WalB: Block-level WAL for Efficient Incremental Backup

Dec 2, 2010

Takashi HOSHINO

Cybozu Labs, Inc.

Contents

Motivation

- There is no good backup solution
  - Online
  - Small performance overhead
  - Supports various applications
  - Cost-effective
- We need
  - Consistent full backup
  - Incremental backup
  - Block-level backup
  - To use commodity hardware and free software only

Requirements inside Cybozu

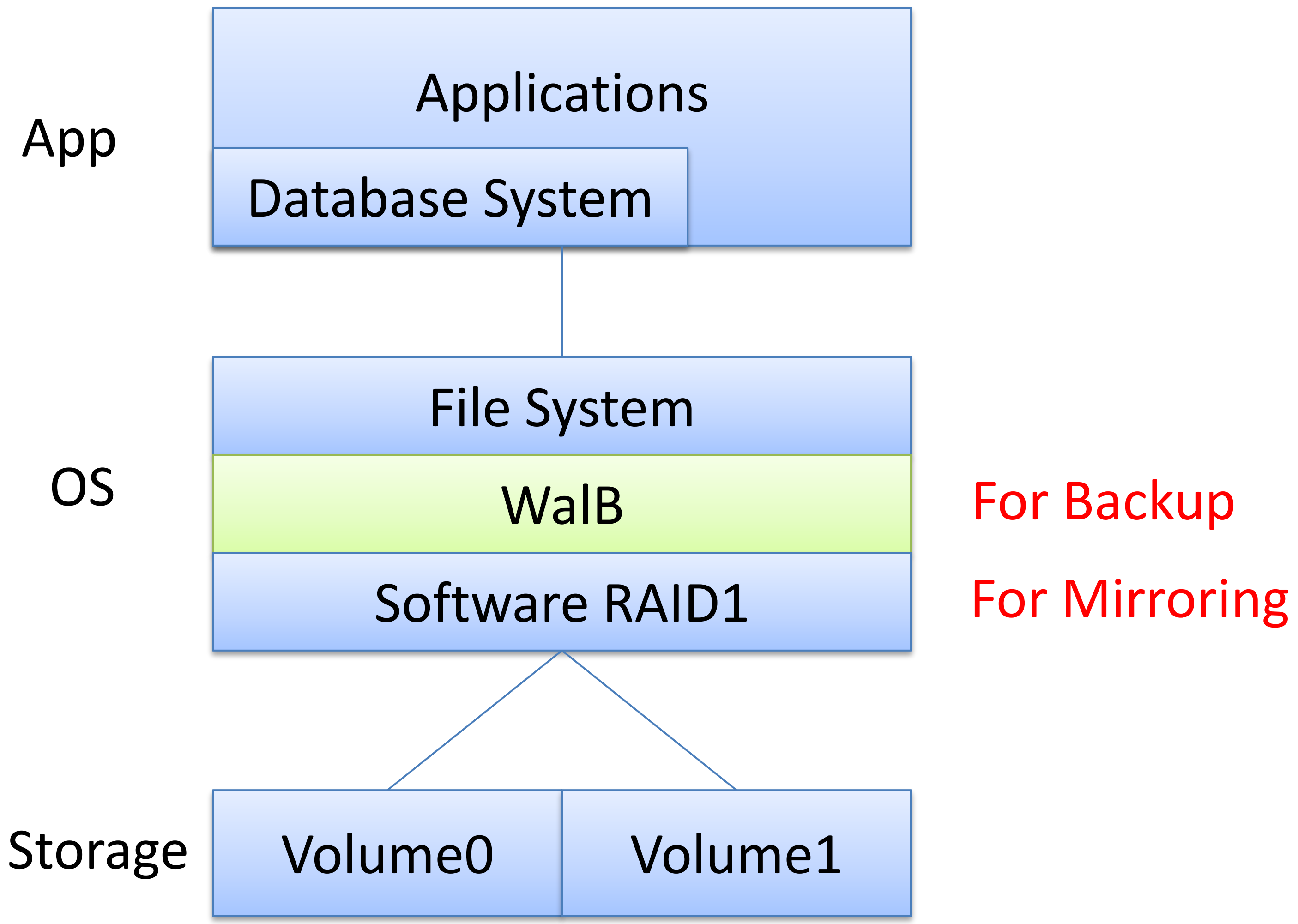
- Guarantee backup interval
  - within 5-10min
- Keep multiple backup archives
  - also in remote site
- Keep multiple snapshots
  - per 1day for 1week

WalB

- A wrapper block device driver
  - Data device to store data
  - Log device to store WAL (write-ahead log)
- Related user-land tools
  - Device controller
  - Log extractor

