The NILFS2 Filesystem: Review and Challenges

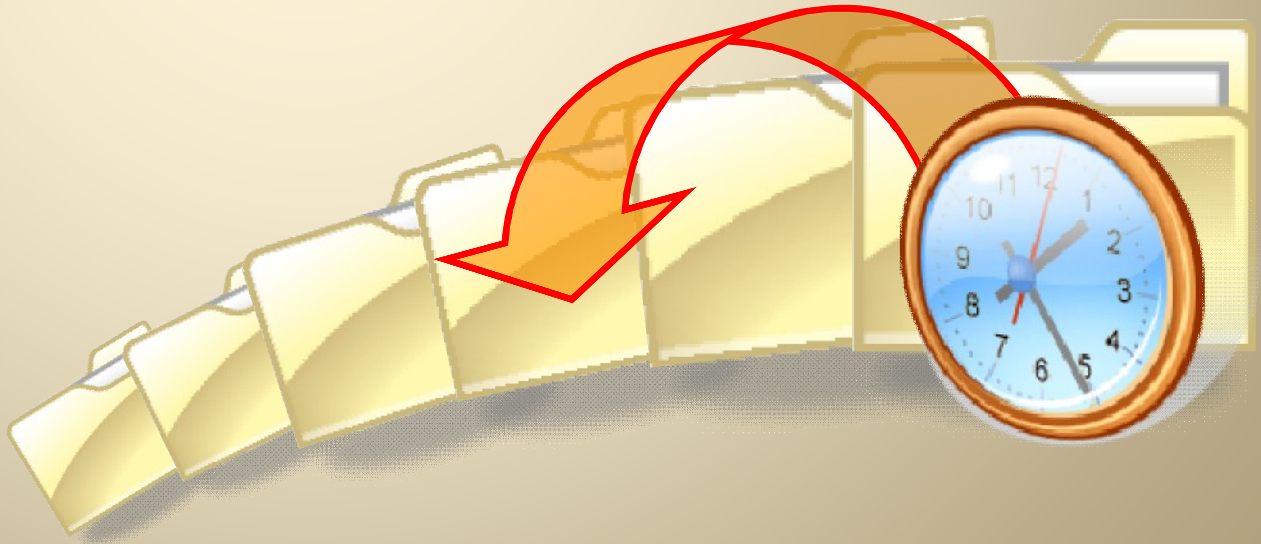
Ryusuke KONISHI
NTT Cyberspace Laboratories
NTT Corporation

Agenda

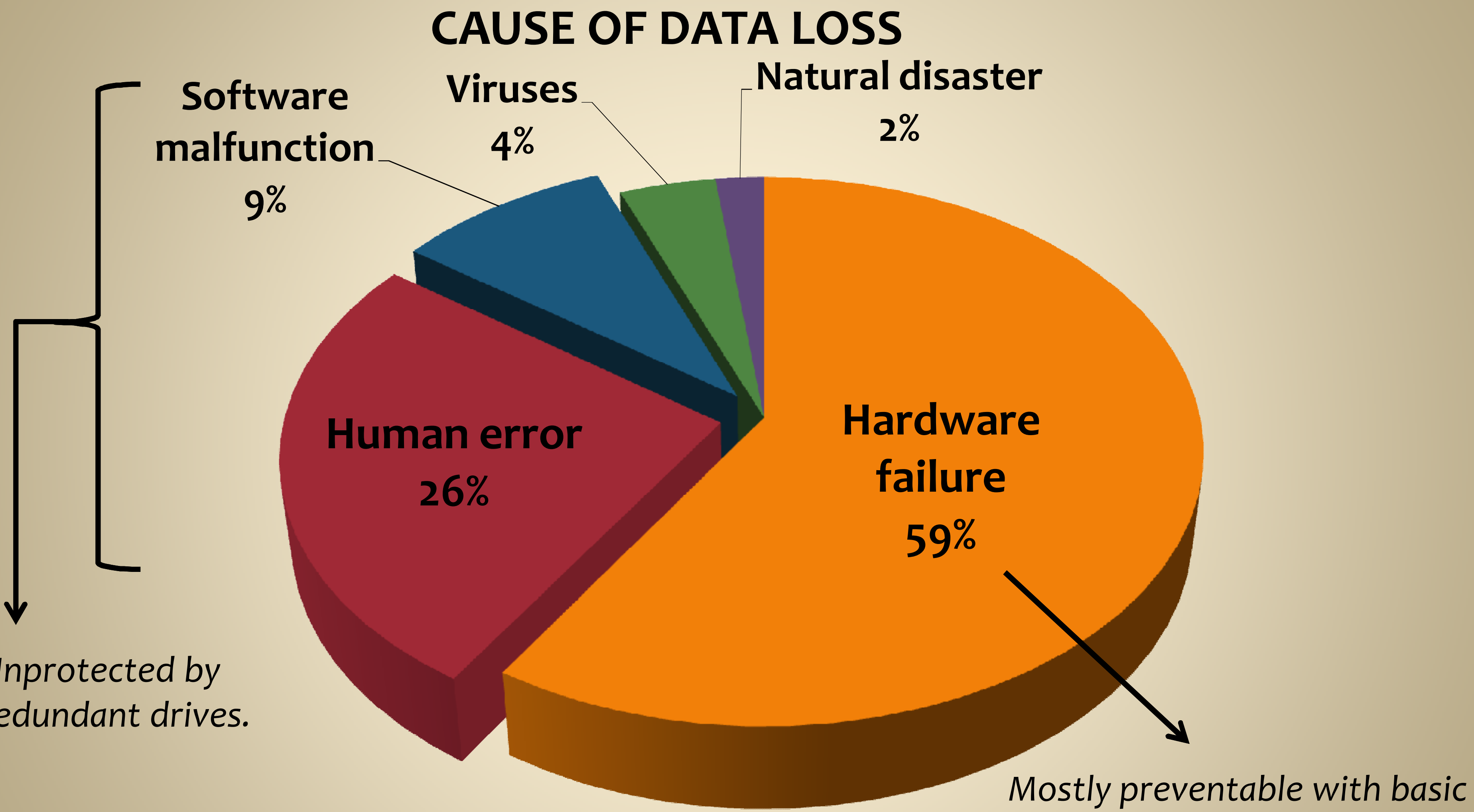
- NILFS Introduction
- File System Design
- Development Status
- Wished features & Challenges

What's NILFS

- **NILFS** is the Linux file system supporting "continuous snapshotting"
  - ✓Provides versioning capability of entire file system
  - ✓Can retrieve previous states before operation mistake
    - even restores files mistakenly overwritten or destroyed just a few seconds ago.
  - ✓Merged into the mainline kernel 2.6.30



Why NILFS ? (1)



Unprotected by redundant drives.

