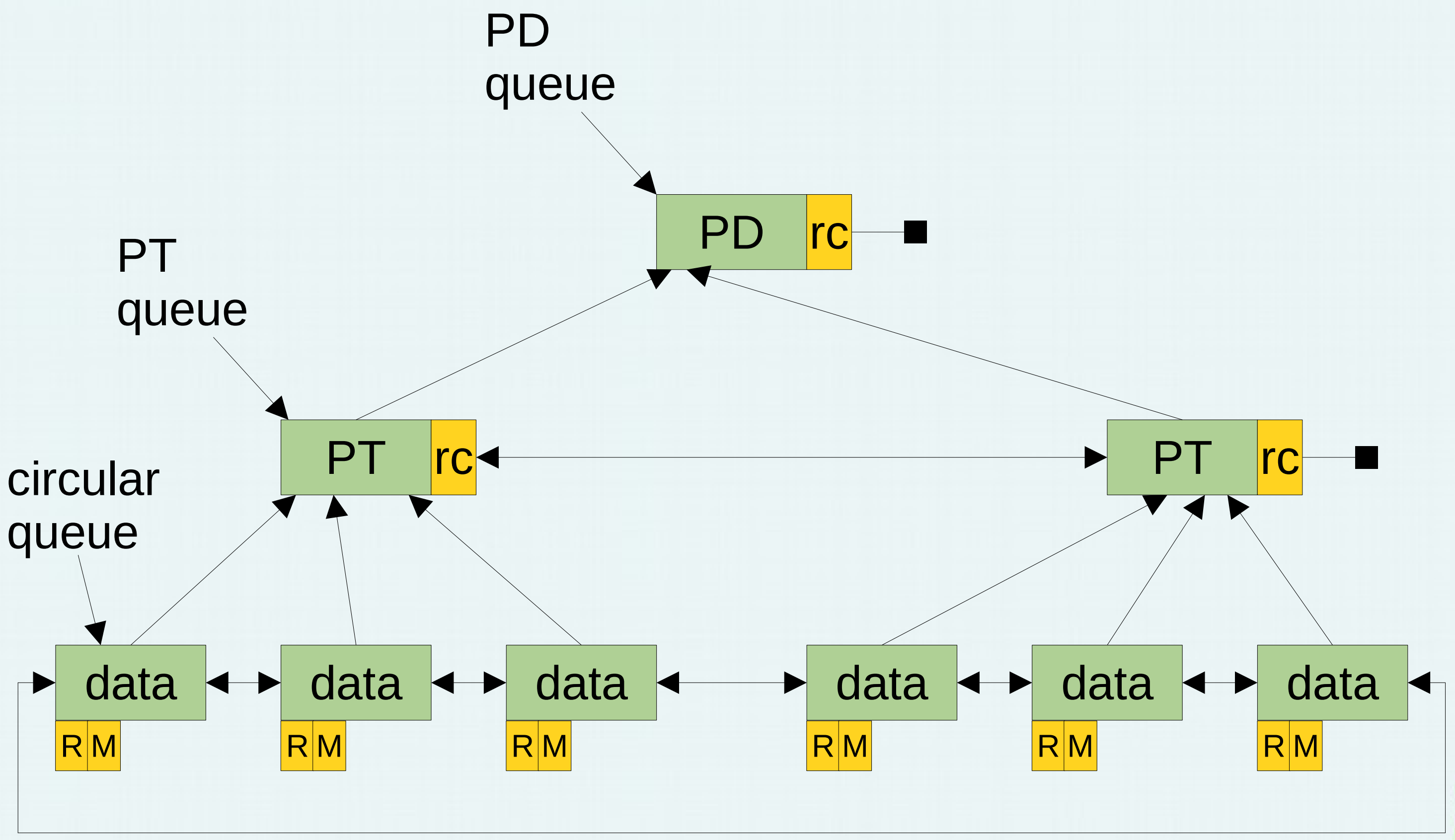
- The simple file system
  - For storing forward map information
  - Supports at most 4 files
  - Doesn't support sub-directory
  - Doesn't support file naming, use a integer number instead
  - Doesn't record the modify time, access time, file size, ...
  - Use page table data structure to store simple file’s metadata