- Target OS and architecture
  - Latest Linux kernel (2.6.36)
  - x86_64 host

System Architecture with WalB

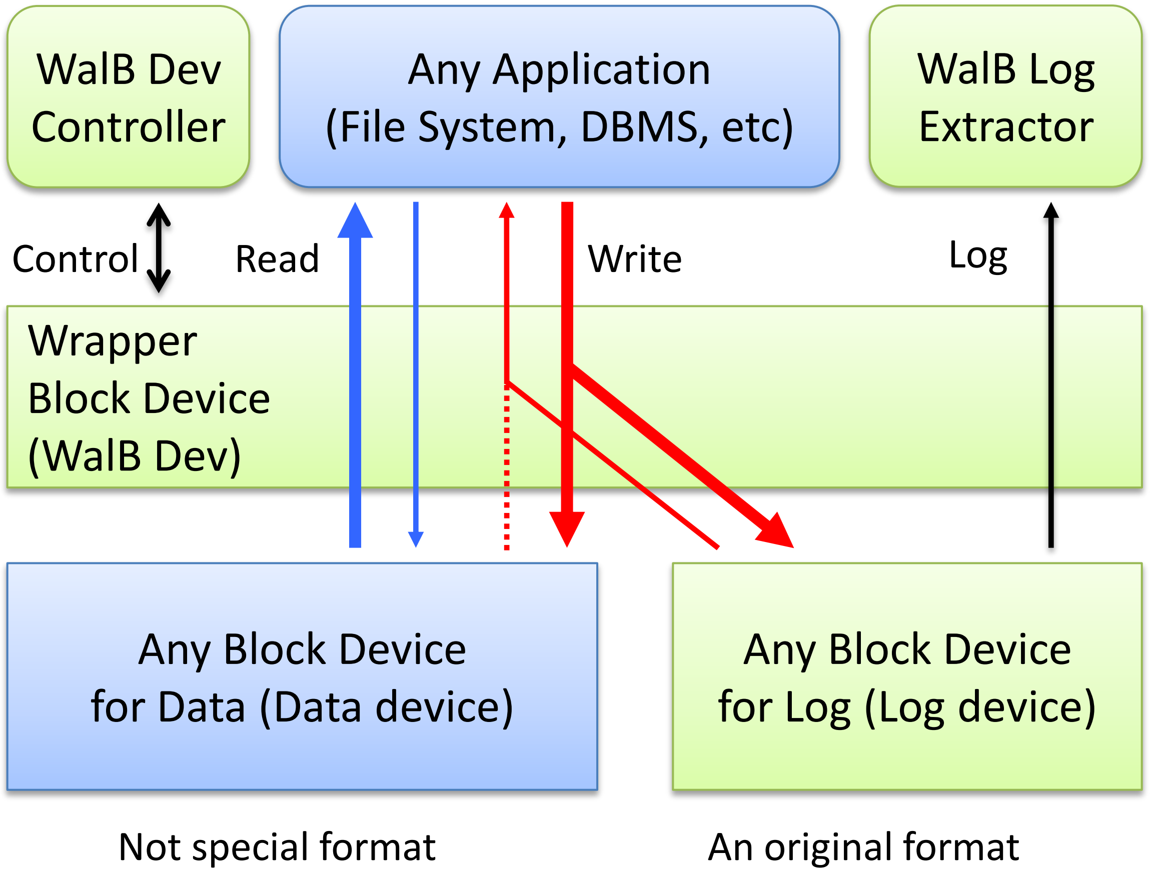


Backup vs Mirroring

	Backup	Mirroring
Typical methods	Making snapshot, Logging writes	RAID1, Replication
Recoverable failures	Operation miss, Application bug	Failure of facilities
Can keep latest data?	No	Yes

We need both functionalities to save data from lost.

WalB Architecture

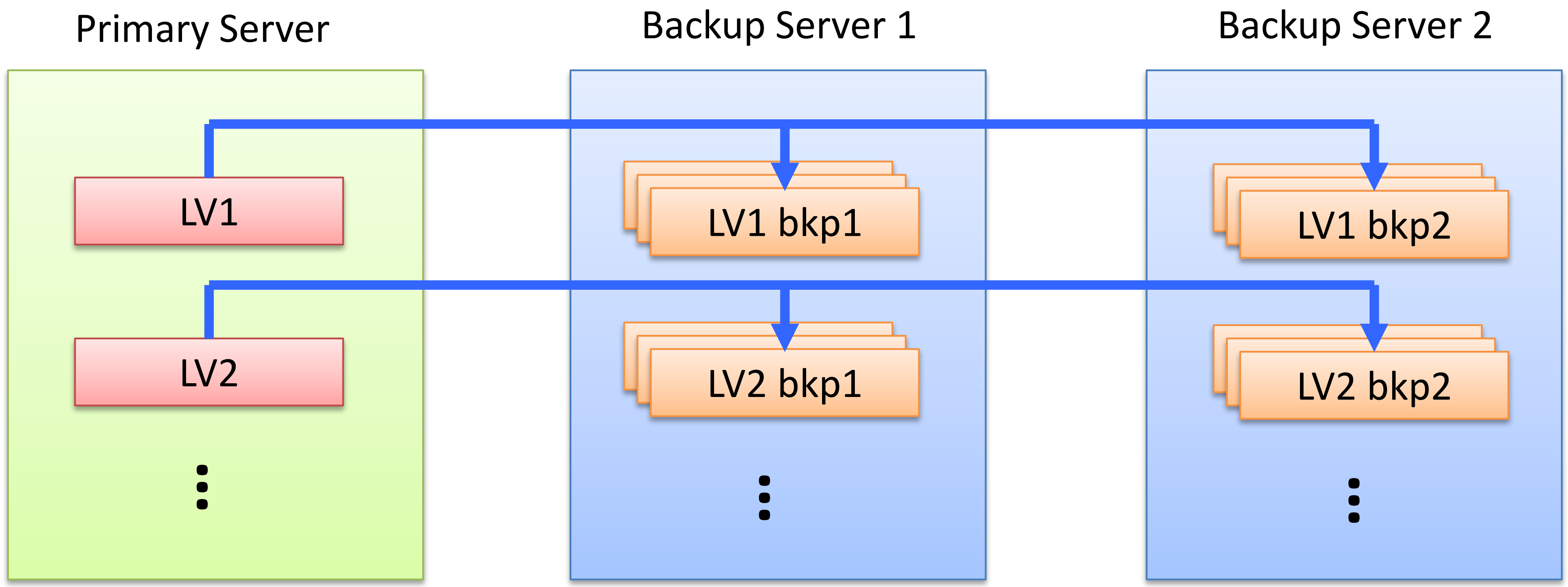

WalB Dev Controller
Any Application (File System, DBMS, etc)
WalB Log Extractor
Control
Read
Write
Log
Wrapper Block Device (WalB Dev)
Any Block Device for Data (Data device)
Any Block Device for Log (Log device)
Not special format
An original format