Mostly preventable with basic high-integrity system (Redundant configuration)

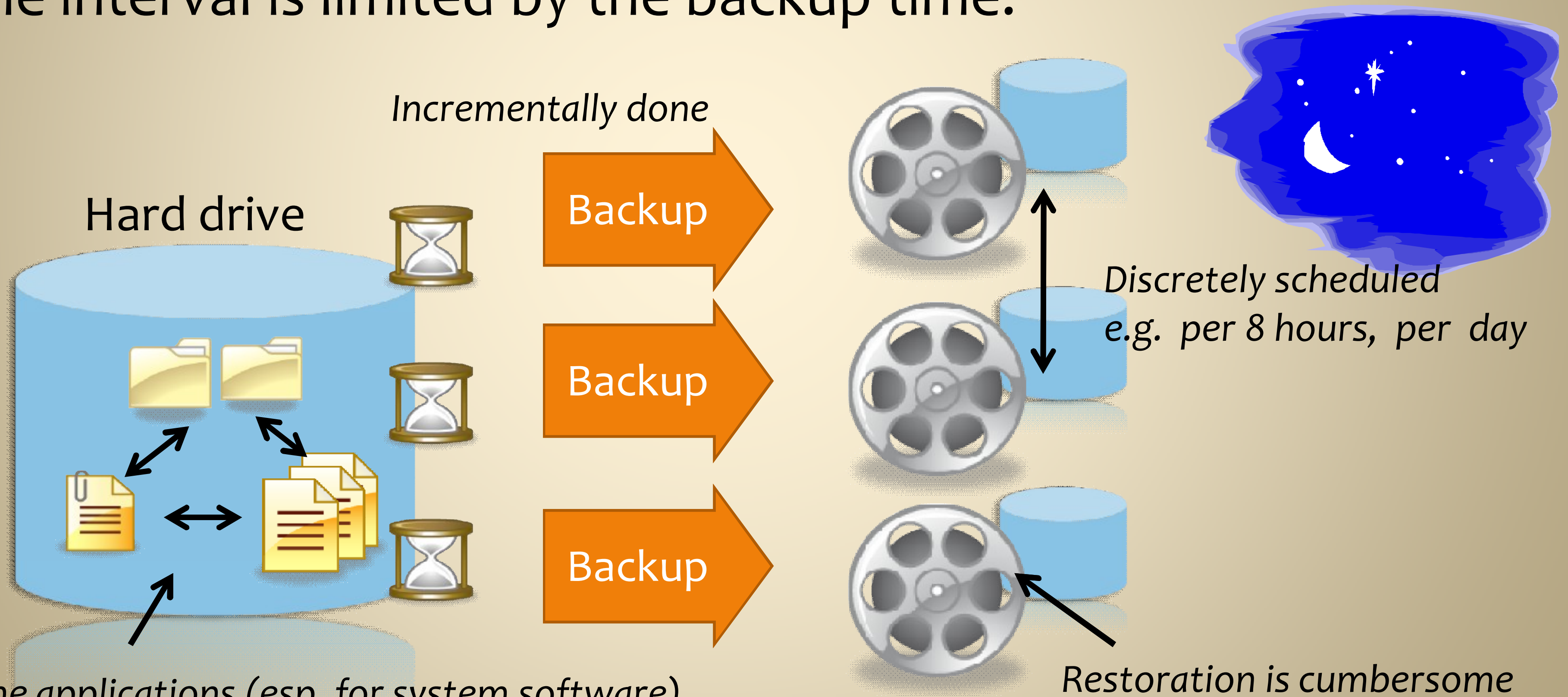
Source: Ontrack Data Recovery, Inc.

Including office PC. The data is based on actual data recoveries performed by Ontrack.

Why NILFS ? (2)

ISSUES IN BACKUP

- Changes after the last backup are not safe
- But, frequent backups place burden on the system as well as the interval is limited by the backup time.



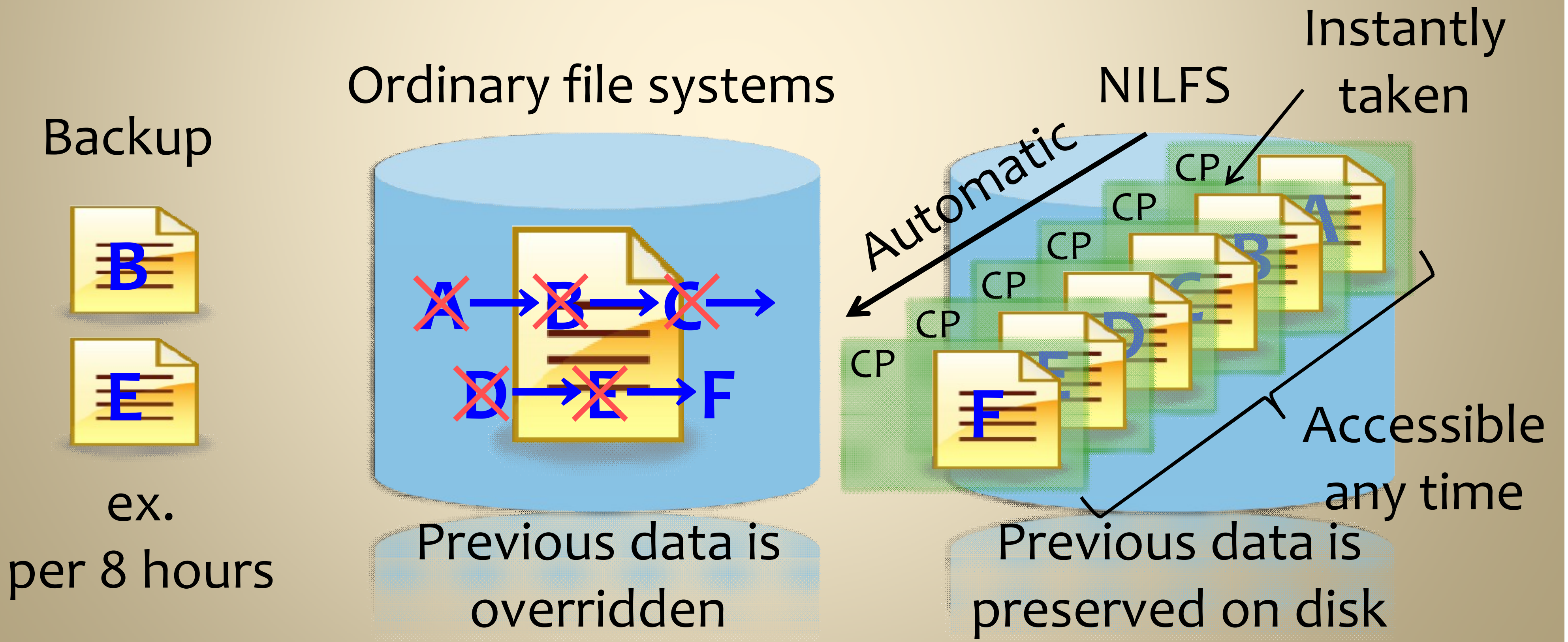
Discretely scheduled e.g. per 8 hours, per day

Some applications (esp. for system software) are sensitive to coherency among files

Restoration is cumbersome (and often fails or unhelpful)

NILFS Solution

- Adopt Log-structured File System approach to continually save data on disk.
- Checkpoints are created every time user makes a change, and each checkpoint is mutable to snapshots later on.