Implementation

- The simple file system
  - Reserve a small portion of logical address for simple file system
    - Since the metadata (PDs and PTs) and data blocks of the simple file system are also stored in logstor's data area
    - Assign a unique logical address for each metadata and data blocks of the simple file system
    - The logical address bigger than 0xFF000000 is reserved for simple file system
  - Use a buffer cache to cache the recently used data and metadata

Implementation

The buffer cache in DRAM for simple file system



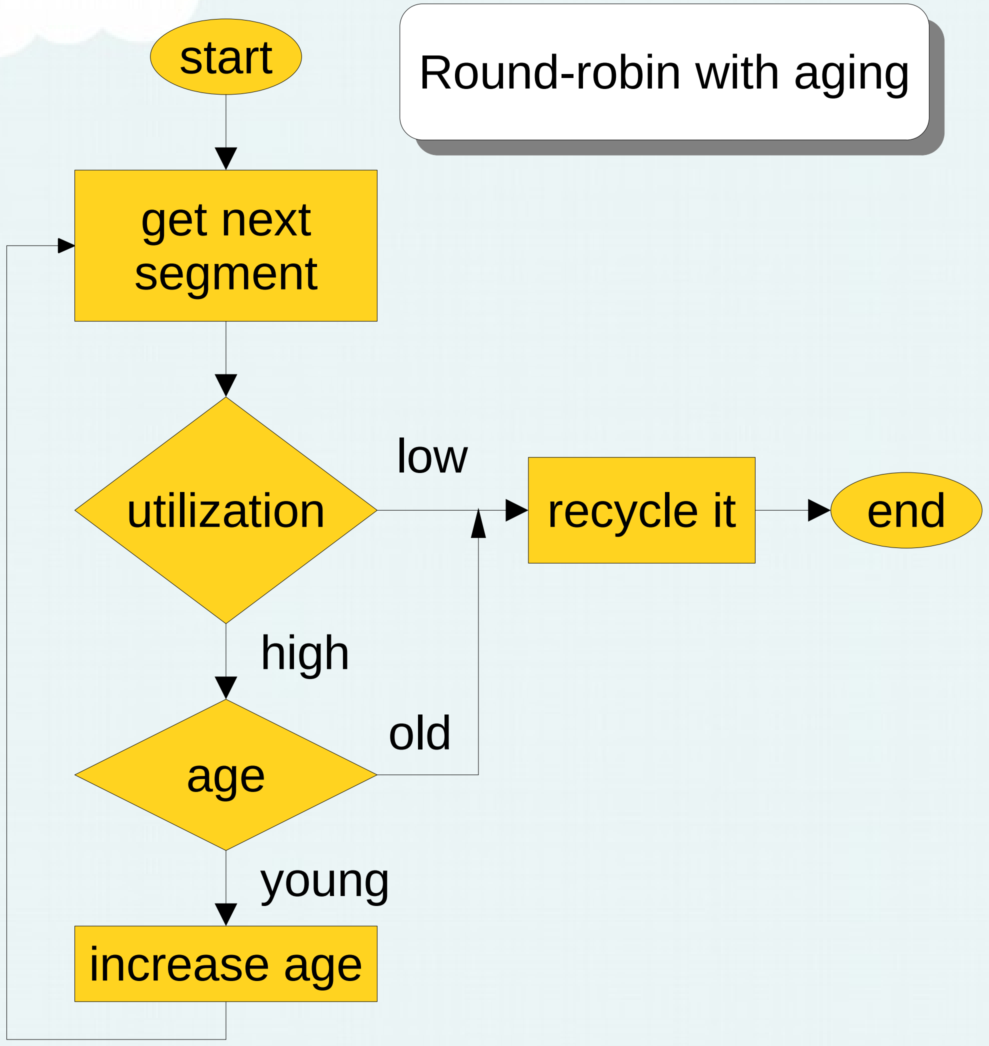
Implementation

- The simple file buffer cache
  - Cache the recently used metadada and data sectors
  - All PD buffers are in PD’s queue
  - All PT buffers are in PT’s queue
  - All the data buffers are in a circular queue
    - Victim is chosen from this queue
    - Replacement policy: second chance
  - All metadata (PDs and PTs) and data buffers are also placed in a hash queue
  - PD buffers and PT buffers are demoted to circular queue when its reference count becomes 0