WalB Functionalities

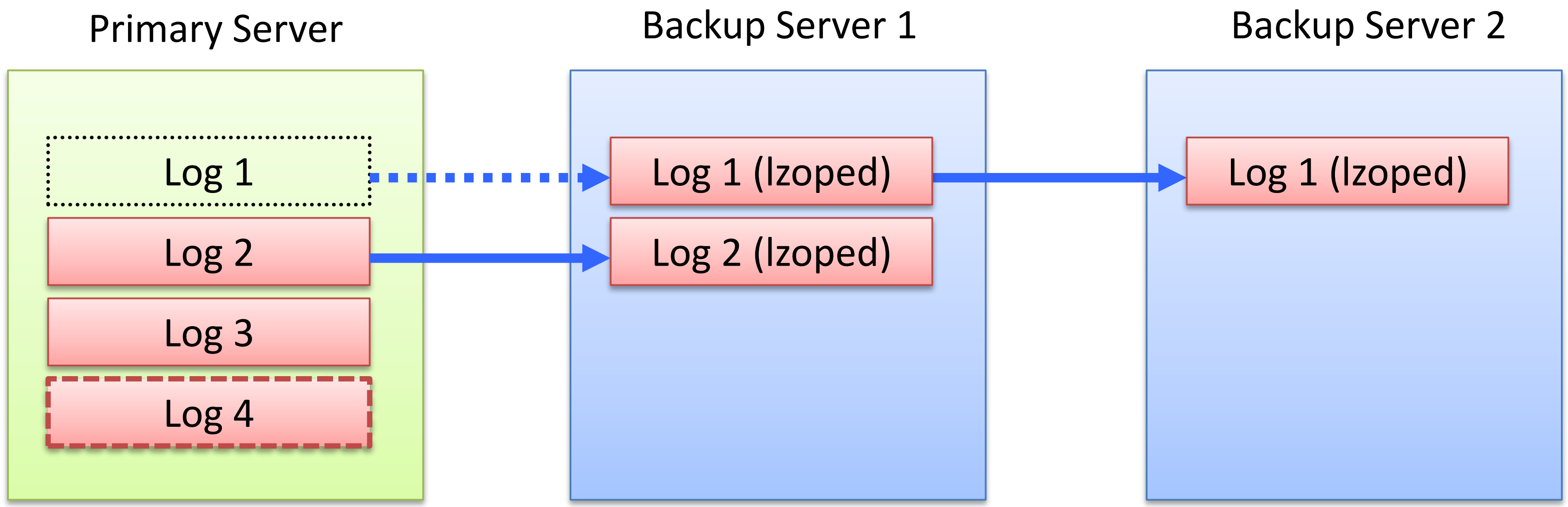
- Online incremental backup
- Online consistent full backup

- Snapshot creation/deletion
  - not accessible due to no index
- Volume resize
  - not resize of log capacity