Comparison of Snapshots

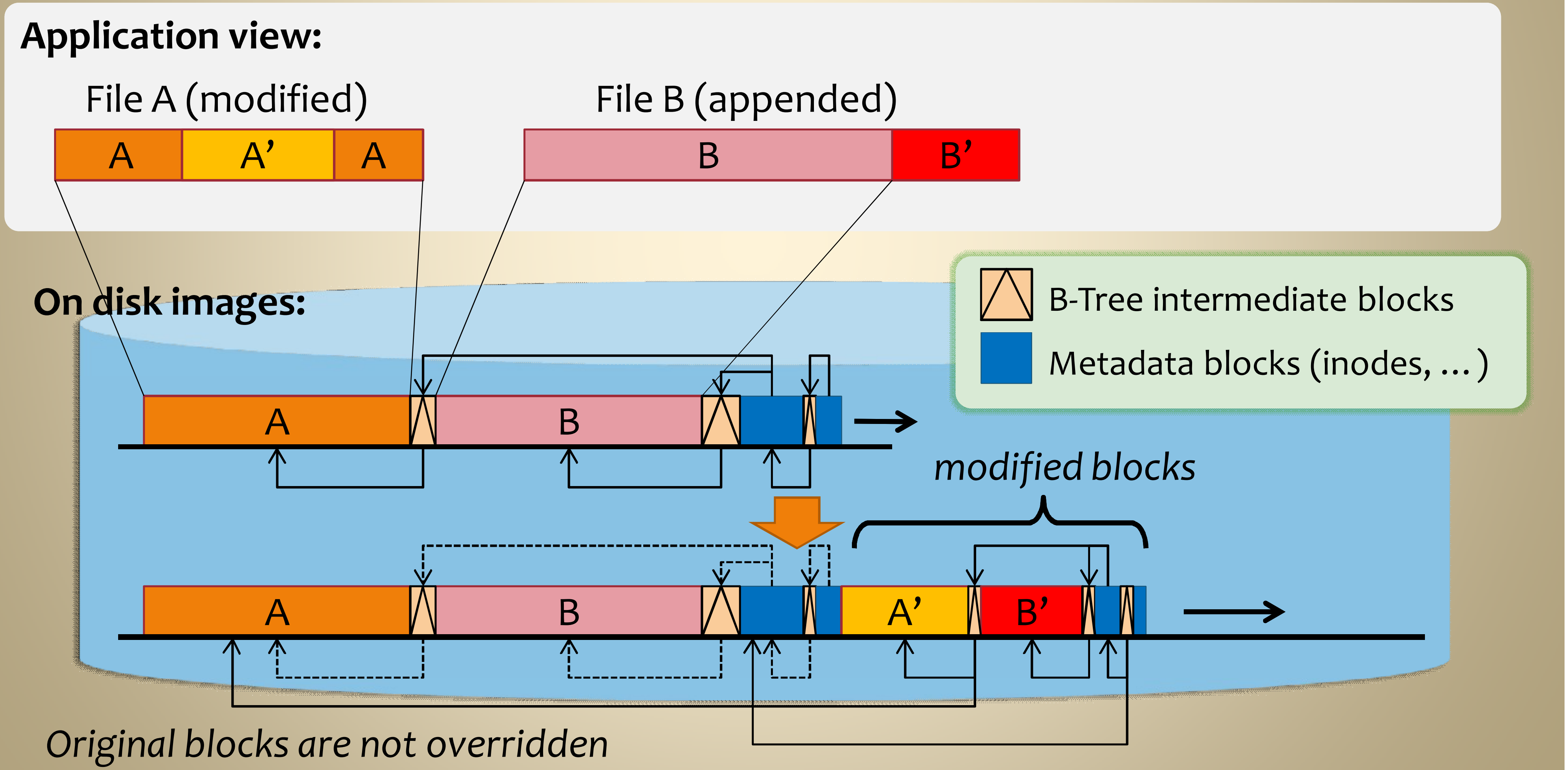
File System (solution)	Maximum Number of Snapshots	Instant Snapshotting	Writable Snapshots	Retroactive Snapshots	Incremental Backup
NTFS (Volume Shadow Copy)	64				Optional (Third party product)
ZFS	Unlimited*1	√	√		√
Btrfs	Unlimited*1	√	√		Planned
NILFS2	Unlimited*1	√		√	Requested
(Apple Time Machine)	Thinned out automatically			—	√
(CDP)	Unlimited*1	—	—	—	√

Backup Solutions

*1: No practical limits (bounded by disk capacity)

NILFS Disk Write

- Only modified blocks are incrementally written to disk
  - ✓ This write scheme is applied even to metadata and intermediate blocks



Application view:

File A (modified)

File B (appended)

On disk images:

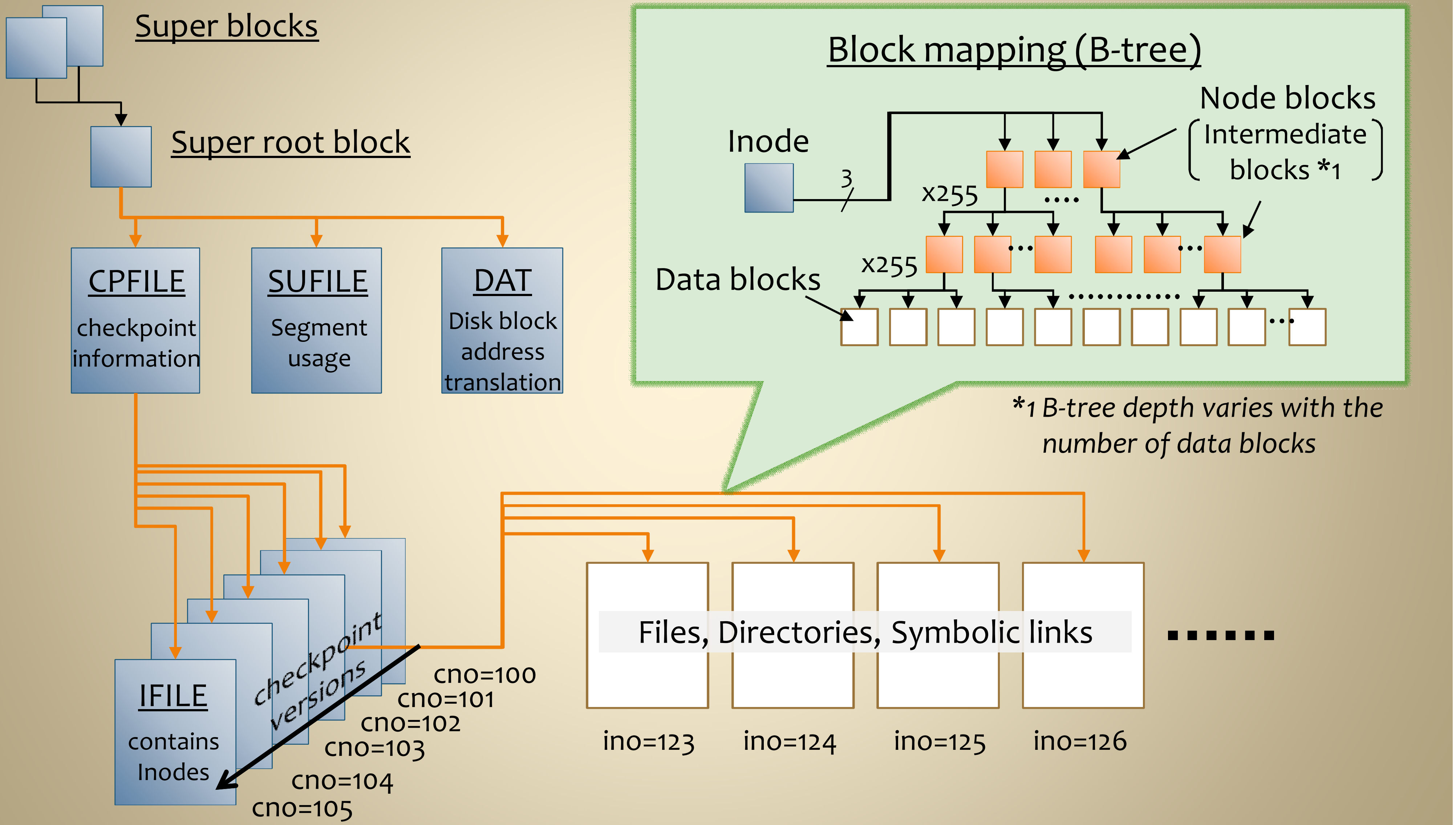
B-Tree intermediate blocks

Metadata blocks (inodes, …)