Garbage Collection

- Hot-cold separation
  - Upper layer write → hot log
    - Upper layer should provide a hint of hotness
  - Logstor simple file write → hot log
  - Valid sectors collected from GC → cold log
- Cleaning policy
  - Segment selection for cleaning
    - Round-robin with aging
    - For wear leveling

Garbage Collection



Round-robin with aging

```
Is_sector_valid(p)
{
inverse(p) → l
forward(l) → p'
if (p == p')
        return true // valid
else
        return false //invalid
}
```

Performance

Test procedure

1. Create logstor device
2. Create a new file system on logstor and enable TRIM
3. Mount logstor to /mnt
4. Copy FreeBSD's src to /mnt
5. Build the kernel

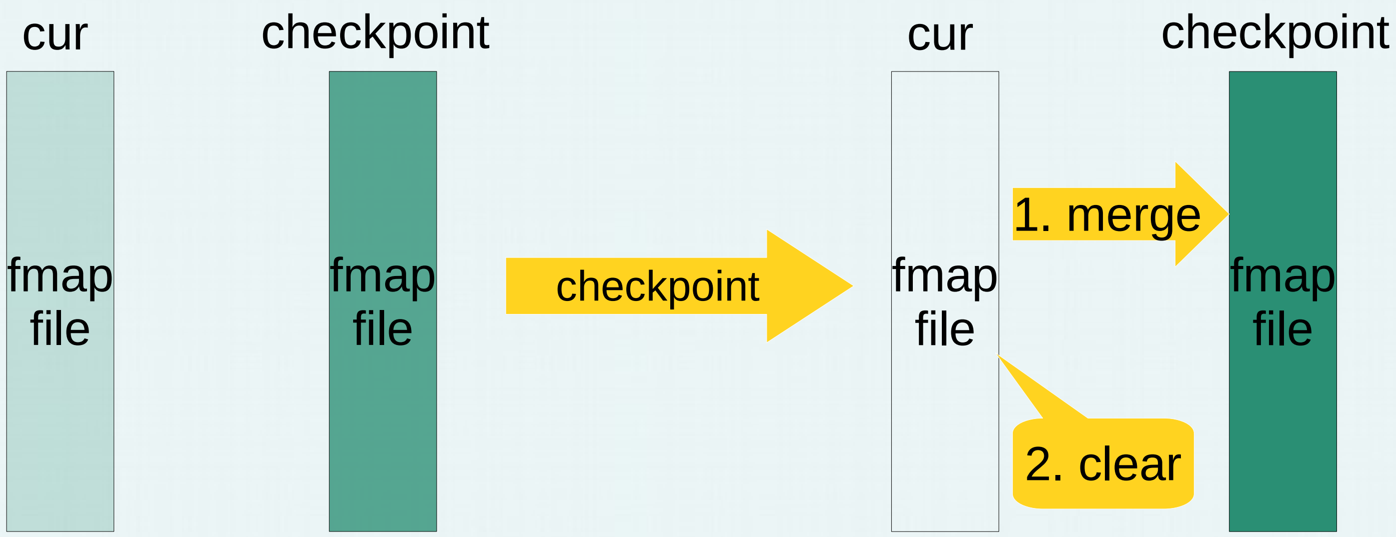
   Set the build target to /mnt/obj
6. Remove /mnt/obj
7. Remove /mnt/src

num	test case	logstor	ggatel
1	cp src	593.75 s	657.80 s
2	build kernel	3,224.51 s	3,128.66 s
3	rm obj	85.21 s	47.72 s
4	build kernel	3,191.08 s	3,100.03 s
5	rm obj	49.97 s	46.89 s
6	rm src	214.47 s	208.48 s