Incremental Backup with WalB

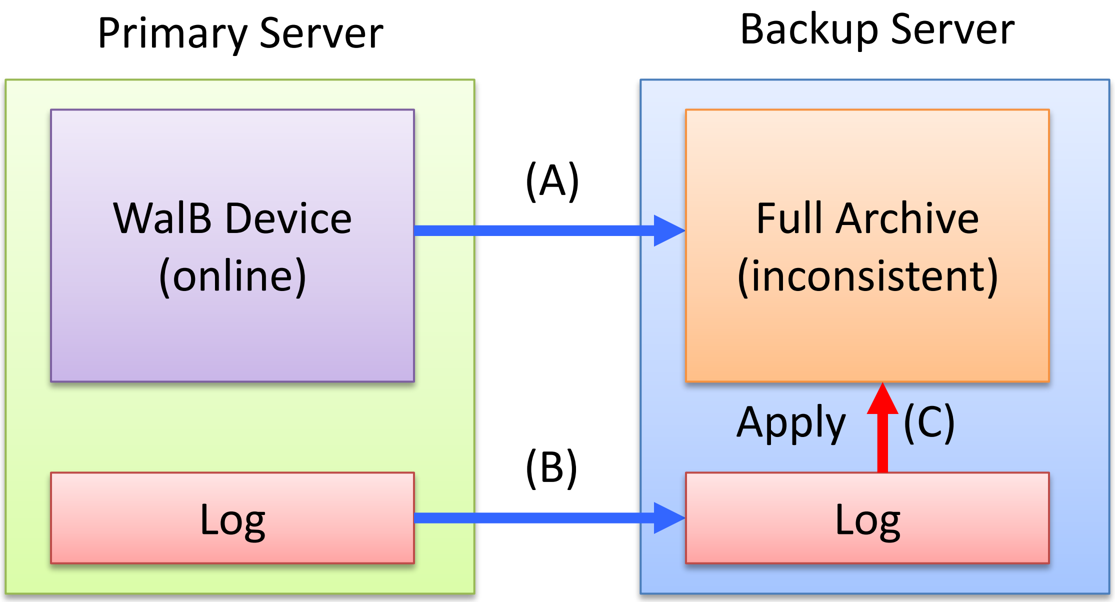


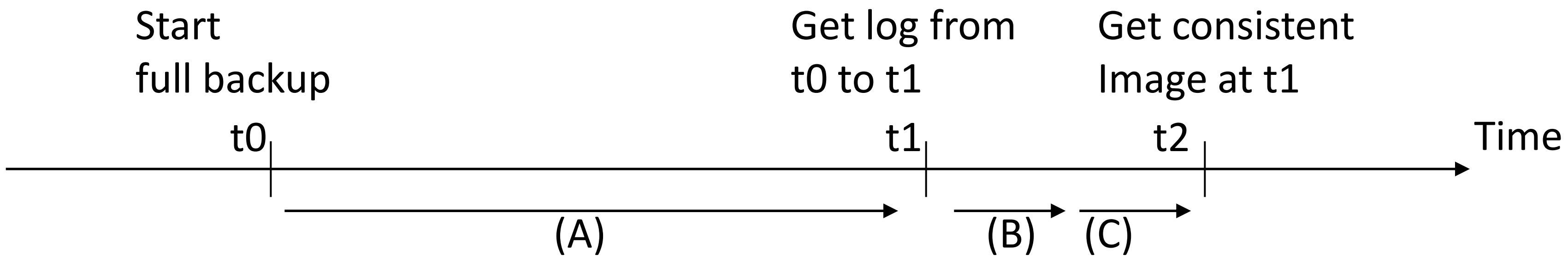
Log Transfer



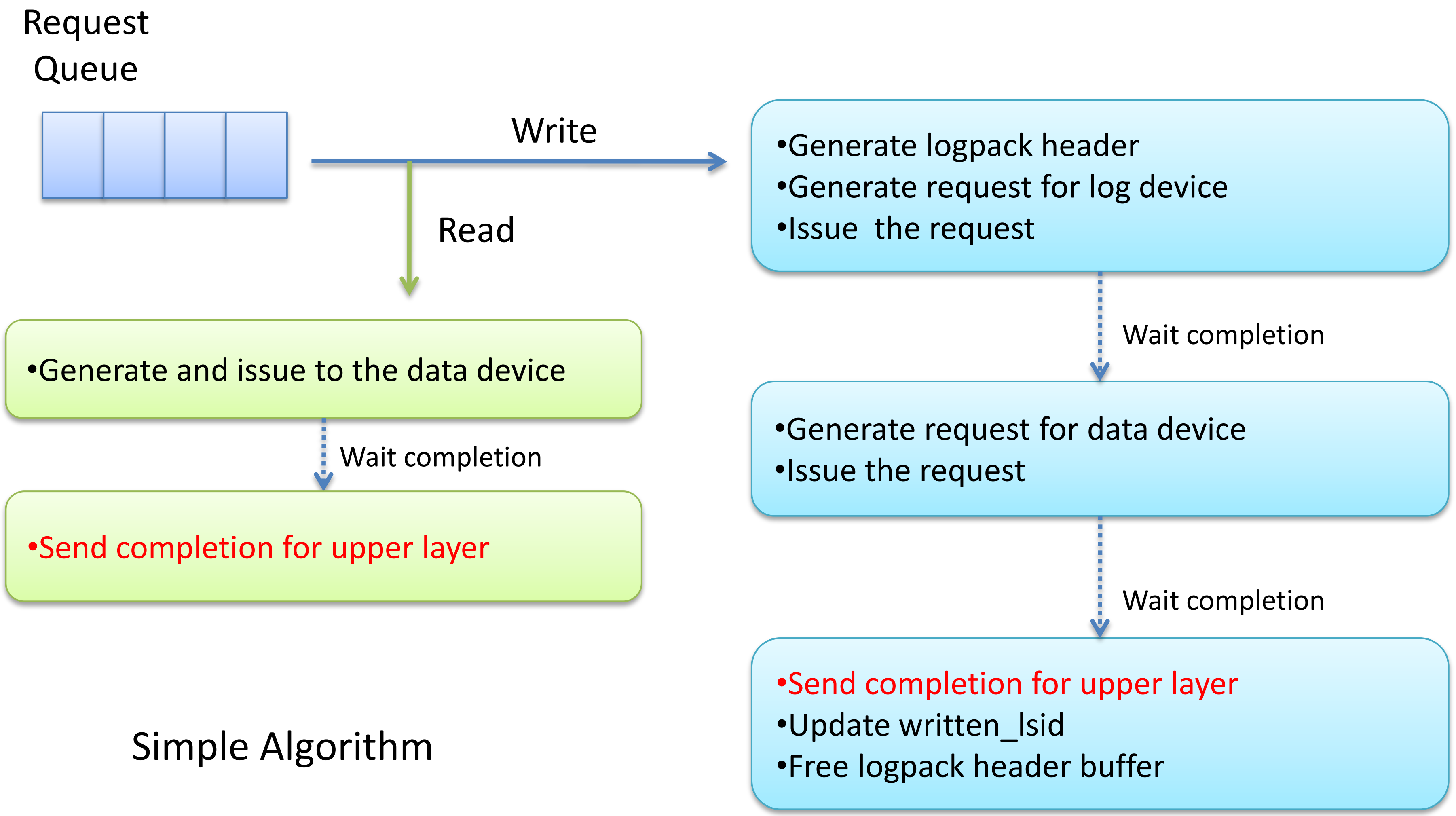
- Each log is compressed with lzop and transferred via ssh/rsh.
- Proxy may be useful for pipelined transfer to multiple sites.
- Delete original log in primary server after all backup servers have a replica.