modified blocks

Original blocks are not overridden

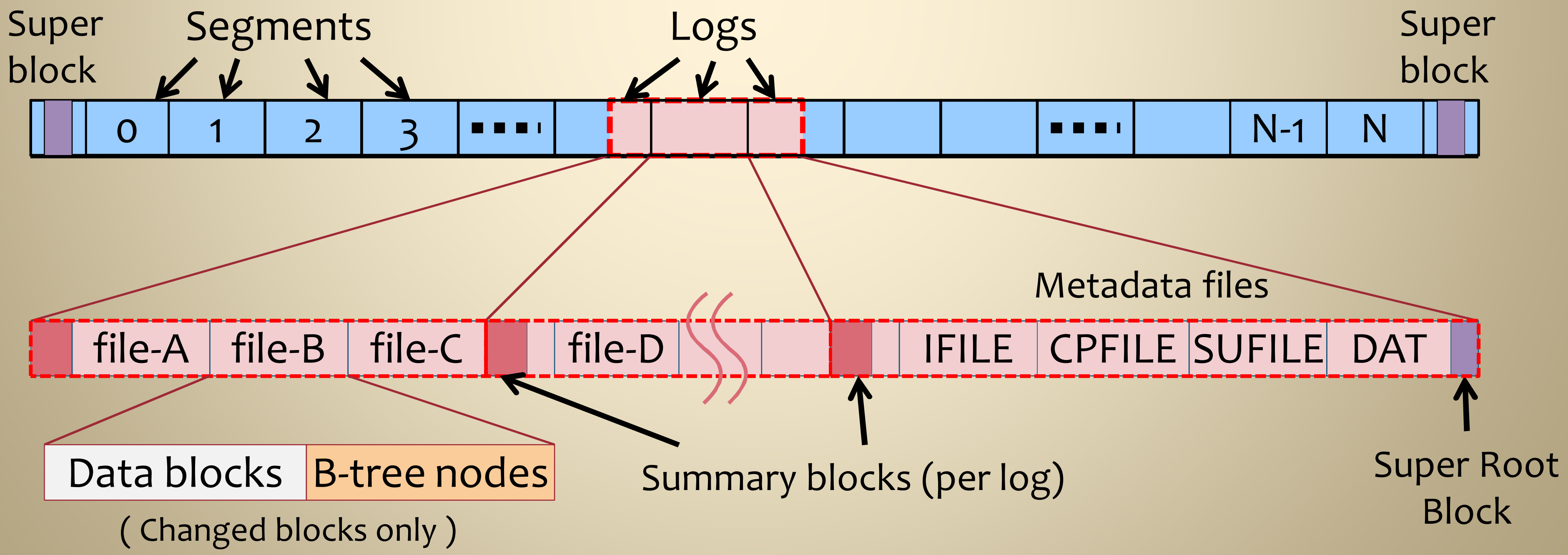
NILFS Metadata Hierarchy



*1 B-tree depth varies with the number of data blocks

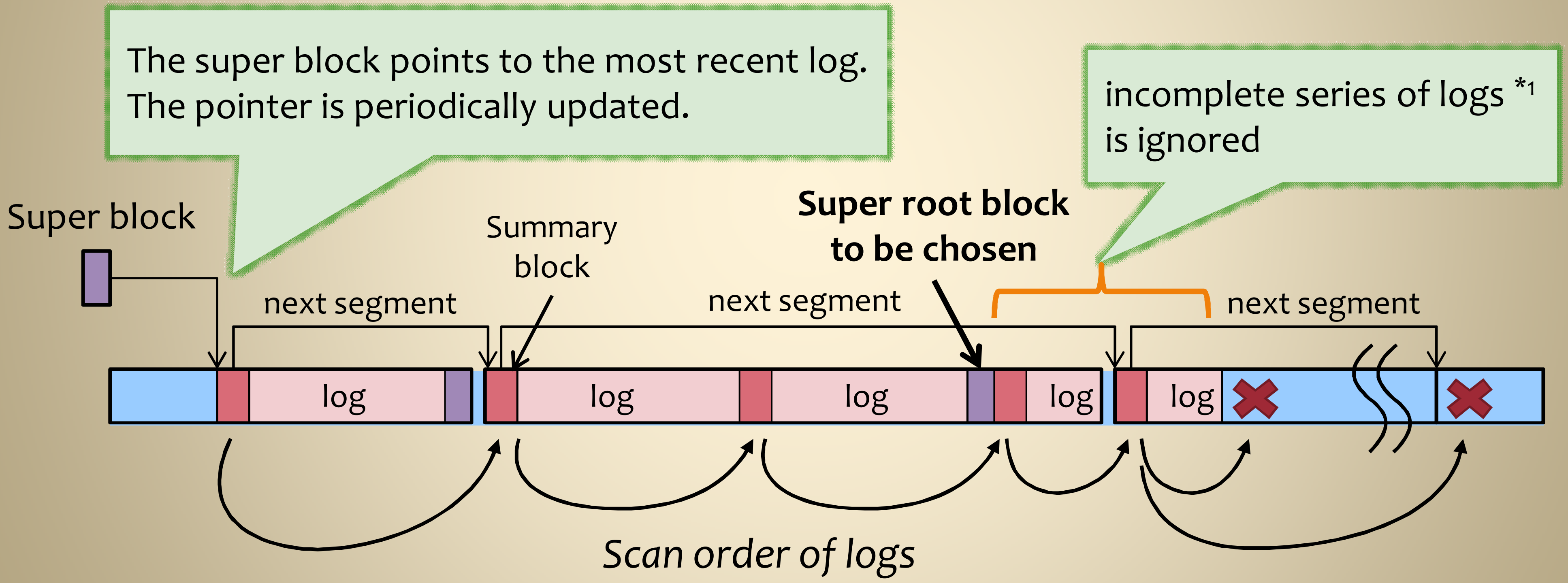
Disk Layout Summary

- Segments
  - ✓ Disk space is allocated or freed per segment
  - ✓ Each segment is filled with logs
- Logs
  - ✓ Organize delta of data and metadata per file
  - ✓ Compose a new version of metadata hierarchy every checkpoint



Mount-time Recovery

- How does NILFS recover from unclean status?
  - ✓ Finds the last log which has a super root block, and done!
  - ✓ Each log is validated with checksums