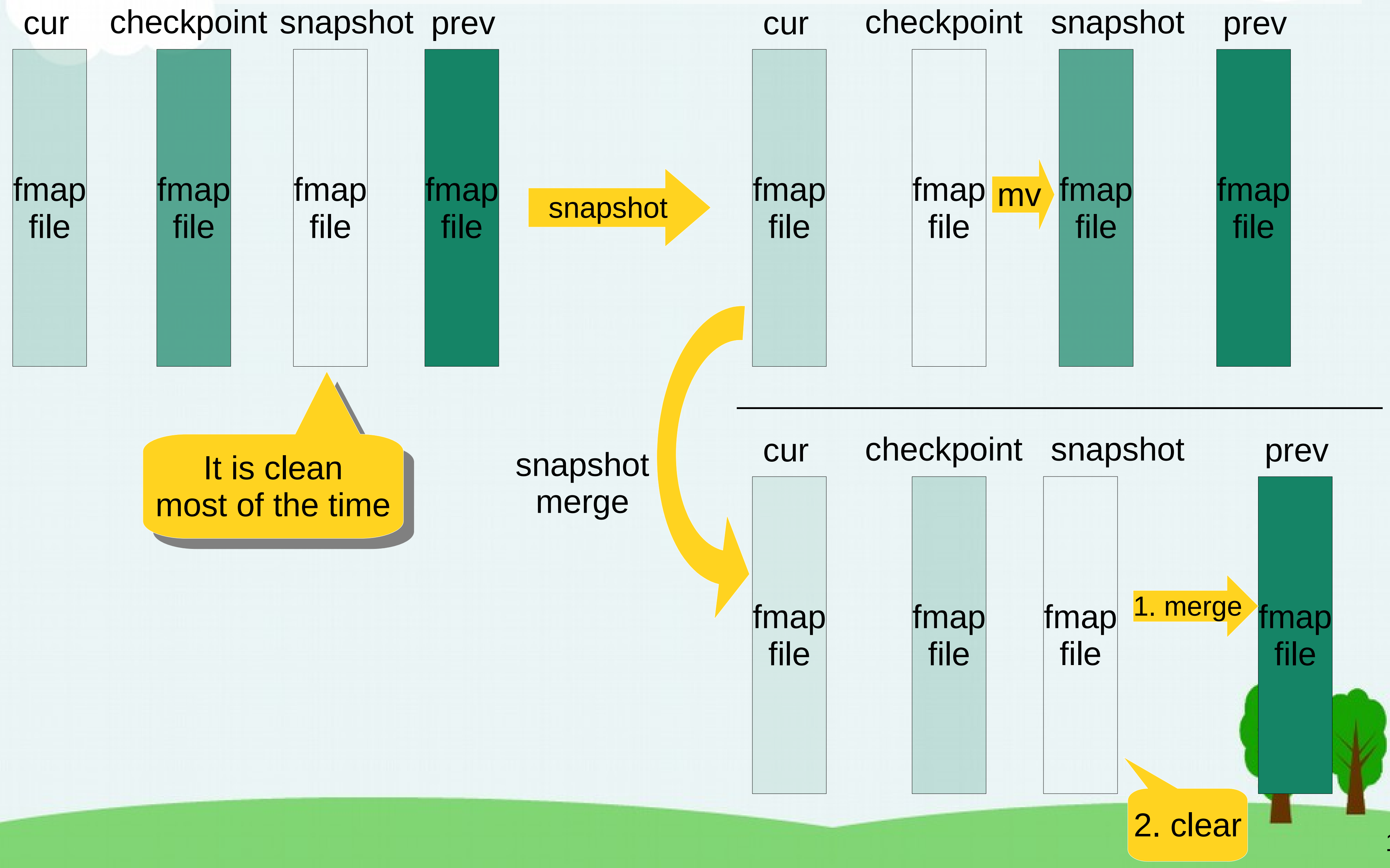
Future Work

- Move logstor to kernel level
- To support checkpoint
- To support disk-level incremental backup

Checkpoint


cur
checkpoint
fmap file
fmap file
checkpoint
cur
checkpoint
fmap file
fmap file
1. merge
2. clear

Disk Incremental Backup



The End

Questions?