Consistent Full Backup with WalB





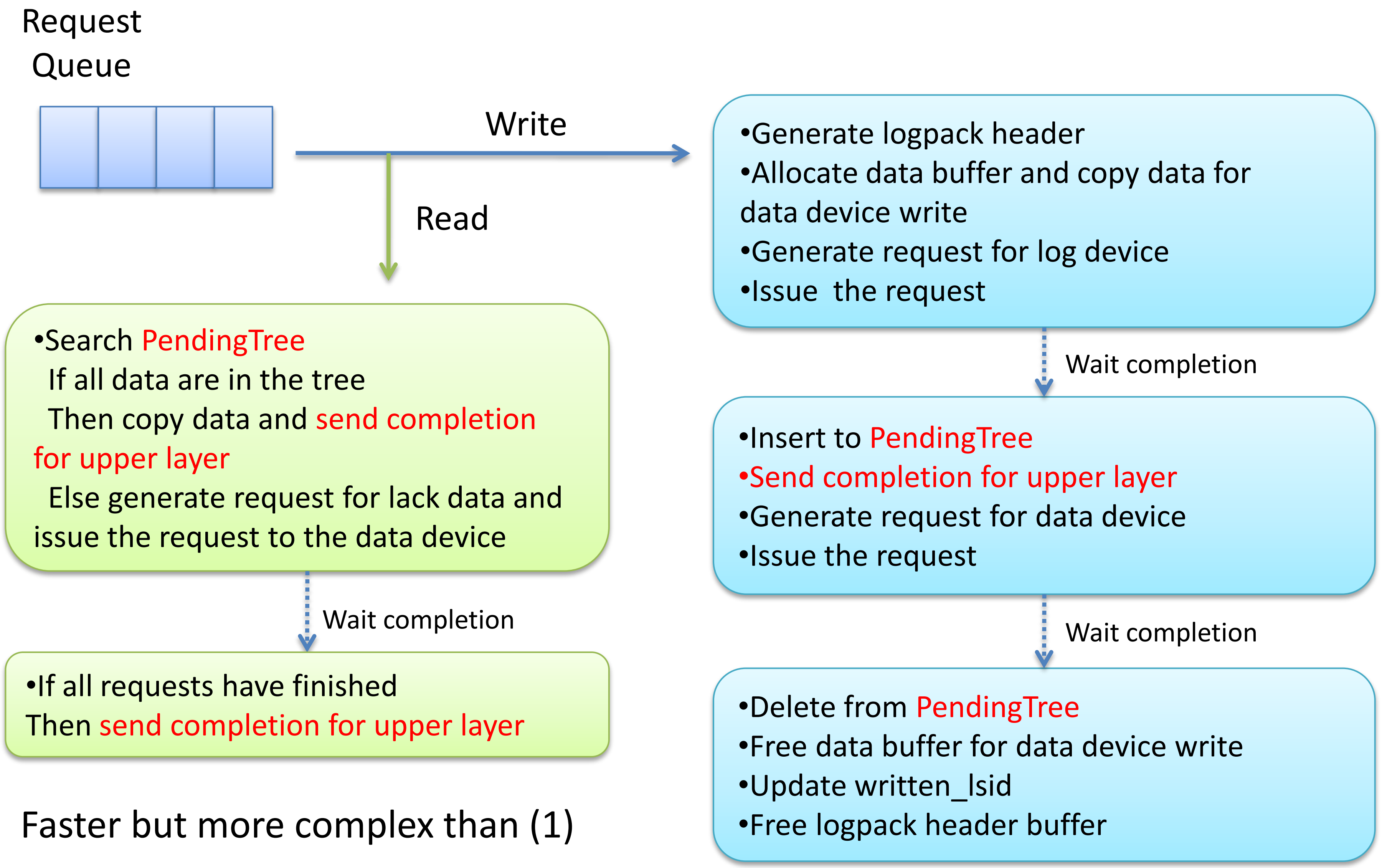
Read/Write Algorithm (1)



Request Queue
Write
Read
•Generate and issue to the data device
Wait completion
•Send completion for upper layer
•Generate logpack header
•Generate request for log device
•Issue the request
Wait completion
•Generate request for data device
•Issue the request
Wait completion
•Send completion for upper layer
•Update written_lsid
•Free logpack header buffer

Simple Algorithm

Read/Write Algorithm (2)



- Search PendingTree
  If all data are in the tree
  Then copy data and send completion for upper layer
  Else generate request for lack data and issue the request to the data device

- If all requests have finished
  Then send completion for upper layer

- Generate logpack header
- Allocate data buffer and copy data for data device write
- Generate request for log device
- Issue the request

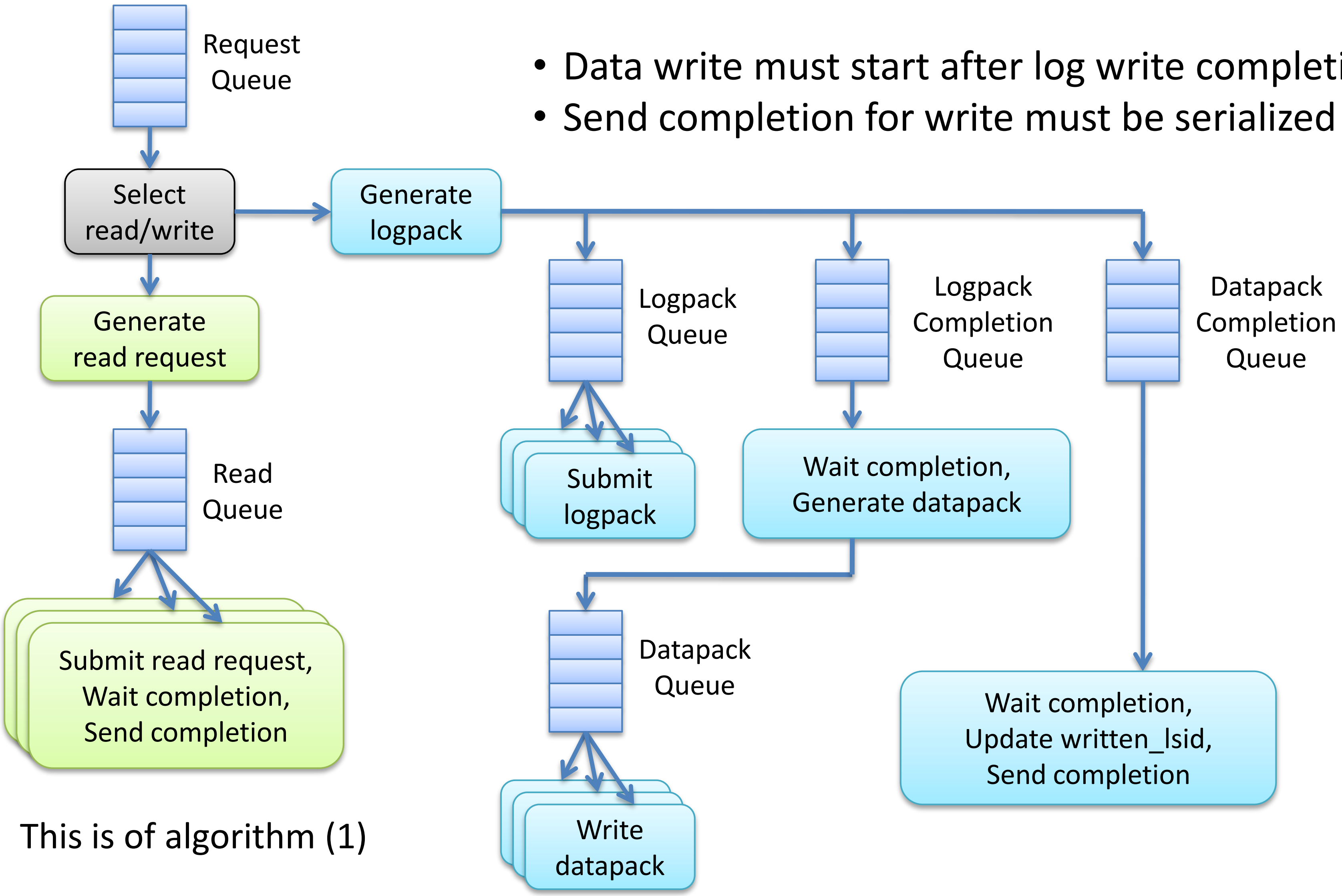
- Insert to PendingTree
- Send completion for upper layer
- Generate request for data device
- Issue the request