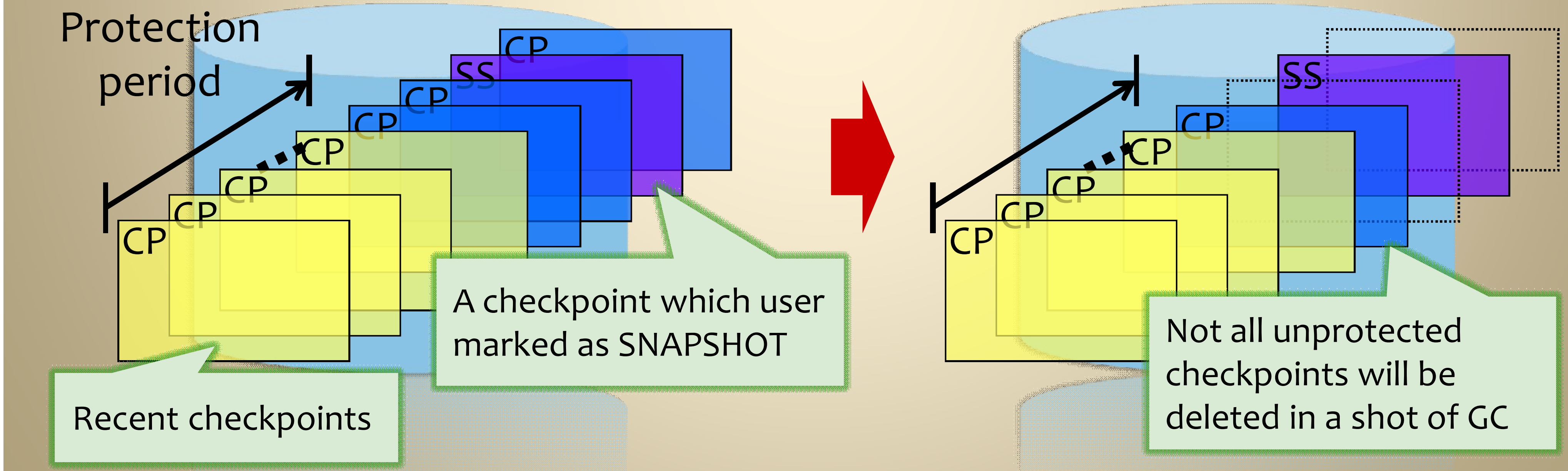


*1 Series of logs may not have the super root block. This type of variant is allowed for optimizations to make synchronous write operation faster.

Garbage Collection (1)

- Creates new disk space to continue writing logs
  - ✓ Essential function of Log structured File Systems
- A disk block is in-use if it belongs to a snapshot or recent checkpoints; unused blocks are freed with their checkpoints

Preserving checkpoints as snapshots



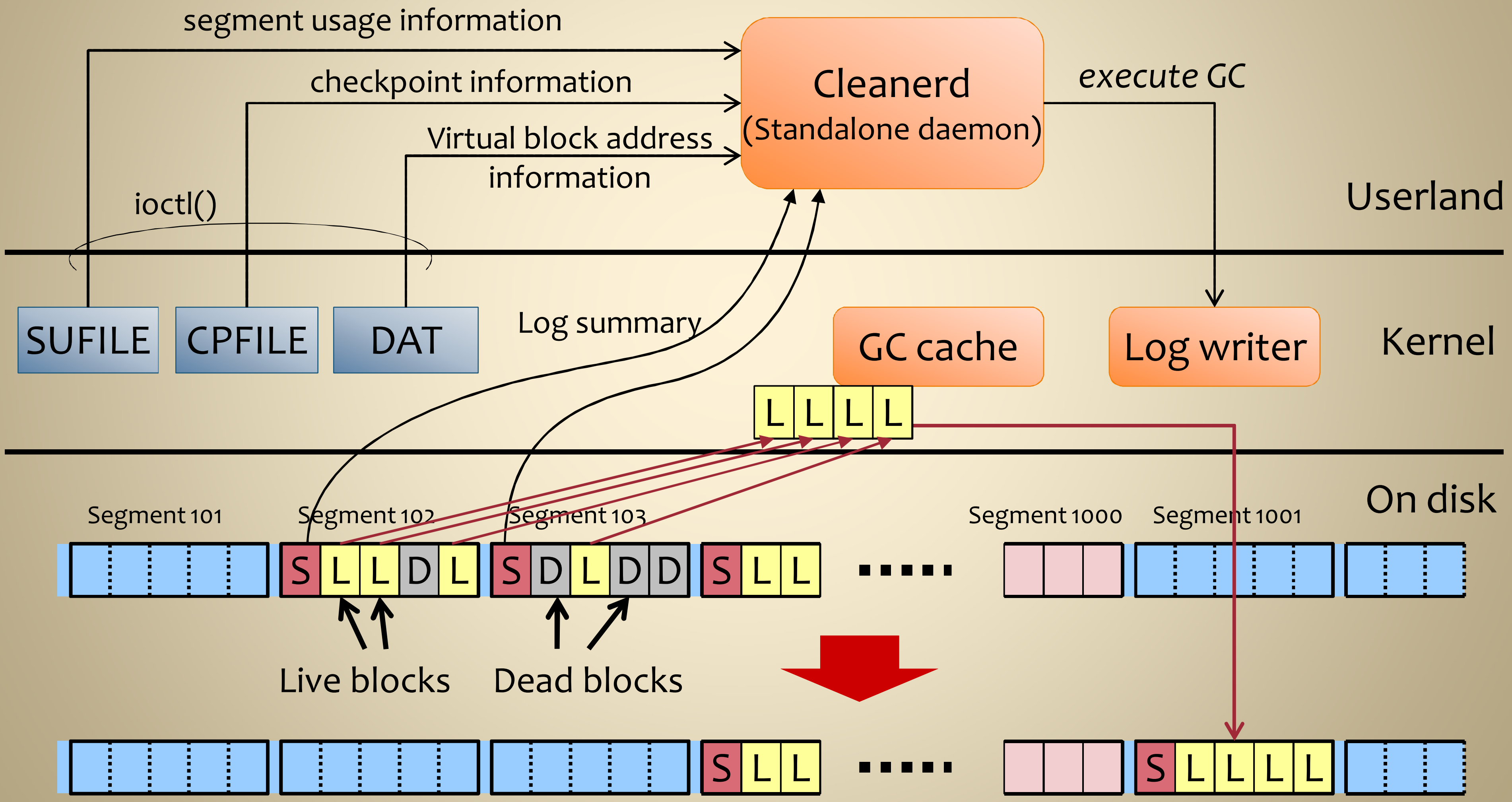
A checkpoint which user marked as SNAPSHOT

Recent checkpoints

Not all unprotected checkpoints will be deleted in a shot of GC

Garbage Collection (2)

Overall view

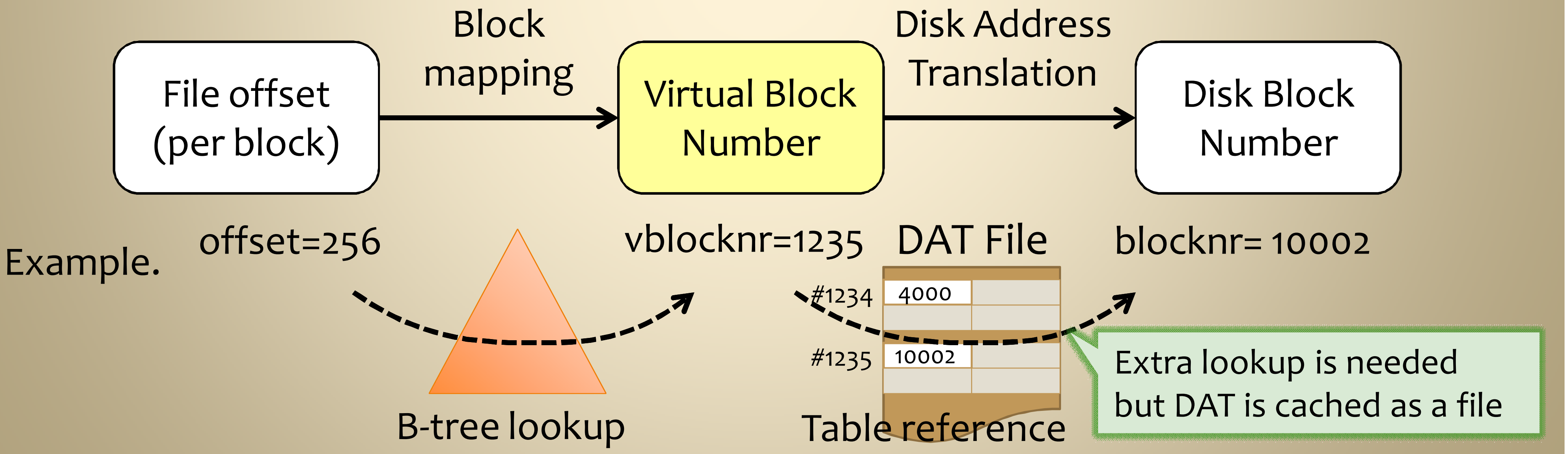


Copyright (C) 2009 NTT Corporation

Garbage Collection (3)

block addressing

- Issue for moving disk blocks
  - ✓ Must rewrite b-tree node blocks and inodes having a pointer to moved blocks
  - ✓ Disk blocks are pointed from many parent blocks because NILFS makes numerous versions
- Solution
  - ✓ Use virtual (i.e. indirect) block numbers instead of real disk block numbers

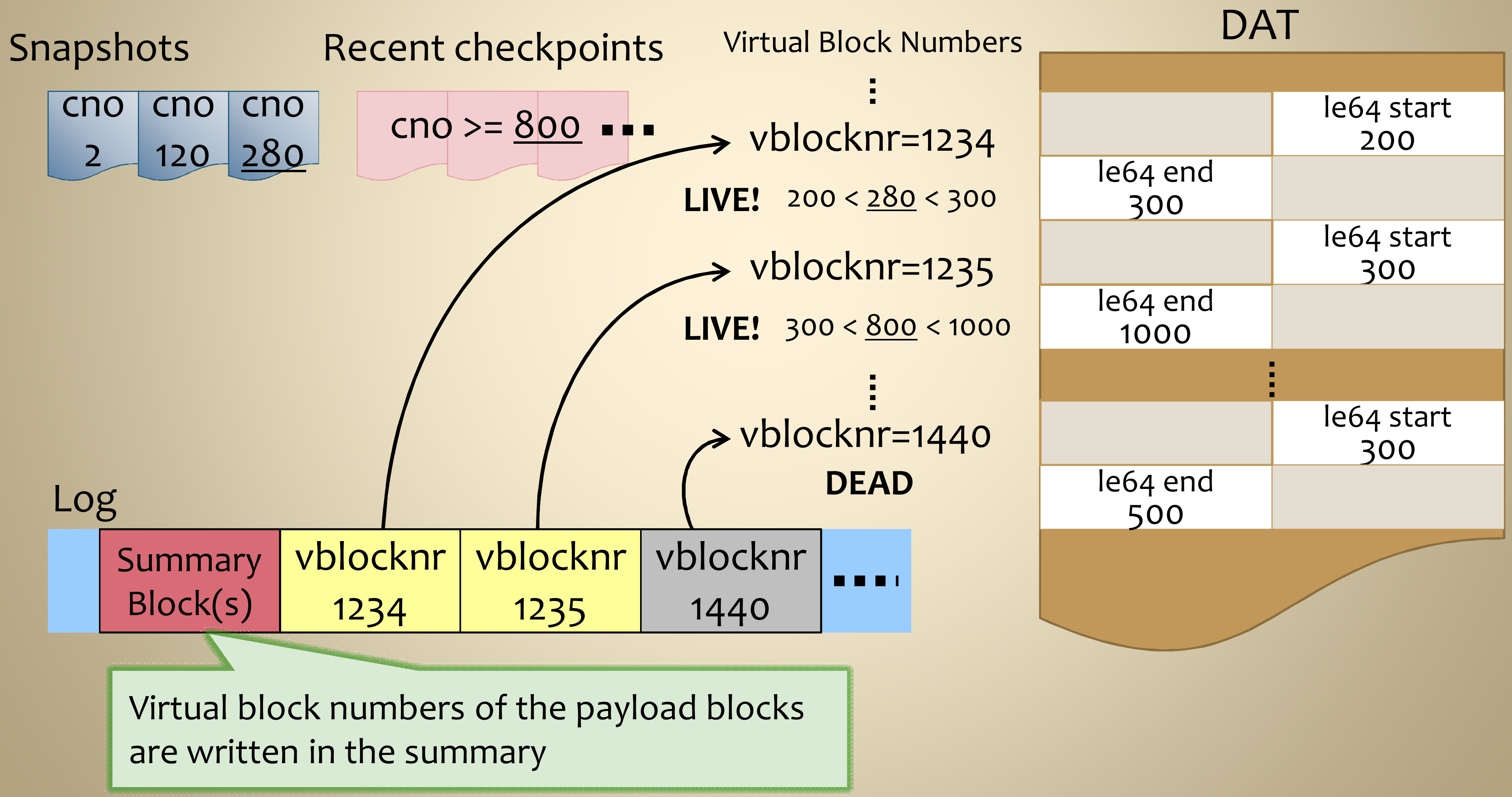


Extra lookup is needed but DAT is cached as a file

Garbage Collection (4)

Live or dead determination

- Cleanerd determines if each disk block is **LIVE** or **DEAD** from DAT



Virtual block numbers of the payload blocks are written in the summary

Current Development Status (1)

- Achievements
  - ✓Snapshots
    - Automatically and continuously taken
    - Mountable as read-only file systems
    - Mountable concurrently with the writable mount (convenient for online backup)
    - Quick listing
    - Easy administration
  - ✓Online disk space reclamation
    - Can maintain multiple snapshots

Current Development Status (2)