- Delete from PendingTree
- Free data buffer for data device write
- Update written_lsid
- Free logpack header buffer

Faster but more complex than (1)

Parallel Task Processing

- Data write must start after log write completion
- Send completion for write must be serialized



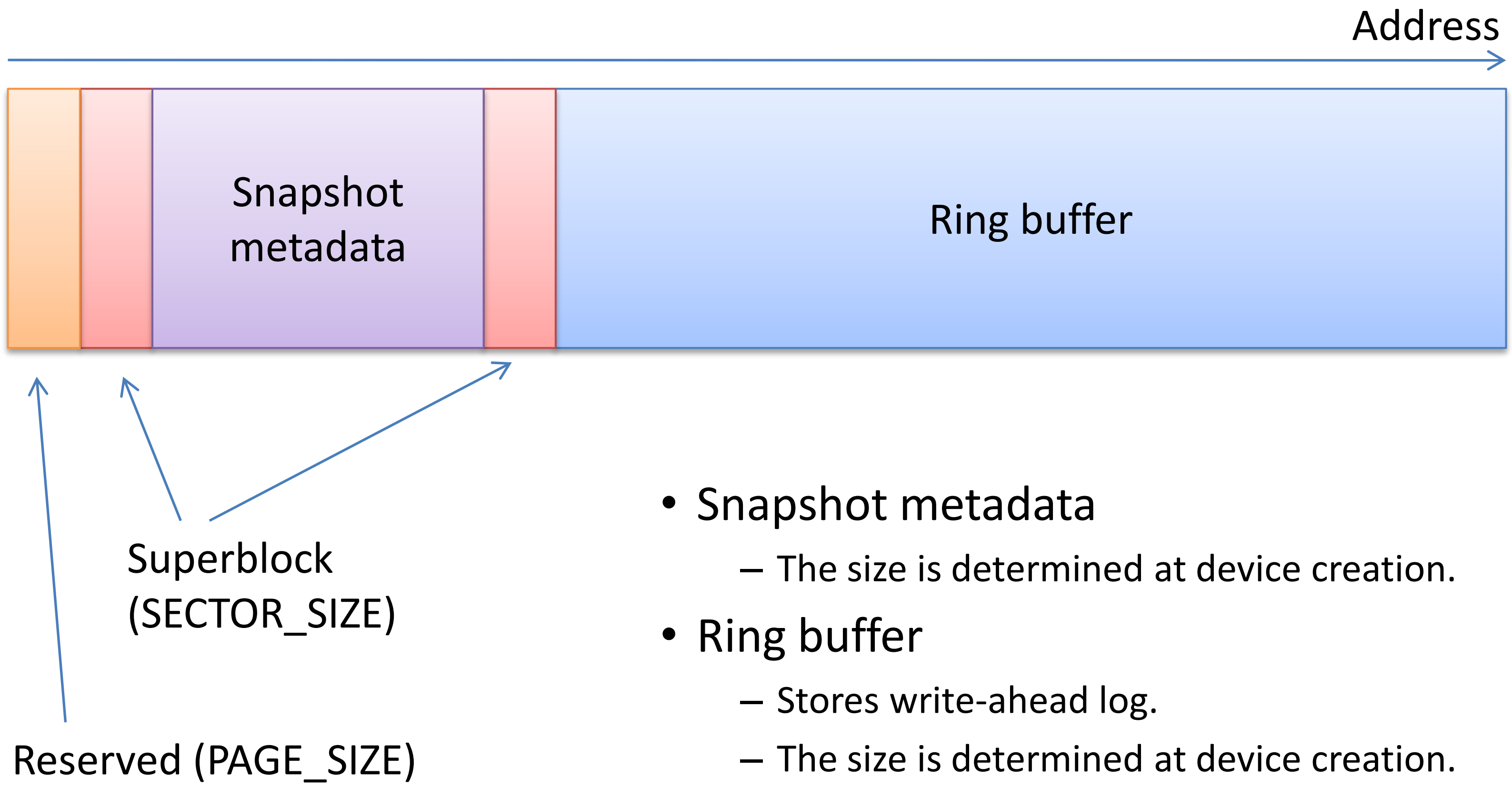
Request Queue
Select read/write
Generate logpack
Generate read request
Read Queue
Submit read request, Wait completion, Send completion
Logpack Queue
Submit logpack
Logpack Completion Queue
Wait completion, Generate datapack
Datapack Queue
Write datapack
Datapack Completion Queue
Wait completion, Update written_lsid, Send completion