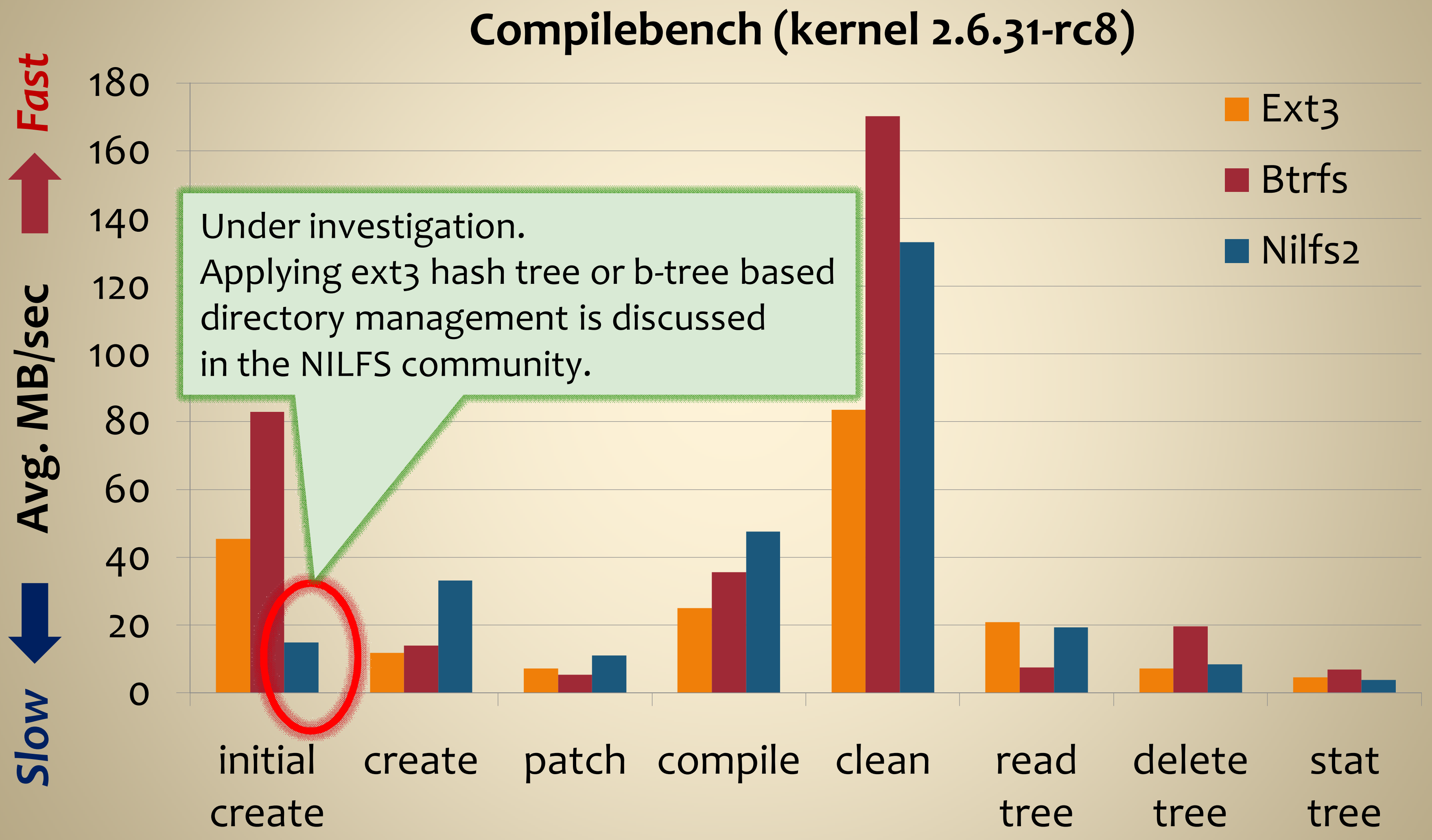
- Achievements
  - ✓Other Features
    - Quick recovery on-mount after system crash
    - B-tree based file and meta data management
    - 64-bit data structures; support many files, large files and disks
    - Block sizes smaller than page size (e.g. 1KB or 2KB)
    - Redundant super blocks (automatic switch)
    - 64-bit on-disk timestamps which are free of the year 2038 problem
    - Nano second timestamps

Current Development Status (3)

- Todo
  - ✓ On-disk atime
  - ✓ Extended attributes (work in progress)
  - ✓ POSIX ACLs
  - ✓ O_DIRECT write
    - Currently fallback to buffered write
  - ✓ Fsck
  - ✓ Resize
  - ✓ Quotas

- Performance issues
  - ✓ Directory operations
  - ✓ Write performance
  - ✓ Optimization for silicon disks (esp. for SSD)

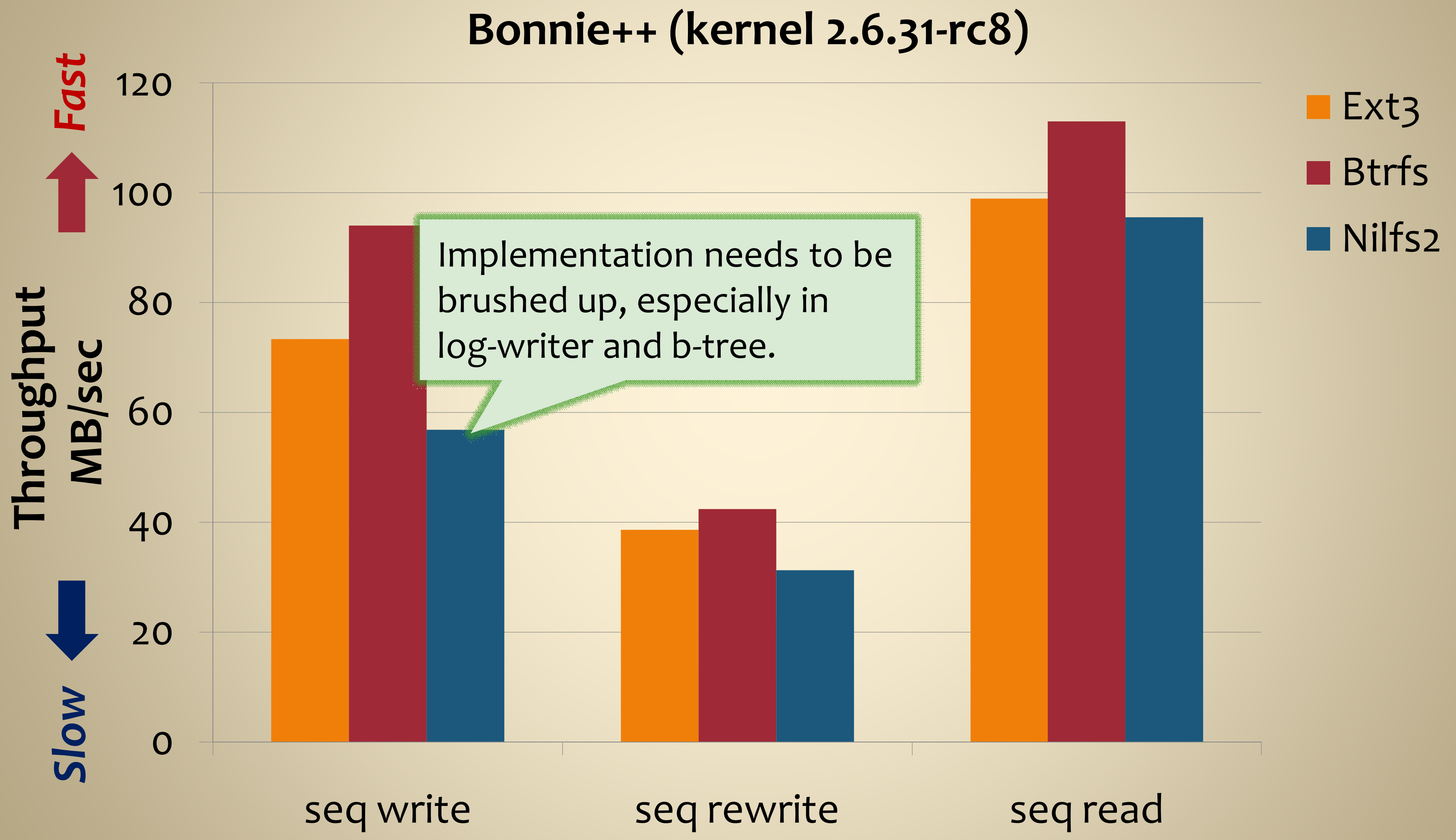
Performance issues (1)



Under investigation.
Applying ext3 hash tree or b-tree based directory management is discussed in the NILFS community.

Hardware specs:
Processor: Pentium Dual-Core CPU E5200 @ 2.49GHz, Chipset: Intel 4 Series Chipset + ICH10R, Memory: 2989MB, Disk: ST3500620AS

Performance issues (2)

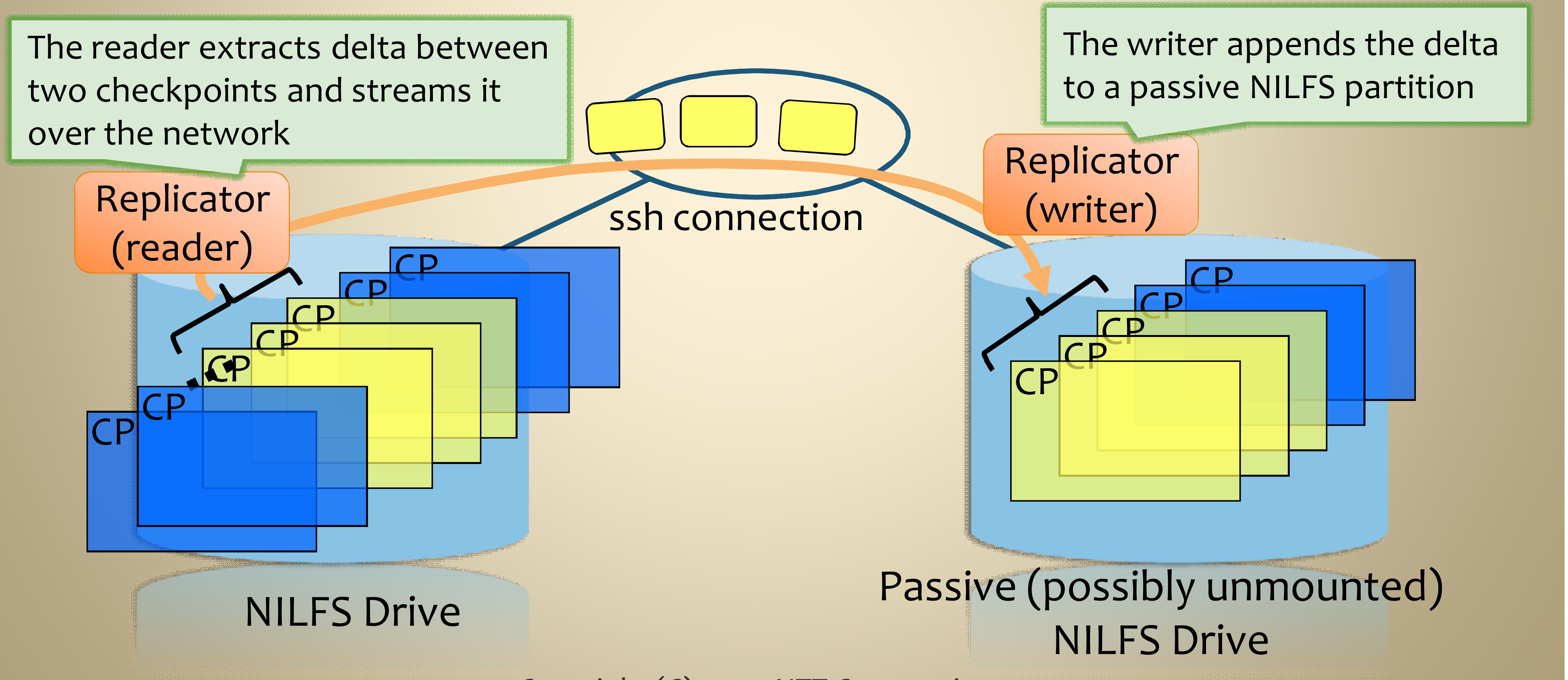


Implementation needs to be brushed up, especially in log-writer and b-tree.

Hardware specs:
Processor: Pentium Dual-Core CPU E5200 @ 2.49GHz, Chipset: Intel 4 Series Chipset + ICH10R, Memory: 2989MB, Disk: ST3500620AS

Checkpoint-based Replication (1)

- Faster and robust online backup like ZFS
  - ✓ Back up checkpoints instead of usual files
  - ✓ Similar features are planned for btrfs, TUX3, and the Device Mapper (dm replication)



The reader extracts delta between two checkpoints and streams it over the network

The writer appends the delta to a passive NILFS partition

Checkpoint-based Replication (2)

KEY CHALLENGES DISCUSSED IN THE NILFS COMMUNITY

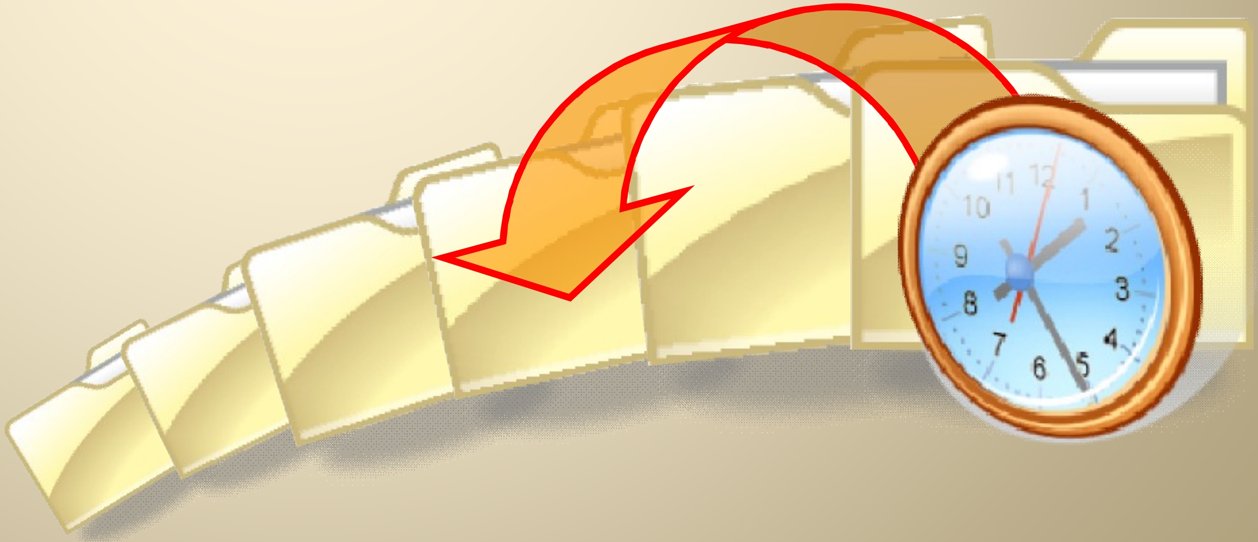
- How to extract delta between two checkpoints ?
  - ✓Two approaches
    - Scan logs in creation order (just gets delta from logs)
    - Scan DAT to gather blocks changed during given period
  - ✓Have pros and cons
    - The former seems to be efficient, but has a limit due to GC.
    - Replicator may use either or both of these methods
- Rollback on the destination file system
  - ✓Needed before starting replication especially to thin out the backups with GC

Refining GC

- Better garbage collector is much needed
  - ✓ Better data retention policy to prevent disk full
  - ✓ Self-regulating speed
  - ✓ Smarter selection algorithm of target segments to reduce I/O
- Further chance of optimization and enhancement
  - ✓ Background data verification
  - ✓ Defragmentation
  - ✓ De-duplication
  - ✓ Background disk format upgrade

Conclusion

- NILFS is in the mainline kernel
  - ✓You can go back in time just before you scream “**Ohhh Nooo…!!**”
  - ✓Instant failure recovery. Simple administration.
  - ✓Potential for innovative application
  - ✓… and most importantly, WORKING STABLY :)
- Contribution is welcome
  - ✓Various topics in GC, snapshot tools, and time-oriented tools.
  - ✓Let’s drop the (EXPERIMENTAL) flag!



Questions?

- Project page
  - ✓http://www.nilfs.org/
- Mailing-list
  - ✓users (at) nilfs.org
  - ✓users-ja (at) nilfs.org
- Contact Information
  - ✓Ryusuke KONISHI <ryusuke (at) osrg.net>

Thank you for listening !