This is of algorithm (1)

WalB Data Format

- Data device
  - The same image as wrapping block device
- Log device
  - Overview
  - Snapshot metadata
  - Ring buffer
  - Logpack

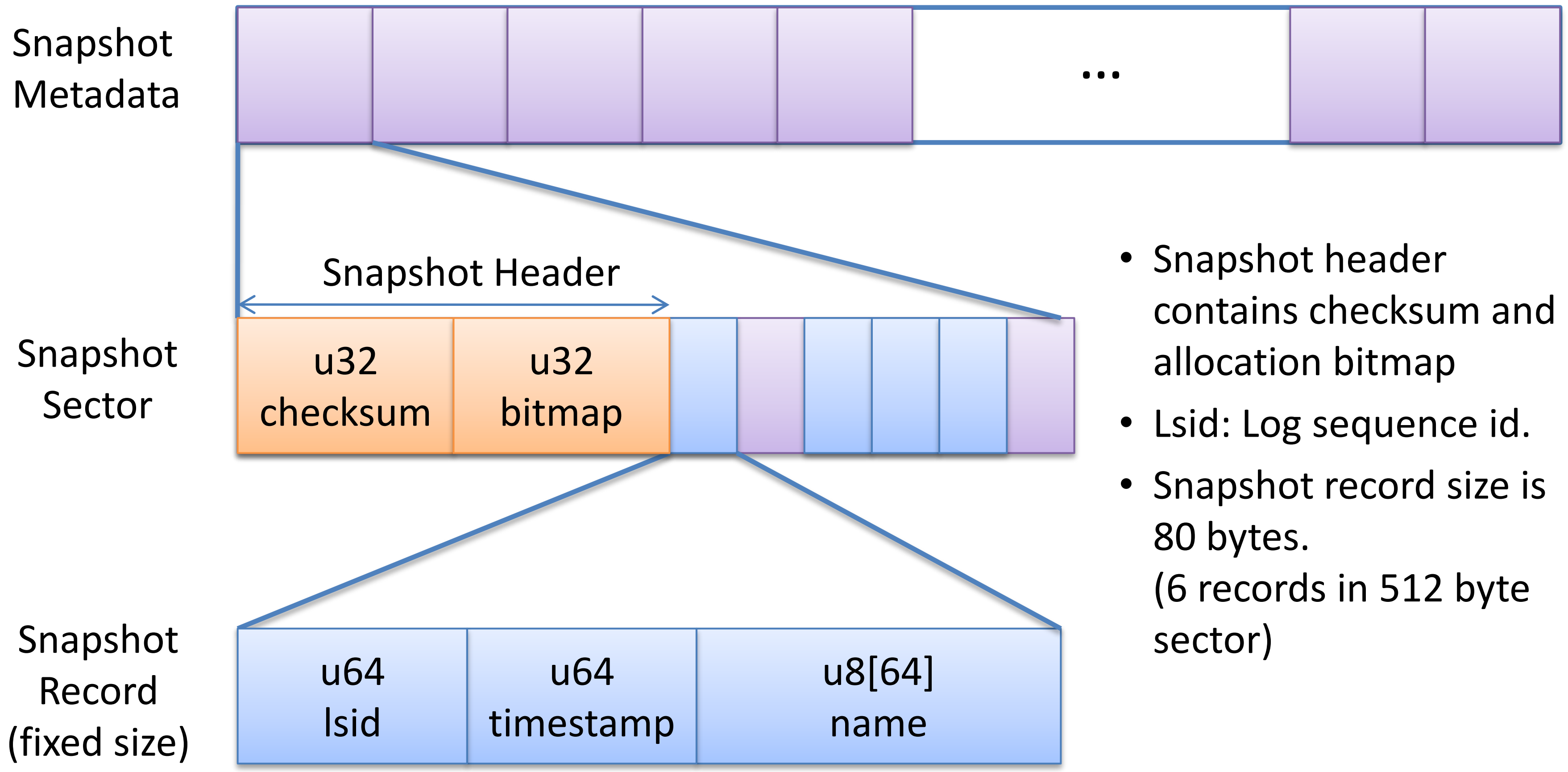
Log Device Format



PAGE_SIZE = 4096 bytes
SECTOR_SIZE = 512 or 4096 bytes

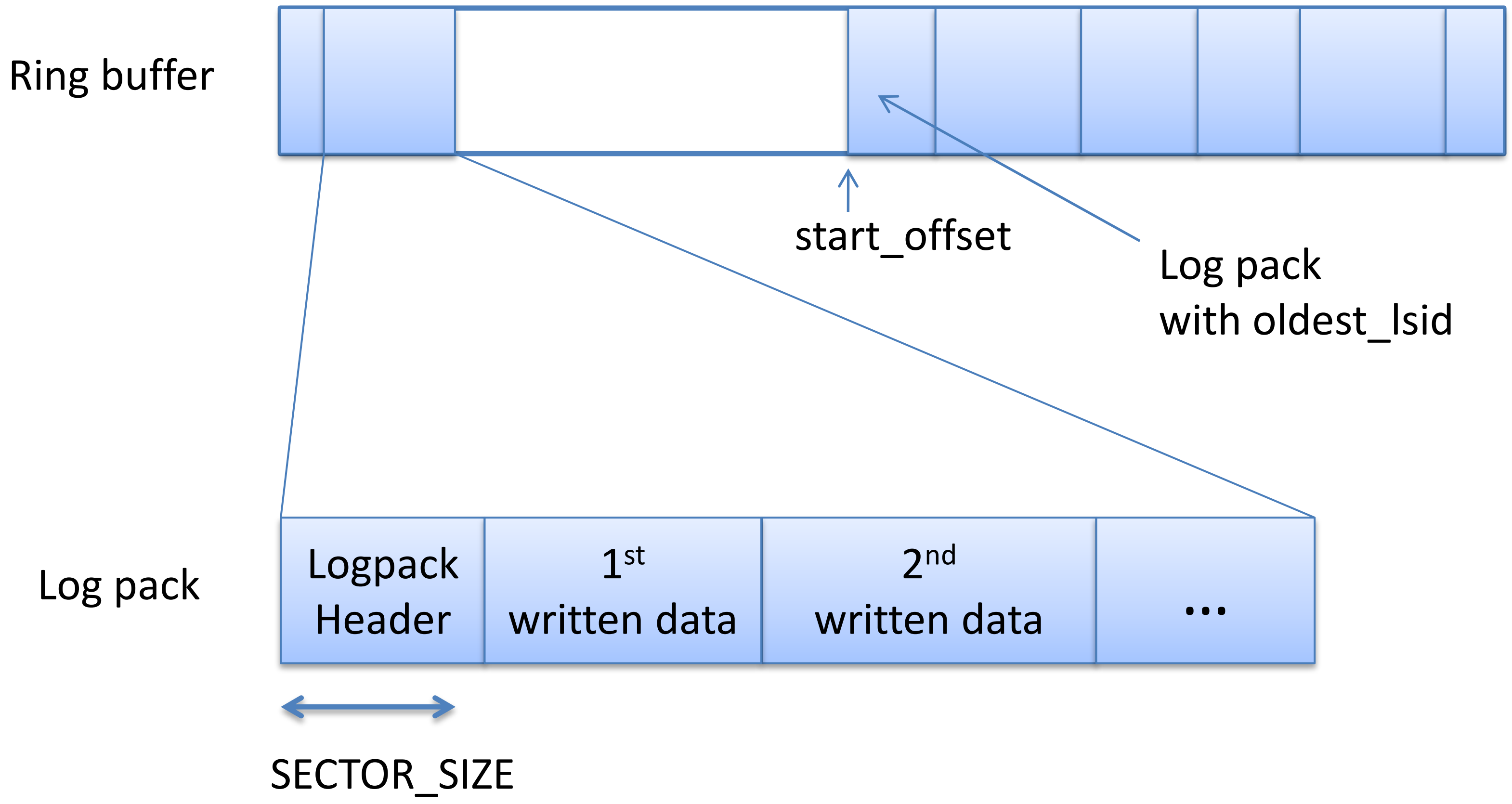
- Snapshot metadata
  - The size is determined at device creation.
- Ring buffer
  - Stores write-ahead log.
  - The size is determined at device creation.
  - The size can not be changed.

Snapshot Metadata

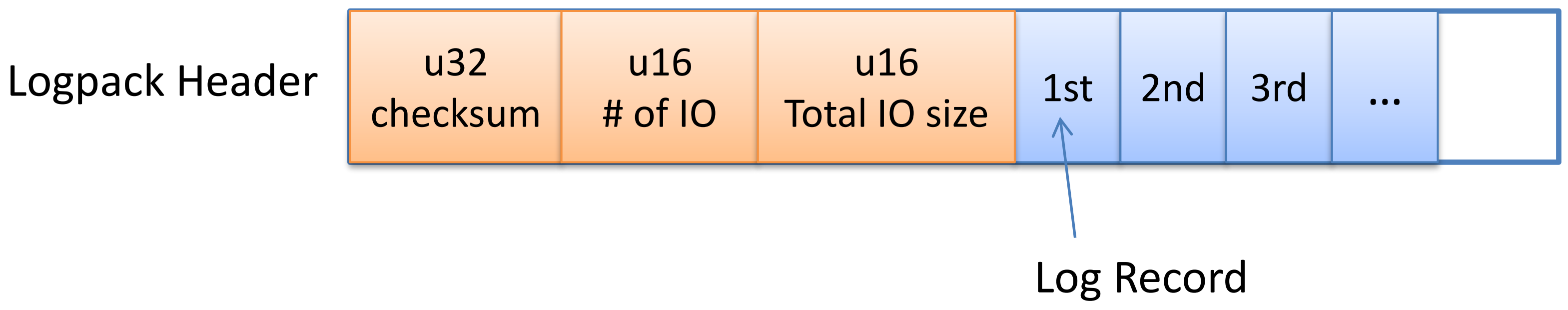


- Snapshot header contains checksum and allocation bitmap
- Lsid: Log sequence id.
- Snapshot record size is 80 bytes.
  (6 records in 512 byte sector)

Ring Buffer


Ring buffer
start_offset
Log pack
with oldest_lsid
Log pack
Logpack
Header
1st
written data
2nd
written data
…
SECTOR_SIZE

Logpack Header



Log Record

```
u64 lsid;          /* Log sequence id */
u64 offset;        /* IO offset by the sector. */
u16 lsid_local;    /* local sequence id
                      as the data offset in the log record. */
u16 size;          /* IO size by the sector. */
u16 is_exist;      /* 0 if this record does not exist. */
u16 reserved1;
```

Pros and Cons

	WalB (redo log)	Snapshot with redo log	Snapshot with undo log
Read	No overhead	Index search	Bitmap search
Write	Write twice (1)	Index modification	Bitmap search/modification and old data copy
	No overhead (2)		
Typical Soft.	---	ZFS, BtrFS	LVM

WalB + Index ~= Block device with snapshot management

Current Progress

- Survey and study linux kernel programming
- Design of rough architecture
- Prototype implementation
- Basic evaluation and redesign if required
- Implementation of full functionalities and test
- Operation inside Cybozu
- Publication as GPLv2

- Merging to device-mapper if required
- Merging to main repository (hopefully)

Summary

- Motivaion
  - No good backup solution
    covering various applications
- WalB
  - Is a block device driver with WAL
  - Provides efficient incremental backup